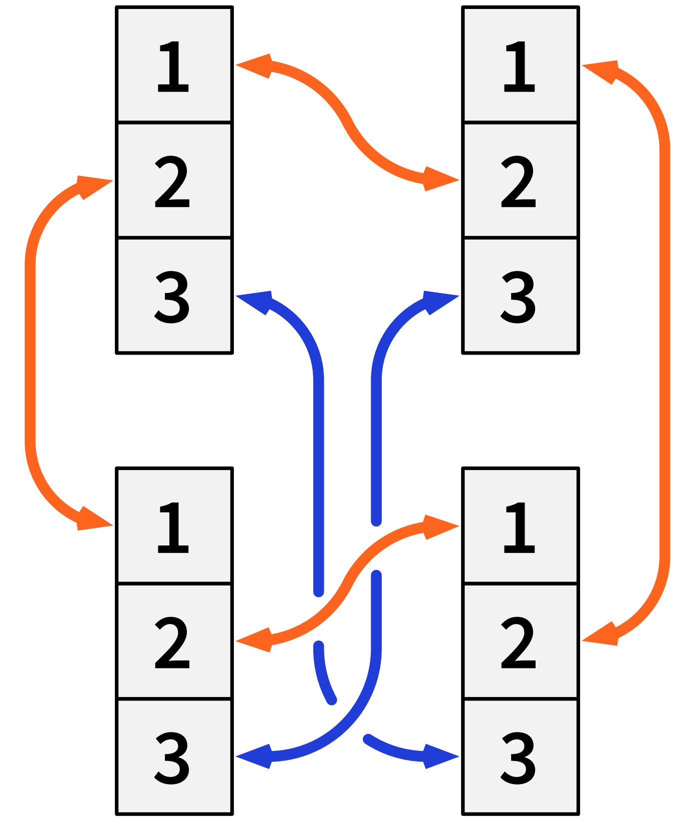
ICS-E5020
Distributed Algorithms

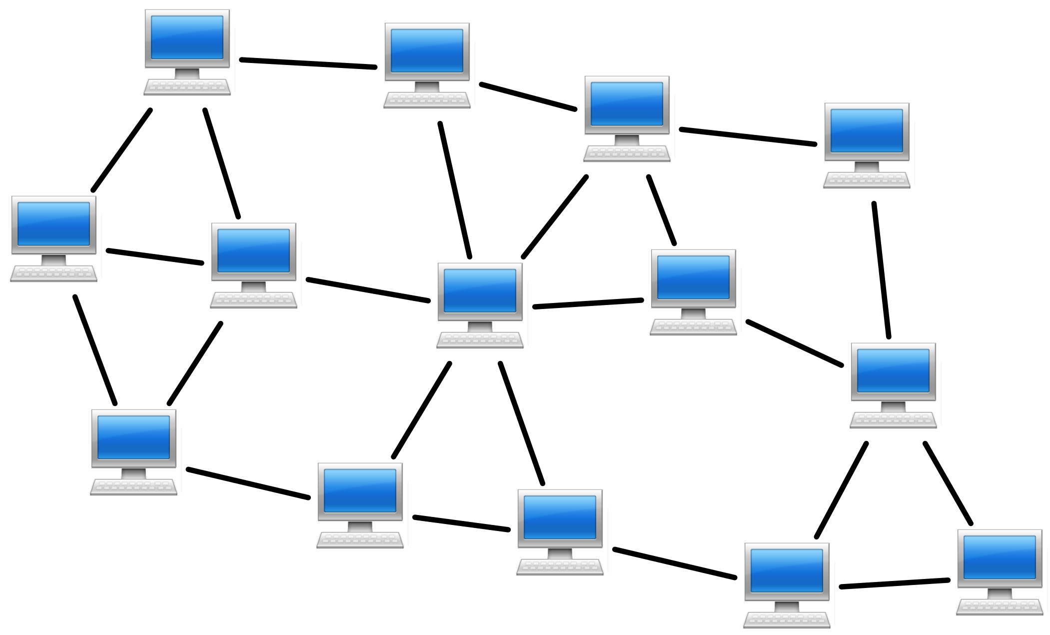
Jukka Suomela
Aalto University
Autumn 2015

iki.fi/suo/da-2015



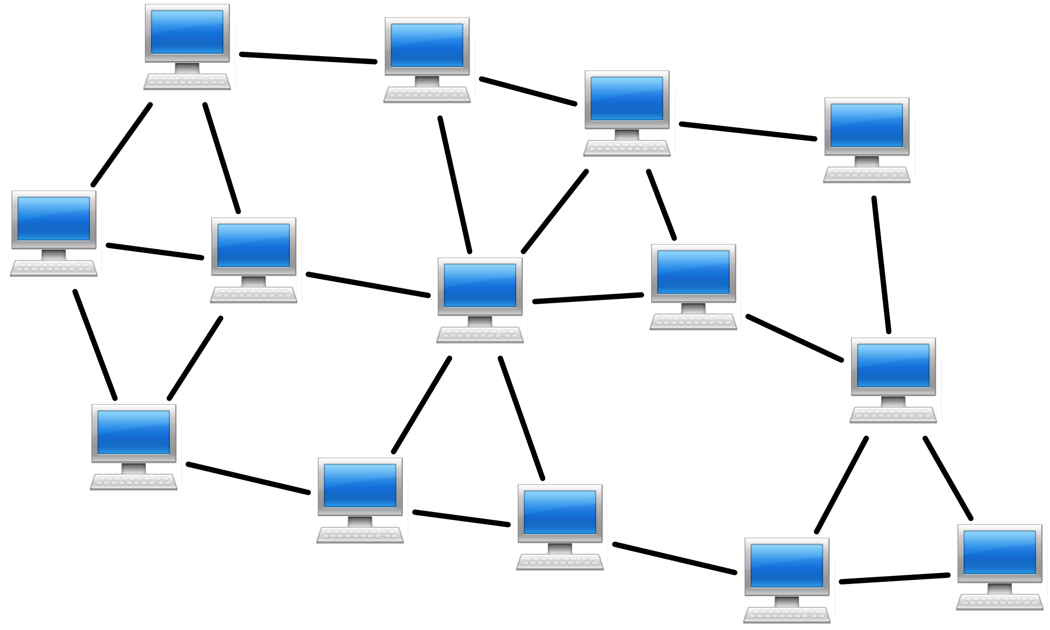
Distributed Algorithms

Algorithms for computer networks



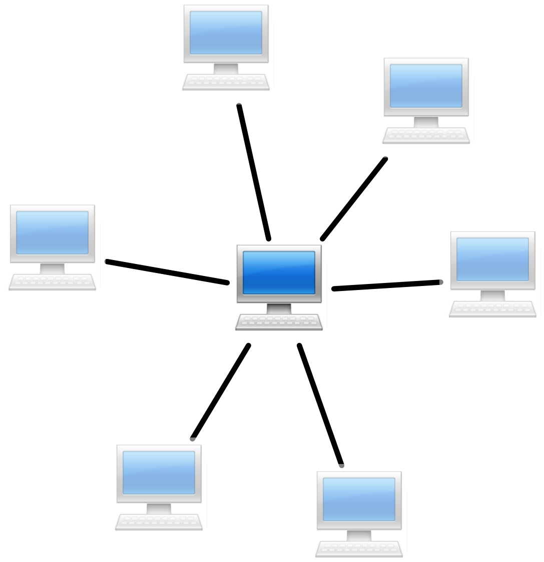
Distributed Algorithms

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



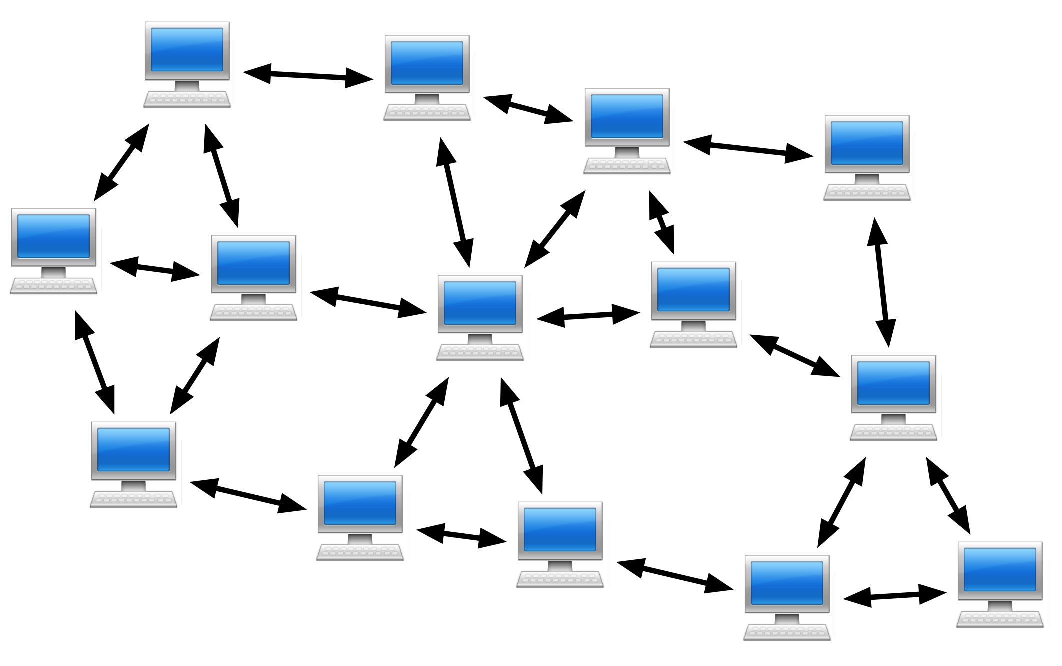
Distributed Algorithms

Initially each computer only aware of its immediate neighbourhood



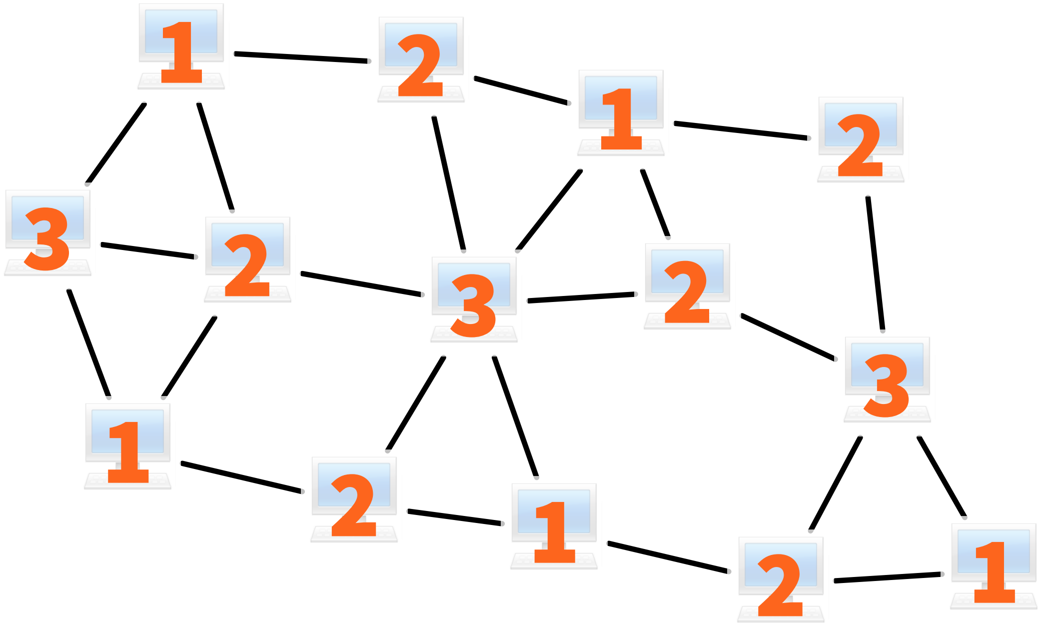
Distributed Algorithms

Nodes can exchange messages with their neighbours to learn more…



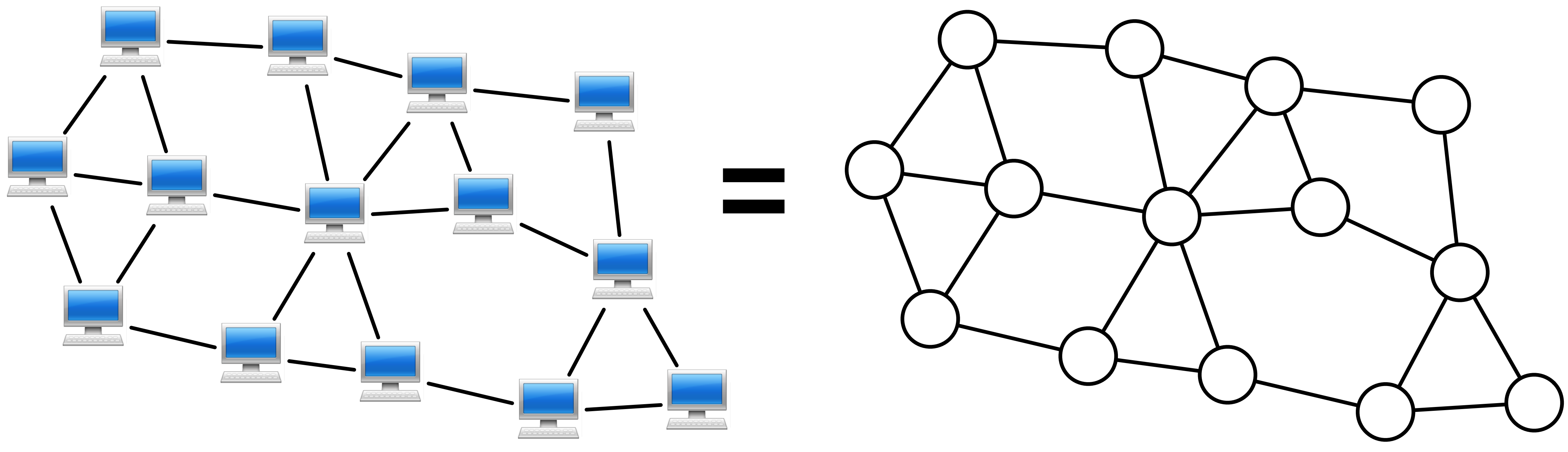
Distributed Algorithms

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



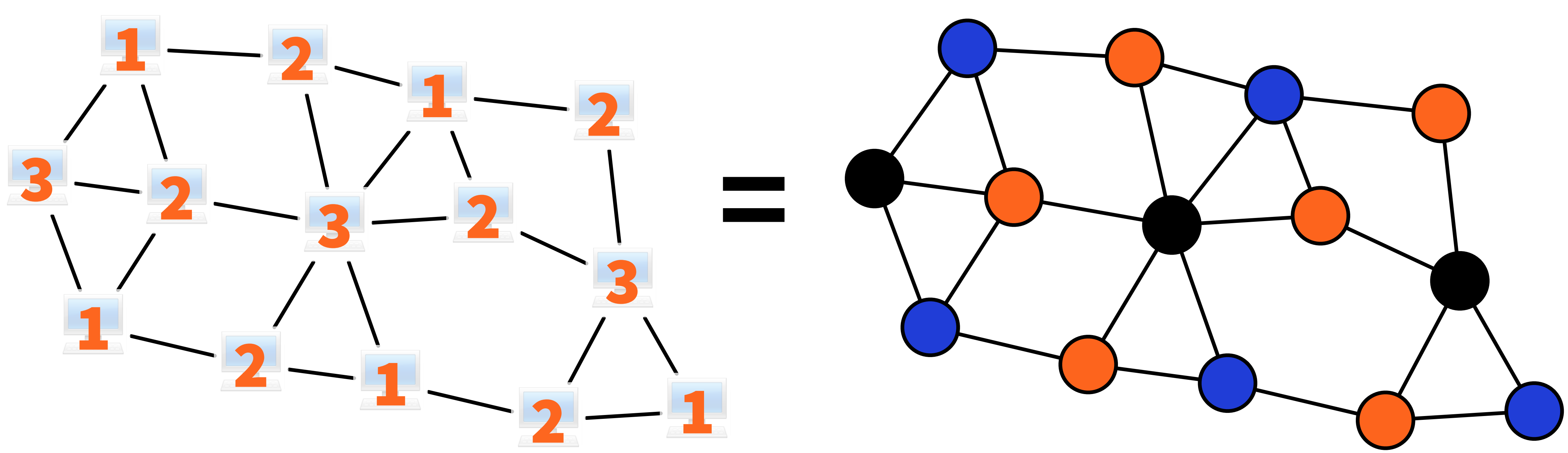
Distributed Algorithms

Focus on graph problems:
network topology = input graph



Distributed Algorithms

Focus on graph problems:
local outputs = solution (here: graph colouring)

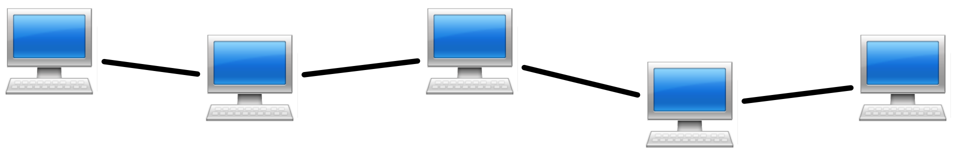


Distributed Algorithms

Typical research question:

“How fast can we solve graph problem X?”

Time = number of communication rounds

- **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 1

– Warm-up: positive results

Running example:
3-colouring a path

Given a path:

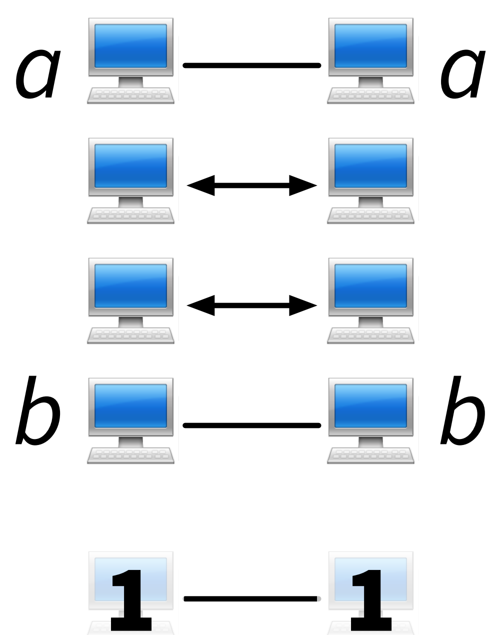
Output a proper 3-colouring, e.g.:

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**



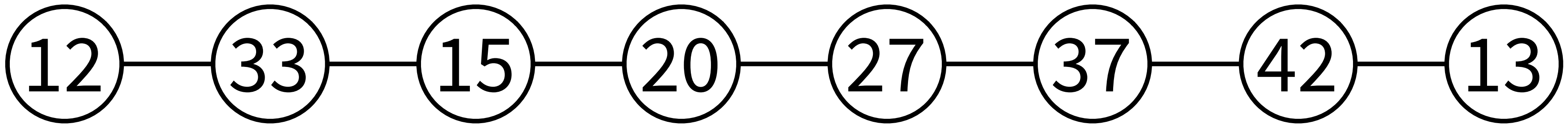
same initial state
same messages sent
same messages received
same new state
…
same output

Challenge: **Symmetry breaking**

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

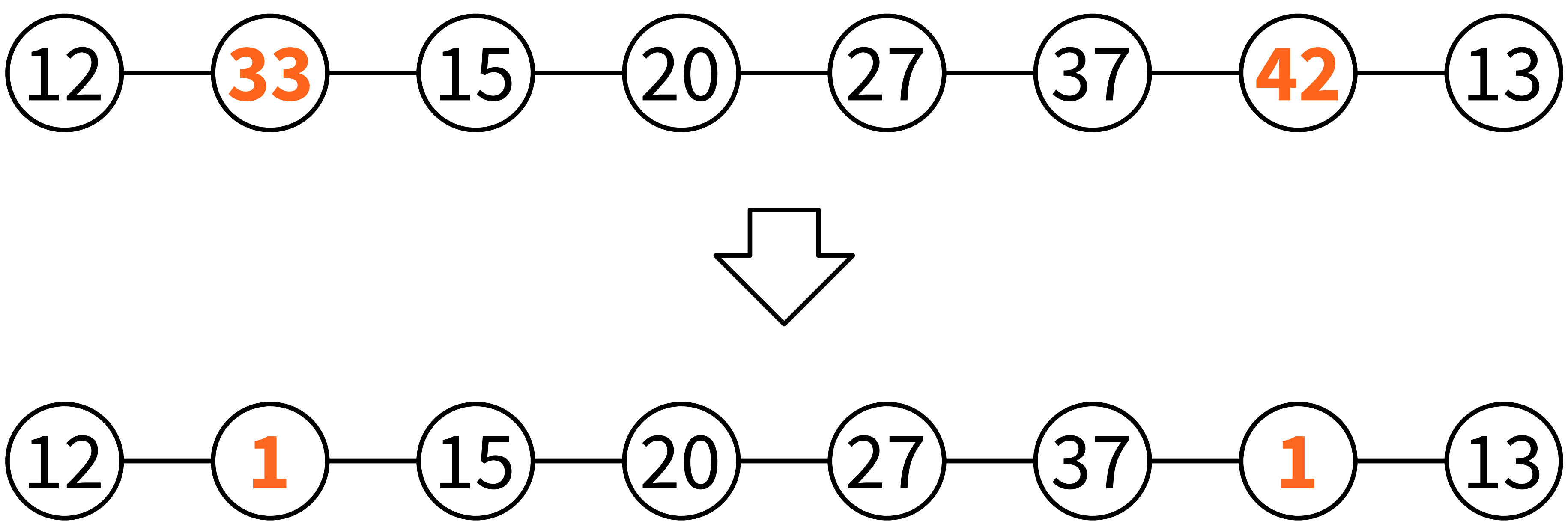
Algorithm P3C:
Using unique IDs

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



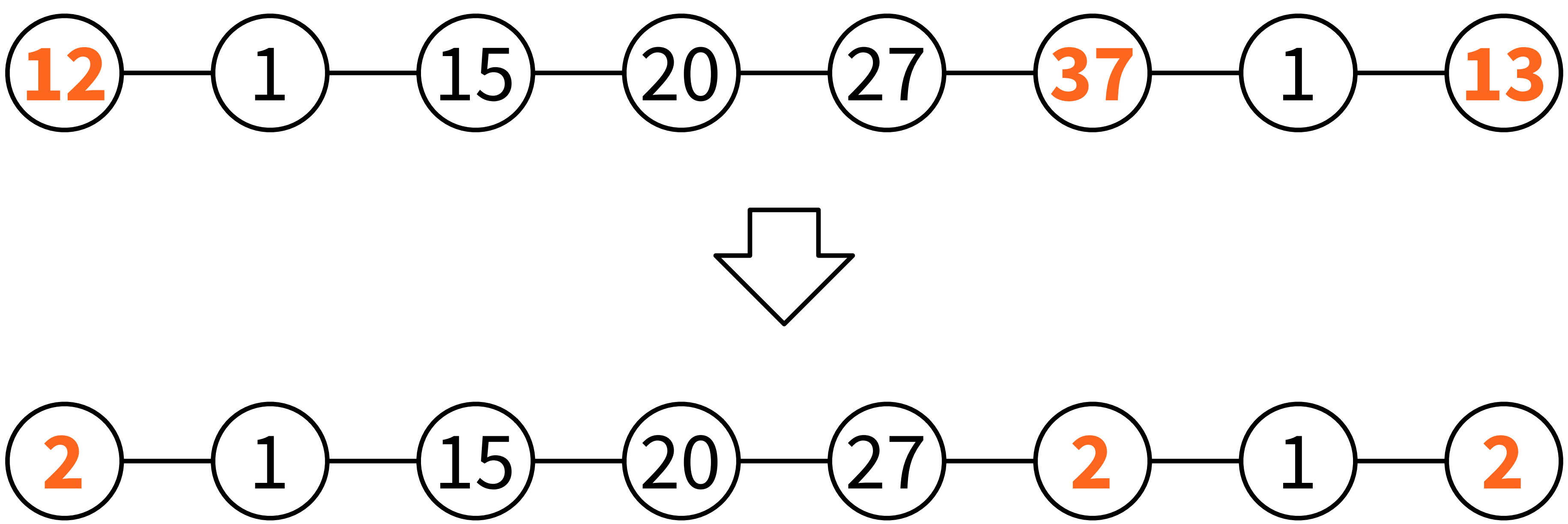
Algorithm P3C: **Using unique IDs**

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



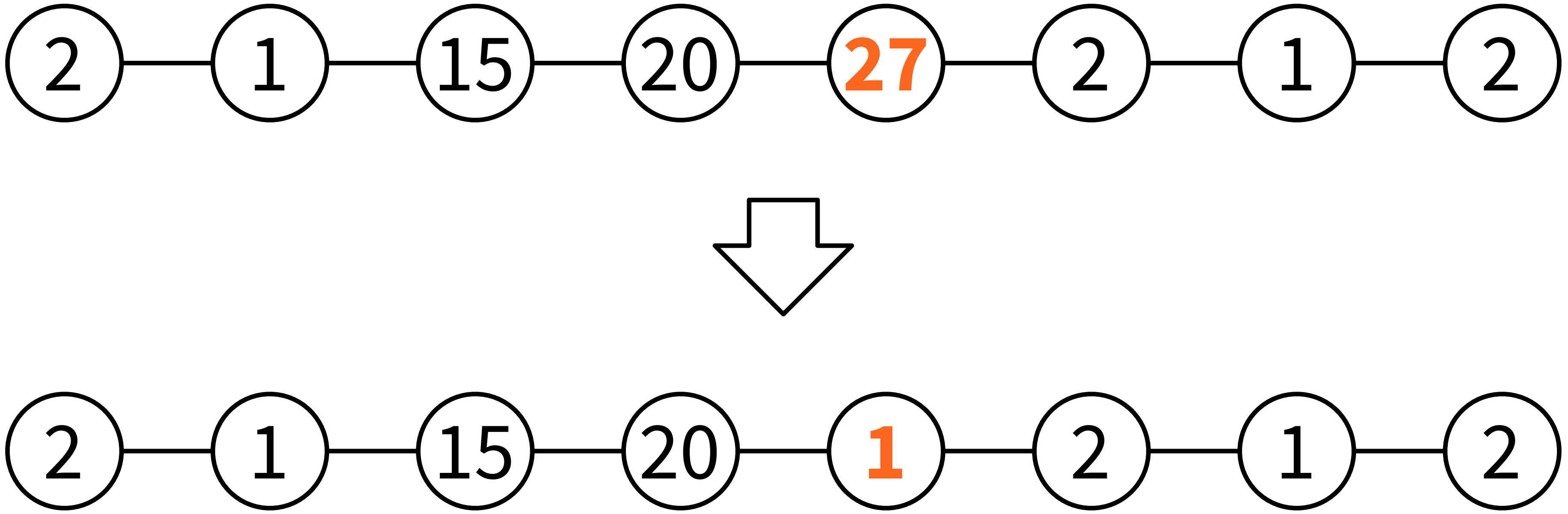
Algorithm P3C:
Using unique IDs

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



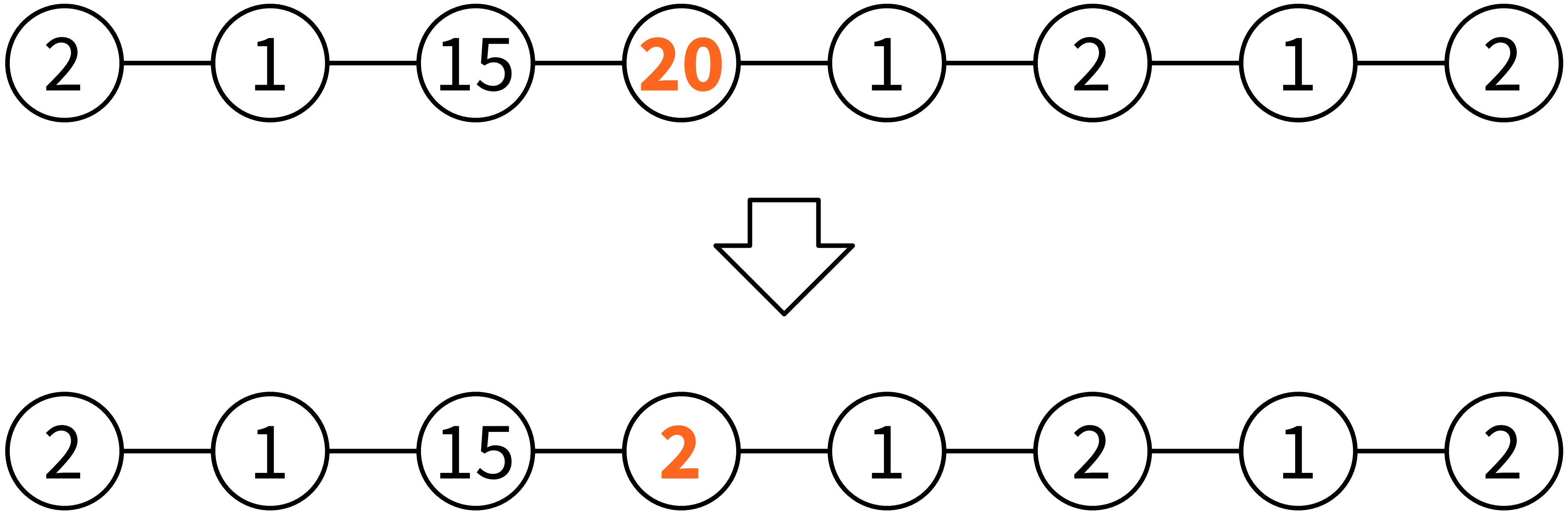
Algorithm P3C: **Using unique IDs**

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



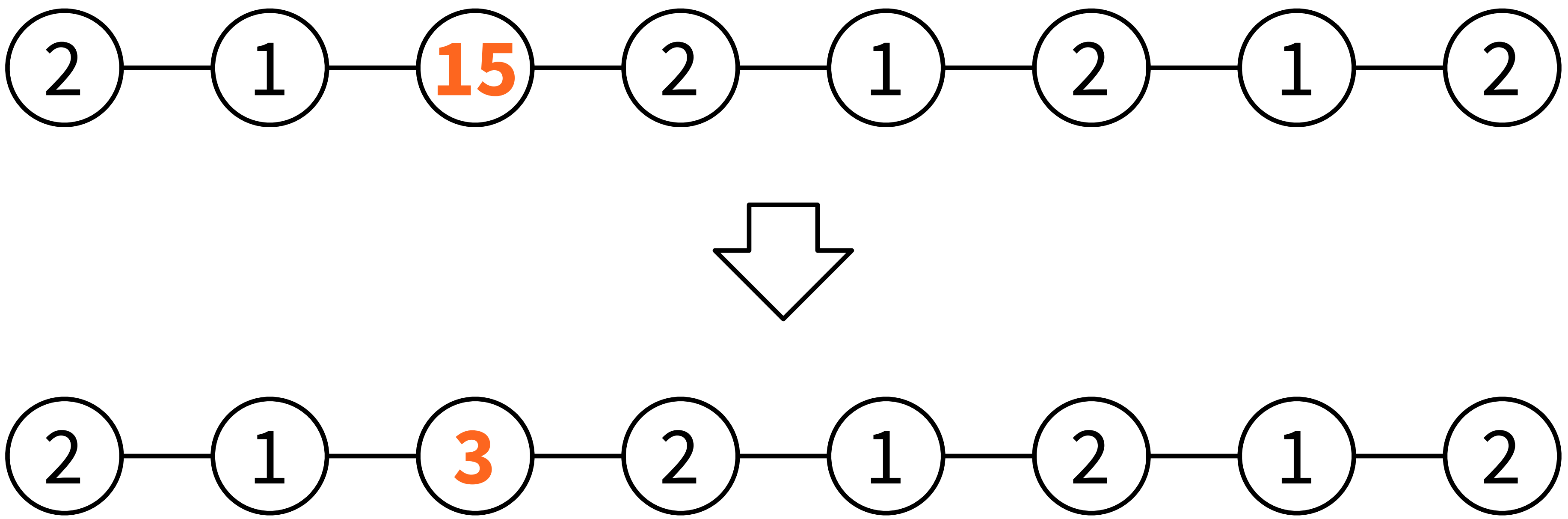
Algorithm P3C:
Using unique IDs

- **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
  - $2^4 \rightarrow 2 \cdot 3 = 6$ 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

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

Algorithm P3CBit: **Fast colour reduction**

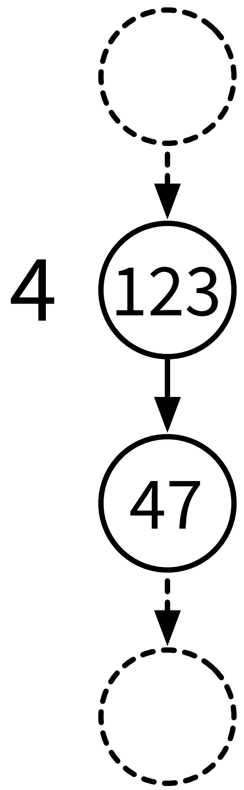
$\boldsymbol{c_0} = \mathbf{123} = \mathbf{01111011_2}$ (my colour)

$\boldsymbol{c_1} = \mathbf{47} = \mathbf{00101111_2}$ (successor's colour)

$\boldsymbol{i} = \mathbf{2}$ (bits numbered 0, 1, 2, … from right)

$\boldsymbol{b} = \mathbf{0}$ (in my colour bit number *i* was 0)

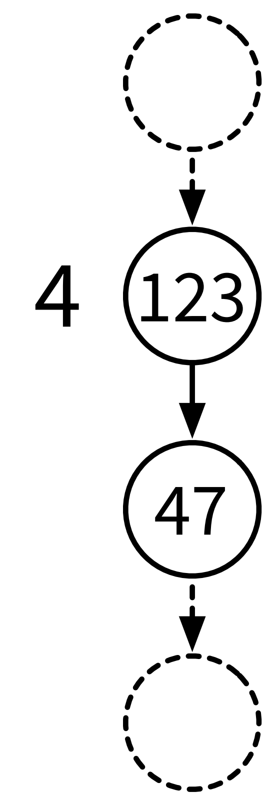
$\boldsymbol{c} = \mathbf{2 \cdot 2 + 0 = 4}$ (my new colour)



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

Algorithm P3CBit:
Fast colour reduction

c_0 = **123 = 01111011**$_2$ (my colour)
c_1 = **47 = 00101111**$_2$ (successor's colour)



Successor will pick one of these colours:
14+**0**, 12+**0**, 10+**1**, 8+**0**, 6+**1**, 4+**1**, 2+**1**, 0+**1**

None of these conflict with my choice:
4+**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 → different c
Successor picks same i → different b → different c

My new colour ≠ my successor's new colour

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

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

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 $o(\log^* n)$