4th Workshop on
Graphs, Algorithms and Machine Learning
Cala Millor, Mallorca
March 24, 2026

Polynomial Local Search: The complexity of finding a local optimum

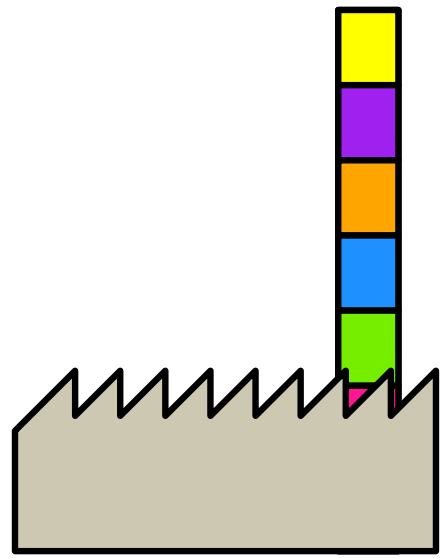
Dominik Scheder, TU Chemnitz

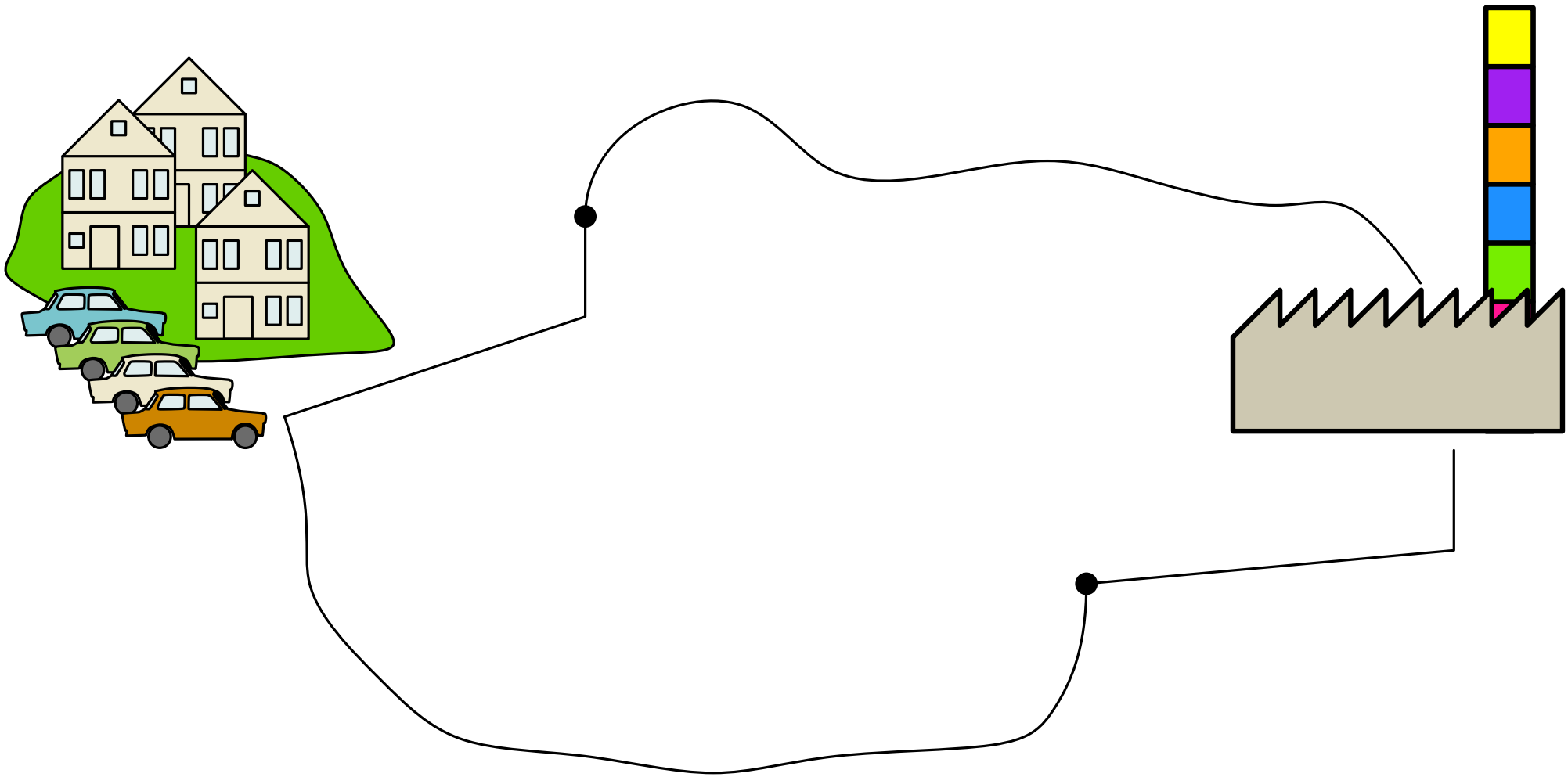
4th Workshop on
Graphs, Algorithms and Machine Learning
Cala Millor, Mallorca
March 24, 2026

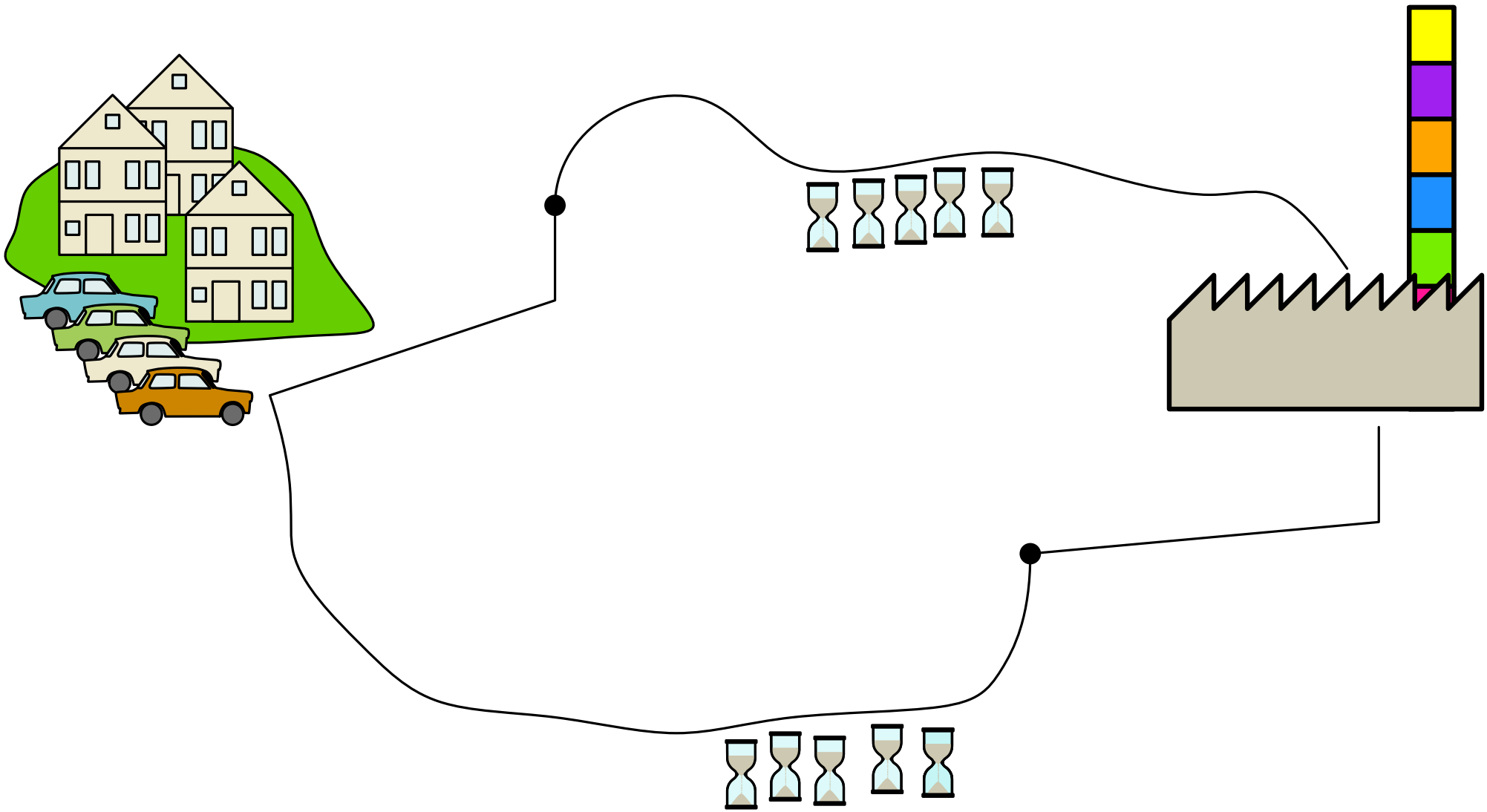
Polynomial Local Search: The complexity of finding a local optimum

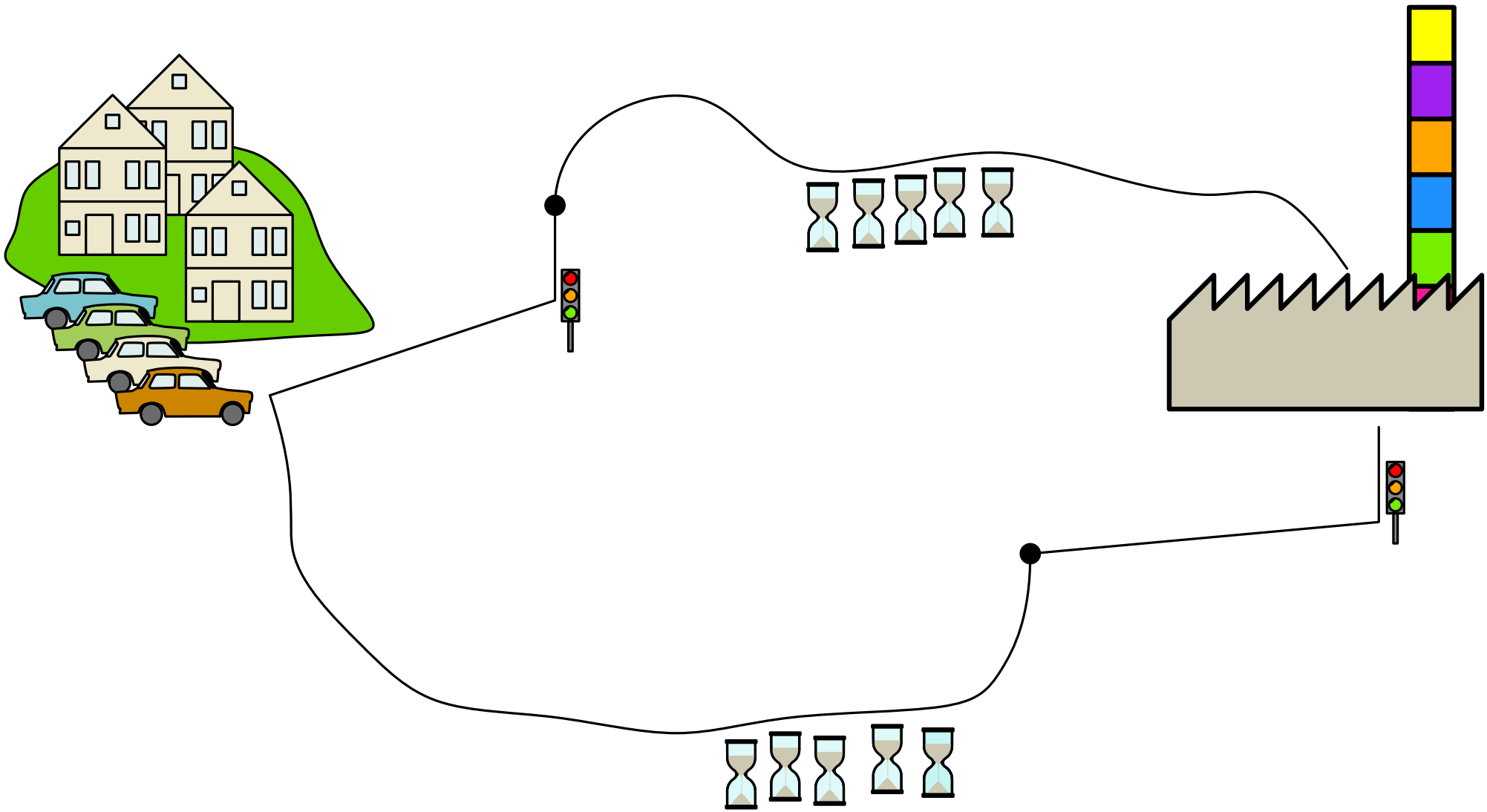
Dominik Scheder, TU Chemnitz

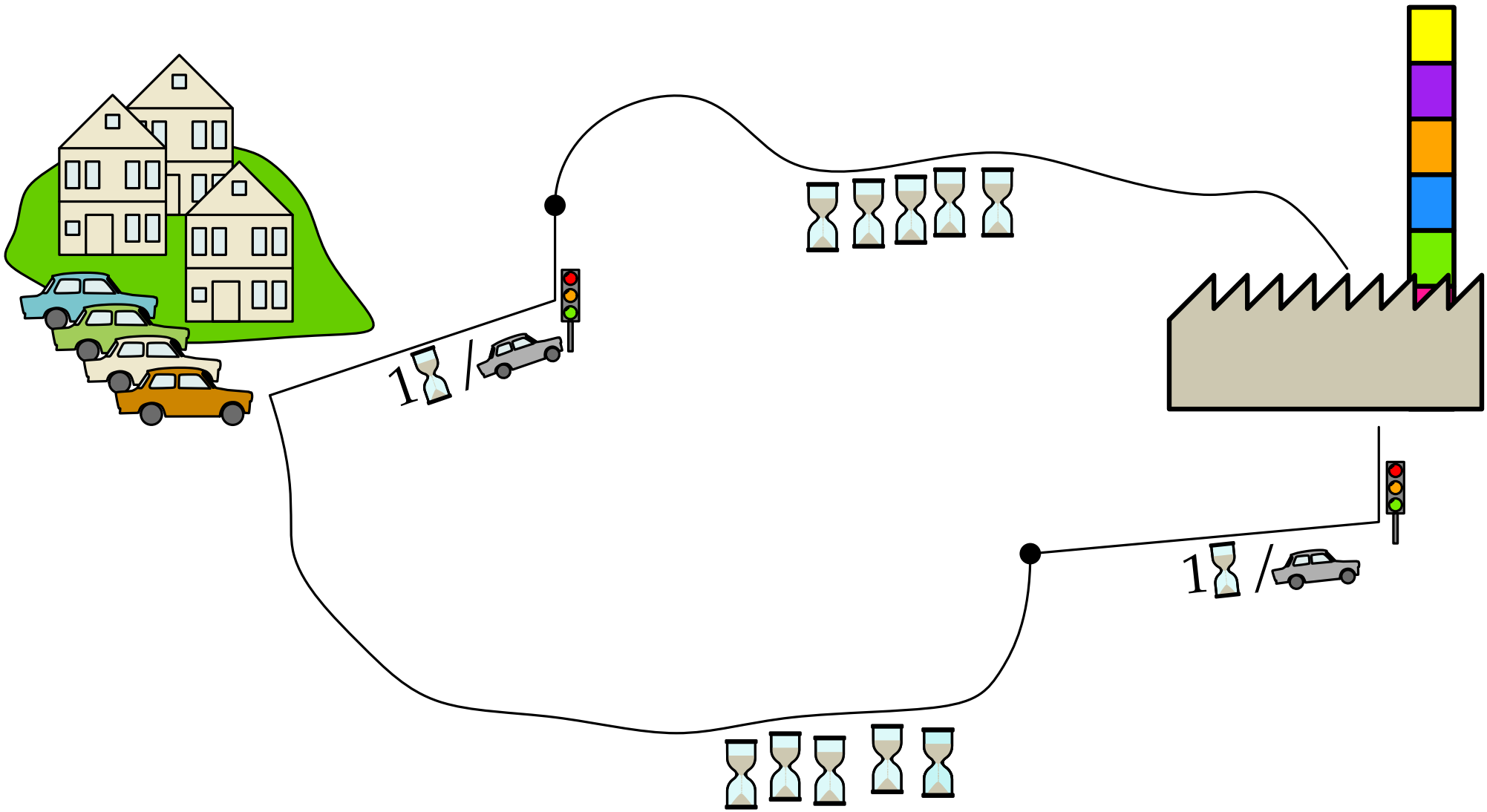
Based on joint work with Johannes Tantow, also TU Chemnitz

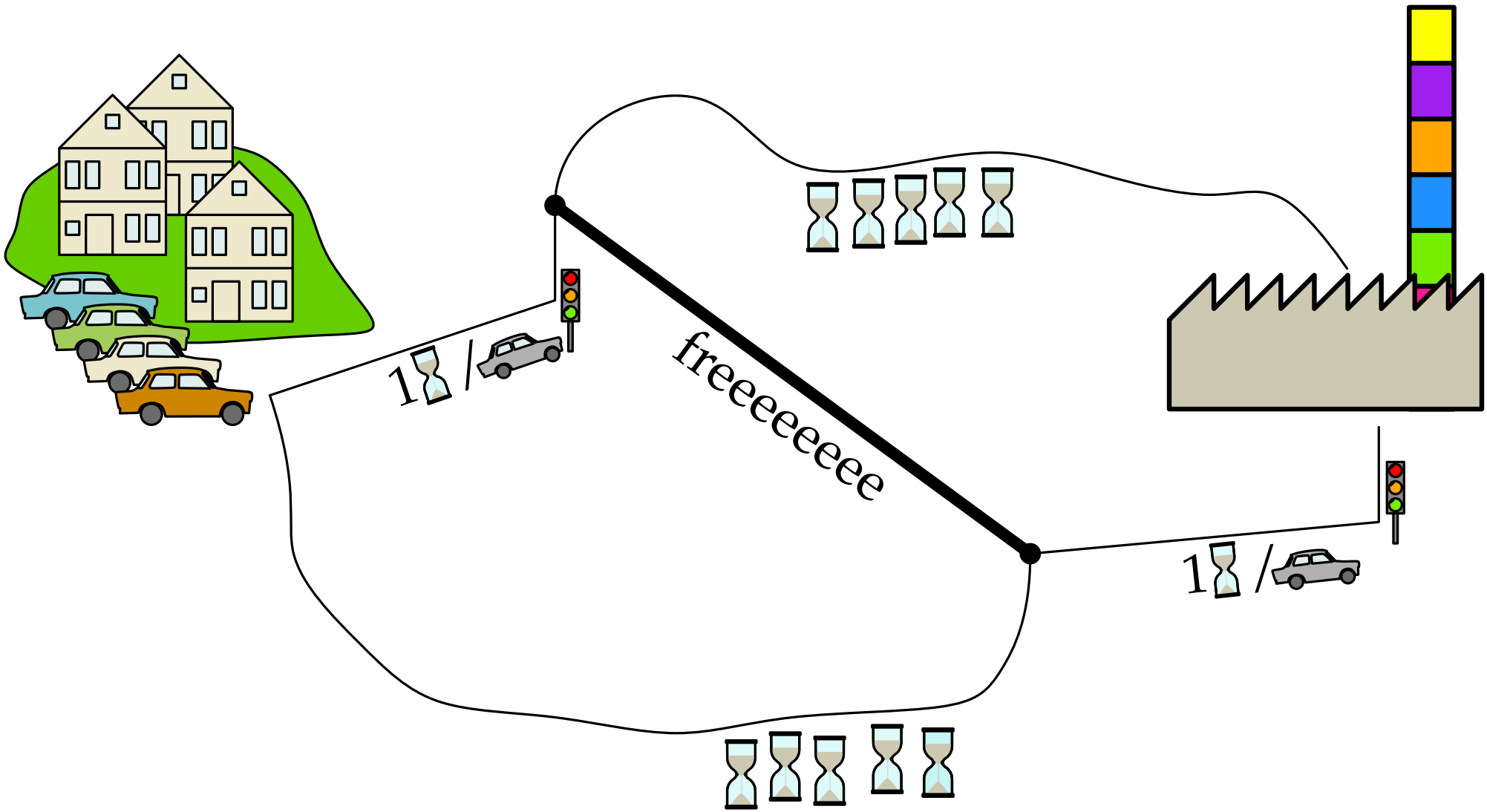


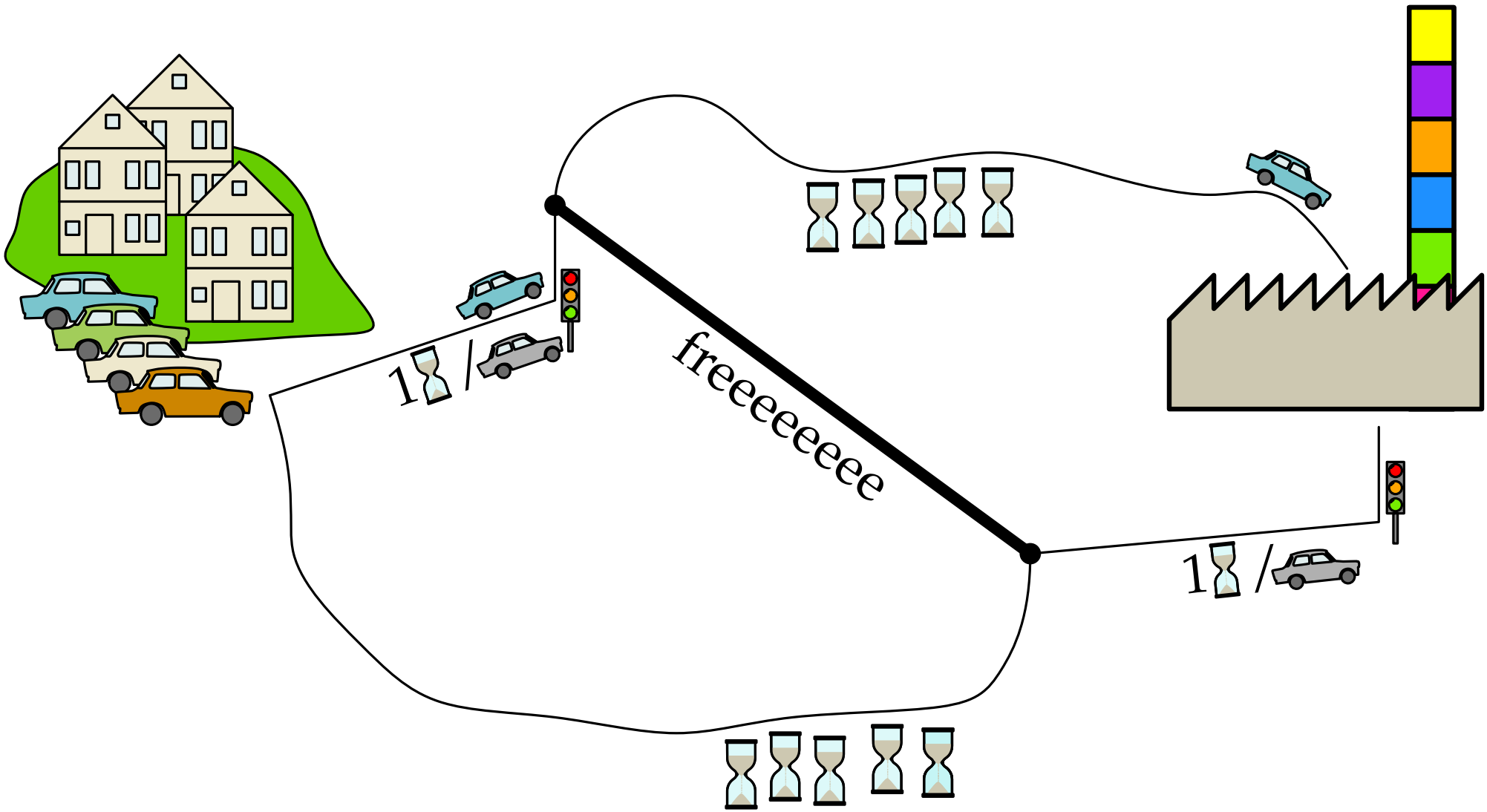


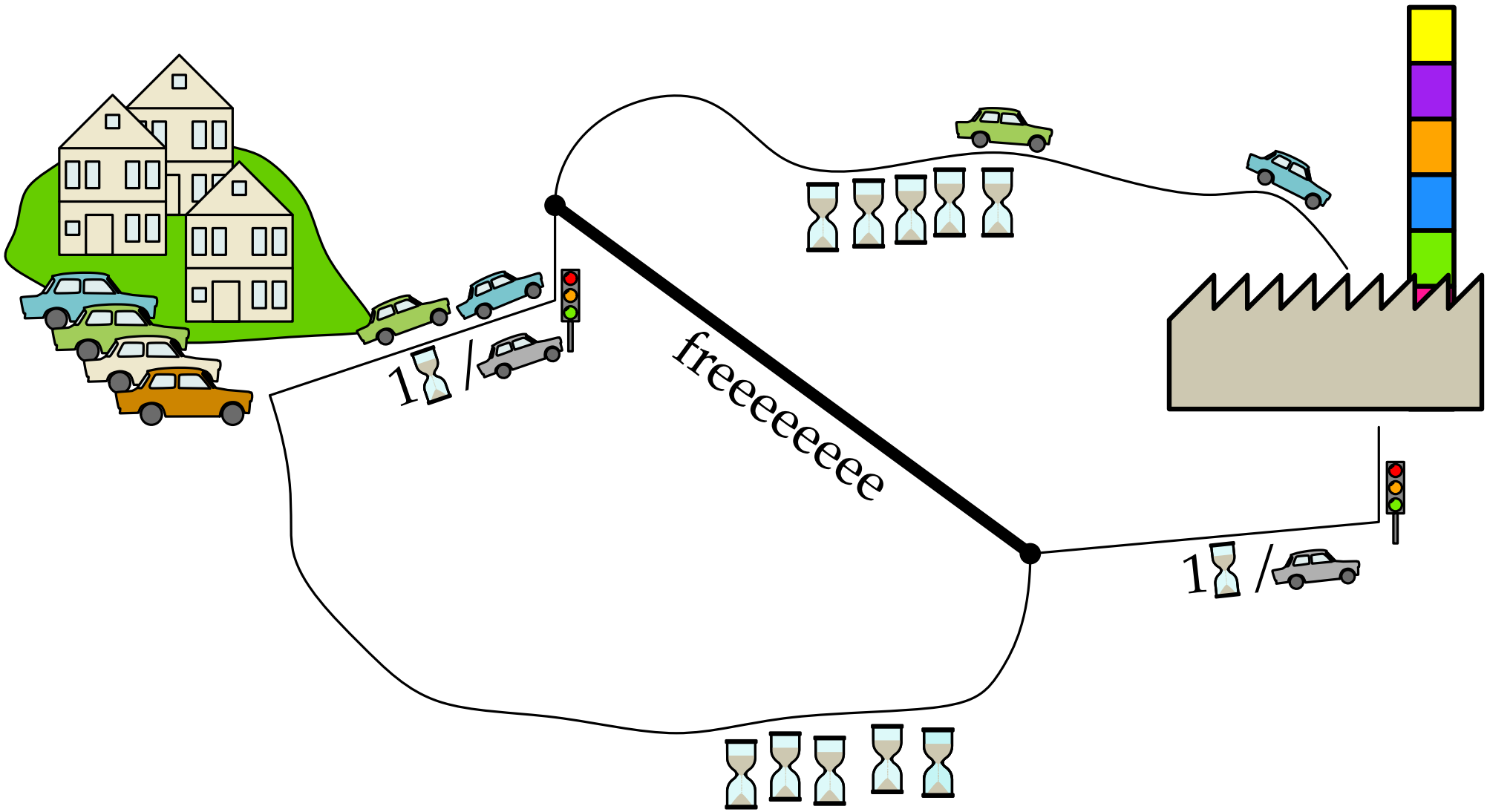


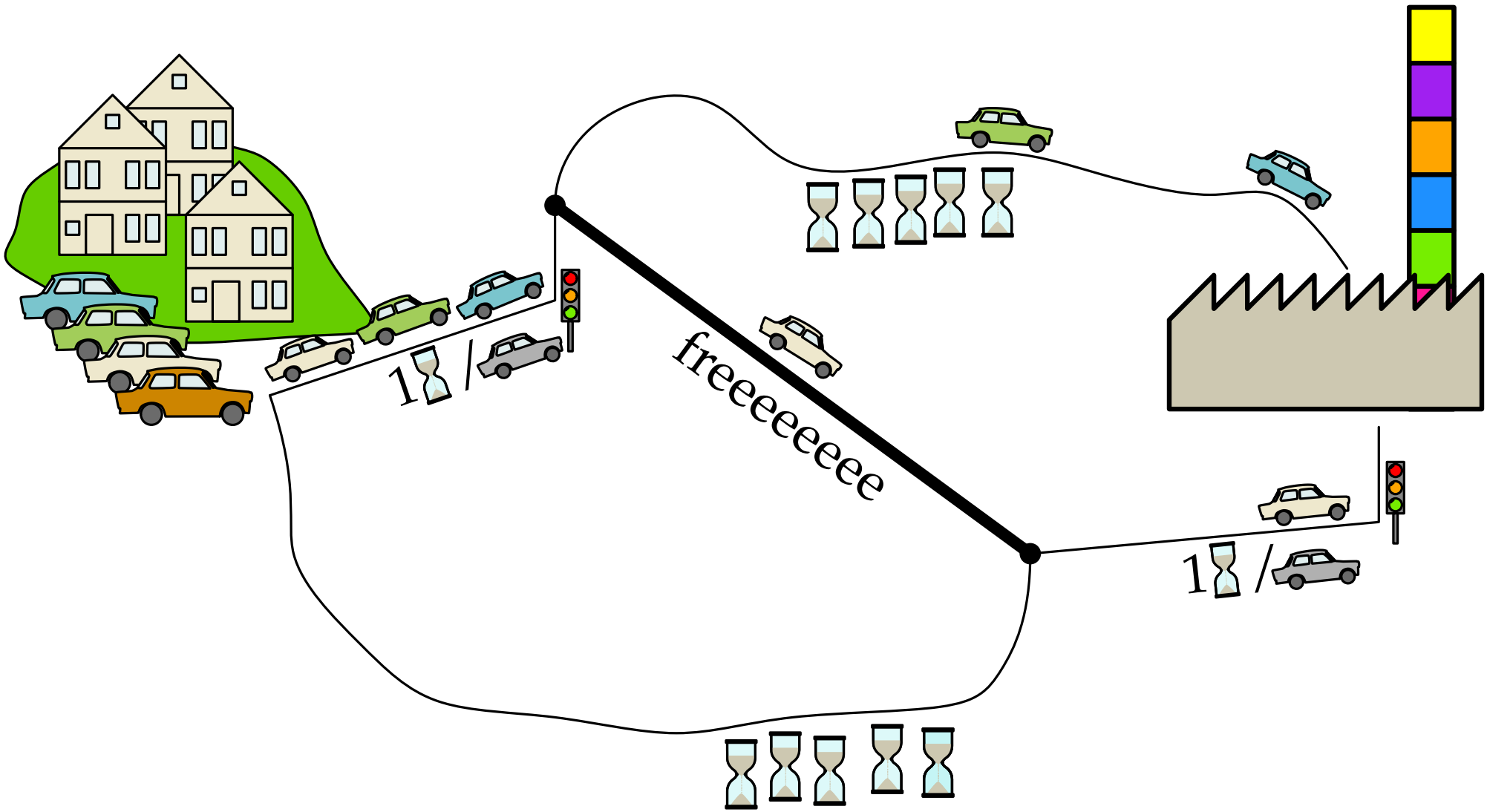


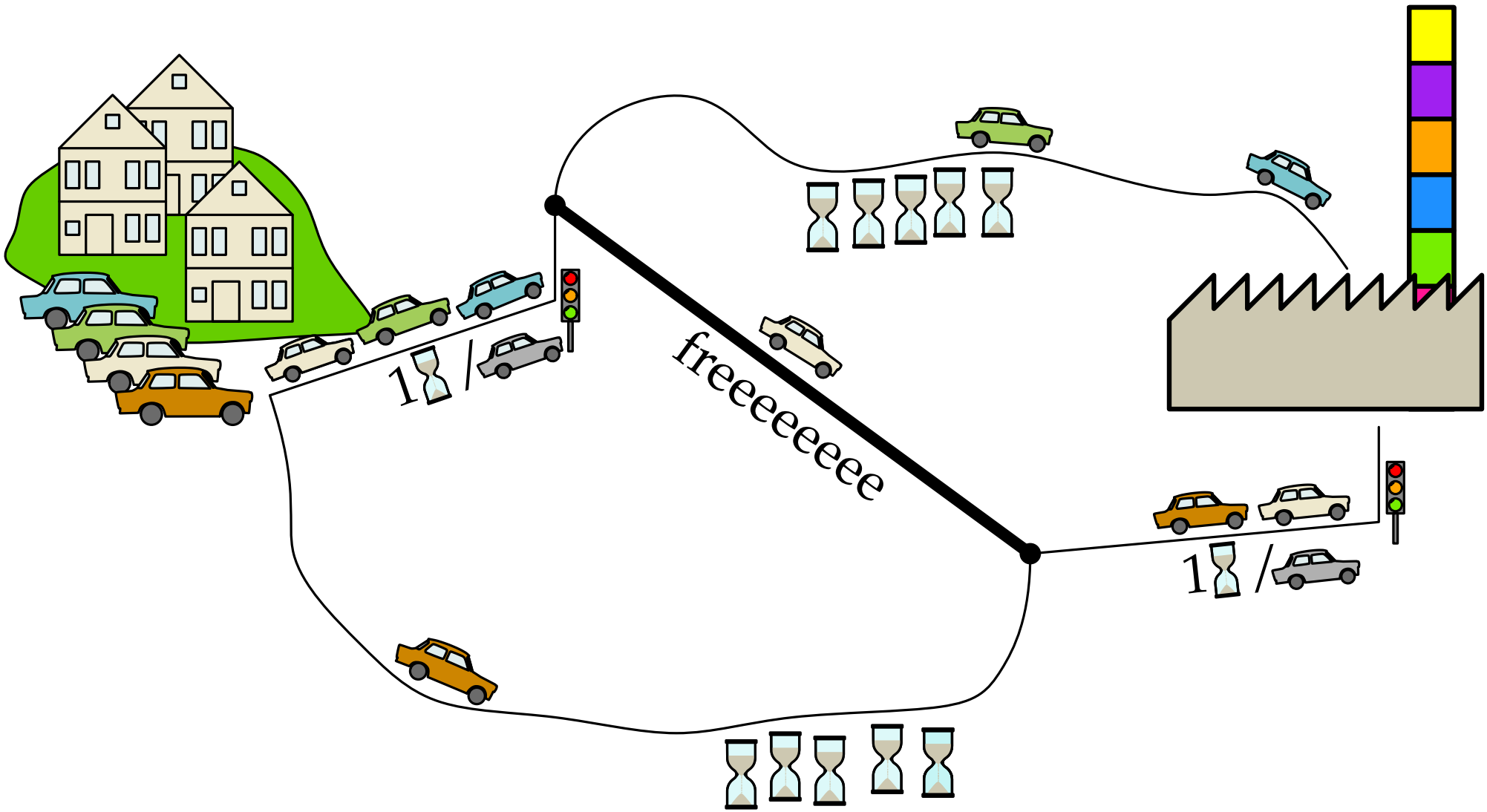


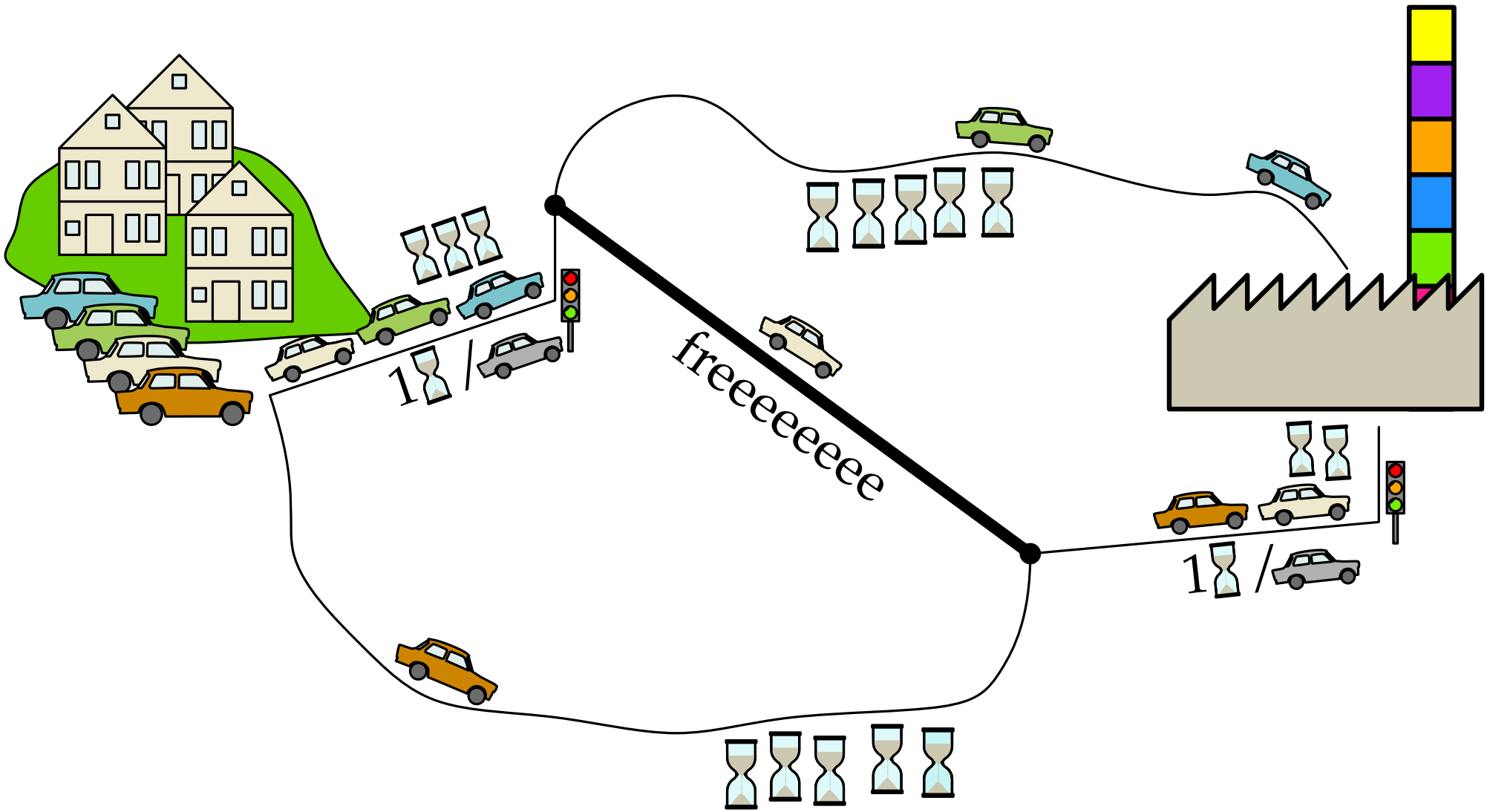

1
freeeeeeeee
1


1
freeeeeeeee
1

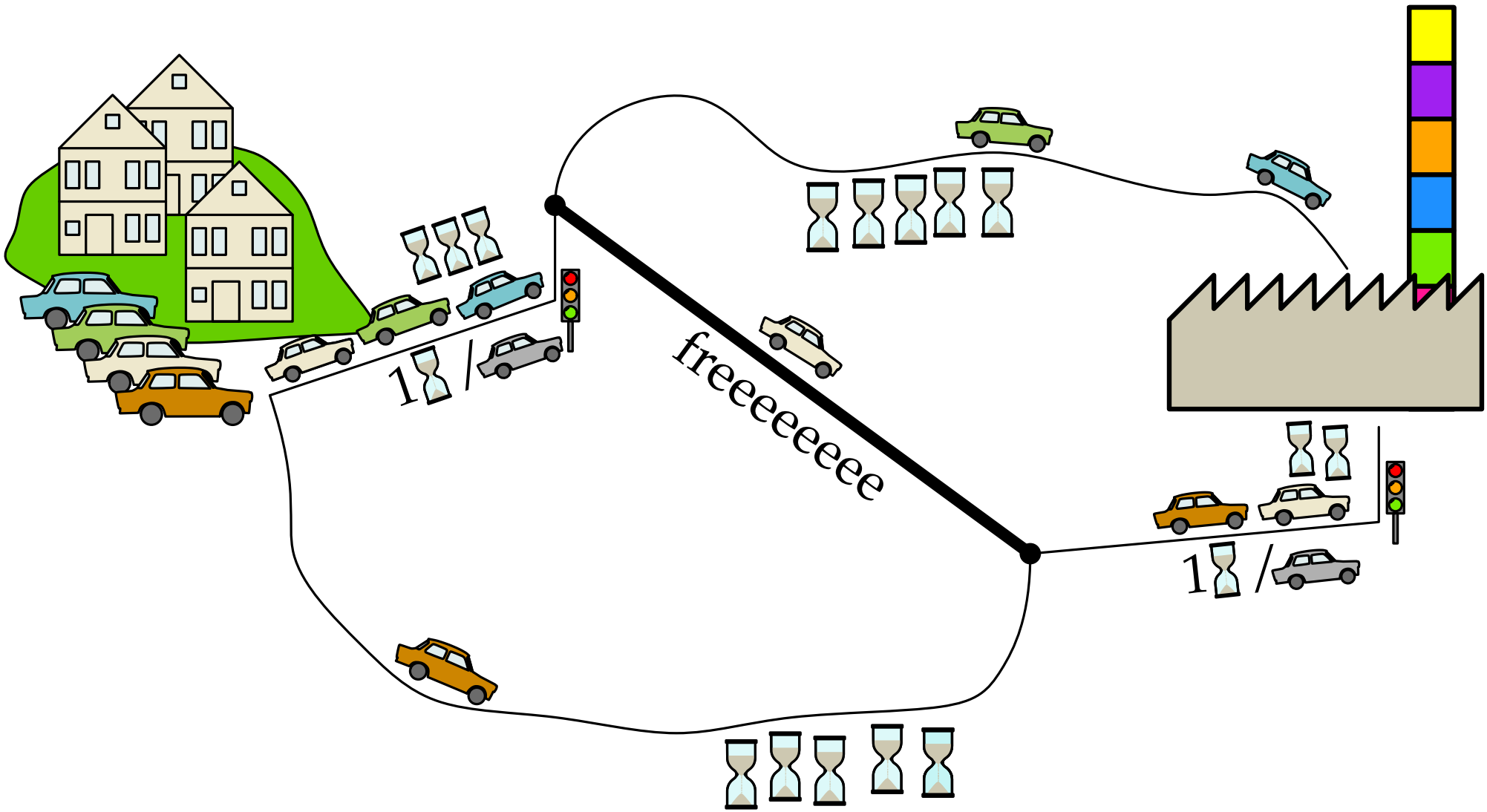

freeeeeeeee

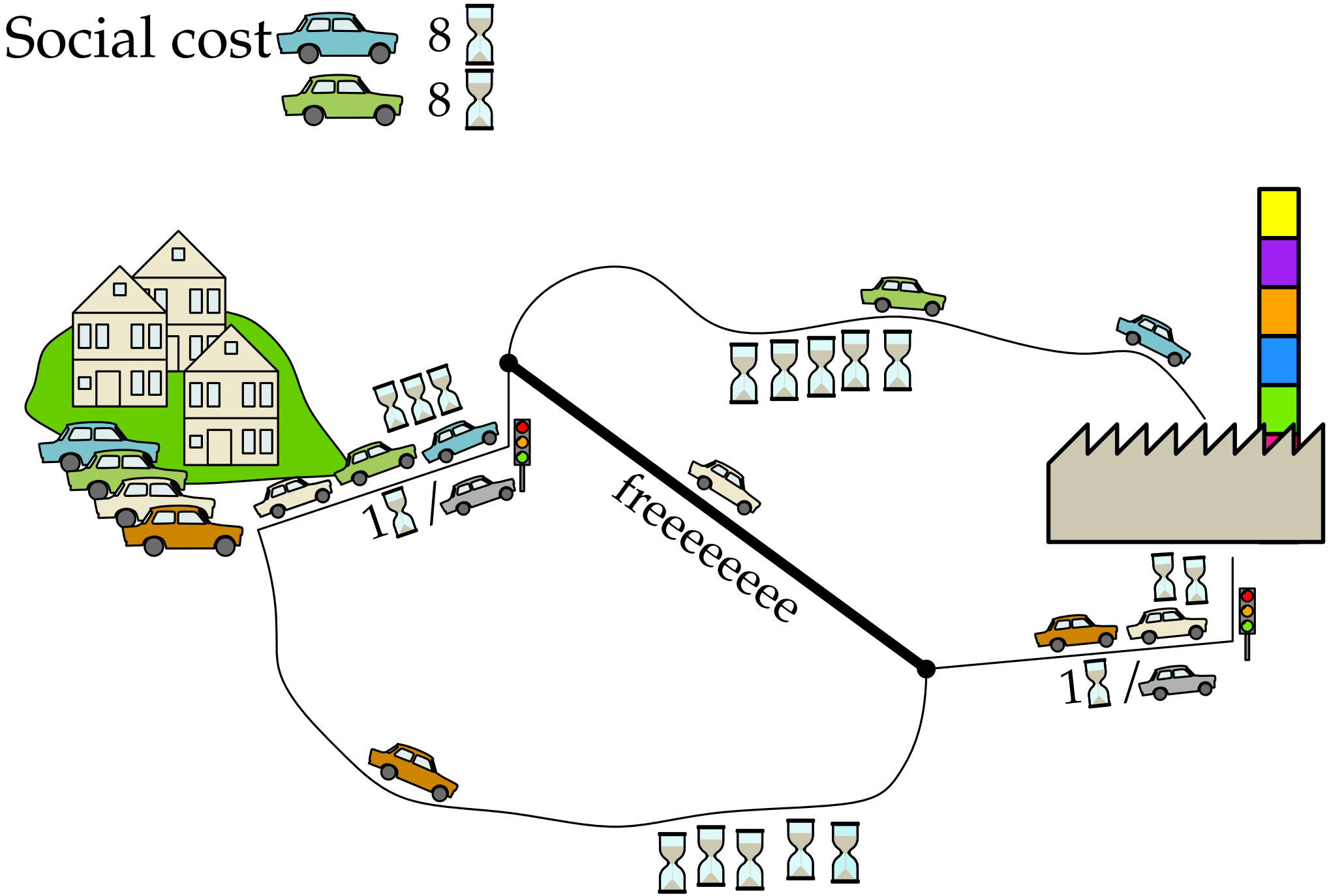

1
freeeeeeeee
1

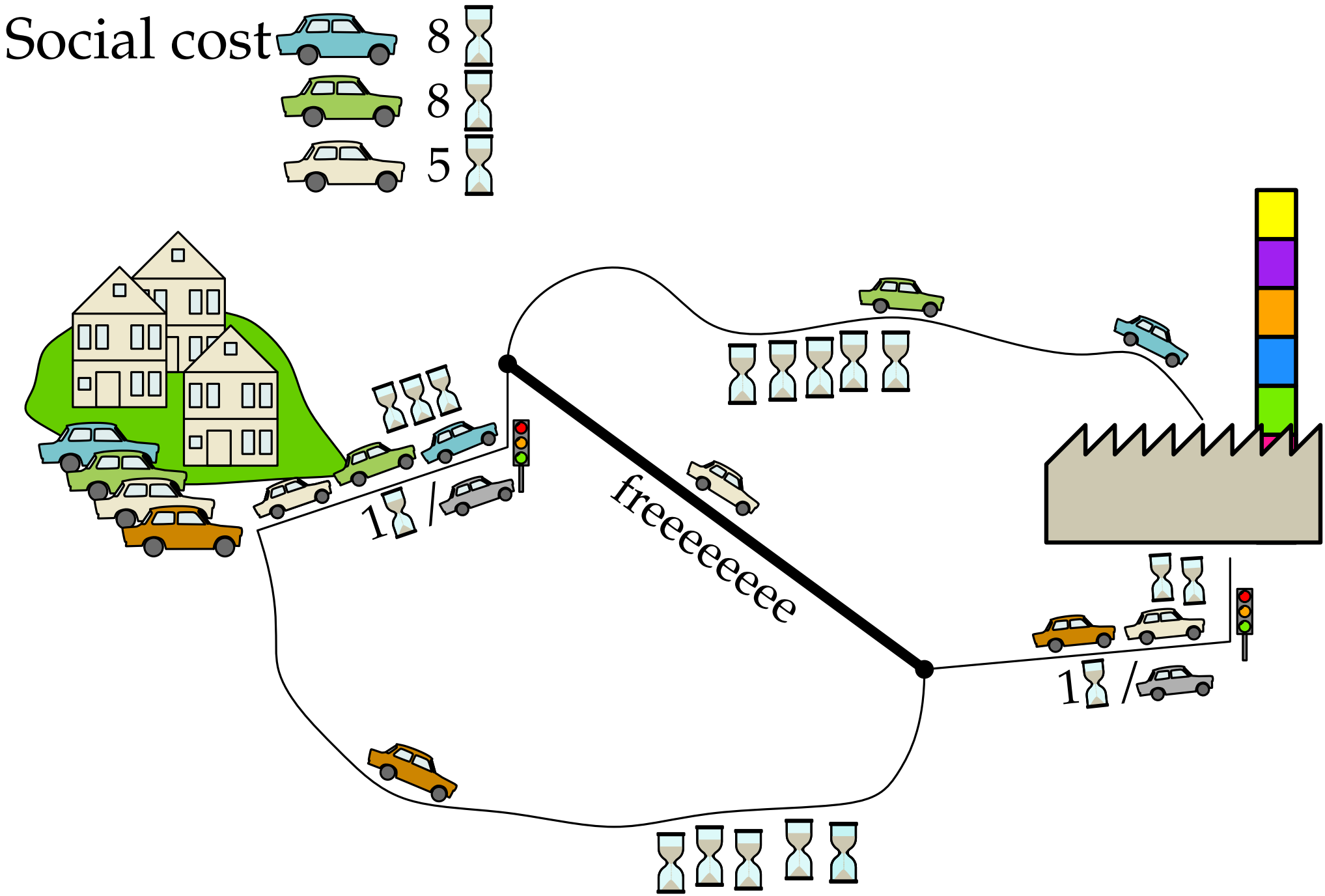

freeeeeeeee

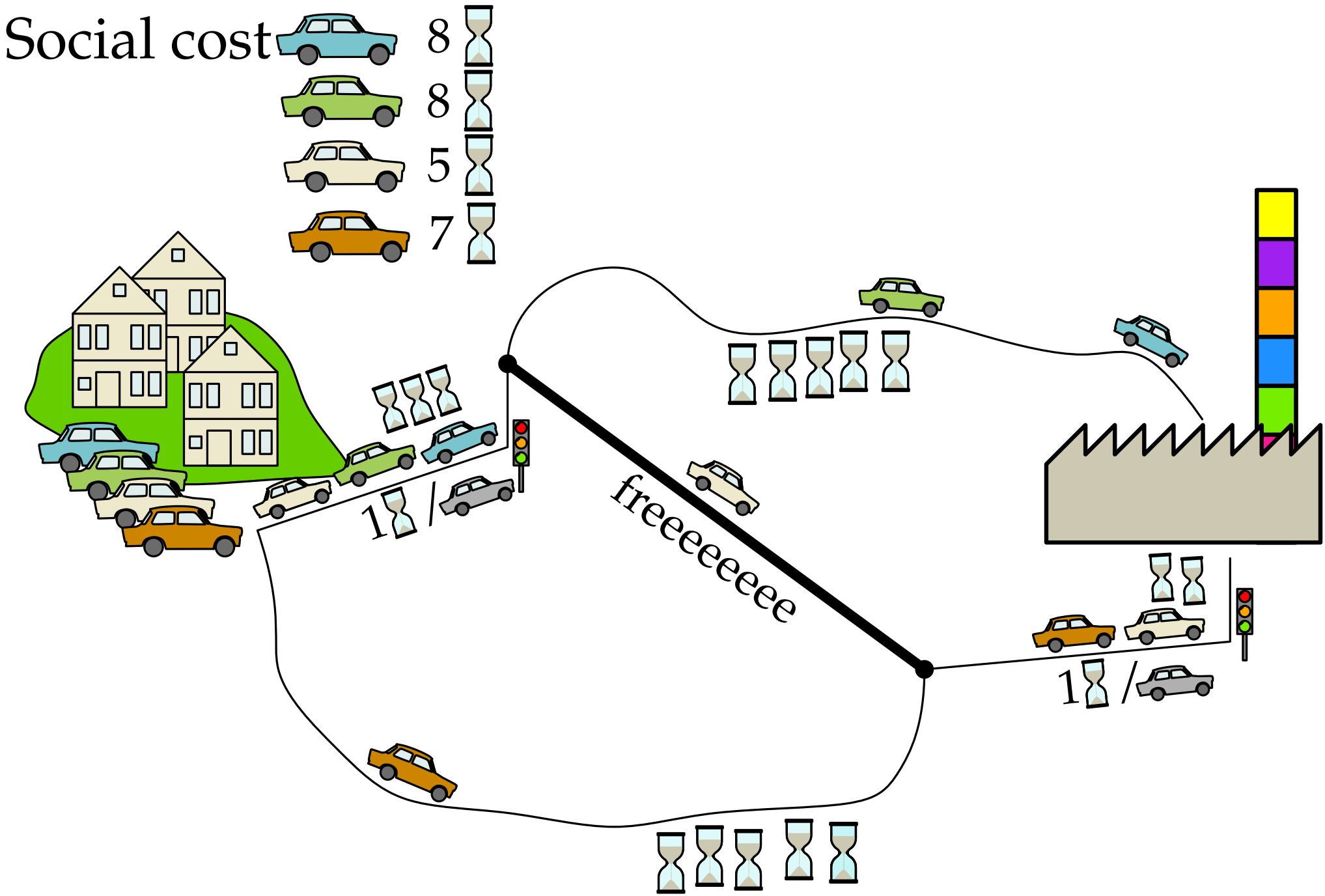

1
freeeeeeeee
1

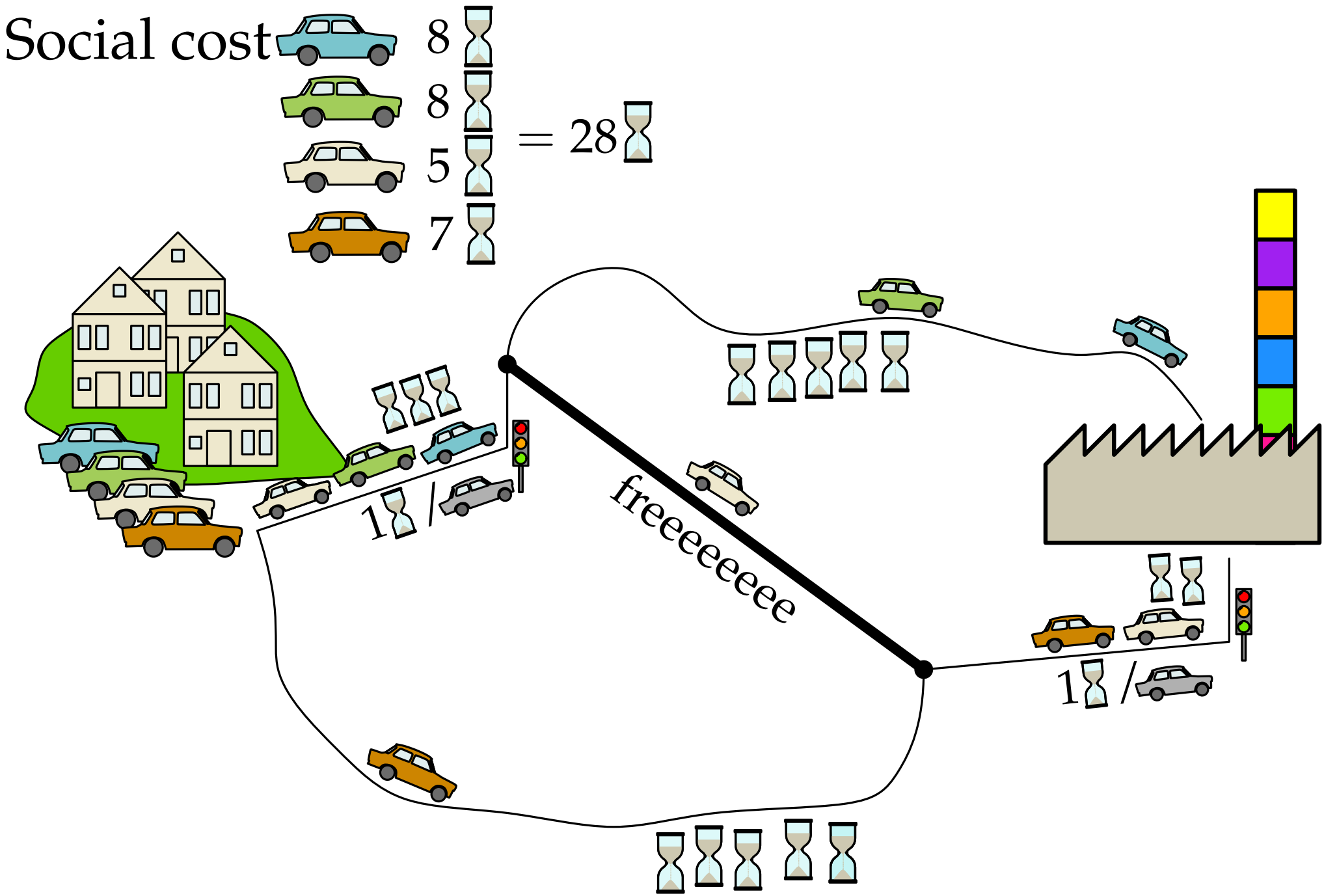
Social cost 8

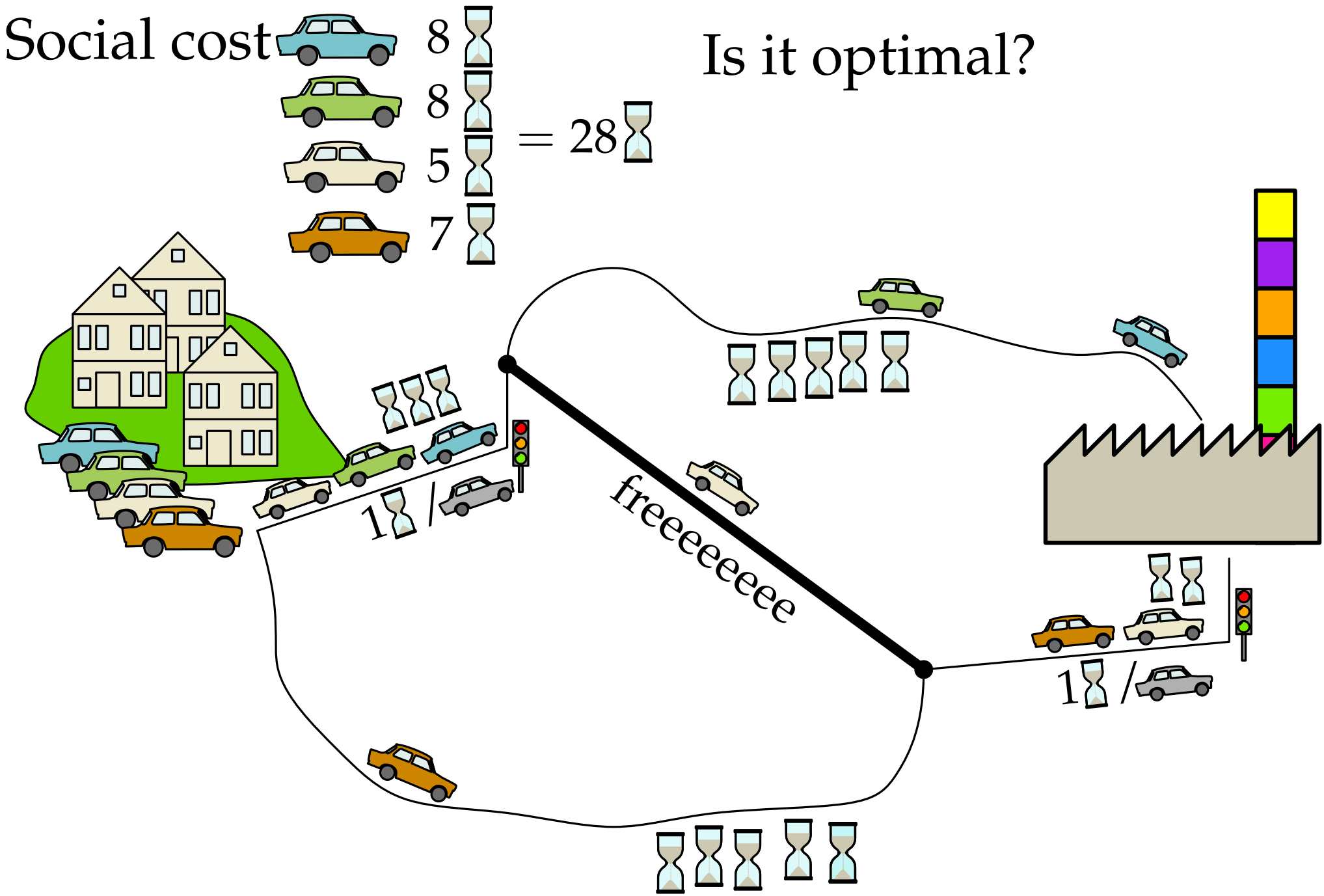



Social cost
8
8
1
freeeeeeeee
1

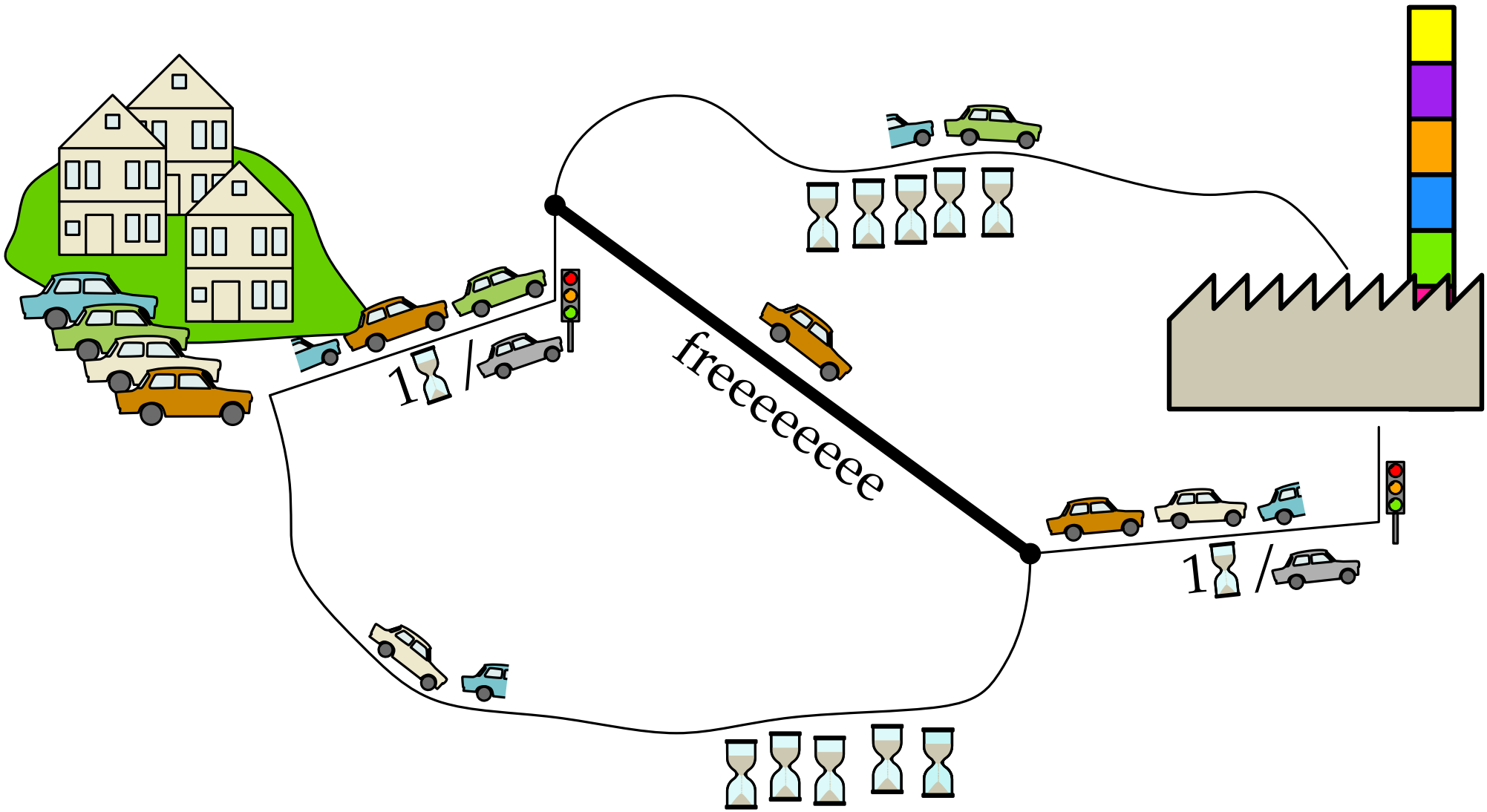

Social cost
8
8
5
1
freeeeeeeee
1

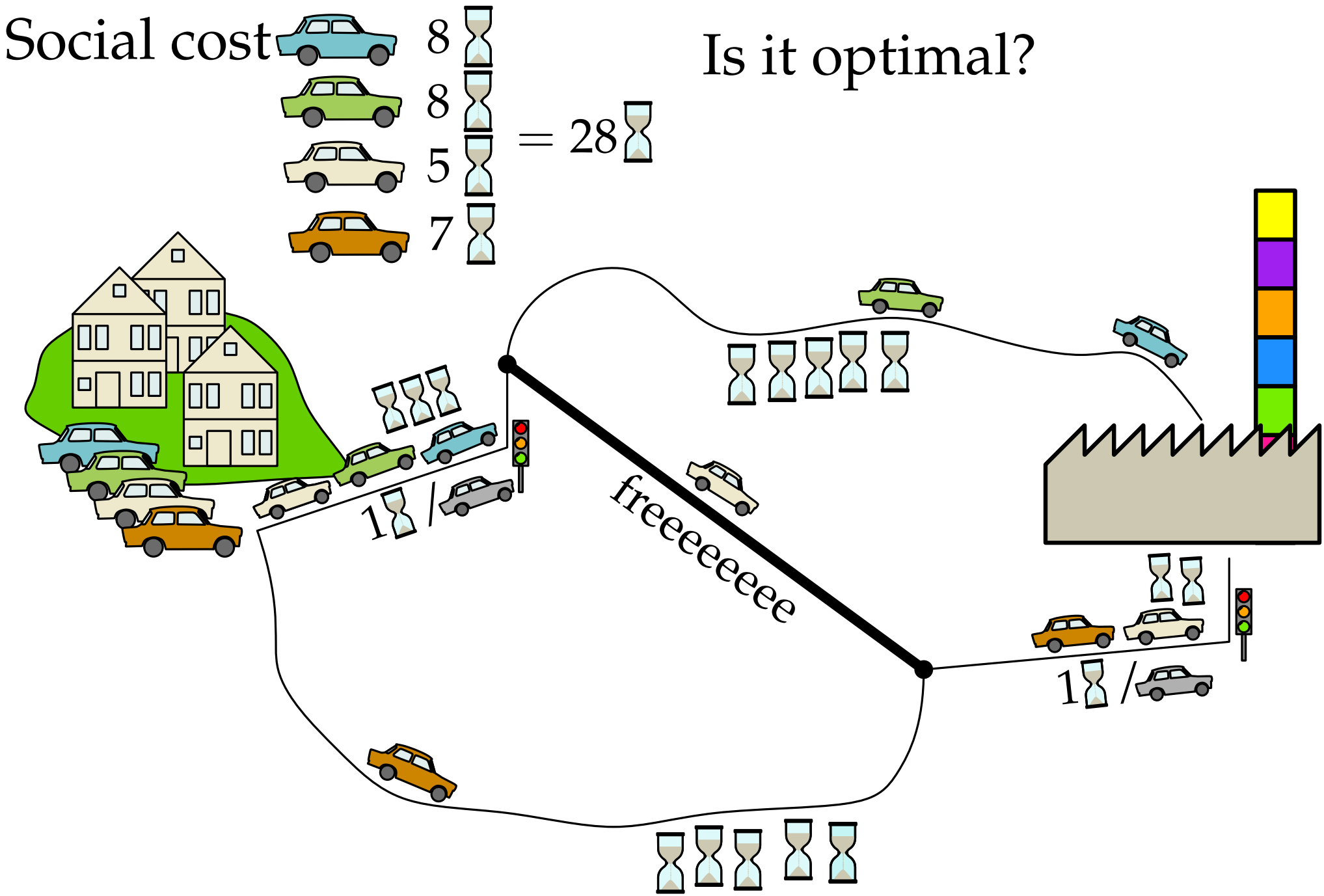

Social cost
8
8
5
7
1
freeeeeeeee
1


Social cost
8
8
5
7
= 28
1
freeeeeeeee
1


Social cost
8
8
5
7
= 28
Is it optimal?
freeeeeeeee
1
1

social cost = 27.5




Social cost
8
8
5
7
= 28
Is it optimal?
freeeeeeeee
1
1

General congestions games

General congestions games

A set $\mathcal{P}$ of players (previously: the four cars).

General congestions games

A set $\mathcal{P}$ of players (previously: the four cars).

A set $\mathcal{R}$ of resources (previously: the road segments).

General congestions games

A set $\mathcal{P}$ of players (previously: the four cars).

A set $\mathcal{R}$ of resources (previously: the road segments).

A *strategy* is a set $s \subseteq \mathcal{R}$

General congestions games

A set $\mathcal{P}$ of players (previously: the four cars).

A set $\mathcal{R}$ of resources (previously: the road segments).

A *strategy* is a set $s \subseteq \mathcal{R}$

For each player $p \in \mathcal{P}$, a set S_p of *strategies*.

General congestions games

A set $\mathcal{P}$ of players (previously: the four cars).

A set $\mathcal{R}$ of resources (previously: the road segments).

A *strategy* is a set $s \subseteq \mathcal{R}$

For each player $p \in \mathcal{P}$, a set S_p of *strategies*.

(previously: $S_p =$ the set of paths from city to factory; the same for every player)

General congestions games

A set $\mathcal{P}$ of players (previously: the four cars).

A set $\mathcal{R}$ of resources (previously: the road segments).

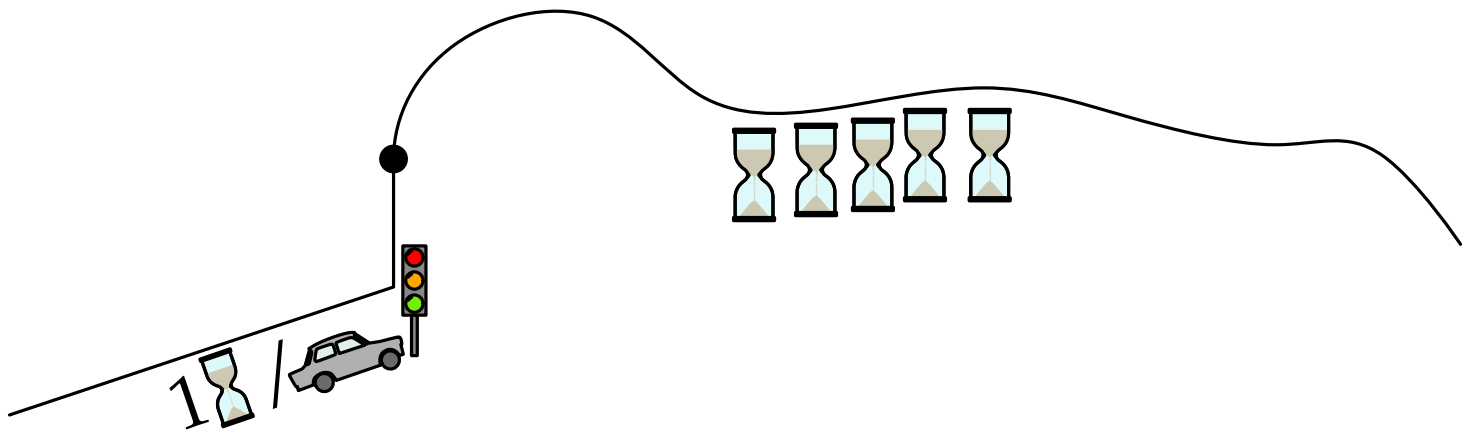
A *strategy* is a set $s \subseteq \mathcal{R}$

For each player $p \in \mathcal{P}$, a set S_p of *strategies*.

(previously: $S_p =$ the set of paths from city to factory; the same for every player)

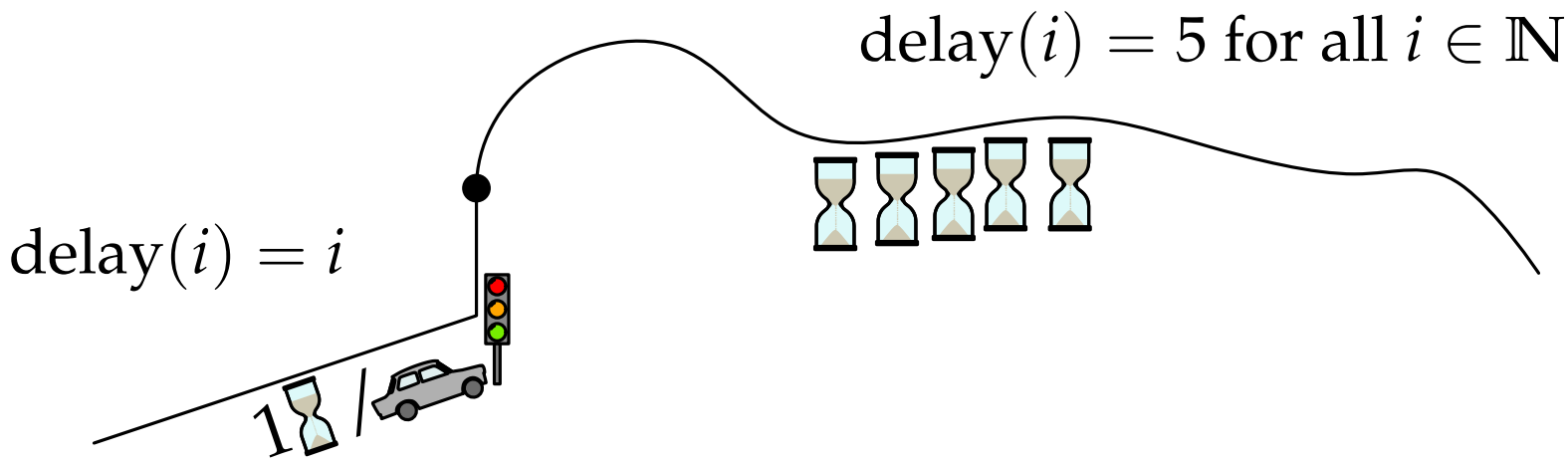
A delay function $\text{delay}_r : \mathbb{N} \to \mathbb{R}$ for each resource $r \in \mathcal{R}$.

General congestions games



A delay function $\text{delay}_r : \mathbb{N} \to \mathbb{R}$ for each resource $r \in \mathcal{R}$.

General congestions games



A delay function $\text{delay}_r : \mathbb{N} \to \mathbb{R}$ for each resource $r \in \mathcal{R}$.

General congestions games

Definition. A congestion game is given by $(\mathcal{P}, \mathcal{R}, (S_p)_{p\in\mathcal{P}}, (\text{delay}_r)_{r\in\mathcal{R}})$.

General congestions games

Definition. A congestion game is given by $(\mathcal{P}, \mathcal{R}, (S_p)_{p\in\mathcal{P}}, (\text{delay}_r)_{r\in\mathcal{R}})$.

- $\mathcal{P}$: set of players
- $\mathcal{R}$: set of resources
- $S_p \subseteq 2^{\mathcal{R}}$, set of strategies of player p
- $\text{delay}_r : \mathbb{N} \to \mathbb{R}$: delay at resource r

General congestions games

Definition. A congestion game is given by $(\mathcal{P}, \mathcal{R}, (S_p)_{p\in\mathcal{P}}, (\text{delay}_r)_{r\in\mathcal{R}})$.

- $\mathcal{P}$: set of players
- $\mathcal{R}$: set of resources
- $S_p \subseteq 2^{\mathcal{R}}$, set of strategies of player p
- $\text{delay}_r : \mathbb{N} \to \mathbb{R}$: delay at resource r

A *strategy profile* is a vector $\vec{s} = (s_p)_{p\in\mathcal{P}}$ with $s_p \in S_p$: a choice of strategy for every player.

General congestions games

Definition. A congestion game is given by $(\mathcal{P}, \mathcal{R}, (S_p)_{p\in\mathcal{P}}, (\text{delay}_r)_{r\in\mathcal{R}})$.

- $\mathcal{P}$: set of players
- $\mathcal{R}$: set of resources
- $S_p \subseteq 2^{\mathcal{R}}$, set of strategies of player p
- $\text{delay}_r : \mathbb{N} \to \mathbb{R}$: delay at resource r

A *strategy profile* is a vector $\vec{s} = (s_p)_{p\in\mathcal{P}}$ with $s_p \in S_p$: a choice of strategy for every player.

The *load* of r under $\vec{s}$ is $\text{load}(r, \vec{s}) := |\{p \in \mathcal{P} \mid r \in s_p\}|$.

General congestions games

Definition. A congestion game is given by $(\mathcal{P}, \mathcal{R}, (S_p)_{p\in\mathcal{P}}, (\text{delay}_r)_{r\in\mathcal{R}})$.

- $\mathcal{P}$: set of players
- $\mathcal{R}$: set of resources
- $S_p \subseteq 2^{\mathcal{R}}$, set of strategies of player p
- $\text{delay}_r : \mathbb{N} \to \mathbb{R}$: delay at resource r

A *strategy profile* is a vector $\vec{s} = (s_p)_{p\in\mathcal{P}}$ with $s_p \in S_p$: a choice of strategy for every player.

The *load* of r under $\vec{s}$ is $\text{load}(r, \vec{s}) := |\{p \in \mathcal{P} \mid r \in s_p\}|$.

(the number of players using it)

General congestions games

Definition. A congestion game is given by $(\mathcal{P}, \mathcal{R}, (S_p)_{p\in\mathcal{P}}, (\text{delay}_r)_{r\in\mathcal{R}})$.

- $\mathcal{P}$: set of players
- $\mathcal{R}$: set of resources
- $S_p \subseteq 2^{\mathcal{R}}$, set of strategies of player p
- $\text{delay}_r : \mathbb{N} \to \mathbb{R}$: delay at resource r

A *strategy profile* is a vector $\vec{s} = (s_p)_{p\in\mathcal{P}}$ with $s_p \in S_p$: a choice of strategy for every player.

The *load* of r under $\vec{s}$ is $\text{load}(r, \vec{s}) := |\{p \in \mathcal{P} \mid r \in s_p\}|$.
The *delay at* r is $\text{delay}_r(\text{load}(r, \vec{s}))$.

General congestions games

Definition. A congestion game is given by $(\mathcal{P}, \mathcal{R}, (S_p)_{p\in\mathcal{P}}, (\text{delay}_r)_{r\in\mathcal{R}})$.

- $\mathcal{P}$: set of players
- $\mathcal{R}$: set of resources
- $S_p \subseteq 2^{\mathcal{R}}$, set of strategies of player p
- $\text{delay}_r : \mathbb{N} \to \mathbb{R}$: delay at resource r

A *strategy profile* is a vector $\vec{s} = (s_p)_{p\in\mathcal{P}}$ with $s_p \in S_p$: a choice of strategy for every player.

The *load* of r under $\vec{s}$ is $\text{load}(r, \vec{s}) := |\{p \in \mathcal{P} \mid r \in s_p\}|$.
The *delay at* r is $\text{delay}_r(\text{load}(r, \vec{s}))$.

$$\text{The cost for } p \text{ is } \text{cost}(p, \vec{s}) = \sum_{r\in s_p} \text{delay}_r(\text{load}(r, \vec{s}))$$

Yet Another Optimization Problem

Problem. Given a congestion game $(\mathcal{P}, \mathcal{R}, (S_p)_{p\in\mathcal{P}}, (\text{delay}_r)_{r\in\mathcal{R}})$, determine a strategy profile $\vec{s}$ that achieves the social optimum, i.e., that minimizes

$$\text{socialcost}(\vec{s}) := \sum_{p\in\mathcal{P}} \text{cost}(p, \vec{s}) \ .$$

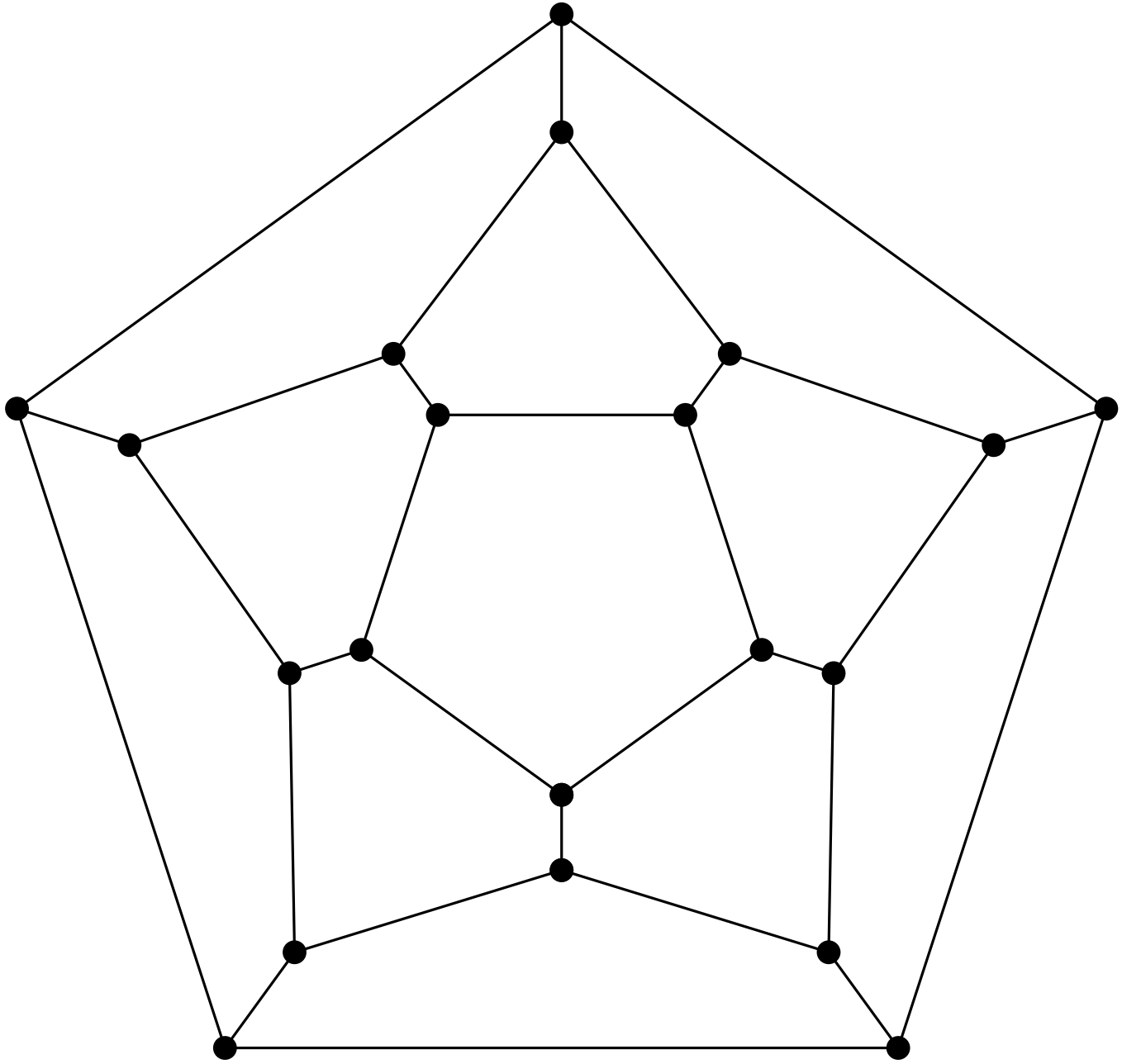
Yet Another Optimization Problem

Problem. Given a congestion game $(\mathcal{P}, \mathcal{R}, (S_p)_{p\in\mathcal{P}}, (\text{delay}_r)_{r\in\mathcal{R}})$, determine a strategy profile $\vec{s}$ that achieves the social optimum, i.e., that minimizes

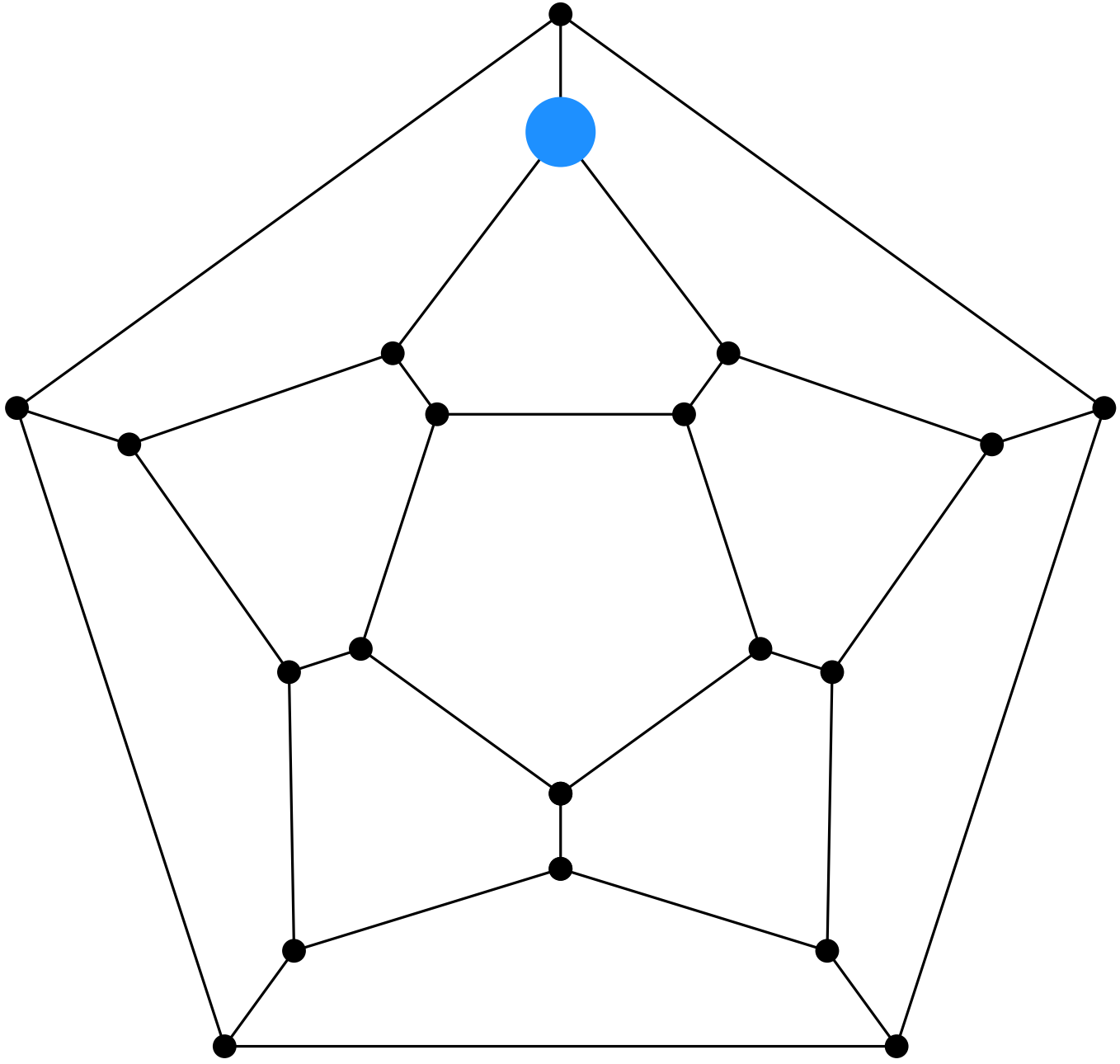
$$\text{socialcost}(\vec{s}) := \sum_{p\in\mathcal{P}} \text{cost}(p, \vec{s}) \ .$$

Theorem. This is NP-hard because we can use it to find a maximum independent set.

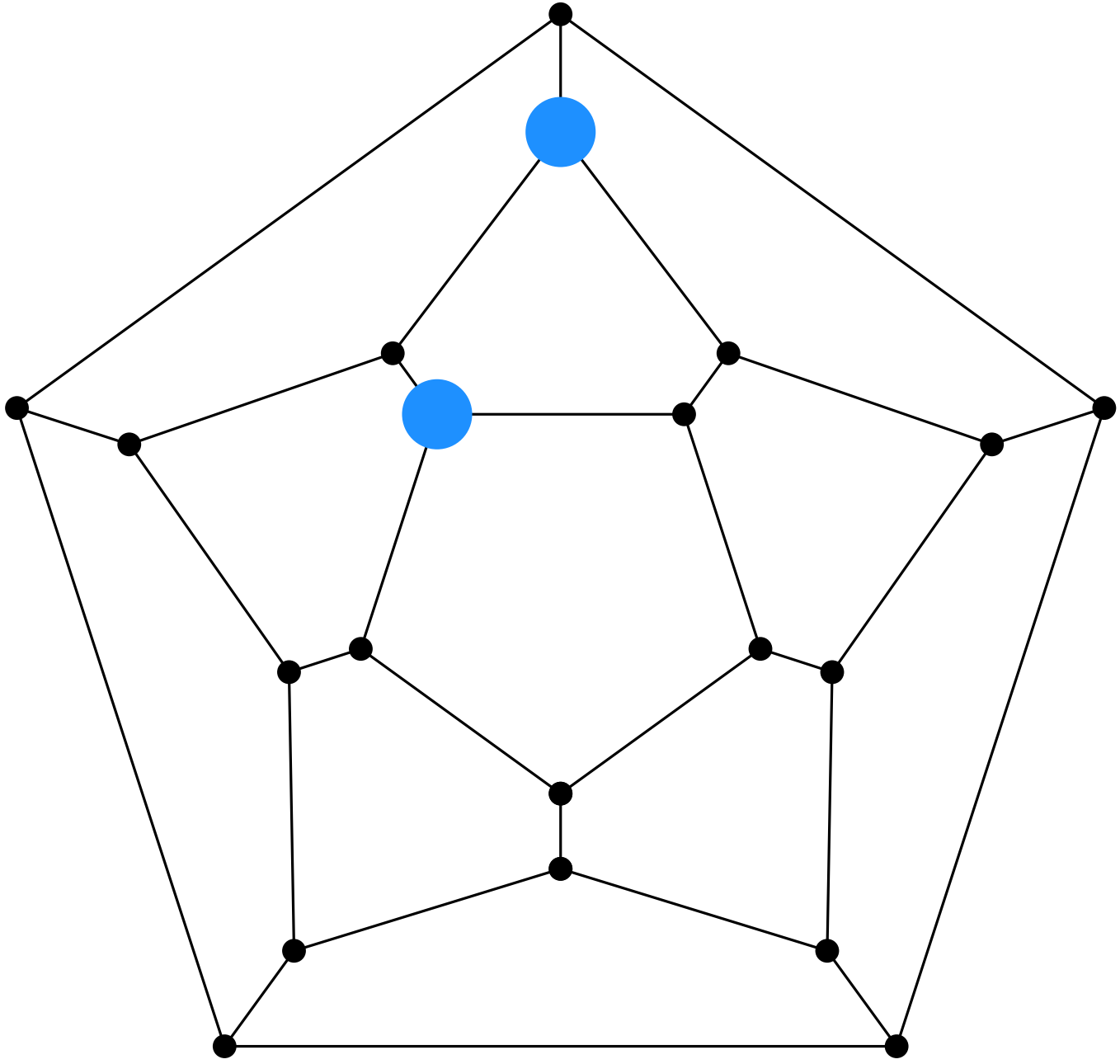
The Maximum Independent Set Problem



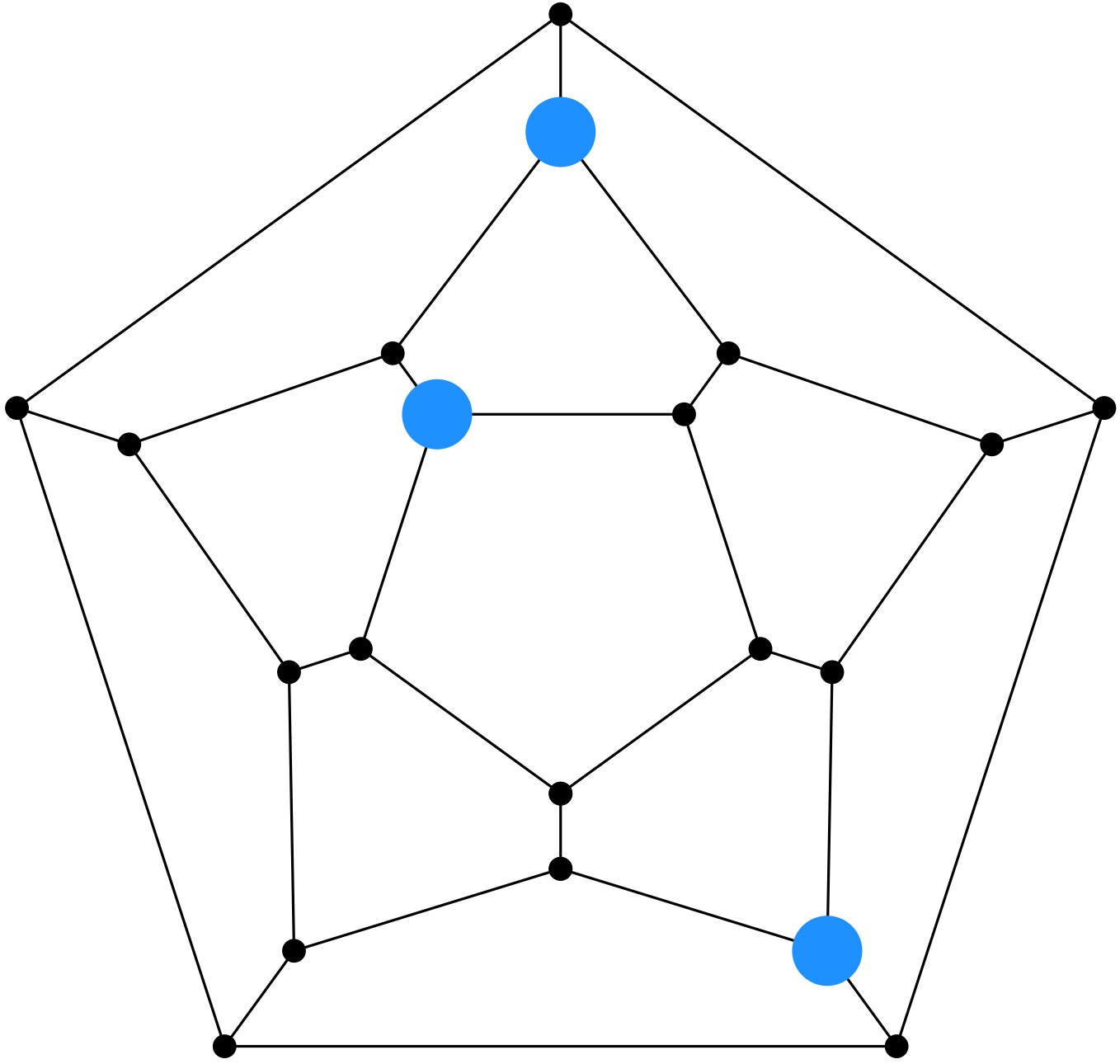
The Maximum Independent Set Problem



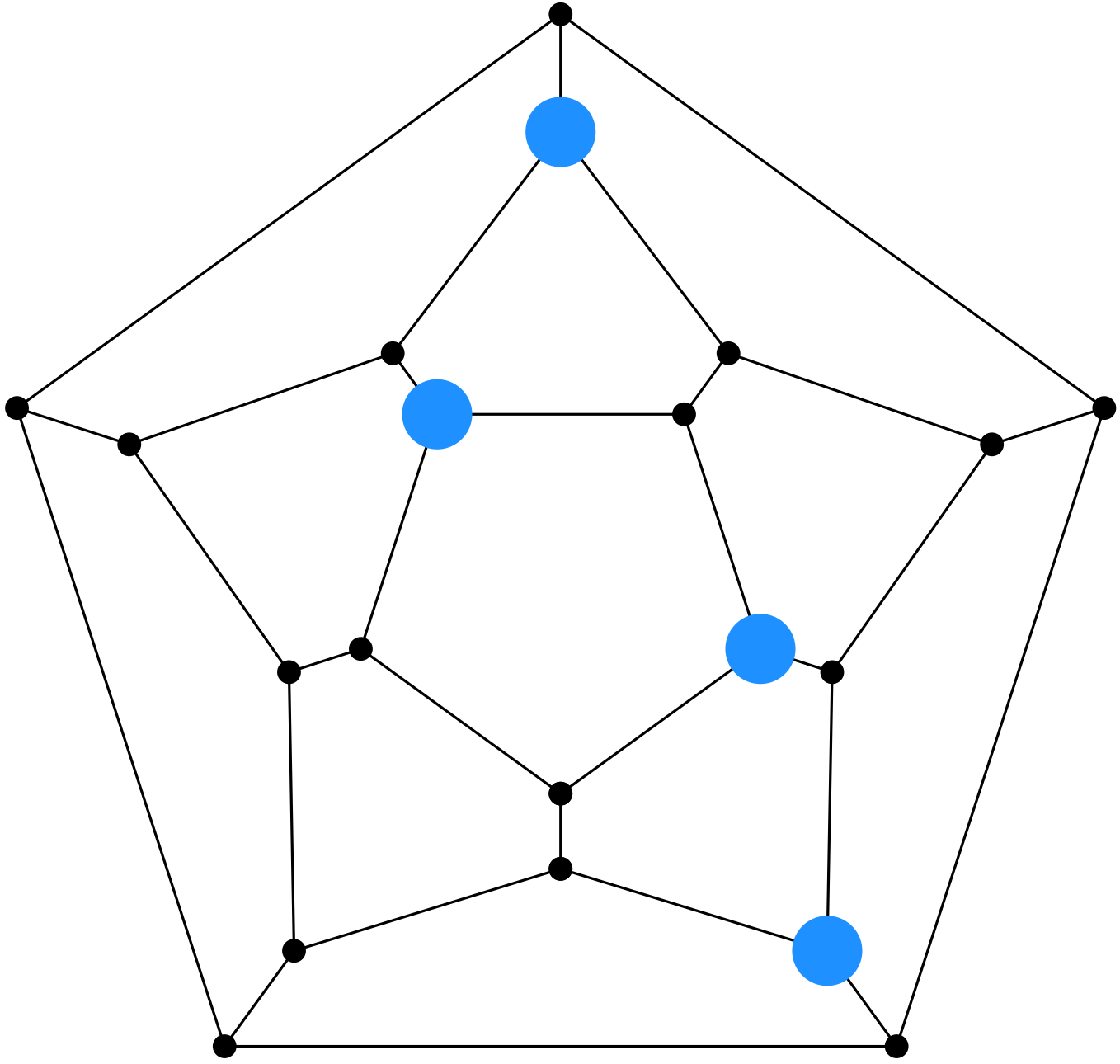
The Maximum Independent Set Problem



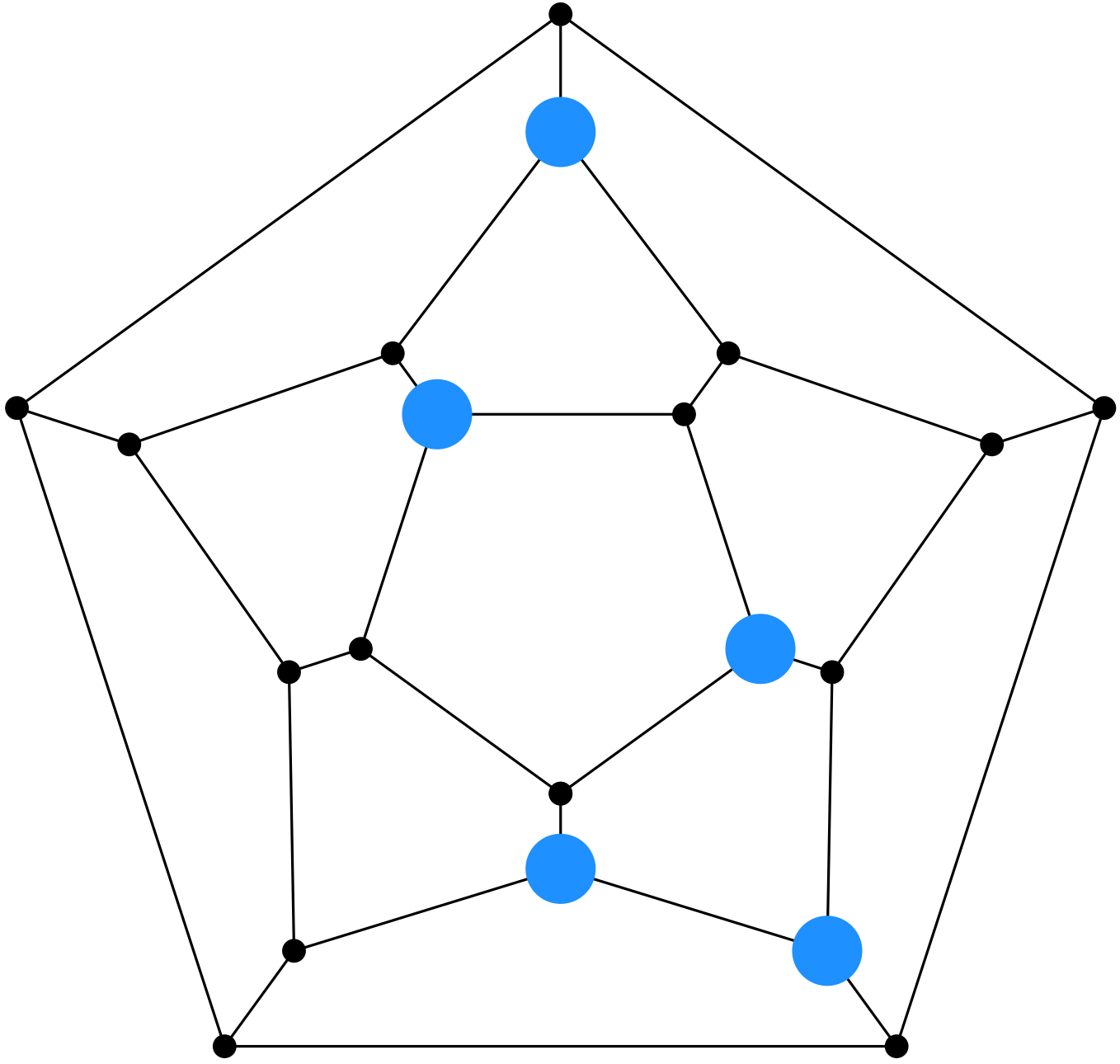
The Maximum Independent Set Problem



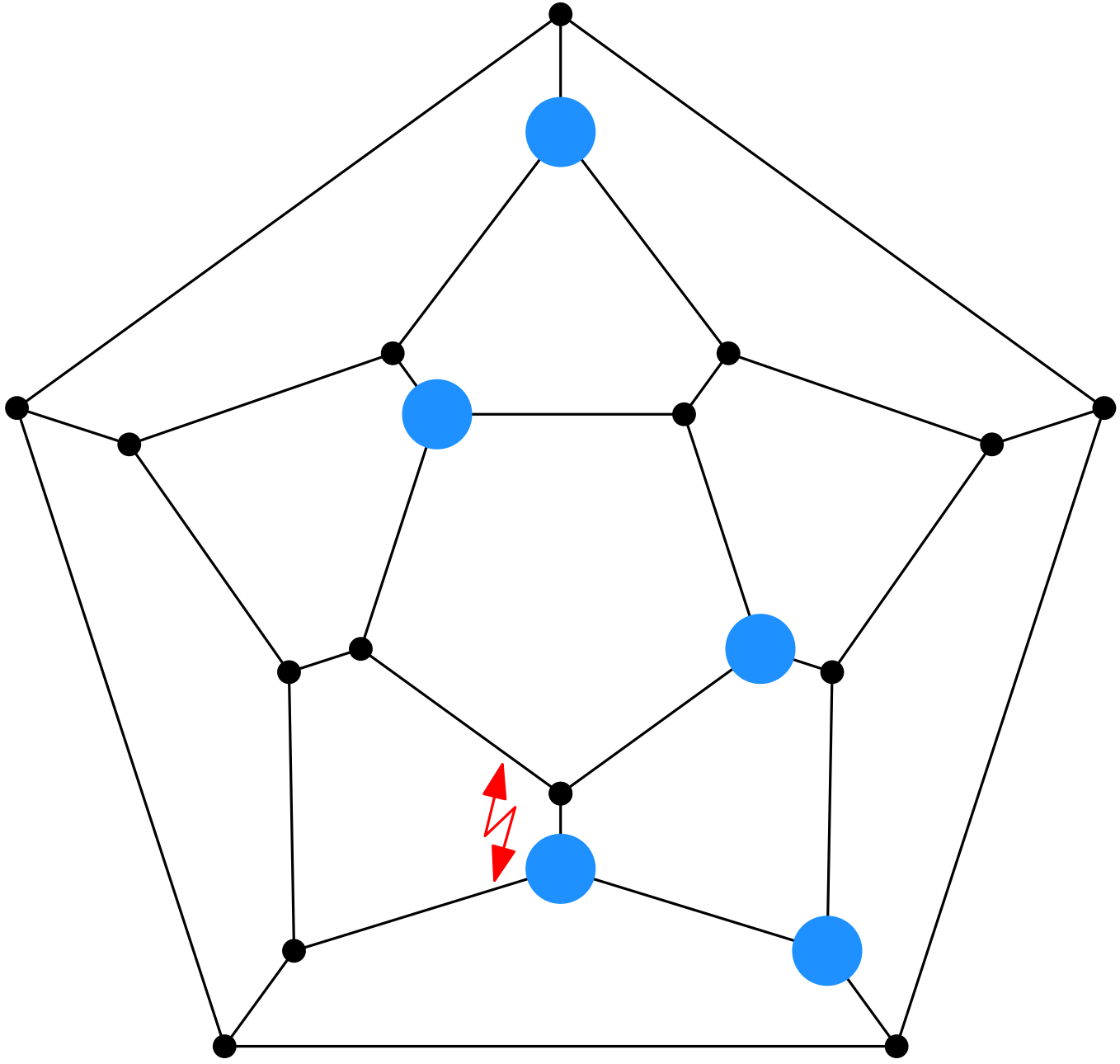
The Maximum Independent Set Problem



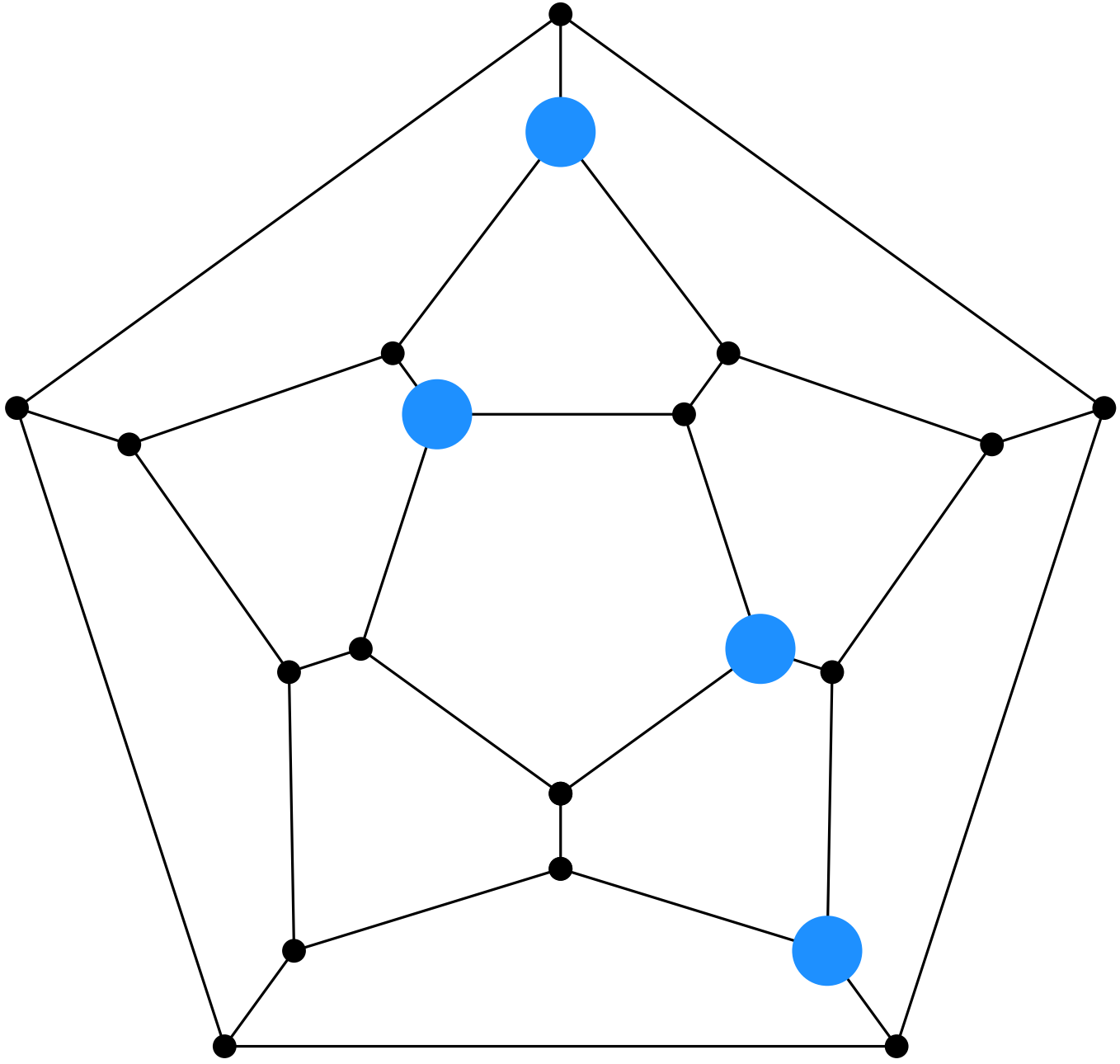
The Maximum Independent Set Problem



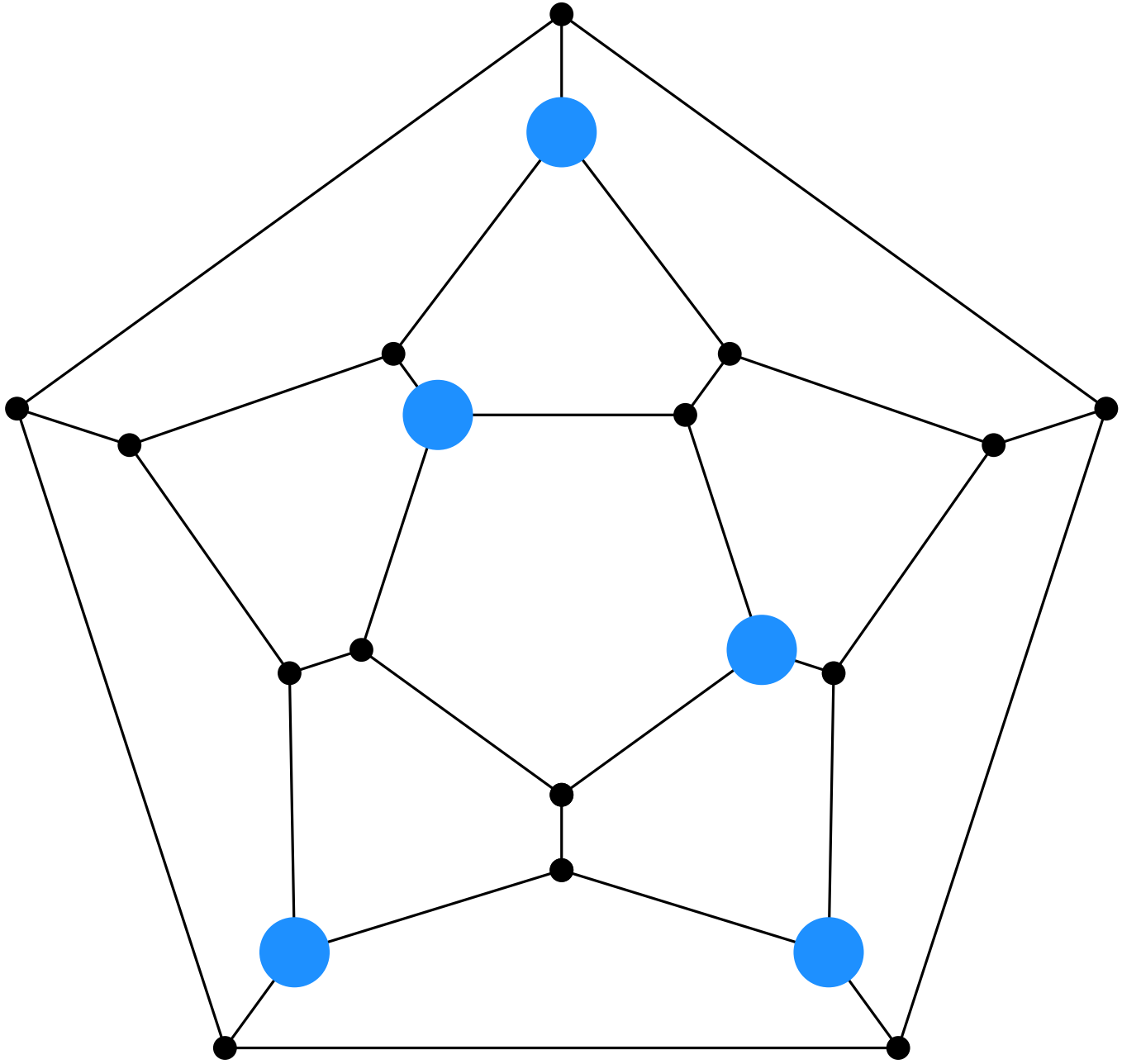
The Maximum Independent Set Problem



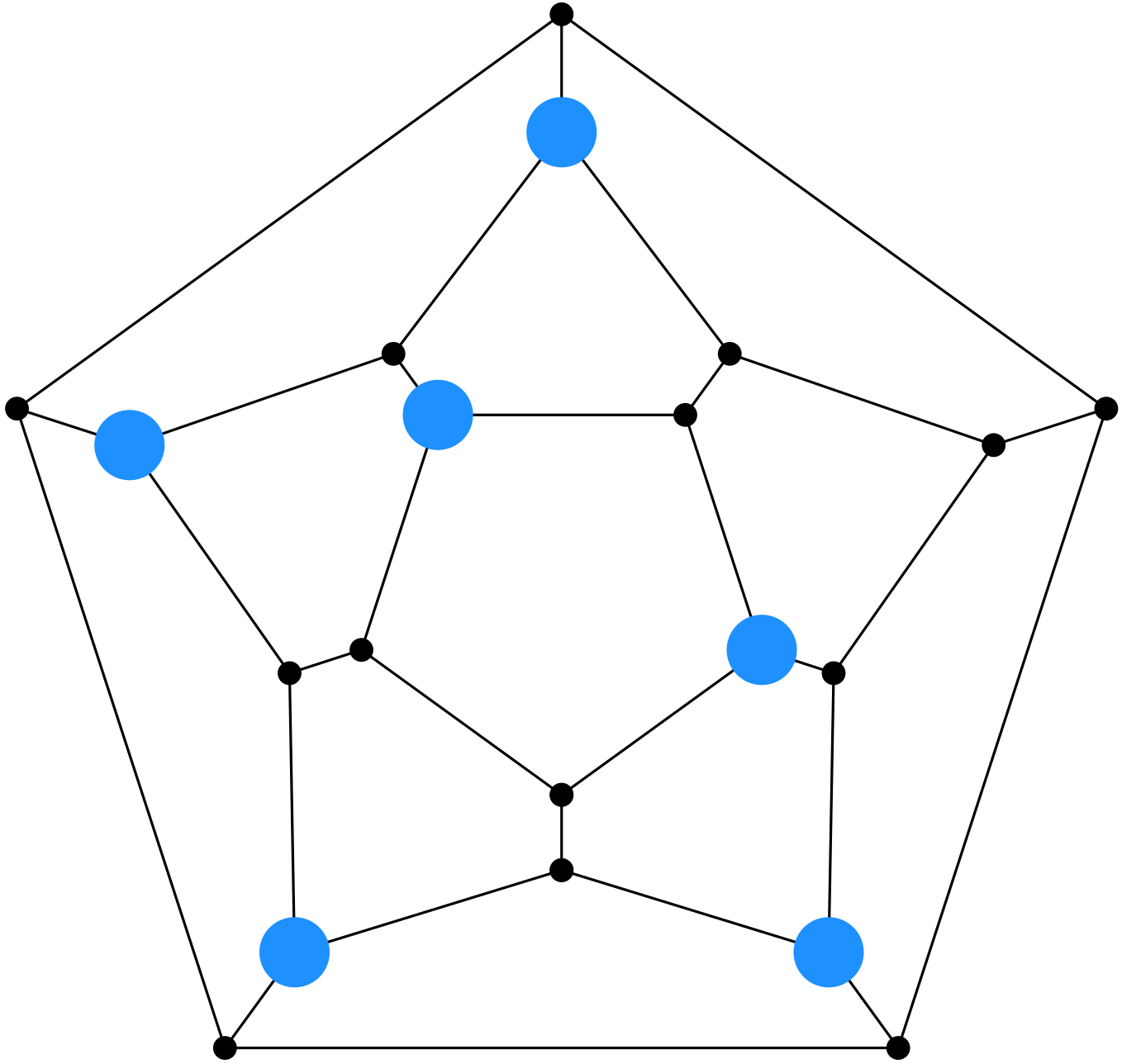
The Maximum Independent Set Problem



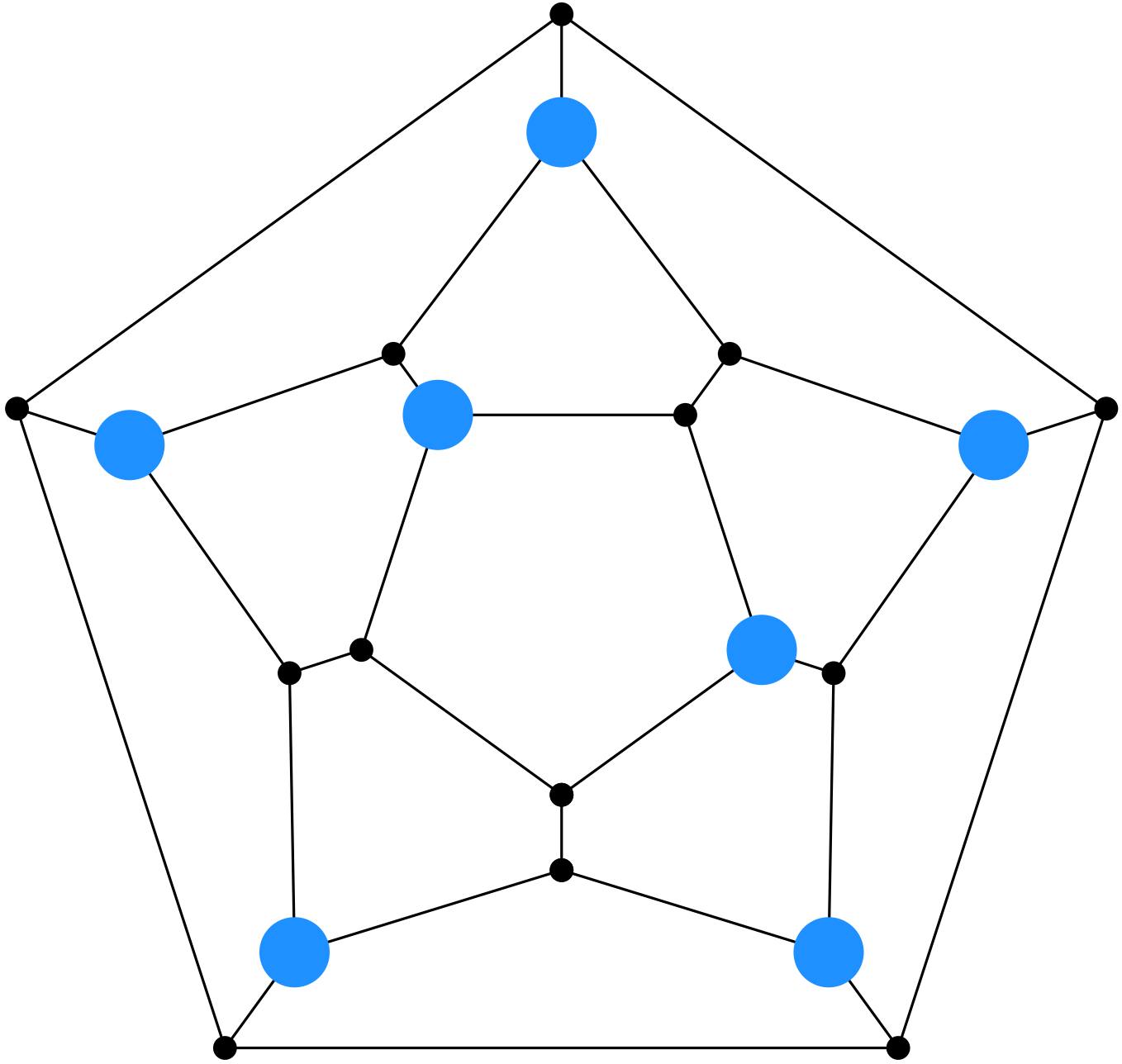
The Maximum Independent Set Problem



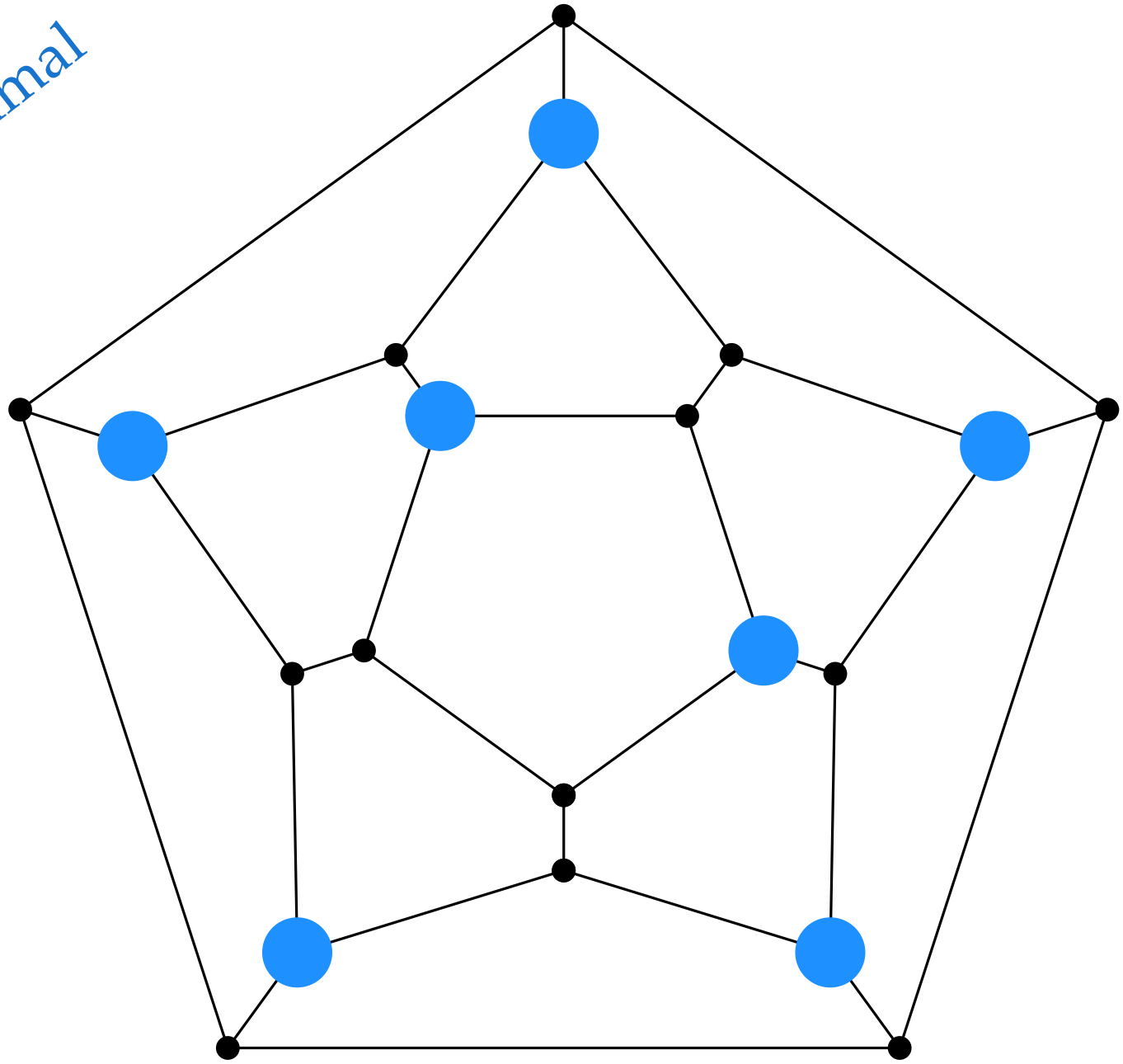
The Maximum Independent Set Problem



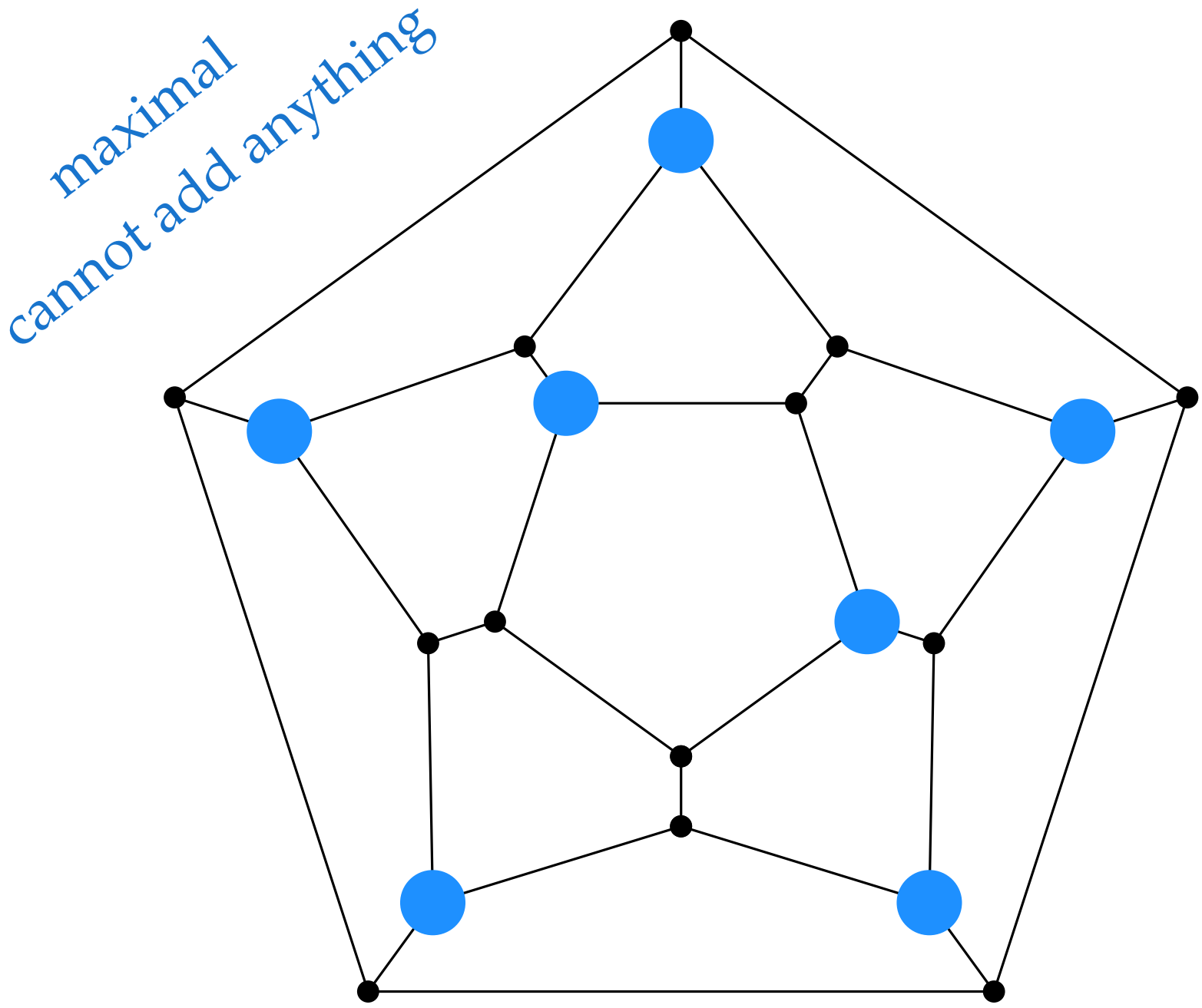
The Maximum Independent Set Problem



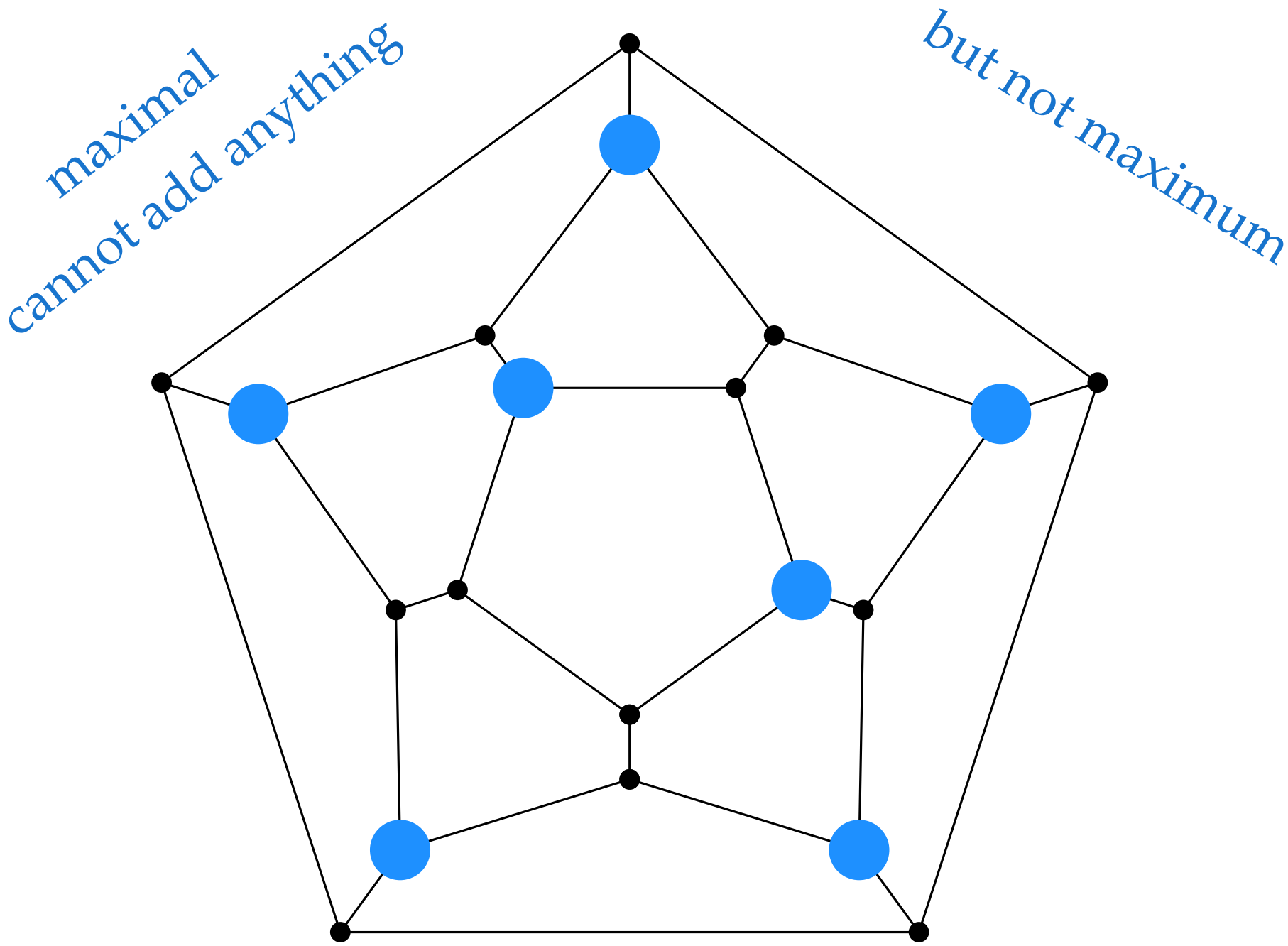
The Maximum Independent Set Problem



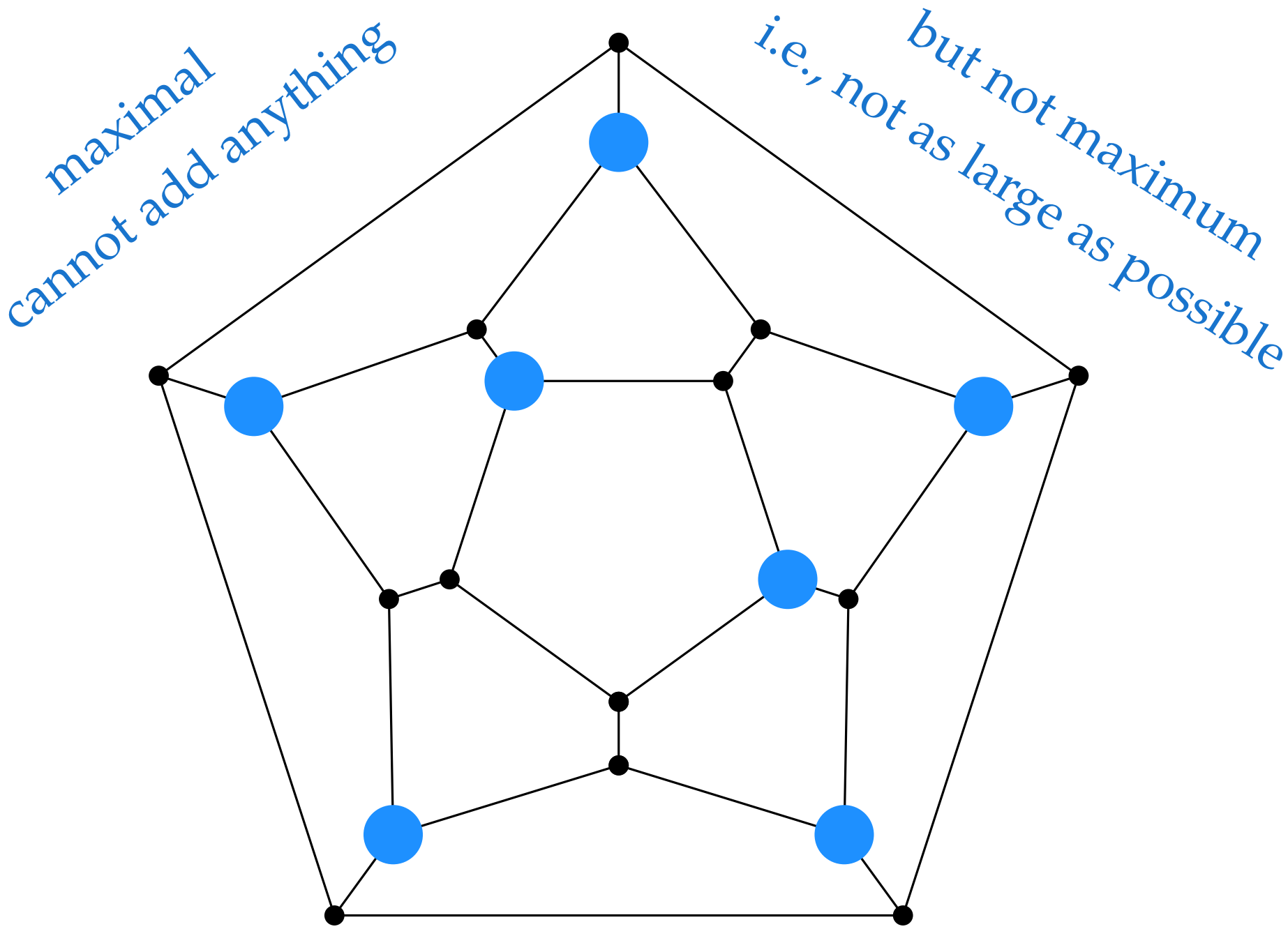
The Maximum Independent Set Problem



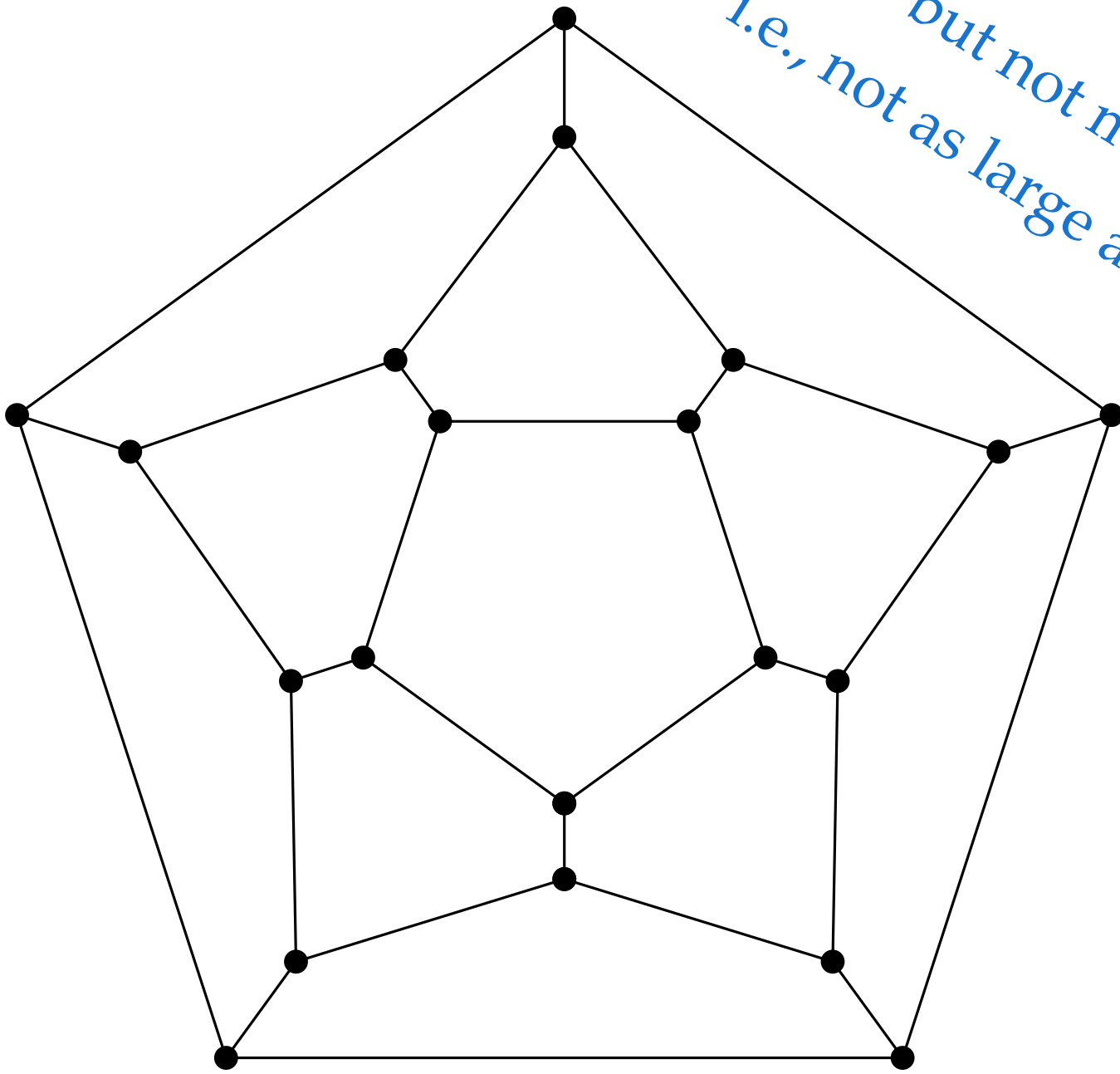
The Maximum Independent Set Problem



The Maximum Independent Set Problem



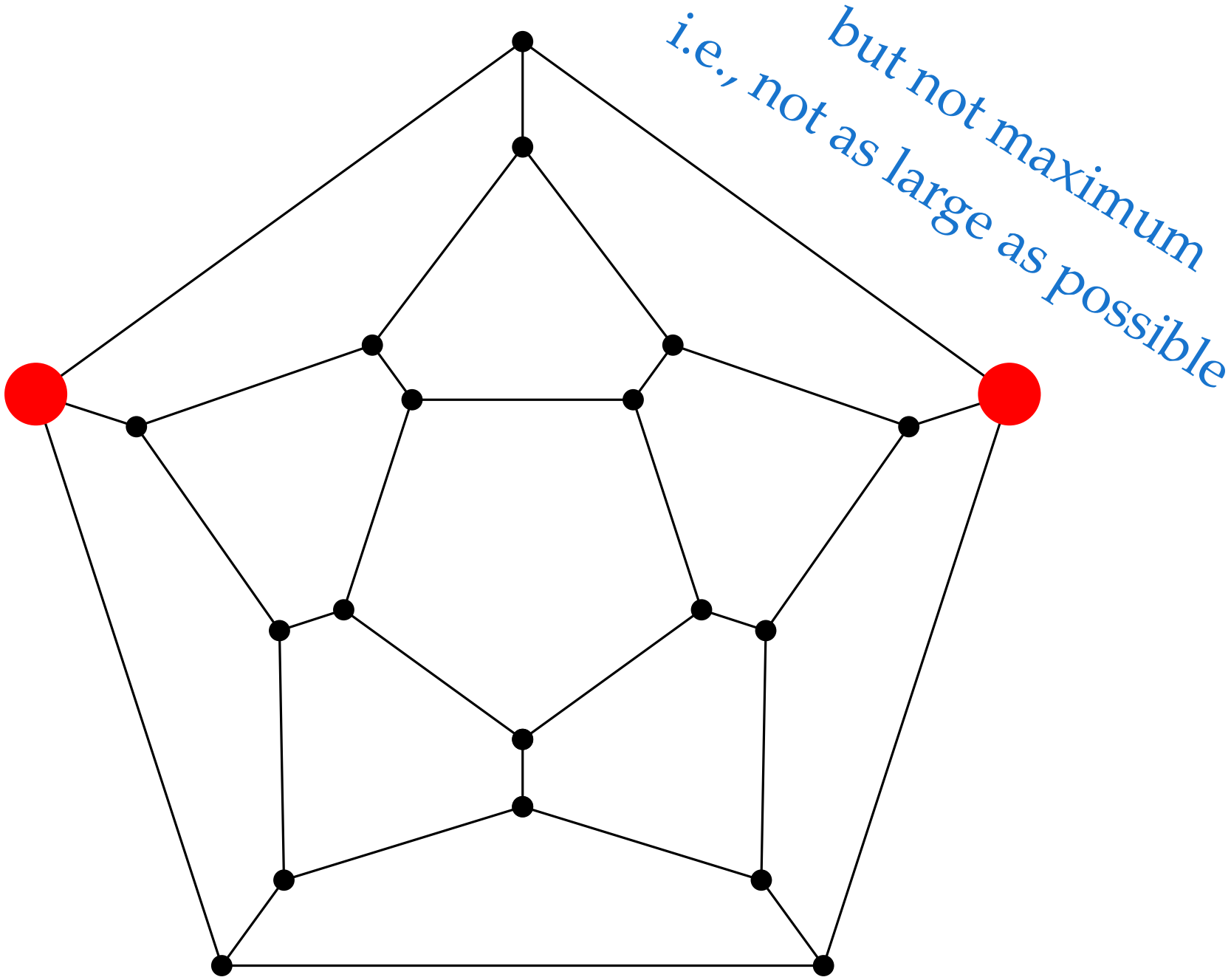
The Maximum Independent Set Problem



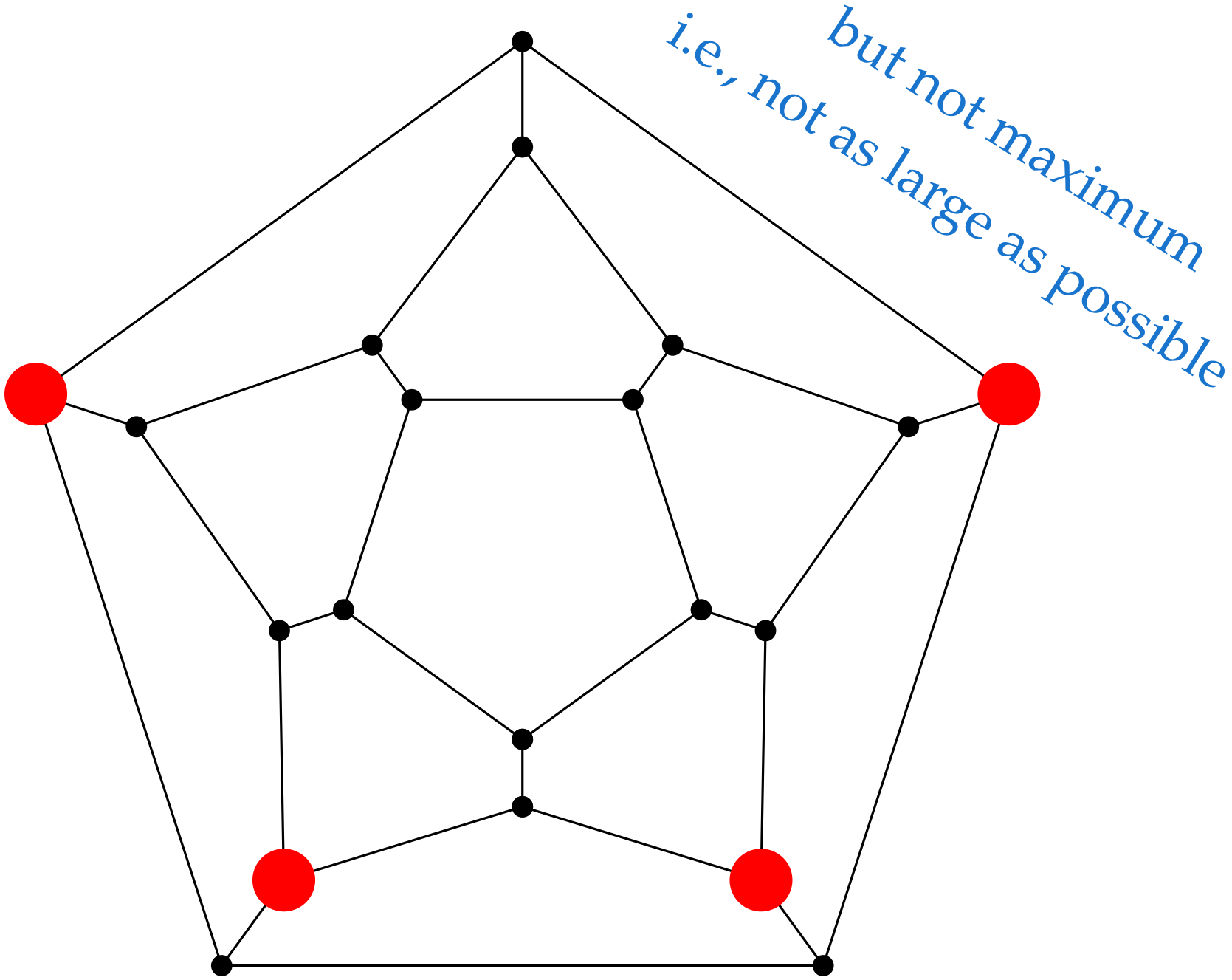
but not maximum

i.e., not as large as possible

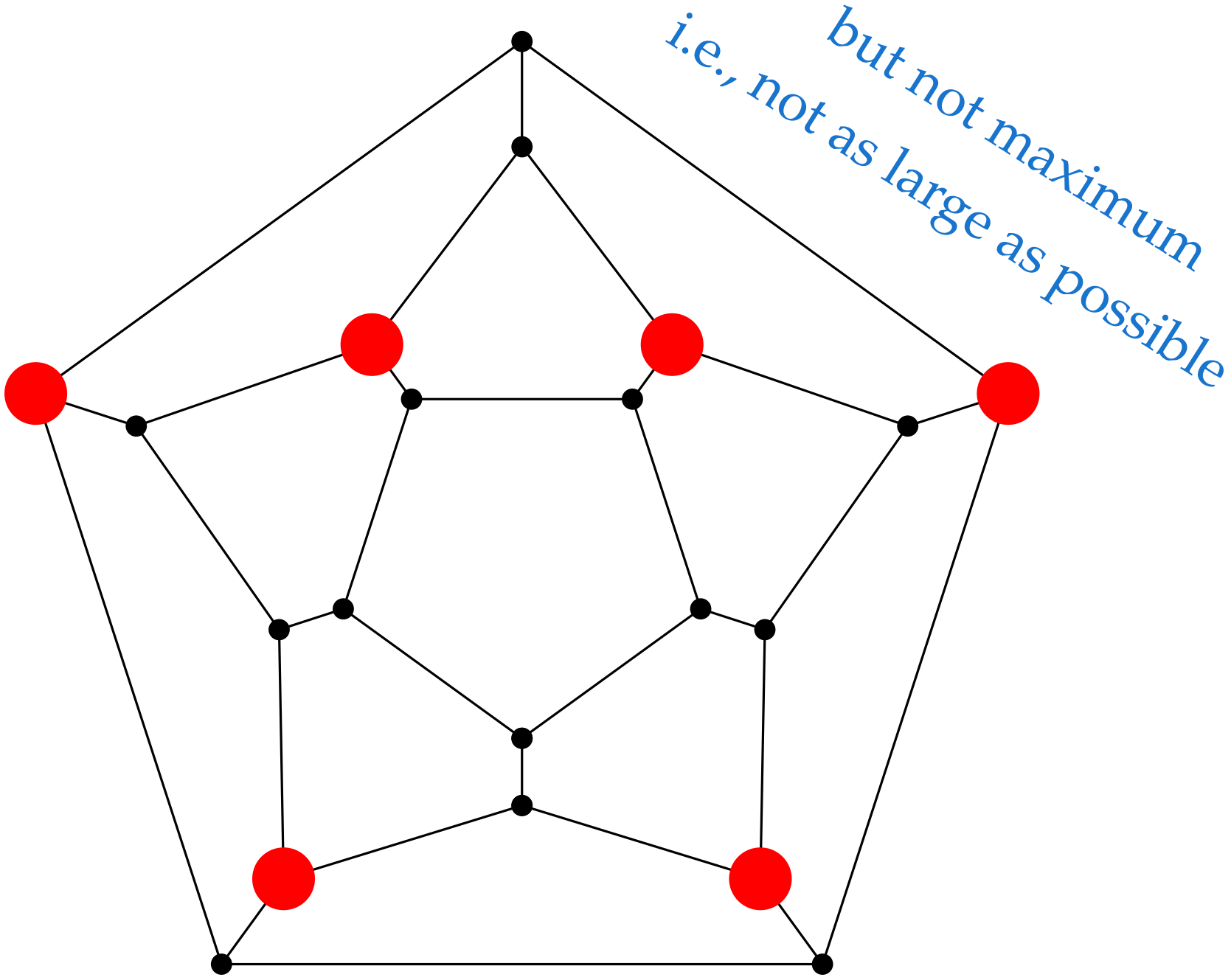
The Maximum Independent Set Problem



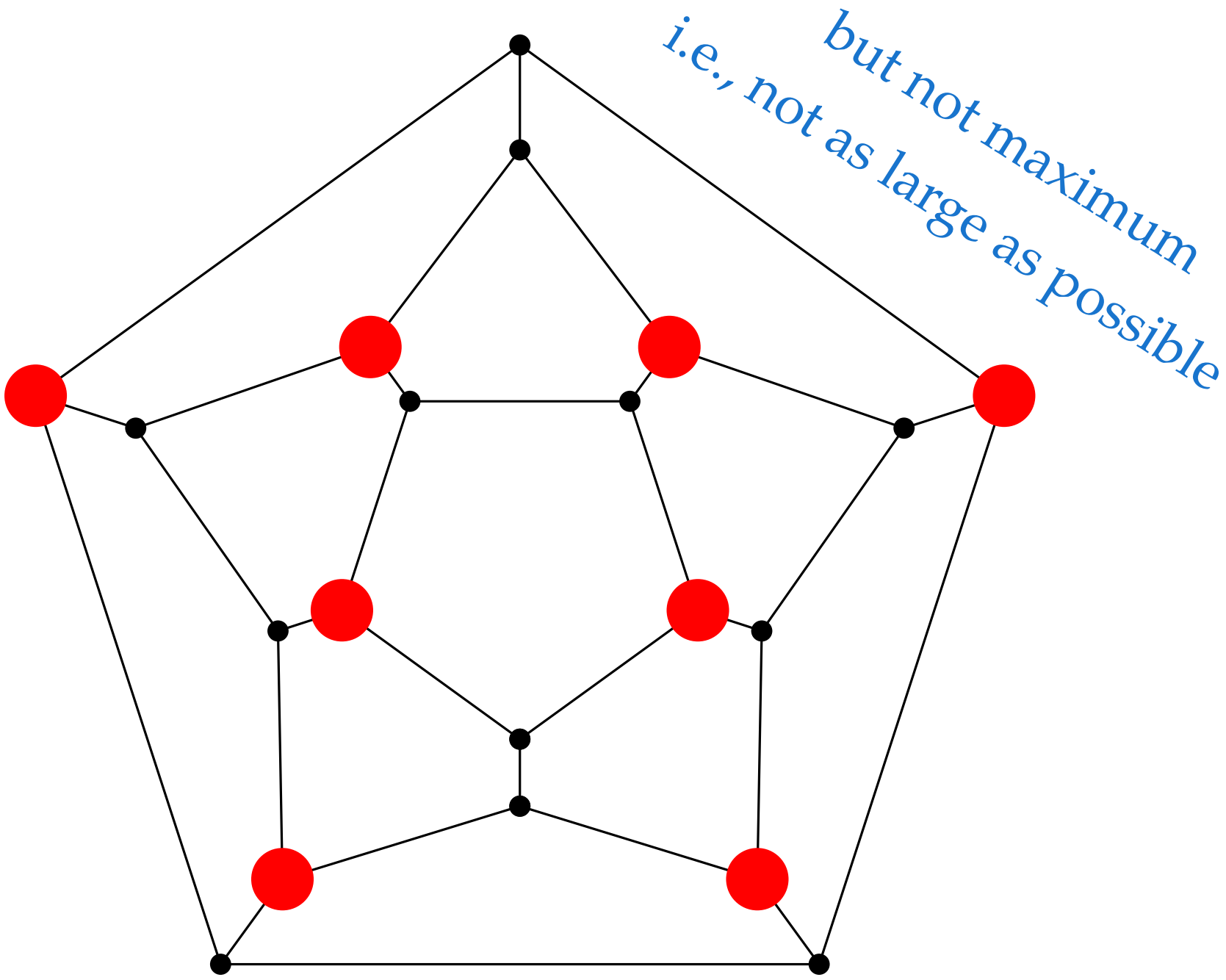
The Maximum Independent Set Problem



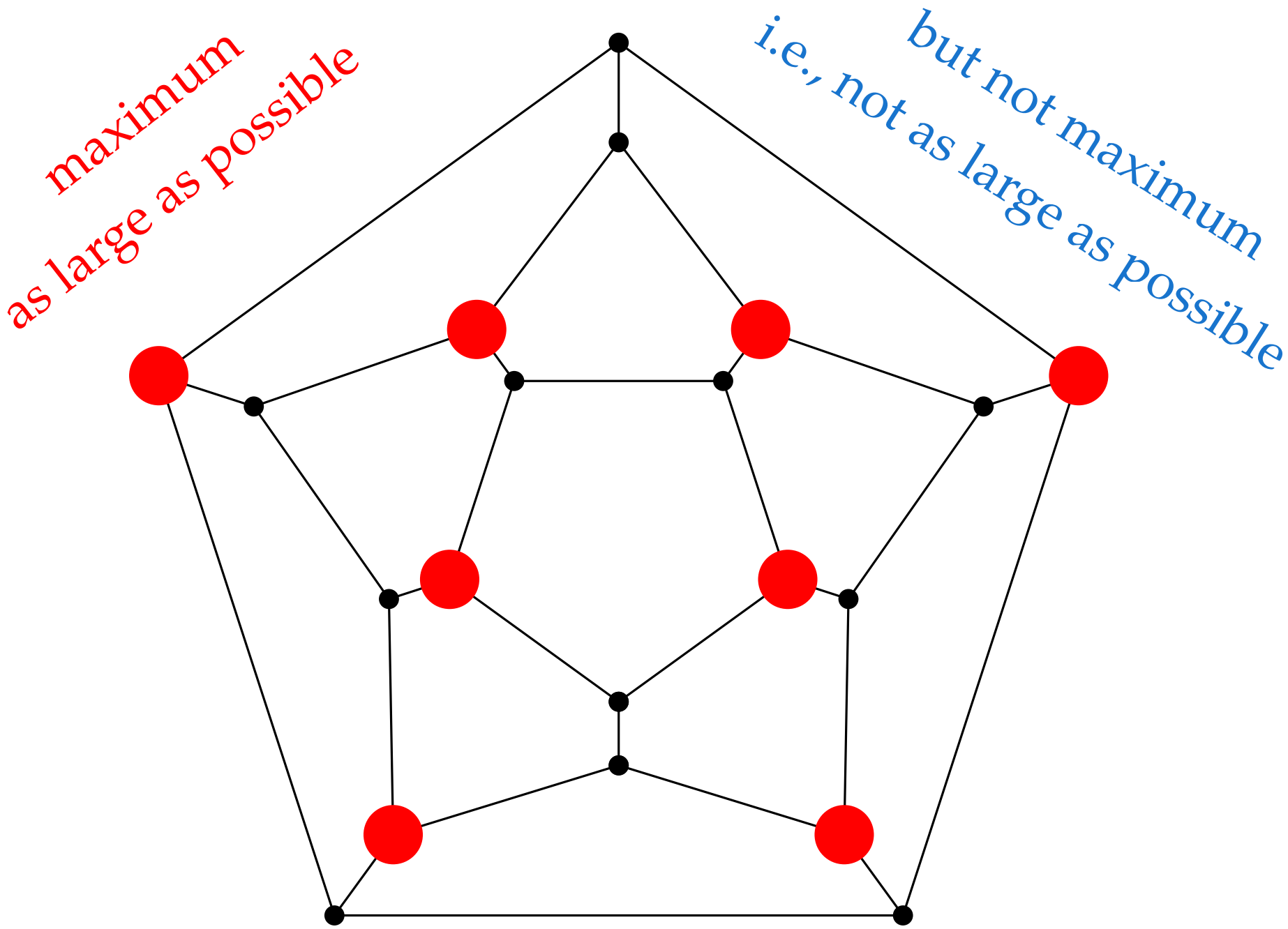
The Maximum Independent Set Problem



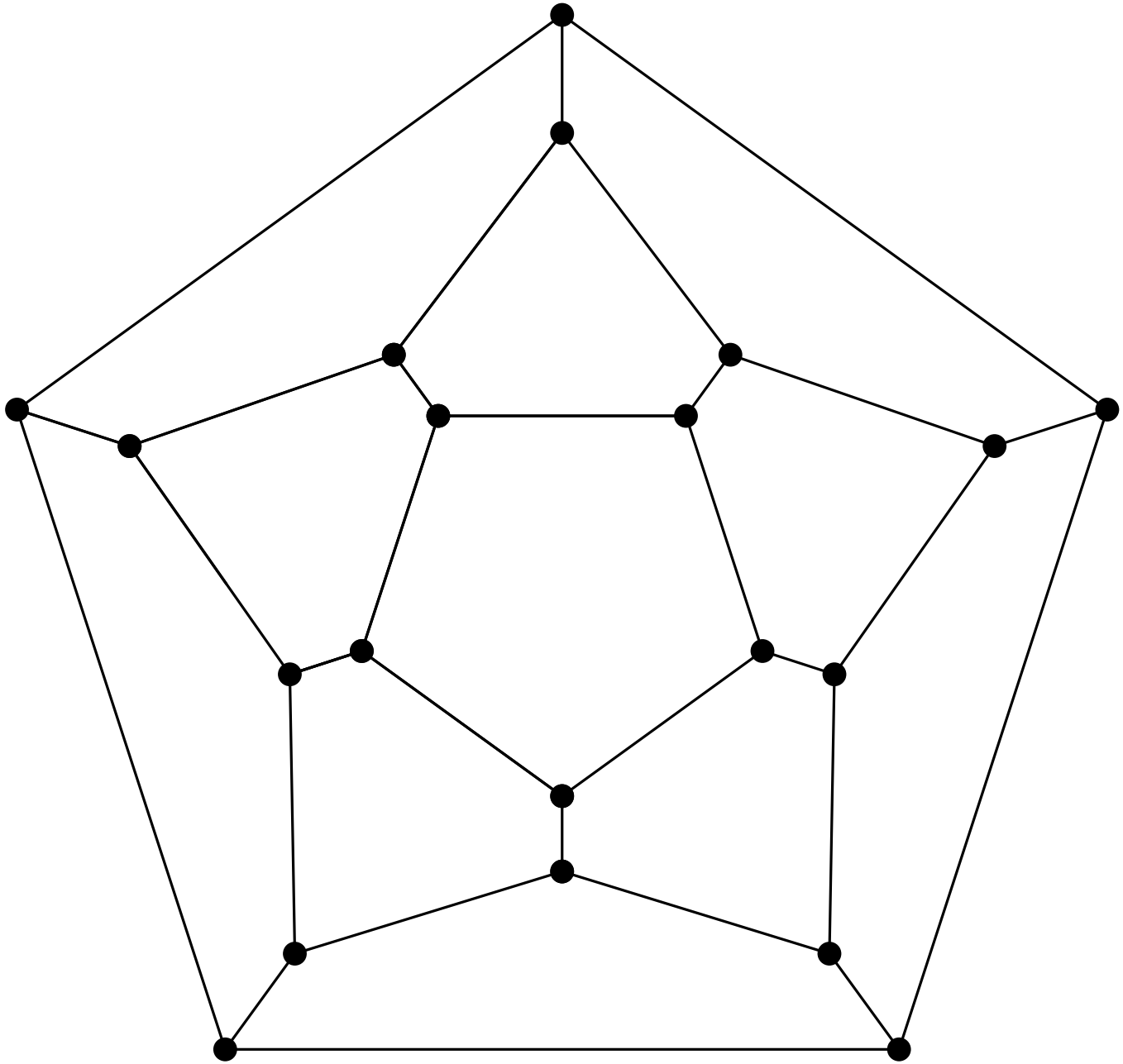
The Maximum Independent Set Problem



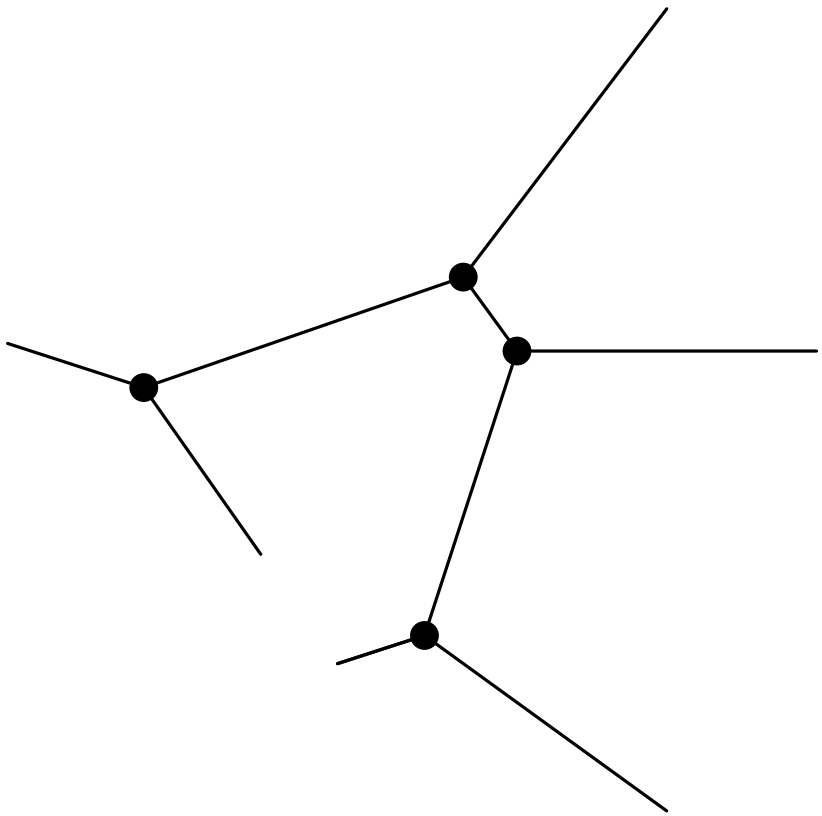
The Maximum Independent Set Problem



Using Congestion Games to Solve Max-IS

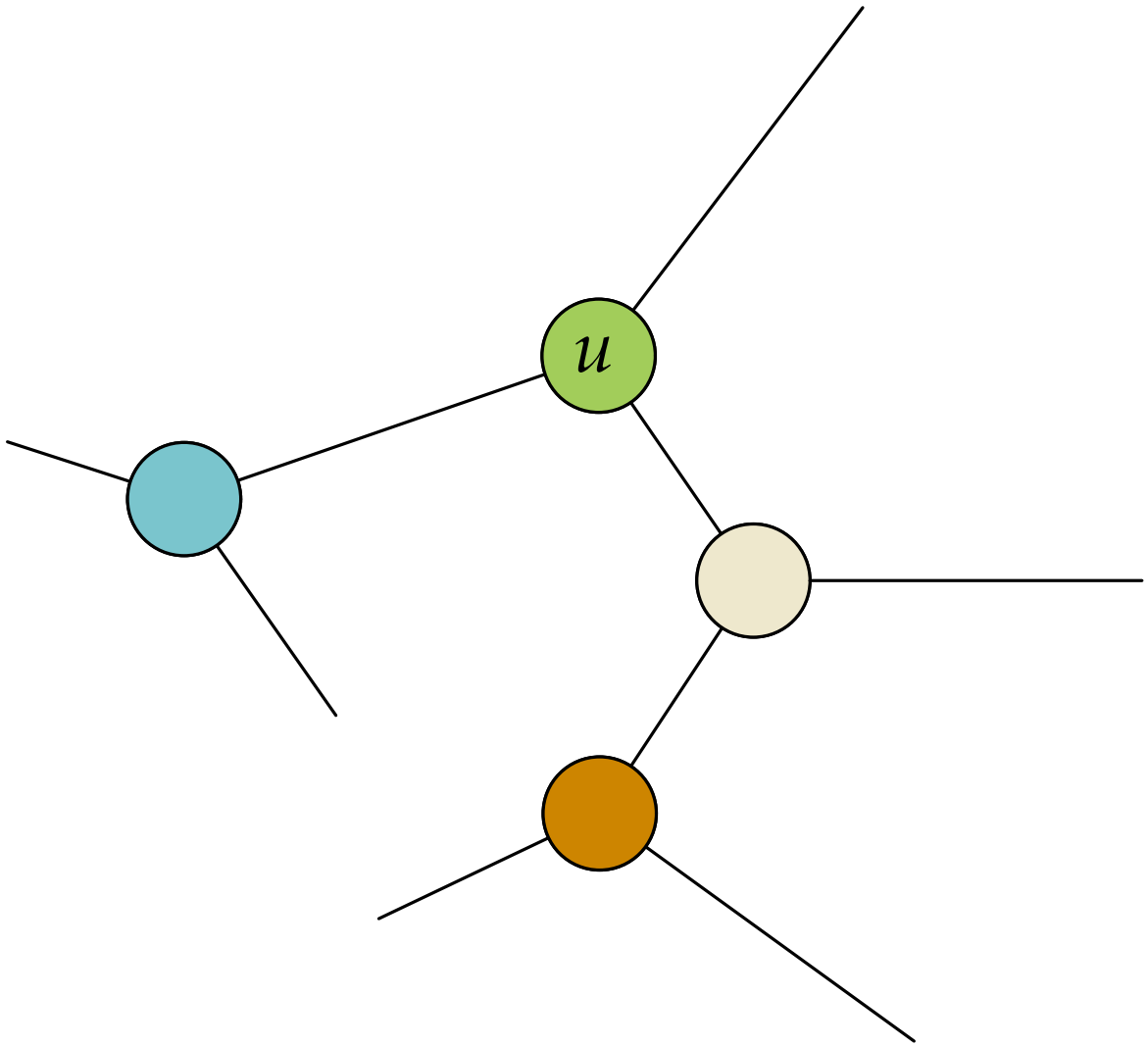


Using Congestion Games to Solve Max-IS



Using Congestion Games to Solve Max-IS

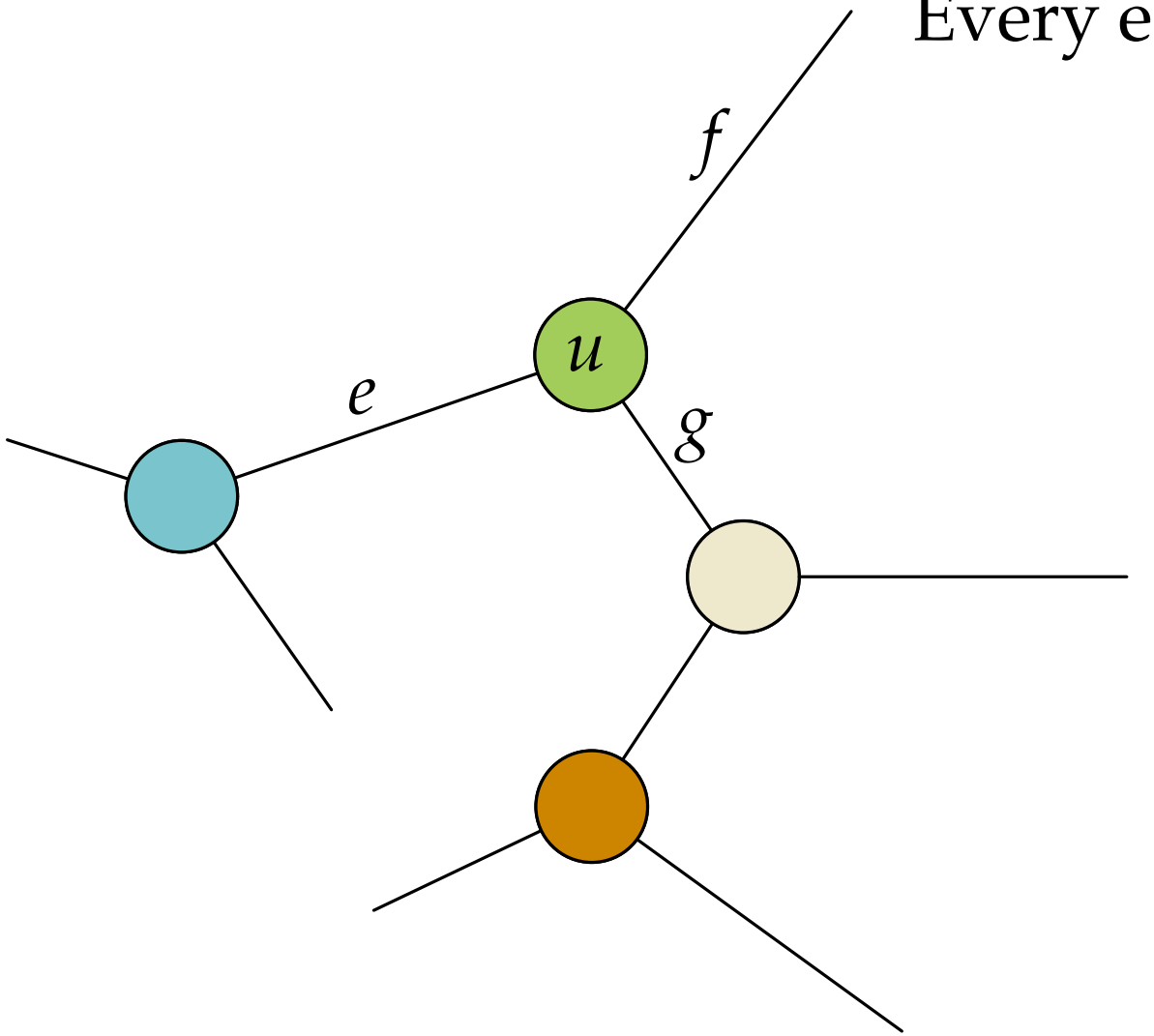
Every vertex is a player



Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

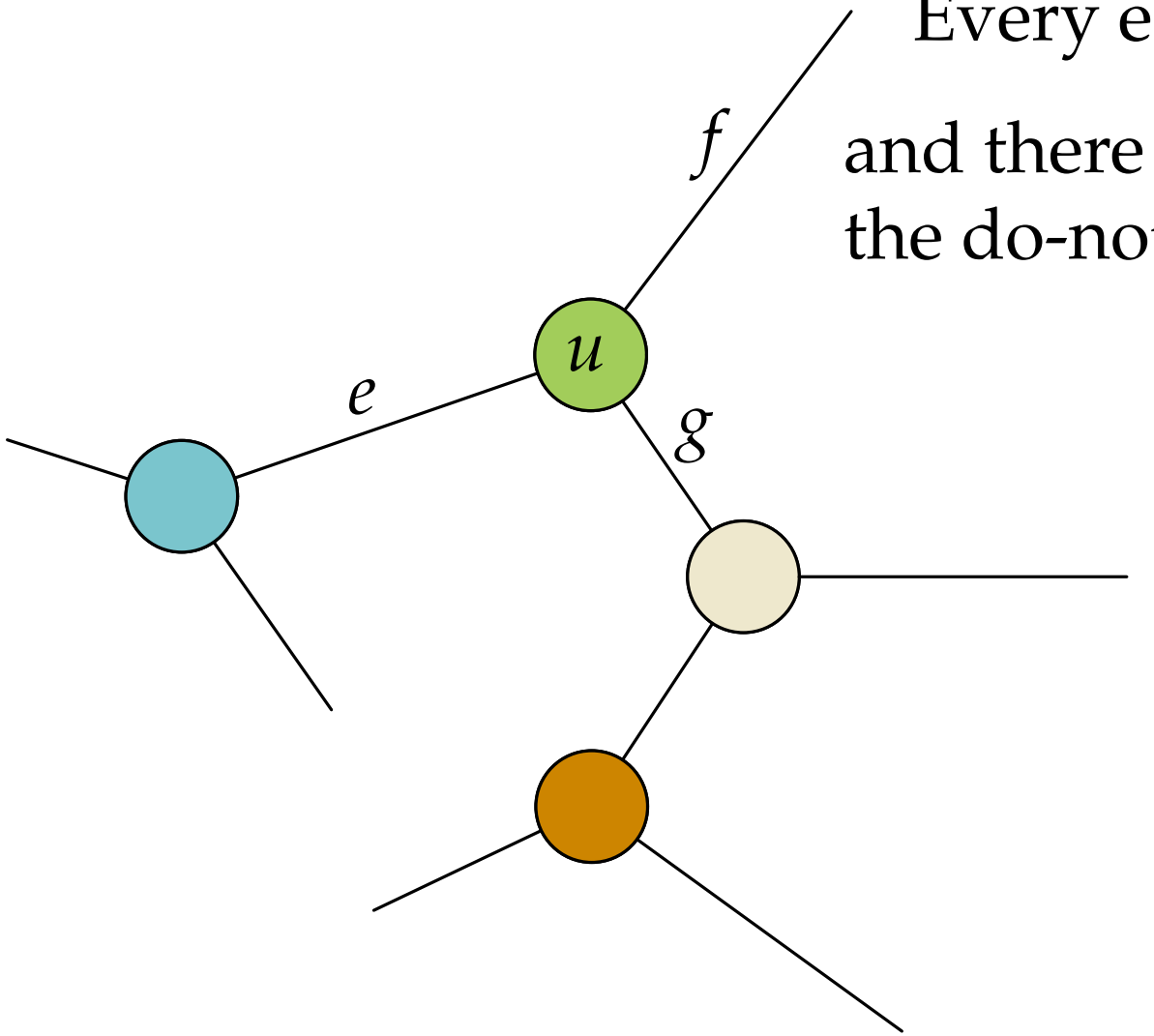


Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource, the do-nothing-box.



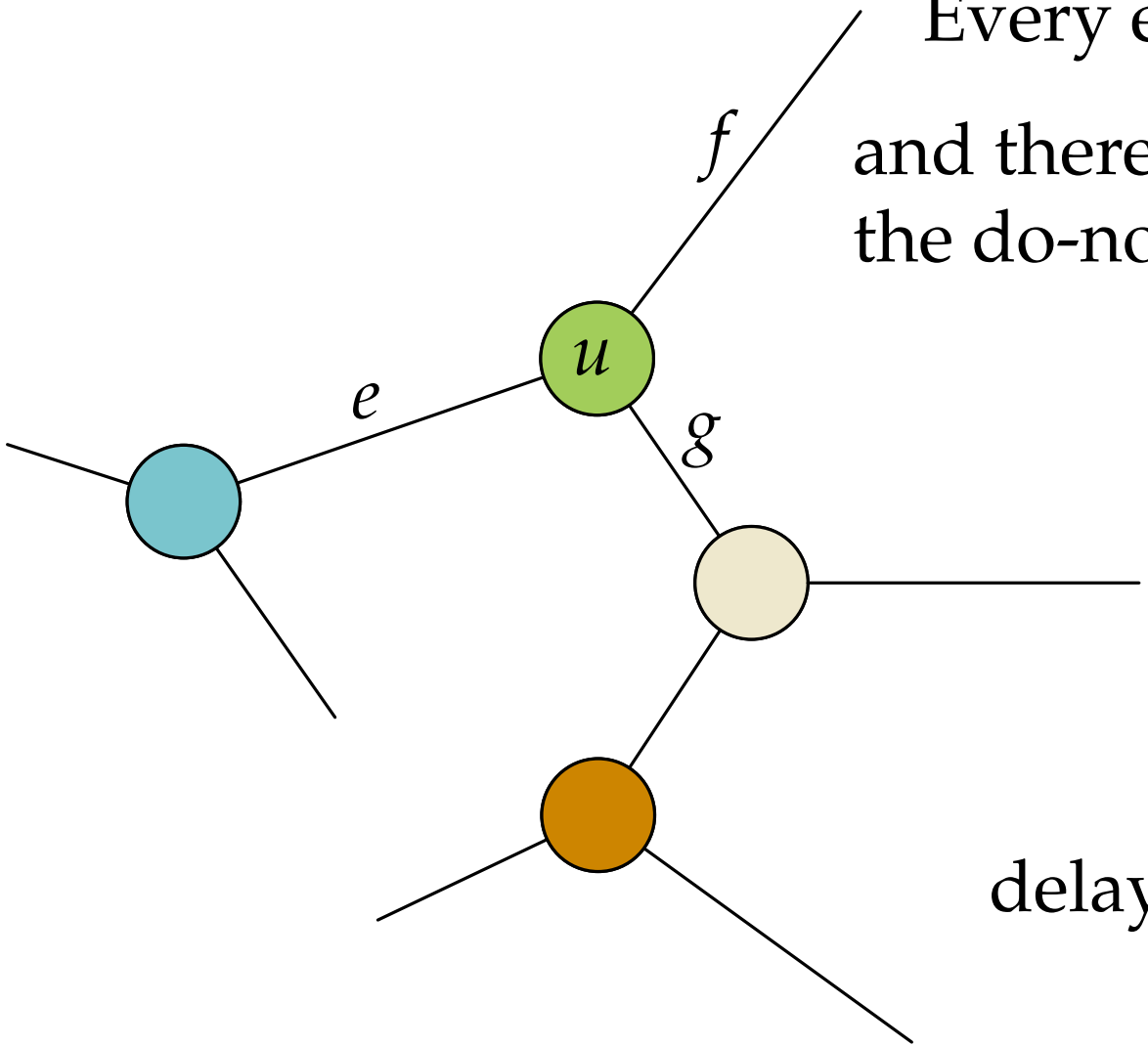
$\text{delay}(\text{DNB}, i) = 1$

Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource, the do-nothing-box.



$$\text{delay}(\text{DNB}, i) = 1$$

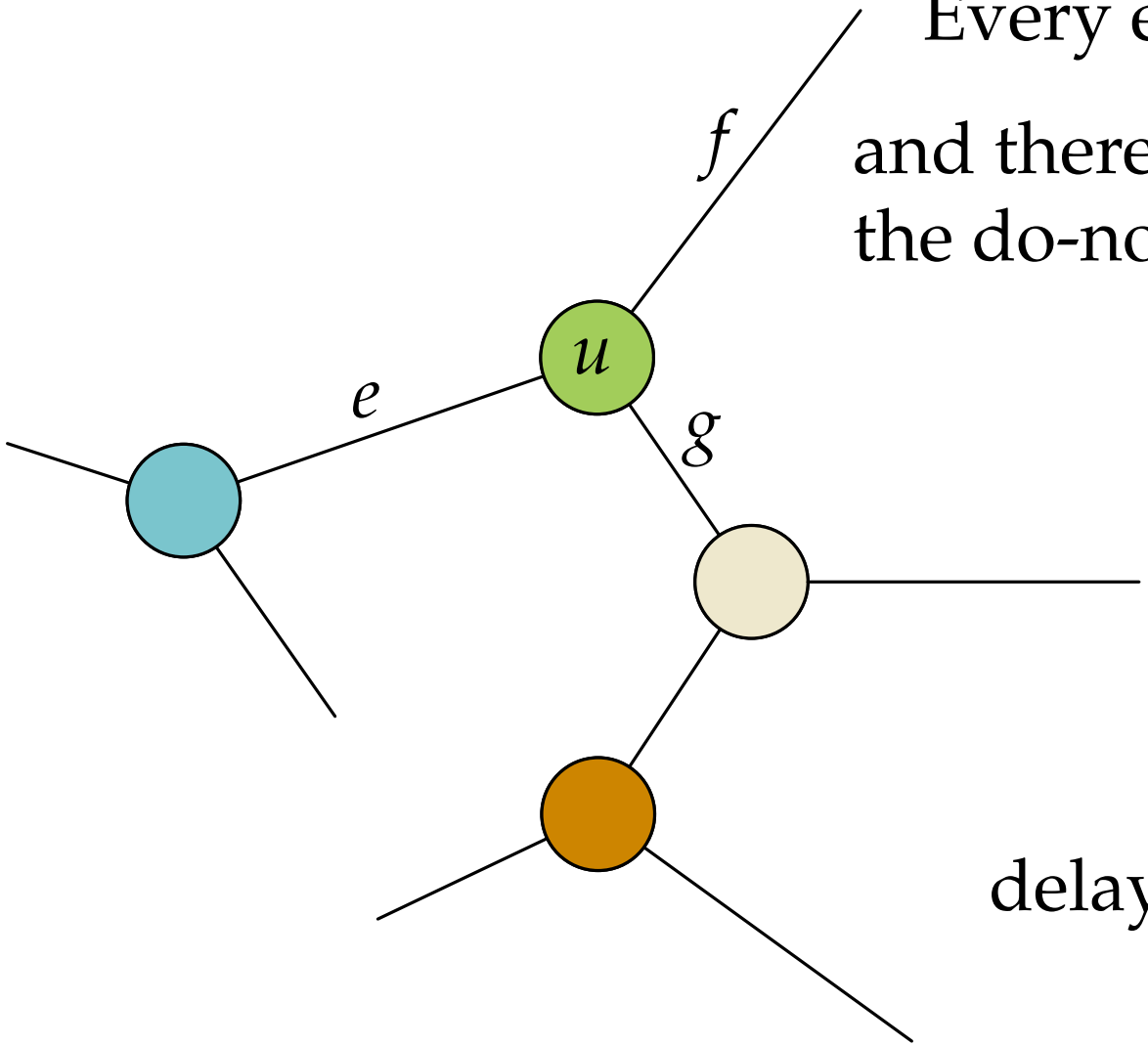
$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource, the do-nothing-box.



$\text{delay}(\text{DNB}, i) = 1$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

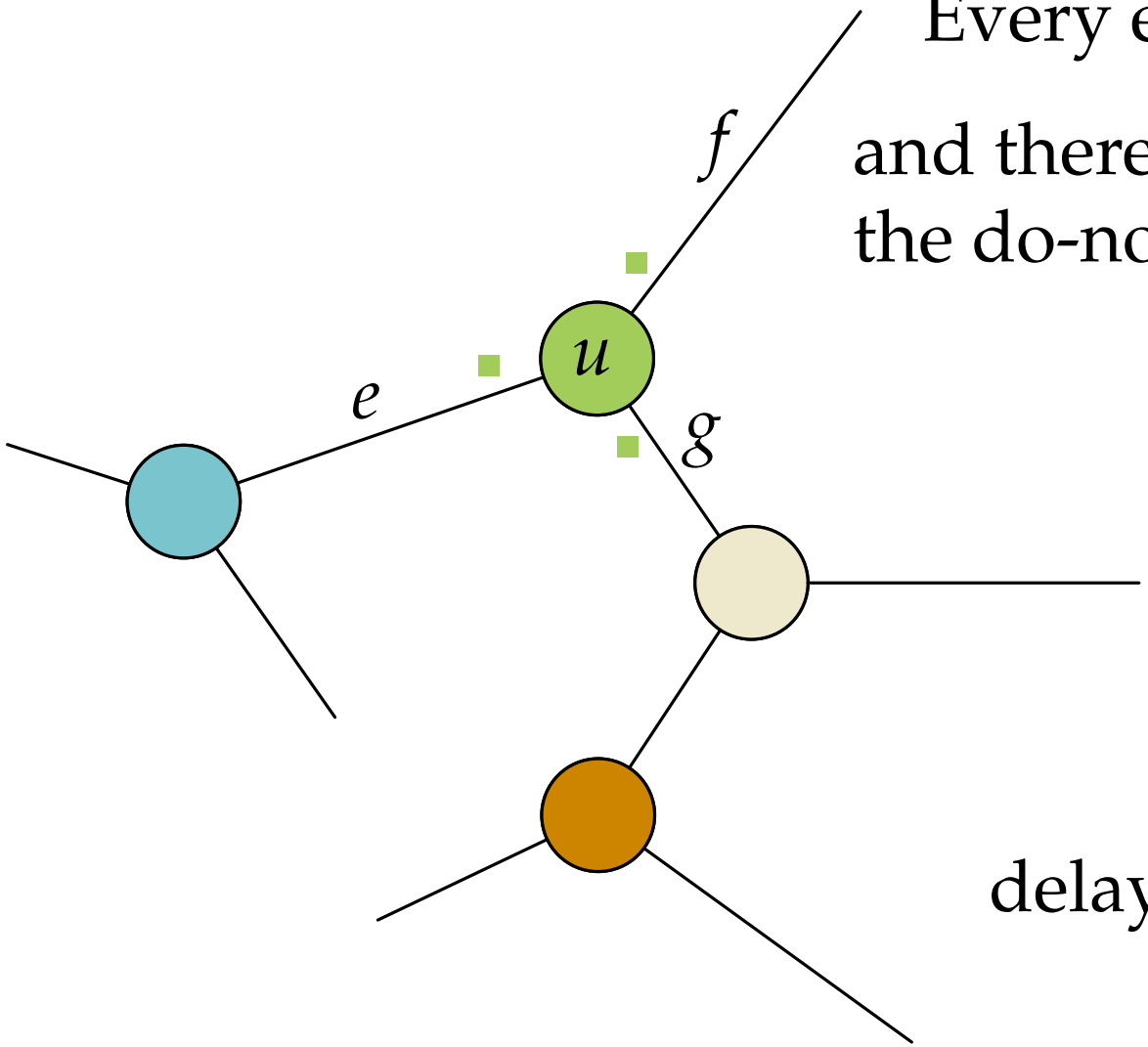
$S_u =$

Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource,
the do-nothing-box.



$\text{delay}(\text{DNB}, i) = 1$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

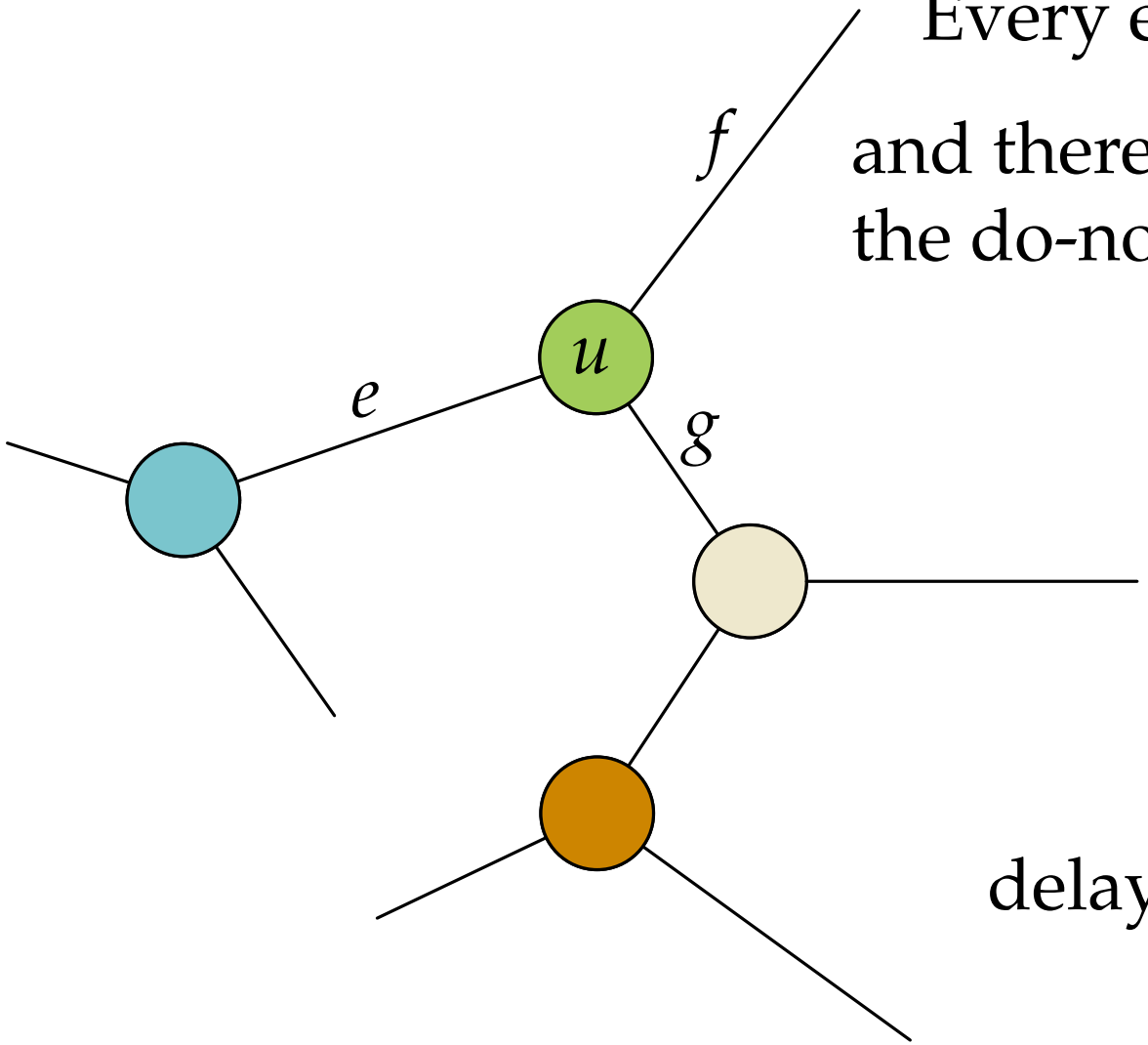
$S_u = \{\{e, f, g\} \qquad \}$

Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource, the do-nothing-box.



$\text{delay}(\text{DNB}, i) = 1$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

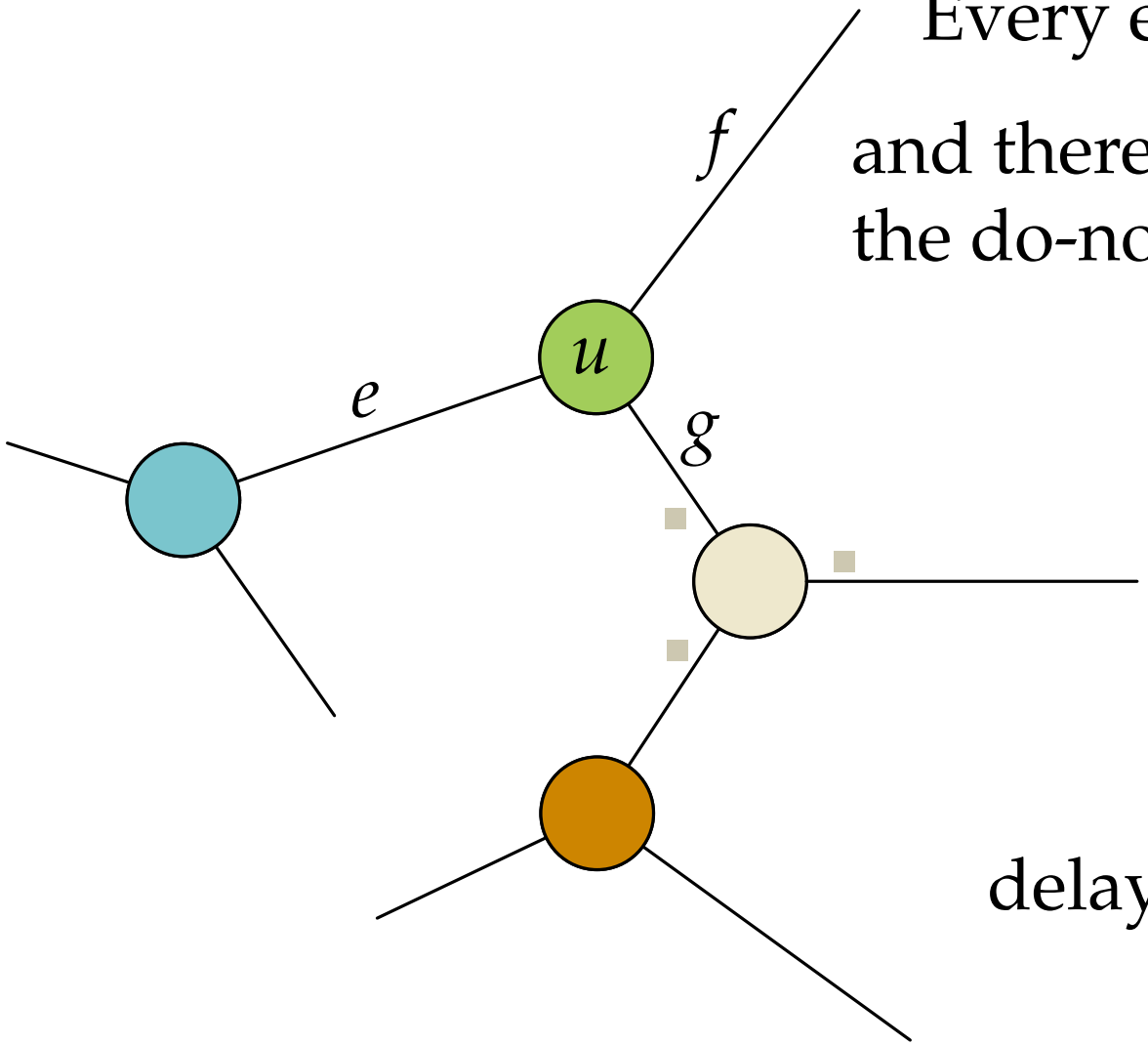
$S_u = \{\{e, f, g\}, \{\text{DNB}\}\}$

Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource, the do-nothing-box.



$\text{delay}(\text{DNB}, i) = 1$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

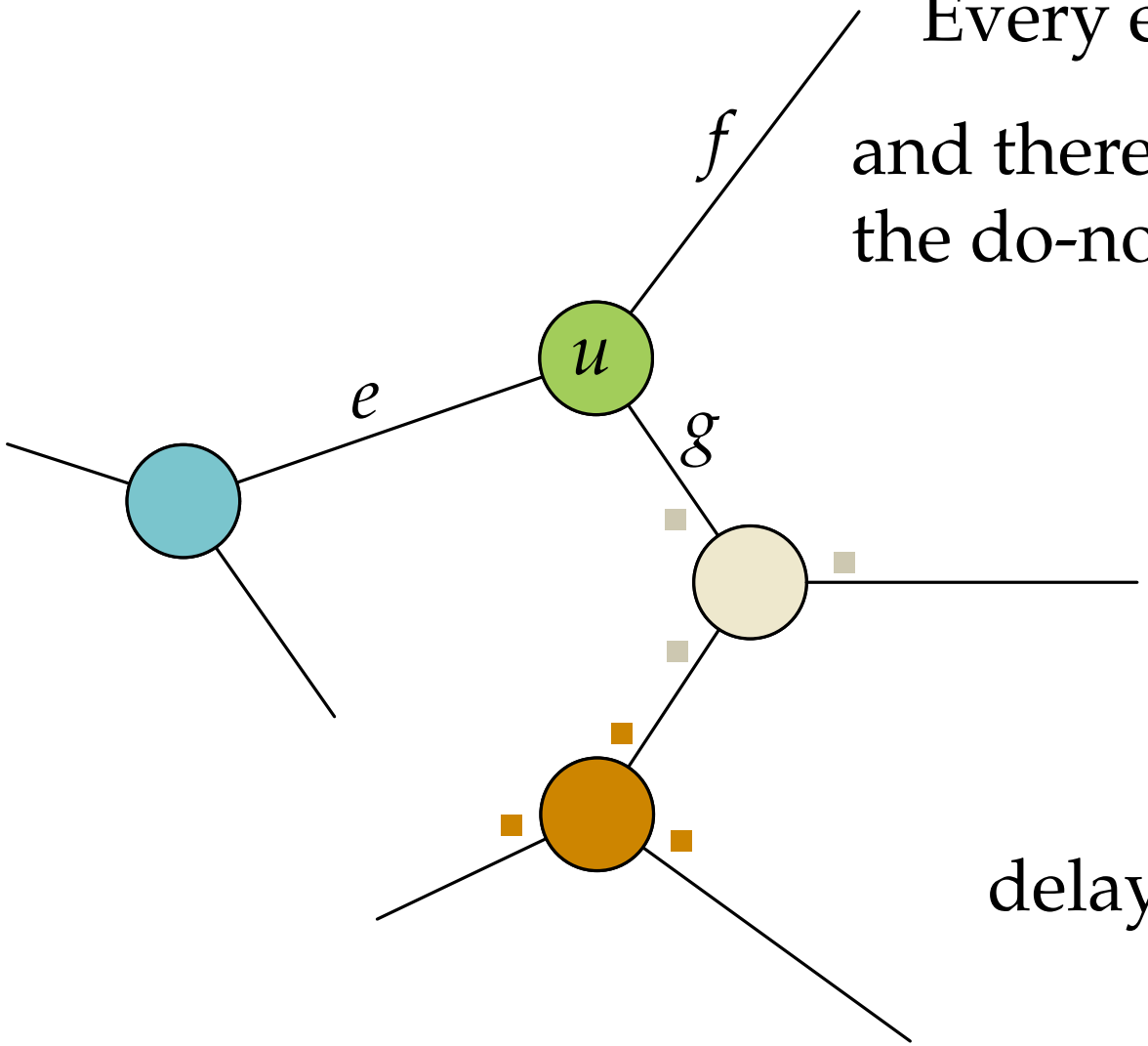
$S_u = \{\{e, f, g\}, \{\text{DNB}\}\}$

Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource, the do-nothing-box.



$\text{delay}(\text{DNB}, i) = 1$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

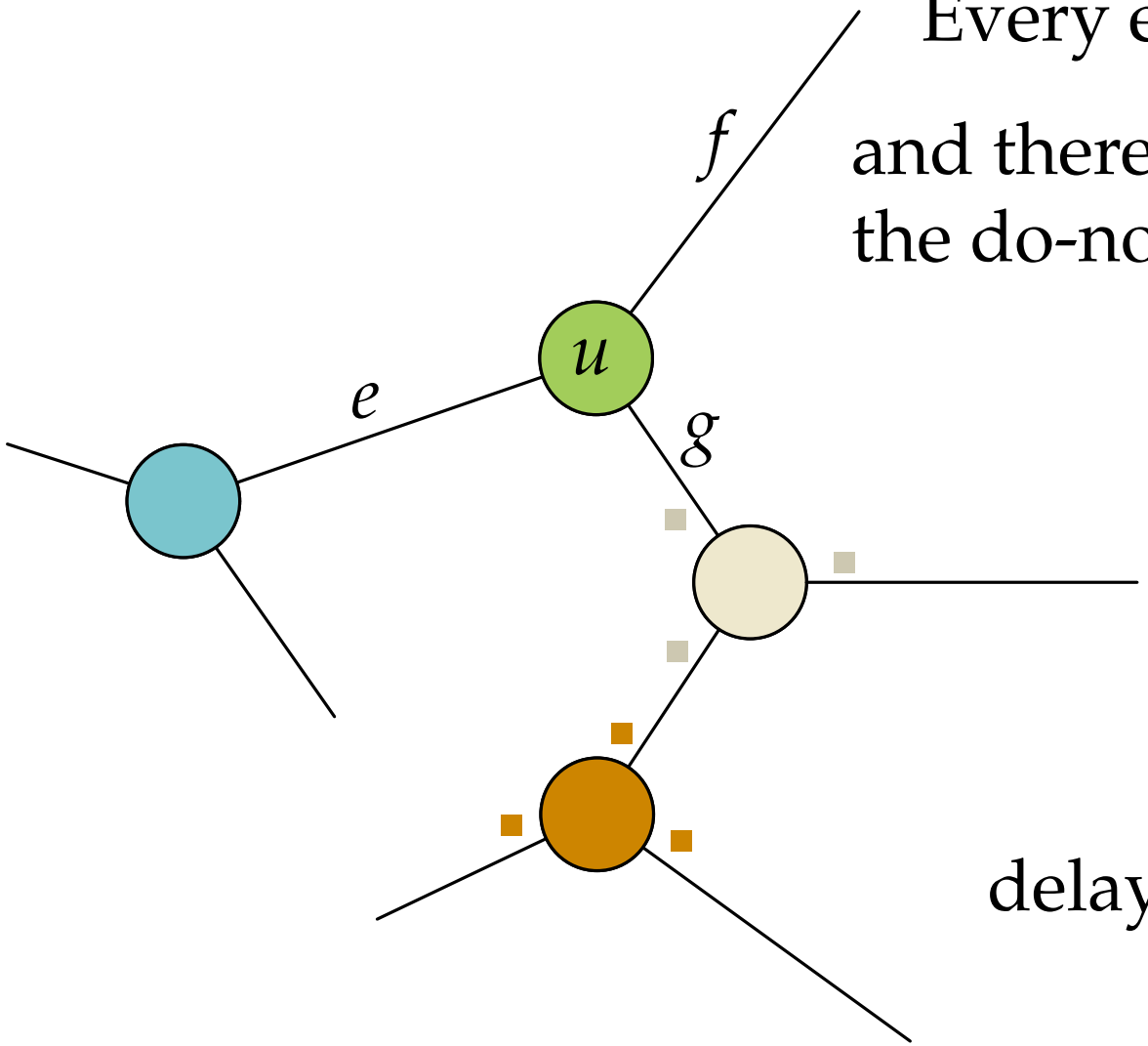
$S_u = \{\{e, f, g\}, \{\text{DNB}\}\}$

Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource, the do-nothing-box.



$\text{delay}(\text{DNB}, i) = 1$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

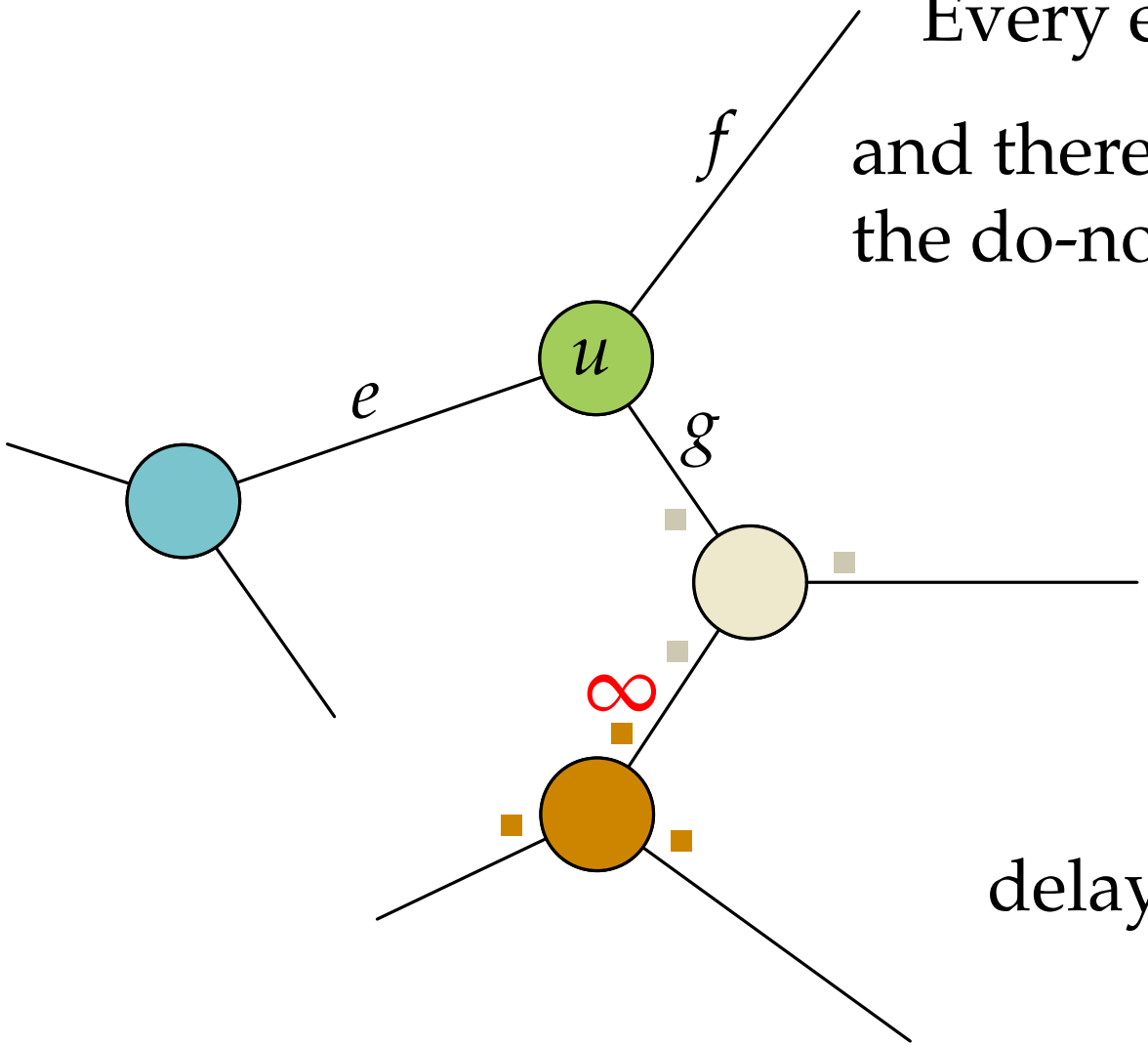
$S_u = \{\{e, f, g\}, \{\text{DNB}\}\}$

Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource, the do-nothing-box.



$\text{delay}(\text{DNB}, i) = 1$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

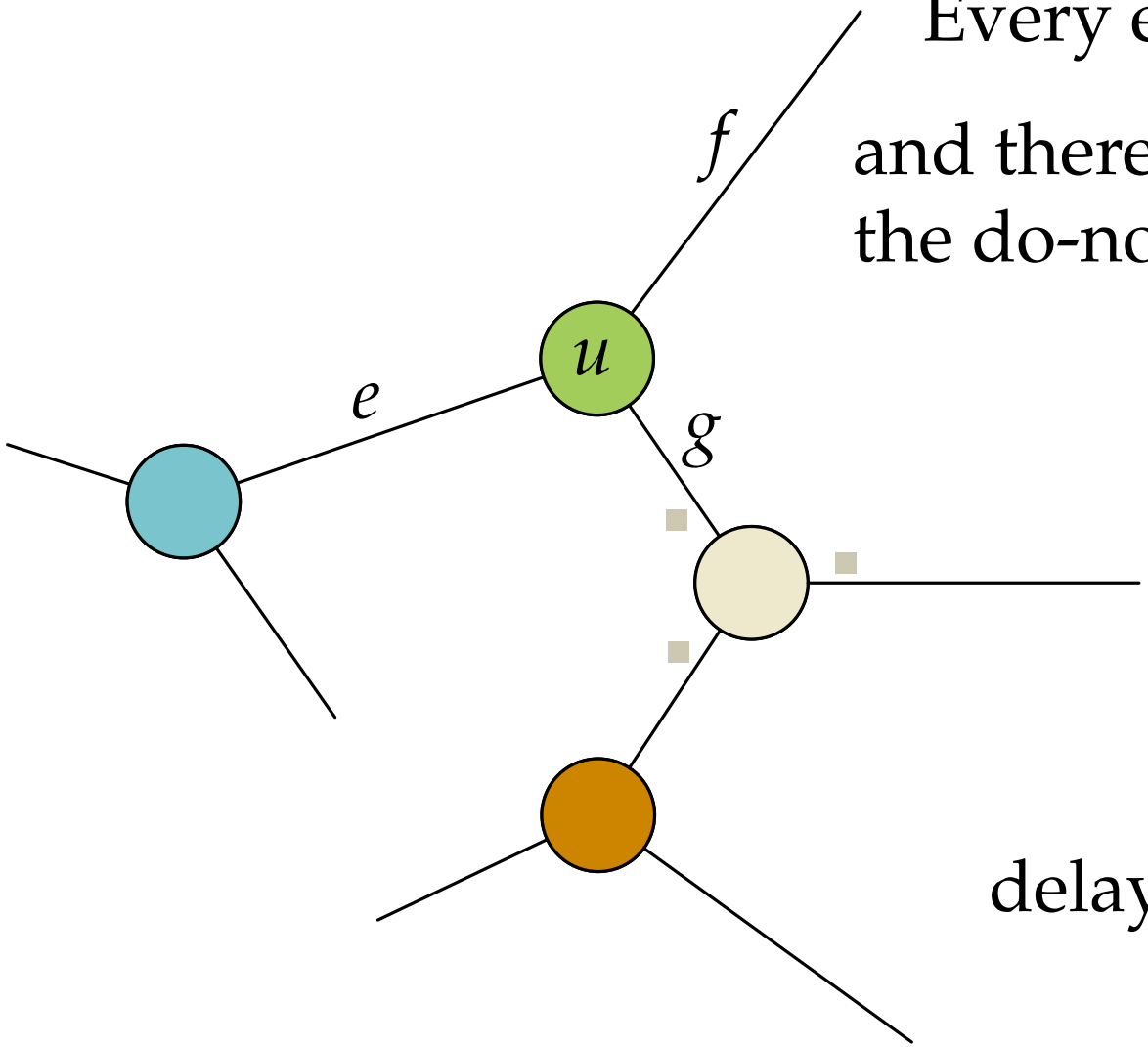
$S_u = \{\{e, f, g\}, \{\text{DNB}\}\}$

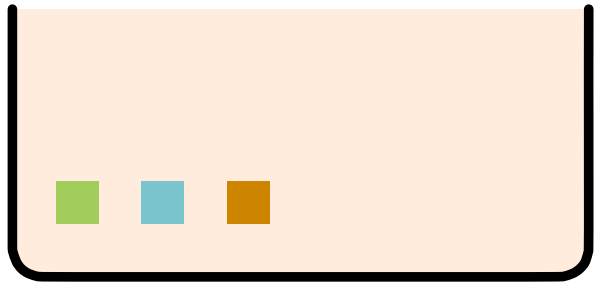
Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource, the do-nothing-box.





$\text{delay}(\text{DNB}, i) = 1$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

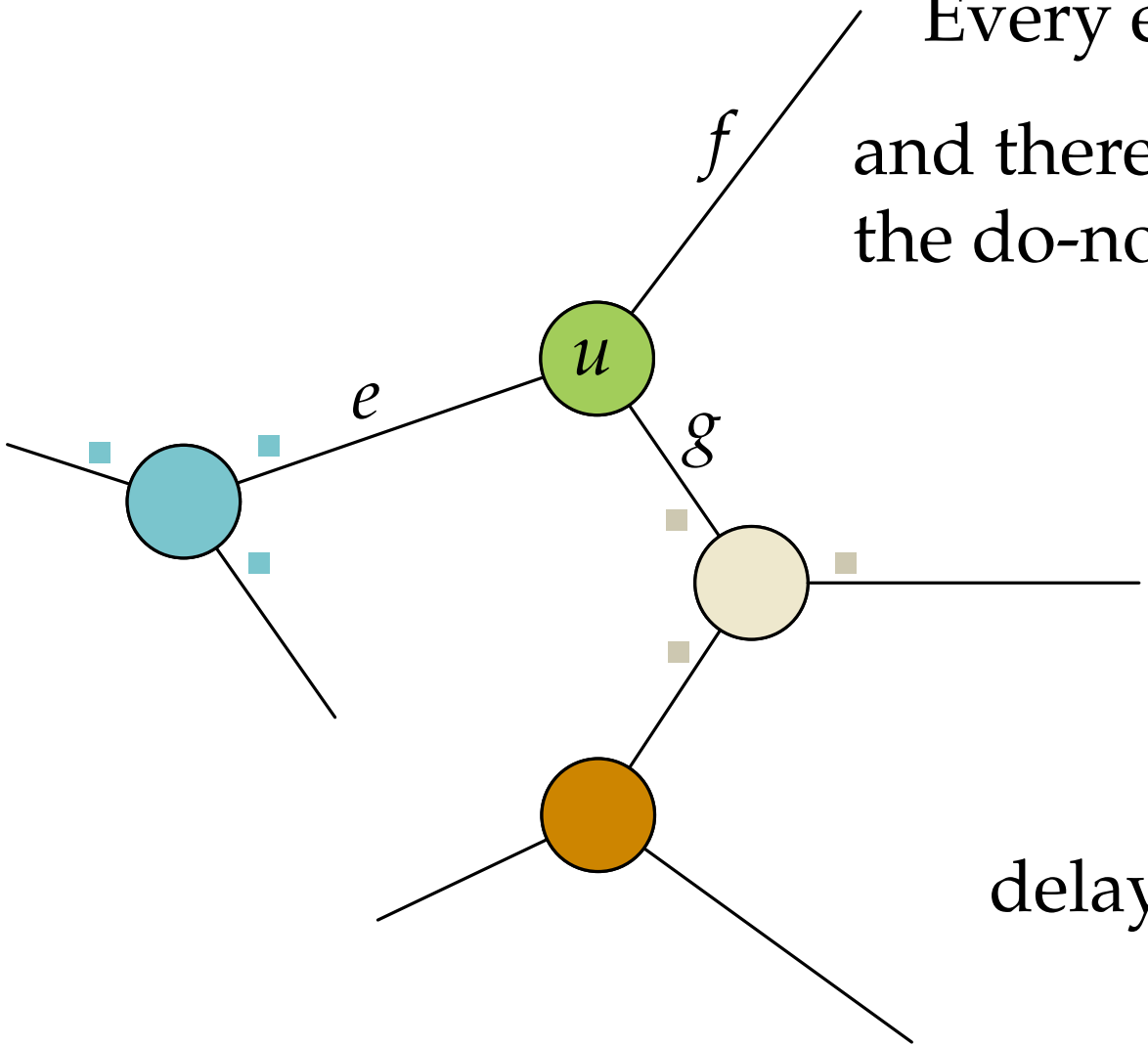
$S_u = \{\{e, f, g\}, \{\text{DNB}\}\}$

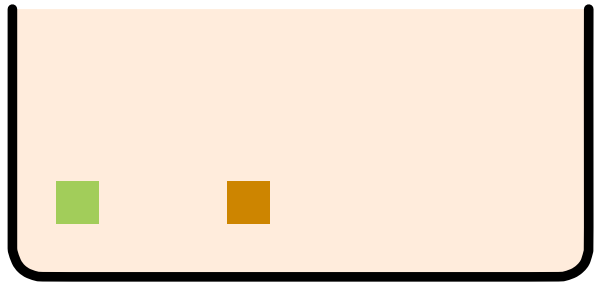
Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource, the do-nothing-box.





$\text{delay}(\text{DNB}, i) = 1$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

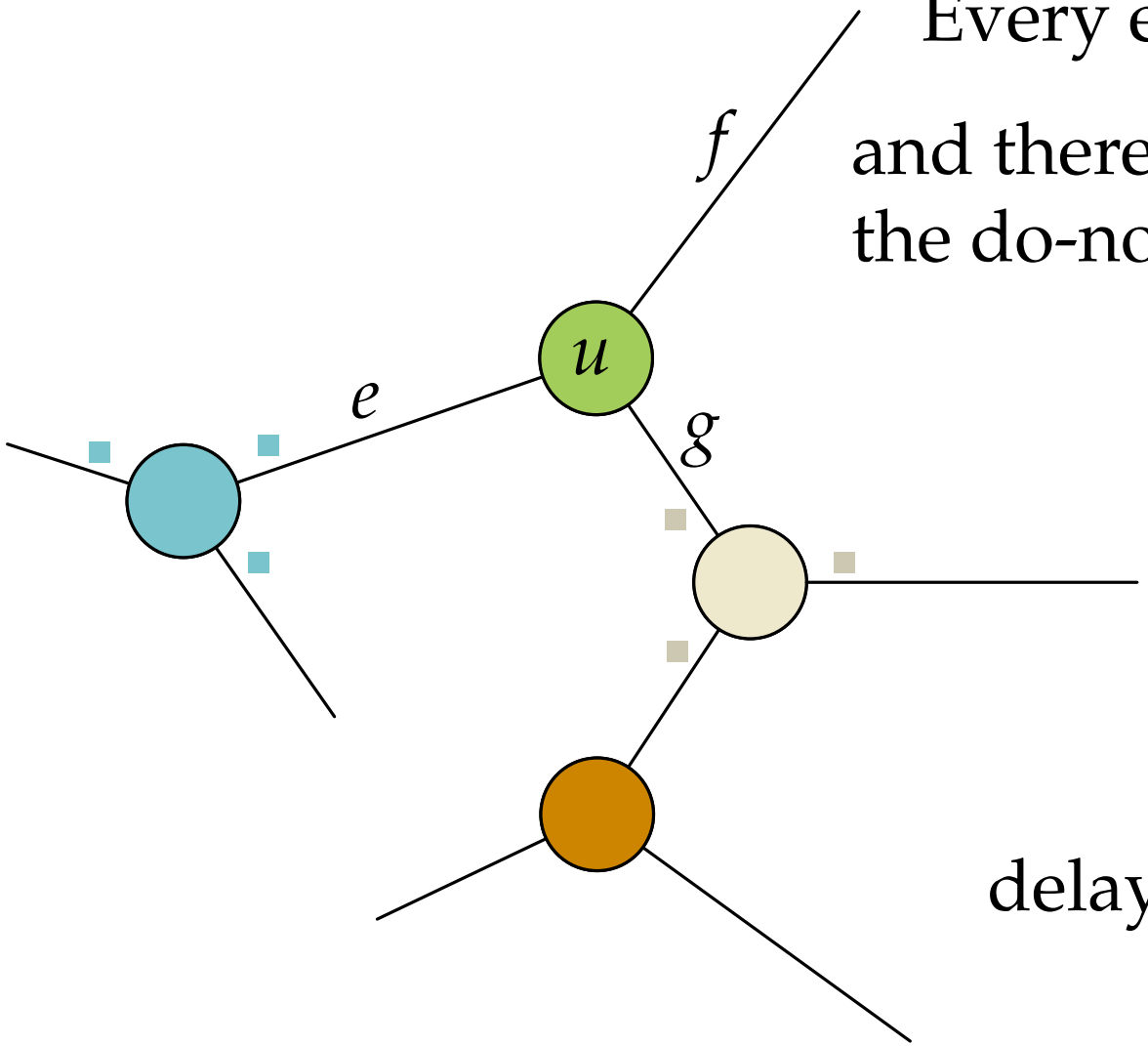
$S_u = \{\{e, f, g\}, \{\text{DNB}\}\}$

Using Congestion Games to Solve Max-IS

Every vertex is a player

Every edge is a resource

and there is another resource, the do-nothing-box.



social cost = 2

$\text{delay}(\text{DNB}, i) = 1$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

$S_u = \{\{e, f, g\}, \{\text{DNB}\}\}$

Using Congestion Games to Solve Max-IS

$$S_u := \{\{e \in E \mid u \in e\}, \{\text{DNB}\}\}$$

Observation. The social optimum is $n - |\text{Max-IS}|$.

$$\text{delay}(\text{DNB}, i) = 1$$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

$$S_u = \{\{e, f, g\}, \{\text{DNB}\}\}$$

Using Congestion Games to Solve Max-IS

$$S_u := \{\{e \in E \mid u \in e\}, \{\text{DNB}\}\}$$

Observation. The social optimum is $n - |\text{Max-IS}|$.

Corollary. Computing the social optimum is NP-hard.

$$\text{delay}(\text{DNB}, i) = 1$$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

$$S_u = \{\{e, f, g\}, \{\text{DNB}\}\}$$

Using Congestion Games to Solve Max-IS

$$S_u := \{\{e \in E \mid u \in e\}, \{\text{DNB}\}\}$$

Observation. The social optimum is $n - |\text{Max-IS}|$.

Corollary. Computing the social optimum is NP-hard.

But. It's usually not the social optimum we are looking for.

$$\text{delay}(\text{DNB}, i) = 1$$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

$$S_u = \{\{e, f, g\}, \{\text{DNB}\}\}$$

Social optimum in an ideal world

Social optimum in an ideal world

Collective farm holiday, A. A. Plastov, 1937

Image source: `https://rusmuseumvrm.ru`

Social optimum in an ideal world

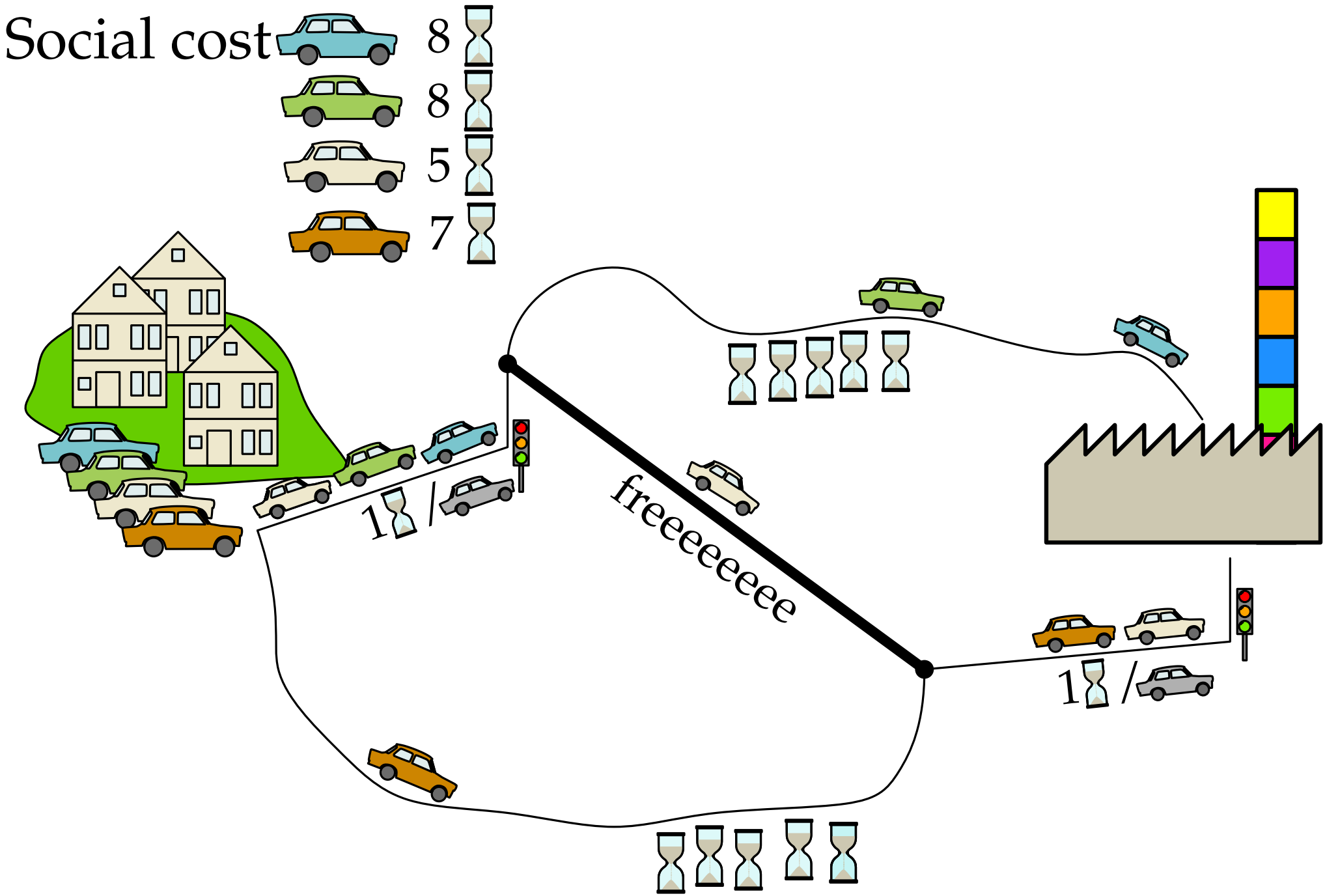
Collective farm holiday, A. A. Plastov, 1937

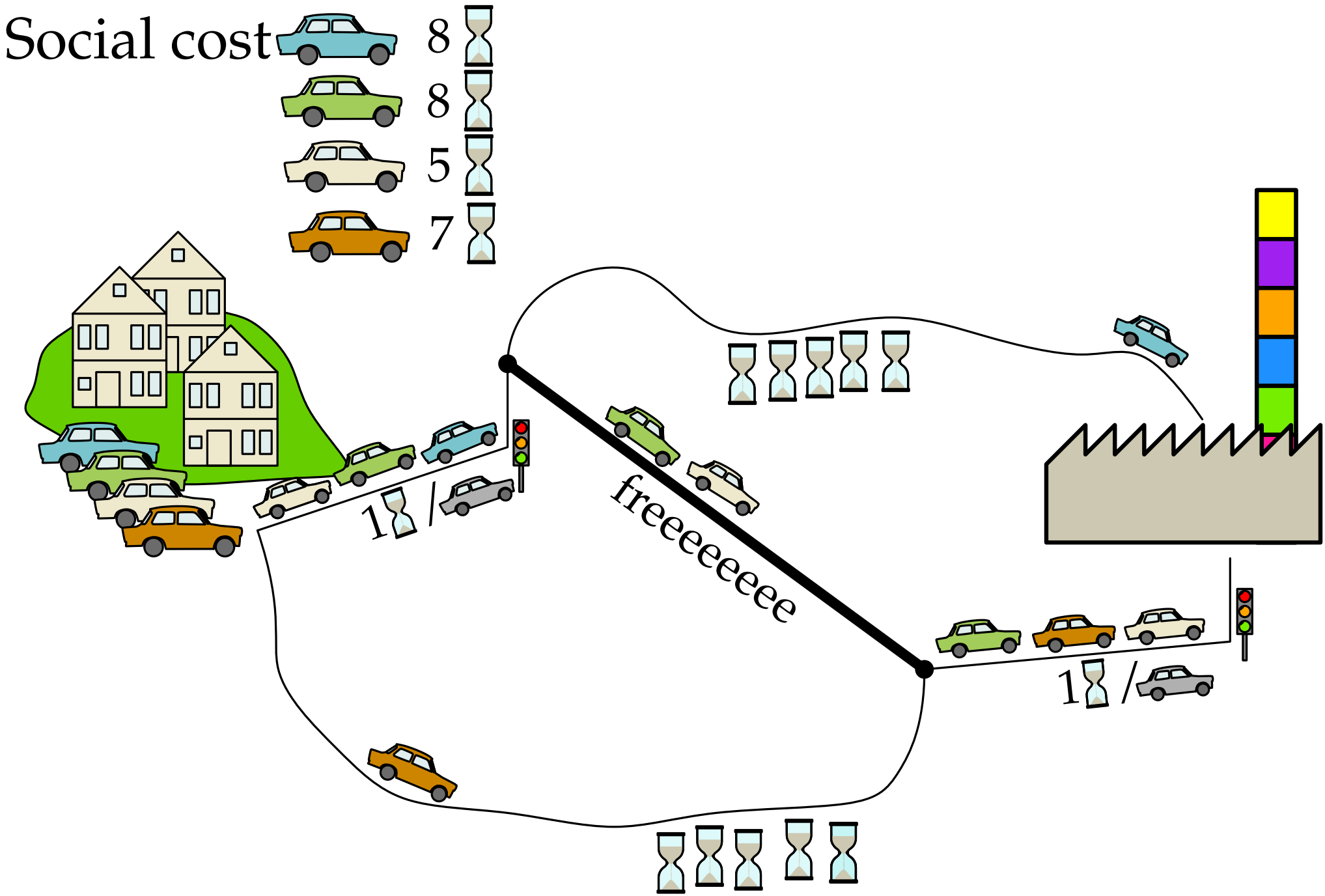
Image source: `https://rusmuseumvrm.ru`

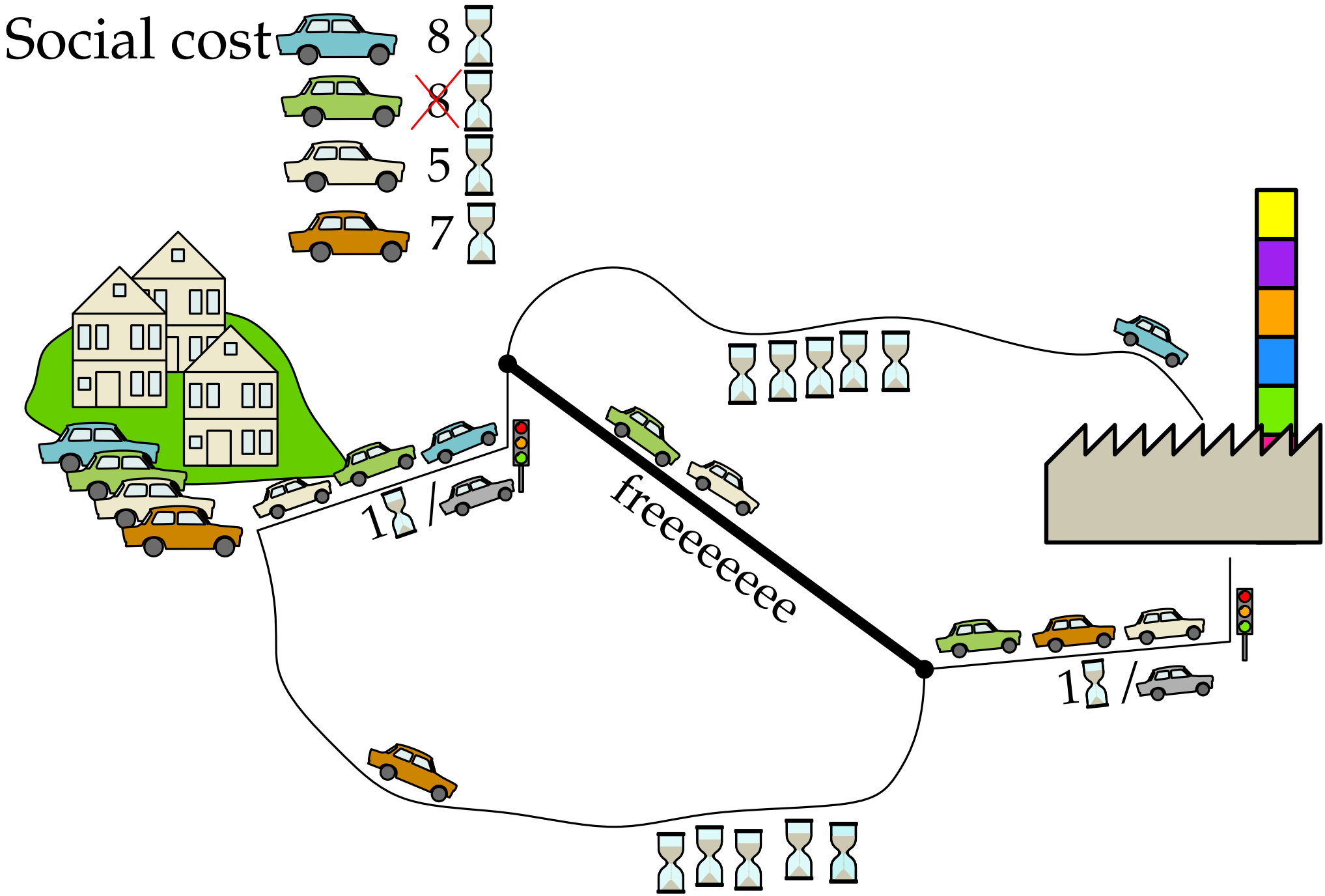
Outcome in the real world

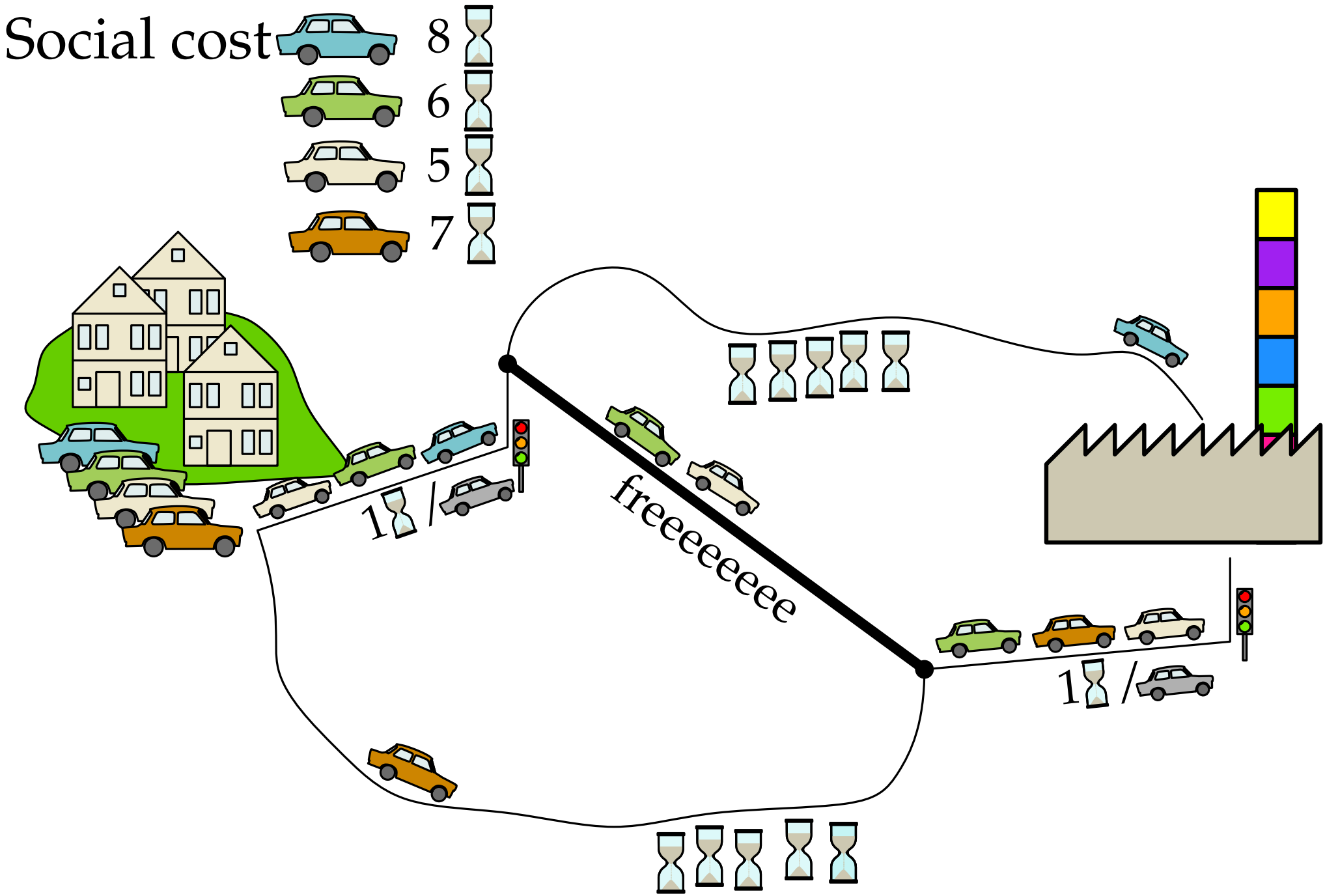
Unemployed men queued outside a depression soup kitchen opened in Chicago by Al Capone

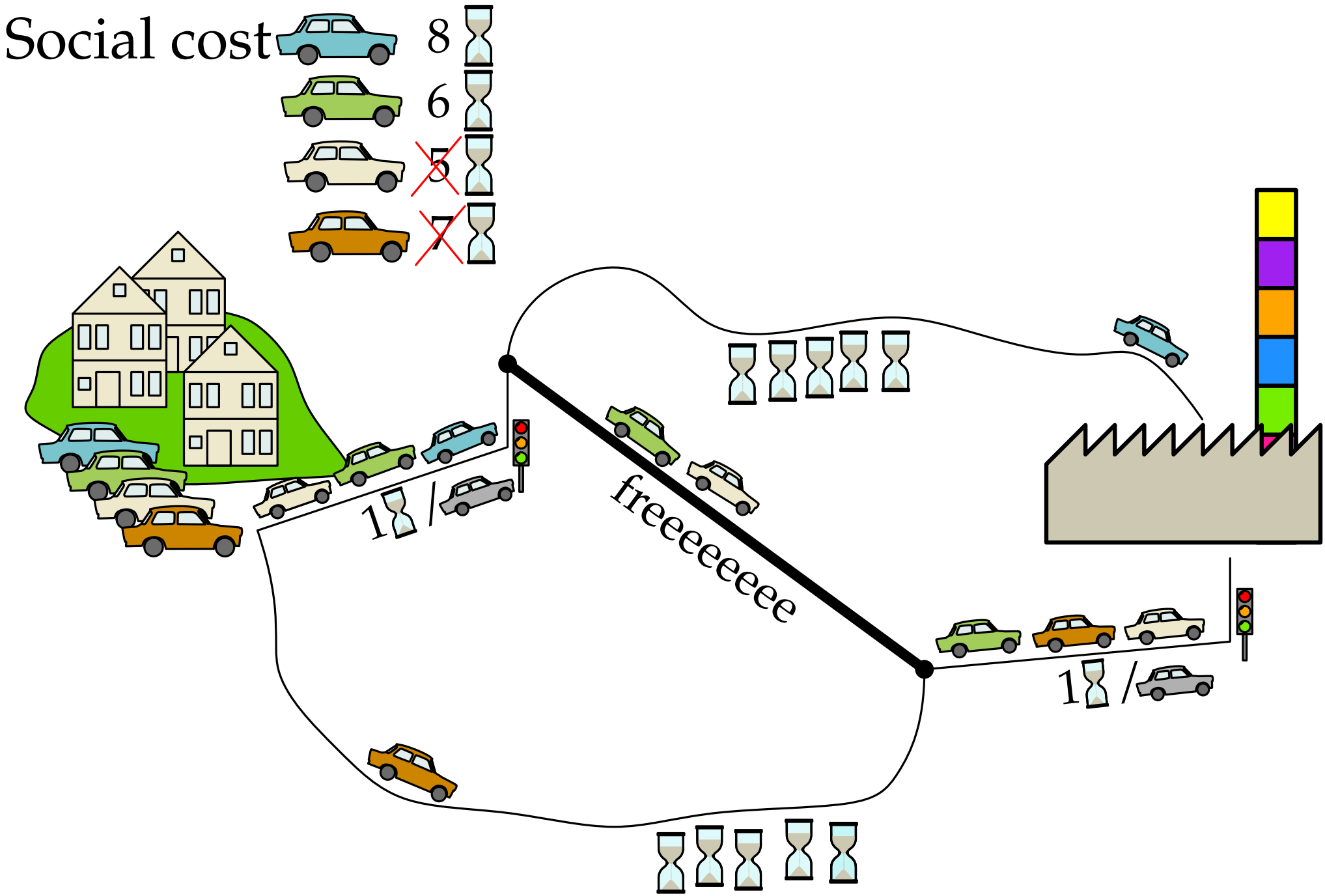
Image source: wikimedia

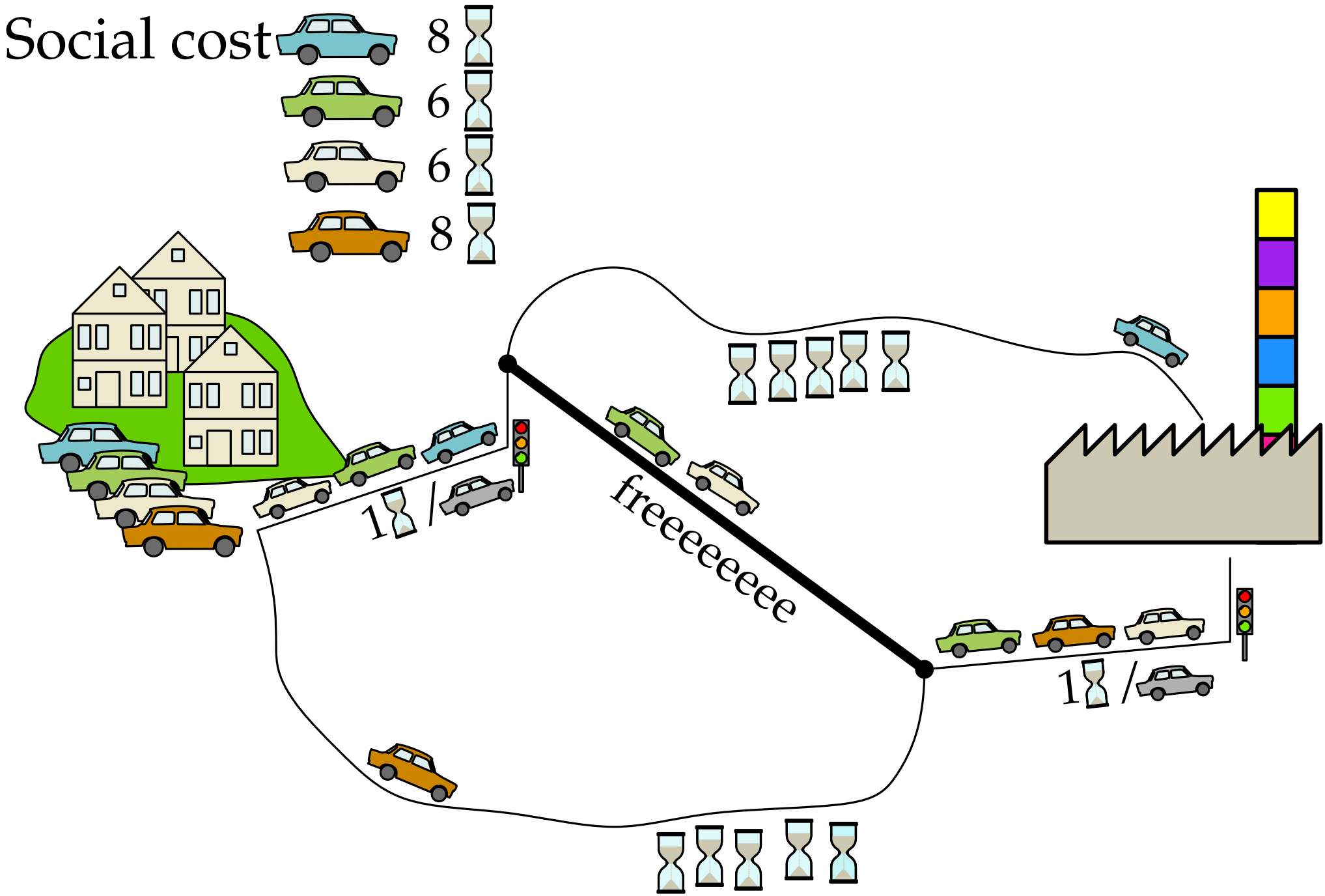

Social cost
8
8
5
7
1
freeeeeeeee
1

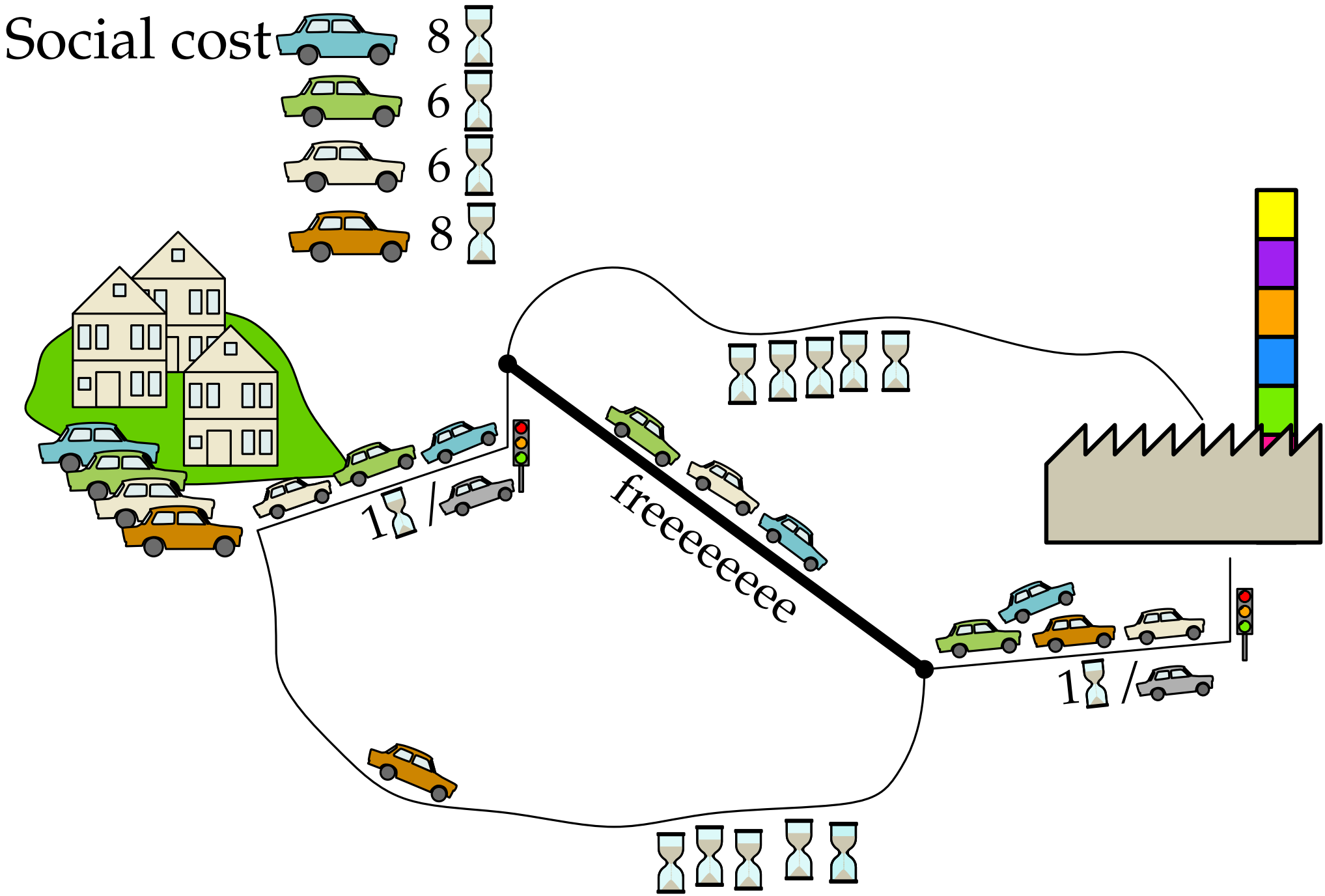

Social cost
8
8
5
7
1
freeeeeeeee
1

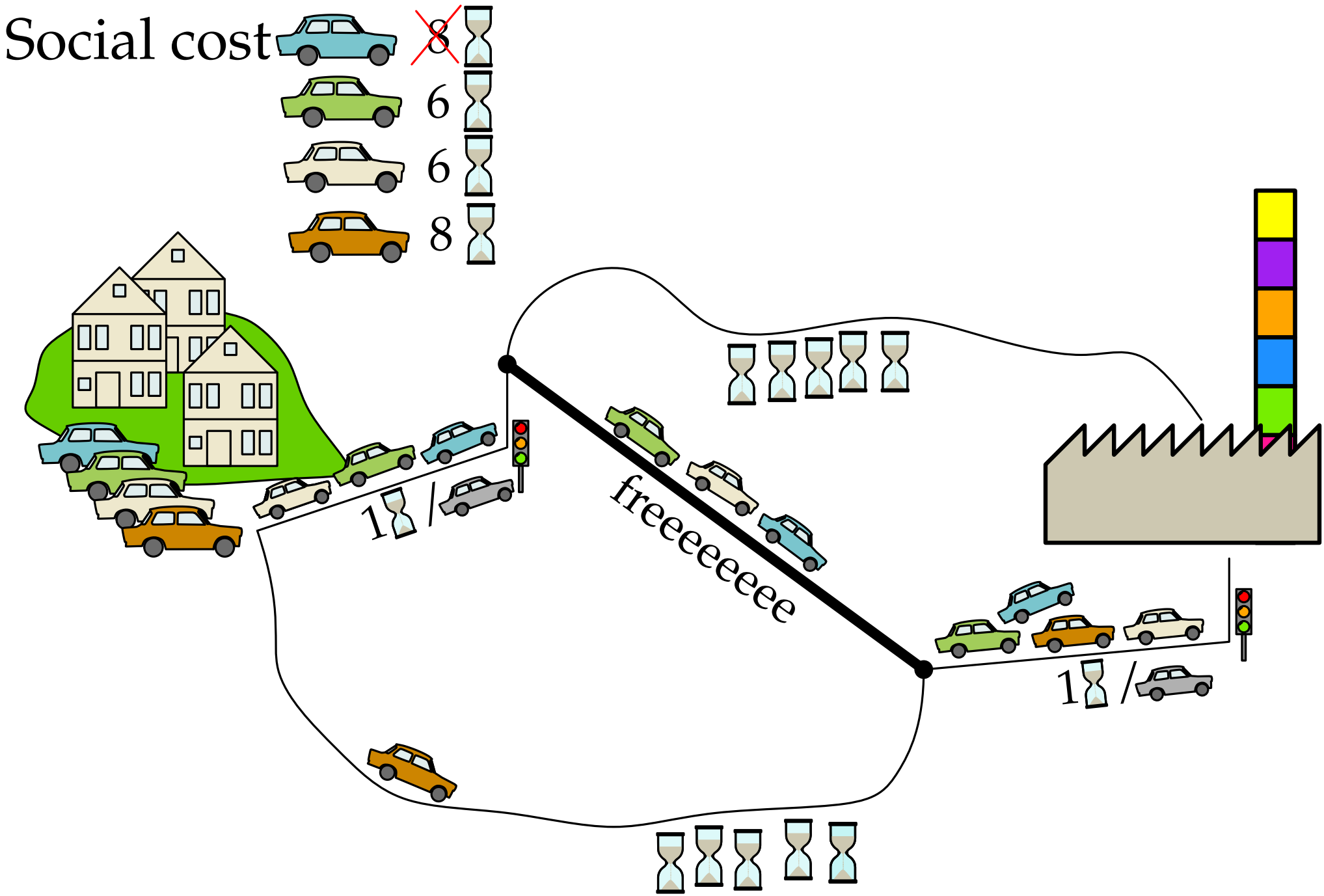

Social cost
8
8
5
7
1
freeeeeeeee
1

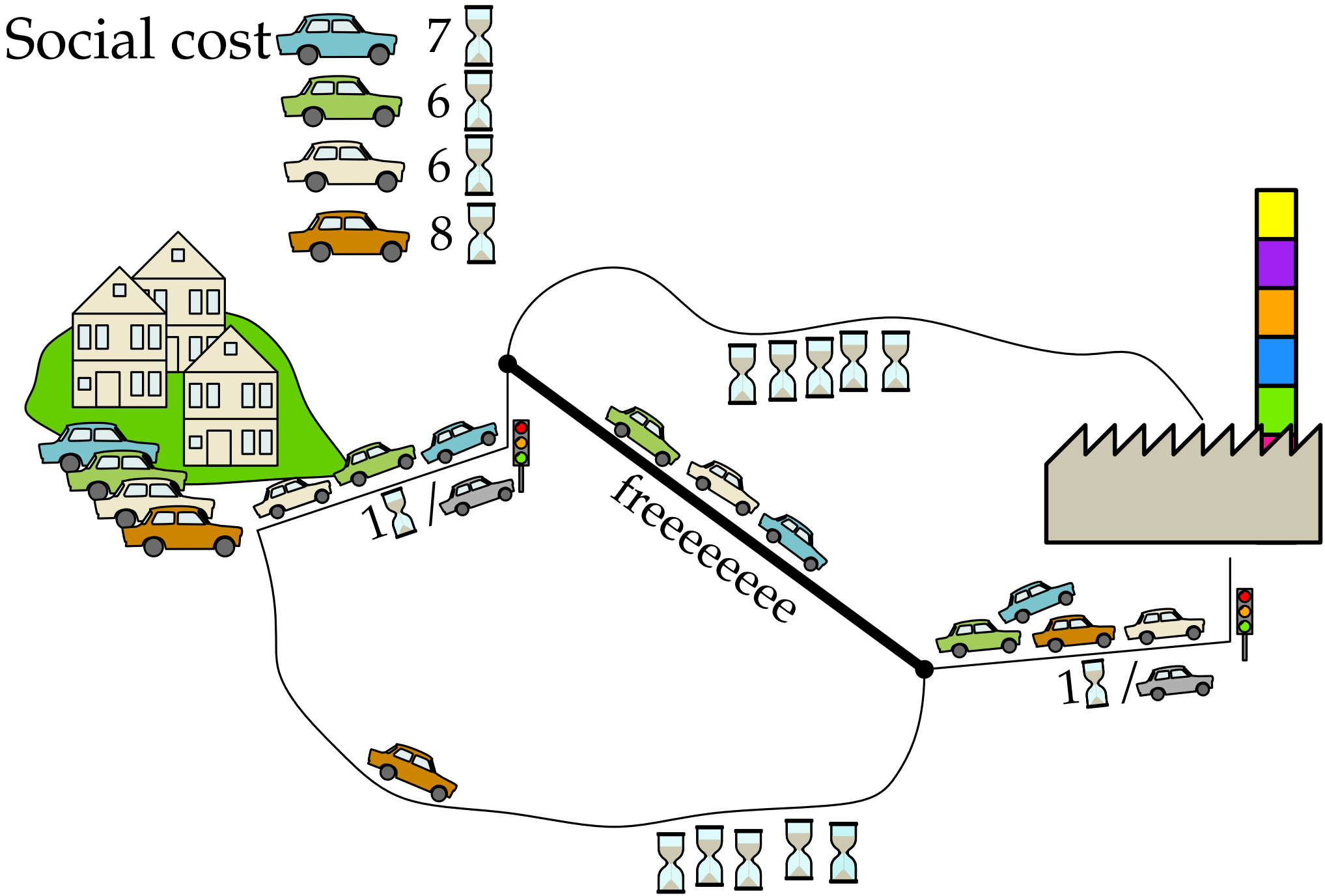

Social cost
8
6
5
7
1
freeeeeeeee
1

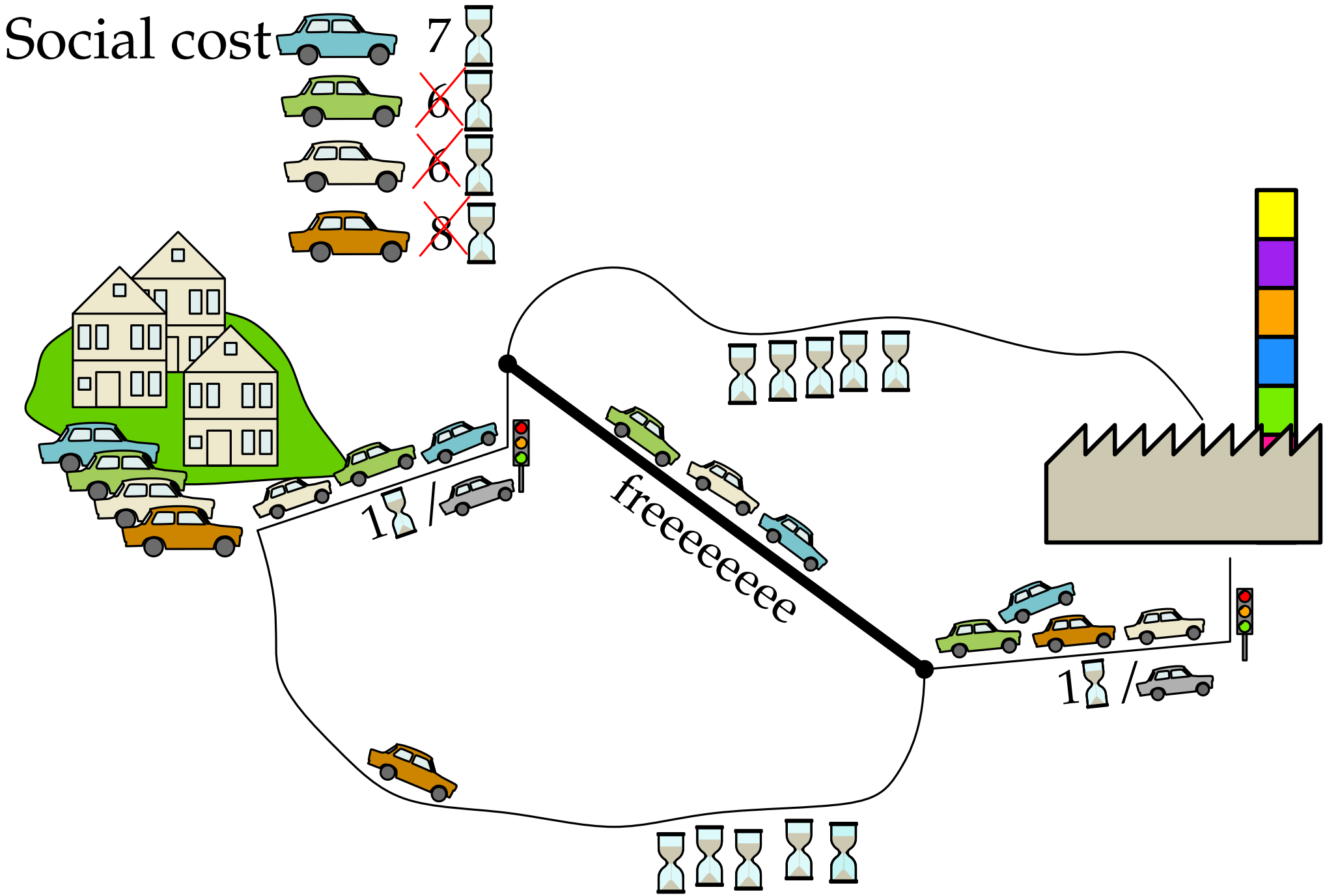

Social cost
8
6
5
7
freeeeeeeee

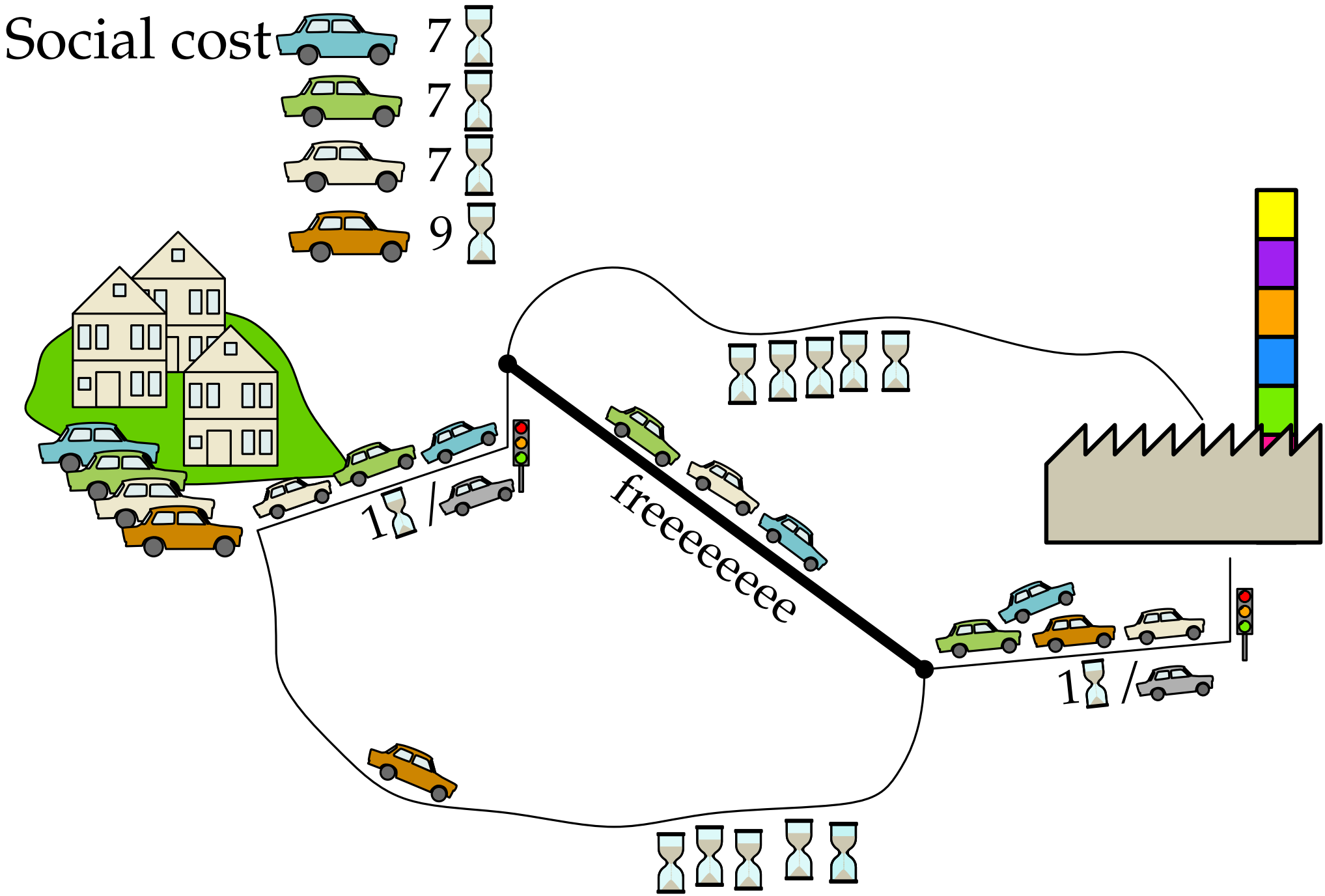

Social cost
8
6
6
8
1
freeeeeeeee
1

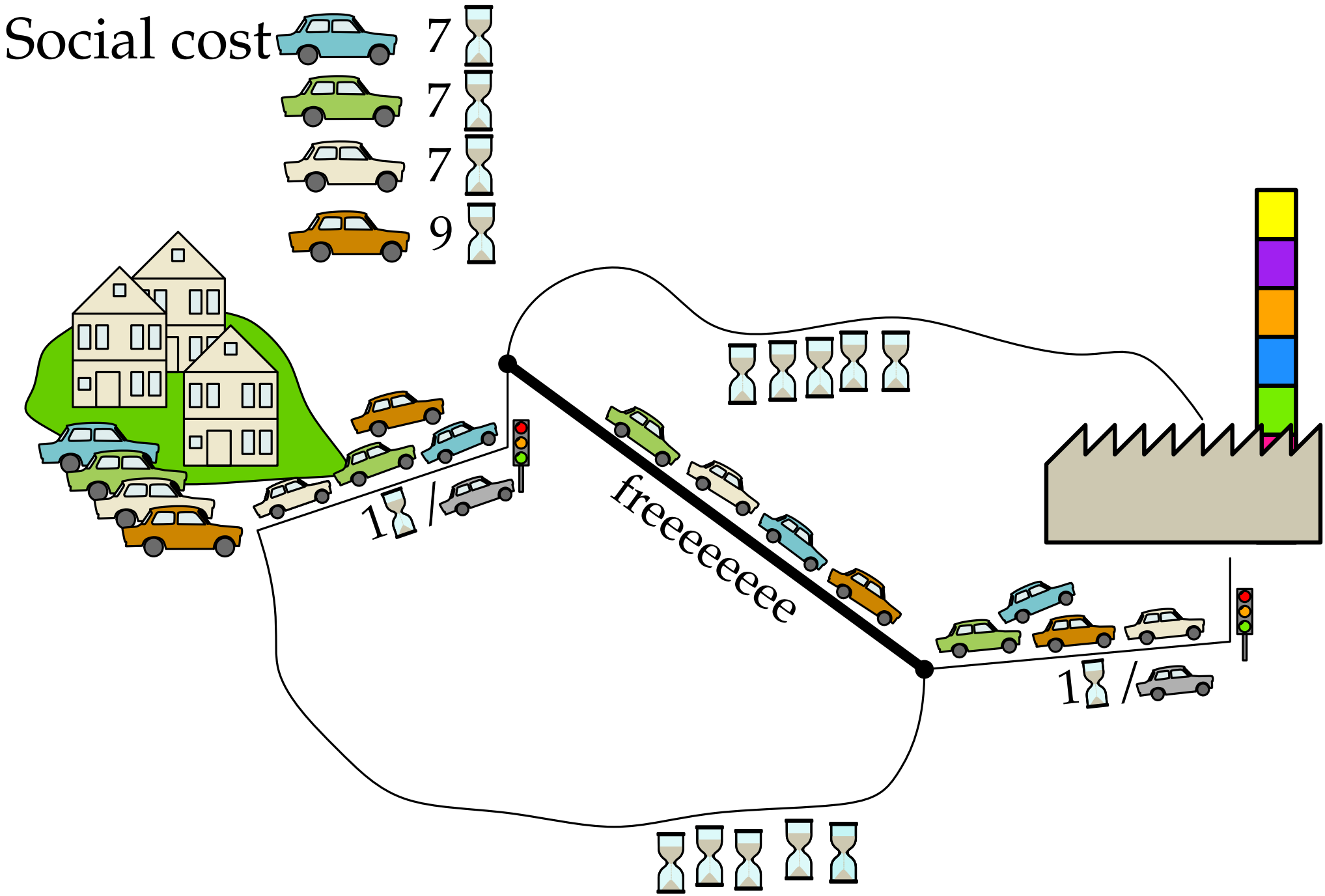

Social cost
8
6
6
8
1
freeeeeeeee
1

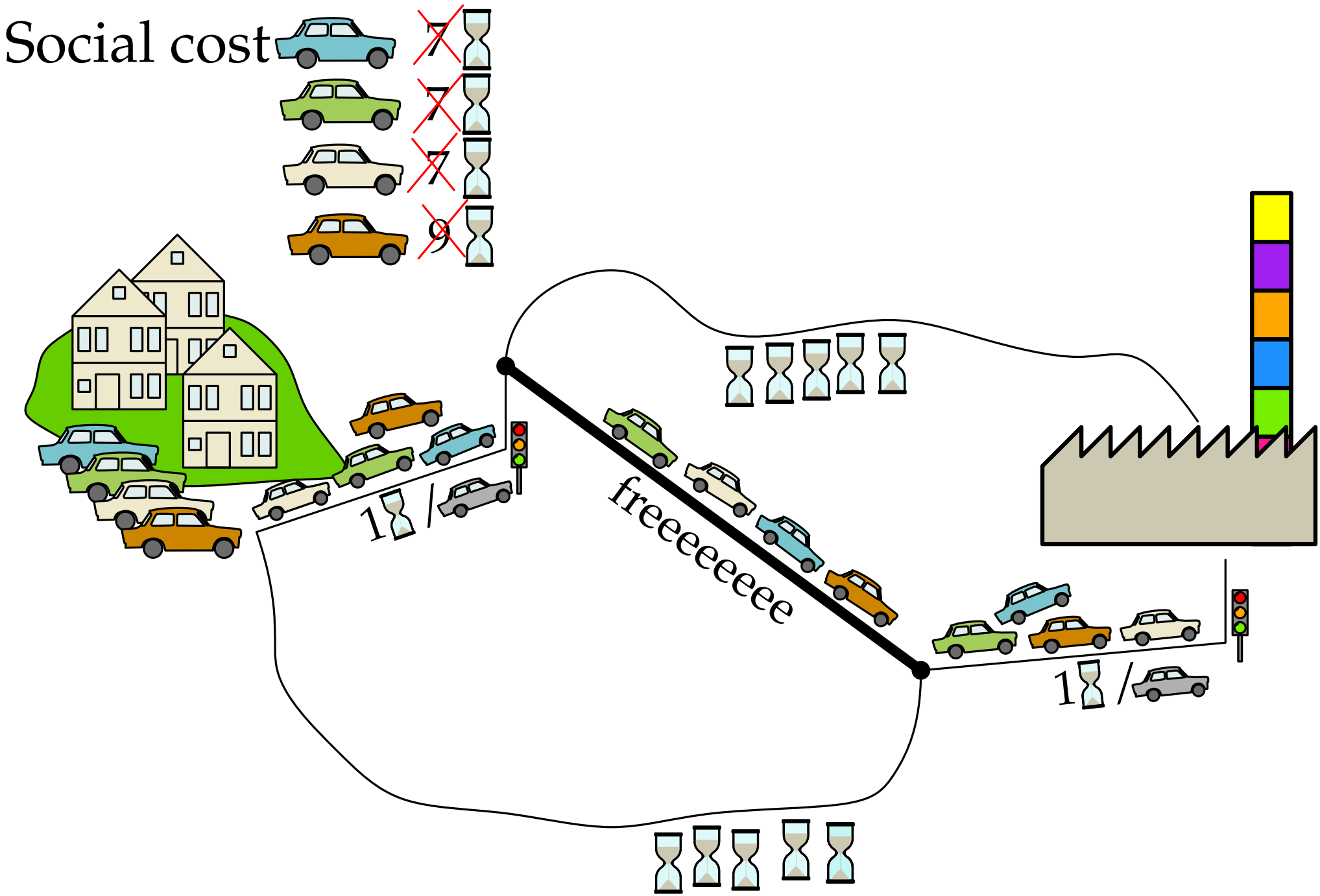

Social cost
8
6
6
8
1
freeeeeeeee
1

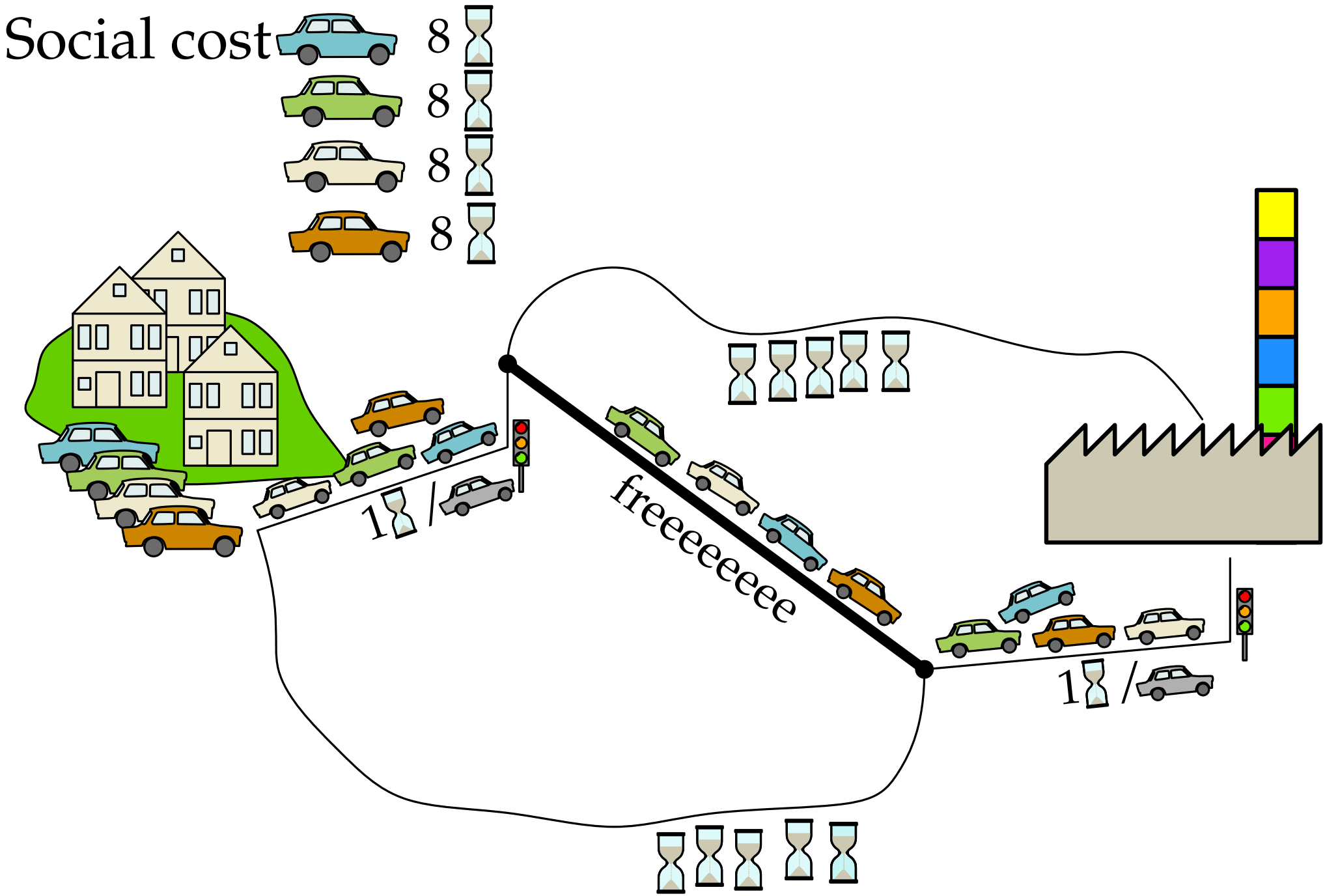

Social cost
7
6
6
8
1
freeeeeeeee
1

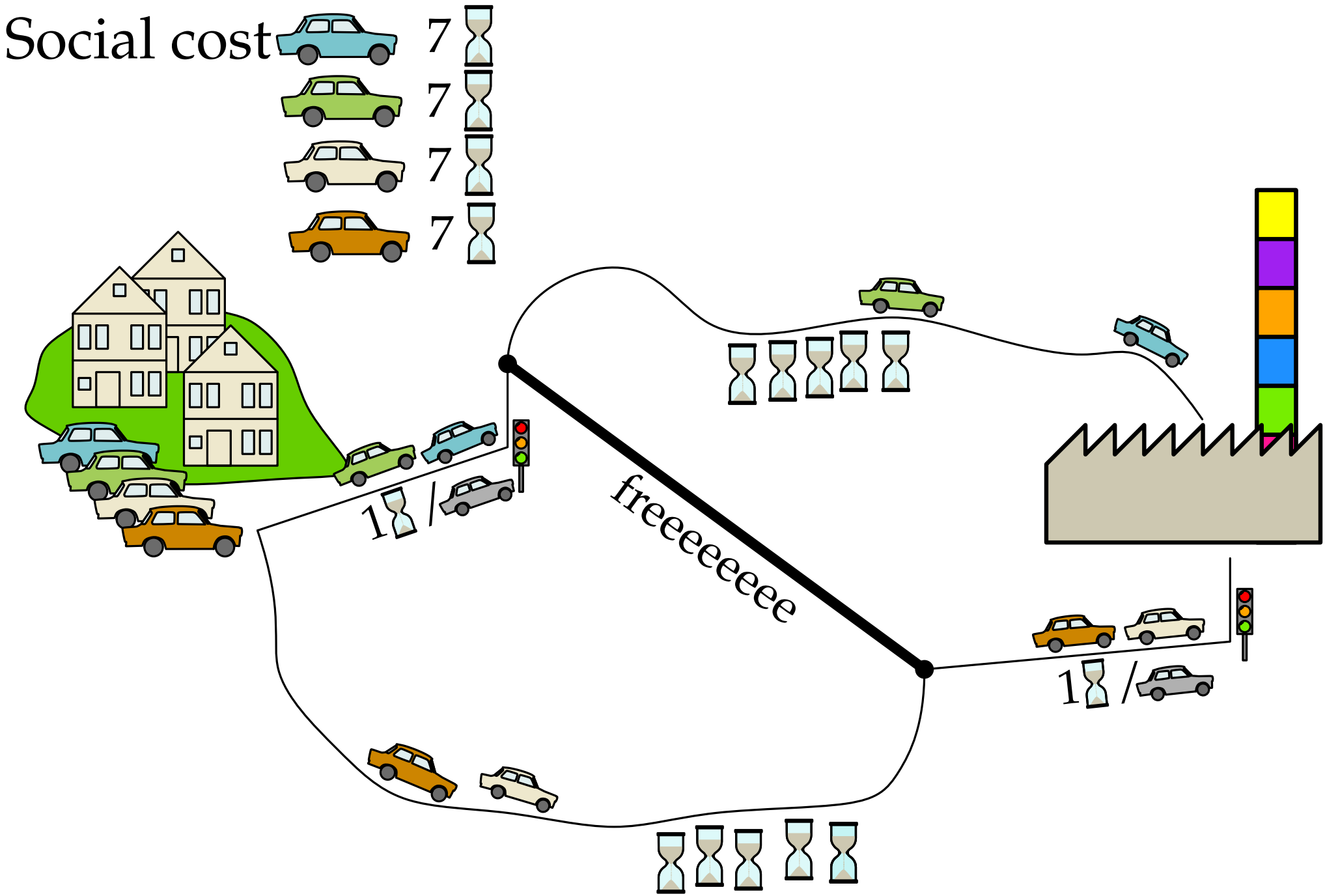

Social cost
7
6
6
8
1
freeeeeeeee
1

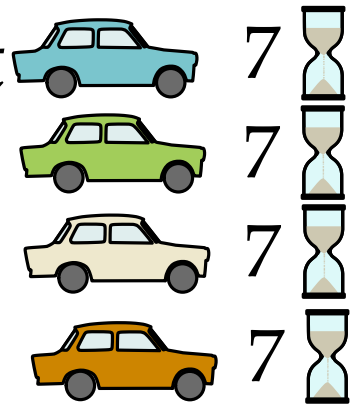

Social cost
7
7
7
9
1
freeeeeeeee
1


Social cost
7
7
7
9
1
freeeeeeeee
1

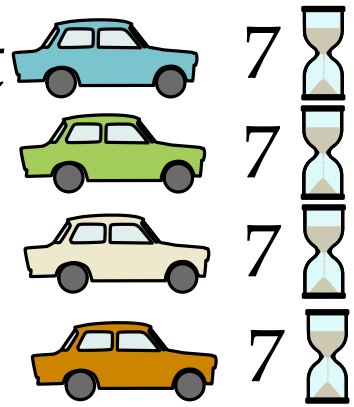

Social cost
7
7
7
9
1
freeeeeeeee
1


Social cost
8
8
8
8
1
freeeeeeeee
1


Social cost
7
7
7
7
freeeeeeeee
1
1

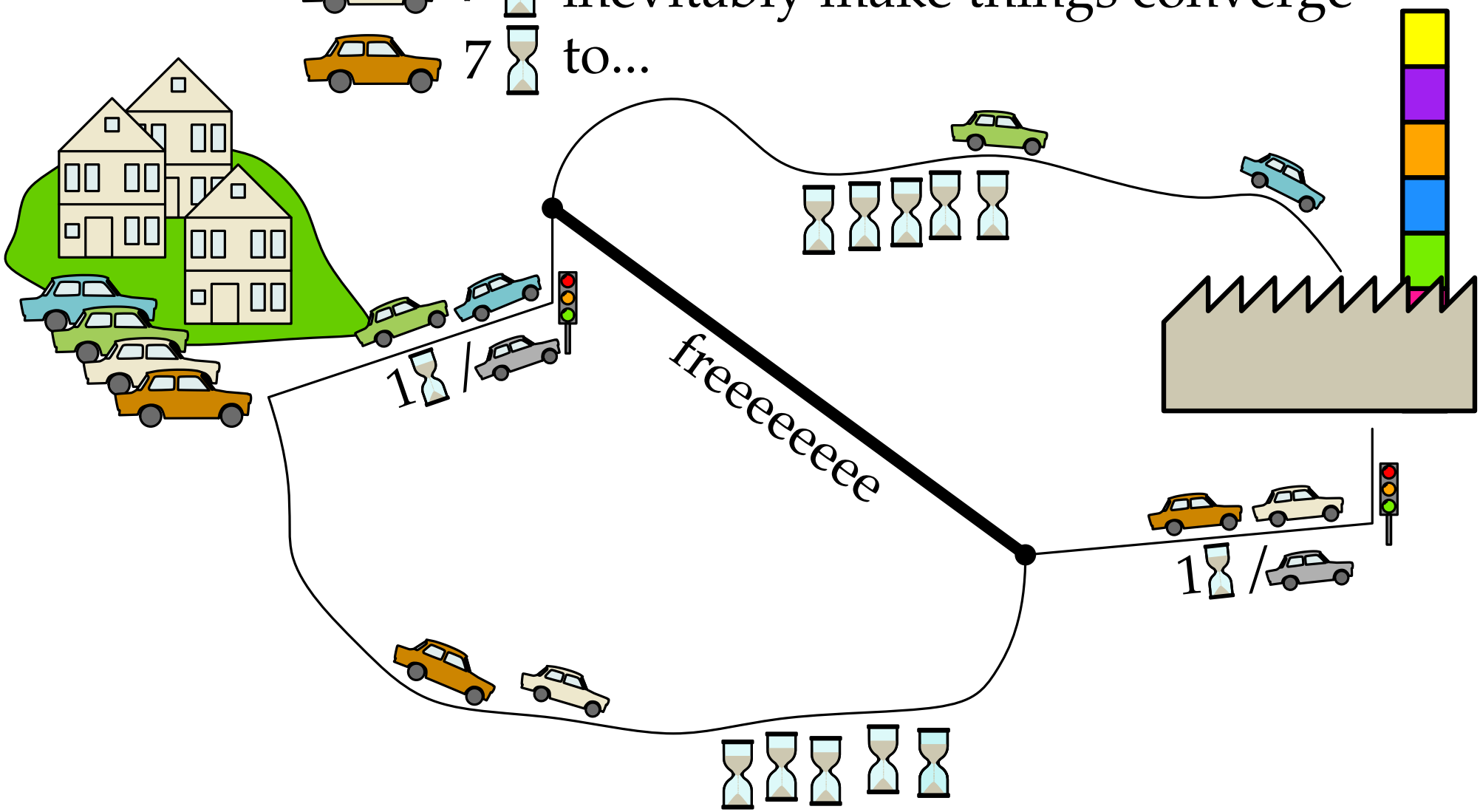
Social cost  This is optimal.

Social cost



This is optimal.

But individual decisions will inevitably make things converge to...



Social cost


8


8


8


8

This is optimal.

But individual decisions will inevitably make things converge to...

Social optimum: The strategy profile that minimizes the total cost to the players.

Social optimum: The strategy profile that minimizes the total cost to the players.

Nash equilibrium. A strategy profile where no player can improve by changing her strategy.

Social optimum: The strategy profile that minimizes the total cost to the players.

Nash equilibrium. A strategy profile where no player can improve by changing her strategy.

Question. Does a Nash equilibrium always exist? Sure they won't run in cycles?

Question. Does a Nash equilibrium always exist? Sure they won't run in cycles?

image source: wikimedia

Question. Does a Nash equilibrium always exist? Sure they won't run in cycles?

image source: wikimedia

Question. Does a Nash equilibrium always exist? Sure they won't run in cycles?

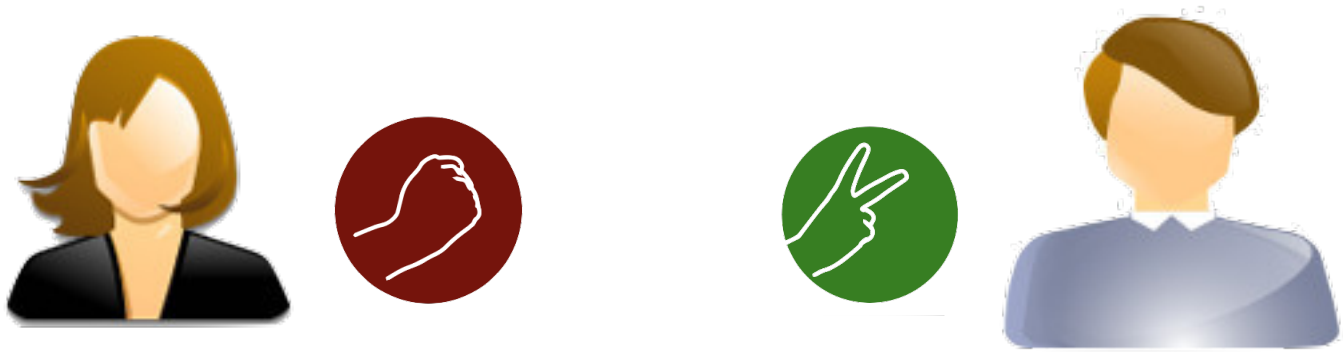


image source: wikimedia

Question. Does a Nash equilibrium always exist? Sure they won't run in cycles?

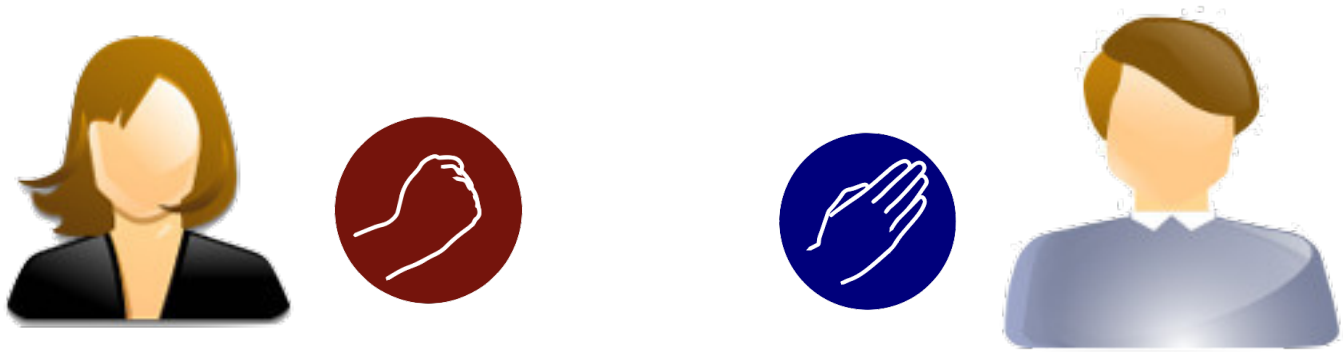


image source: wikimedia

Question. Does a Nash equilibrium always exist? Sure they won't run in cycles?

image source: wikimedia

Question. Does a Nash equilibrium always exist? Sure they won't run in cycles?

image source: wikimedia

Question. Does a Nash equilibrium always exist? Sure they won't run in cycles?

image source: wikimedia

Question. Does a Nash equilibrium always exist? Sure they won't run in cycles?

image source: wikimedia

Question. Does a Nash equilibrium always exist? Sure they won't run in cycles?

...and so proceed ad infinitum...

image source: wikimedia

Question. Does a Nash equilibrium always exist? Sure they won't run in cycles?

...and so proceed ad infinitum...

Yes, there is the concept of a *mixed Nash equilibrium* using randomness, but we will not consider this in this talk.

image source: wikimedia

Nash equilibrium. A strategy profile where no player can improve by changing her strategy.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Nash equilibrium. A strategy profile where no player can improve by changing her strategy.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Proof. Just start with some strategy profile $\vec{s}$.

Nash equilibrium. A strategy profile where no player can improve by changing her strategy.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Proof. Just start with some strategy profile $\vec{s}$. As long as some player can improve by switching her strategy, let her switch.

Nash equilibrium. A strategy profile where no player can improve by changing her strategy.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Proof. Just start with some strategy profile $\vec{s}$. As long as some player can improve by switching her strategy, let her switch.
This always improves the social cost.

Nash equilibrium. A strategy profile where no player can improve by changing her strategy.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Proof. Just start with some strategy profile $\vec{s}$. As long as some player can improve by switching her strategy, let her switch.
This always improves the social cost.
There are finitely many profiles $\vec{s}$, so after some time a minimum will be reached.

Nash equilibrium. A strategy profile where no player can improve by changing her strategy.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Proof. Just start with some strategy profile $\vec{s}$. As long as some player can improve by switching her strategy, let her switch.

not true, as we've seen

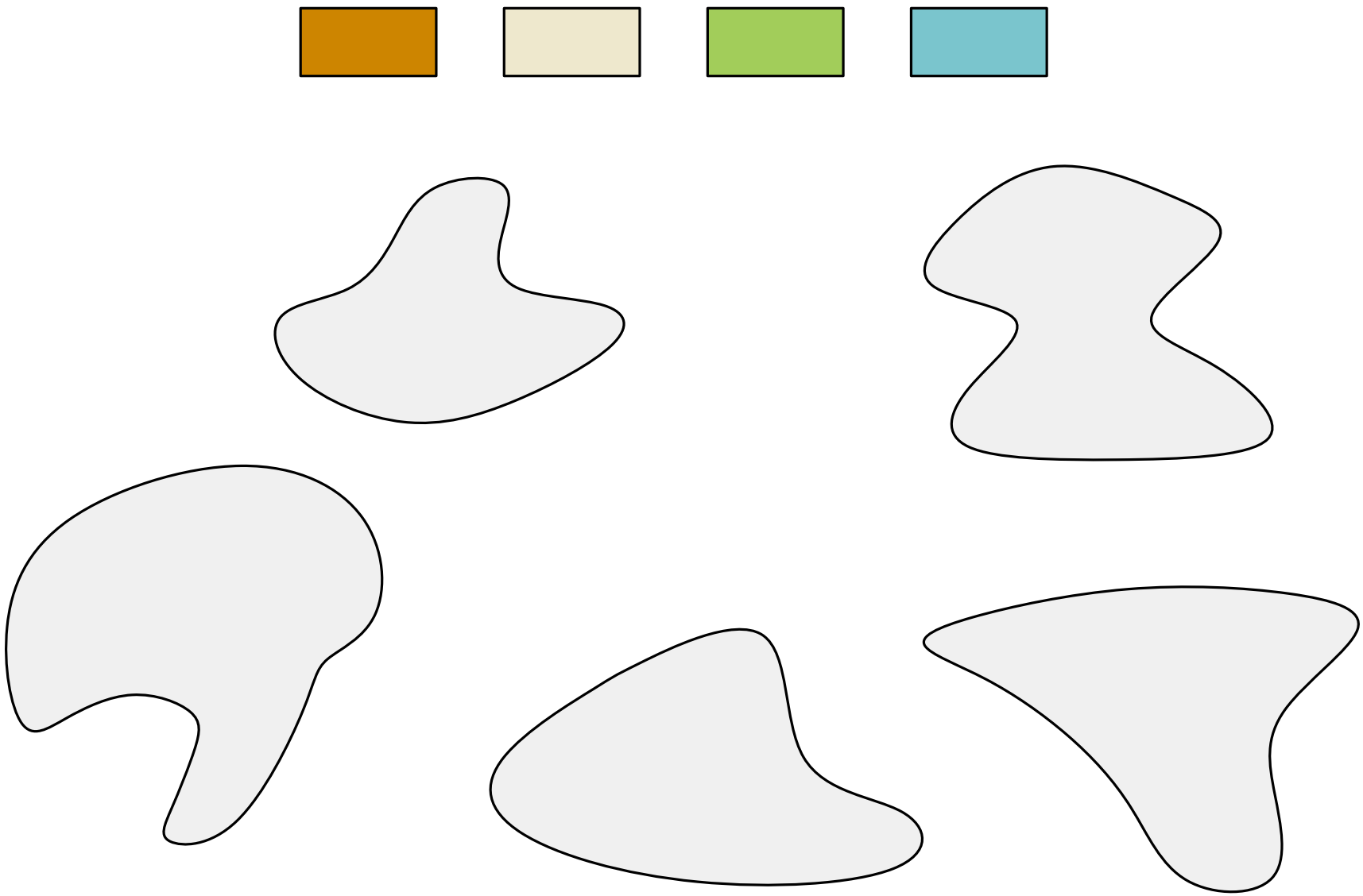
This always improves the social cost.

There are finitely many profiles $\vec{s}$, so after some time a minimum will be reached.

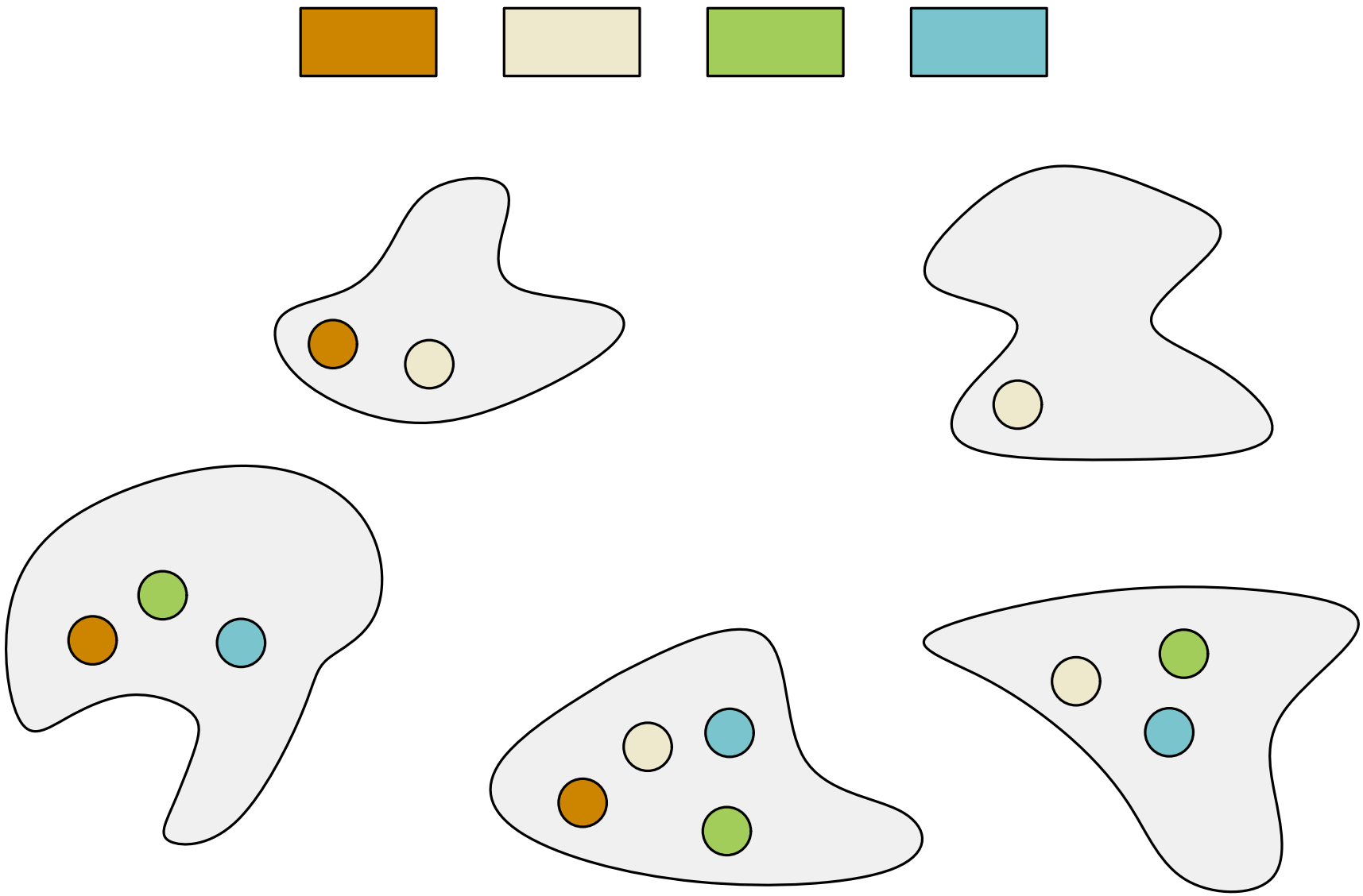
Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

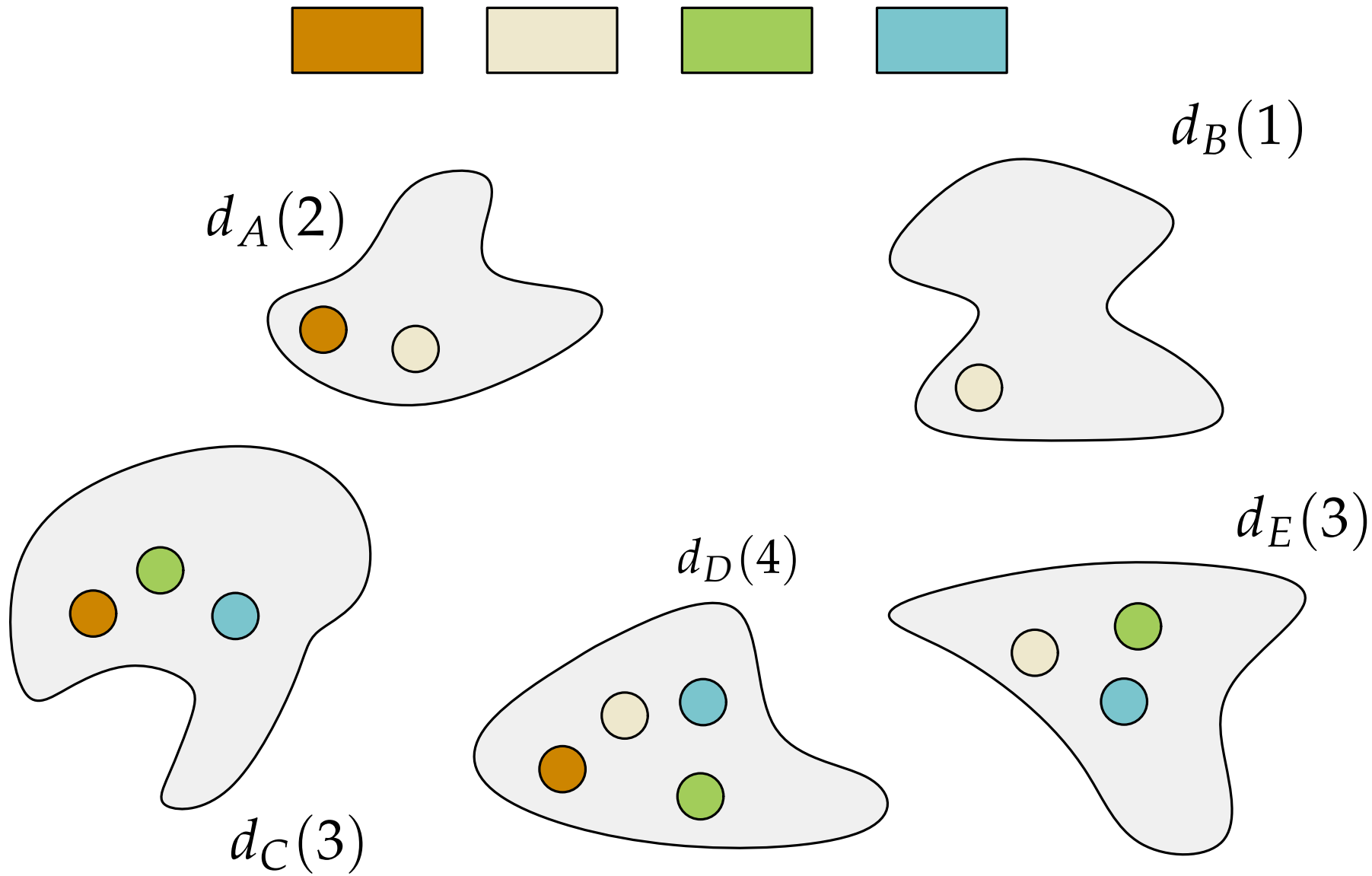
Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.



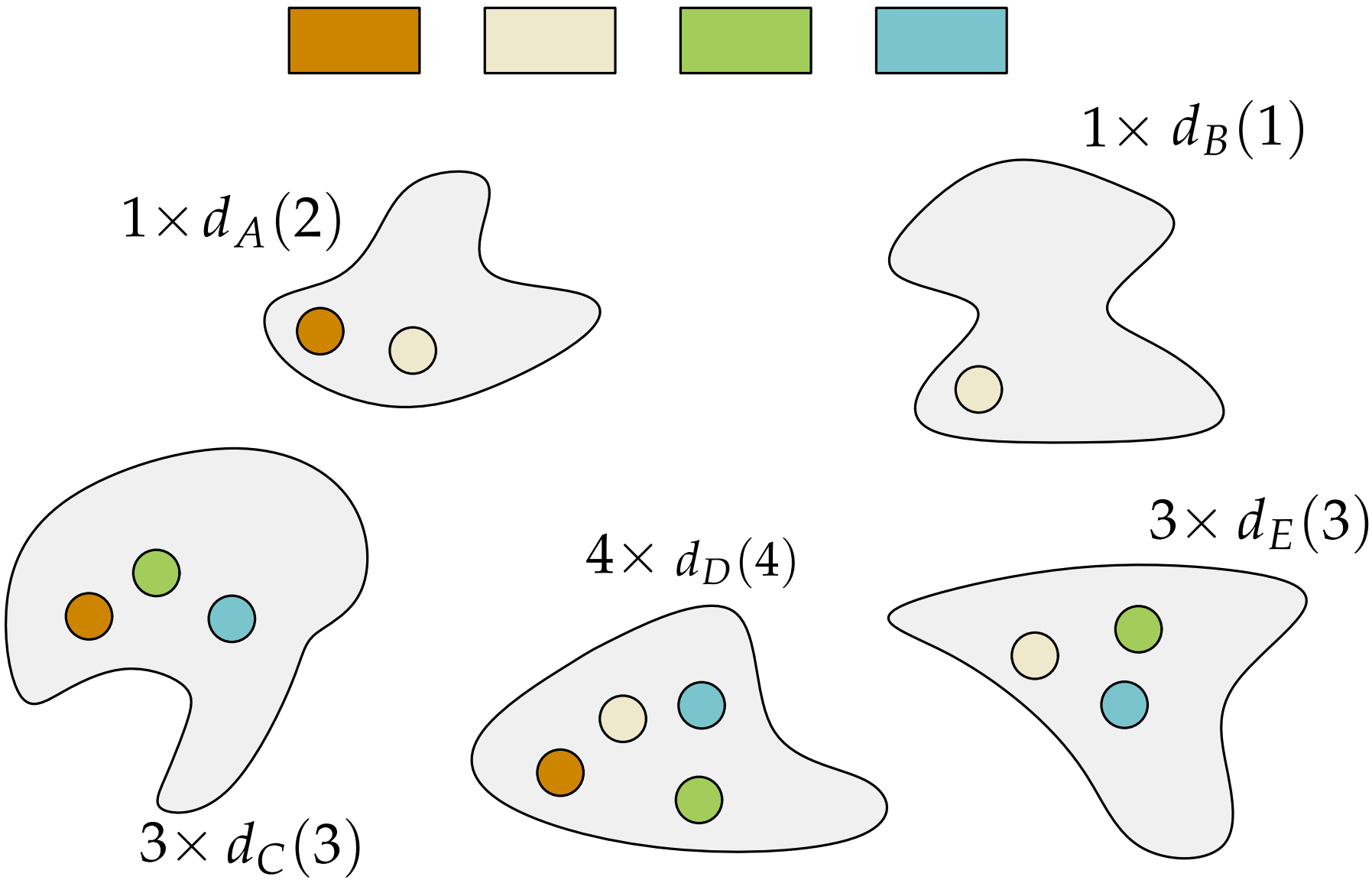
Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.



Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

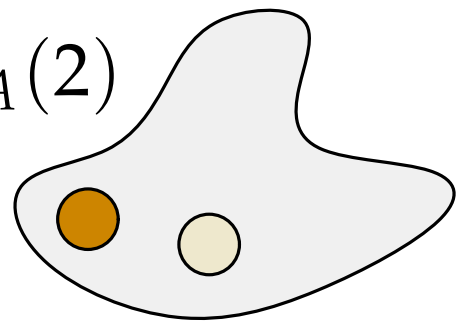


Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

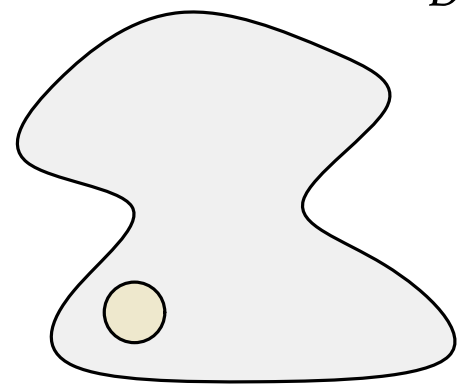


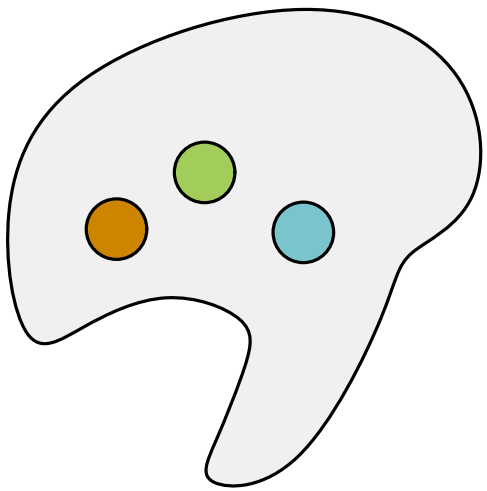
Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

$d_A(1)+d_A(2)$

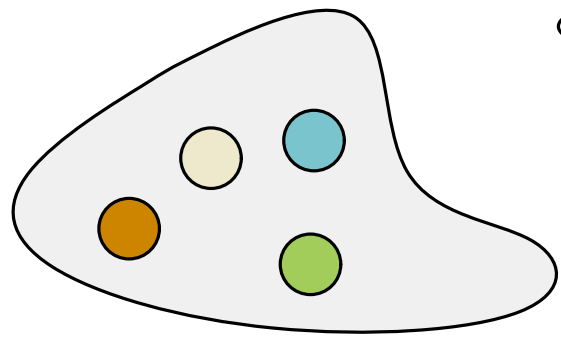


$d_B(1)$

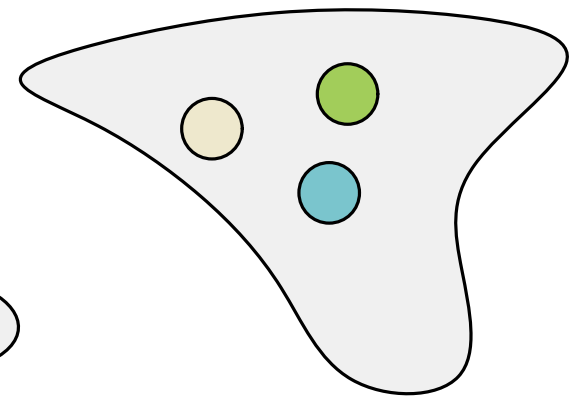




$d_D(1)+\cdots+d_D(4)$



$d_E(1)+d_E(2)+d_E(3)$

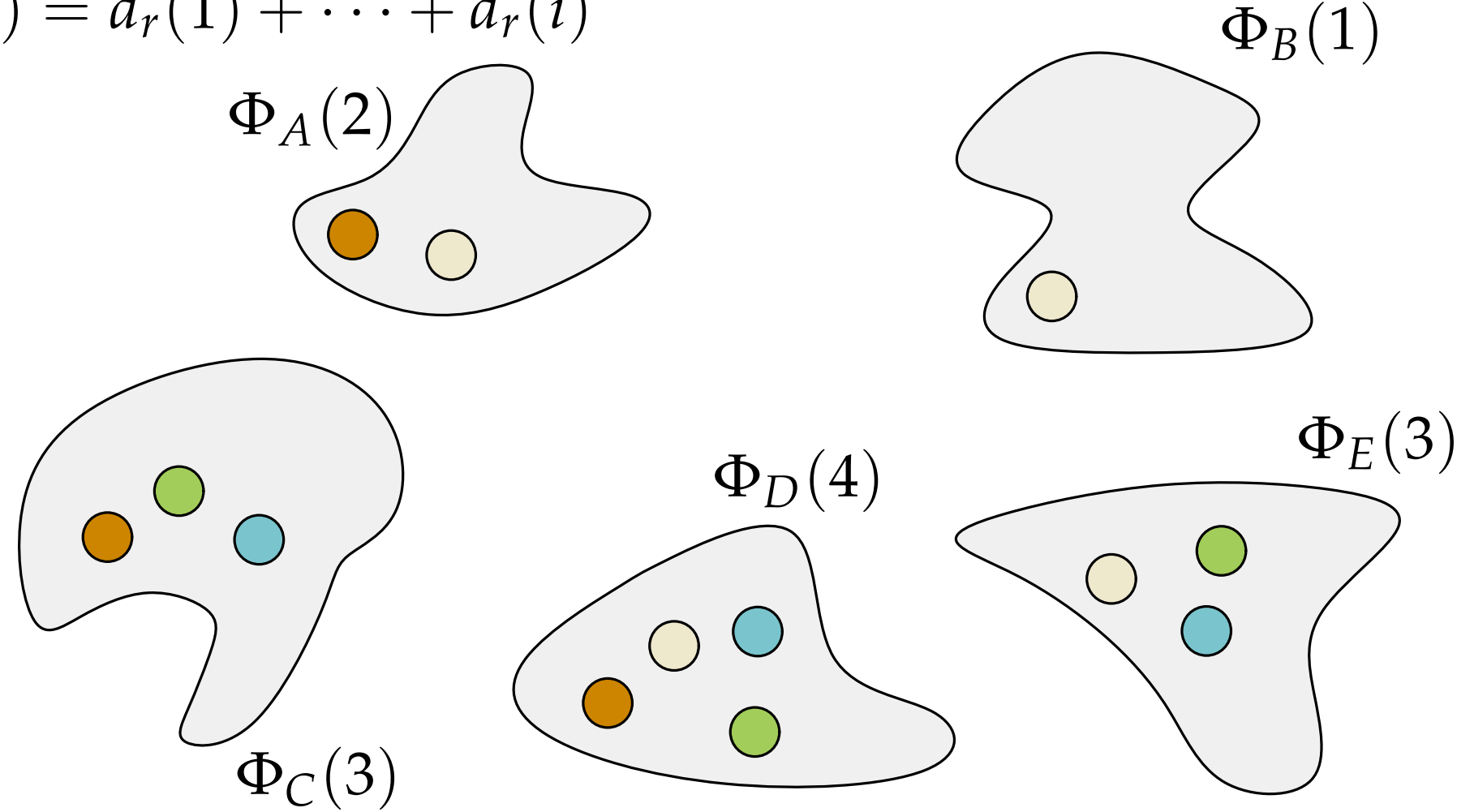


$d_C(1)+d_C(2)+d_C(3)$

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Potential function

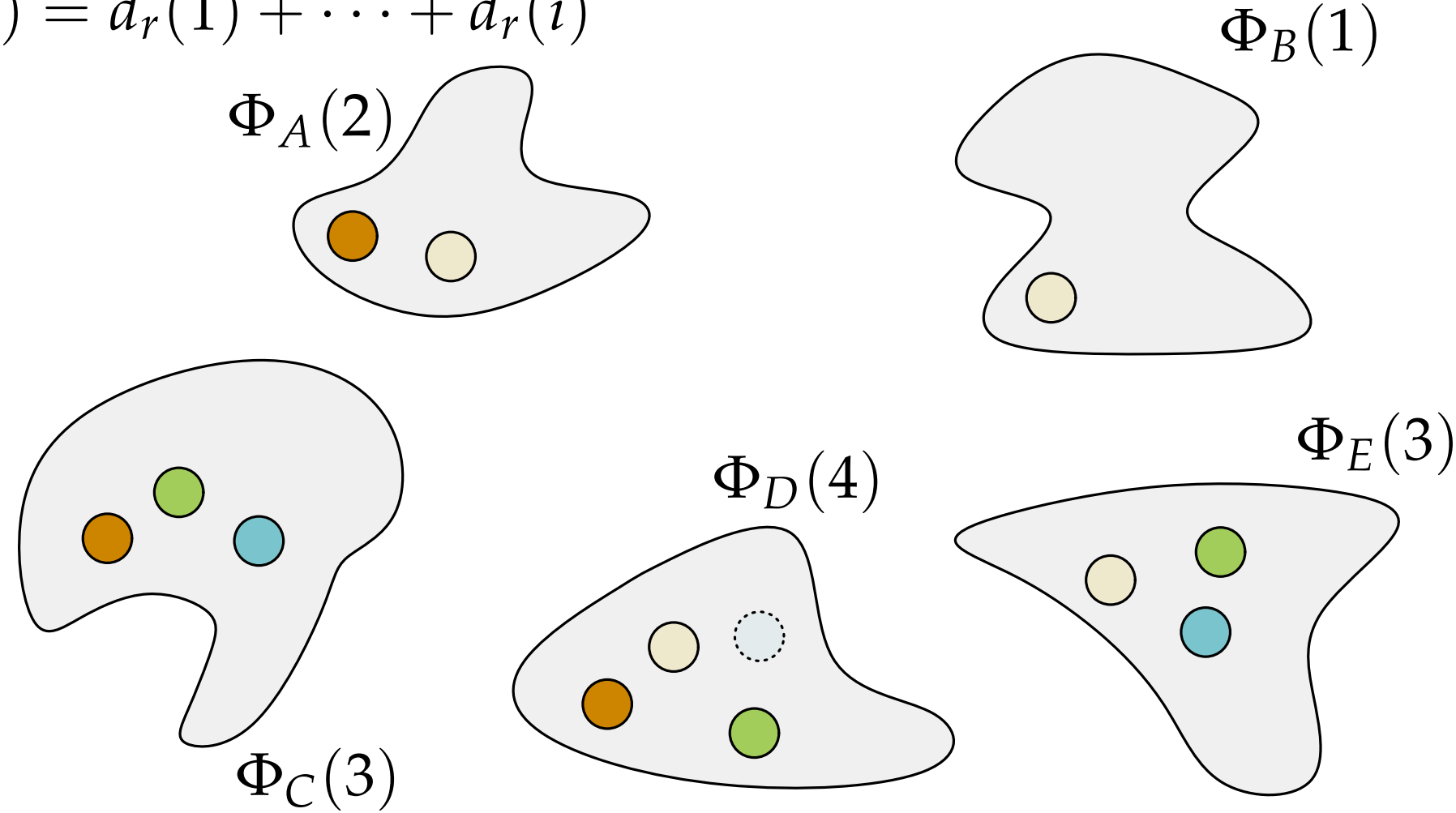
$$\Phi_r(i) = d_r(1) + \cdots + d_r(i)$$



Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Potential function

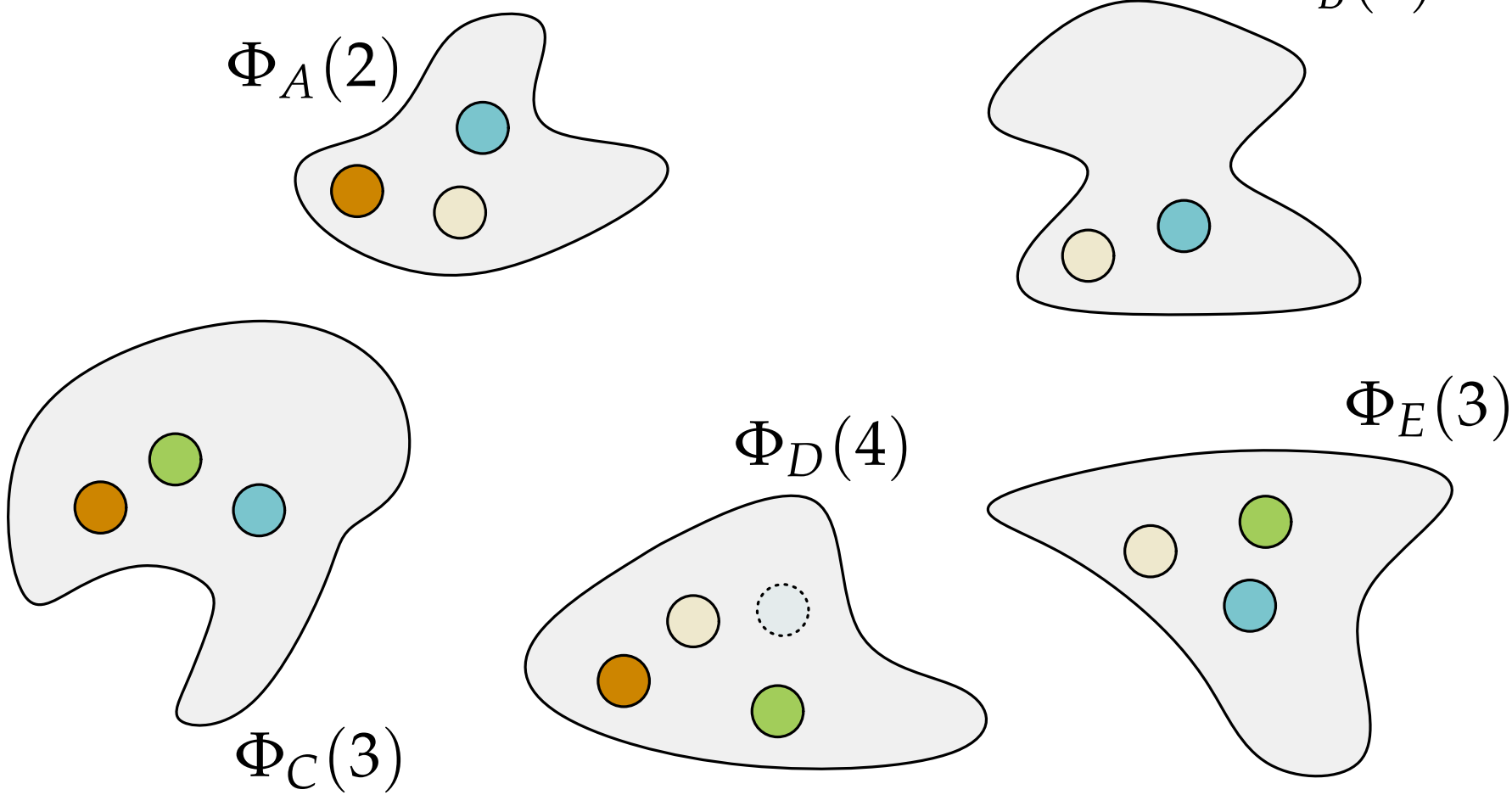
$$\Phi_r(i) = d_r(1) + \cdots + d_r(i)$$



Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Potential function

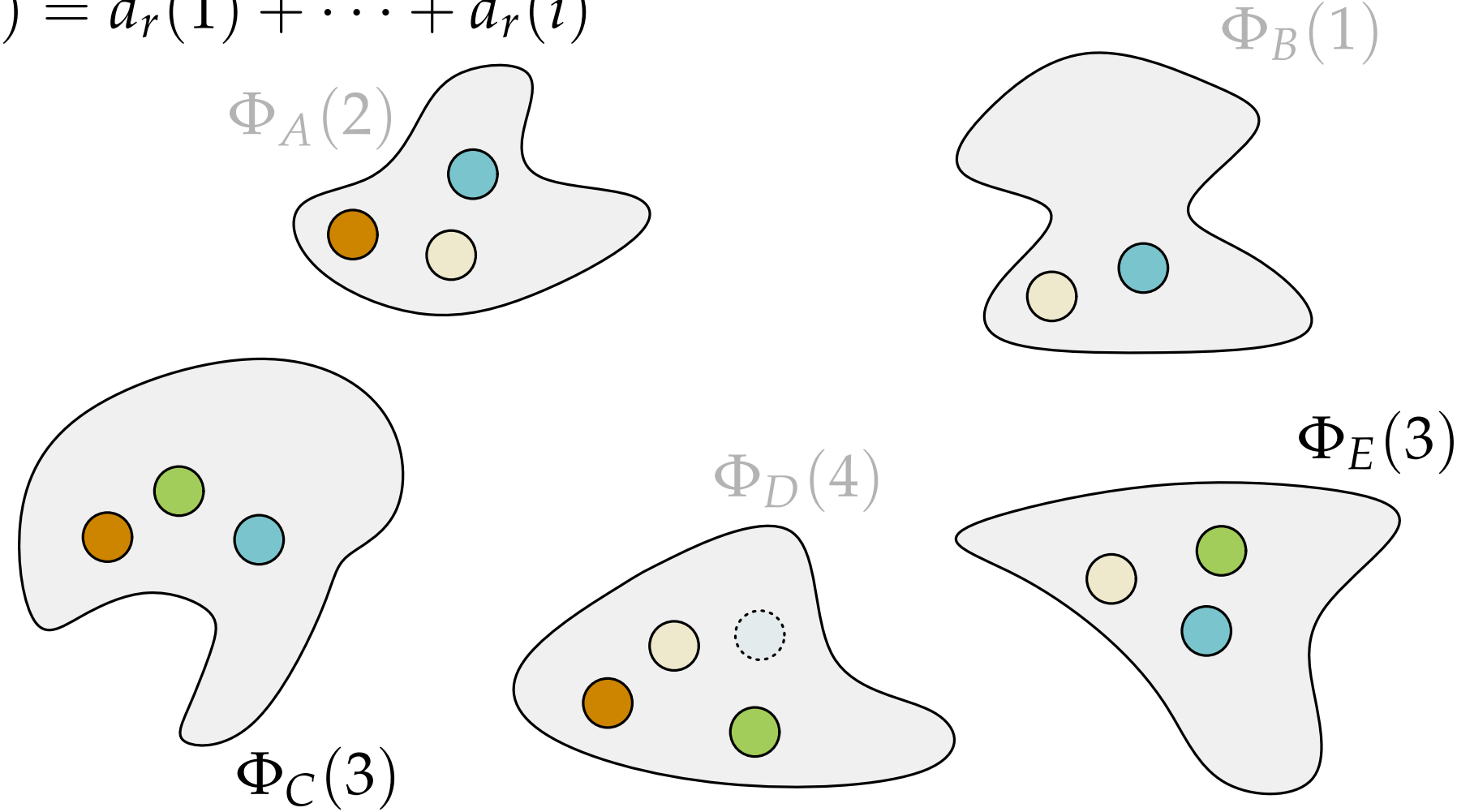
$$\Phi_r(i) = d_r(1) + \cdots + d_r(i)$$



Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Potential function 

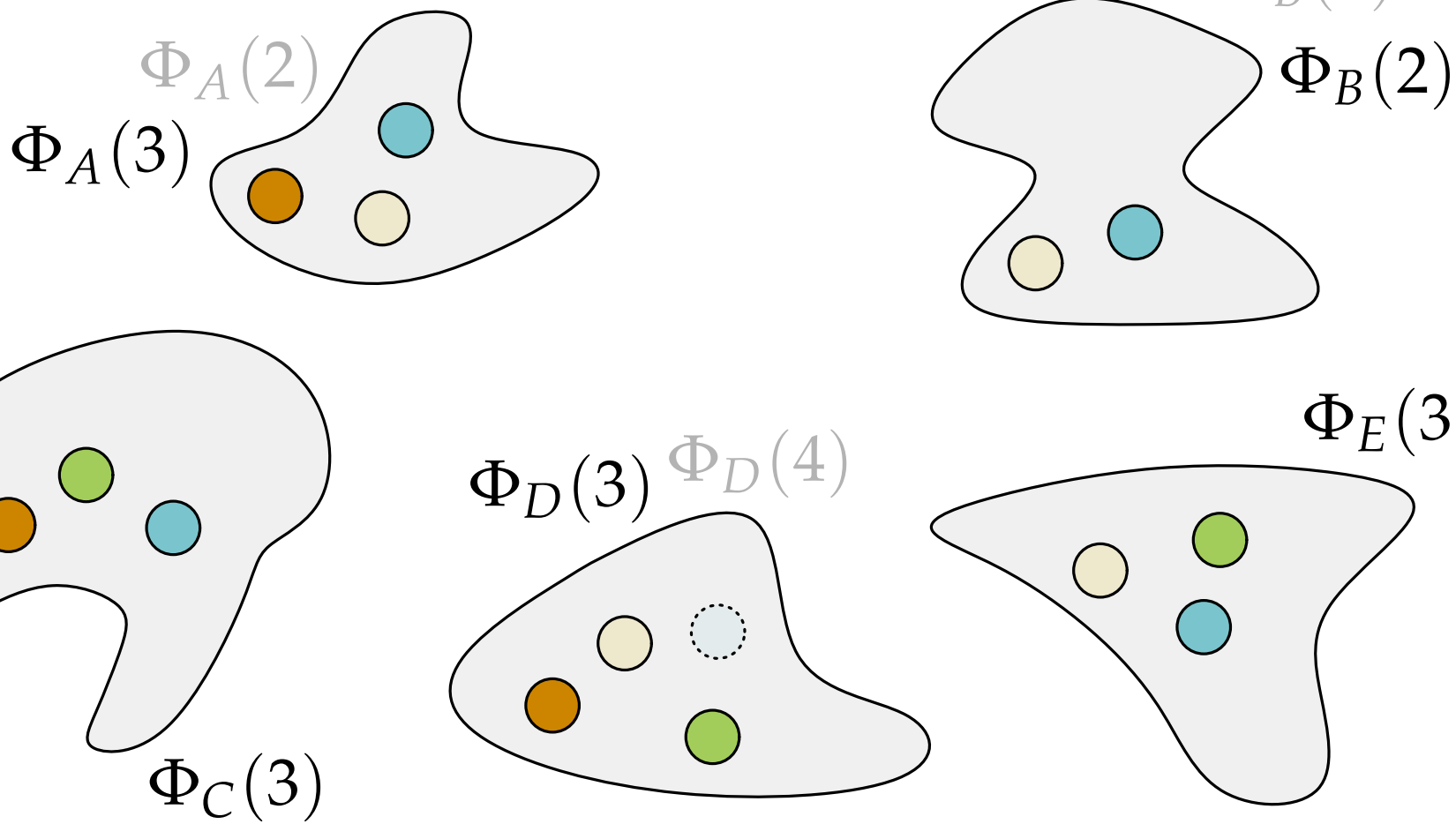
$$\Phi_r(i) = d_r(1) + \cdots + d_r(i)$$



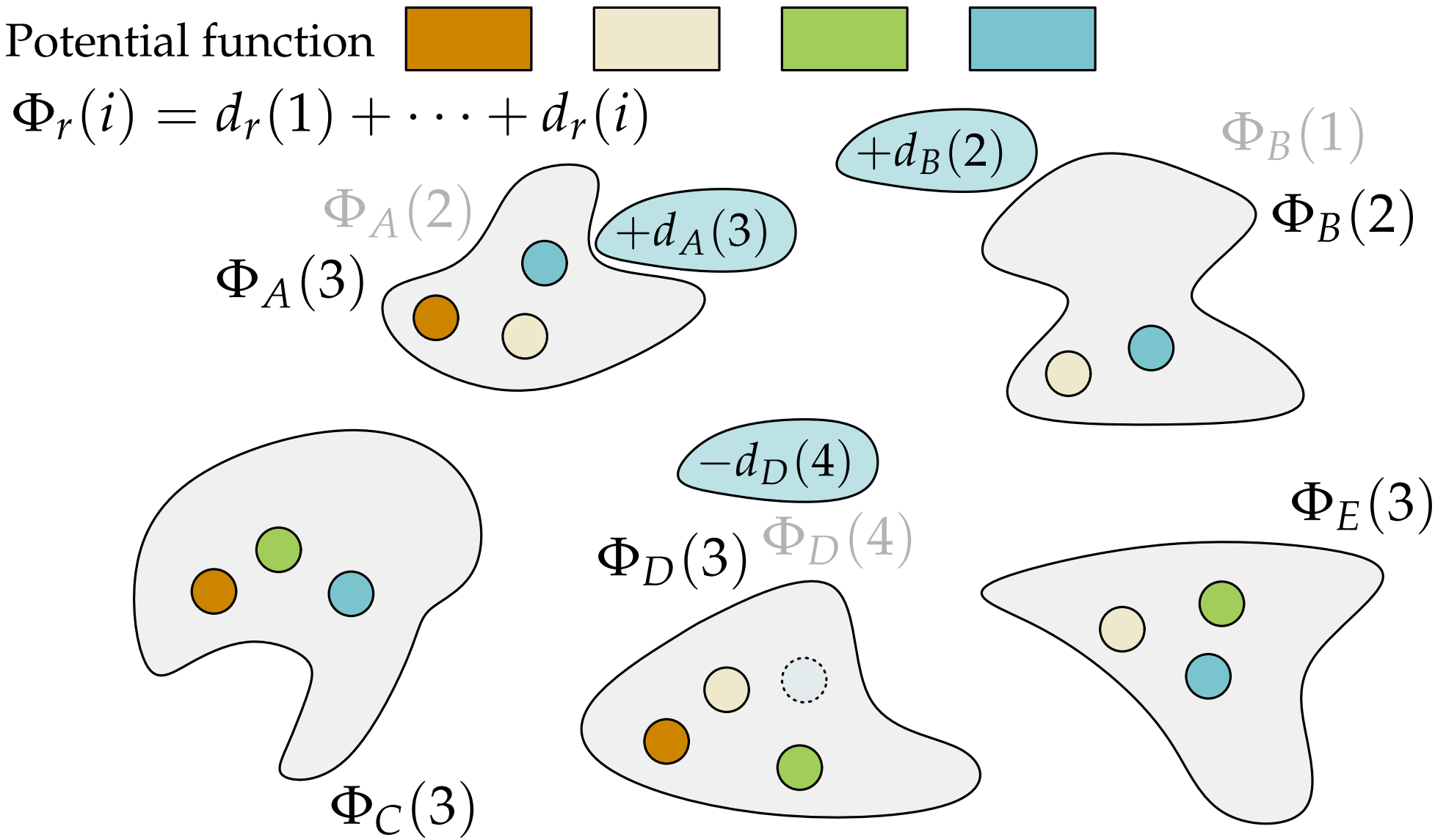
Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Potential function

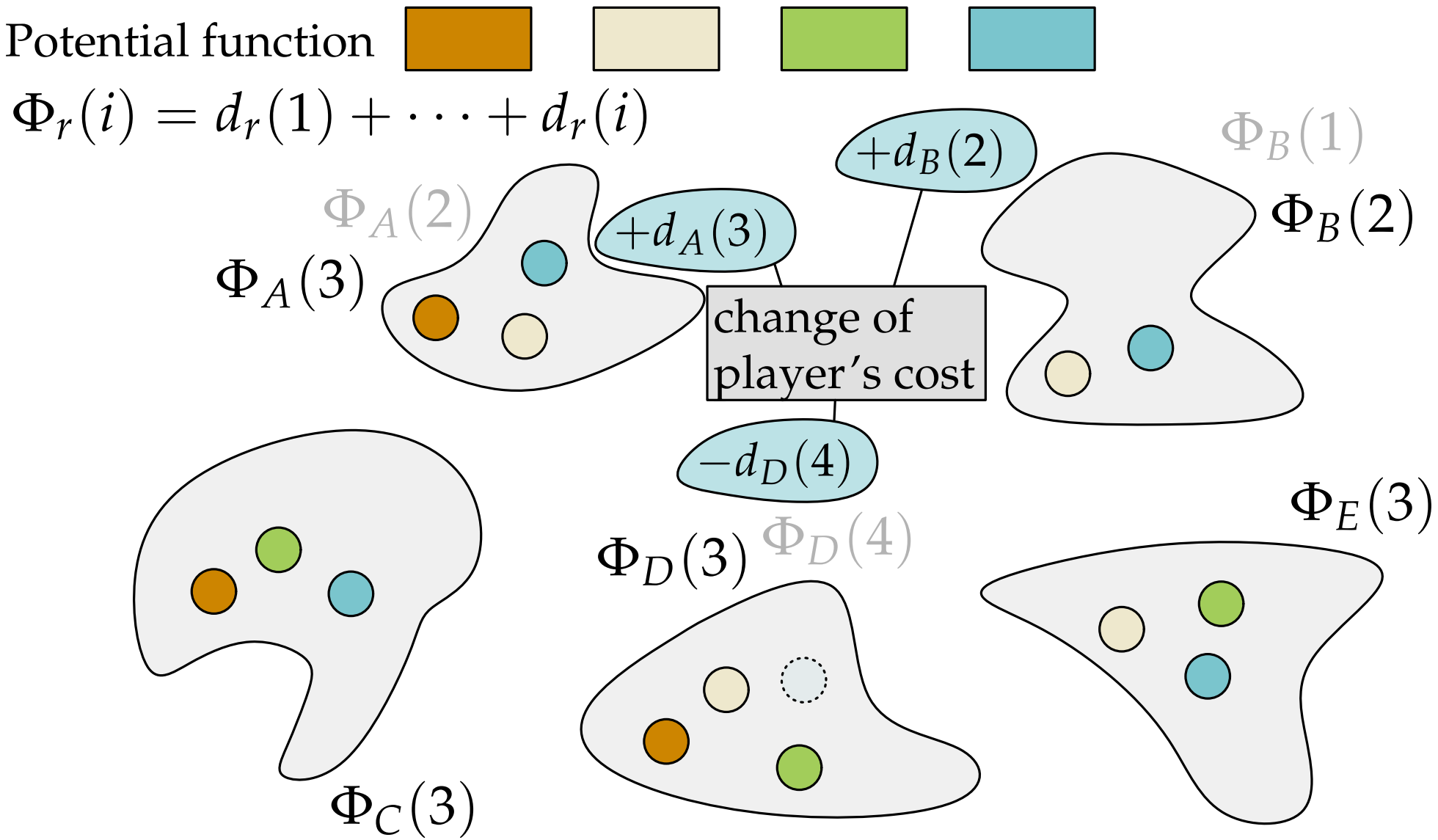
$$\Phi_r(i) = d_r(1) + \cdots + d_r(i)$$



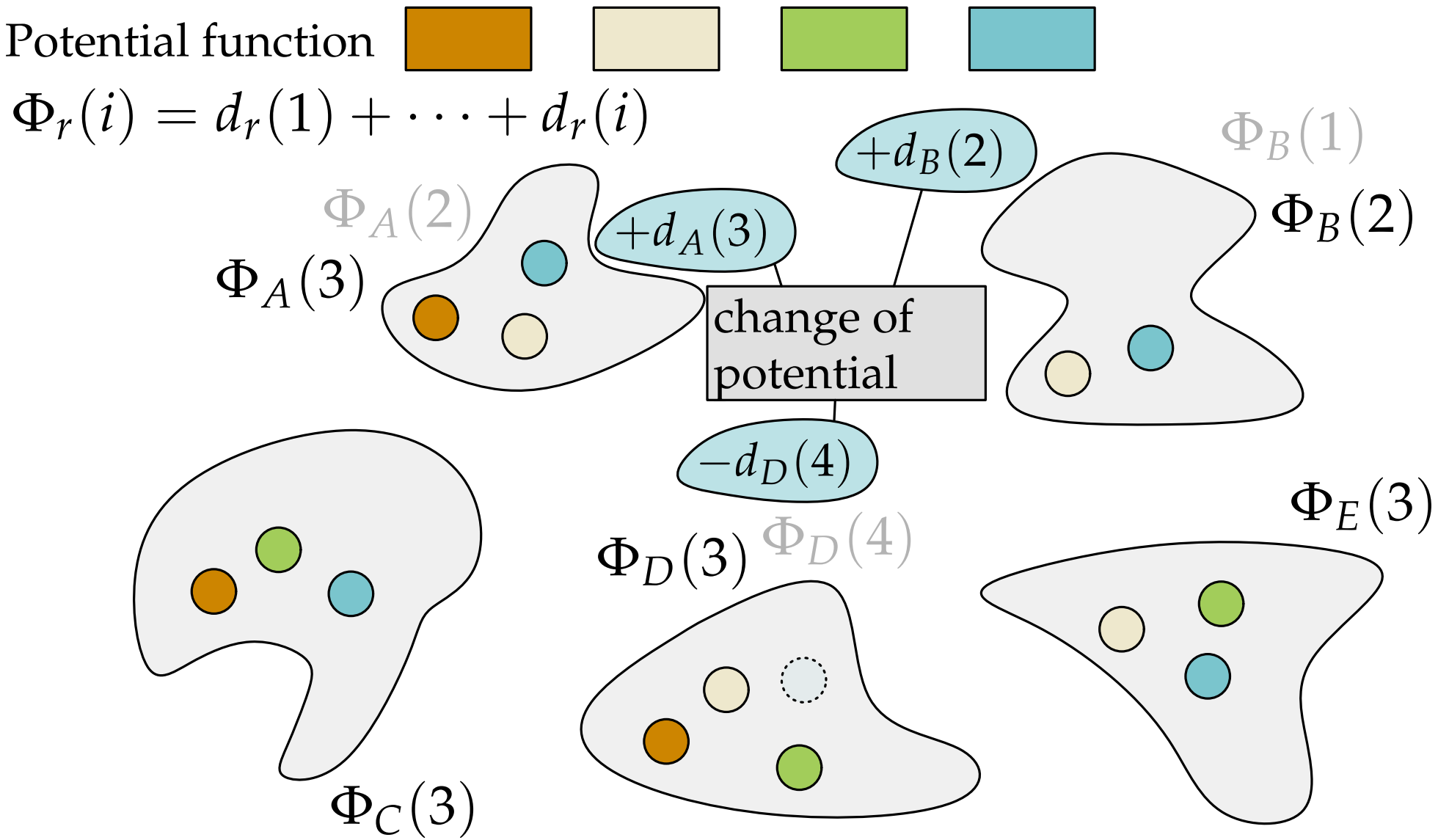
Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.



Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.



Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.



Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Potential function $\Phi_r(i) = \text{delay}_r(1) + \cdots + \text{delay}_r(i)$

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Potential function $\Phi_r(i) = \text{delay}_r(1) + \cdots + \text{delay}_r(i)$

$$\Phi(\vec{s}) = \sum_{r \in \mathcal{R}} \Phi_r(\text{load}(r, \vec{s}))$$

$$= \sum_{r \in \mathcal{R}} \sum_{i=1}^{\text{load}(r,\vec{s})} \text{delay}_r(i)$$

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Potential function $\Phi_r(i) = \text{delay}_r(1) + \cdots + \text{delay}_r(i)$

$$\Phi(\vec{s}) = \sum_{r \in \mathcal{R}} \Phi_r(\text{load}(r, \vec{s}))$$

$$= \sum_{r \in \mathcal{R}} \sum_{i=1}^{\text{load}(r,\vec{s})} \text{delay}_r(i)$$

If player p can improve by switching from $\vec{s}$ ot $\vec{s}'$, then $\Phi(\vec{s}') < \Phi(\vec{s})$.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

Potential function $\Phi_r(i) = \text{delay}_r(1) + \cdots + \text{delay}_r(i)$

$$\Phi(\vec{s}) = \sum_{r \in \mathcal{R}} \Phi_r(\text{load}(r, \vec{s}))$$

$$= \sum_{r \in \mathcal{R}} \sum_{i=1}^{\text{load}(r,\vec{s})} \text{delay}_r(i)$$

If player p can improve by switching from $\vec{s}$ ot $\vec{s}'$, then $\Phi(\vec{s}') < \Phi(\vec{s})$.

A profile $\vec{s}$ minimizing $\Phi(\vec{s})$ is a Nash equilibrium.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

A profile $\vec{s}$ minimizing $\Phi(\vec{s})$ is a Nash equilibrium.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

A profile $\vec{s}$ minimizing $\Phi(\vec{s})$ is a Nash equilibrium.

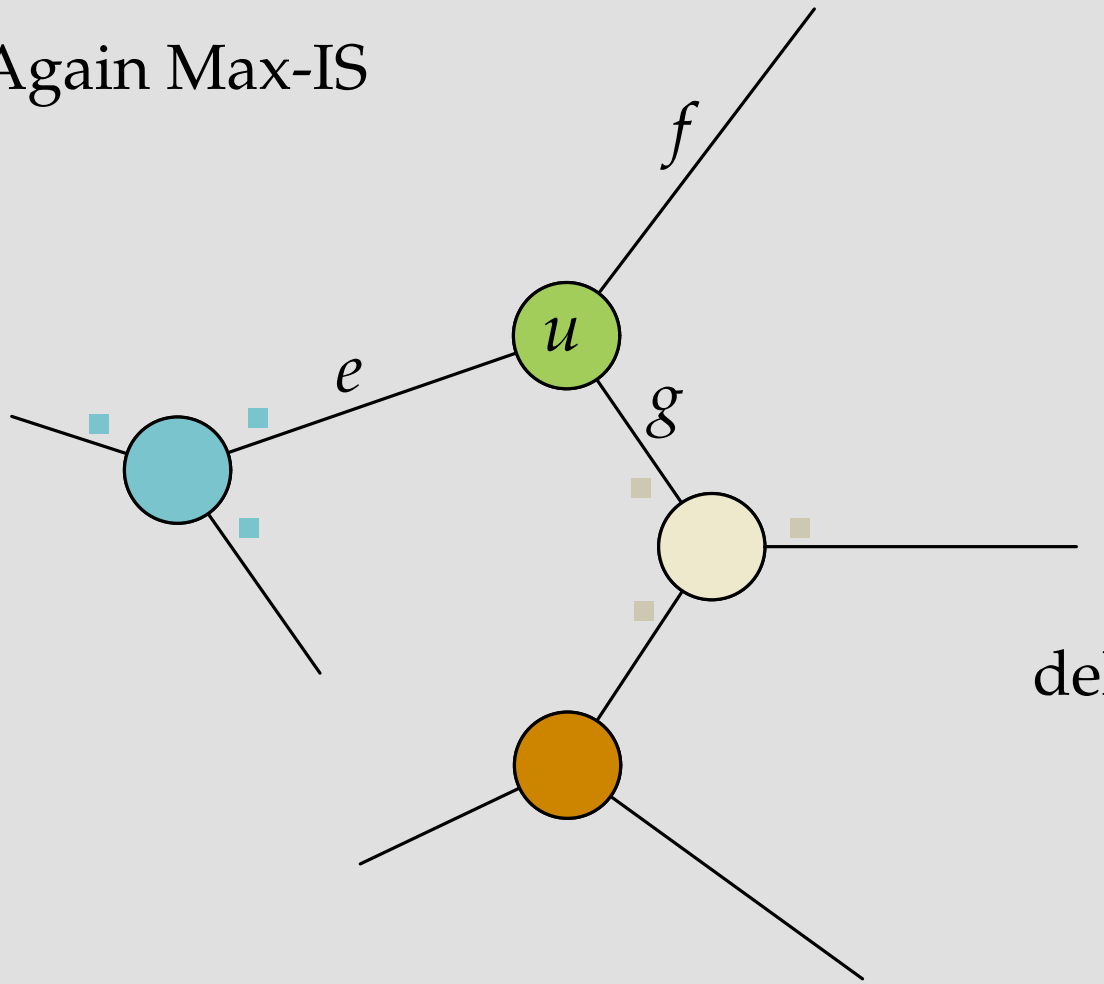
How hard to find such a minimum?

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

A profile $\vec{s}$ minimizing $\Phi(\vec{s})$ is a Nash equilibrium.

How hard to find such a minimum?

Again Max-IS



social cost = 2

$\text{delay}(\text{DNB}, i) = 1$

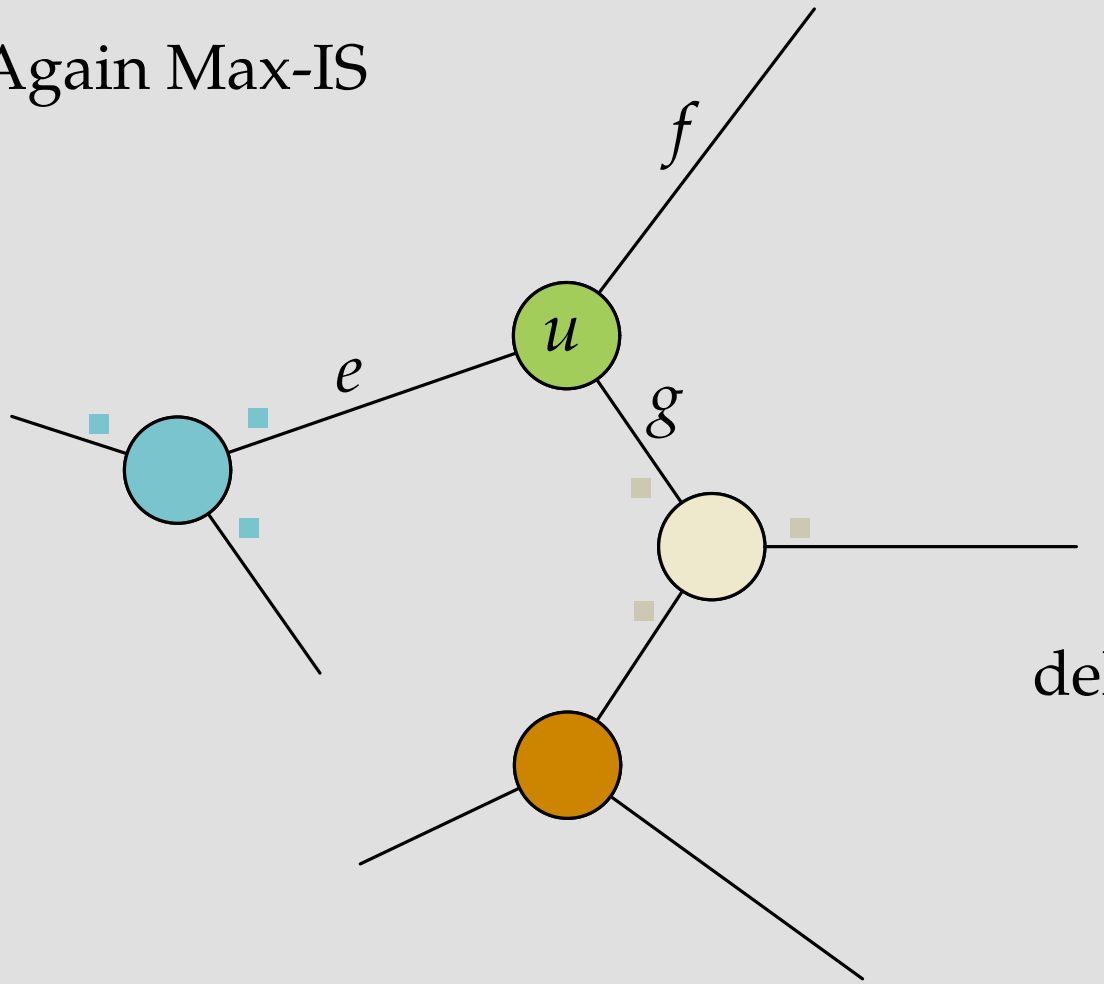
$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

A profile $\vec{s}$ minimizing $\Phi(\vec{s})$ is a Nash equilibrium.

How hard to find such a minimum?

Again Max-IS



potential = 2
social cost = 2

$\text{delay}(\text{DNB}, i) = 1$

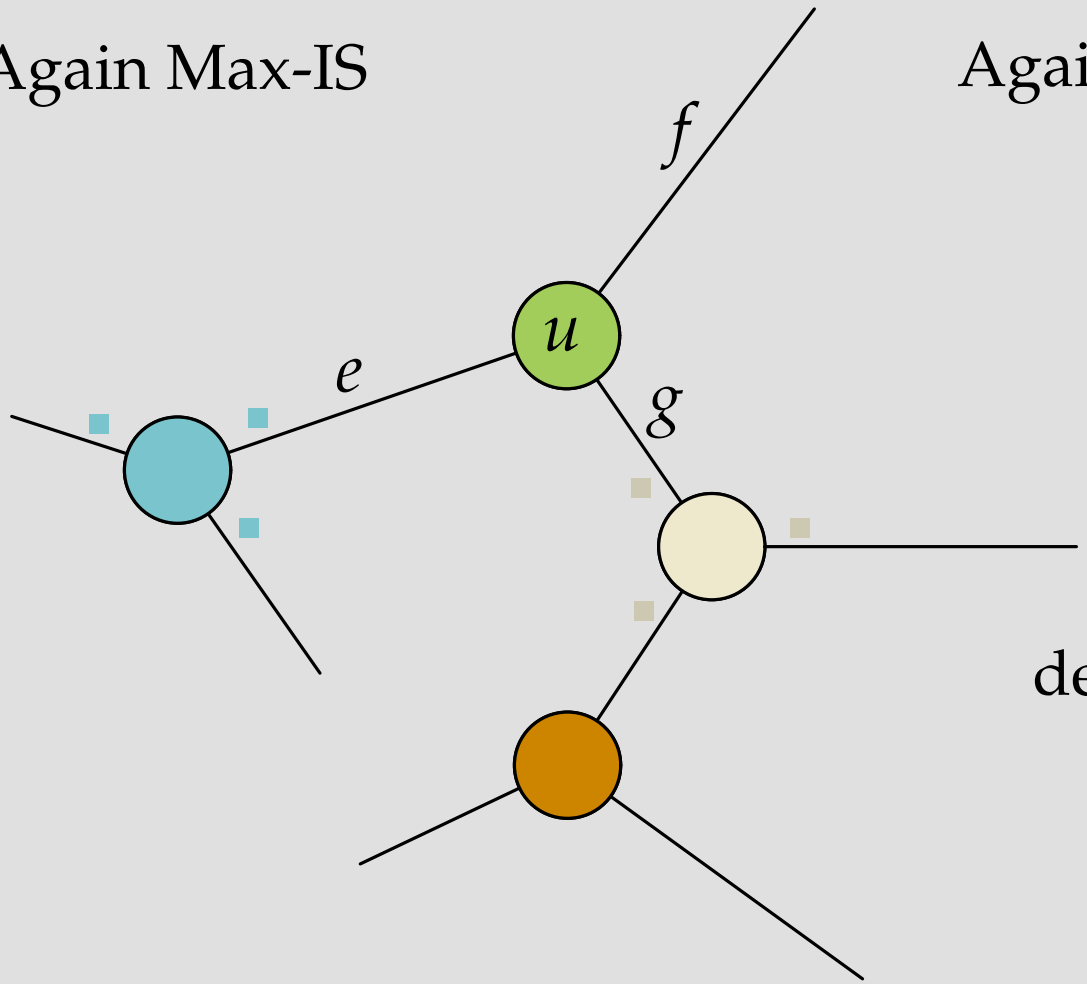
$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

A profile $\vec{s}$ minimizing $\Phi(\vec{s})$ is a Nash equilibrium.

How hard to find such a minimum?

Again Max-IS



Again NP-hard

potential = 2
social cost = 2

$\text{delay}(\text{DNB}, i) = 1$

$$\text{delay}(e, i) = \begin{cases} 0 & \text{if } i \leq 1 \\ \infty & \text{if } i \geq 2\,. \end{cases}$$

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

A profile $\vec{s}$ minimizing $\Phi(\vec{s})$ is a Nash equilibrium.

How hard to find such a minimum? It's NP-hard.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

A profile $\vec{s}$ minimizing $\Phi(\vec{s})$ is a Nash equilibrium.

How hard to find such a minimum? It's NP-hard.

A *local minimum* $\vec{s}$ of $\Phi(\vec{s})$ is also a Nash equilibrium.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

A profile $\vec{s}$ minimizing $\Phi(\vec{s})$ is a Nash equilibrium.

How hard to find such a minimum? It's NP-hard.

A *local minimum* $\vec{s}$ of $\Phi(\vec{s})$ is also a Nash equilibrium.

Question. How hard is it to find a local minimum of such a function?

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

A profile $\vec{s}$ minimizing $\Phi(\vec{s})$ is a Nash equilibrium.

How hard to find such a minimum? It's NP-hard.

A *local minimum* $\vec{s}$ of $\Phi(\vec{s})$ is also a Nash equilibrium.

Question. How hard is it to find a local minimum of such a function?

Informal Definition. PLS (Polynomail Local Search) is the complexity class of total search problems that can be phrased as looking for a local minimum of a function.

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.

A profile $\vec{s}$ minimizing $\Phi(\vec{s})$ is a Nash equilibrium.

How hard to find such a minimum? It's NP-hard.

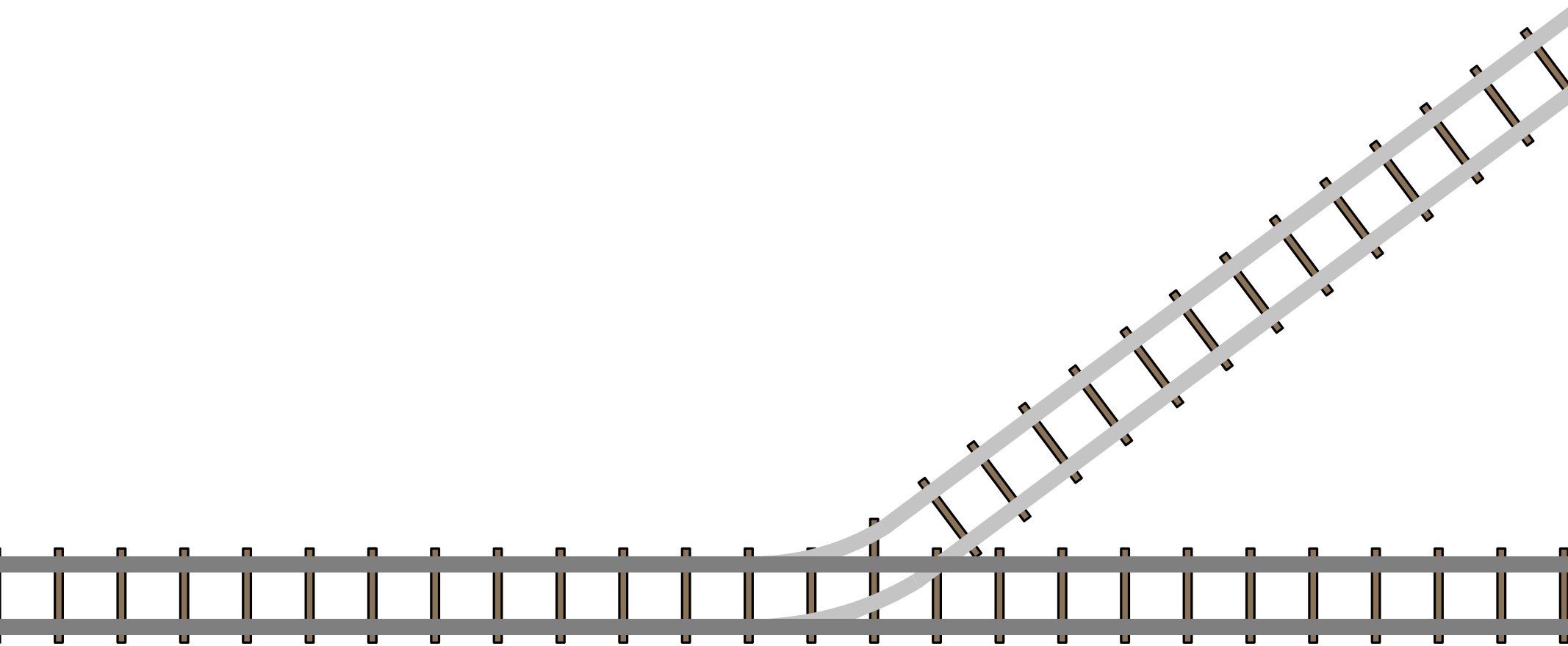
A *local minimum* $\vec{s}$ of $\Phi(\vec{s})$ is also a Nash equilibrium.

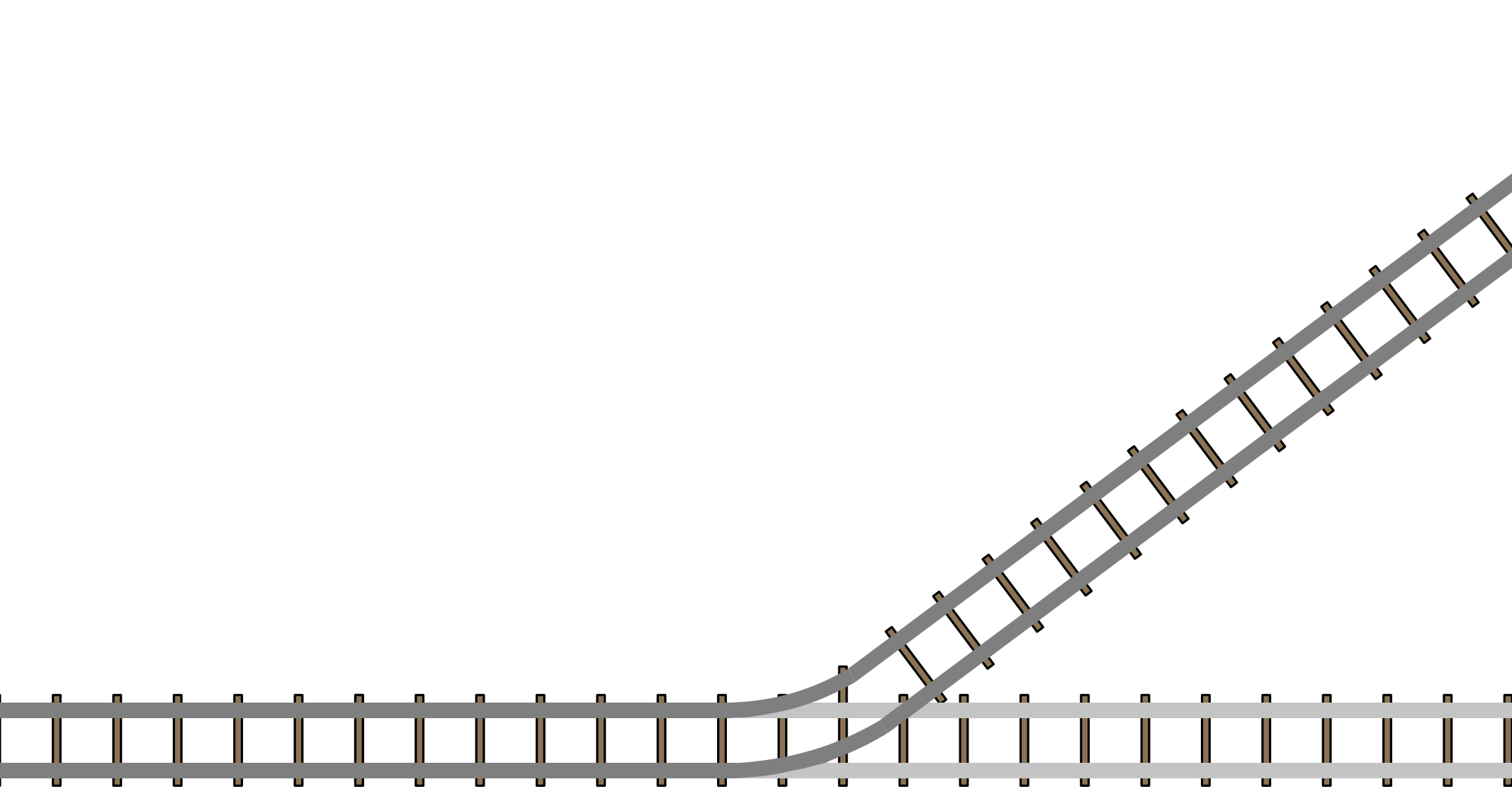
Question. How hard is it to find a local minimum of such a function?

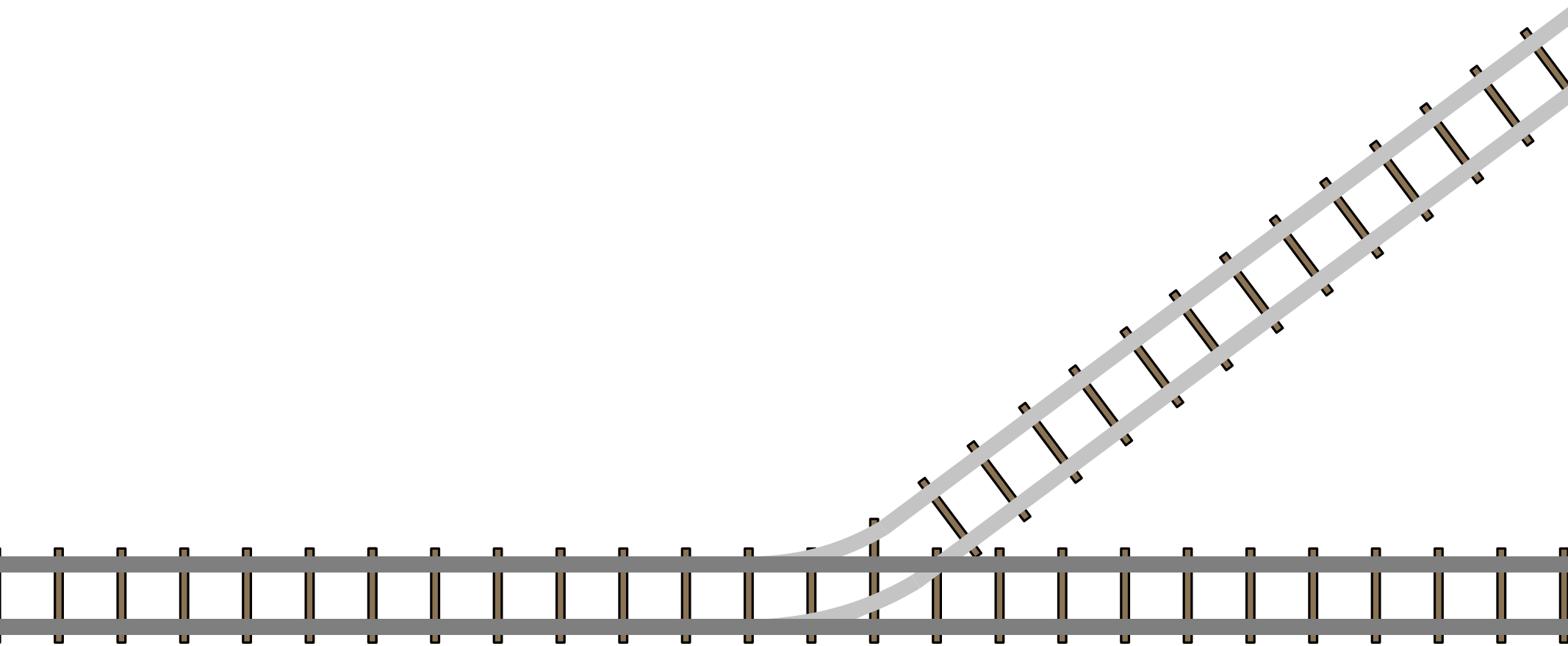
Informal Definition. PLS (Polynomail Local Search) is the complexity class of total search problems that can be phrased as looking for a local minimum of a function.

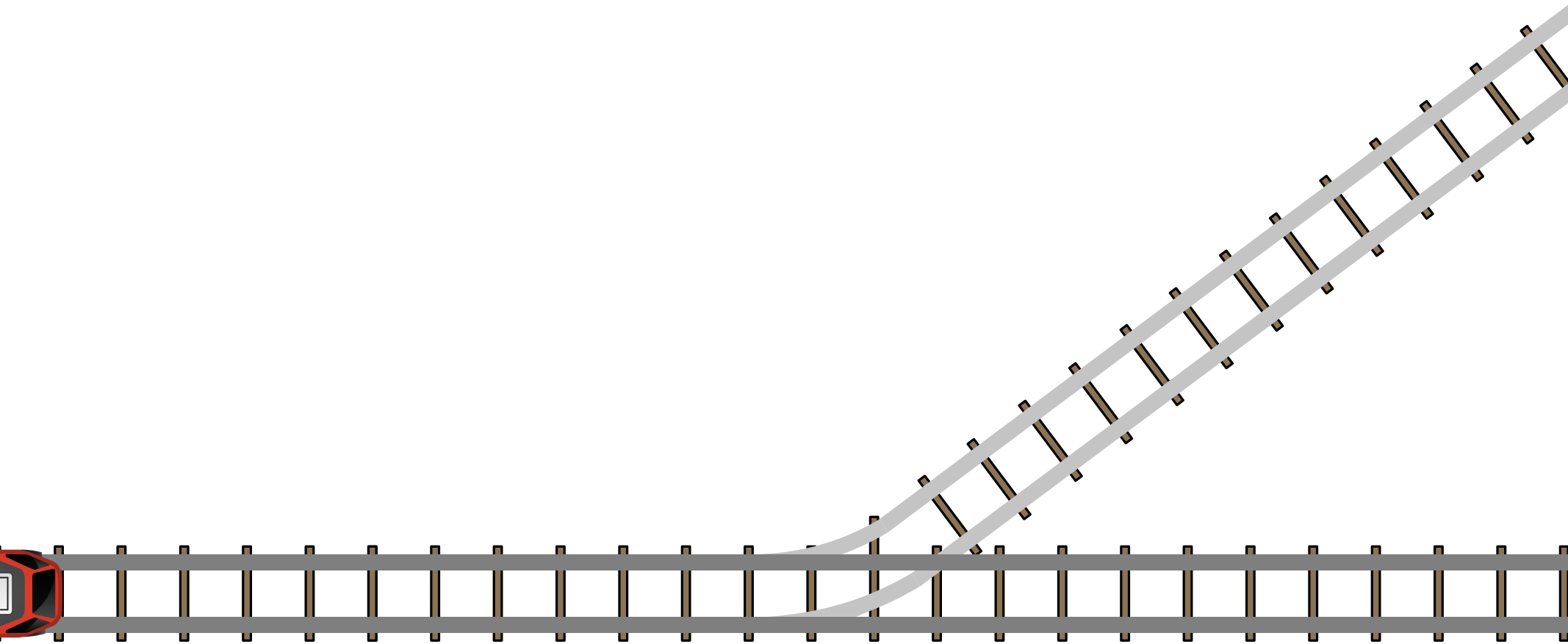
Arrival

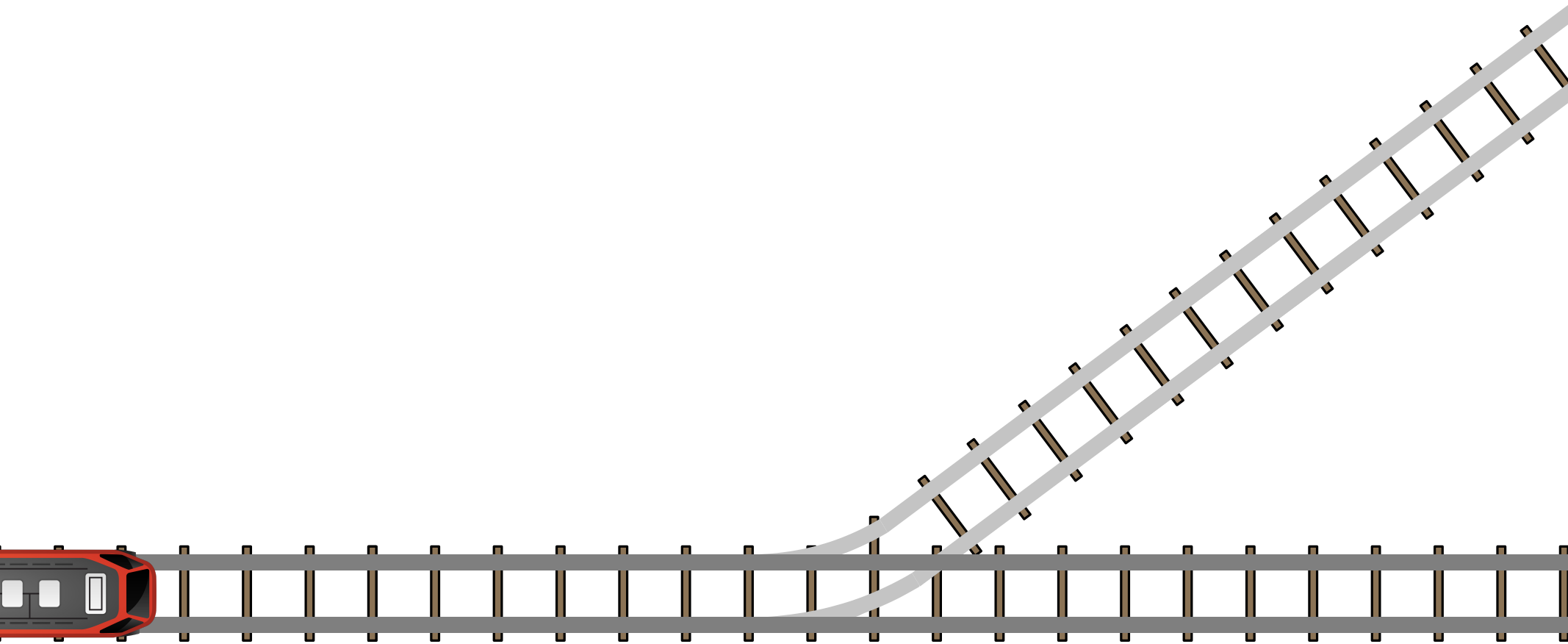
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)

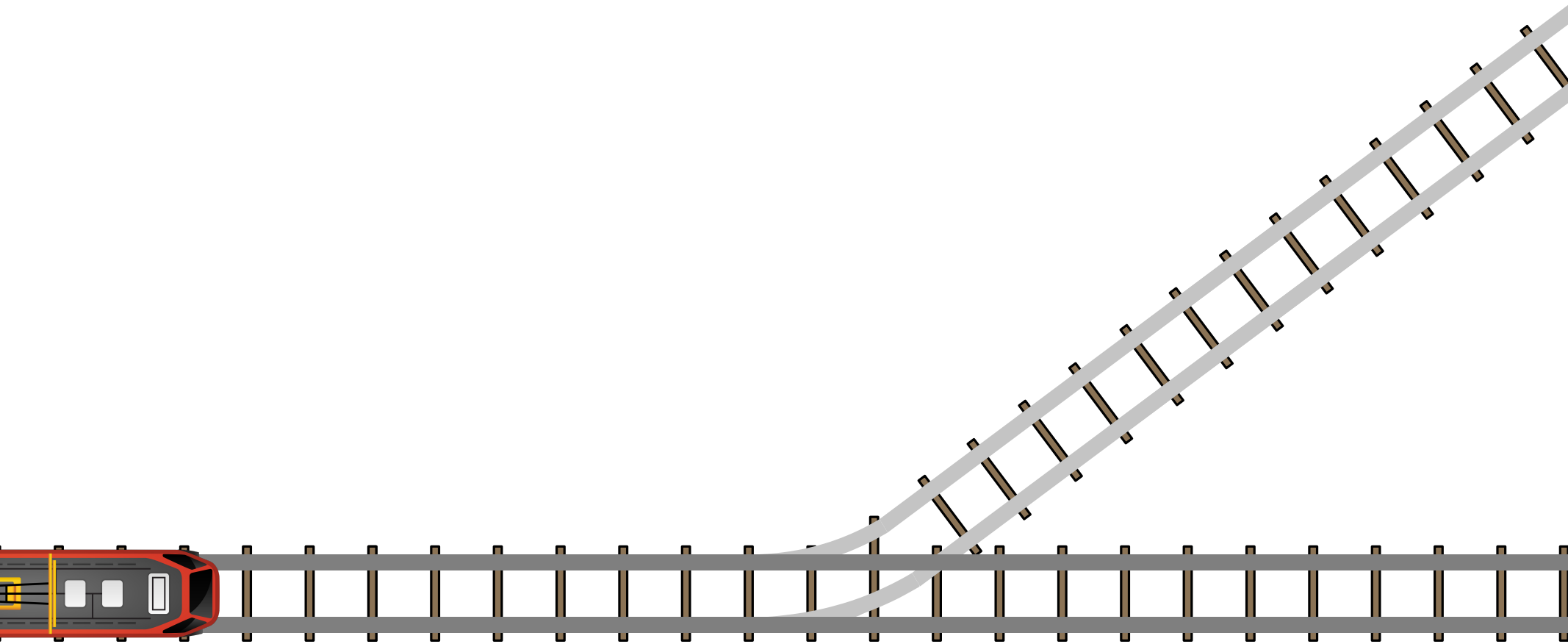


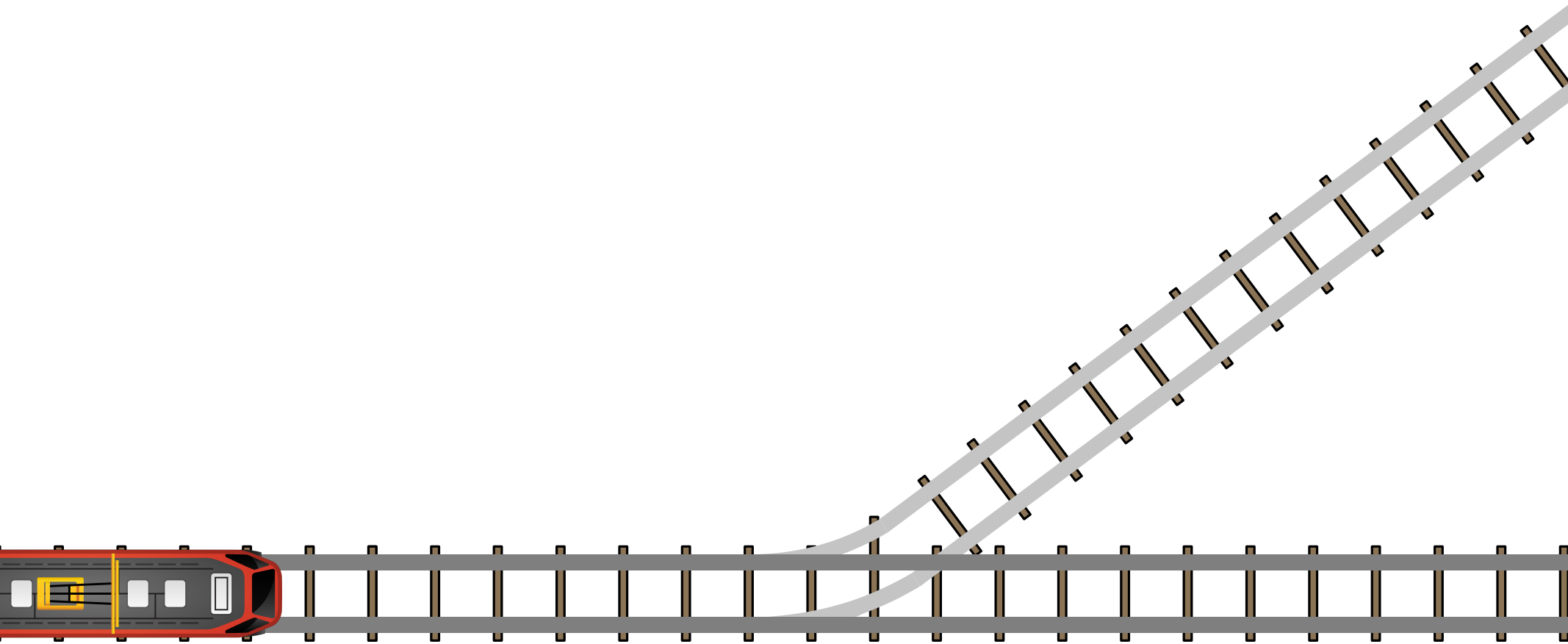


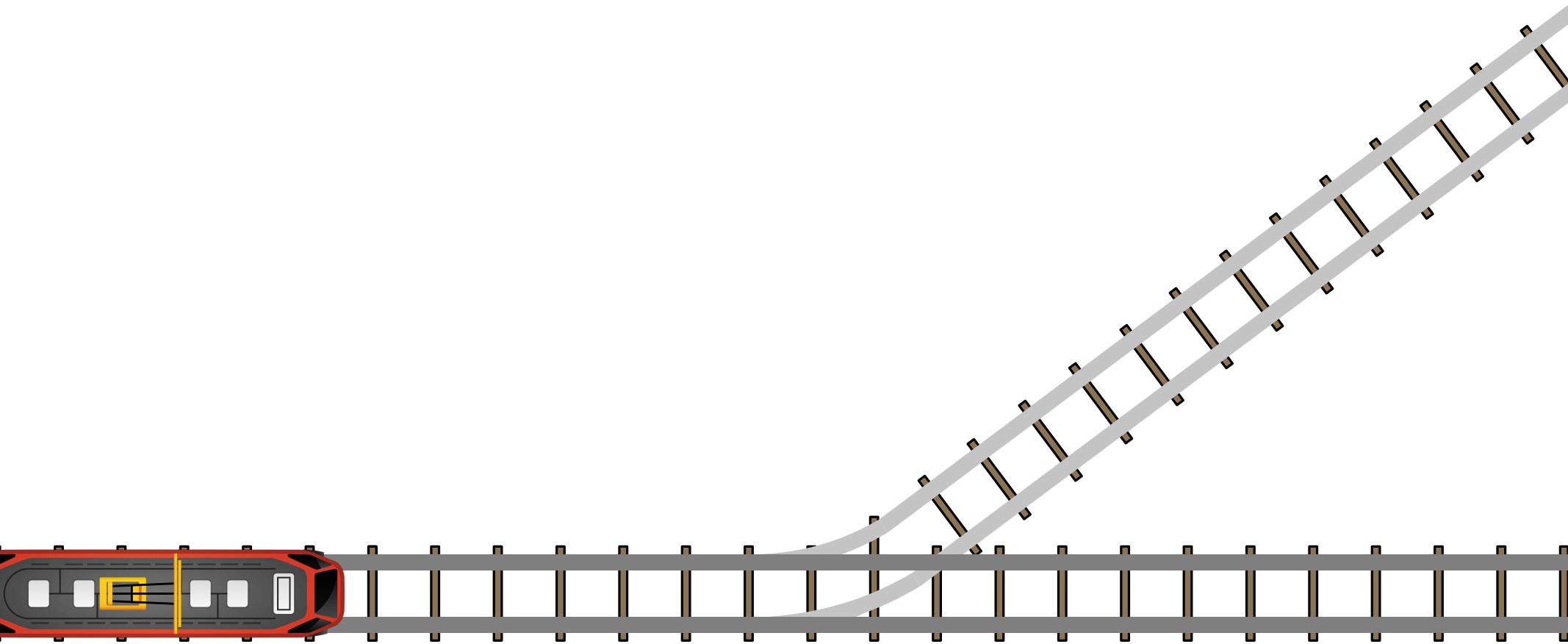


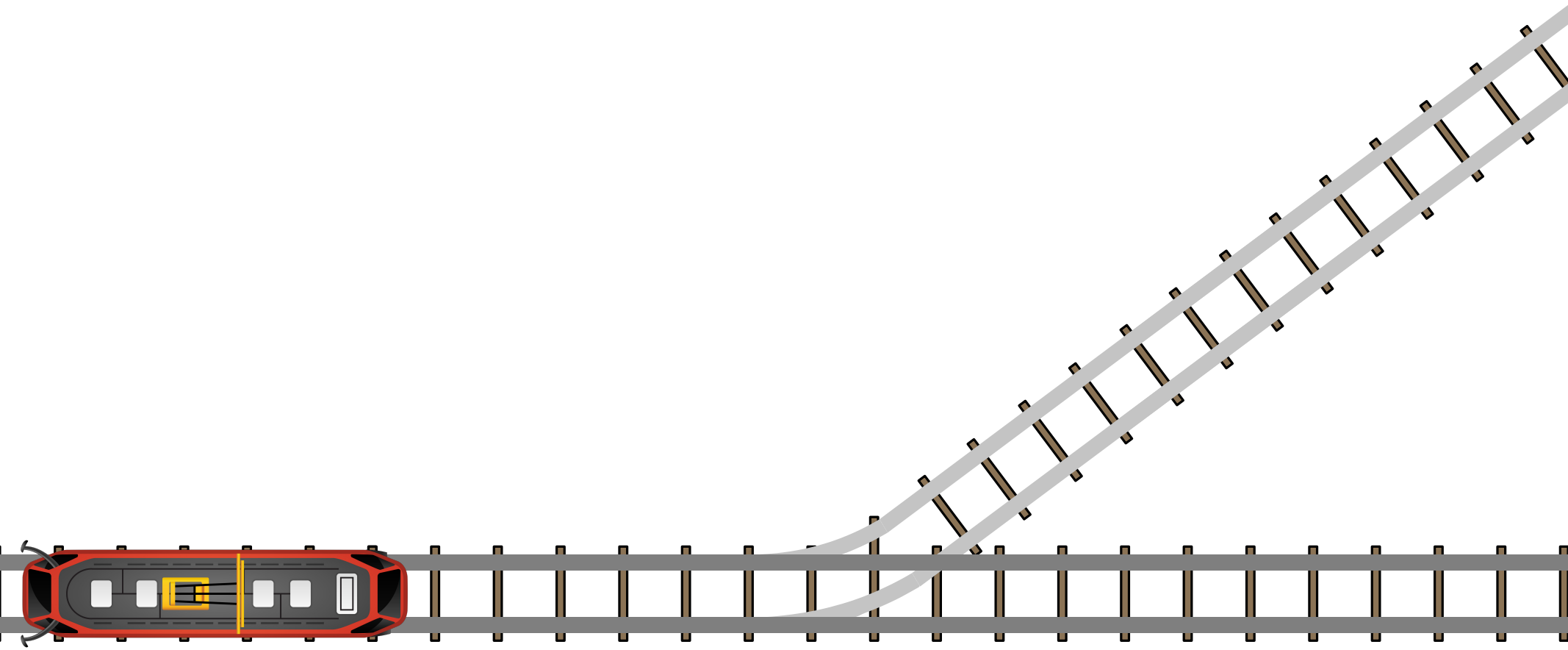


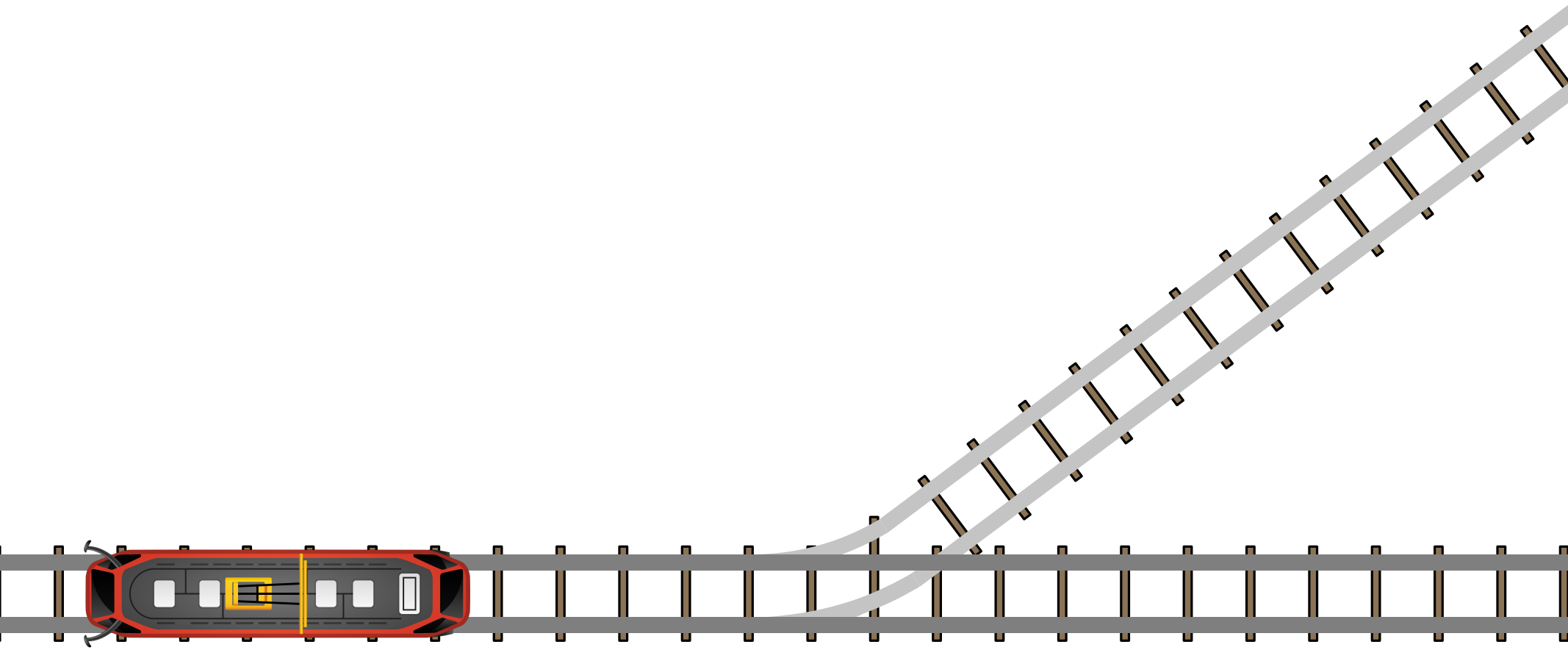


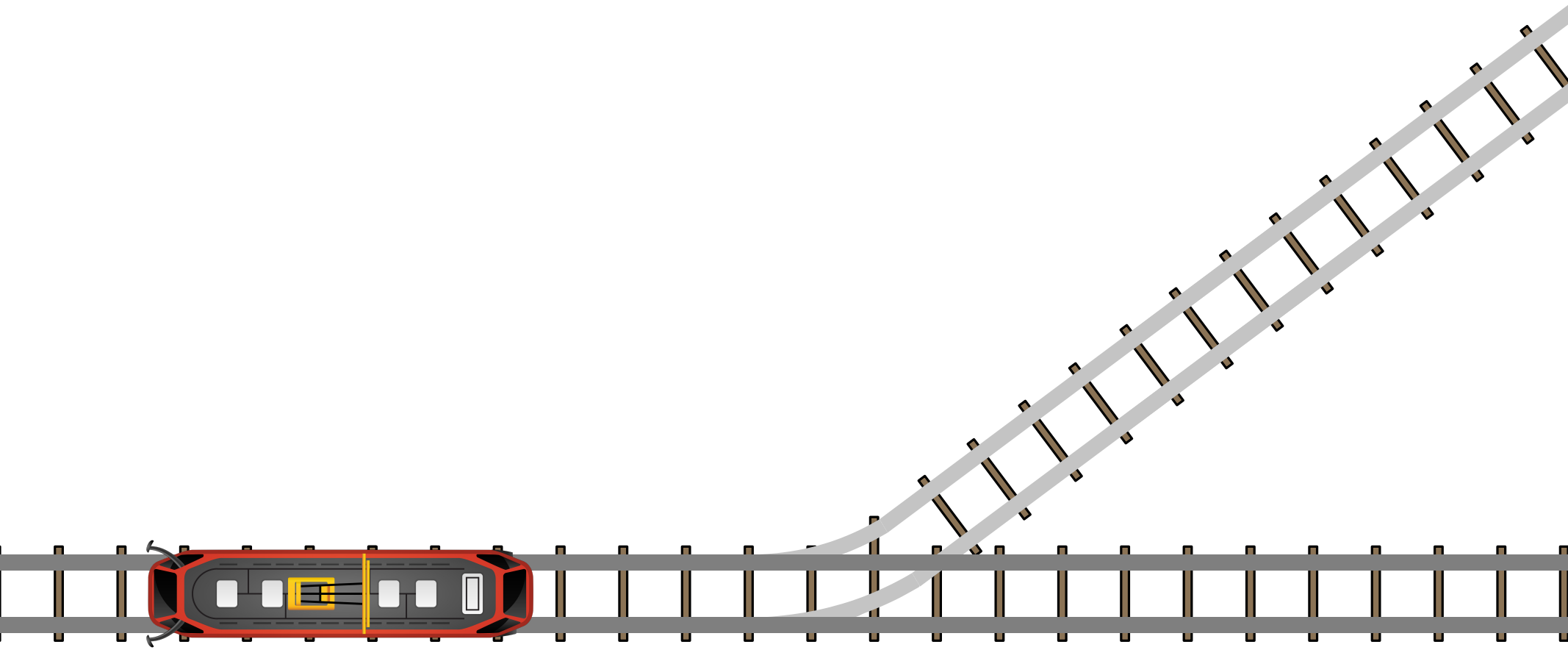




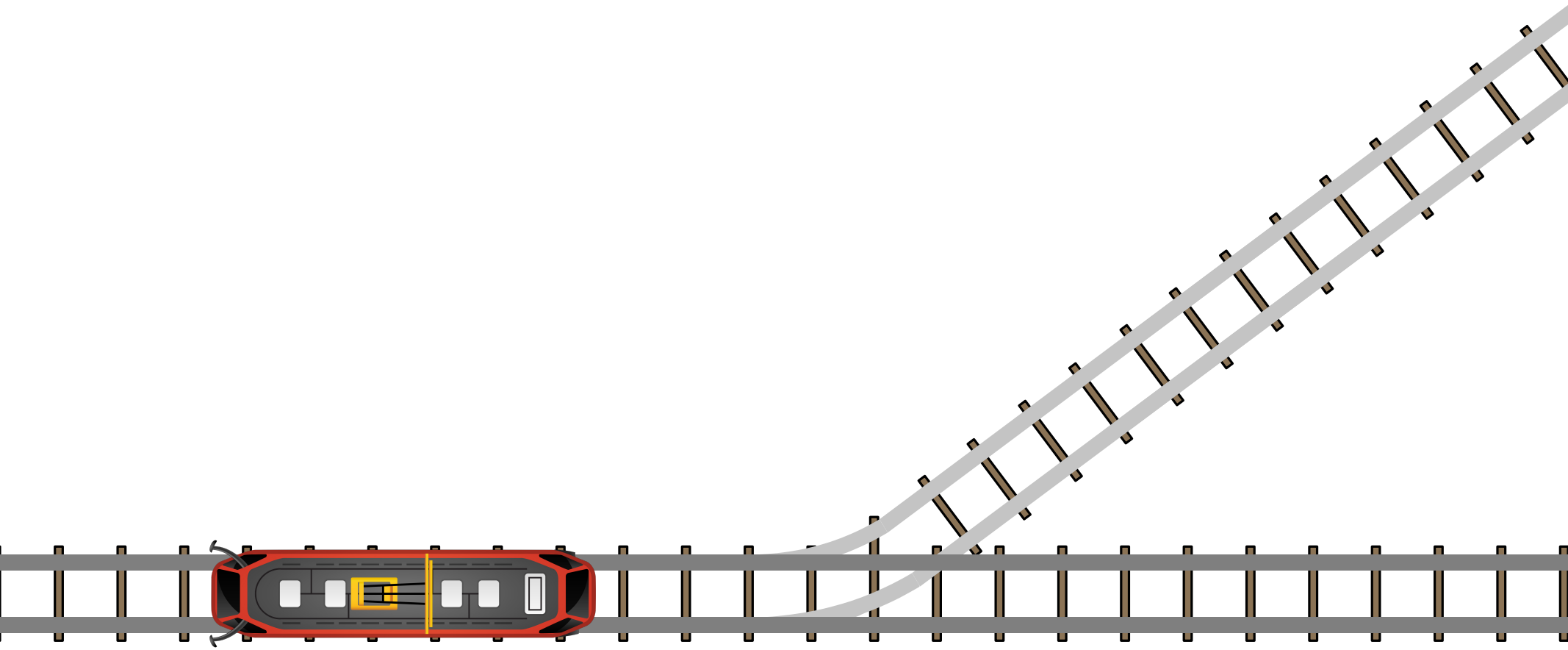




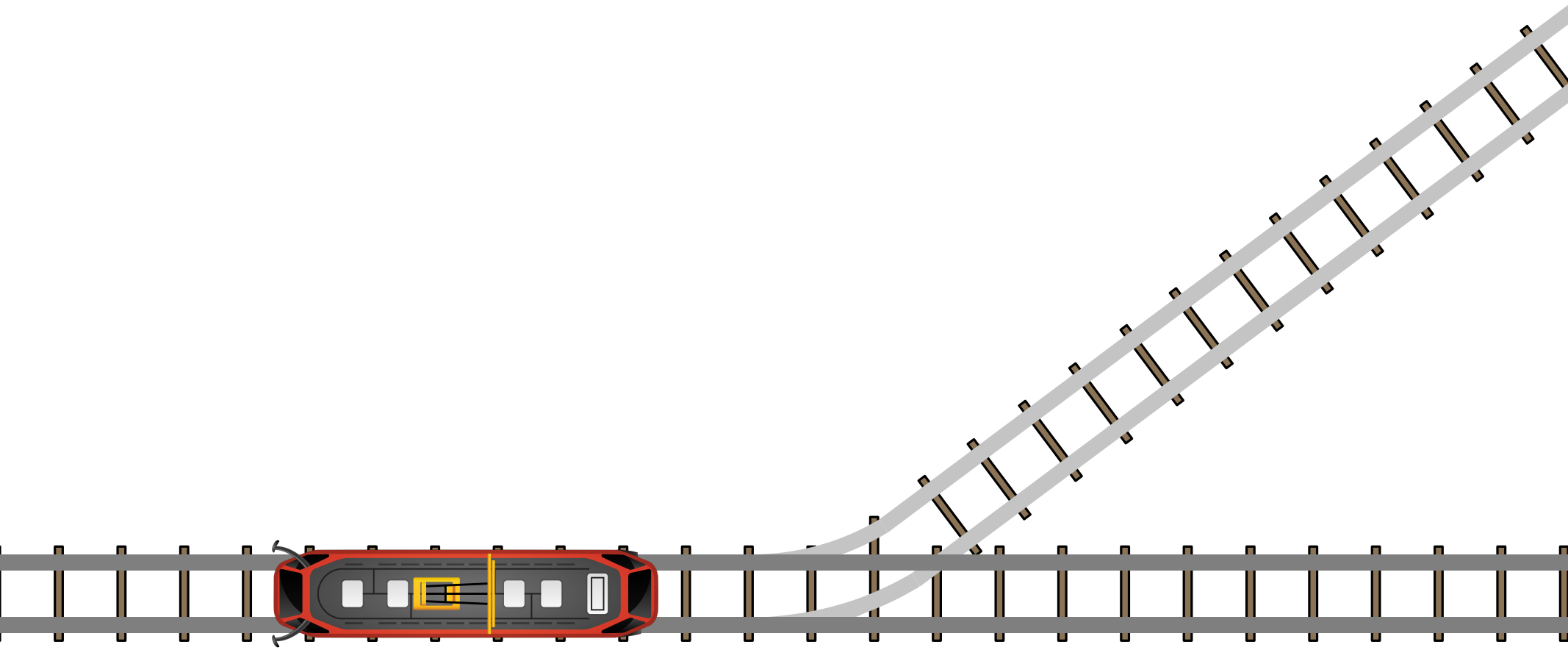




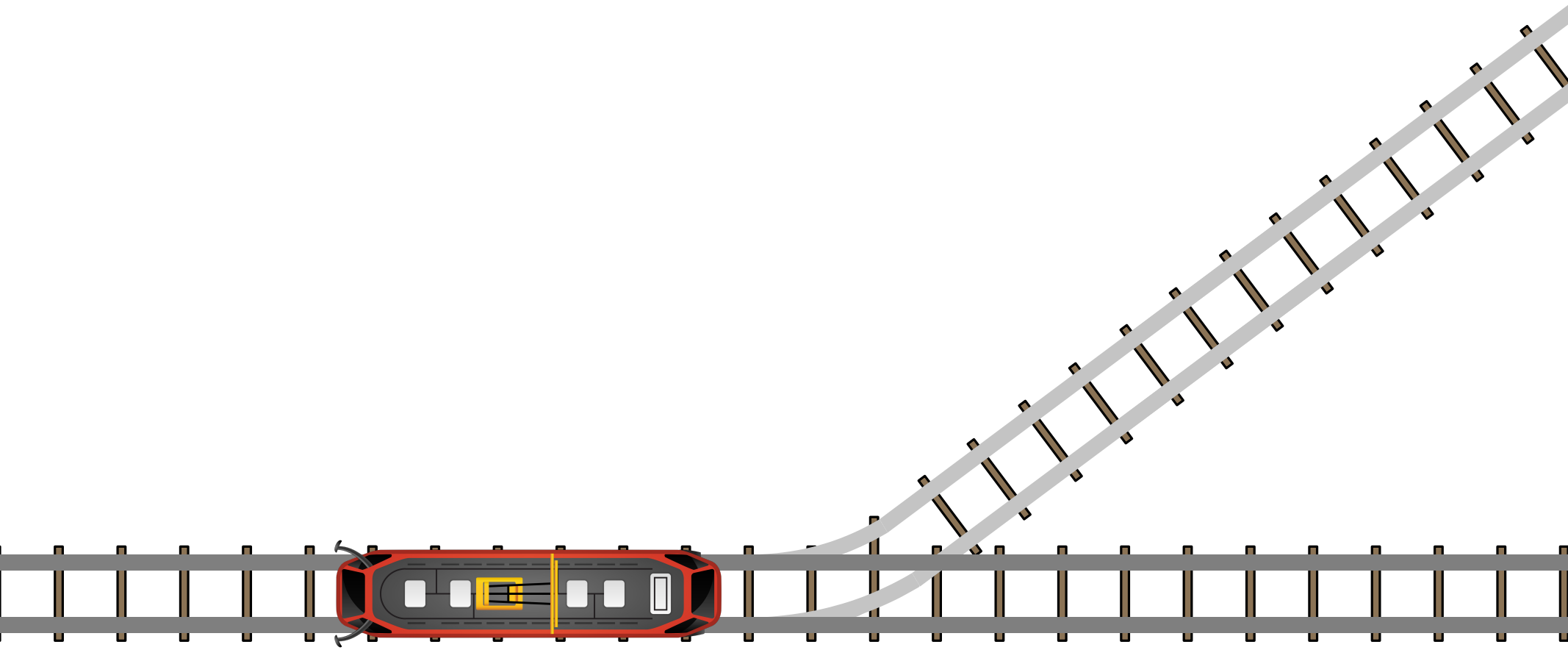
Designed by macrovector / Freepik



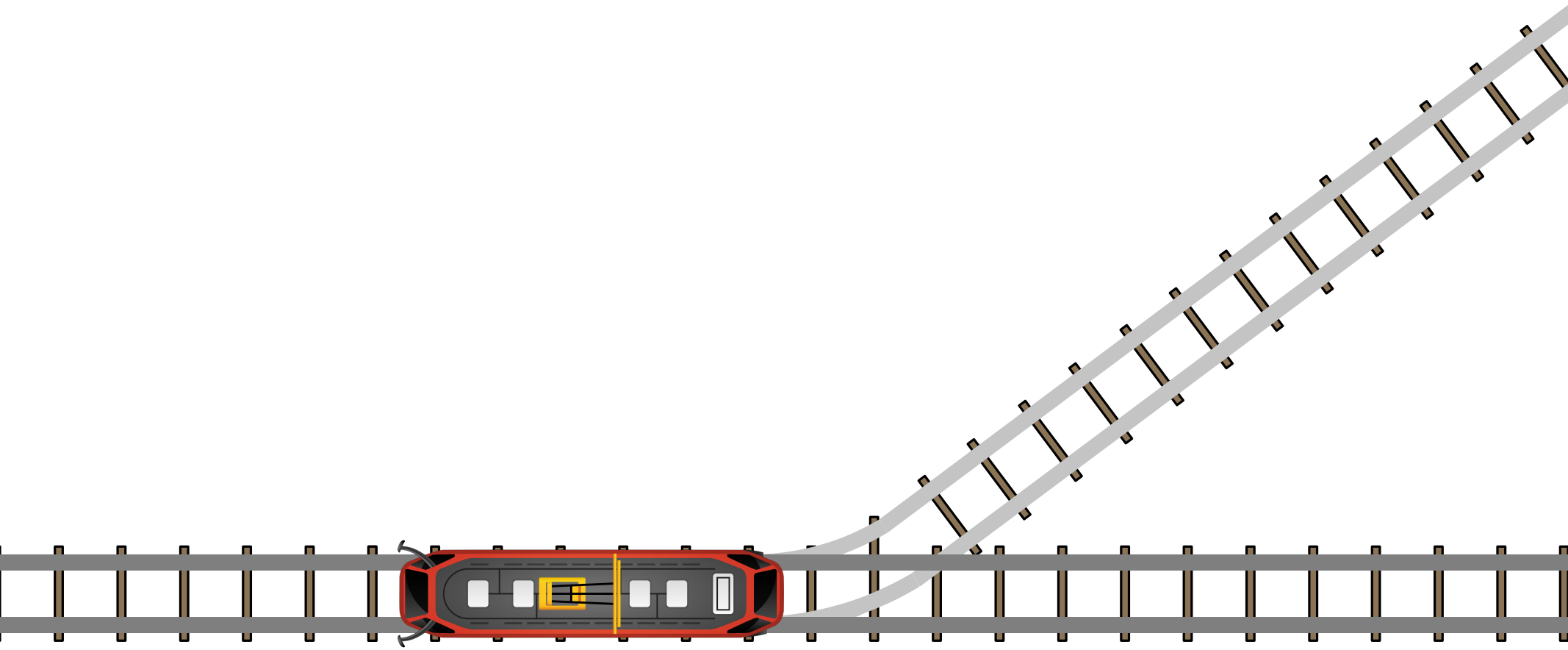
Designed by macrovector / Freepik



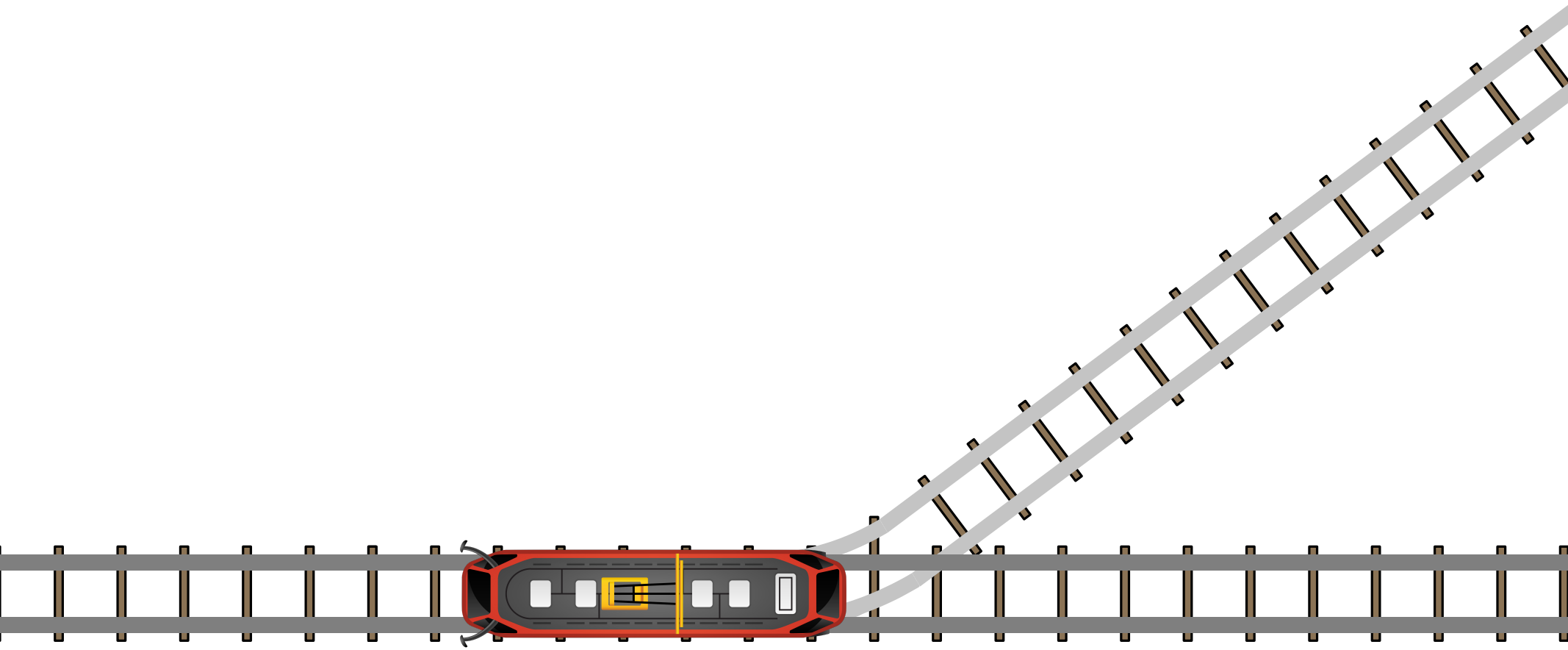
Designed by macrovector / Freepik



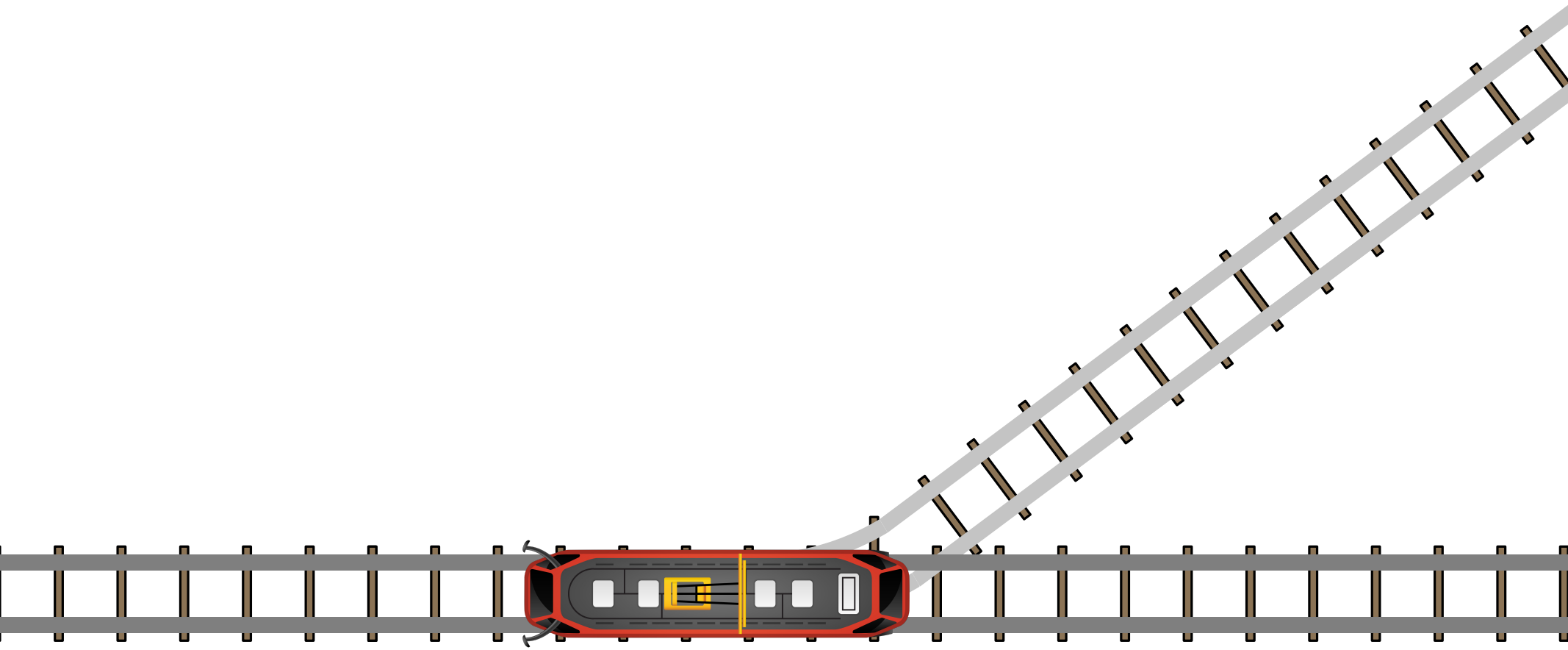
Designed by macrovector / Freepik



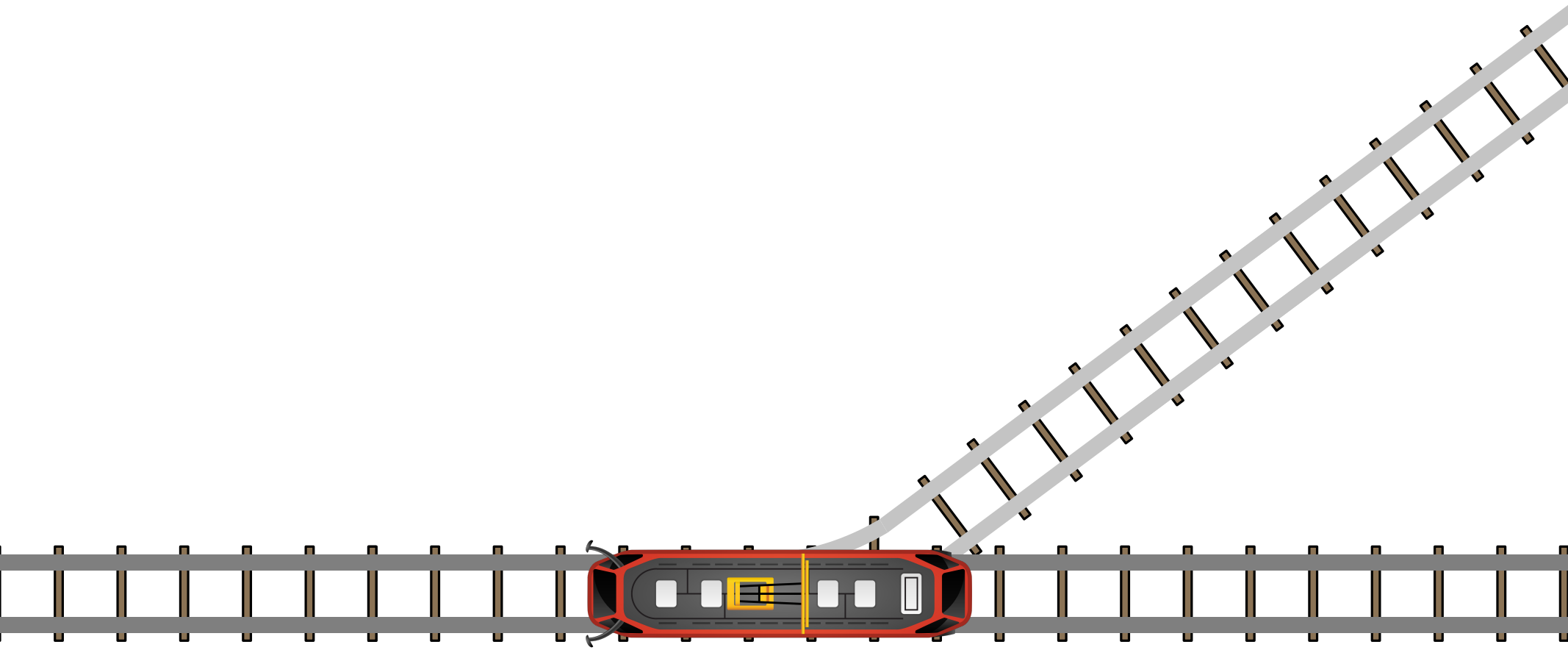
Designed by macrovector / Freepik



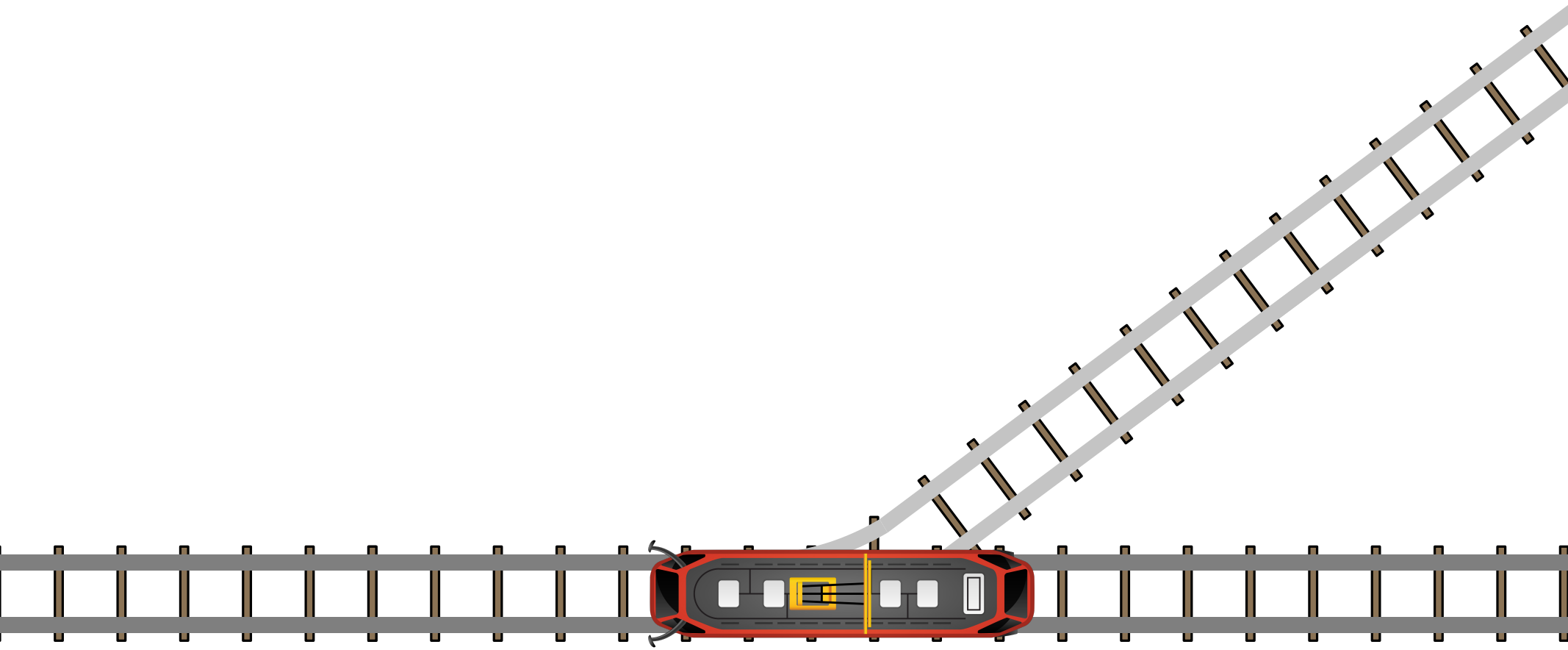
Designed by macrovector / Freepik



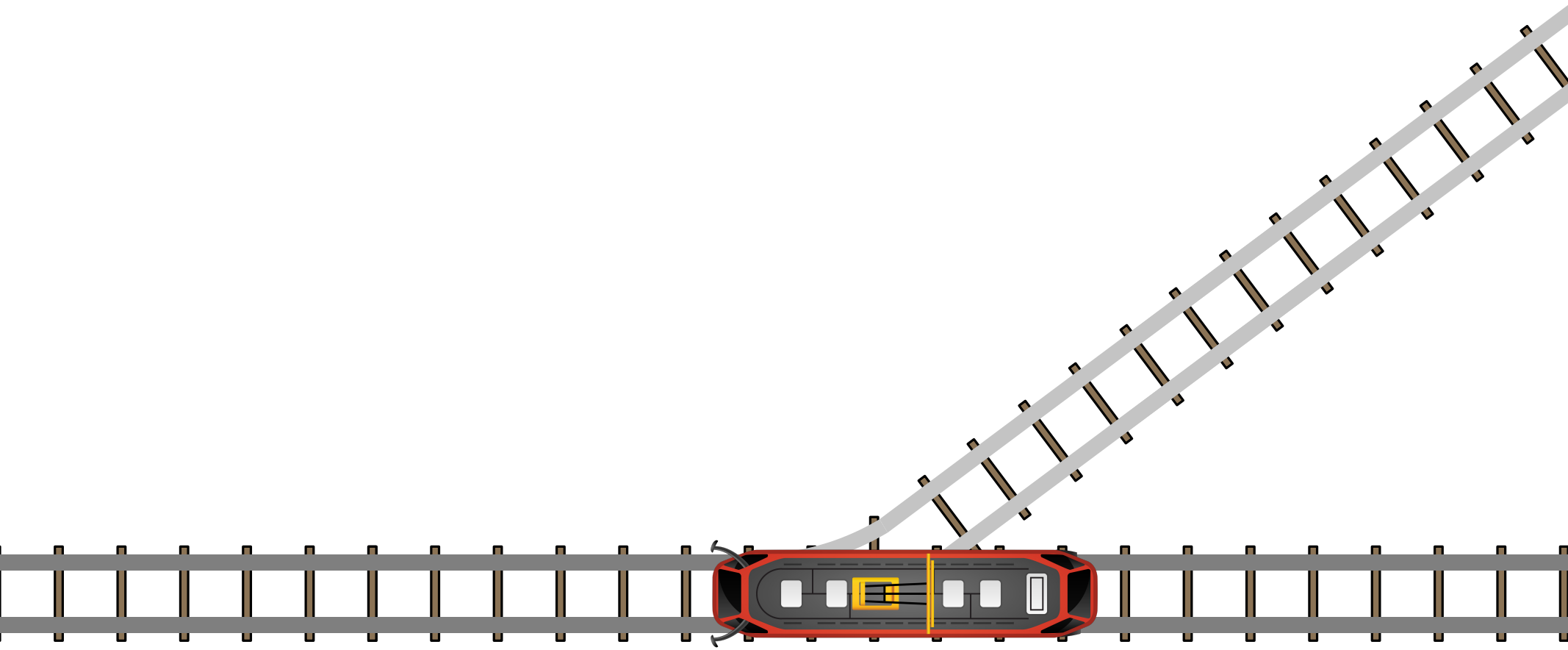
Designed by macrovector / Freepik



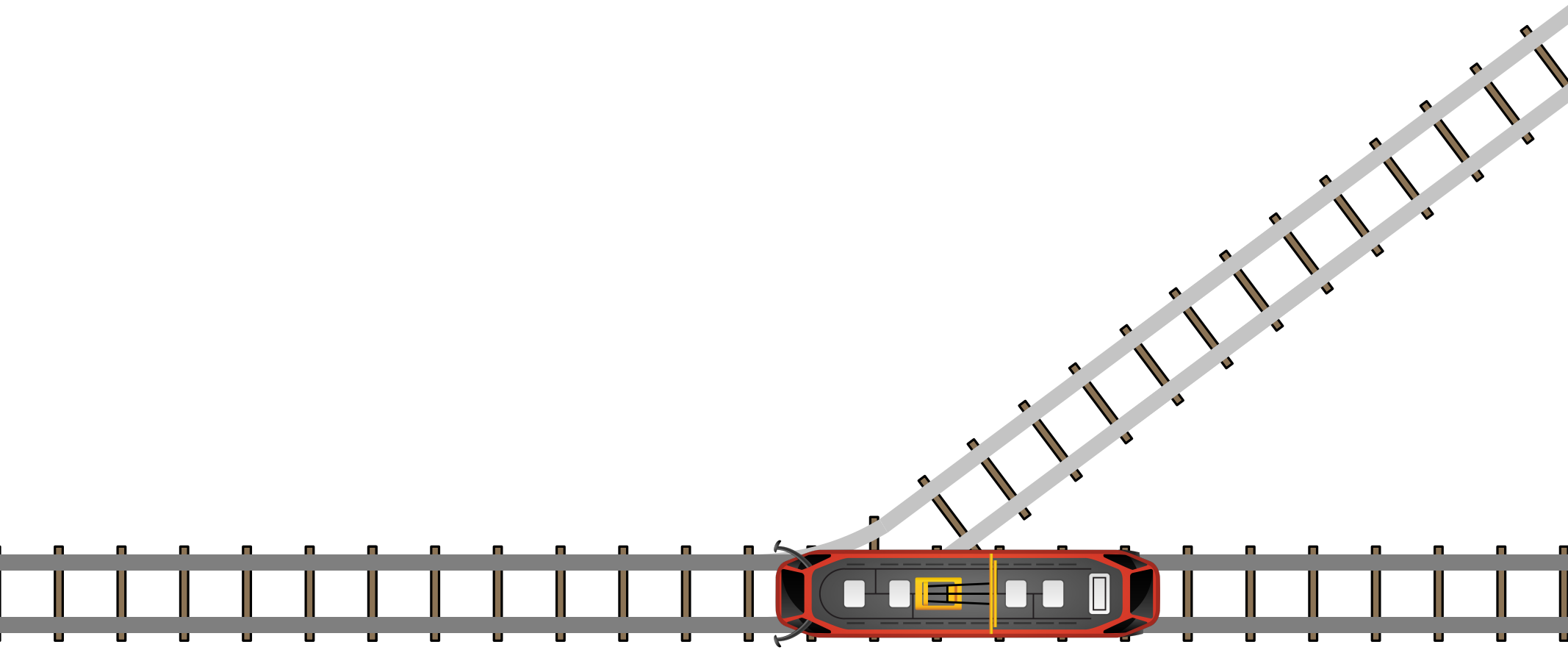
Designed by macrovector / Freepik



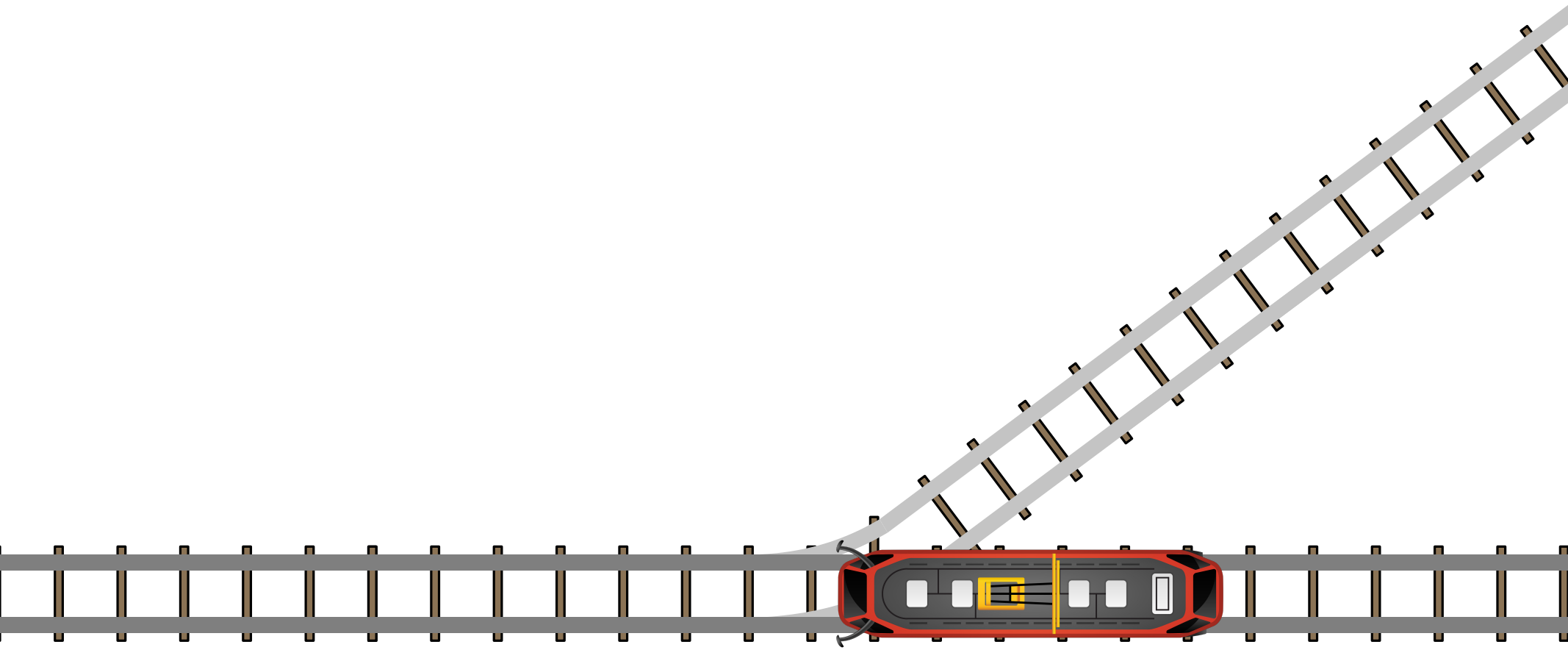
Designed by macrovector / Freepik



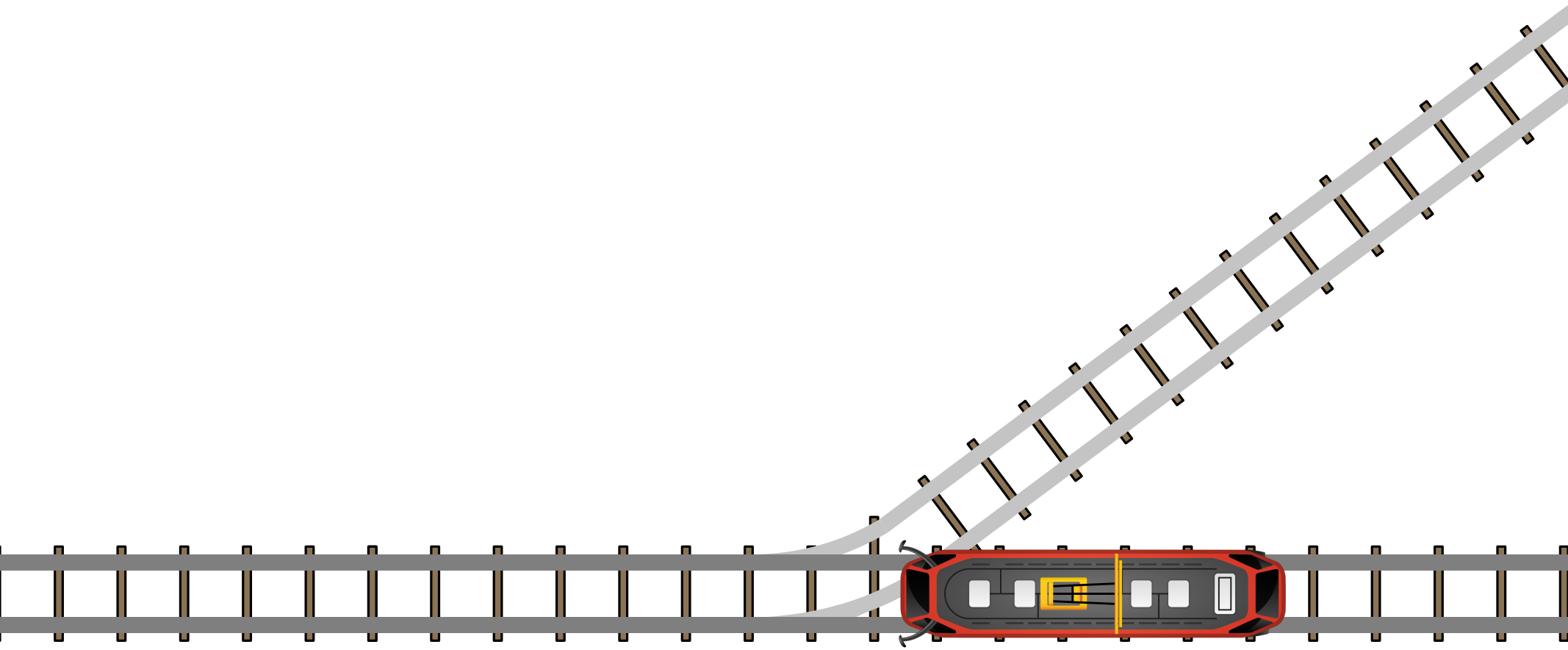
Designed by macrovector / Freepik



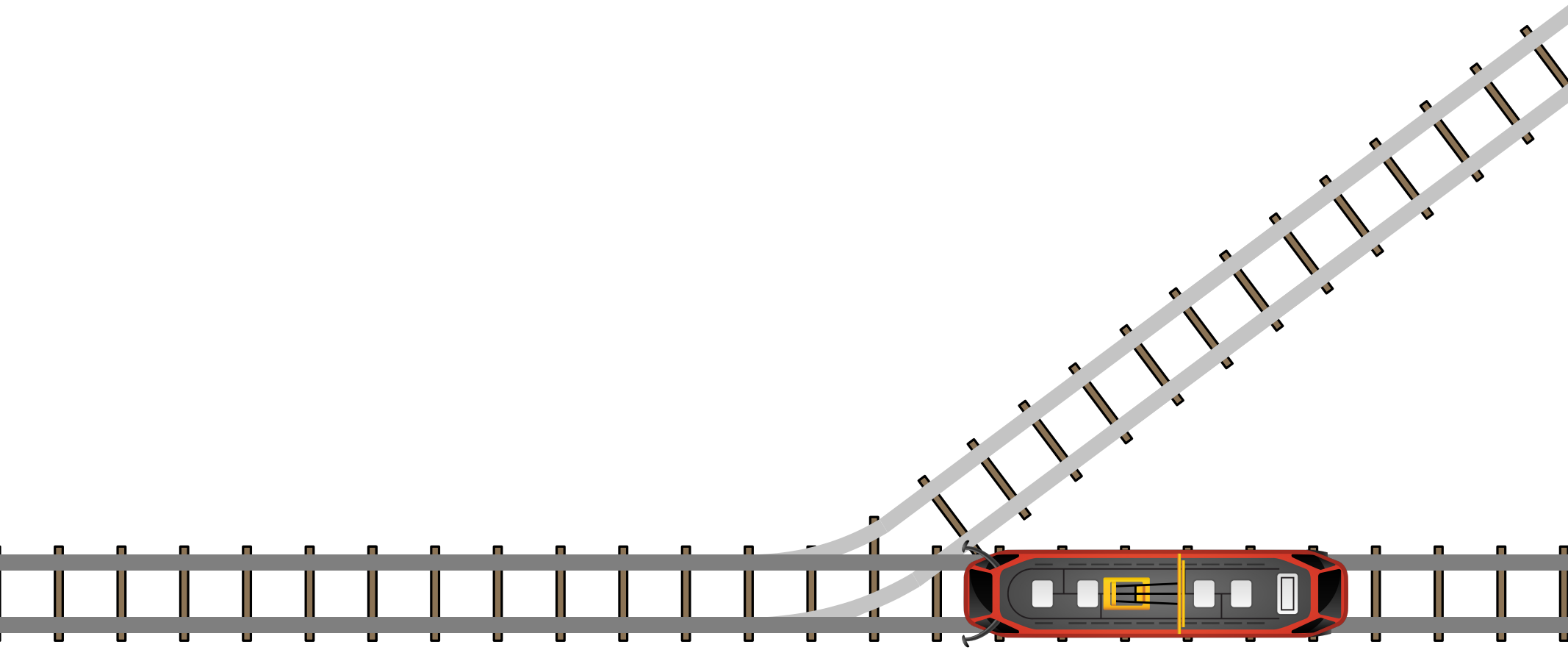
Designed by macrovector / Freepik



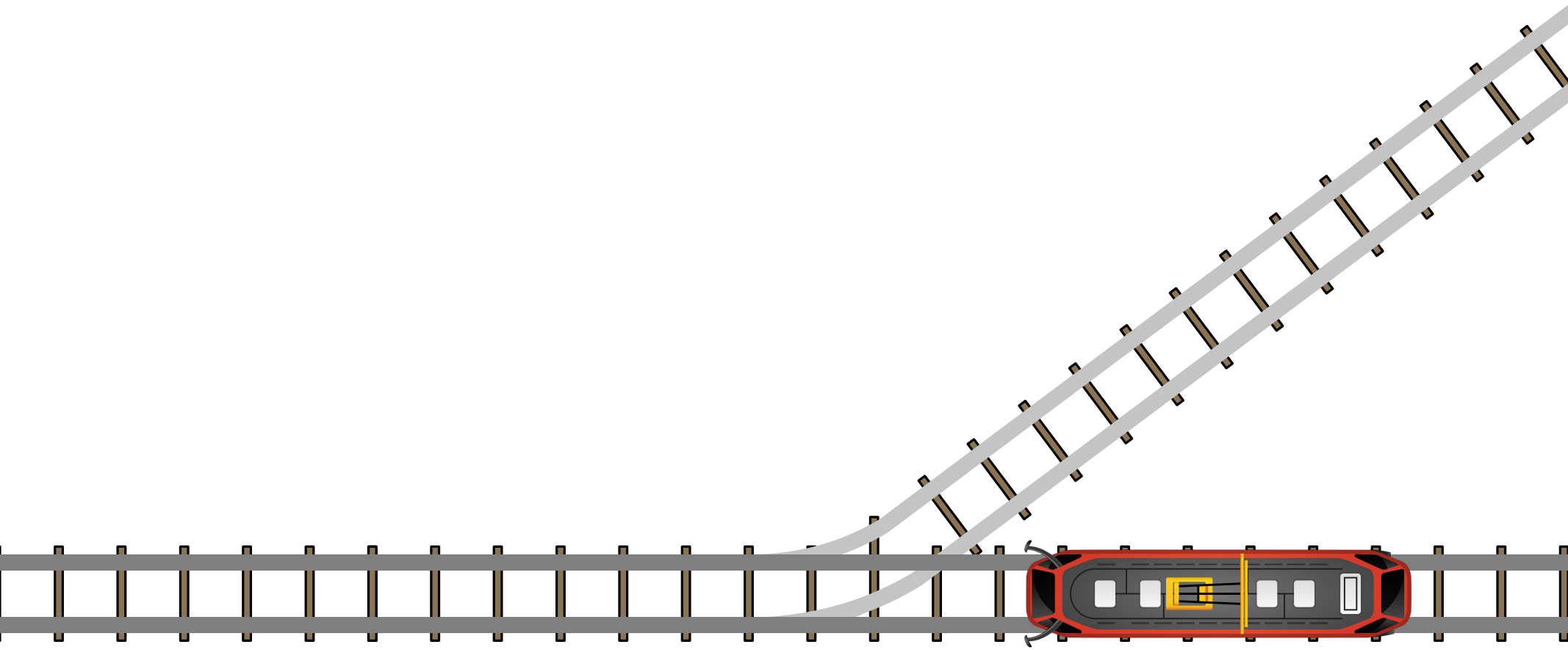
Designed by macrovector / Freepik



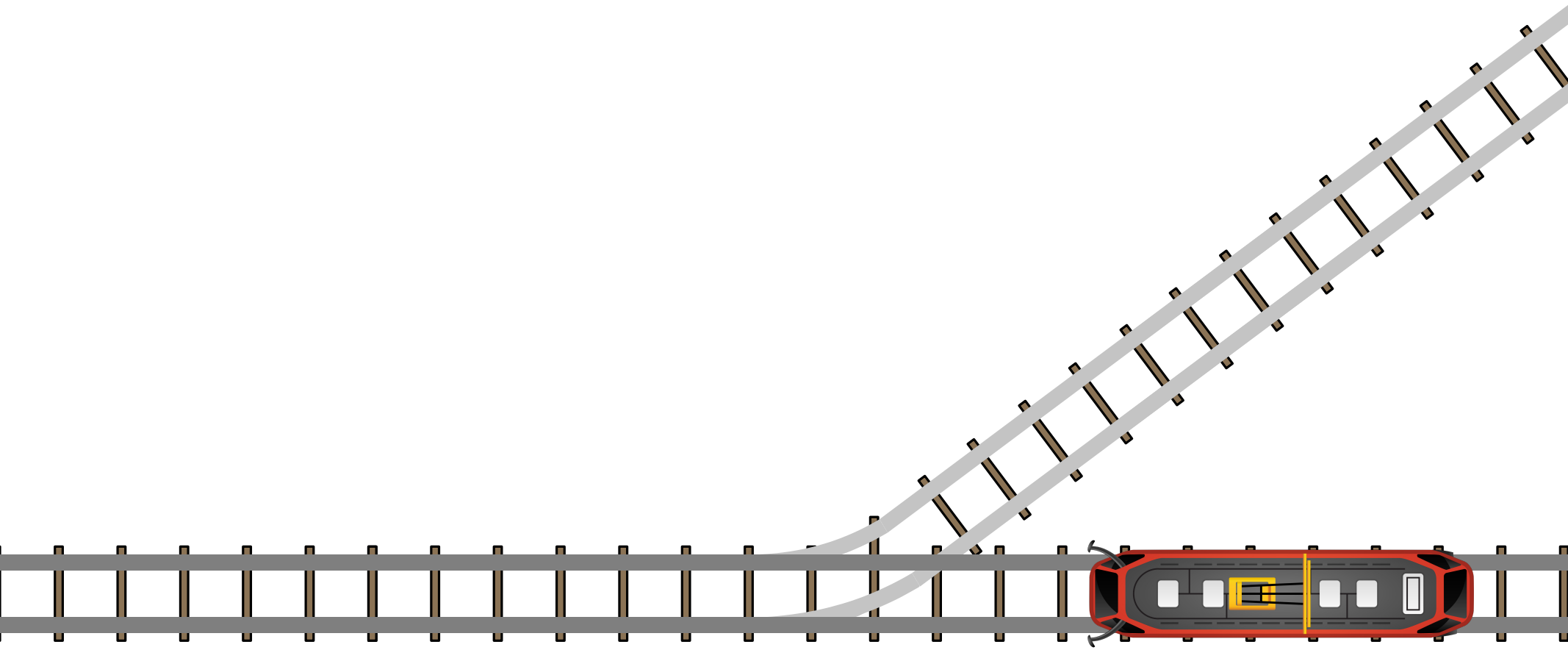
Designed by macrovector / Freepik



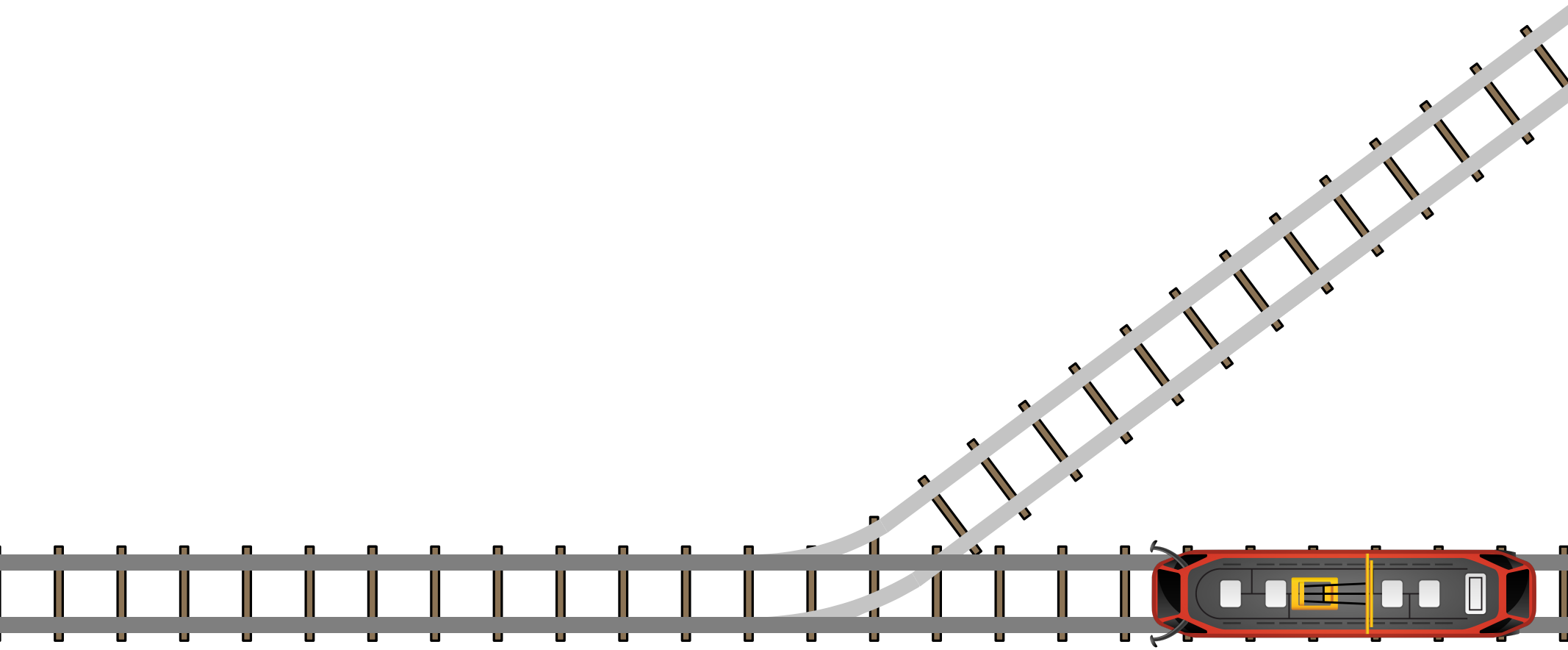
Designed by macrovector / Freepik



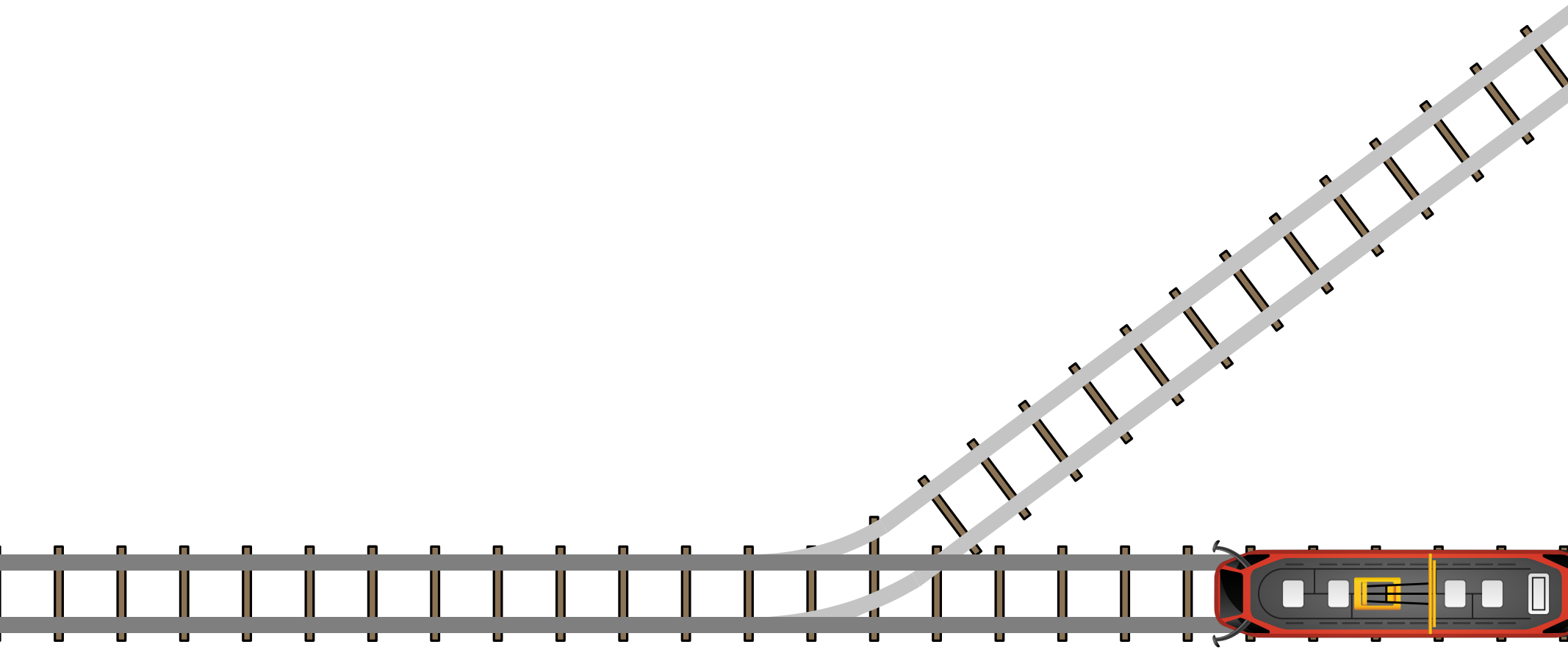
Designed by macrovector / Freepik



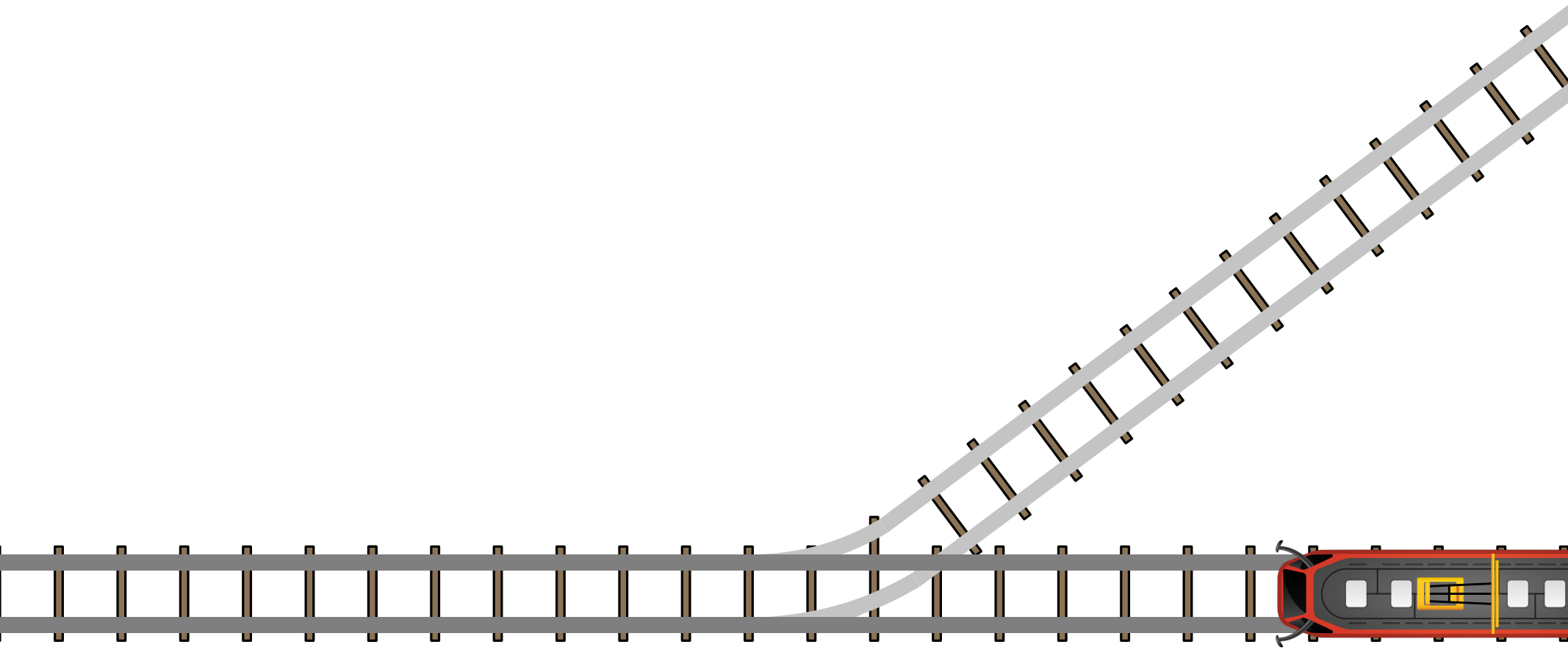
Designed by macrovector / Freepik



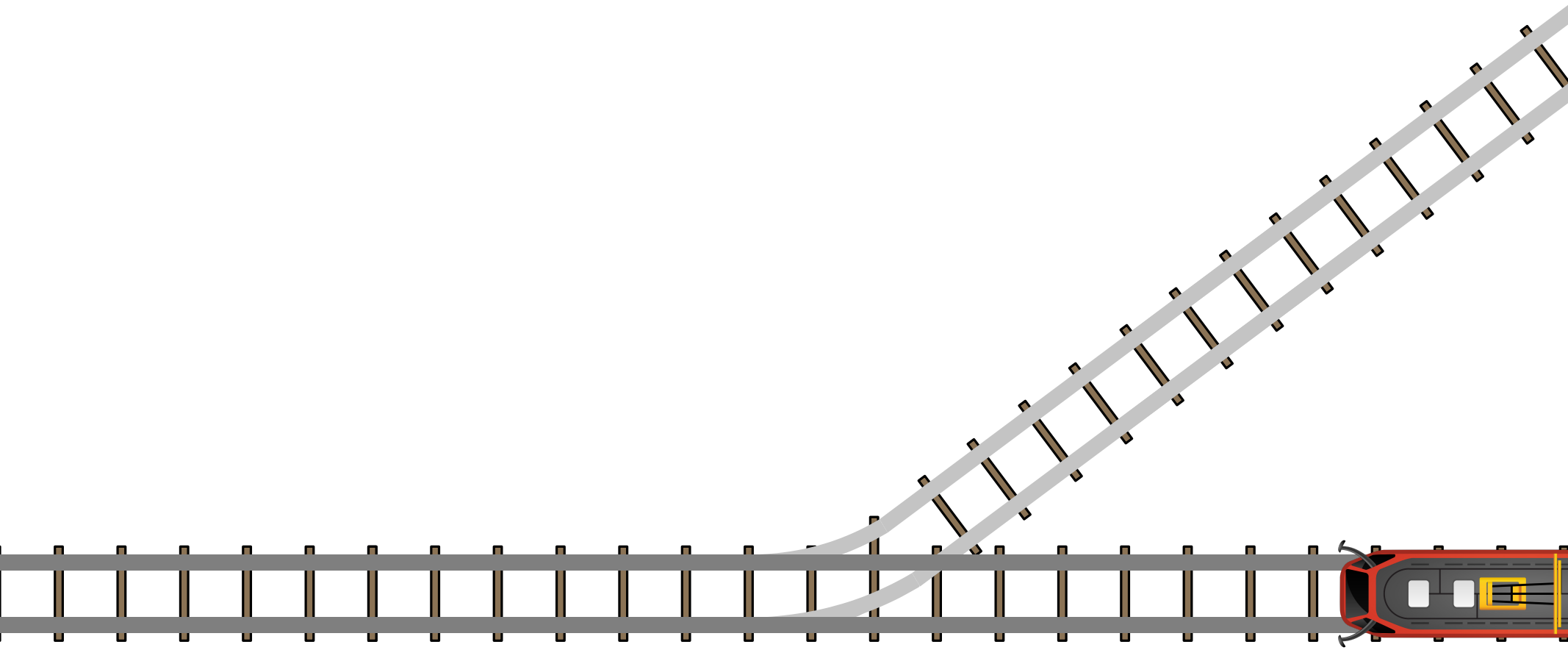
Designed by macrovector / Freepik



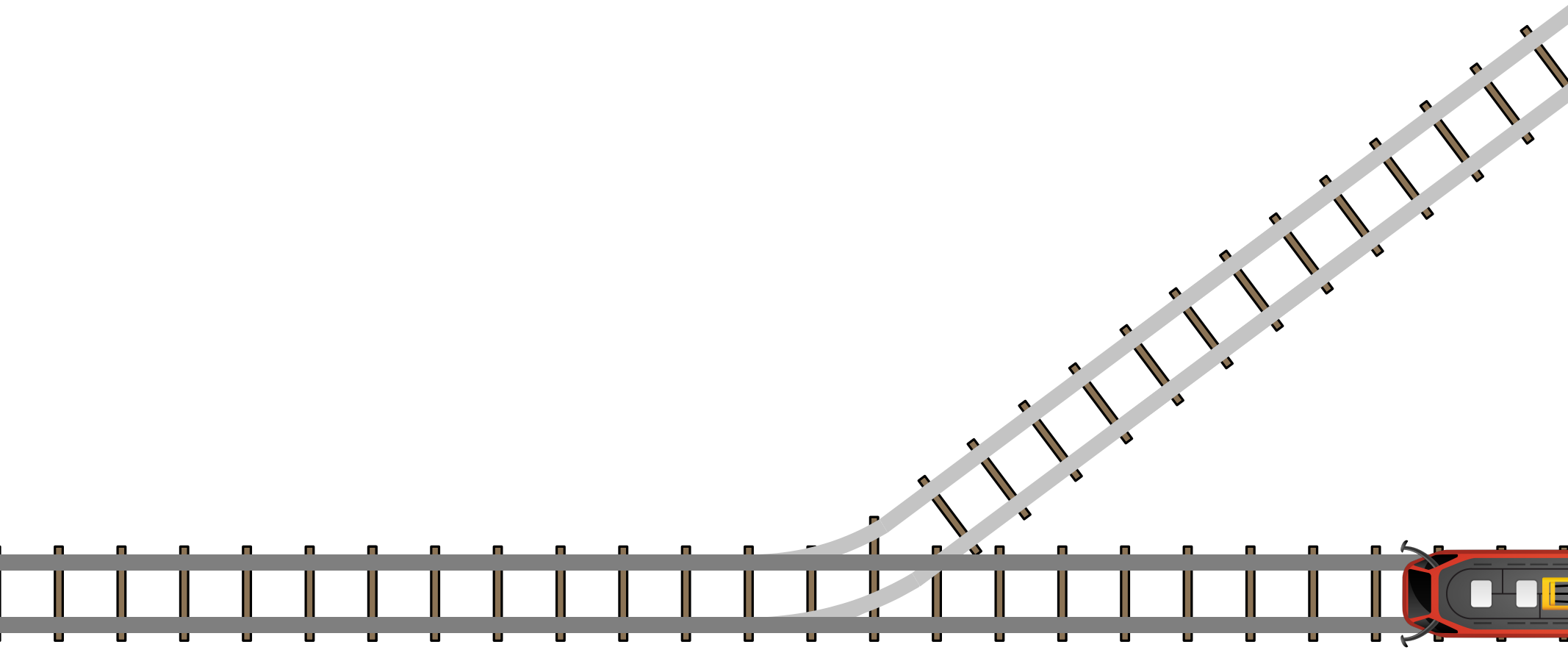
Designed by macrovector / Freepik



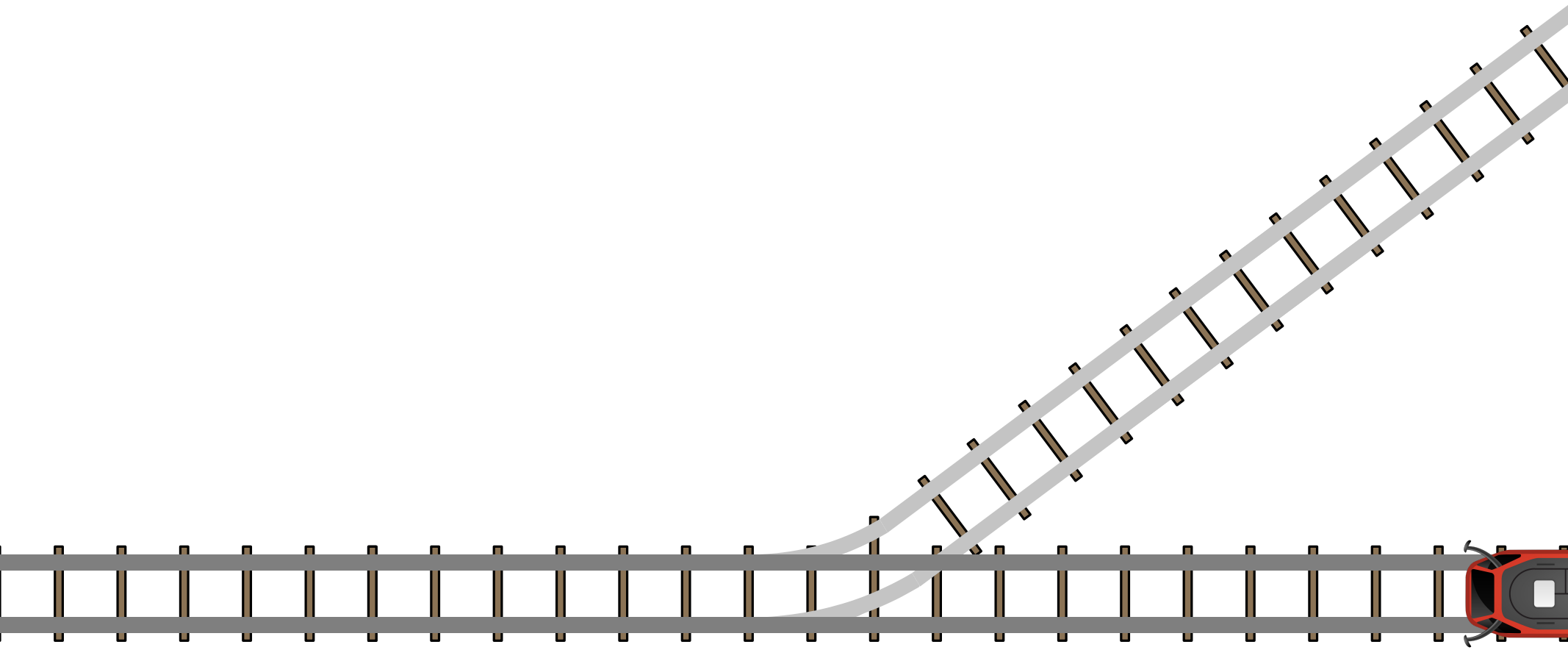
Designed by macrovector / Freepik



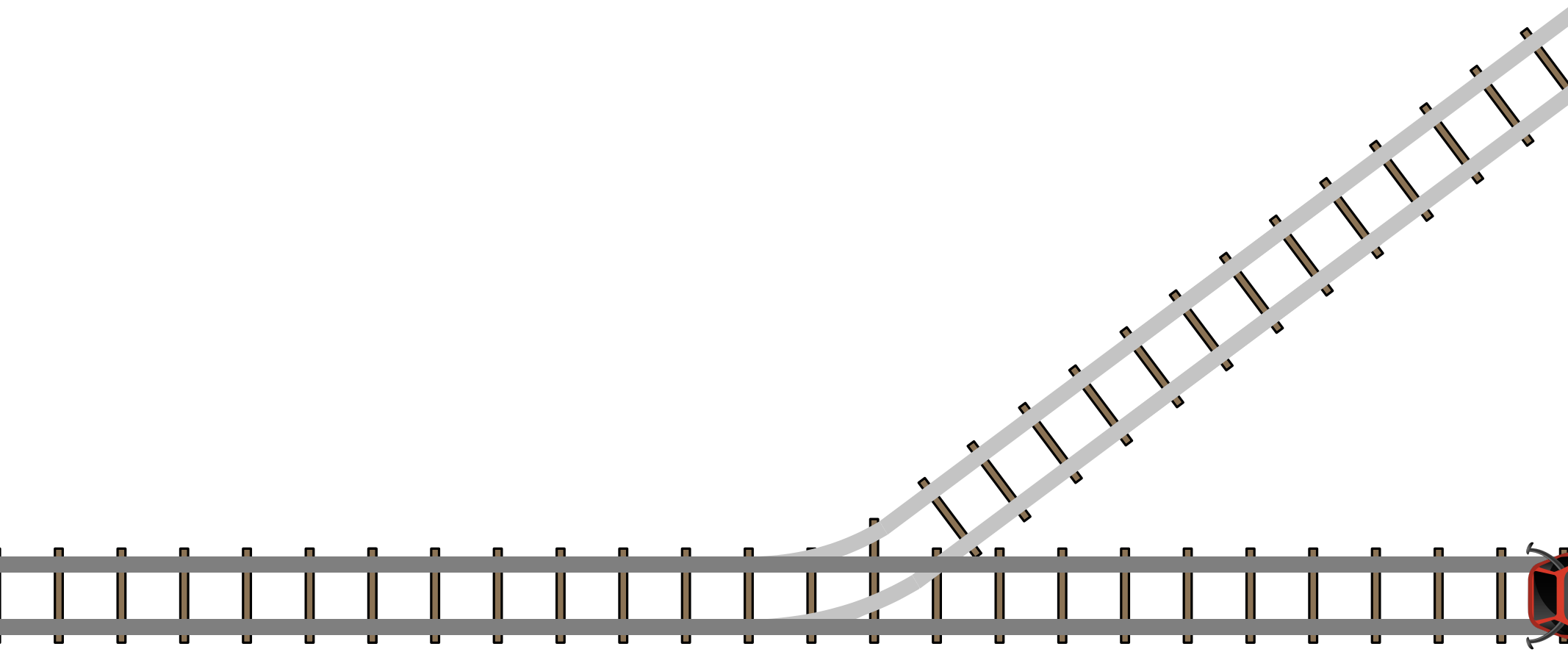
Designed by macrovector / Freepik



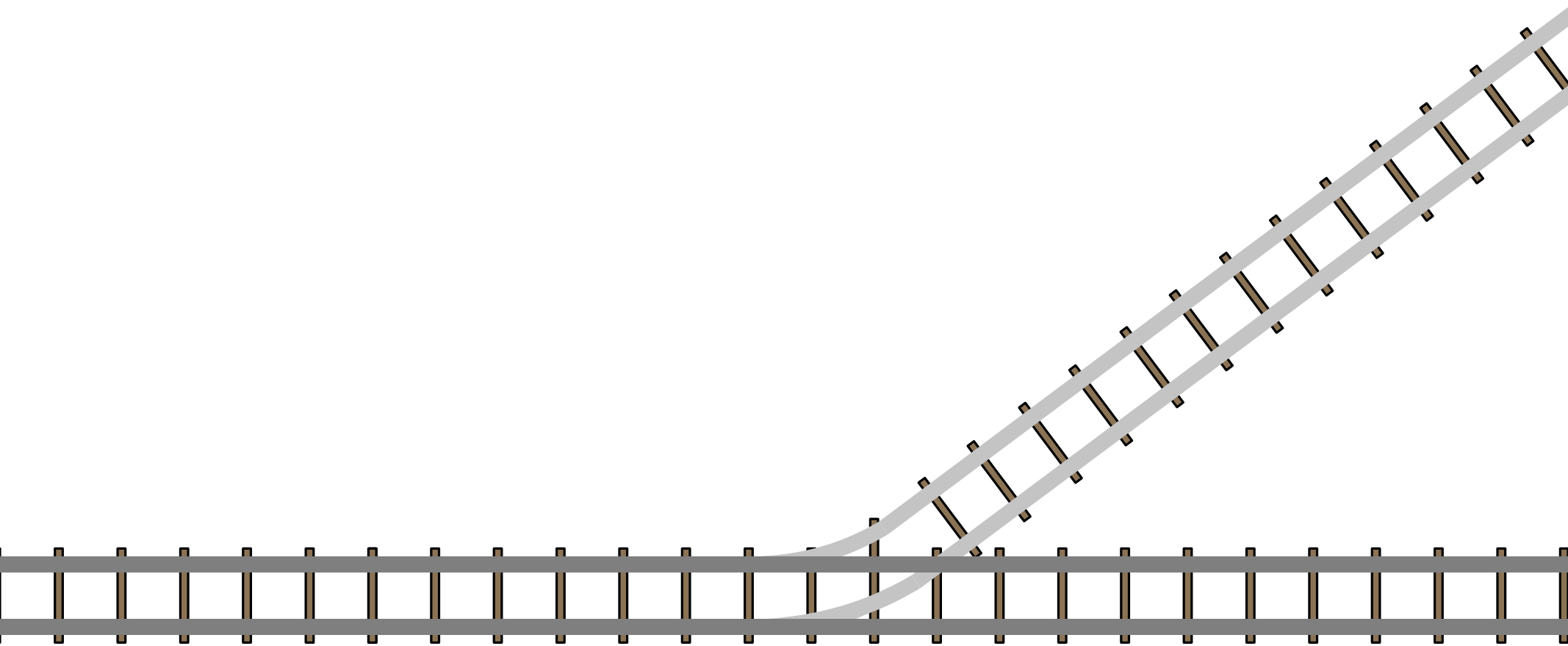
Designed by macrovector / Freepik



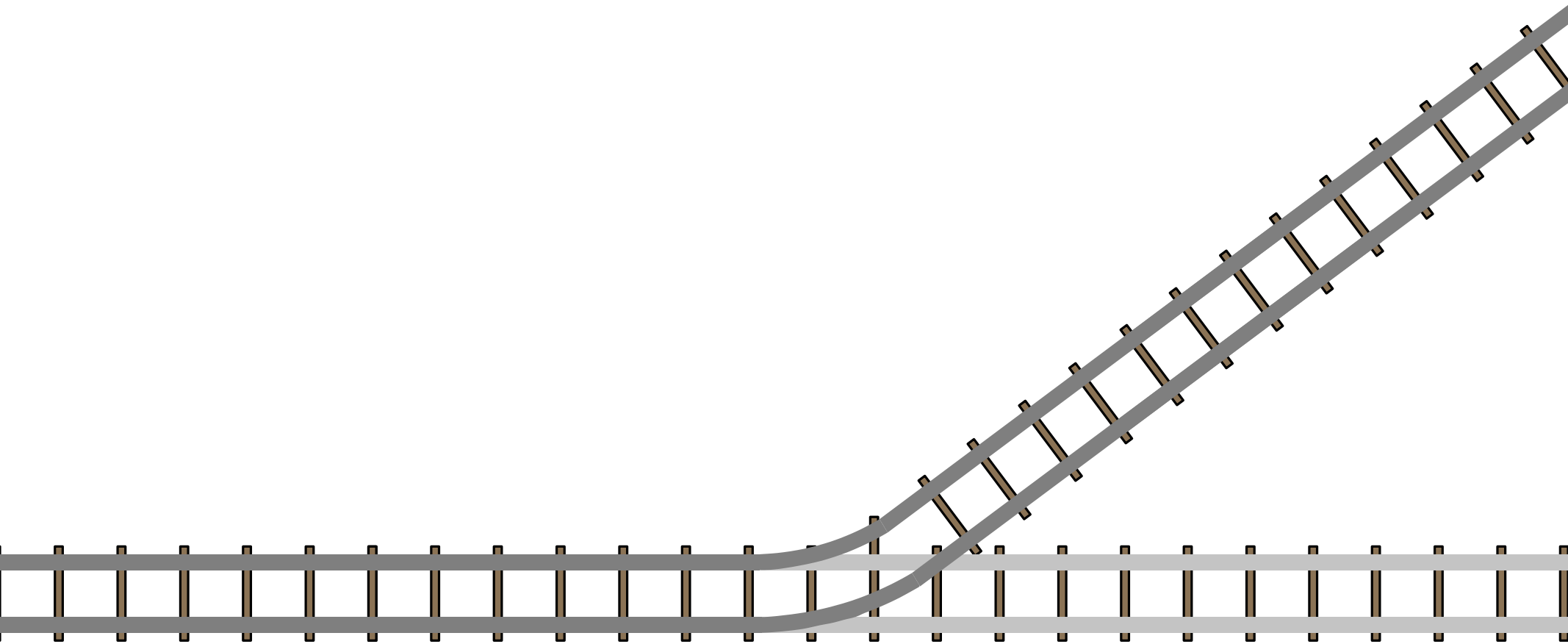
Designed by macrovector / Freepik



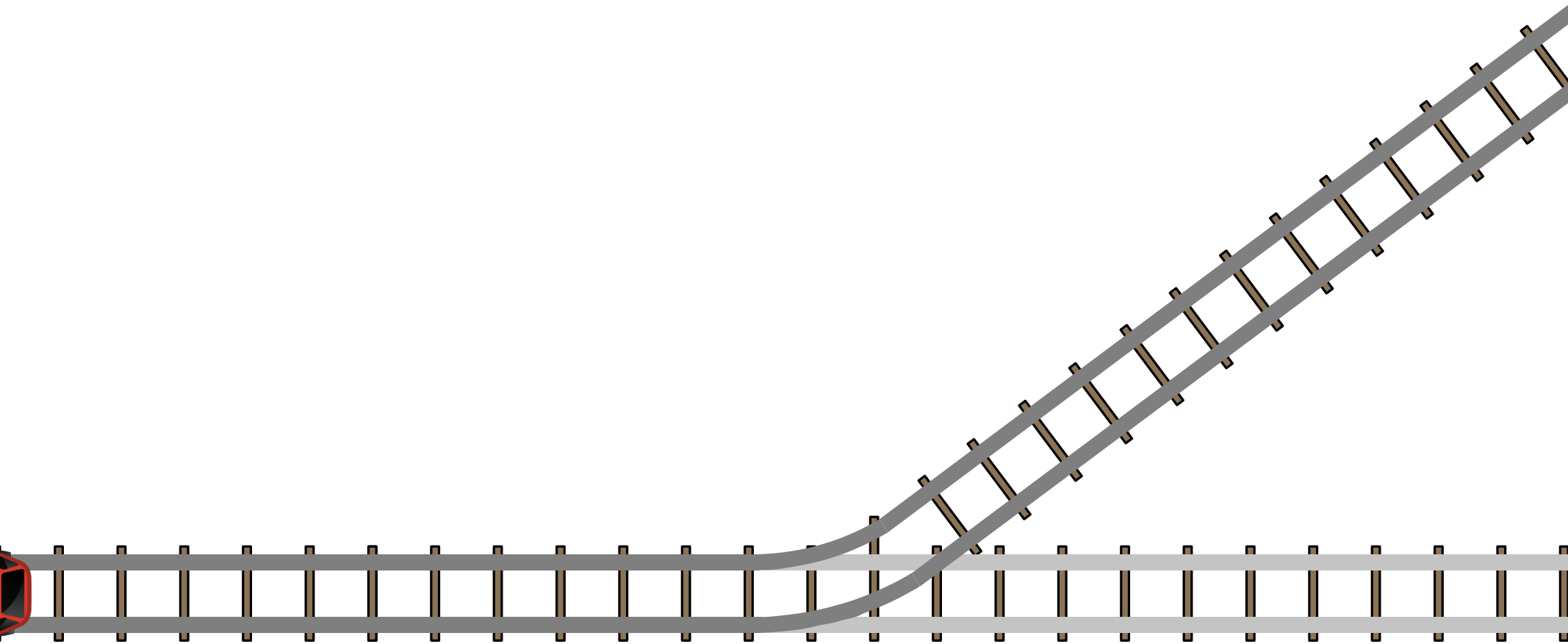
Designed by macrovector / Freepik



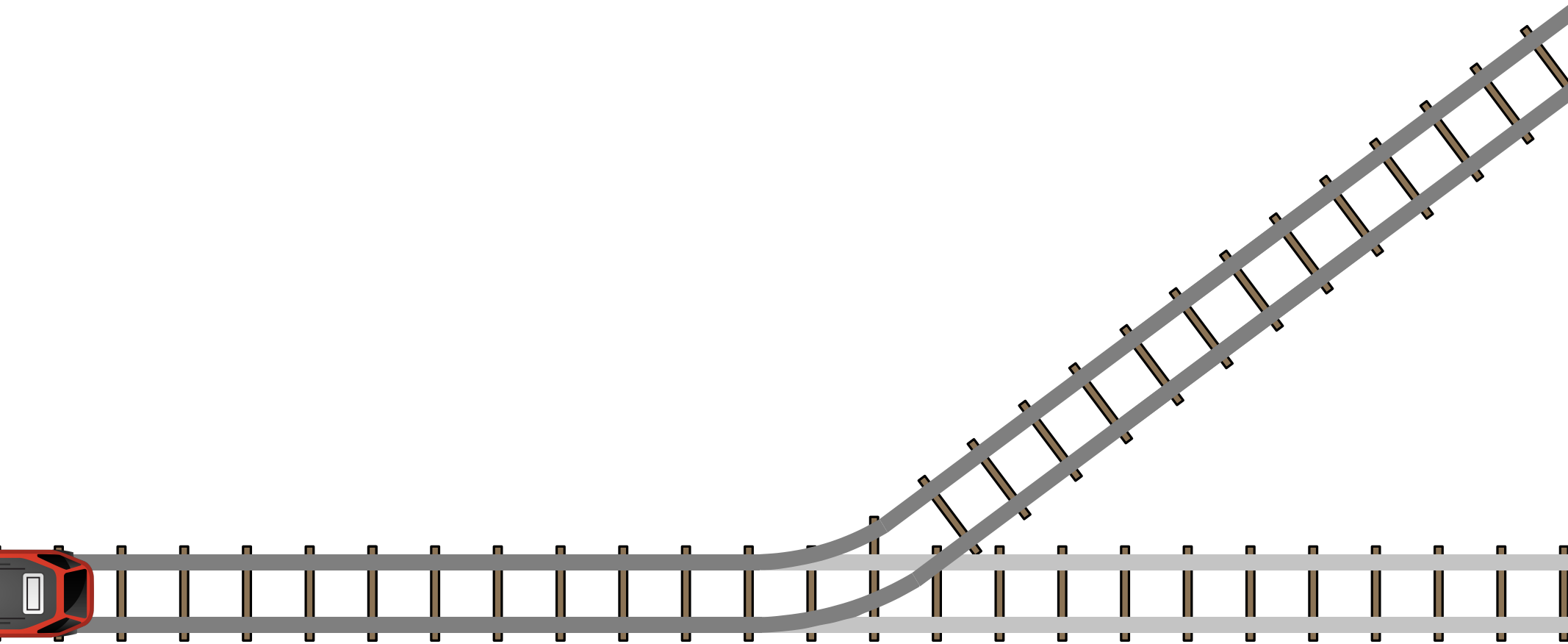
Designed by macrovector / Freepik



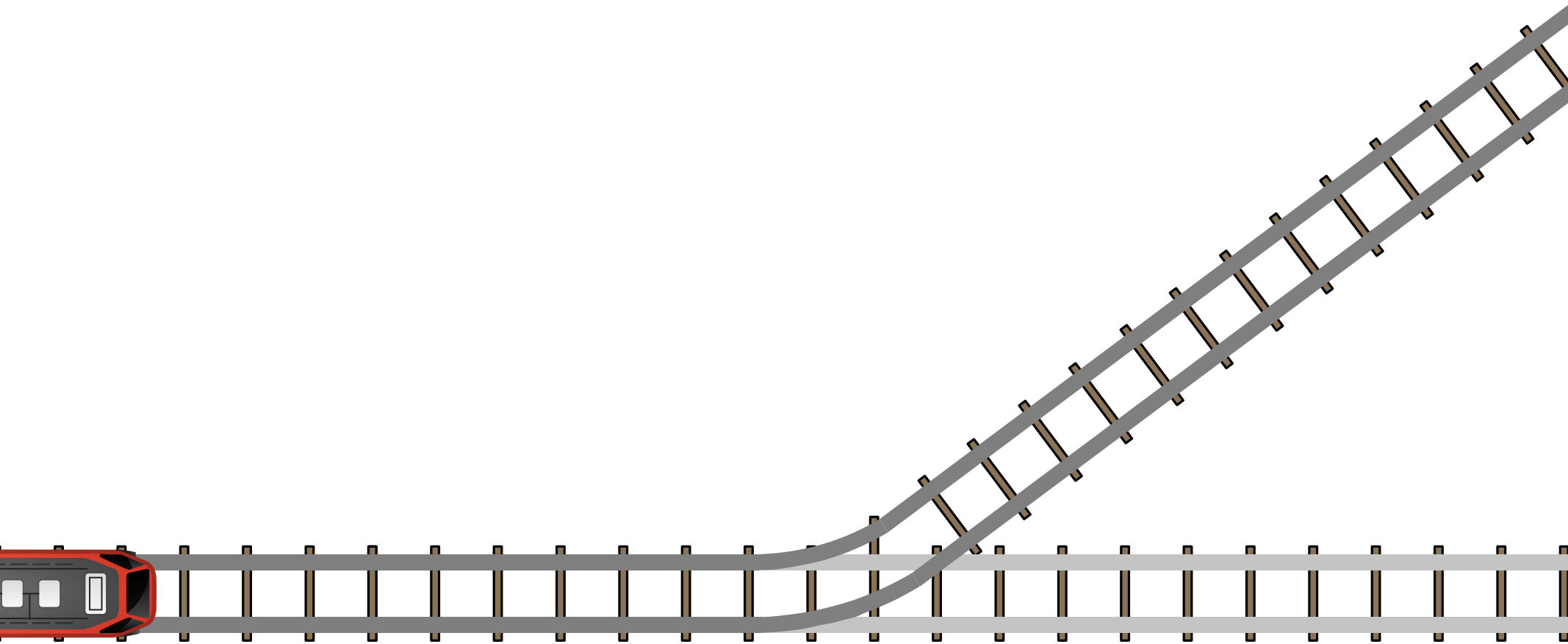
Designed by macrovector / Freepik



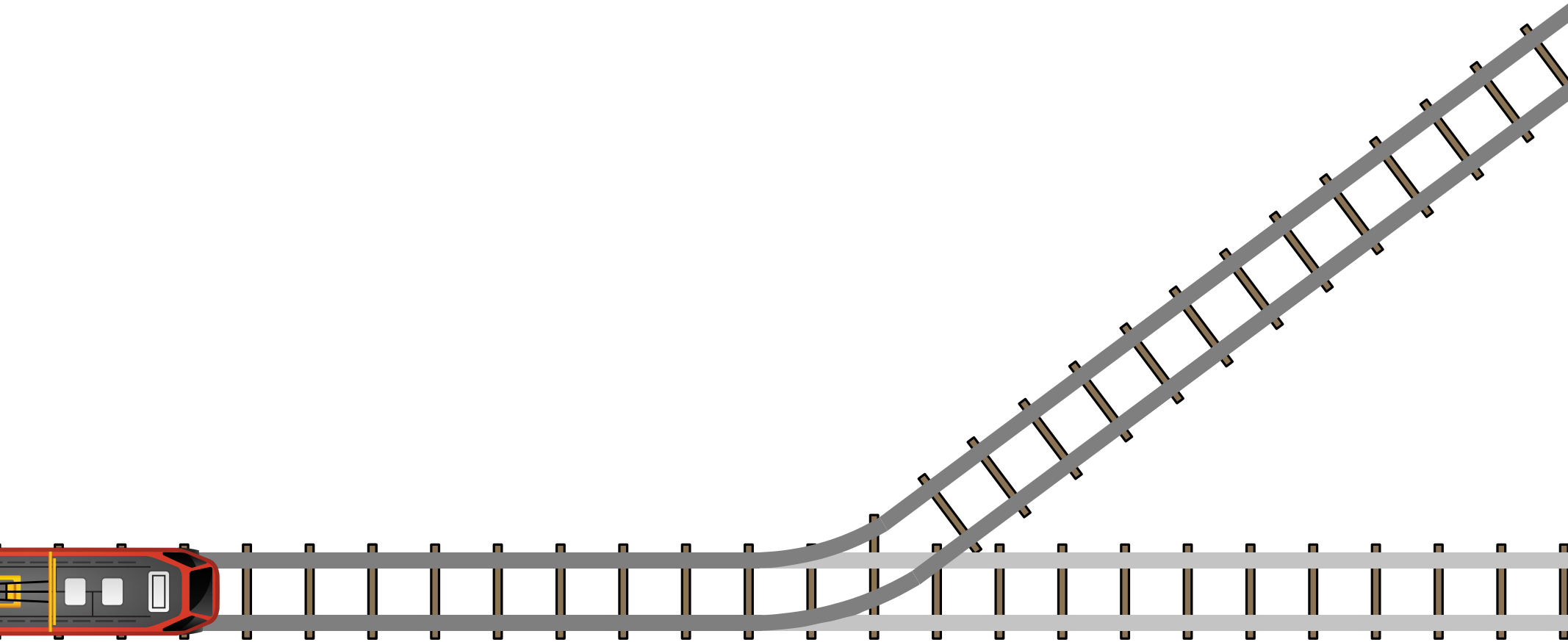
Designed by macrovector / Freepik



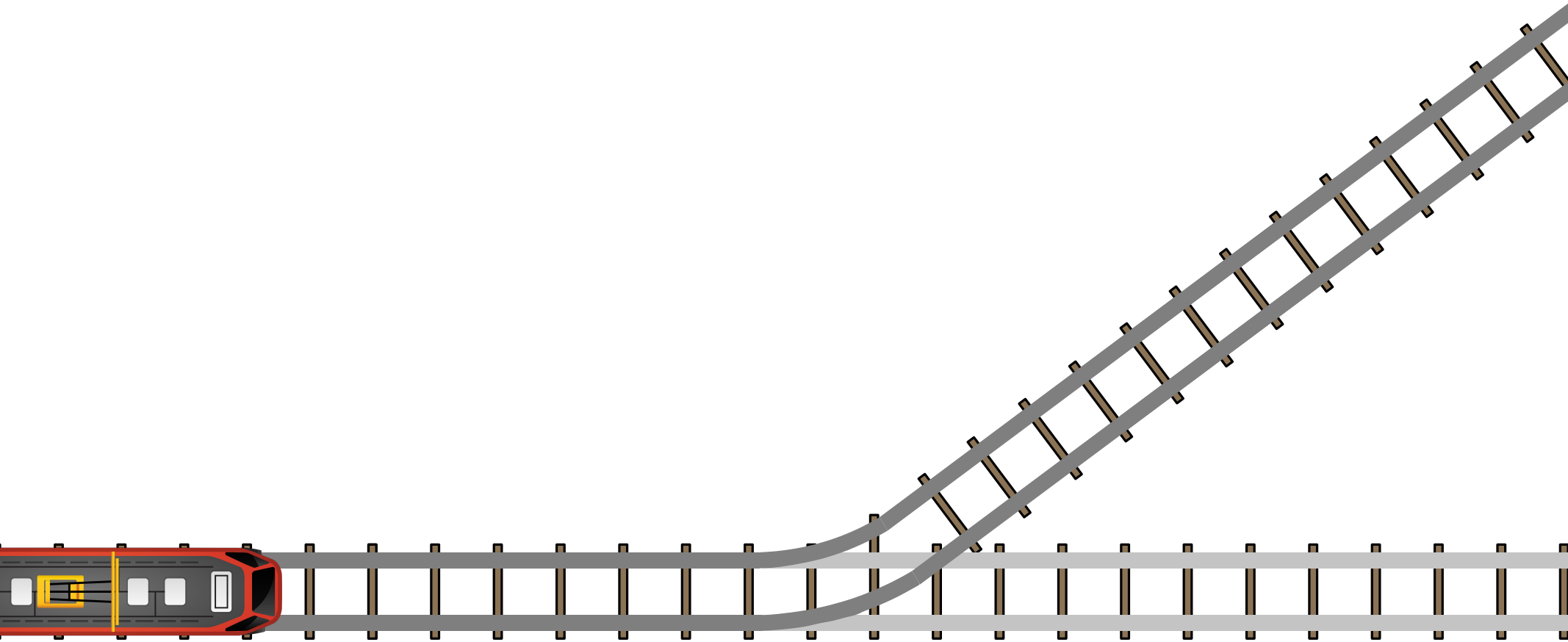
Designed by macrovector / Freepik



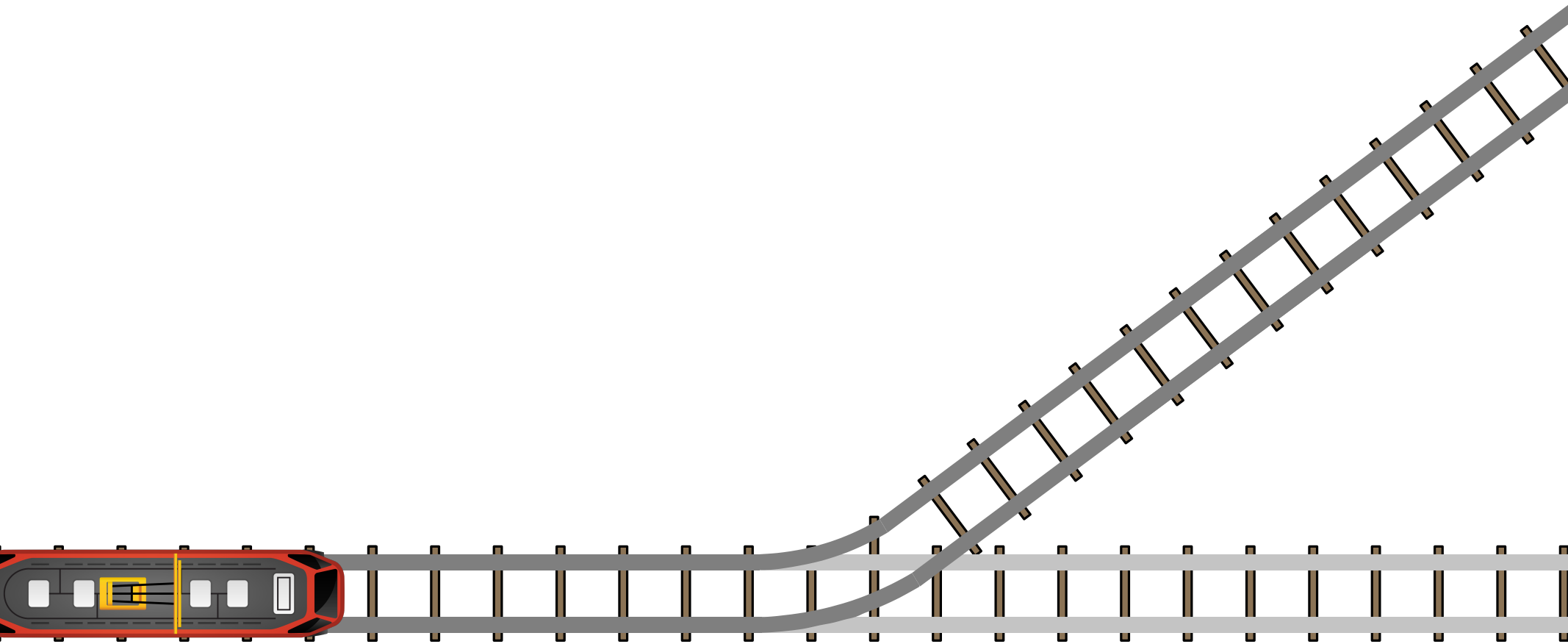
Designed by macrovector / Freepik



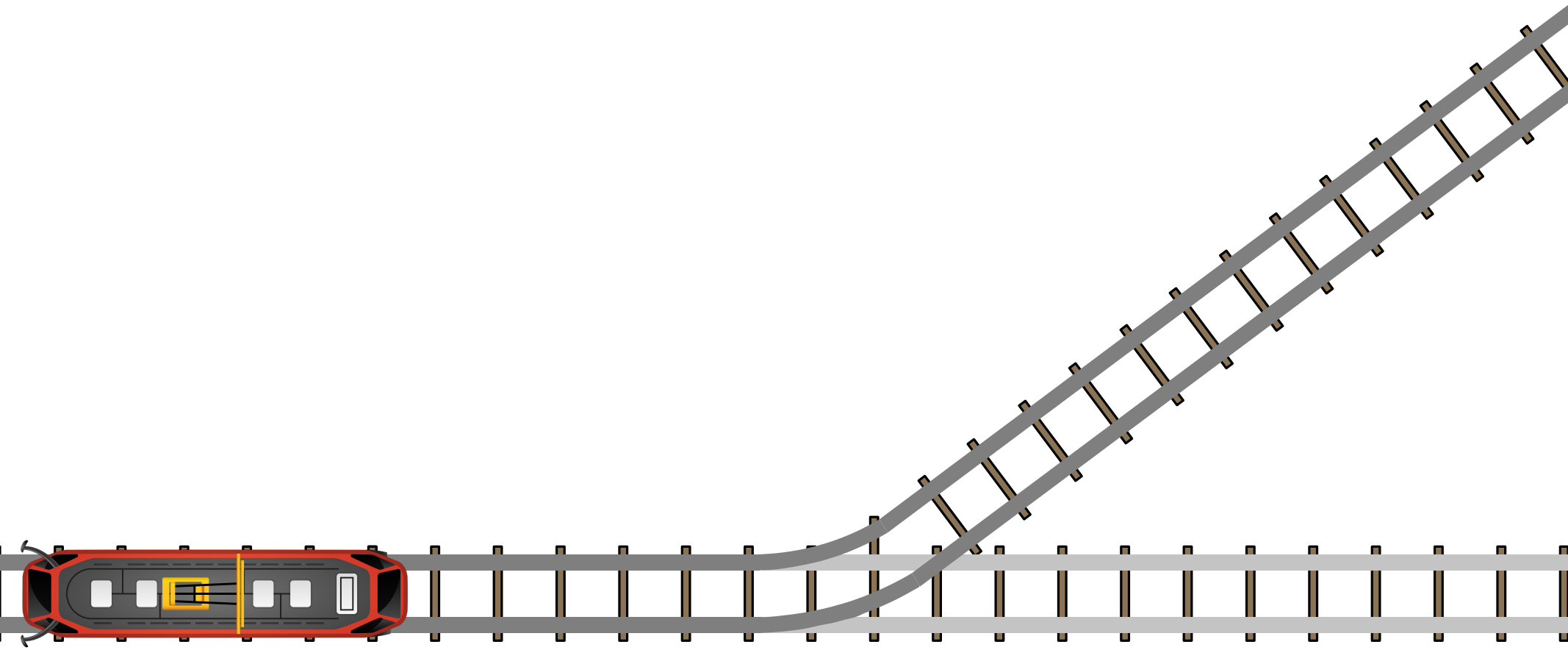
Designed by macrovector / Freepik



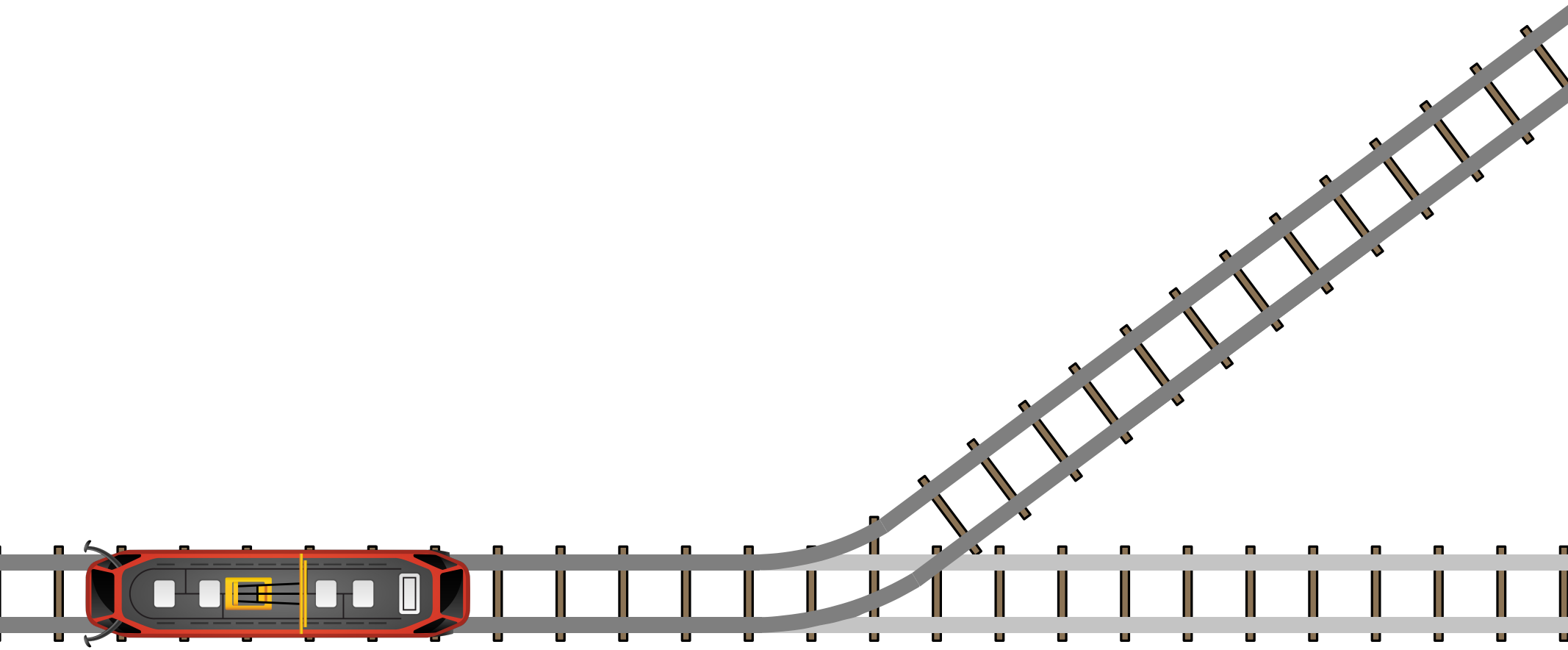
Designed by macrovector / Freepik



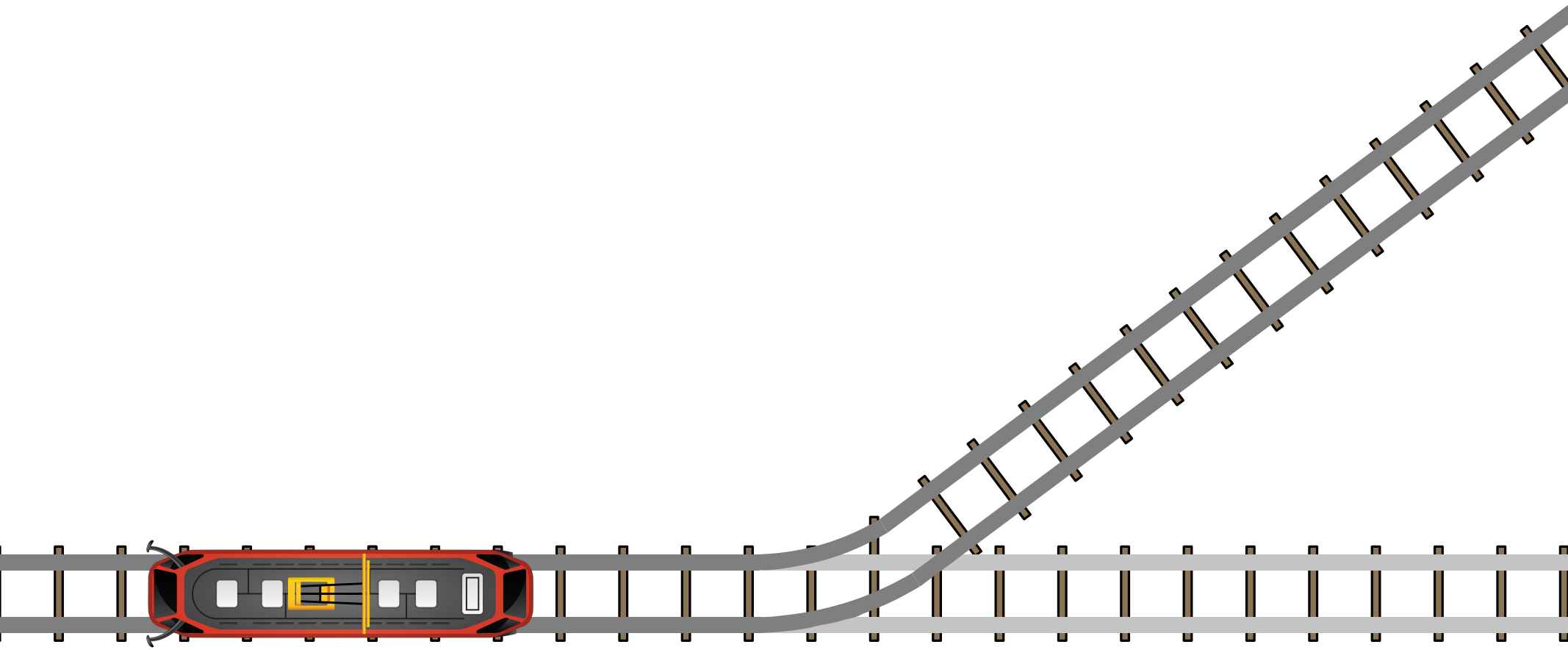
Designed by macrovector / Freepik



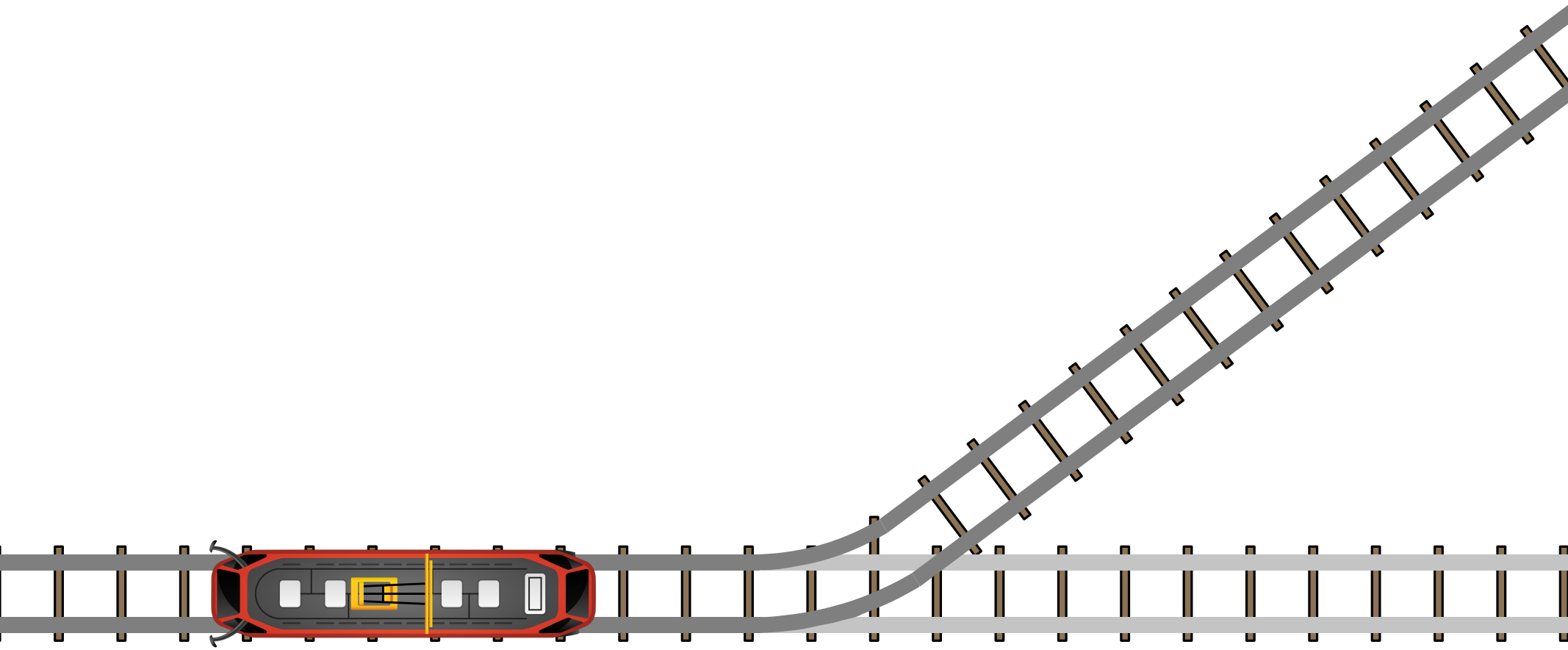
Designed by macrovector / Freepik



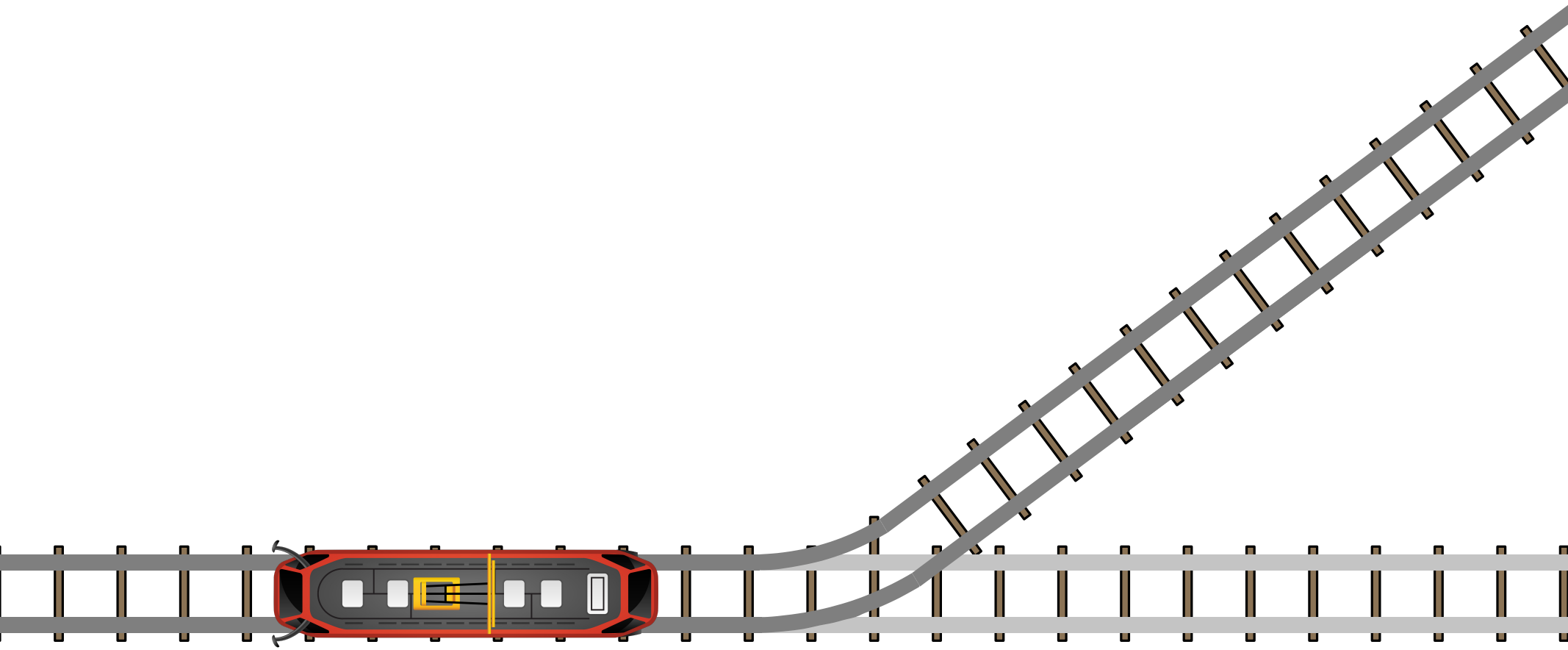
Designed by macrovector / Freepik



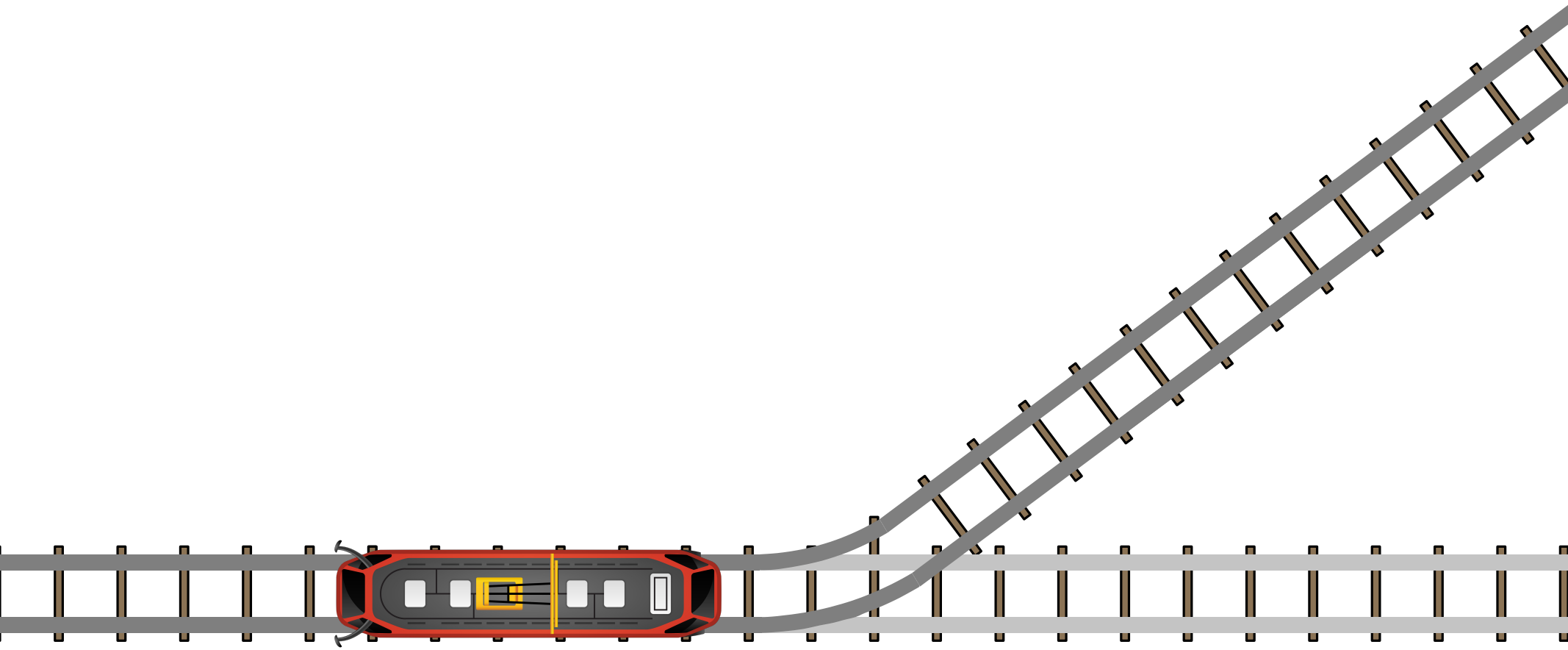
Designed by macrovector / Freepik



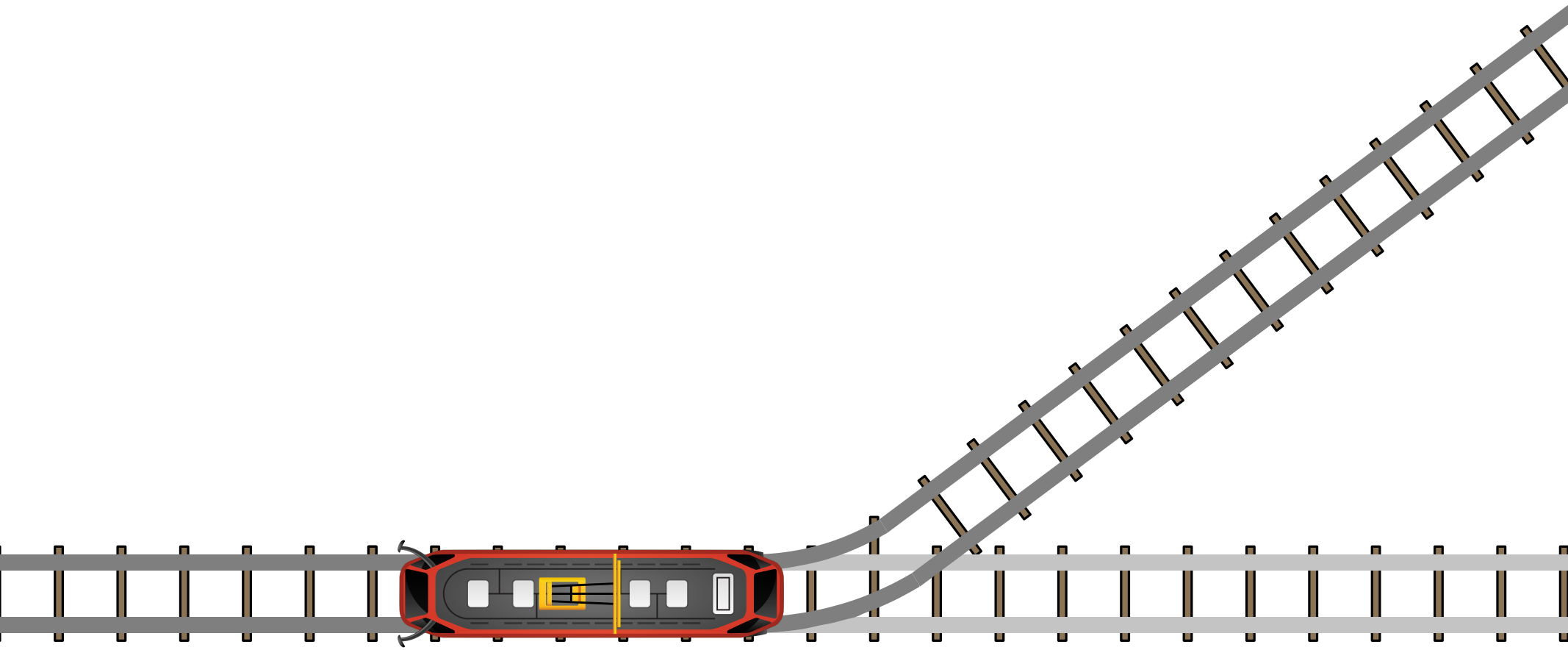
Designed by macrovector / Freepik



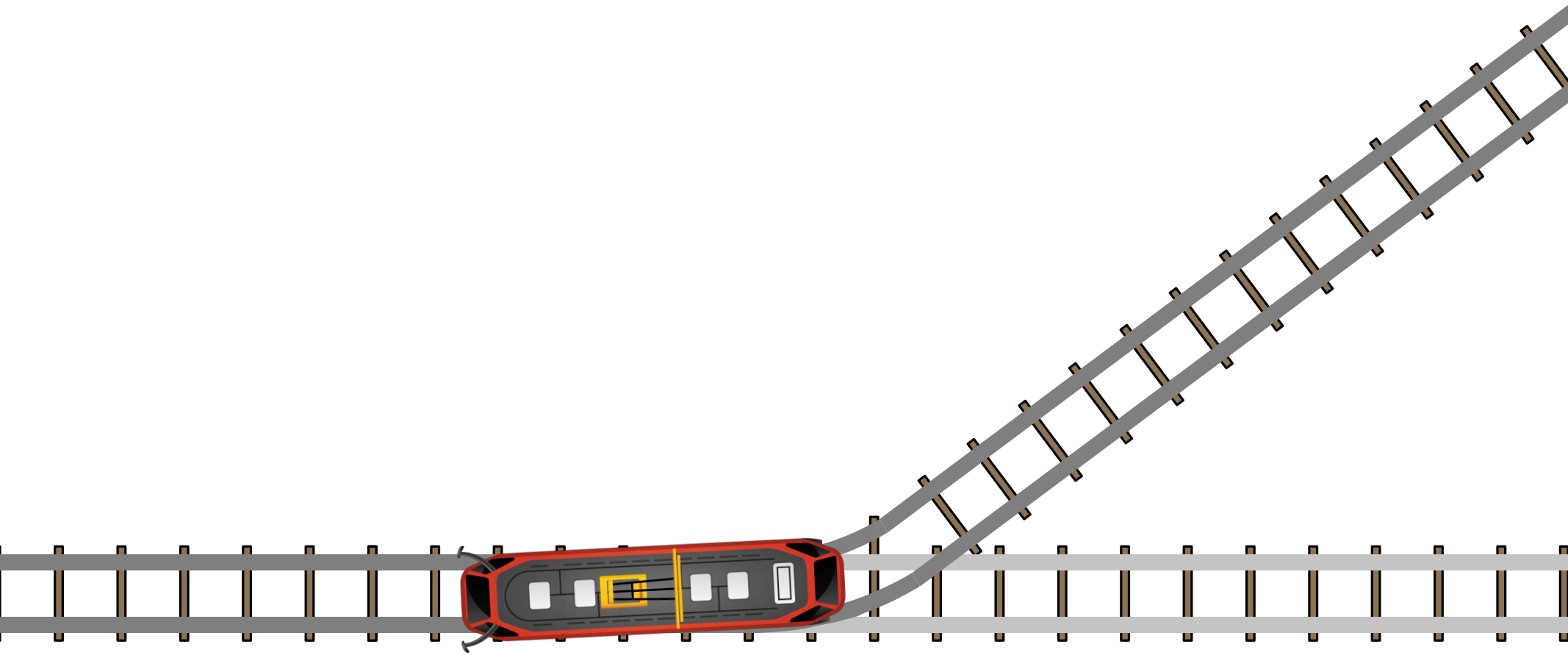
Designed by macrovector / Freepik



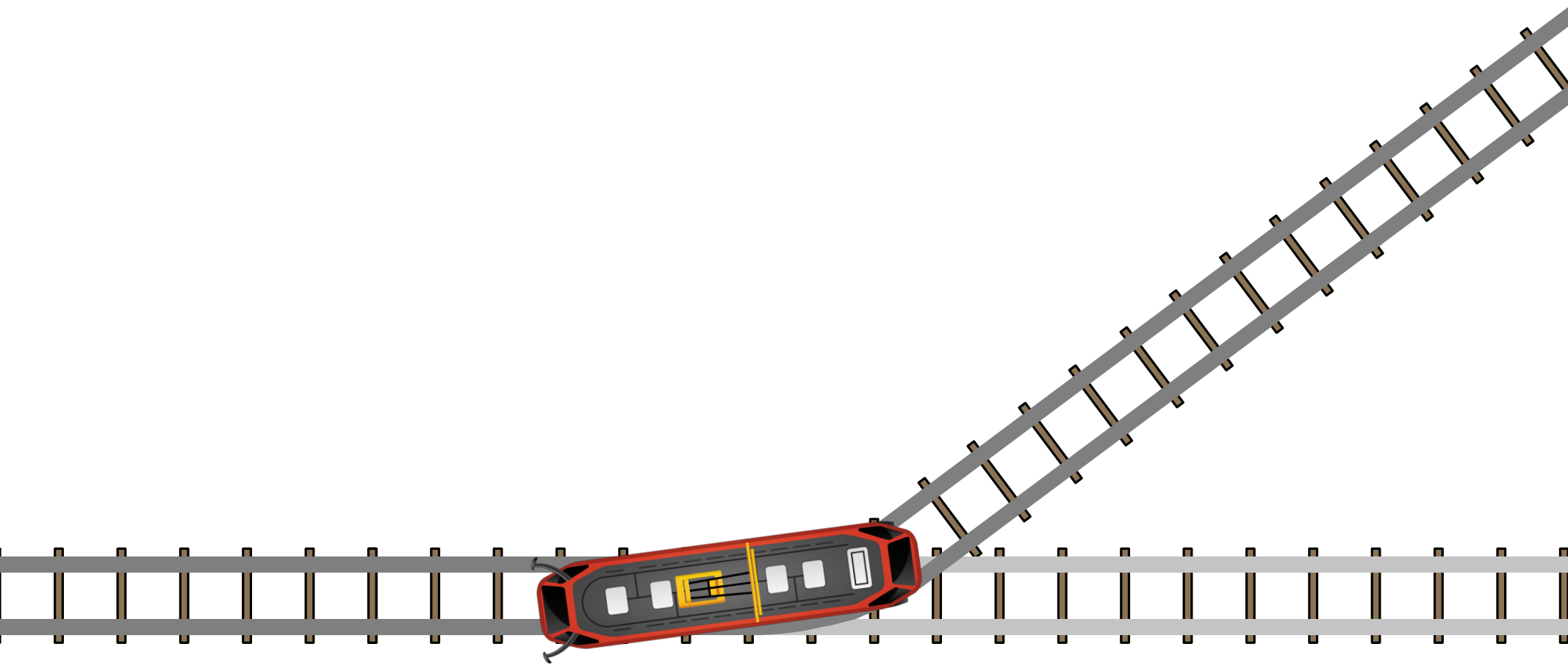
Designed by macrovector / Freepik



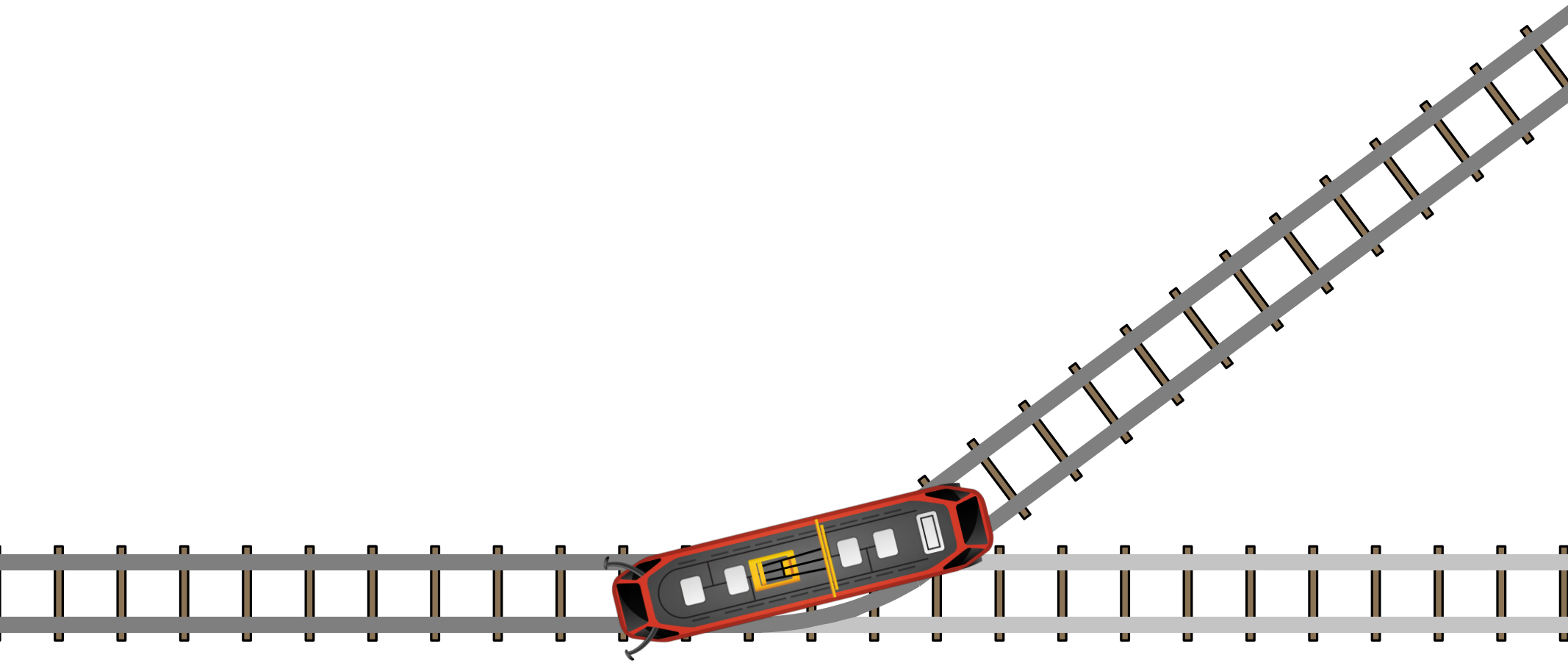
Designed by macrovector / Freepik



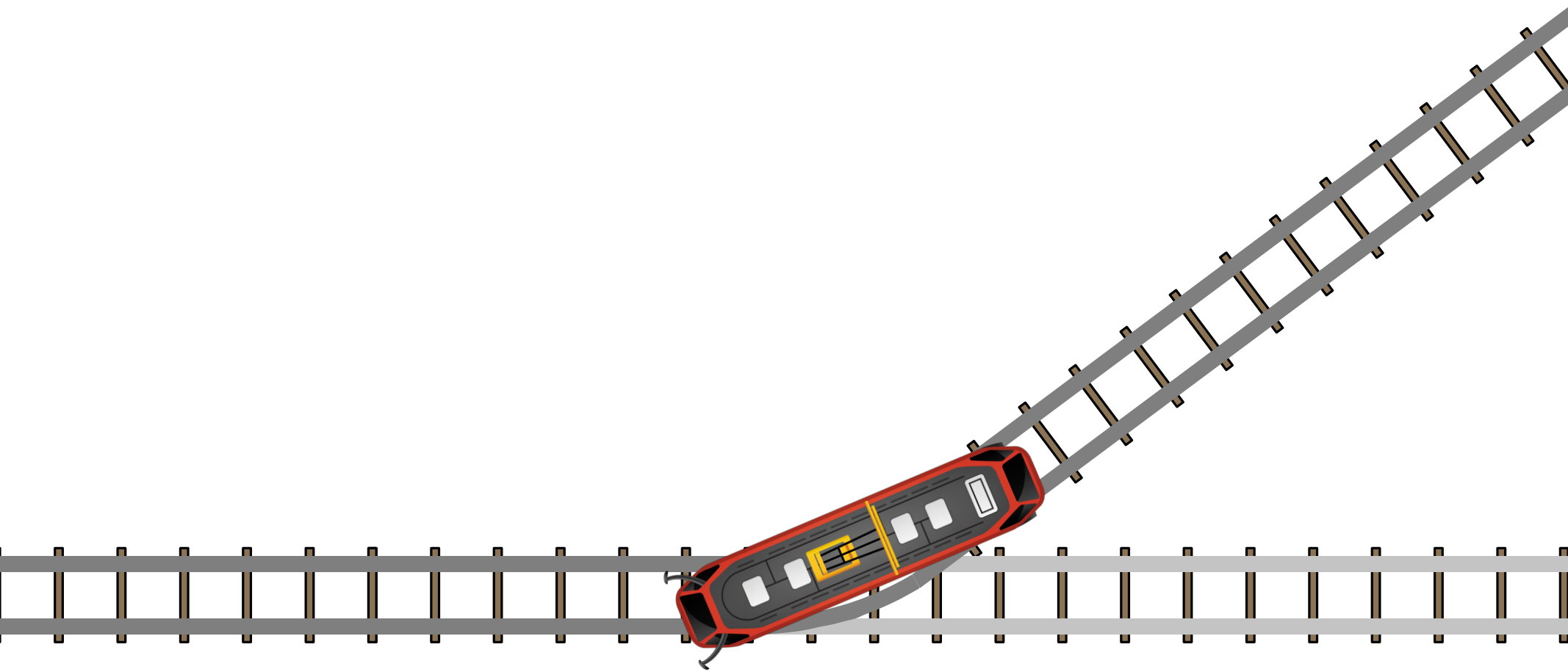
Designed by macrovector / Freepik



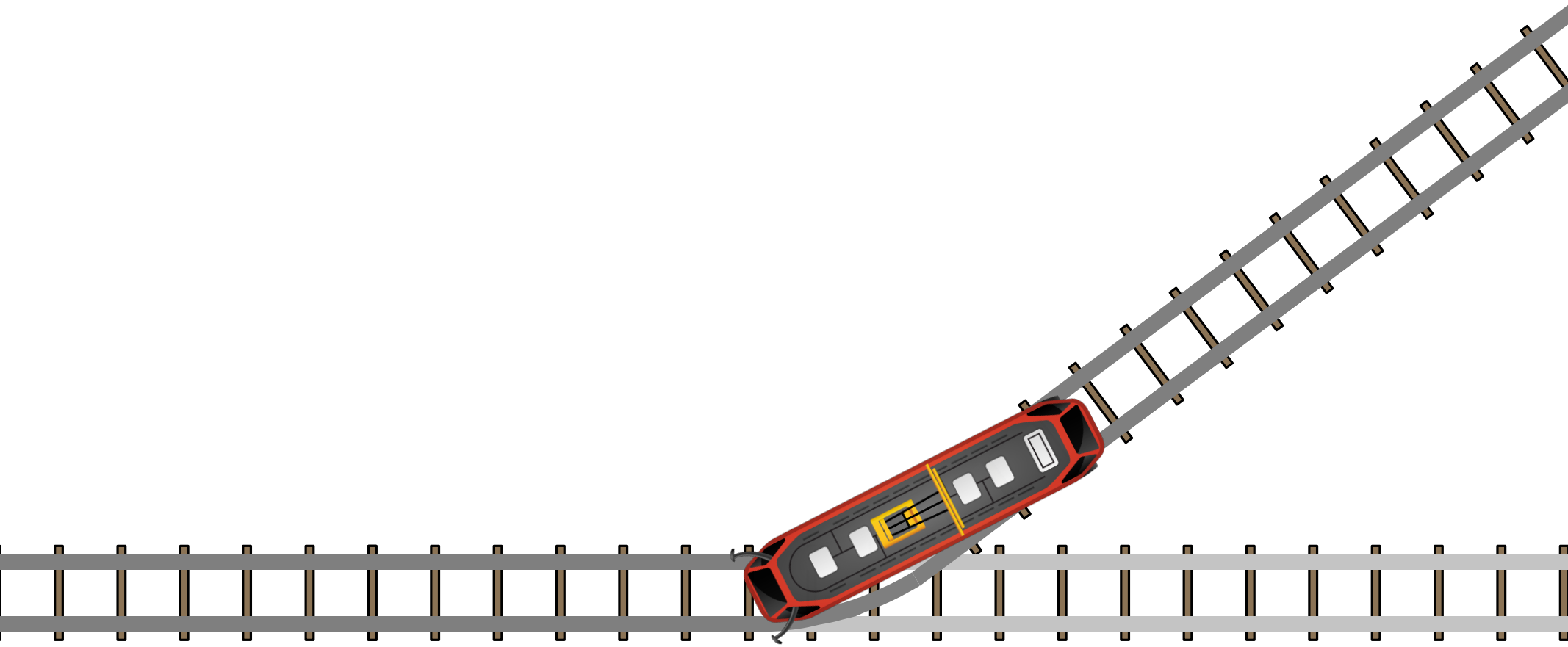
Designed by macrovector / Freepik



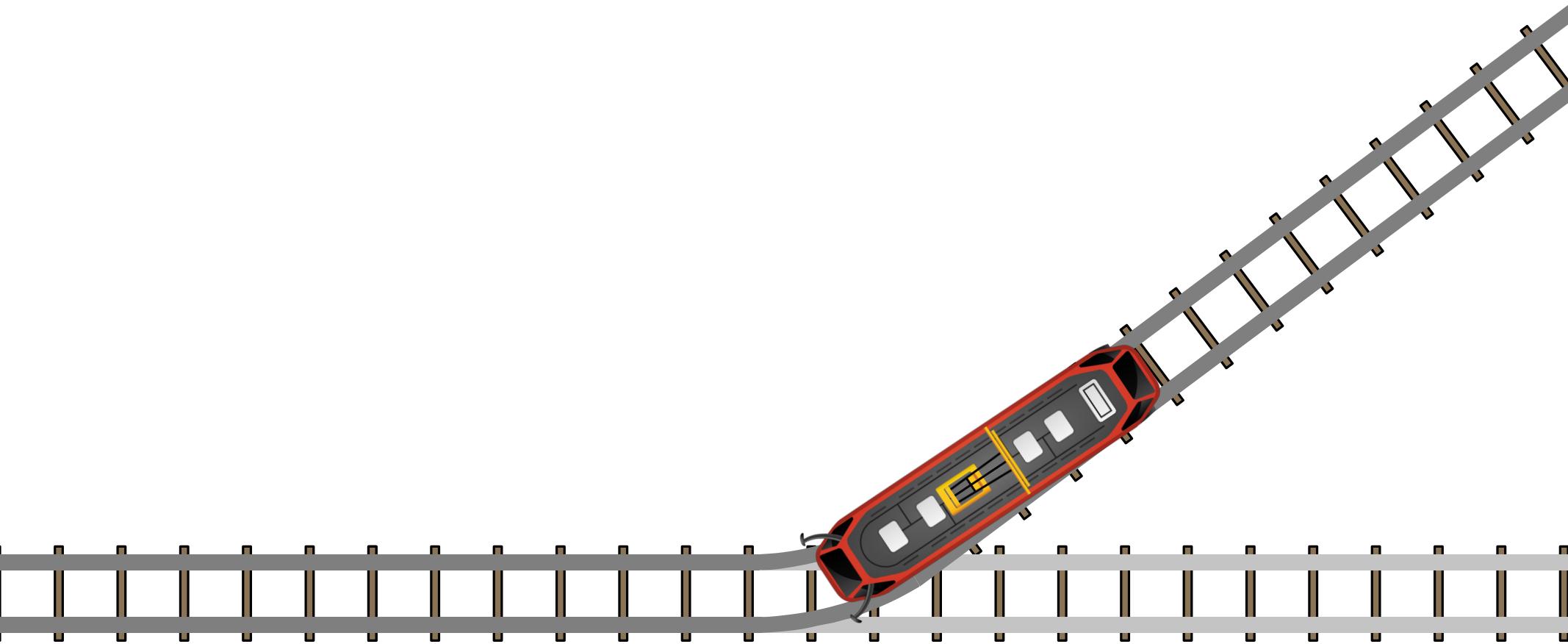
Designed by macrovector / Freepik

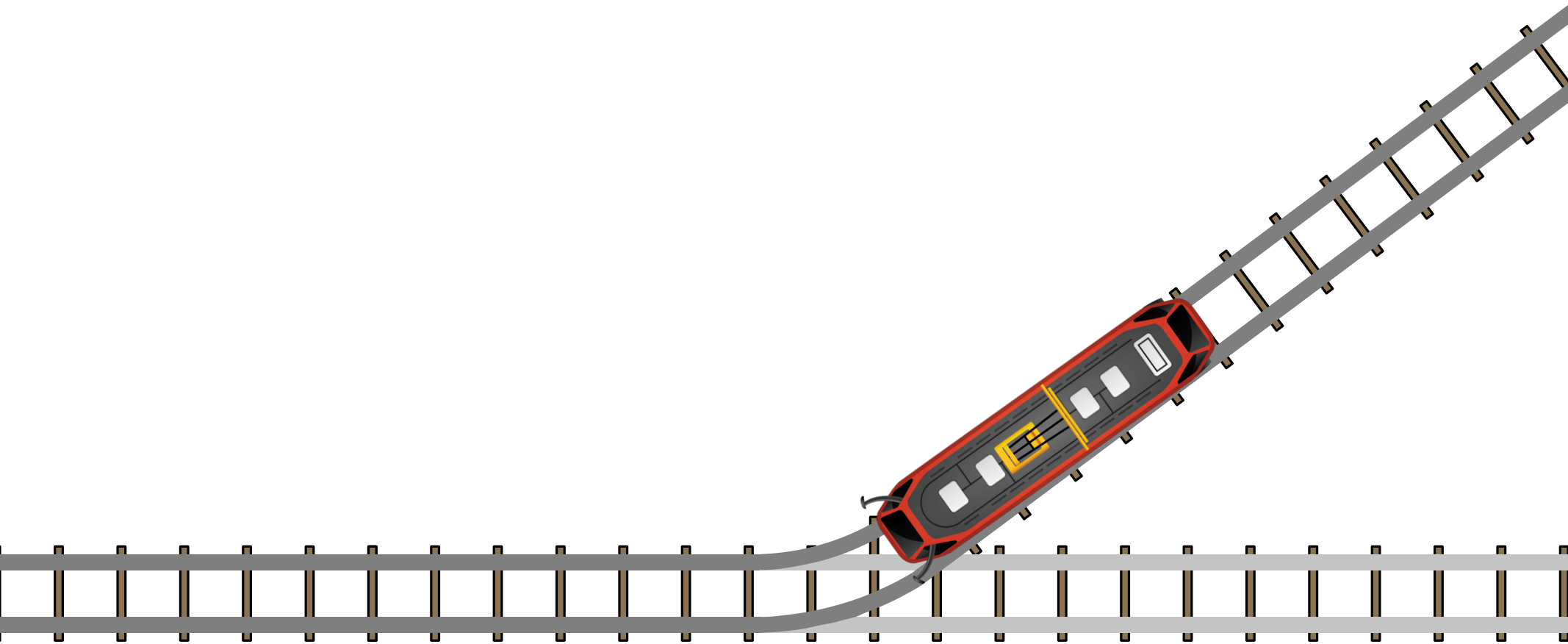


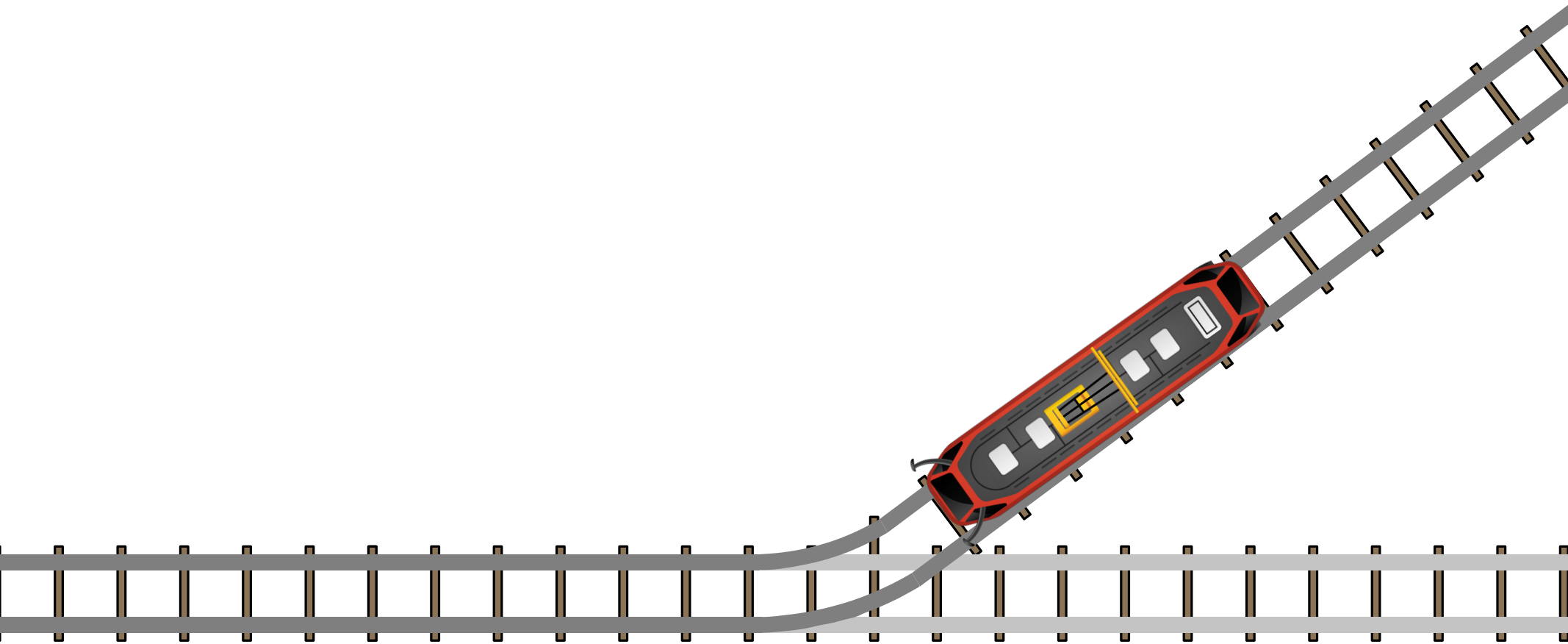
Designed by macrovector / Freepik

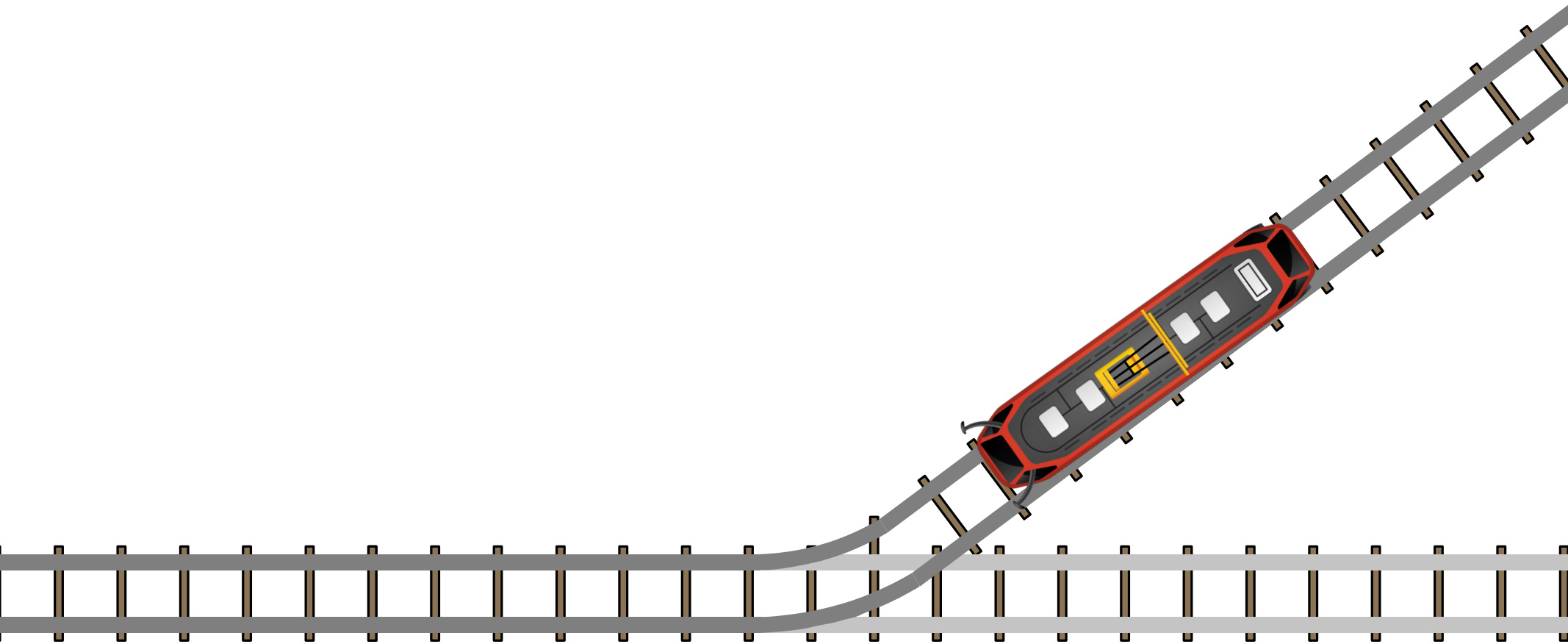


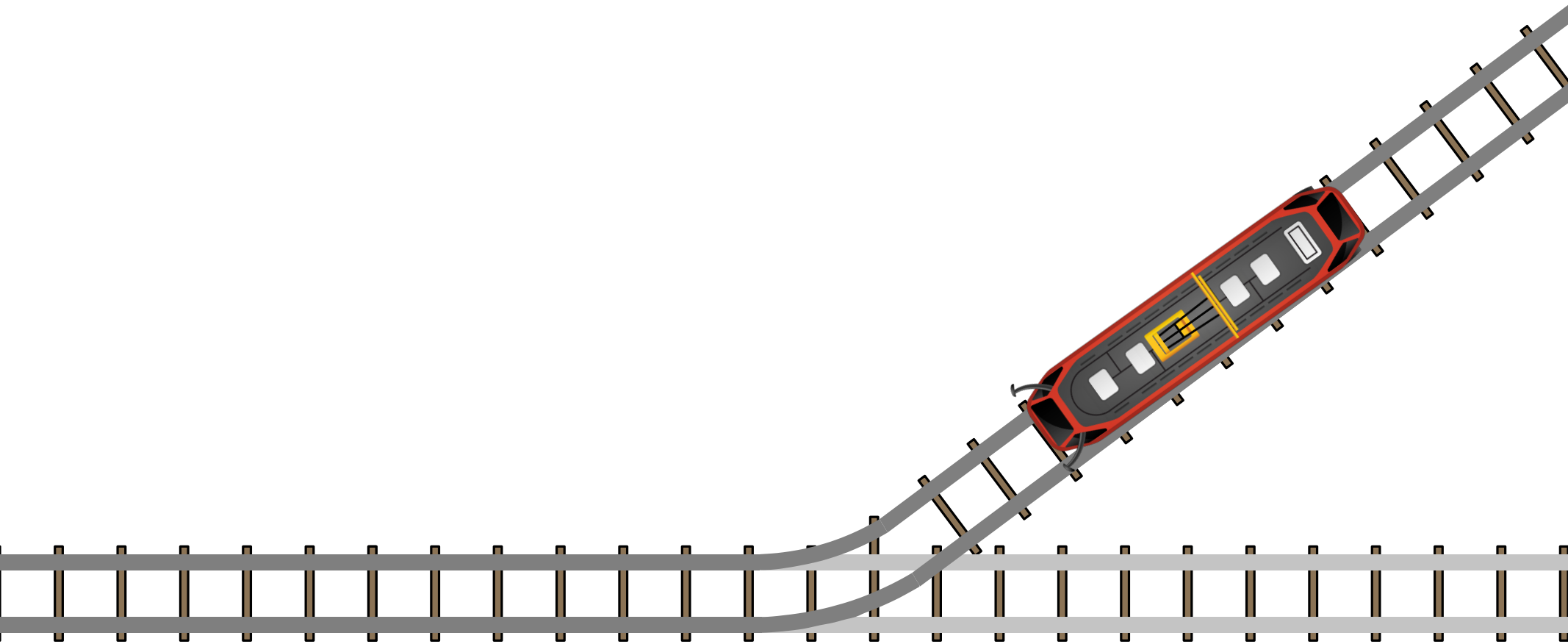
Designed by macrovector / Freepik

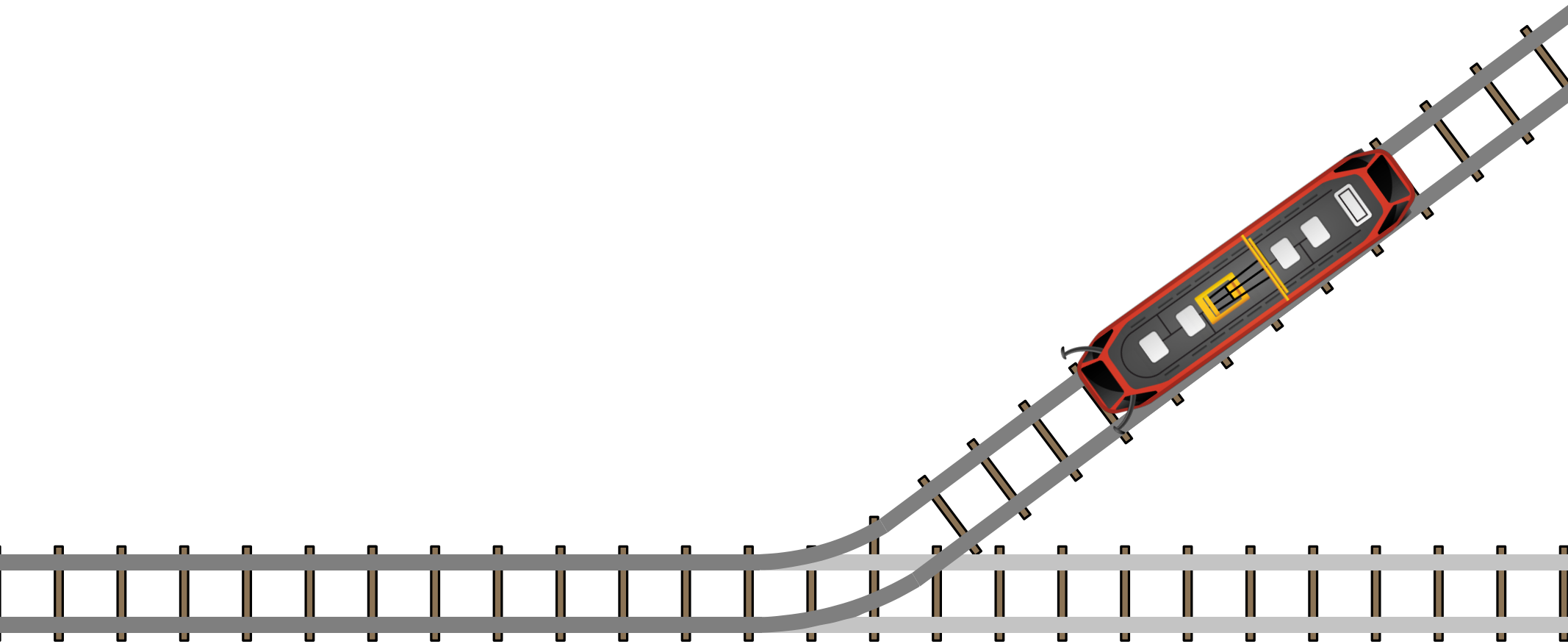


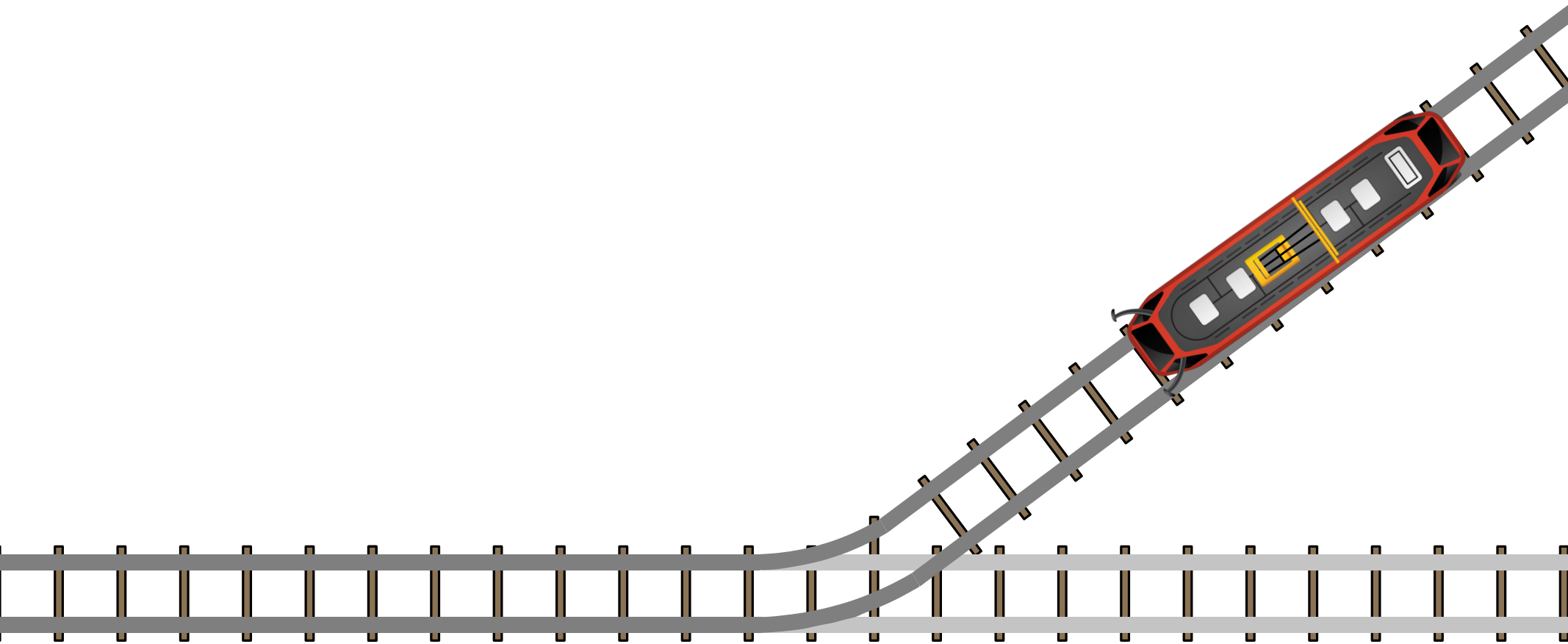


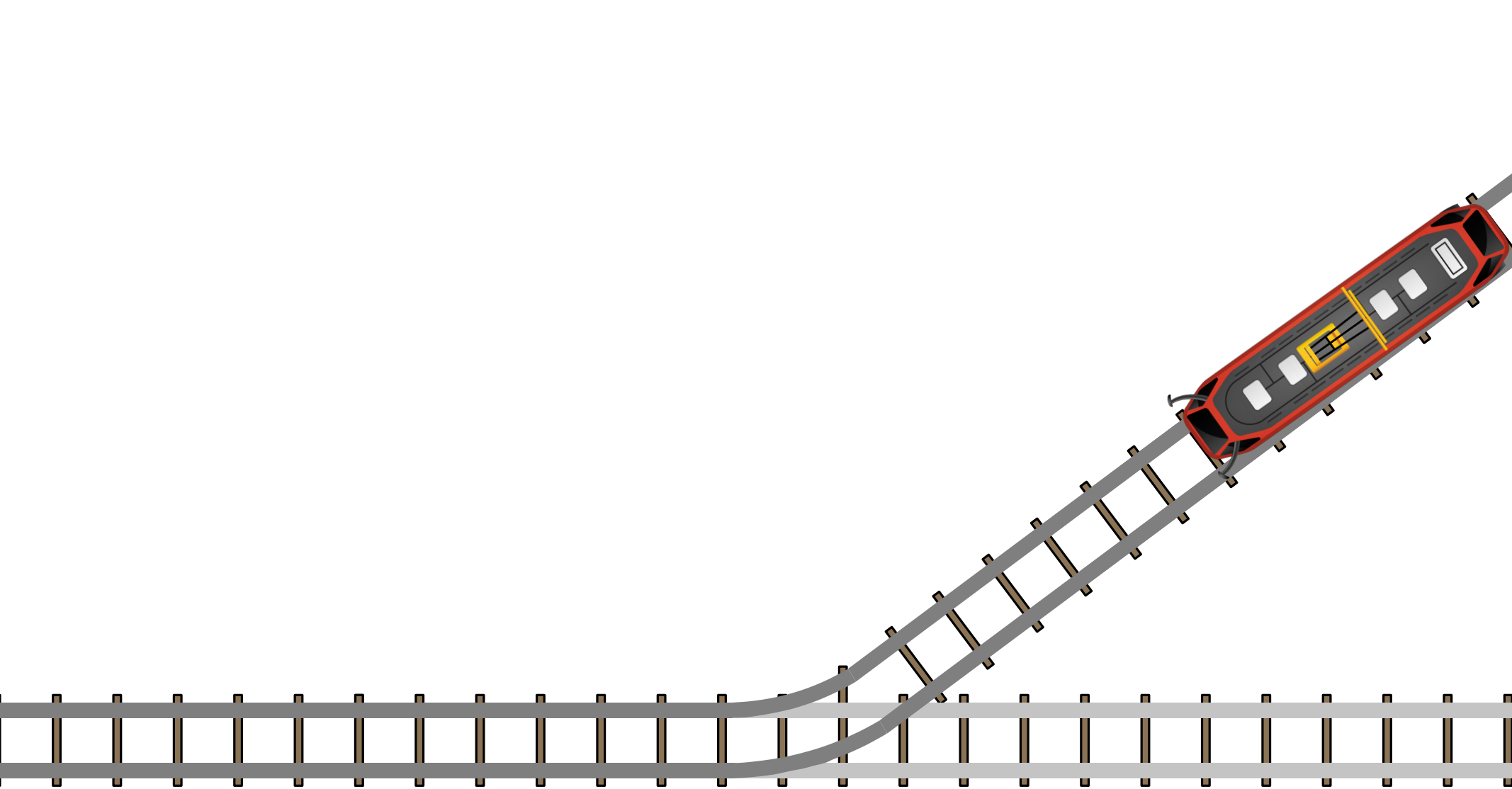


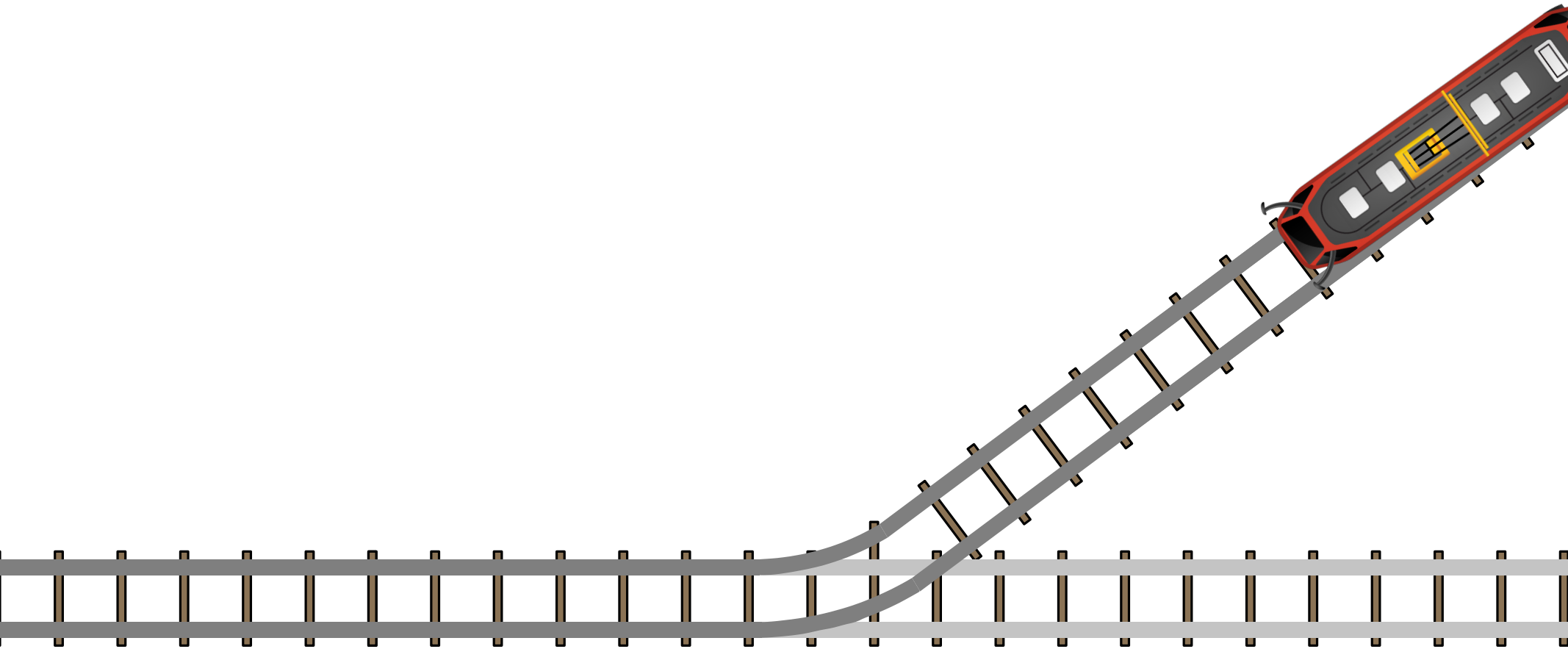


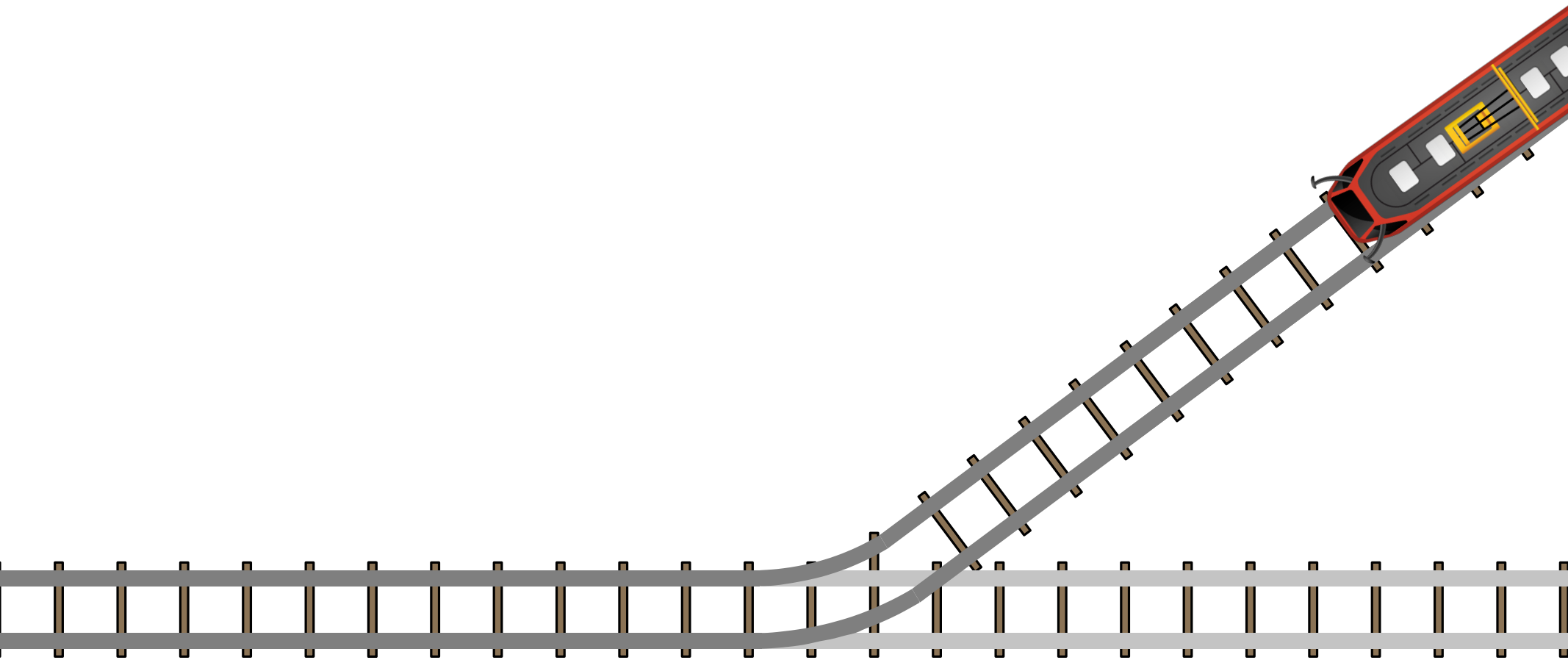


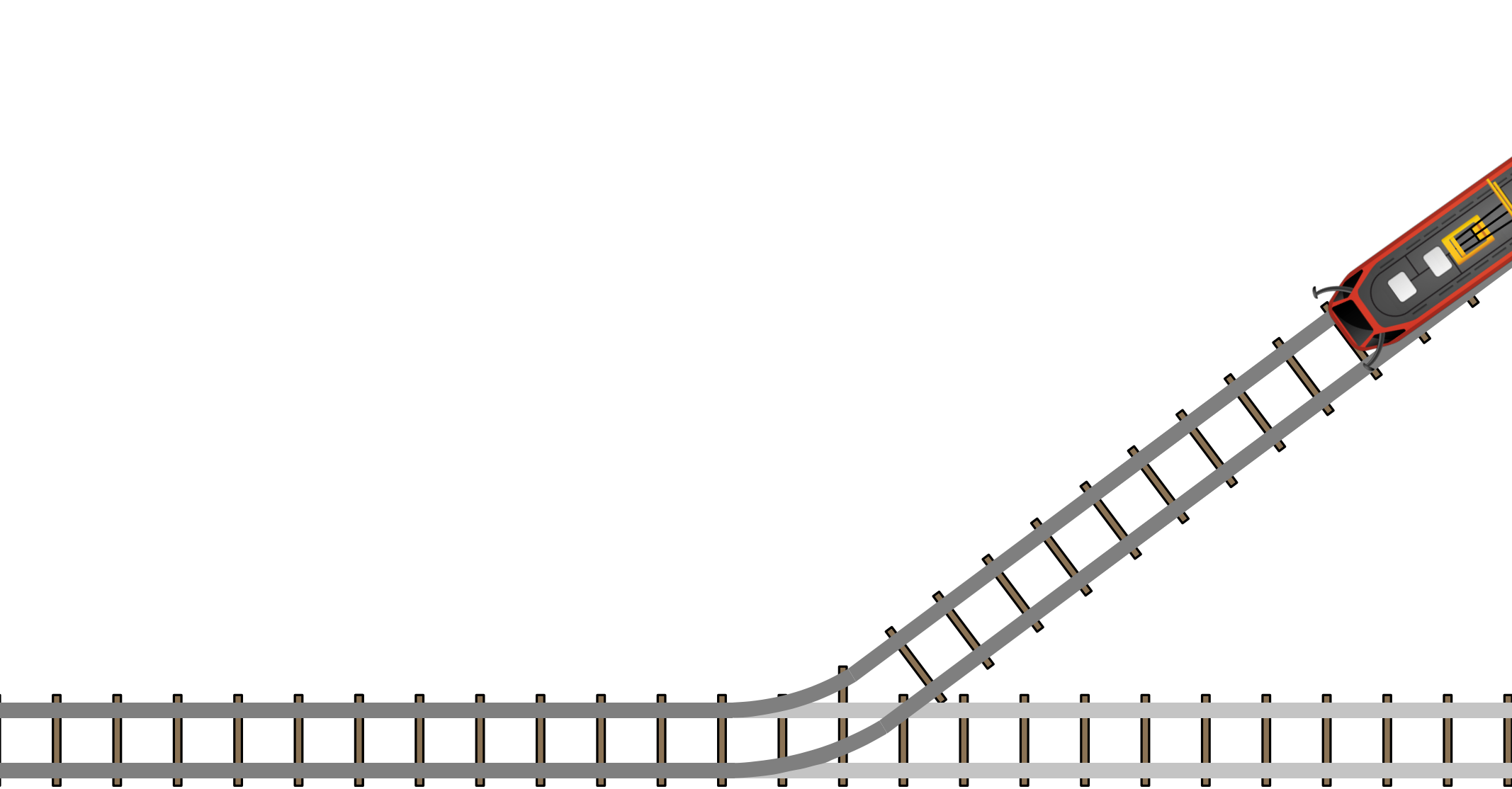


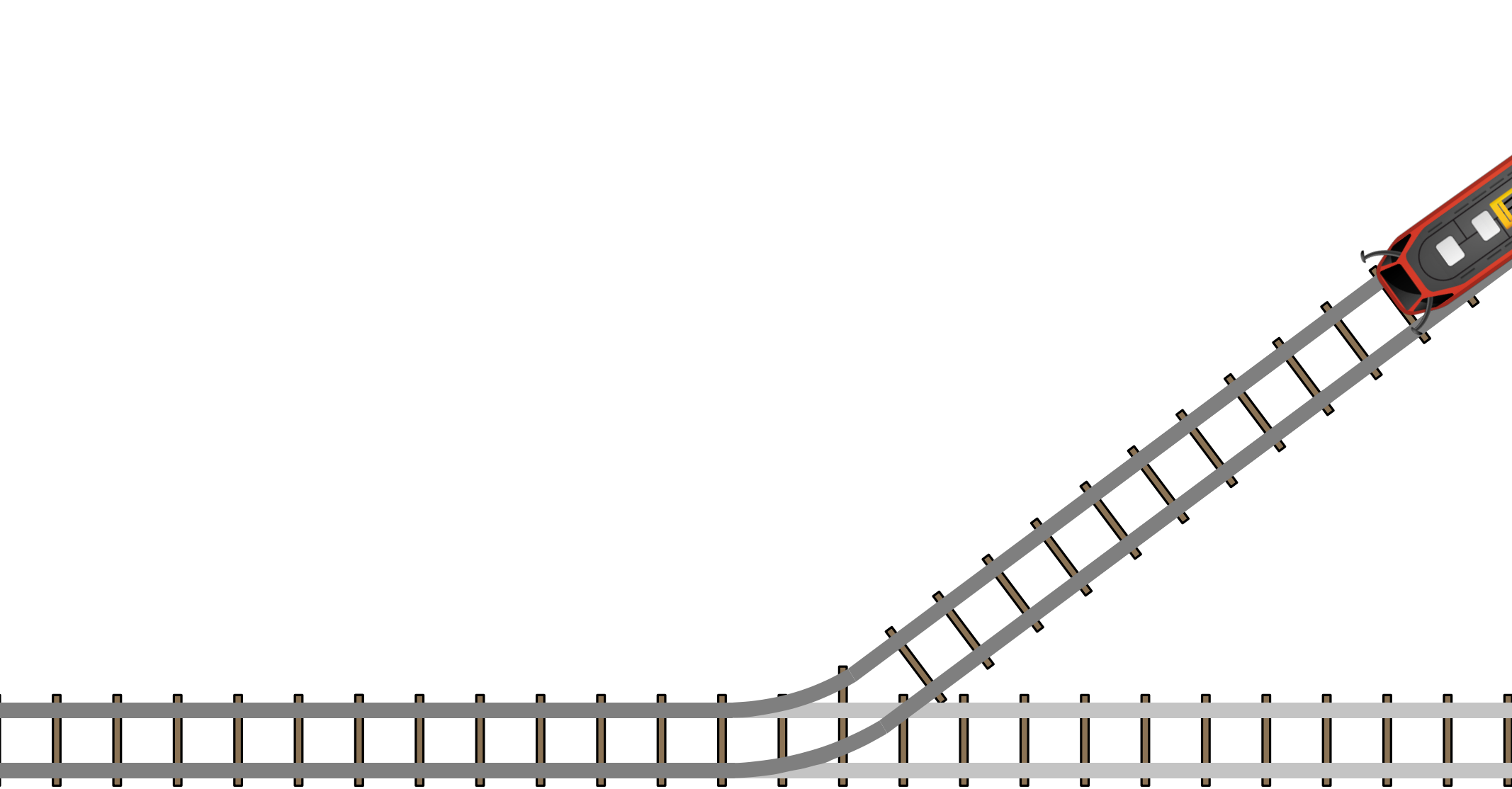


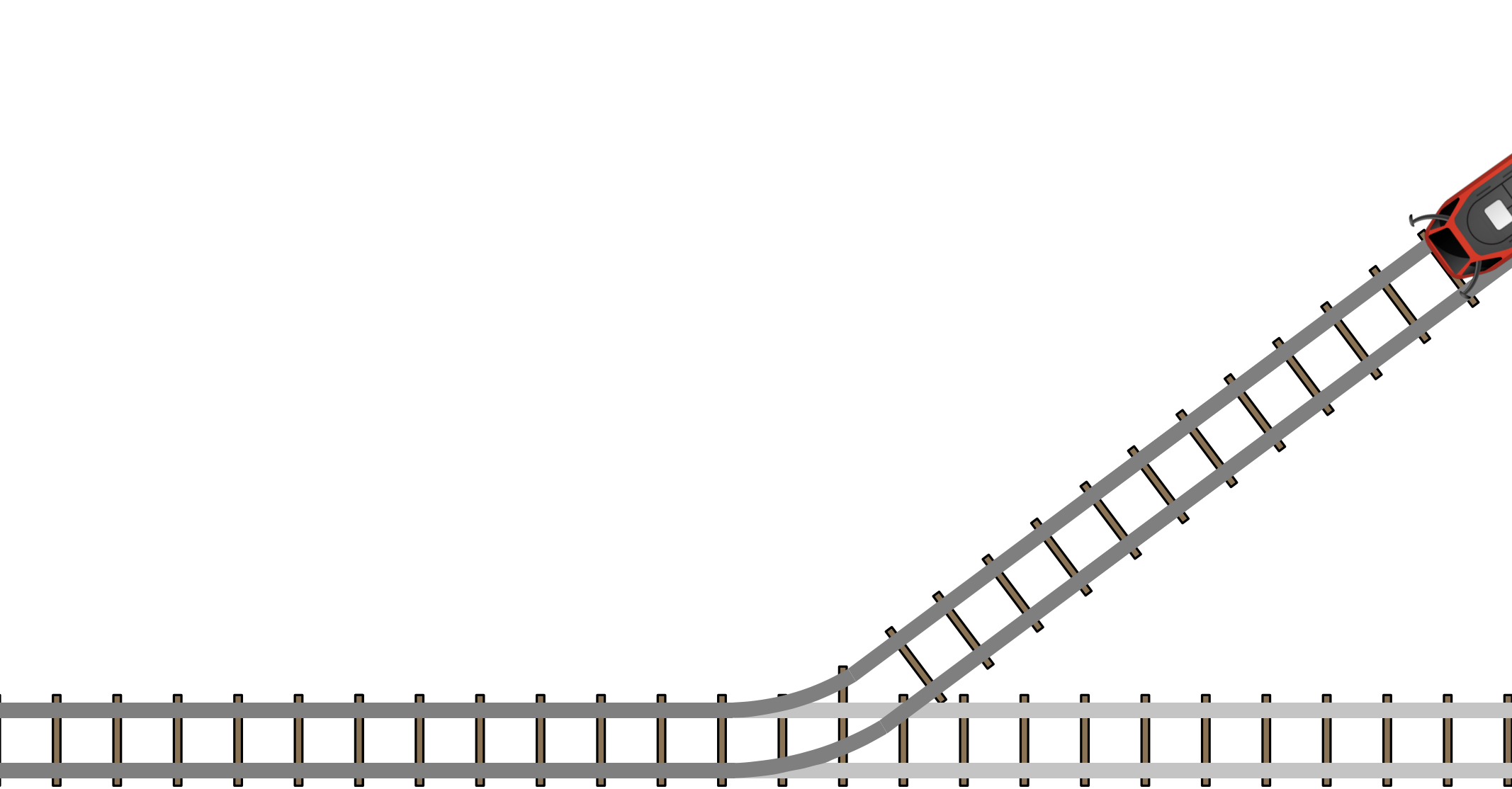


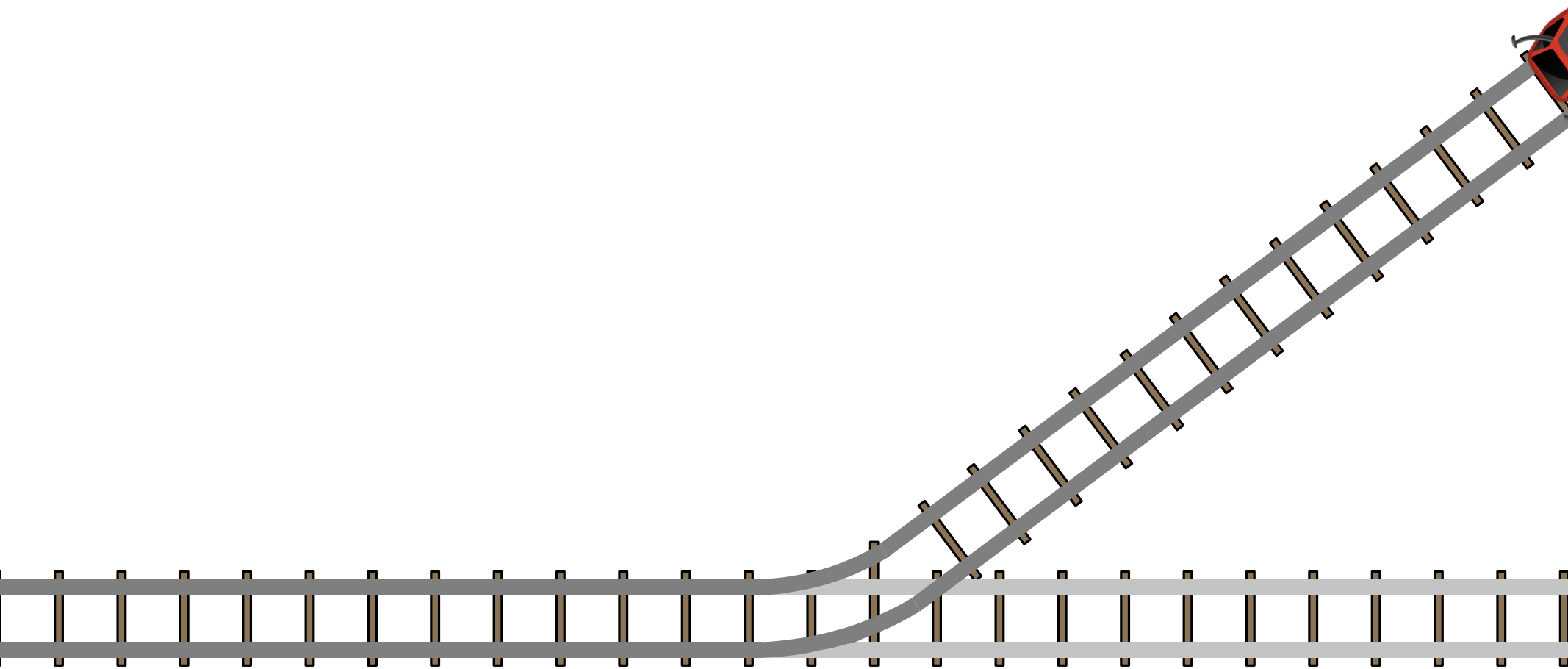


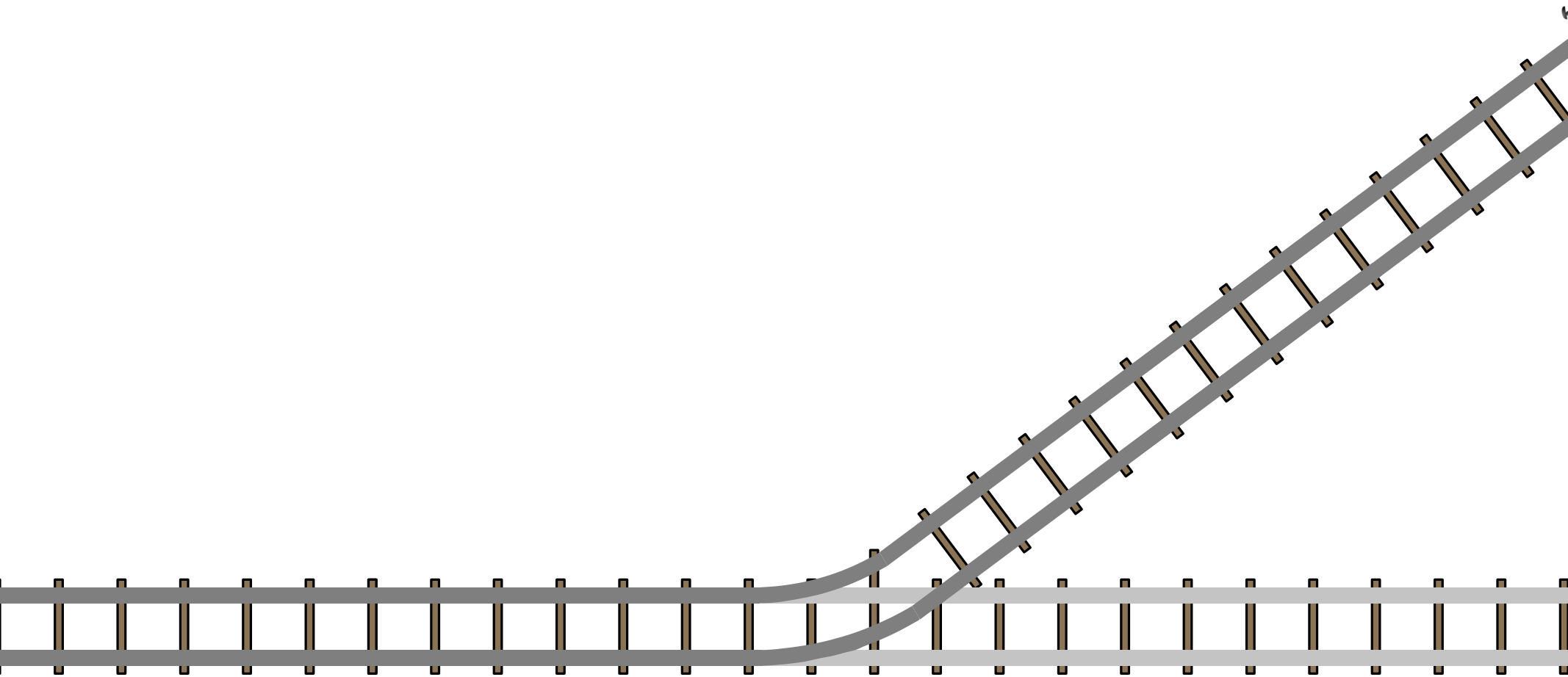


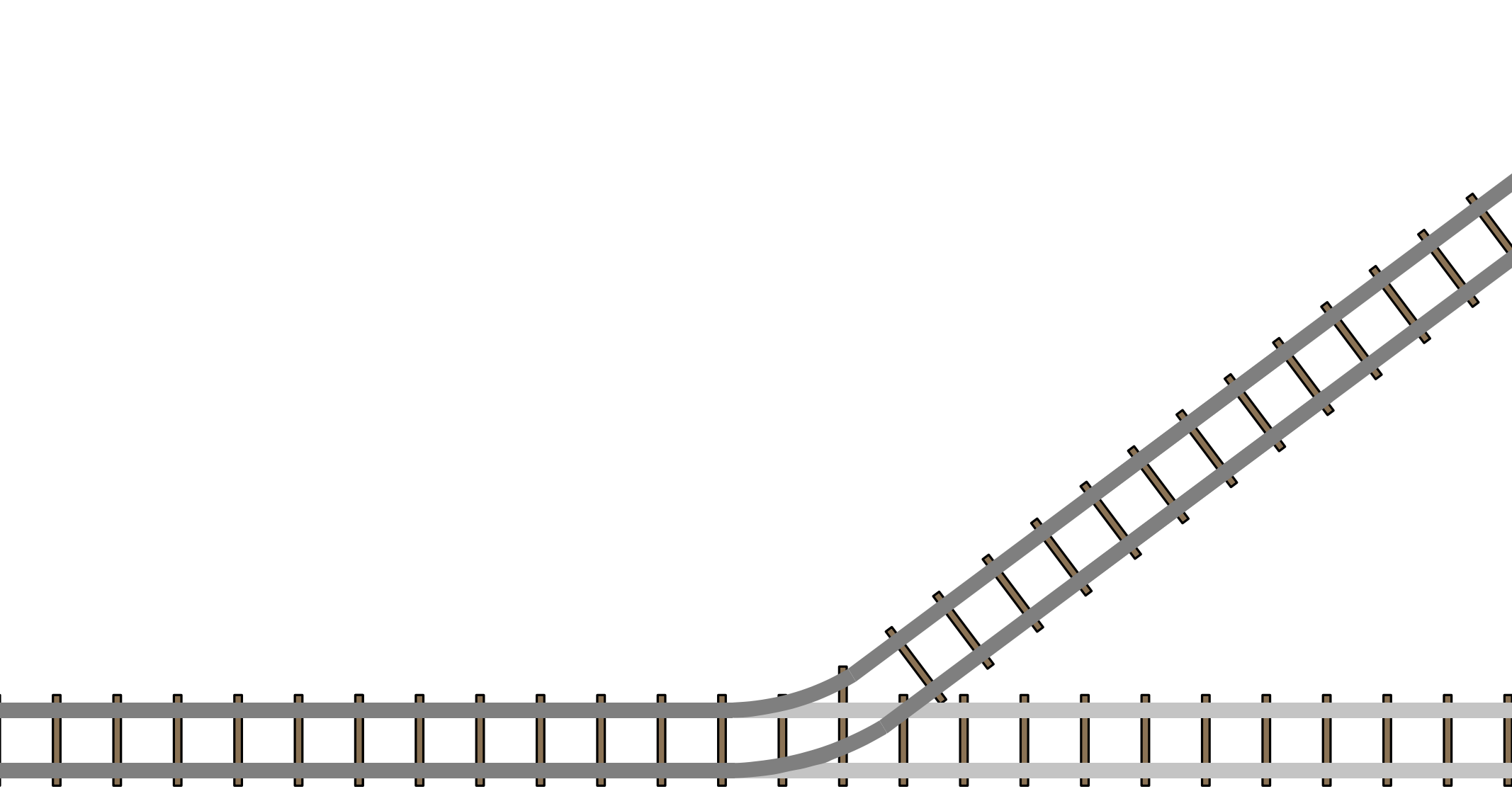


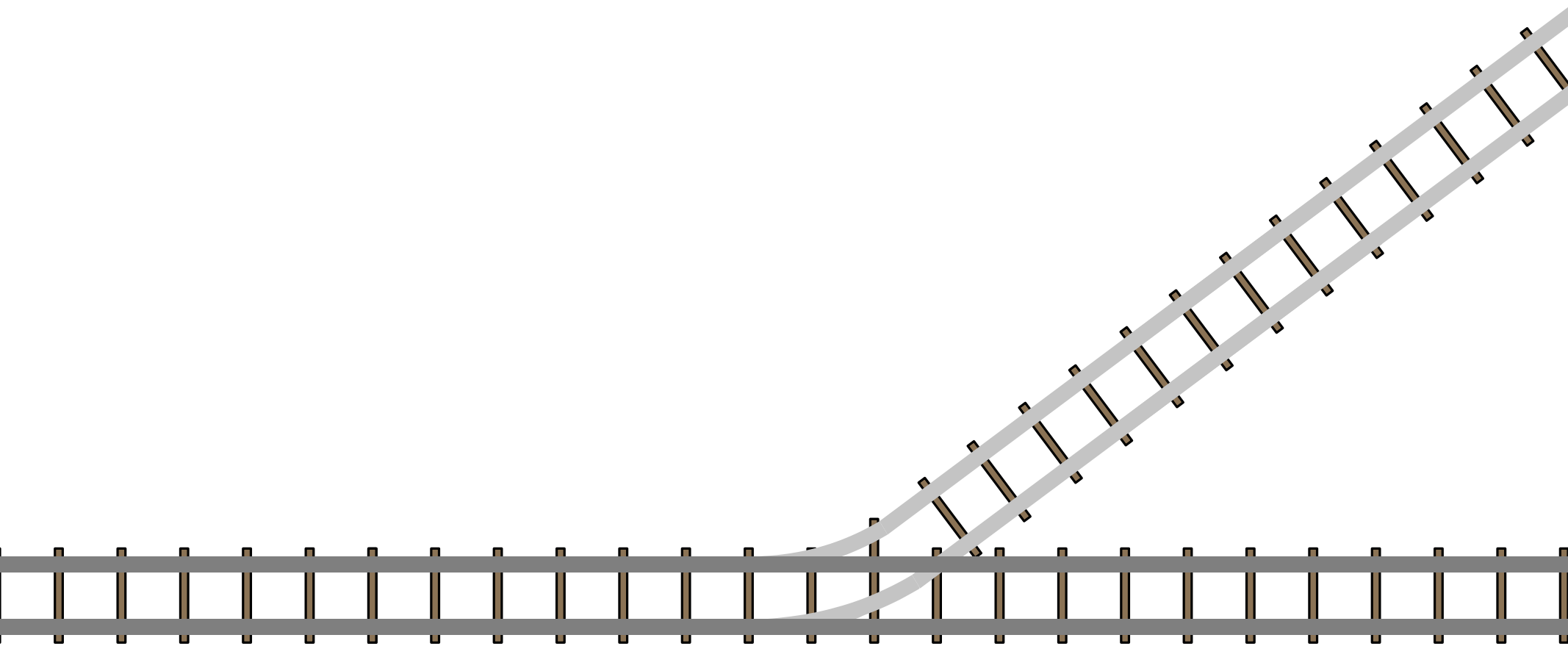






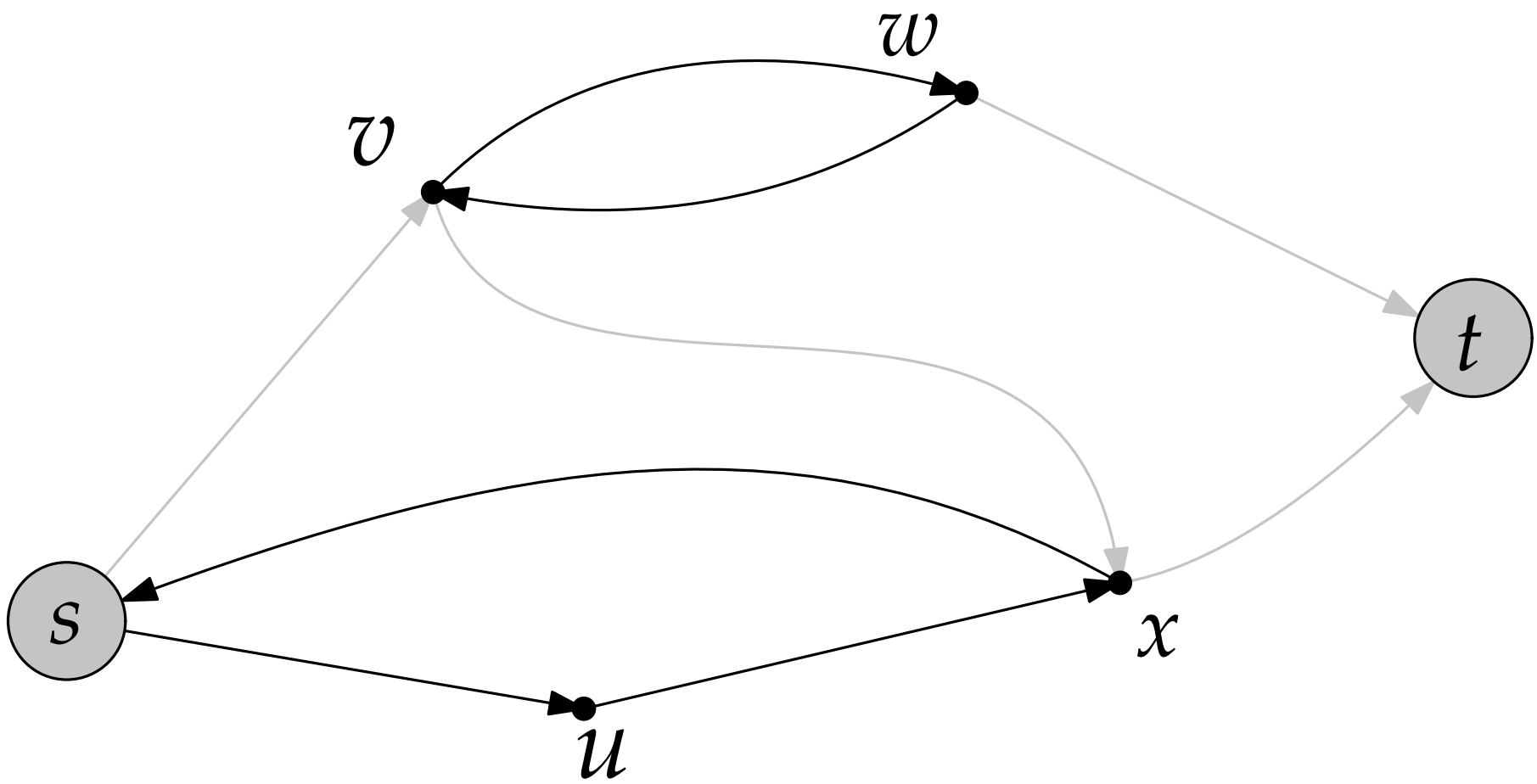






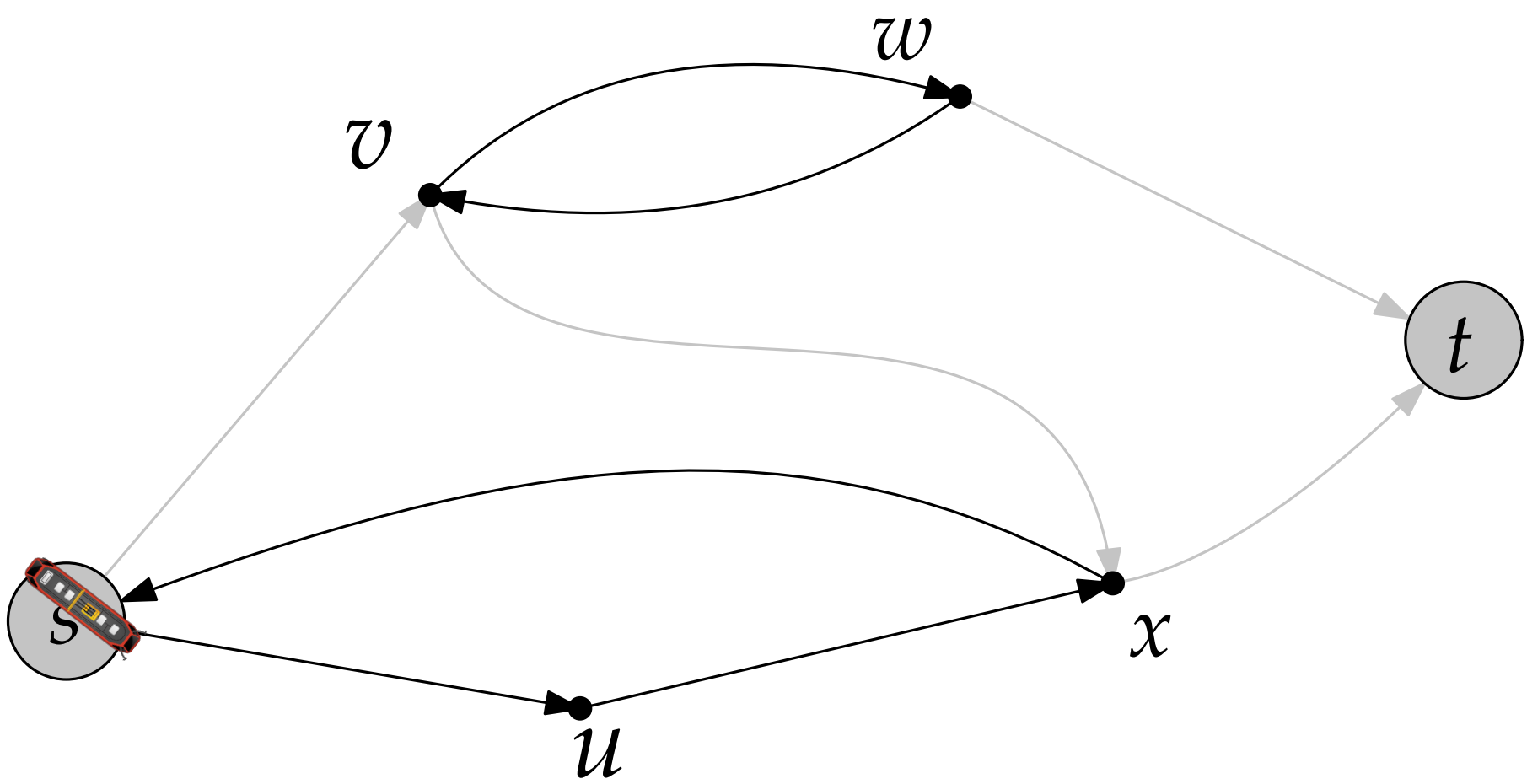
Arrival

Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Arrival

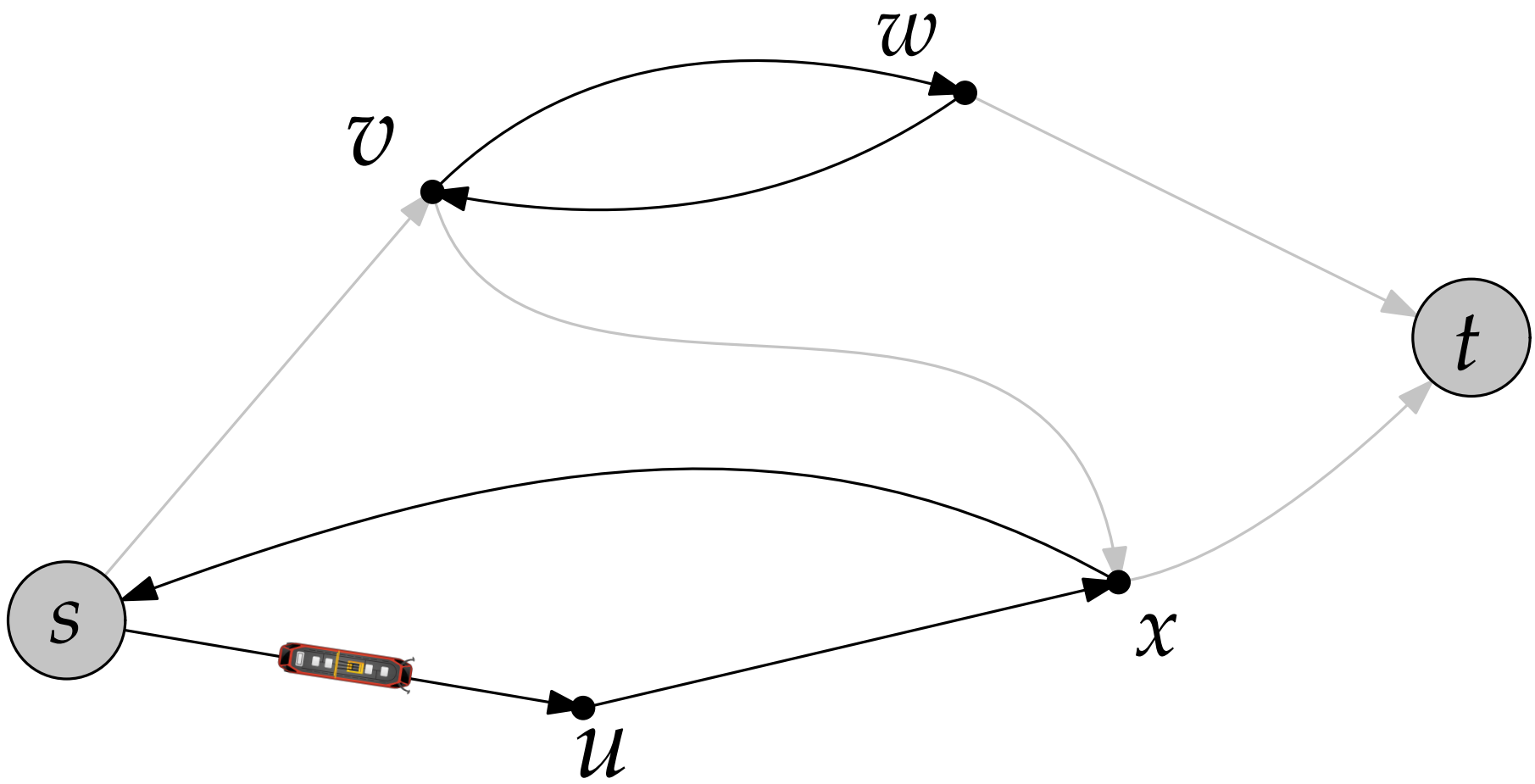
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

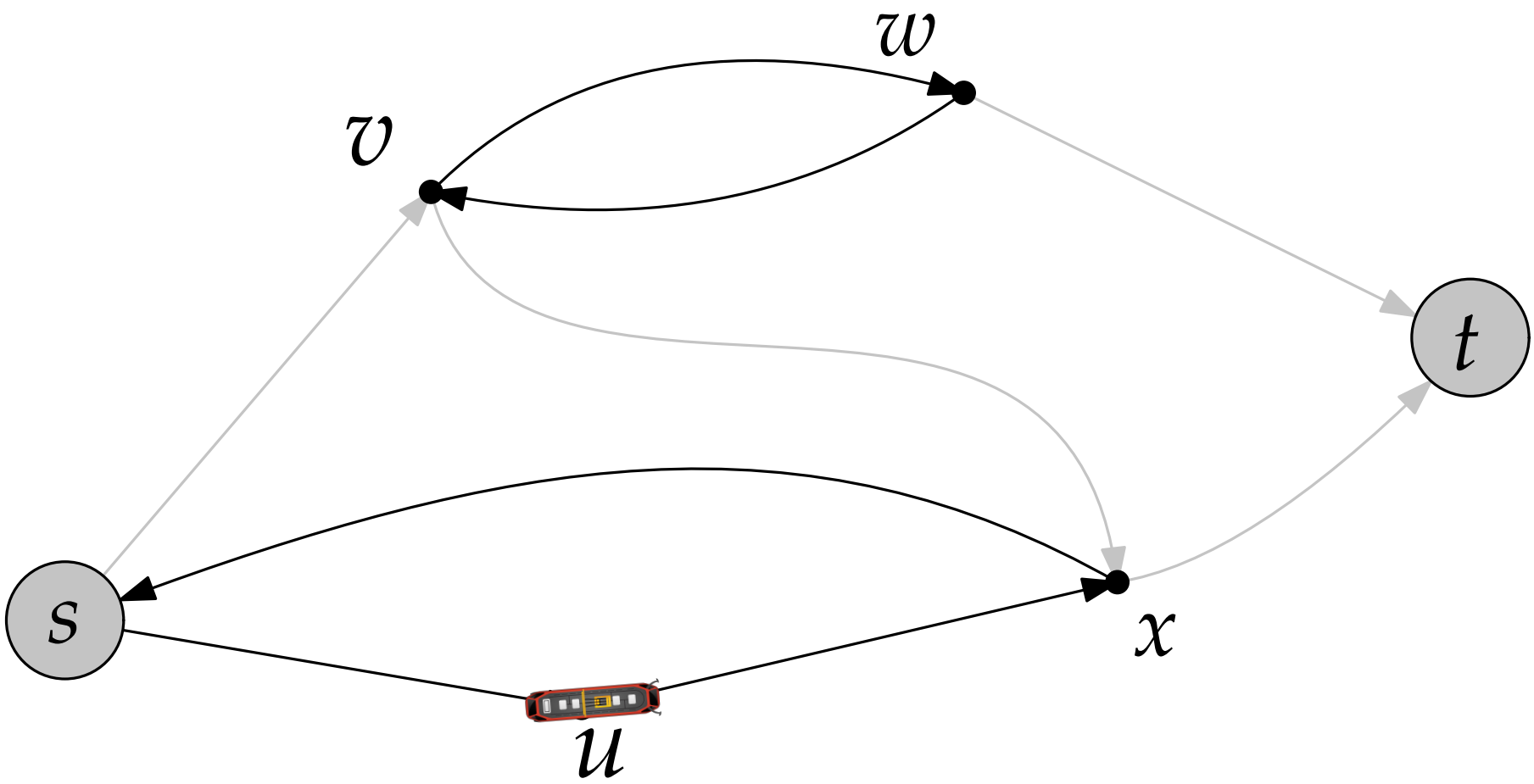
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

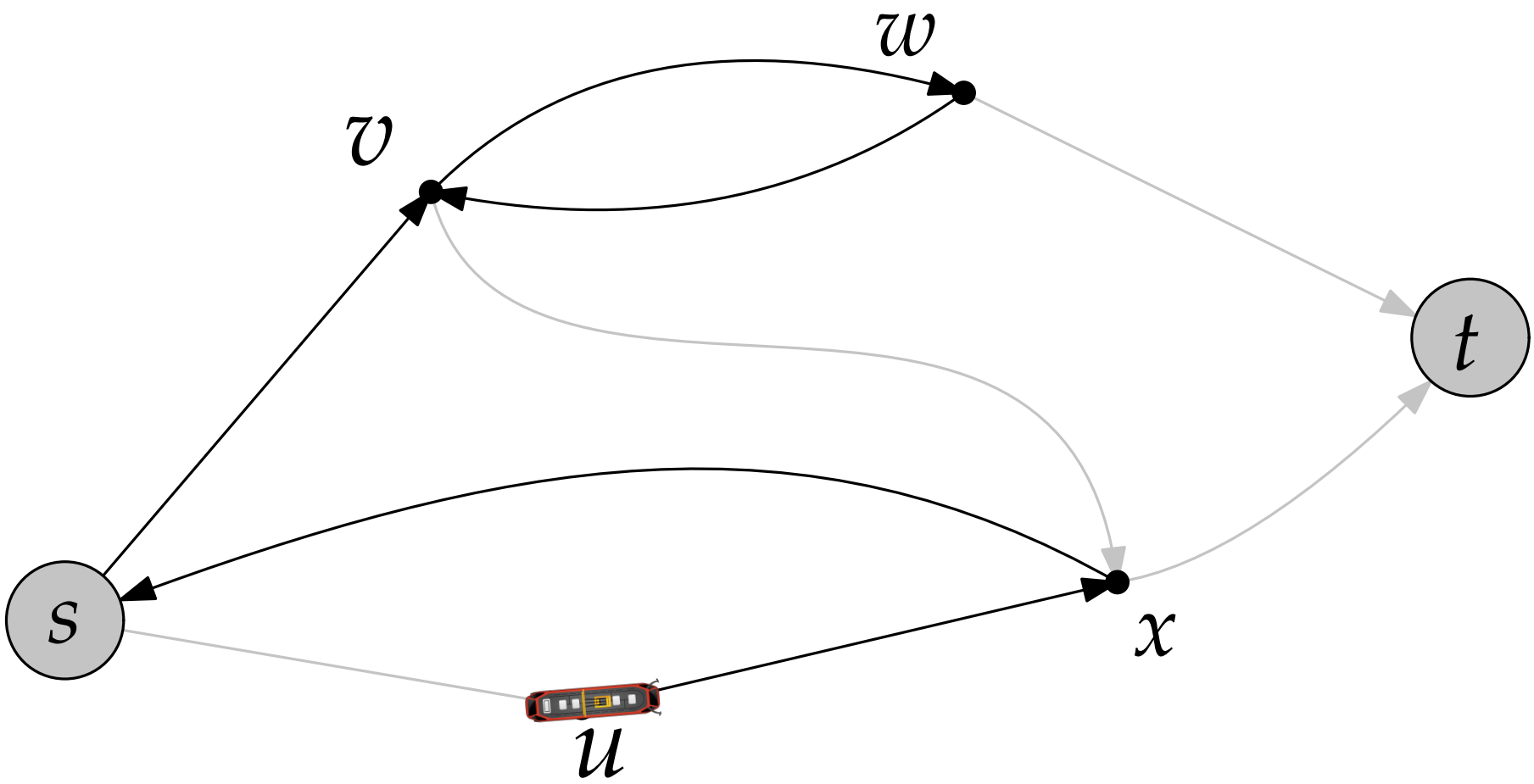
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

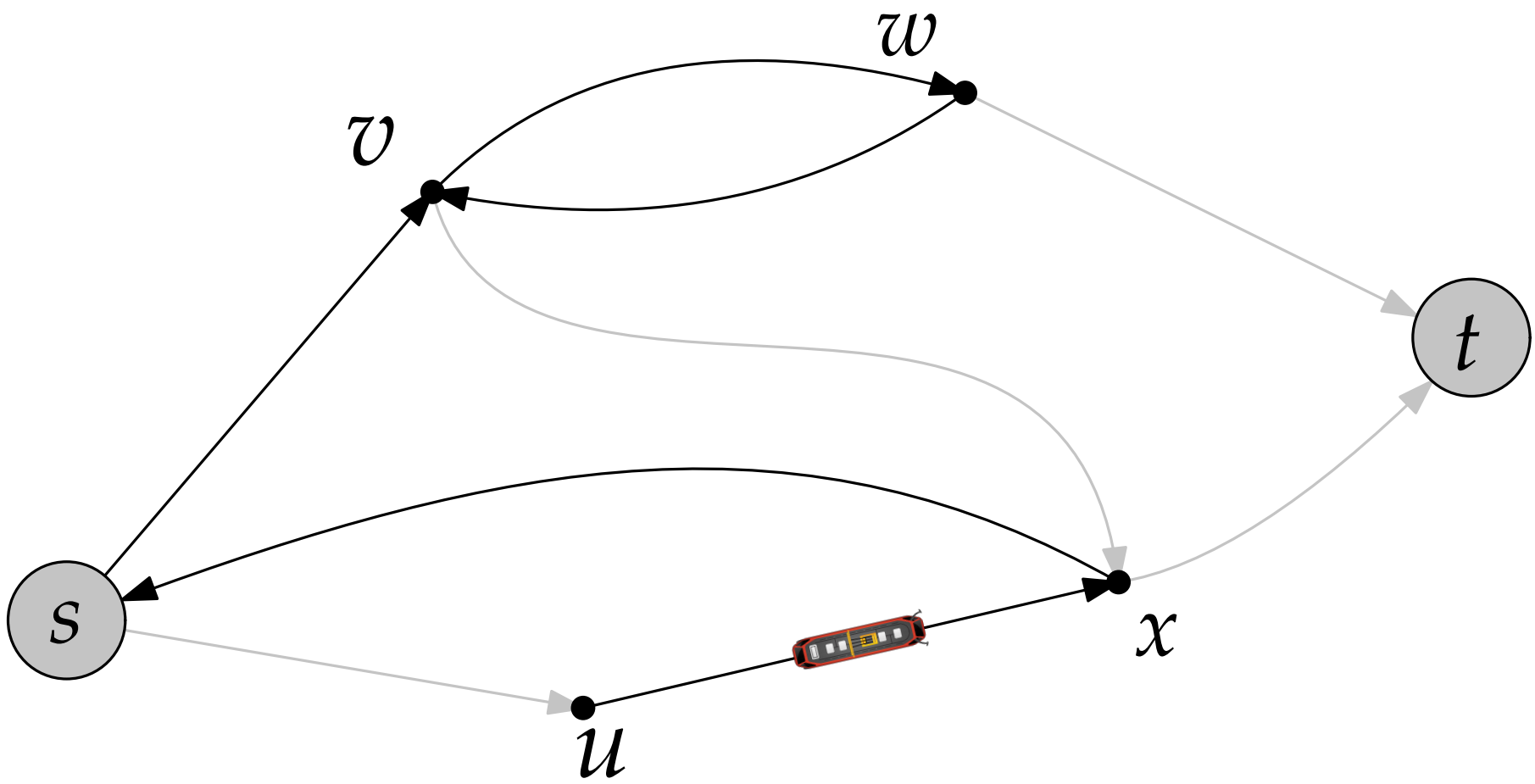
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

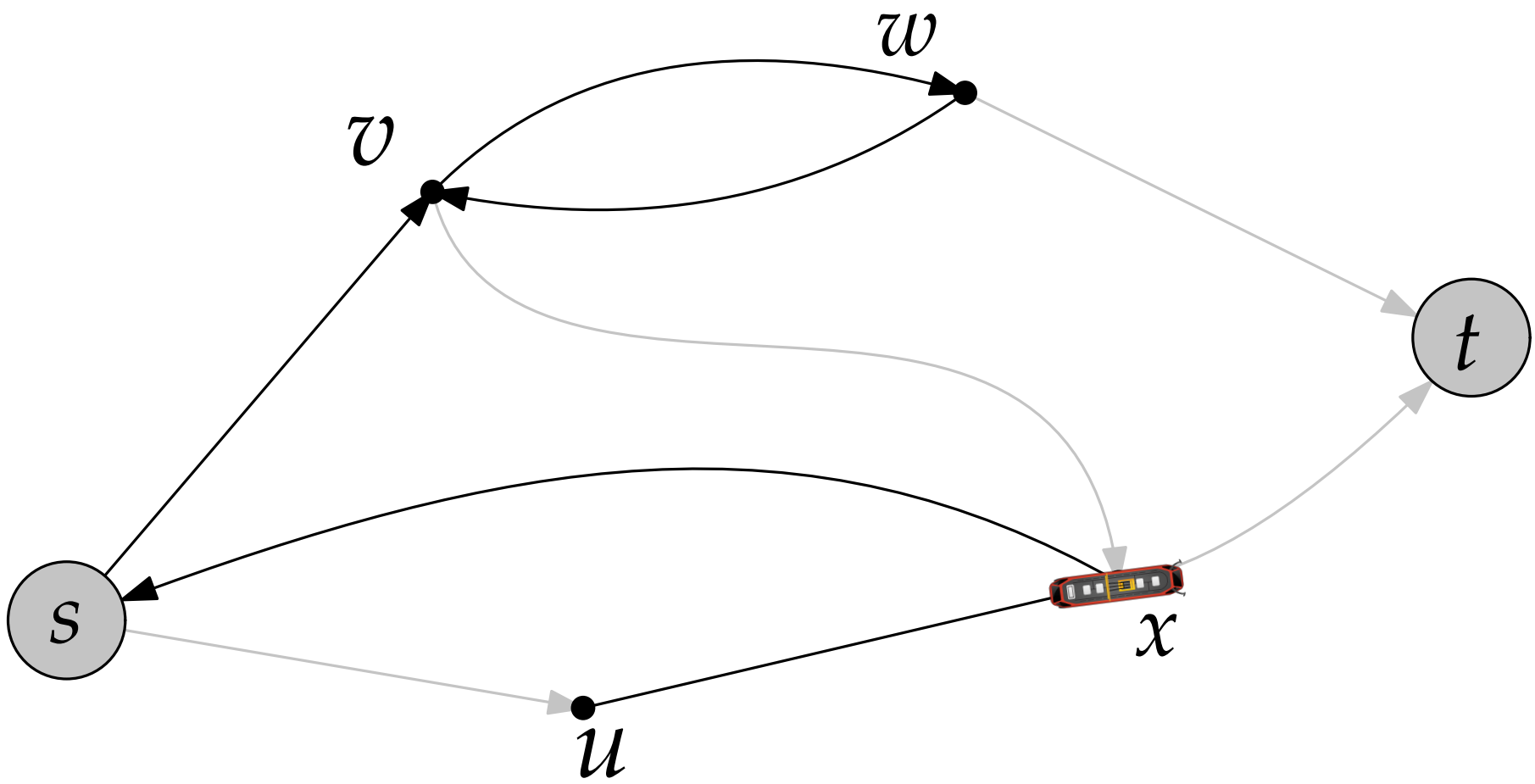
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

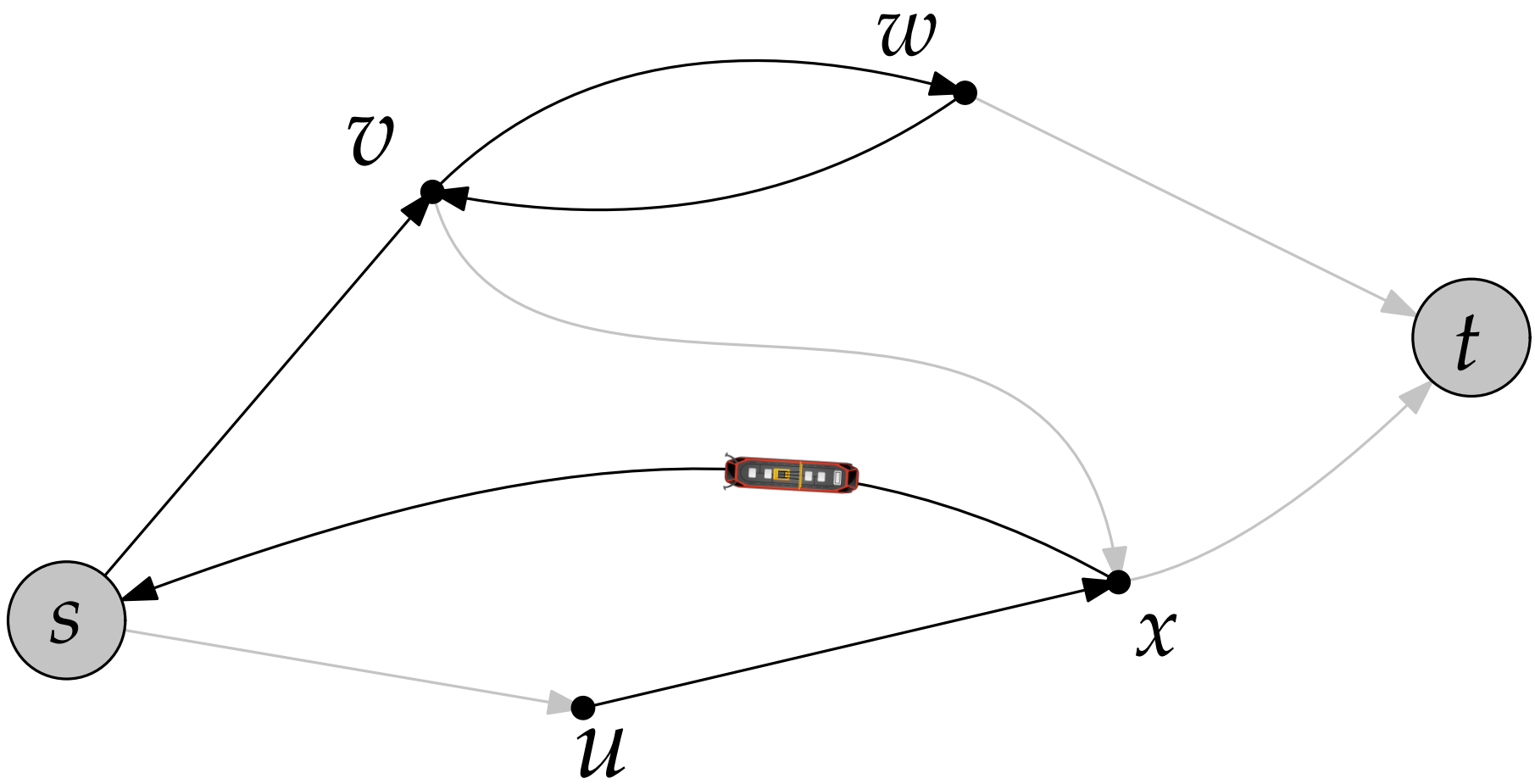
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

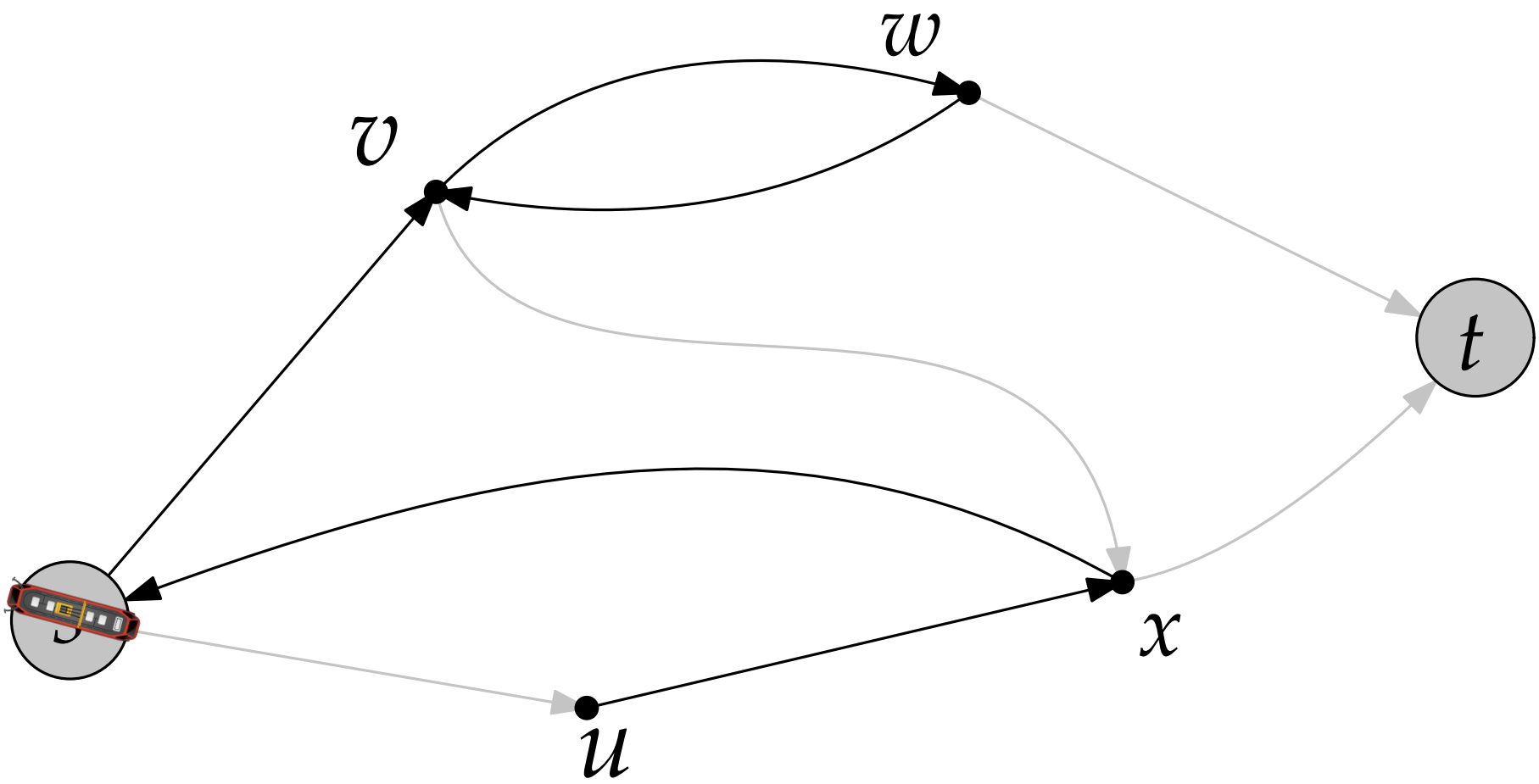
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

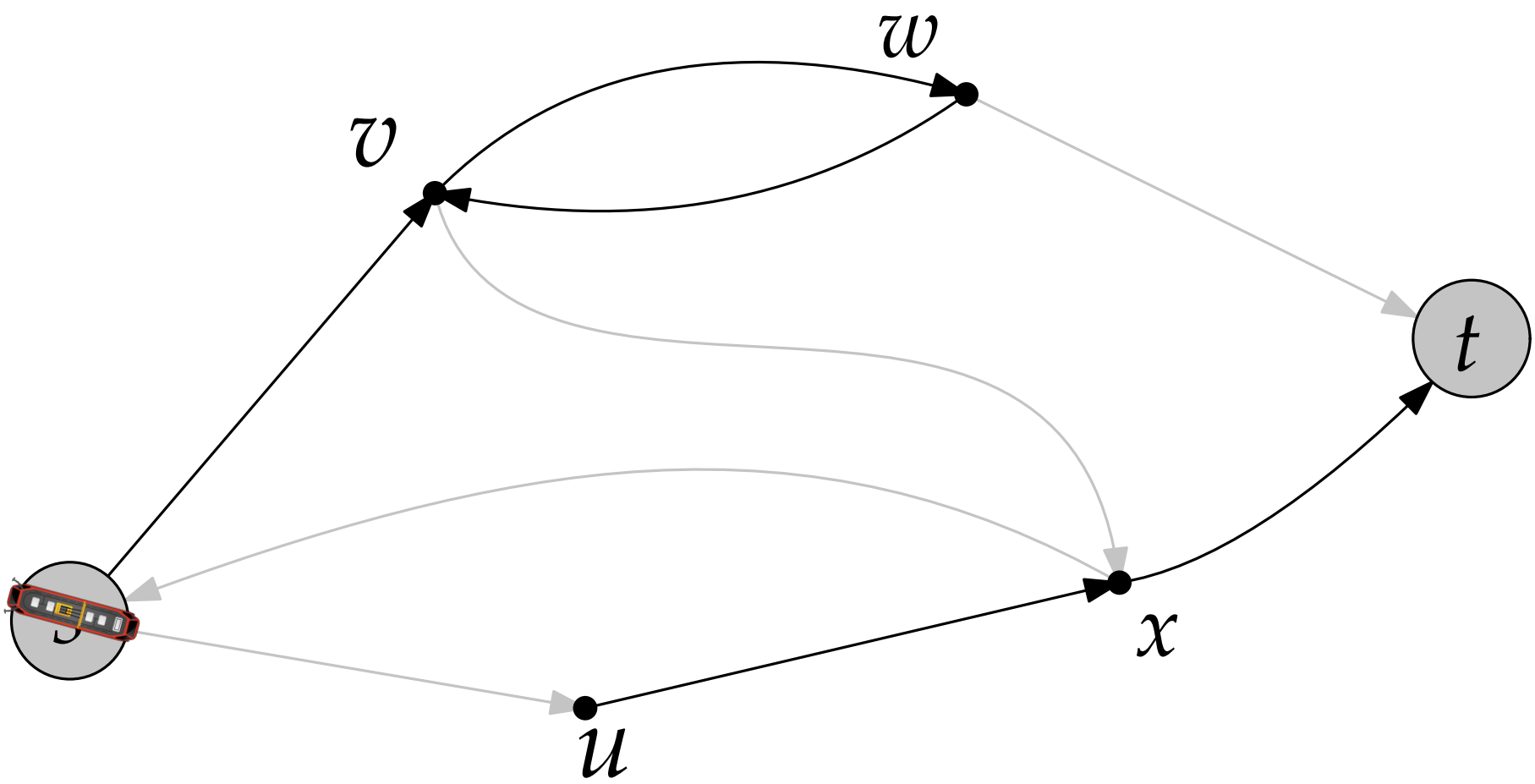
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

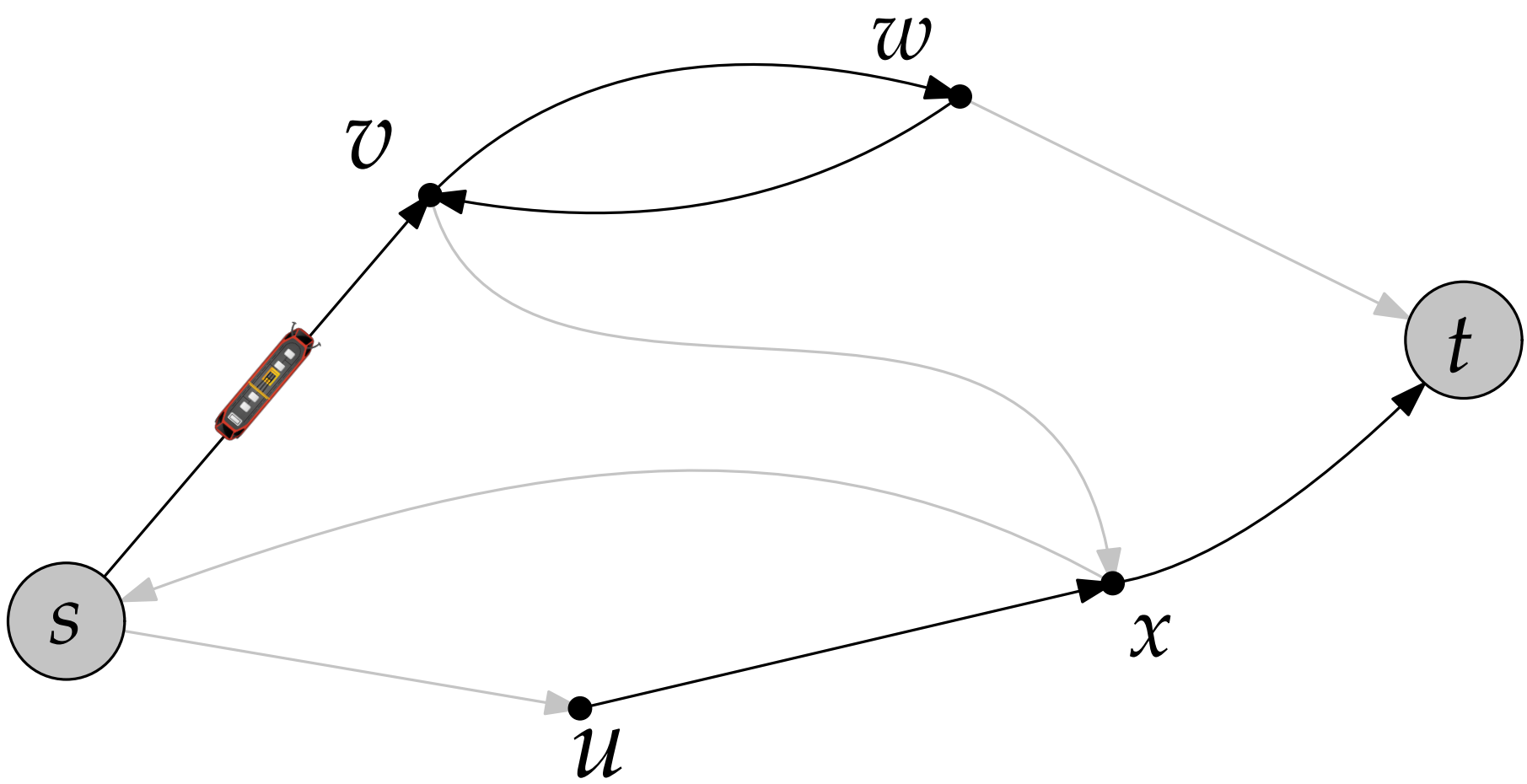
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

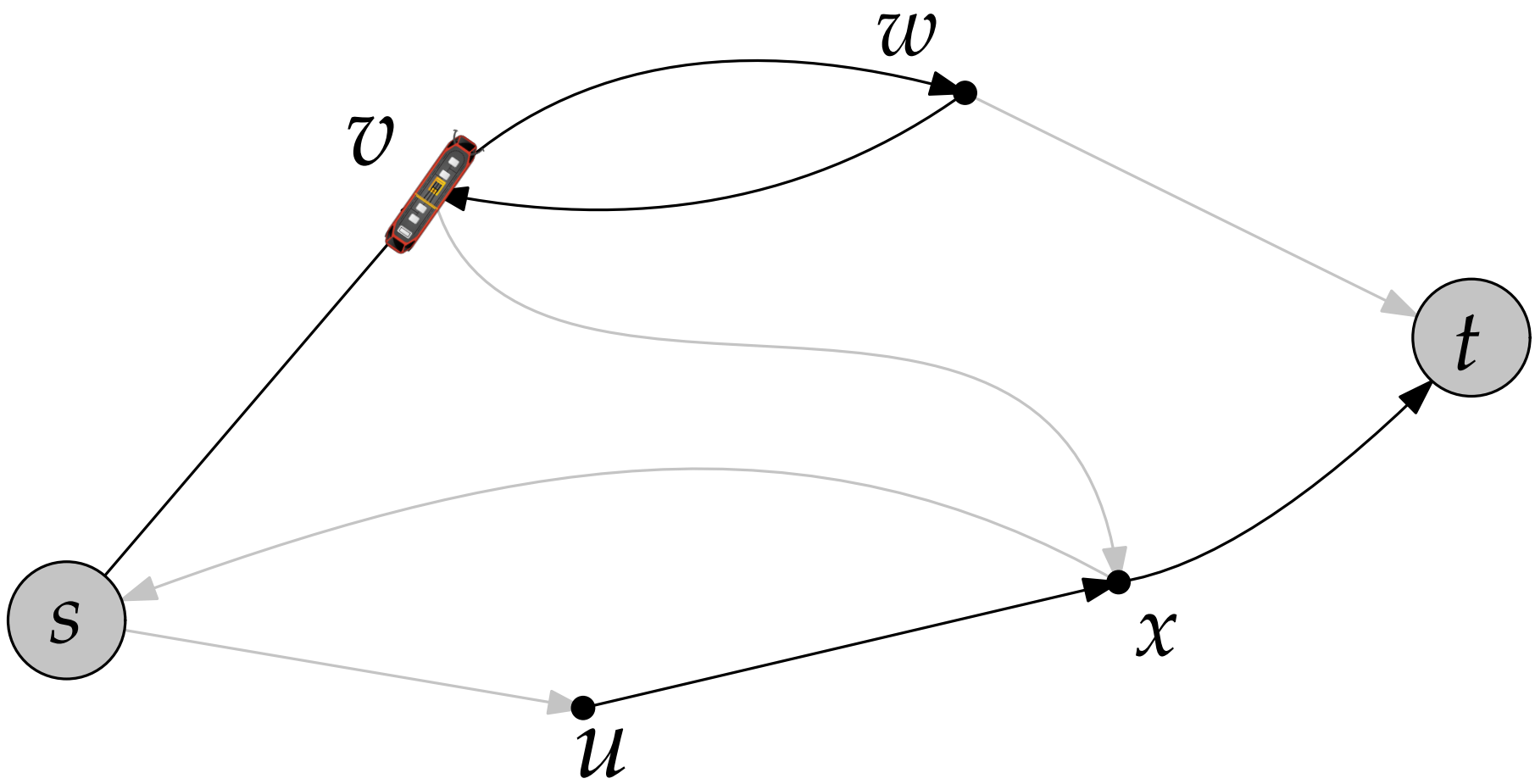
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

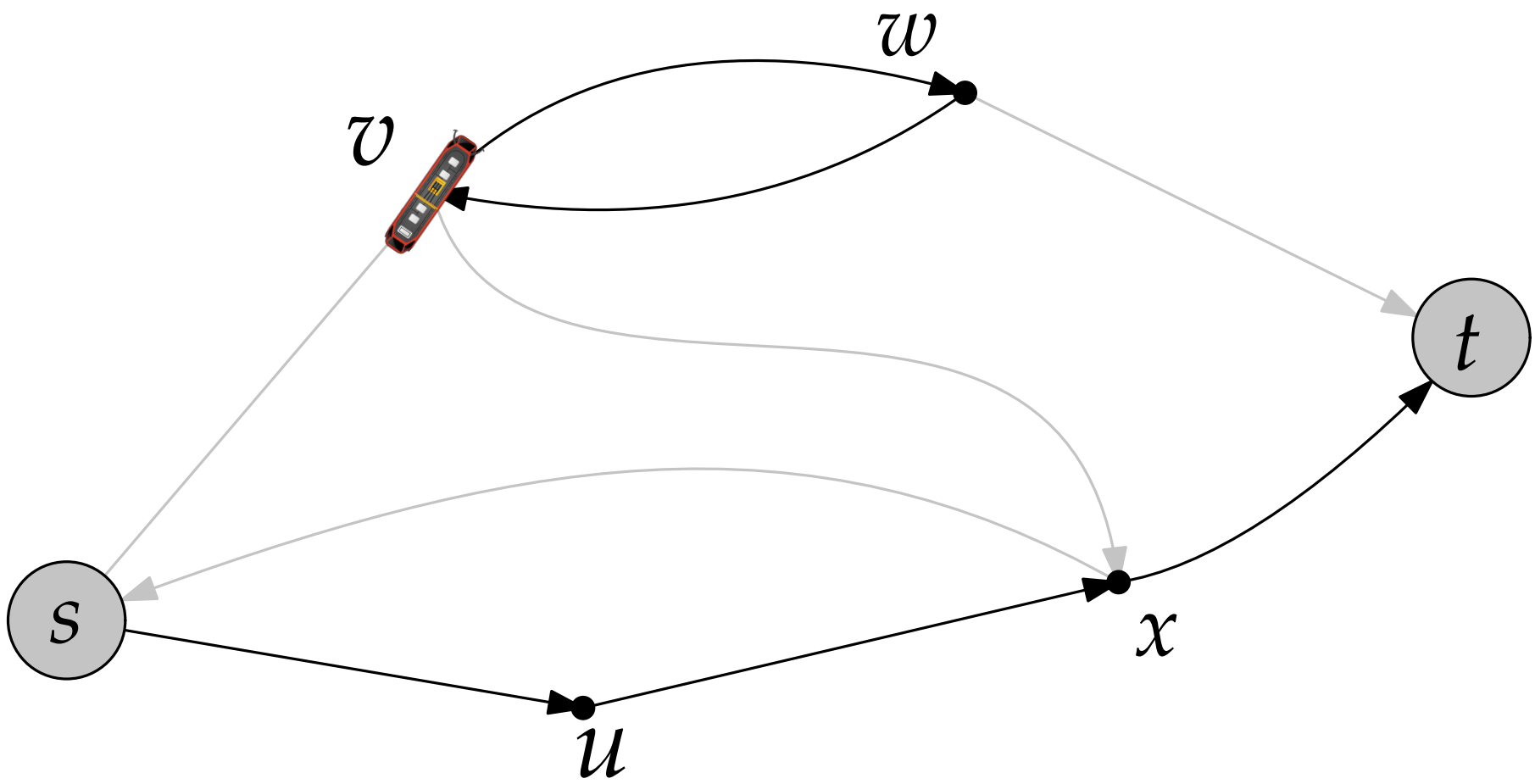
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

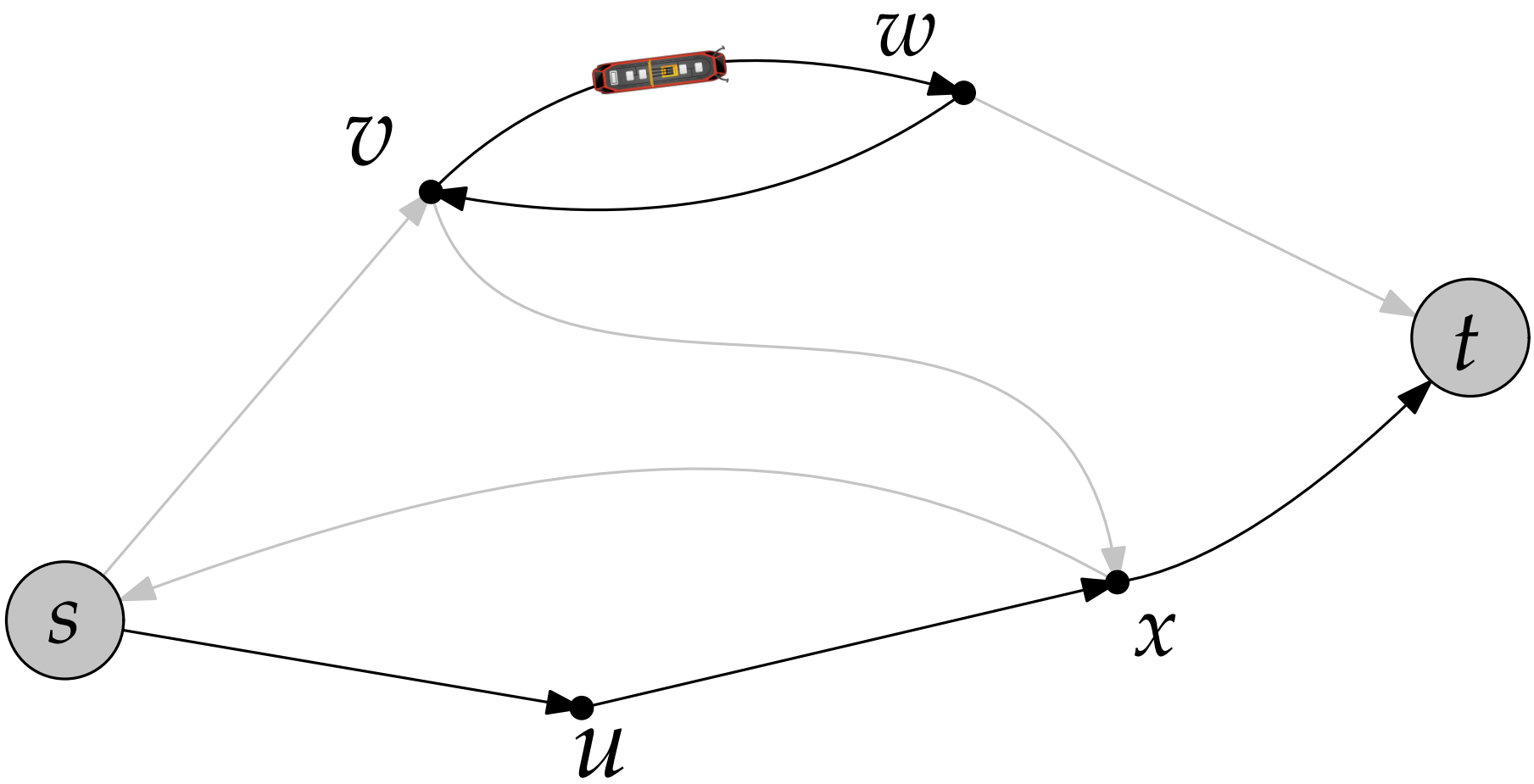
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

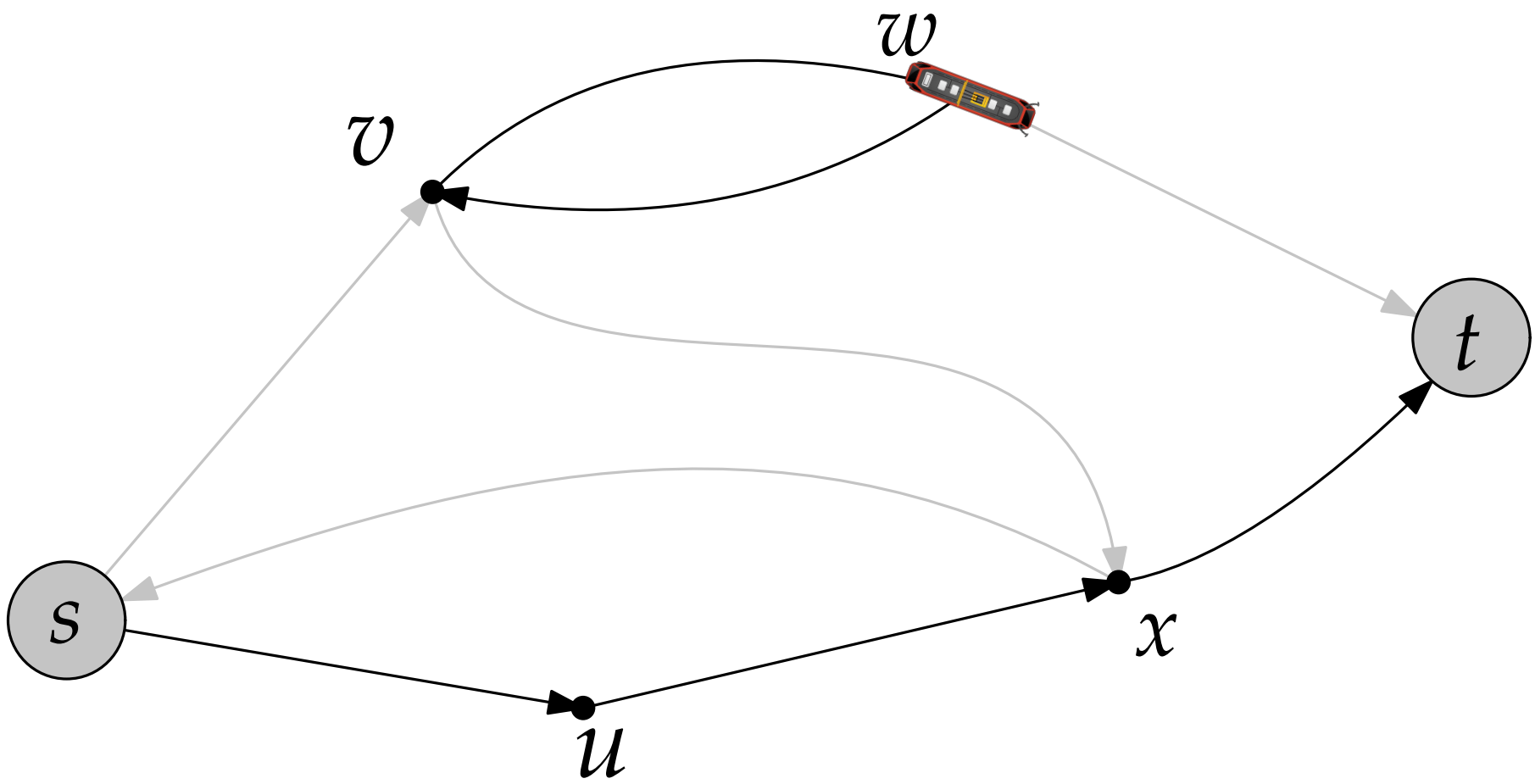
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

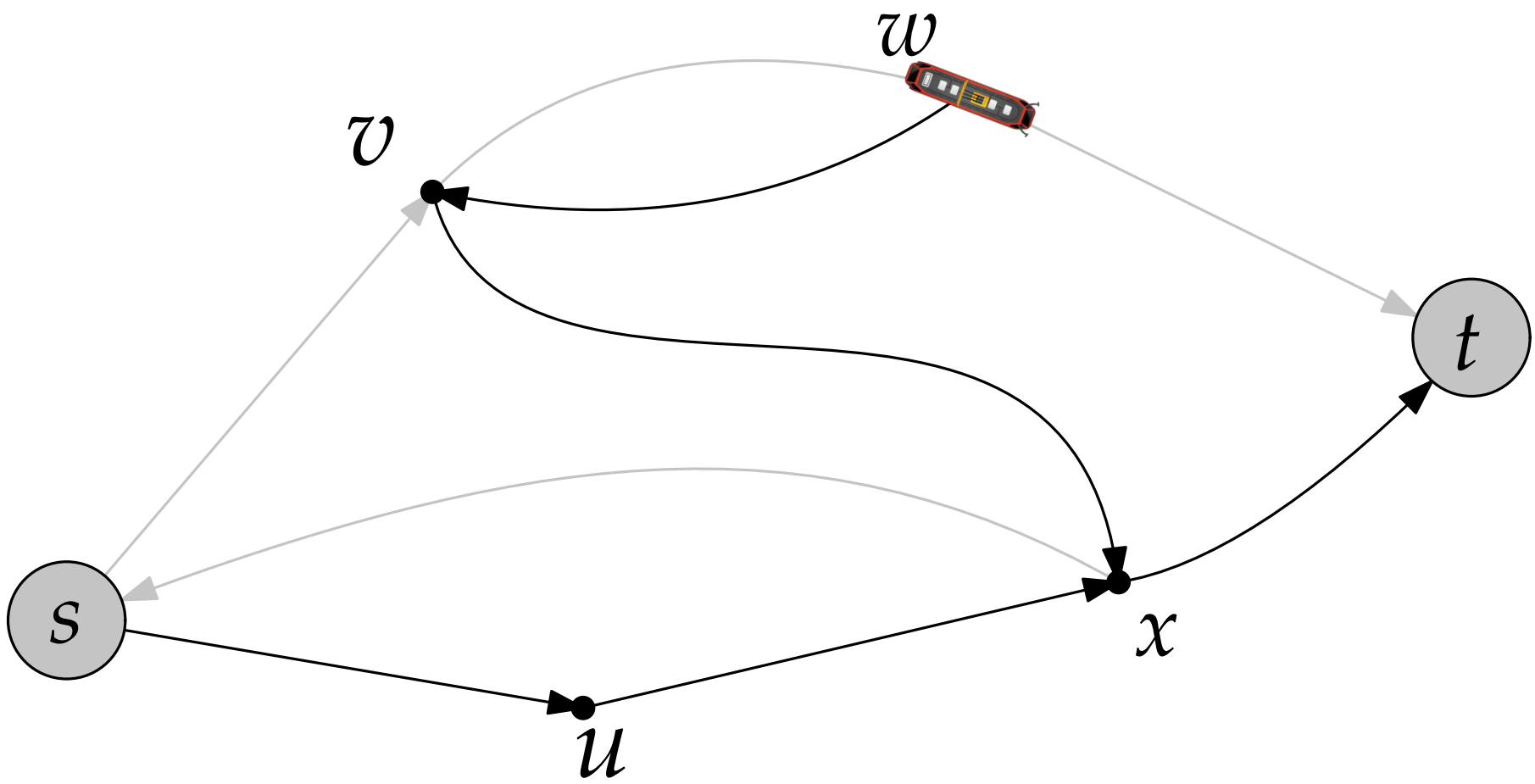
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

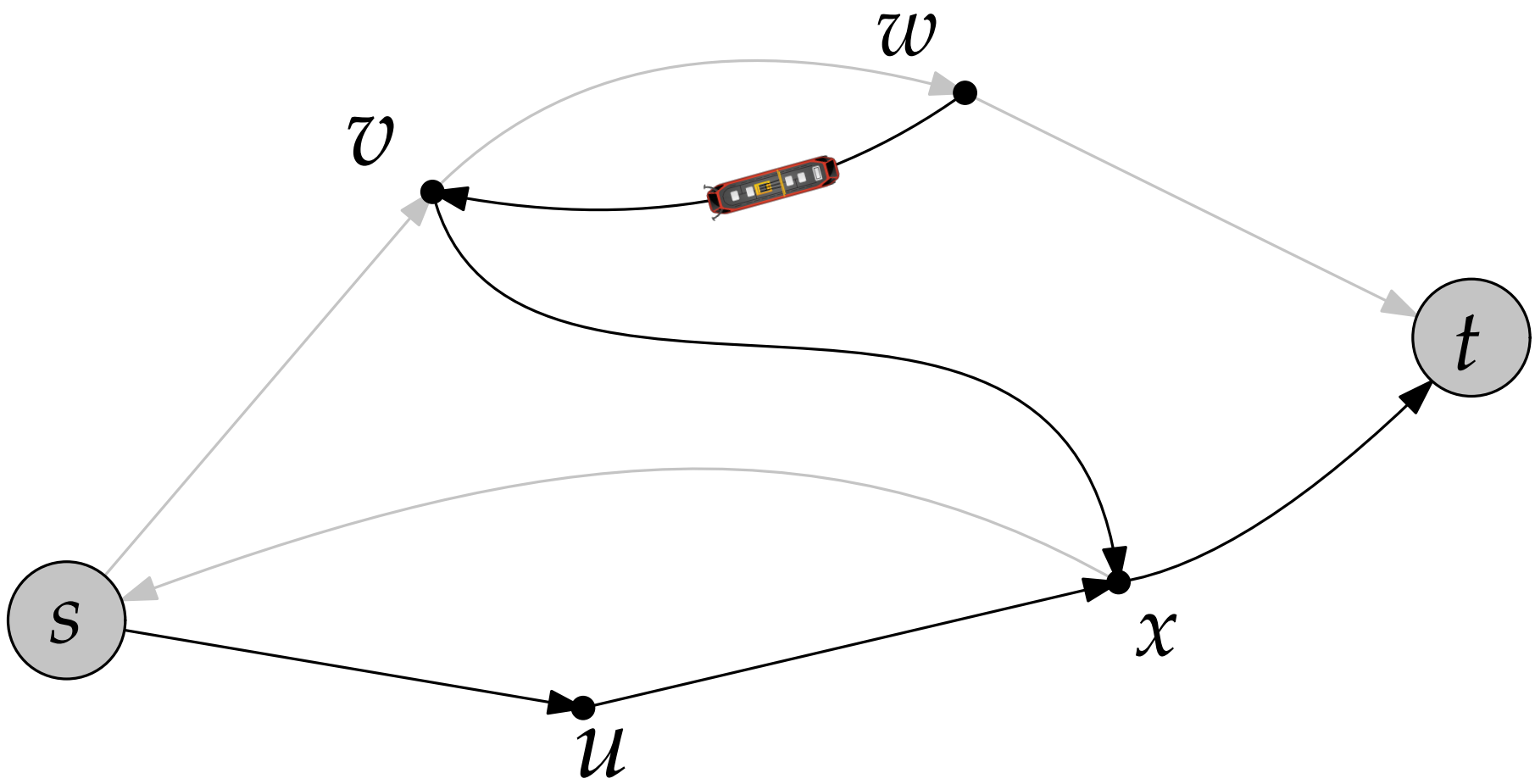
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

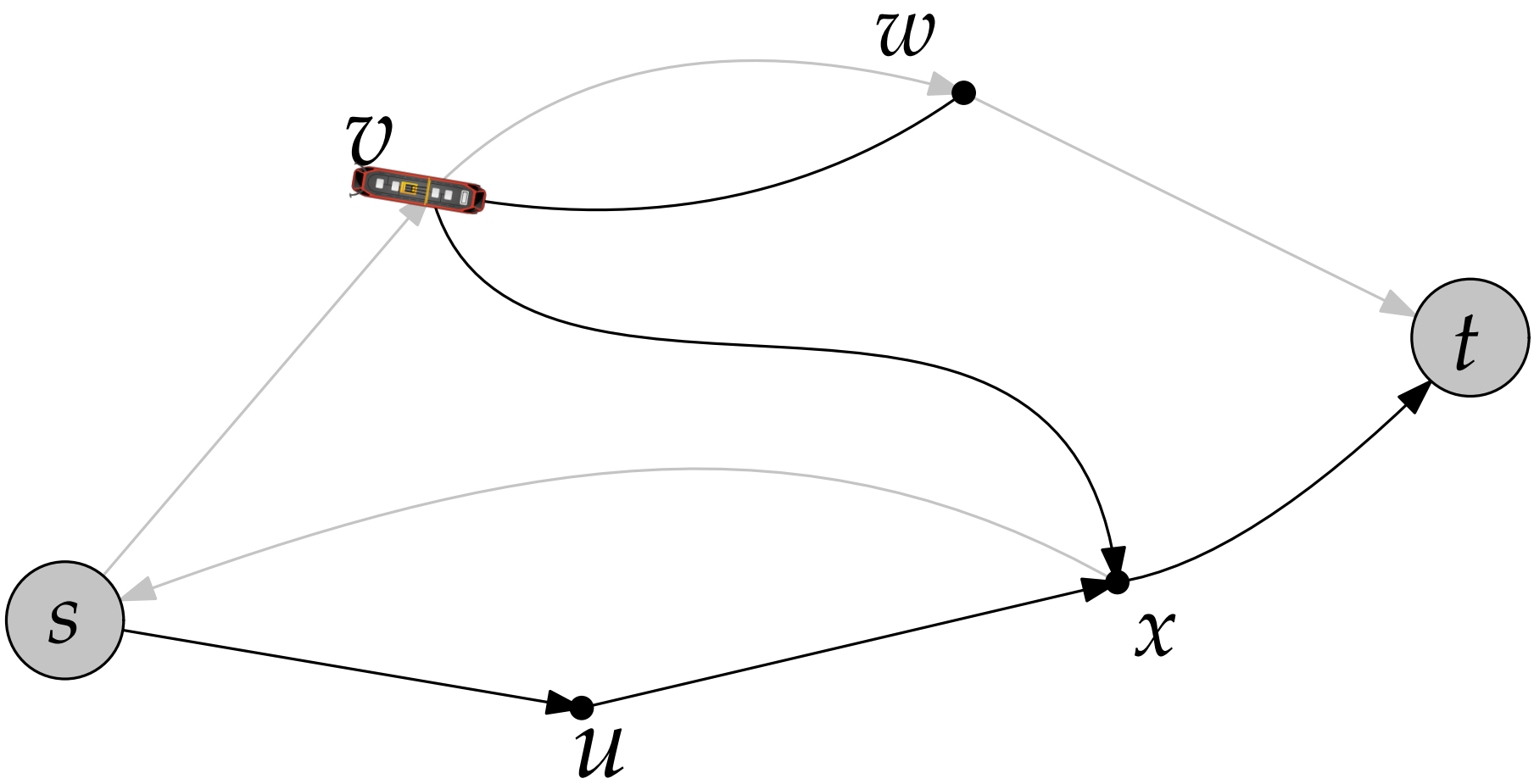
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

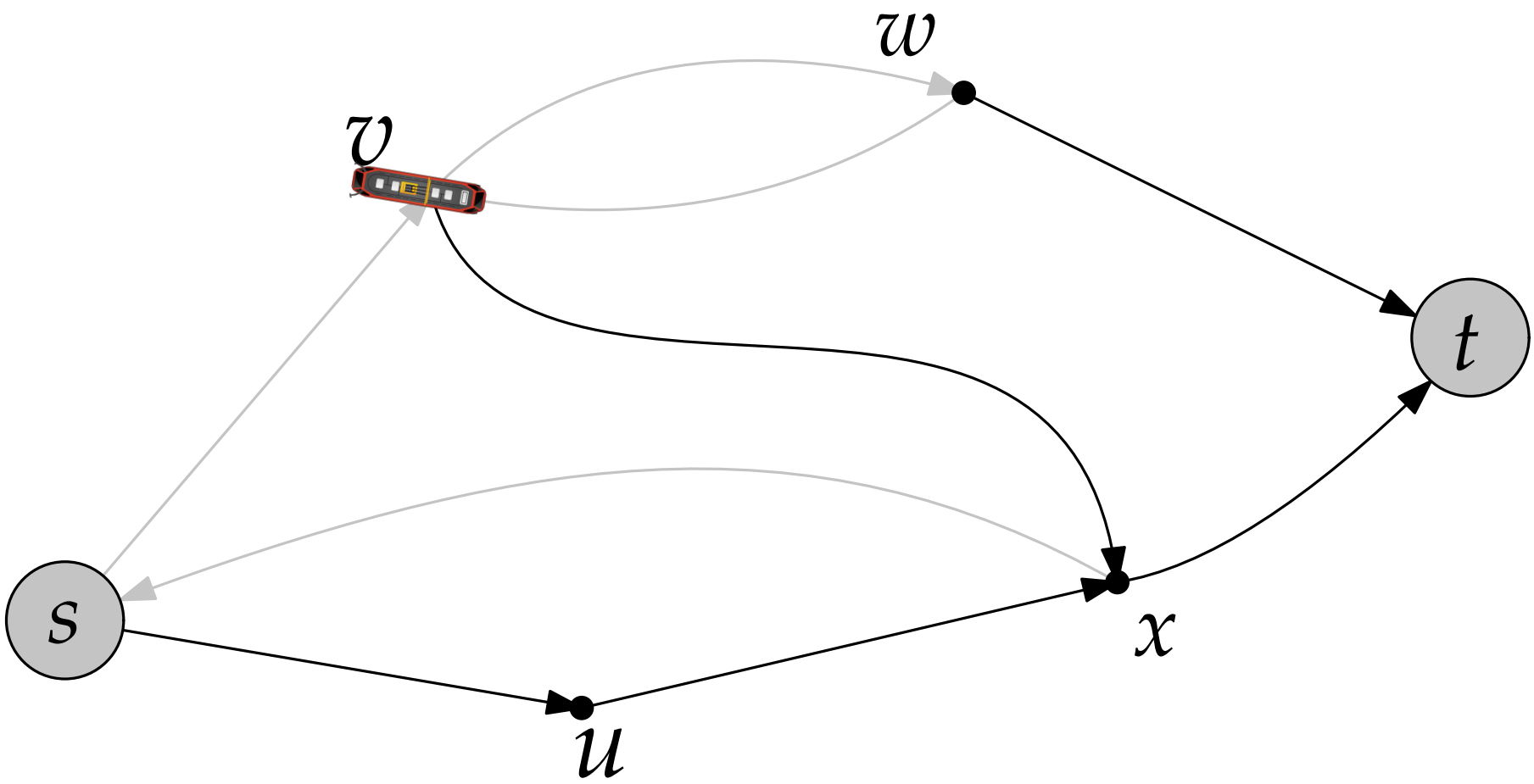
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

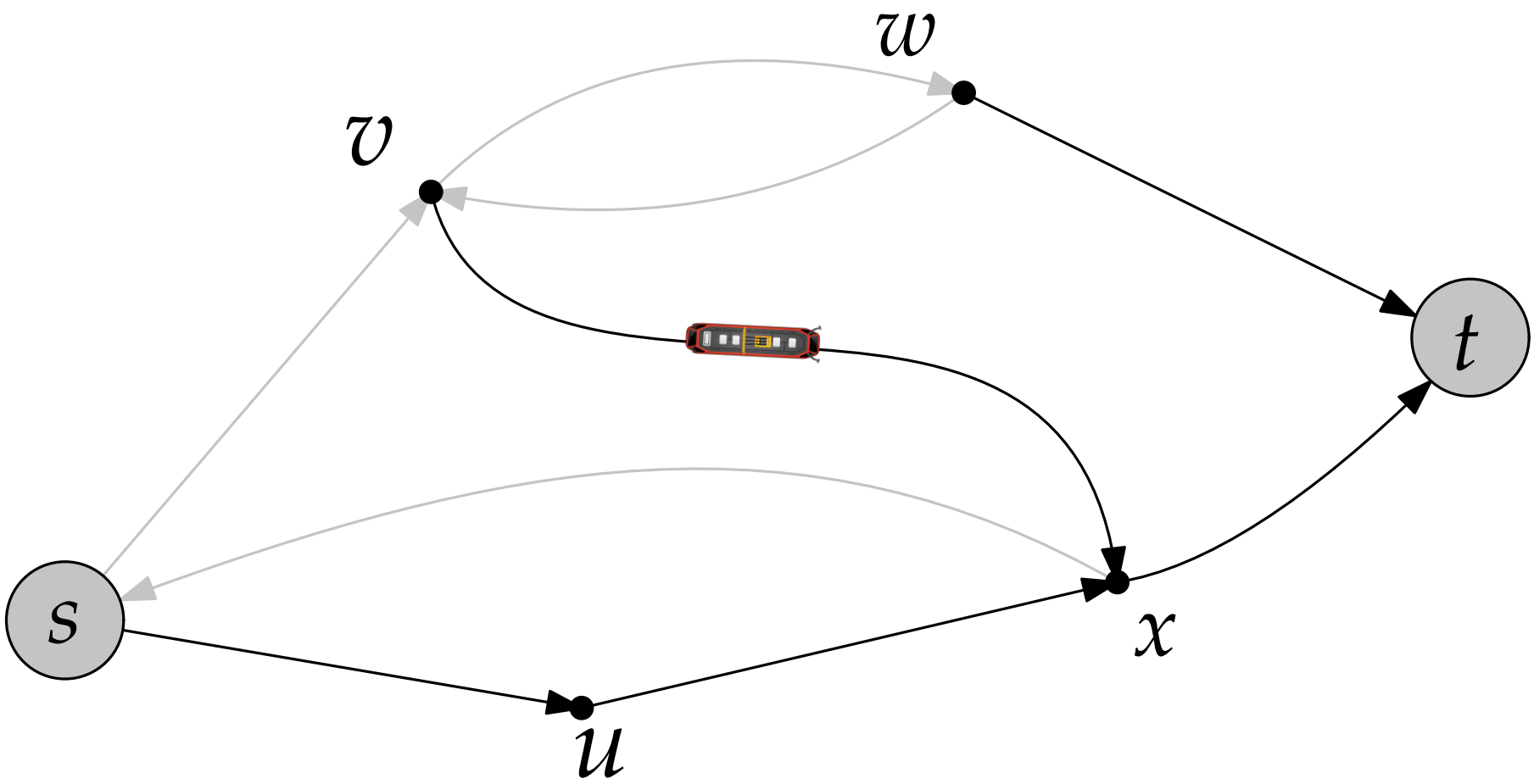
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

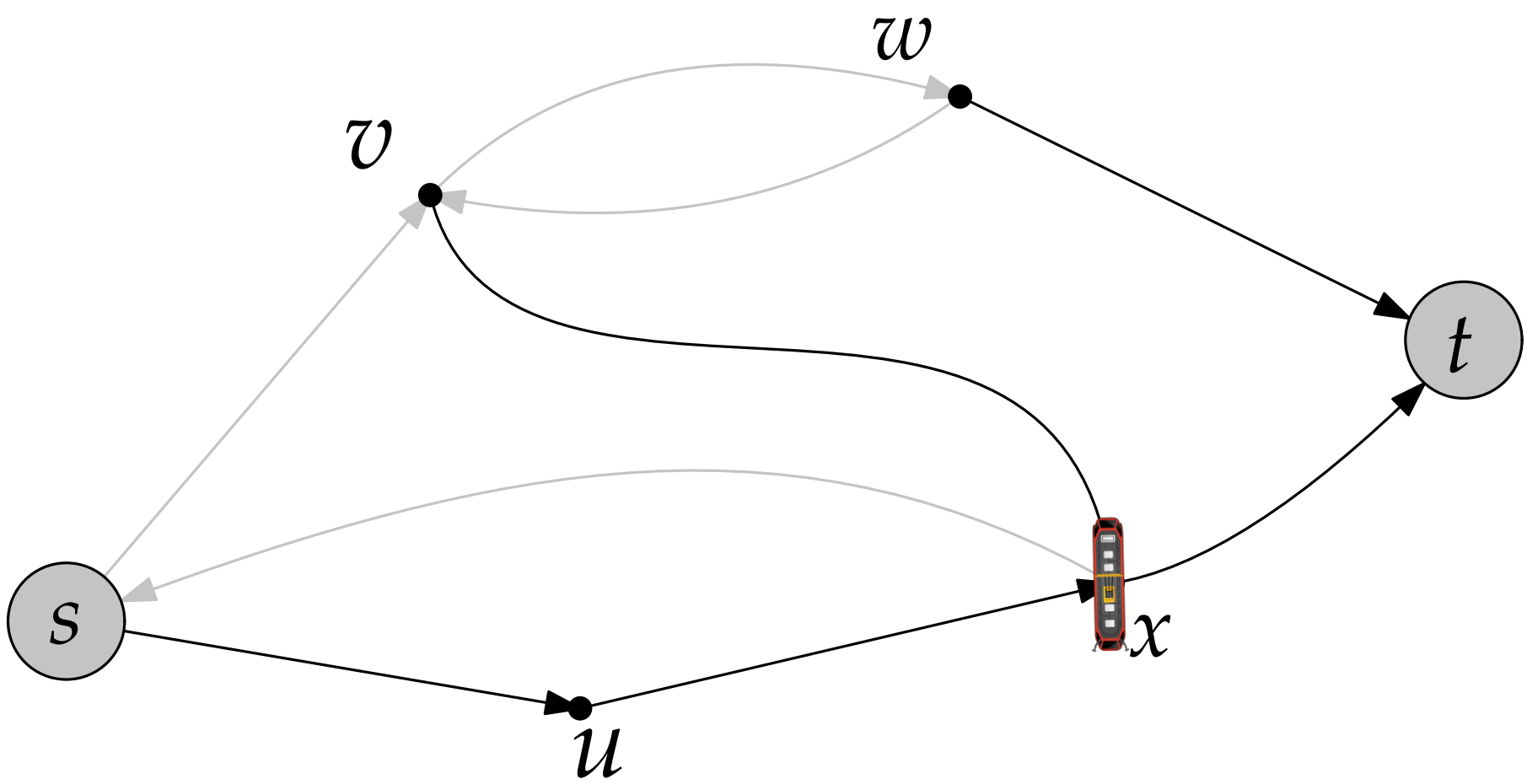
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

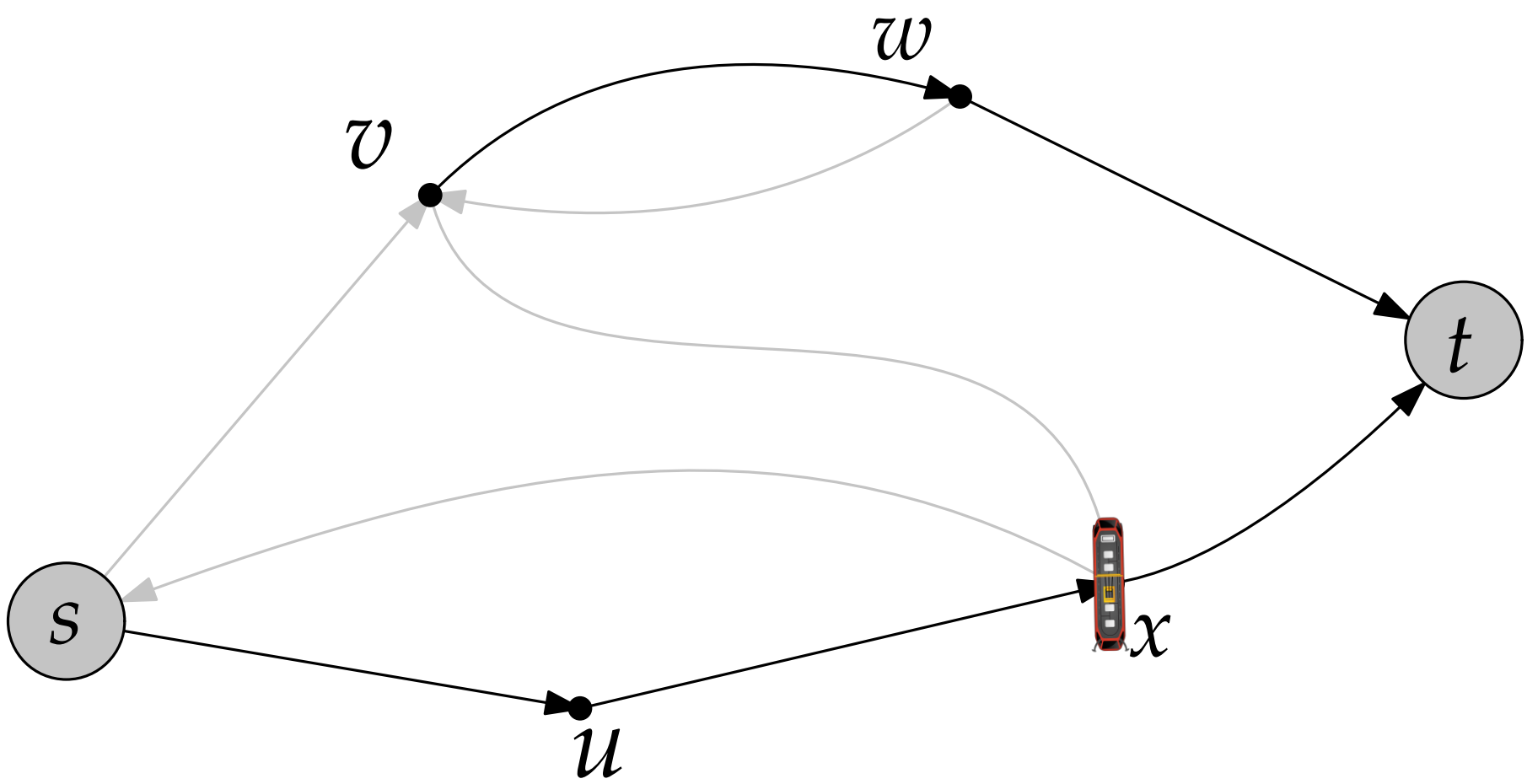
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

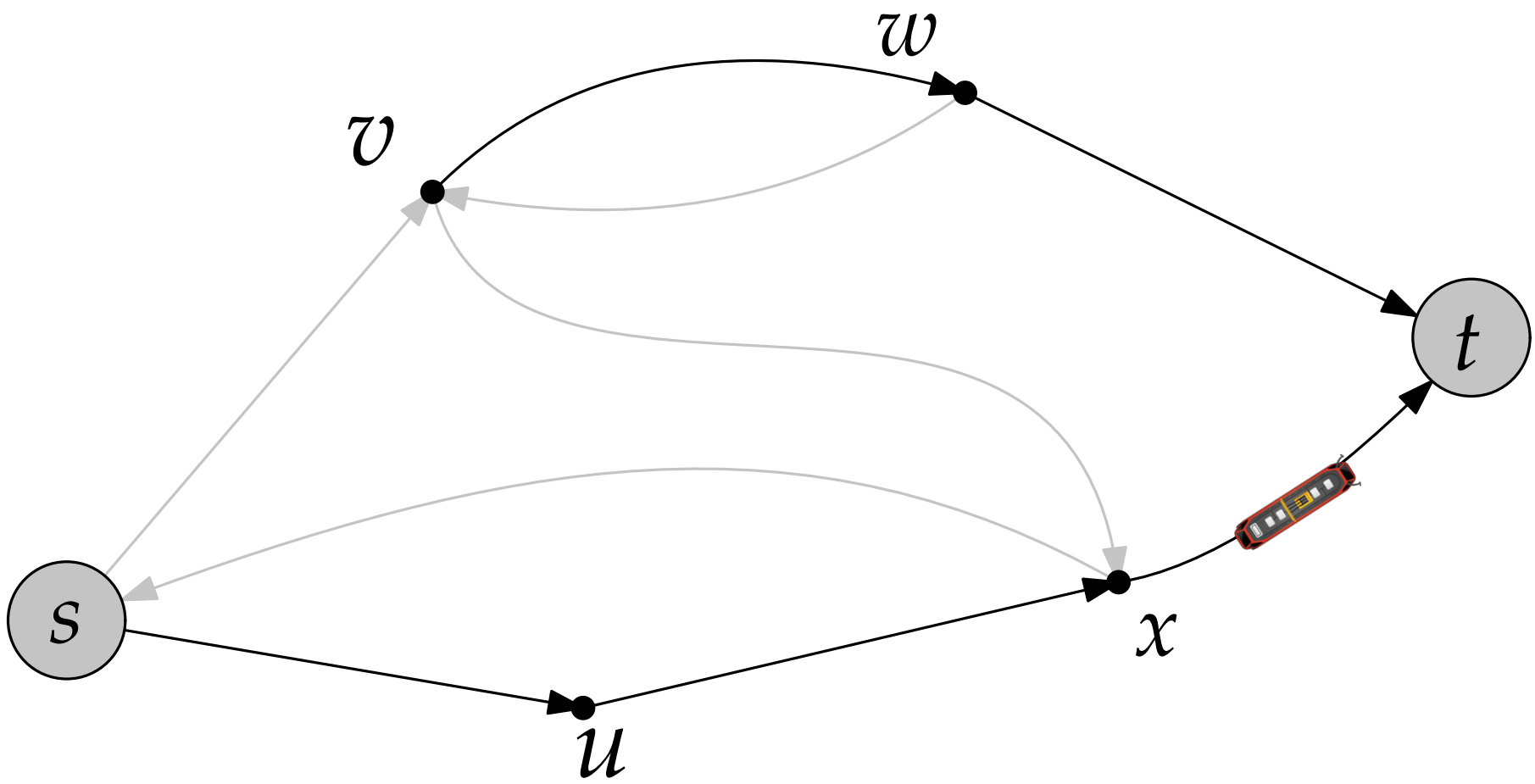
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

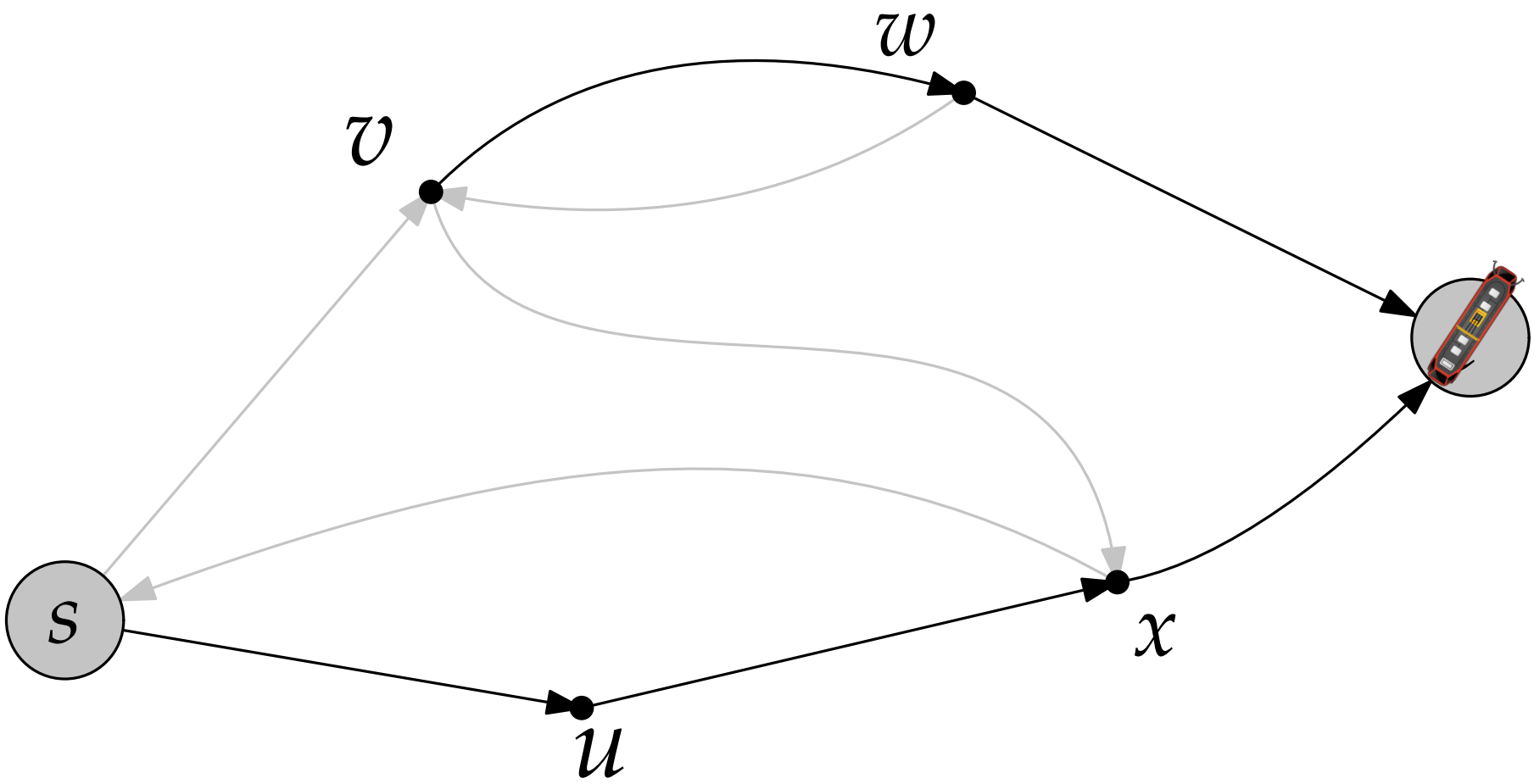
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

Arrival

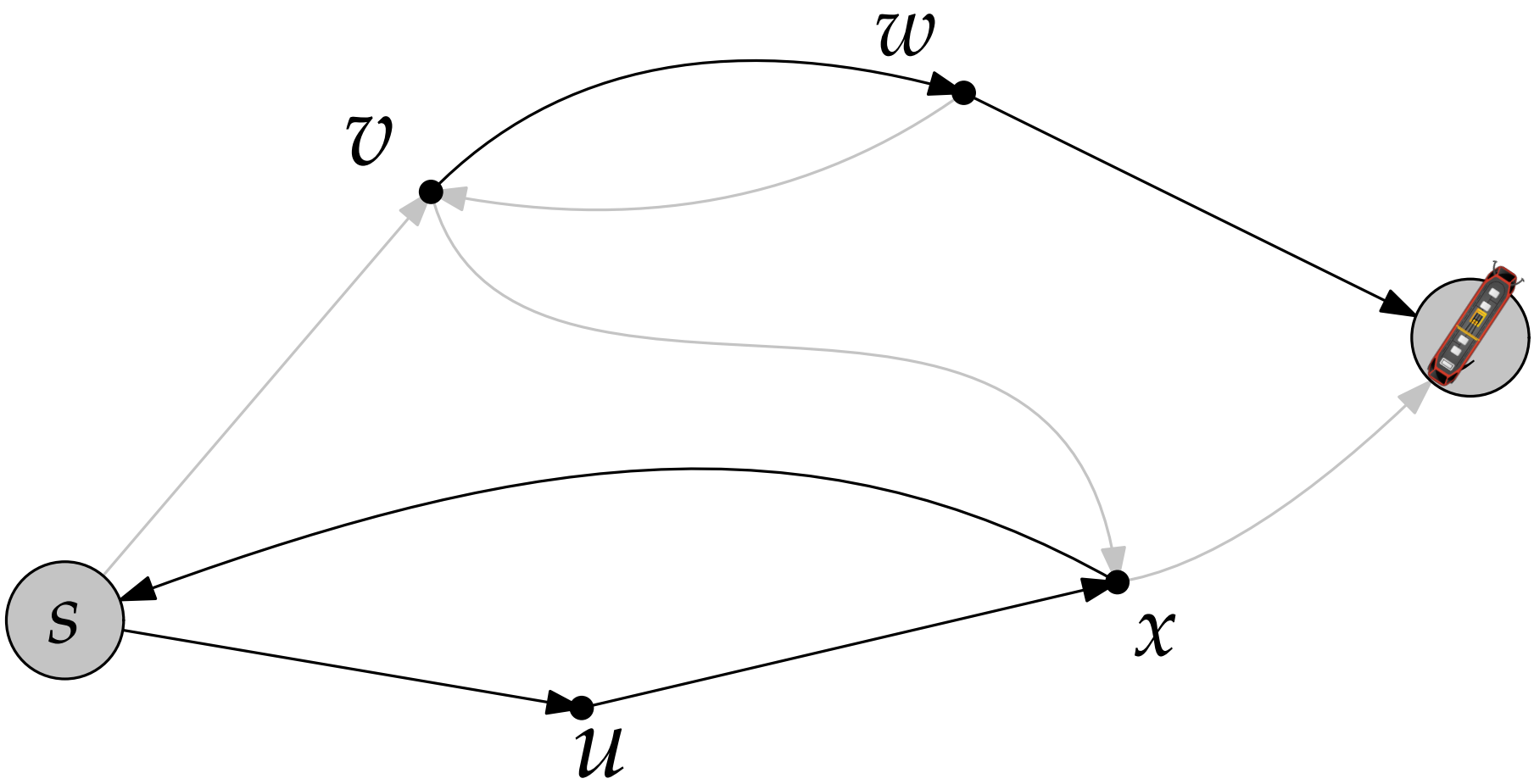
Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



Will the train ever reach the station?

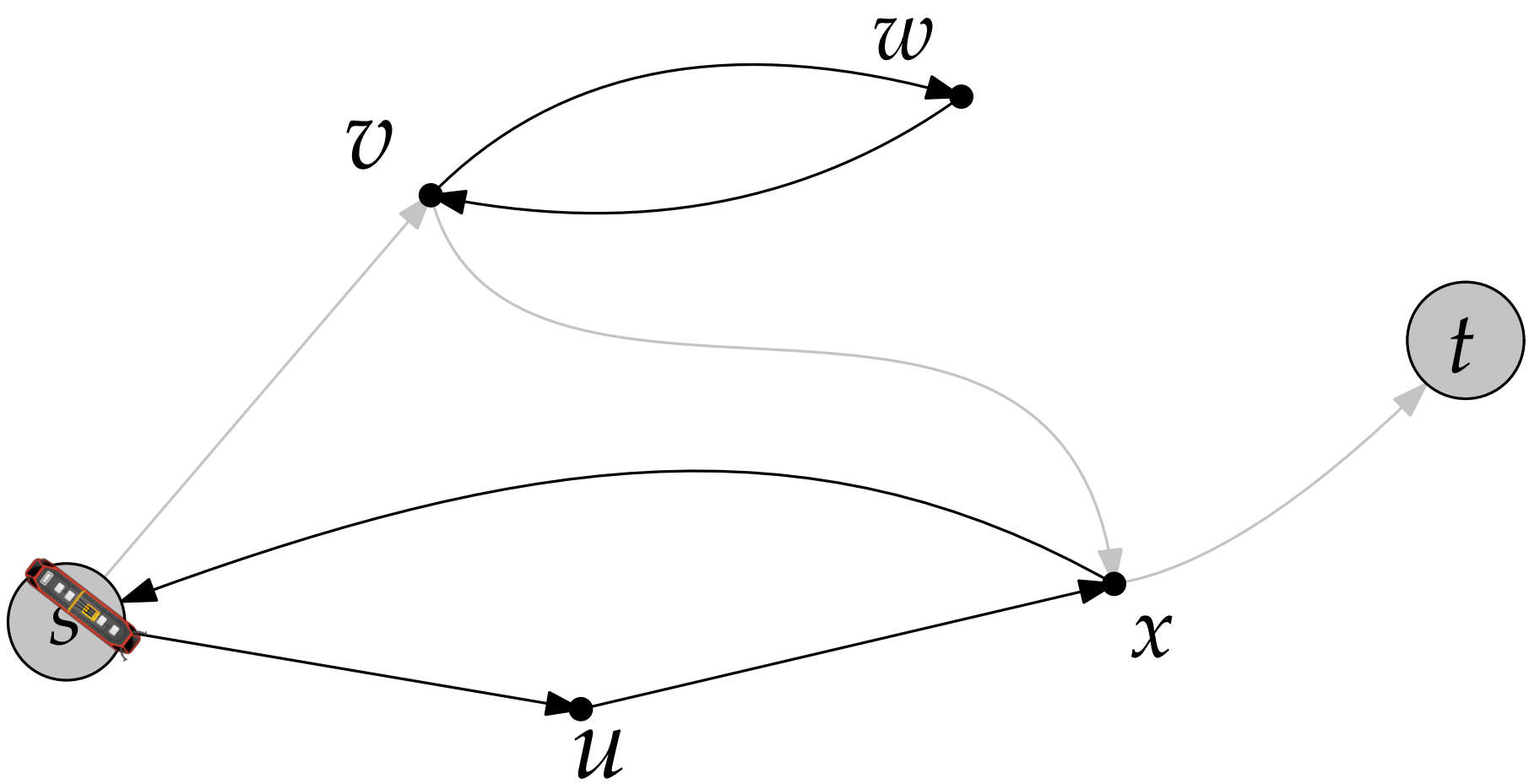
Arrival

Dohrau, Gärtner, Kohler, Matoušek, Welzl (2017)



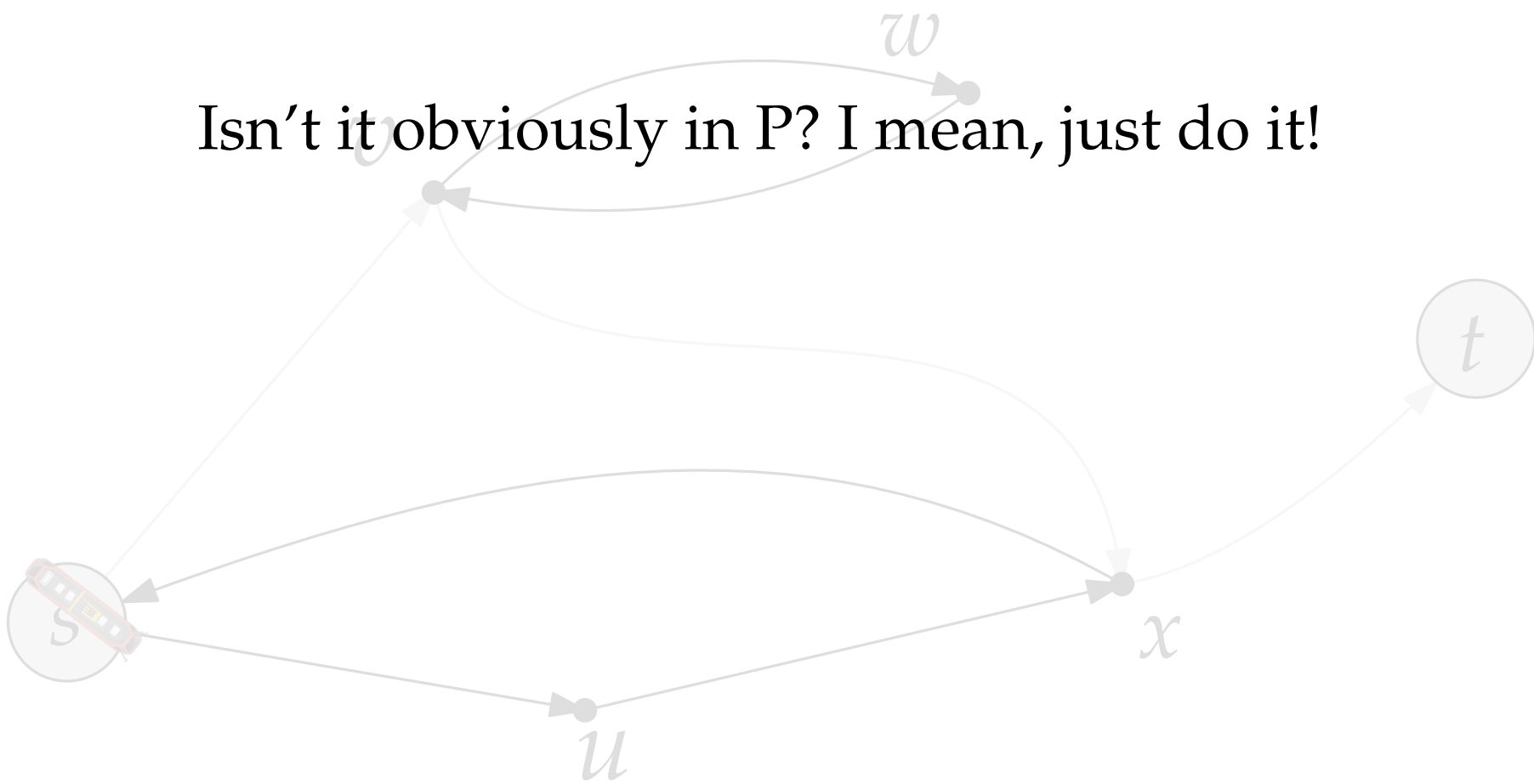
Will the train ever reach the station?

Arrival

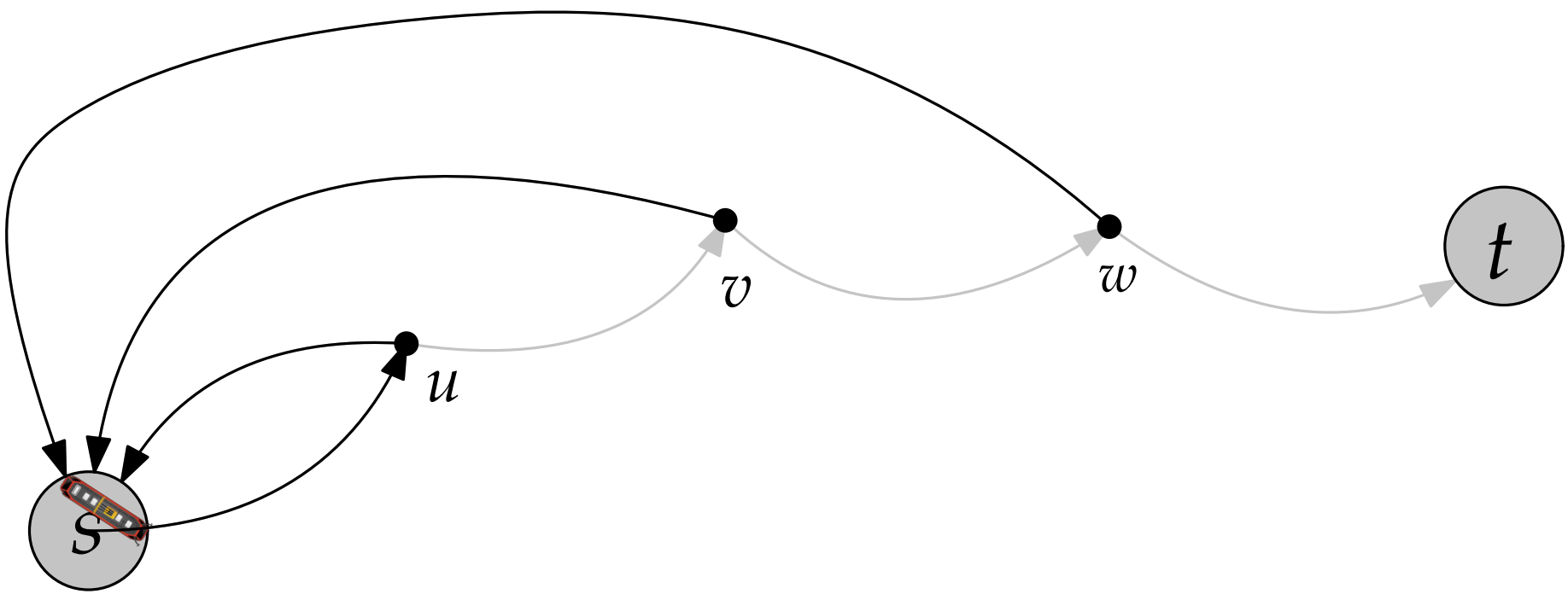


Arrival

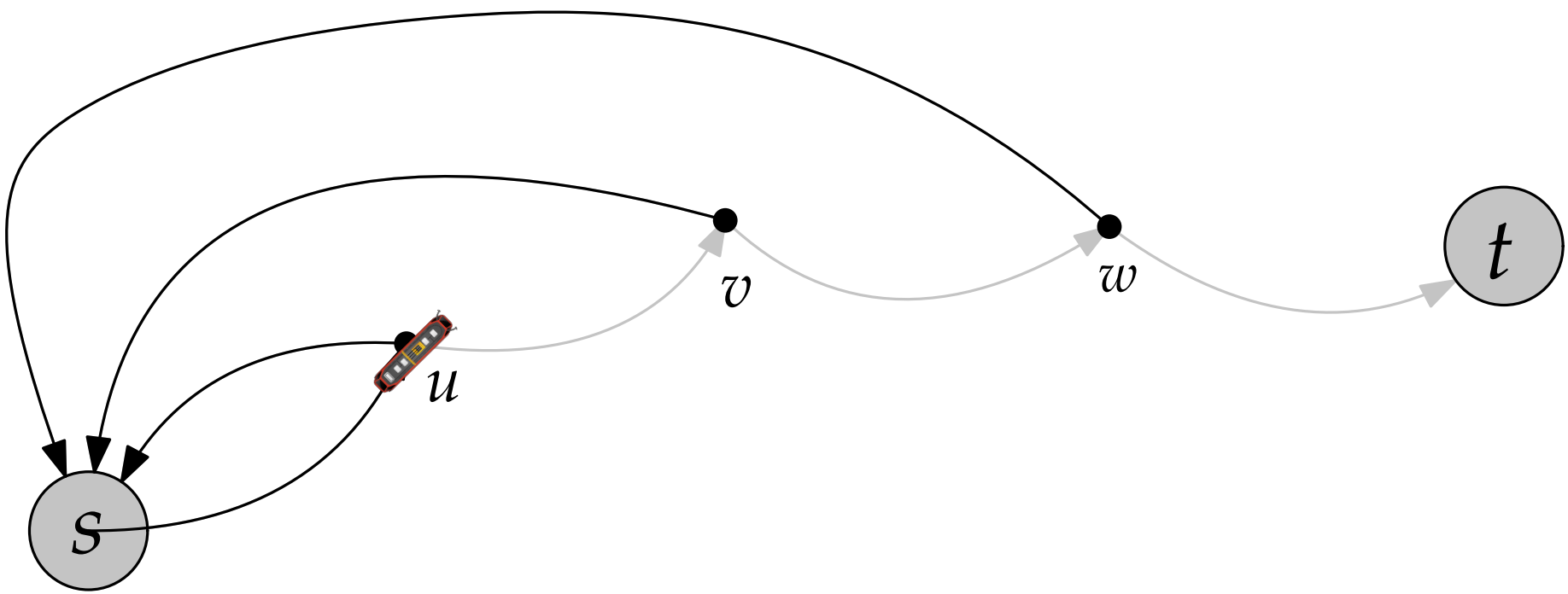
Isn't it obviously in P? I mean, just do it!



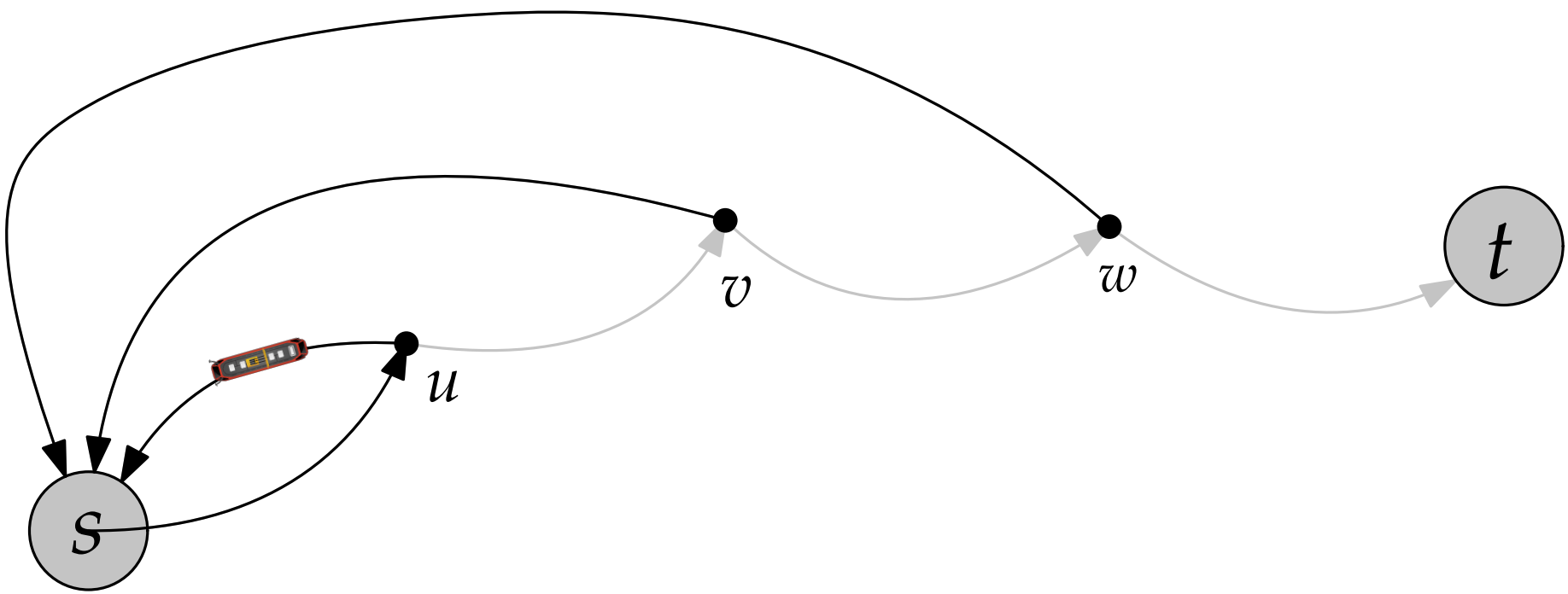
Arrival



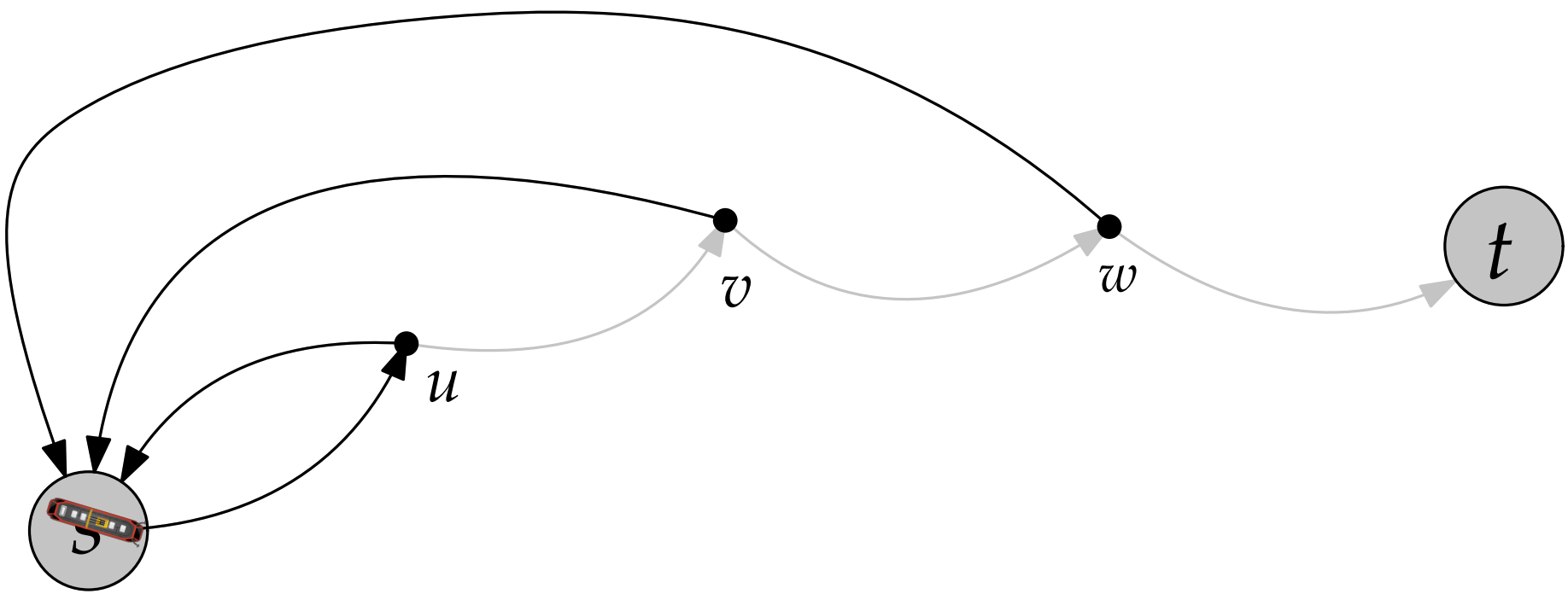
Arrival



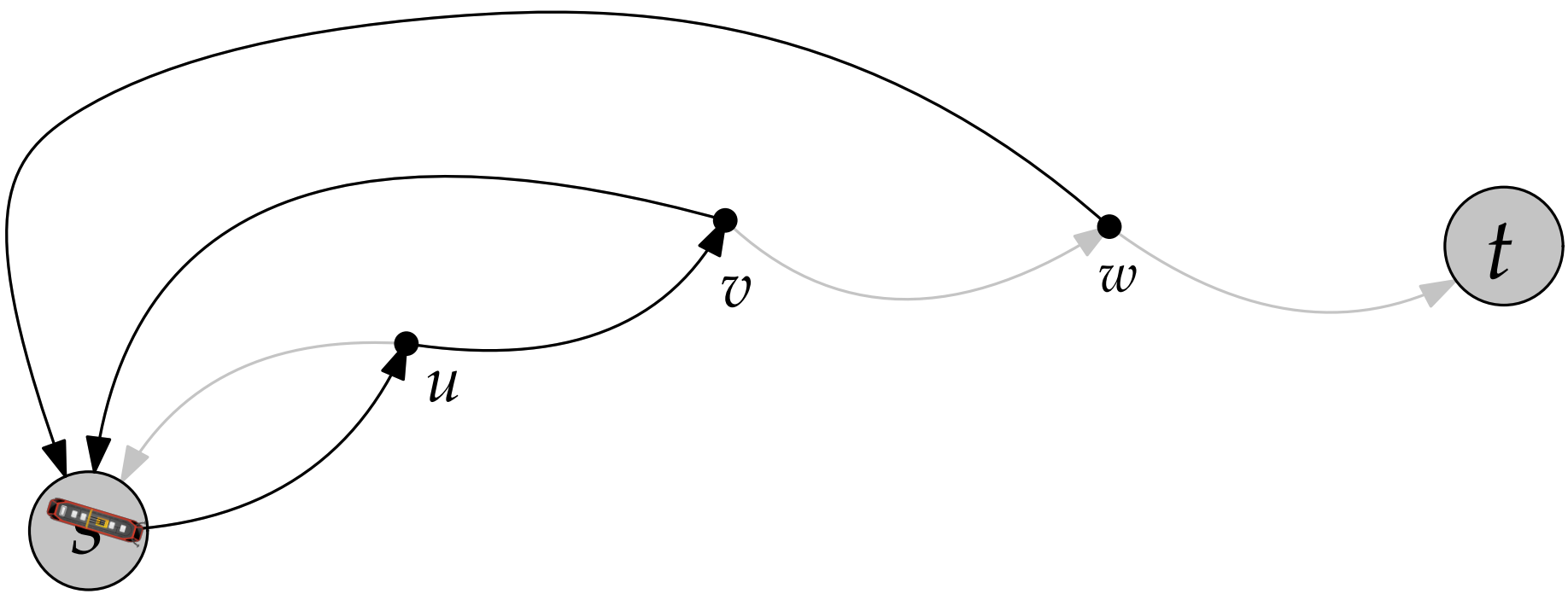
Arrival



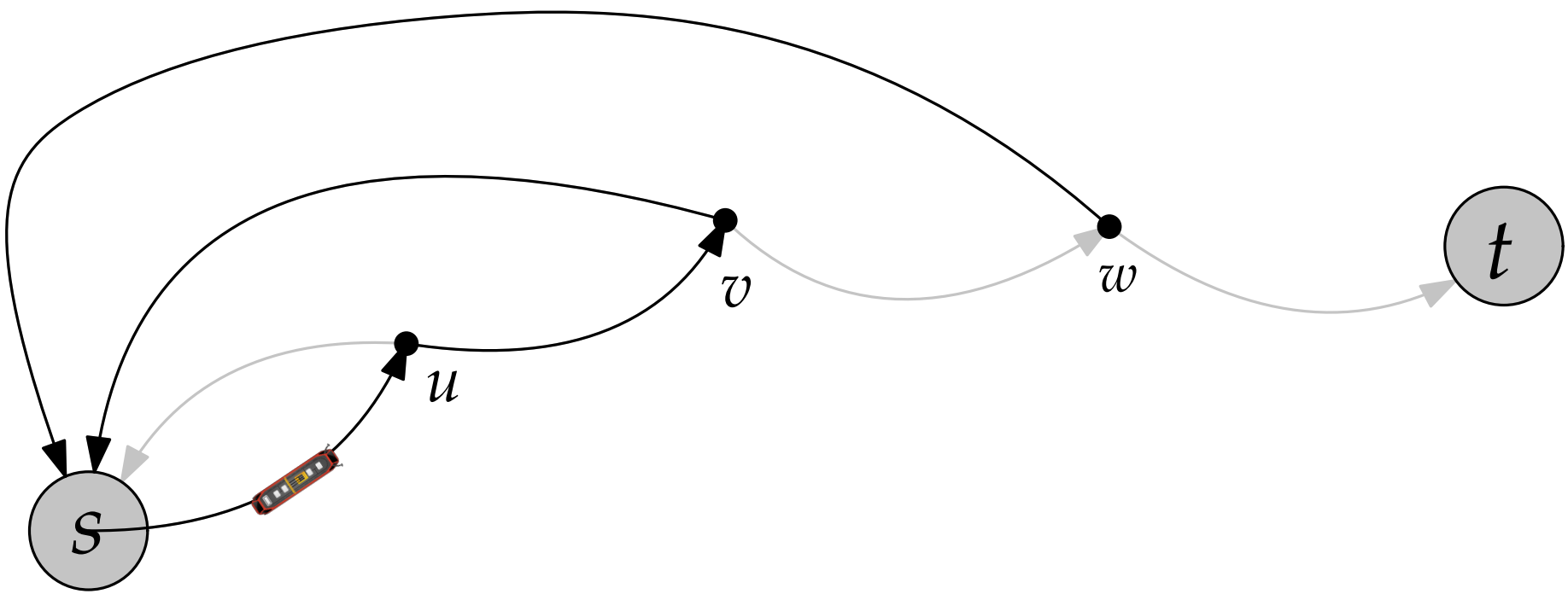
Arrival



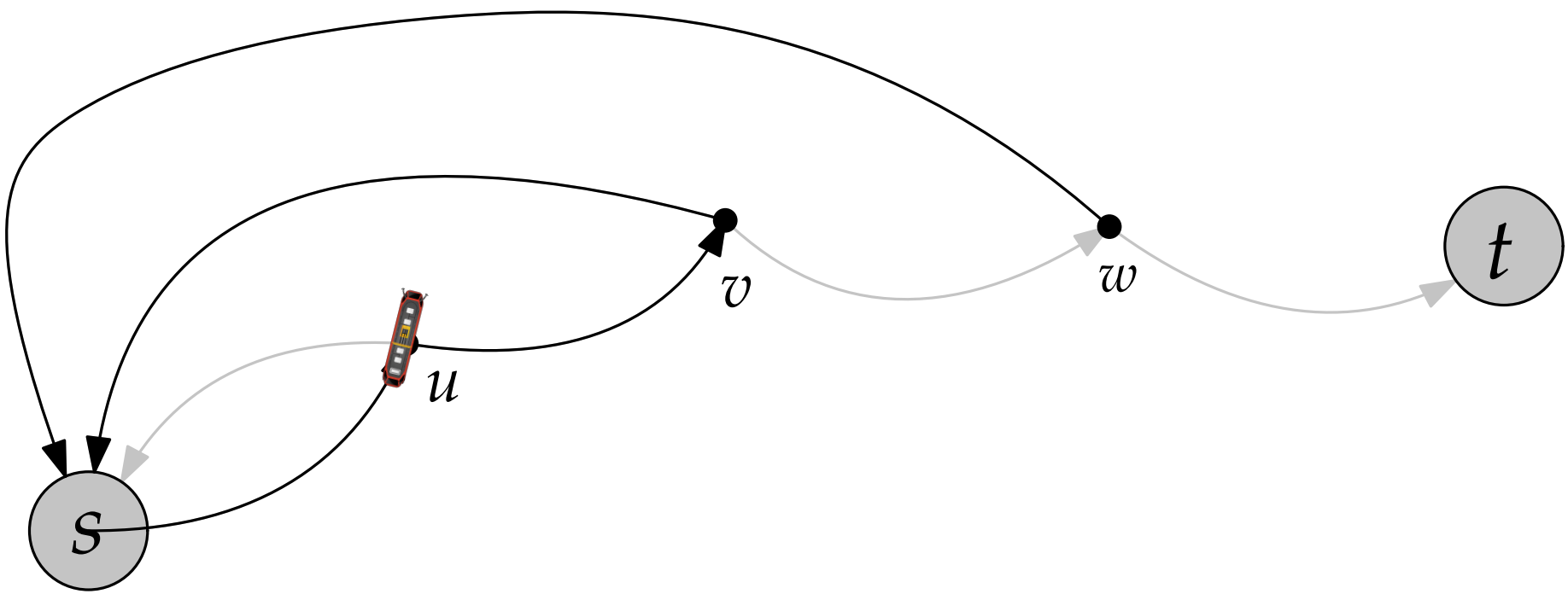
Arrival



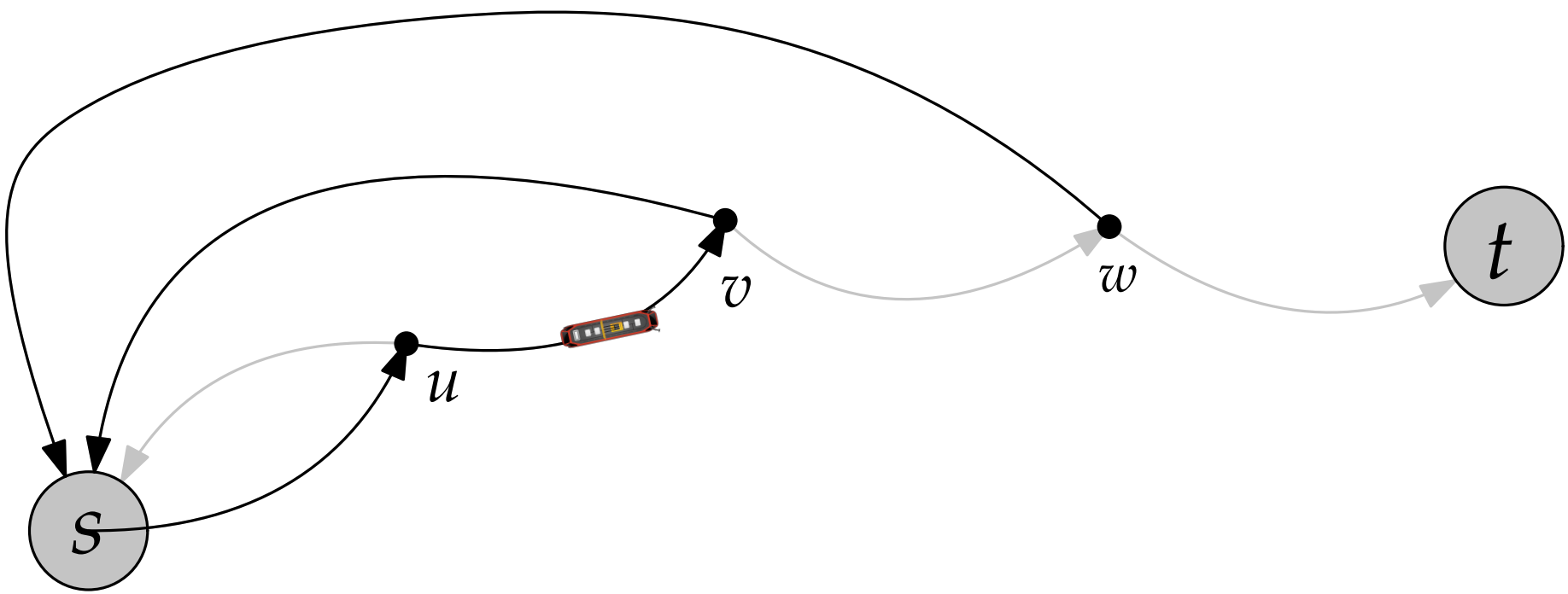
Arrival



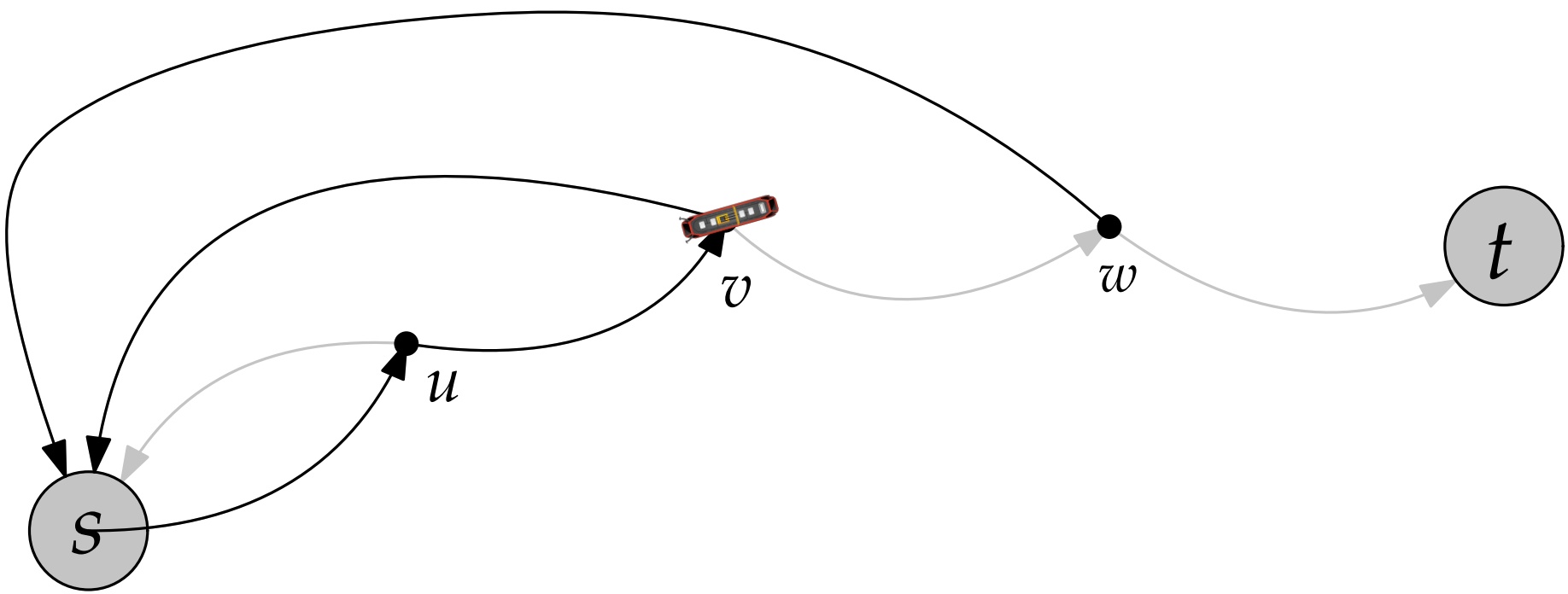
Arrival



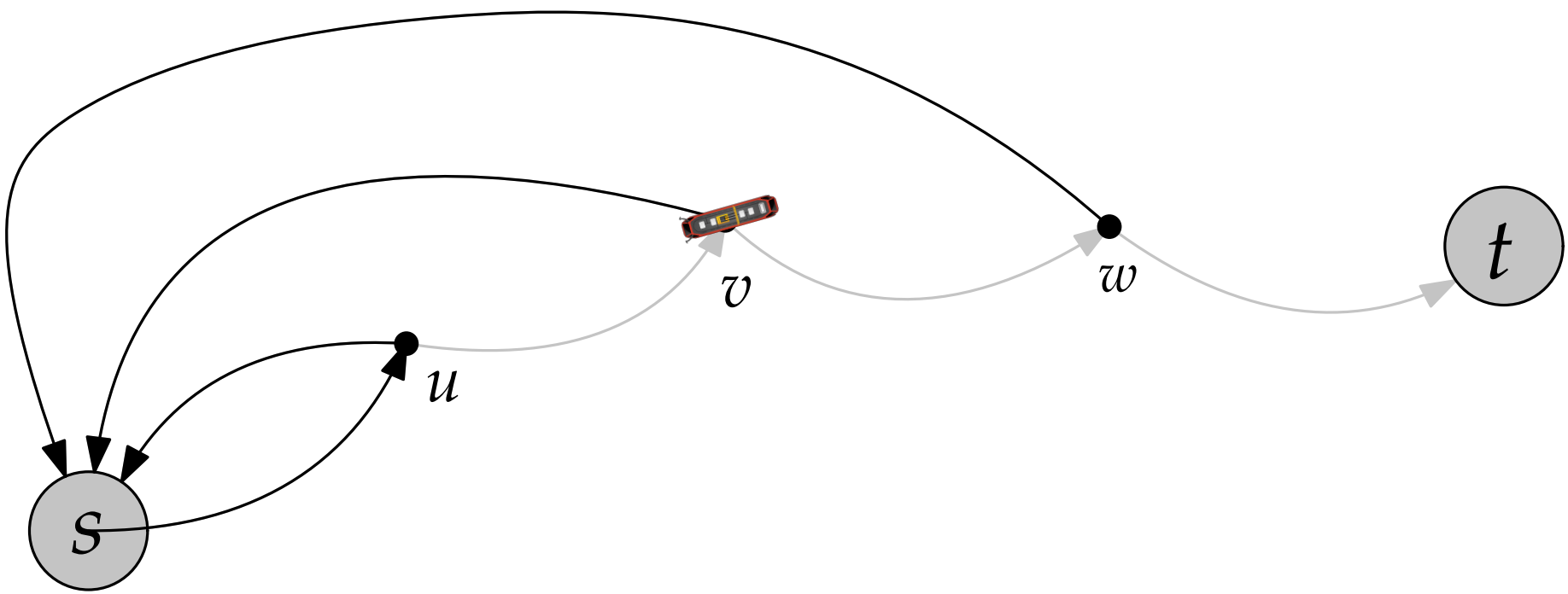
Arrival



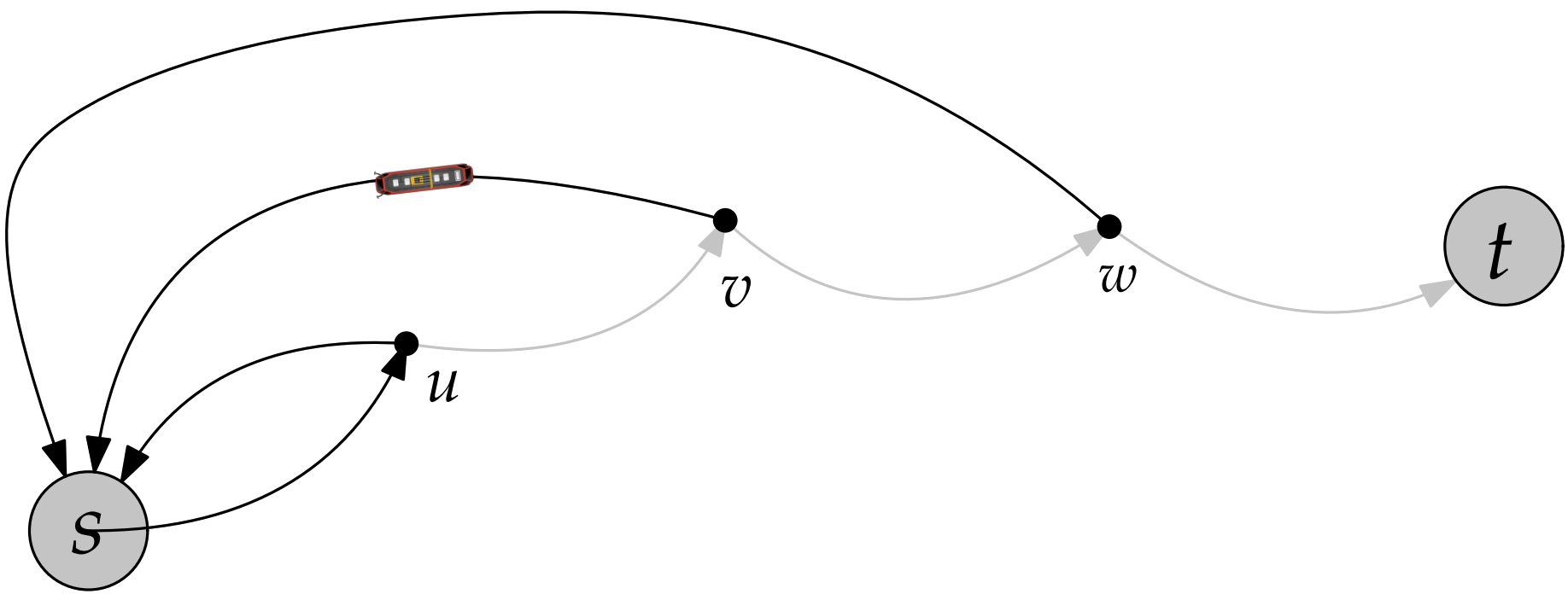
Arrival



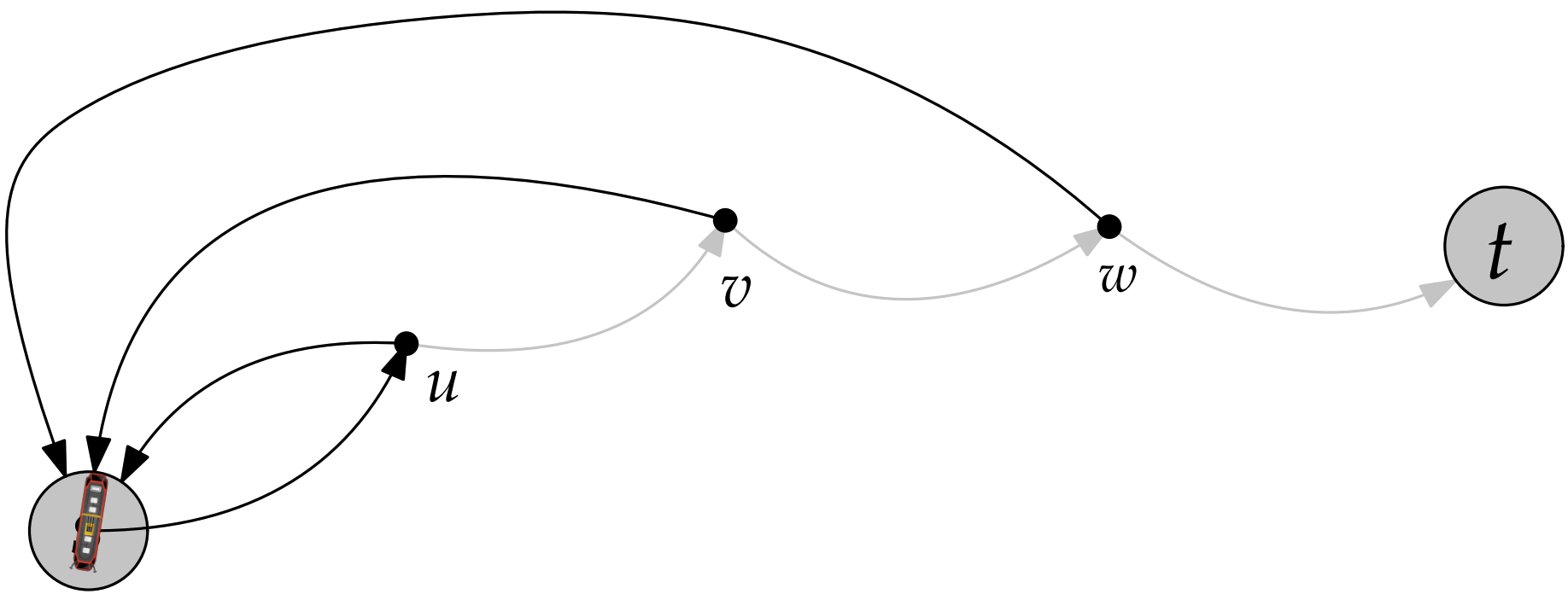
Arrival



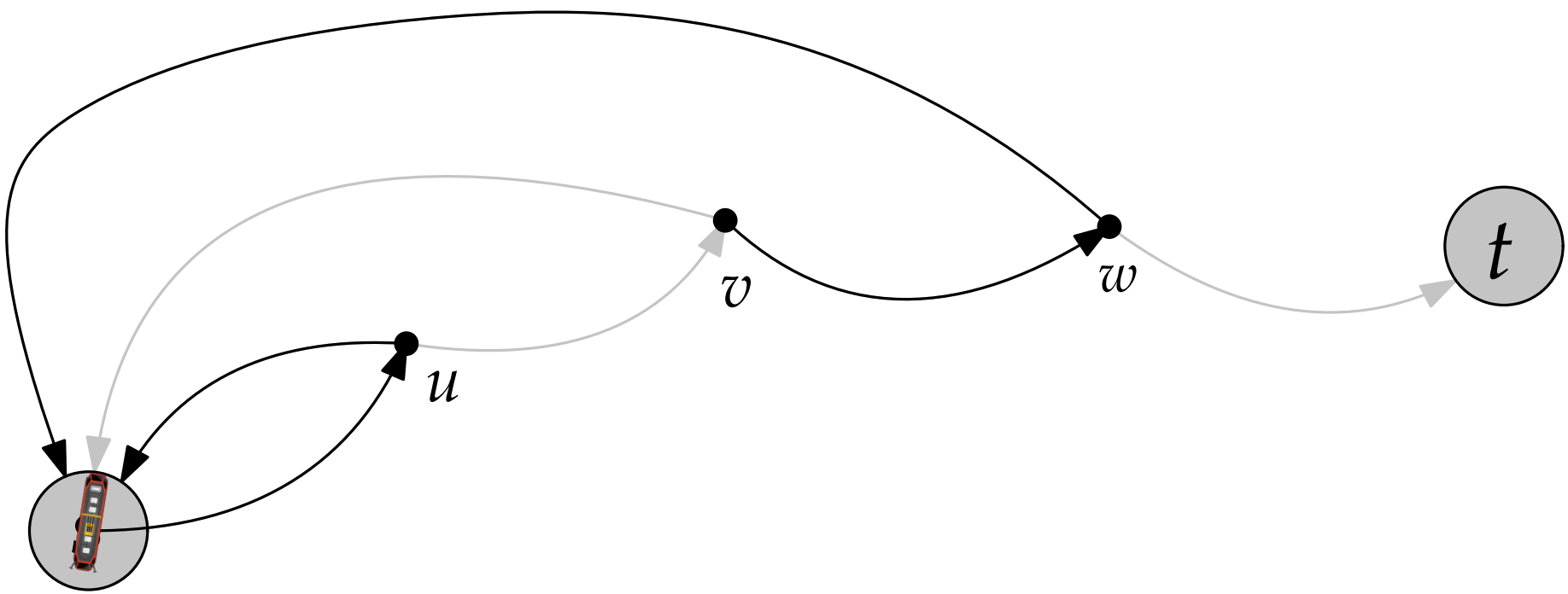
Arrival



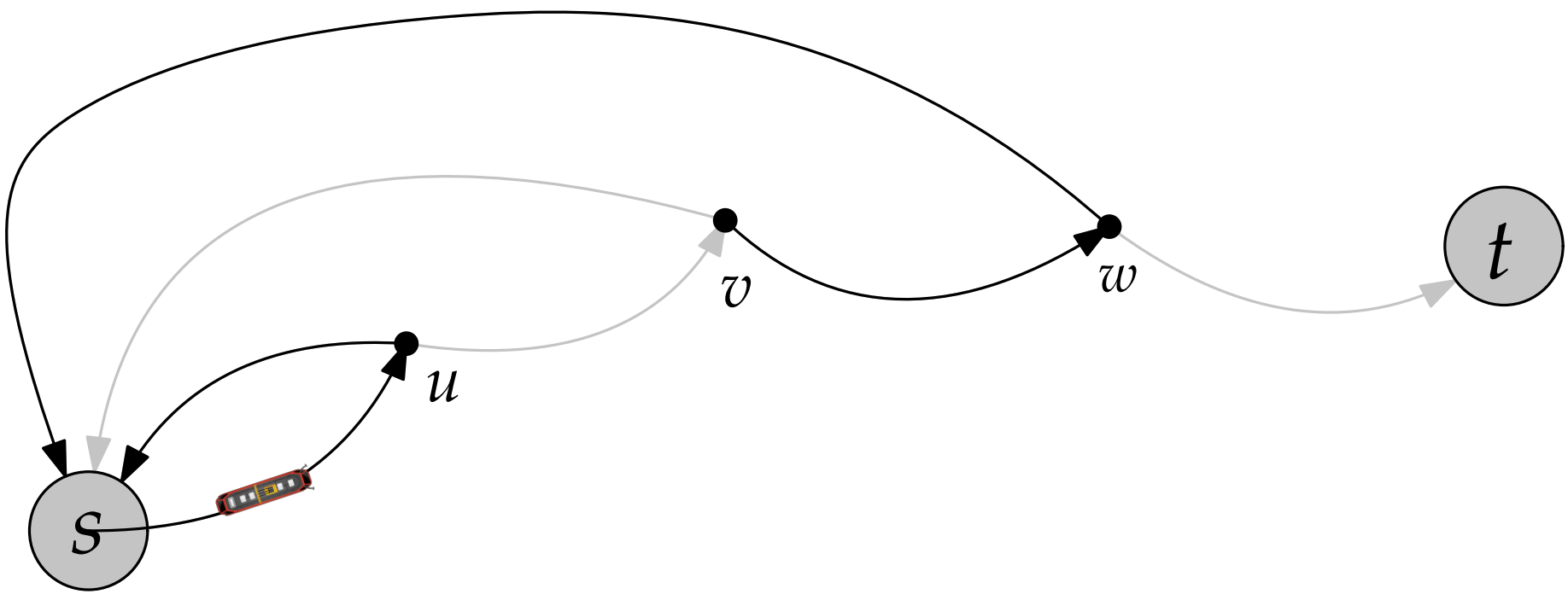
Arrival



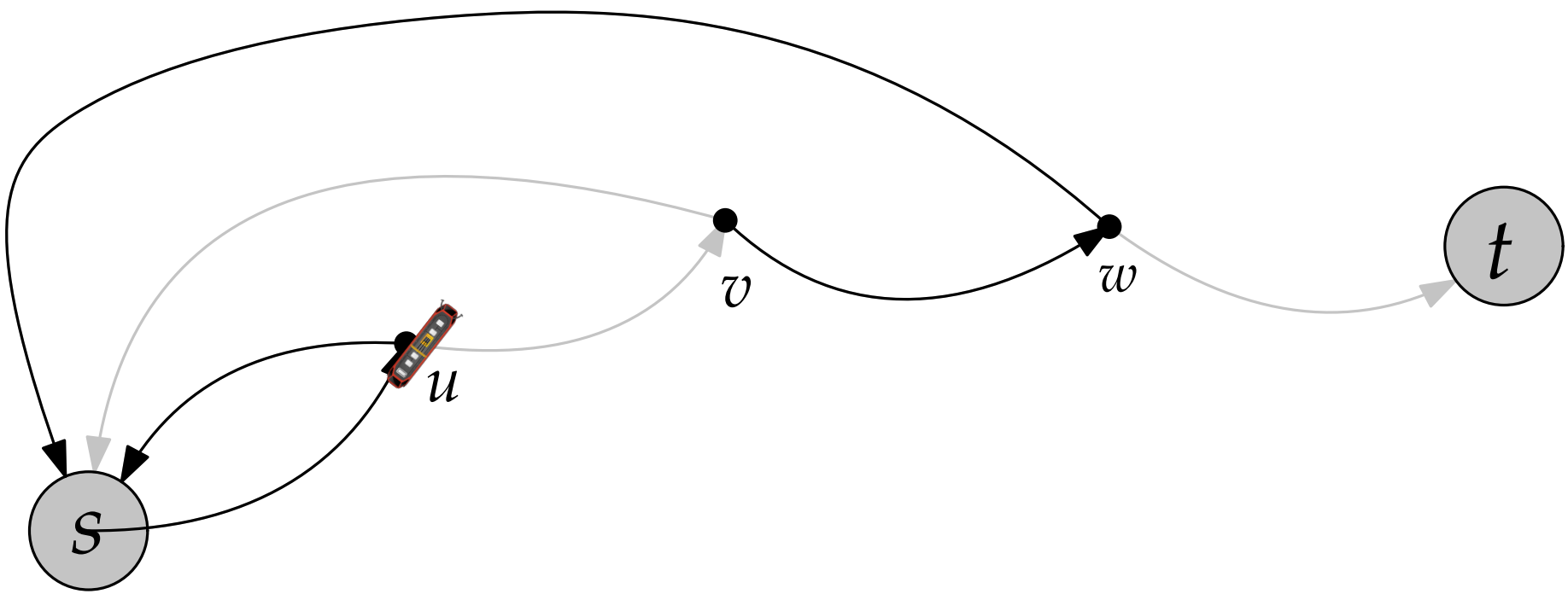
Arrival



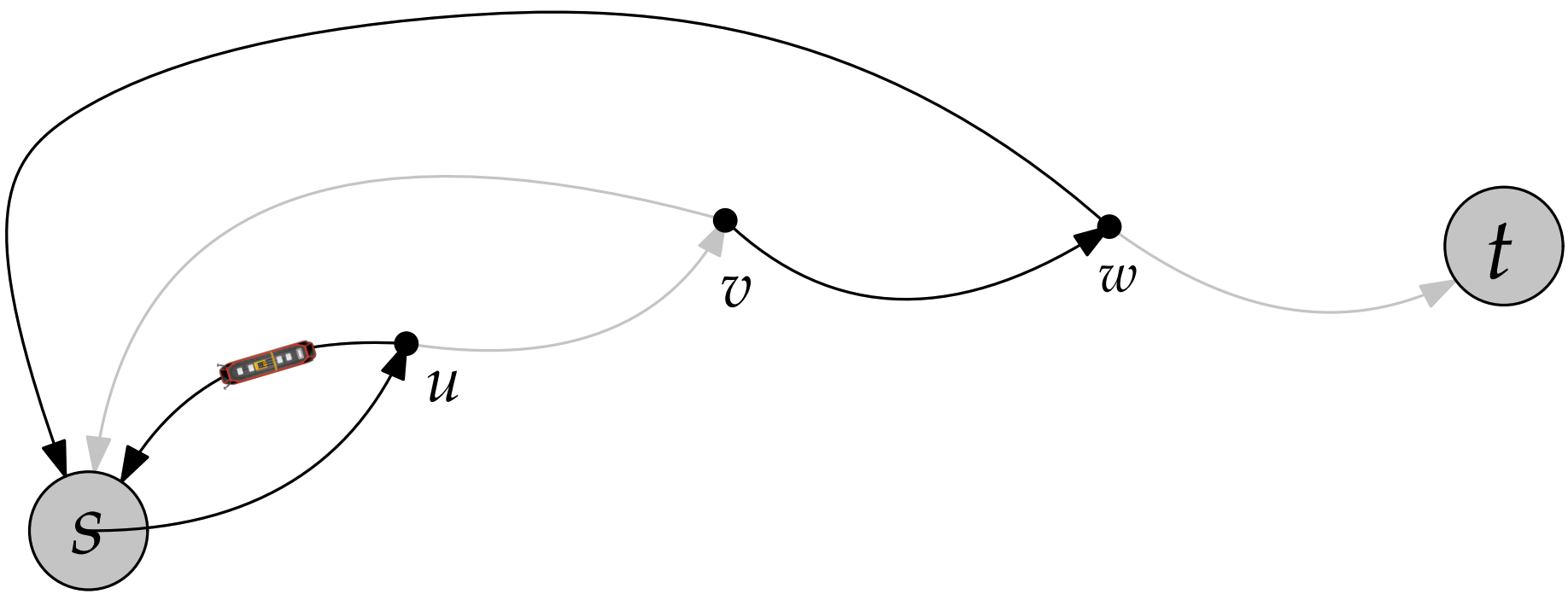
Arrival



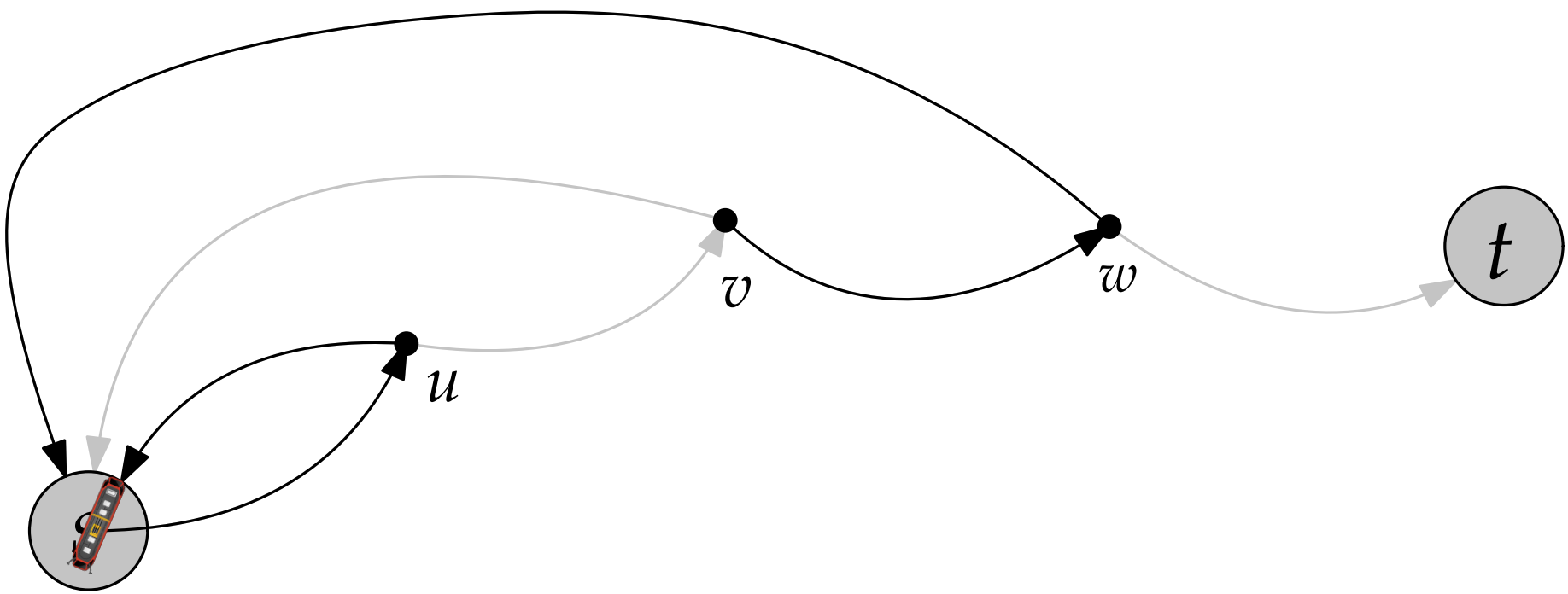
Arrival



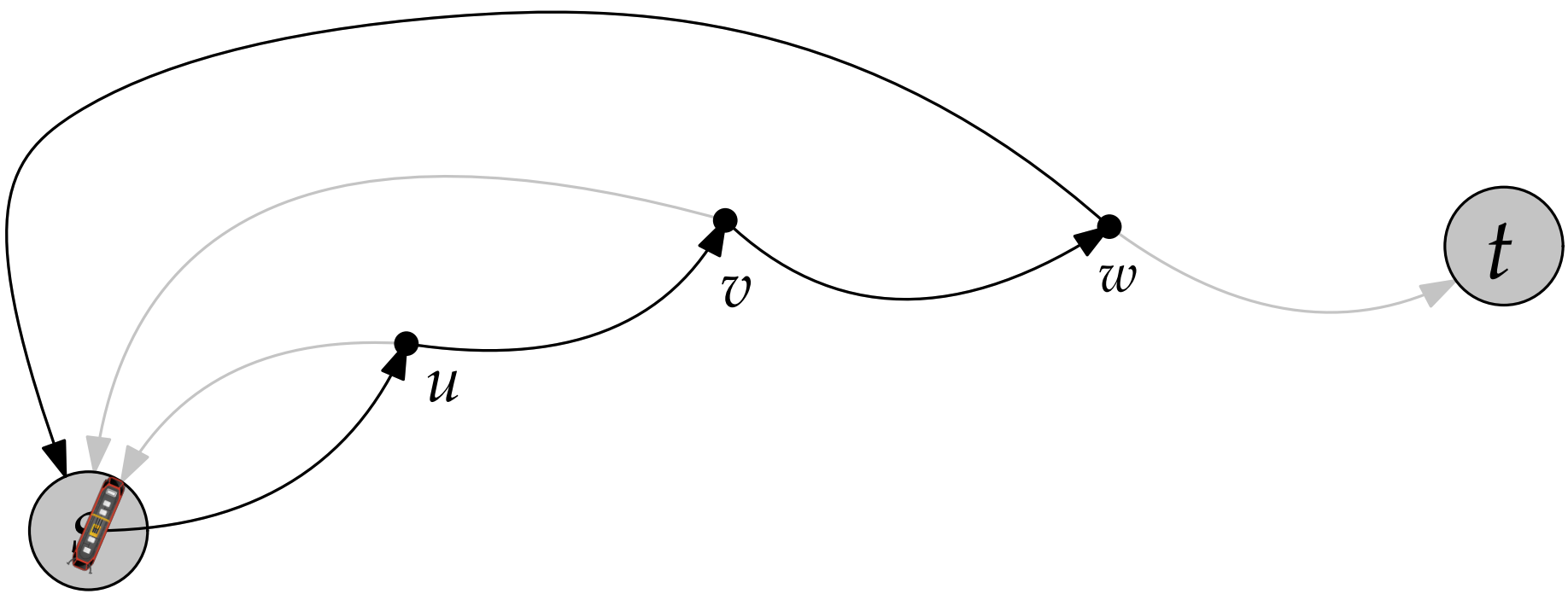
Arrival



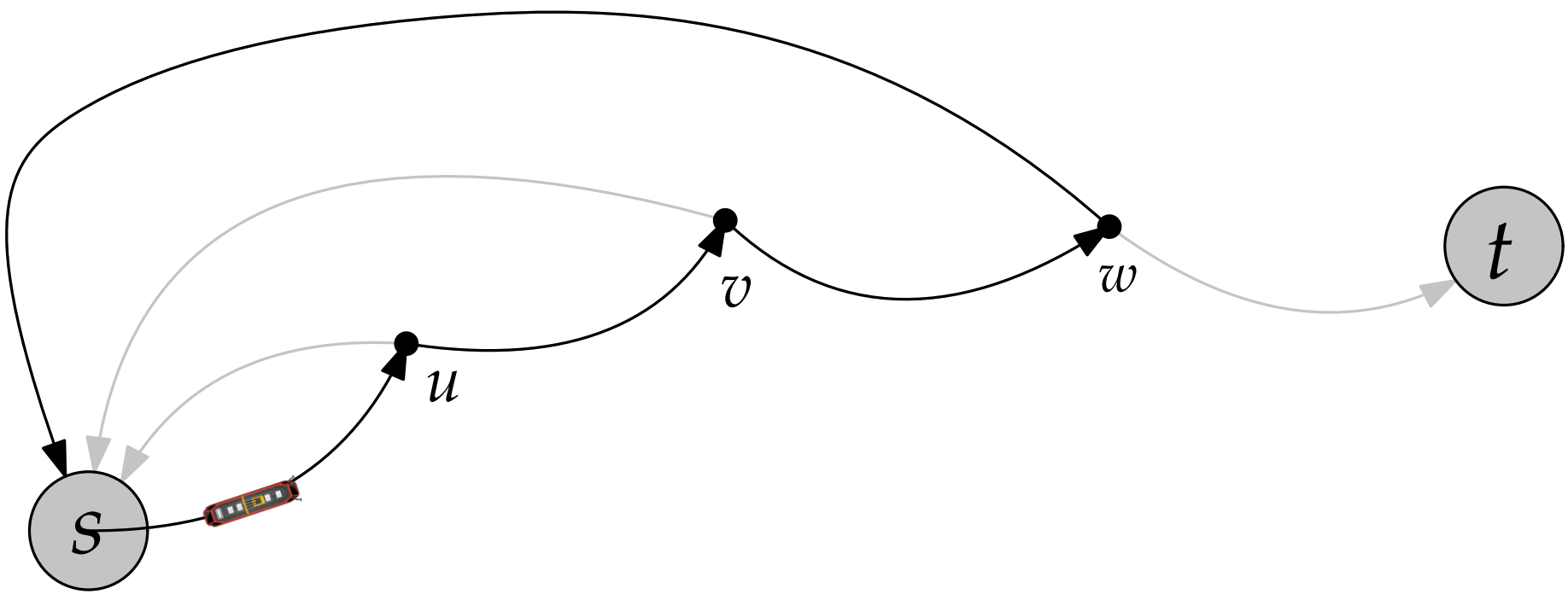
Arrival



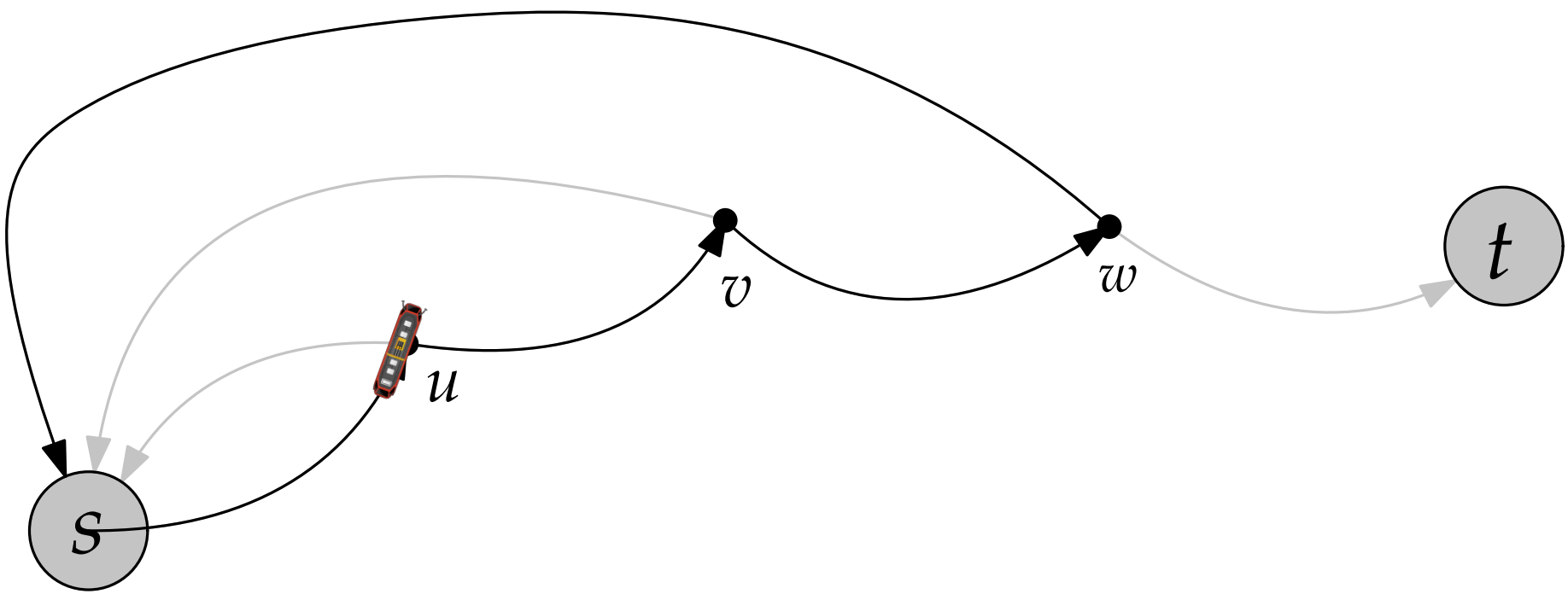
Arrival



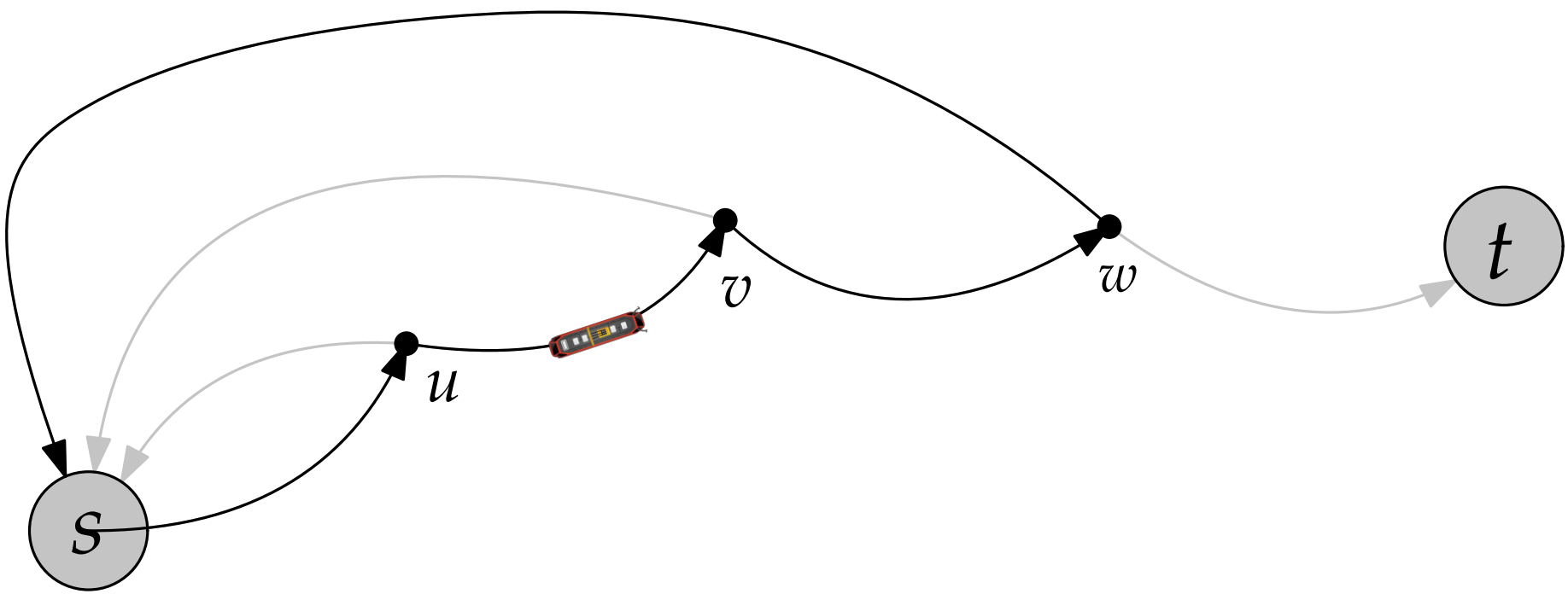
Arrival



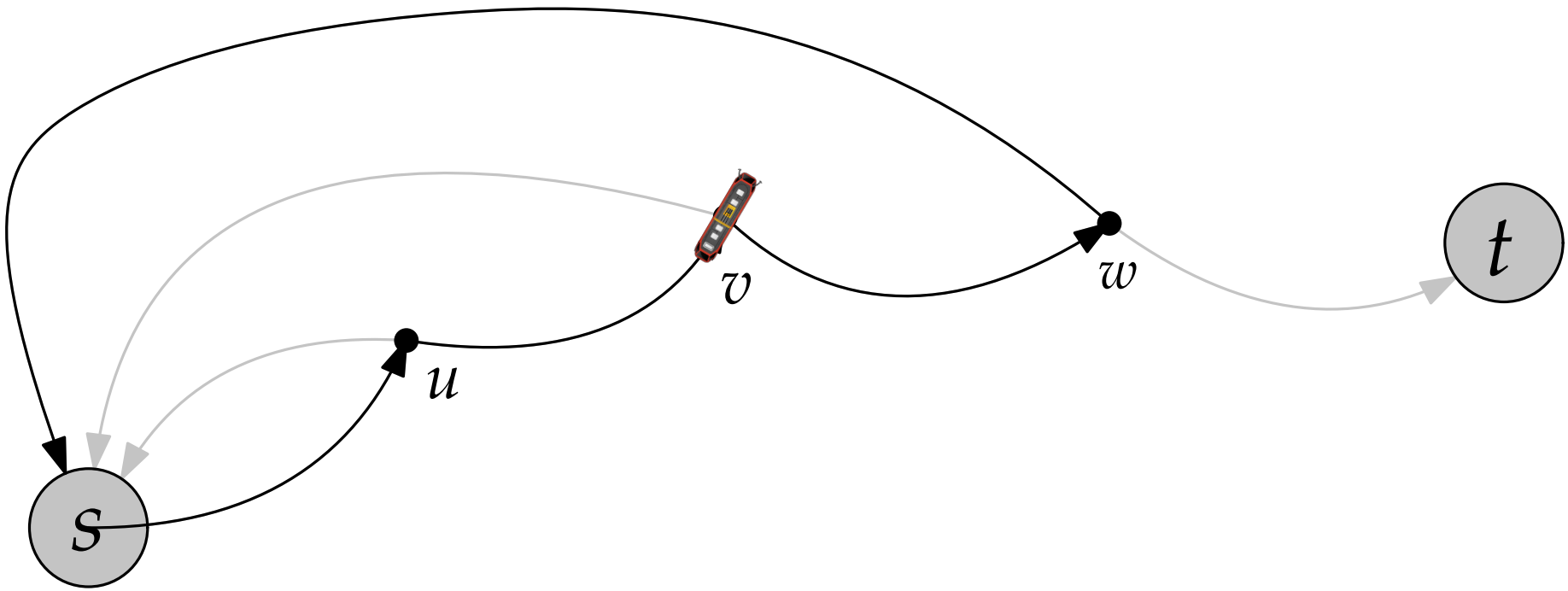
Arrival



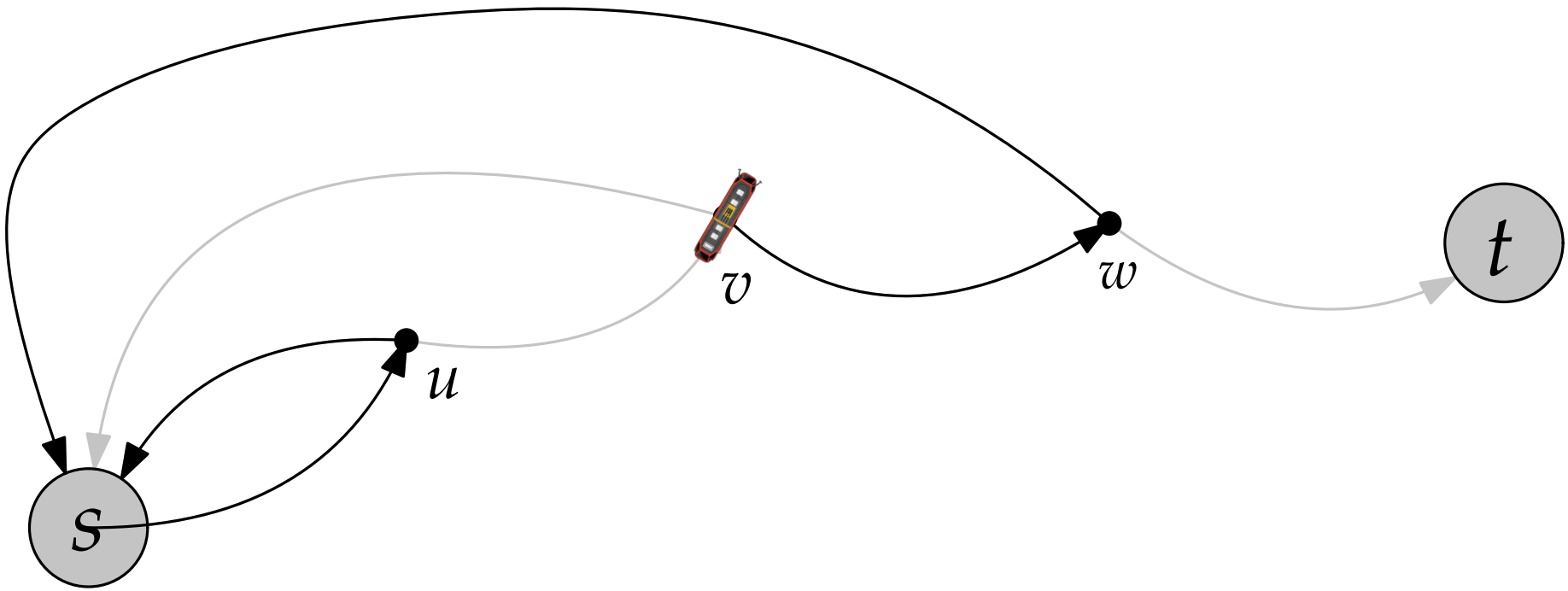
Arrival



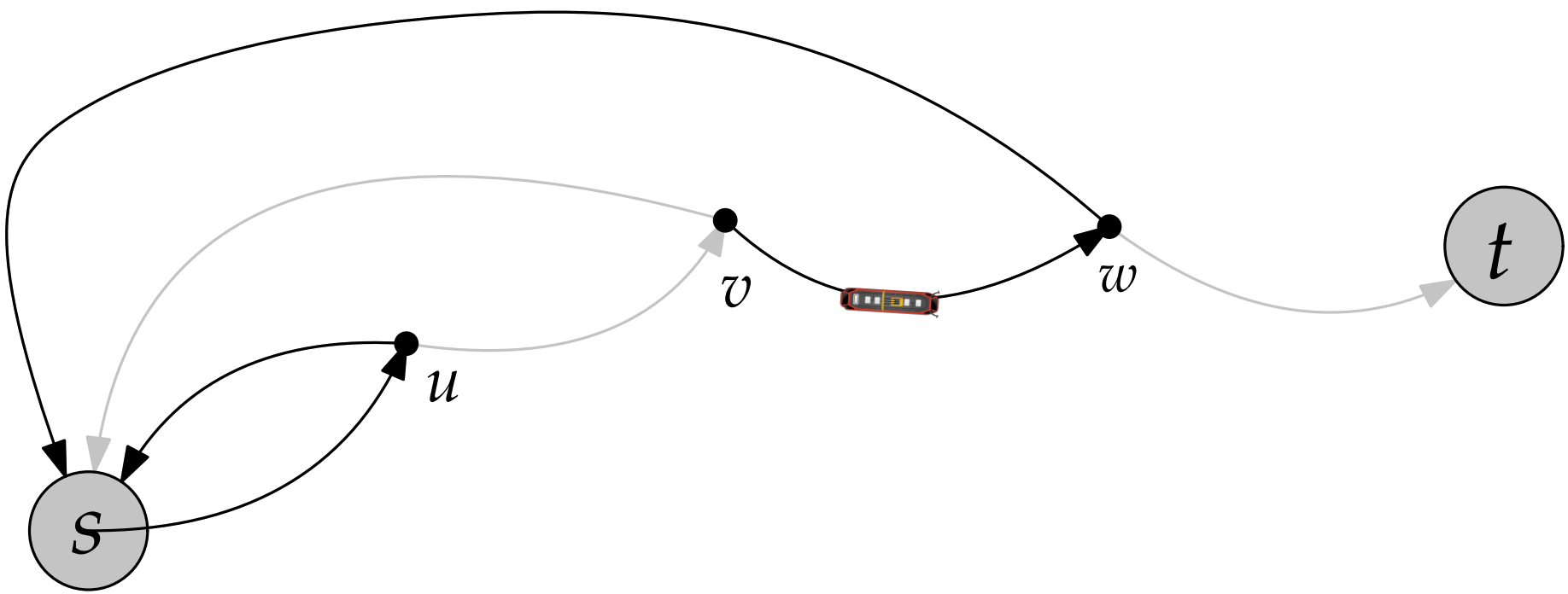
Arrival



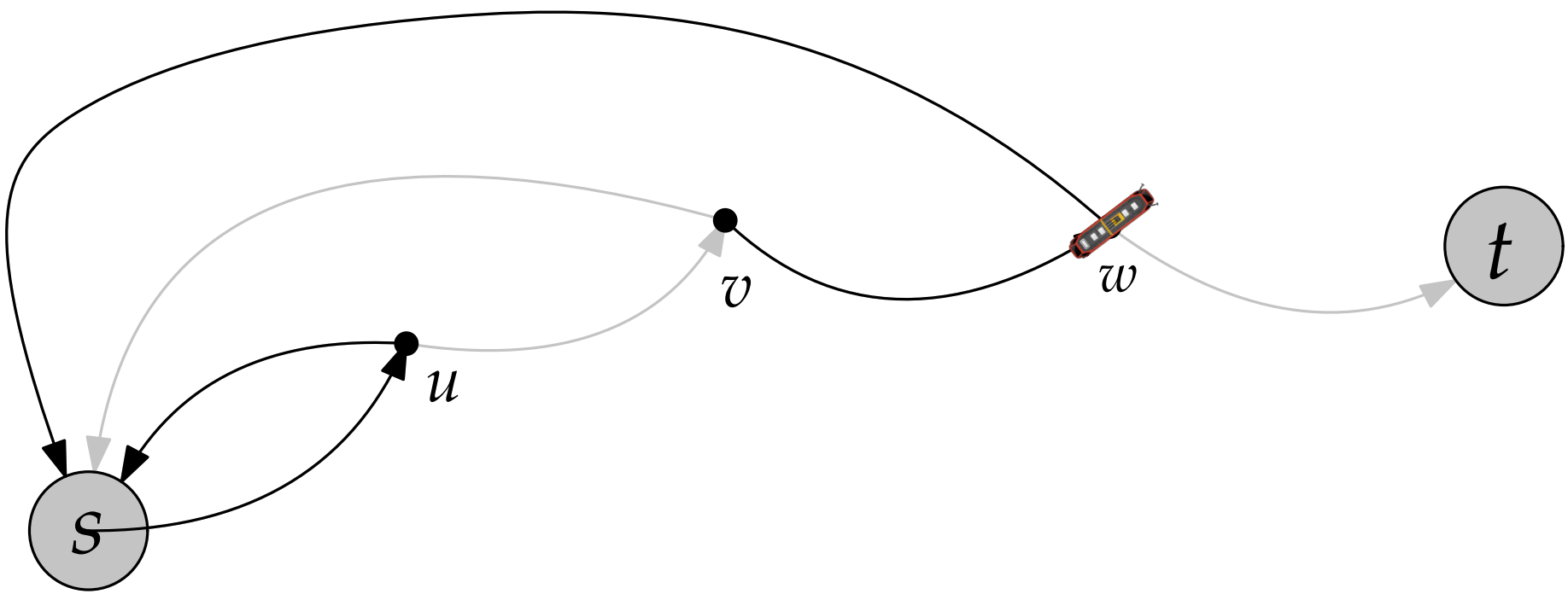
Arrival



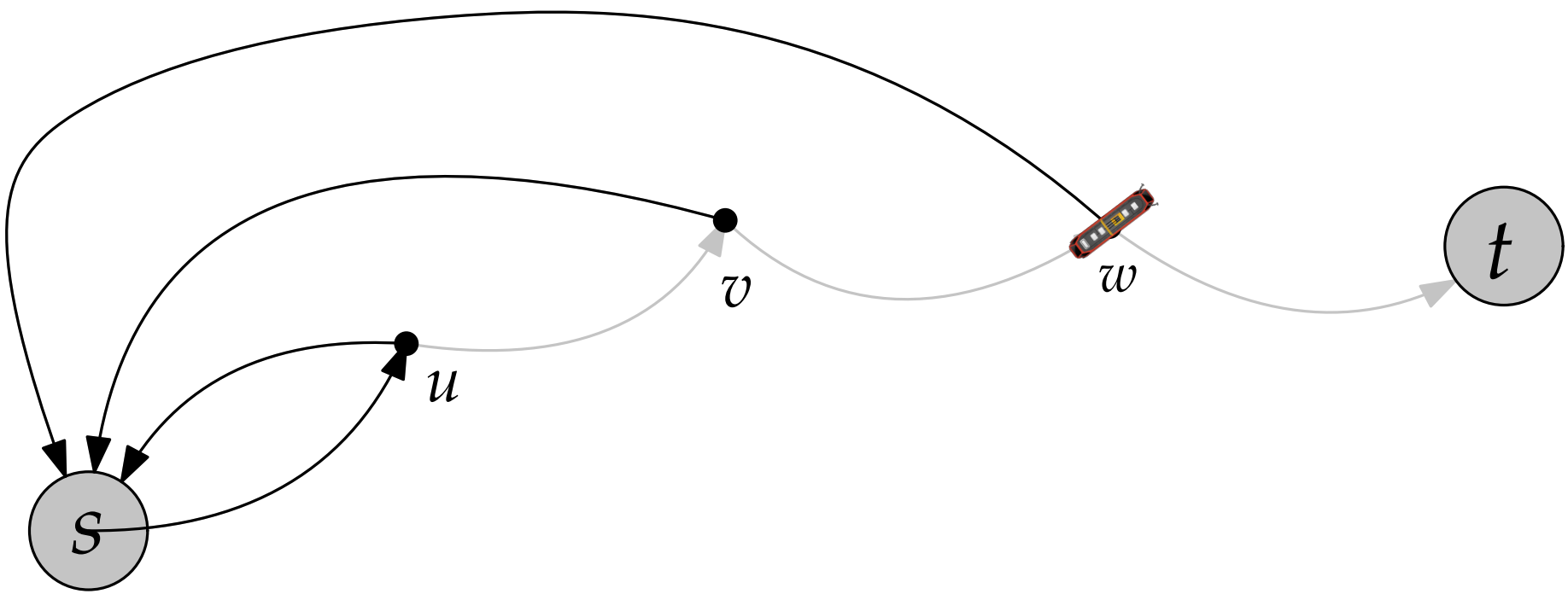
Arrival



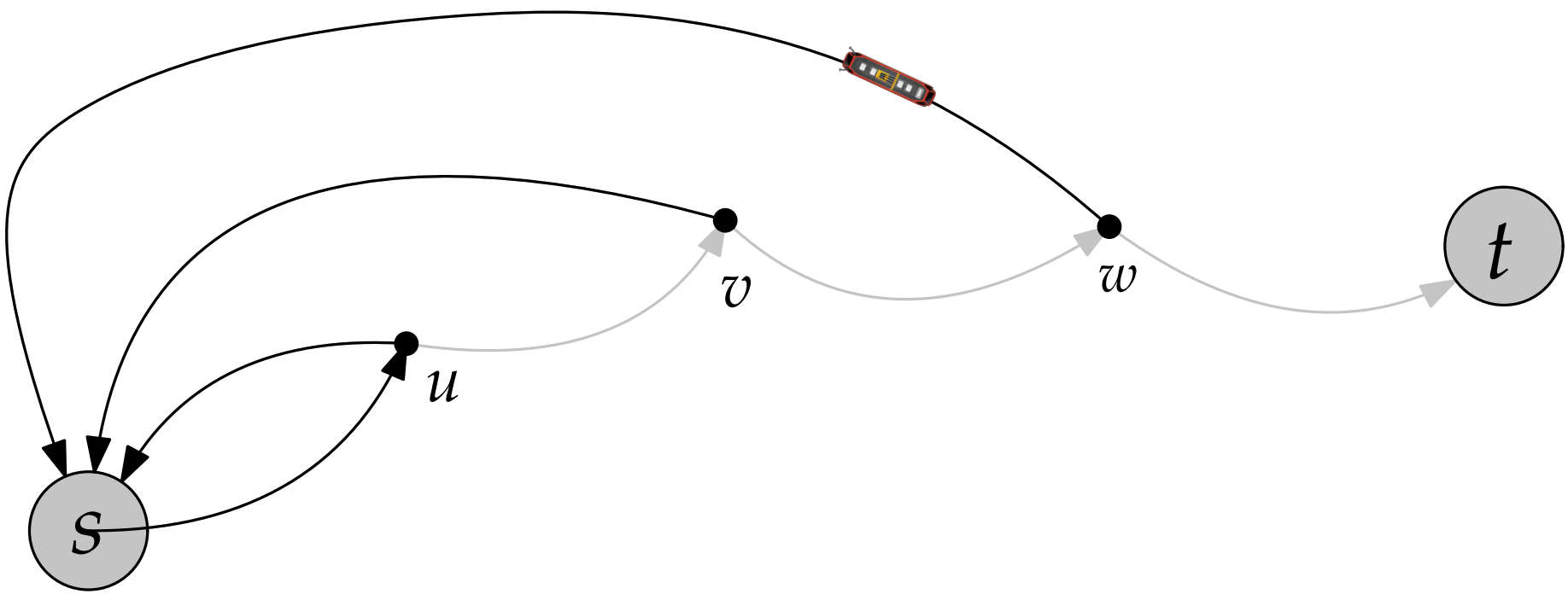
Arrival



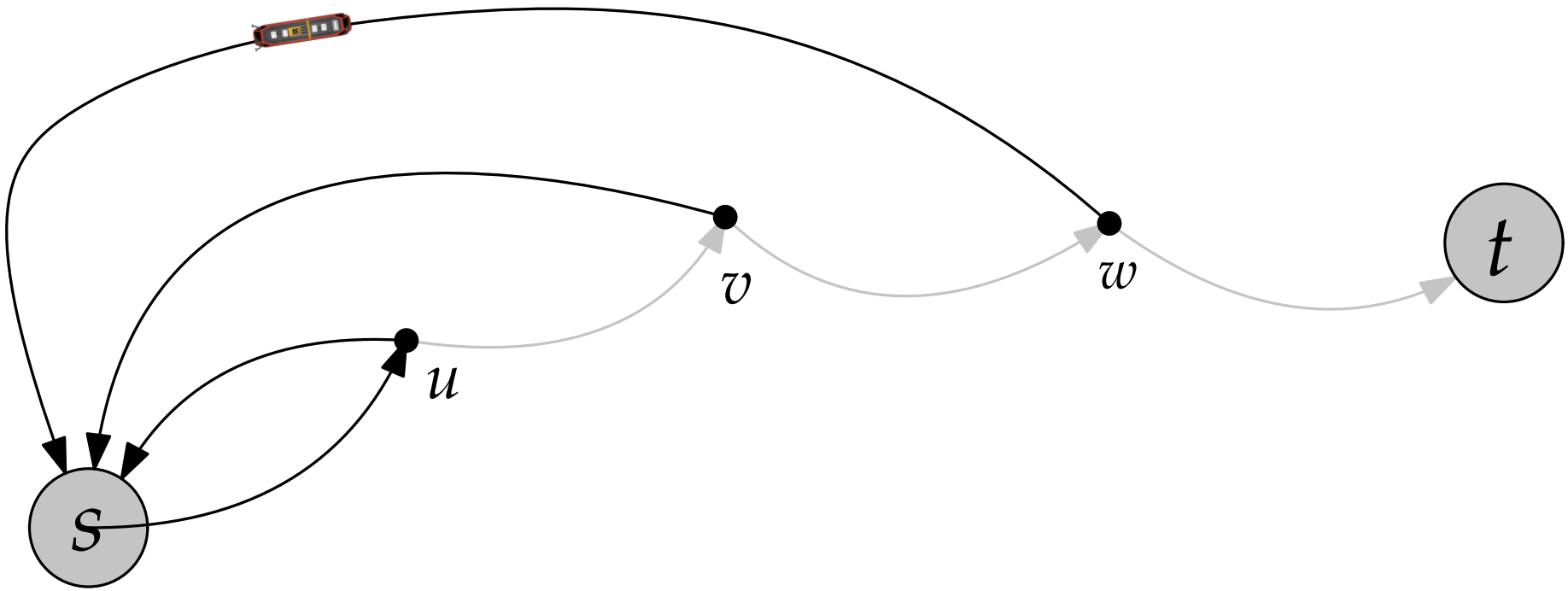
Arrival



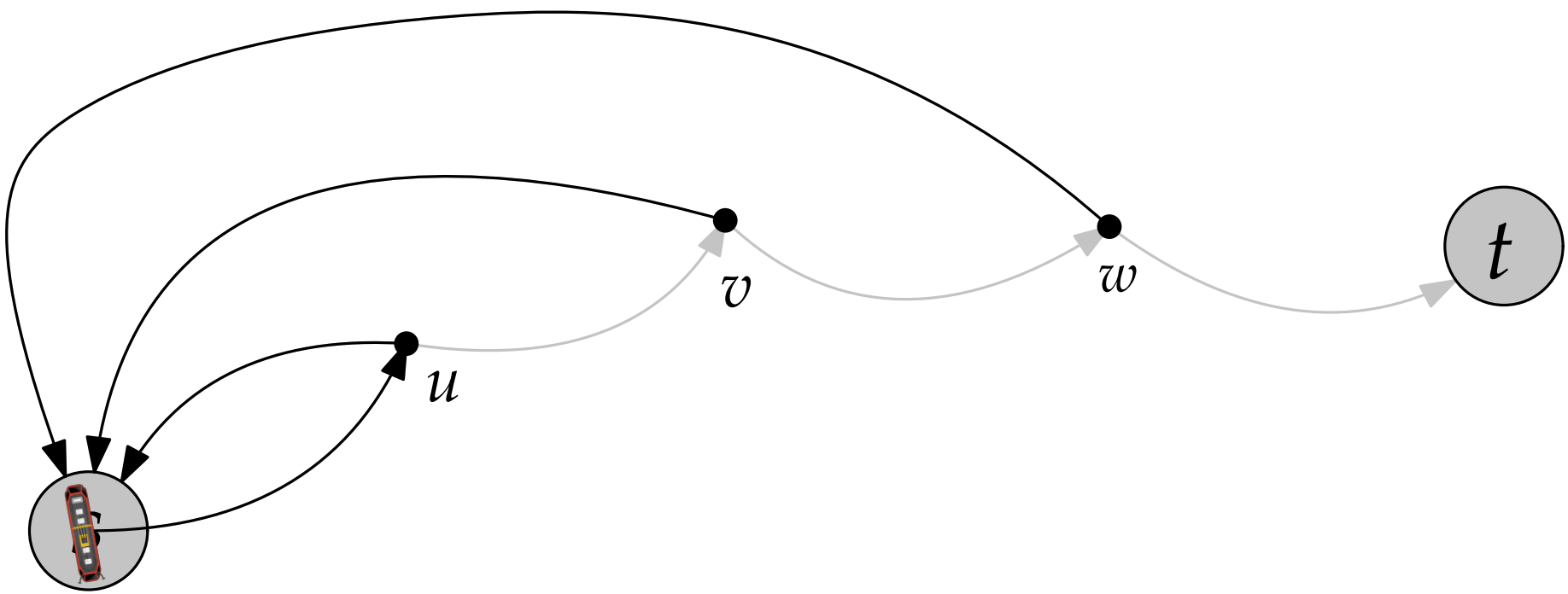
Arrival



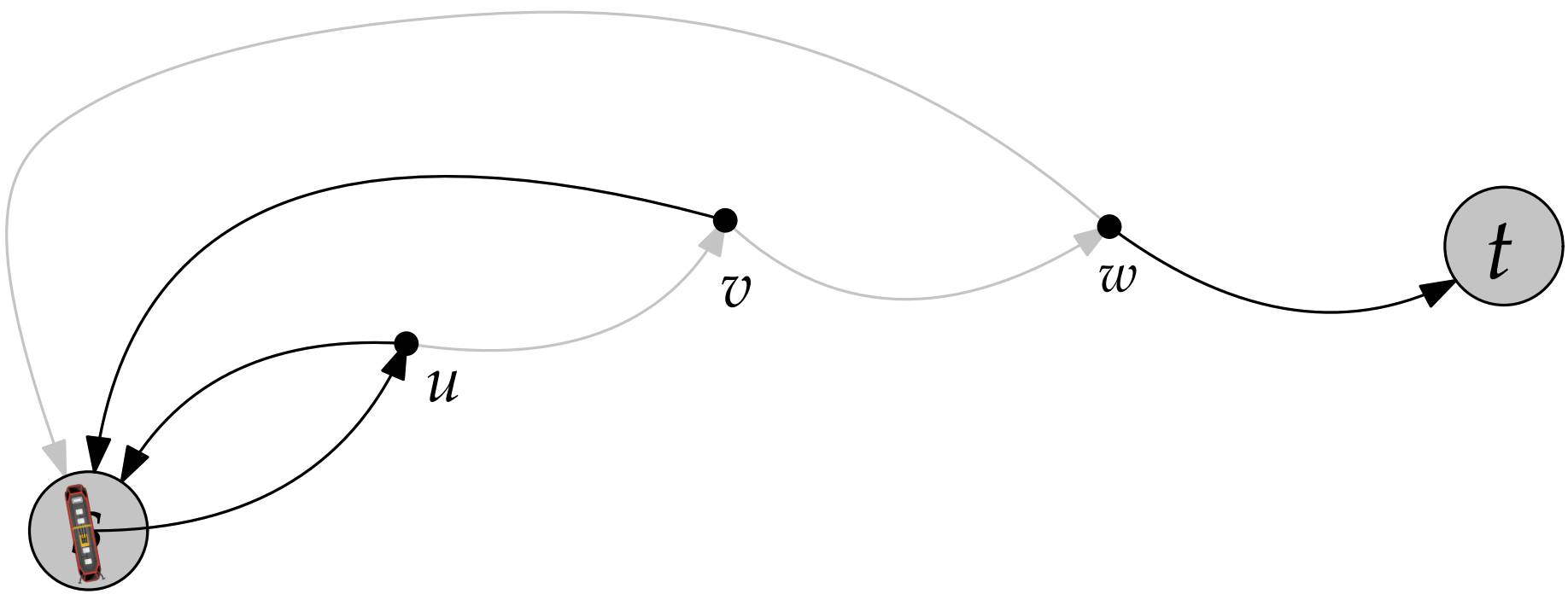
Arrival



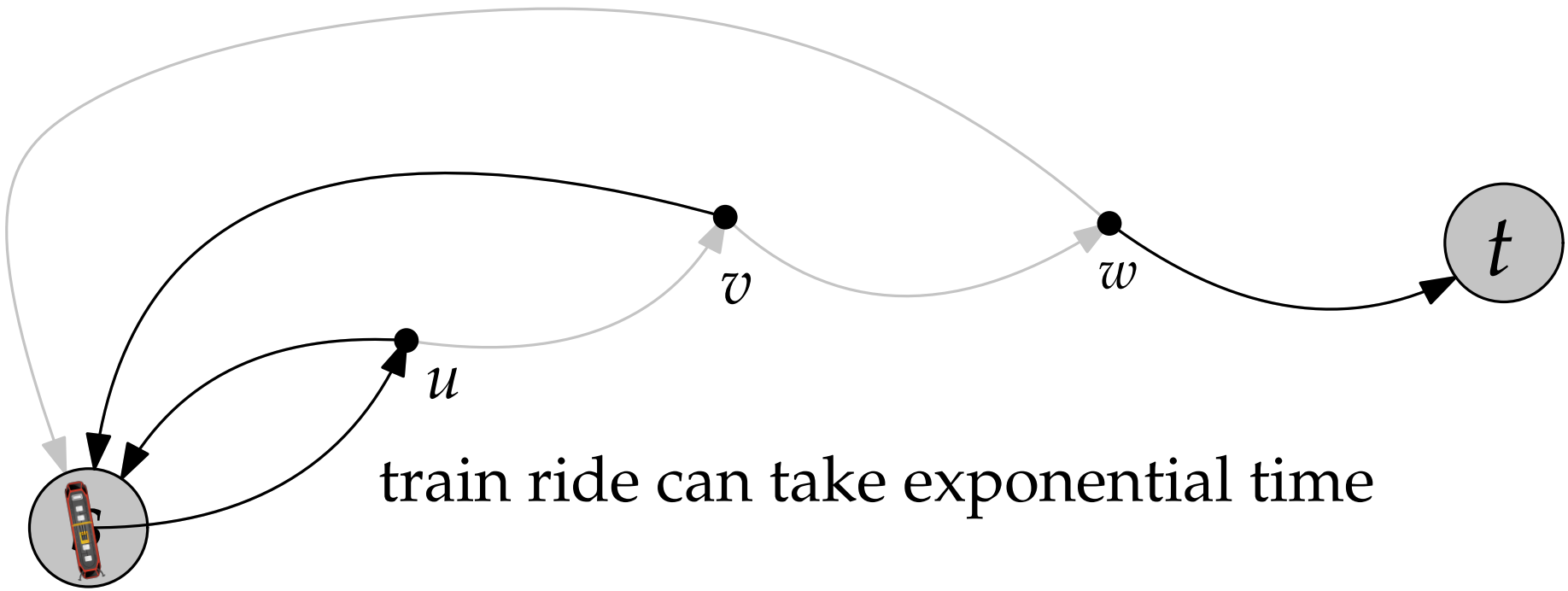
Arrival



Arrival

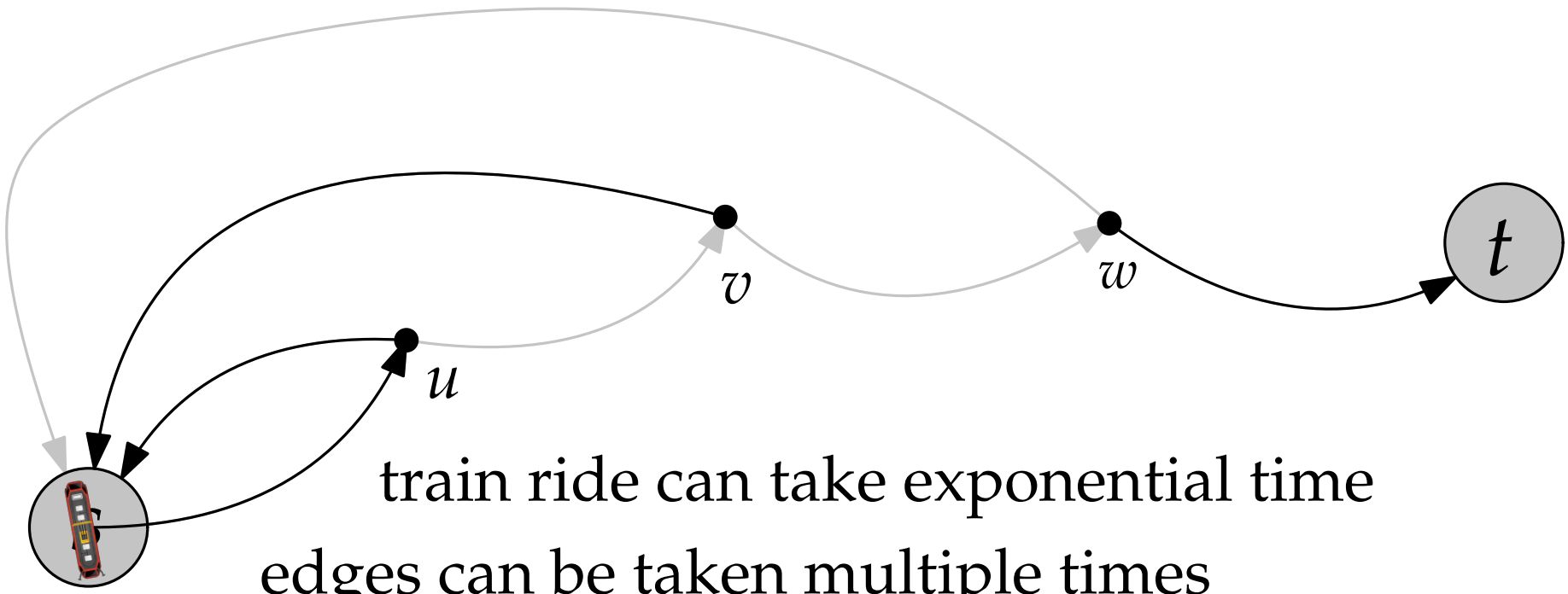


Arrival



train ride can take exponential time

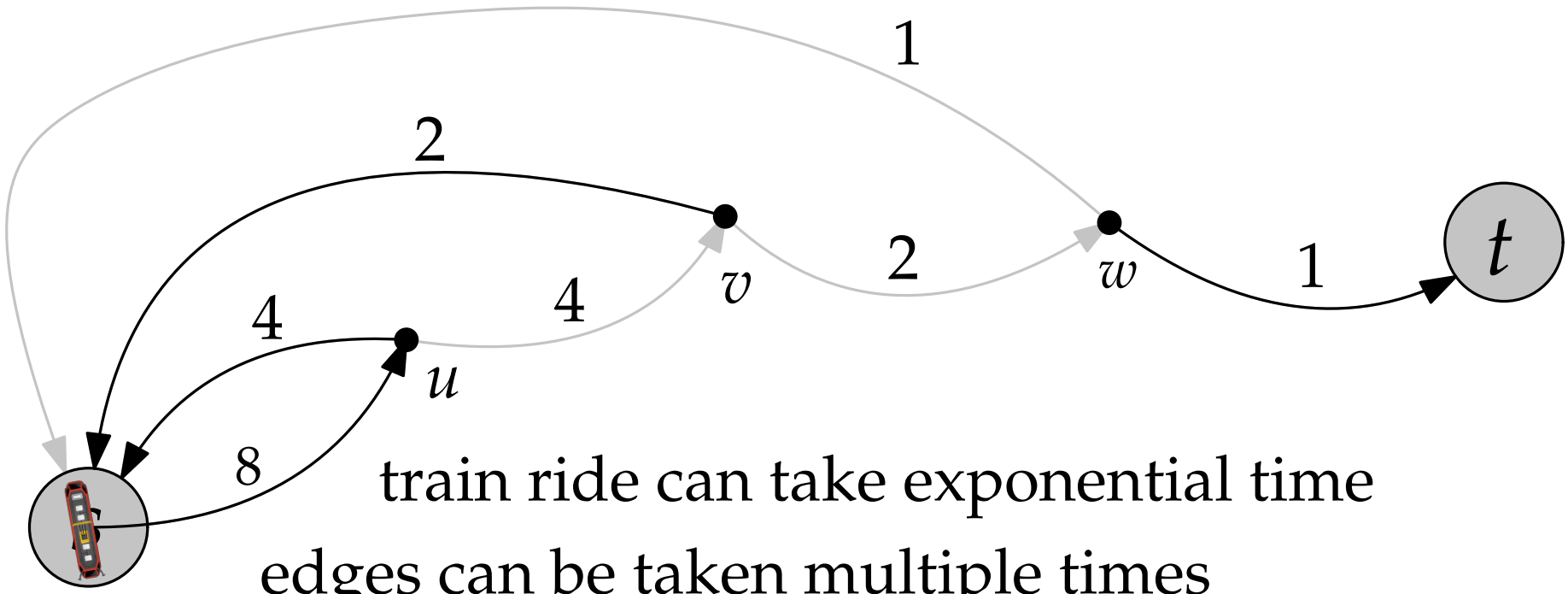
Arrival



train ride can take exponential time

edges can be taken multiple times

Arrival

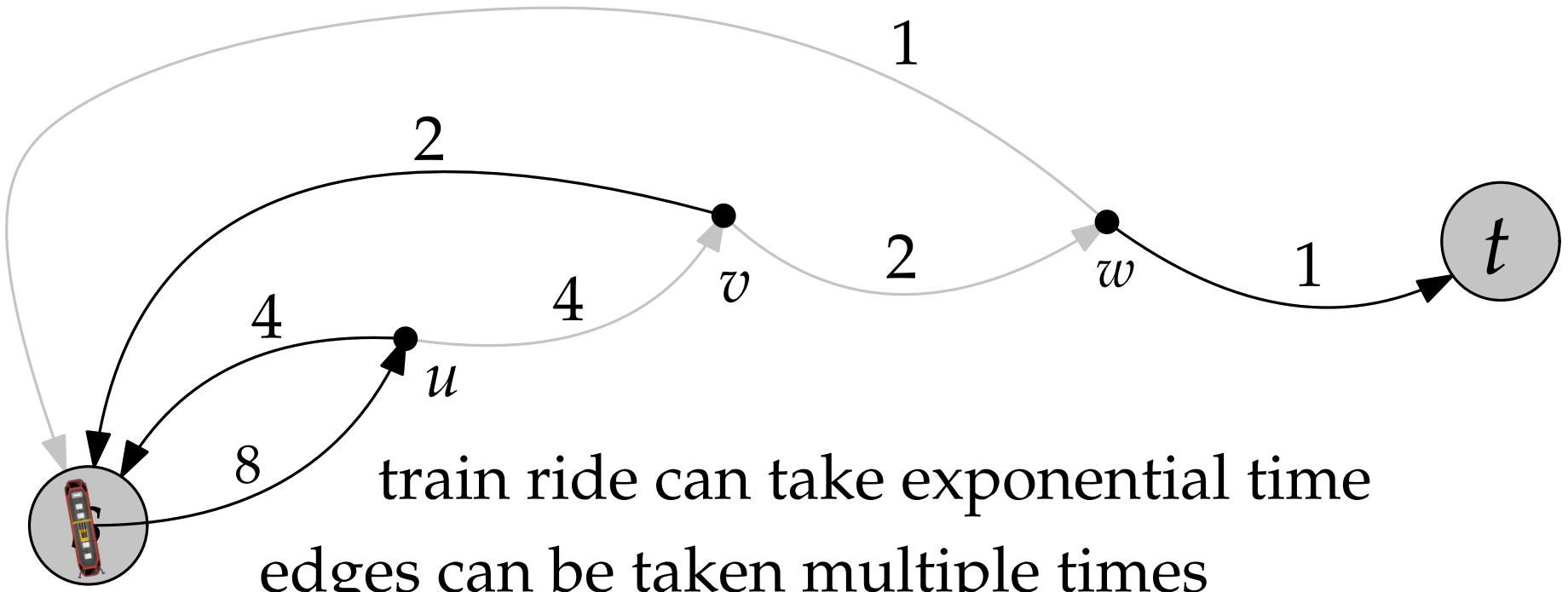


train ride can take exponential time

edges can be taken multiple times

ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in NP.



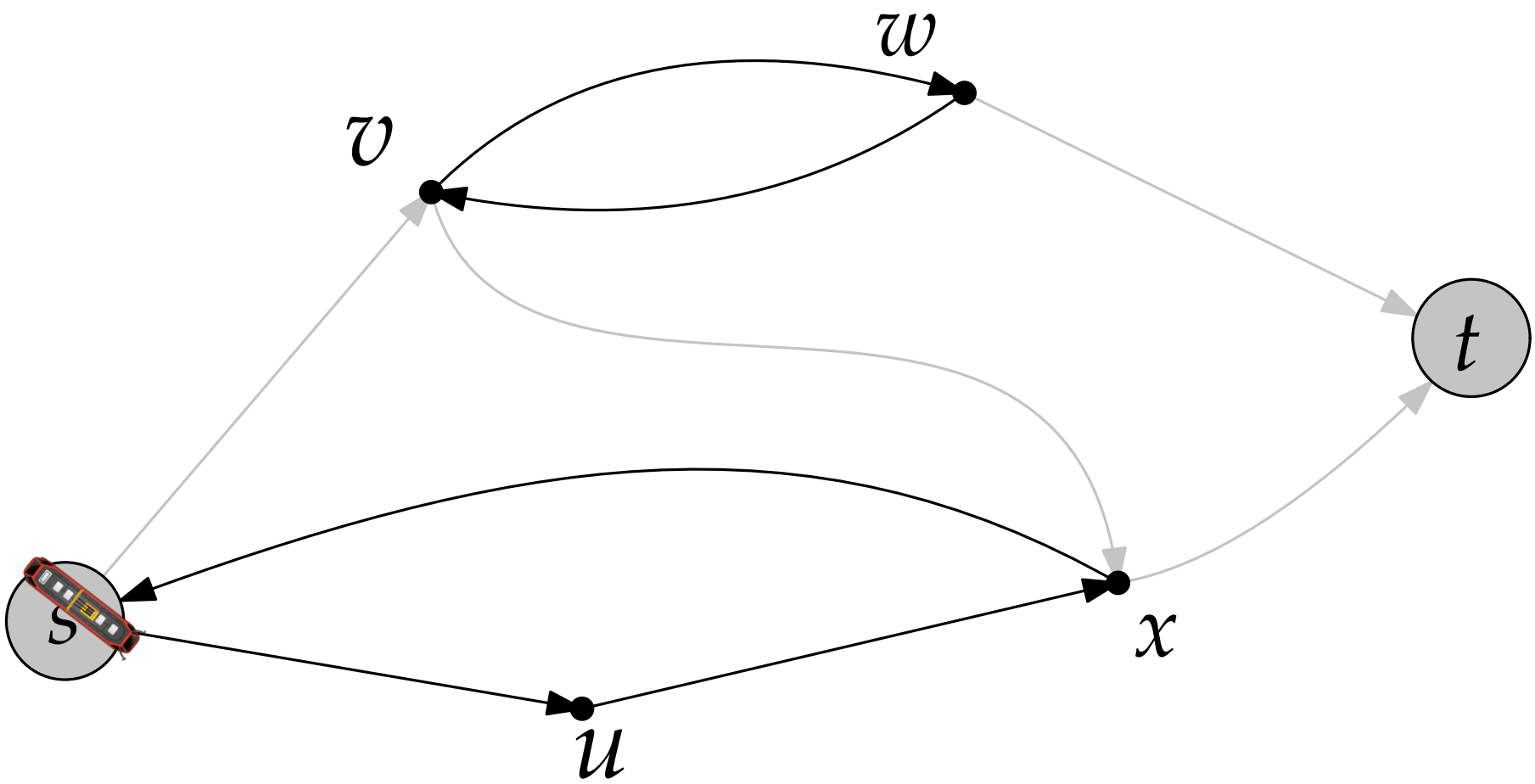
train ride can take exponential time

edges can be taken multiple times

We call this the *profile* of the train ride from s to t.

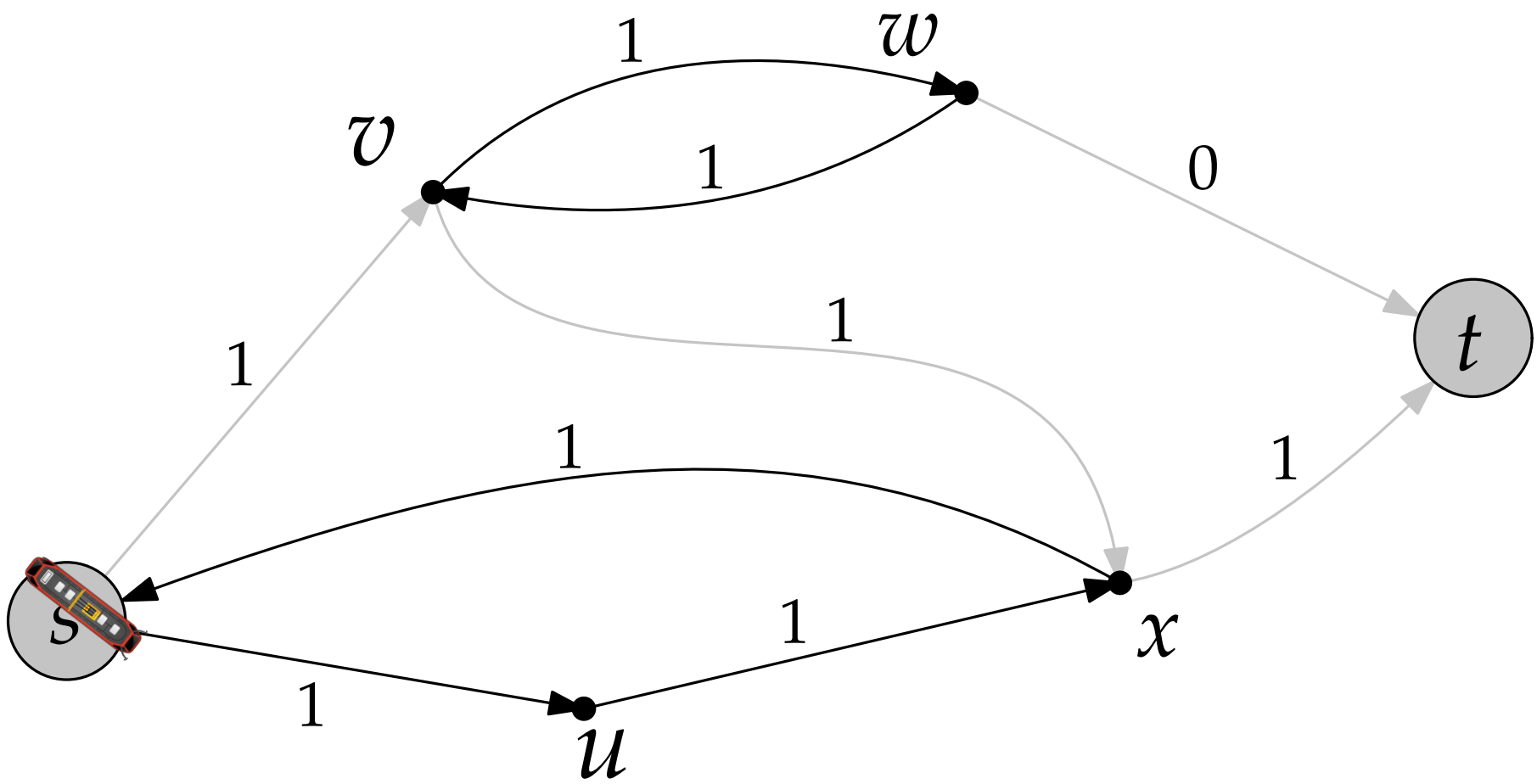
ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in NP.



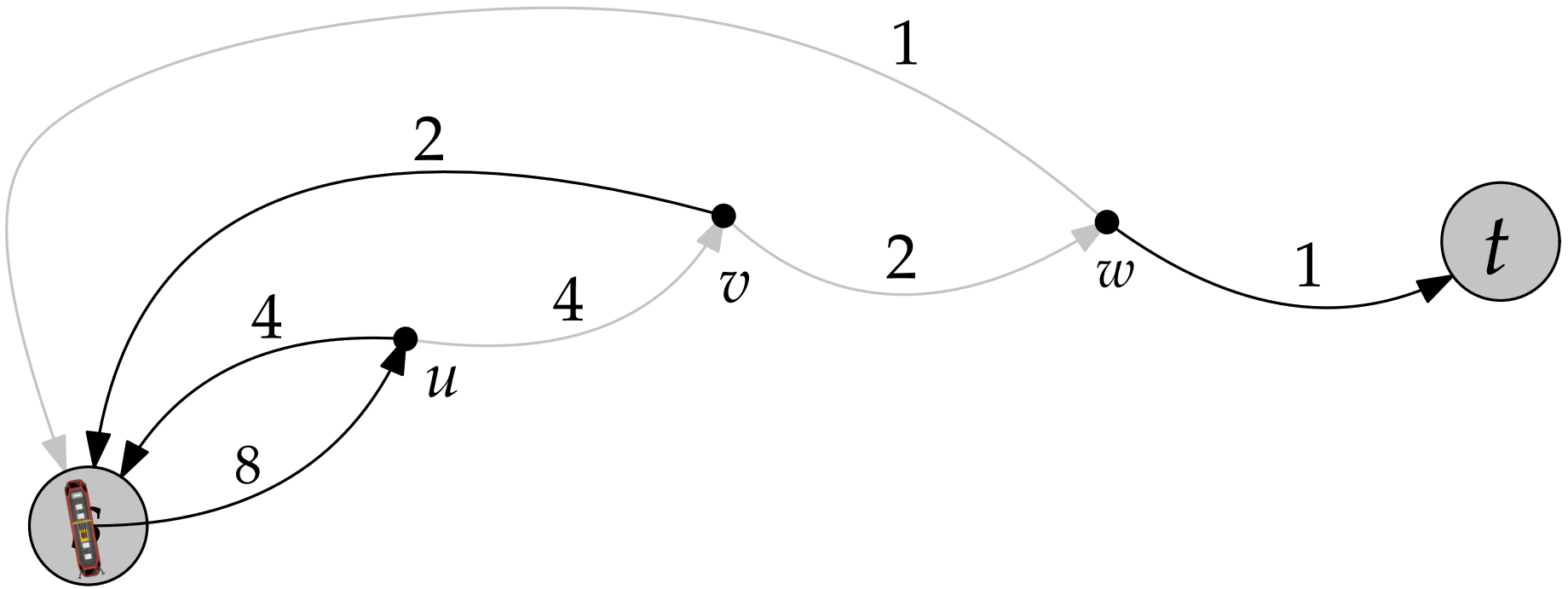
ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in NP.



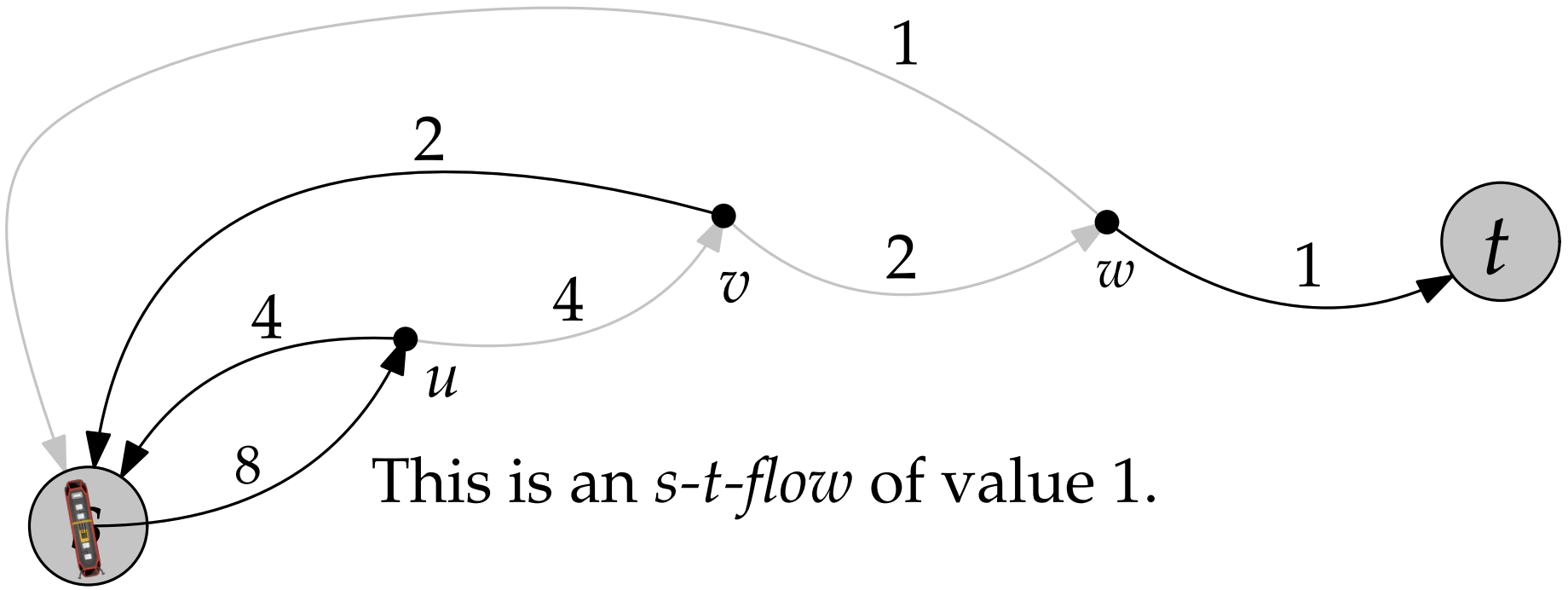
ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in NP.



ARRIVAL

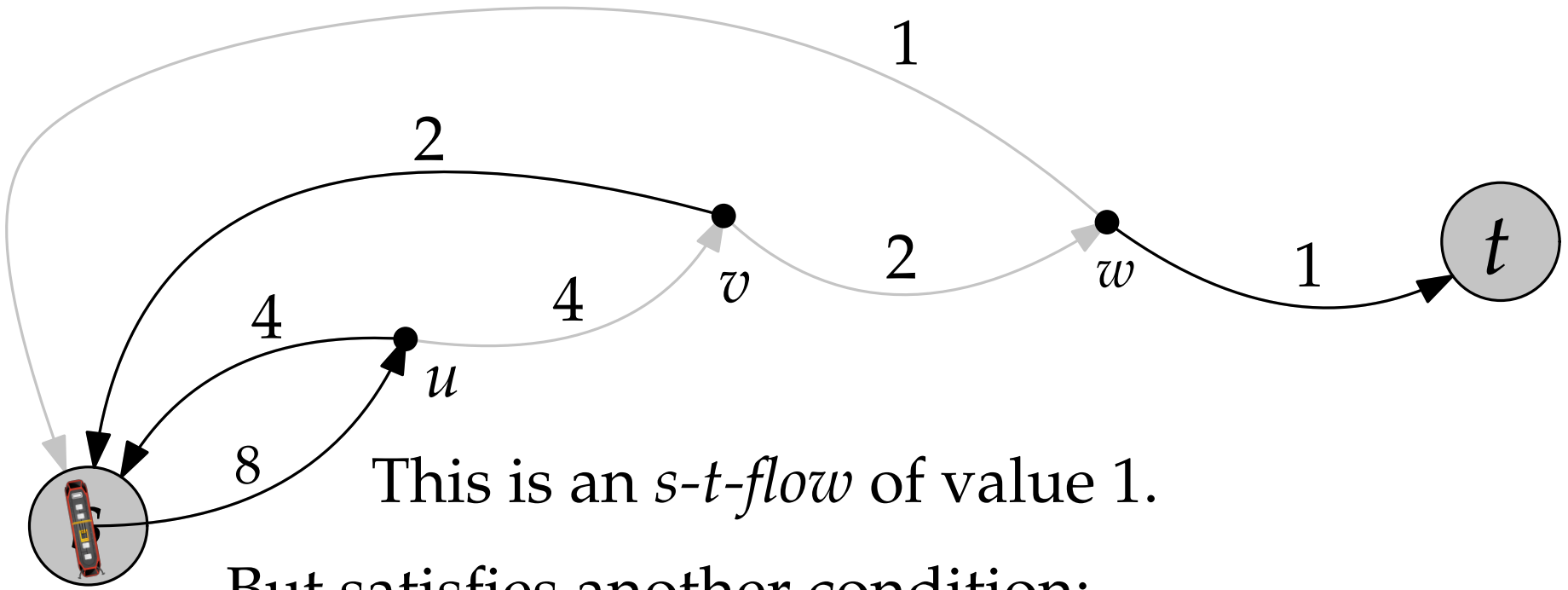
Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in NP.



This is an *s-t-flow* of value 1.

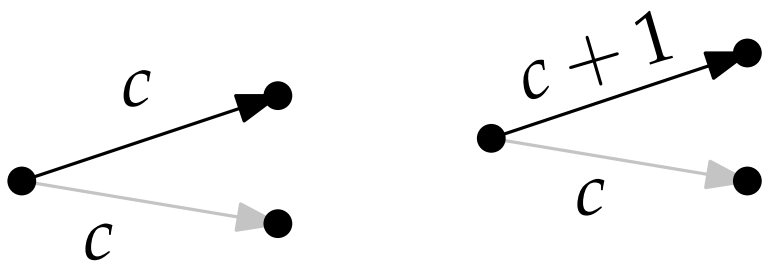
ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in NP.



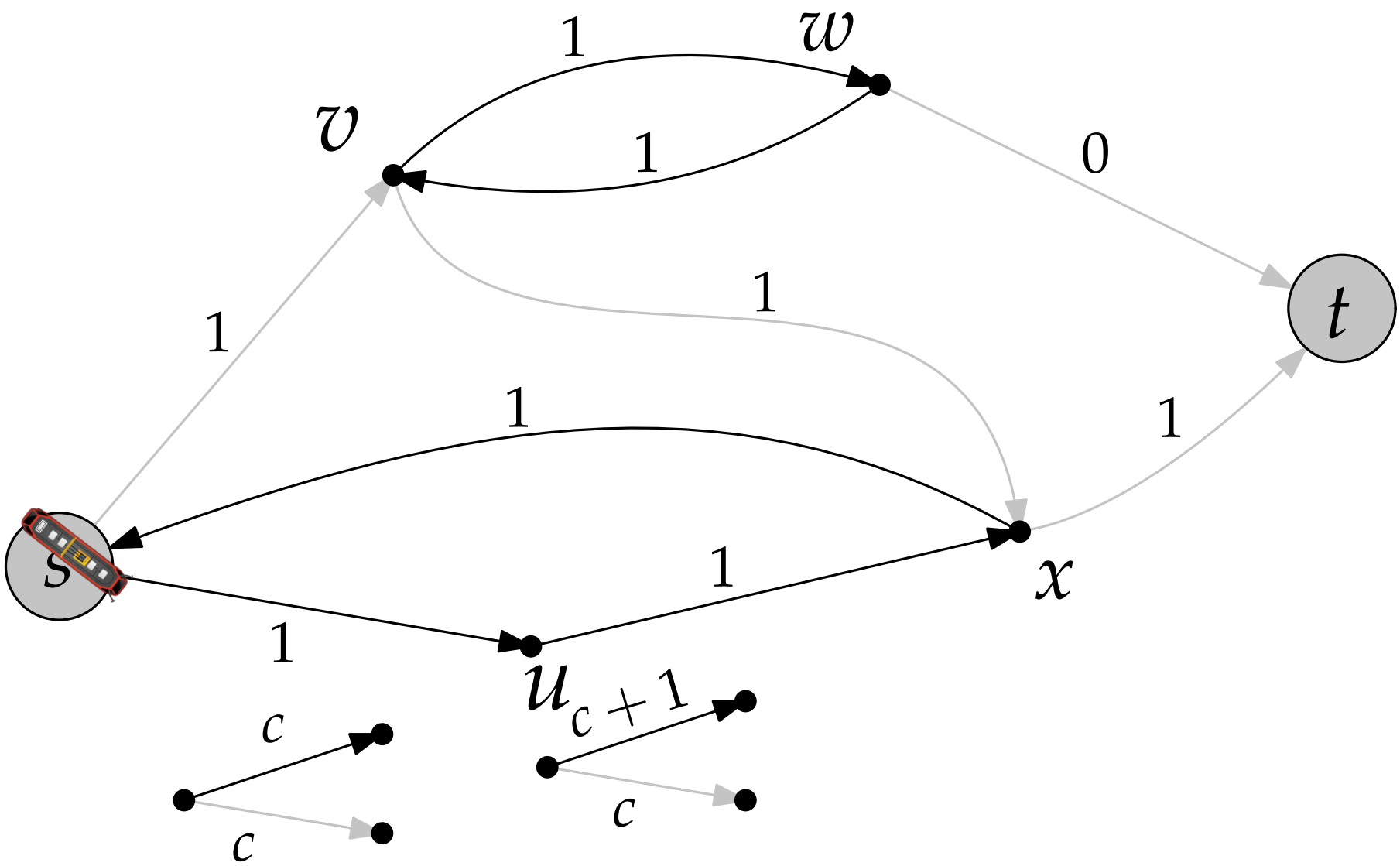
This is an *s-t-flow* of value 1.

But satisfies another condition:



ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in NP.



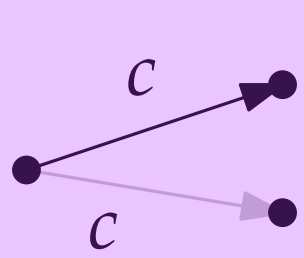
ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in NP.

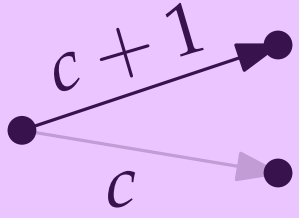
Definition. A function $f : E \to \mathbb{N}$ is a *train flow* from s to t if

$$\sum_w f(v,w) - \sum_u f(u,v) = \begin{cases} 1 & \text{if } v = s \\ -1 & \text{if } v = t \\ 0 & \text{else.} \end{cases}$$

and



or



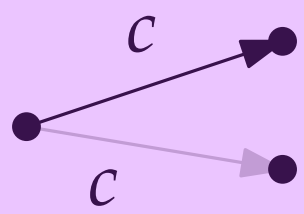
at all vertices.

Arrival

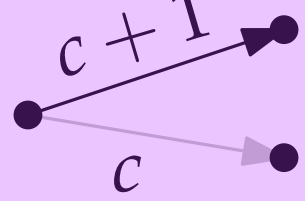
Definition. A function $f : E \to \mathbb{N}$ is a *train flow* from s to t if

$$\sum_w f(v,w) - \sum_u f(u,v) = \begin{cases} 1 & \text{if } v = s \\ -1 & \text{if } v = t \\ 0 & \text{else.} \end{cases}$$

and



or



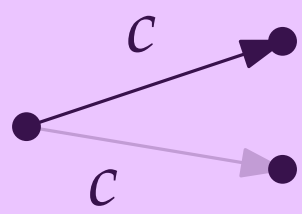
at all vertices.

Arrival

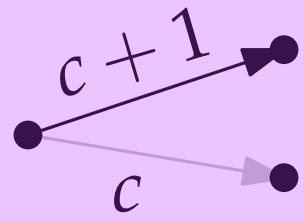
Definition. A function $f : E \to \mathbb{N}$ is a *train flow* from s to t if

$$\sum_w f(v,w) - \sum_u f(u,v) = \begin{cases} 1 & \text{if } v = s \\ -1 & \text{if } v = t \\ 0 & \text{else.} \end{cases}$$

and



or



at all vertices.

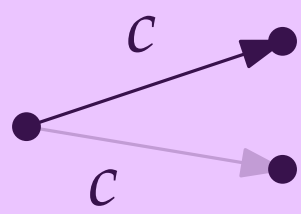
Observation. Every train ride profile is a train flow.

Arrival

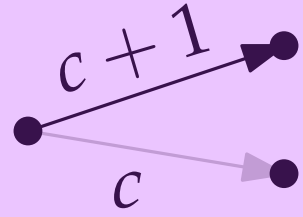
Definition. A function $f : E \to \mathbb{N}$ is a *train flow* from s to t if

$$\sum_w f(v,w) - \sum_u f(u,v) = \begin{cases} 1 & \text{if } v = s \\ -1 & \text{if } v = t \\ 0 & \text{else.} \end{cases}$$

and



or



at all vertices.

Observation. Every train ride profile is a train flow.

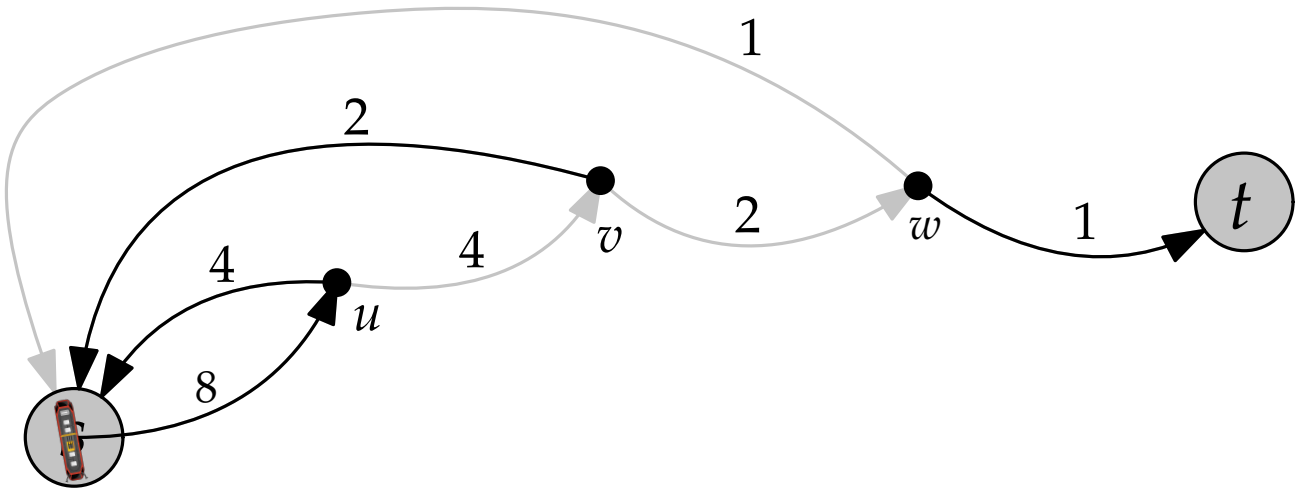
Question. Is every train flow a train ride profile?

Arrival

Question. Is every train flow a train ride profile?

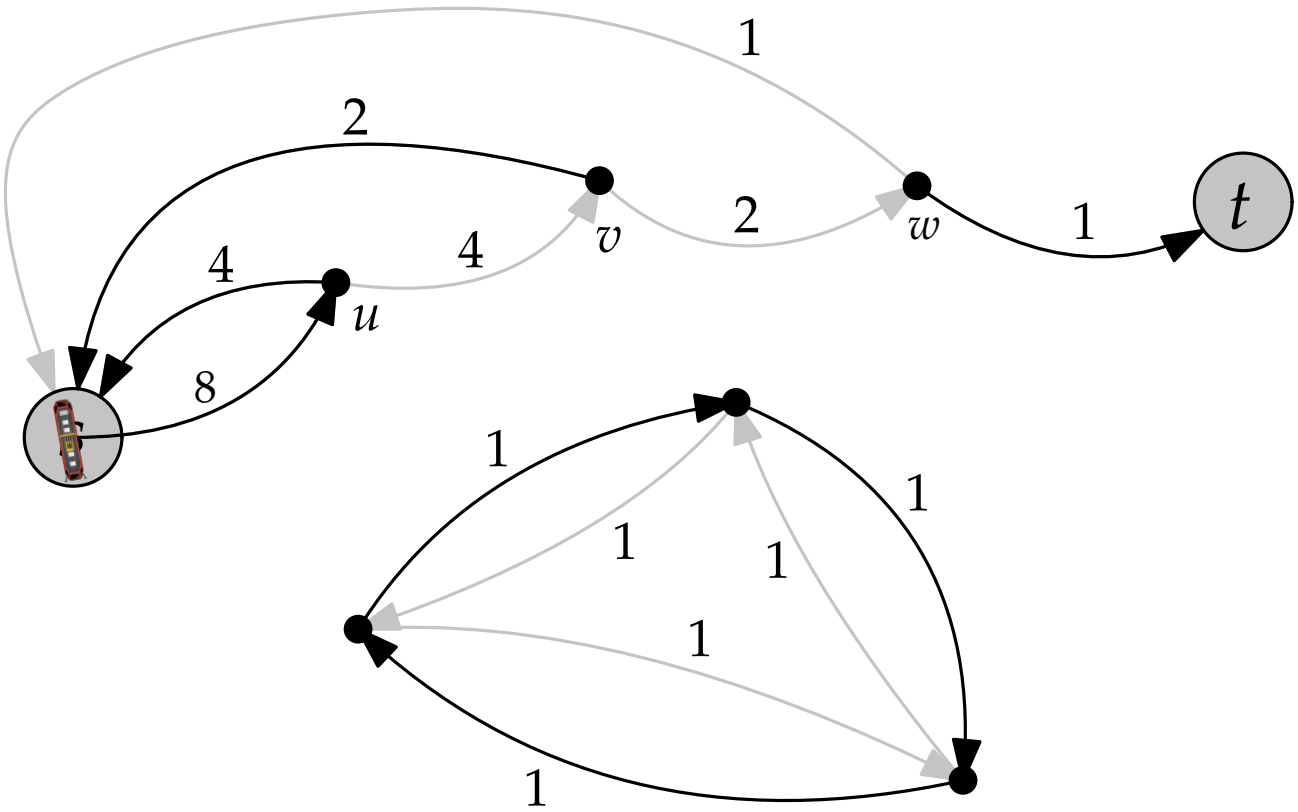
Arrival

Question. Is every train flow a train ride profile?



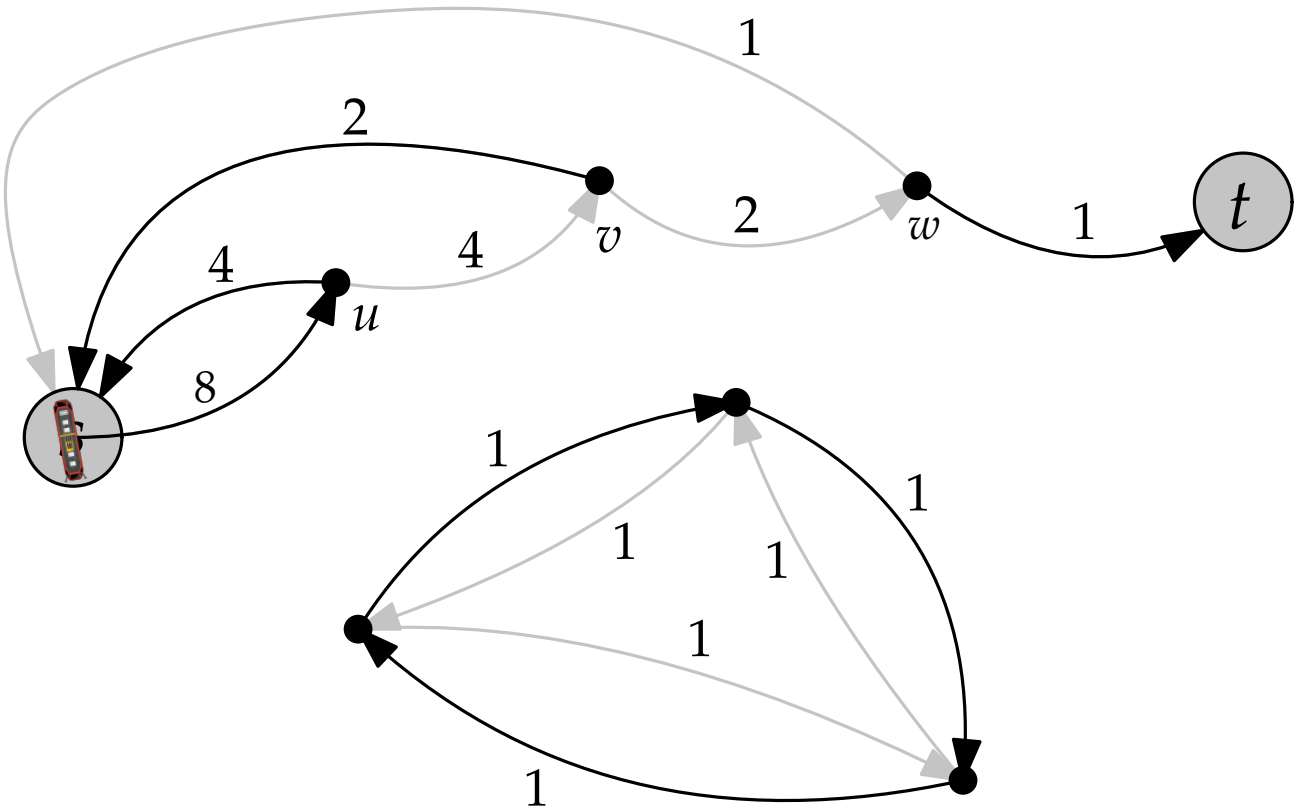
Arrival

Question. Is every train flow a train ride profile?



ARRIVAL

Question. Is every train flow a train ride profile?



...this is a train flow but not a train ride profile...

ARRIVAL

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

ARRIVAL

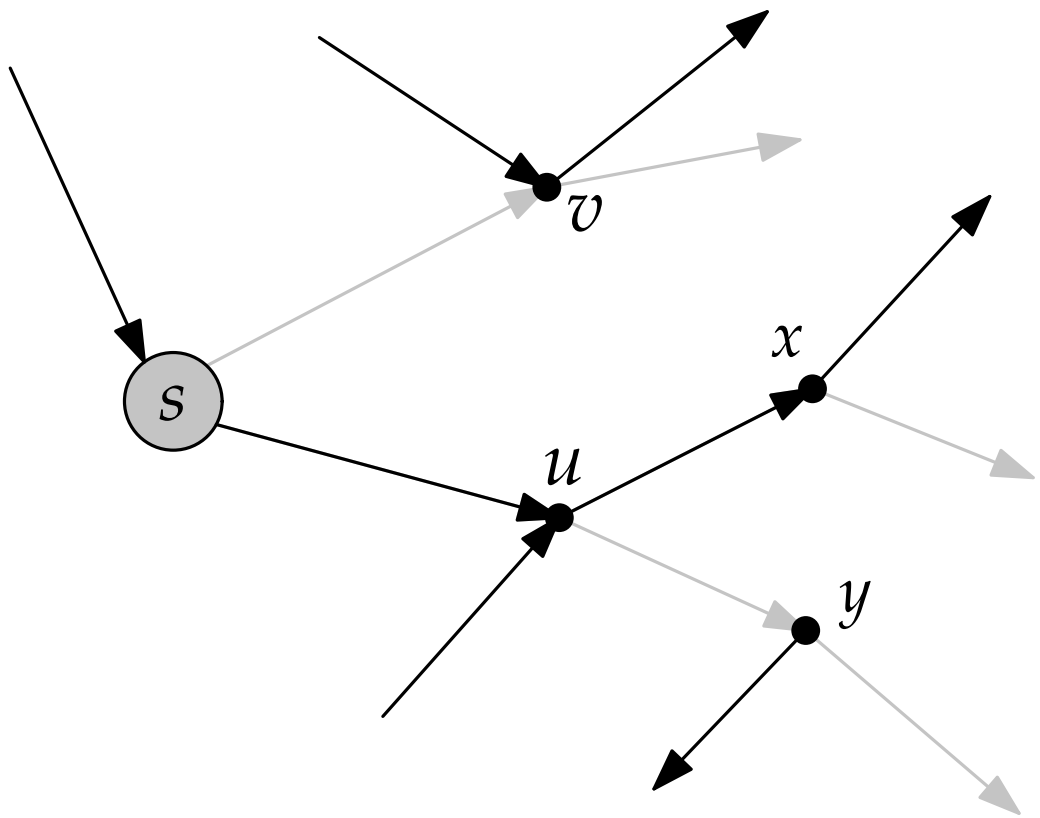
Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!

Arrival

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!

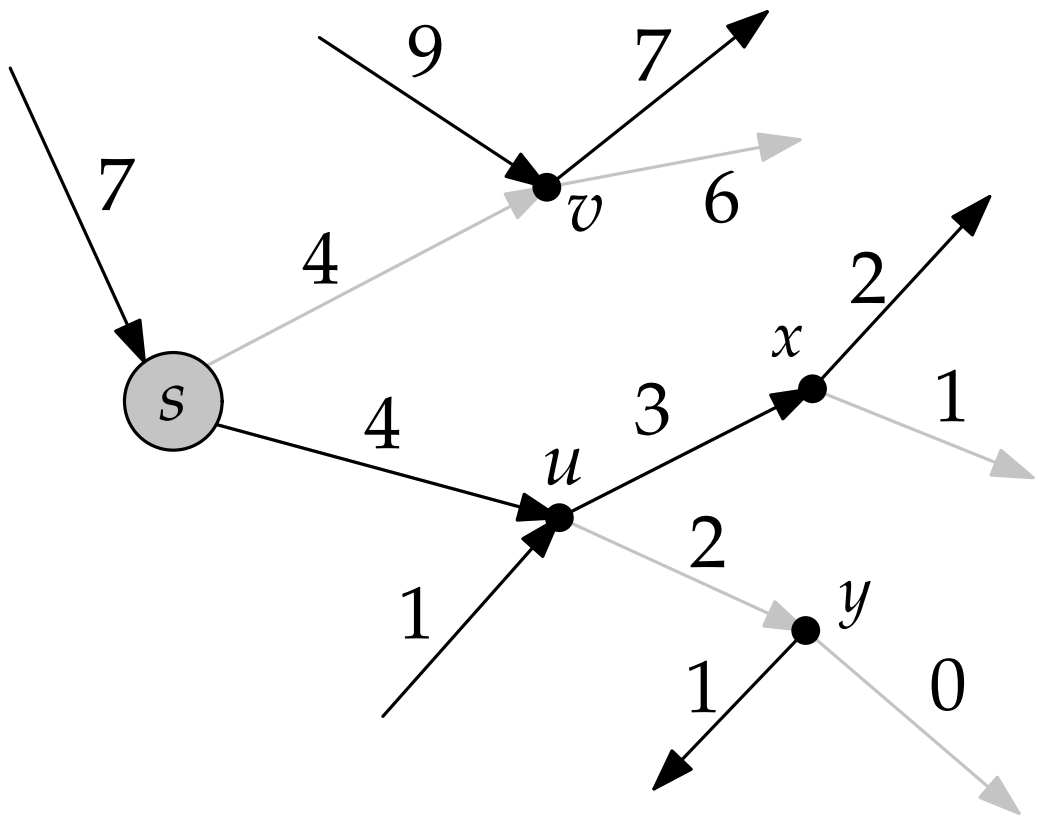


ARRIVAL

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!

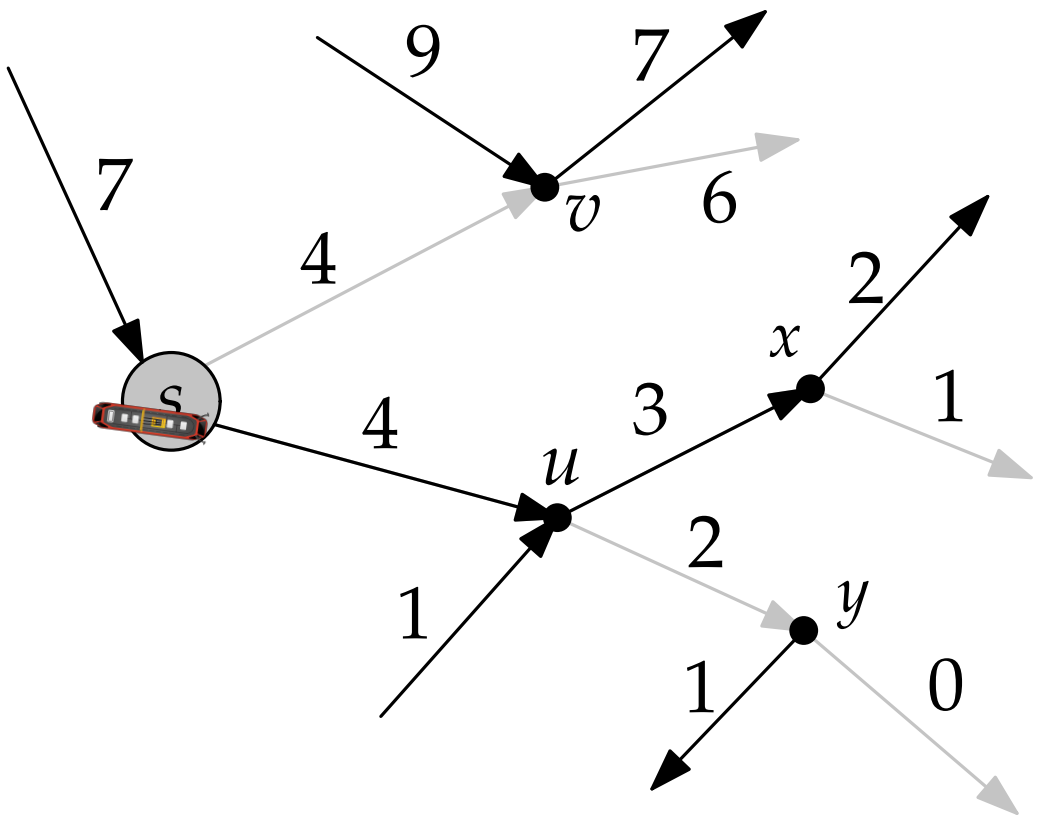
train flow from s to t



Arrival

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!

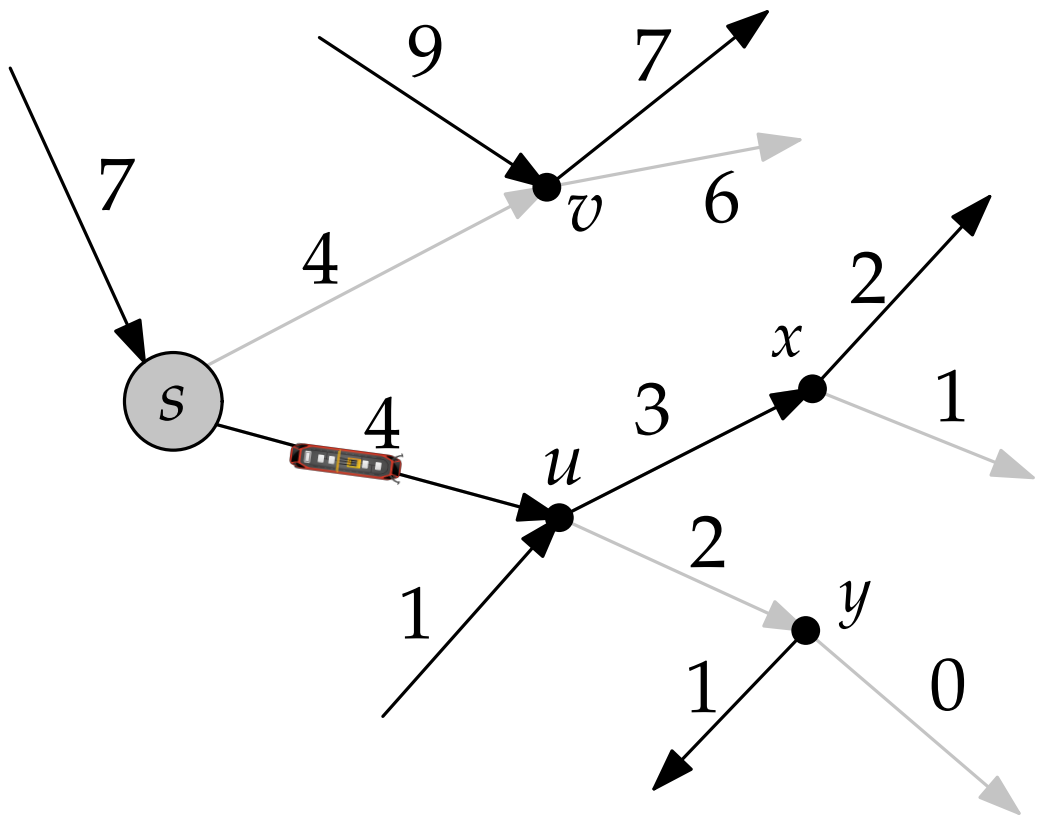


Arrival

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!

train flow from s to t

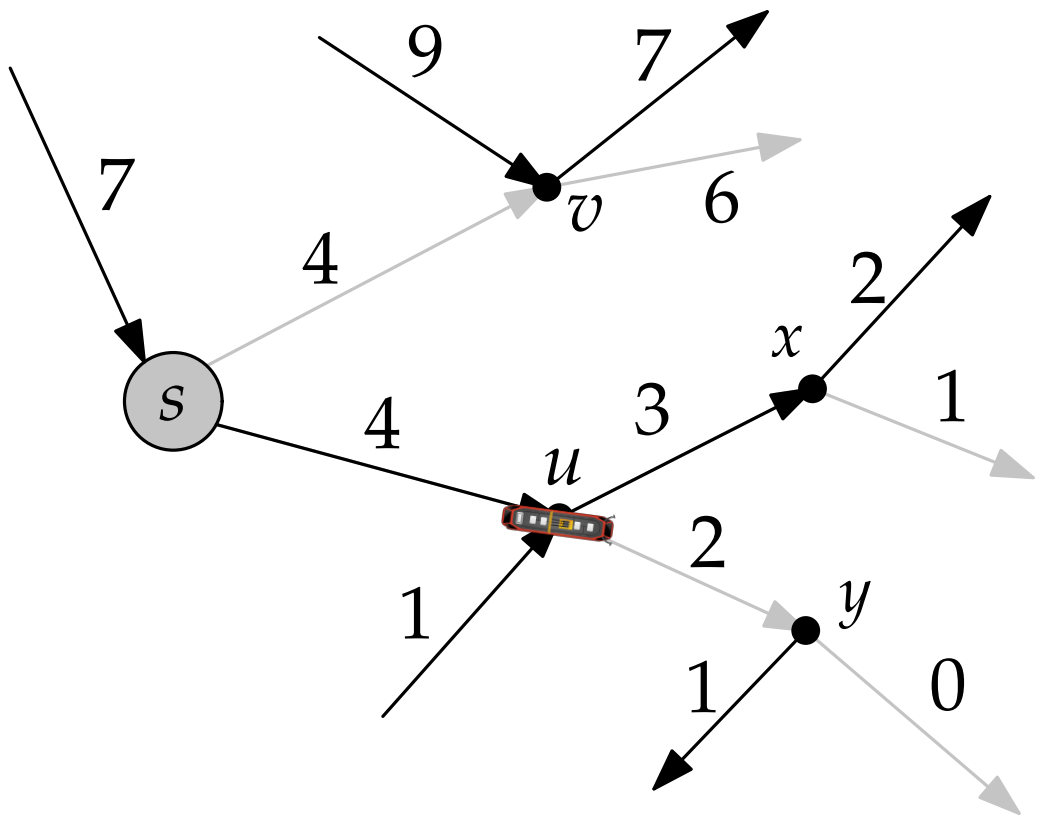


ARRIVAL

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!

train flow from s to t

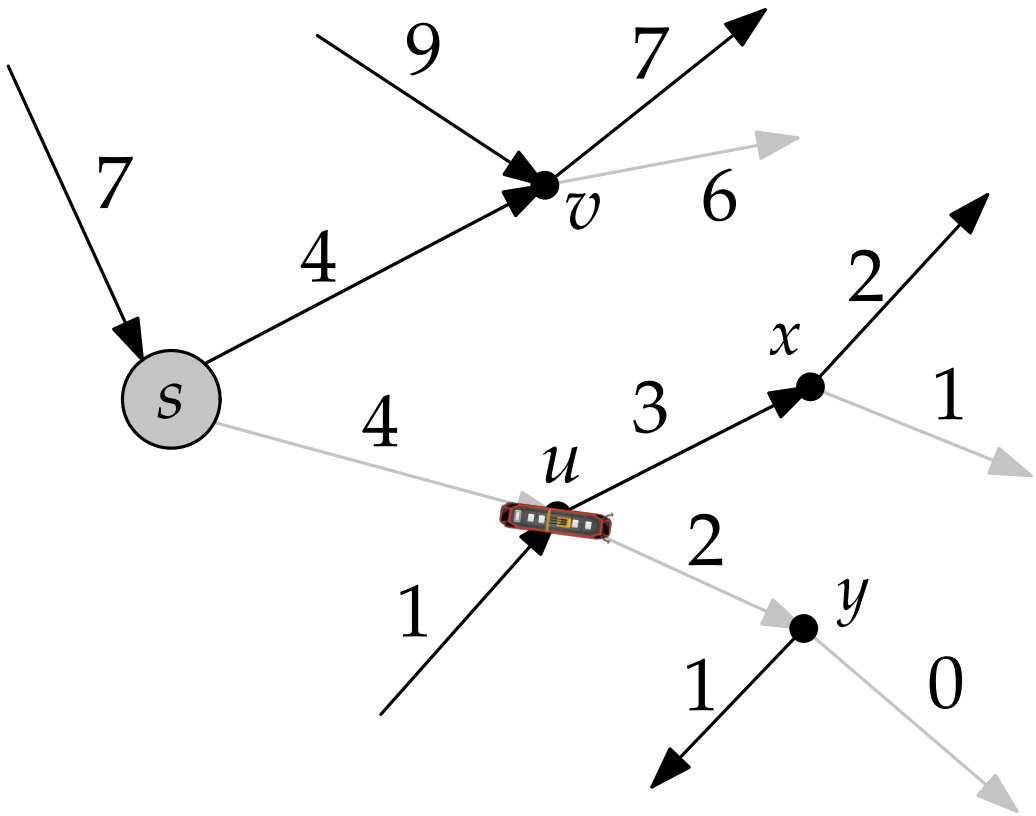


ARRIVAL

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!

train flow from s to t

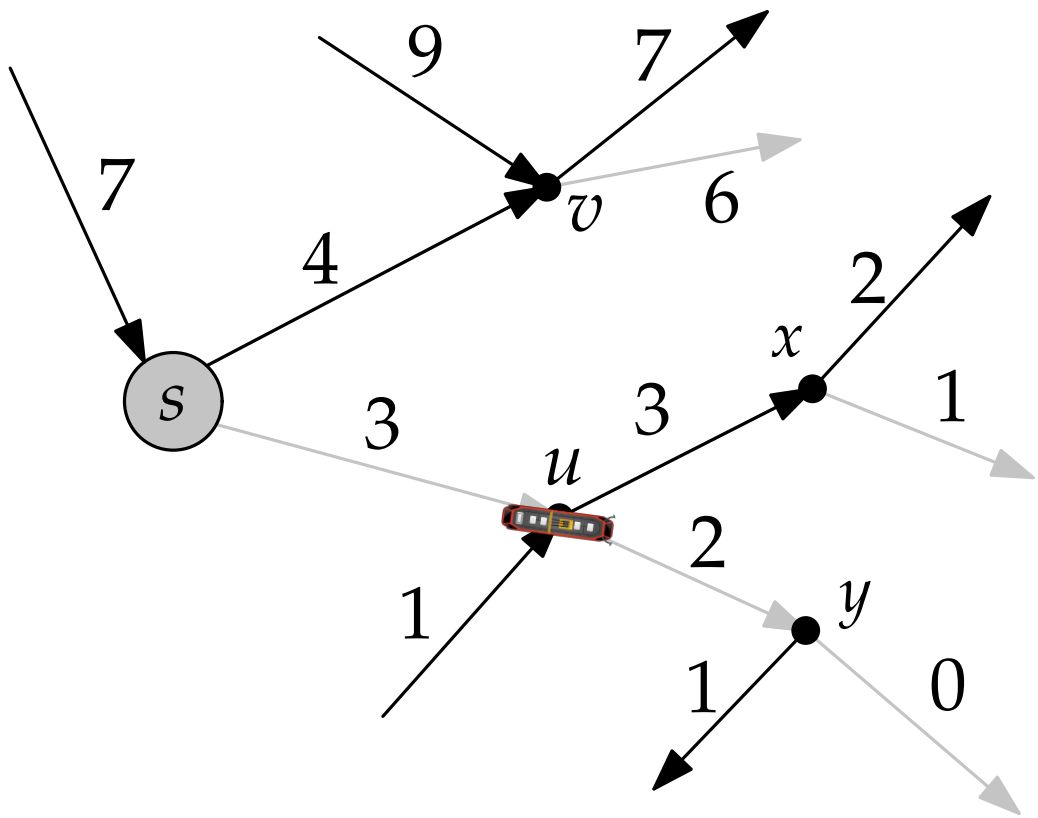


ARRIVAL

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!

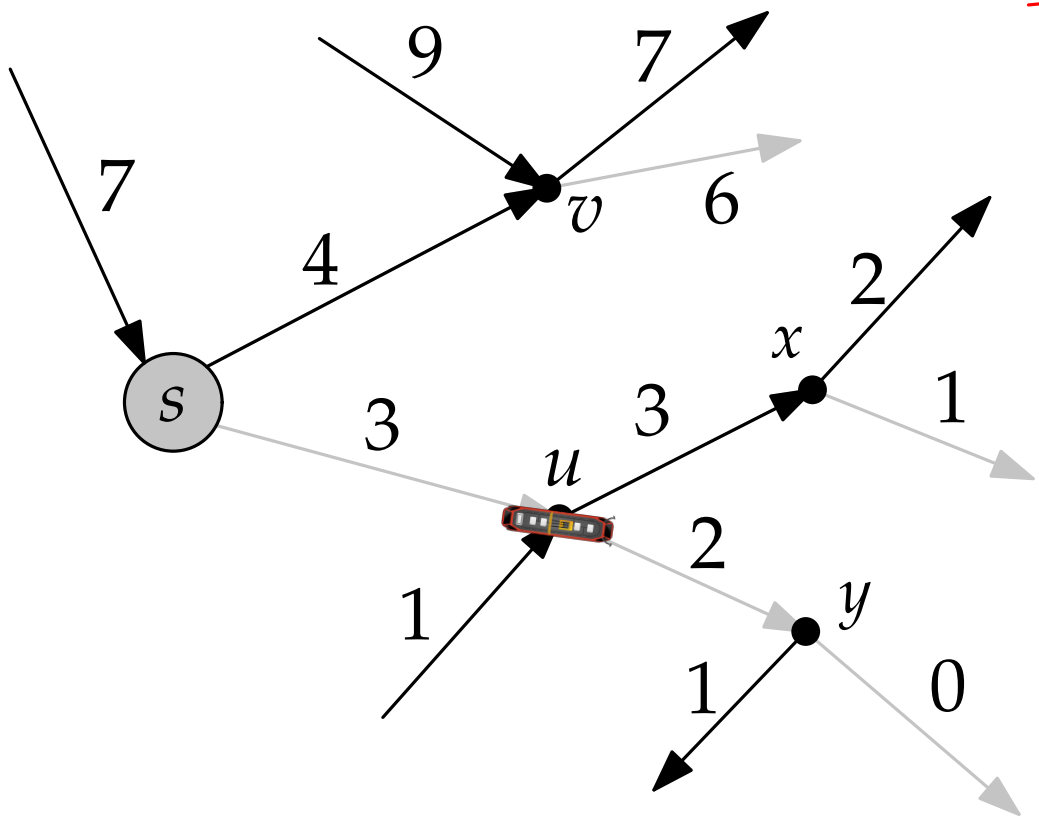
train flow from s to t



Arrival

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!



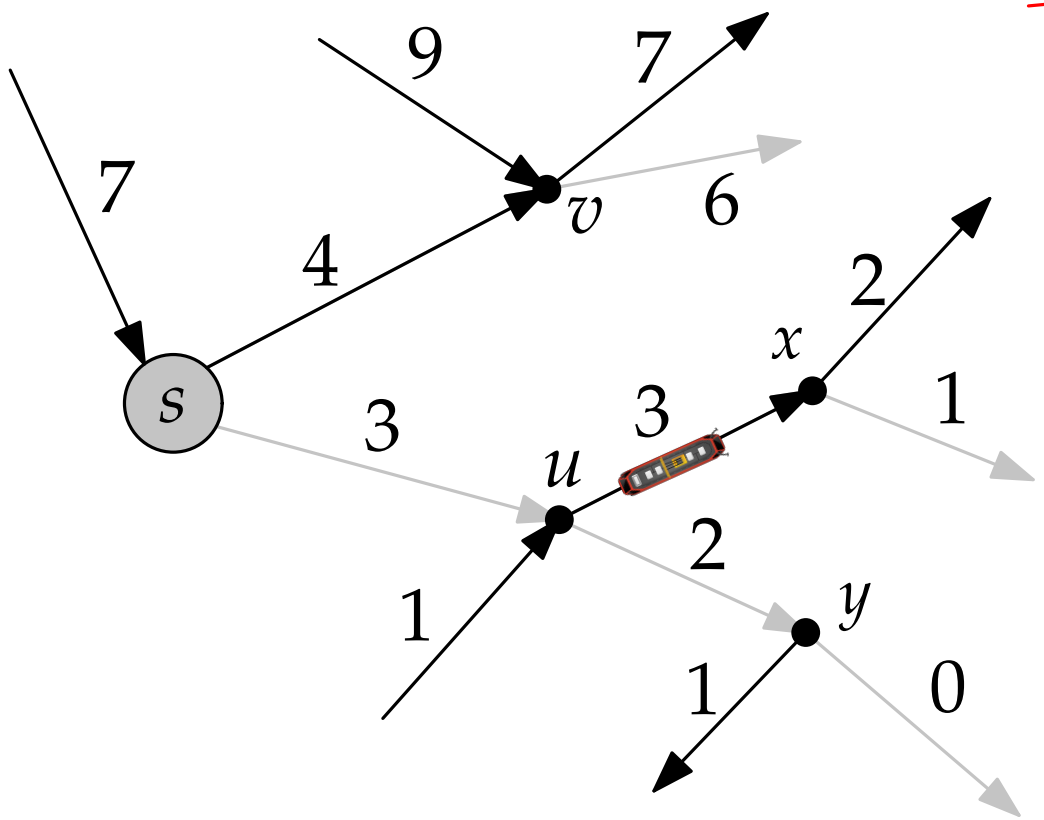
~~train flow from s to t~~

train flow from u to t

Arrival

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!



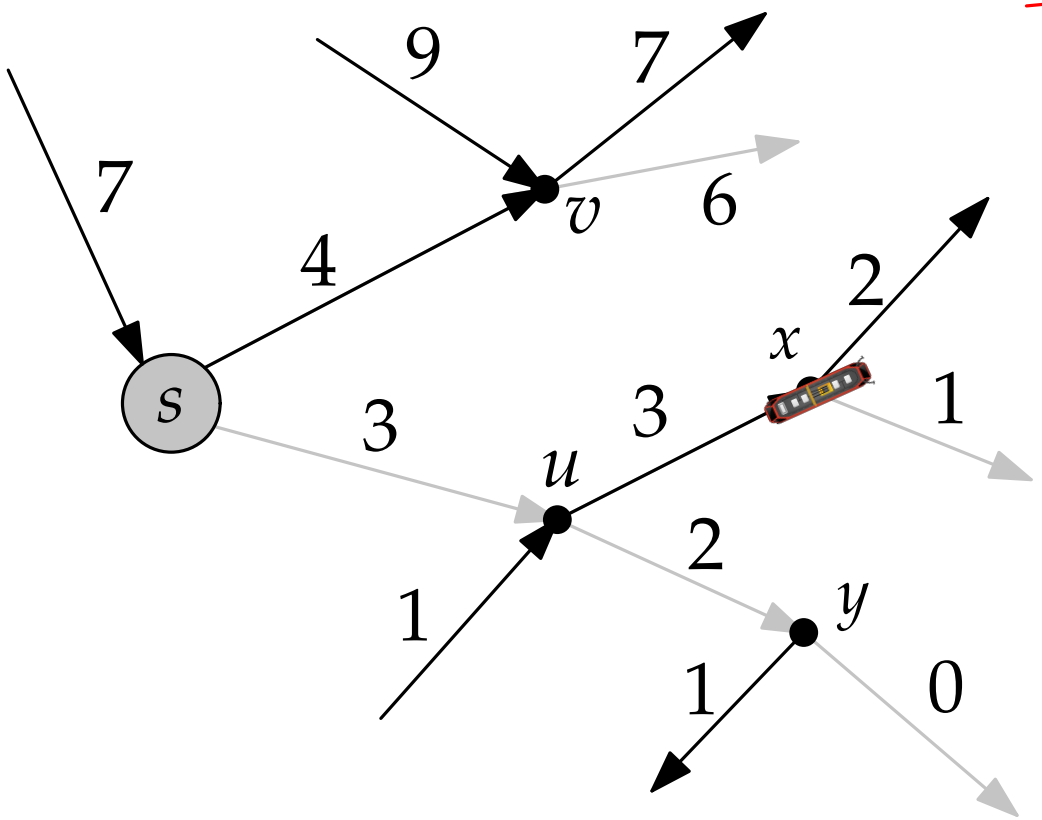
~~train flow from s to t~~

train flow from u to t

ARRIVAL

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!



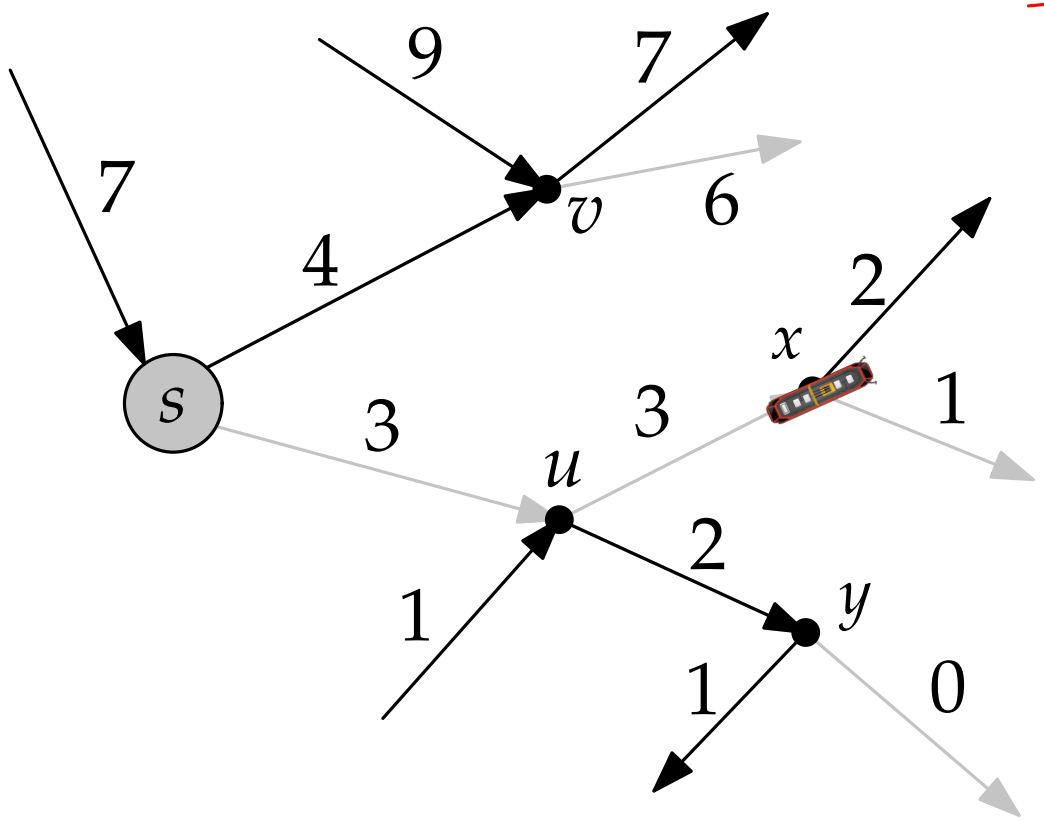
~~train flow from s to t~~

train flow from u to t

Arrival

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!



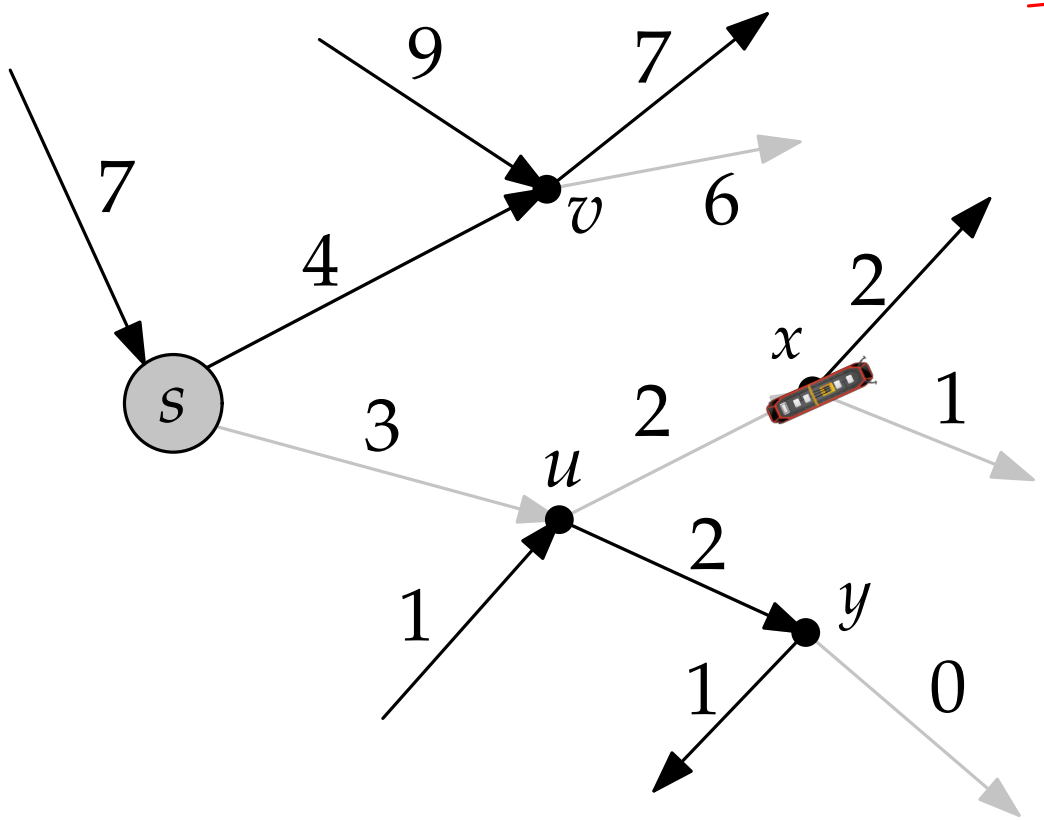
~~train flow from s to t~~

train flow from u to t

Arrival

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!



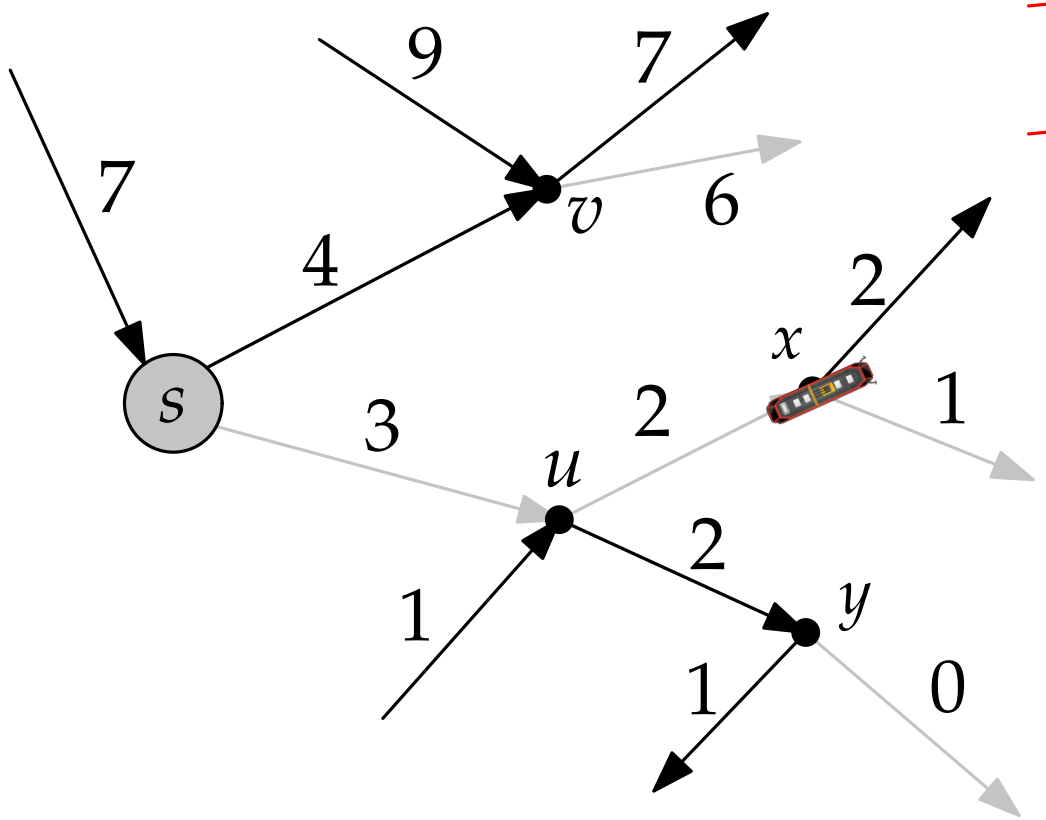
~~train flow from s to t~~

train flow from u to t

Arrival

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!



~~train flow from s to t~~

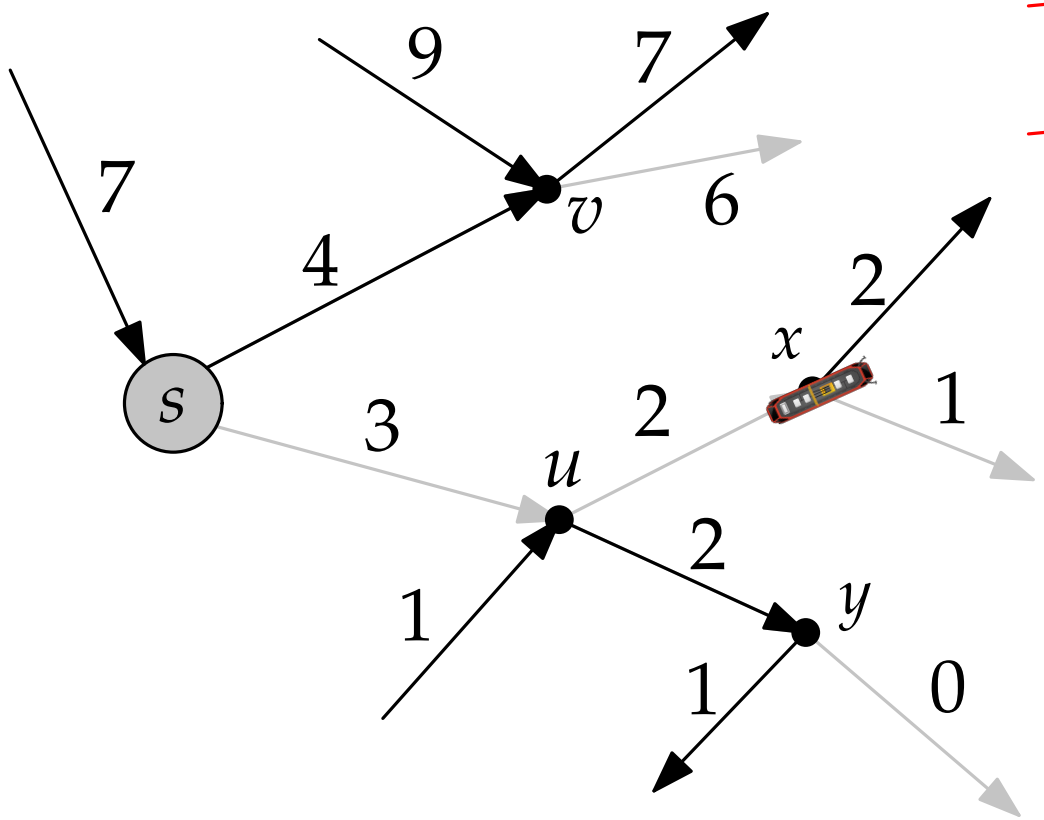
~~train flow from u to t~~

train flow from x to t

Arrival

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!



~~train flow from s to t~~

~~train flow from u to t~~

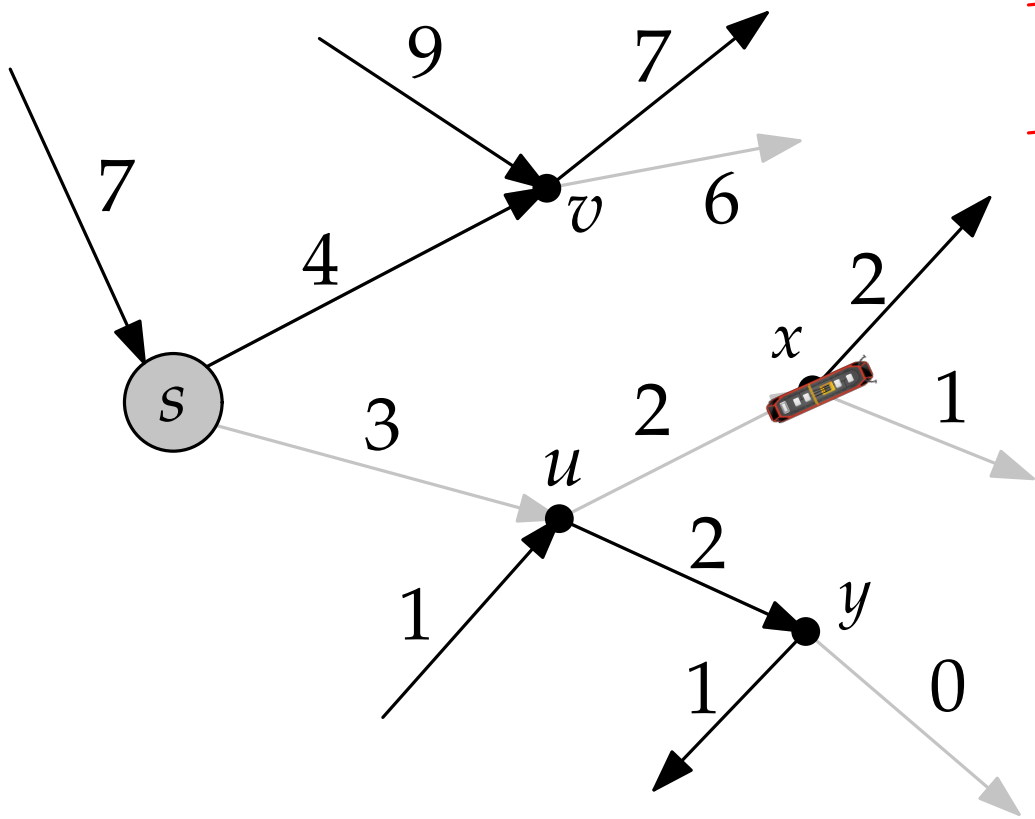
train flow from x to t

numbers decrease

Arrival

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!



~~train flow from s to t~~

~~train flow from u to t~~

train flow from x to t

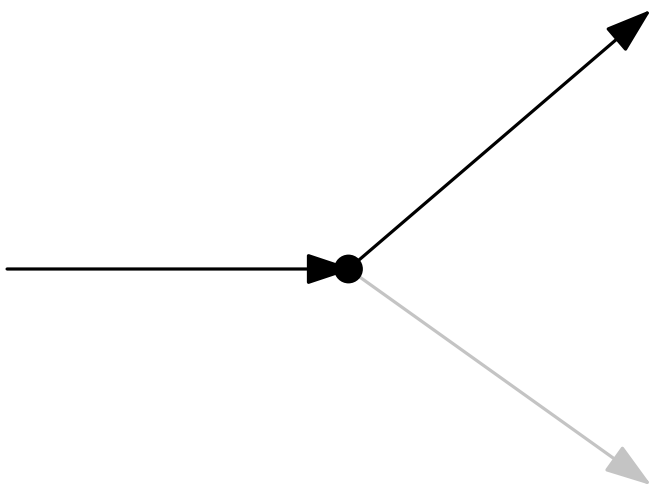
numbers decrease

never become negative

Arrival

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!



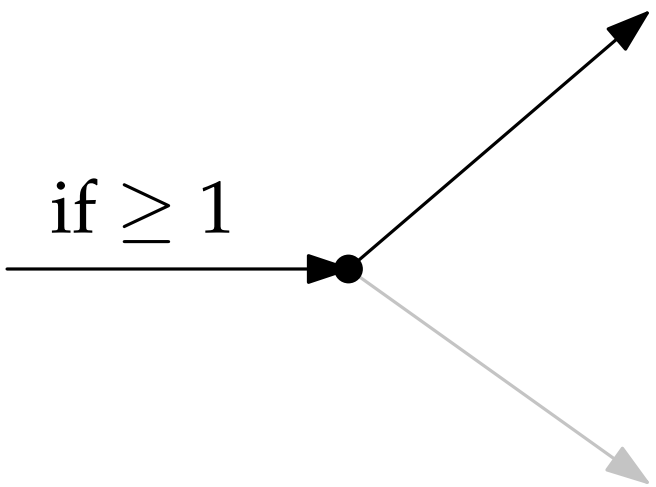
numbers decrease

never become negative

ARRIVAL

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!



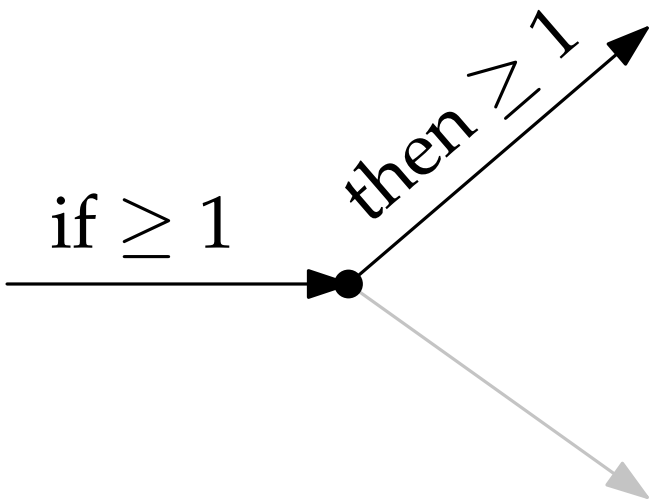
numbers decrease

never become negative

ARRIVAL

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Proof. Just let the train ride!



numbers decrease

never become negative

ARRIVAL

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl). ARRIVAL is in NP.

ARRIVAL

Lemma (Dohrau, Gärtner, Kohler, Matoušek, Welzl). If a train flow f exists then the train reaches the destination t.

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl). ARRIVAL is in NP.

Proof. Given $f : E \to \mathbb{N}$, verify that it is a train flow from s to t.
If yes, then f is a certificate that the train reaches the station.
Thus, ARRIVAL is in NP.

ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in co-NP.

ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in co-NP.

A vertex u is a *trap* if G contains no path from u to t.

ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in co-NP.

A vertex u is a *trap* if G contains no path from u to t.

Observation. If train reaches a trap, it will never reach the destination t.

ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in co-NP.

A vertex u is a *trap* if G contains no path from u to t.

Observation. If train reaches a trap, it will never reach the destination t.

Less obvious Observation. If the train does not eventually reach t, then it will reach a trap.

ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in co-NP.

A vertex u is a *trap* if G contains no path from u to t.

Observation. If train reaches a trap, it will never reach the destination t.

Less obvious Observation. If the train does not eventually reach t, then it will reach a trap.

A train flow from s to a trap is a certificate that the train does not reach destination t.

ARRIVAL

Theorem (Dohrau, Gärtner, Kohler, Matoušek, Welzl).
ARRIVAL is in co-NP.

A vertex u is a *trap* if G contains no path from u to t.

Observation. If train reaches a trap, it will never reach the destination t.

Less obvious Observation. If the train does not eventually reach t, then it will reach a trap.

A train flow from s to a trap is a certificate that the train does not reach destination t.
So ARRIVAL is in co-NP.

ARRIVAL

Theorem (Karthik C.S.). ARRIVAL is in PLS.

ARRIVAL

Theorem (Karthik C.S.). ARRIVAL is in PLS. That is, computing where the train ends up (destination t or some trap) can be phrased as a quest for finding a local minimum of a certain function.

ARRIVAL

Theorem (Karthik C.S.). ARRIVAL is in PLS. That is, computing where the train ends up (destination t or some trap) can be phrased as a quest for finding a local ~~minimum~~ maximum of a certain function.

ARRIVAL

Theorem (Karthik C.S.). ARRIVAL is in PLS. That is, computing where the train ends up (destination t or some trap) can be phrased as a quest for finding a local ~~minimum~~ maximum of a certain function.

$\mathcal{F}$: set of configurations

ARRIVAL

Theorem (Karthik C.S.). ARRIVAL is in PLS. That is, computing where the train ends up (destination t or some trap) can be phrased as a quest for finding a local ~~minimum~~ maximum of a certain function.

$\mathcal{F}$: set of configurations

$\text{val} : \mathcal{F} \to \mathbb{N}$

ARRIVAL

Theorem (Karthik C.S.). ARRIVAL is in PLS. That is, computing where the train ends up (destination t or some trap) can be phrased as a quest for finding a local ~~minimum~~ maximum of a certain function.

$\mathcal{F}$: set of configurations

$\text{val} : \mathcal{F} \to \mathbb{N}$

$\text{succ} \subseteq \mathcal{F} \times \mathcal{F}$

ARRIVAL

Theorem (Karthik C.S.). ARRIVAL is in PLS. That is, computing where the train ends up (destination t or some trap) can be phrased as a quest for finding a local ~~minimum~~ maximum of a certain function.

$\mathcal{F}$: set of configurations

$\text{val} : \mathcal{F} \to \mathbb{N}$

$\text{succ} \subseteq \mathcal{F} \times \mathcal{F}$

Goal. Find a *local maximum*. That is, a configuration $f \in \mathcal{F}$ with $\text{val}(g) \leq \text{val}(f)$ for all successors g of f.

ARRIVAL

Theorem (Karthik C.S.). ARRIVAL is in PLS. That is, computing where the train ends up (destination t or some trap) can be phrased as a quest for finding a local ~~minimum~~ maximum of a certain function.

$\mathcal{F}$: set of configurations, all train flows from s to some u

$\text{val} : \mathcal{F} \to \mathbb{N}$

$\text{succ} \subseteq \mathcal{F} \times \mathcal{F}$

Goal. Find a *local maximum*. That is, a configuration $f \in \mathcal{F}$ with $\text{val}(g) \leq \text{val}(f)$ for all successors g of f.

ARRIVAL

Theorem (Karthik C.S.). ARRIVAL is in PLS. That is, computing where the train ends up (destination t or some trap) can be phrased as a quest for finding a local ~~minimum~~ maximum of a certain function.

$\mathcal{F}$: set of configurations, all train flows from s to some u

$\text{val} : \mathcal{F} \to \mathbb{N}, \sum_{e \in E} f(e)$.

$\text{succ} \subseteq \mathcal{F} \times \mathcal{F}$

Goal. Find a *local maximum*. That is, a configuration $f \in \mathcal{F}$ with $\text{val}(g) \leq \text{val}(f)$ for all successors g of f.

ARRIVAL

Theorem (Karthik C.S.). ARRIVAL is in PLS. That is, computing where the train ends up (destination t or some trap) can be phrased as a quest for finding a local ~~minimum~~ maximum of a certain function.

$\mathcal{F}$: set of configurations, all train flows from s to some u

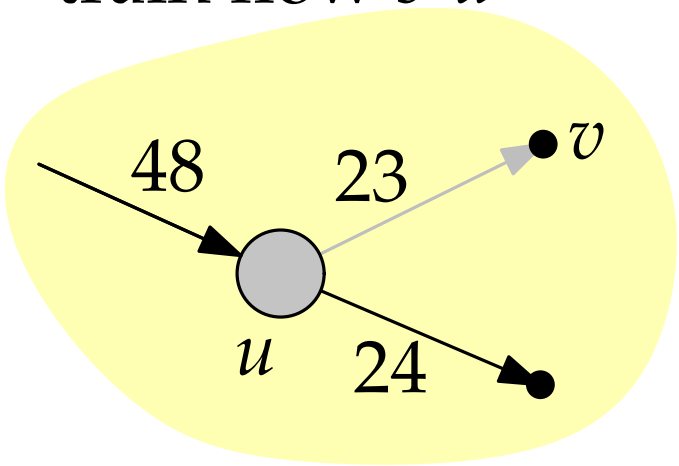
$\text{val} : \mathcal{F} \to \mathbb{N}$, $\sum_{e \in E} f(e)$.

$\text{succ} \subseteq \mathcal{F} \times \mathcal{F}$, what would happen next

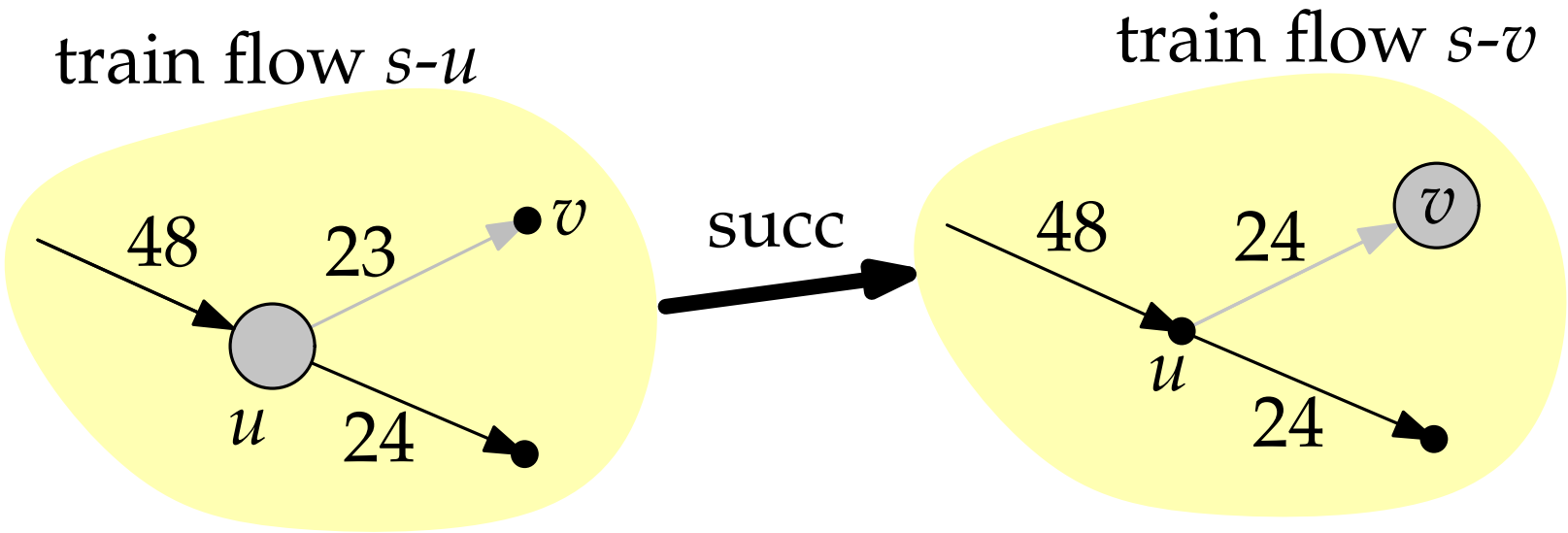
Goal. Find a *local maximum*. That is, a configuration $f \in \mathcal{F}$ with $\text{val}(g) \leq \text{val}(f)$ for all successors g of f.

Arrival

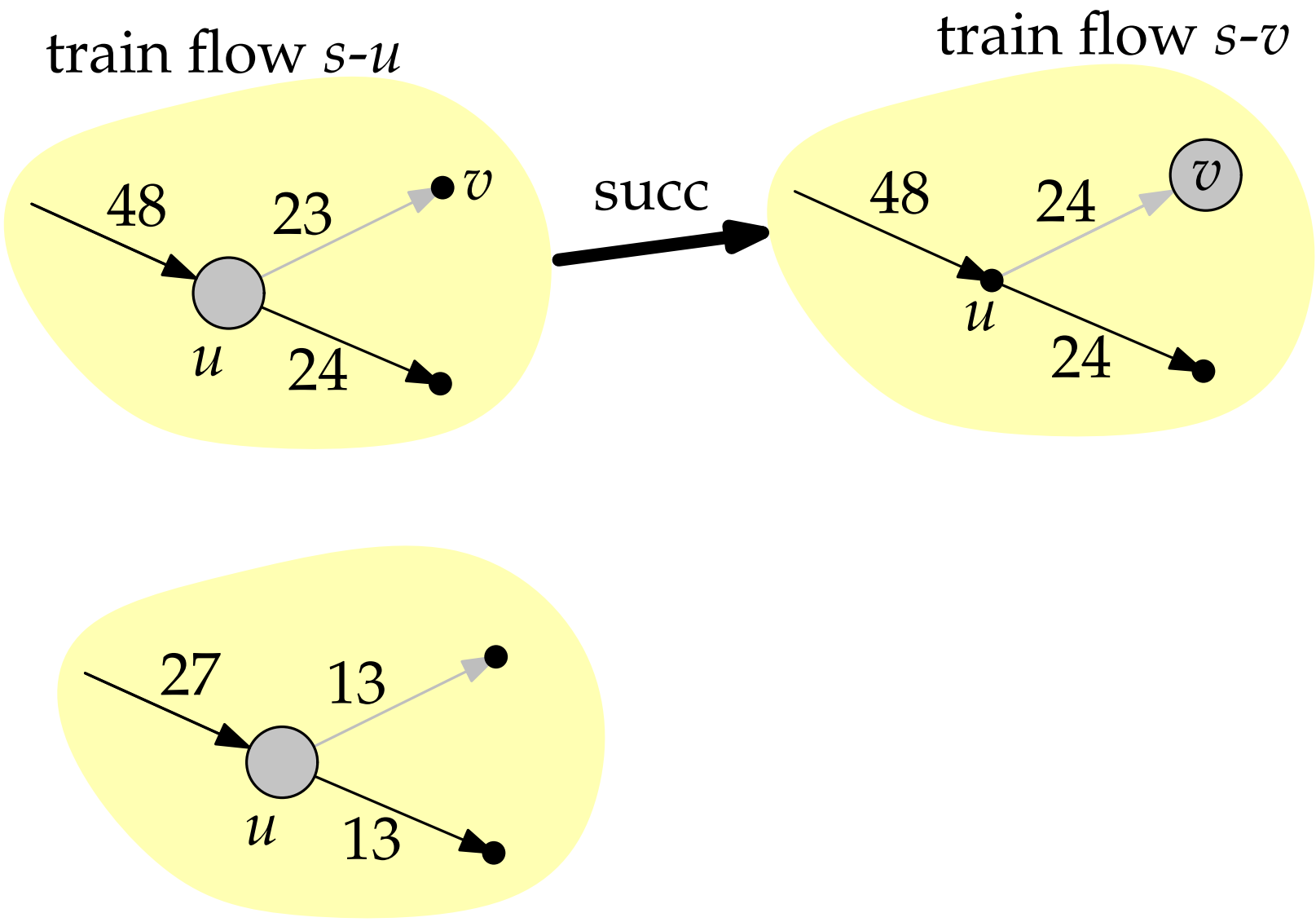
train flow s-u



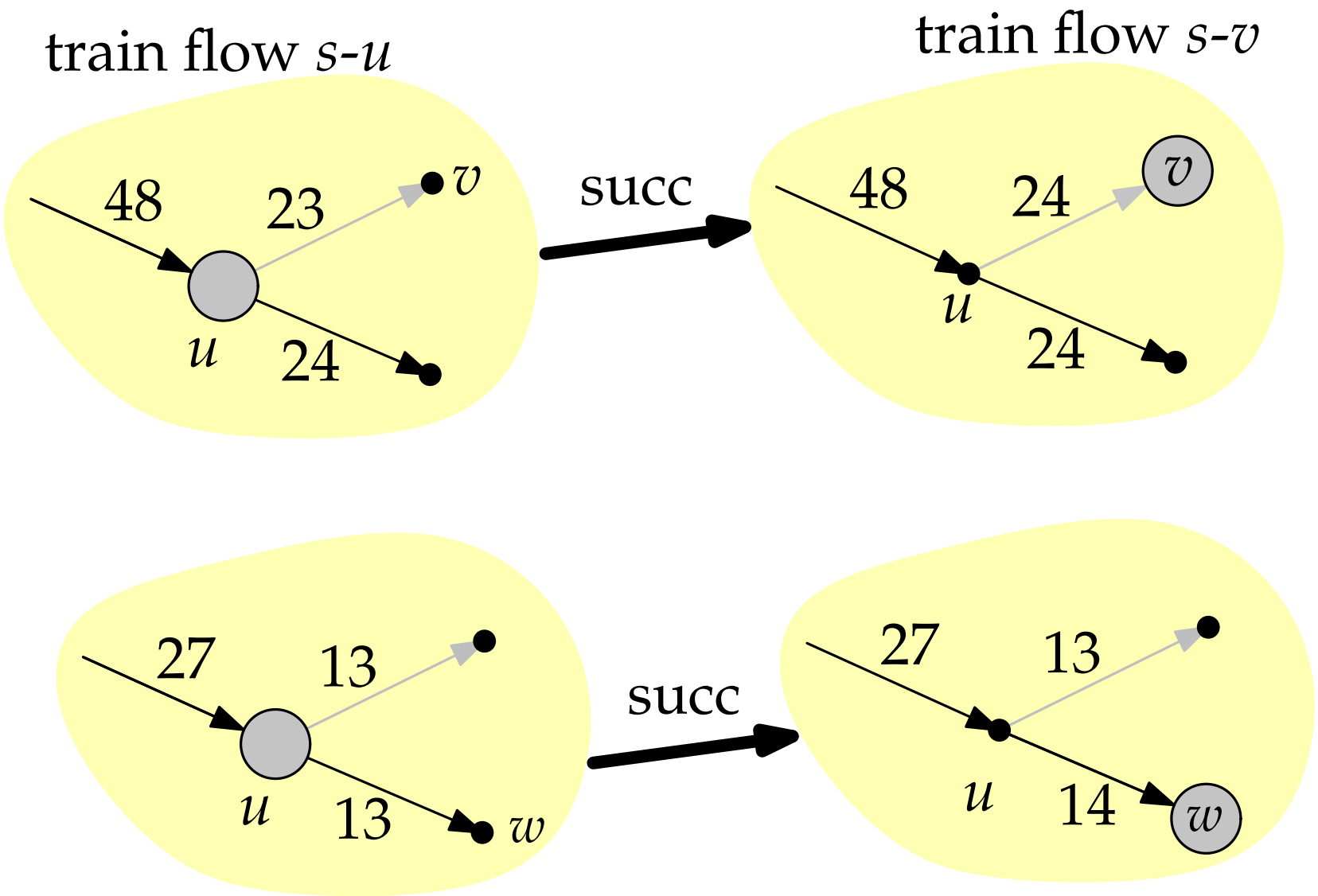
Arrival



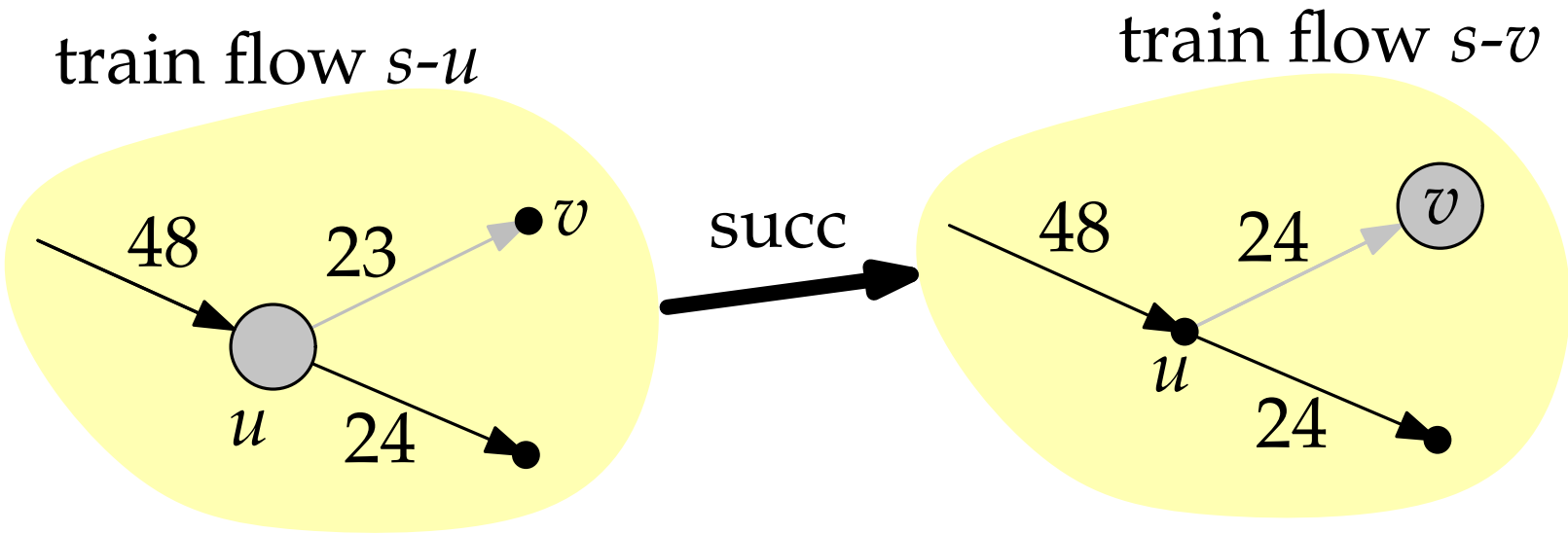
Arrival



Arrival

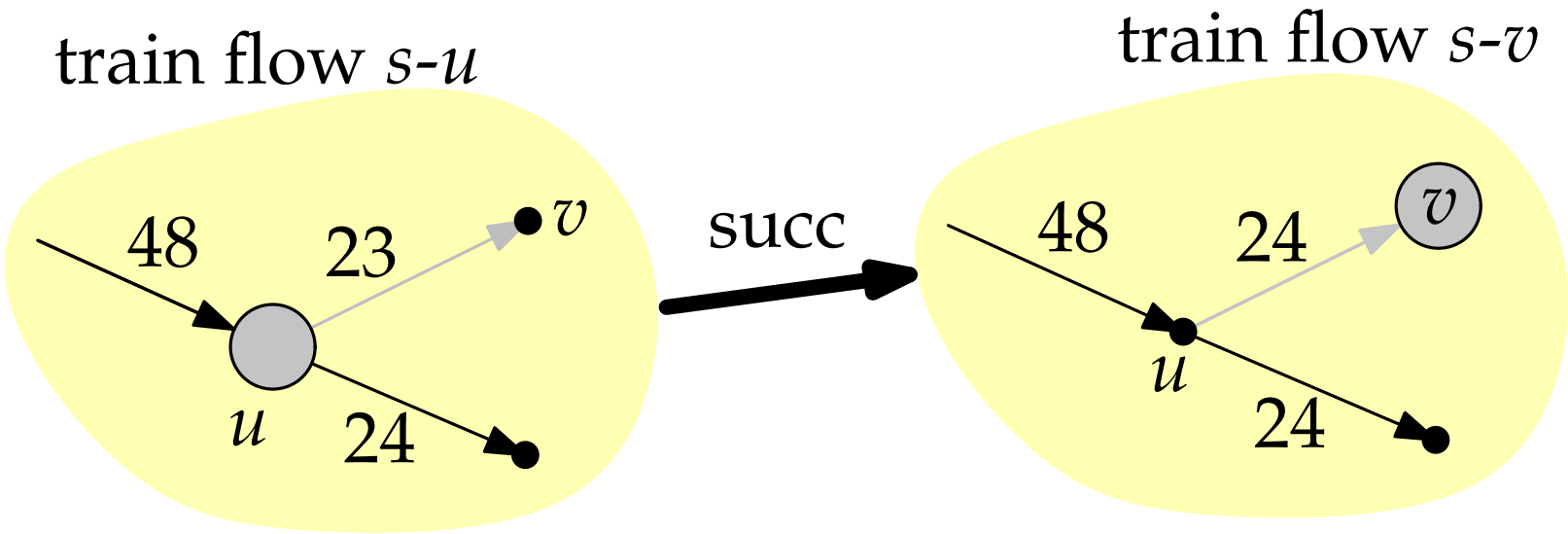

train flow s-u
48
23
v
u
24
succ
train flow s-v
48
24
v
u
24
27
13
u
13
w
succ
27
13
u
14
w

ARRIVAL



Exception: $\mathrm{succ}(f) := f$ when f is a train flow from s to t or to some trap.

ARRIVAL



Exception: $\mathrm{succ}(f) := f$ when f is a train flow from s to t or to some trap.

Theorem (Karthik C.S.). Deciding ARRIVAL is no harder than finding a local maximum of the succ function.
Thus, ARRIVAL is in PLS.

PLS—More Formal Definition

PLS—More Formal Definition

Informal. A PLS-problem asks us to compute a *solution* that is *locally optimal* with respect to some *neighborhood relation* and a *valuation / cost / revenue* function.

PLS—More Formal Definition

Informal. A PLS-problem asks us to compute a *solution* that is *locally optimal* with respect to some *neighborhood relation* and a *valuation / cost / revenue* function.

Congestion Games:

- Solutions: all strategy profiles
- Neighbors: when one player switches strategy: $\vec{s} \to \vec{s}'$
- Cost: total potential Φ

PLS—More Formal Definition

Informal. A PLS-problem asks us to compute a *solution* that is *locally optimal* with respect to some *neighborhood relation* and a *valuation / cost / revenue* function.

Congestion Games:

- Solutions: all strategy profiles
- Neighbors: when one player switches strategy: $\vec{s} \to \vec{s}'$
- Cost: total potential Φ

ARRIVAL

- Solutions: trains flows from s to some u
- Neighbor: successor flow
- Value: sum of numbers

PLS—More Formal Definition

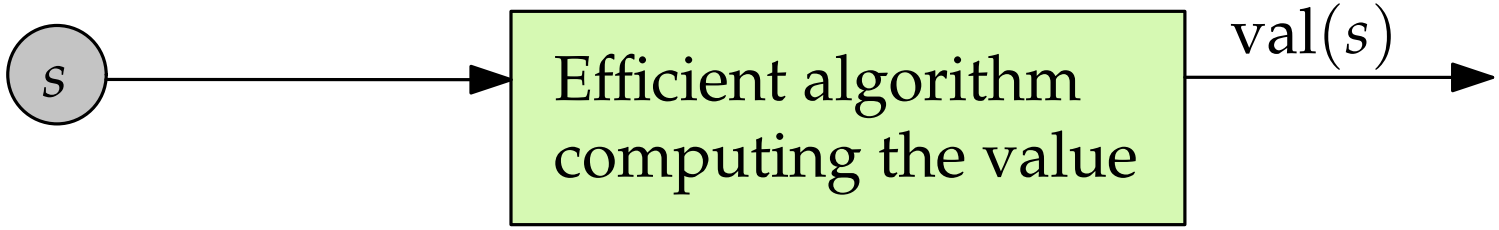
"The" generic PLS problem:

PLS—More Formal Definition

"The" generic PLS problem:

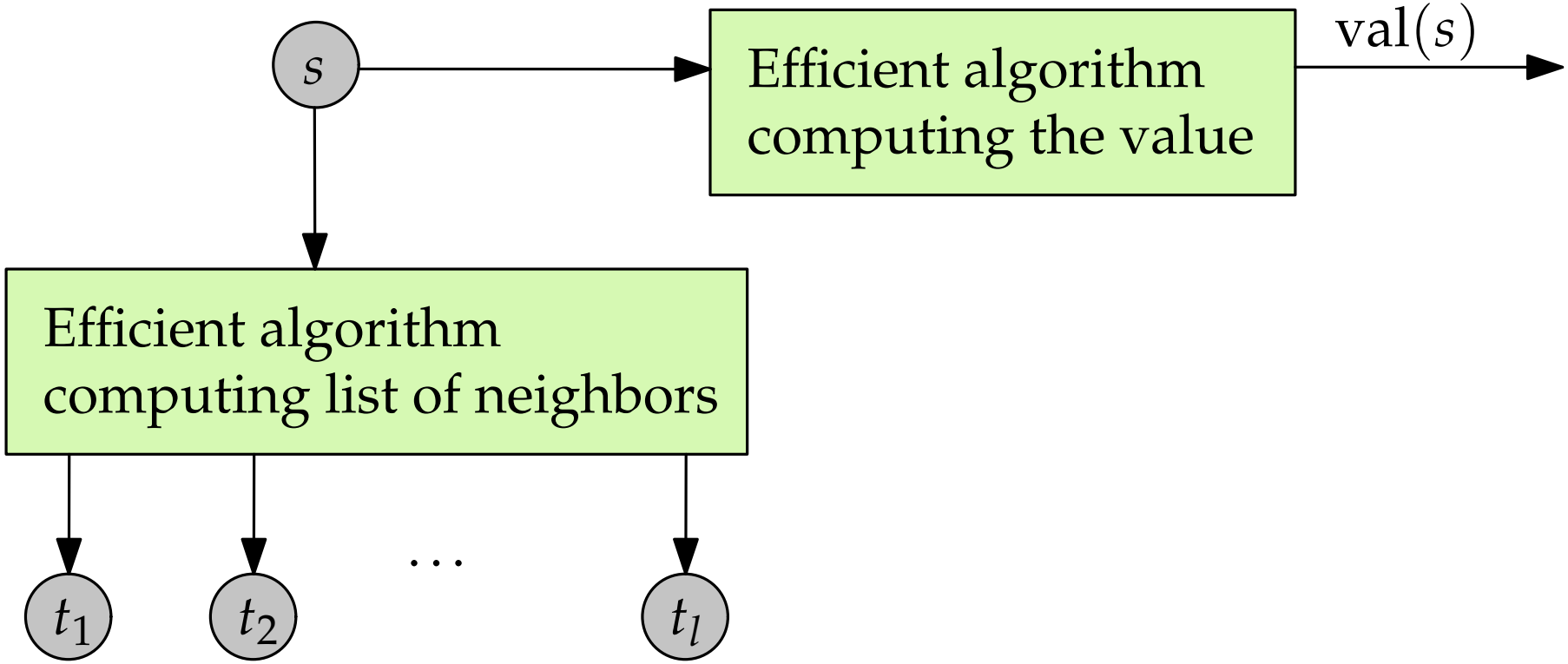
PLS—More Formal Definition

"The" generic PLS problem:



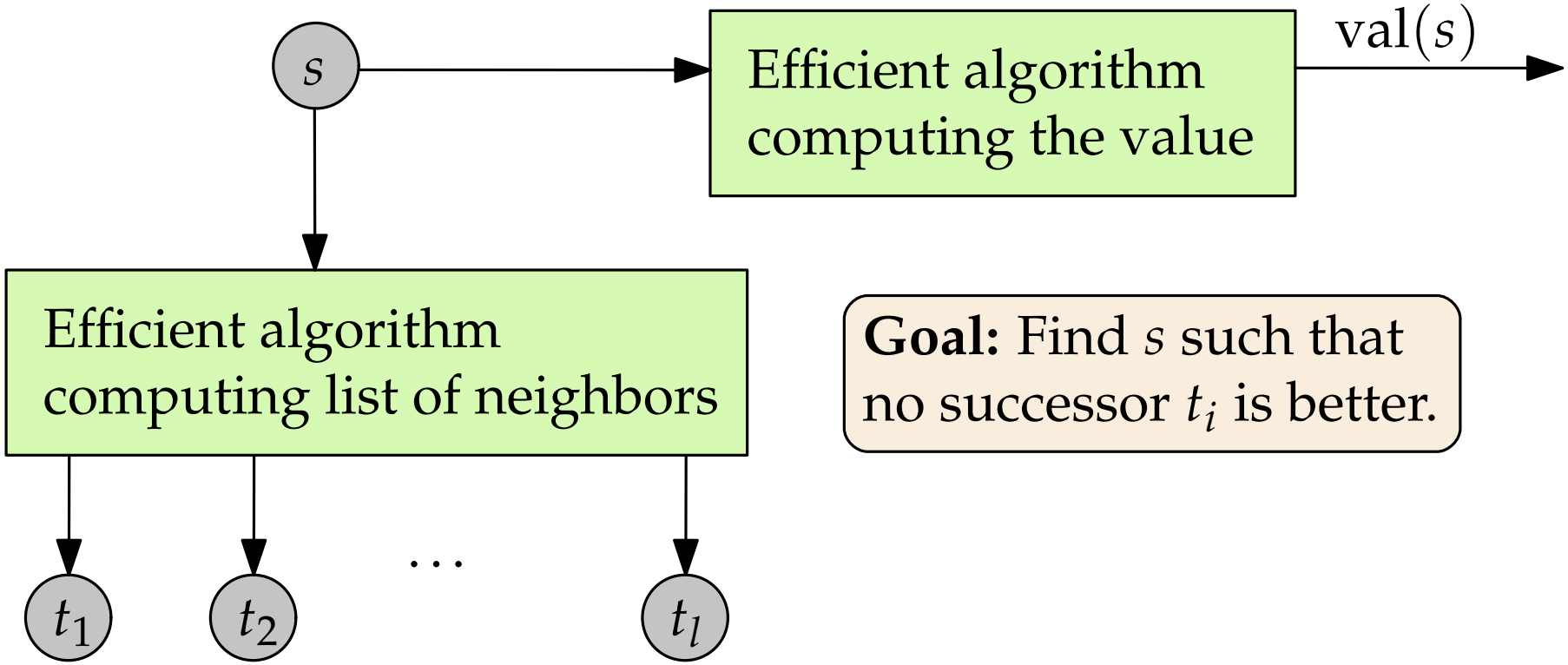
PLS—More Formal Definition

"The" generic PLS problem:



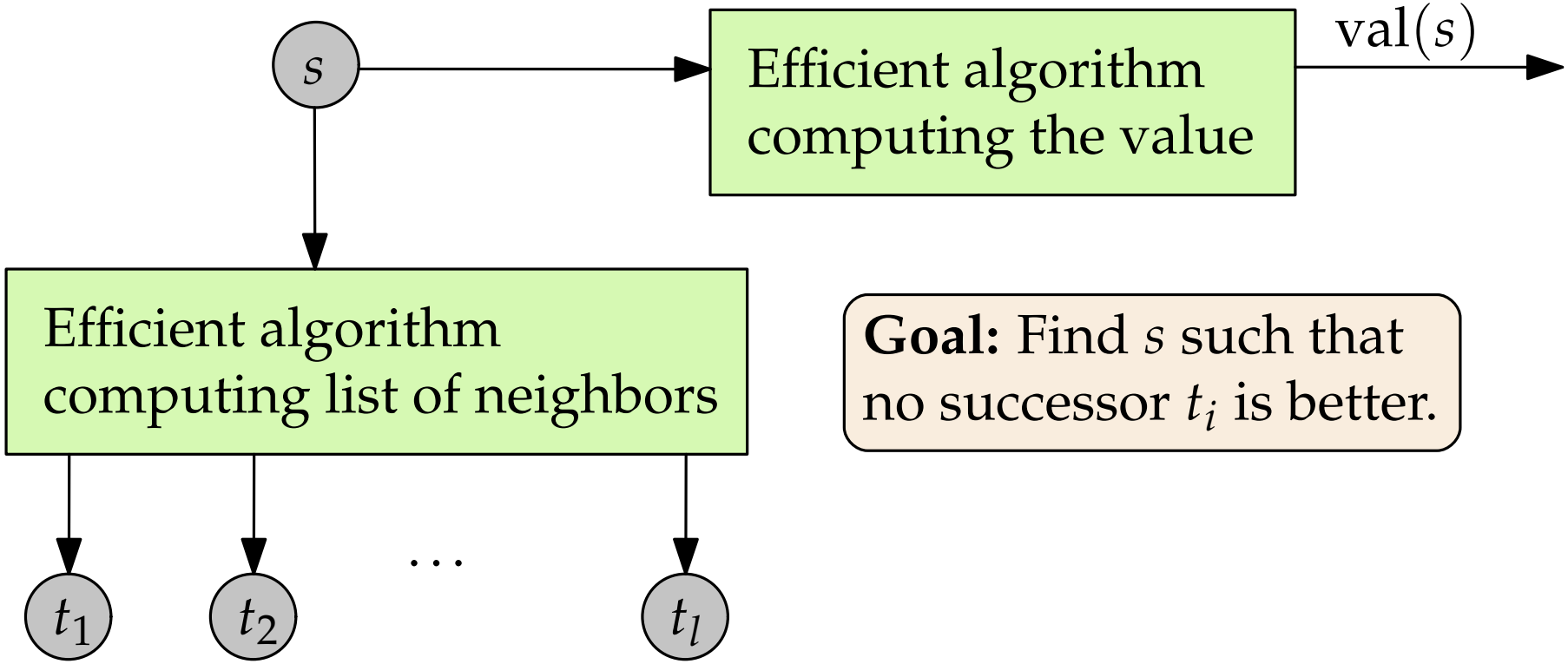
PLS—More Formal Definition

"The" generic PLS problem:



PLS—More Formal Definition

"The" generic PLS problem:

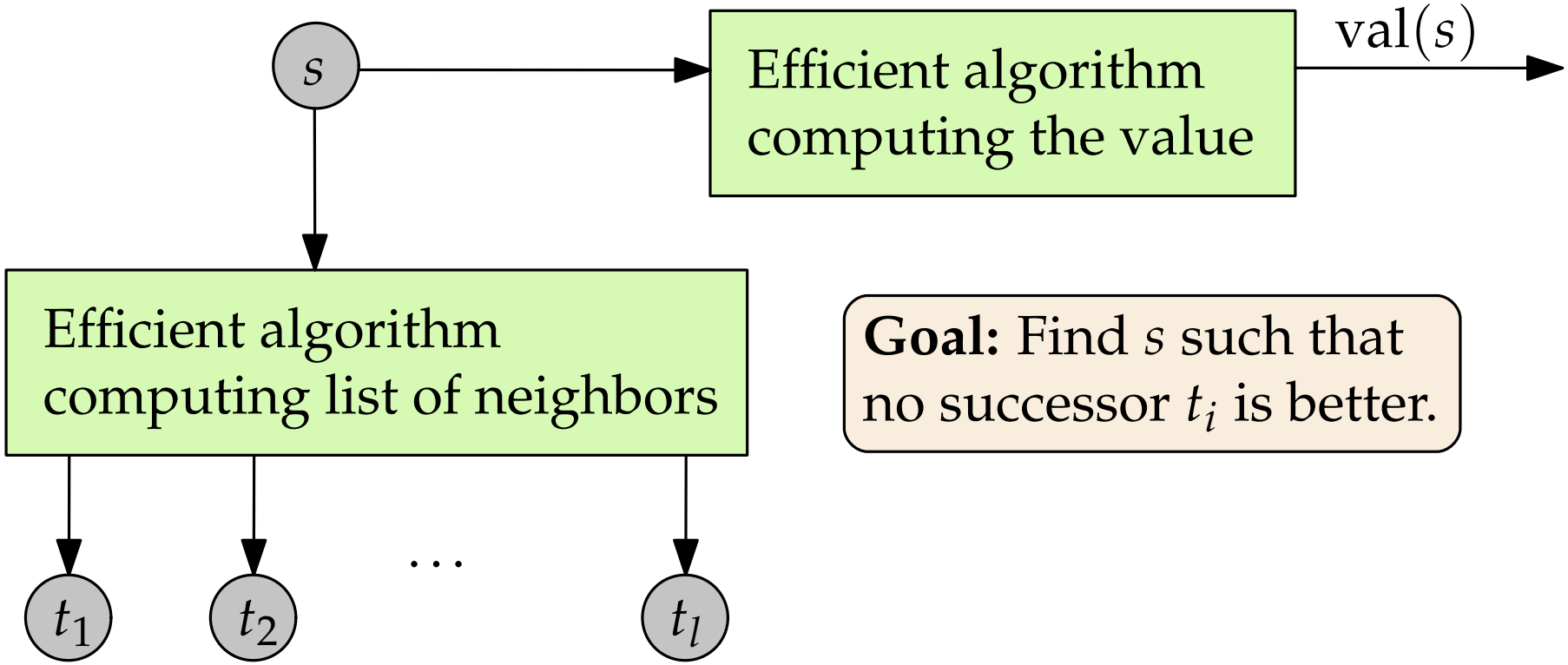


Goal: Find s such that no successor t_i is better.

l should be polynomial in $|s|$.

PLS—More Formal Definition

"The" generic PLS problem:

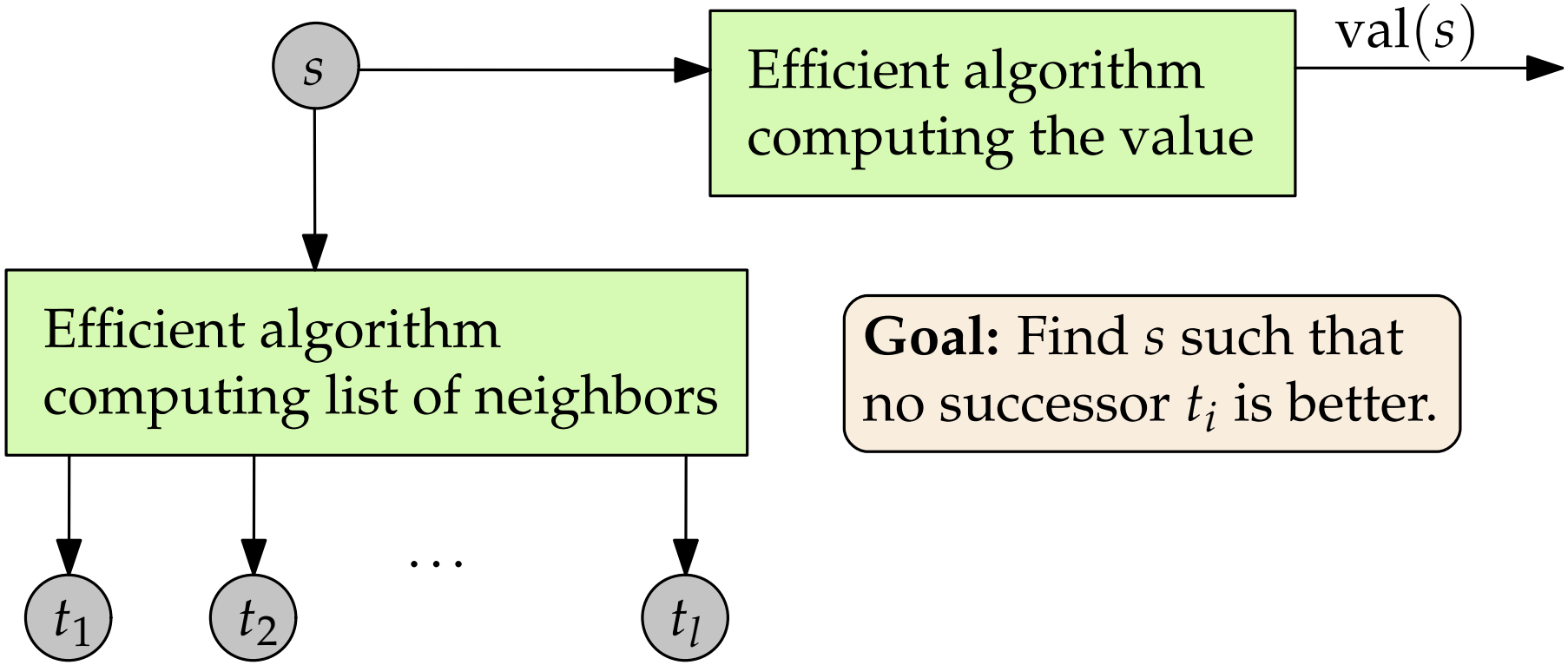


l should be polynomial in $|s|$.

$|t|$ polynomial in $|s|$ for every "reachable" t.

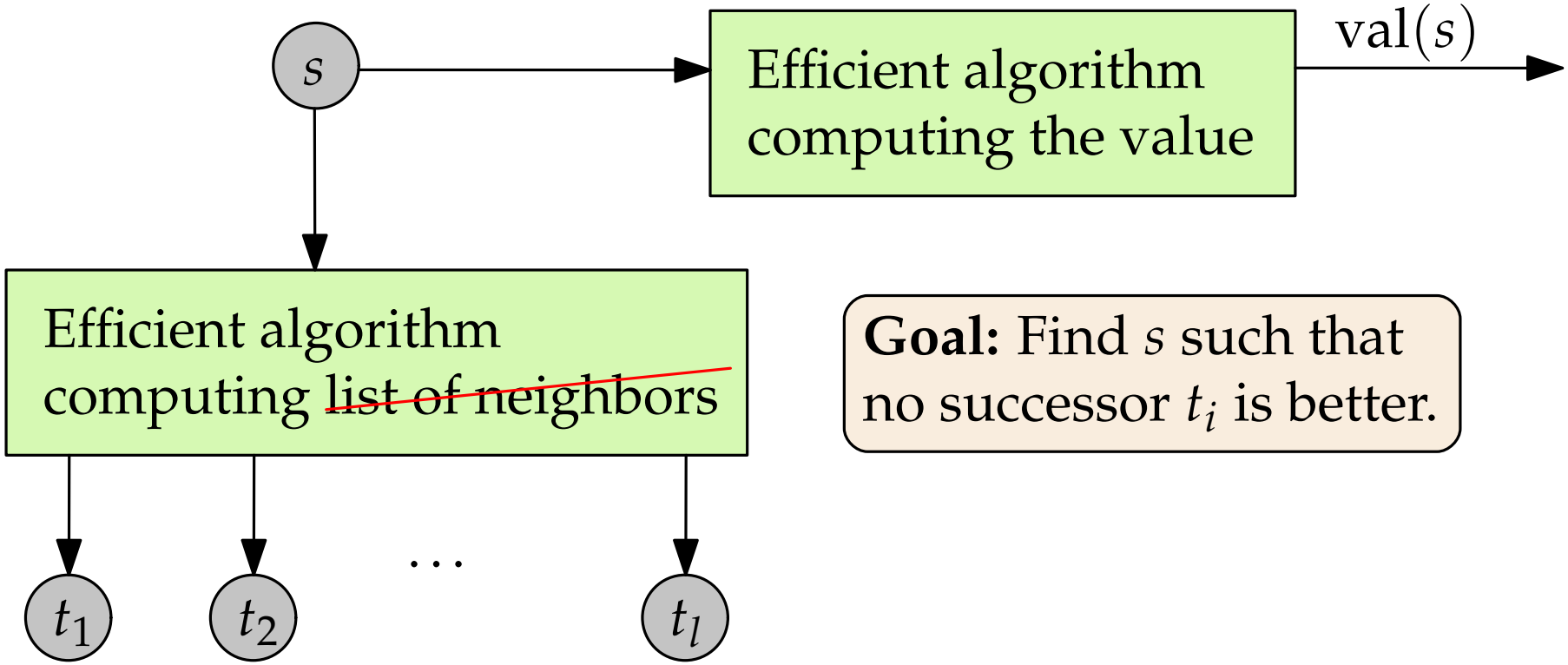
PLS—More Formal Definition

"The" generic PLS problem:



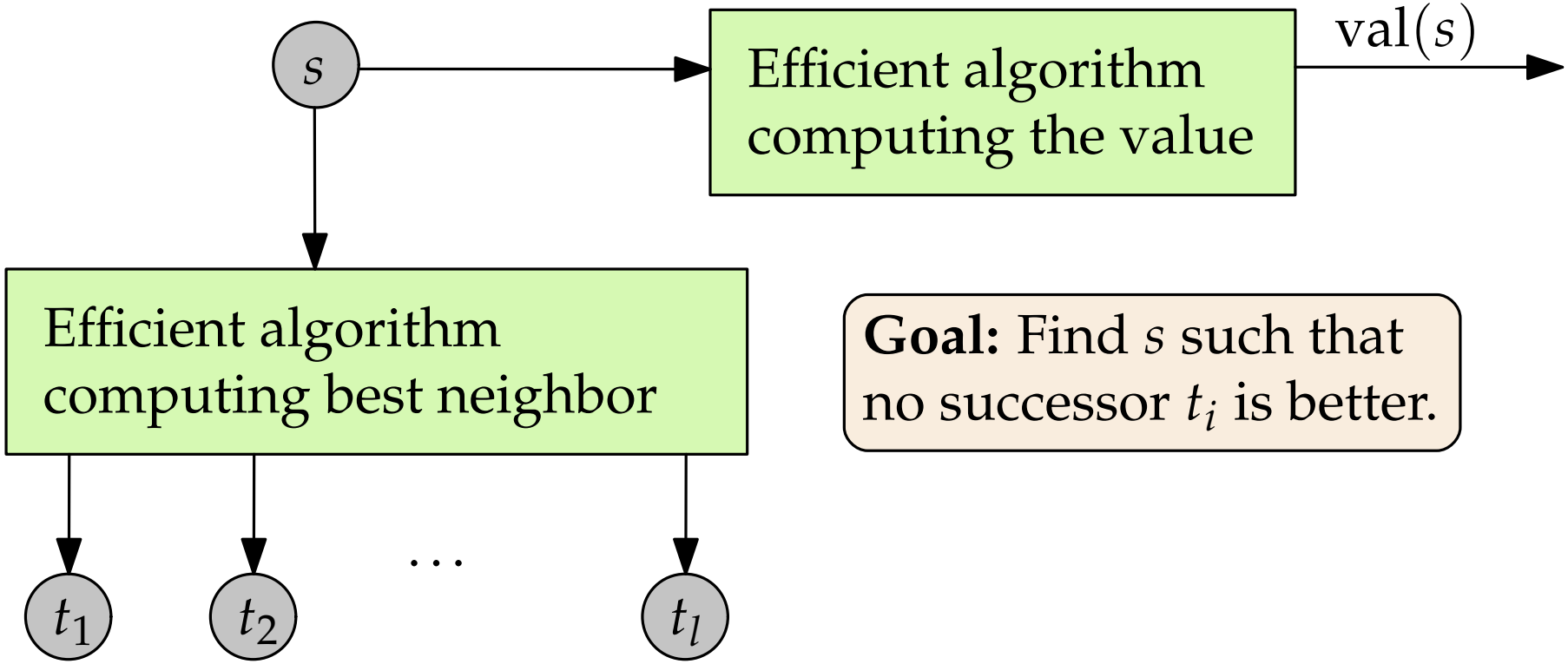
PLS—More Formal Definition

"The" generic PLS problem:



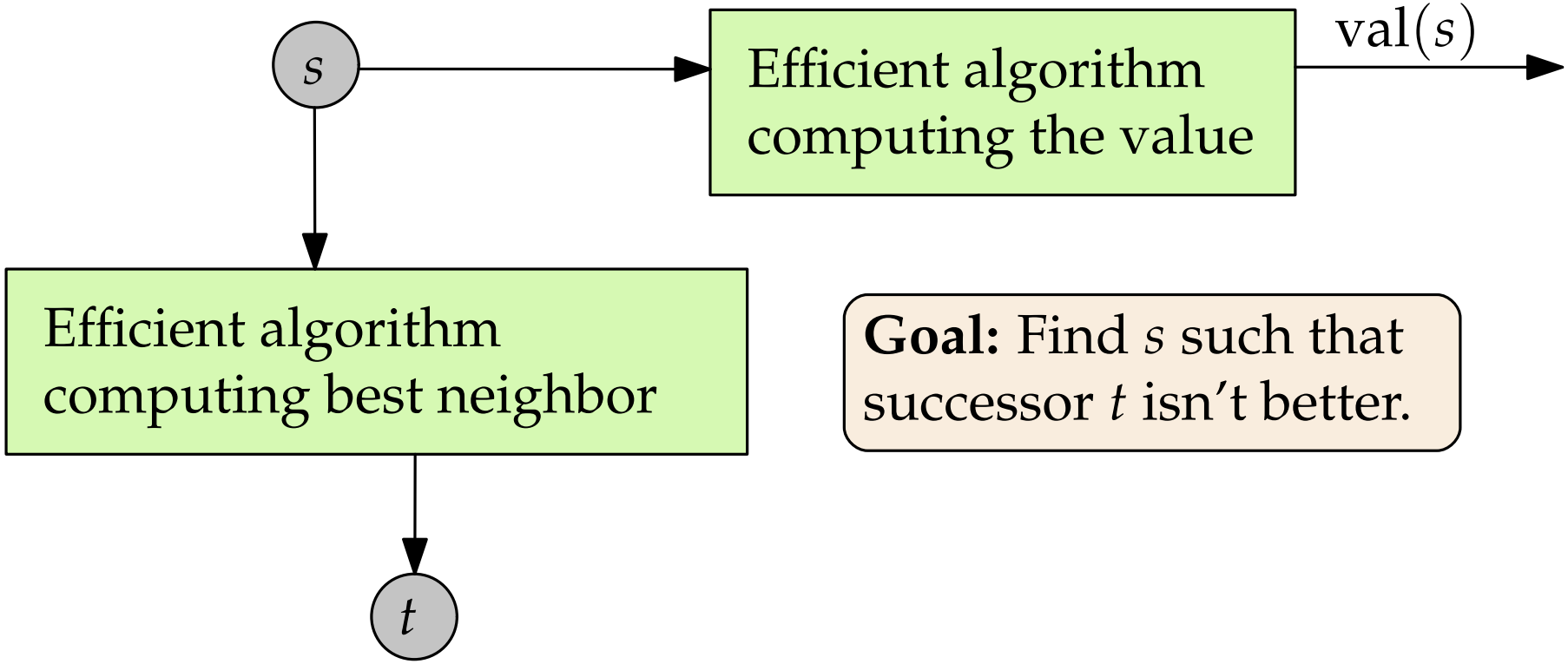
PLS—More Formal Definition

"The" generic PLS problem:



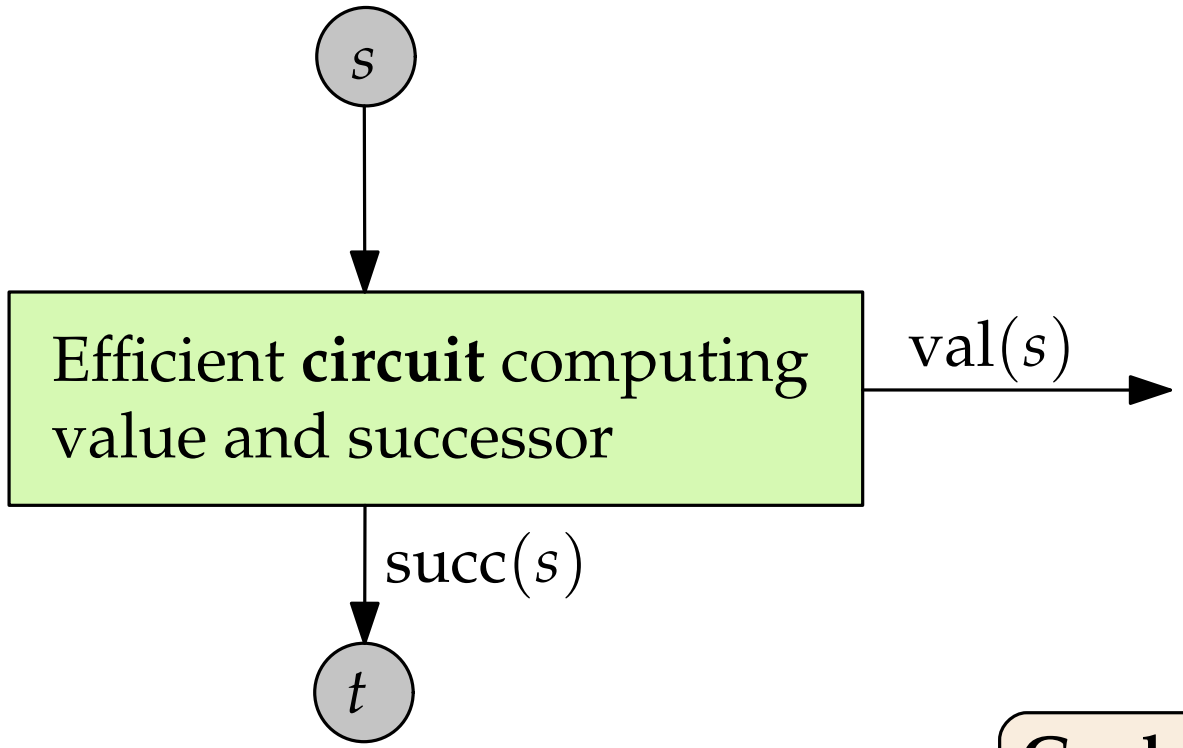
PLS—More Formal Definition

"The" generic PLS problem:



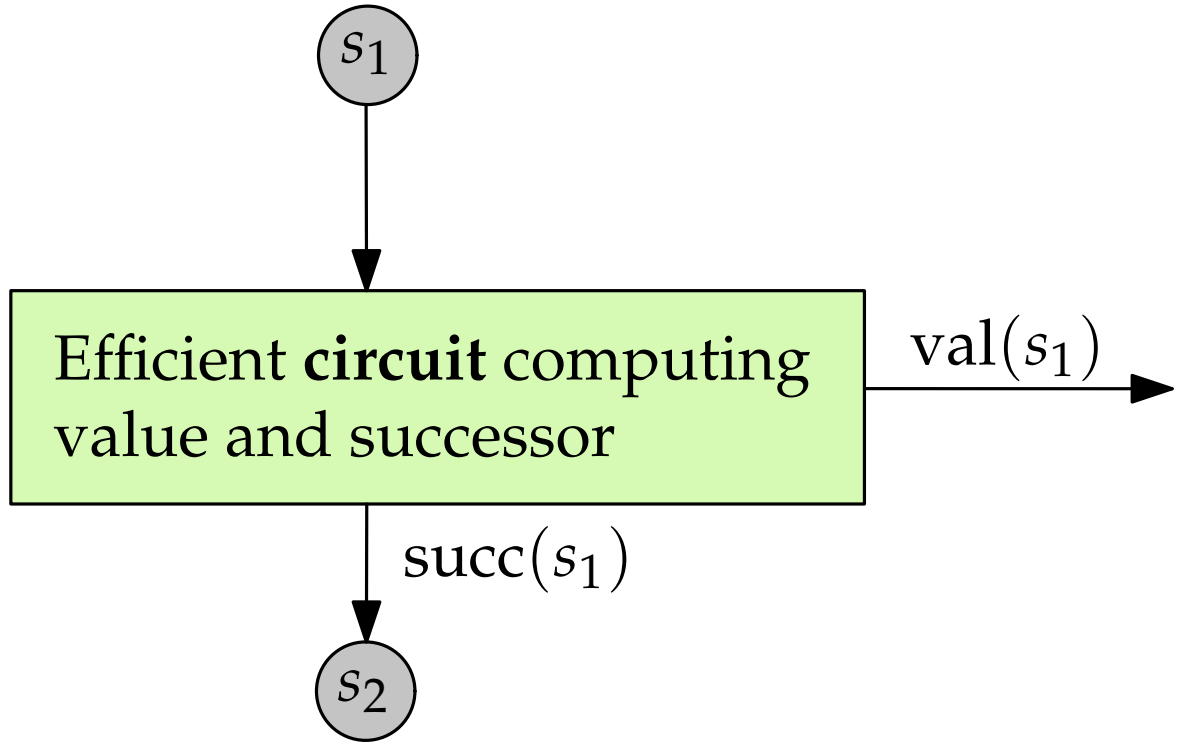
PLS—More Formal Definition

"The" generic PLS problem:



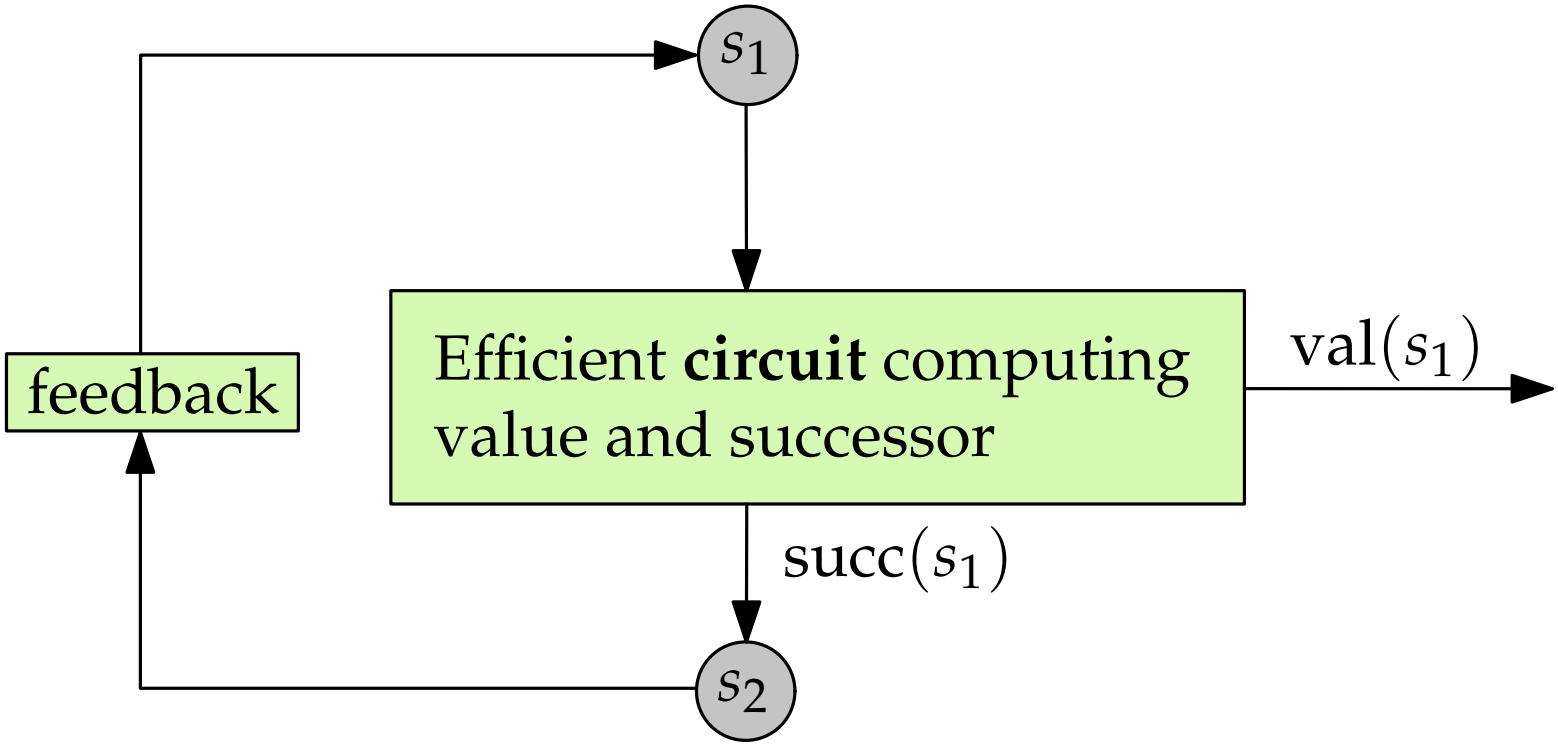
Goal: Find s such that successor t isn't better.

IterateCircuit



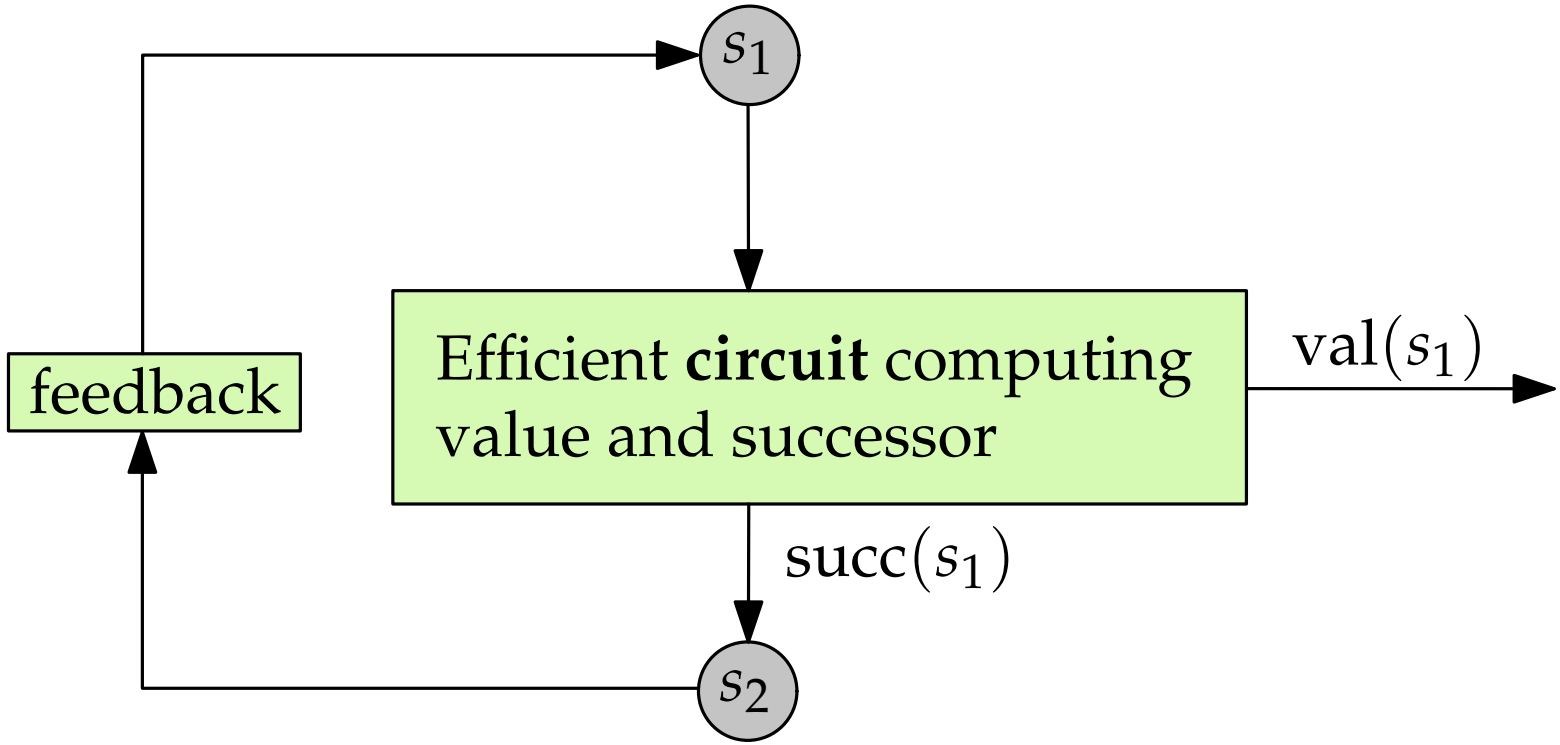
Goal: Find s such that $\mathrm{val}(s) \geq \mathrm{val}(\mathrm{succ}(s))$

IterateCircuit



Goal: Find s such that $\mathrm{val}(s) \geq \mathrm{val}(\mathrm{succ}(s))$

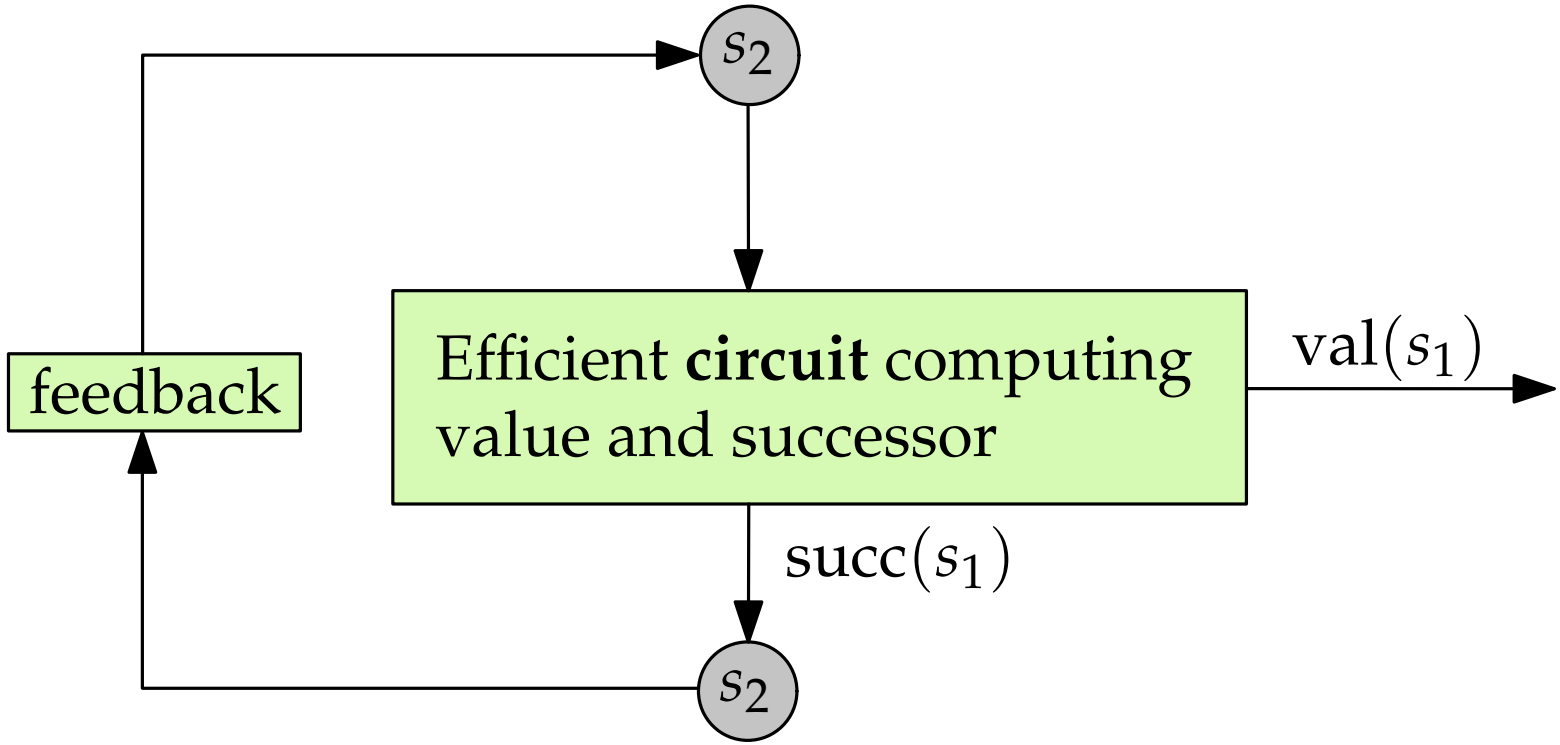
IterateCircuit



The "standard algorithm":

Goal: Find s such that $\mathrm{val}(s) \geq \mathrm{val}(\mathrm{succ}(s))$

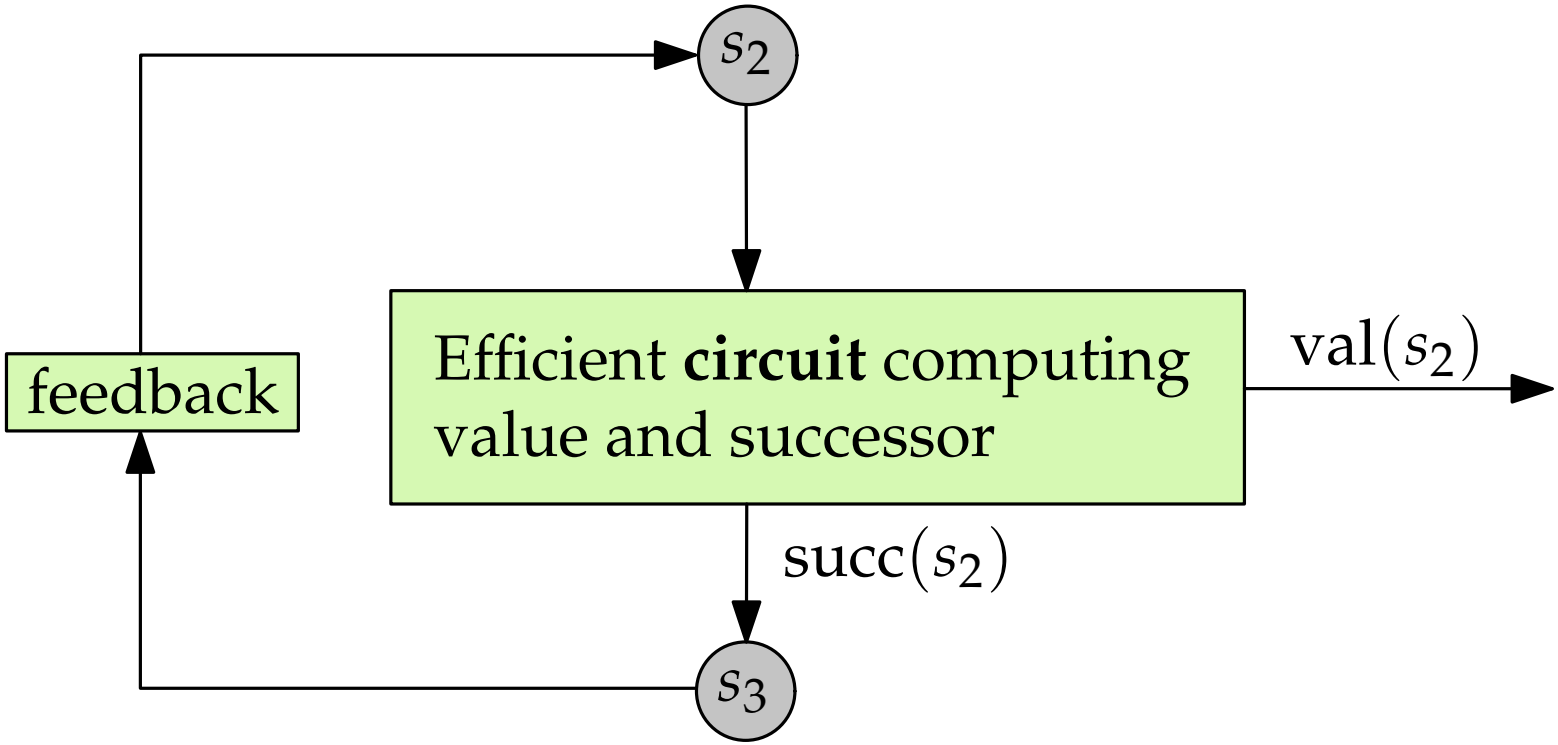
IterateCircuit



The "standard algorithm":

Goal: Find s such that $\text{val}(s) \geq \text{val}(\text{succ}(s))$

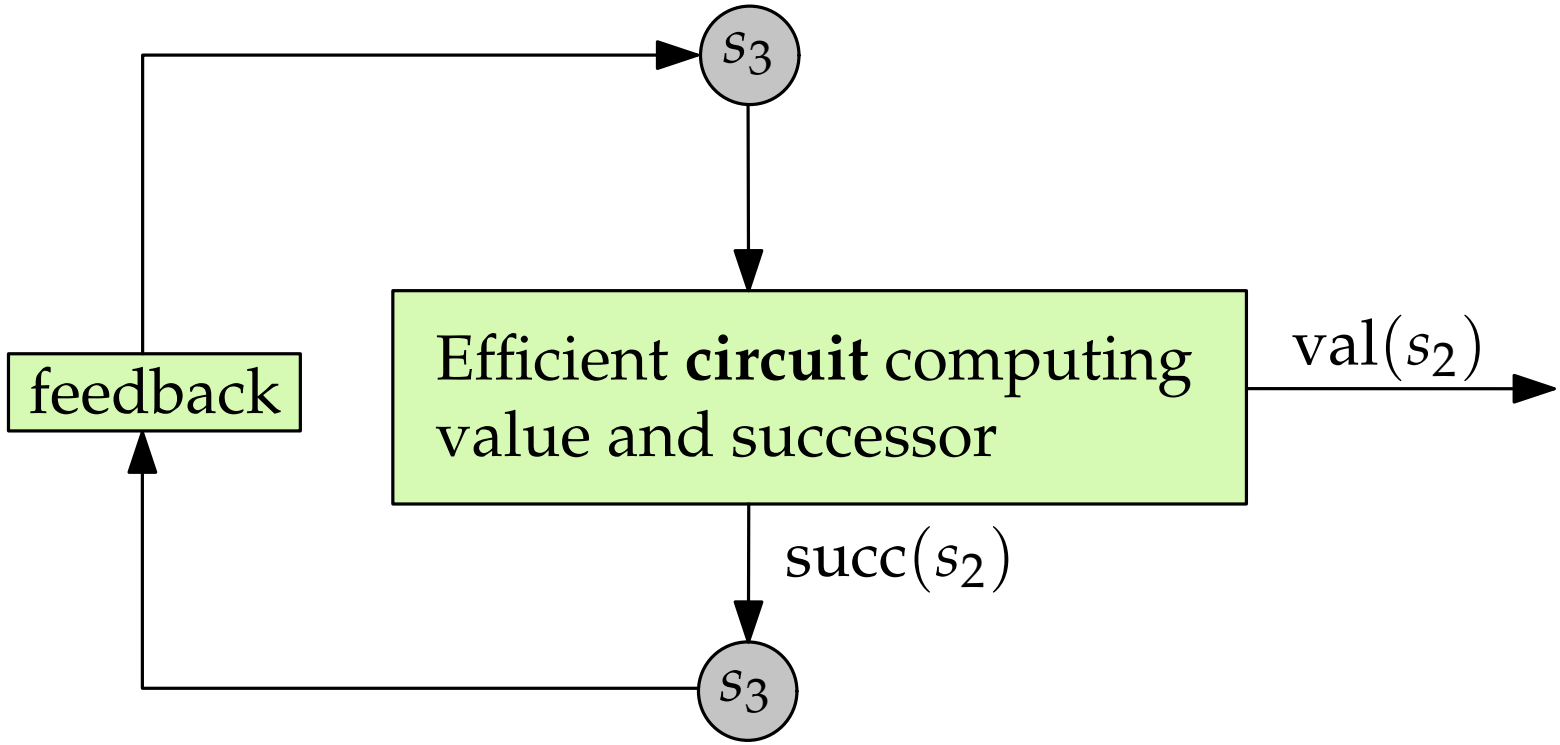
IterateCircuit



The "standard algorithm":

Goal: Find s such that $\text{val}(s) \geq \text{val}(\text{succ}(s))$

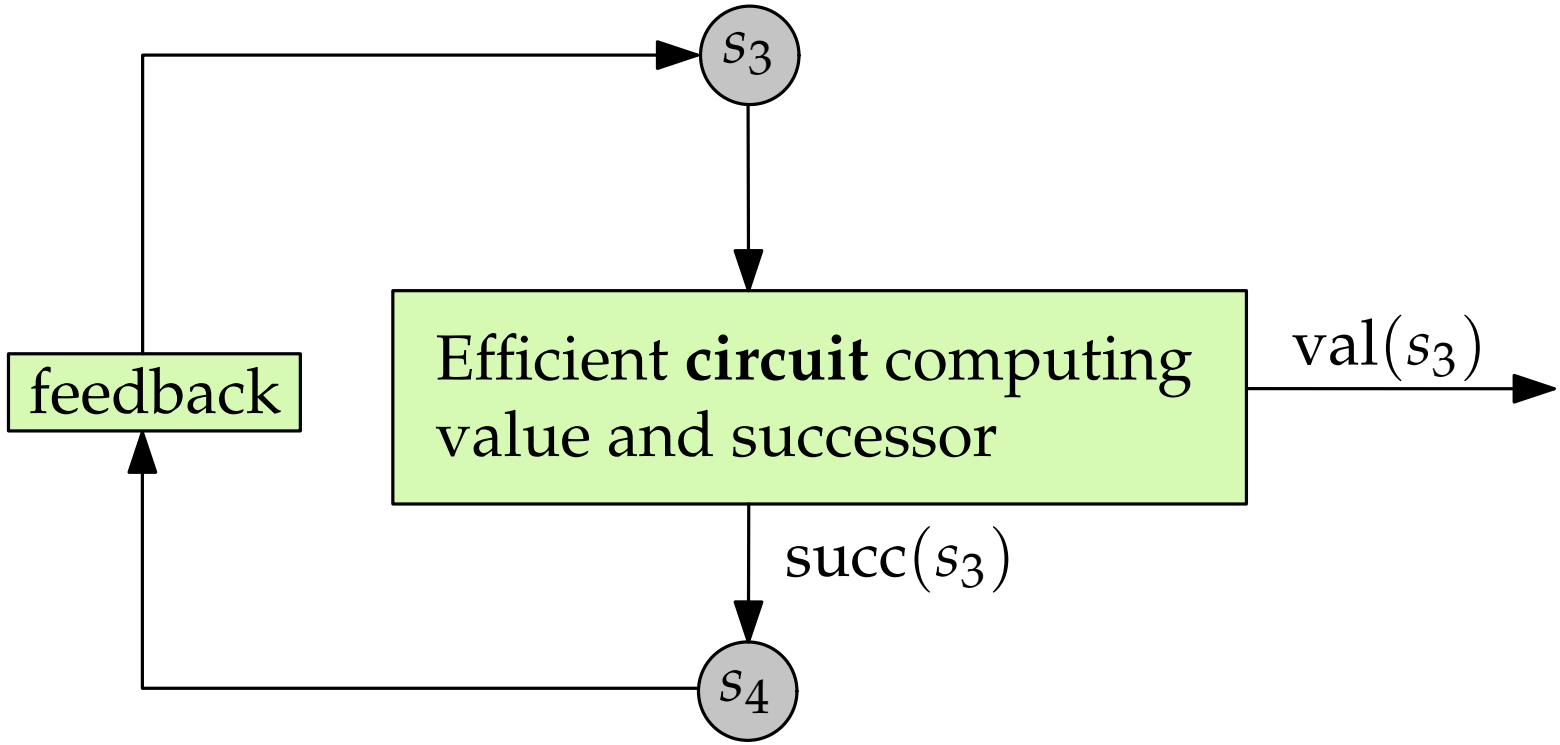
IterateCircuit



The "standard algorithm":

Goal: Find s such that $\text{val}(s) \geq \text{val}(\text{succ}(s))$

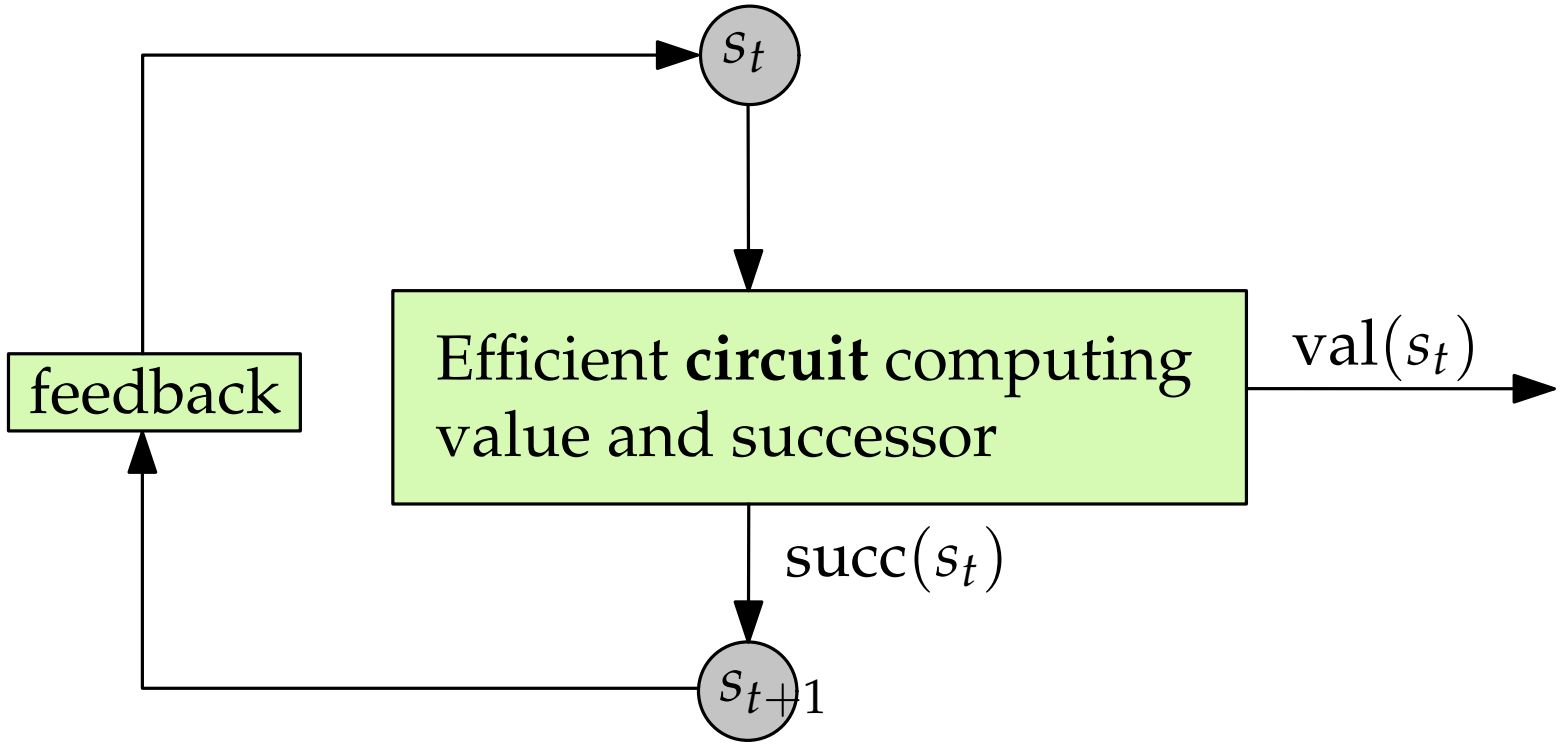
IterateCircuit



The "standard algorithm":

Goal: Find s such that $\text{val}(s) \geq \text{val}(\text{succ}(s))$

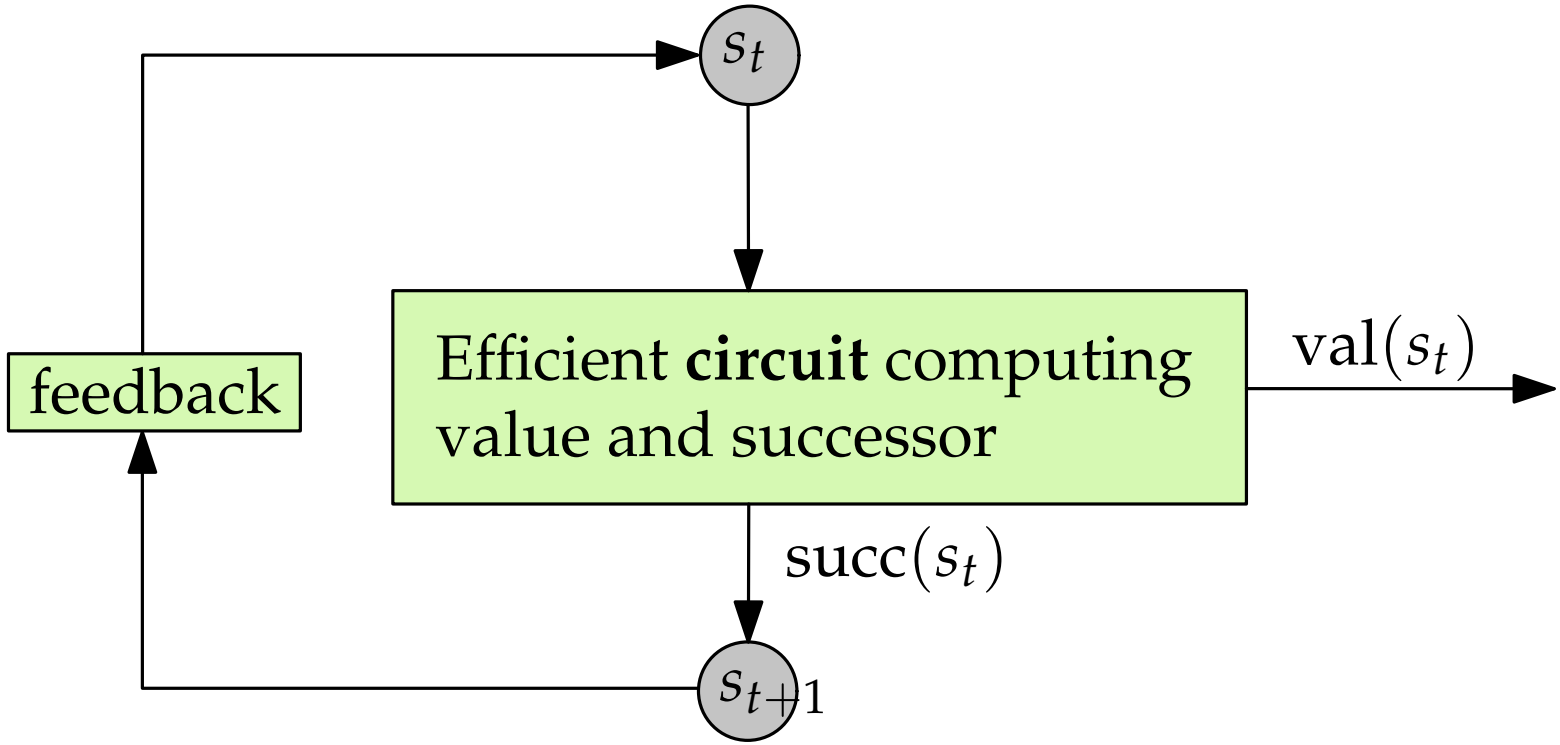
IterateCircuit



The "standard algorithm":

Goal: Find s such that $\text{val}(s) \geq \text{val}(\text{succ}(s))$

IterateCircuit

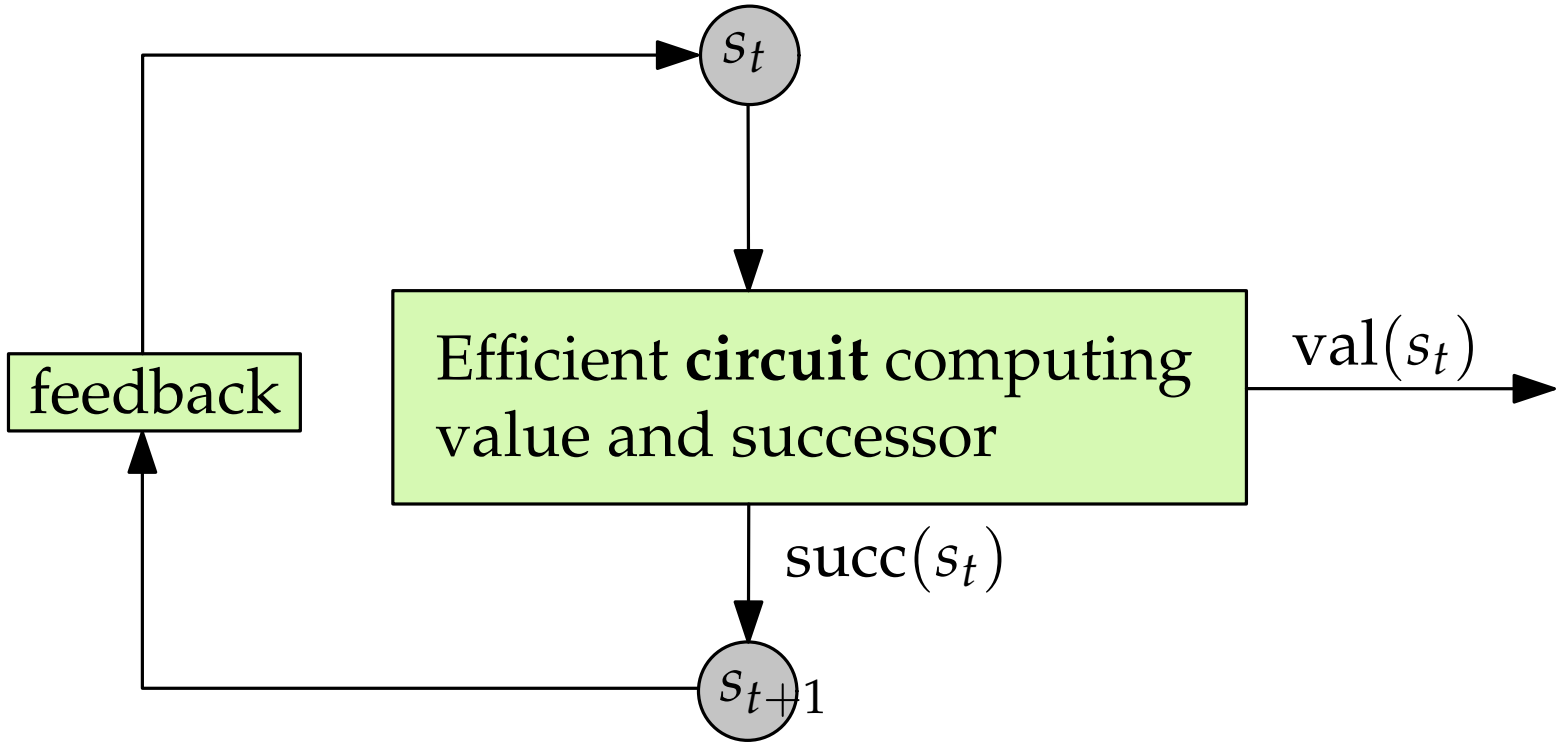


The "standard algorithm":

This t can become exponential.

Goal: Find s such that $\mathrm{val}(s) \geq \mathrm{val}(\mathrm{succ}(s))$

IterateCircuit



Goal: Find s such that $\text{val}(s) \geq \text{val}(\text{succ}(s))$

The "standard algorithm":

This t can become exponential.

PLS $\subseteq$ P not known, conjectured to be false.

Definition. PLS is the class of all problems that can be reduced to ITERATECIRCUIT.

Definition. PLS is the class of all problems that can be reduced to ITERATECIRCUIT.

Definition. A problem L is PLS-complete if ITERATECIRCUIT can be reduced to L.

Definition. PLS is the class of all problems that can be reduced to ITERATECIRCUIT.

Definition. A problem L is PLS-complete if ITERATECIRCUIT can be reduced to L.

ITERATECIRCUIT is very generic and not "user friendly". We want a more accessible PLS-complete problem.

Definition. PLS is the class of all problems that can be reduced to ITERATECIRCUIT.

Definition. A problem L is PLS-complete if ITERATECIRCUIT can be reduced to L.

ITERATECIRCUIT is very generic and not "user friendly". We want a more accessible PLS-complete problem.

Theorem (Johnson, Papadimitriou, Yannakakis). Local-Max-k-SAT is PLS-complete.

Definition. PLS is the class of all problems that can be reduced to ITERATECIRCUIT.

Definition. A problem L is PLS-complete if ITERATECIRCUIT can be reduced to L.

ITERATECIRCUIT is very generic and not "user friendly". We want a more accessible PLS-complete problem.

Theorem (Johnson, Papadimitriou, Yannakakis). Local-Max-k-SAT is PLS-complete.

SATISFIABILITY (SAT)

a propositional formula in conjunctive normal form
(a CNF formula)

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

SATISFIABILITY (SAT)

a propositional formula in conjunctive normal form
(a CNF formula)

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 0$
$x_4 \mapsto 0$

a truth assignment α

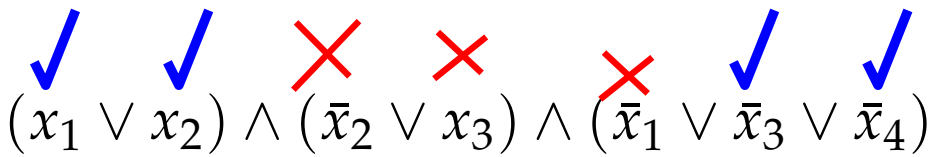
SATISFIABILITY (SAT)

a propositional formula in conjunctive normal form
(a CNF formula)

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 0$
$x_4 \mapsto 0$

a truth assignment α



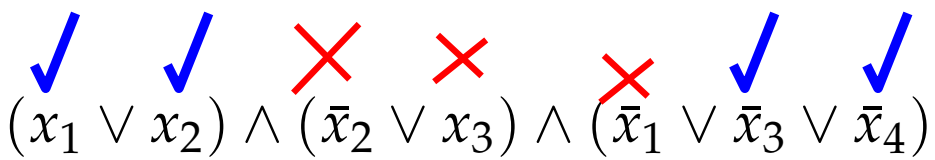
SATISFIABILITY (SAT)

a propositional formula in conjunctive normal form
(a CNF formula)

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 0$
$x_4 \mapsto 0$

a truth assignment α



$F(\alpha) = 0$

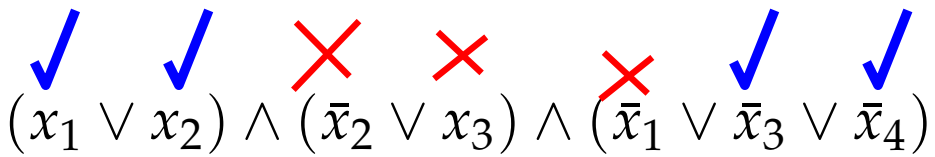
SATISFIABILITY (SAT)

a propositional formula in conjunctive normal form
(a CNF formula)

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 0$
$x_4 \mapsto 0$

a truth assignment α



$F(\alpha) = 0$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 1$
$x_4 \mapsto 0$

a truth assignment β

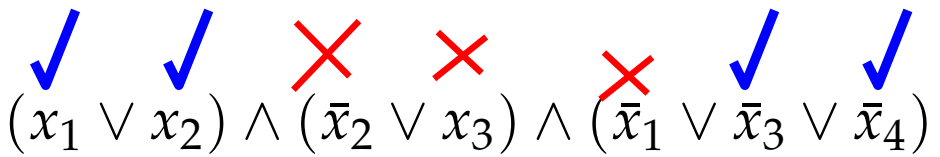
SATISFIABILITY (SAT)

a propositional formula in conjunctive normal form
(a CNF formula)

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 0$
$x_4 \mapsto 0$

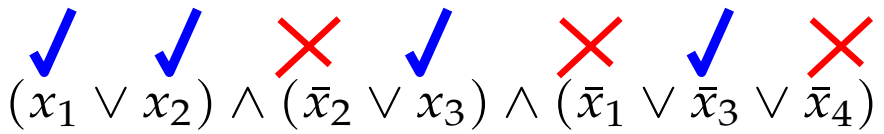
a truth assignment α



$F(\alpha) = 0$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 1$
$x_4 \mapsto 0$

a truth assignment β



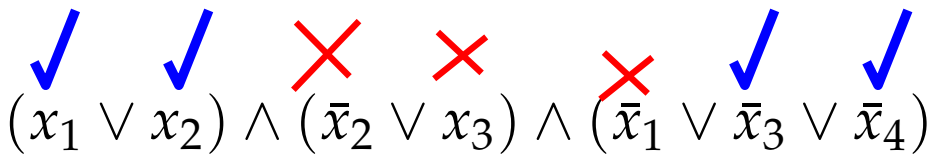
SATISFIABILITY (SAT)

a propositional formula in conjunctive normal form
(a CNF formula)

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 0$
$x_4 \mapsto 0$

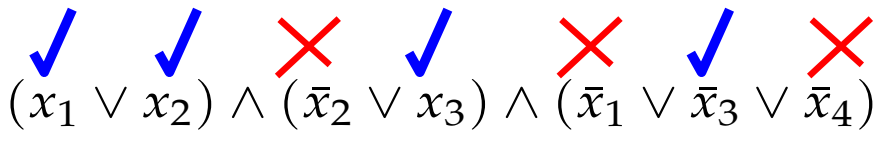
a truth assignment α



$F(\alpha) = 0$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 1$
$x_4 \mapsto 0$

a truth assignment β



$F(\beta) = 1$

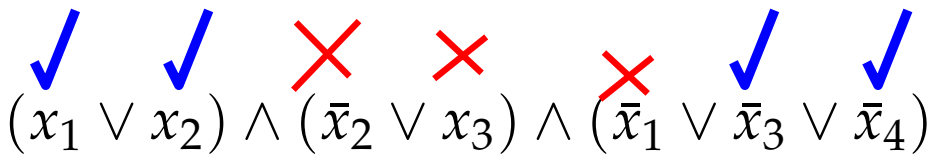
SATISFIABILITY (SAT)

a propositional formula in conjunctive normal form
(a CNF formula)

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 0$
$x_4 \mapsto 0$

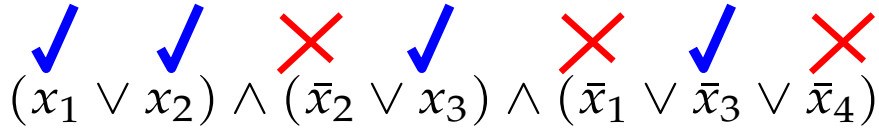
a truth assignment α



$F(\alpha) = 0$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 1$
$x_4 \mapsto 0$

a truth assignment β
a *satisfying assignment*



$F(\beta) = 1$

SATISFIABILITY (SAT)

a propositional formula in conjunctive normal form
(a CNF formula)

largest clause: 3

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 0$
$x_4 \mapsto 0$

a truth assignment α

✓ ✓ ✗ ✗ ✗ ✓ ✓

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

$F(\alpha) = 0$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 1$
$x_4 \mapsto 0$

a truth assignment β

a *satisfying assignment*

✓ ✓ ✗ ✓ ✗ ✓ ✗

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

$F(\beta) = 1$

3-SATISFIABILITY (3-SAT)

a propositional formula in conjunctive normal form
(a 3-CNF formula)

largest clause: 3

$$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 0$
$x_4 \mapsto 0$

a truth assignment α

✓ ✓ ✗ ✗ ✗ ✓ ✓
$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$

$F(\alpha) = 0$

$x_1 \mapsto 1$
$x_2 \mapsto 1$
$x_3 \mapsto 1$
$x_4 \mapsto 0$

a truth assignment β

a *satisfying assignment*

✓ ✓ ✗ ✓ ✗ ✓ ✗
$(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$

$F(\beta) = 1$

Problem (k-SAT). Given a k-CNF formula F, find a truth assignment α such that $F(\alpha) = 1$.

Problem (k-SAT). Given a k-CNF formula F, find a truth assignment α such that $F(\alpha) = 1$.

Problem (CIRCUIT-SAT). Given a Boolean circuit C, find a truth assignment α such that $C(\alpha) = 1$.

Problem (k-SAT). Given a k-CNF formula F, find a truth assignment α such that $F(\alpha) = 1$.

Problem (CIRCUIT-SAT). Given a Boolean circuit C, find a truth assignment α such that $C(\alpha) = 1$.

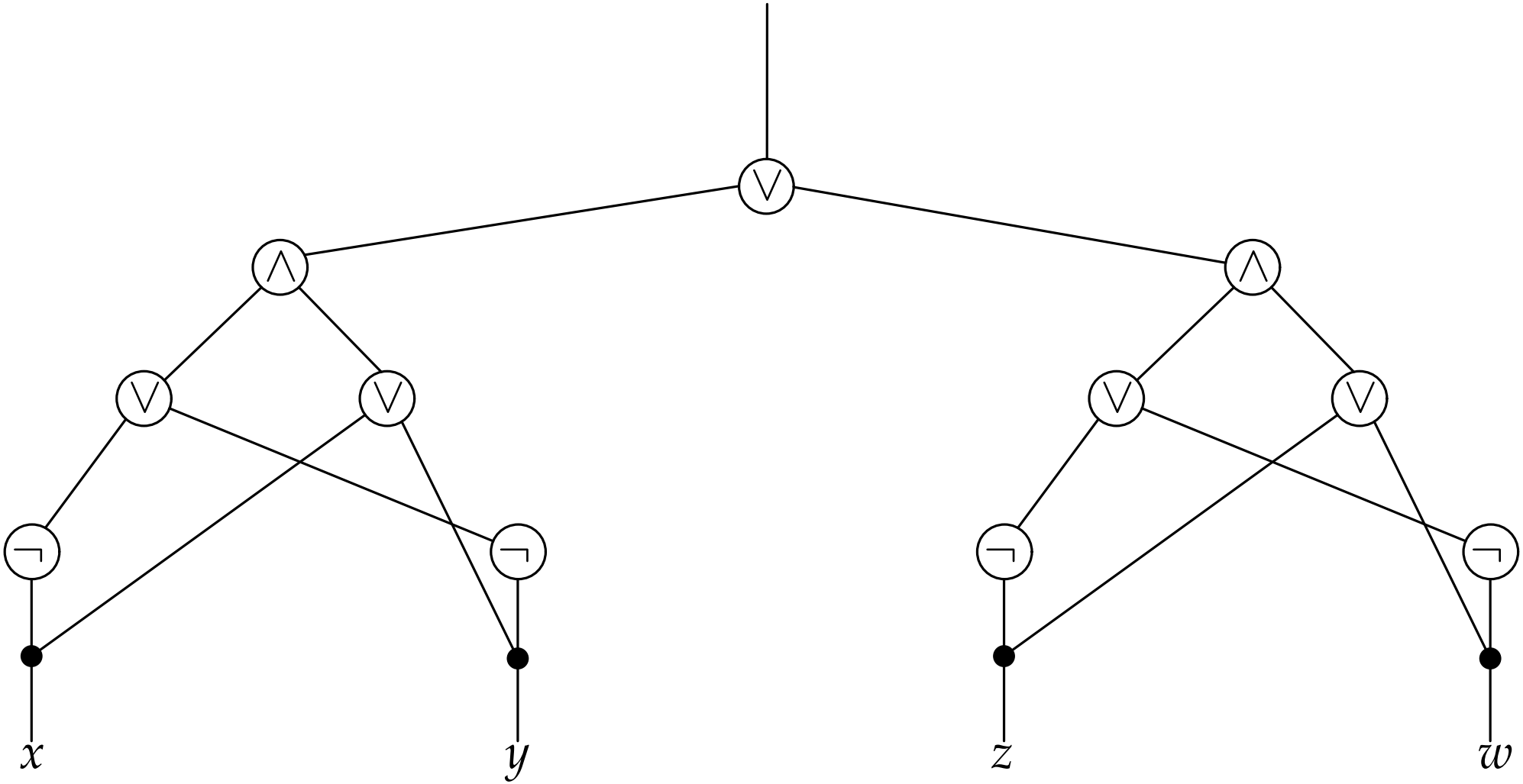
Theorem (Cook-Levin). CIRCUIT-SAT can be reduced to 3-SAT.

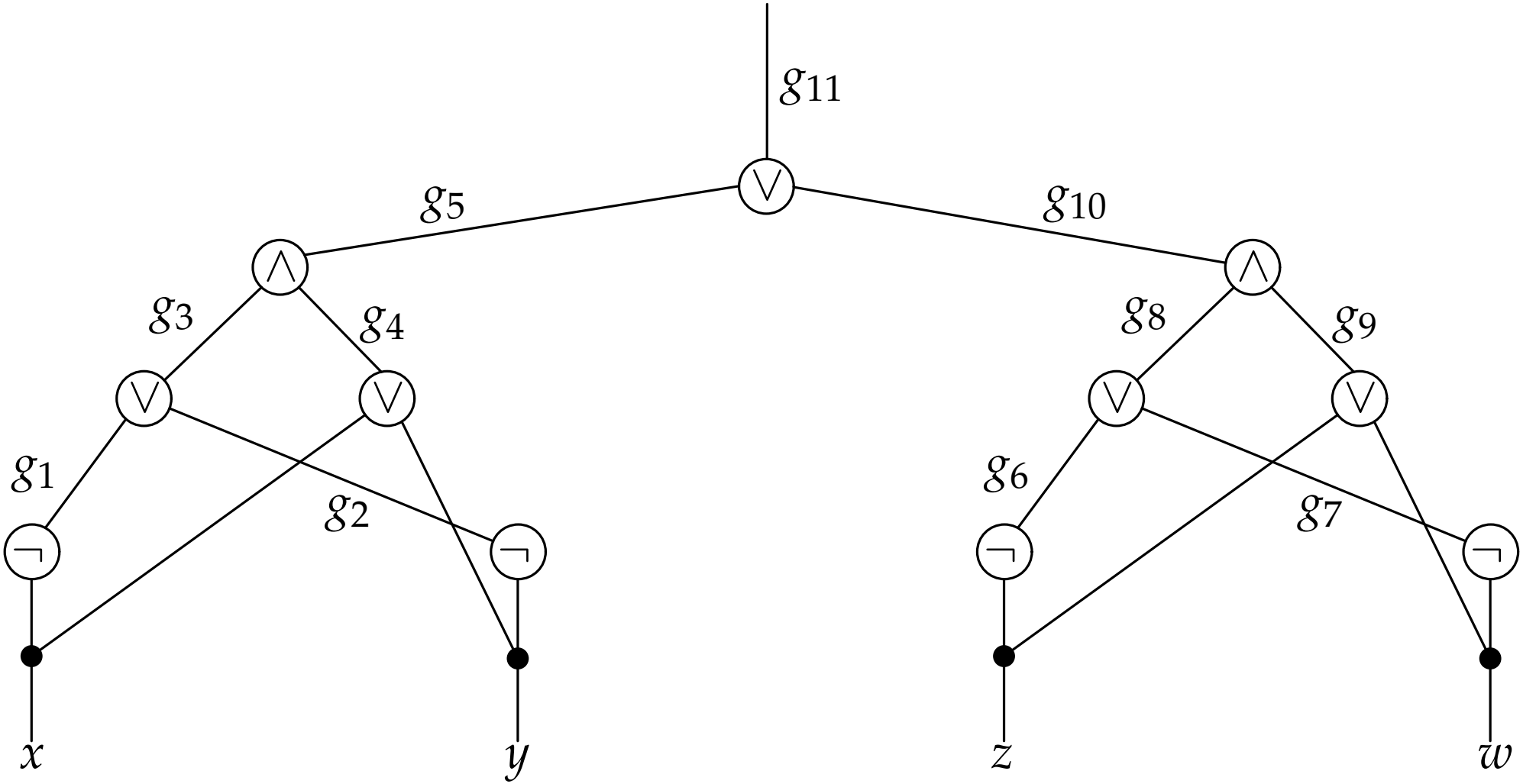
Problem (k-SAT). Given a k-CNF formula F, find a truth assignment α such that $F(\alpha) = 1$.

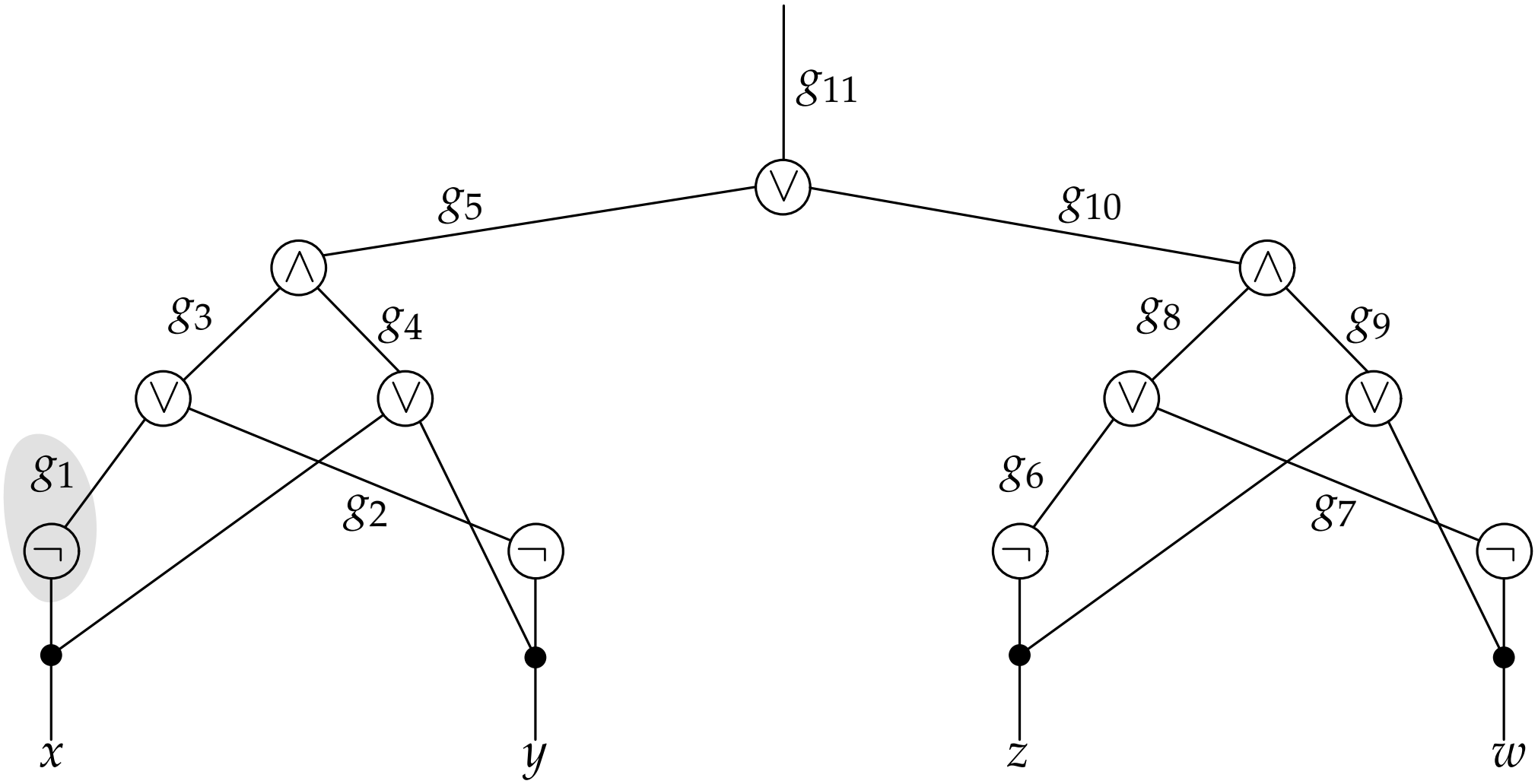
Problem (CIRCUIT-SAT). Given a Boolean circuit C, find a truth assignment α such that $C(\alpha) = 1$.

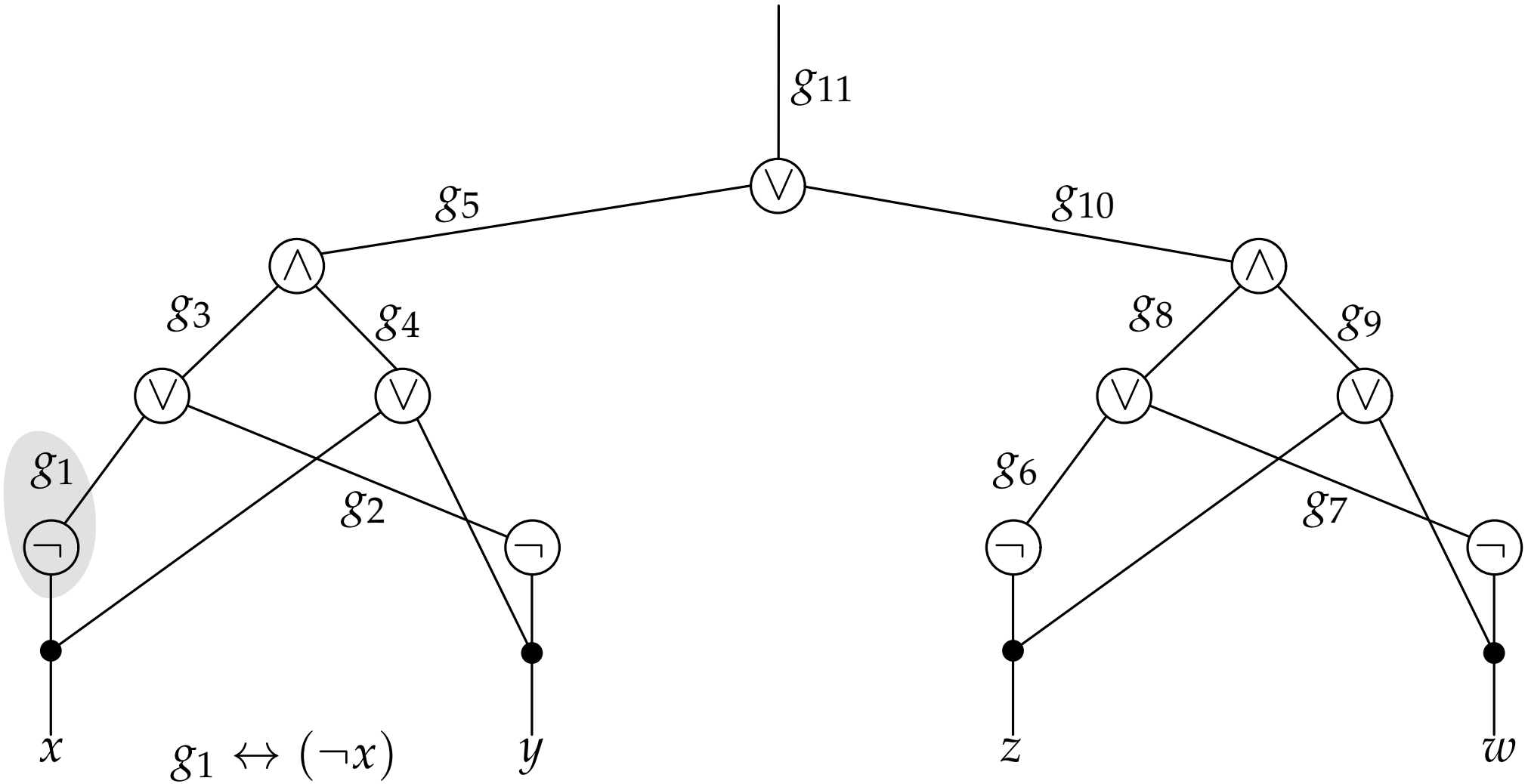
Theorem (Cook-Levin). CIRCUIT-SAT can be reduced to 3-SAT. In particular: we can transform a circuit $C(x)$ into a 3-CNF formula $F(x, y)$ such that

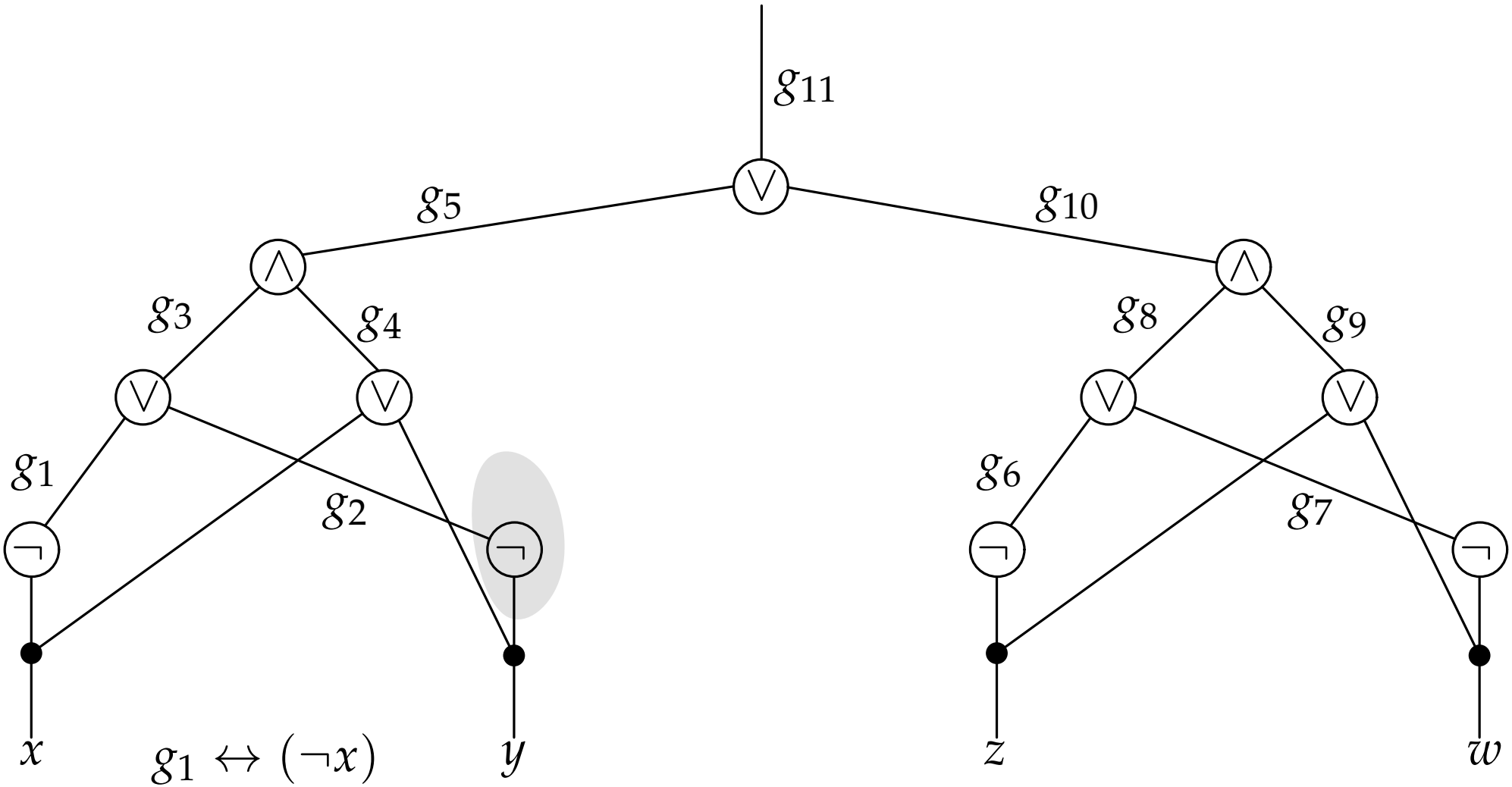
$$\forall x \in \{0,1\}^n : C(x) = 1 \iff \exists y \in \{0,1\}^m : F(x,y) = 1 \,.$$

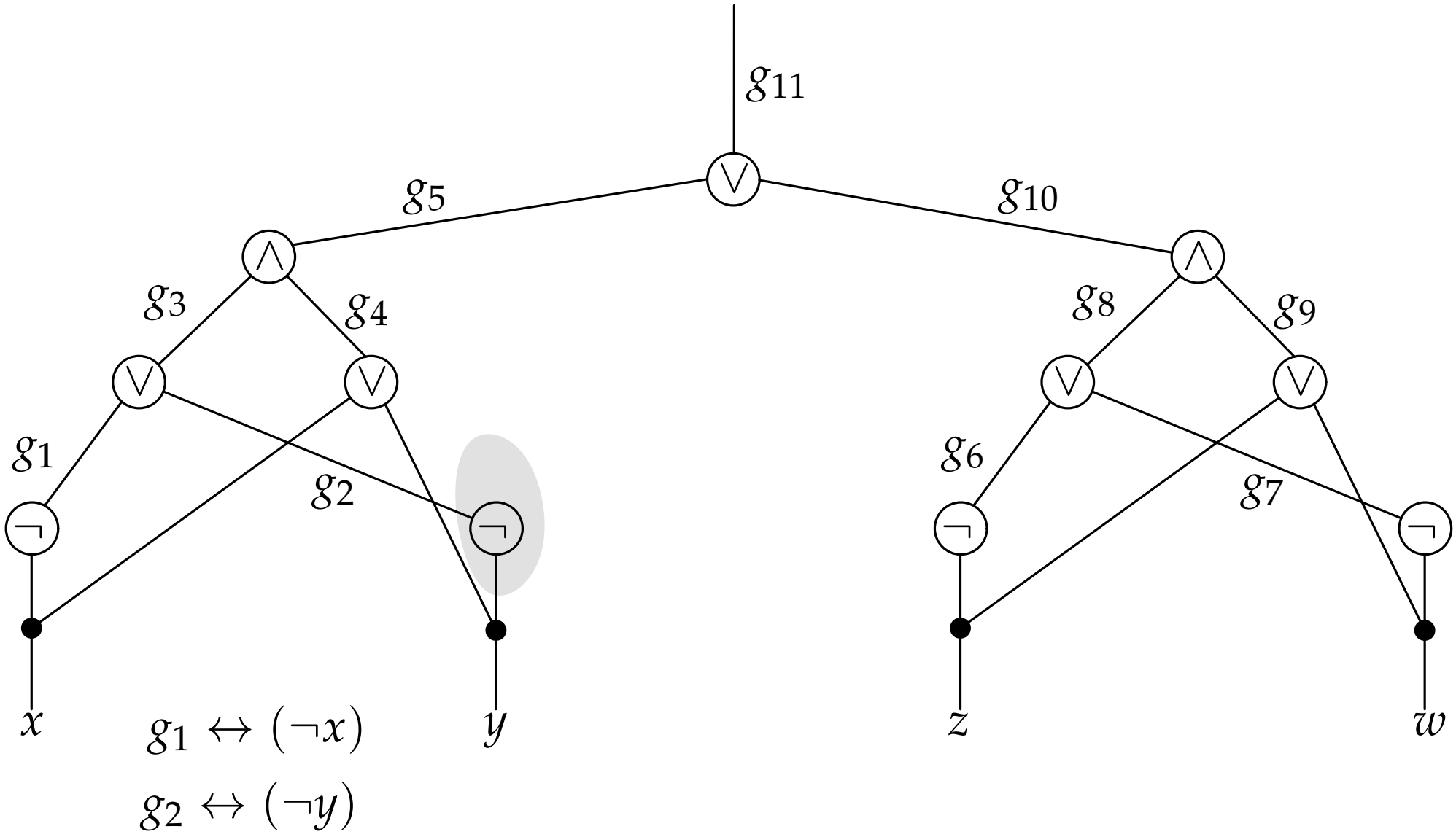

∨
∧
∧
∨
∨
∨
∨
¬
¬
¬
¬
x
y
z
w

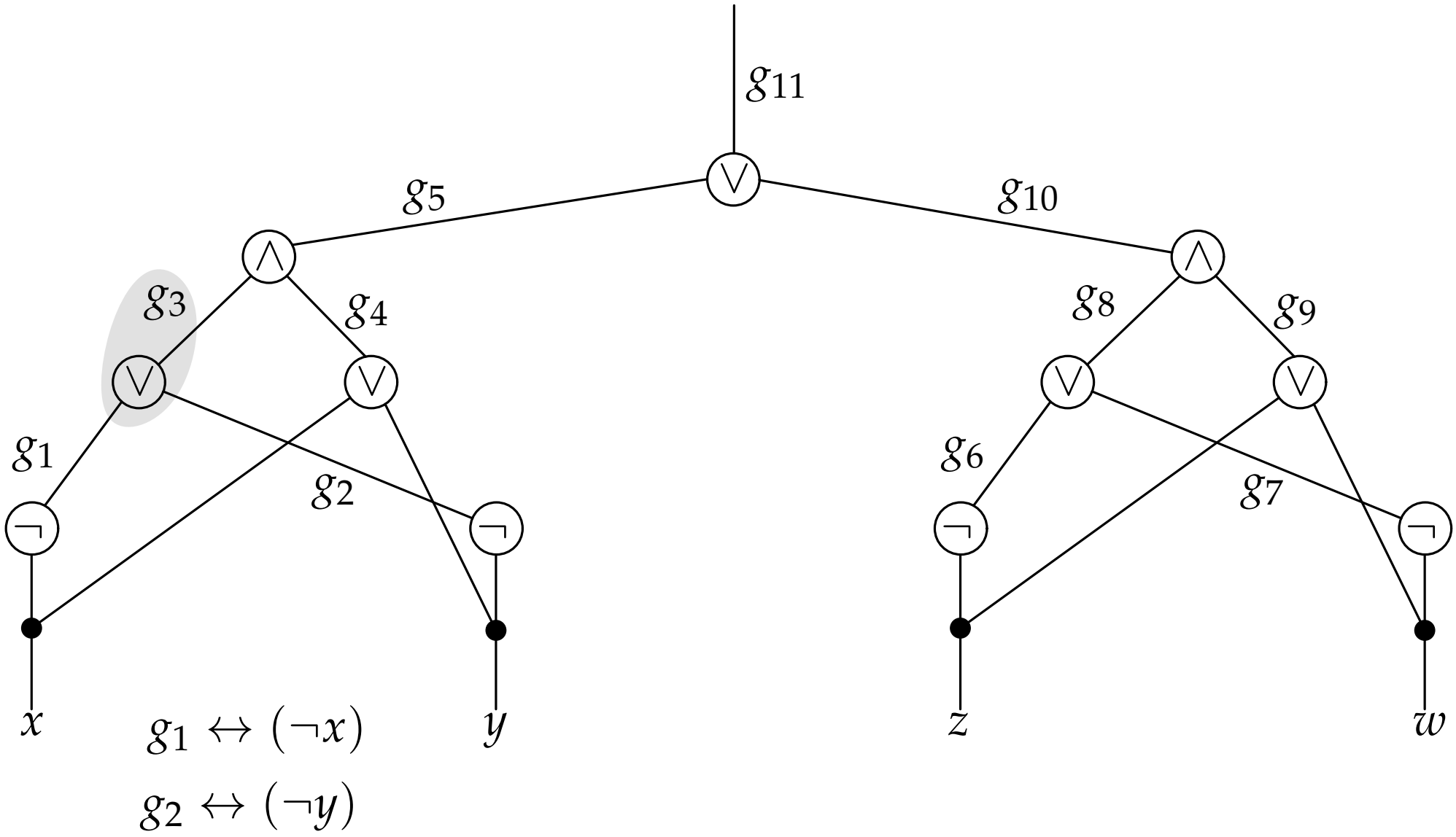

g_{11}
g_5
g_{10}
g_3
g_4
g_8
g_9
g_1
g_2
g_6
g_7
x
y
z
w

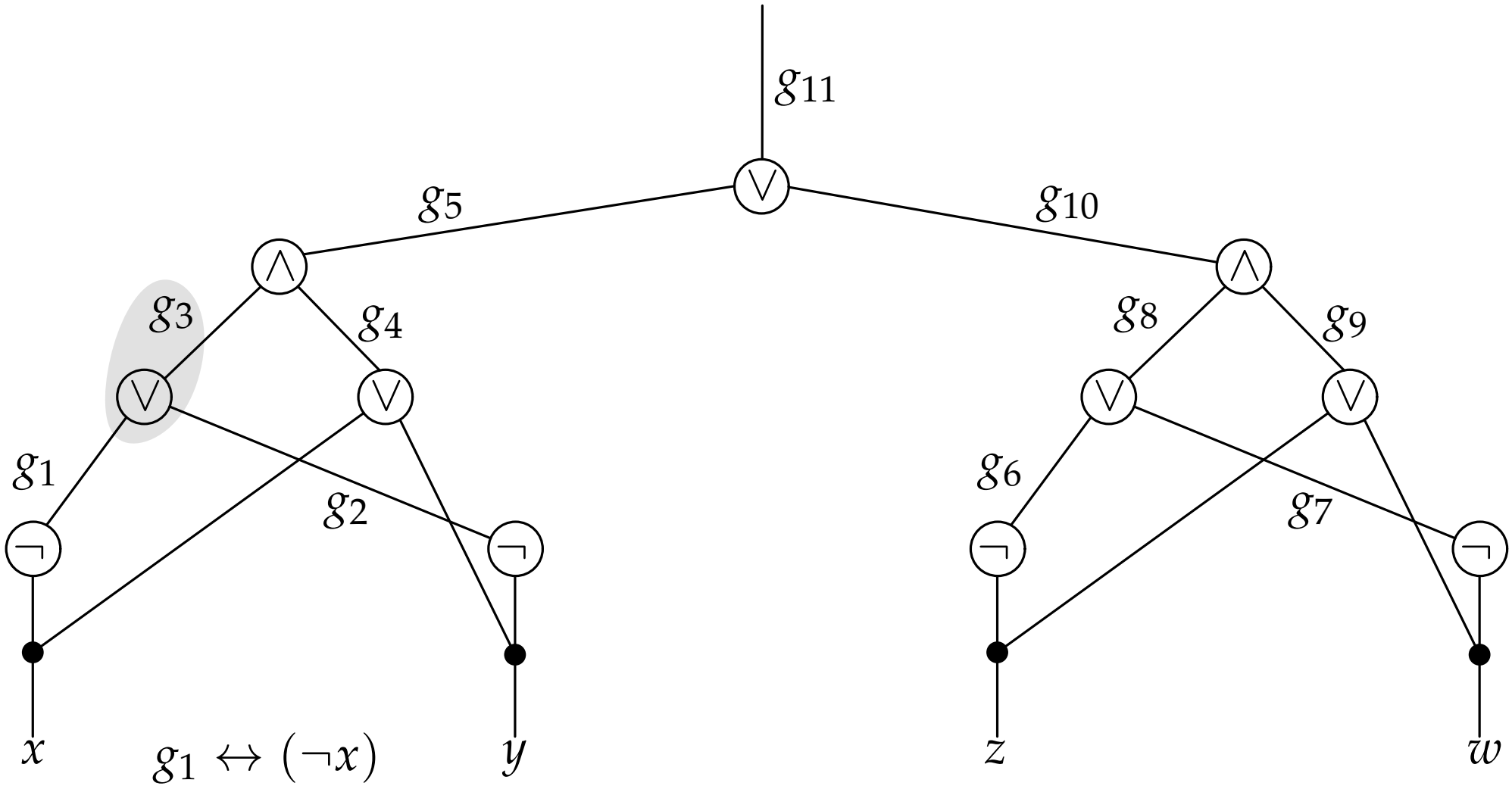

g_{11}
g_5
g_{10}
g_3
g_4
g_8
g_9
g_1
g_2
g_6
g_7
x
y
z
w


g_{11}
g_5
g_{10}
g_3
g_4
g_8
g_9
g_1
g_2
g_6
g_7
x
y
z
w
$g_1 \leftrightarrow (\neg x)$


g_{11}
g_5
g_{10}
g_3
g_4
g_8
g_9
g_1
g_2
g_6
g_7
x
y
z
w
$g_1 \leftrightarrow (\neg x)$


g_{11}
g_5
g_{10}
g_3
g_4
g_8
g_9
g_1
g_2
g_6
g_7
x
y
z
w
$g_1 \leftrightarrow (\neg x)$
$g_2 \leftrightarrow (\neg y)$

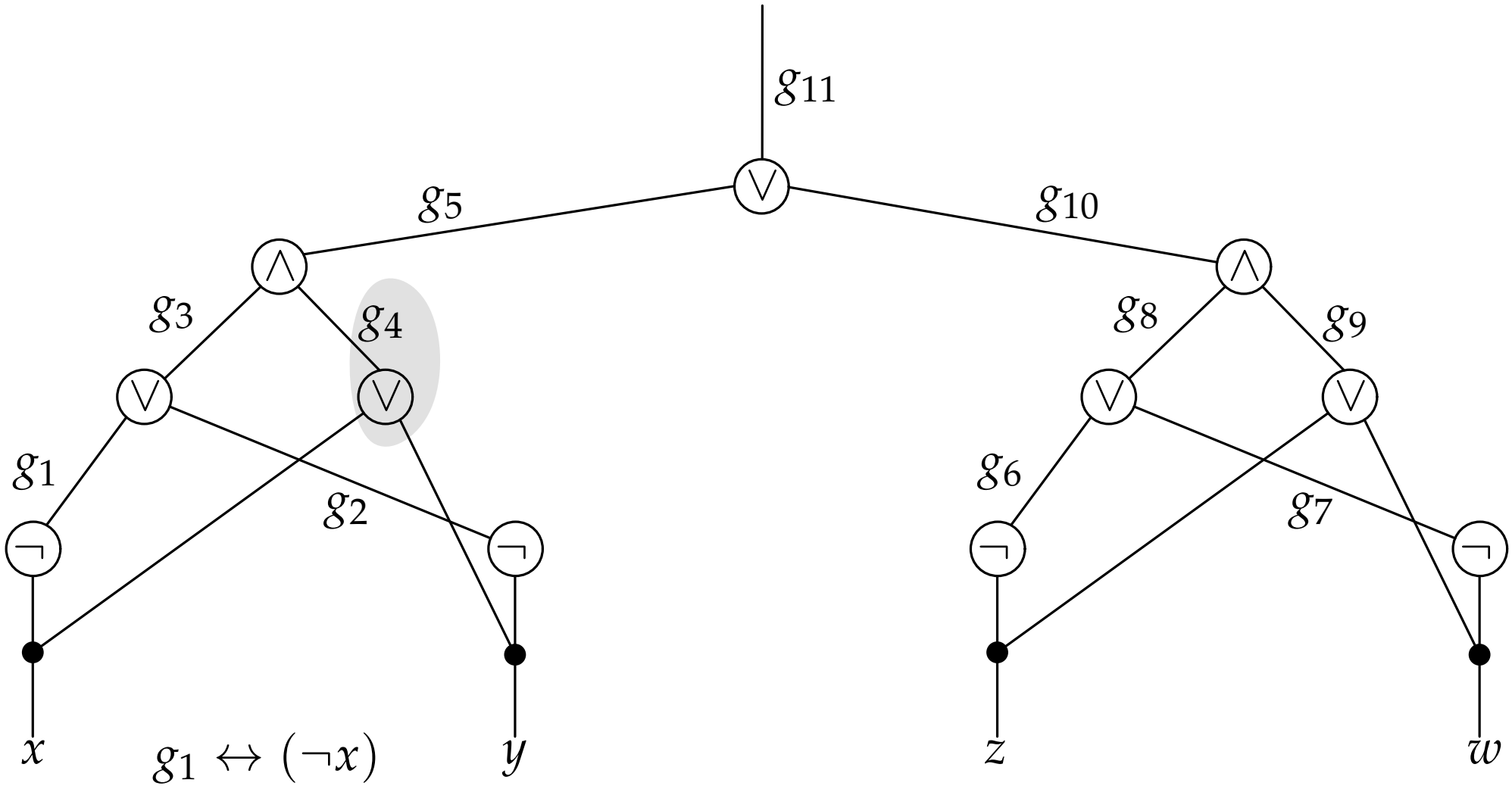

g_{11}
g_5
g_{10}
g_3
g_4
g_8
g_9
g_1
g_2
g_6
g_7
x
y
z
w
$g_1 \leftrightarrow (\neg x)$
$g_2 \leftrightarrow (\neg y)$



$$g_1 \leftrightarrow (\neg x)$$

$$g_2 \leftrightarrow (\neg y)$$

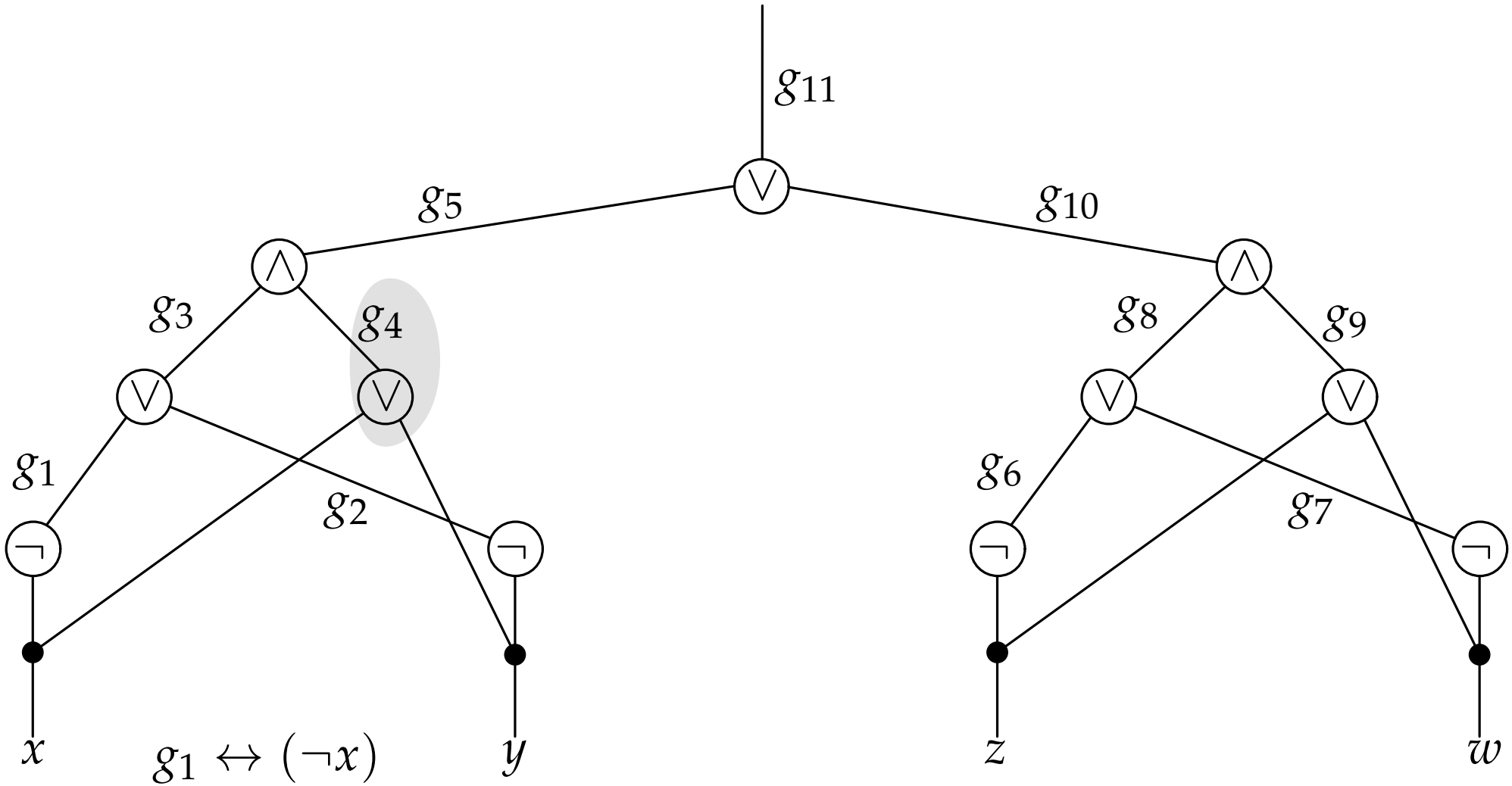
$$g_3 \leftrightarrow (g_1 \vee g_2)$$



$g_1 \leftrightarrow (\neg x)$

$g_2 \leftrightarrow (\neg y)$

$g_3 \leftrightarrow (g_1 \vee g_2)$

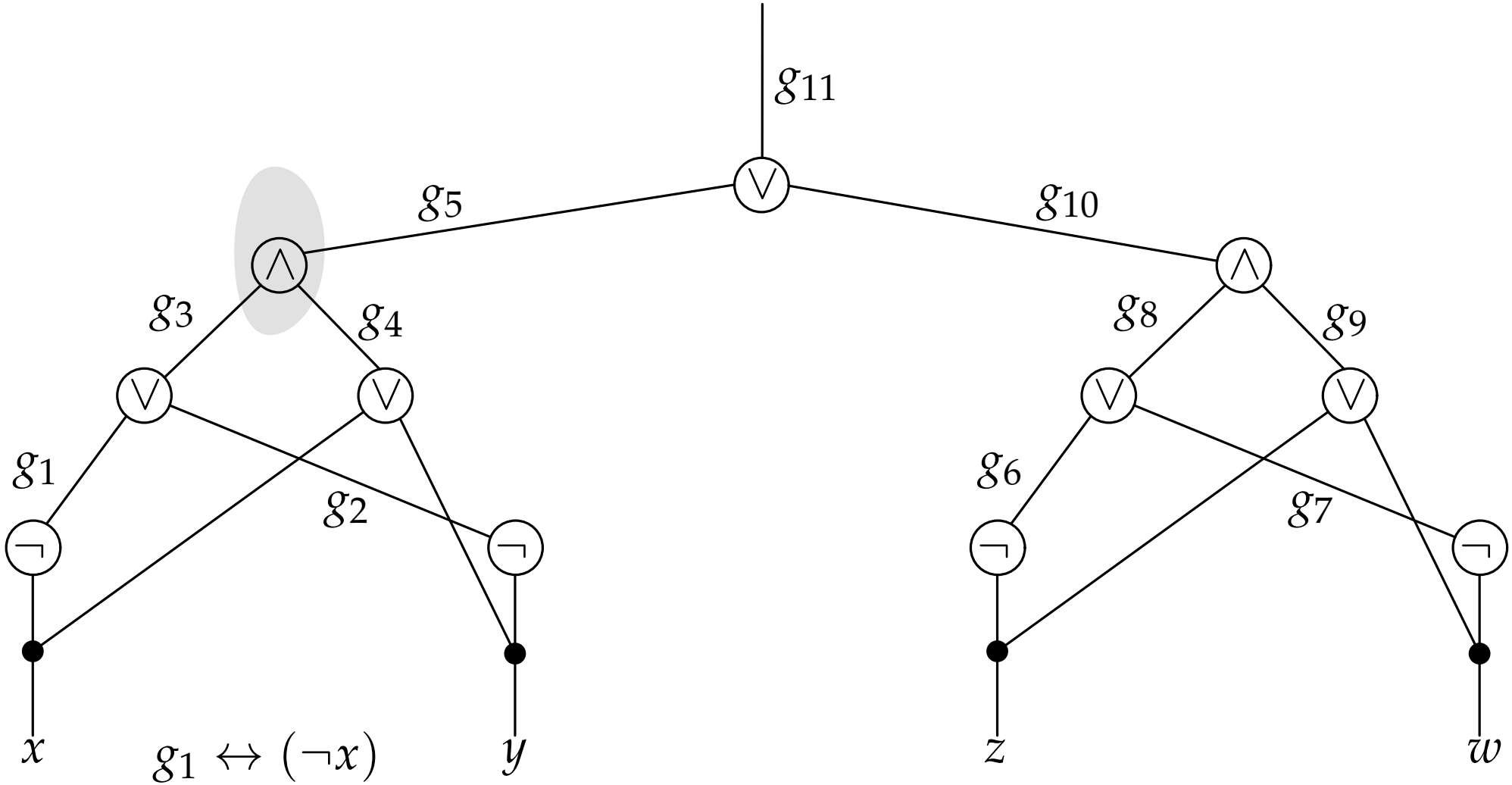


$$g_1 \leftrightarrow (\neg x)$$

$$g_2 \leftrightarrow (\neg y)$$

$$g_3 \leftrightarrow (g_1 \vee g_2)$$

$$g_4 \leftrightarrow (x \vee y)$$

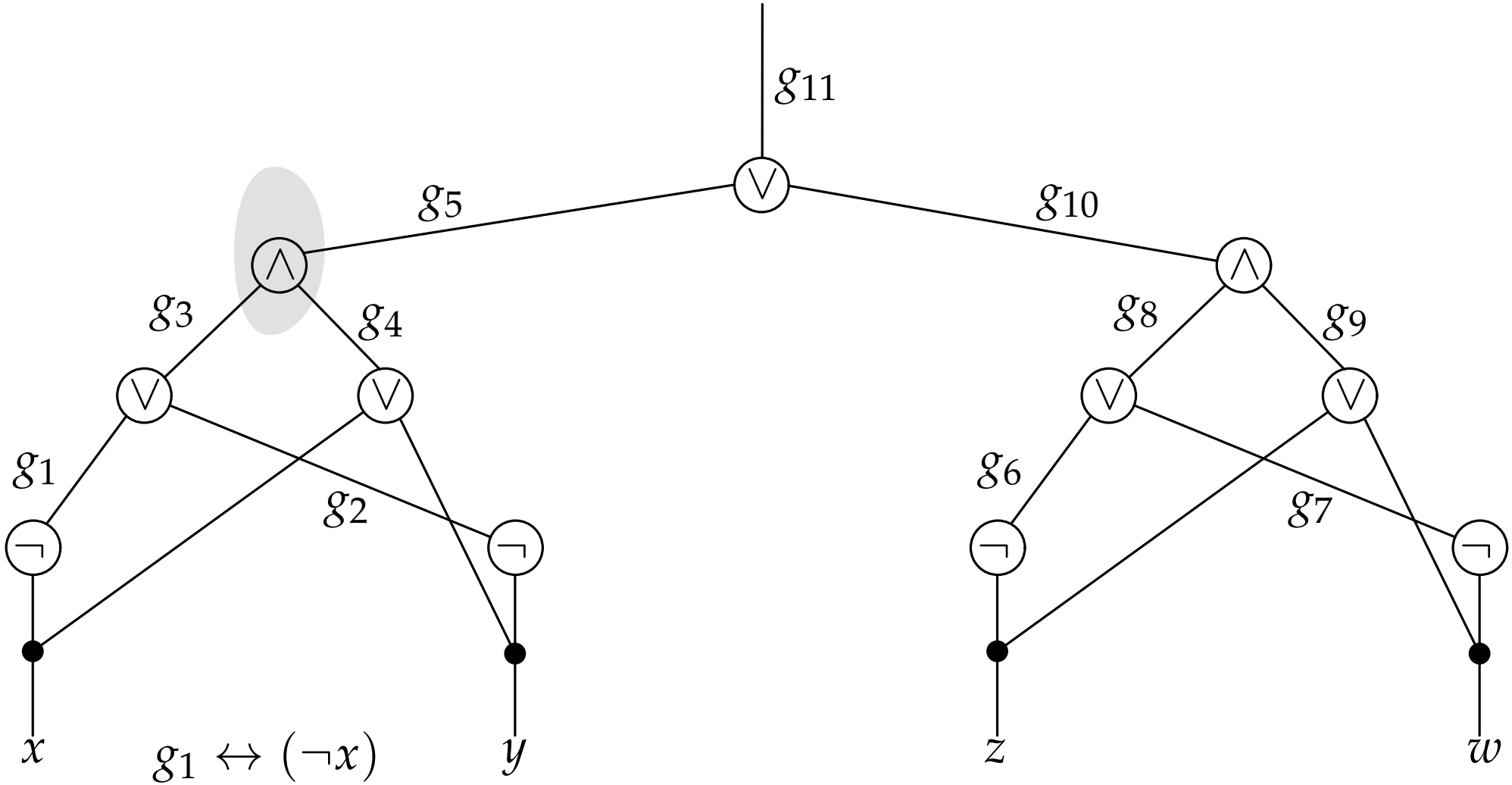


$$g_1 \leftrightarrow (\neg x)$$

$$g_2 \leftrightarrow (\neg y)$$

$$g_3 \leftrightarrow (g_1 \vee g_2)$$

$$g_4 \leftrightarrow (x \vee y)$$



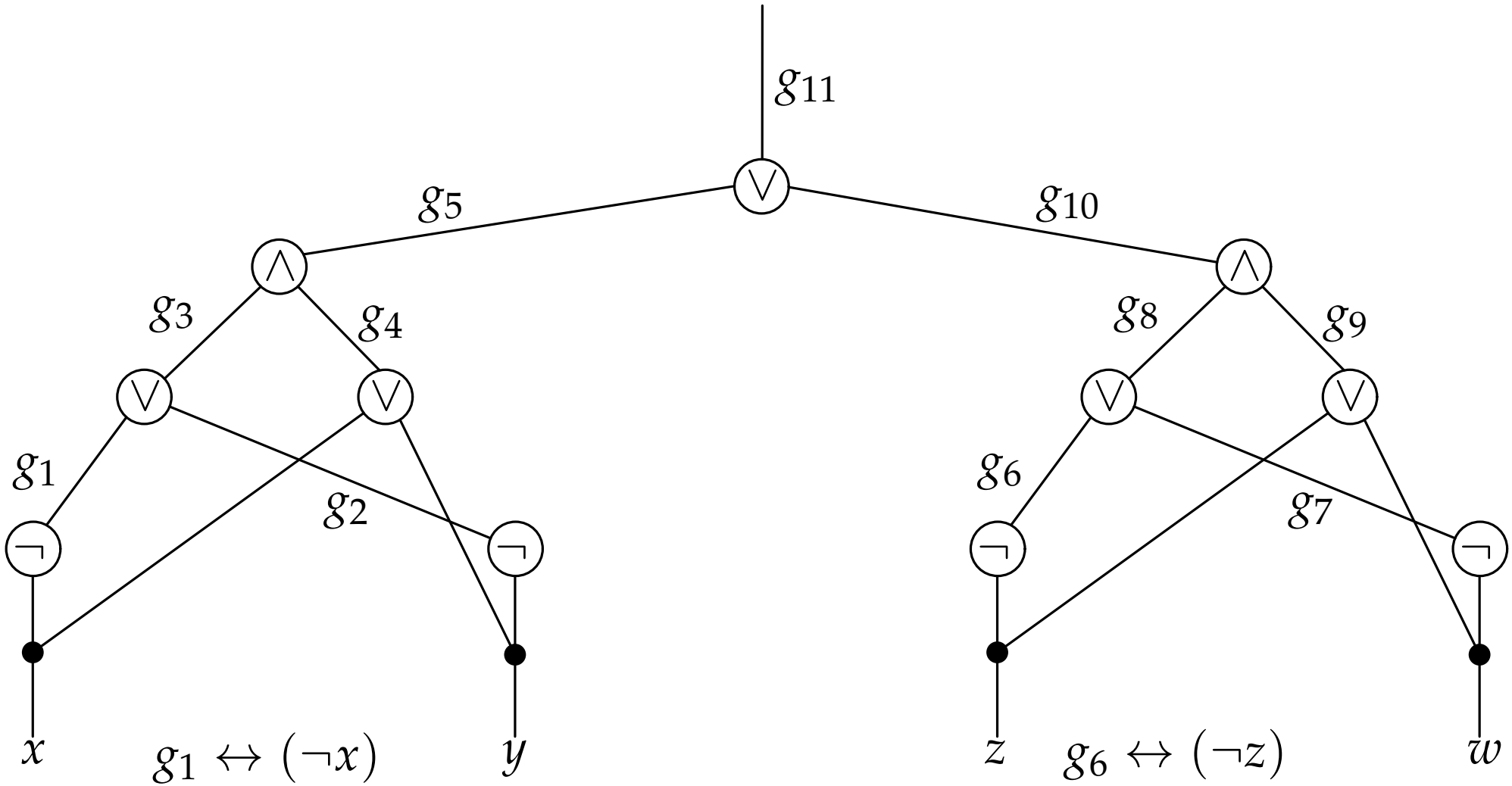
$$g_1 \leftrightarrow (\neg x)$$

$$g_2 \leftrightarrow (\neg y)$$

$$g_3 \leftrightarrow (g_1 \vee g_2)$$

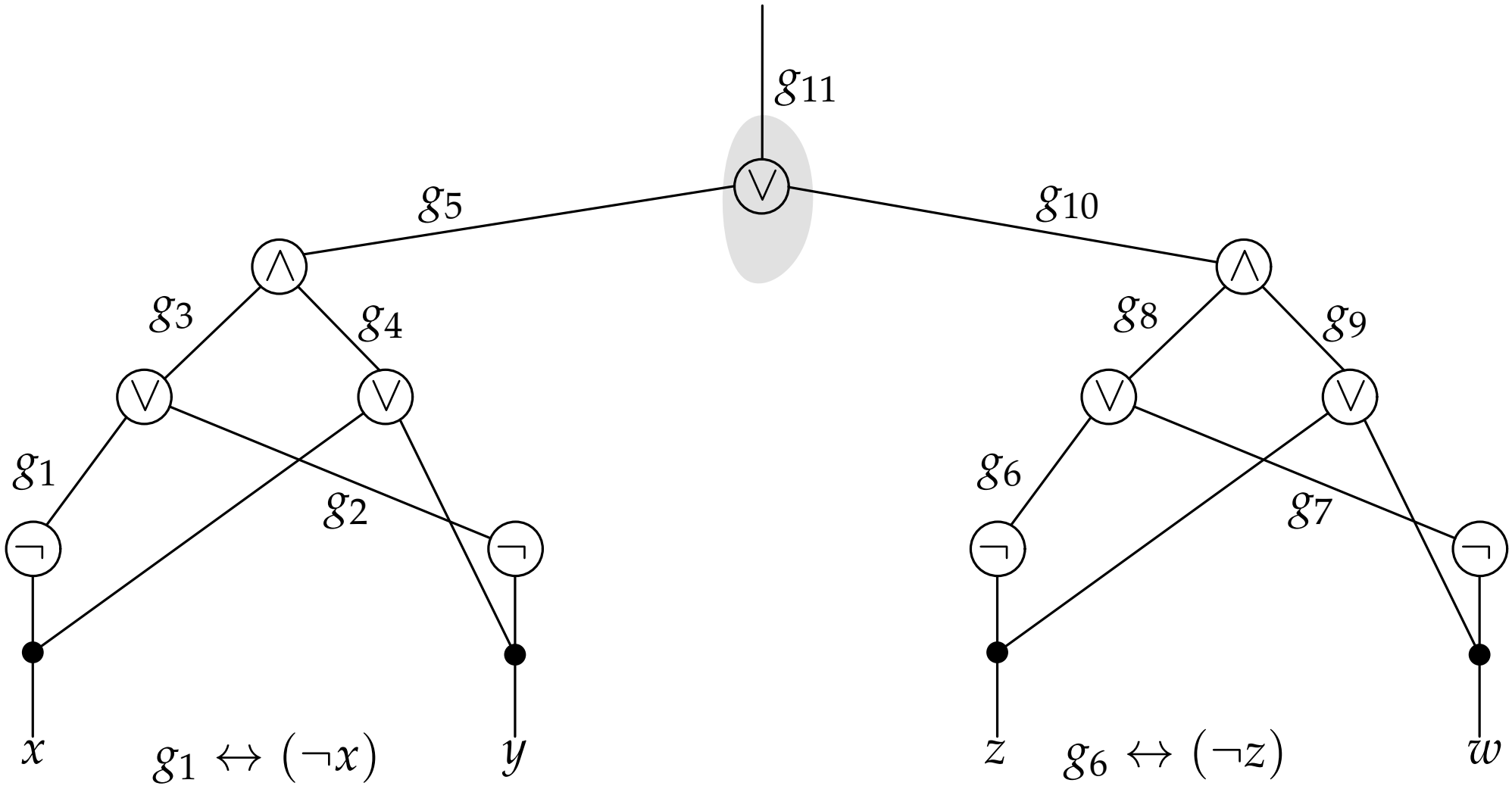
$$g_4 \leftrightarrow (x \vee y)$$

$$g_5 \leftrightarrow (g_3 \wedge g_4)$$



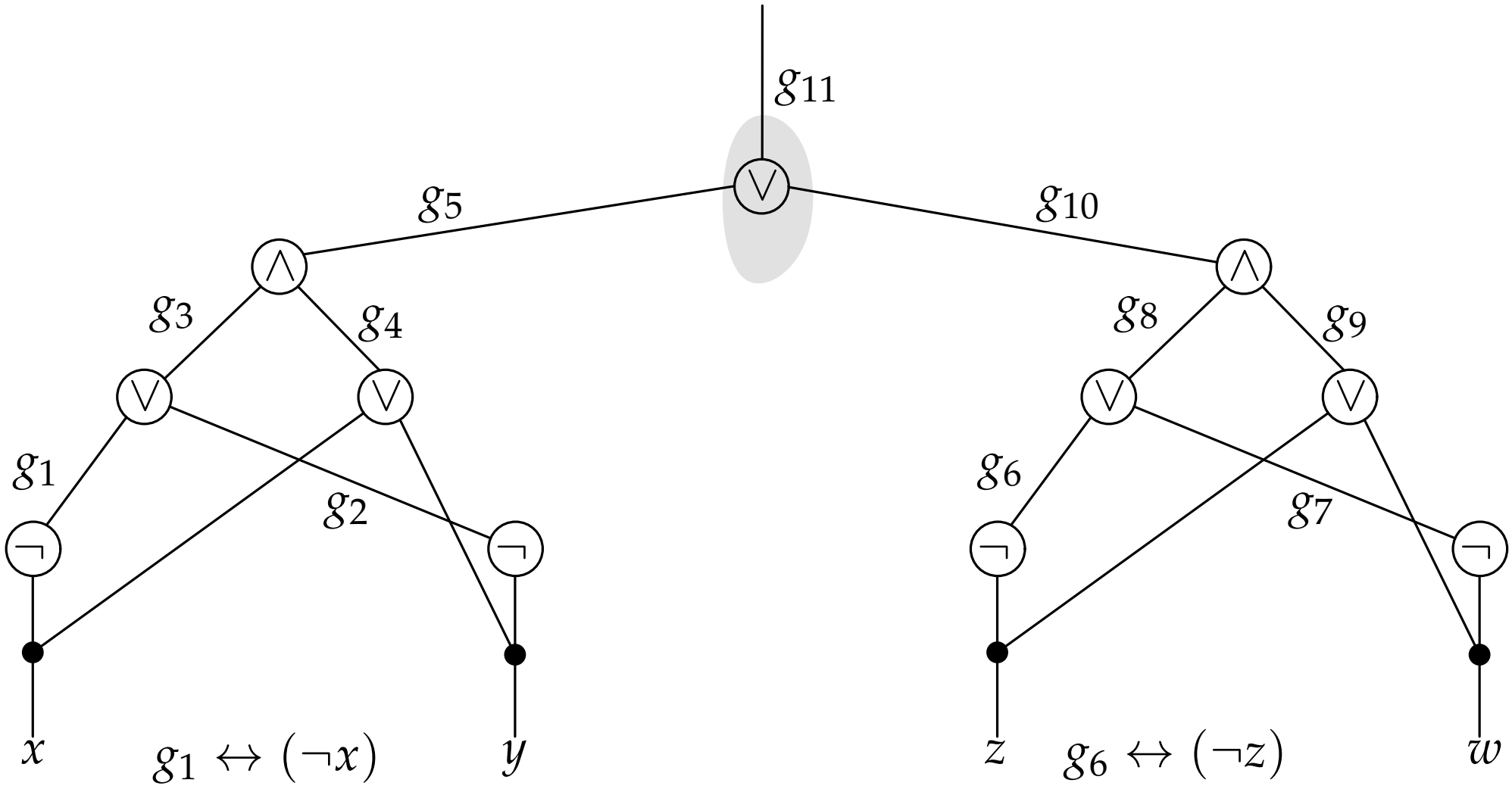
$$g_1 \leftrightarrow (\neg x)$$
$$g_2 \leftrightarrow (\neg y)$$
$$g_3 \leftrightarrow (g_1 \vee g_2)$$
$$g_4 \leftrightarrow (x \vee y)$$
$$g_5 \leftrightarrow (g_3 \wedge g_4)$$

$$g_6 \leftrightarrow (\neg z)$$
$$g_7 \leftrightarrow (\neg w)$$
$$g_8 \leftrightarrow (g_6 \vee g_7)$$
$$g_9 \leftrightarrow (z \vee w)$$
$$g_{10} \leftrightarrow (g_8 \wedge g_9)$$



$$g_1 \leftrightarrow (\neg x)$$
$$g_2 \leftrightarrow (\neg y)$$
$$g_3 \leftrightarrow (g_1 \vee g_2)$$
$$g_4 \leftrightarrow (x \vee y)$$
$$g_5 \leftrightarrow (g_3 \wedge g_4)$$

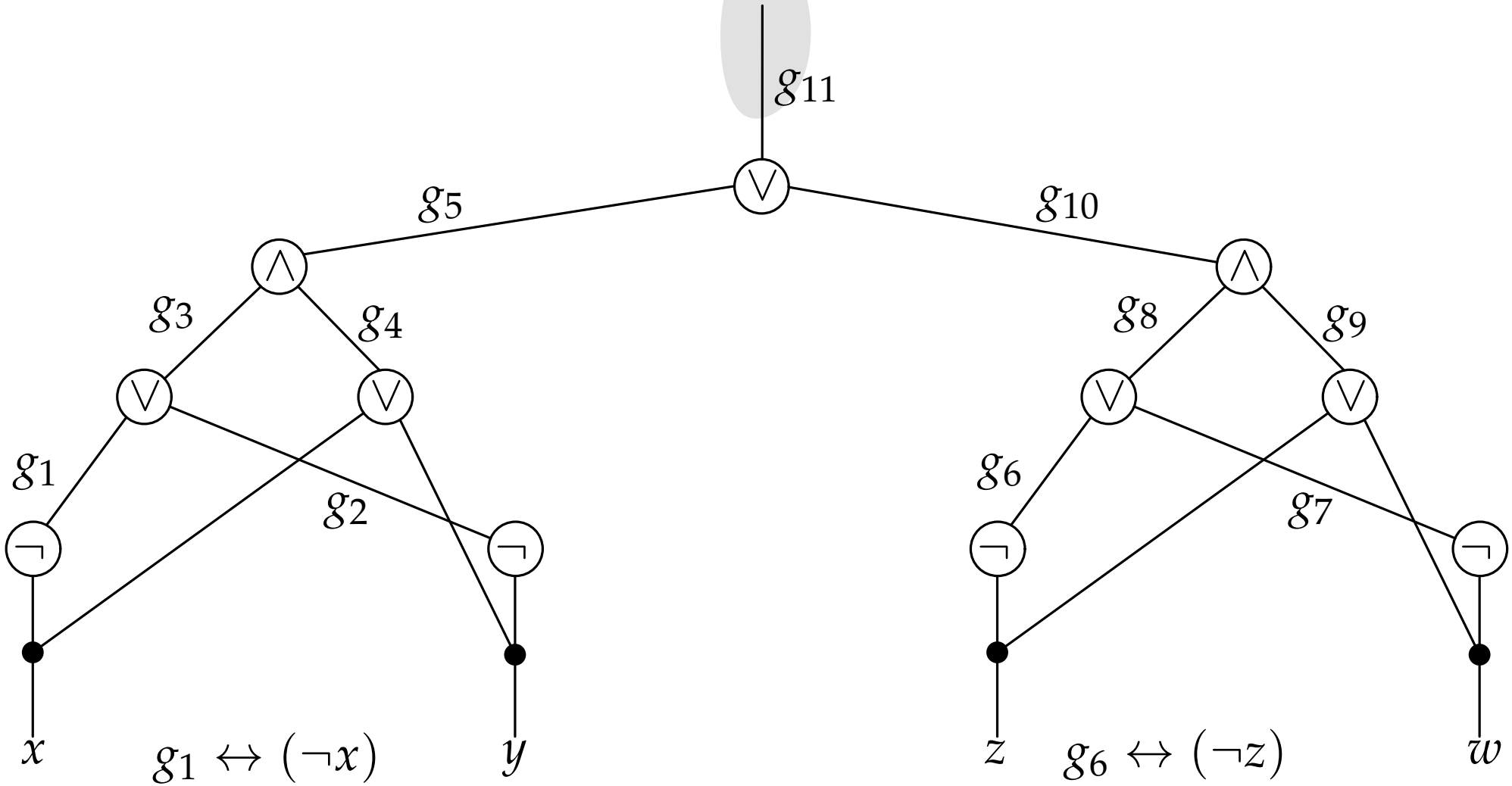
$$g_6 \leftrightarrow (\neg z)$$
$$g_7 \leftrightarrow (\neg w)$$
$$g_8 \leftrightarrow (g_6 \vee g_7)$$
$$g_9 \leftrightarrow (z \vee w)$$
$$g_{10} \leftrightarrow (g_8 \wedge g_9)$$



$$g_1 \leftrightarrow (\neg x)$$
$$g_2 \leftrightarrow (\neg y)$$
$$g_3 \leftrightarrow (g_1 \vee g_2)$$
$$g_4 \leftrightarrow (x \vee y)$$
$$g_5 \leftrightarrow (g_3 \wedge g_4)$$

$$g_{11} \leftrightarrow (g_5 \vee g_{10})$$

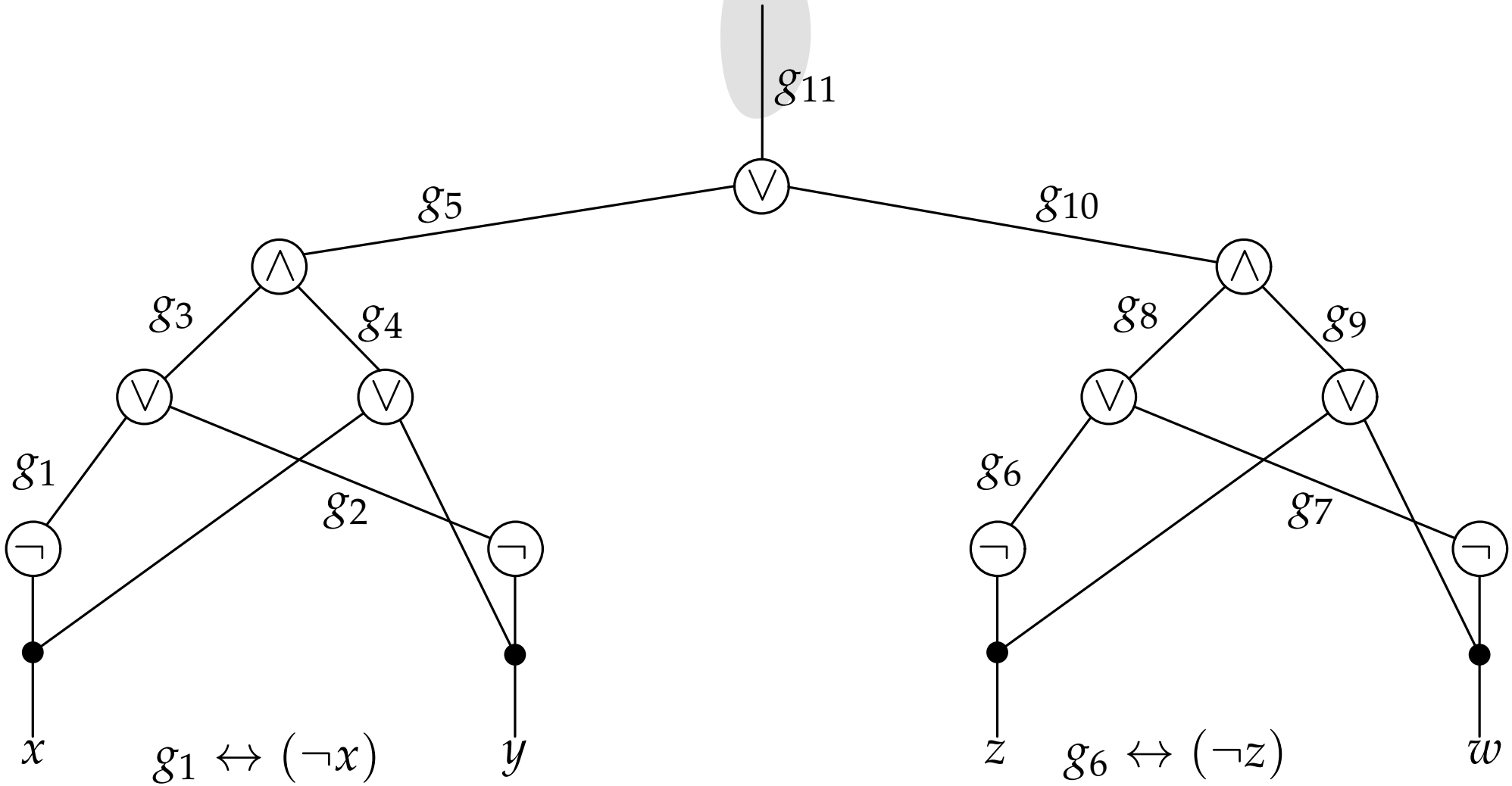
$$g_6 \leftrightarrow (\neg z)$$
$$g_7 \leftrightarrow (\neg w)$$
$$g_8 \leftrightarrow (g_6 \vee g_7)$$
$$g_9 \leftrightarrow (z \vee w)$$
$$g_{10} \leftrightarrow (g_8 \wedge g_9)$$



$$g_1 \leftrightarrow (\neg x)$$
$$g_2 \leftrightarrow (\neg y)$$
$$g_3 \leftrightarrow (g_1 \vee g_2)$$
$$g_4 \leftrightarrow (x \vee y)$$
$$g_5 \leftrightarrow (g_3 \wedge g_4)$$

$$g_{11} \leftrightarrow (g_5 \vee g_{10})$$

$$g_6 \leftrightarrow (\neg z)$$
$$g_7 \leftrightarrow (\neg w)$$
$$g_8 \leftrightarrow (g_6 \vee g_7)$$
$$g_9 \leftrightarrow (z \vee w)$$
$$g_{10} \leftrightarrow (g_8 \wedge g_9)$$

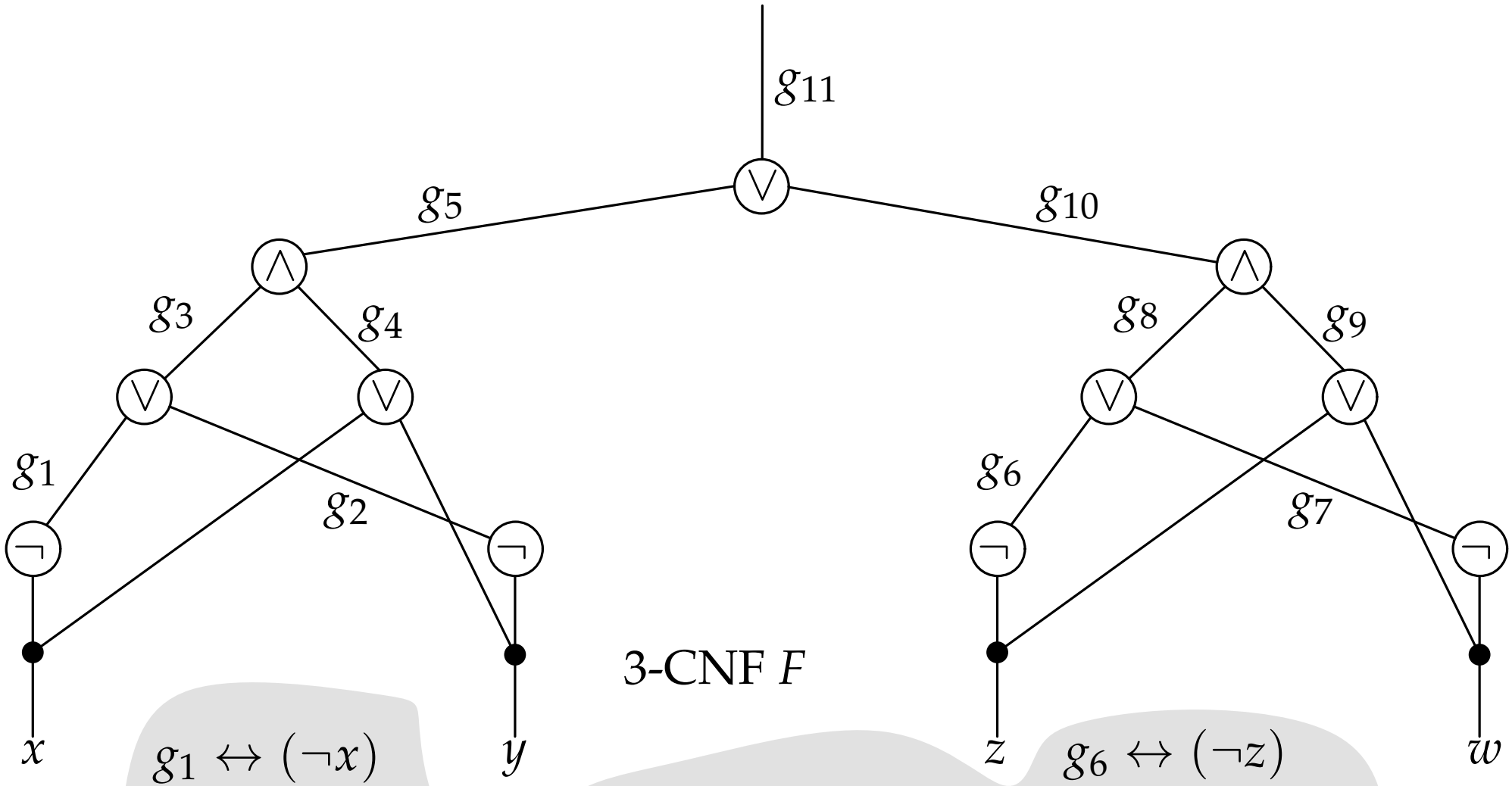


$$g_1 \leftrightarrow (\neg x)$$
$$g_2 \leftrightarrow (\neg y)$$
$$g_3 \leftrightarrow (g_1 \vee g_2)$$
$$g_4 \leftrightarrow (x \vee y)$$
$$g_5 \leftrightarrow (g_3 \wedge g_4)$$

$$g_{11} \leftrightarrow (g_5 \vee g_{10})$$

$$(g_{11})$$

$$g_6 \leftrightarrow (\neg z)$$
$$g_7 \leftrightarrow (\neg w)$$
$$g_8 \leftrightarrow (g_6 \vee g_7)$$
$$g_9 \leftrightarrow (z \vee w)$$
$$g_{10} \leftrightarrow (g_8 \wedge g_9)$$

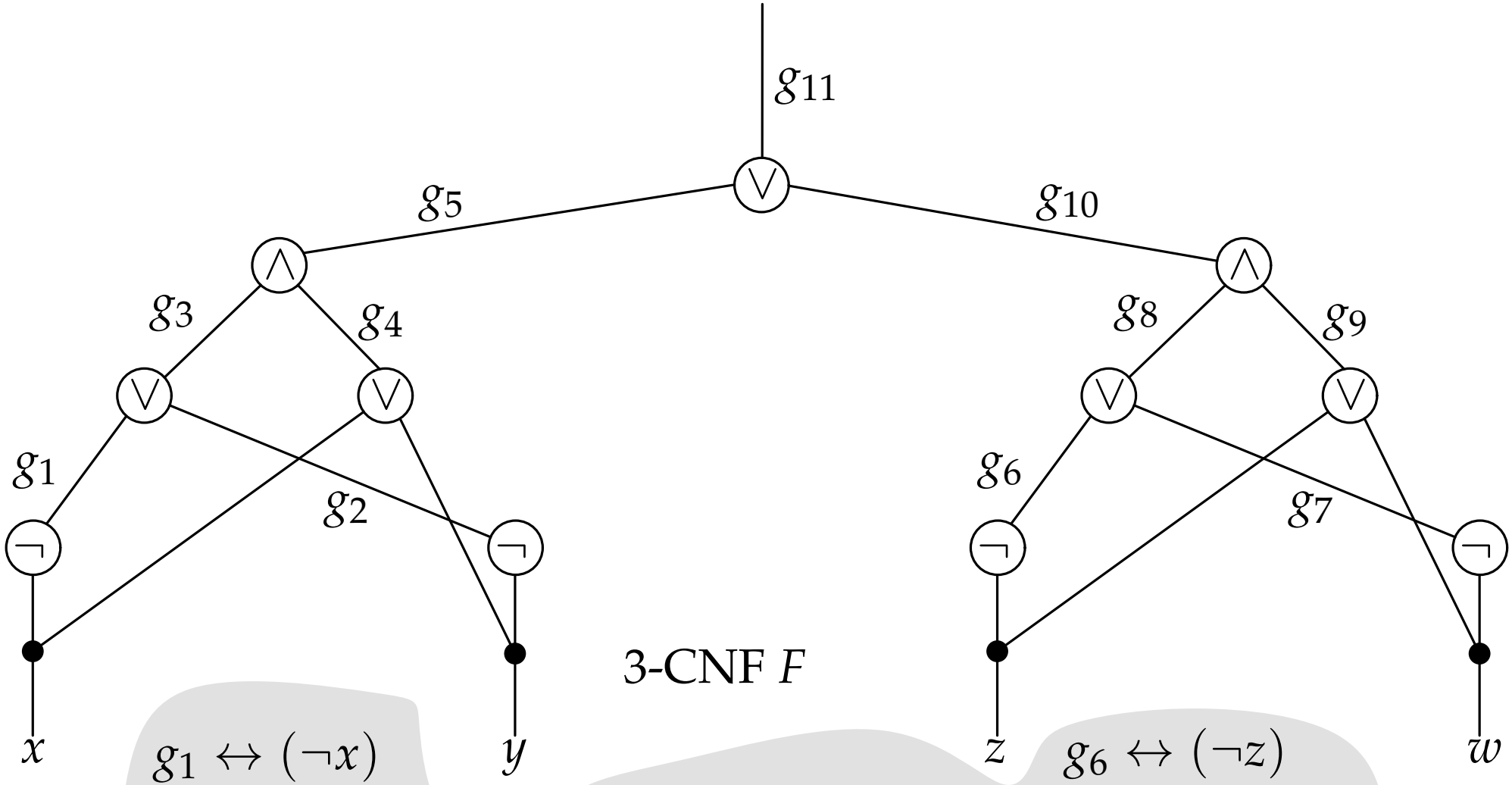


3-CNF F

$$g_1 \leftrightarrow (\neg x)$$
$$g_2 \leftrightarrow (\neg y)$$
$$g_3 \leftrightarrow (g_1 \vee g_2)$$
$$g_4 \leftrightarrow (x \vee y)$$
$$g_5 \leftrightarrow (g_3 \wedge g_4)$$

$$g_{11} \leftrightarrow (g_5 \vee g_{10})$$
$$(g_{11})$$

$$g_6 \leftrightarrow (\neg z)$$
$$g_7 \leftrightarrow (\neg w)$$
$$g_8 \leftrightarrow (g_6 \vee g_7)$$
$$g_9 \leftrightarrow (z \vee w)$$
$$g_{10} \leftrightarrow (g_8 \wedge g_9)$$



3-CNF F

$$g_1 \leftrightarrow (\neg x)$$
$$g_2 \leftrightarrow (\neg y)$$
$$g_3 \leftrightarrow (g_1 \vee g_2)$$
$$g_4 \leftrightarrow (x \vee y)$$
$$g_5 \leftrightarrow (g_3 \wedge g_4)$$

$$g_{11} \leftrightarrow (g_5 \vee g_{10})$$
$$(g_{11})$$
$$C(x,y,z,w) = 1$$
$$\Longleftrightarrow$$
$$\exists \vec{g} : F(x,y,z,w,\vec{g}) = 1$$

$$g_6 \leftrightarrow (\neg z)$$
$$g_7 \leftrightarrow (\neg w)$$
$$g_8 \leftrightarrow (g_6 \vee g_7)$$
$$g_9 \leftrightarrow (z \vee w)$$
$$g_{10} \leftrightarrow (g_8 \wedge g_9)$$

CIRCUIT-SAT to 3-SAT

$$C(x, y, z, w) = 1$$
$$\Longleftrightarrow$$
$$\exists \vec{g} : F(x, y, z, w, \vec{g}) = 1$$

CIRCUIT-SAT to 3-SAT

can solve 3-SAT $\Longrightarrow$ can solve CIRCUIT-SAT

$$C(x, y, z, w) = 1$$
$$\Longleftrightarrow$$
$$\exists \vec{g} : F(x, y, z, w, \vec{g}) = 1$$

CIRCUIT-SAT to 3-SAT

can solve 3-SAT $\Longrightarrow$ can solve CIRCUIT-SAT

Theorem (Cook-Levin). 3-SAT is NP-complete.

$$C(x, y, z, w) = 1$$
$$\Longleftrightarrow$$
$$\exists \vec{g} : F(x, y, z, w, \vec{g}) = 1$$

Max-SAT

$$(x \vee y) \quad (x \vee z) \quad (\bar{x} \vee \bar{y}) \quad (\bar{x} \vee \bar{z}) \quad (y \vee z) \quad (x \vee \bar{y} \vee \bar{z})$$

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	

Max-SAT

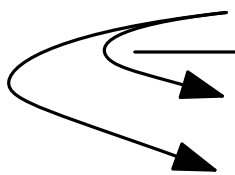
32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

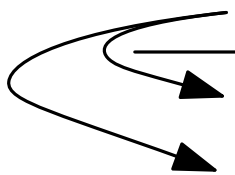
x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	



Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

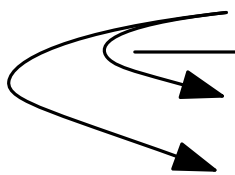
x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	



Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

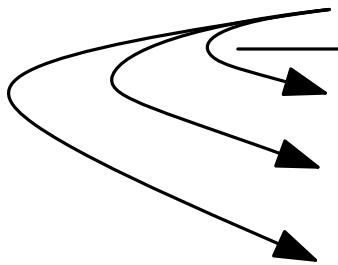
x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	55



Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

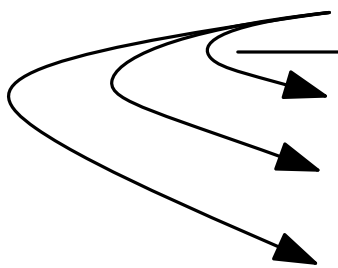
x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	55
1	0	1	



Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

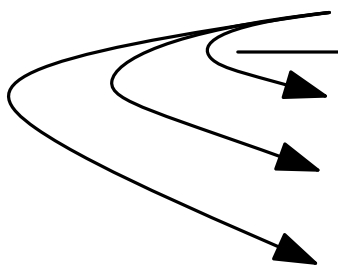
x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	55
1	0	1	



Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	55
1	0	1	59



Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	55
1	0	1	59

local optimum

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	55
1	0	1	59
0	1	1	

local optimum

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

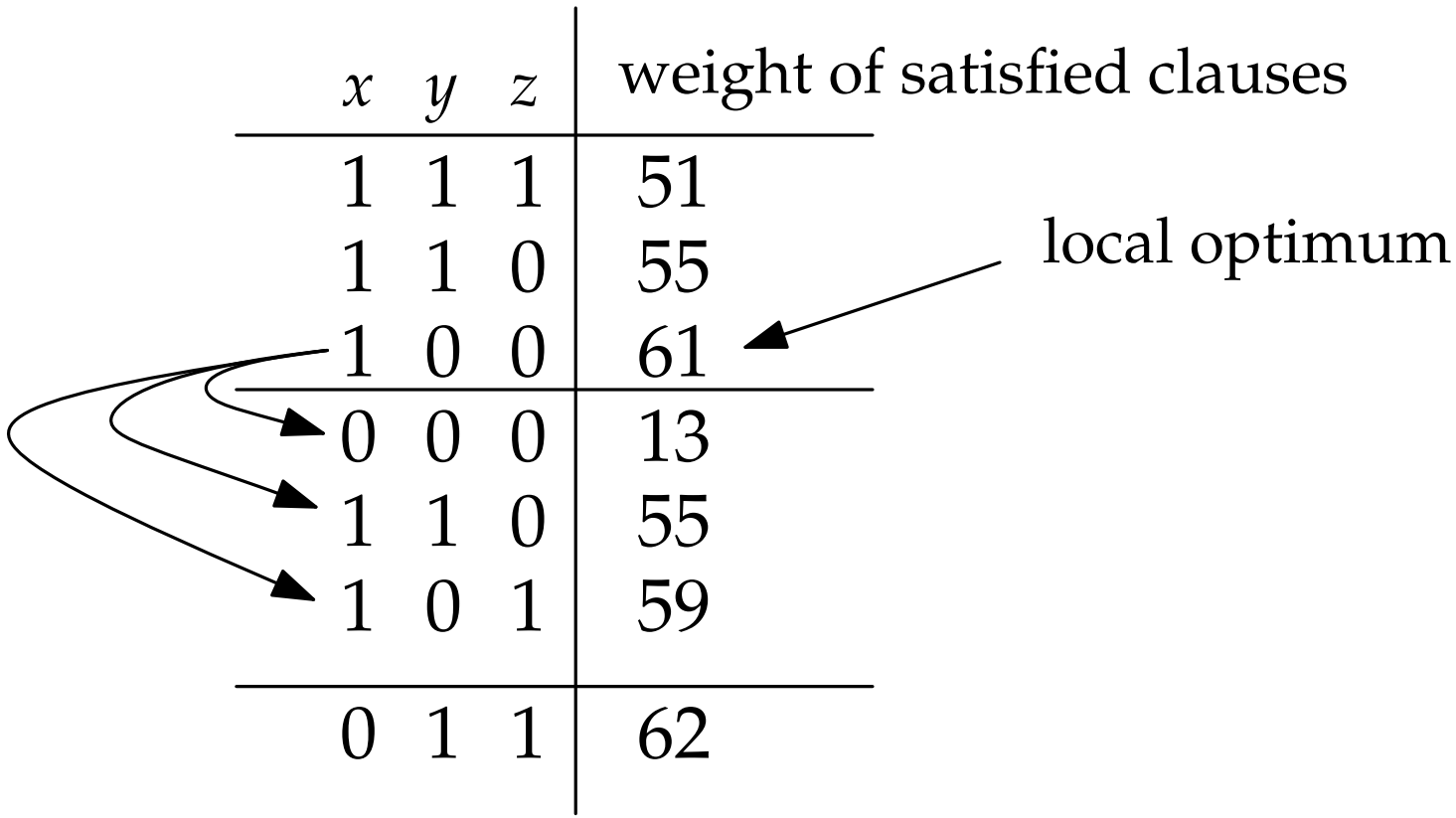
x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	55
1	0	1	59
0	1	1	

local optimum

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	55
1	0	1	59
0	1	1	62



Max-SAT

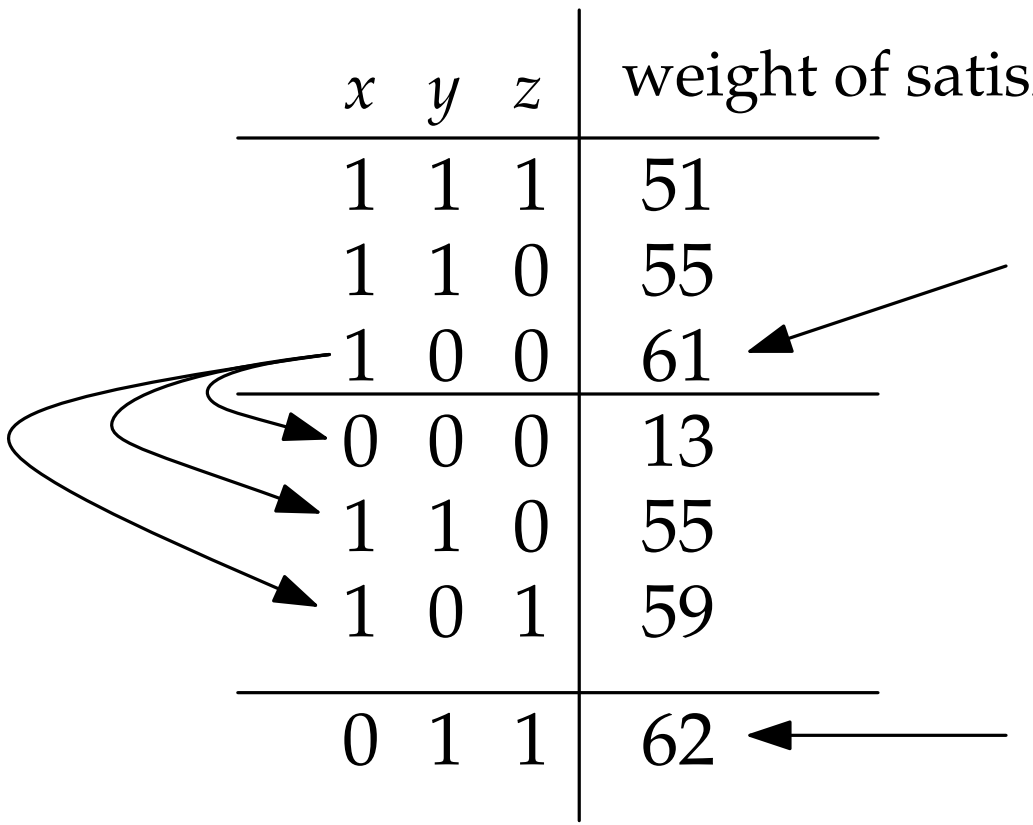
32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses	
1	1	1	51	
1	1	0	55	
1	0	0	61	local optimum
0	0	0	13	
1	1	0	55	
1	0	1	59	
0	1	1	62	global optimum

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	55
1	0	1	59
0	1	1	62



local optimum

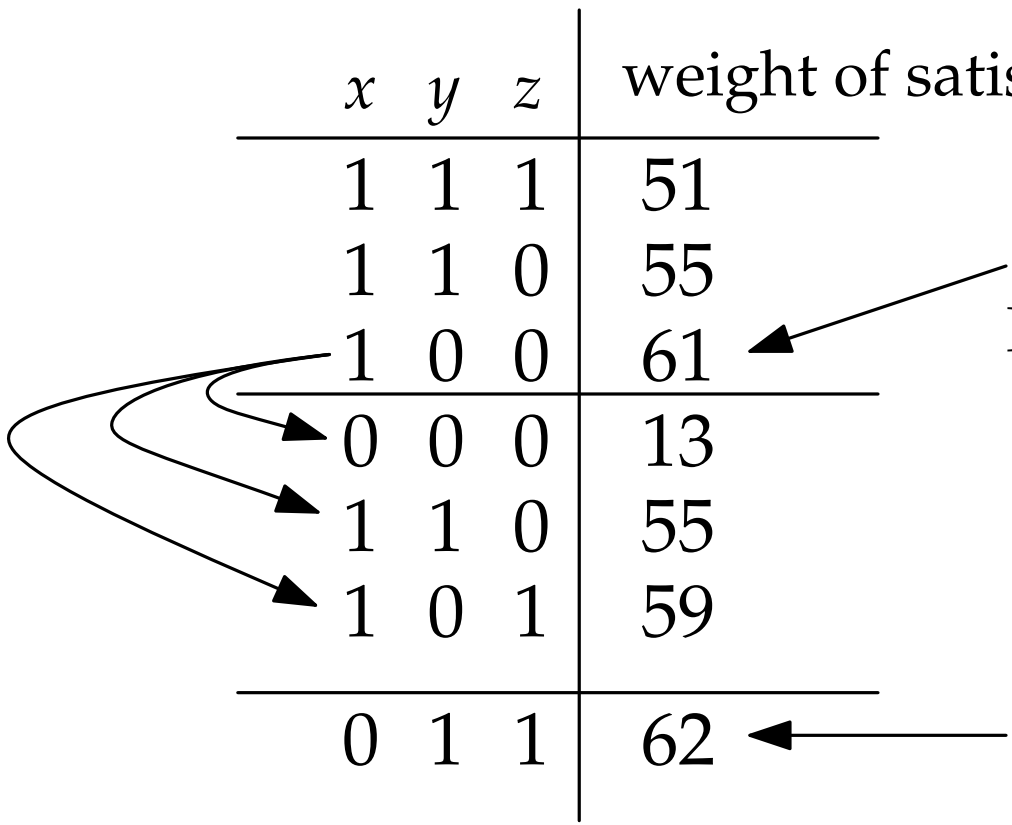
global optimum

NP-hard to find (Max-SAT)

Max-SAT

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

x	y	z	weight of satisfied clauses
1	1	1	51
1	1	0	55
1	0	0	61
0	0	0	13
1	1	0	55
1	0	1	59
0	1	1	62



local optimum

How hard can it be???

global optimum

NP-hard to find (Max-SAT)

Local Max-k-SAT

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Local Max-k-SAT

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Proof. Take an instance C of ITERATECIRCUIT and build a k-CNF-formula F so that "locally optimizing" F mimics iterating the circuit.

Local Max-k-SAT

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Proof. Take an instance C of ITERATECIRCUIT and build a k-CNF-formula F so that "locally optimizing" F mimics iterating the circuit.

Warmup. Given a circuit $C(\vec{x})$ and an input $\vec{a} \in \{0,1\}^m$,

Local Max-k-SAT

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Proof. Take an instance C of ITERATECIRCUIT and build a k-CNF-formula F so that "locally optimizing" F mimics iterating the circuit.

Warmup. Given a circuit $C(\vec{x})$ and an input $\vec{a} \in \{0,1\}^m$, build a formula $F(\vec{g}, \vec{z})$ such that

Local Max-k-SAT

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Proof. Take an instance C of ITERATECIRCUIT and build a k-CNF-formula F so that "locally optimizing" F mimics iterating the circuit.

Warmup. Given a circuit $C(\vec{x})$ and an input $\vec{a} \in \{0,1\}^m$, build a formula $F(\vec{g},\vec{z})$ such that every local optimum for F sets $\vec{z}$ to $C(\vec{a})$.

Local Max-k-SAT

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Proof. Take an instance C of ITERATECIRCUIT and build a k-CNF-formula F so that "locally optimizing" F mimics iterating the circuit.

Warmup. Given a circuit $C(\vec{x})$ and an input $\vec{a} \in \{0,1\}^m$, build a formula $F(\vec{g}, \vec{z})$ such that every local optimum for F sets $\vec{z}$ to $C(\vec{a})$.

In words: finding a local optimum of F implicitly evaluates the circuit C.

Local Max-k-SAT

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

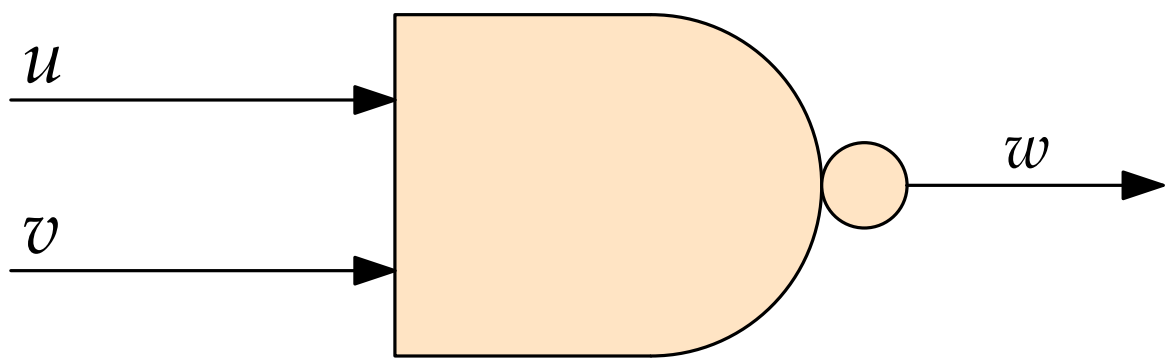
Proof. Take an instance C of ITERATECIRCUIT and build a k-CNF-formula F so that "locally optimizing" F mimics iterating the circuit.

Warmup. Given a circuit $C(\vec{x})$ and an input $\vec{a} \in \{0,1\}^m$, build a formula $F(\vec{g},\vec{z})$ such that every local optimum for F sets $\vec{z}$ to $C(\vec{a})$.

In words: finding a local optimum of F implicitly evaluates the circuit C.

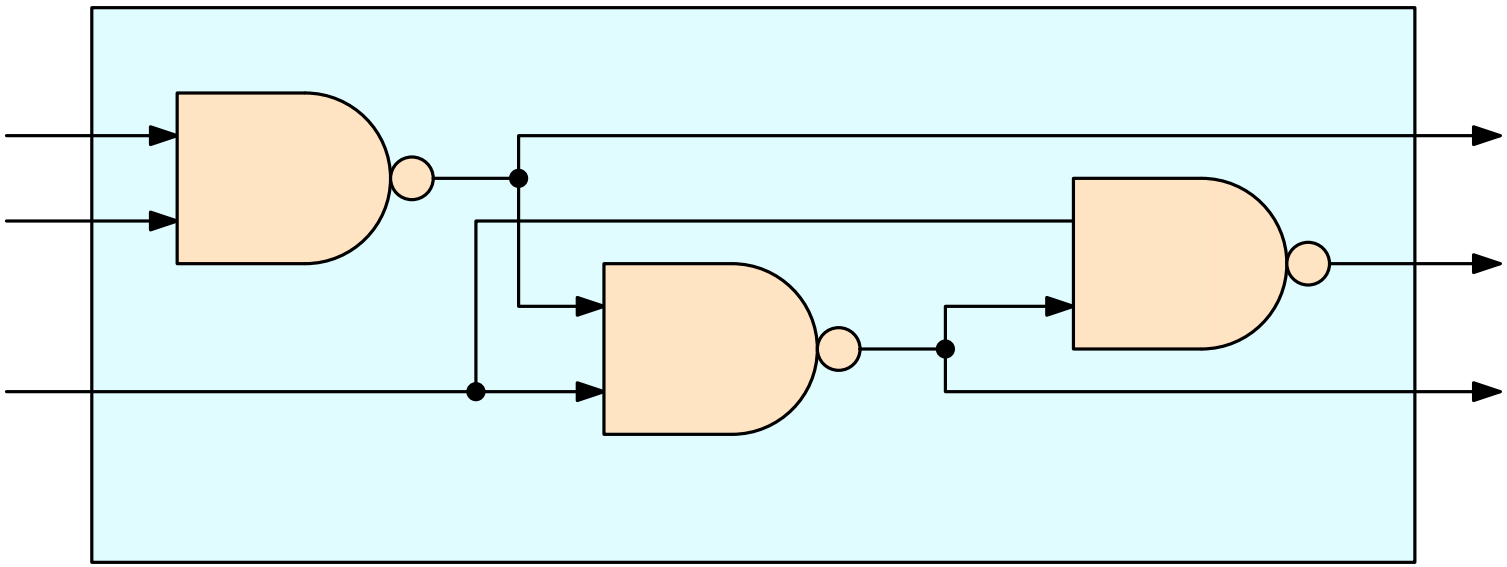
First step: make sure C uses only NAND gates.

This is a NAND gate



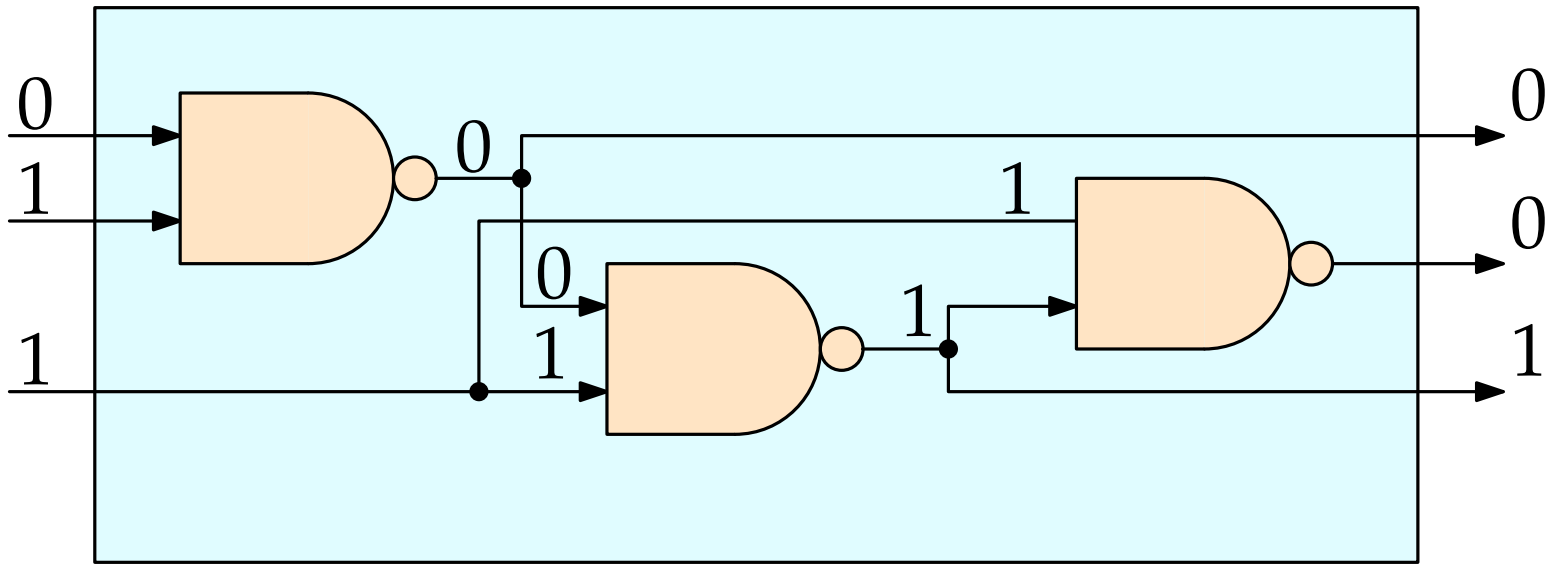
u	v	w
0	0	1
0	1	1
1	0	1
1	1	0

This is a circuit of NAND gates



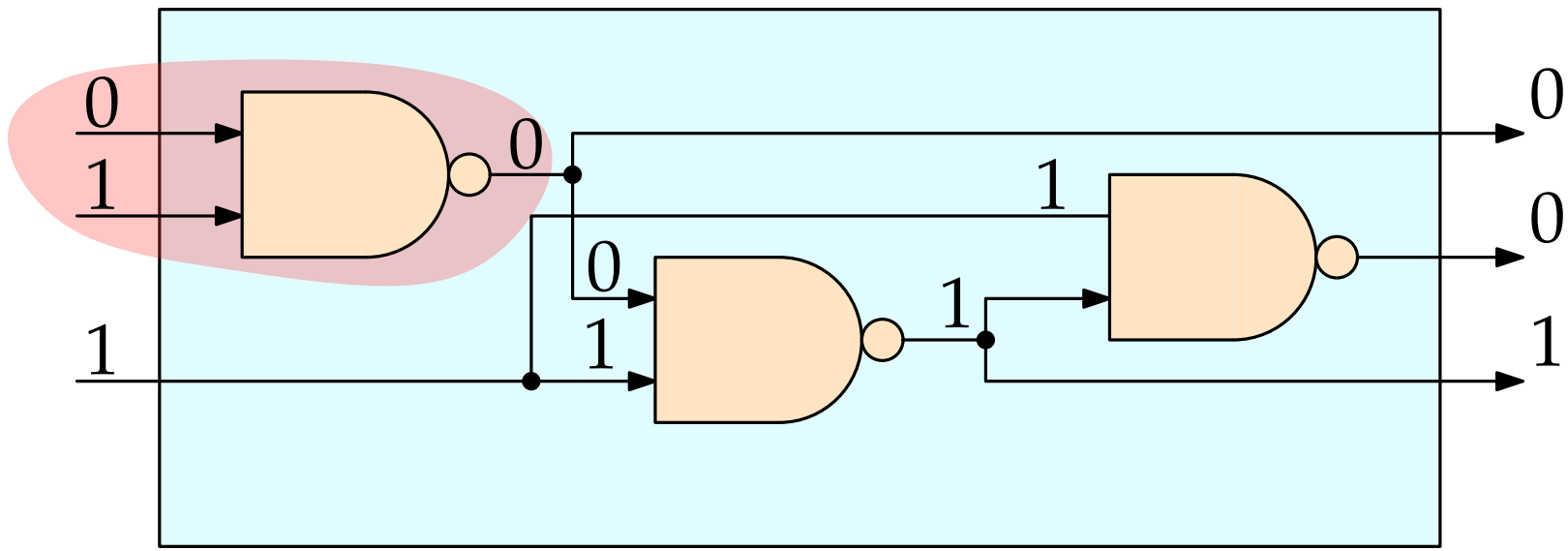
u	v	w
0	0	1
0	1	1
1	0	1
1	1	0

This is a configuration of the circuit



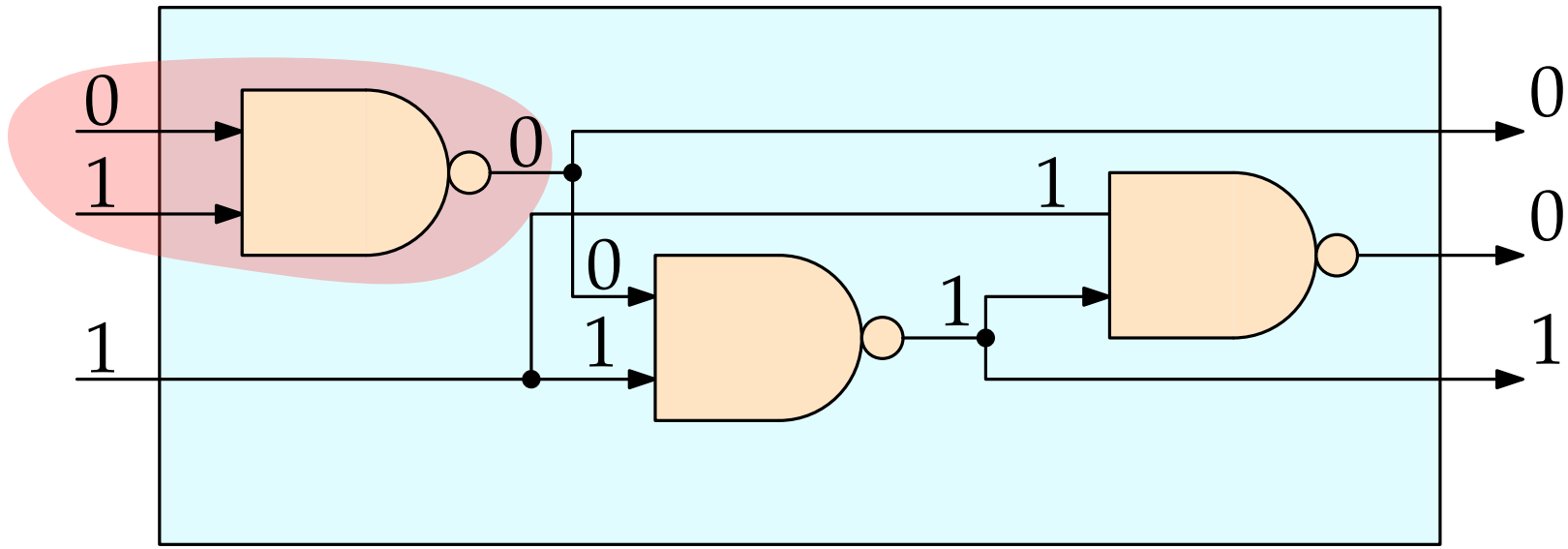
u	v	w
0	0	1
0	1	1
1	0	1
1	1	0

This is a configuration of the circuit
Wait, what?



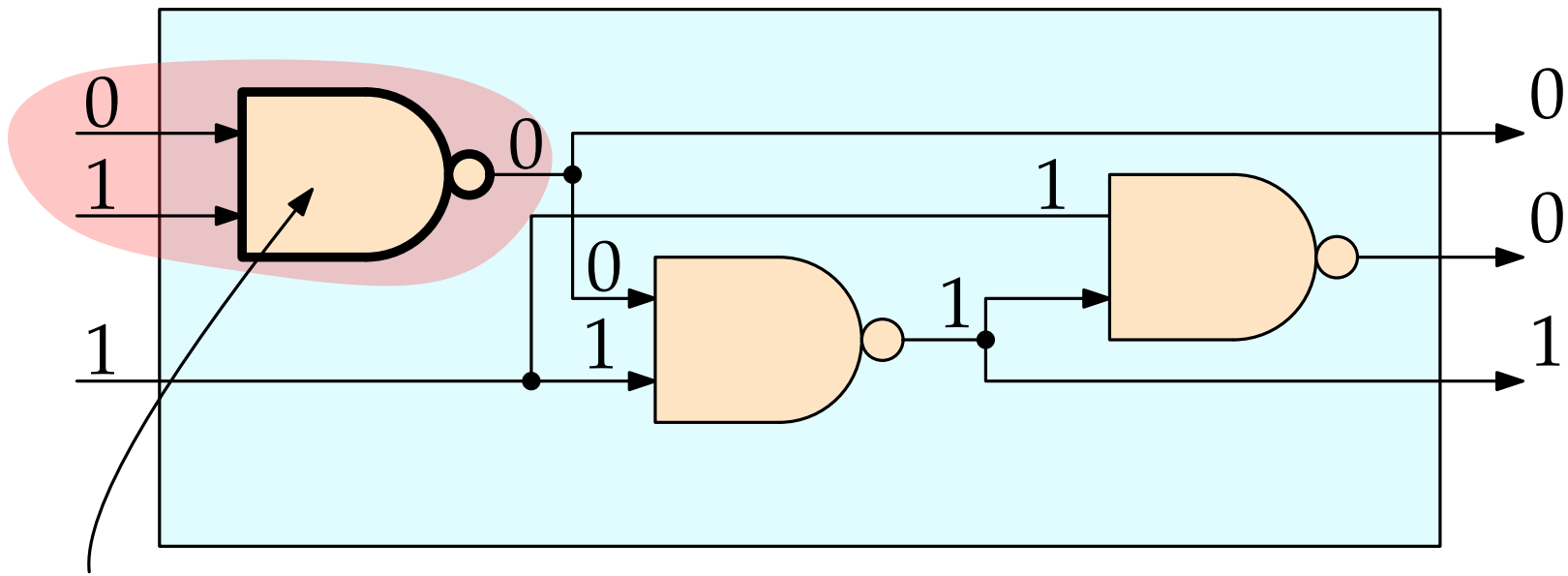
u	v	w
0	0	1
0	1	1
1	0	1
1	1	0

This is a configuration of the circuit
Wait, what?
Yes! We allow gate outputs to be incorrect.



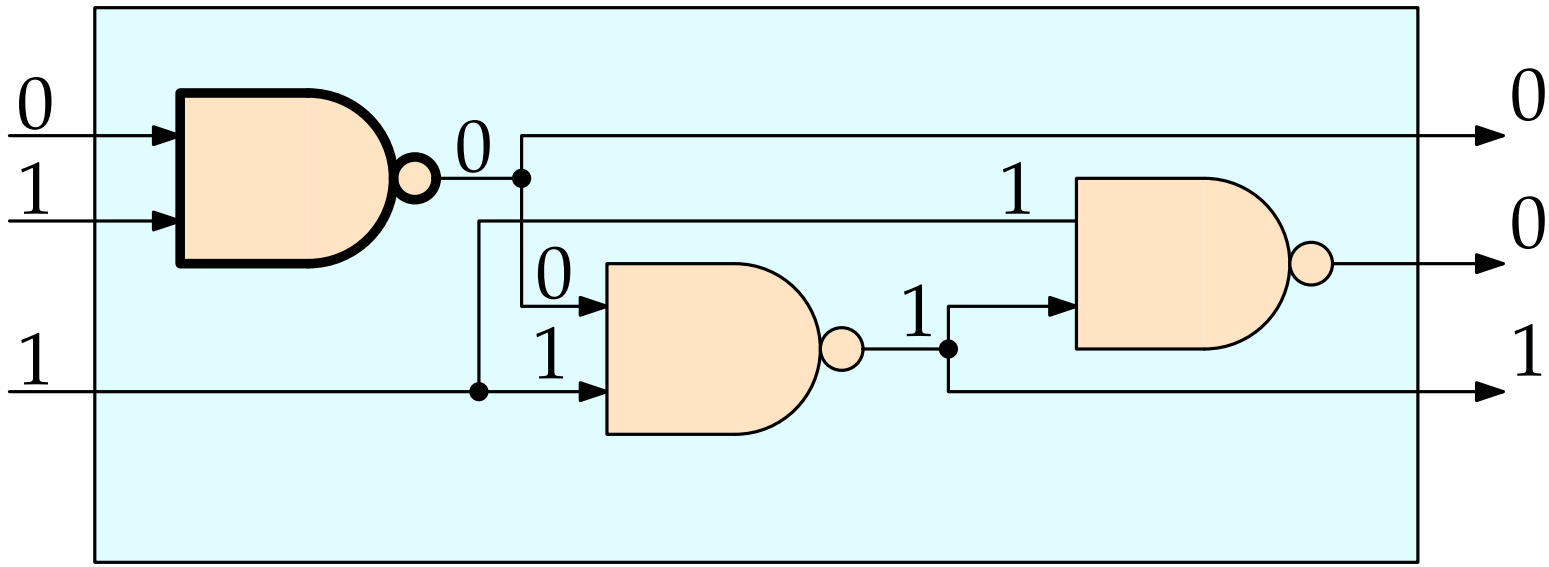
u	v	w
0	0	1
0	1	1
1	0	1
1	1	0

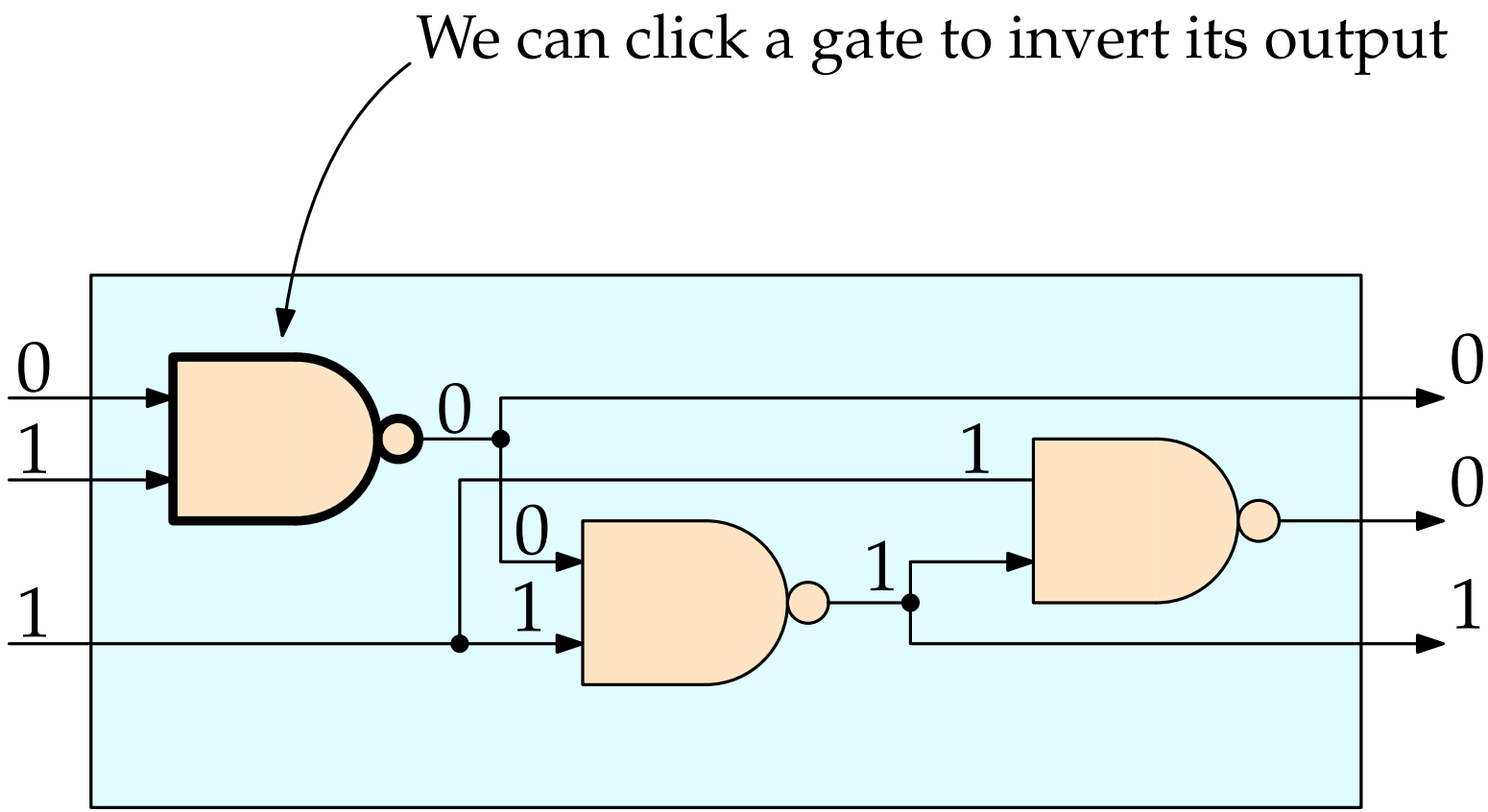
This is a configuration of the circuit
Wait, what?
Yes! We allow gate outputs to be incorrect.

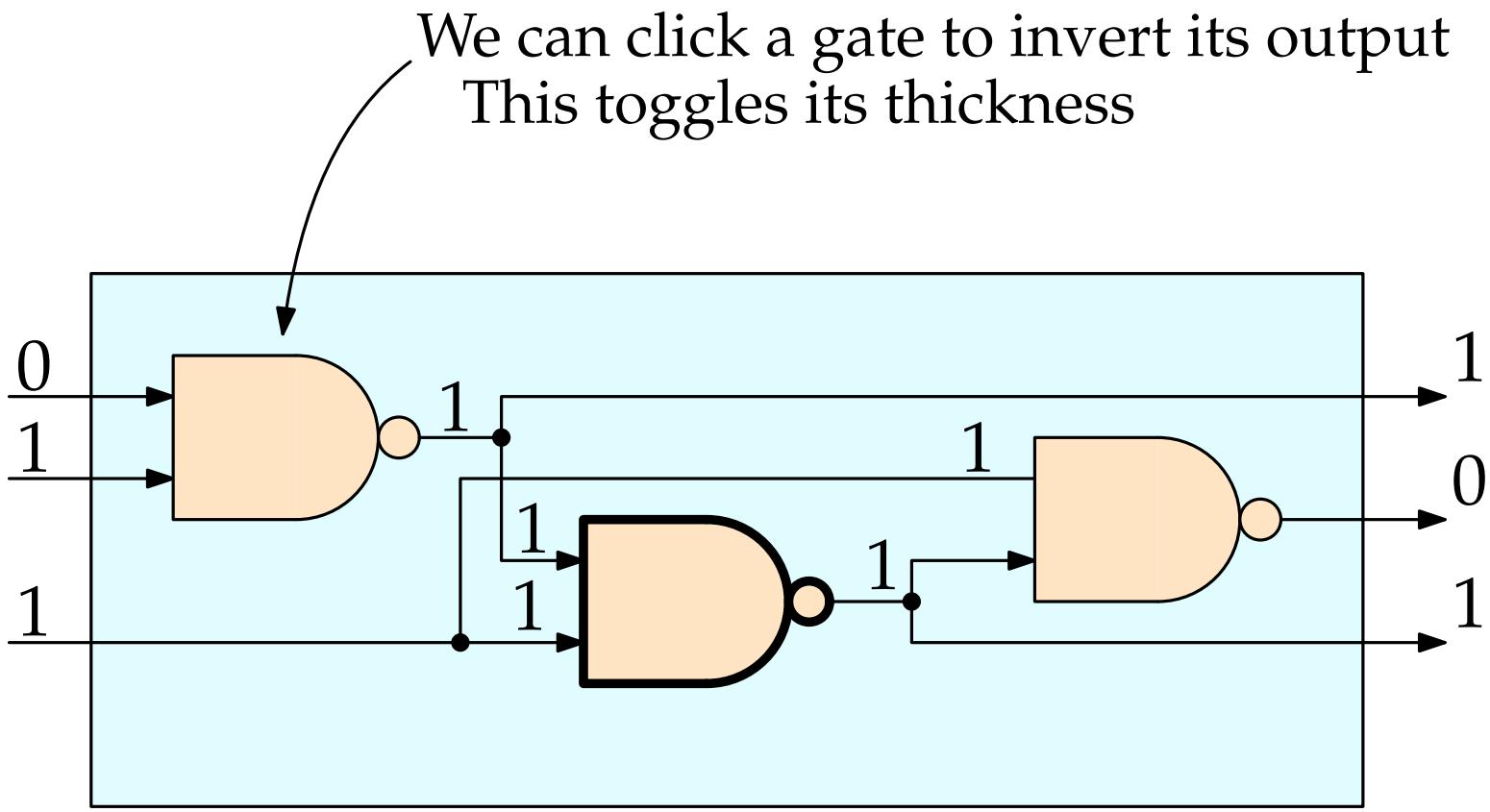


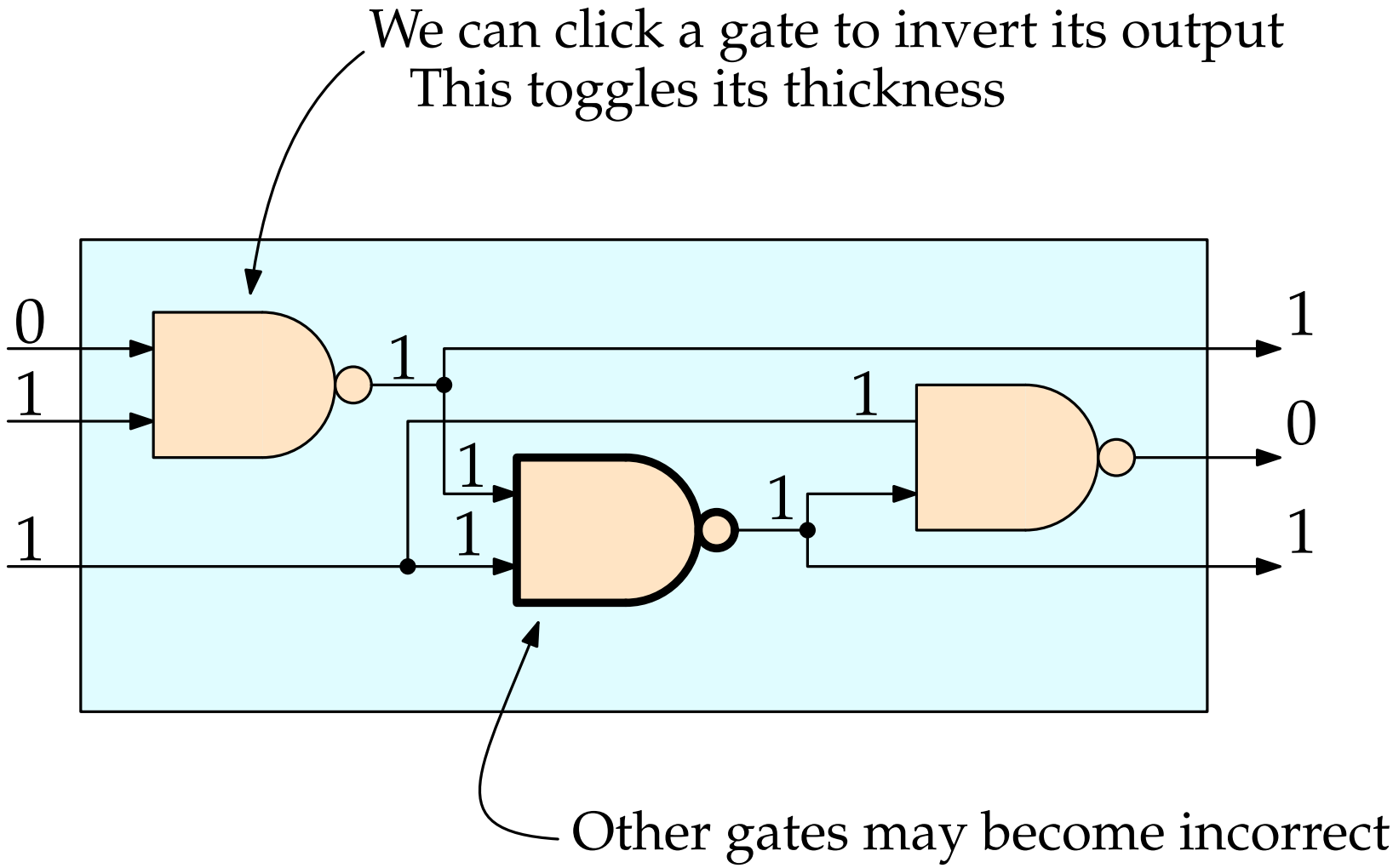
Incorrect gate = thick

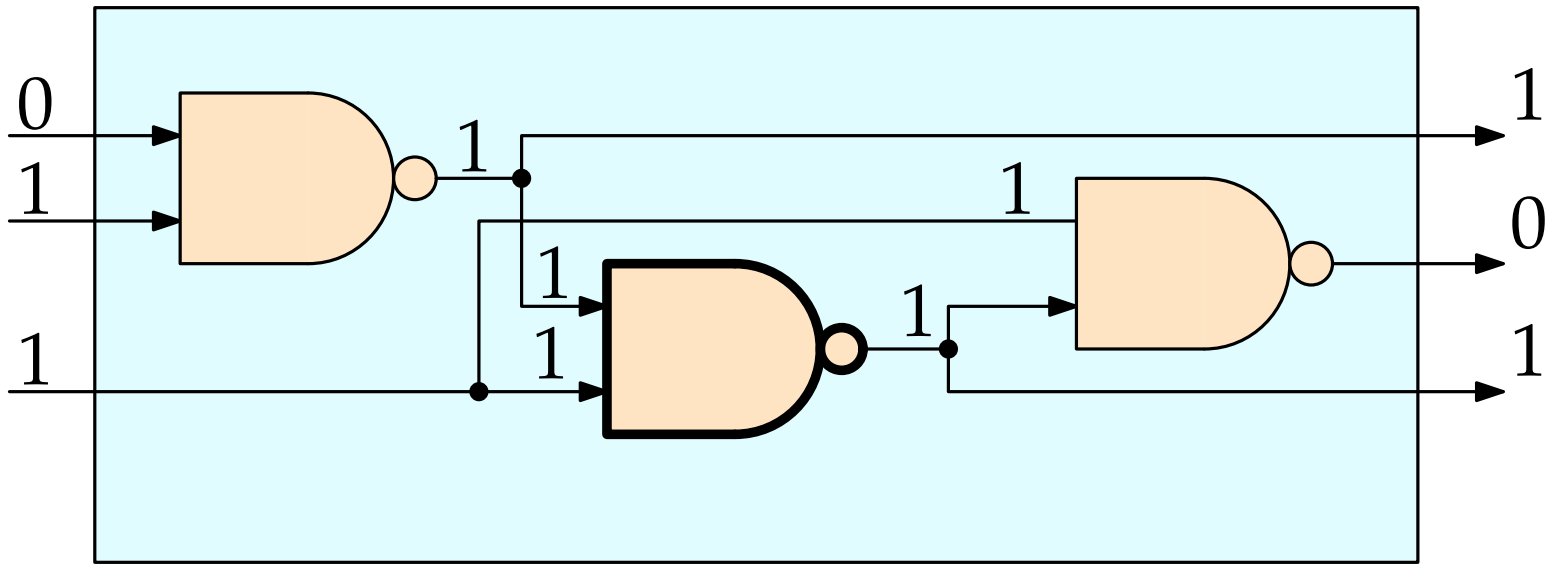
u	v	w
0	0	1
0	1	1
1	0	1
1	1	0

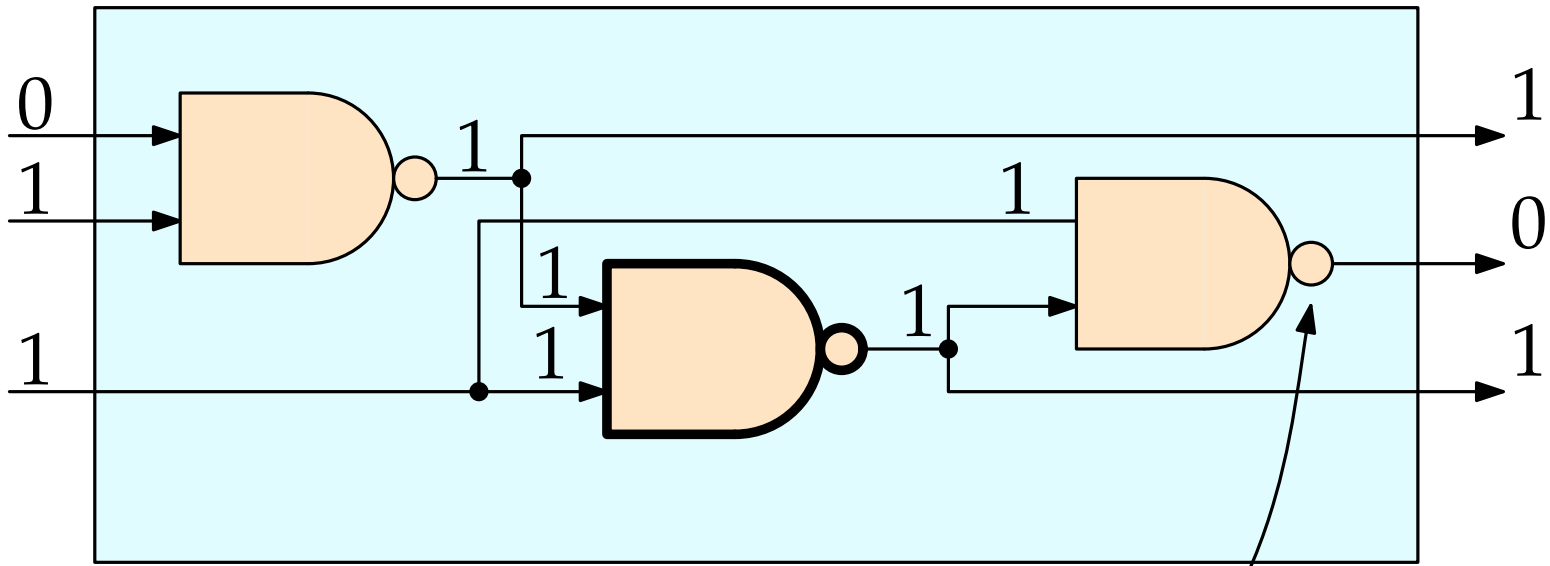

0
1
1
0
0
1
0
1
1
0
0
1


We can click a gate to invert its output
0
1
1
0
0
1
1
0
1
0
1

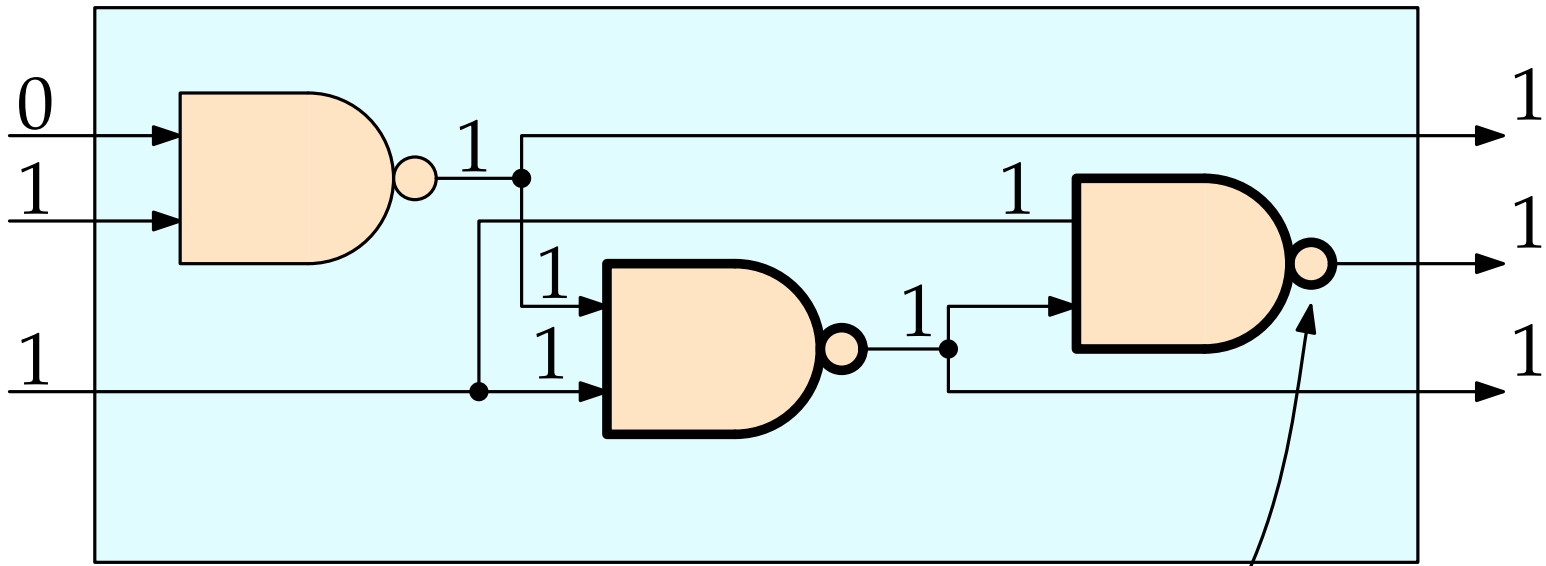

We can click a gate to invert its output
This toggles its thickness
0
1
1
1
1
1
1
1
1
0
1


We can click a gate to invert its output
This toggles its thickness
0
1
1
1
1
1
1
1
1
0
1
Other gates may become incorrect

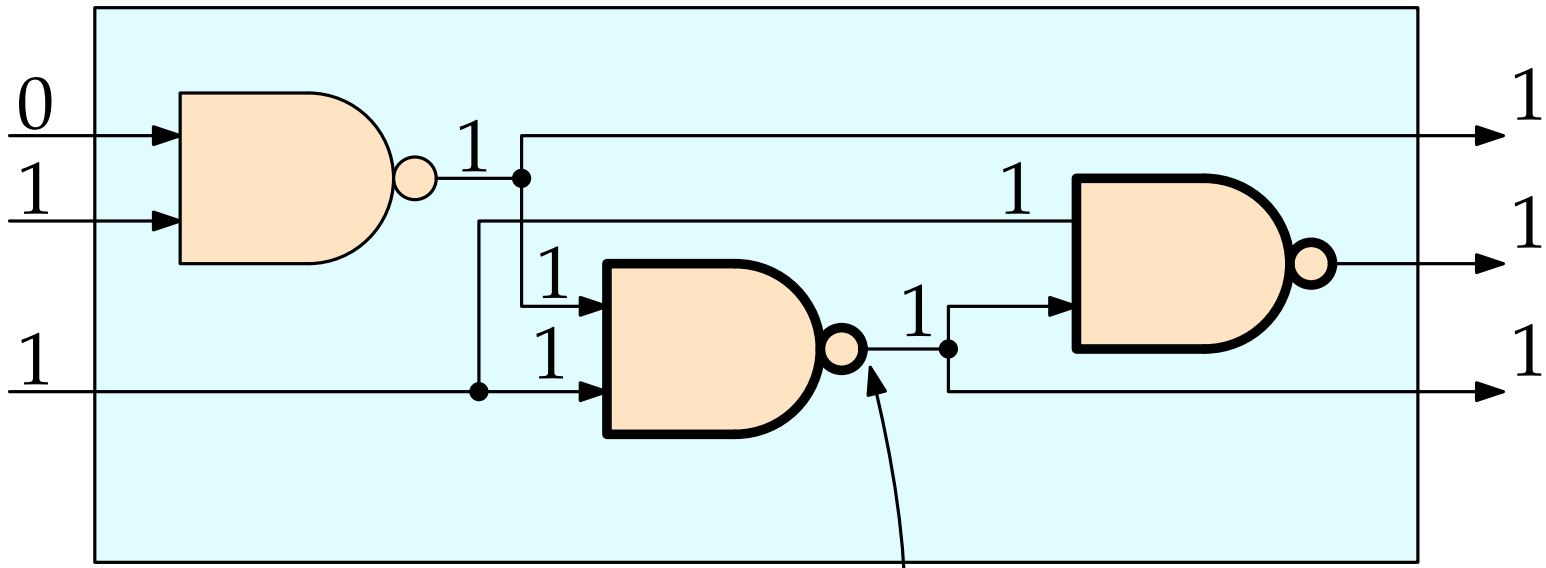

0
1
1
1
1
1
1
1
1
1
0
1



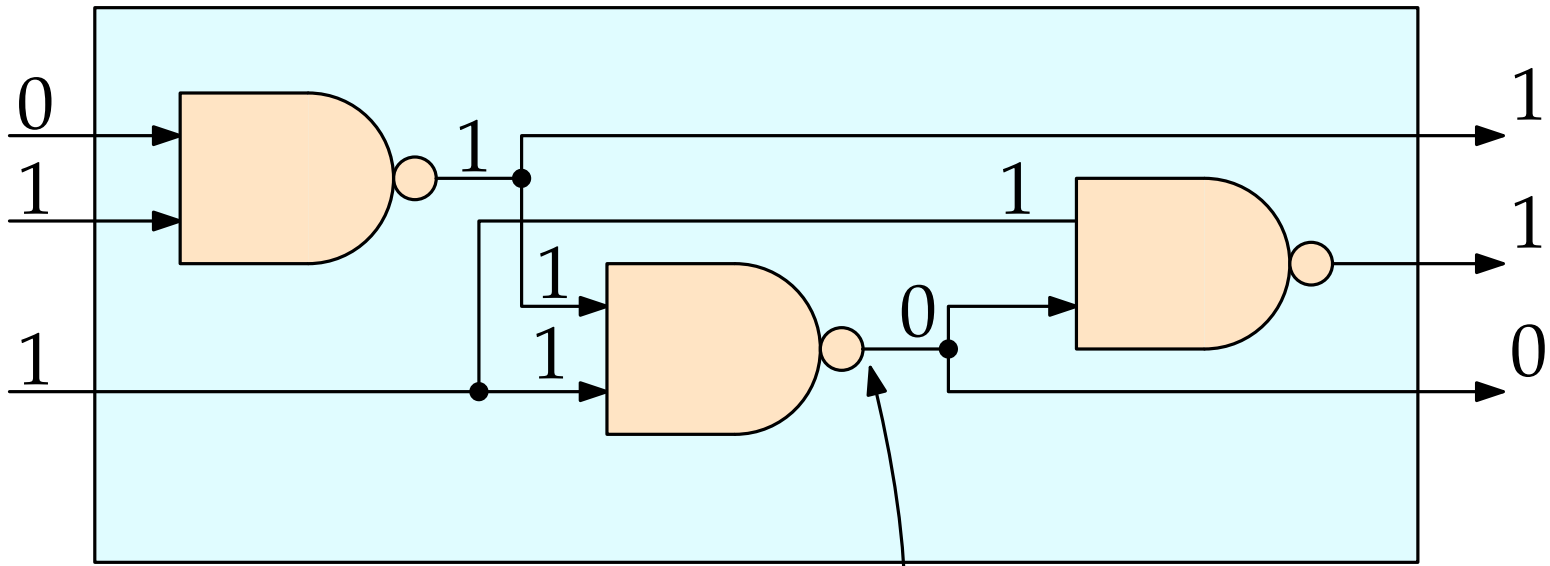
We can also click a gate
that's currently correct



We can also click a gate
that's currently correct

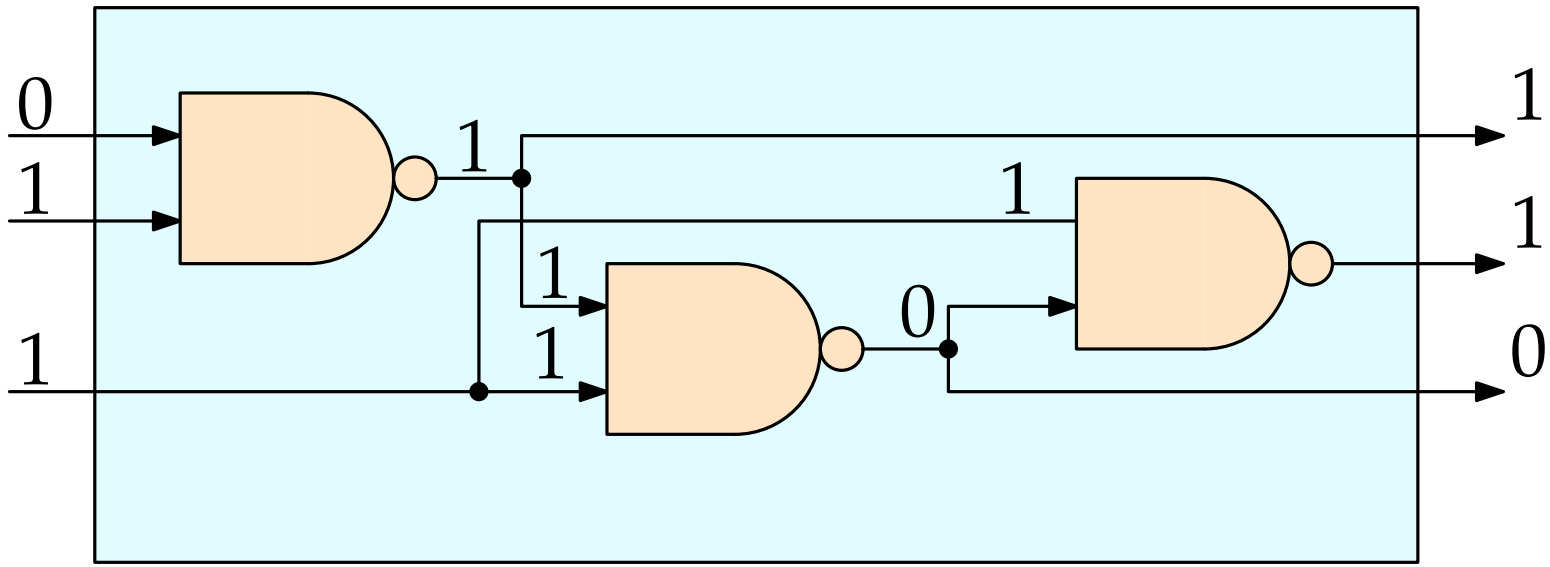


But ultimately we want all gates to be correct, so we click here

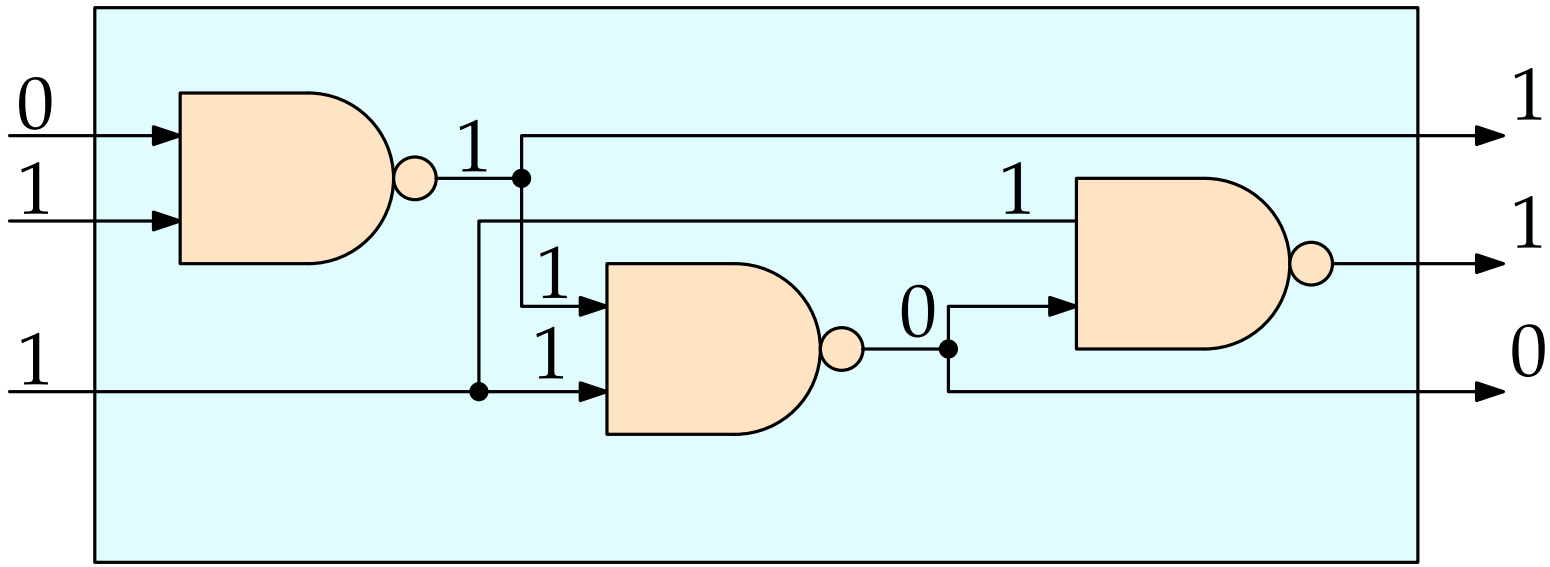


But ultimately we want all gates to be correct, so we click here

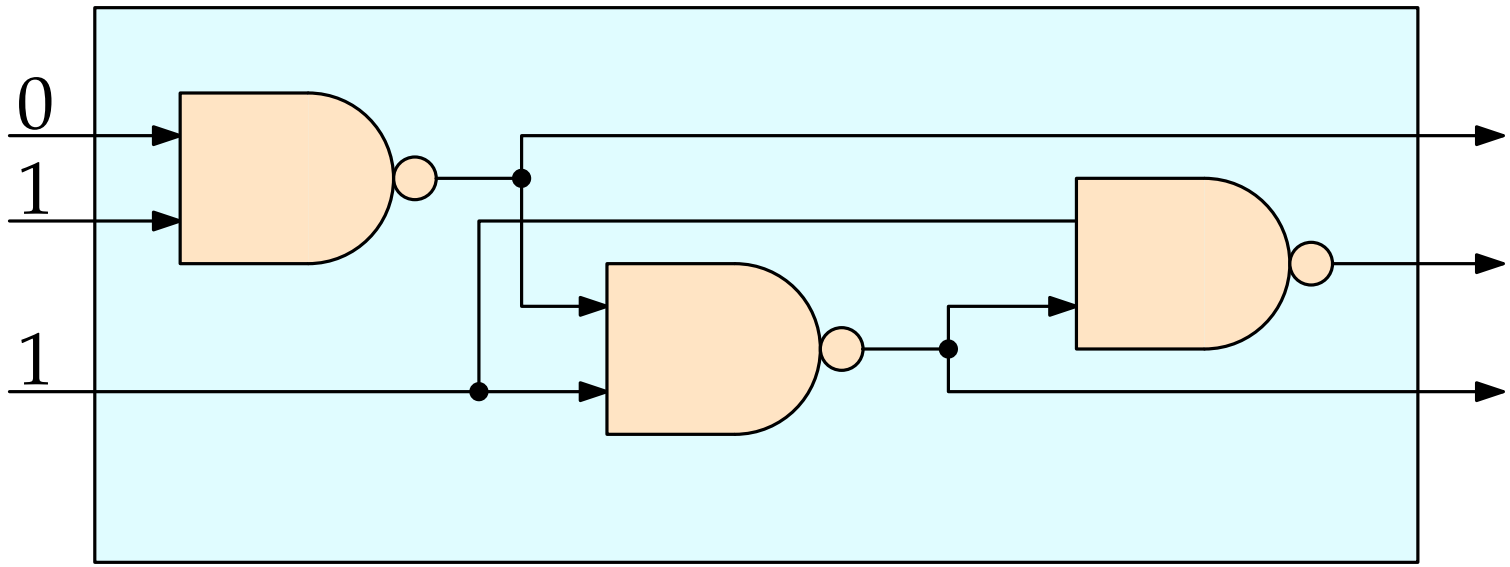
When all gates are correct, then the output is correct.

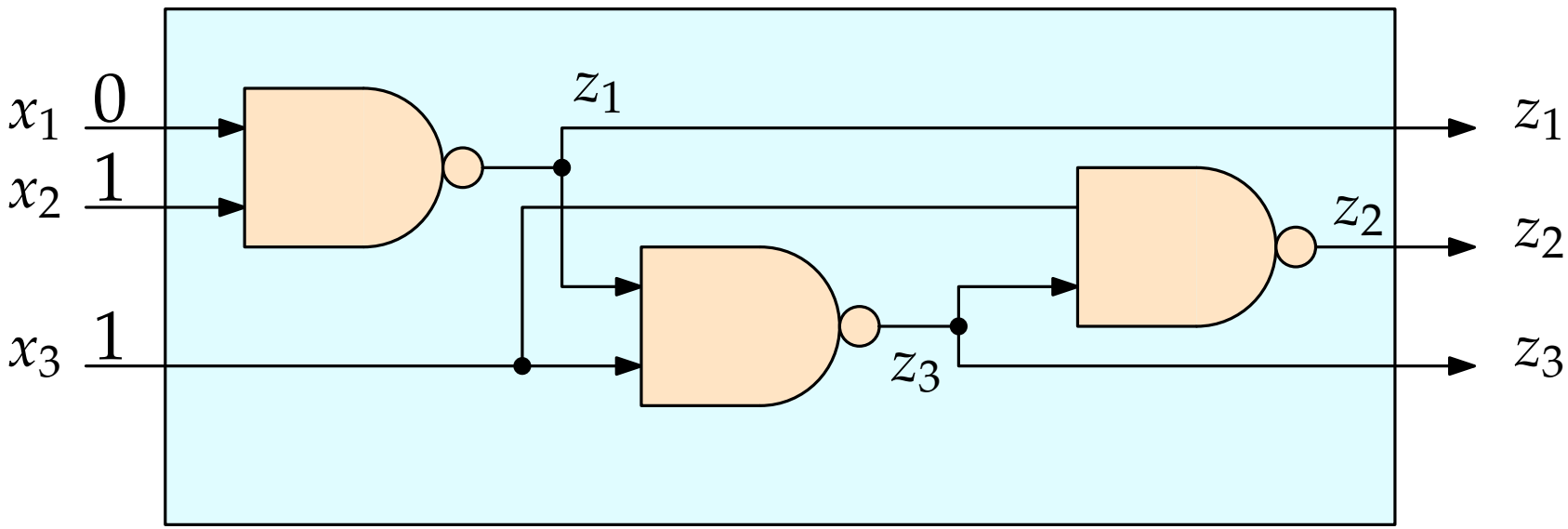


When all gates are correct, then the output is correct.

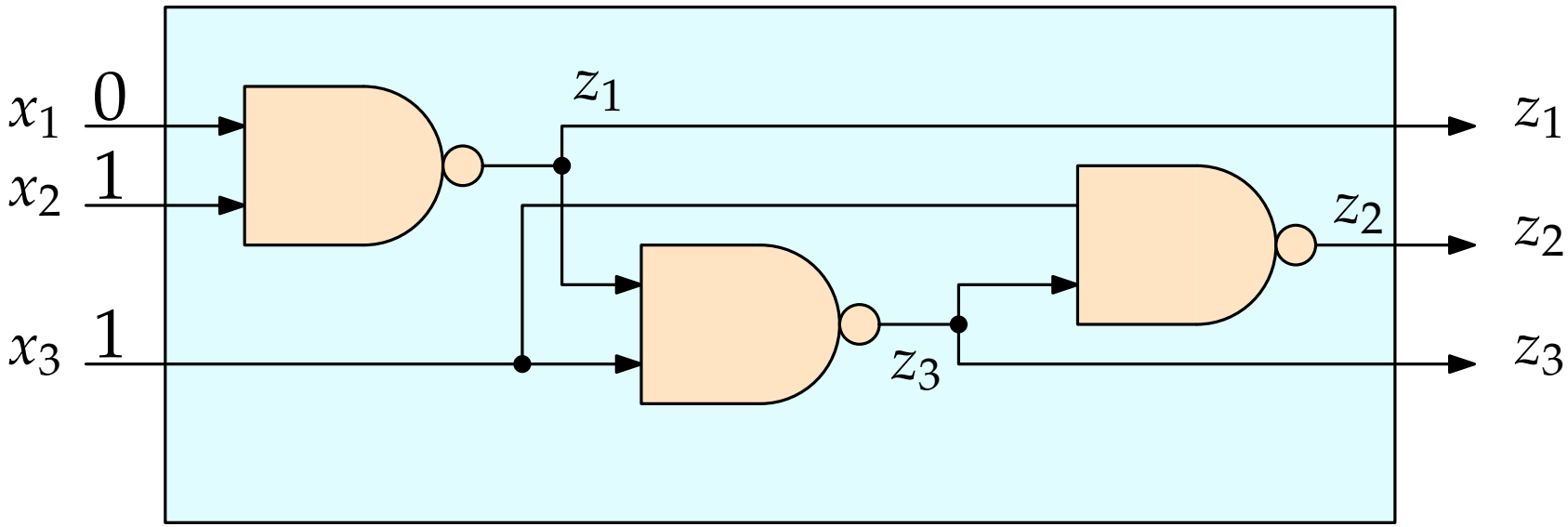


Now build a 3-CNF formula to simulate this!

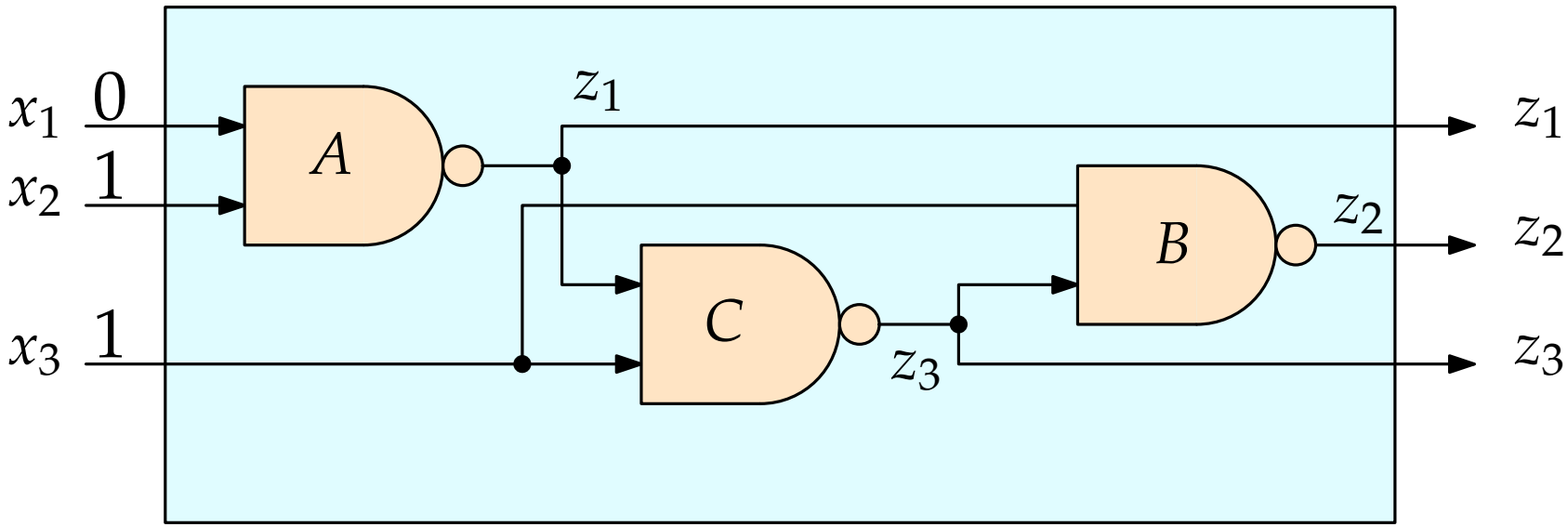

0
1
1


x_1 0
x_2 1
x_3 1
z_1
z_2
z_3
z_1
z_2
z_3

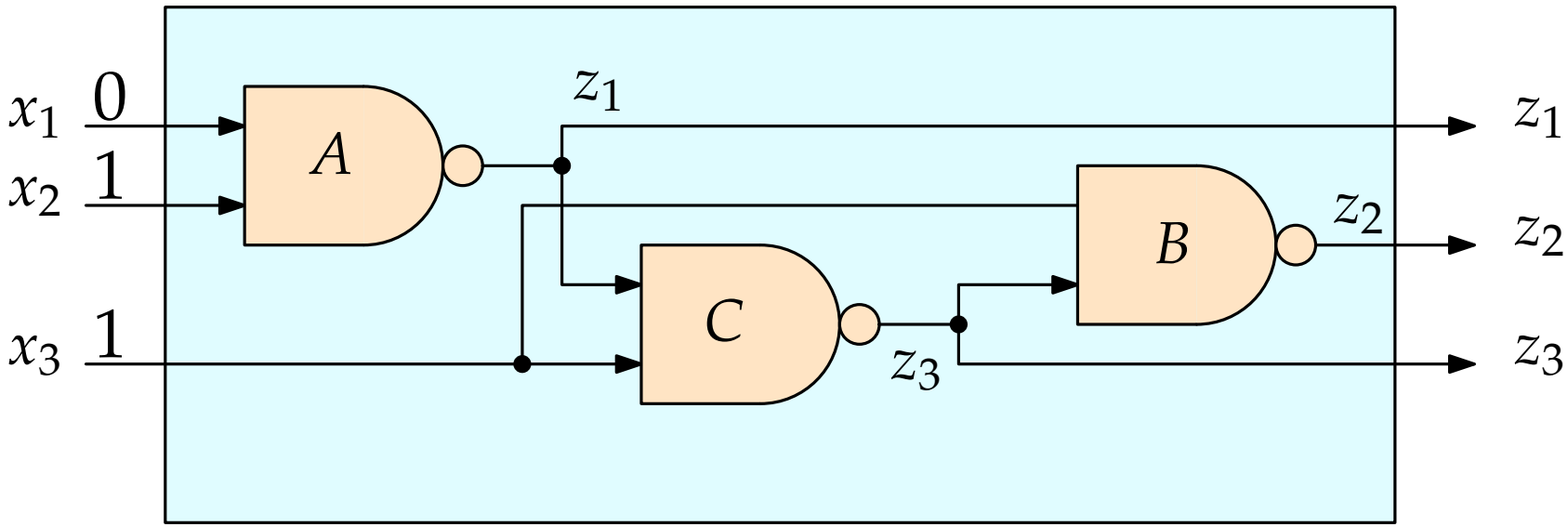
Here, every gate output is also a circuit output. But that need not the case in general.



Here, every gate output is also a circuit output. But that need not the case in general.

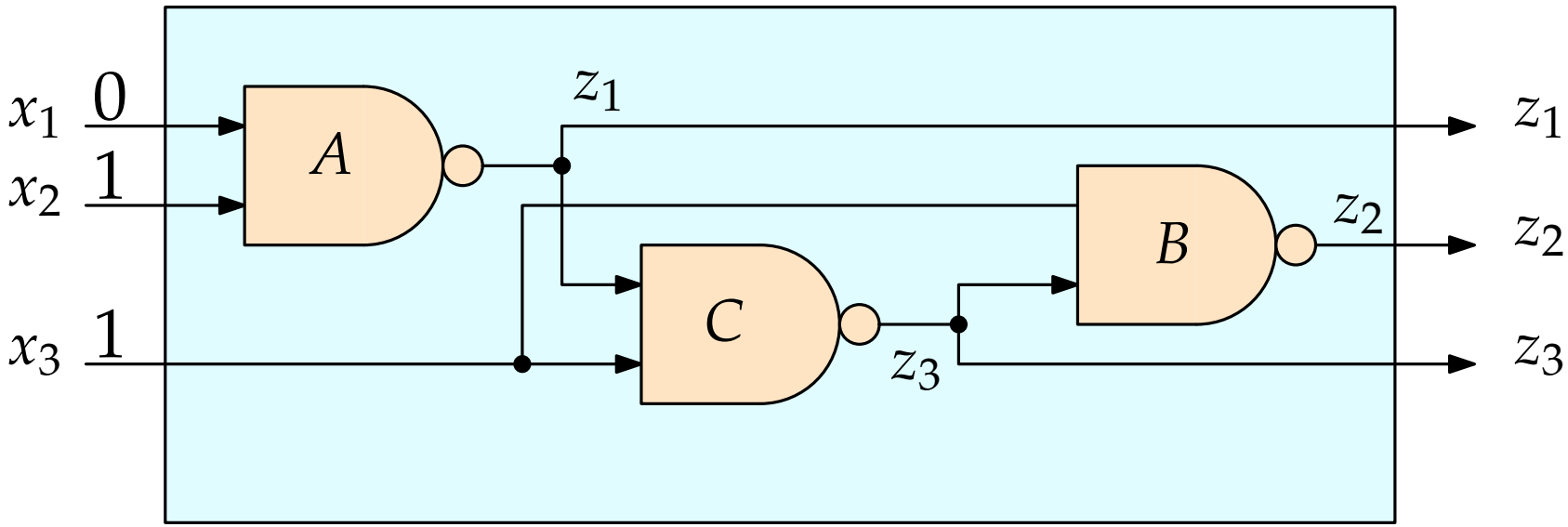


Here, every gate output is also a circuit output. But that need not the case in general.



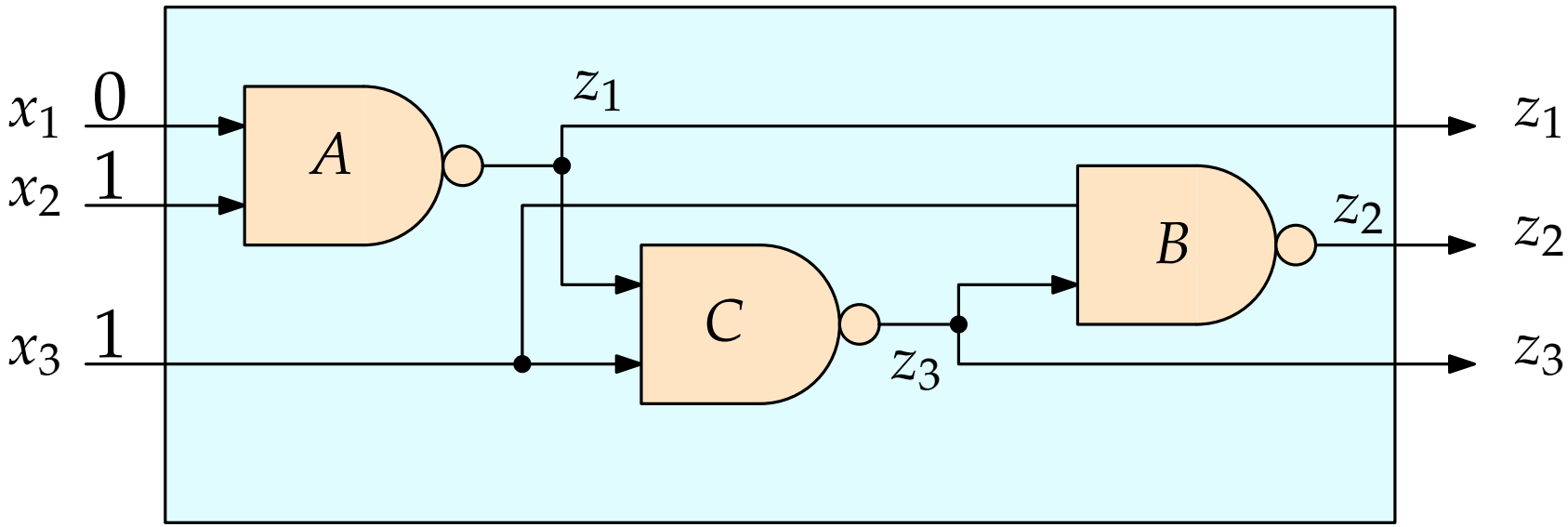
$(\bar{x}_1)$ (x_2) (x_3)

Here, every gate output is also a circuit output. But that need not the case in general.



$$(\bar{x}_1) \quad (x_2) \quad (x_3) \quad (z_1 \leftrightarrow \neg(x_1 \wedge x_2))$$
$$(z_3 \leftrightarrow \neg(z_1 \wedge x_3))$$
$$(z_2 \leftrightarrow \neg(x_3 \wedge z_3))$$

Here, every gate output is also a circuit output. But that need not the case in general.

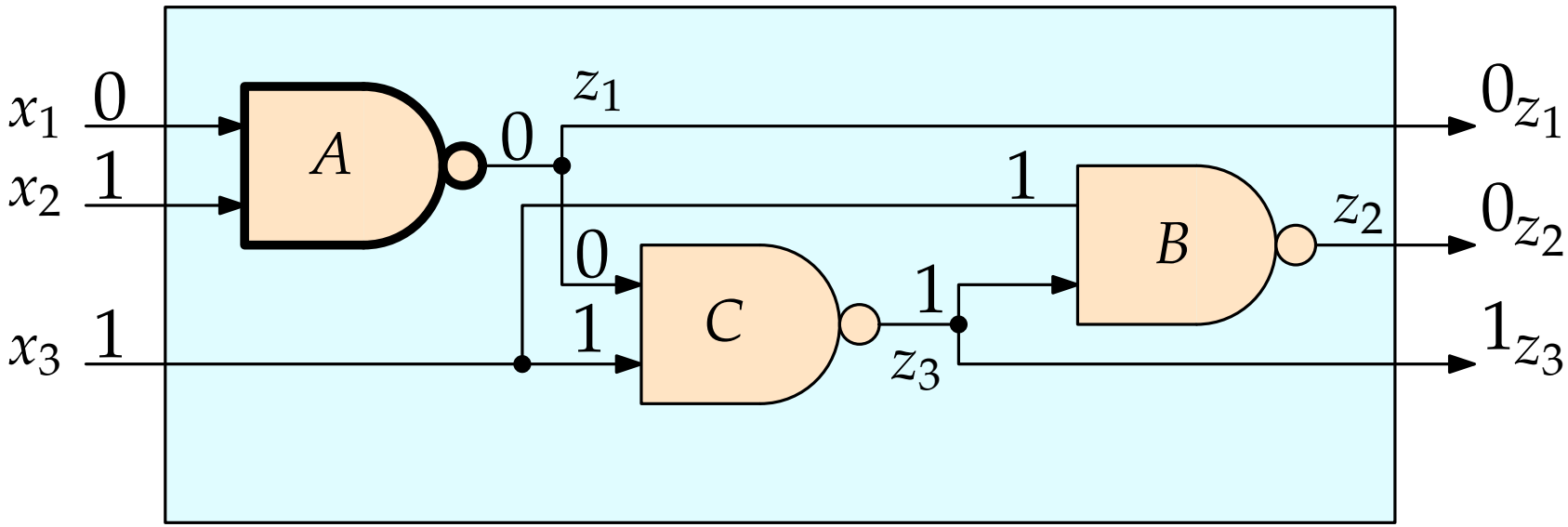


$(\bar{x}_1)$ (x_2) (x_3) $(z_1 \leftrightarrow \neg(x_1 \wedge x_2))$ 4

32 16 8 $(z_3 \leftrightarrow \neg(z_1 \wedge x_3))$ 2

$(z_2 \leftrightarrow \neg(x_3 \wedge z_3))$ 1

Here, every gate output is also a circuit output. But that need not the case in general.

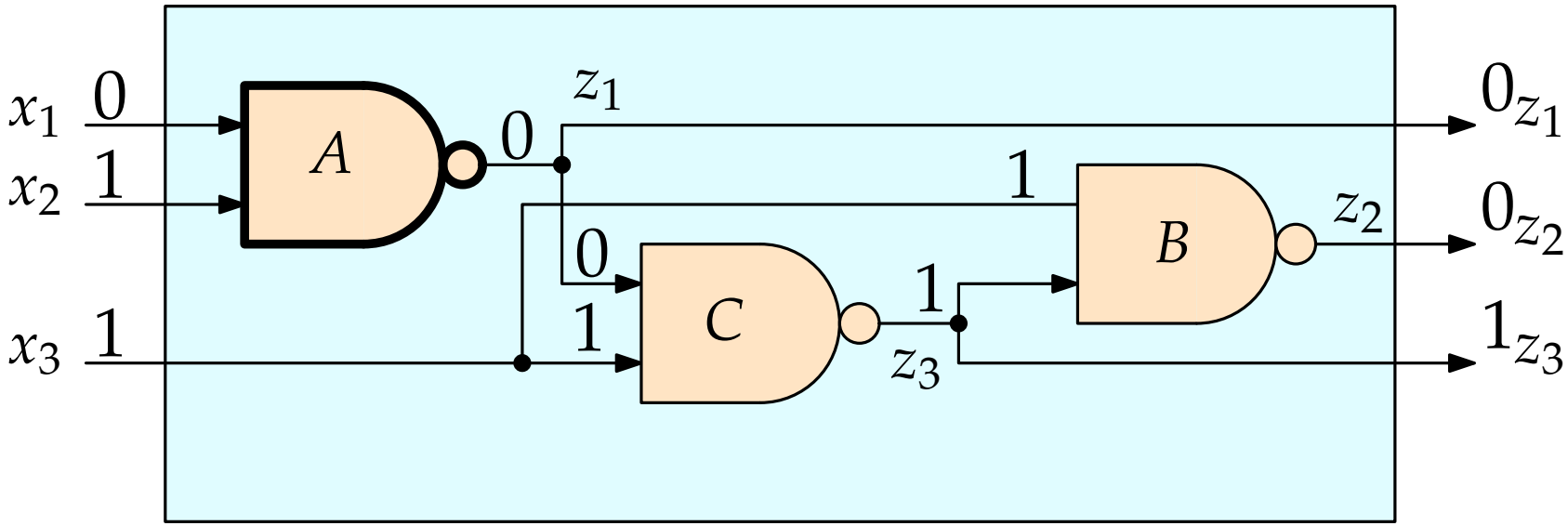


$(\bar{x}_1)$ (x_2) (x_3) $(z_1 \leftrightarrow \neg(x_1 \wedge x_2))$ 4

32 16 8 $(z_3 \leftrightarrow \neg(z_1 \wedge x_3))$ 2

$(z_2 \leftrightarrow \neg(x_3 \wedge z_3))$ 1

Here, every gate output is also a circuit output. But that need not the case in general.

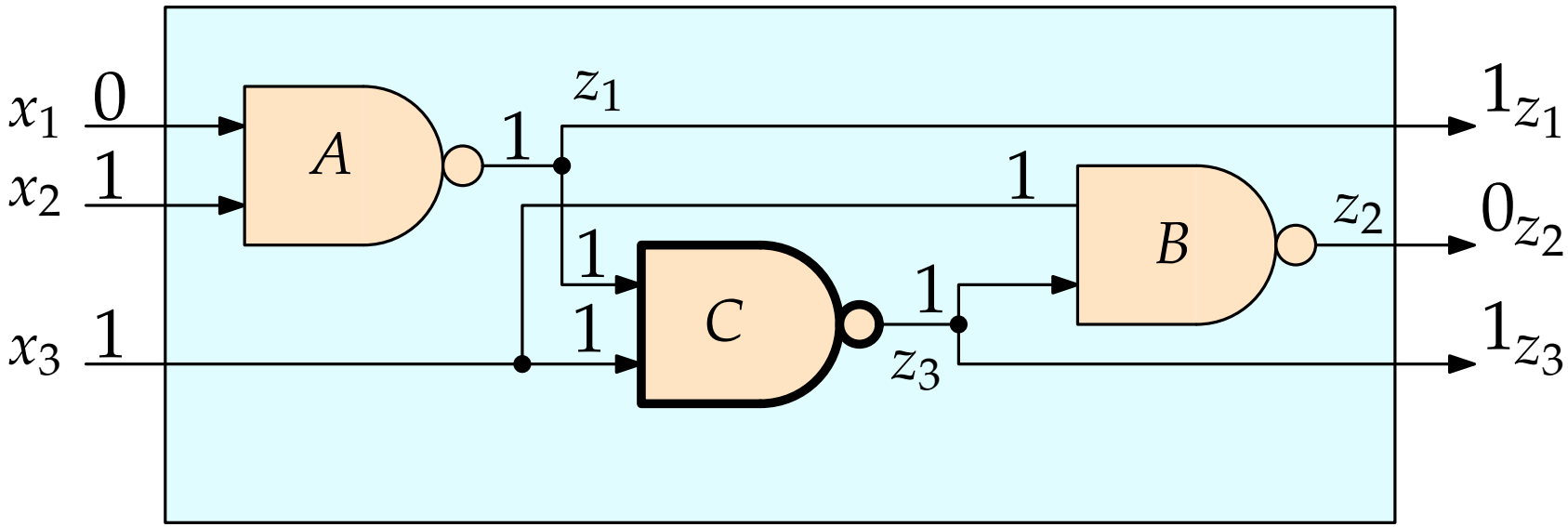


$(\bar{x}_1)$ (x_2) (x_3) $(z_1 \leftrightarrow \neg(x_1 \wedge x_2))$ 4

32 16 8 $(z_3 \leftrightarrow \neg(z_1 \wedge x_3))$ 2

$(z_2 \leftrightarrow \neg(x_3 \wedge z_3))$ 1

Here, every gate output is also a circuit output. But that need not the case in general.

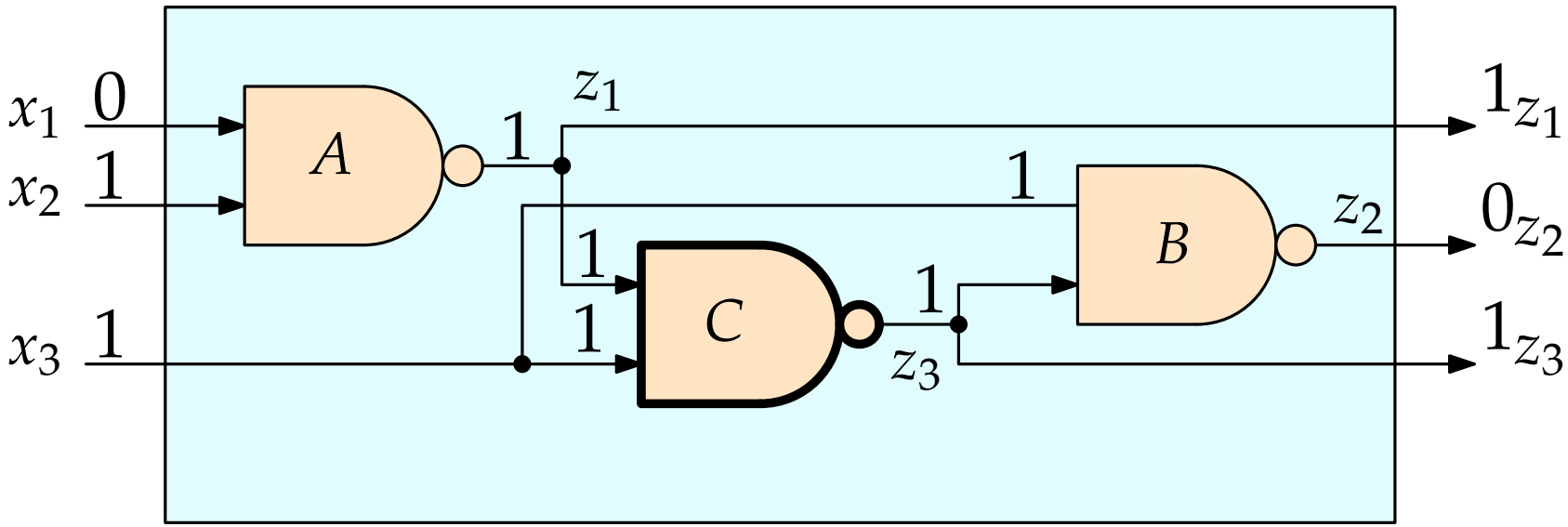


$(\bar{x}_1)$ (x_2) (x_3) $(z_1 \leftrightarrow \neg(x_1 \wedge x_2))$ 4

32 16 8 $(z_3 \leftrightarrow \neg(z_1 \wedge x_3))$ 2

$(z_2 \leftrightarrow \neg(x_3 \wedge z_3))$ 1

Here, every gate output is also a circuit output. But that need not the case in general.

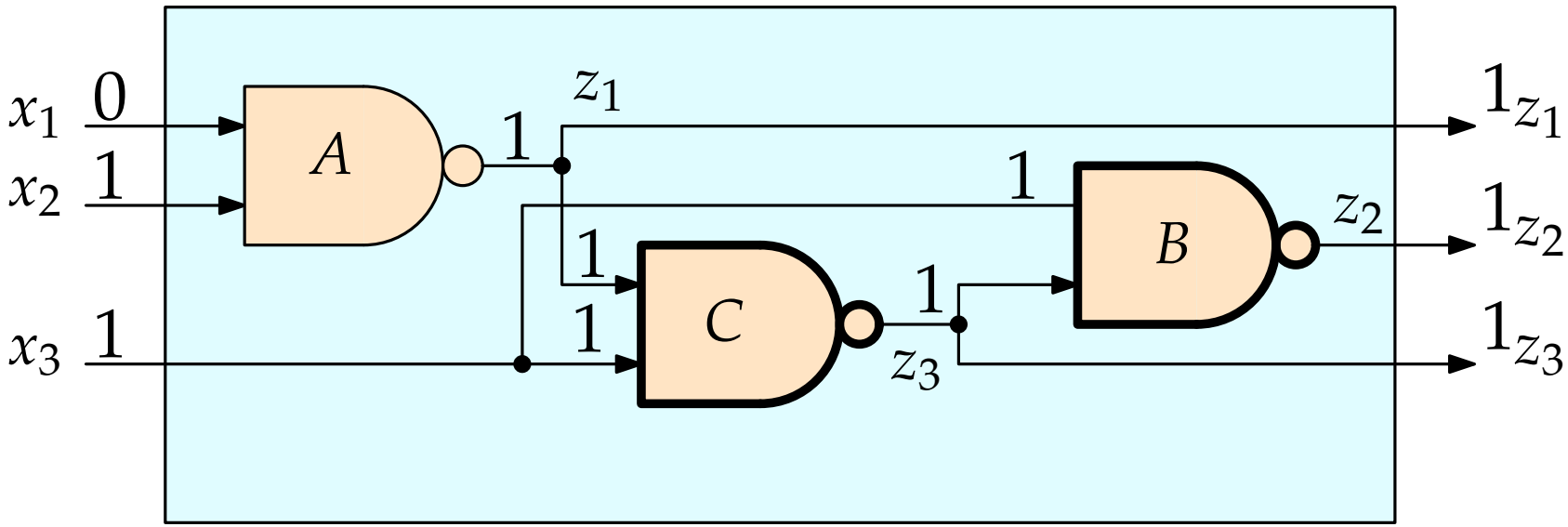


$(\bar{x}_1)$ (x_2) (x_3) $(z_1 \leftrightarrow \neg(x_1 \wedge x_2))$ 4

32 16 8 $(z_3 \leftrightarrow \neg(z_1 \wedge x_3))$ 2

$(z_2 \leftrightarrow \neg(x_3 \wedge z_3))$ 1

Here, every gate output is also a circuit output. But that need not the case in general.

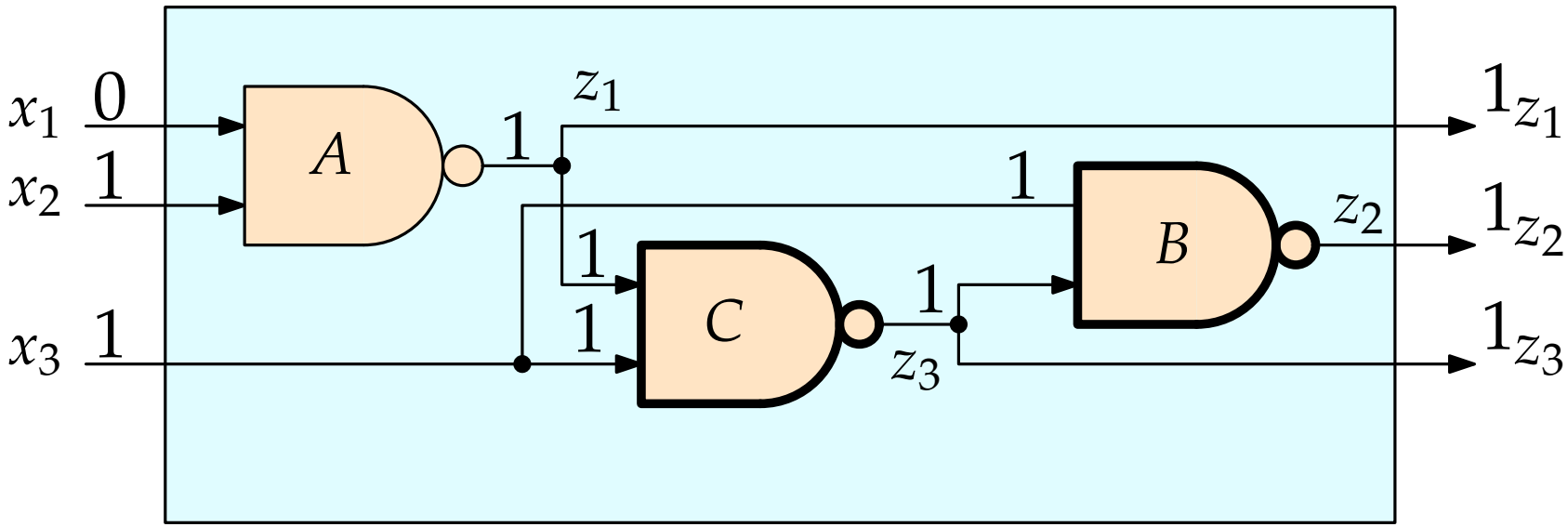


$(\bar{x}_1)$ (x_2) (x_3) $(z_1 \leftrightarrow \neg(x_1 \wedge x_2))$ 4

32 16 8 $(z_3 \leftrightarrow \neg(z_1 \wedge x_3))$ 2

$(z_2 \leftrightarrow \neg(x_3 \wedge z_3))$ 1

Here, every gate output is also a circuit output. But that need not the case in general.

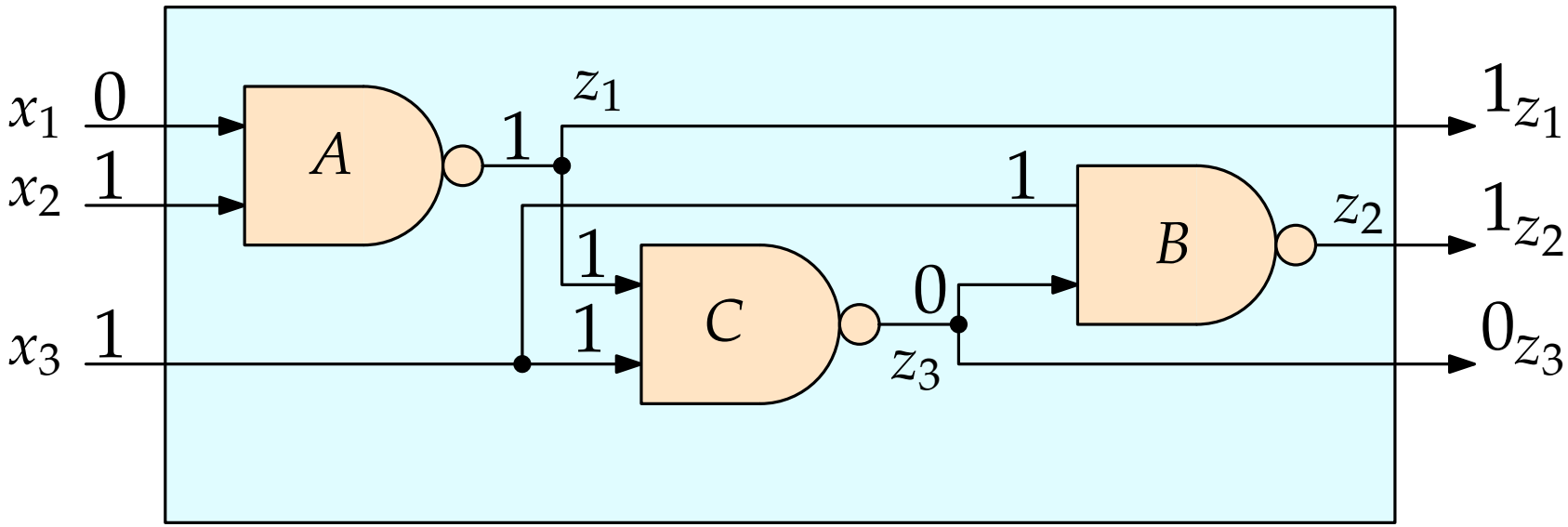


$(\bar{x}_1)$ (x_2) (x_3) $(z_1 \leftrightarrow \neg(x_1 \wedge x_2))$ 4

32 16 8 $(z_3 \leftrightarrow \neg(z_1 \wedge x_3))$ 2

$(z_2 \leftrightarrow \neg(x_3 \wedge z_3))$ 1

Here, every gate output is also a circuit output. But that need not the case in general.

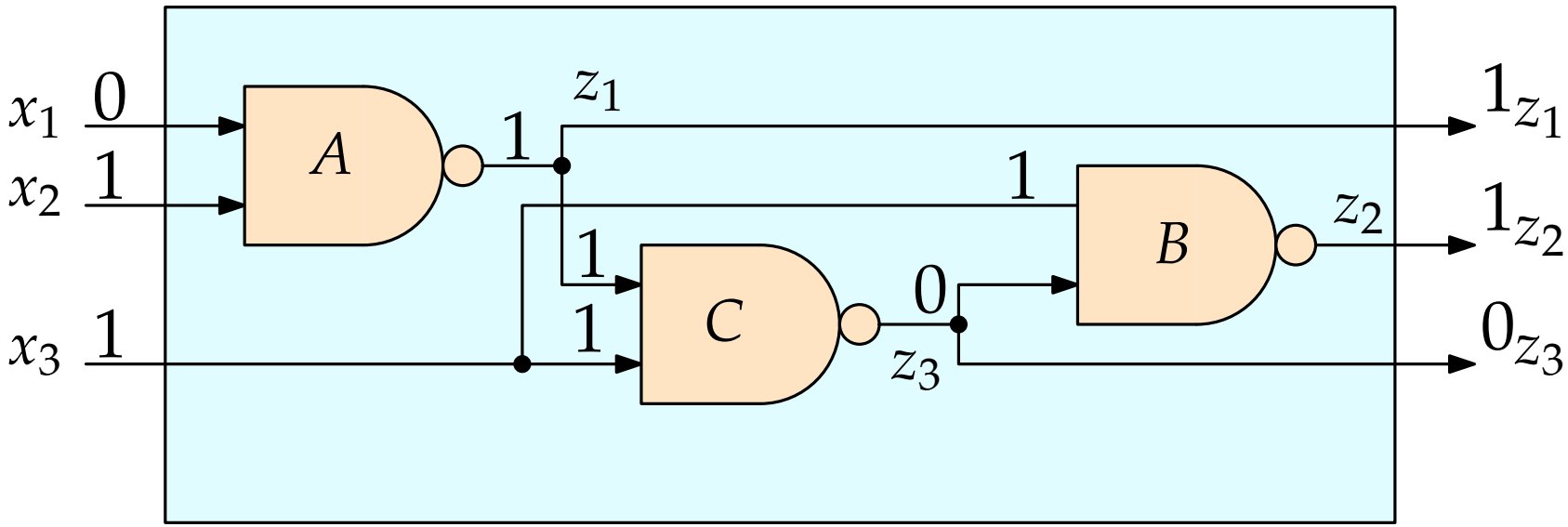


$(\bar{x}_1)$ (x_2) (x_3) $(z_1 \leftrightarrow \neg(x_1 \wedge x_2))$ 4

32 16 8 $(z_3 \leftrightarrow \neg(z_1 \wedge x_3))$ 2

$(z_2 \leftrightarrow \neg(x_3 \wedge z_3))$ 1

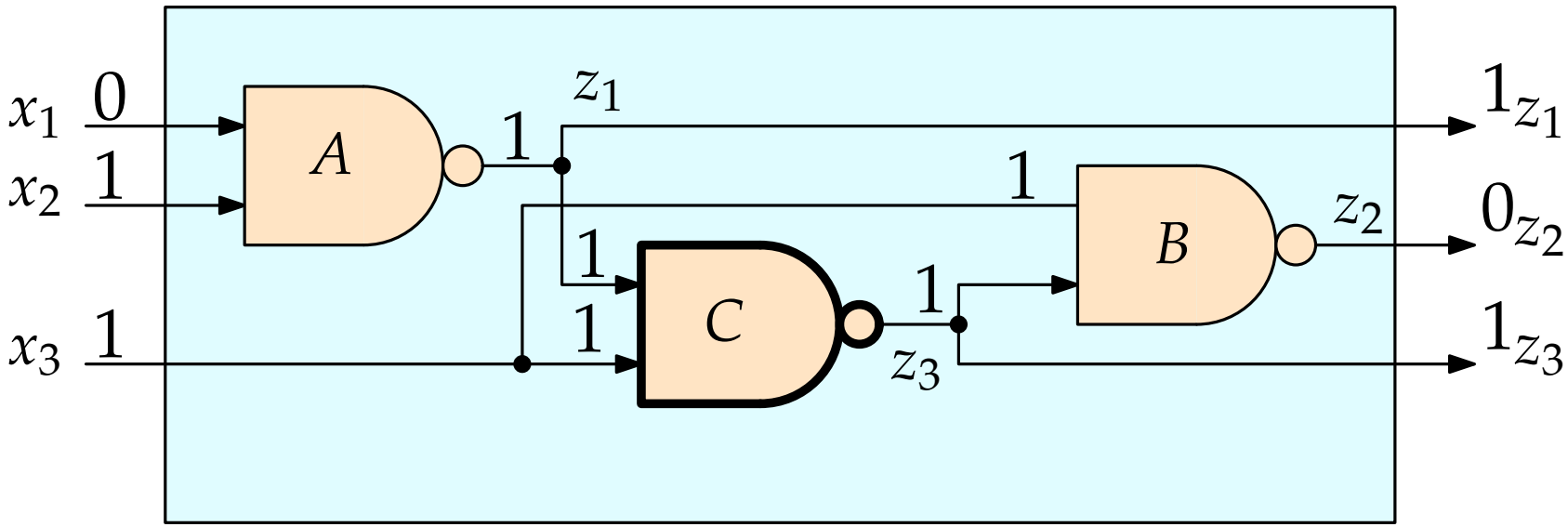
Here, every gate output is also a circuit output. But that need not the case in general.

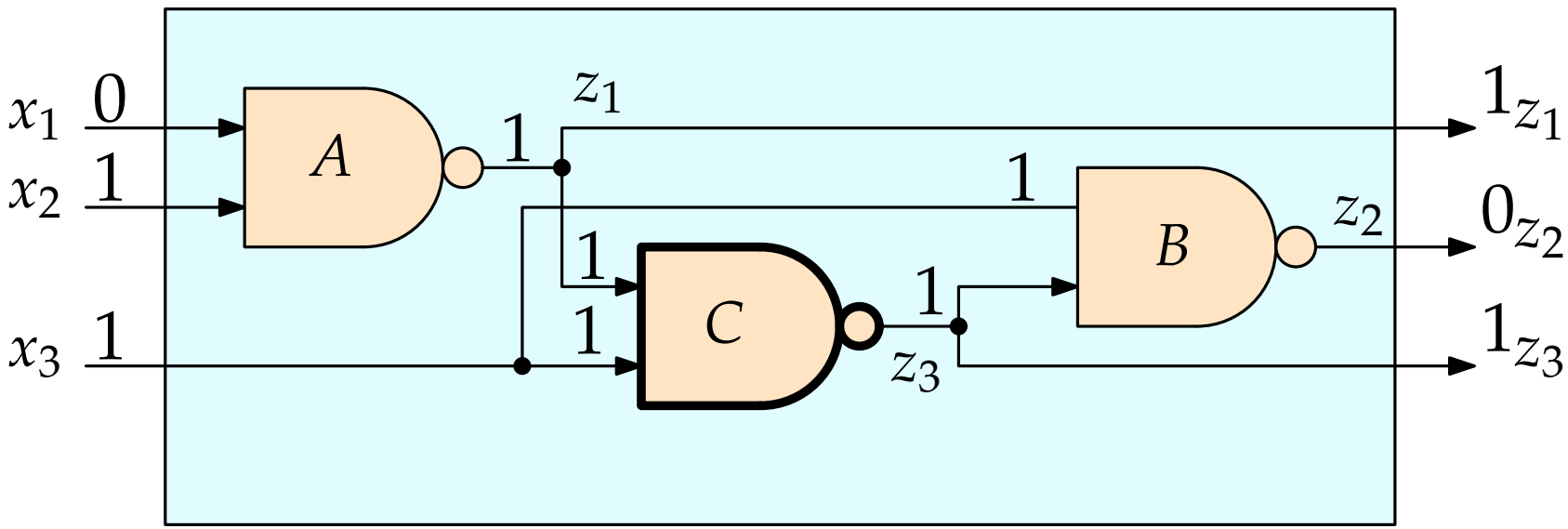


$(\bar{x}_1)$ (x_2) (x_3) $(z_1 \leftrightarrow \neg(x_1 \wedge x_2))$ 4

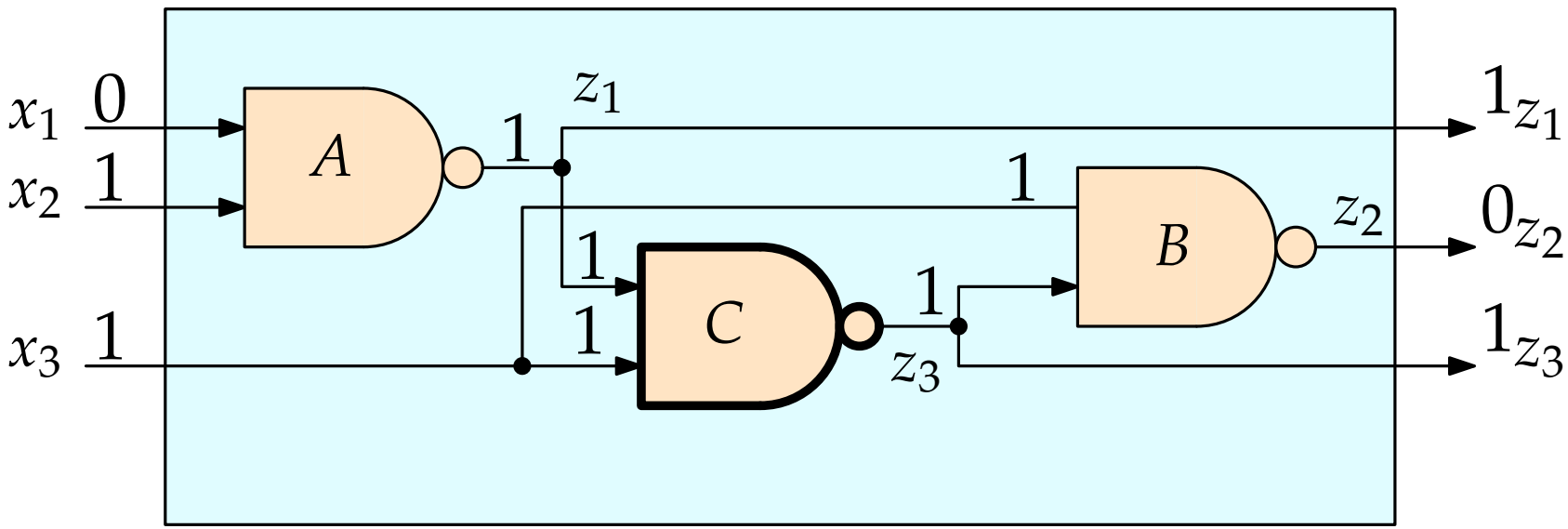
32 16 8 $(z_3 \leftrightarrow \neg(z_1 \wedge x_3))$ 2

$(z_2 \leftrightarrow \neg(x_3 \wedge z_3))$ 1

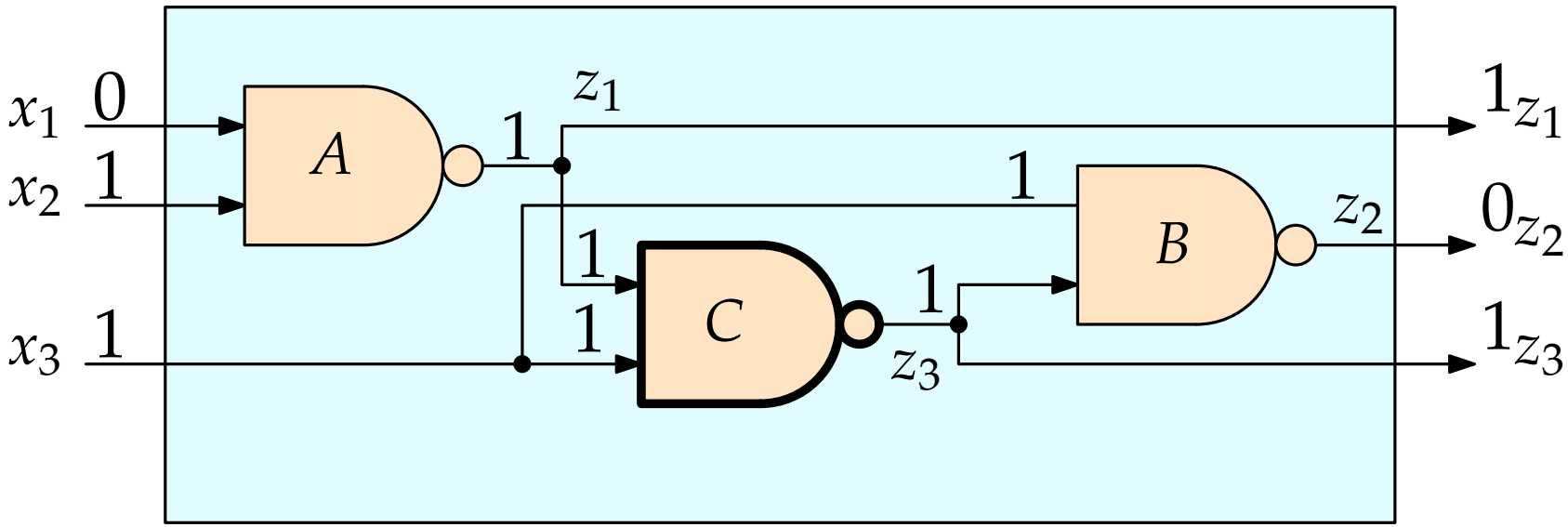

x_1 0
x_2 1
x_3 1
A
B
C
1
1
1
1
1
z_1
z_2
z_3
1_{z_1}
0_{z_2}
1_{z_3}



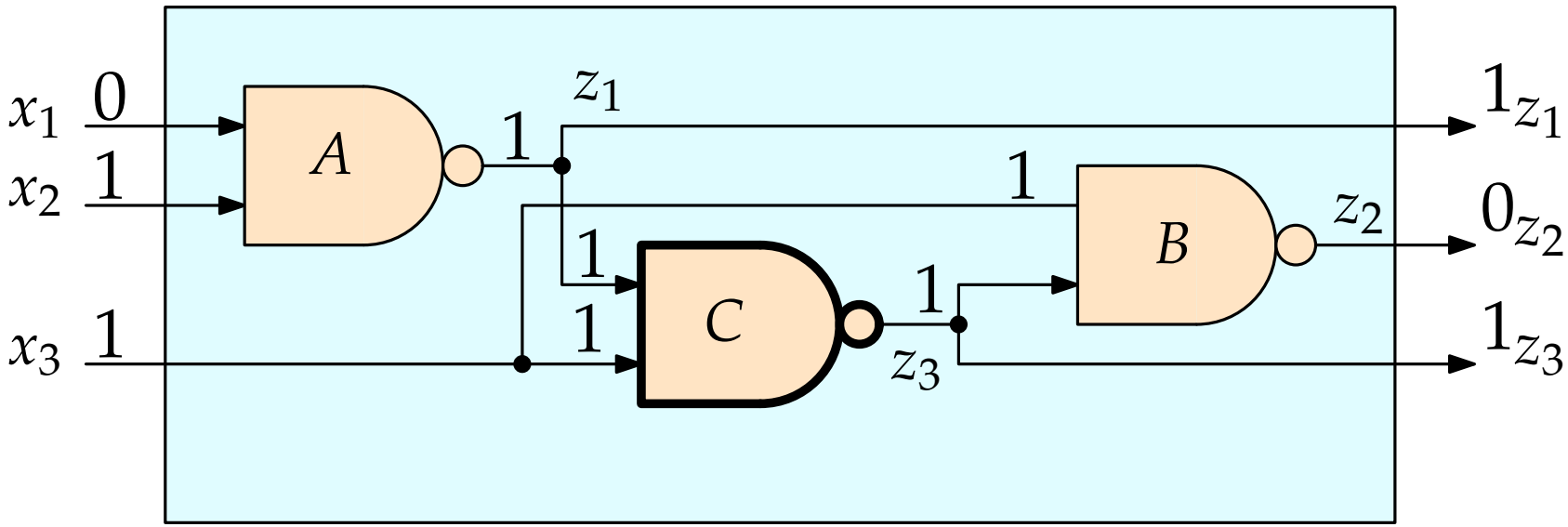
As long as a gate as violated, click the leftmost violated gate.



As long as a gate as violated, click the leftmost violated gate. This will satisfy its clause C.



As long as a gate as violated, click the leftmost violated gate.
This will satisfy its clause C.
Downstream clauses might become violated.



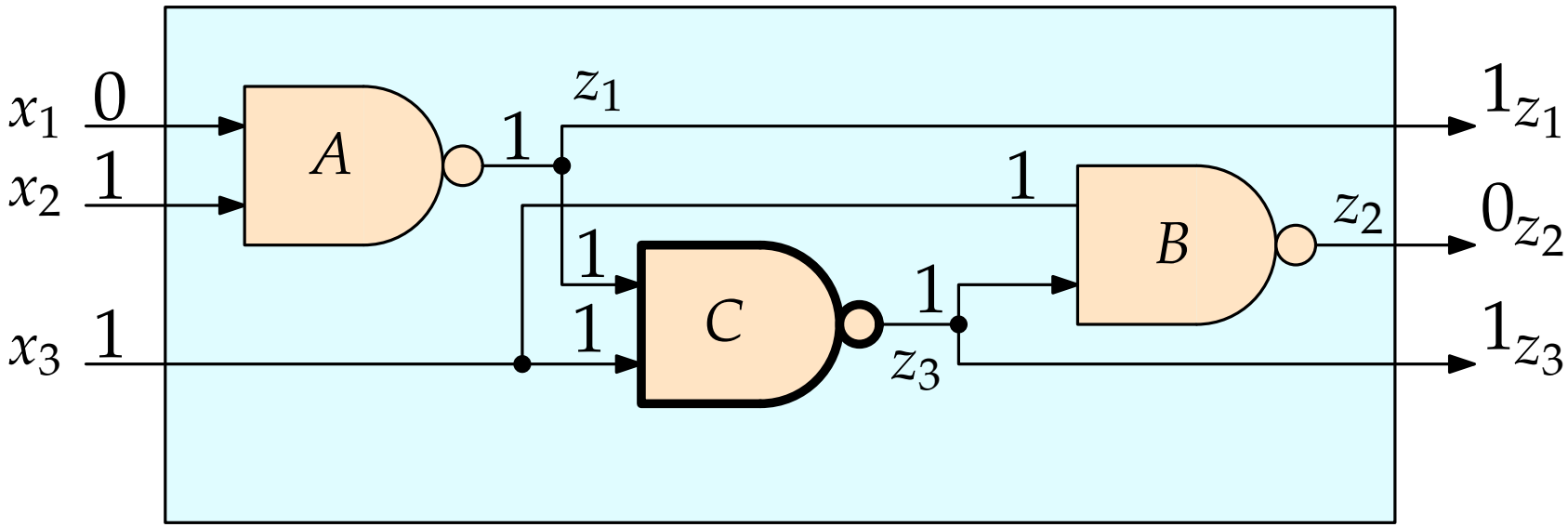
As long as a gate as violated, click the leftmost violated gate.

This will satisfy its clause C.

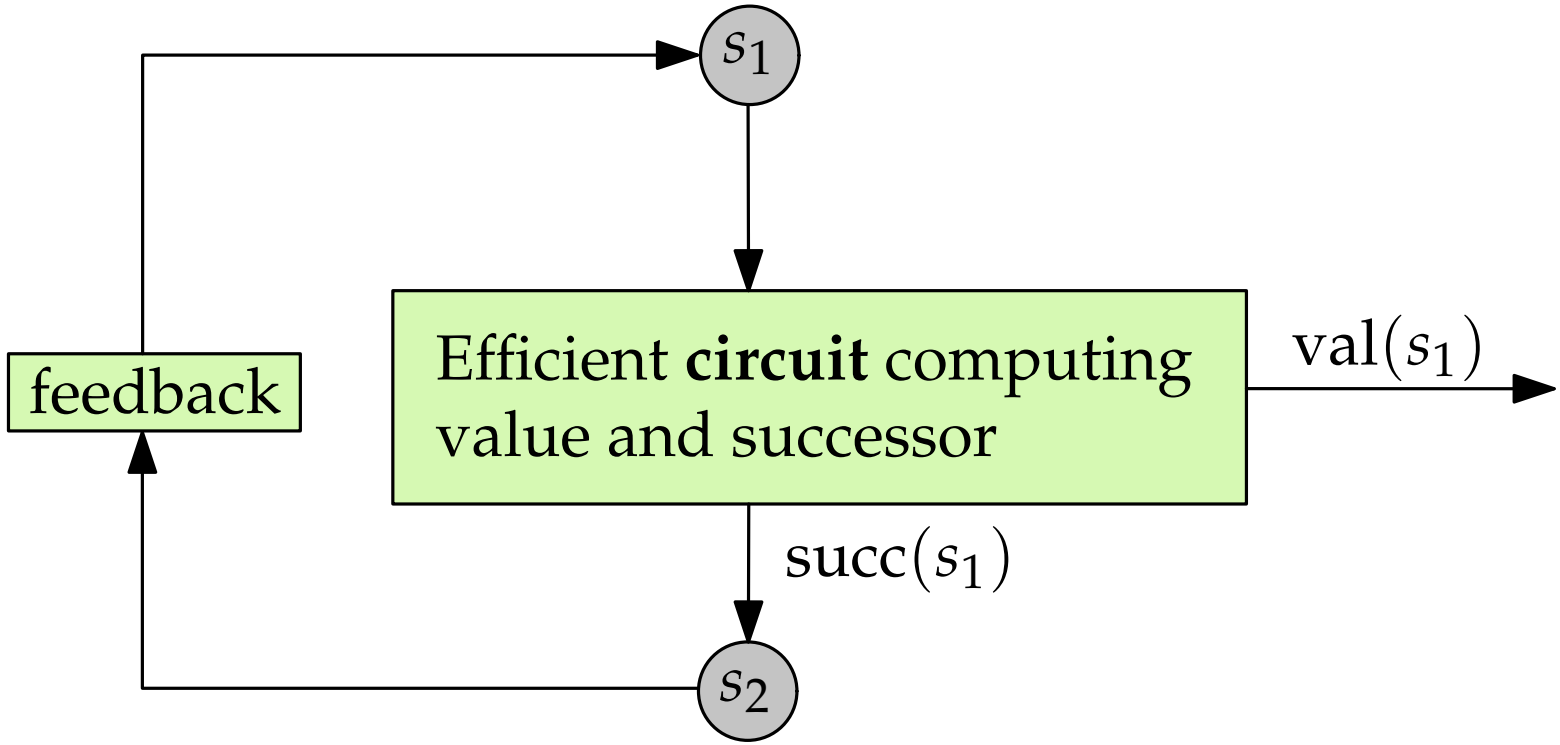
Downstream clauses might become violated.

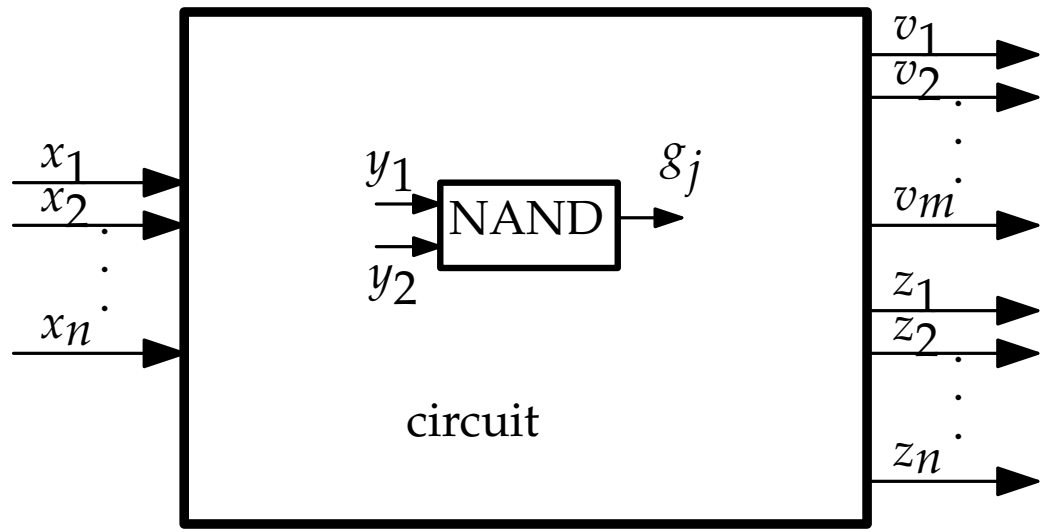
It's fine because C is more valuable than all downstream clauses combined.

Values of $\vec{z}$ in the local optimum are the correct evaluation of C.

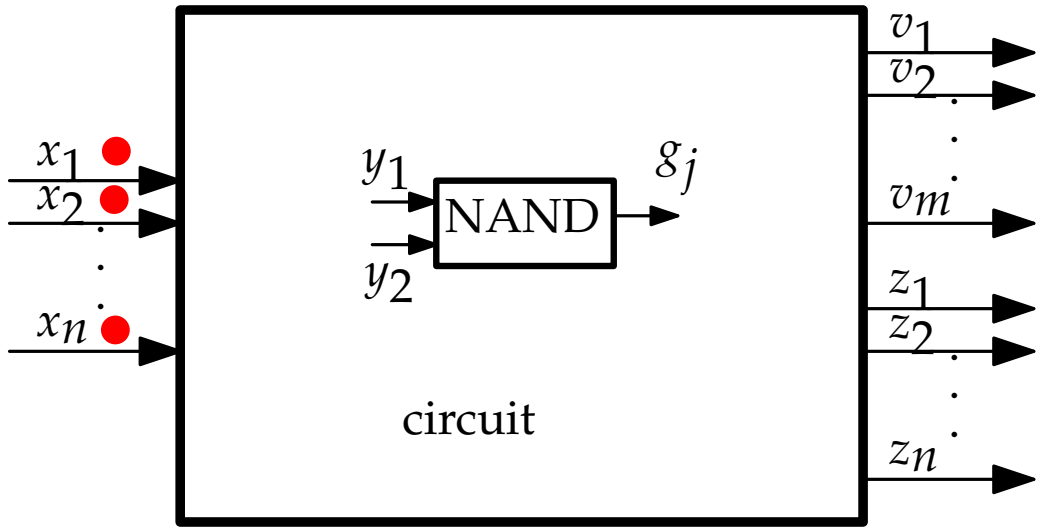


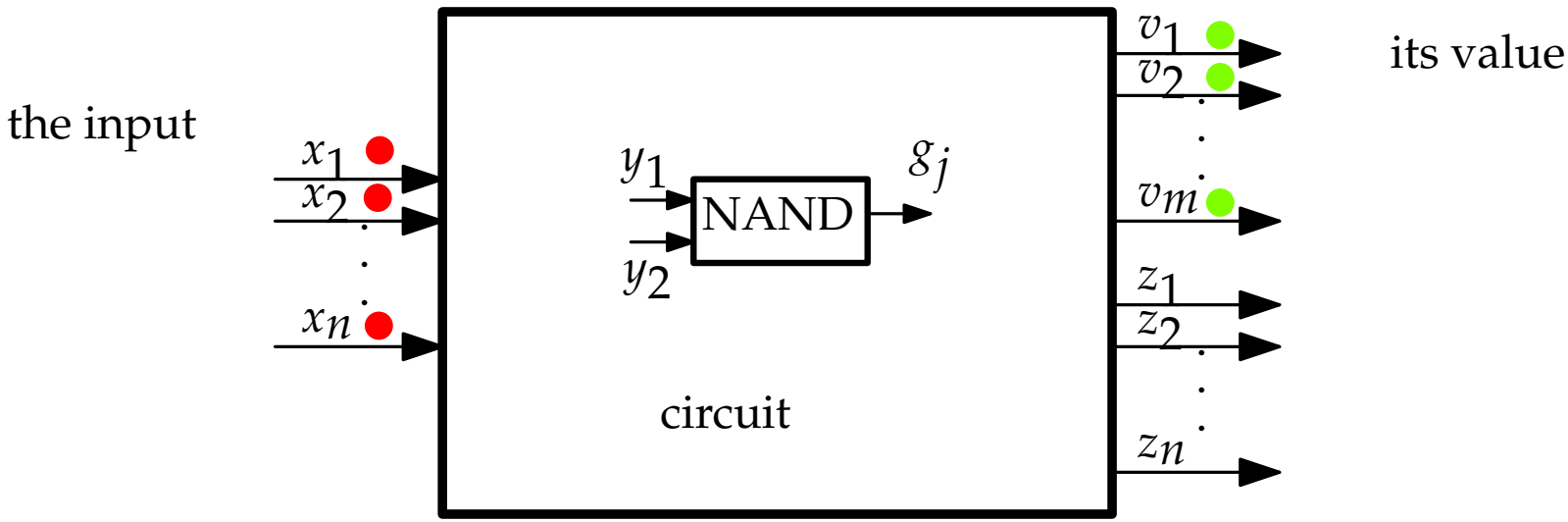
As long as a gate as violated, click the leftmost violated gate.
This will satisfy its clause C.
Downstream clauses might become violated.
It's fine because C is more valuable than all downstream clauses combined.

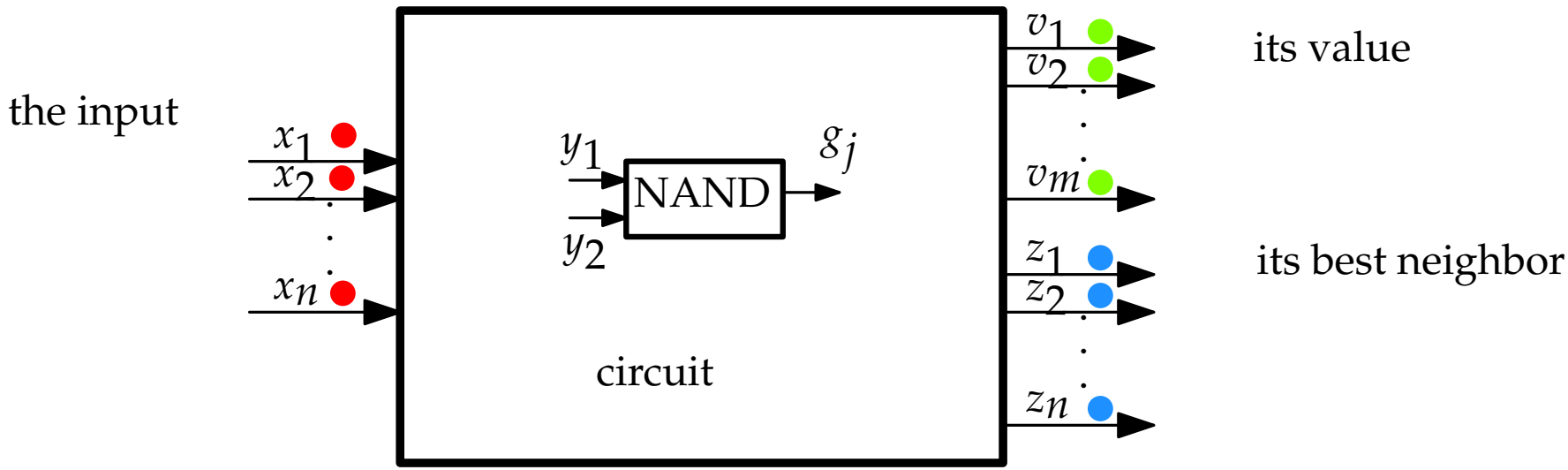

s_1
Efficient **circuit** computing value and successor
val(s_1)
feedback
succ(s_1)
s_2

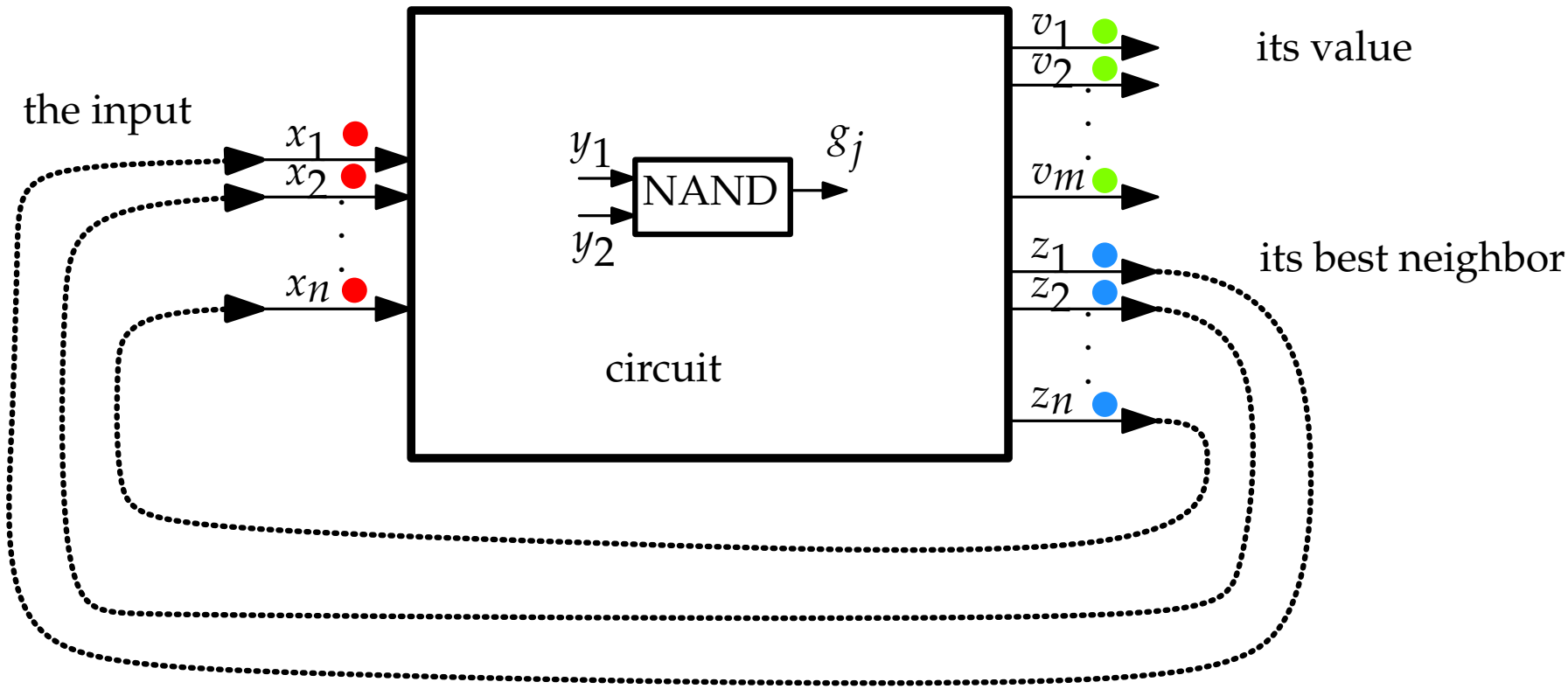

x_1
x_2
x_n
y_1
y_2
NAND
g_j
circuit
v_1
v_2
v_m
z_1
z_2
z_n

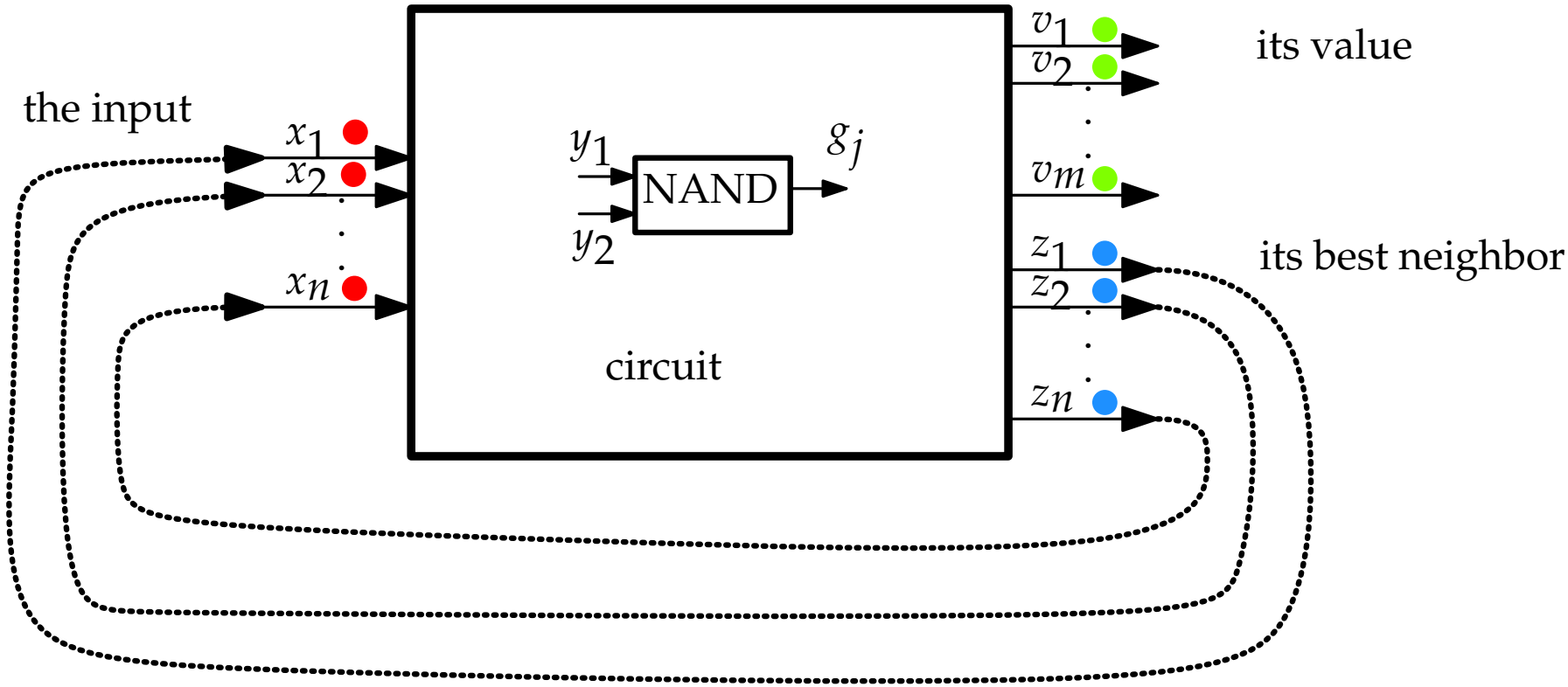
the input



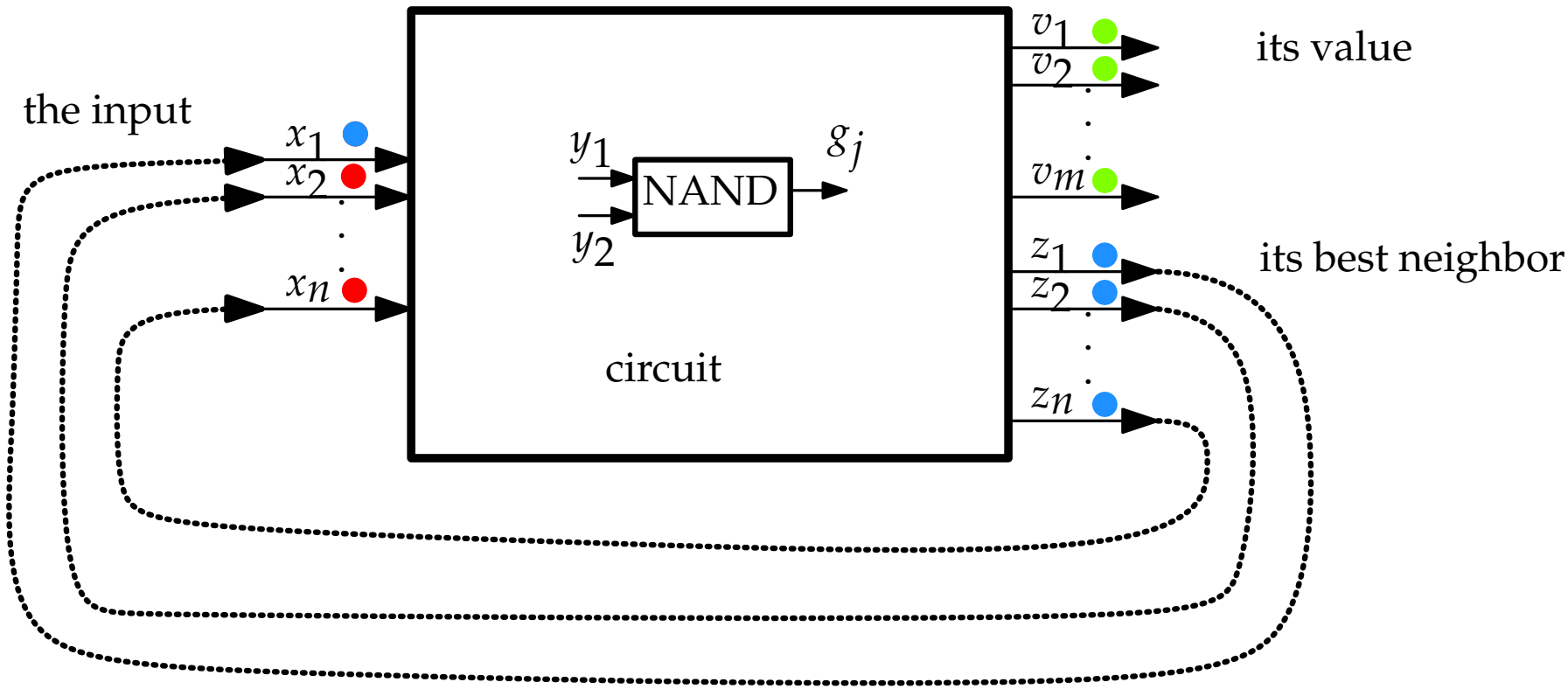

the input
x_1
x_2
x_n
y_1
y_2
NAND
g_j
circuit
v_1
v_2
v_m
z_1
z_2
z_n
its value


the input
x_1
x_2
x_n
y_1
y_2
NAND
g_j
circuit
v_1
v_2
v_m
z_1
z_2
z_n
its value
its best neighbor

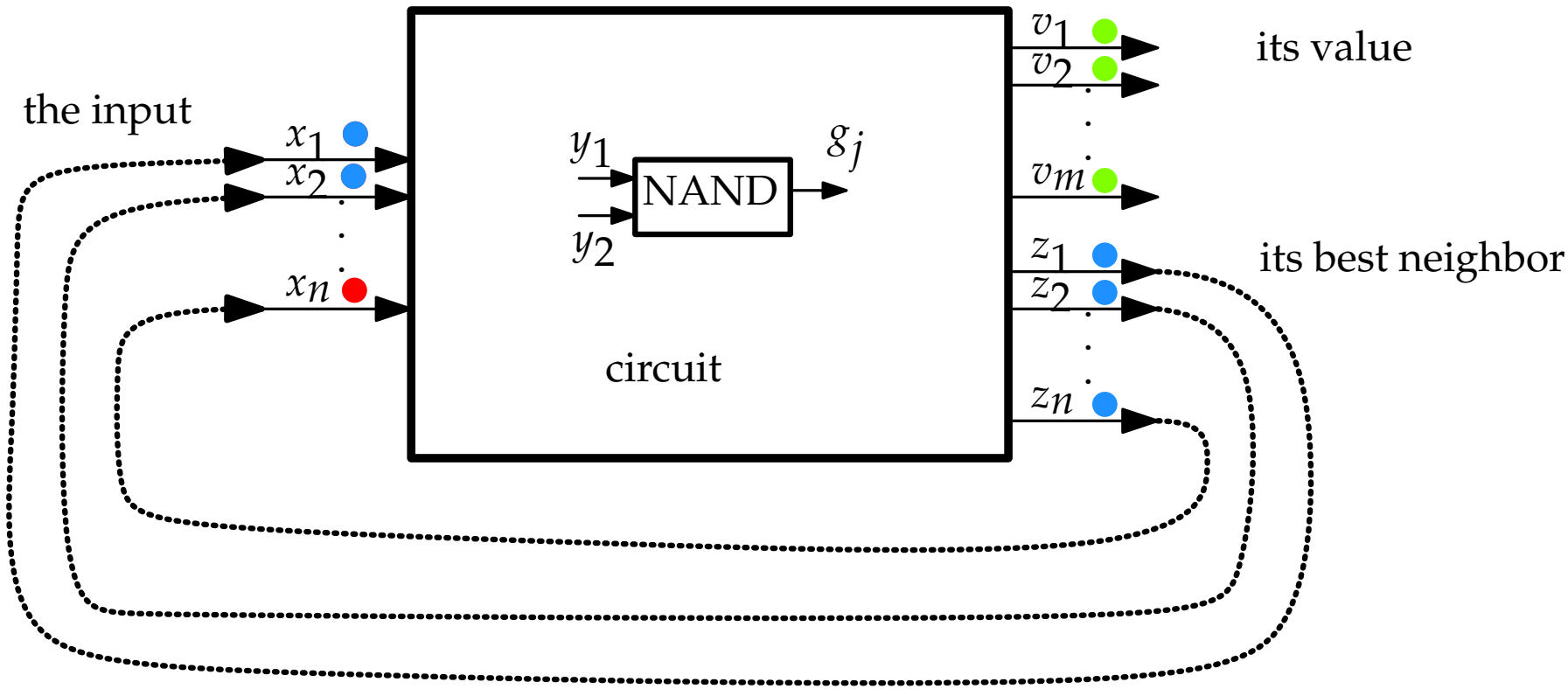

the input
x_1
x_2
x_n
y_1
y_2
NAND
g_j
circuit
v_1
v_2
v_m
its value
z_1
z_2
z_n
its best neighbor



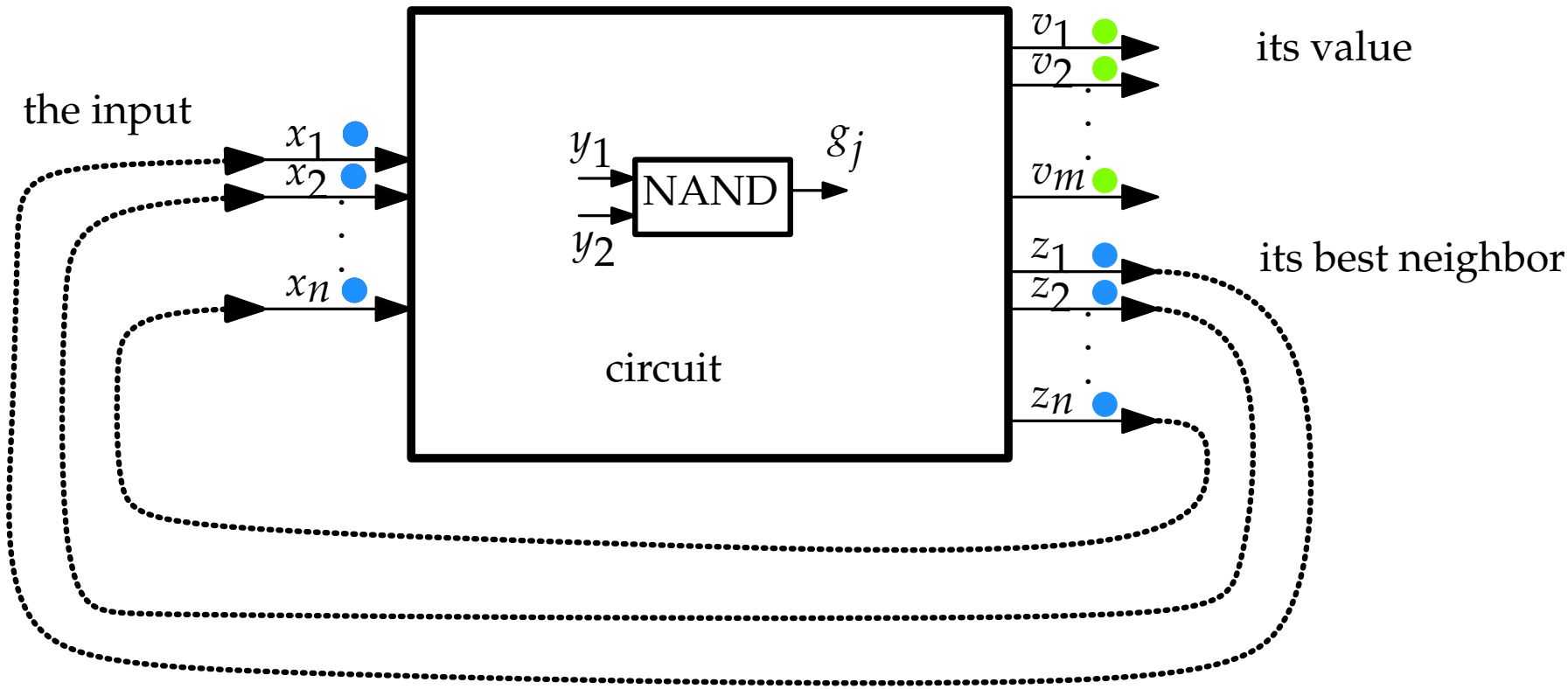
Want: Once $\vec{v}$ and $\vec{z}$ are correct, the "user" should have an incentive to flip the x_j to match the z_j to iterate the evaluation.



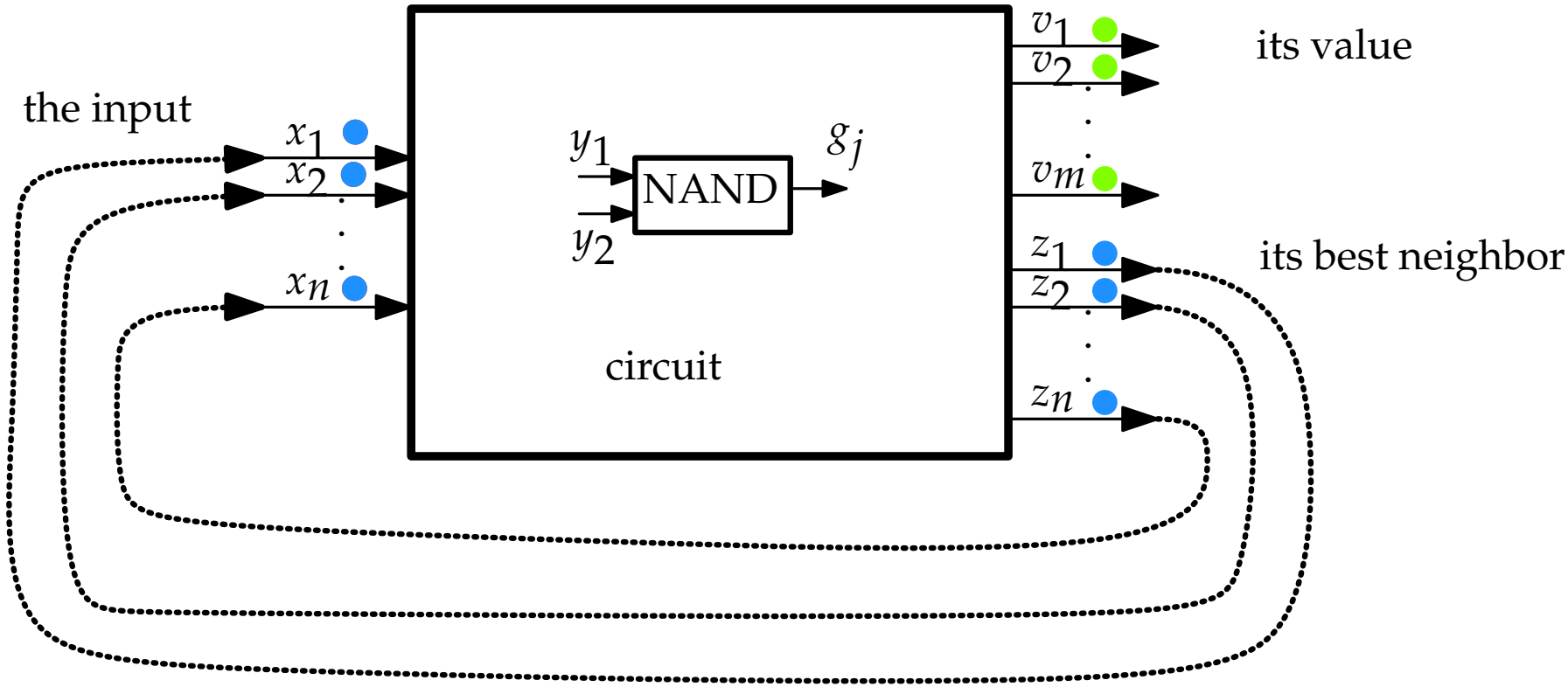
Want: Once $\vec{v}$ and $\vec{z}$ are correct, the "user" should have an incentive to flip the x_j to match the z_j to iterate the evaluation.



Want: Once $\vec{v}$ and $\vec{z}$ are correct, the "user" should have an incentive to flip the x_j to match the z_j to iterate the evaluation.

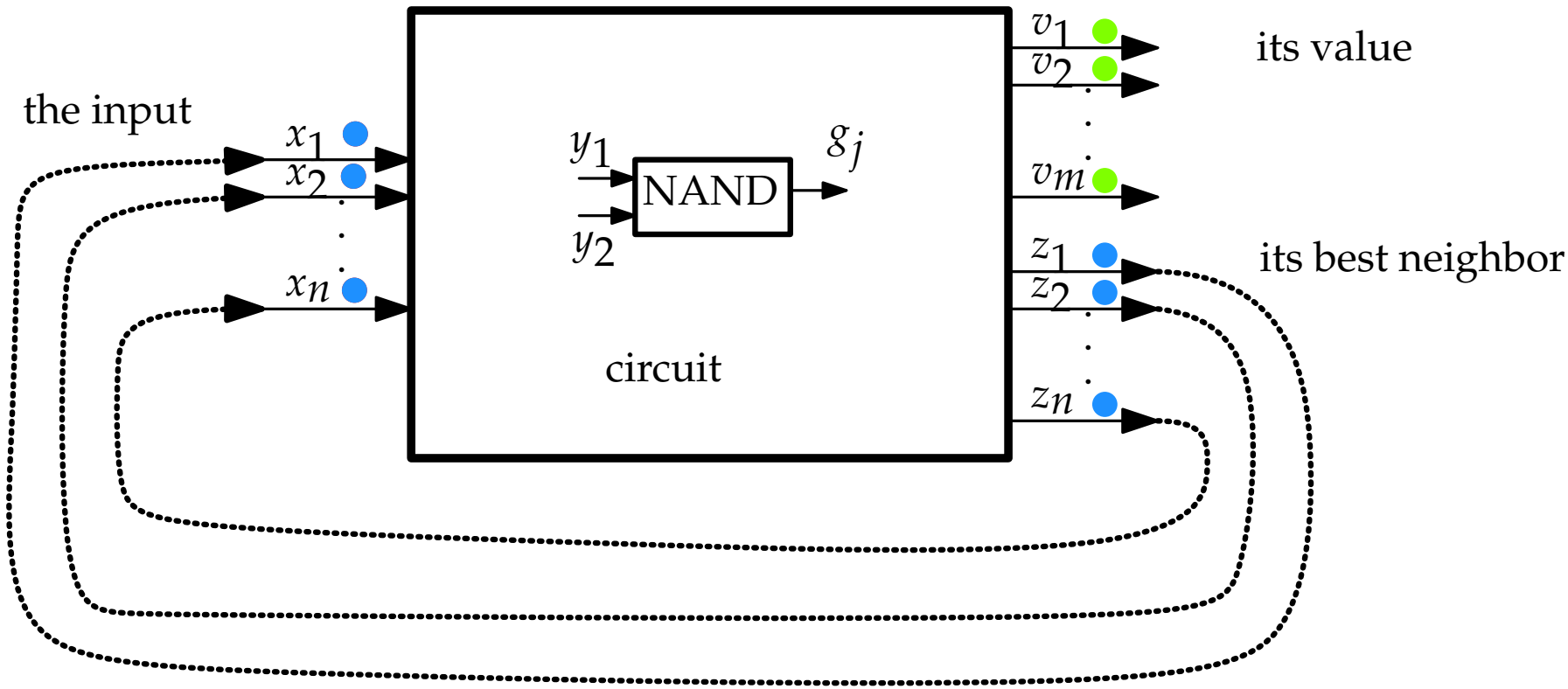


Want: Once $\vec{v}$ and $\vec{z}$ are correct, the "user" should have an incentive to flip the x_j to match the z_j to iterate the evaluation. But this will violate gates!



Want: Once $\vec{v}$ and $\vec{z}$ are correct, the "user" should have an incentive to flip the x_j to match the z_j to iterate the evaluation. But this will violate gates!

And should only work if $\text{val}(\vec{z})$ is better than $\text{val}(\vec{x})$!

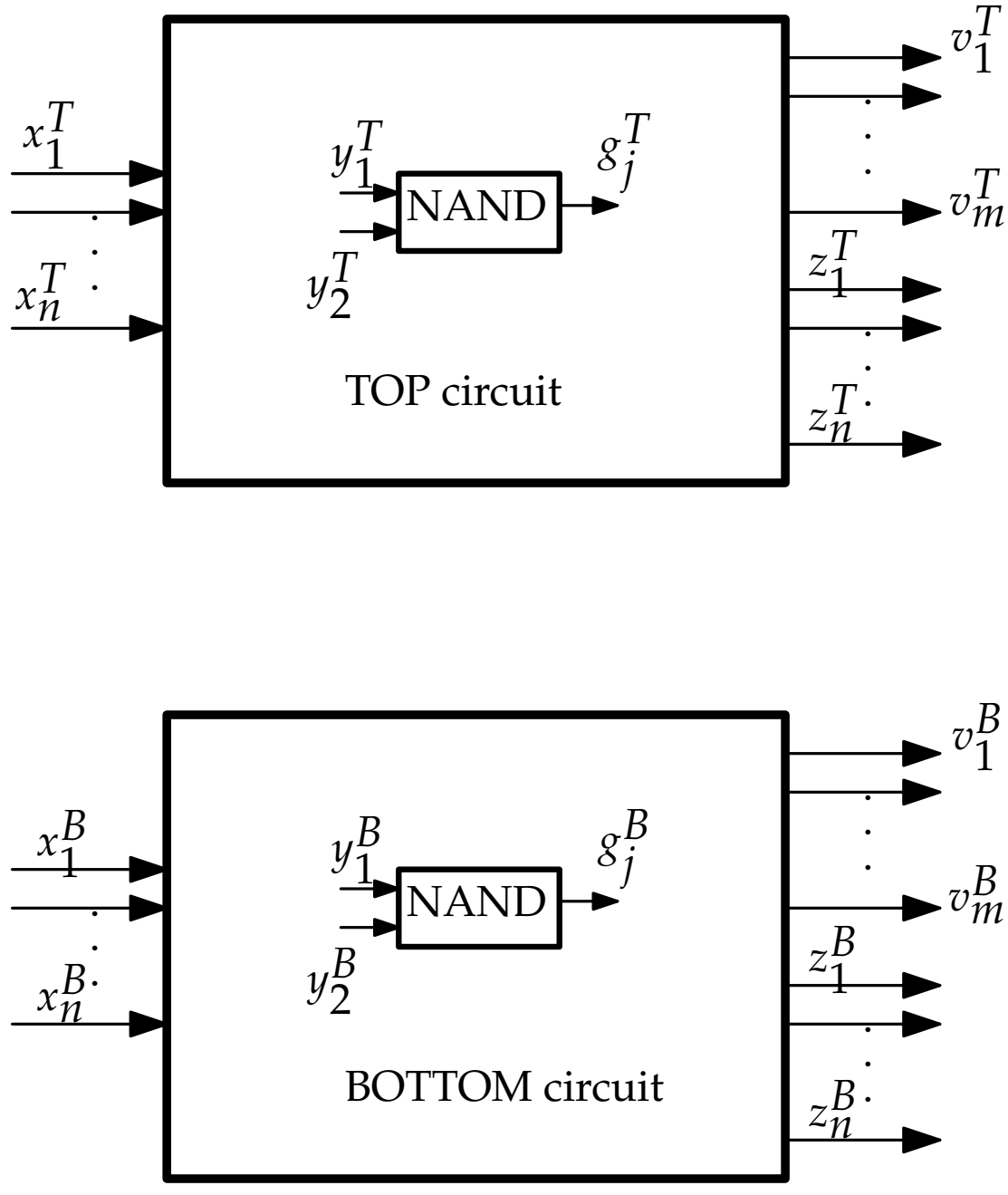


Want: Once $\vec{v}$ and $\vec{z}$ are correct, the "user" should have an incentive to flip the x_j to match the z_j to iterate the evaluation. But this will violate gates!

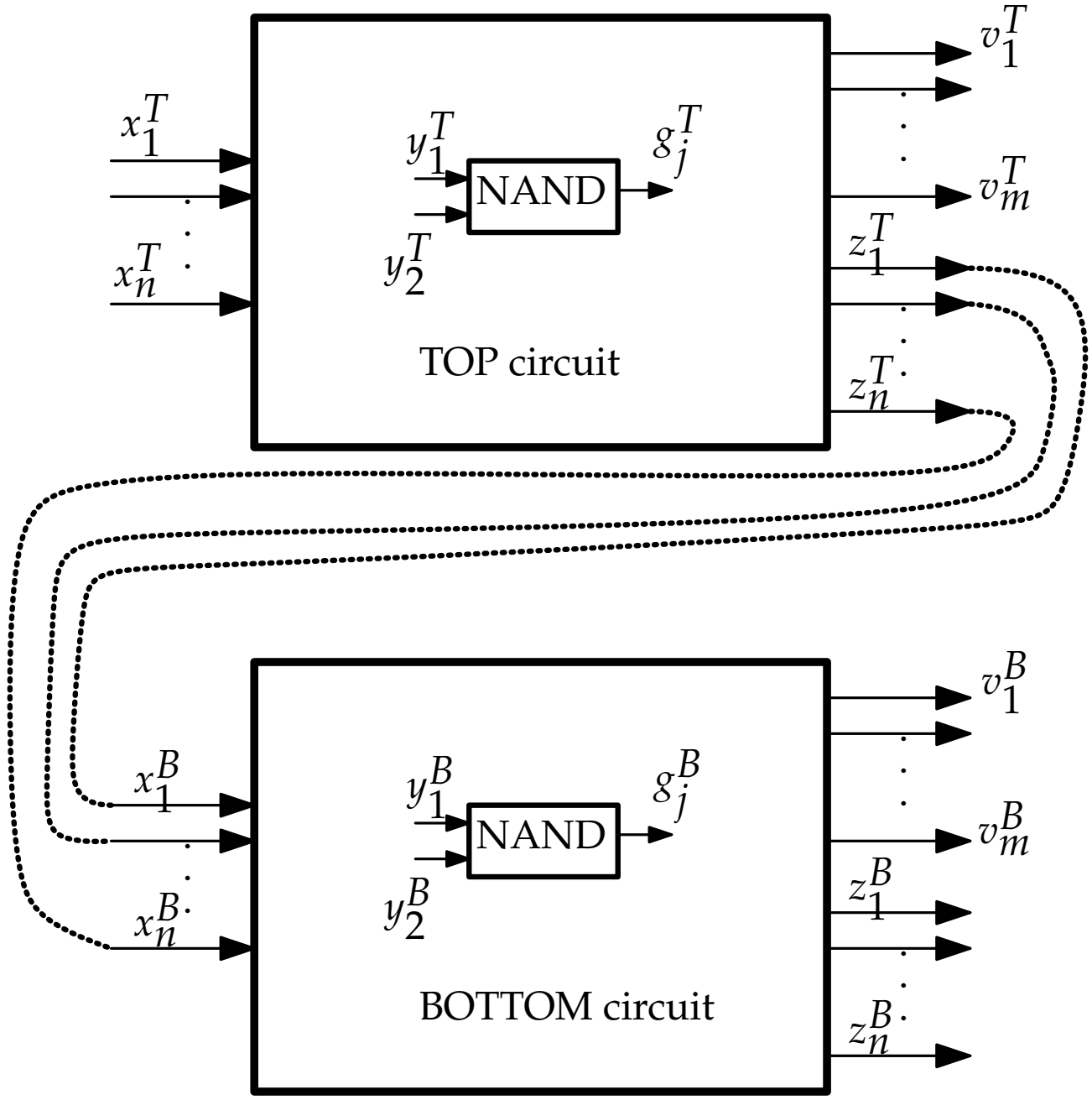
And should only work if $\text{val}(\vec{z})$ is better than $\text{val}(\vec{x})$!

But we haven't evaluated $\text{val}(\vec{z})$ yet. How could we?

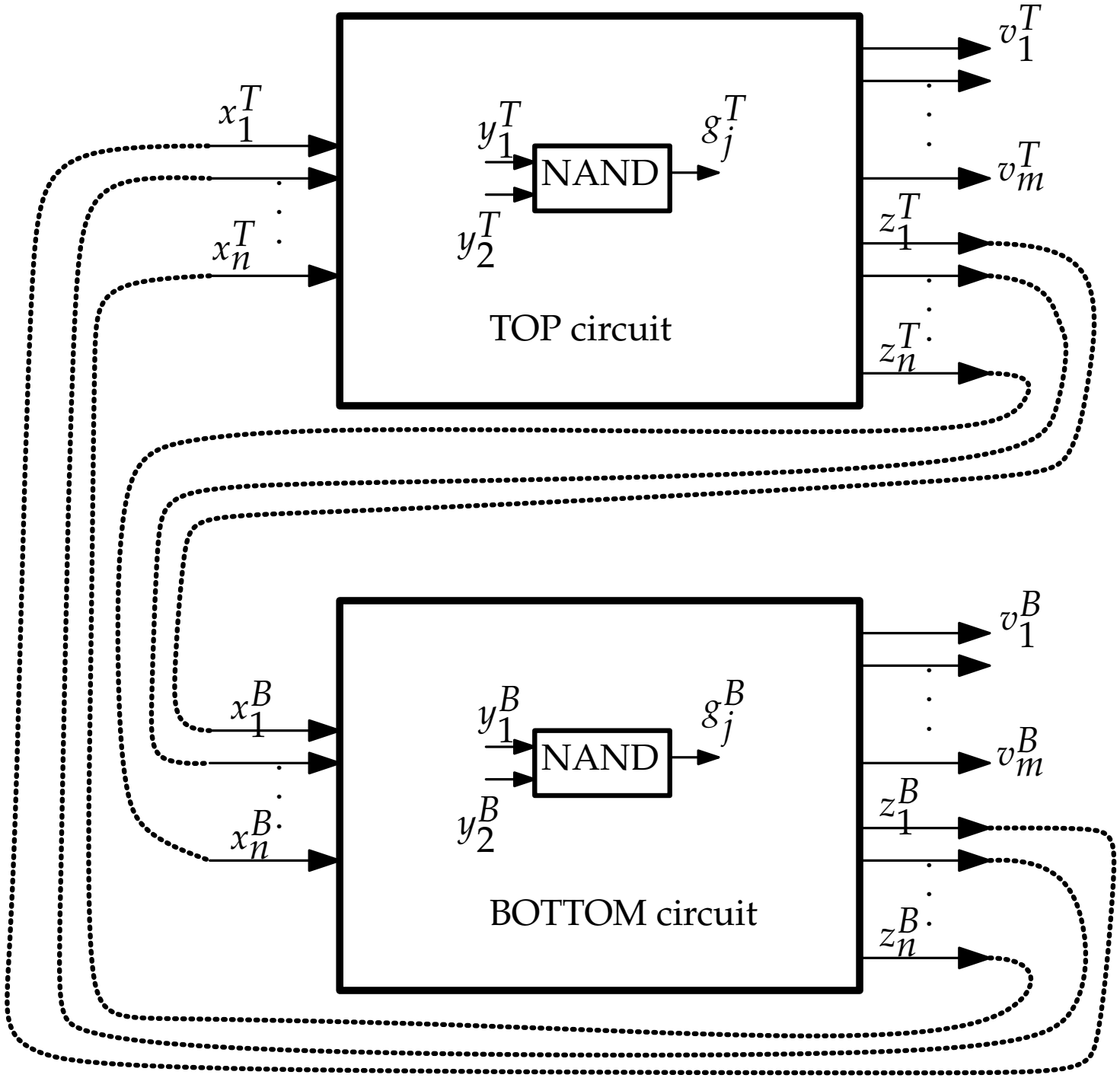
Krentels' Circuit Layout



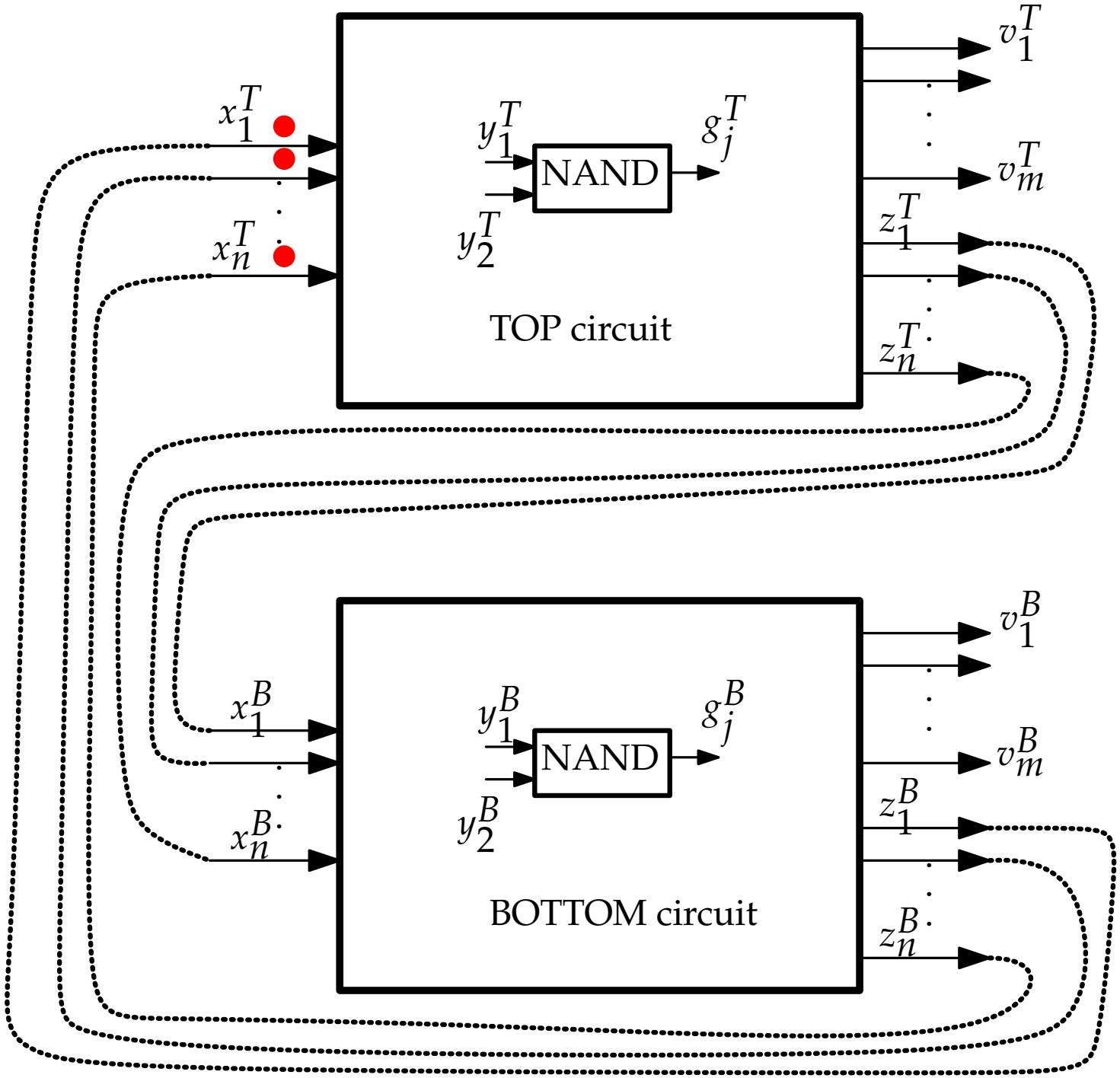
Krentels' Circuit Layout



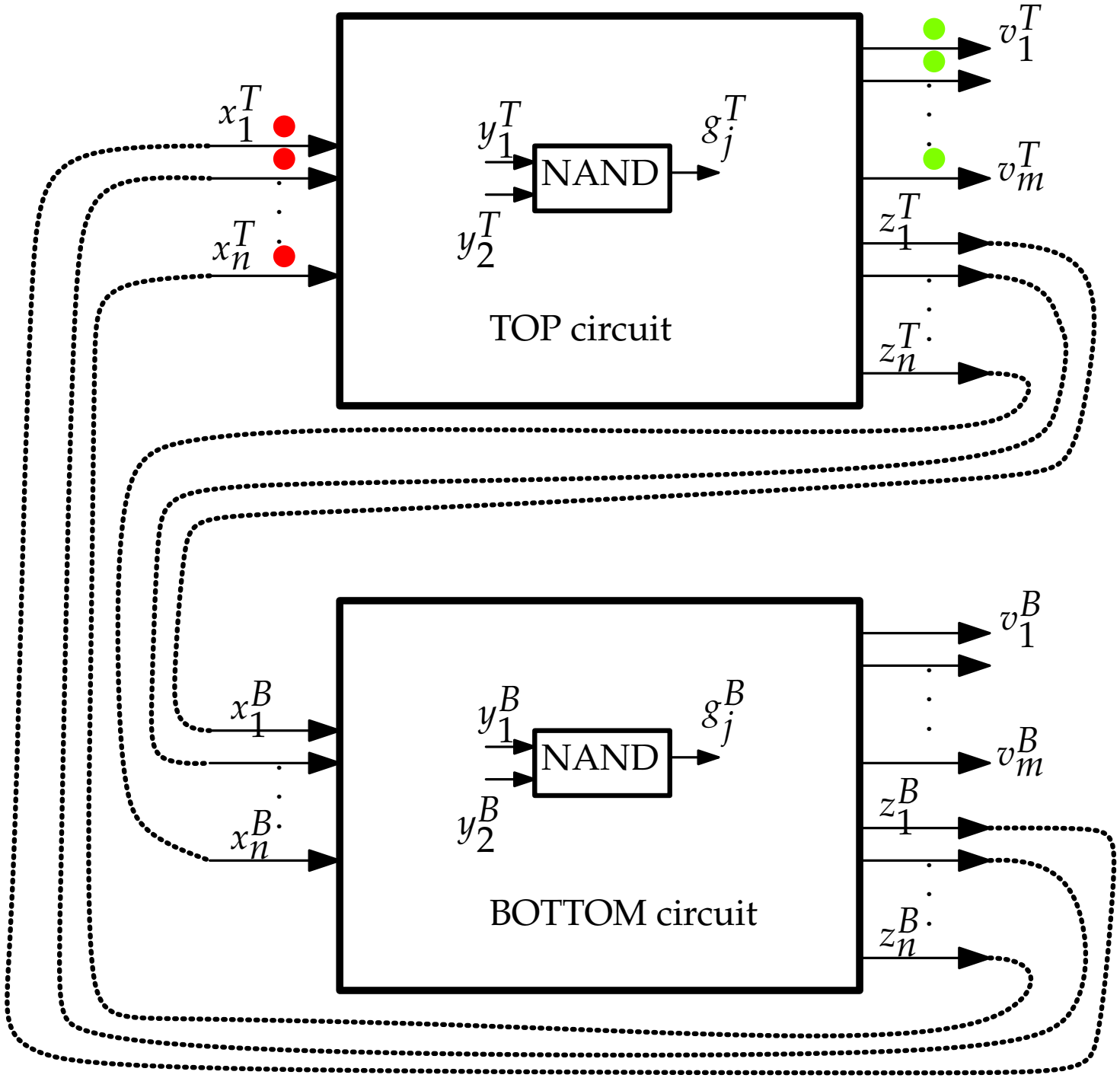
Krentels' Circuit Layout



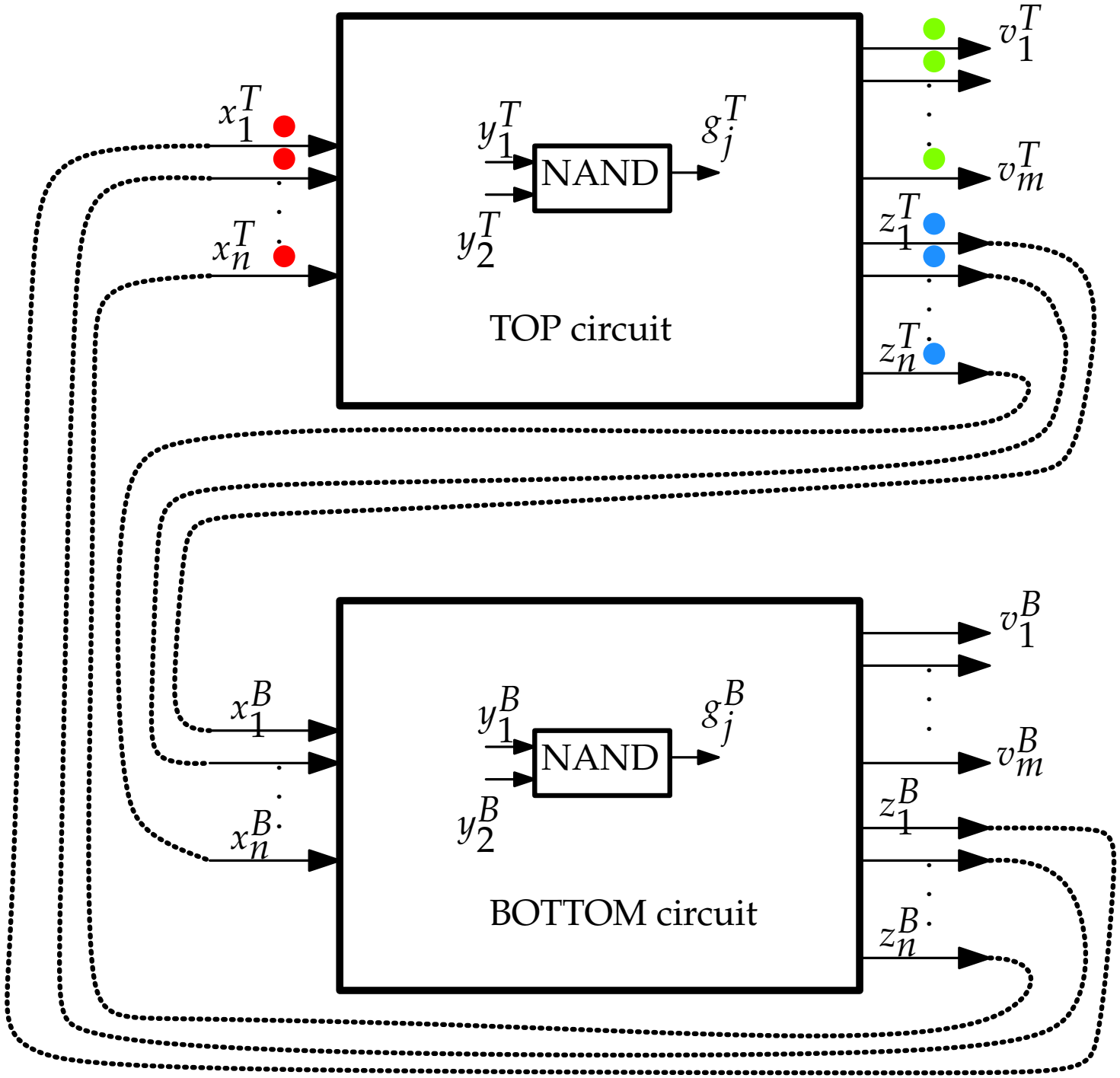
Krentels' Circuit Layout



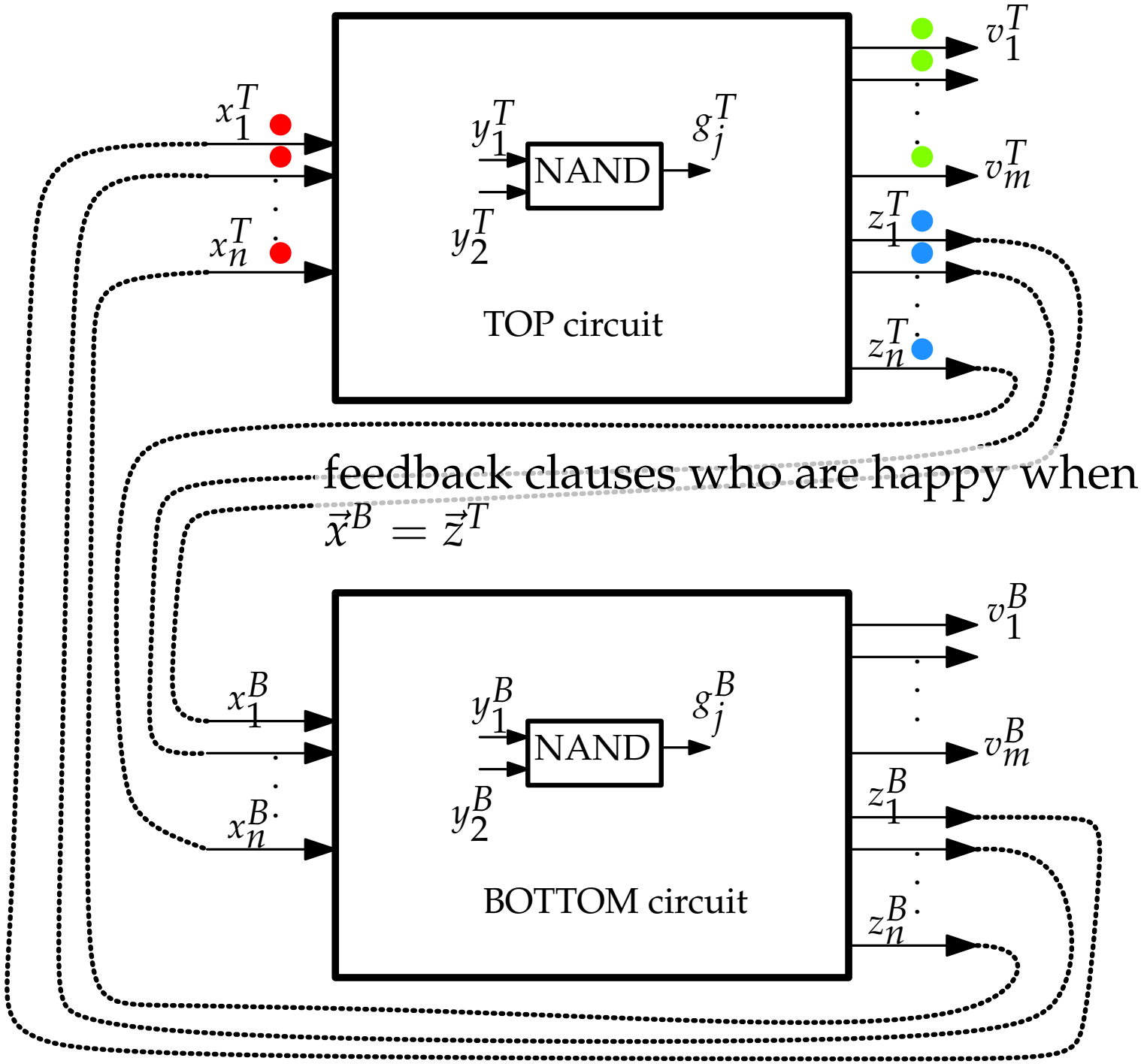
Krentels' Circuit Layout



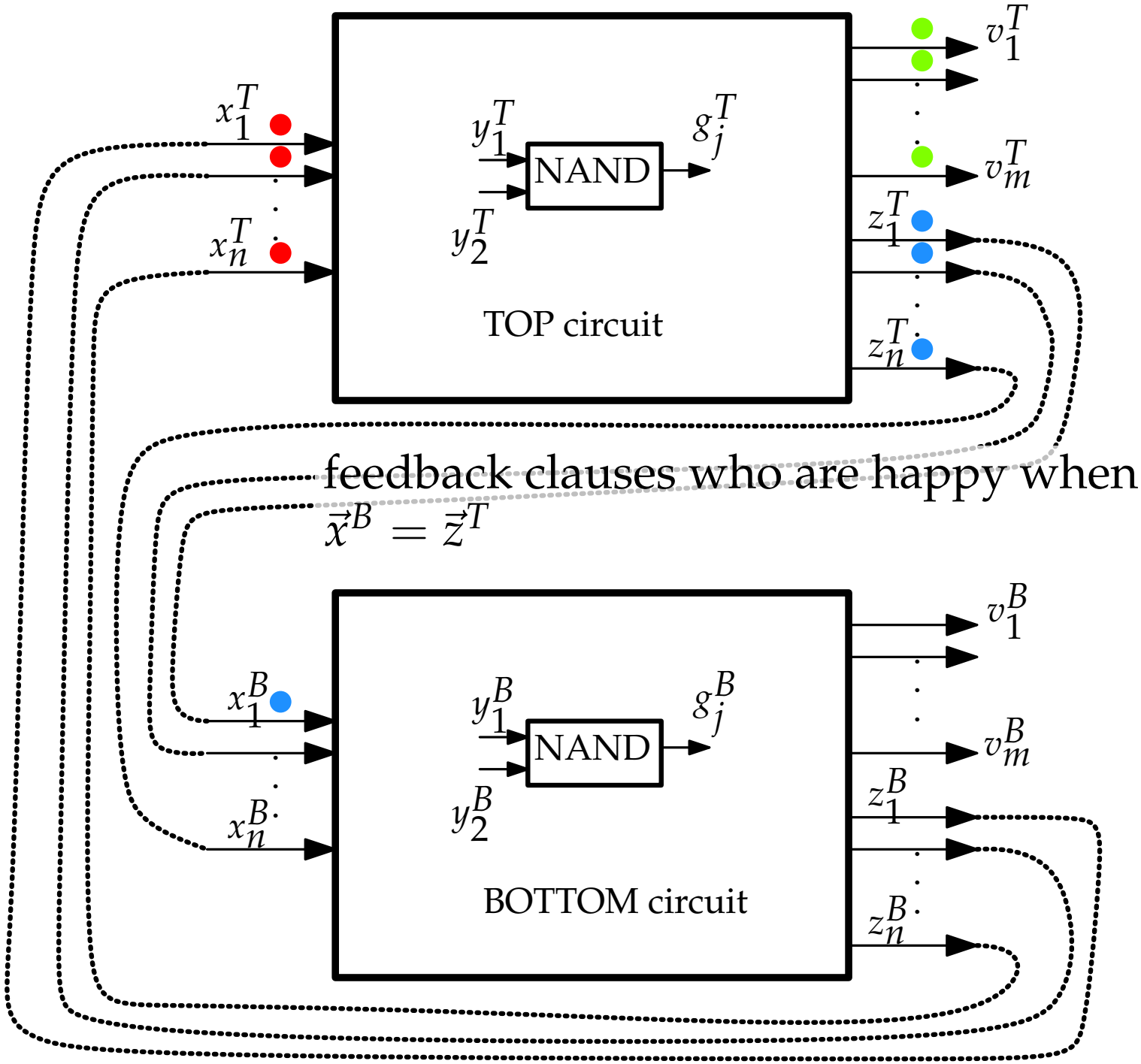
Krentels' Circuit Layout



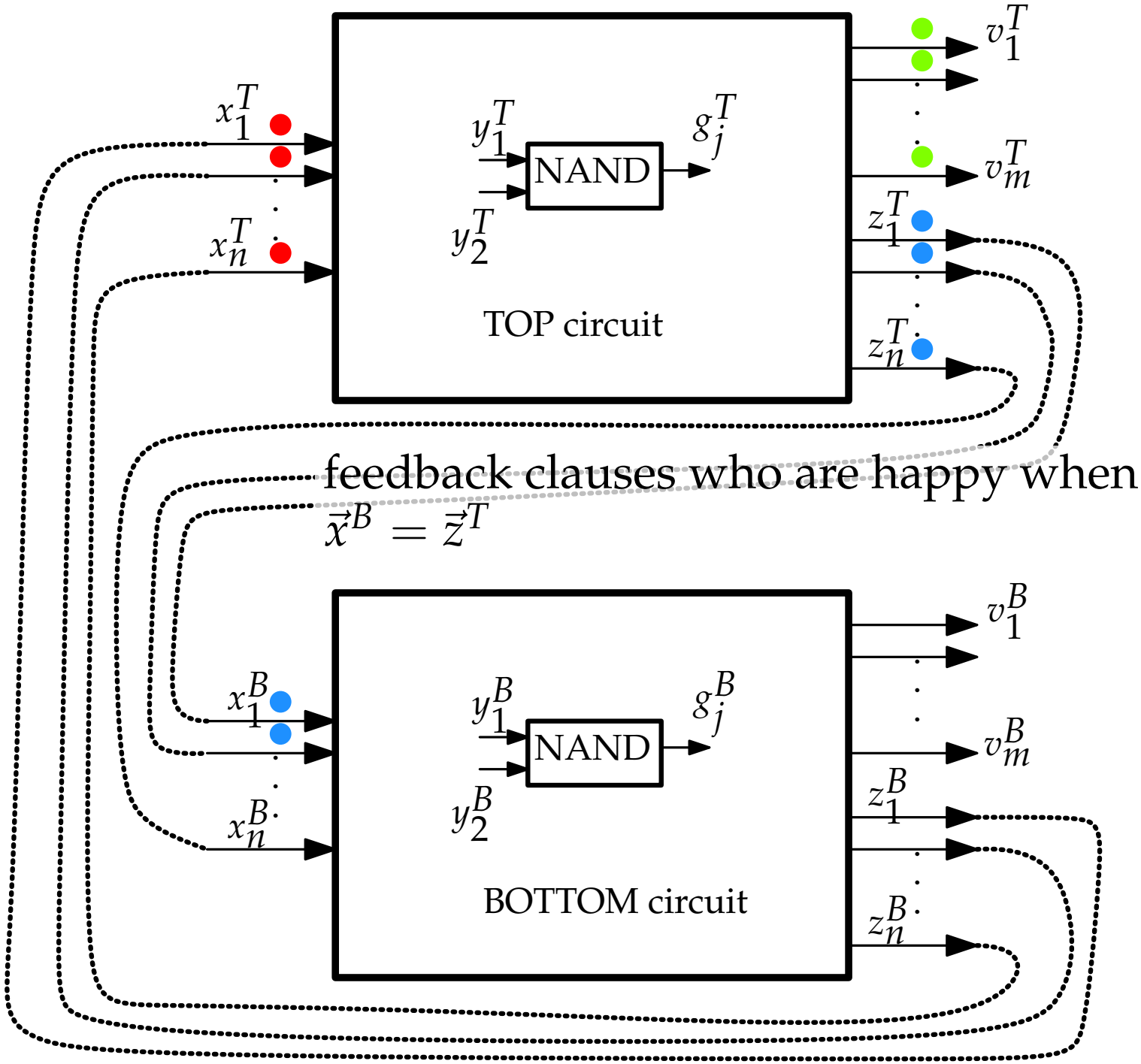
Krentels' Circuit Layout



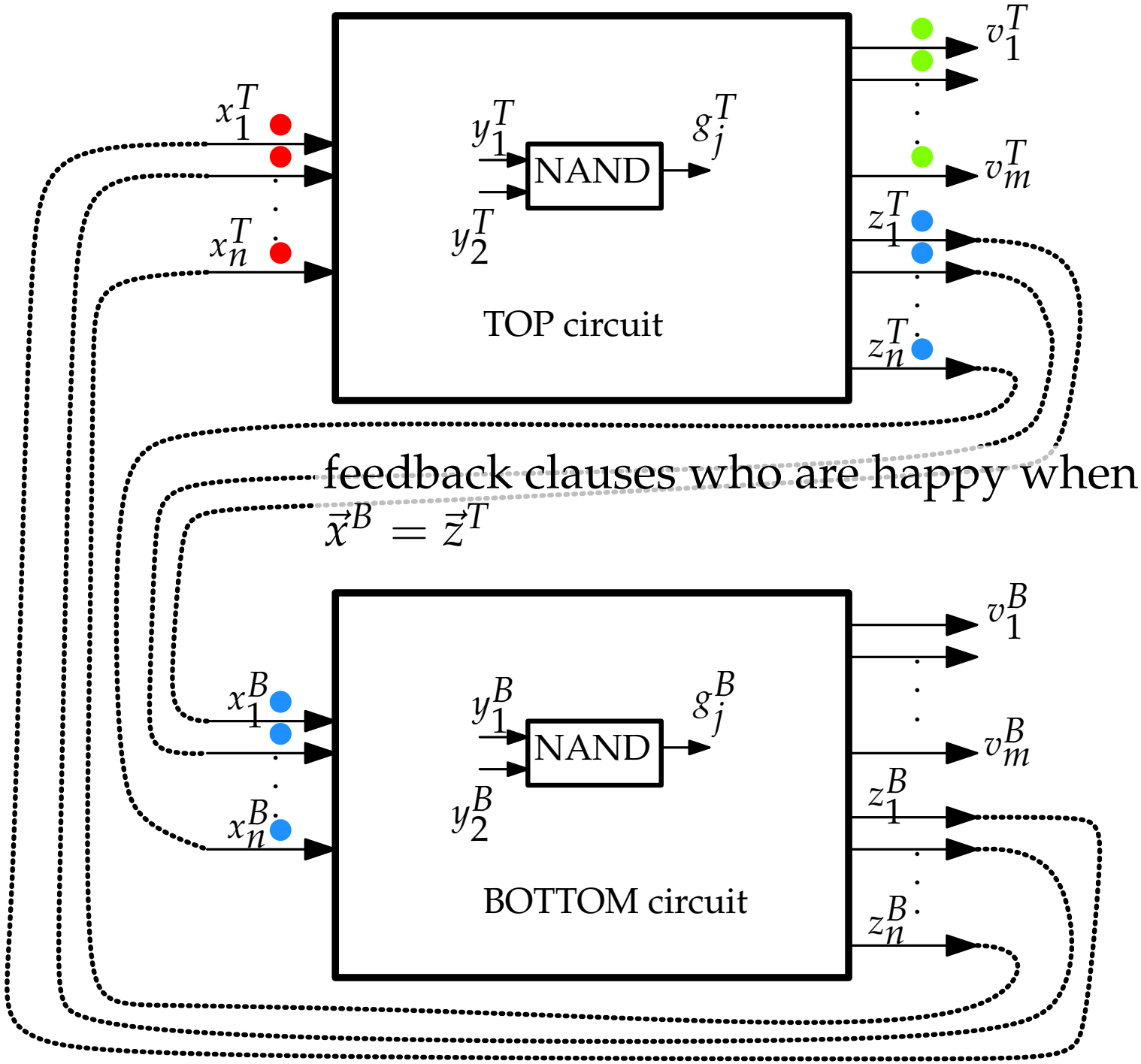
Krentels' Circuit Layout



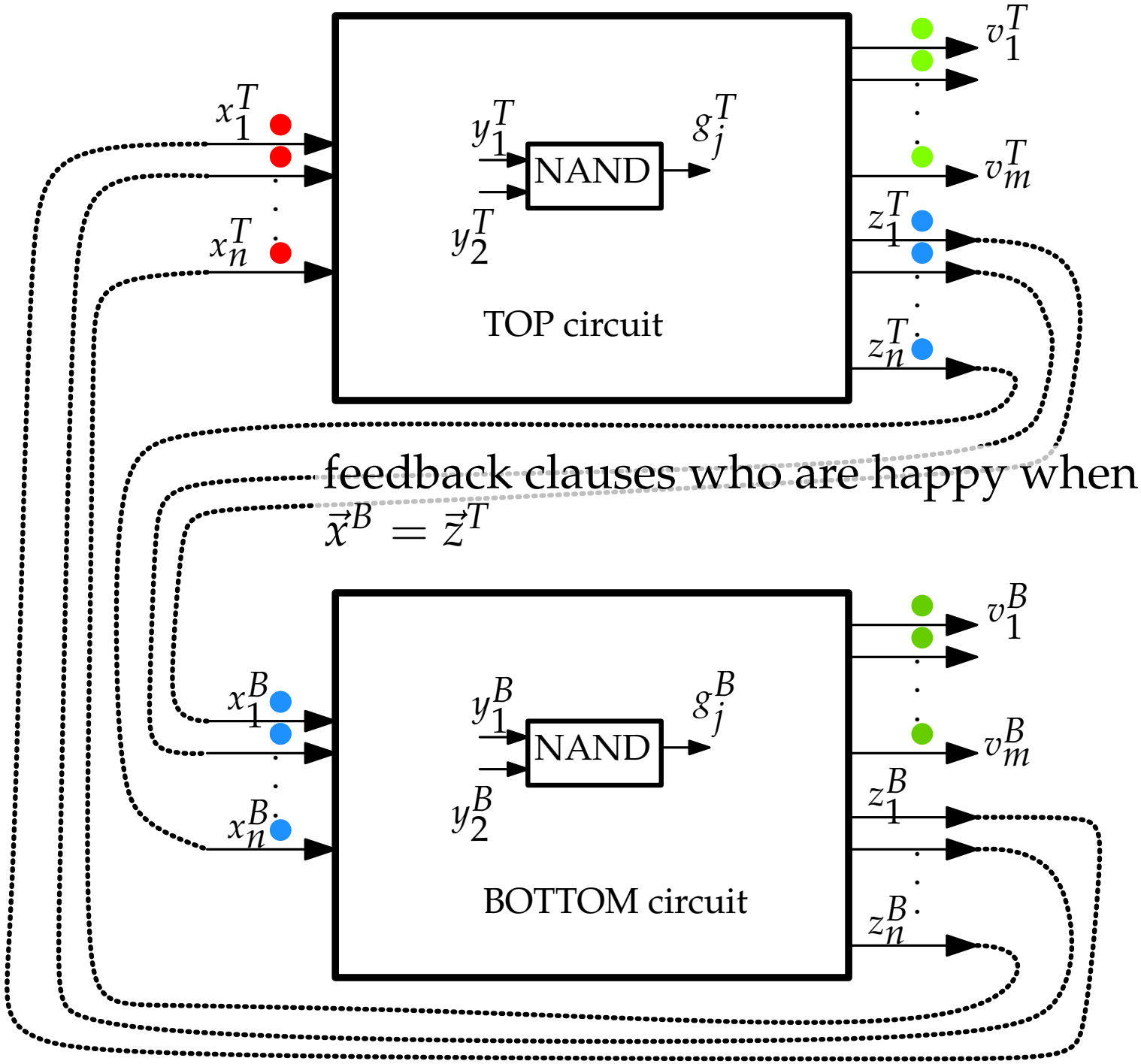
Krentels' Circuit Layout



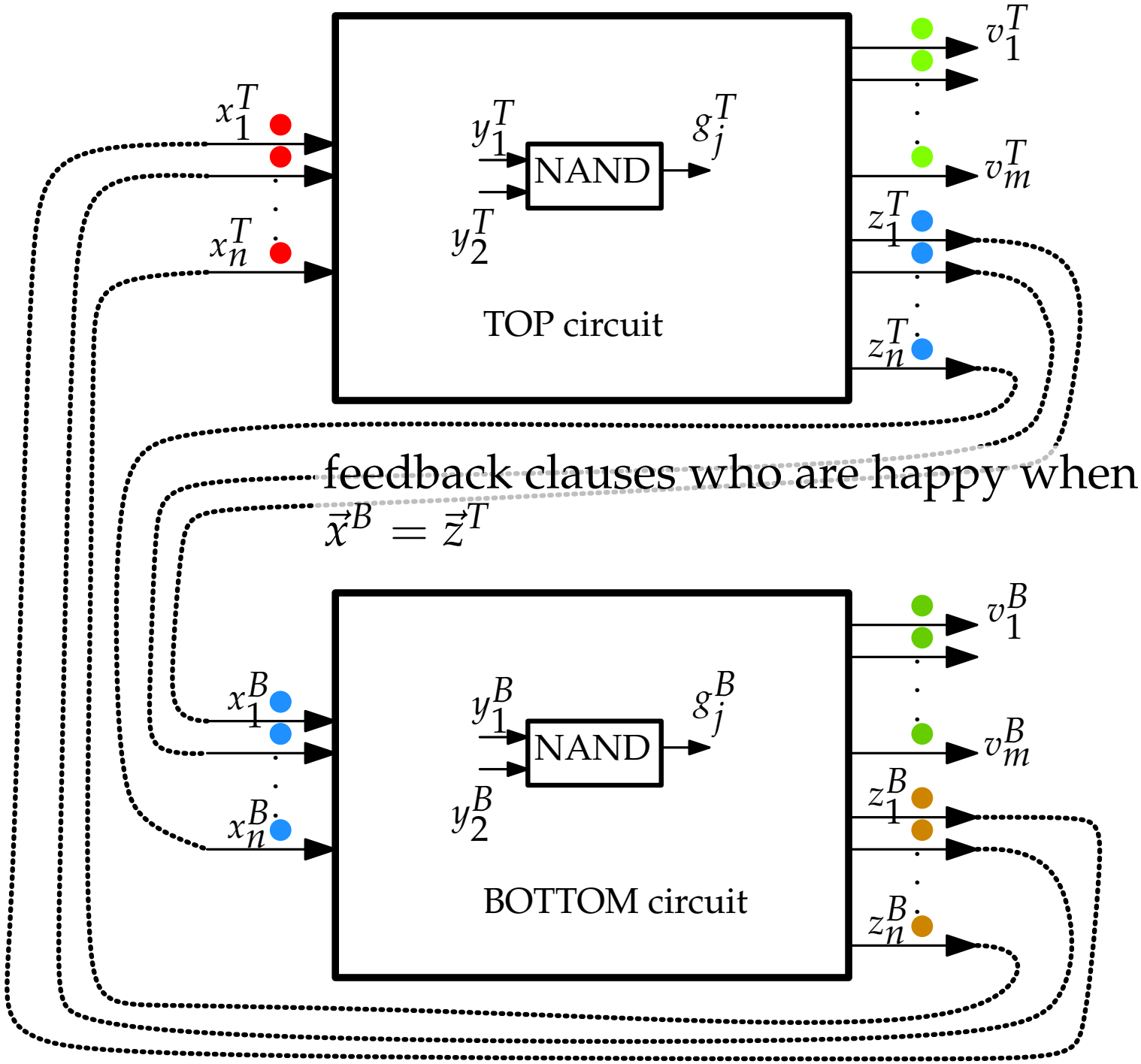
Krentels' Circuit Layout



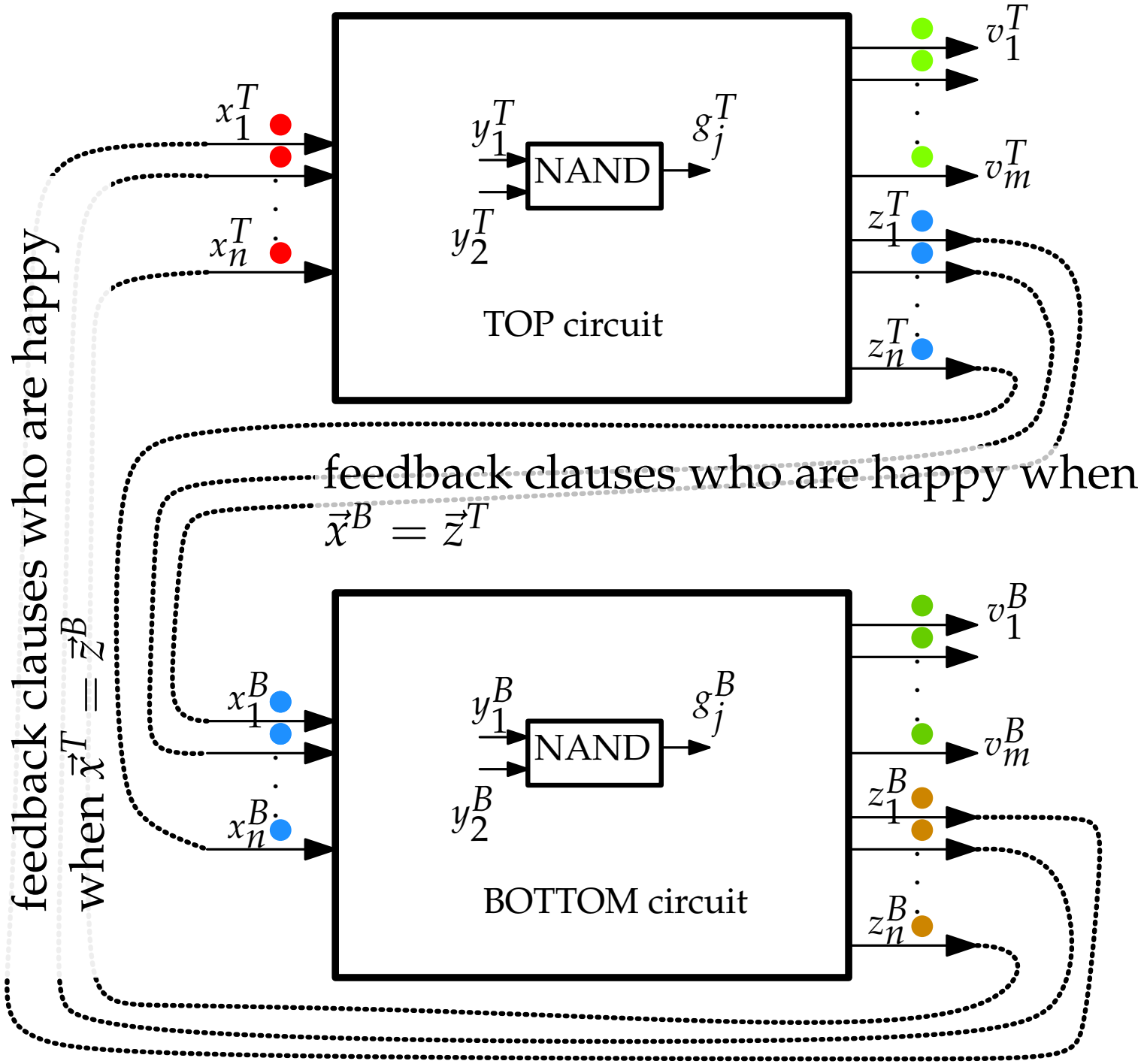
Krentels' Circuit Layout

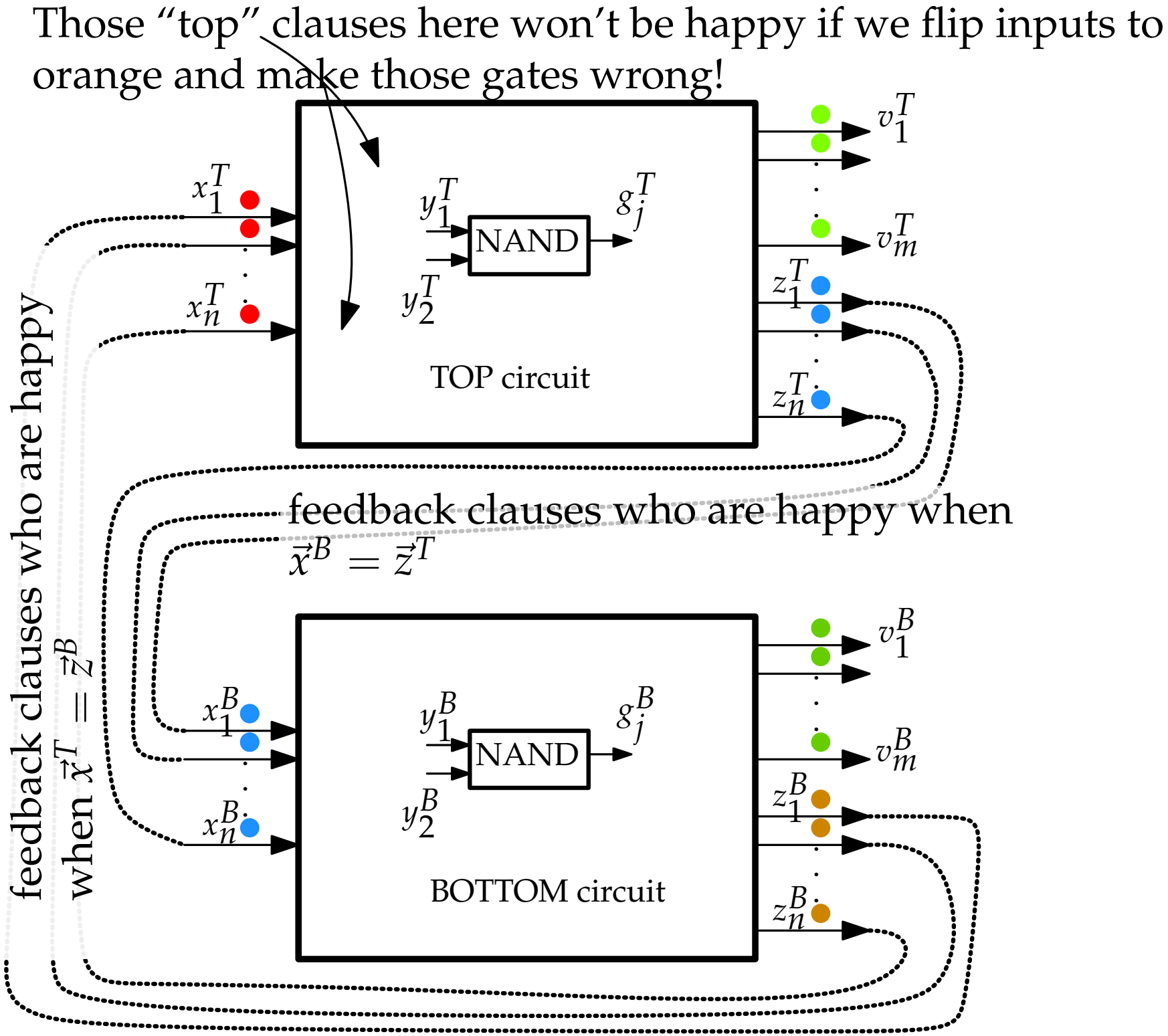


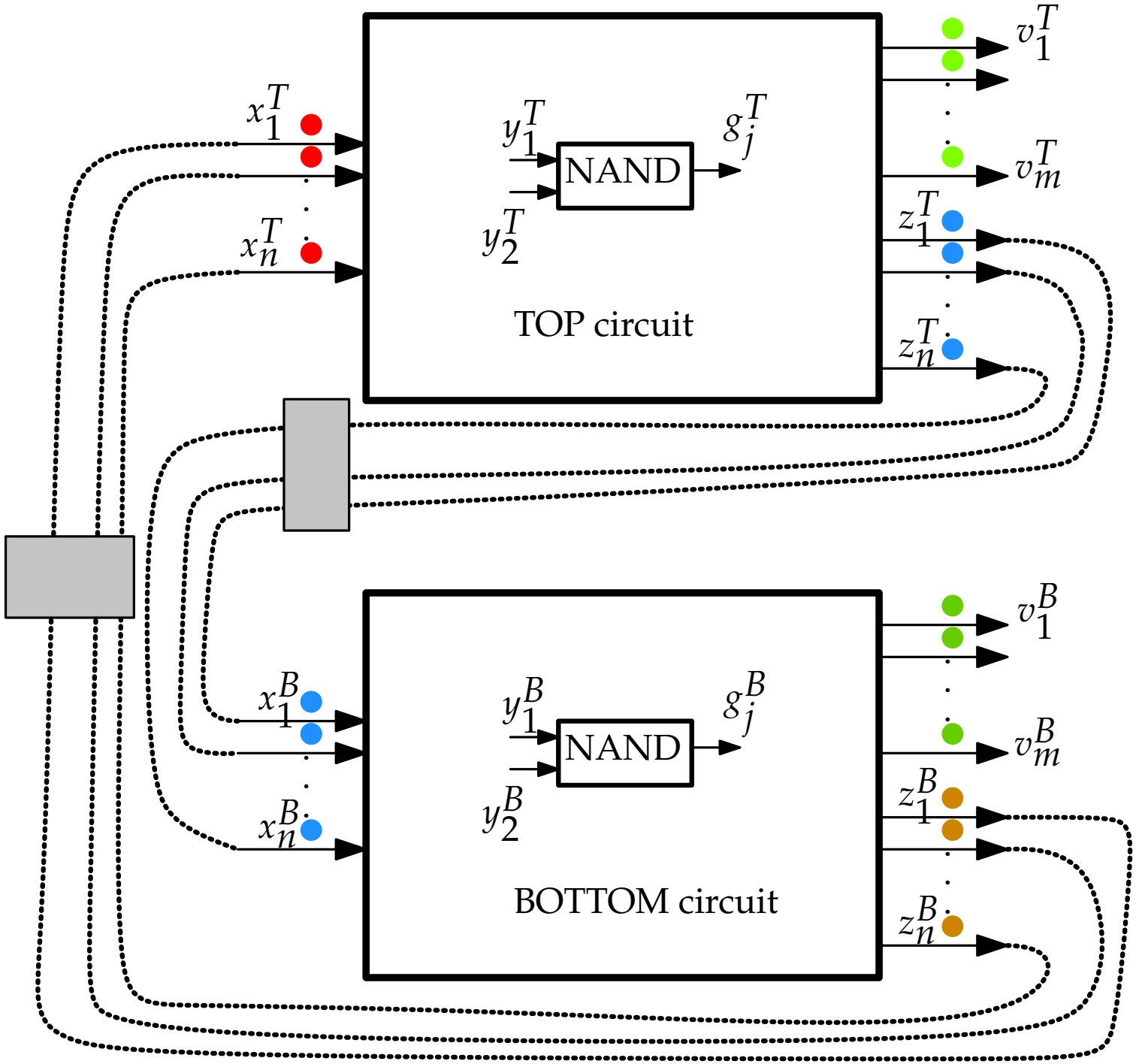
Krentels' Circuit Layout

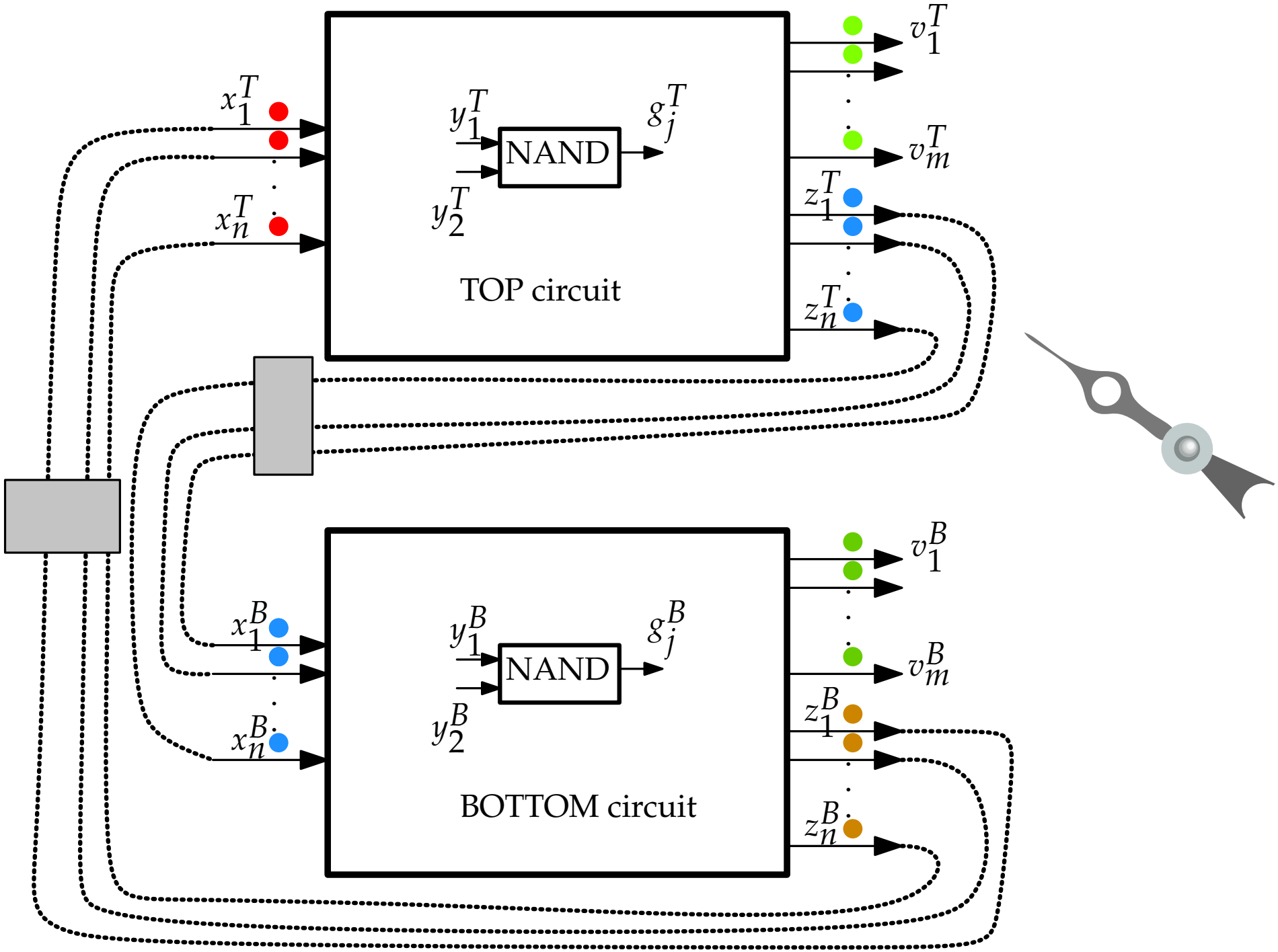


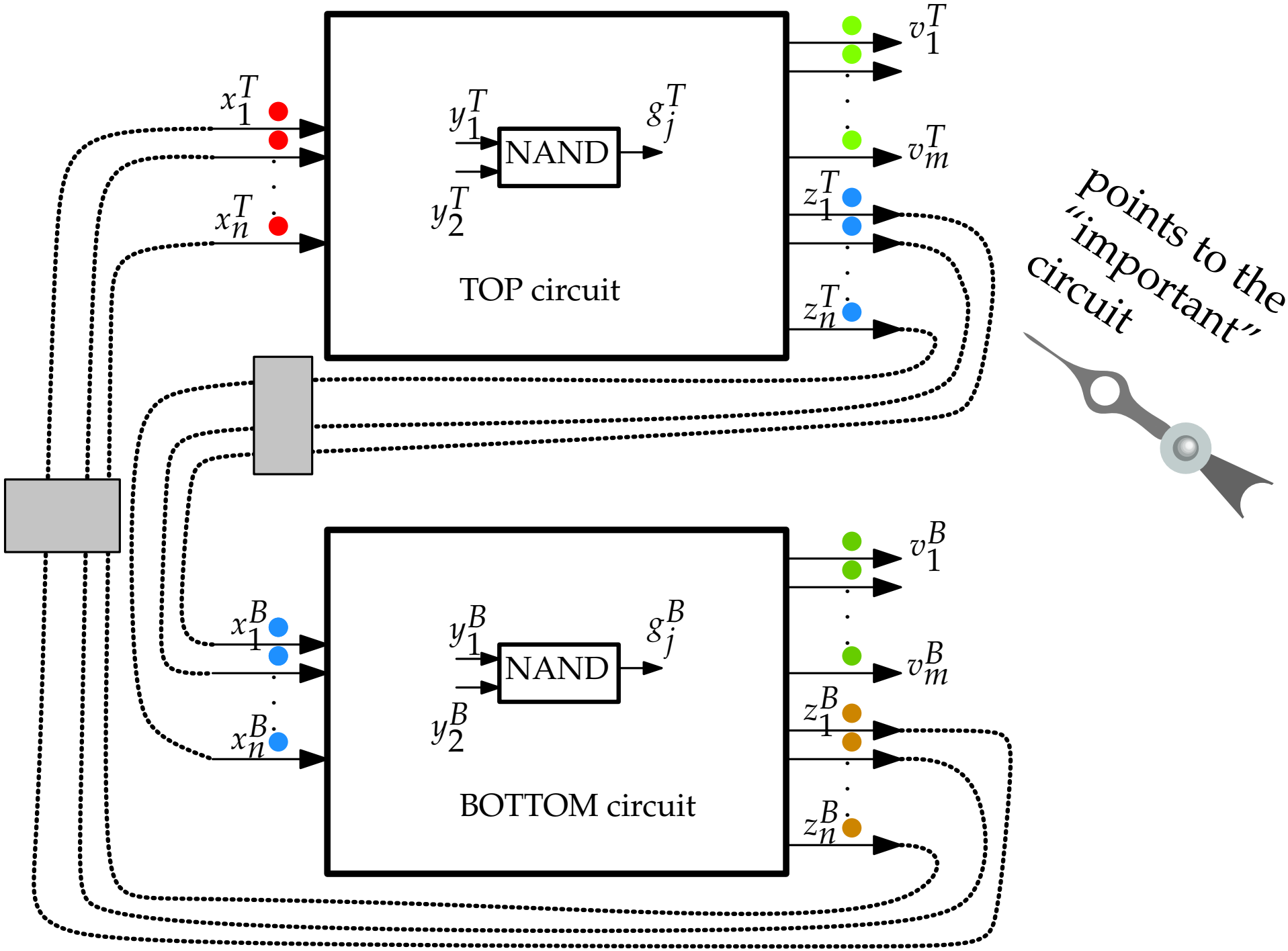
Krentels' Circuit Layout

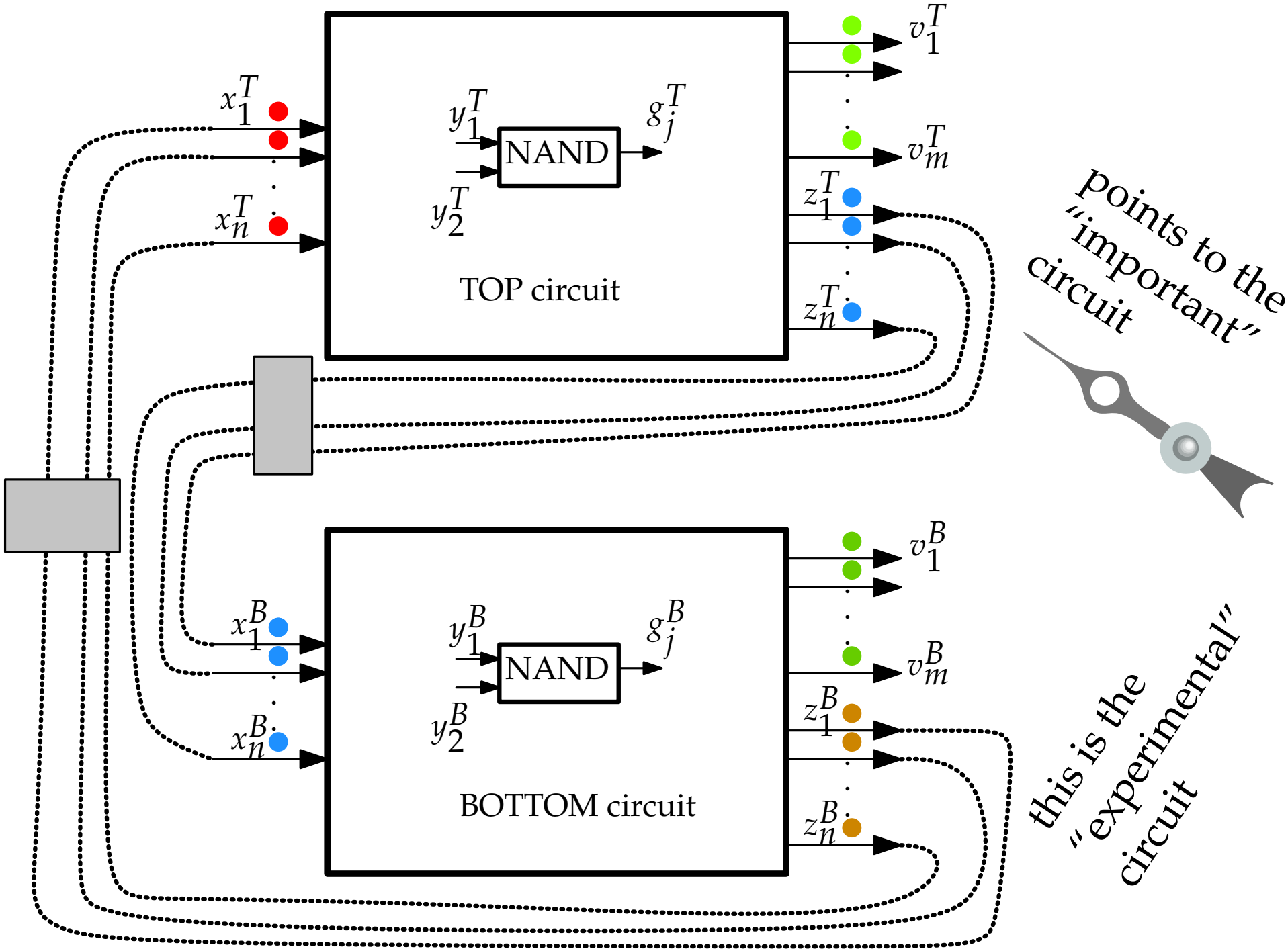


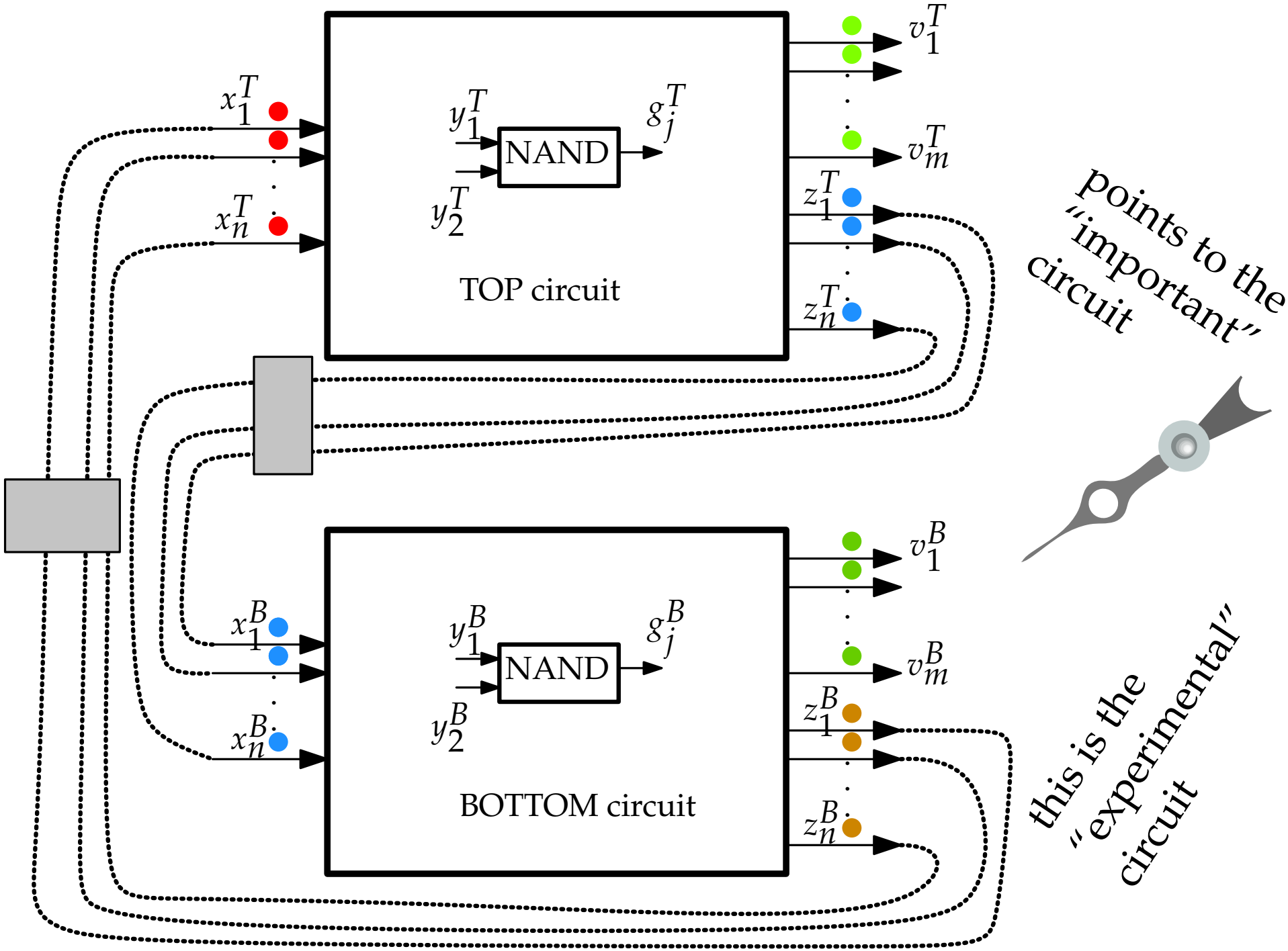

Those "top" clauses here won't be happy if we flip inputs to orange and make those gates wrong!
x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
feedback clauses who are happy when $\vec{x}^B = \vec{z}^T$
feedback clauses who are happy when $\vec{x}^T = \vec{z}^B$
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B

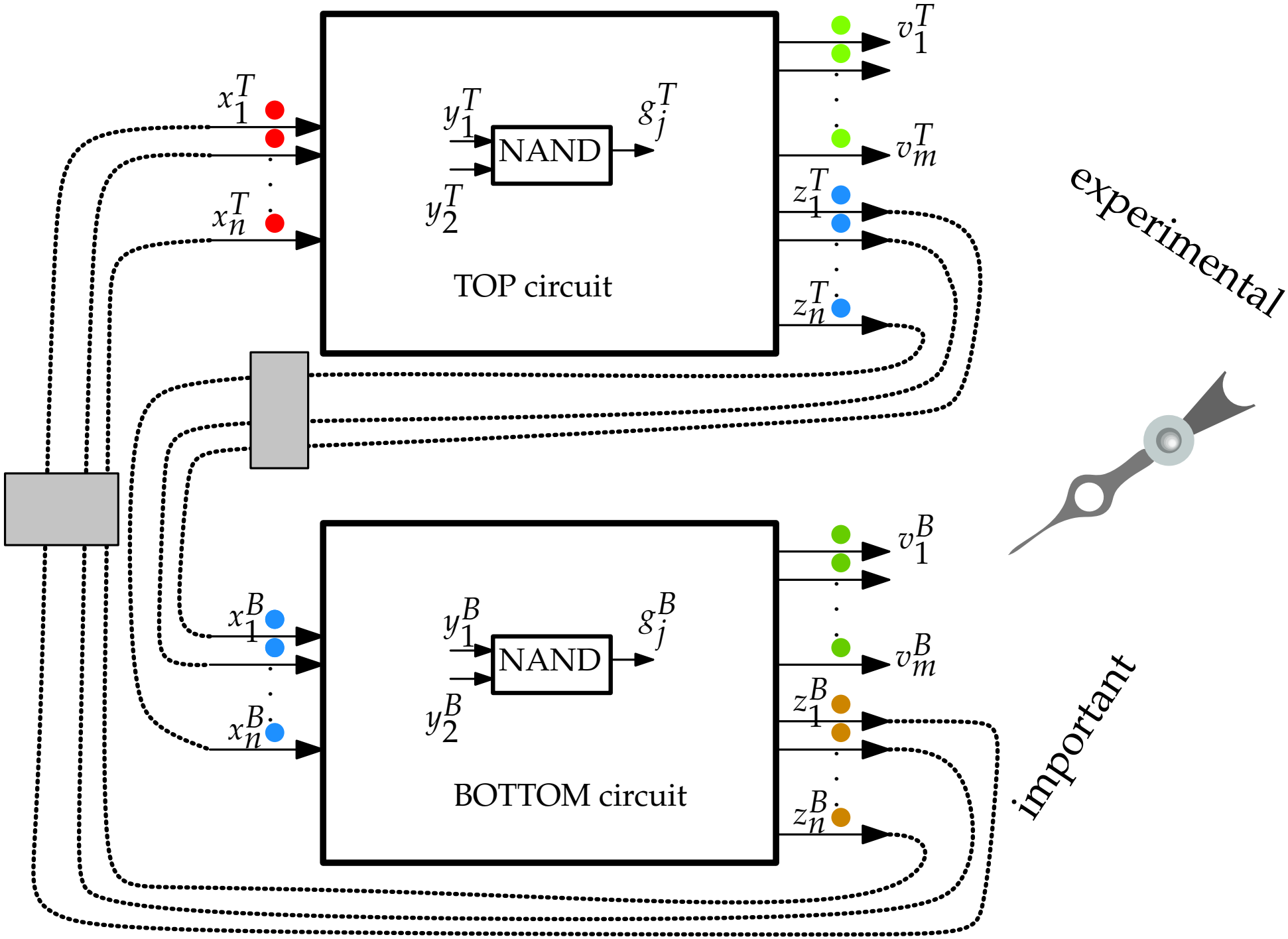

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B

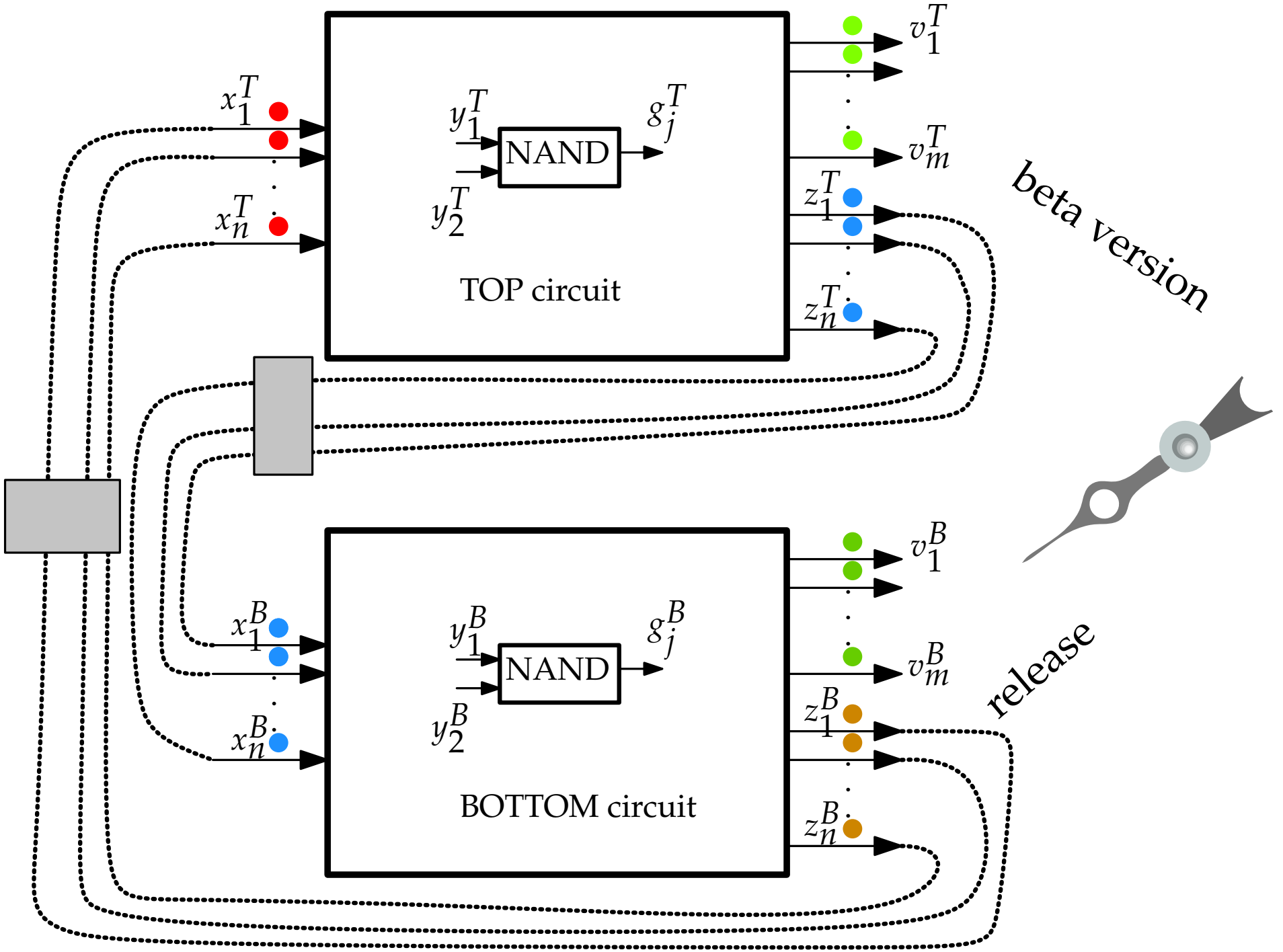

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B

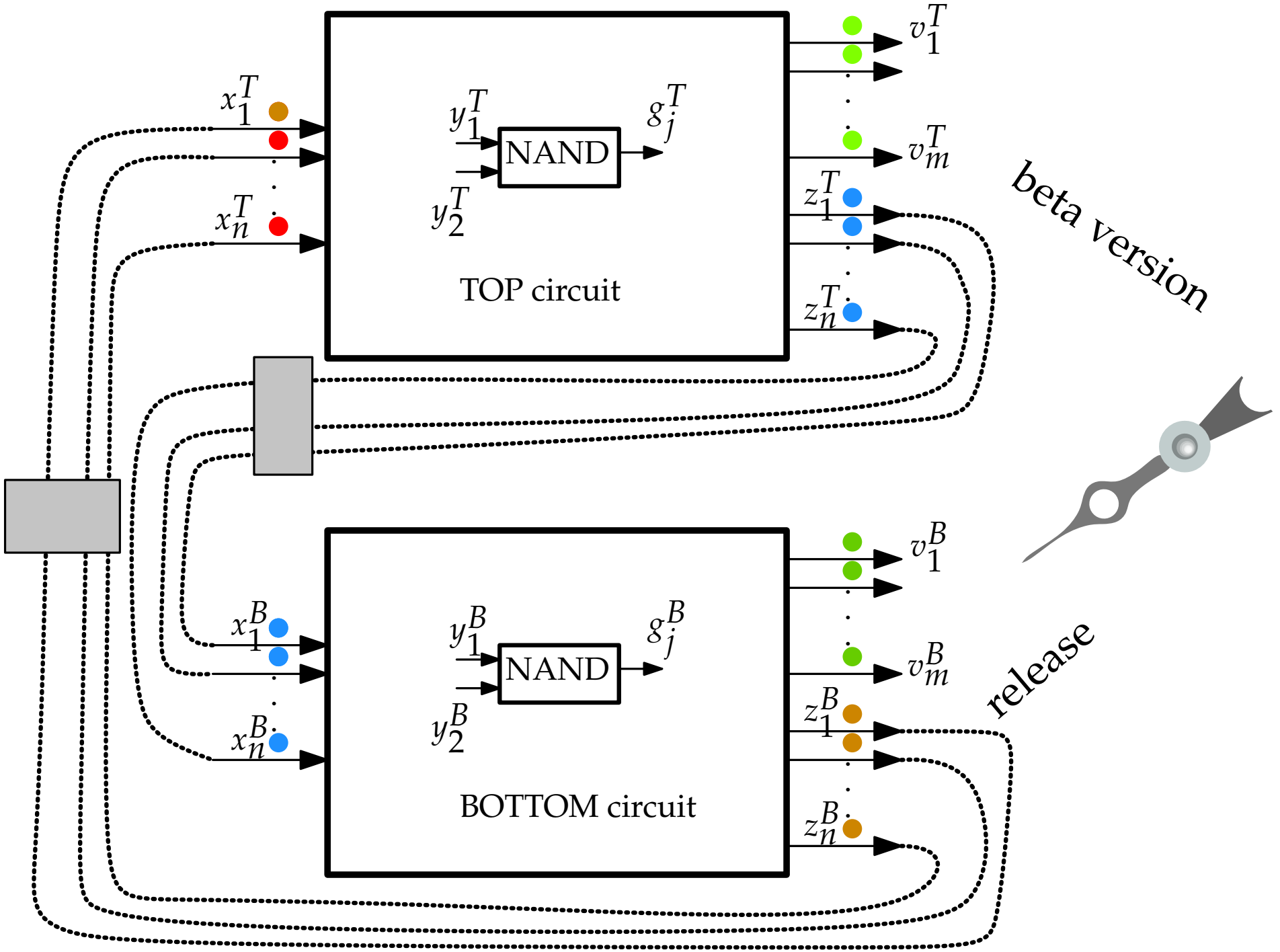

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
points to the "important" circuit
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B

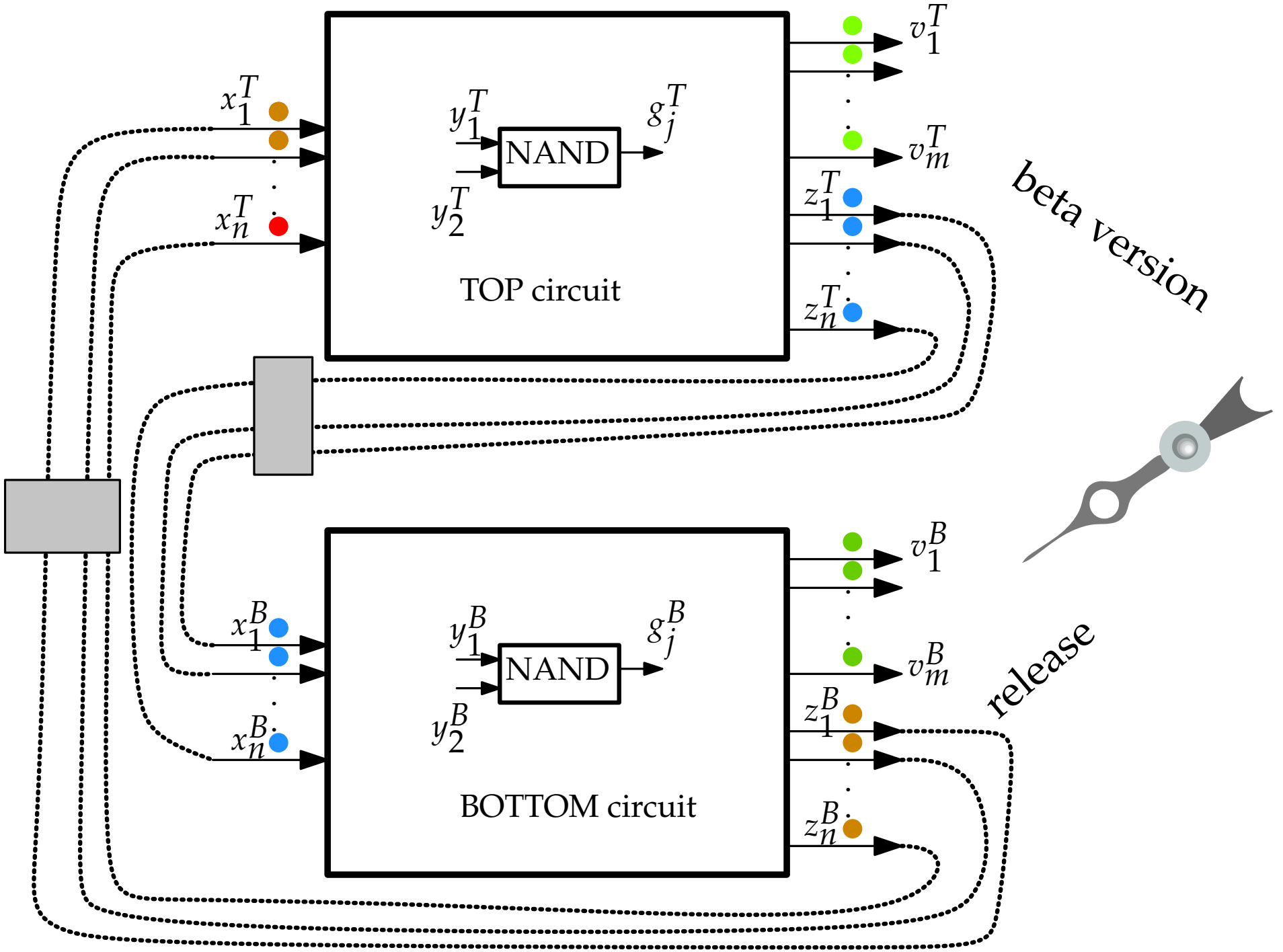

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
points to the "important" circuit
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
this is the "experimental" circuit

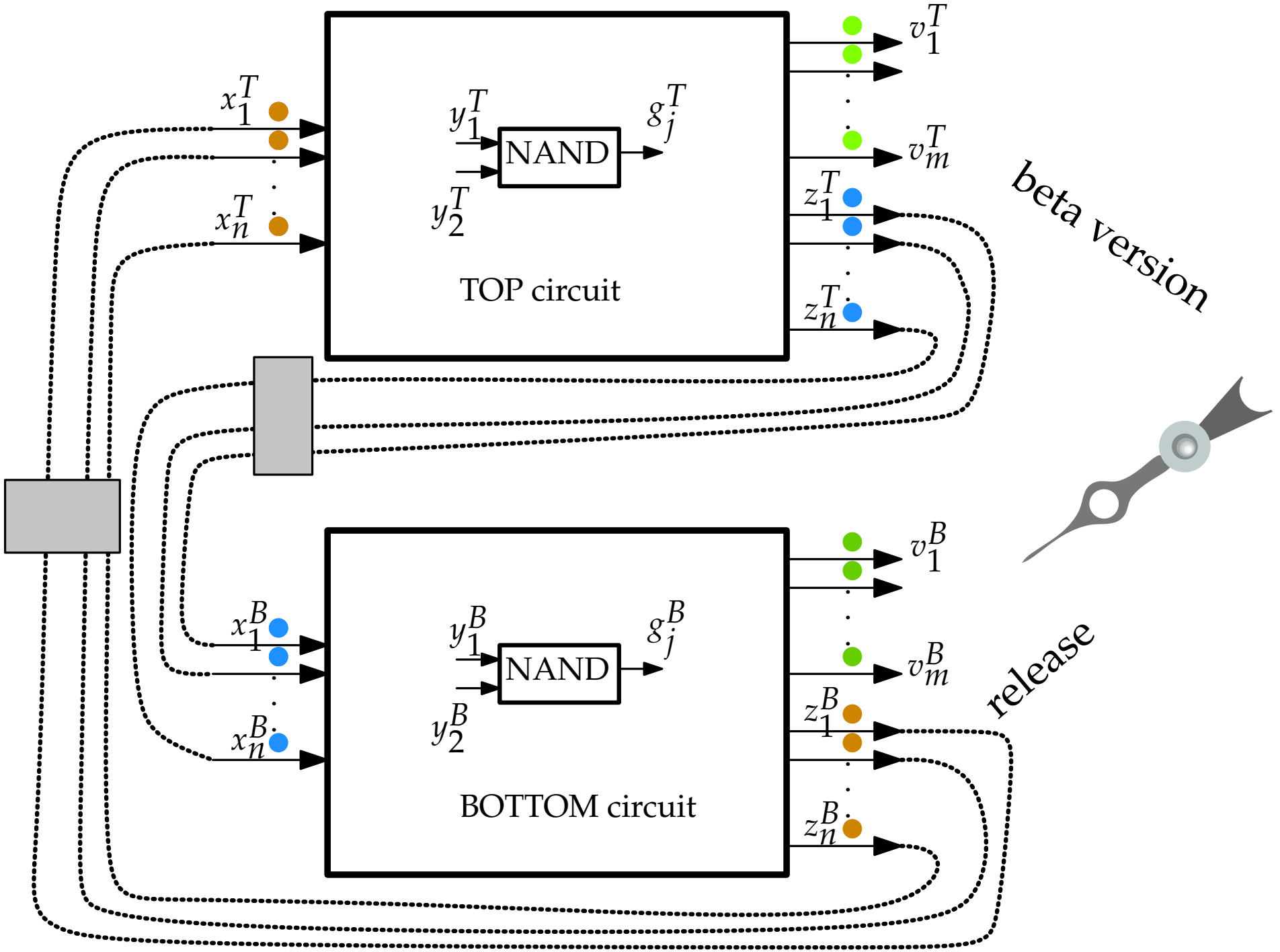

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
points to the "important" circuit
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
this is the "experimental" circuit

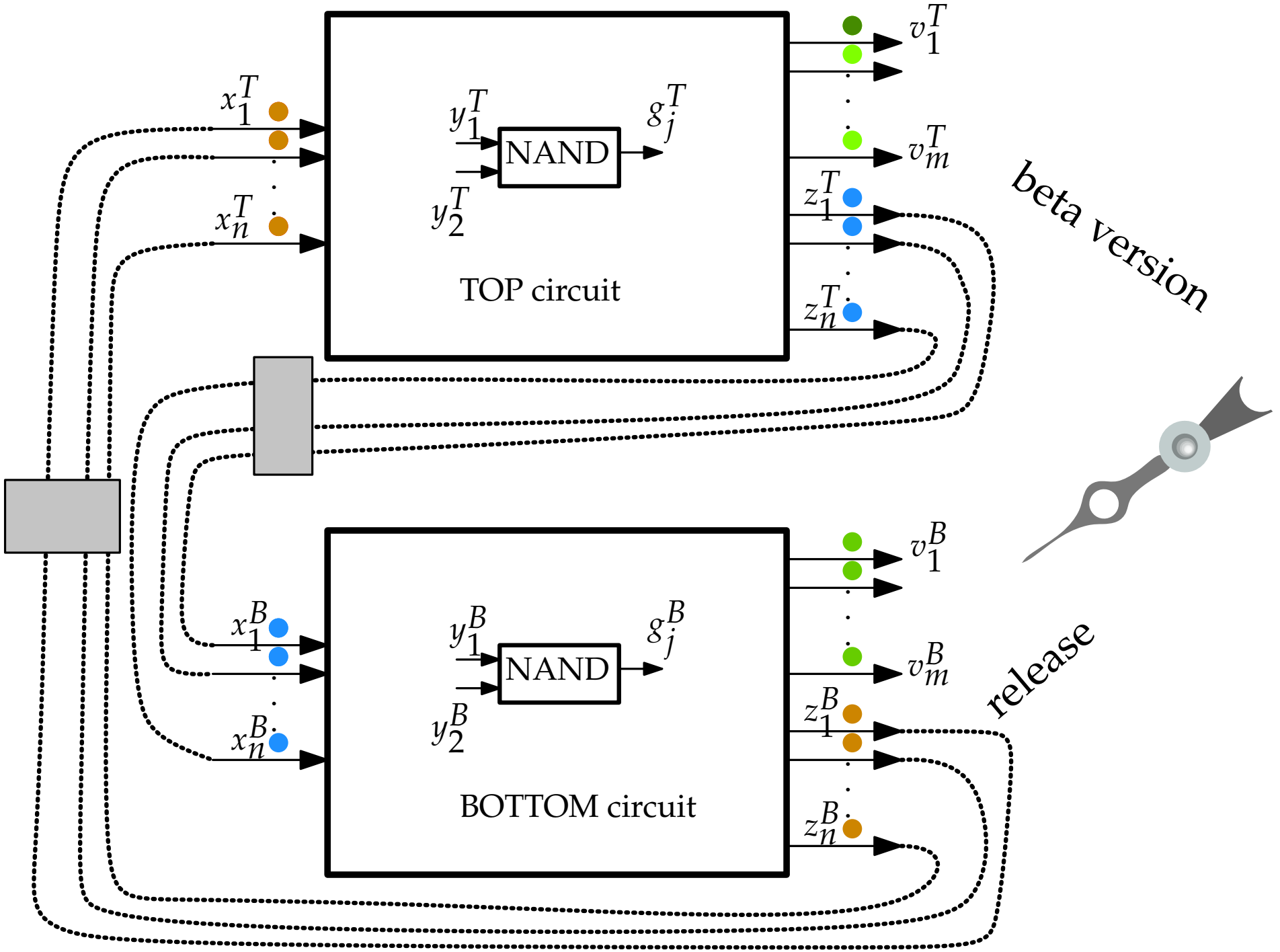

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
experimental
important

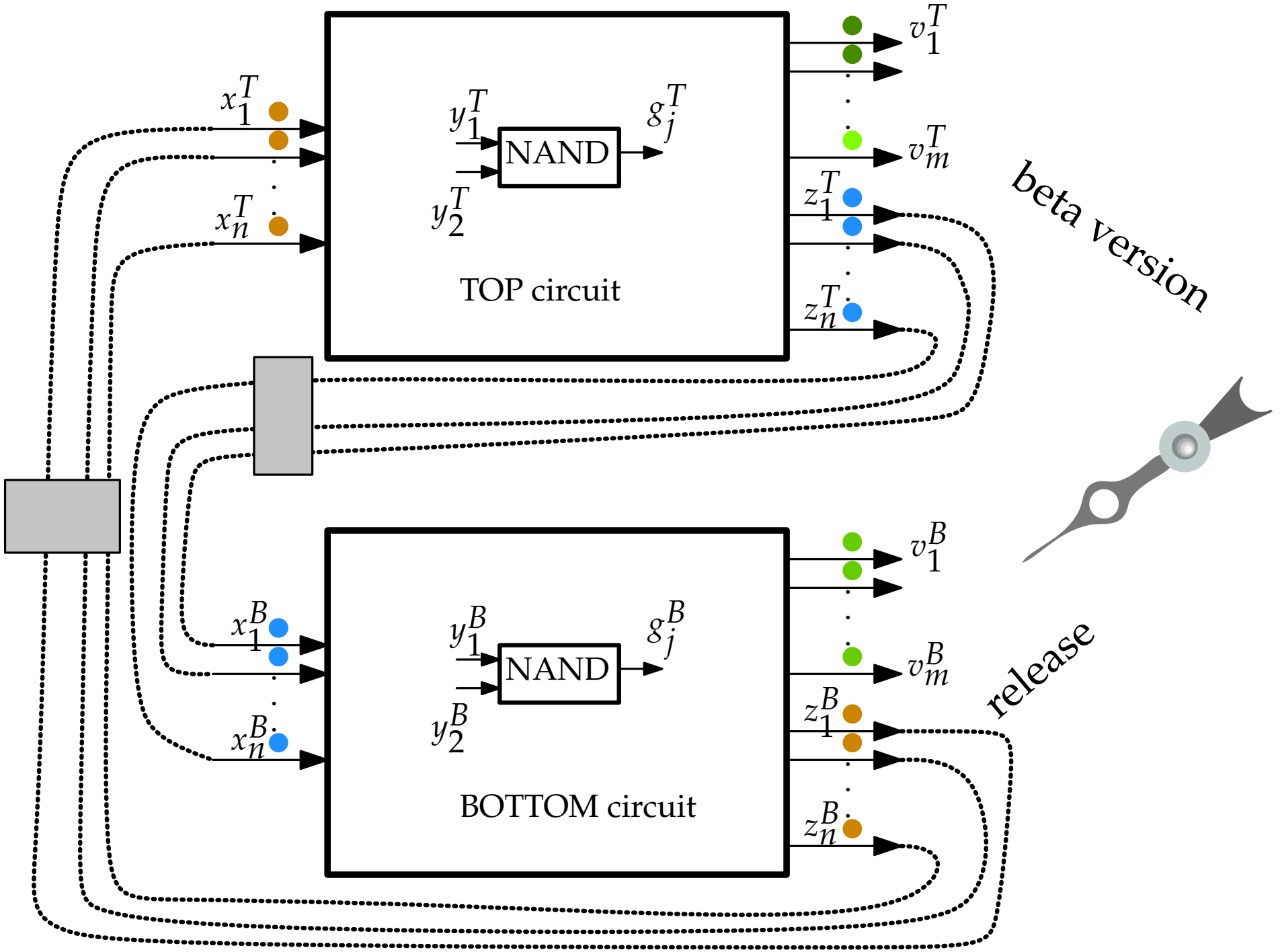

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
beta version
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
release

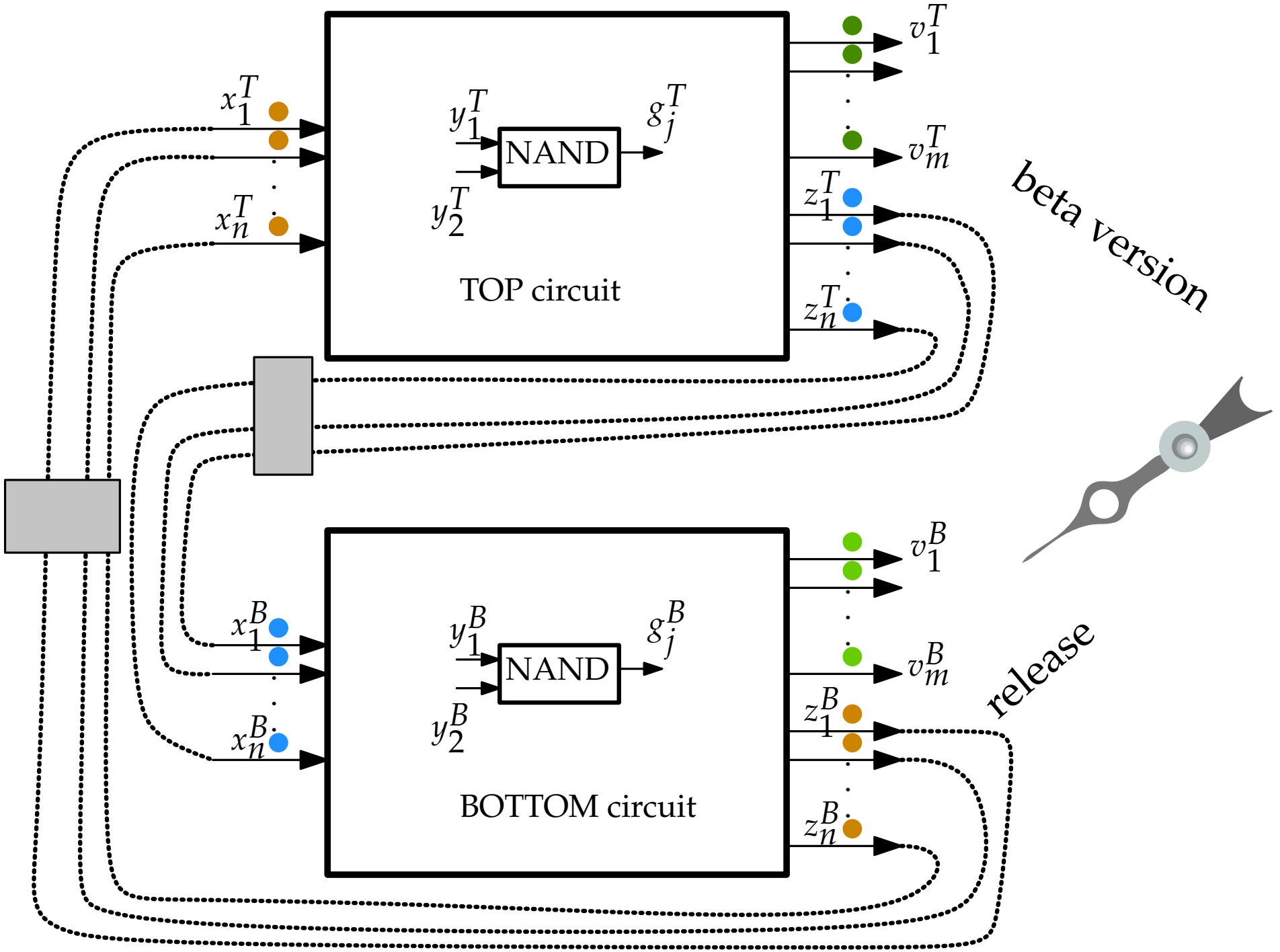

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
beta version
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
release

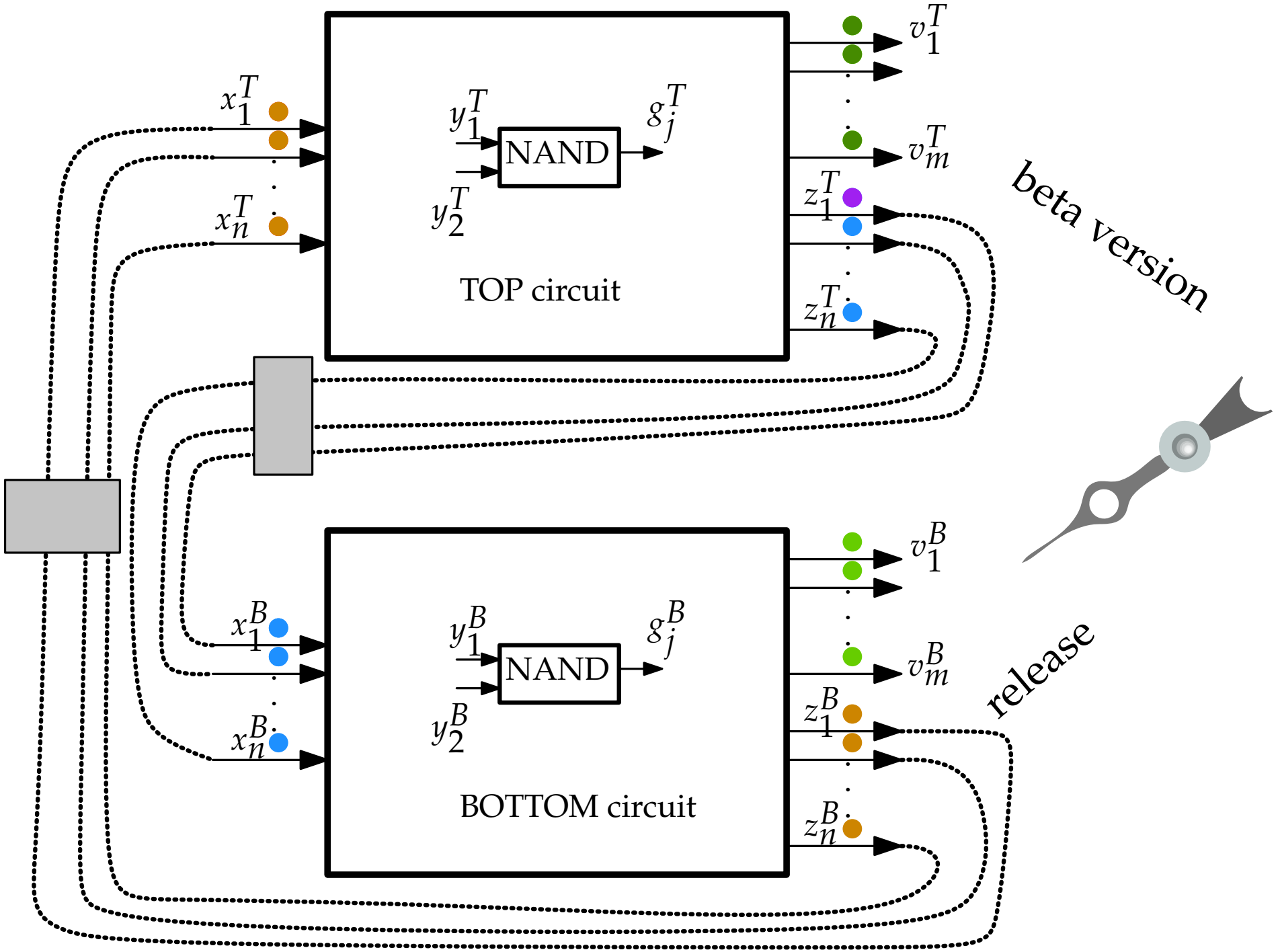

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
beta version
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
release

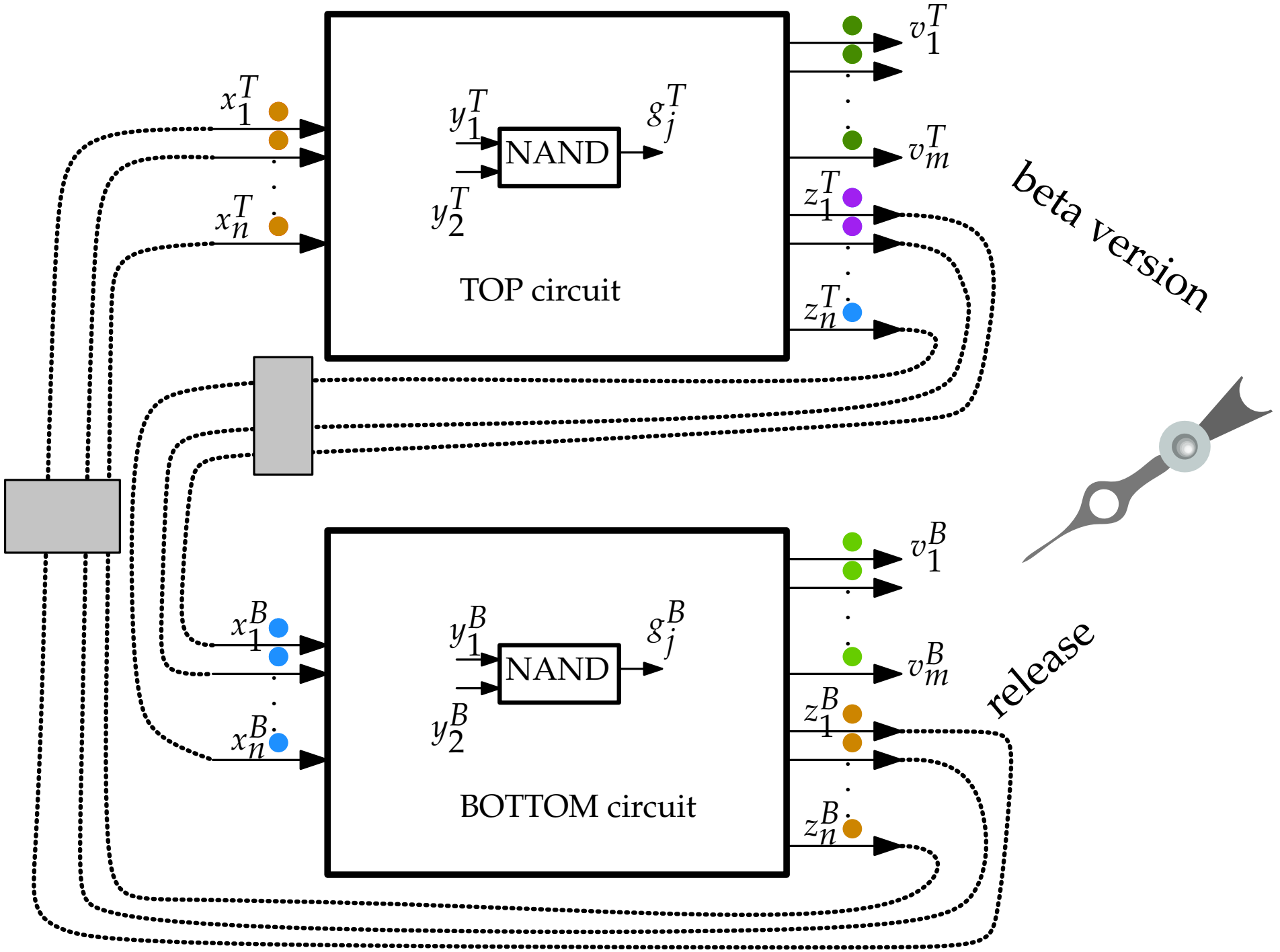

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
beta version
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
release

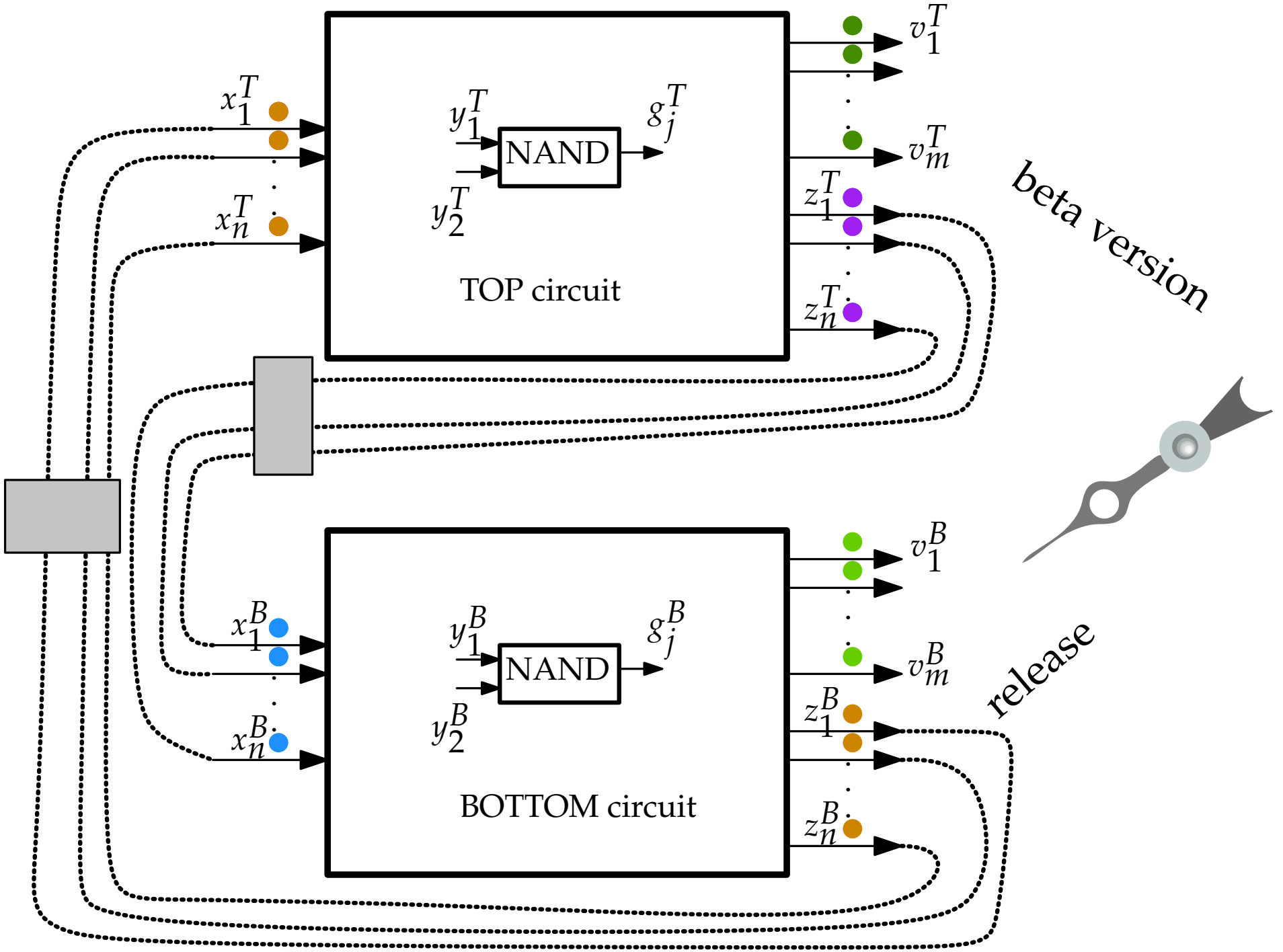

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
beta version
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
release

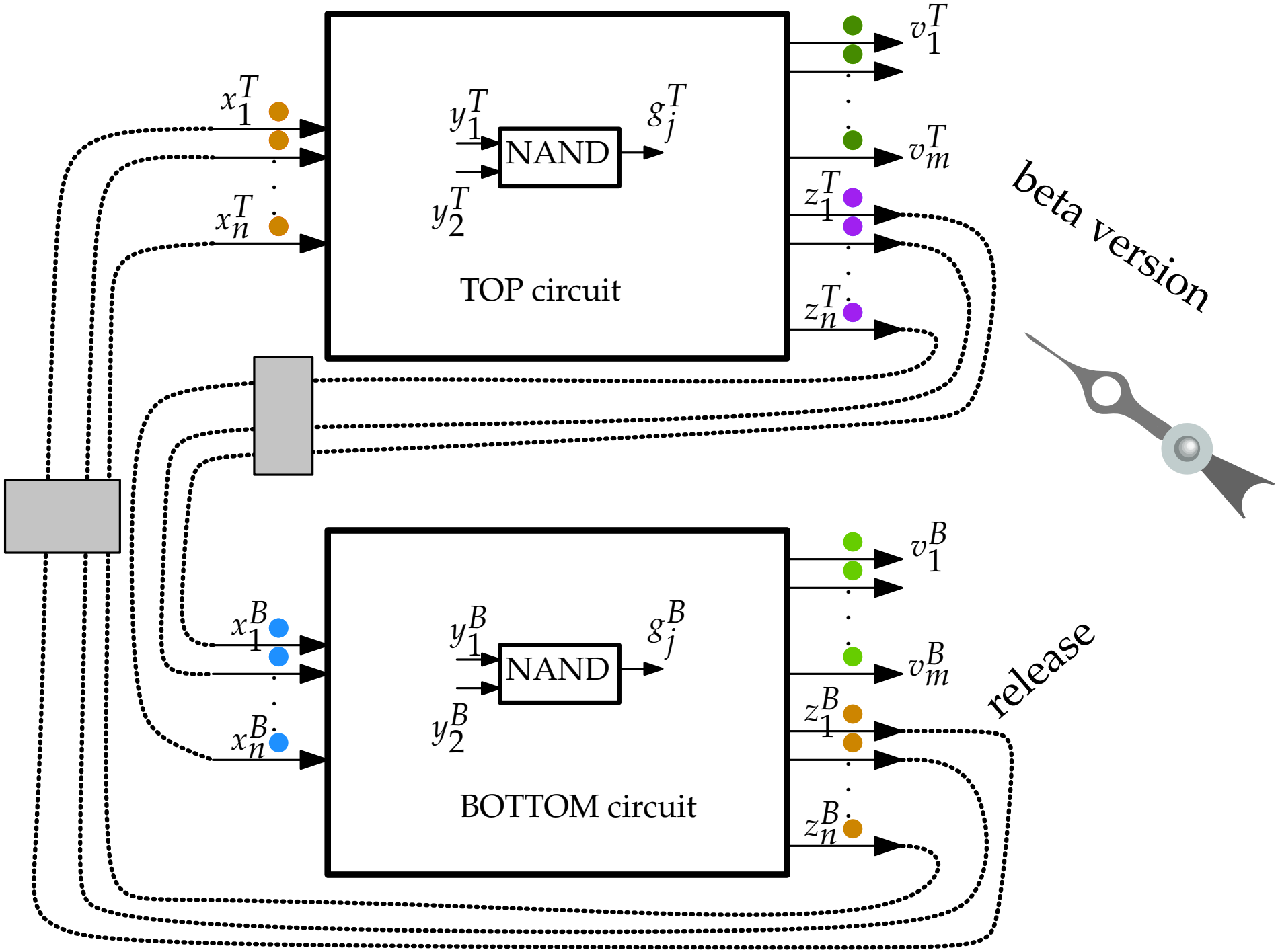

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
beta version
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
release

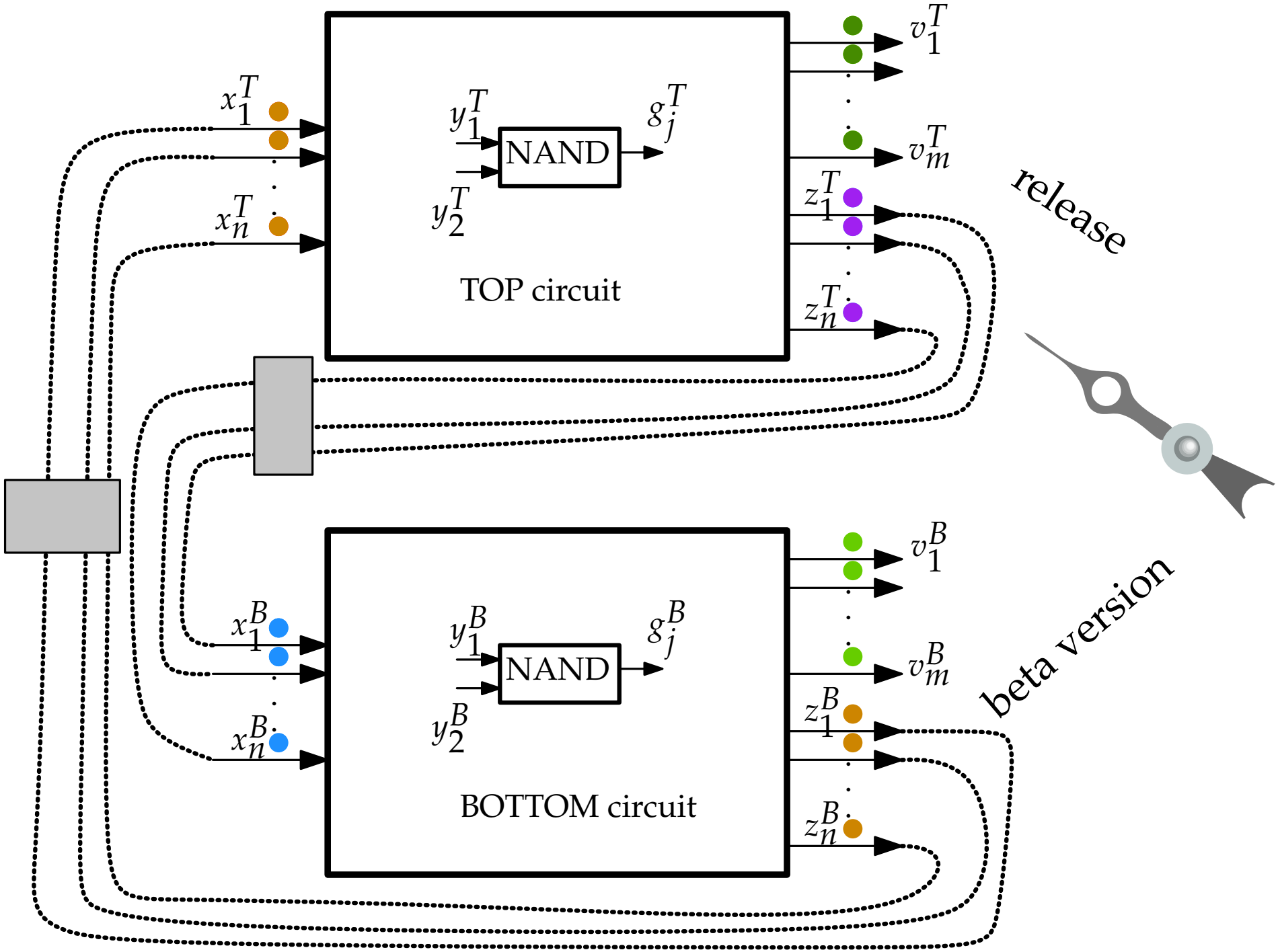

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
beta version
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
release


x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
beta version
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
release

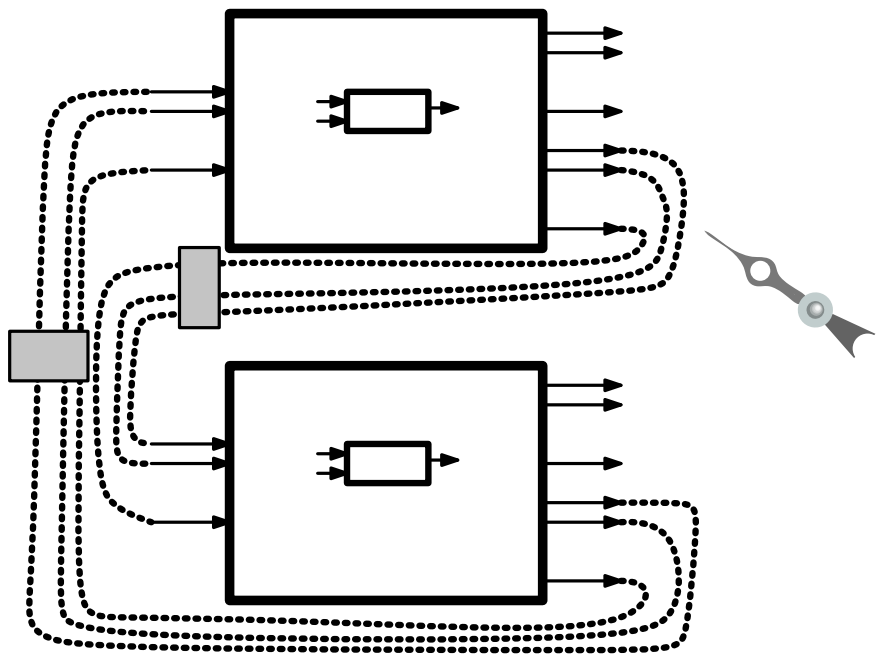

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
beta version
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
release


x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
beta version
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
release


x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
beta version
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
release

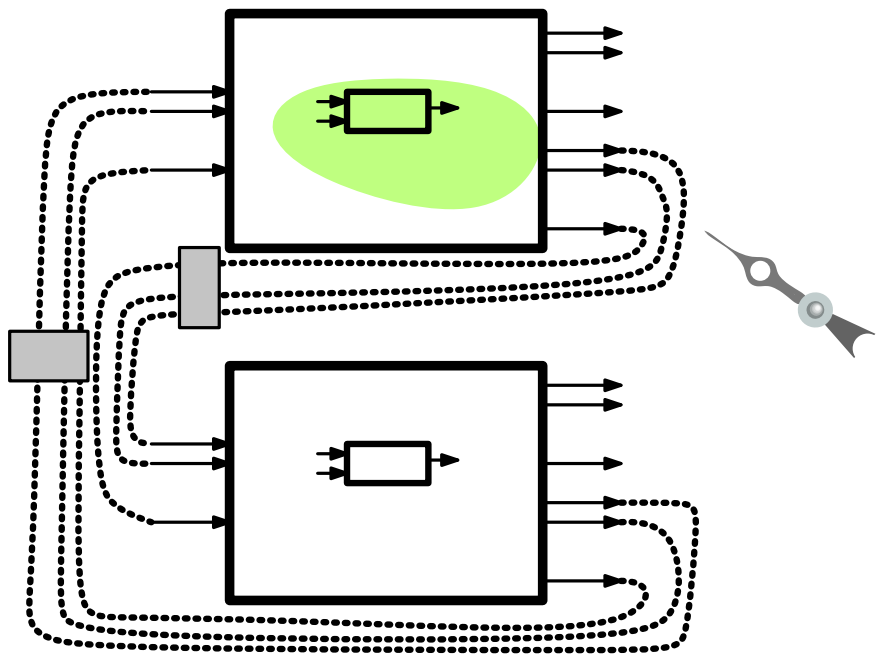

x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
release
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
beta version

Let's build a formula encoding all of this mechanism. Design four clause groups:



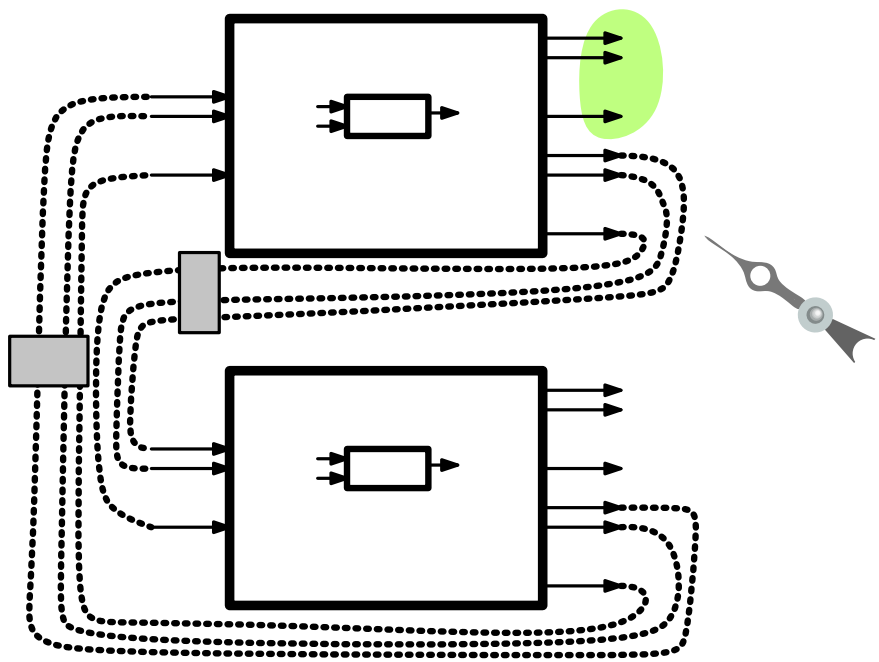
Let's build a formula encoding all of this mechanism. Design four clause groups:

- Group 1: All gates in the *release* circuit should be correct.



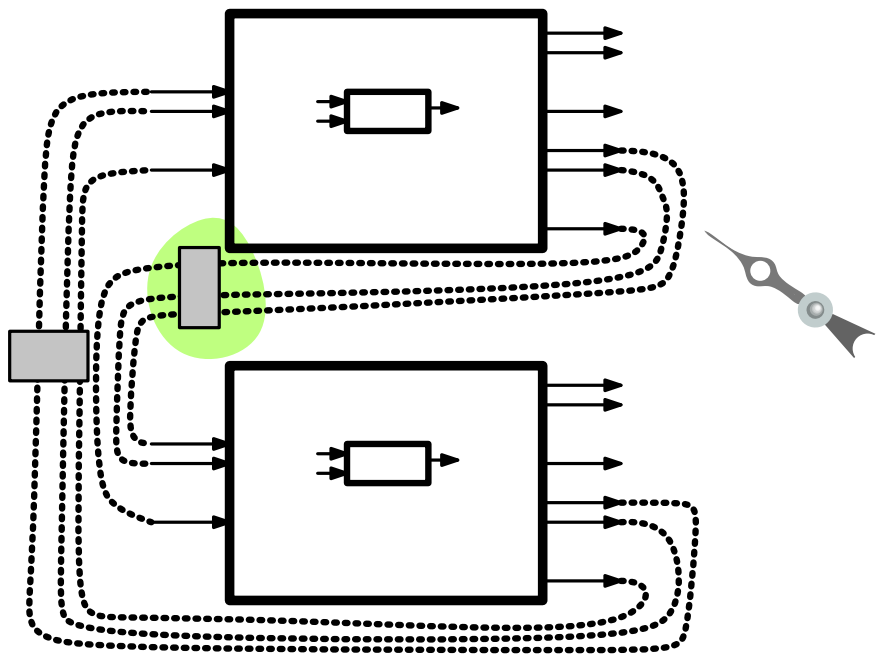
Let's build a formula encoding all of this mechanism. Design four clause groups:

- Group 1: All gates in the *release* circuit should be correct.
- Group 2: The output of the *release* circuit should be big.



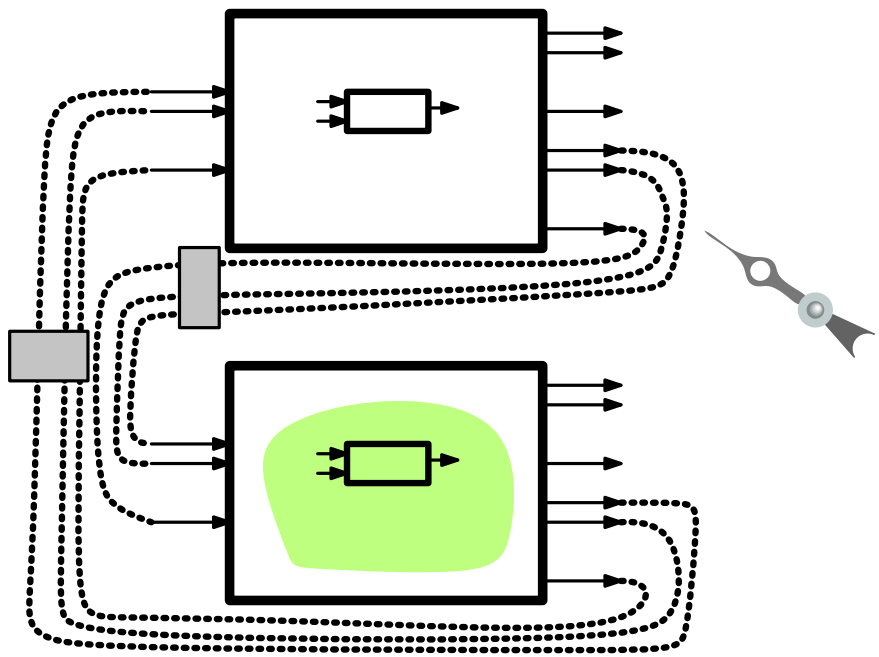
Let's build a formula encoding all of this mechanism. Design four clause groups:

- Group 1: All gates in the *release* circuit should be correct.
- Group 2: The output of the *release* circuit should be big.
- Group 3: The input of the *beta version* circuit is the output of the *release circuit*



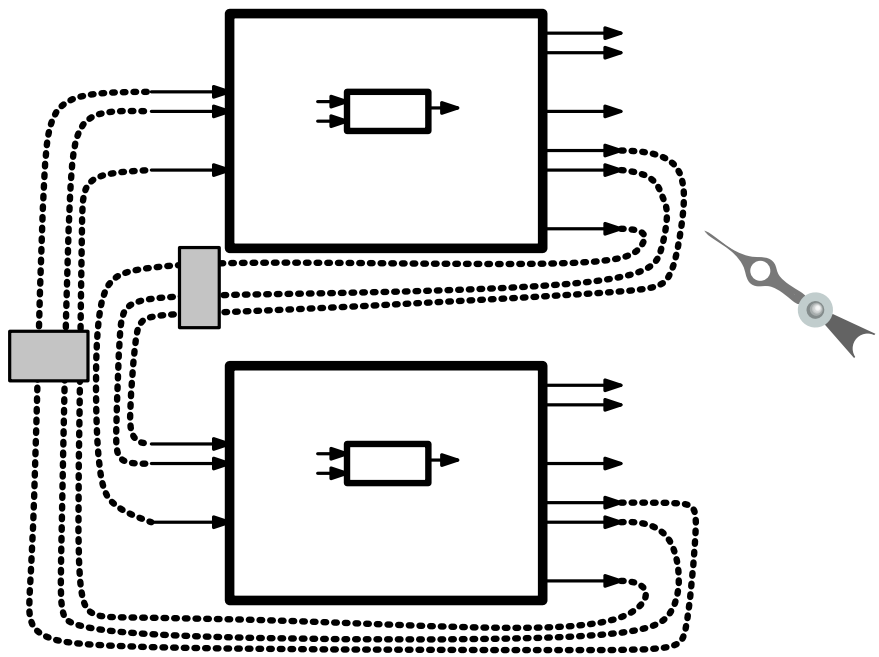
Let's build a formula encoding all of this mechanism. Design four clause groups:

- Group 1: All gates in the *release* circuit should be correct.
- Group 2: The output of the *release* circuit should be big.
- Group 3: The input of the *beta version* circuit is the output of the *release circuit*
- Group 4: The *beta version* circuit should be correct.



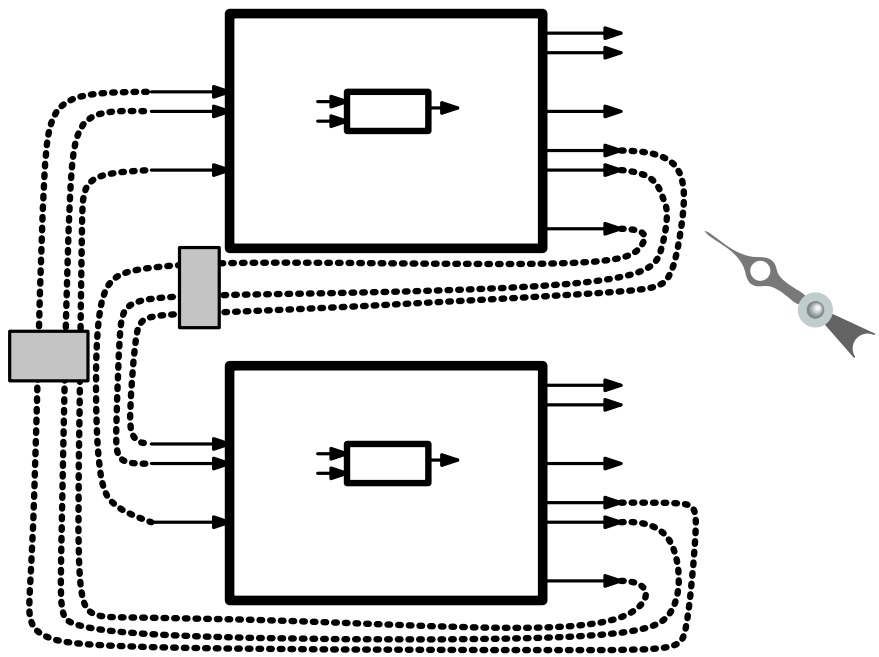
Let's build a formula encoding all of this mechanism. Design four clause groups:

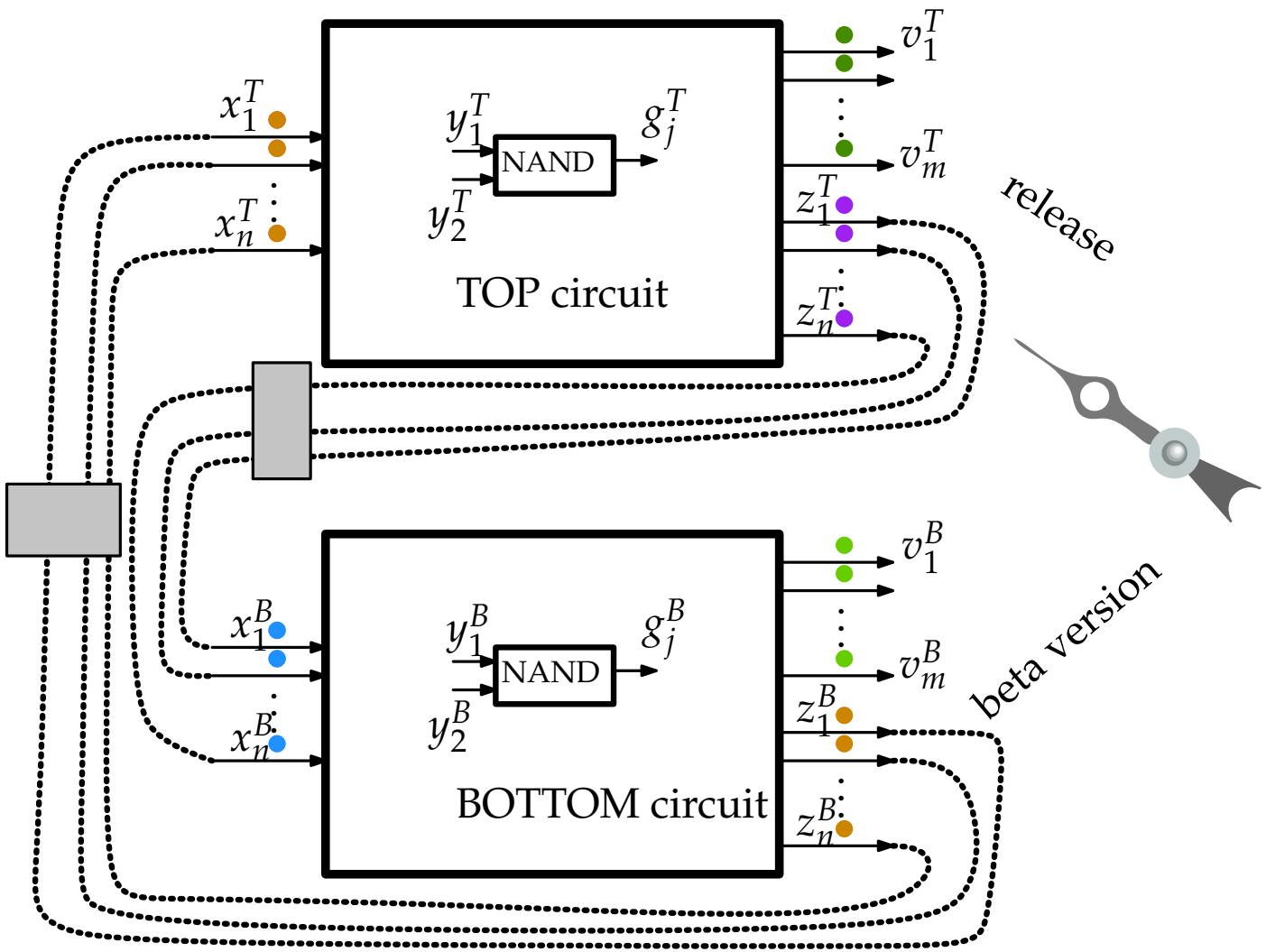
- Group 1: All gates in the *release* circuit should be correct.
- Group 2: The output of the *release* circuit should be big.
- Group 3: The input of the *beta version* circuit is the output of the *release circuit*
- Group 4: The *beta version* circuit should be correct.



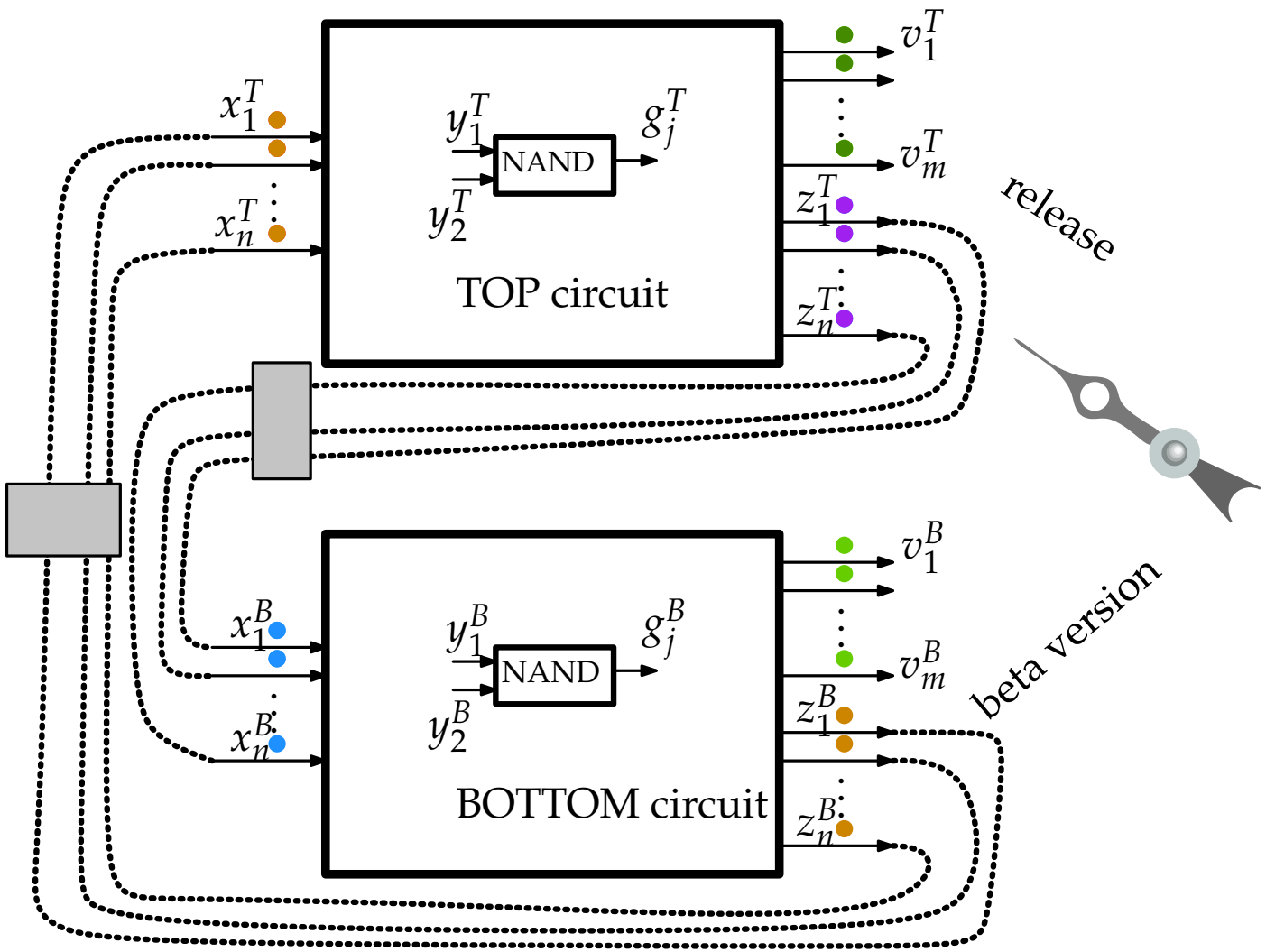
Let's build a formula encoding all of this mechanism. Design four clause groups:

- Group 1: All gates in the *release* circuit should be correct.
- Group 2: The output of the *release* circuit should be big.
- Group 3: The input of the *beta version* circuit is the output of the *release circuit*
- Group 4: The *beta version* circuit should be correct.

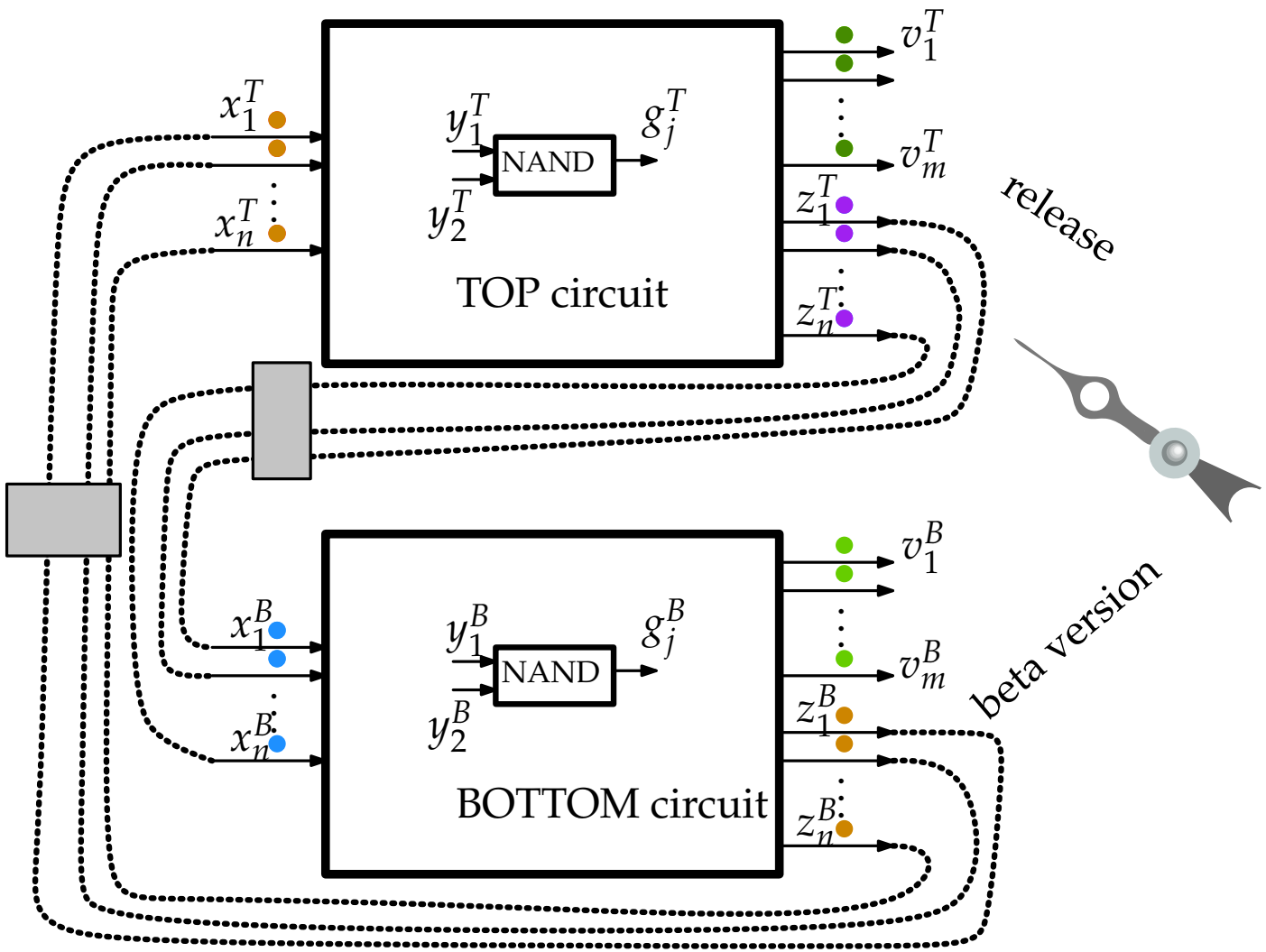



x_1^T
x_n^T
y_1^T
y_2^T
NAND
g_j^T
TOP circuit
v_1^T
v_m^T
z_1^T
z_n^T
release
x_1^B
x_n^B
y_1^B
y_2^B
NAND
g_j^B
BOTTOM circuit
v_1^B
v_m^B
z_1^B
z_n^B
beta version

It would be good if v_i^T were 1 if TOP is the release.

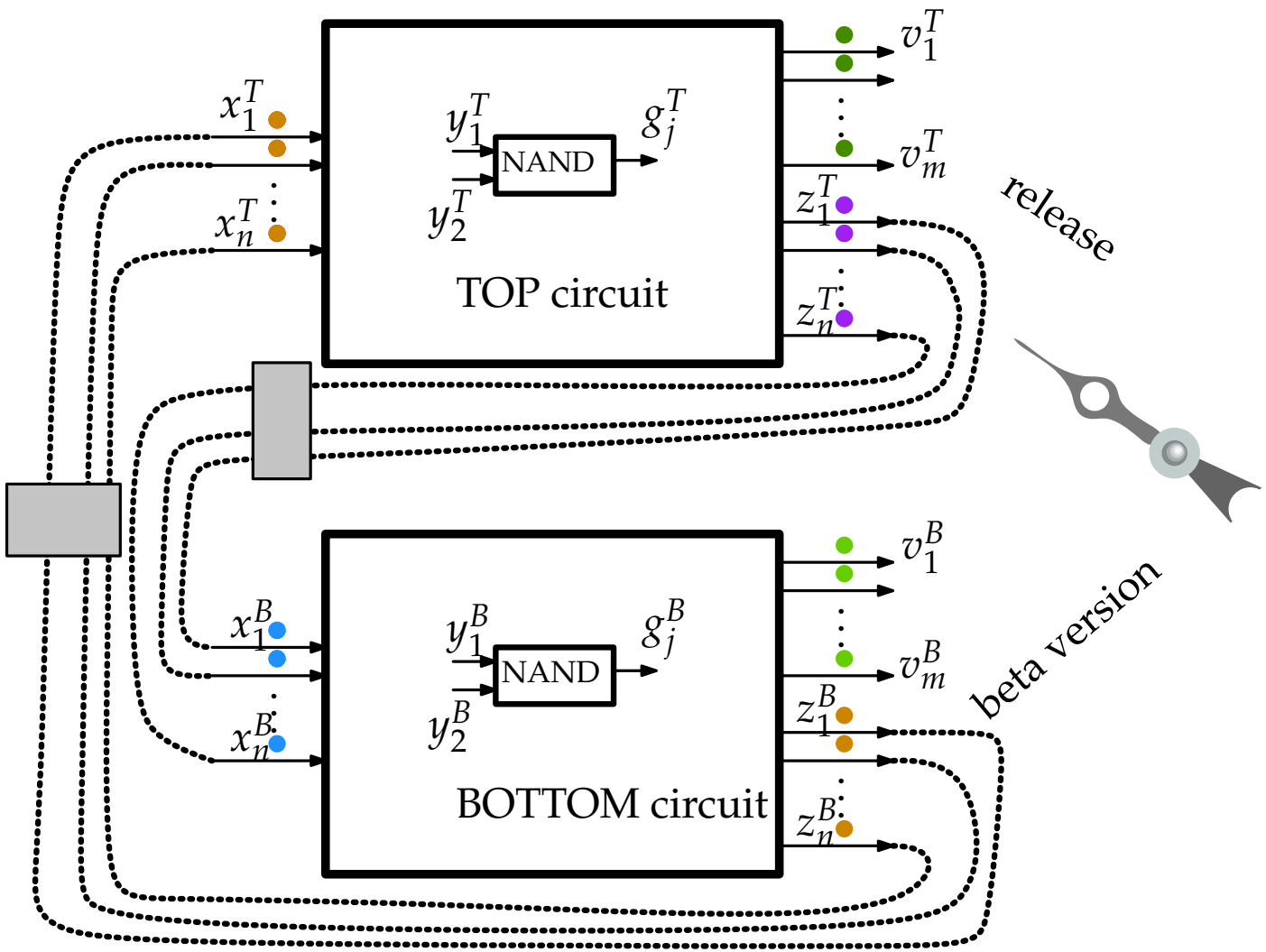


It would be good if v_i^T were 1 if TOP is the release.
If TOP is beta version it wouldn't matter.



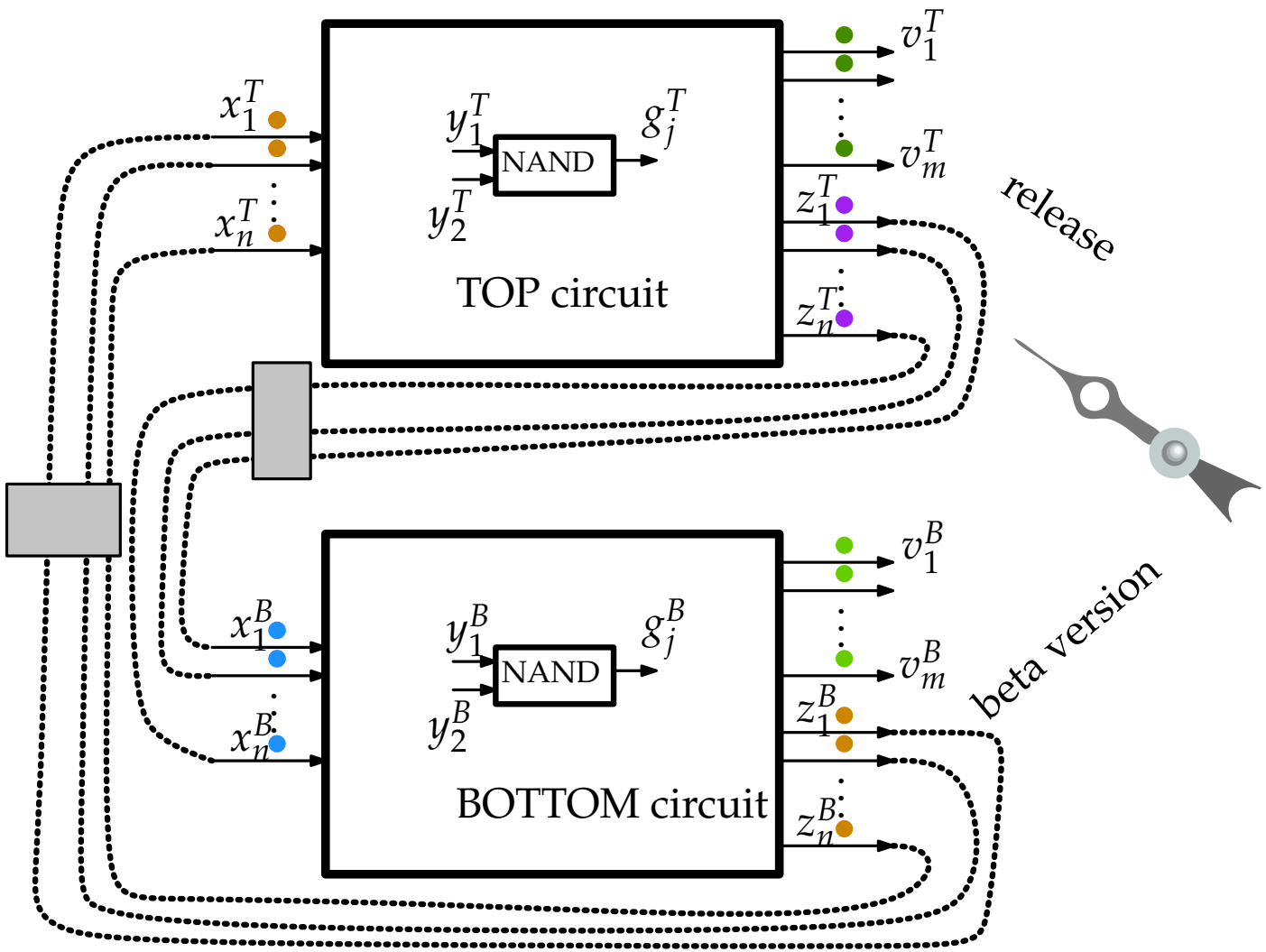
It would be good if v_i^T were 1 if TOP is the release.
If TOP is beta version it wouldn't matter.

It would be good if v_i^B were 1 if BOTTOM is the release.



It would be good if v_i^T were 1 if TOP is the release.
If TOP is beta version it wouldn't matter.

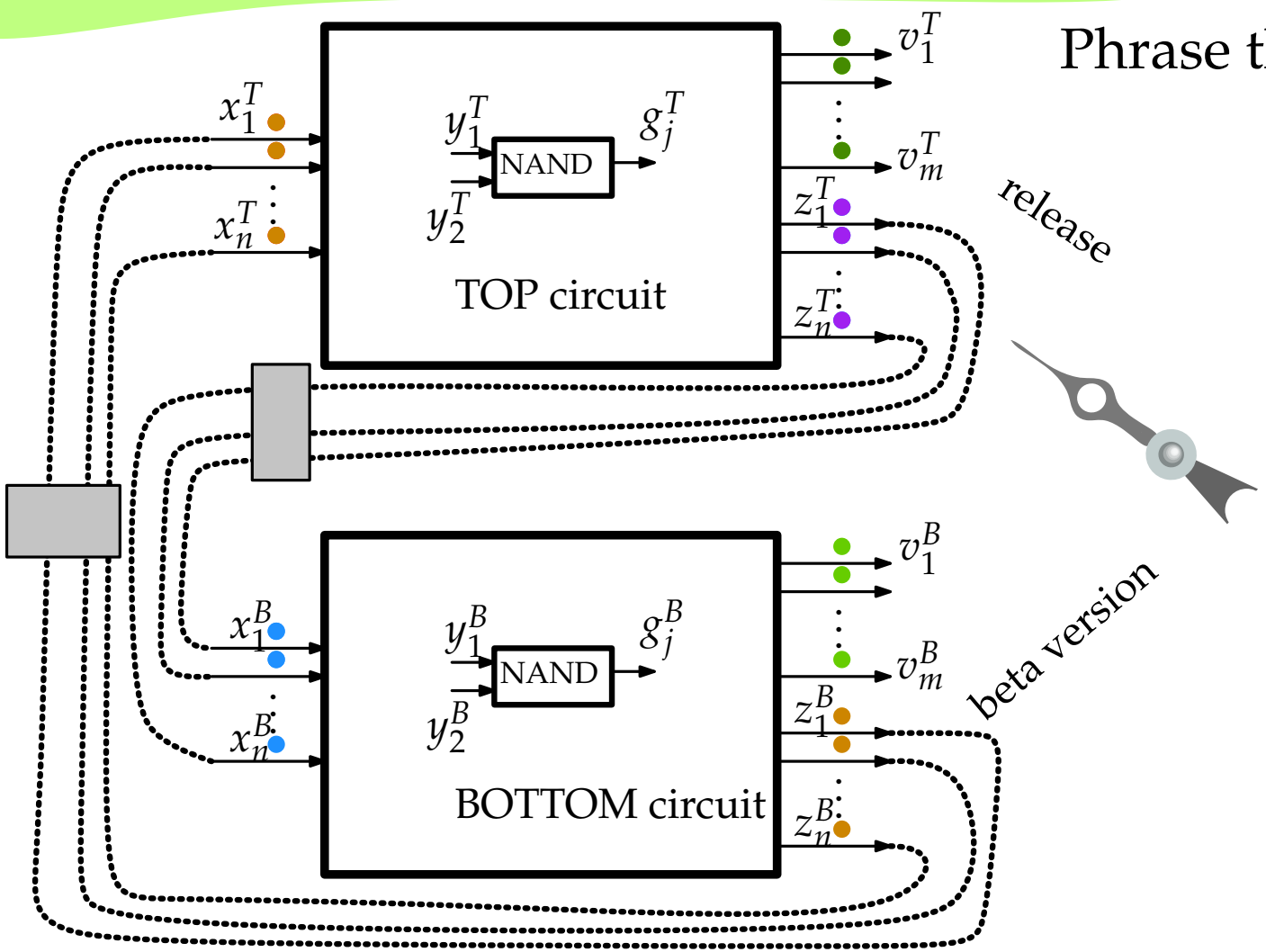
It would be good if v_i^B were 1 if BOTTOM is the release.
If BOTTOM is beta version it wouldn't matter.



It would be good if v_i^T were 1 if TOP is the release.
If TOP is beta version it wouldn't matter.

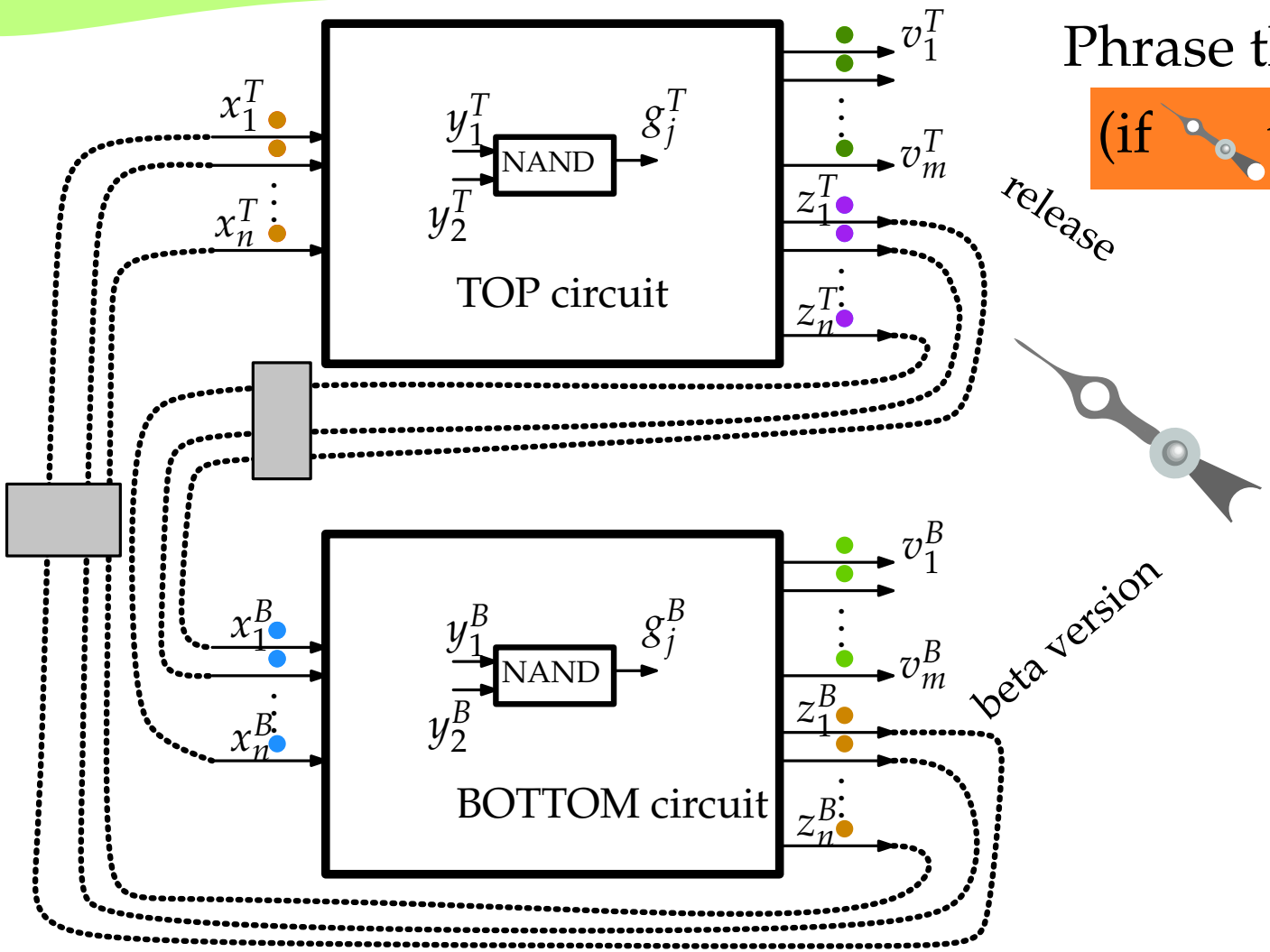
It would be good if v_i^B were 1 if BOTTOM is the release.
If BOTTOM is beta version it wouldn't matter.

Phrase this as a 3-clasue.



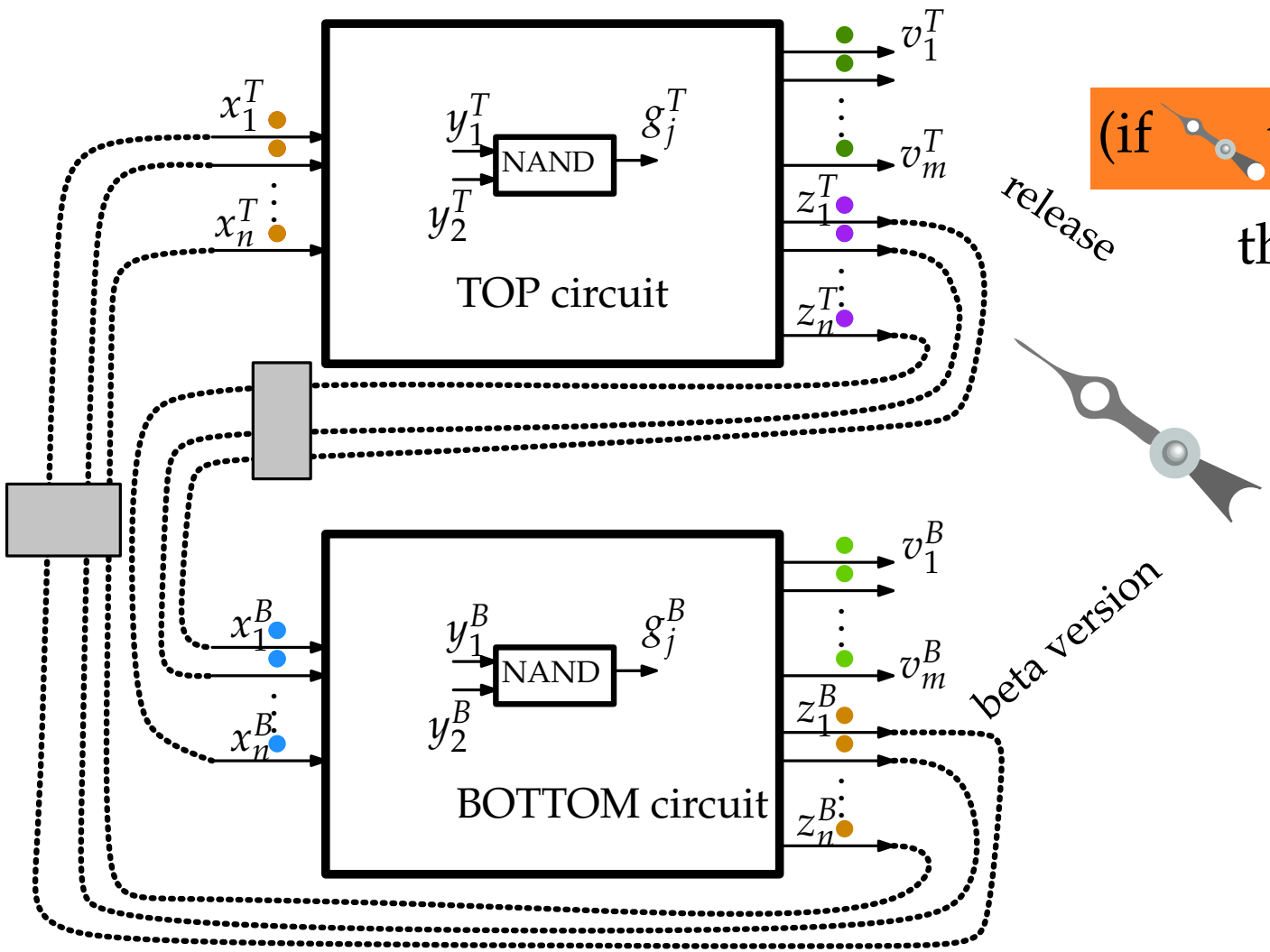
It would be good if v_i^T were 1 if TOP is the release.
If TOP is beta version it wouldn't matter.

It would be good if v_i^B were 1 if BOTTOM is the release.
If BOTTOM is beta version it wouldn't matter.



Phrase this as a 3-clasue.

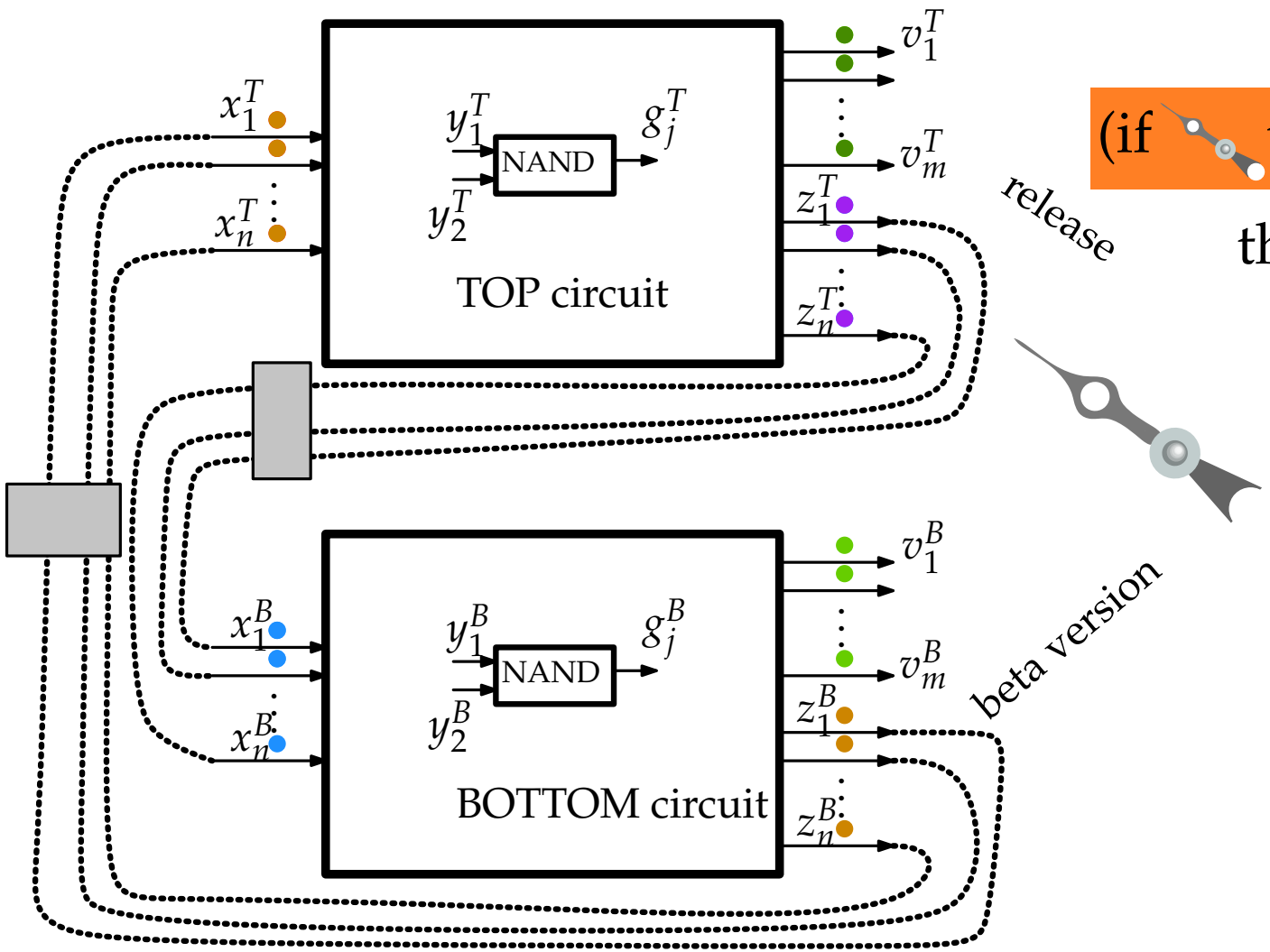
(if then x_i^T else x_i^B)



(if then x_i^T else x_i^B)

that's not a clause

$(\ \rightarrow x_i^T) \wedge (\ \rightarrow x_i^B)$

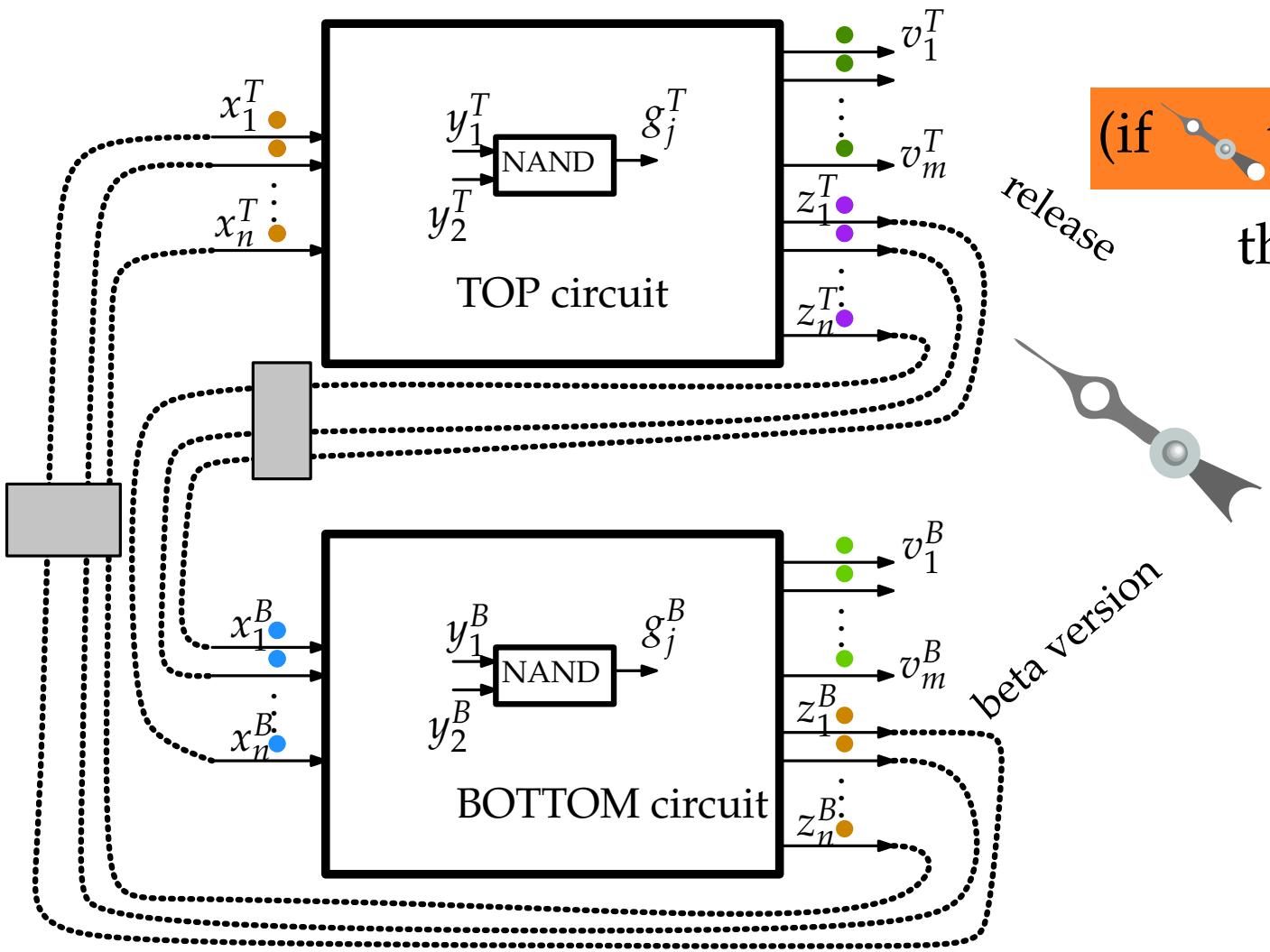


(if then x_i^T else x_i^B)

that's not a clause

$(\ \rightarrow x_i^T) \wedge (\ \rightarrow x_i^B)$

It's two 2-clauses!



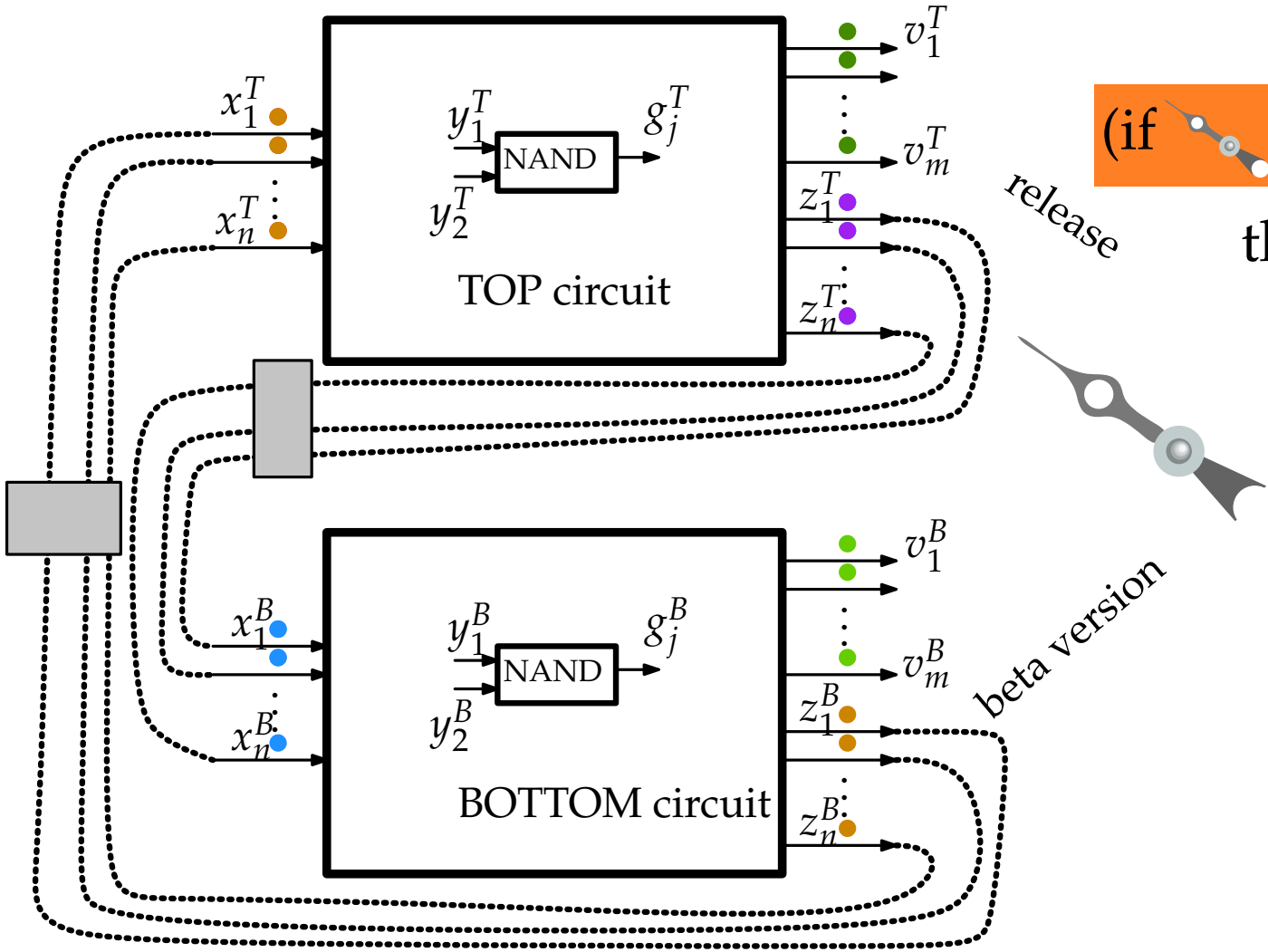
$(\text{if}\ \ \text{then}\ x_i^T\ \text{else}\ x_i^B)$

that's not a clause

$(\text{  } \rightarrow x_i^T) \wedge (\text{  } \rightarrow x_i^B)$

Both have equal weight!

It's two 2-clauses!



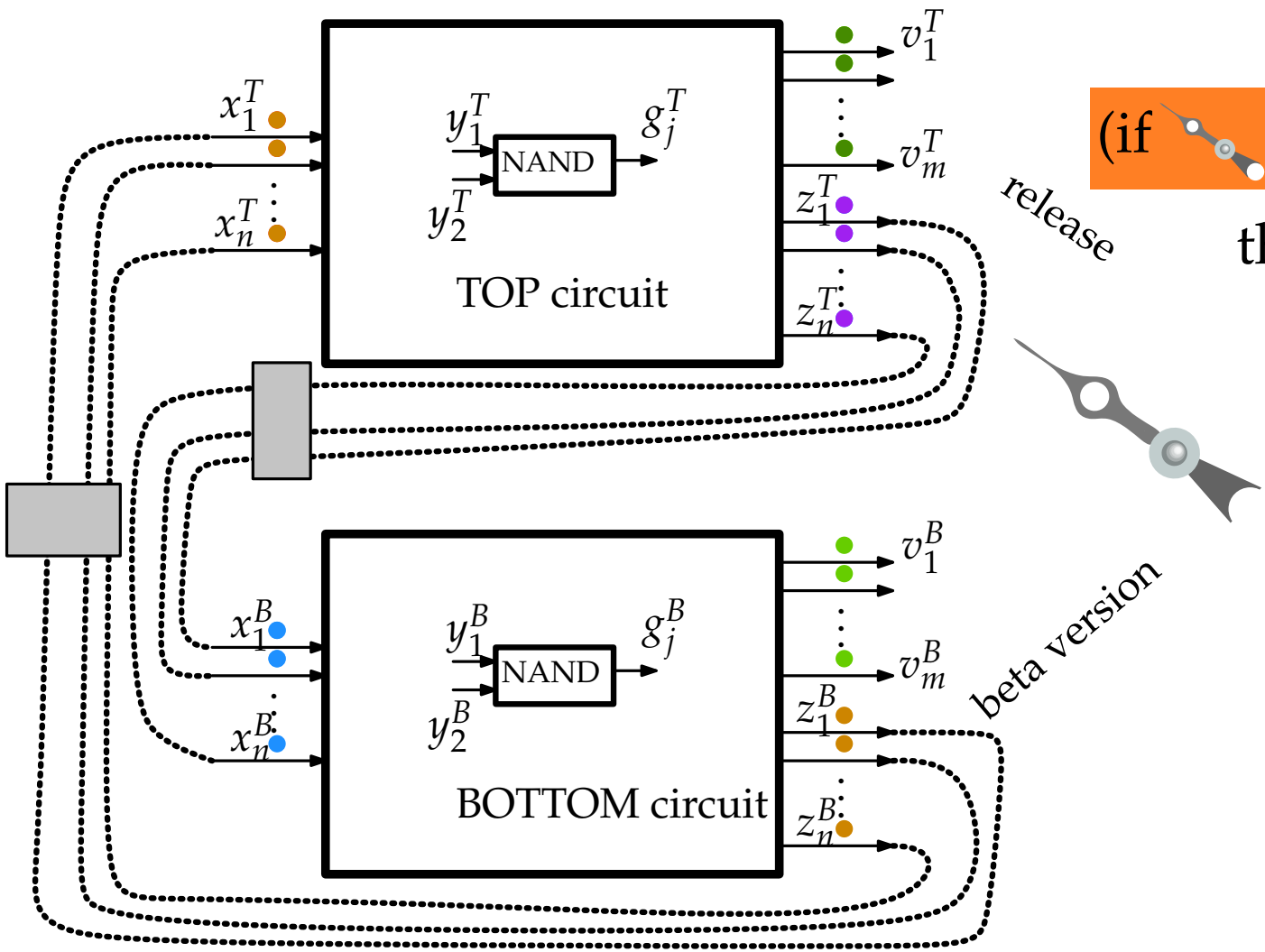
$(\text{if } \text{} \text{ then } x_i^T \text{ else } x_i^B)$

that's not a clause

$(\ \rightarrow x_i^T) \wedge (\ \rightarrow x_i^B)$

It's two 2-clauses!

Both have equal weight!
But we want: each clause weighs more than all downstream clauses combined!



$(\text{if}\ \ \text{then}\ x_i^T\ \text{else}\ x_i^B)$

that's not a clause

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Theorem (S., Tantow). Local Max-k-SAT with lexicographic weights is PLS-complete.

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Theorem (S., Tantow). Local Max-k-SAT with lexicographic weights is PLS-complete.

Clause C weighs more than all downstream clauses together.

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Theorem (S., Tantow). Local Max-k-SAT with lexicographic weights is PLS-complete.

Clause C weighs more than all downstream clauses together.

$$F = C_1 \wedge C_2 \wedge C_3 \wedge \cdots \wedge C_{m-1} \wedge C_m$$

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Theorem (S., Tantow). Local Max-k-SAT with lexicographic weights is PLS-complete.

Clause C weighs more than all downstream clauses together.

$$F = C_1 \wedge C_2 \wedge C_3 \wedge \cdots \wedge C_{m-1} \wedge C_m$$

$$2^{m-1} \quad 2^{m-2} \quad 2^{m-3} \qquad 2 \qquad 1$$

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Theorem (S., Tantow). Local Max-k-SAT with lexicographic weights is PLS-complete.

Clause C weighs more than all downstream clauses together.

$$F = C_1 \wedge C_2 \wedge C_3 \wedge \cdots \wedge C_{m-1} \wedge C_m$$

$$2^{m-1} \quad 2^{m-2} \quad 2^{m-3} \qquad 2 \qquad 1$$

assignment $\vec{x} \in \{0,1\}^n$

Theorem (Johnson, Papadimitriou, Yannakakis). Local Max-k-SAT is PLS-complete.

Theorem (S., Tantow). Local Max-k-SAT with lexicographic weights is PLS-complete.

Clause C weighs more than all downstream clauses together.

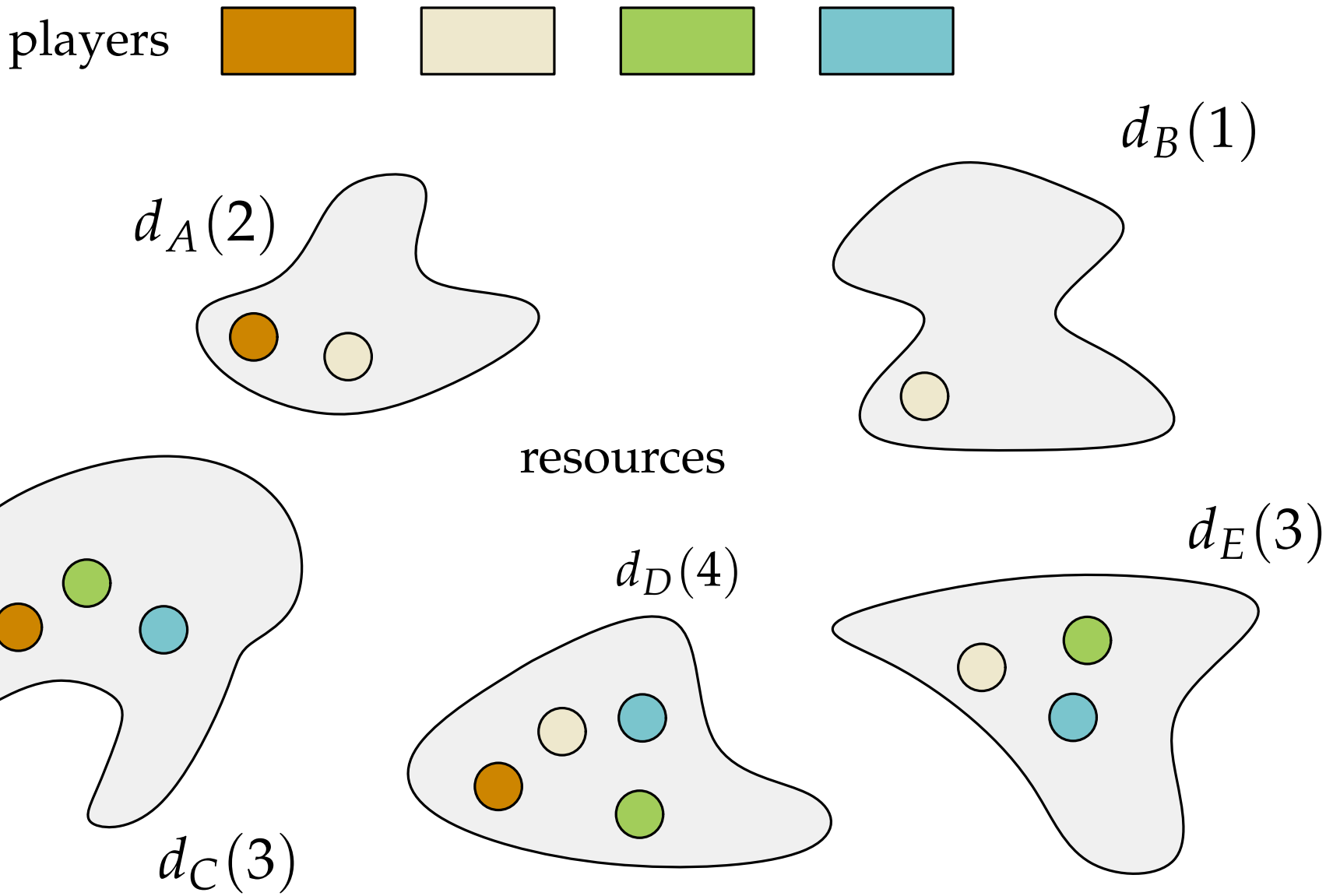
$$F = \overset{0}{C_1} \wedge \overset{1}{C_2} \wedge \overset{1}{C_3} \wedge \cdots \wedge \overset{0}{C_{m-1}} \wedge \overset{0}{C_m}$$

$$2^{m-1} \quad 2^{m-2} \quad 2^{m-3} \qquad 2 \qquad 1$$

assignment $\vec{x} \in \{0,1\}^n$

From k-SAT to Congestion Games

Theorem (Rosenthal). Every congestion game has a pure Nash equilibrium.



From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

$$(x \vee y) \quad (x \vee z) \quad (\bar{x} \vee \bar{y}) \quad (\bar{x} \vee \bar{z}) \quad (y \vee z) \quad (x \vee \bar{y} \vee \bar{z})$$

From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

Players: x y z

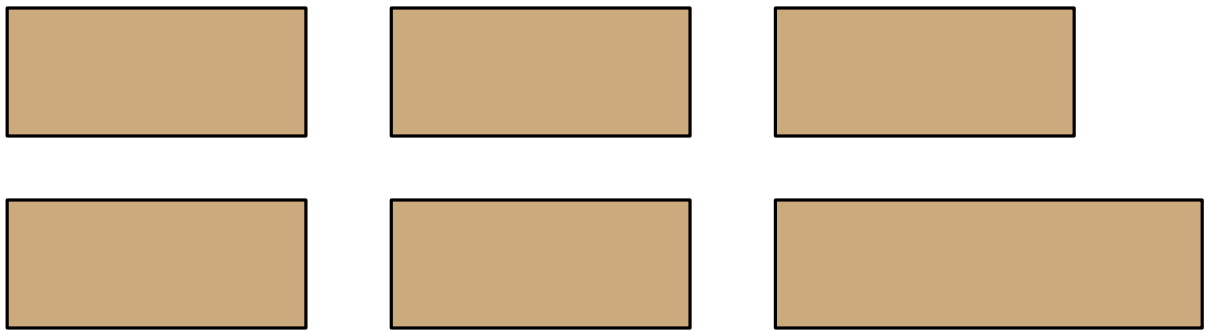
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

Players: x y z

Resources:



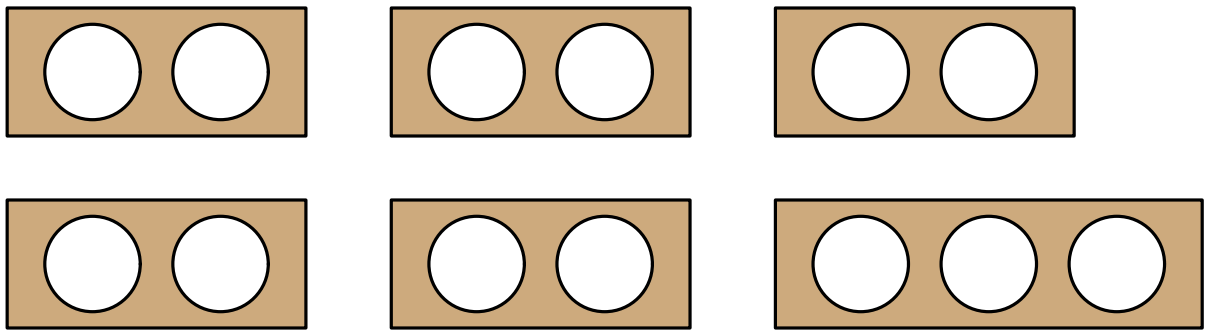
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

Players: x y z

Resources:



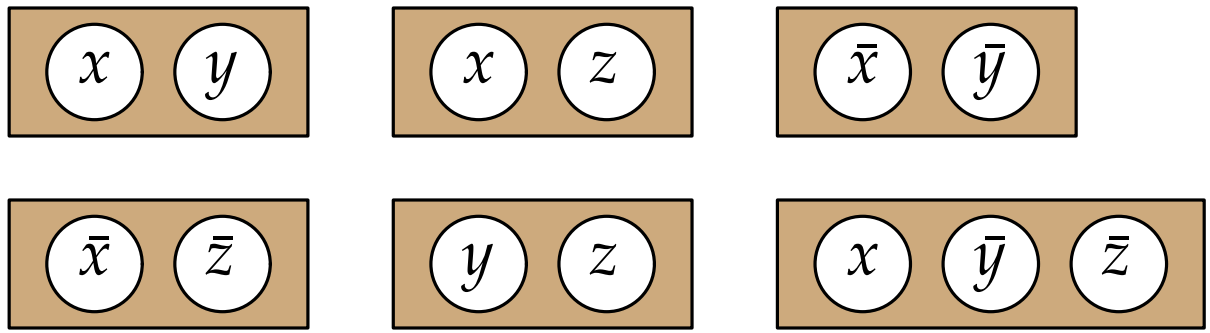
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

Players: x y z

Resources:



From k-SAT to Congestion Games

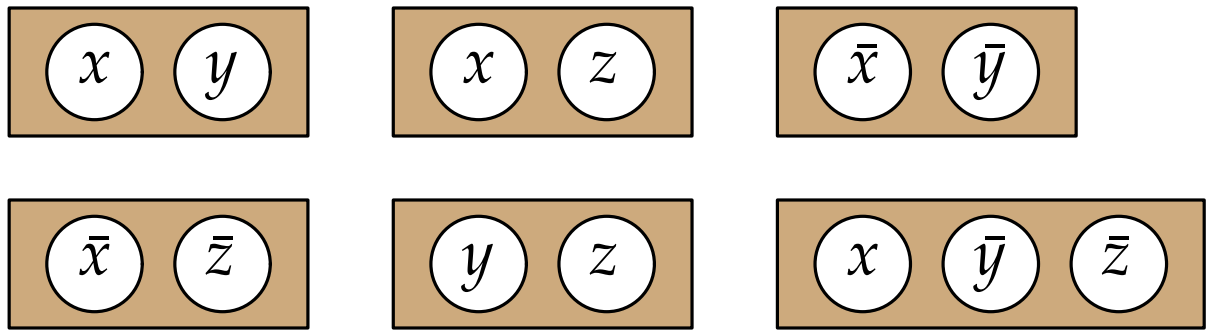
Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$

Players: x y z

Strategies:

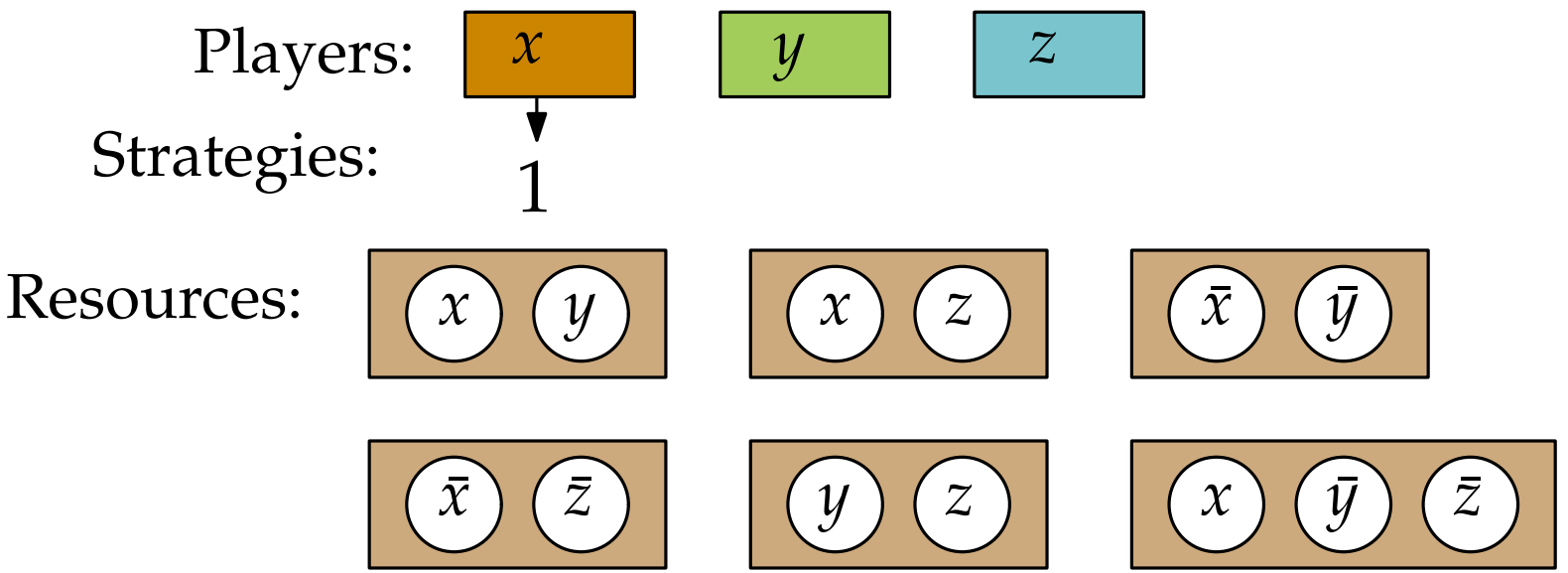
Resources:



From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

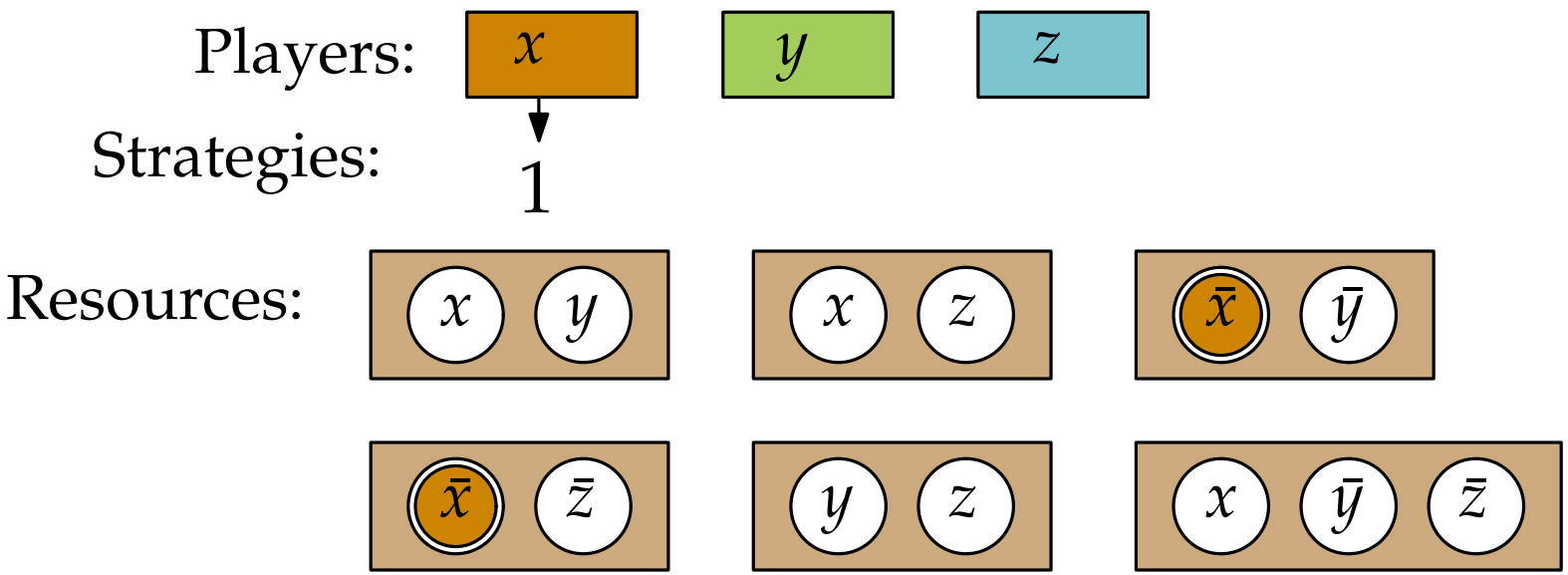
32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$



From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

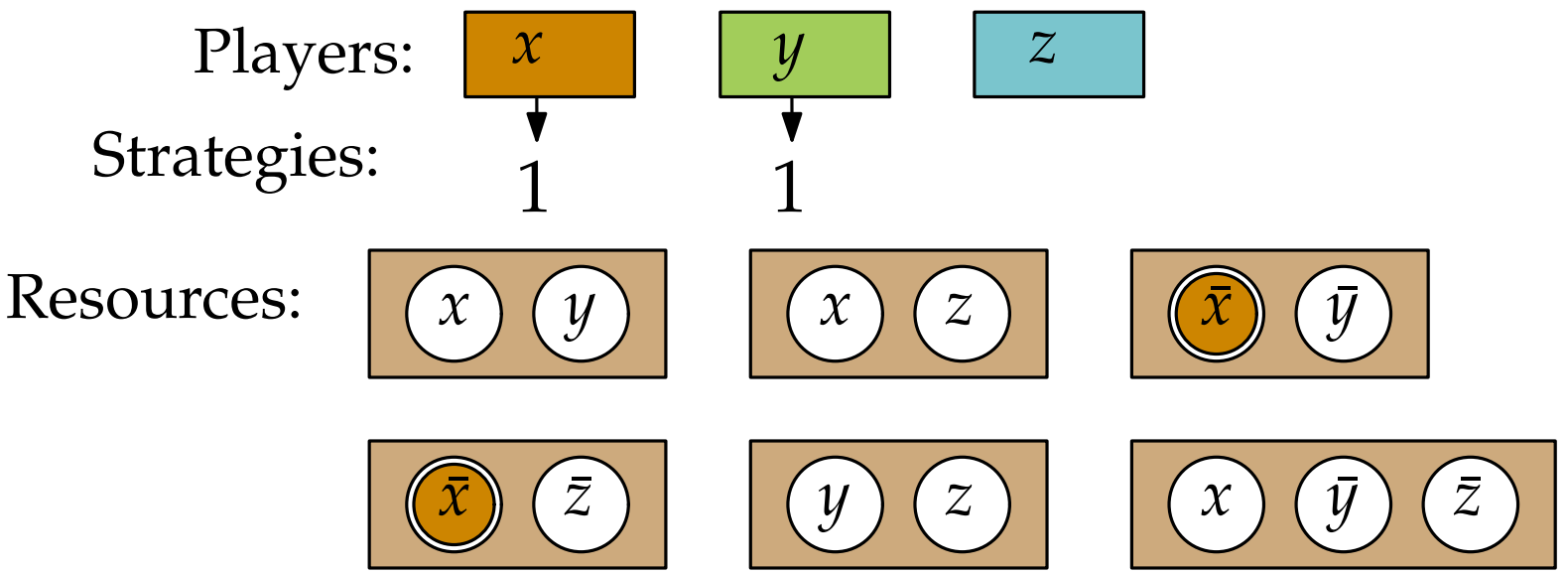
32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$



From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

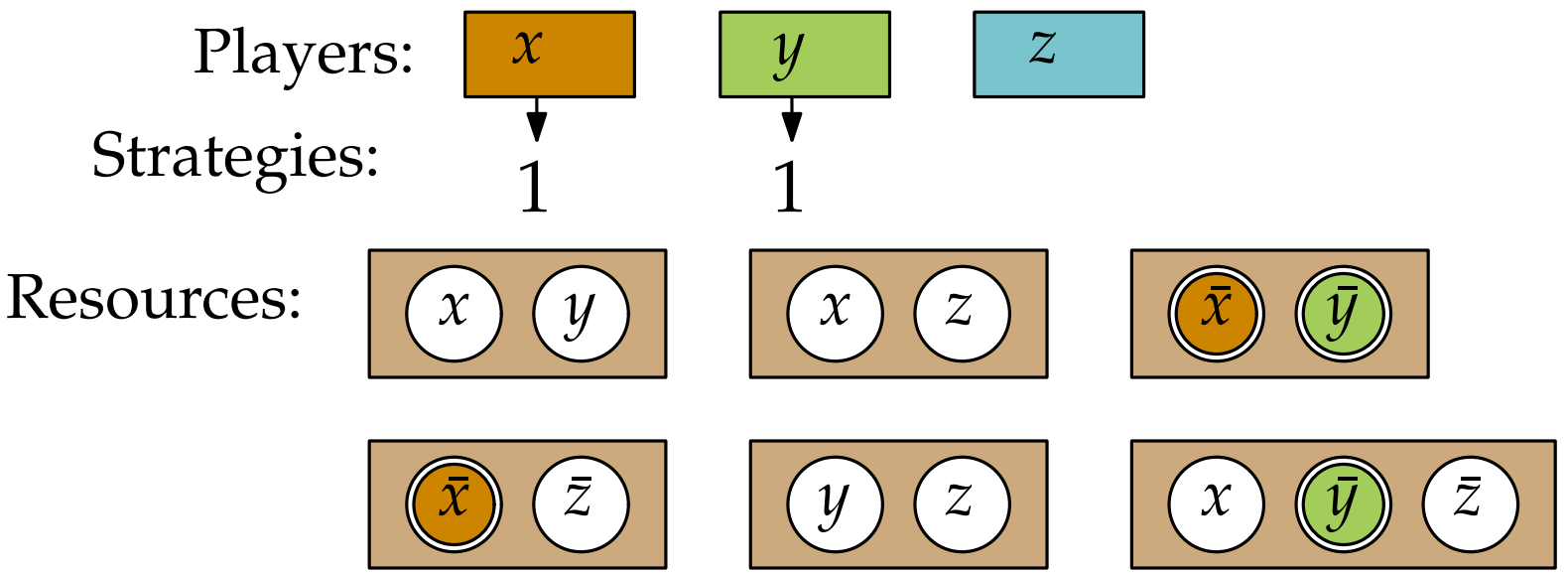
32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$



From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

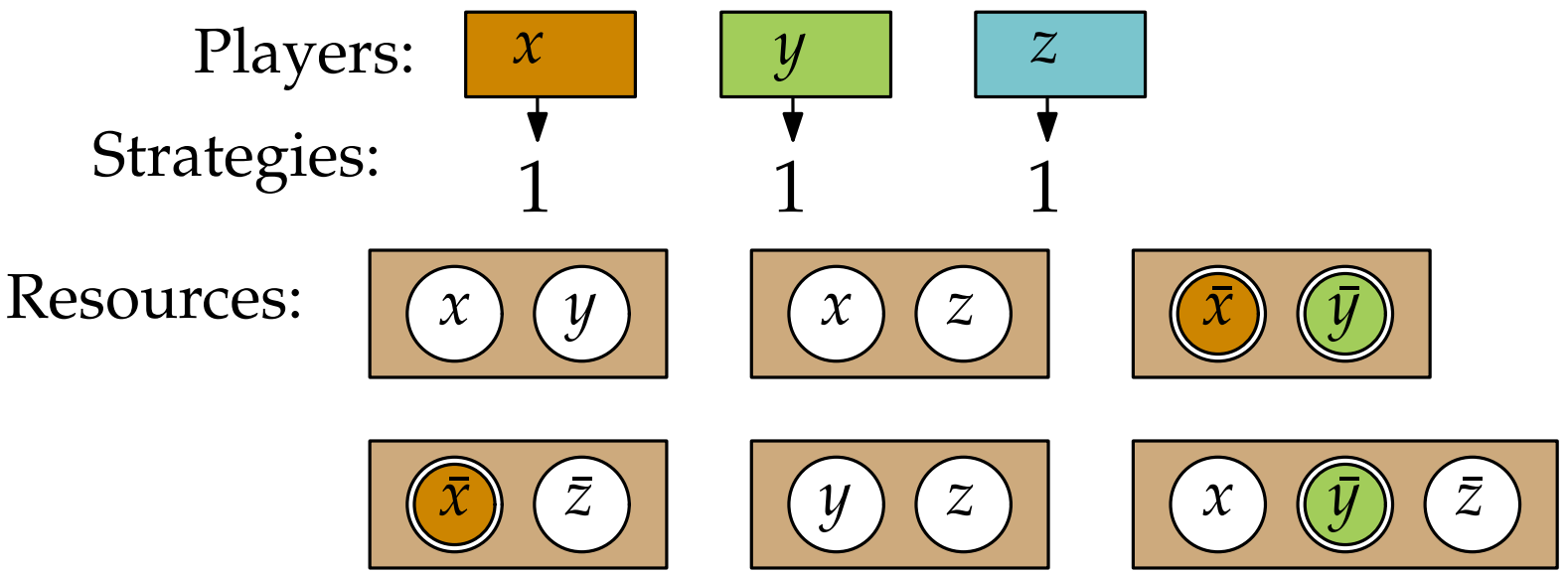
32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$



From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

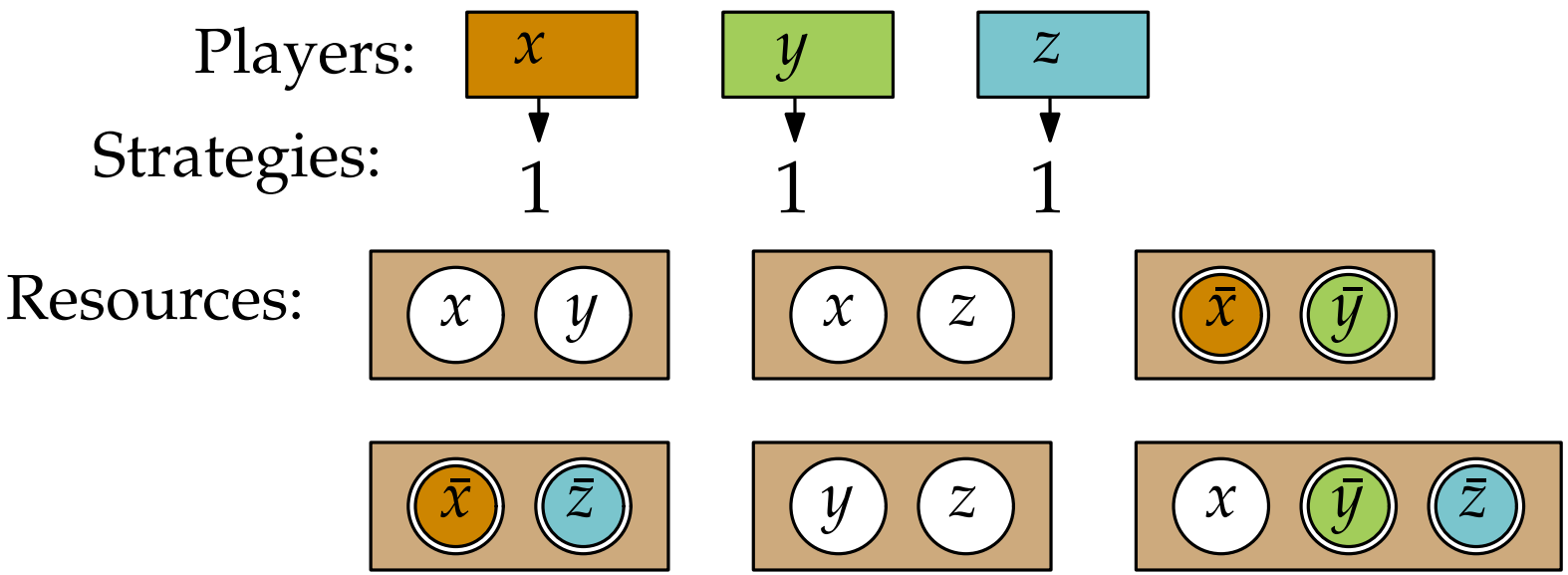
32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$



From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

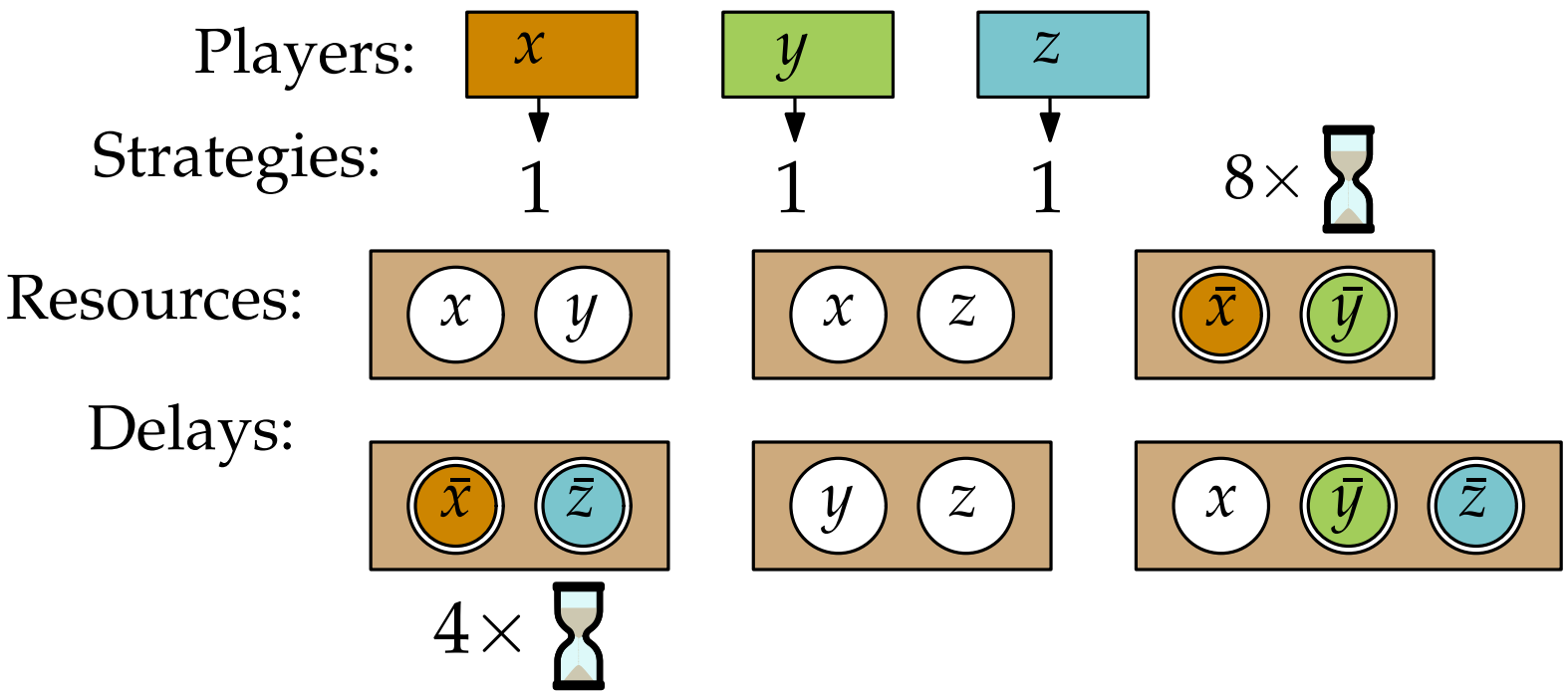
32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$



From k-SAT to Congestion Games

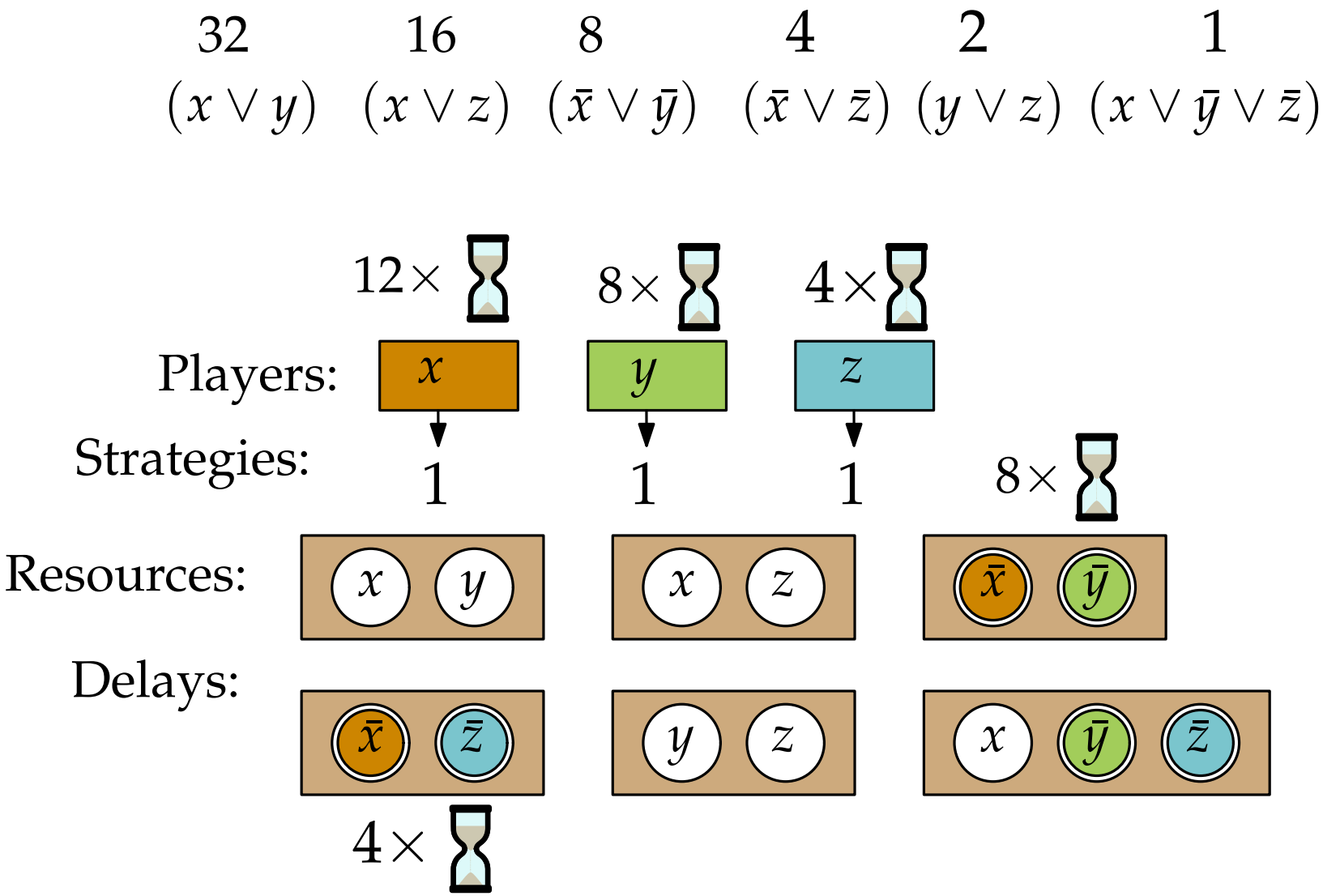
Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.

32	16	8	4	2	1
$(x \vee y)$	$(x \vee z)$	$(\bar{x} \vee \bar{y})$	$(\bar{x} \vee \bar{z})$	$(y \vee z)$	$(x \vee \bar{y} \vee \bar{z})$



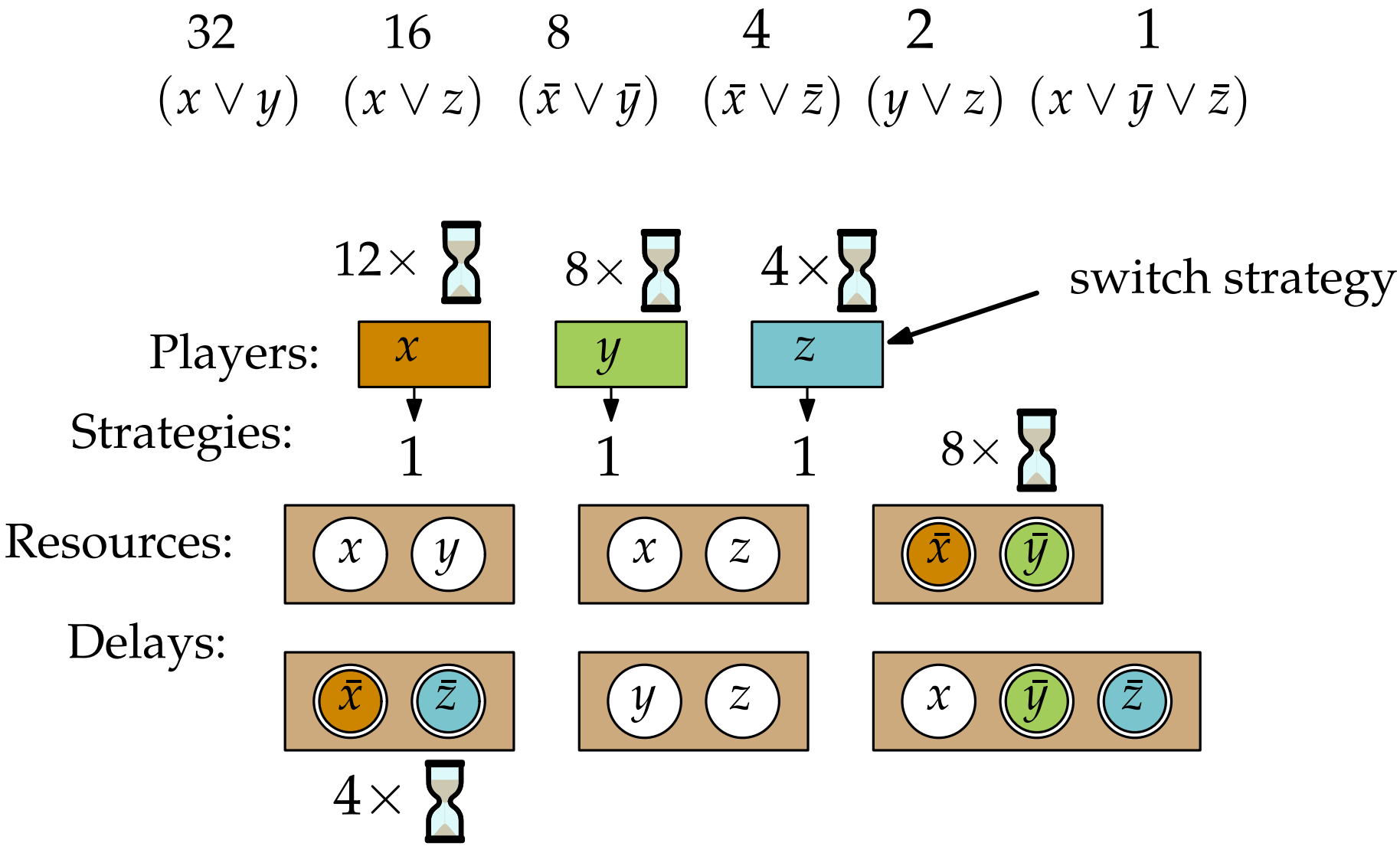
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



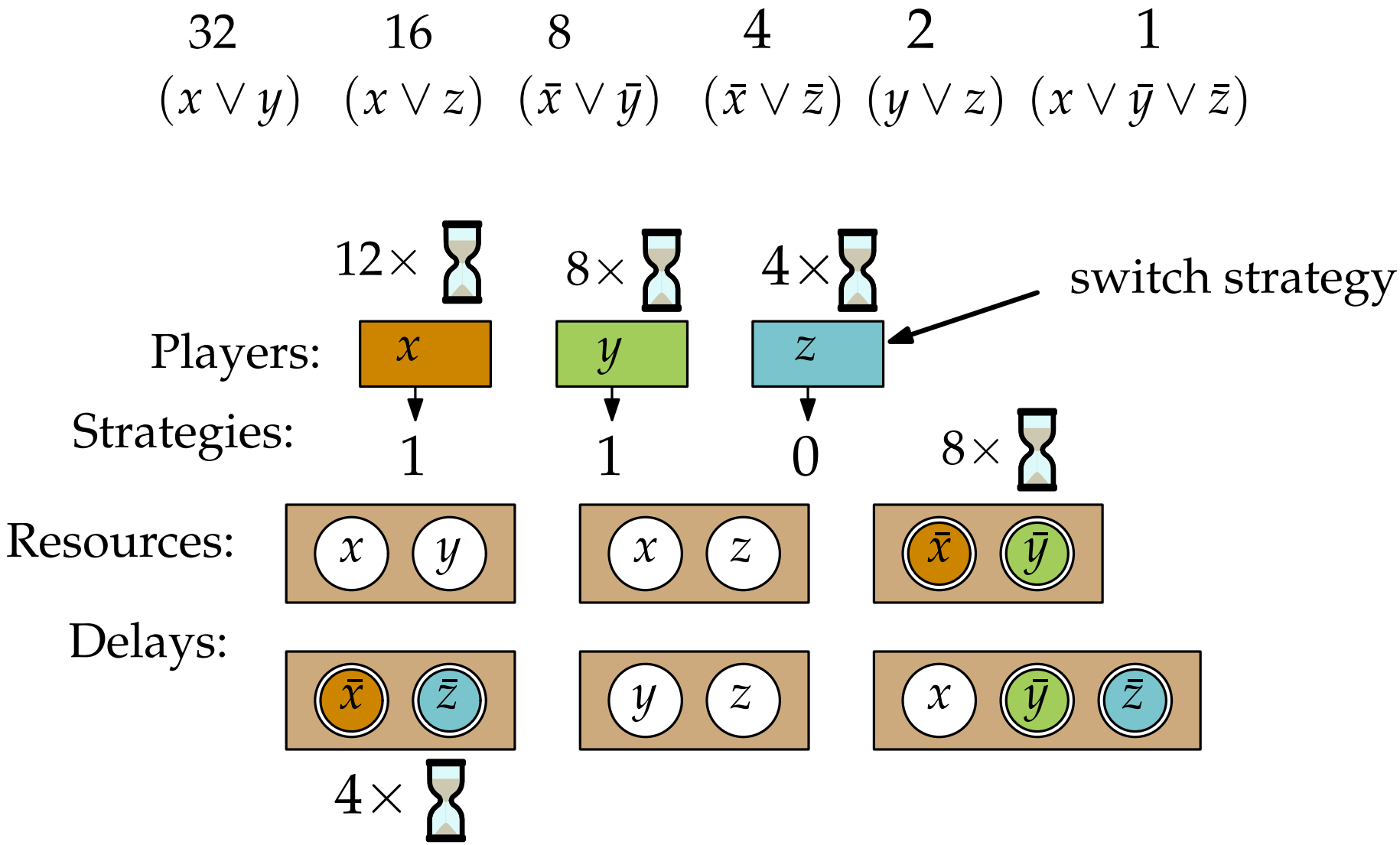
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



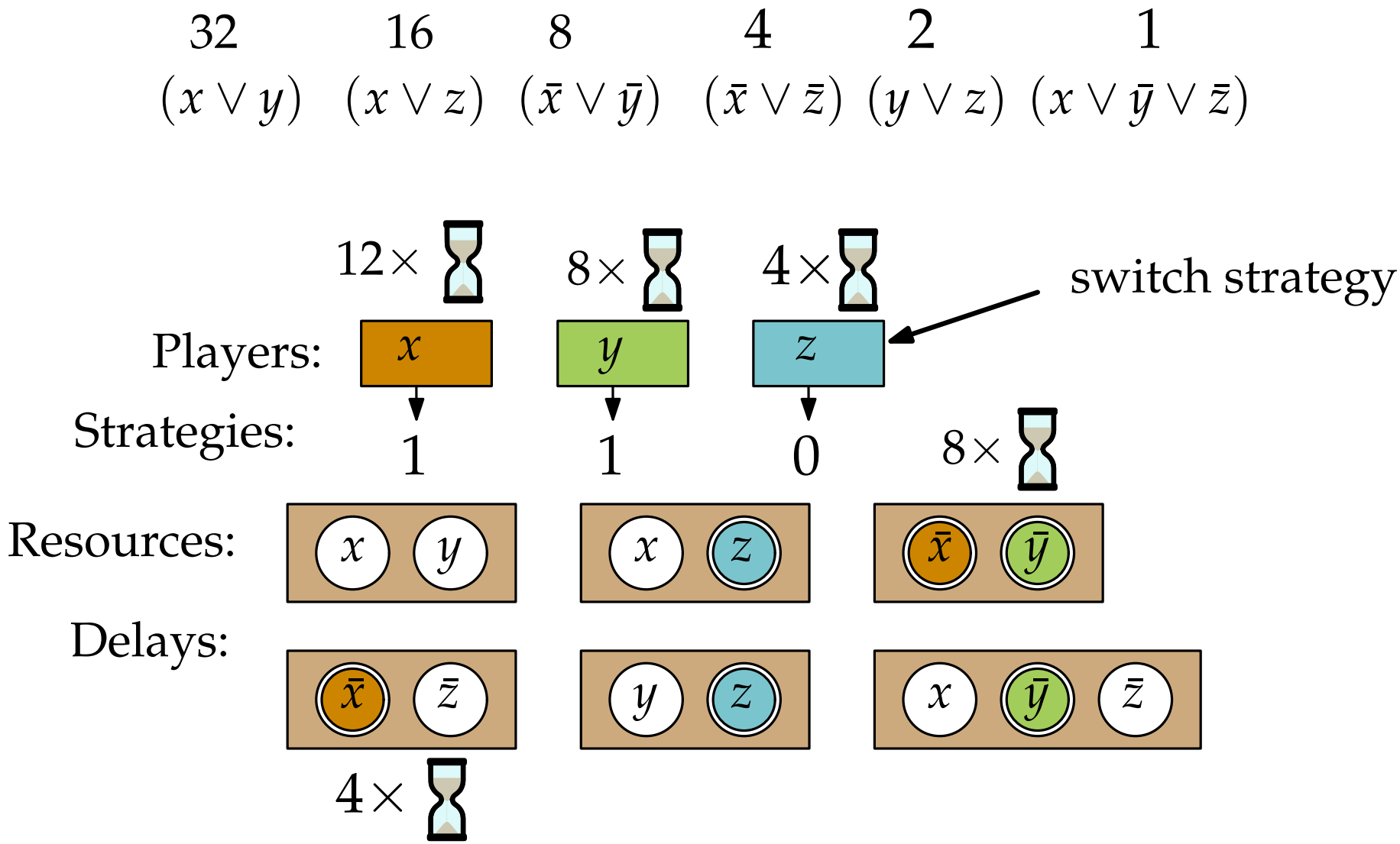
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



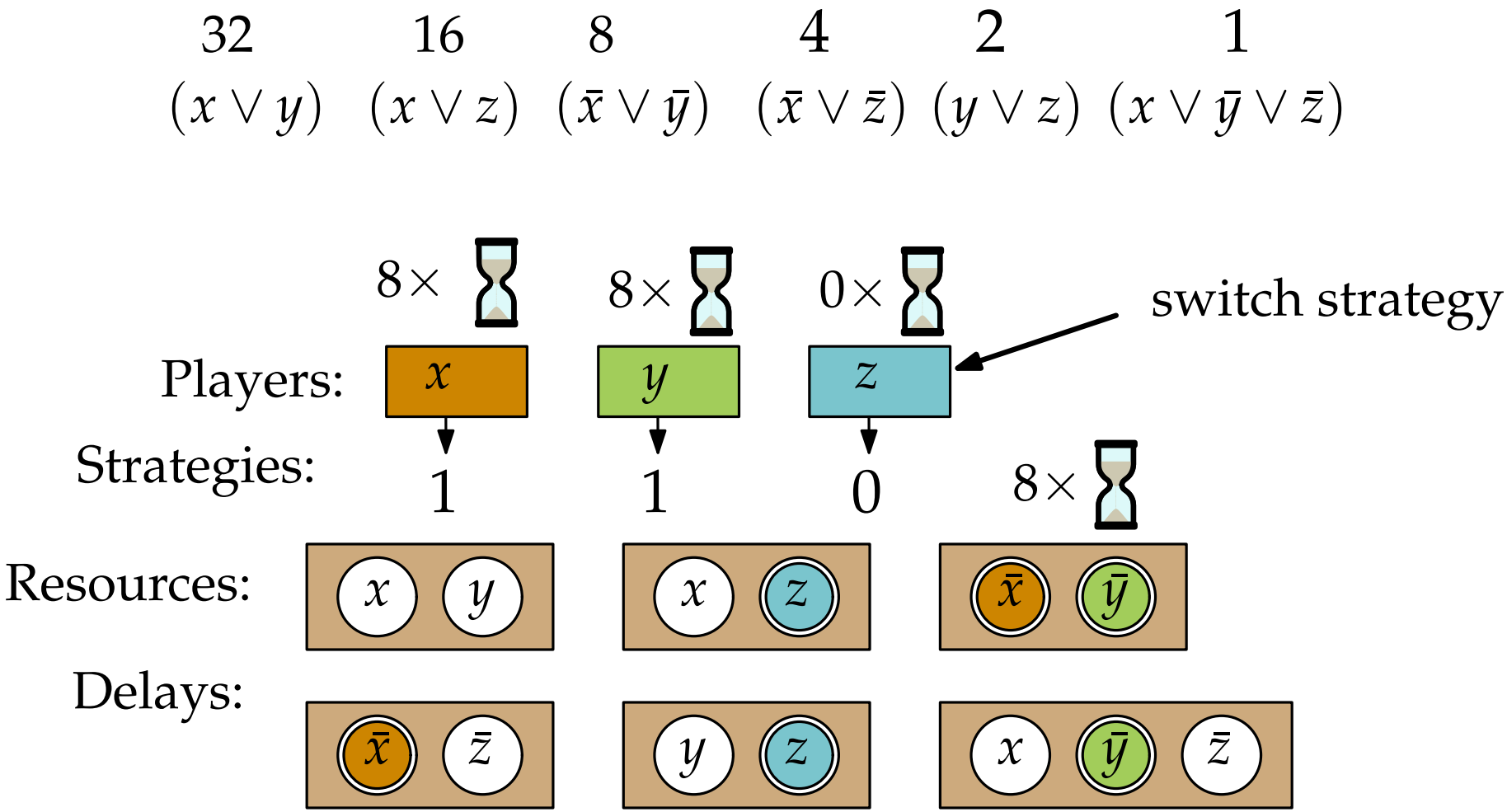
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



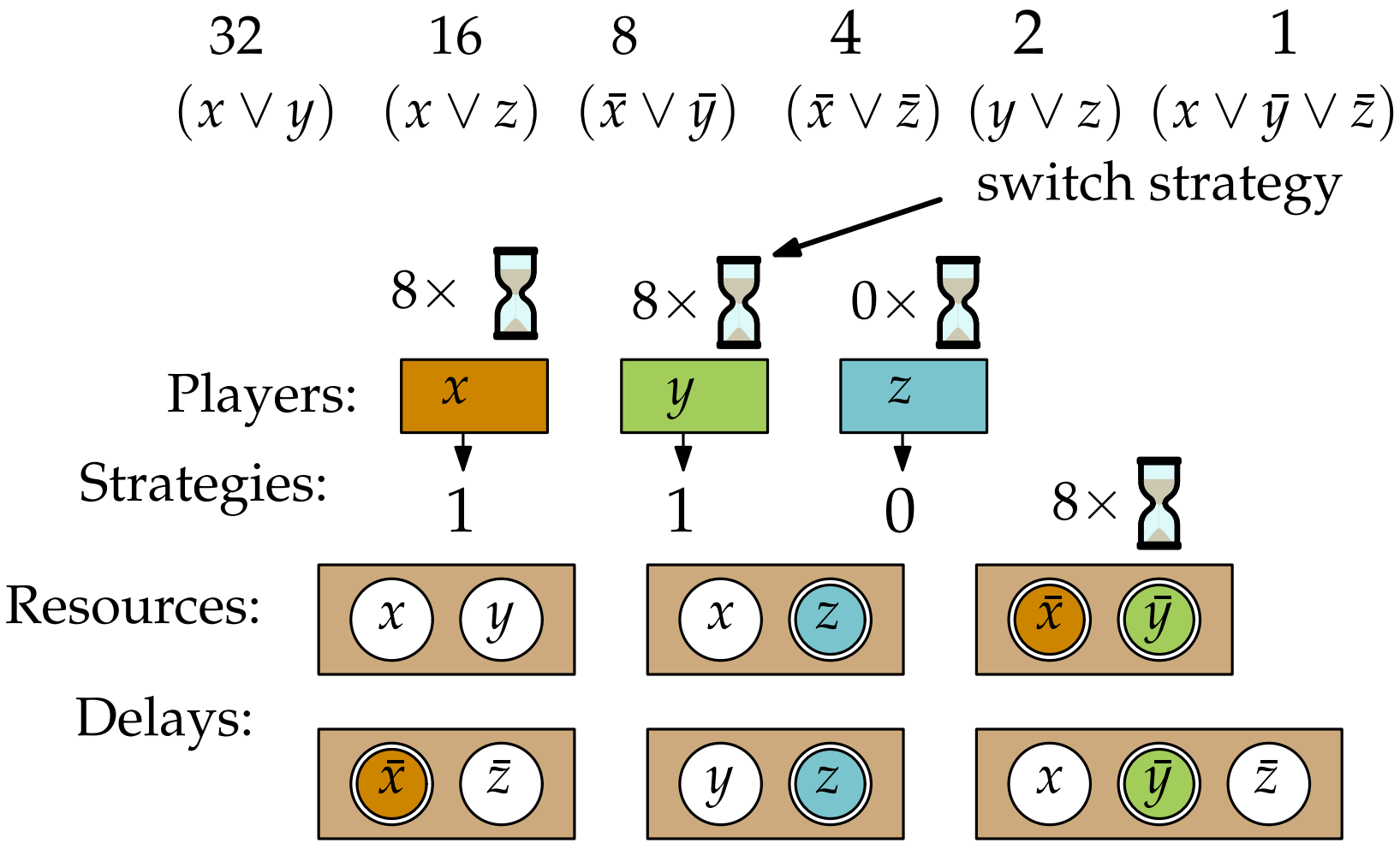
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



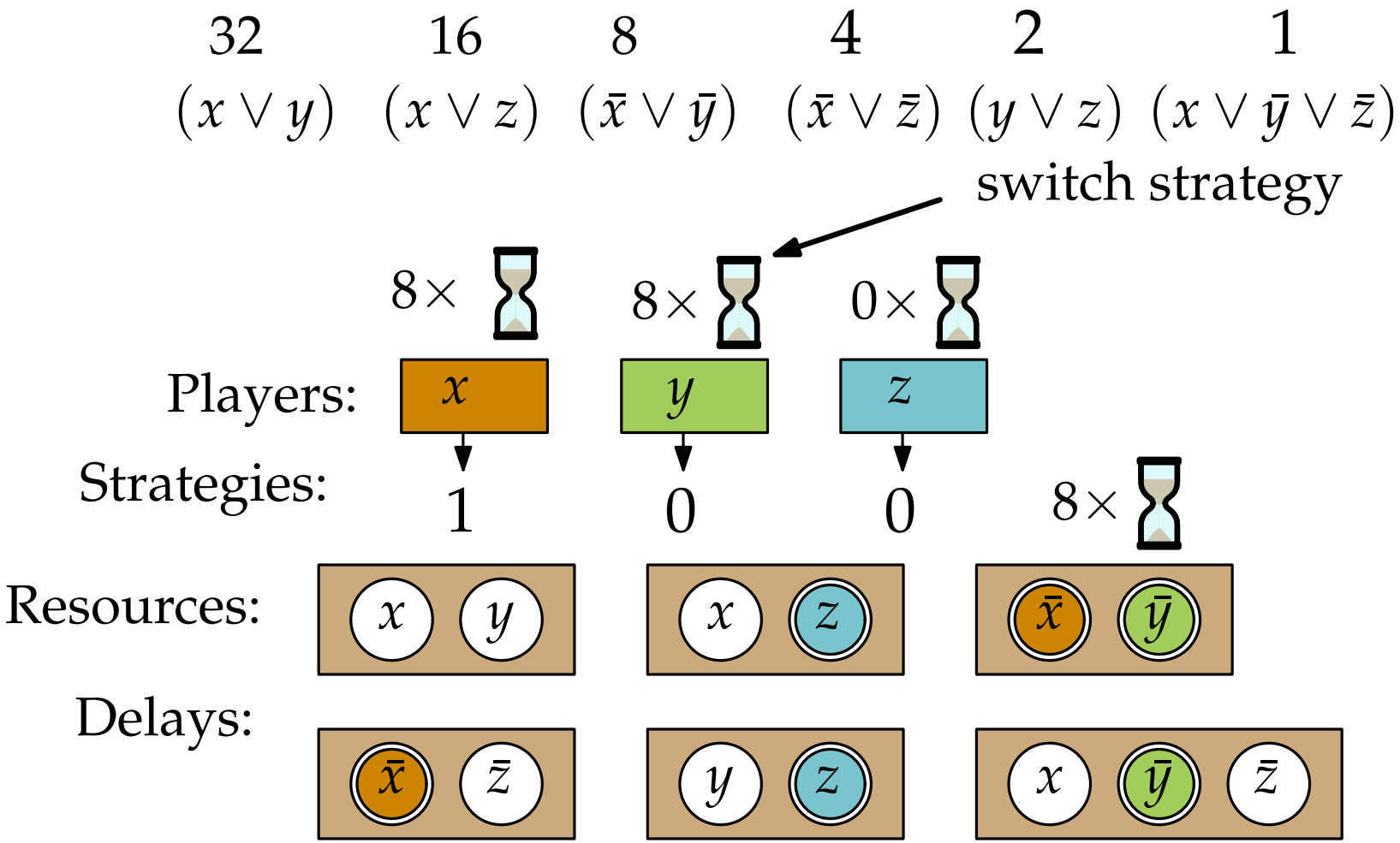
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



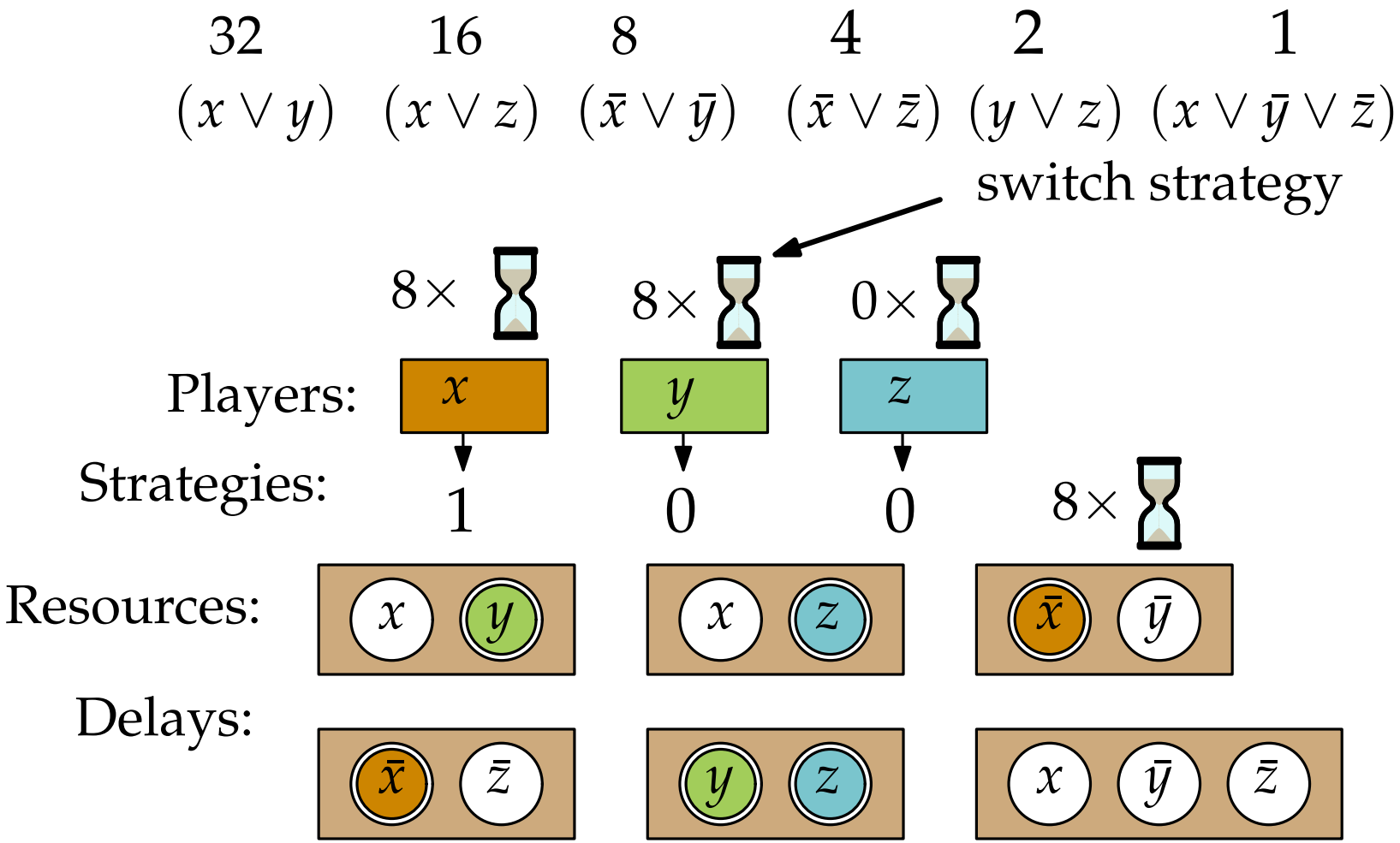
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



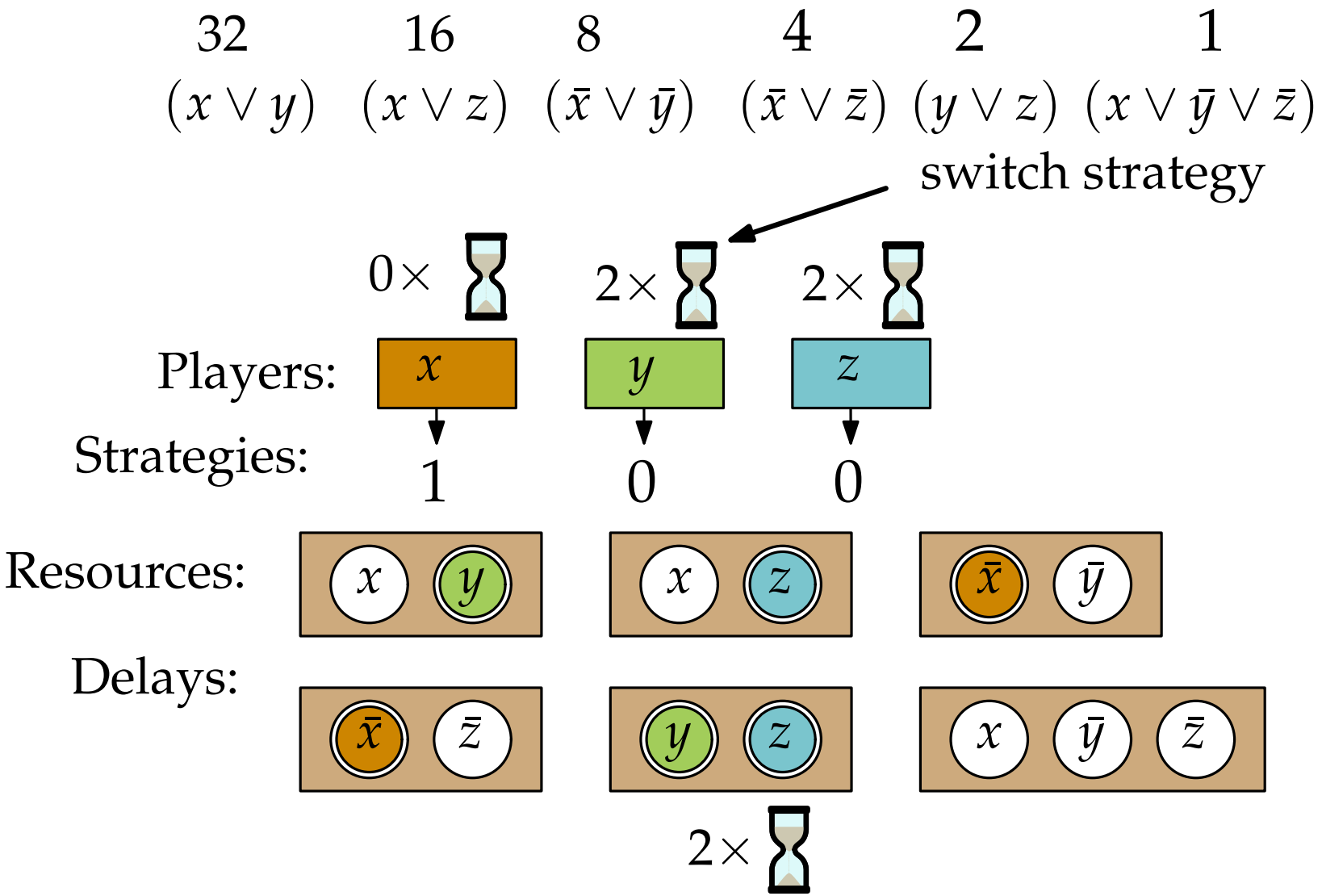
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



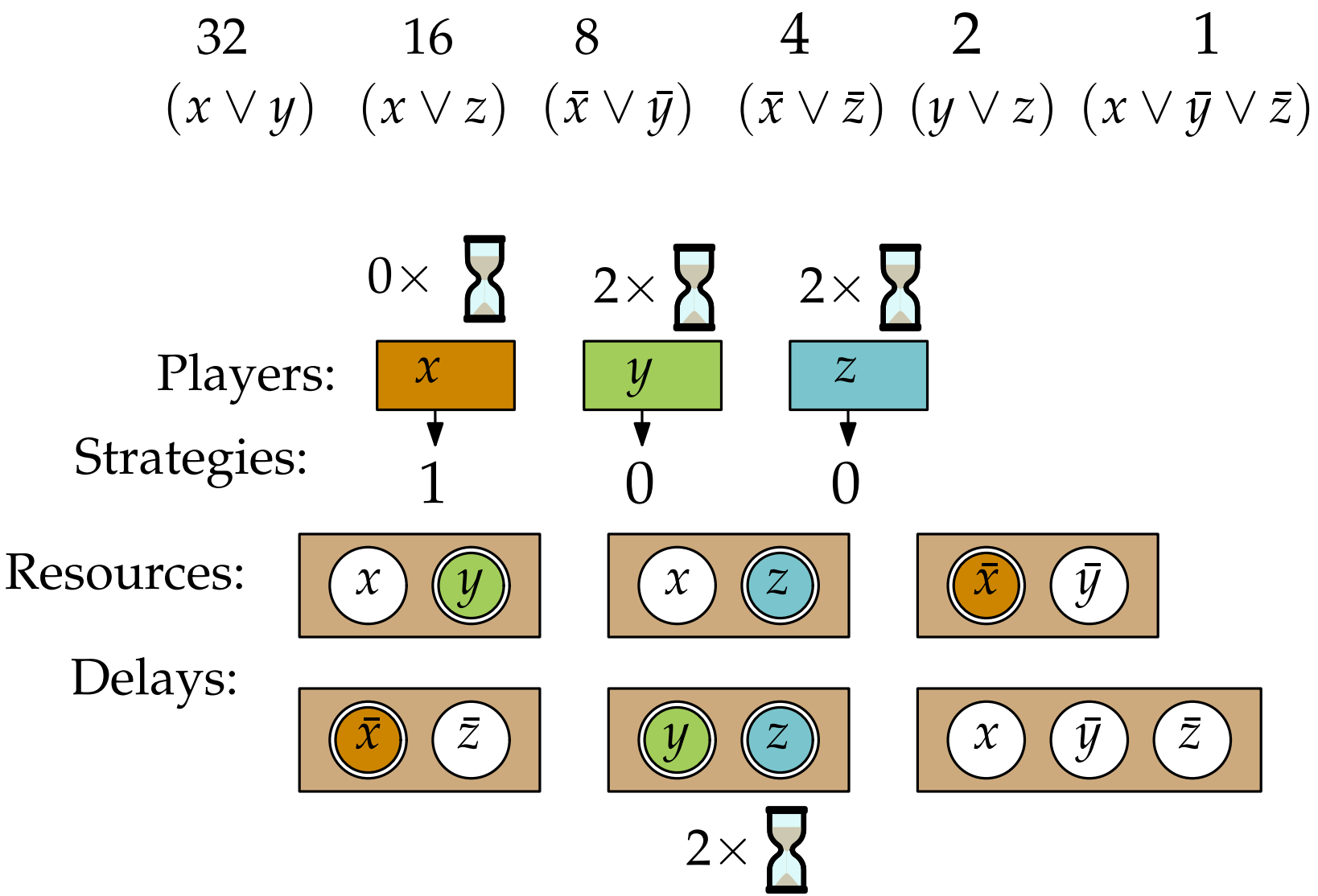
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



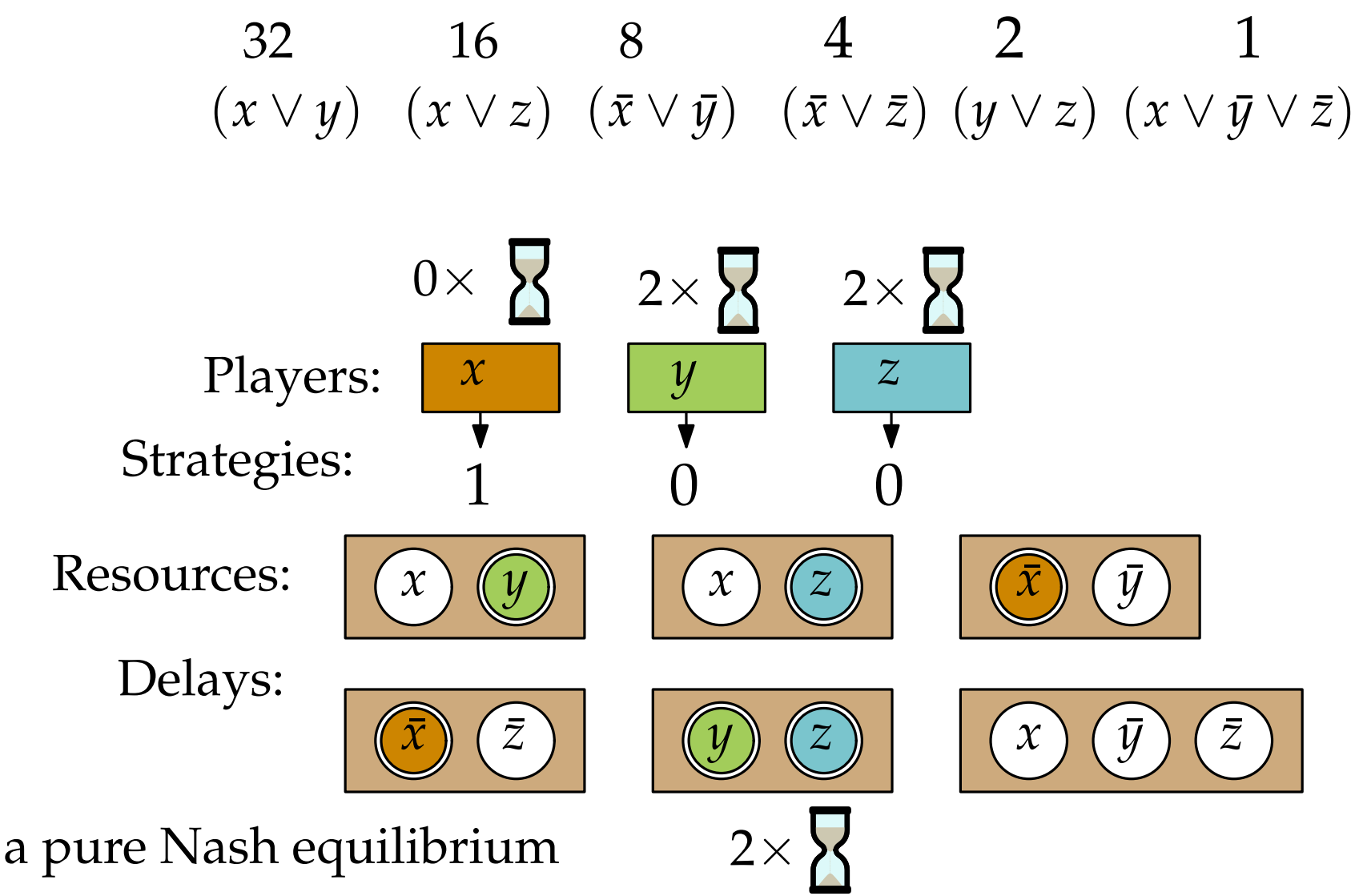
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



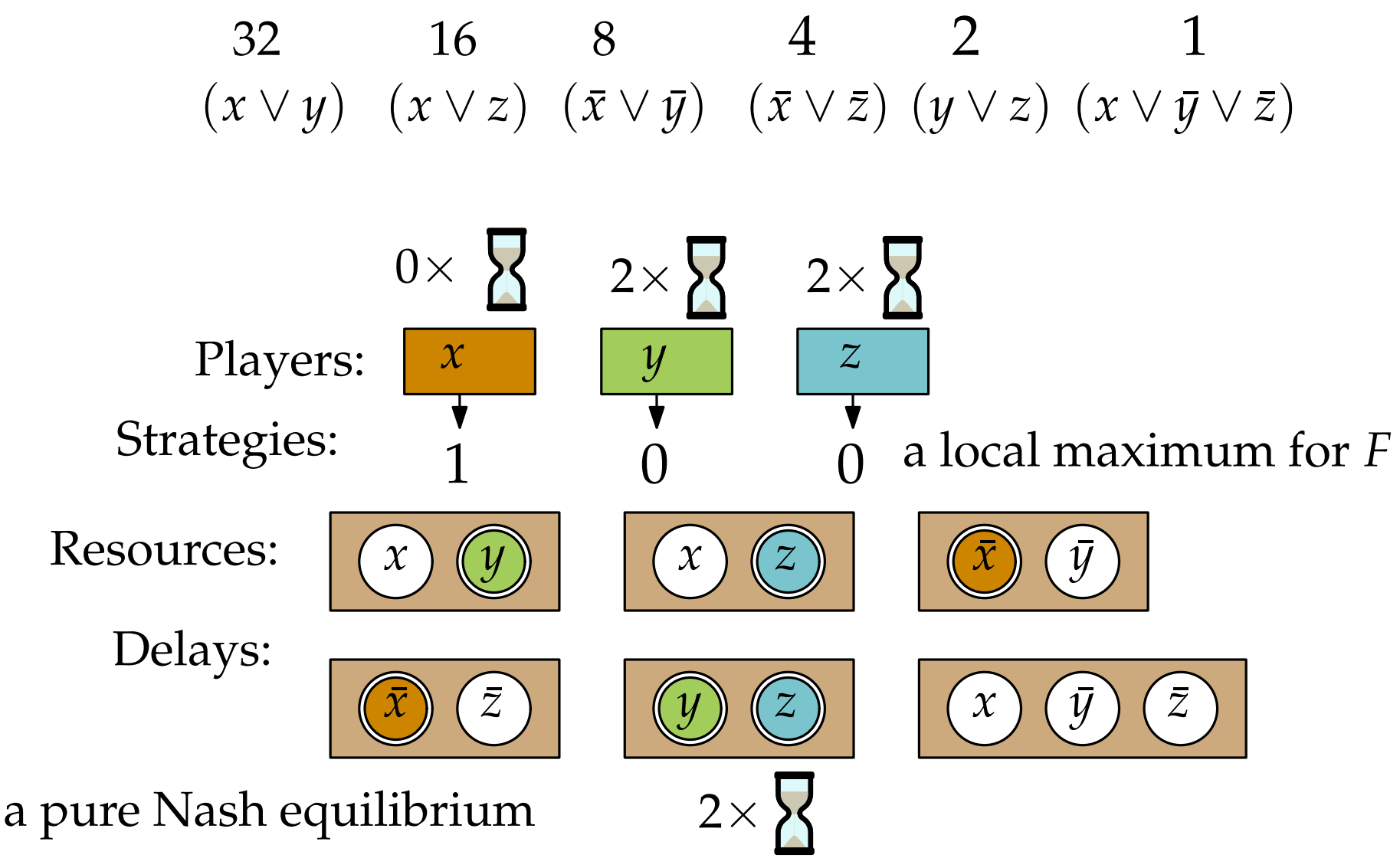
From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



From k-SAT to Congestion Games

Theorem (Ackermann and Skopalik, 2008). Finding a pure Nash equilibrium in congestion games is PLS-complete.



From k-SAT to Congestion Games

Nash equilibrium. $\forall$ player p: Switching strategy does not improve her delay.

From k-SAT to Congestion Games

Nash equilibrium. $\forall$ player p: Switching strategy does not improve her delay.

α-Nash equilibrium. $\forall$ player p: Switching strategy may improve her delay, but not more than by a factor of α.

From k-SAT to Congestion Games

Nash equilibrium. $\forall$ player p: Switching strategy does not improve her delay.

α-Nash equilibrium. $\forall$ player p: Switching strategy may improve her delay, but not more than by a factor of α.

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

From k-SAT to Congestion Games

Nash equilibrium. $\forall$ player p: Switching strategy does not improve her delay.

α-Nash equilibrium. $\forall$ player p: Switching strategy may improve her delay, but not more than by a factor of α.

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Very simple proof (S., Tantow).

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof.

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

C_1 C_2 C_3 C_4 C_5 C_6 C_7

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

We may assume the weights are "lexicographic".

C_1 C_2 C_3 C_4 C_5 C_6 C_7

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

We may assume the weights are "lexicographic".

64	32	16	8	4	2	1
C_1	C_2	C_3	C_4	C_5	C_6	C_7

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

We may assume the weights are "lexicographic".

	64	32	16	8	4	2	1
	C_1	C_2	C_3	C_4	C_5	C_6	C_7

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

We may assume the weights are "lexicographic".

	64	32	16	8	4	2	1
	C_1	C_2	C_3	C_4	C_5	C_6	C_7
α							

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

We may assume the weights are "lexicographic".

	64	32	16	8	4	2	1
	C_1	C_2	C_3	C_4	C_5	C_6	C_7
α	0	1	0	0	1	1	1

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

We may assume the weights are "lexicographic".

	64	32	16	8	4	2	1
	C_1	C_2	C_3	C_4	C_5	C_6	C_7
α	0	1	0	0	1	1	1
β							

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

We may assume the weights are "lexicographic".

	64	32	16	8	4	2	1
	C_1	C_2	C_3	C_4	C_5	C_6	C_7
α	0	1	0	0	1	1	1
β	0	1	0	1	0	0	0

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

We may assume the weights are "lexicographic".

	64	32	16	8	4	2	1
	C_1	C_2	C_3	C_4	C_5	C_6	C_7
α	0	1	0	0	1	1	1
β	0	1	0	1	0	0	0

Which is better?

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

We may assume the weights are "lexicographic".

	64	32	16	8	4	2	1
	C_1	C_2	C_3	C_4	C_5	C_6	C_7
α	0	1	0	0	1	1	1
β	0	1	0	1	0	0	0

Which is better? No need for computation! Of course β is better!

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

We may assume the weights are "lexicographic".

	729	243	81	27	9	3	1
	C_1	C_2	C_3	C_4	C_5	C_6	C_7
α	0	1	0	0	1	1	1
β	0	1	0	1	0	0	0

Which is better? No need for computation! Of course β is better!

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

We may assume the weights are "lexicographic".

	729	243	81	27	9	3	1
	C_1	C_2	C_3	C_4	C_5	C_6	C_7
α	0	1	0	0	1	1	1
β	0	1	0	1	0	0	0

Which is better? No need for computation! Of course β is better!

More than twice as heavy than those guys!

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

$F =$ 

From k-SAT to Congestion Games

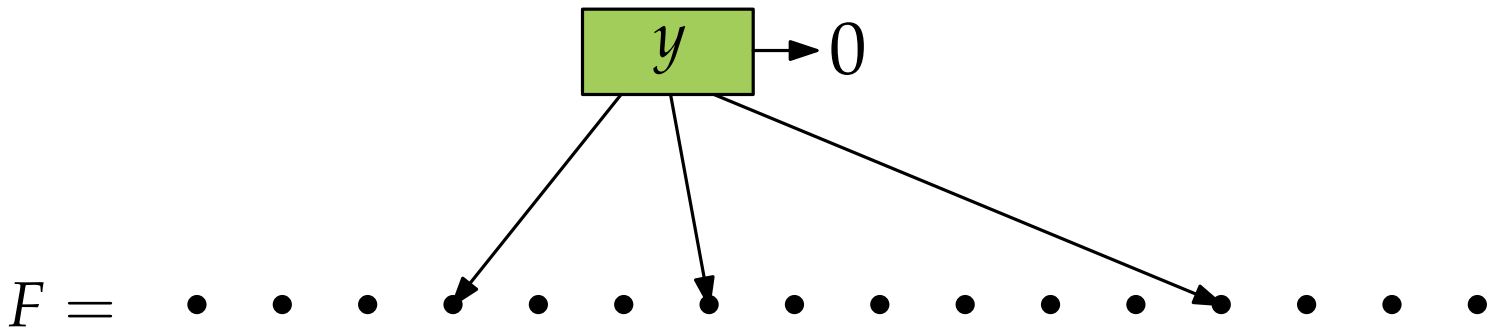
Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

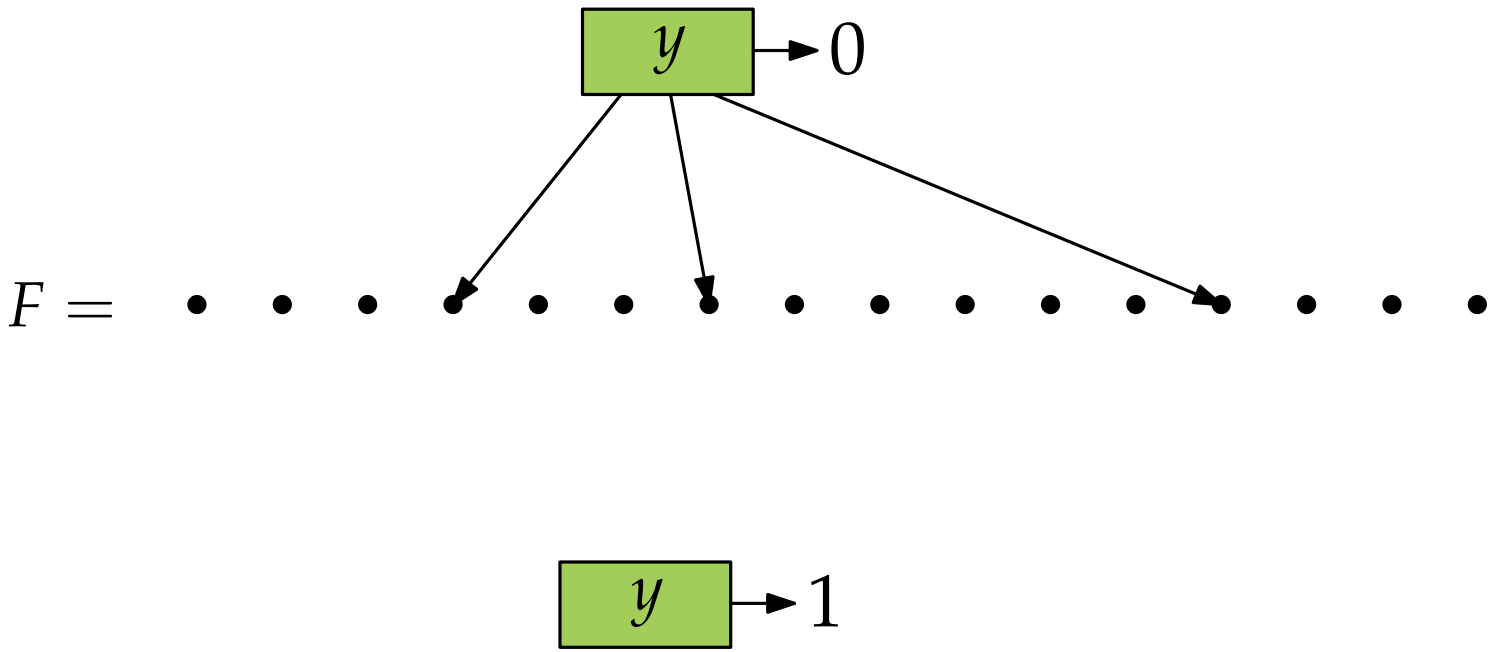
Proof. Consider an instance of Local Max-k-SAT:



From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

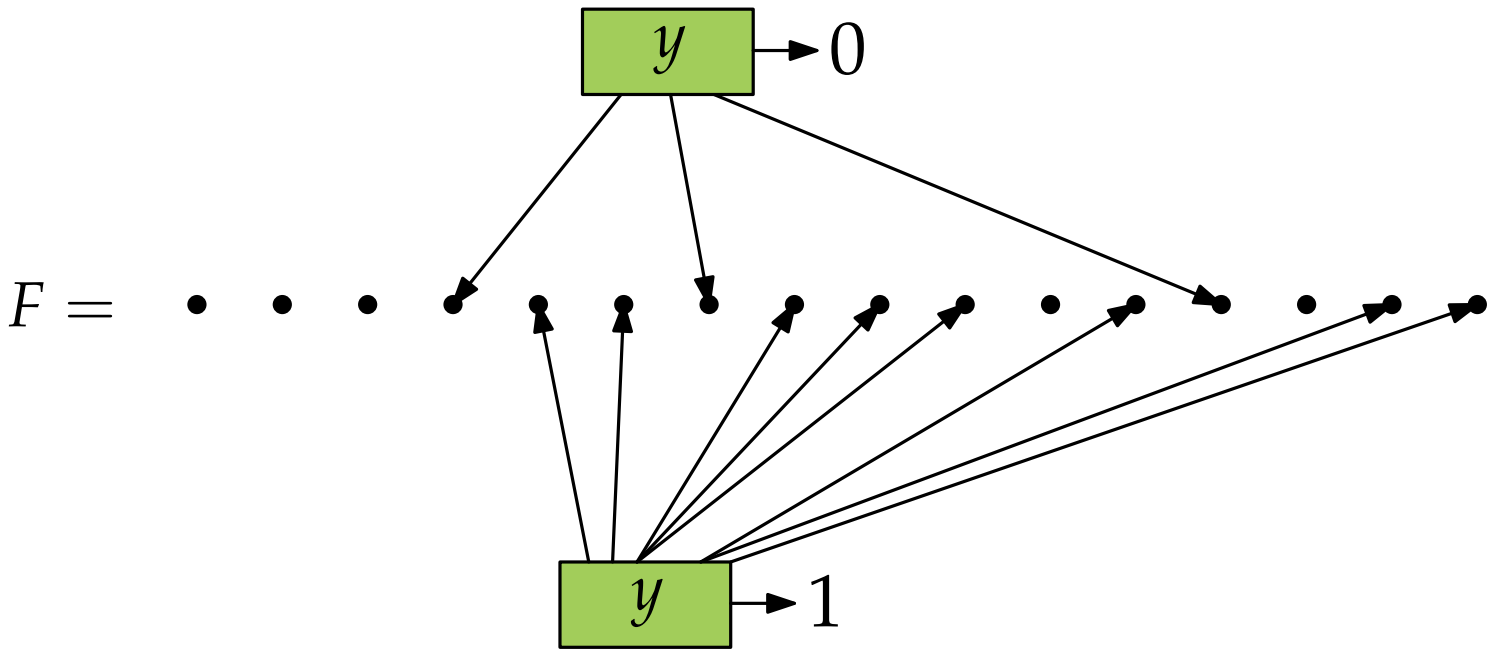


Now suppose switching to 1 reduces the delay of Player y

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

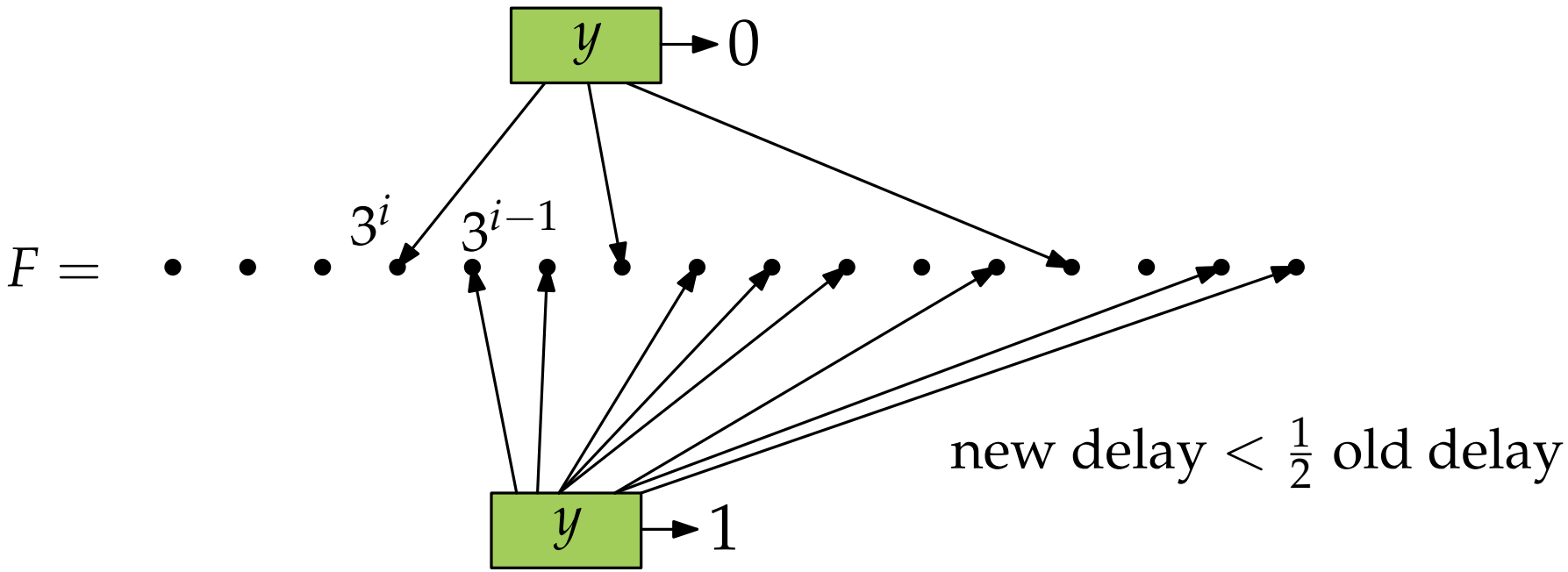


Now suppose switching to 1 reduces the delay of Player y

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:

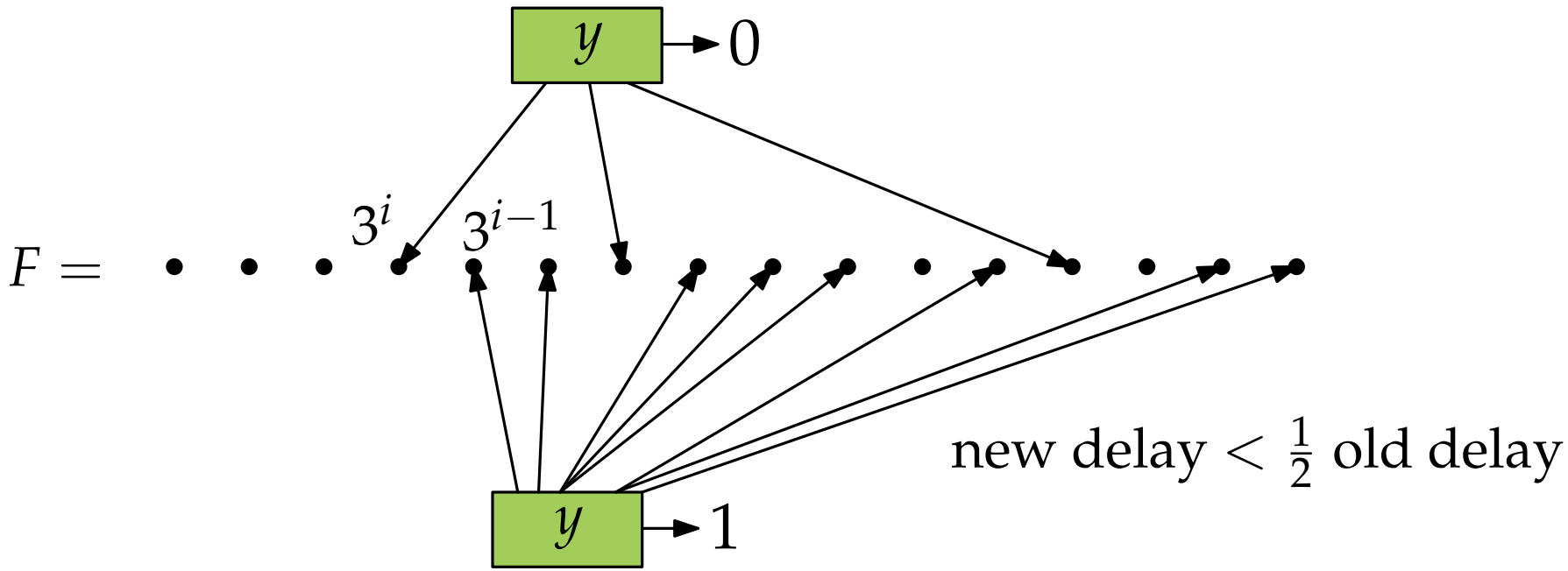


Now suppose switching to 1 reduces the delay of Player y

From k-SAT to Congestion Games

Theorem (Skopalik and Vöcking). Finding a pure α-Nash equilibrium in congestion games is PLS-hard.

Proof. Consider an instance of Local Max-k-SAT:



Now suppose switching to 1 reduces the delay of Player y

A 2-Nash equilibrium will give us a locally optimal truth assignment!

Gràcies per la vostra atenció!

¡Gracias por su atención!

Danke für Eure Aufmerksamkeit!

Thank you for your attention!