

CS-E4510

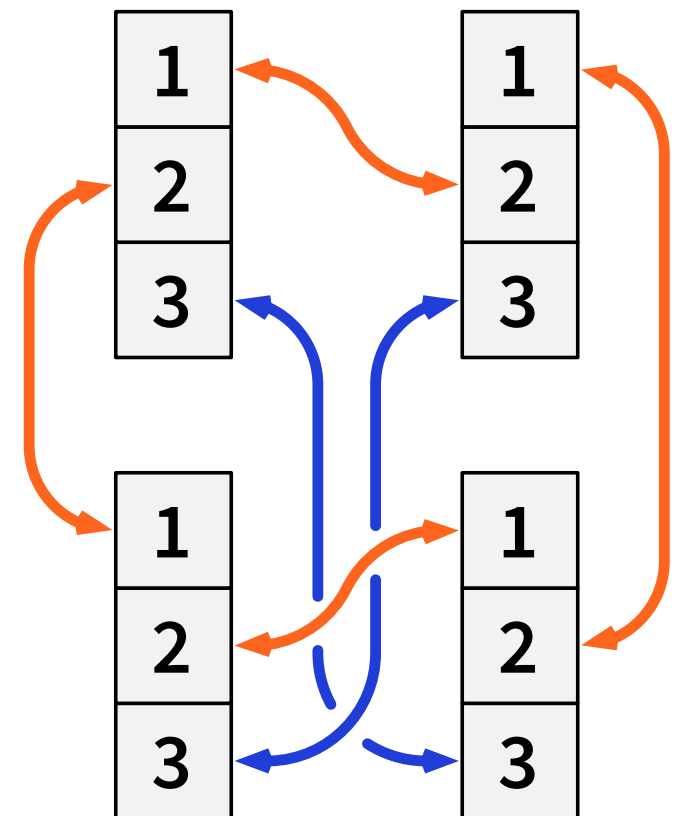
Distributed Algorithms

Jukka Suomela

Aalto University

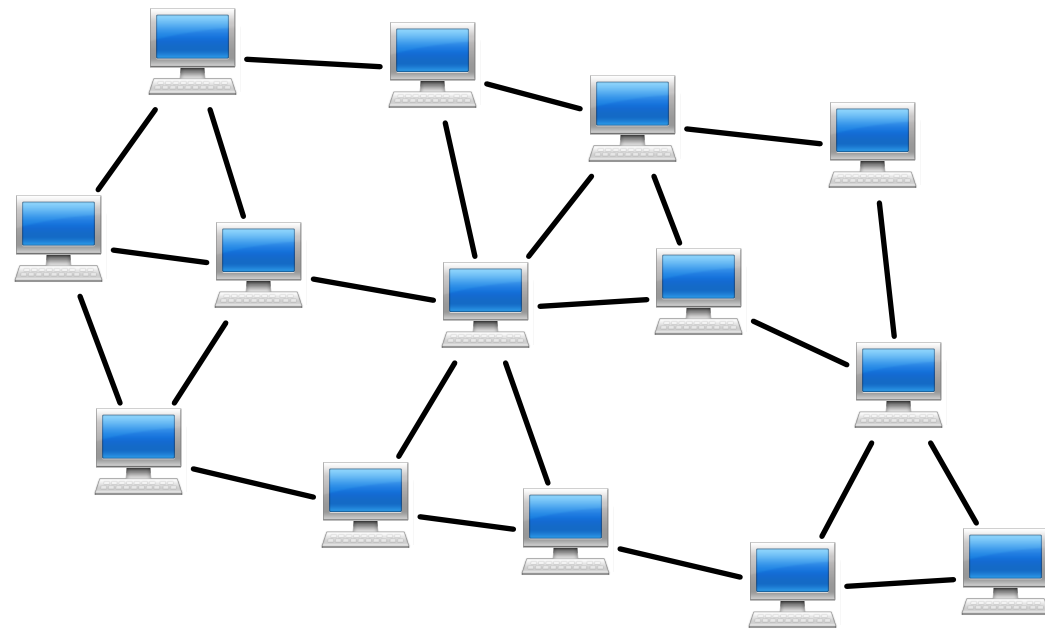
Autumn 2017

iki.fi/suo/da



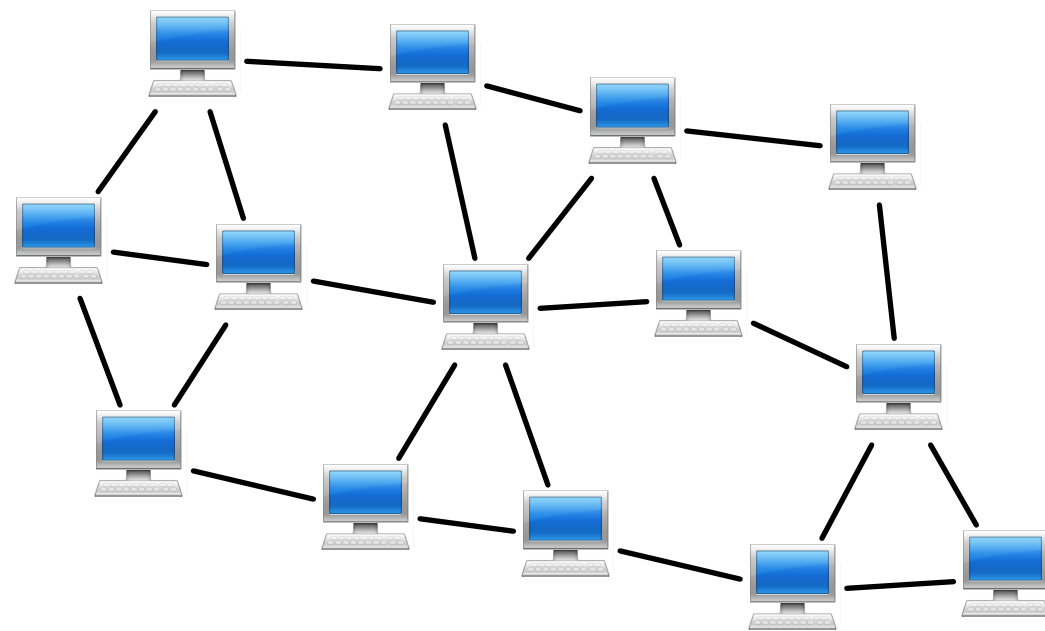
Distributed Algorithms

Algorithms for computer networks



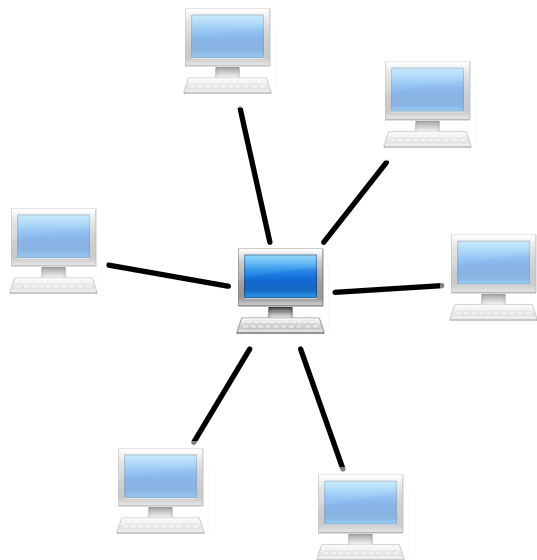
Distributed Algorithms

Identical computers in an **unknown network**,
all running the **same algorithm**



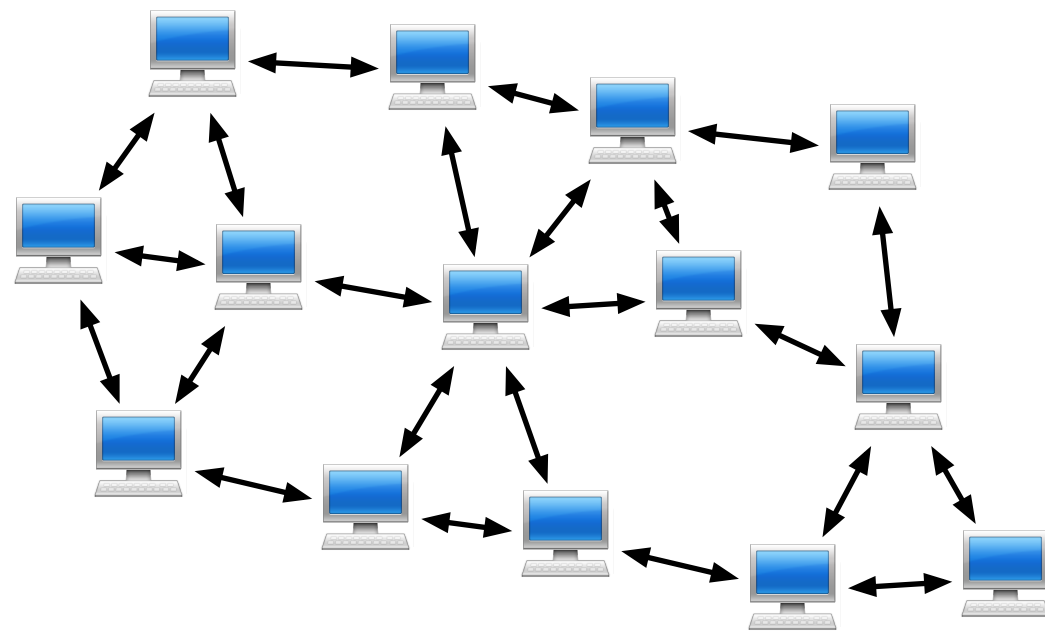
Distributed Algorithms

Initially each computer only aware of its immediate neighbourhood



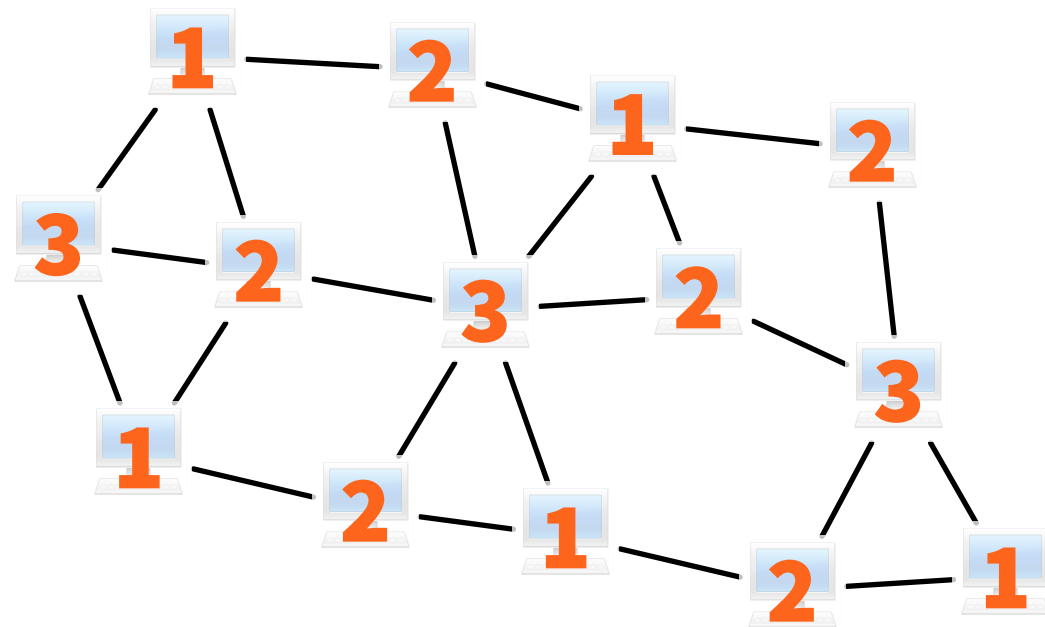
Distributed Algorithms

**Nodes can exchange messages
with their neighbours to learn more...**



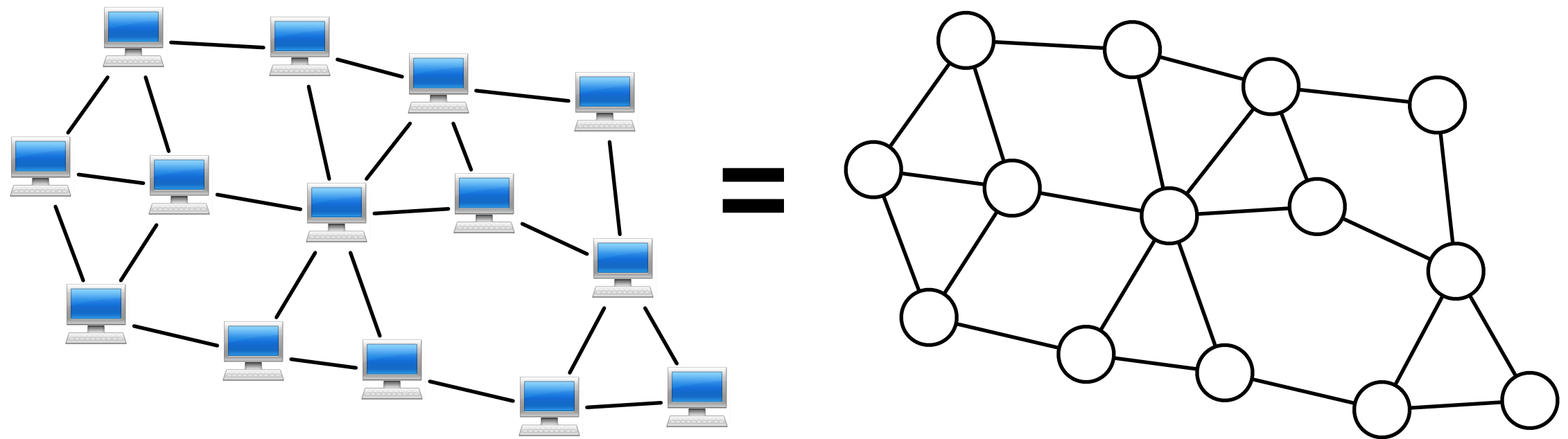
Distributed Algorithms

Finally, each computer has to stop and produce its own **local output**



Distributed Algorithms

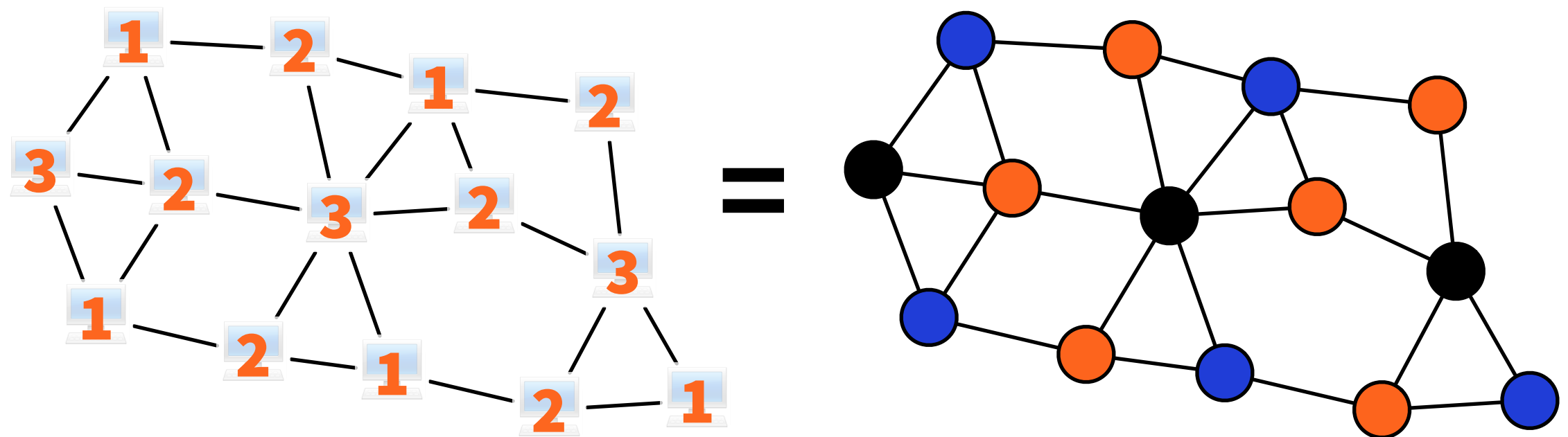
**Focus on graph problems:
network topology = input graph**



Distributed Algorithms

Focus on graph problems:

local outputs = solution (here: graph colouring)



Distributed Algorithms

Typical research question:

“How fast can we solve graph problem X ?”

Time = number of communication rounds

Why?

- 1. Applications in large-scale real-world communication networks**

Why?

- 1. Applications in large-scale real-world communication networks**
- 2. New perspective to theory of computation**

New perspective to theory of computing

- **New kinds of computational resources:**
 - old: **time & space**
 - new: **distance & bandwidth**
- **New kinds of algorithm design challenges:**
 - **parallelism & coordination**

Why?

- 1. Applications in large-scale real-world communication networks**
- 2. New perspective to theory of computation**
- 3. Understanding nature**

Understanding nature: **Algorithmic lens**

- **Distributed systems in different areas:**
 - **sociology**: collaboration networks
 - **economy**: job markets, auctions
 - **ecology**: animal populations
 - **biology**: organs, tissues
 - **chemistry**: chemical reactions ...

Understanding nature:

Algorithmic lens

- **Natural science perspective:**
 - fix a process and analyse it
- **Computational perspective:**
 - fix a goal (“computational problem”)
 - ask if there is **any** process (“algorithm”) that reaches the goal efficiently

Understanding nature: **Algorithmic lens**

- **Model nature as a distributed system**
- **Proving that something cannot be done efficiently with distributed algorithms: discovering fundamental limitations of nature**
 - producing hypotheses: “this process is slow (or our model of nature is wrong)”

Why?

- 1. Applications in large-scale real-world communication networks**
- 2. New perspective to theory of computation**
- 3. Understanding nature**

- **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**

Practicalities

- All practical information in **MyCourses**
- **Textbook:**
 - freely available online
- **Exercises:**
 - every week, starting this week!

Weekly exercises

- **Tuesday at noon:** quiz (2 points)
- **Wednesday at midnight:** 1 exercise (2 points)
- **Friday at midnight:** 2 exercises (2+2 points)
- **Whenever you want:** challenging exercises (4 points each)

Grading

- **Two midterm exams:** pass/fail
- **Weekly exercises:** max **96** points (+ extra)
- **Grading:**
 - grade **1/5**: pass exams
 - grade **5/5**: pass exams + at least **80** points

Learning objectives

- **Models of distributed computing**
- **Algorithm design and analysis**
- **Computability and computational complexity**
- **Graph theory**

WARNING: THEORY

100% mathematics

(definitions, theorems, proofs...)

0% practice

(programming, hardware, protocols...)

Week 1

- Warm-up: positive results

Running example: **3-colouring a path**

Given a path:



Output a proper 3-colouring, e.g.:



Model of computing:

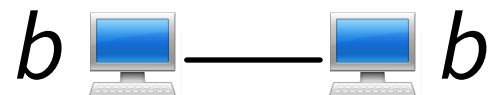
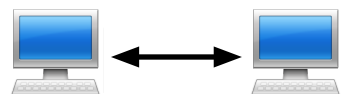
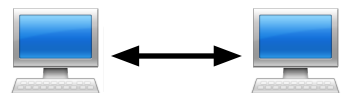
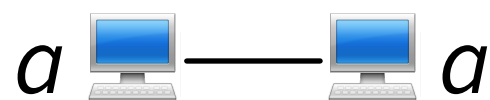
Send, receive, update

- **All nodes in parallel:**
 - send messages to their neighbours
 - receive messages from neighbours
 - update their state
- **Stopping state = final output**
 - can send/receive, but not update any more

Challenge:

Symmetry breaking

- **Identical nodes, everything deterministic and synchronised: cannot break symmetry**



same initial state

same messages sent

same messages received

same new state

...

same output

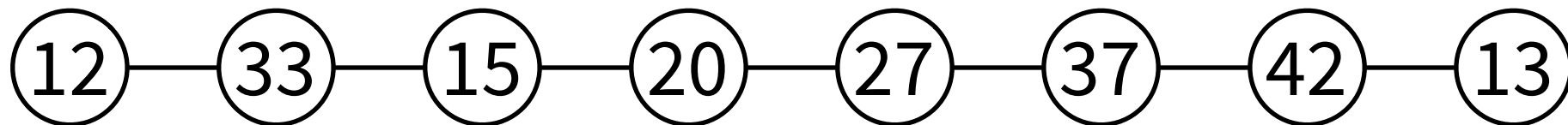
Challenge:

Symmetry breaking

- **Identical nodes, everything deterministic and synchronised: cannot break symmetry**
- **Solutions:**
 - assume unique identifiers
 - use randomised algorithms

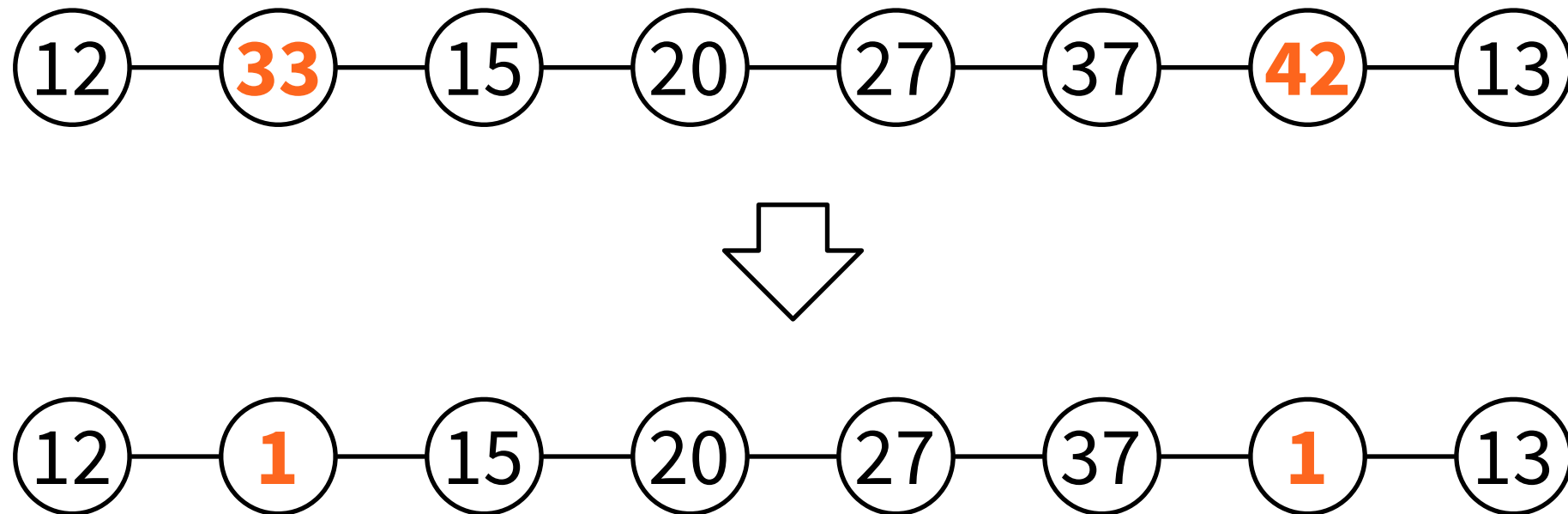
Algorithm P3C: **Using unique IDs**

- **Unique IDs = proper colouring with large number of colours**
- **Goal: reduce the number of colours**



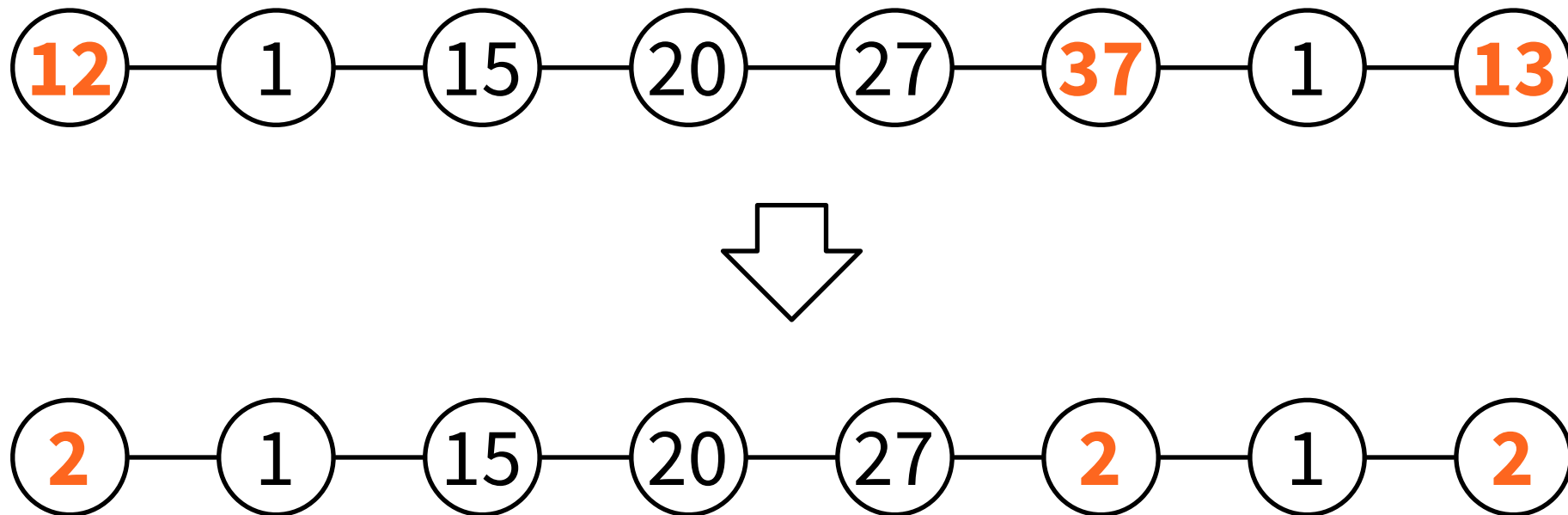
Algorithm P3C: Using unique IDs

- Idea: **local maxima** pick a new colour



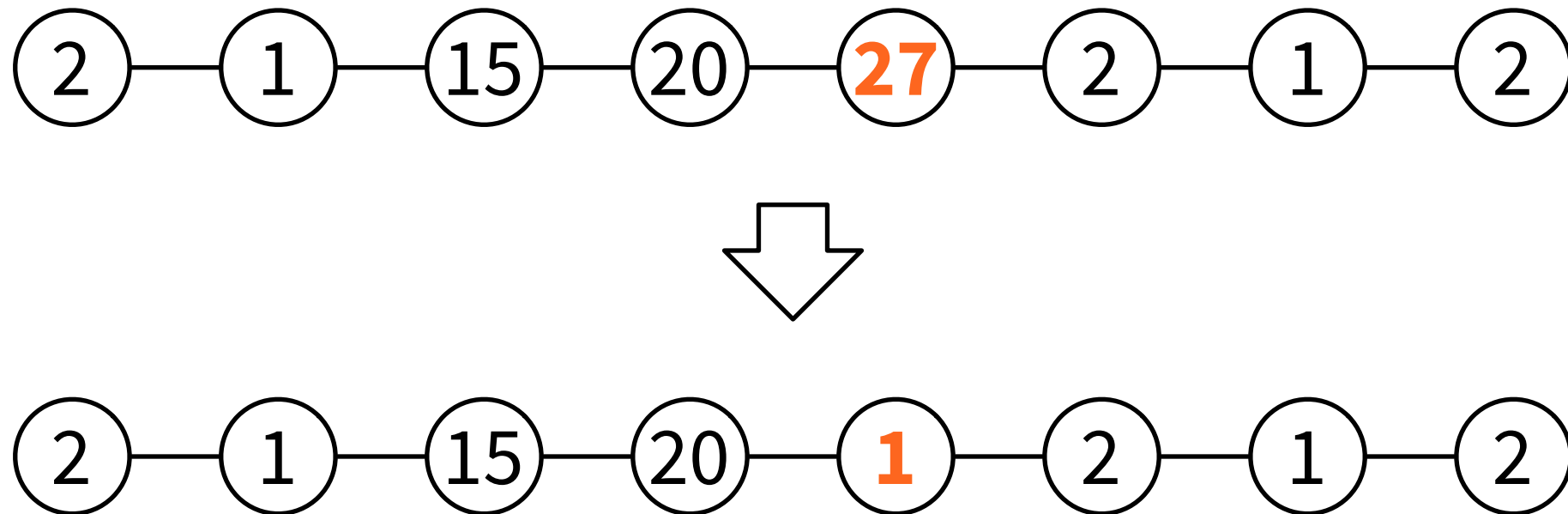
Algorithm P3C: Using unique IDs

- Idea: **local maxima** pick a new colour



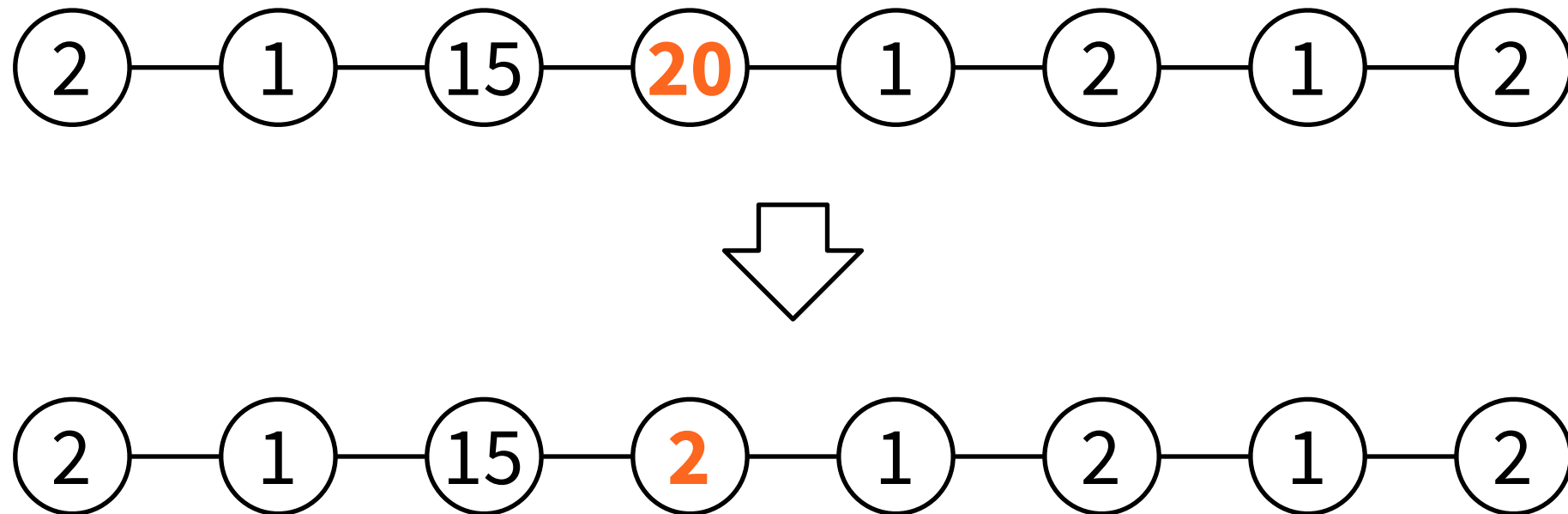
Algorithm P3C: Using unique IDs

- Idea: **local maxima** pick a new colour



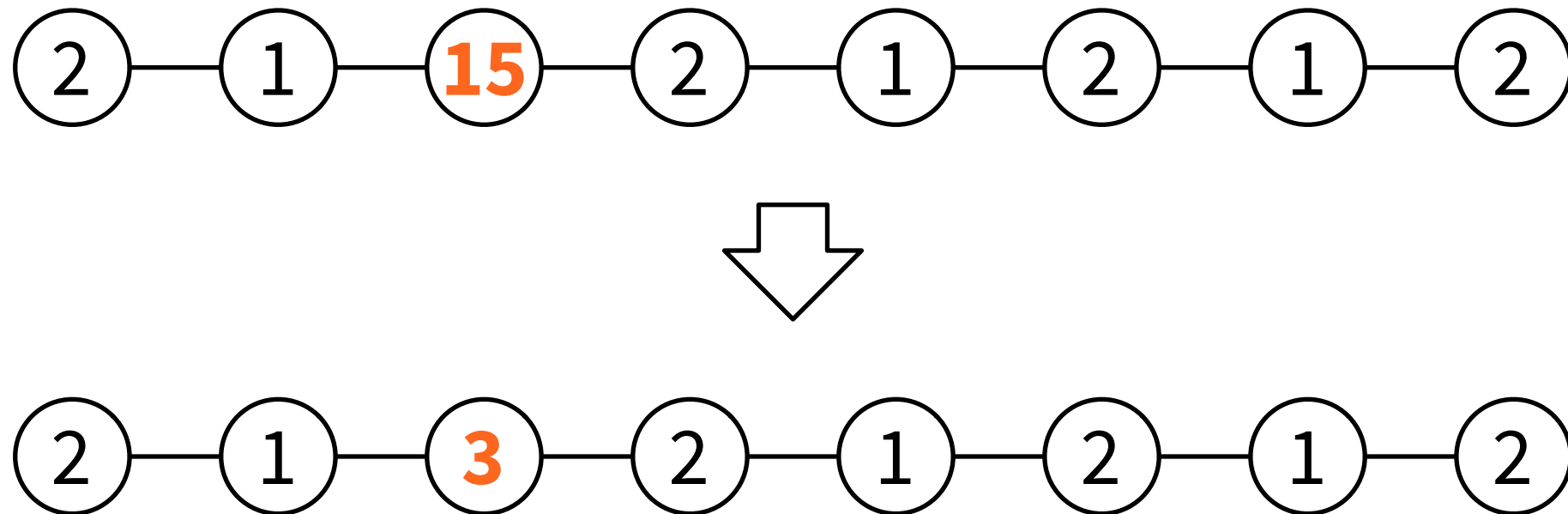
Algorithm P3C: Using unique IDs

- Idea: **local maxima** pick a new colour



Algorithm P3C: Using unique IDs

- Idea: **local maxima** pick a new colour



Algorithm P3C:

Using unique IDs

- **Inform neighbours of your current colour**
- **If your colour $>$ colours of your neighbours:**
 - pick a free colour from $\{1, 2, 3\}$
that is not used by any neighbour
- **Stopping states = $\{1, 2, 3\}$**

Performance

- **P3C: worst case $O(n)$**
- **We can do better!**

Algorithm P3CRand: **Using randomness**

- **Initialise: state = unhappy, colour = 1**
- **While state = unhappy:**
 - pick a new random colour from {1, 2, 3}
 - compare colours with neighbours
 - if different, set state = happy

Performance

- **P3C: worst case $O(n)$**
- **P3CRand: $O(\log n)$ with high probability**
- **We can do better!**
 - and we do not even need randomness

Algorithm P3CBit:

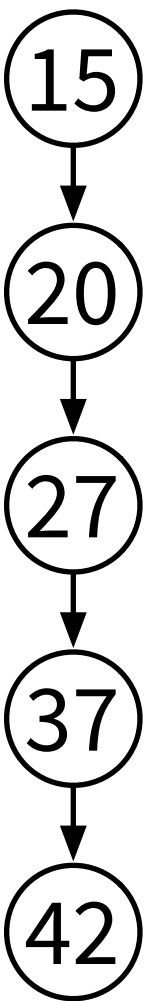
Fast colour reduction

- **Unique IDs = proper colouring with large number of colours**
- **Idea: reduce the number of colours from 2^k to $2k$ in one step**

Algorithm P3CBit:

Fast colour reduction

- **Unique IDs = proper colouring with large number of colours**
- **Idea: reduce the number of colours from 2^k to $2k$ in one step**
- **Note: we will assume a **directed path!****
(general case left as an exercise)



Algorithm P3CBit:

Fast colour reduction

- **Example: 128-bit unique IDs**
 - $2^{128} \rightarrow 2 \cdot 128 = 2^8$ colours
 - $2^8 \rightarrow 2 \cdot 8 = 2^4$ colours
 - $2^4 \rightarrow 2 \cdot 4 = 2^3$ colours
 - $2^3 \rightarrow 2 \cdot 3 = 6$ colours
- **From 2^{128} to 6 colours in 4 steps! How?**

Algorithm P3CBit:

Fast colour reduction

c_0 = my current colour as a k -bit string

c_1 = successor's colour as a k -bit string

i = **index of a bit that differs between c_0 and c_1**

b = **value of bit i in c_0**

$c = 2i + b$ = my new colour

$i \in \{0, \dots, k-1\}, \quad b \in \{0, 1\}, \quad c \in \{0, \dots, 2k-1\}$

Algorithm P3CBit:

Fast colour reduction

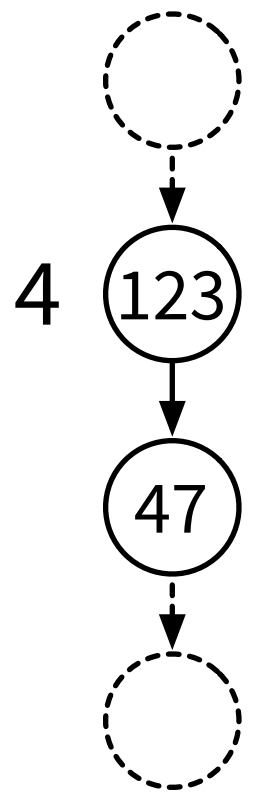
$c_0 = 123 = 01111\mathbf{0}11_2$ (my colour)

$c_1 = 47 = 00101\mathbf{1}11_2$ (successor's colour)

$i = 2$ (bits numbered 0, 1, 2, ... from right)

$b = 0$ (in my colour bit number i was 0)

$c = 2 \cdot 2 + 0 = 4$ (my new colour)



$k = 8$, reducing from $2^8 = 256$ to $2 \cdot 8 = 16$ colours

Algorithm P3CBit:

Fast colour reduction

$c_0 = 123 = 01111\mathbf{0}11_2$ (my colour)

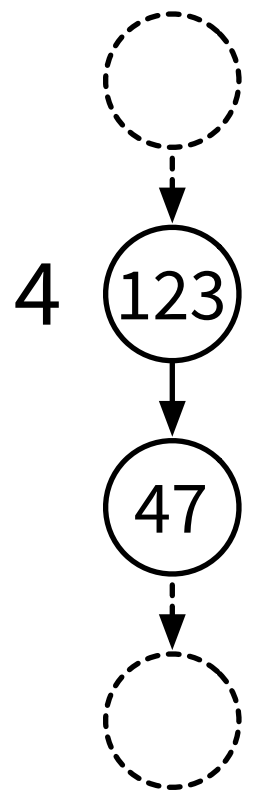
$c_1 = 47 = 0010\mathbf{1}1\mathbf{1}1_2$ (successor's colour)

Successor will pick one of these colours:

$14+\mathbf{0}$, $12+\mathbf{0}$, $10+\mathbf{1}$, $8+\mathbf{0}$, $6+\mathbf{1}$, $4+\mathbf{1}$, $2+\mathbf{1}$, $0+\mathbf{1}$

None of these conflict with my choice:

$4+\mathbf{0}$



Algorithm P3CBit:

Fast colour reduction

i = index of a bit that differs between c_0 and c_1

b = value of bit i in c_0

$c = 2i + b$ = my new colour

Successor picks different $i \rightarrow$ different c

Successor picks same $i \rightarrow$ different $b \rightarrow$ different c

My new colour \neq my successor's new colour

Performance

- **P3C: worst case $O(n)$**
 - assuming unique IDs
- **P3CRand: $O(\log n)$ with high probability**
- **P3CBit: $O(\log^* n)$**
 - assuming unique IDs are polynomial in n

Performance

- **P3CBit:** $O(\log^* n)$
 - assuming unique IDs are polynomial in n
- **Next week:** **this is optimal!**
 - no deterministic distributed algorithm can 3-colour a path in time $o(\log^* n)$