Lower Bounds

Dennis Olivetti

University of Freiburg, Germany

Lower Bounds

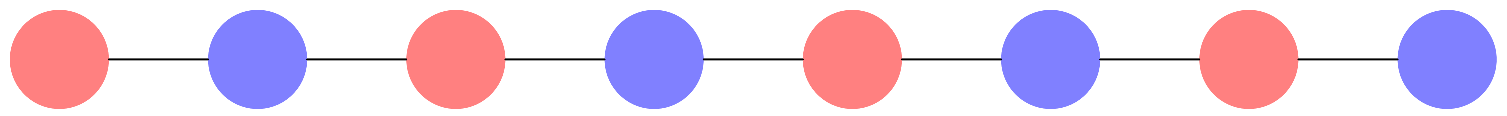
"*The 2-coloring problem requires* **$\Omega(n)$** rounds"

Dennis Olivetti

University of Freiburg, Germany

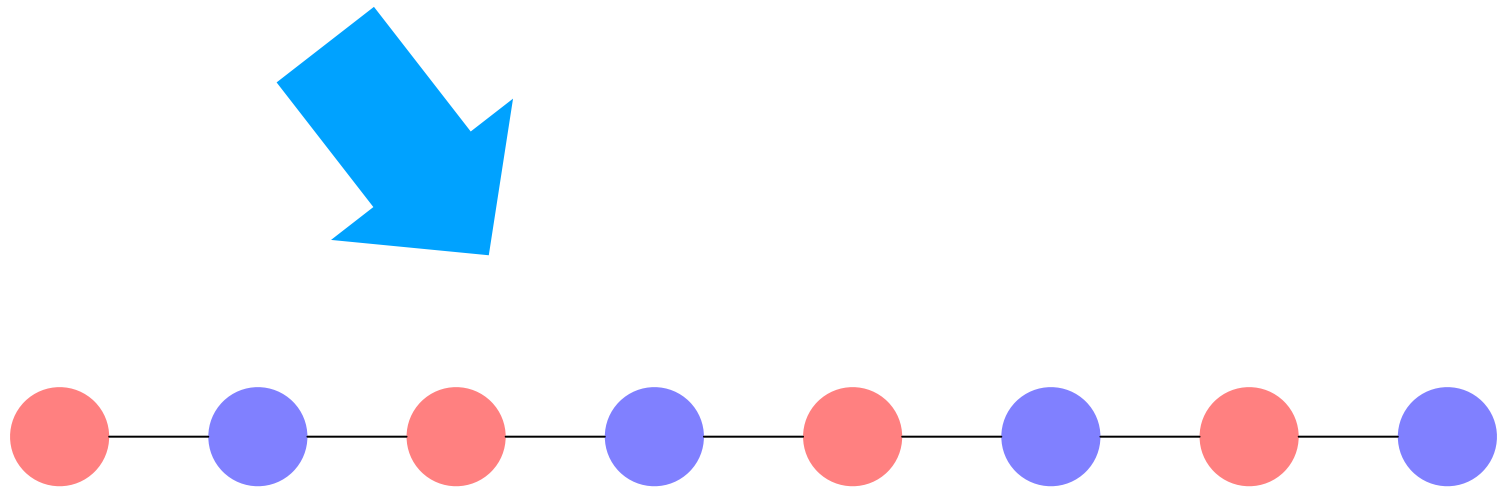
Lower bounds

- **2** coloring



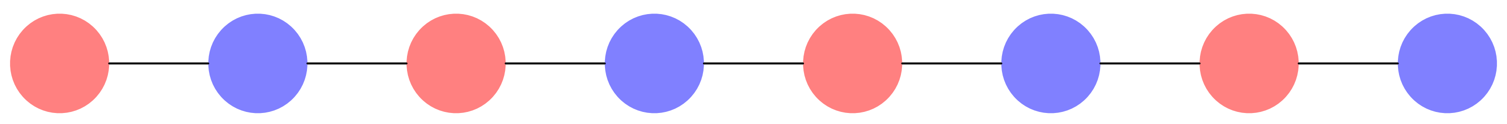
Lower bounds

- **2** coloring



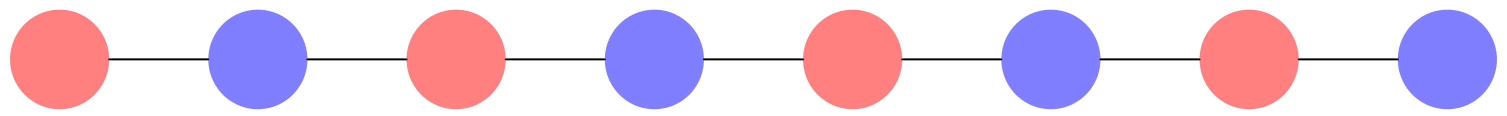
Lower bounds

- **2** coloring
  - can be solved in **O(n)** rounds



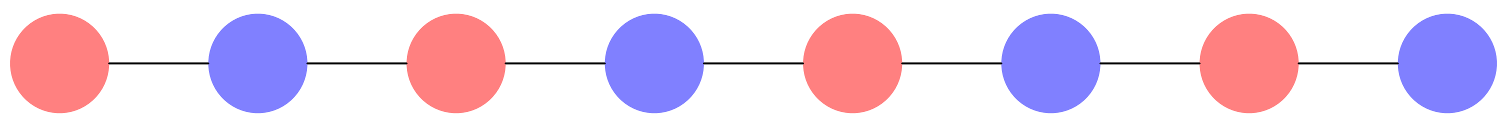
Lower bounds

- **2** coloring
  - can be solved in **O(n)** rounds
  - requires **Ω(n)** rounds



Lower bounds

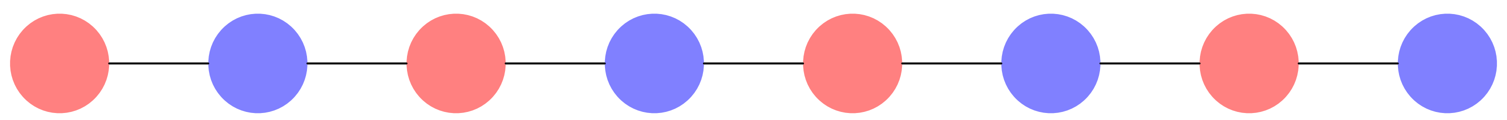
- 2 coloring
  - can be solved in O(n) rounds
  - requires Ω(n) rounds

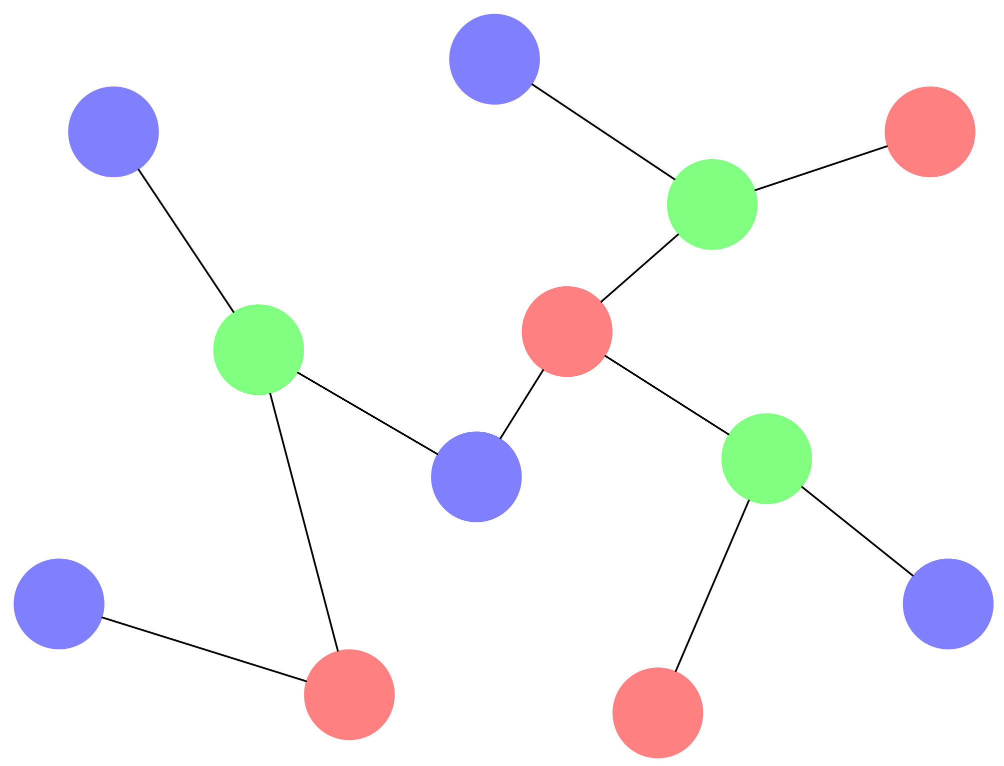


Lower bounds

- 2 coloring
  - can be solved in $O(n)$ rounds
  - requires $\Omega(n)$ rounds

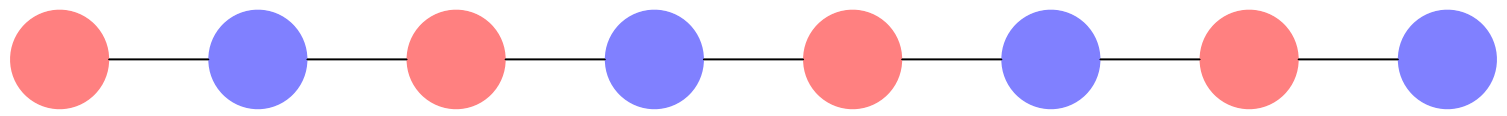
- Coloring trees



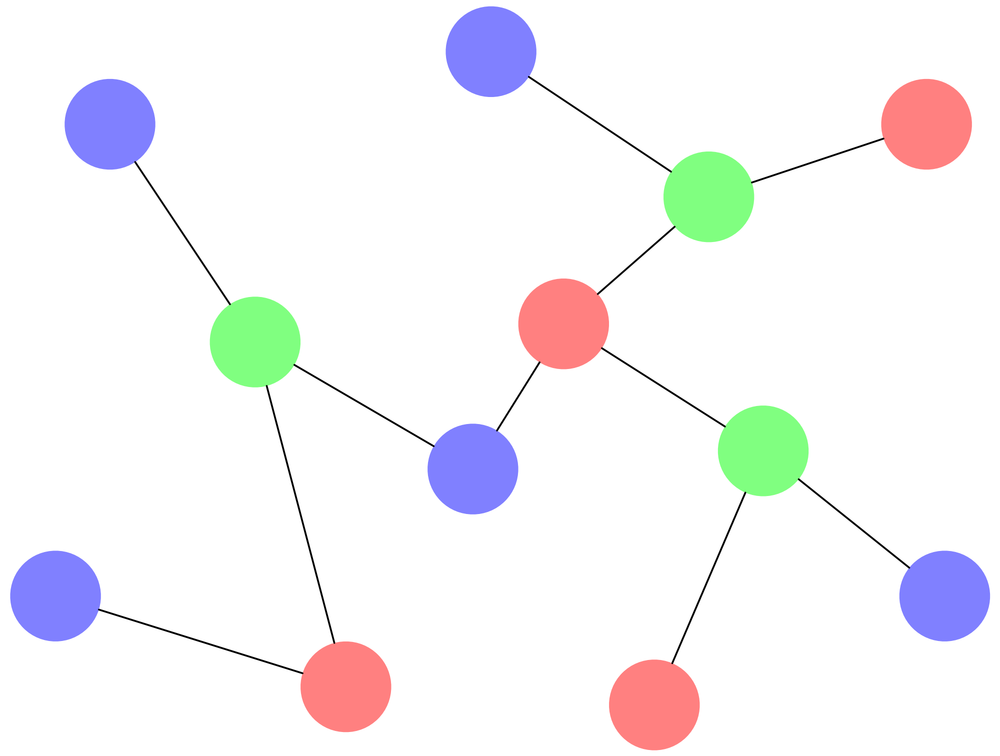


Lower bounds

- **2** coloring
  - can be solved in **O(n)** rounds
  - requires **Ω(n)** rounds

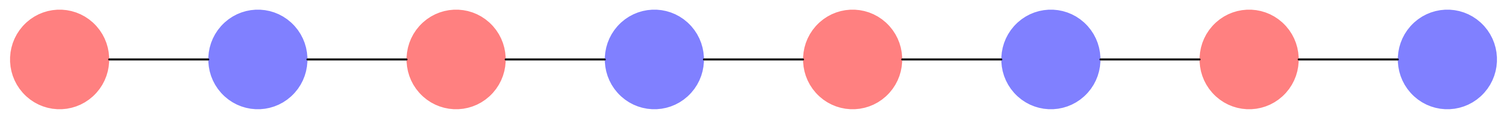


- Coloring trees
  - **3** coloring trees can be done in **O(log n)** rounds

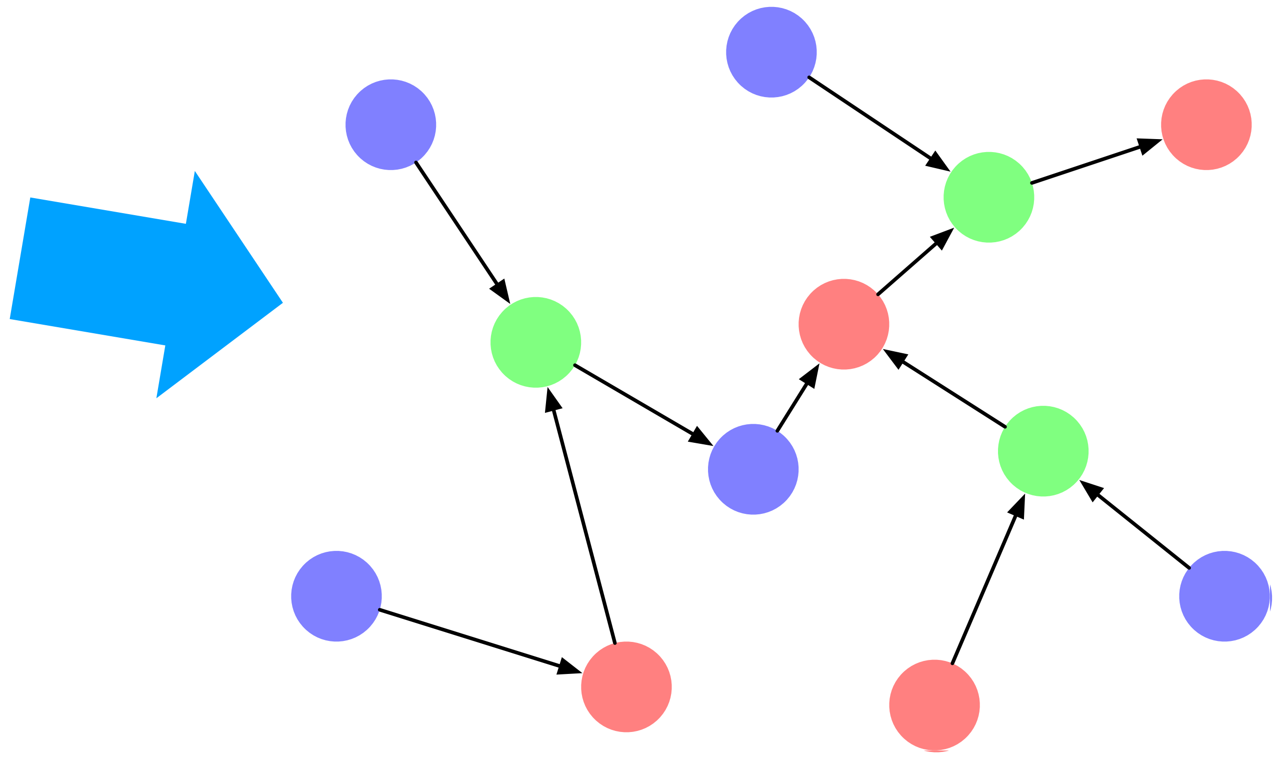


Lower bounds

- **2** coloring
  - can be solved in **O(n)** rounds
  - requires **Ω(n)** rounds

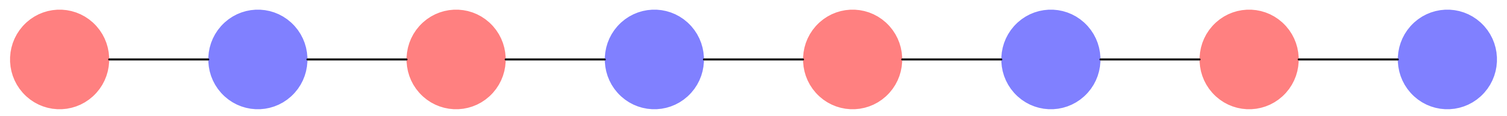


- Coloring trees
  - **3** coloring trees can be done in **O(log n)** rounds
  - **3** coloring **rooted** trees can be done in **O(log* n)** rounds

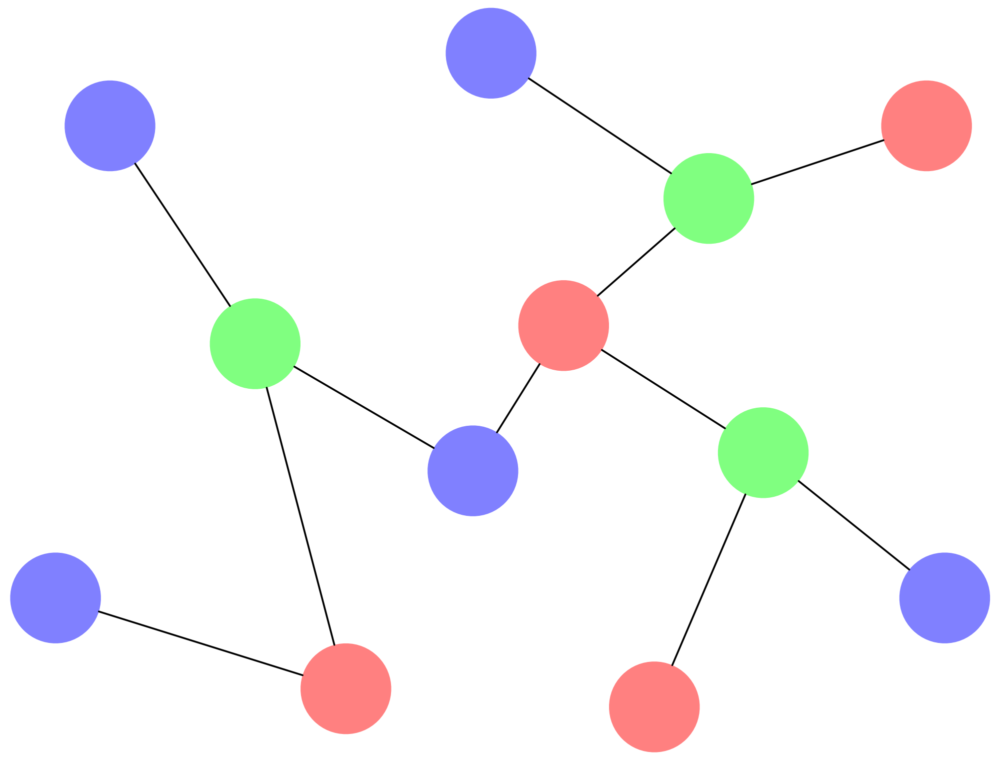


Lower bounds

- 2 coloring
  - can be solved in $O(n)$ rounds
  - requires $\Omega(n)$ rounds

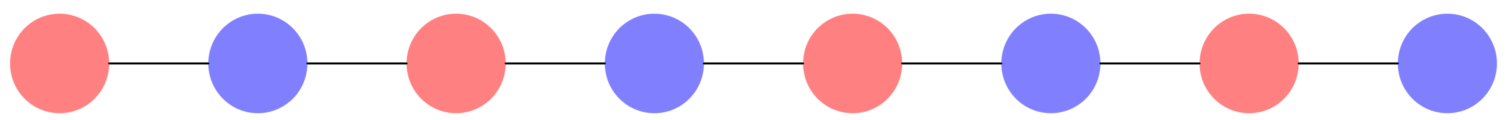


- Coloring trees
  - 3 coloring trees can be done in $O(\log n)$ rounds
  - 3 coloring **rooted** trees can be done in $O(\log^* n)$ rounds
  - $o(\Delta / \log \Delta)$ coloring trees of maximum degree Δ requires $\Omega(\log_\Delta n)$ rounds

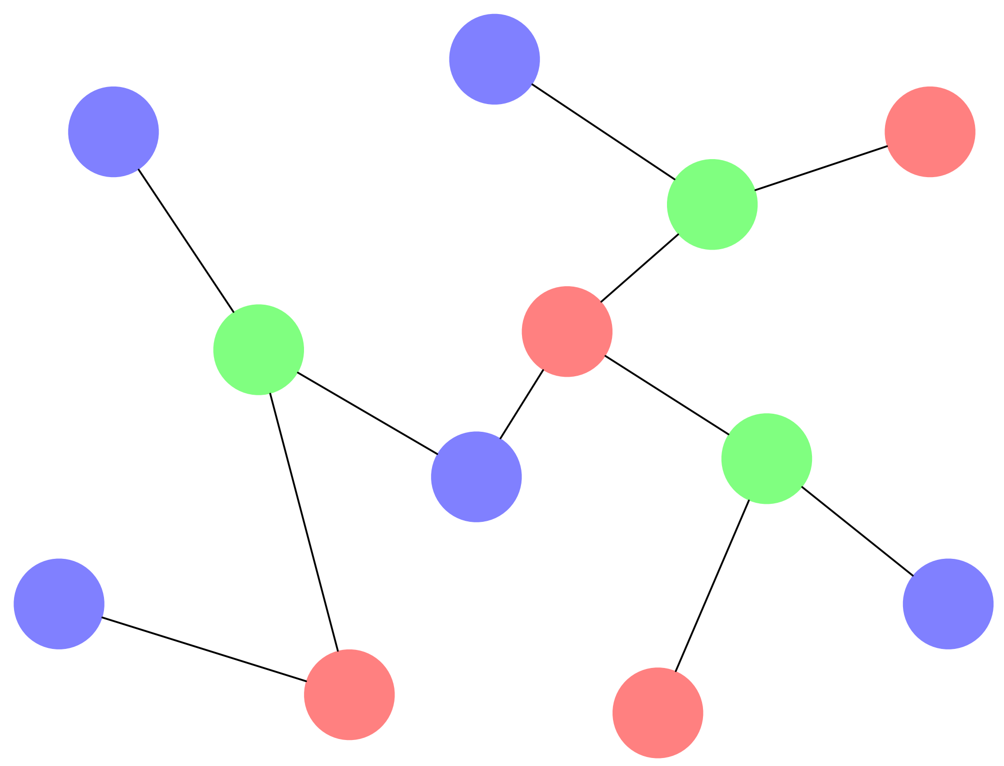


Lower bounds

- **2** coloring
  - can be solved in **O(n)** rounds
  - requires **Ω(n)** rounds

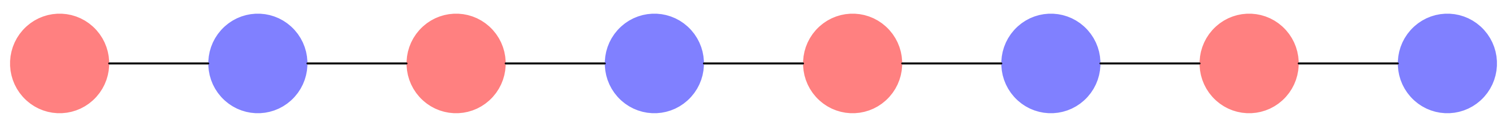


- Coloring trees
  - **3** coloring trees can be done in **O(log n)** rounds
  - **3** coloring **rooted** trees can be done in **O(log* n)** rounds
  - **o(Δ / log Δ)** coloring trees of maximum degree **Δ** requires **$\Omega(\log_{\Delta} n)$** rounds

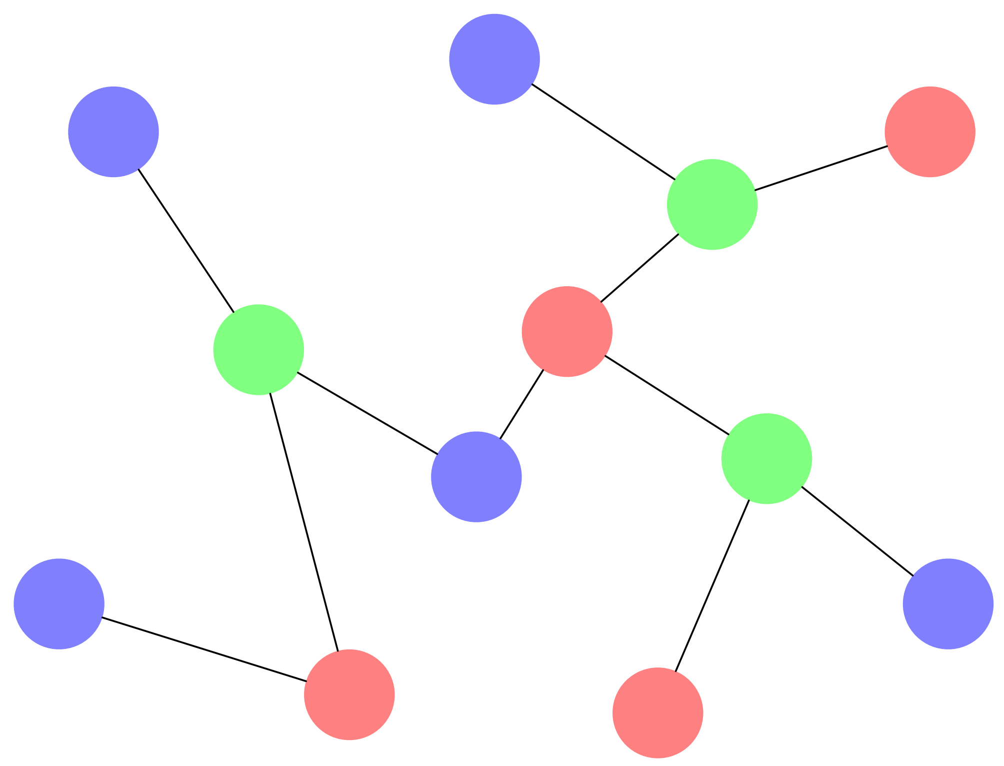


Lower bounds

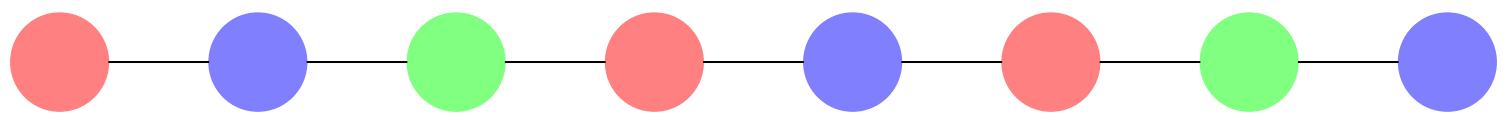
- 2 coloring
  - can be solved in $O(n)$ rounds
  - requires $\Omega(n)$ rounds



- Coloring trees
  - 3 coloring trees can be done in $O(\log n)$ rounds
  - 3 coloring **rooted** trees can be done in $O(\log^* n)$ rounds
  - $o(\Delta / \log \Delta)$ coloring trees of maximum degree Δ requires $\Omega(\log_\Delta n)$ rounds

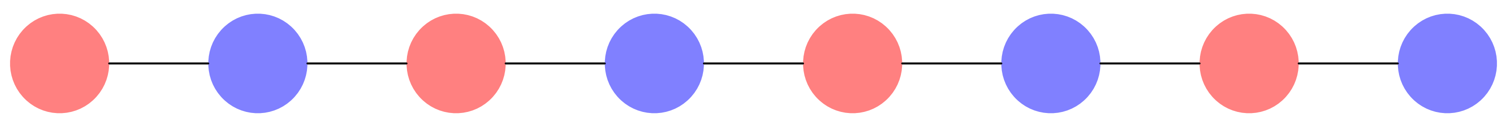


- 3 coloring paths or cycles

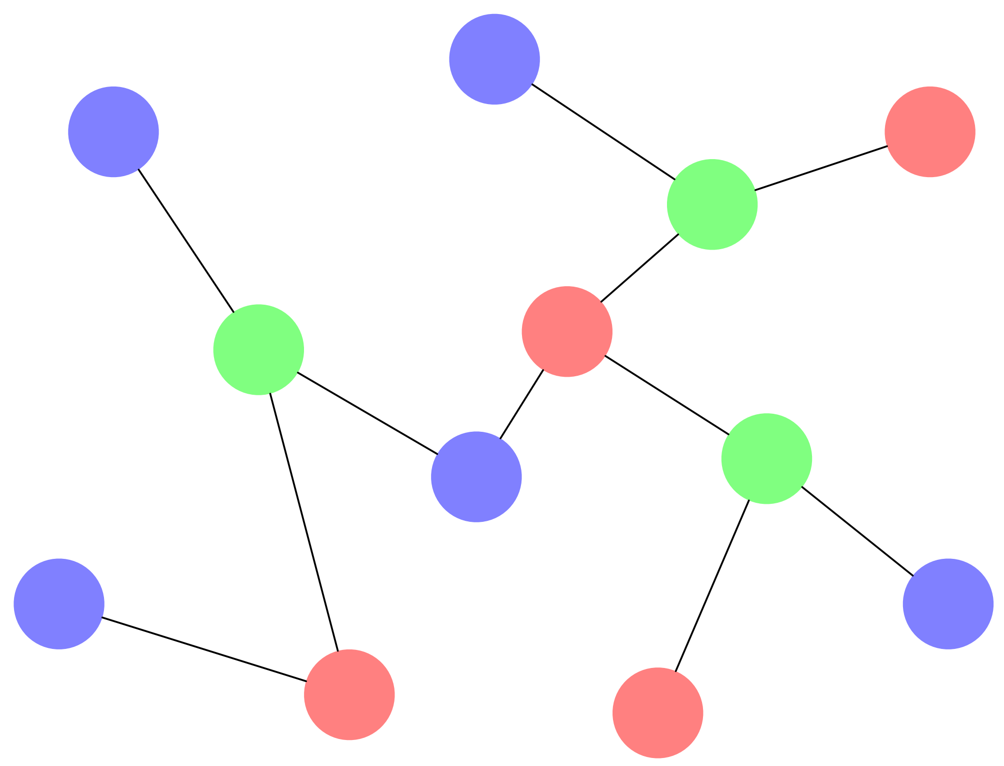


Lower bounds

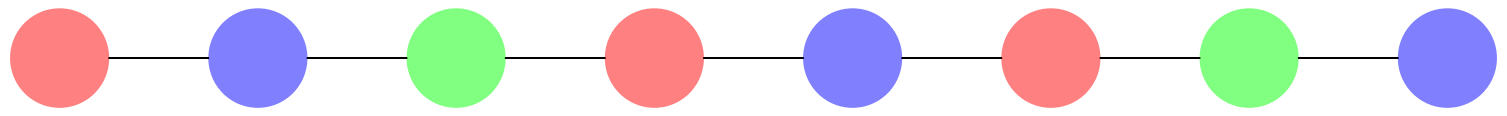
- 2 coloring
  - can be solved in O(n) rounds
  - requires Ω(n) rounds



- Coloring trees
  - 3 coloring trees can be done in O(log n) rounds
  - 3 coloring **rooted** trees can be done in O(log* n) rounds
  - o(Δ / log Δ) coloring trees of maximum degree Δ requires $\Omega(\log_\Delta n)$ rounds

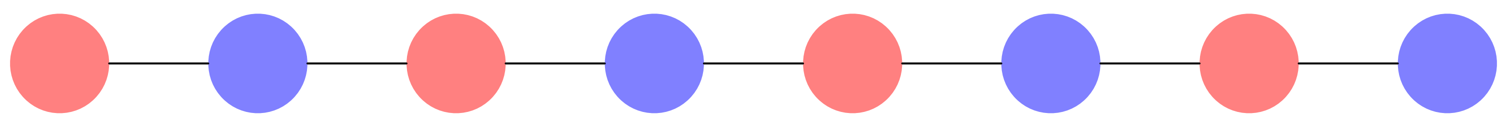


- 3 coloring paths or cycles
  - can be solved in O(log* n) rounds

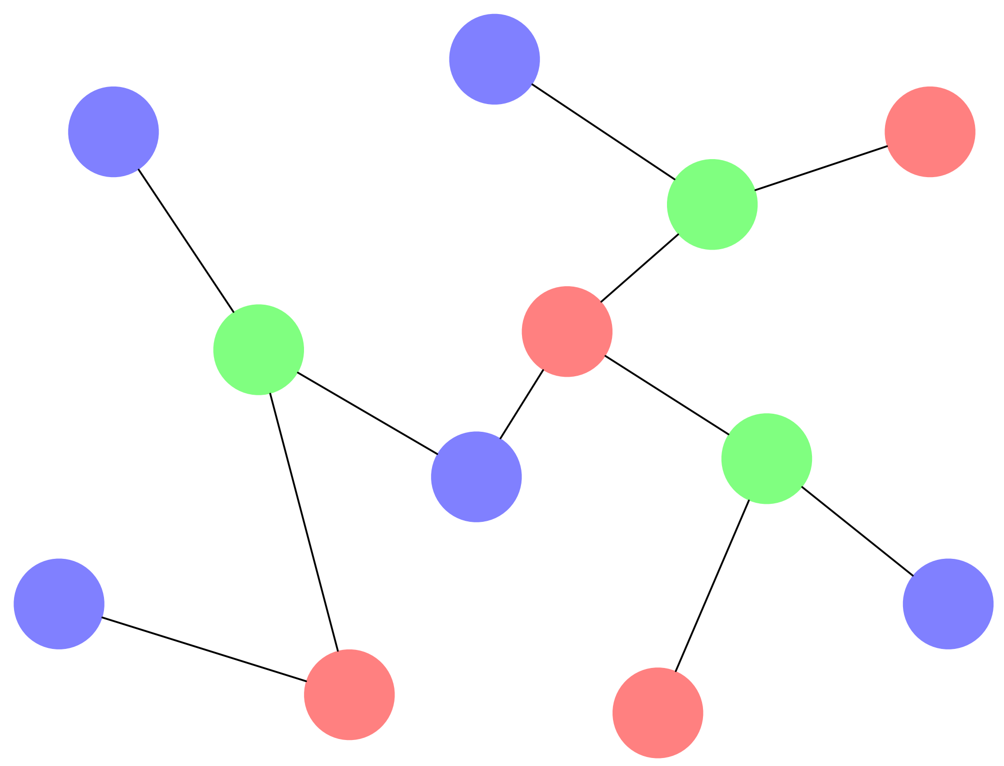


Lower bounds

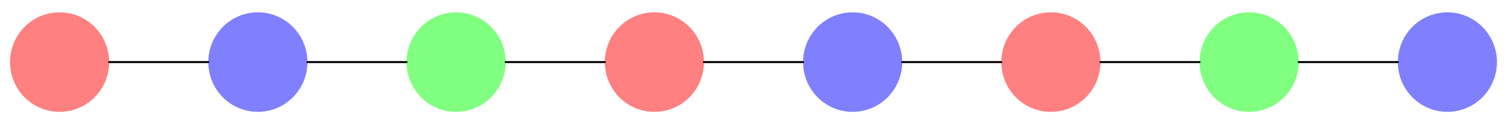
- 2 coloring
  - can be solved in $O(n)$ rounds
  - requires $\Omega(n)$ rounds



- Coloring trees
  - 3 coloring trees can be done in $O(\log n)$ rounds
  - 3 coloring **rooted** trees can be done in $O(\log^* n)$ rounds
  - $o(\Delta / \log \Delta)$ coloring trees of maximum degree Δ requires $\Omega(\log_\Delta n)$ rounds

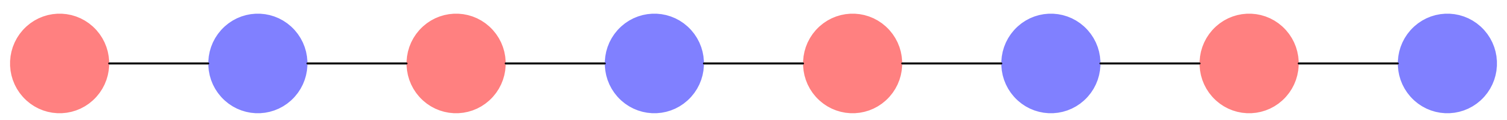


- 3 coloring paths or cycles
  - can be solved in $O(\log^* n)$ rounds
  - requires $\Omega(\log^* n)$ rounds

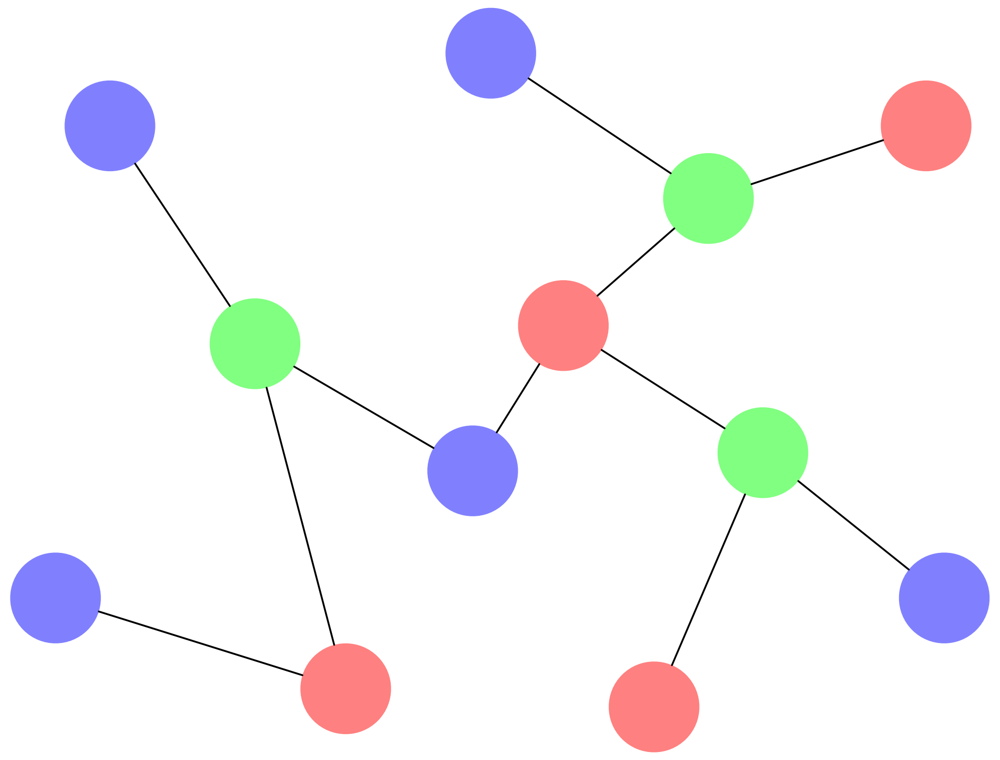


Lower bounds

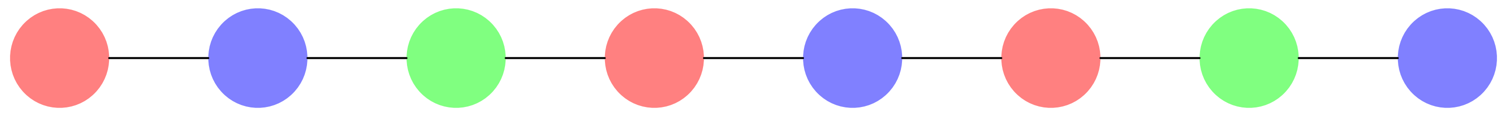
- **2** coloring
  - can be solved in **O(n)** rounds
  - requires **Ω(n)** rounds



- Coloring trees
  - **3** coloring trees can be done in **O(log n)** rounds
  - **3** coloring **rooted** trees can be done in **O(log* n)** rounds
  - **o(Δ / log Δ)** coloring trees of maximum degree **Δ** requires **$\Omega(\log_{\Delta} n)$** rounds



- **3** coloring paths or cycles
  - can be solved in **O(log* n)** rounds
  - requires **Ω(log* n)** rounds



Locality

Locality

Concept that allows us to prove **lower bounds** even when:

Locality

Concept that allows us to prove **lower bounds** even when:

- **messages** can be arbitrarily **large**

Locality

Concept that allows us to prove **lower bounds** even when:

- **messages** can be arbitrarily **large**
- local **computation** is **unbounded**

Locality

Concept that allows us to prove **lower bounds** even when:

- **messages** can be arbitrarily **large**
- local **computation** is **unbounded**

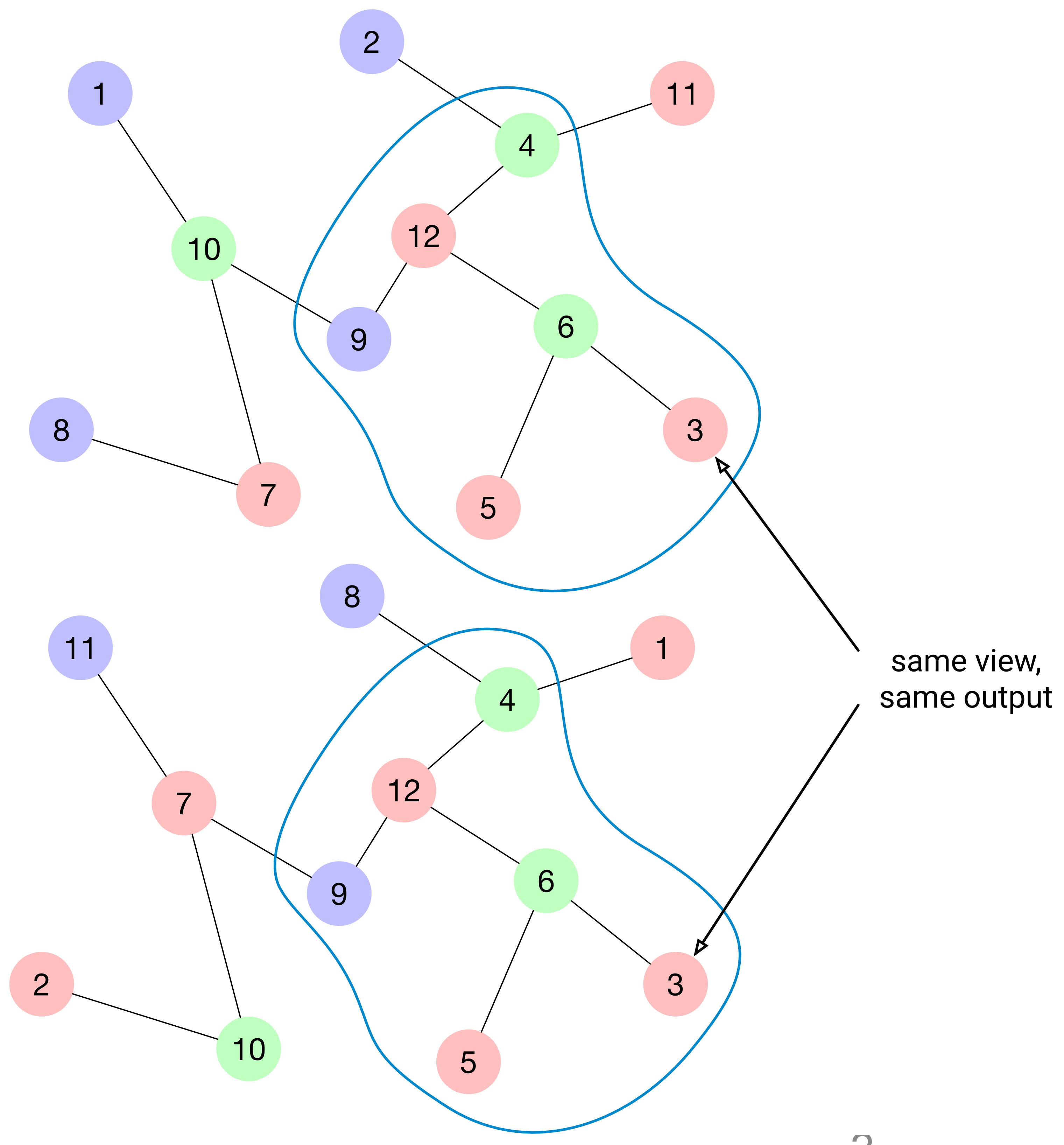
"same view = same output"

Locality

Concept that allows us to prove **lower bounds** even when:

- **messages** can be arbitrarily **large**
- local **computation** is **unbounded**

"same view = same output"



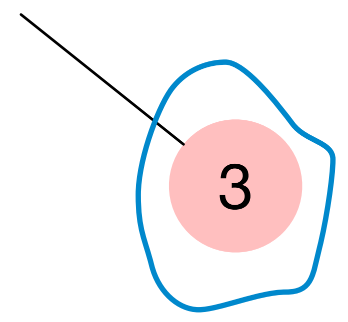
Locality

Locality

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

Locality

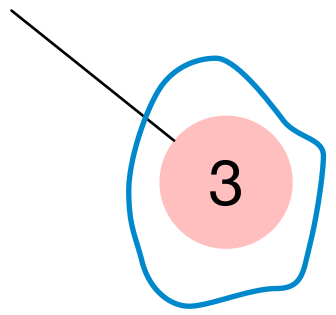
If the size of the messages and the local computation is unbounded, all synchronous **T**-round algorithms have a ***normal form***:



Locality

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

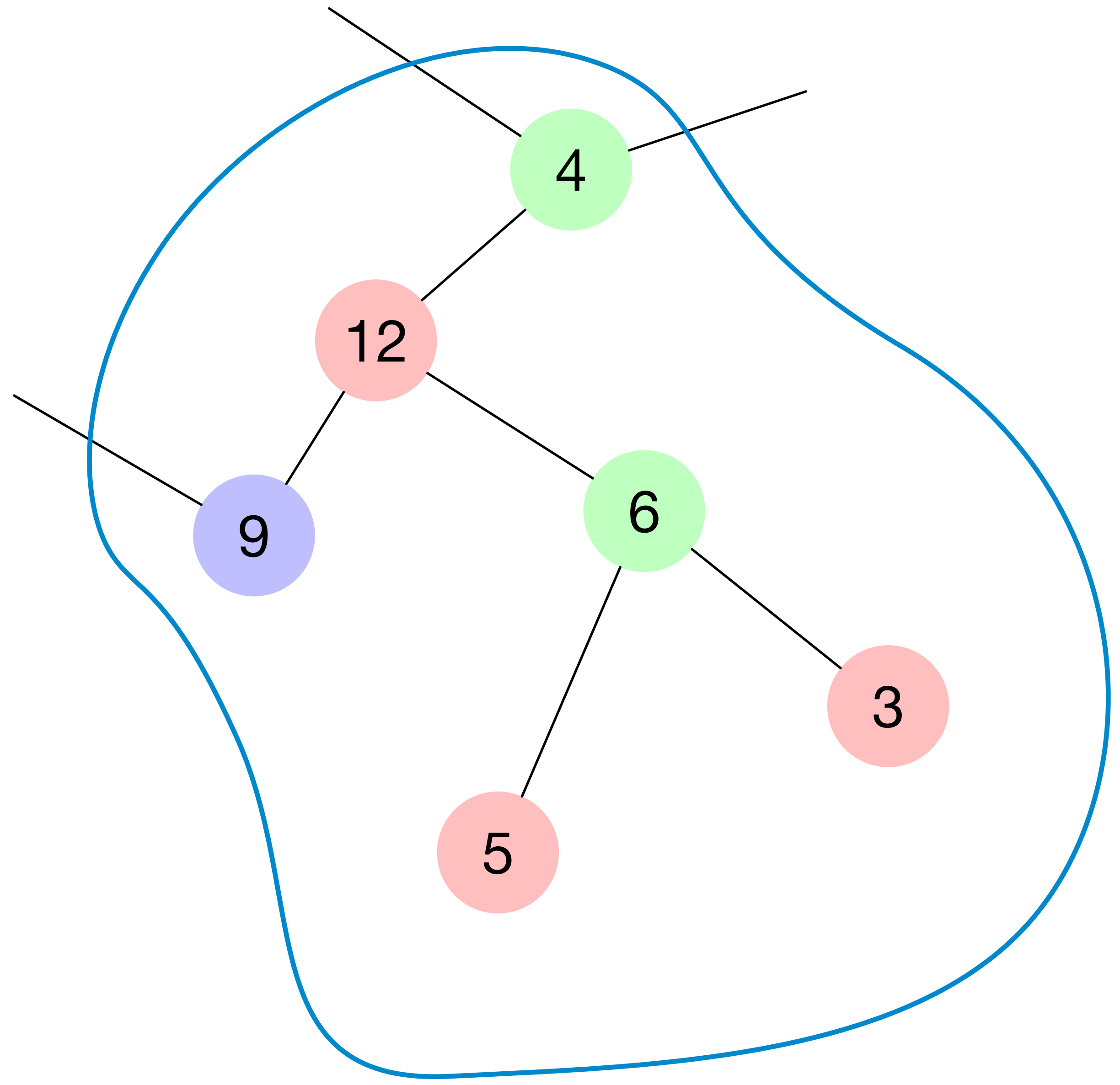
1. Gather the radius-T neighborhood



Locality

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

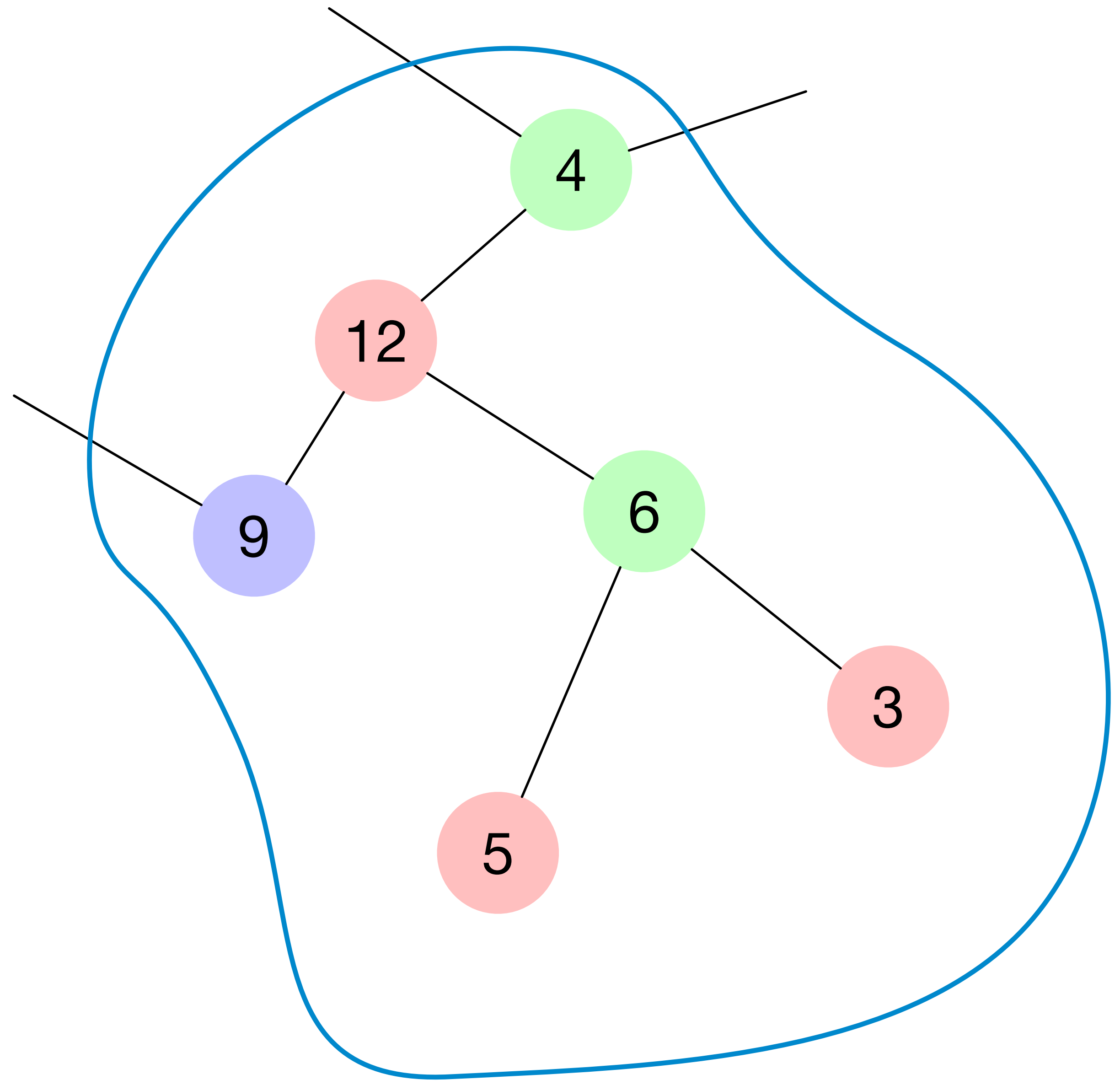
1. Gather the radius-T neighborhood



Locality

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

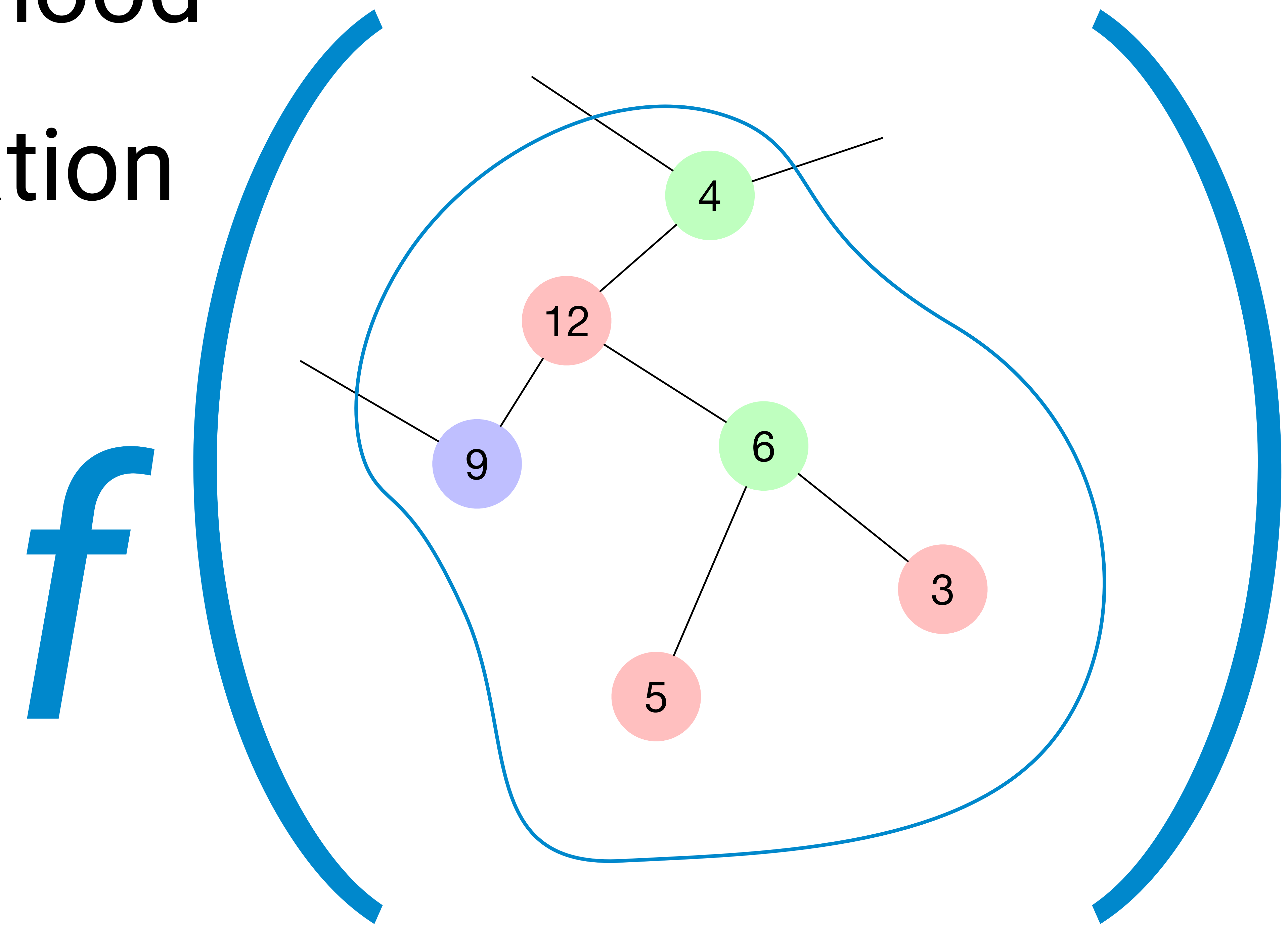
1. Gather the radius-T neighborhood
2. Perform some local computation



Locality

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

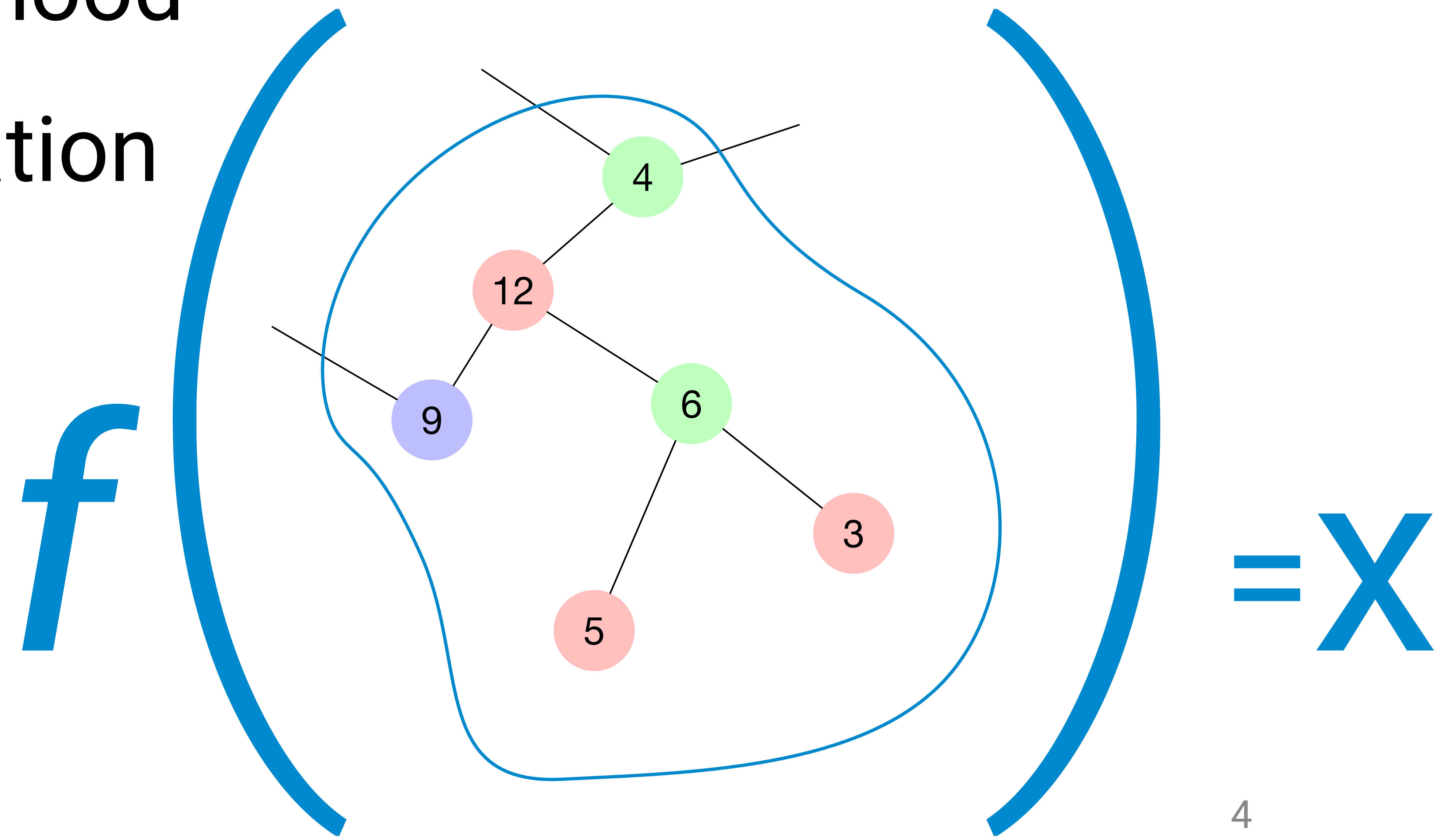
1. Gather the radius-T neighborhood
2. Perform some local computation



Locality

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

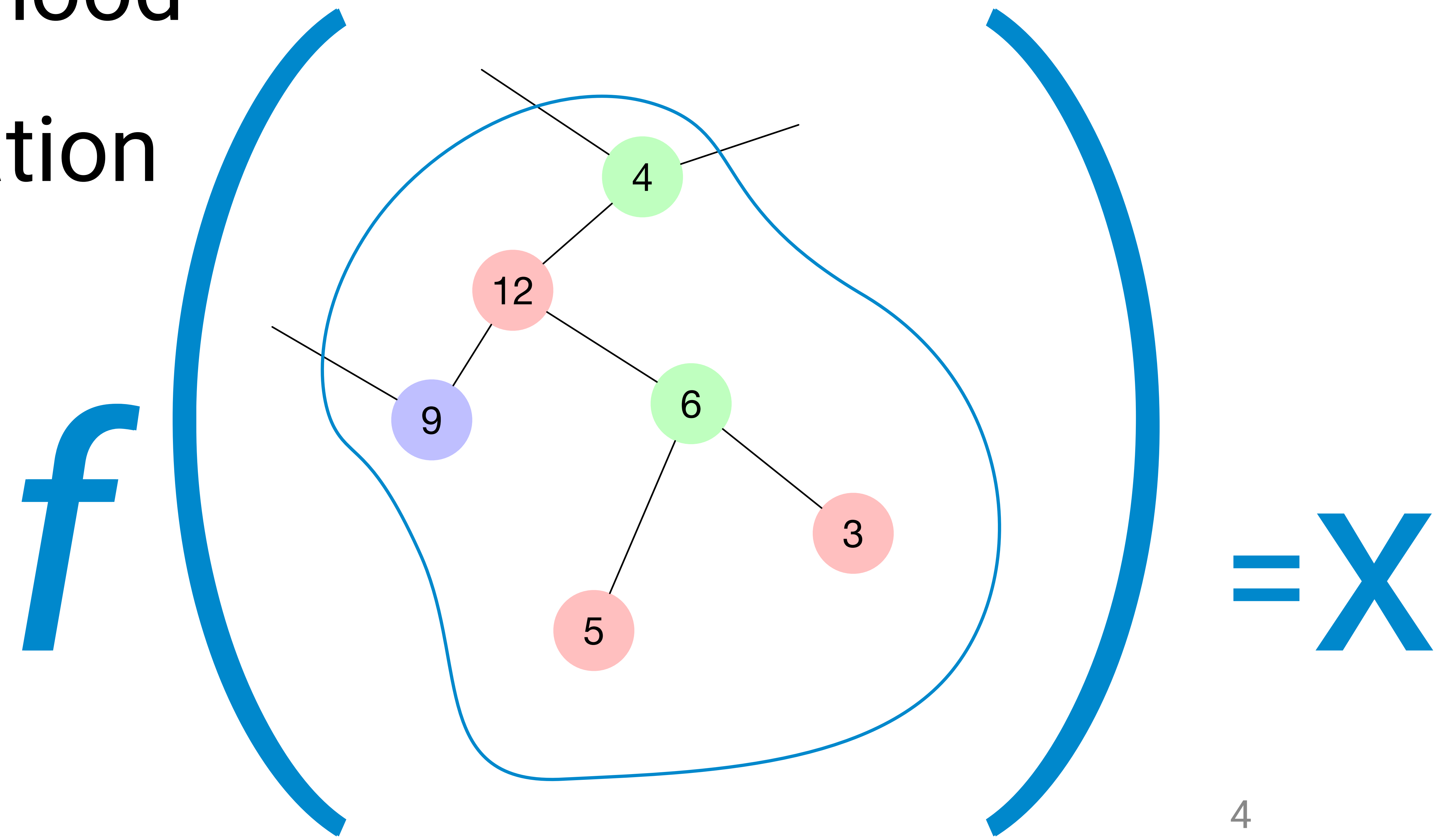
1. Gather the radius-T neighborhood
2. Perform some local computation



Locality

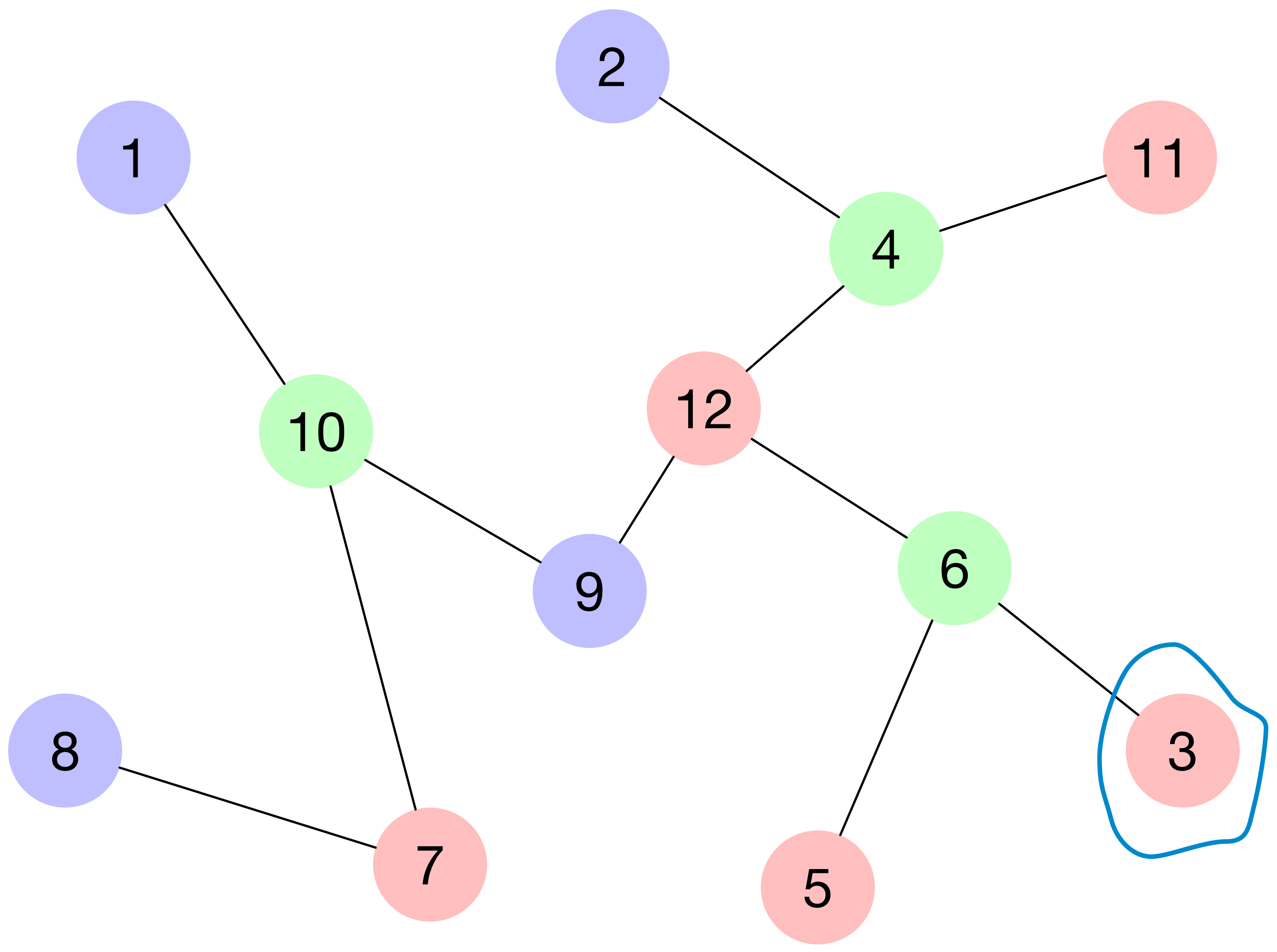
If the size of the messages and the local computation is unbounded, all synchronous T-round algorithms have a ***normal form***:

1. Gather the radius-T neighborhood
2. Perform some local computation
3. Output a result



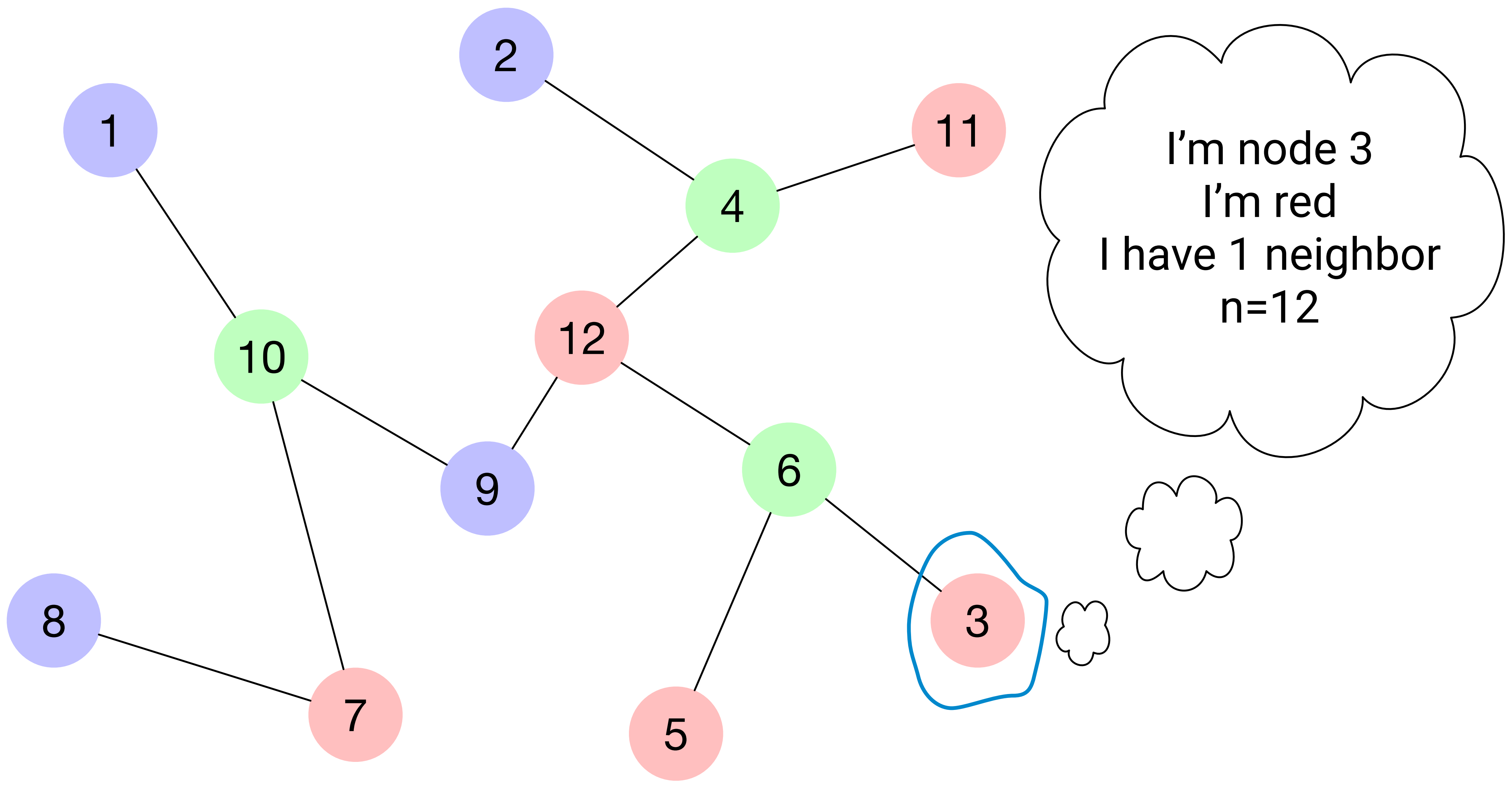
Locality (Example)

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



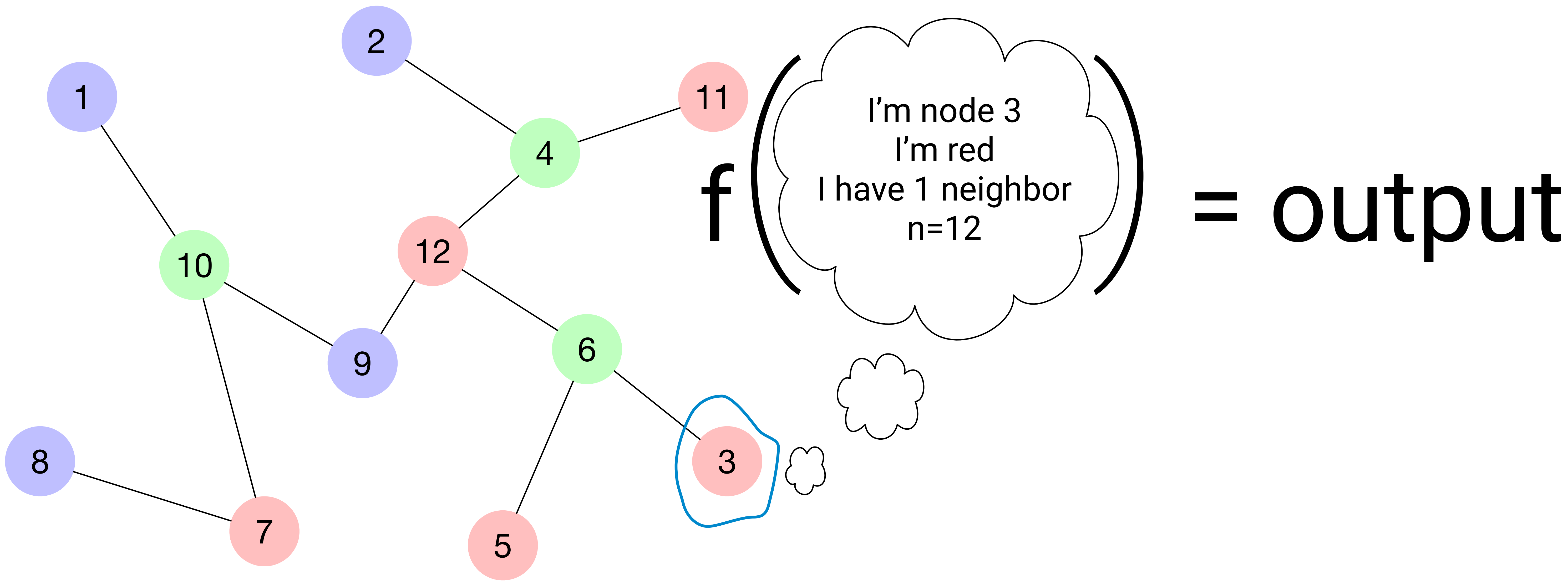
Locality (Example)

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

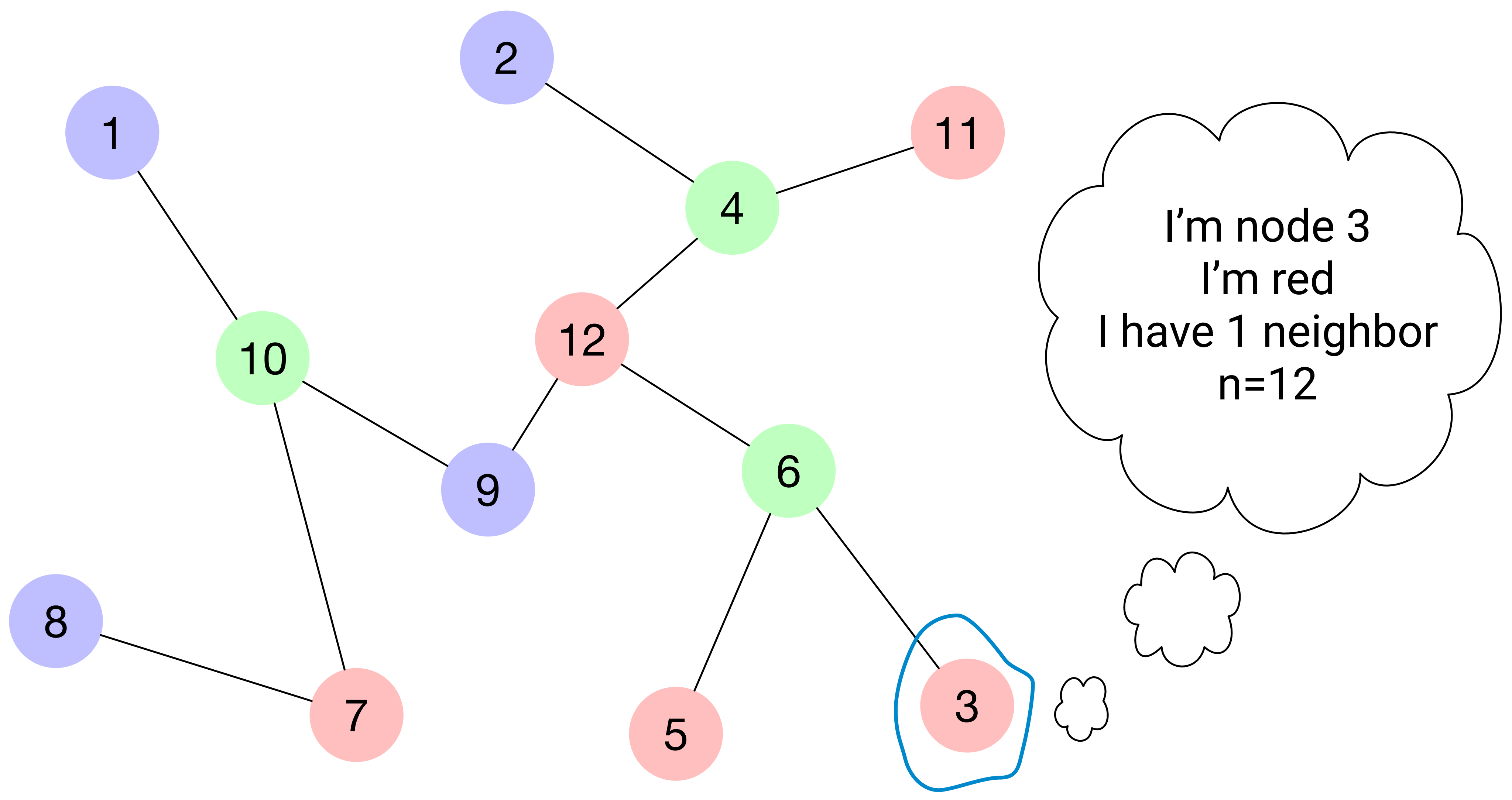


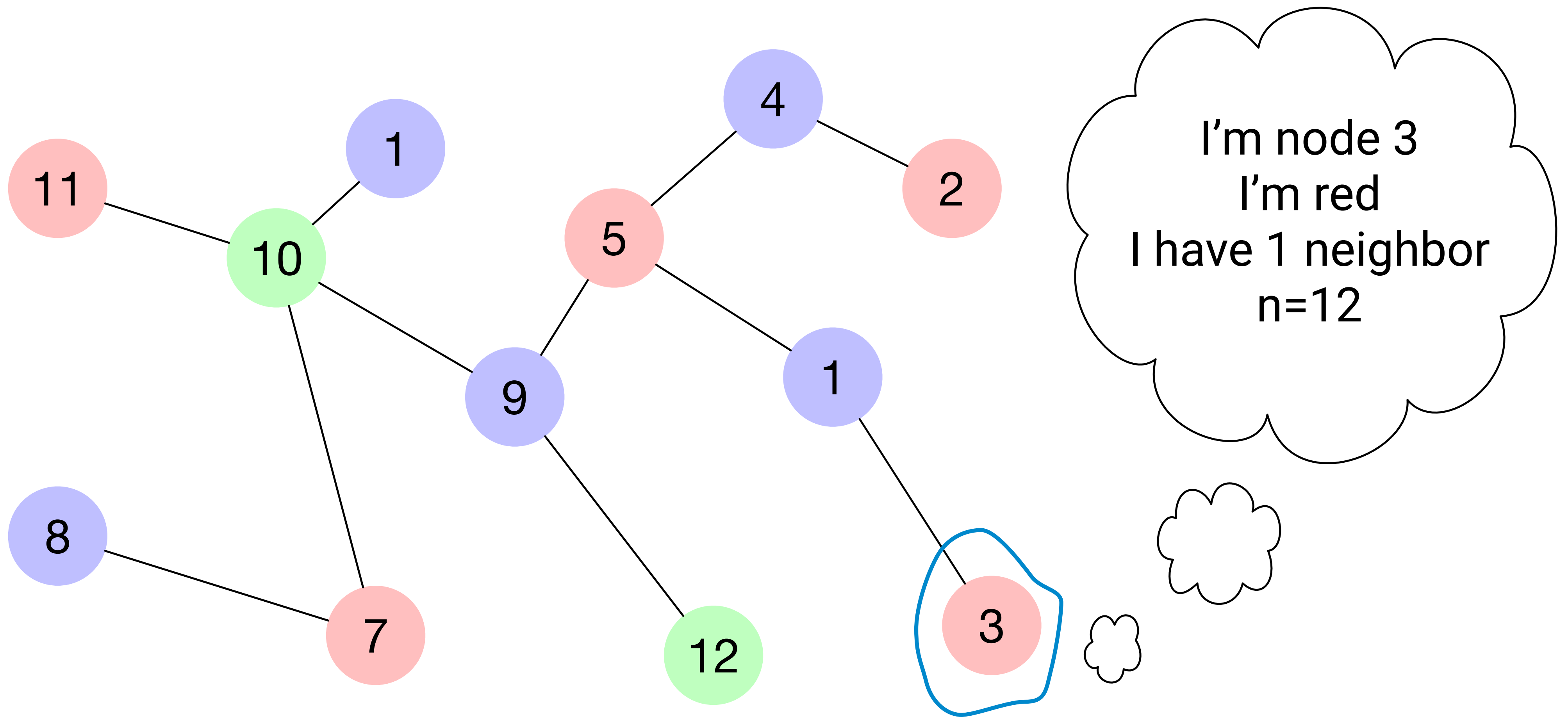
Locality (Example)

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

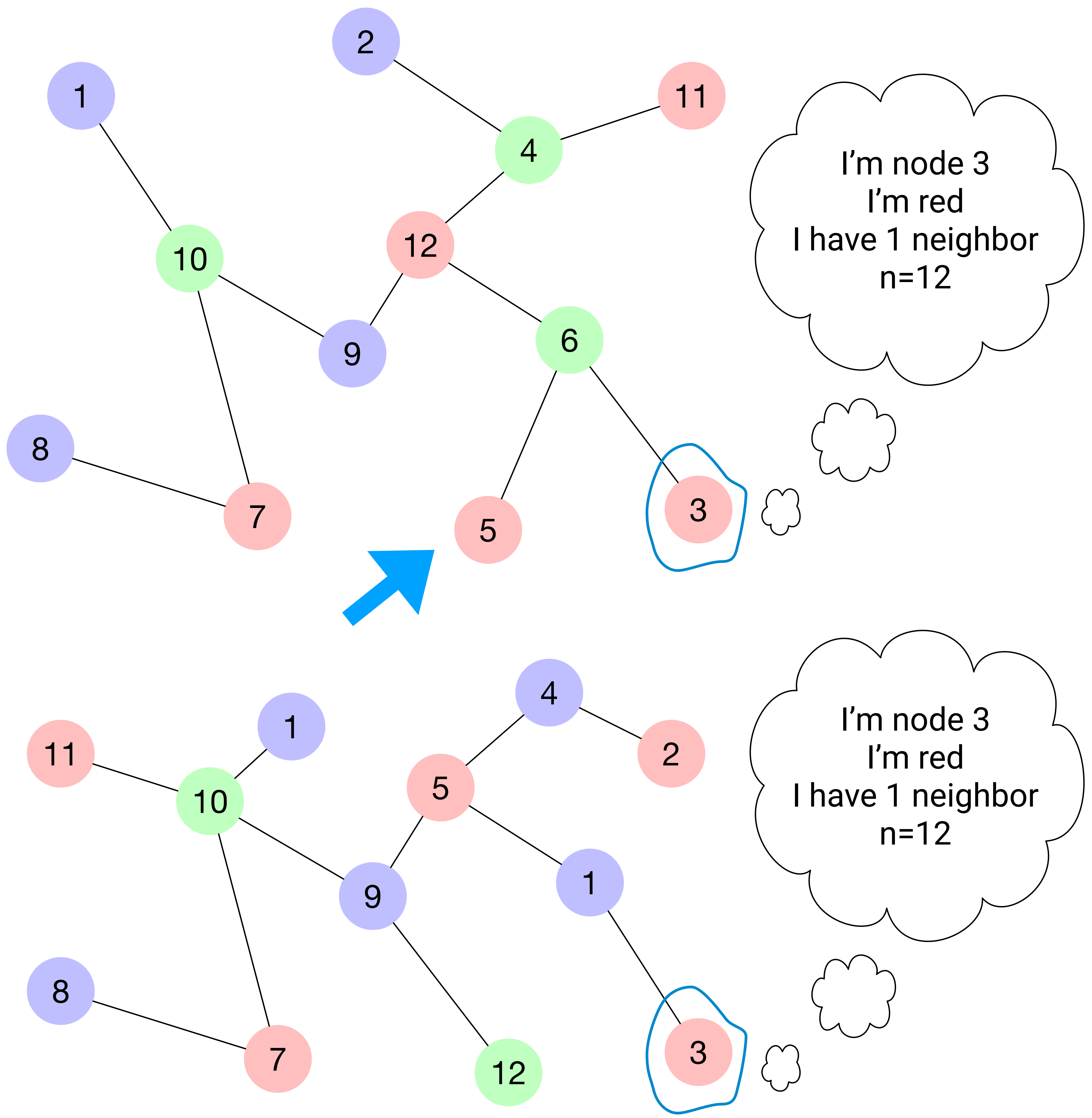


Locality (Example)

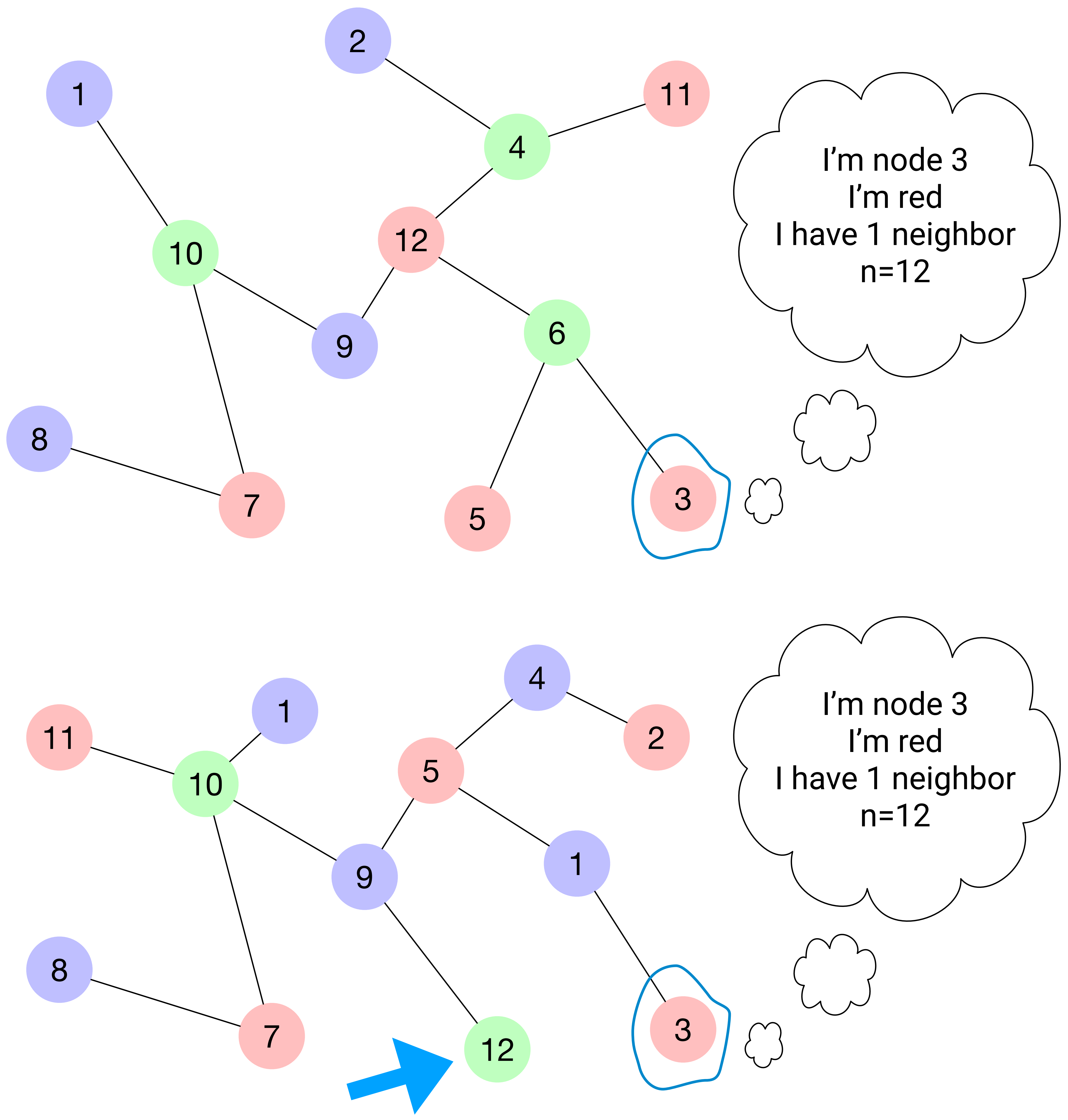




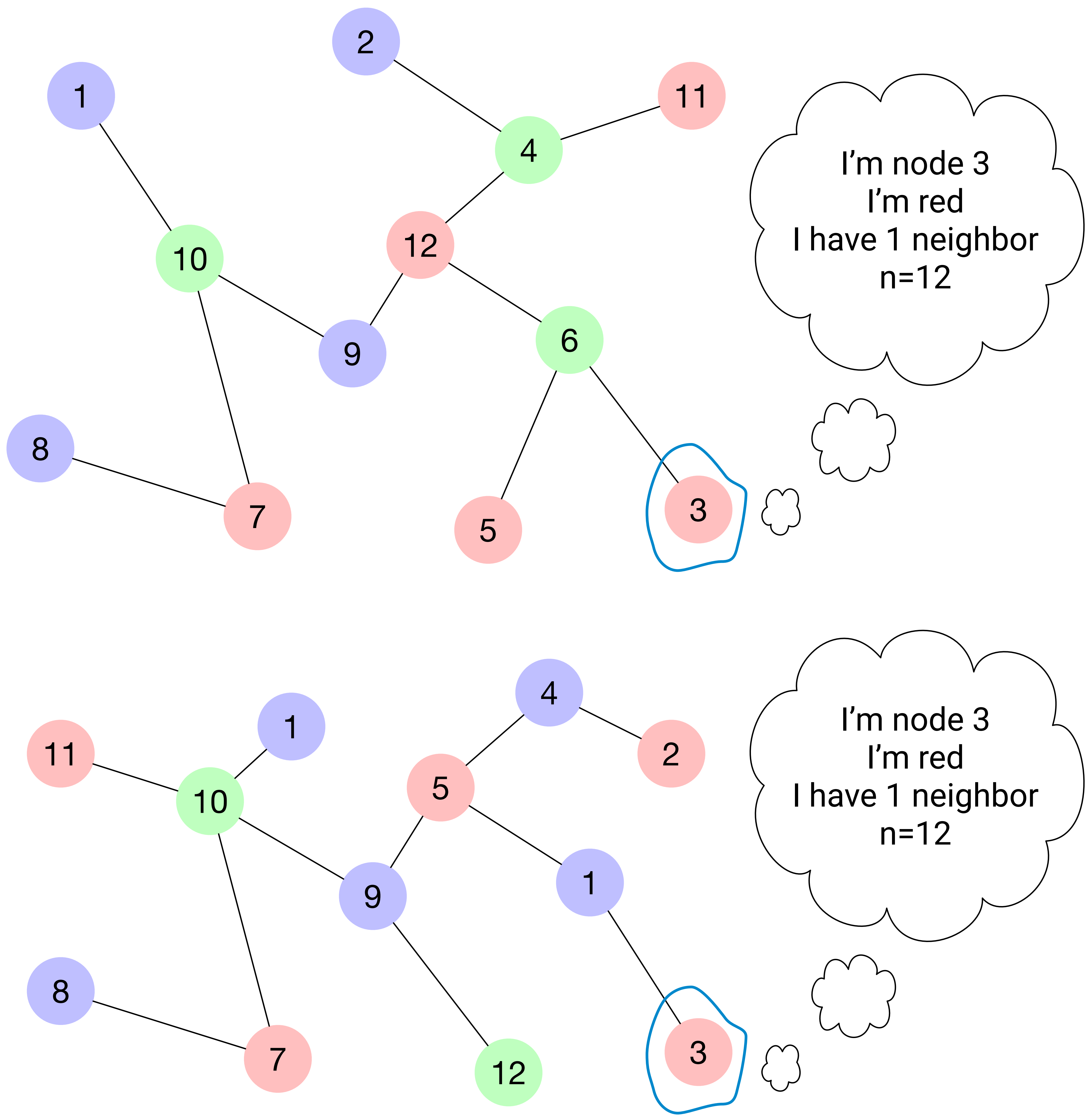
Locality (Example)



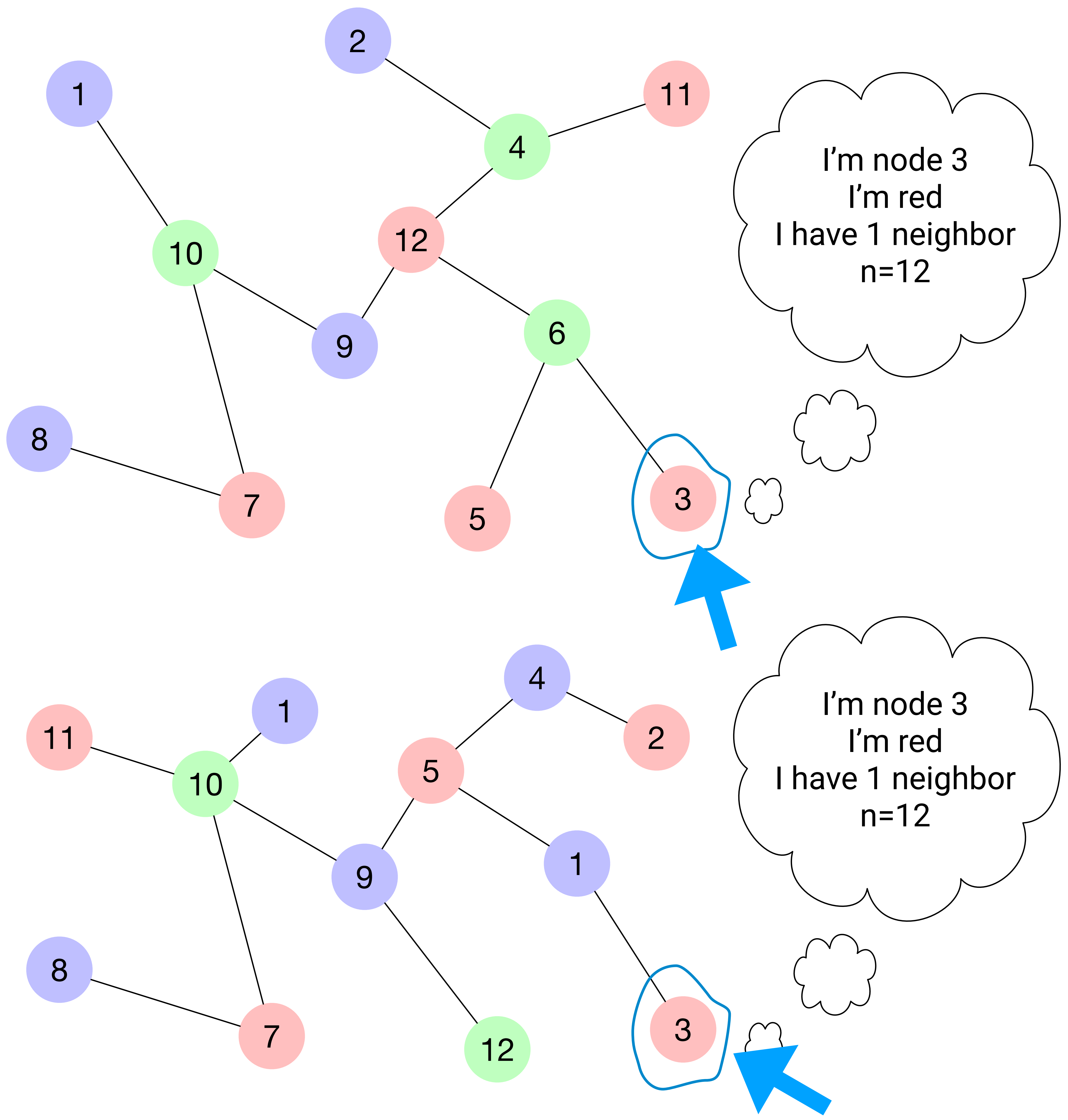
Locality (Example)



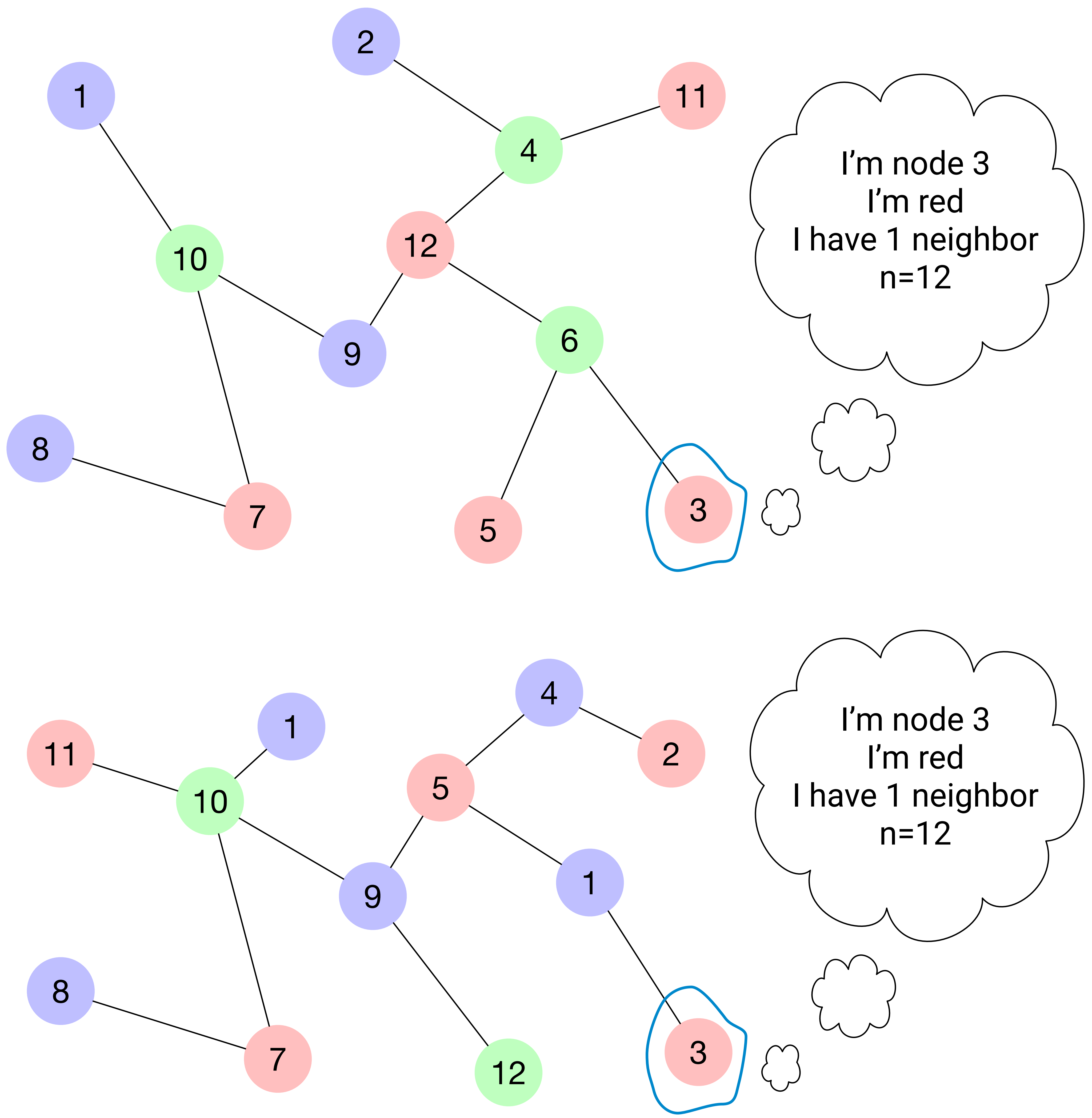
Locality (Example)



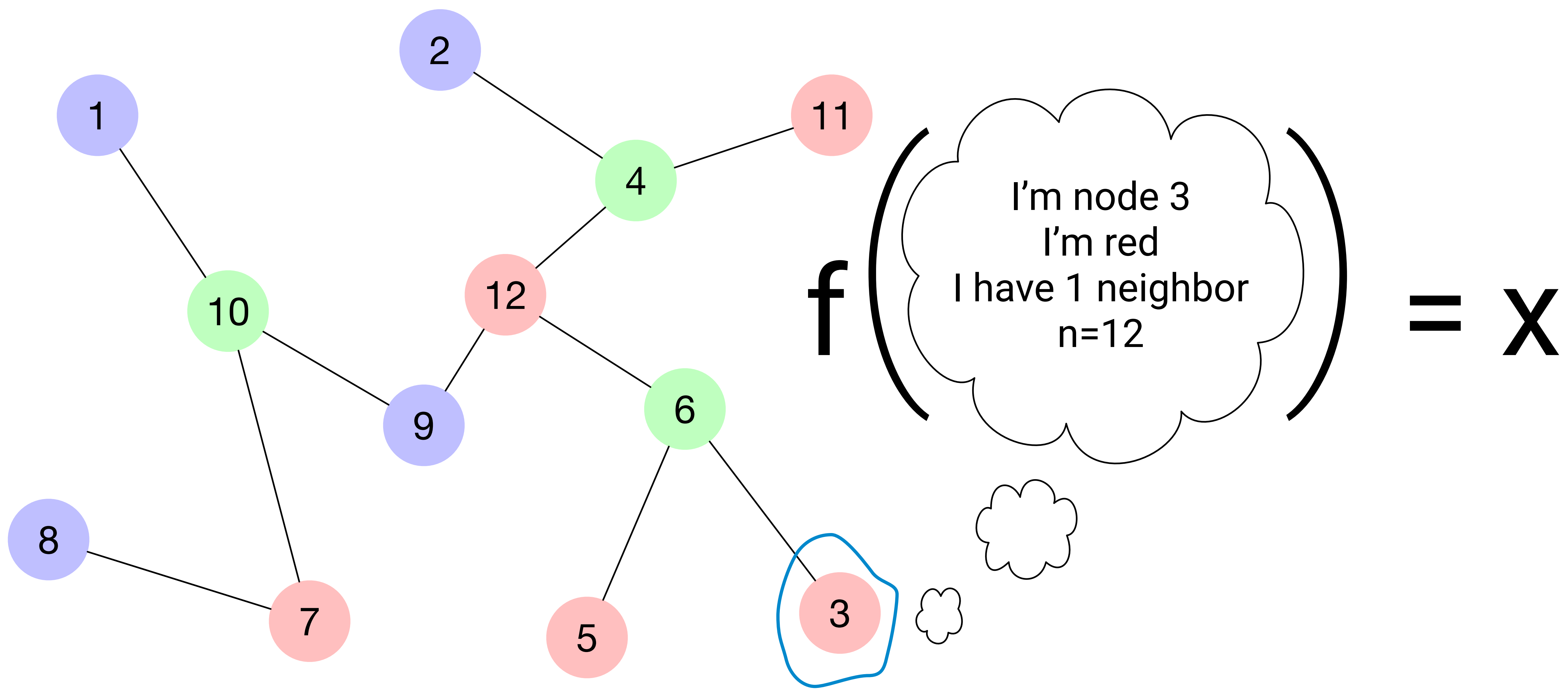
Locality (Example)

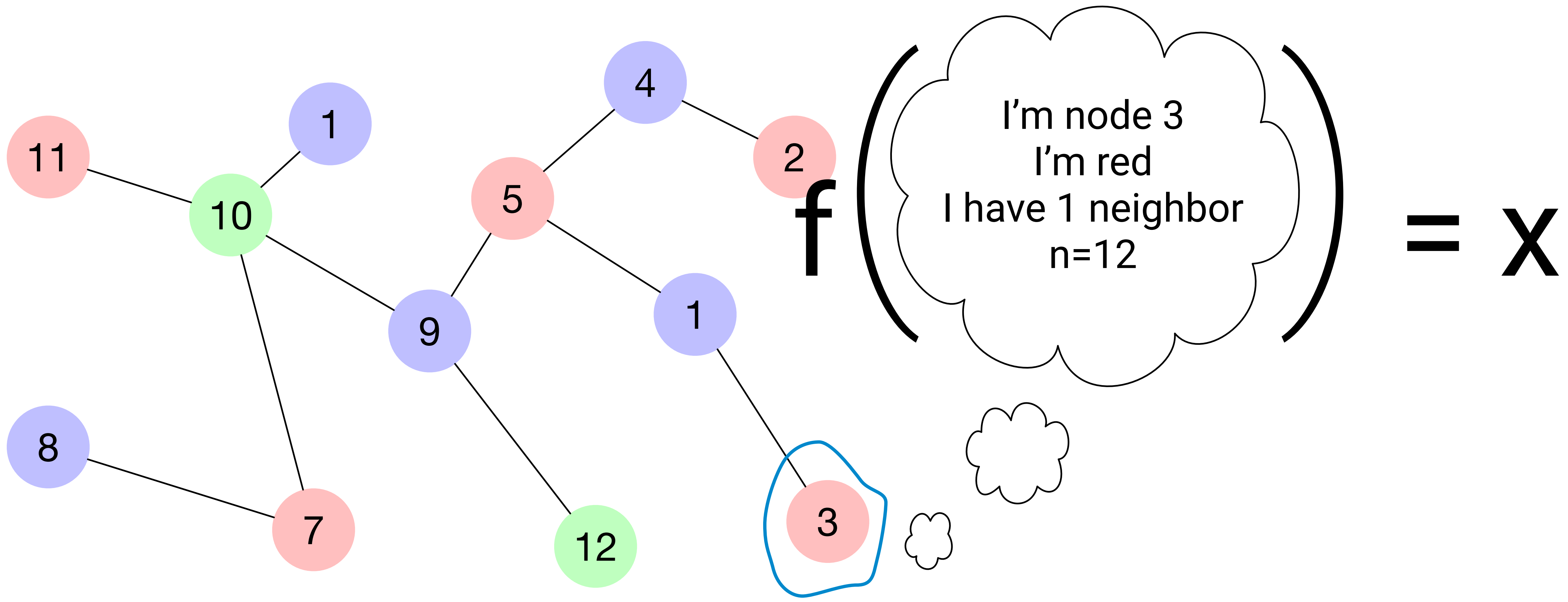


Locality (Example)

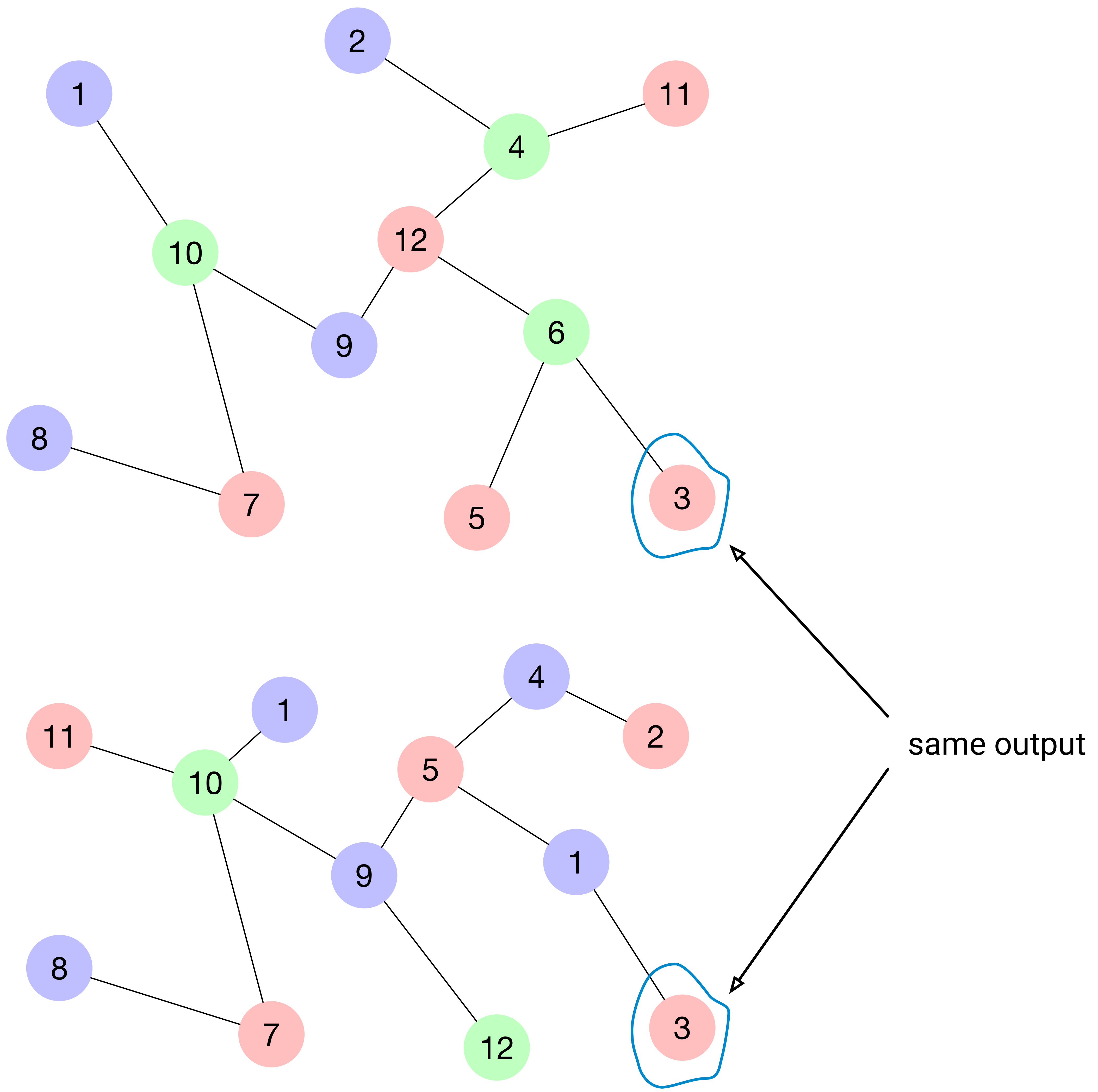


Locality (Example)



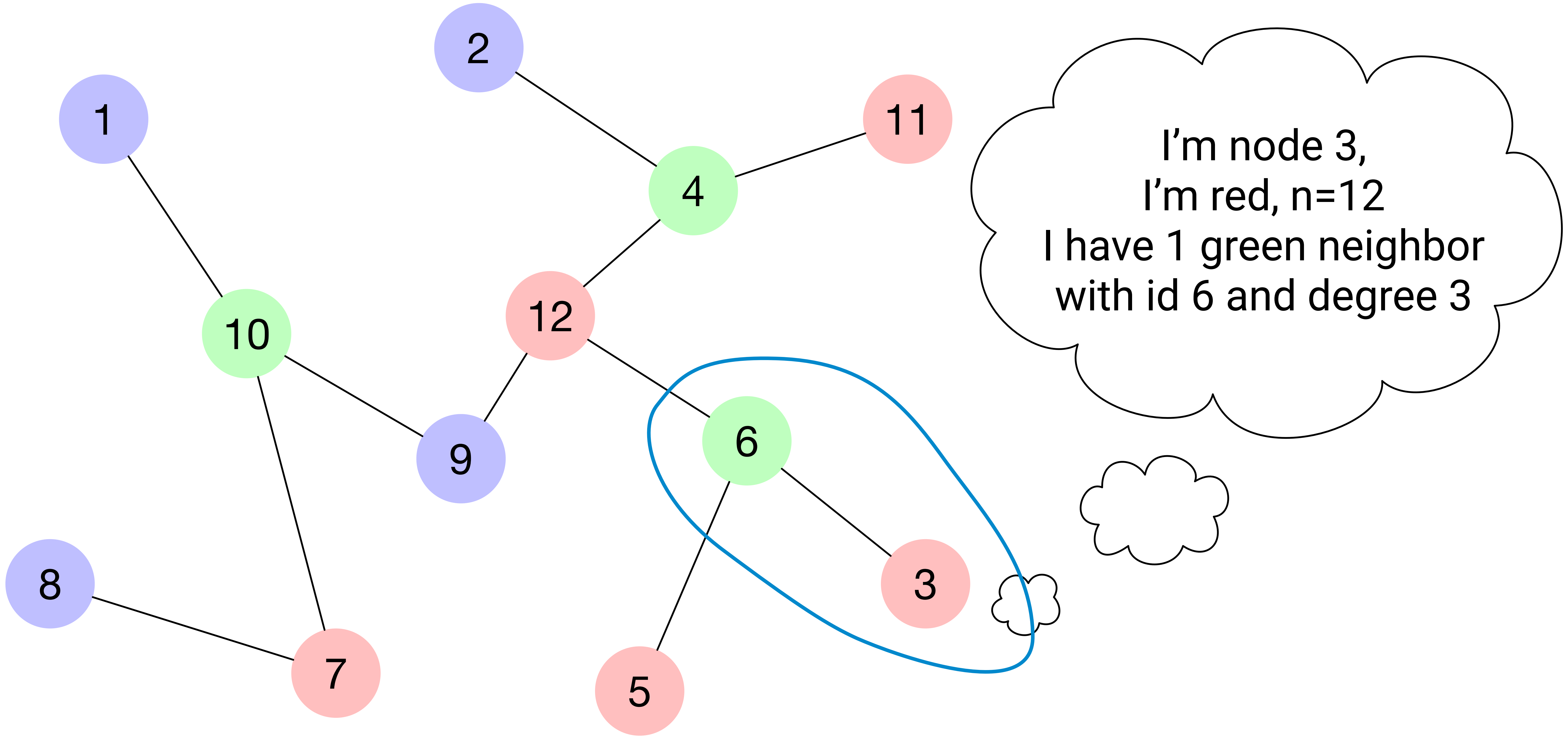


Locality (Example)



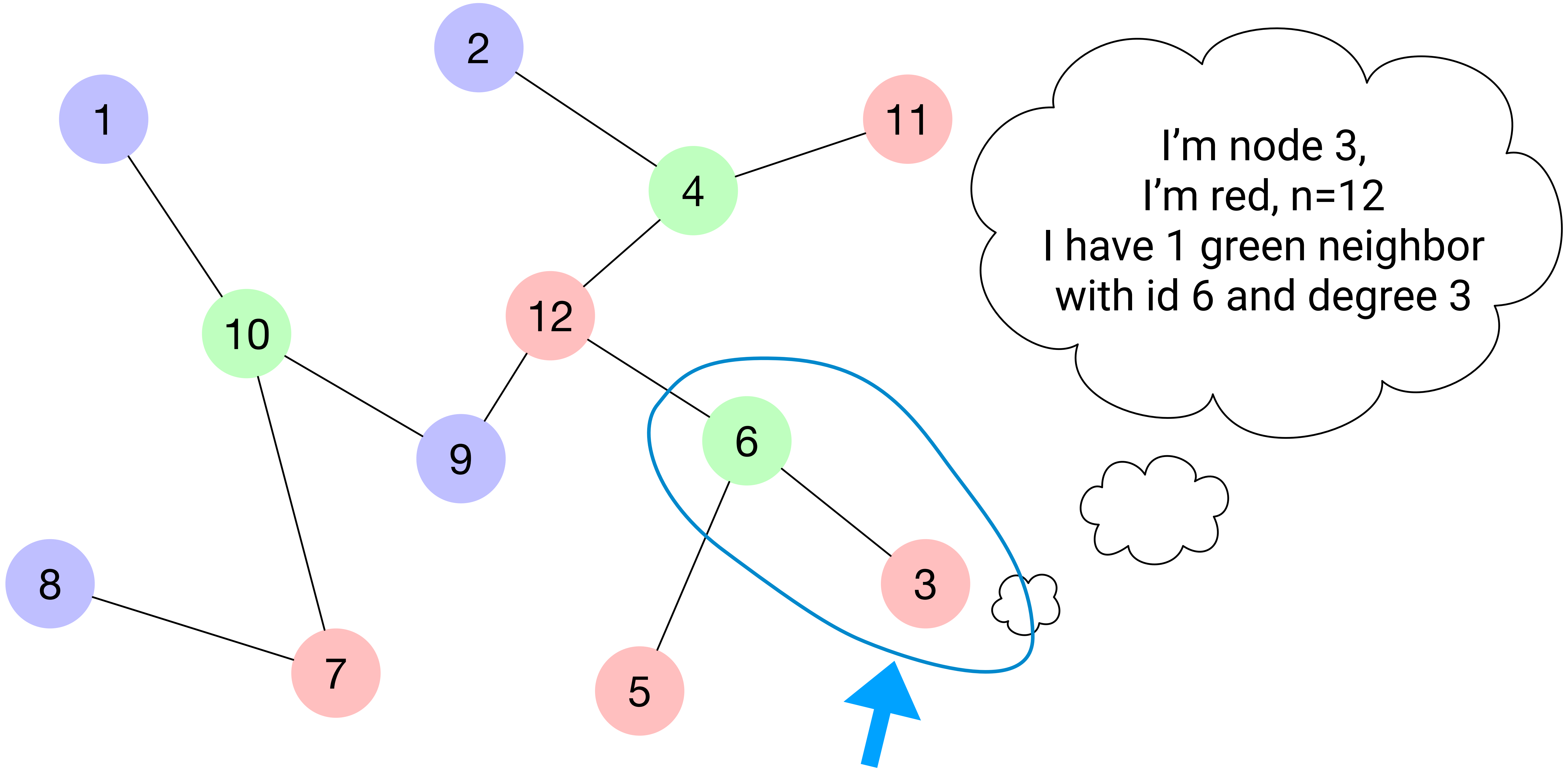
Locality (Example)

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



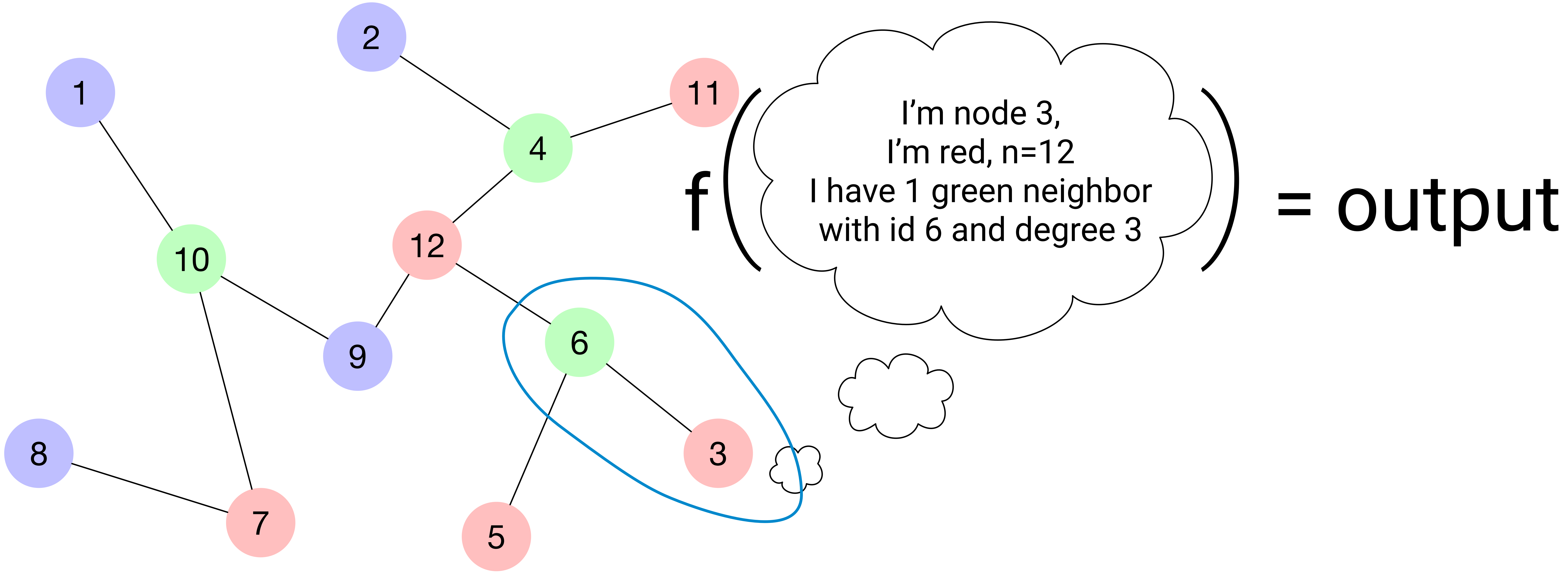
Locality (Example)

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



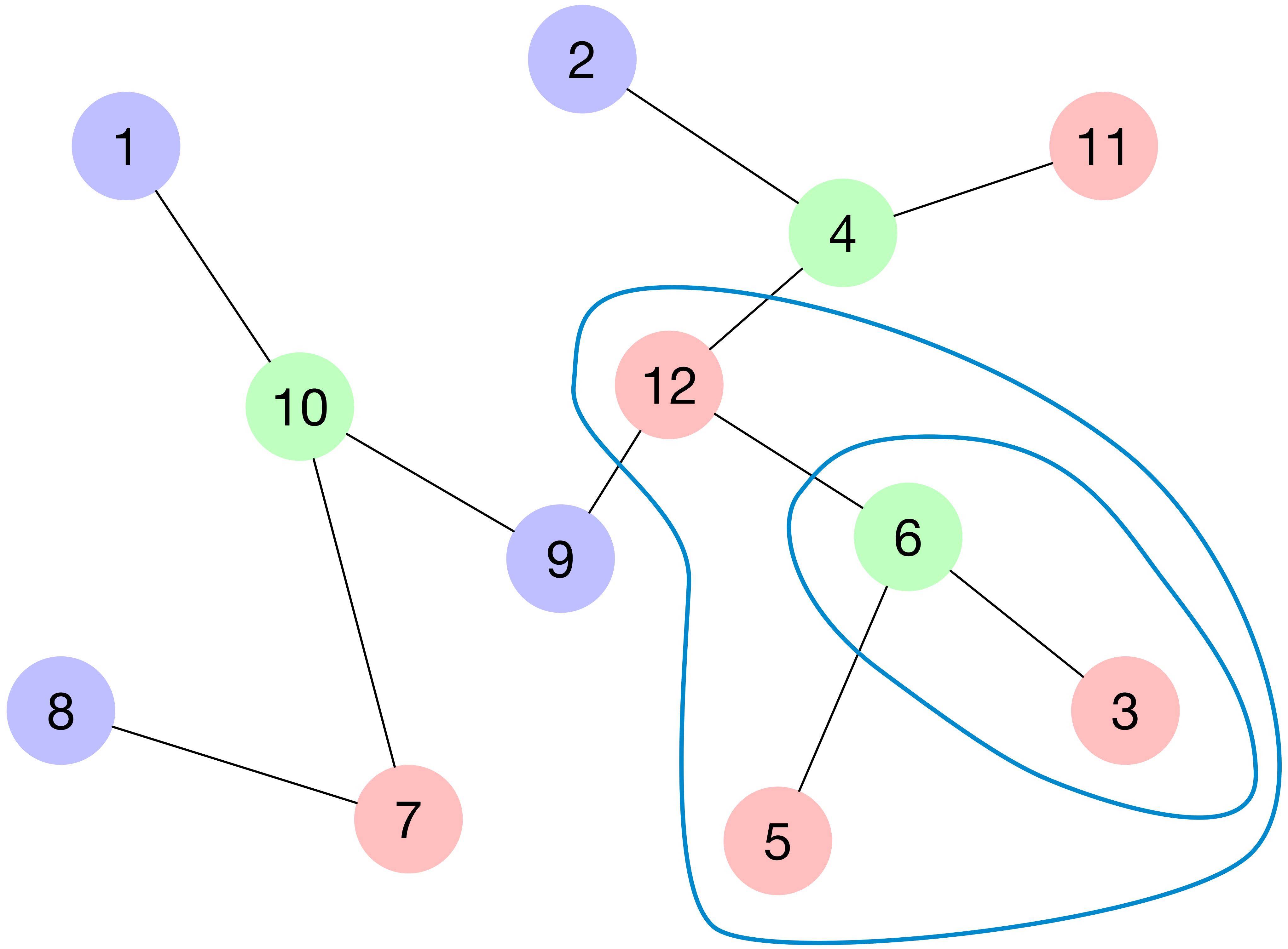
Locality (Example)

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



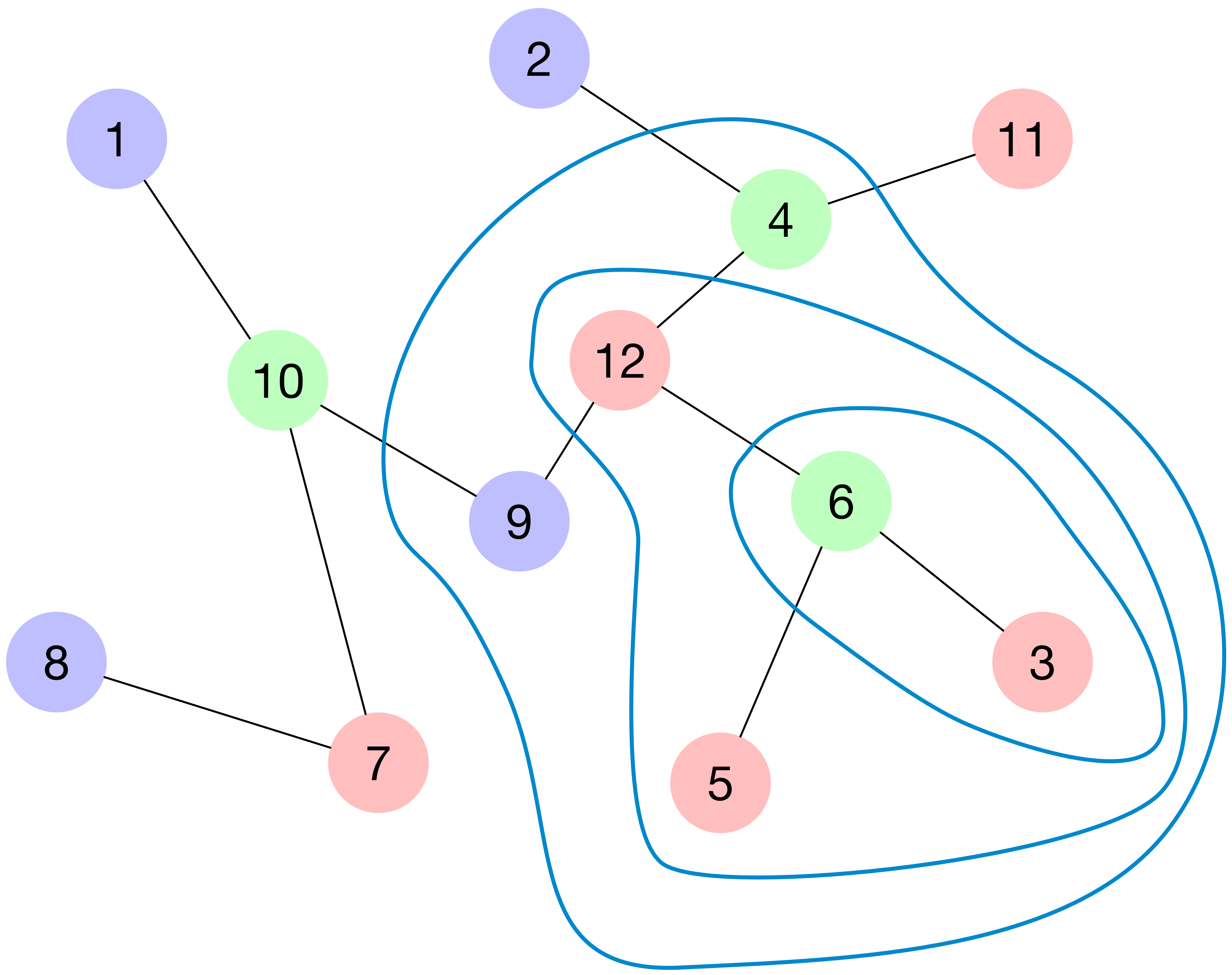
Locality (Example)

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 by induction.

Locality

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

Proof by induction.

- **T = 0.** trivial

Locality

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

Proof by induction.

- **T = 0.** trivial
- **T > 0.**

Locality

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

Proof by induction.

- **T = 0.** trivial
- **T > 0.**
  - assume: after **T-1** rounds, the **state** of a node only depends on its **(T-1)-radius** neighborhood

Locality

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

Proof by induction.

- **T = 0.** trivial
- **T > 0.**
  - assume: after **T-1** rounds, the **state** of a node only depends on its **(T-1)-radius** neighborhood
  - prove: after **T** rounds, the **state** of a node only depends on its **T-radius** neighborhood.

Locality

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

Proof by induction.

- **T = 0.** trivial
- **T > 0.**
  - assume: after **T-1** rounds, the **state** of a node only depends on its **(T-1)-radius** neighborhood
  - prove: after **T** rounds, the **state** of a node only depends on its **T-radius** neighborhood.

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 by induction.

- **T = 0.** trivial
- **T > 0.**
  - assume: after **T-1** rounds, the **state** of a node only depends on its **(T-1)-radius** neighborhood
  - prove: after **T** rounds, the **state** of a node only depends on its **T-radius** neighborhood.

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 by induction.

- **T = 0.** trivial
- **T > 0.**
  - assume: after **T-1** rounds, the **state** of a node only depends on its **(T-1)-radius** neighborhood
  - prove: after **T** rounds, the **state** of a node only depends on its **T-radius** neighborhood.

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 by induction.

- T = 0. trivial
- T > 0.
  - assume: after T-1 rounds, the state of a node only depends on its (T-1)-radius neighborhood
  - prove: after T rounds, the state of a node only depends on its T-radius neighborhood.

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

Locality

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

Proof by induction.

- **T = 0.** trivial
- **T > 0.**
    - assume: after **T-1** rounds, the **state** of a node only depends on its **(T-1)-radius** neighborhood
    - prove: after **T** rounds, the **state** of a node only depends on its **T-radius** neighborhood.

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**

Locality

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

Proof by induction.

- **T = 0.** trivial
- **T > 0.**
  - assume: after **T-1** rounds, the **state** of a node only depends on its **(T-1)-radius** neighborhood
  - prove: after **T** rounds, the **state** of a node only depends on its **T-radius** neighborhood.

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**

Locality

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

Proof by induction.

- **T = 0.** trivial
- **T > 0.**
  - assume: after T-1 rounds, the **state** of a node only depends on its **(T-1)-radius** neighborhood
  - prove: after T rounds, the **state** of a node only depends on its **T-radius** neighborhood.

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

- The **state** of node **v** at time T-1, and ← by inductive hypothesis: only depends on the **(T-1)-radius** neighborhood of **v**
- The **messages** received by **v** at time T, that only depend on:
  - the **state** of the **neighbors** of v at time **T-1** ←

Locality

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

Proof by induction.

- **T = 0.** trivial
- **T > 0.**
  - assume: after **T-1** rounds, the **state** of a node only depends on its **(T-1)-radius** neighborhood
  - prove: after **T** rounds, the **state** of a node only depends on its **T-radius** neighborhood.

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

- The **state** of node **v** at time **T-1**, and ← by inductive hypothesis: only depends on the **(T-1)-radius** neighborhood of **v**
- The **messages** received by **v** at time **T**, that only depend on:
  - the **state** of the **neighbors** of v at time **T-1** ← by inductive hypothesis: only depends on the **(T-1)-radius** neighborhood of the **neighbors** of v

Locality

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

Proof by induction.

- T = 0. trivial
- T > 0.
  - assume: after T-1 rounds, the state of a node only depends on its (T-1)-radius neighborhood
  - prove: after T rounds, the state of a node only depends on its T-radius neighborhood.

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

by inductive hypothesis:
only depends on the (T-1)-radius neighborhood of v

by inductive hypothesis:
only depends on the (T-1)-radius neighborhood of the neighbors of v

Locality

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

Proof by induction.

- **T = 0.** trivial
- **T > 0.**
  - assume: after **T-1** rounds, the **state** of a node only depends on its **(T-1)-radius** neighborhood
  - prove: after **T** rounds, the **state** of a node only depends on its **T-radius** neighborhood.

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**

by inductive hypothesis:
only depends on the **(T-1)-radius** neighborhood of **v**

by inductive hypothesis:
only depends on the **(T-1)-radius** neighborhood of the **neighbors** of v

Locality

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

Proof by induction.

- **T = 0.** trivial
- **T > 0.**
  - assume: after T-1 rounds, the **state** of a node only depends on its **(T-1)-radius** neighborhood
  - prove: after T rounds, the **state** of a node only depends on its **T-radius** neighborhood.

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

- The **state** of node **v** at time T-1, and ← by inductive hypothesis: only depends on the **(T-1)-radius** neighborhood of **v**
- The **messages** received by **v** at time T, that only depend on:
  - the **state** of the **neighbors** of v at time T-1 ← by inductive hypothesis: only depends on the **(T-1)-radius** neighborhood of the **neighbors** of v

contained in the T-radius neighborhood of v

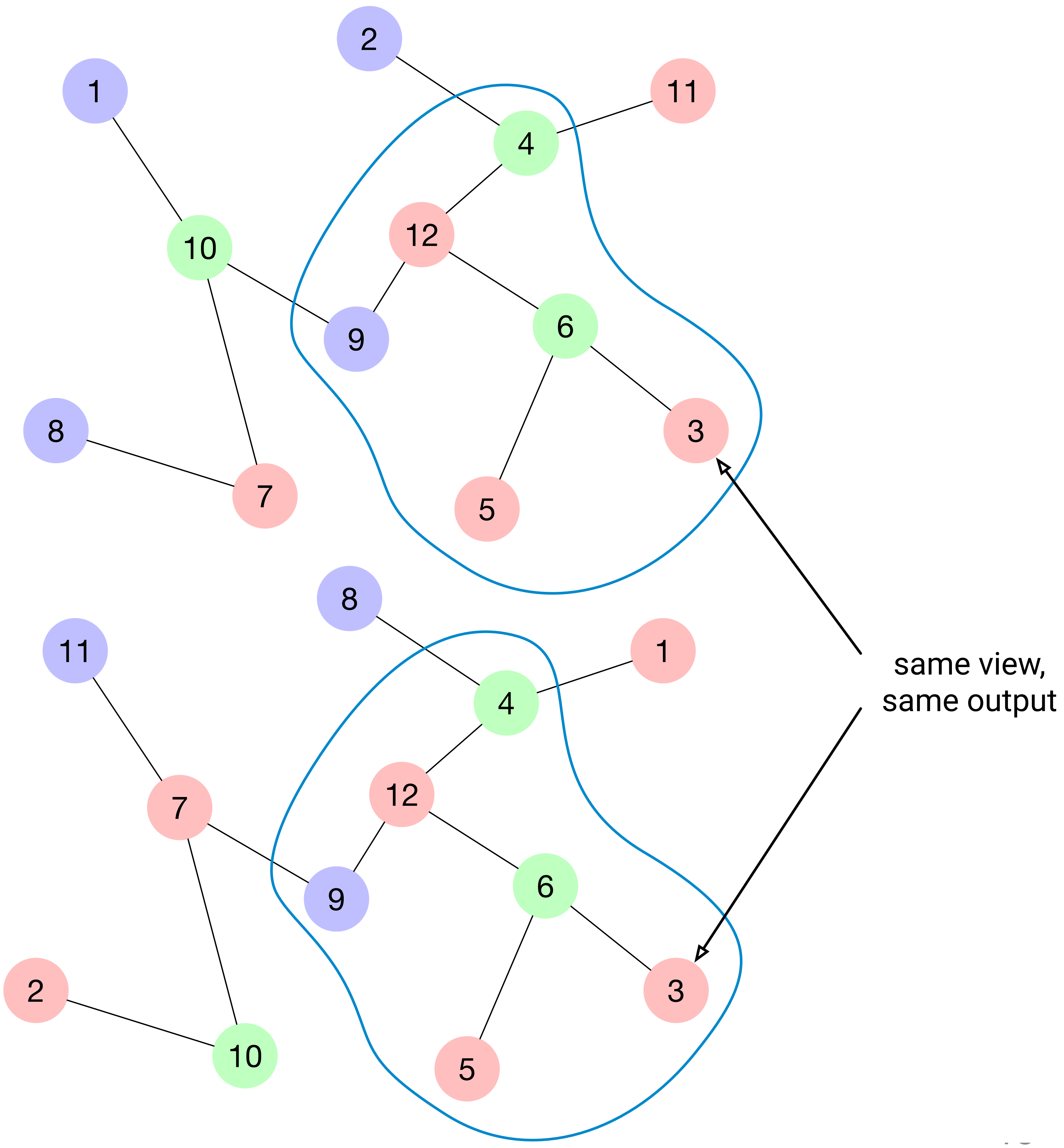
Main technique to prove lower bounds

same radius-T neighborhood

⇩

any T-round algorithm

outputs the same



Main technique to prove lower bounds

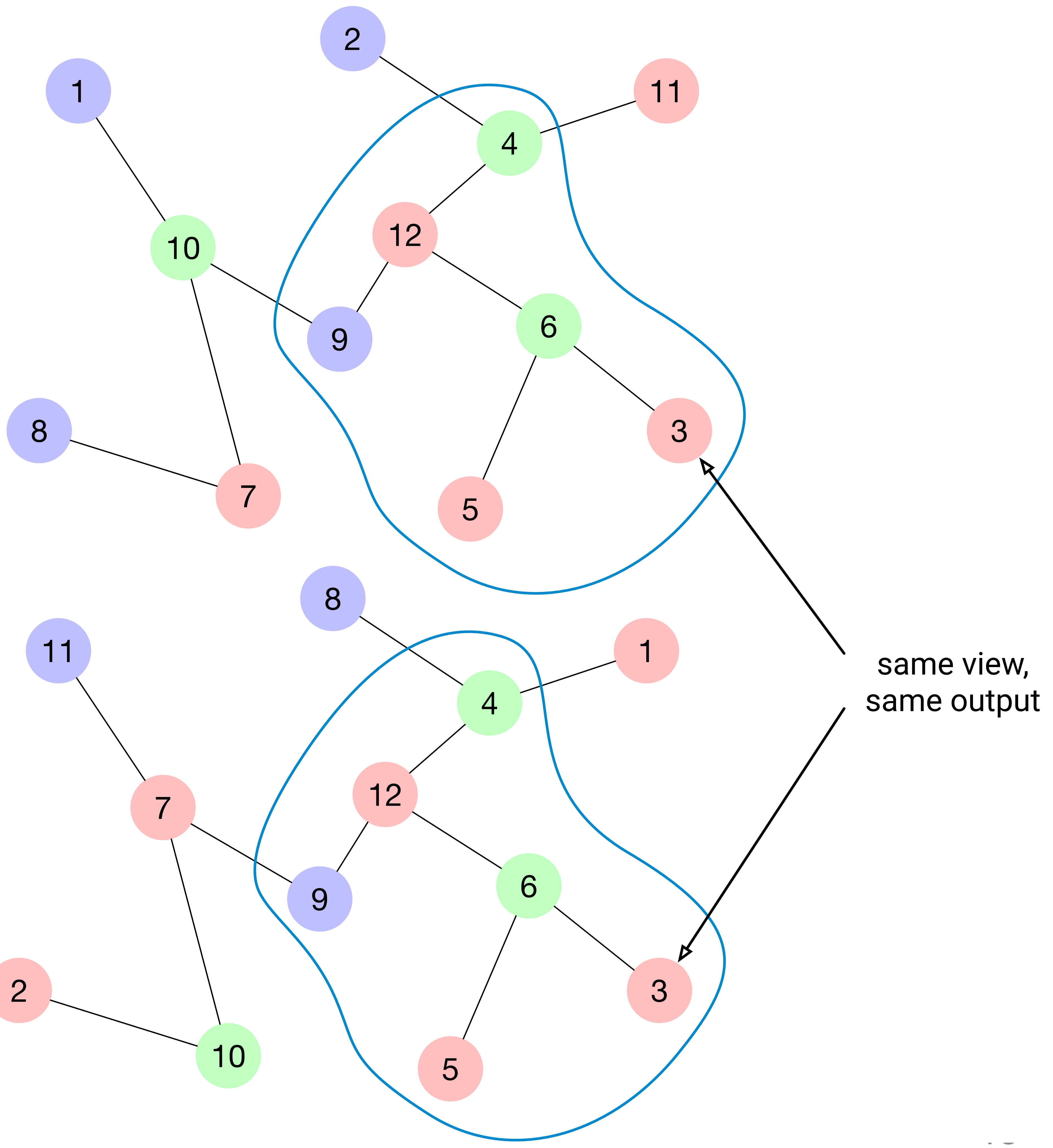
same radius-T neighborhood

⇩

any T-round algorithm

outputs the same

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



Main technique to prove lower bounds

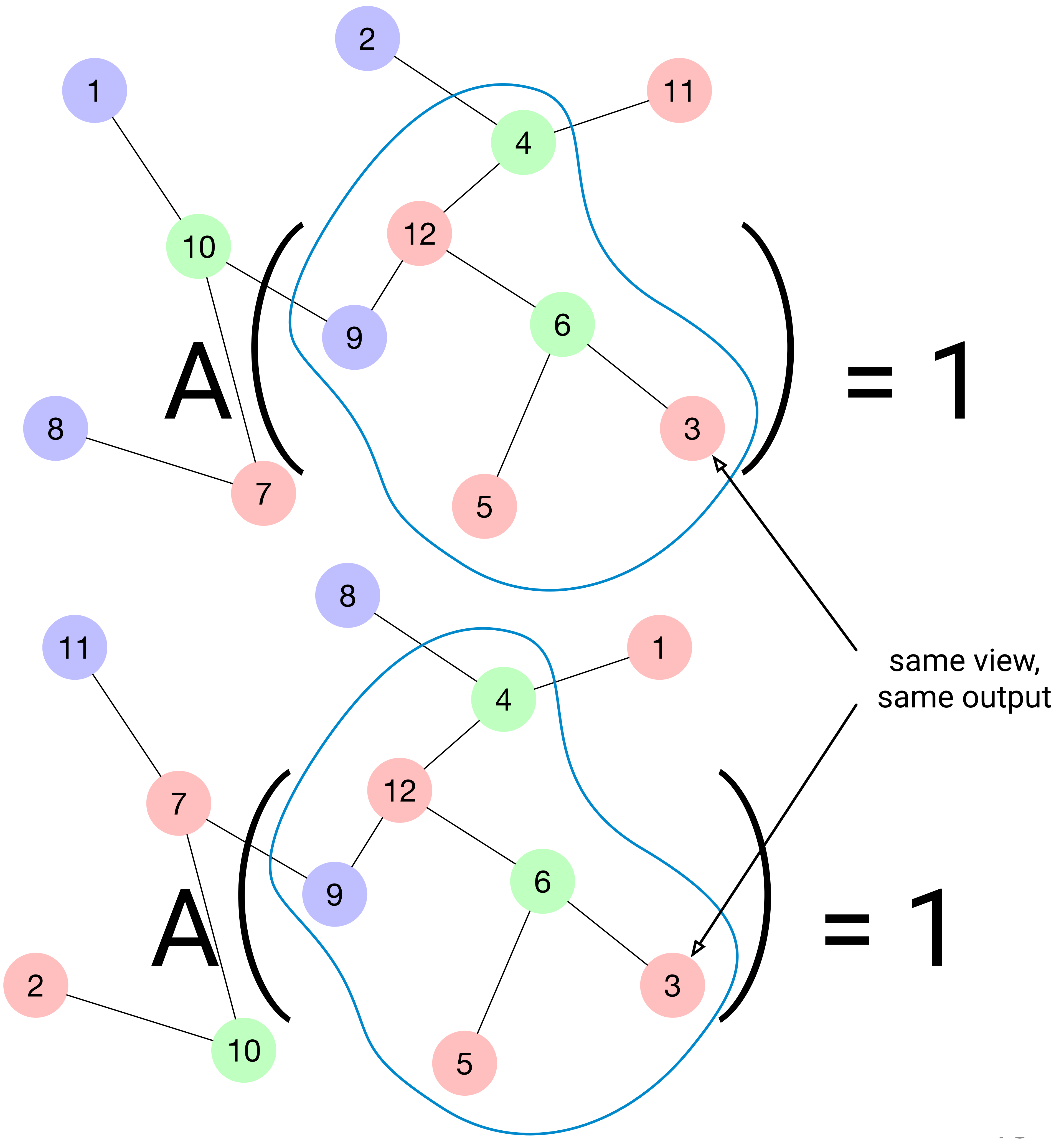
same radius-T neighborhood

⇩

any T-round algorithm

outputs the same

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



Main technique to prove lower bounds

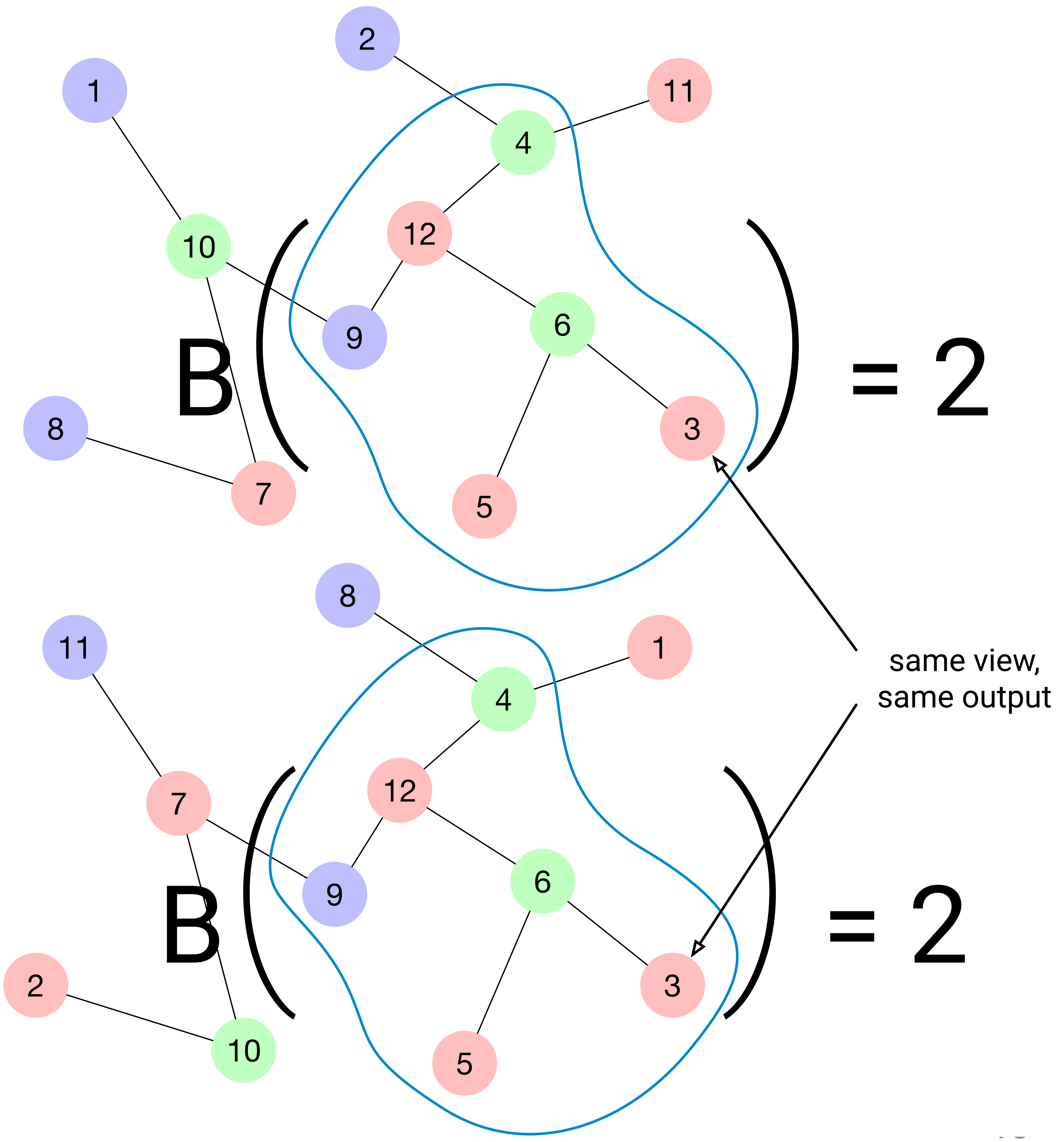
same radius-T neighborhood

⇩

any T-round algorithm

outputs the same

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



Main technique to prove lower bounds

It can be extended to randomized algorithms:

same radius-T view

⇩

same **probability distribution**

over the outputs

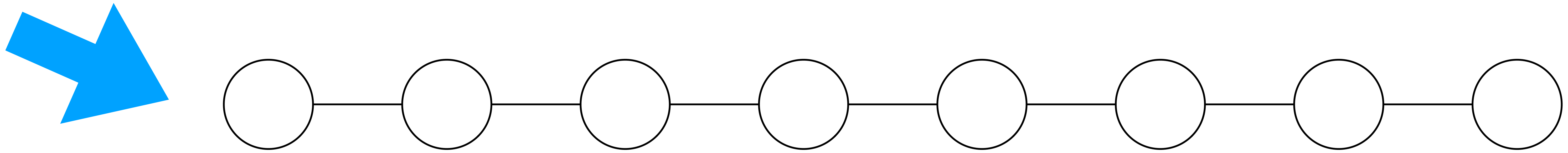
2-coloring

2-coloring

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

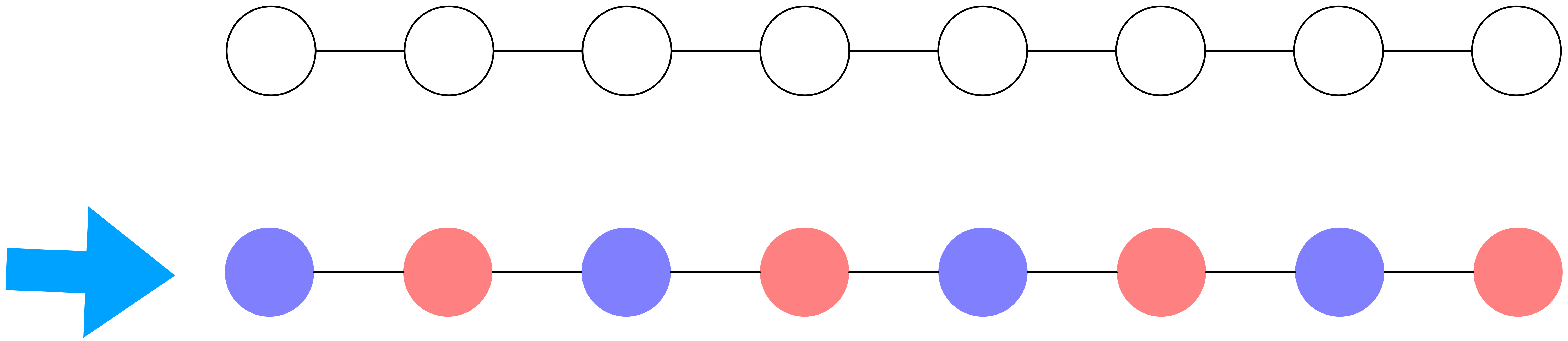
2-coloring

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



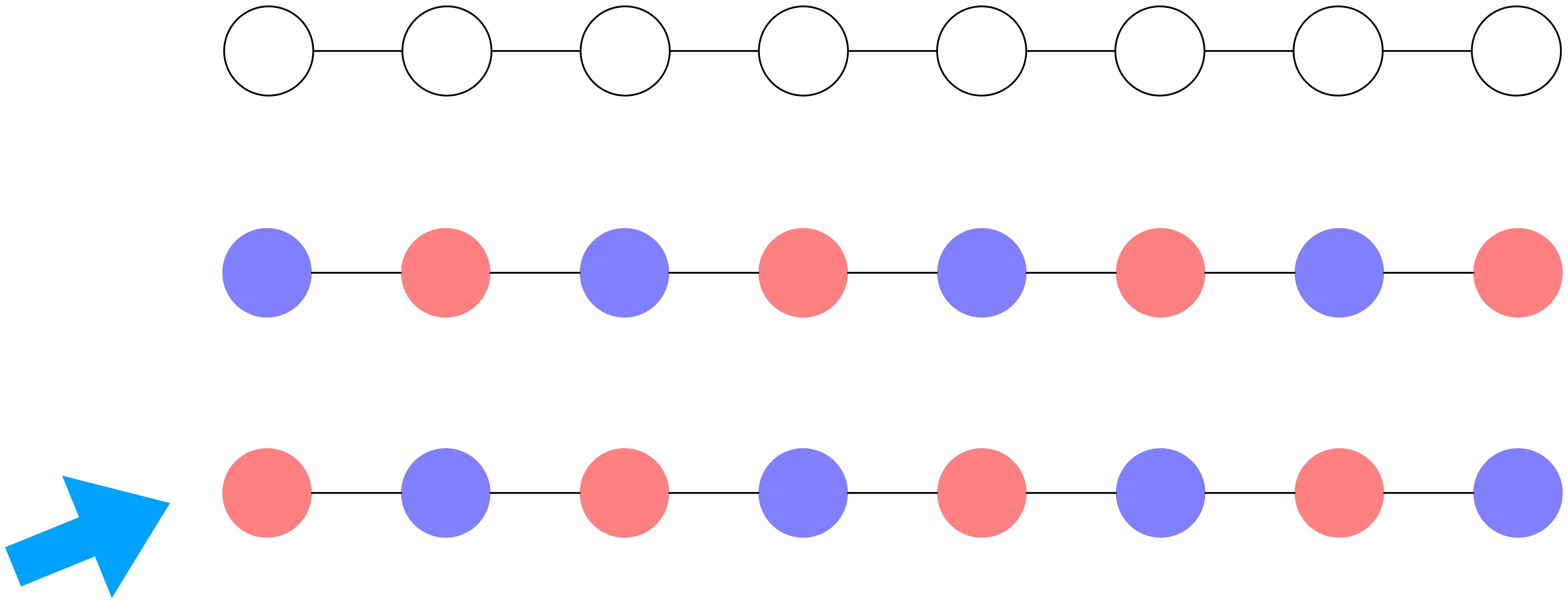
2-coloring

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



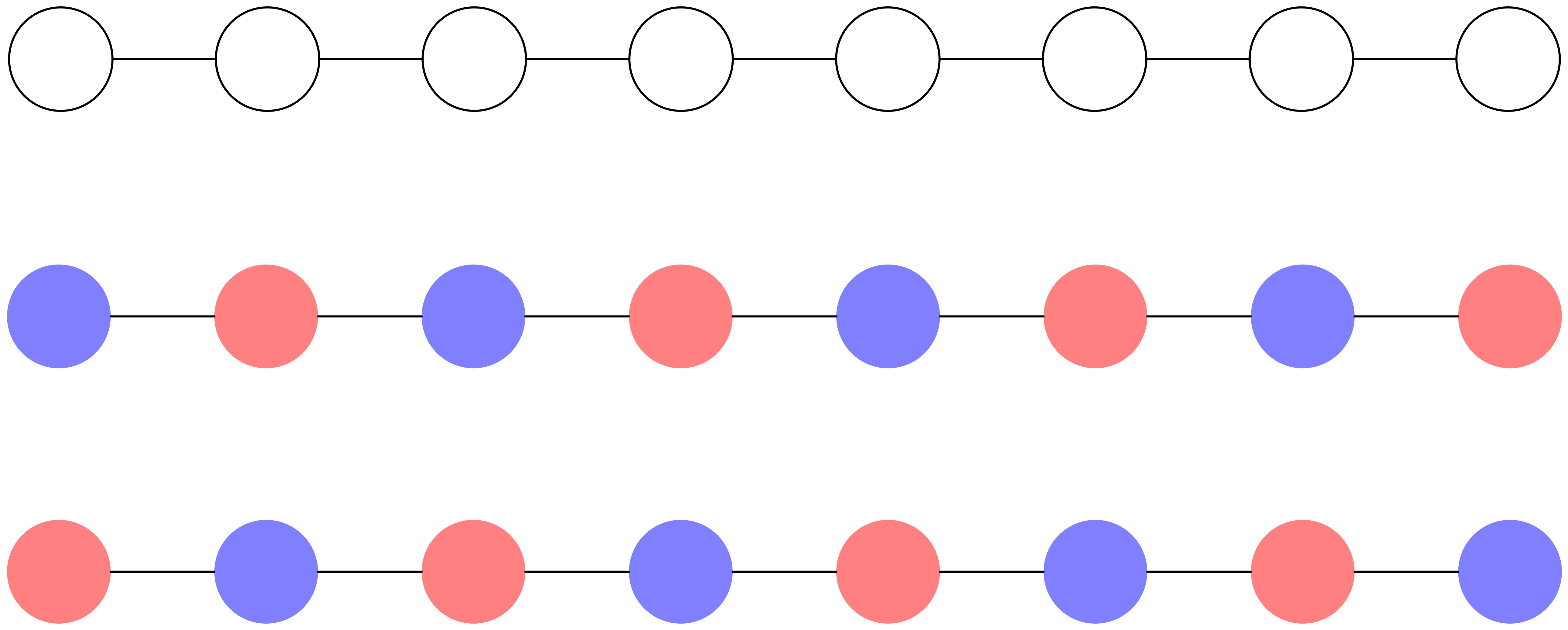
2-coloring

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



2-coloring

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



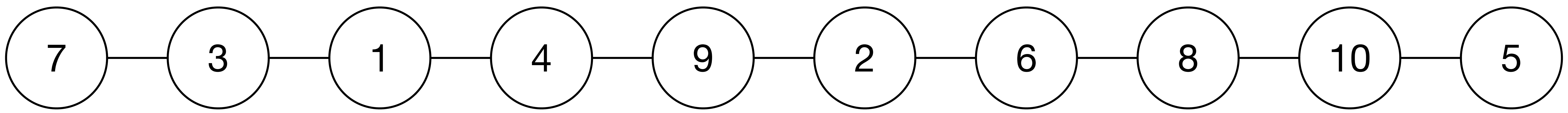
2-coloring

2-coloring

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

2-coloring

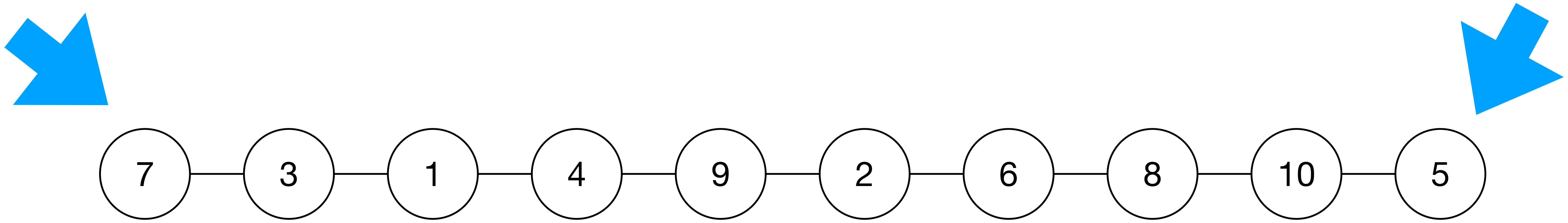
We can solve 2-coloring in **O(n)** rounds on paths



2-coloring

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

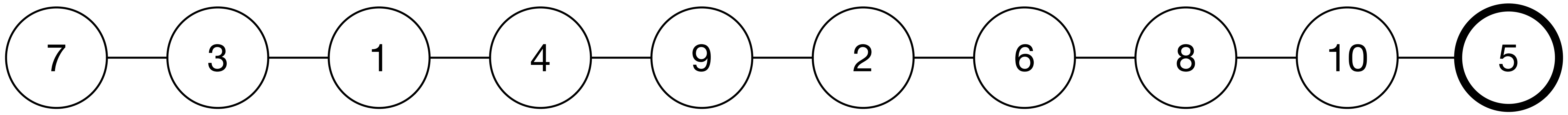
1. Find minimum among endpoints



2-coloring

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

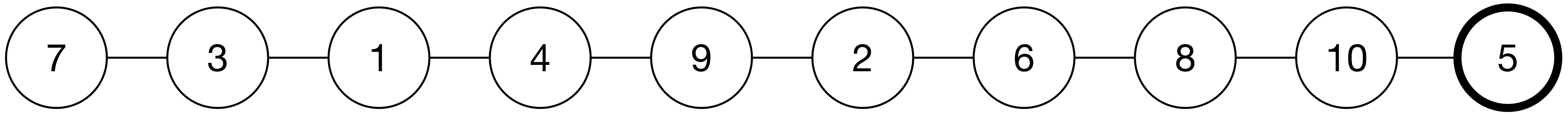
1. Find minimum among endpoints



2-coloring

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

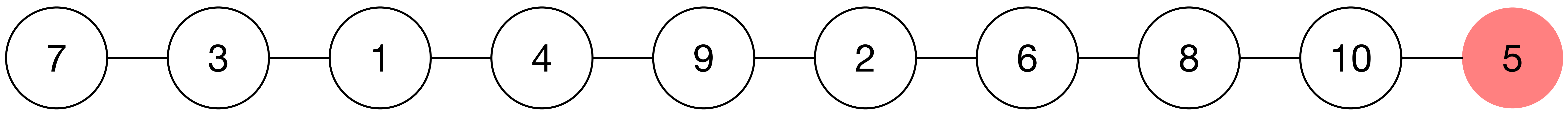
1. Find minimum among endpoints
2. Color it **red**



2-coloring

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

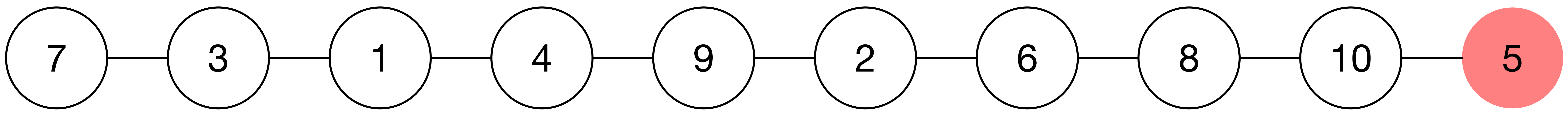
1. Find minimum among endpoints
2. Color it **red**



2-coloring

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

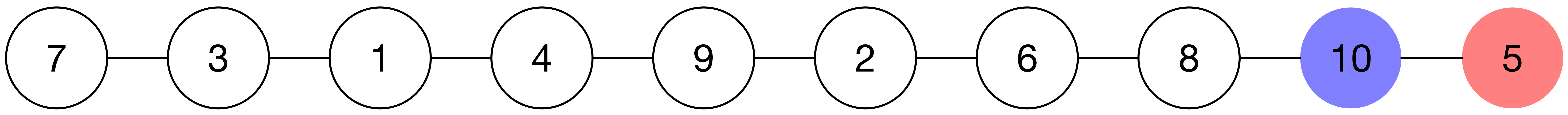
1. Find minimum among endpoints
2. Color it **red**
3. Propagate



2-coloring

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

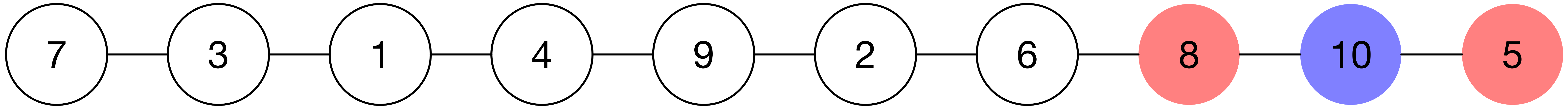
1. Find minimum among endpoints
2. Color it **red**
3. Propagate



2-coloring

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

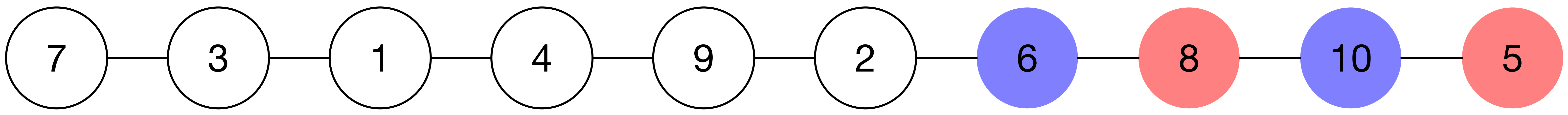
1. Find minimum among endpoints
2. Color it **red**
3. Propagate



2-coloring

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

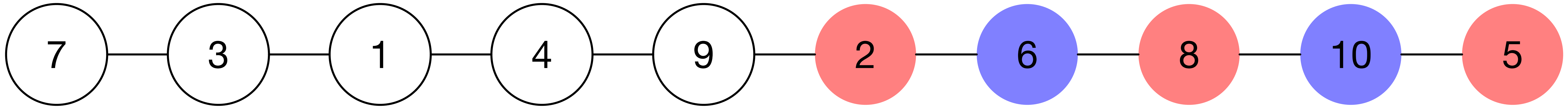
1. Find minimum among endpoints
2. Color it **red**
3. Propagate



2-coloring

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

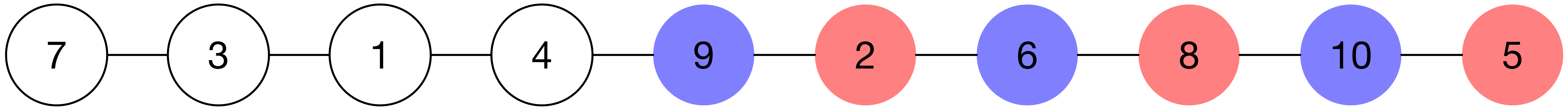
1. Find minimum among endpoints
2. Color it **red**
3. Propagate



2-coloring

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

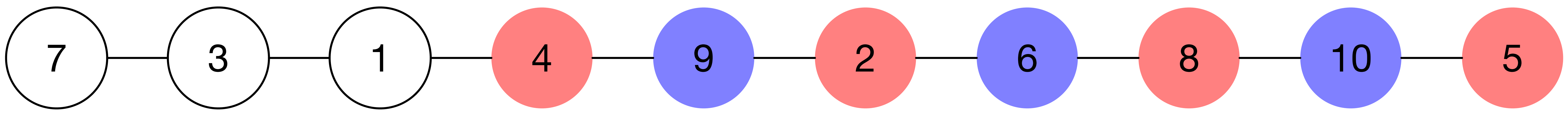
1. Find minimum among endpoints
2. Color it **red**
3. Propagate



2-coloring

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

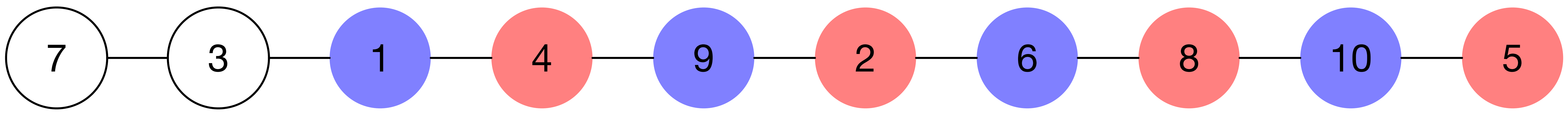
1. Find minimum among endpoints
2. Color it **red**
3. Propagate



2-coloring

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

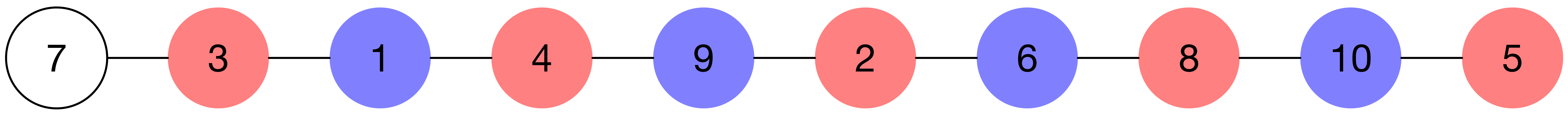
1. Find minimum among endpoints
2. Color it **red**
3. Propagate



2-coloring

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

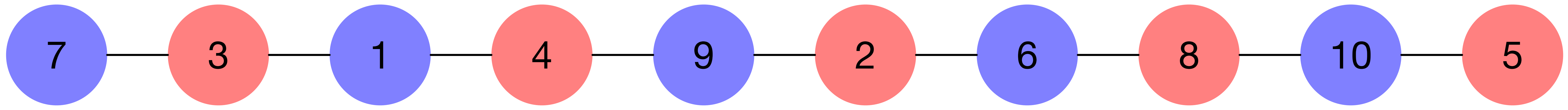
1. Find minimum among endpoints
2. Color it **red**
3. Propagate



2-coloring

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

1. Find minimum among endpoints
2. Color it **red**
3. Propagate



2-coloring

2-coloring

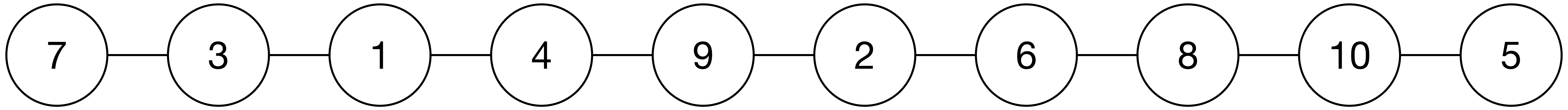
We can prove that **$\Omega(n)$** rounds are required, even if:

- The value of **n** is **known** to all nodes
- IDs are exactly from **$\{1, ..., n\}$**
- Nodes can use **randomization**

2-coloring

We can prove that **Ω(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**

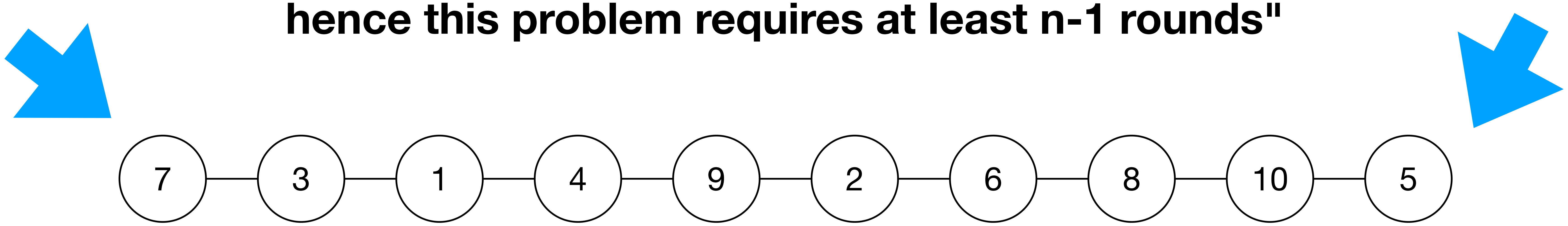


2-coloring

We can prove that **Ω(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**

"Of course these two nodes need to communicate, hence this problem requires at least n-1 rounds"

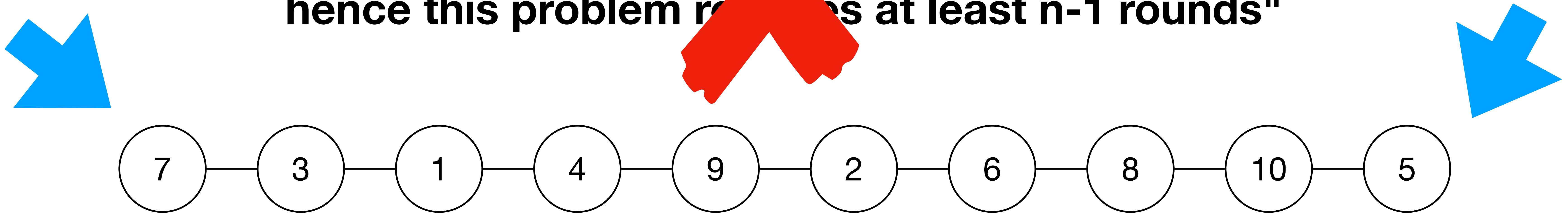


2-coloring

We can prove that **Ω(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**

"Of course these two n s need to communicate,
hence this problem r es at least n-1 rounds"



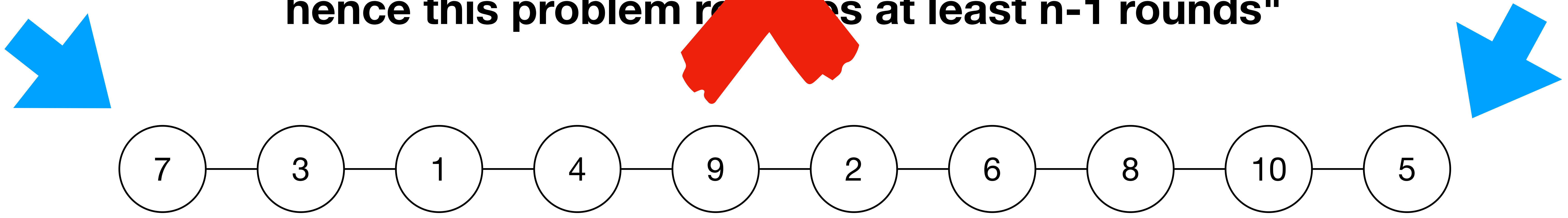
2-coloring

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 paths can be solved in n/2+1 rounds

"Of course these two n s need to communicate, hence this problem r s at least n-1 rounds"



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

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**
- $T(n) \in o(n)$: $\forall \epsilon, \exists n_0, \forall n > n_0, T(n) < \epsilon n$

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**
- $T(n) \in o(n)$: $\forall \epsilon, \exists n_0, \forall n > n_0, T(n) < \epsilon n$

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**
- $T(n) \in o(n)$: $\forall\ \epsilon, \exists\ n_0, \forall\ n > n_0, T(n) < \epsilon\ n$

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**
- $T(n) \in o(n)$: $\forall\, \epsilon$, $\exists\, n_0$, $\forall\, n > n_0$, $T(n) < \epsilon\, n$

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**
- $T(n) \in o(n)$: $\forall\, \epsilon$, $\exists\, n_0$, $\forall\, n > n_0$, $T(n) < \epsilon\, n$

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**
- $T(n) \in o(n)$: $\forall\ \epsilon$, $\exists\ n_0$, $\forall\ n > n_0$, $T(n) < \epsilon\, n$
- $\epsilon = 1/5$. $\exists\ n_0$, $\forall\ n > n_0$, $T(n) < n/5$

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**
- $T(n) \in o(n)$: $\forall\, \epsilon, \exists\, n_0, \forall\, n > n_0, T(n) < \epsilon\, n$
- $\epsilon = 1/5$. $\exists\, n_0, \forall\, n > n_0,$ $T(n) < n/5$

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**
- $T(n) \in o(n)$: $\forall \epsilon, \exists n_0, \forall n > n_0, T(n) < \epsilon n$
- $\epsilon = 1/5$. $\exists n_0, \forall n > n_0,$ $T(n) < n/5$

For any n large enough

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**
- $T(n) \in o(n)$: $\forall\ \epsilon, \exists\ n_0, \forall\ n > n_0, T(n) < \epsilon\ n$
- $\epsilon = 1/5$. $\exists\ n_0, \forall\ n > n_0, T(n) < n/5$

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**
- $T(n) \in o(n)$: $\forall \epsilon, \exists n_0, \forall n > n_0, T(n) < \epsilon n$
- $\epsilon = 1/5$. $\exists n_0, \forall n > n_0, T(n) < n/5$
- If we take **n large enough**, the algorithm must terminate in **less than n/5** rounds.

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**
- $T(n) \in o(n)$: $\forall \epsilon, \exists n_0, \forall n > n_0, T(n) < \epsilon n$
- $\epsilon = 1/5$. $\exists n_0, \forall n > n_0, T(n) < n/5$
- If we take **n large enough**, the algorithm must terminate in **less than n/5** rounds.

2-coloring lower bound

- Let us prove that **n/5** rounds are not enough, for all (large enough) **n**
- We use the **principle of locality**. 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

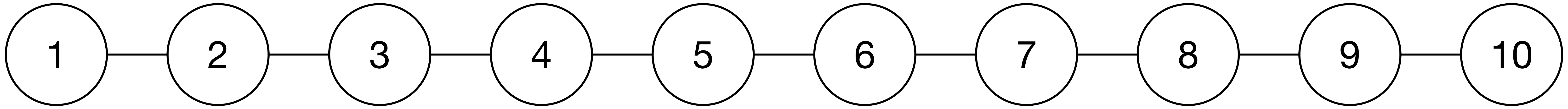
2-coloring lower bound

2-coloring lower bound

For n = 10, **T = 2** is not enough

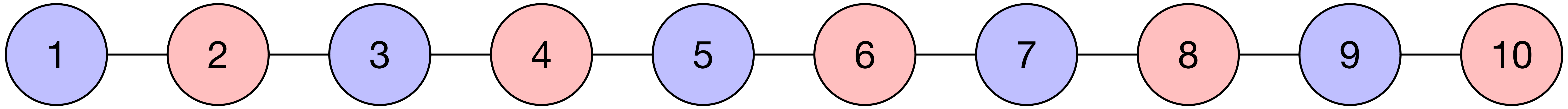
2-coloring lower bound

For n = 10, **T = 2** is not enough



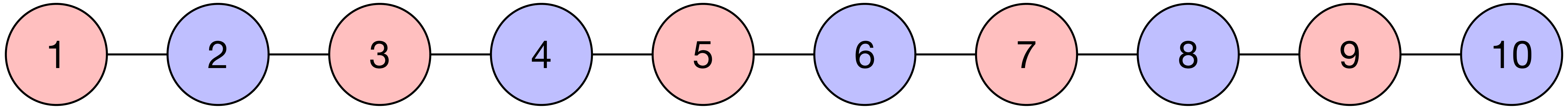
2-coloring lower bound

For n = 10, **T = 2** is not enough



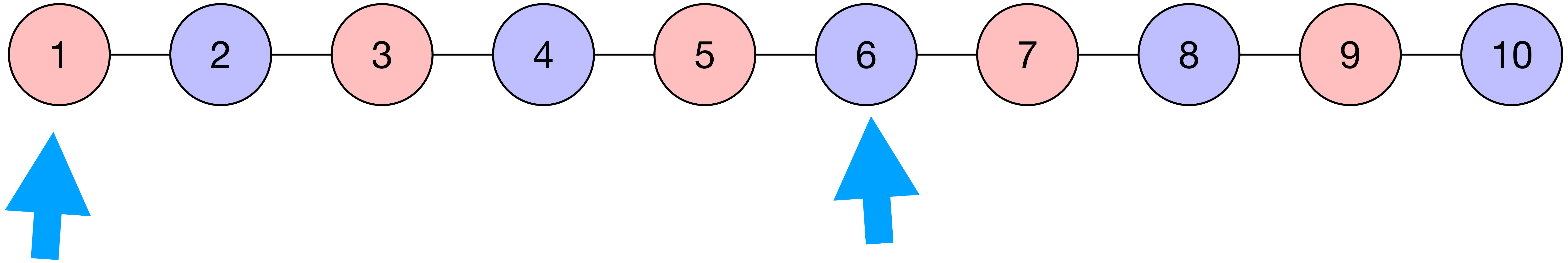
2-coloring lower bound

For n = 10, **T = 2** is not enough



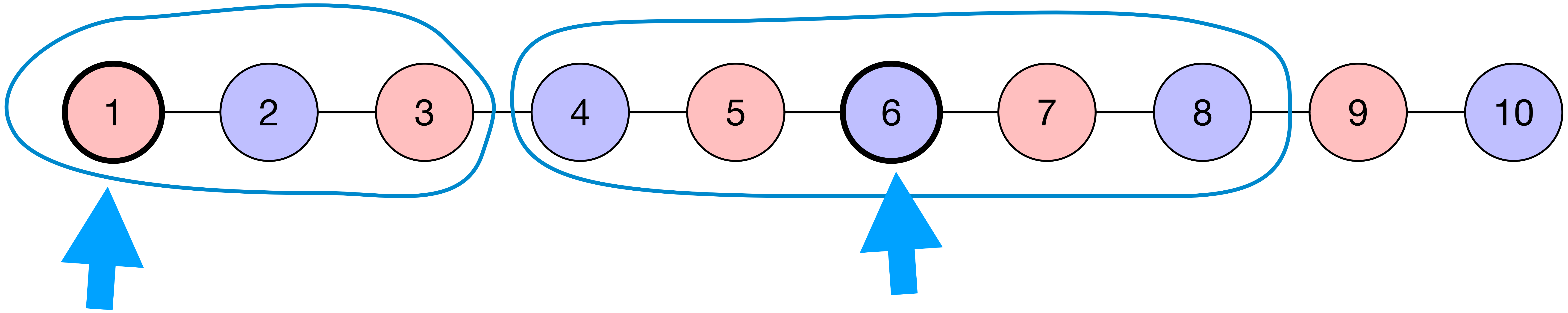
2-coloring lower bound

For n = 10, **T = 2** is not enough



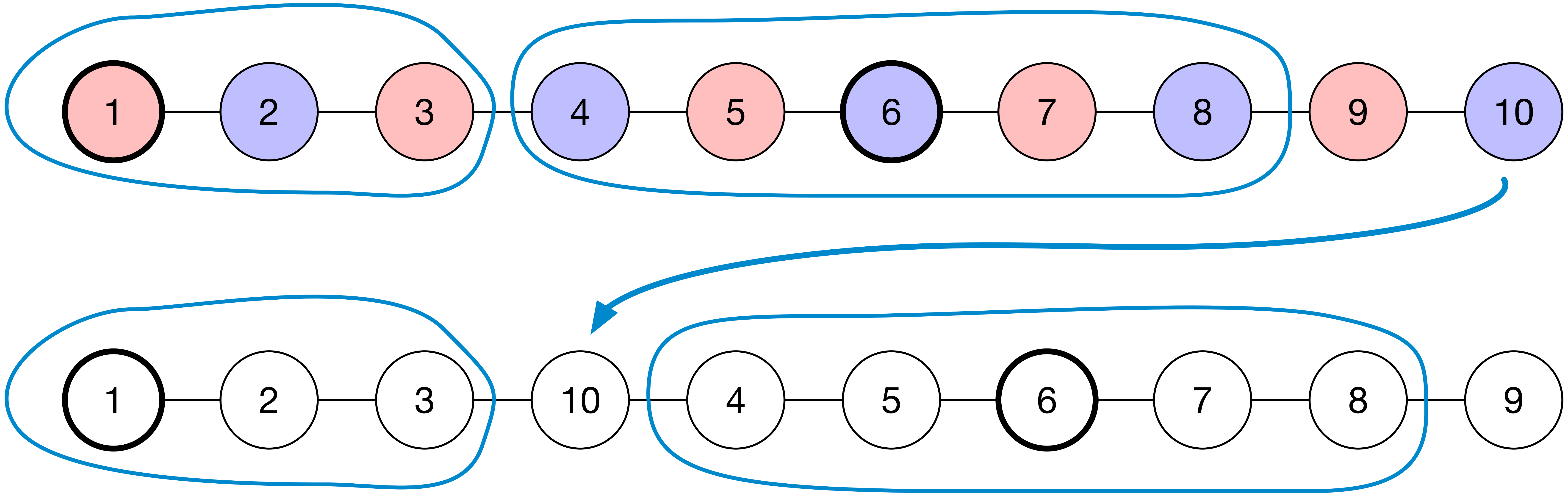
2-coloring lower bound

For n = 10, **T = 2** is not enough



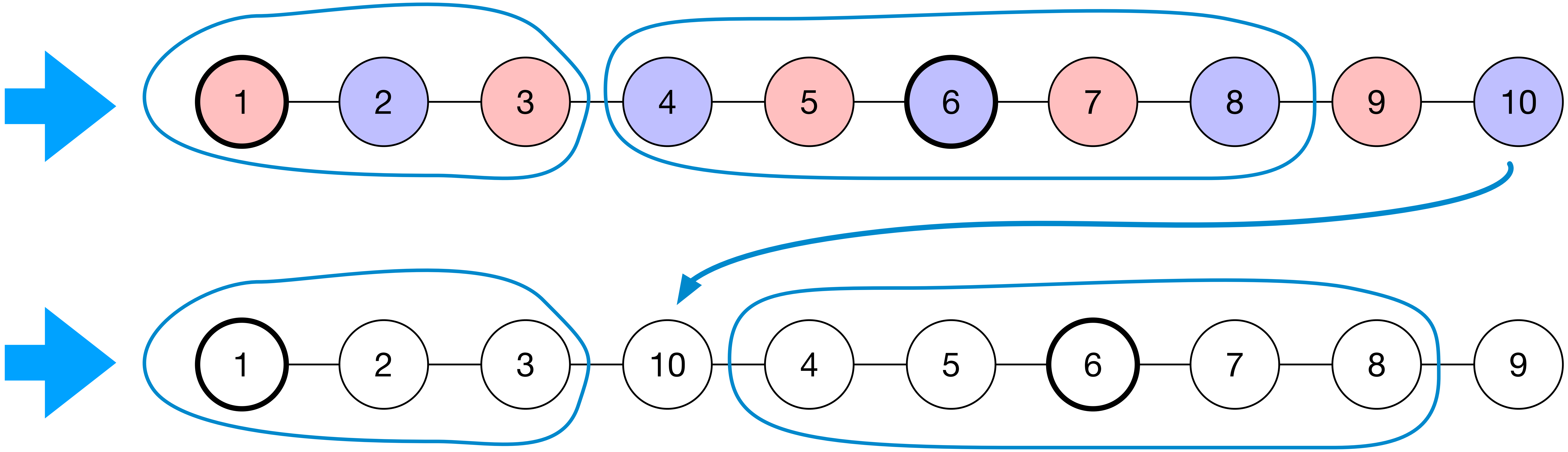
2-coloring lower bound

For n = 10, **T = 2** is not enough



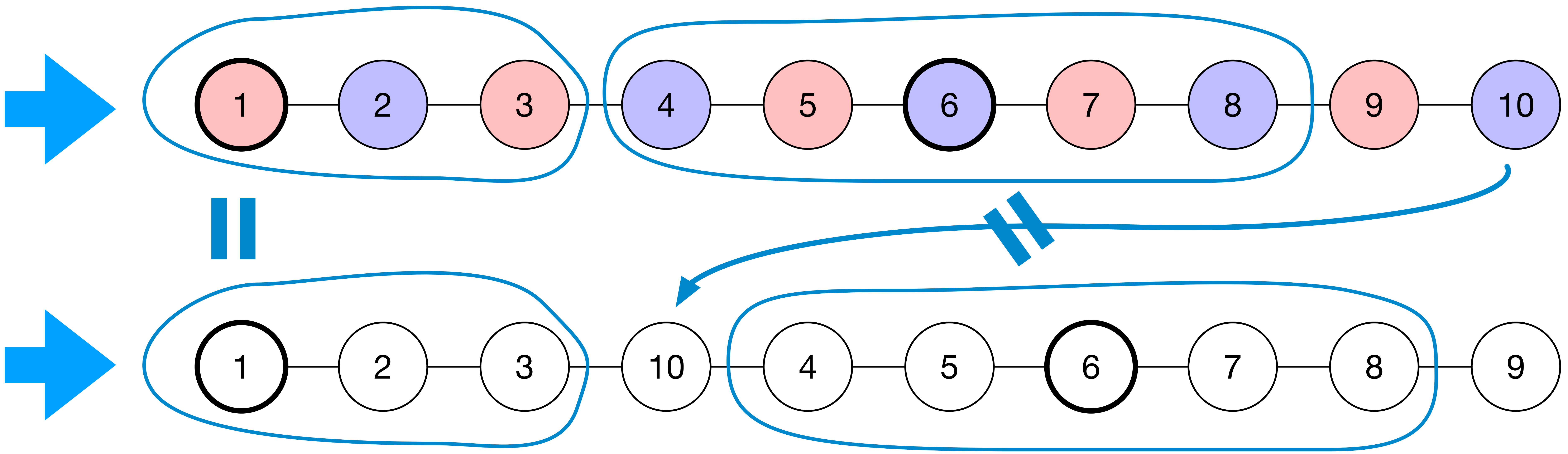
2-coloring lower bound

For n = 10, **T = 2** is not enough



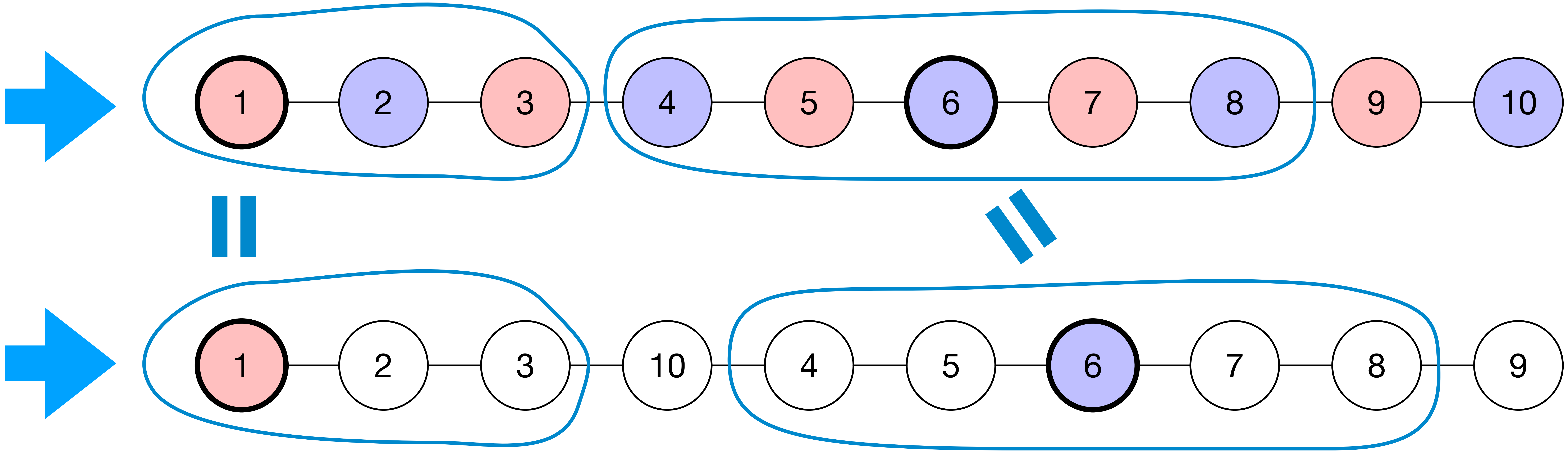
2-coloring lower bound

For n = 10, **T = 2** is not enough



2-coloring lower bound

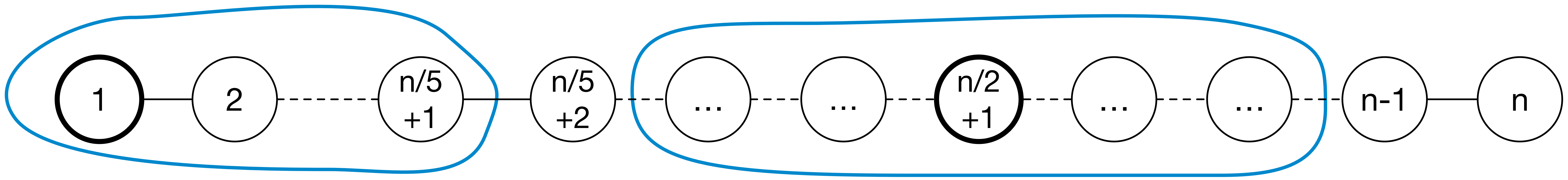
For n = 10, **T = 2** is not enough



2-coloring lower bound

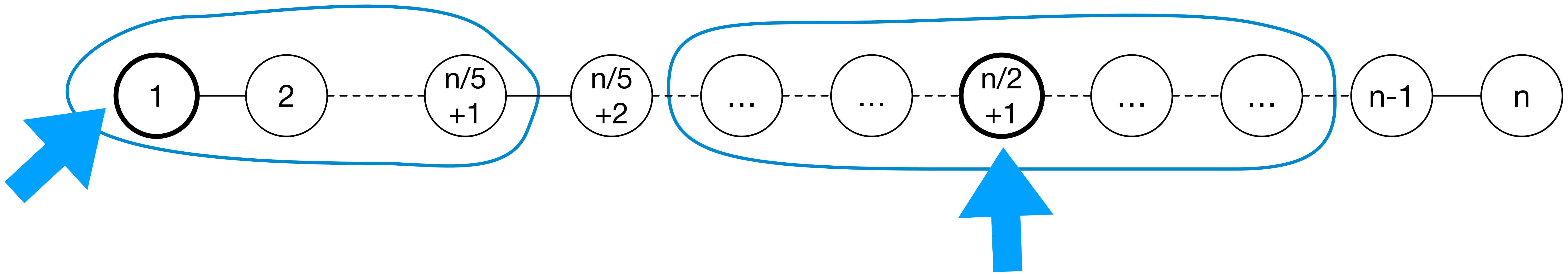
2-coloring lower bound

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



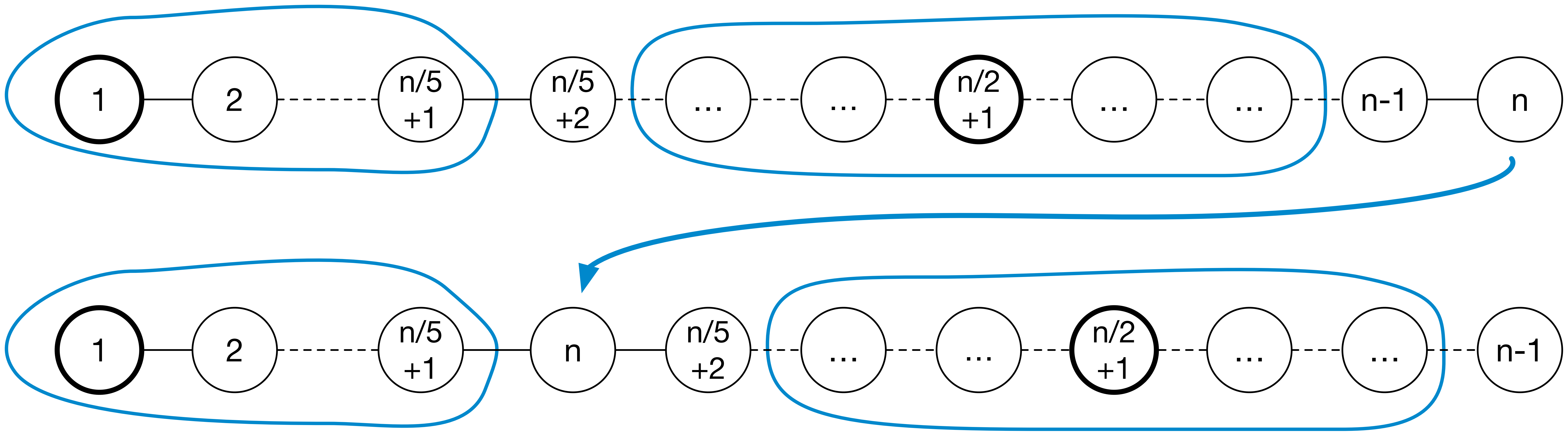
2-coloring lower bound

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



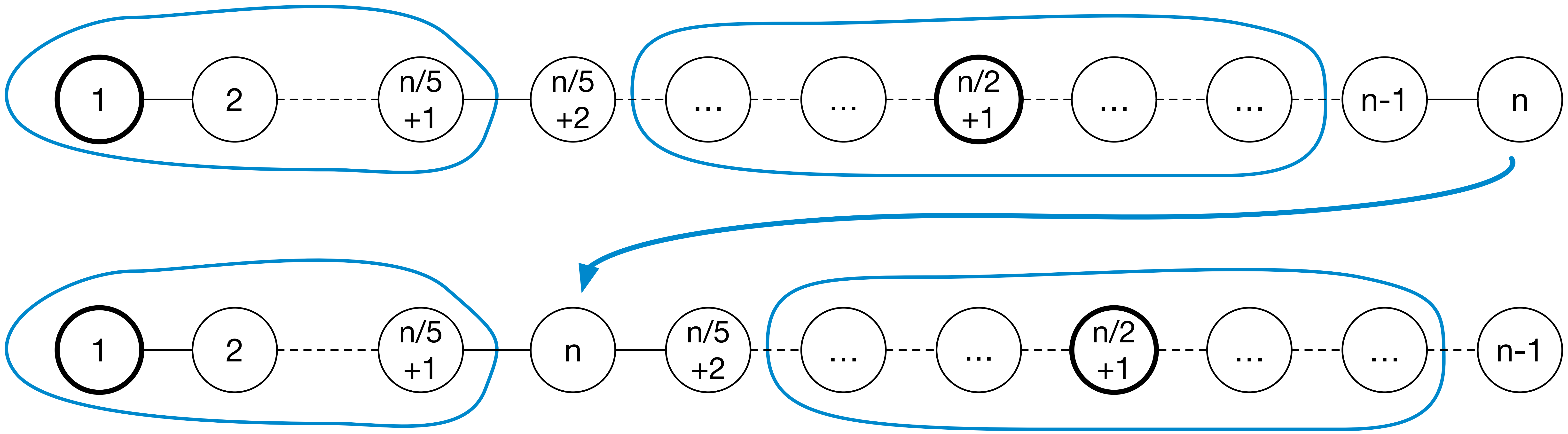
2-coloring lower bound

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

  ⇩

  same **probability distribution**

  over the outputs

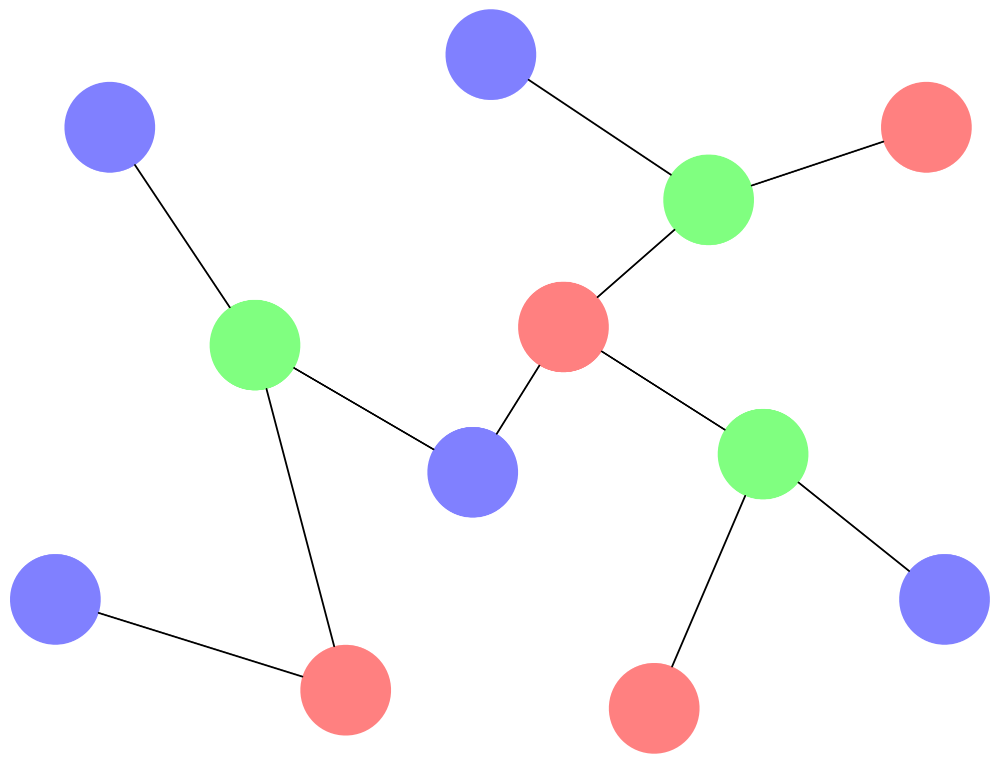
Coloring trees

Coloring trees

- The **3-coloring** problem can be solved in:

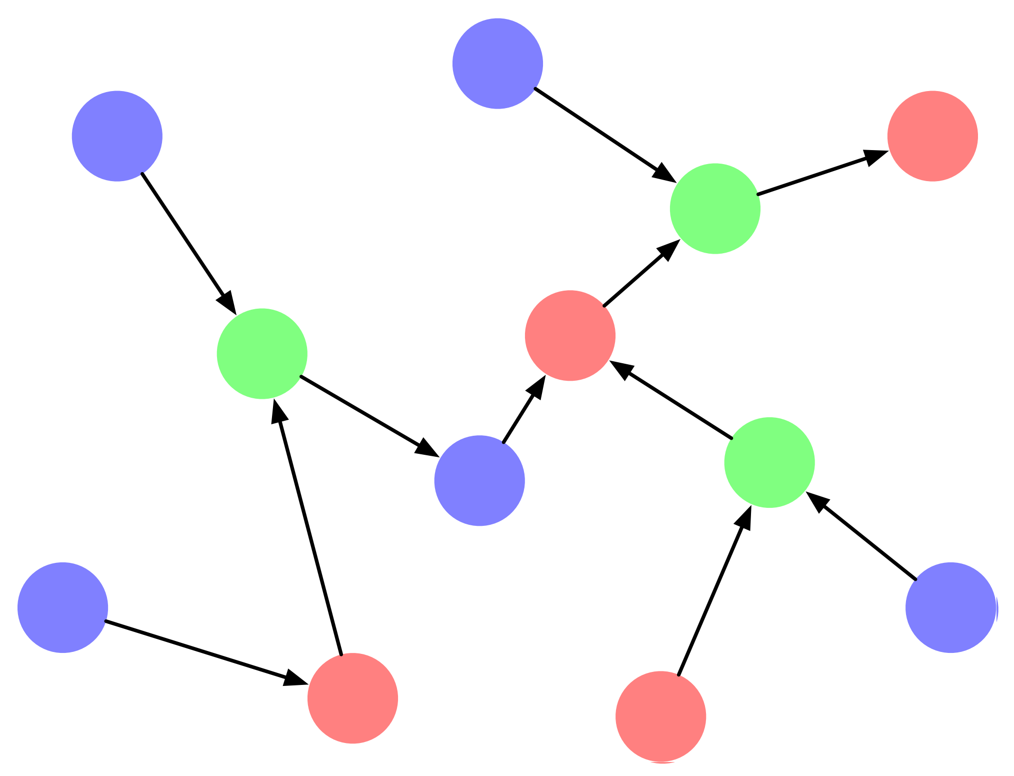
Coloring trees

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



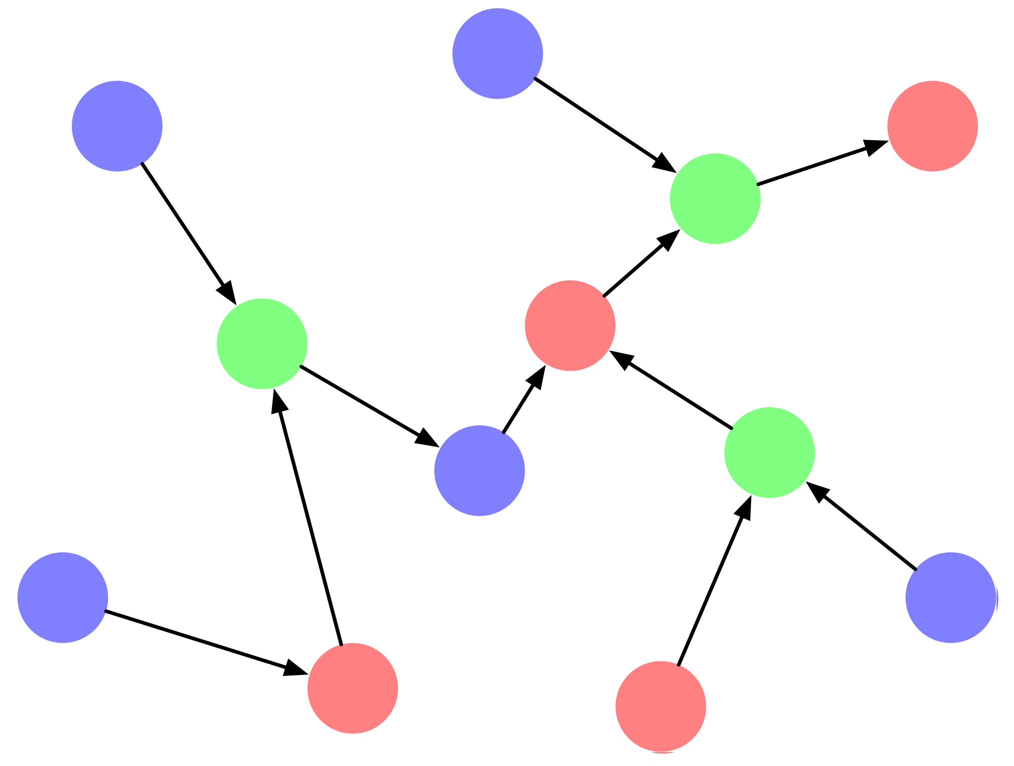
Coloring trees

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



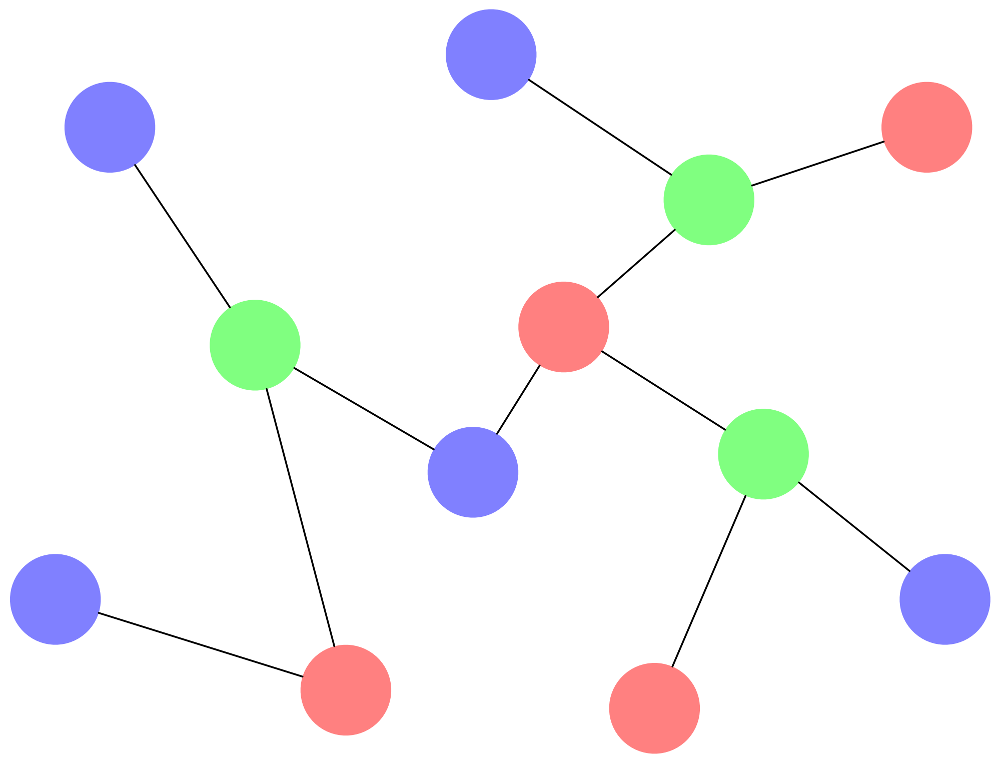
Coloring trees

- 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?

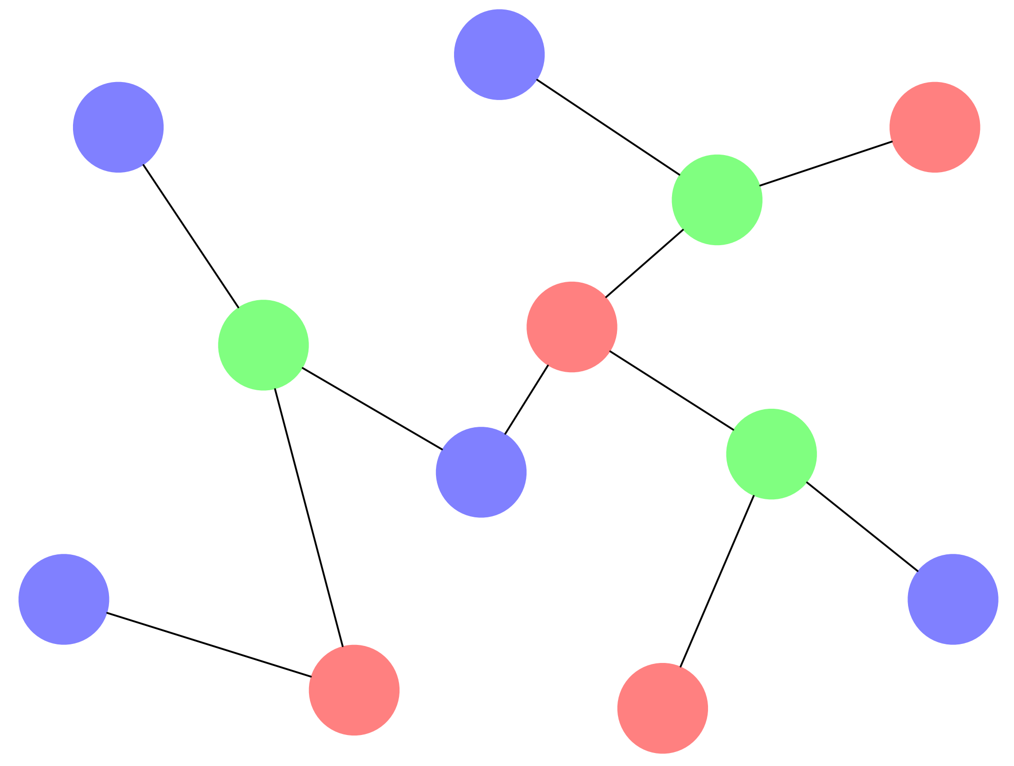
Coloring trees



- 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 of maximum degree **Δ** with **Δ** colors requires **$\Omega(\log_\Delta n)$** rounds

Coloring trees

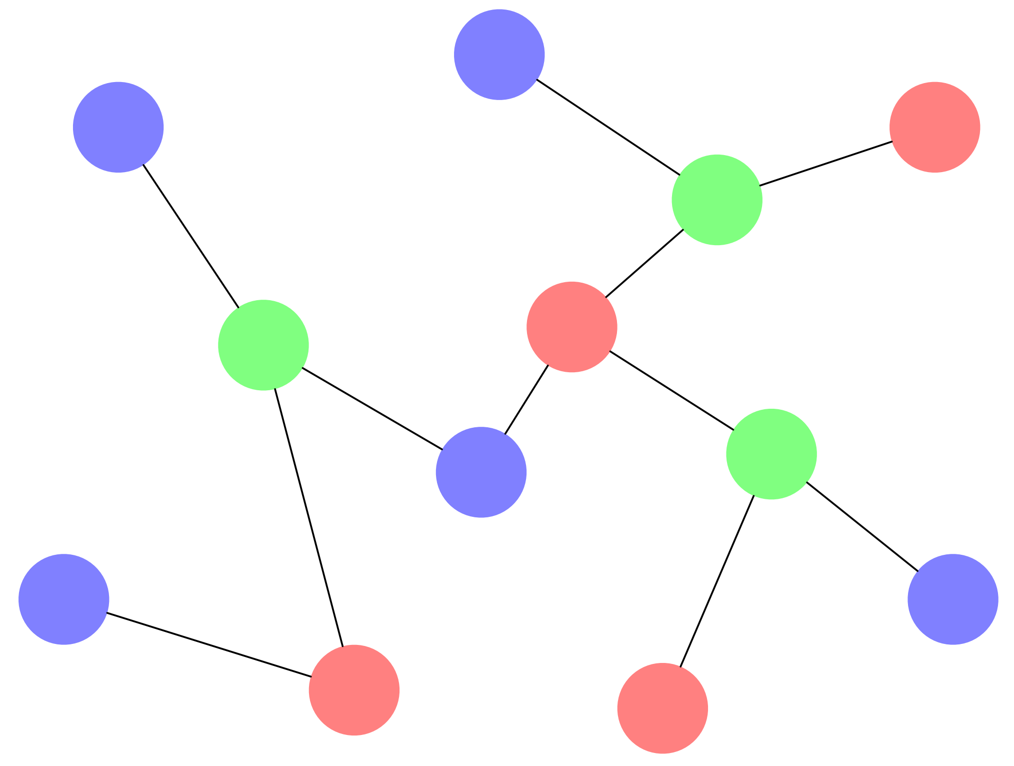
- The **3-coloring** problem can be solved in:
  - **$O(\log n)$** rounds on **trees**
  - **$O(\log^* n)$** rounds on rooted **trees**



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

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

⇒ **3**-coloring trees of (large enough) constant degree requires **Ω(log n)** rounds

Coloring trees lower bound

Coloring trees lower bound

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

Coloring trees lower bound

"Few"

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

Coloring trees lower bound

"Few" "3"

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

Coloring trees lower bound

"Few" "3"

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

- We use the fact that there are graphs that:

Coloring trees lower bound

"Few" "3"

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

- We use the fact that there are graphs that:
  - cannot be colored using **$o(\Delta / \log \Delta)$ colors**

Coloring trees lower bound

"Few" "3"

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

"Large chromatic number"

- We use the fact that there are graphs that:
  - cannot be colored using $o(\Delta / \log \Delta)$ **colors**

Coloring trees lower bound

"Few" "3"

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

- We use the fact that there are graphs that:

"Large chromatic number"

  - 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

"Few" "3"

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

"Large chromatic number"

- 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

"High girth"

Coloring trees lower bound

"Few" "3"

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

"NOT Trees!"

- We use the fact that there are graphs that:

"Large chromatic number"

  - cannot be colored using $o(\Delta / \log \Delta)$ **colors**
  - they look like a tree, in every $o(\log_\Delta n)$ radius neighborhood

"High girth"

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:

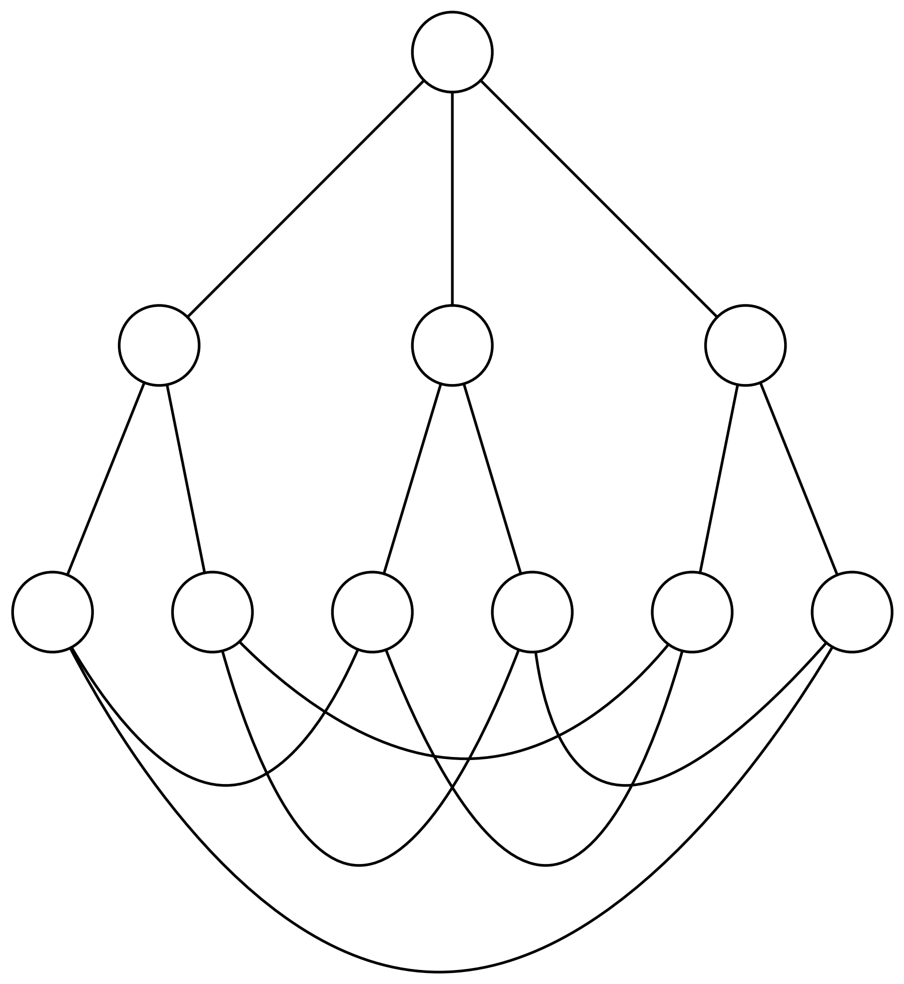
- all nodes have **degree Δ**
- the **girth** is $\Omega(\log_\Delta n)$
- the **chromatic number** is $\Omega(\Delta / \log \Delta)$

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 **$\Omega(\Delta / \log \Delta)$**



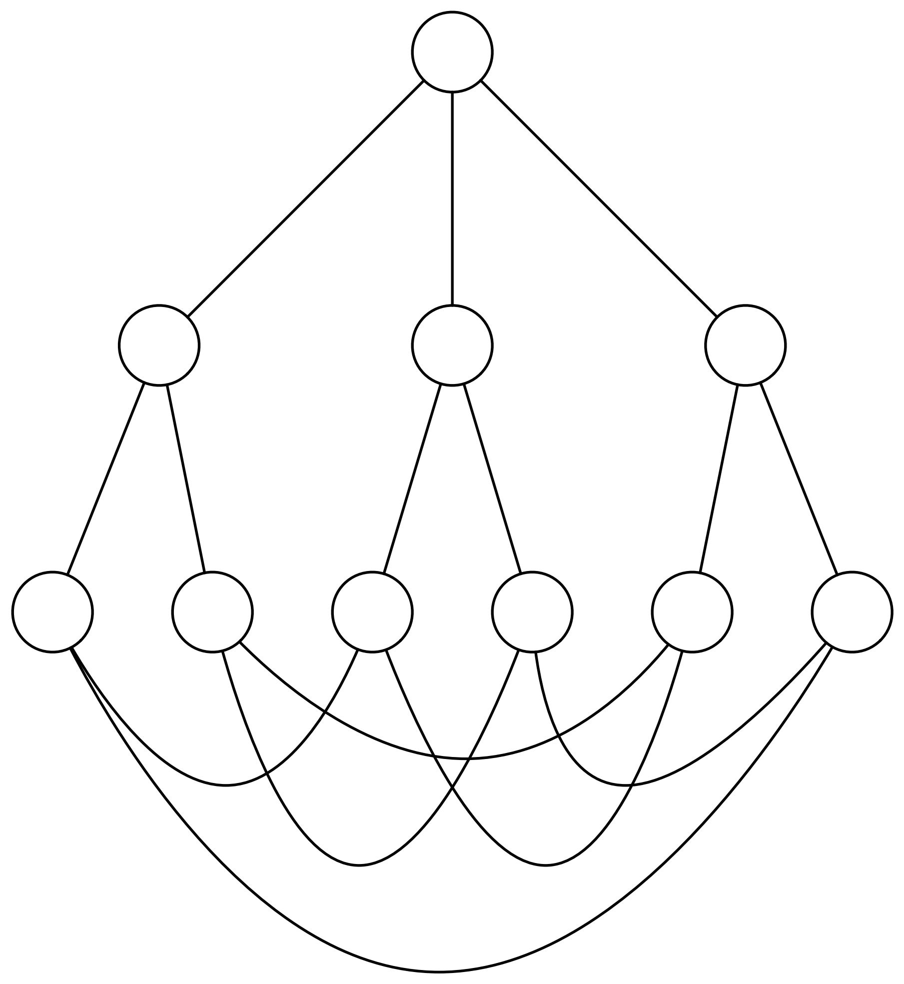
Coloring trees lower bound

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 $\Omega(\Delta / \log \Delta)$

T=2

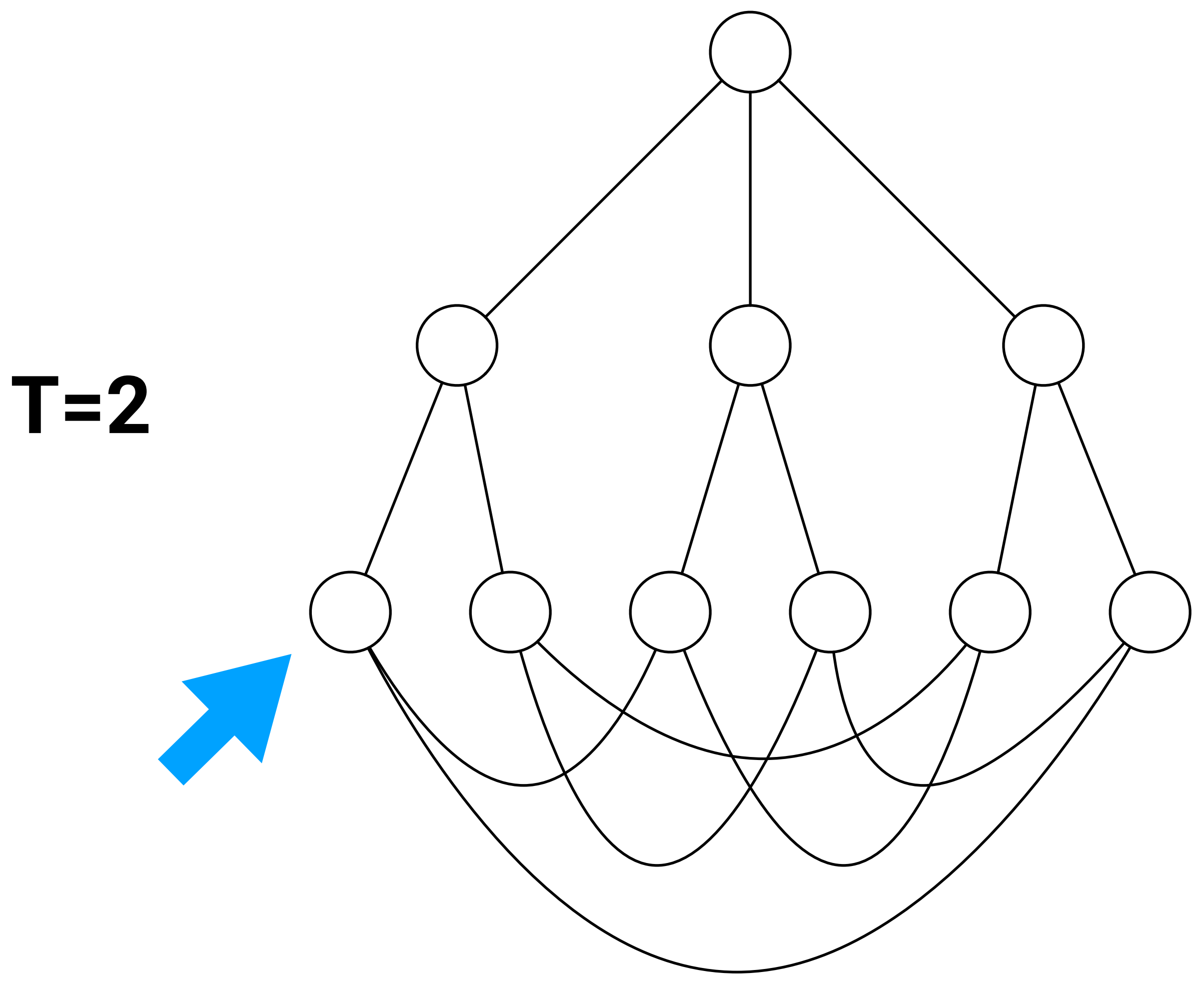


Coloring trees lower bound

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 **$\Omega(\Delta / \log \Delta)$**

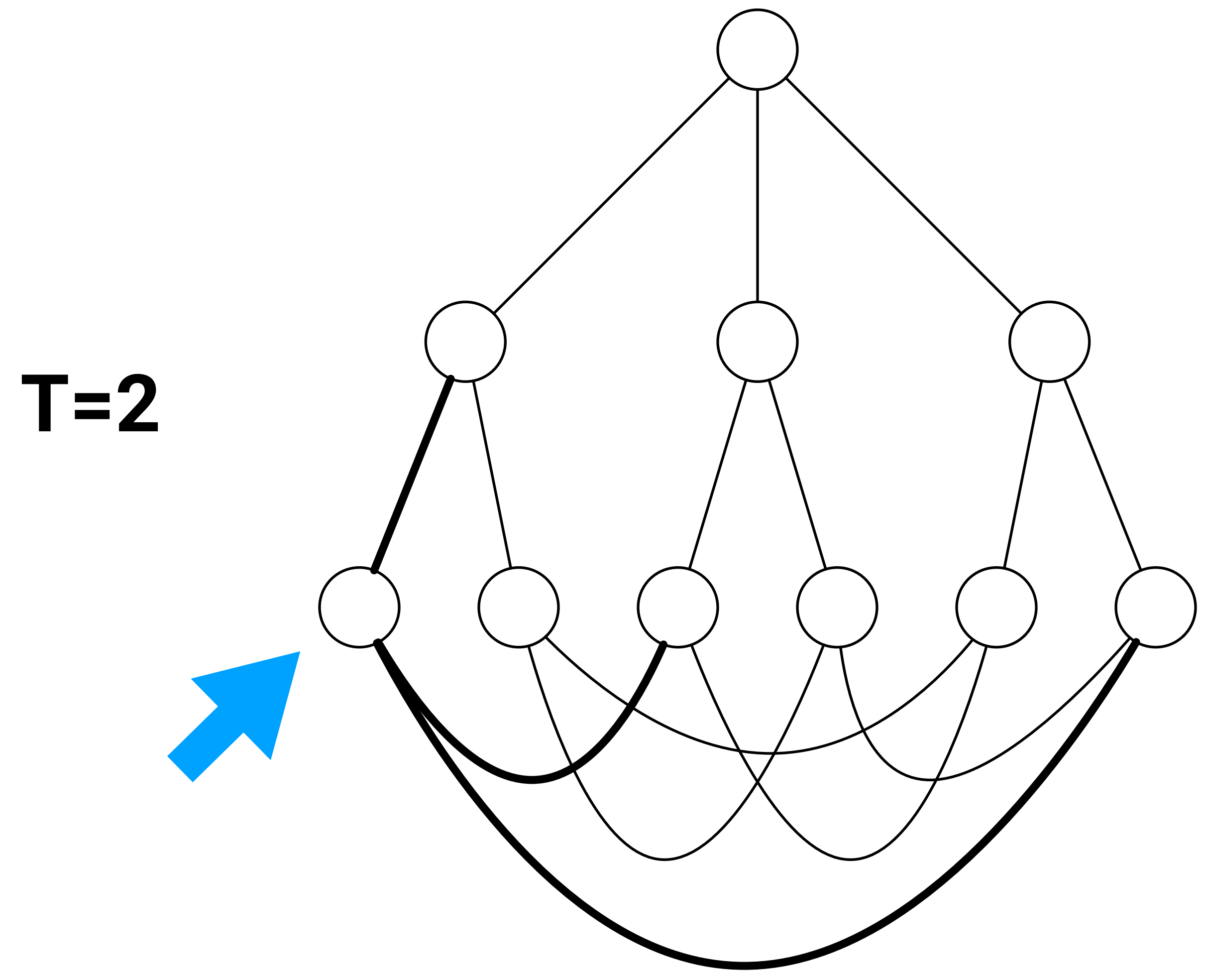


Coloring trees lower bound

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 **$\Omega(\Delta / \log \Delta)$**

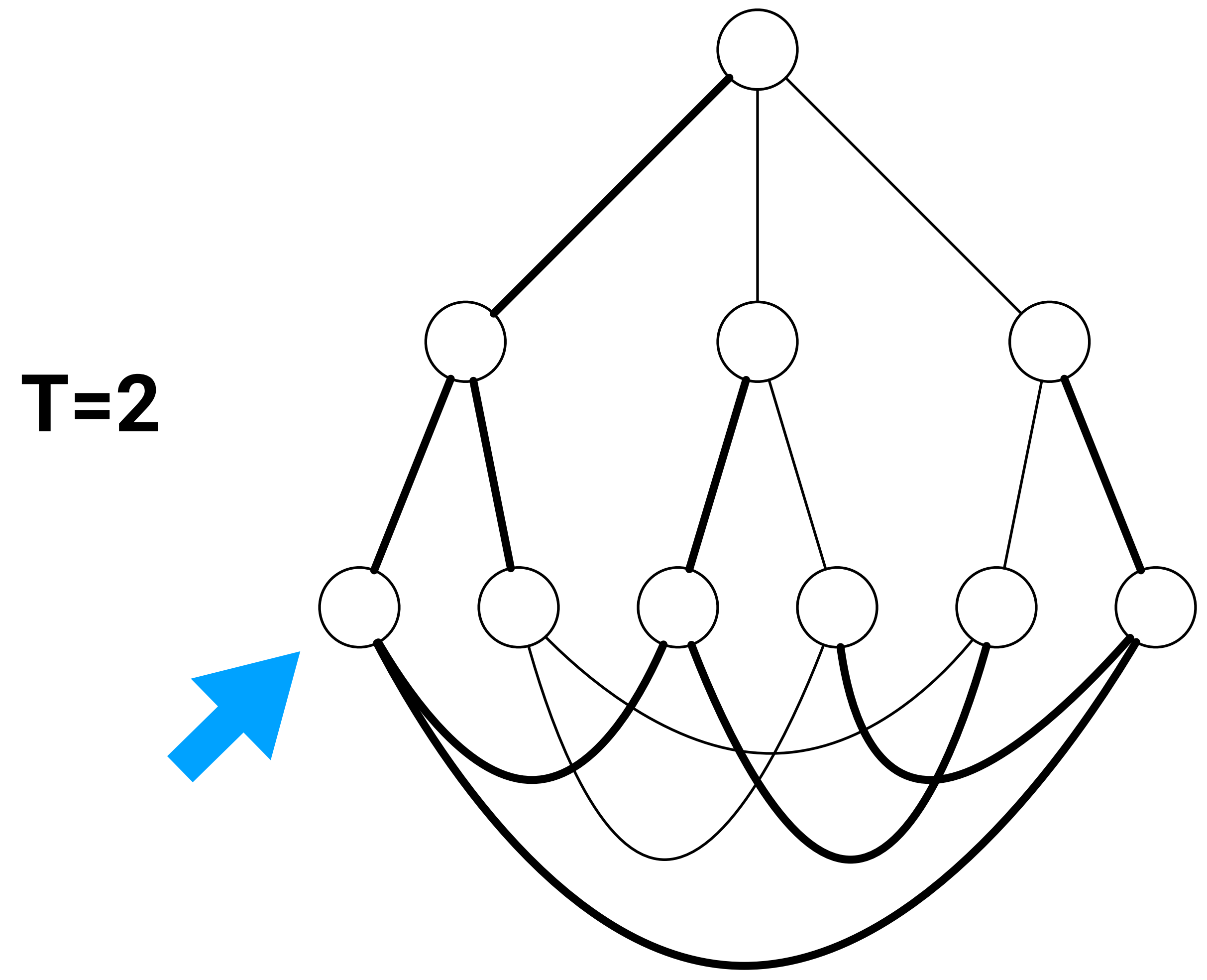


Coloring trees lower bound

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 Δ requires $\Omega(\log_\Delta n)$ rounds

Coloring trees lower bound

- We want to prove that **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **Δ** 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 **Δ** 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**?

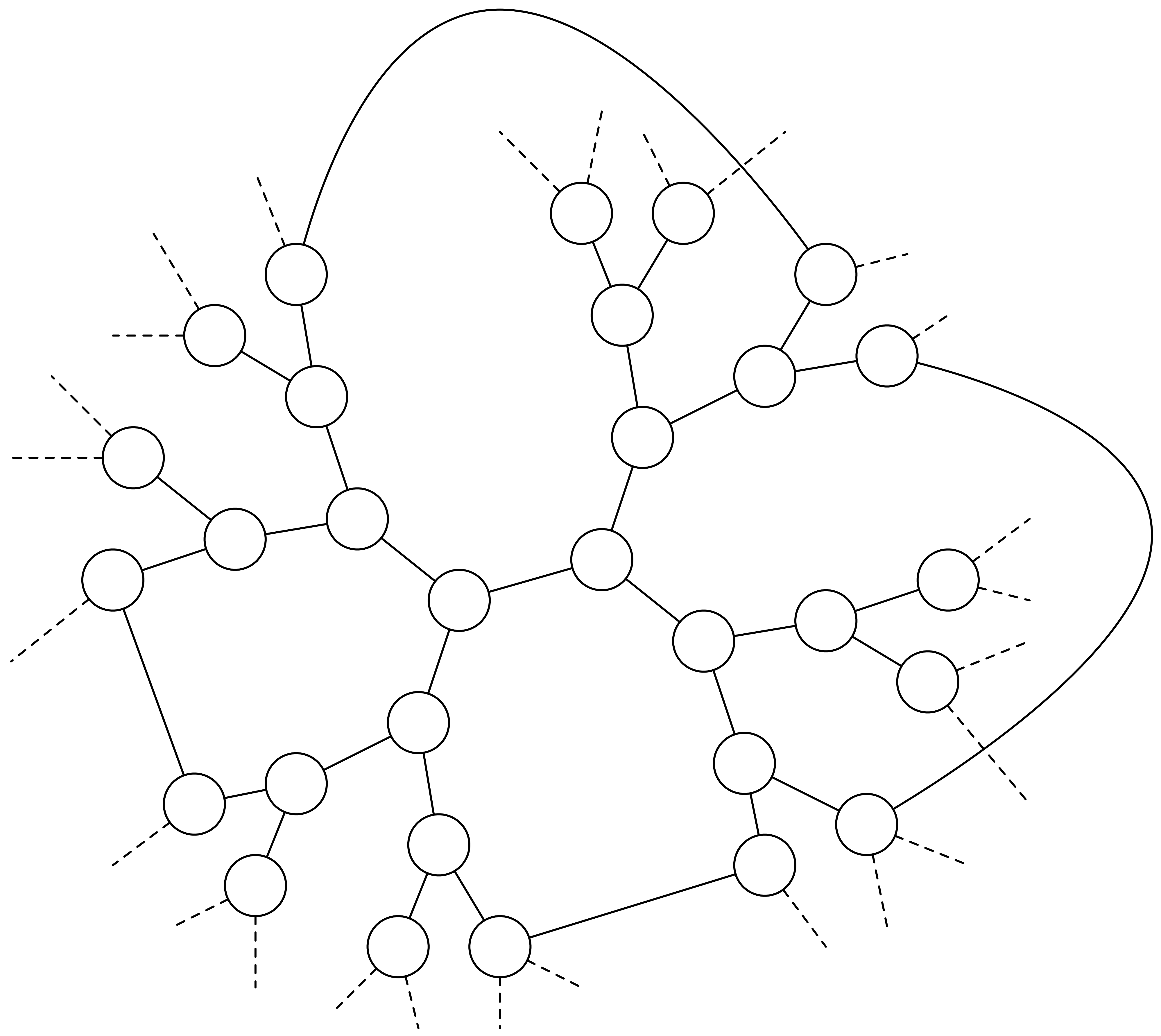
Coloring trees lower bound

- We want to prove that $o(\Delta / \log \Delta)$ coloring trees of maximum degree Δ 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 $\Omega(\Delta / \log \Delta)$

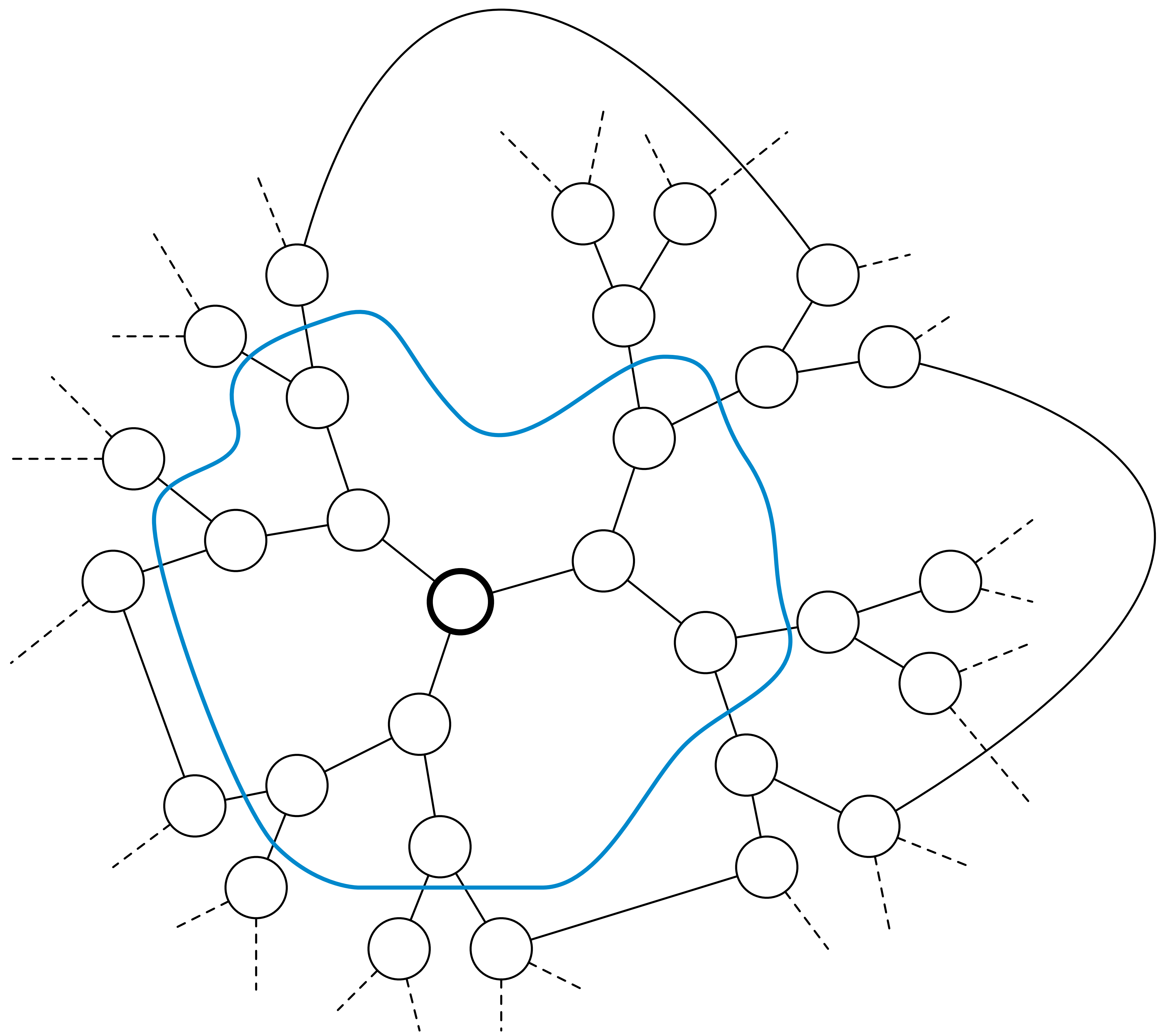
Coloring trees lower bound

- We want to prove that **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **Δ** 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**

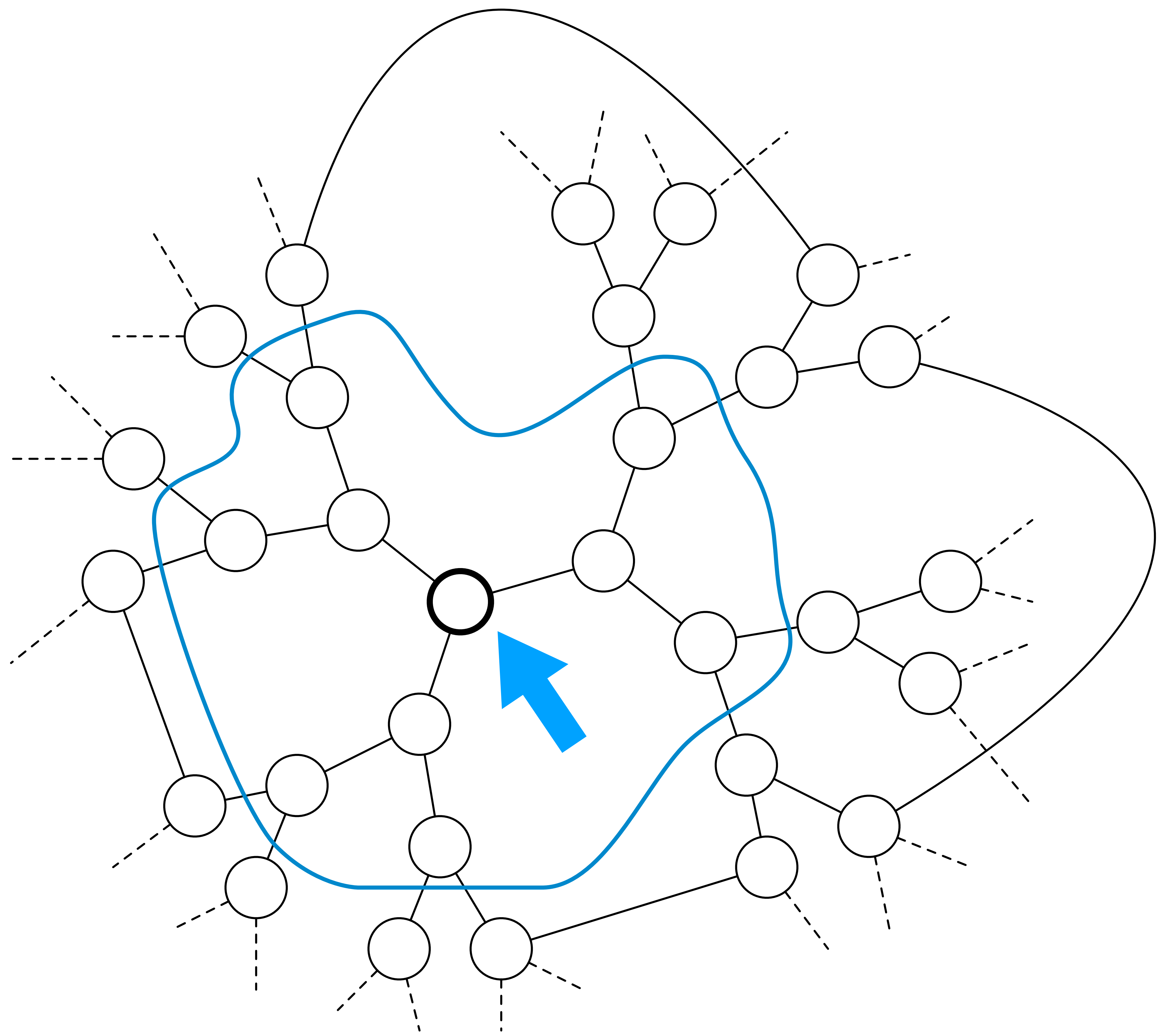
Coloring trees lower bound



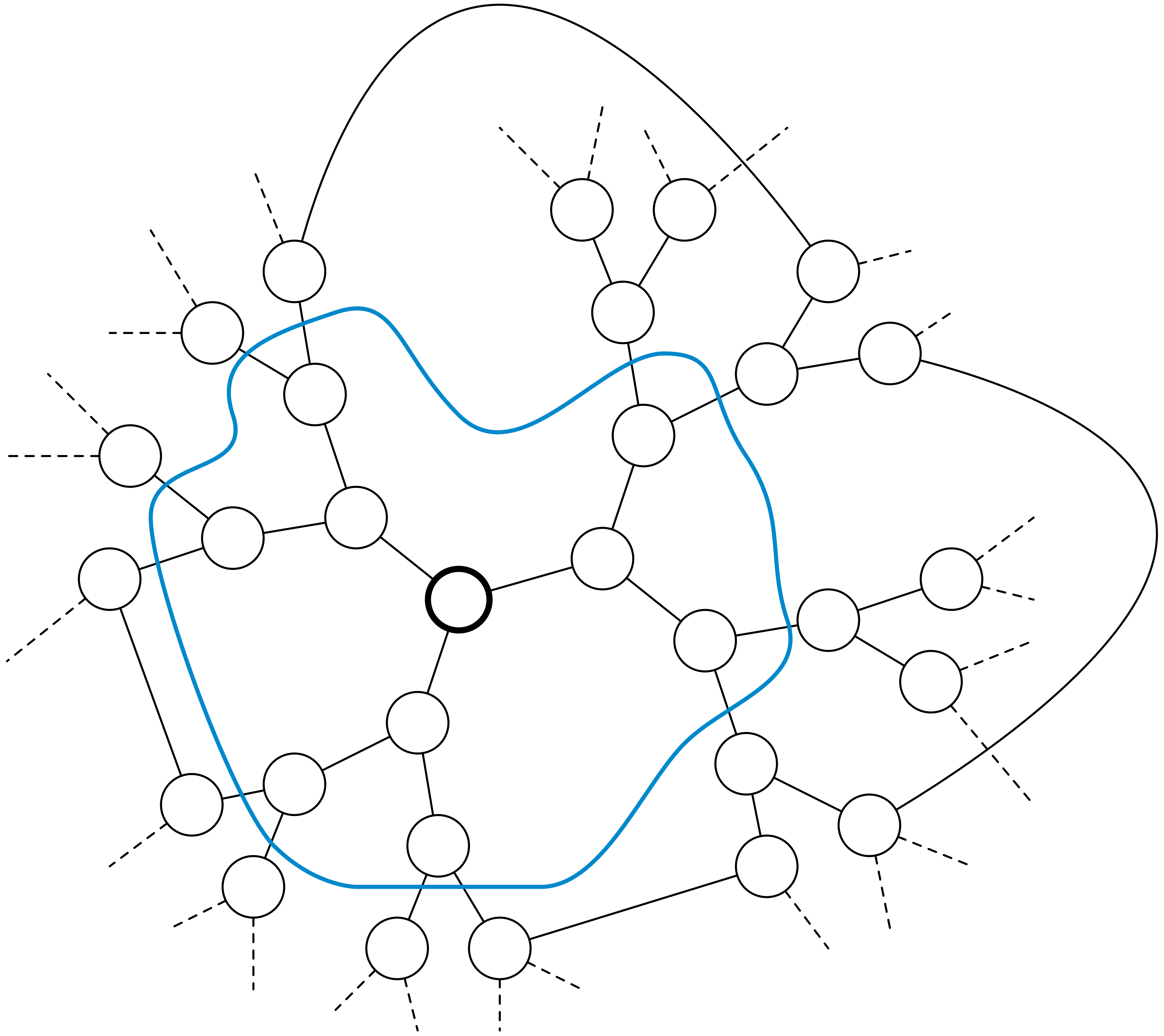
Coloring trees lower bound



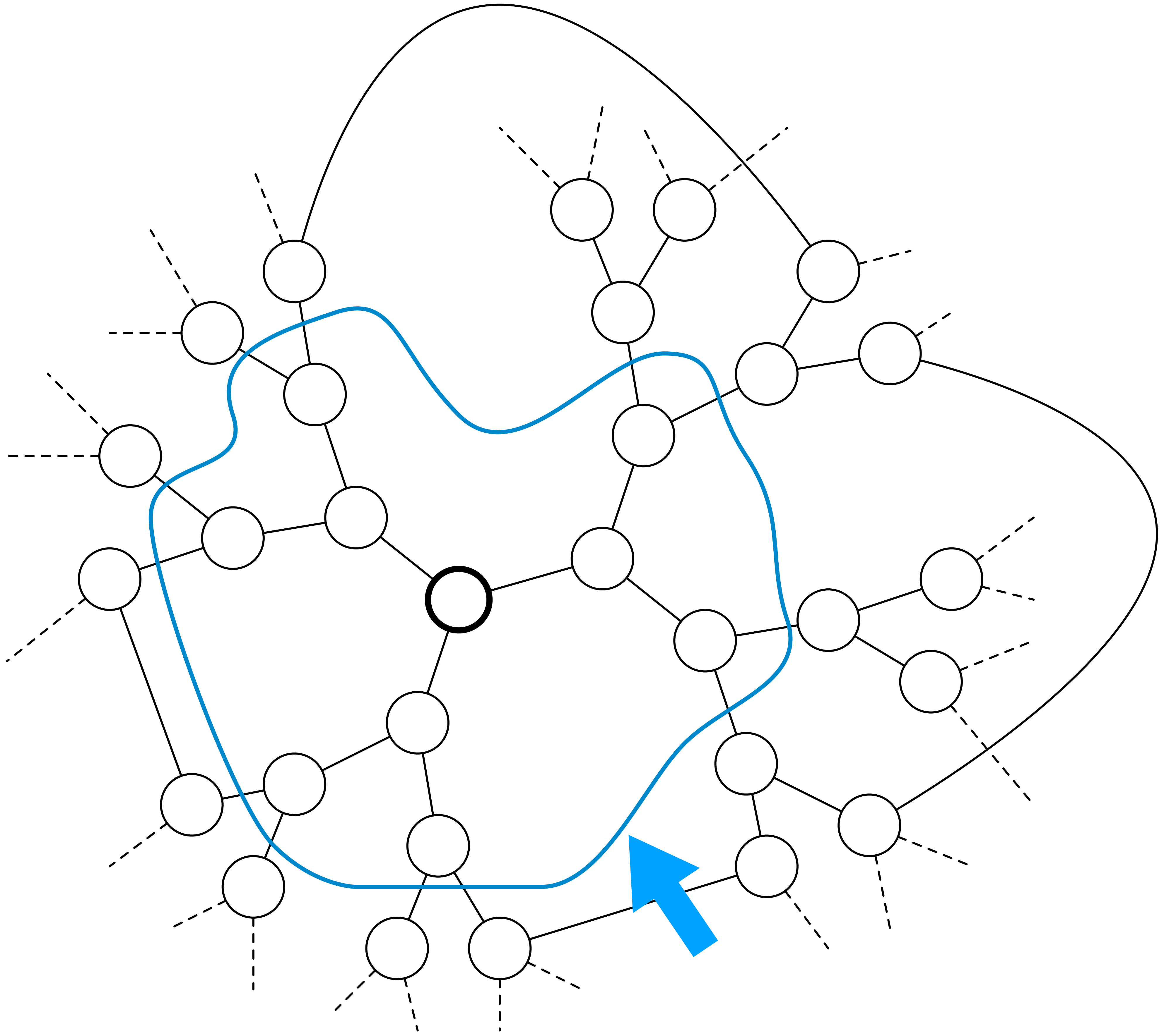
Coloring trees lower bound



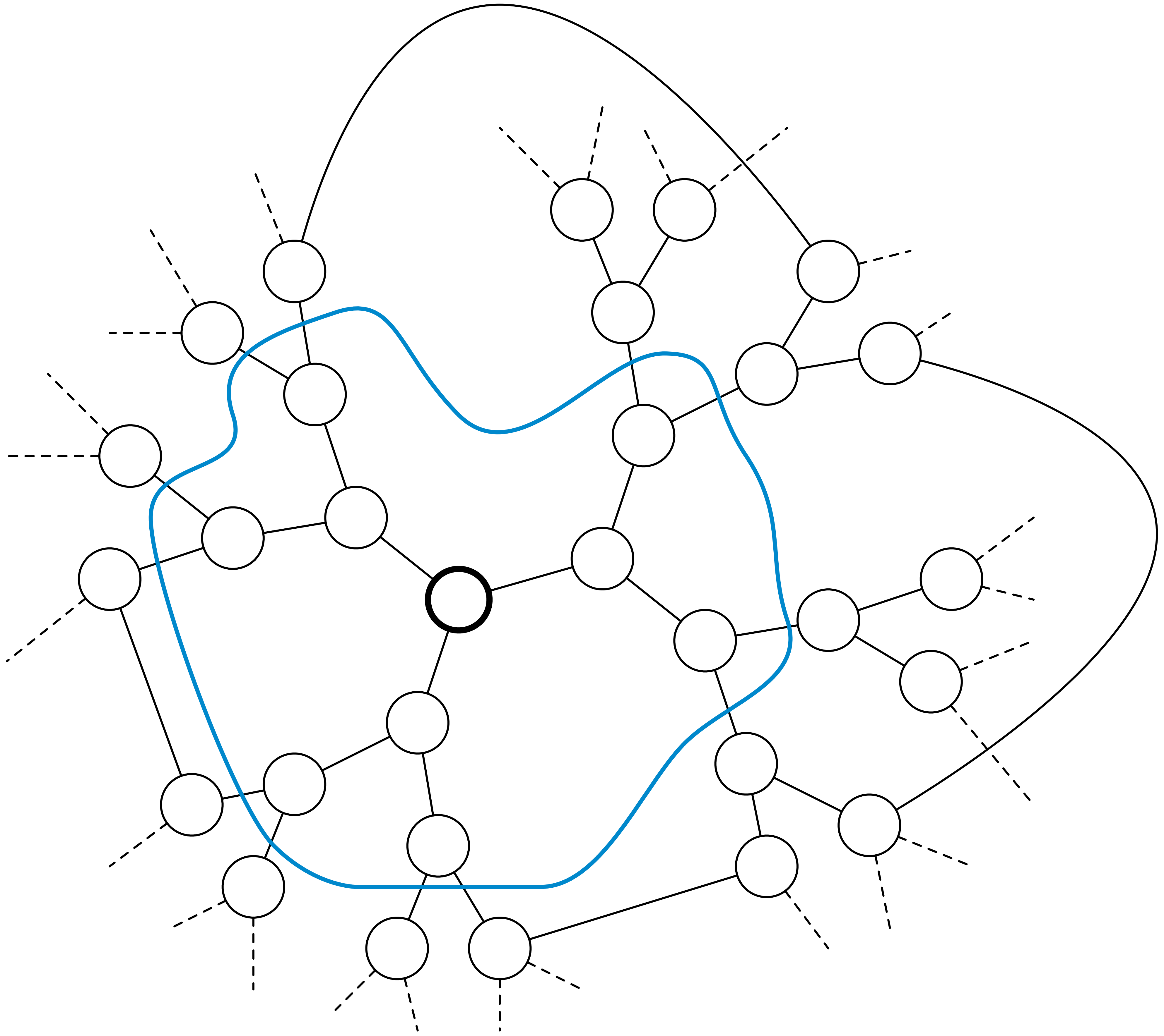
Coloring trees lower bound



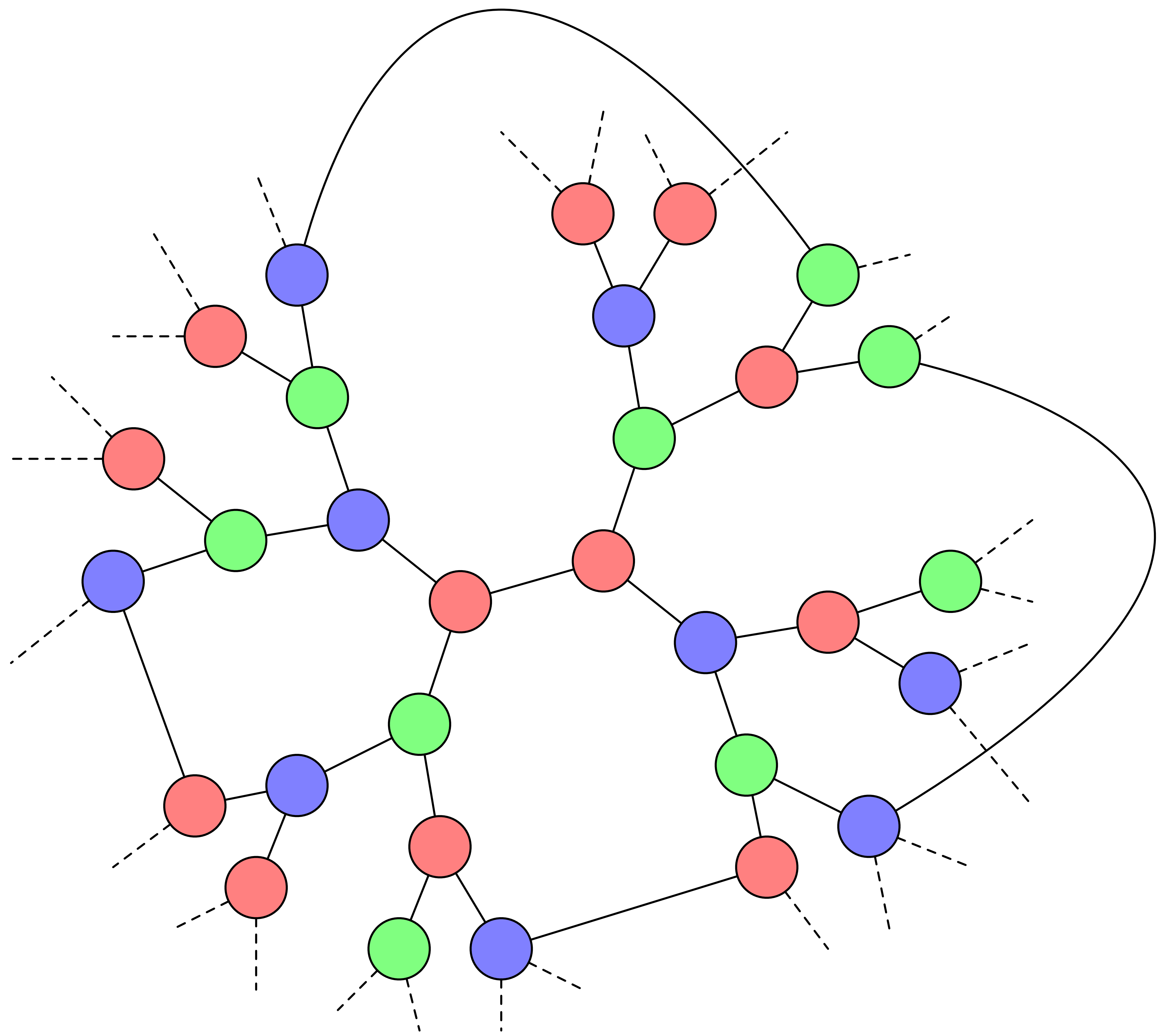
Coloring trees lower bound



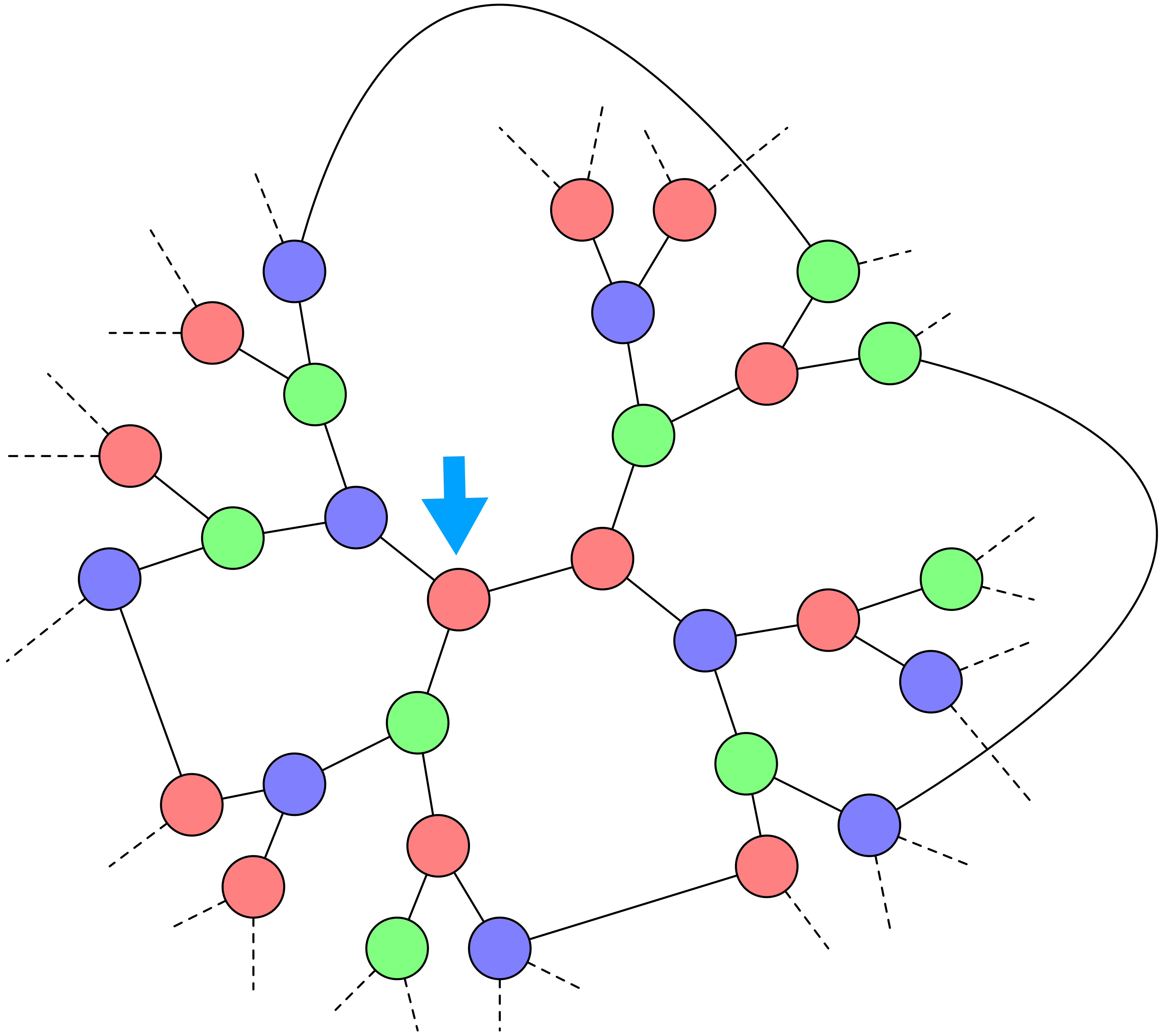
Coloring trees lower bound



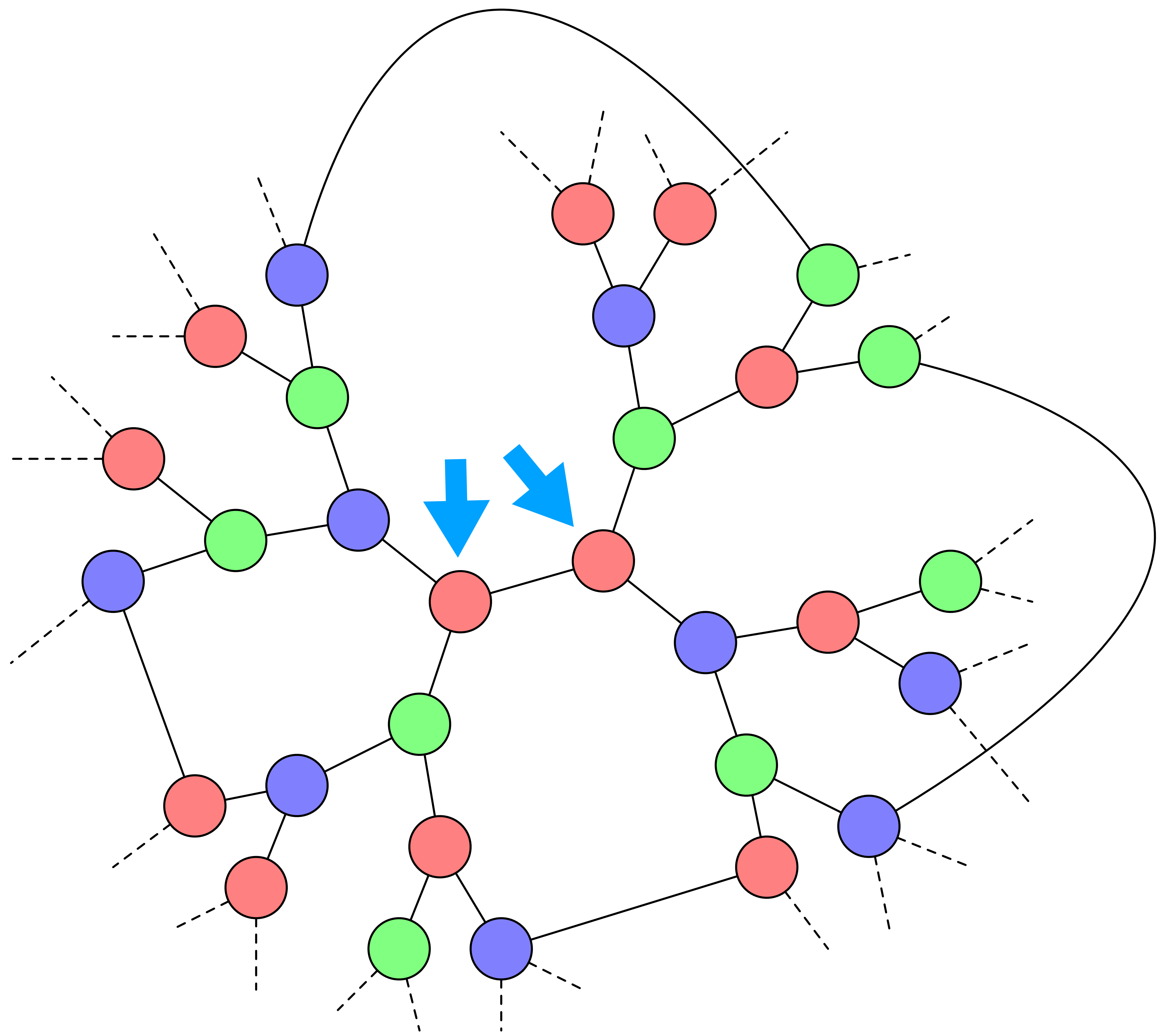
Coloring trees lower bound



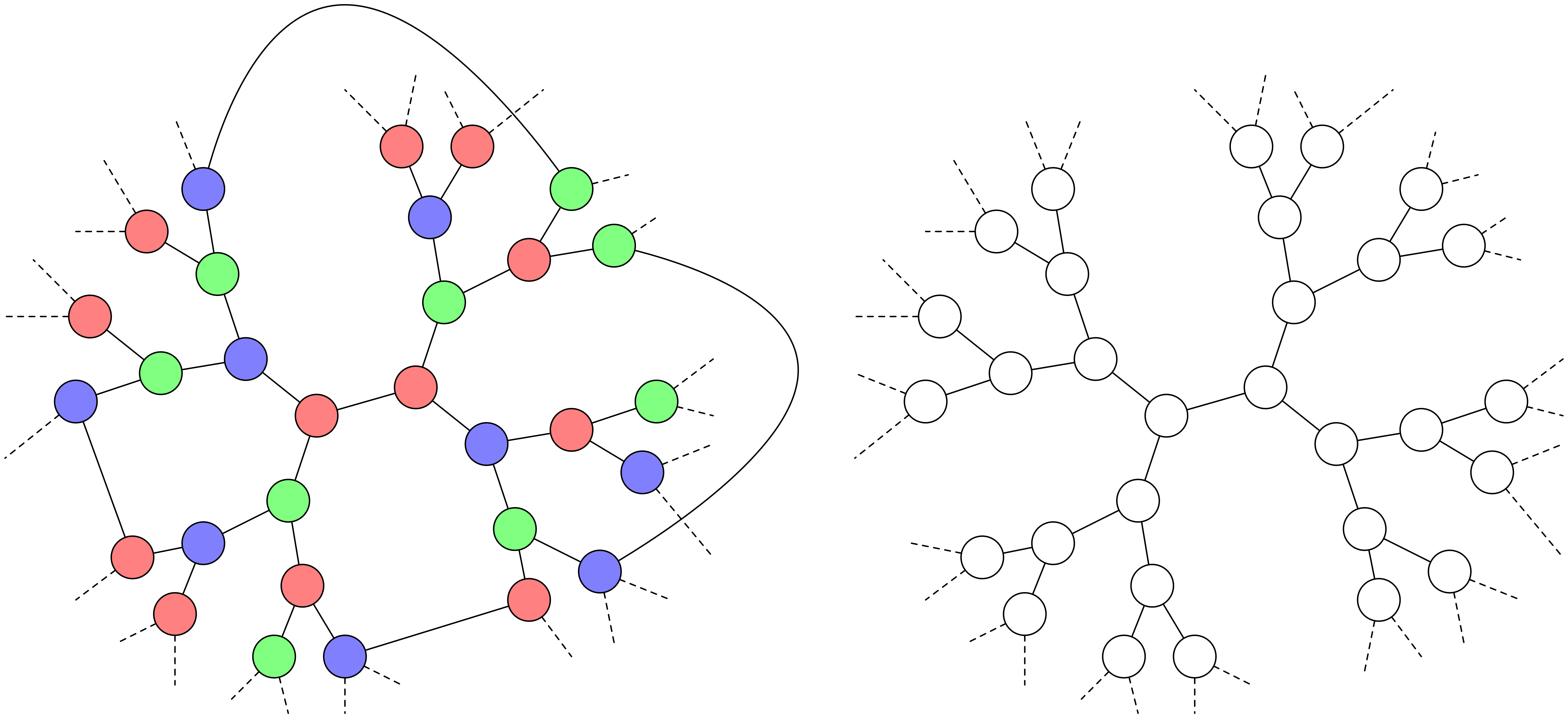
Coloring trees lower bound



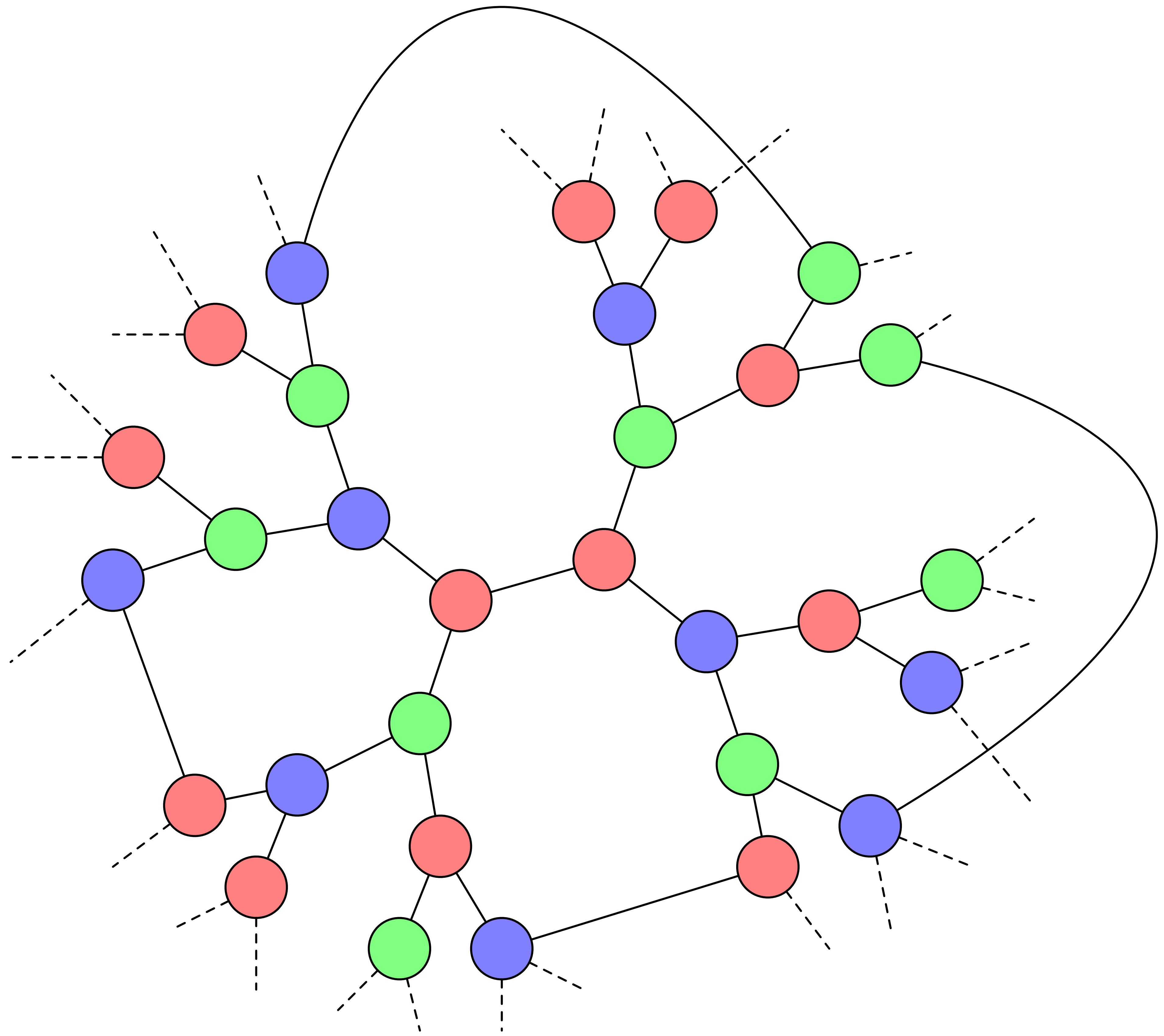
Coloring trees lower bound



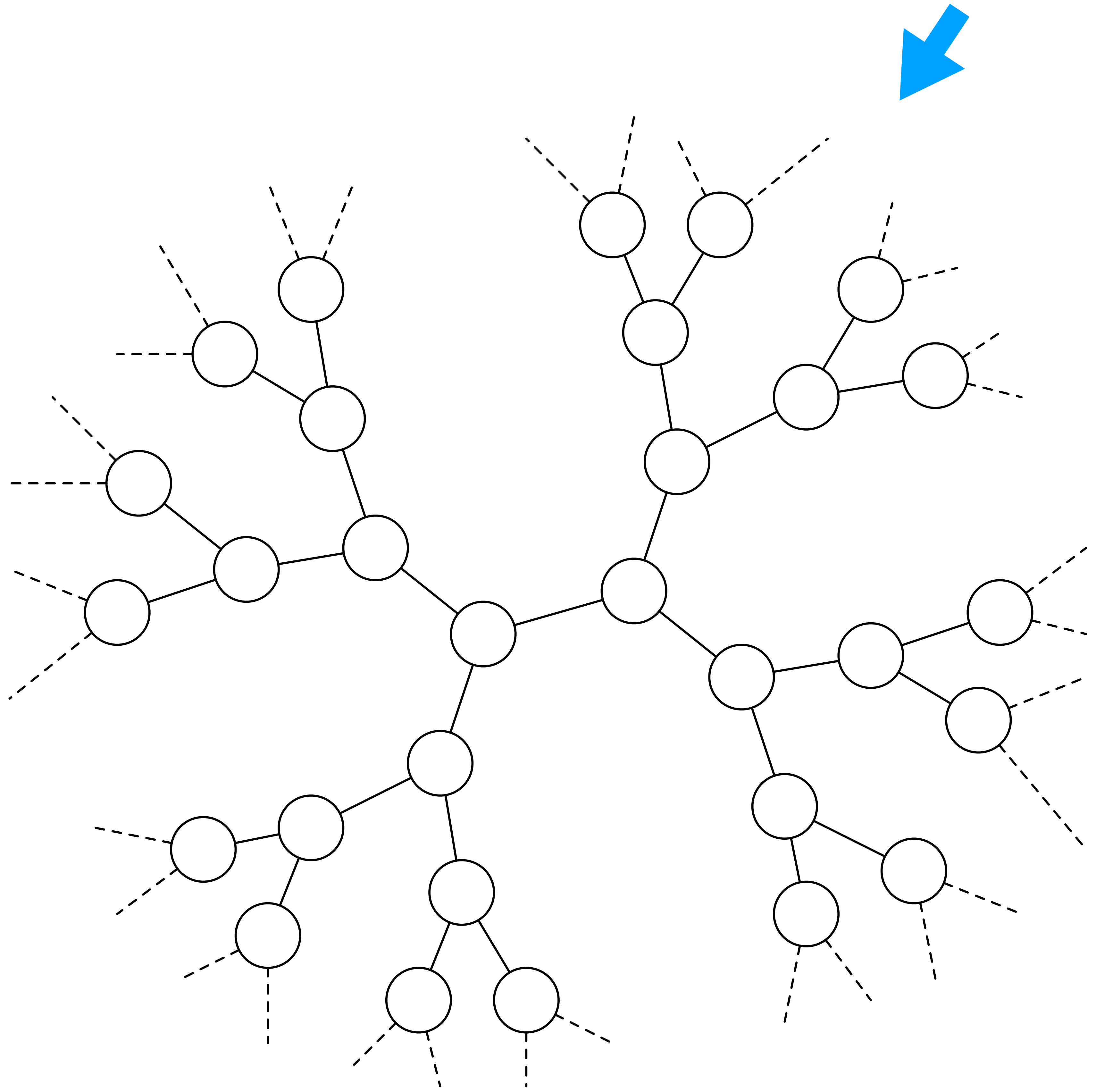
Coloring trees lower bound



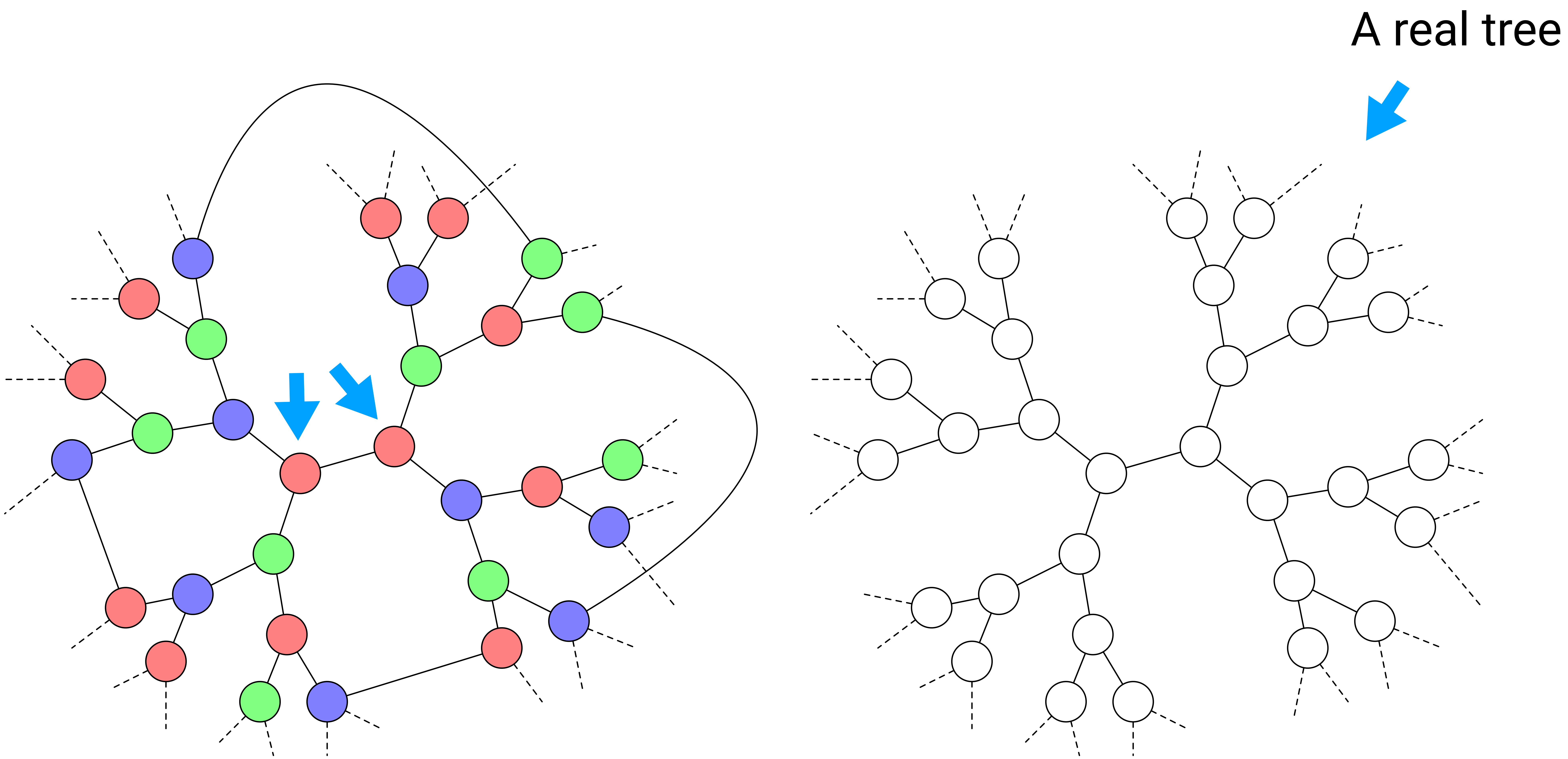
Coloring trees lower bound



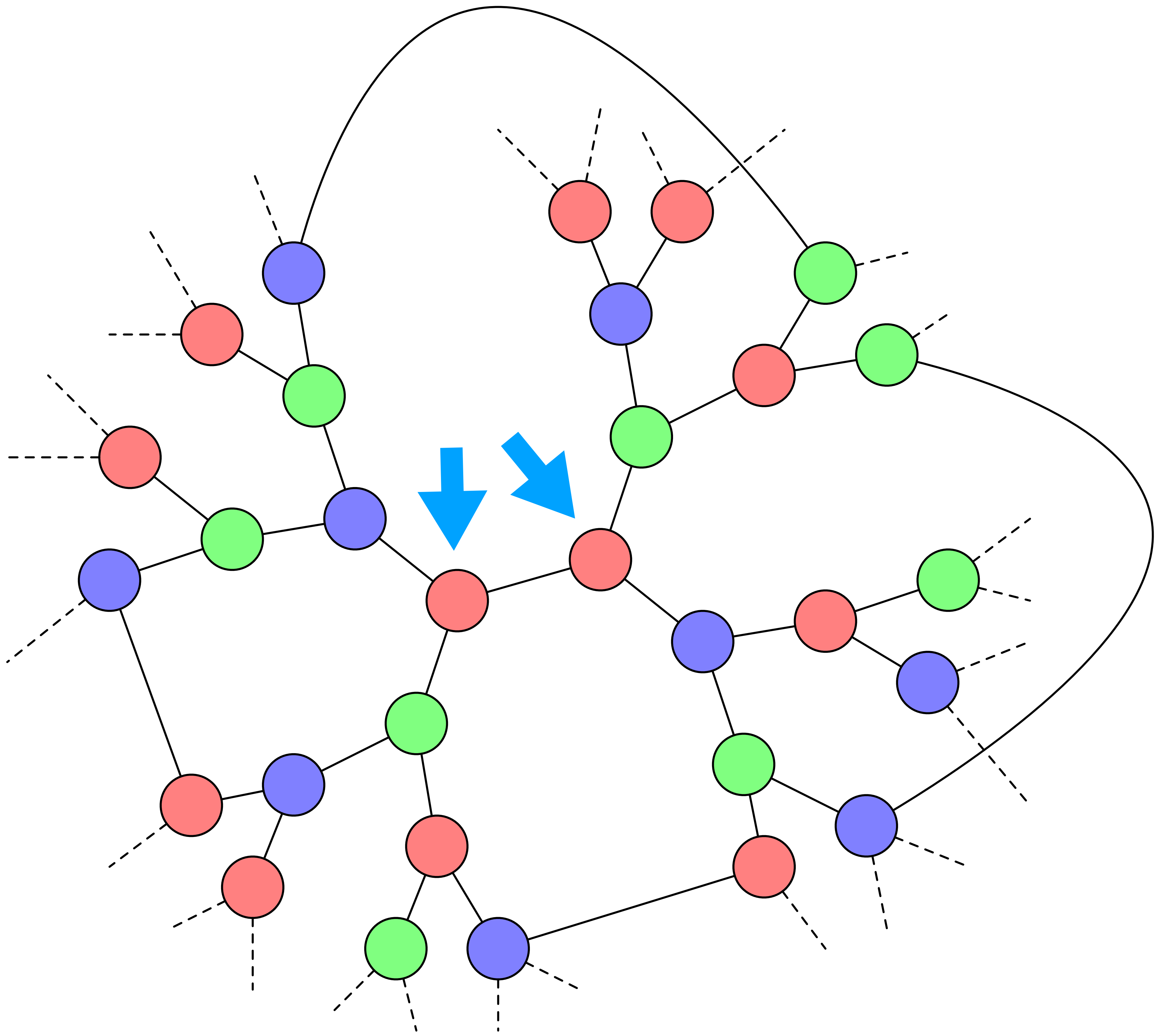
A real tree

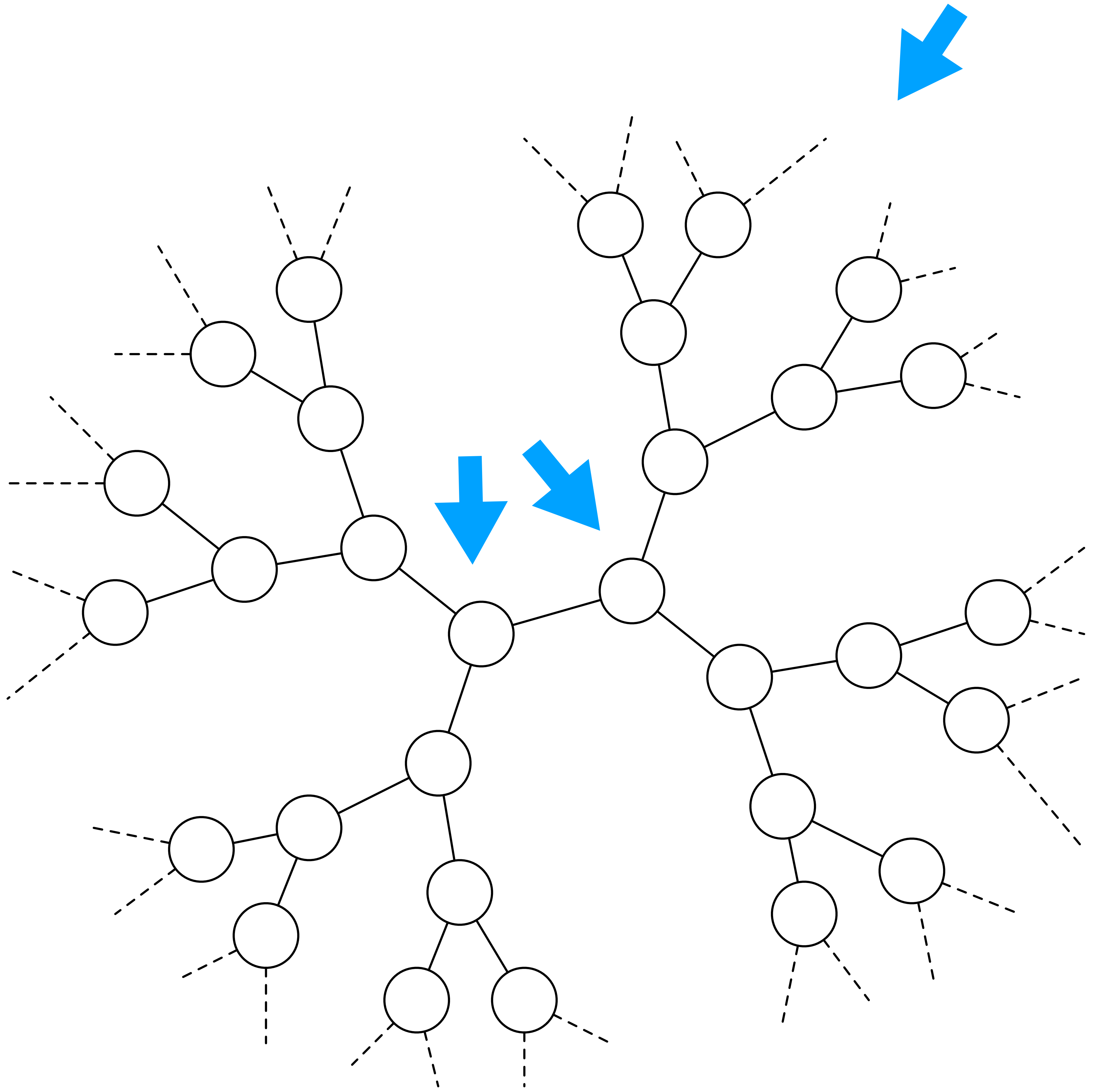


Coloring trees lower bound

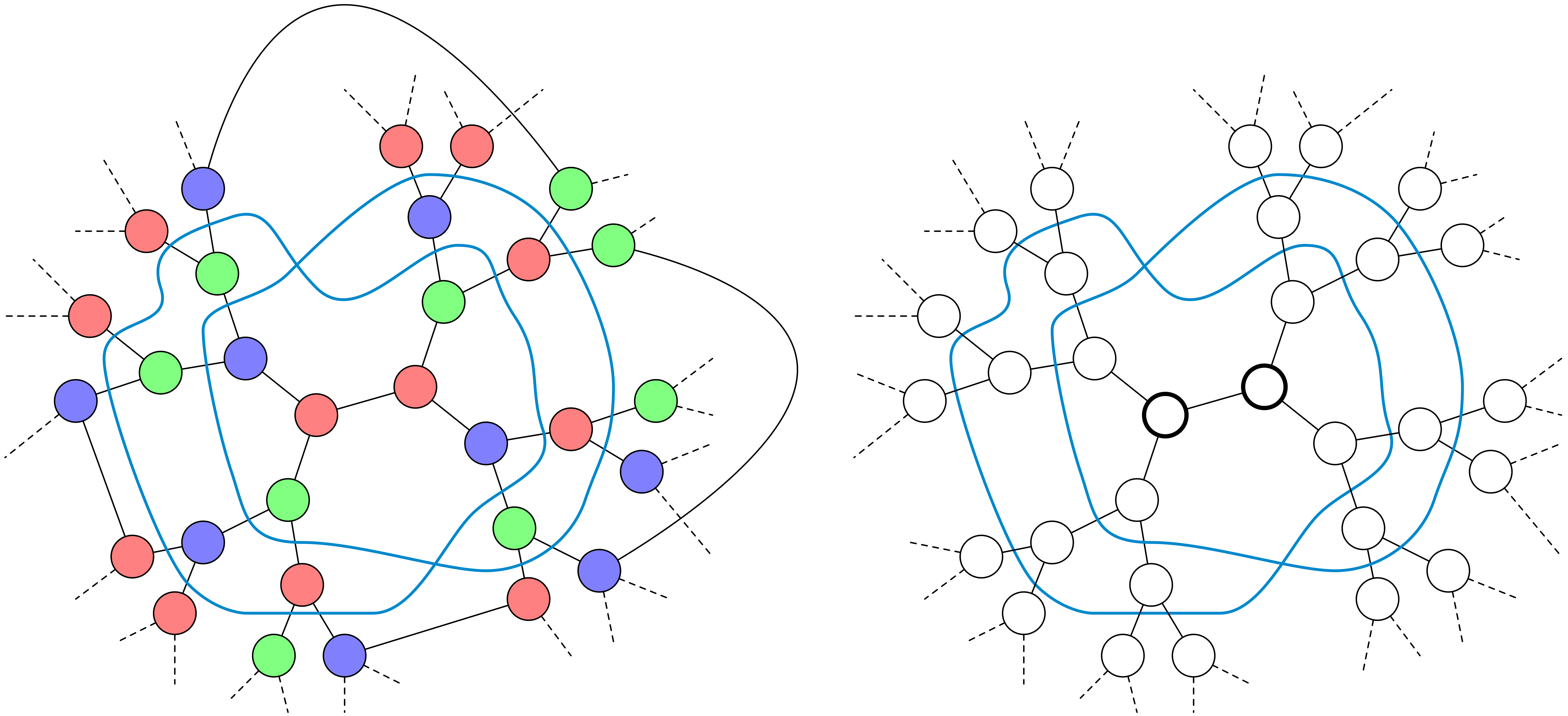


Coloring trees lower bound

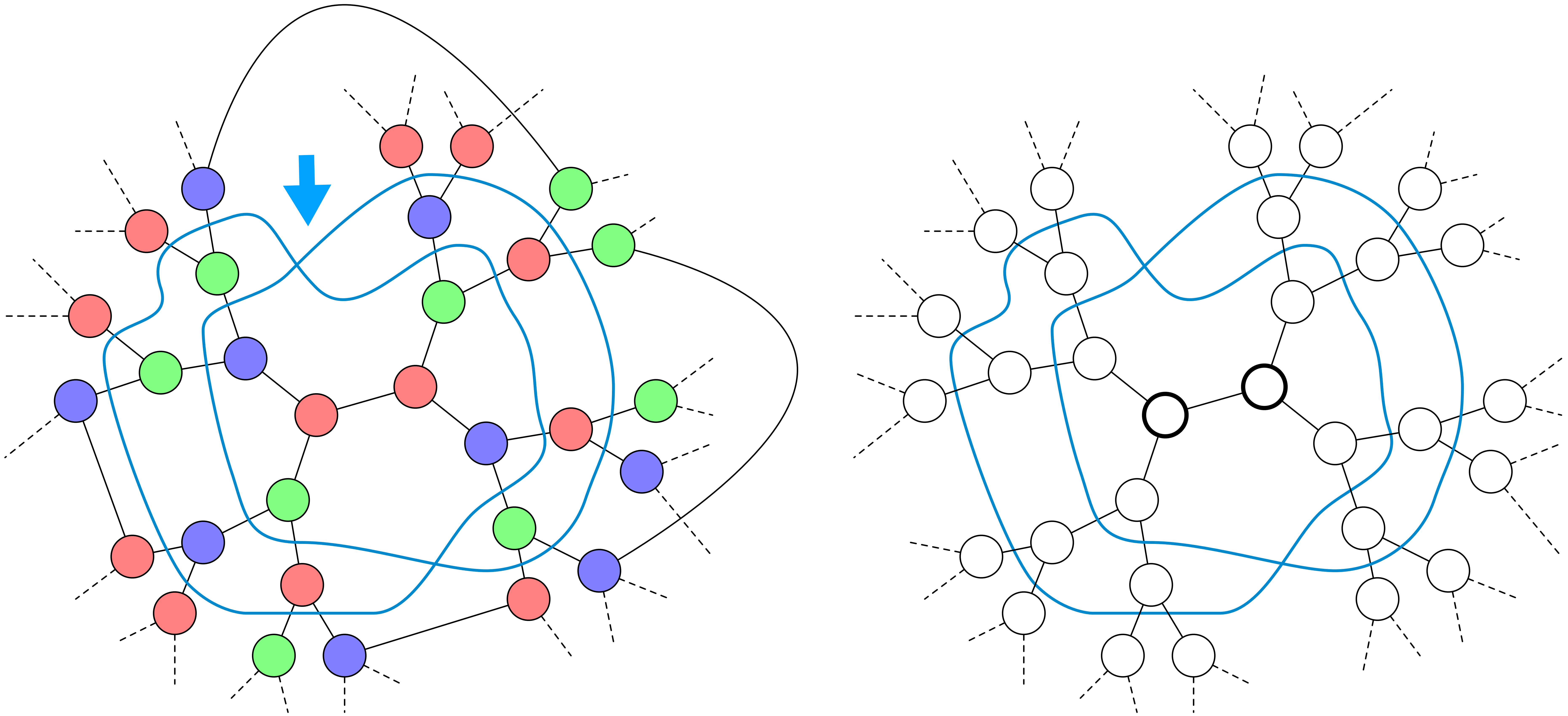




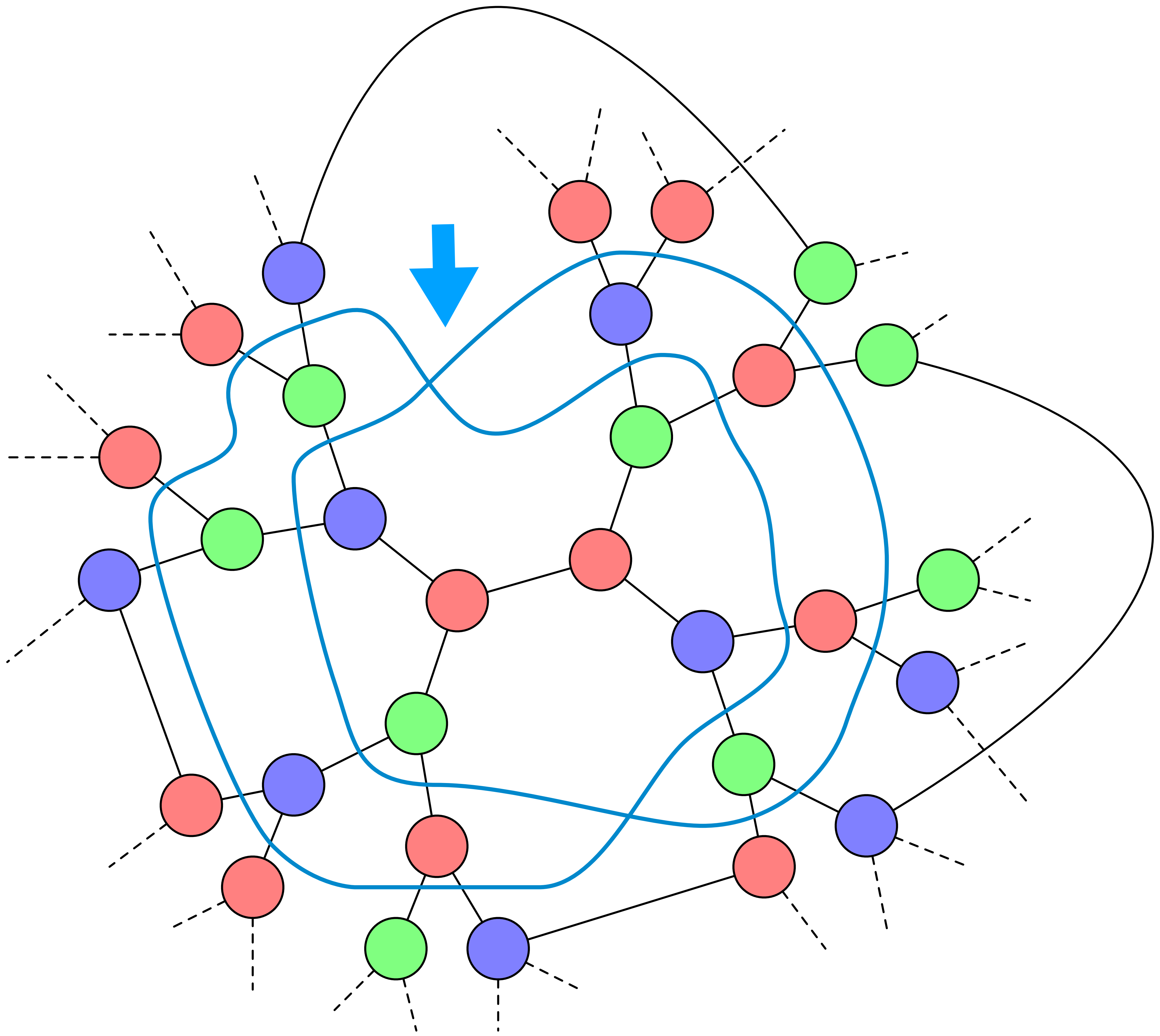
Coloring trees lower bound

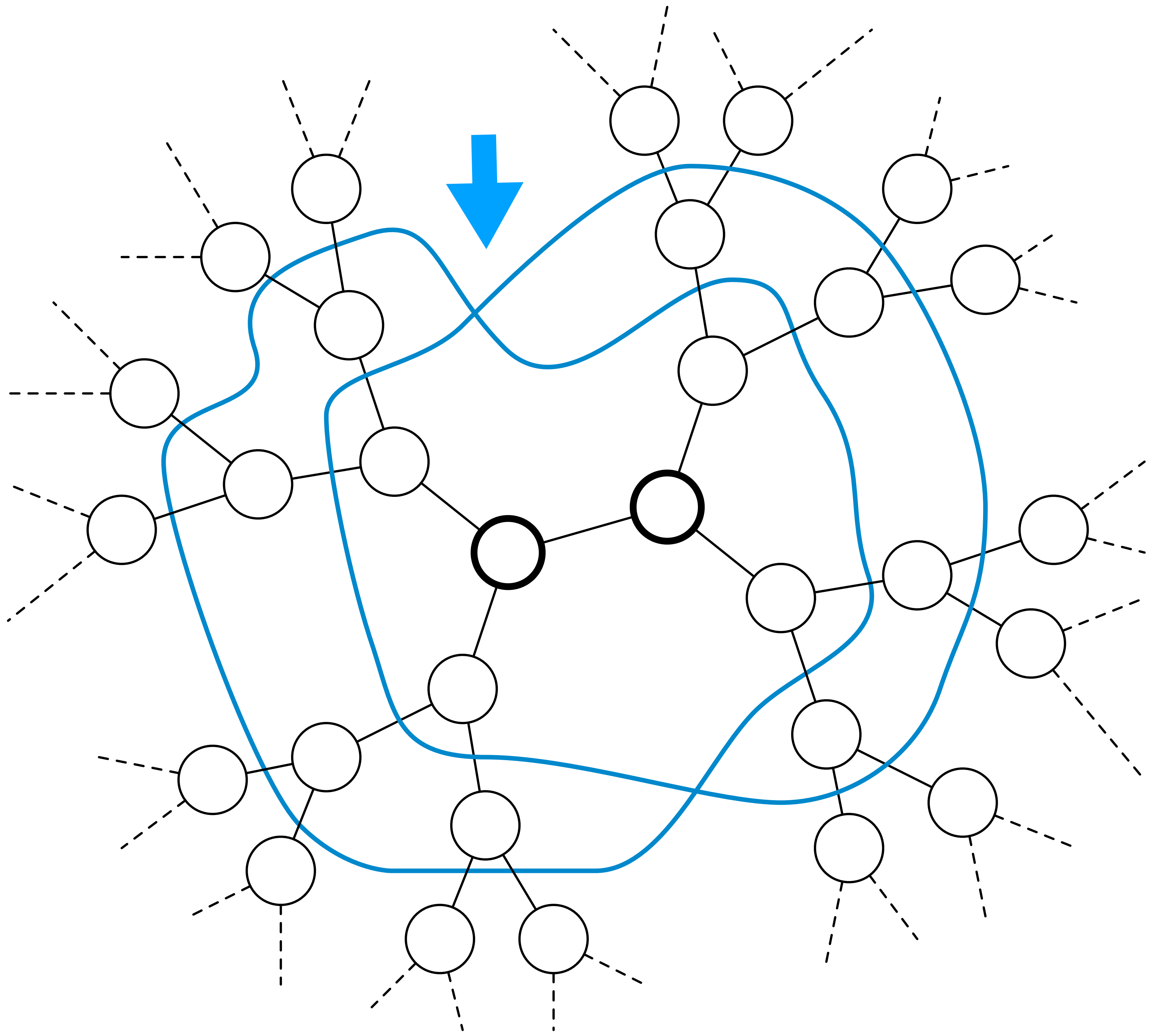


Coloring trees lower bound

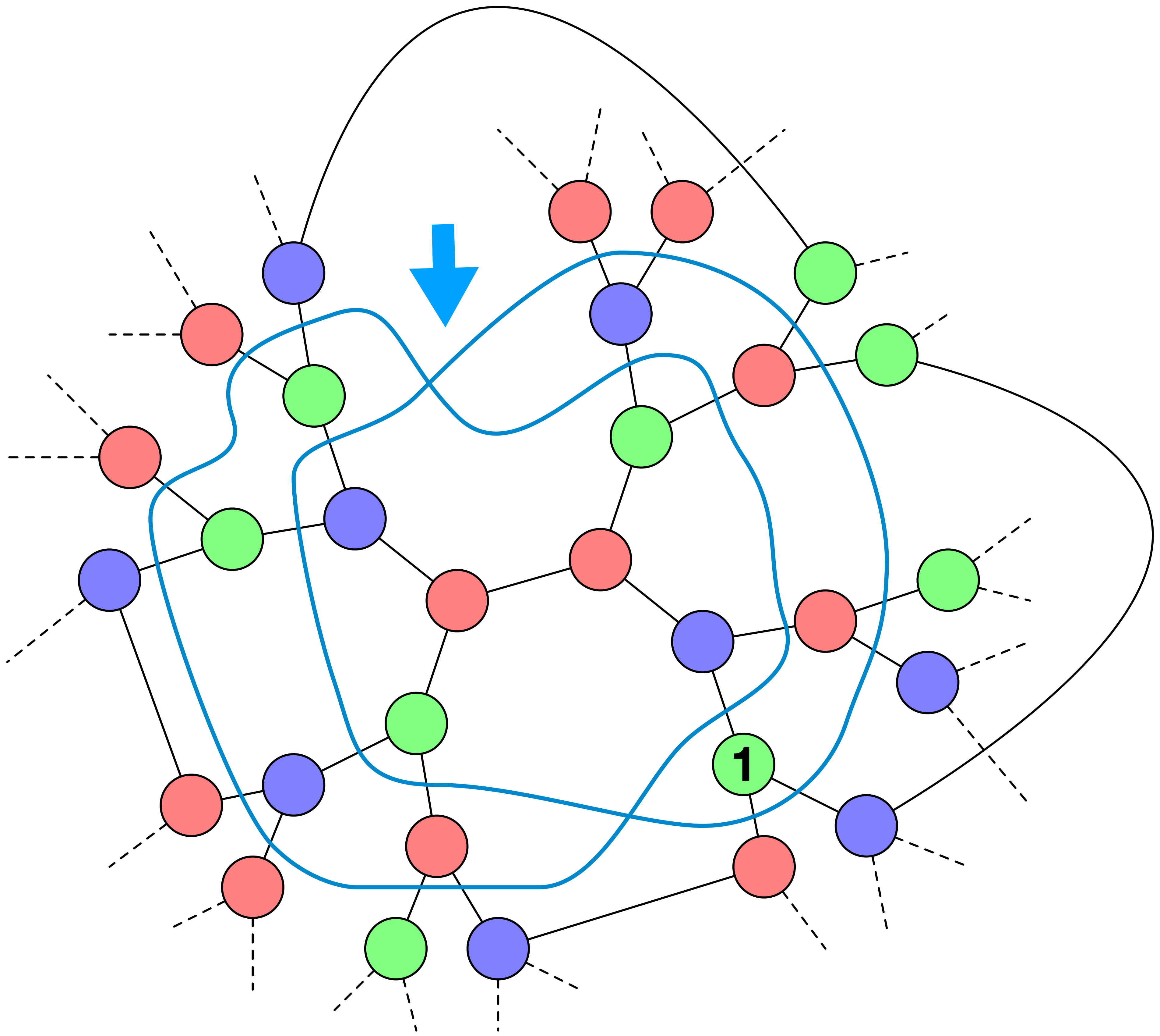


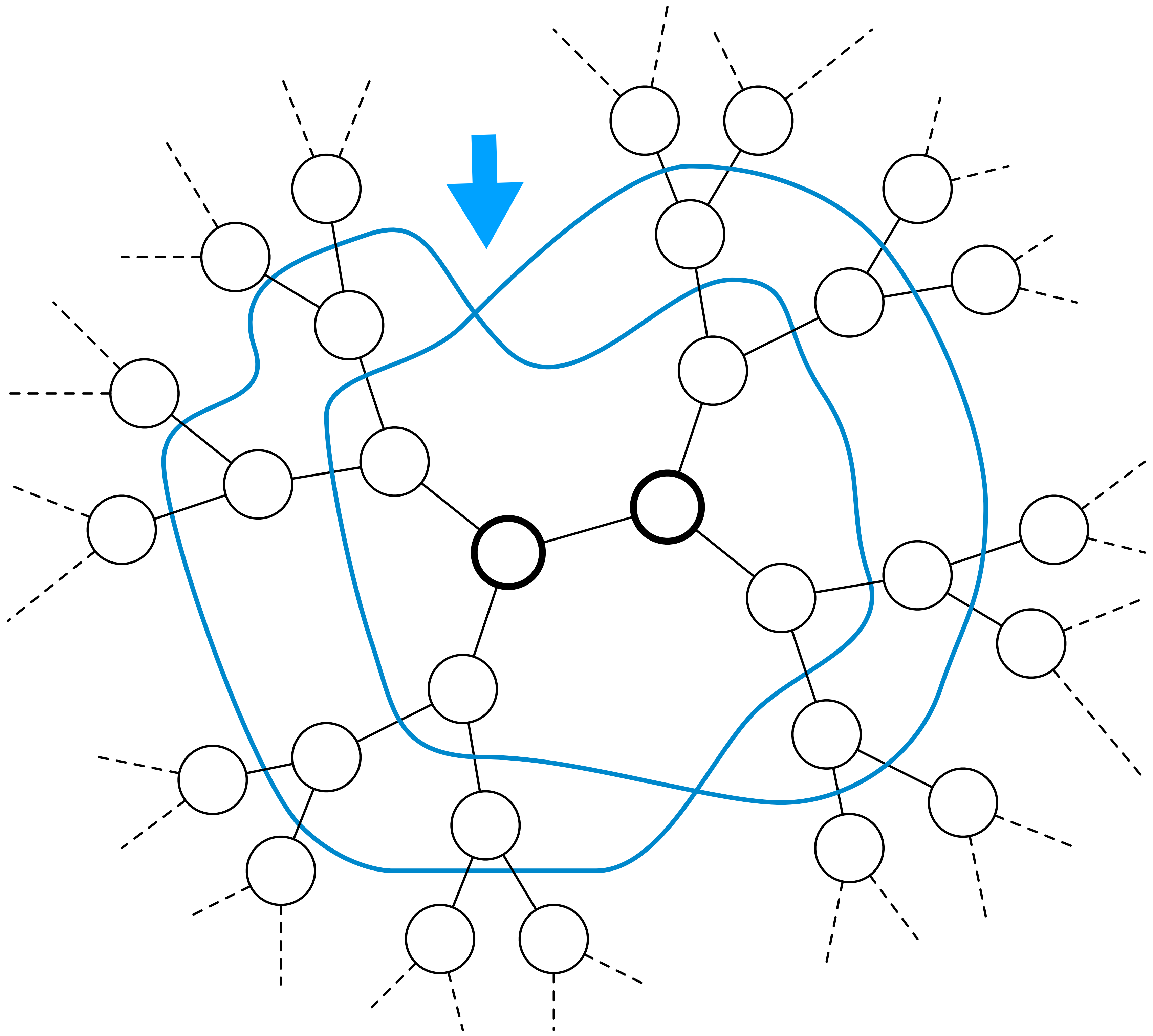
Coloring trees lower bound



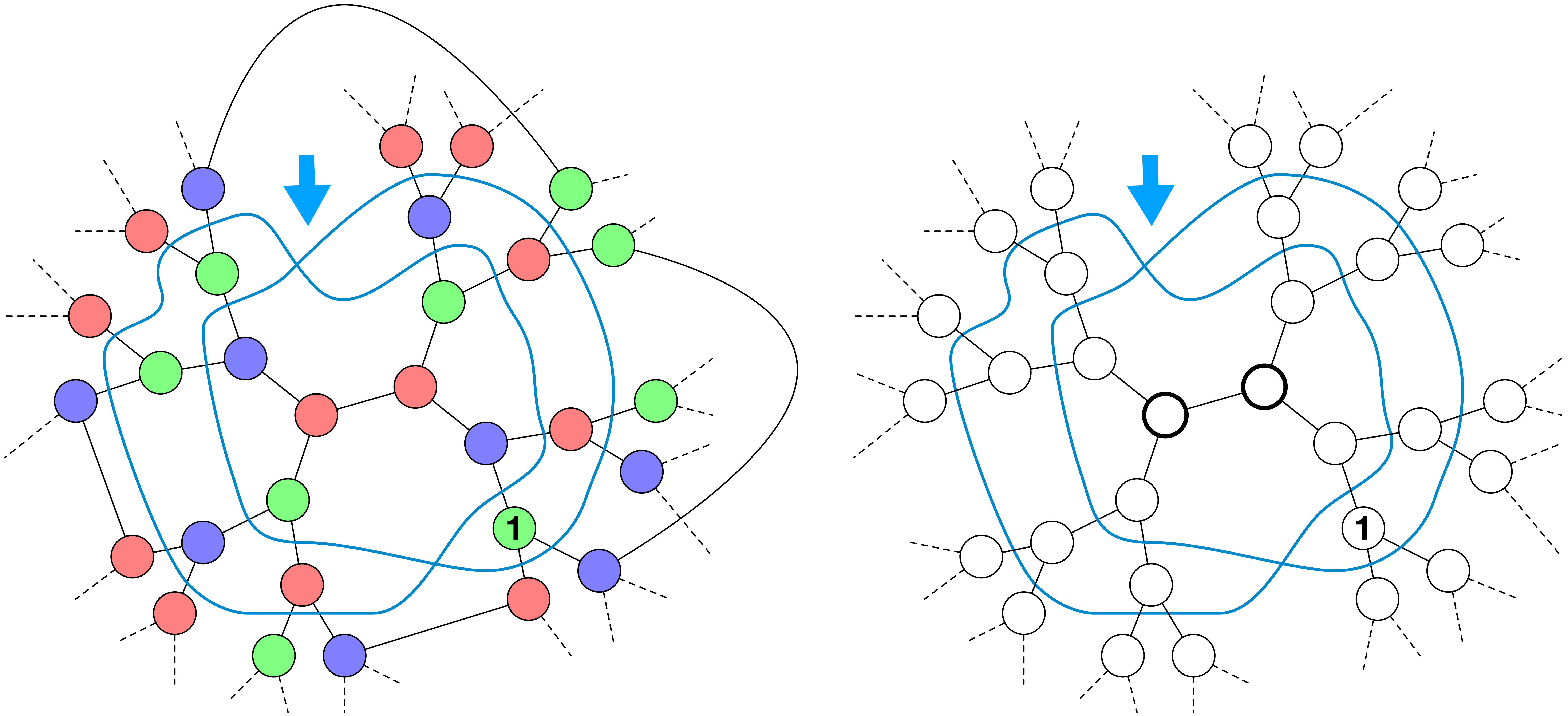


Coloring trees lower bound

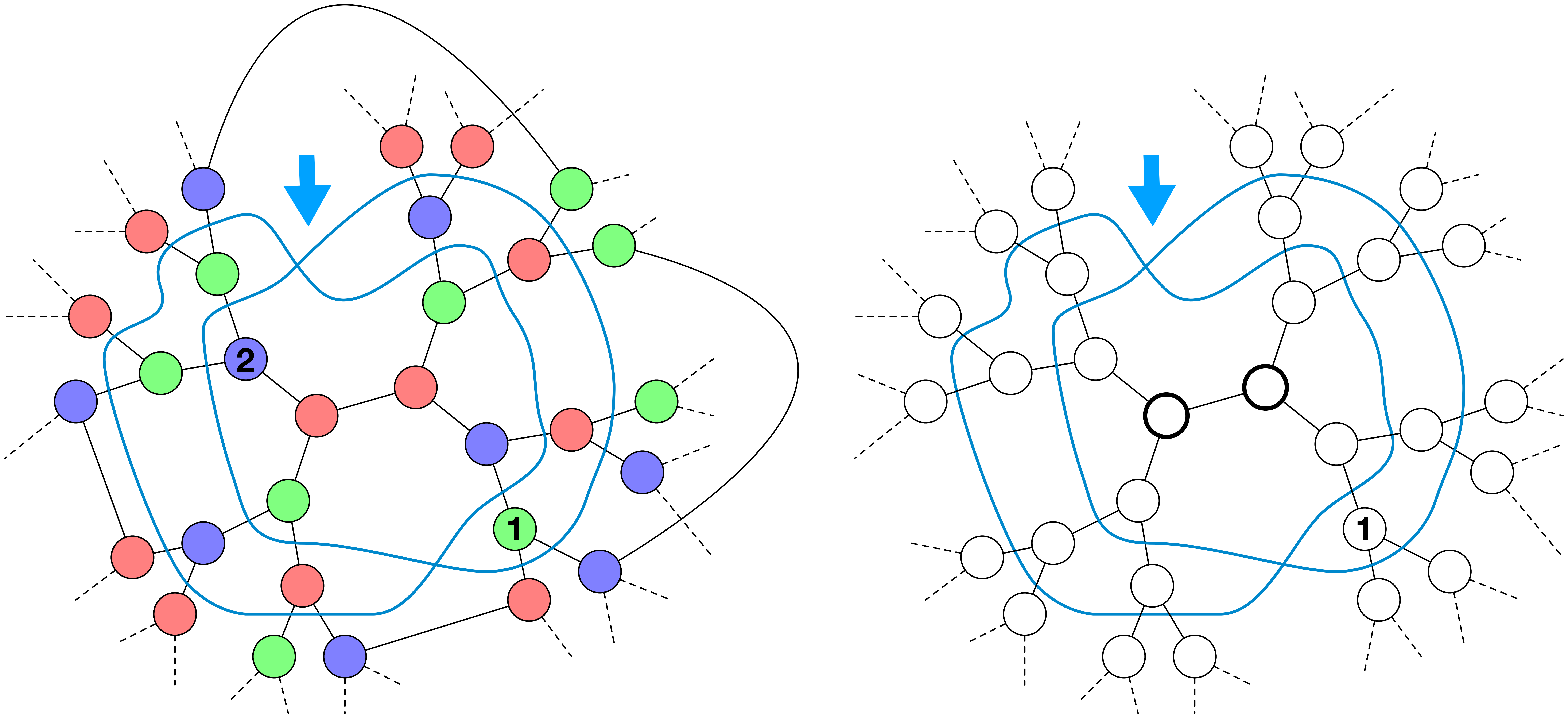




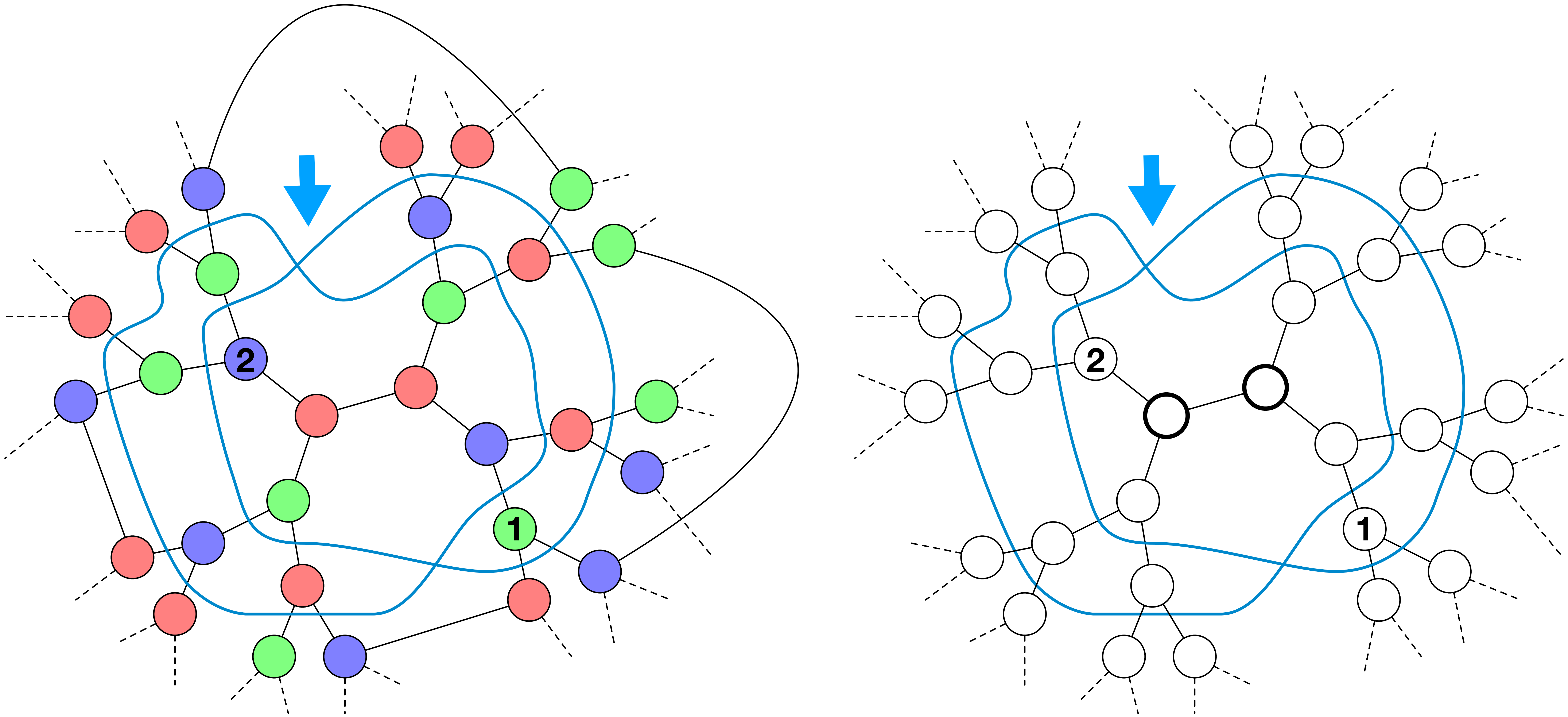
Coloring trees lower bound



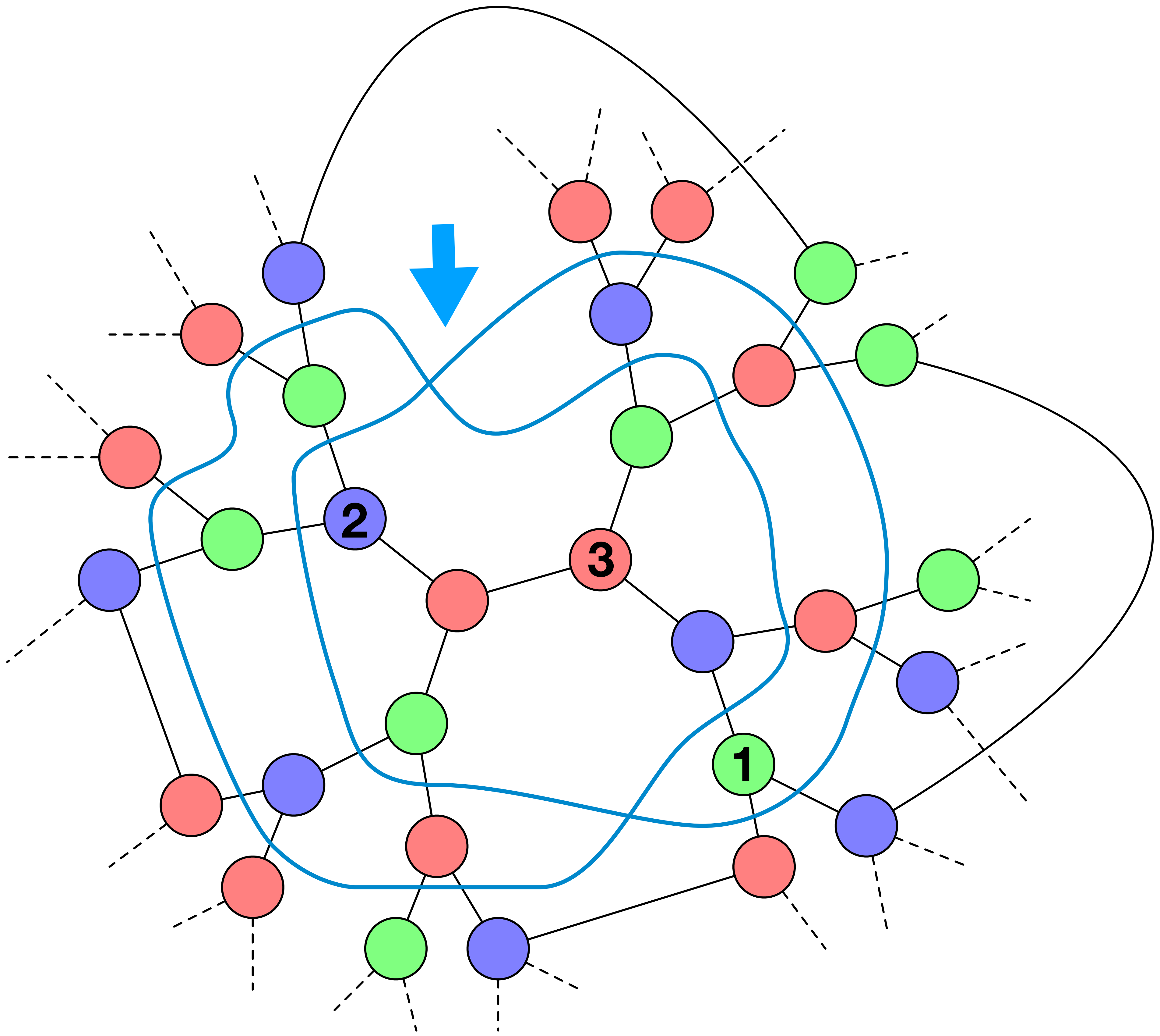
Coloring trees lower bound

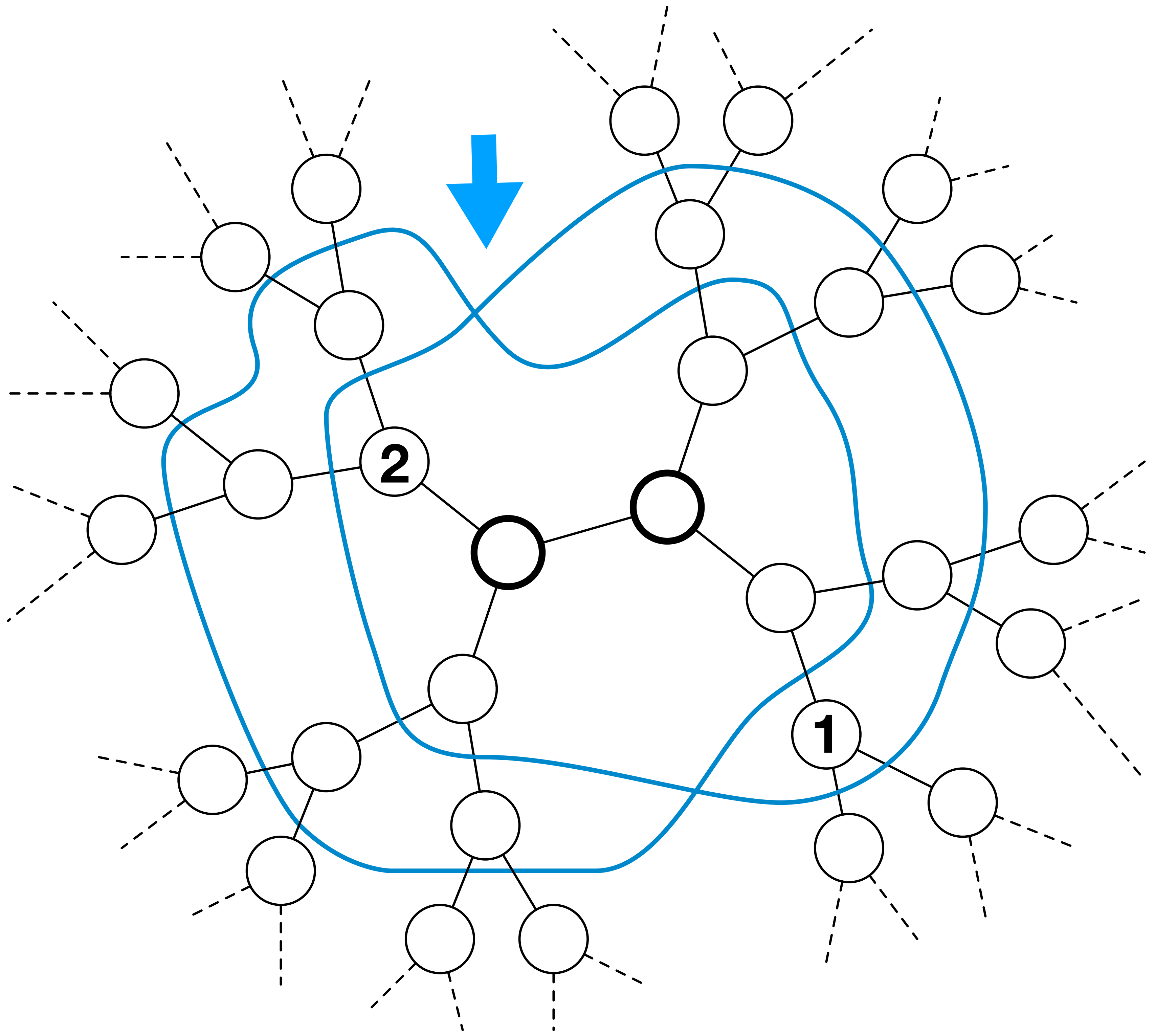


Coloring trees lower bound

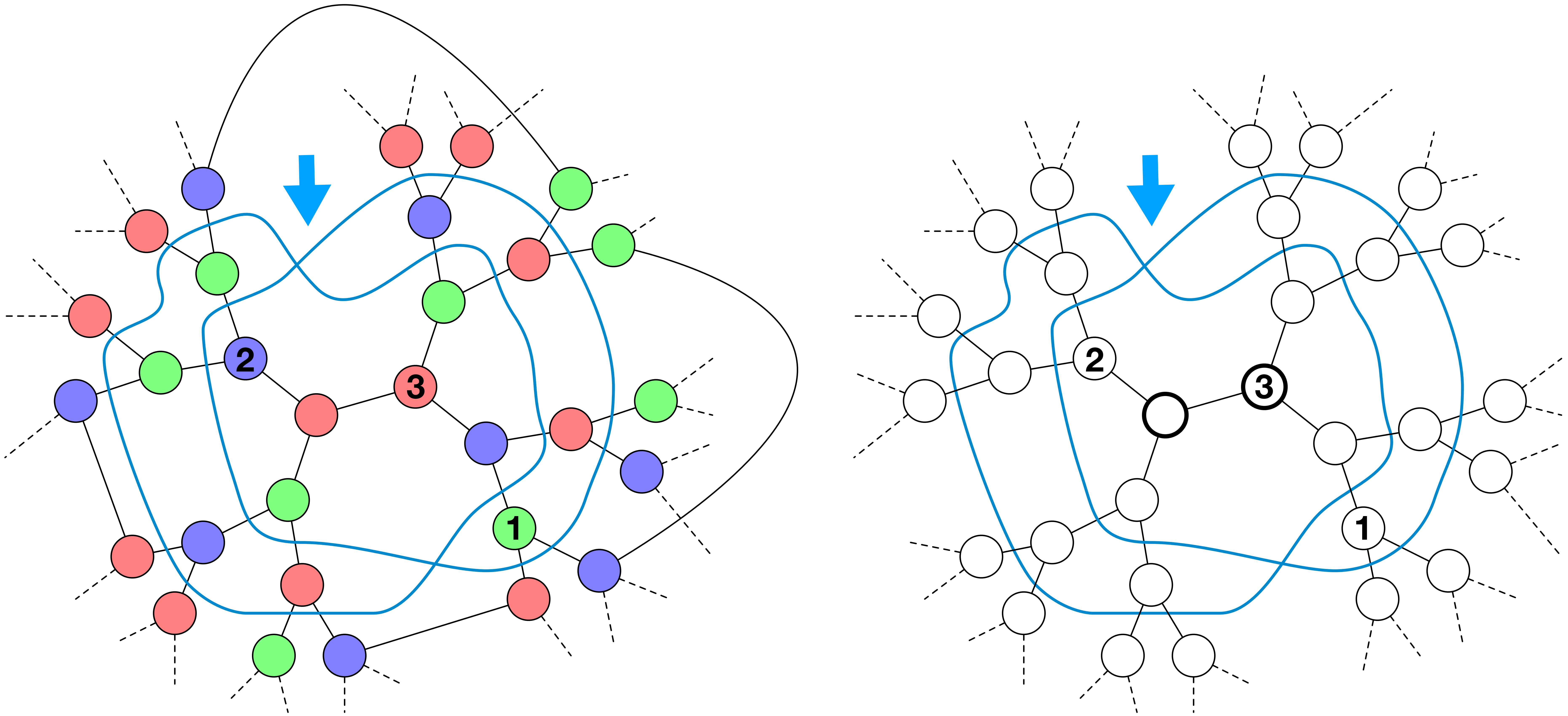


Coloring trees lower bound

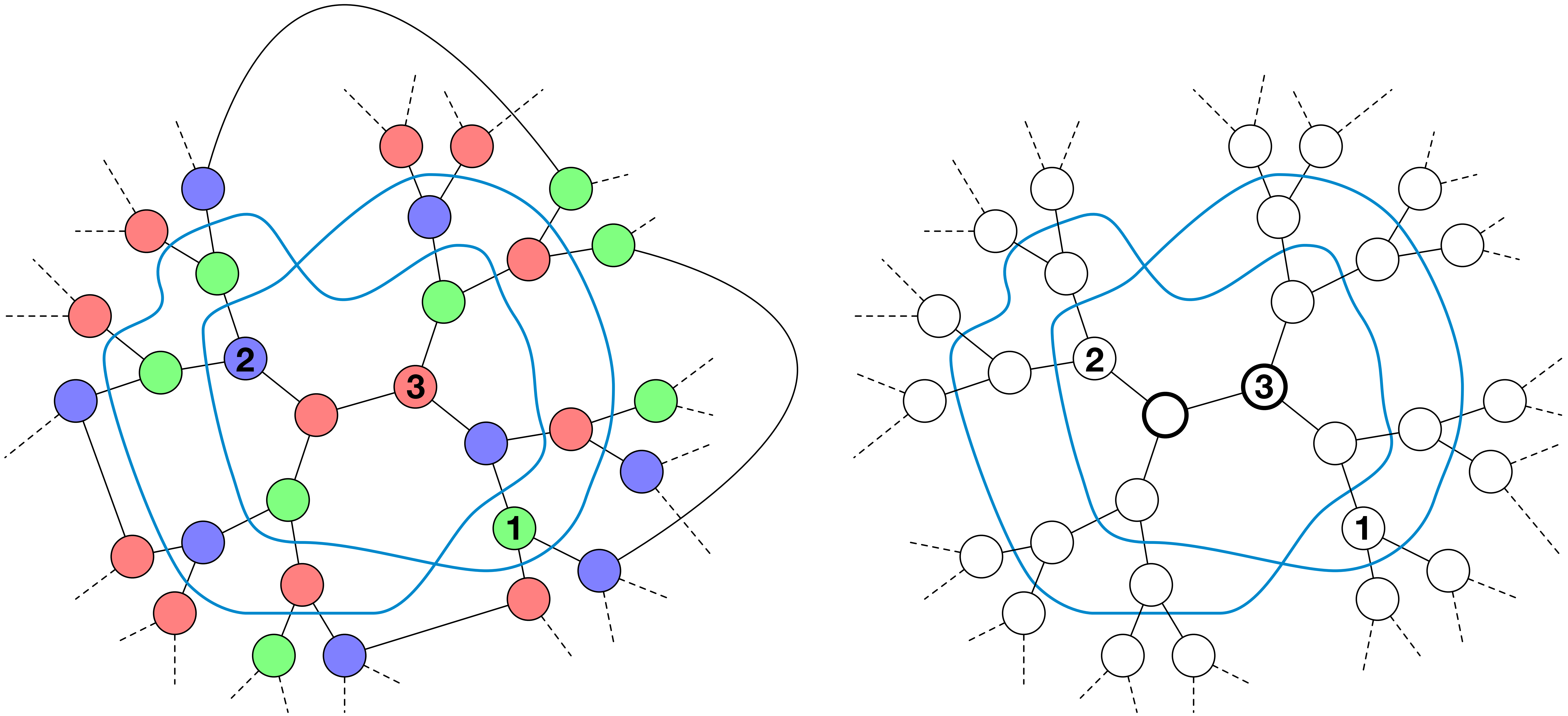




Coloring trees lower bound

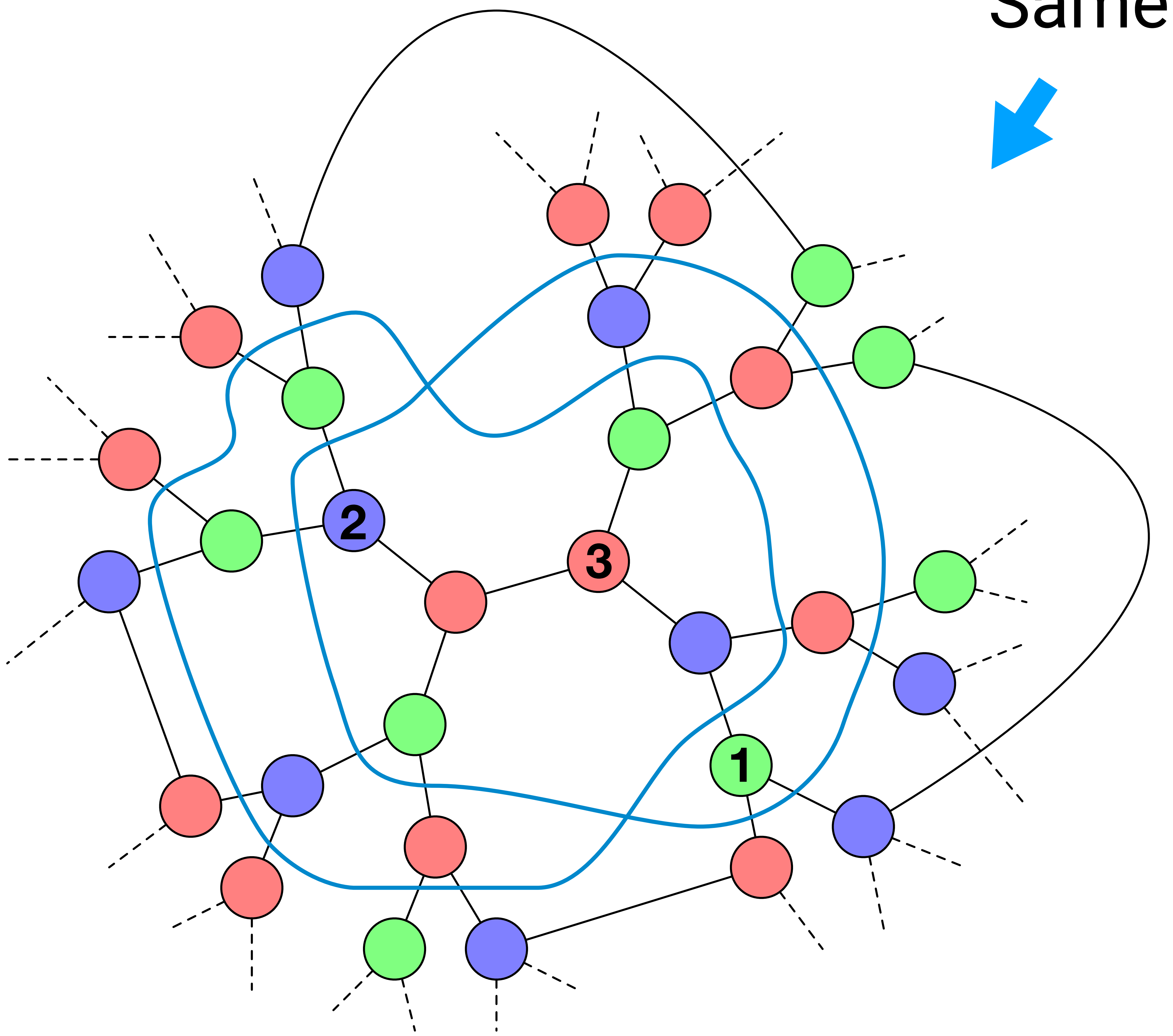


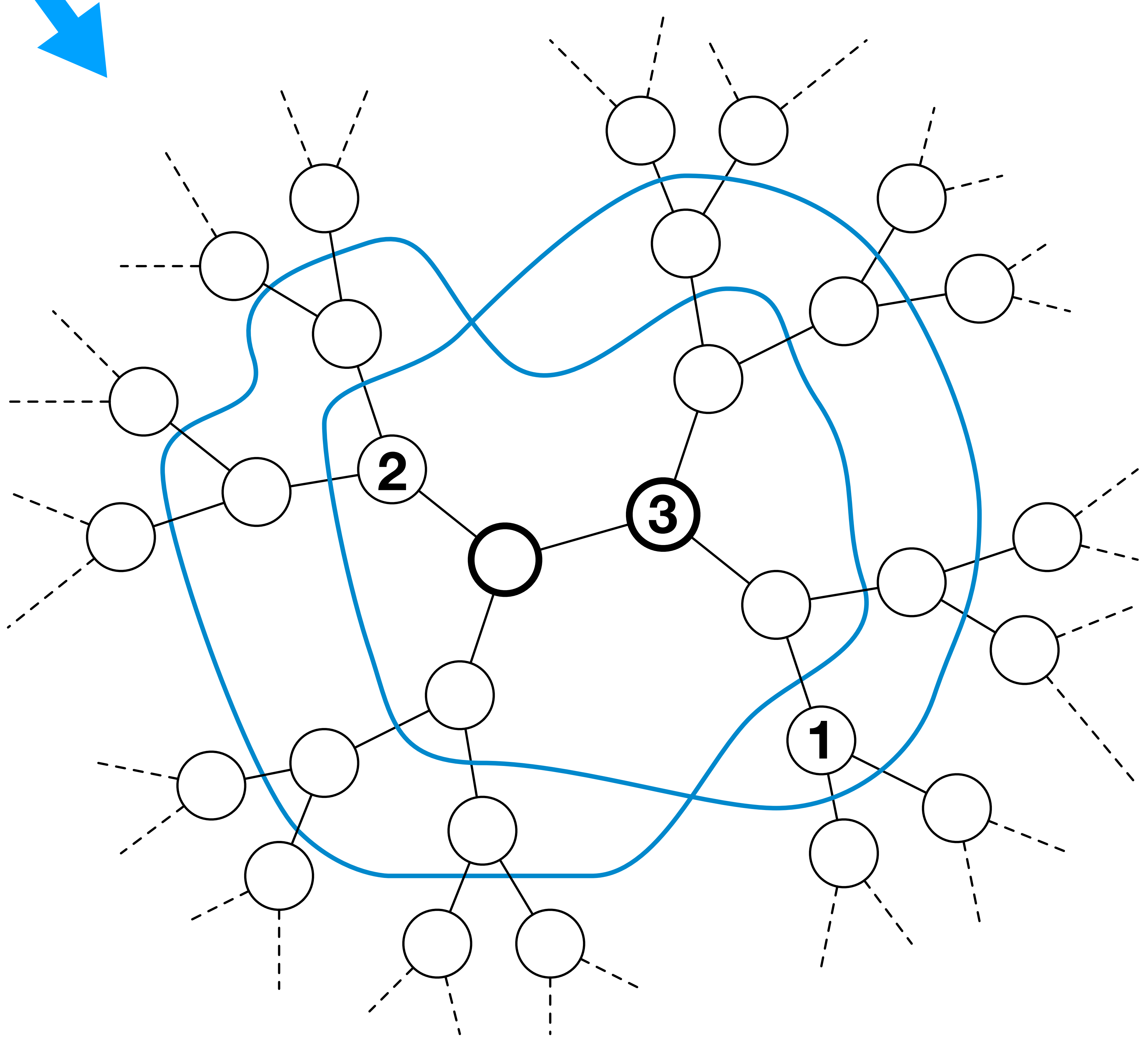
Coloring trees lower bound



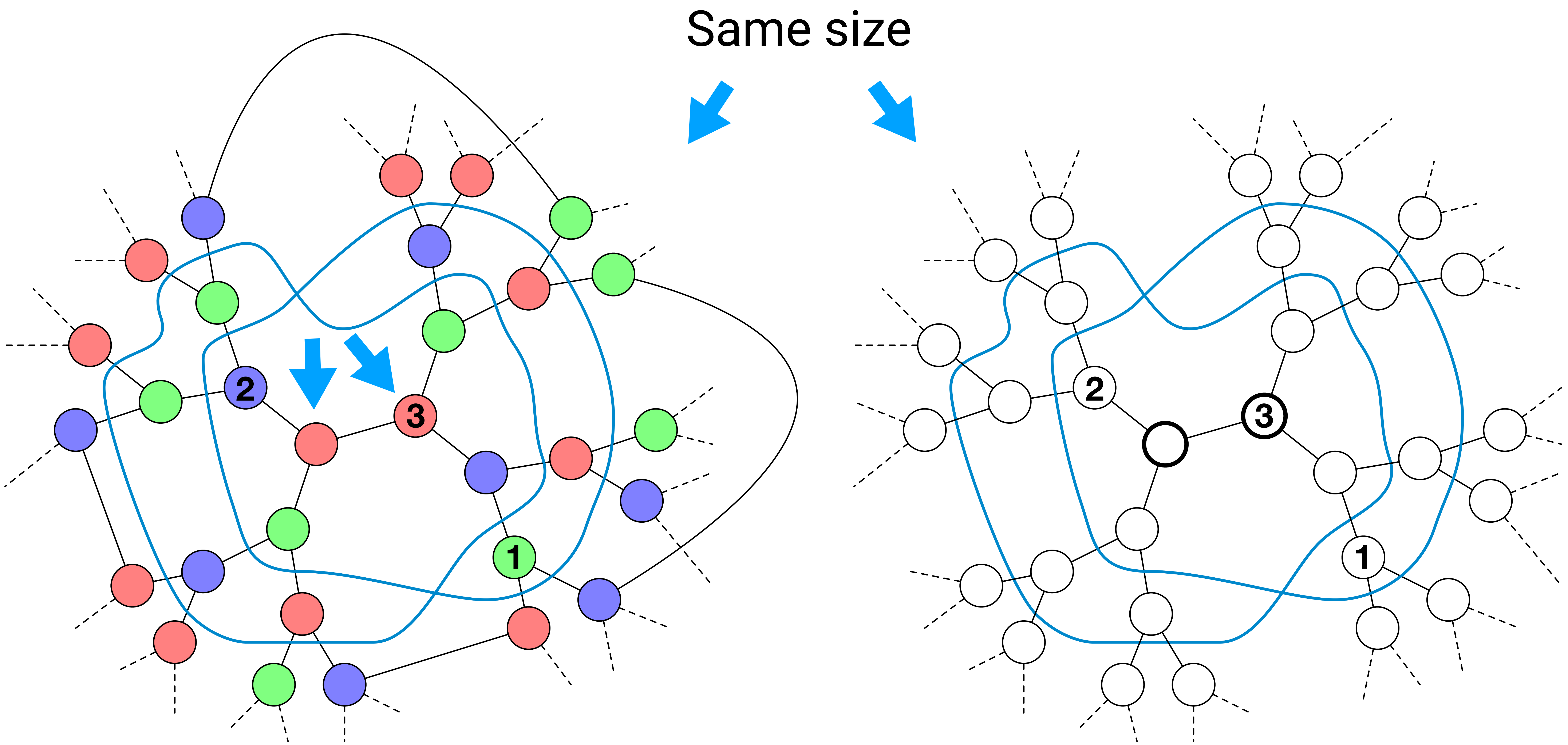
Coloring trees lower bound

Same size



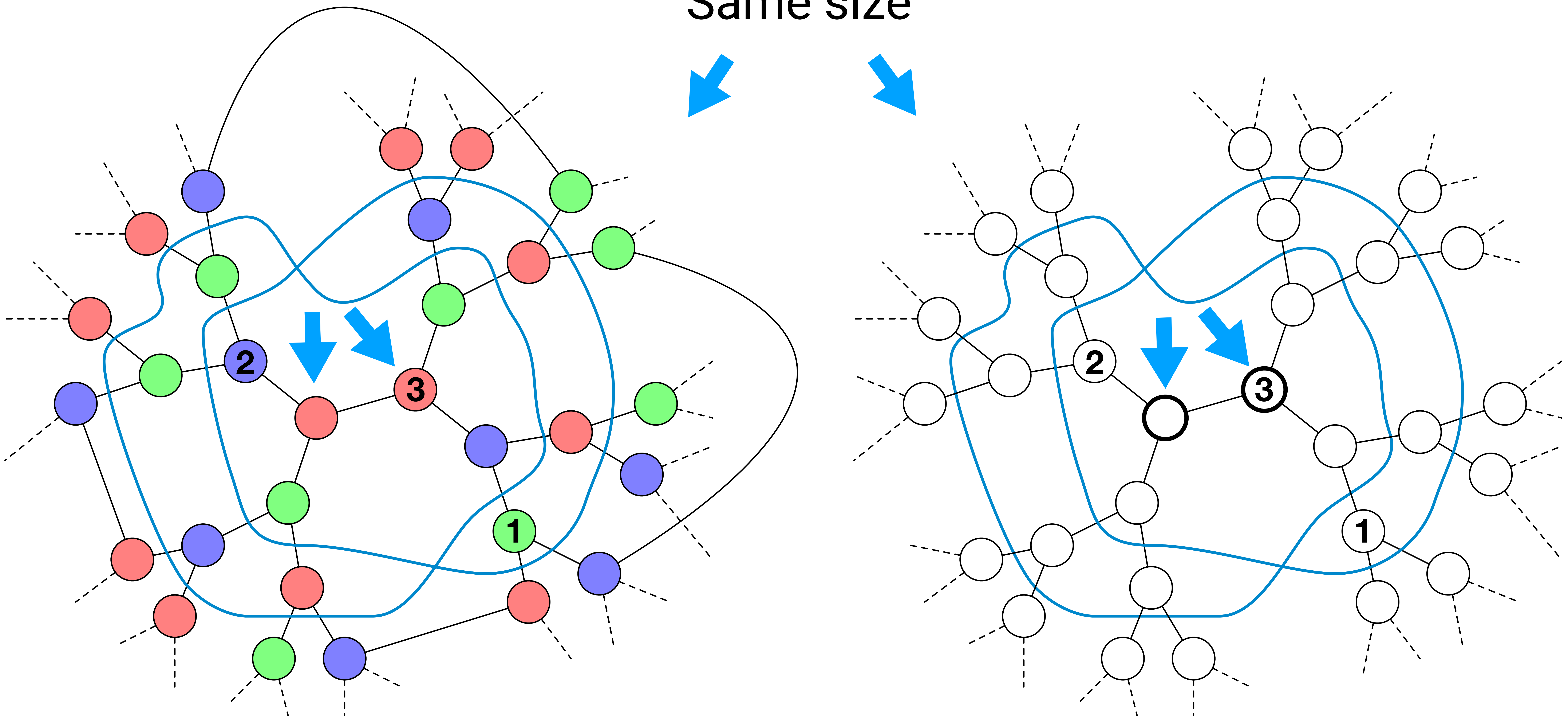


Coloring trees lower bound

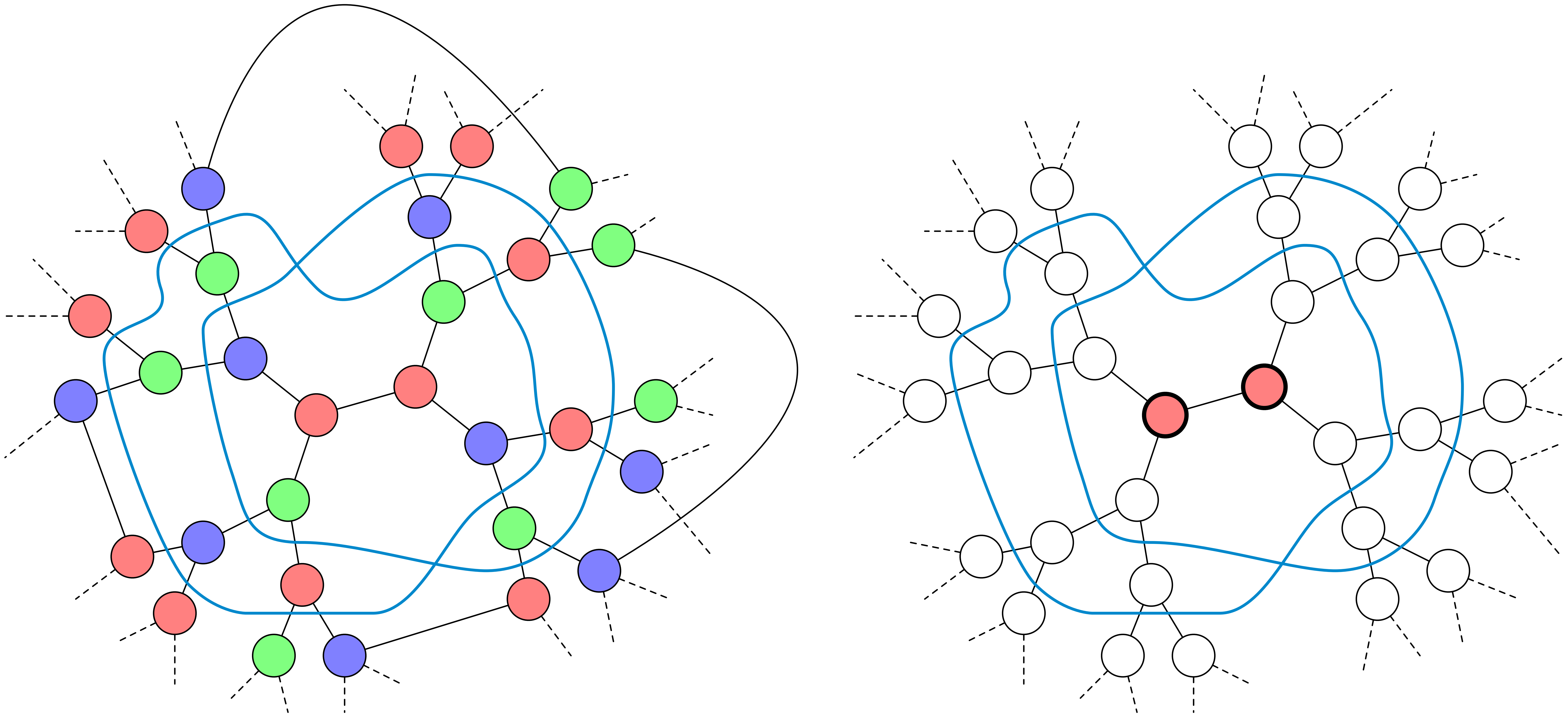


Coloring trees lower bound

Same size

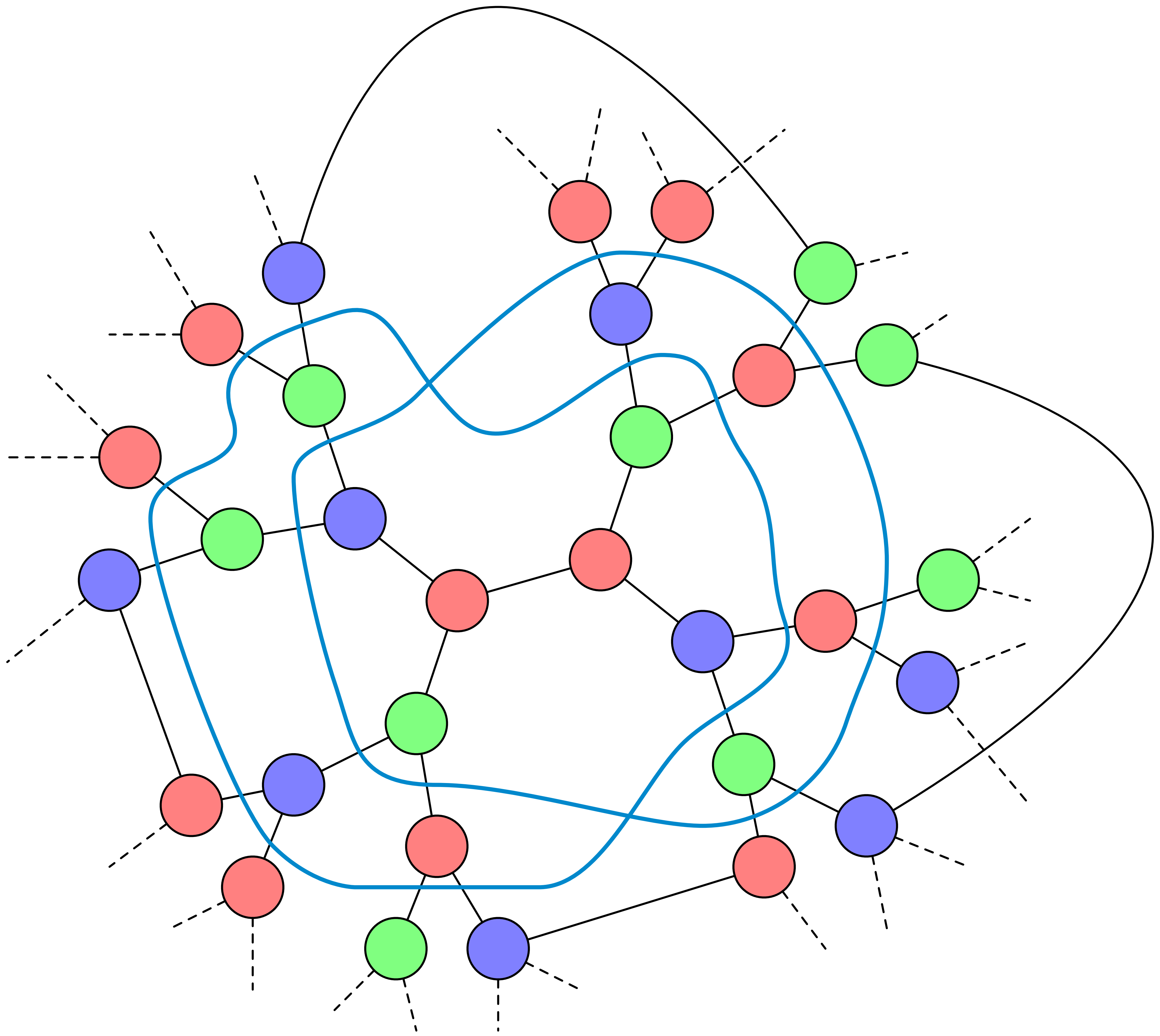


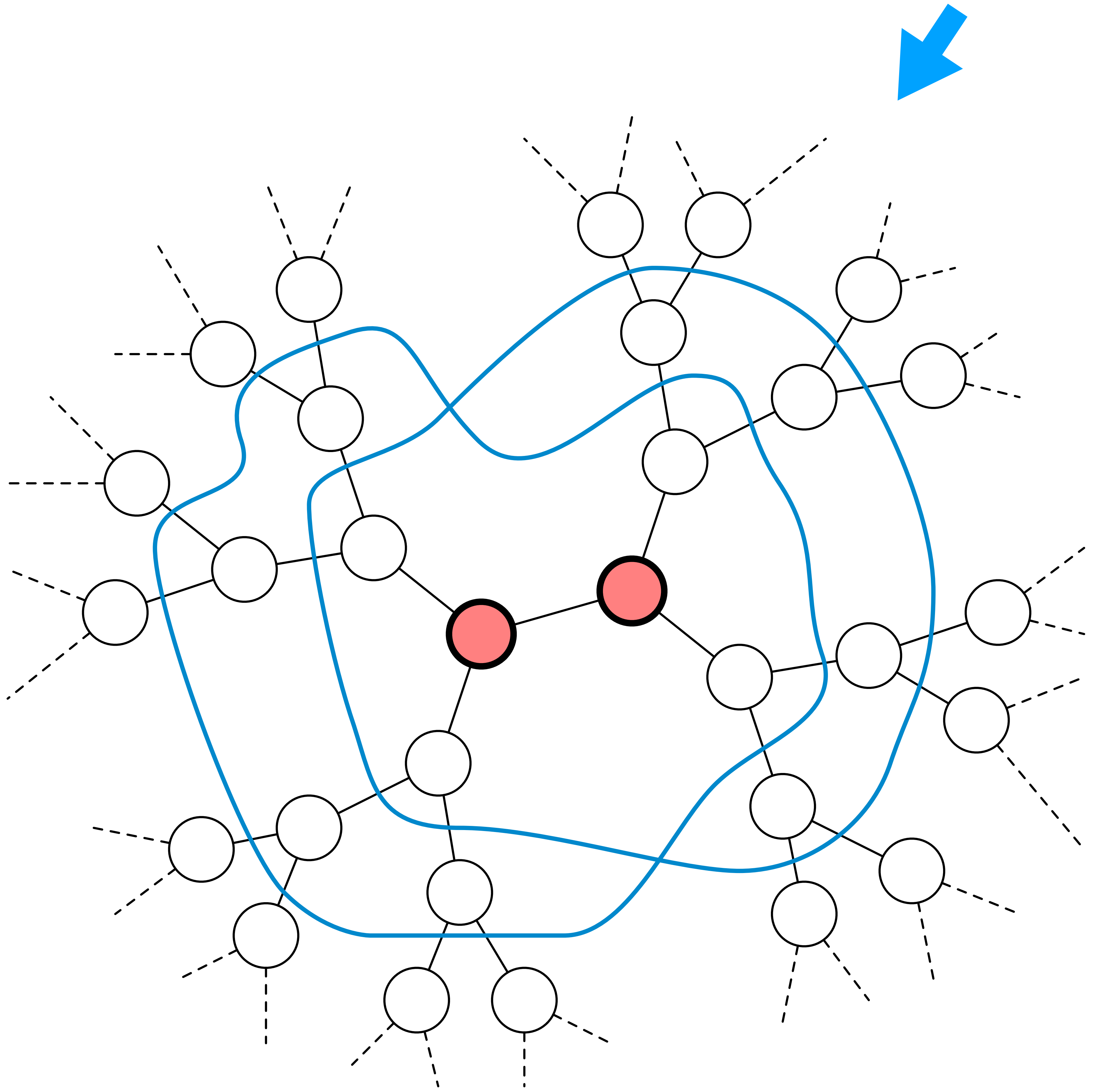
Coloring trees lower bound



Coloring trees lower bound

Fail on a real tree!





Coloring trees lower bound

Coloring trees lower bound

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

Coloring trees lower bound

- We want to prove that **$o(\Delta / \log \Delta)$** coloring trees of maximum degree **Δ** 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 Δ 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 Δ-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 Δ 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 Δ-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 Δ 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 Δ-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

- We want to prove that $o(\Delta / \log \Delta)$ coloring trees of maximum degree Δ 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 Δ-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

- We want to prove that $o(\Delta / \log \Delta)$ coloring trees of maximum degree Δ 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 Δ-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

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

Coloring trees lower bound

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

Coloring trees lower bound

- We saw how to prove:
  - Coloring trees of maximum degree **Δ** with **$o(\Delta / \log \Delta)$** colors requires **$\Omega(\log_\Delta n)$** rounds
- It is possible to prove:
  - Coloring trees of maximum degree **Δ** with **Δ** colors requires **$\Omega(\log_\Delta n)$** rounds
  - Different techniques are required to prove such result
  - Note that, if **$\Delta=O(1)$**, then **$(\Delta+1)$-coloring** can be solved in just **$O(\log^* n)$**

Coloring paths and cycles

Coloring paths and cycles

- 3-coloring paths or cycles can be done in **$O(\log^* n)$** rounds

Coloring paths and cycles

- 3-coloring paths or cycles can be done in **O(log* n)** rounds
  - **log* n** grows very slowly, but it is not a constant

Coloring paths and cycles

- 3-coloring paths or cycles can be done in **O(log* n)** rounds
  - **log* n** grows very slowly, but it is not a constant
  - Can we do better? Can we solve 3-coloring in **O(1)**?

Coloring paths and cycles

- 3-coloring paths or cycles can be done in **$O(\log^* n)$** rounds
  - **$\log^* n$** grows very slowly, but it is not a constant
  - Can we do better? Can we solve 3-coloring in **$O(1)$**?

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

Coloring paths and cycles

- 3-coloring paths or cycles can be done in **O(log* n)** rounds
  - **log* n** grows very slowly, but it is not a constant
  - Can we do better? Can we solve 3-coloring in **O(1)**?

- 3-coloring paths or cycles requires **Ω(log* n)** rounds

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

Coloring paths and cycles

- 3-coloring paths or cycles can be done in **O(log* n)** rounds
  - **log* n** grows very slowly, but it is not a constant
  - Can we do better? Can we solve 3-coloring in **O(1)**?

- 3-coloring paths or cycles requires **Ω(log* n)** rounds

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

Deterministic

Coloring paths and cycles

- 3-coloring paths or cycles can be done in **O(log* n)** rounds
  - **log* n** grows very slowly, but it is not a constant
  - Can we do better? Can we solve 3-coloring in **O(1)**?

- 3-coloring paths or cycles requires **Ω(log* n)** rounds

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

Deterministic (Linial '87) — Randomized (Naor '91)

Coloring paths and cycles

- 3-coloring paths or cycles can be done in **O(log* n)** rounds
  - **log* n** grows very slowly, but it is not a constant
  - Can we do better? Can we solve 3-coloring in **O(1)**?

- 3-coloring paths or cycles requires **Ω(log* n)** rounds

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

Deterministic ↑ [Linial '87] Randomized ↑ [Naor '91] With Ramsey theory ↑ [Naor, Stockmeyer '93]

Coloring paths and cycles

- 3-coloring paths or cycles can be done in **O(log* n)** rounds
  - **log* n** grows very slowly, but it is not a constant
  - Can we do better? Can we solve 3-coloring in **O(1)**?

- 3-coloring paths or cycles requires **Ω(log* n)** rounds

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

Deterministic — Randomized — With Ramsey theory — Easier proof

Coloring paths and cycles

- 3-coloring paths or cycles can be done in **O(log* n)** rounds
  - **log* n** grows very slowly, but it is not a constant
  - Can we do better? Can we solve 3-coloring in **O(1)**?

- 3-coloring paths or cycles requires **Ω(log* n)** rounds

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

Deterministic — Randomized — With Ramsey theory — Easier proof

Coloring paths and cycles

Coloring paths and cycles

- Simple observation:

Coloring paths and cycles

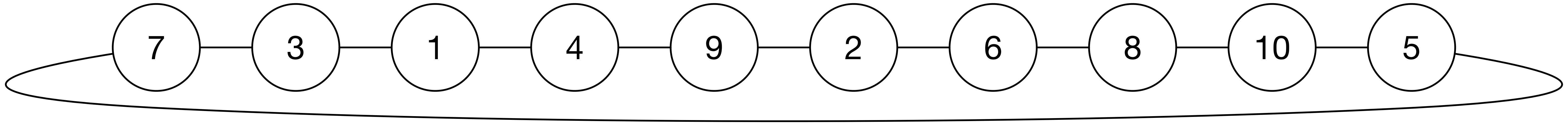
- Simple observation:
  - Assume nodes of a cycle have unique IDs in **{1, ..., n}**

Coloring paths and cycles

- Simple observation:
  - Assume nodes of a cycle have unique IDs in **{1, ..., n}**
  - We can color the nodes with **n colors** in **0 rounds**

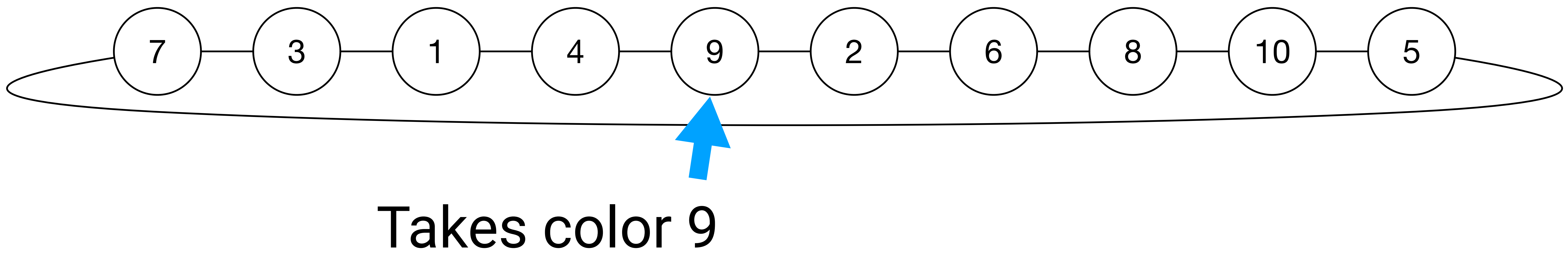
Coloring paths and cycles

- Simple observation:
  - Assume nodes of a cycle have unique IDs in **{1, ..., n}**
  - We can color the nodes with **n colors** in **0 rounds**



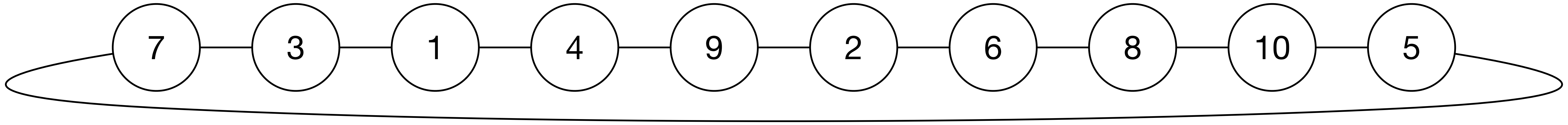
Coloring paths and cycles

- Simple observation:
  - Assume nodes of a cycle have unique IDs in **{1, ..., n}**
  - We can color the nodes with **n colors** in **0 rounds**



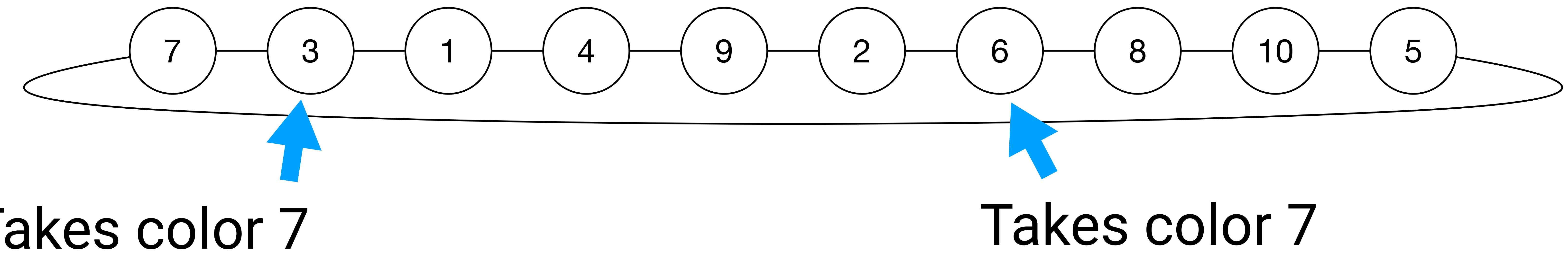
Coloring paths and cycles

- Simple observation:
  - Assume nodes of a cycle have unique IDs in **{1, ..., n}**
  - We can color the nodes with **n colors** in **0 rounds**
  - We **cannot** color the nodes with **n-1 colors** in **0 rounds**



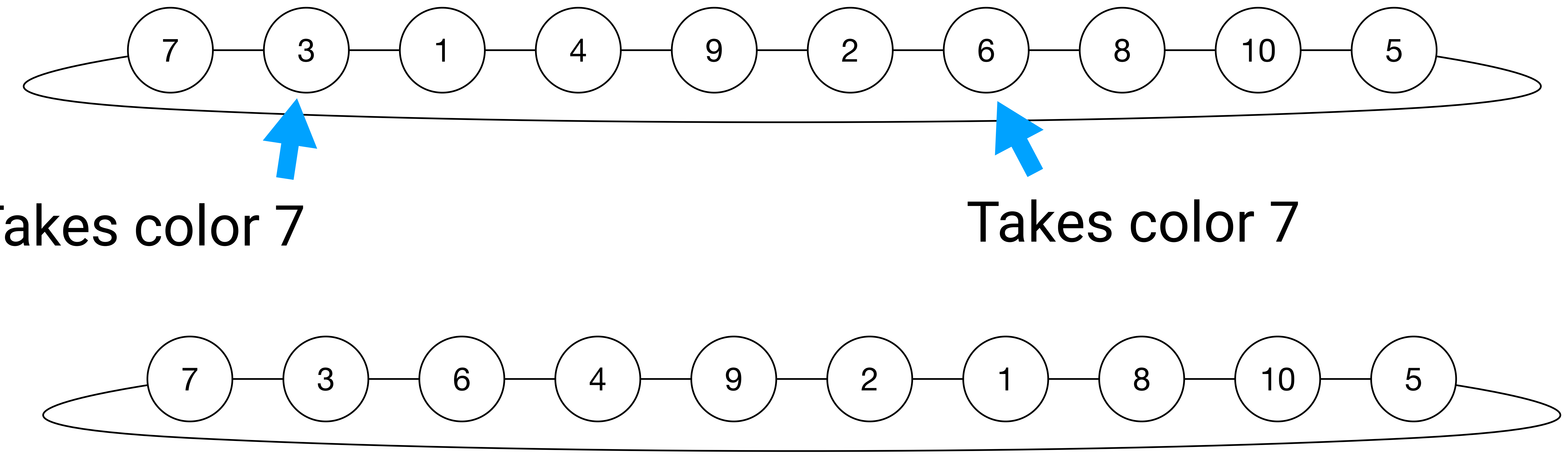
Coloring paths and cycles

- Simple observation:
  - Assume nodes of a cycle have unique IDs in **{1, ..., n}**
  - We can color the nodes with **n colors** in **0 rounds**
  - We **cannot** color the nodes with **n-1 colors** in **0 rounds**



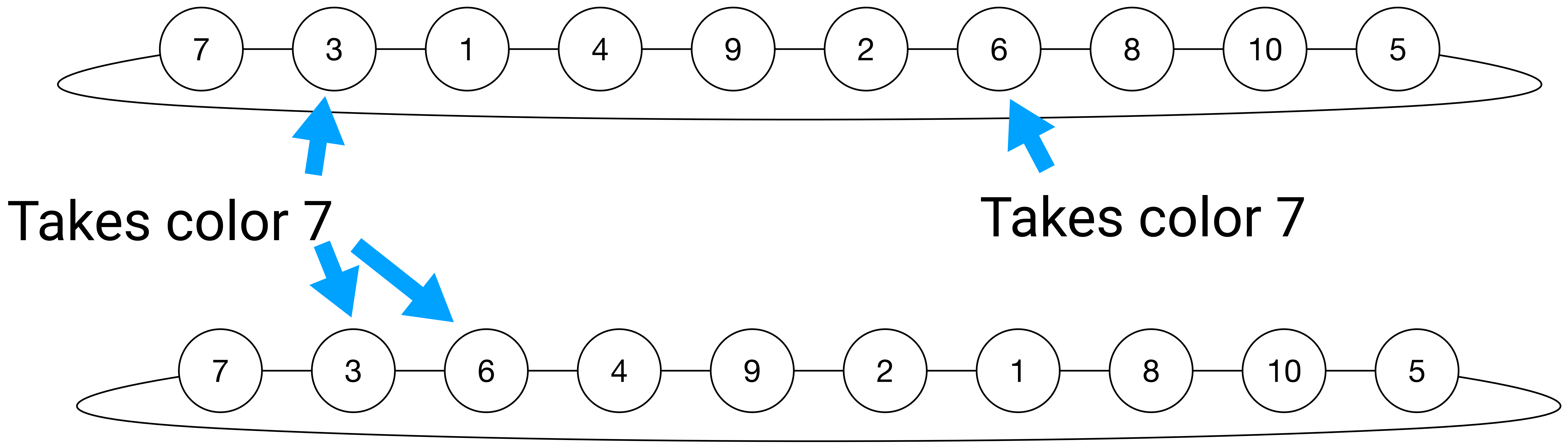
Coloring paths and cycles

- Simple observation:
  - Assume nodes of a cycle have unique IDs in **{1, ..., n}**
  - We can color the nodes with **n colors** in **0 rounds**
  - We **cannot** color the nodes with **n-1 colors** in **0 rounds**



Coloring paths and cycles

- Simple observation:
  - Assume nodes of a cycle have unique IDs in **{1, ..., n}**
  - We can color the nodes with **n colors** in **0 rounds**
  - We **cannot** color the nodes with **n-1 colors** in **0 rounds**



Coloring paths and cycles

- Simple observation:
  - Assume nodes of a cycle have unique IDs in **{1, ..., n}**
  - We can color the nodes with **n colors** in **0 rounds**
  - We **cannot** color the nodes with **n-1 colors** in **0 rounds**

$\Rightarrow$

We **cannot** color the nodes
with **o(n)** colors in **0 rounds**

Coloring paths and cycles

Coloring paths and cycles

1. We **cannot** color the nodes with **$o(n)$** colors in **0** rounds

Coloring paths and cycles

1. We **cannot** color the nodes with **$o(n)$** colors in **0** rounds
2. If **c**-coloring can be solved in **T** rounds, then **2^c**-coloring can be solved in **$T-1$** rounds

Coloring paths and cycles

1. We **cannot** color the nodes with **o(n)** colors in **0** rounds
2. If **c**-coloring can be solved in **T** rounds, then 2^c-coloring can be solved in **T-1** rounds

- Suppose we can solve **o(log n)**-coloring in **1** round. Then, **by (2)** we can solve **o(n)**-coloring in **0** rounds, that is a **contradiction with (1)**

Coloring paths and cycles

1. We **cannot** color the nodes with **o(n)** colors in **0** rounds
2. If **c**-coloring can be solved in **T** rounds, then 2^c-coloring can be solved in **T-1** rounds

- Suppose we can solve **o(log n)**-coloring in **1** round. Then, **by (2)** we can solve **o(n)**-coloring in **0** rounds, that is a **contradiction with (1)**
- We **cannot** color the nodes with **o(log n)** colors in **1** round

Coloring paths and cycles

1. We **cannot** color the nodes with **o(n)** colors in **0** rounds
2. If **c**-coloring can be solved in **T** rounds, then 2^c-coloring can be solved in **T-1** rounds

- Suppose we can solve **o(log n)**-coloring in **1** round. Then, **by (2)** we can solve **o(n)**-coloring in **0** rounds, that is a **contradiction with (1)**
- We **cannot** color the nodes with **o(log n)** colors in **1** round
- We **cannot** color the nodes with **o(log log n)** colors in **2** rounds

Coloring paths and cycles

1. We **cannot** color the nodes with **o(n)** colors in **0** rounds
2. If **c**-coloring can be solved in **T** rounds, then 2^c-coloring can be solved in **T-1** rounds

- Suppose we can solve **o(log n)**-coloring in **1** round. Then, **by (2)** we can solve **o(n)**-coloring in **0** rounds, that is a **contradiction with (1)**
- We **cannot** color the nodes with **o(log n)** colors in **1** round
- We **cannot** color the nodes with **o(log log n)** colors in **2** rounds
- We **cannot** color the nodes with **o(log log log n)** colors in **3** rounds

Coloring paths and cycles

1. We **cannot** color the nodes with **o(n)** colors in **0** rounds
2. If **c**-coloring can be solved in **T** rounds, then 2^c-coloring can be solved in **T-1** rounds

- Suppose we can solve **o(log n)**-coloring in **1** round. Then, **by (2)** we can solve **o(n)**-coloring in **0** rounds, that is a **contradiction with (1)**
- We **cannot** color the nodes with **o(log n)** colors in **1** round
- We **cannot** color the nodes with **o(log log n)** colors in **2** rounds
- We **cannot** color the nodes with **o(log log log n)** colors in **3** rounds
- ...

Coloring paths and cycles

1. We **cannot** color the nodes with **o(n)** colors in **0** rounds
2. If **c**-coloring can be solved in **T** rounds, then 2^c-coloring can be solved in **T-1** rounds

- Suppose we can solve **o(log n)**-coloring in **1** round. Then, **by (2)** we can solve **o(n)**-coloring in **0** rounds, that is a **contradiction with (1)**
- We **cannot** color the nodes with **o(log n)** colors in **1** round
- We **cannot** color the nodes with **o(log log n)** colors in **2** rounds
- We **cannot** color the nodes with **o(log log log n)** colors in **3** rounds
- ...
- We **cannot** color the nodes with **O(1)**-colors in **o(log*n)** rounds

Coloring paths and cycles

1. We **cannot** color the nodes with $o(n)$ colors in **0** rounds
2. If **c**-coloring can be solved in **T** rounds, then 2^c-coloring can be solved in **T-1** rounds

- Suppose we can solve $o(\log n)$-coloring in **1** round. Then, **by (2)** we can solve $o(n)$-coloring in **0** rounds, that is a **contradiction with (1)**
- We **cannot** color the nodes with $o(\log n)$ colors in **1** round
- We **cannot** color the nodes with $o(\log \log n)$ colors in **2** rounds
- We **cannot** color the nodes with $o(\log \log \log n)$ colors in **3** rounds
- ...
- We **cannot** color the nodes with $O(1)$-colors in $o(\log^* n)$ rounds
- We **cannot** color the nodes with **3**-colors in $o(\log^* n)$ rounds

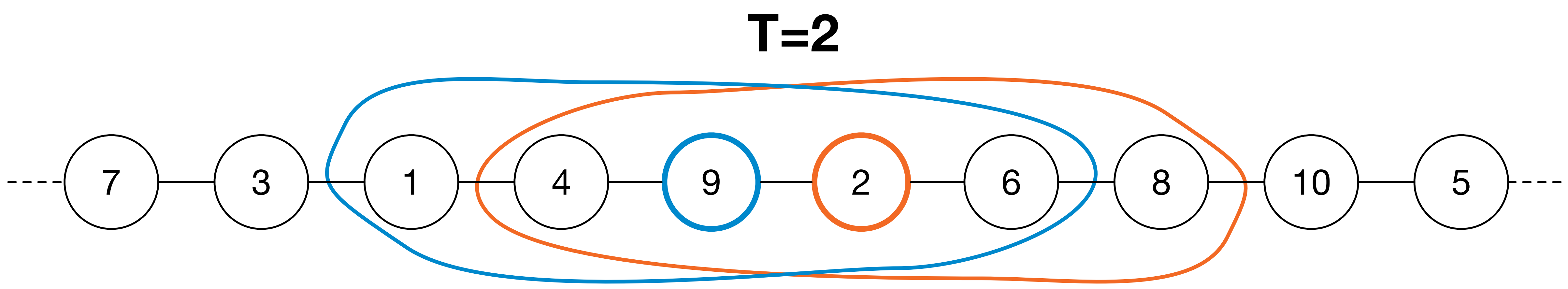
Coloring algorithms

Coloring algorithms

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

Coloring algorithms

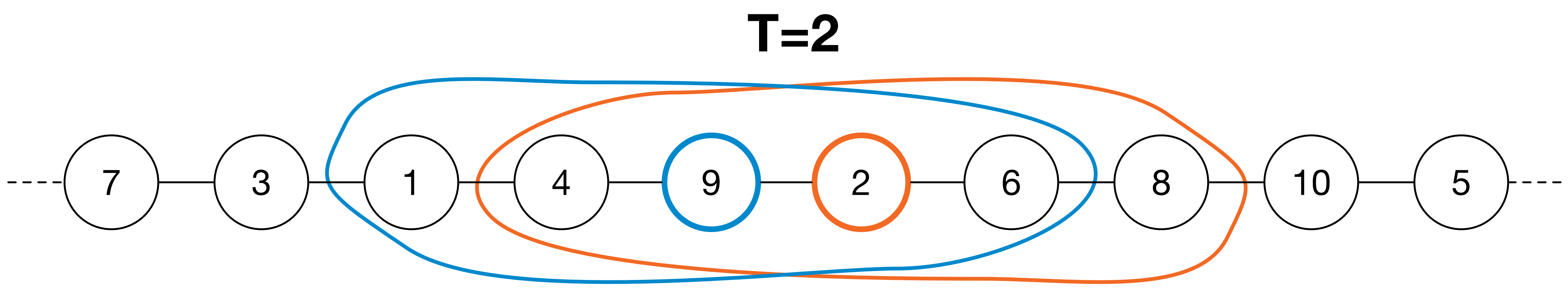
- We can see an algorithm **A** as a function satisfying that:



Coloring algorithms

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

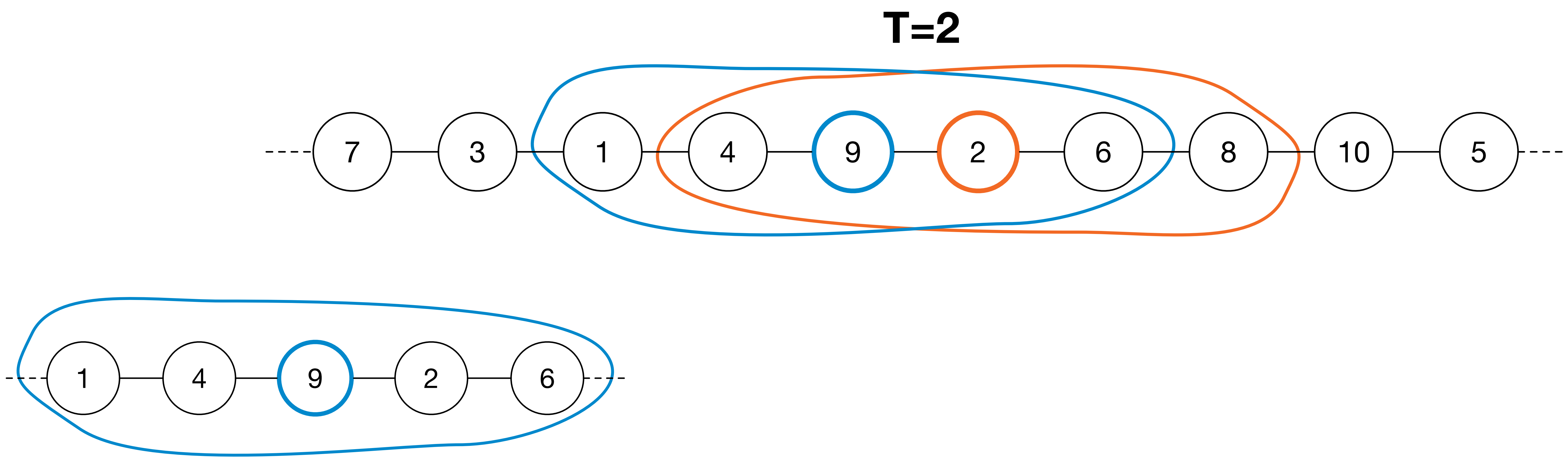
$$A_n(x_1, \ldots, x_{2T+1}) \in \{1, 2, 3\}$$



Coloring algorithms

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

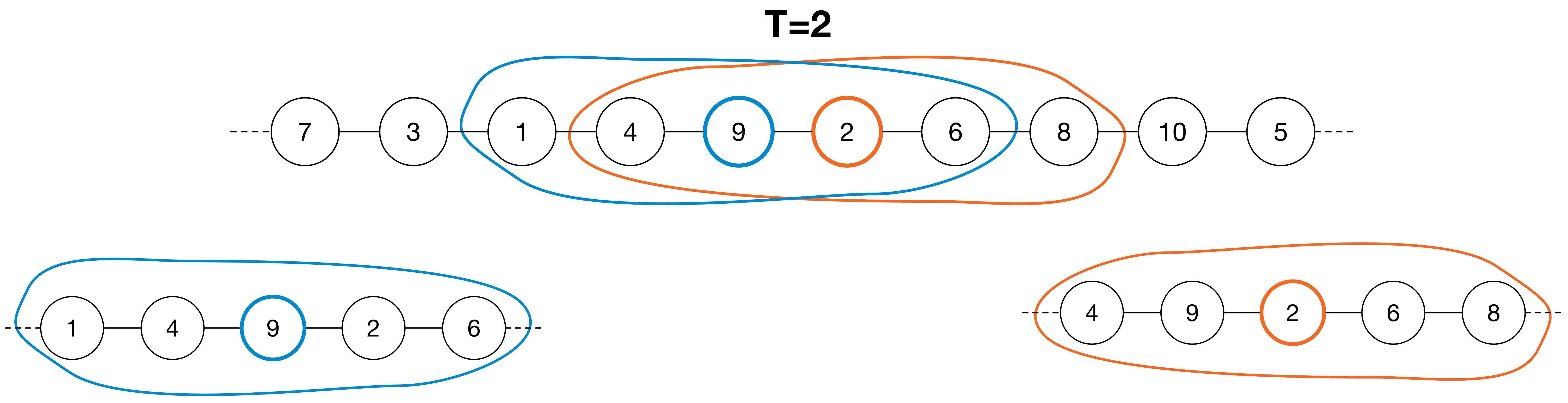
$$A_n(x_1, ..., x_{2T+1}) \in \{1, 2, 3\}$$



Coloring algorithms

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

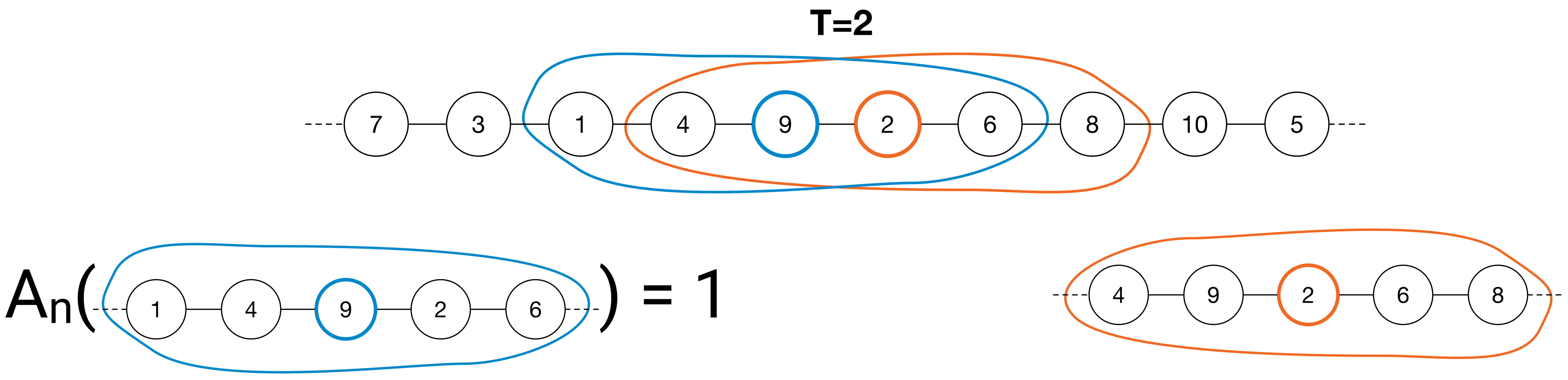
$$A_n(x_1, \ldots, x_{2T+1}) \in \{1, 2, 3\}$$



Coloring algorithms

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

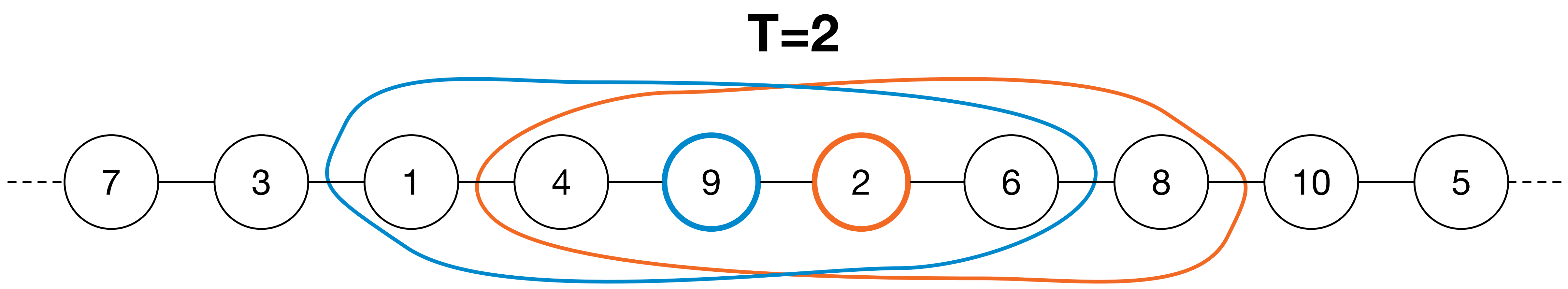
$$A_n(x_1, \ldots, x_{2T+1}) \in \{1, 2, 3\}$$



Coloring algorithms

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

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



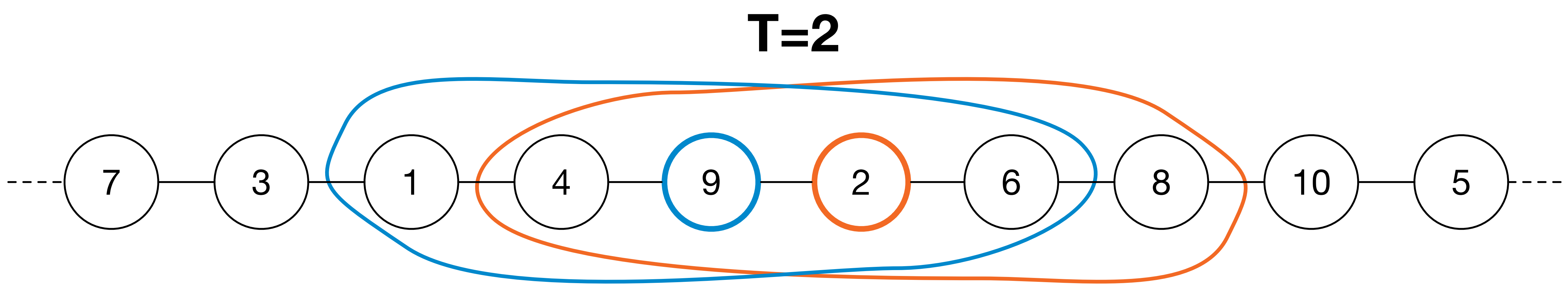
Coloring algorithms

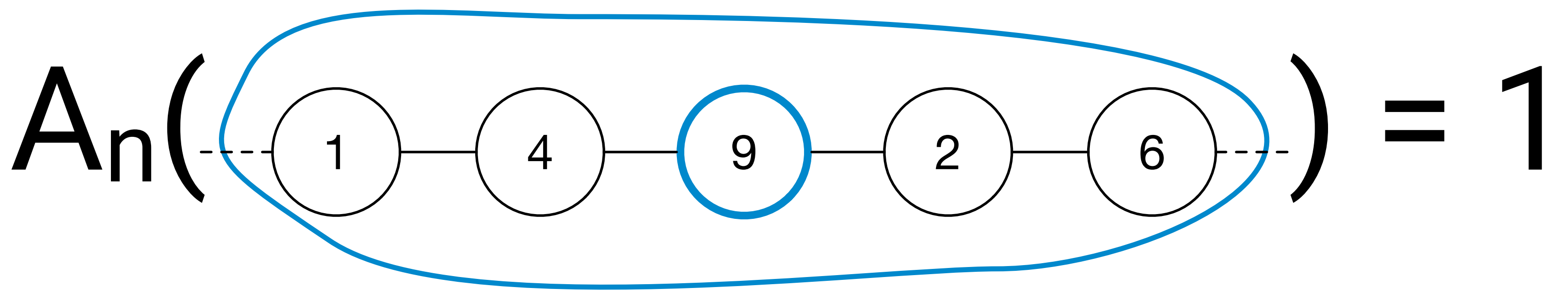
- We can see an algorithm **A** as a function satisfying that:

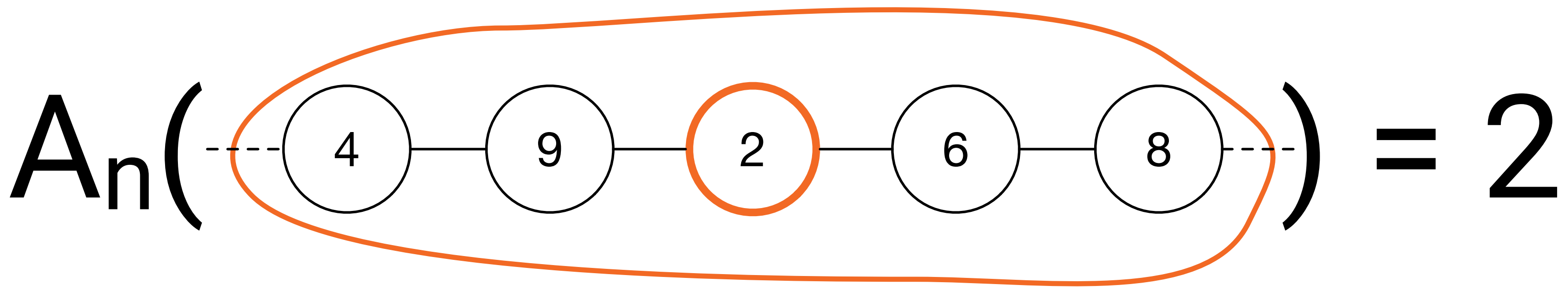
$$A_n(x_1, \ldots, x_{2T+1}) \in \{1, 2, 3\}$$

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

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:

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 $\{1, \ldots, n\}$

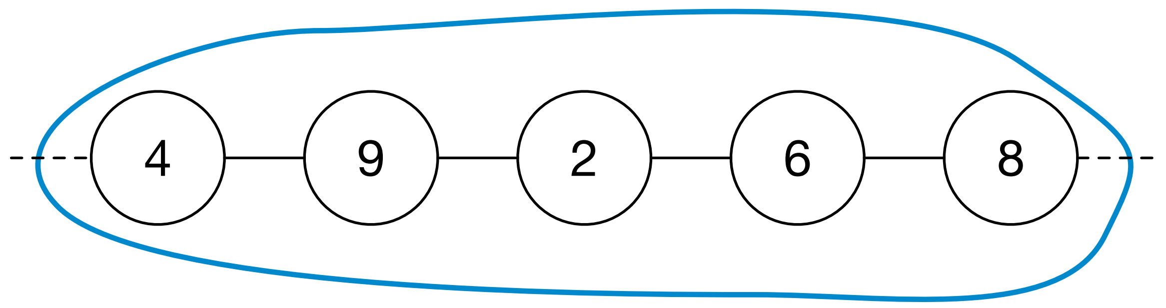
Coloring functions

- 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 $\{1, \ldots, n\}$



$A_n(4, 9, 2, 6, 8) \in \{1, 2, 3\}$

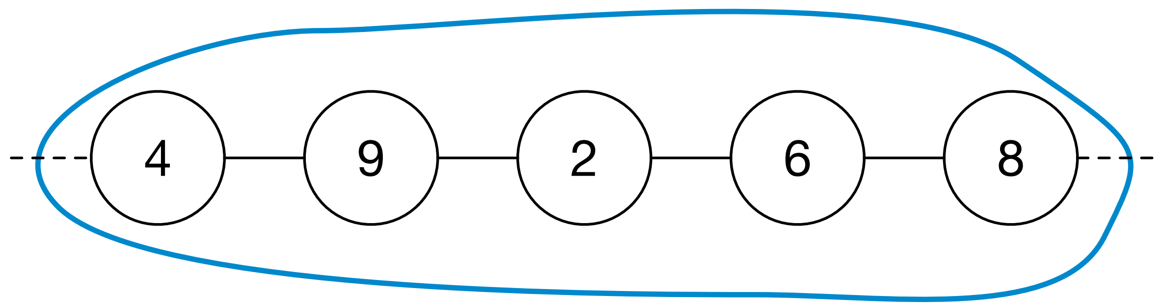
Coloring functions

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

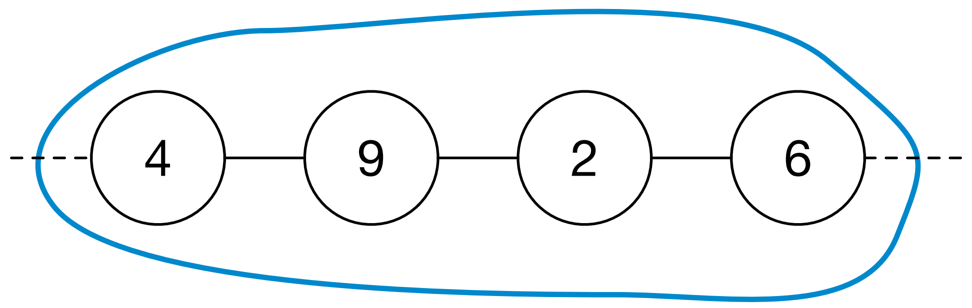
$$A_n(x_1, \ldots, x_k) \in \{1, 2, \ldots, c\}$$

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

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



$A_n(4, 9, 2, 6, 8) \in \{1, 2, 3\}$



$A_n(4, 9, 2, 6) \in \{1, \ldots, 10\}$

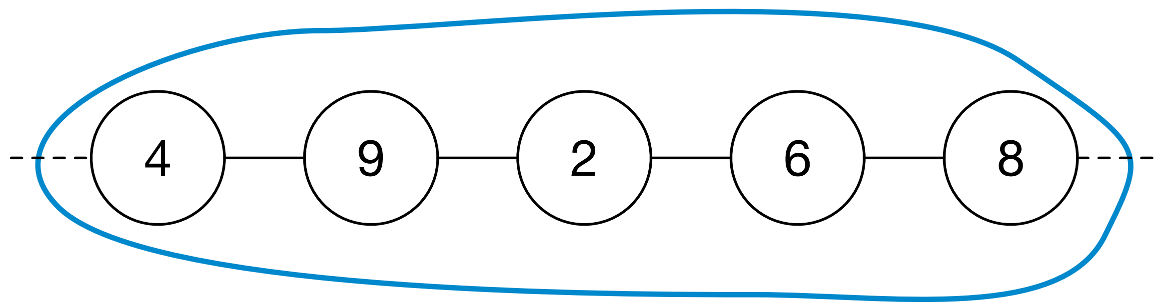
Coloring functions

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

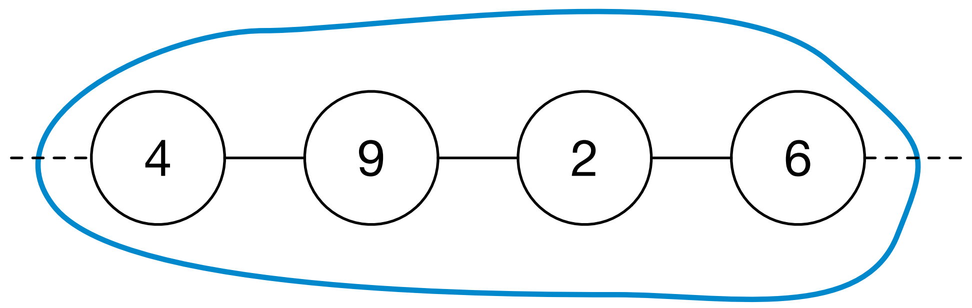
$$A_n(x_1, \ldots, x_k) \in \{1, 2, \ldots, c\}$$

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

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



$A_n(4, 9, 2, 6, 8) \in \{1, 2, 3\}$



$A_n(4, 9, 2, 6) \in \{1, \ldots, 10\}$

4-ary 10-coloring function

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 $\{1, \ldots, n\}$

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

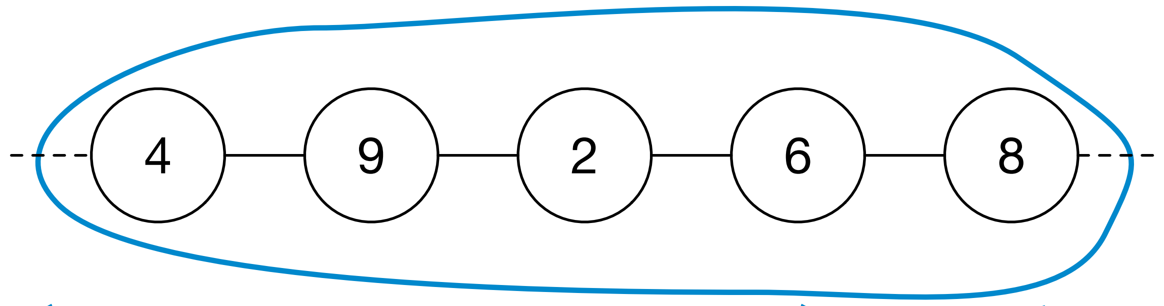
- A is a k-ary c-coloring function if:

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

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

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

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



$A_n(4, 9, 2, 6, 8) \in \{1, 2, 3\}$

Coloring functions

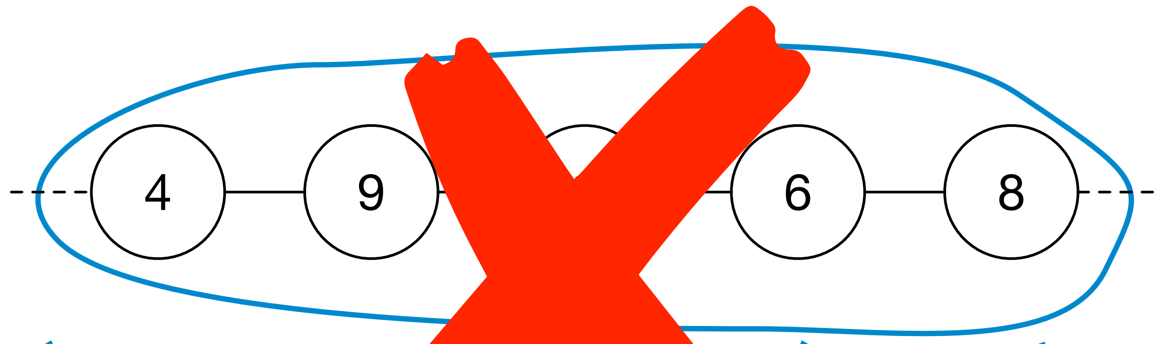
- A is a k-ary c-coloring function if:

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

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

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

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



$A_n(4, 9, 2, 6, 8) \in \{1, 2, 3\}$

$A_n(4, 9, 2, 6, 8)$ is undefined

Coloring functions

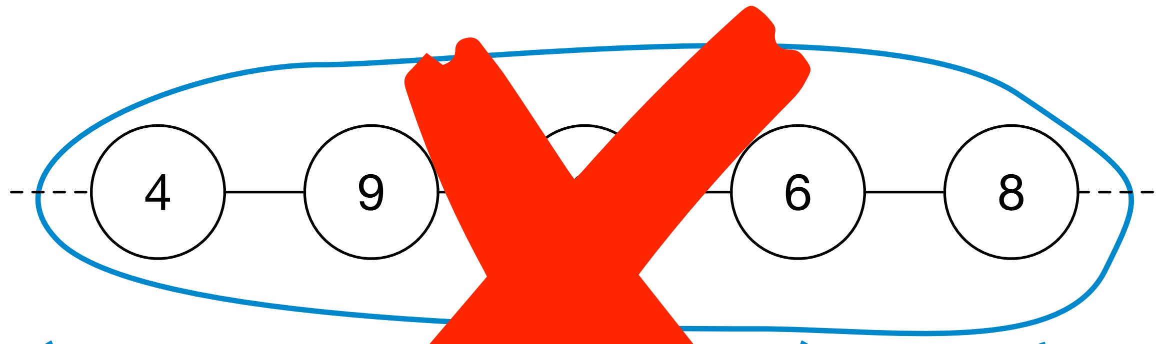
- **A** is a **k**-ary **c**-coloring function if:

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

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

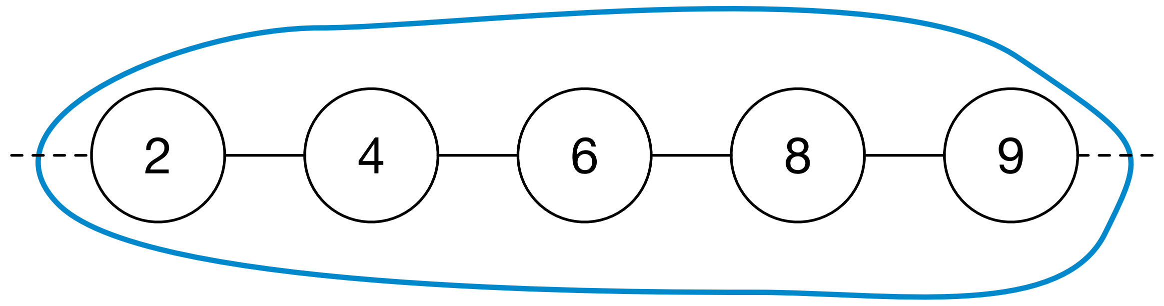
assuming $x_1, \ldots, x_{k+1}$ are all distinct numbers from $\{1, \ldots, n\}$

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



$A_n(4, 9, 2, 6, 8) \in \{1, 2, 3\}$

$A_n(4, 9, 2, 6, 8)$ is undefined



$A_n(2, 4, 6, 8, 9) \in \{1, 2, 3\}$

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 for **3**-coloring defines a **2T+1**-ary **3**-coloring function

Coloring functions

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

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

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

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

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

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

We cannot use a function to construct an algorithm.

$f(4, 9, 2, 6, 8)$ is undefined!

Coloring functions

Coloring functions

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

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

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

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

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 this statement by induction

Coloring functions (base case)

Coloring functions (base case)

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

$$c \geq n$$

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 (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

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

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

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

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

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

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

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

If $1 \leq x_1 \leq x_2 \leq n$,

then $A_n(x_1) \neq A_n(x_2)$

Coloring functions (base case)

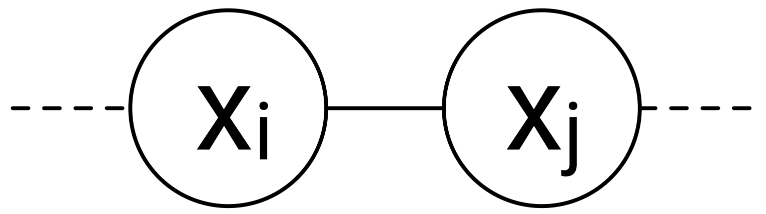
- For any **1**-ary **c**-coloring function:

$$c \geq n$$

- 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

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



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

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

If $1 \leq x_1 \leq x_2 \leq n$,

then $A_n(x_1) \neq A_n(x_2)$

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

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

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

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

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

  Notice that there are 2^c possible outputs

  $B_{10}(2,4,5,7)$

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

  Notice that there are 2^c possible outputs

$B_{10}(2,4,5,7)$
$= \{$

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

  Notice that there are 2^c possible outputs

$B_{10}(2,4,5,7)$
$= \{$
$A_{10}(2,4,5,7,8),$

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

  Notice that there are 2^c possible outputs

$B_{10}(2,4,5,7)$
= {
$A_{10}(2,4,5,7,8)$,
$A_{10}(2,4,5,7,9)$,

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

  Notice that there are 2^c possible outputs

$B_{10}(2,4,5,7)$
$= \{$
$A_{10}(2,4,5,7,8)$,
$A_{10}(2,4,5,7,9)$,
$A_{10}(2,4,5,7,10)$

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

  Notice that there are 2^c possible outputs

$B_{10}(2,4,5,7)$
$= \{$
$A_{10}(2,4,5,7,8)$,
$A_{10}(2,4,5,7,9)$,
$A_{10}(2,4,5,7,10)$
$\}$

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

  Notice that there are 2^c possible outputs

  Let us now prove that it is a coloring function

$B_{10}(2,4,5,7)$
$= \{$
$A_{10}(2,4,5,7,8)$,
$A_{10}(2,4,5,7,9)$,
$A_{10}(2,4,5,7,10)$
$\}$

Coloring functions (inductive case)

Coloring functions (inductive case)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, y) \mid n \geq y > x_{k-1} \}$
- We need to prove that:

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

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)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, y) \mid n \geq y > x_{k-1} \}$
- We need to prove that:

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

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)

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

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

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

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

$B_n(x_1, \ldots, x_{k-1}) =$
$\{ A_n(x_1, \ldots, x_{k-1}, y) \mid$ y is larger than $x_{k-1} \}$

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

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

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

$\{ A_n(x_1, \ldots, x_{k-1}, y) \mid$ y is larger than $x_{k-1} \}$

since $x_k > x_{k-1}$,

Coloring functions (inductive case)

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

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

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

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

- Let $x = A_n(x_1, ..., x_k)$

$B_n(x_1, ..., x_{k-1}) =$

$\{ A_n(x_1, ..., x_{k-1}, y) \mid$ y is larger than $x_{k-1} \}$

since $x_k > x_{k-1}$,

then $x = A_n(x_1, ..., x_{k-1}, x_k) \in B_n(x_1, ..., x_{k-1})$

Coloring functions (inductive case)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, y) \mid n \geq y > x_{k-1} \}$
- 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 $x \in B_n(x_1, \ldots, x_{k-1})$

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

$\{ A_n(x_1, \ldots, x_{k-1}, y) \mid y \text{ is larger than } x_{k-1} \}$

since $x_k > x_{k-1}$,

then $x = A_n(x_1, \ldots, x_{k-1}, x_k) \in B_n(x_1, \ldots, x_{k-1})$

Coloring functions (inductive case)

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

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

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

  satisfying $1 \leq x_1 \leq x_2 \leq ... \leq x_{k-1} \leq x_k \leq n$
- Let $x = A_n(x_1, ..., x_k)$
- By definition of B, we have that $x \in B_n(x_1, ..., x_{k-1})$

$B_n(x_1, ..., x_{k-1}) =$

$\{ A_n(x_1, ..., x_{k-1}, y) \mid y \text{ is larger than } x_{k-1} \}$

since $x_k > x_{k-1}$,

then $x = A_n(x_1, ..., x_{k-1}, x_k) \in B_n(x_1, ..., x_{k-1})$

Coloring functions (inductive case)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, y) \mid n \geq y > x_{k-1} \}$
- 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 $x \in B_n(x_2, \ldots, x_k)$

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

$\{ A_n(x_1, \ldots, x_{k-1}, y) \mid$ y is larger than $x_{k-1} \}$

since $x_k > x_{k-1}$,

then $x = A_n(x_1, \ldots, x_{k-1}, x_k) \in B_n(x_1, \ldots, x_{k-1})$

Coloring functions (inductive case)

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

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

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

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

$B_n(x_1, ..., x_{k-1}) =$

$\{ A_n(x_1, ..., x_{k-1}, y) \mid$ y is larger than $x_{k-1} \}$

since $x_k > x_{k-1}$,

then $x = A_n(x_1, ..., x_{k-1}, x_k) \in B_n(x_1, ..., x_{k-1})$

$B_n(x_2, ..., x_k)$ contains x

Coloring functions (inductive case)

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

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

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

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

$B_n(x_1, \dots, x_{k-1}) =$

$\{ A_n(x_1, \dots, x_{k-1}, y) \mid y \text{ is larger than } x_{k-1} \}$

since $x_k > x_{k-1}$,

then $x = A_n(x_1, \dots, x_{k-1}, x_k) \in B_n(x_1, \dots, x_{k-1})$

$B_n(x_2, \dots, x_k)$ contains x

$\Rightarrow$

Coloring functions (inductive case)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, y) \mid n \geq y > x_{k-1} \}$
- 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 $x \in B_n(x_2, \ldots, x_k)$

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

$\{ A_n(x_1, \ldots, x_{k-1}, y) \mid$ y is larger than $x_{k-1} \}$

since $x_k > x_{k-1}$,

then $x = A_n(x_1, \ldots, x_{k-1}, x_k) \in B_n(x_1, \ldots, x_{k-1})$

$B_n(x_2, \ldots, x_k)$ contains x

$\Rightarrow$

$\exists\, y > x_k$ such that $A_n(x_2, \ldots, x_k, y) = x$

Coloring functions (inductive case)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, y) \mid n \geq y > x_{k-1} \}$
- 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 $y > x_k$ such that $A_n(x_2, \ldots, x_k, y) = x$

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

$\{ A_n(x_1, \ldots, x_{k-1}, y) \mid y \text{ is larger than } x_{k-1} \}$

since $x_k > x_{k-1}$,

then $x = A_n(x_1, \ldots, x_{k-1}, x_k) \in B_n(x_1, \ldots, x_{k-1})$

$B_n(x_2, \ldots, x_k)$ contains x

$\Rightarrow$

$\exists\, y > x_k$ such that $A_n(x_2, \ldots, x_k, y) = x$

Coloring functions (inductive case)

- $B_n(x_1, \ldots, x_{k-1}) = \{ A_n(x_1, \ldots, x_{k-1}, y) \mid n \geq y > x_{k-1} \}$
- 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 $y > x_k$ such that $A_n(x_2, \ldots, x_k, y) = x$
- $A_n(x_1, \ldots, x_k) = A_n(x_2, \ldots, x_k, y)$, such that $y > x_k$, contradiction!

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

$\{ A_n(x_1, \ldots, x_{k-1}, y) \mid y \text{ is larger than } x_{k-1} \}$

since $x_k > x_{k-1}$,

then $x = A_n(x_1, \ldots, x_{k-1}, x_k) \in B_n(x_1, \ldots, x_{k-1})$

$B_n(x_2, \ldots, x_k)$ contains x

$\Rightarrow$

$\exists\, y > x_k$ such that $A_n(x_2, \ldots, x_k, y) = x$

Coloring functions (putting things together)

Coloring functions (putting things together)

- Given:

Coloring functions (putting things together)

- Given:
  - a **T**-rounds coloring algorithm that solves **3**-coloring

Coloring functions (putting things together)

- Given:
  - a T-rounds coloring algorithm that solves 3-coloring
- We construct:

Coloring functions (putting things together)

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

Coloring functions (putting things together)

- Given:
  - a **T**-rounds coloring algorithm that solves **3**-coloring
- 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 that solves **3**-coloring
- 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 that solves **3**-coloring
- 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 that solves 3-coloring
- 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 that solves **3**-coloring
- 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 that solves **3**-coloring
- 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 $T = \Omega(\log^* n)$

Coloring functions (putting things together)

- Given:
  - a T-rounds coloring algorithm that solves 3-coloring
- 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 $T = \Omega(\log^* n)$

If c-coloring can be solved in T rounds,

then 2^c-coloring can be solved in $T-1$ rounds.

Coloring functions (putting things together)

- Given:
  - a **T**-rounds coloring algorithm that solves **3**-coloring
- 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 $T = \Omega(\log^* n)$

If **c**-coloring can be solved in **T** rounds, then 2^c-coloring can be solved in **T-1** rounds.

Round elimination technique

Round elimination technique

- **Given:**

Round elimination technique

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

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

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:**

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 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

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