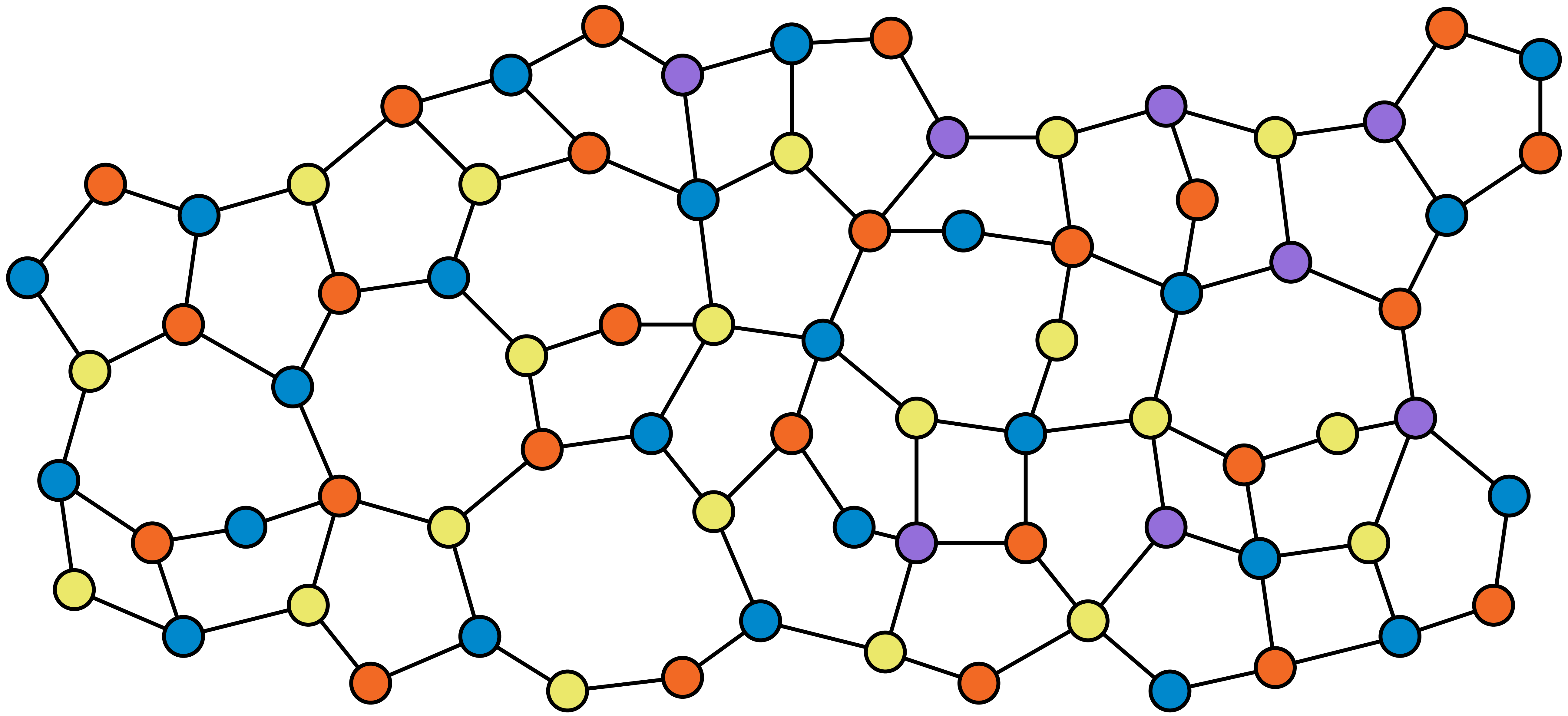
Distributed Coloring and MIS (part I)

Distributed systems

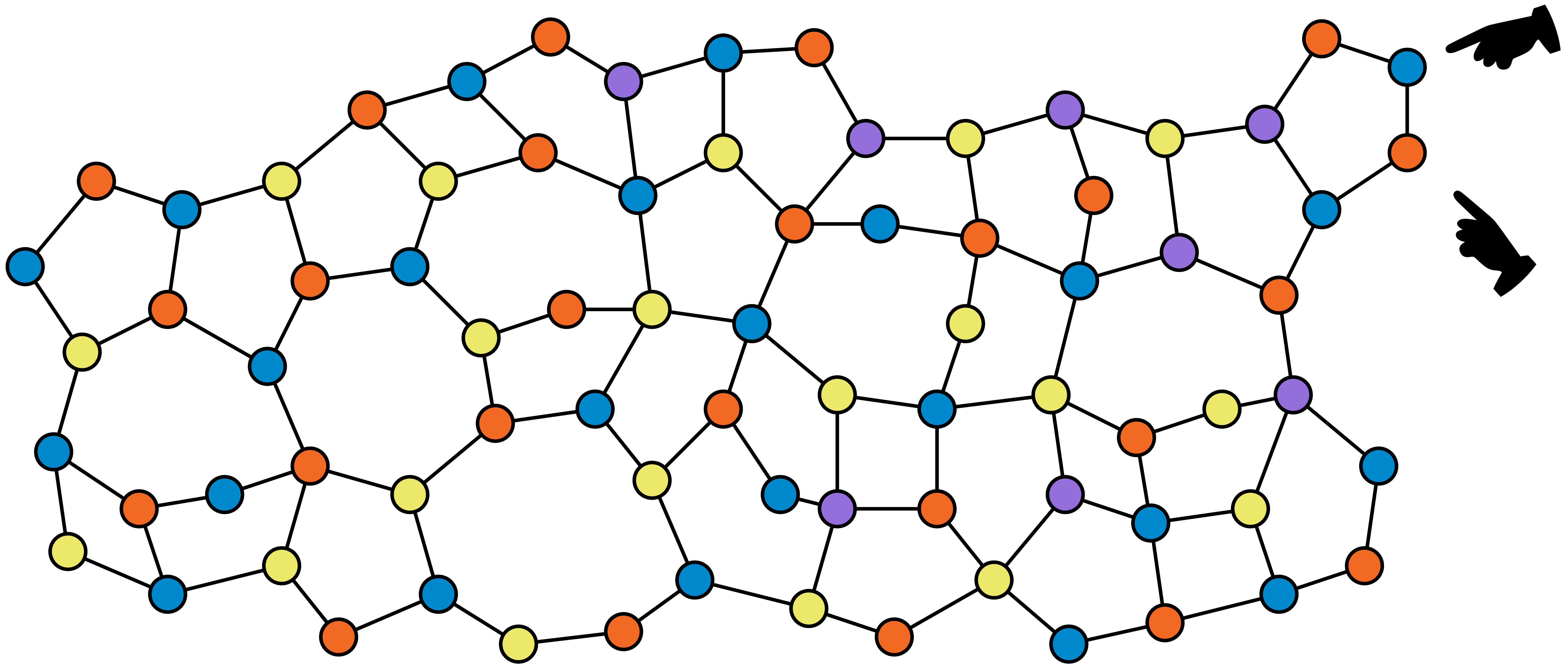
Alkida Balliu
University of Freiburg

Vertex coloring



Objective: Assign a color to each node such that:

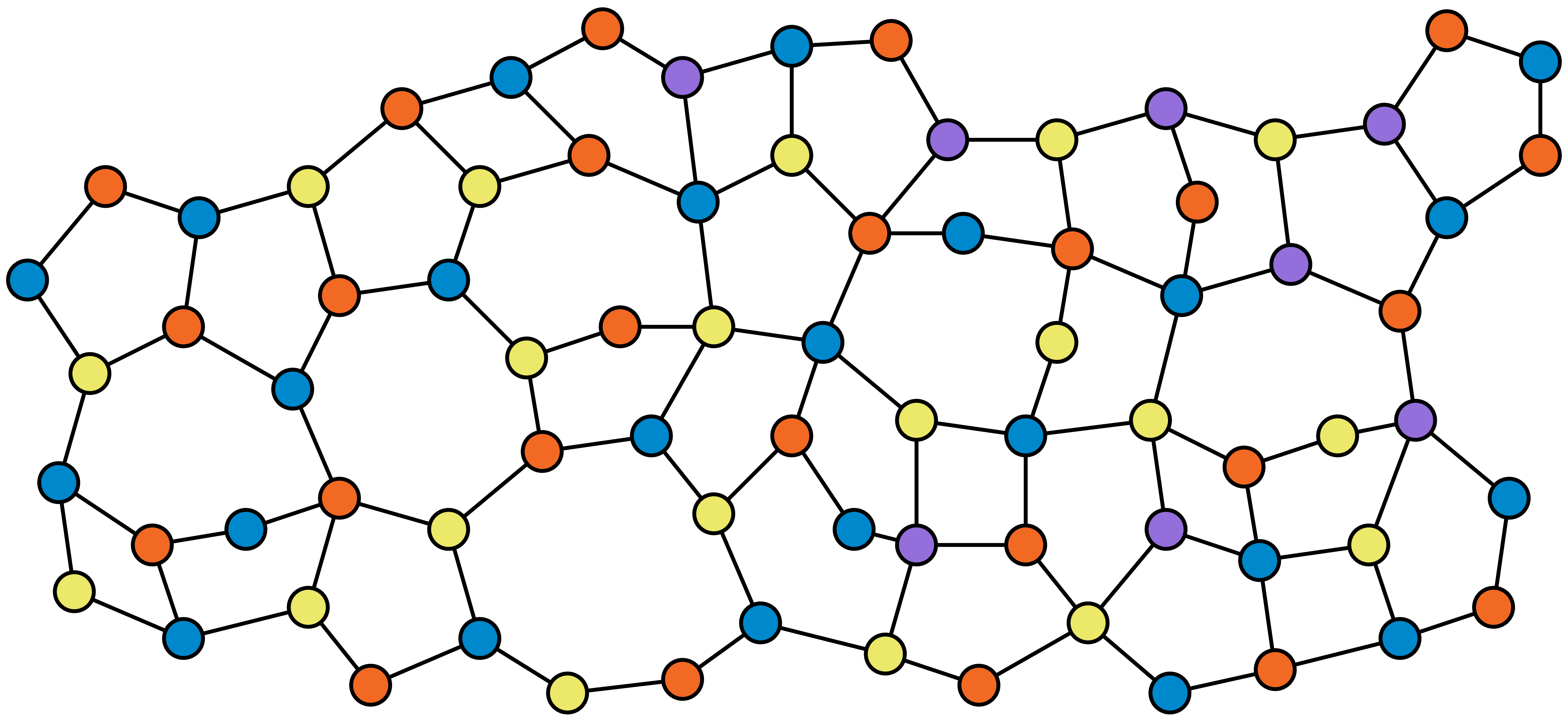
Vertex coloring



Objective: Assign a color to each node such that:

- **Neighbouring nodes** get **different colors**

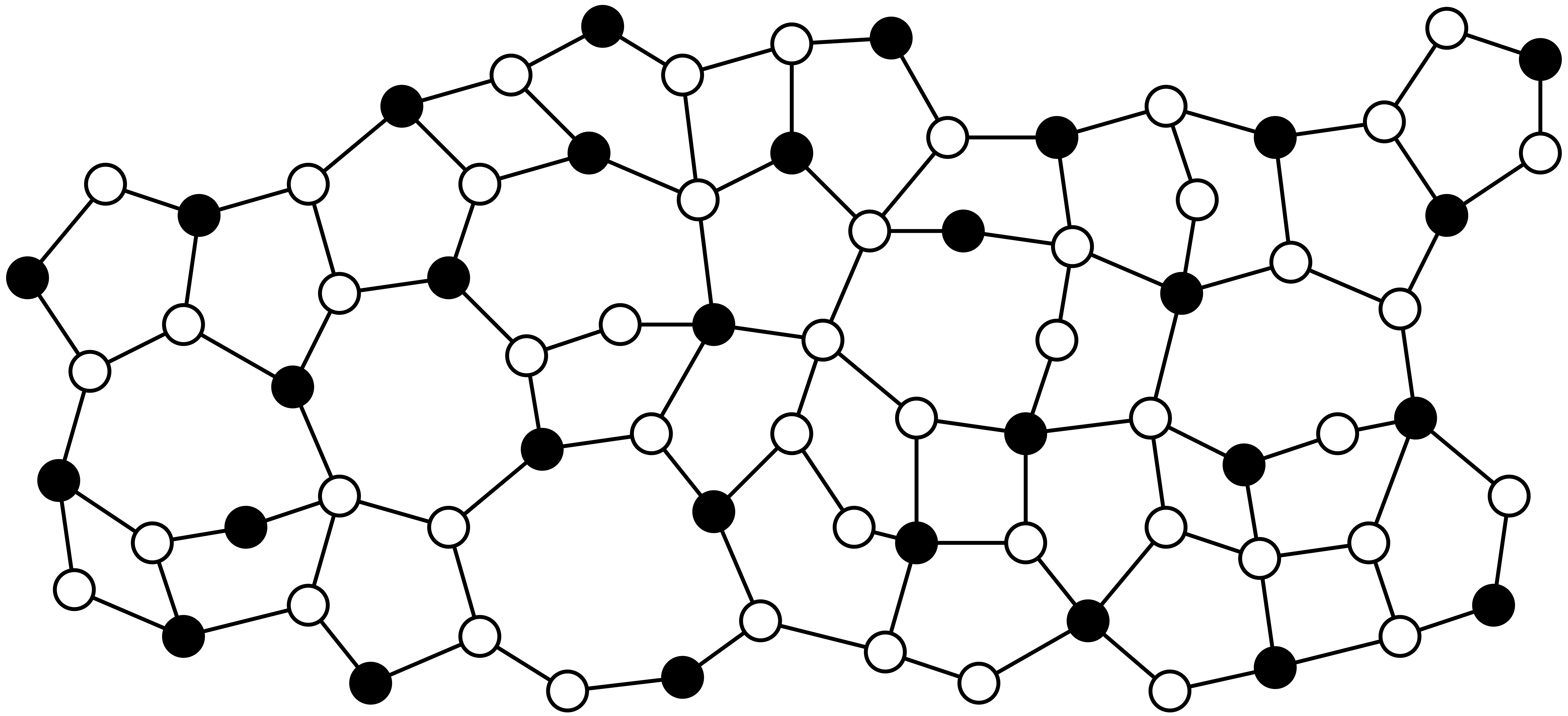
Vertex coloring



Objective: Assign a color to each node such that:

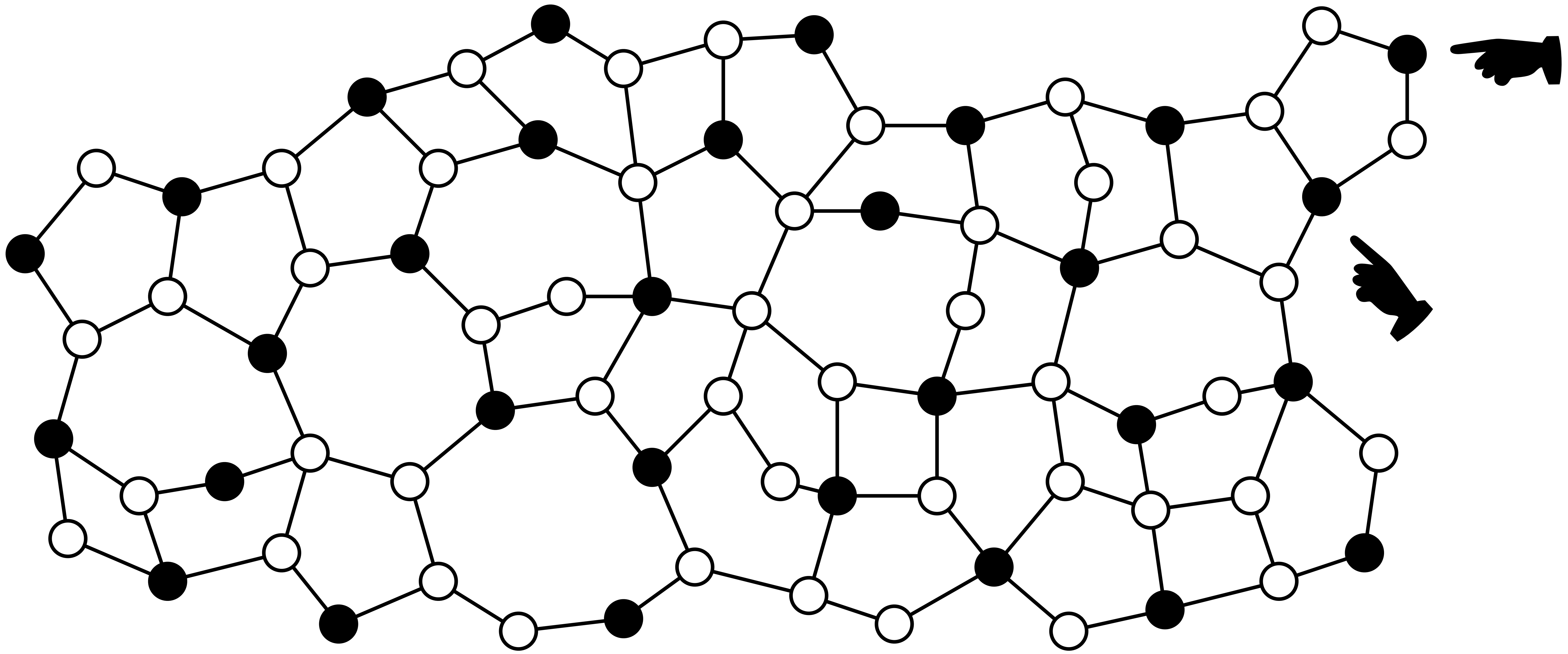
- **Neighbouring nodes** get **different colors**
- The total **number of** different **colors** is as **small** as possible

Maximal independent set (MIS)



Objective: Select nodes such that:

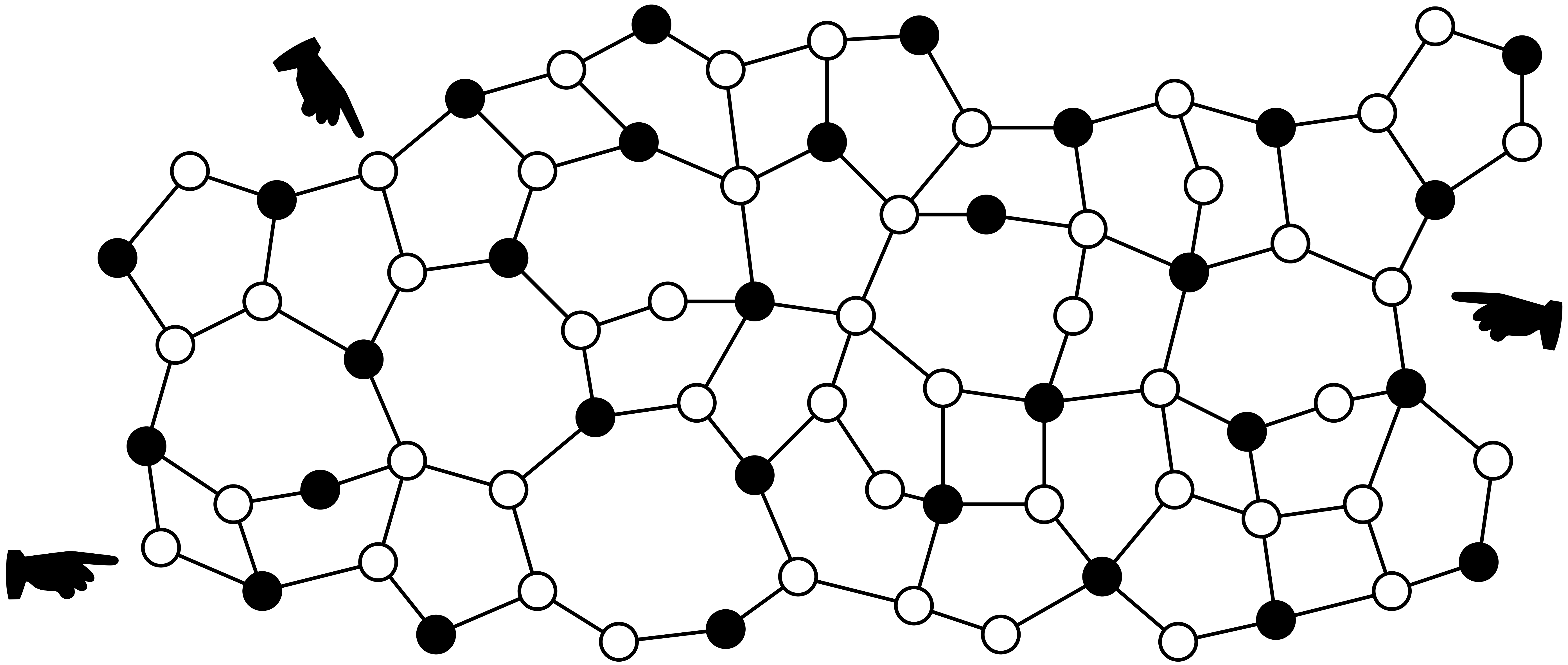
Maximal independent set (MIS)



Objective: Select nodes such that:

- Selected nodes form an **independent set** (they are not neighbors)

Maximal independent set (MIS)

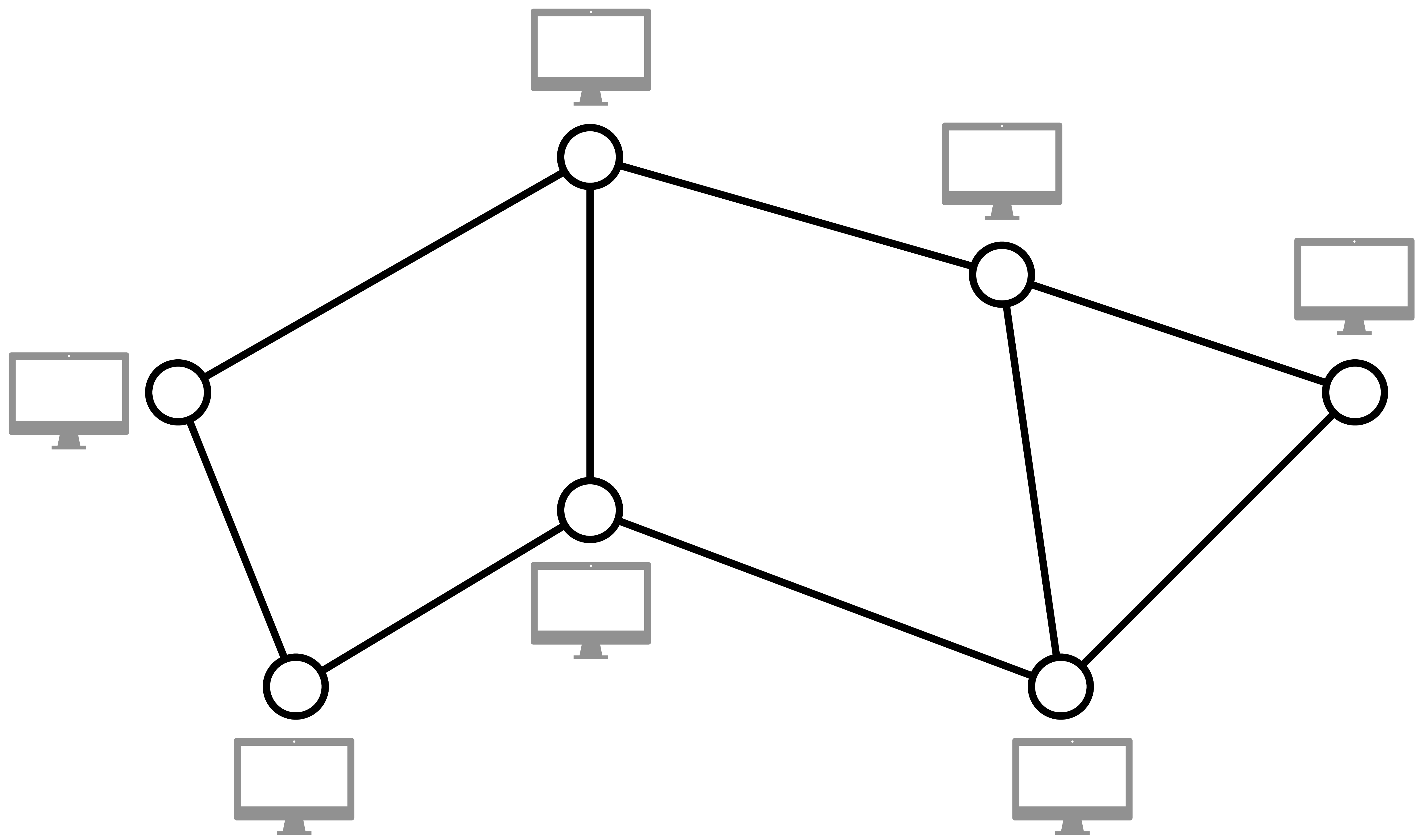


Objective: Select nodes such that:

- Selected nodes form an **independent set** (they are not neighbors)
- The independent set is **maximal** (any non-selected node has at least one neighbor that is selected)

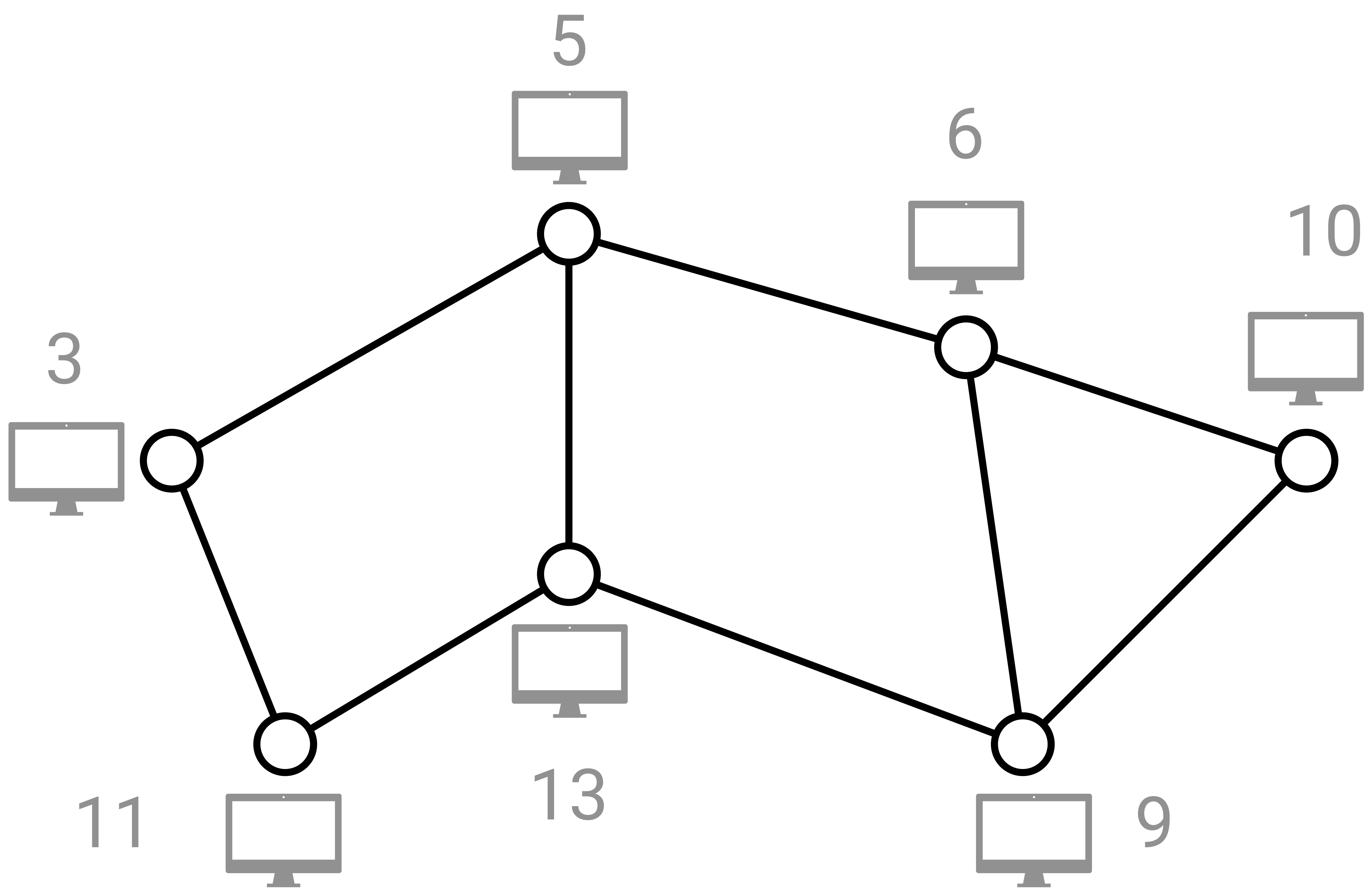
Distributed graph coloring

- The **network** is modeled as a **graph**



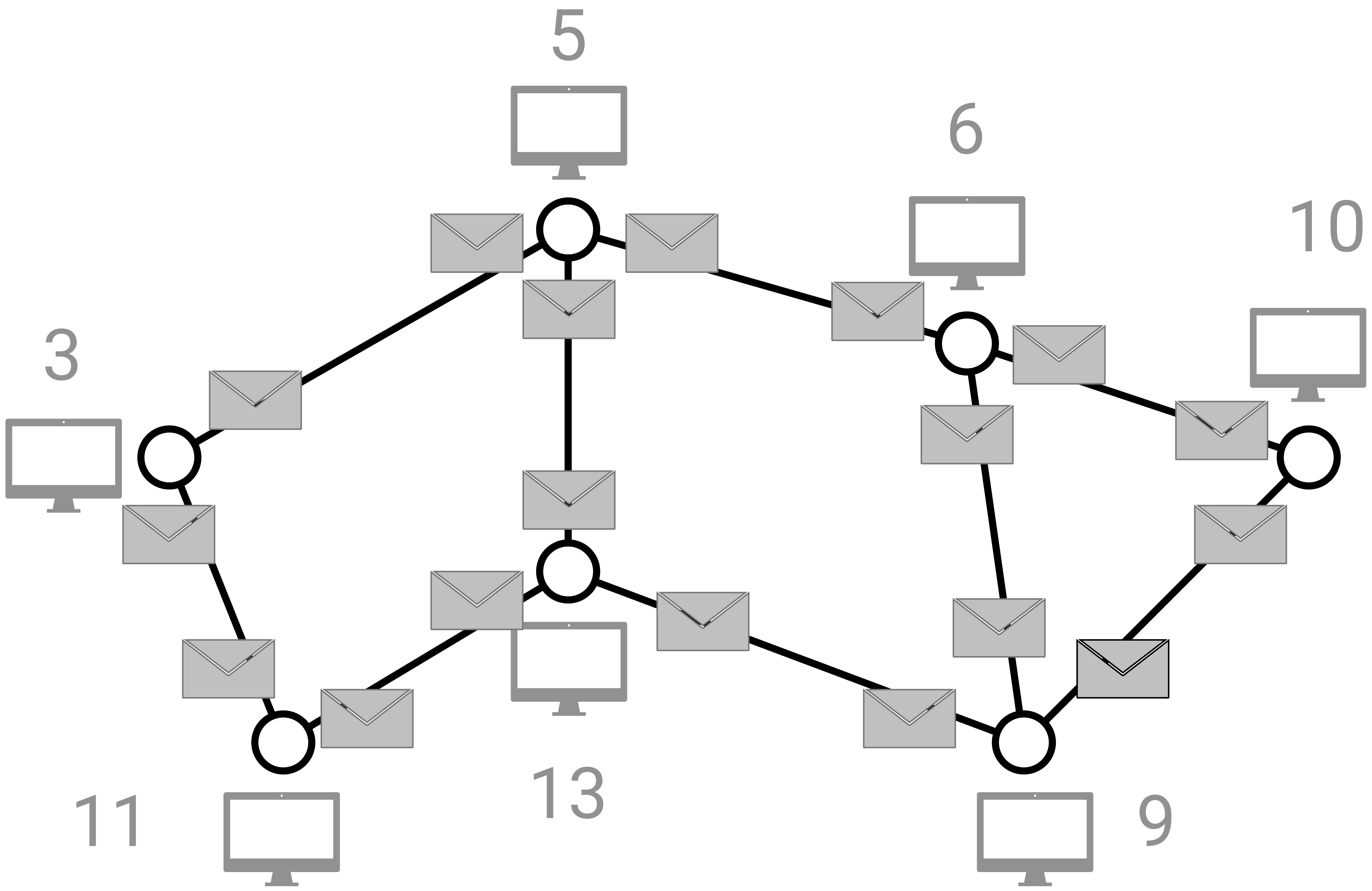
Distributed graph coloring

- The **network** is modeled as a **graph**



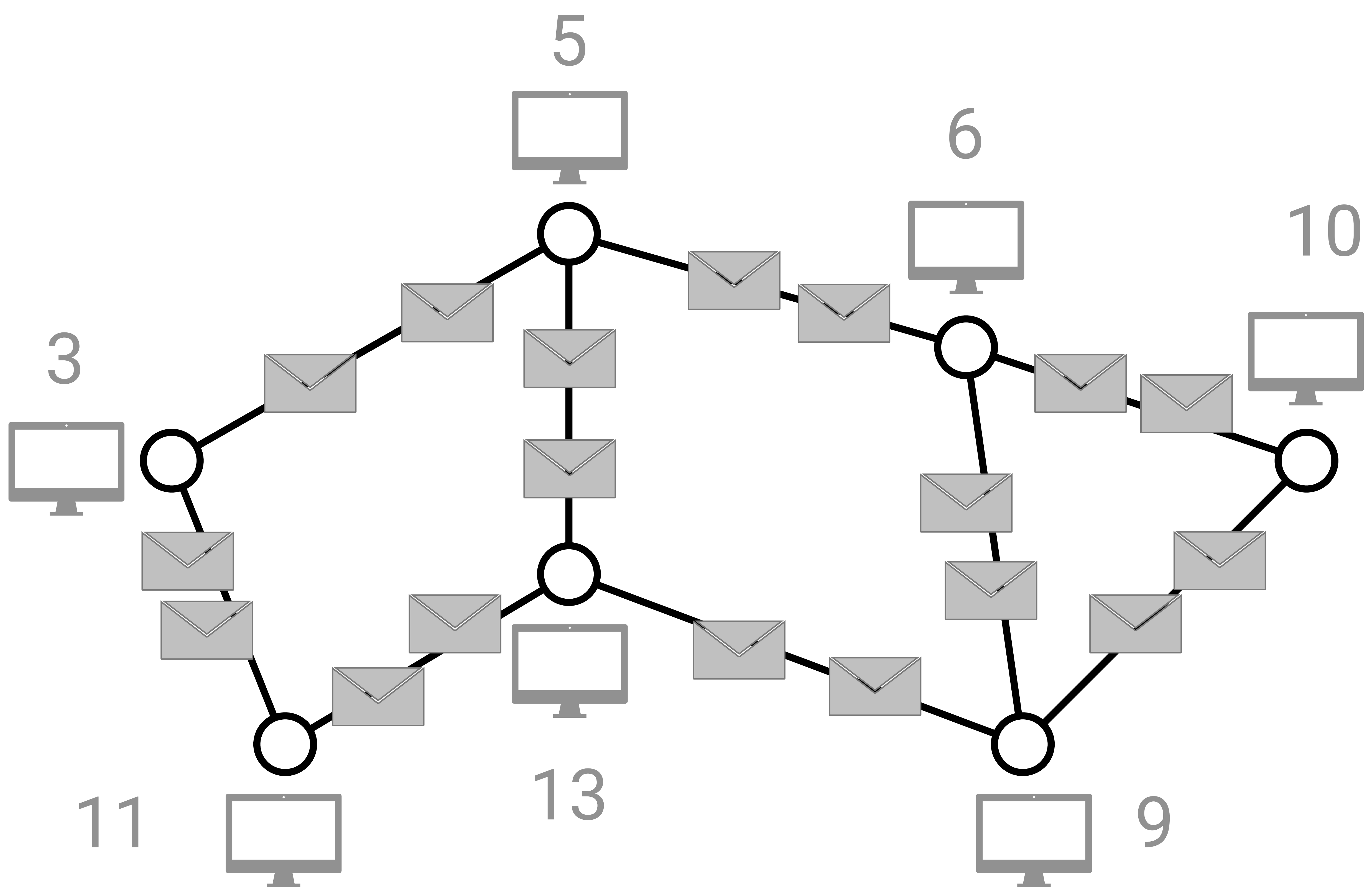
Distributed graph coloring

- Communication: **message passing**



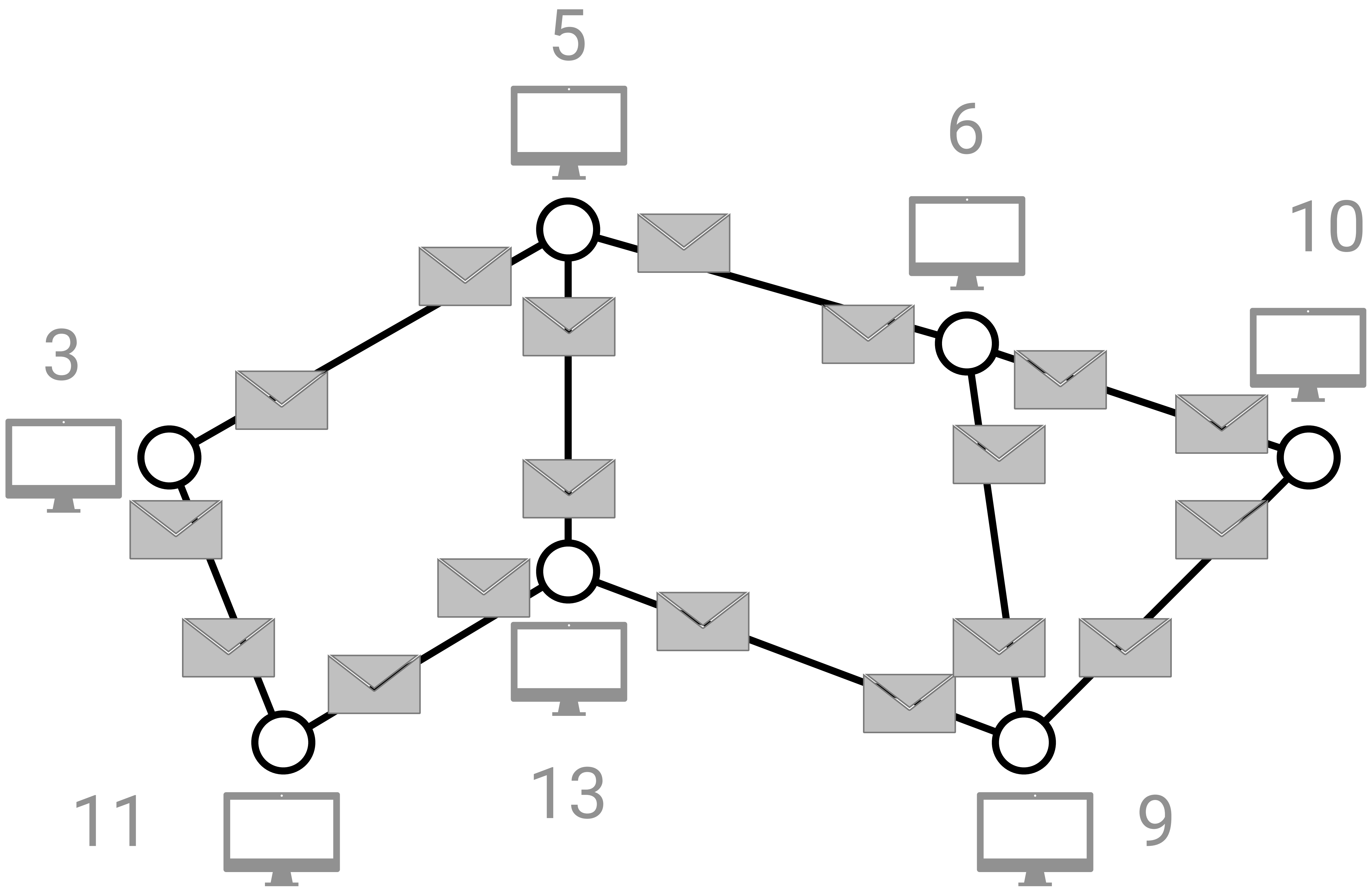
Distributed graph coloring

- Communication: **message passing**



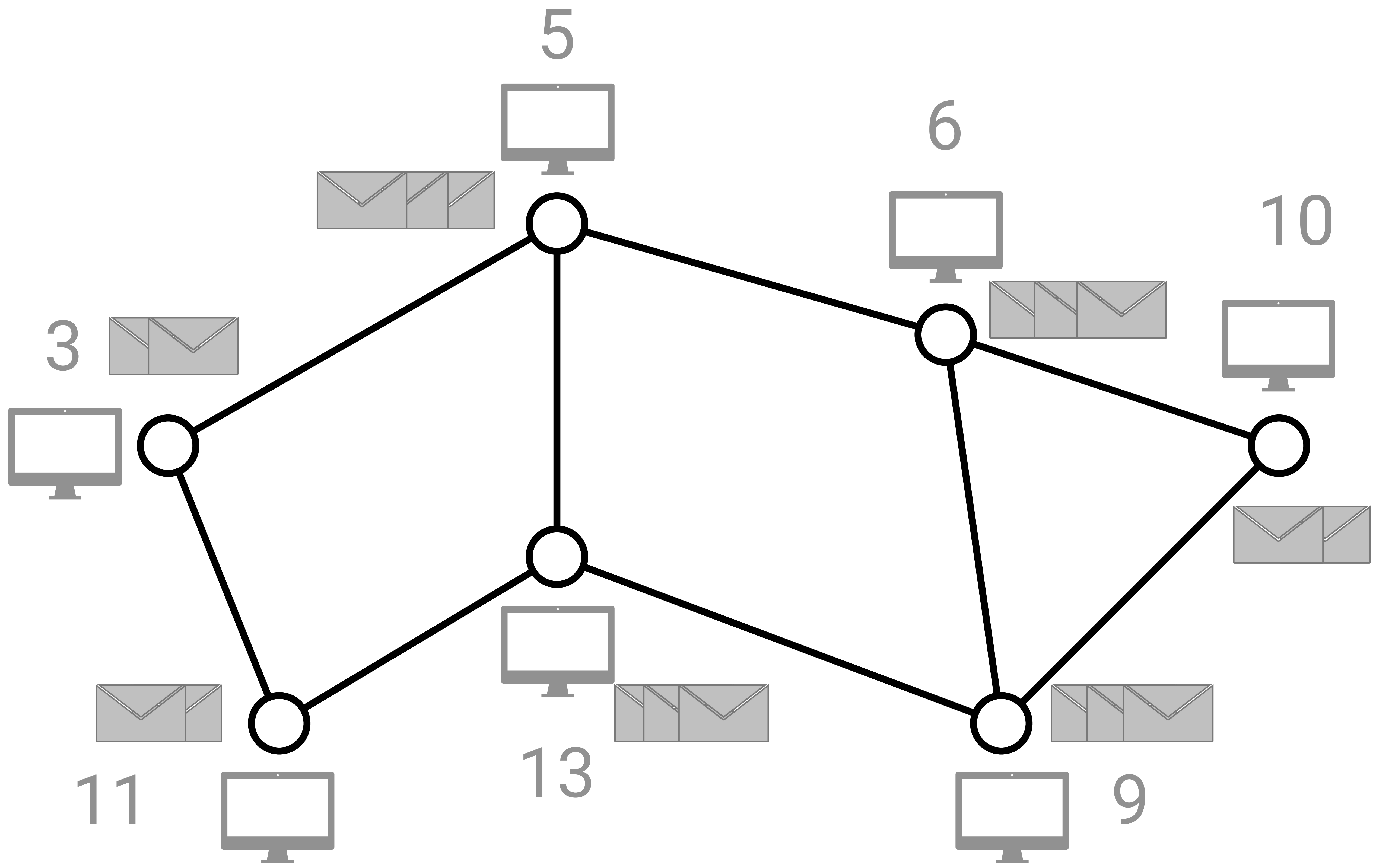
Distributed graph coloring

- Communication: **message passing**



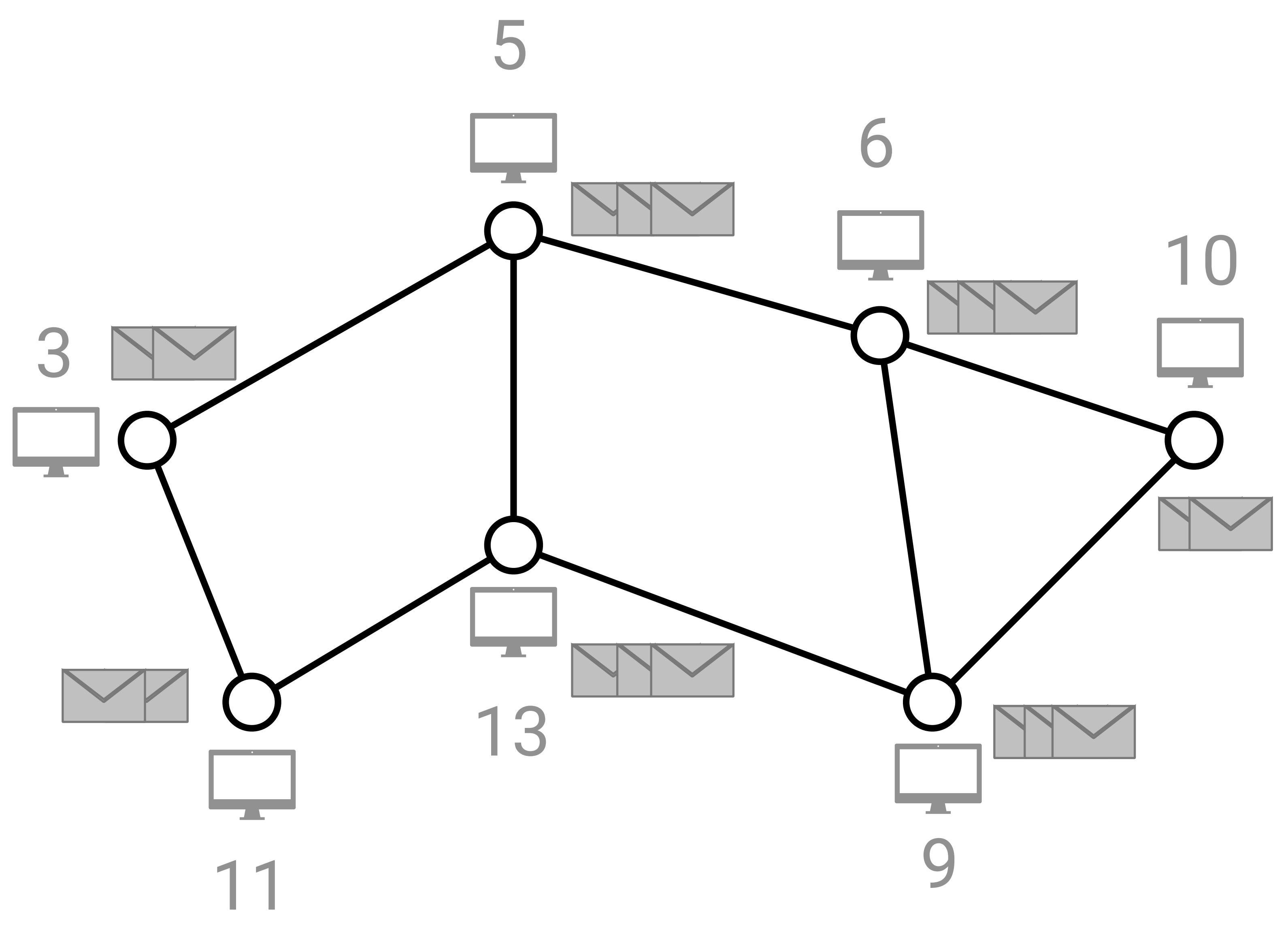
Distributed graph coloring

- Communication: **message passing**



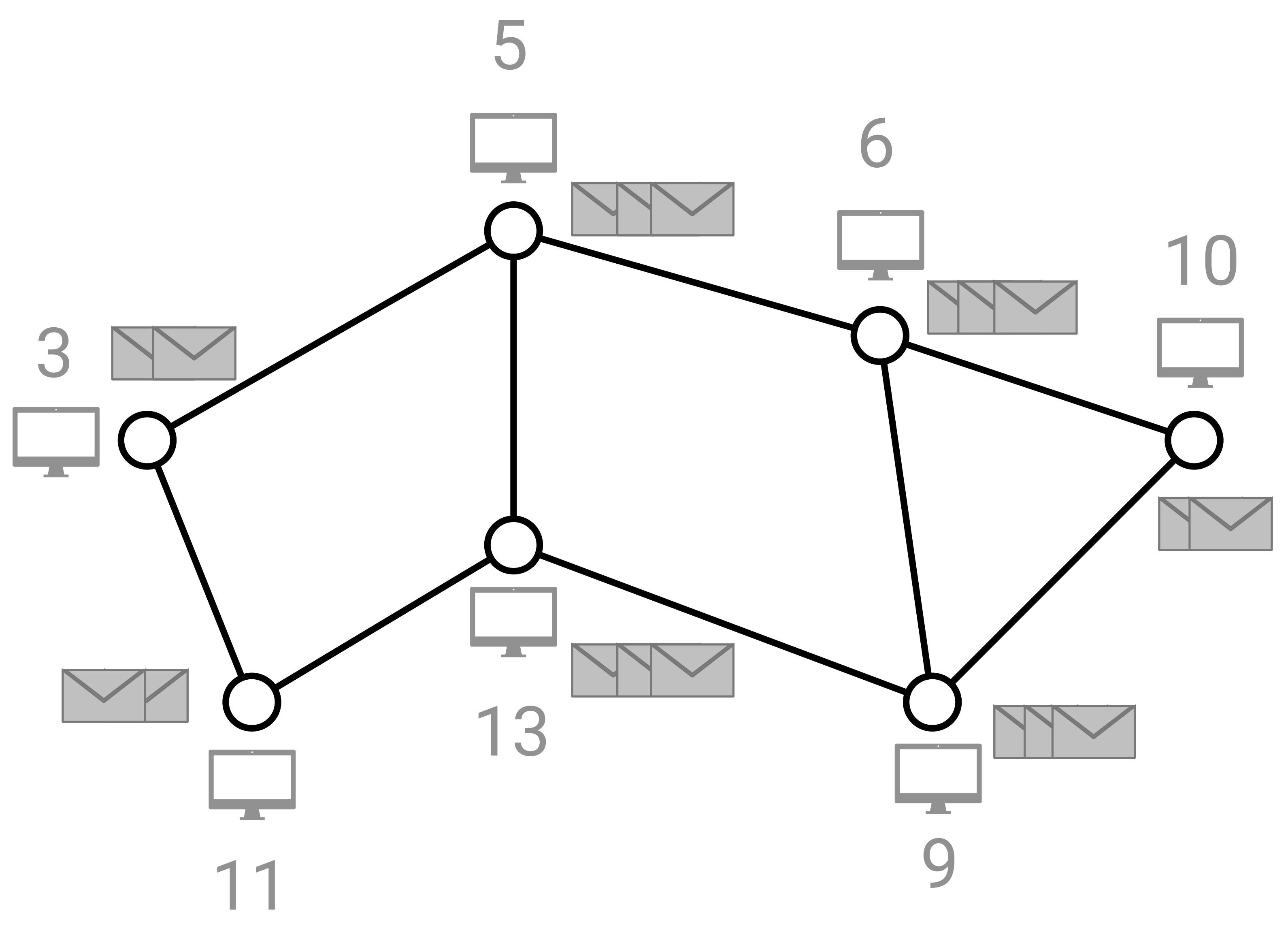
Distributed graph coloring

- **Synchronous rounds**:
  - Each node does some internal computation
  - Sends messages to neighbors
  - Receives messages from neighbors



Distributed graph coloring

- **Synchronous rounds**:
  - Each node does some internal computation
  - Sends messages to neighbors
  - Receives messages from neighbors



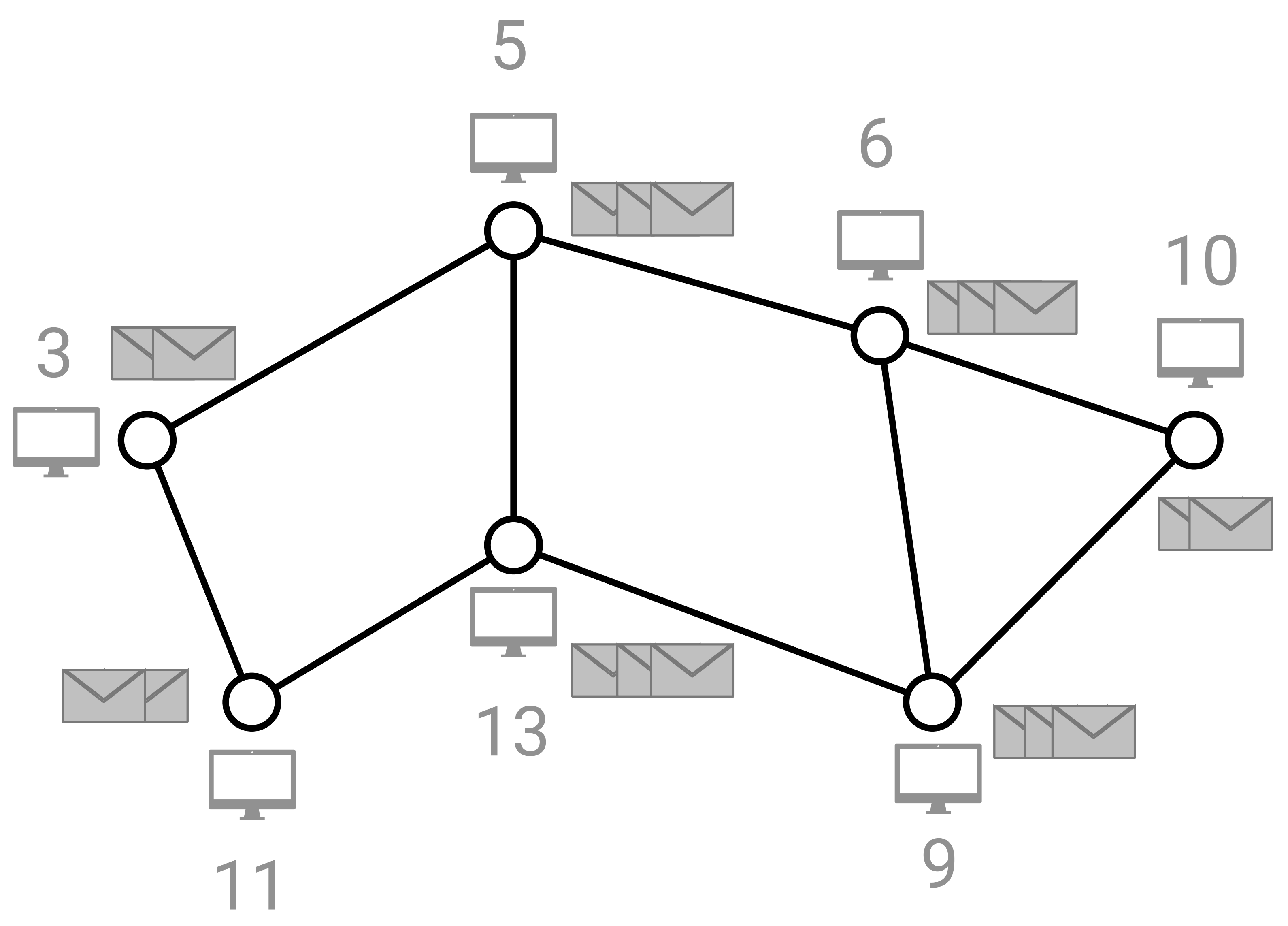
Time complexity = **number of rounds**

LOCAL model

- **Unbounded** internal computation
- **Unbounded** size of messages

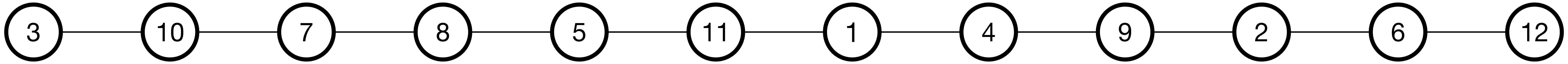
Notation:

- **n**, number of nodes
- **Δ**, maximum degree in the graph
- **deg(*v*)**, degree of node *v*



Distributed graph algorithms

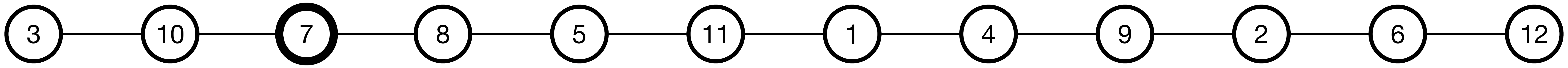
- **Objective**: solve some graph problem (e.g., MIS, vertex coloring)



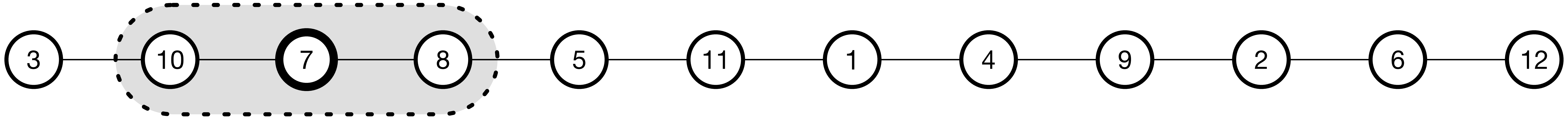
- **At the start**: each node knows *only* its own ID

- **At the end**: each node must know its part of the output
  - Coloring: its **color**
  - MIS: whether it is **in** or **out** the MIS

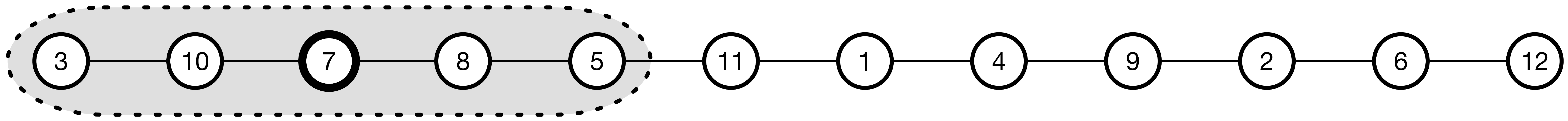
Distributed graph algorithms



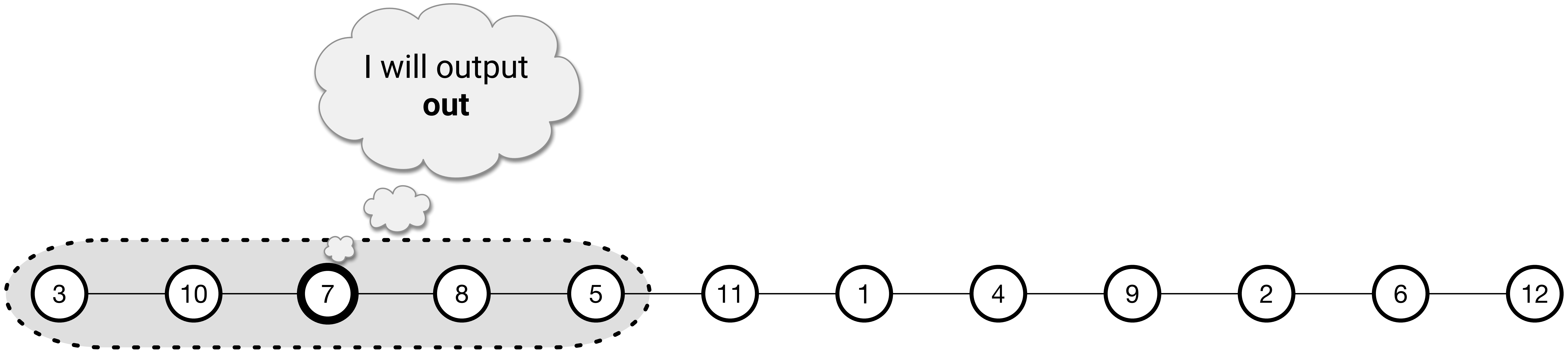
Distributed graph algorithms



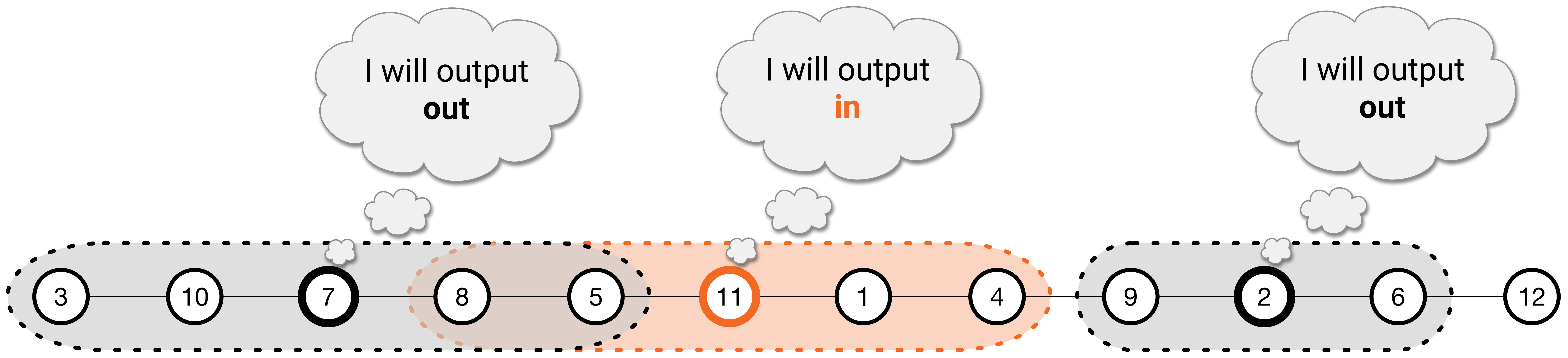
Distributed graph algorithms



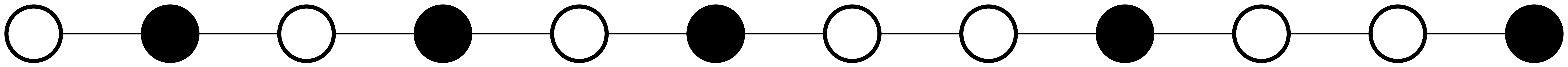
Distributed graph algorithms



Distributed graph algorithms



Distributed graph algorithms



Local outputs form a consistent **global** solution

Application of coloring and MIS

- **Wireless Networks**:
  - Assign **communication channels** while **avoiding collisions** (coloring)
  - Basic **clustering** in wireless networks (MIS)

- **Generally**:
  - Important **symmetry breaking** problems
  - Used as **subroutine** in many algorithms
  - **Techniques** for solving these problems may apply for solving other problems of interest

Sequential greedy coloring

Sequential greedy coloring

MIS:

```
S := ∅
for all v ∈ V do    // go through nodes in an arbitrary order
   if v has no neighbor in S, add v to S
```

- S is an **independent set**, and each node $u \notin S$ has a neighbor in S (S is **maximal**)

Sequential greedy coloring

MIS:

```
S := ∅
for all v ∈ V do    // go through nodes in an arbitrary order
   if v has no neighbor in S, add v to S
```

- S is an **independent set**, and each node $u \notin S$ has a neighbor in S (S is **maximal**)

Coloring (use colors 1, 2, 3, …)

```
all nodes uncoloured
for all v ∈ V do    // go through nodes in an arbitrary order
   assign to v the smallest color not used by its neighbors
```

- Computes a valid (a.k.a. proper) coloring

Sequential greedy coloring

MIS:

```
S := ∅
for all v ∈ V do    // go through nodes in an arbitrary order
   if v has no neighbor in S, add v to S
```

- S is an **independent set**, and each node $u \notin S$ has a neighbor in S (S is **maximal**)

Coloring (use colors 1, 2, 3, …)

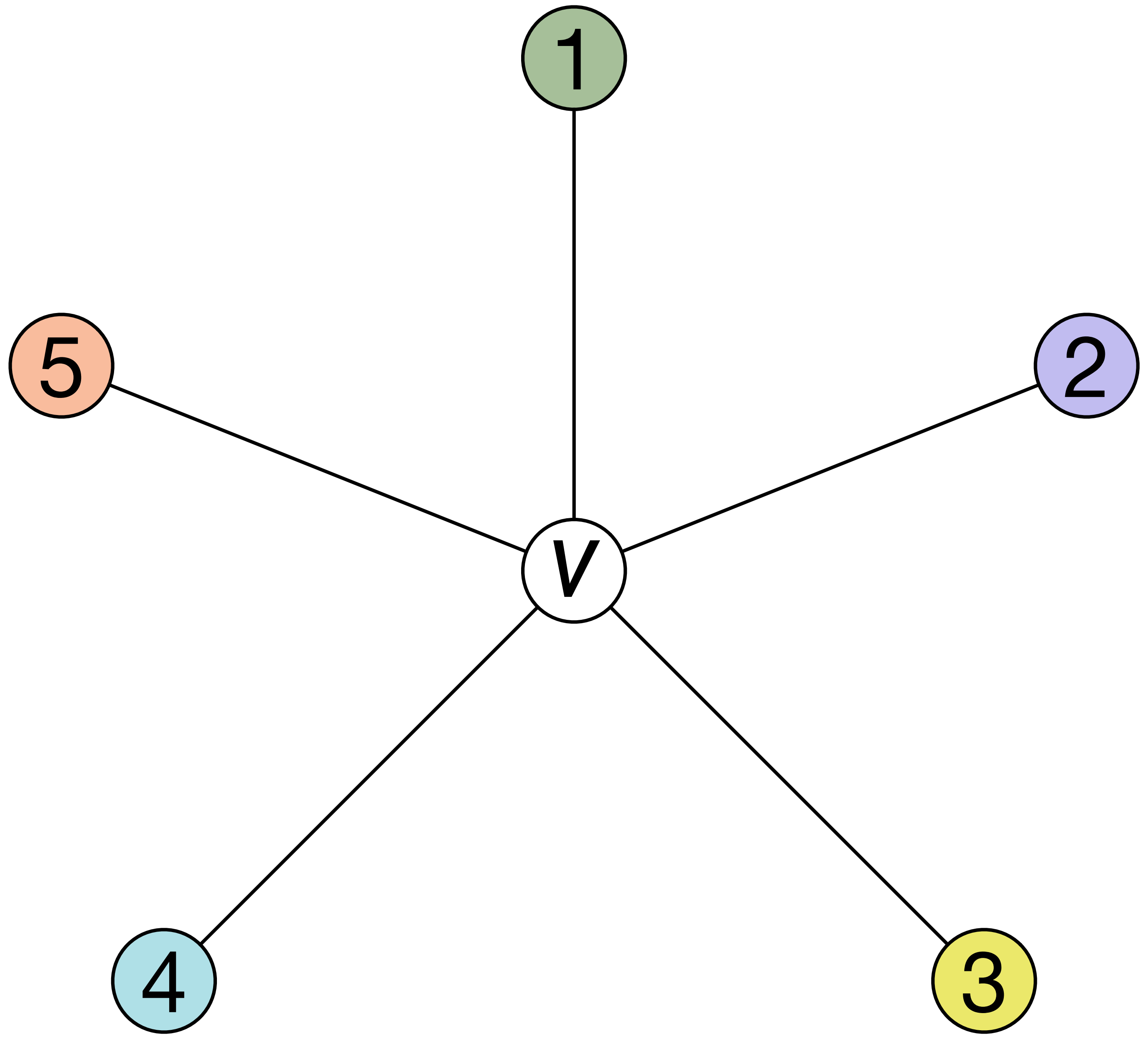
```
all nodes uncoloured
for all v ∈ V do    // go through nodes in an arbitrary order
   assign to v the smallest color not used by its neighbors
```

- Computes a valid (a.k.a. proper) coloring
- **What is the number of colors?**

Greedy vertex coloring: how many colors?

- node *v* **cannot get color 1**: there must exist a neighbor of v with color 1
- node *v* **cannot get color 2**: there must exist a neighbor of v with color 2
- node *v* **cannot get color 3**: there must exist a neighbor of v with color 3

⋮

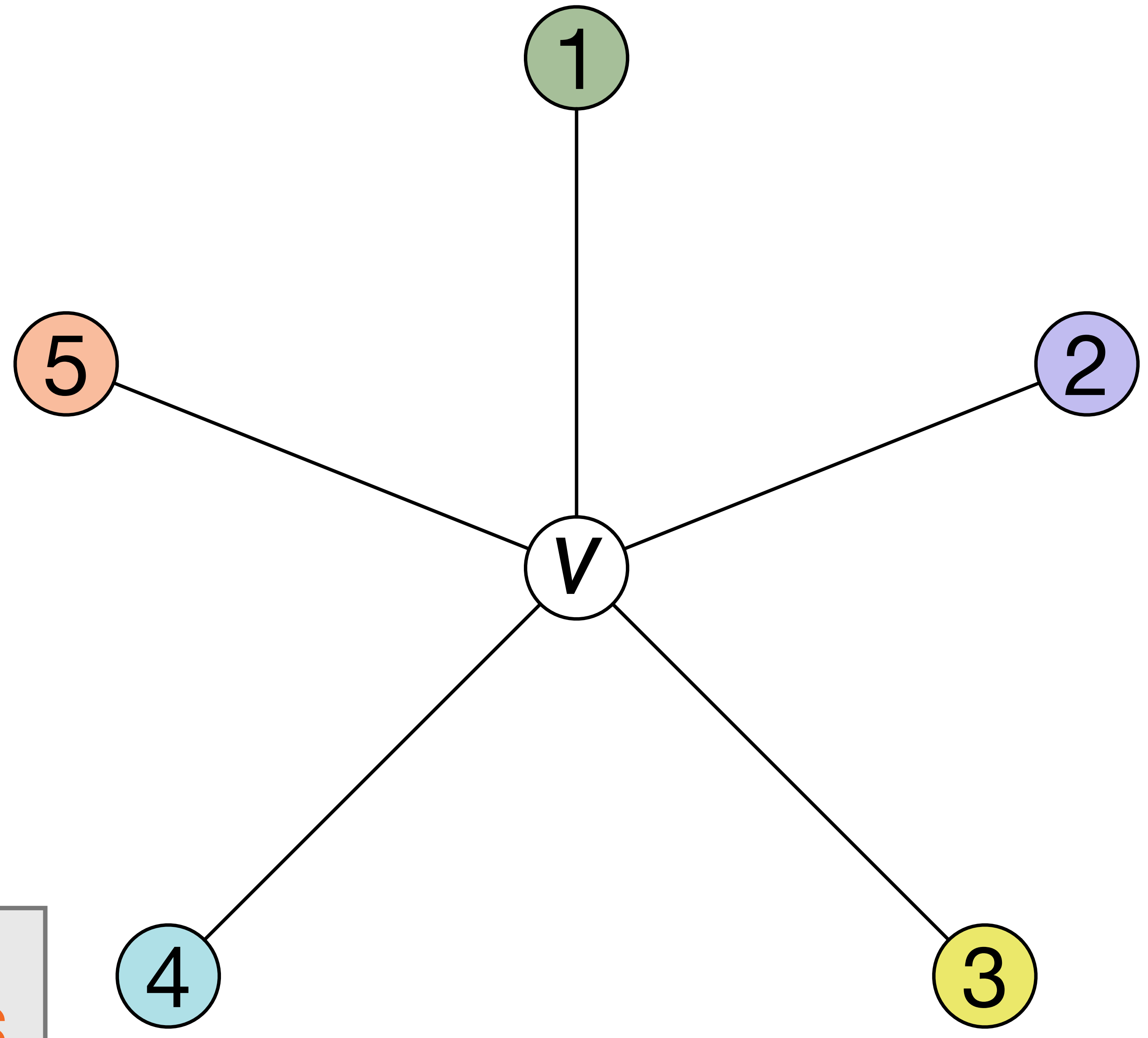


Greedy vertex coloring: how many colors?

- node *v* **cannot get color 1**: there must exist a neighbor of v with color 1
- node *v* **cannot get color 2**: there must exist a neighbor of v with color 2
- node *v* **cannot get color 3**: there must exist a neighbor of v with color 3

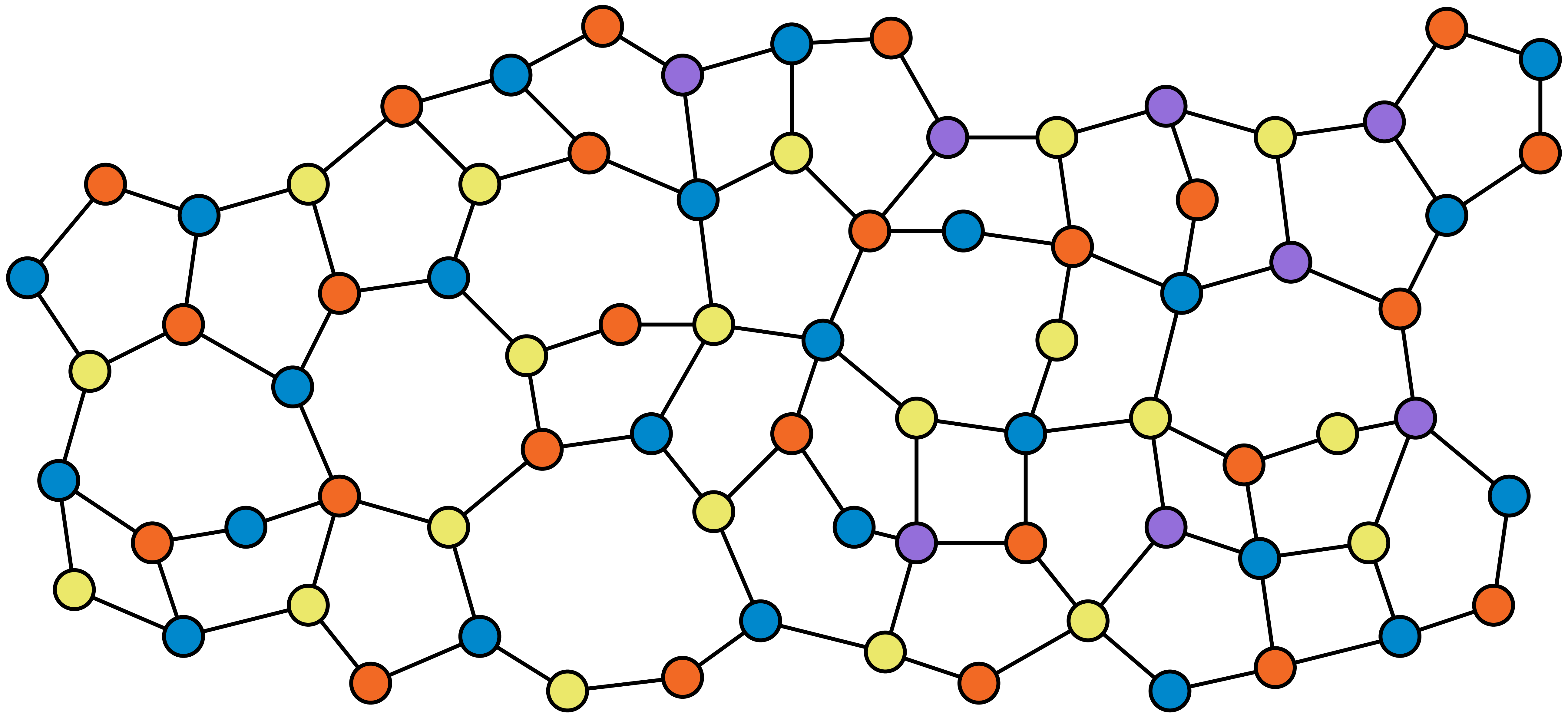
⋮

- Each node *v* gets one of the **first** **deg(*v*) + 1** colors
- Hence one of the first deg(*v*) + 1 colors is free for *v*
- For each node *v*, **color(*v*)** ≤ deg(*v*) + 1 **≤** **Δ + 1**



Theorem: greedy vertex coloring requires **at most Δ + 1 colors**

Distributed vertex coloring



Usually, the **target number of colors** is **$\Delta + 1$**

- Sometimes we want less colors, and we will see some of such examples

Distributed coloring algorithm

How can we color in a distributed way?

- Each node picks the **smallest available color**
  - Available = color not picked by any neighbor
  - How to **avoid conflicts** between neighbors?
  - **Neighbors** should **not** choose a color **at the same time**!

Distributed greedy vertex coloring

Distributed greedy **coloring** for a node *v*

1. **wait** until all **neighbors** of *v* with a **smaller ID** have a color
2. *v* chooses the **smallest available color**
3. *v* **informs** its neighbors

- No two neighbors choose a color at the same time: **proper coloring** with **at most $\Delta + 1$ colors**
- Computes the same coloring as the sequential greedy algorithm when going through the nodes in order defined by IDs

Distributed greedy vertex coloring

Distributed greedy coloring for a node v

1. **wait** until all **neighbors** of v with a **smaller ID** have a color
2. v chooses the **smallest available color**
3. v **informs** its neighbors

- No two neighbors choose a color at the same time: **proper coloring** with **at most $\Delta + 1$ colors**
- Computes the same coloring as the sequential greedy algorithm when going through the nodes in order defined by IDs

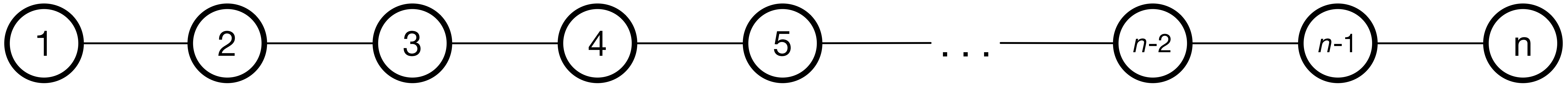
Distributed greedy MIS for a node v

1. **wait** until all **neighbors** of v with a **smaller ID** are decided
2. v **joins** MIS **if no neighbor** of v is already **in** the **MIS**
3. v **informs** its neighbors

Distributed greedy: time complexity

Theorem: The **distributed greedy algorithms** for $(\Delta + 1)$-vertex coloring and MIS terminate after at most ***O*(*n*) rounds**

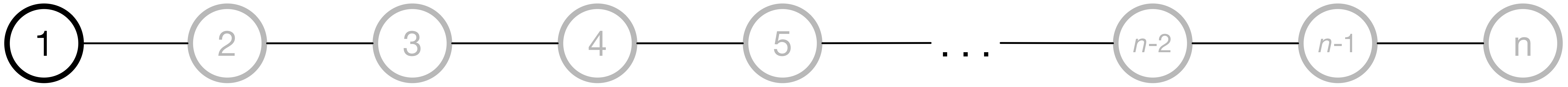
- In each round, at least one **new node is processed**
  - the node with smallest ID among the unprocessed nodes
- ***O*(*n*) rounds** is very slow but unfortunately it **is tight**



Distributed greedy: time complexity

Theorem: The **distributed greedy algorithms** for $(\Delta + 1)$-vertex coloring and MIS terminate after at most ***O*(*n*) rounds**

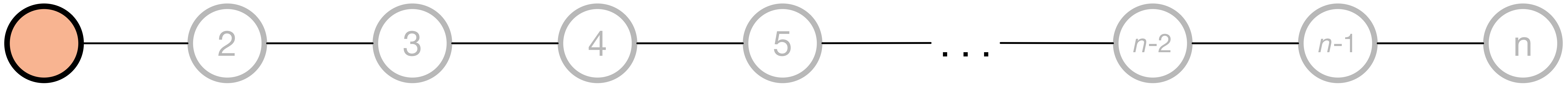
- In each round, at least one **new node is processed**
  - the node with smallest ID among the unprocessed nodes
- ***O*(*n*) rounds** is very slow but unfortunately it **is tight**



Distributed greedy: time complexity

Theorem: The **distributed greedy algorithms** for $(\Delta + 1)$-vertex coloring and MIS terminate after at most ***O(n)* rounds**

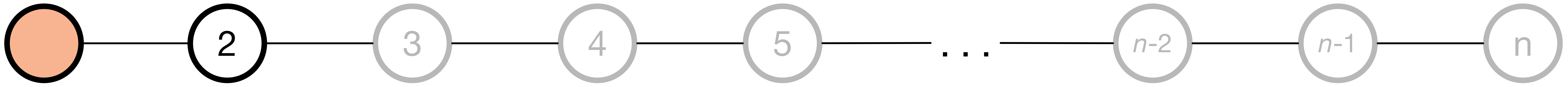
- In each round, at least one **new node is processed**
  - the node with smallest ID among the unprocessed nodes
- ***O(n)* rounds** is very slow but unfortunately it **is tight**



Distributed greedy: time complexity

Theorem: The **distributed greedy algorithms** for $(\Delta + 1)$-vertex coloring and MIS terminate after at most ***O*(*n*) rounds**

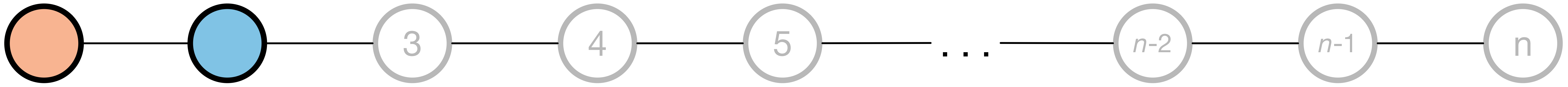
- In each round, at least one **new node is processed**
  - the node with smallest ID among the unprocessed nodes
- ***O*(*n*) rounds** is very slow but unfortunately it **is tight**



Distributed greedy: time complexity

Theorem: The **distributed greedy algorithms** for $(\Delta + 1)$-vertex coloring and MIS terminate after at most ***O*(*n*) rounds**

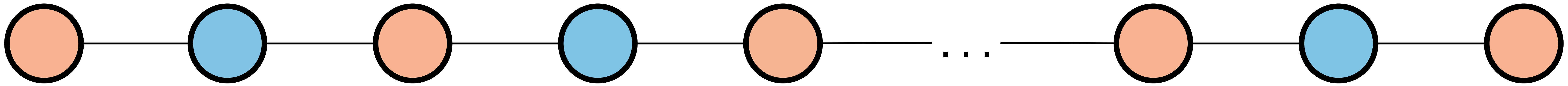
- In each round, at least one **new node is processed**
  - the node with smallest ID among the unprocessed nodes
- ***O*(*n*) rounds** is very slow but unfortunately it **is tight**



Distributed greedy: time complexity

Theorem: The **distributed greedy algorithms** for $(\Delta + 1)$-vertex coloring and MIS terminate after at most **$O(n)$ rounds**

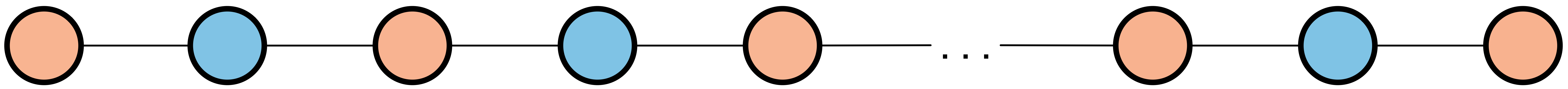
- In each round, at least one **new node is processed**
  - the node with smallest ID among the unprocessed nodes
- **$O(n)$ rounds** is very slow but unfortunately it **is tight**



Distributed greedy: time complexity

Theorem: The **distributed greedy algorithms** for $(\Delta + 1)$-vertex coloring and MIS terminate after at most **$O(n)$ rounds**

- In each round, at least one **new node is processed**
  - the node with smallest ID among the unprocessed nodes
- **$O(n)$ rounds** is very slow but unfortunately it **is tight**



- Can we be **faster**?
  - How to process many nodes in parallel while avoiding conflicts?
- **Observation**: we can be faster if we are already given a proper coloring with C colors

From C-coloring to $(\Delta + 1)$-coloring and MIS

Assumption: we are **given a proper *C*-coloring** of the nodes (with colors 1, 2, … , *C*)

- In both algorithms, we can replace IDs with these colors

The algorithm runs in phases 1, 2, … , *C*

In phase *i*:

- Nodes with initial **color *i*** are processed
  - Coloring: pick smallest available color
  - MIS: join MIS if no neighbor is in MIS
- At the end of the phase, newly processed nodes **inform neighbors**

- The algorithm works because **only non-adjacent nodes** are processed in **parallel**
- Time complexity: ***C* rounds**

From C-coloring to $(\Delta + 1)$-coloring and MIS

Assumption: we are **given a proper *C*-coloring** of the nodes (with colors 1, 2, … , *C*)

- In both algorithms, we can replace IDs with these colors

The algorithm runs in phases 1, 2, … , *C*

In phase *i*:

- Nodes with initial **color *i*** are processed
  - Coloring: pick smallest available color
  - MIS: join MIS if no neighbor is in MIS
- At the end of the phase, newly processed nodes **inform neighbors**

- The algorithm works because **only non-adjacent nodes** are processed in **parallel**
- Time complexity: ***C* rounds**

Can we do better?

From C-coloring to (Δ + 1)-coloring and MIS

Assumption: we are **given a ~~proper~~ C-coloring** of the nodes (with colors 1, 2, … , C)

- In both algorithms, we can replace IDs with these colors

The algorithm runs in phases 1, 2, … , C
In phase *i*:
• Nodes with initial **color *i*** are processed
    ‣ Coloring: pick smallest available color
    ‣ MIS: join MIS if no neighbor is in MIS
• At the end of the phase, newly processed nodes **inform neighbors**

- The algorithm works because **only non-adjacent nodes** are processed in **parallel**
- Time complexity: ***C* rounds**

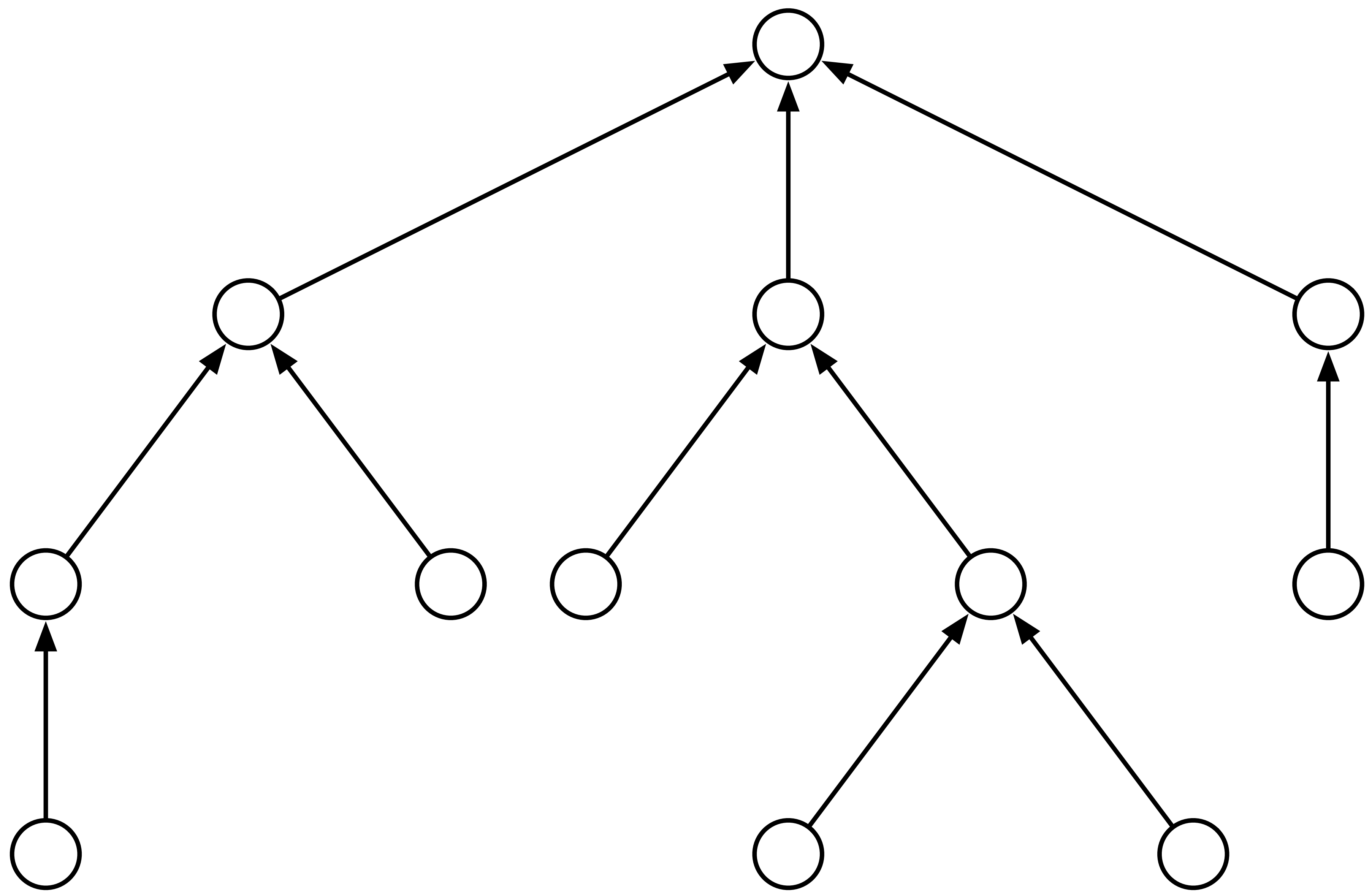
Can we do better?

Coloring special graph classes

Let's first take a look at special classes of graphs

Rooted trees:

- Graph is a tree, each **node knows** which neighbor is **its parent**
- The **root knows it is the root**



Coloring special graph classes

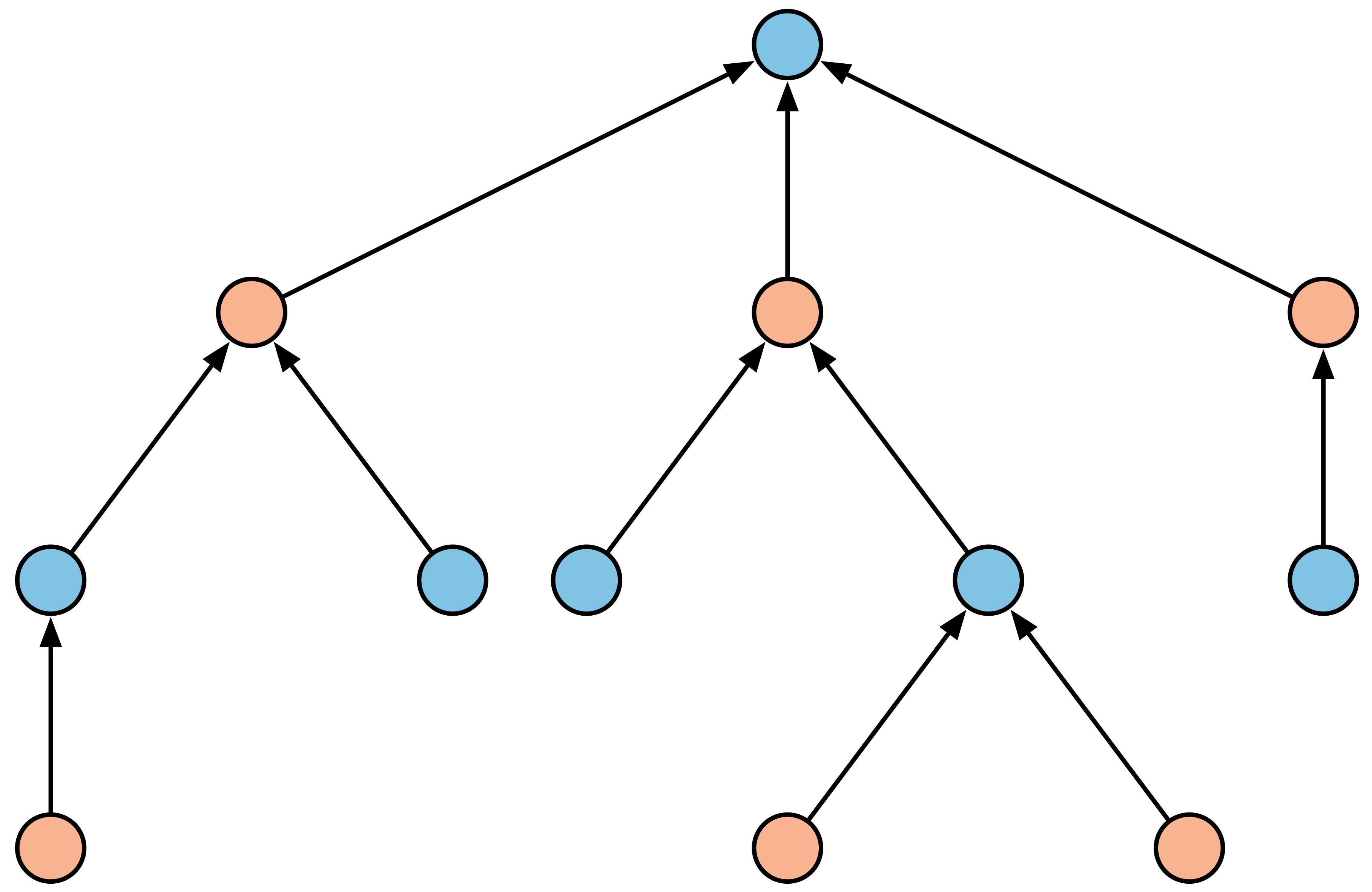
Trees can be colored with 2 colors:

- Color 0: even distance to root
- Color 1: odd distance to root

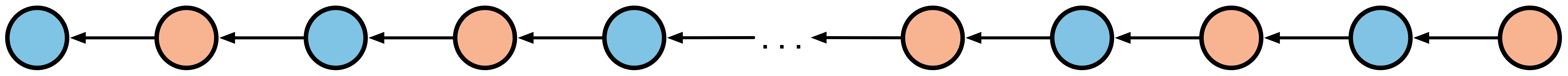
Distributed algorithm:

- Color level by level, starting at the root

Time complexity: $O(D)$



This is tight and can be $\Theta(n)$:



Nodes need to know the parity of their distance to the root (formal argument in a later lecture)

Coloring rooted trees with more colors

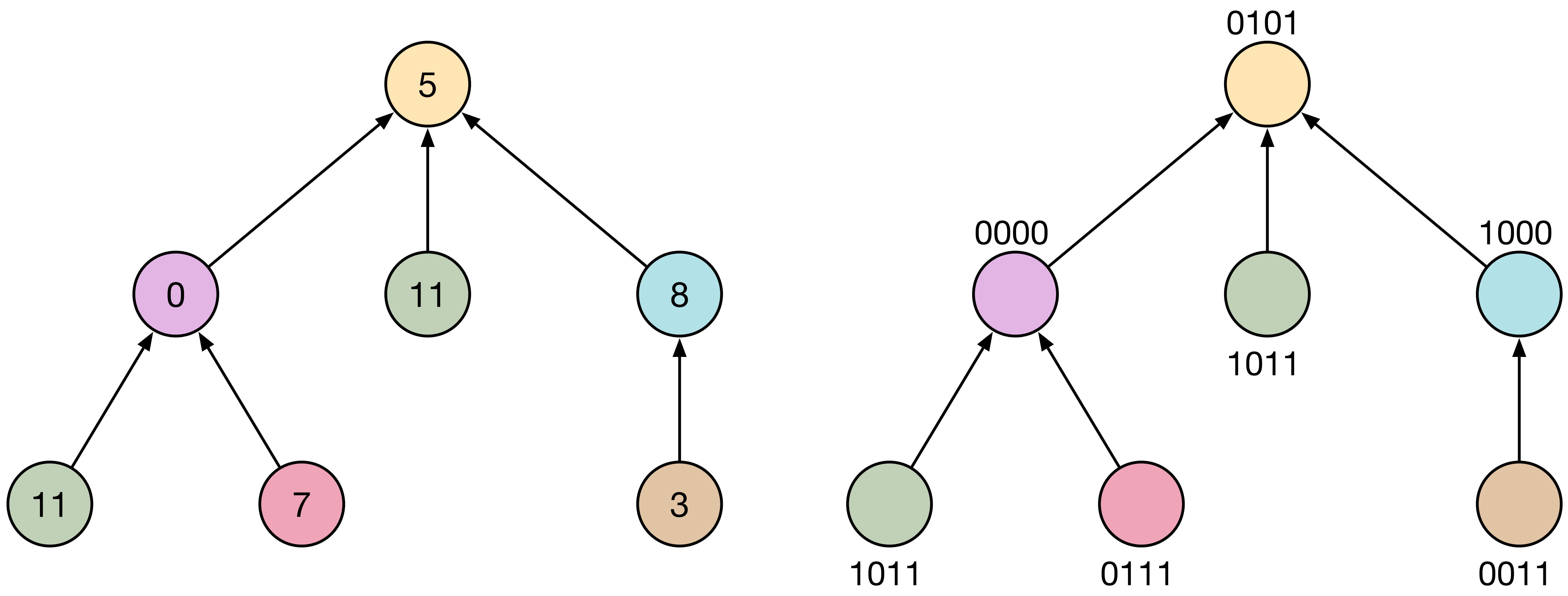
Color reduction:

- **Assume** we are given **a proper coloring** with ***C* colors**
  - Initially, if we have unique IDs from an ID space of size N, we have $C = N$
- Can we **reduce** the number of **colors**?
  - What happens if we reduce them iteratively?

Coloring rooted trees with more colors

Specific assumption:

- Initital coloring with colors in {0, … , C - 1} for some $C \in \mathbb{N}$ (each node knows C)
- Interpret color as **bit string of length $\lceil \log_2 C \rceil$**
- Example for C = 12



Cole-Vishkin color reduction scheme

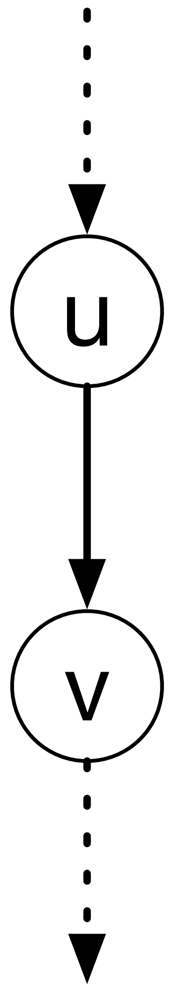
- Consider node u and its parent v with colors c_u and c_v ($c_u \neq c_v$)
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - $i_u \coloneqq$ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

New color of *u*:
$c'_u = 2 \cdot i_u + b_u$

Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - i_u ≔ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

c_u = 60346
c_v = 13242

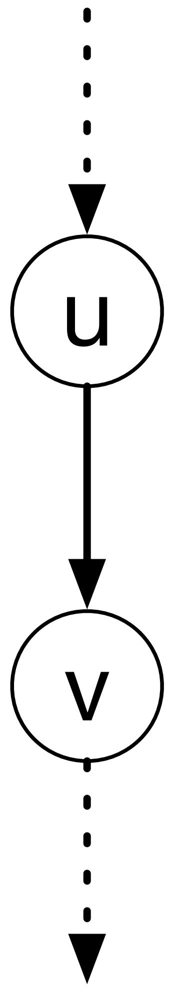


New color of *u*:
$c'_u = 2 \cdot i_u + b_u$

Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - i_u ≔ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

$c_u = 60346$
$c_v = 13242$



x_u: 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 0

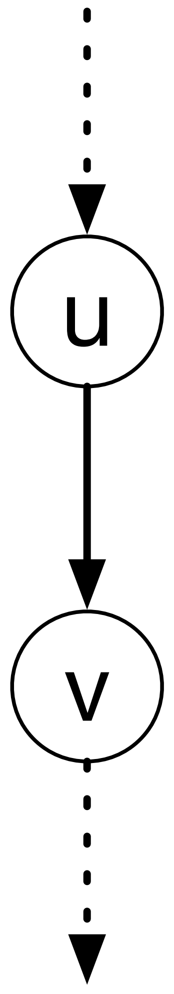
x_v: 0 0 1 1 0 0 1 1 1 0 1 1 1 0 1 0

New color of *u*:
$c'_u = 2 \cdot i_u + b_u$

Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - i_u ≔ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

$c_u = 60346$
$c_v = 13242$
$i_u =$



x_u: 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 0

x_v: 0 0 1 1 0 0 1 1 1 0 1 1 1 0 1 0

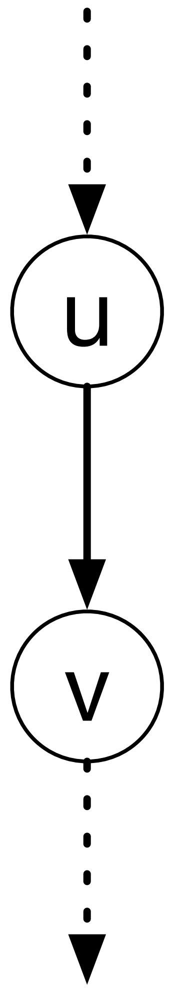
New color of *u*:

$c'_u = 2 \cdot i_u + b_u$

Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - i_u ≔ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

$c_u = 60346$
$c_v = 13242$
$i_u =$



x_u: 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 0

↕

x_v: 0 0 1 1 0 0 1 1 1 0 1 1 1 0 1 0

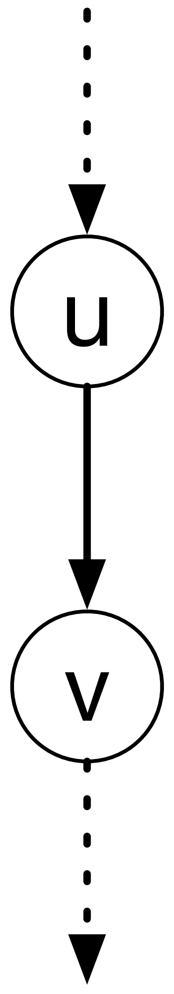
New color of *u*:

$c'_u = 2 \cdot i_u + b_u$

Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - i_u ≔ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

$c_u = 60346$
$c_v = 13242$
$i_u =$



x_u: 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 0

↕ ↕

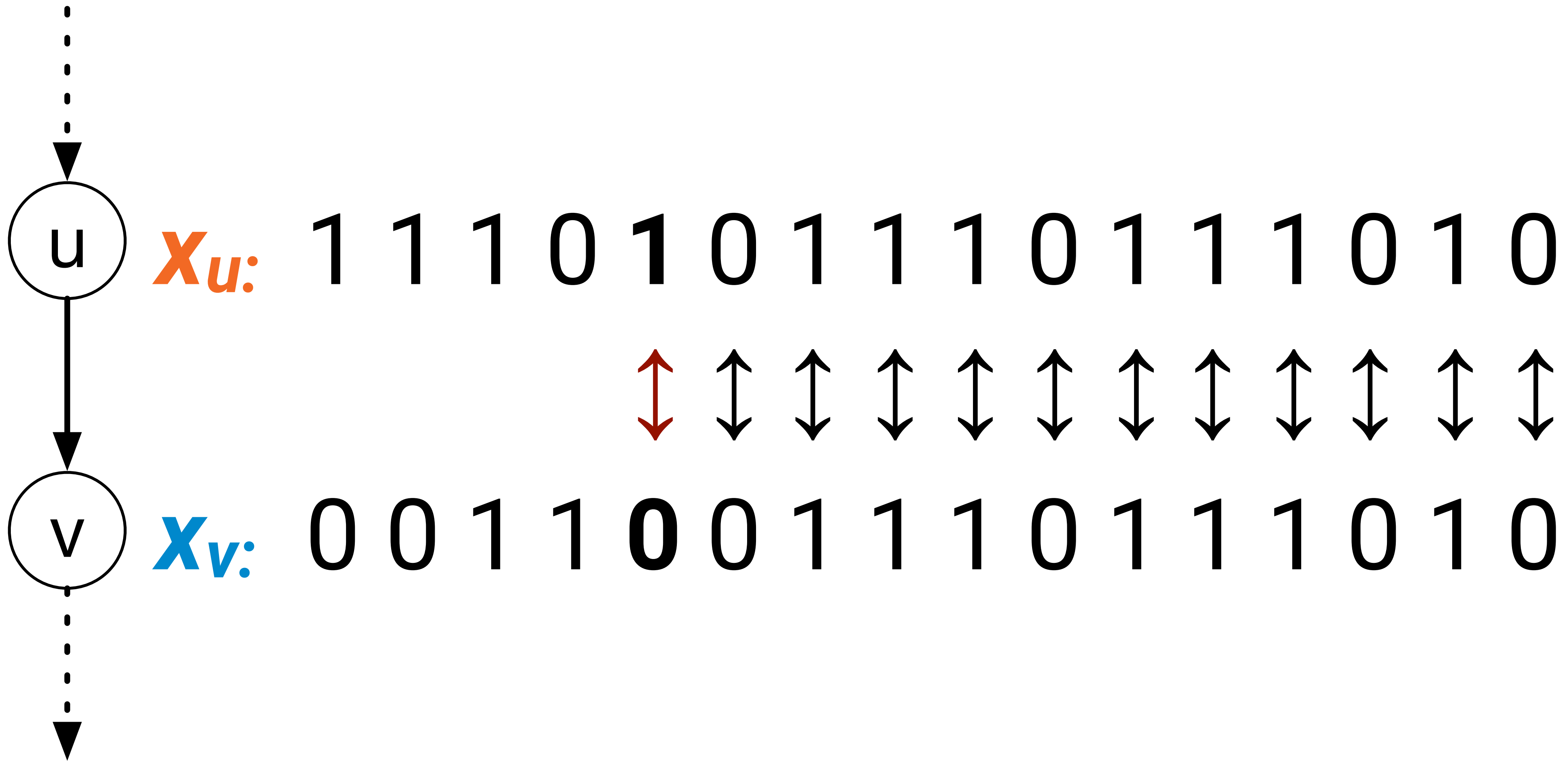
x_v: 0 0 1 1 0 0 1 1 1 0 1 1 1 0 1 0

New color of u:
$c'_u = 2 \cdot i_u + b_u$

Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - i_u ≔ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

$c_u = 60346$
$c_v = 13242$
$i_u = 11$

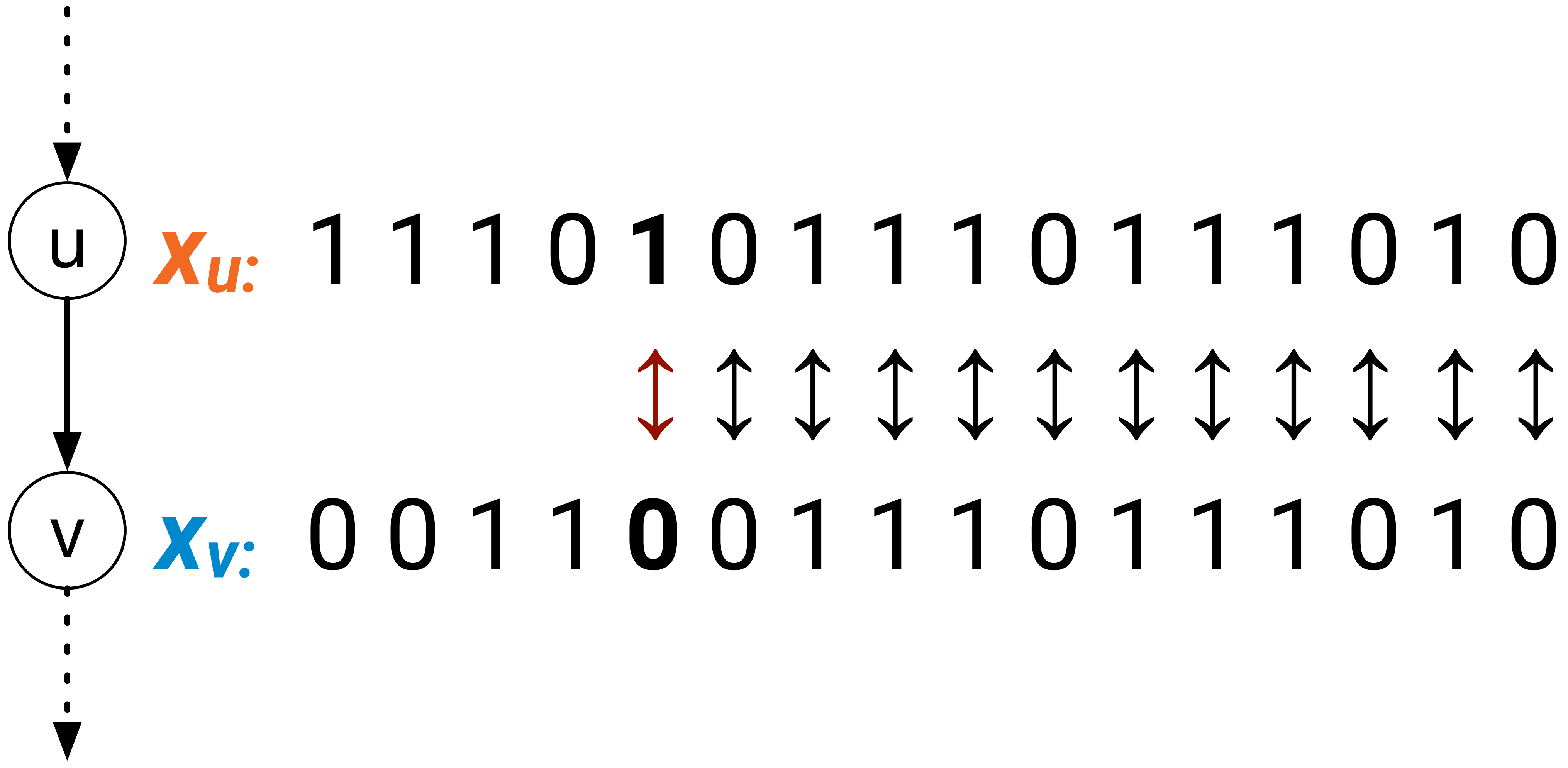


New color of u:

$c'_u = 2 \cdot i_u + b_u$

Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - i_u ≔ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u



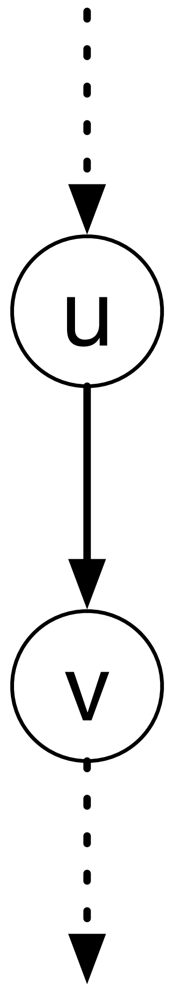
$c_u = 60346$
$c_v = 13242$
$i_u = 11$
$b_u = 1$

New color of u:

$c'_u = 2 \cdot i_u + b_u$

Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - i_u ≔ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u



x_u: 1 1 1 0 **1** 0 1 1 1 0 1 1 1 0 1 0

x_v: 0 0 1 1 **0** 0 1 1 1 0 1 1 1 0 1 0

$c_u = 60346$
$c_v = 13242$
$i_u = 11$
$b_u = 1$

New color of *u*:

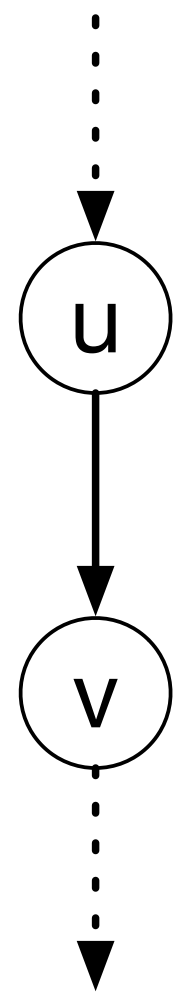
$c'_u = 2 \cdot i_u + b_u$

$c'_u = 2 \cdot 11 + 1 = 23$

Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - $i_u \coloneqq$ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

Theorem: For any two neighbors, if $c_u \neq c_v$ then it holds $c'_u \neq c'_v$

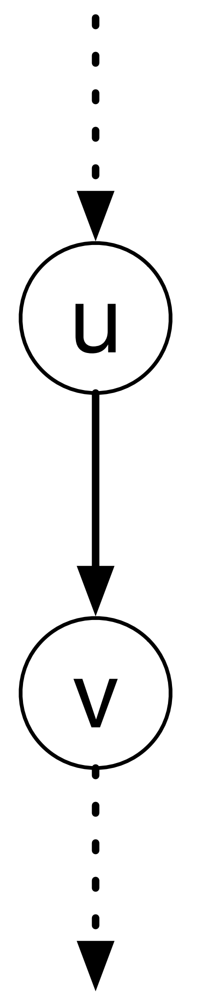


Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - i_u ≔ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

Theorem: For any two neighbors, if $c_u \neq c_v$ then it holds $c'_u \neq c'_v$

Proof:



- we have that $c'_u = 2 \cdot i_u + b_u$ and $c'_v = 2 \cdot i_v + b_v$
- we have that $c'_u \neq c'_v$ if and only if $i_u \neq i_v$ or $b_u \neq b_v$

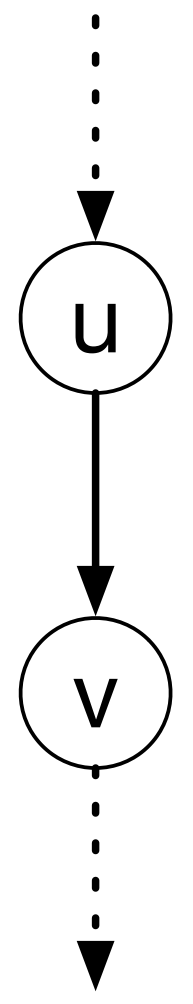
Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - i_u ≔ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

Theorem: For any two neighbors, if $c_u \neq c_v$ then it holds $c'_u \neq c'_v$

Proof:

- we have that $c'_u = 2 \cdot i_u + b_u$ and $c'_v = 2 \cdot i_v + b_v$
- we have that $c'_u \neq c'_v$ if and only if $i_u \neq i_v$ or $b_u \neq b_v$
- w.l.o.g., assume v is the parent of u



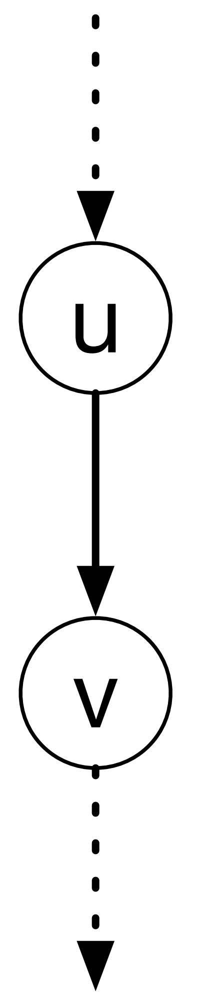
Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - $i_u \coloneqq$ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

Theorem: For any two neighbors, if $c_u \neq c_v$ then it holds $c'_u \neq c'_v$

Proof:

- we have that $c'_u = 2 \cdot i_u + b_u$ and $c'_v = 2 \cdot i_v + b_v$
- we have that $c'_u \neq c'_v$ if and only if $i_u \neq i_v$ or $b_u \neq b_v$
- w.l.o.g., assume v is the parent of u
- if $i_u \neq i_v$ then we are done



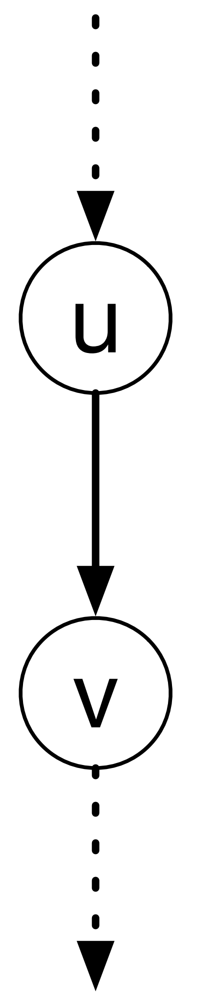
Cole-Vishkin color reduction scheme

- Consider node u and its parent v with colors c_u and c_v
  - x_u: binary representation of c_u
  - x_v: binary representation of c_v
- Define:
  - i_u ≔ {index of the first bit where x_u and x_v differ}
  - $b_u \in \{0, 1\}$ is the bit of x_u in position i_u

Theorem: For any two neighbors, if $c_u \neq c_v$ then it holds $c'_u \neq c'_v$

Proof:

- we have that $c'_u = 2 \cdot i_u + b_u$ and $c'_v = 2 \cdot i_v + b_v$
- we have that $c'_u \neq c'_v$ if and only if $i_u \neq i_v$ or $b_u \neq b_v$
- w.l.o.g., assume v is the parent of u
- if $i_u \neq i_v$ then we are done
- if $i_u = i_v = i$ it means that, in that position, the bits differ, hence $b_u \neq b_v$



Cole-Vishkin color reduction scheme

1. How much do we **reduce the colors in one step**?

2. How much can we reduce the colors if we **iteratively** apply the **color reduction** scheme?

3. What is the **runtime** of this procedure?

Cole-Vishkin color reduction scheme

How much do we reduce the colors in one step?

- Each node u has an initial color c_u
- c_u can be written as a $\lceil \log_2 C \rceil$-bit number

Cole-Vishkin color reduction scheme

How much do we reduce the colors in one step?

- Each node u has an initial color c_u
- c_u can be written as a $\lceil \log_2 C \rceil$-bit number
- Therefore:

$$i_u \in \{0, 1, \ldots, \lceil \log_2 C \rceil - 1\}$$

- And thus:

$$c'_u = 2 \cdot i_u + b_u \leq 2 \cdot i_u + 1 \leq 2\lceil \log_2 C \rceil - 1$$

Cole-Vishkin color reduction scheme

How much do we reduce the colors in one step?

- Each node u has an initial color c_u
- c_u can be written as a $\lceil \log_2 C \rceil$-bit number
- Therefore:

$$i_u \in \{0, 1, \ldots, \lceil \log_2 C \rceil - 1\}$$

- And thus:

$$c'_u = 2 \cdot i_u + b_u \leq 2 \cdot i_u + 1 \leq 2\lceil \log_2 C \rceil - 1$$

Theorem: In one color reduction step, the number of colors is reduced **from C to $2\lceil \log_2 C \rceil$**

Cole-Vishkin color reduction scheme

How much can we reduce the colors if we iteratively apply the color reduction scheme?

- In one color reduction step, the number of colors is reduced **from C to $2\lceil \log_2 C \rceil$**

Theorem: Applying the **color reduction** step **iteratively**, the algorithm eventually computes a coloring with the **six colors** $\{0, 1, \ldots, 5\}$

Proof: $C > 2\lceil \log_2 C \rceil$ for all $C > 6$

Cole-Vishkin color reduction scheme

How much can we reduce the colors if we iteratively apply the color reduction scheme?

- In one color reduction step, the number of colors is reduced **from C to $2\lceil \log_2 C \rceil$**

Theorem: Applying the **color reduction** step **iteratively**, the algorithm eventually computes a coloring with the **six colors** $\{0, 1, \ldots, 5\}$

Proof: $C > 2\lceil \log_2 C \rceil$ for all $C > 6$

What is the runtime of this procedure?

Rooted tree coloring: time complexity

The log-star function:

- For a real number $n > 1$ and an integer $i \geq 1$, we define

$$\log_2^{(i)} n := \log_2(\log_2^{(i-1)} n) \qquad \log_2^{(1)} n := \log_2 n$$

- For an integer $n \geq 2$, the function log* n is defined as

$$\log^* n := \min\{i : \log_2^i n \leq 1\}$$

- **log* *n***: number of times one has to apply the function $\log_2 n$ in order to obtain a number that is ≤ 1
- Examples:

$$\log^* 2 = 1, \log^* 4 = 2, \log^* 16 = 3, \log^* 2^{16} = 4, \log^* 2^{2^{16}} = 5$$

Rooted tree coloring: time complexity

The log-star function:

- For a real number $n > 1$ and an integer $i \geq 1$, we define

$$\log_2^{(i)} n := \log_2(\log_2^{(i-1)} n) \qquad \log_2^{(1)} n := \log_2 n$$

- For an integer $n \geq 2$, the function log* n is defined as

$$\log^* n := \min\{i : \log_2^i n \leq 1\}$$

Theorem: When starting with colors in {0, … , n - 1} the Cole-Vishkin color reduction algorithm computes a **6-coloring** of a **rooted tree** in ***O*(log* *n*)** rounds

Rooted tree coloring: time complexity

The log-star function:

- For a real number $n > 1$ and an integer $i \geq 1$, we define

$$\log_2^{(i)} n := \log_2(\log_2^{(i-1)} n) \qquad \log_2^{(1)} n := \log_2 n$$

- For an integer $n \geq 2$, the function log* n is defined as

$$\log^* n := \min\{i : \log_2^i n \leq 1\}$$

Theorem: When starting with colors in {0, … , n - 1} the Cole-Vishkin color reduction algorithm computes a **6-coloring** of a **rooted tree** in ***O*(log* *n*)** rounds

Proof sketch: Colors are reduced as follows

$$n \rightarrow 2\lceil \log_2 n \rceil$$

Rooted tree coloring: time complexity

The log-star function:

- For a real number *n* > 1 and an integer *i* ≥ 1, we define

$$\log_2^{(i)} n := \log_2(\log_2^{(i-1)} n) \qquad \log_2^{(1)} n := \log_2 n$$

- For an integer *n* ≥ 2, the function log* n is defined as

$$\log^* n := \min\{i : \log_2^i n \leq 1\}$$

Theorem: When starting with colors in {0, … , *n* - 1} the Cole-Vishkin color reduction algorithm computes a **6-coloring** of a **rooted tree** in ***O*(log* *n*)** rounds

Proof sketch: Colors are reduced as follows

$$n \to O(\log n)$$

Rooted tree coloring: time complexity

The log-star function:

- For a real number $n > 1$ and an integer $i \geq 1$, we define

$$\log_2^{(i)} n := \log_2(\log_2^{(i-1)} n) \qquad \log_2^{(1)} n := \log_2 n$$

- For an integer $n \geq 2$, the function log* n is defined as

$$\log^* n := \min\{i : \log_2^i n \leq 1\}$$

Theorem: When starting with colors in {0, … , n - 1} the Cole-Vishkin color reduction algorithm computes a **6-coloring** of a **rooted tree** in ***O*(log* *n*)** rounds

Proof sketch: Colors are reduced as follows

$$n \to O(\log n) \to O(\log\log n) \to O(\log\log\log n) \to \ldots$$

From six to three colors

Coloring rooted trees:

- We have seen that computing a **2-coloring** requires $\Omega(D)$
- We have seen how to compute a **6-coloring** in $O(\log^* n)$ rounds
- What about **3, 4, and 5 colors**?

From six to three colors

Coloring rooted trees:

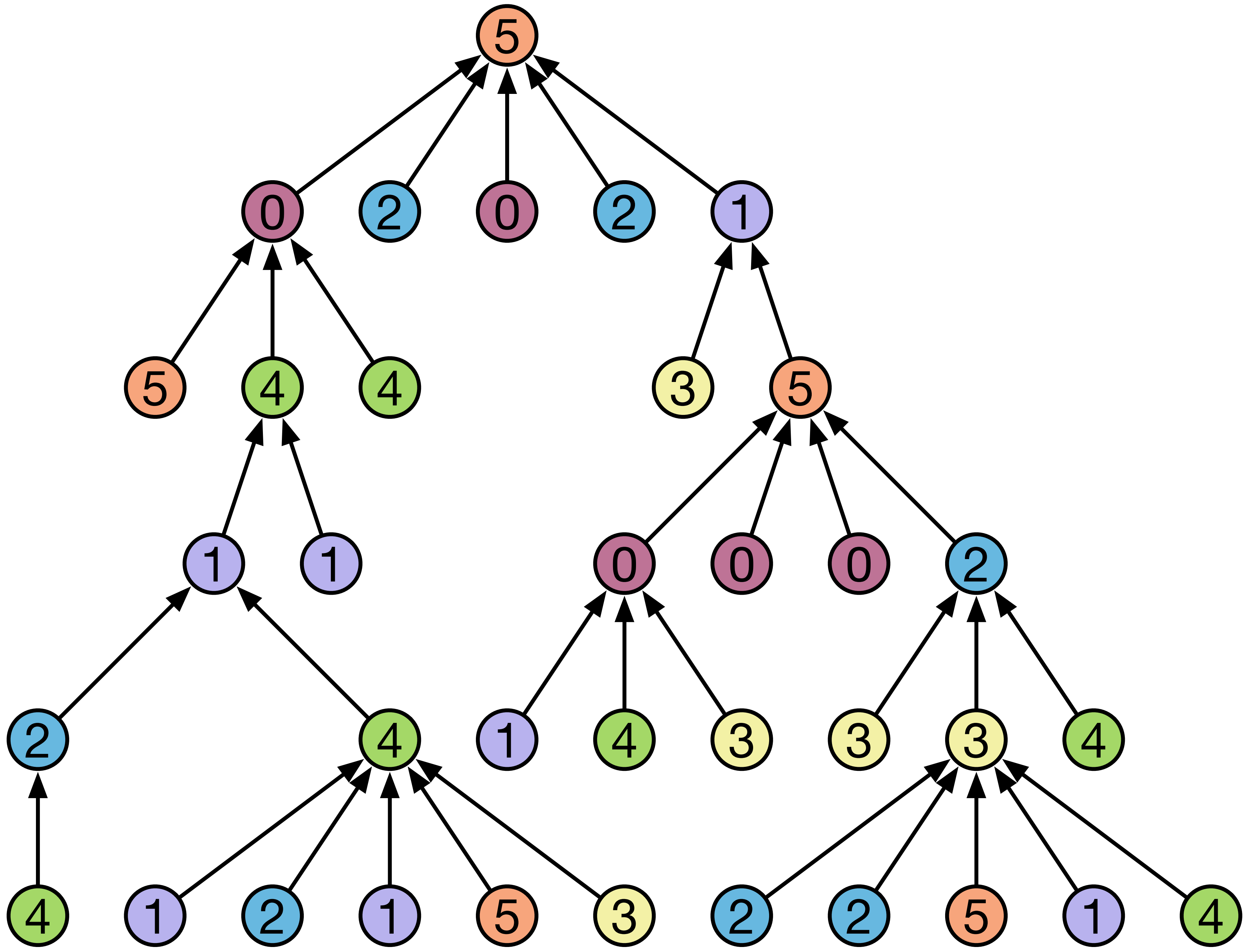
- We have seen that computing a **2-coloring** requires $\Omega(D)$
- We have seen how to compute a **6-coloring** in $O(\log^* n)$ rounds
- What about **3, 4, and 5 colors**?

Reducing from 6 to 5 colors:

- Can we recolour nodes with color 5 with a smaller color?
  - If $\Delta \leq 4$, for every node with color 5 there is a free color in {0, … , 4} available: recolor them in parallel in one round
  - What can we do if $\Delta > 4$?

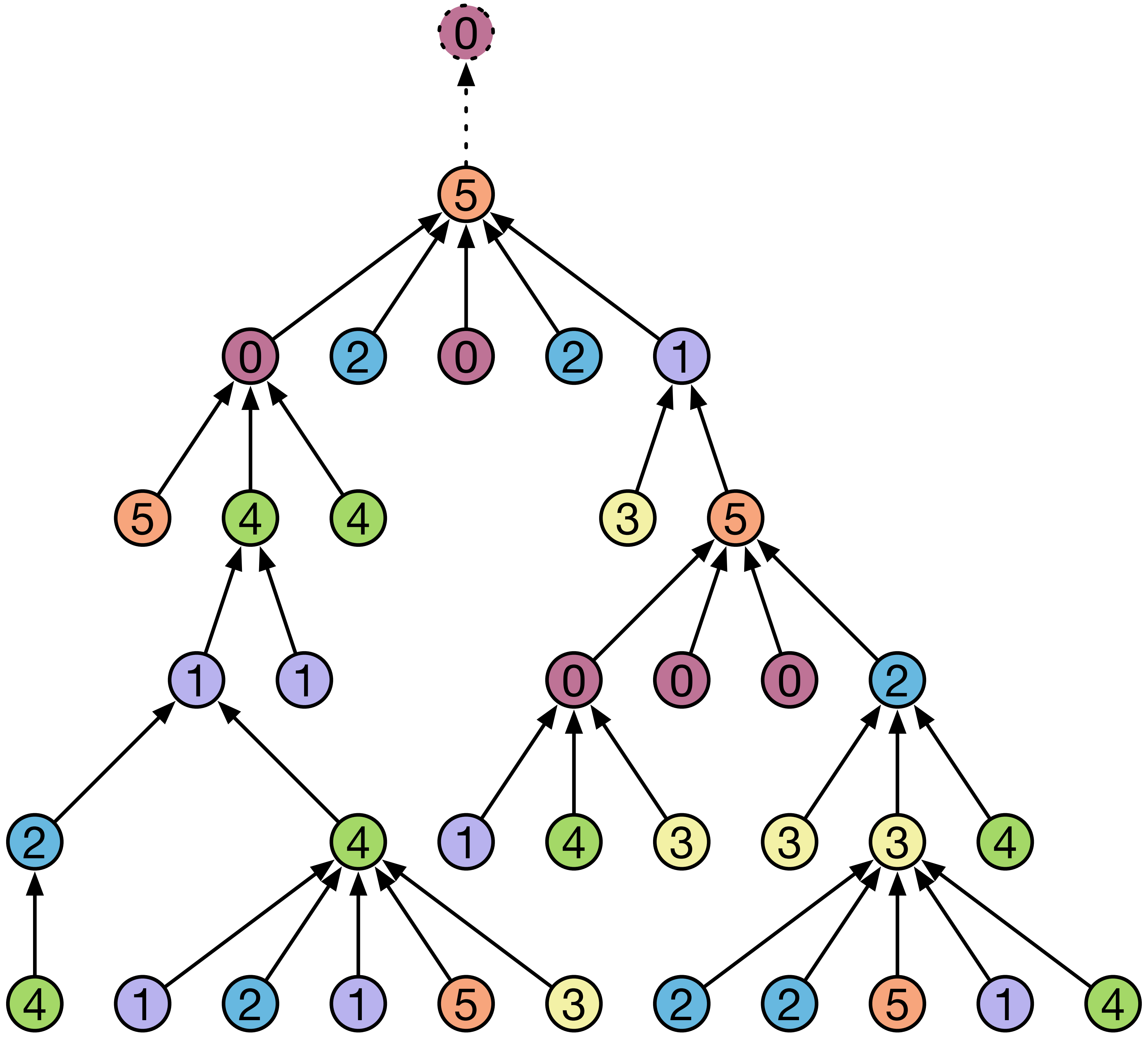
From six to five colors

- Consider a rooted tree colored with 6 colors from {0, … , 5}
- Can we **get rid of color 5**?
- **Solution**: **shift down colors**



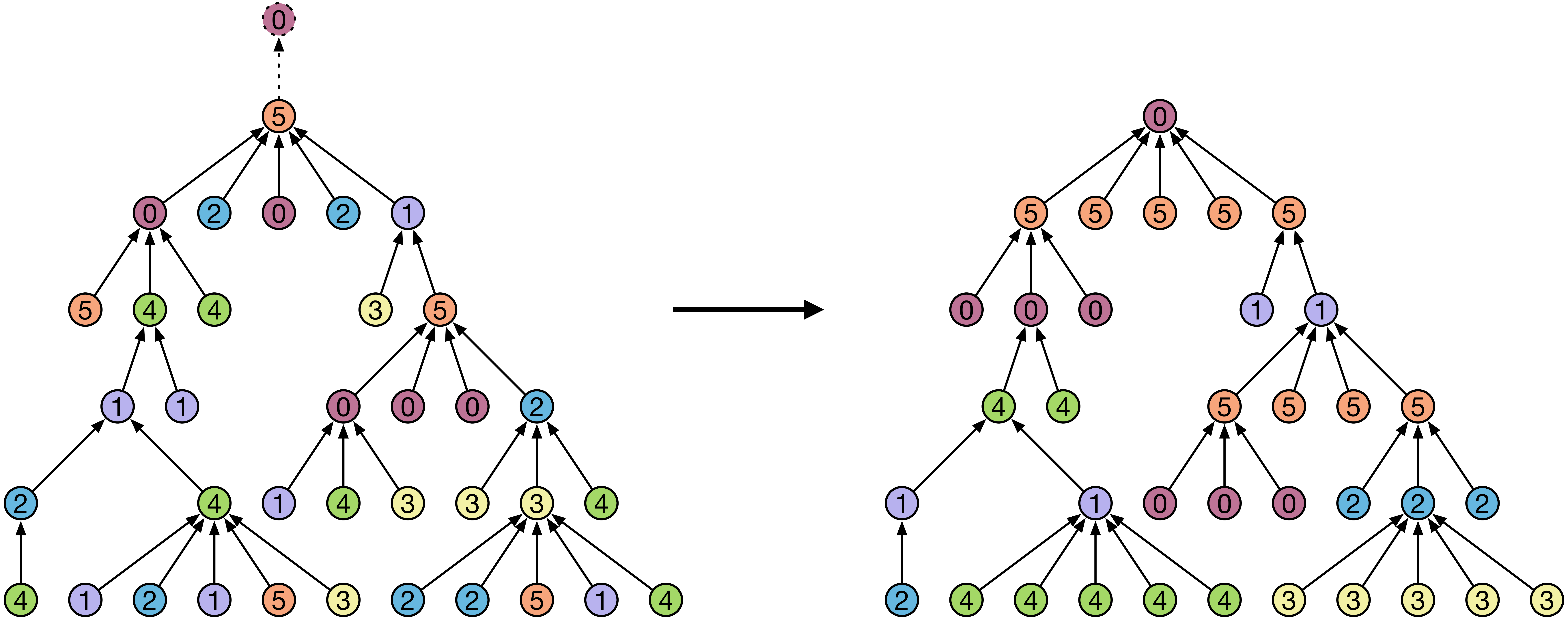
From six to five colors

- Consider a rooted tree colored with 6 colors from {0, … , 5}
- Can we **get rid of color 5**?
- **Solution**: **shift down colors**



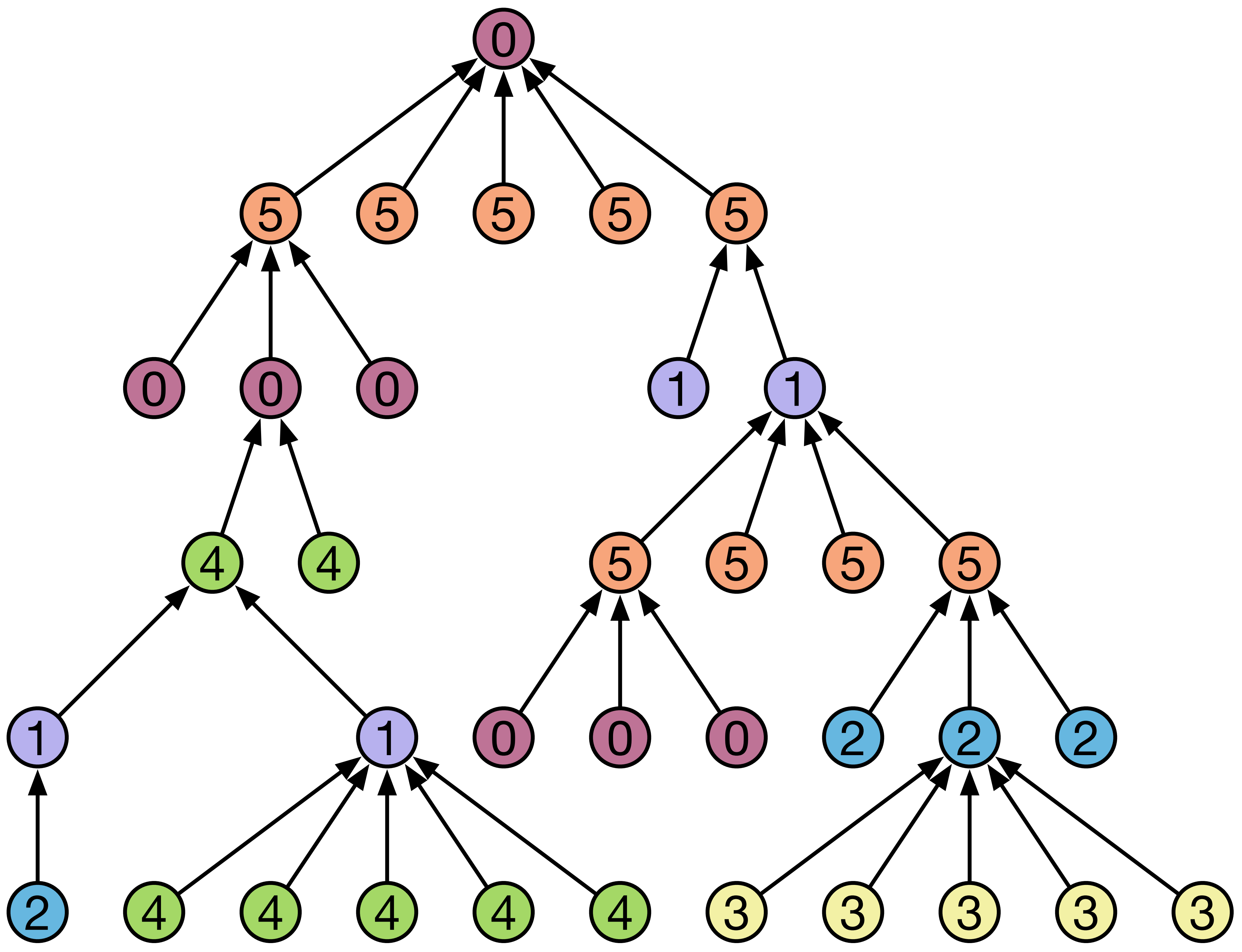
From six to five colors

- Consider a rooted tree colored with 6 colors from {0, … , 5}
- Can we **get rid of color 5**?
- **Solution**: **shift down colors**



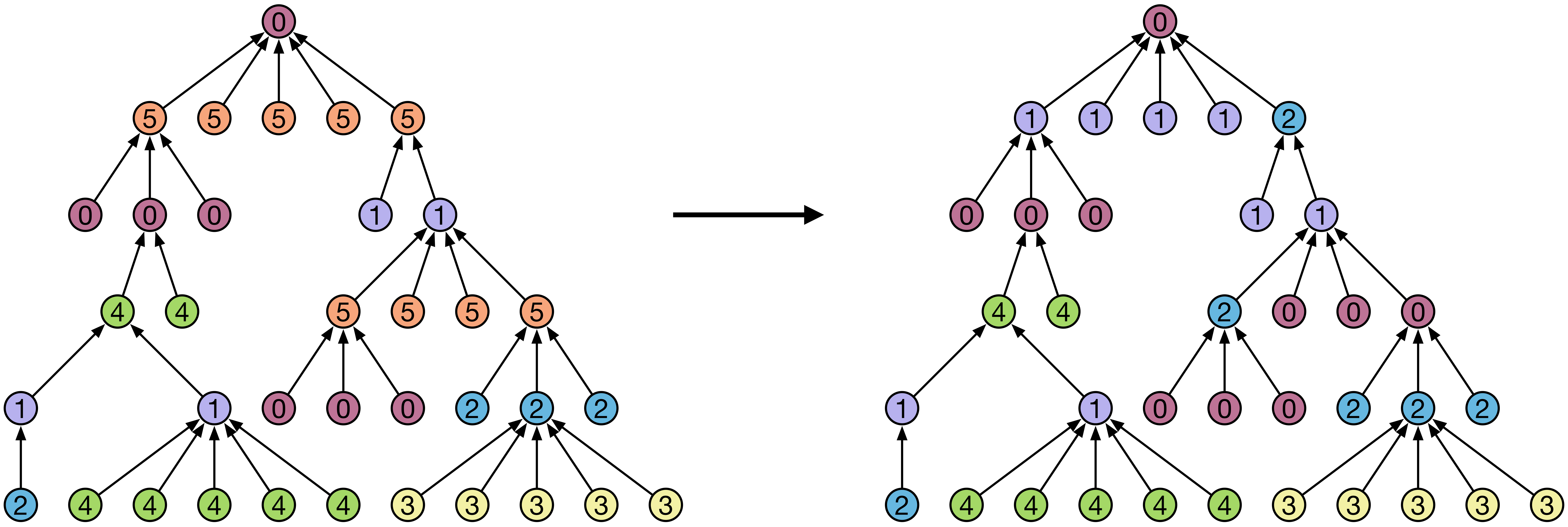
From six to five colors

- Consider a rooted tree colored with 6 colors from {0, … , 5}
- Can we **get rid of color 5**?
- **Solution**: **shift down colors**



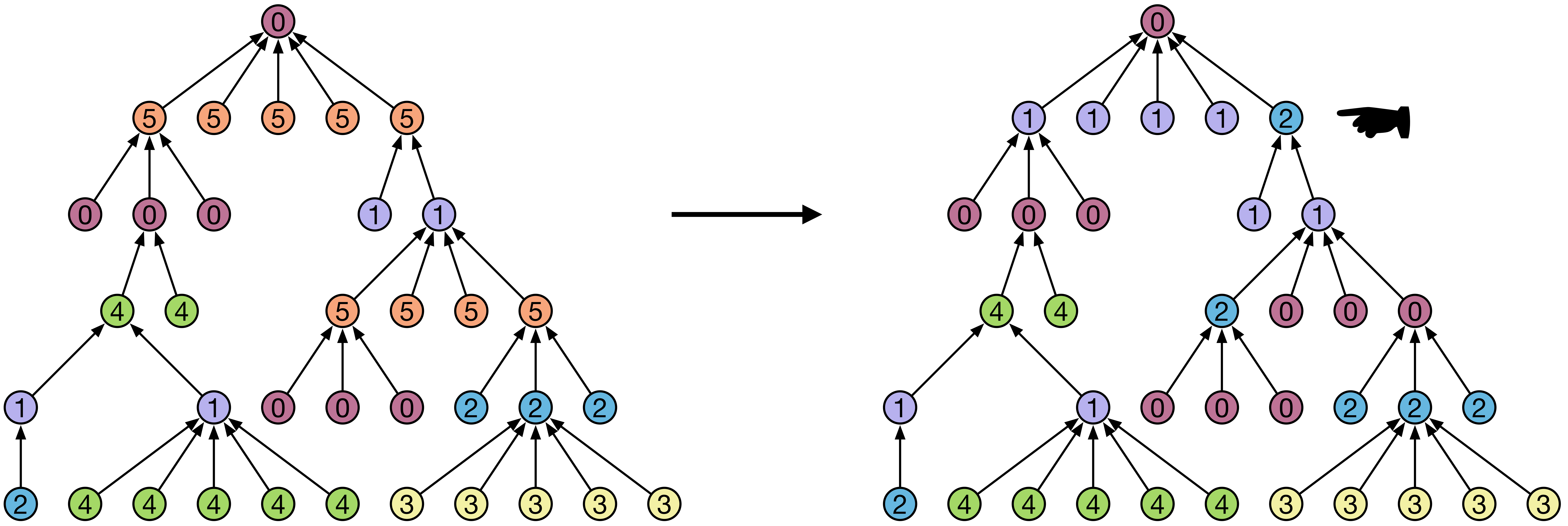
From six to five colors

- Consider a rooted tree colored with 6 colors from {0, … , 5}
- Can we **get rid of color 5**?
- **Solution**: **shift down colors**



From six to five colors

- Consider a rooted tree colored with 6 colors from {0, … , 5}
- Can we **get rid of color 5**?
- **Solution**: **shift down colors**



From six to three colors

Color reduction phase for rooted trees

1. **Shift-down step**

2. **Color reduction step**

Theorem: As long as the number of **colors C** is **larger than three**, we can **reduce** the number of **colors by one in two rounds**

Rooted trees: coloring and MIS

Cole-Vishkin (to get 6-coloring) + color reduction = 3-coloring

Theorem: When starting with colors in {0, … , n - 1}, there is a distributed algorithm to computes a **3-coloring of a rooted tree** in **$O(\log^* n)$** rounds

- Unique IDs in {0, … , n - 1} can be used as an initial coloring

Rooted trees: coloring and MIS

Cole-Vishkin (to get 6-coloring) + color reduction = 3-coloring

Theorem: When starting with colors in $\{0, \ldots, n - 1\}$, there is a distributed algorithm to computes a **3-coloring of a rooted tree** in $O(\log^* n)$ rounds

- Unique IDs in $\{0, \ldots, n - 1\}$ can be used as an initial coloring

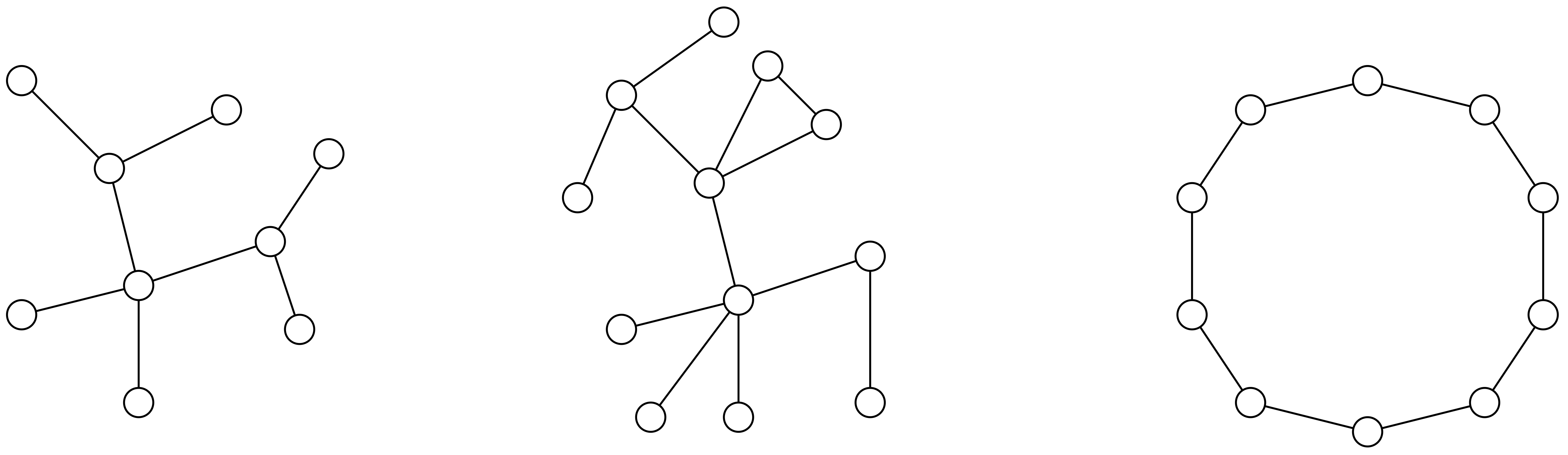
Theorem: When starting with colors in $\{0, \ldots, n - 1\}$, there is a distributed algorithm to computes an **MIS of a rooted tree** in $O(\log^* n)$ rounds

- One first computes a 6-coloring (or a 3-coloring)
- Then an MIS can be computed in $O(1)$ rounds
  - We have seen before that from a C-coloring we get MIS in C rounds

Coloring directed pseudoforests

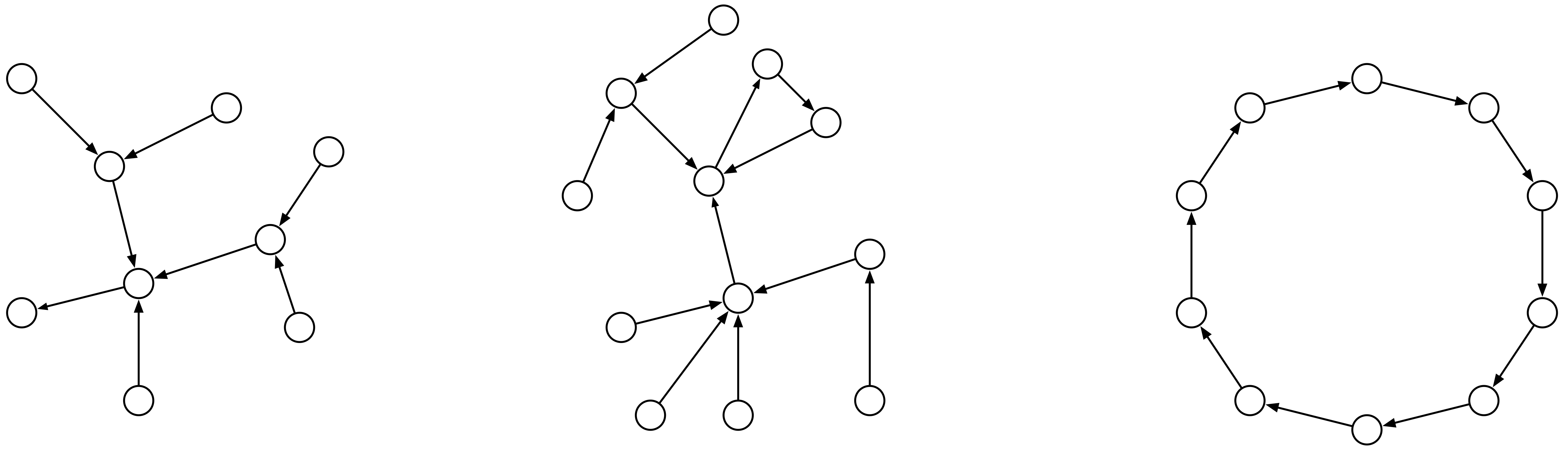
Pseudoforest

- A graph in which each connected component has at most one cycle



Directed pseudoforest

- A graph where the out-degree of every node is at most 1



Coloring directed pseudoforests

Directed pseudoforest

- A graph where the out-degree of every node is at most 1

Claim: The **3-coloring** algorithm for rooted trees can also be applied in a **directed pseudoforest**

Coloring directed pseudoforests

Directed pseudoforest

- A graph where the out-degree of every node is at most 1

Claim: The **3-coloring** algorithm for rooted trees can also be applied in a **directed pseudoforest**

- The **Cole-Vishkin** algorithm works as before
  - Nodes with out-degree 1 treat their out-neighbors as parent
  - Other nodes behave like the root and imagine an out-neighbor with some color

Coloring directed pseudoforests

Directed pseudoforest

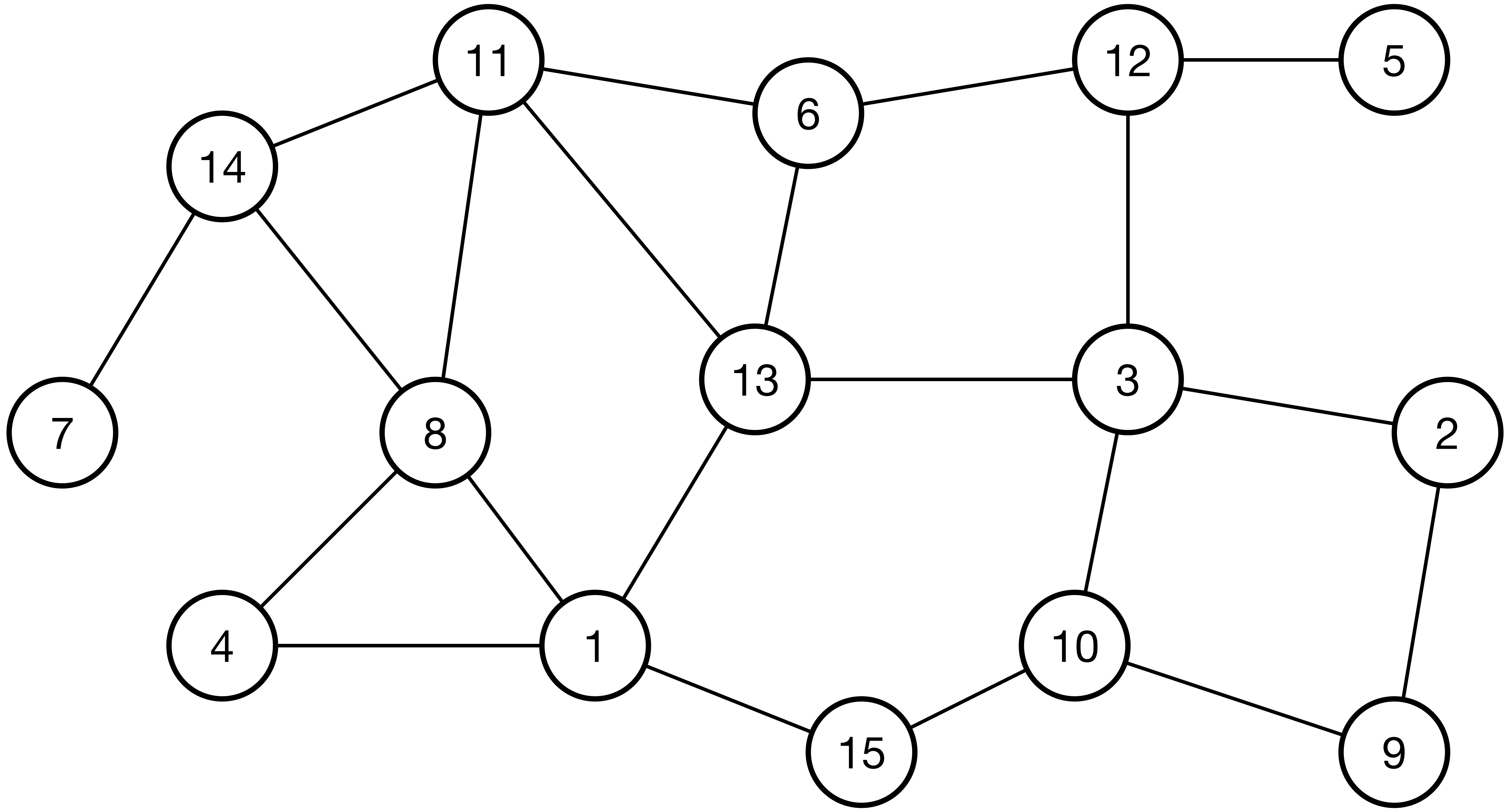
- A graph where the out-degree of every node is at most 1

Claim: The **3-coloring** algorithm for rooted trees can also be applied in a **directed pseudoforest**

- The **Cole-Vishkin** algorithm works as before
  - Nodes with out-degree 1 treat their out-neighbors as parent
  - Other nodes behave like the root and imagine an out-neighbor with some color
- The **color reduction** algorithm also works in the same way
  - Shift-down: Every node with out-degree 1 picks the color of their out-neighbor, every other node just picks a new color (either 0 or 1)
  - All in-neighbors of a node then have the same color and each node therefore only sees 2 different colors among its neighbors

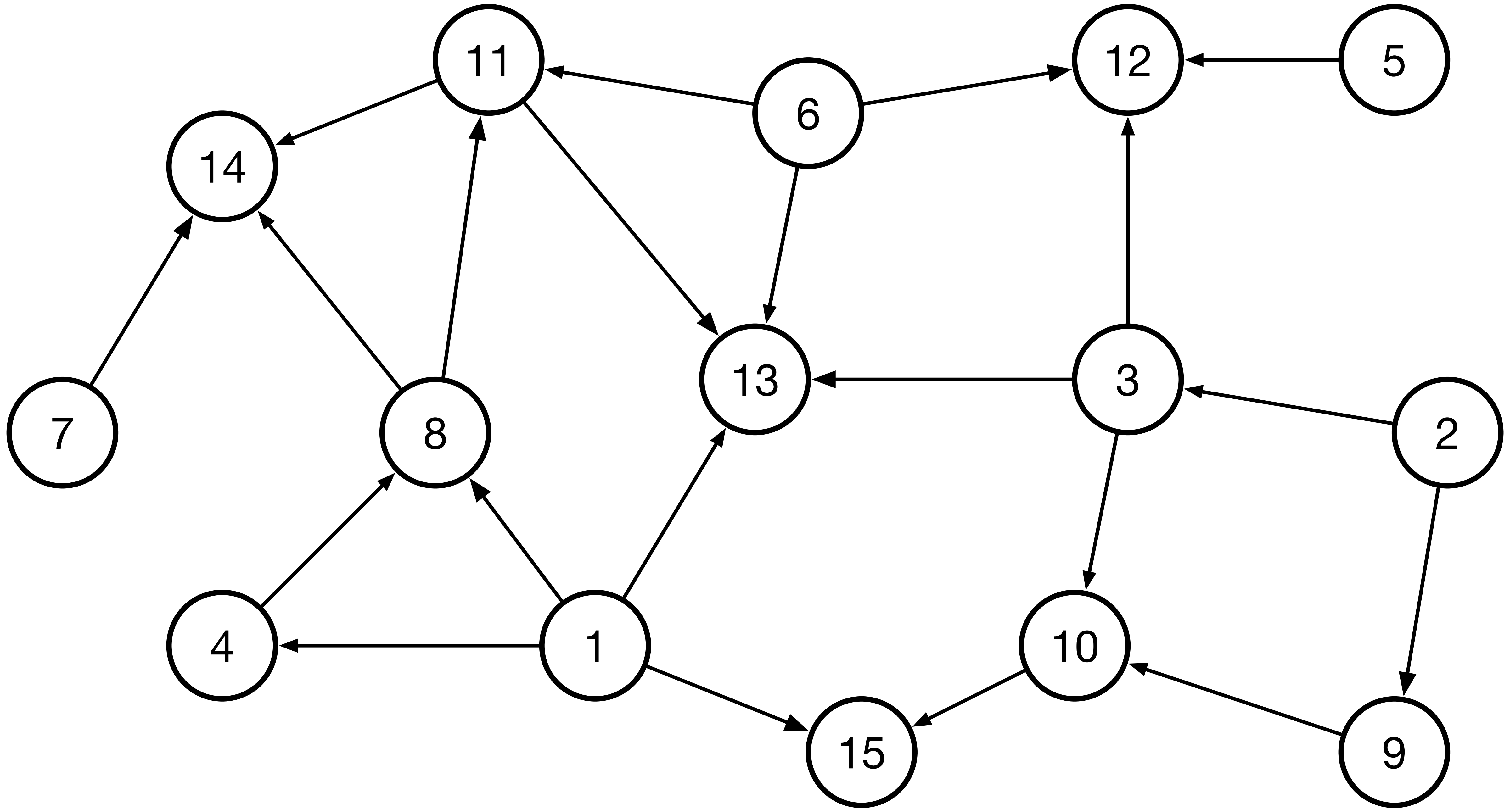
Coloring graphs with maximum degree Δ

- We first **orient each edge** on the graph arbitrarily
  - E.g., orient edge $\{u, v\}$ from u to v iff $ID(u) < ID(v)$
- Assume that a node v has d_v out-degree edges. Node v labels these edges **from 1 to d_v** (note that $d_v \leq \Delta$)



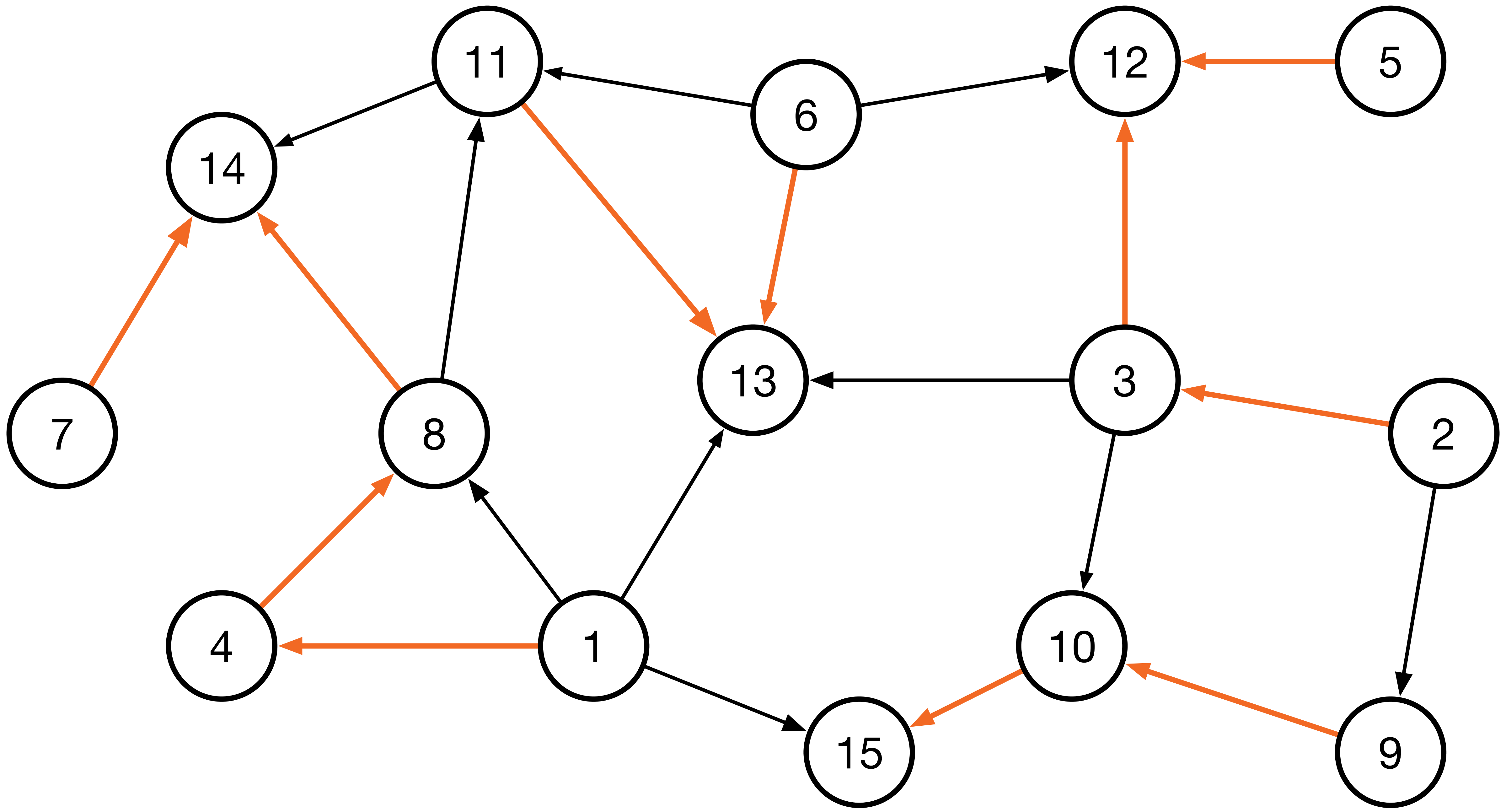
Coloring graphs with maximum degree Δ

- We first **orient each edge** on the graph arbitrarily
  - E.g., orient edge $\{u, v\}$ from u to v iff $ID(u) < ID(v)$
- Assume that a node v has d_v out-degree edges. Node v labels these edges **from 1 to d_v** (note that $d_v \leq \Delta$)



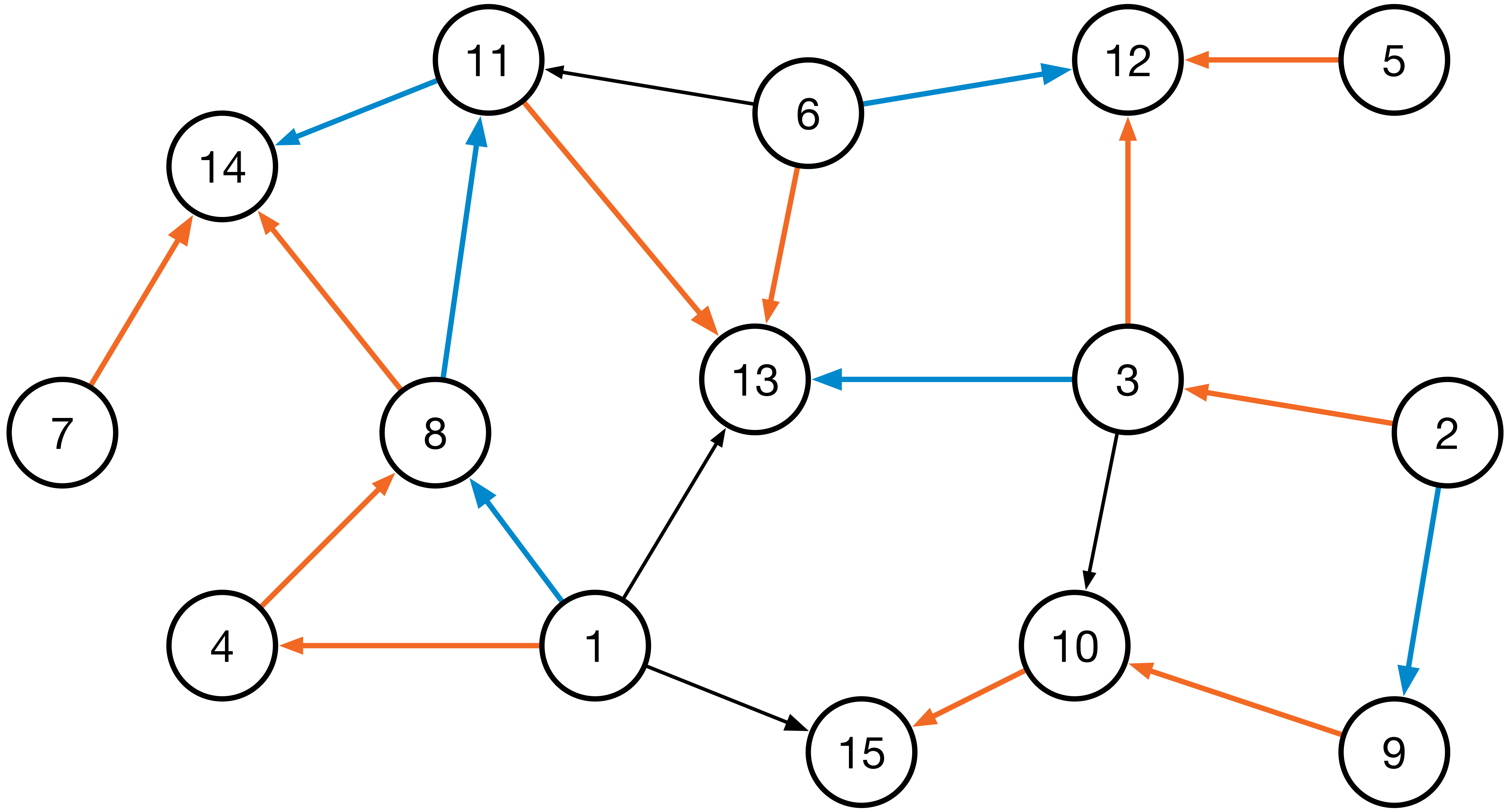
Coloring graphs with maximum degree Δ

- We first **orient each edge** on the graph arbitrarily
  - E.g., orient edge $\{u, v\}$ from u to v iff $ID(u) < ID(v)$
- Assume that a node v has d_v out-degree edges. Node v labels these edges **from 1 to d_v** (note that $d_v \leq \Delta$)



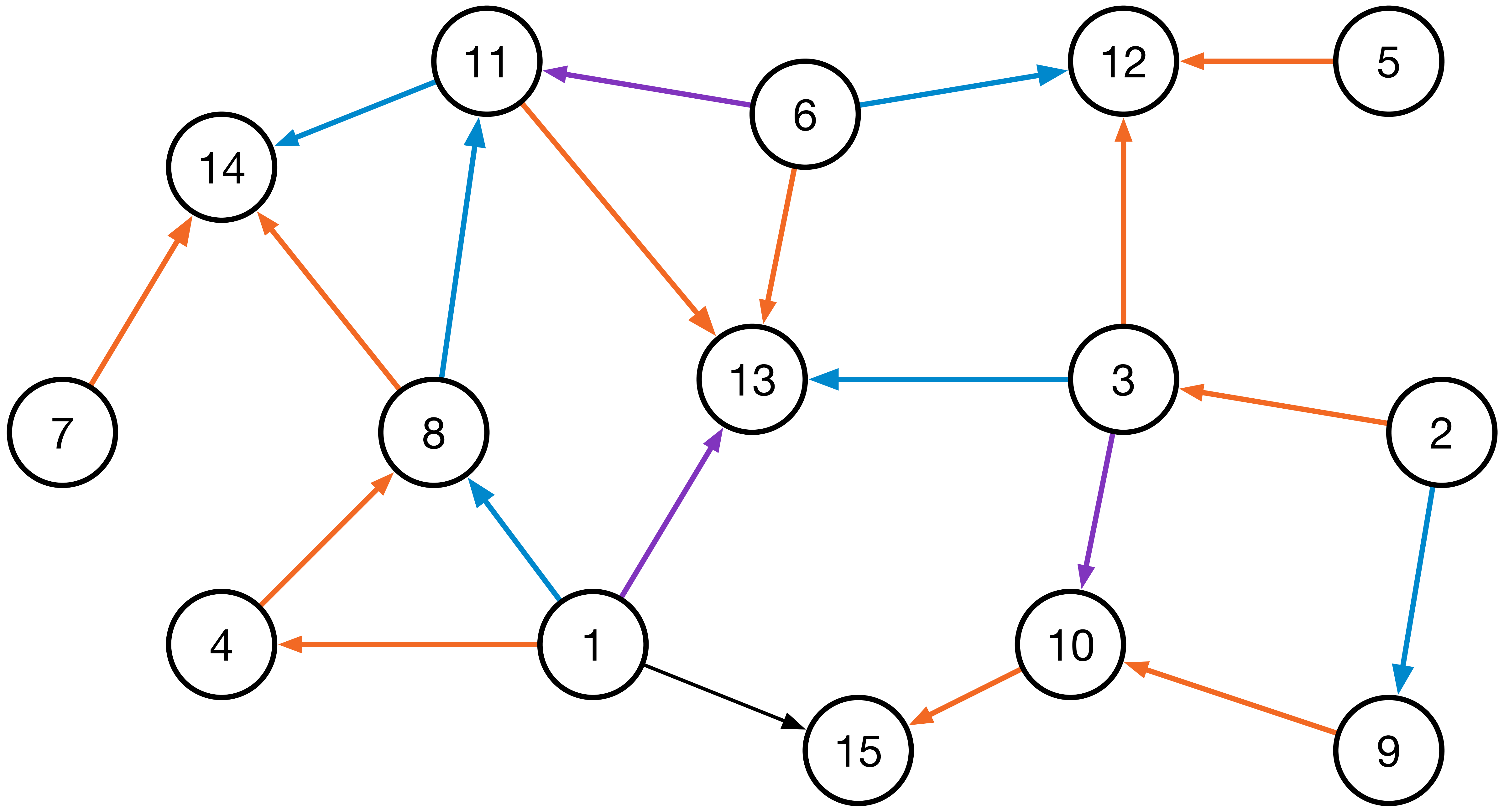
Coloring graphs with maximum degree Δ

- We first **orient each edge** on the graph arbitrarily
  - E.g., orient edge $\{u, v\}$ from u to v iff $ID(u) < ID(v)$
- Assume that a node v has d_v out-degree edges. Node v labels these edges **from 1 to d_v** (note that $d_v \leq \Delta$)



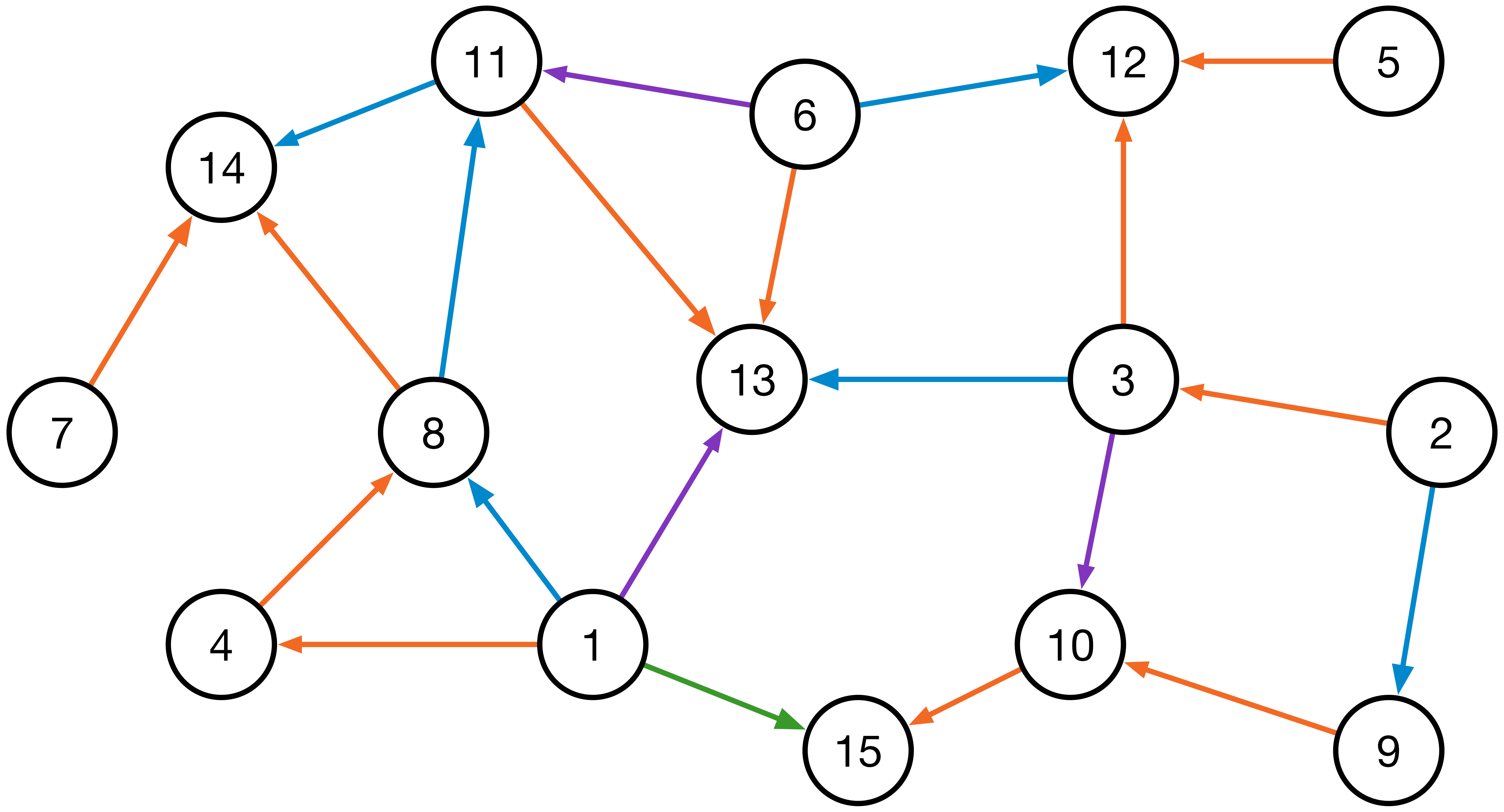
Coloring graphs with maximum degree Δ

- We first **orient each edge** on the graph arbitrarily
  - E.g., orient edge $\{u, v\}$ from u to v iff ID(u) < ID(v)
- Assume that a node v has d_v out-degree edges. Node v labels these edges **from 1 to d_v** (note that $d_v \leq \Delta$)



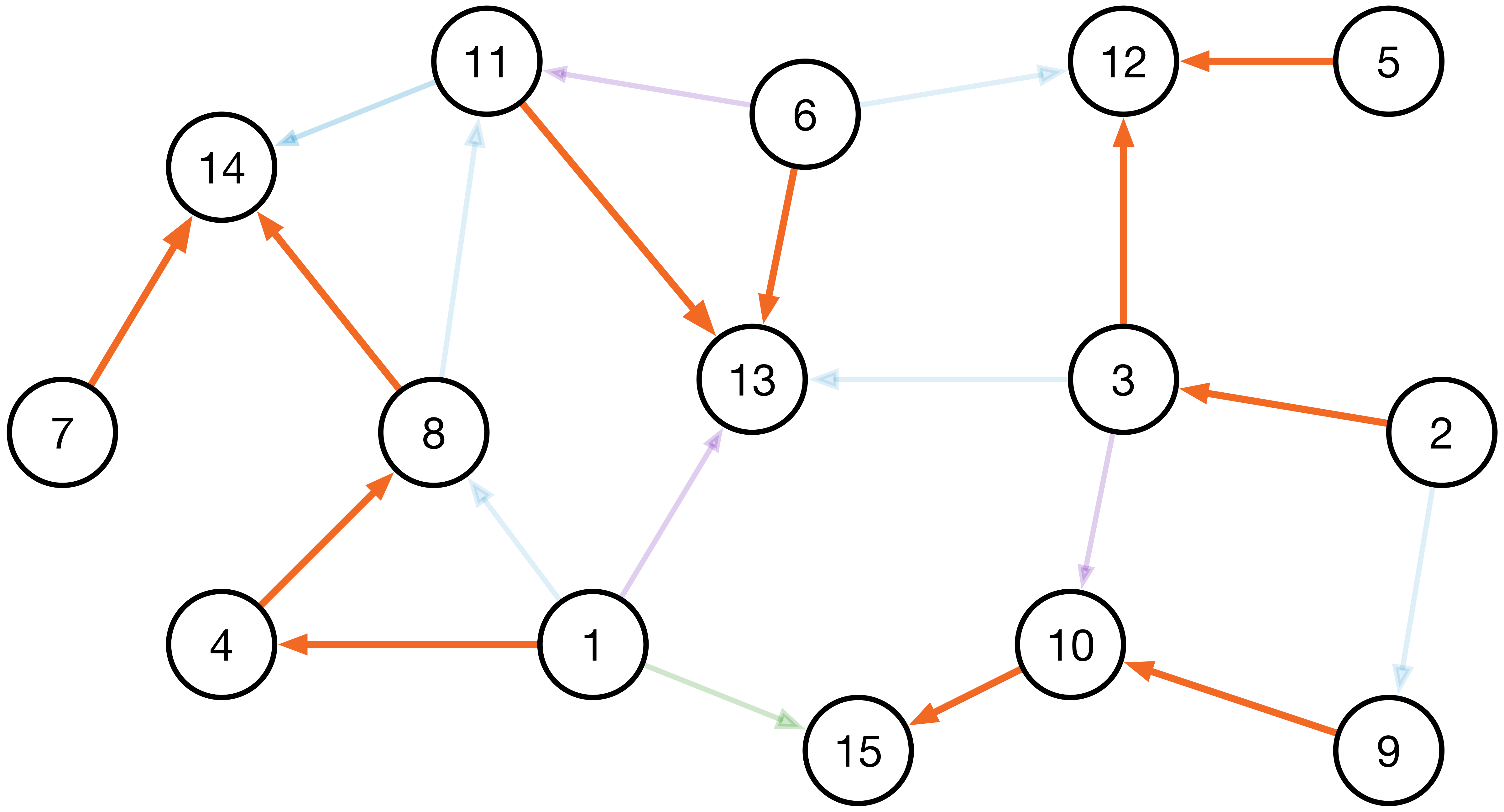
Coloring graphs with maximum degree Δ

- We first **orient each edge** on the graph arbitrarily
  - E.g., orient edge {u, v} from u to v iff ID(u) < ID(v)
- Assume that a node v has d_v out-degree edges. Node v labels these edges **from 1 to d_v** (note that $d_v \leq \Delta$)



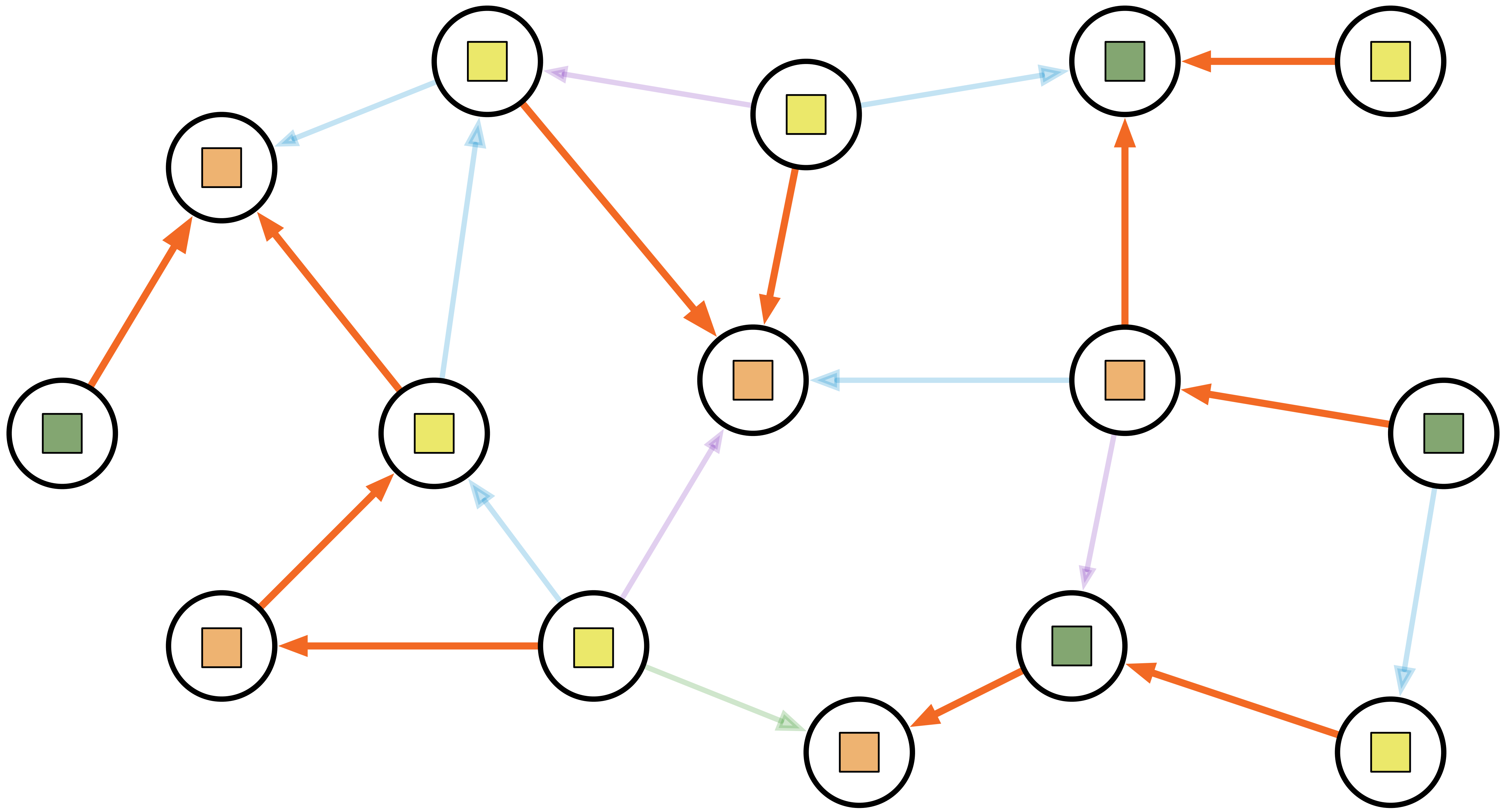
Coloring graphs with maximum degree Δ

- Every node has **at most one outgoing edge** for each label
- For all $i \in \{1, \ldots, \Delta\}$, compute a **3-coloring of G_i** in **$O(\log^* n)$** rounds



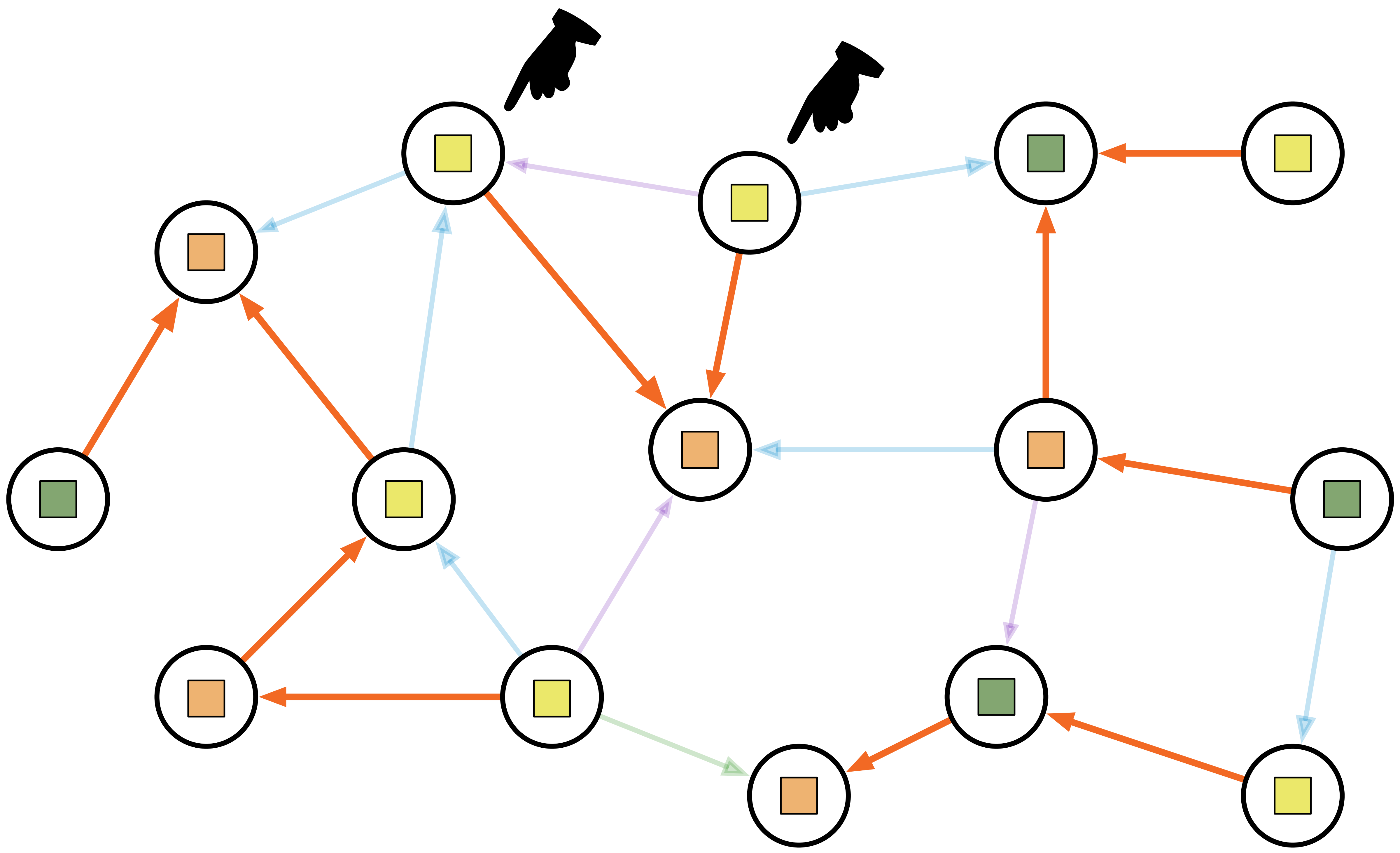
Coloring graphs with maximum degree Δ

- Every node has **at most one outgoing edge** for each label
- For all $i \in \{1, \ldots, \Delta\}$, compute a **3-coloring of G_i** in **$O(\log^* n)$** rounds



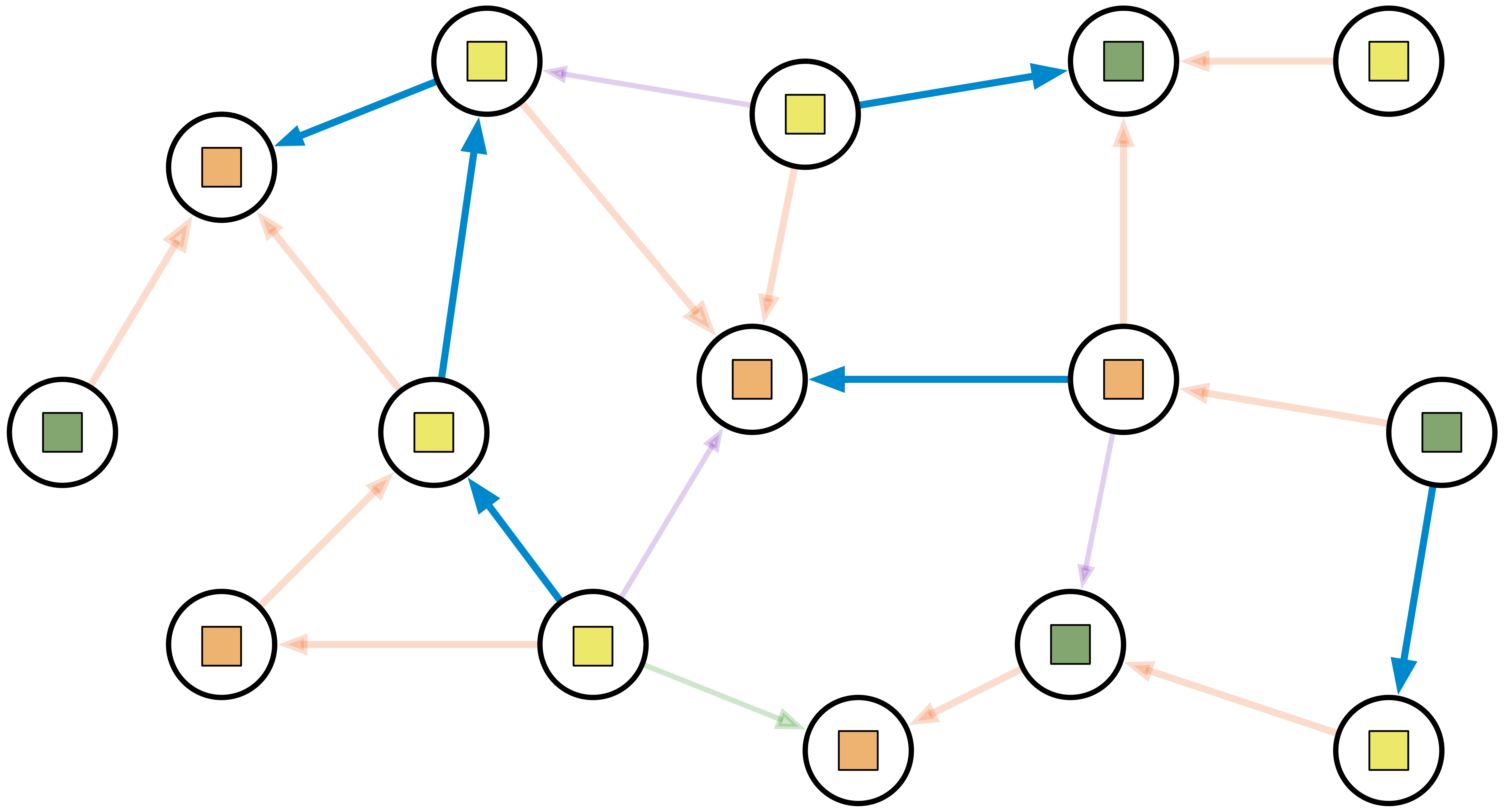
Coloring graphs with maximum degree Δ

- Every node has **at most one outgoing edge** for each label
- For all $i \in \{1, \dots, \Delta\}$, compute a **3-coloring of G_i** in **$O(\log^* n)$** rounds



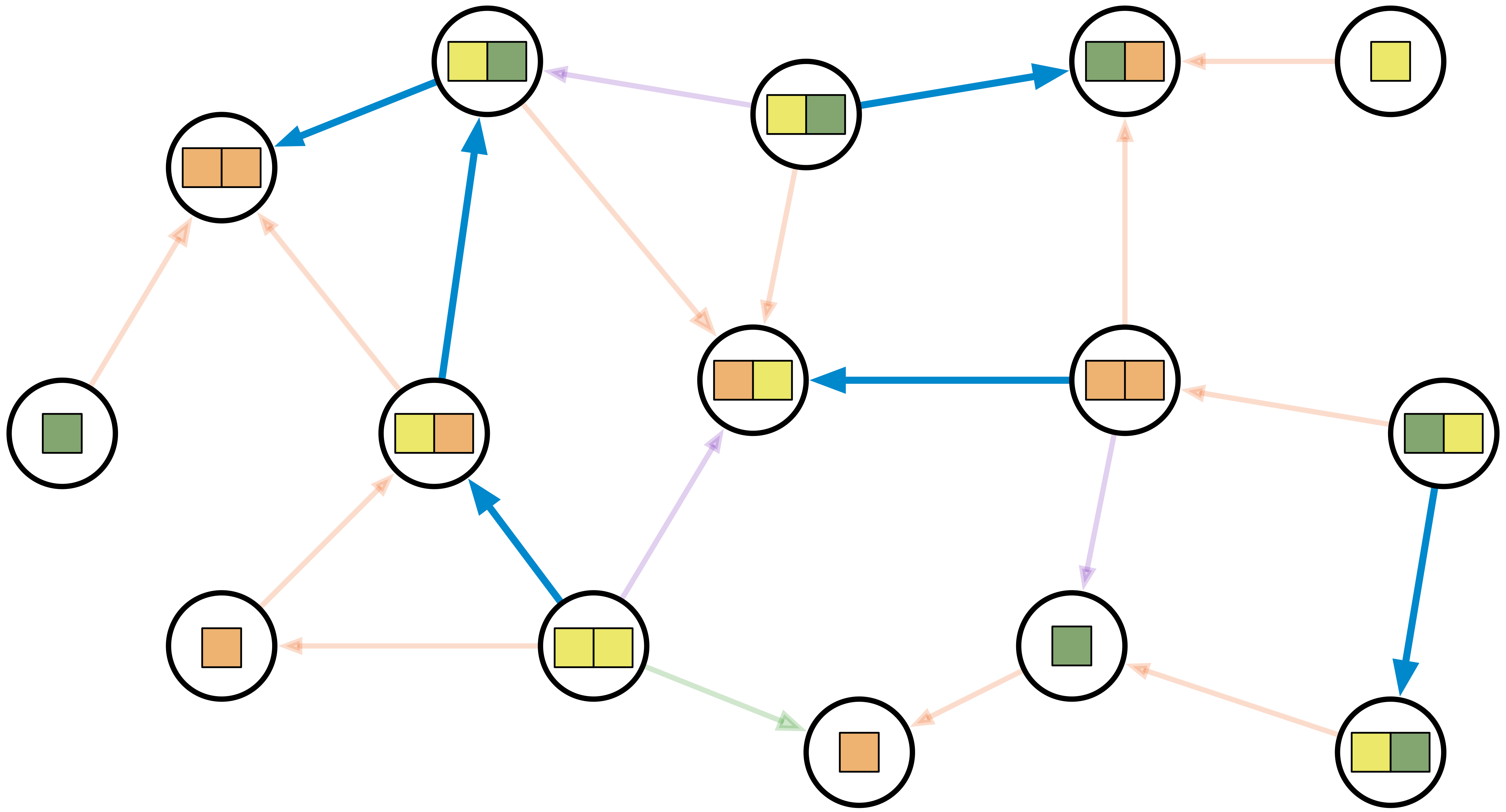
Coloring graphs with maximum degree Δ

- Every node has **at most one outgoing edge** for each label
- For all $i \in \{1, \ldots, \Delta\}$, compute a **3-coloring of G_i** in **$O(\log^* n)$** rounds



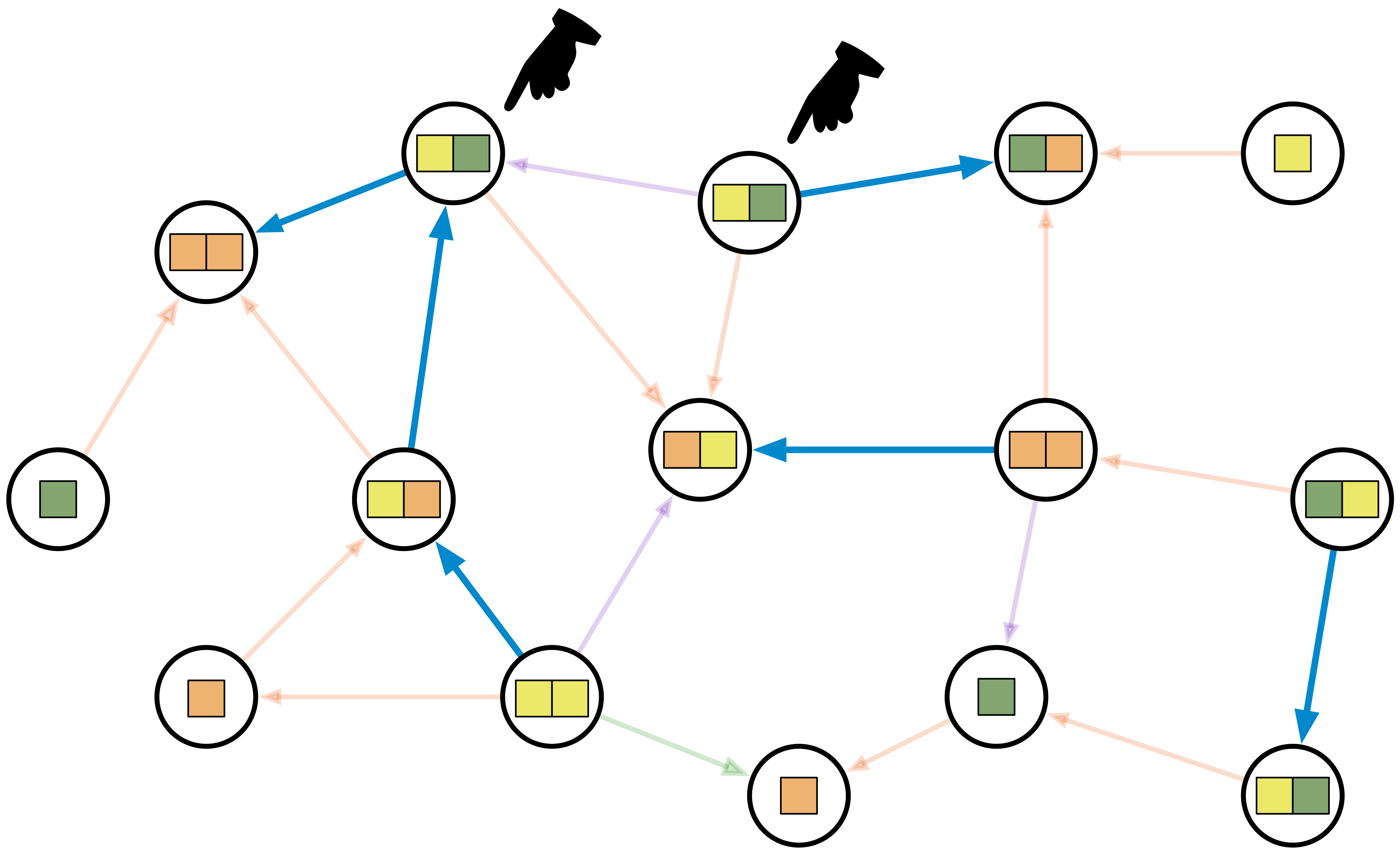
Coloring graphs with maximum degree Δ

- Every node has **at most one outgoing edge** for each label
- For all $i \in \{1, \dots, \Delta\}$, compute a **3-coloring of G_i** in **$O(\log^* n)$** rounds



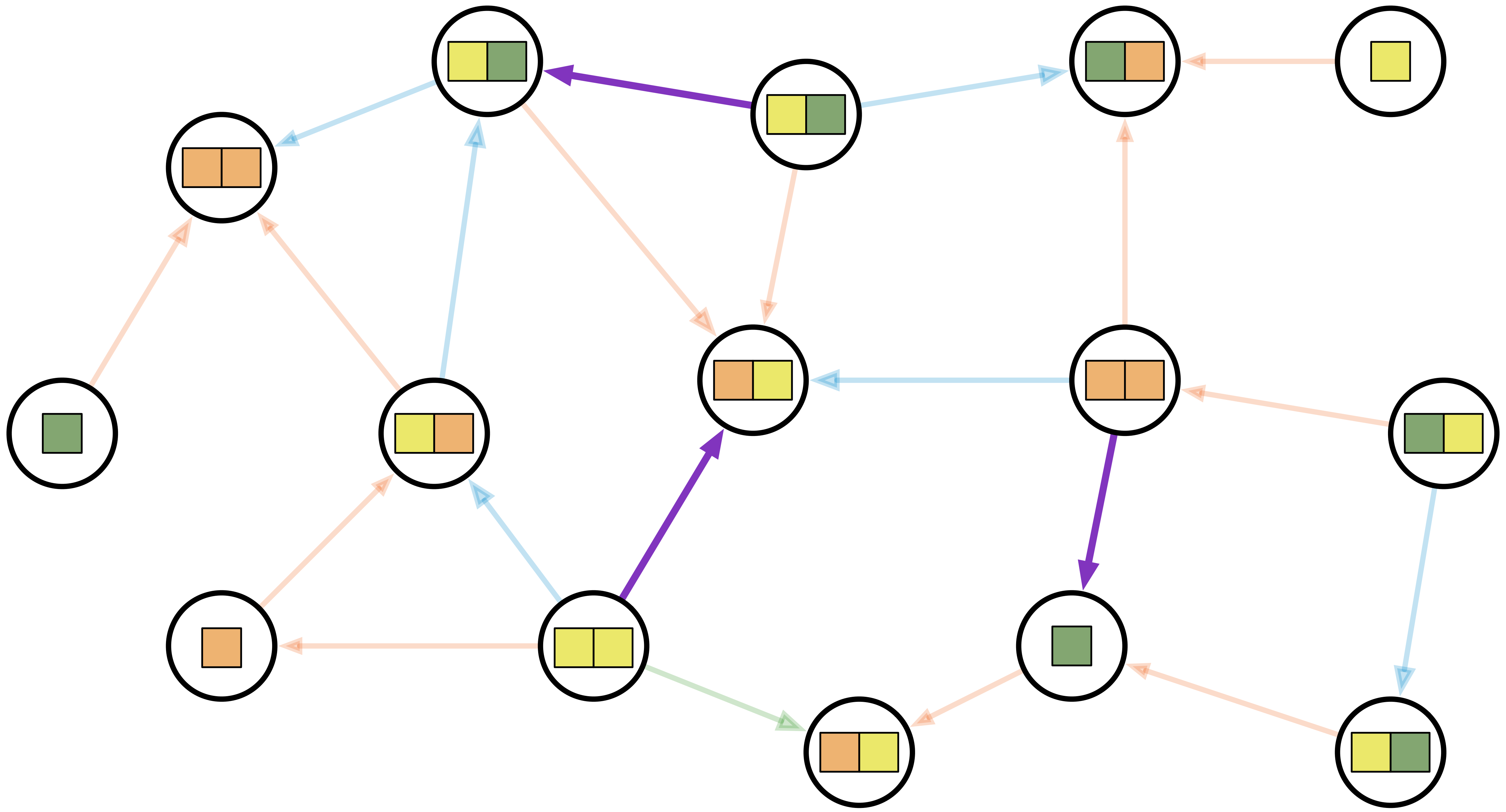
Coloring graphs with maximum degree Δ

- Every node has **at most one outgoing edge** for each label
- For all $i \in \{1, \dots, \Delta\}$, compute a **3-coloring of G_i** in **$O(\log^* n)$** rounds



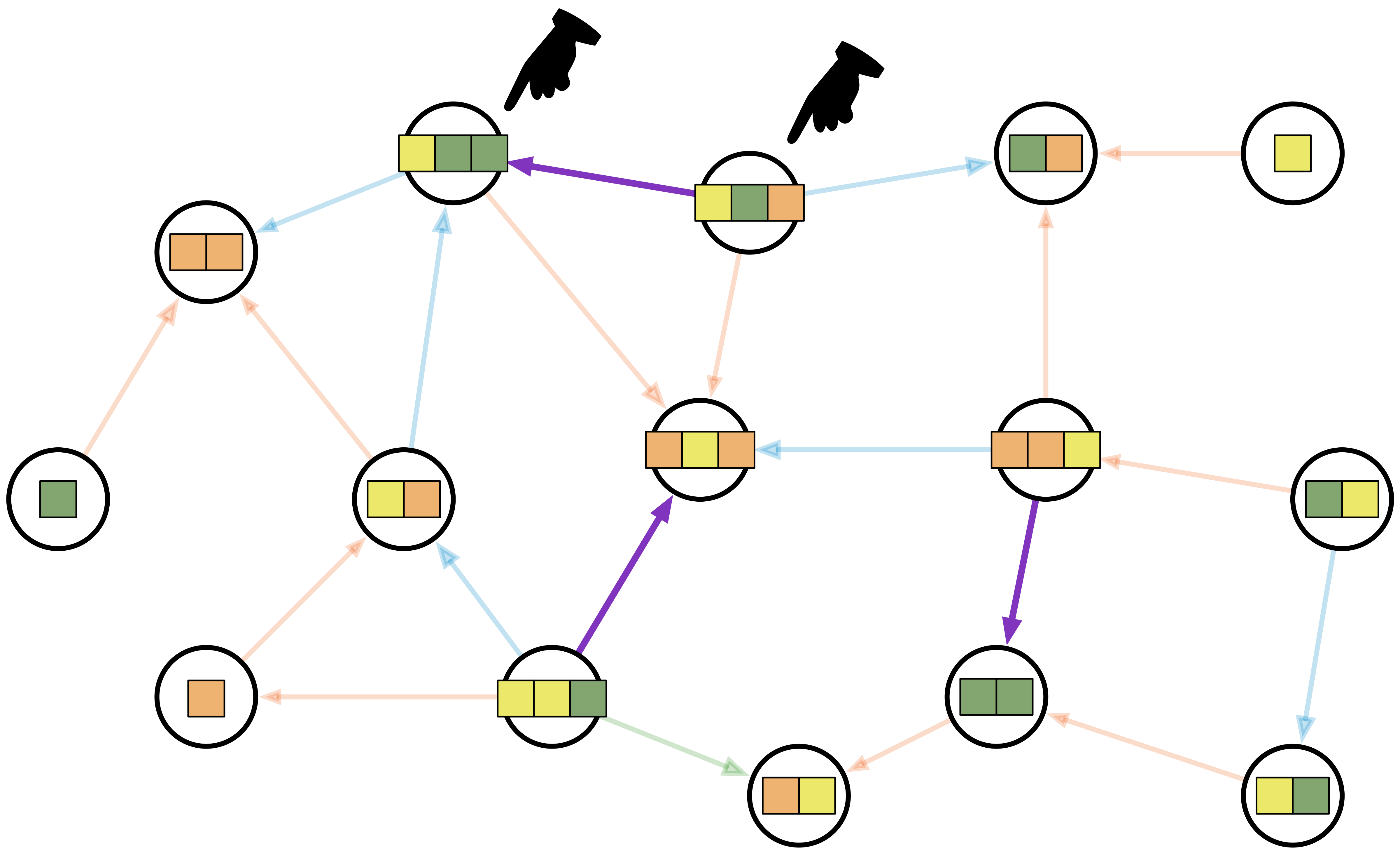
Coloring graphs with maximum degree Δ

- Every node has **at most one outgoing edge** for each label
- For all $i \in \{1, \ldots, \Delta\}$, compute a **3-coloring of G_i** in **$O(\log^* n)$** rounds



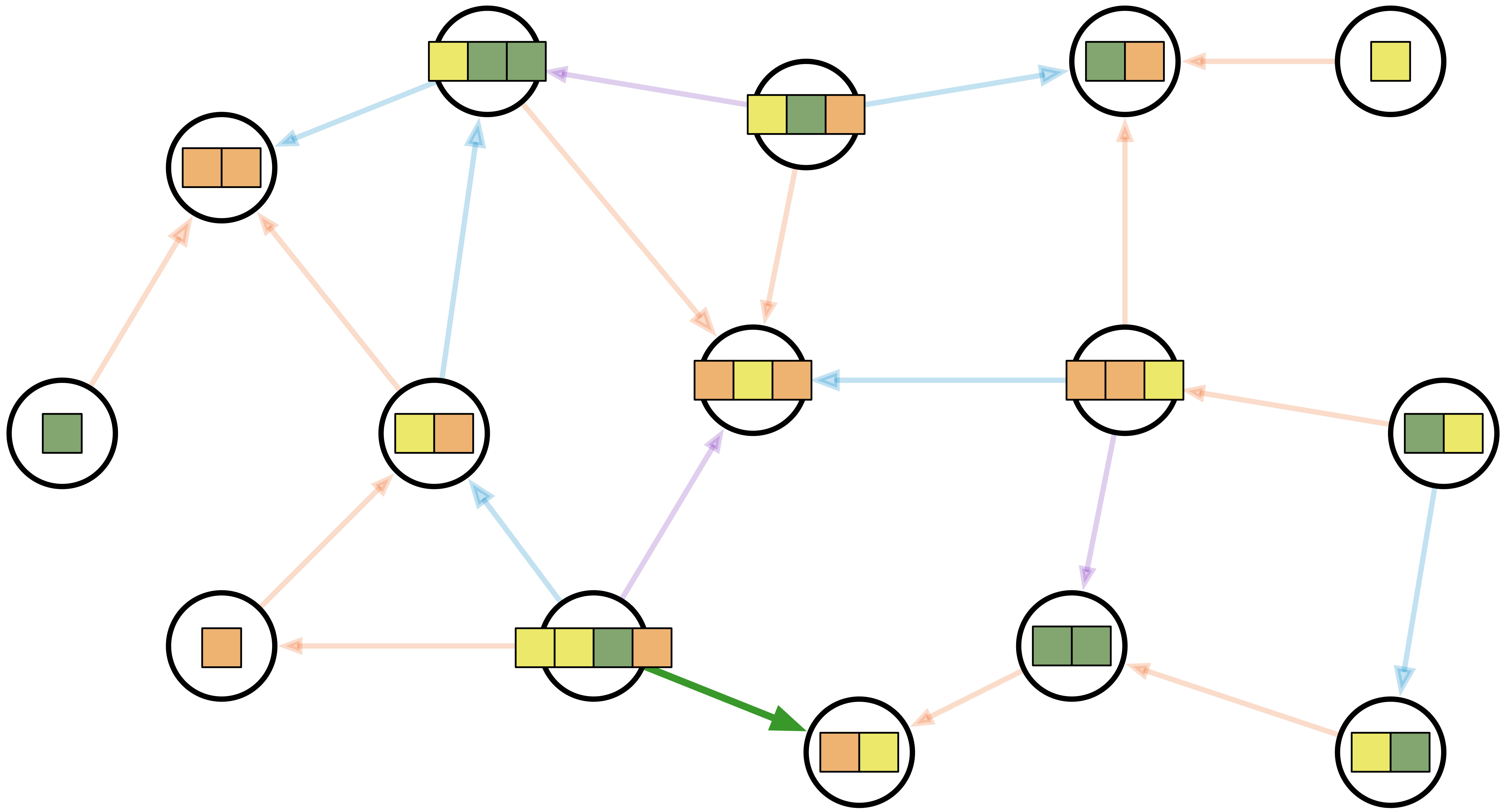
Coloring graphs with maximum degree Δ

- Every node has **at most one outgoing edge** for each label
- For all $i \in \{1, \dots, \Delta\}$, compute a **3-coloring of G_i** in **$O(\log^* n)$** rounds



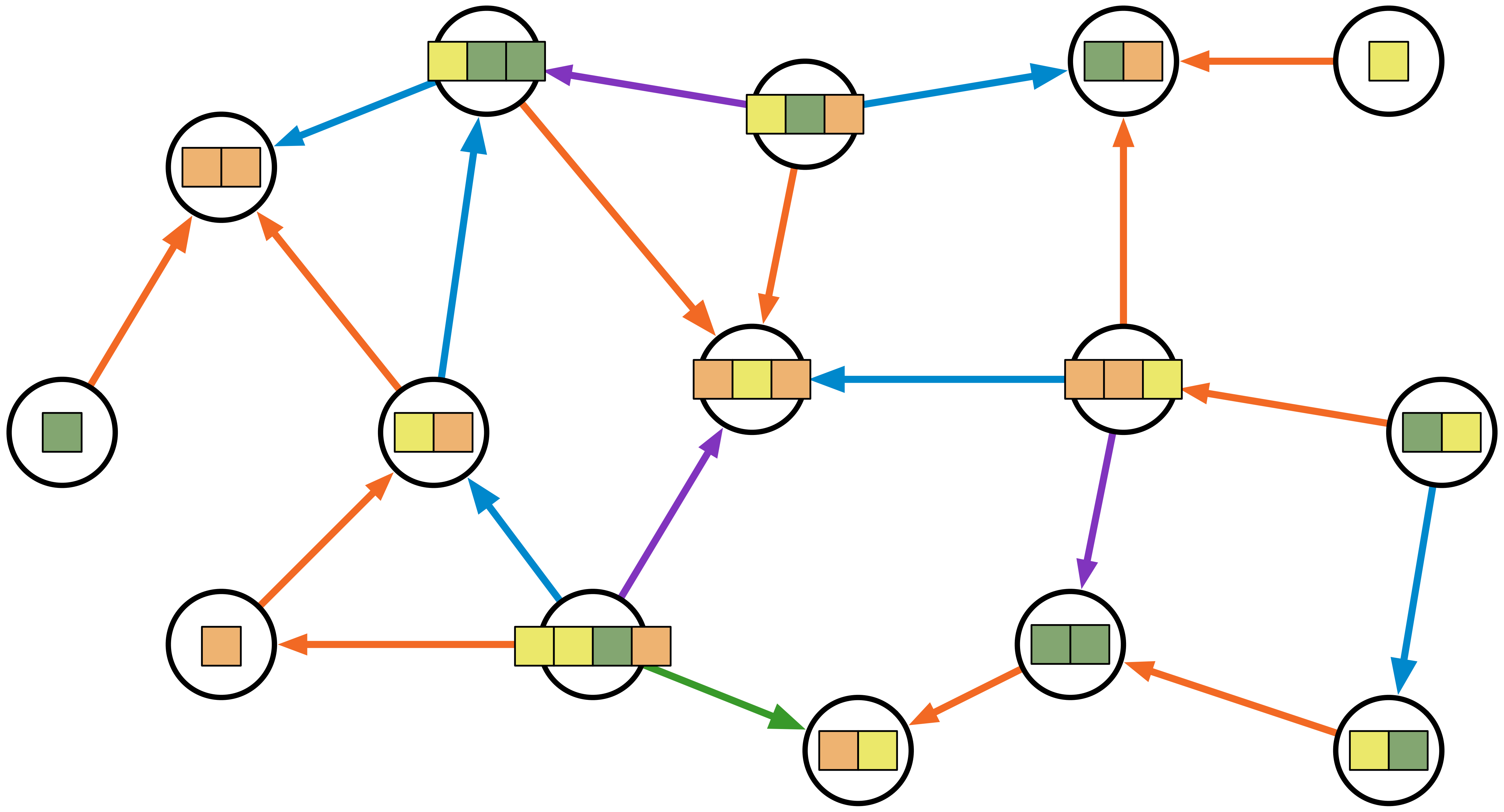
Coloring graphs with maximum degree Δ

- Every node has **at most one outgoing edge** for each label
- For all $i \in \{1, \dots, \Delta\}$, compute a **3-coloring of G_i** in **$O(\log^* n)$** rounds



Coloring graphs with maximum degree Δ

- Every node $v \in V$ then gets a vector **$c_v \in \{0, 1, 2\}^\Delta$** of colors, where $c_{v,i}$ is the color of v in graph G_i
- For every two neighbors u and v, we have **$c_u \neq c_v$**
  - If the edge $\{u, v\}$ has label i, we have $c_{u,i} \neq c_{v,i}$



Coloring graphs with maximum degree Δ

Theorem: For a graph with maximum degree Δ, there is a distributed algorithm to compute a **3^{Δ}-coloring in $O(\log^* n)$ rounds**

Coloring graphs with maximum degree Δ

Theorem: For a graph with maximum degree Δ, there is a distributed algorithm to compute a **3^{Δ}-coloring in $O(\log^* n)$ rounds**

- As we saw, the ***n*** in $O(\log^* n)$ represent the size of **initial input coloring**
- Usually, we assume that the **IDs** represent the **initial input coloring**, but **how large can the ID space be**?
  - **Usual assumption**: IDs are **from 1 to n^c**, where n is the number of nodes and c is a constant
  - The algorithm would have the same runtime even if IDs were to be from 0 to $2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}$, where the power tower is of size at most $O(\log^* n)$

Coloring bounded-degree graphs

Theorem: For a graph with maximum degree Δ, there is a distributed algorithm to compute a **3^Δ-coloring in $O(\log^* n)$ rounds**

- If $\Delta = O(1)$, then $3^\Delta = O(1)$: we get a $C = 3^\Delta$ **coloring** in $O(\log^* n)$ rounds (where **C is a constant**)

- We saw that if a C-coloring is given, we can compute a $(\Delta + 1)$-coloring and an MIS in C rounds

Theorem: For a graph with maximum degree $\Delta = O(1)$, there are distributed algorithms to compute a **$(\Delta + 1)$-coloring and an MIS in $O(\log^* n)$ rounds**

Coloring unrooted trees

How can we **color a tree** that is **not rooted**?

Coloring unrooted trees

How can we **color a tree** that is **not rooted**?

- **Electing a root** and **orienting** towards the root costs **$\Theta(D)$ rounds**!
- **Rooted tree** → **out-degree** of each node is **at most 1**

Coloring unrooted trees

How can we **color a tree** that is **not rooted**?

- **Electing a root** and **orienting** towards the root costs **$\Theta(D)$ rounds**!
- **Rooted tree** → **out-degree** of each node is **at most 1**
- **Graphs of max degree Δ** → **out-degree** of each node is **at most Δ** (3^{Δ}-coloring)

Coloring unrooted trees

How can we **color a tree** that is **not rooted**?

- **Electing a root** and **orienting** towards the root costs **$\Theta(D)$ rounds**!
- **Rooted tree** → **out-degree** of each node is **at most 1**
- **Graphs of max degree Δ** → **out-degree** of each node is **at most Δ** (3^{Δ}-coloring)
- **Goal** → **out-degree** of each node is **at most c** (for a constant c)
  - We can use the algorithm from before to obtain $C = 3^c$-coloring

Coloring unrooted trees

How can we **color a tree** that is **not rooted**?

- **Electing a root** and **orienting** towards the root costs **$\Theta(D)$ rounds**!
- **Rooted tree** → **out-degree** of each node is **at most 1**
- **Graphs of max degree Δ** → **out-degree** of each node is **at most Δ** (3^{Δ}-coloring)
- **Goal** → **out-degree** of each node is **at most c** (for a constant c)
  - We can use the algorithm from before to obtain $C = 3^c$-coloring
- **How can we compute such an orientation for a small c?**
  - Let's try **$c = 2$** (this would give a 9-coloring)

Computing an orientation with out-degree 2

Observation 1: Computing an orientation with **out-degree ≤ 2** is trivial for of **degree ≤ 2** (orient arbitrarily)

Computing an orientation with out-degree 2

Observation 1: Computing an orientation with **out-degree ≤ 2** is trivial for of **degree ≤ 2** (orient arbitrarily)

Observation 2: In an n-node tree, at **least *n*/3 nodes** have **degree ≤ 2**

Computing an orientation with out-degree 2

Observation 1: Computing an orientation with out-degree ≤ 2 is trivial for of degree ≤ 2 (orient arbitrarily)

Observation 2: In an n-node tree, at **least *n*/3 nodes** have **degree ≤ 2**

$$\text{number of edges} = n - 1$$

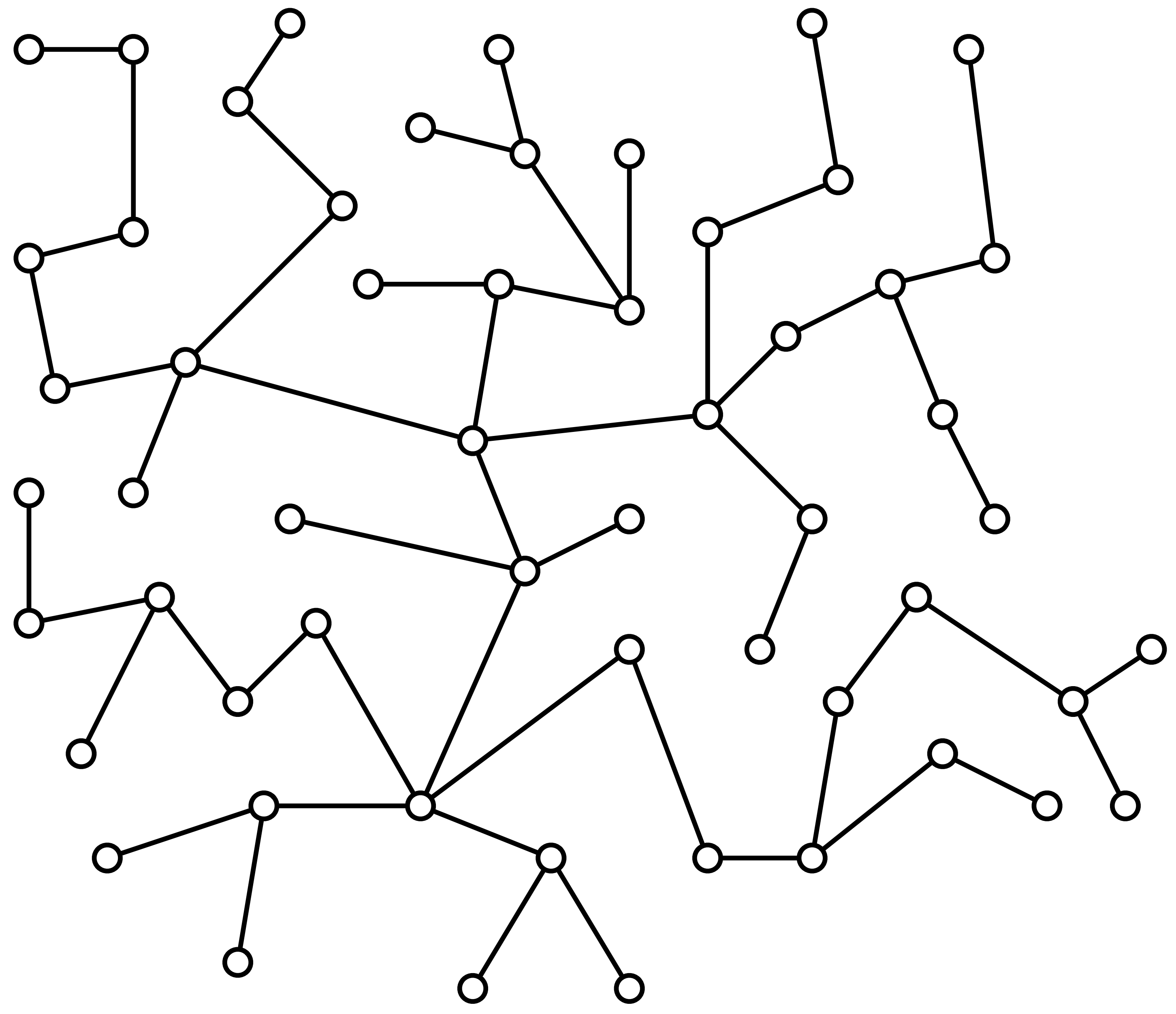
$$\sum_{v \in V} \deg(v) = 2n - 2 < 2n$$

- Assume that *k* nodes have degree ≥ 3

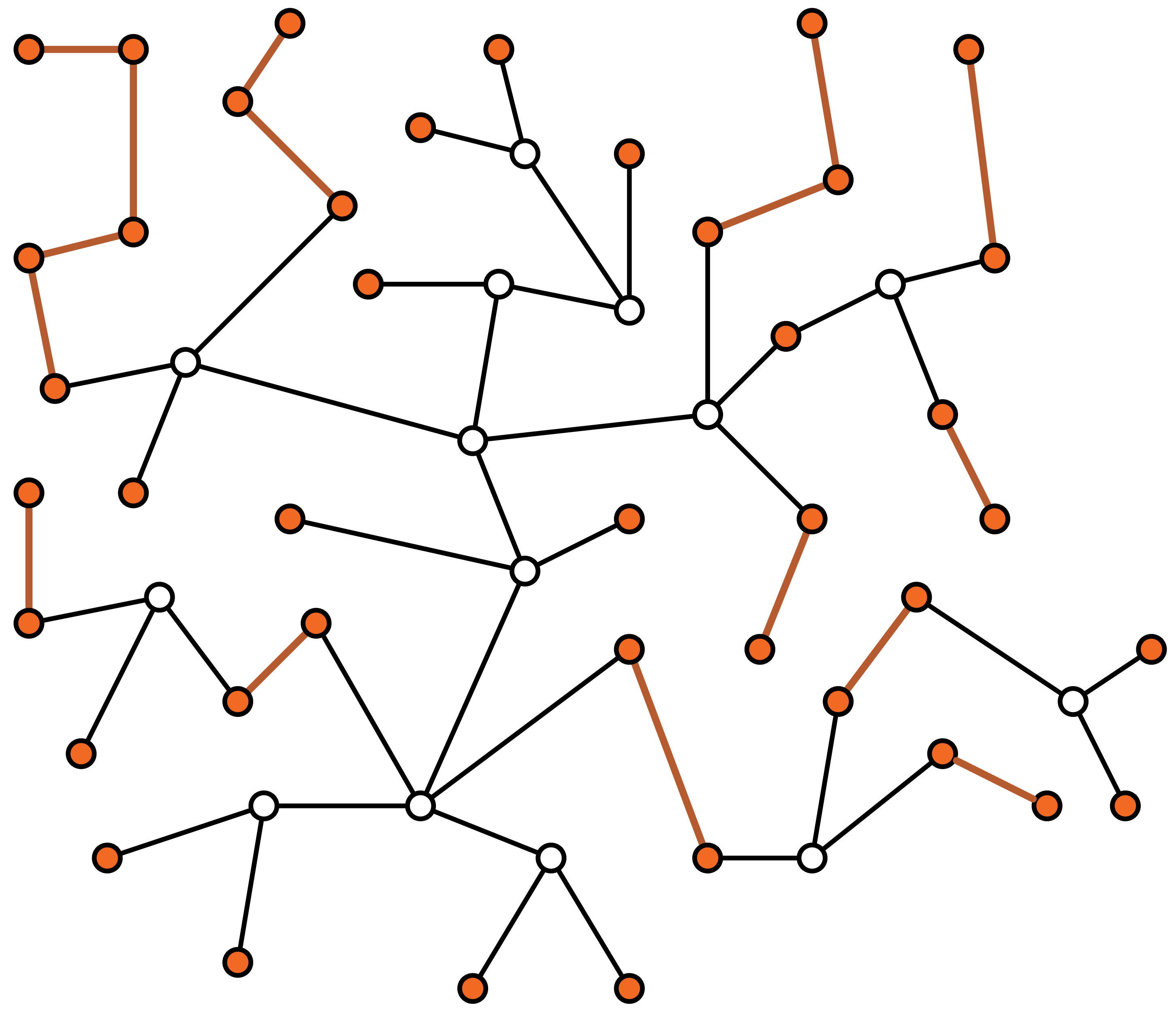
$$\sum_{v \in V} \deg(v) \geq 3k < 2n$$

$$k < \frac{2}{3} n$$

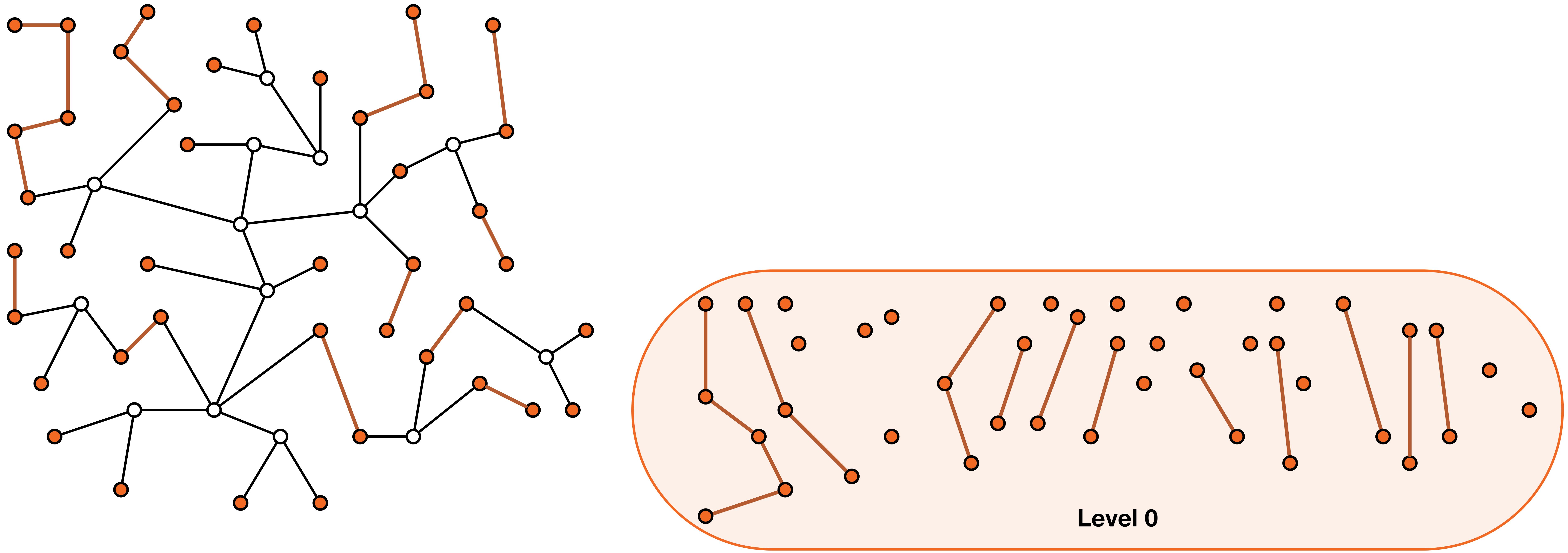
Computing an orientation with out-degree 2



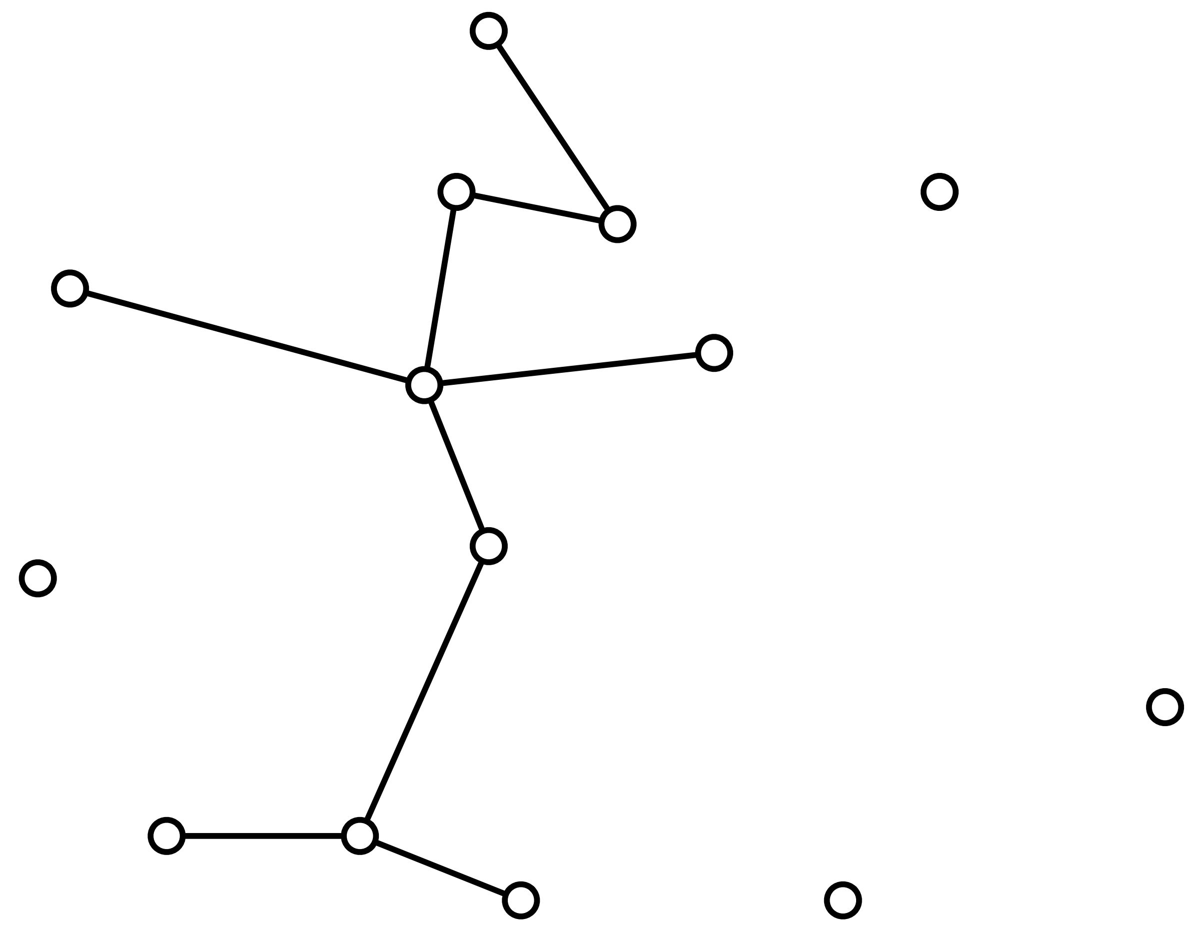
Computing an orientation with out-degree 2

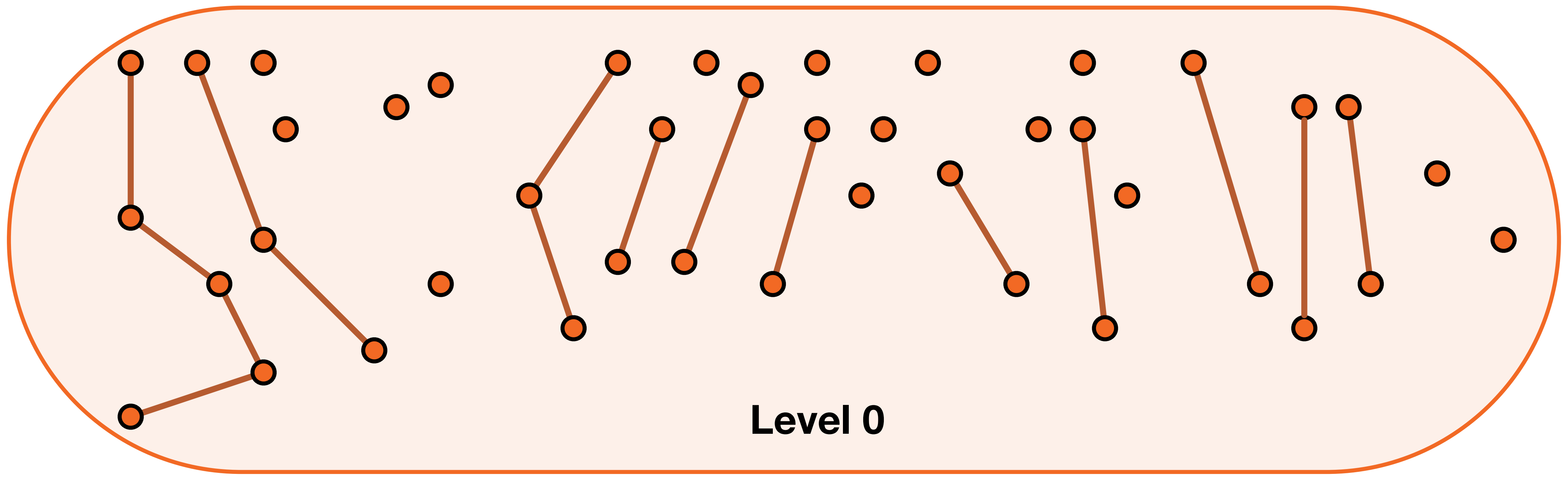


Computing an orientation with out-degree 2

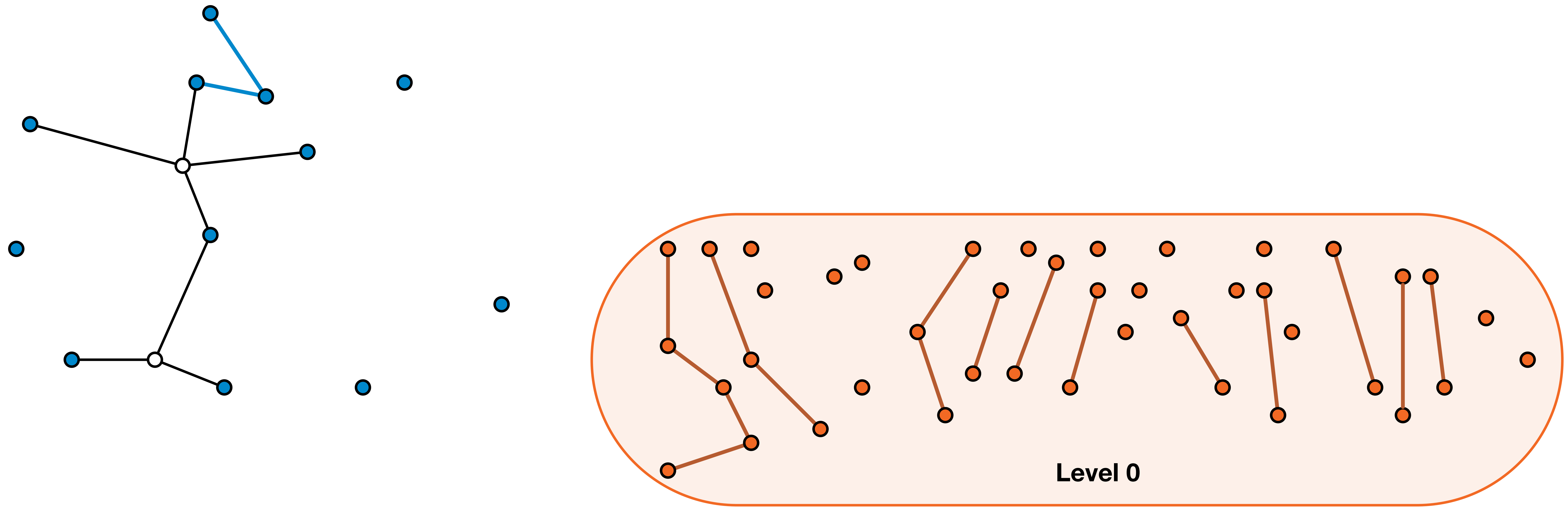


Computing an orientation with out-degree 2

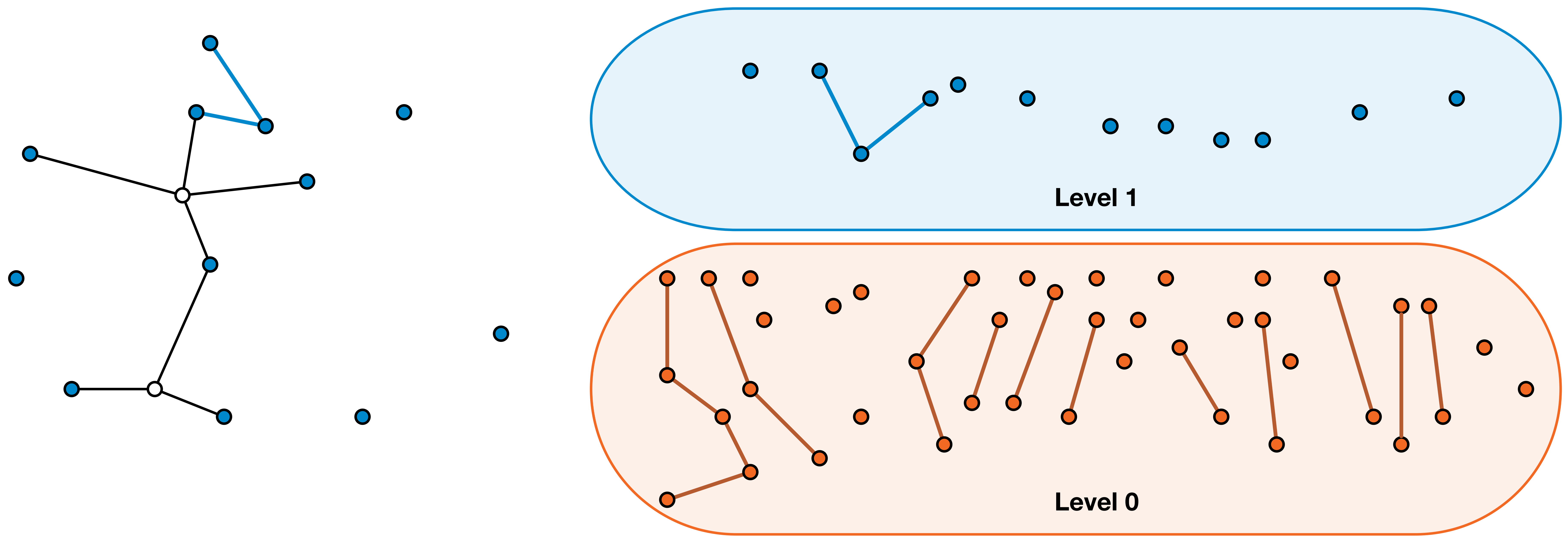




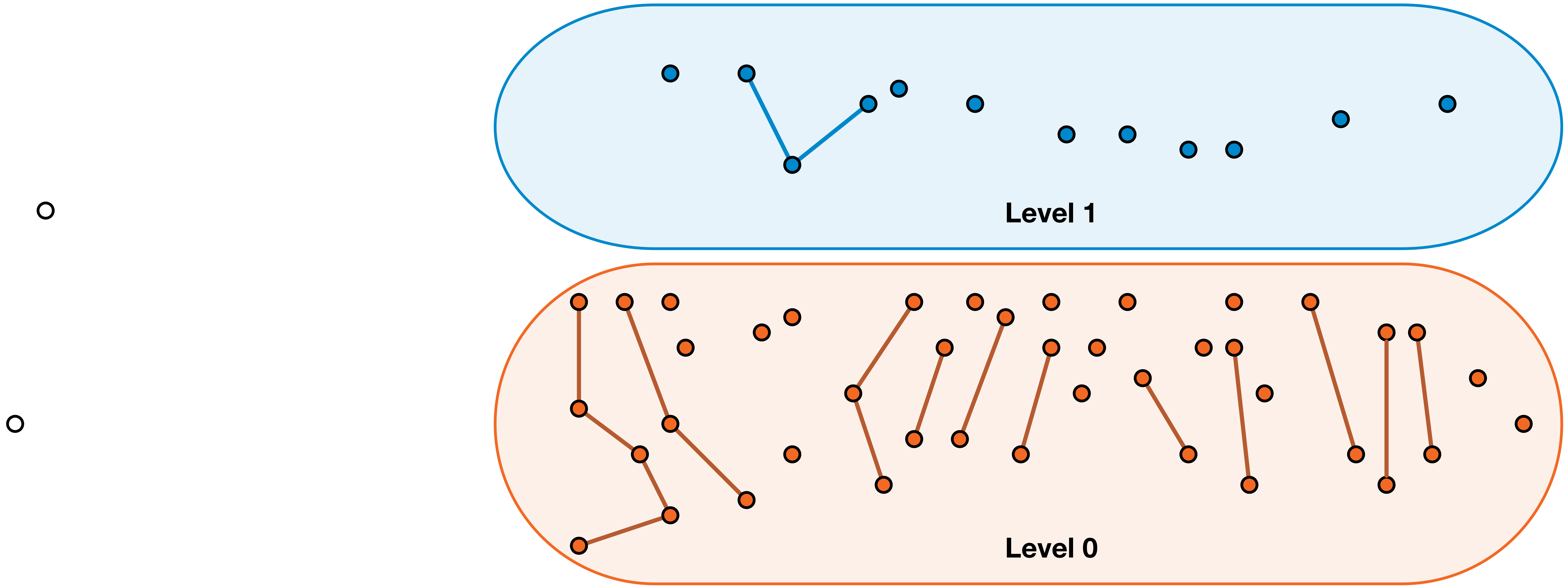
Computing an orientation with out-degree 2



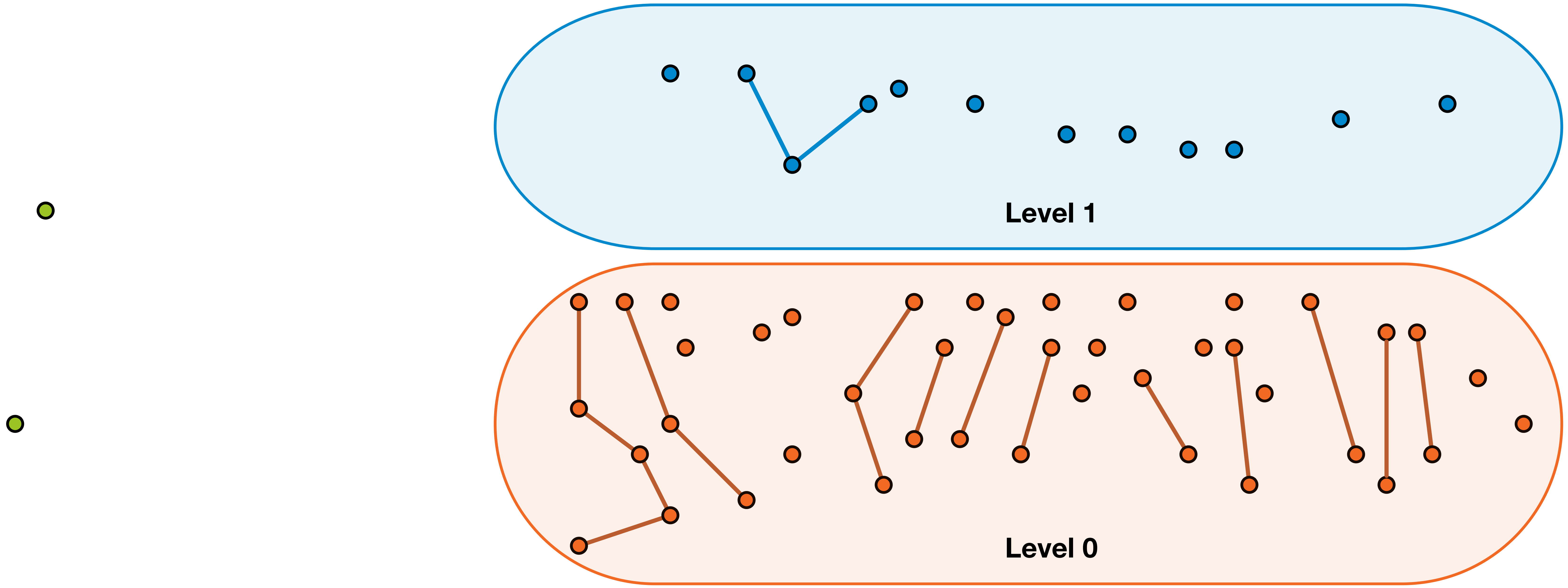
Computing an orientation with out-degree 2



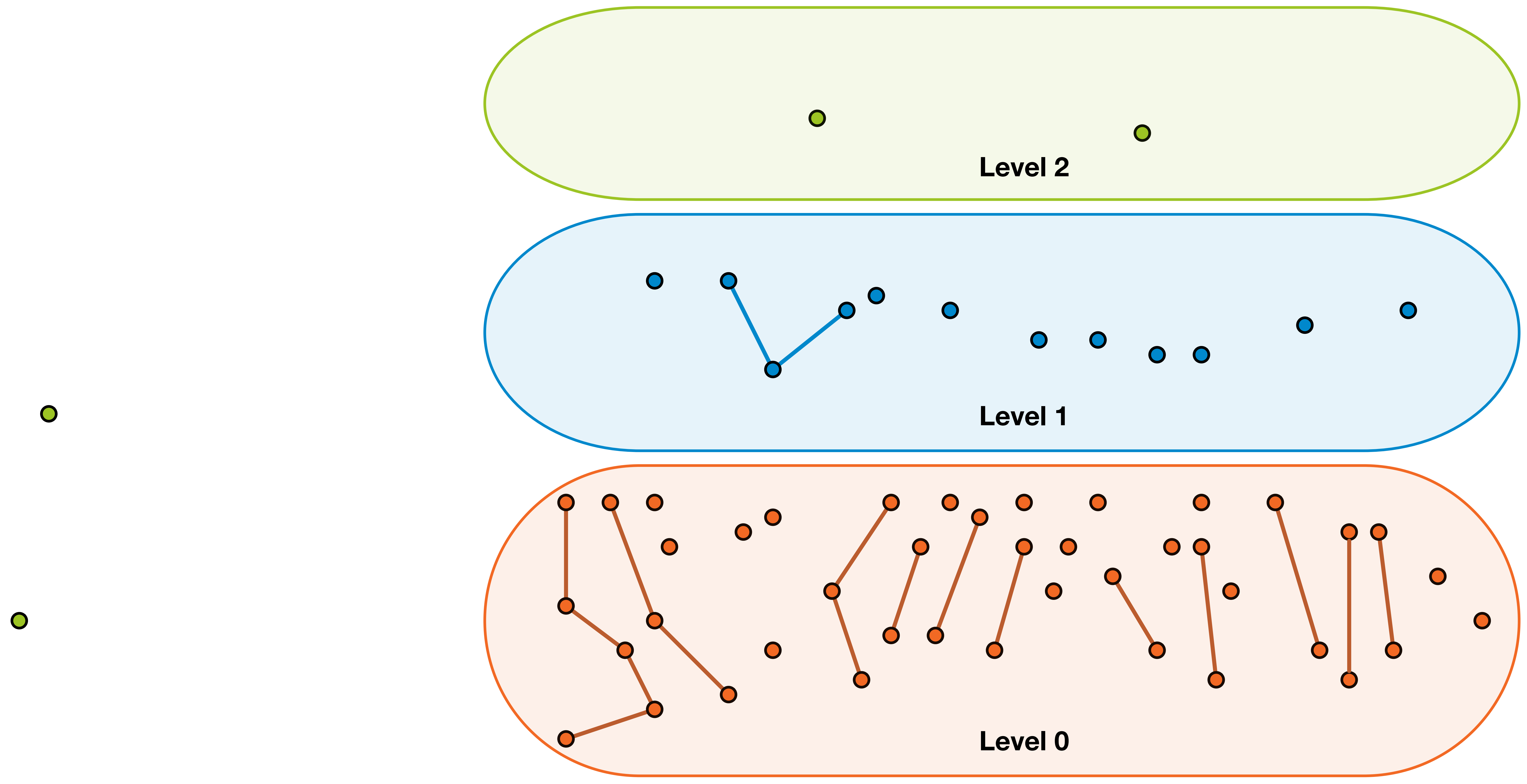
Computing an orientation with out-degree 2



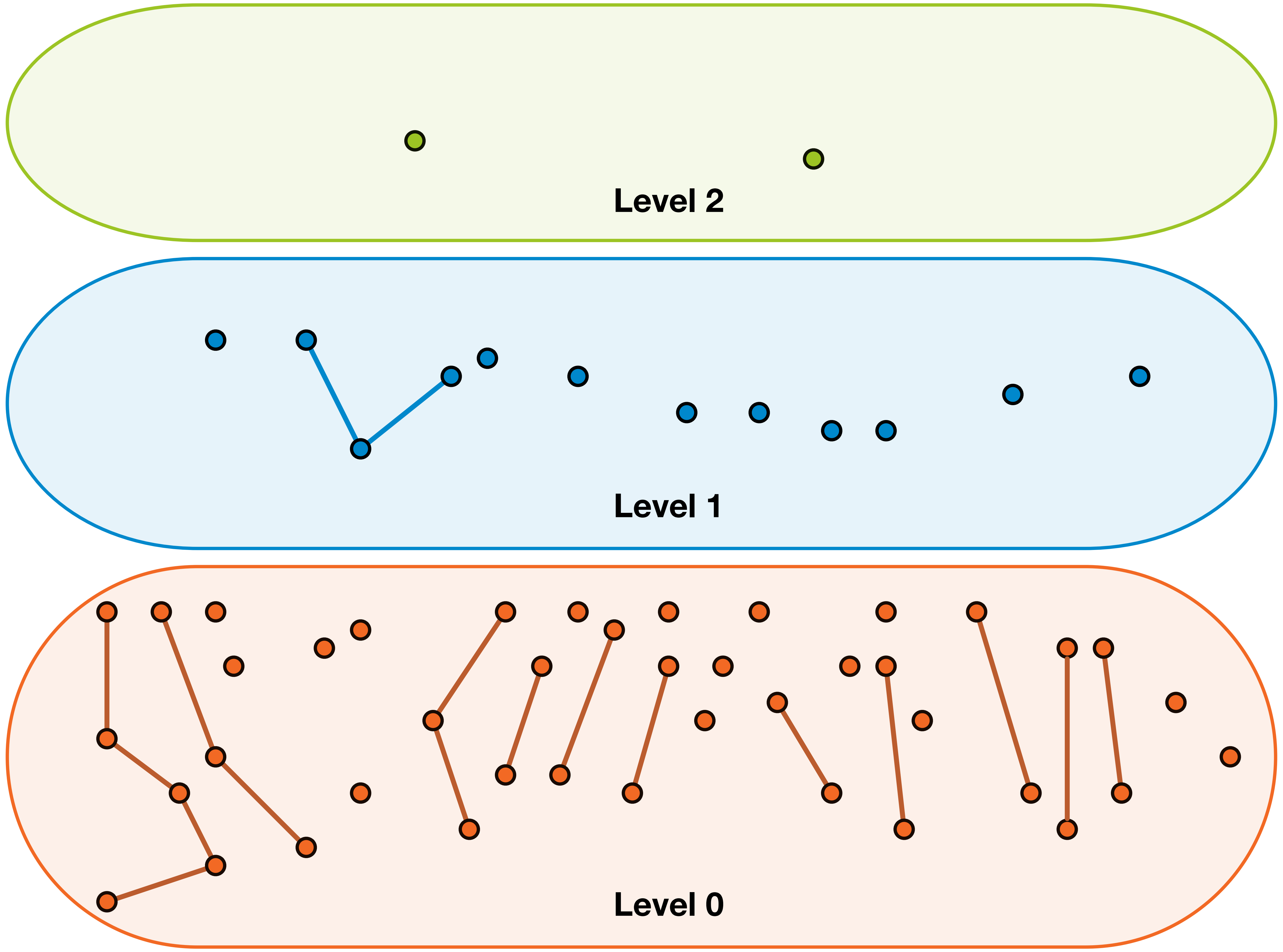
Computing an orientation with out-degree 2



Computing an orientation with out-degree 2

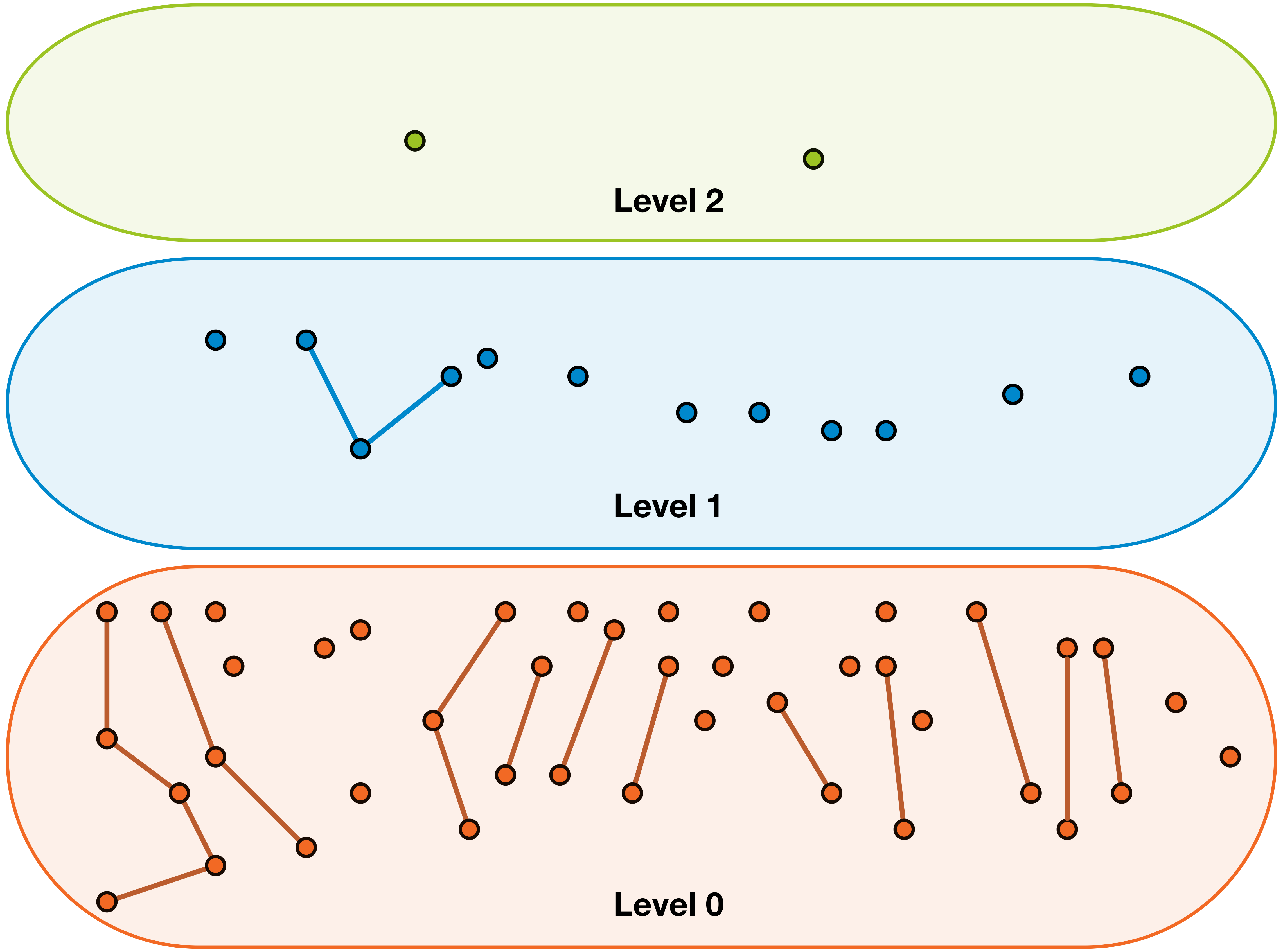


Computing an orientation with out-degree 2



Computing an orientation with out-degree 2

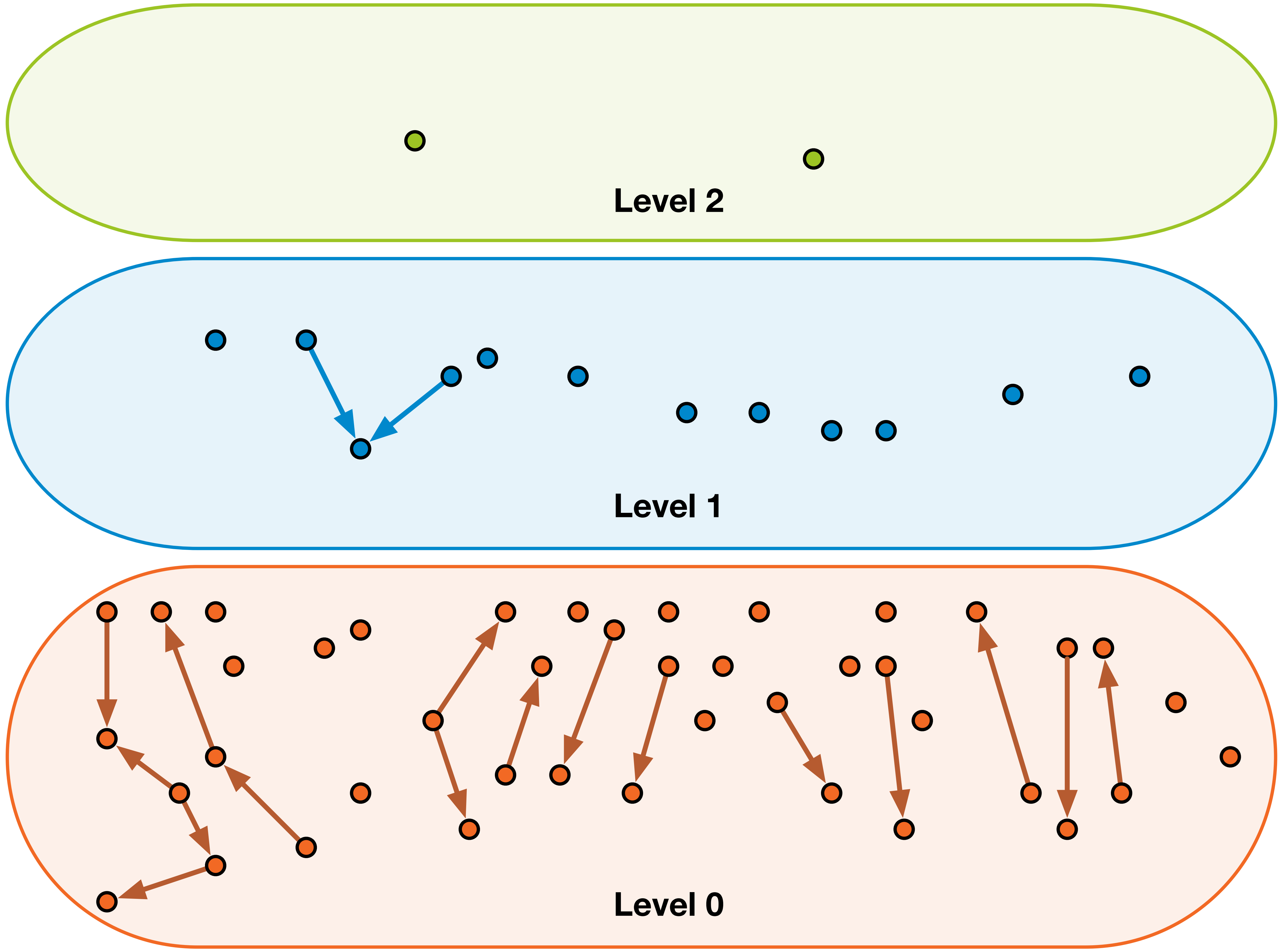
How to orient edges?



Computing an orientation with out-degree 2

How to orient edges?

Edges **inside** each level:
orient **arbitrarily**

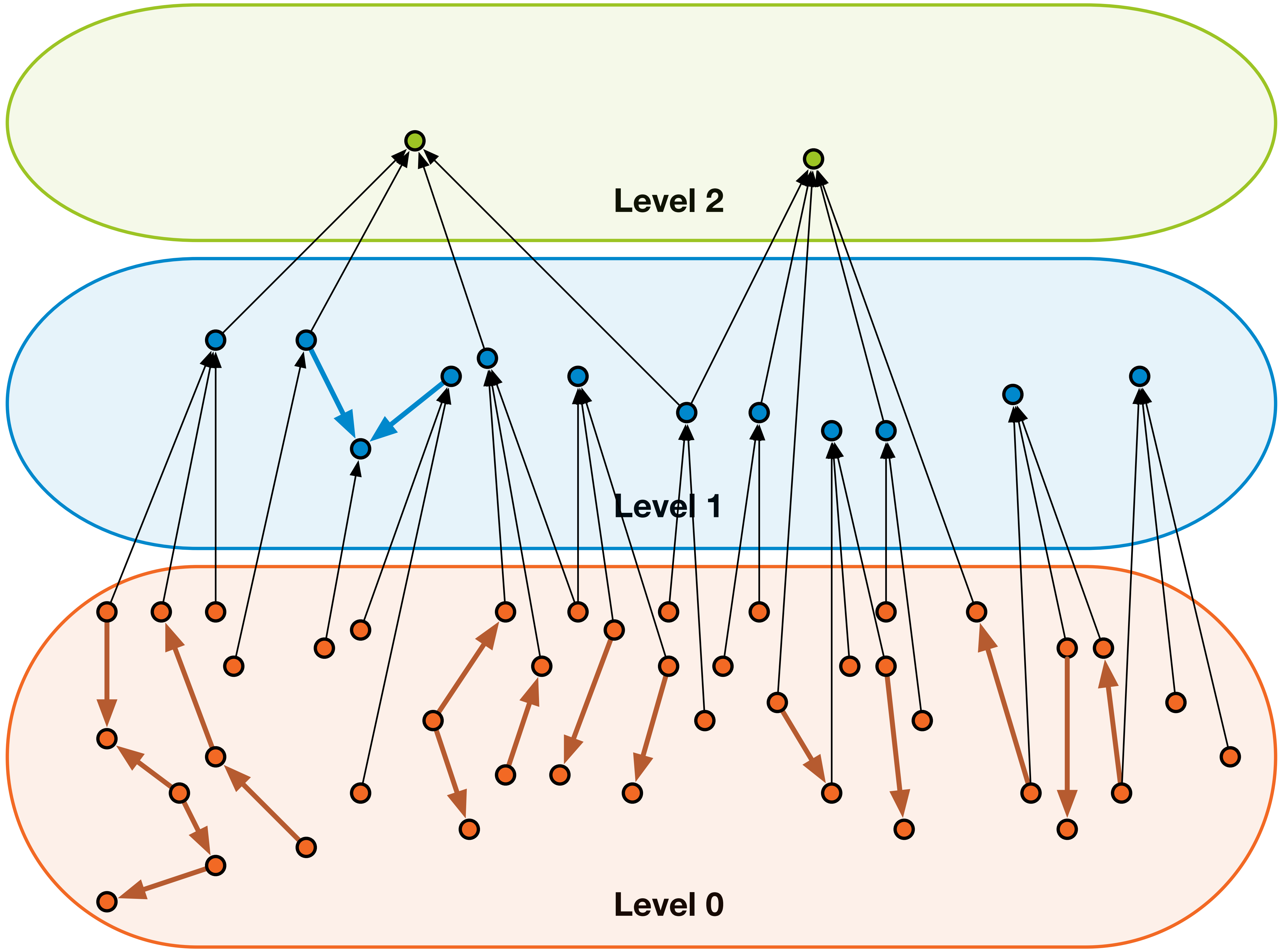


Computing an orientation with out-degree 2

How to orient edges?

Edges **inside** each level: orient **arbitrarily**

Edges **between** levels: orient **from smallest to largest**



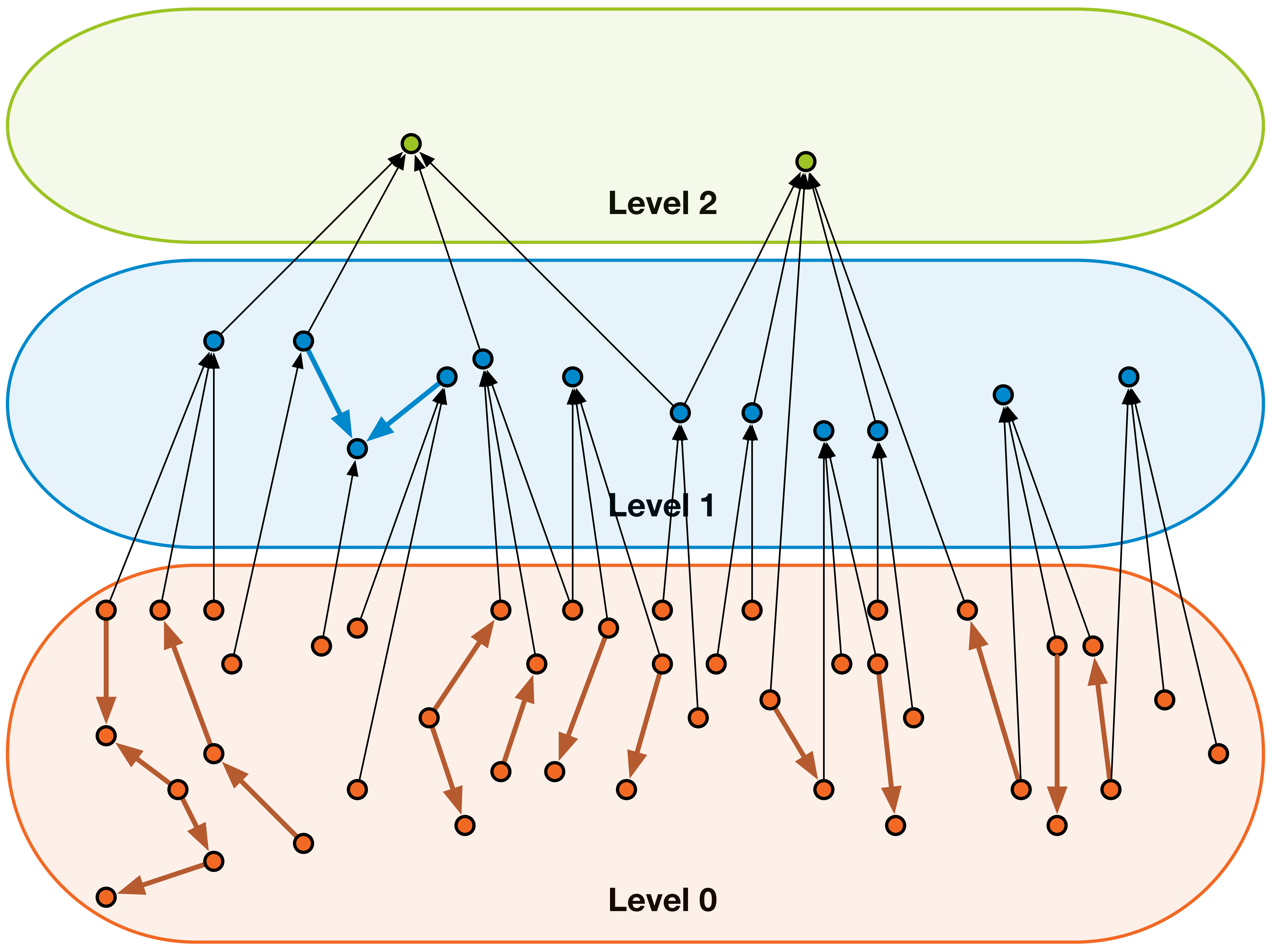
Computing an orientation with out-degree 2

How to orient edges?

Edges **inside** each level: orient **arbitrarily**

Edges **between** levels: orient **from smallest to largest**

Nodes in **Level 0** have degree ≤ 2

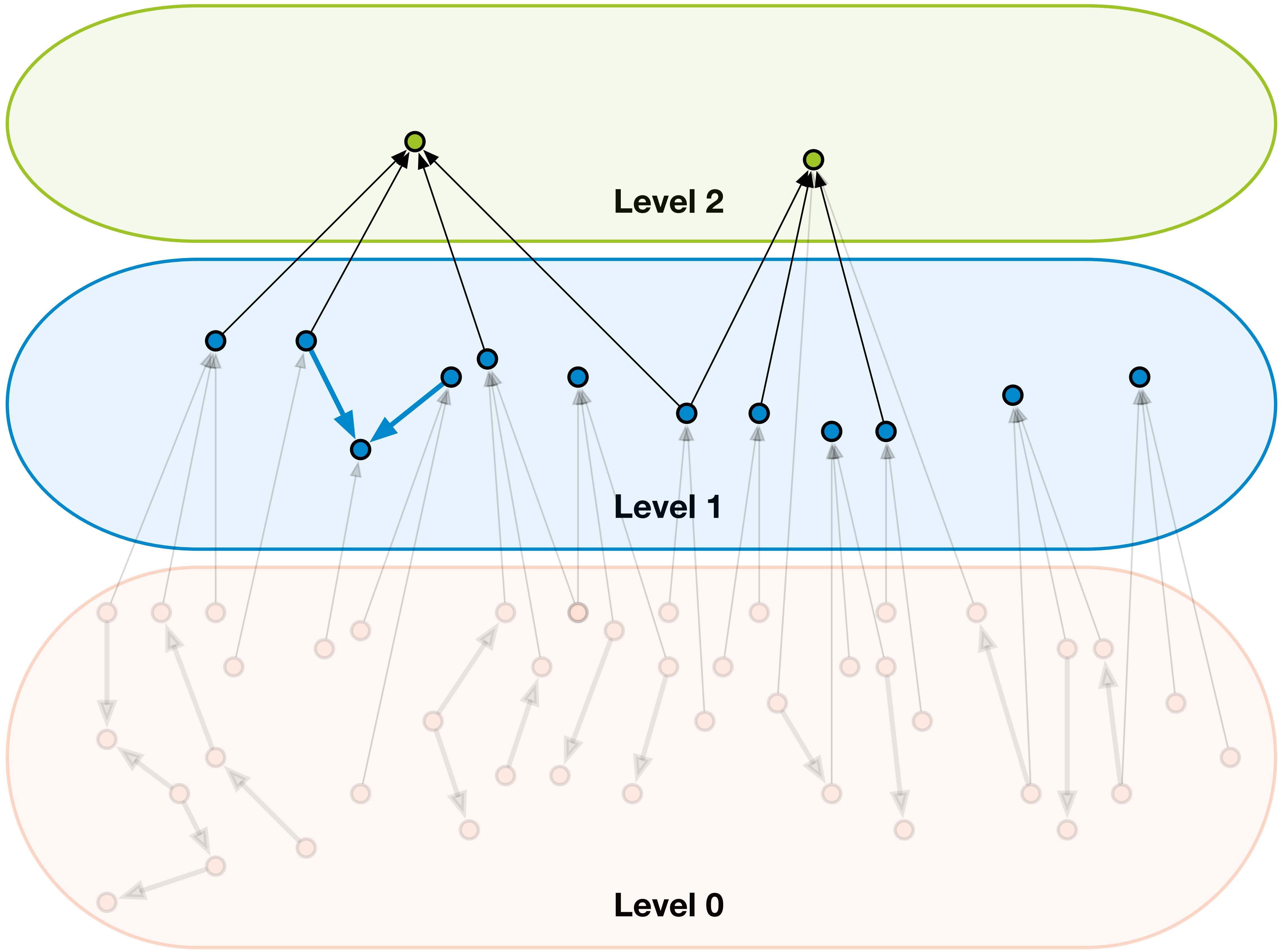


Computing an orientation with out-degree 2

How to orient edges?

Edges **inside** each level: orient **arbitrarily**

Edges **between** levels: orient **from smallest to largest**



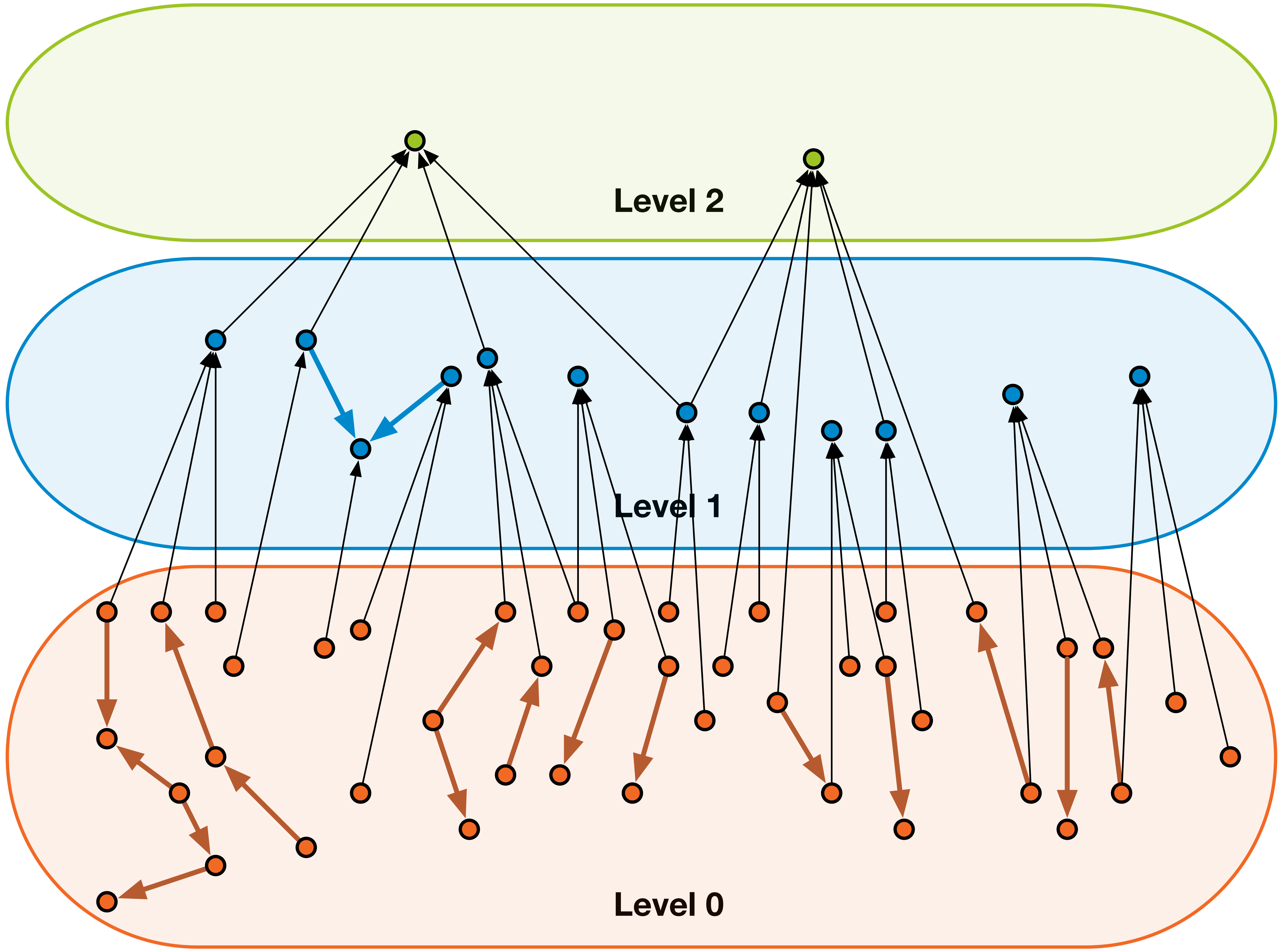
Computing an orientation with out-degree 2

How to orient edges?

Edges **inside** each level: orient **arbitrarily**

Edges **between** levels: orient **from smallest to largest**

Nodes in **Level *i*** have **degree ≤ 2** in the graph induced by nodes in **Level j ≥ i**



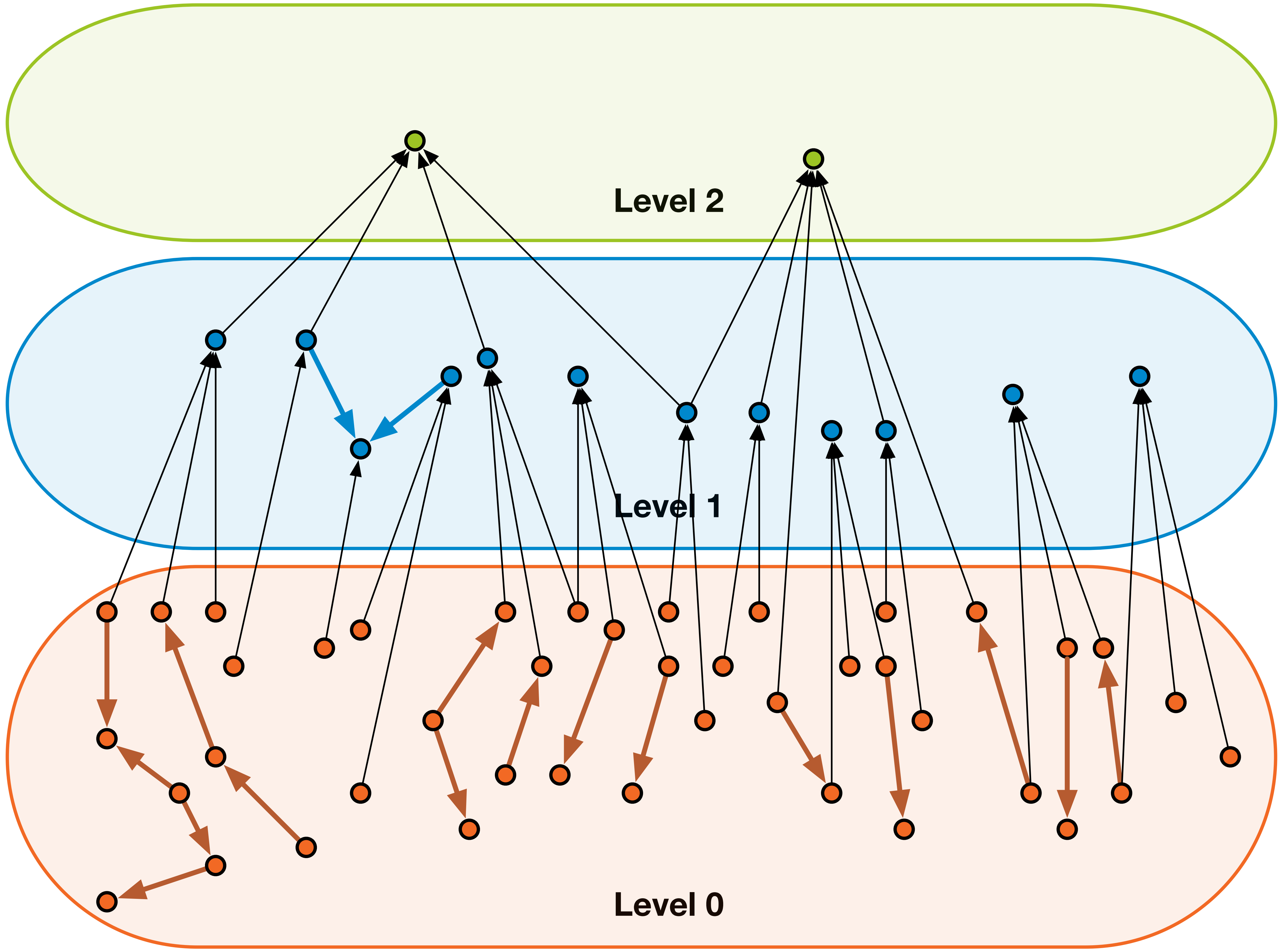
Computing an orientation with out-degree 2

How to orient edges?

Edges **inside** each level: orient **arbitrarily**

Edges **between** levels: orient **from smallest to largest**

How many levels?



Computing an orientation with out-degree 2

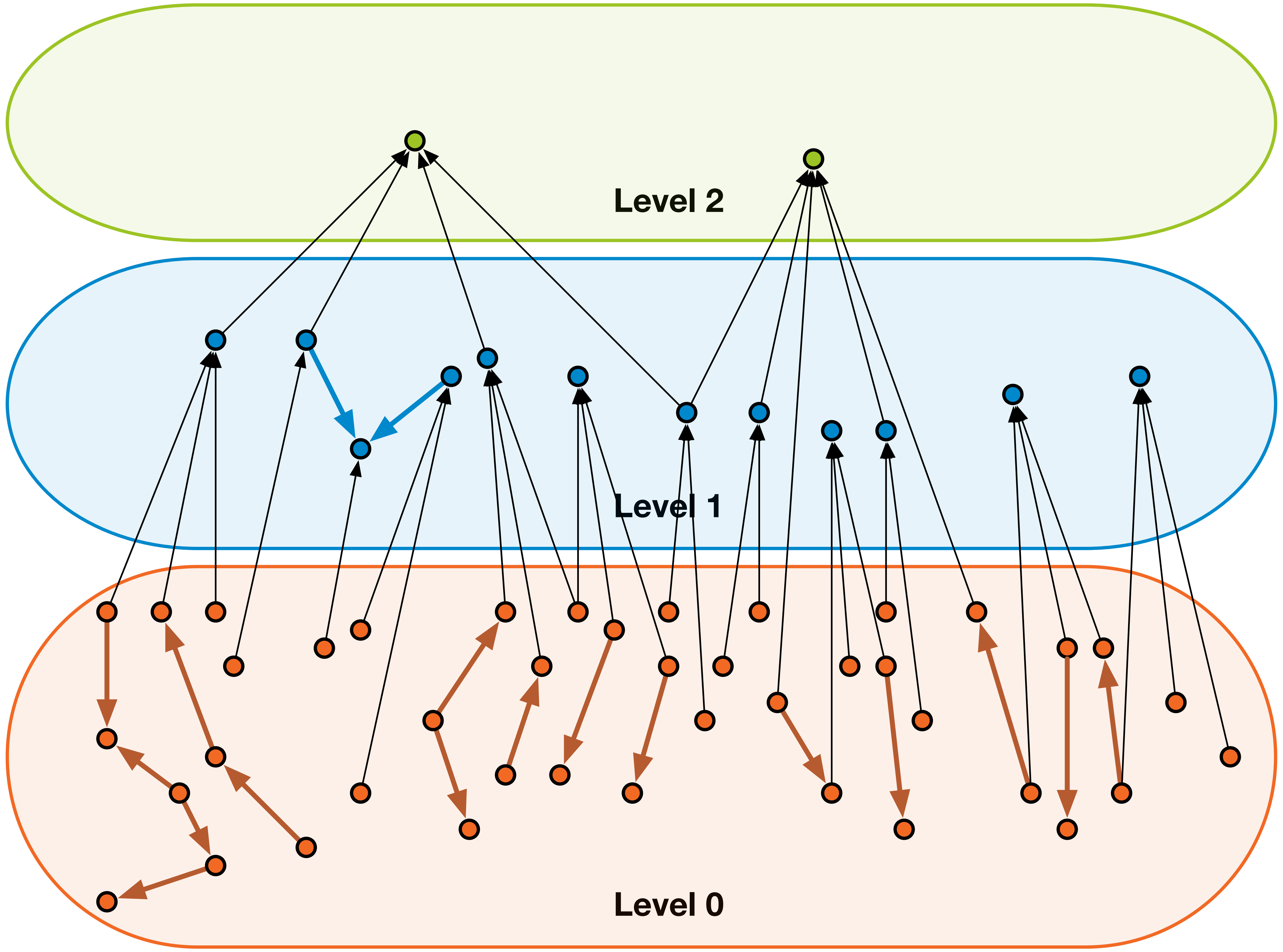
How to orient edges?

Edges **inside** each level: orient **arbitrarily**

Edges **between** levels: orient **from smallest to largest**

How many levels?

Nr. of nodes in Level ≥ *i*: **at most** $n \cdot (2/3)^i$



Computing an orientation with out-degree 2

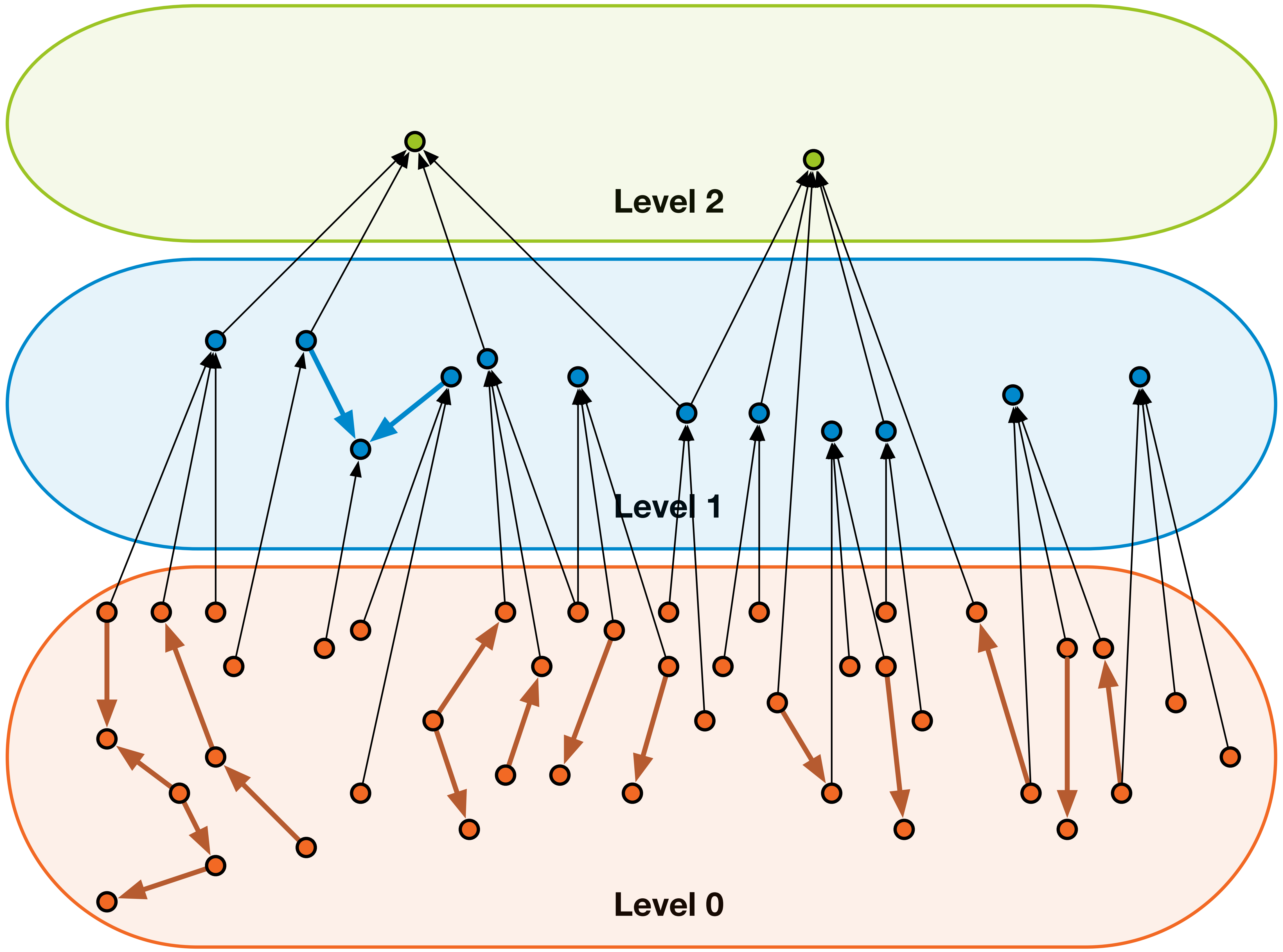
How to orient edges?

Edges **inside** each level: orient **arbitrarily**

Edges **between** levels: orient **from smallest to largest**

How many levels?

Nr. of nodes in Level ≥ *i*: **at most $n \cdot (2/3)^i$**



Each time we process a constant fraction of the nodes: **$O(\log n)$ levels**

9-coloring unrooted trees

1. Compute an **orientation** with **out-degree ≤ 2** in ***O*(log *n*) rounds**
    - This creates **two directed forests** (it's not a pseudoforest since in a tree there are no cycles)
2. **Color** each **forest** with **3 colors** in ***O*(log* *n*) rounds**
    - Every **node *v*** then has **two colors**: $c_{v,1}$ for forest 1 and $c_{v,2}$ for forest 2
    - The **total** number of **colors** used is $3^{\text{out-degree}} \leq 3^2 = 9$
    - For every edge {u, v}, we have $\boldsymbol{c_{u,1} \neq c_{v,1}}$ or $\boldsymbol{c_{u,2} \neq c_{v,2}}$

Remark: The algorithm also works for (undirected) pseudoforests

Summary

Coloring trees

- Trees can be colored with **2 colors**, this however requires time $\Omega(D)$
- **Rooted** trees can be **3-colored** in time $O(\log^* n)$
- **Unrooted** trees can be **9-colored** in time $O(\log n)$ *(it is possible to obtain 3 colors!)*

Coloring general graphs with maximum degree Δ

- **3^Δ-coloring** can be done in time $O(\log^* n)$
- **$(\Delta + 1)$-coloring** can be done in time $O(3^\Delta + \log^* n)$
  - If $\Delta = O(1)$, this is $O(\log^* n)$
  - This algorithm can be improved significantly: the **current best runtime** is roughly $O(\sqrt{\Delta} + \log^* n)$

Outlook

- **Next lecture**: **randomized algorithms** for $(\Delta + 1)$-*coloring* and *MIS* in general graphs
- **Later lecture**: we will see that, for **deterministic** algorithms, some **bounds** from today's lecture **are tight**