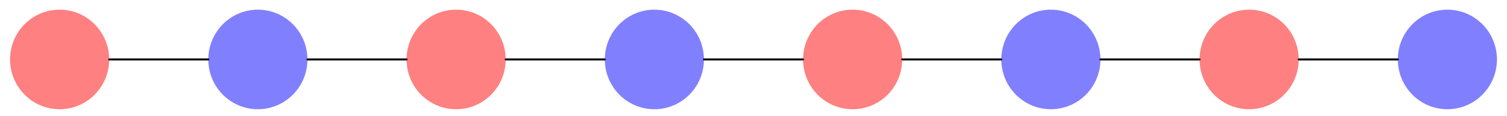
# Lower Bounds

**Dennis Olivetti**

University of Freiburg, Germany

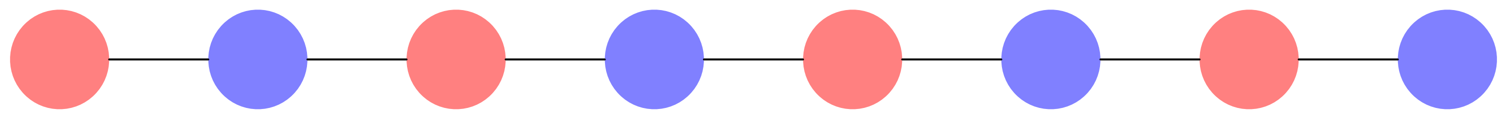
# Lower bounds

- **2** coloring requires **$\Omega(n)$** rounds



# Lower bounds

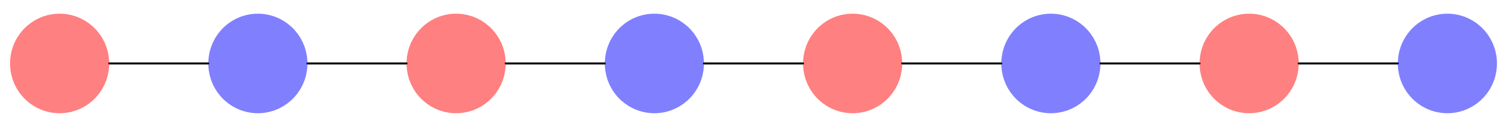
- **2** coloring requires **$\Omega(n)$** rounds



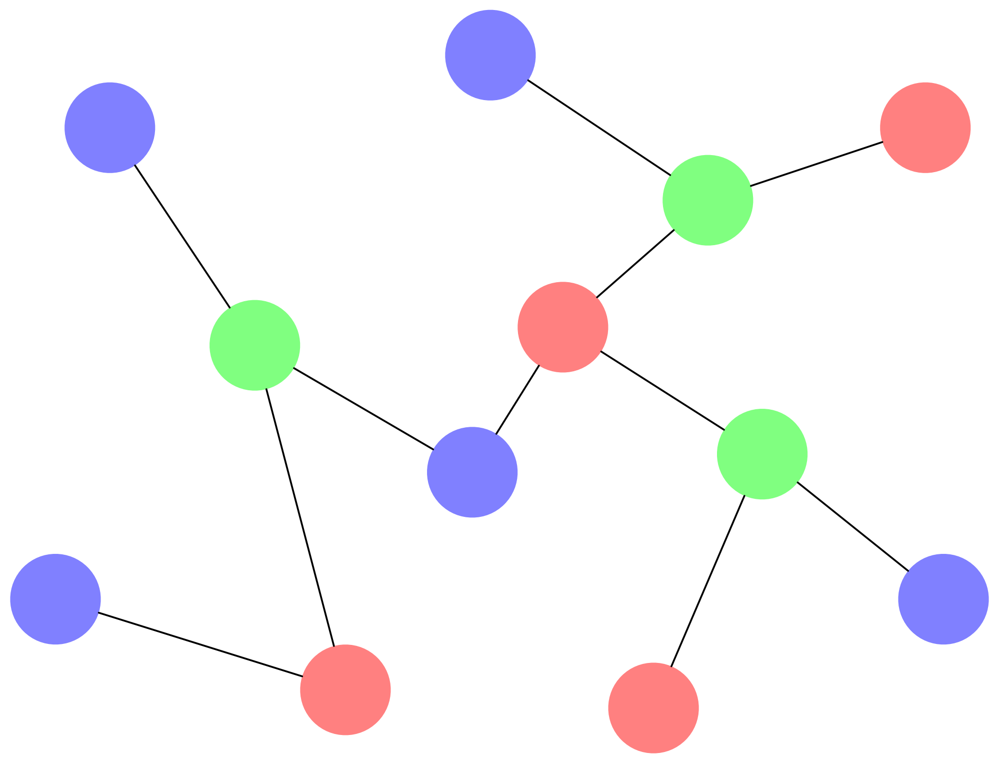
- **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **$\Delta$** requires **$\Omega(\log_\Delta n)$** rounds

# Lower bounds

- **2** coloring requires **$\Omega(n)$** rounds

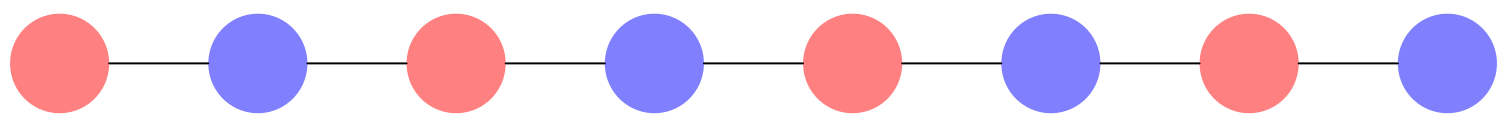


- **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **$\Delta$** requires **$\Omega(\log_\Delta n)$** rounds

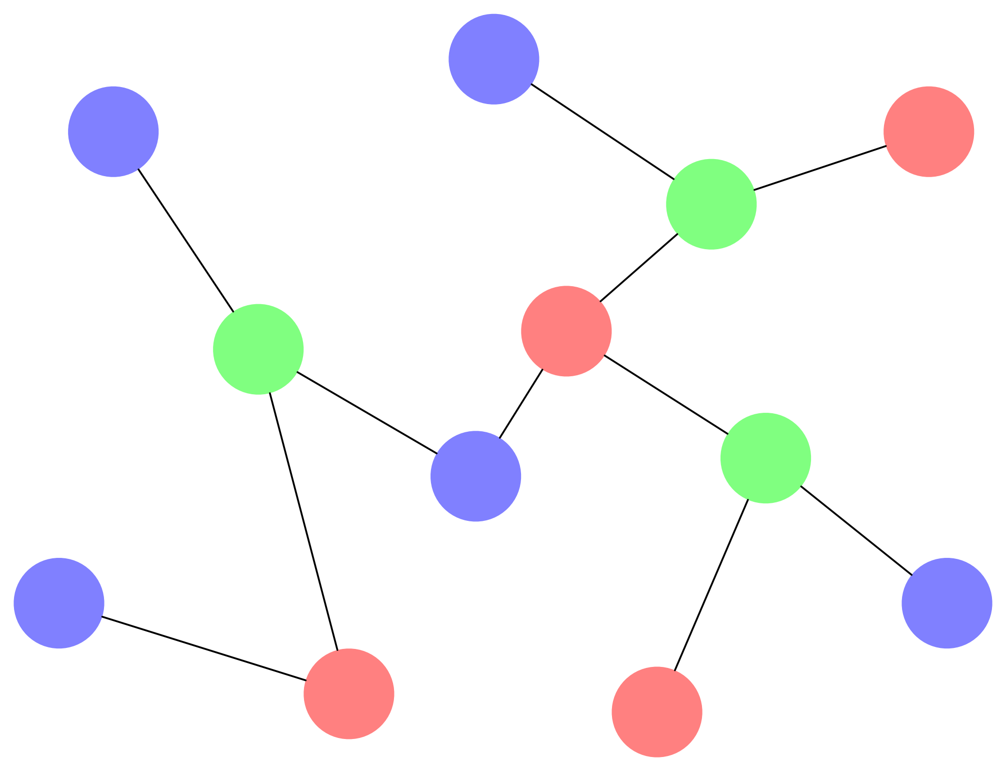


# Lower bounds

- **2** coloring requires **$\Omega(n)$** rounds



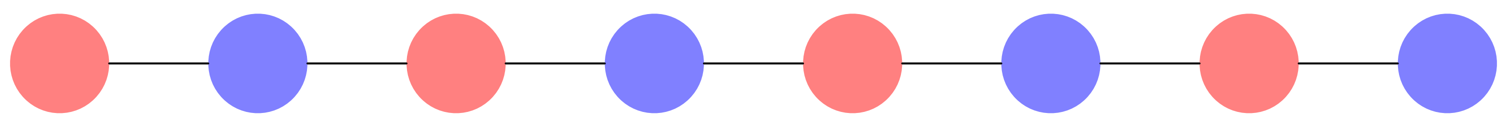
- **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **$\Delta$** requires **$\Omega(\log_{\Delta} n)$** rounds



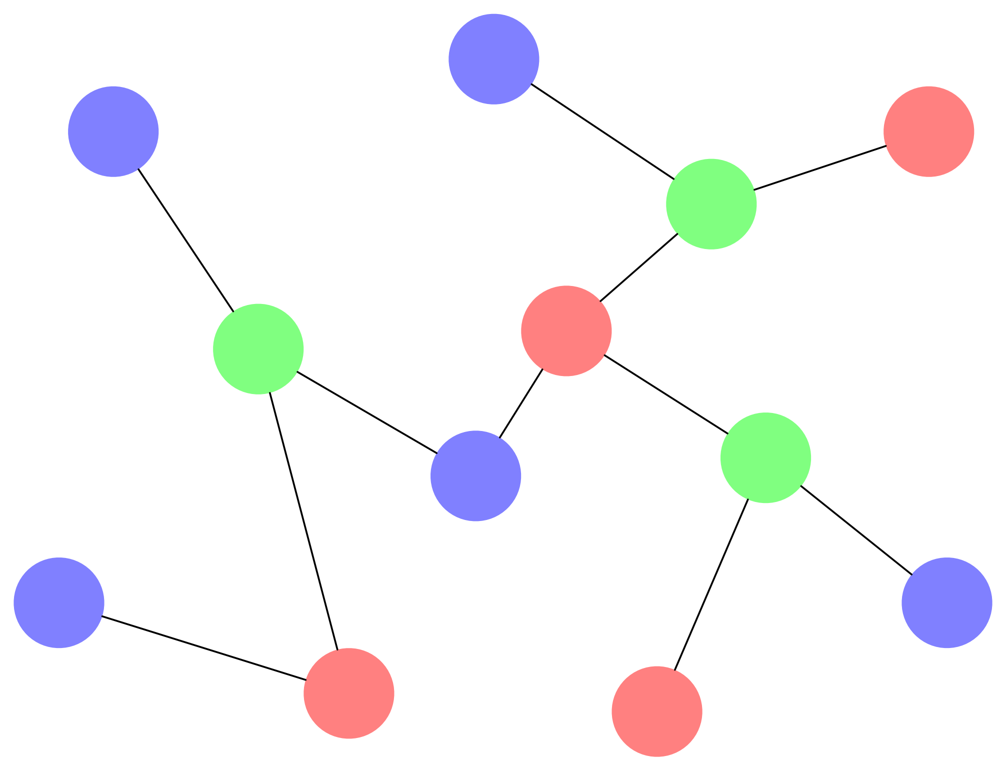
- **3** coloring paths or cycles requires **$\Omega(\log^* n)$** rounds

# Lower bounds

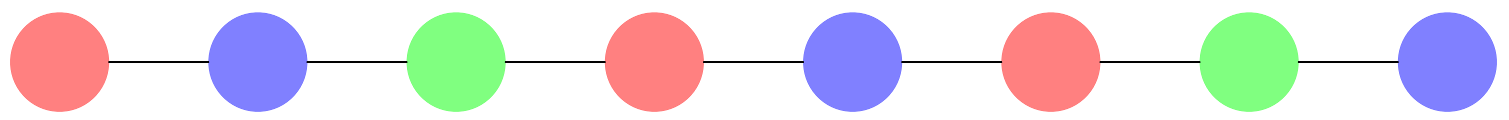
- **2** coloring requires **$\Omega(n)$** rounds



- **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **$\Delta$** requires **$\Omega(\log_\Delta n)$** rounds



- **3** coloring paths or cycles requires **$\Omega(\log^* n)$** rounds



# Locality

# Locality

- If the size of the messages and the local computation is unbounded, all synchronous T-round algorithms have a ***normal form***:

# Locality

- If the size of the messages and the local computation is unbounded, all synchronous T-round algorithms have a ***normal form***:
  - Gather the radius-T view

# Locality

- If the size of the messages and the local computation is unbounded, all synchronous T-round algorithms have a ***normal form***:
  - Gather the radius-T view
  - Perform some local computation

# Locality

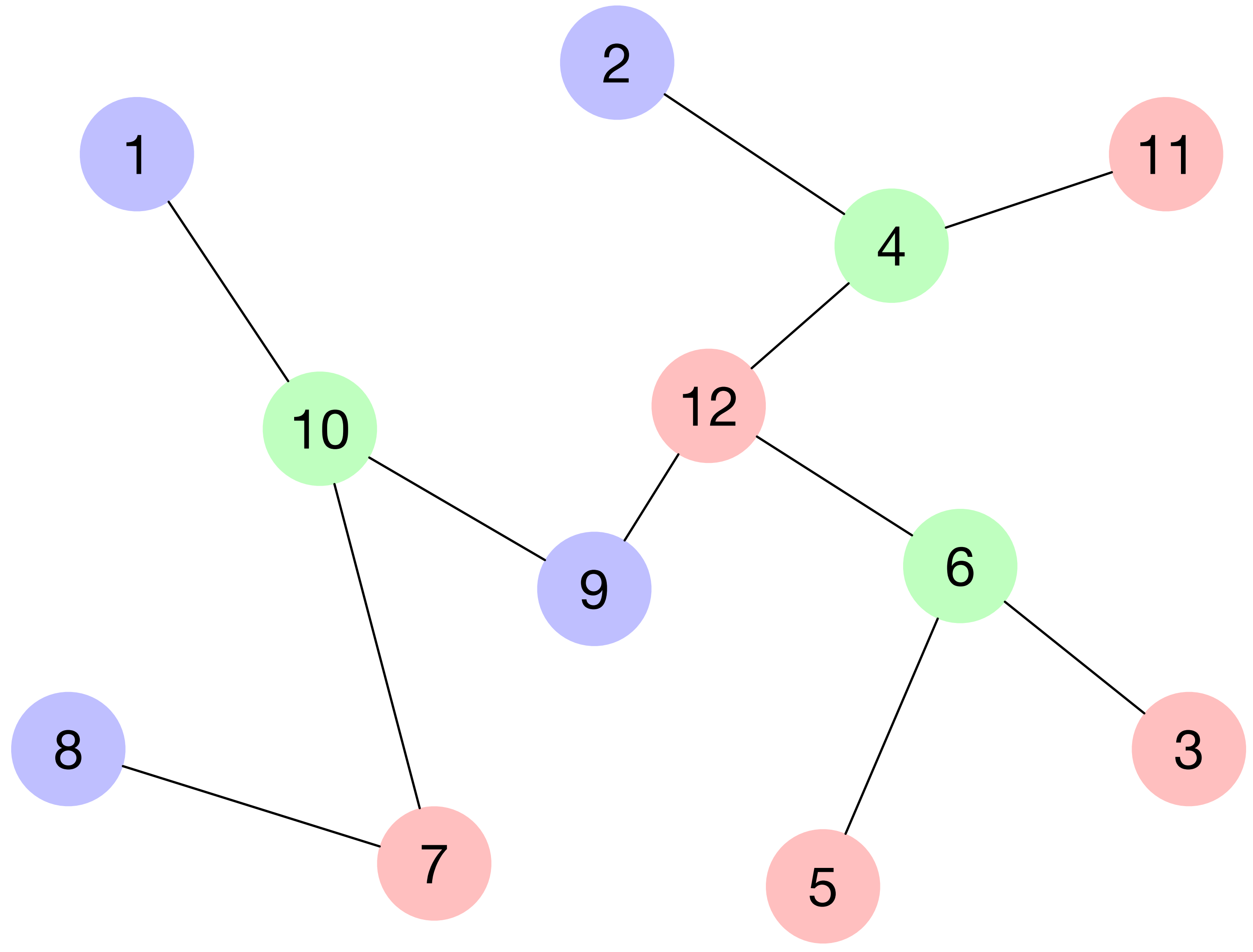
- If the size of the messages and the local computation is unbounded, all synchronous T-round algorithms have a ***normal form***:
  - Gather the radius-T view
  - Perform some local computation
  - Output a result

# Locality (Example)

- A 0-round algorithm is just a mapping from input to output

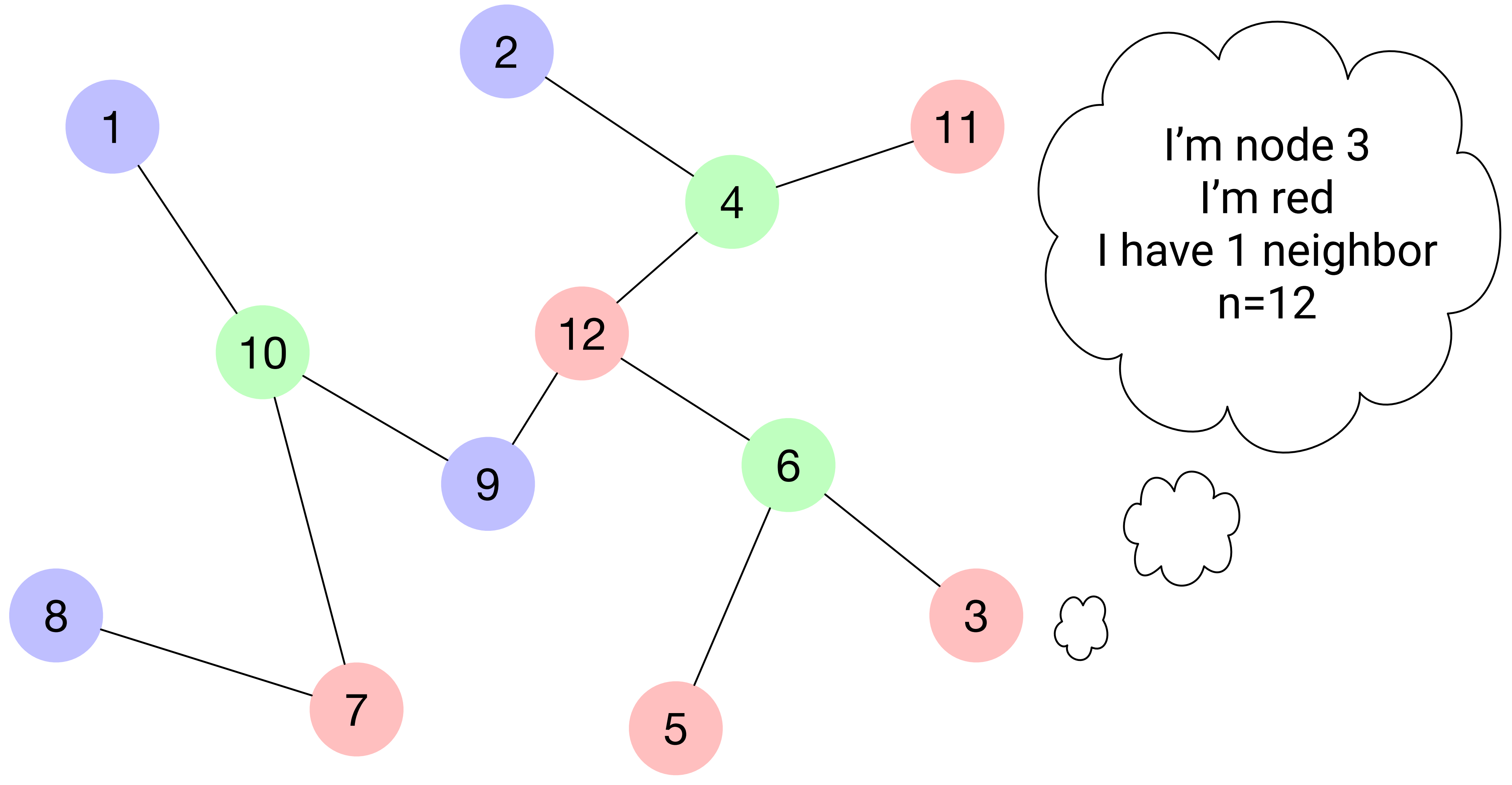
# Locality (Example)

- A 0-round algorithm is just a mapping from input to output

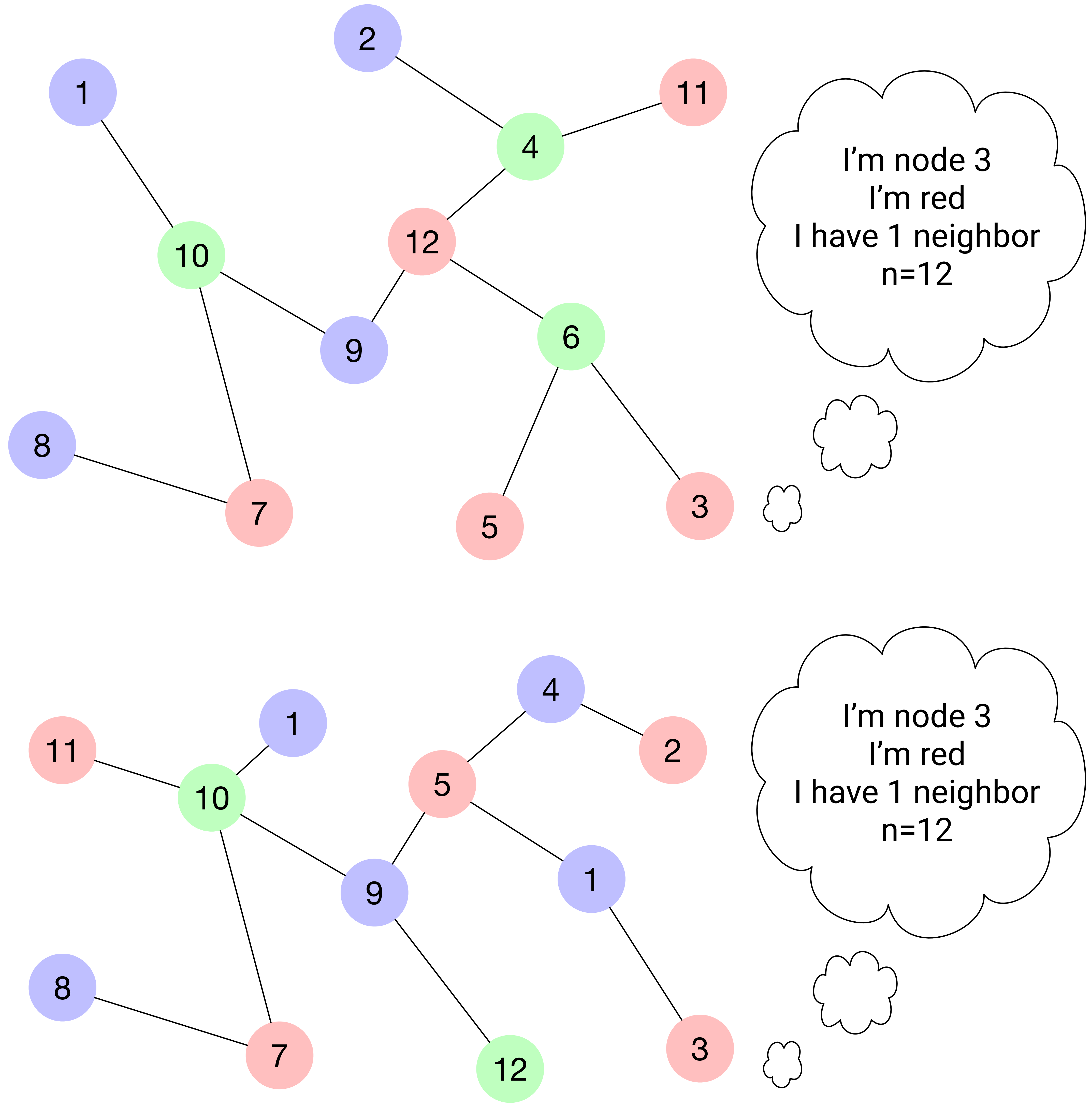


# Locality (Example)

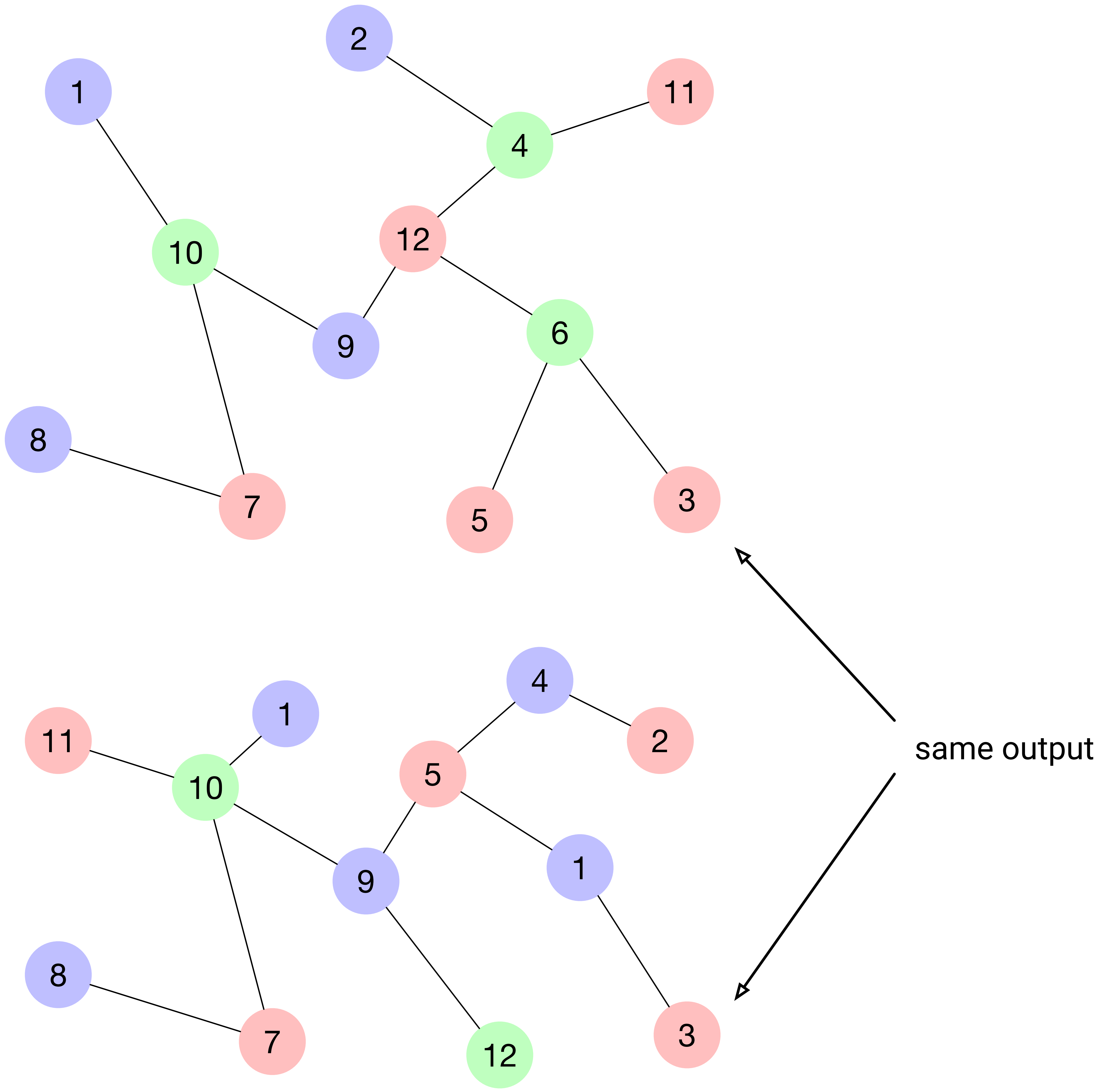
- A 0-round algorithm is just a mapping from input to output



# Locality (Example)

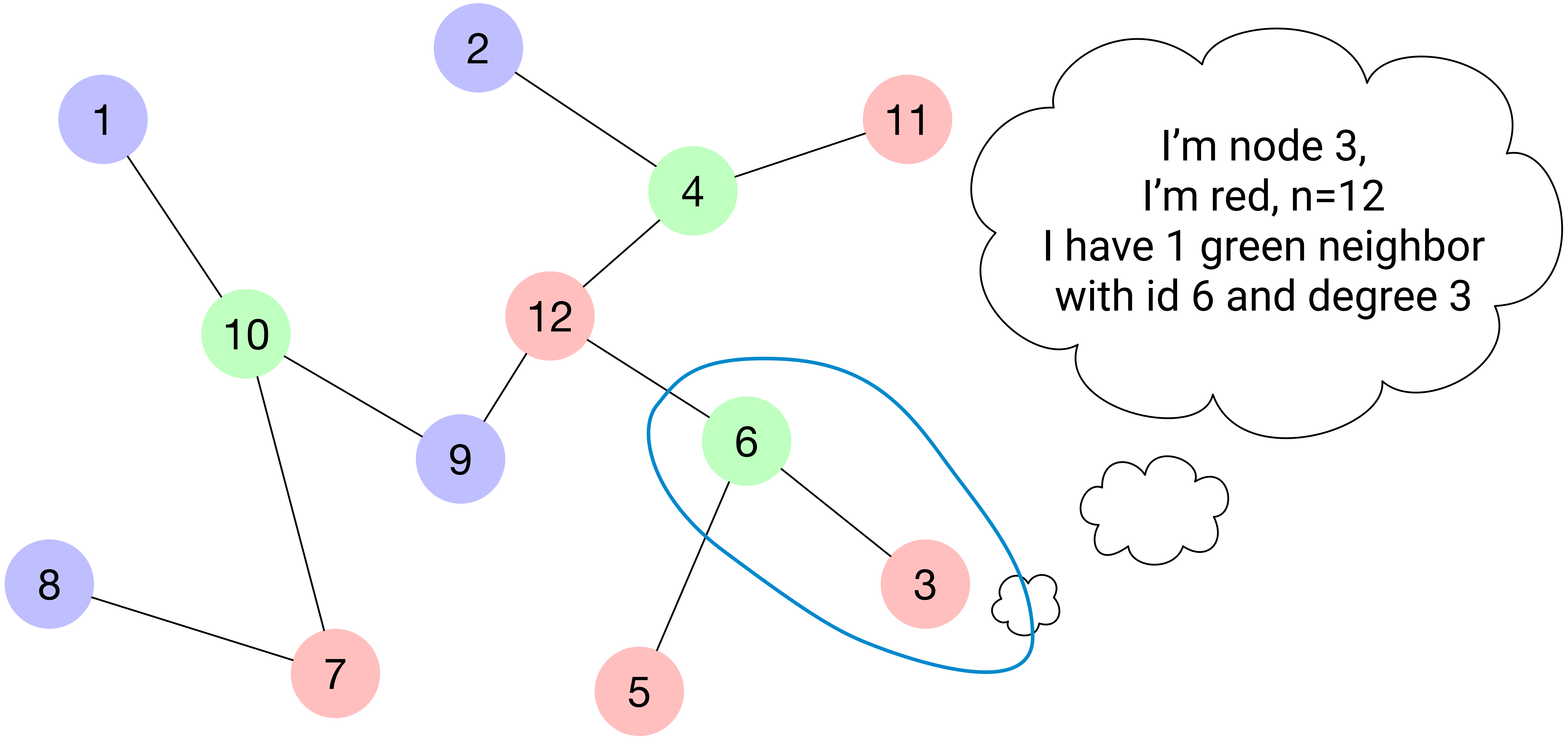


# Locality (Example)



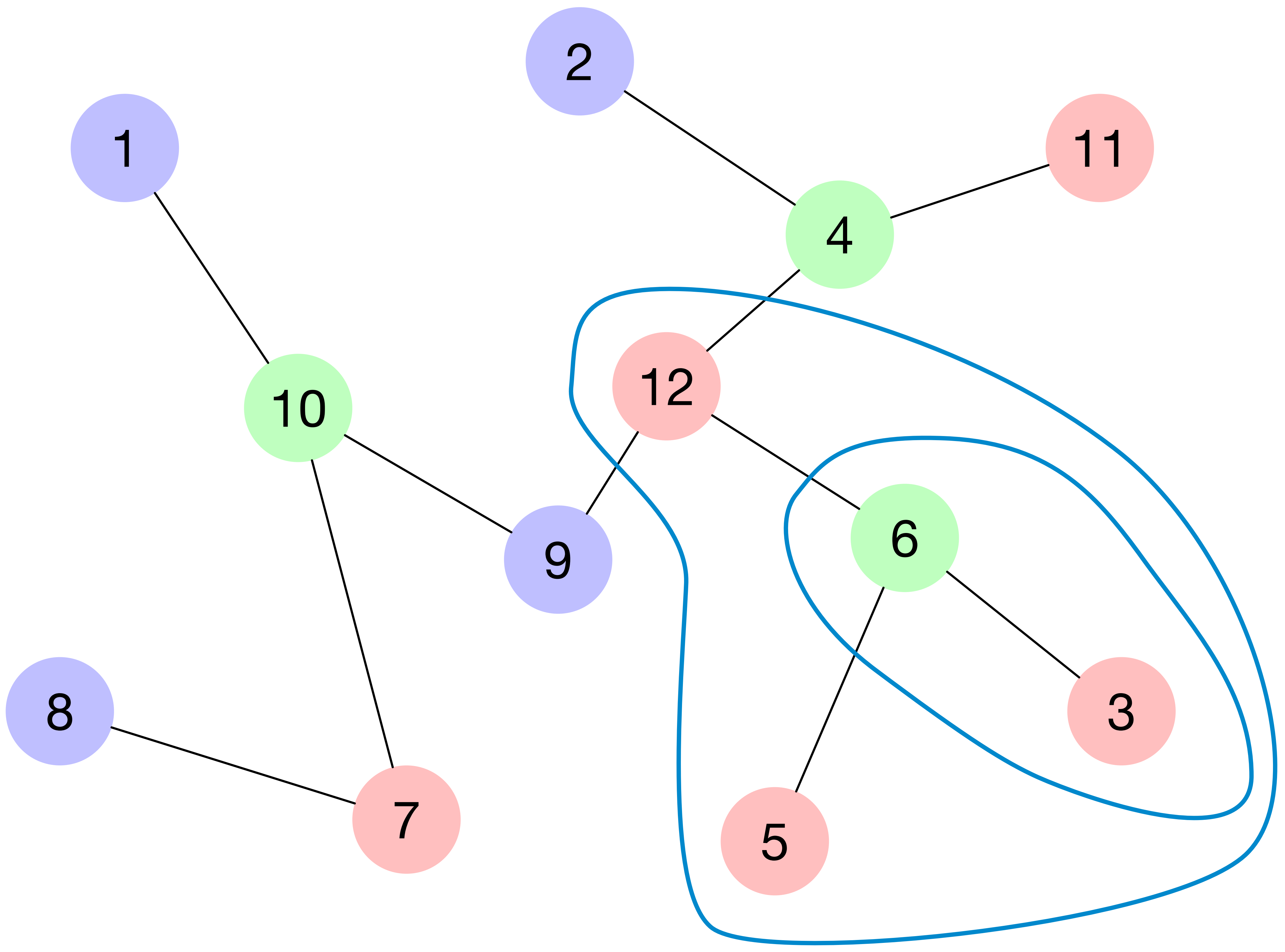
# Locality (Example)

- A 1-round algorithm is just a mapping from radius-1 balls to outputs



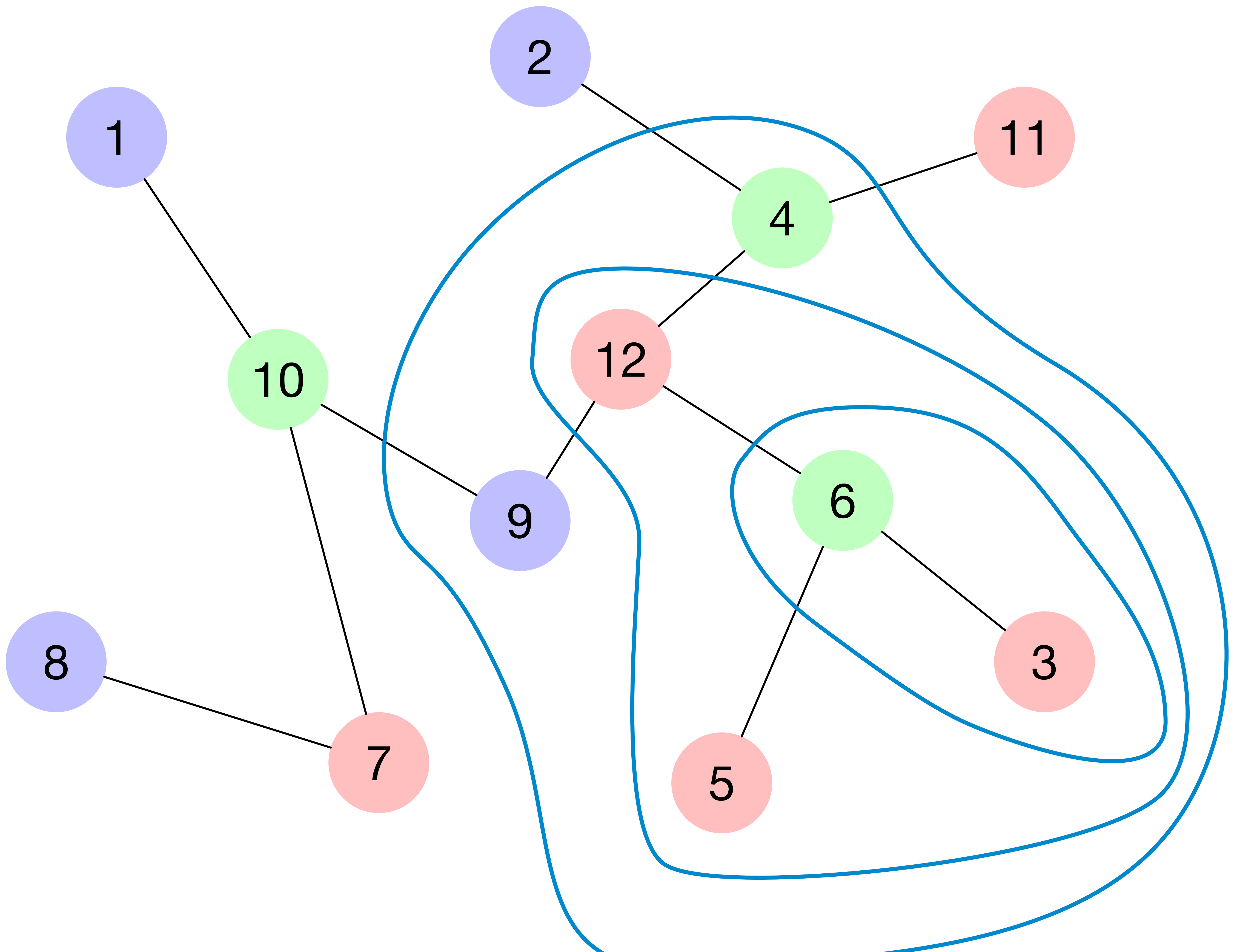
# Locality (Example)

- A T-round algorithm is just a mapping from radius-T balls to outputs



# Locality (Example)

- A T-round algorithm is just a mapping from radius-T balls to outputs



# Locality

A $T$-round algorithm is just a mapping from radius-$T$ balls to outputs.

# Locality

A T-round algorithm is just a mapping from radius-T balls to outputs.

Proof:

# Locality

A T-round algorithm is just a mapping from radius-T balls to outputs.

Proof:

- The **state** of node **v** at time T, depends on:

# Locality

A T-round algorithm is just a mapping from radius-T balls to outputs.

Proof:

- The **state** of node **v** at time **T**, depends on:
  - The **state** of node **v** at time **T-1**, and

# Locality

A T-round algorithm is just a mapping from radius-T balls to outputs.

Proof:

- The **state** of node **v** at time **T**, depends on:
  - The **state** of node **v** at time **T-1**, and
  - The **messages** received by **v** at time **T**, that only depend on:

# Locality

A T-round algorithm is just a mapping from radius-T balls to outputs.

Proof:

- The **state** of node **v** at time **T**, depends on:
  - The **state** of node **v** at time **T-1**, and
  - The **messages** received by **v** at time **T**, that only depend on:
    - the **state** of the **neighbors** of v at time **T-1**

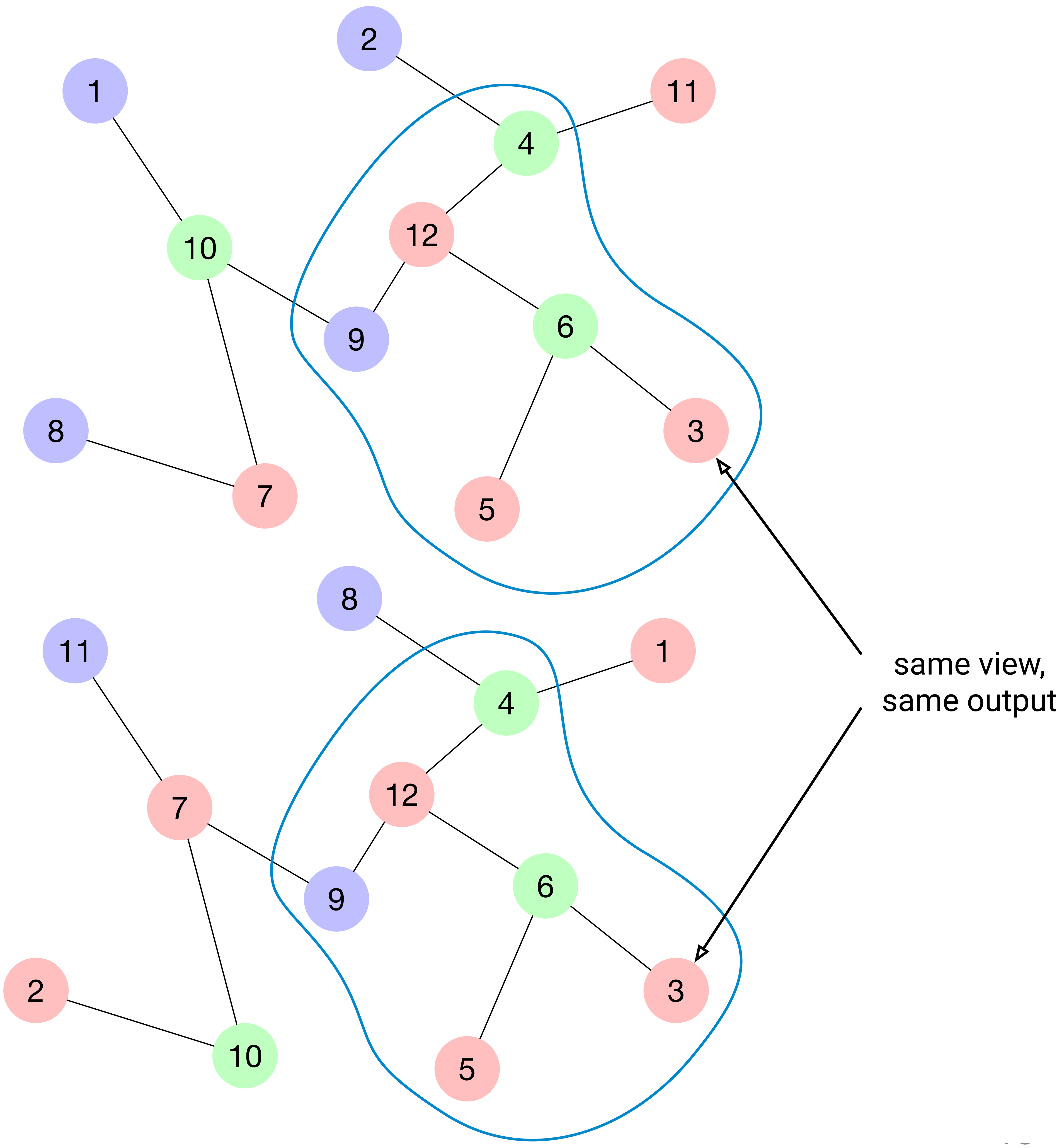
# Main technique to prove lower bounds

same radius-T view

⇩

*any* T-round algorithm

outputs the same



# Main technique to prove lower bounds

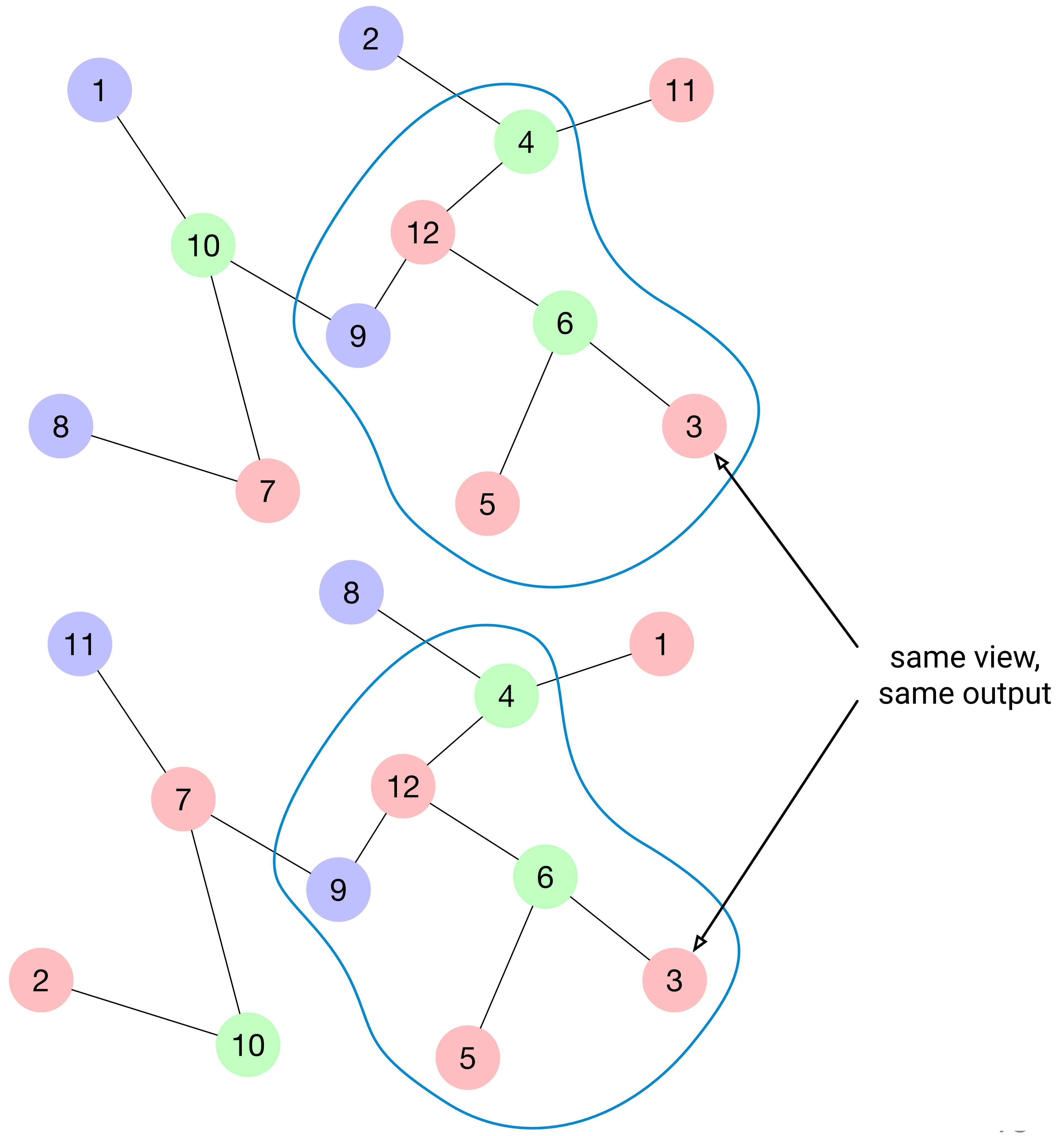
same radius-T view

⇩

*any* T-round algorithm

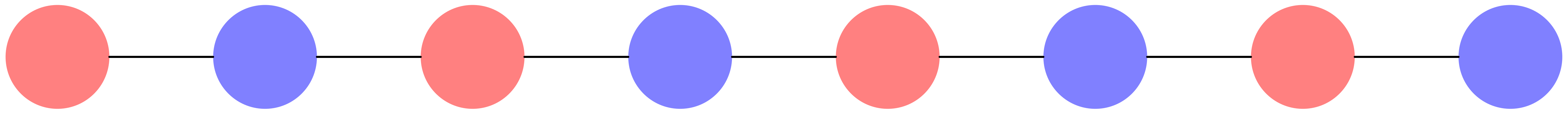
outputs the same

(different algorithms may output different things,
but all algorithms will output the same in both instances)



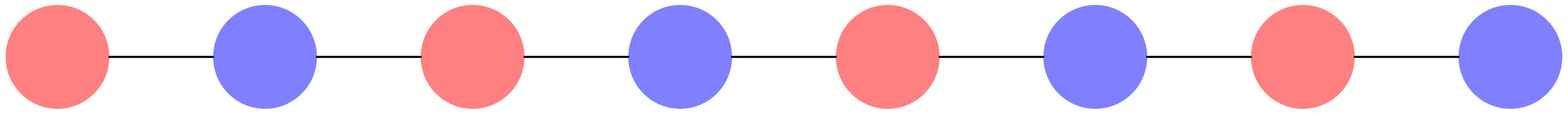
# 2-coloring

- We can solve 2-coloring in **O(n)** rounds on paths



# 2-coloring

- We can solve 2-coloring in **$O(n)$** rounds on paths
- We can prove that **$\Omega(n)$** rounds are required, even if:
  - The value of **$n$** is **known** to all nodes
  - IDs are exactly from **$\{1, ..., n\}$**
  - Nodes can use **randomization**



# 2-coloring lower bound

# 2-coloring lower bound

- We want to prove that coloring requires **$\Omega(n)$** on paths

# 2-coloring lower bound

- We want to prove that coloring requires **$\Omega$(n)** on paths
- We will prove that any **T(n) $\in$ o(n)** rounds algorithm must **fail**.

# 2-coloring lower bound

- We want to prove that coloring requires **$\Omega(n)$** on paths
- We will prove that any **$T(n) \in o(n)$** rounds algorithm must **fail**.
- $T(n) \in o(n)$: $\forall\, \epsilon, \exists\, k, \forall\, n > k, T(n) < \epsilon\, n$

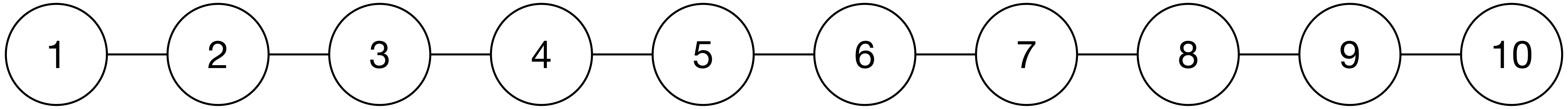
# 2-coloring lower bound

- We want to prove that coloring requires **$\Omega(n)$** on paths
- We will prove that any **$T(n) \in o(n)$** rounds algorithm must **fail**.
- $T(n) \in o(n)$: $\forall\ \epsilon, \exists\ k, \forall\ n > k, T(n) < \epsilon\ n$
- If we take **n large enough**, the algorithm must terminate in **at most n/5** rounds.

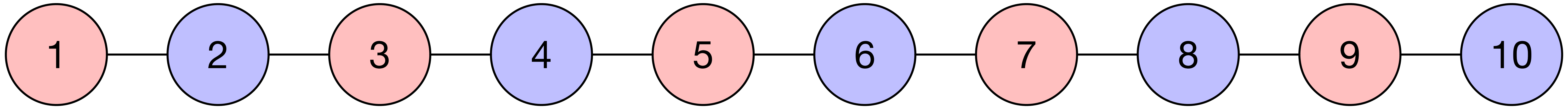
# 2-coloring lower bound

- Let us prove that **n/5** rounds are not enough, for all n.
- The high level idea is that we build two instances such that:
  - There are two pairs of nodes that have the same view in both instances
  - Such nodes cannot output the same in both instances

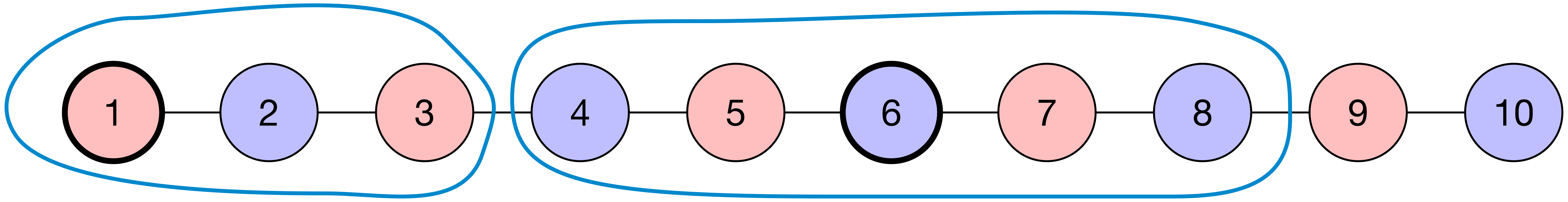
# 2-coloring lower bound



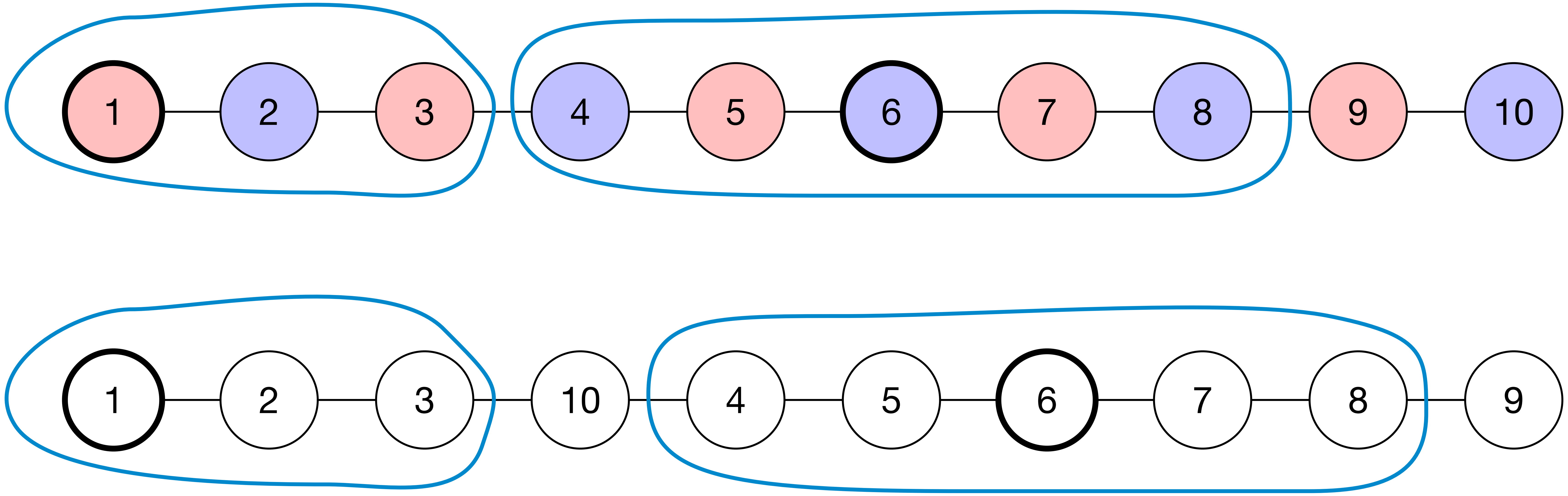
# 2-coloring lower bound



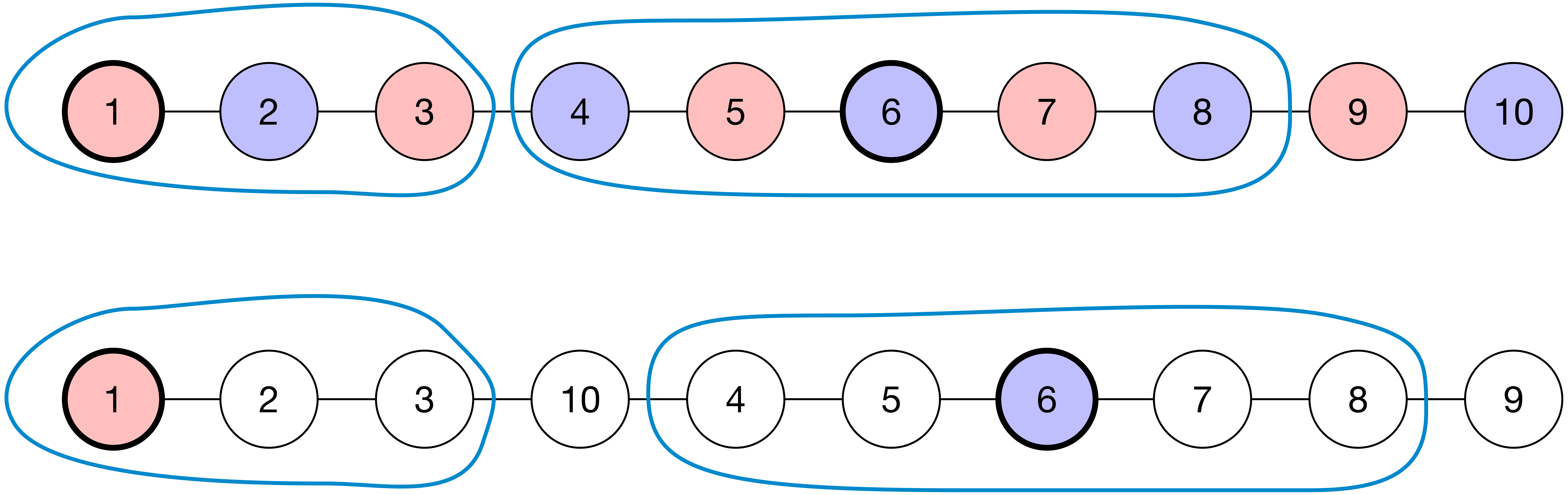
# 2-coloring lower bound



# 2-coloring lower bound



# 2-coloring lower bound



# 2-coloring lower bound

# 2-coloring lower bound

- Consider the path of length n, where there is an edge between nodes **i** and **i+1**

# 2-coloring lower bound

- Consider the path of length n, where there is an edge between nodes **i** and **i+1**

**(1)** - **(2)** - … - **(n/5+1)** - **(n/5+2)** - … - **(n/2+1)** - … - **(n-1)** - **(n)**

# 2-coloring lower bound

- Consider the path of length n, where there is an edge between nodes **i** and **i+1**

  **(1) - (2) - … - (n/5+1) - (n/5+2) - … - (n/2+1) - … - (n-1) - (n)**

- Create a new instance, obtained by **removing** the edges **{n-1,n}** and **{n/5+1,n/5+2}**, and **adding** the edges **{n/5+1,n}** and **{n,n/5+2}**

# 2-coloring lower bound

- Consider the path of length n, where there is an edge between nodes **i** and **i+1**

**(1)** - **(2)** - ... - **(n/5+1)** - **(n/5+2)** - ... - **(n/2+1)** - ... - **(n-1)** - **(n)**

- Create a new instance, obtained by **removing** the edges **{n-1,n}** and **{n/5+1,n/5+2}**, and **adding** the edges **{n/5+1,n}** and **{n,n/5+2}**

**(1)** - **(2)** - ... - **(n/5+1)** - **(n)** - **(n/5+2)** - ... - **(n/2+1)** - ...- **(n-1)**

# 2-coloring lower bound

- Consider the path of length n, where there is an edge between nodes **i** and **i+1**

  (1) - (2) - ... - (n/5+1) - (n/5+2) - ... - (n/2+1) - ... - (n-1) - (n)

- Create a new instance, obtained by **removing** the edges **{n-1,n}** and **{n/5+1,n/5+2}**, and **adding** the edges **{n/5+1,n}** and **{n,n/5+2}**

  (1) - (2) - ... - (n/5+1) - (n) - (n/5+2) - ... - (n/2+1) - ...- (n-1)

- For large enough n, nodes **1** and **(n/2+1)** have the **same radius-n/5 view**, hence they must output the same in both instances, but this is wrong (the distances of these nodes in the two instances have different parity)

# 2-coloring lower bound

- Consider the path of length n, where there is an edge between nodes **i** and **i+1**

  (1) - (2) - … - (n/5+1) - (n/5+2) - … - (n/2+1) - … - (n-1) - (n)

- Create a new instance, obtained by **removing** the edges **{n-1,n}** and **{n/5+1,n/5+2}**, and **adding** the edges **{n/5+1,n}** and **{n,n/5+2}**

  (1) - (2) - … - (n/5+1) - (n) - (n/5+2) - … - (n/2+1) - …- (n-1)

- For large enough n, nodes **1** and **(n/2+1)** have the **same radius-n/5 view**, hence they must output the same in both instances, but this is wrong (the distances of these nodes in the two instances have different parity)

# 2-coloring lower bound (randomized)

# 2-coloring lower bound (randomized)

- The proof works for **deterministic** algorithms, but it can be extended to work also for **randomized** algorithms.

# 2-coloring lower bound (randomized)

- The proof works for **deterministic** algorithms, but it can be extended to work also for **randomized** algorithms.

- Main ingredient:

same radius-T view

$$\Downarrow$$

same **probability distribution**

over the outputs

# Coloring trees

# Coloring trees

- The **3-coloring** problem can be solved in:
  - **$O(\log n)$** rounds on **trees**
  - **$O(\log^* n)$** rounds on rooted **trees**

# Coloring trees

- The **3-coloring** problem can be solved in:
  - **$O(\log n)$** rounds on **trees**
  - **$O(\log^* n)$** rounds on rooted **trees**
- What can we do in **$o(\log n)$** rounds on trees?

# Coloring trees

- The **3-coloring** problem can be solved in:
  - **O(log n)** rounds on **trees**
  - **O(log* n)** rounds on rooted **trees**
- What can we do in **o(log n)** rounds on trees?
- Do we really need to have a **rooted** tree to solve **3**-coloring fast?

# Coloring trees

- The **3-coloring** problem can be solved in:
  - **O(log n)** rounds on **trees**
  - **O(log* n)** rounds on rooted **trees**
- What can we do in **o(log n)** rounds on trees?
- Do we really need to have a **rooted** tree to solve **3**-coloring fast?

# Coloring trees

- The **3-coloring** problem can be solved in:
  - **$O(\log n)$** rounds on **trees**
  - **$O(\log^* n)$** rounds on rooted **trees**
- What can we do in **$o(\log n)$** rounds on trees?
- Do we really need to have a **rooted** tree to solve **3**-coloring fast?

**$o(\Delta / \log \Delta)$** coloring trees of maximum degree **$\Delta$** requires **$\Omega(\log_\Delta n)$** rounds

# Coloring trees lower bound

# Coloring trees lower bound

- $o(\Delta / \log \Delta)$ coloring trees of maximum degree $\Delta$ requires $\Omega(\log_\Delta n)$ rounds

# Coloring trees lower bound

- **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **$\Delta$** requires **$\Omega(\log_\Delta n)$** rounds
- We use the fact that there are graphs that:
  - cannot be colored using **$o(\Delta / \log \Delta)$ colors**
  - they look like a tree, in every **$o(\log_\Delta n)$** radius neighborhood

# Coloring trees lower bound

# Coloring trees lower bound

**Theorem (Bollobas '78):**

# Coloring trees lower bound

**Theorem (Bollobas '78):**

There exists an infinite family **H** of **n**-node graphs where:

# Coloring trees lower bound

**Theorem (Bollobas '78):**

There exists an infinite family **H** of **n**-node graphs where:

- all nodes have **degree Δ**

# Coloring trees lower bound

**Theorem (Bollobas '78):**

There exists an infinite family **H** of **n**-node graphs where:

- all nodes have **degree Δ**
- the **girth** is $\Omega(\log_\Delta n)$

# Coloring trees lower bound

**Theorem (Bollobas '78):**

There exists an infinite family **H** of **n**-node graphs where:

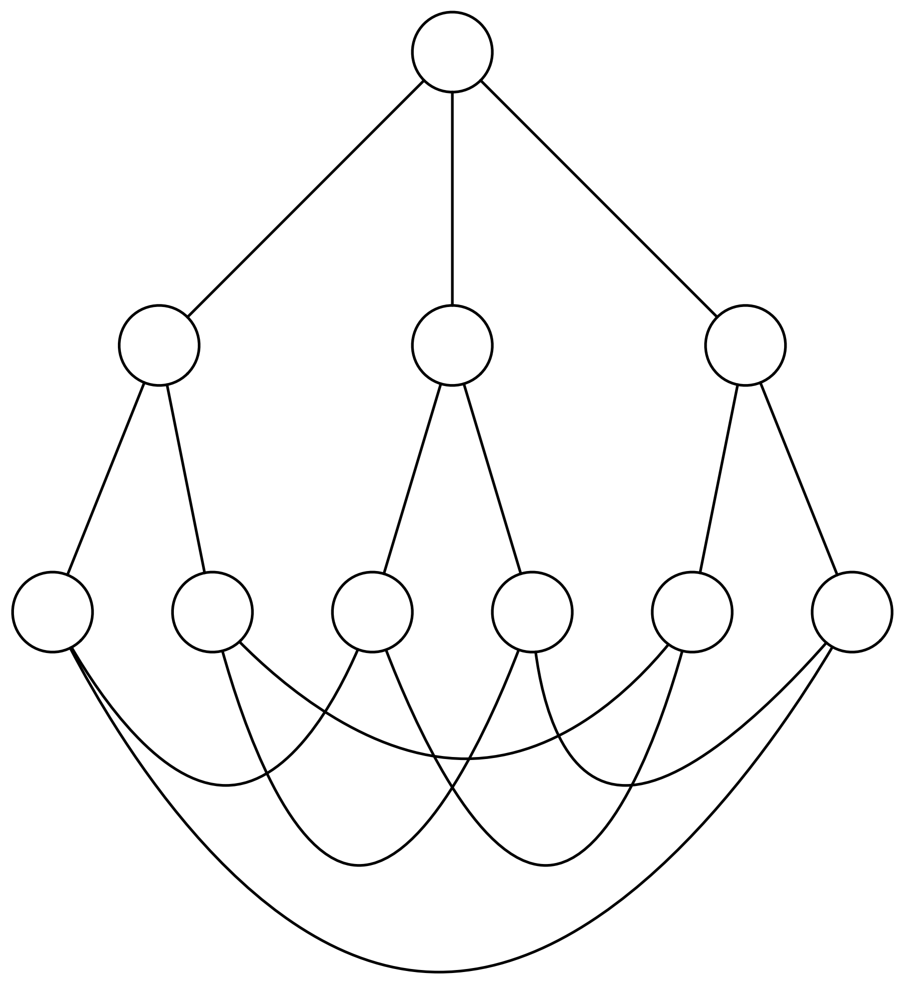
- all nodes have **degree Δ**
- the **girth** is $\Omega(\log_\Delta n)$
- the **chromatic number** is $\Omega(\Delta / \log \Delta)$

# Coloring trees lower bound

**Theorem (Bollobas '78):**

There exists an infinite family **H** of **n**-node graphs where:

- all nodes have **degree Δ**
- the **girth** is $\Omega(\log_\Delta n)$
- the **chromatic number** is $\Omega(\Delta / \log \Delta)$



# Coloring trees lower bound

# Coloring trees lower bound

- We want to prove that **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **$\Delta$** requires **$\Omega(\log_\Delta n)$** rounds.

# Coloring trees lower bound

- We want to prove that **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **$\Delta$** requires **$\Omega(\log_\Delta n)$** rounds.

- Let us assume that there is an algorithm **A** that colors trees using **$o(\Delta / \log \Delta)$** colors and runs in **$o(\log_\Delta n)$** rounds. We show that we reach a contradiction.

# Coloring trees lower bound

- We want to prove that $o(\Delta / \log \Delta)$ coloring trees of maximum degree $\Delta$ requires $\Omega(\log_\Delta n)$ rounds.
- Let us assume that there is an algorithm **A** that colors trees using $o(\Delta / \log \Delta)$ colors and runs in $o(\log_\Delta n)$ rounds. We show that we reach a contradiction.

# Coloring trees lower bound

- We want to prove that $o(\Delta / \log \Delta)$ coloring trees of maximum degree $\Delta$ requires $\Omega(\log_\Delta n)$ rounds.
- Let us assume that there is an algorithm **A** that colors trees using $o(\Delta / \log \Delta)$ colors and runs in $o(\log_\Delta n)$ rounds. We show that we reach a contradiction.
- What happens if we run **A** on the graphs of the family **H**?

# Coloring trees lower bound

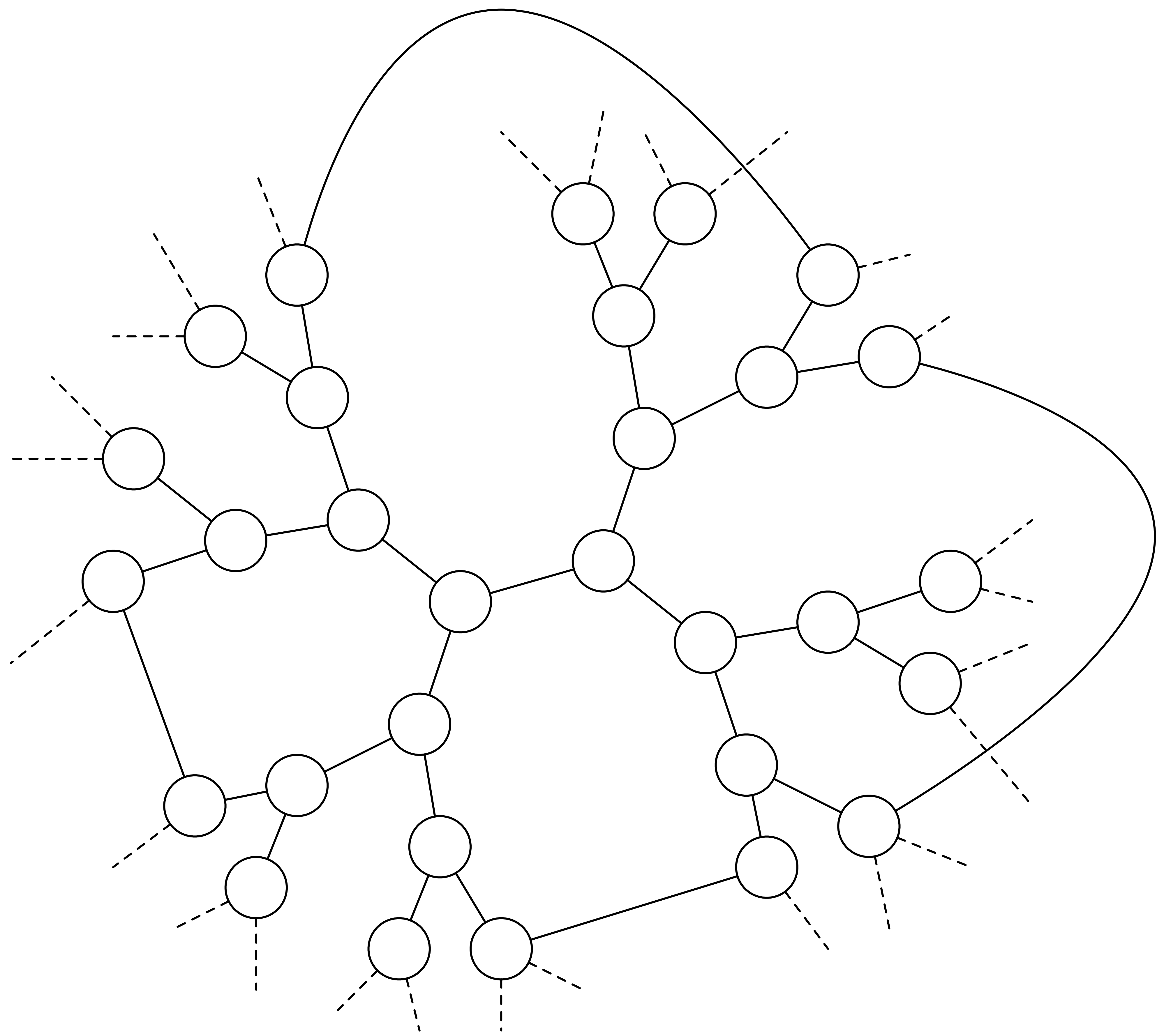
- We want to prove that **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **$\Delta$** requires **$\Omega(\log_\Delta n)$** rounds.
- Let us assume that there is an algorithm **A** that colors trees using **$o(\Delta / \log \Delta)$** colors and runs in **$o(\log_\Delta n)$** rounds. We show that we reach a contradiction.
- What happens if we run **A** on the graphs of the family **H**?
  - It must **fail**! Such graphs cannot be colored using **$o(\Delta / \log \Delta)$** colors, since the **chromatic number** is **$\Omega(\Delta / \log \Delta)$**

# Coloring trees lower bound

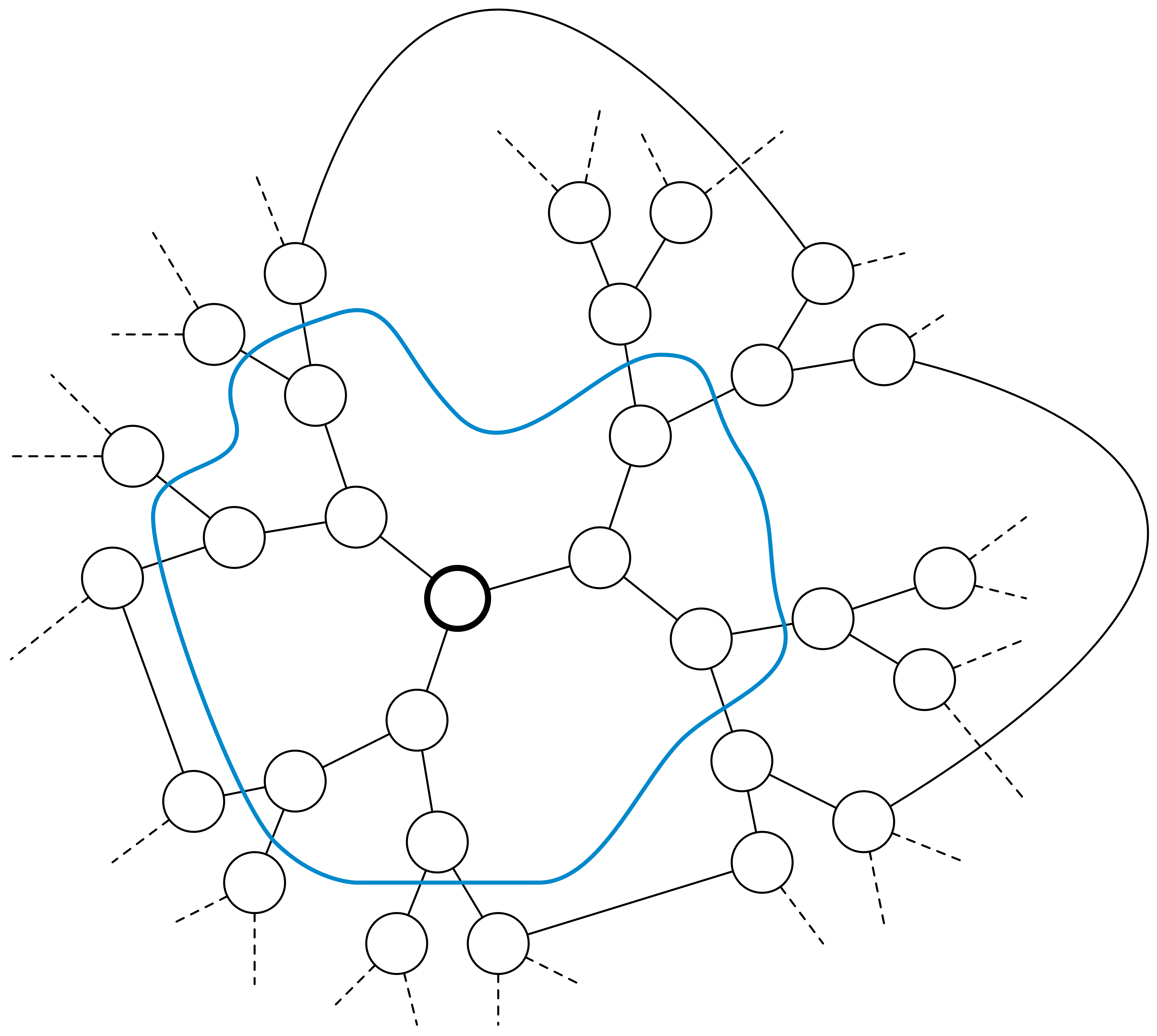
- We want to prove that $o(\Delta / \log \Delta)$ coloring trees of maximum degree $\Delta$ requires $\Omega(\log_\Delta n)$ rounds.
- Let us assume that there is an algorithm **A** that colors trees using $o(\Delta / \log \Delta)$ colors and runs in $o(\log_\Delta n)$ rounds. We show that we reach a contradiction.

- What happens if we run **A** on the graphs of the family **H**?
  - It must **fail**! Such graphs cannot be colored using $o(\Delta / \log \Delta)$ colors, since the **chromatic number** is $\Omega(\Delta / \log \Delta)$
  - We now prove that such failure implies that **A** must also **fail** on some specific **tree**

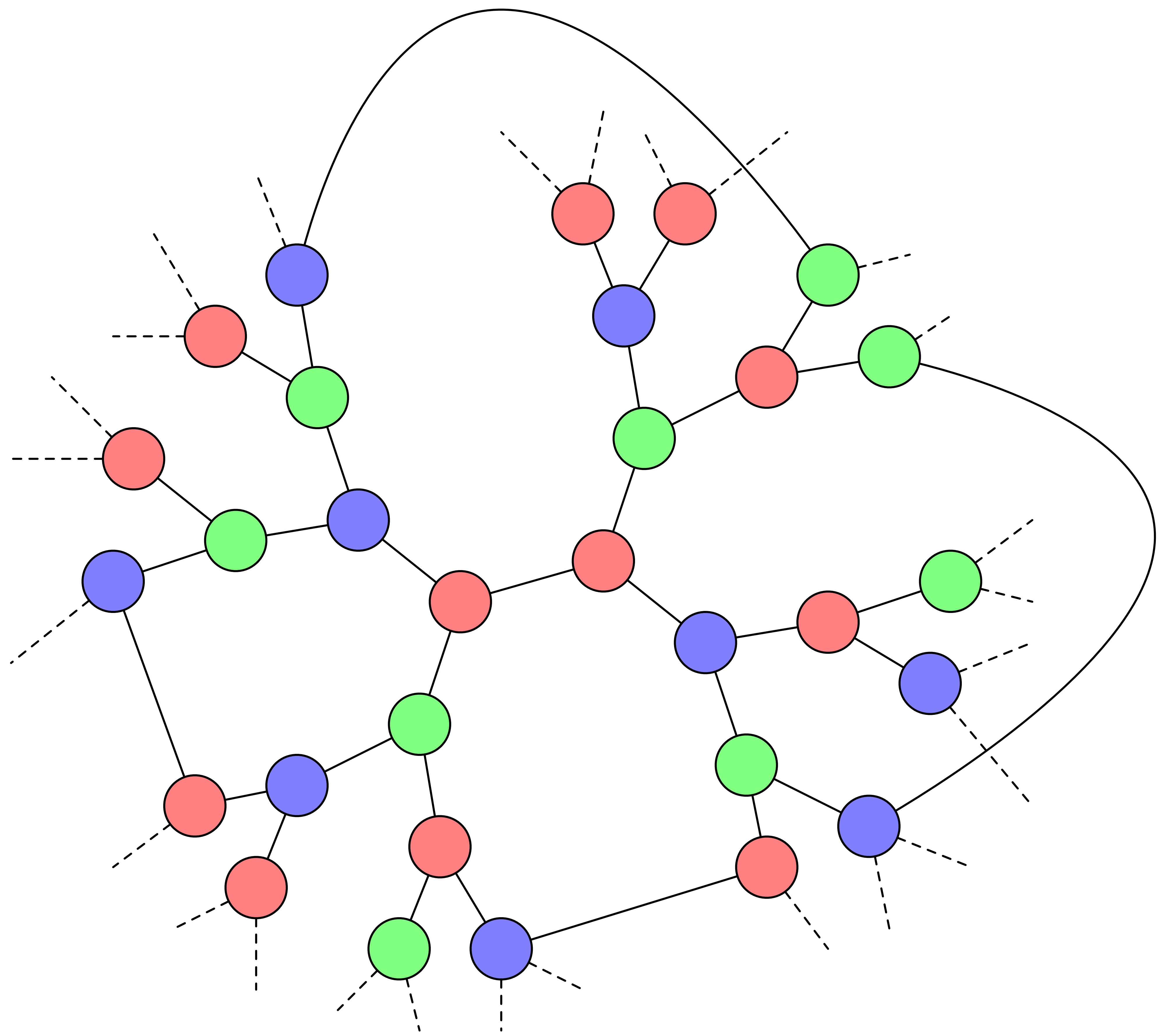
# Coloring trees lower bound



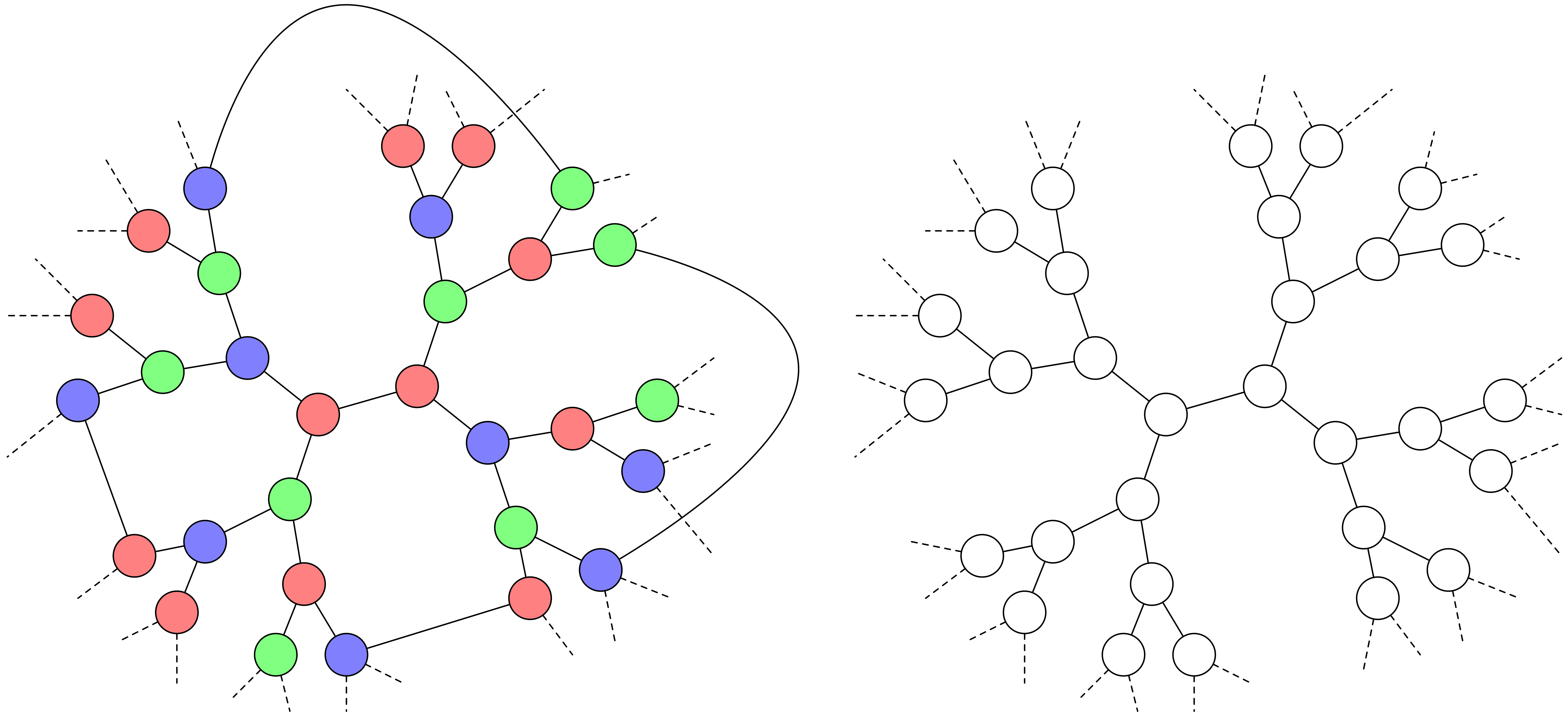
# Coloring trees lower bound



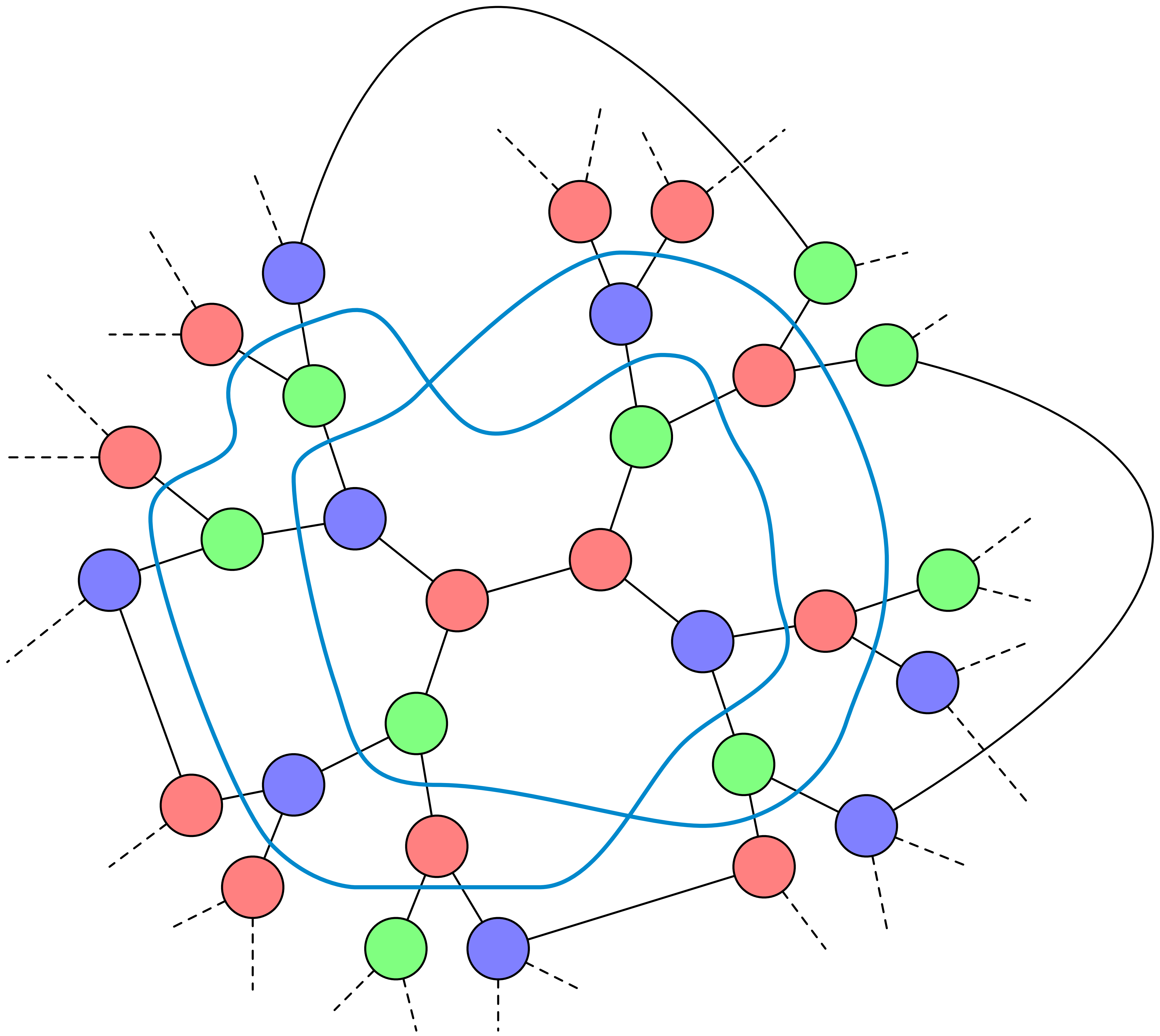
# Coloring trees lower bound

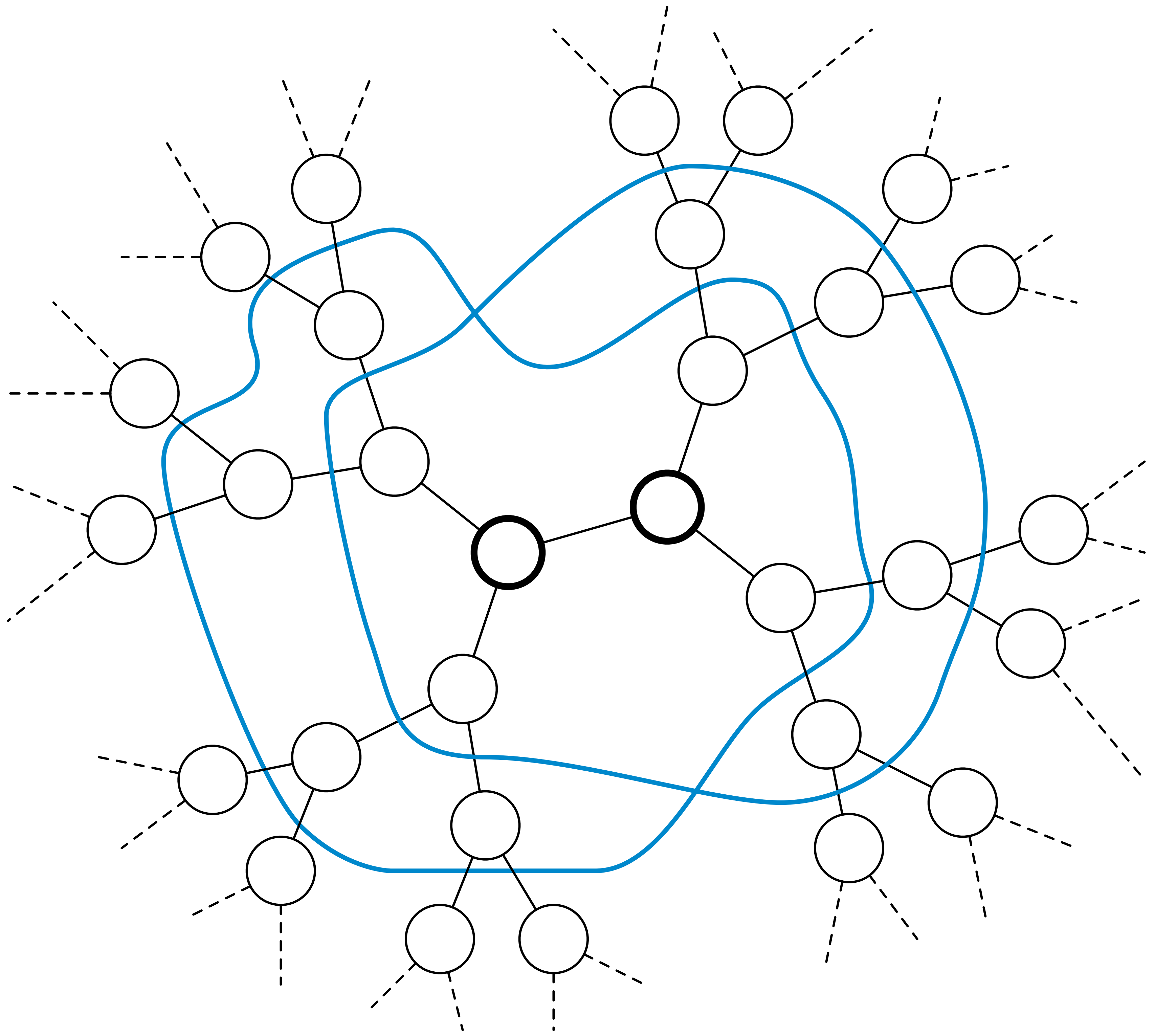


# Coloring trees lower bound

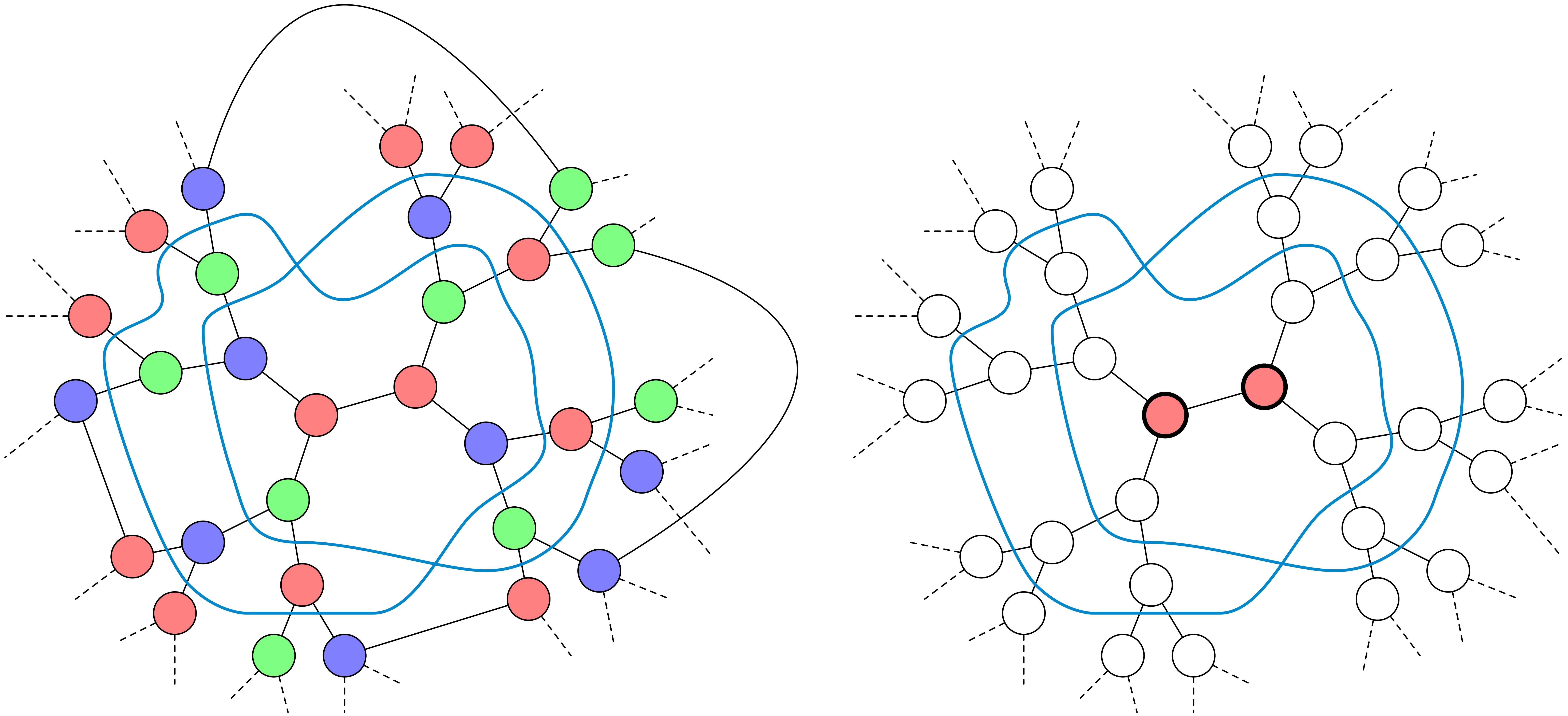


# Coloring trees lower bound





# Coloring trees lower bound



# Coloring trees lower bound

# Coloring trees lower bound

- We want to prove that **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **$\Delta$** requires **$\Omega(\log_\Delta n)$** rounds.

# Coloring trees lower bound

- We want to prove that $o(\Delta / \log \Delta)$ coloring trees of maximum degree $\Delta$ requires $\Omega(\log_\Delta n)$ rounds.
- Let us assume that there is an algorithm **A** that colors trees using $o(\Delta / \log \Delta)$ colors and runs in $o(\log_\Delta n)$ rounds.

# Coloring trees lower bound

- We want to prove that $o(\Delta / \log \Delta)$ coloring trees of maximum degree $\Delta$ requires $\Omega(\log_\Delta n)$ rounds.
- Let us assume that there is an algorithm **A** that colors trees using $o(\Delta / \log \Delta)$ colors and runs in $o(\log_\Delta n)$ rounds.
- We run **A** on the graphs of the family **H** (graphs that are $\Delta$-regular, with girth $\Omega(\log_\Delta n)$, and chromatic number $\Omega(\Delta / \log \Delta)$). It must fail.

# Coloring trees lower bound

- We want to prove that **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **$\Delta$** requires **$\Omega(\log_\Delta n)$** rounds.
- Let us assume that there is an algorithm **A** that colors trees using **$o(\Delta / \log \Delta)$** colors and runs in **$o(\log_\Delta n)$** rounds.
- We run **A** on the graphs of the family **H** (graphs that are **$\Delta$**-regular, with girth **$\Omega(\log_\Delta n)$**, and chromatic number **$\Omega(\Delta / \log \Delta)$**). It must fail.
- We take two neighboring nodes that gave the same output, and the subgraph **T'** incuced by the union of their views. We create a tree **T** containing **T'** as a subtree.

# Coloring trees lower bound

- We want to prove that $o(\Delta / \log \Delta)$ coloring trees of maximum degree $\Delta$ requires $\Omega(\log_\Delta n)$ rounds.
- Let us assume that there is an algorithm **A** that colors trees using $o(\Delta / \log \Delta)$ colors and runs in $o(\log_\Delta n)$ rounds.
- We run **A** on the graphs of the family **H** (graphs that are $\Delta$-regular, with girth $\Omega(\log_\Delta n)$, and chromatic number $\Omega(\Delta / \log \Delta)$). It must fail.
- We take two neighboring nodes that gave the same output, and the subgraph **T'** incuced by the union of their views. We create a tree **T** containing **T'** as a subtree.
- **A** must **fail** on the **tree T**. Contradiction!

# Coloring trees lower bound

# Coloring trees lower bound

- We saw how to prove:

# Coloring trees lower bound

- We saw how to prove:
  - Coloring trees of maximum degree **$\Delta$** with **$o(\Delta / \log \Delta)$** colors requires **$\Omega(\log_\Delta n)$** rounds.

# Coloring trees lower bound

- We saw how to prove:
  - Coloring trees of maximum degree **$\Delta$** with **$o(\Delta / \log \Delta)$** colors requires **$\Omega(\log_\Delta n)$** rounds.
- It is possible to prove:

# Coloring trees lower bound

- We saw how to prove:
  - Coloring trees of maximum degree **$\Delta$** with **$o(\Delta / \log \Delta)$** colors requires **$\Omega(\log_\Delta n)$** rounds.
- It is possible to prove:
  - Coloring trees of maximum degree **$\Delta$** with **$\Delta$** colors requires **$\Omega(\log_\Delta n)$** rounds.

# Coloring trees lower bound

- We saw how to prove:
  - Coloring trees of maximum degree **$\Delta$** with **$o(\Delta / \log \Delta)$** colors requires **$\Omega(\log_\Delta n)$** rounds.
- It is possible to prove:
  - Coloring trees of maximum degree **$\Delta$** with **$\Delta$** colors requires **$\Omega(\log_\Delta n)$** rounds.
  - Different techniques are required to prove such result.

# Coloring paths and cycles

# Coloring paths and cycles

- 3-coloring paths or cycles requires **$\Omega(\log^* n)$** rounds

# Coloring paths and cycles

- 3-coloring paths or cycles requires **$\Omega(\log^* n)$** rounds

[Linial '87] [Naor '91] [Naor, Stockmeyer '93] [Laurinharju, Suomela '14]

# Coloring paths and cycles

- 3-coloring paths or cycles requires **$\Omega(\log^* n)$** rounds

  [Linial '87] [Naor '91] [Naor, Stockmeyer '93] [Laurinharju, Suomela '14]

- High level idea:

# Coloring paths and cycles

- 3-coloring paths or cycles requires **$\Omega(\log^* n)$** rounds

  [Linial '87] [Naor '91] [Naor, Stockmeyer '93] [Laurinharju, Suomela '14]

- High level idea:
  - If **$c$**-coloring can be solved in **$T$** rounds, then **$2^c$**-coloring can be solved in **$T-1$** rounds

# Coloring paths and cycles

- 3-coloring paths or cycles requires **$\Omega(\log^* n)$** rounds

  [Linial '87] [Naor '91] [Naor, Stockmeyer '93] [Laurinharju, Suomela '14]

- High level idea:
  - If **c**-coloring can be solved in **T** rounds, then **$2^c$**-coloring can be solved in **T-1** rounds
  - **o(n)** coloring **cannot** be solved in **0** rounds

# Coloring paths and cycles

- 3-coloring paths or cycles requires **$\Omega(\log^* n)$** rounds

  [Linial '87] [Naor '91] [Naor, Stockmeyer '93] [Laurinharju, Suomela '14]

- High level idea:
  - If **$c$**-coloring can be solved in **$T$** rounds, then **$2^c$**-coloring can be solved in **$T-1$** rounds
  - **$o(n)$** coloring **cannot** be solved in **0** rounds
  - If we start from **$T=o(\log^* n)$** we get a **contradiction**

# Coloring algorithms

- We can see an algorithm $A$ as a function satisfying that:

$$A_n(x_1, \ldots, x_{2T+1}) \in \{1, 2, 3\}$$

$$A_n(x_1, \ldots, x_{2T+1}) \neq A_n(x_2, \ldots, x_{2T+2})$$

assuming $x_1, \ldots, x_{2T+2}$ are all distinct numbers from $\{1, \ldots, n\}$

# Coloring functions

# Coloring functions

- $A$ is a $k$-ary $c$-coloring function if:

$$A_n(x_1, \ldots, x_k) \in \{1, 2, \ldots, c\}$$

$$A_n(x_1, \ldots, x_k) \neq A_n(x_2, \ldots, x_{k+1})$$

assuming $x_1, \ldots, x_{k+1}$ are all distinct numbers from $\{1, \ldots, n\}$

satisfying $1 \leq x_1 \leq x_2 \leq \ldots \leq x_k \leq x_{k+1} \leq n$

# Coloring functions

- $A$ is a $k$-ary $c$-coloring function if:

$$A_n(x_1, \ldots, x_k) \in \{1, 2, \ldots, c\}$$

$$A_n(x_1, \ldots, x_k) \neq A_n(x_2, \ldots, x_{k+1})$$

assuming $x_1, \ldots, x_{k+1}$ are all distinct numbers from $\{1, \ldots, n\}$

satisfying $1 \leq x_1 \leq x_2 \leq \ldots \leq x_k \leq x_{k+1} \leq n$

- Any algorithm defines a $2T+1$-ary $3$-coloring function

# Coloring functions

# Coloring functions

- We will prove that for any **k**-ary **3**-coloring function:

$$k+1 \geq \log^* n$$

# Coloring functions

- We will prove that for any **k**-ary **3**-coloring function:

$$k+1 \geq \log^* n$$

- Since a **2T+1**-rounds coloring algorithm implies a **2T+1**-ary **3**-coloring function, we get that

$$2T+2 \geq \log^* n$$

$$T = \Omega(\log^* n)$$

# Coloring functions

- We will prove that for any **k**-ary **3**-coloring function:

$$k+1 \geq \log^* n$$

- Since a **2T+1**-rounds coloring algorithm implies a **2T+1**-ary **3**-coloring function, we get that

$$2T+2 \geq \log^* n$$

$$T = \Omega(\log^* n)$$

- We prove such statement by induction

# Coloring functions (base case)

# Coloring functions (base case)

- For any **1**-ary **c**-coloring function:

$$\mathbf{c \geq n}$$

# Coloring functions (base case)

- For any **1**-ary **c**-coloring function:

$$c \geq n$$

- Proof by contradiction. Assume that a **1**-ary **c**-coloring function exists, such that $c < n$

# Coloring functions (base case)

- For any **1**-ary **c**-coloring function:

$$c \geq n$$

- Proof by contradiction. Assume that a **1**-ary **c**-coloring function exists, such that $c < n$

- There must exist two numbers $1 \leq x_i < x_j \leq n$ such that

$$A_n(x_i) = A_n(x_j)$$

# Coloring functions (inductive case)

# Coloring functions (inductive case)

- We are given **A**, that is a **k**-ary **c**-coloring function

# Coloring functions (inductive case)

- We are given **A**, that is a **k**-ary **c**-coloring function
- We show that we can construct **B**, a **k-1**-ary $2^c$-coloring function

# Coloring functions (inductive case)

- We are given **A**, that is a **k**-ary **c**-coloring function
- We show that we can construct **B**, a **k-1**-ary **$2^c$**-coloring function
- Proof:

# Coloring functions (inductive case)

- We are given **A**, that is a **k**-ary **c**-coloring function
- We show that we can construct **B**, a **k-1**-ary $2^c$-coloring function
- Proof:

  We define $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, x_k) \mid n \geq x_k > x_{k-1} \}$

# Coloring functions (inductive case)

- We are given $A$, that is a $k$-ary $c$-coloring function
- We show that we can construct $B$, a $k-1$-ary $2^c$-coloring function
- Proof:

  We define $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, x_k) \mid n \geq x_k > x_{k-1} \}$

  Notice that there are $2^c$ possible outputs

# Coloring functions (inductive case)

- We are given **A**, that is a **k**-ary **c**-coloring function
- We show that we can construct **B**, a **k-1**-ary $2^c$-coloring function
- Proof:

  We define $B_n(x_1, ..., x_{k-1}) = \{ A_n(x_1, ..., x_{k-1}, x_k) \mid n \geq x_k > x_{k-1} \}$

  Notice that there are $2^c$ possible outputs

  Let us now prove that it is a coloring function

# Coloring functions (inductive case)

# Coloring functions (inductive case)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, x_k) \mid n \geq x_k > x_{k-1} \}$
- Assume for a contradiction that:

$$B_n(x_1, \ldots, x_{k-1}) = B_n(x_2, \ldots, x_k)$$

assuming $x_1, \ldots, x_k$ are all distinct numbers from $\{1, \ldots, n\}$

satisfying $1 \leq x_1 \leq x_2 \leq \ldots \leq x_{k-1} \leq x_k \leq n$

# Coloring functions (inductive case)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, x_k) \mid n \geq x_k > x_{k-1} \}$
- Assume for a contradiction that:

$$B_n(x_1, \ldots, x_{k-1}) = B_n(x_2, \ldots, x_k)$$

assuming $x_1, \ldots, x_k$ are all distinct numbers from $\{1, \ldots, n\}$

satisfying $1 \leq x_1 \leq x_2 \leq \ldots \leq x_{k-1} \leq x_k \leq n$

- Let $x = A_n(x_1, \ldots, x_k)$

# Coloring functions (inductive case)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, x_k) \mid n \geq x_k > x_{k-1} \}$
- Assume for a contradiction that:

$$B_n(x_1, \ldots, x_{k-1}) = B_n(x_2, \ldots, x_k)$$

assuming $x_1, \ldots, x_k$ are all distinct numbers from $\{1, \ldots, n\}$

satisfying $1 \leq x_1 \leq x_2 \leq \ldots \leq x_{k-1} \leq x_k \leq n$

- Let $x = A_n(x_1, \ldots, x_k)$
- By definition of $B$, we have that $x \in B_n(x_1, \ldots, x_{k-1})$

# Coloring functions (inductive case)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, x_k) \mid n \geq x_k > x_{k-1} \}$
- Assume for a contradiction that:

$$B_n(x_1, \ldots, x_{k-1}) = B_n(x_2, \ldots, x_k)$$

  assuming $x_1, \ldots, x_k$ are all distinct numbers from $\{1, \ldots, n\}$

  satisfying $1 \leq x_1 \leq x_2 \leq \ldots \leq x_{k-1} \leq x_k \leq n$

- Let $x = A_n(x_1, \ldots, x_k)$
- By definition of $B$, we have that $x \in B_n(x_1, \ldots, x_{k-1})$
- By assumption, we also have $x \in B_n(x_2, \ldots, x_k)$

# Coloring functions (inductive case)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1,} x_k) \mid n \geq x_k > x_{k-1} \}$
- Assume for a contradiction that:

$$B_n(x_1, \ldots, x_{k-1}) = B_n(x_2, \ldots, x_k)$$

assuming $x_1, \ldots, x_k$ are all distinct numbers from $\{1, \ldots, n\}$

satisfying $1 \leq x_1 \leq x_2 \leq \ldots \leq x_{k-1} \leq x_k \leq n$

- Let $x = A_n(x_1, \ldots, x_k)$
- By definition of $B$, we have that $x \in B_n(x_1, \ldots, x_{k-1})$
- By assumption, we also have $x \in B_n(x_2, \ldots, x_k)$
- This implies that there exists some $x_{k+1} > x_k$ such that $A_n(x_2, \ldots, x_{k,} x_{k+1}) = x$

# Coloring functions (putting things together)

# Coloring functions (putting things together)

- Given:

# Coloring functions (putting things together)

- Given:
  - a $T$-rounds coloring algorithm

# Coloring functions (putting things together)

- Given:
  - a $T$-rounds coloring algorithm
- We construct:

# Coloring functions (putting things together)

- Given:
  - a **T**-rounds coloring algorithm
- We construct:
  - a **k**-ary **3**-coloring function, where **k=2T+1**

# Coloring functions (putting things together)

- Given:
  - a **T**-rounds coloring algorithm
- We construct:
  - a **k**-ary **3**-coloring function, where **k=2T+1**
  - a **k**-ary $2^2$-coloring function

# Coloring functions (putting things together)

- Given:
  - a **T**-rounds coloring algorithm
- We construct:
  - a **k**-ary **3**-coloring function, where **k=2T+1**
  - a **k**-ary $2^2$-coloring function
  - a **k-1**-ary $2^{2^2}$-coloring function

# Coloring functions (putting things together)

- Given:
  - a **T**-rounds coloring algorithm
- We construct:
  - a **k**-ary **3**-coloring function, where **k=2T+1**
  - a **k**-ary $2^{2}$-coloring function
  - a **k-1**-ary $2^{2^{2}}$-coloring function
  - a **k-2**-ary $2^{2^{2^{2}}}$-coloring function

# Coloring functions (putting things together)

- Given:
  - a $T$-rounds coloring algorithm
- We construct:
  - a $k$-ary $3$-coloring function, where $k=2T+1$
  - a $k$-ary $2^2$-coloring function
  - a $k-1$-ary $2^{2^2}$-coloring function
  - a $k-2$-ary $2^{2^{2^2}}$-coloring function
  - ...

# Coloring functions (putting things together)

- Given:
  - a $T$-rounds coloring algorithm
- We construct:
  - a $k$-ary $3$-coloring function, where $k=2T+1$
  - a $k$-ary $2^2$-coloring function
  - a $k-1$-ary $2^{2^2}$-coloring function
  - a $k-2$-ary $2^{2^{2^2}}$-coloring function
  - ...
  - a $1$-ary ${}^{k+1}2$-coloring function (${}^{k+1}2$ is a power tower of height $k+1$)

# Coloring functions (putting things together)

- Given:
  - a **T**-rounds coloring algorithm
- We construct:
  - a **k**-ary **3**-coloring function, where **k=2T+1**
  - a **k**-ary $2^2$-coloring function
  - a **k-1**-ary $2^{2^2}$-coloring function
  - a **k-2**-ary $2^{2^{2^2}}$-coloring function
  - ...
  - a **1**-ary ${}^{k+1}2$-coloring function (${}^{k+1}2$ is a power tower of height **k+1**)
- In the base case we proved that ${}^{k+1}2 \geq n$, which implies $k+1 \geq \log^* n$, hence $T = \Omega(\log^* n)$

# Round elimination technique

# Round elimination technique

- **Given:**

# Round elimination technique

- **Given:**
  - algorithm $A_0$ solves problem $P_0$ in $T$ rounds

# Round elimination technique

- **Given:**
  - algorithm $\boldsymbol{A_0}$ solves problem $\boldsymbol{P_0}$ in $\boldsymbol{T}$ rounds
- **We construct:**

# Round elimination technique

- **Given:**
  - algorithm $\boldsymbol{A_0}$ solves problem $\boldsymbol{P_0}$ in $\boldsymbol{T}$ rounds
- **We construct:**
  - algorithm $\boldsymbol{A_1}$ solves problem $\boldsymbol{P_1}$ in $\boldsymbol{T - 1}$ rounds

# Round elimination technique

- **Given:**
  - algorithm $\boldsymbol{A_0}$ solves problem $\boldsymbol{P_0}$ in $\boldsymbol{T}$ rounds
- **We construct:**
  - algorithm $\boldsymbol{A_1}$ solves problem $\boldsymbol{P_1}$ in $\boldsymbol{T-1}$ rounds
  - algorithm $\boldsymbol{A_2}$ solves problem $\boldsymbol{P_2}$ in $\boldsymbol{T-2}$ rounds

# Round elimination technique

- **Given:**
  - algorithm $\boldsymbol{A_0}$ solves problem $\boldsymbol{P_0}$ in $\boldsymbol{T}$ rounds
- **We construct:**
  - algorithm $\boldsymbol{A_1}$ solves problem $\boldsymbol{P_1}$ in $\boldsymbol{T-1}$ rounds
  - algorithm $\boldsymbol{A_2}$ solves problem $\boldsymbol{P_2}$ in $\boldsymbol{T-2}$ rounds
  - algorithm $\boldsymbol{A_3}$ solves problem $\boldsymbol{P_3}$ in $\boldsymbol{T-3}$ rounds

    …

# Round elimination technique

- **Given:**
  - algorithm $\boldsymbol{A_0}$ solves problem $\boldsymbol{P_0}$ in $T$ rounds
- **We construct:**
  - algorithm $\boldsymbol{A_1}$ solves problem $\boldsymbol{P_1}$ in $T - 1$ rounds
  - algorithm $\boldsymbol{A_2}$ solves problem $\boldsymbol{P_2}$ in $T - 2$ rounds
  - algorithm $\boldsymbol{A_3}$ solves problem $\boldsymbol{P_3}$ in $T - 3$ rounds

    …
  - algorithm $\boldsymbol{A_T}$ solves problem $\boldsymbol{P_T}$ in $0$ rounds

# Round elimination technique

- **Given:**
  - algorithm $\boldsymbol{A_0}$ solves problem $\boldsymbol{P_0}$ in $T$ rounds
- **We construct:**
  - algorithm $\boldsymbol{A_1}$ solves problem $\boldsymbol{P_1}$ in $T - 1$ rounds
  - algorithm $\boldsymbol{A_2}$ solves problem $\boldsymbol{P_2}$ in $T - 2$ rounds
  - algorithm $\boldsymbol{A_3}$ solves problem $\boldsymbol{P_3}$ in $T - 3$ rounds

    …
  - algorithm $\boldsymbol{A_T}$ solves problem $\boldsymbol{P_T}$ in $0$ rounds
- **We prove:**

# Round elimination technique

- **Given:**
  - algorithm $A_0$ solves problem $P_0$ in $T$ rounds
- **We construct:**
  - algorithm $A_1$ solves problem $P_1$ in $T - 1$ rounds
  - algorithm $A_2$ solves problem $P_2$ in $T - 2$ rounds
  - algorithm $A_3$ solves problem $P_3$ in $T - 3$ rounds

    …
  - algorithm $A_T$ solves problem $P_T$ in $0$ rounds
- **We prove:**
  - $P_T$ cannot be solved in 0 rounds, so $A_0$ cannot exist

# Round elimination technique

- **Given:**
  - algorithm $\boldsymbol{A_0}$ solves problem $\boldsymbol{P_0}$ in $T$ rounds
- **We construct:**
  - algorithm $\boldsymbol{A_1}$ solves problem $\boldsymbol{P_1}$ in $T - 1$ rounds
  - algorithm $\boldsymbol{A_2}$ solves problem $\boldsymbol{P_2}$ in $T - 2$ rounds
  - algorithm $\boldsymbol{A_3}$ solves problem $\boldsymbol{P_3}$ in $T - 3$ rounds

    …
  - algorithm $\boldsymbol{A_T}$ solves problem $\boldsymbol{P_T}$ in **0** rounds
- **We prove:**
  - $\boldsymbol{P_T}$ cannot be solved in 0 rounds, so $\boldsymbol{A_0}$ cannot exist

Given a problem $\boldsymbol{P_i}$, satisfying that the correctness of the solution can be checked locally, the problem $\boldsymbol{P_{i+1}}$ can be defined mechanically [Brandt '19]