ETH zürich

D-INFK

ETH zürich

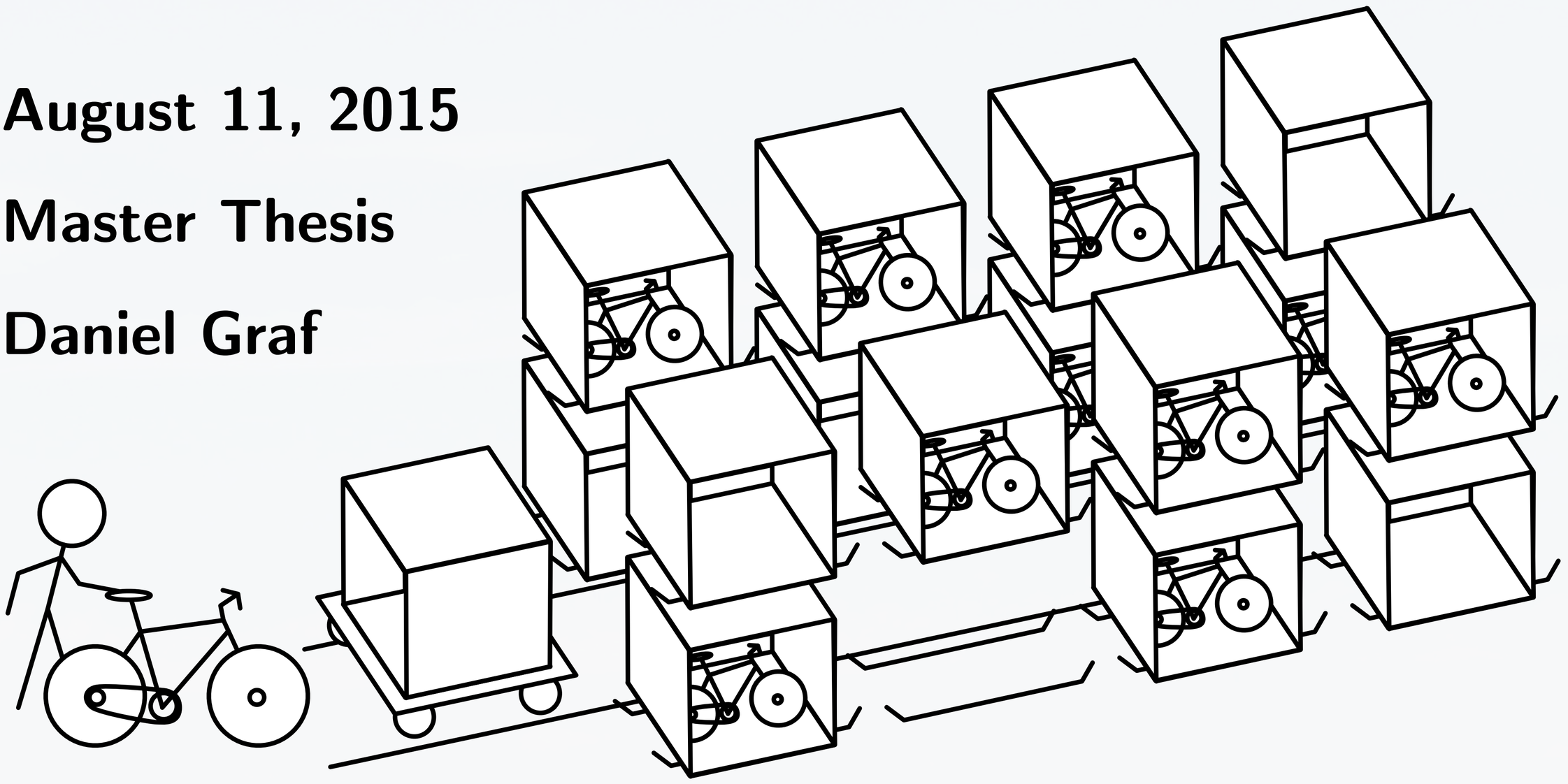
D-INFK

# Scheduling and Sorting Algorithms for Robotic Warehousing Systems

**August 11, 2015**

**Master Thesis**

**Daniel Graf**



ETH zürich
Daniel Graf

# Motivation



ETH zürich
Daniel Graf

# Bike Loft

1. Introduction
1.1 Motivation


Video by 3sat and Bike Loft GmbH

ETH zürich
Daniel Graf

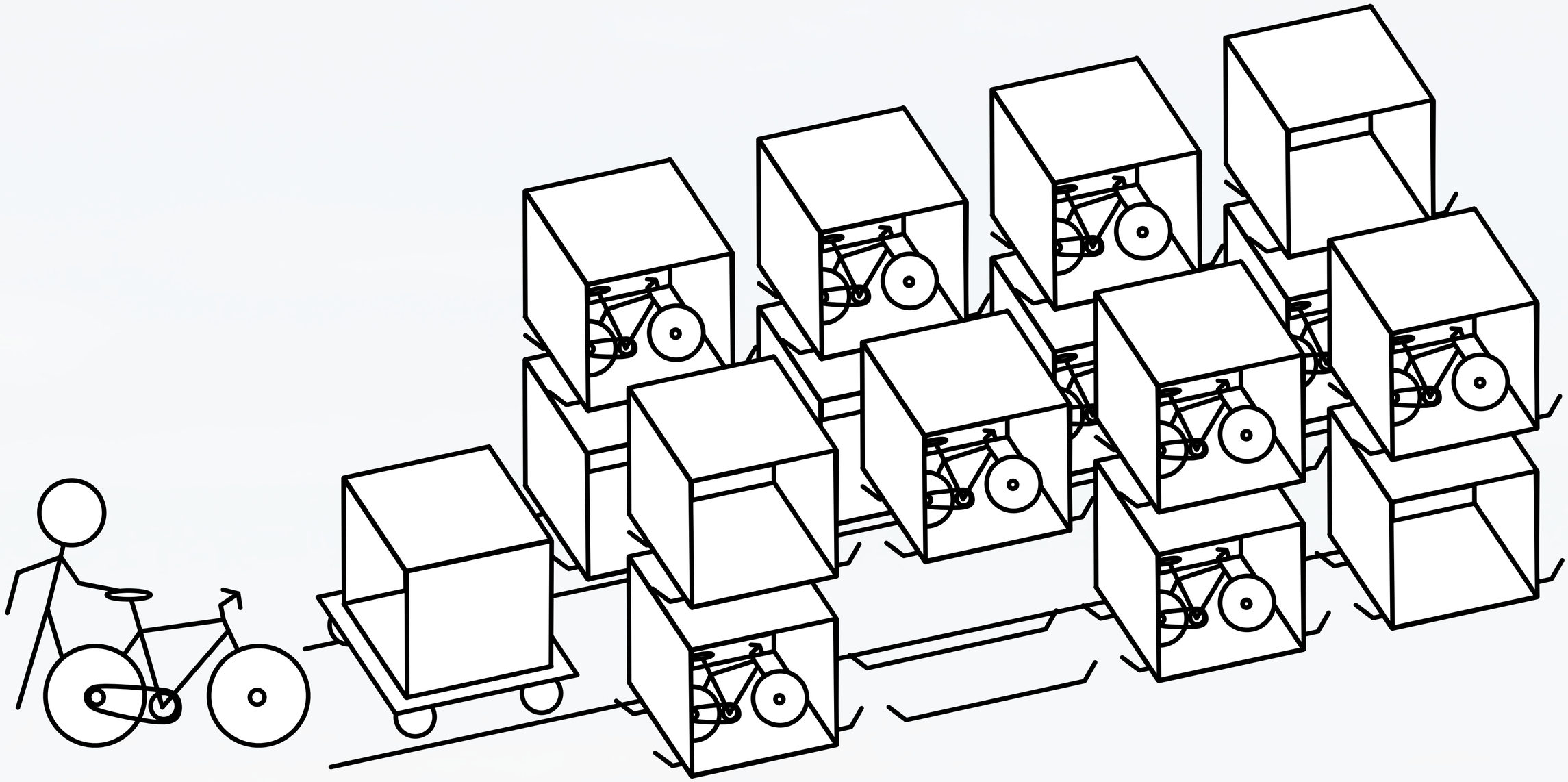
# Bike Loft

1. Introduction
1.1 Motivation

ETH zürich
Daniel Graf

# Bike Loft

1. Introduction
1.1 Motivation



# Overview



Scheduling problem

- How do we efficiently serve the customers?

Sorting problem

- How do we efficiently rearrange the stored boxes?

Survey of other bicycle storage systems

ETH zürich

Daniel Graf

# Scheduling



# Scheduling



Given:

- robot specification
- customer arrival and departure times

Wanted:

- customer-to-slot-and-door-assignment
  that does not keep the customers waiting

ETH zürich

Daniel Graf

# Sorting

ETH zürich

Daniel Graf

# Sorting

1. Introduction

1.2 Overview

# Sorting



Given:

- a graph that describes the layout
- an initial permutation of the boxes
- starting position of the robot

Wanted:

- shortest possible sorting walk

# Contributions



Scheduling Problem

- algorithms for arrival-only with a single door
- *NP*-hardness of the two-door problem
- *NP*-hardness of the all-day problem

Sorting Problem

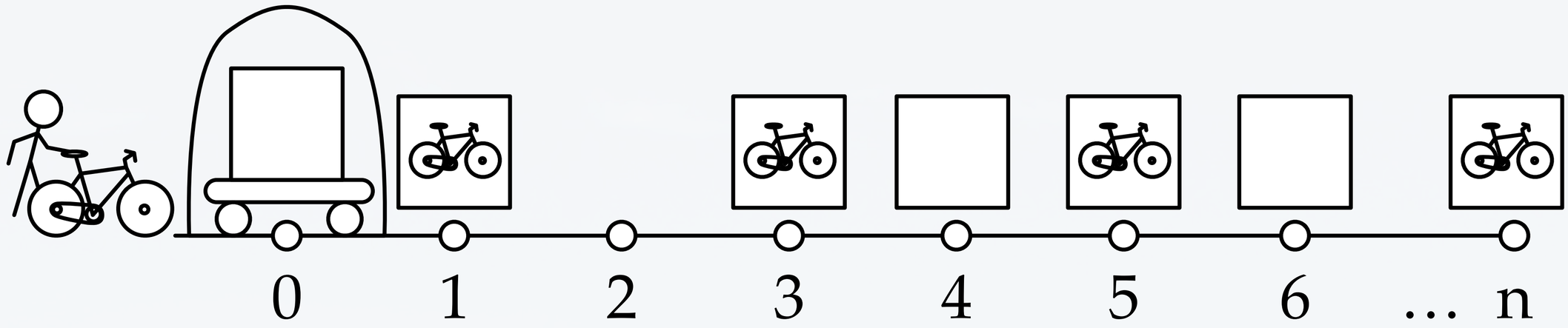
- algorithms for sorting on paths, trees and cycles
- *NP*-hardness for planar graphs
- *NP*-hardness for trees when minimizing swaps

ETH zürich
Daniel Graf

# Scheduling Problem

# Model






Storage System Model

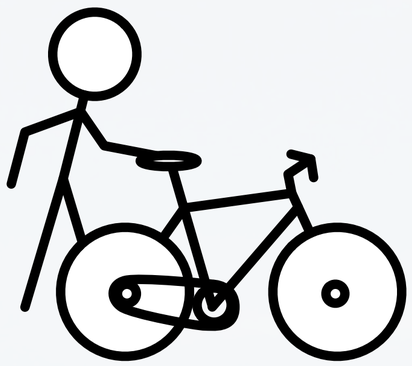
- one-dimensional layout
- $n$ slots that can store a single box each
- initially all boxes are empty
- single door for customer interaction

ETH zürich
Daniel Graf

# Customer Model

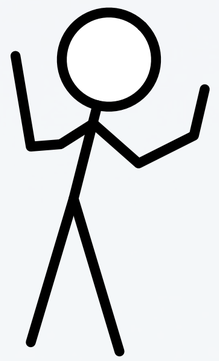


$t_i^a$



arrival

$t_i^d$



departure

Customer Model

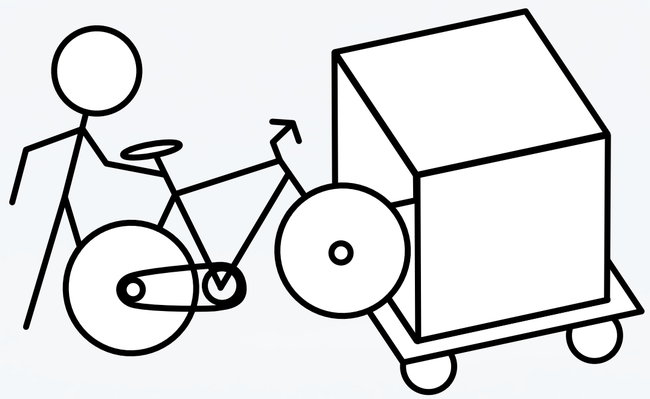
- $m$ customers, each with arrival time $t_i^a$ and departure time $t_i^d$ that are known in advance

ETH zürich
Daniel Graf

# Customer Model

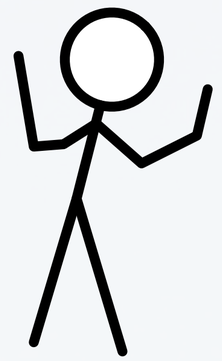


$t_i^a$



arrival

$t_i^d$



departure

Customer Model

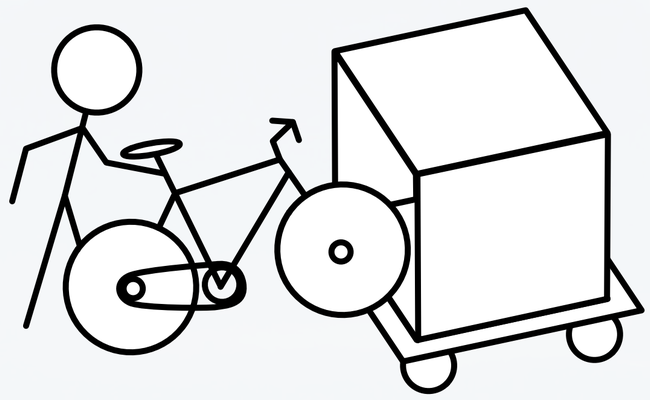
- $m$ customers, each with arrival time $t_i^a$ and departure time $t_i^d$ that are known in advance

ETH zürich
Daniel Graf

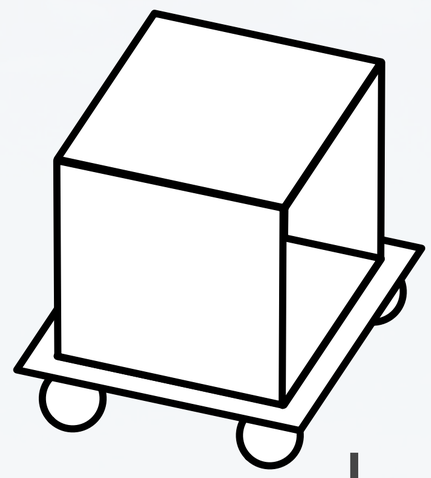
# Customer Model



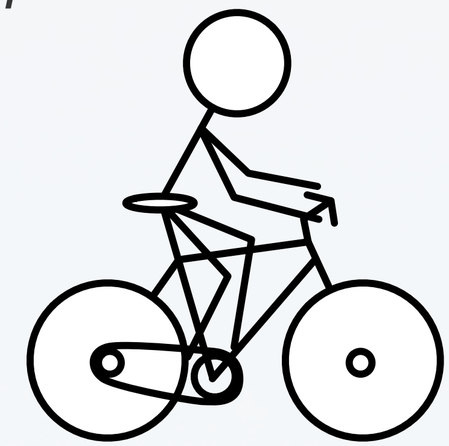
$t_i^a$



arrival



$t_i^d$



departure

Customer Model

- $m$ customers, each with arrival time $t_i^a$ and departure time $t_i^d$ that are known in advance

ETH zürich

Daniel Graf

# Robot Model



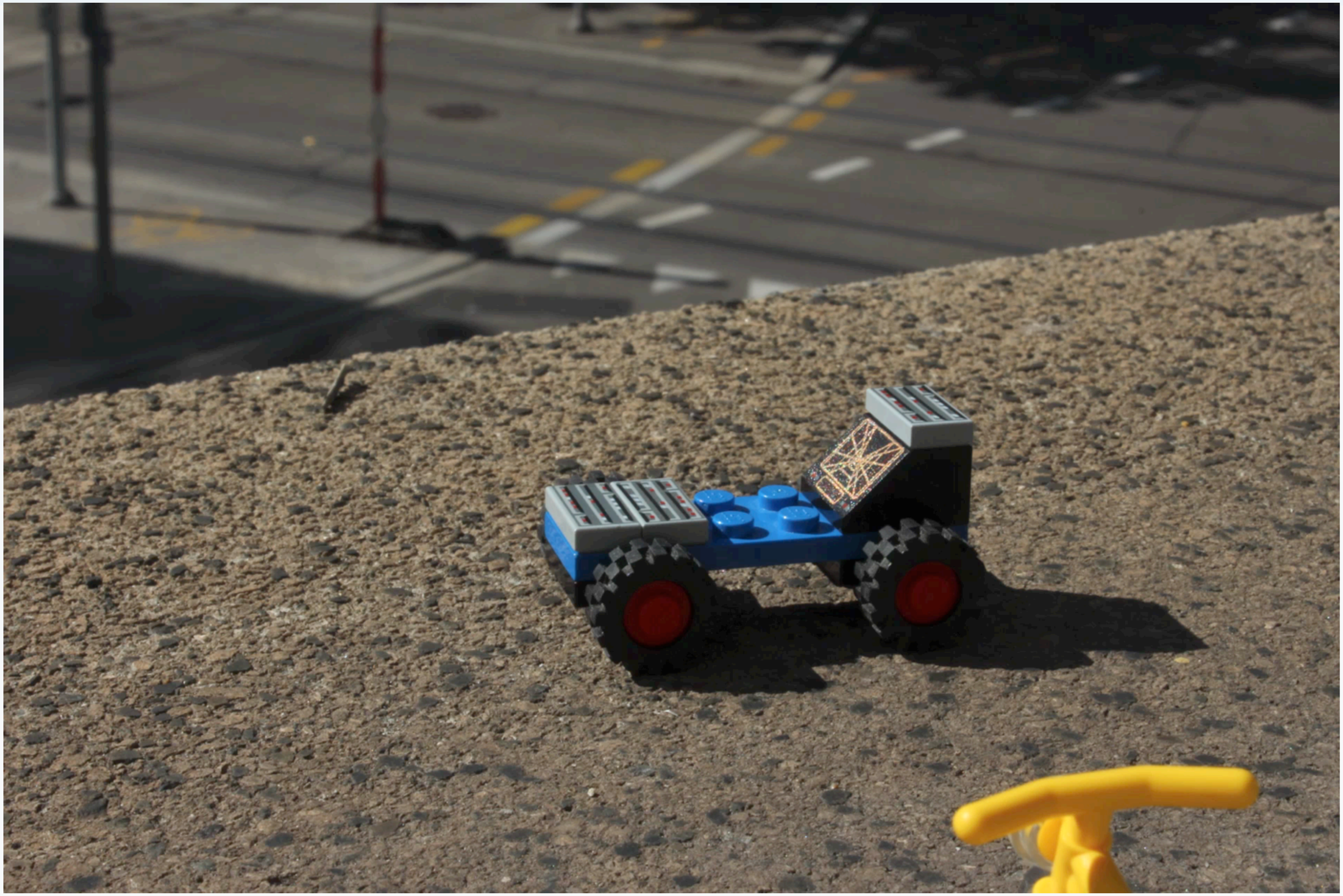
ETH zürich

Daniel Graf

# Robot Model

2. Scheduling

2.1 Model



ETH zürich

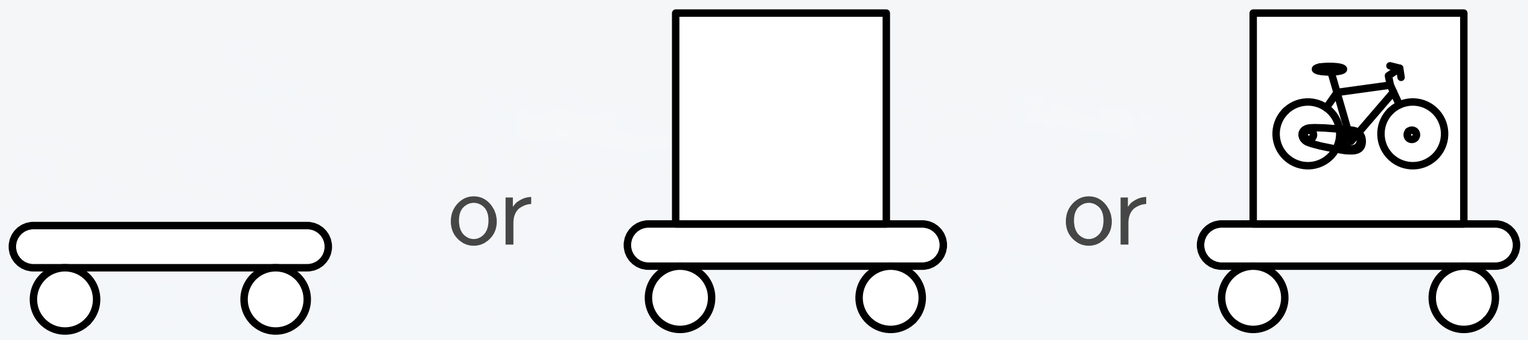
Daniel Graf

# Robot Model



# Robot Model





## Robot Model

- a single box at a time
- unit velocity, no acceleration
- instant box handling
- starts at the door
- distinction: can boxes be swapped or not?

# Model



Goal

- no customer should ever wait at the door
  (neither for an empty box nor for his bike)

Task

- find a robot schedule that achieves this

Assumption

- boxes not rearranged while in storage

ETH zürich
Daniel Graf

# Arrival Only



Early morning scenario - no departures yet.

# Arrival Only



Every slot can be used for at most one customer.

If $n = m$, we can describe the schedule by a permutation $\pi \in S_n$.

The box of customer $i$ gets stored at slot $\pi(i)$.

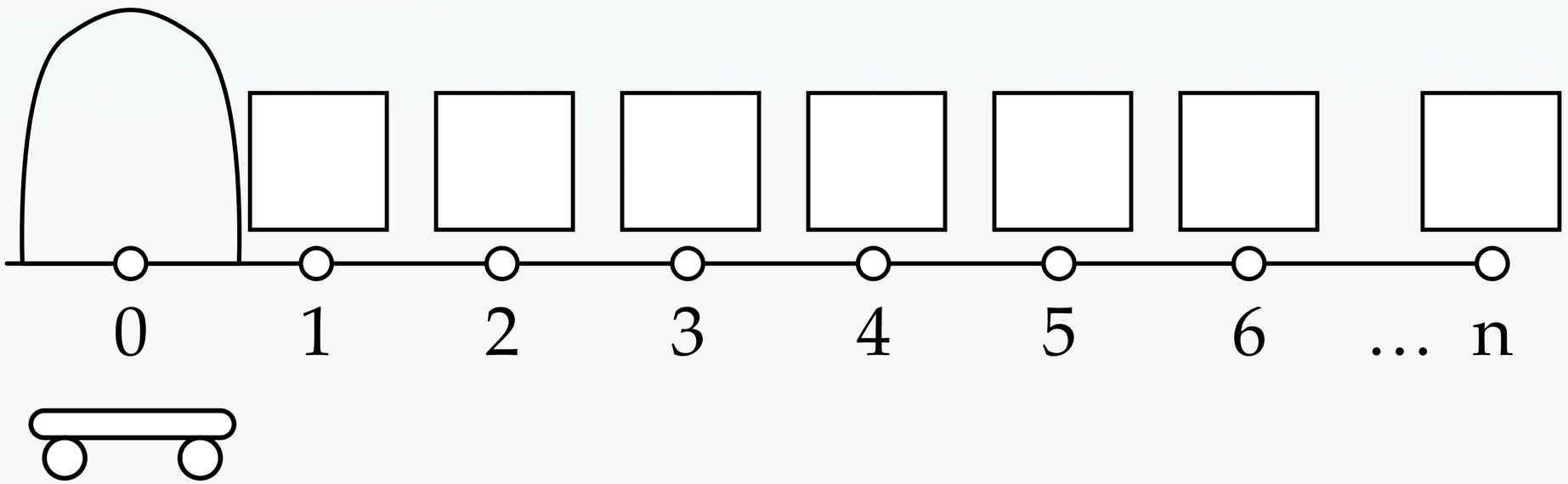
Which permutations correspond to feasible, wait-free robot schedules?

# No Swapping



No swaps and initially no box on the robot.

How to serve the first customer?



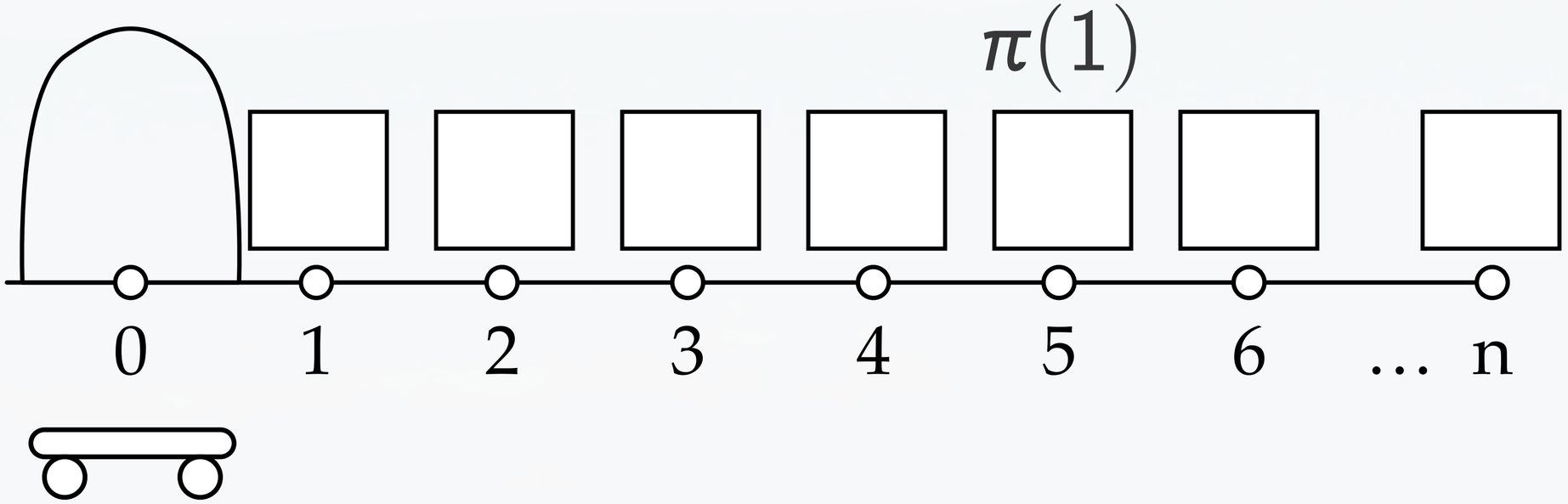
# No Swapping



No swaps and initially no box on the robot.

How to serve the first customer?

Getting the empty box at slot $\pi(1)$ ready for the first customer takes time $2\pi(1)$.



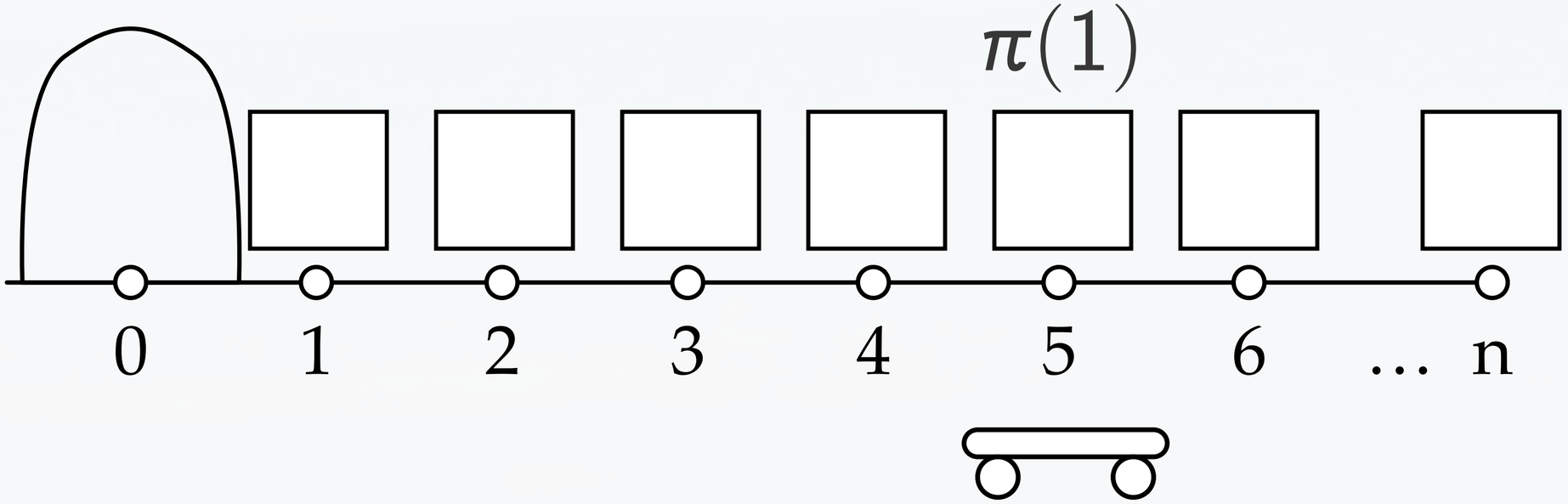
# No Swapping



No swaps and initially no box on the robot.

How to serve the first customer?

Getting the empty box at slot $\pi(1)$ ready for the first customer takes time $2\pi(1)$.



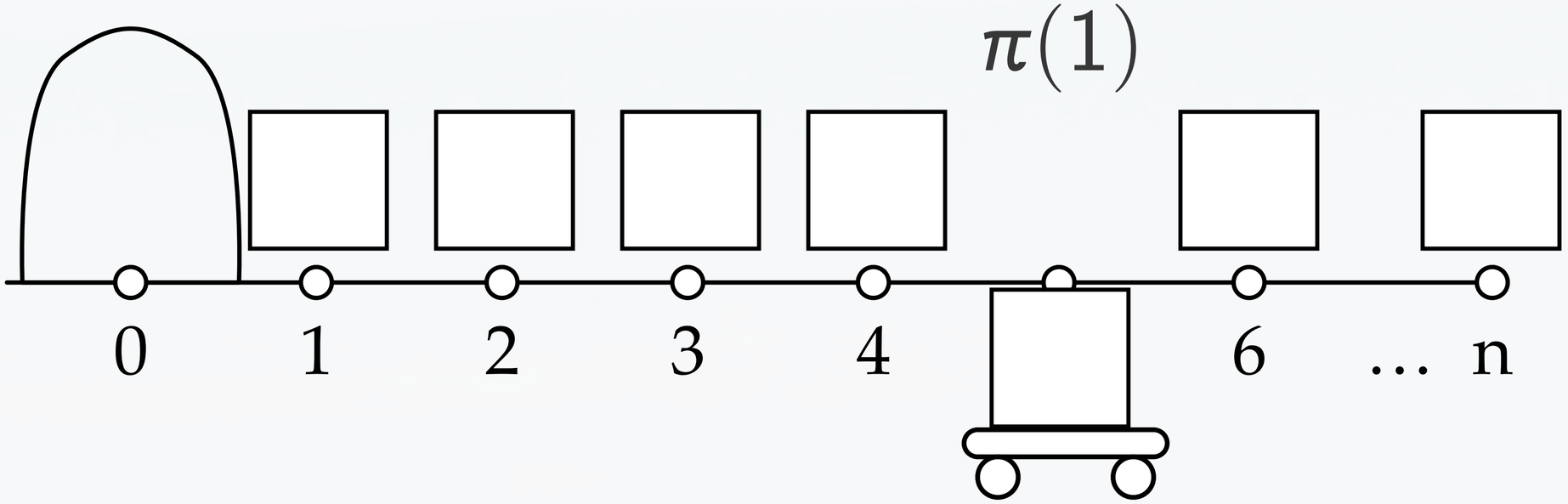
# No Swapping



No swaps and initially no box on the robot.

How to serve the first customer?

Getting the empty box at slot $\pi(1)$ ready for the first customer takes time $2\pi(1)$.



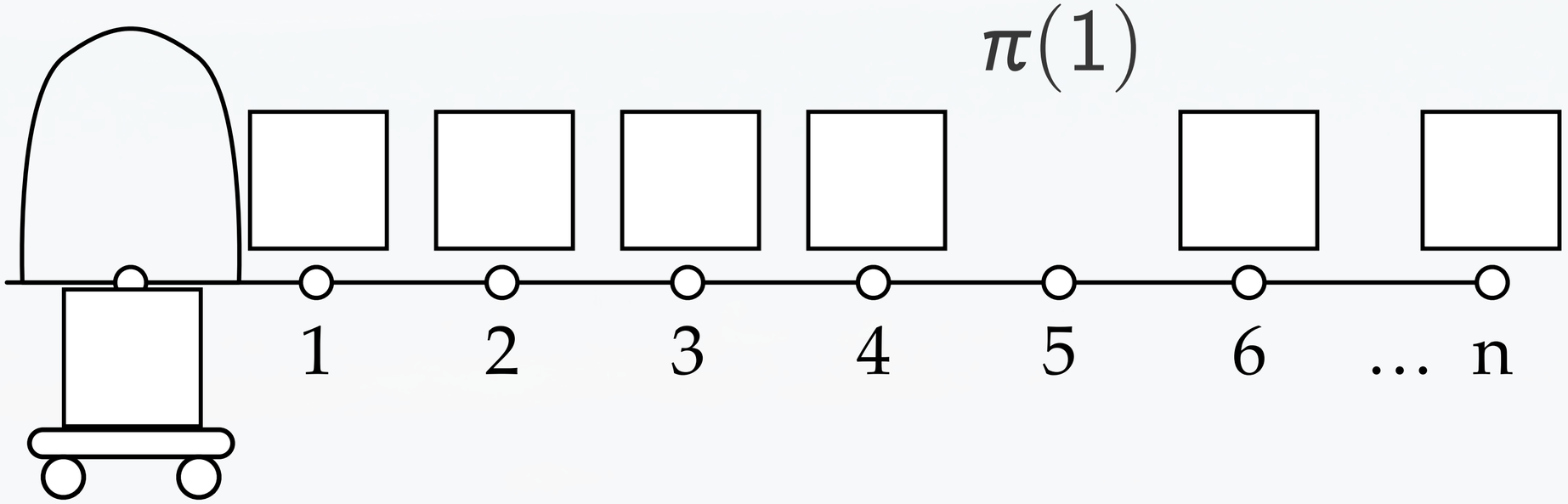
# No Swapping



No swaps and initially no box on the robot.

How to serve the first customer?

Getting the empty box at slot $\pi(1)$ ready for the first customer takes time $2\pi(1)$.



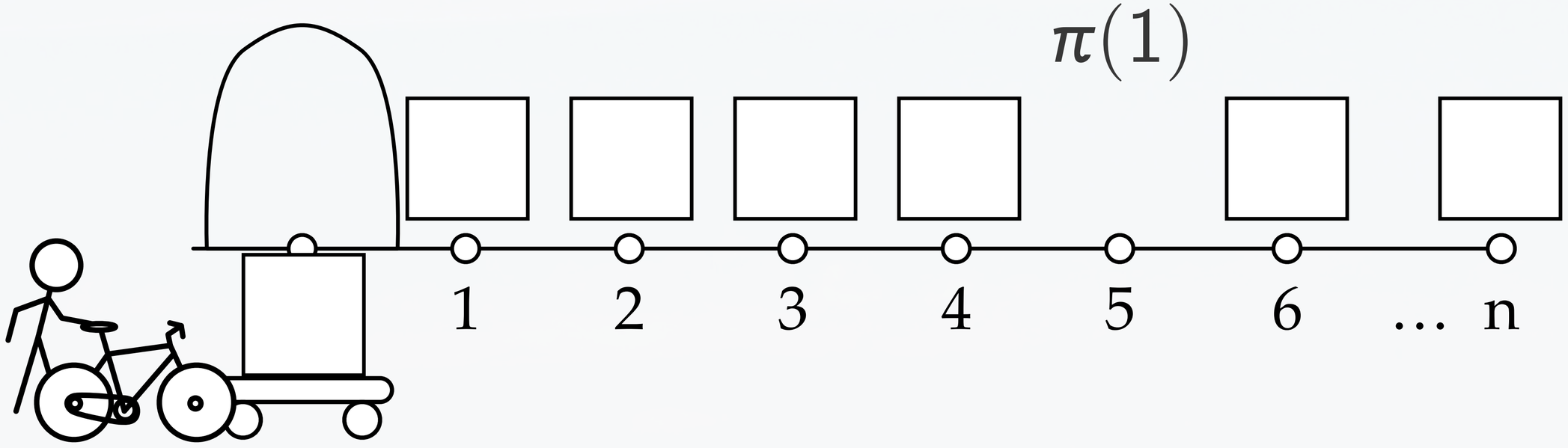
# No Swapping



No swaps and initially no box on the robot.

How to serve the first customer?

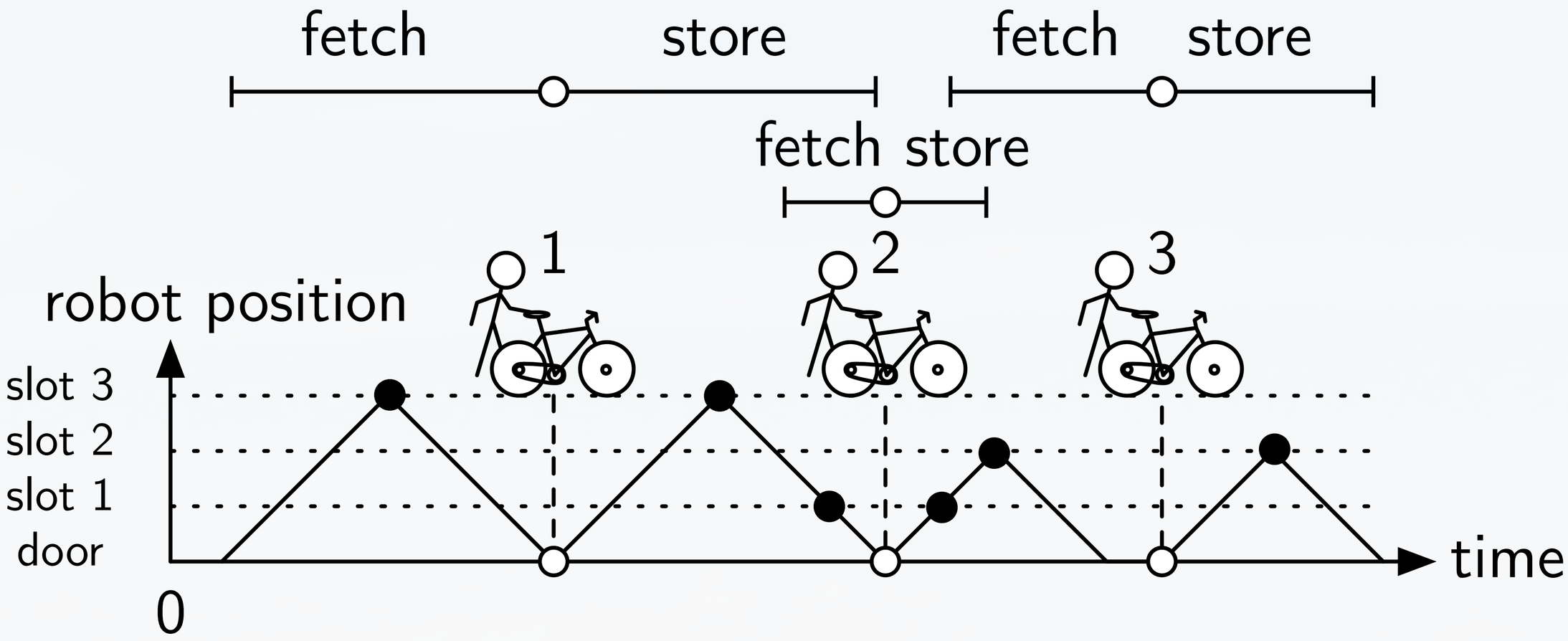
Getting the empty box at slot $\pi(1)$ ready for the first customer takes time $2\pi(1)$.



# No Swapping



Example:



# No Swapping



Between the arrivals of customers *i and i+1*:

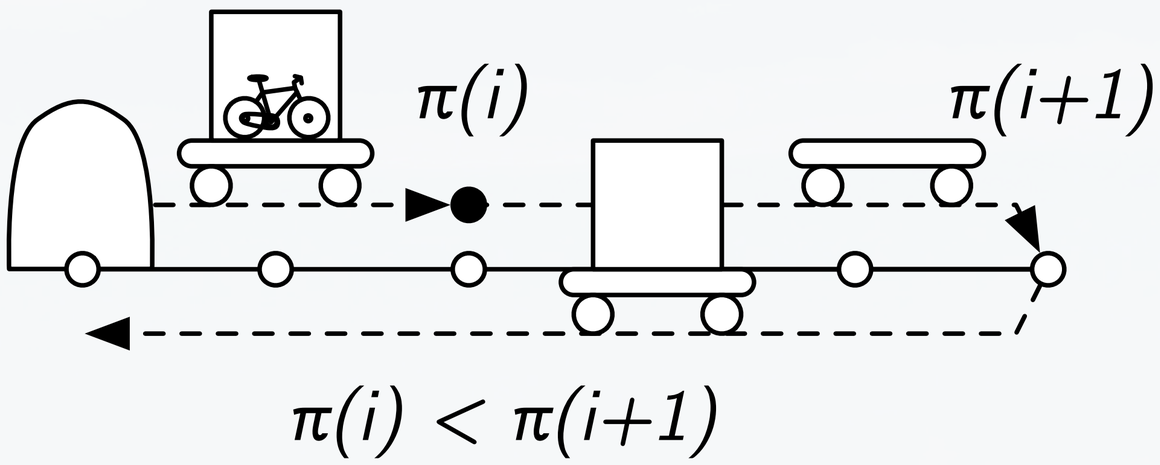
- store the full box at slot $\pi(i)$
- fetch the empty box from slot $\pi(i+1)$.

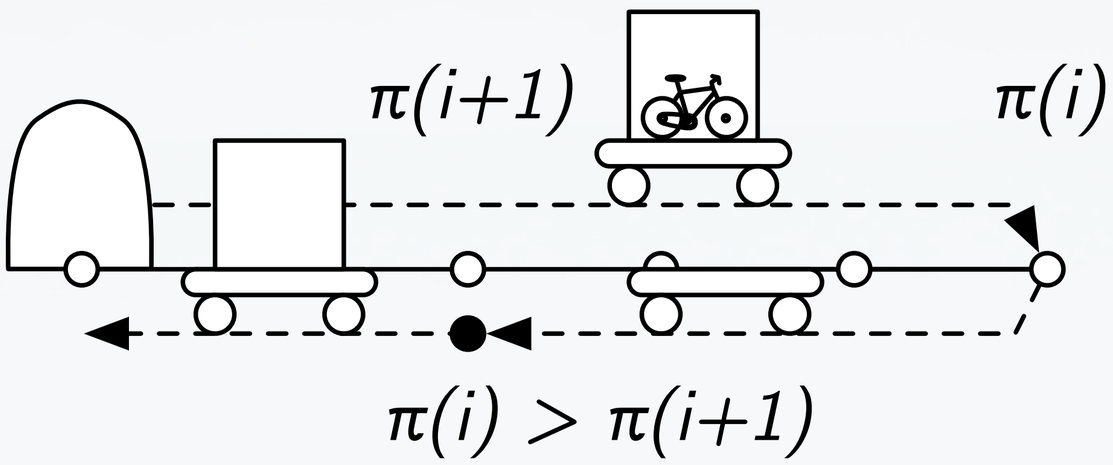
# No Swapping



Between the arrivals of customers *i and i+1*:

- store the full box at slot $\pi(i)$
- fetch the empty box from slot $\pi(i+1)$.




In either case, the robot needs $2 \cdot \max(\pi(i), \pi(i+1))$.

ETH zürich
Daniel Graf

# No Swapping



*Definition:* ***Inter request times***

Let $\Delta = \{\Delta_0, \Delta_1, \ldots, \Delta_n\}$, where $\Delta_i = t^a_{i+1} - t^a_i$.

# No Swapping



*Definition:* ***Inter request times***

Let $\Delta = \{\Delta_0, \Delta_1, \ldots, \Delta_n\}$, where $\Delta_i = t^a_{i+1} - t^a_i$.

*Definition:* **No-Swap Arrival-Only *problem***

Given $\Delta$, find $\pi$ with $2 \cdot \max(\pi(i), \pi(i+1)) \leq \Delta_i$.

ETH zürich
Daniel Graf

# No Swapping



*Definition:* ***Inter request times***

Let $\Delta = \{\Delta_0, \Delta_1, \ldots, \Delta_n\}$, where $\Delta_i = t^a_{i+1} - t^a_i$.

*Definition:* **No-Swap Arrival-Only *problem***

Given $\Delta$, find $\pi$ with $2 \cdot \max(\pi(i), \pi(i+1)) \leq \Delta_i$.

*Equivalently:*

Given $\Delta$, find $\pi$ with $\pi(i) \leq \min\left(\frac{\Delta_{i-1}}{2}, \frac{\Delta_i}{2}\right)$.

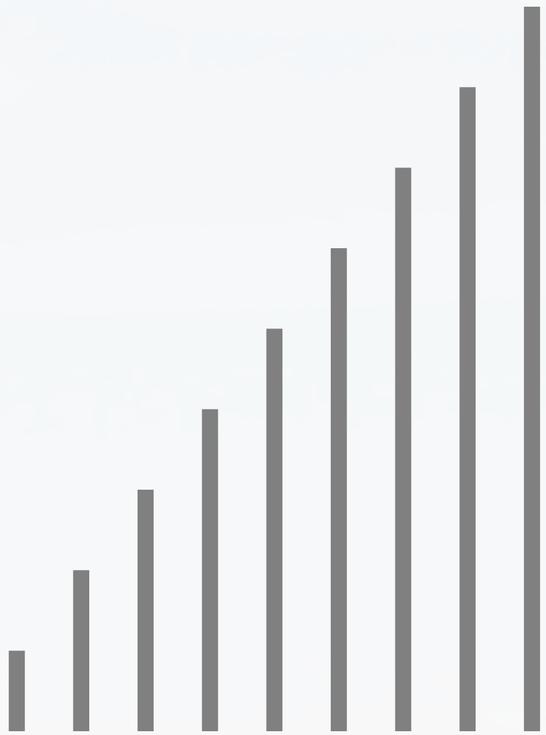
ETH zürich
Daniel Graf

# Constrained π

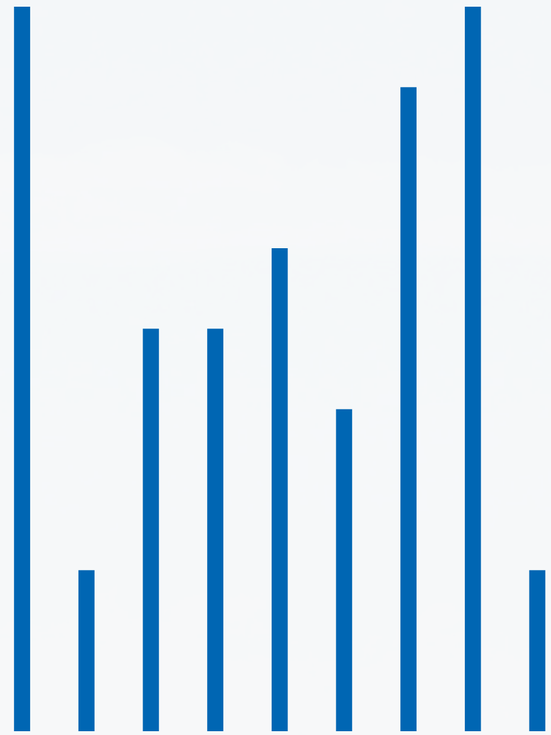


*Definition:* **Constrained Permutation *problem***

Given $X = \{x_1, \ldots, x_n\}$, find $\pi \in S_n$ with $\pi(i) \leq x_i$.



$\pi$



$X$

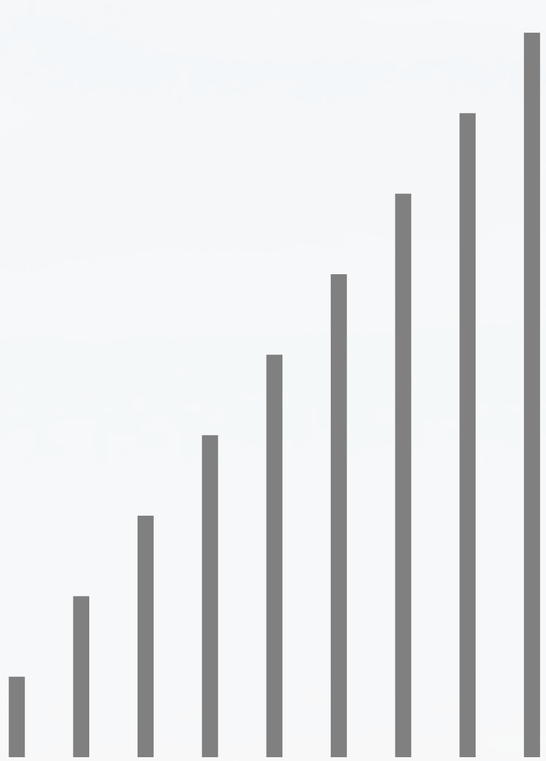
ETH zürich
Daniel Graf

# Constrained π

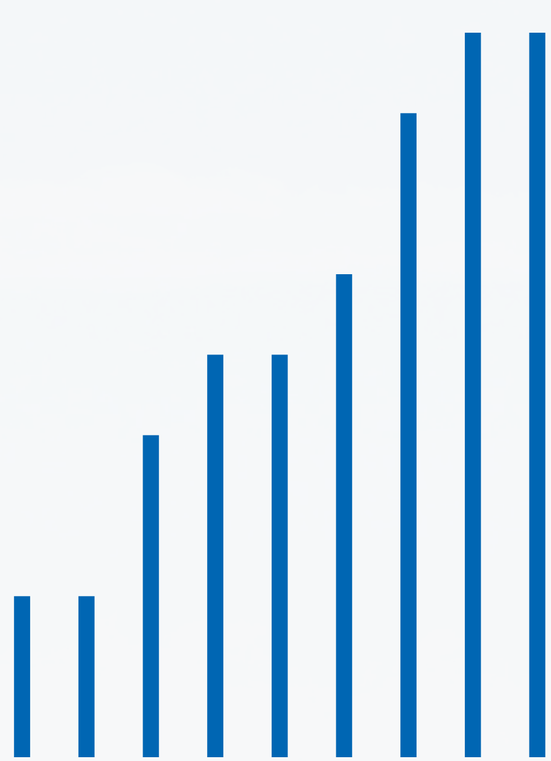


*Definition:* **Constrained Permutation *problem***

Given $X = \{x_1, \ldots, x_n\}$, find $\pi \in S_n$ with $\pi(i) \leq x_i$.



$\pi$



$X$

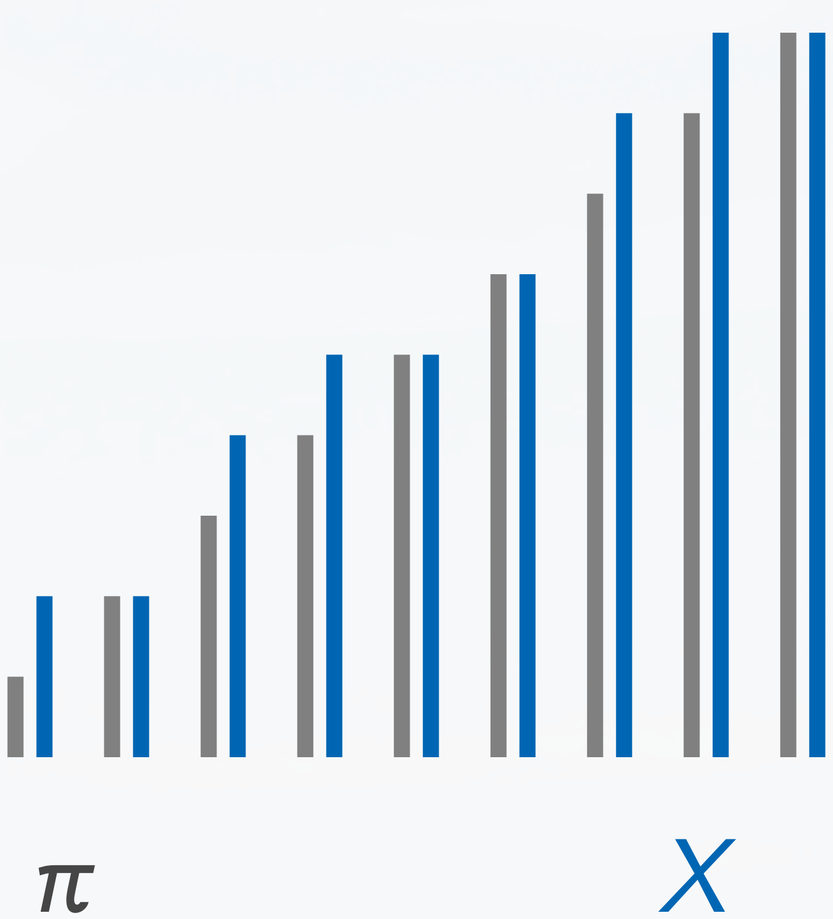
ETH zürich
Daniel Graf

# Constrained π



*Definition:* **Constrained Permutation *problem***

Given $X = \{x_1, \ldots, x_n\}$, find $\pi \in S_n$ with $\pi(i) \leq x_i$.



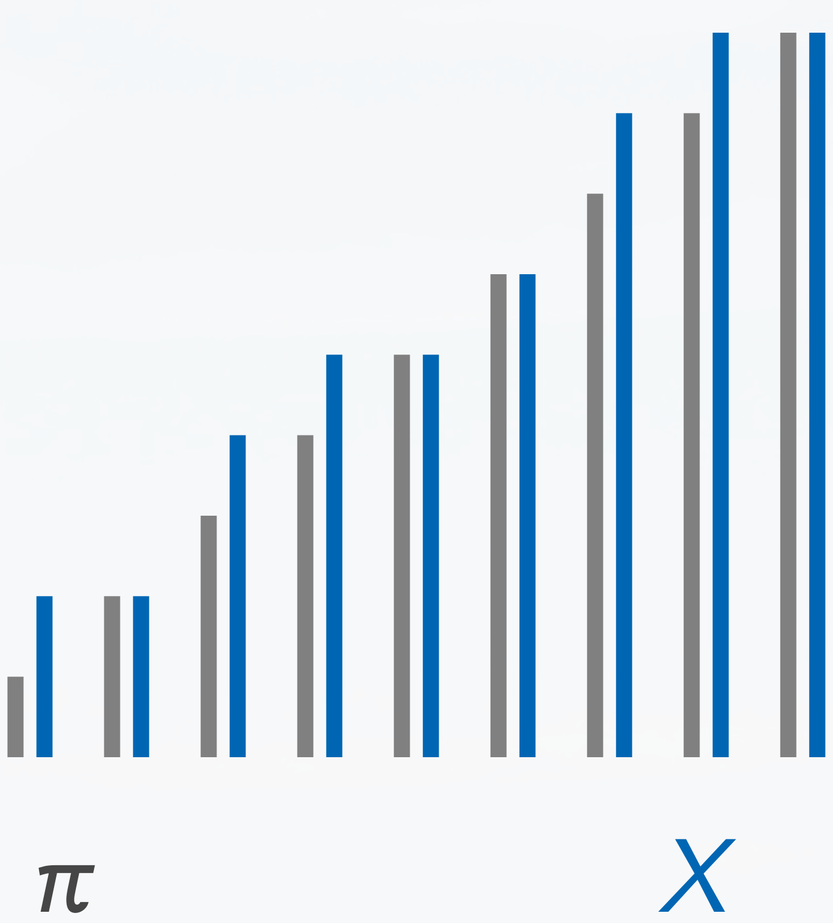
ETH zürich
Daniel Graf

# Constrained π



*Definition:* **Constrained Permutation *problem***

Given $X = \{x_1, \ldots, x_n\}$, find $\pi \in S_n$ with $\pi(i) \leq x_i$.




*Solution:*

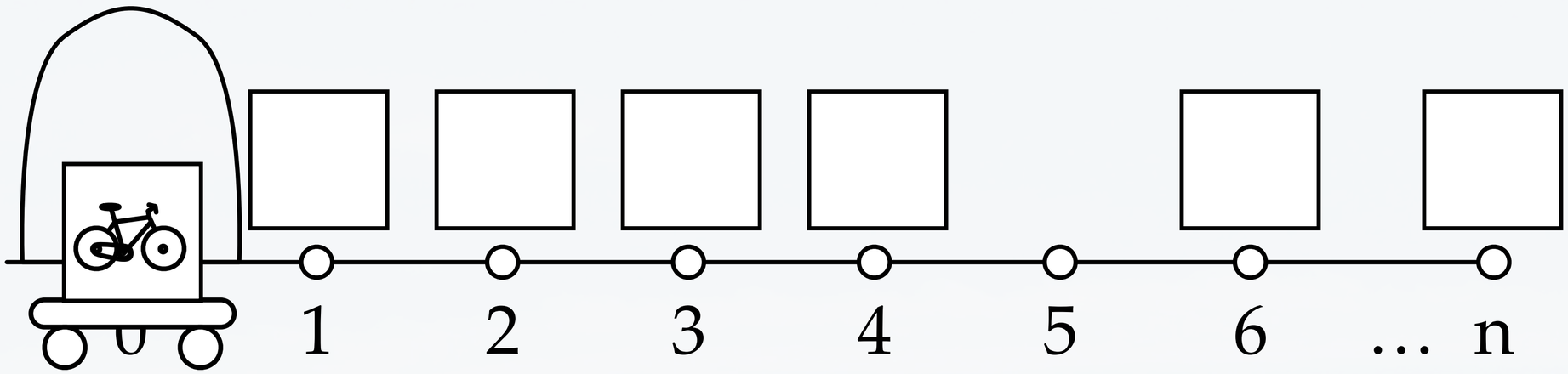
Sort-and-compare in linear time using bucket sort.

ETH zürich
Daniel Graf

# Return to Door



*Side note:* What if we let the robot always return to the door in order to decouple the requests?

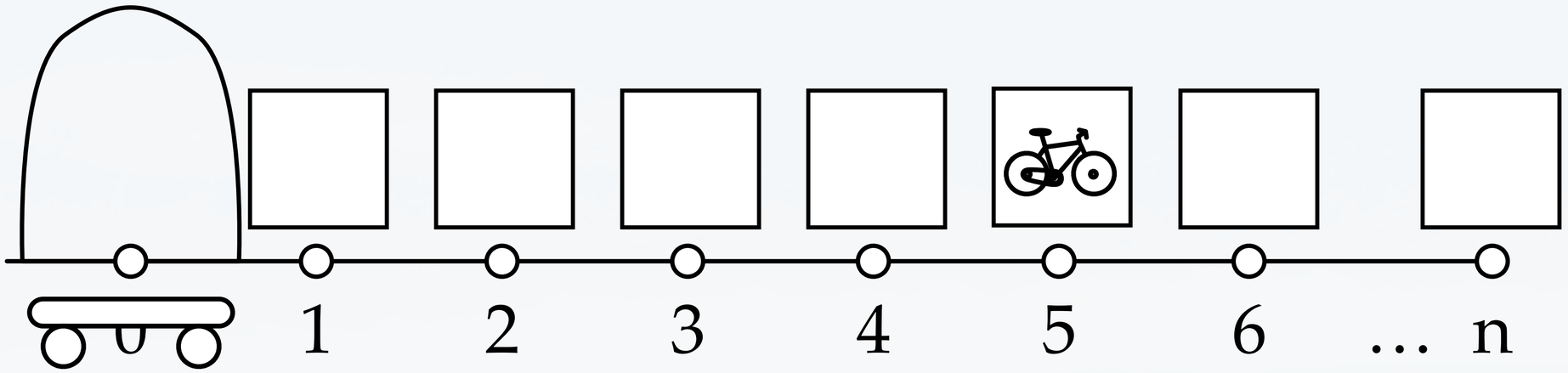


ETH zürich
Daniel Graf

# Return to Door



*Side note:* What if we let the robot always return to the door in order to decouple the requests?

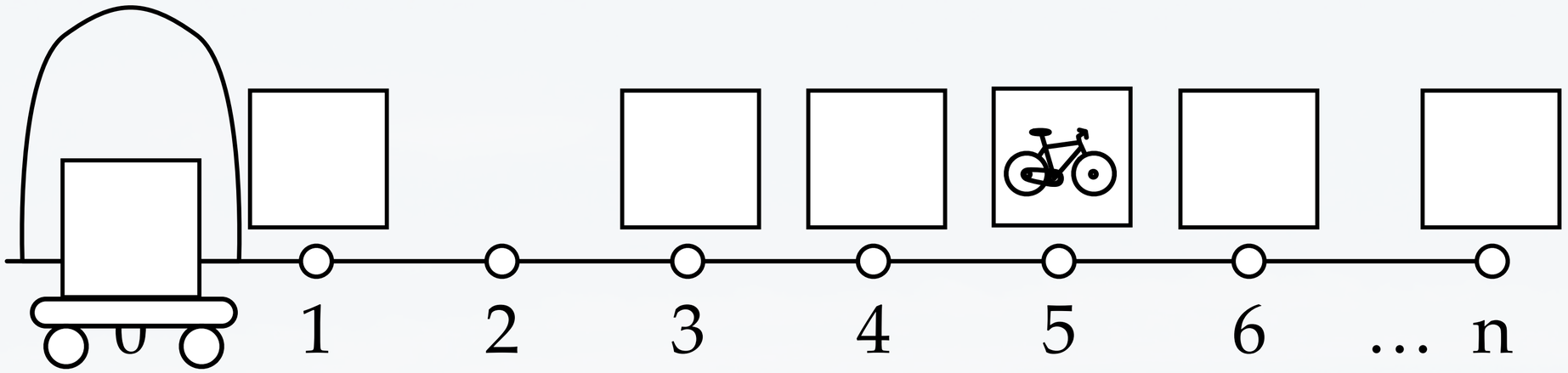


ETH zürich
Daniel Graf

# Return to Door



*Side note:* What if we let the robot always return to the door in order to decouple the requests?

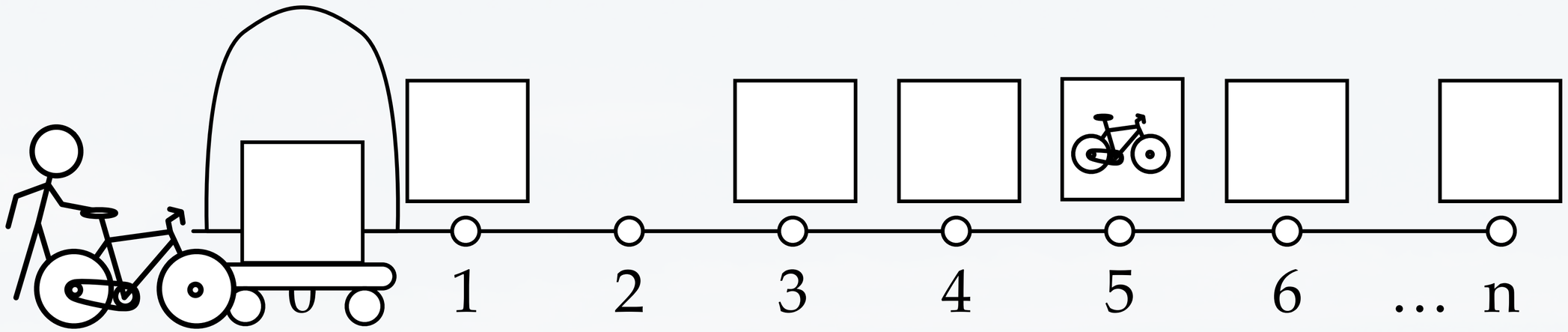


ETH zürich
Daniel Graf

# Return to Door



*Side note:* What if we let the robot always return to the door in order to decouple the requests?

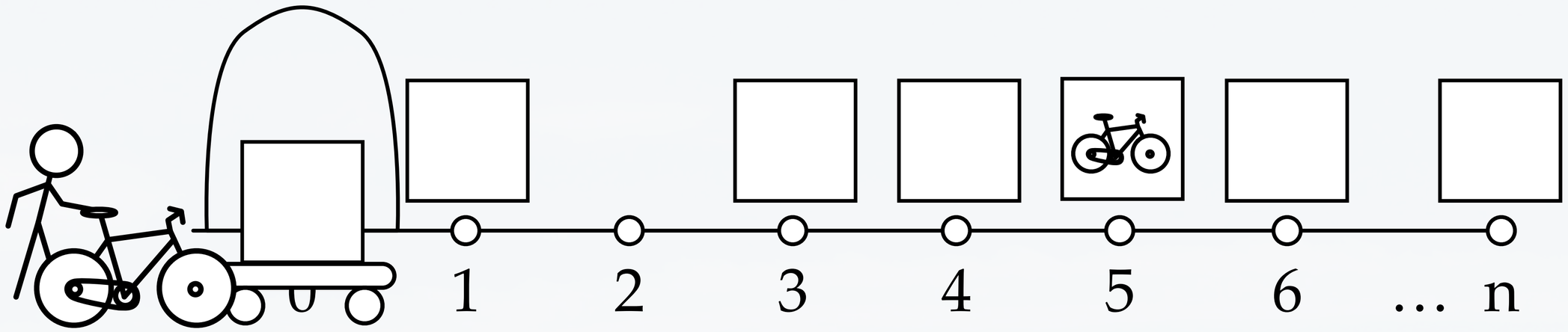


ETH zürich
Daniel Graf

# Return to Door



*Side note:* What if we let the robot always return to the door in order to decouple the requests?




*Definition:* **Return Arrival-Only *problem***

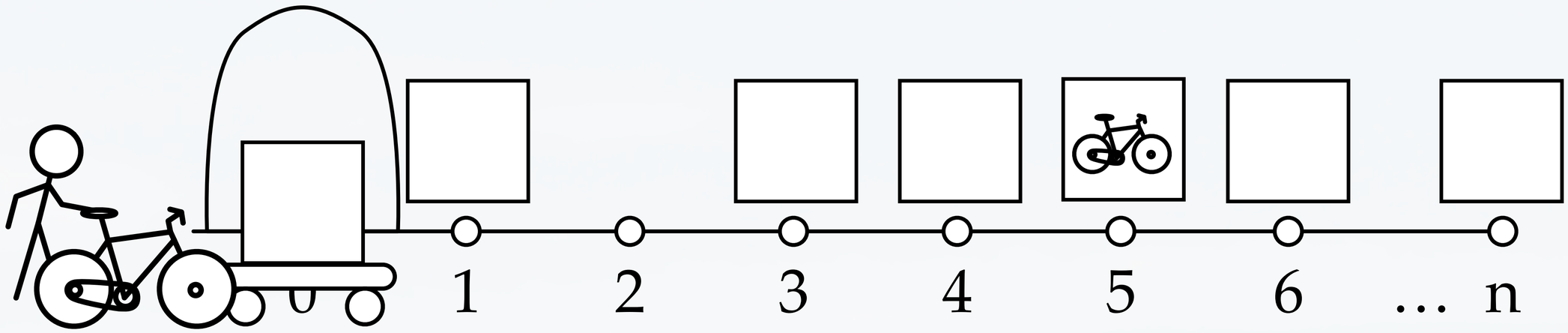
Given $\Delta$, find $\pi$ with $2\pi(i) + 2\pi(i+1) \leq \Delta_i$.

ETH zürich
Daniel Graf

# Return to Door



*Side note:* What if we let the robot always return to the door in order to decouple the requests?




*Definition:* **Return Arrival-Only *problem***
Given $\Delta$, find $\pi$ with $2\pi(i) + 2\pi(i+1) \leq \Delta_i$.

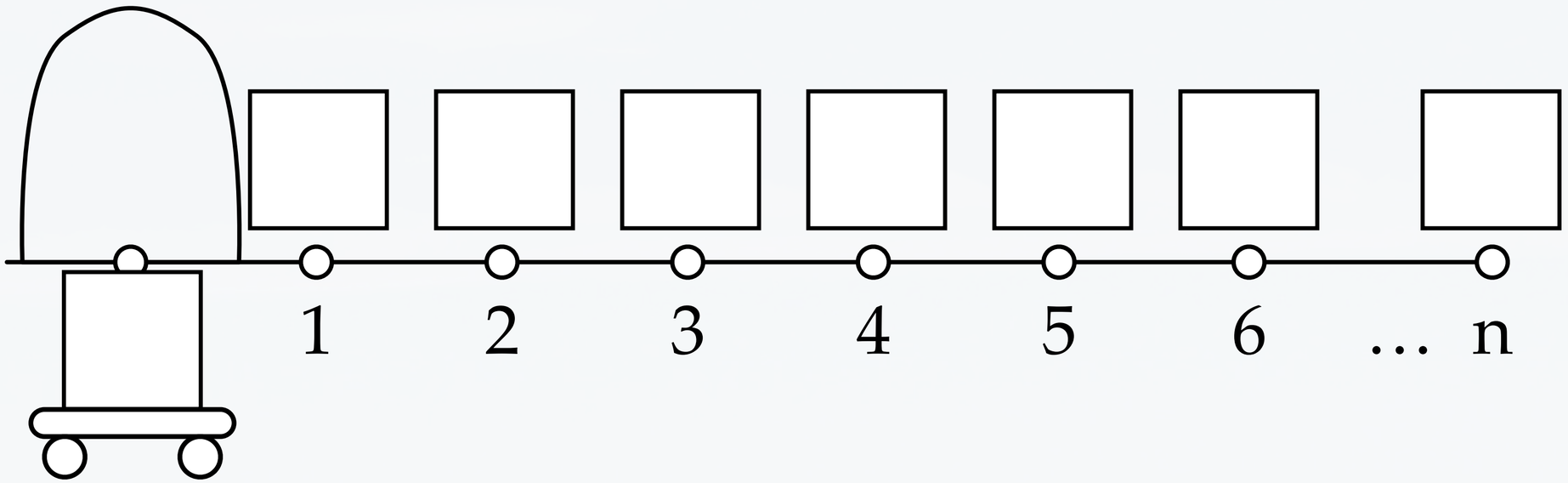
[2015, Couëtoux, Labourel] Strongly *NP*-complete.

ETH zürich
Daniel Graf

# Box-Swapping



Allow swaps and start with a box on the robot.

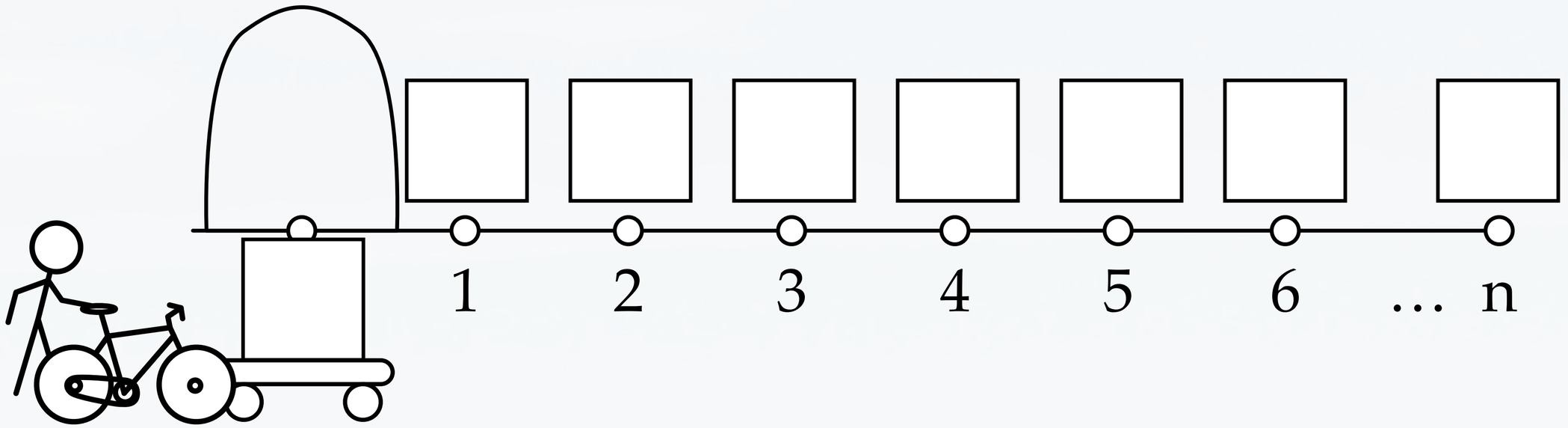


ETH zürich
Daniel Graf

# Box-Swapping



Allow swaps and start with a box on the robot.

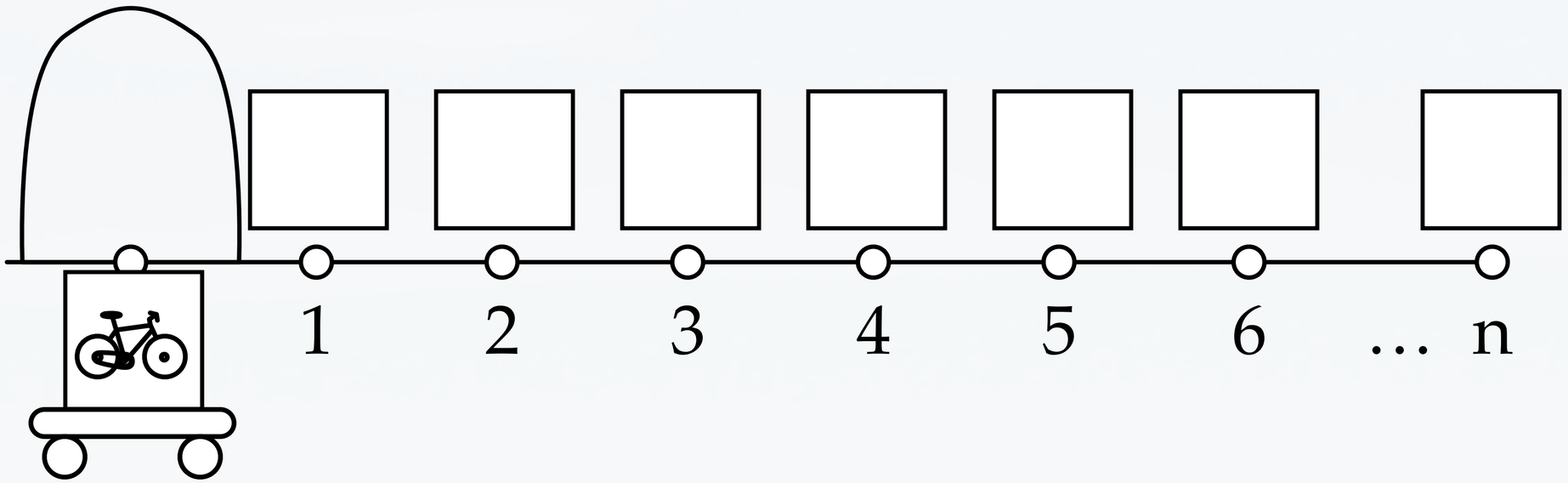


ETH zürich
Daniel Graf

# Box-Swapping



Allow swaps and start with a box on the robot.

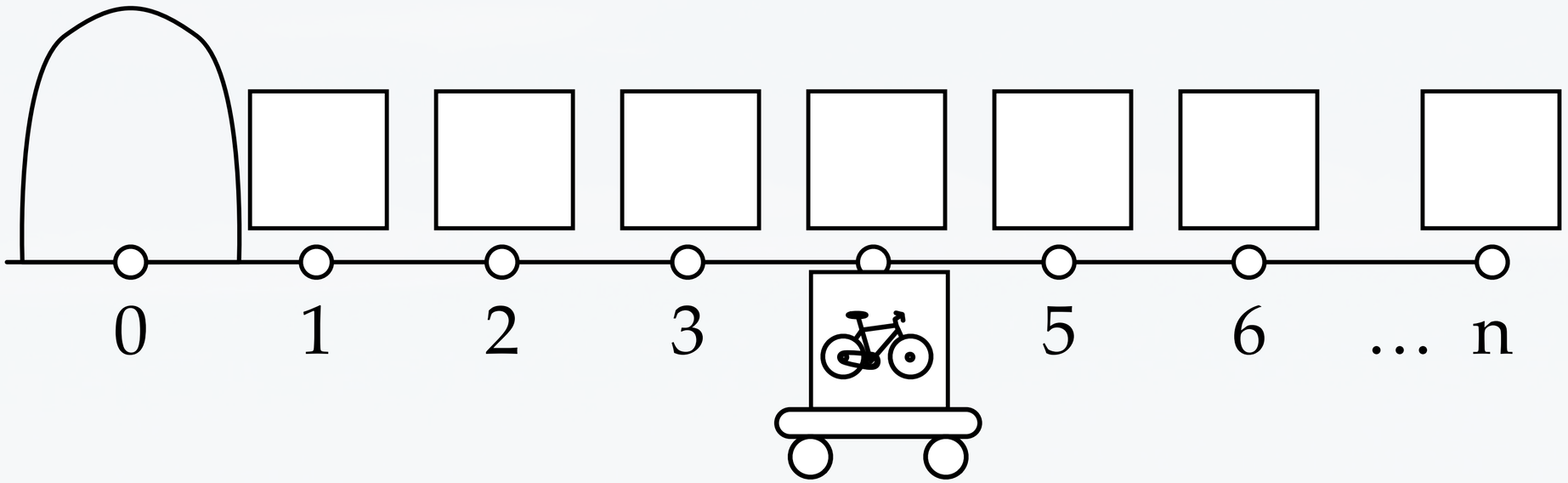


ETH zürich
Daniel Graf

# Box-Swapping



Allow swaps and start with a box on the robot.

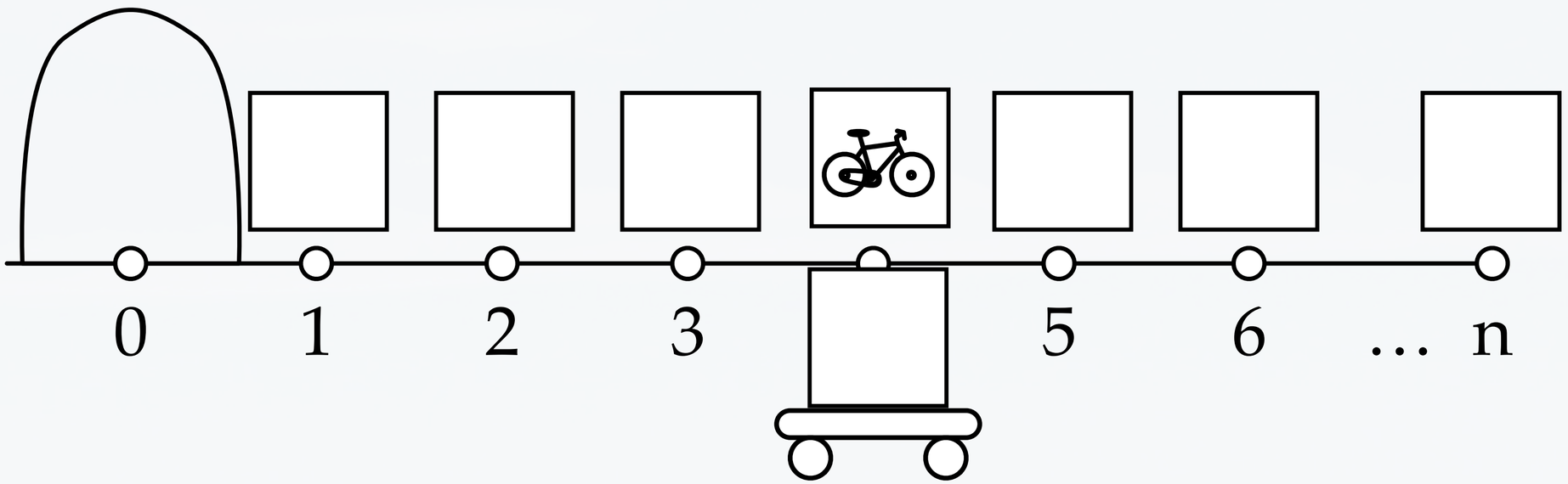


ETH zürich
Daniel Graf

# Box-Swapping



Allow swaps and start with a box on the robot.

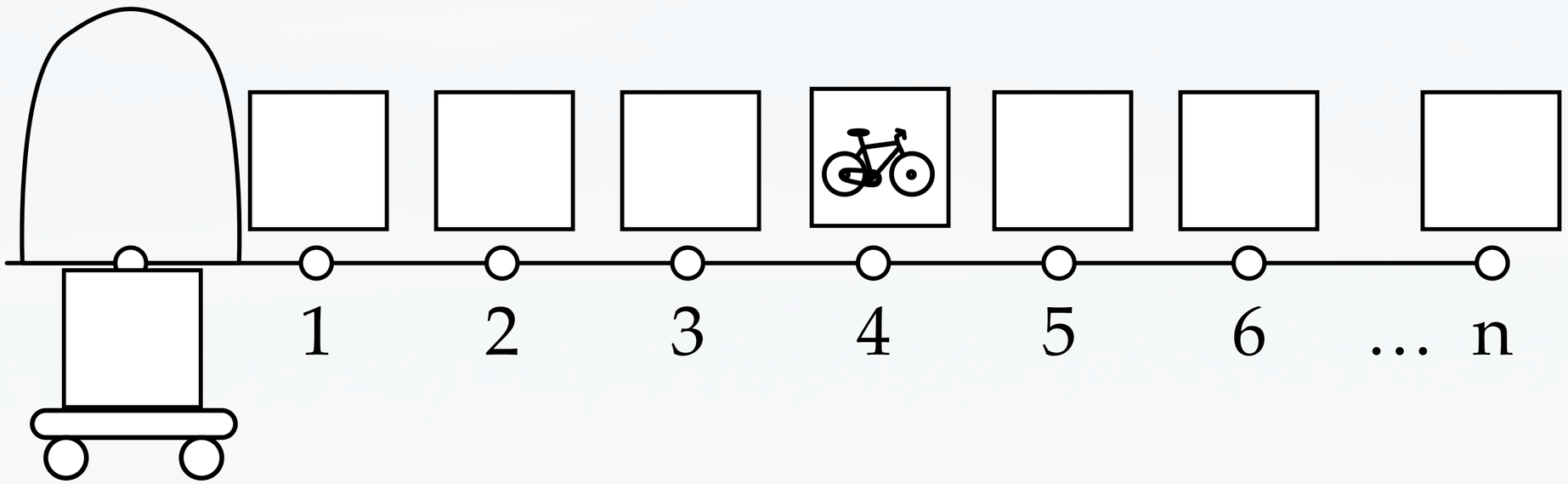


ETH zürich
Daniel Graf

# Box-Swapping



Allow swaps and start with a box on the robot.

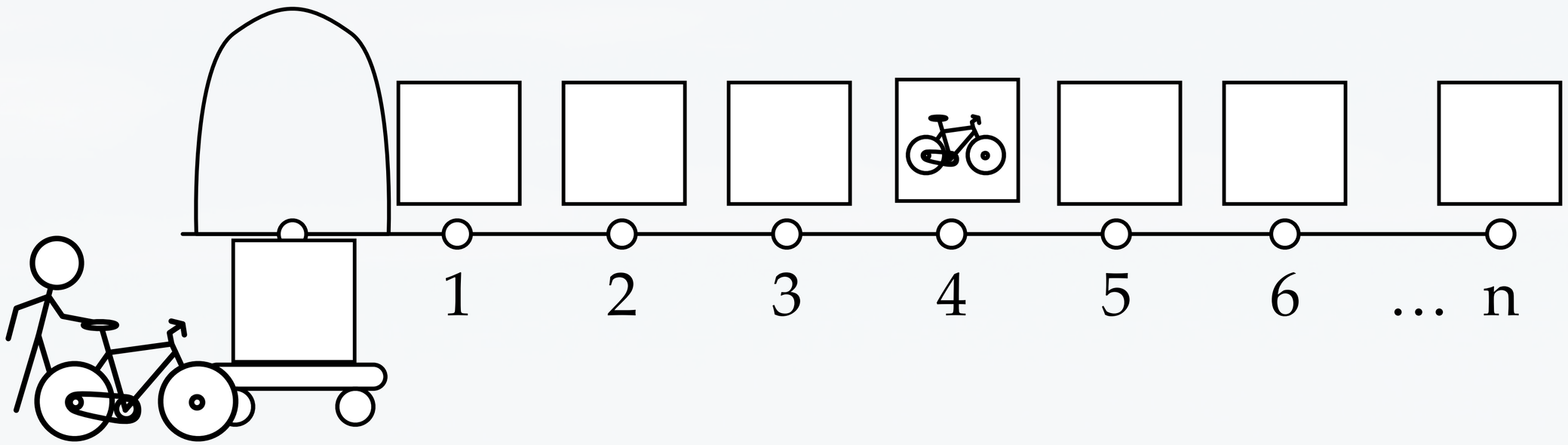


ETH zürich
Daniel Graf

# Box-Swapping



Allow swaps and start with a box on the robot.

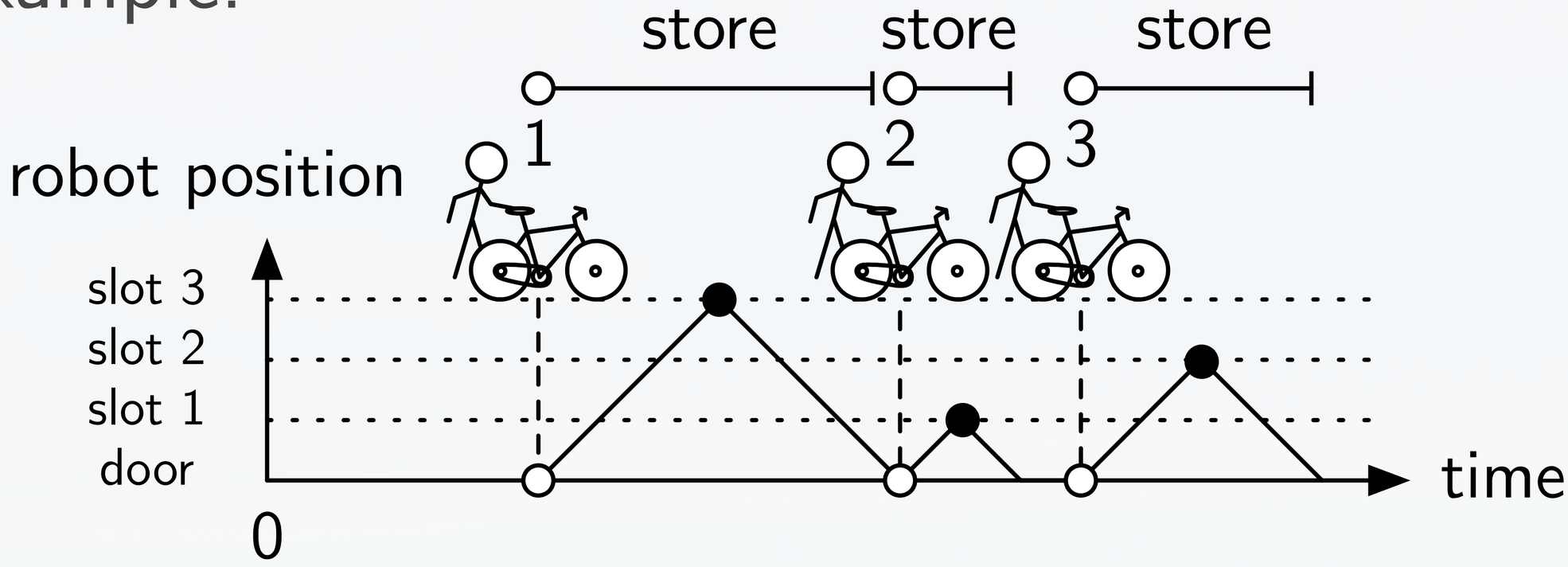


ETH zürich
Daniel Graf

# Box-Swapping



Example:

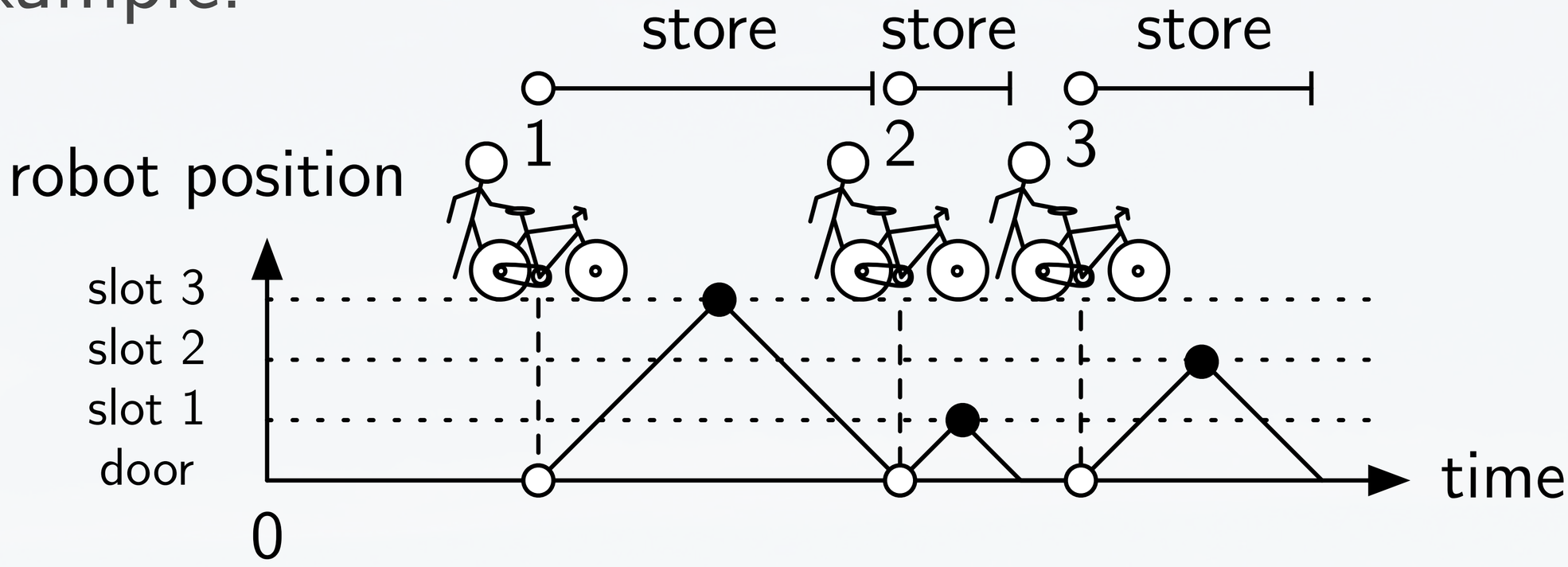


ETH zürich
Daniel Graf

# Box-Swapping



Example:




*Definition:* ***Swap Arrival-Only problem***

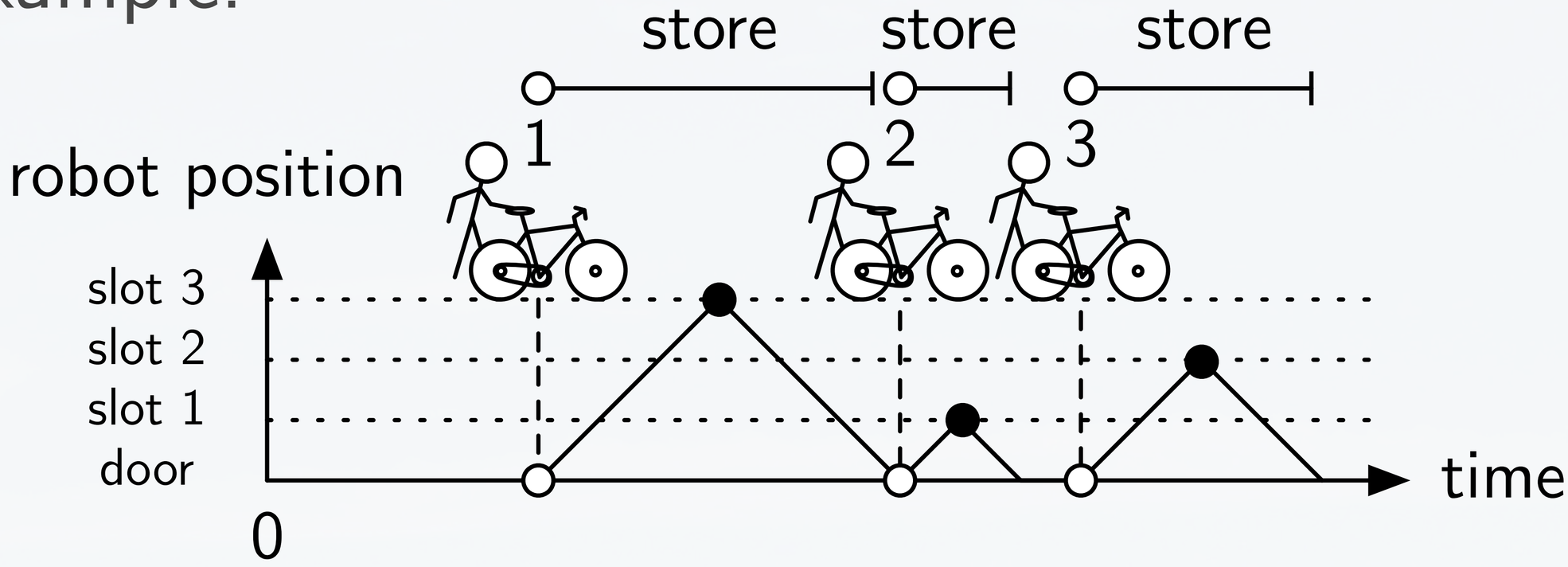
Given $\Delta$, find $\pi$ with $2\pi(i) \leq \Delta_i$.

ETH zürich
Daniel Graf

# Box-Swapping



Example:




*Definition:* ***Swap Arrival-Only problem***

Given $\Delta$, find $\pi$ with $2\pi(i) \leq \Delta_i$.

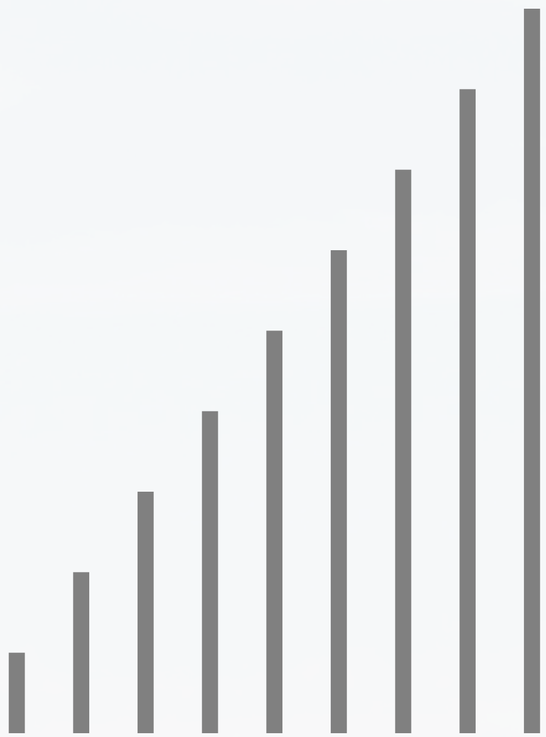
$\rightarrow$ *decide with sort-and-compare*

# Constrained π

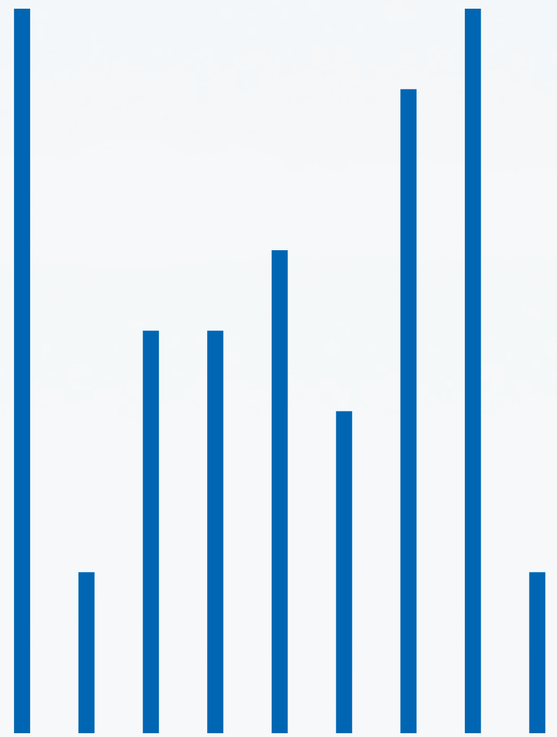


*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.



$\pi$



$X$

ETH zürich
Daniel Graf

# Constrained π



*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.

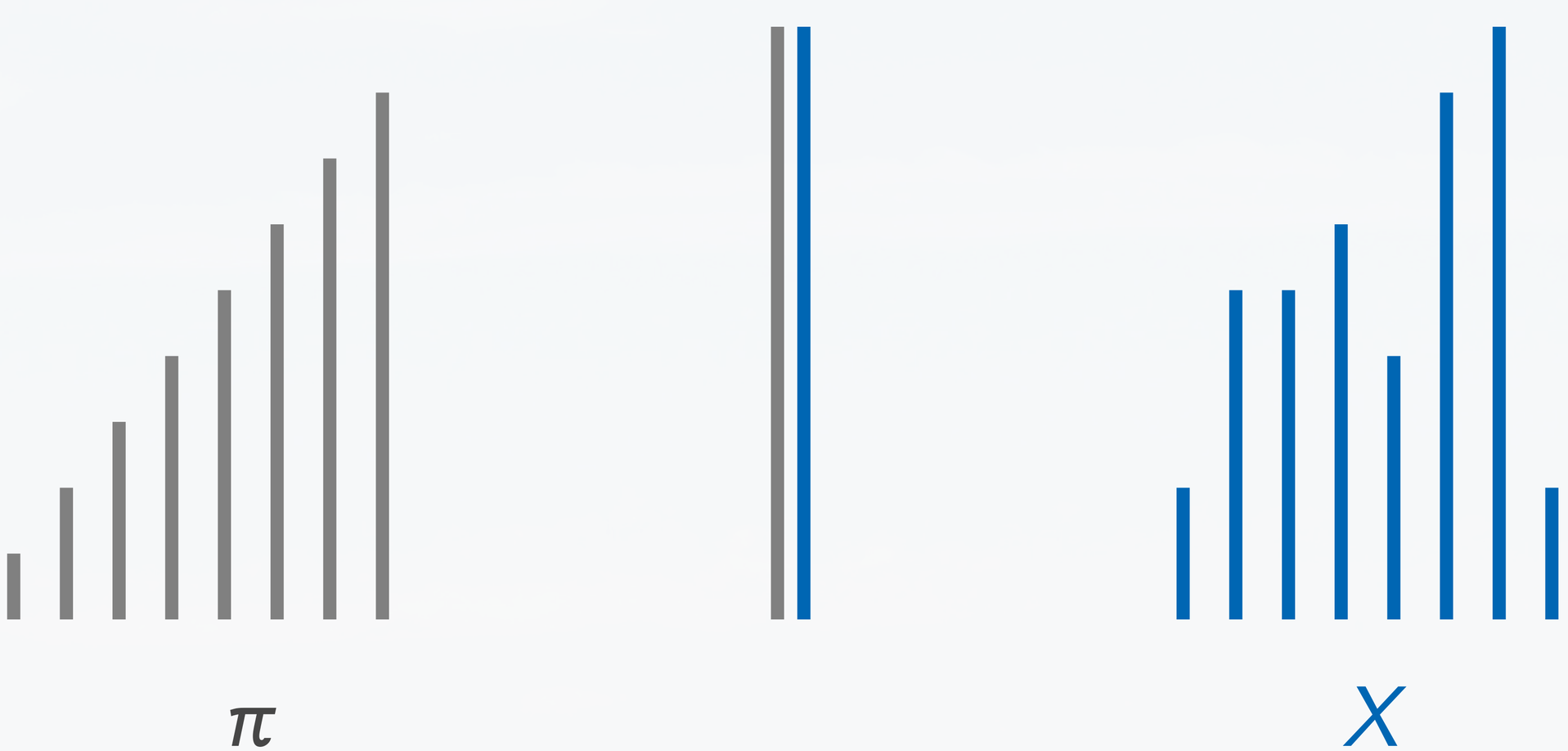
ETH zürich
Daniel Graf

# Constrained π



*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.



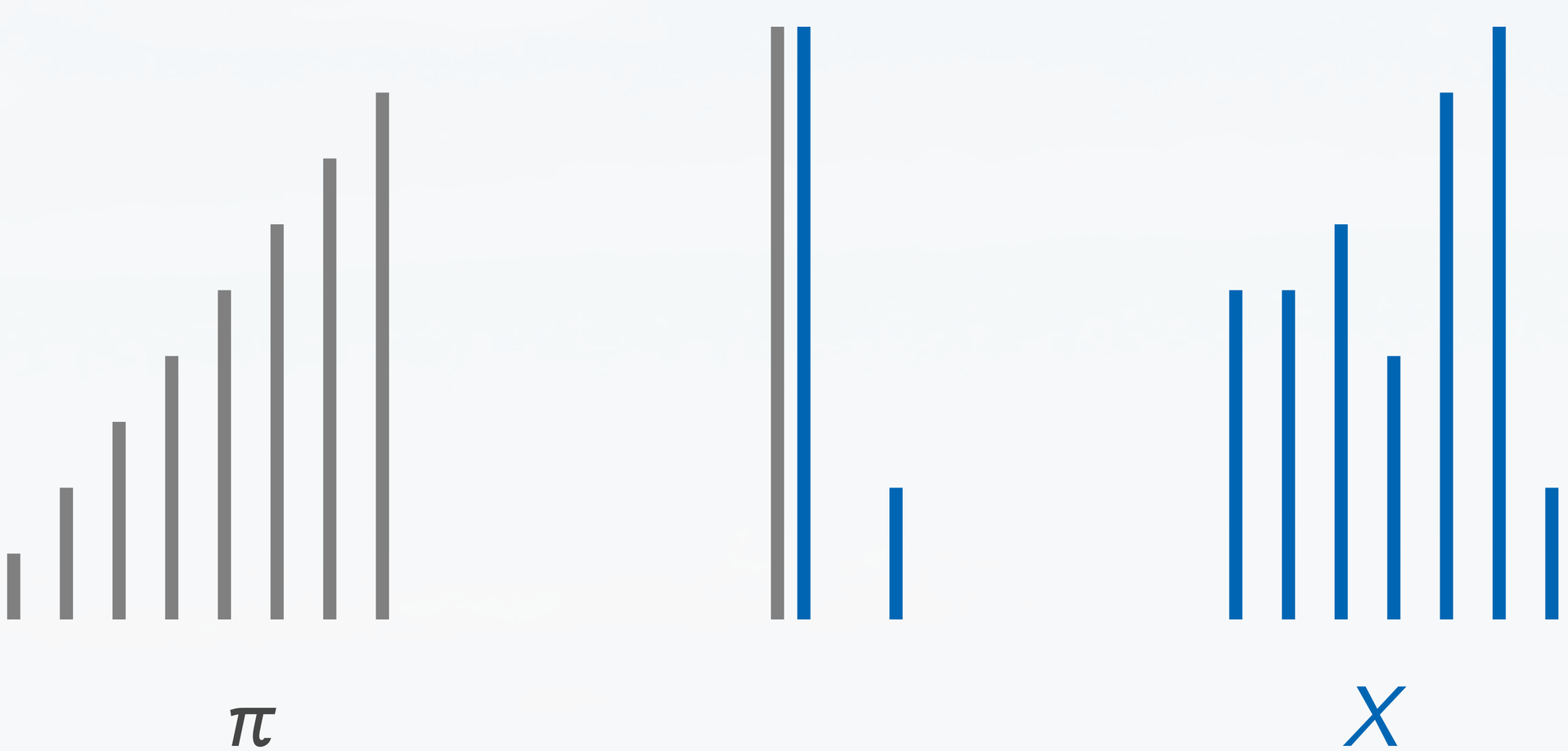
ETH zürich
Daniel Graf

# Constrained π



*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.



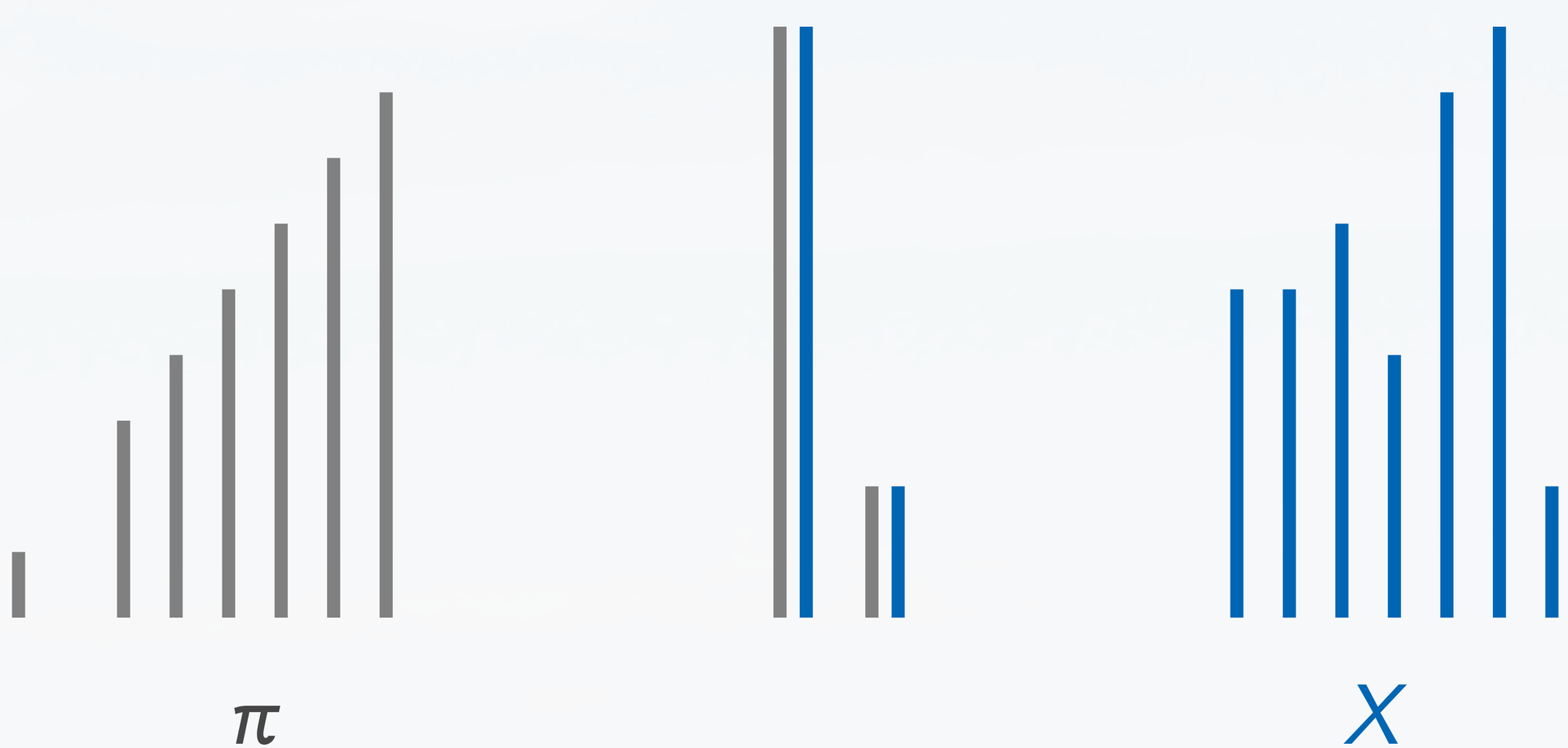
ETH zürich
Daniel Graf

# Constrained π



*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.



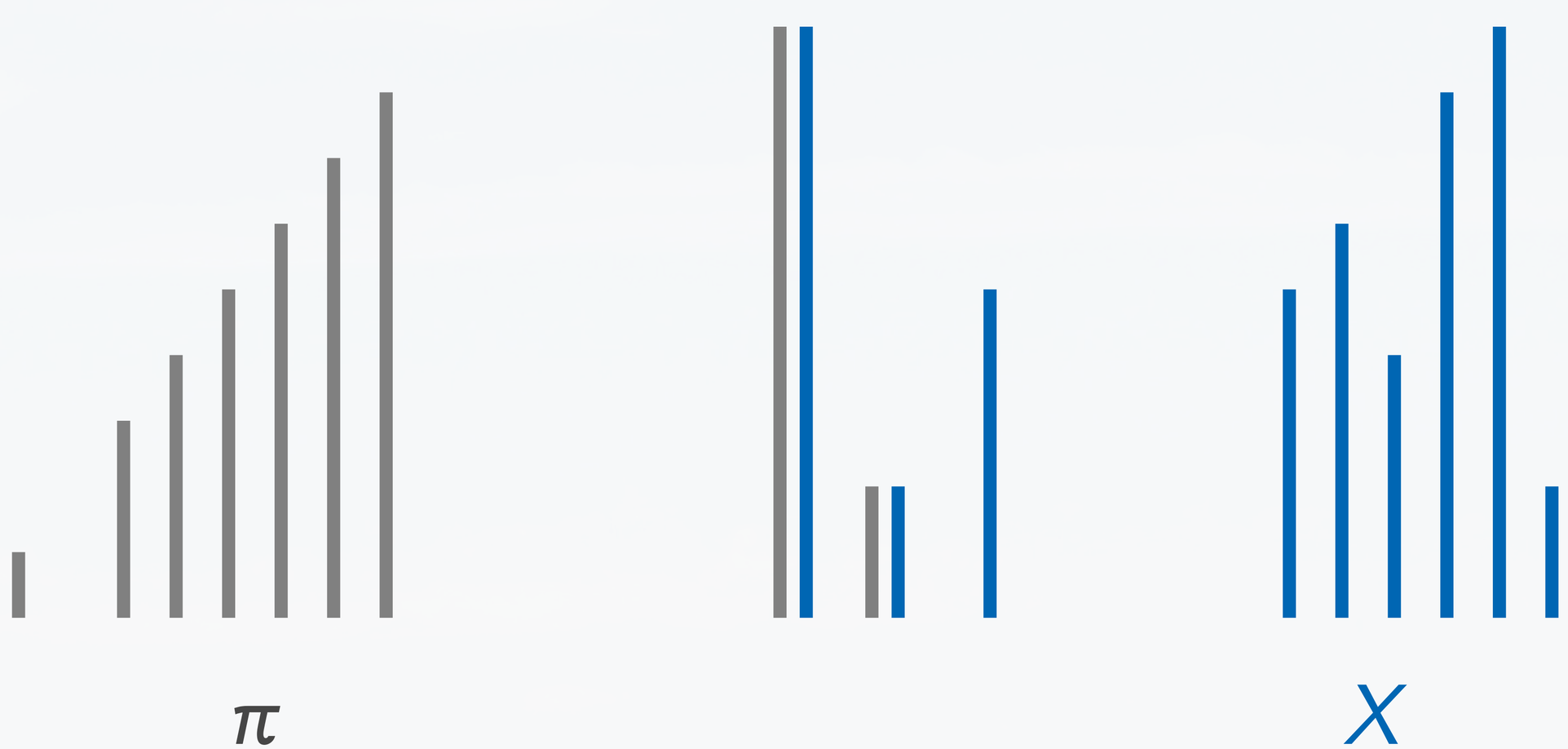
ETH zürich
Daniel Graf

# Constrained π



*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.



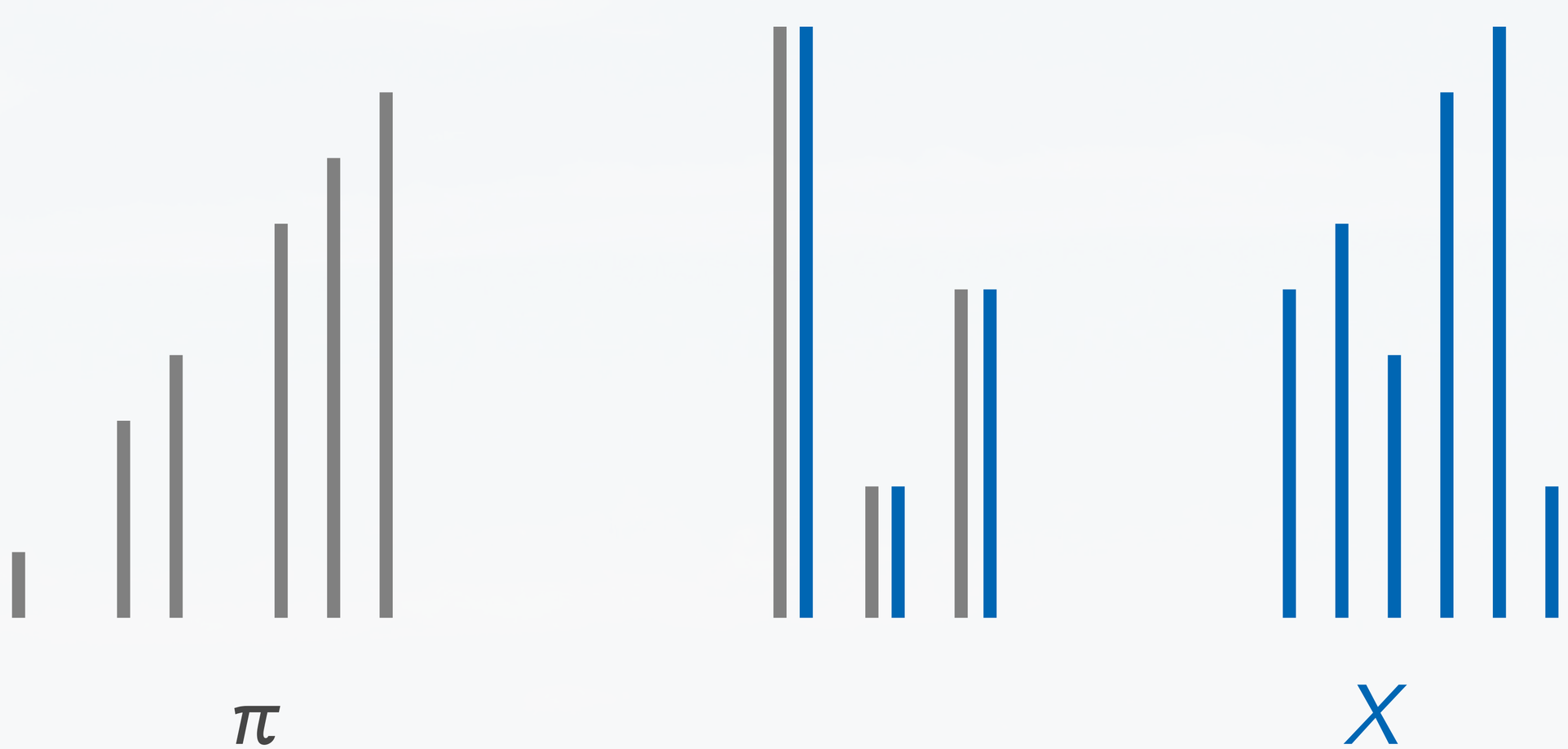
ETH zürich
Daniel Graf

# Constrained π



*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.



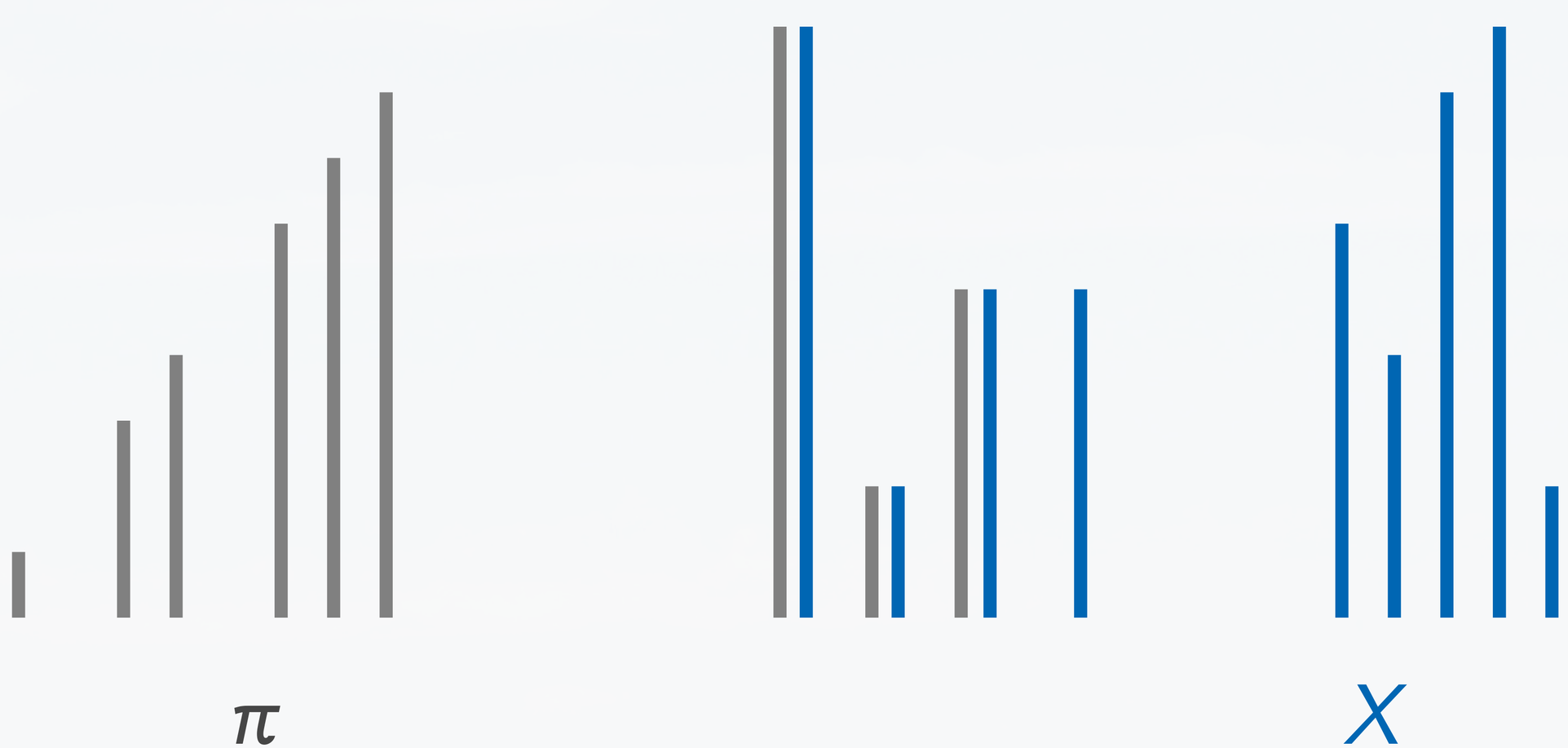
ETH zürich
Daniel Graf

# Constrained π



*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.



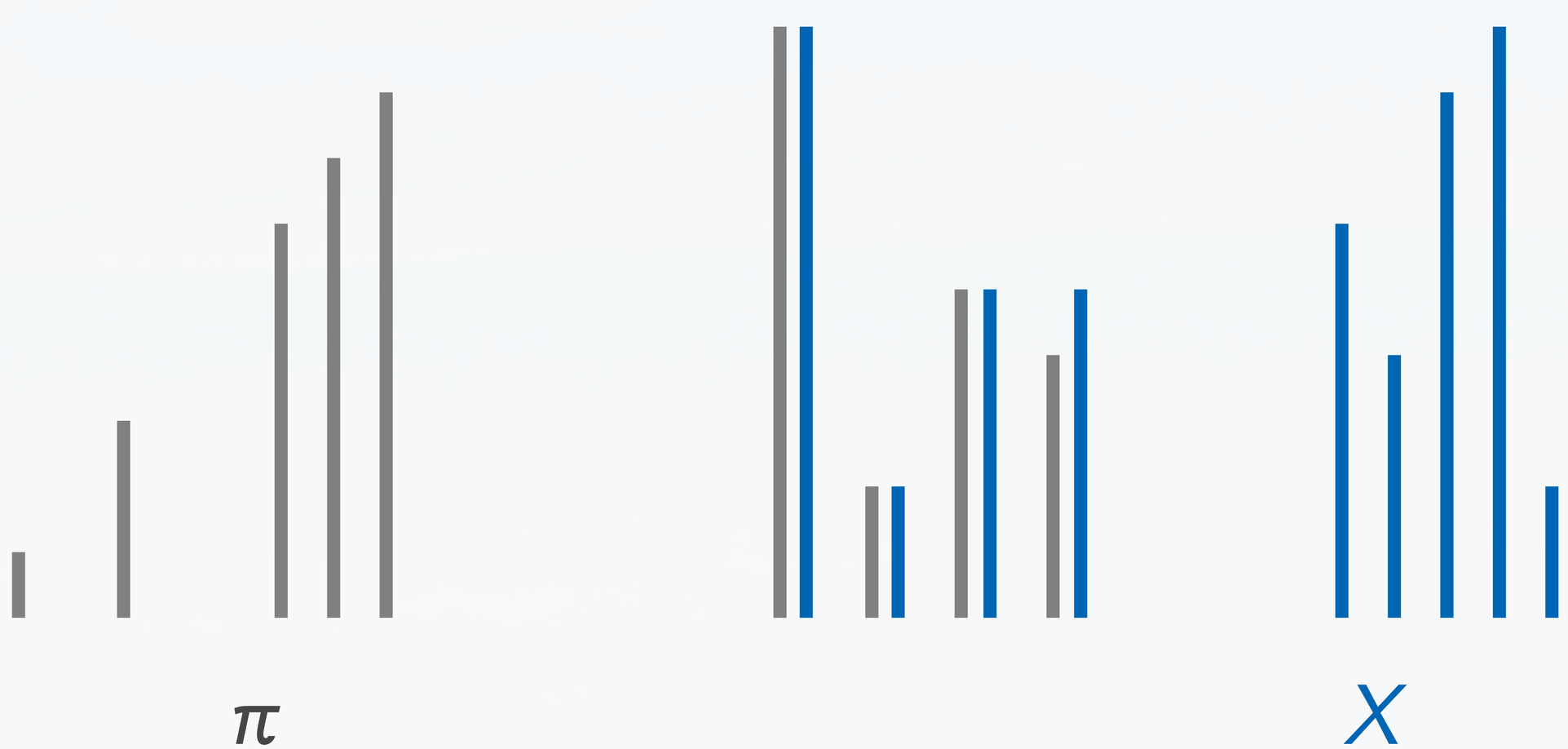
ETH zürich
Daniel Graf

# Constrained π



*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.



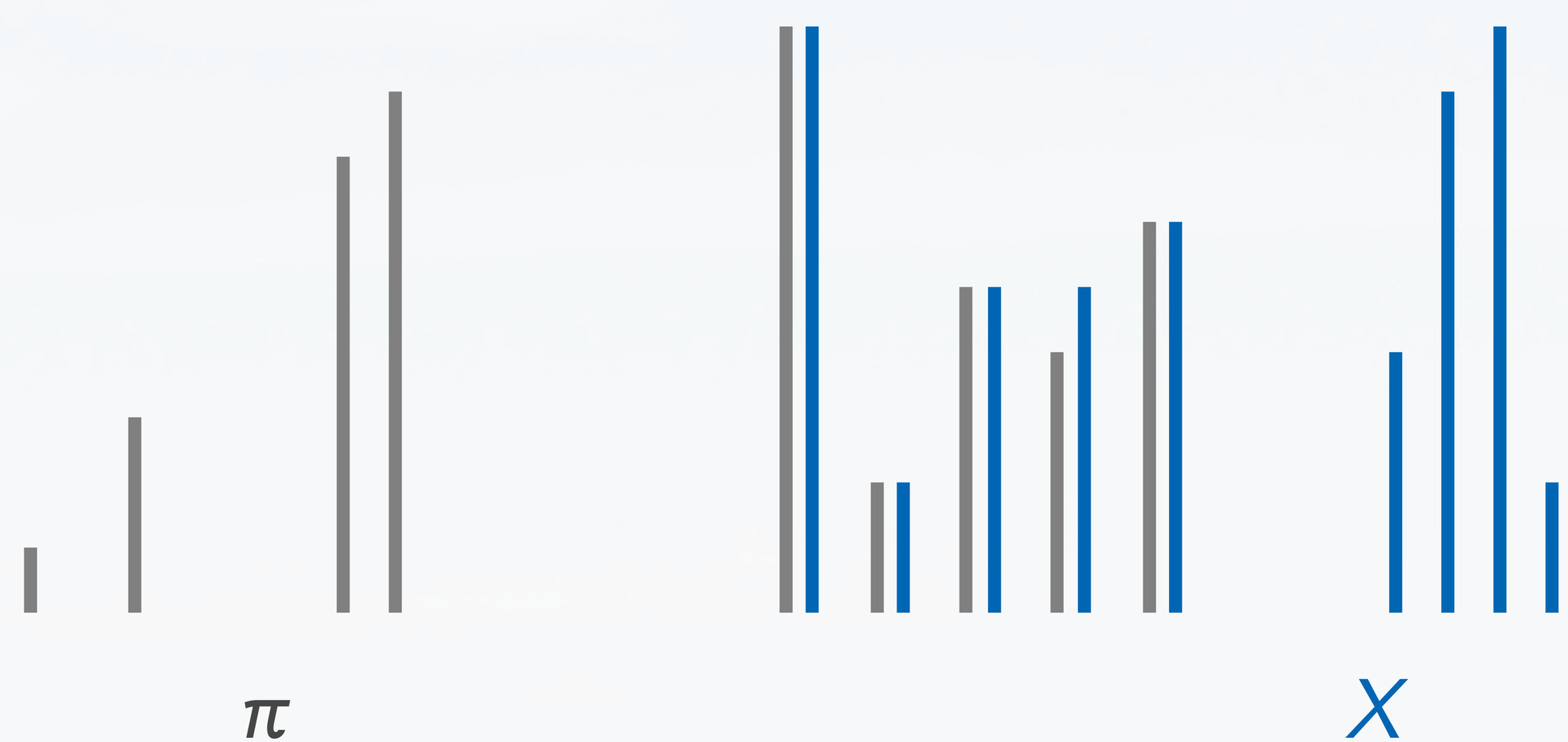
ETH zürich
Daniel Graf

# Constrained π



*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.



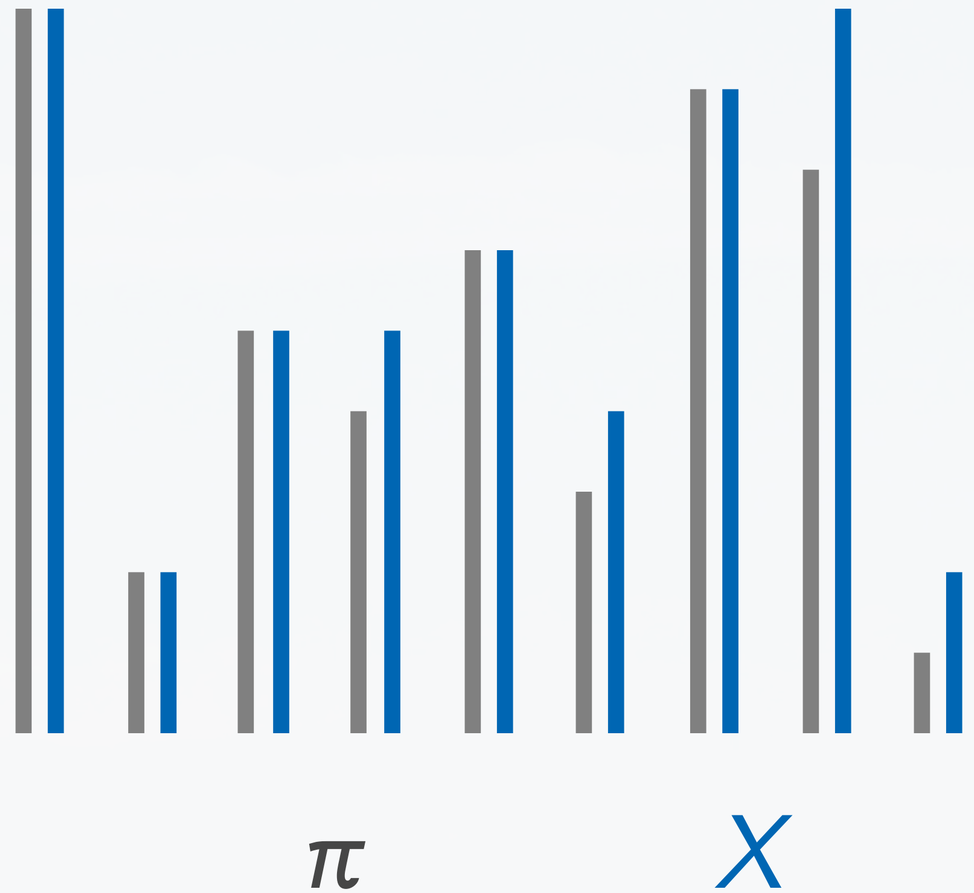
ETH zürich
Daniel Graf

# Constrained π



*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.



ETH zürich
Daniel Graf

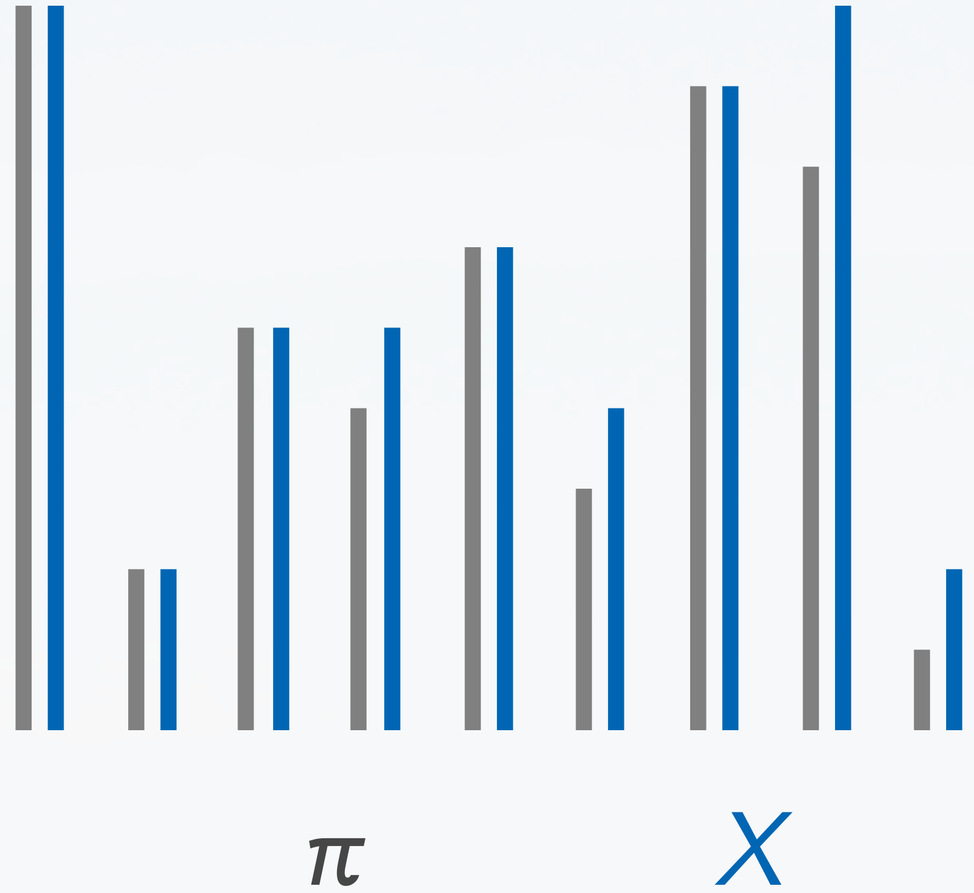
# Constrained π



*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.

Implementation using union find in $\mathcal{O}(n \cdot \alpha(n))$.



ETH zürich
Daniel Graf

# Constrained π



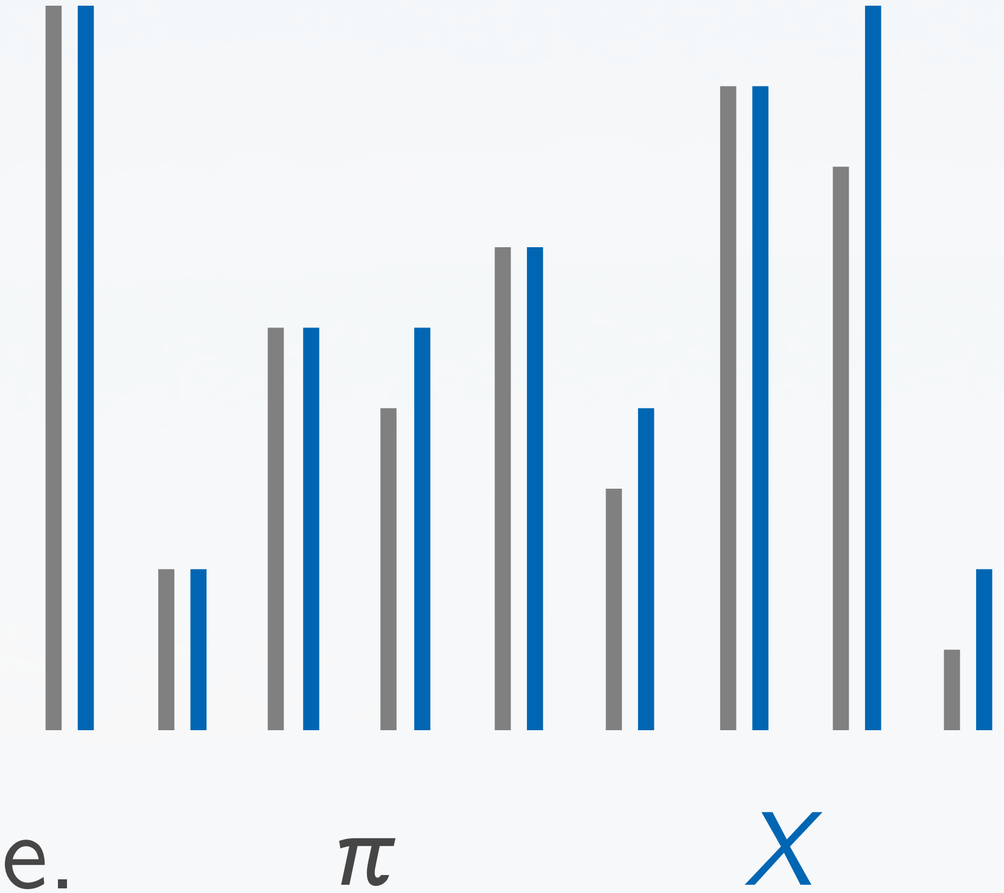
*Alternative approach:* Greedy assignment

Go through the constraints in order and use the largest possible, unassigned permutation entry.

Implementation using union find in $\mathcal{O}(n \cdot \alpha(n))$.

*Main advantage:*

Also works if the constraints arrive online.

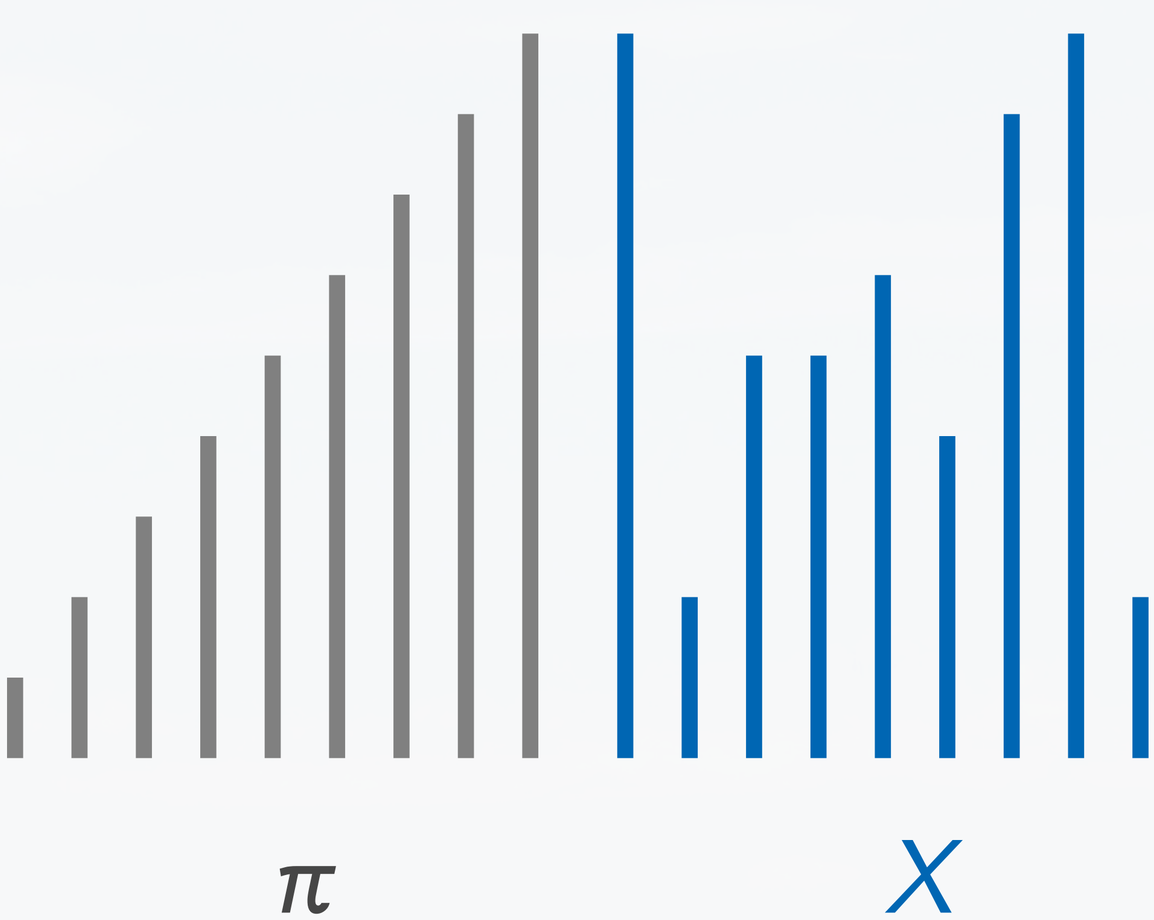


ETH zürich
Daniel Graf

# Online Setting



How much information about the future is needed to fill the slots optimally?

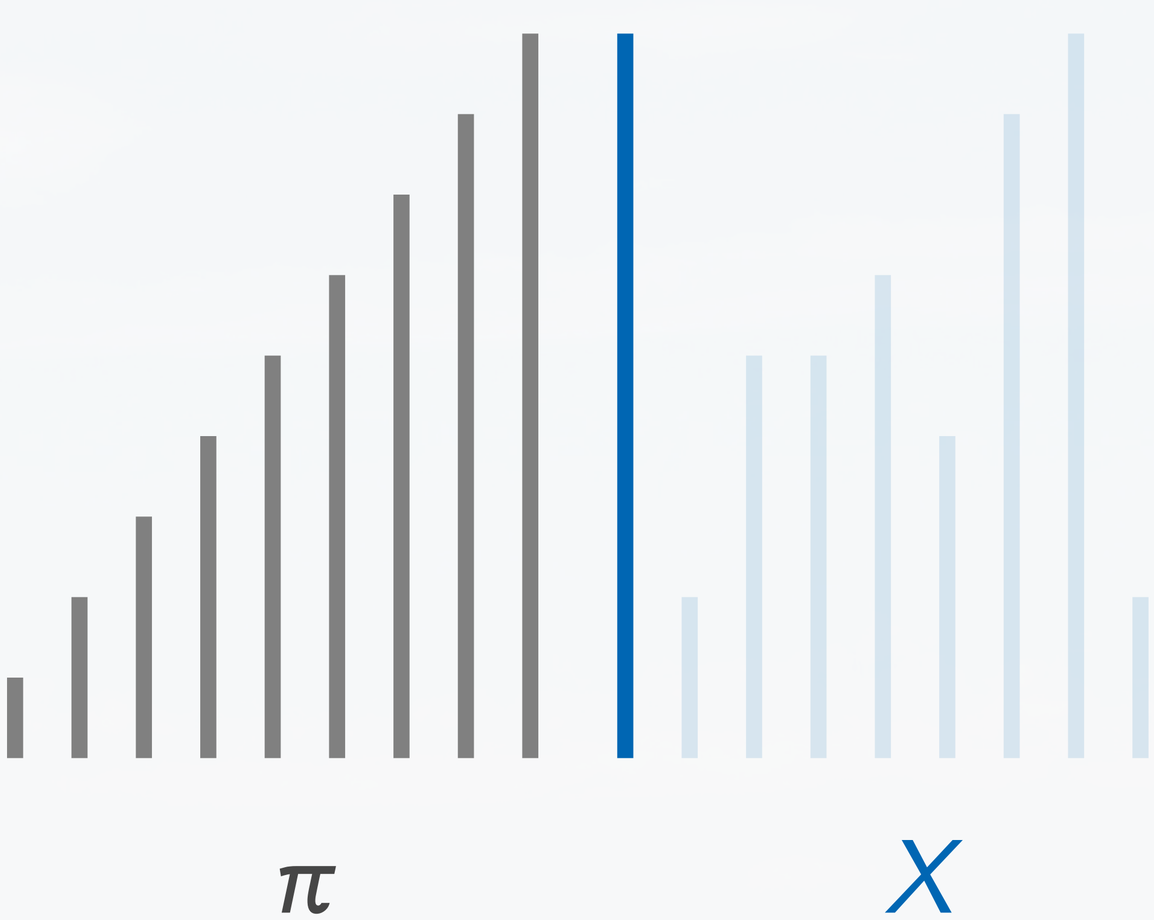


ETH zürich
Daniel Graf

# Online Setting



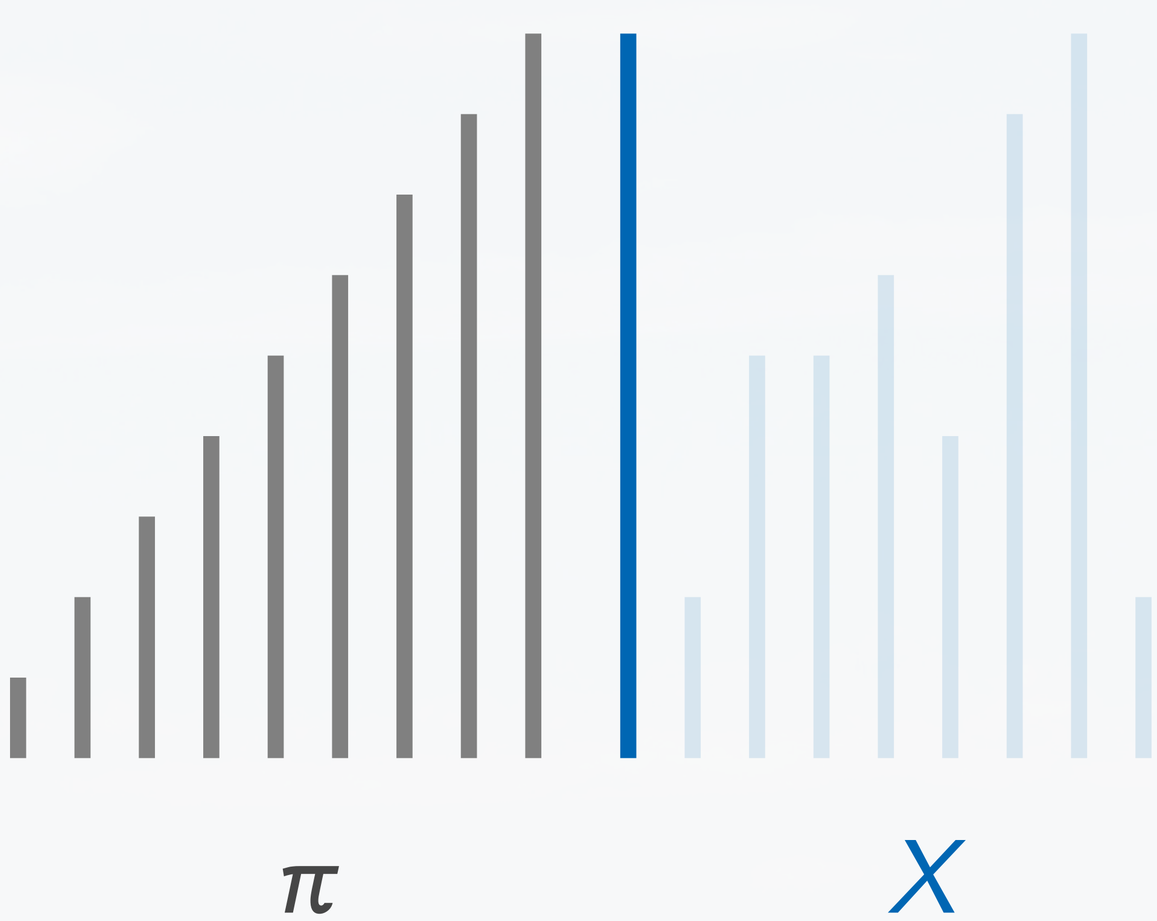
How much information about the future is needed to fill the slots optimally?



# Online Setting



How much information about the future is needed to fill the slots optimally?




In our setting:

*No-swap model:*

$\rightarrow$ next 2 arrivals

*Swap model:*

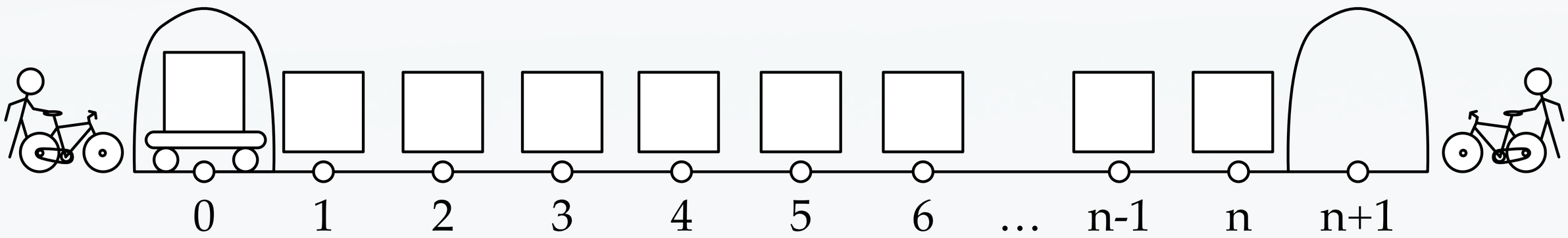
$\rightarrow$ only next arrival

# Two Doors



New setting:

- second door at the other end
- swap-robot, carrying a box initially
- still arrival-only and no rearrangement




Who picks the door for each customer?

ETH zürich
Daniel Graf

# Fixed Doors

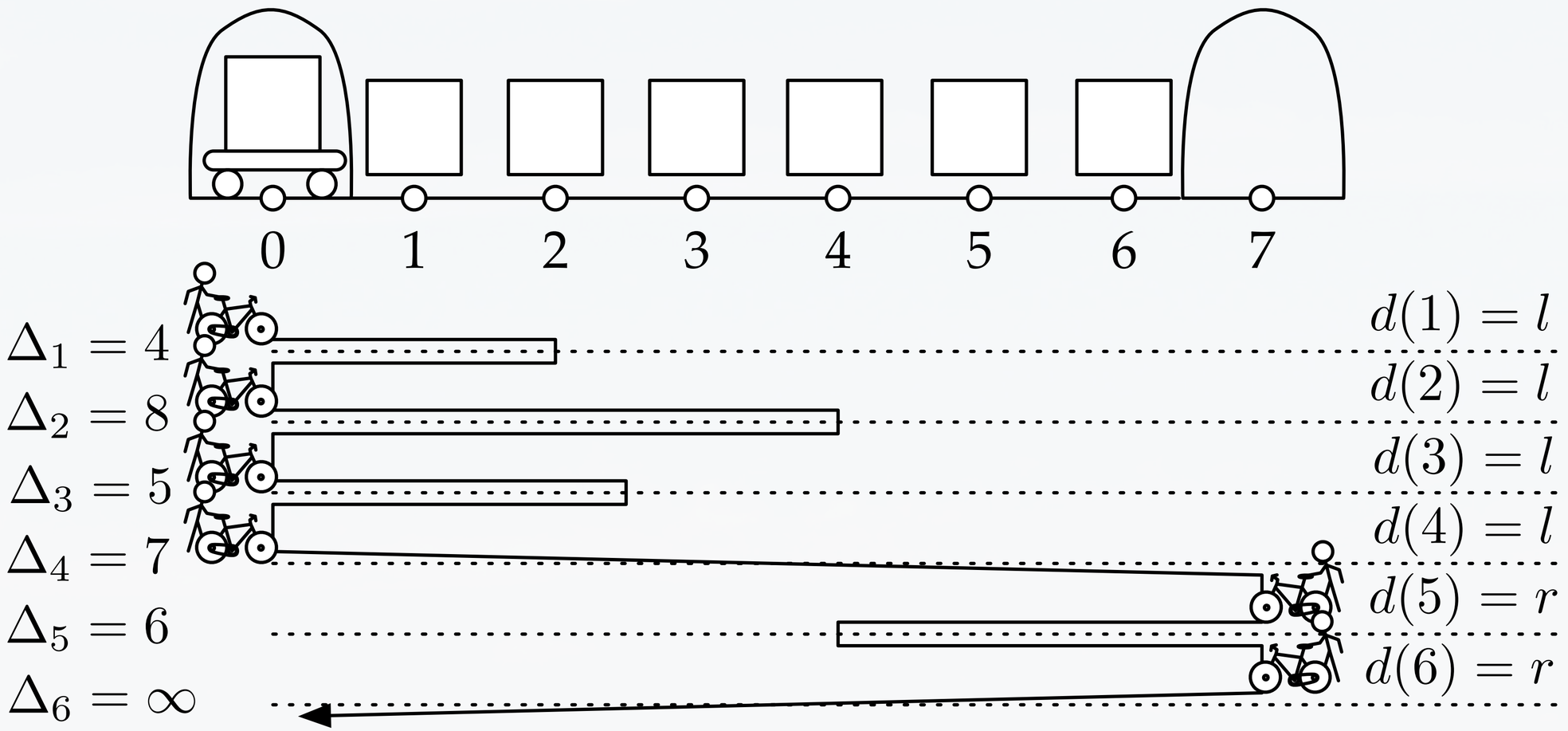


*Given:*

- inter arrival times $\Delta$
- door assignment $d$

*Wanted:*

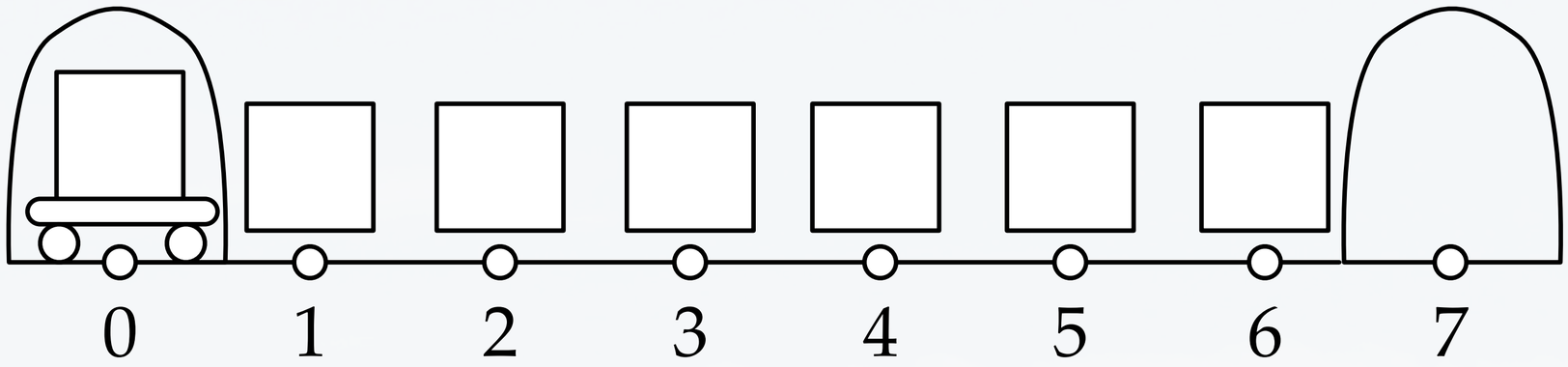
- slot assignment $\pi$



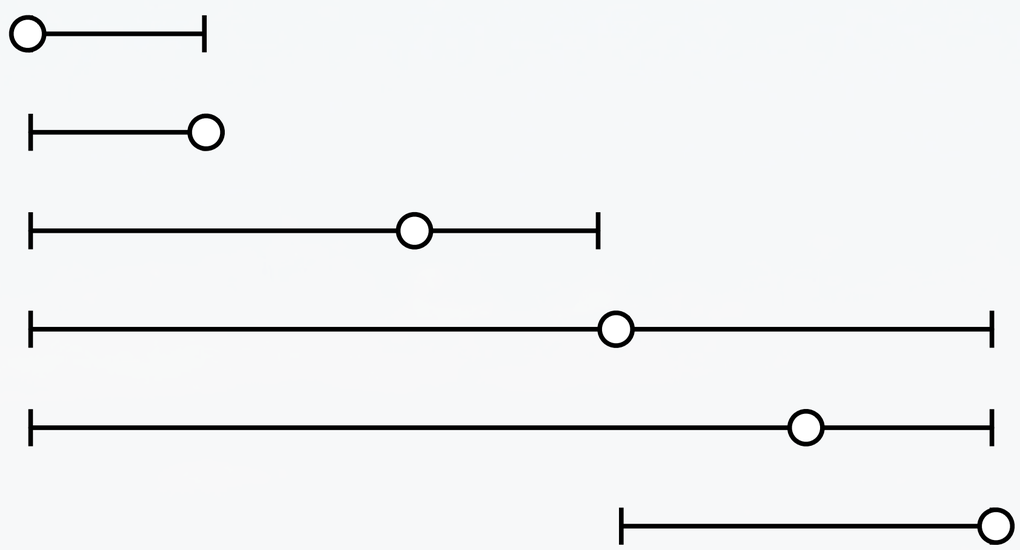
# Fixed Doors



*Solution:* earliest deadline first scheduling




$$\sigma = (1, 3, 2, 4, 6, 5)$$



ETH zürich
Daniel Graf

# Flexible Doors



*Given:*

- inter arrival times $\Delta$

*Wanted:*

- slot assignment $\pi$
- door assignment $d$

ETH zürich
Daniel Graf

# Flexible Doors



*Given:*

- inter arrival times $\Delta$

*Wanted:*

- slot assignment $\pi$
- door assignment $d$

*Theorem:* For a given $\Delta$, it is *NP*-complete to decide whether feasible $\pi$ and $d$ exist.

ETH zürich
Daniel Graf

# Flexible Doors



*Given:*

- inter arrival times $\Delta$

*Wanted:*

- slot assignment $\pi$
- door assignment $d$

*Theorem:* For a given $\Delta$, it is *NP*-complete to decide whether feasible $\pi$ and $d$ exist.

*Proof:* Reduction from *3SAT*.

# 3SAT



*Definition:* **3SAT**

- input: CNF-formula $F$ with
  - $M$ clauses $C_1, C_2, \ldots, C_M$ with $C_i = \{l_{i_1}, l_{i_2}, l_{i_3}\}$
  - $N$ variables $x_1, \ldots, x_N$
- output: can we satisfy all the clauses?

# 3SAT



*Definition:* **3SAT**

- input: CNF-formula $F$ with
  - $M$ clauses $C_1, C_2, \ldots, C_M$ with $C_i = \{l_{i_1}, l_{i_2}, l_{i_3}\}$
  - $N$ variables $x_1, \ldots, x_N$
- output: can we satisfy all the clauses?

*Example:* $F = \{\{x_1, x_3, \bar{x}_4\}, \{x_2, \bar{x}_3, x_4\}, \{\bar{x}_1, x_2, \bar{x}_4\}\}$

# 3SAT



*Definition:* **3SAT**

- input: CNF-formula $F$ with
  - $M$ clauses $C_1, C_2, \ldots, C_M$ with $C_i = \{l_{i_1}, l_{i_2}, l_{i_3}\}$
  - $N$ variables $x_1, \ldots, x_N$
- output: can we satisfy all the clauses?

*Example:* $F = \{\{x_1, x_3, \bar{x}_4\}, \{x_2, \bar{x}_3, x_4\}, \{\bar{x}_1, x_2, \bar{x}_4\}\}$

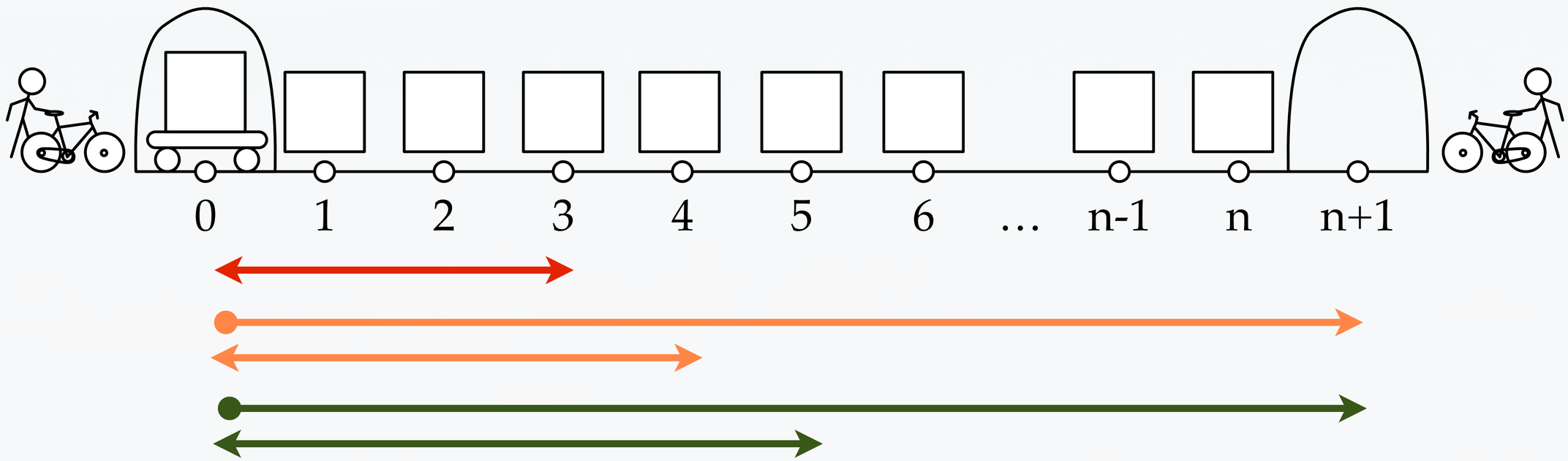
*Theorem [1971, Cook], [1973, Levin]:*
3SAT is *NP*-complete.

# Request Sizes



We classify inter request times $\Delta_i$ by duration:

- short: $\Delta_i \leq n$: the other door is unreachable
- medium: $n < \Delta_i < 2n$: in between
- long: $2n < \Delta_i$: switch or not and reach any slot



# Request Blocks



Consecutive short requests have to be assigned to the same door *en bloc*.

Transform a given *3SAT* formula into a sequence of blocks and interleave them with long requests.

# Request Blocks



Consecutive short requests have to be assigned to the same door *en bloc*.

Transform a given *3SAT* formula into a sequence of blocks and interleave them with long requests.

$$F = \{\{x_1, x_3, \bar{x}_4\}, \{x_2, \bar{x}_3, x_4\}, \{\bar{x}_1, x_2, \bar{x}_4\}\}$$

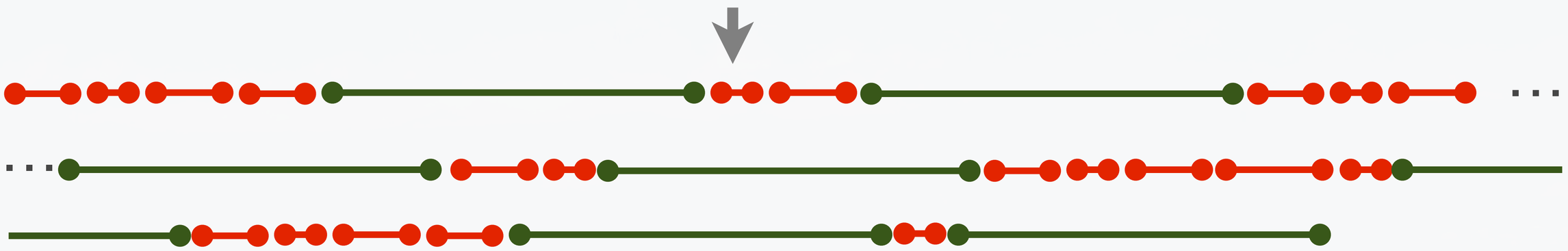
# Request Blocks



Consecutive short requests have to be assigned to the same door *en bloc*.

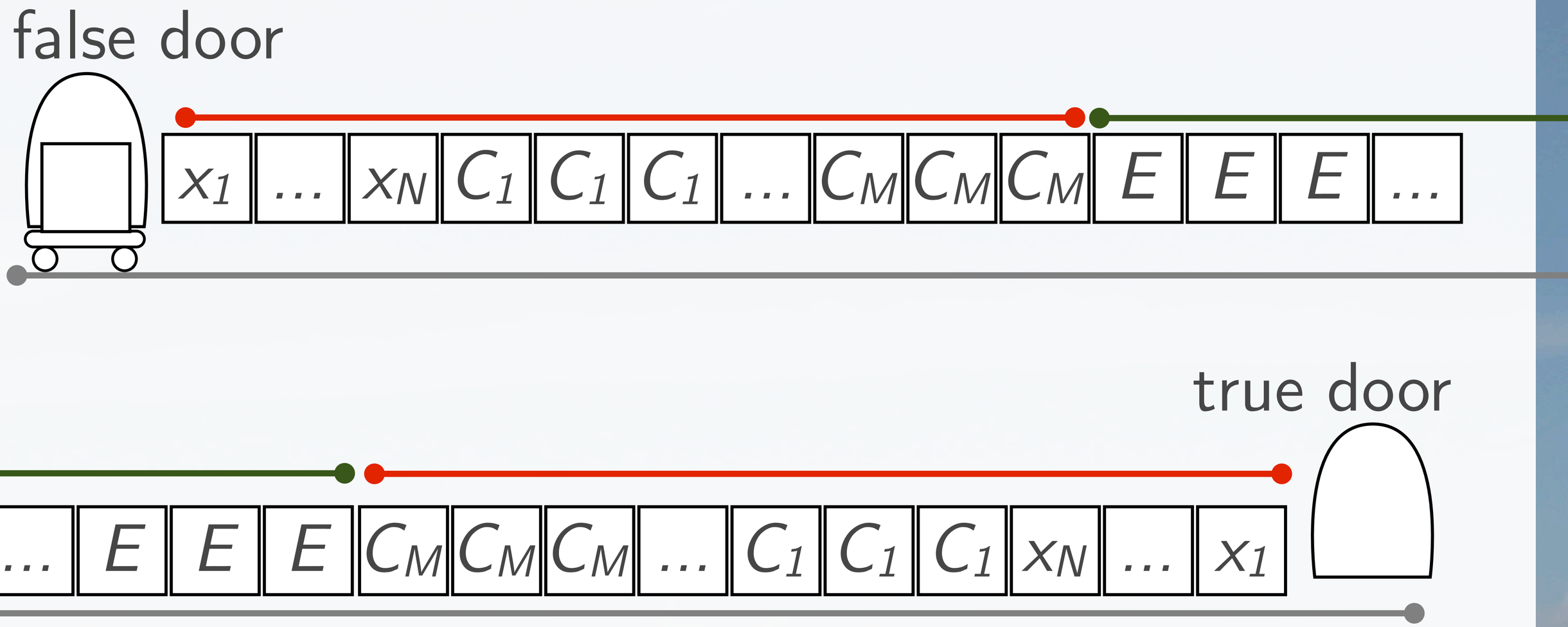
Transform a given *3SAT* formula into a sequence of blocks and interleave them with long requests.

$$F = \{\{x_1, x_3, \bar{x}_4\}, \{x_2, \bar{x}_3, x_4\}, \{\bar{x}_1, x_2, \bar{x}_4\}\}$$



# Layout

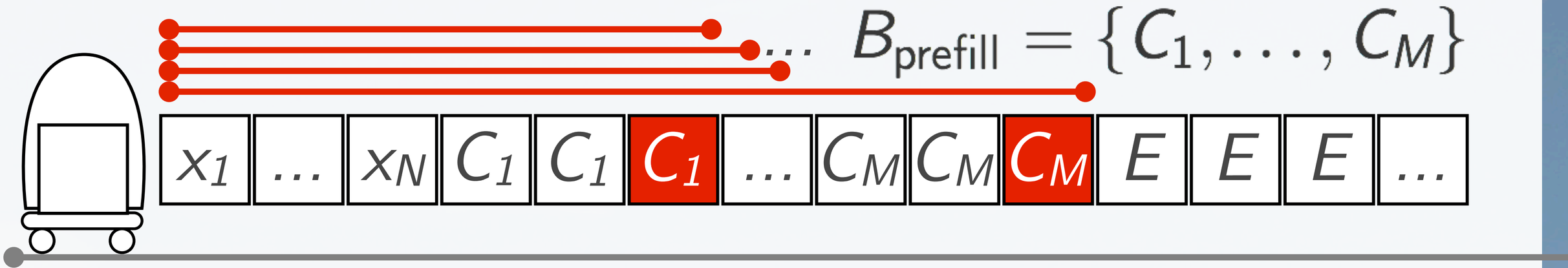



false door
$x_1$ ... $x_N$ $C_1$ $C_1$ $C_1$ ... $C_M$ $C_M$ $C_M$ $E$ $E$ $E$ ...
... $E$ $E$ $E$ $C_M$ $C_M$ $C_M$ ... $C_1$ $C_1$ $C_1$ $x_N$ ... $x_1$
true door

# Blocks



**Prefill block:** fill one slot per clause on the false side

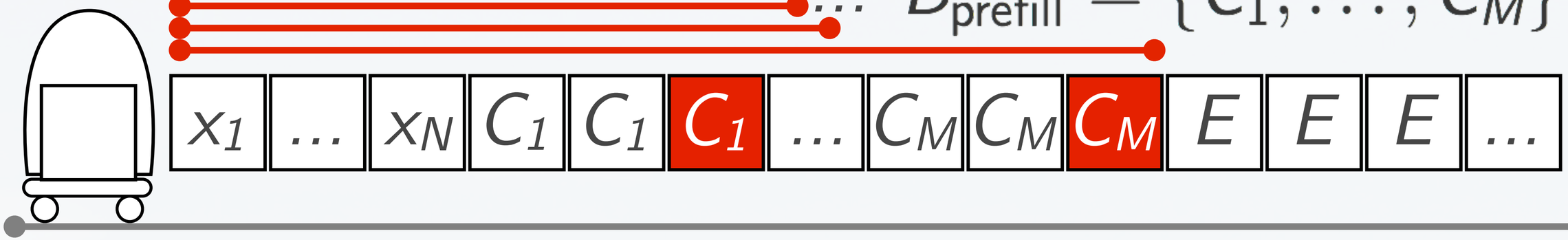


# Blocks



**Prefill block:** fill one slot per clause on the false side

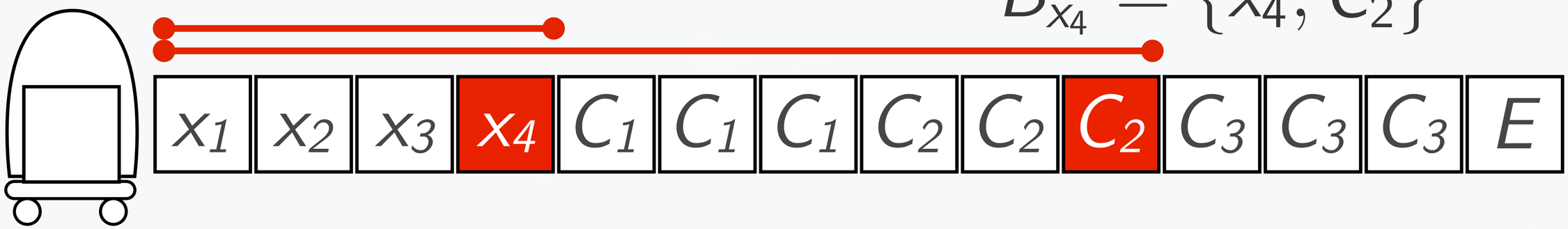
$B_{\text{prefill}} = \{C_1, \ldots, C_M\}$




**Literal blocks:** variable and all clauses containing it

*Example:* $F = \{\{x_1, x_3, \bar{x}_4\}, \{x_2, \bar{x}_3, x_4\}, \{\bar{x}_1, x_2, \bar{x}_4\}\}$

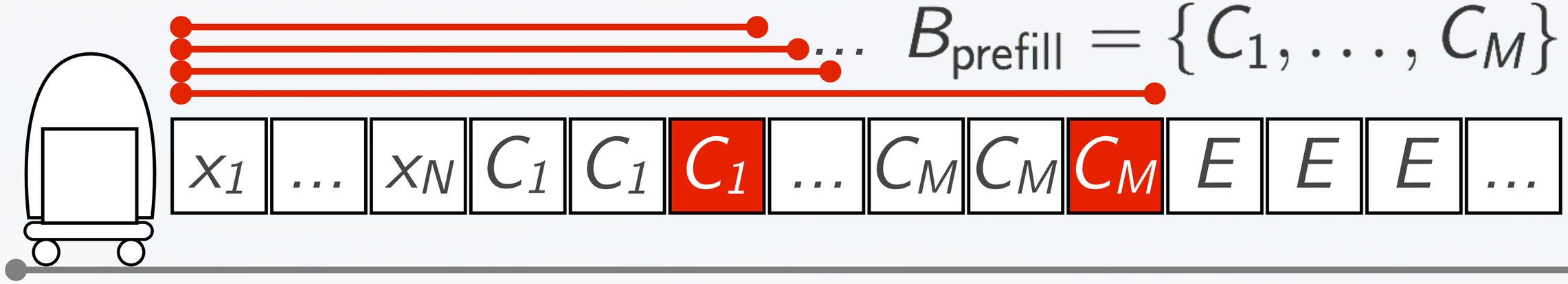
$B_{x_4} = \{x_4, C_2\}$



# Blocks

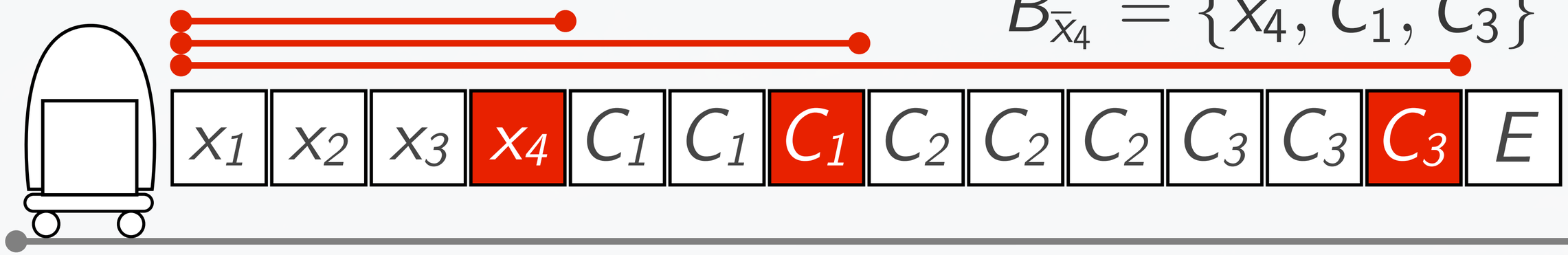


**Prefill block:** fill one slot per clause on the false side




**Literal blocks:** variable and all clauses containing it

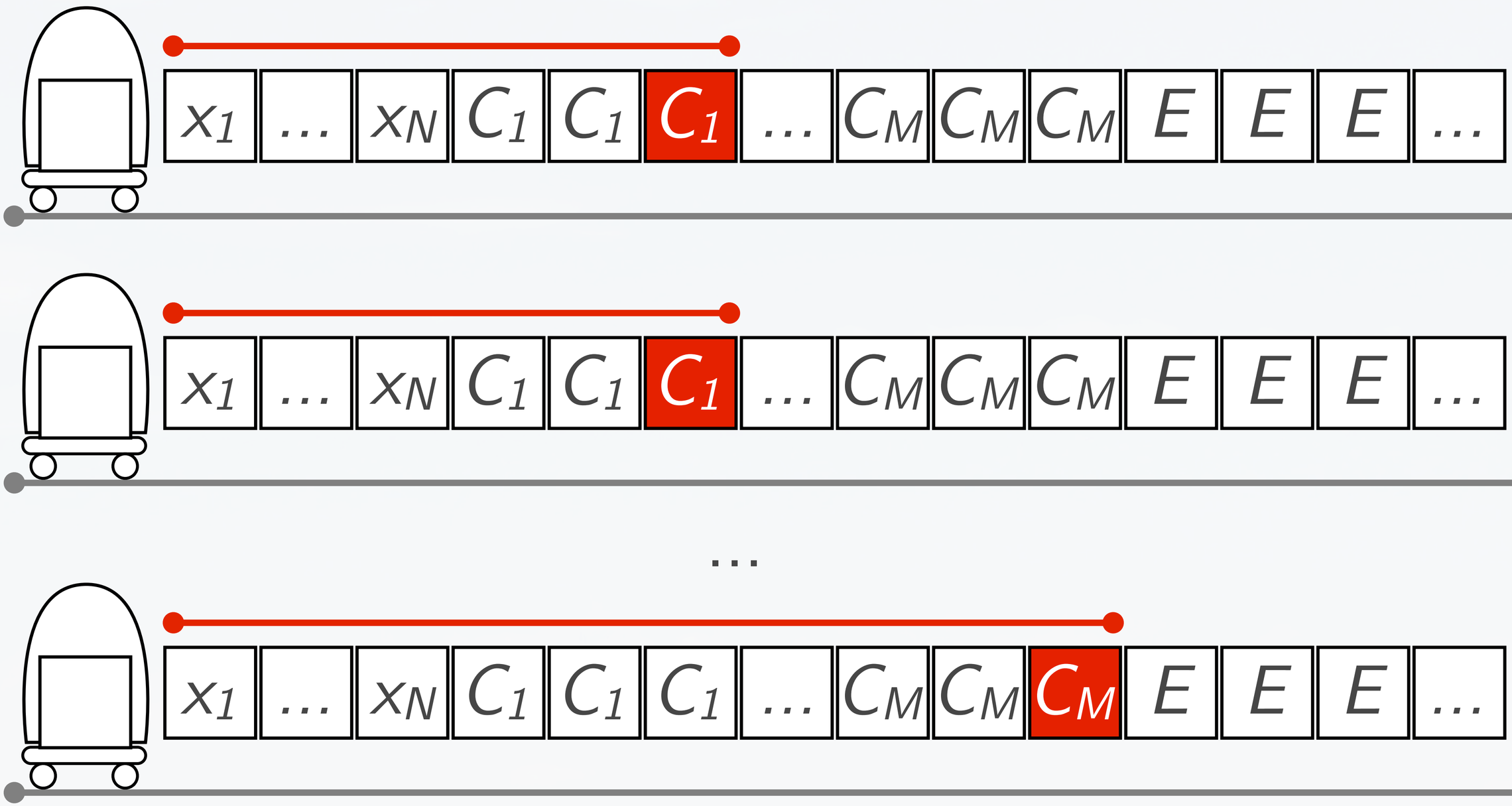
*Example:* $F = \{\{x_1, x_3, \bar{x}_4\}, \{x_2, \bar{x}_3, x_4\}, \{\bar{x}_1, x_2, \bar{x}_4\}\}$



# Blocks



**Fillup blocks:** two single request blocks per clause



# Proof



*Claim:* The two door problem has a solution if and only if the *3SAT* formula $F$ is satisfiable.

# Proof



*Claim:* The two door problem has a solution if and only if the *3SAT* formula $F$ is satisfiable.

*Proof:*

- Per label there are as many slots as requests.
- The long requests fill up the $E$ slots.
- Opposite literals get assigned to opposite doors.
- Per clause at least one literal block has to get assigned to the true door as one slot on the false side is already taken by the prefill block.

# Entire Day



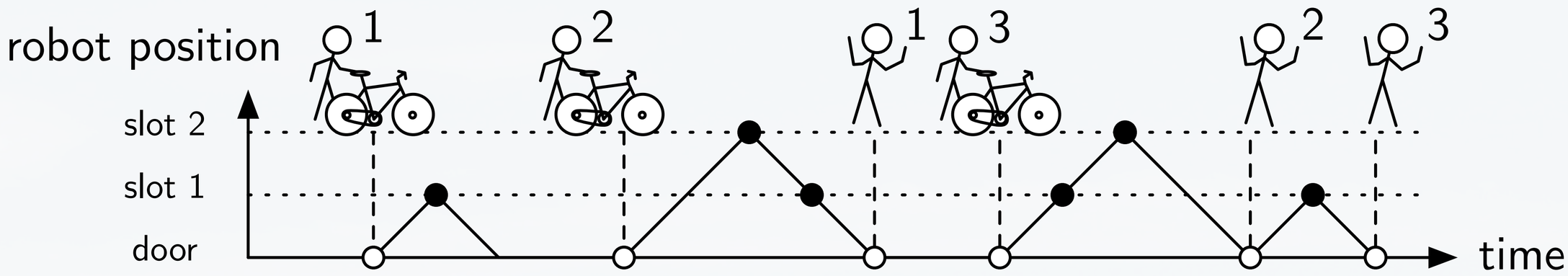
Back to a single door, but also with departures:

# Entire Day



Back to a single door, but also with departures:

*Example:*

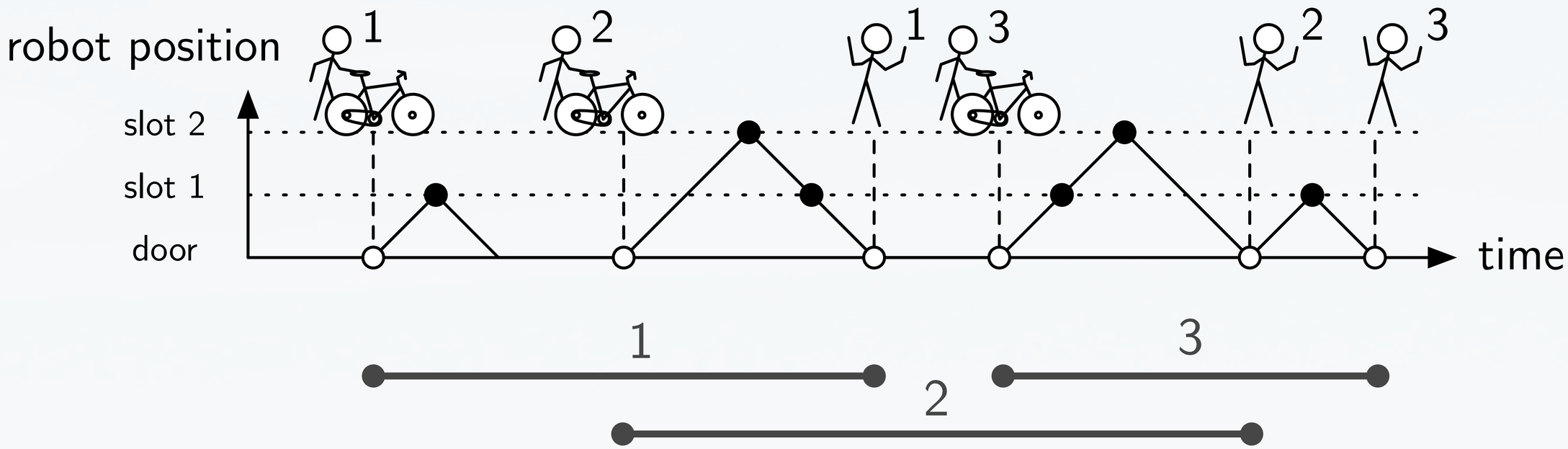


# Entire Day



Back to a single door, but also with departures:

*Example:*

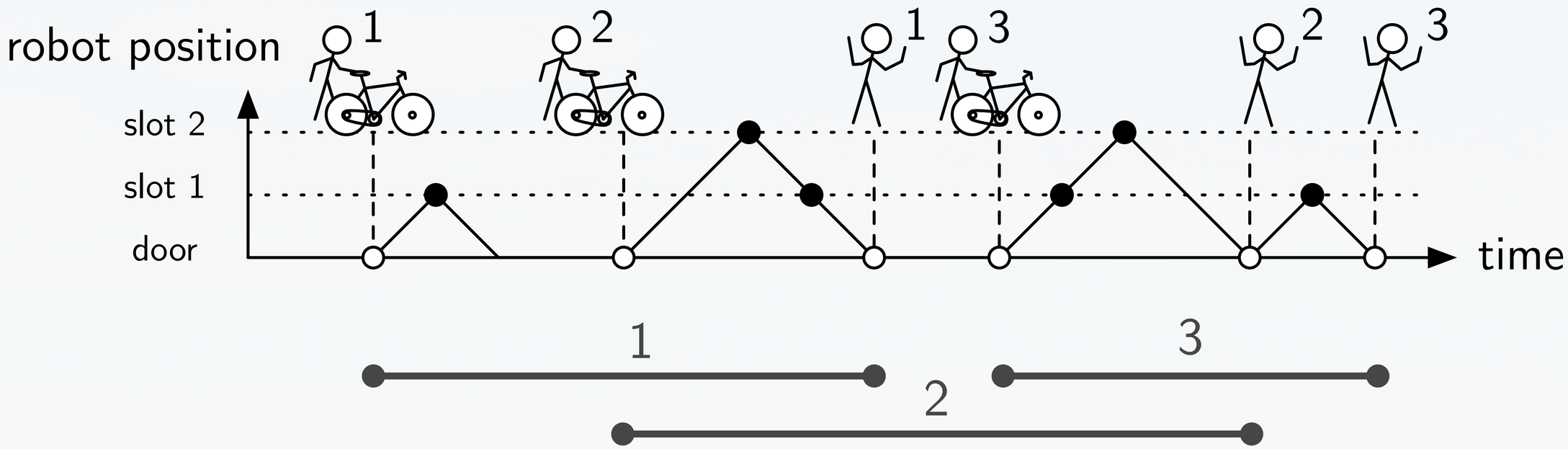


# Entire Day



Back to a single door, but also with departures:

*Example:* It is not just coloring intervals!

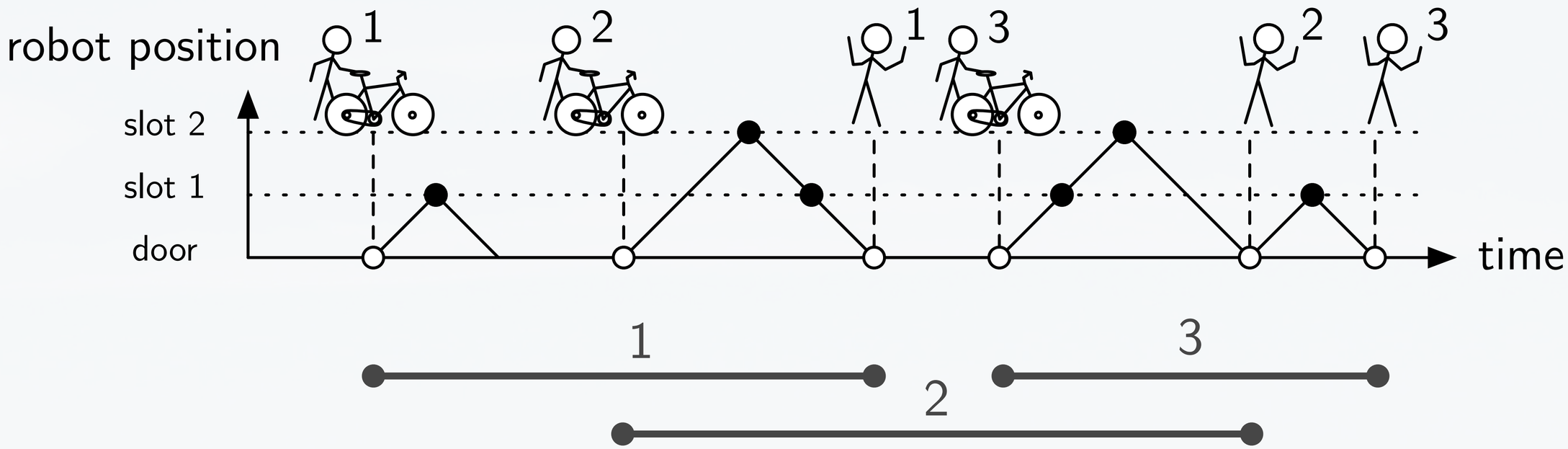


# Entire Day



Back to a single door, but also with departures:

*Example:* It is not just coloring intervals!



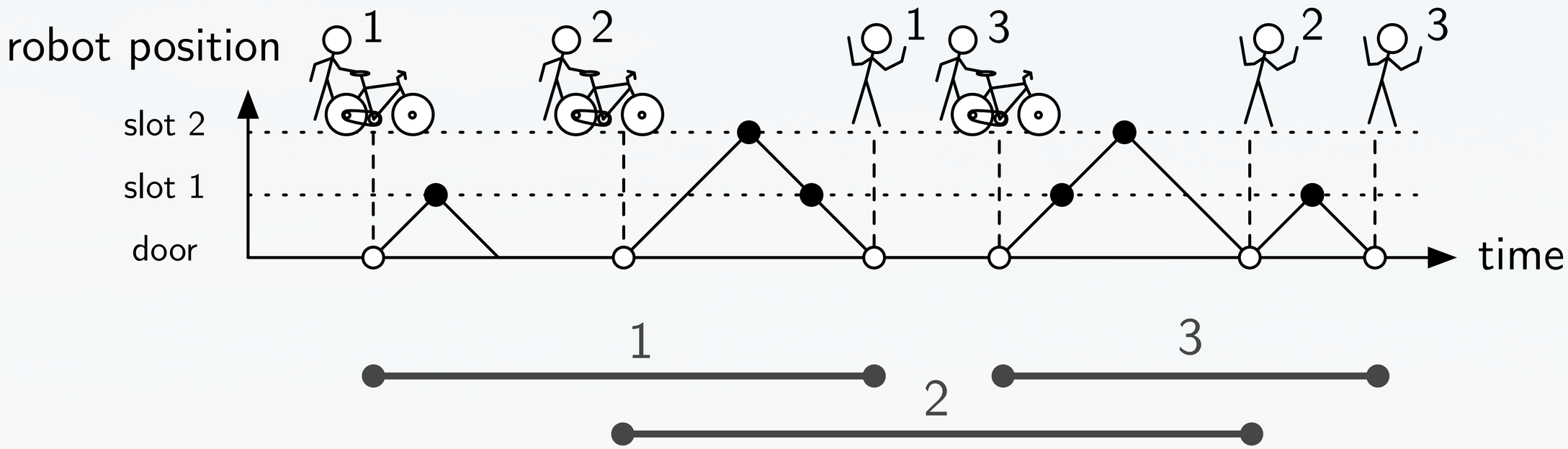

*Theorem:* It is *NP*-complete to find a feasible $\pi$.

# Entire Day



Back to a single door, but also with departures:

*Example:* It is not just coloring intervals!




*Theorem:* It is *NP*-complete to find a feasible $\pi$.

*Proof:* Reduction from *Circular Arc Coloring*.

ETH zürich
Daniel Graf

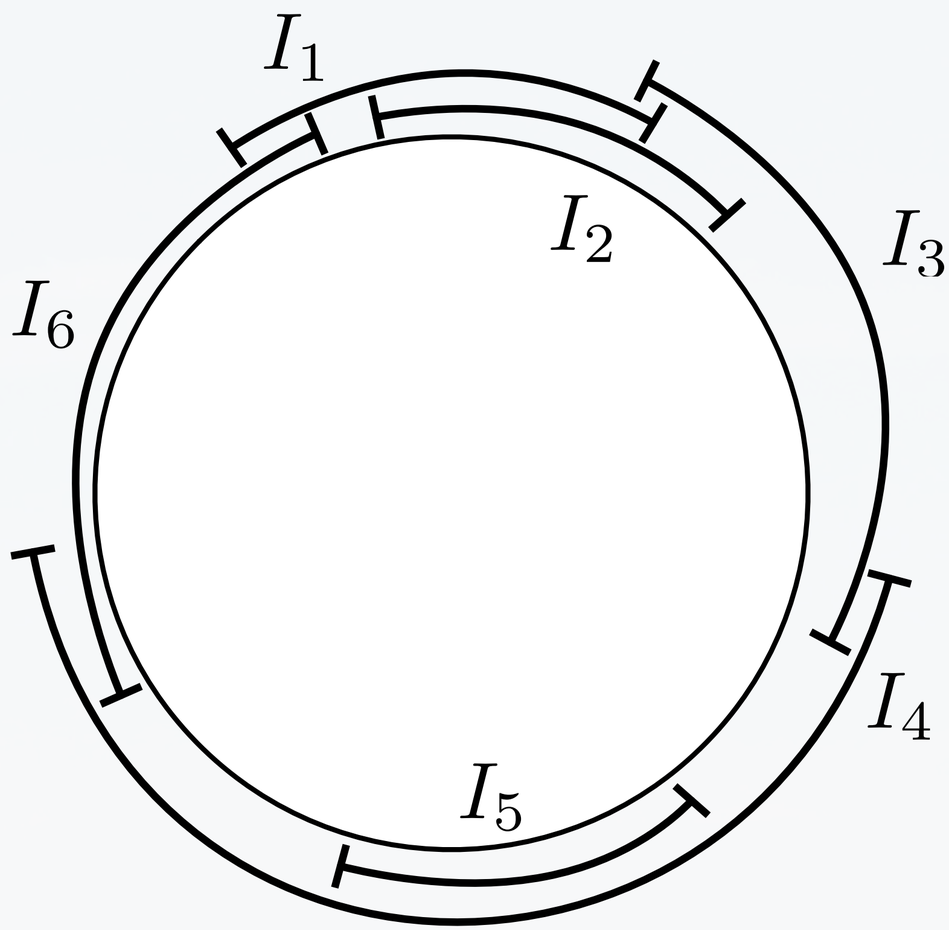
# Arc Coloring



*Definition:* **Circular Arc Coloring**

Given $I$ arcs on a circle, is there a $k$-coloring of their intersection graph?

*I=6, k=3*



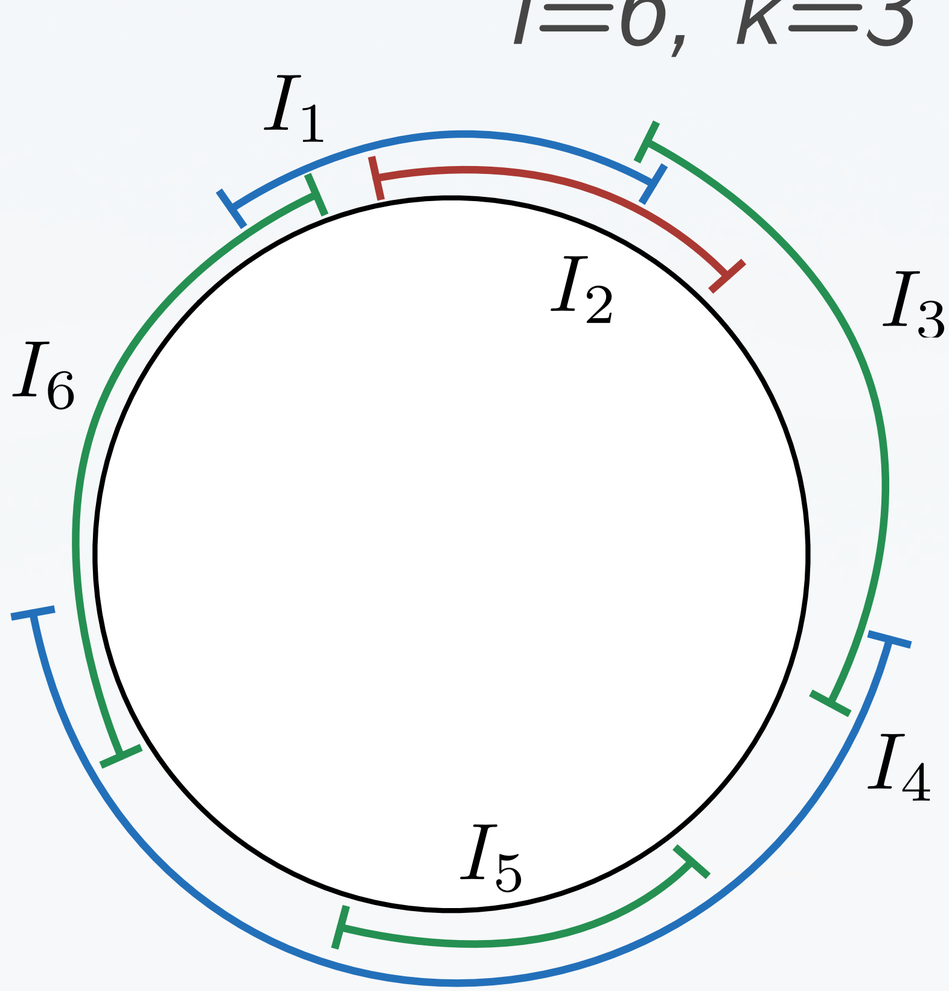
ETH zürich
Daniel Graf

# Arc Coloring



*Definition:* **Circular Arc Coloring**

Given $l$ arcs on a circle, is there a $k$-coloring of their intersection graph?



ETH zürich
Daniel Graf

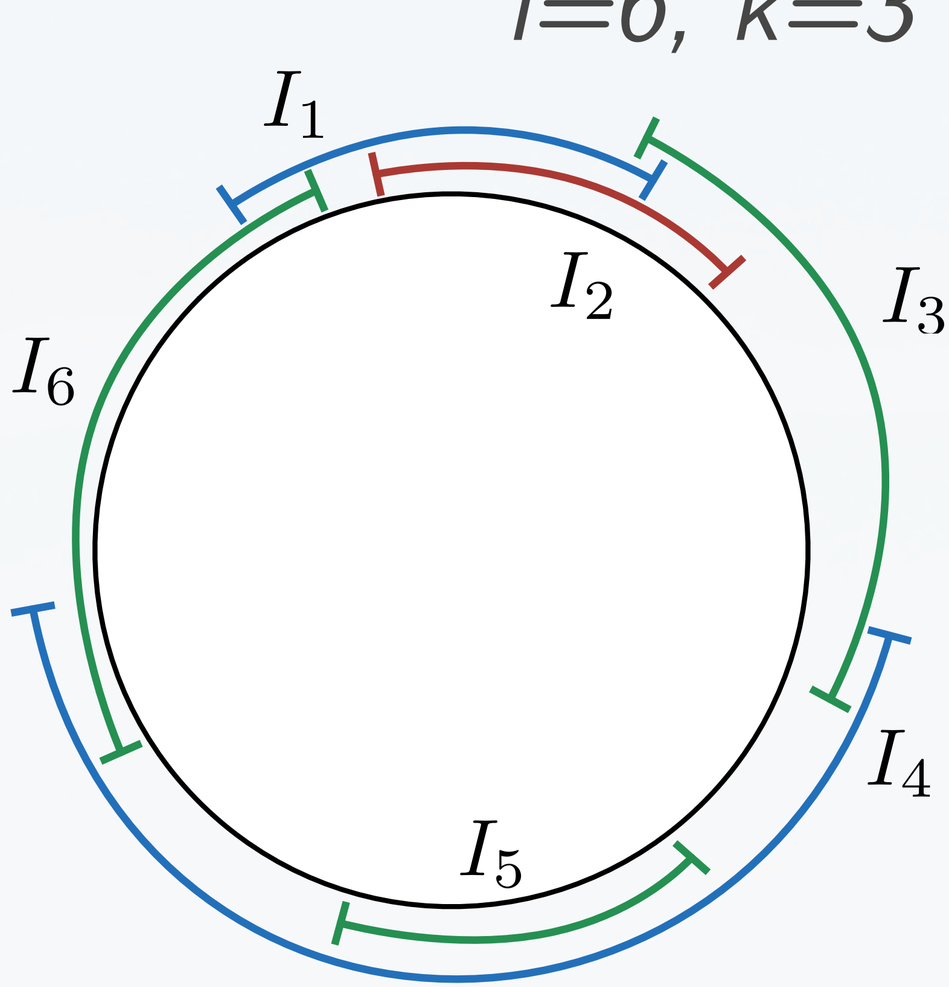
# Arc Coloring



*Definition:* **Circular Arc Coloring**

Given $I$ arcs on a circle, is there a $k$-coloring of their intersection graph?

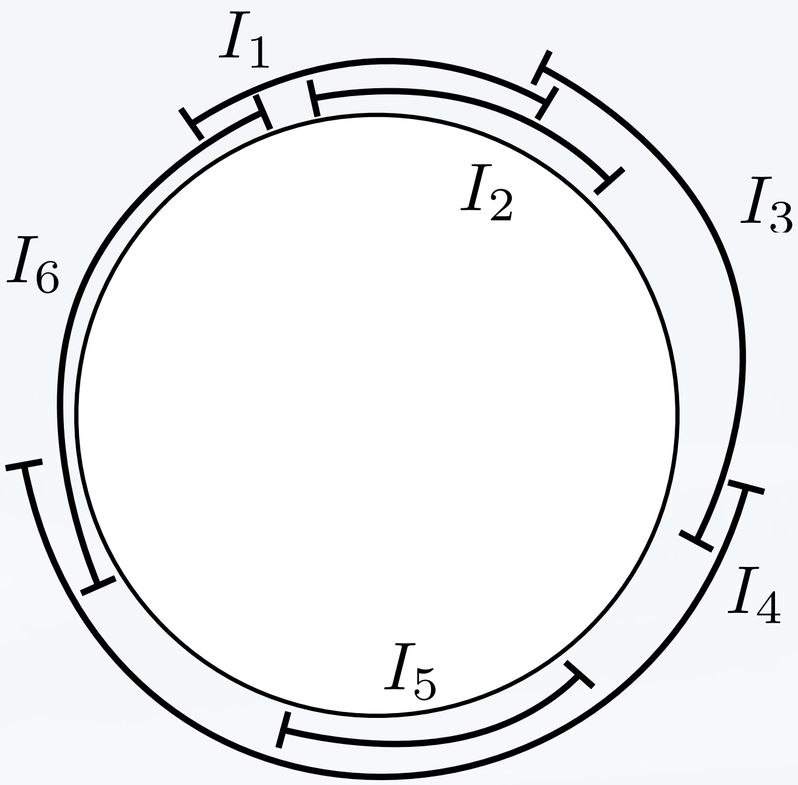
*Theorem [1980, Garey et al.]:*
Circular Arc Coloring is *NP*-complete.



ETH zürich
Daniel Graf

# Reduction





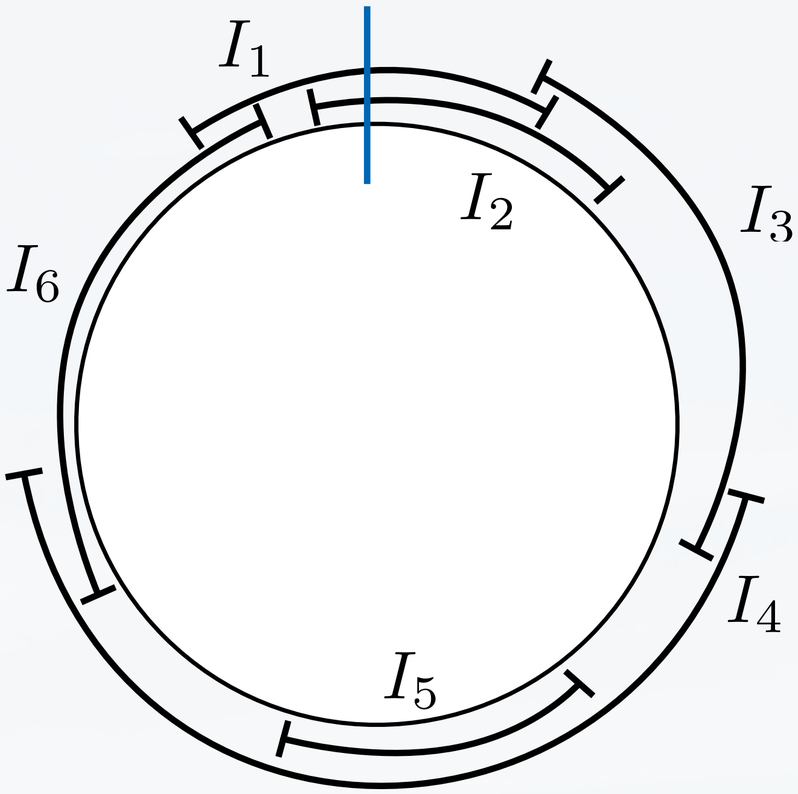

Outline of the reduction

- split the arcs at position 0
- arbitrarily, but consistently color the arcs across this cut
- transform it into an all-day instance with $k$ slots

ETH zürich
Daniel Graf

# Reduction





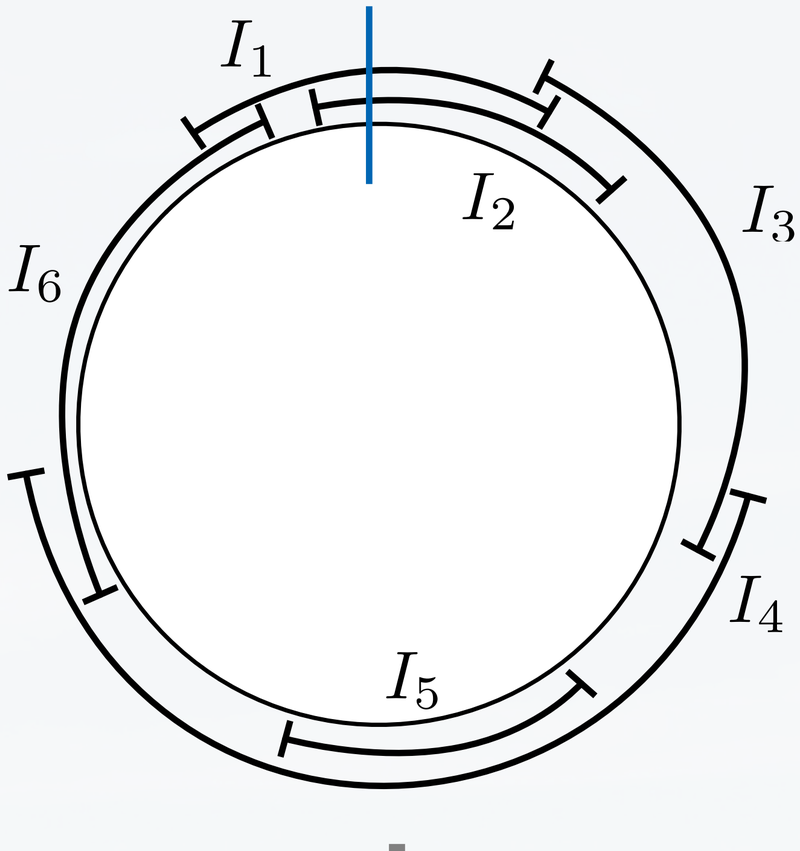

Outline of the reduction

- split the arcs at position 0
- arbitrarily, but consistently color the arcs across this cut
- transform it into an all-day instance with $k$ slots

ETH zürich
Daniel Graf

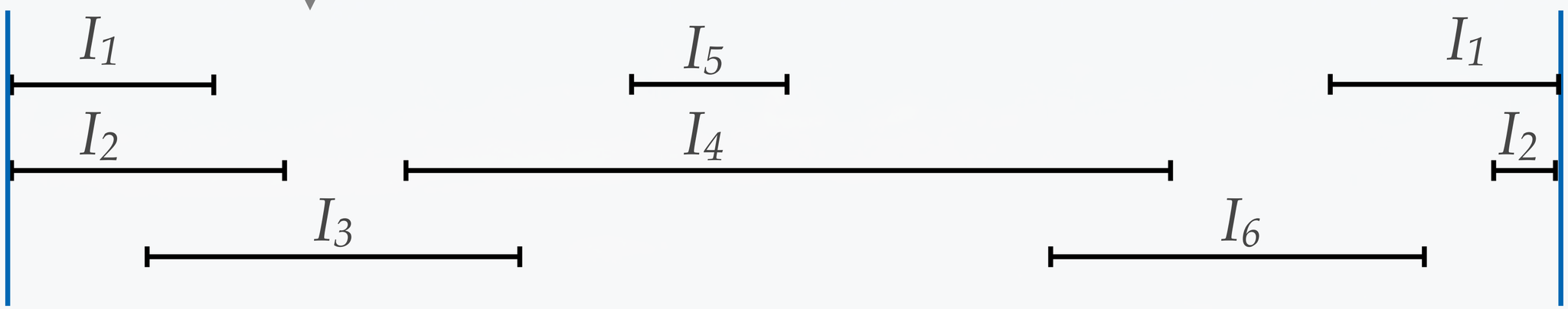
# Reduction






Outline of the reduction

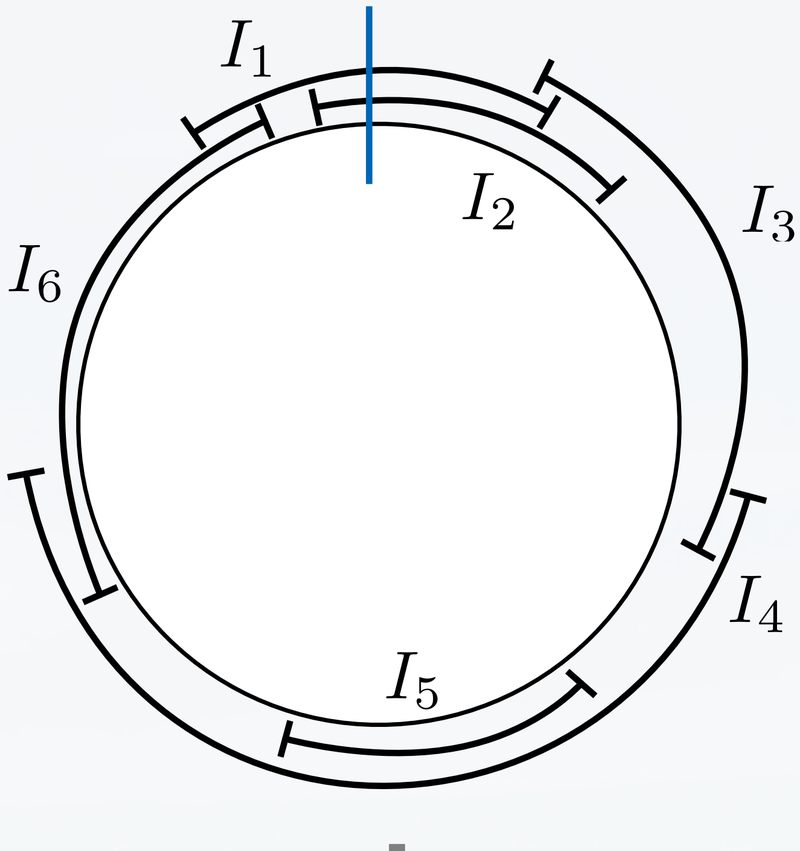
- split the arcs at position 0
- arbitrarily, but consistently color the arcs across this cut
- transform it into an all-day instance with $k$ slots



ETH zürich
Daniel Graf

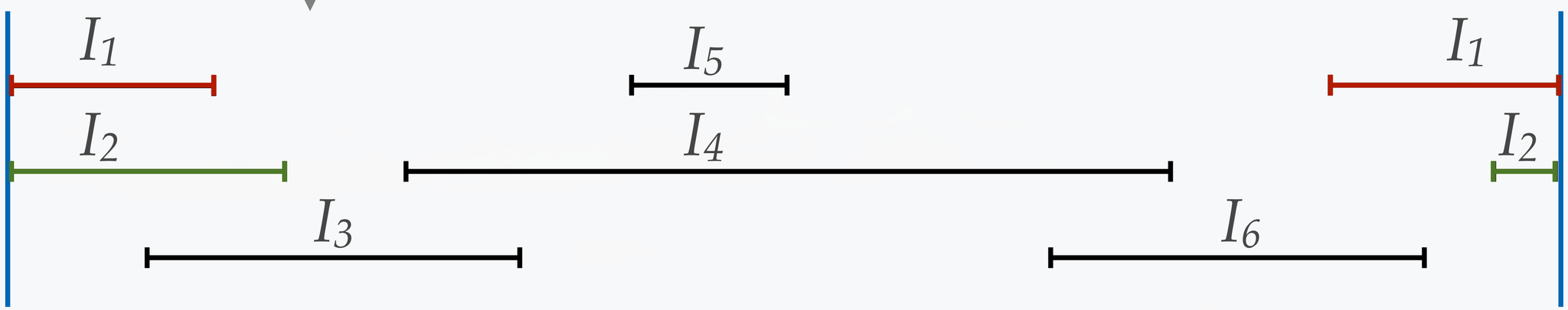
# Reduction






Outline of the reduction

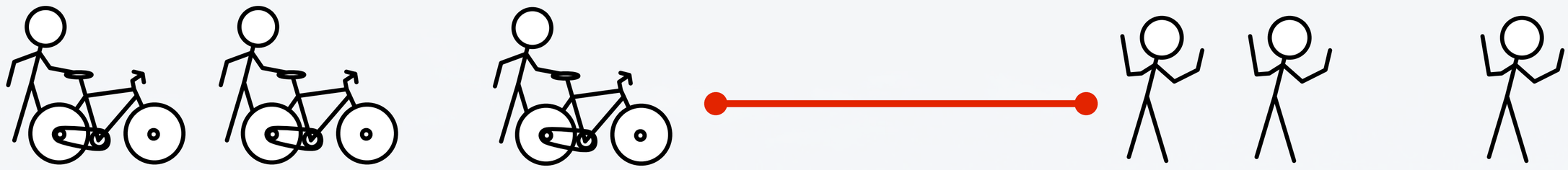
- split the arcs at position 0
- arbitrarily, but consistently color the arcs across this cut
- transform it into an all-day instance with $k$ slots



ETH zürich
Daniel Graf

# Lunch Break





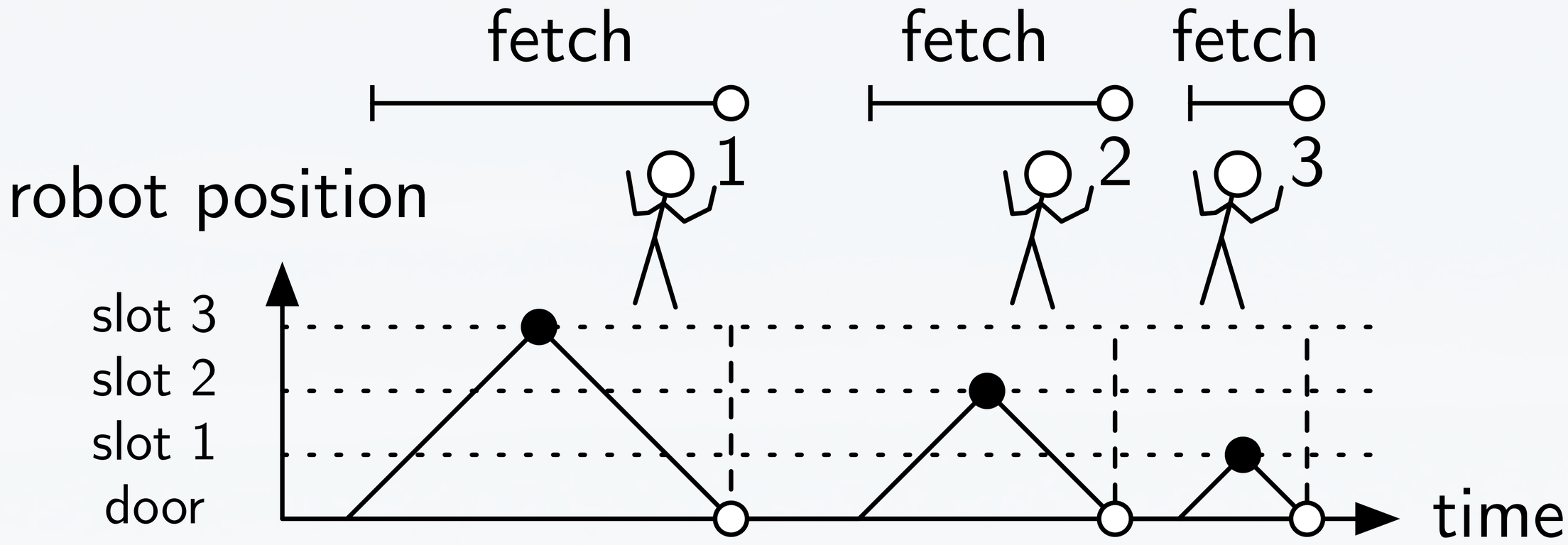
What if all the customers arrive in the morning and depart only in the evening?

- Every slot used only for a single customer
- Arrival-only problem in the morning
- Sorting problem during lunchtime
- Departure-only problem in the afternoon

# Afternoon



*Example:*

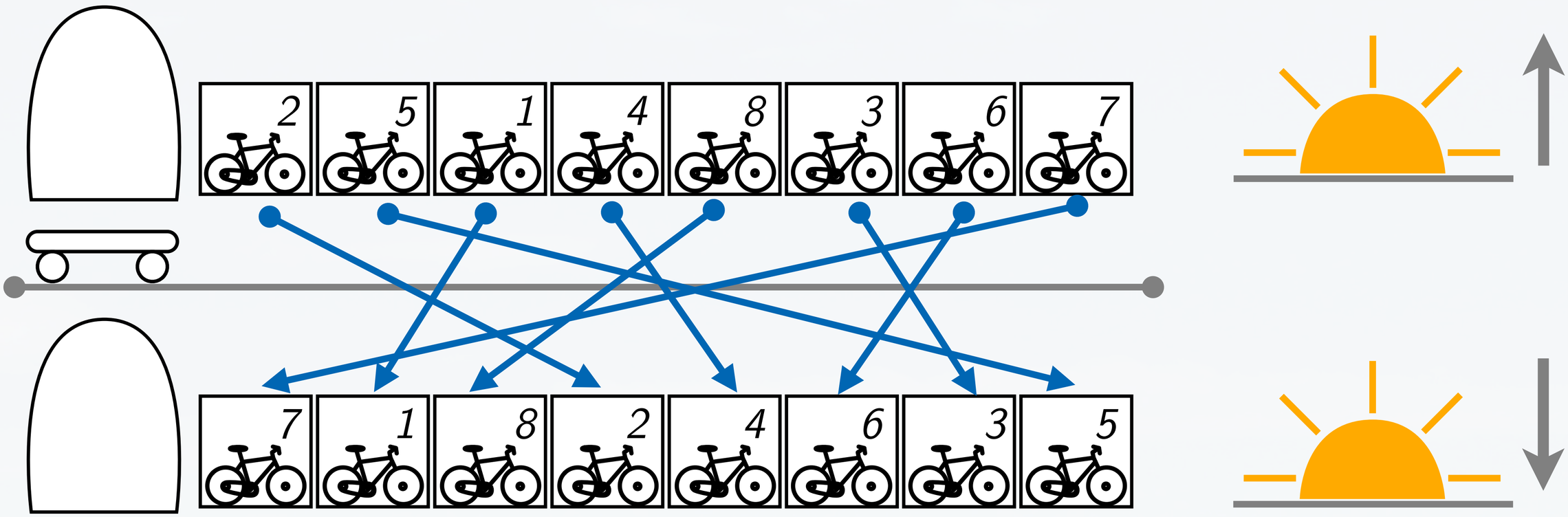



Solving another instance of the constrained permutation problem.

ETH zürich
Daniel Graf

# Lunch Break










How do we sort this as quickly as possible?

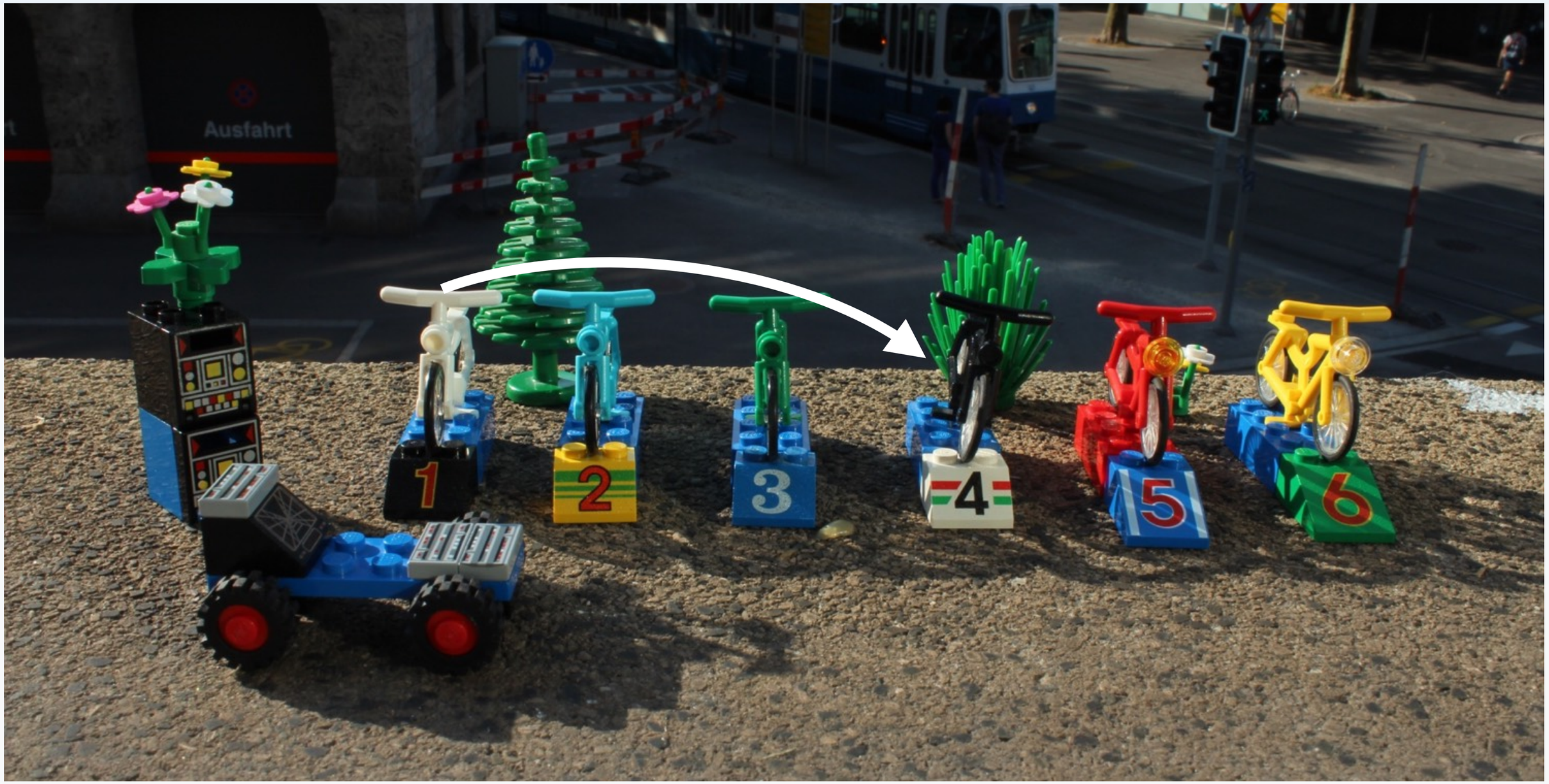
ETH zürich
Daniel Graf

# Sorting Problem

ETH zürich

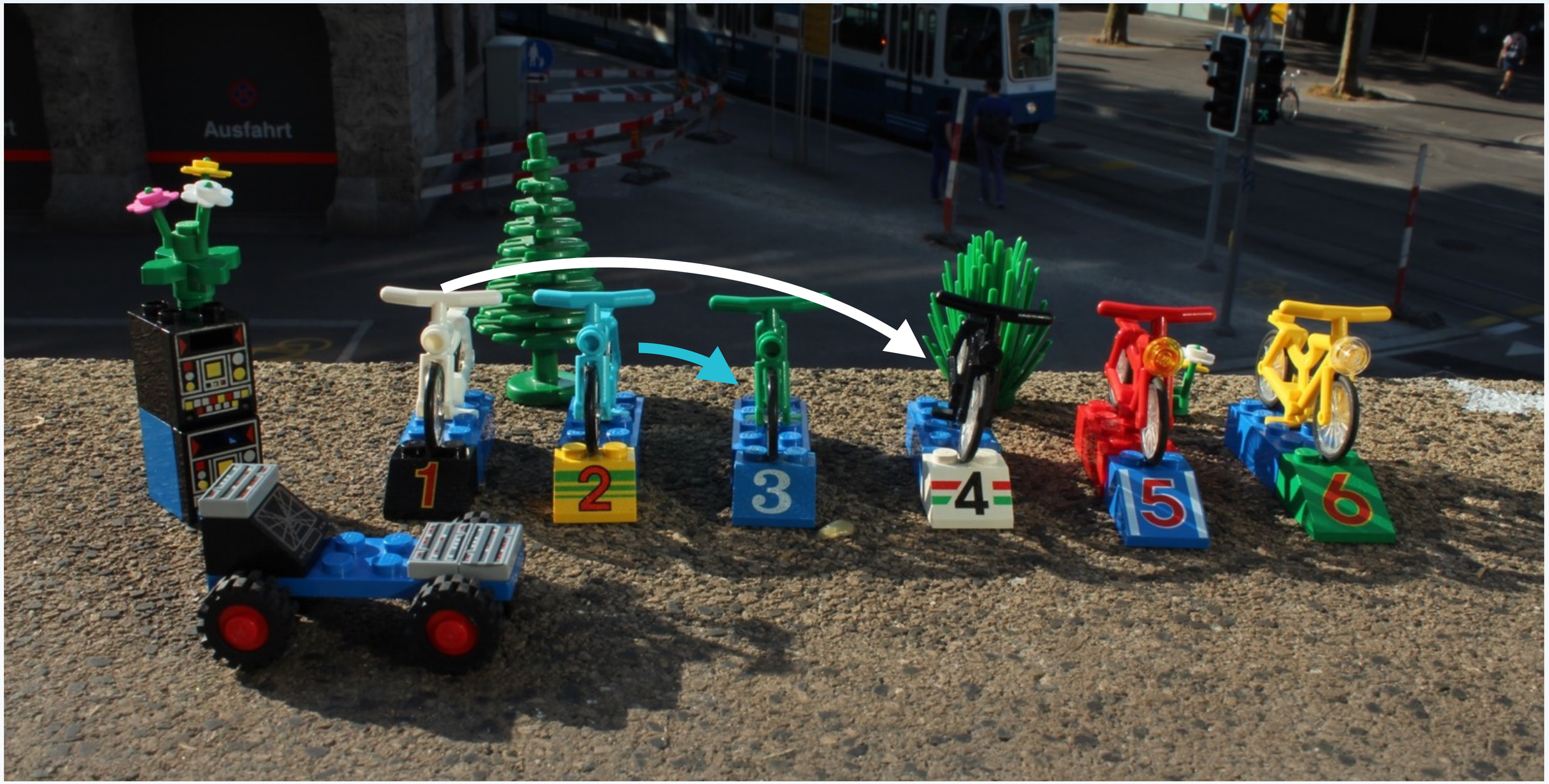
Daniel Graf

# Sorting Problem



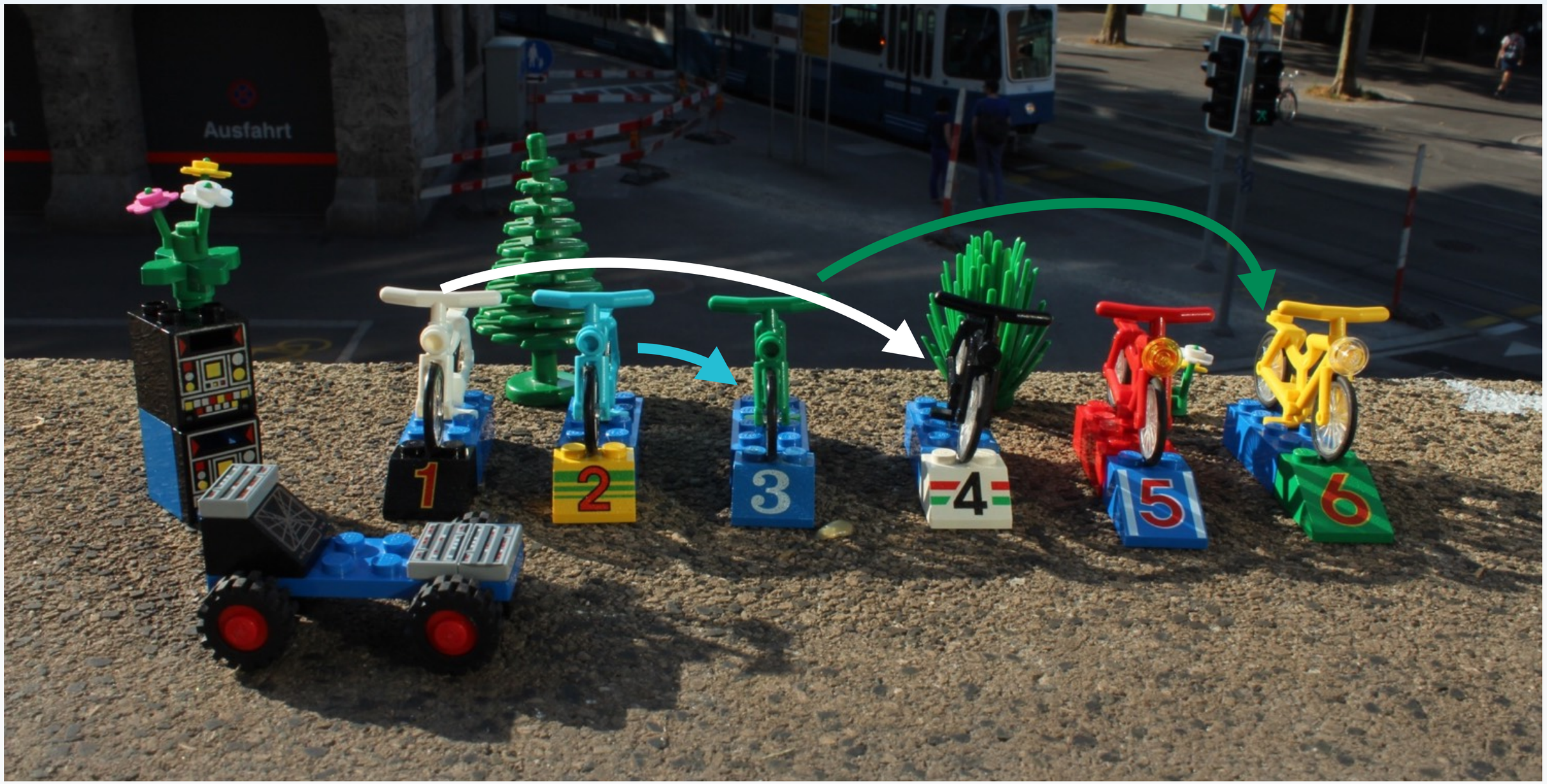
ETH zürich
Daniel Graf

# Sorting Problem



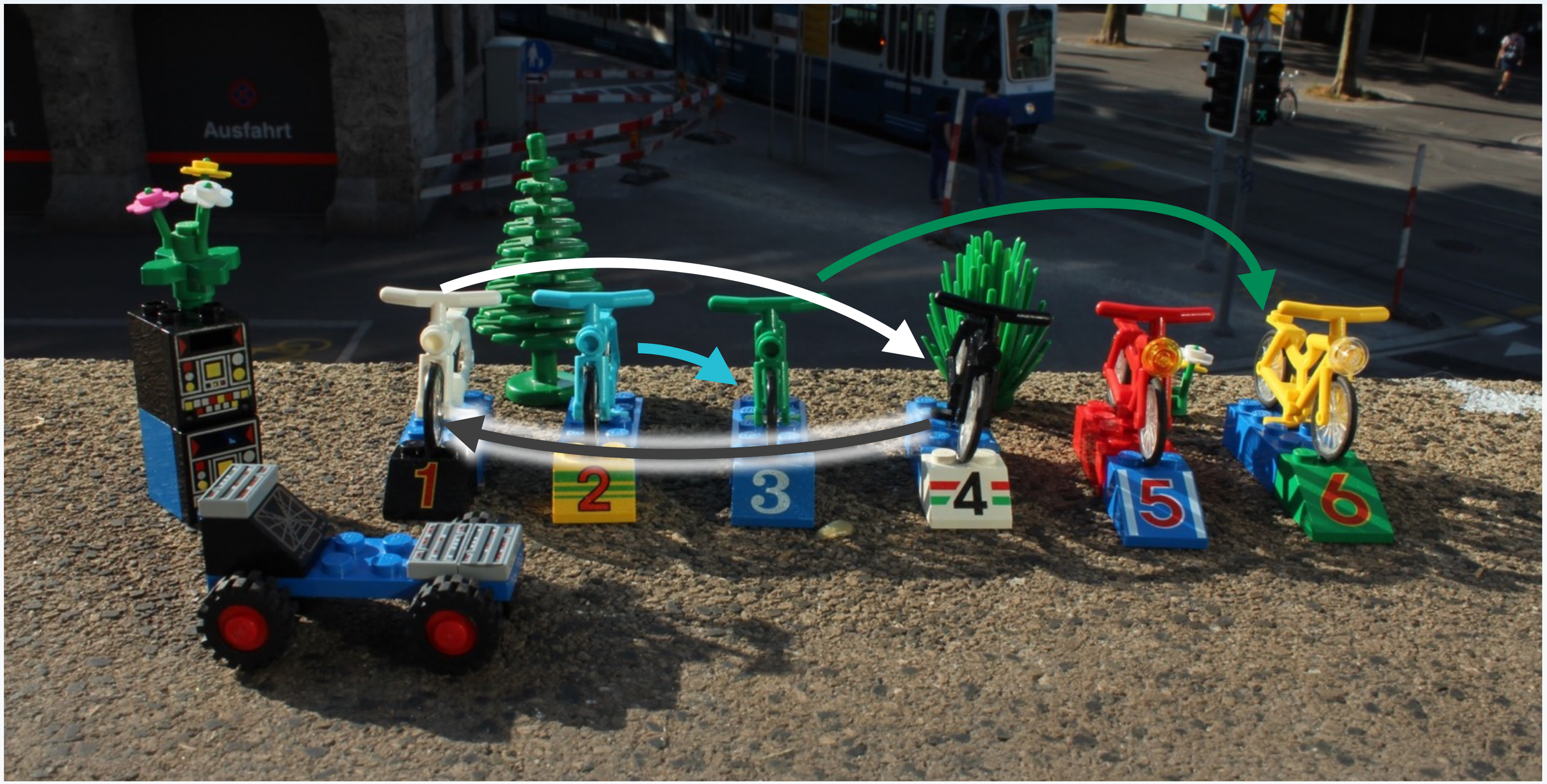
ETH zürich
Daniel Graf

# Sorting Problem



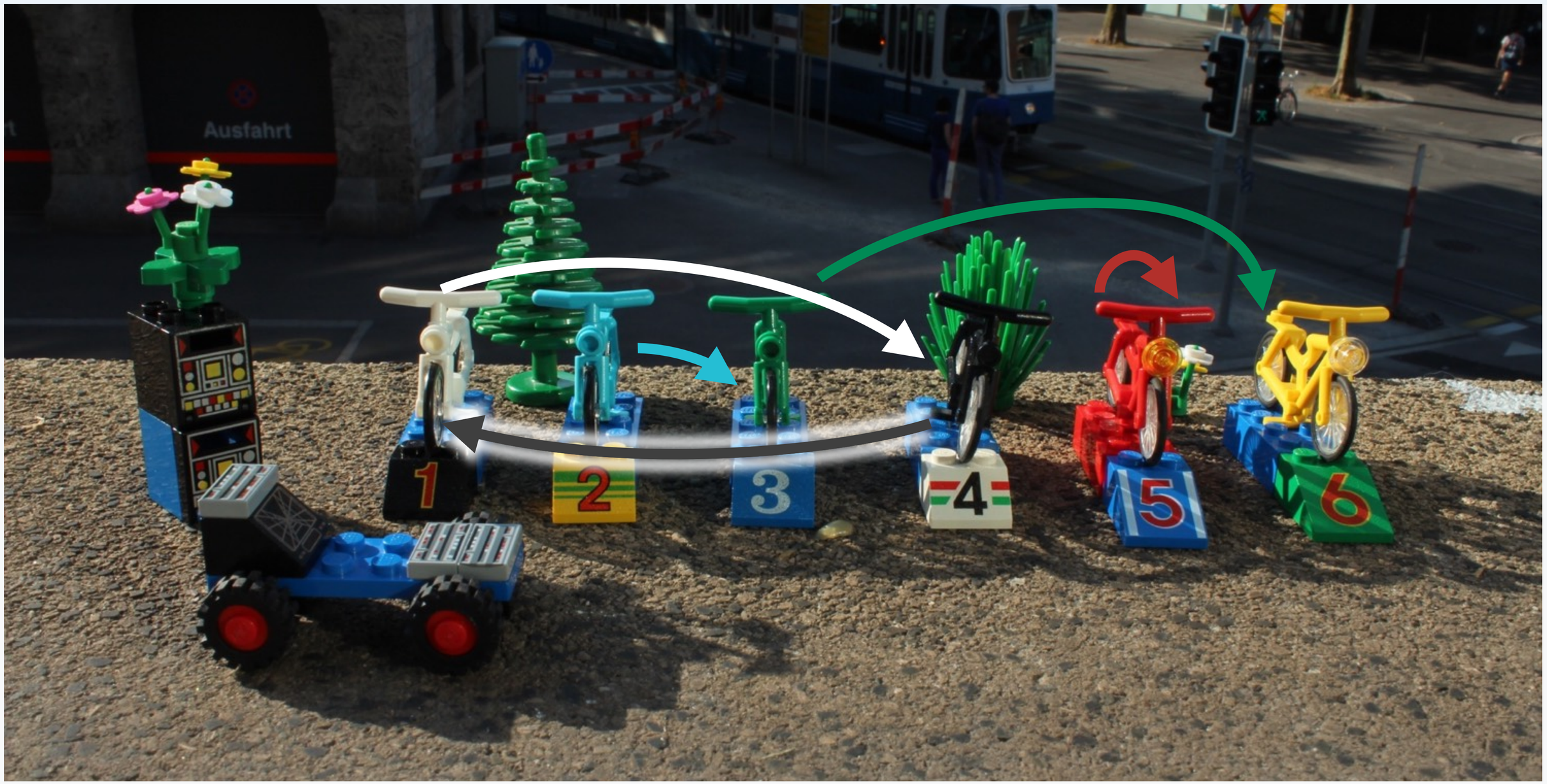
ETH zürich
Daniel Graf

# Sorting Problem



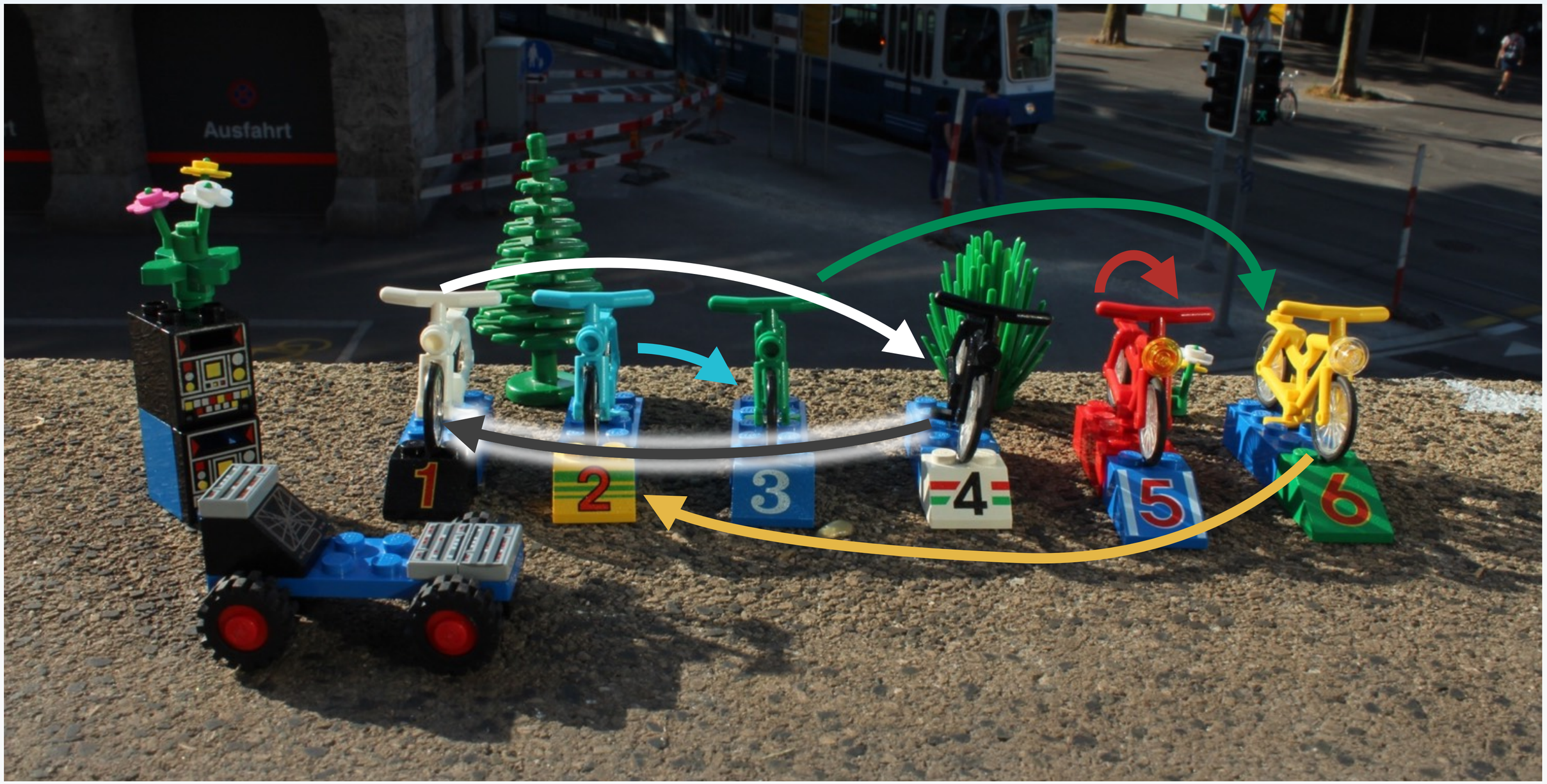
ETH zürich
Daniel Graf

# Sorting Problem



ETH zürich
Daniel Graf

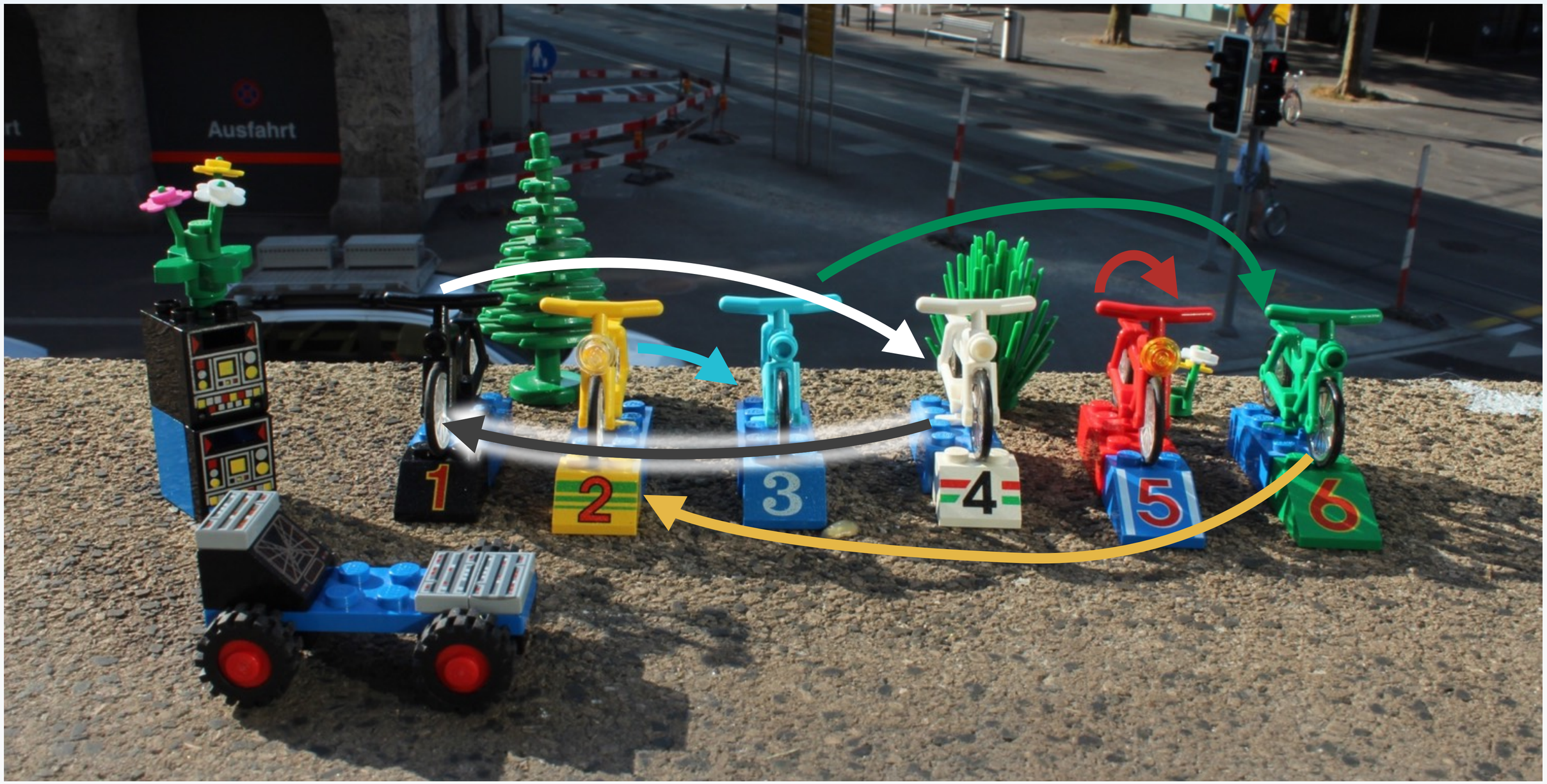
# Sorting Problem



ETH zürich

Daniel Graf

# Sorting Problem

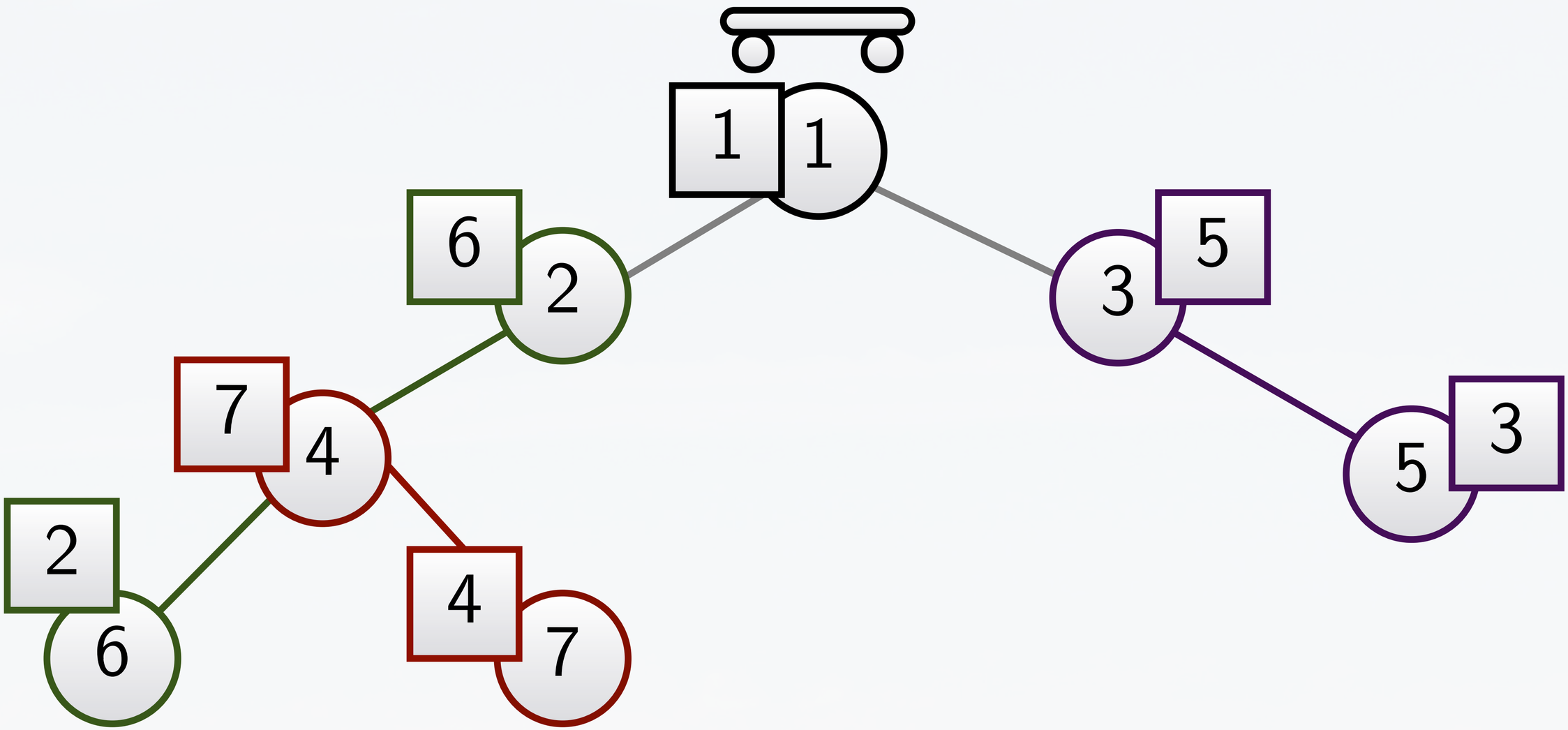


ETH zürich
Daniel Graf

# Sorting Problem

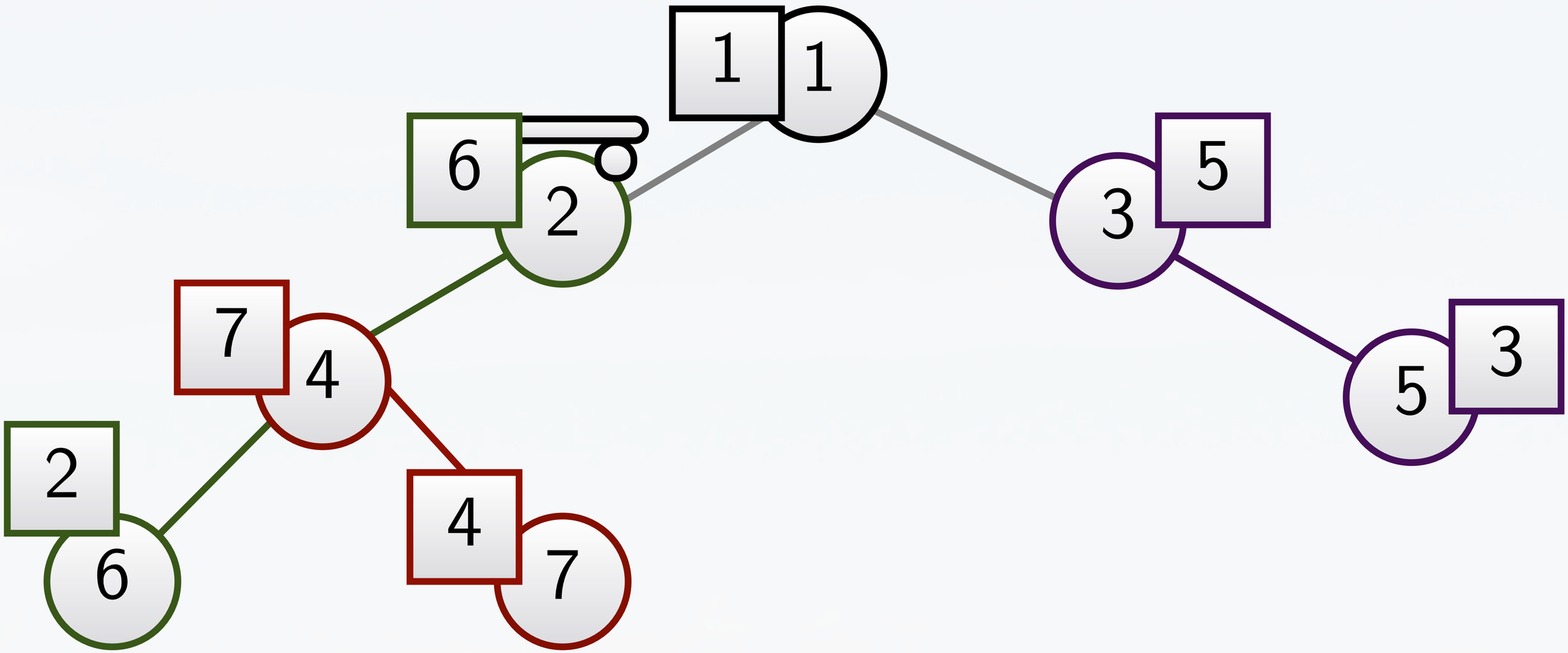
# Example





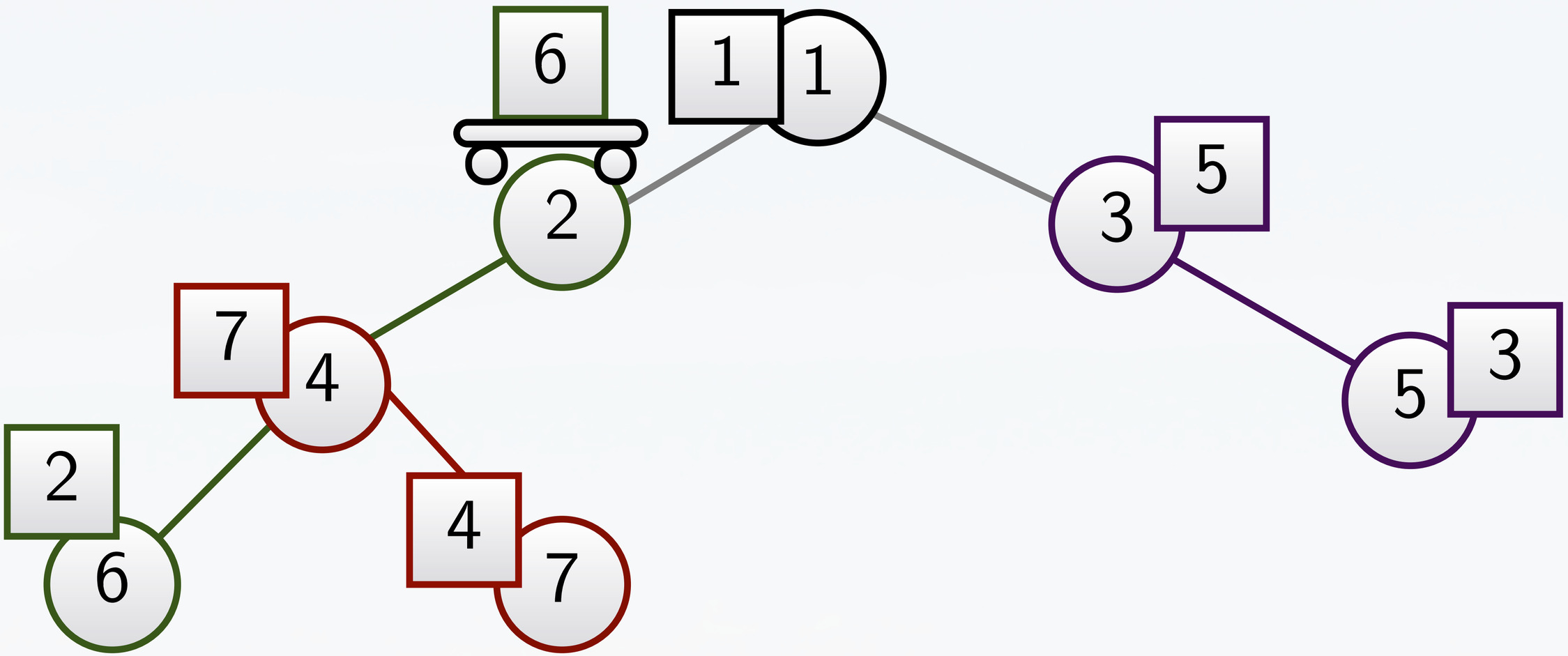
# Example





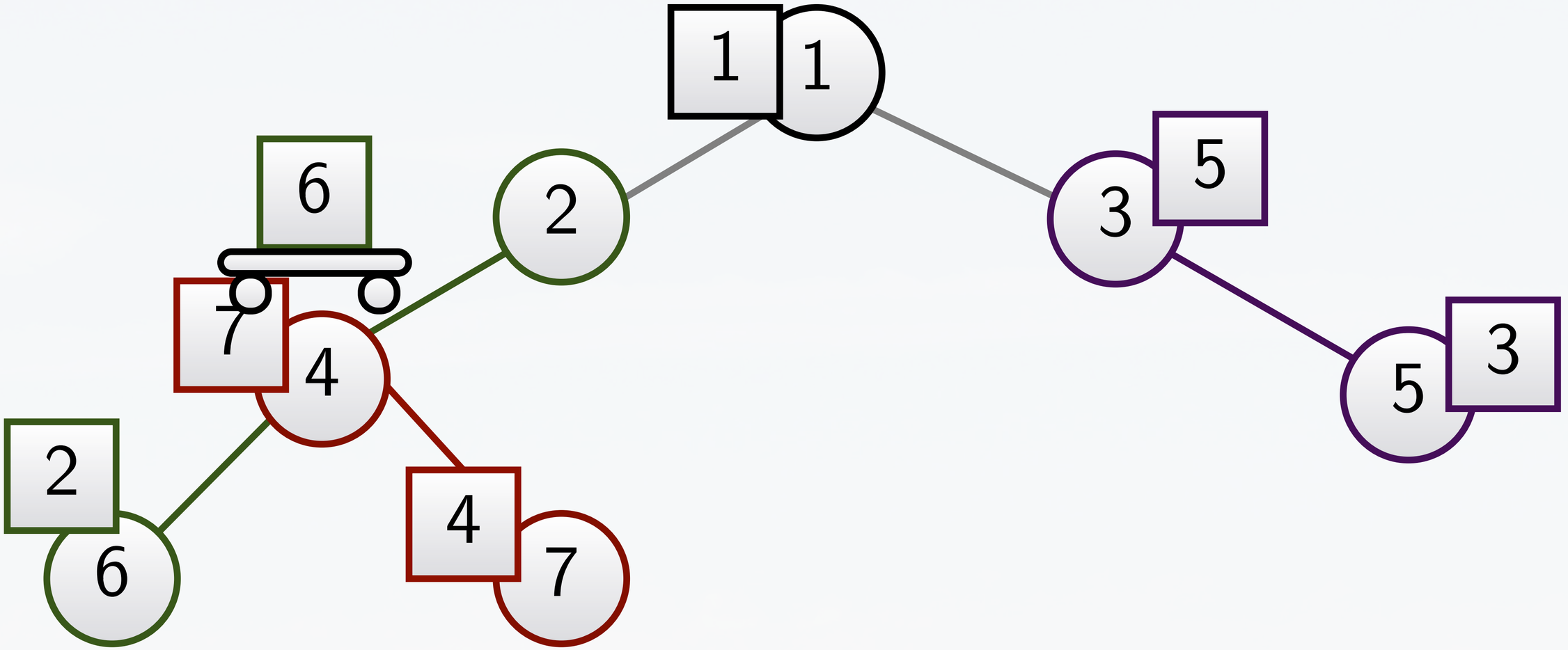
# Example





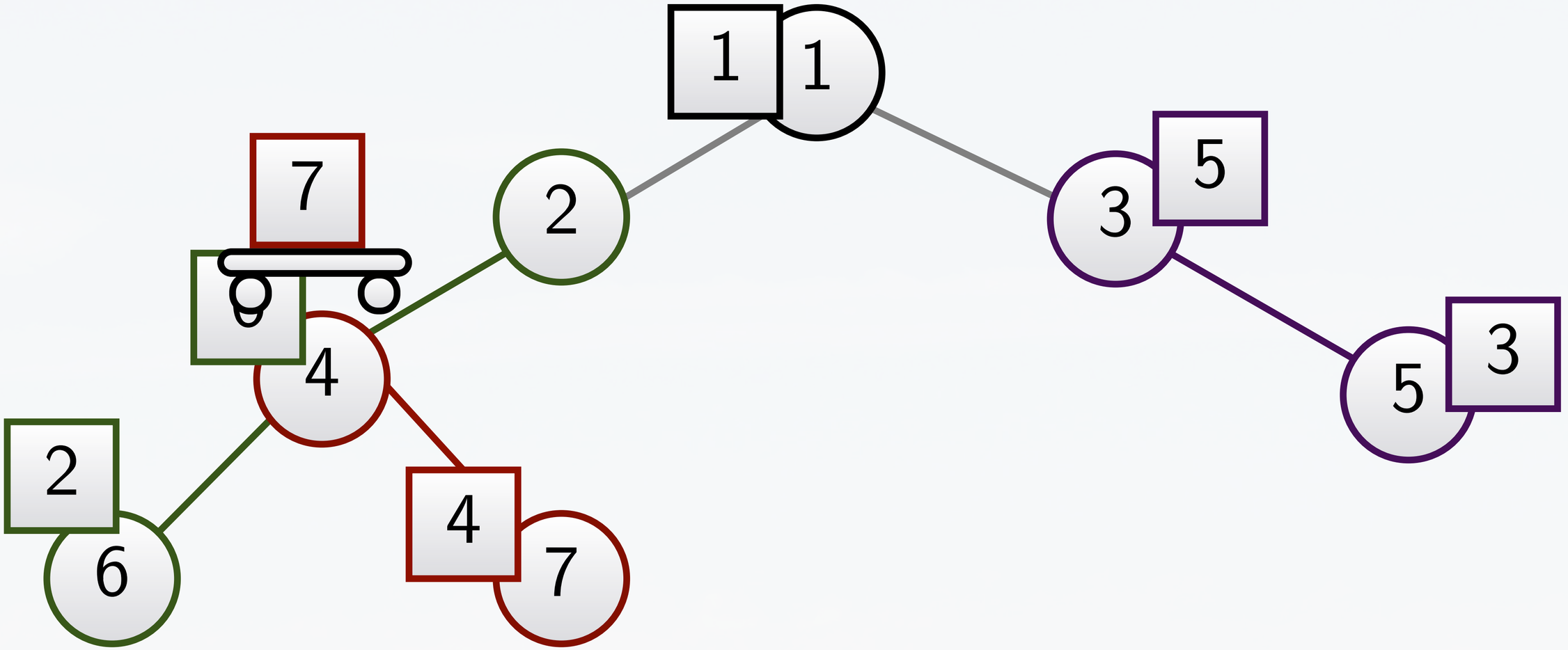
# Example





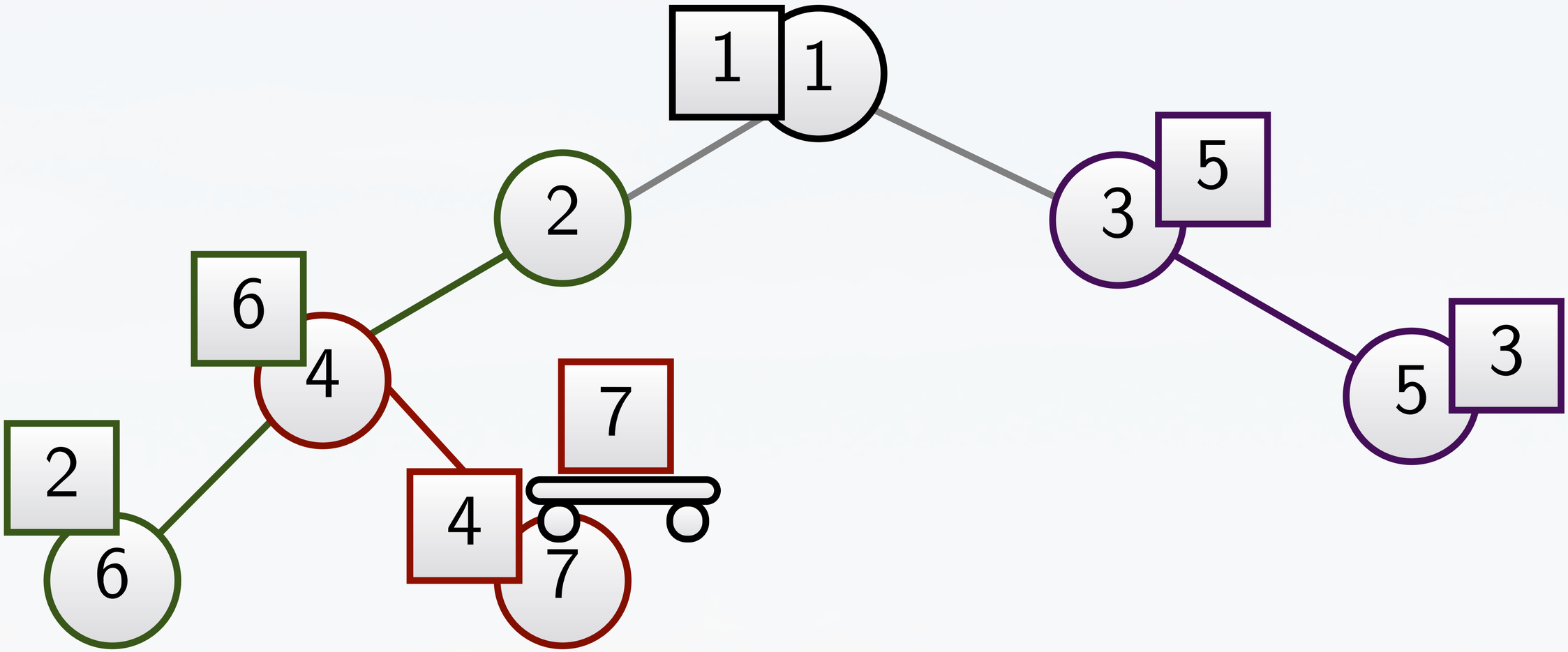
# Example





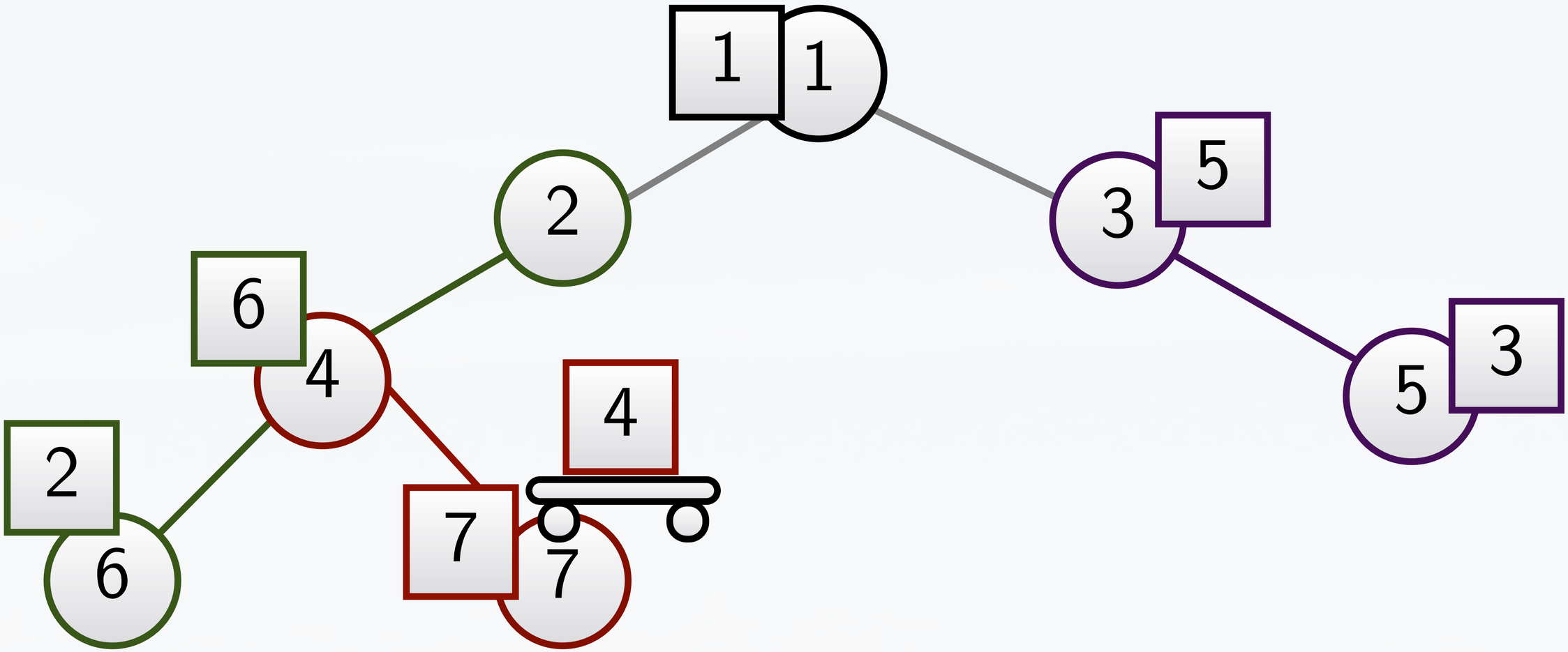
# Example





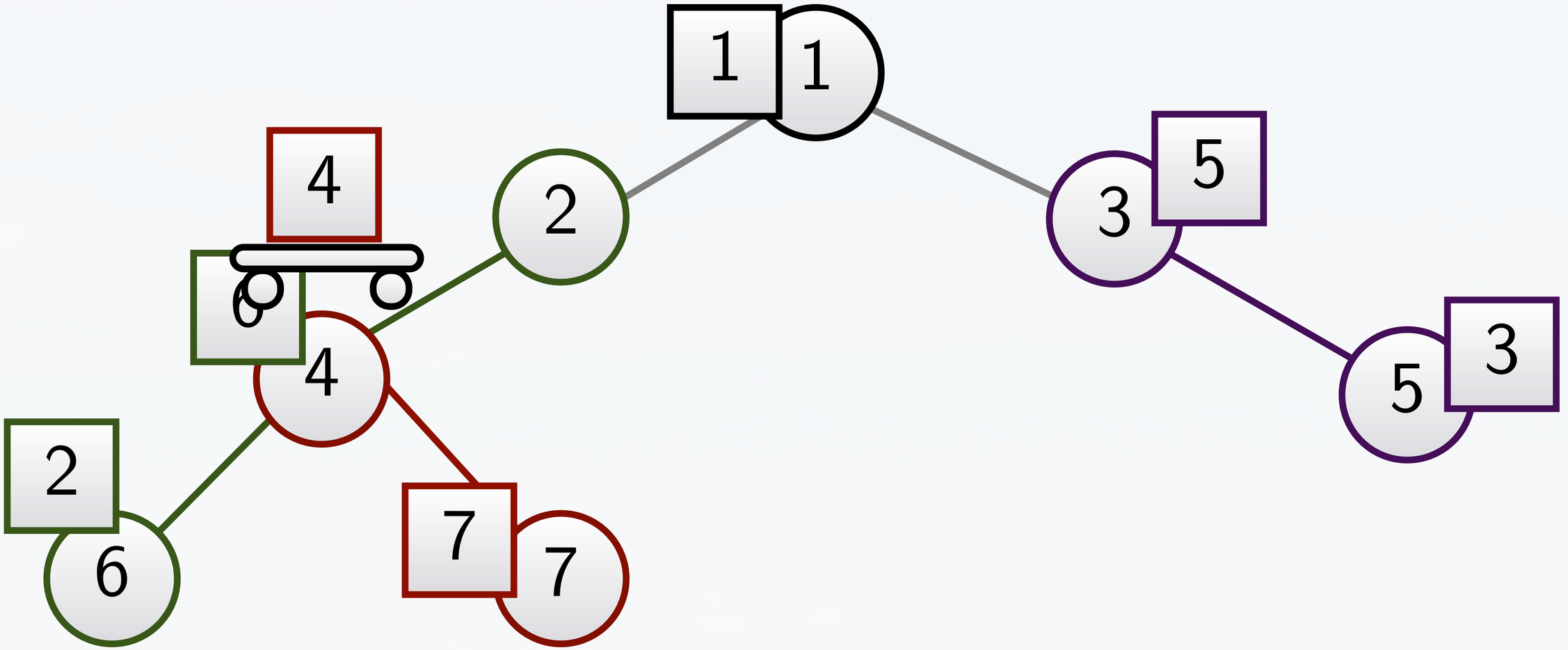
# Example





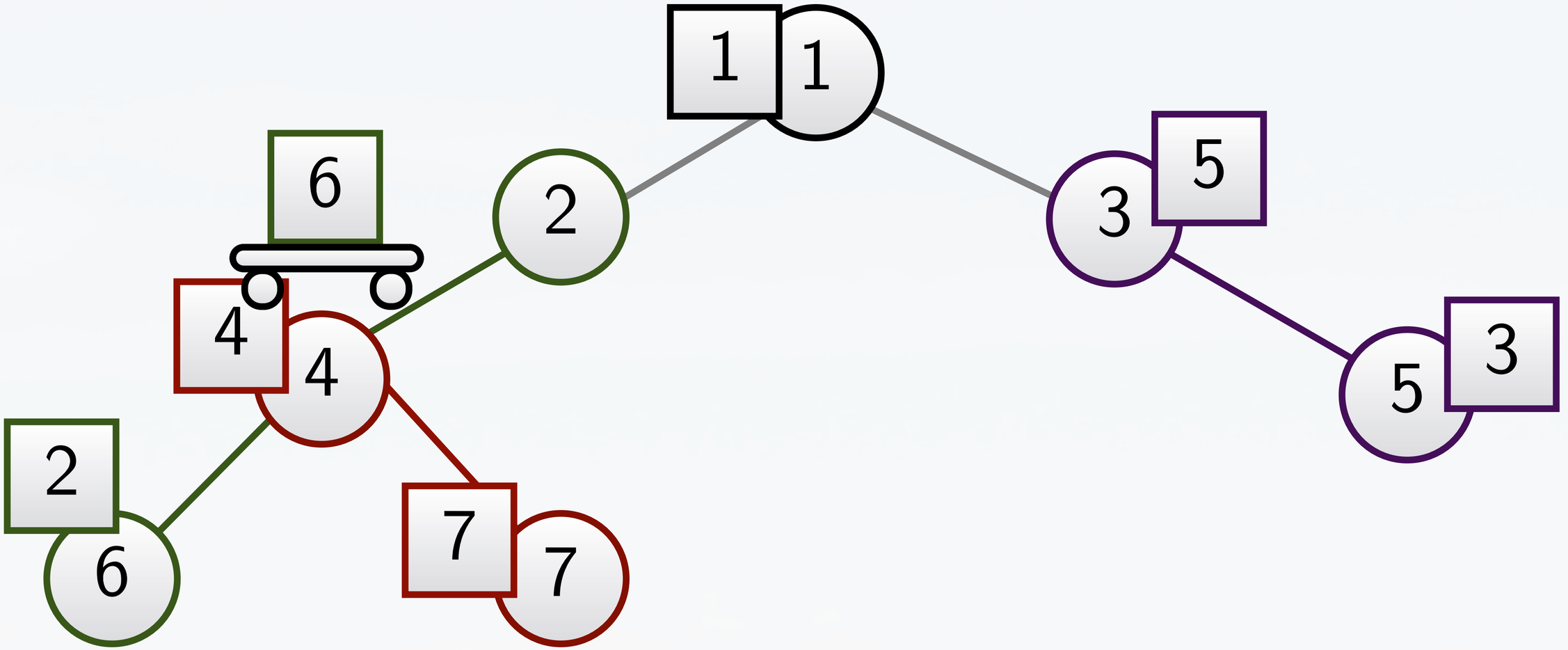
# Example





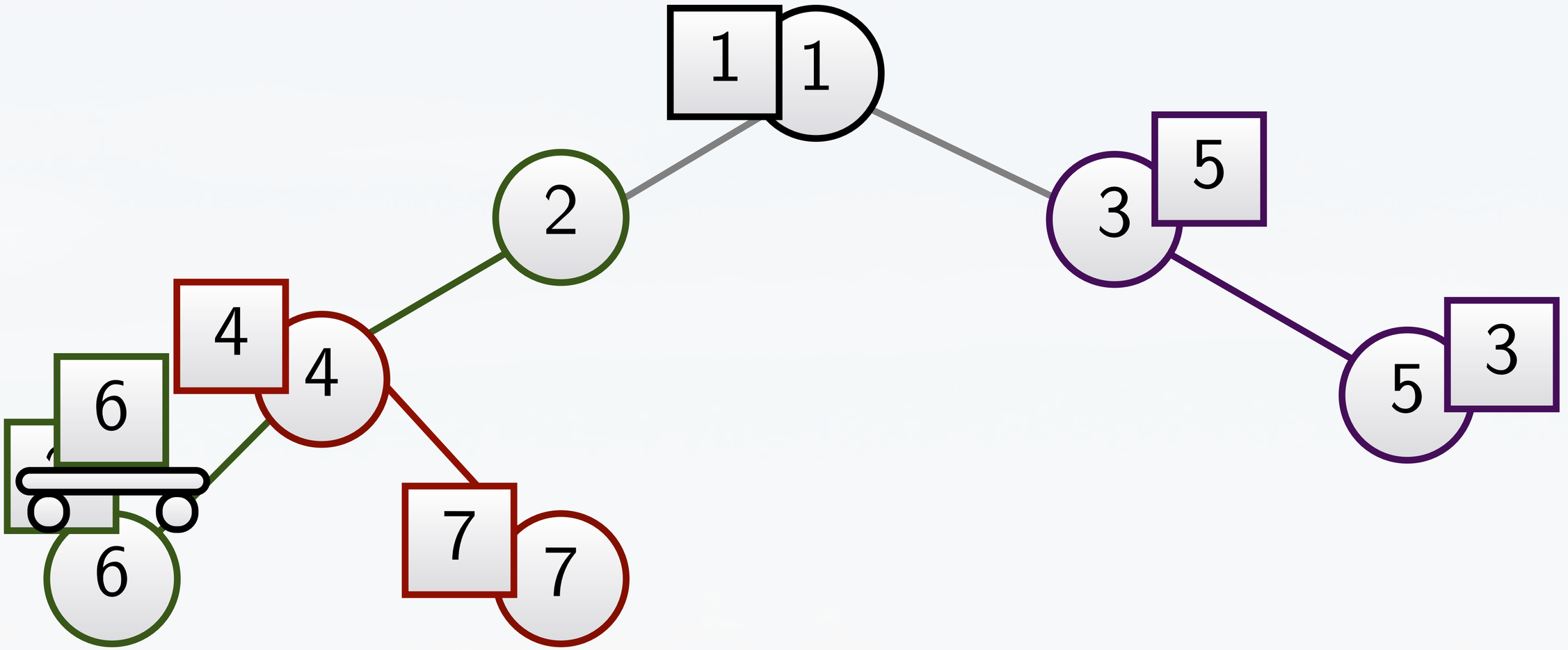
# Example





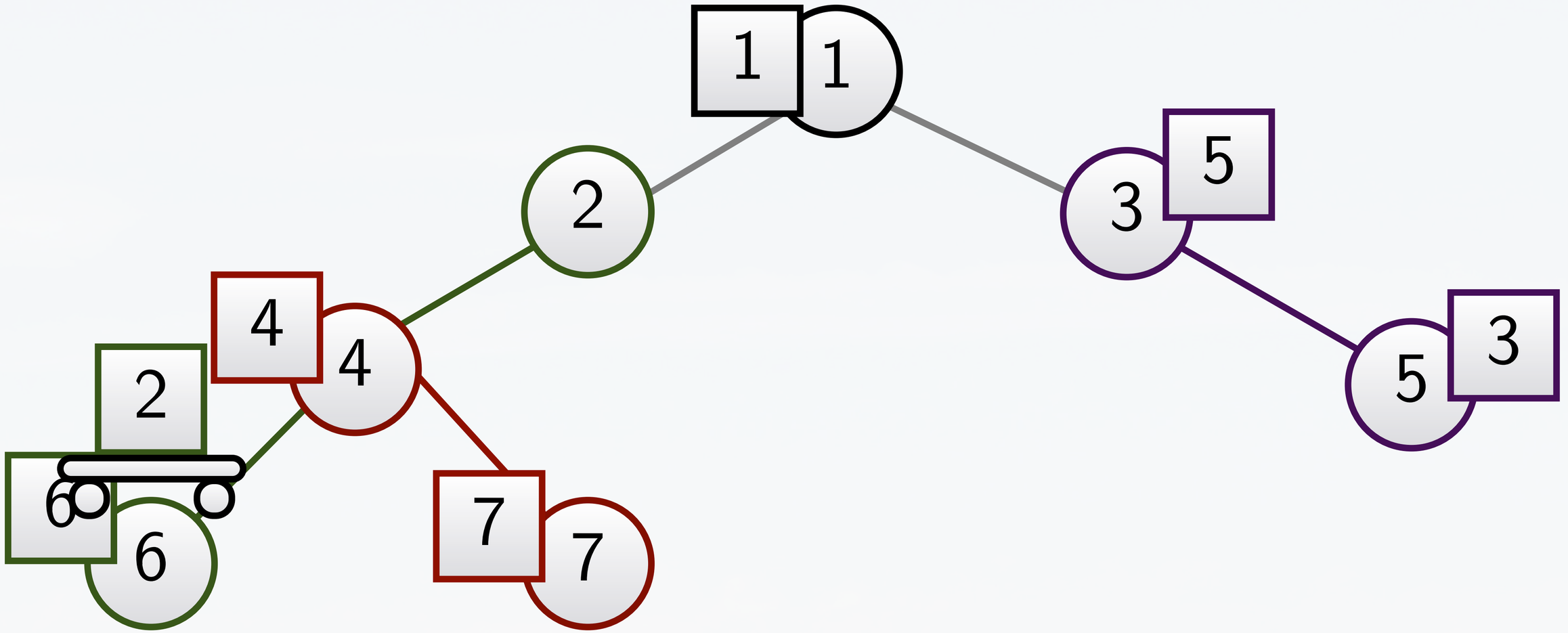
# Example





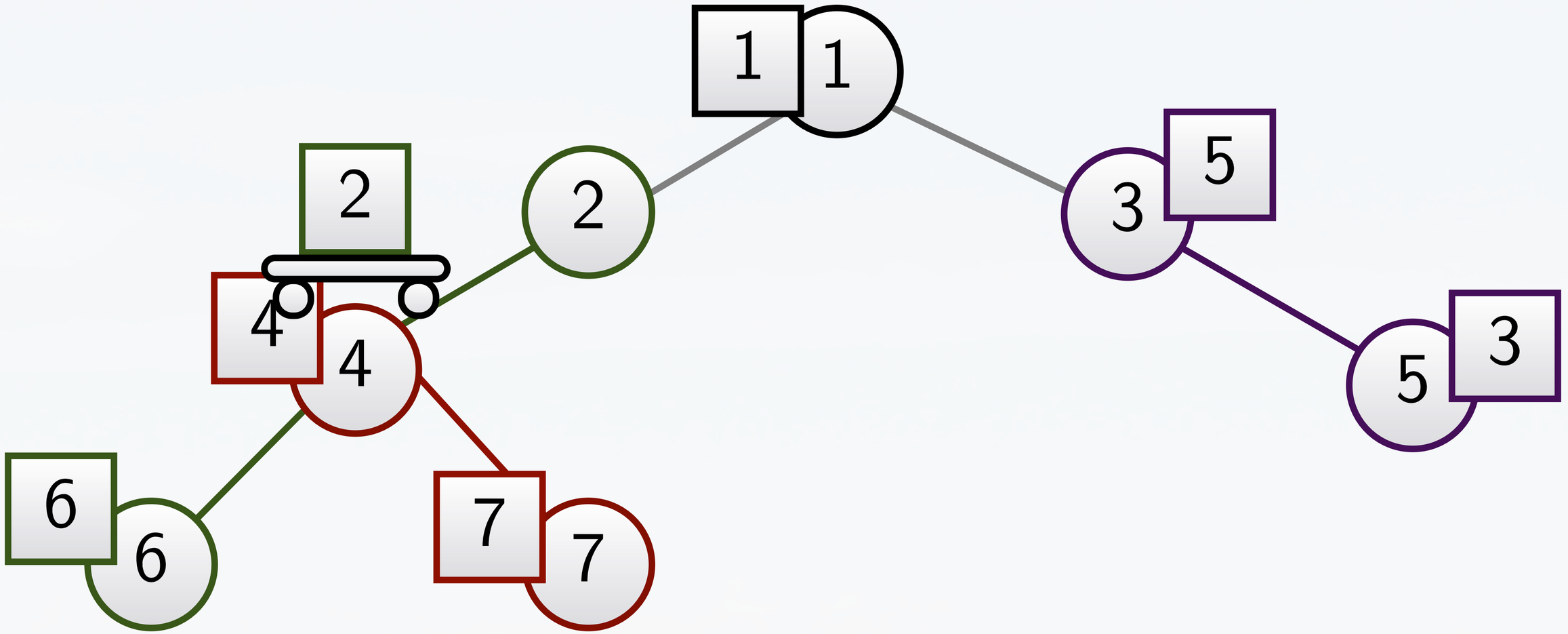
# Example





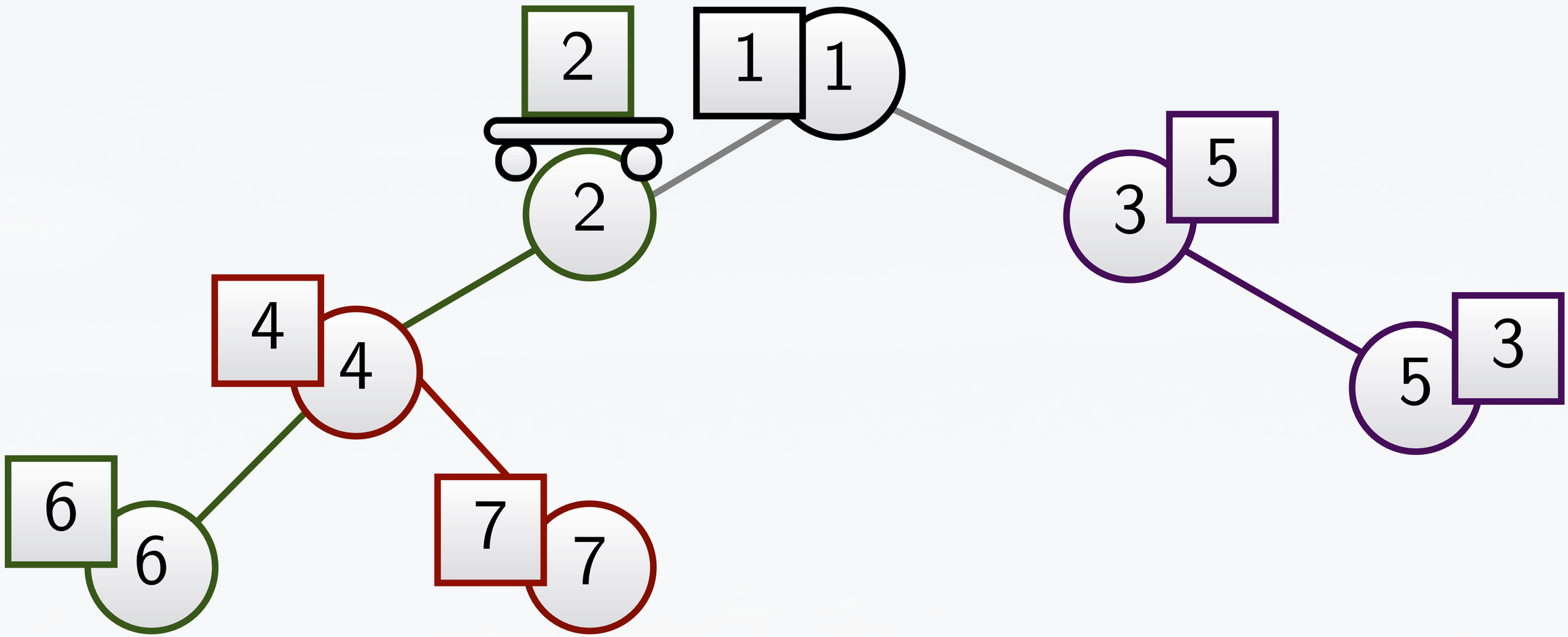
# Example





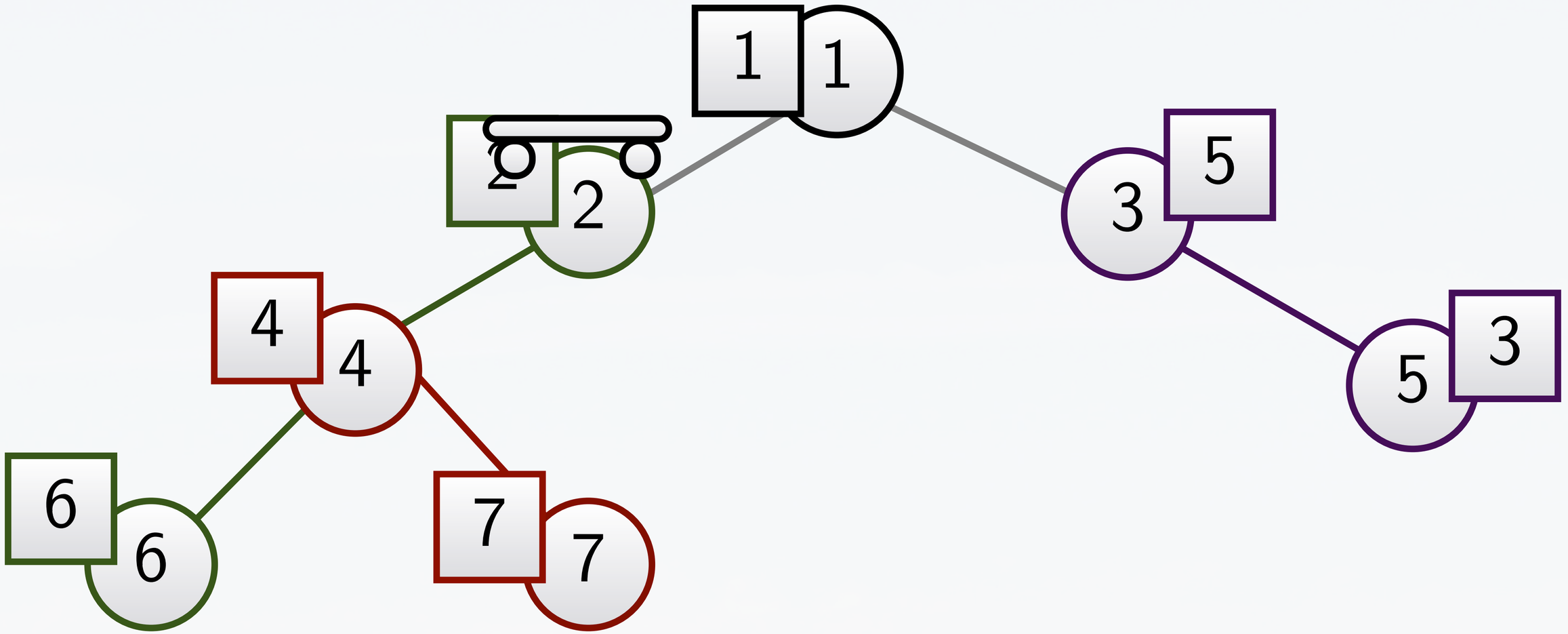
# Example





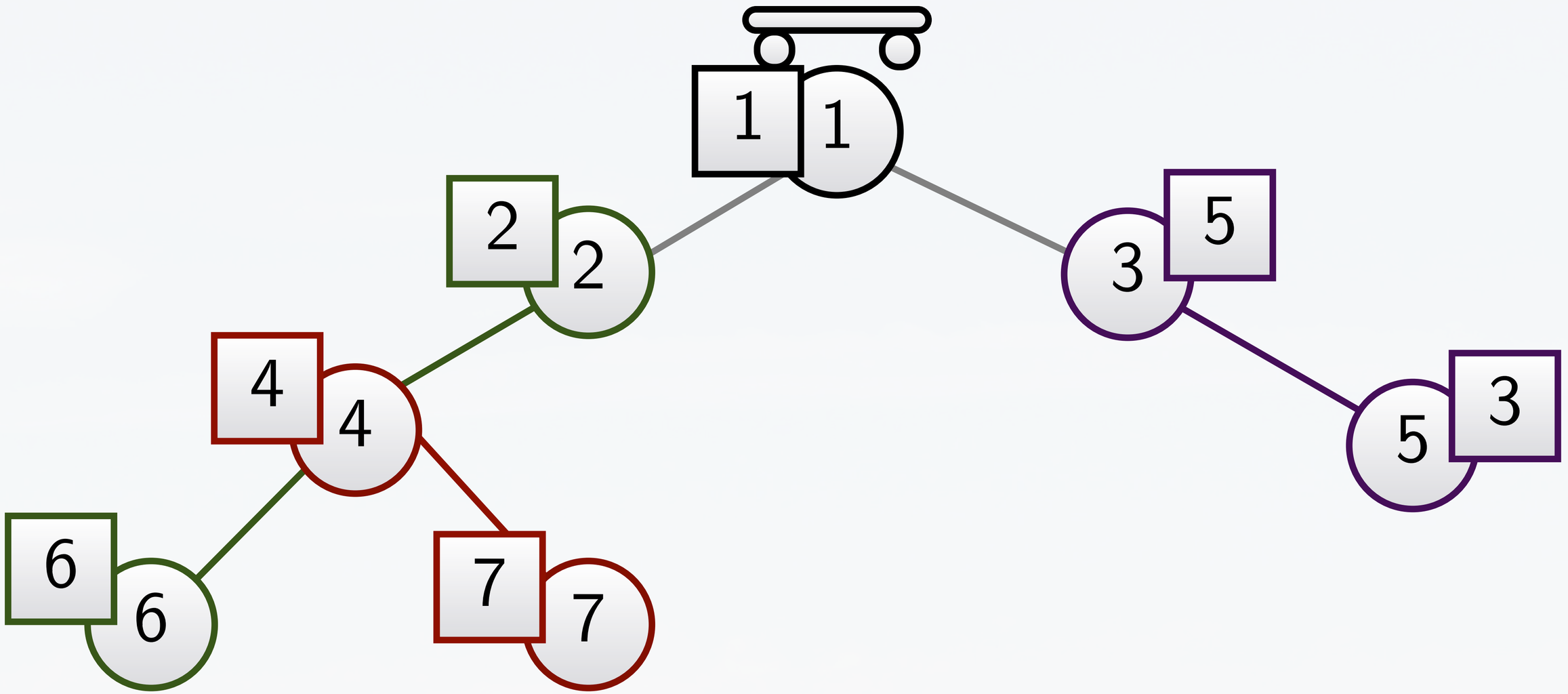
# Example





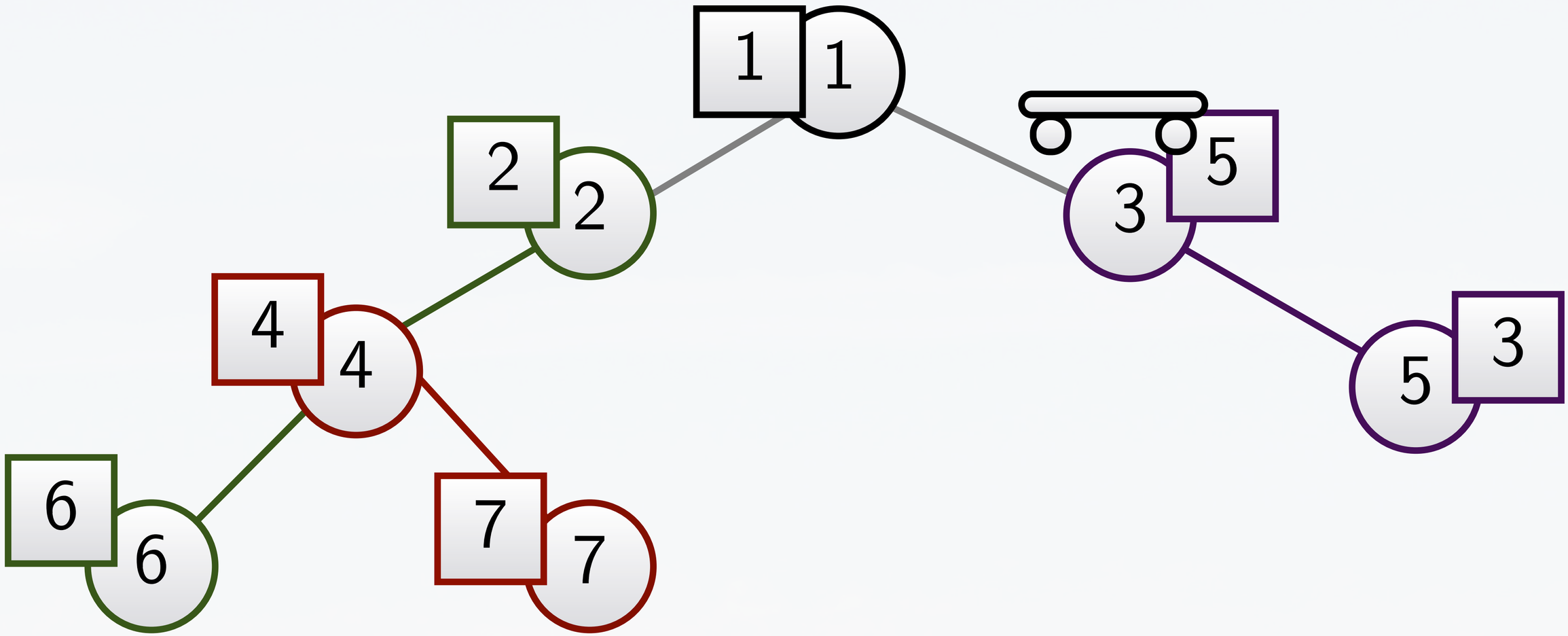
# Example





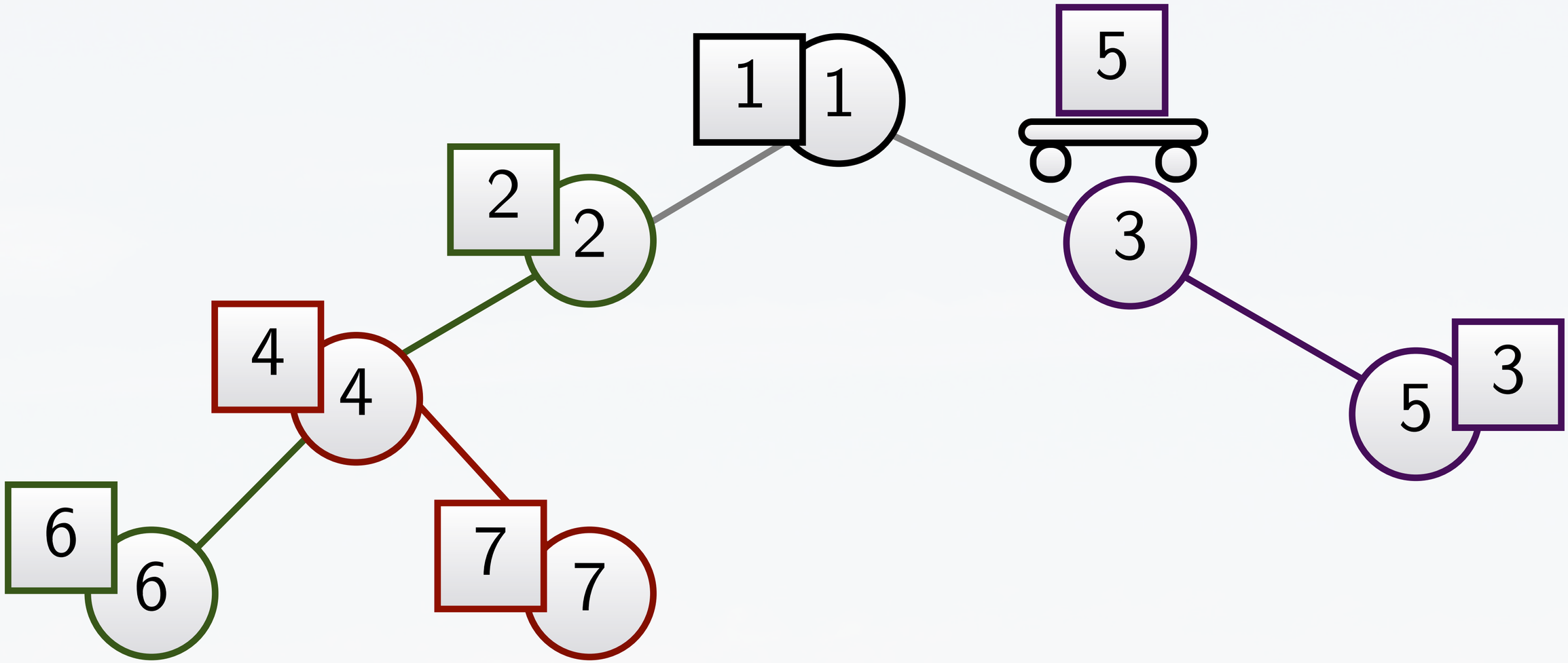
# Example





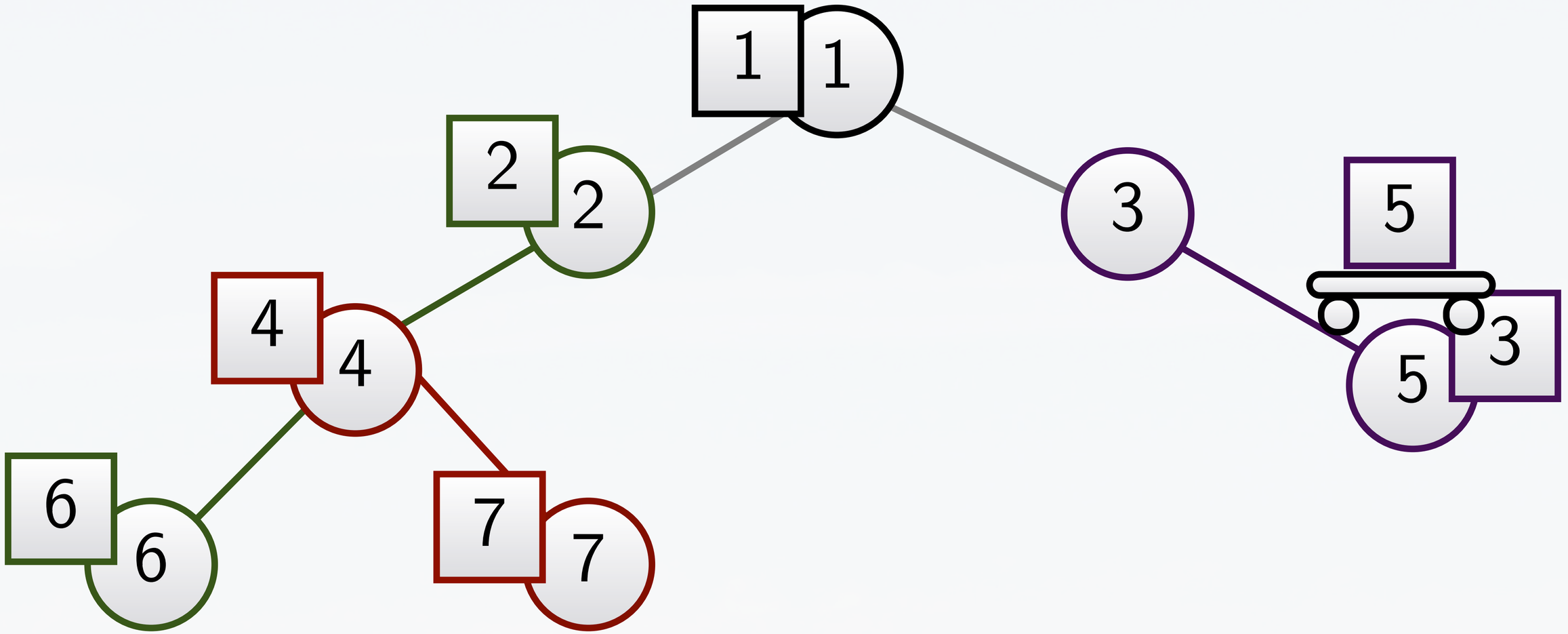
# Example





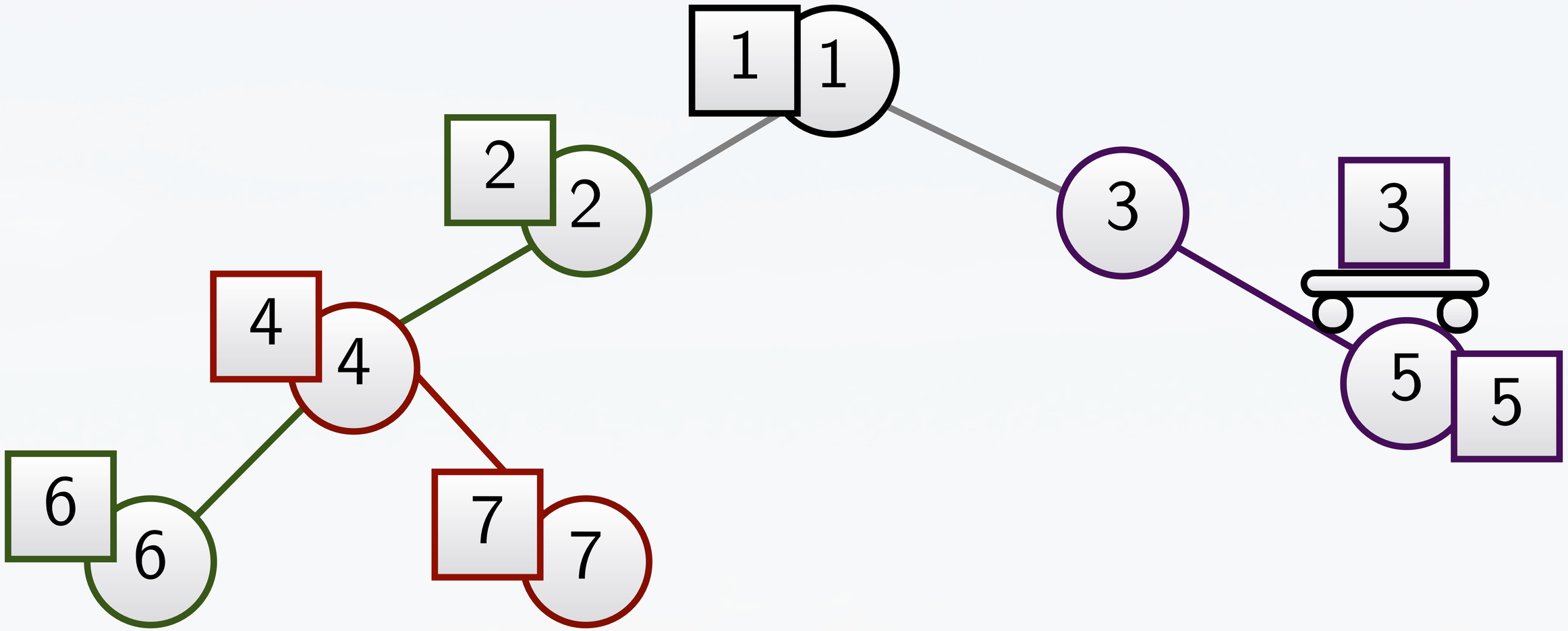
# Example





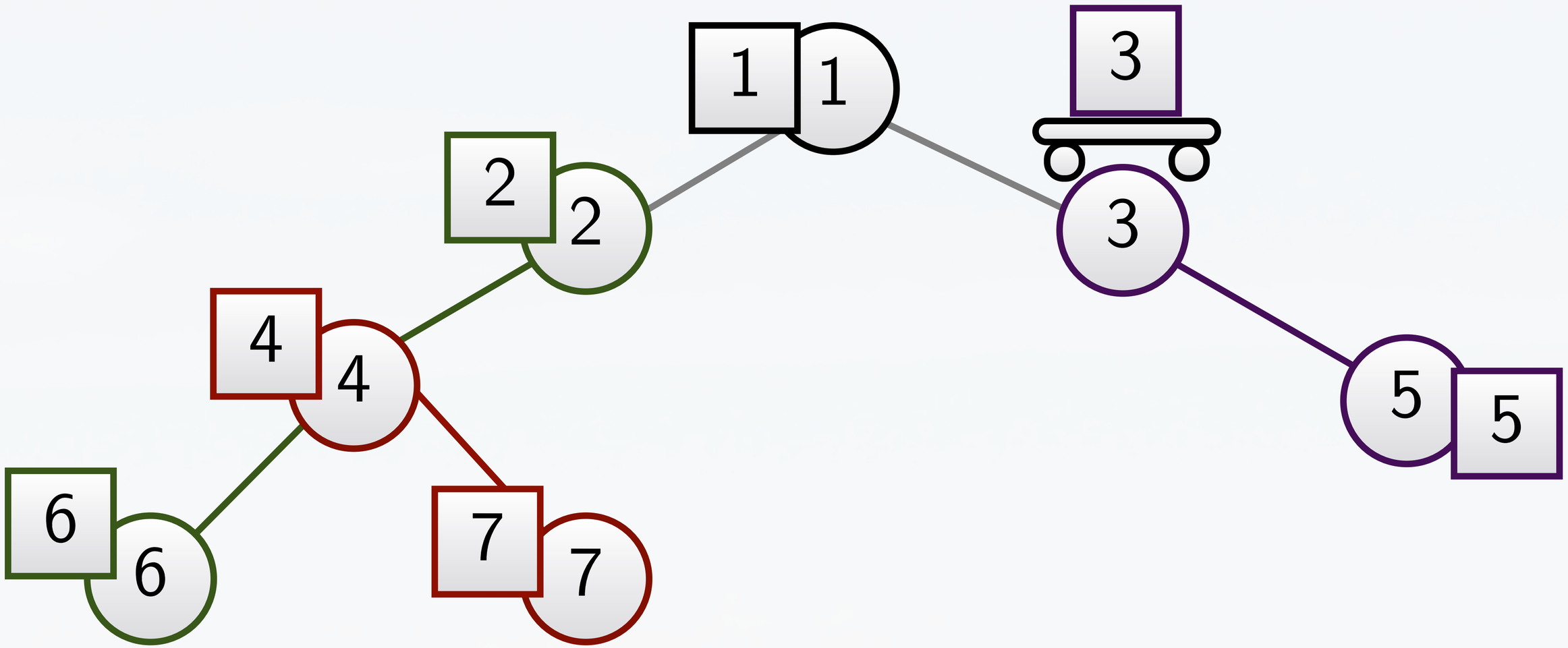
# Example





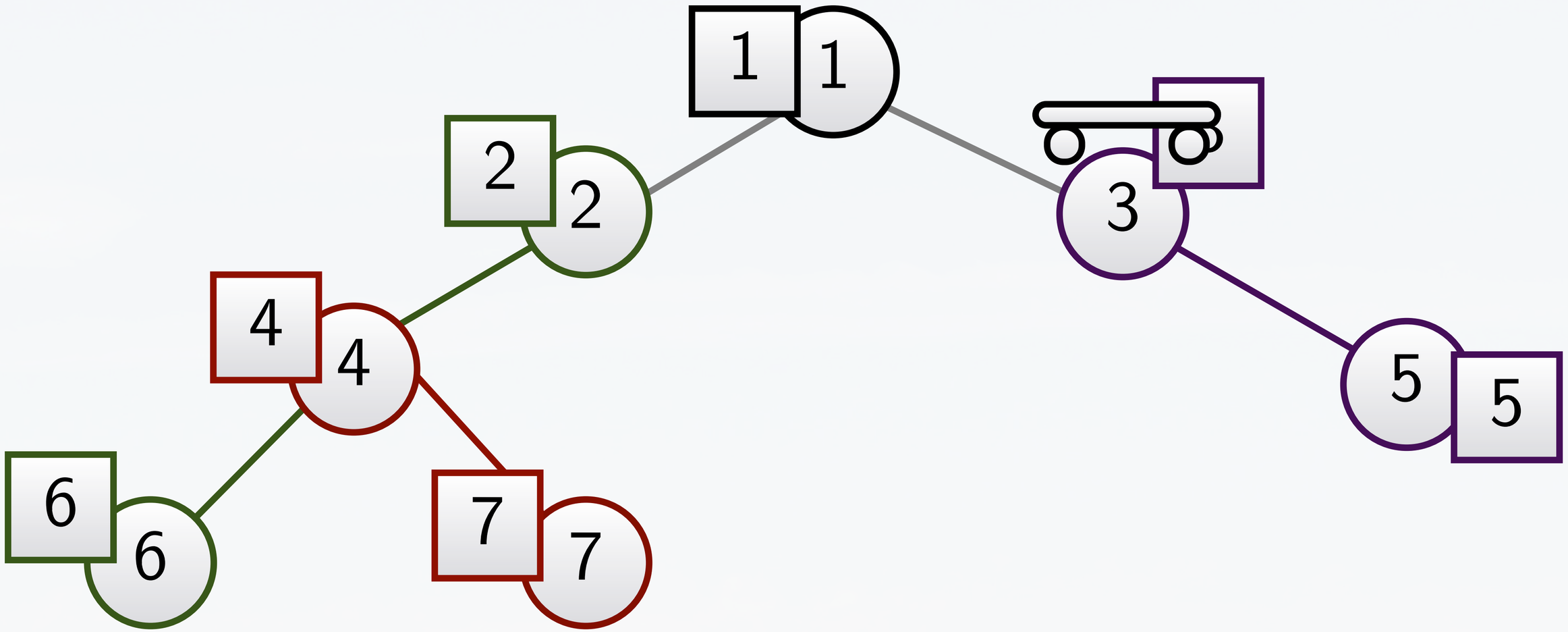
# Example





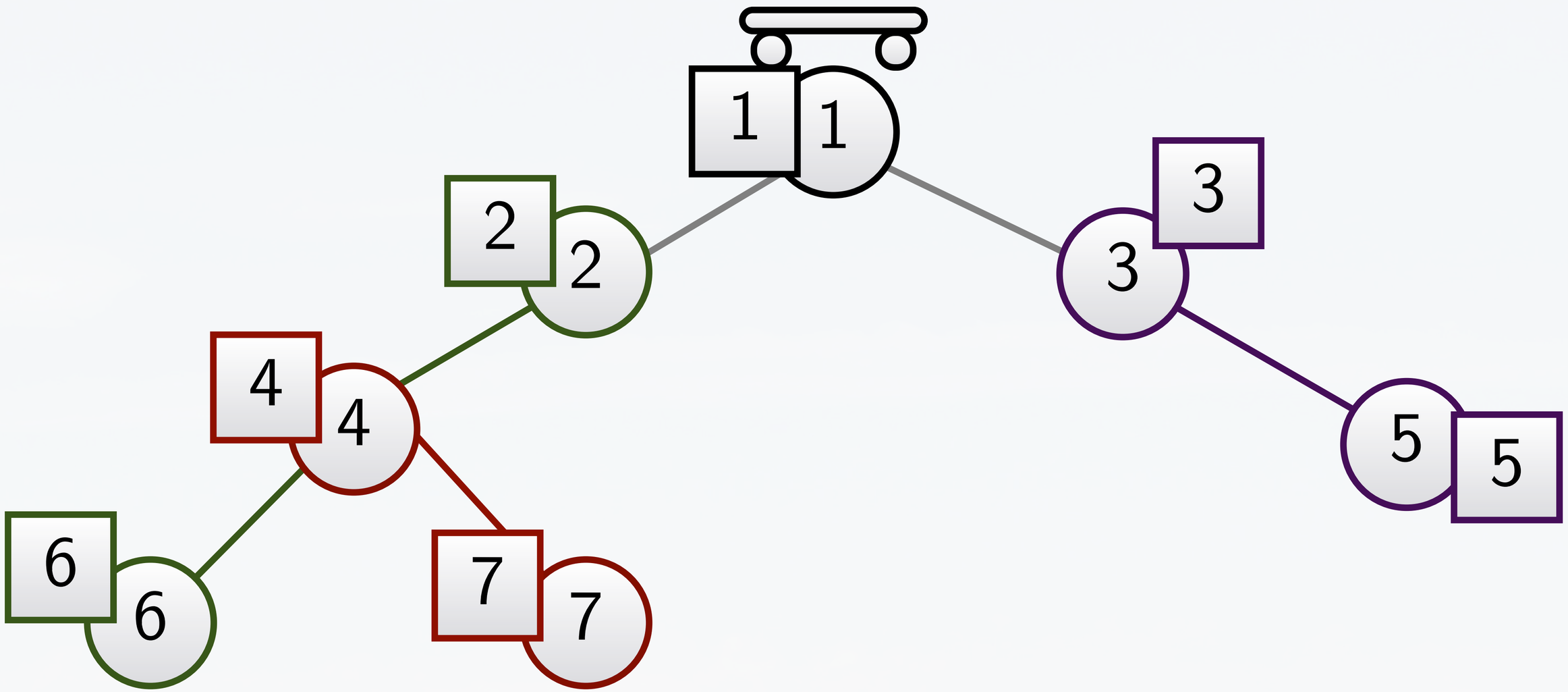
# Example





# Example





# General Bounds



*Definition:* ***essential step***

A step is essential, if it moves a box closer to its target position than it was in any previous state.

# General Bounds



*Definition:* ***essential step***

A step is essential, if it moves a box closer to its target position than it was in any previous state.

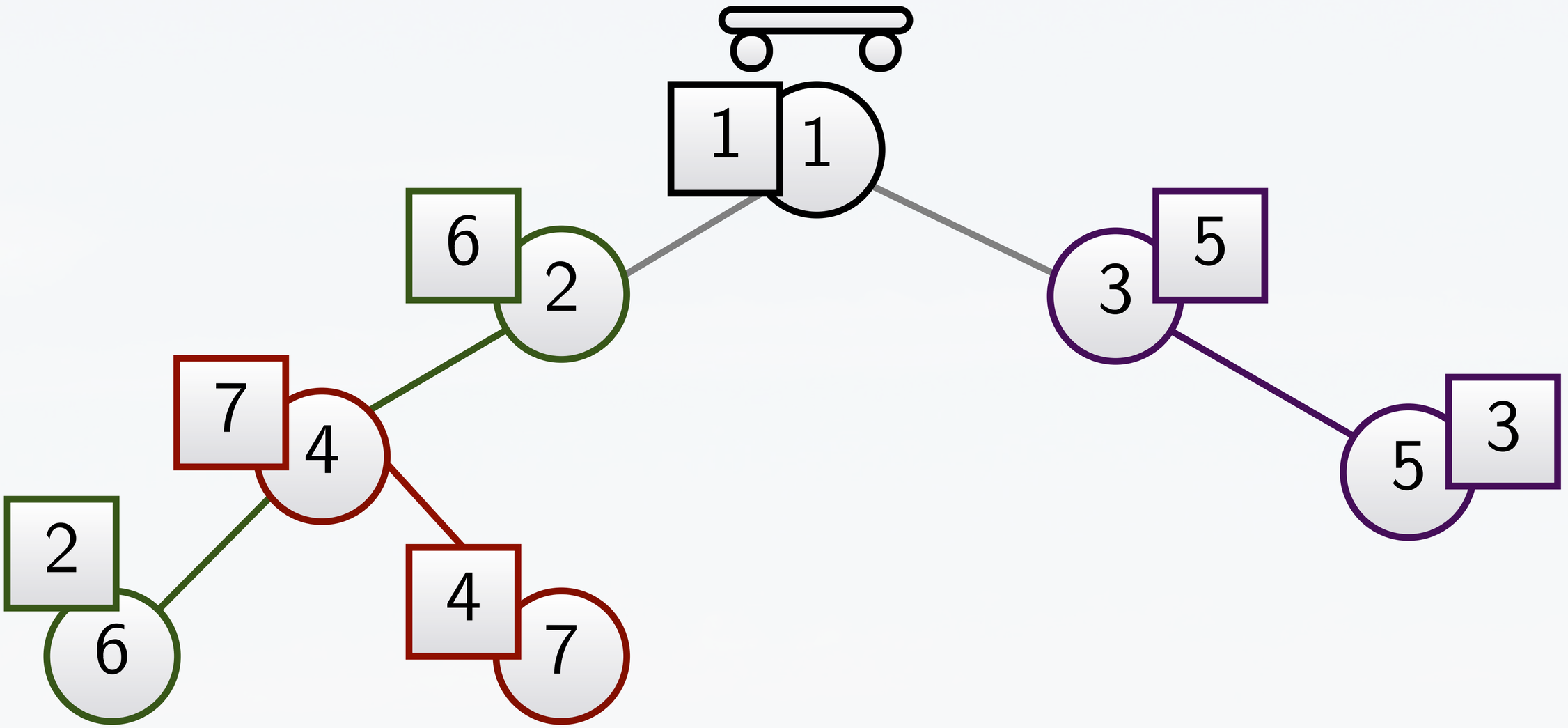
*Lemma: (Lower bound by essential steps)*

Every sorting walk contains at least $d(\pi) := \sum_{i=1}^{n} d(i, \pi(i))$ steps.

# General Bounds



$$d(\pi) = 1 + 1 + 1 + 1 + 2 + 2 = 8$$



# General Bounds



*Lemma: (Upper bound using depth first search)*
We can always find a shorting walk of length $d(\pi)+(2n-2)$.

# General Bounds



*Lemma: (Upper bound using depth first search)*
We can always find a shorting walk of length $d(\pi)+(2n-2)$.

*Proof:*

- We sort cycle after cycle of the permutation $\pi$.
- A DFS traversal of $G$ reaches every cycle of $\pi$.

# General Bounds



*Lemma: (Upper bound using depth first search)*
We can always find a shorting walk of length $d(\pi)+(2n-2)$.

*Proof:*

- We sort cycle after cycle of the permutation $\pi$.
- A DFS traversal of $G$ reaches every cycle of $\pi$.

*Remaining:* How can we interleave the cycles to close this linear gap between the bounds?

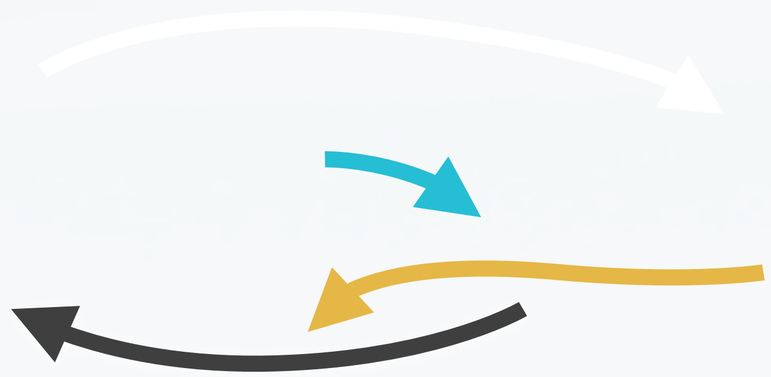
ETH zürich
Daniel Graf

# On Paths



Path graph with the robot starting at the border.

Straightforward if they form a single cycle.



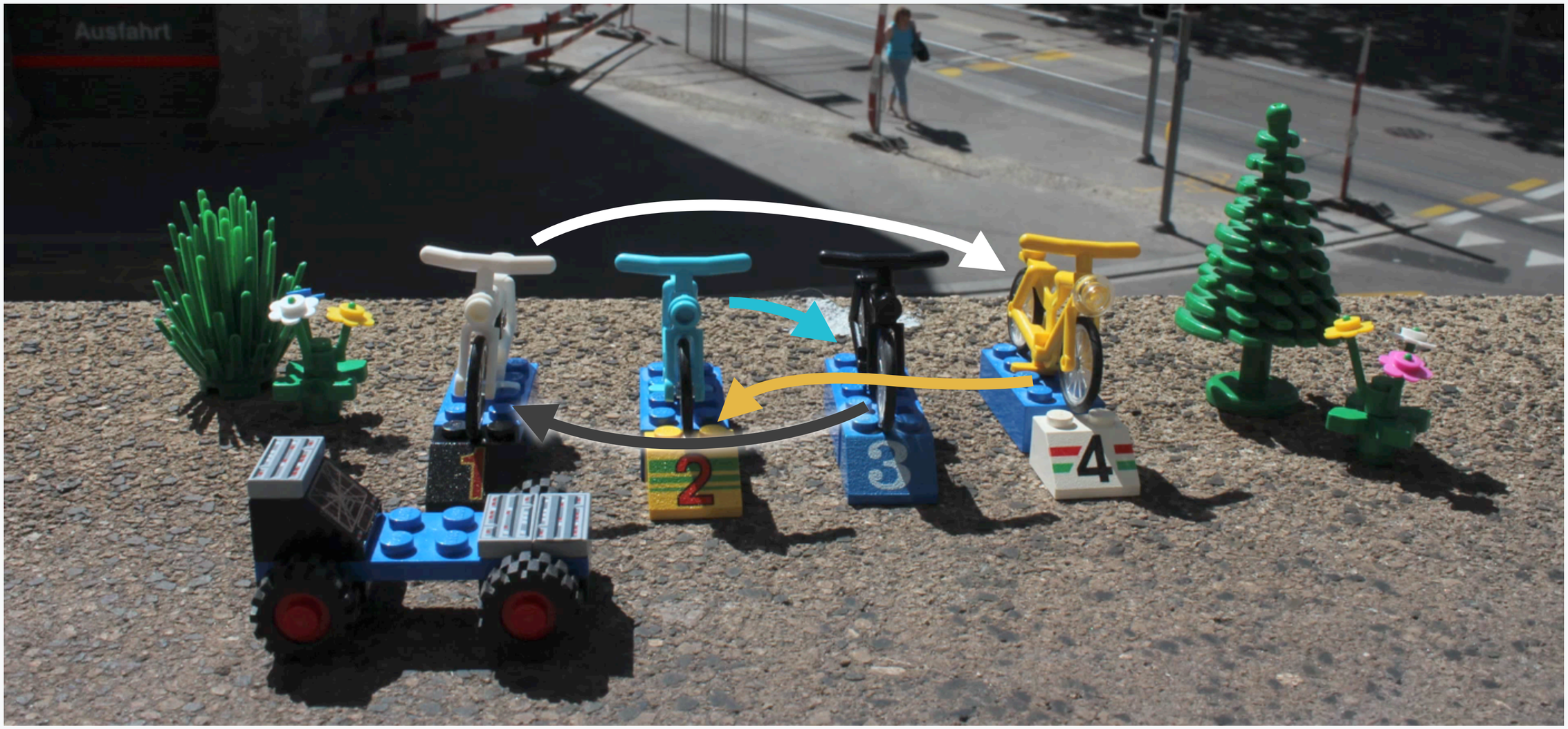
ETH zürich
Daniel Graf

# On Paths



Path graph with the robot starting at the border.

Straightforward if they form a single cycle.



ETH zürich
Daniel Graf

# On Paths



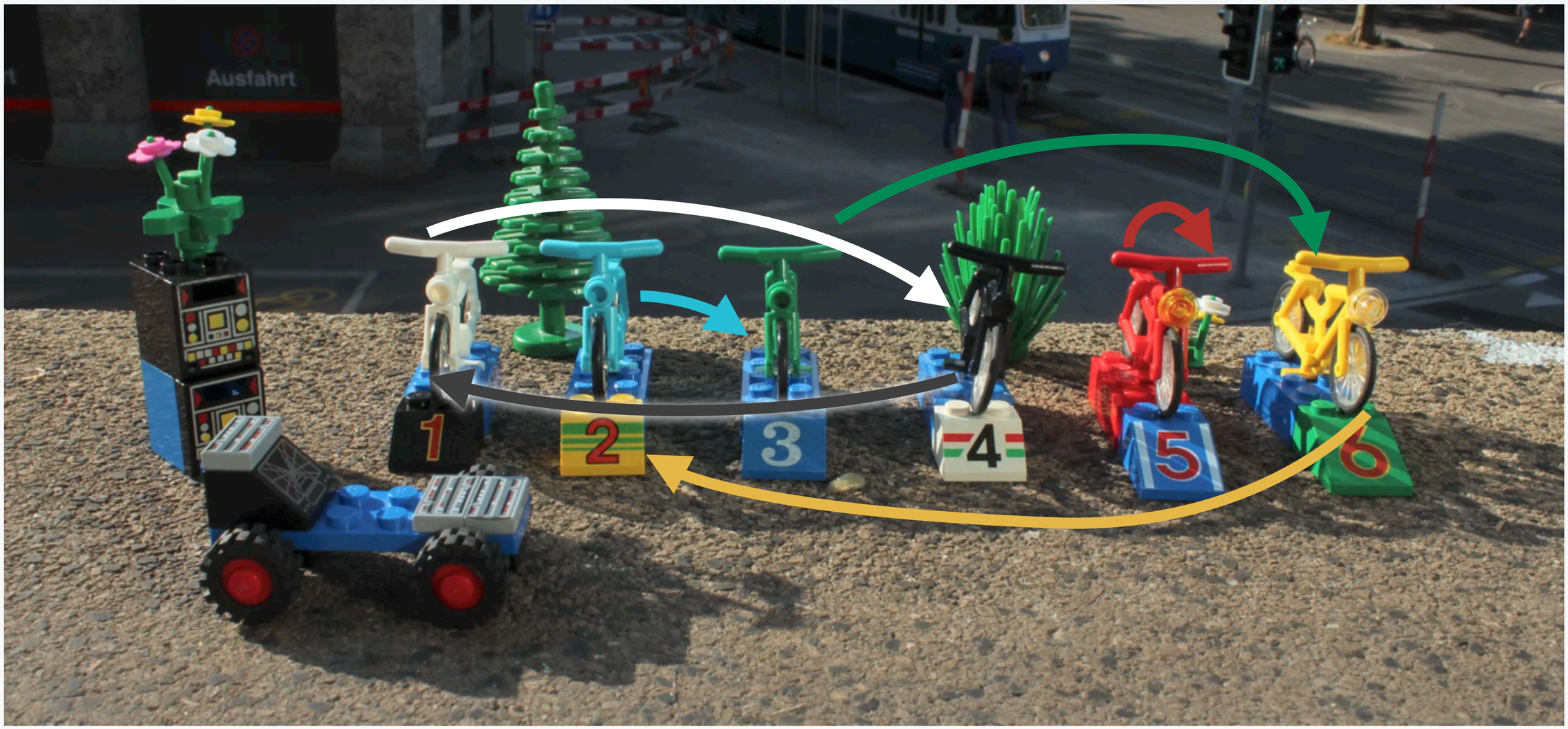
*Observation:* Overlapping cycles can be sorted without non-essential steps.

ETH zürich
Daniel Graf

# On Paths



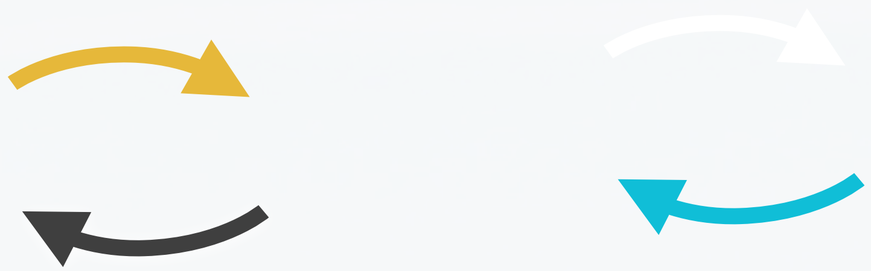
*Observation:* Overlapping cycles can be sorted without non-essential steps.



# On Paths



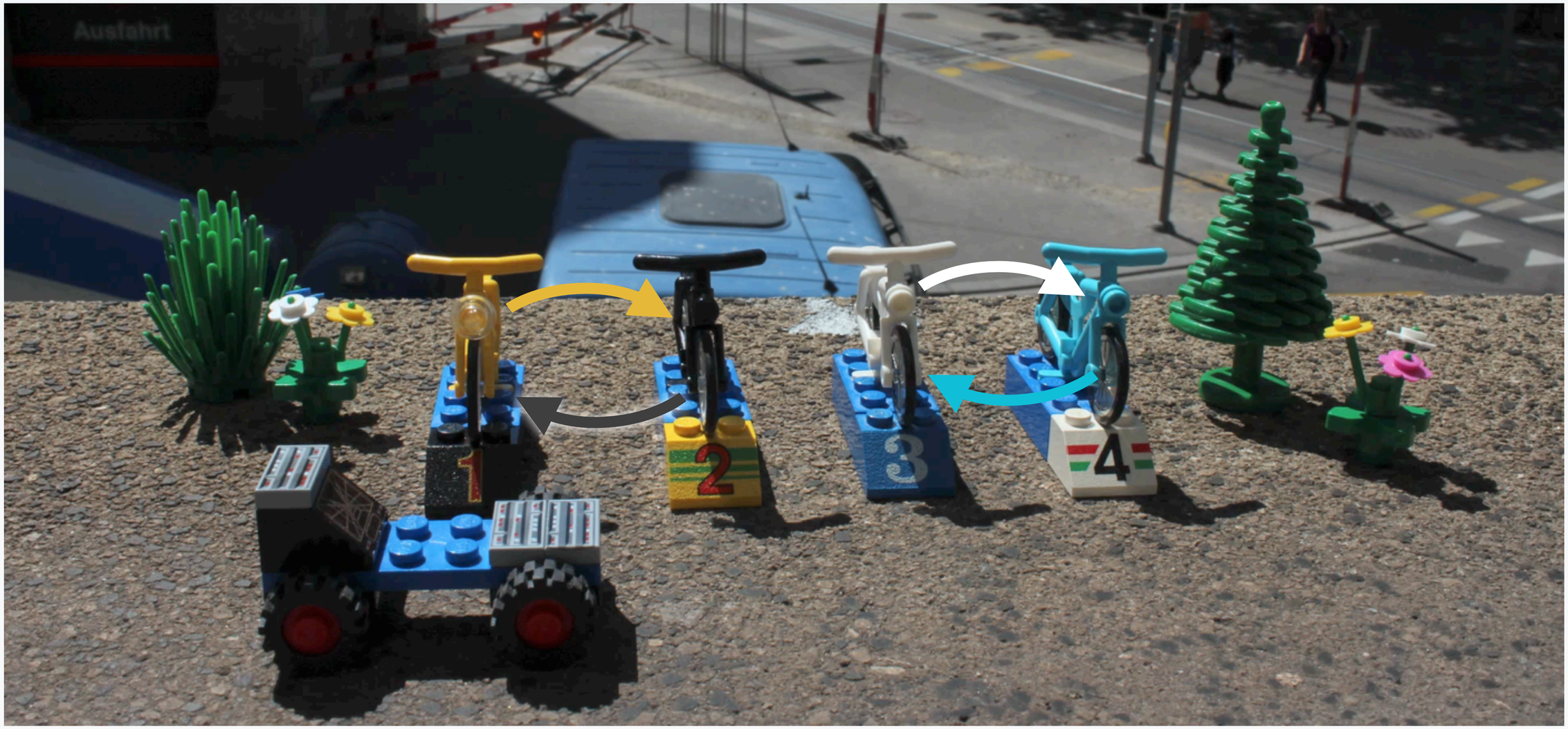
*Observation:* Non-essential steps are only needed across edges that are between cycles of $\pi$.



# On Paths



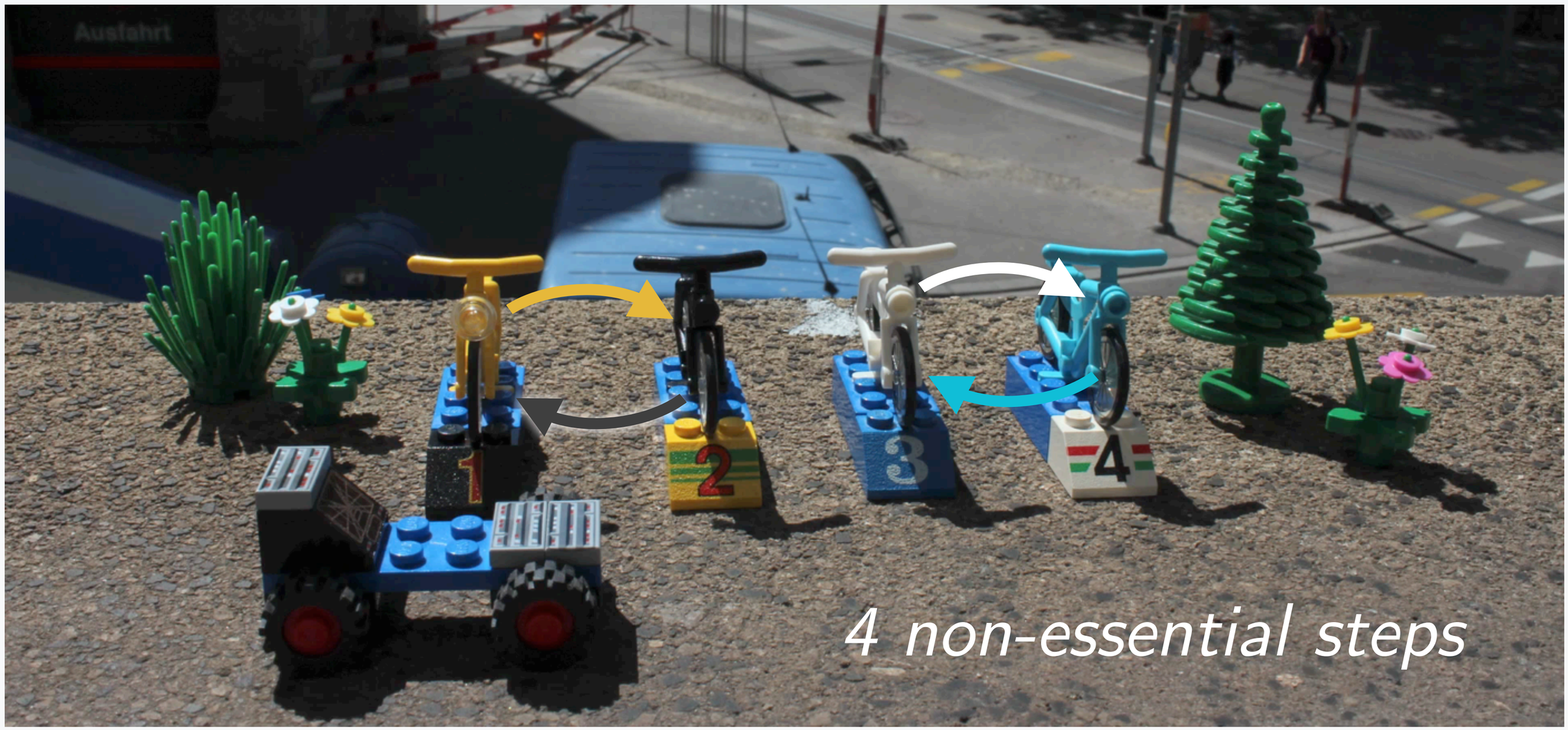
*Observation:* Non-essential steps are only needed across edges that are between cycles of $\pi$.



# On Paths



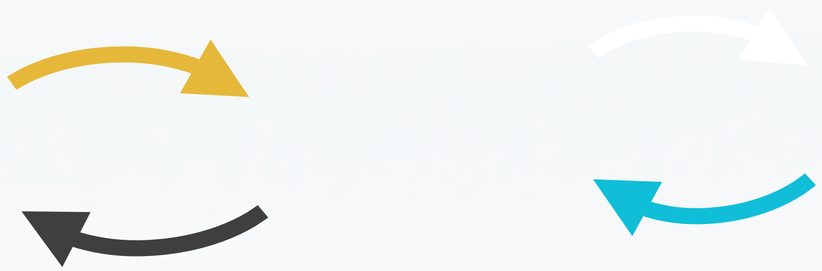
*Observation:* Non-essential steps are only needed across edges that are between cycles of $\pi$.



# On Paths



*Observation:* Non-essential steps are only needed across edges that are between cycles of $\pi$.

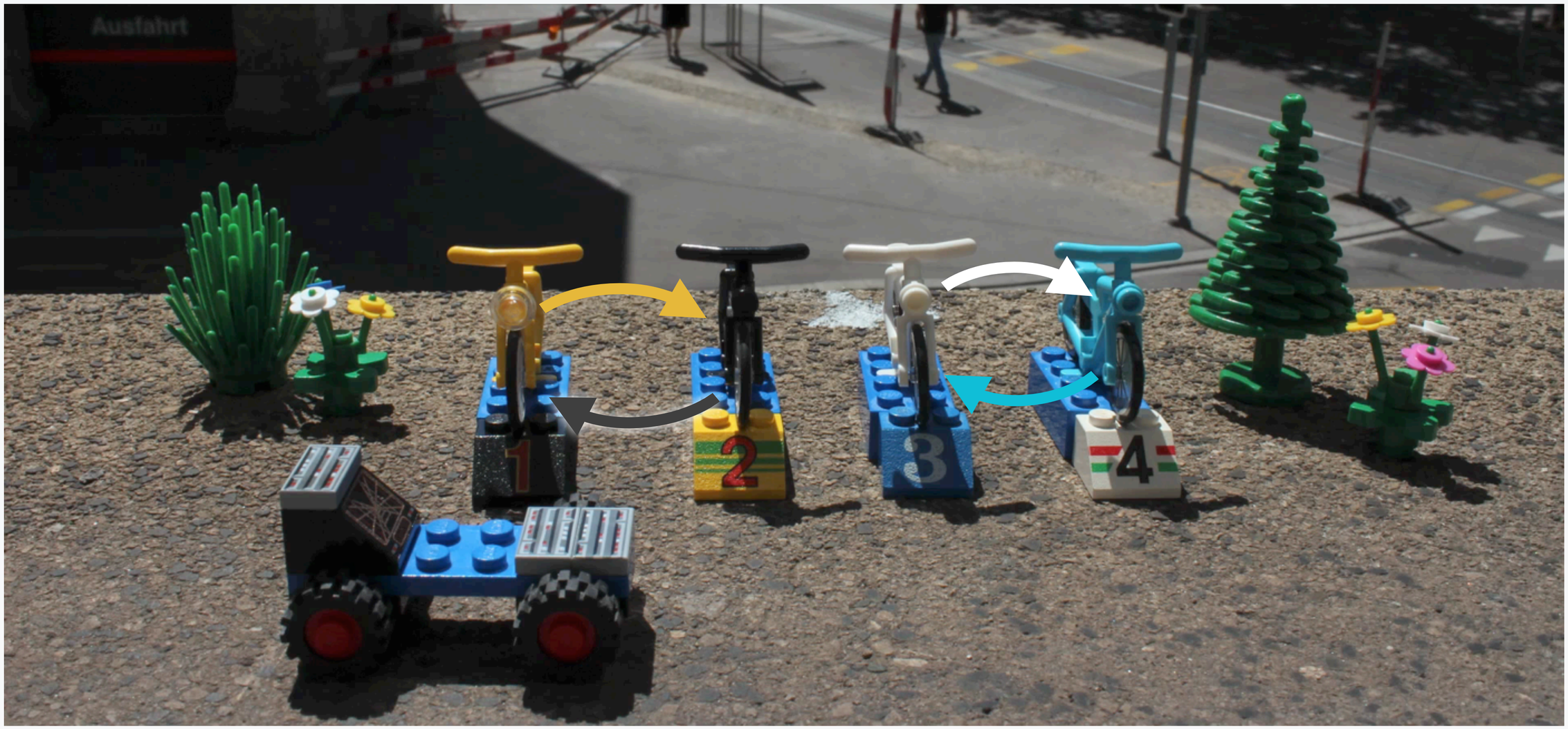


ETH zürich
Daniel Graf

# On Paths



*Observation:* Non-essential steps are only needed across edges that are between cycles of $\pi$.

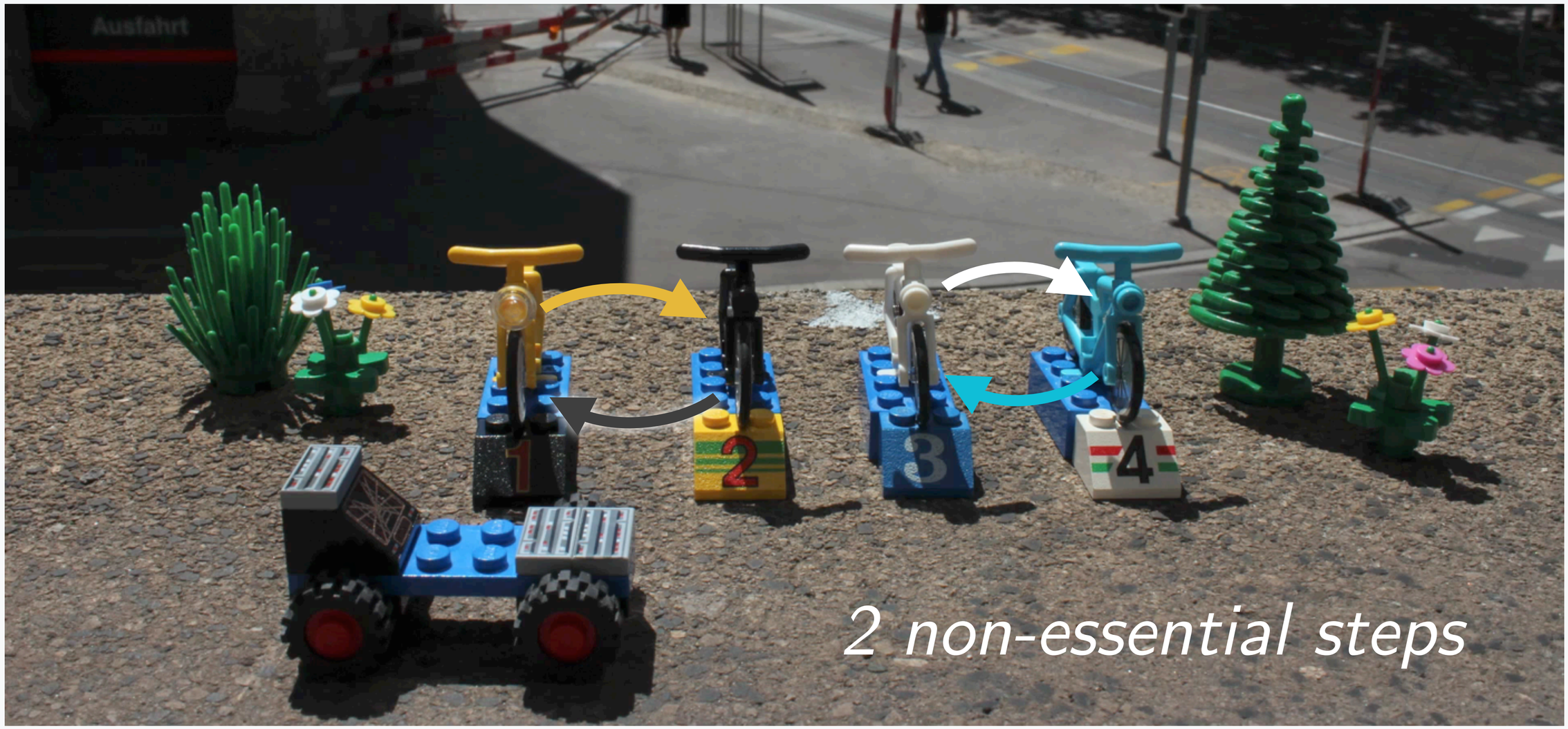


ETH zürich
Daniel Graf

# On Paths



*Observation:* Non-essential steps are only needed across edges that are between cycles of $\pi$.



ETH zürich
Daniel Graf

# On Paths



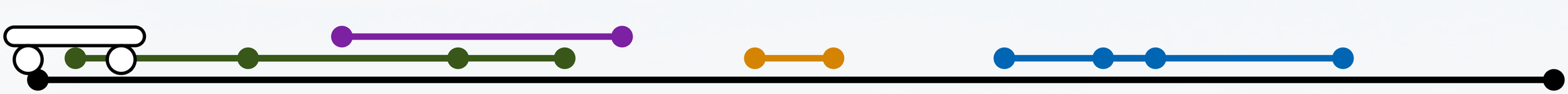
*Theorem:* The shortest sorting path on a path $P$ has $d(\pi)+2\cdot(\#\textit{uncovered edges between cycles})$ steps and can be found in time $\mathcal{O}(n^2)$.

ETH zürich
Daniel Graf

# On Paths



*Theorem:* The shortest sorting path on a path $P$ has $d(\pi)+2\cdot(\#\textit{uncovered edges between cycles})$ steps and can be found in time $\mathcal{O}(n^2)$.

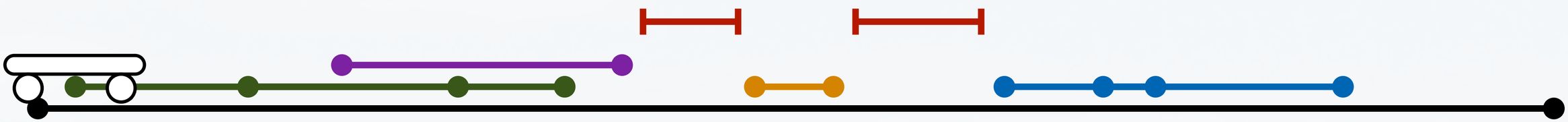


ETH zürich
Daniel Graf

# On Paths



*Theorem:* The shortest sorting path on a path $P$ has $d(\pi)+2\cdot(\#\textit{uncovered edges between cycles})$ steps and can be found in time $\mathcal{O}(n^2)$.

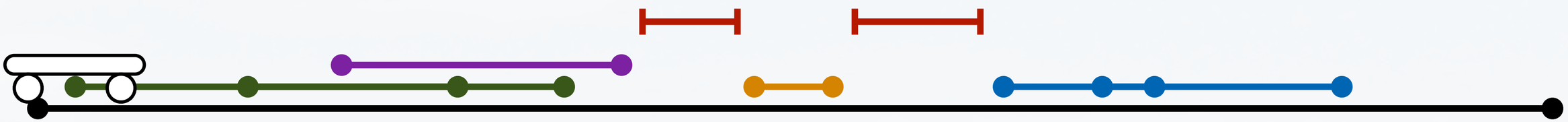


ETH zürich
Daniel Graf

# On Paths



*Theorem:* The shortest sorting path on a path $P$ has $d(\pi)+2\cdot(\#\textit{uncovered edges between cycles})$ steps and can be found in time $\mathcal{O}(n^2)$.



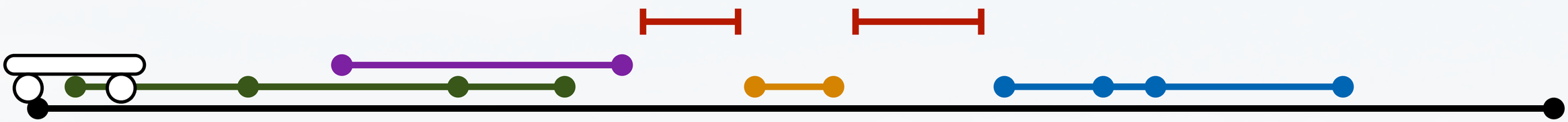
*Proof:* Induction over the cycles of $\pi$.

ETH zürich
Daniel Graf

# On Paths

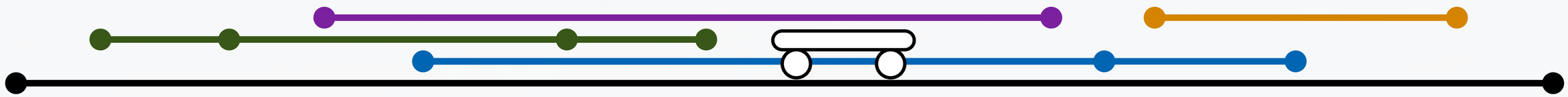


*Theorem:* The shortest sorting path on a path $P$ has $d(\pi)+2\cdot(\#\textit{uncovered edges between cycles})$ steps and can be found in time $\mathcal{O}(n^2)$.



*Proof:* Induction over the cycles of $\pi$.

*Open:* What if we do not start at the border?



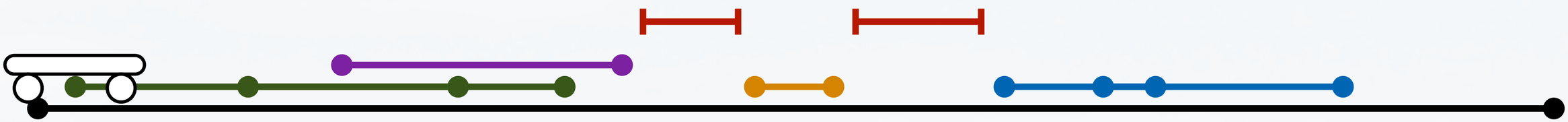
Non-essential steps needed also inside the cycles.

ETH zürich
Daniel Graf

# On Paths

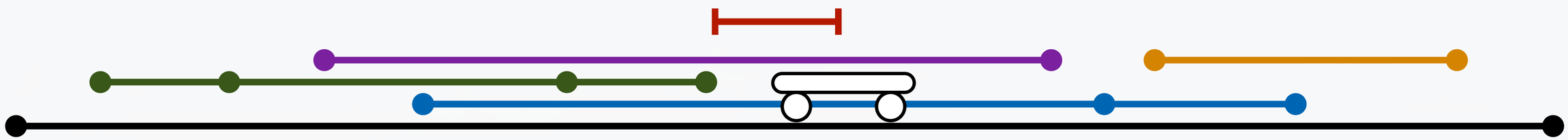


*Theorem:* The shortest sorting path on a path $P$ has $d(\pi)+2\cdot(\#\textit{uncovered edges between cycles})$ steps and can be found in time $\mathcal{O}(n^2)$.



*Proof:* Induction over the cycles of $\pi$.

*Open:* What if we do not start at the border?

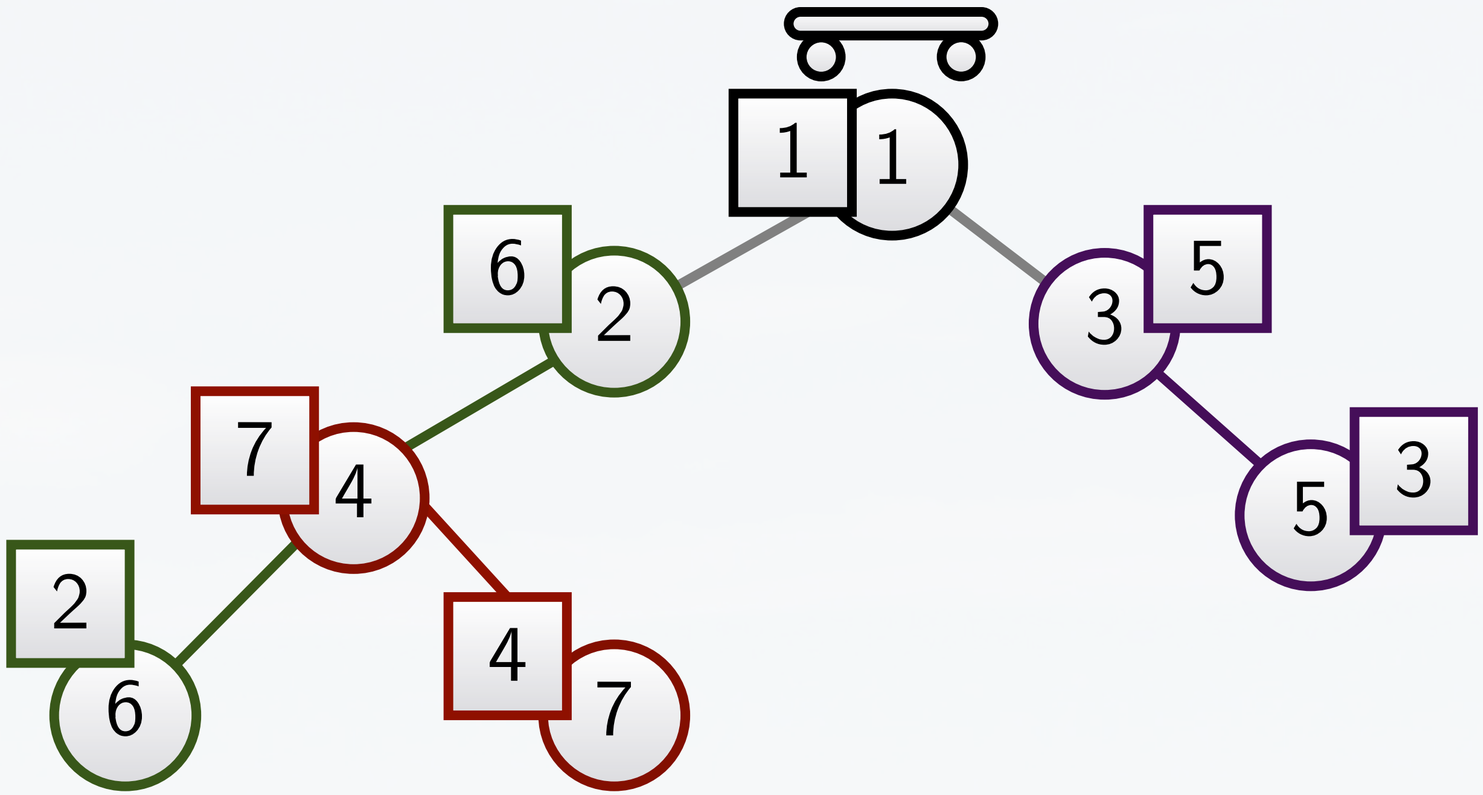


Non-essential steps needed also inside the cycles.

# On Trees



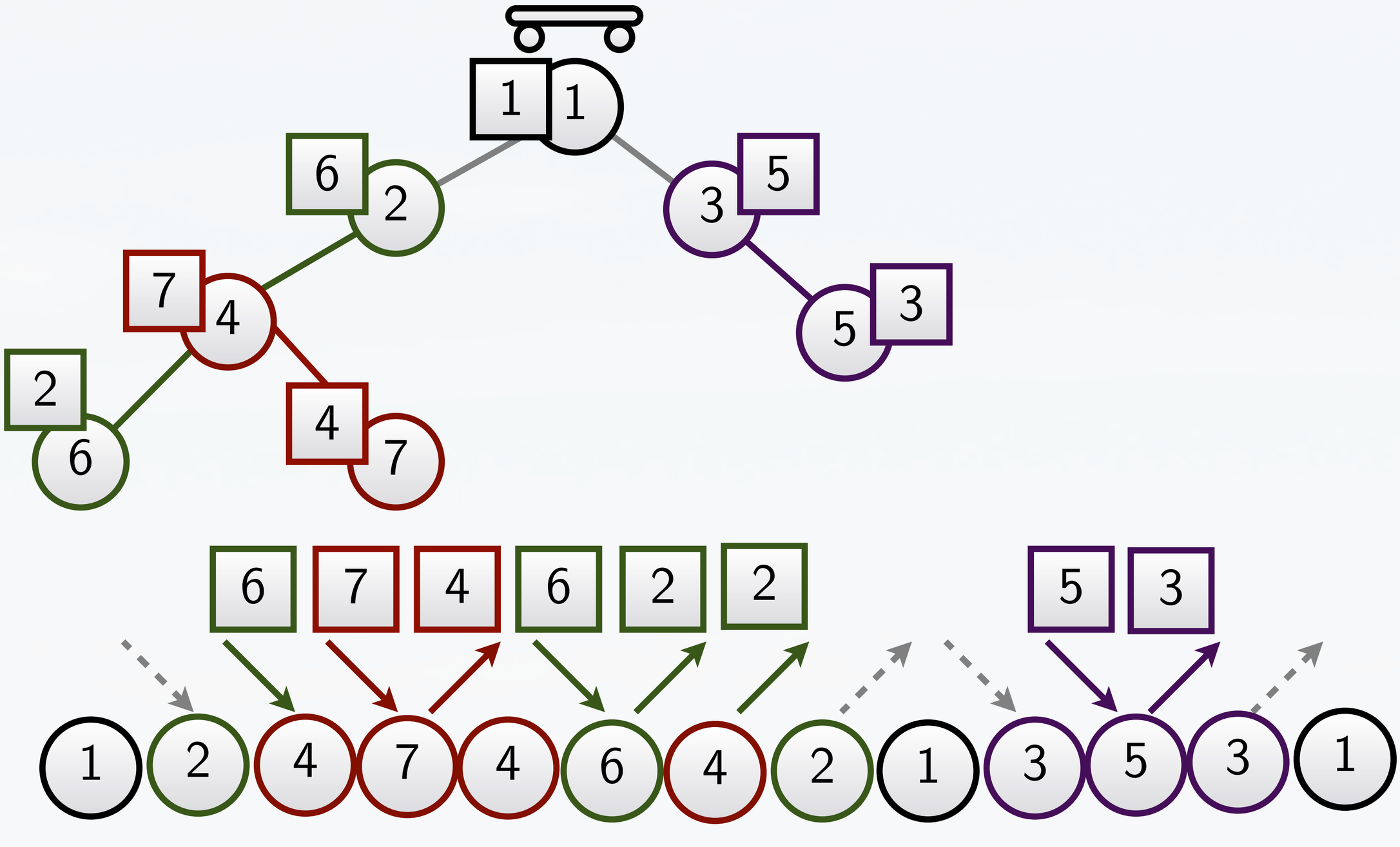
How do we minimize the non-essential steps?



# On Trees



How do we minimize the non-essential steps?

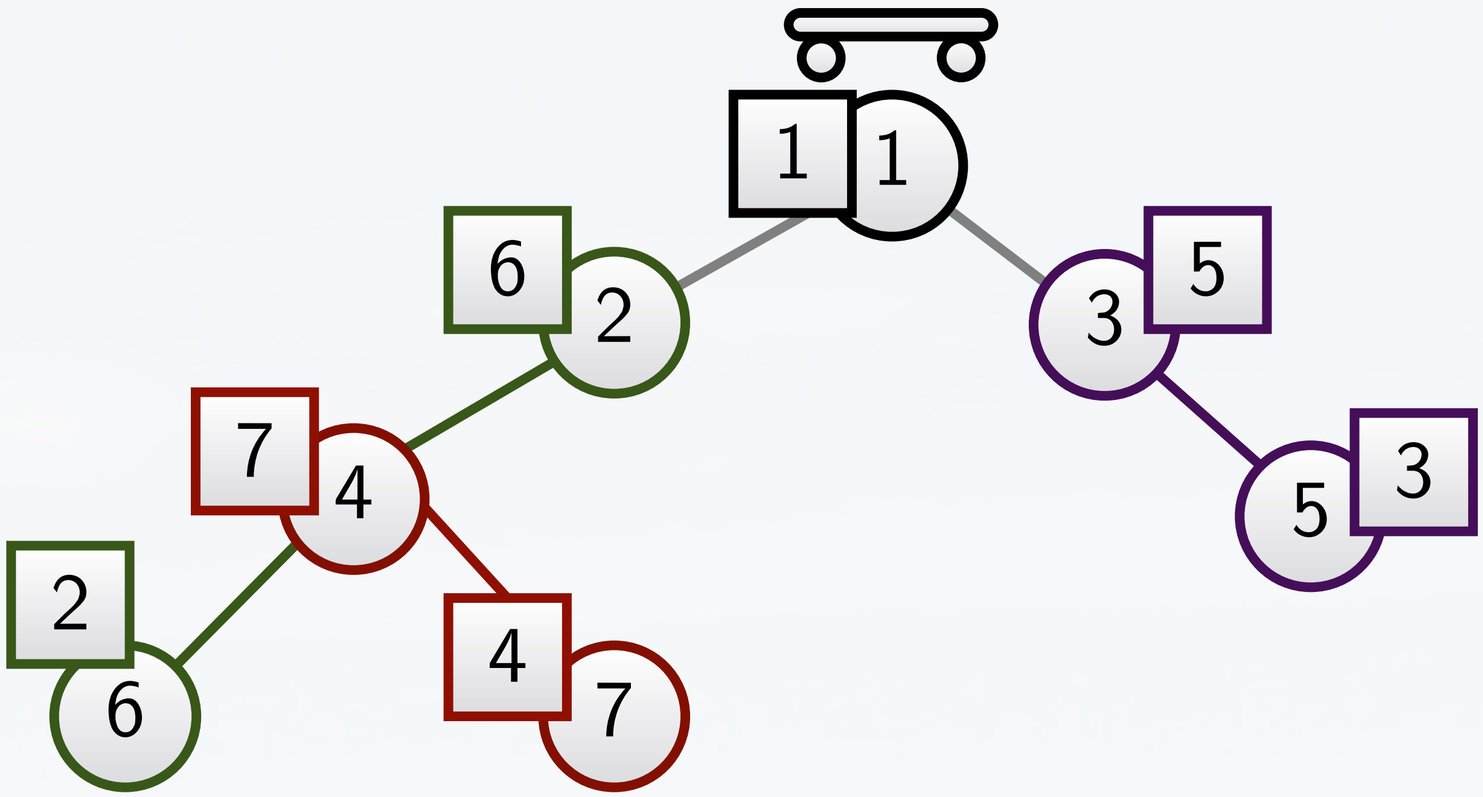


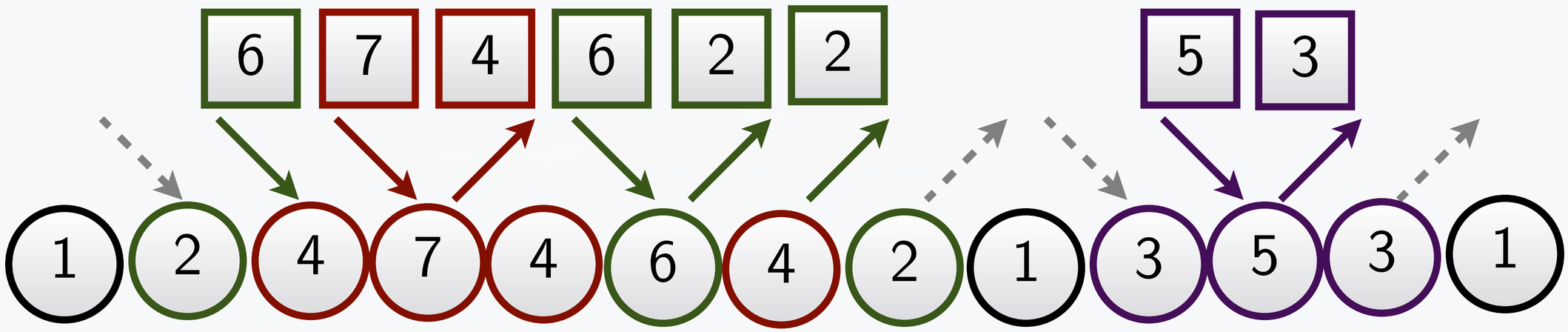
# On Trees



How do we minimize the non-essential steps?

cycle anchor tree

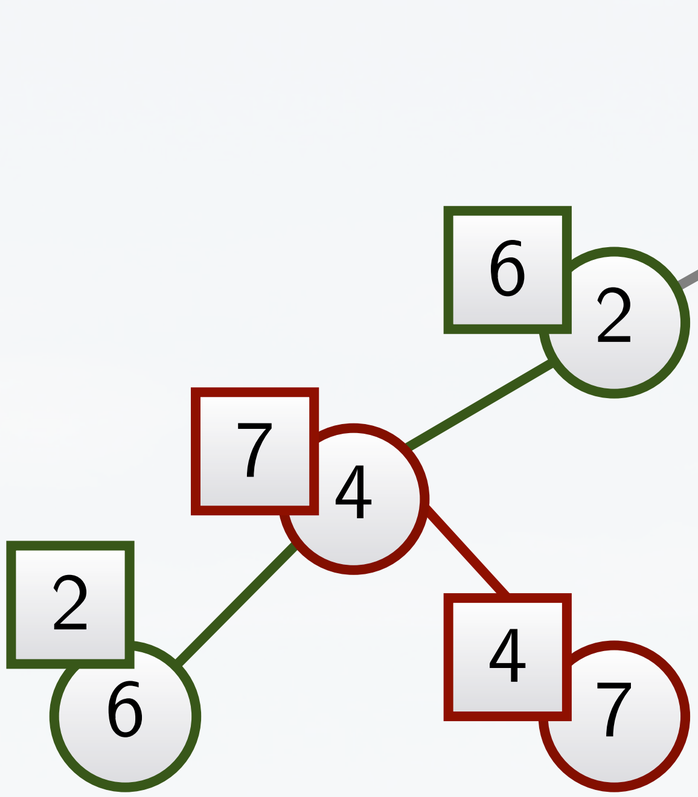


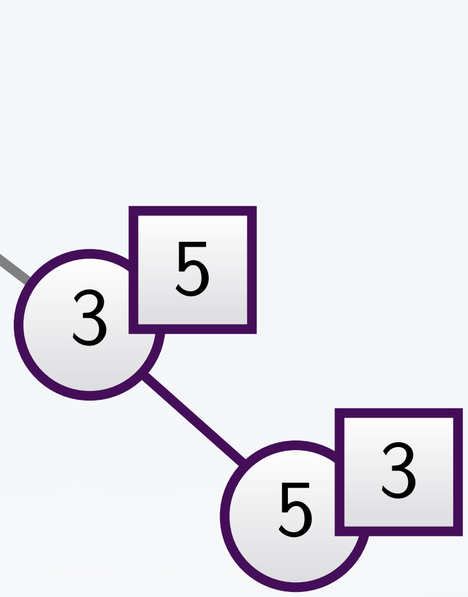
# On Trees

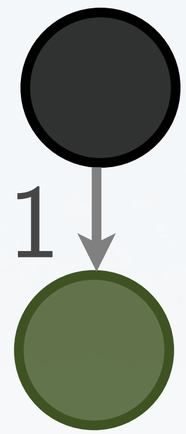


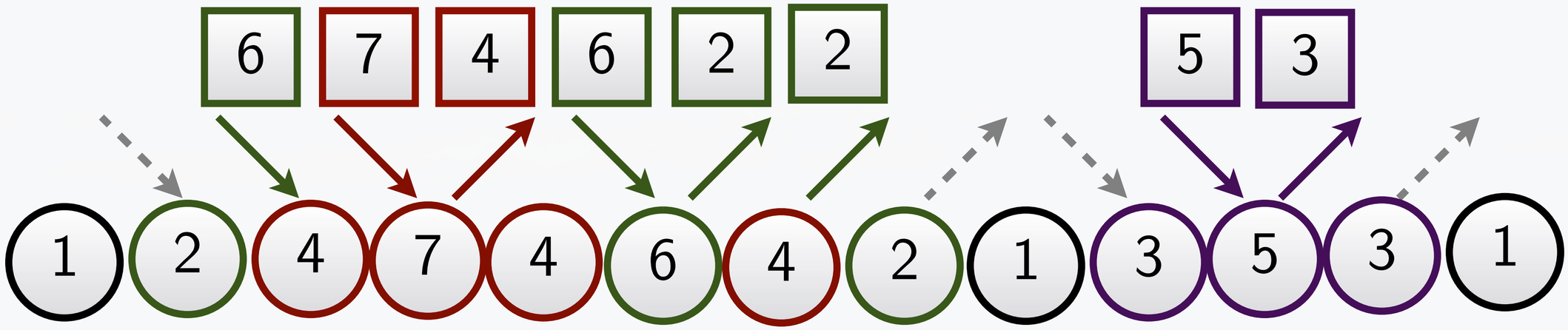
How do we minimize the non-essential steps?




cycle anchor tree



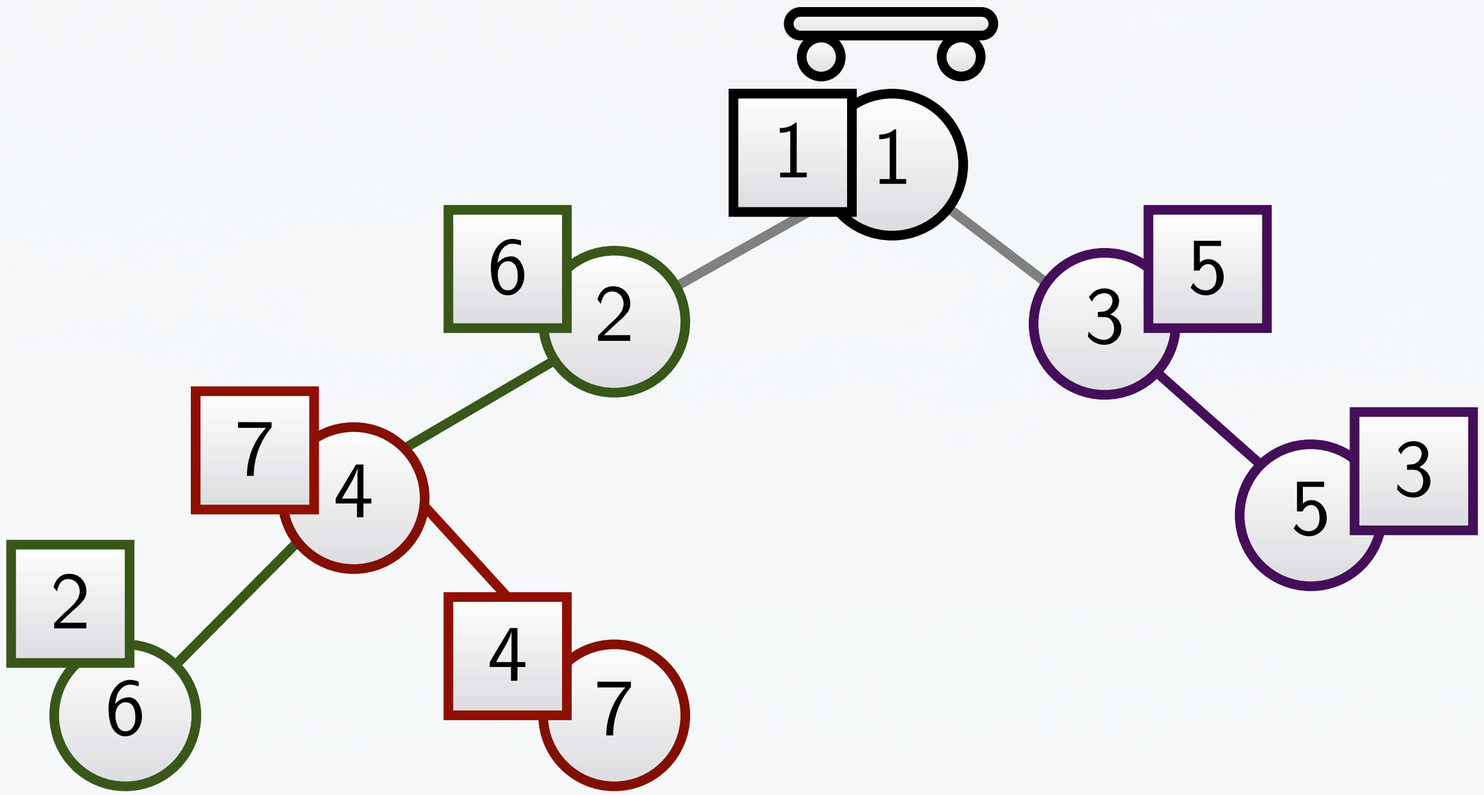



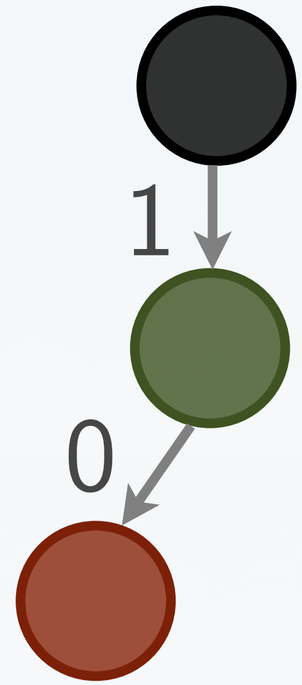
# On Trees

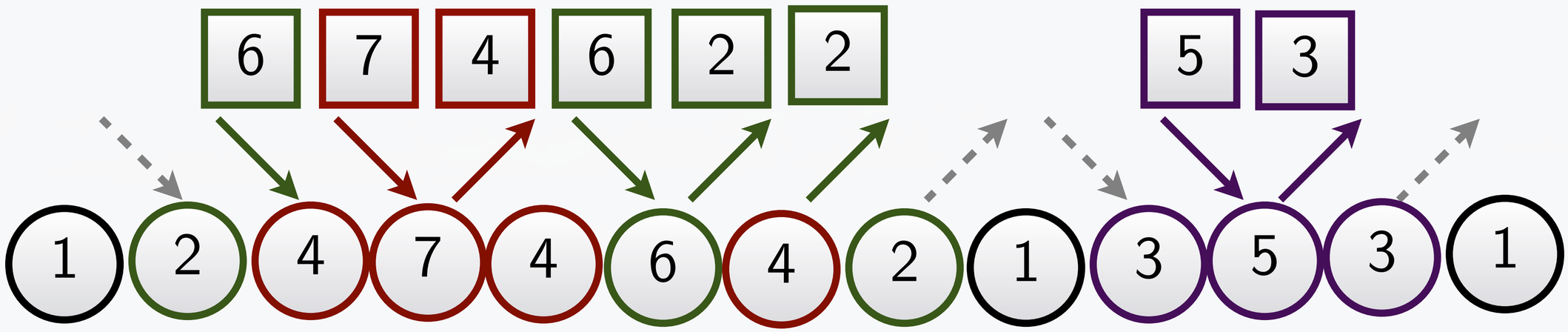


How do we minimize the non-essential steps?




cycle anchor tree

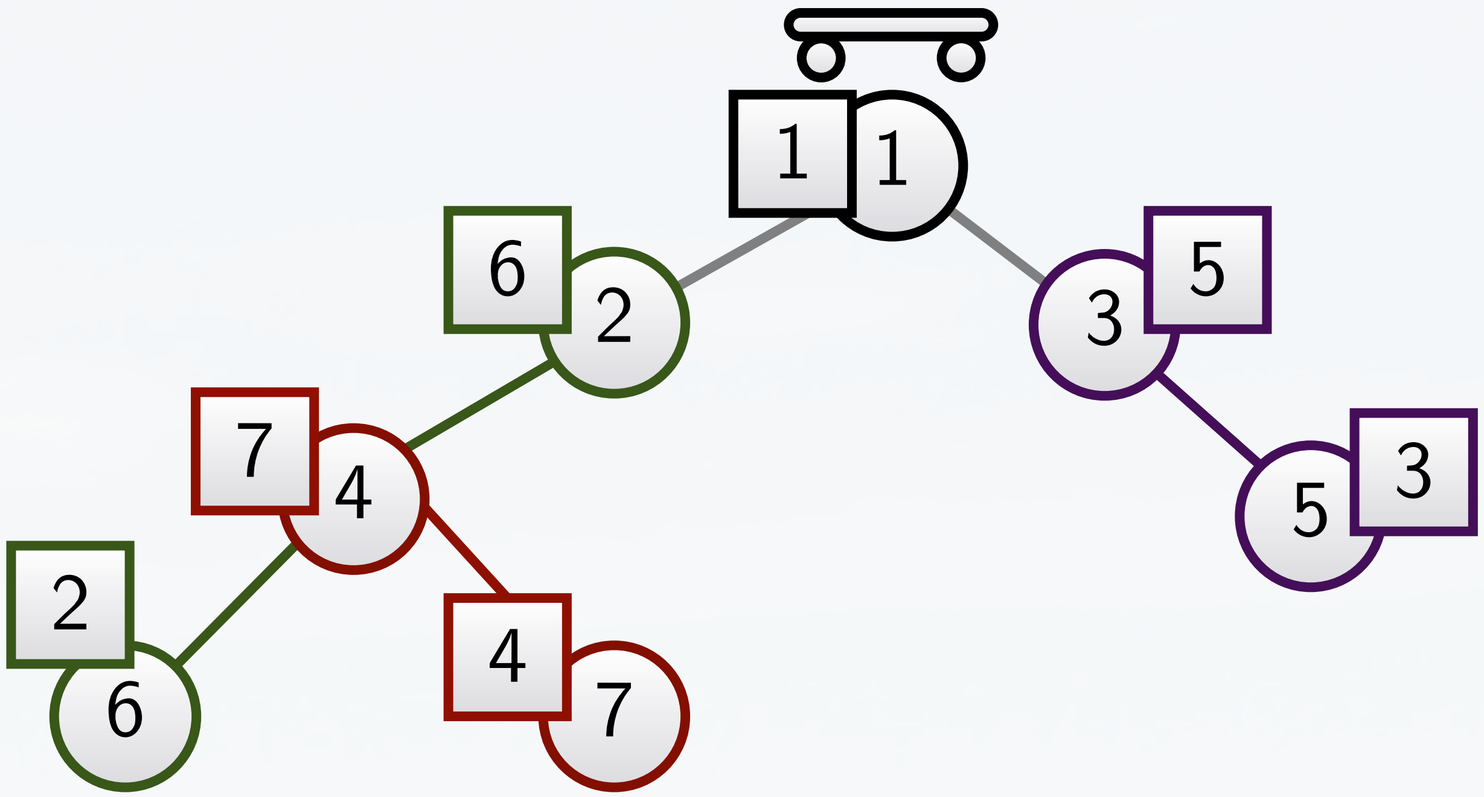


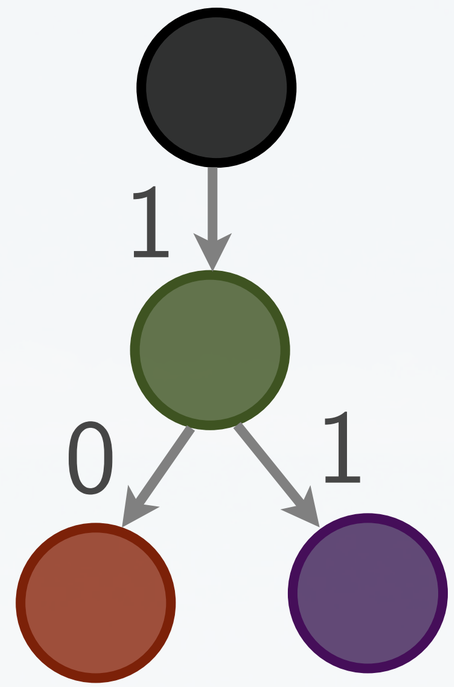
# On Trees

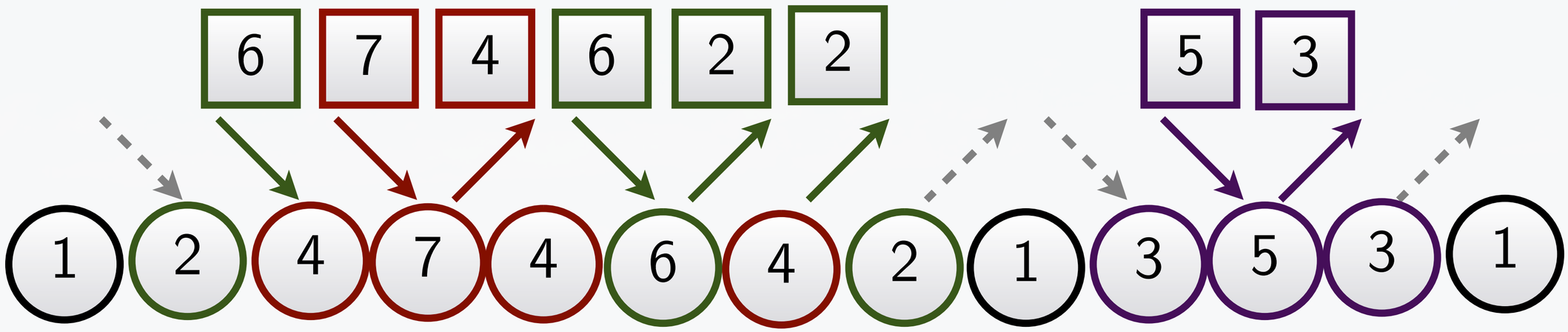


How do we minimize the non-essential steps?




cycle anchor tree



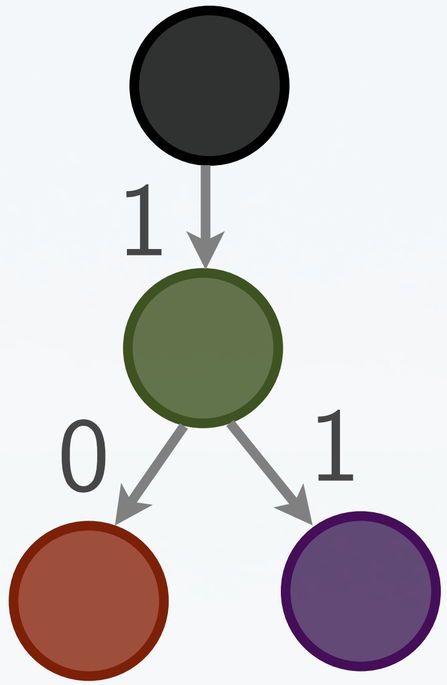
# On Trees



How do we minimize the non-essential steps?

*Edge cost* $c((C_1, C_2)) =$ minimal number of down steps from any vertex covered by $C_1$ to a vertex containing a box of $C_2$

cycle anchor tree



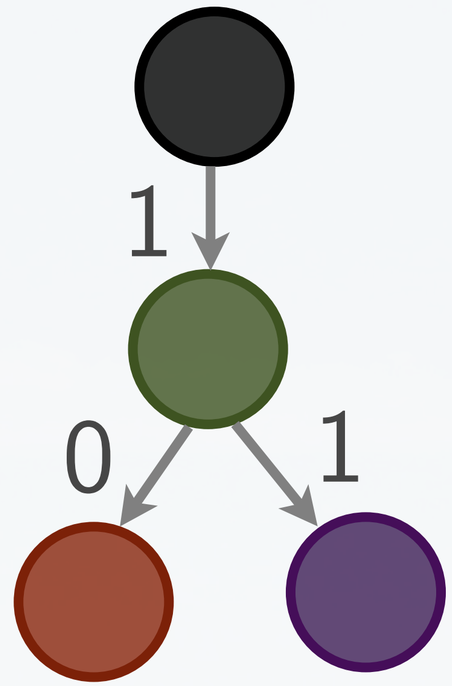
# On Trees



How do we minimize the non-essential steps?

*Edge cost* $c((C_1,C_2)) =$ minimal number of down steps from any vertex covered by $C_1$ to a vertex containing a box of $C_2$

cycle anchor tree



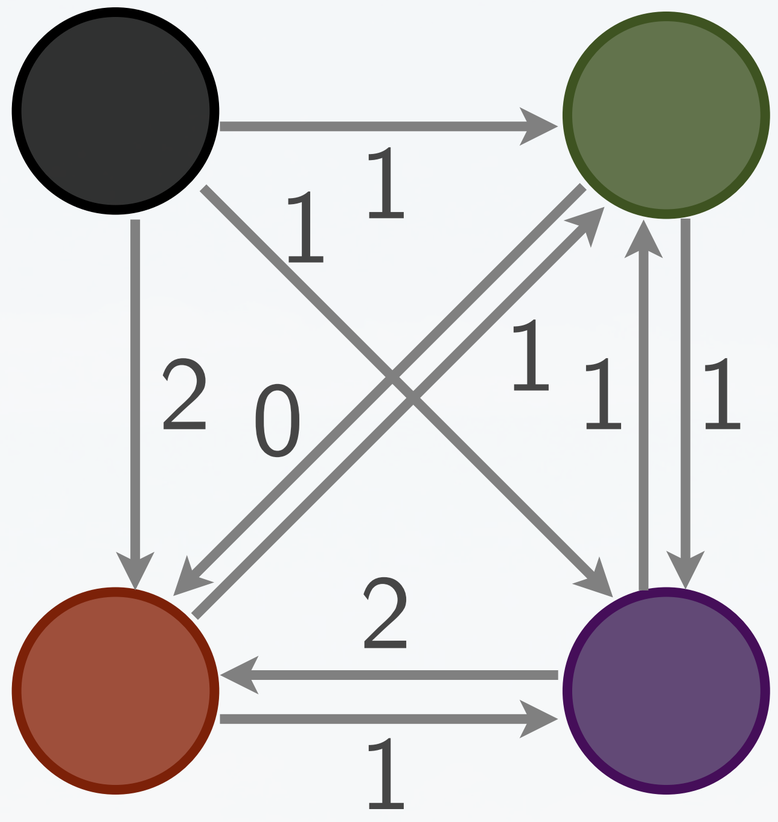

*Observation:* A cycle anchor tree determines the length of all corresponding sorting walks.

# On Trees



How do we find the cheapest cycle anchor tree?

- Start with the complete directed graph on all cycles.
- Find a cheapest direct subgraph rooted at ● using Edmond's algorithm for optimum branchings.



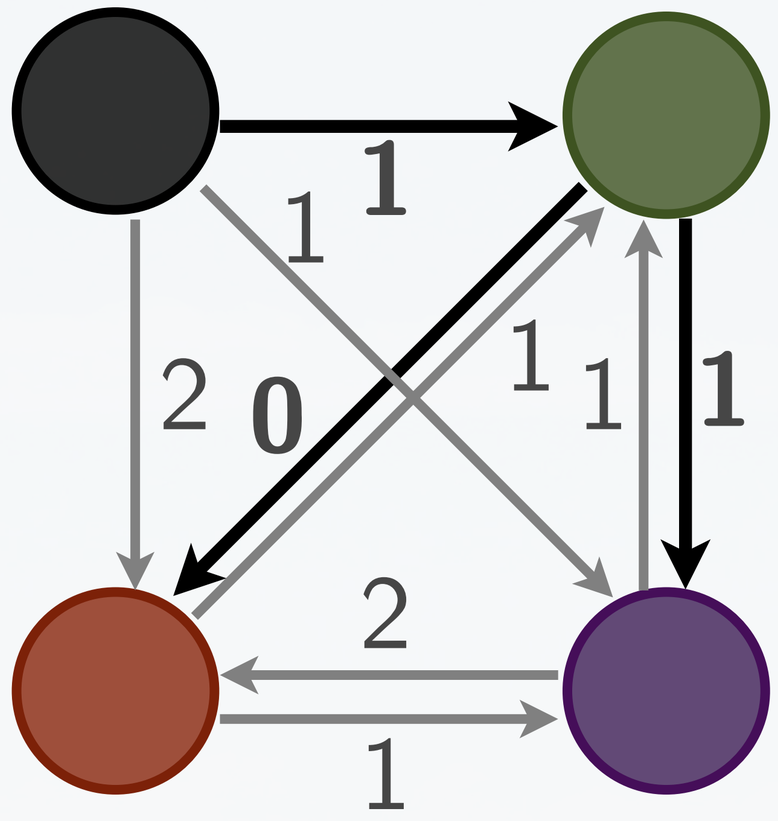
ETH zürich
Daniel Graf

# On Trees



How do we find the cheapest cycle anchor tree?

- Start with the complete directed graph on all cycles.
- Find a cheapest direct subgraph rooted at ● using Edmond's algorithm for optimum branchings.



ETH zürich
Daniel Graf

# On Trees



*Theorem:* The shortest sorting path on a tree $T$ can be found in time $\mathcal{O}(n^2)$.

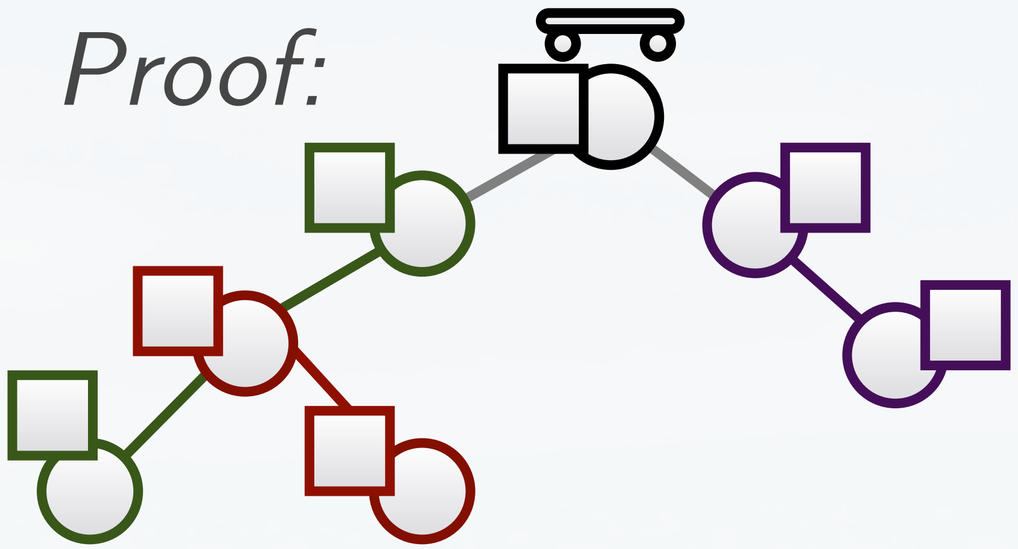
ETH zürich
Daniel Graf

# On Trees



*Theorem:* The shortest sorting path on a tree $T$ can be found in time $\mathcal{O}(n^2)$.

*Proof:*



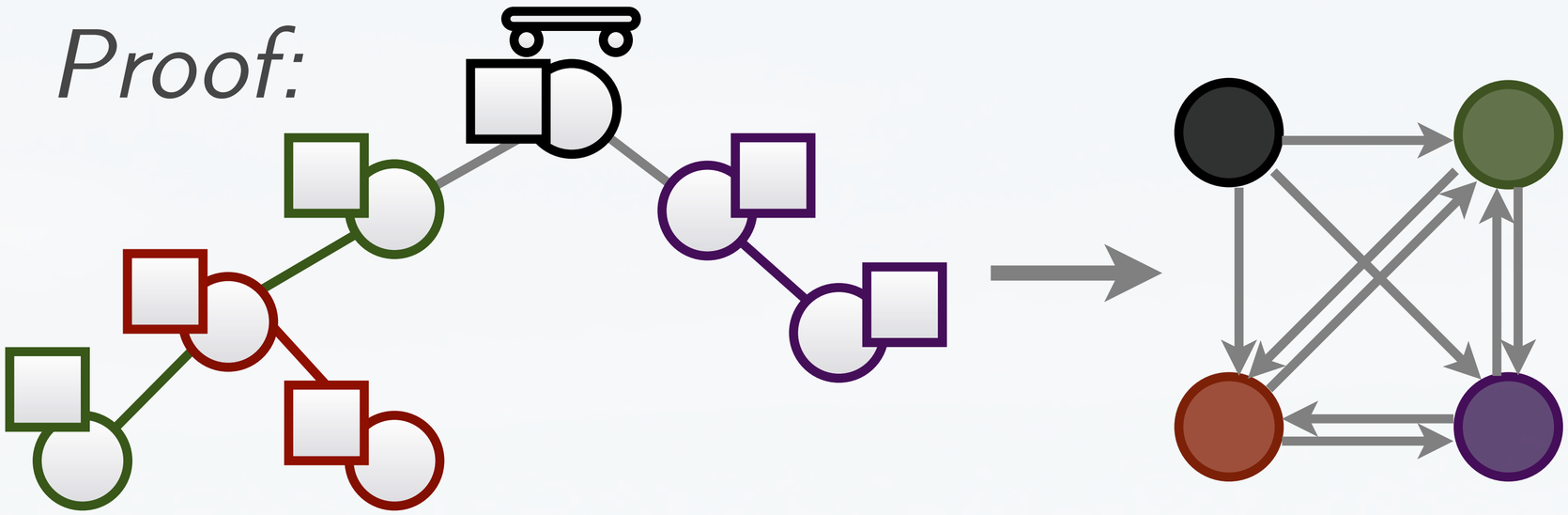
ETH zürich
Daniel Graf

# On Trees



*Theorem:* The shortest sorting path on a tree $T$ can be found in time $\mathcal{O}(n^2)$.

*Proof:*



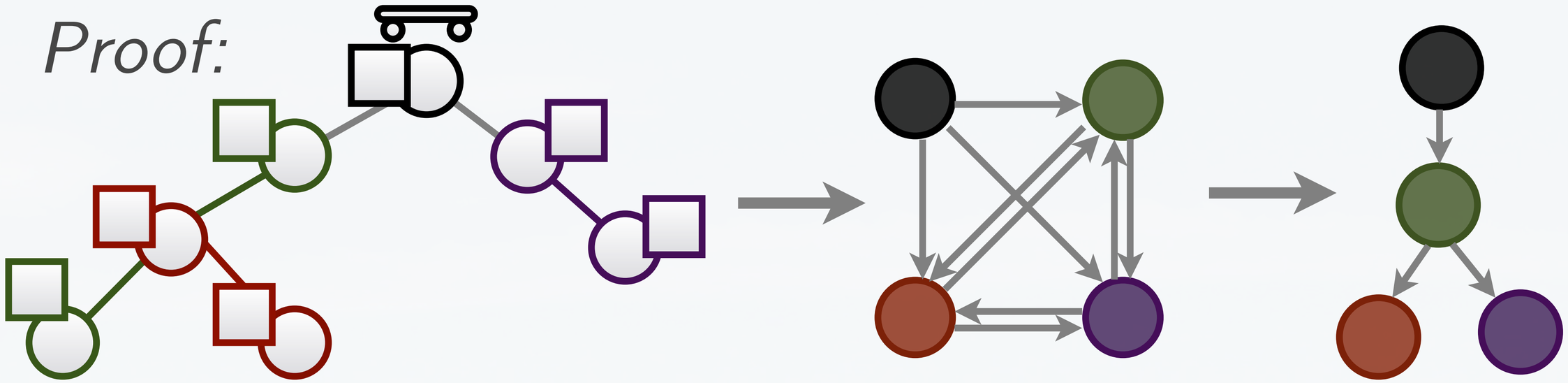
ETH zürich
Daniel Graf

# On Trees



*Theorem:* The shortest sorting path on a tree $T$ can be found in time $\mathcal{O}(n^2)$.

*Proof:*



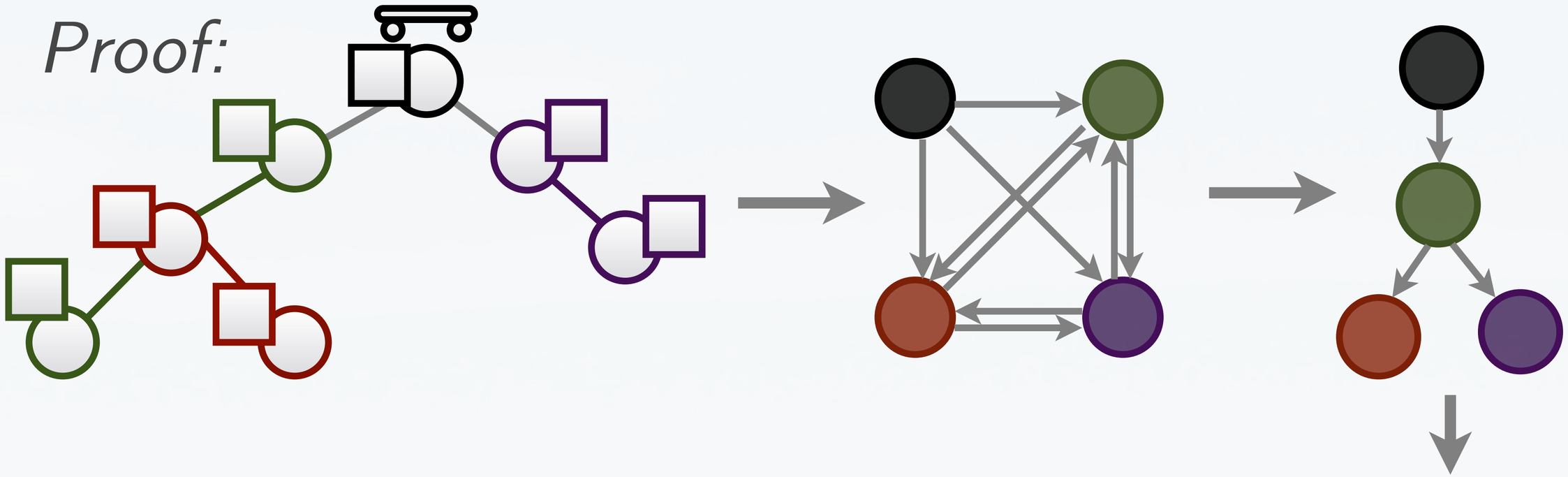
ETH zürich
Daniel Graf

# On Trees



*Theorem:* The shortest sorting path on a tree $T$ can be found in time $\mathcal{O}(n^2)$.

*Proof:*



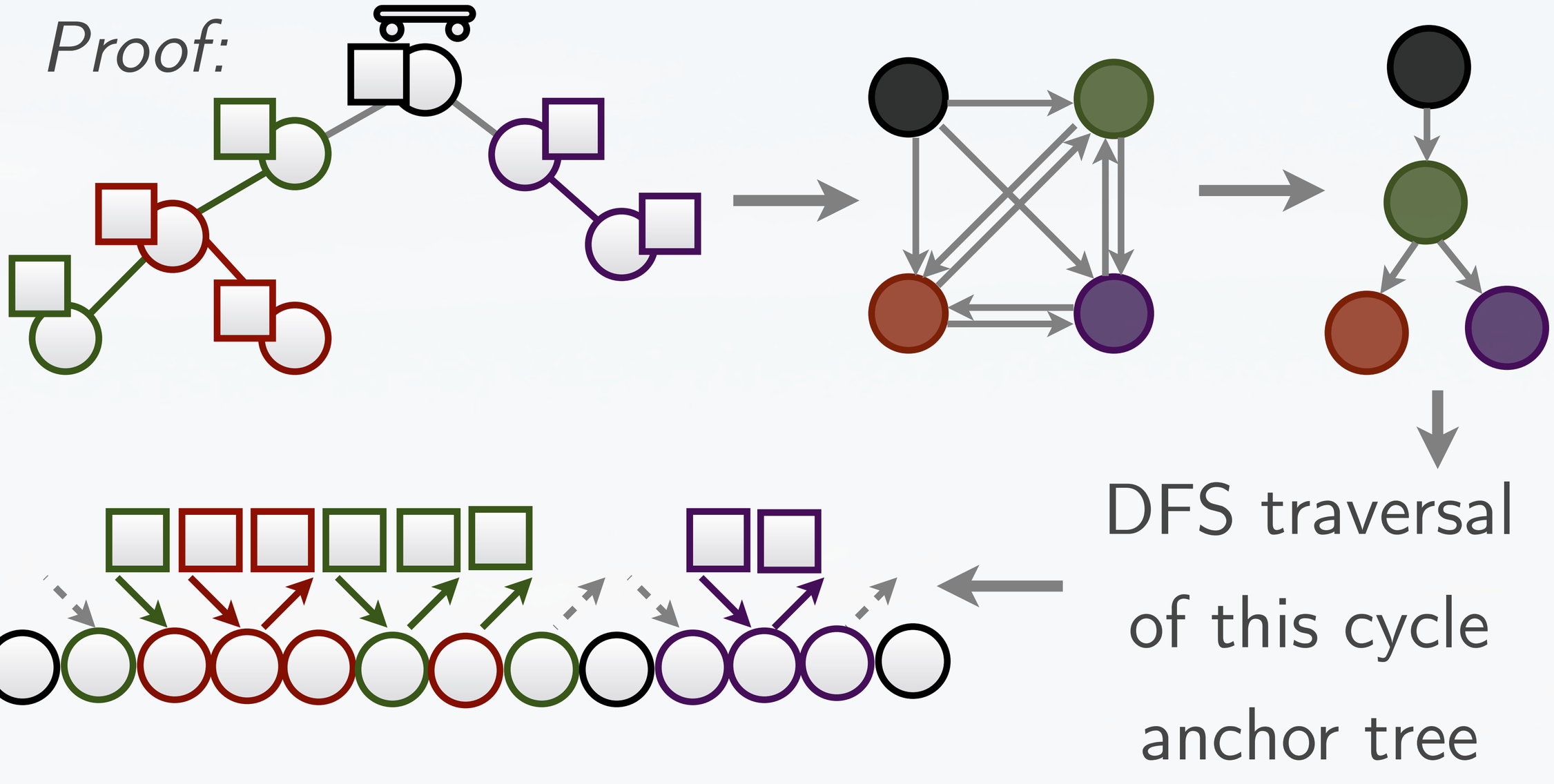
ETH zürich
Daniel Graf

# On Trees



*Theorem:* The shortest sorting path on a tree $T$ can be found in time $\mathcal{O}(n^2)$.

*Proof:*



ETH zürich
Daniel Graf

# On Circles



*Theorem:* The shortest sorting path on a circle $C$ can be found in time $\mathcal{O}(n^3)$.

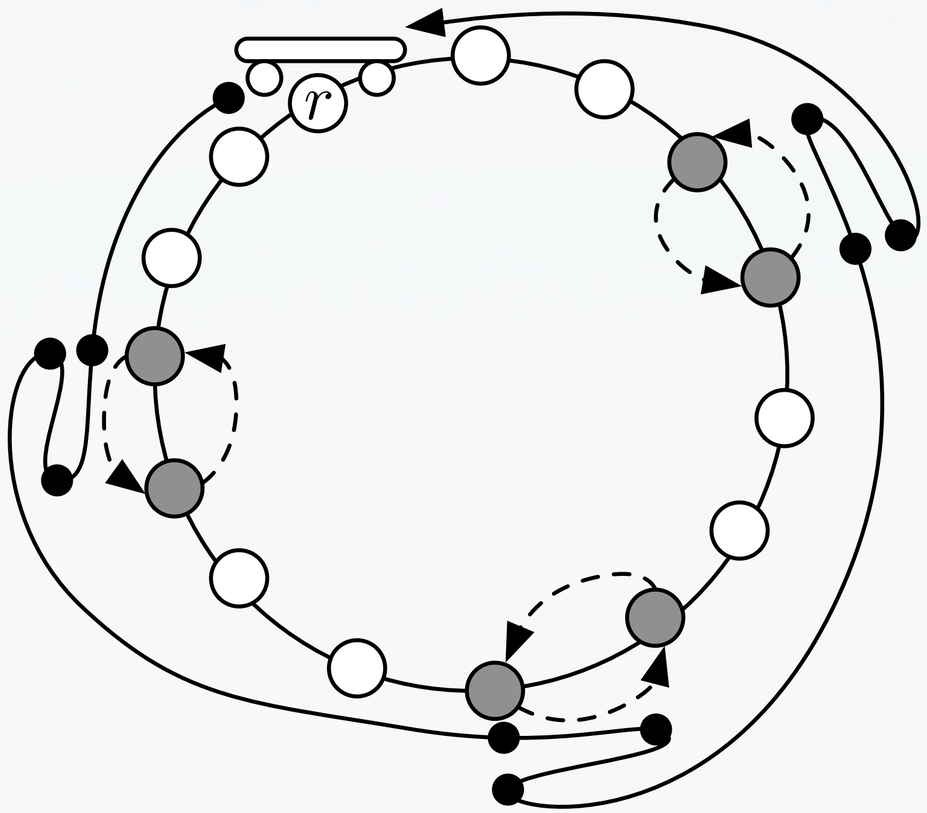
ETH zürich
Daniel Graf

# On Circles



*Theorem:* The shortest sorting path on a circle $C$ can be found in time $\mathcal{O}(n^3)$.

*Proof Ideas:* Full circle might be necessary.



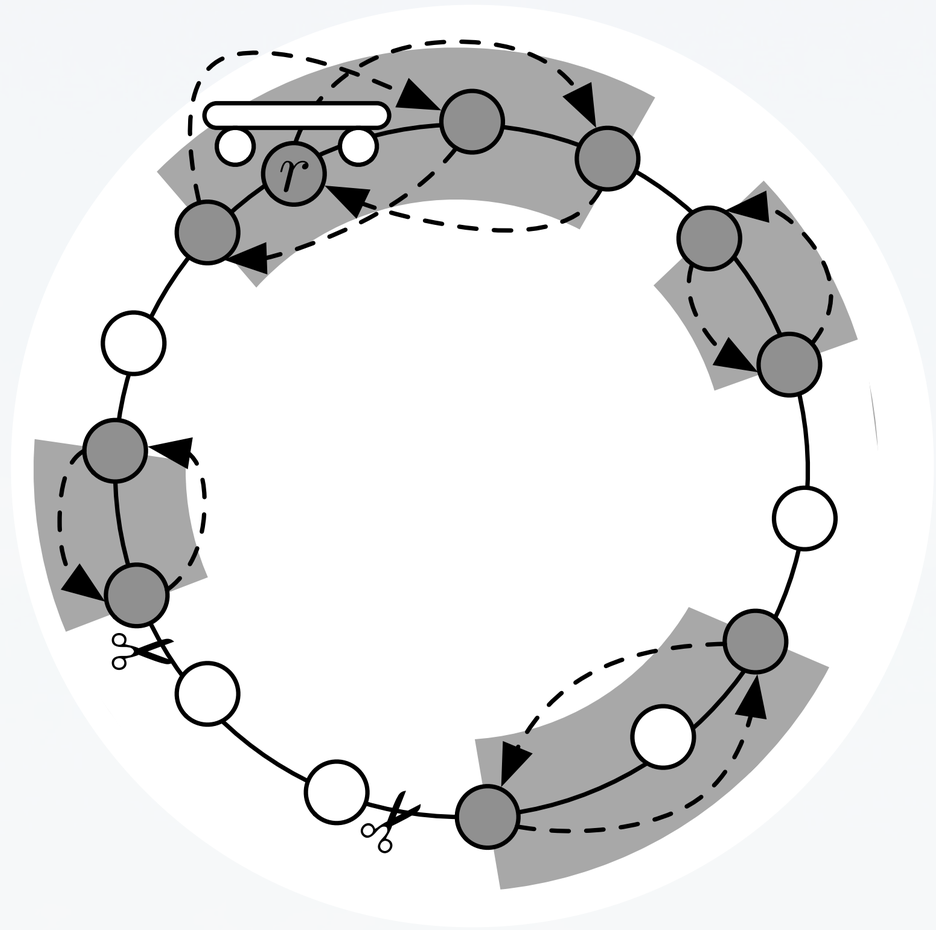
ETH zürich
Daniel Graf

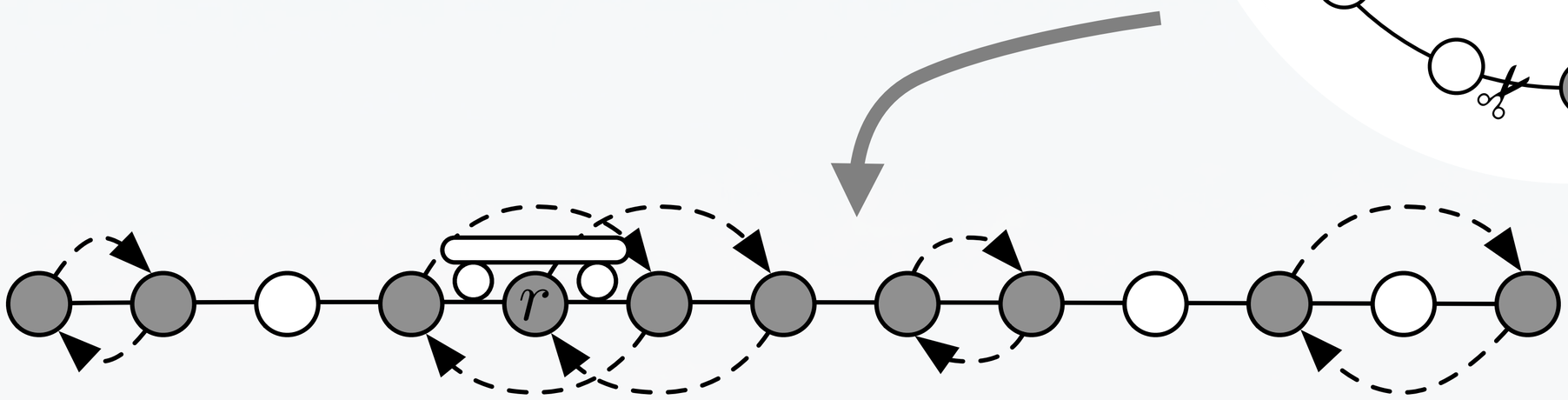
# On Circles



*Theorem:* The shortest sorting path on a circle $C$ can be found in time $\mathcal{O}(n^3)$.

*Proof Ideas:* Reduce it to sorting on a path.



# Planar Graphs



What if we look at more general graphs and allow multiple cycles?

# Planar Graphs



What if we look at more general graphs and allow multiple cycles?

*Theorem:* Finding a shortest sorting walk is *NP*-complete for planar graphs.

# Planar Graphs



What if we look at more general graphs and allow multiple cycles?

*Theorem:* Finding a shortest sorting walk is *NP*-complete for planar graphs.

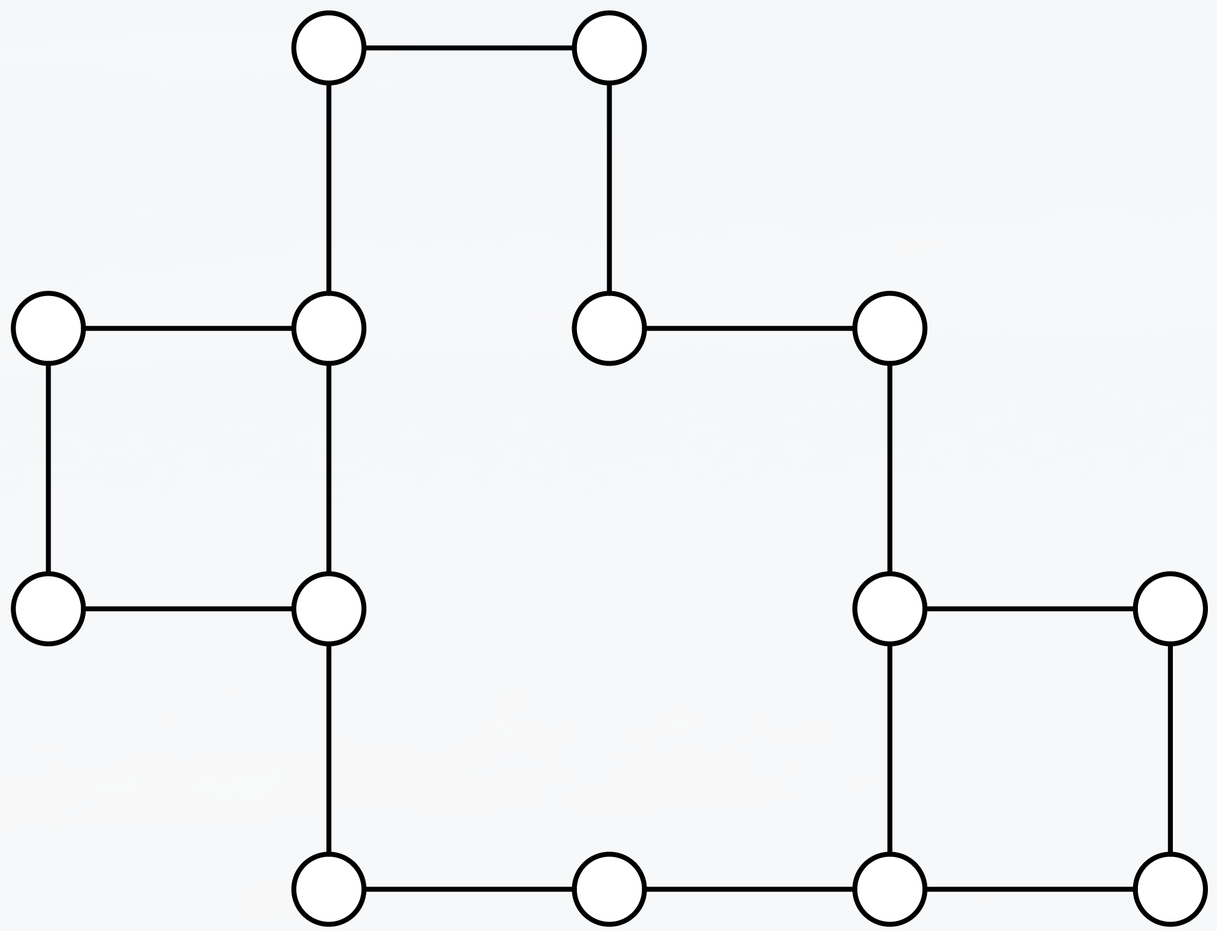
*Proof:* Reduction from the problem of finding Hamiltonian circuits for grid graphs.

ETH zürich
Daniel Graf

# Planar Graphs



*Theorem [Itai et al. 1981]:* Finding Hamiltonian circuits for grid graphs is *NP*-complete.

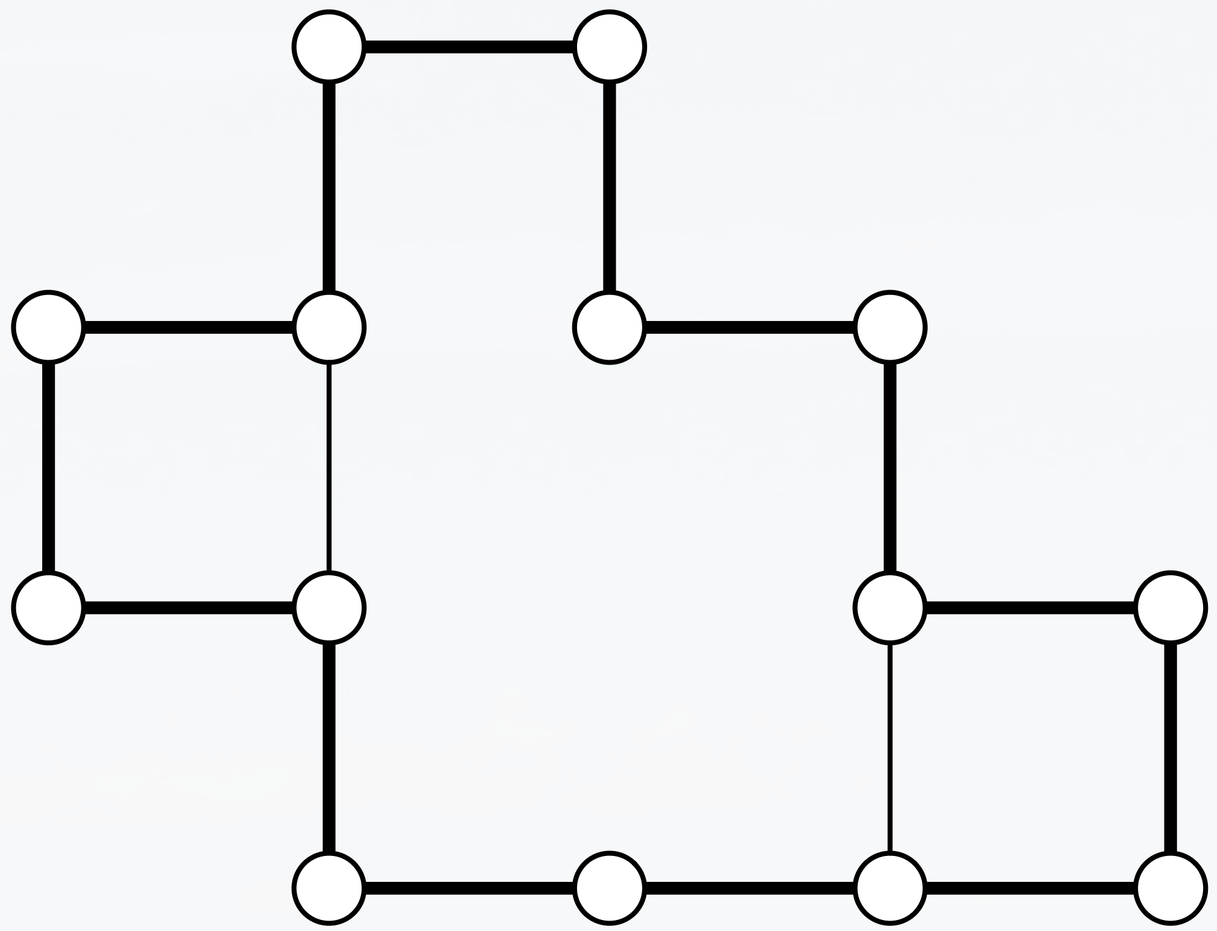


ETH zürich
Daniel Graf

# Planar Graphs



*Theorem [Itai et al. 1981]:* Finding Hamiltonian circuits for grid graphs is *NP*-complete.



ETH zürich
Daniel Graf

# Planar Graphs



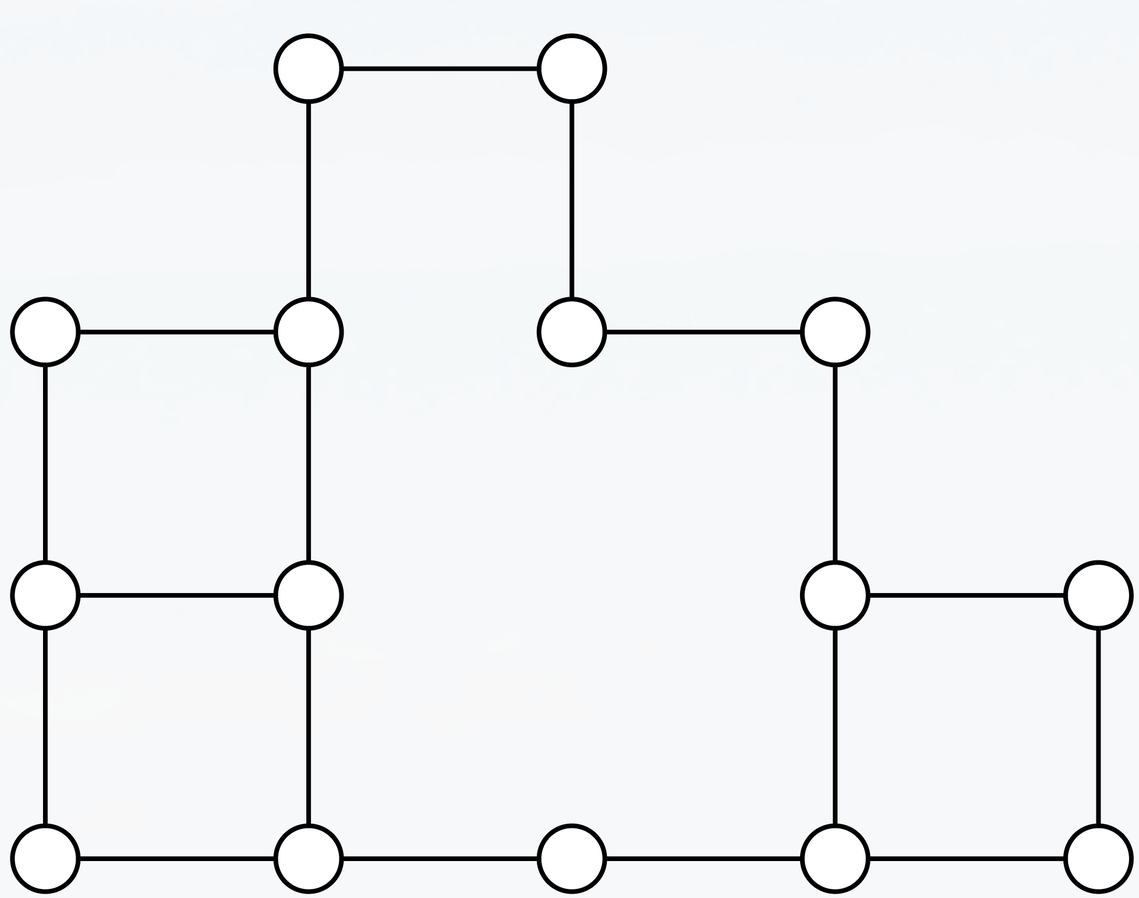
*Theorem:* Finding a shortest sorting walk is *NP*-complete for planar graphs.

ETH zürich
Daniel Graf

# Planar Graphs



*Theorem:* Finding a shortest sorting walk is *NP*-complete for planar graphs.

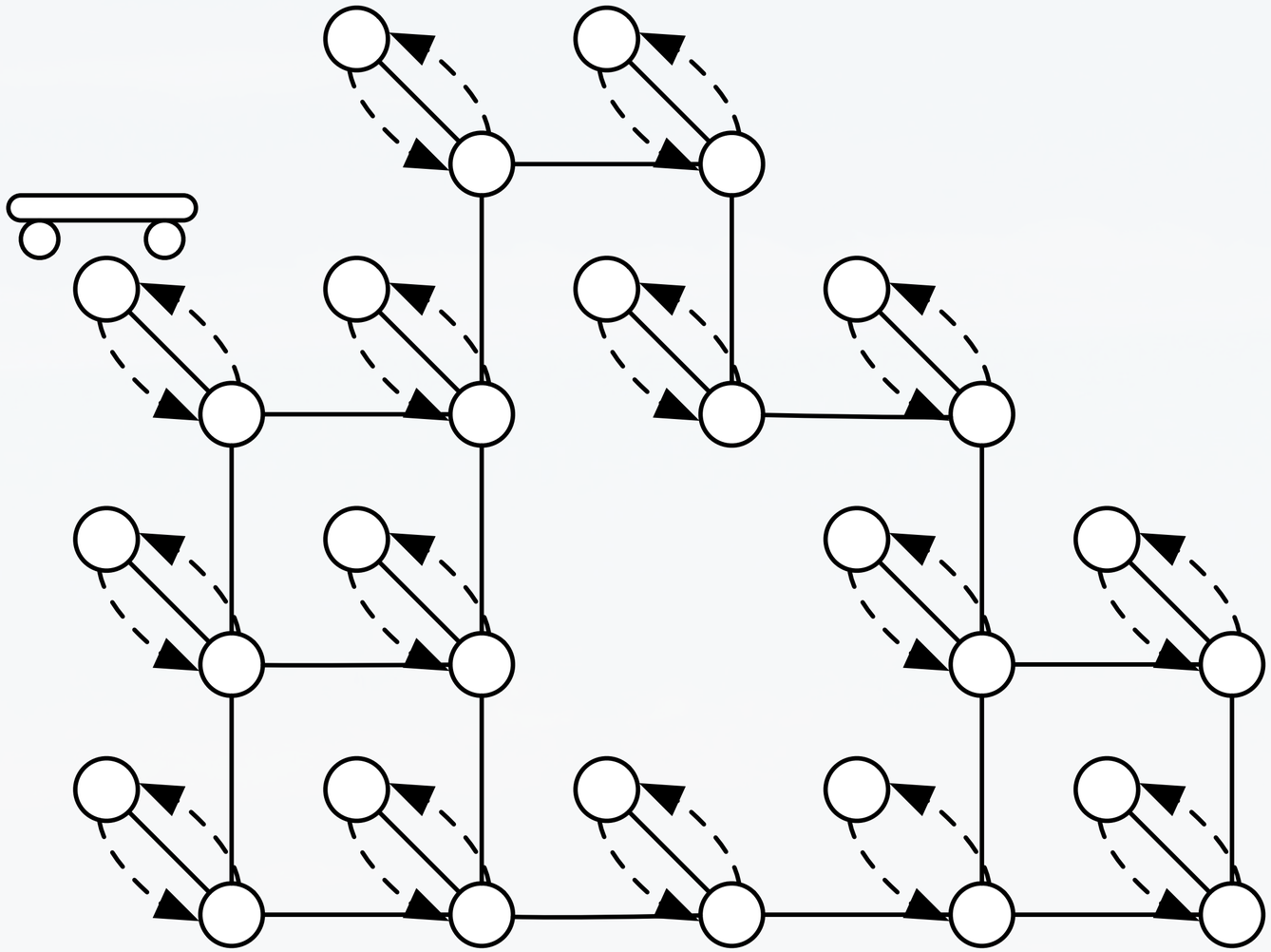


ETH zürich
Daniel Graf

# Planar Graphs



*Theorem:* Finding a shortest sorting walk is *NP*-complete for planar graphs.



# Fewest Swaps



Box handling takes significant time in practice.

Different cost function:

- count number of swaps
- minimizing number of steps as a second priority

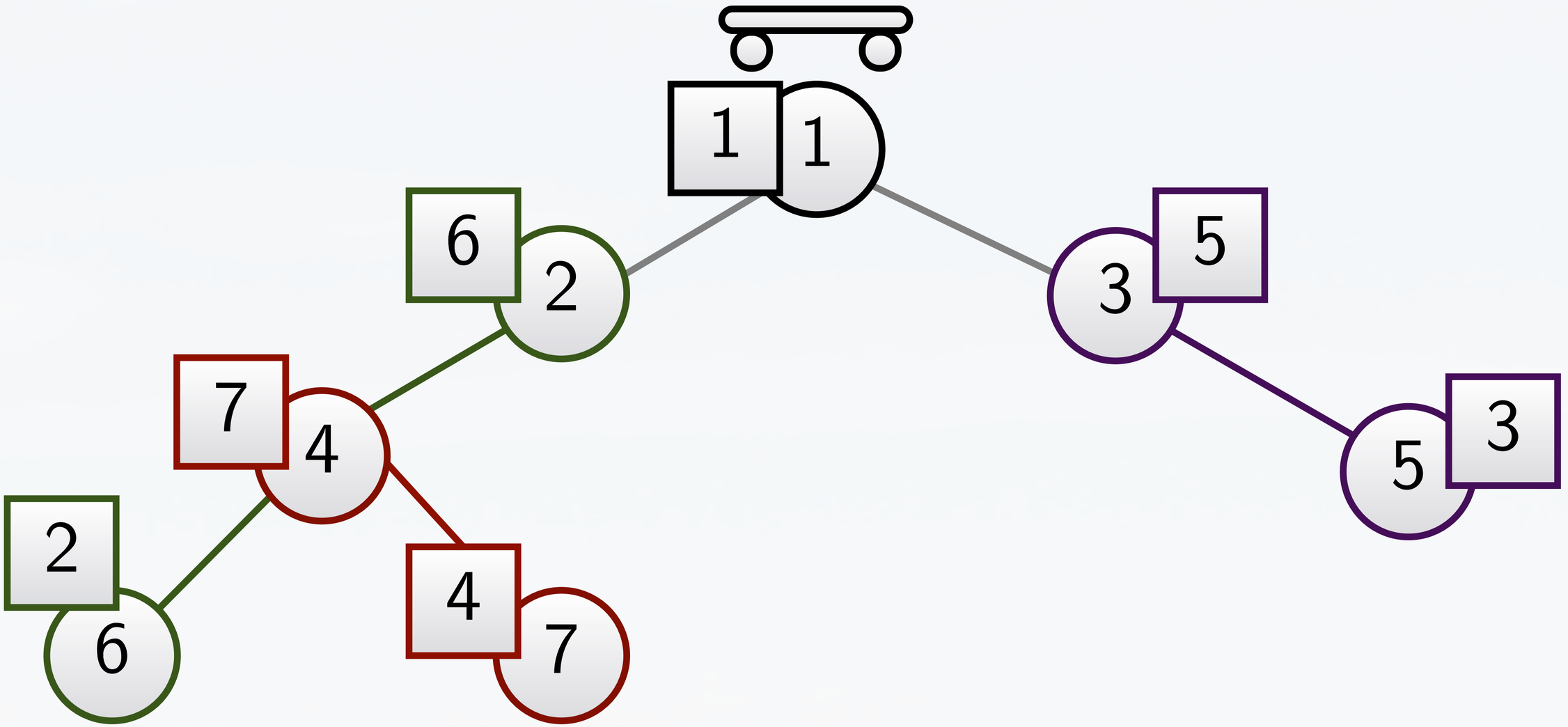
Observations:

- no interleaving of cycles anymore
- non-essential steps form a subtree containing $r$ and a vertex of every cycle of $\pi$.

ETH zürich
Daniel Graf

# Fewest Swaps

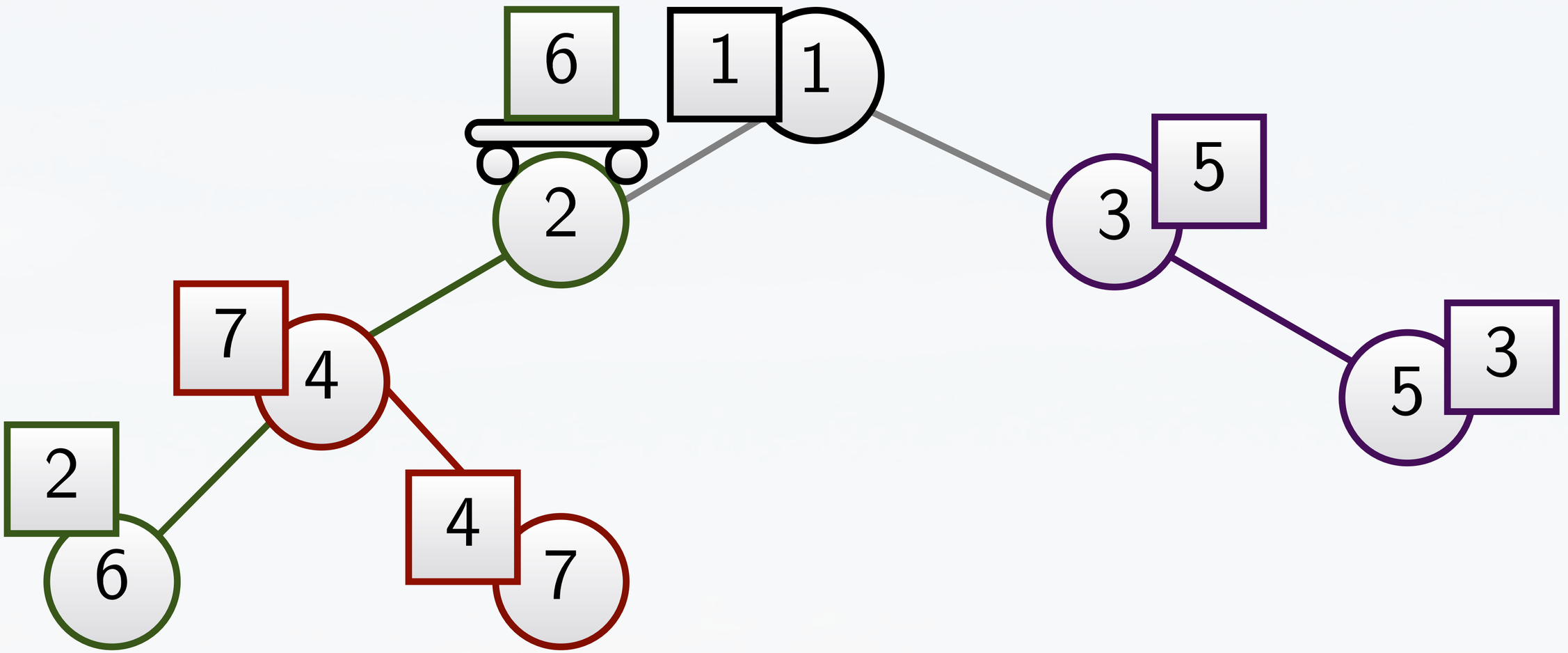




ETH zürich
Daniel Graf

# Fewest Swaps

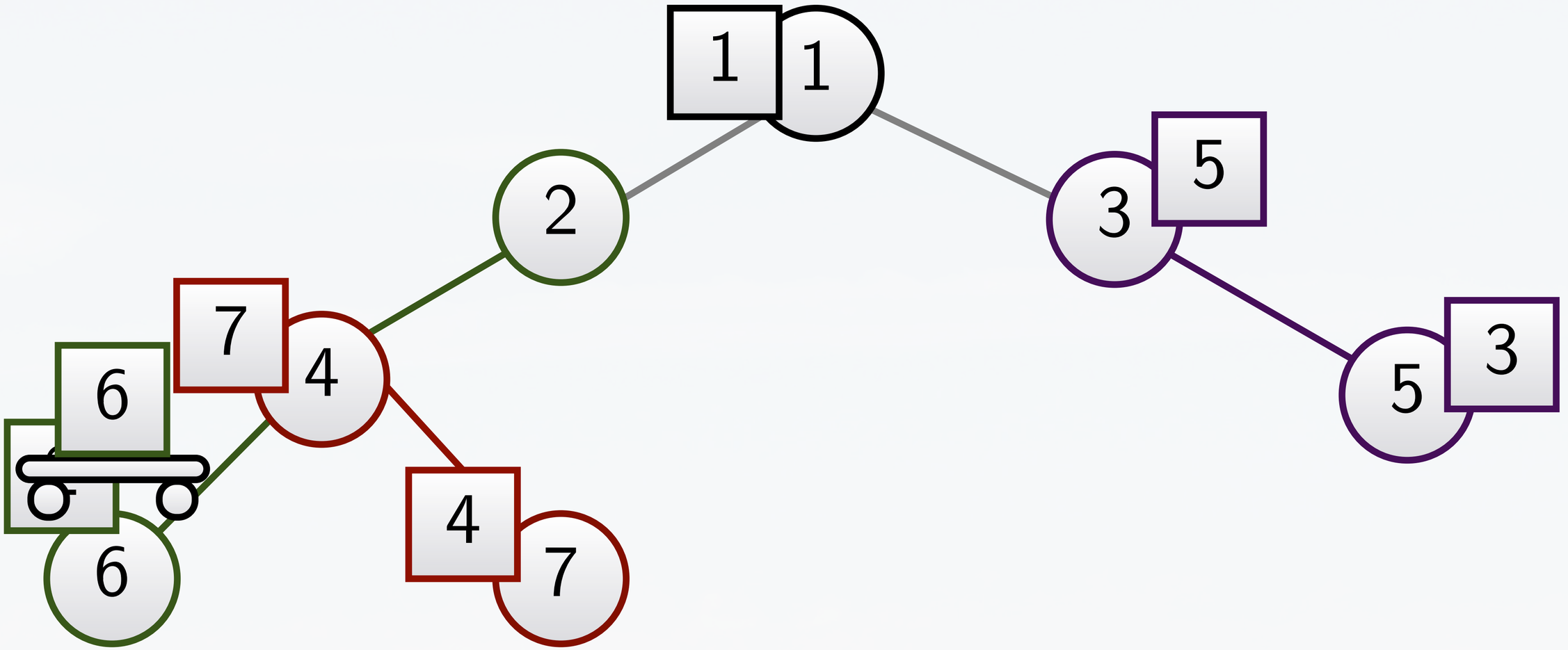




ETH zürich
Daniel Graf

# Fewest Swaps

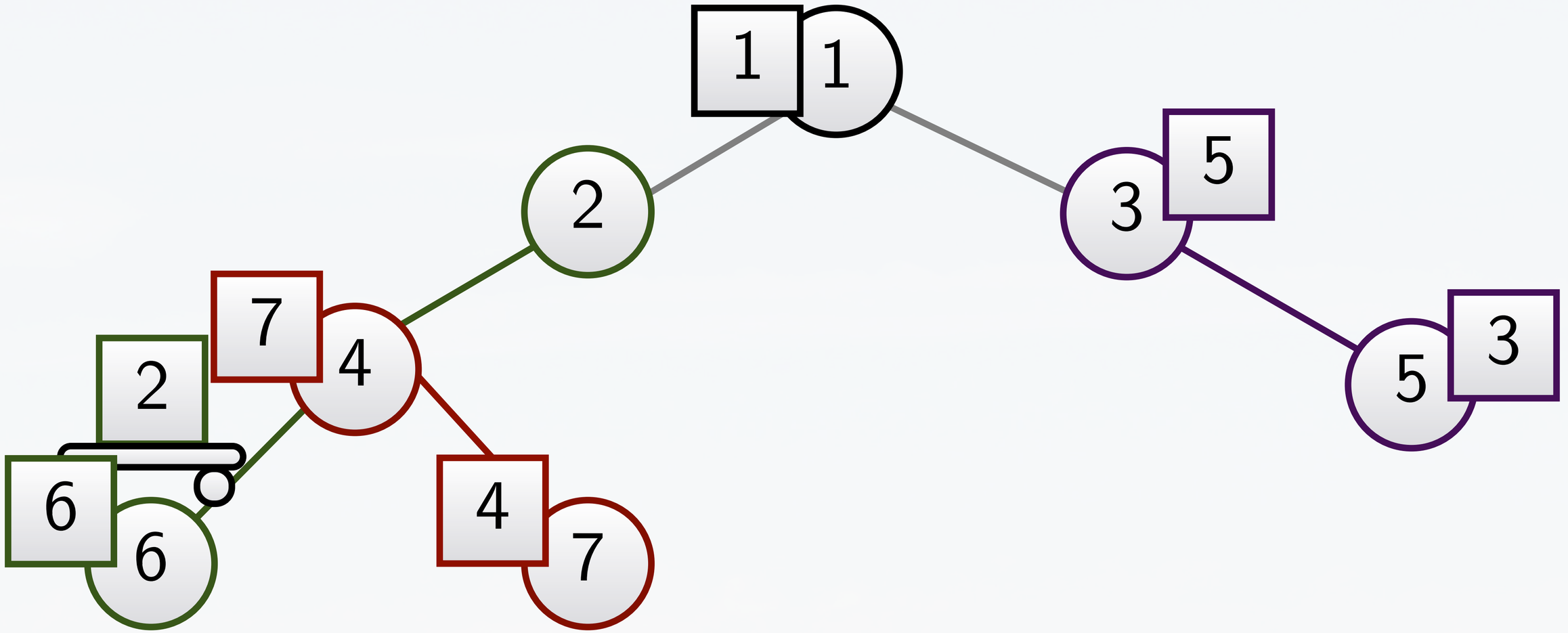




ETH zürich
Daniel Graf

# Fewest Swaps

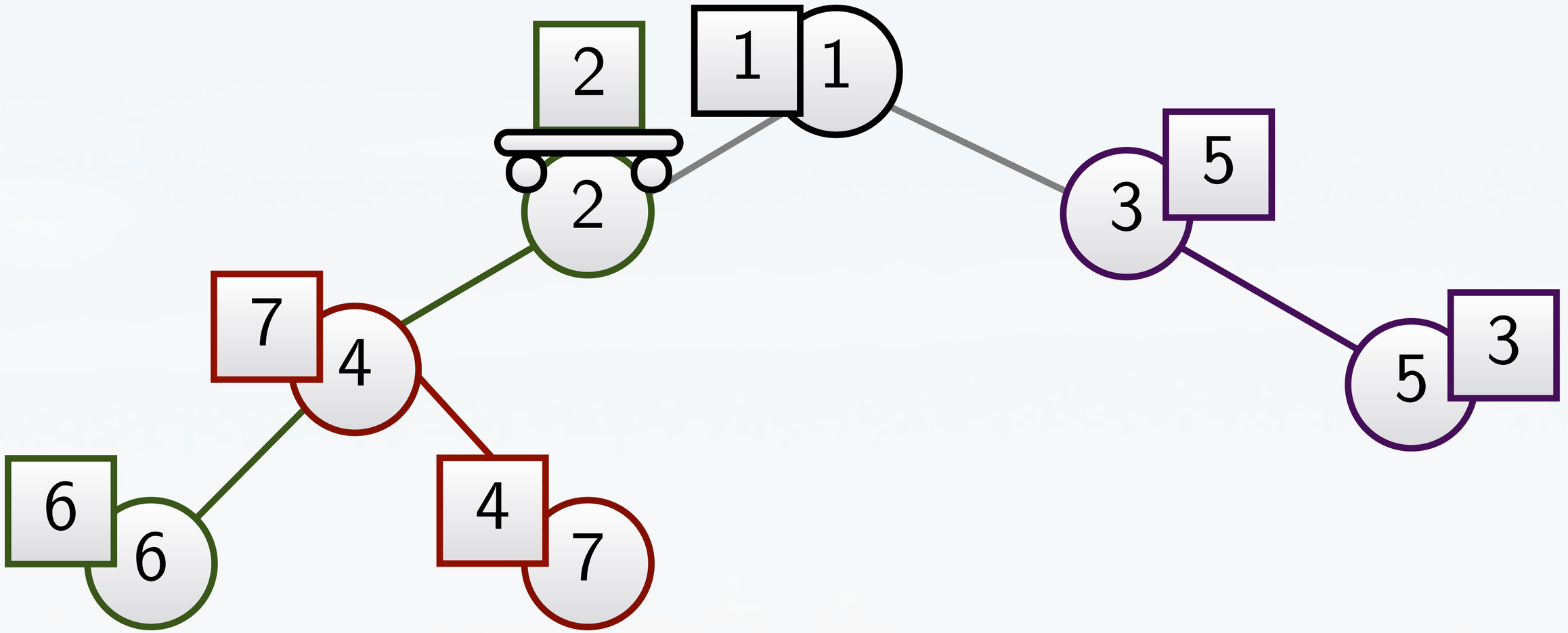




ETH zürich
Daniel Graf

# Fewest Swaps

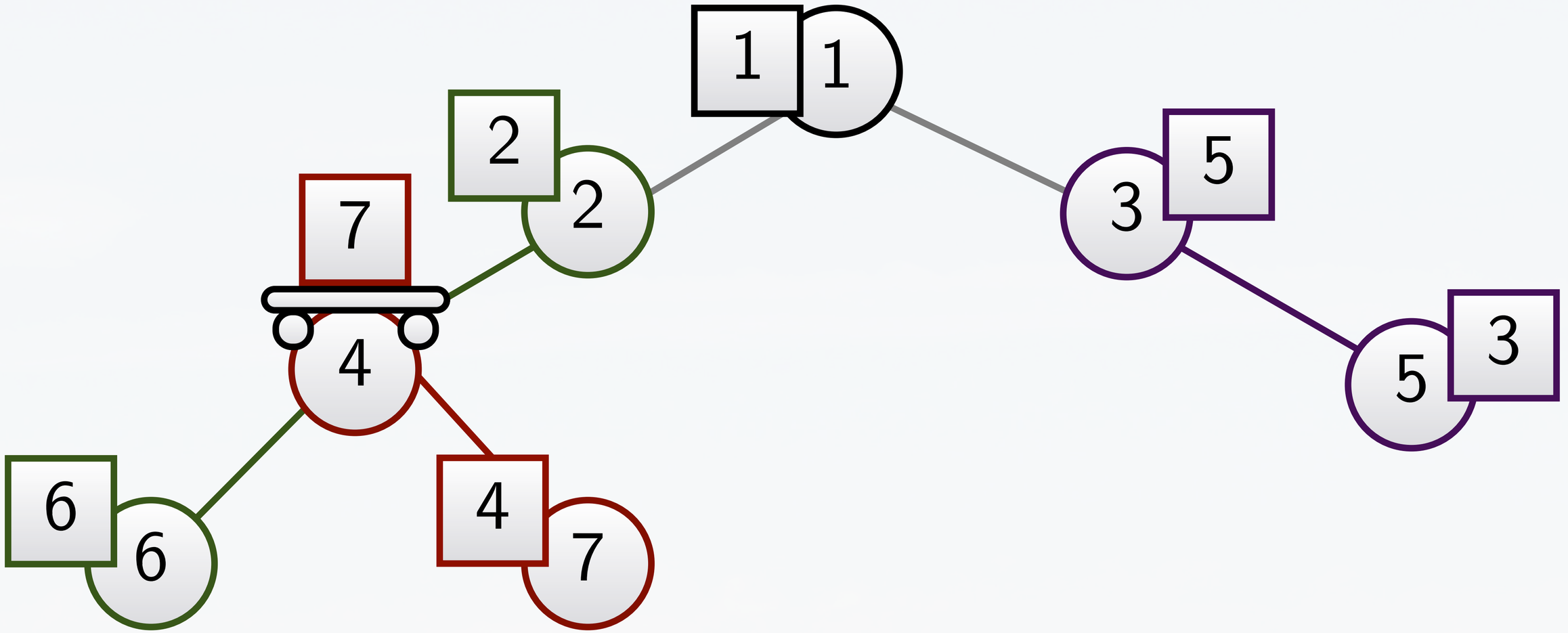




ETH zürich
Daniel Graf

# Fewest Swaps

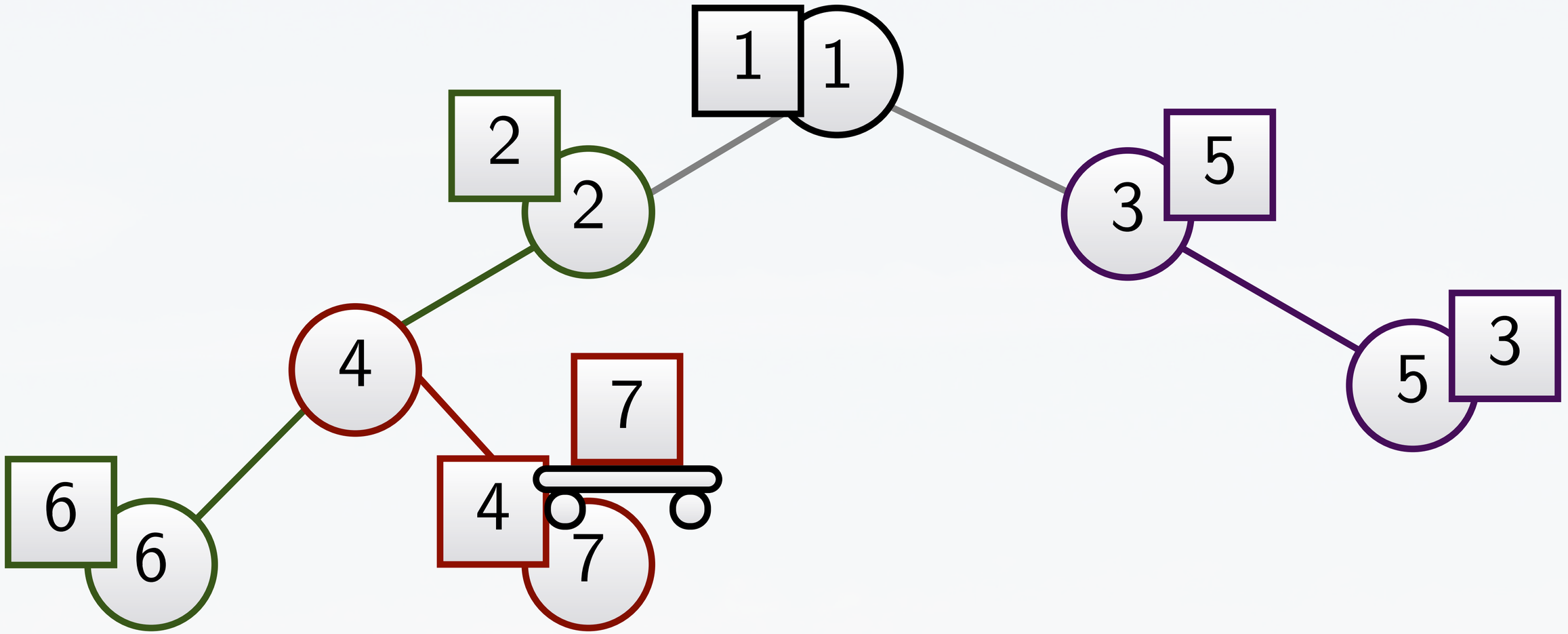




ETH zürich
Daniel Graf

# Fewest Swaps

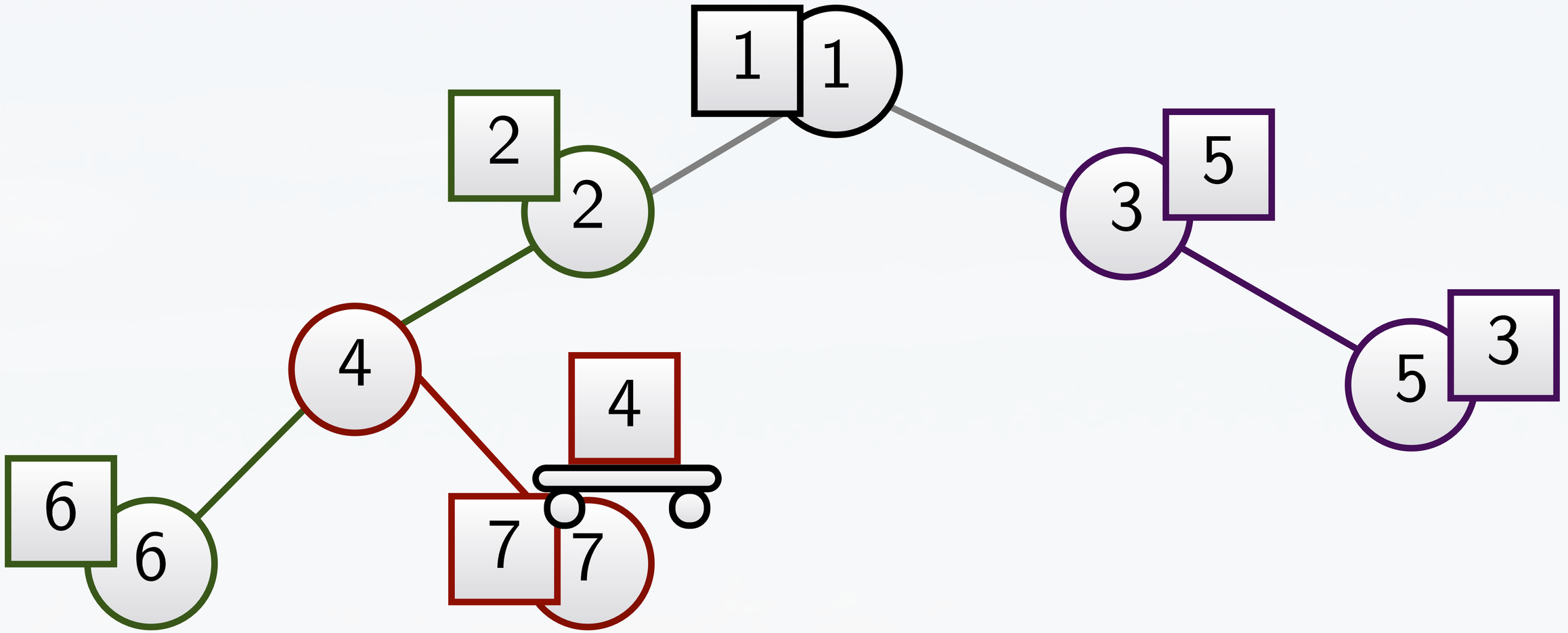




ETH zürich
Daniel Graf

# Fewest Swaps

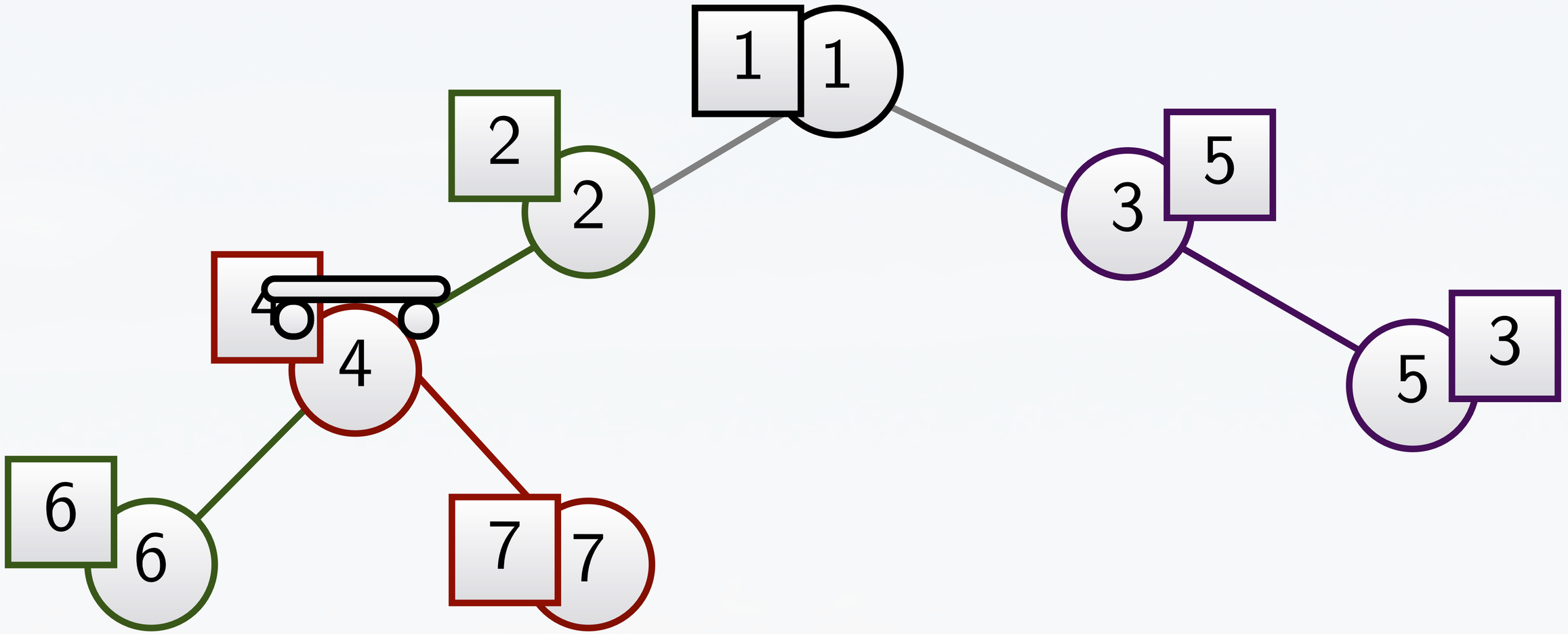




ETH zürich
Daniel Graf

# Fewest Swaps

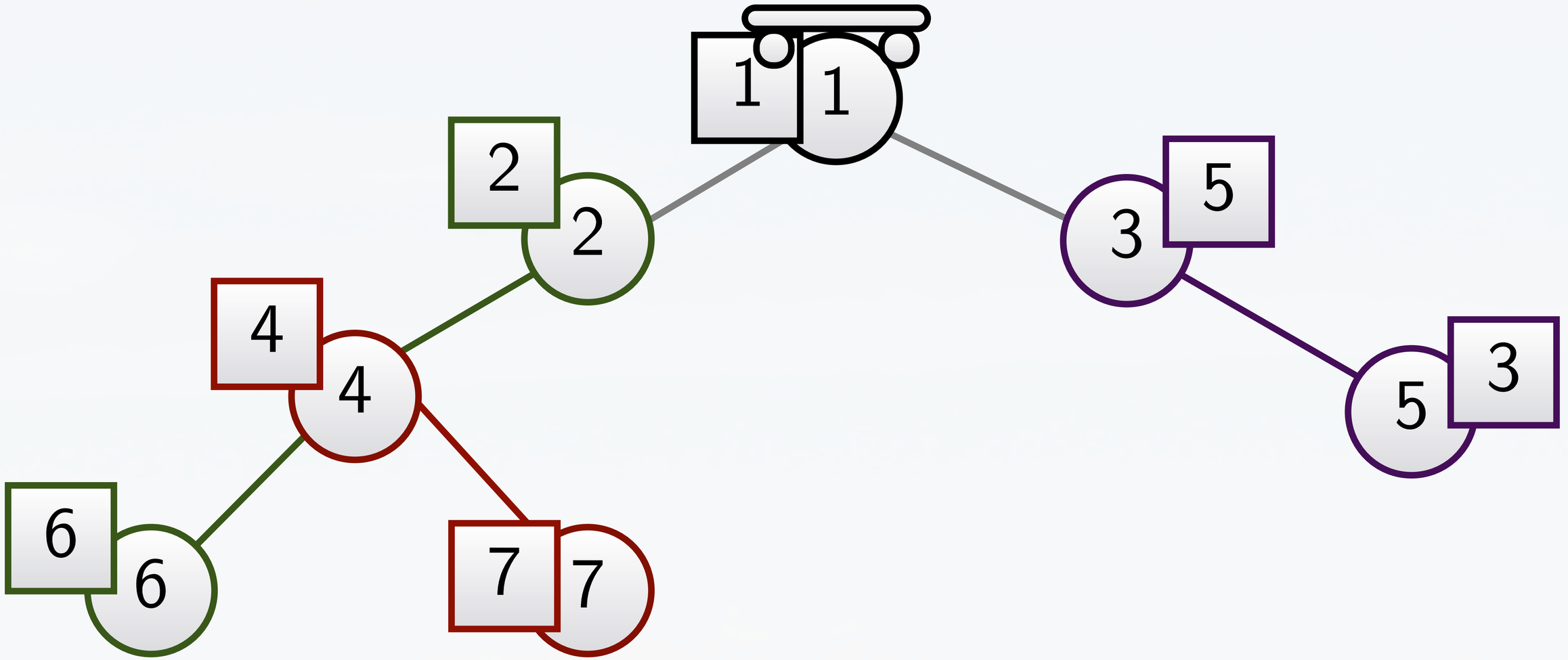




ETH zürich
Daniel Graf

# Fewest Swaps

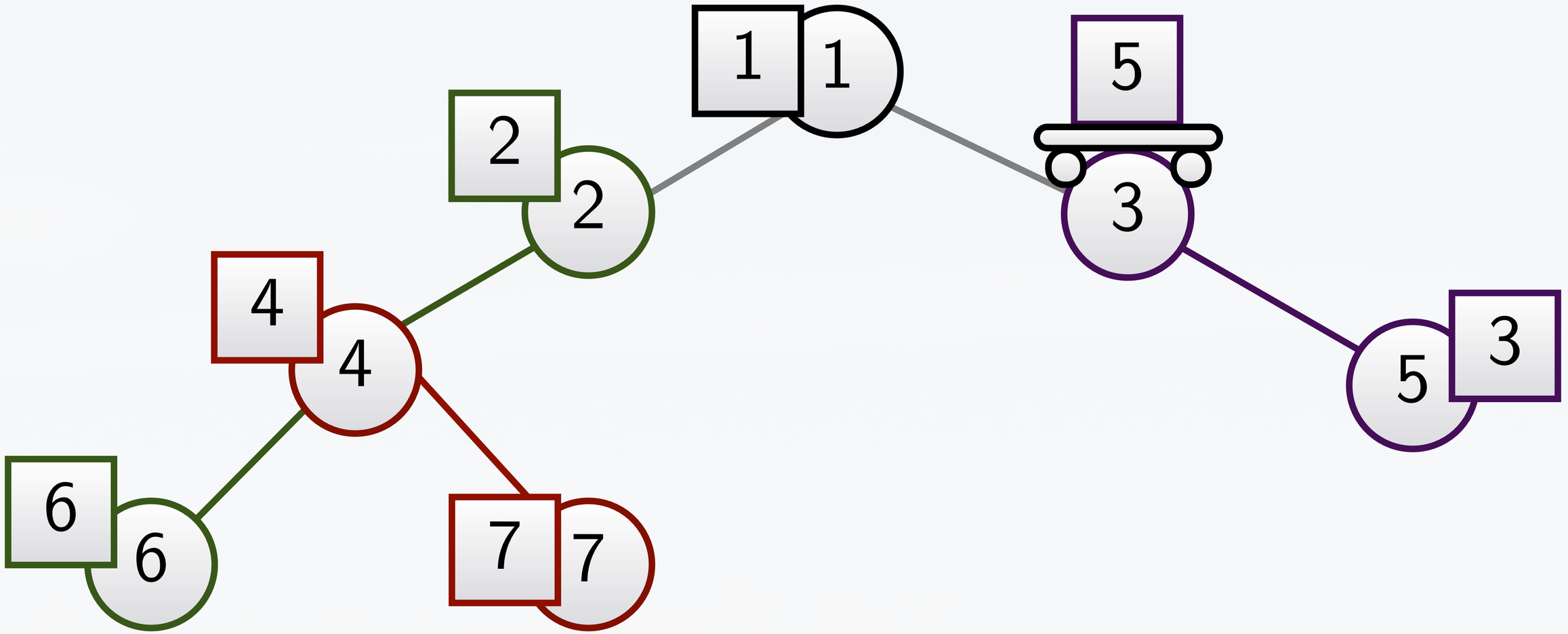




ETH zürich
Daniel Graf

# Fewest Swaps

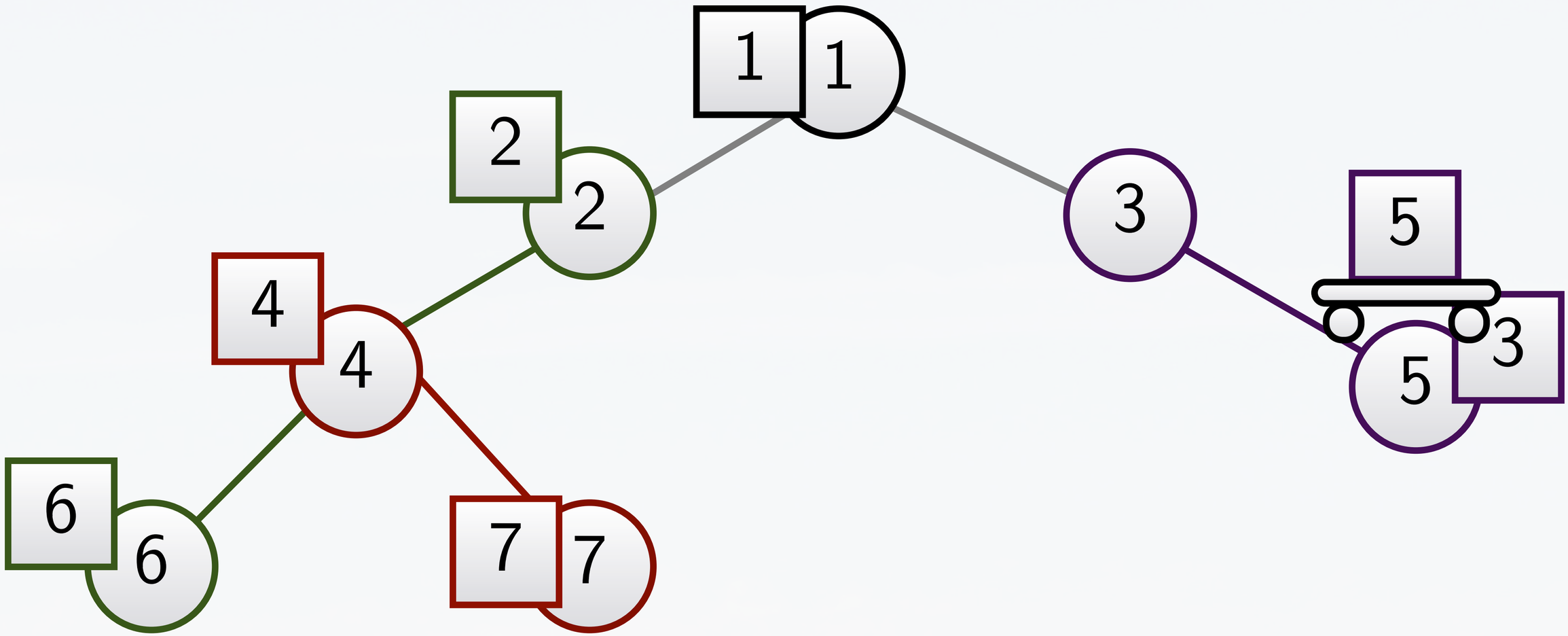




ETH zürich
Daniel Graf

# Fewest Swaps

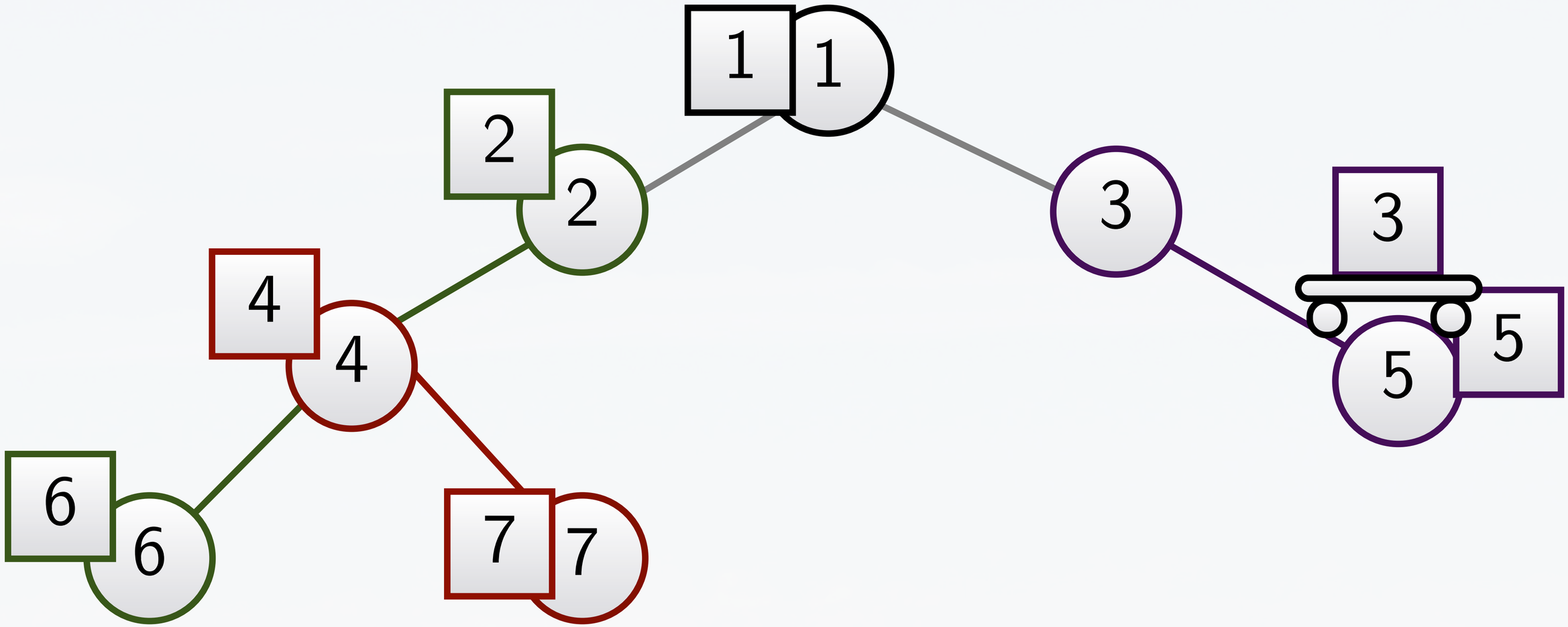
3. Sorting
3.3 Swap-Count



ETH zürich
Daniel Graf

# Fewest Swaps

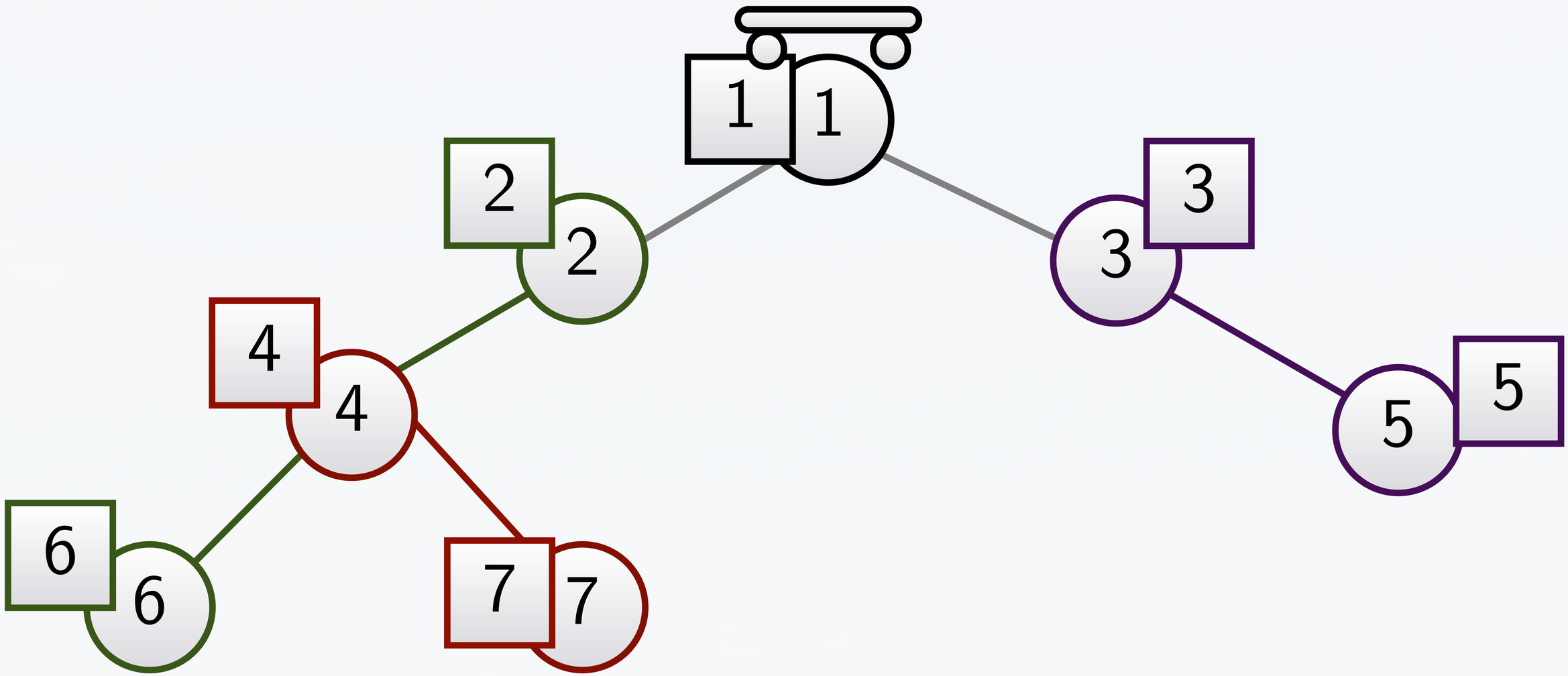




ETH zürich
Daniel Graf

# Fewest Swaps





# Fewest Swaps



*Theorem:* Finding a swap-optimal sorting walk on a tree is NP-complete.

ETH zürich
Daniel Graf

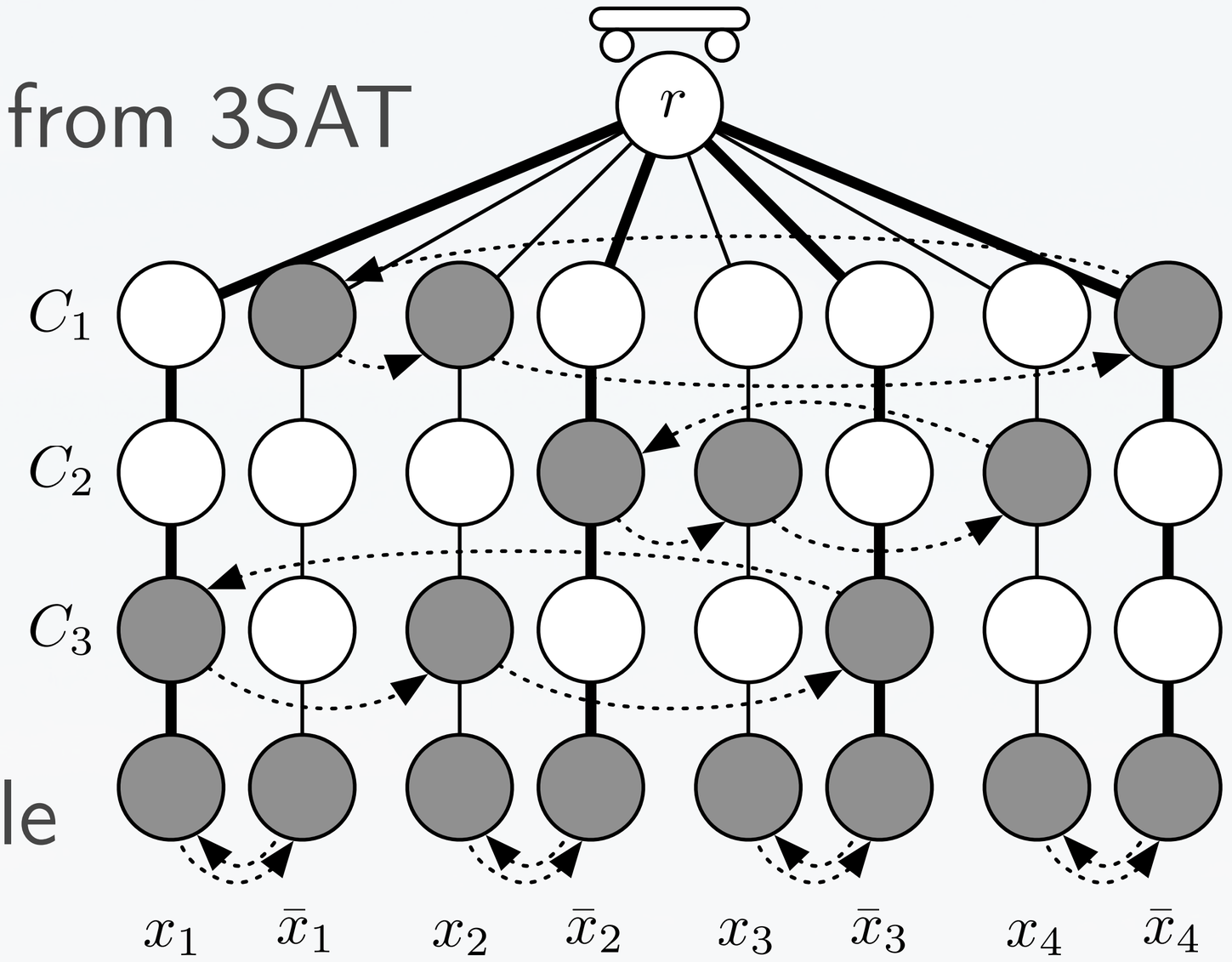
# Fewest Swaps



*Theorem:* Finding a swap-optimal sorting walk on a tree is NP-complete.

*Proof:* Reduction from 3SAT

- Spider graph
- One leg per literal
- One cycle per clause & variable



# Related Work



Sorting of physical objects

- using prefix reversals, stacks and queues, etc.

Sorting using a given set of possible swaps

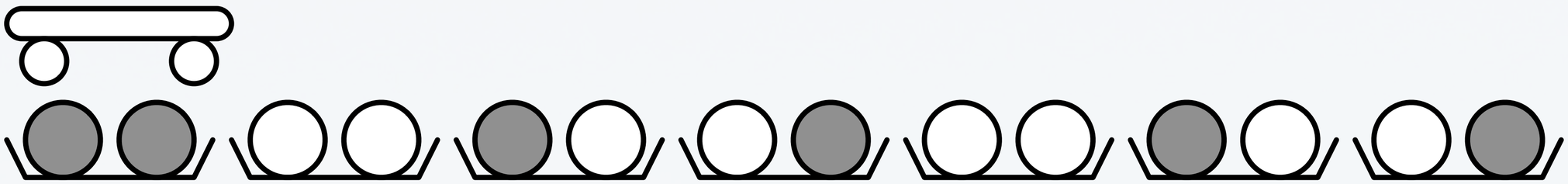
- swapping any pair [1849, Cayley]
- swapping adjacent pairs [1973, Knuth]
  - bubble sort is optimal (inversion number)
  - 2-approximation on trees [2014, Yamanaka et al.]

Group Steiner Tree Problem [1999, Ihler et al.]

# Task Kutije



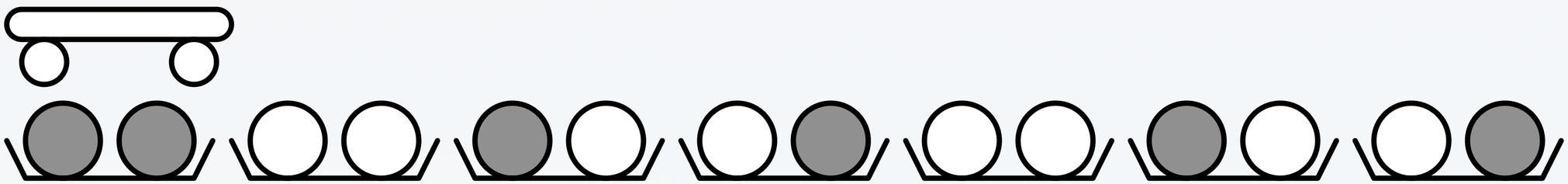
Coding competition task by Luka Kalinovcic, 2006



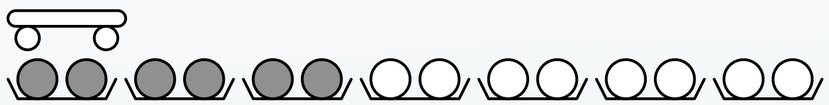
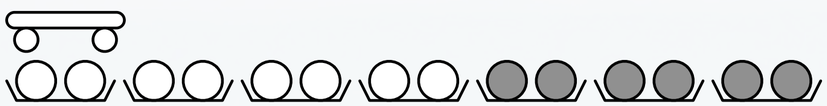
# Task Kutije



Coding competition task by Luka Kalinovcic, 2006



Robot that can carry up to two marbles at a time.

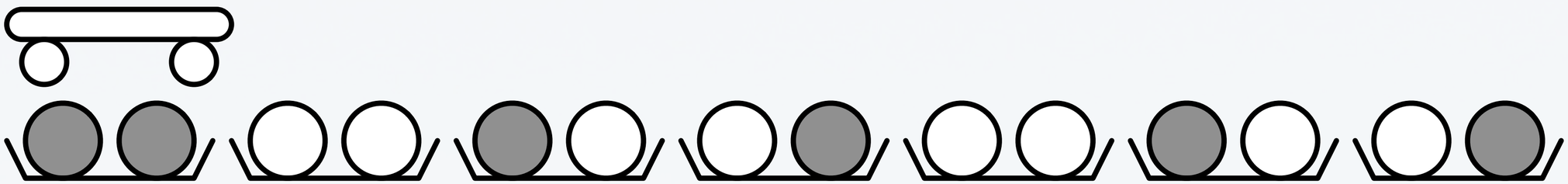
*Goal:*  or 

ETH zürich
Daniel Graf

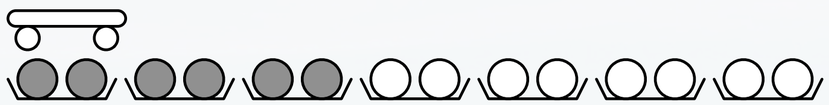
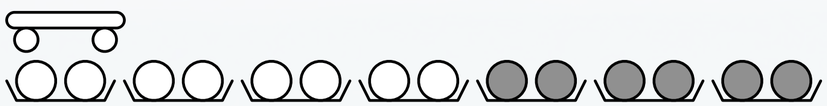
# Task Kutije



Coding competition task by Luka Kalinovcic, 2006



Robot that can carry up to two marbles at a time.

*Goal:*  or 

*Solution:* Build graph of lower-bound constraints. Find an Eulerian path through it. Use an essential-step-argument to show optimality.

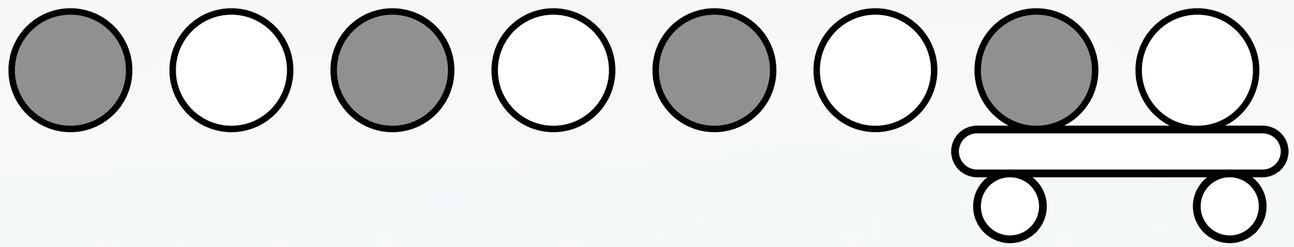
ETH zürich
Daniel Graf

# Tait's Puzzle



Coin Puzzle by Peter Tait (1890), ICPC WF 2014

In every move this robot can carry exactly two neighboring marbles for an arbitrary distance.

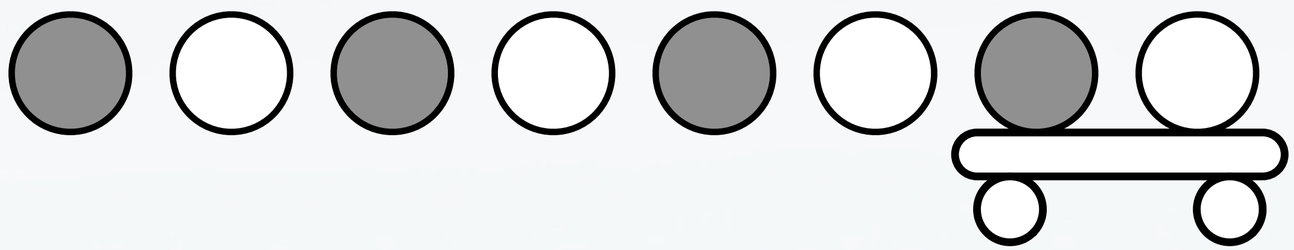


# Tait's Puzzle



Coin Puzzle by Peter Tait (1890), ICPC WF 2014

In every move this robot can carry exactly two neighboring marbles for an arbitrary distance.



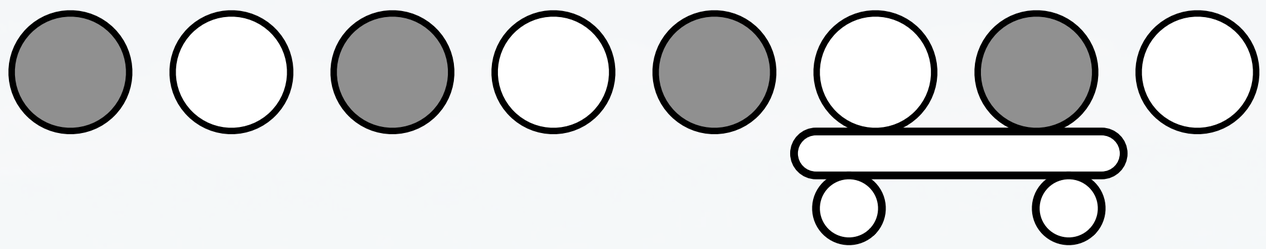
*Solution:* Show inductively that it is always possible to do in $n$ steps for $n$ pairs of marbles.

# Tait's Puzzle



Coin Puzzle by Peter Tait (1890), ICPC WF 2014

In every move this robot can carry exactly two neighboring marbles for an arbitrary distance.



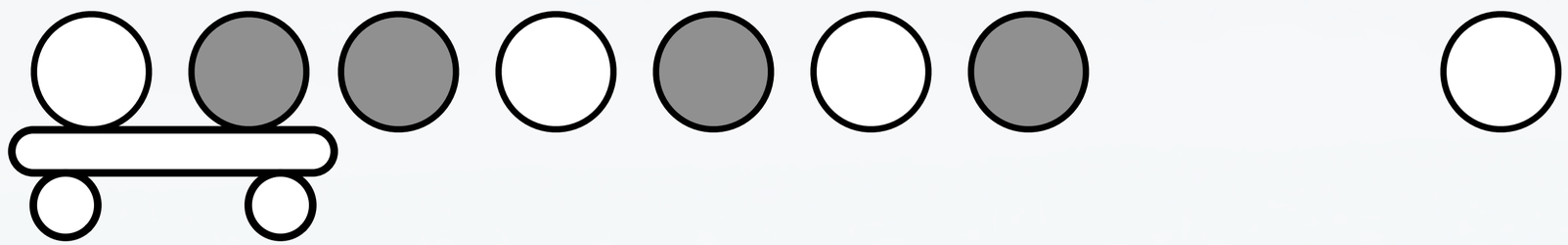
*Solution:* Show inductively that it is always possible to do in $n$ steps for $n$ pairs of marbles.

# Tait's Puzzle



Coin Puzzle by Peter Tait (1890), ICPC WF 2014

In every move this robot can carry exactly two neighboring marbles for an arbitrary distance.



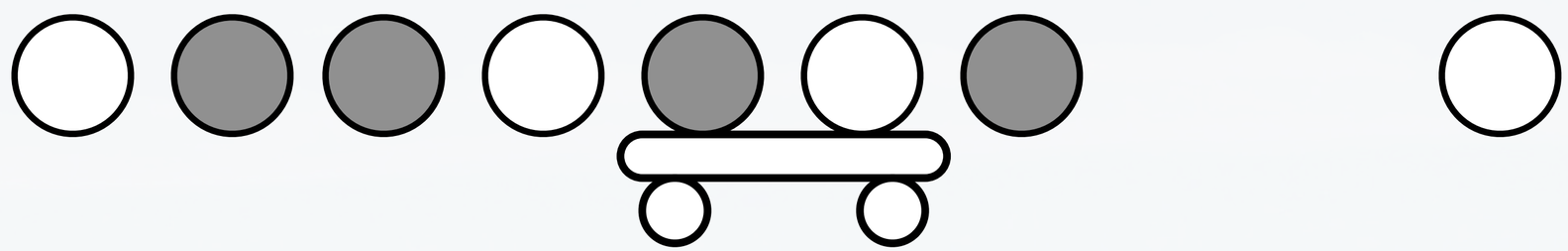
*Solution:* Show inductively that it is always possible to do in $n$ steps for $n$ pairs of marbles.

# Tait's Puzzle



Coin Puzzle by Peter Tait (1890), ICPC WF 2014

In every move this robot can carry exactly two neighboring marbles for an arbitrary distance.



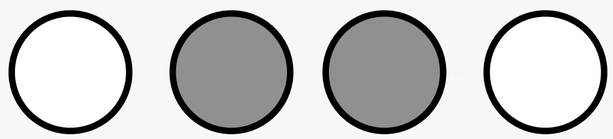
*Solution:* Show inductively that it is always possible to do in $n$ steps for $n$ pairs of marbles.

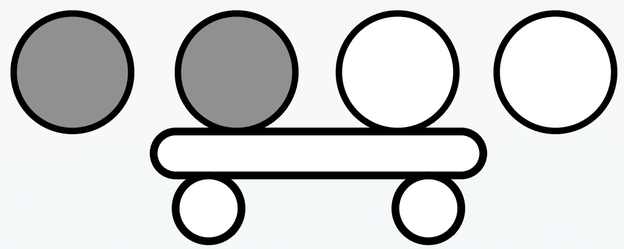
# Tait's Puzzle



Coin Puzzle by Peter Tait (1890), ICPC WF 2014

In every move this robot can carry exactly two neighboring marbles for an arbitrary distance.





*Solution:* Show inductively that it is always possible to do in $n$ steps for $n$ pairs of marbles.

# Tait's Puzzle



Coin Puzzle by Peter Tait (1890), ICPC WF 2014

In every move this robot can carry exactly two neighboring marbles for an arbitrary distance.

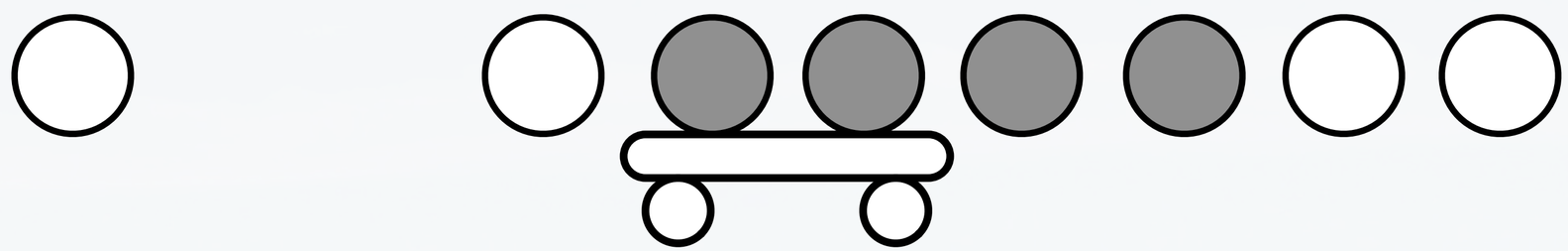
*Solution:* Show inductively that it is always possible to do in $n$ steps for $n$ pairs of marbles.

# Tait's Puzzle



Coin Puzzle by Peter Tait (1890), ICPC WF 2014

In every move this robot can carry exactly two neighboring marbles for an arbitrary distance.



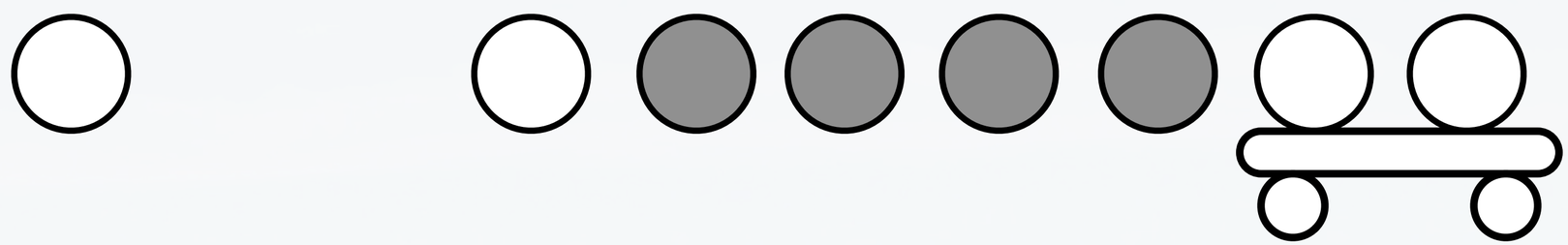
*Solution:* Show inductively that it is always possible to do in $n$ steps for $n$ pairs of marbles.

# Tait's Puzzle



Coin Puzzle by Peter Tait (1890), ICPC WF 2014

In every move this robot can carry exactly two neighboring marbles for an arbitrary distance.



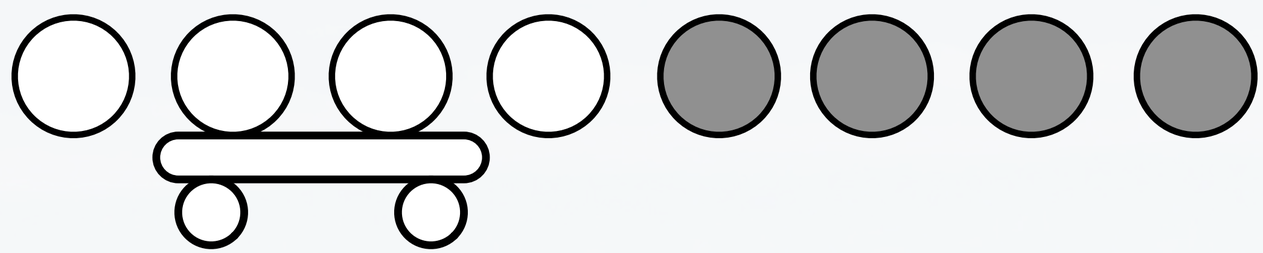
*Solution:* Show inductively that it is always possible to do in $n$ steps for $n$ pairs of marbles.

# Tait's Puzzle



Coin Puzzle by Peter Tait (1890), ICPC WF 2014

In every move this robot can carry exactly two neighboring marbles for an arbitrary distance.



*Solution:* Show inductively that it is always possible to do in $n$ steps for $n$ pairs of marbles.

ETH zürich
Daniel Graf

# Other Systems

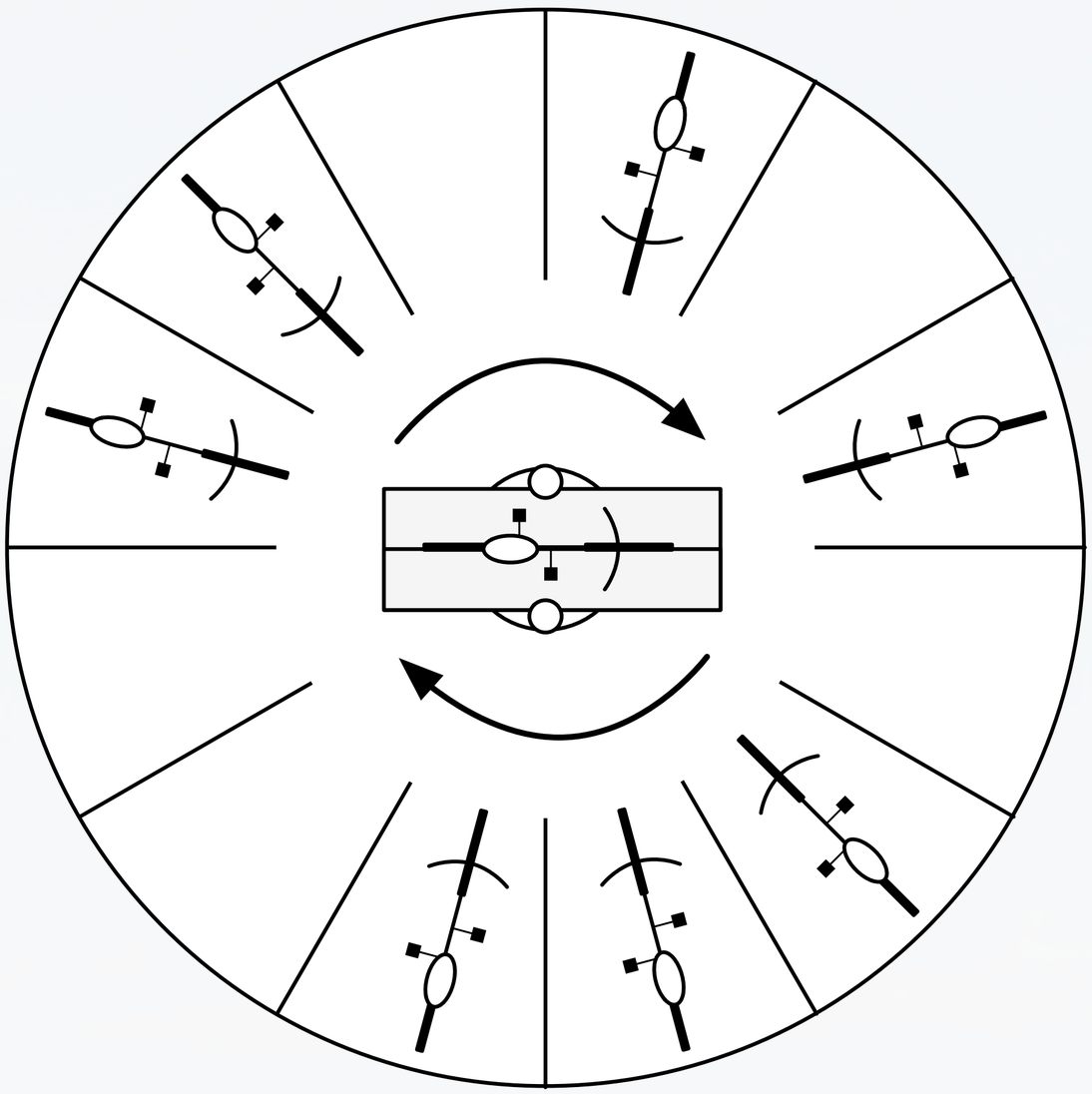
## What else did people try for automated parking?

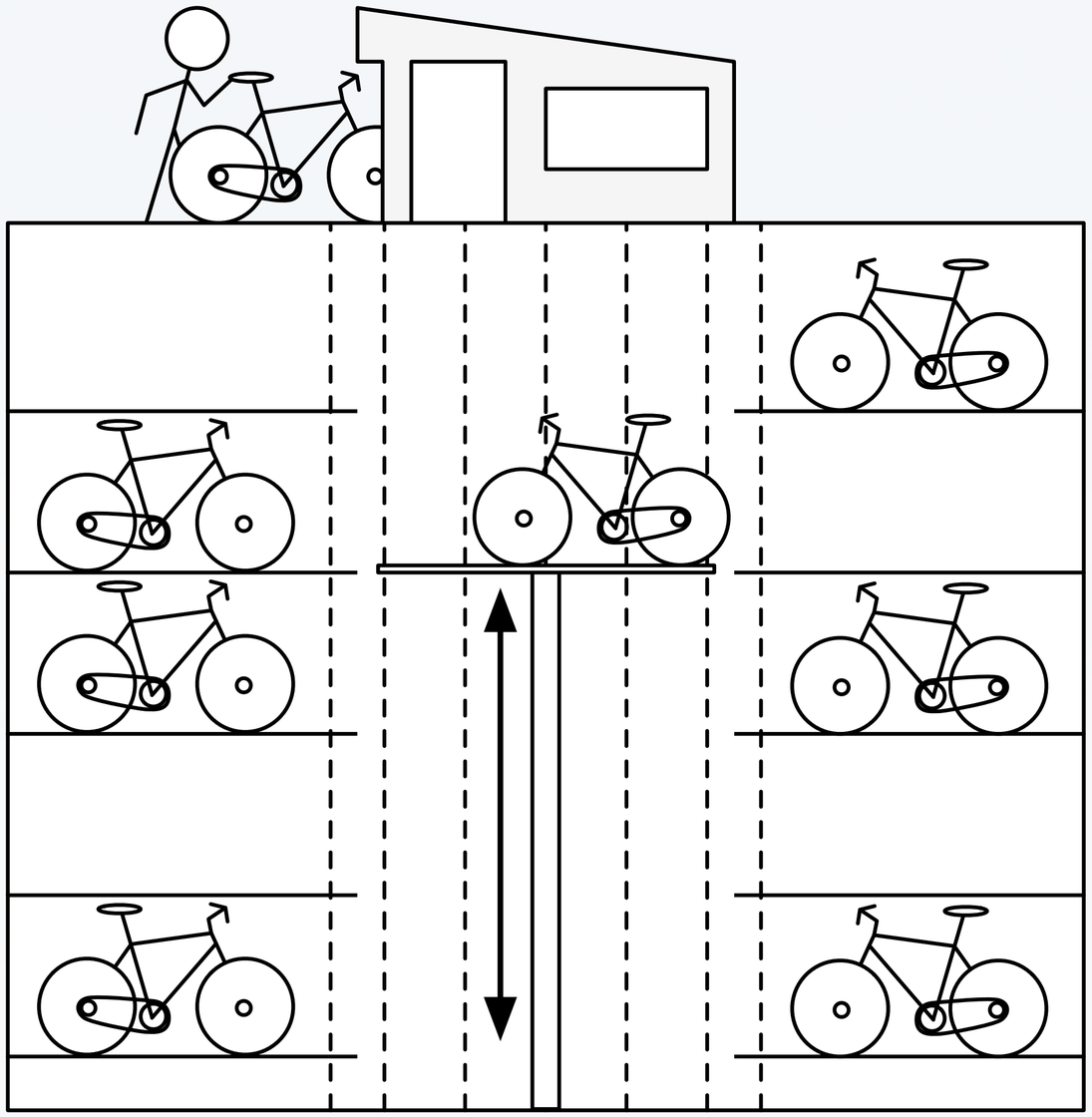
ETH zürich
Daniel Graf

# Cylindrical



Vertical lift with circular storage slots around it





# ECO Cycle



## Example: ECO Cycle in Tokyo by Giken Seisakusho


Video by Giken Seisakusho Co. LTD.

# Züri West



## Example: Auto Züri West by Skyline Parking

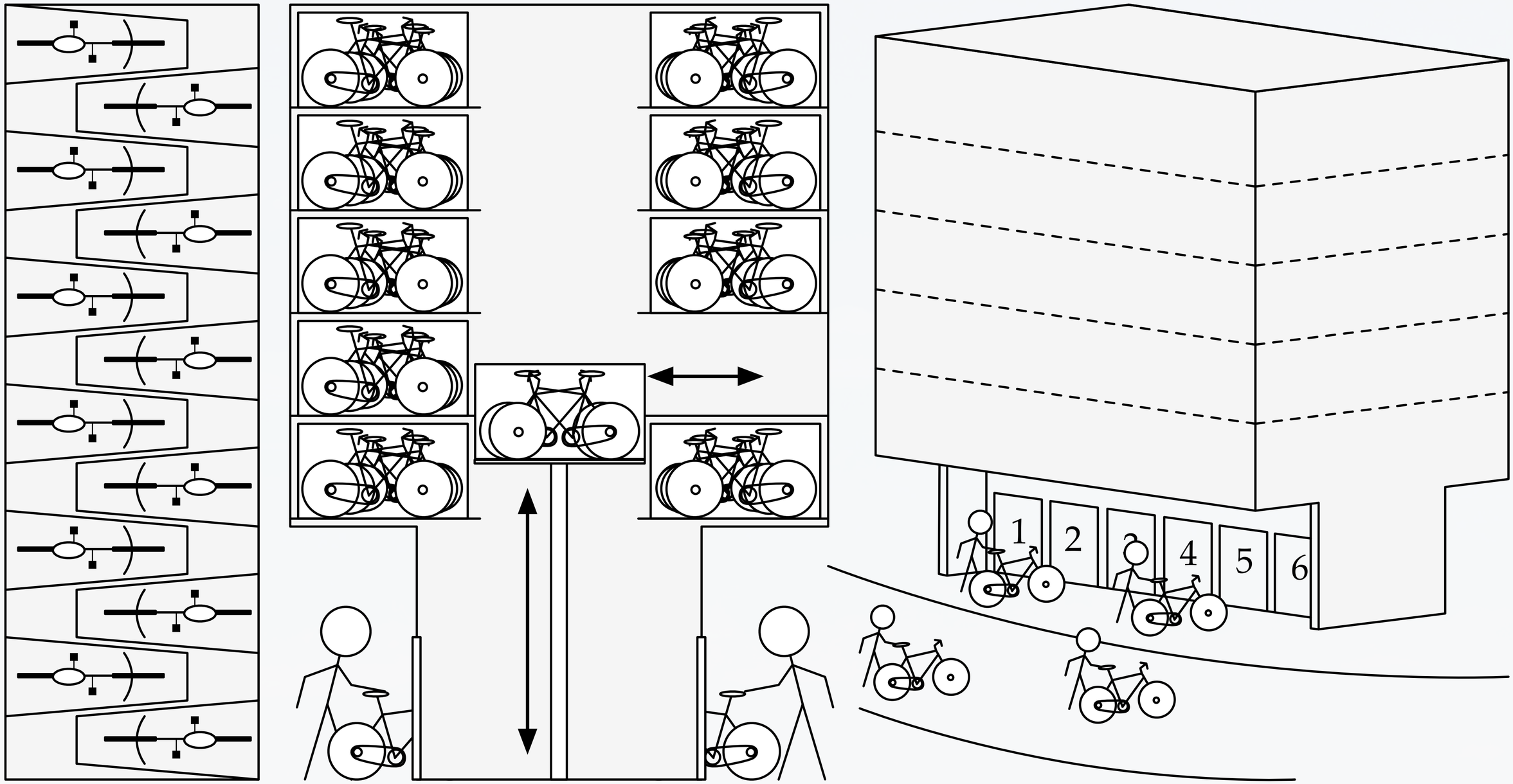

Video by Skyline Parking AG

ETH zürich
Daniel Graf

# Pallet Based



## Vertical lift for pallets of 12 bicycles



# Radhaus



## Example: Radhaus in Offenburg (D)

# Radhaus



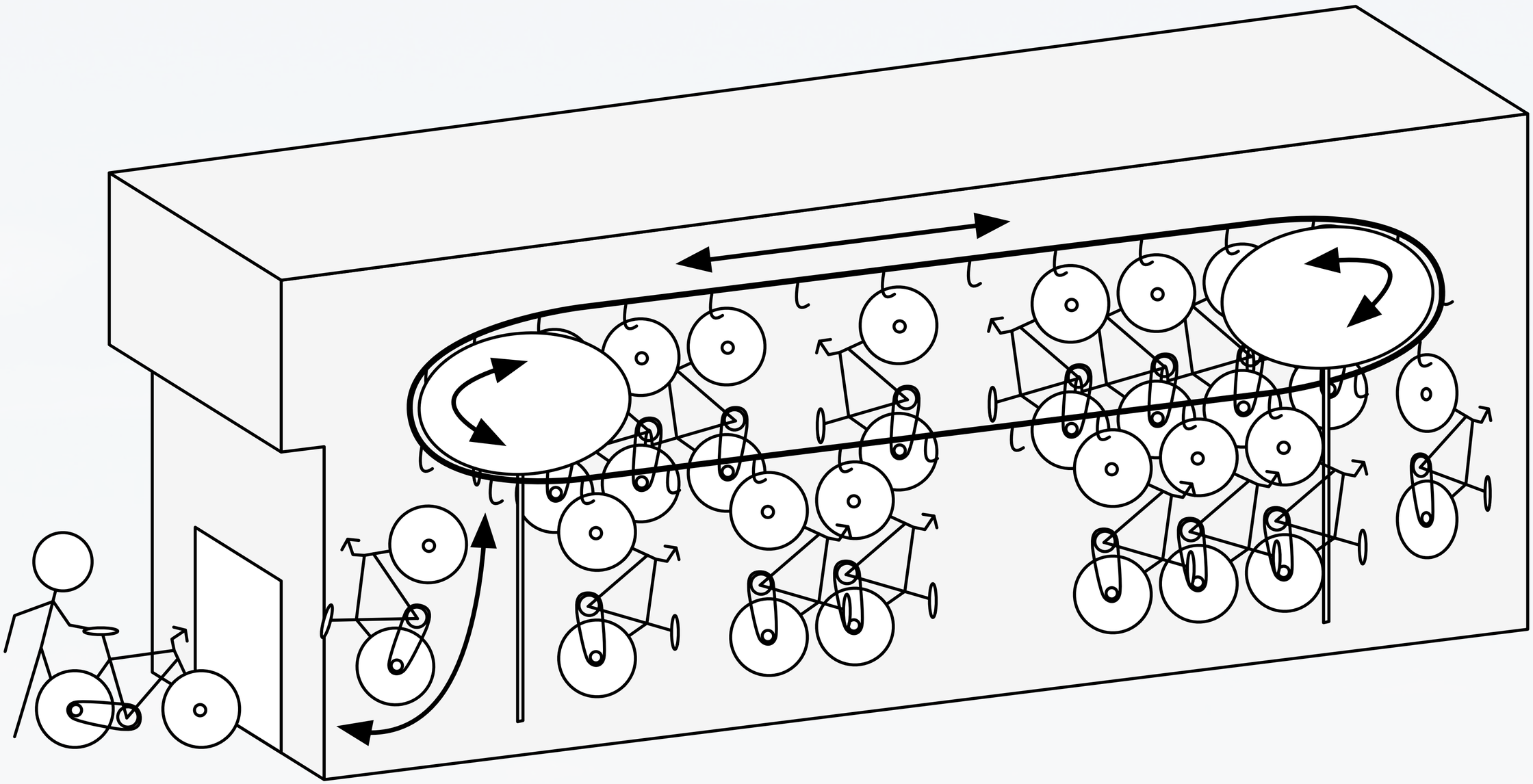
## Example: Radhaus in Offenburg (D)

ETH zürich
Daniel Graf

# Conveyor Belt



Cyclic chain of hooks, one central access point



# VeloMinck



## Example: VeloMinck in the Netherlands

Video by Lo Minck Systemen BV

# VeloMinck



## Example: VeloMinck in the Netherlands

Video by Lo Minck Systemen BV

ETH zürich
Daniel Graf

# Car Paternoster



## Example: Paternoster Car Parking in Chicago 1932

ETH zürich
Daniel Graf

# Conclusion



# Contributions



Scheduling Problem

- algorithms for arrival-only with a single door
- *NP*-hardness of the two-door problem
- *NP*-hardness of the all-day problem

Sorting Problem

- algorithms for sorting on paths, trees and cycles
- *NP*-hardness for planar graphs
- *NP*-hardness for trees when minimizing swaps

# Future Work



Open research questions

- approximation algorithms for two-door setting
- sorting problem on other graph classes
- further robot variations, multiple robots
- testing these models with real usage data

ETH zürich
Daniel Graf

# Questions



ETH zürich

Daniel Graf

# Thank you for your attention!

ETH zürich