

- **Weeks 1–2: informal introduction**

- network = path



- **Week 3: graph theory**

- **Weeks 4–7: models of computing**

- what can be computed (efficiently)?

- **Weeks 8–11: lower bounds**

- what cannot be computed (efficiently)?

- **Week 12: recap**

Week 2

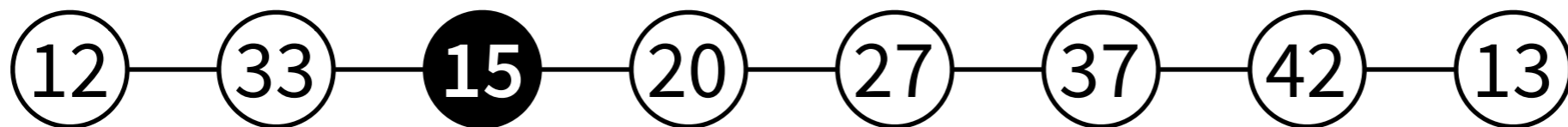
- Warm-up: negative results

Locality

- **Output of a node can only depend on what it knows**
- **After T time steps, a node can only know about things up to distance T**

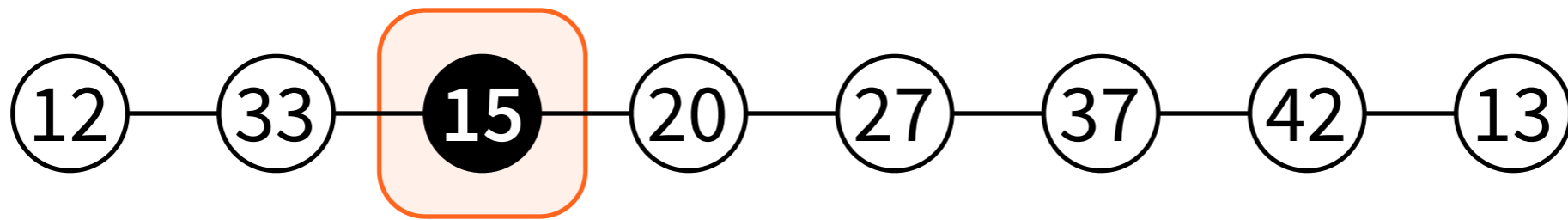
Locality

- **Who knows that node 15 exists?**
 - initially, only node 15
 - everyone else has to learn it by exchanging messages



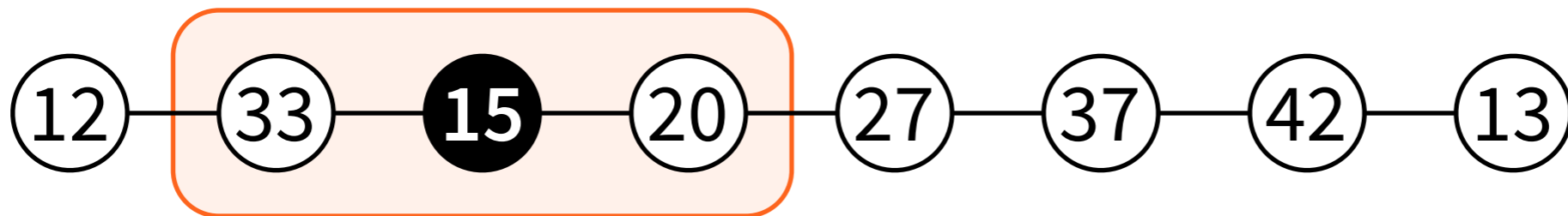
Locality

- **Who knows about node 15 at time $T = 0$?**
 - initial state, before we exchange any messages



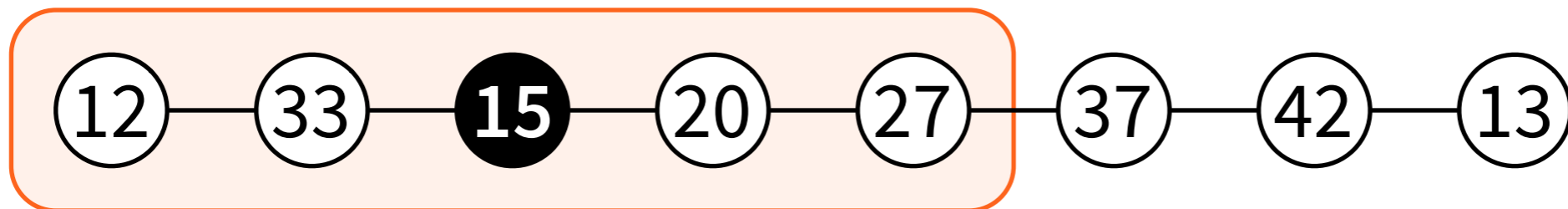
Locality

- **Who knows about node 15 at time $T = 1$?**
 - after 1 communication round



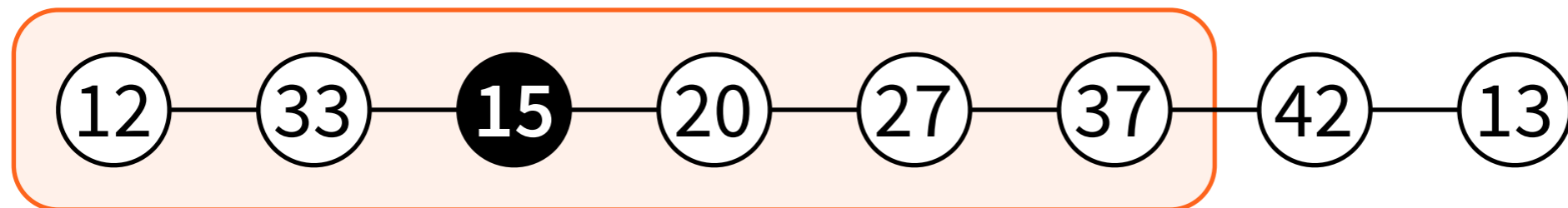
Locality

- **Who knows about node 15 at time $T = 2$?**
 - after 2 communication rounds



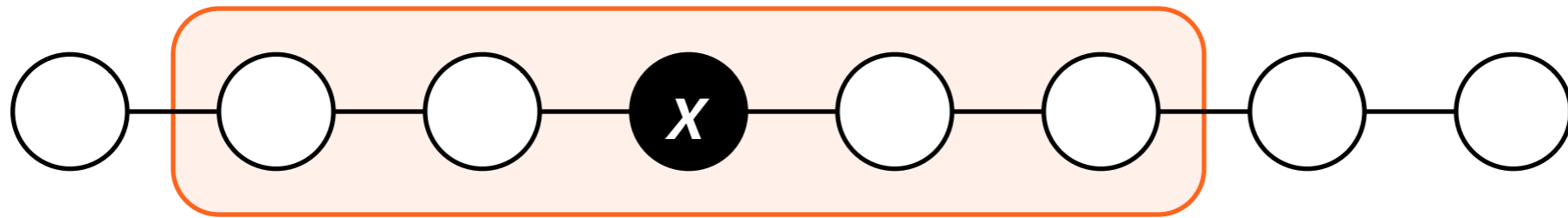
Locality

- **Who knows about node 15 at time $T = 3$?**
 - after 3 communication rounds



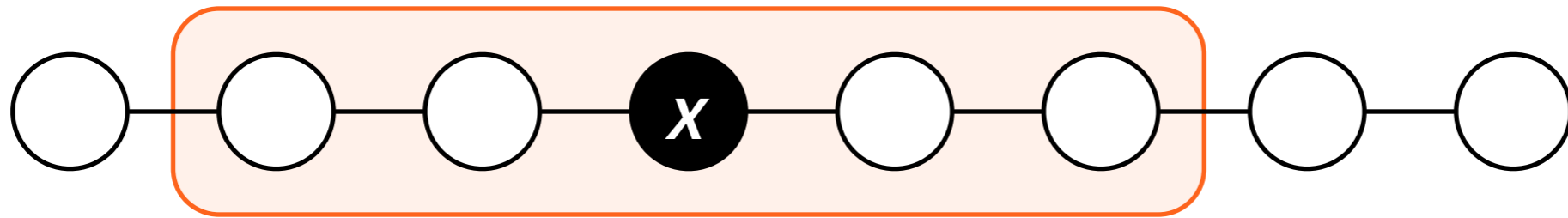
Locality

- After T communication rounds, only nodes up to distance T from node x can know anything about node x
 - distance = “number of hops”



Locality

- After T communication rounds, node x can only know about other nodes that are within distance T from it
 - distance = “number of hops”

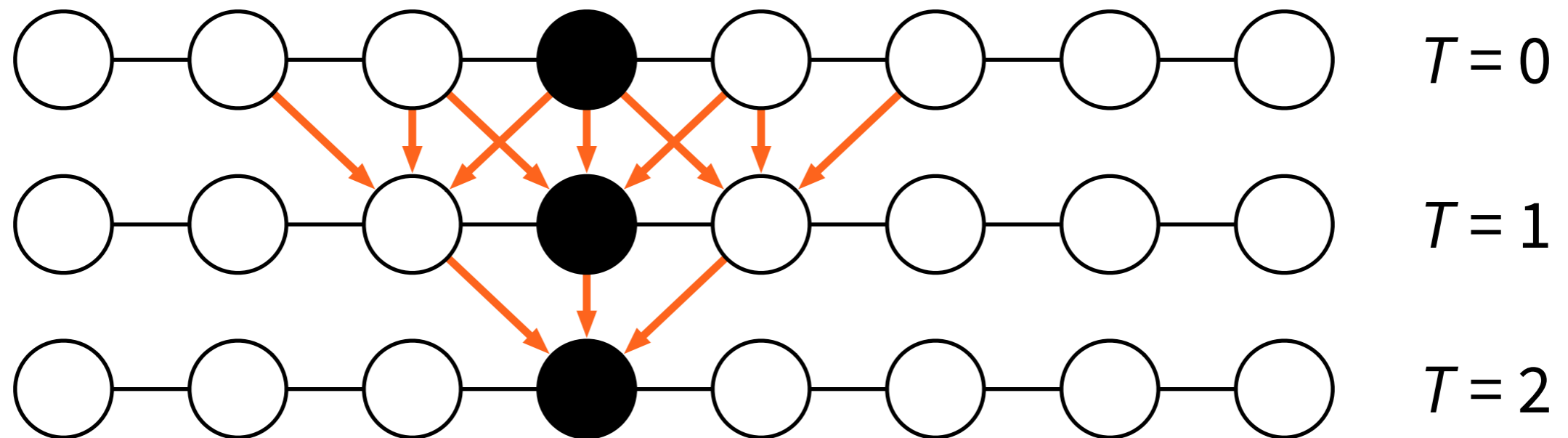


Locality

- **My state at time T only depends on:**
 - my state at time $T - 1$, and
 - messages that I received on round T , which only depend on:
 - the state of my neighbours at time $T - 1$

Locality

- **State at time T only depends on initial information within distance T**

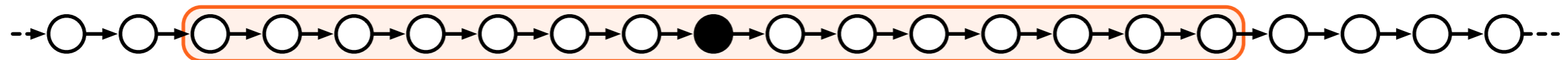


Locality

- **Time = distance**
- **Fast algorithm = “local” algorithm**
 - outputs only depend on local neighbourhoods

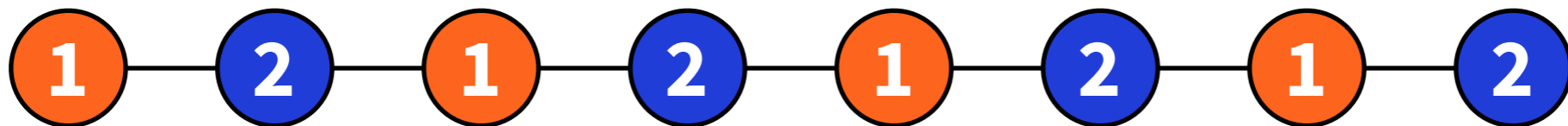
Example: 3-colouring

- Recall: given 128-bit unique identifiers, 3-colouring possible in **7 rounds**
- Equivalently: each node can pick its colour based on what it sees in its **radius-7 neighbourhood**



Using locality to prove lower bounds

- **Example: 2-colouring of a path**
- **Upper bound: possible in time $O(n)$**
- **Lower bound: not possible in time $o(n)$**



Algorithm for 2-colouring

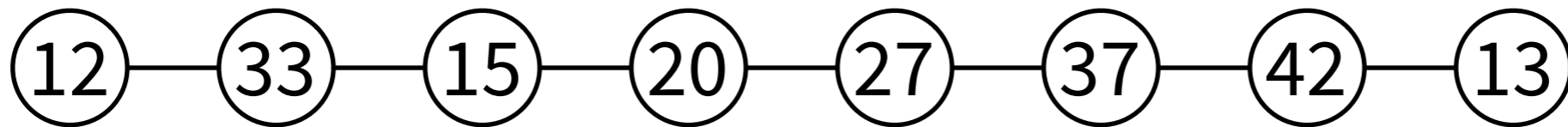
- **Assumption: path, unique identifiers**

Algorithm for 2-colouring

- **Assumption: path, unique identifiers**
- **Two phases:**
 - find the endpoint with smaller identifier
 - starting from this end, assign colours
1, 2, 1, 2, ...

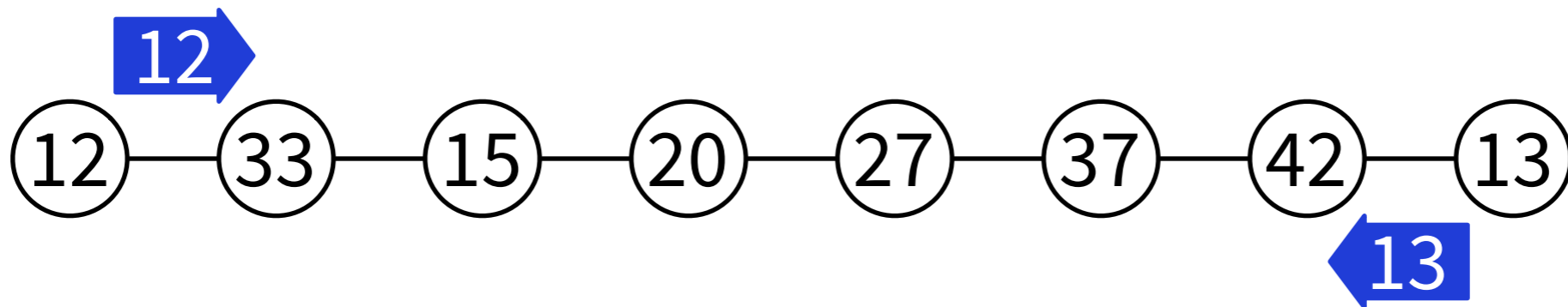
Algorithm for 2-colouring

- **Messages:**
 - “**ID** x ” = there is an endpoint with identifier x
 - “**colour** c ” = my colour is c



Algorithm for 2-colouring

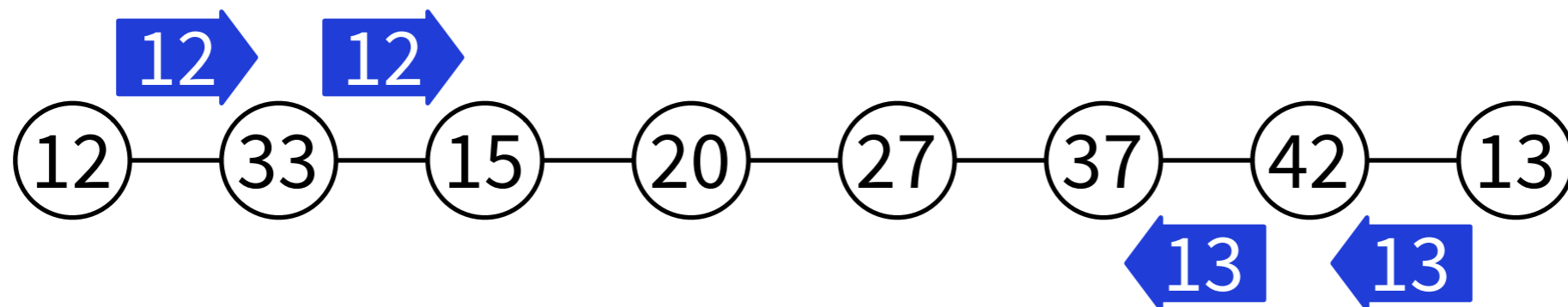
- **Messages:**
 - “**ID x** ” = there is an endpoint with identifier x
 - “**colour c** ” = my colour is c



Algorithm for 2-colouring

- **Messages:**

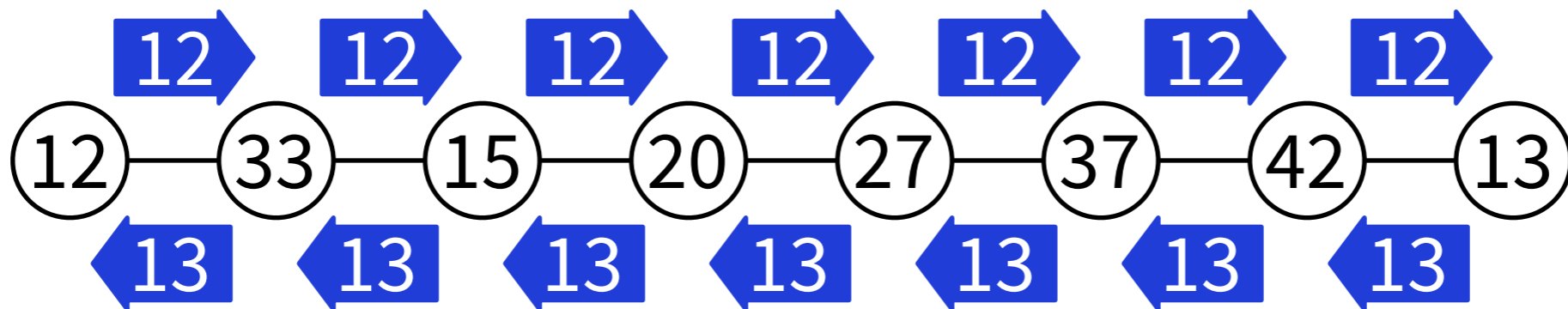
- “**ID** x ” = there is an endpoint with identifier x
- “**colour** c ” = my colour is c



Algorithm for 2-colouring

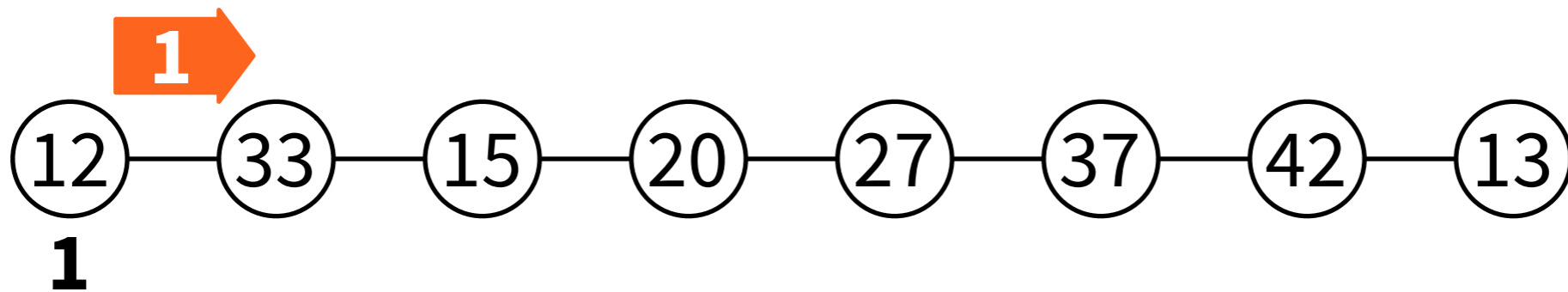
- **Messages:**

- “**ID** x ” = there is an endpoint with identifier x
- “**colour** c ” = my colour is c



Algorithm for 2-colouring

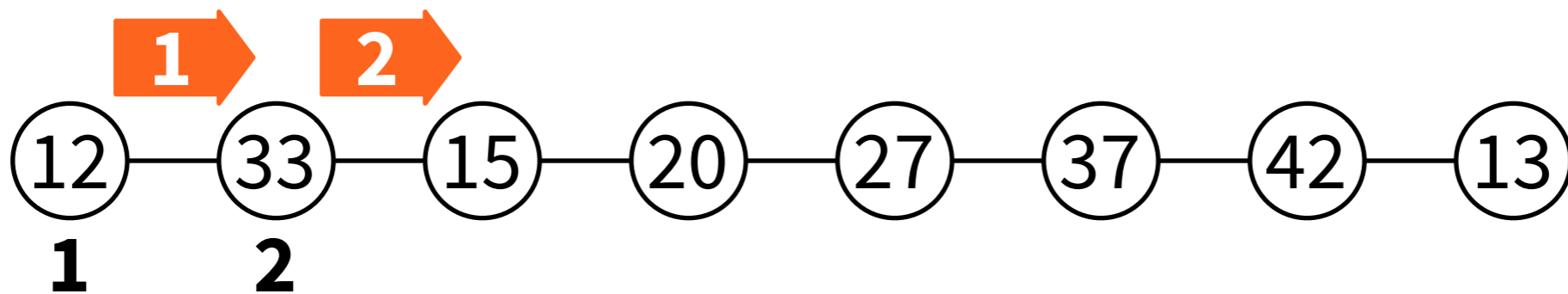
- **Messages:**
 - “**ID** x ” = there is an endpoint with identifier x
 - “**colour** c ” = my colour is c



Algorithm for 2-colouring

- **Messages:**

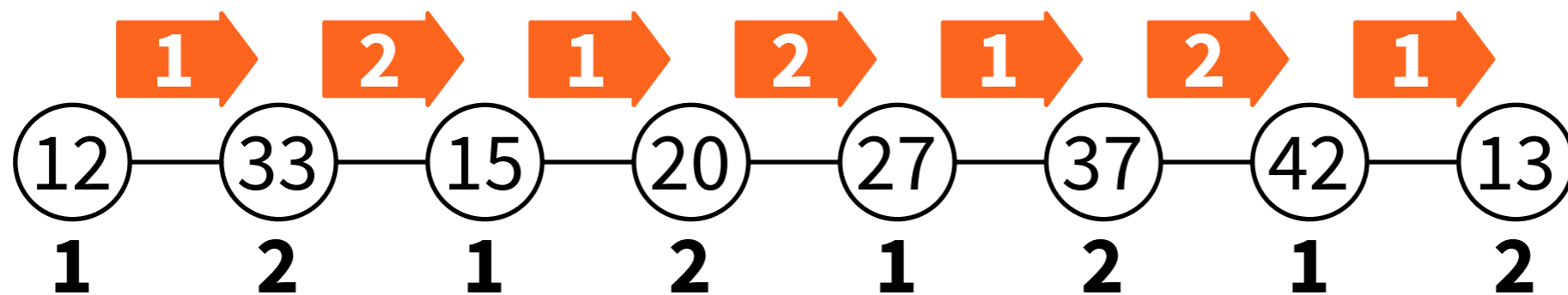
- “**ID** x ” = there is an endpoint with identifier x
- “**colour** c ” = my colour is c



Algorithm for 2-colouring

- **Messages:**

- “**ID** x ” = there is an endpoint with identifier x
- “**colour** c ” = my colour is c



Algorithm for 2-colouring

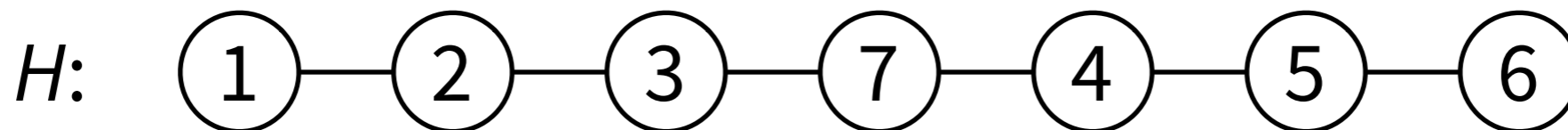
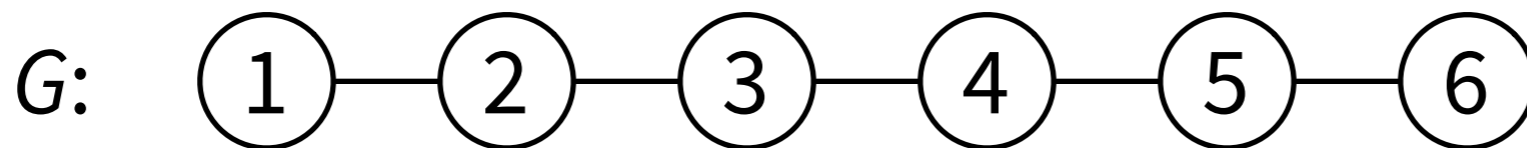
- **2-colouring possible in $O(n)$ rounds**
- **Goal: prove that this is optimal!**
 - there is no algorithm that finds a 2-colouring in time $o(n)$
 - assumptions: path, unique identifiers

Lower bound for 2-colouring

- **Assume: there is an $o(n)$ -time algorithm A**
- **For large n , running time $\ll n/2$**
- **Idea: construct **two possible worlds**,
show that A fails in one of them**

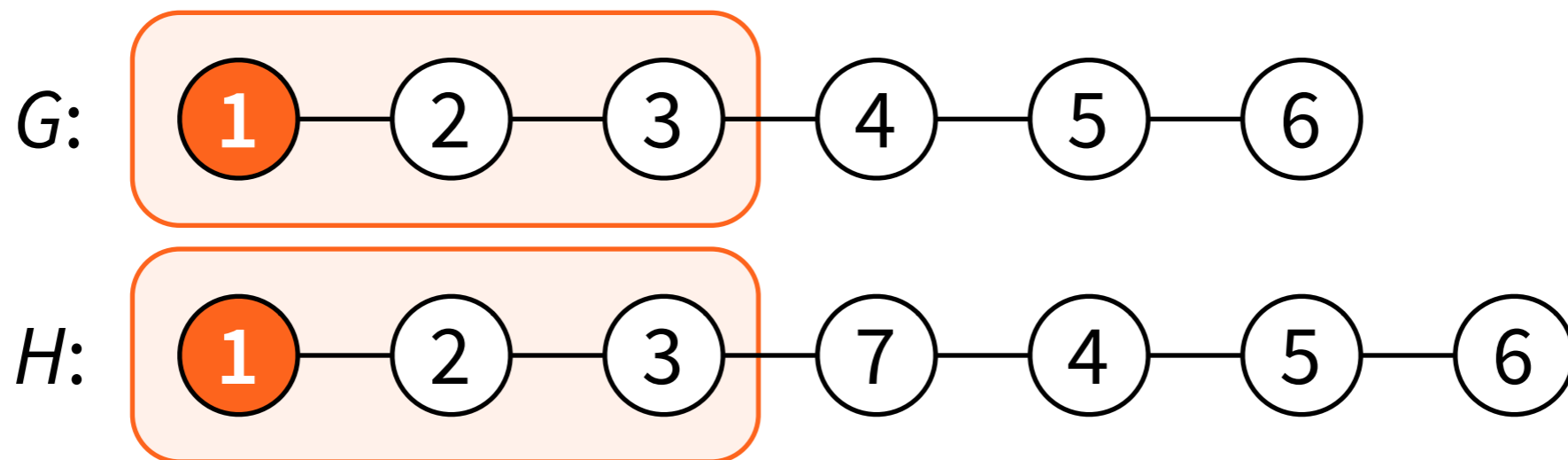
Lower bound for 2-colouring

- Long paths with $2k$ and $2k+1$ nodes,
algorithm runs in $\leq k-1$ rounds



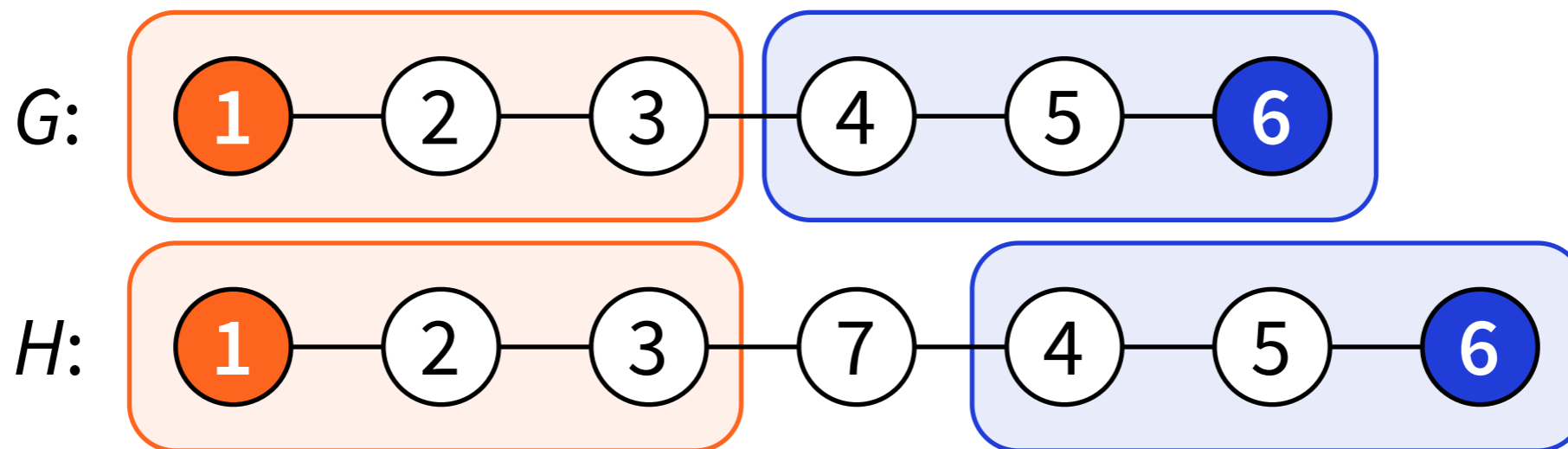
Lower bound for 2-colouring

- **Same $(k-1)$ -neighbourhood, same output**



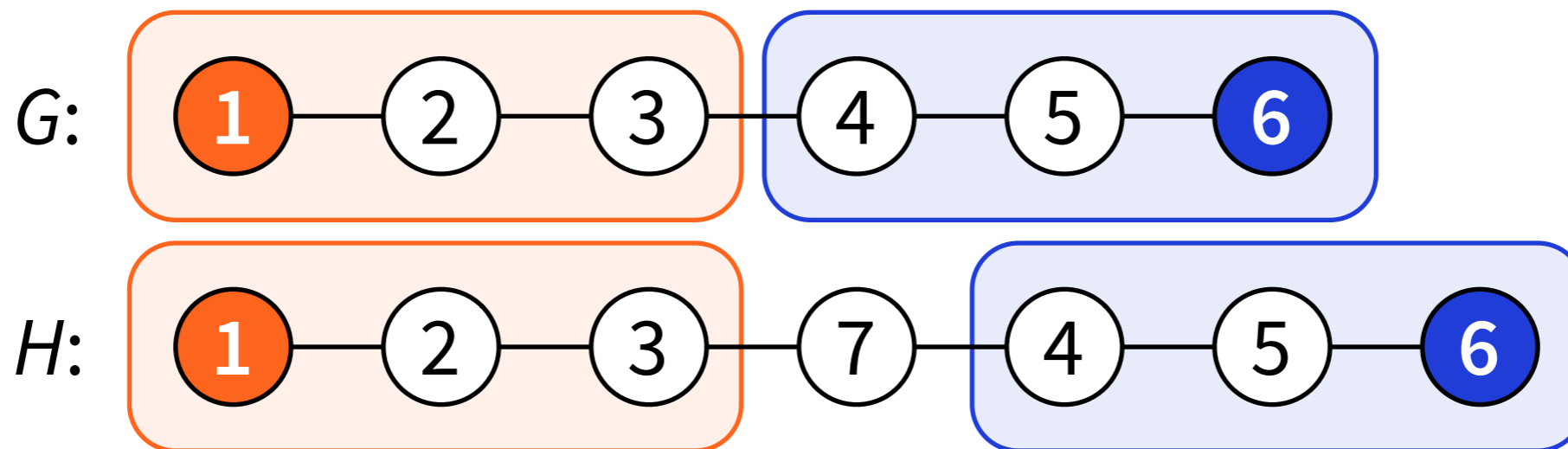
Lower bound for 2-colouring

- **Same $(k-1)$ -neighbourhood, same output**



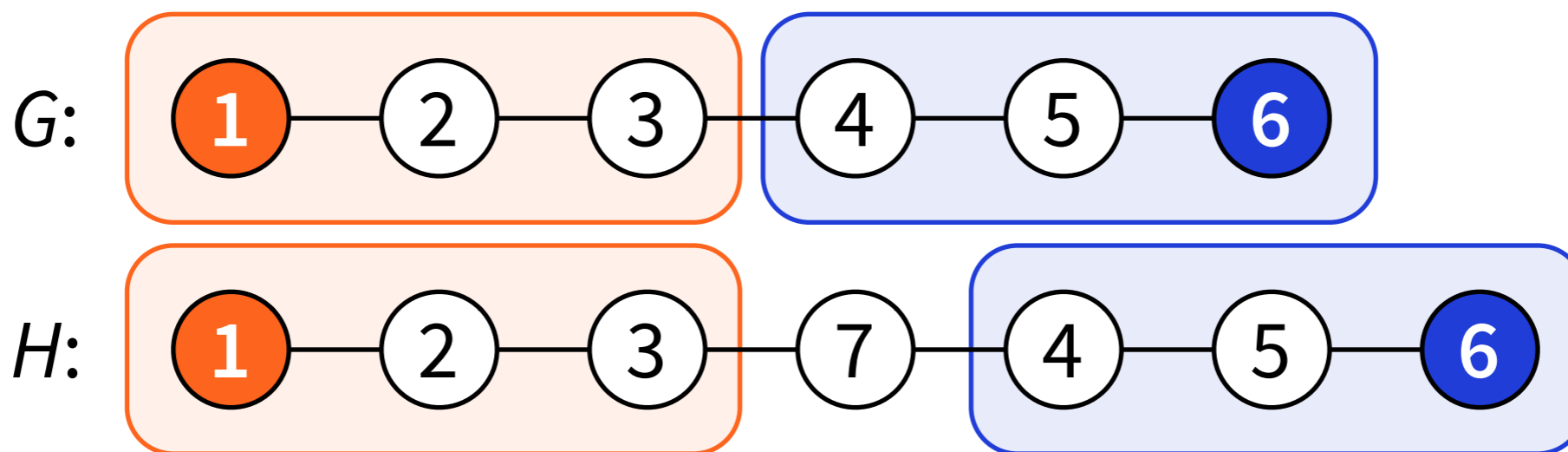
Lower bound for 2-colouring

- **Contradiction – why?**



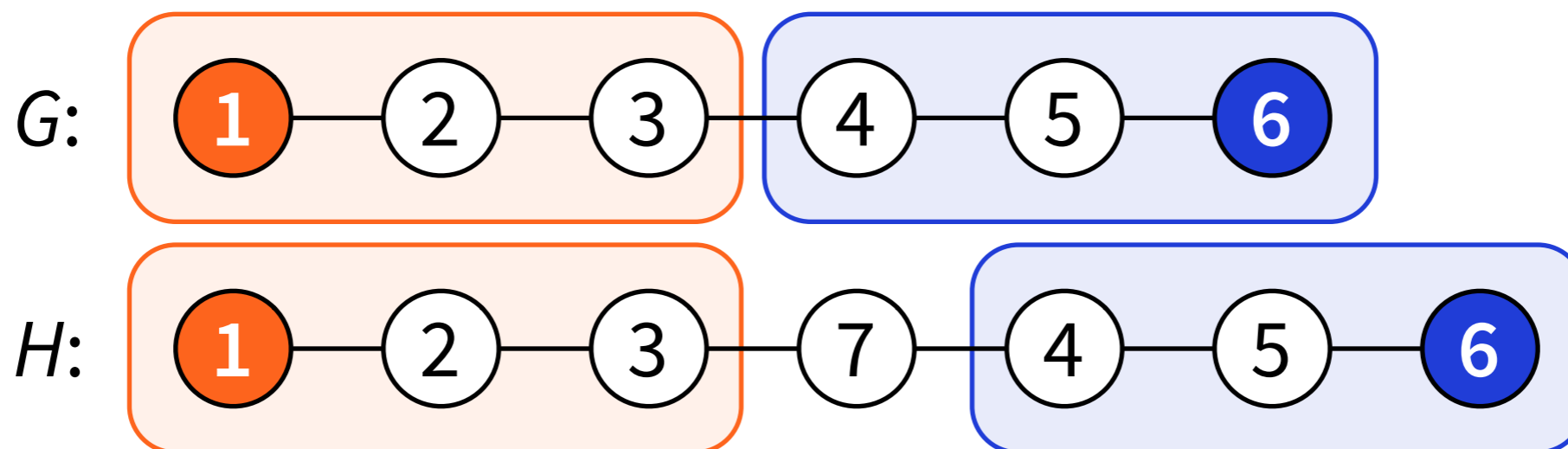
Lower bound for 2-colouring

- ***G***: nodes 1 and 6 must have different colours
- ***H***: nodes 1 and 6 must have the same colour



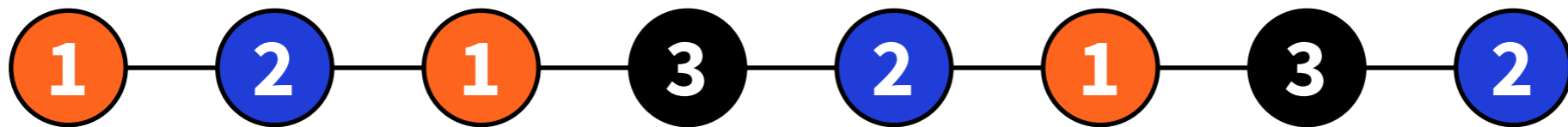
Lower bound for 2-colouring

- **Conclusion: there is no algorithm that finds a 2-colouring of a path in time $o(n)$**



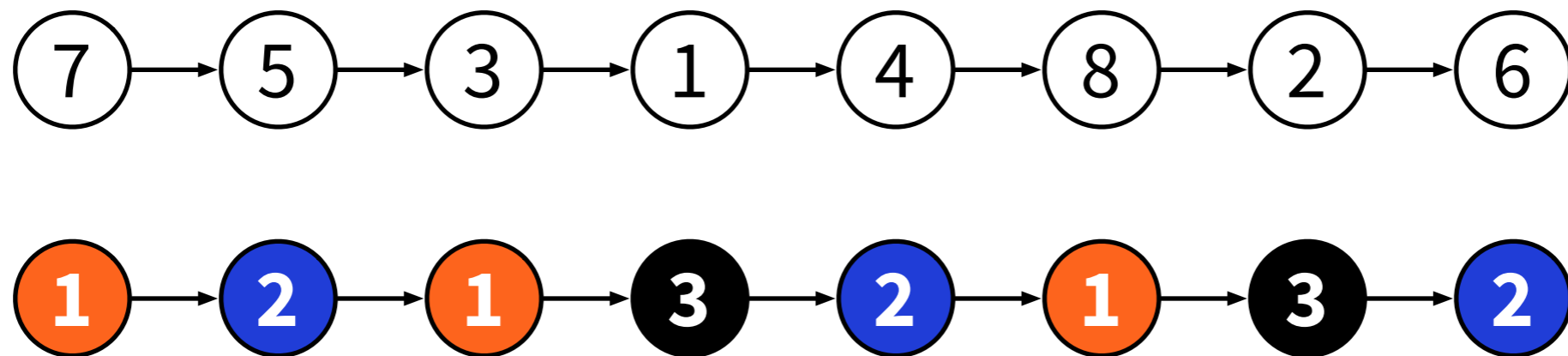
Using locality to prove lower bounds

- **Example: 3-colouring of a path**
- **Upper bound: possible in time $O(\log^* n)$**
- **Lower bound: not possible in time $o(\log^* n)$**



Lower bound for 3-colouring

- **Given: directed path with n nodes, identifiers are a permutation of $\{1, 2, \dots, n\}$**

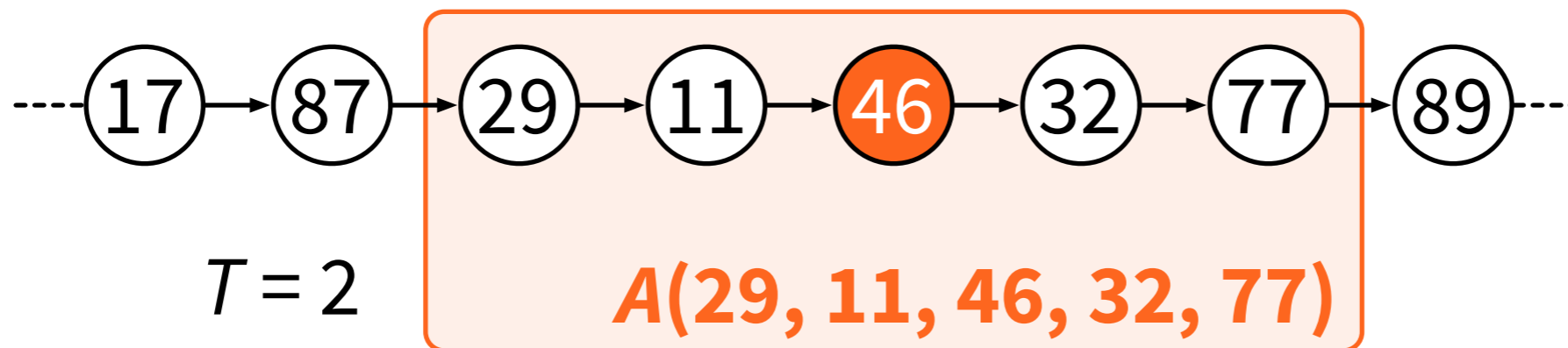


Lower bound for 3-colouring

- **Given:** directed path with n nodes, identifiers are a permutation of $\{1, 2, \dots, n\}$
- **Assume:** there is an algorithm A that finds a 3-colouring in time T
- **Goal:** prove that $T \geq \frac{1}{2} \log^*(n) - 1$

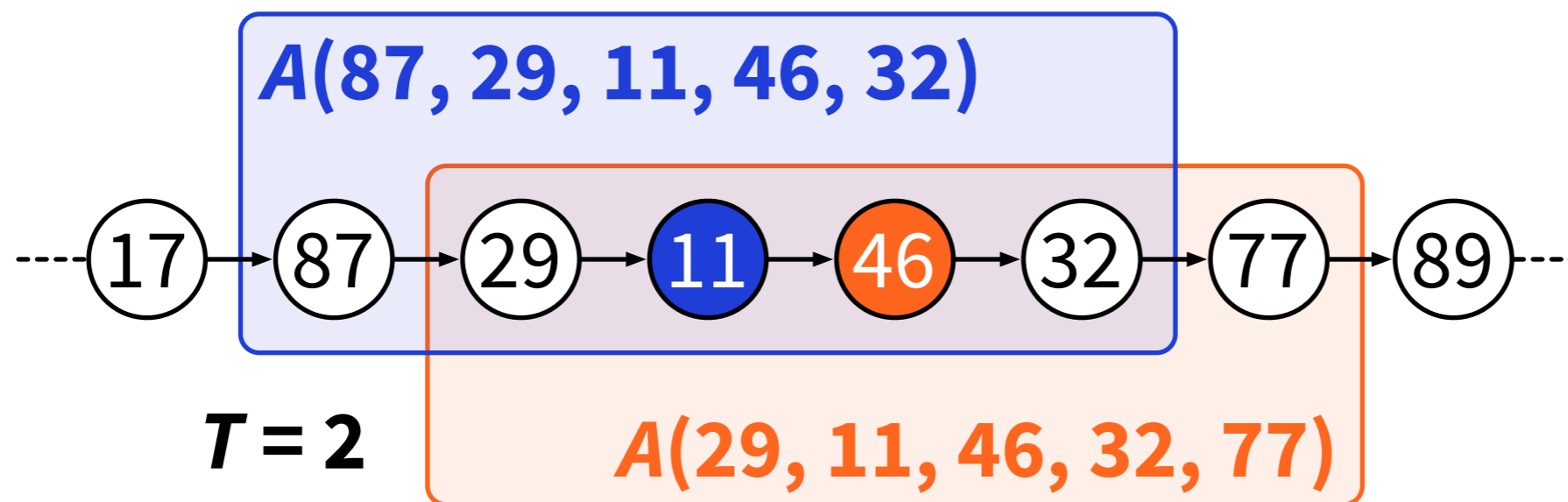
Algorithm for 3-colouring paths

- Running time $T = \text{output}$ only depends on radius- T neighbourhood of the node
- Algorithm = **k -ary function** where $k = 2T+1$



Algorithm for 3-colouring paths

$$A(87, 29, 11, 46, 32) \neq A(29, 11, 46, 32, 77)$$



Algorithm for c -colouring paths

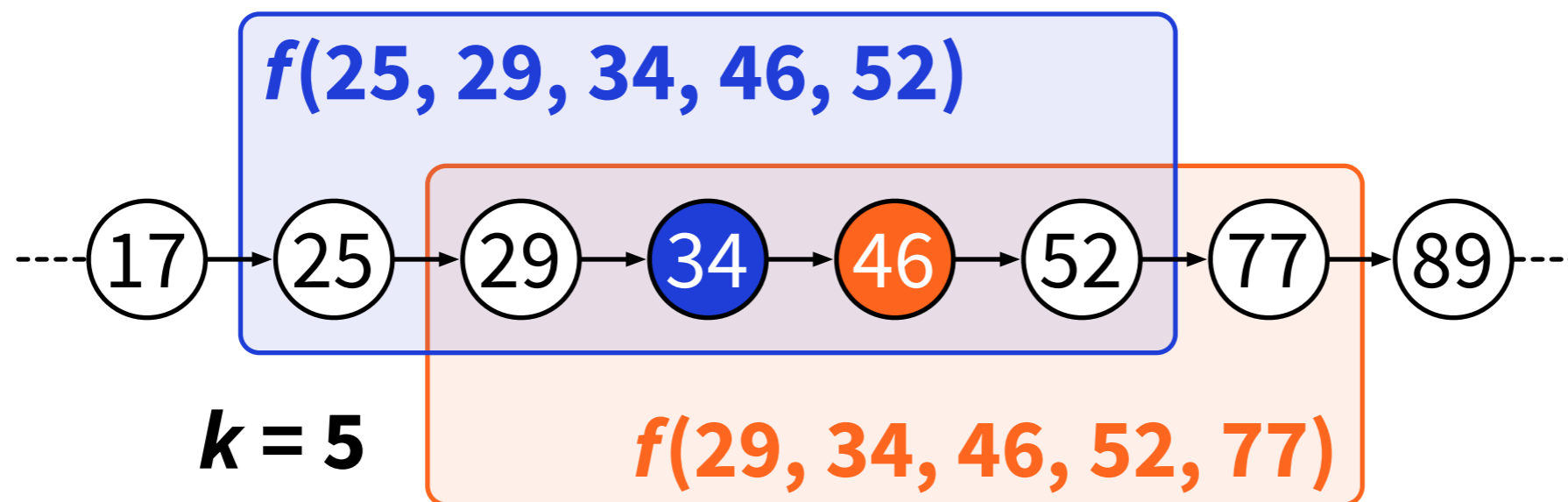
- $A(x_1, \dots, x_k) \in \{1, \dots, c\}$
for all distinct $x_1, \dots, x_k \in \{1, \dots, n\}$
- $A(x_1, \dots, x_k) \neq A(x_2, \dots, x_{k+1})$
for all distinct $x_1, \dots, x_{k+1} \in \{1, \dots, n\}$

Definition: “***k*-ary *c*-colouring function**”

- $f(x_1, \dots, x_k) \in \{1, \dots, c\}$
for all $1 \leq x_1 < \dots < x_k \leq n$
- $f(x_1, \dots, x_k) \neq f(x_2, \dots, x_{k+1})$
for all $1 \leq x_1 < \dots < x_{k+1} \leq n$
- We only care what happens
with **increasing identifiers**

k-ary c-colouring function

$$f(25, 29, 34, 46, 52) \neq f(29, 34, 46, 52, 77)$$



***k*-ary c-colouring function**

- **Assume: *A* is a distributed algorithm that finds a 3-colouring in directed *n*-cycles in time *T***
- **Then: *A* is also a *k*-ary 3-colouring function for $k = 2T + 1$**
- **Plan: show that $k + 1 \geq \log^* n$**

Lemma 1

- If f is a **1**-ary **c** -colouring function, then **$c \geq n$**
- **Intuition:**
 - you cannot do anything useful without ***some communication***

Lemma 1

- If f is a **1**-ary **c** -colouring function, then **$c \geq n$**
- **Proof:**
 - pigeonhole principle
 - if $c < n$, there is a collision $f(x) = f(y)$ for some $1 \leq x < y \leq n$, contradiction

Lemma 2

- If f is a k -ary c -colouring function, then we can construct a $(k - 1)$ -ary 2^c -colouring function g
- Intuition:
 - we can always construct a *faster* algorithm, if we can use a *larger colour palette*

Lemma 2

- If f is a k -ary c -colouring function, then we can construct a $(k - 1)$ -ary 2^c -colouring function g
- **Proof:**
 - $g'(x_1, \dots, x_{k-1}) = \{f(x_1, \dots, x_{k-1}, y) : y \geq x_{k-1}\}$
 - $g(x_1, \dots, x_{k-1}) = h(g'(x_1, \dots, x_{k-1}))$
 - $h =$ bijection that maps sets to colours

Lemma 2: proof

- $g'(x_1, \dots, x_{k-1}) = \{f(x_1, \dots, x_{k-1}, y) : y > x_{k-1}\}$
- $g(x_1, \dots, x_{k-1}) = h(g'(x_1, \dots, x_{k-1}))$
- $h =$ bijection that maps sets to colours
- **By construction:** $g(x_1, \dots, x_{k-1}) \in \{1, \dots, 2^c\}$
- **Need to show:** $g(x_1, \dots, x_{k-1}) \neq g(x_2, \dots, x_k)$
for all $1 \leq x_1 < \dots < x_k \leq n$

Lemma 2: proof

- $g'(x_1, \dots, x_{k-1}) = \{f(x_1, \dots, x_{k-1}, y) : y > x_{k-1}\}$
- **Need to show:** $g'(x_1, \dots, x_{k-1}) \neq g'(x_2, \dots, x_k)$
for all $1 \leq x_1 < \dots < x_k \leq n$

Lemma 2: proof

- $1 \leq x_1 < x_2 < \dots < x_k \leq n$
- $g'(x_1, \dots, x_{k-1}) = \{f(x_1, \dots, x_{k-1}, y) : y > x_{k-1}\}$
- $g'(x_2, \dots, x_k) = \{f(x_2, \dots, x_k, z) : z > x_k\}$
- $f(x_1, \dots, x_{k-1}, x_k) \in g'(x_1, \dots, x_{k-1})$
- $f(x_1, \dots, x_{k-1}, x_k) \notin g'(x_2, \dots, x_k)$
- $g'(x_1, \dots, x_{k-1}) \neq g'(x_2, \dots, x_k)$

Lemma 2

- If f is a k -ary c -colouring function, then we can construct a $(k - 1)$ -ary 2^c -colouring function g

Iterate Lemma 2

$$i_2 = 2^{2^{\dots^2}} \text{ (} i \text{ twos)}$$

- **k -ary 3 -colouring function \rightarrow**
- k -ary $^2 2$ -colouring function \rightarrow**
- $(k - 1)$ -ary $^3 2$ -colouring function \rightarrow**
- $(k - 2)$ -ary $^4 2$ -colouring function \rightarrow**
- $(k - 3)$ -ary $^5 2$ -colouring function \rightarrow**
- \dots**
- 1 -ary $^{k+1} 2$ -colouring function**

Lemma 1 + Lemma 2

$$i_2 = 2^{2^{\dots^2}} \quad (i \text{ twos})$$

- **Lemma 2:**

- k -ary **3**-colouring function \rightarrow
1-ary $k+1$ **2**-colouring function

- **Lemma 1:**

- $k+1$ **2** $\geq n$ (that is, $k+1 \geq \log^* n$)

Lower bound for 3-colouring

- **Assume:** A is a distributed algorithm that finds a **3-colouring in directed n -paths in time T**
- **Then:** A is also a **k -ary 3-colouring function** for $k = 2T + 1$
- **Then:** $k + 1 \geq \log^* n$,
therefore: $T \geq \frac{1}{2} \log^*(n) - 1$

Conclusions: tight bounds

- **2-colouring paths:**
 - possible in time $O(n)$
 - not possible in time $o(n)$
- **3-colouring paths:**
 - possible in time $O(\log^* n)$
 - not possible in time $o(\log^* n)$

Assuming:
directed path,
unique IDs =
 $\{1, 2, \dots, n\}$

Conclusions: tight bounds

- **2-colouring paths:**

- possible in time $O(n)$
- not possible in time $o(n)$

- **3-colouring paths:**

- possible in time $O(\log^* n)$
- not possible in time $o(\log^* n)$

Richard Cole and
Uzi Vishkin (1986)

Nathan Linial (1992)

- **Weeks 1–2: informal introduction**

- network = path



- **Week 3: graph theory**

- **Weeks 4–7: models of computing**

- what can be computed (efficiently)?

- **Weeks 8–11: lower bounds**

- what cannot be computed (efficiently)?

- **Week 12: recap**