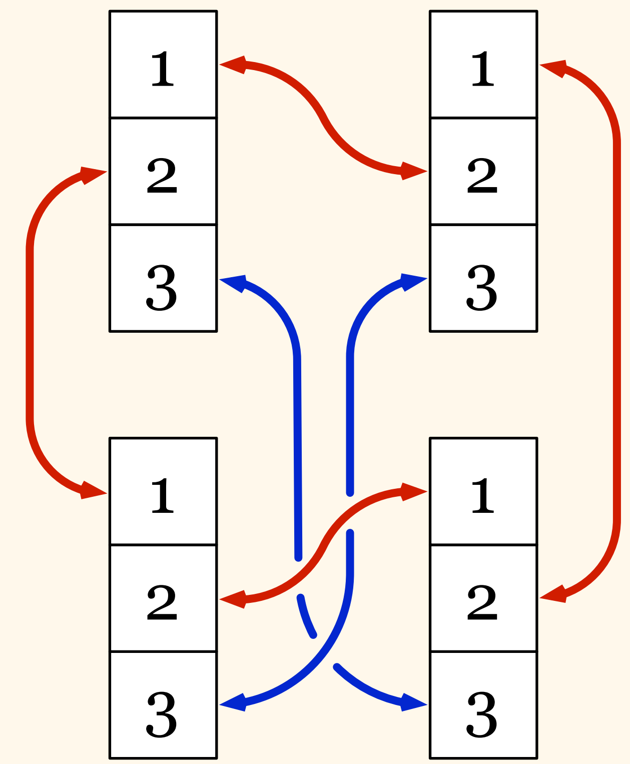
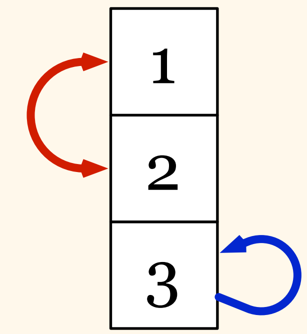
Deterministic Distributed Algorithms

www.iki.fi/suo/dda



Jukka Suomela

University of Helsinki,
March–April 2012



Introduction

DDA Course

Lecture 1.1

13 March 2012

Practicalities

- Read the course web page: www.iki.fi/suo/dda
- Pay attention to:
  - *course content* — theory, not practice
  - *course format* — not a typical lecture course
  - *course tracking system* — use it!
  - *online support* — two online forums

Course Content

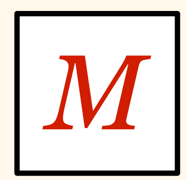
- Fundamental questions:
    - what can be computed?
    - what can be computed fast?
- Model of computation:
    - distributed systems

Traditional Perspective

Programmer:	$\boxed{M}$	constructs a machine
Adversary:	$x \rightarrow \boxed{M}$	chooses any valid input
Machine:	$x \rightarrow \boxed{M} \; y \rightarrow$	does computation, prints a valid output

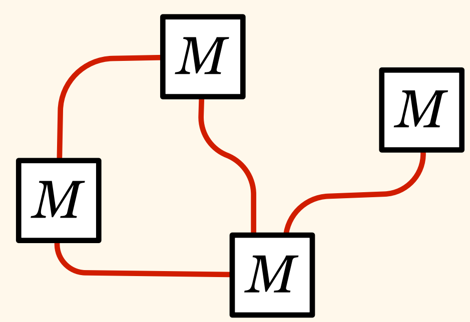
Distributed Algorithms

Programmer:



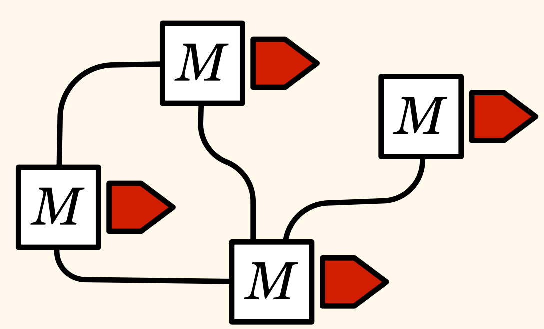
constructs
a machine

Adversary:



constructs
a *network*

Network:



does *communication*,
prints a valid output

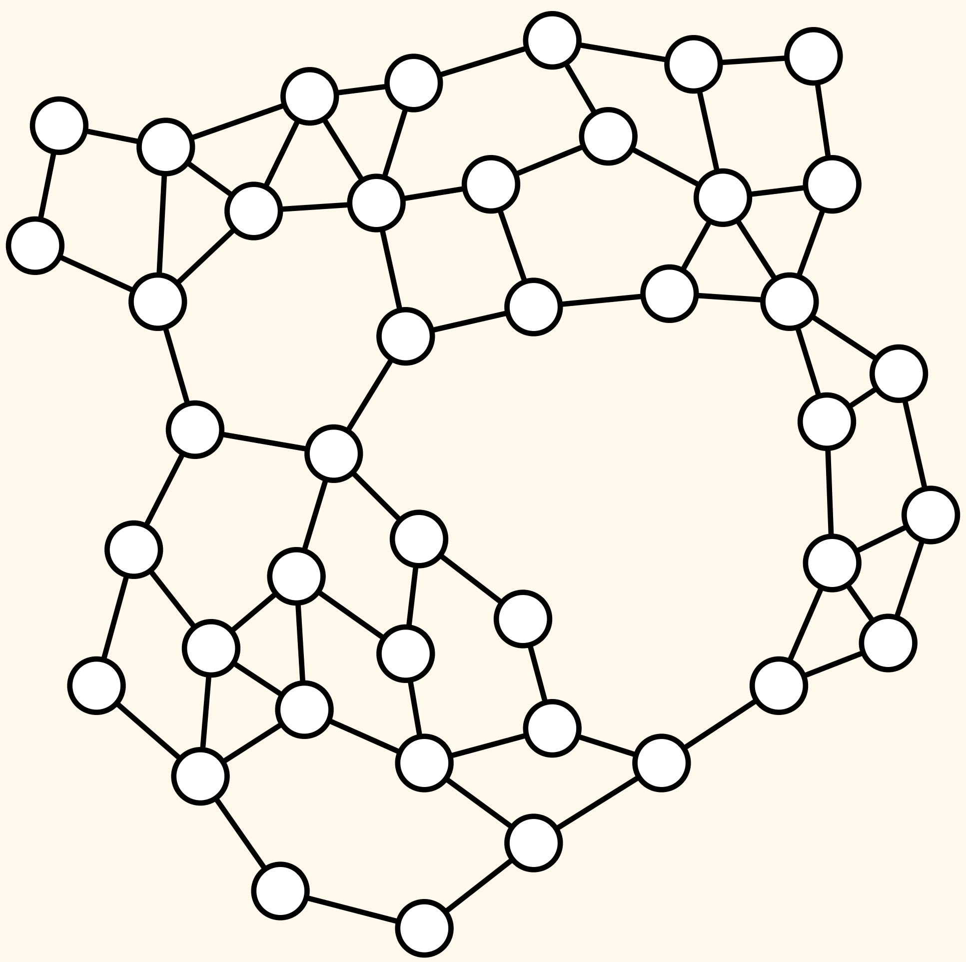
You Will Learn...

- A new mindset: how to reason about distributed and parallel systems
  - not a bad skill in the multi-core era
- Combinatorial optimisation
- Some math that has plenty of applications in computer science
  - graph theory, Ramsey theory, ...

Plan: Two Models

- Week 1: some graph theory
- Weeks 2–4: *“port-numbering model”*
  - weeks 2 and 4: positive results,
    week 3: negative results
- Weeks 5–6: *“unique identifiers”*
  - week 5: positive results,
    week 6: negative results

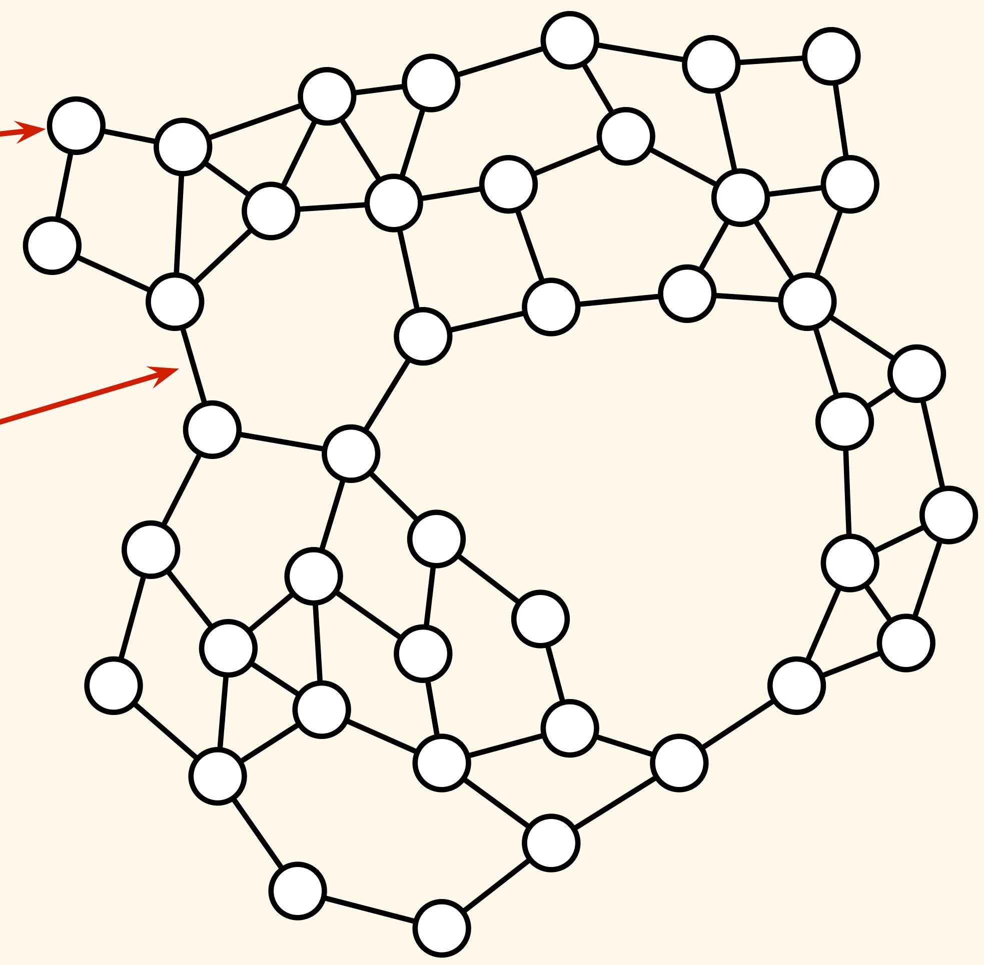
Graphs



Graphs

node

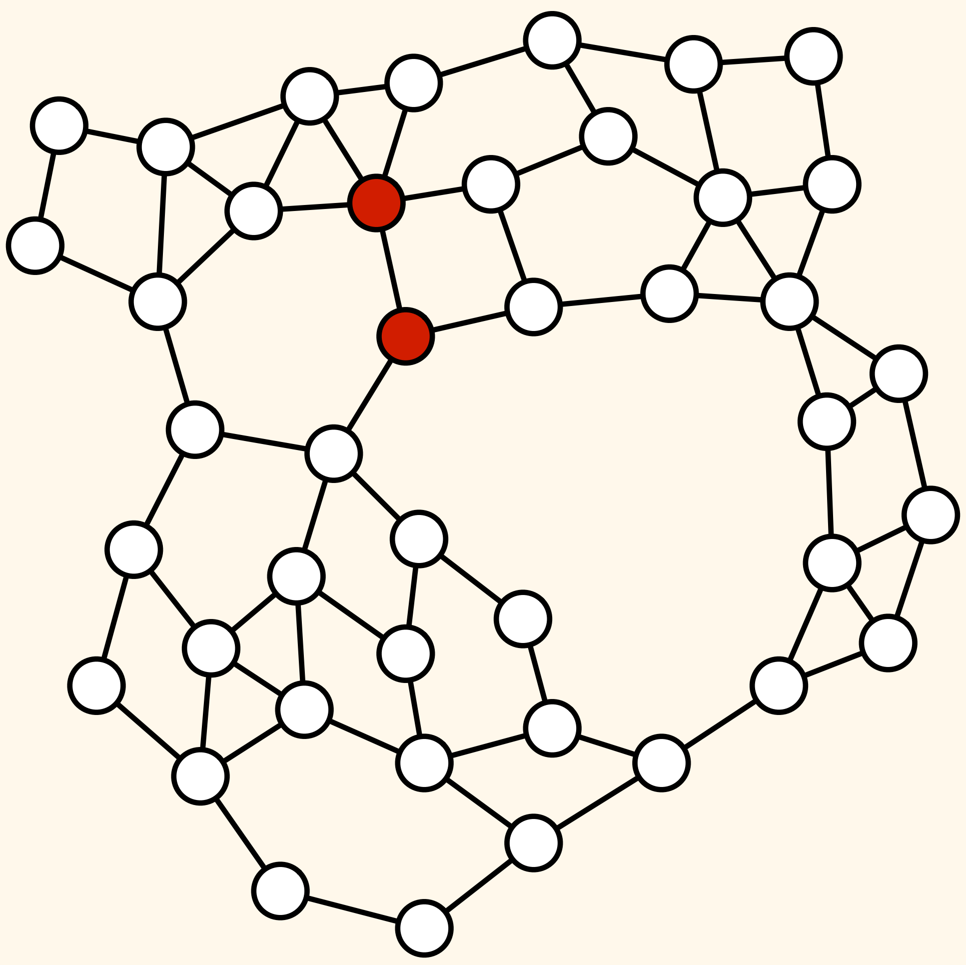
edge



Graphs

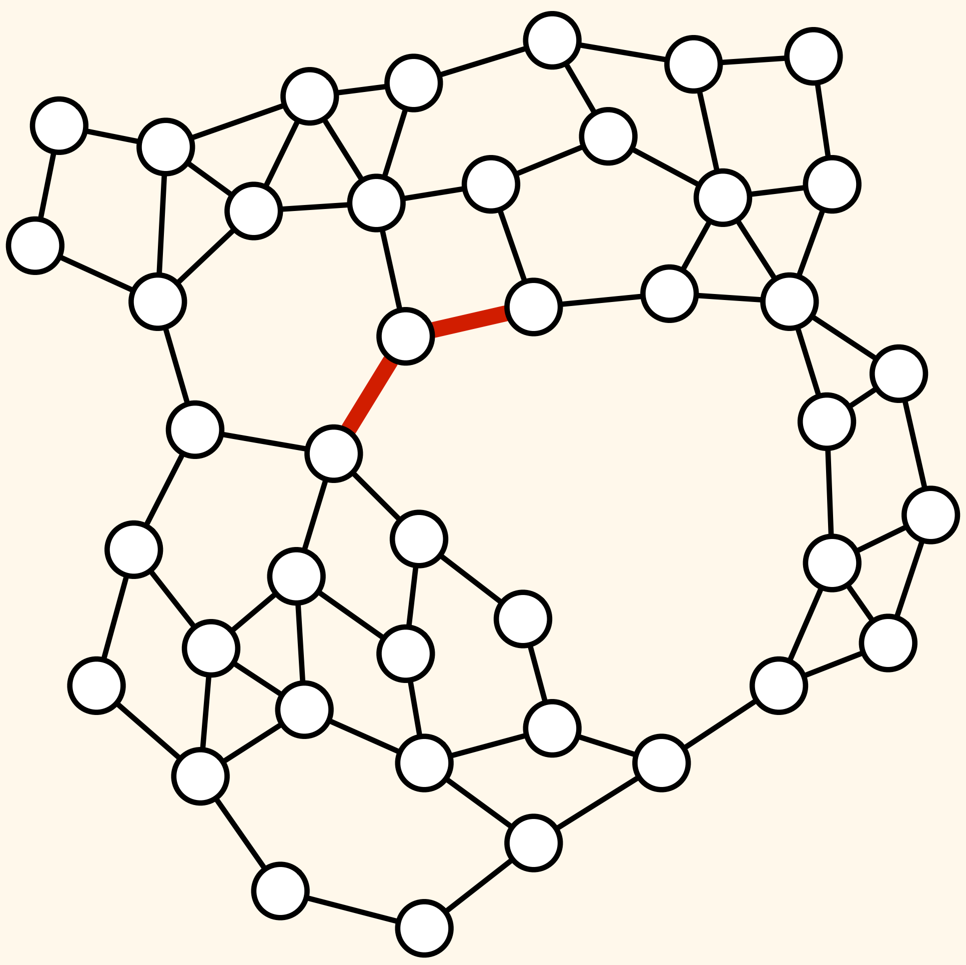
adjacent nodes

neighbours



Graphs

adjacent edges



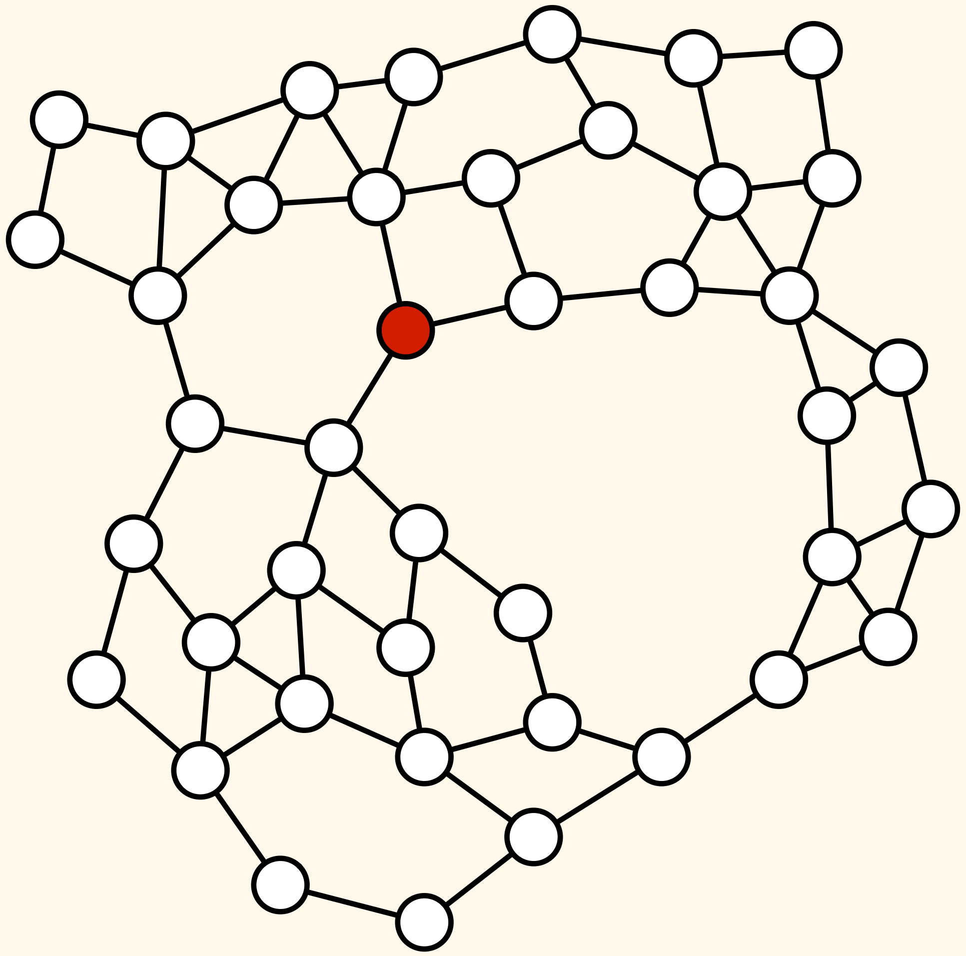
Graphs

node with 3 neighbours

adjacent to 3 nodes

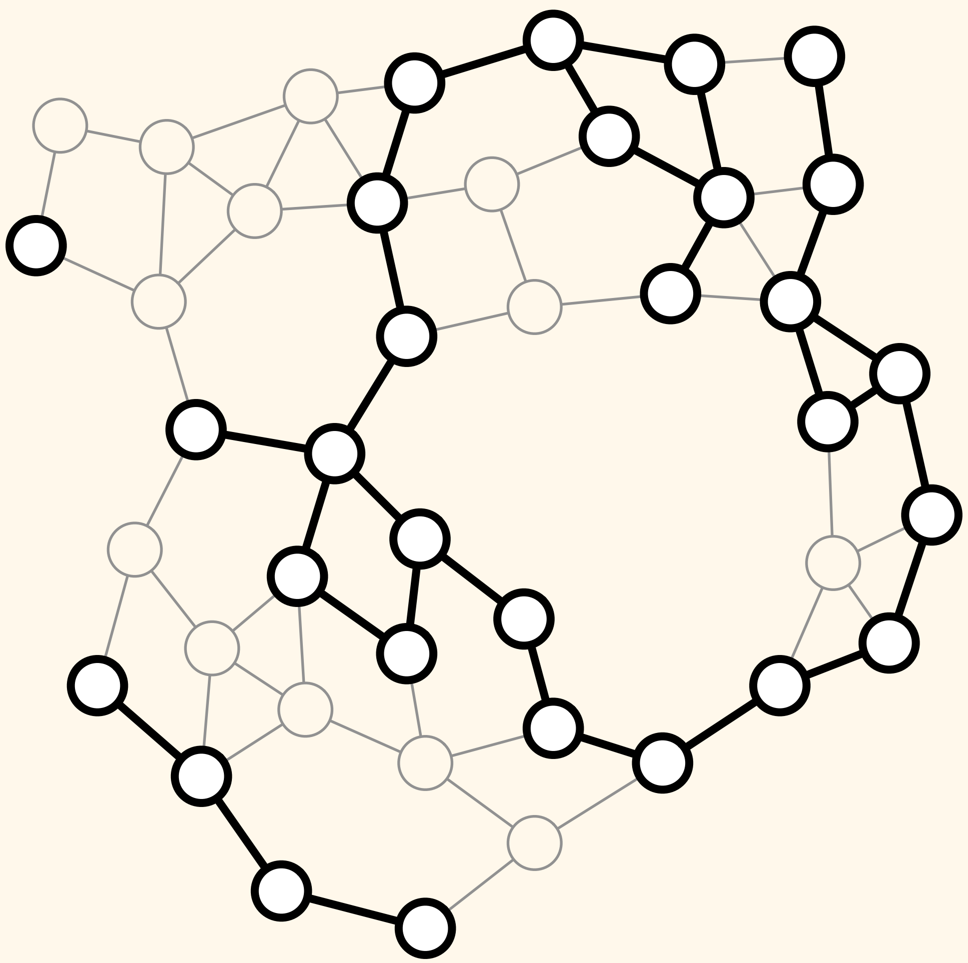
incident to 3 edges

degree is 3



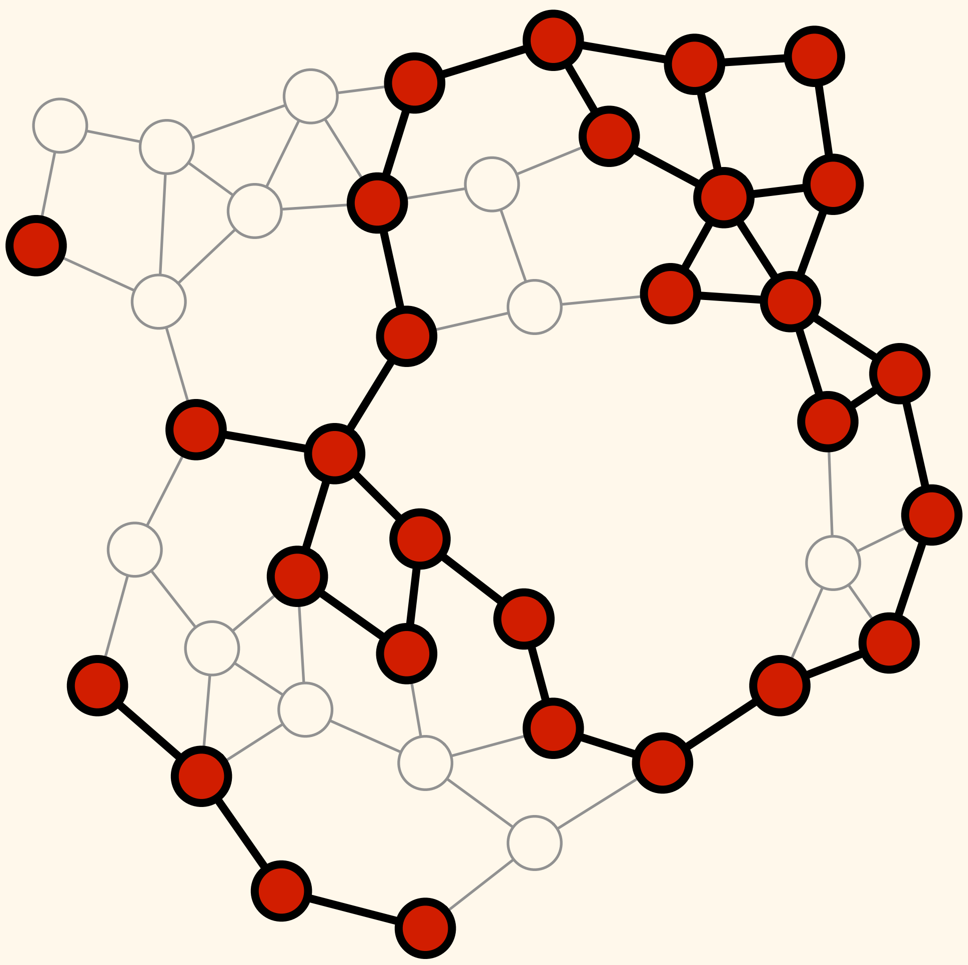
Graphs

subgraph



Graphs

subgraph induced
by the red nodes



all red nodes

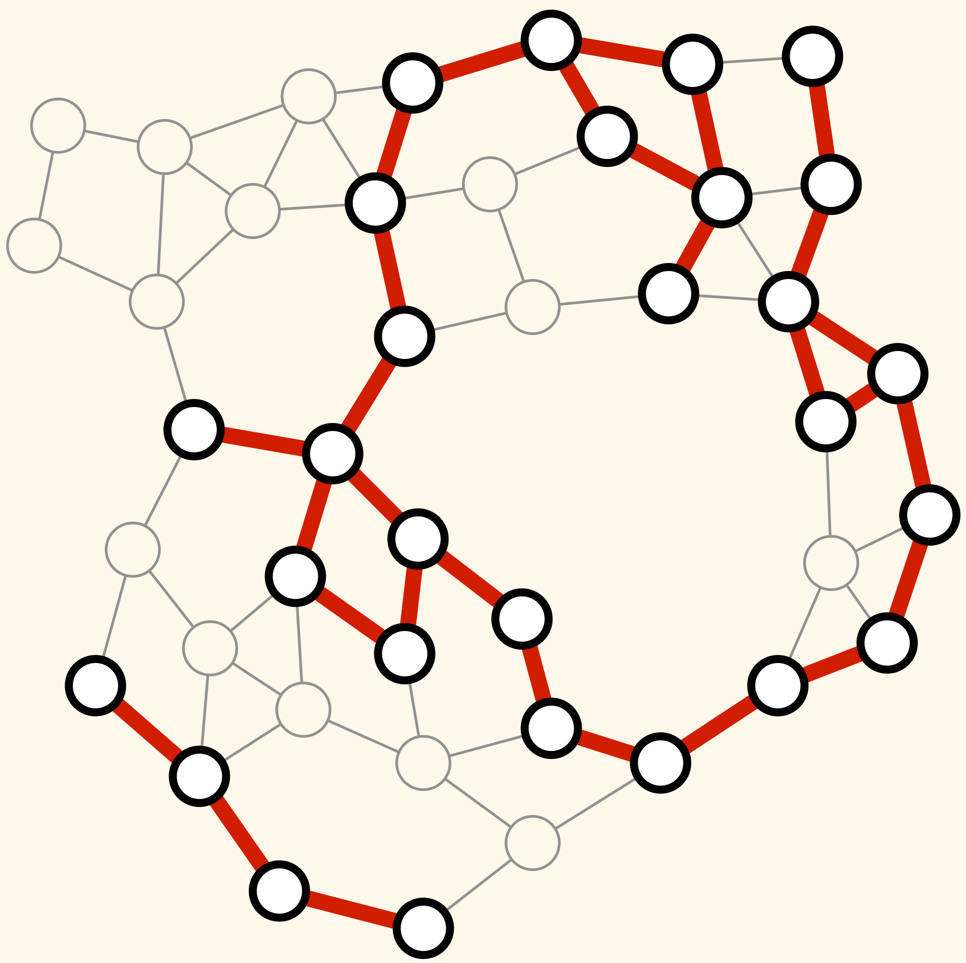
all edges that join
a pair of red nodes

Graphs

subgraph induced
by the red edges

all red edges

all nodes that are
incident to red edges

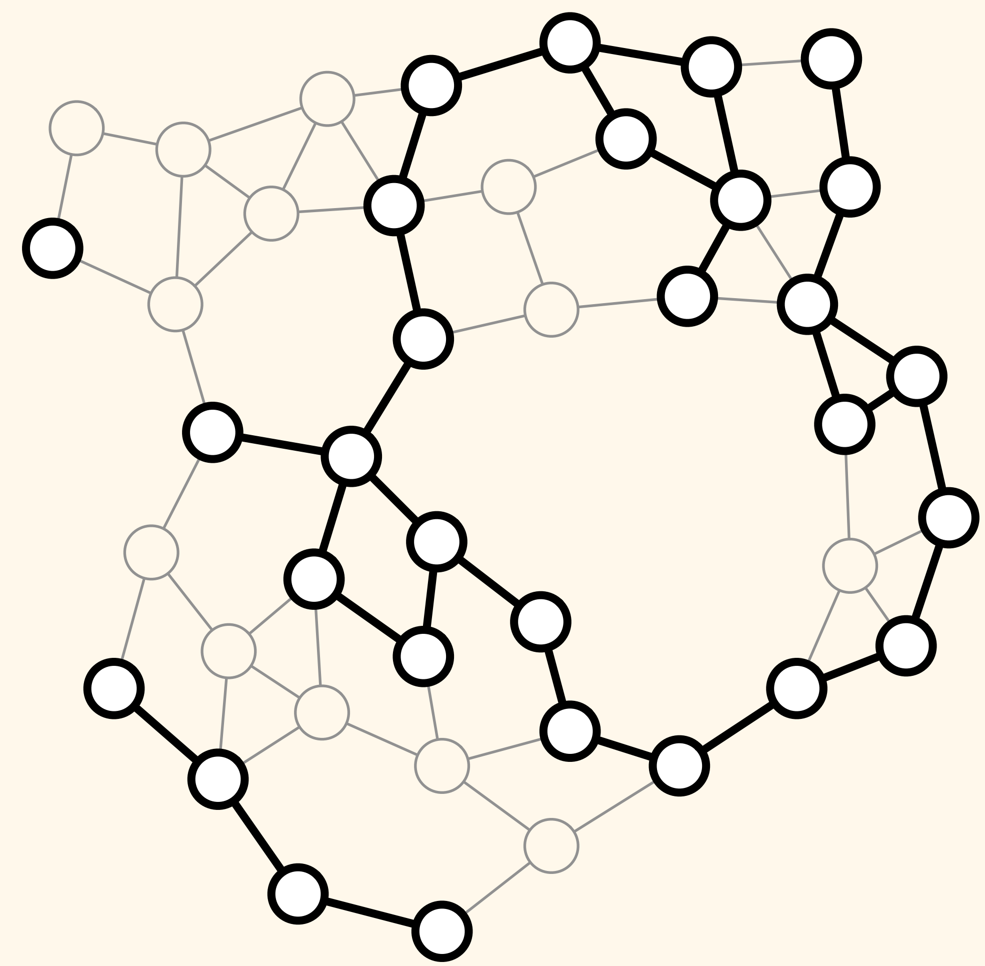


Graphs

not a node-induced subgraph

not an edge-induced subgraph

not a spanning subgraph



Graphs

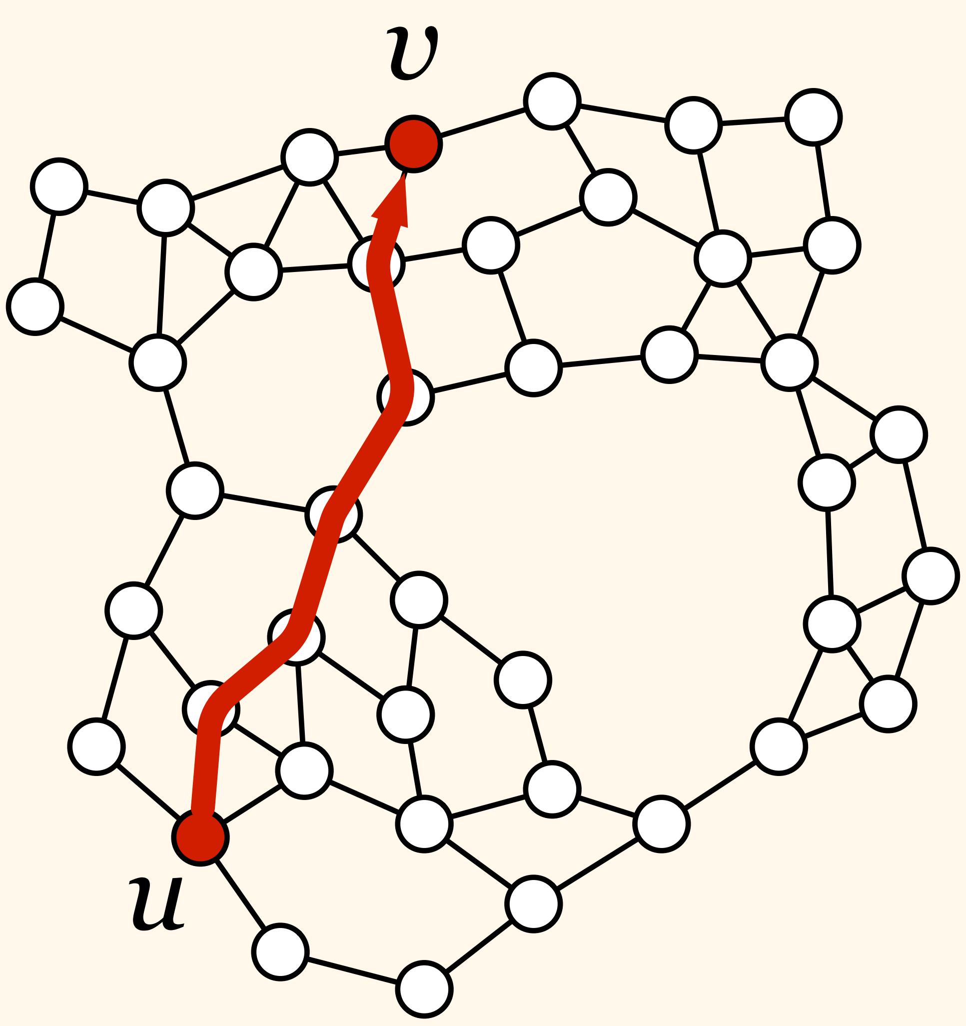
a shortest path from u to v

length 6

(six *edges*, seven *nodes*)

$\text{dist}(u, v) = 6$

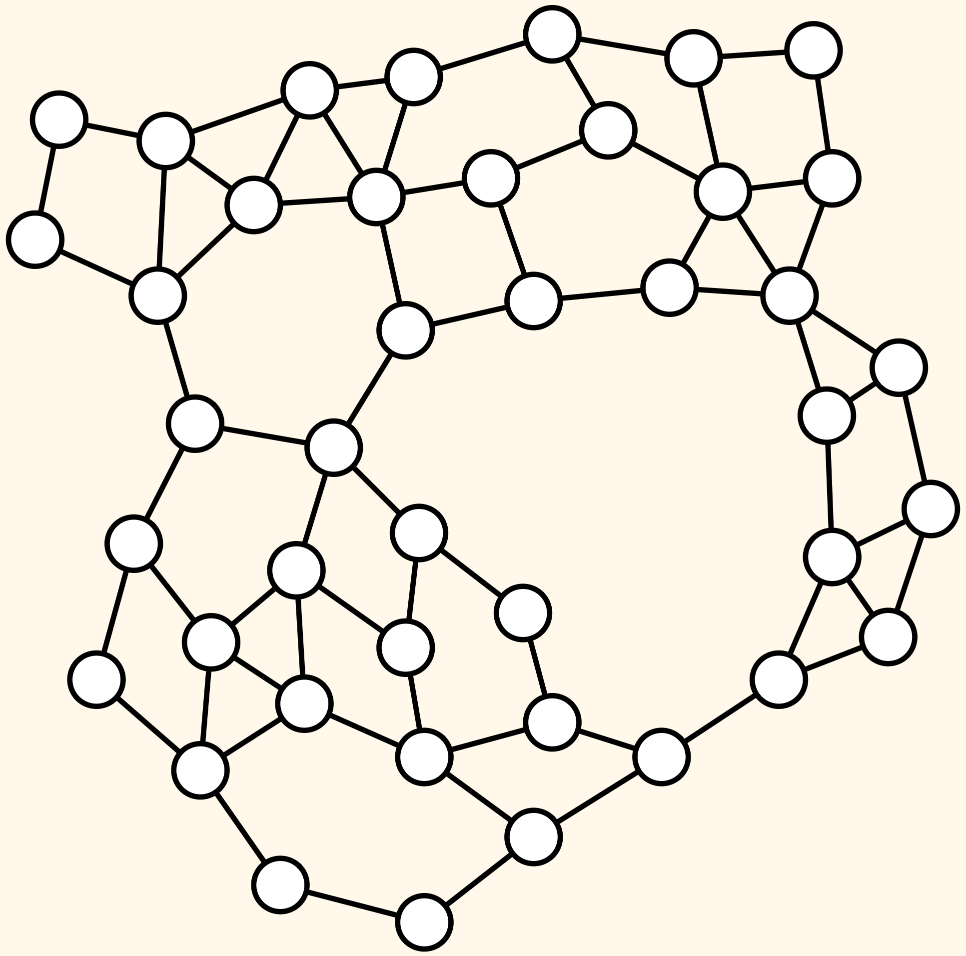
$\text{diameter} \geq 6$



Graphs

connected graph

one connected component

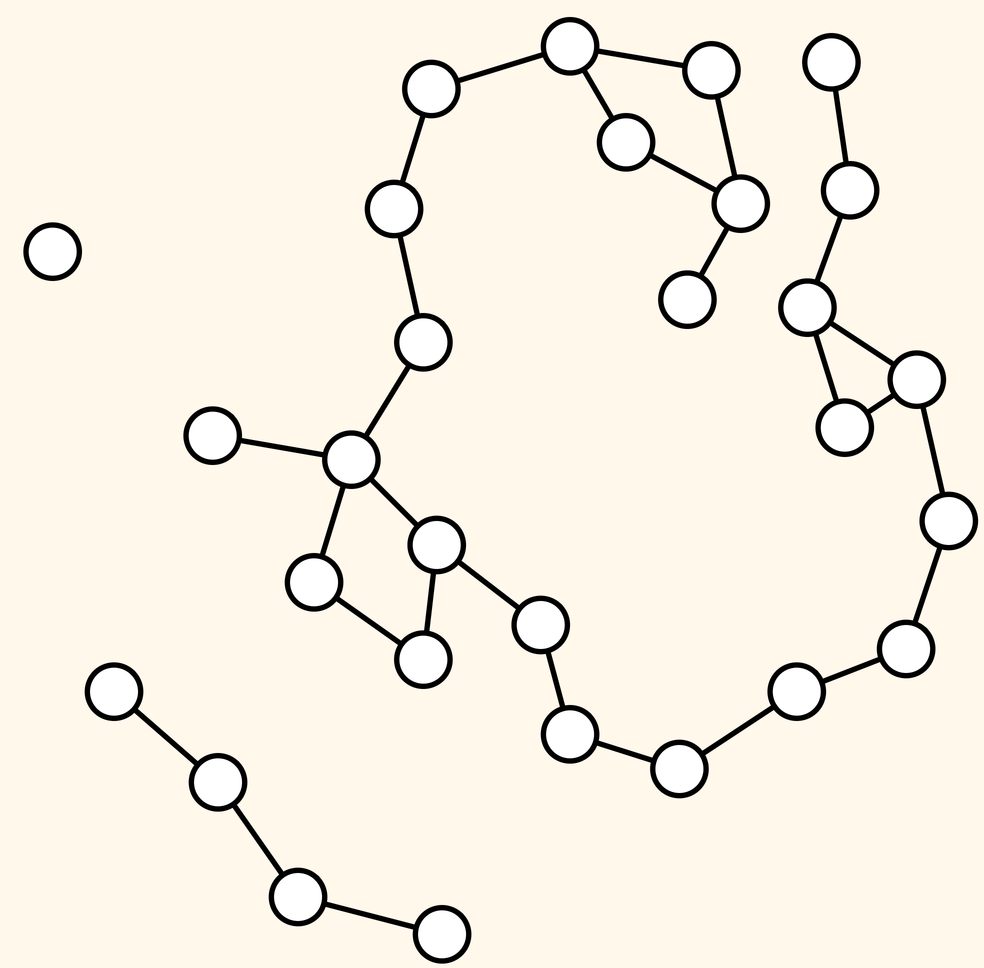


Graphs

not a connected graph

three connected components

one isolated node

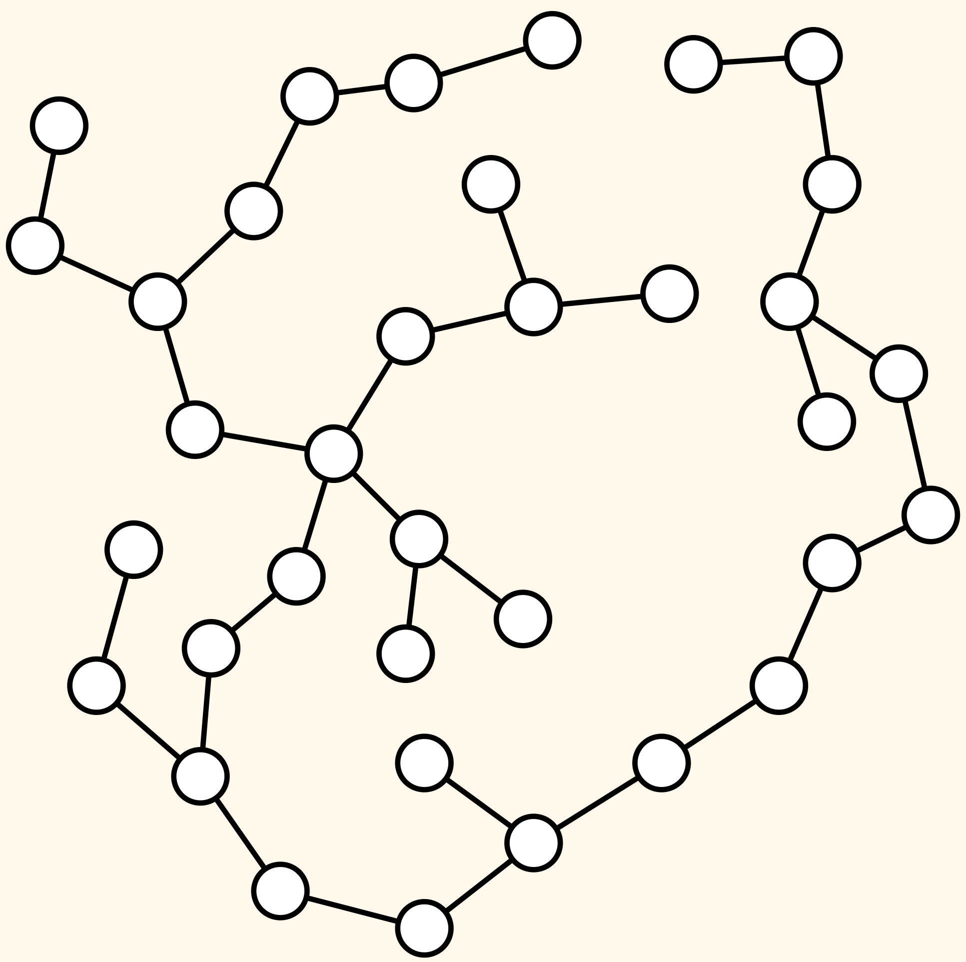


Graphs

tree

connected

no cycles

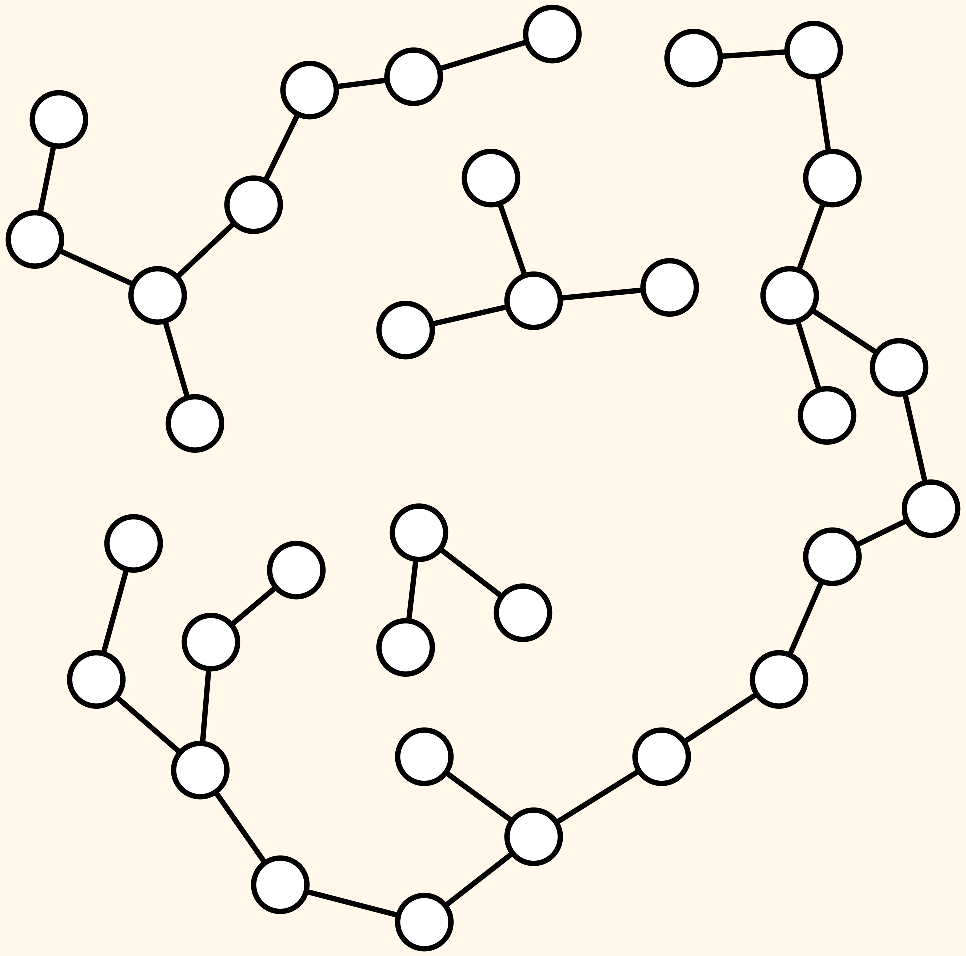


Graphs

forest

four connected components

no cycles

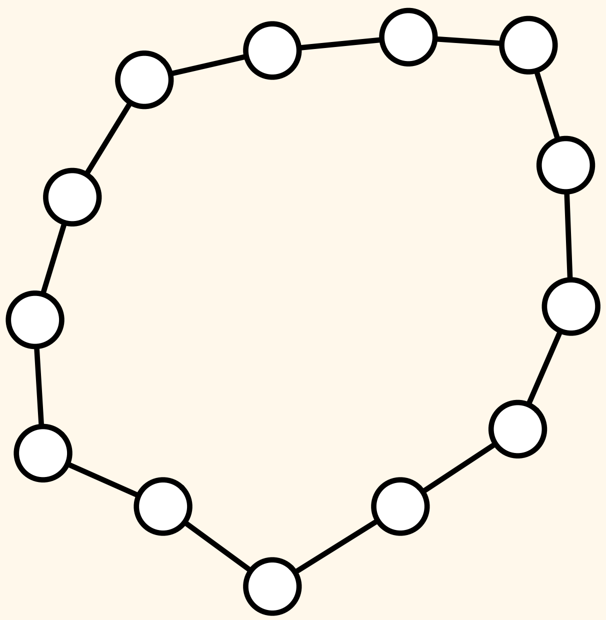


Graphs

cycle graph

connected

2-regular



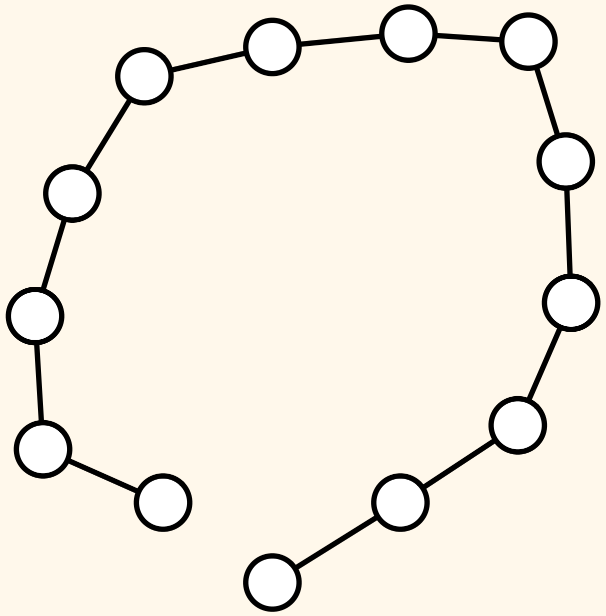
Graphs

path graph

tree

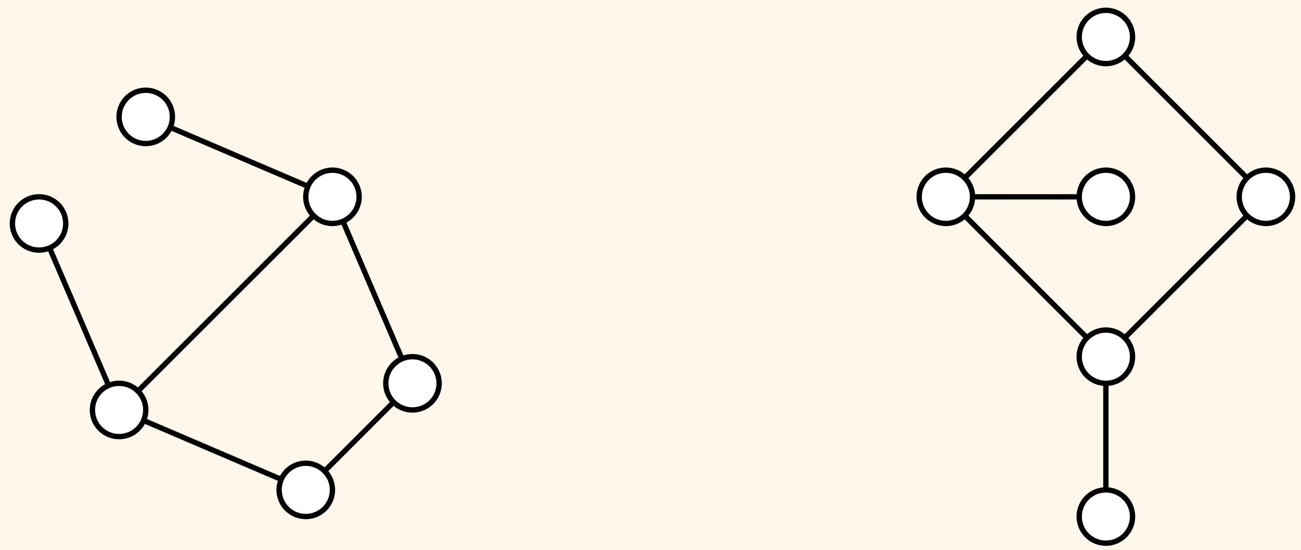
connected

maximum degree 2



Graphs

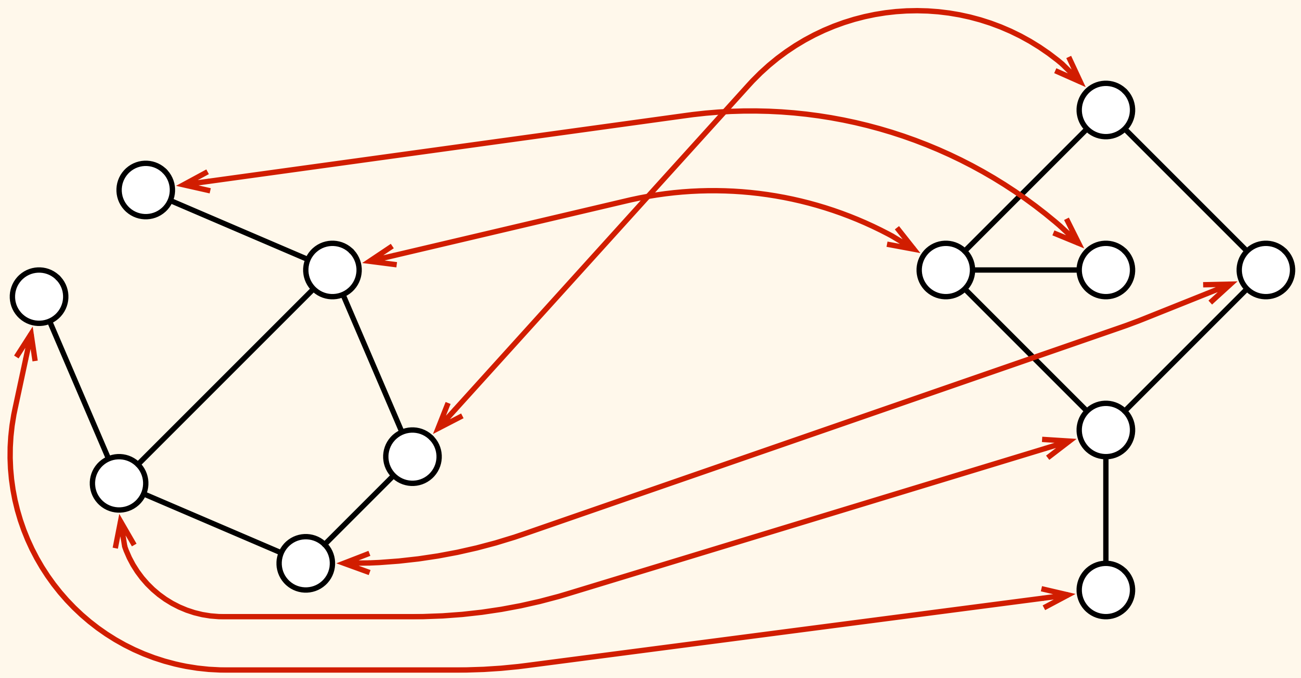
two isomorphic graphs



Graphs

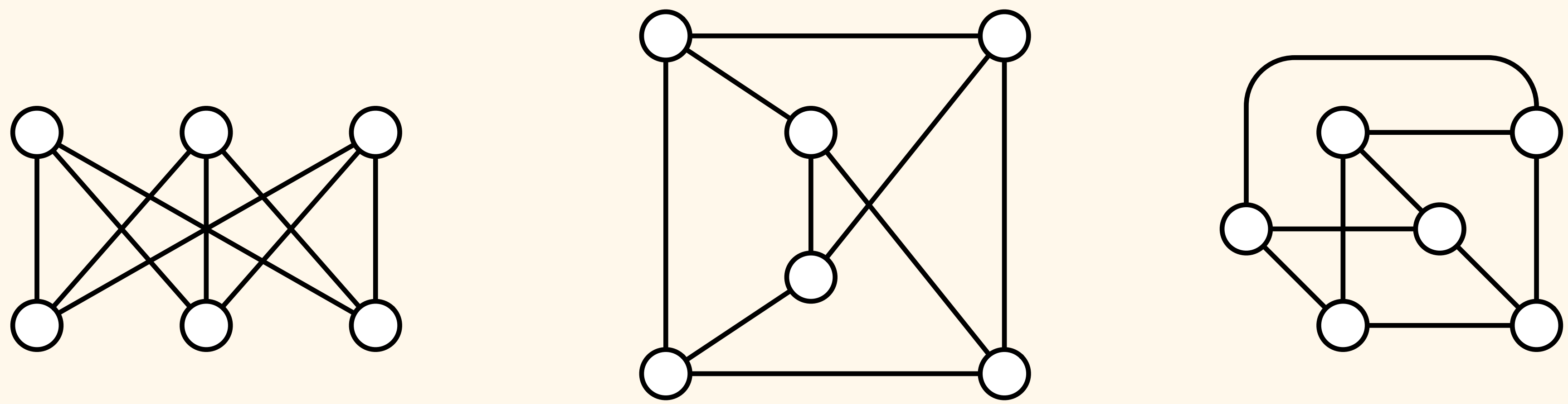
two isomorphic graphs

bijection that preserves the structure



Graphs

three isomorphic graphs



Graph Problems

- Recall the definitions:
    - independent set — vertex cover — dominating set
    - matching — edge cover — edge dominating set
    - vertex colouring — domatic partition
    - edge colouring — edge domatic partition
- Examples in the course material…

Optimisation

- Maximisation problems:
  - *maximal* = cannot add anything
  - *maximum* = largest possible size
  - α-approximation = at least $1/\alpha$ times maximum
- Example: independent set
  - maximal is trivial to find greedily,
    maximum may be very difficult to find

Optimisation

- Minimisation problems:
  - *minimal* = cannot remove anything
  - *minimum* = smallest possible size
  - α-approximation = at most α times minimum
- Example: vertex cover
  - minimal is trivial to find greedily,
    minimum may be very difficult to find

Optimisation

Terminology:

"α-approximation of minimum vertex cover"

implies two properties:

1. vertex cover
2. at most α times as large as minimum vertex cover

Approximations are always feasible solutions!

Exercises

- Warm-up puzzles
- Exercises of Chapter 1

Discussion & Exercises

DDA Course

Lecture 1.2

15 March 2012

Course Tracker

- **24** students registered for the course
- **7** reports in the course tracker
- Exercise 1.8: most popular, solved by 4/7
- Exercise 1.1: most difficult, 3/7 need help

Feedback

- Difficult: “approximation”

Plan

- Today we will:
  - review the concept of “approximation”
  - solve Exercise 1.1 together
  - discuss other exercises
- No new theory!
  - just make sure you are comfortable with
    the concepts of Chapter 1 by the end of the week…

Approximation

- Let $G = (V, E)$
- Assume that a minimum vertex cover of G has *3* nodes
- Assume that $C \subseteq V$ is a vertex cover, and there are *3, 4, 5, or 6* nodes in C
- Then "C is a *2-approximation* of a minimum vertex cover"

Approximation

- Let $G = (V, E)$
- Assume that a minimum vertex cover of G has at least *100* nodes
- Assume that $C \subseteq V$ is a vertex cover, and there are at most *105* nodes in C
- Then "C is a *1.05-approximation* of a minimum vertex cover"

Approximation

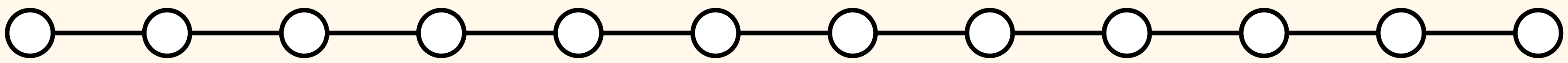
- Let $G = (V, E)$
- Assume that a maximum matching of G has *8* edges
- Assume that $M \subseteq E$ is a matching, and there are *4, 5, 6, 7, or 8* edges in M
- Then "M is a *2-approximation* of a maximum matching"

Approximation

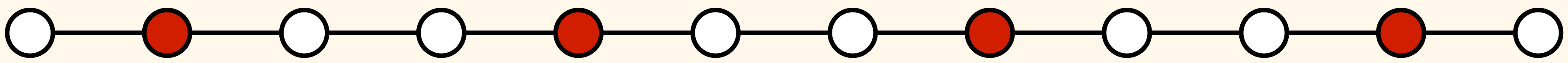
- Let $G = (V, E)$
- Assume that a maximum matching of G has at most *105* edges
- Assume that $M \subseteq E$ is a matching, and there are at least *100* edges in M
- Then "M is a *1.05-approximation* of a maximum matching"

Approximation

graph:



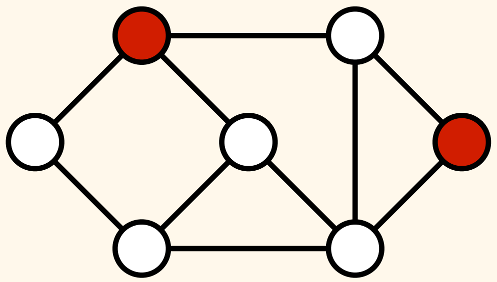
minimum dominating set:

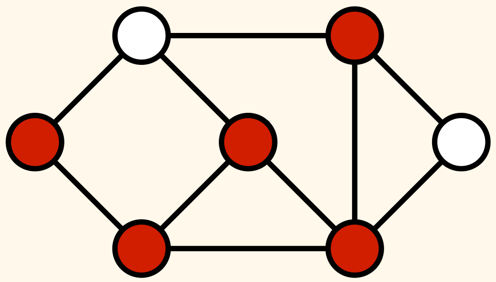


1.5-approximation of minimum dominating set:

Exercise 1.1a

- Let $I \subseteq V$ and $C = V \setminus I$
- *Claim*: I is an independent set iff C is a vertex cover

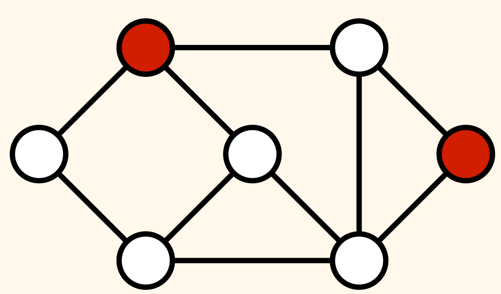
I: 

C: 

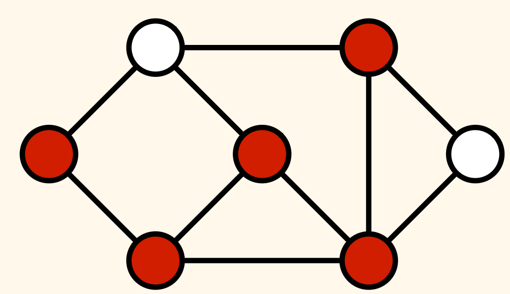
Exercise 1.1a

- Let $I \subseteq V$ and $C = V \setminus I$
- *Claim*: I is an independent set iff C is a vertex cover
- *Idea*: verify each edge

I:



C:



Exercise 1.1a

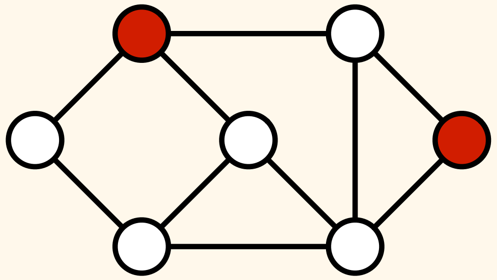
- Assume that I is an independent set:
  - let $e \in E$
  - definition of independent set: $|e \cap I| \leq 1$
  - edges have two endpoints: $|e \cap V| = 2$
  - therefore $e \cap (V \setminus I) \neq \varnothing$
- Therefore $V \setminus I$ is a vertex cover

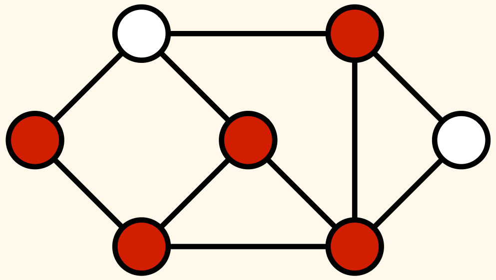
Exercise 1.1a

- Assume that C is a vertex cover:
  - let $e \in E$
  - definition of vertex cover: $e \cap C \neq \emptyset$
  - edges have two endpoints: $|e \cap V| = 2$
  - therefore $|e \cap (V \setminus C)| \leq 1$
- Therefore $V \setminus C$ is an independent set

Exercise 1.1a

- Let $I \subseteq V$ and $C = V \setminus I$
- *Claim*: I is an independent set iff C is a vertex cover
- *Proof*: verify each edge

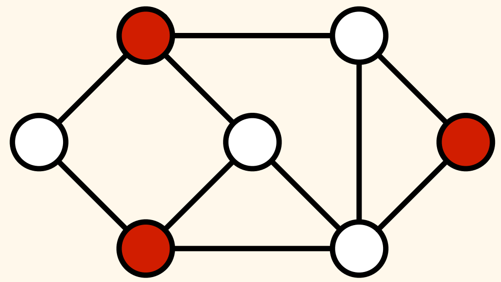
I: 

C: 

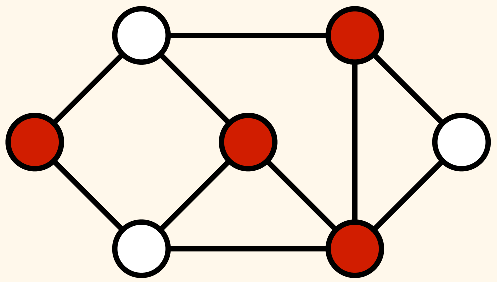
Exercise 1.1b

- Let $I \subseteq V$ and $C = V \setminus I$
- *Claim*: I is a maximal independent set iff C is a minimal vertex cover

I:



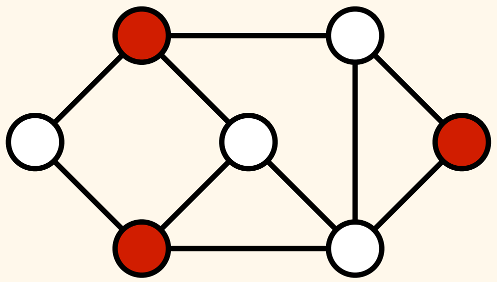
C:



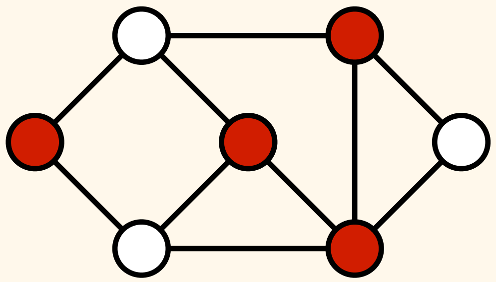
Exercise 1.1b

- Let $I \subseteq V$ and $C = V \setminus I$
- *Claim*: I is a maximal independent set iff C is a minimal vertex cover
- *Idea*: use 1.1a

I:

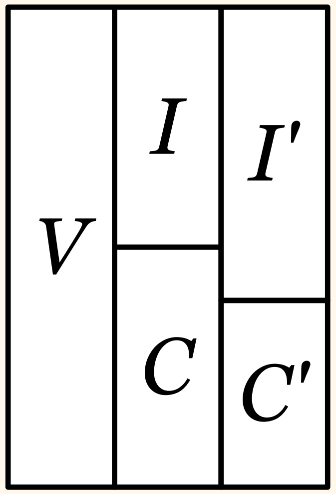


C:



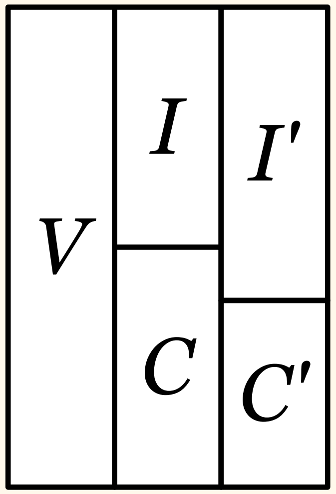
Exercise 1.1b

- Assume: I is a maximal independent set
  - define $C = V \setminus I$
  - then C is a vertex cover
  - assume that $C' \subset C$ is also a vertex cover
  - then $I' = V \setminus C'$ is an independent set
  - we have $I' \supset I$
  - therefore I was not maximal, contradiction



Exercise 1.1b

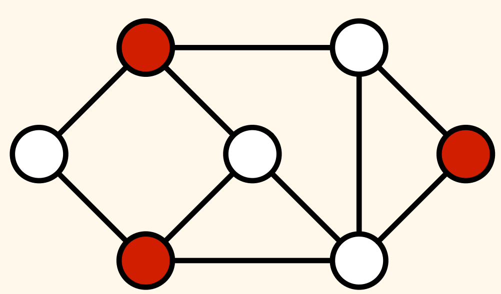
- Assume: C is a minimal vertex cover
  - define $I = V \setminus C$
  - similar: we already know that I is an independent set, only need to show maximality
  - assume that I is not maximal, then C cannot be minimal, contradiction



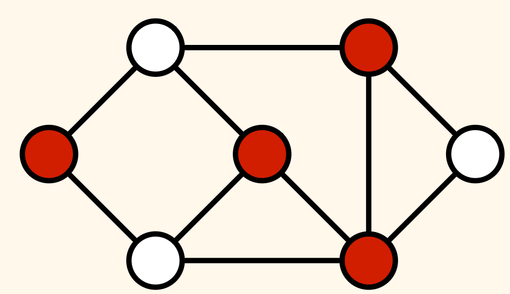
Exercise 1.1c

- Let $I \subseteq V$ and $C = V \setminus I$
- *Claim*: I is a maximum independent set iff C is a minimum vertex cover

I:



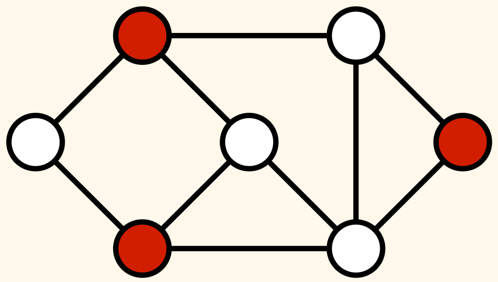
C:



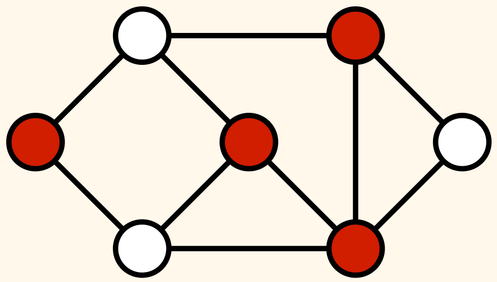
Exercise 1.1c

- Let $I \subseteq V$ and $C = V \setminus I$
- *Claim*: I is a maximum independent set iff C is a minimum vertex cover
- *Idea*: use 1.1a

I:



C:



Exercise 1.1c

- Assume: I is a maximum independent set
  - define $C = V \setminus I$
  - then C is a vertex cover
  - assume that C' is also a vertex cover, $|C'| < |C|$
  - then $I' = V \setminus C'$ is an independent set
  - we have $|I'| = |V| - |C'| > |V| - |C| = |I|$
  - therefore I was not of a maximum size, contradiction

Exercise 1.1c

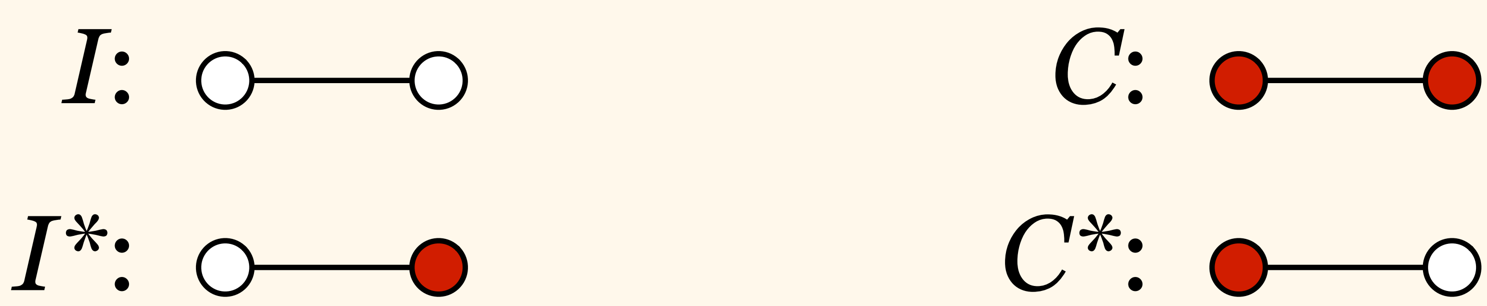
- Assume: C is a minimum vertex cover
  - define $I = V \setminus C$
  - again we already know that I is an independent set
  - similar: assume that there is a larger independent set, then C cannot be a minimum vertex cover, contradiction

Exercise 1.1d

- Show that the following is possible:
  - C is a 2-approximation of minimum vertex cover
  - $I = V \setminus C$ is not a 2-approximation of maximum independent set

Exercise 1.1d

- Show that the following is possible:
  - C is a 2-approximation of minimum vertex cover
  - $I = V \setminus C$ is not a 2-approximation of maximum independent set

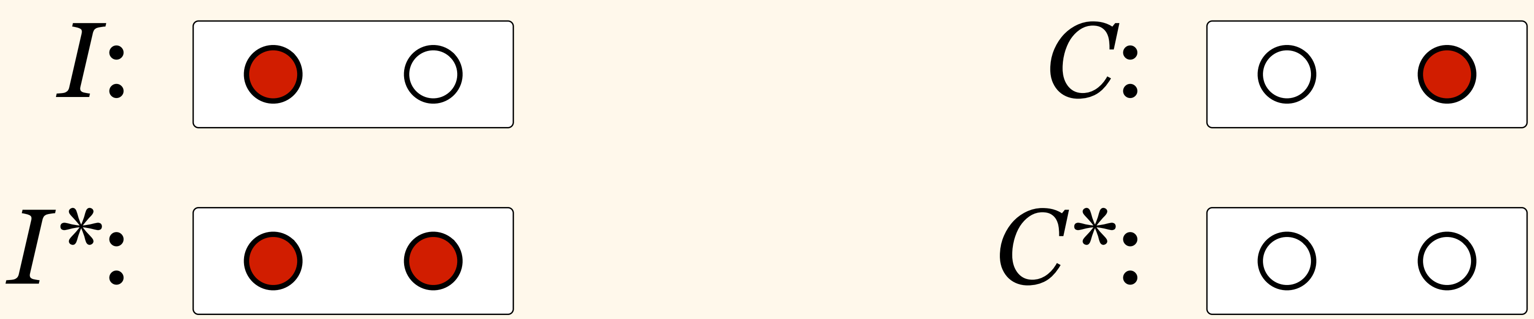


Exercise 1.1e

- Show that the following is possible:
  - I is a 2-approximation of maximum independent set
  - $C = V \setminus I$ is not a 2-approximation of minimum vertex cover

Exercise 1.1e

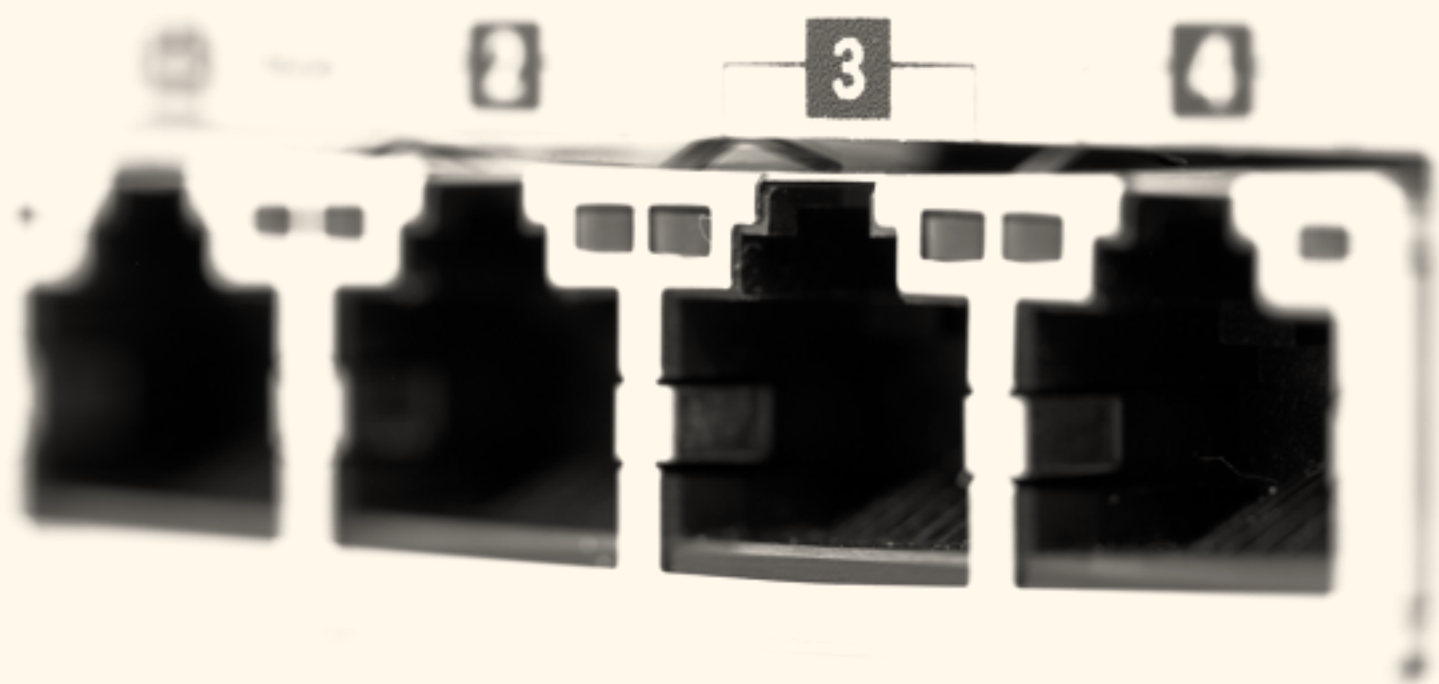
- Show that the following is possible:
  - I is a 2-approximation of maximum independent set
  - $C = V \setminus I$ is not a 2-approximation of minimum vertex cover



Schedule

- Today:
    - questions? comments?
- Tomorrow:
    - last chance to discuss exercises of Chapter 1
- Next week:
    - Chapter 2 — remember to read it *before* the lectures

Port-Numbering Model



DDA Course

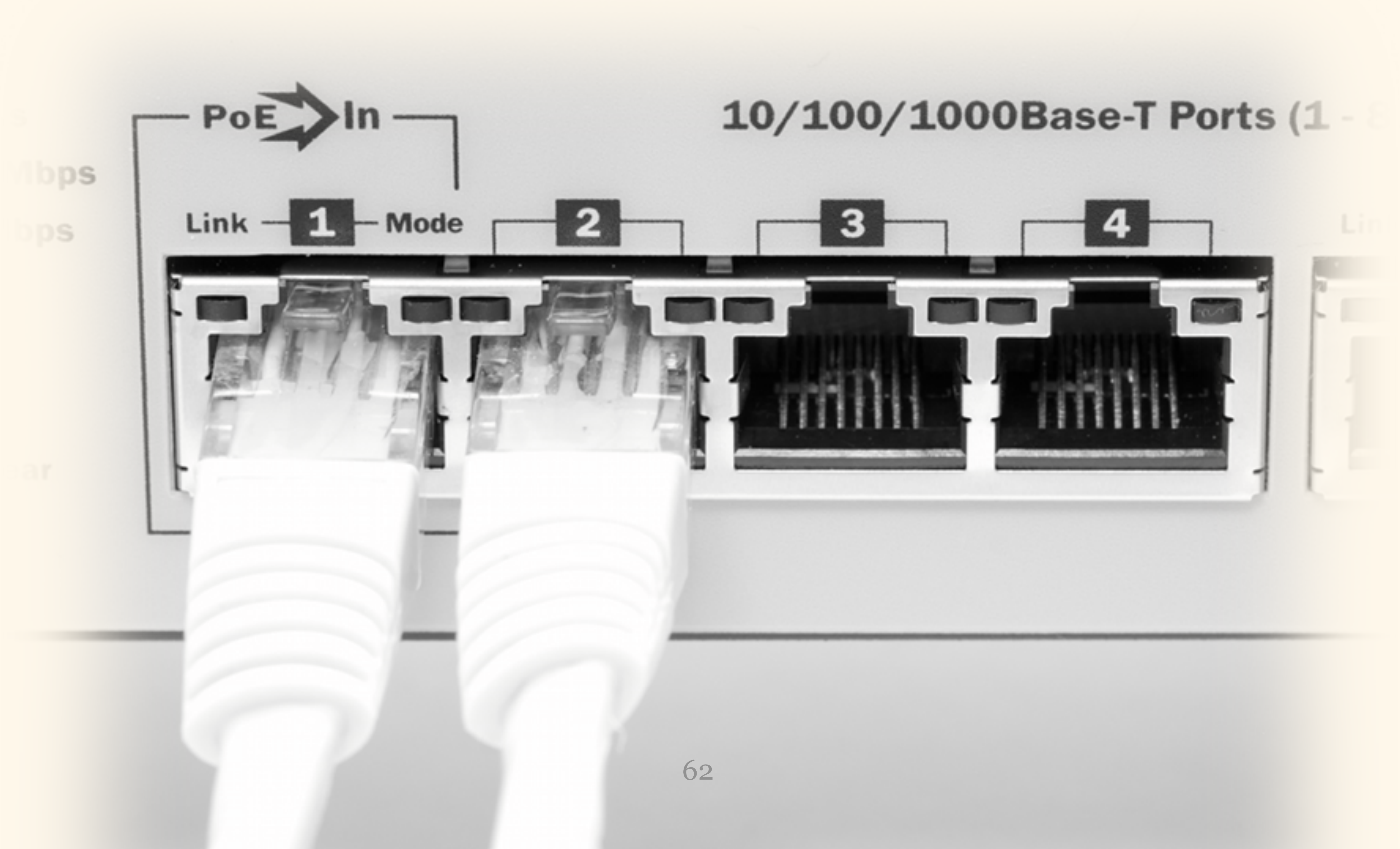
Lecture 2.1

20 March 2012

Distributed Systems

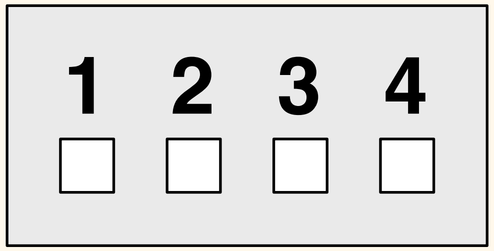
- Intuition:
  - distributed system
    ≈ communication network
    ≈ network equipment + communication links
  - distributed algorithm
    ≈ computer program
- Precisely how are we going to model this?

Port Numbering



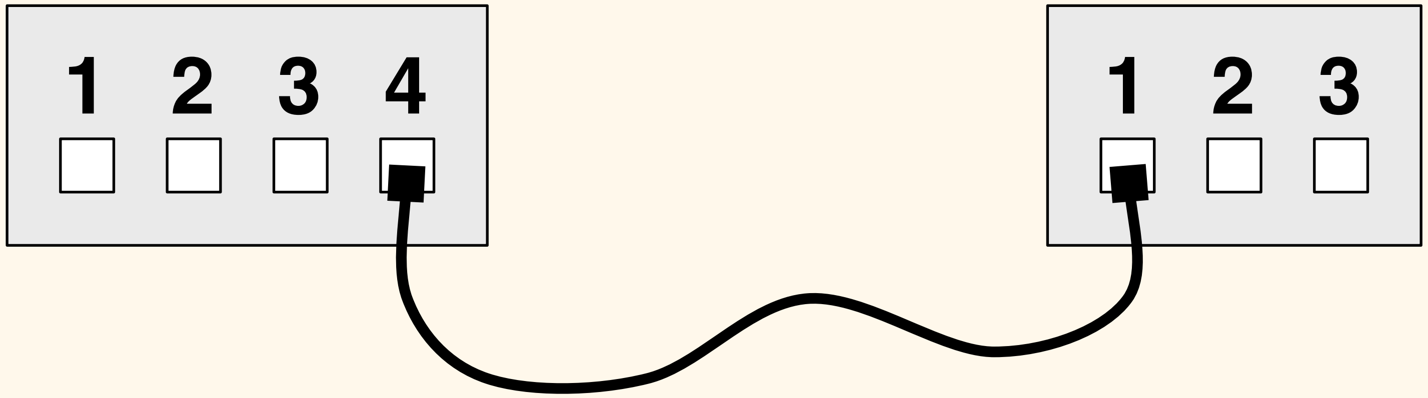
Port Numbering

- Network device = state machine with *communication ports*
- Ports are *numbered*: 1, 2, 3, …



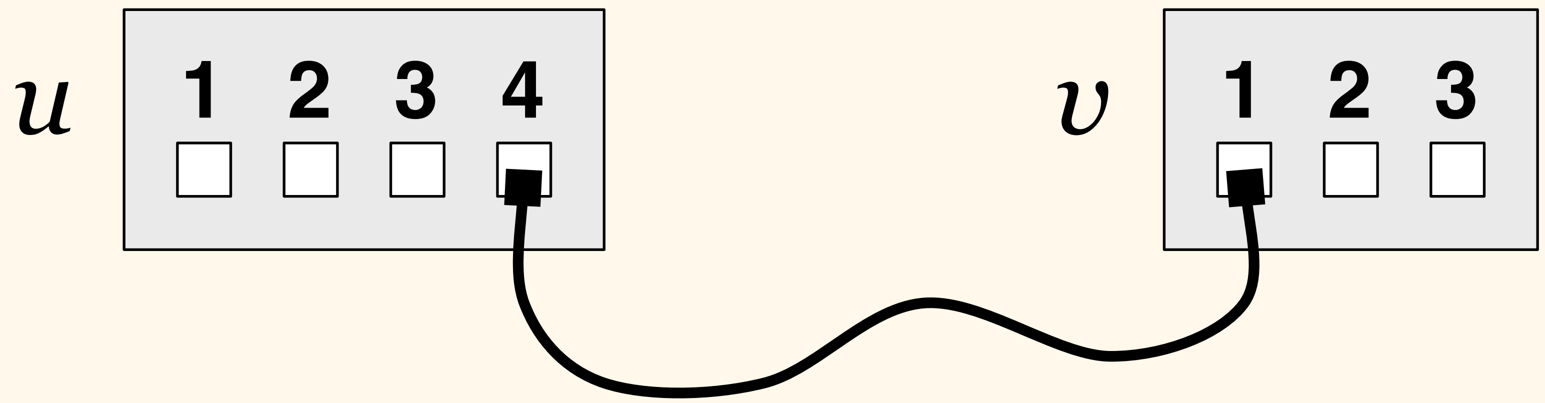
Port-Numbered Network

- Network = several devices, *connections* between ports
    - we will formalise it as a triple $N = (V, P, p)$



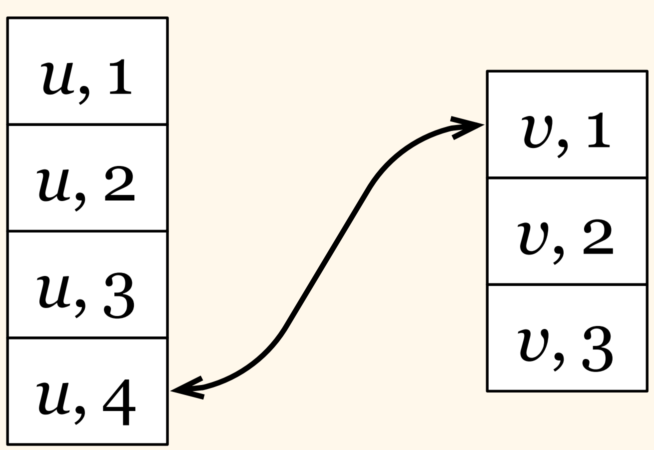
Port-Numbered Network

- nodes $V = \{u, v, \ldots\}$
- ports $P = \{(u, 1), (u, 2), (u, 3), (u, 4), (v, 1), (v, 2), (v, 3), \ldots\}$
- connections $p(u, 4) = (v, 1)$, $p(v, 1) = (u, 4)$, ...



Port-Numbered Network

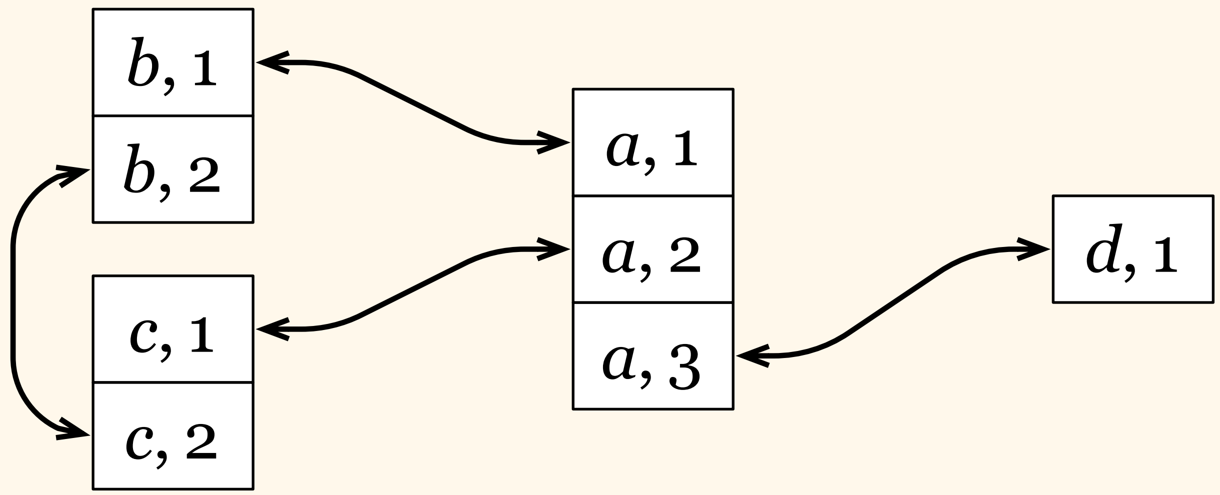
- nodes $V = \{u, v, \ldots\}$
- ports $P = \{(u, 1), (u, 2), (u, 3), (u, 4), (v, 1), (v, 2), (v, 3), \ldots\}$
- connections $p(u, 4) = (v, 1)$, $p(v, 1) = (u, 4)$, ...



not a complete example,
some ports not connected!

Port-Numbered Network

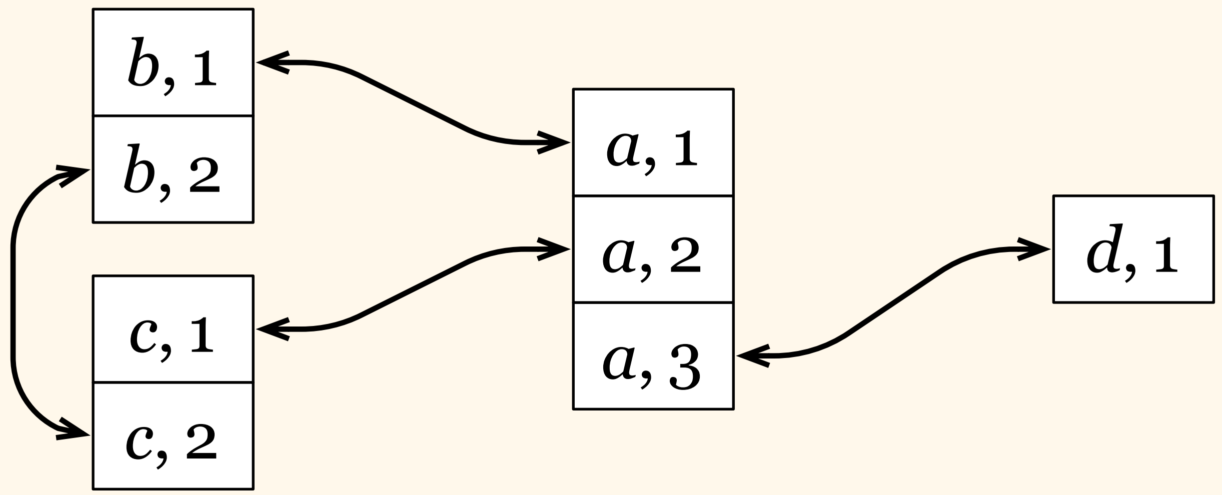
- nodes $V = \{a, b, c, d\}$
- ports $P = \{(a,1), (a,2), (a,3), (b,1), (b,2), (c,1), (c,2), (d,1)\}$
- connections $p(a,1) = (b,1)$, $p(b,1) = (a,1)$, ...



all ports connected

Port-Numbered Network

- nodes V = a finite set
- ports P = a finite set of (node, number) pairs
- connections p = an involution $P \to P$



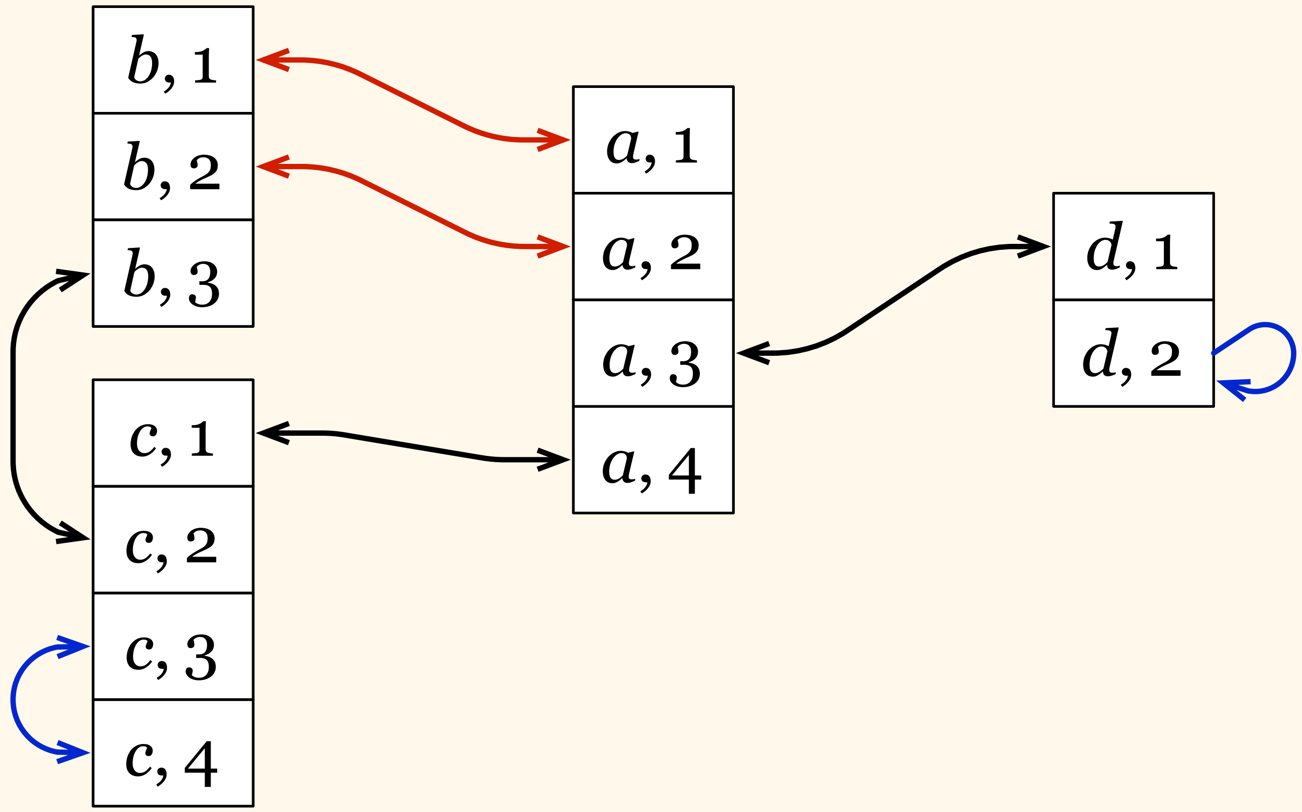
involution:

$p^{-1} = p$

$p(p(x)) = x$

Port-Numbered Network

- We may have *multiple connections* or *loops*



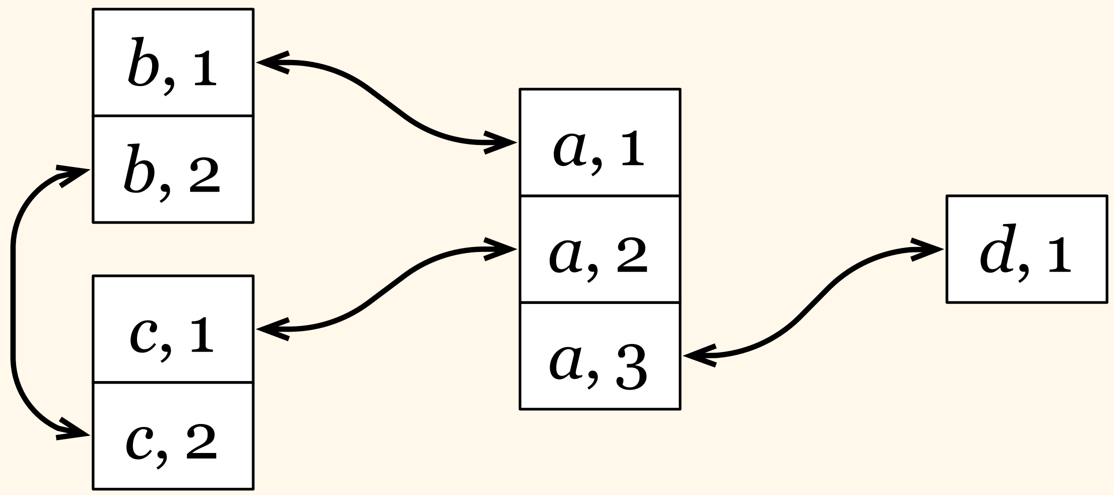
$p(c, 3) = (c, 4)$

$p(c, 4) = (c, 3)$

$p(d, 2) = (d, 2)$

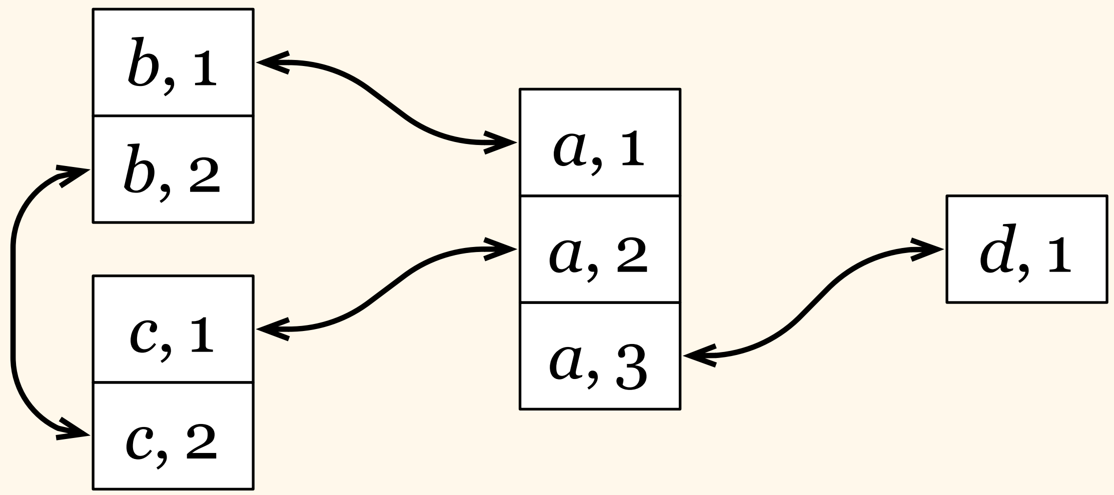
Port-Numbered Network

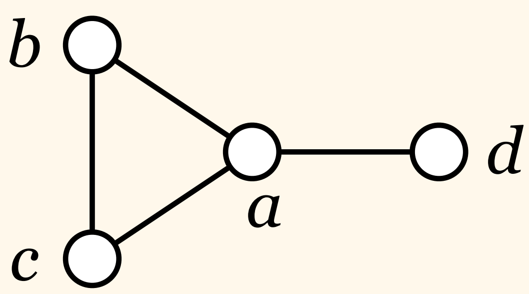
- *Simple* port-numbered network:
  no multiple connections, no loops



Port-Numbered Network

- *Underlying graph* of a simple port-numbered network





Distributed Algorithms

Distributed Algorithm

- State machine, x = current state:
  - $x \leftarrow$ **init**(z): initial state for local input z
  - **send**(x): construct *outgoing messages*
    - send(x) = vector, one element per port
  - $x \leftarrow$ **receive**(x, m): process *incoming messages*
    - m = vector, one element per port

Execution

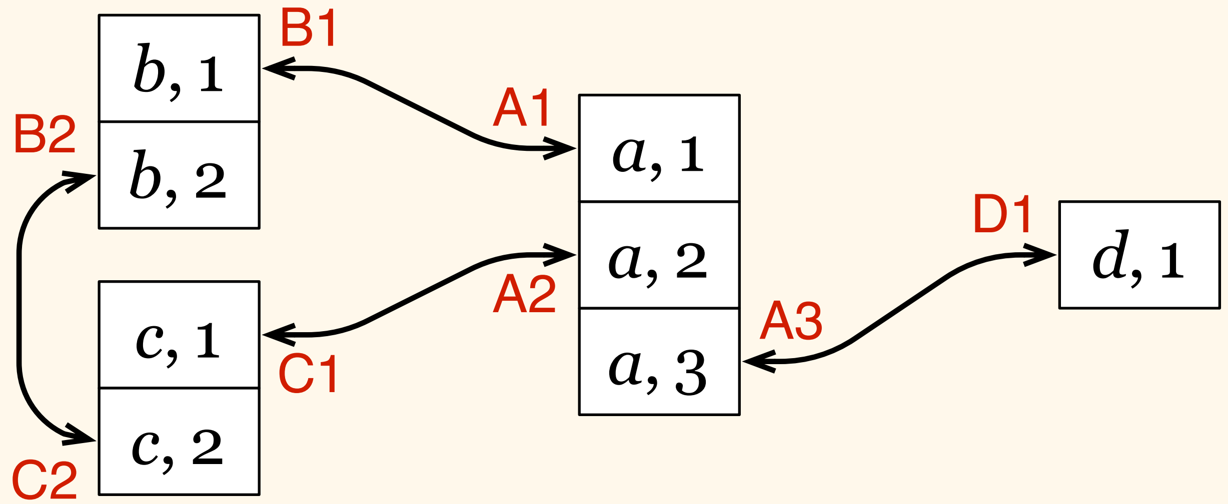
- "Execution of algorithm *A* in network *N*"
- All nodes of *N* are *identical copies* of the same state machine *A*
  - functions **init**, **send**, and **receive** may depend on node degree (number of ports)
  - in all other aspects the nodes are identical

Execution

- All nodes are initialised
- Time step (*communication round*):
  - all nodes construct outgoing messages
  - messages are propagated
  - all nodes process incoming messages
- Continue until all nodes have stopped

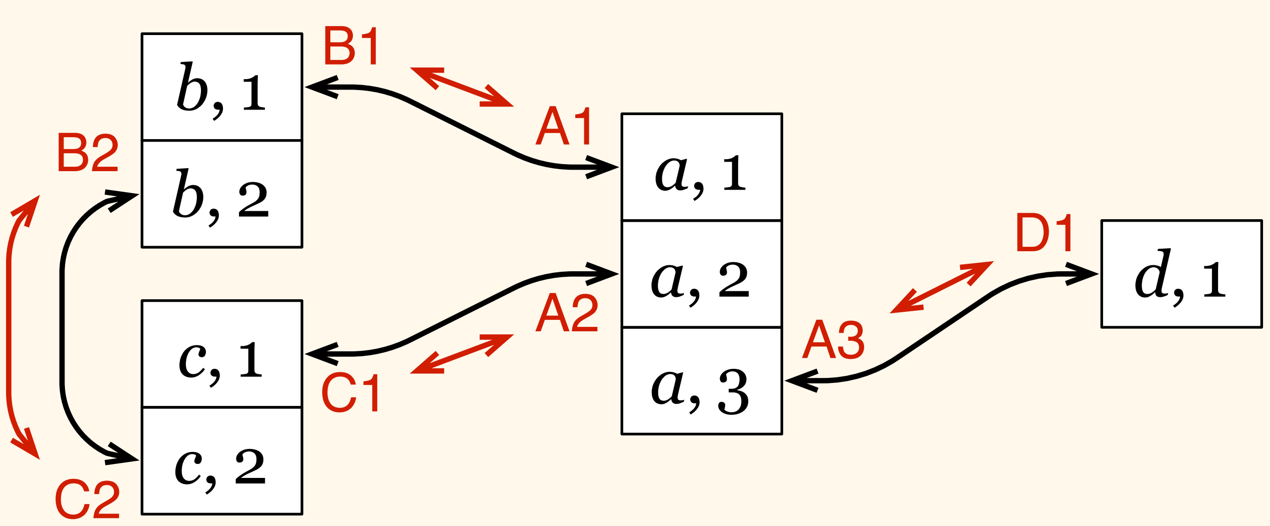
Communication Round

- Construct *outgoing messages*



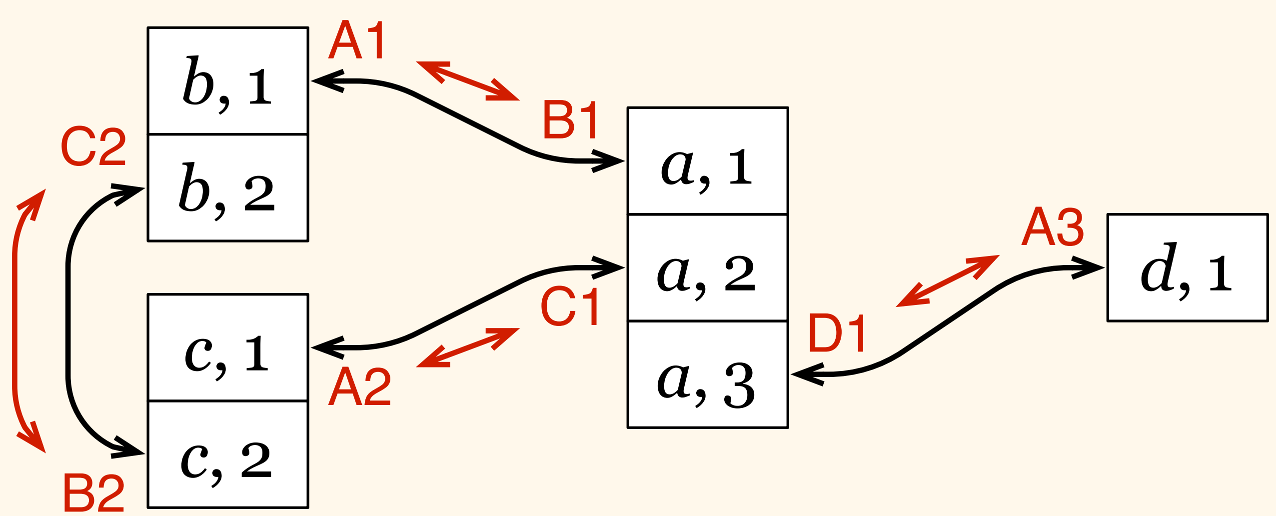
Communication Round

- Construct outgoing messages
- Exchange messages along communication links



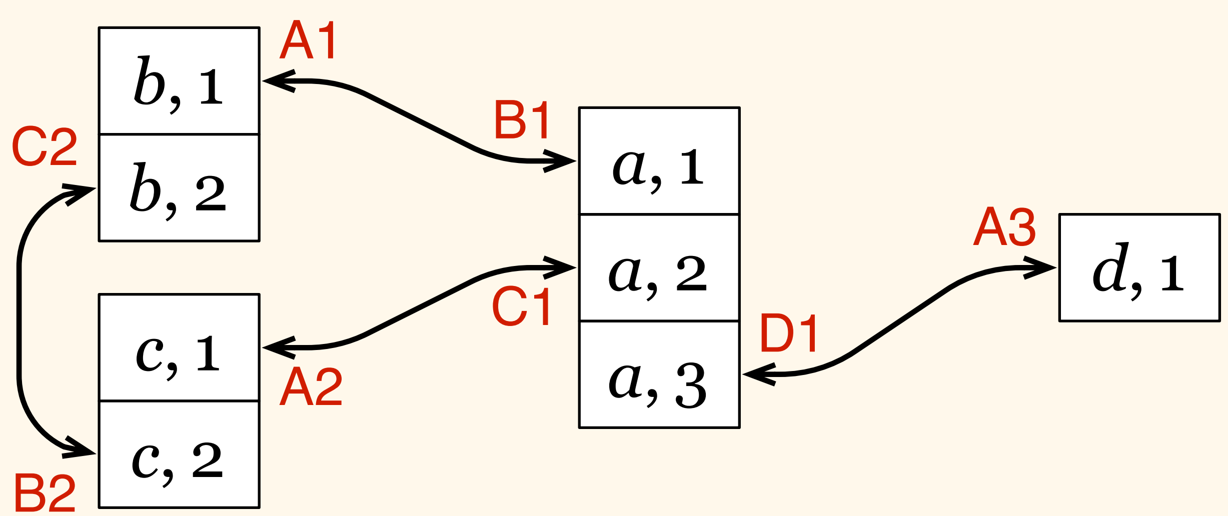
Communication Round

- Construct outgoing messages
- Exchange messages along communication links



Communication Round

- Construct outgoing messages
- Exchange messages along communication links
- Process *incoming messages*



Communication Round

- Construct outgoing messages
- Exchange messages along communication links
- Process incoming messages

- Communication rounds are *synchronous*
- Each step happens synchronously in parallel for all nodes
- Everything is *deterministic*

Distributed Algorithm

- Algorithm designed chooses:
  - how to initialise nodes
  - how to construct outgoing messages
  - how to process incoming messages
- Network structure determines:
  - how messages are propagated between ports

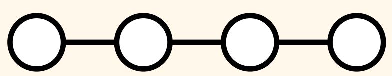
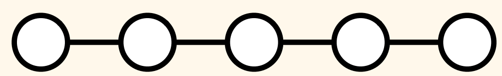
Distributed Algorithm

- "Algorithm A solves graph problem Π on graph family $\mathcal{F}$":
  - for any graph $G \in \mathcal{F}$,
  - for *any simple port-numbered network* N that has G as underlying graph,
  - execution of A on N stops and produces a valid solution of Π

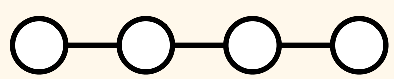
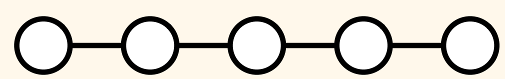
Distributed Algorithm

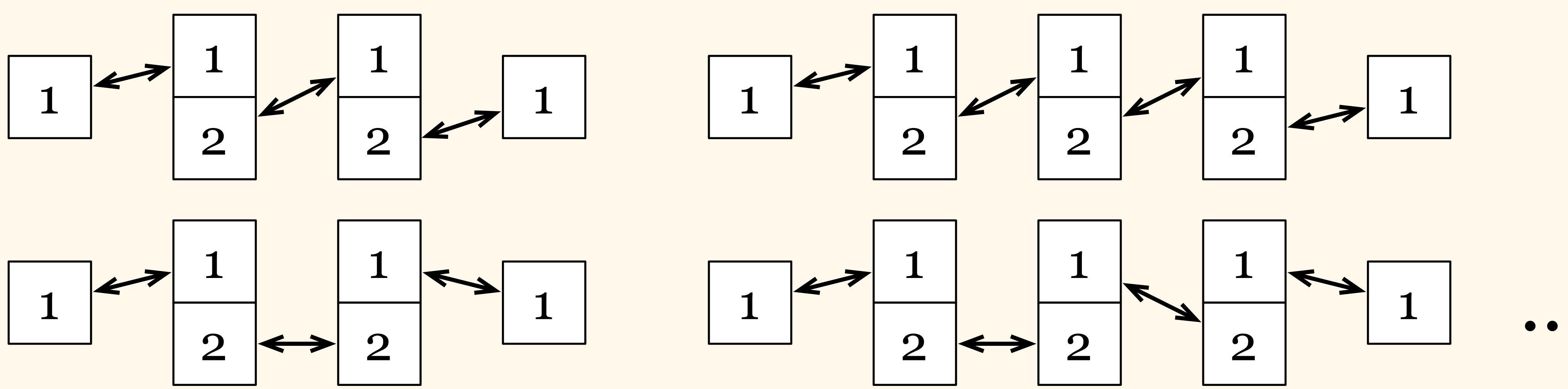
- "Algorithm A finds a minimum vertex cover in any regular graph":
  - for *any simple port-numbered network* N that has a regular graph as underlying graph,
  - execution of A on N stops,
  - the stopping states of the nodes are "**0**" and "**1**",
  - nodes in state "**1**" form a minimum vertex cover

Example

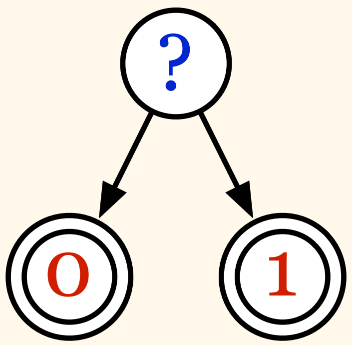
- Design a distributed algorithm that finds a *minimum vertex cover* in $\mathcal{F} = \{$, $\}$

Example

- Design a distributed algorithm that finds a *minimum vertex cover* in $\mathcal{F} = \{$ ,  $\}$



Example



- Nodes of degree 1:
  - $\text{init}_1 = ?$, $\text{send}_1(?) = (A)$
  - $\text{receive}_1(?, A) = 0$, $\text{receive}_1(?, B) = 0$
- Nodes of degree 2:
  - $\text{init}_2 = ?$, $\text{send}_2(?) = (B, B)$
  - $\text{receive}_2(?, A, A) = 1$, $\text{receive}_2(?, A, B) = 1$, $\text{receive}_2(?, B, A) = 1$, $\text{receive}_2(?, B, B) = 0$

Example

- Design a distributed algorithm that finds a *minimum vertex cover* in $\mathcal{F} = \{$○–○–○–○, ○–○–○–○–○$\}$

- Solved!

- Running time: 1 communication round

General Principles

General Principles

- Synchronous execution
    - “worst case”
    - synchronisers exist

General Principles

- Synchronous execution
- Deterministic algorithms
  - cf. the name of this course
  - nodes do not have any source of randomness

General Principles

- Synchronous execution
- Deterministic algorithms
- Anonymous networks
  - identical nodes (except for their degree)
  - Chapters 5–6: what happens if each node has a unique name

General Principles

- Synchronous execution
- Deterministic algorithms
- Anonymous networks
- Time = number of communication rounds
    - focus on communication, not computation…

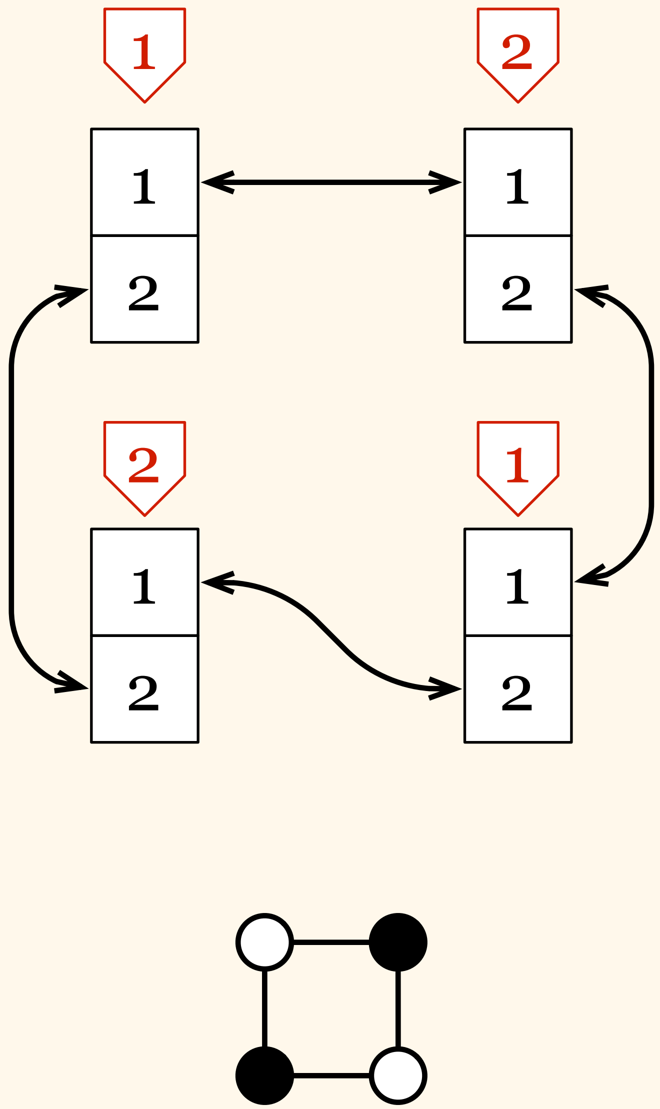
Examples

Maximal Matching

- We will design distributed algorithm BMM that finds a *maximal matching* in any *2-coloured graph*
  - we assume that we are given a proper 2-colouring of the underlying graph as input
  - algorithm will output a maximal matching

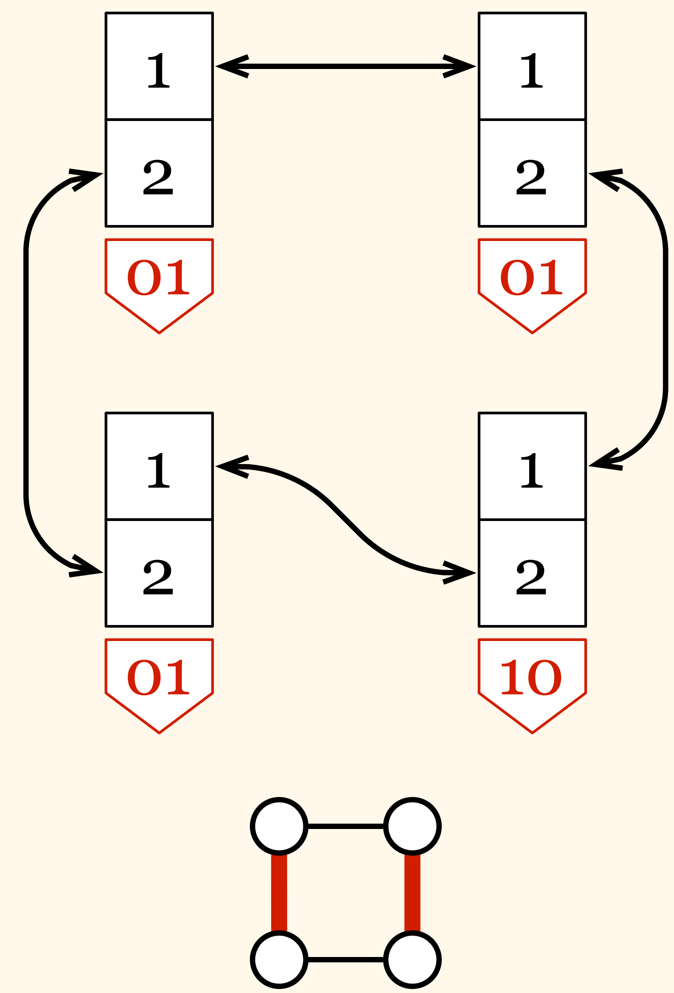
Given

encoding of 2-colouring



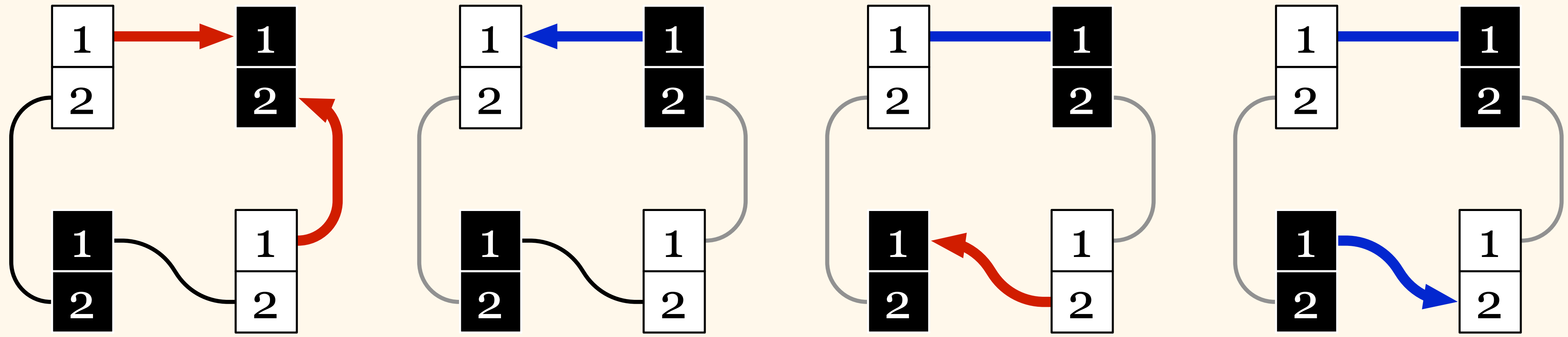
Find

encoding of maximal matching



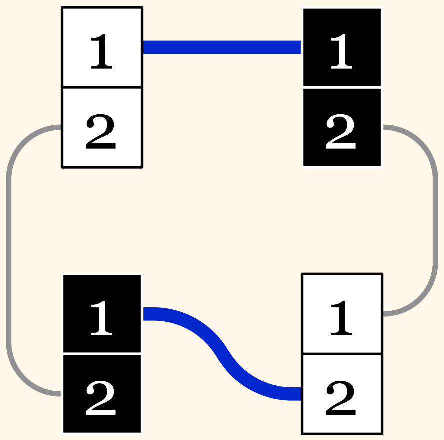
Maximal Matching

- Algorithm idea:
  - white nodes send *proposals* to their ports, one by one
  - black nodes *accept* the first proposal that they get



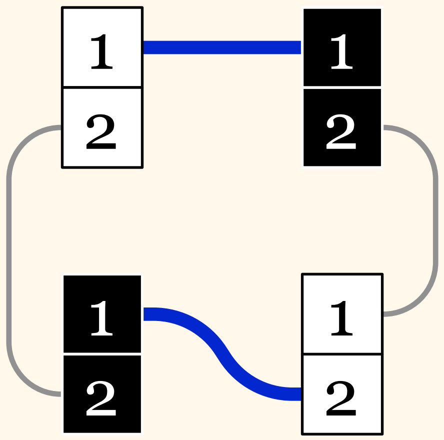
Maximal Matching

- Algorithm idea:
  - white nodes send *proposals* to their ports, one by one
  - black nodes *accept* the first proposal that they get
  - proposal–accept pair = edge in matching
- Running time: $O(\Delta)$
  - Δ = maximum degree



Maximal Matching

- We can find a maximal matching if we are given a 2-colouring
  - some auxiliary information is necessary, as we will see in Chapter 3
- Application: vertex cover approximation
  - works correctly in any network, no need to have 2-colouring!

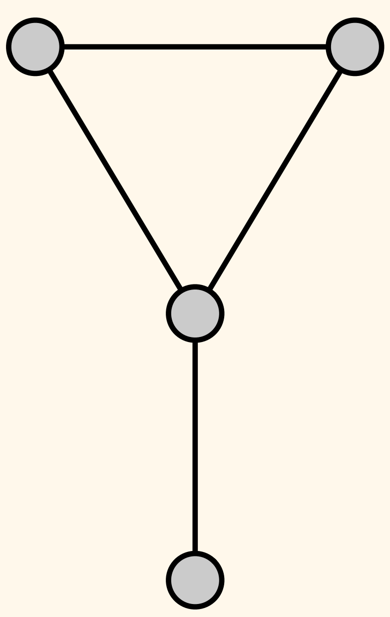


Vertex Cover

- We will design distributed algorithm VC3 that finds a *3-approximation of minimum vertex cover* in any graph
  - each node stops and outputs “0” or “1”
  - nodes that output “1” form a 3-approximation of a minimum vertex cover for the underlying graph

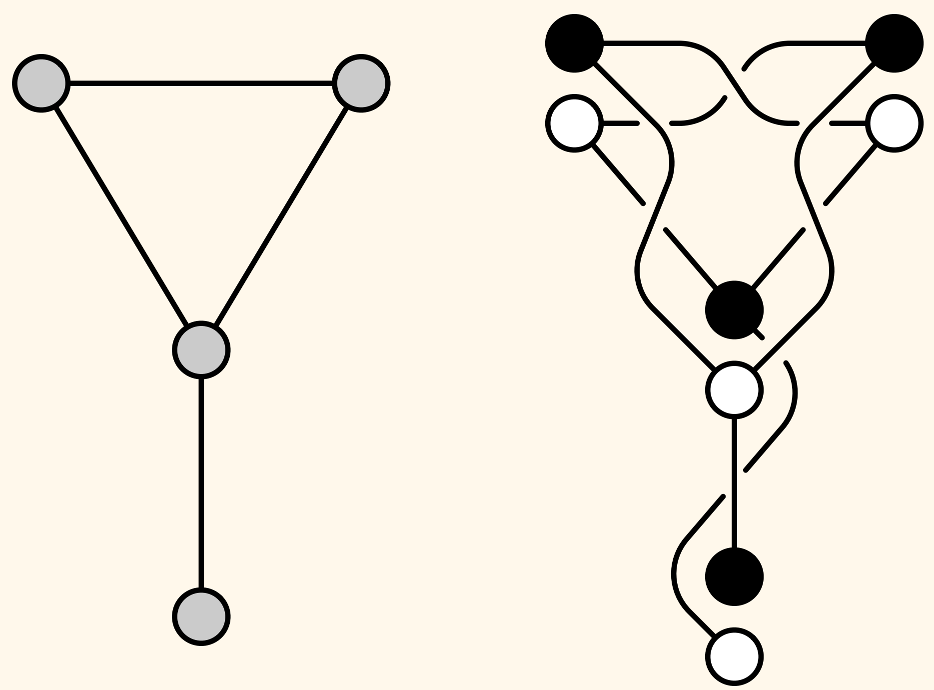
Vertex Cover

- Given: a port-numbered network
  - drawing here just the underlying graph…



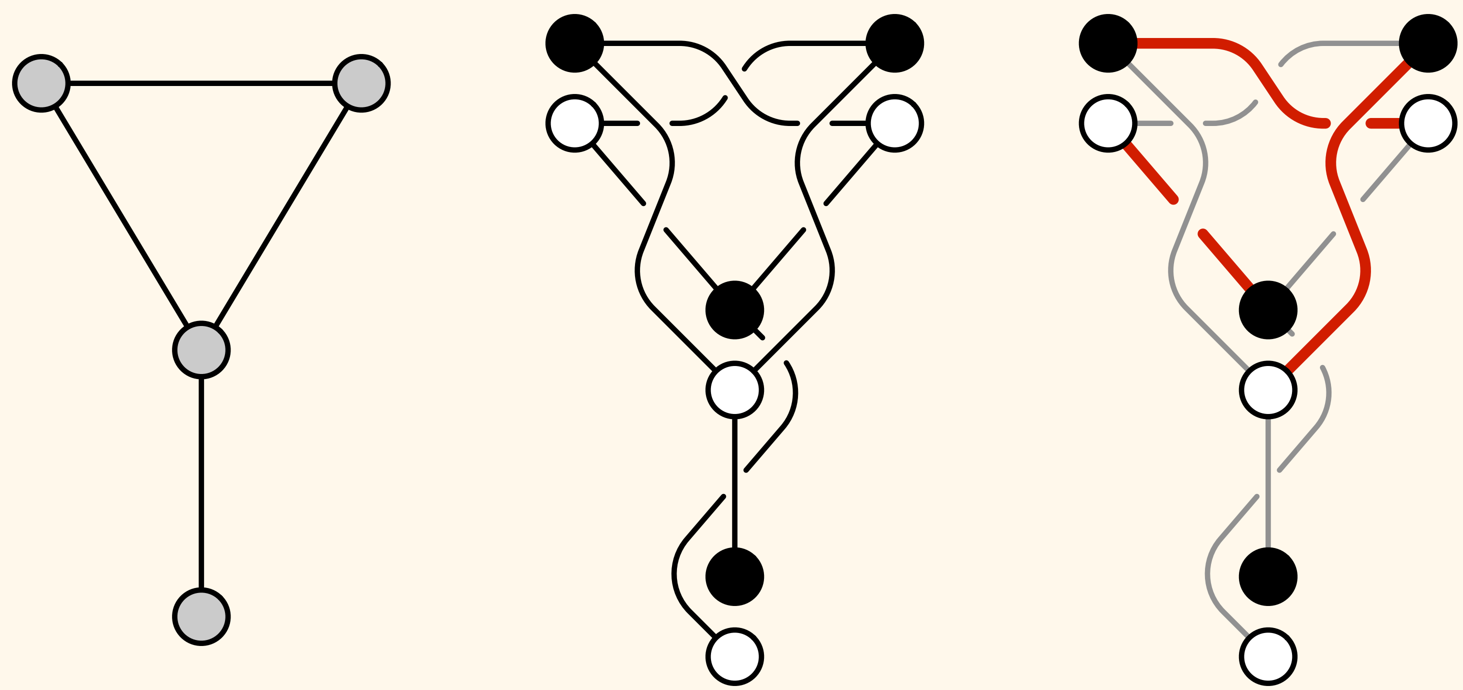
Vertex Cover

- Construct the *bipartite double cover*: two copies of each node, edges across



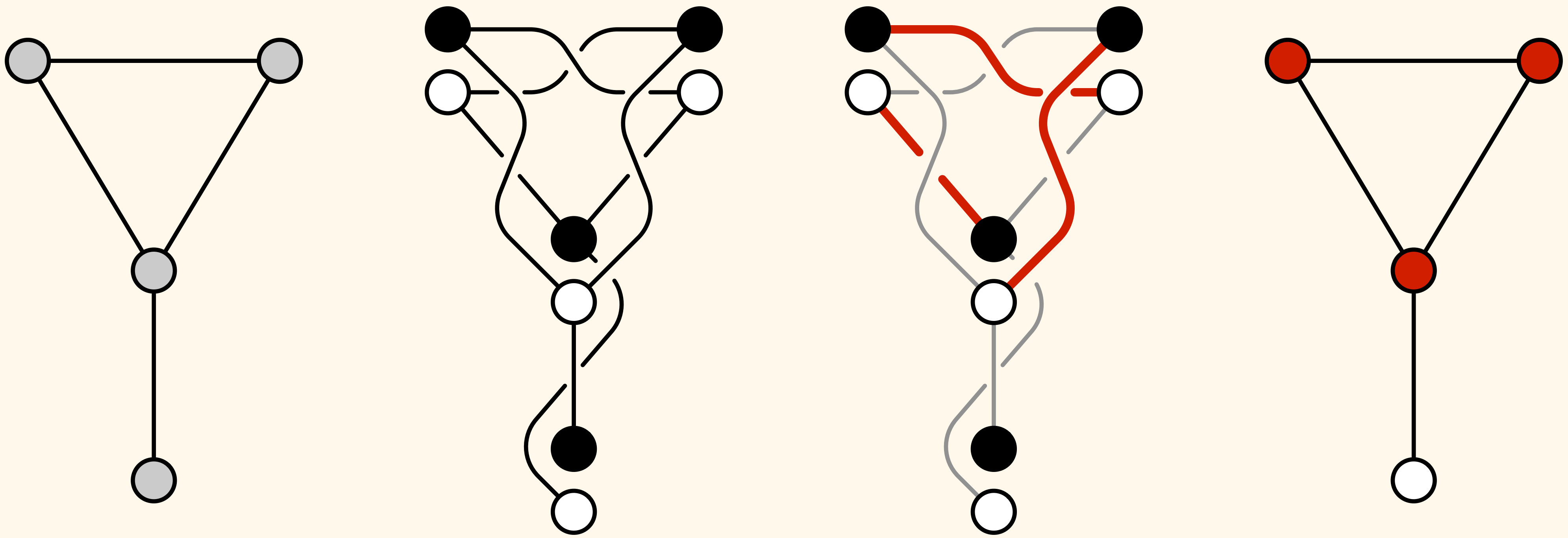
Vertex Cover

- Simulate algorithm BMM, outputs a *maximal matching* *M'*



Vertex Cover

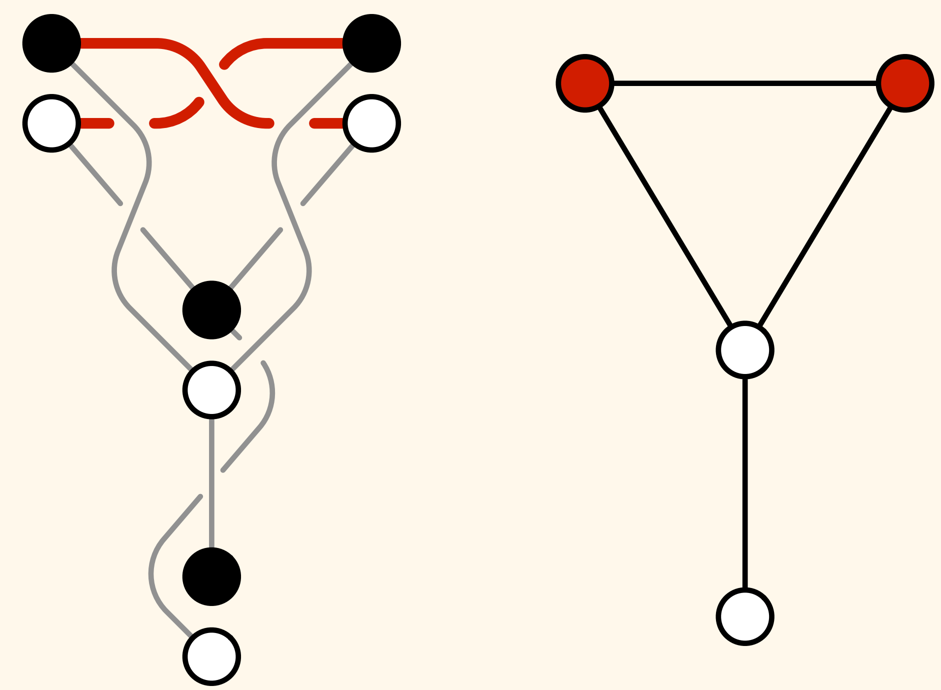
- C = nodes with at least one copy matched: 3-approximation of minimum vertex cover!



Vertex Cover

- C = nodes with at least one copy matched: 3-approximation of minimum vertex cover!

- Why vertex cover?
  - assume that there is an uncovered edge
  - conclude that M' is not maximal

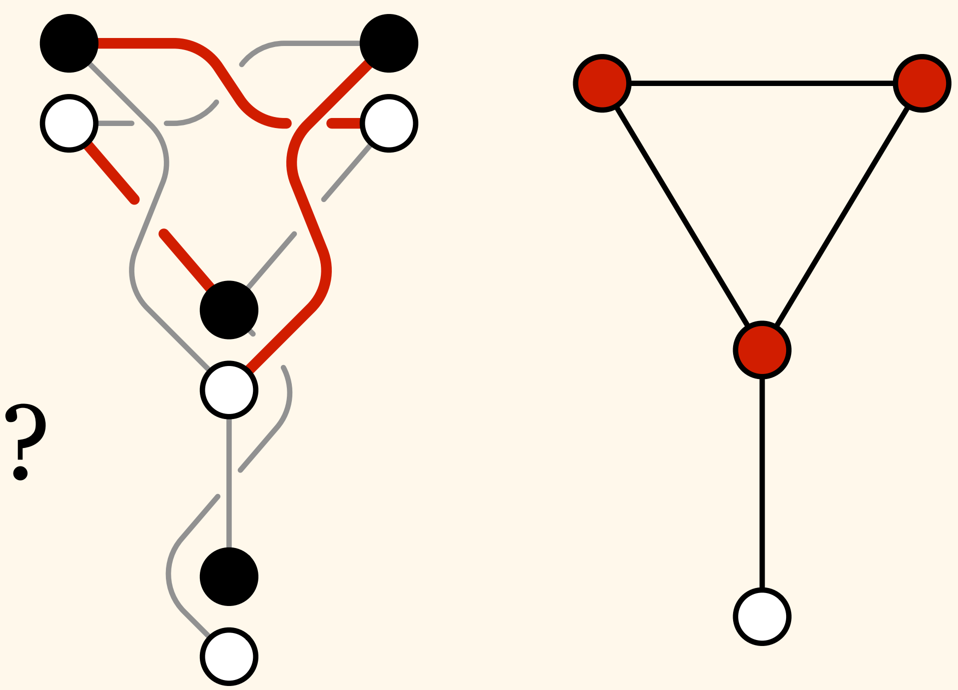


Vertex Cover

- C = nodes with at least one copy matched: 3-approximation of minimum vertex cover!

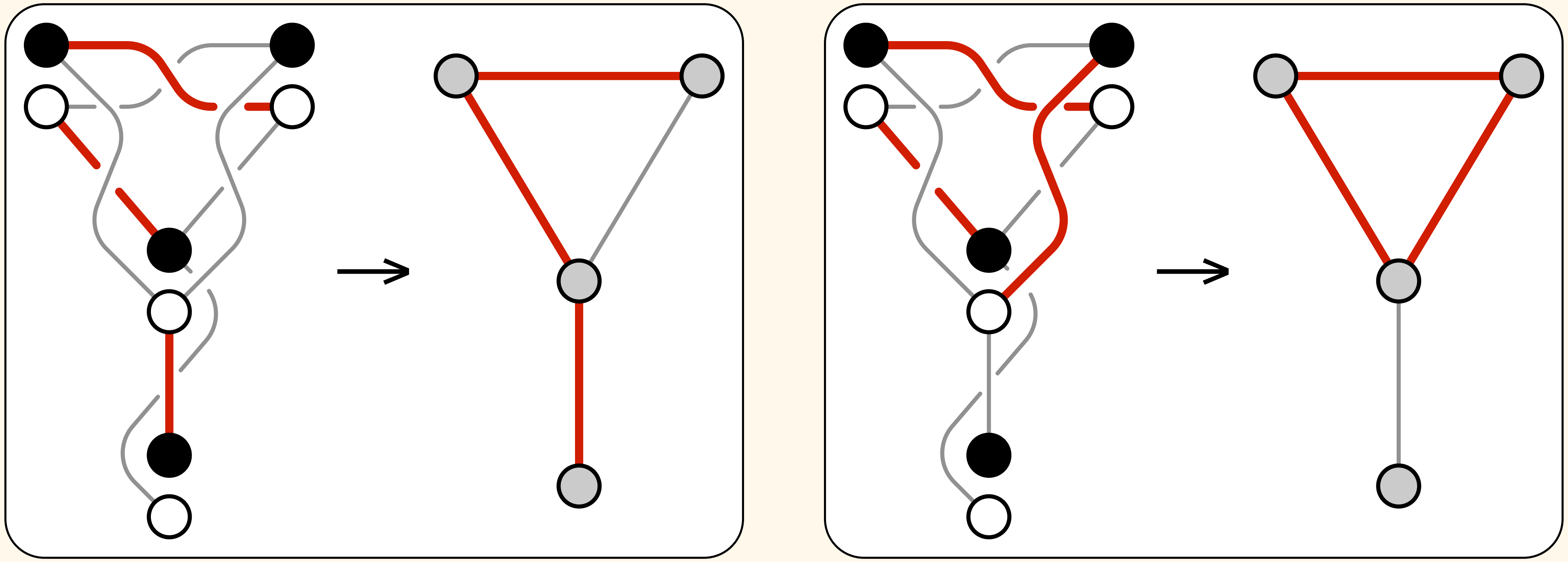
- Why vertex cover?

- Why 3-approximation?



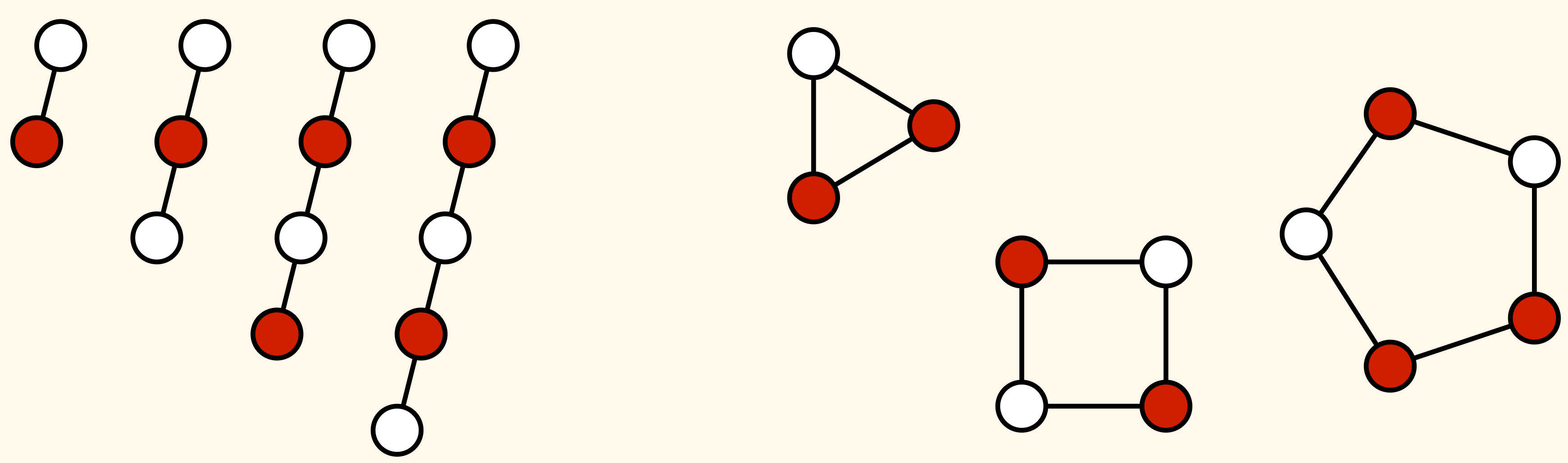
Vertex Cover

- Idea: matching in bipartite double cover
  → paths and/or cycles in original graph



Vertex Cover

- Any vertex cover contains at least 1/3 of nodes of any path or cycle
- 3-approximation if we take all of these



Summary

- We can solve non-trivial problems with distributed algorithms
  - e.g., 3-approximation of minimum vertex cover
- What next?
  - week 3: problems that cannot be solved at all
  - week 4: more positive results
  - weeks 5–6: what changes if the nodes have names?

Discussion & Exercises

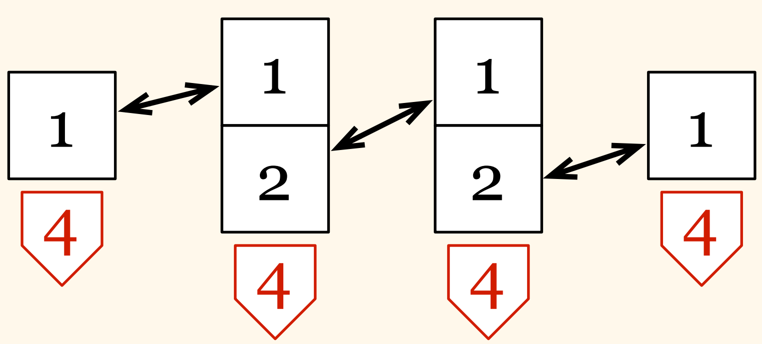
DDA Course

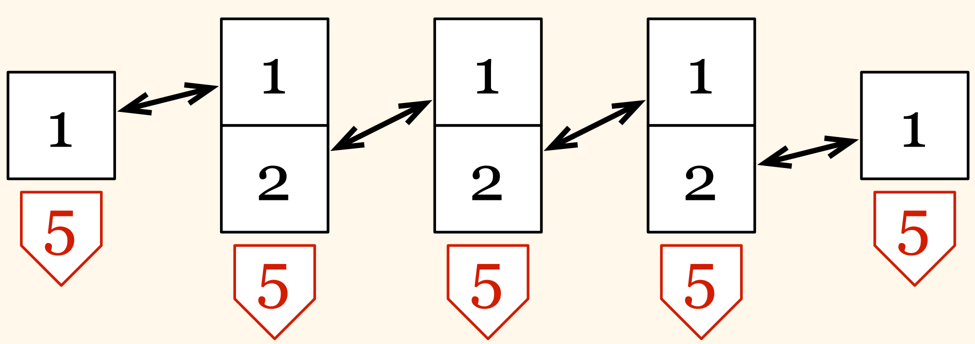
Lecture 2.2

22 March 2012

Counting

- Design a distributed algorithm that counts the number of nodes in any *path graph*
  - given a simple port-numbered network $N = (V, P, p)$ that has a path graph as the underlying graph, all nodes stop and output $|V|$

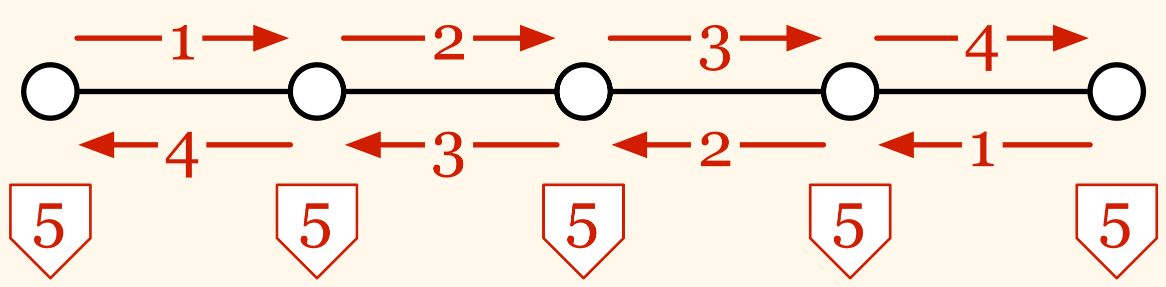




Counting

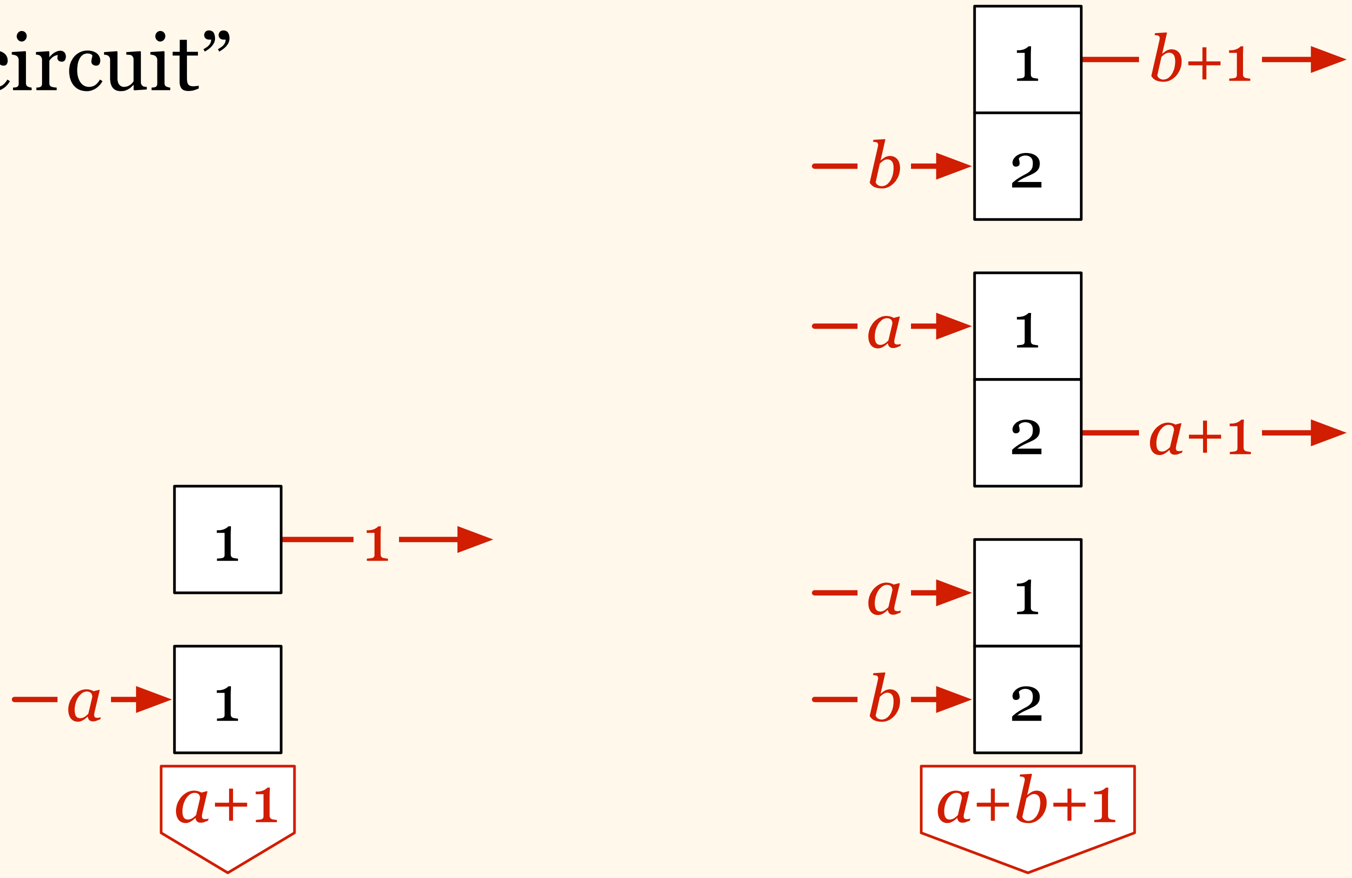
- Design a distributed algorithm that counts the number of nodes in any *path graph*

- Algorithm idea:



Counting

- Algorithm for path graphs
  - “arithmetic circuit”

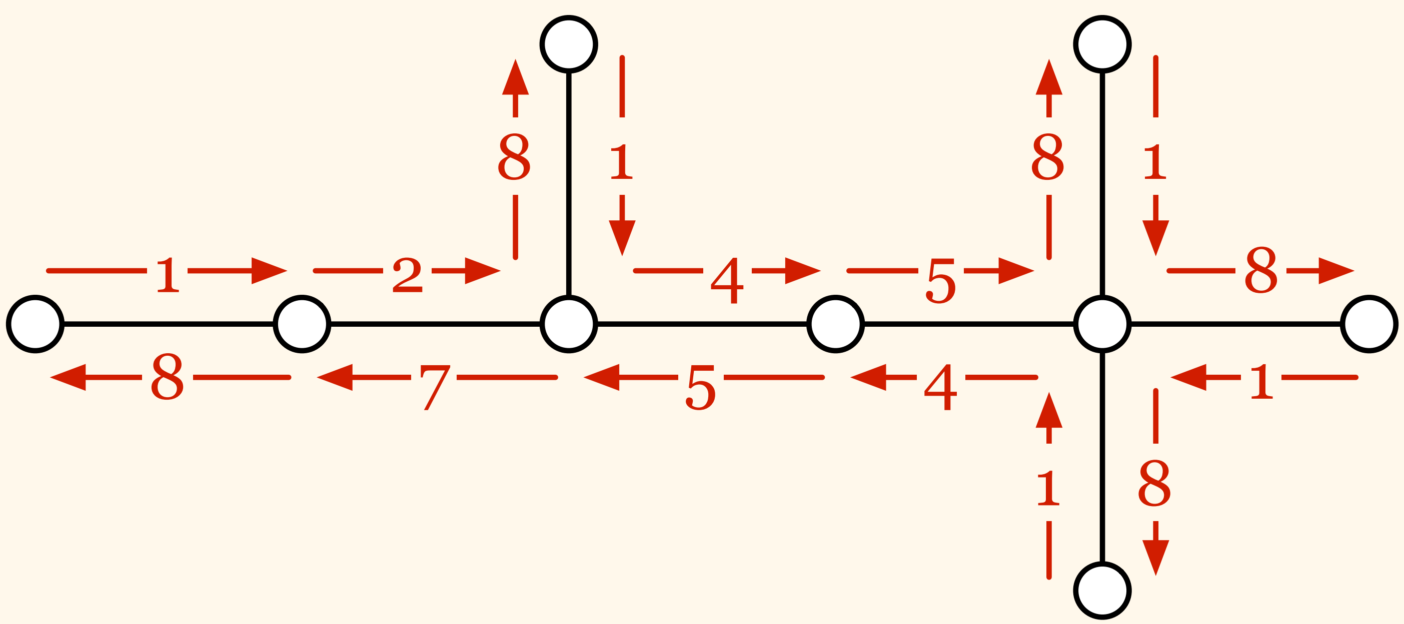


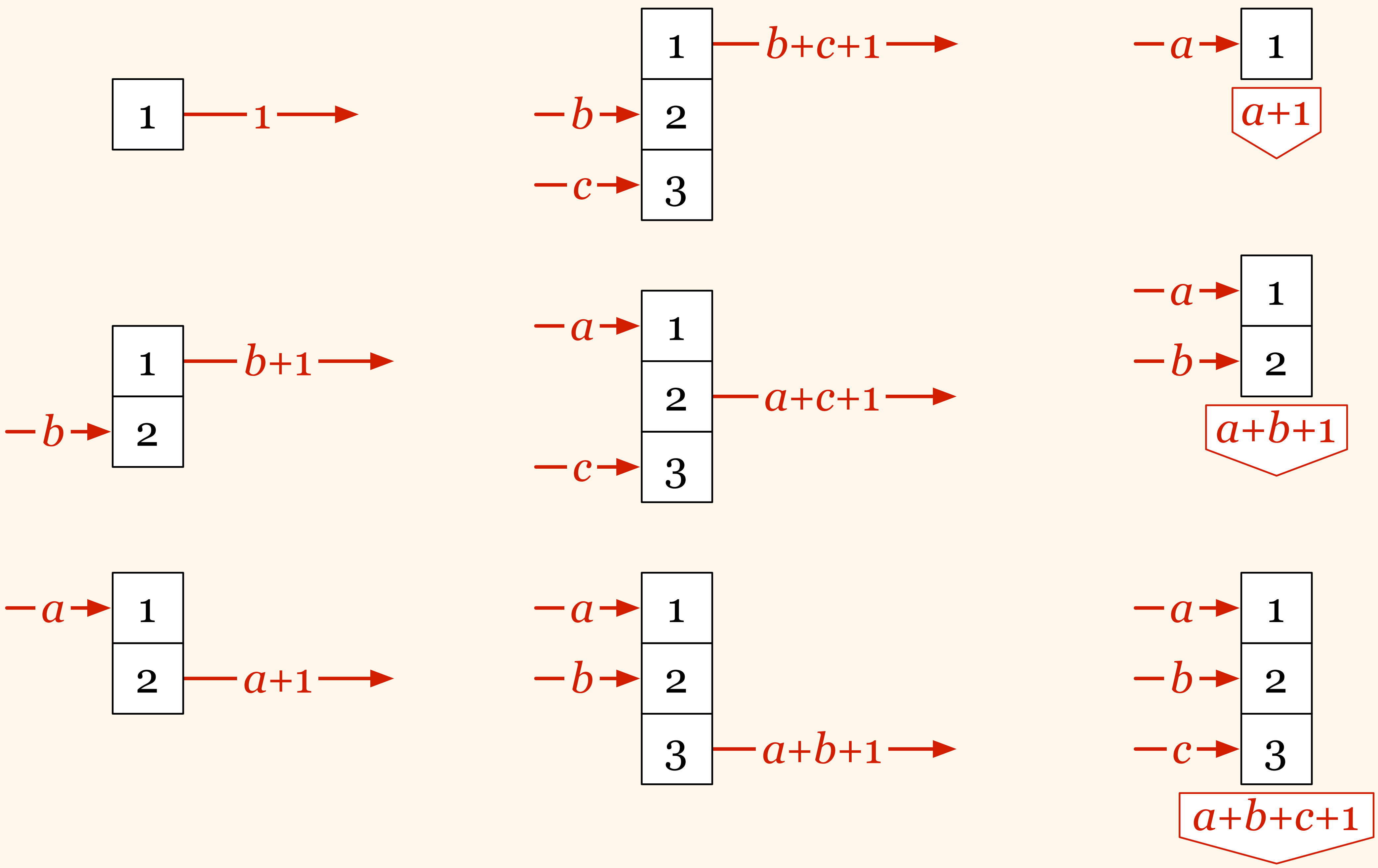
Counting

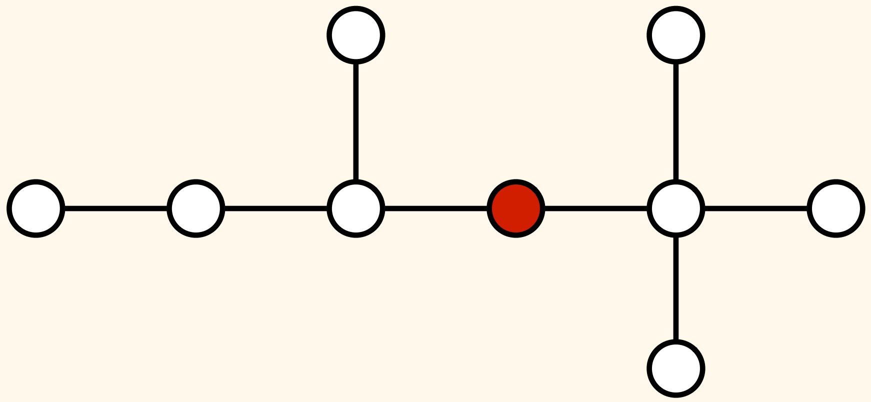
- Design a distributed algorithm that counts the number of nodes in any *tree*
  - given a simple port-numbered network $N = (V, P, p)$ that has a tree as the underlying graph, all nodes stop and output $|V|$

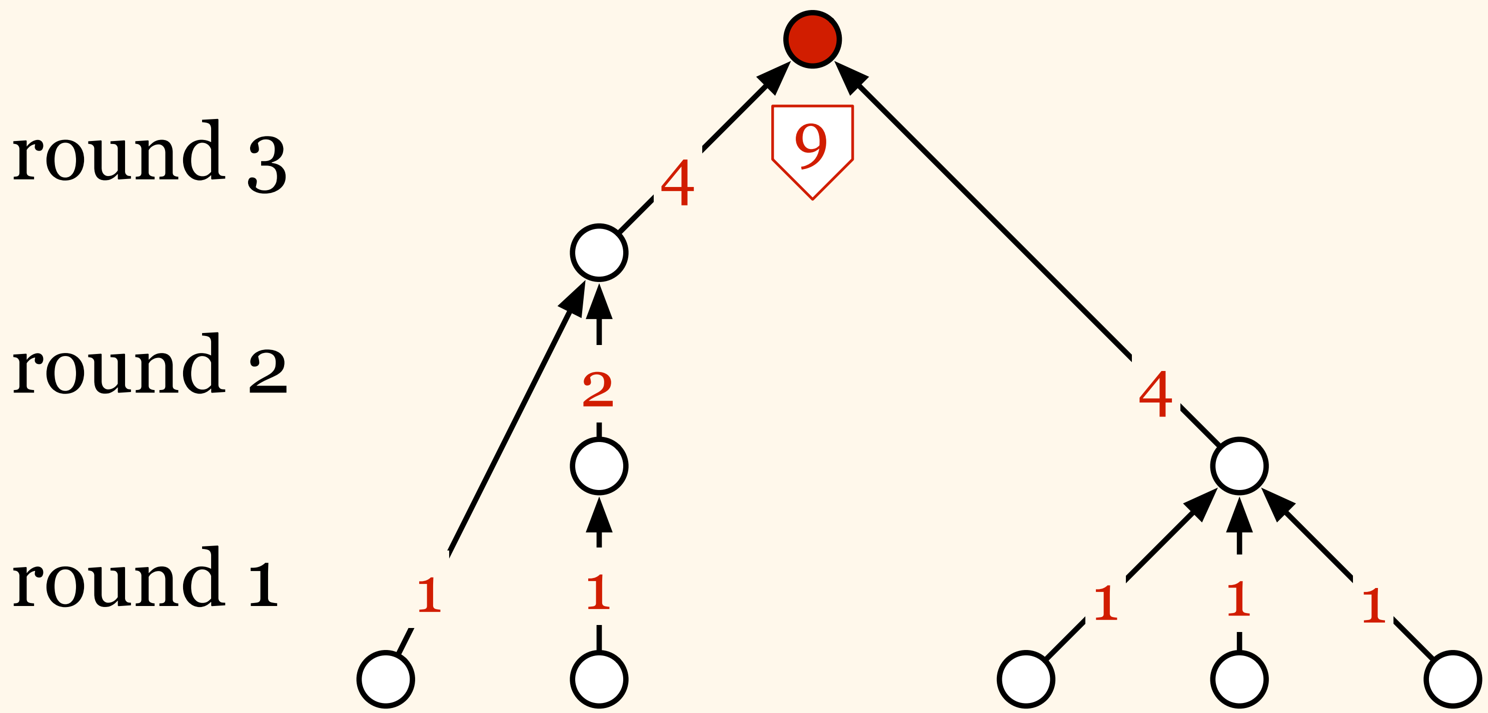
Counting

- Design a distributed algorithm that counts the number of nodes in any *tree*



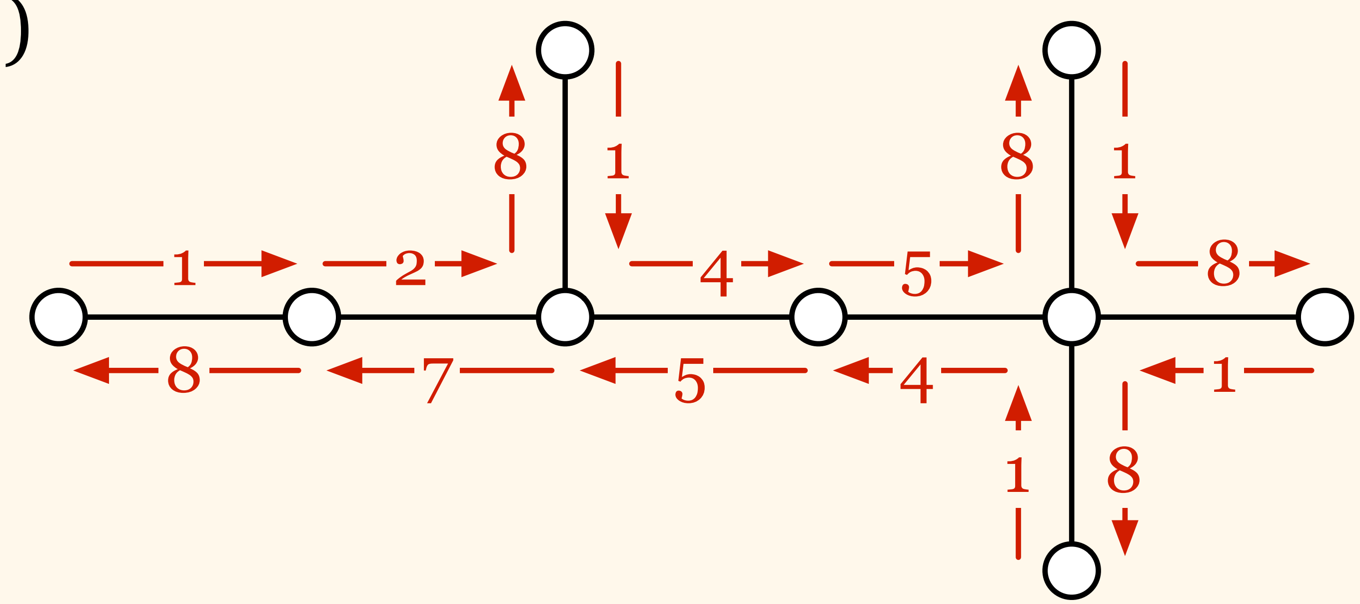

1
1
1
b+c+1
b
2
c
3
a
1
a+1
1
b+1
b
2
a
1
2
a+c+1
c
3
a
1
b
2
a+b+1
a
1
2
a+1
a
1
b
2
3
a+b+1
a
1
b
2
c
3
a+b+c+1




round 3
round 2
round 1
4
9
4
2
1
1
1
1
1

Counting

- Distributed algorithm that counts the number of nodes in any tree
  - same idea: compute *any* property of the tree!
  - time: $O(\mathrm{diam}(G))$



Impossibility

DDA Course

Lecture 3.1

27 March 2012

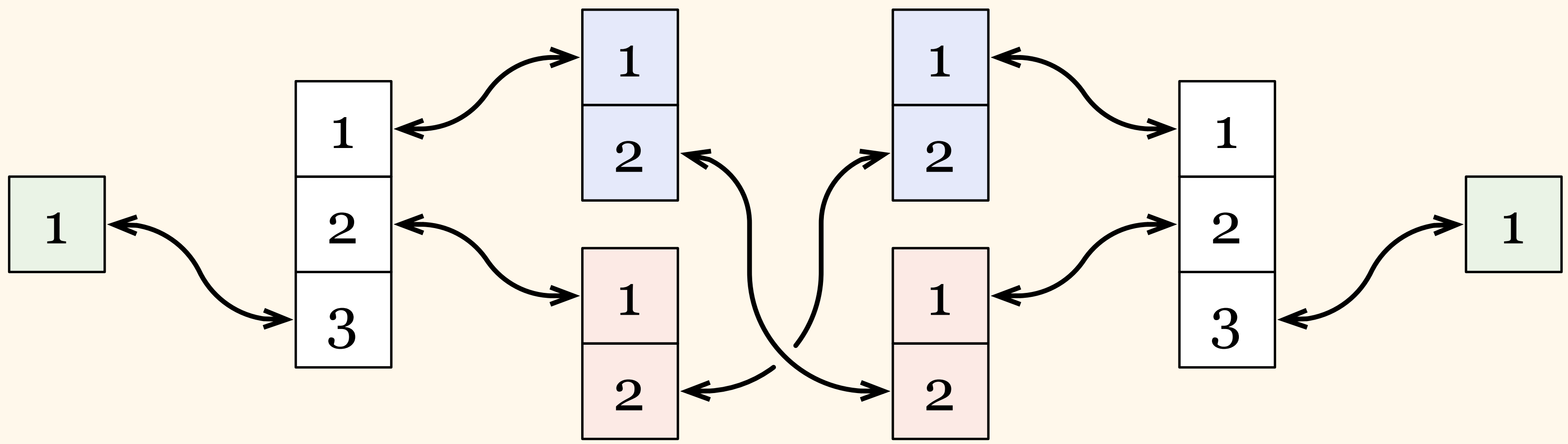
Proof Techniques

- Covering maps
  - problems that cannot solved at all
- Isomorphic local neighbourhoods
  - problems that cannot be solved quickly

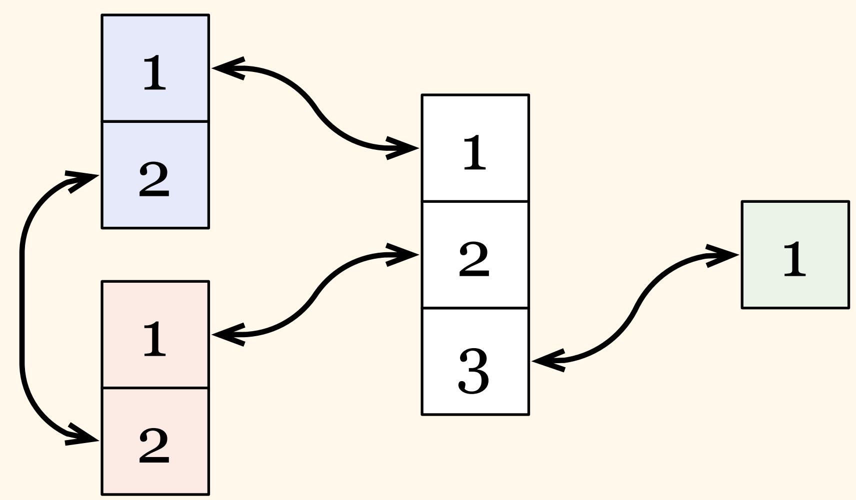
Covering Map

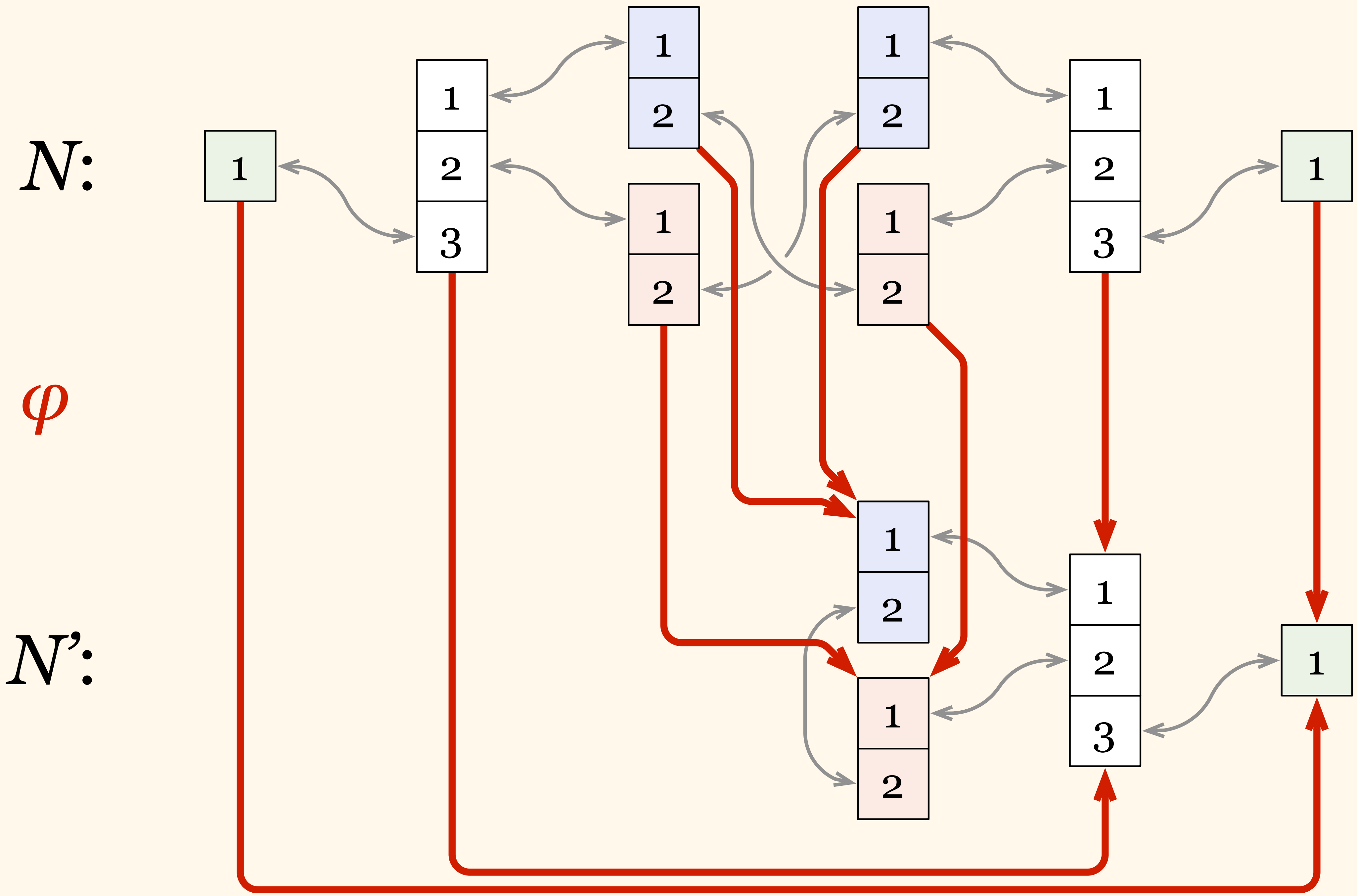
- Networks $N = (V, P, p)$ and $N' = (V', P', p')$
- Surjection φ: $V \to V'$ that *preserves inputs, degrees, connections, and port numbers*

N:

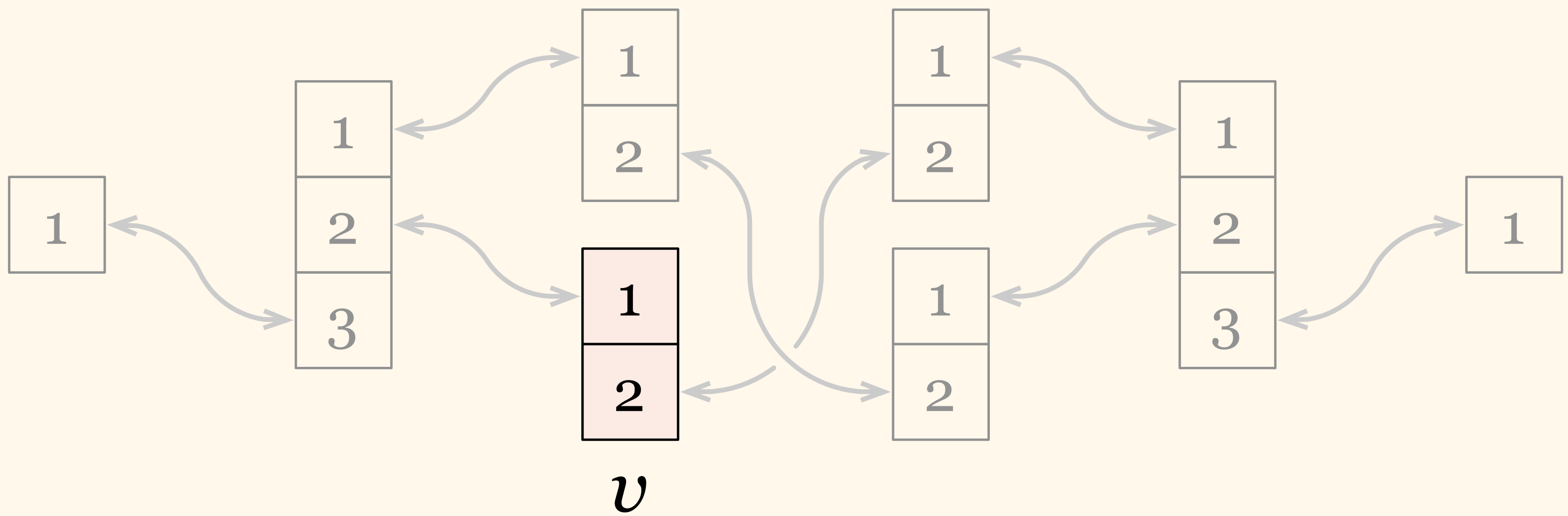


N':



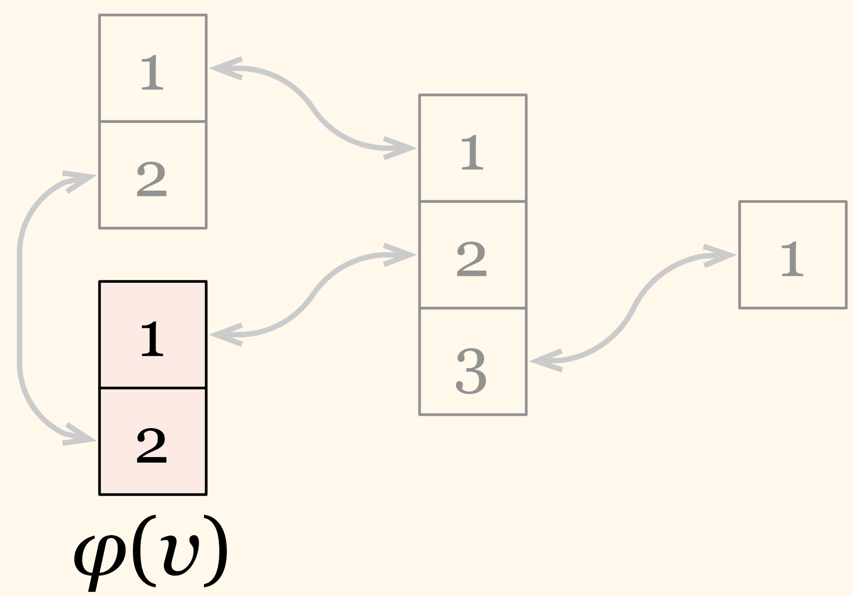

N:
φ
N':
1
1
2
3
1
2
1
2
1
2
1
2
1
2
3
1
1
2
1
2
1
2
3
1

N:



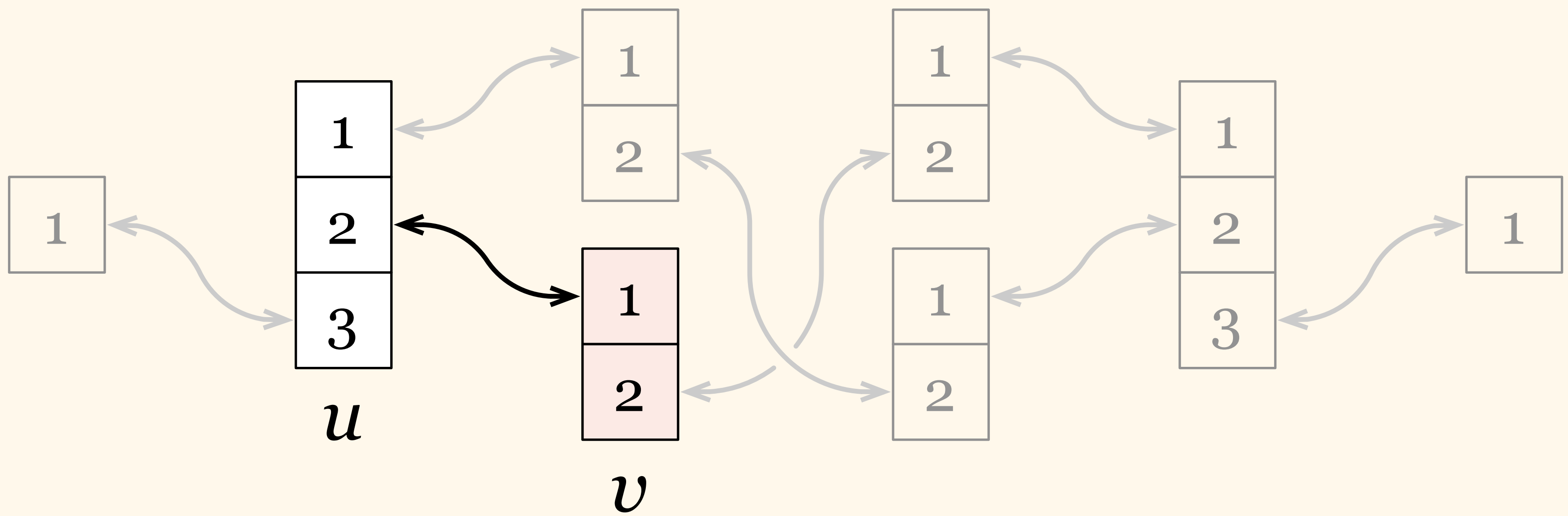
φ

N':



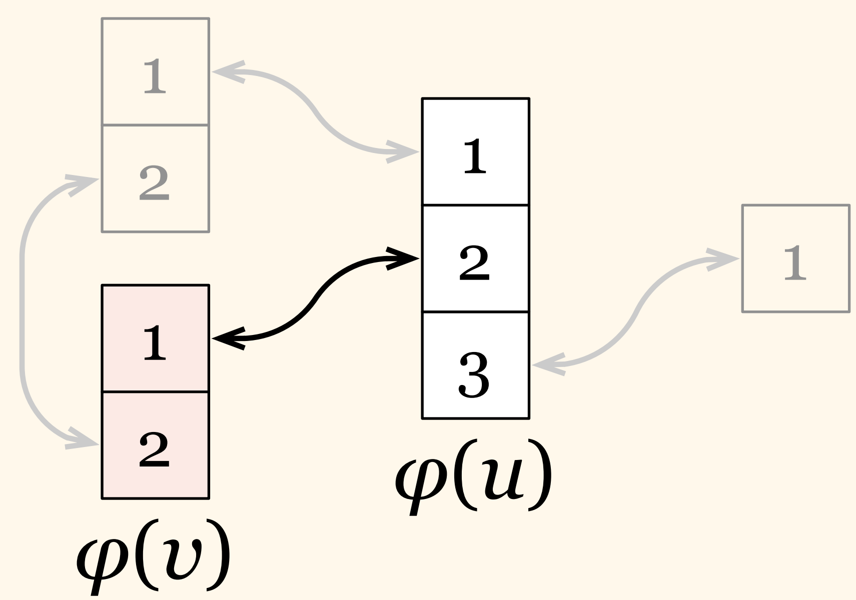
Degrees agree

N:



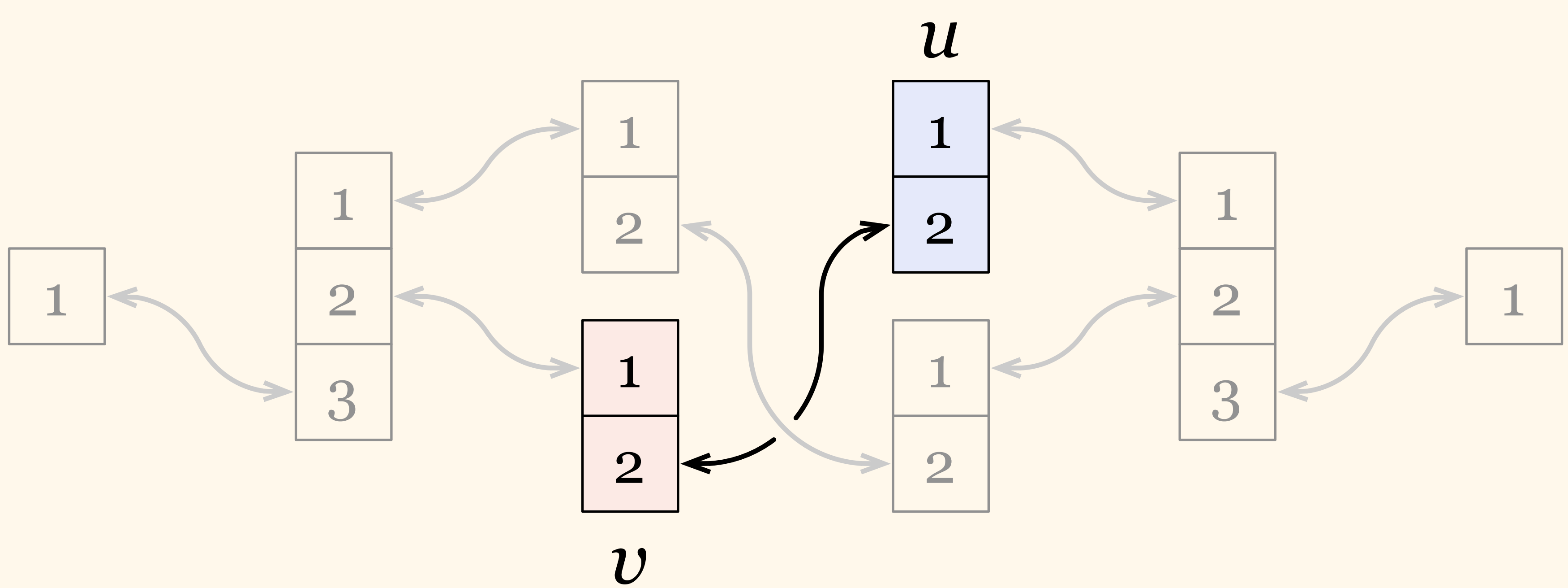
φ

N':

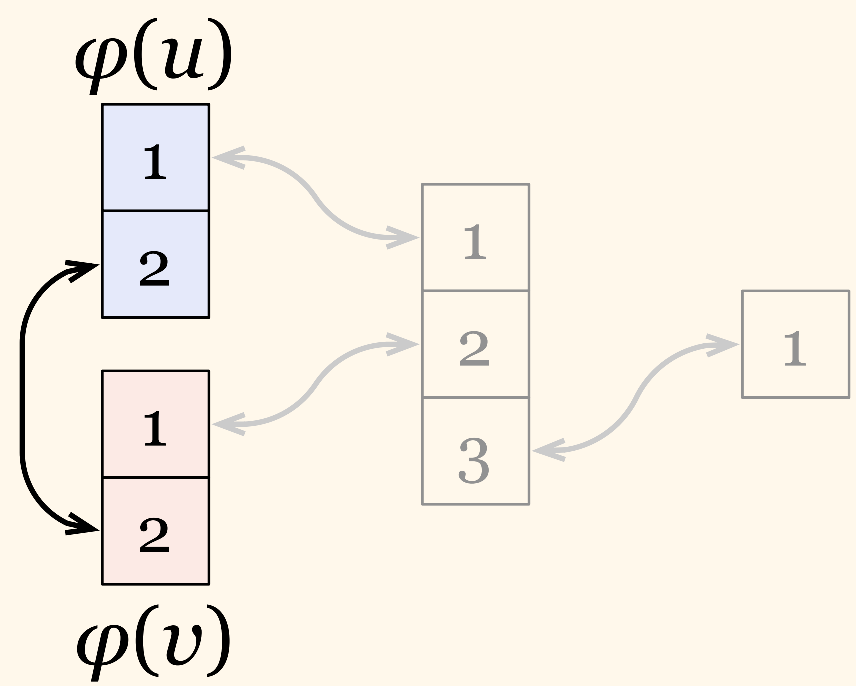


Neighbours in port 1 agree

N:

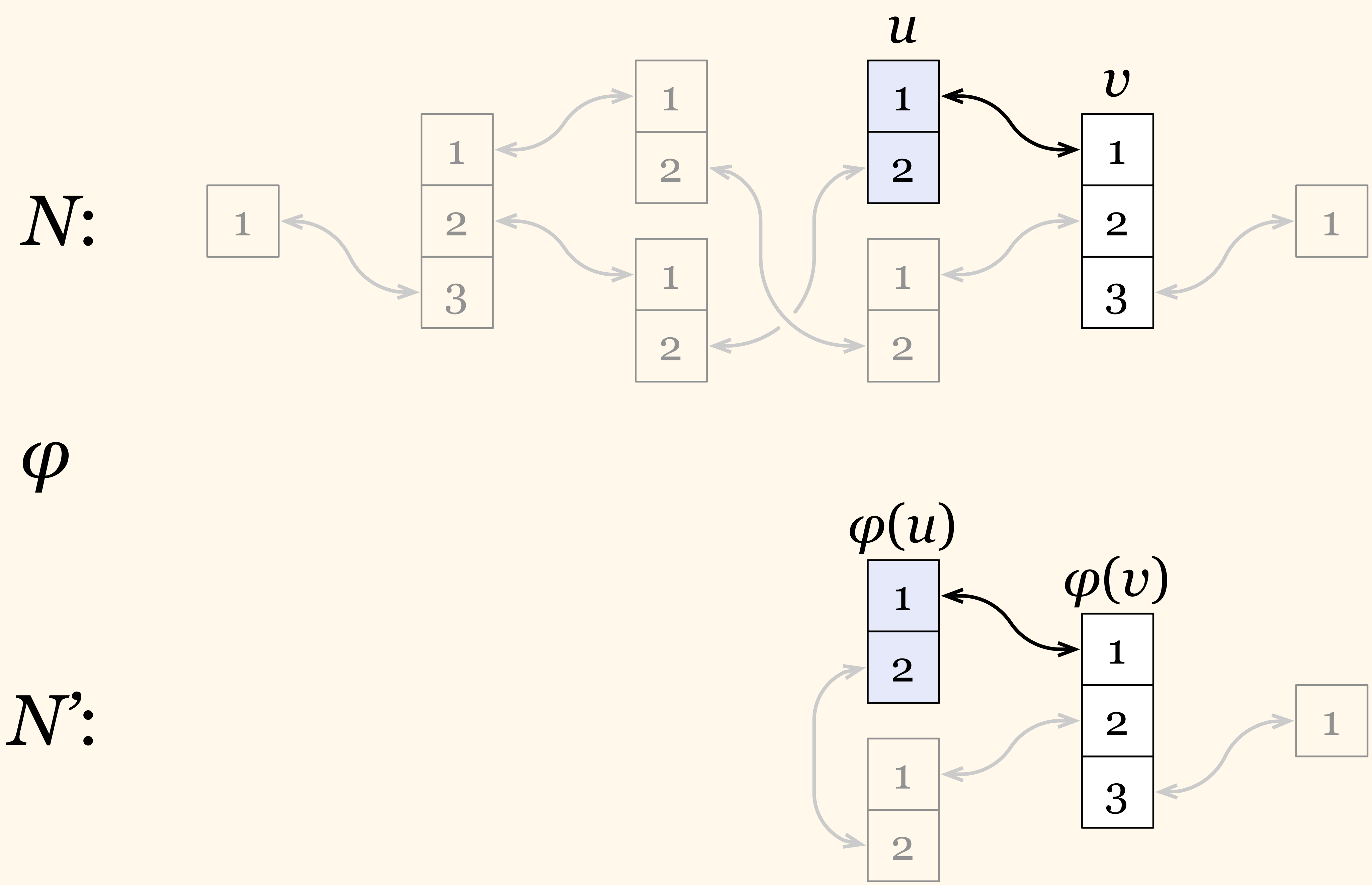


φ



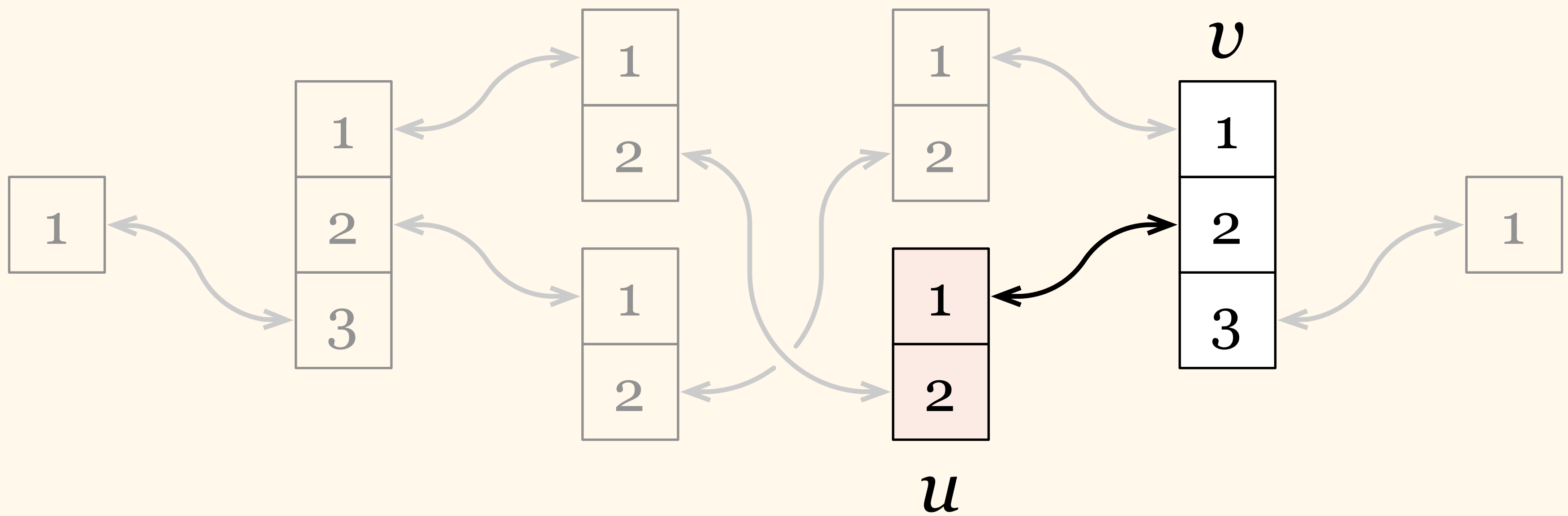
N':

Neighbours in port 2 agree



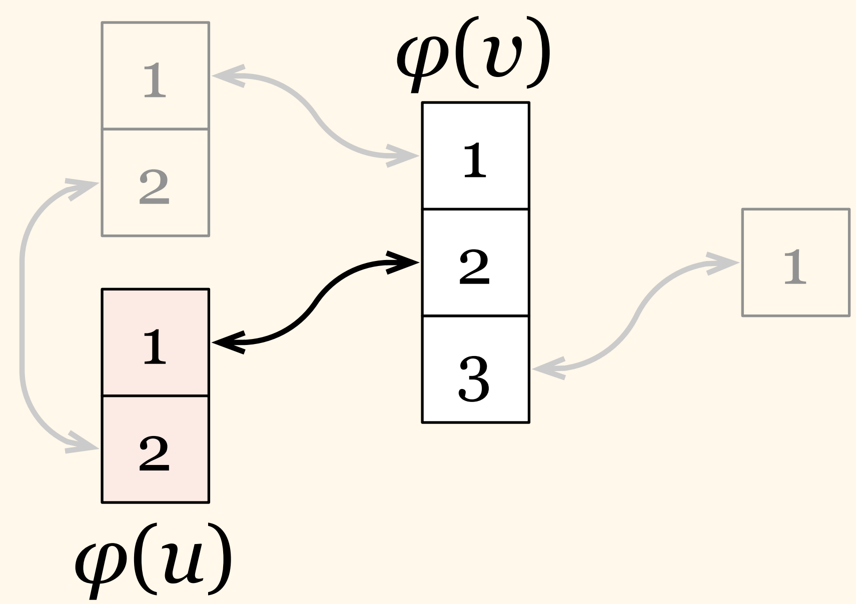
Holds for any pair of nodes

N:



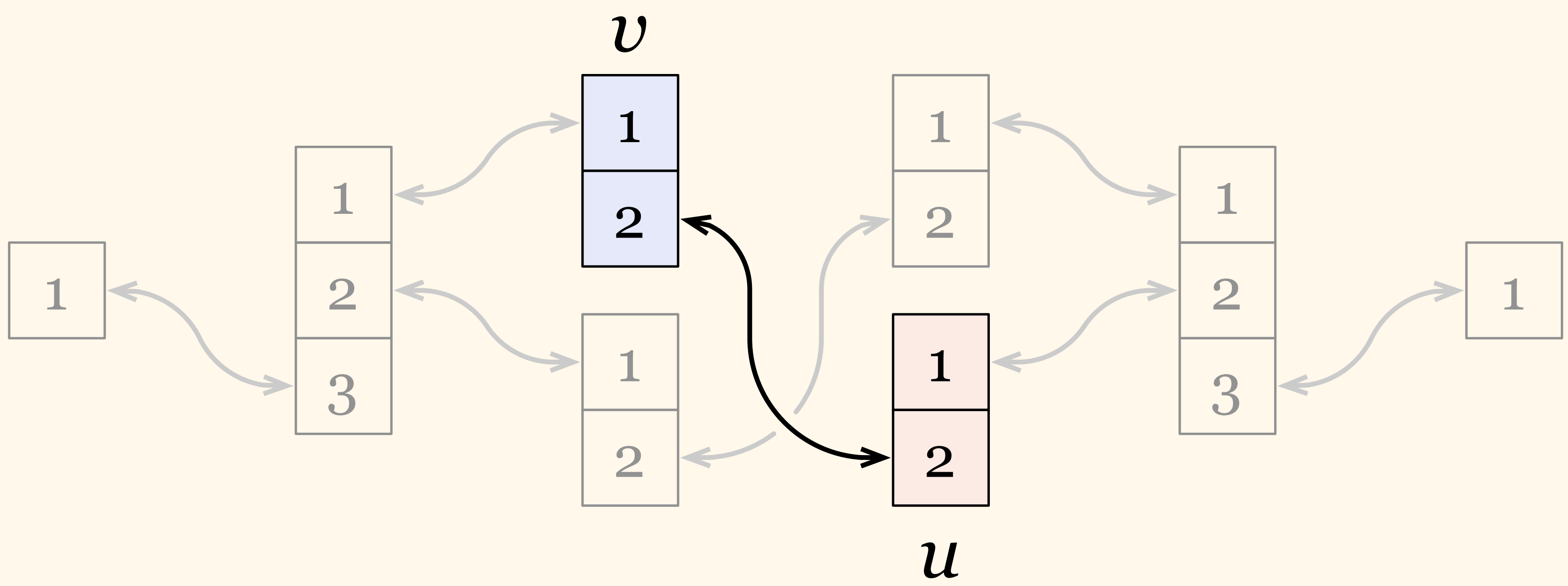
φ

N':



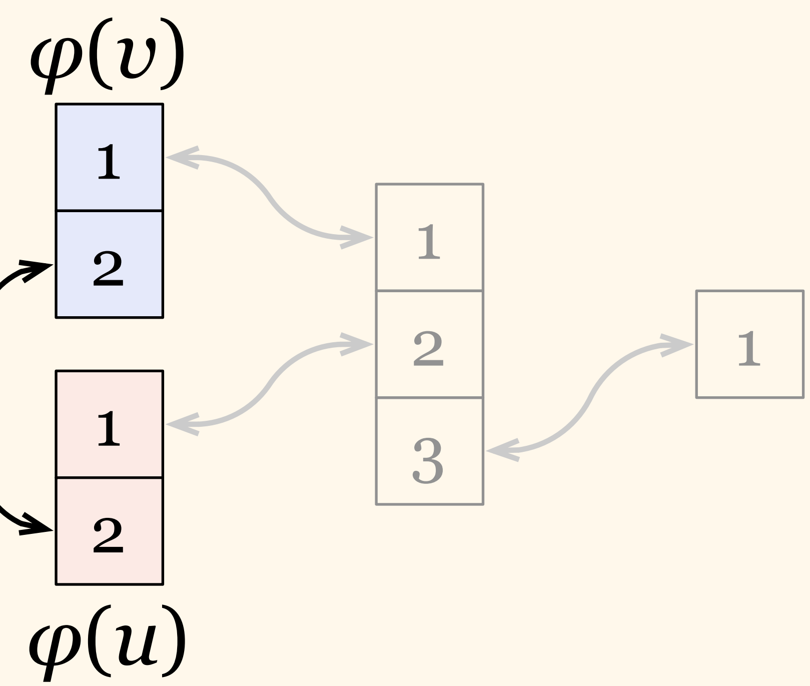
Holds for any pair of nodes

N:



φ

N':

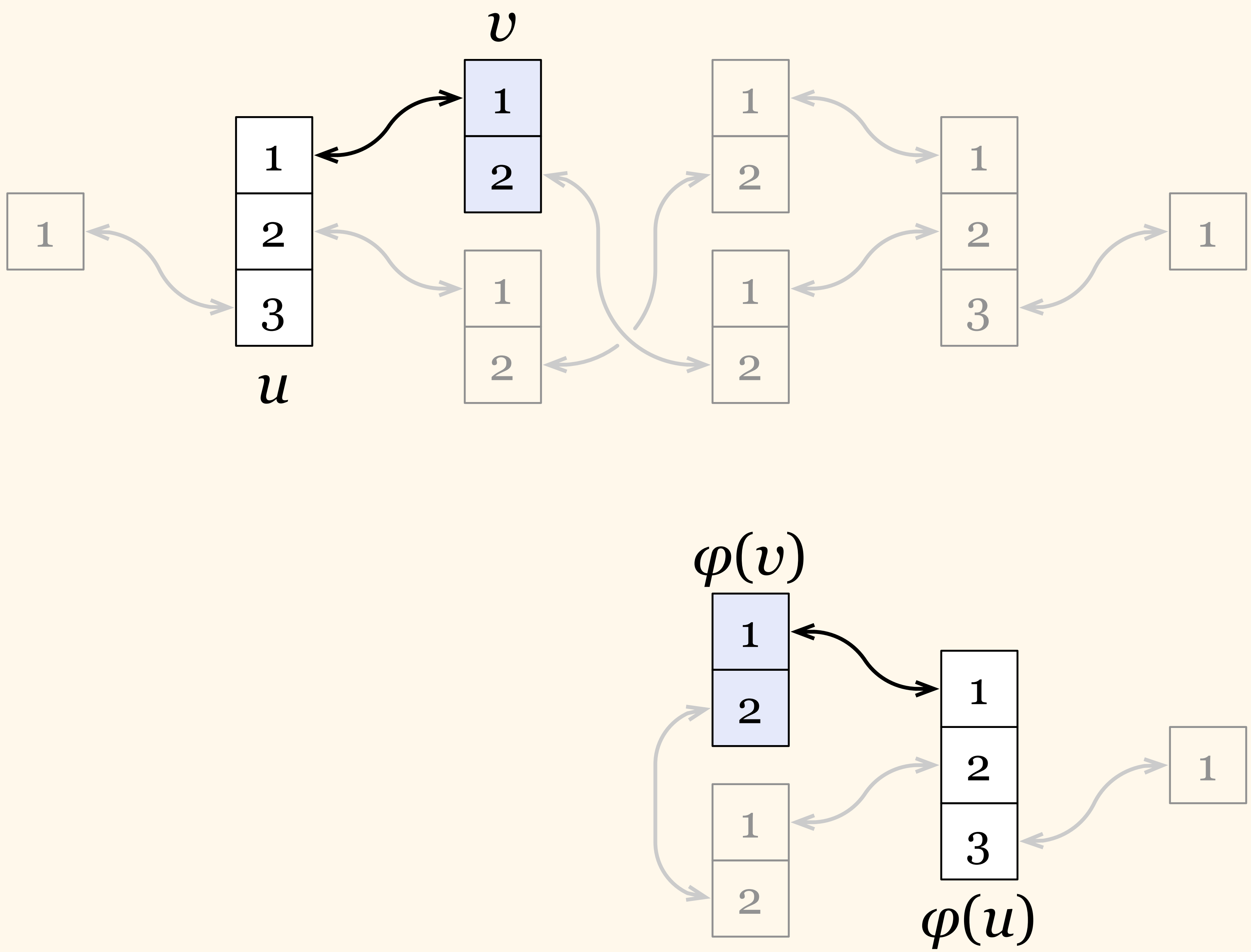


Holds for any pair of nodes

N:

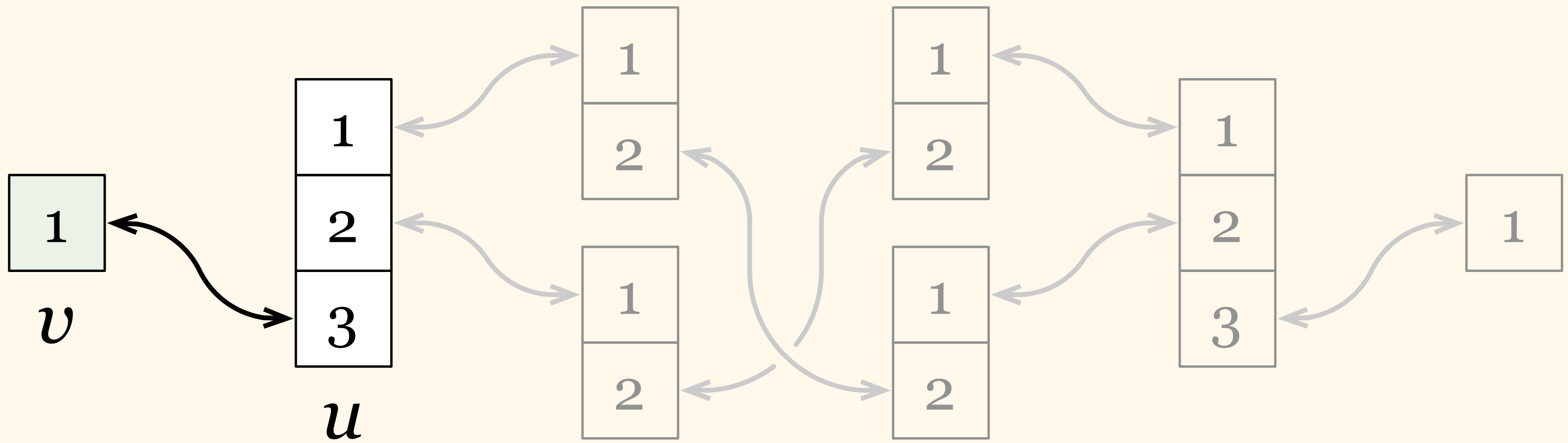
φ

N':



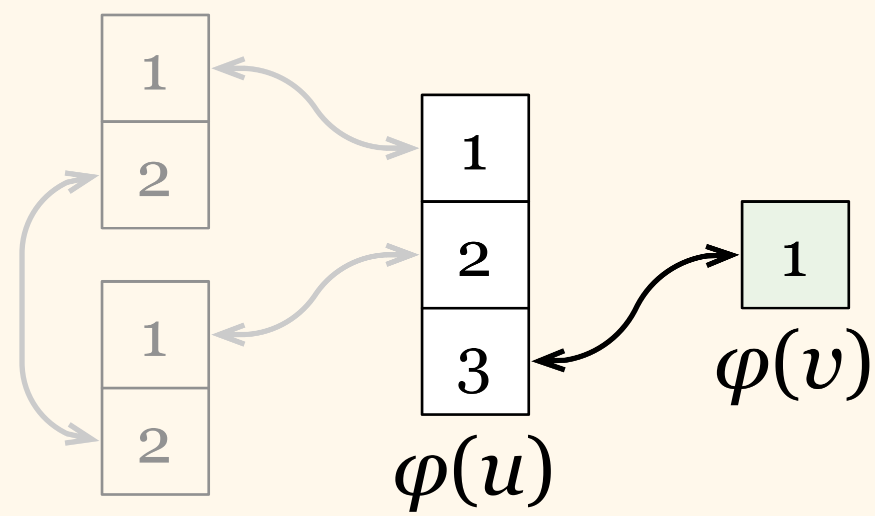
Holds for any pair of nodes

N:



φ

N':



Holds for any pair of nodes

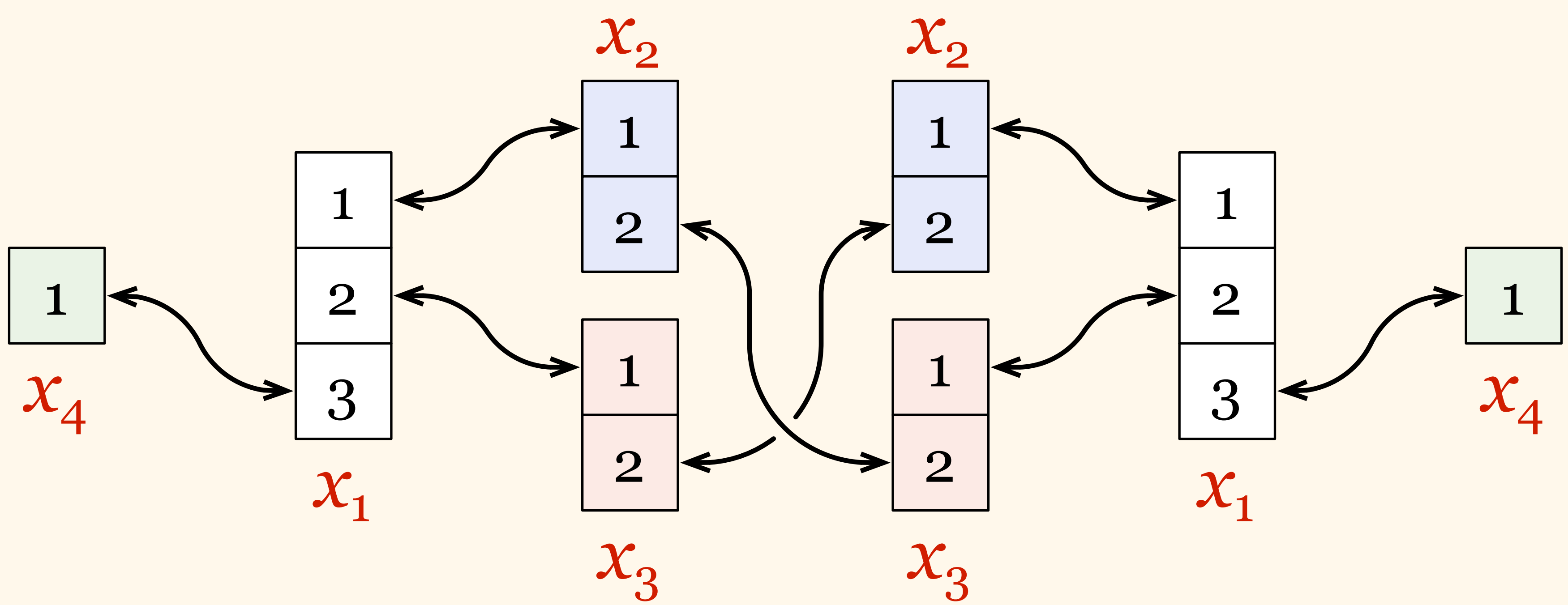
Covering Map

- Networks $N = (V, P, p)$ and $N' = (V', P', p')$
- Surjection φ: $V \to V'$ that preserves inputs, degrees, connections, and port numbers
- **Theorem**: If we run an algorithm A in N and N', then nodes v and $\varphi(v)$ are *always in the same state*

Covering Map

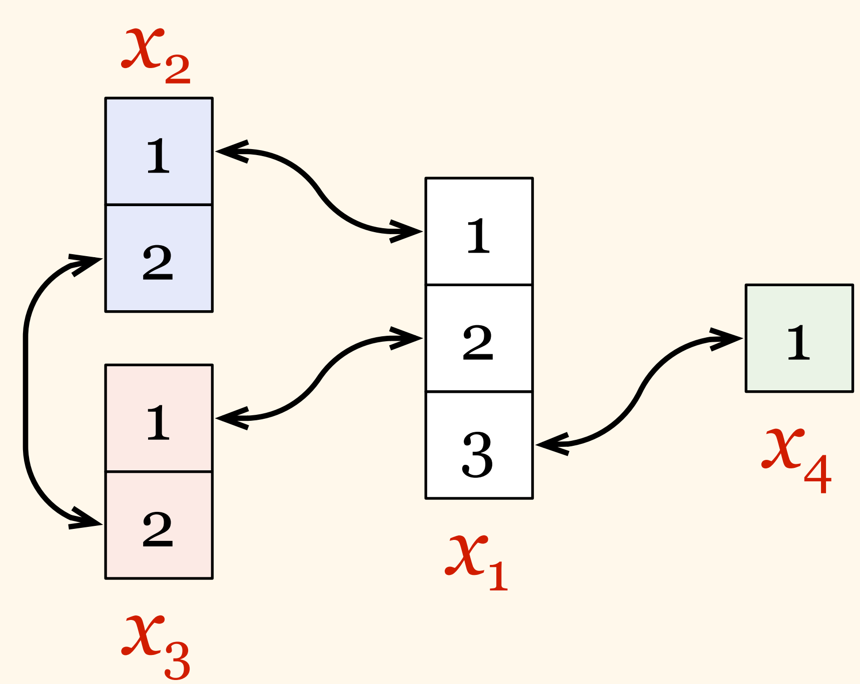
- **Theorem**: If we run an algorithm A in N and N', then nodes v and $\varphi(v)$ are *always in the same state*
- **Proof**: By induction
  - before round i: map φ preserves local states
  - during round i: map φ preserves messages
  - after round i: map φ preserves local states

N:



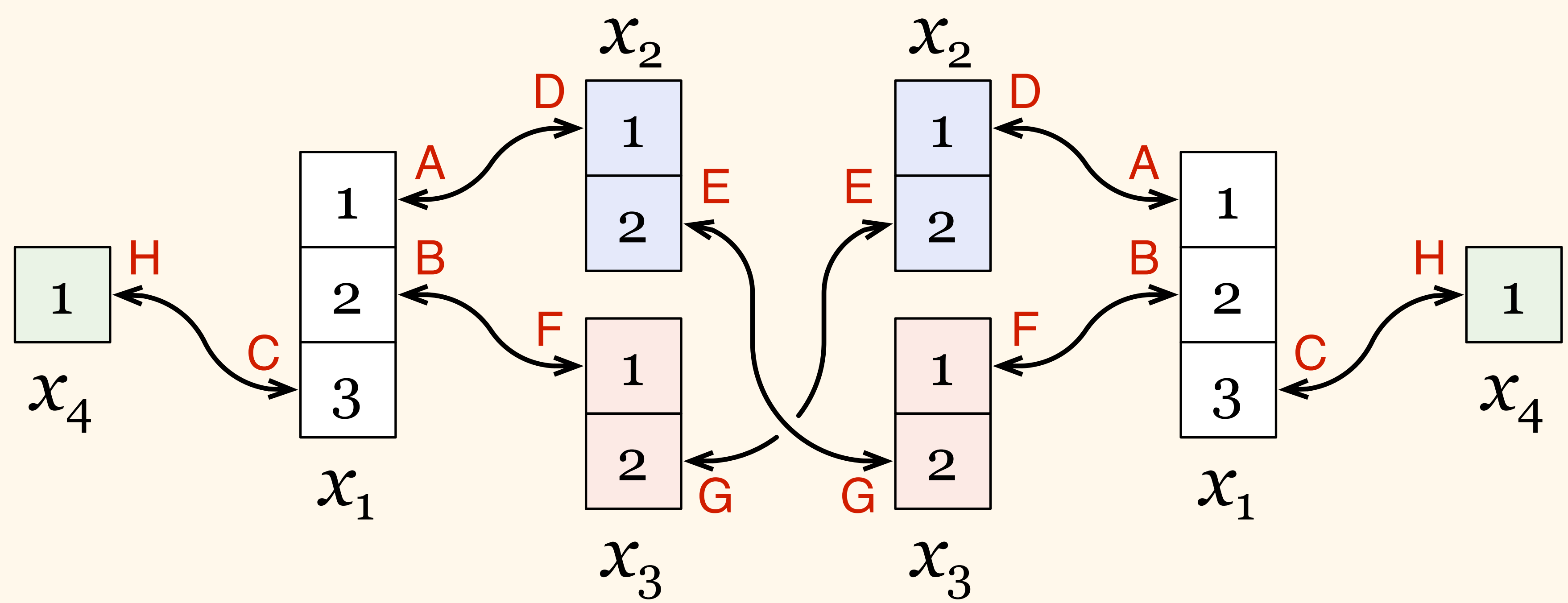
φ

N':



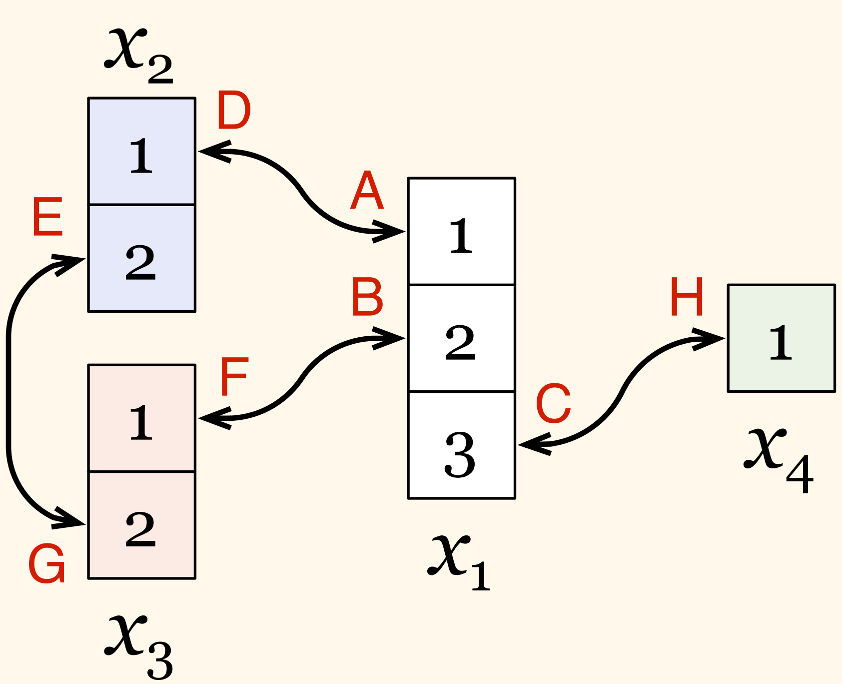
Initially, *local states* agree

N:



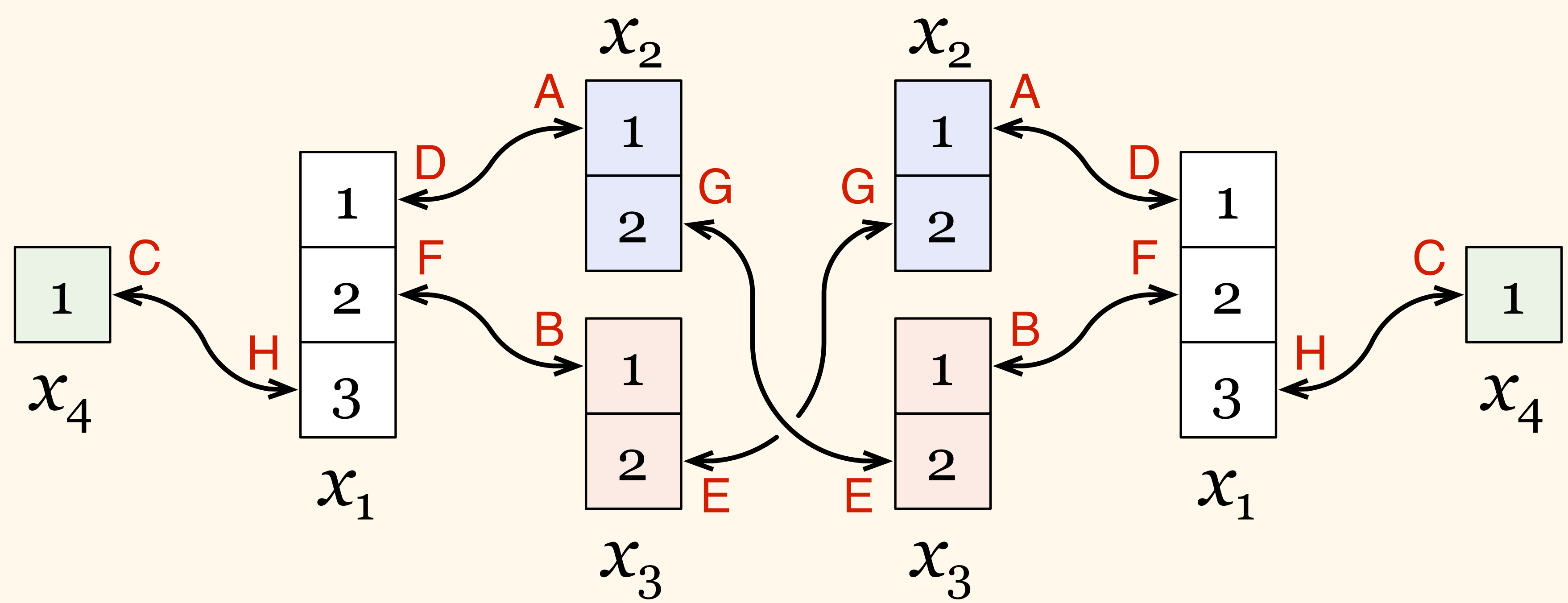
φ

N':



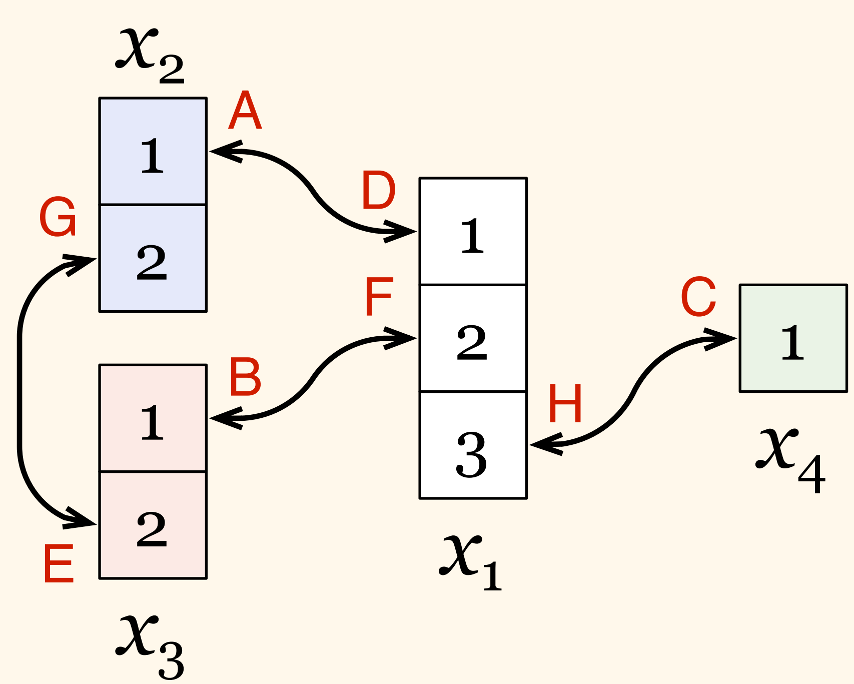
Thus *outgoing messages* agree

N:



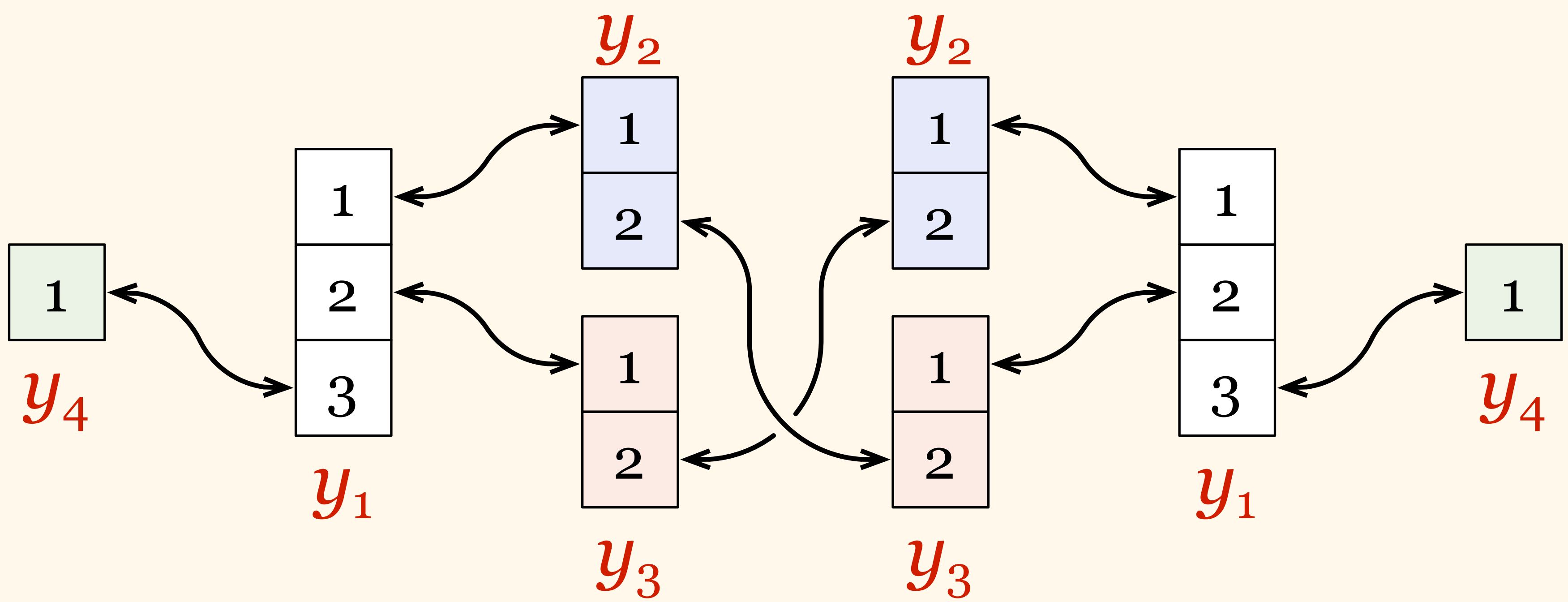
φ

N':



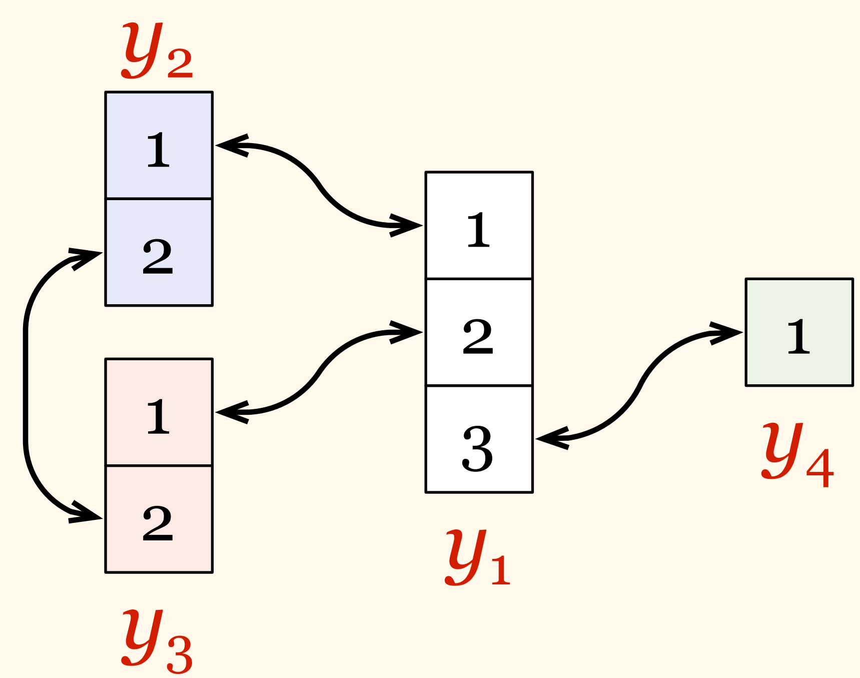
Thus *incoming messages* agree

N:



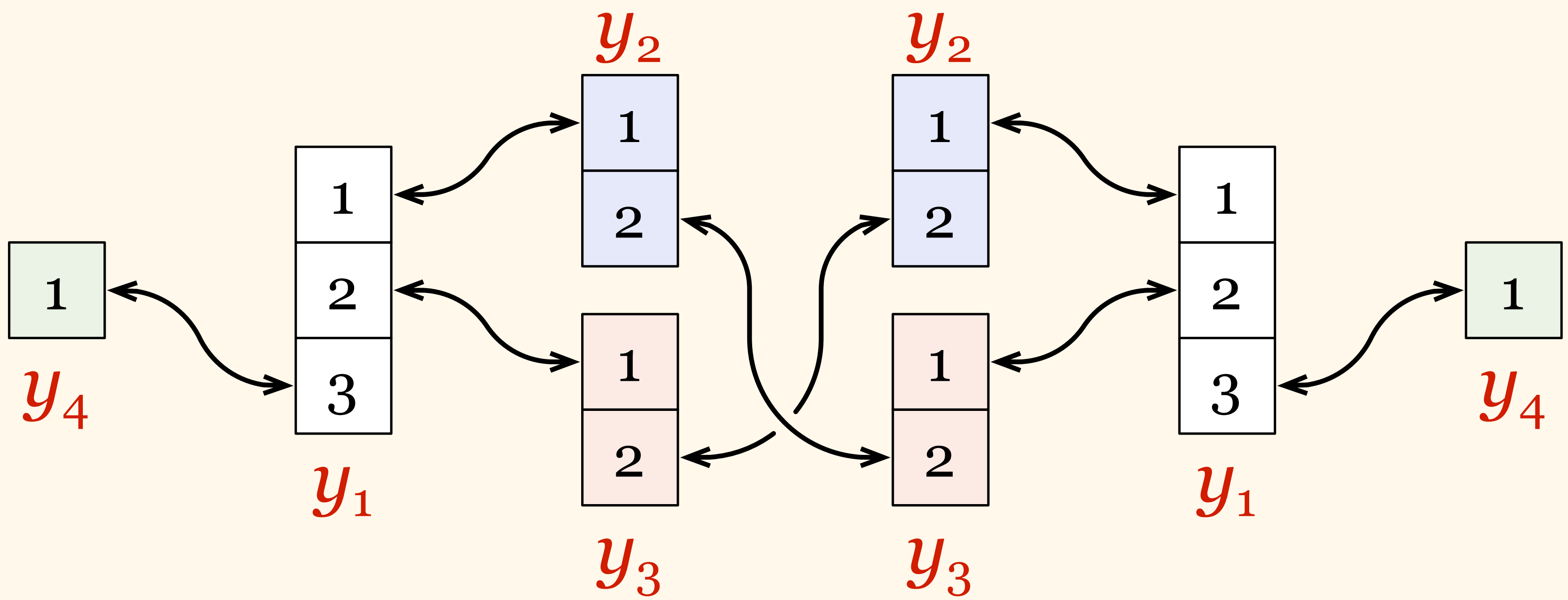
φ

N':



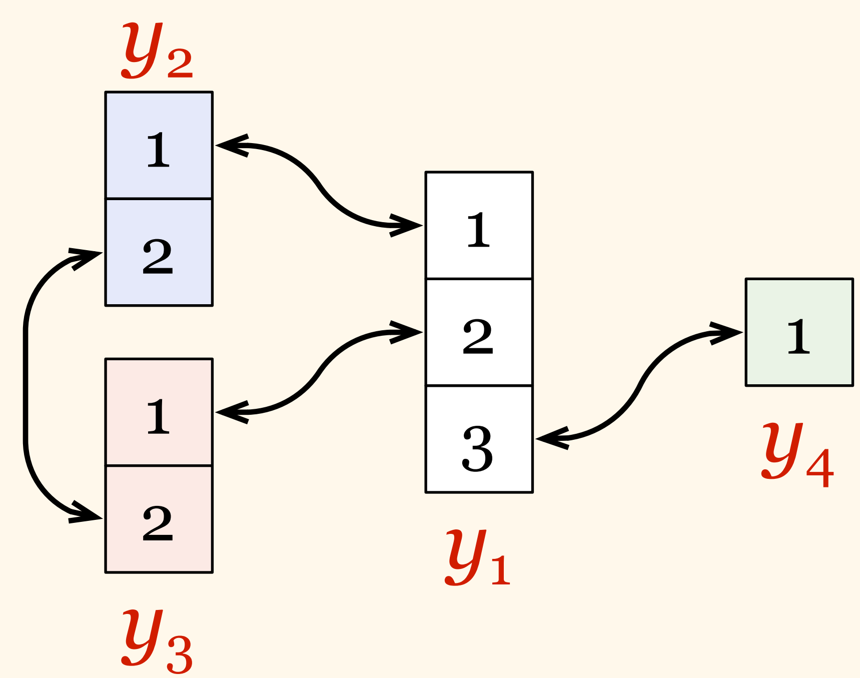
Thus *new local states* agree

N:



φ

N':

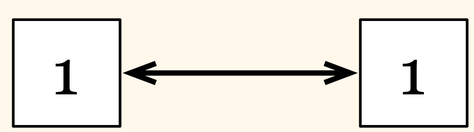


By induction, *local outputs* agree

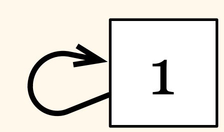
Covering Map

- Application: symmetry breaking in a path graph

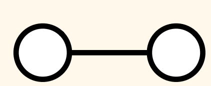
N:



N':

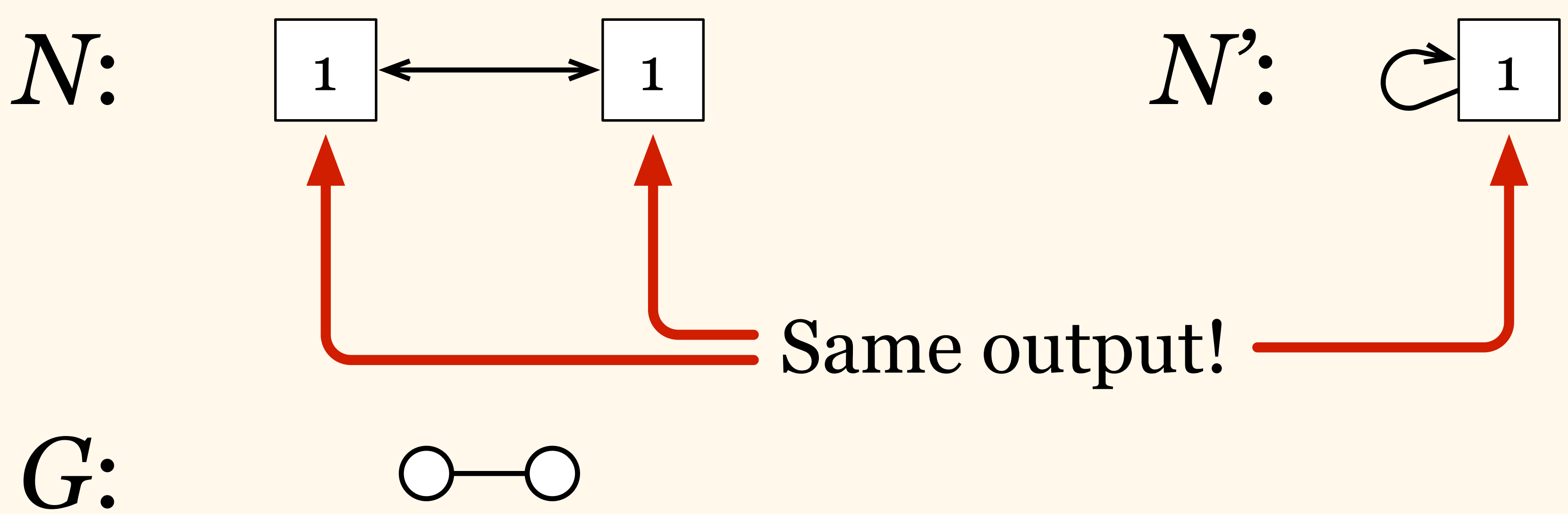


G:



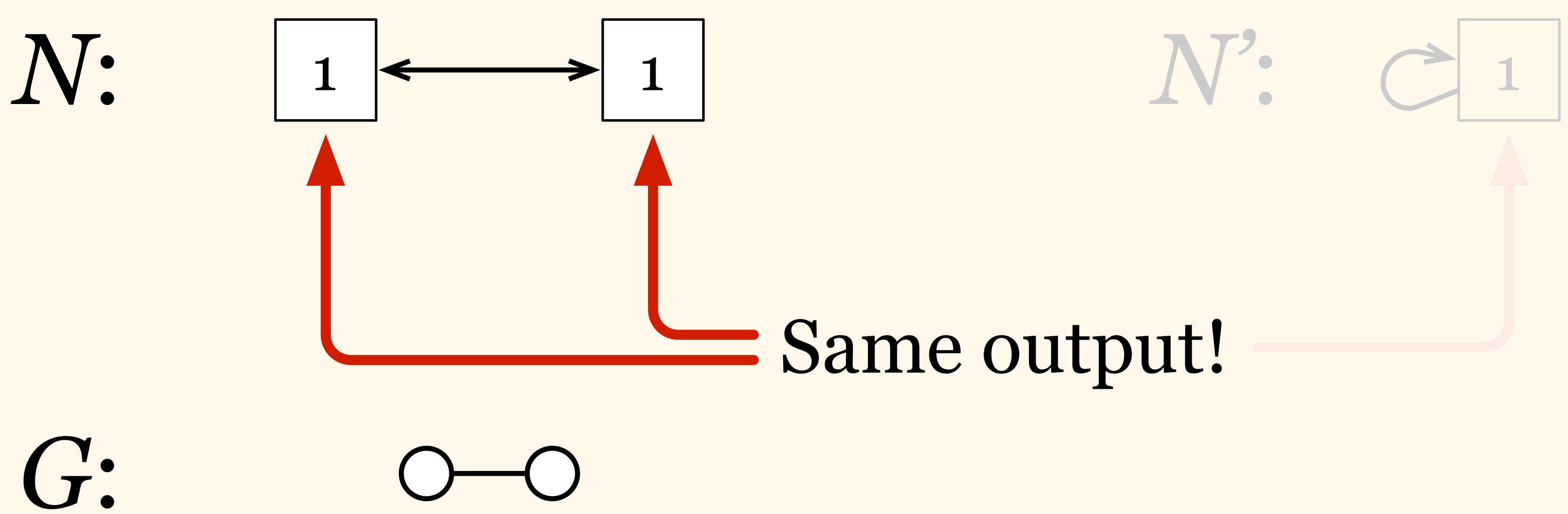
Covering Map

- Application: symmetry breaking in a path graph



Covering Map

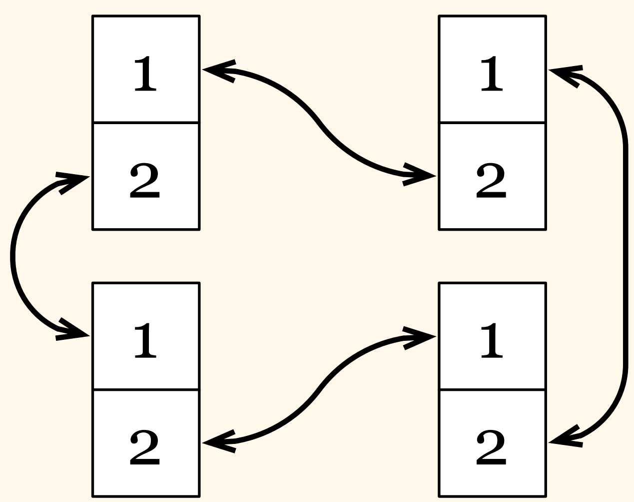
- Application: symmetry breaking in a path graph



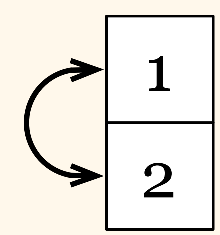
Covering Map

- Application: symmetry breaking in a cycle

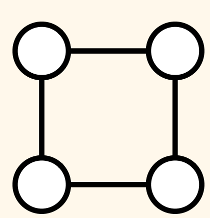
N:



N':



G:

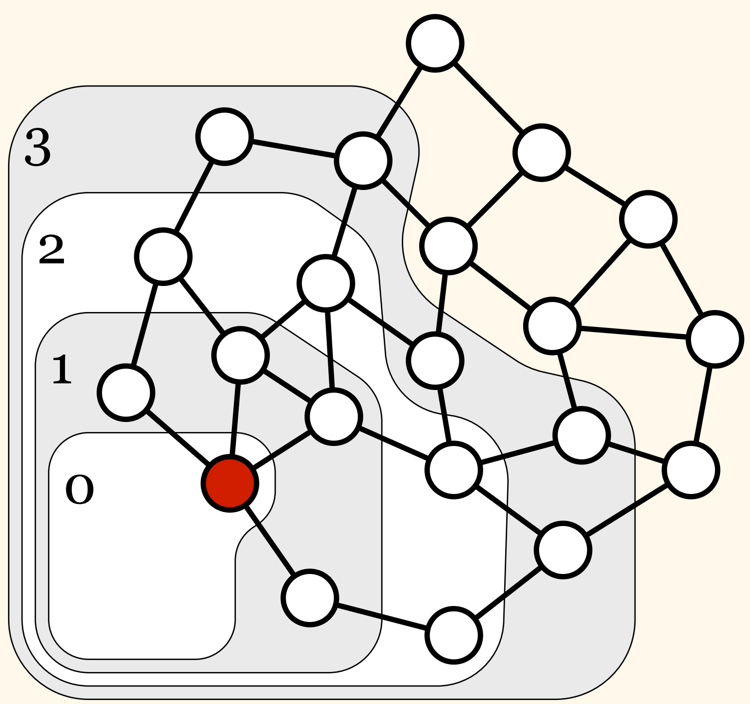


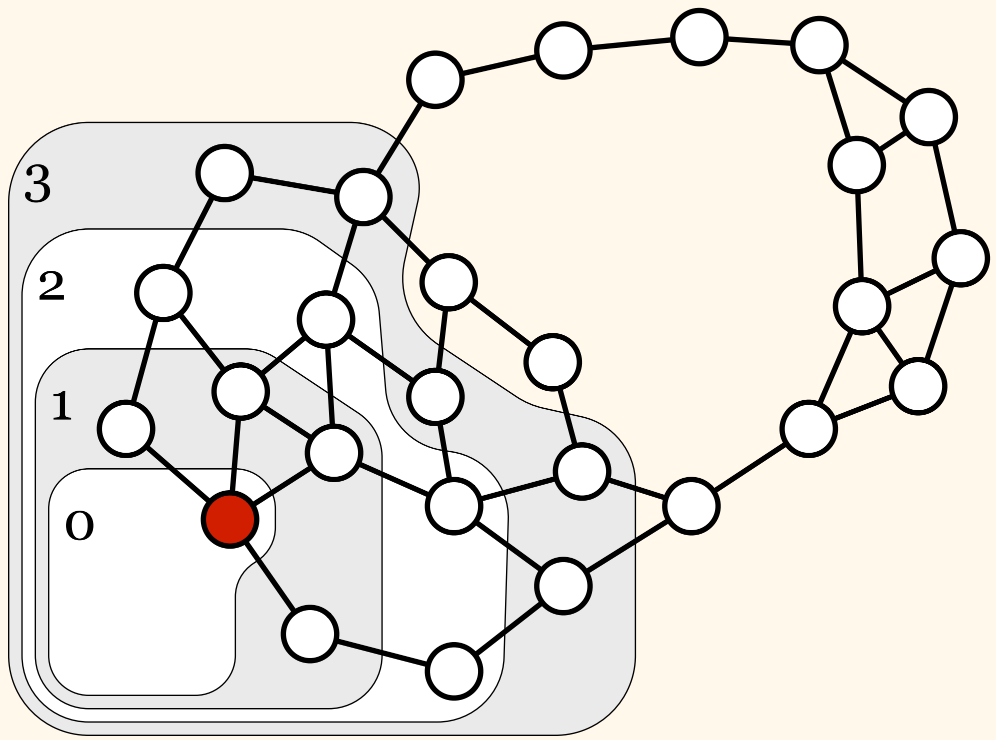
Local Neighbourhoods

- Local neighbourhoods of nodes u and v "look identical" up to distance r
  - isomorphism between radius-r neighbourhood of u and radius-r neighbourhood of v
  - preserves *inputs, degrees, connections, and port numbers*

Local Neighbourhoods

- Local neighbourhoods of nodes u and v “look identical” up to distance r



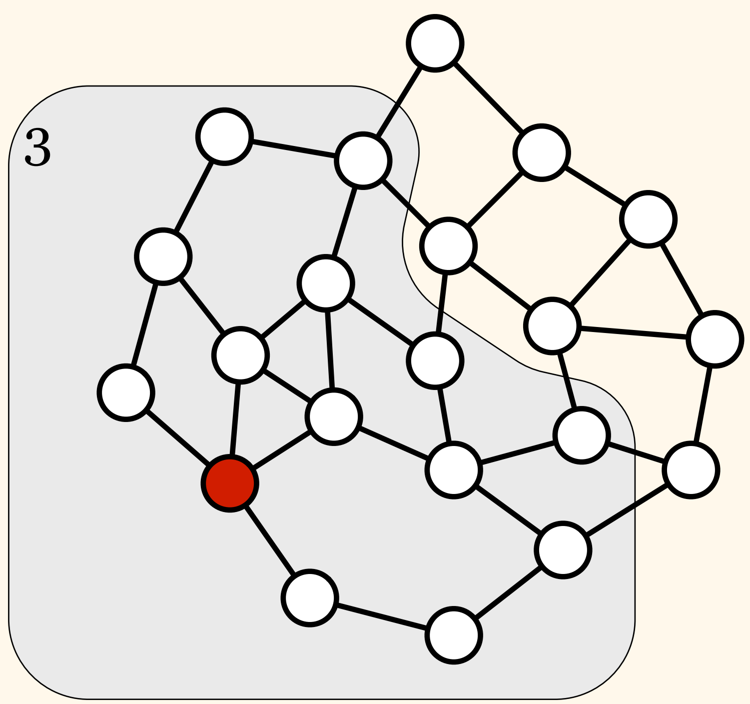


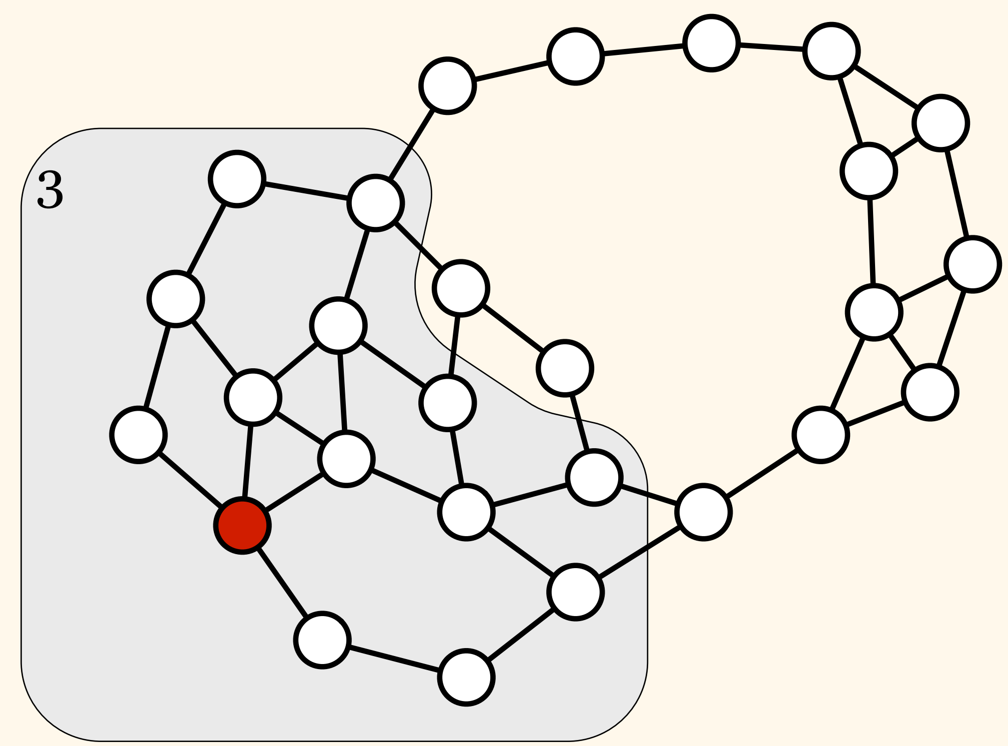
Local Neighbourhoods

- Local neighbourhoods of nodes u and v "look identical" up to distance r
- **Theorem**: In any algorithm, up to time r, the local states of u and v are identical
- *Informal proof*: time $\approx$ distance
- *Formal proof*: by induction on time

Local Neighbourhoods

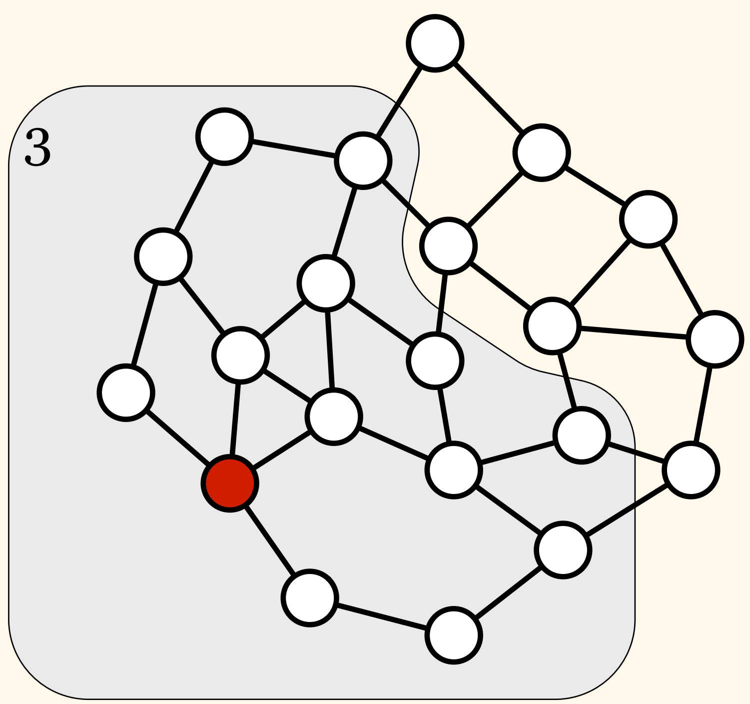
- Time 0: identical *local states* in radius-r neighbourhoods

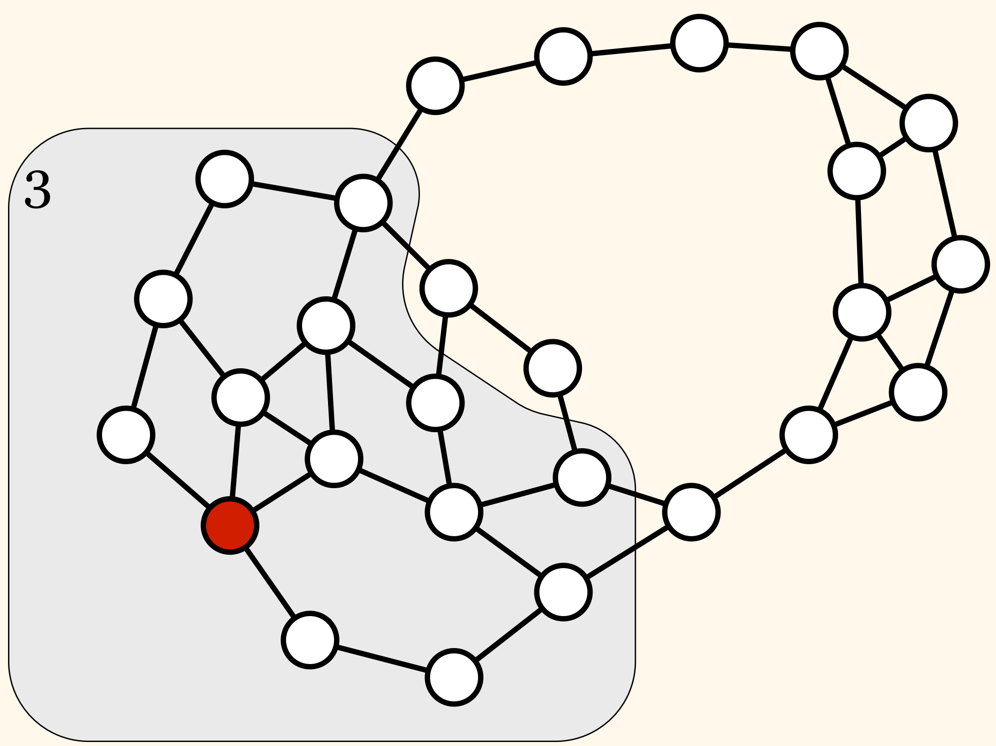




Local Neighbourhoods

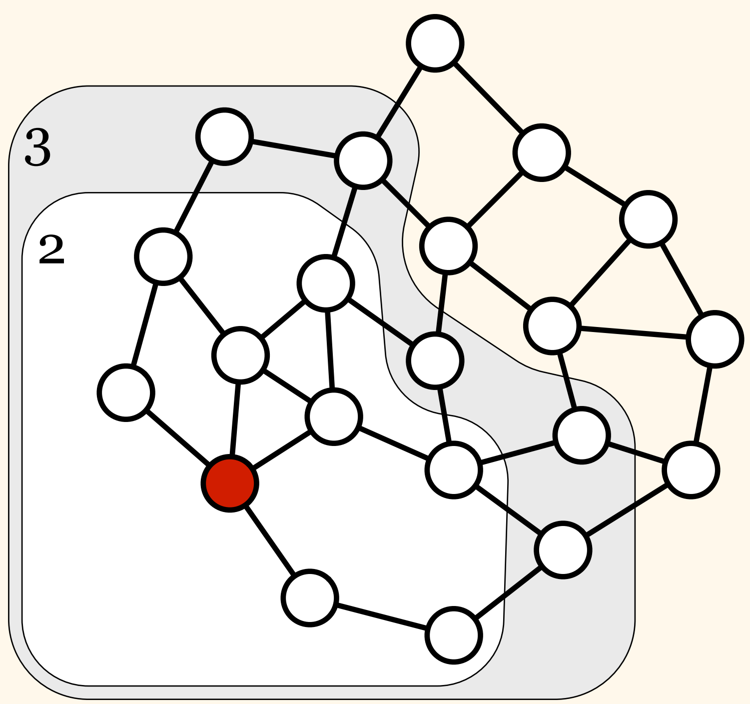
- Time 1: identical *outgoing messages* in radius-r neighbourhoods

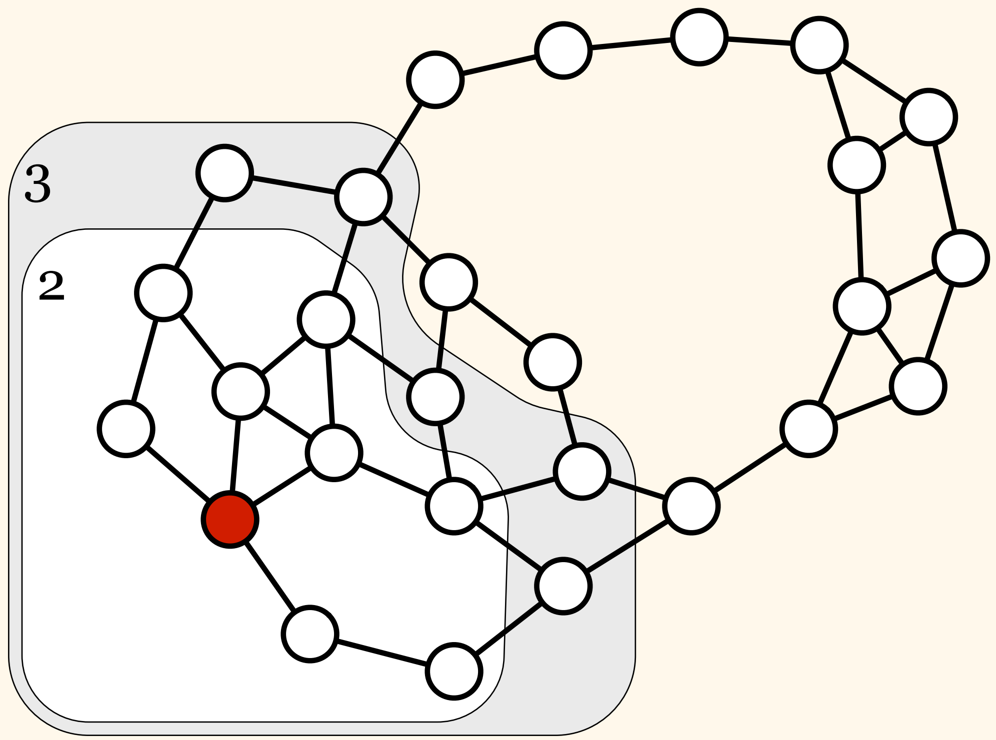




Local Neighbourhoods

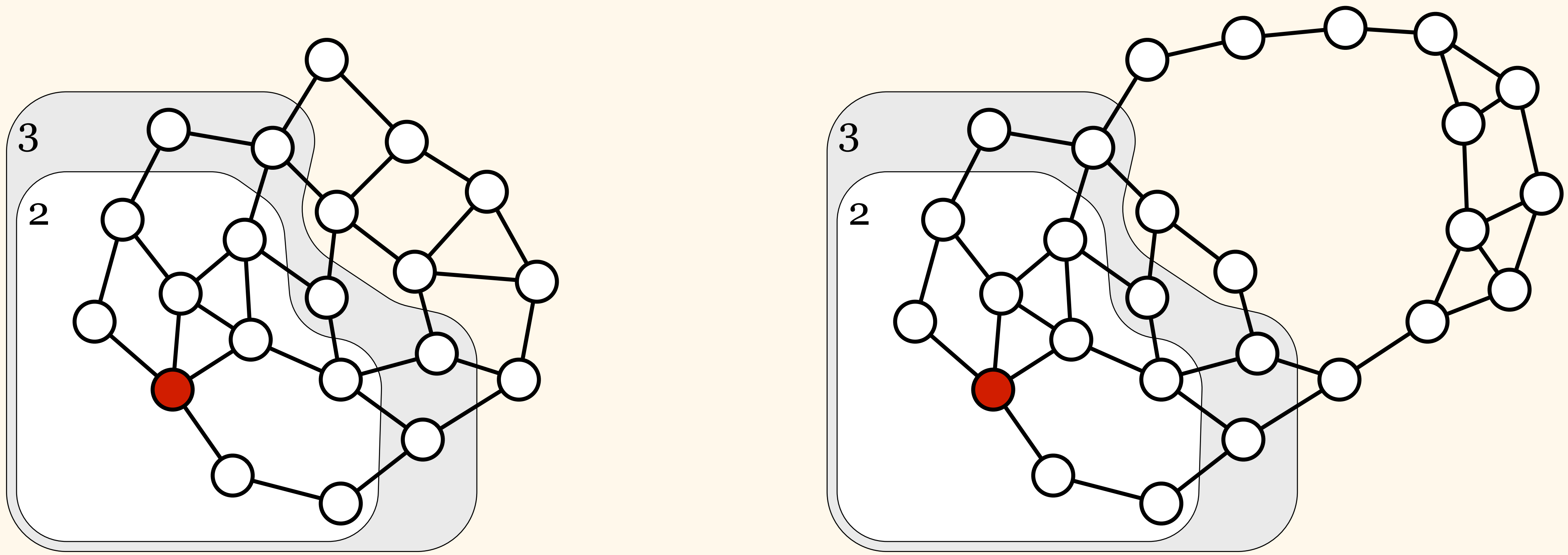
- Time 1: identical *incoming messages* in radius-(r–1) neighbourhoods





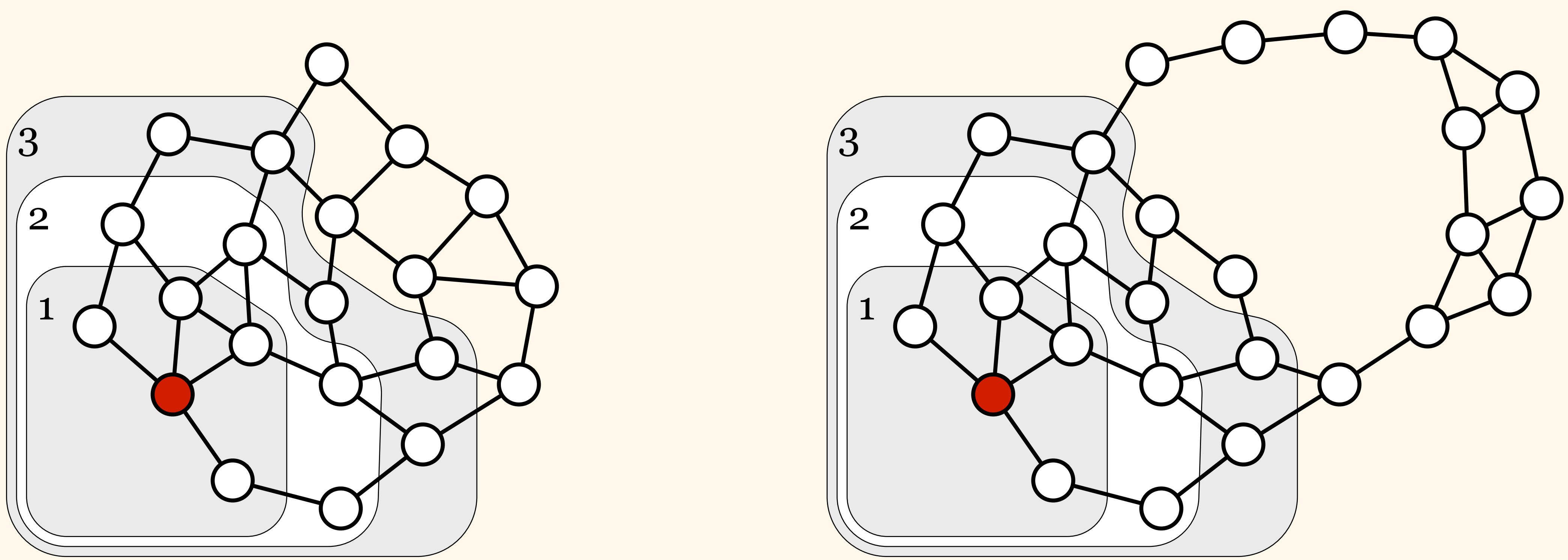
Local Neighbourhoods

- Time 1: identical *local states* in radius-(r–1) neighbourhoods



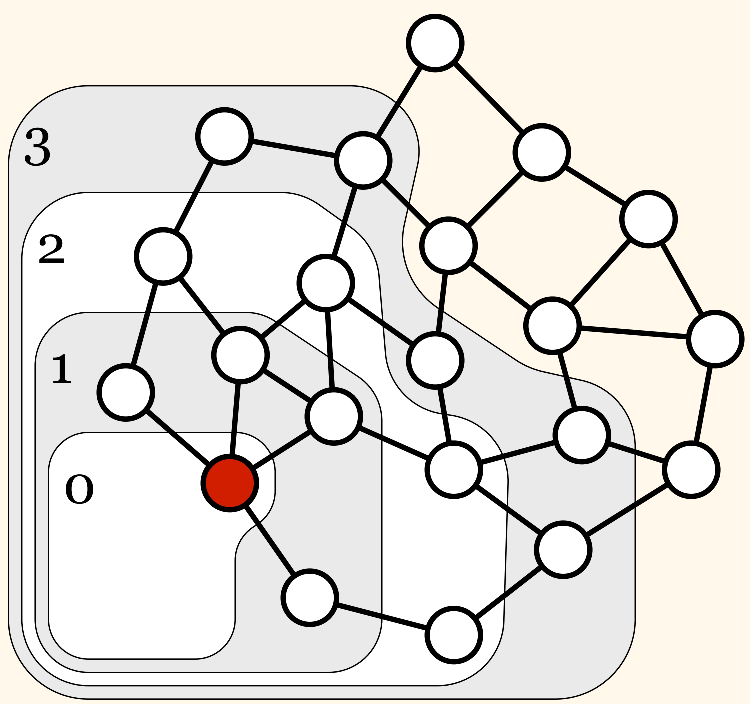
Local Neighbourhoods

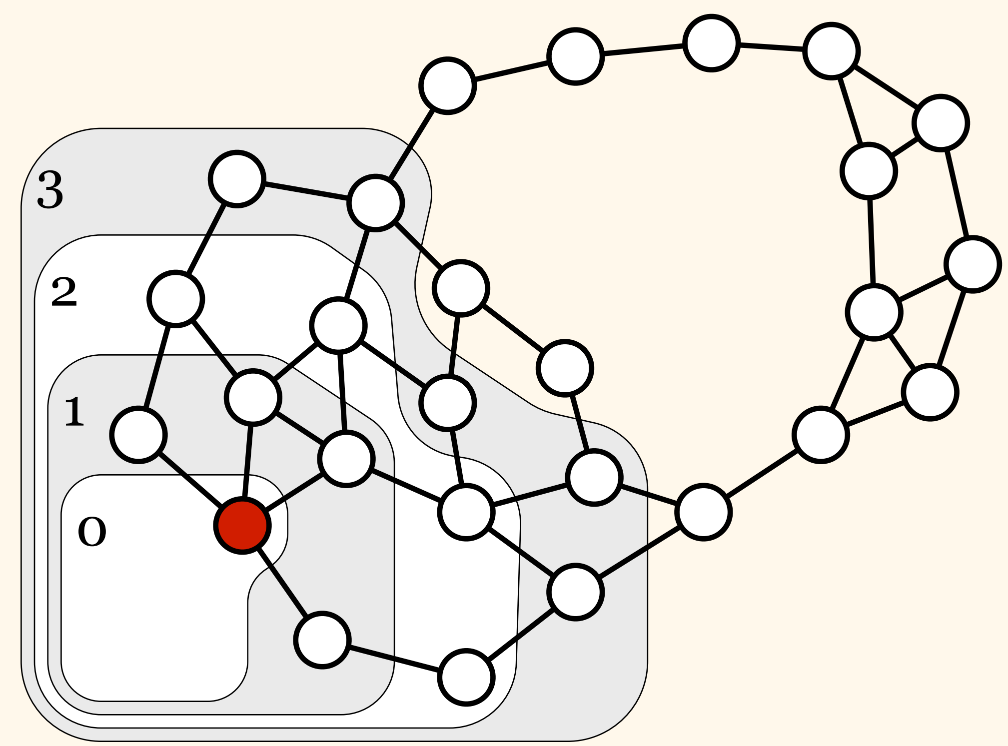
- Time t: identical *local states* in radius-(r–t) neighbourhoods



Local Neighbourhoods

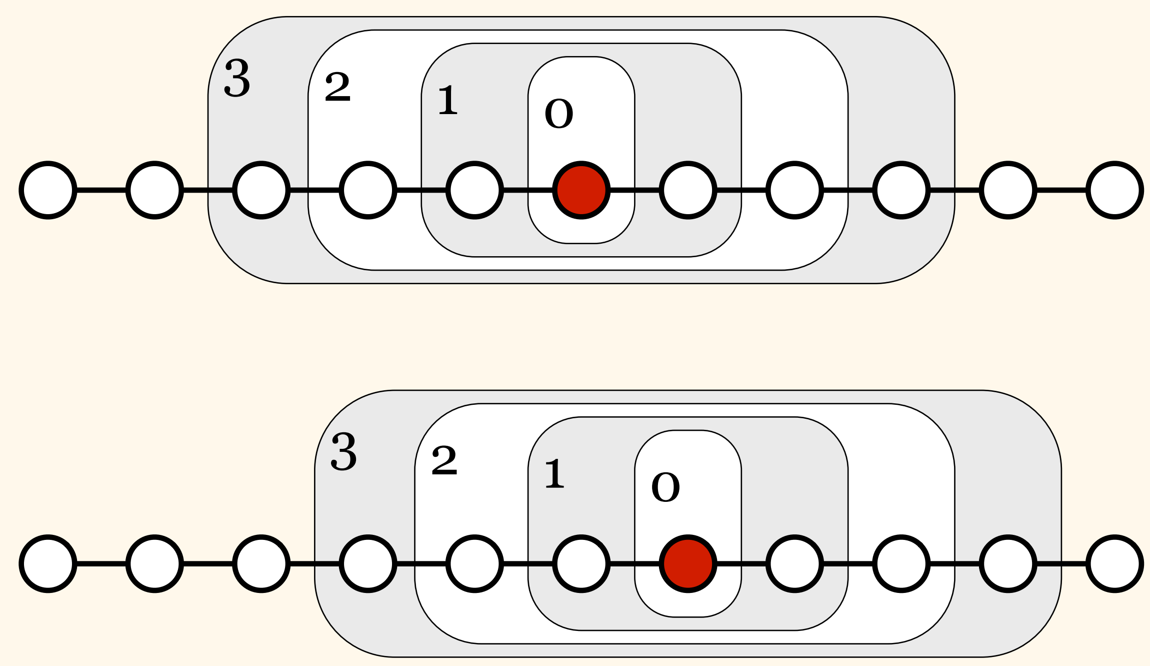
- Time *r*: identical *local states* in radius-0 neighbourhoods





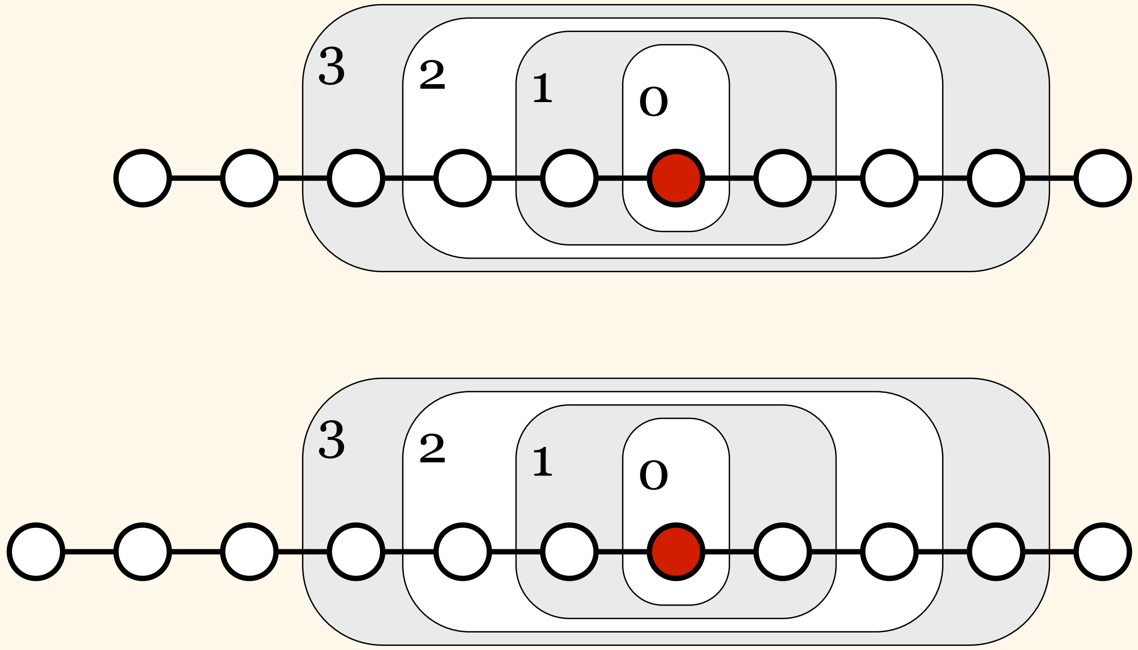
Local Neighbourhoods

- Application: finding midpoint of a path requires $\Omega(n)$ rounds



Local Neighbourhoods

- Application: counting the number of nodes requires $\Omega(n)$ rounds



Proof Techniques

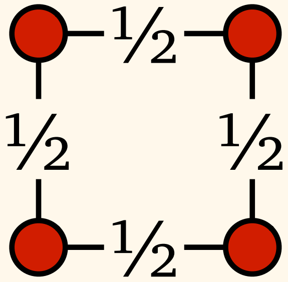
- Covering maps
  - problems that cannot solved at all
- Isomorphic local neighbourhoods
  - problems that cannot be solved quickly
- Plenty of exercises…

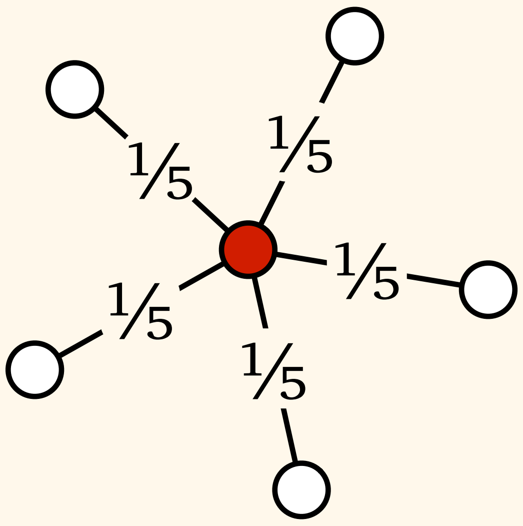
Vertex Covers & Edge Packings

DDA Course

Lecture 4.1

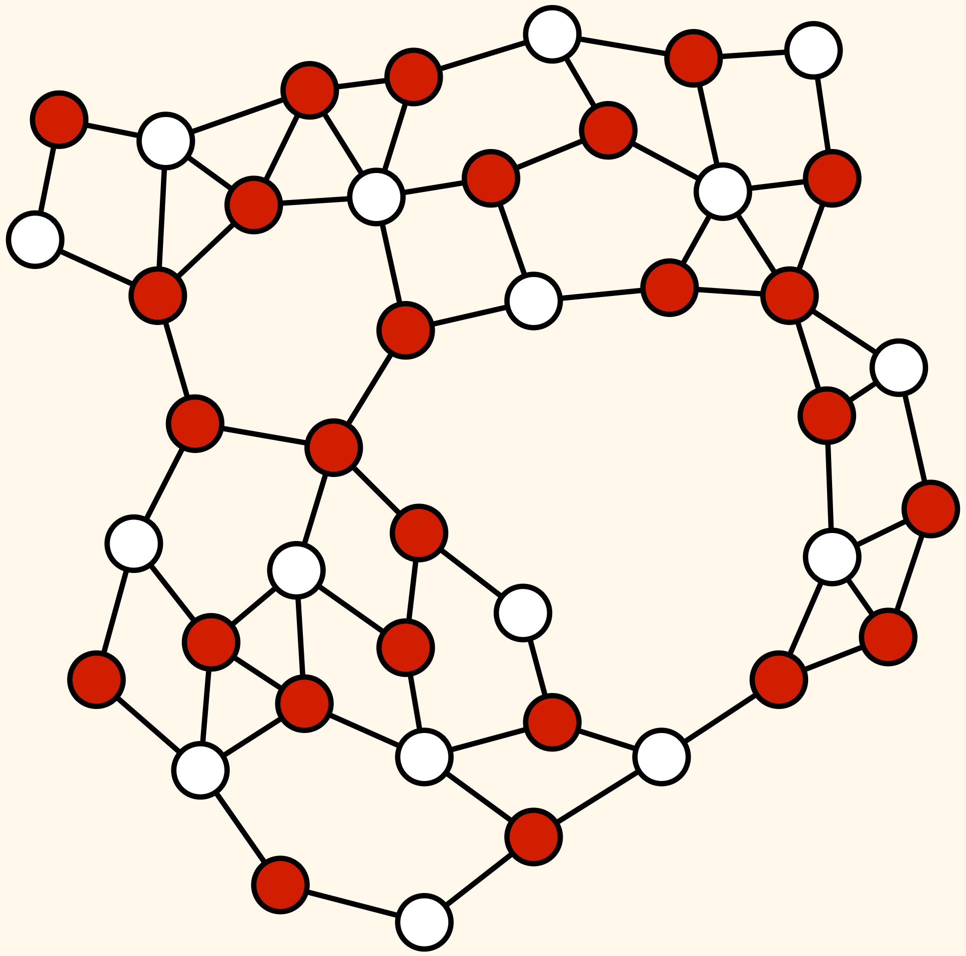
3 April 2012





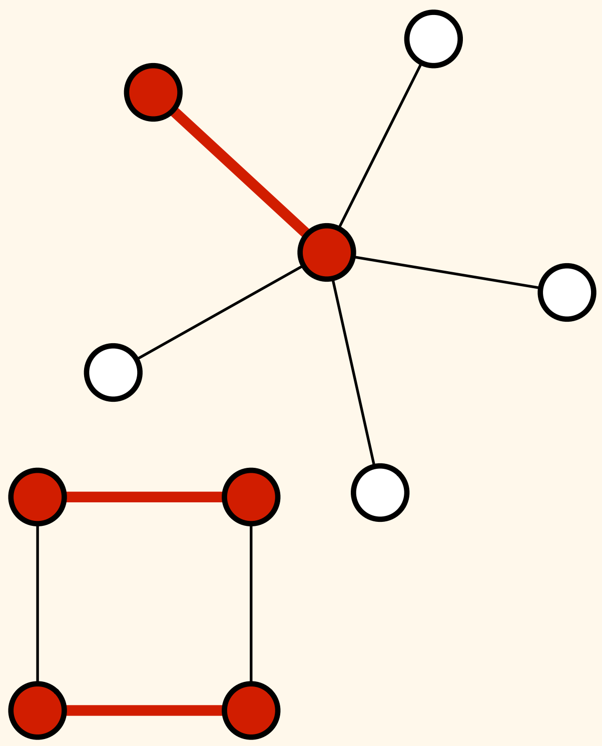
Vertex Cover

- Finding a minimum vertex cover is hard
- How to find good approximations?
- General idea: find something else first, show that it is useful…



Chapter 1

maximal matching

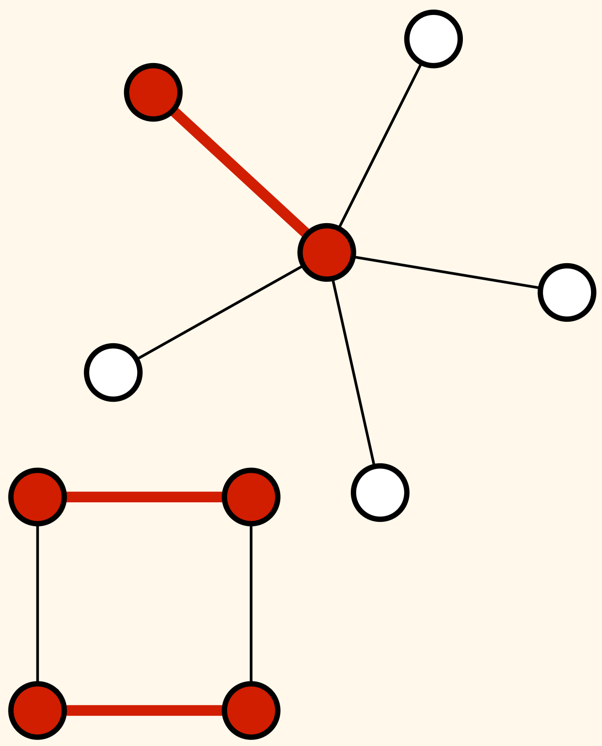


Exercise 1.3:

- find any maximal matching
- take all matched nodes
- 2-approximation of minimum vertex cover

Chapter 1

maximal matching



2-approx.

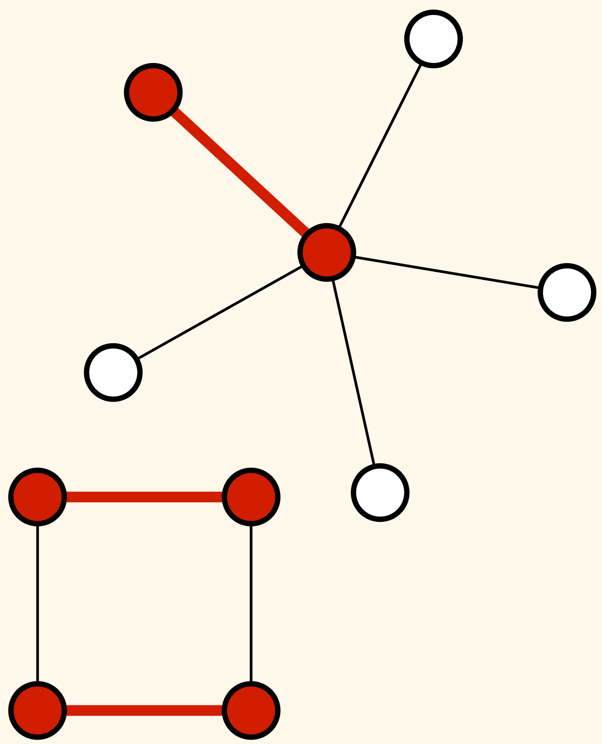
no distributed algorithm

Corollary 3.3:

- there is no distributed algorithm that finds a maximal matching

Chapter 1

maximal matching

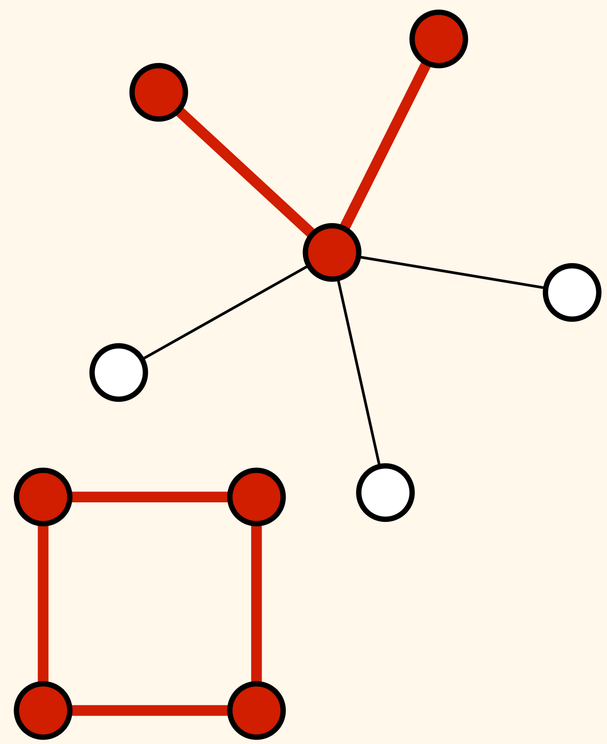


2-approx.

no distributed algorithm

Chapter 2

paths & cycles



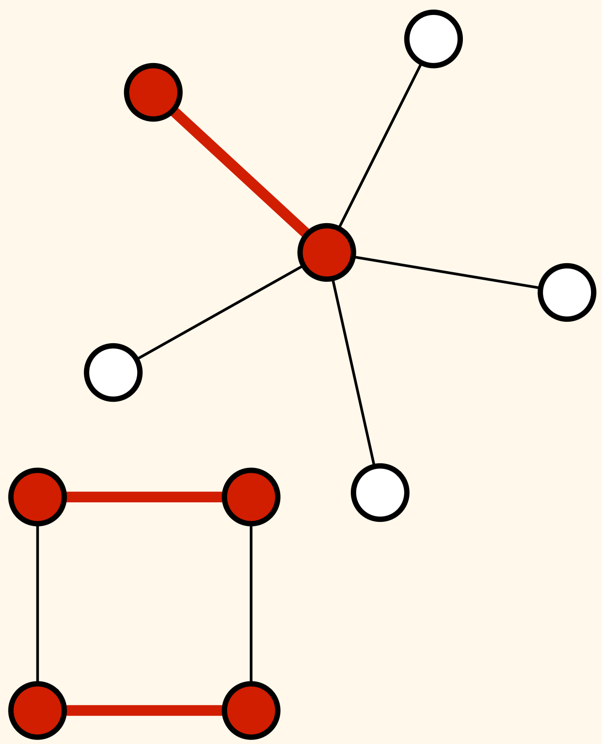
3-approx.

fast distributed algorithm

VC3

Chapter 1

maximal matching

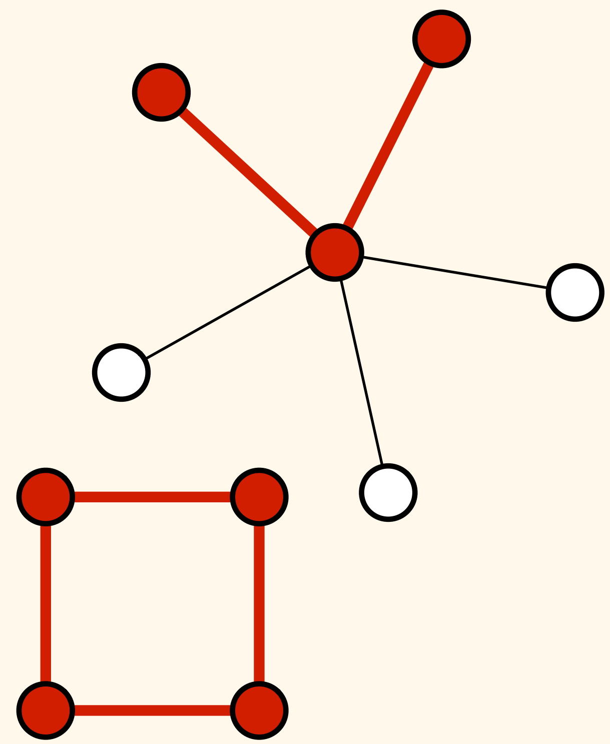


2-approx.

no distributed algorithm

Chapter 2

paths & cycles

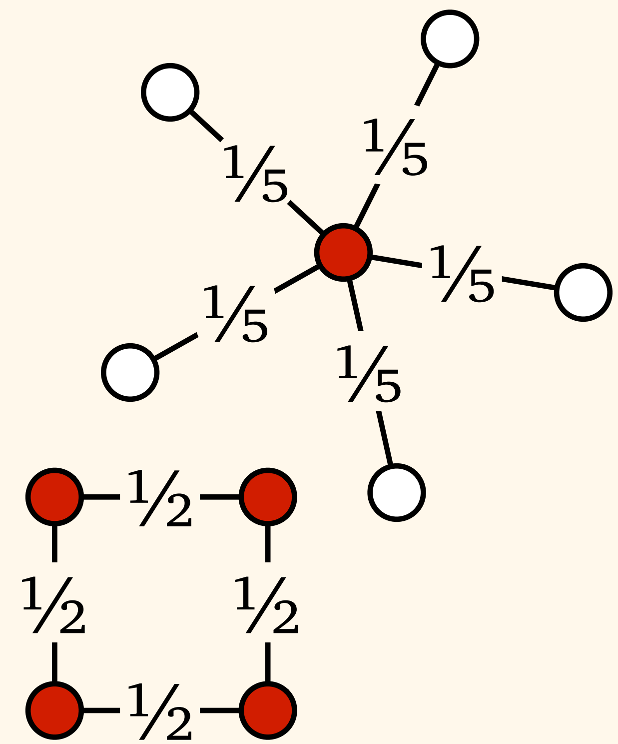


3-approx.

fast distributed algorithm

Chapter 4

edge packing

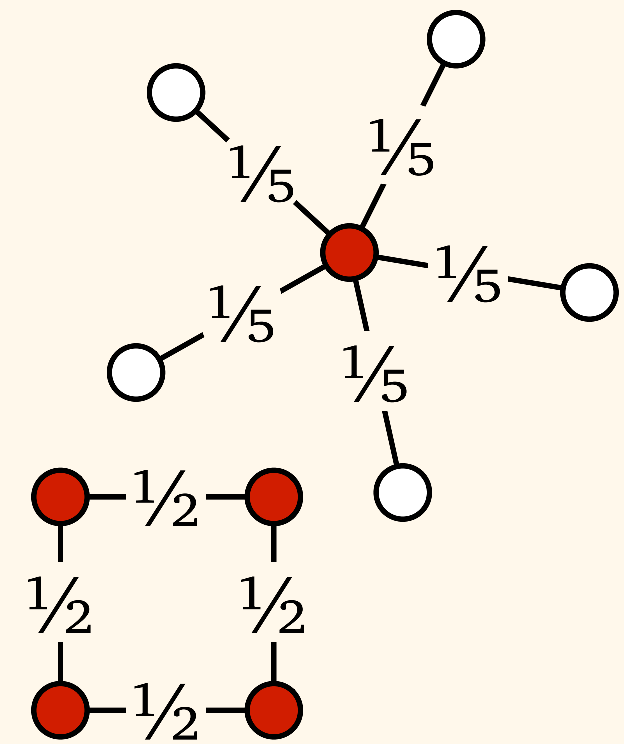


2-approx.

fast distributed algorithm

Edge Packing

- Function $f\colon E \to [0, 1]$
  - $f[v]$ = sum of $f(e)$ over all edges e incident to v
- Constraints: $f[v] \leq 1$

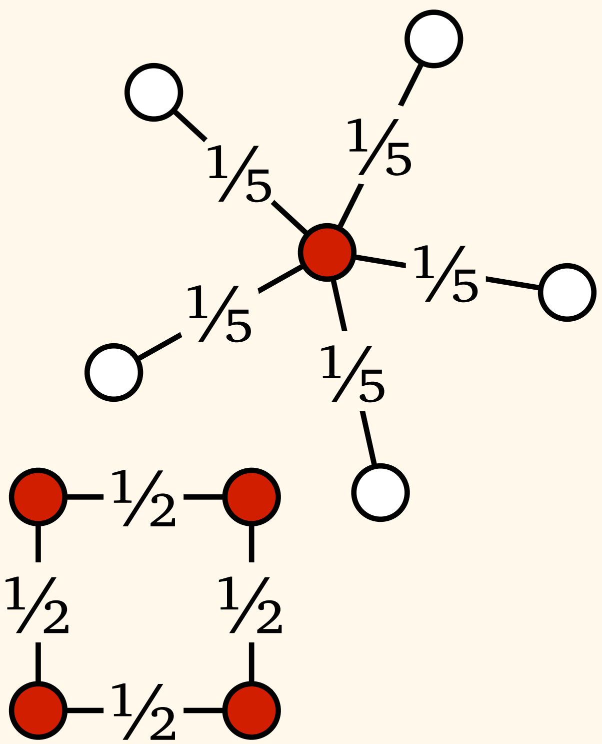


$f[○] = 1/5$

$f[●] = 1$

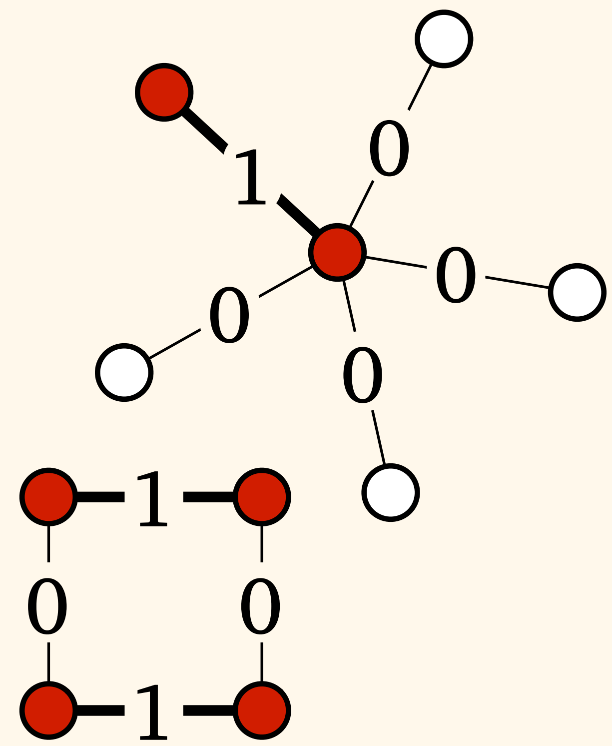
Edge Packing

- Function $f\colon E \to [0, 1]$
  - $f[v]$ = sum of $f(e)$ over all edges e incident to v
- Constraints: $f[v] \le 1$
  - v is *saturated* if $f[v] = 1$
  - edge $e = \{u, v\}$ is *saturated* if u or v is saturated
  - edge packing is *maximal* if all edges are saturated



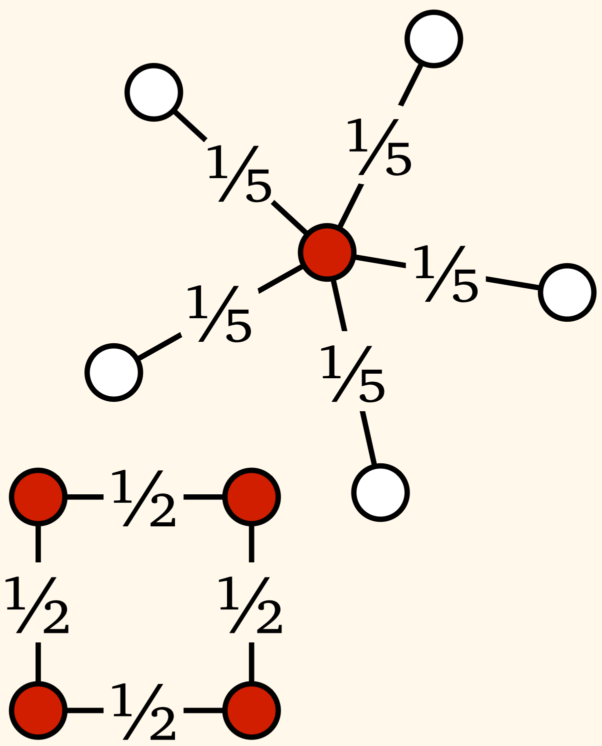
Edge Packing

- Function $f: E \to [0, 1]$
  - $f[v]$ = sum of $f(e)$ over all edges e incident to v
- Constraints: $f[v] \leq 1$
- "Fractional" matching



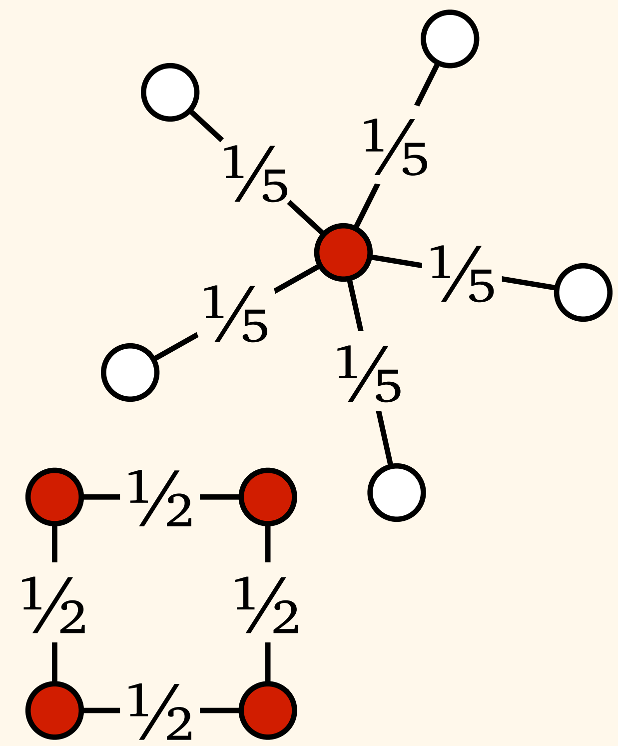
Edge Packing

- Find any maximal edge packing
- Set of saturated nodes: *vertex cover*
    - *Proof*: maximal
      = each edge saturated
      = each edge has a saturated endpoint
      = saturated nodes form a vertex cover

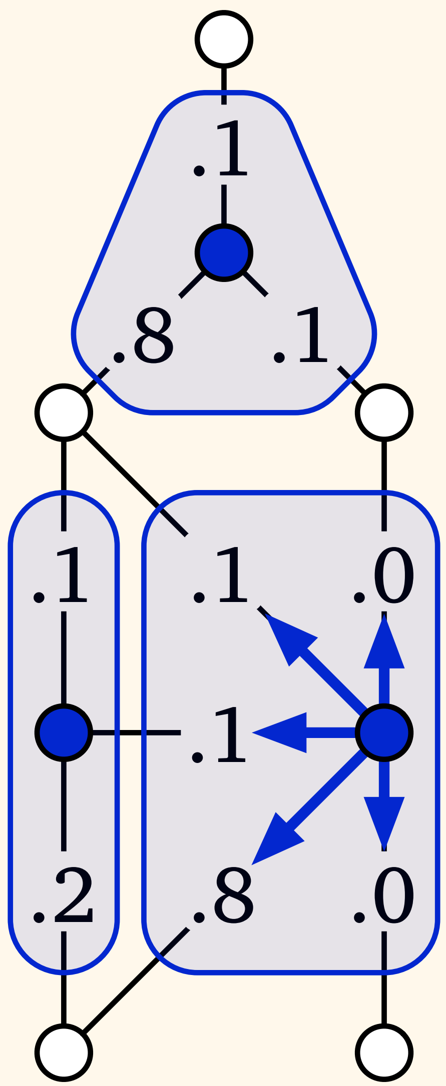


Edge Packing

- Find any maximal edge packing
- Set of saturated nodes: *2-approximation of minimum vertex cover*



Edge Packing



C^*

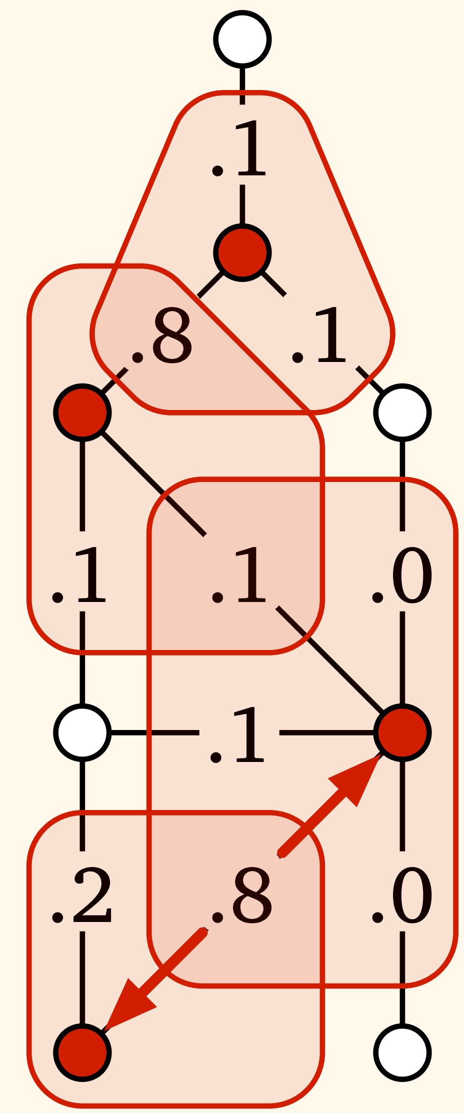
Each node $v \in C^*$ has 1 unit of money

Give $f(e)$ units to each edge e

Double all money

Give $f[v] = 1$ units to each saturated node $v \in C$

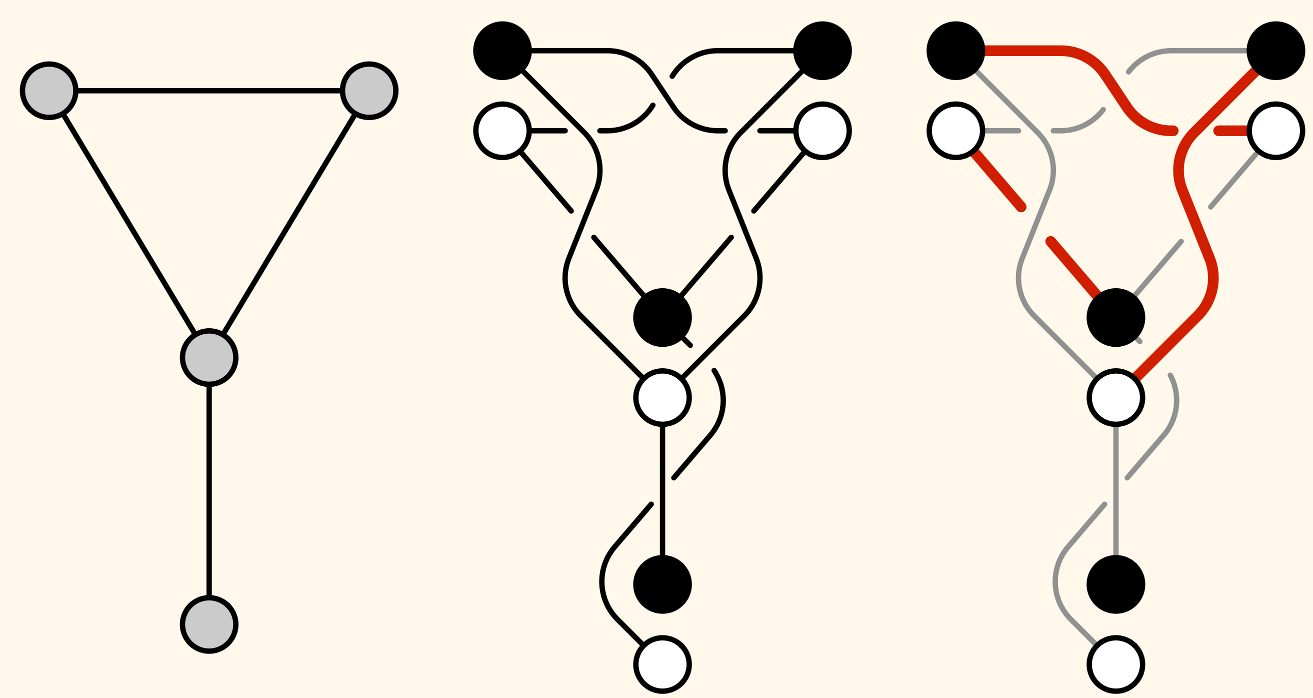
$|C| \leq 2\,|C^*|$

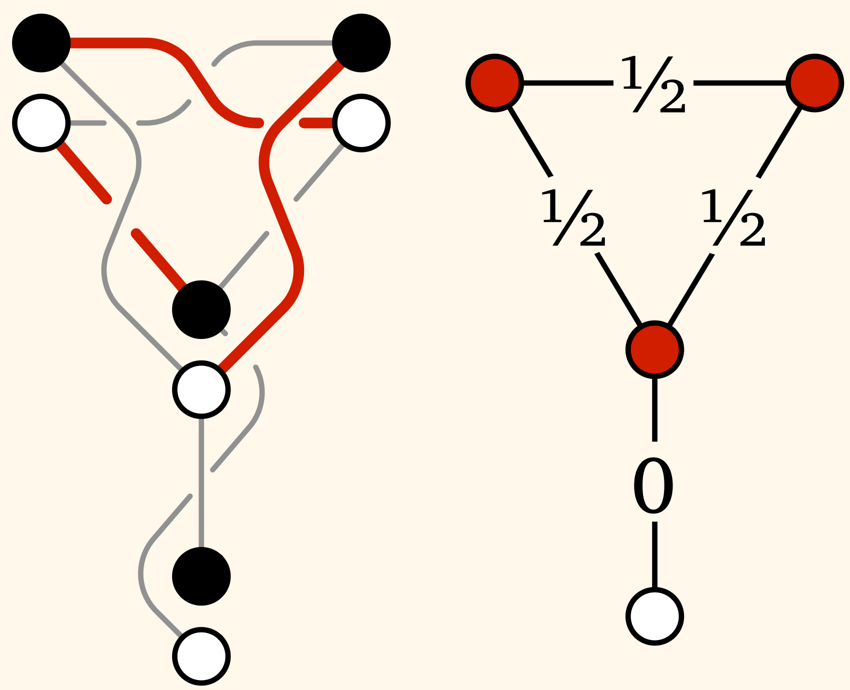


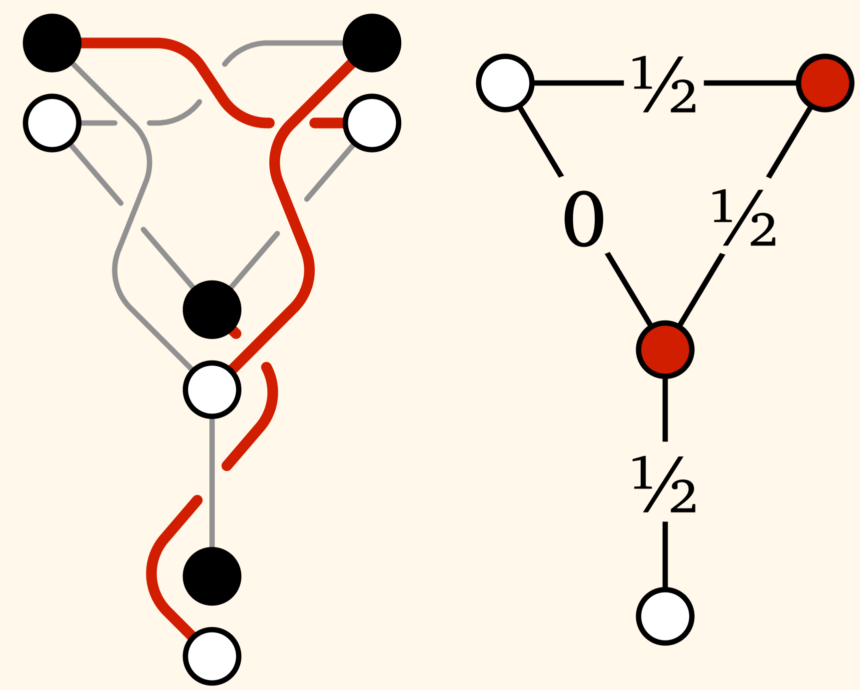
C

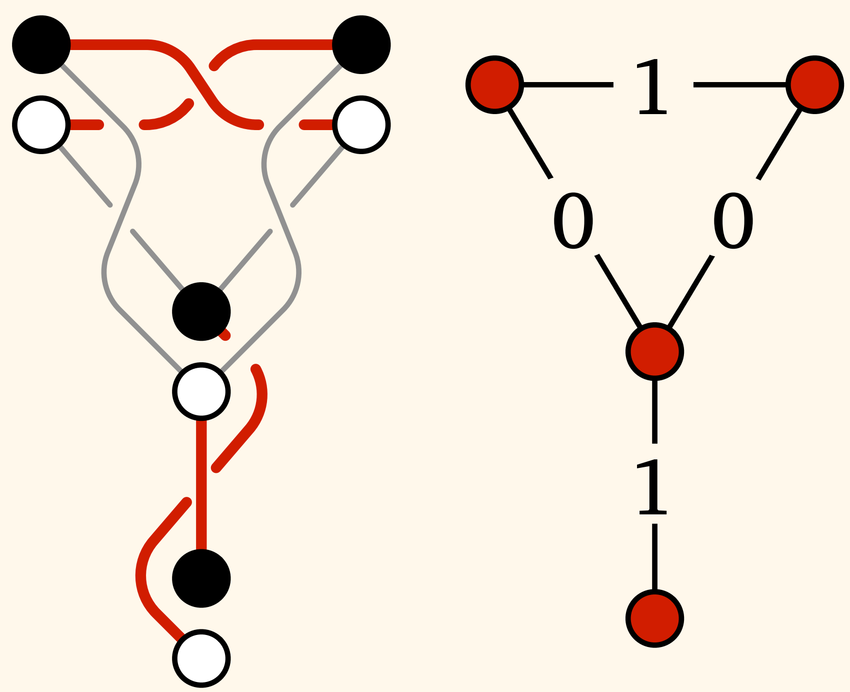
Edge Packing

- How to find maximal edge packings?
- Basic idea:
  - bipartite double covers
  - maximal matching
  - recursively!







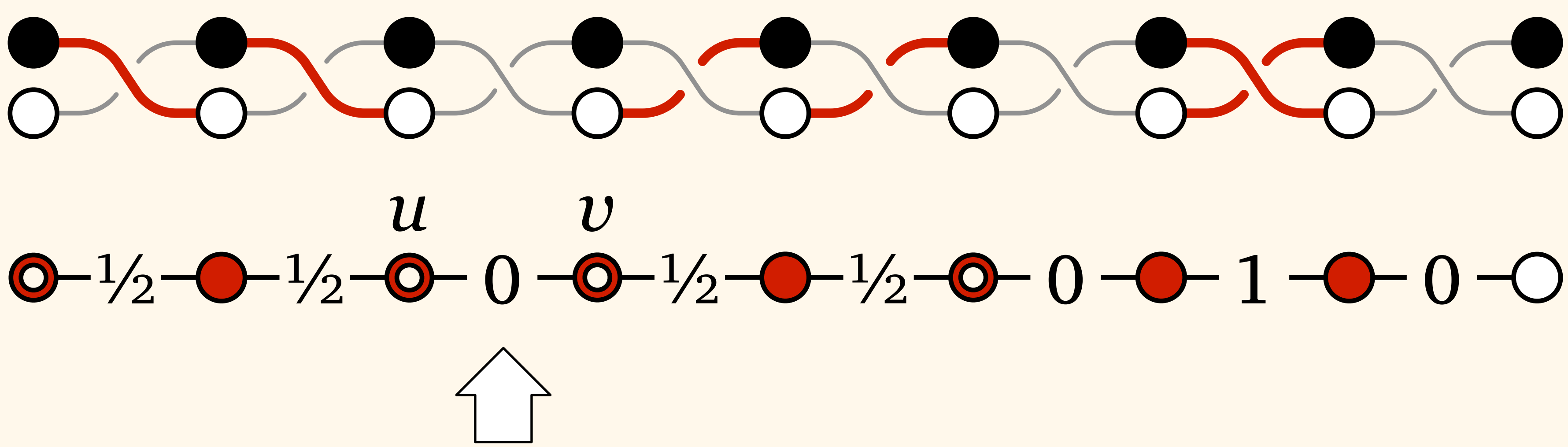


One edge: 1/2

Two edges: 1

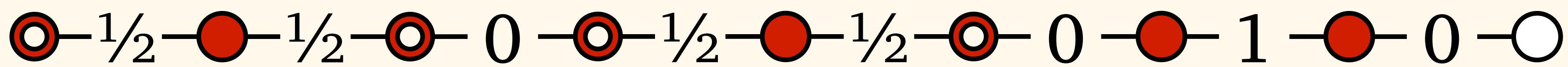
Edge Packing

- In general only "half-saturating"

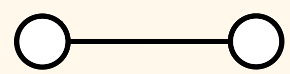


unsaturated edge $e = \{u, v\}$
$f[u] = f[v] = 1/2$

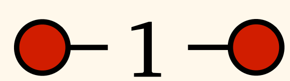
Half-saturating edge packing:



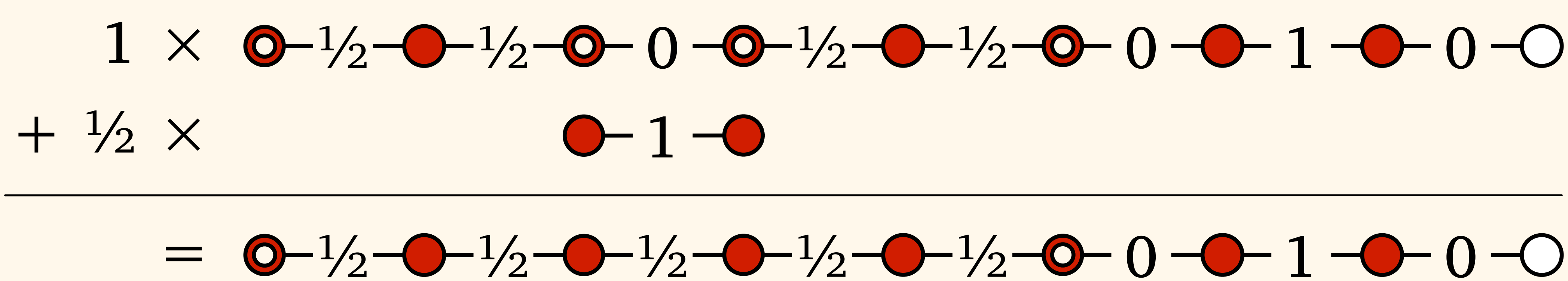
Unsaturated subgraph (*lower degrees*):



Recursively, find a *maximal* edge packing:



Combine solutions — *maximal* edge packing:

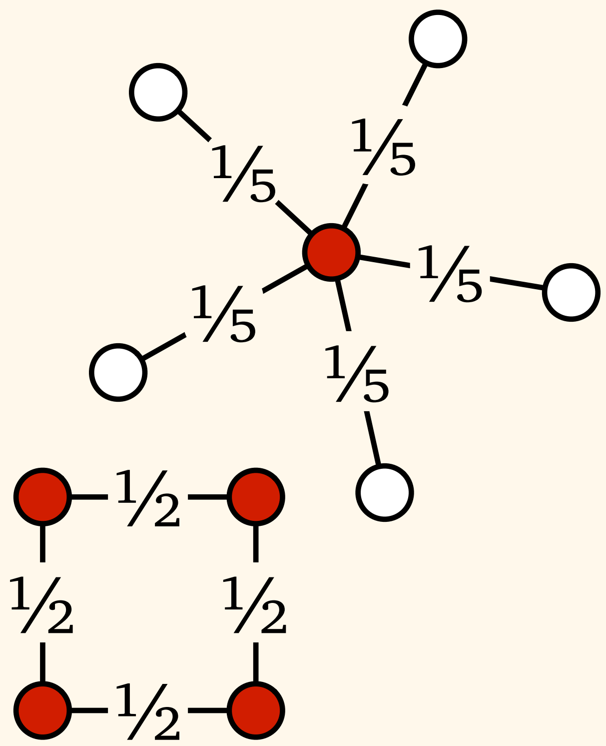


Edge Packing

- Recursion by maximum degree Δ
- Case $\Delta = 1$ trivial
- Assuming that case $\Delta - 1$ has been solved:
  - find a *half-saturating* edge packing f
  - recursively, find a *maximal* edge packing g for unsaturated subgraph (maximum degree $\Delta - 1$)
  - return *maximal* edge packing $h = f + g/2$

Summary

- Distributed algorithms that finds a *maximal edge packing*
  - in any graph of maximum degree Δ in time $O(\Delta^2)$
- Saturated nodes: *2-approximation of minimum vertex cover*



Unique Identifiers

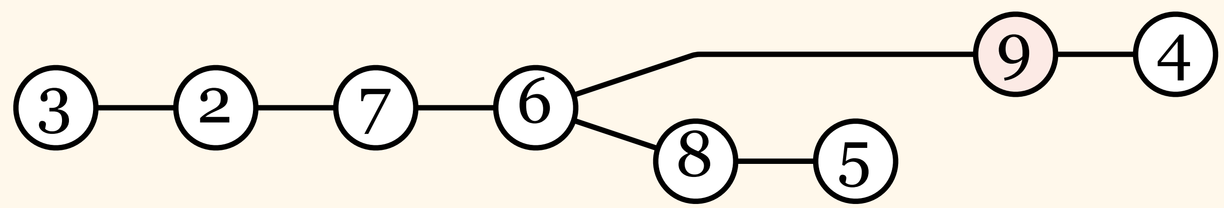
DDA Course

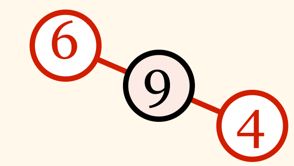
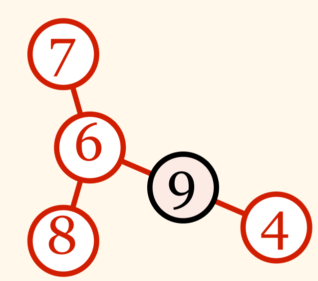
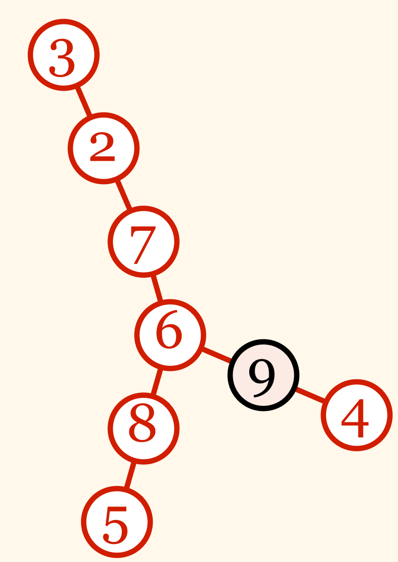
Lecture 5.1

17 April 2012

Unique Identifiers

- Networks with *globally unique identifiers*
  - IPv4 address, IPv6 address, MAC address, IMEI number, …
- "Everything" can be discovered
  - in a connected graph G, all nodes can discover full information about G in time $O(\text{diam}(G))$



round 1:	{2,3}	{2,3} {2,7}	{2,7} {6,7}	{6,7} {6,8} {6,9}	{5,8} {6,8}	{5,8}	{4,9} {6,9}	{4,9}	
round 2:	{2,3} {2,7}	{2,3} {2,7} {6,7}	{2,3} {2,7} {6,7} {6,8} {6,9}	{2,7} {4,9} {5,8} {6,7} {6,8} {6,9}	{5,8} {6,7} {6,8} {6,9}	{5,8} {6,8}	{4,9} {6,7} {6,8} {6,9}	{4,9} {6,9}	
...									
round 5:	{2,3} {2,7} {4,9} {5,8} {6,7} {6,8} {6,9}	{2,3} {2,7} {4,9} {5,8} {6,7} {6,8} {6,9}	{2,3} {2,7} {4,9} {5,8} {6,7} {6,8} {6,9}	{2,3} {2,7} {4,9} {5,8} {6,7} {6,8} {6,9}	{2,3} {2,7} {4,9} {5,8} {6,7} {6,8} {6,9}	{2,3} {2,7} {4,9} {5,8} {6,7} {6,8} {6,9}	{2,3} {2,7} {4,9} {5,8} {6,7} {6,8} {6,9}	{2,3} {2,7} {4,9} {5,8} {6,7} {6,8} {6,9}	

Unique Identifiers

- "Everything" can be discovered
    - in a connected graph G, all nodes can discover full information about G in time $O(\mathrm{diam}(G))$
- "Everything" can be solved
    - once all nodes know G, solving a graph problem is just a local state transition
- Key question: what can be solved *fast*?

Graph Colouring

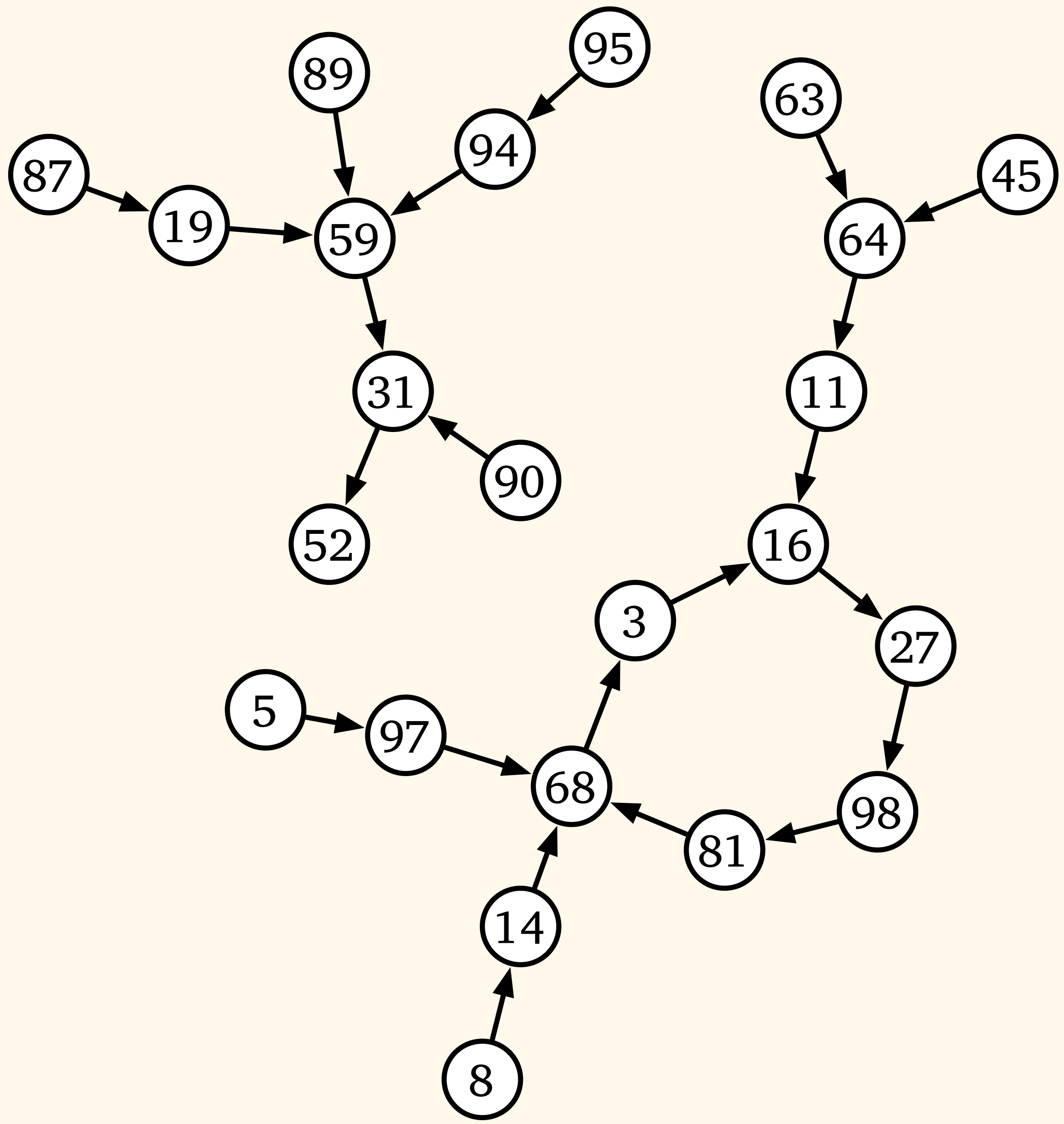
- Given unique identifiers,
  can we find a graph colouring fast?
  - unique identifiers from $\{1, 2, \ldots, x\}$ can be interpreted as a graph colouring with x colours
  - problem: huge number of colours
  - we only need to solve a *colour reduction* problem: given an x-colouring, find a y-colouring for a small $y < x$

Greedy Graph Colouring

- All nodes of colour x pick the smallest free colour in their neighbourhood
  - there is always a free colour in the set $\{1, 2, \ldots, \Delta + 1\}$
  - reduces the number of colours from x to $x - 1$, assuming that $x > \Delta + 1$
- Very slow…

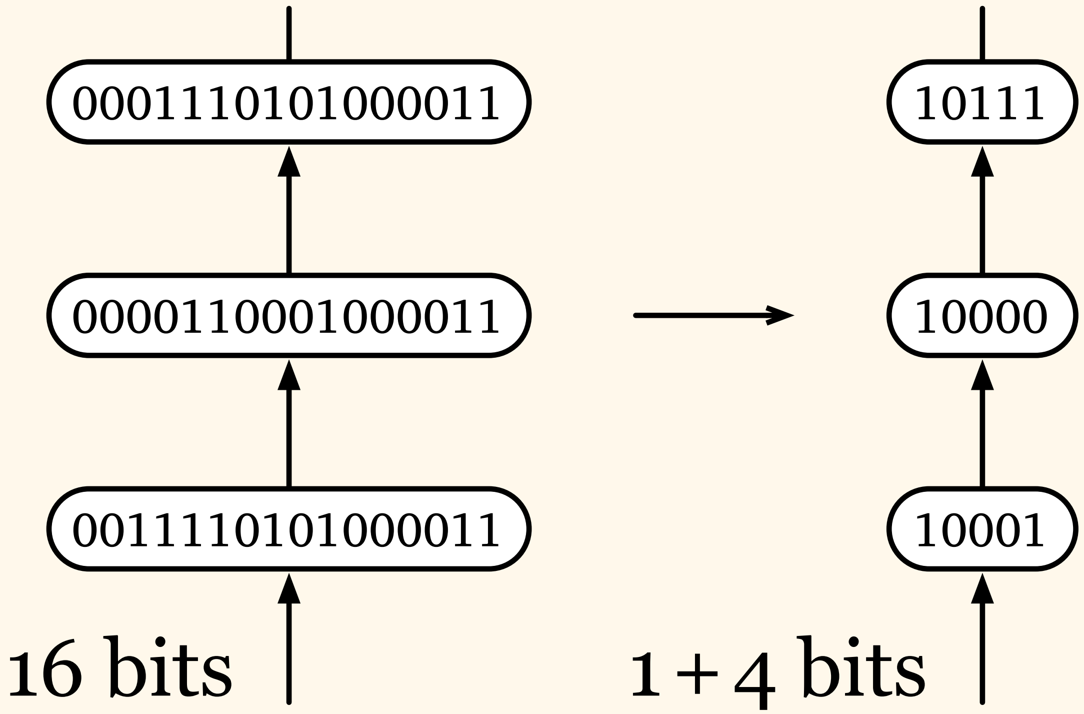
Fast Graph Colouring

- Let's first study a special case...
- *Directed pseudoforest*
  - edges oriented
  - outdegree ≤ 1



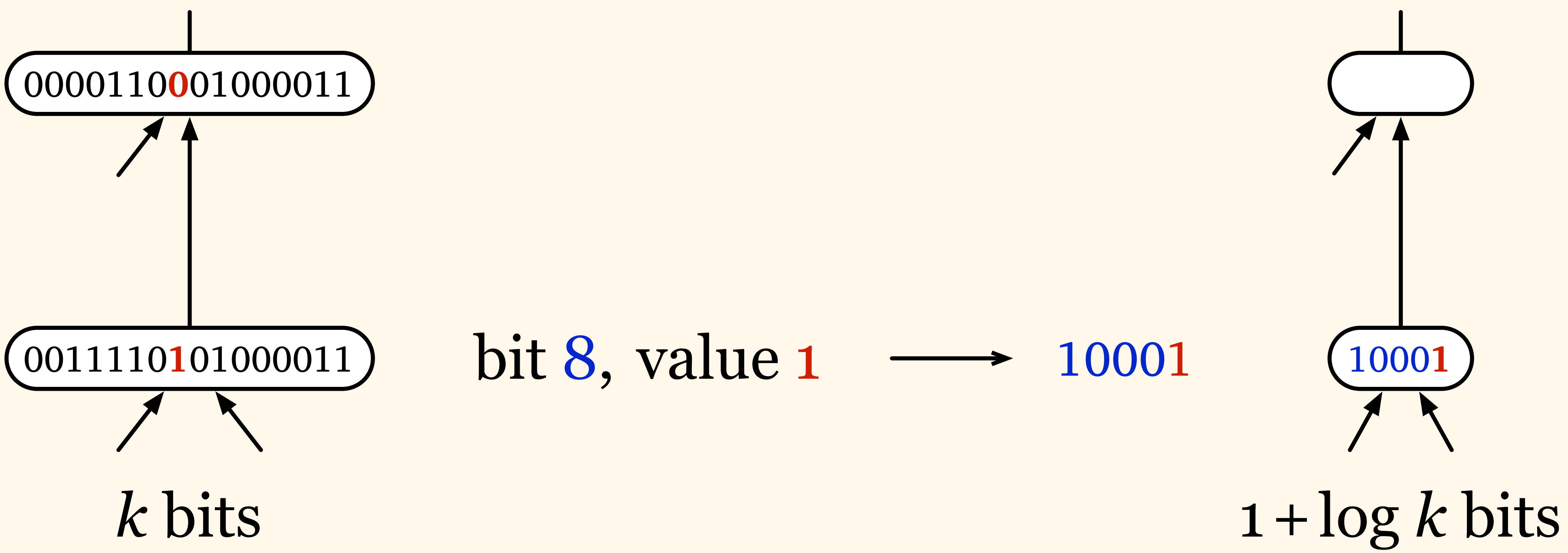
Fast Graph Colouring

- Idea: colour = *binary string*
- Reduce colours:
  - k bits $\to$ $1 + \log_2 k$ bits
  - 2^k colours $\to$ $2k$ colours



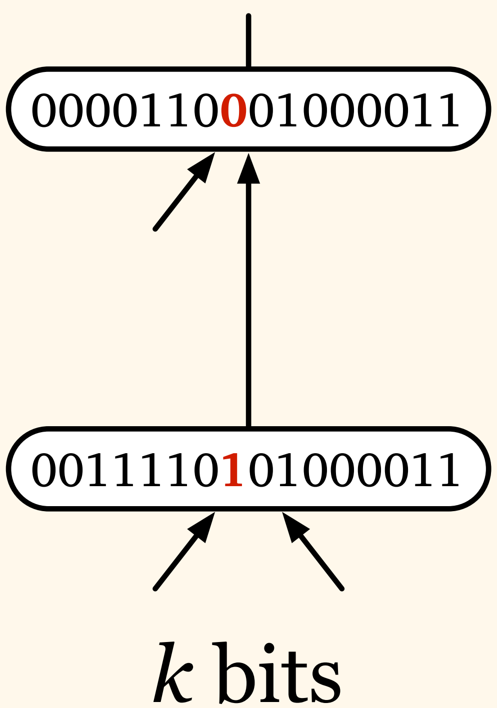
Fast Graph Colouring

- Compare bit string with the successor, find the *first bit that differs*



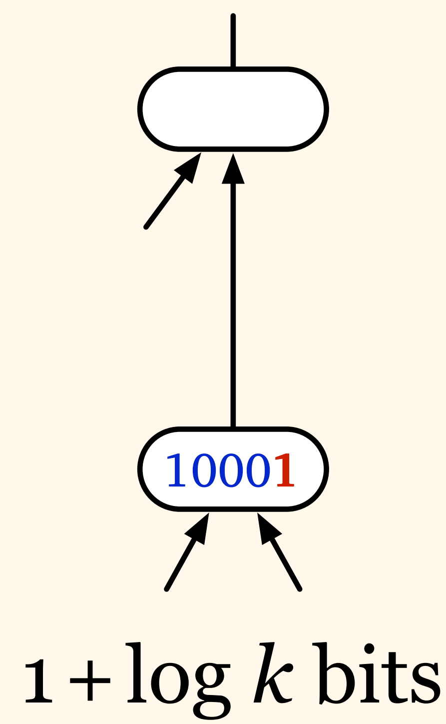
Fast Graph Colouring

- Correct, no matter what the successor does



...
bit 6, value 1 ⟶ 01101
bit 7, value 0 ⟶ 01110
bit 8, value 0 ⟶ 10000
...

bit 8, value 1 ⟶ 10001

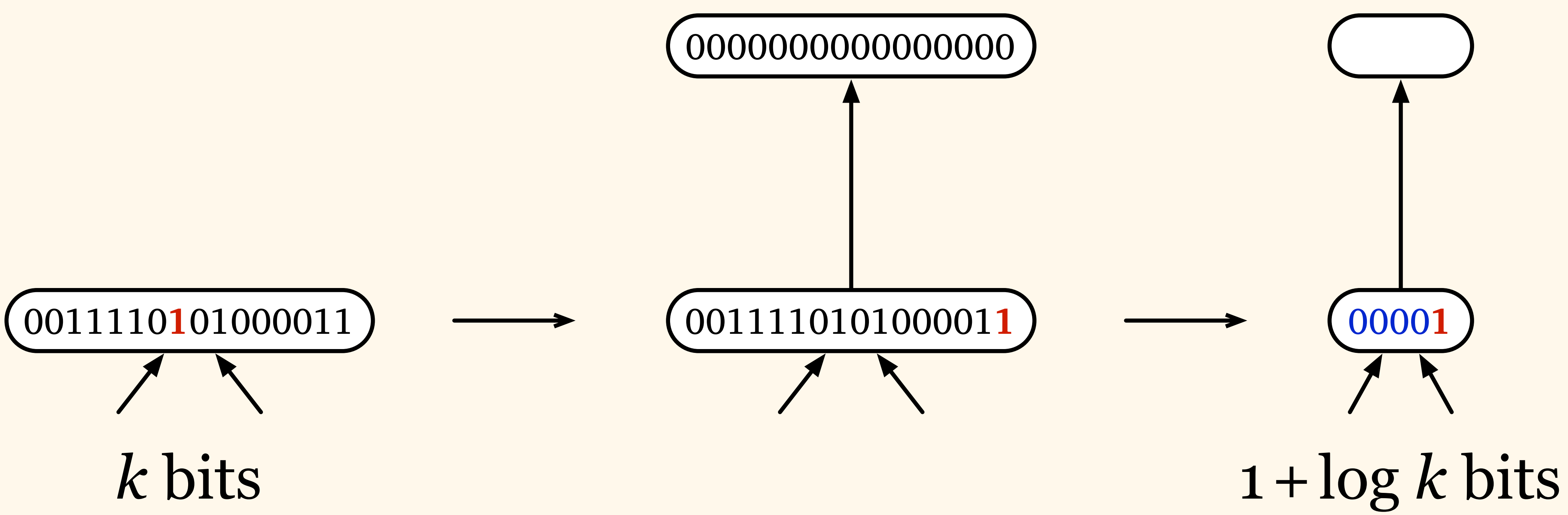


Fast Graph Colouring

- Correct, no matter what the successor does
- For each directed edge (u, v):
  - the new colour of node u is different from the new colour of its successor v
- Proper graph colouring

Fast Graph Colouring

- No successor?
  Pretend that there is one…

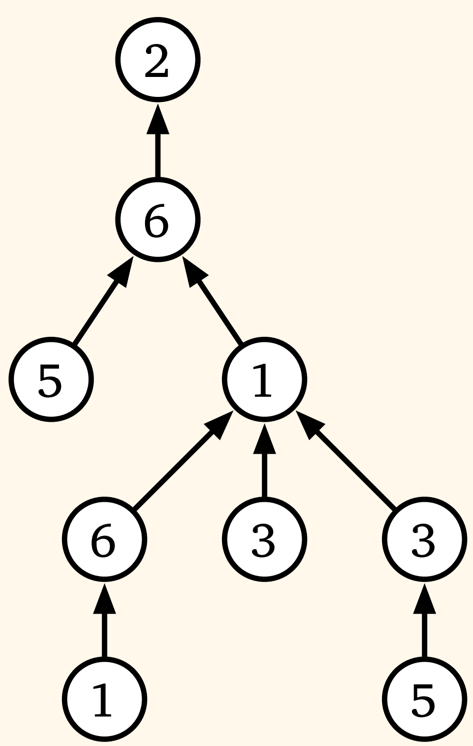


Fast Graph Colouring

- Very fast colour reduction:
  - 2^{128} colours $\rightarrow 2 \cdot 128 = 2^8$ colours
  - 2^8 colours $\rightarrow 2 \cdot 8 = 2^4$ colours
  - 2^4 colours $\rightarrow 2 \cdot 4 = 2^3$ colours
  - 2^3 colours $\rightarrow 2 \cdot 3 = 6$ colours
- But now we are stuck – how to get below 6?

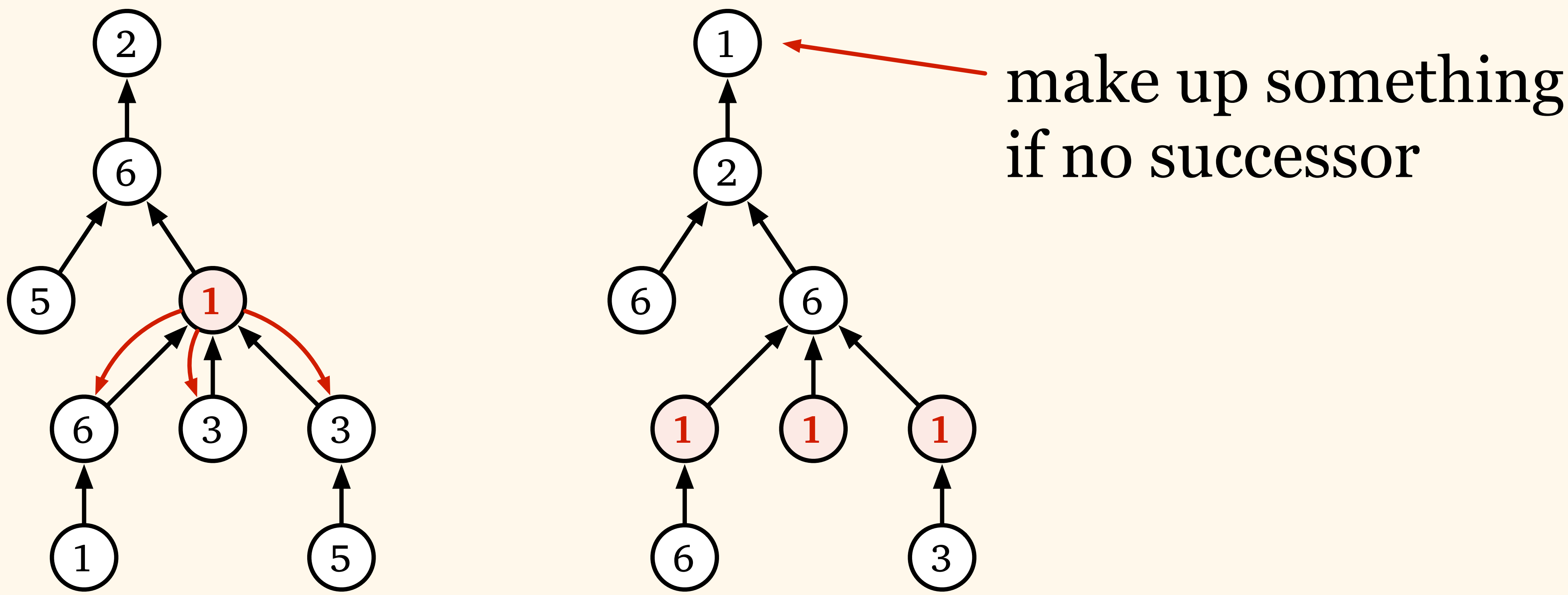
Fast Graph Colouring

- Directed pseudotree with 6 colours:
  how to reduce the number of colours?



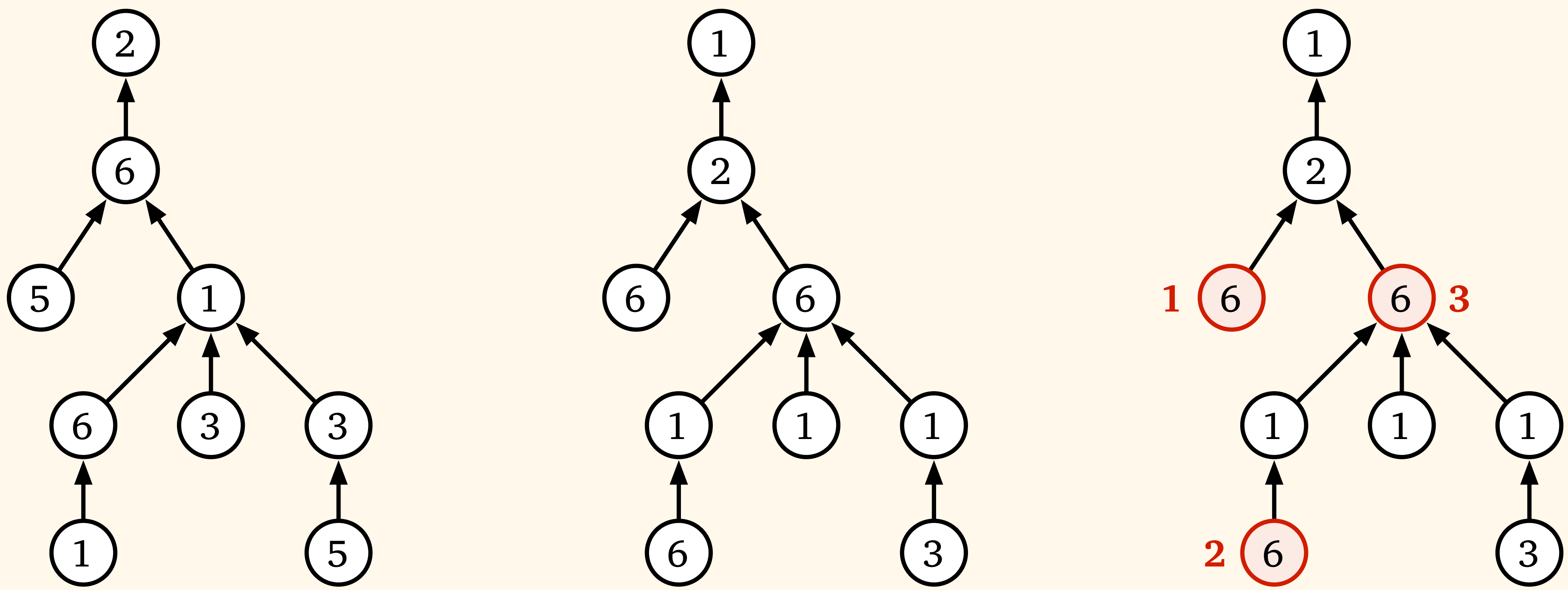
Fast Graph Colouring

- Shift colours "down":
  all predecessors have the same colour



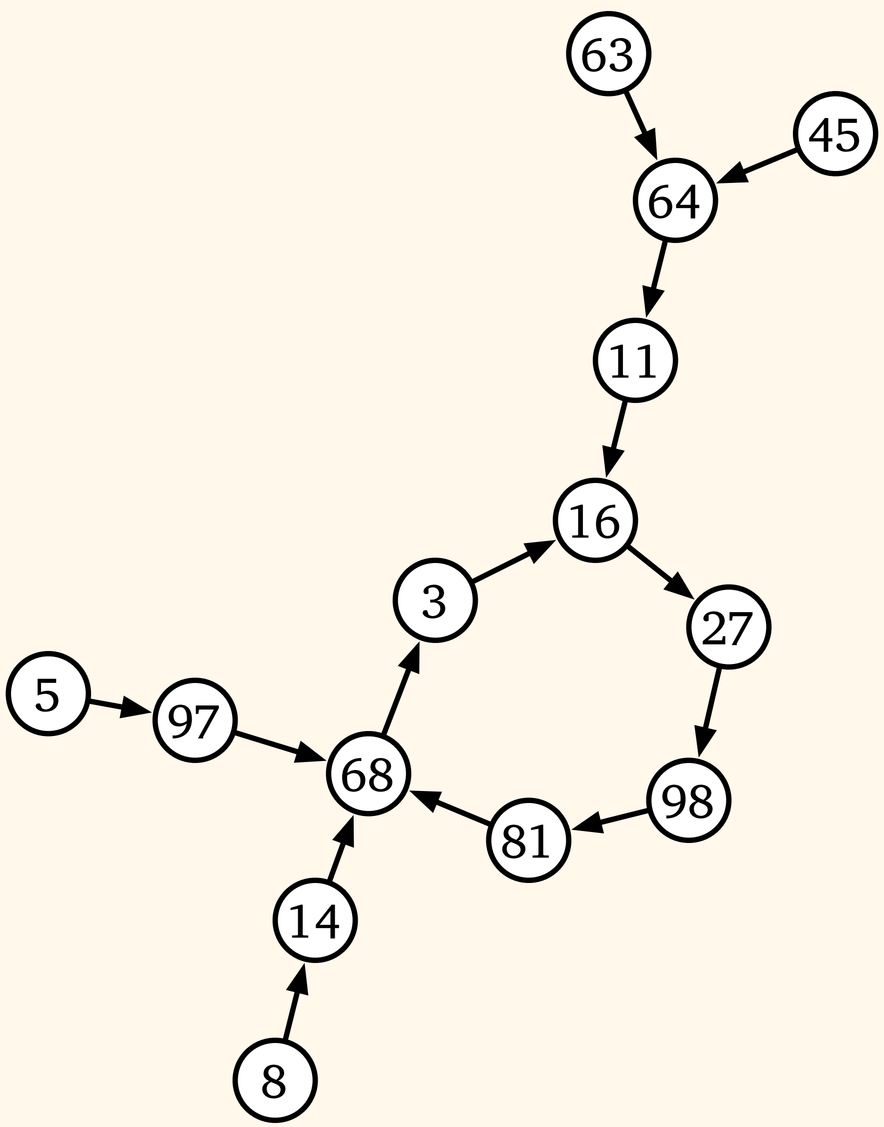
Fast Graph Colouring

- Now greedy works very well:
  there is always a free colour in set {1, 2, 3}



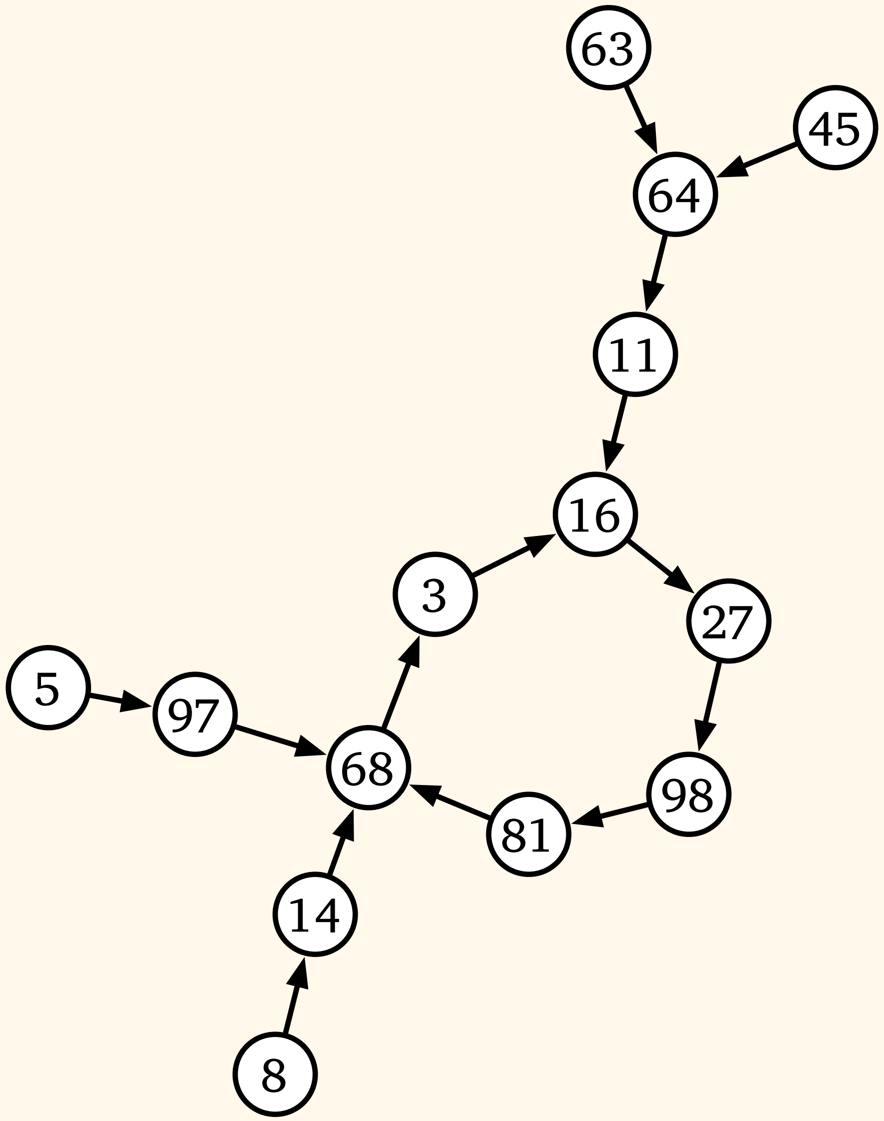
Fast Graph Colouring

- Colour reduction in directed pseudotrees
  - bit comparisons: very quickly from x to 6 colours
  - $2^{128} \rightarrow 2^8 \rightarrow 16 \rightarrow 8 \rightarrow 6$
  - shift + greedy: slowly from 6 to 3 colours
  - $6 \rightarrow 5 \rightarrow 4 \rightarrow 3$



Fast Graph Colouring

- Colour reduction in directed pseudotrees
  - next lecture:
    fast graph colouring
    for arbitrary graphs



Graph Colouring

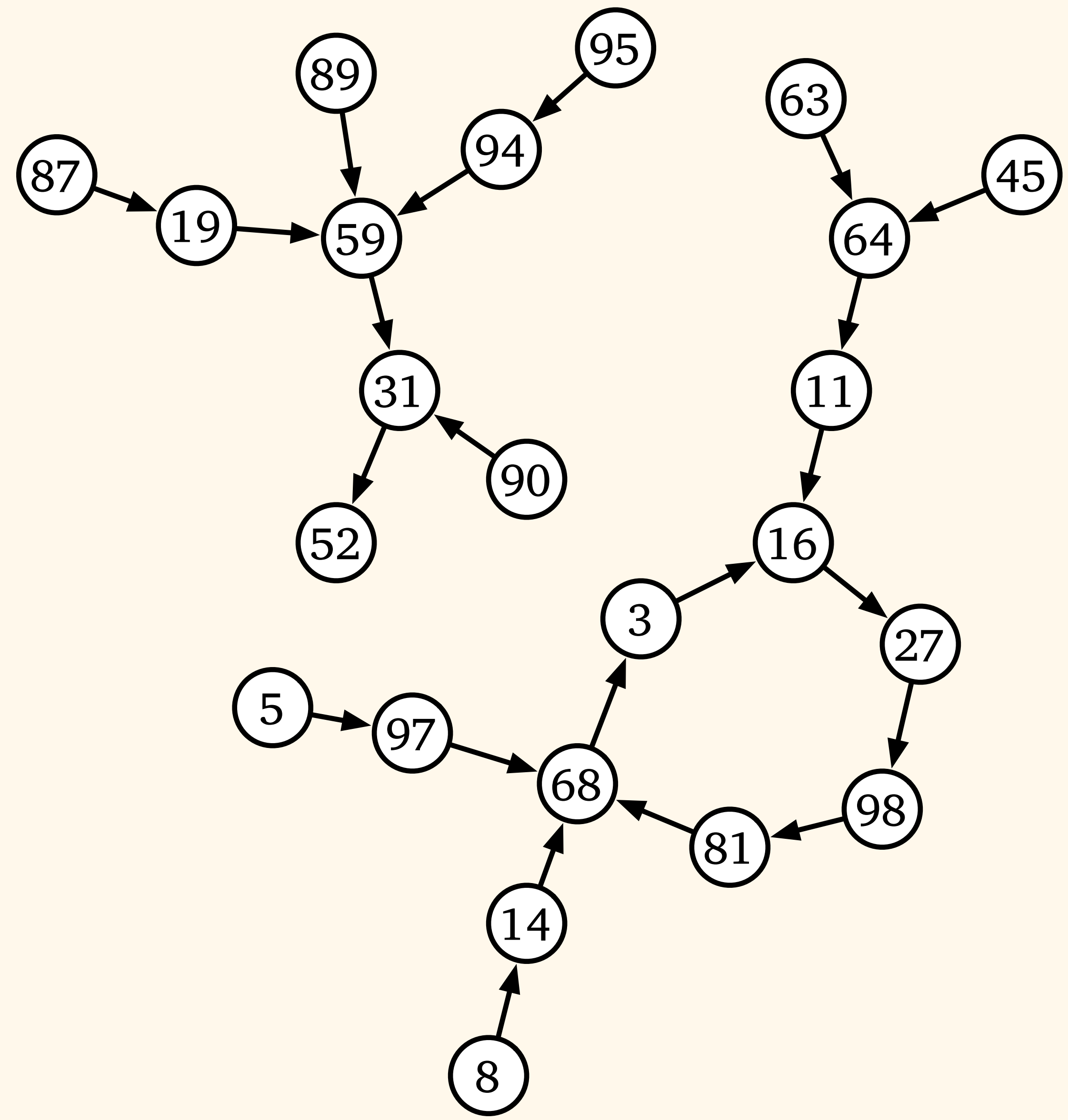
DDA Course

Lecture 5.2

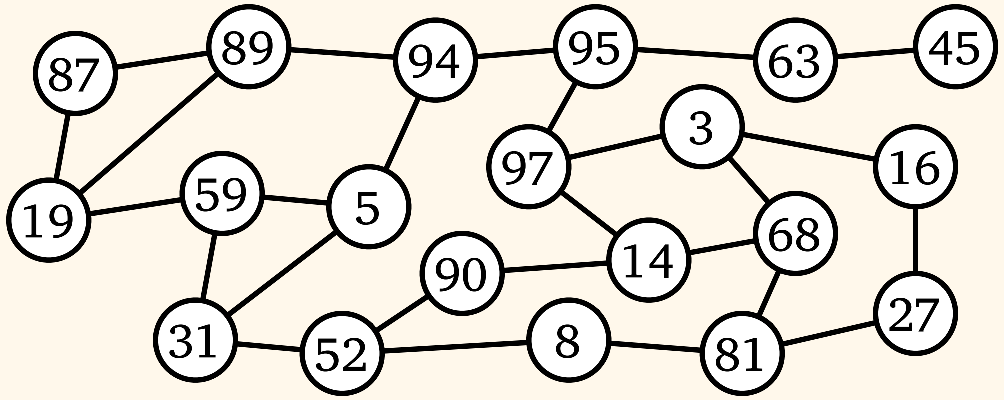
19 April 2012

Fast Graph Colouring

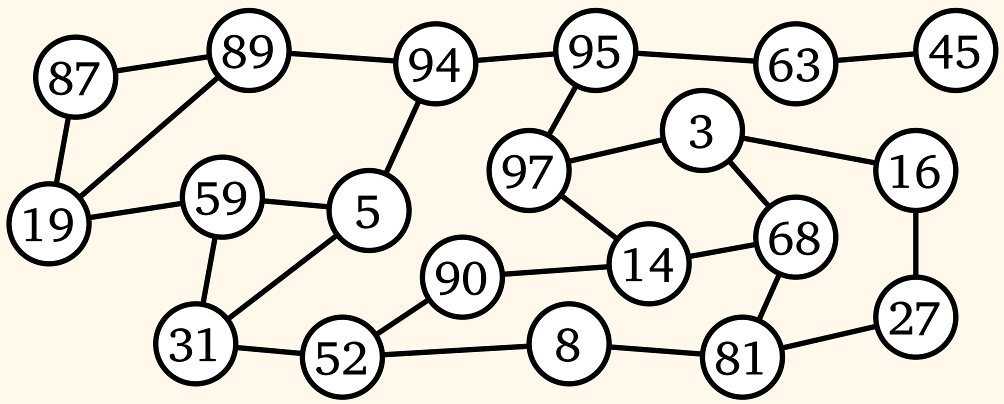
- Previous lecture:
    - colour reduction in *directed pseudoforests*
- Today:
    - colour reduction in general graphs of maximum degree Δ



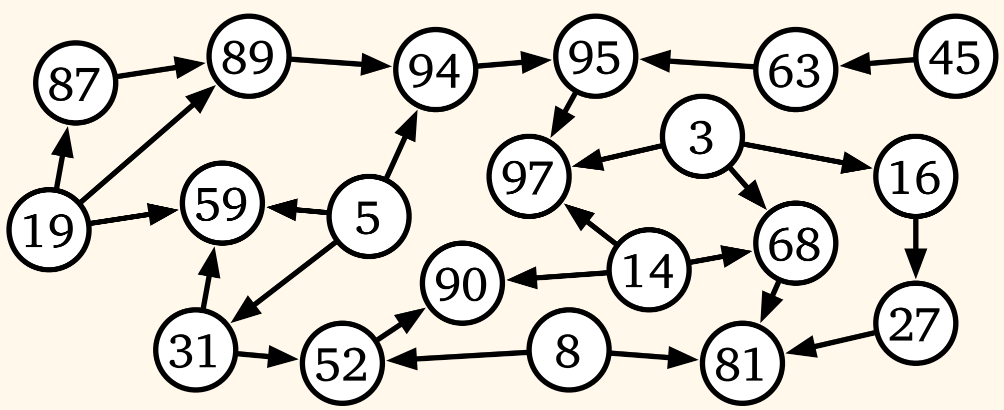
Input:



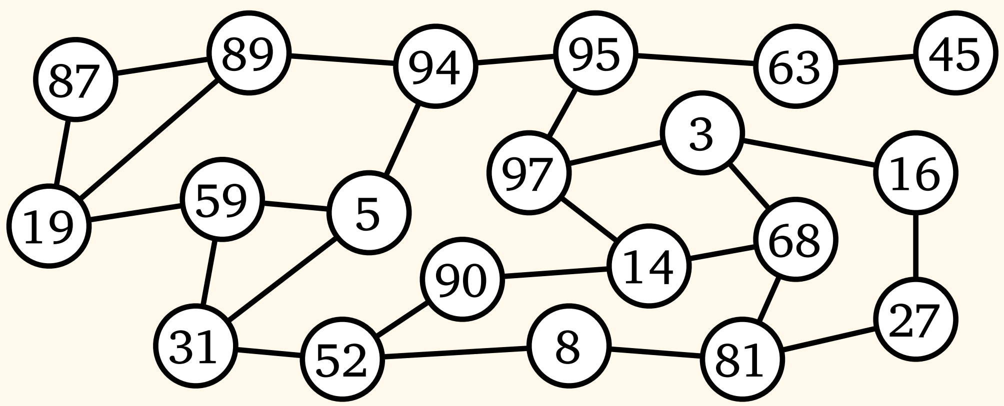
Input:



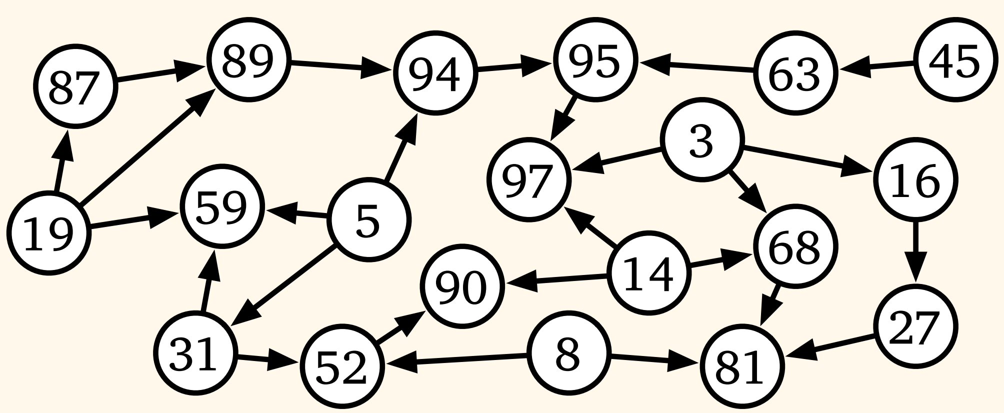
Colours → orientation:



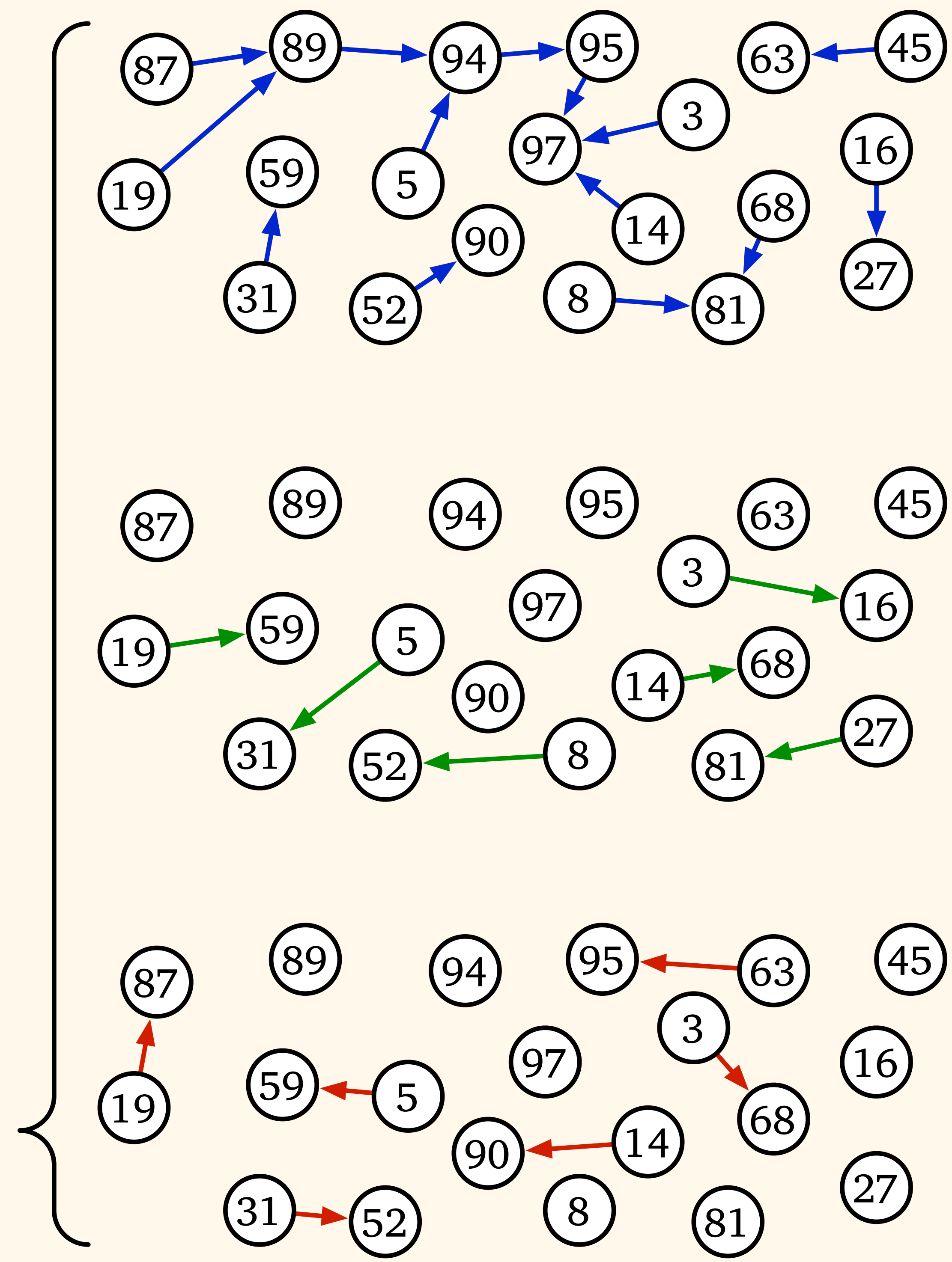
Input:



Colours → orientation:



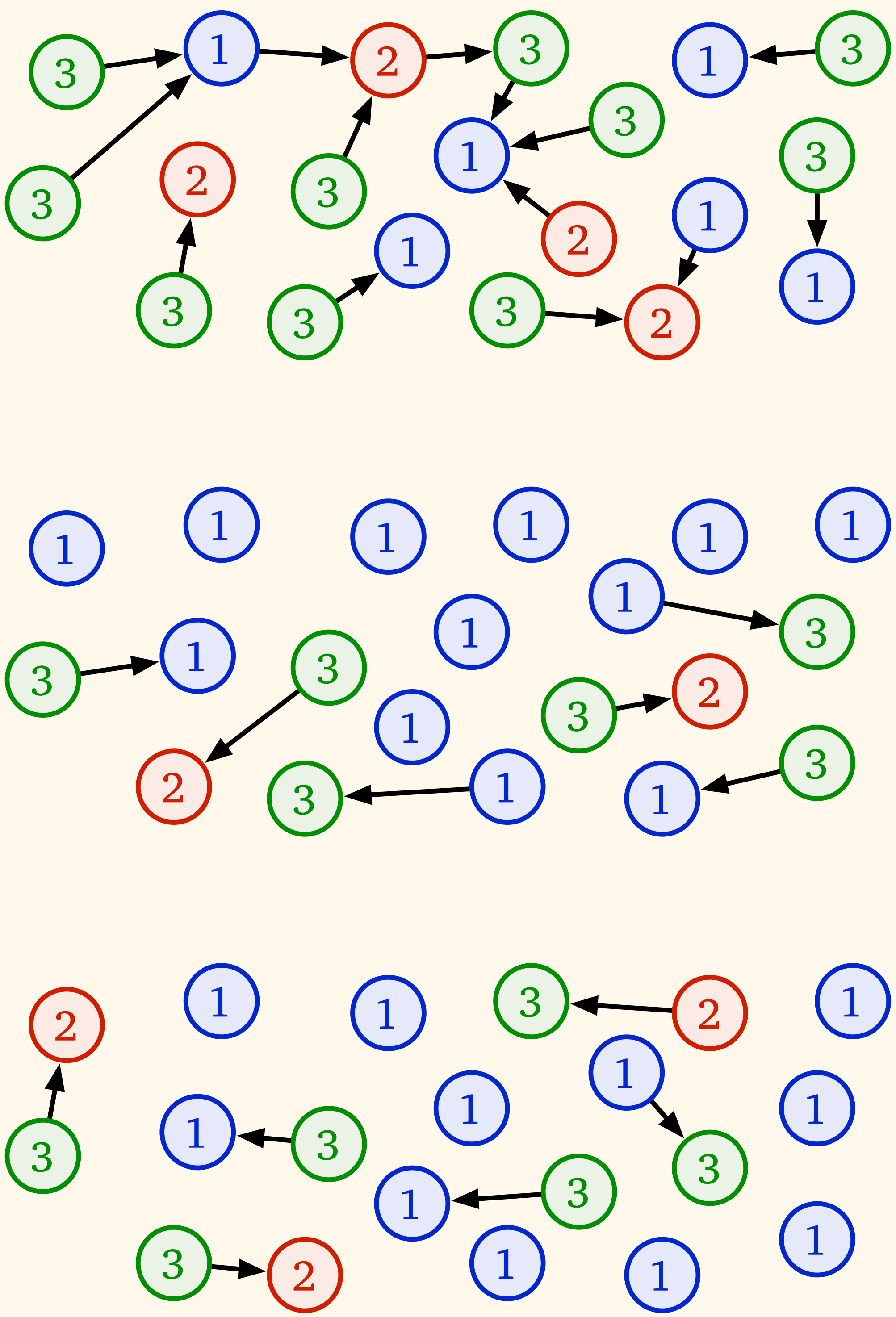
Port numbers → partition in Δ directed pseudoforests

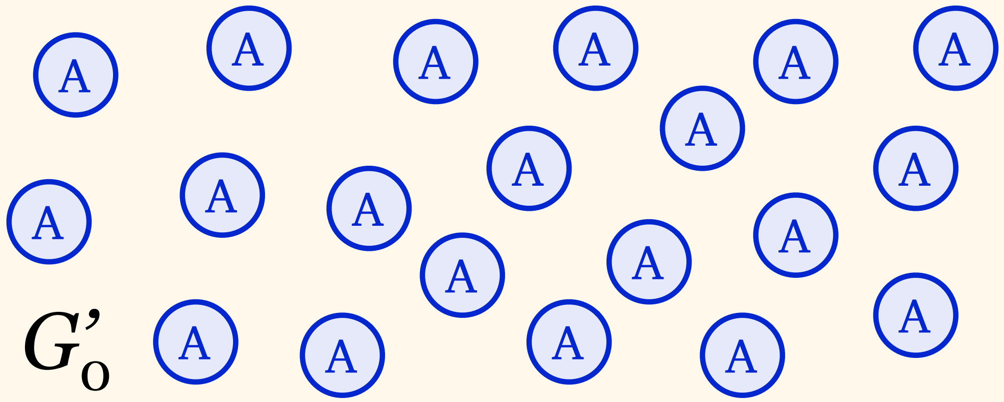


Find a 3-colouring
for each pseudoforest

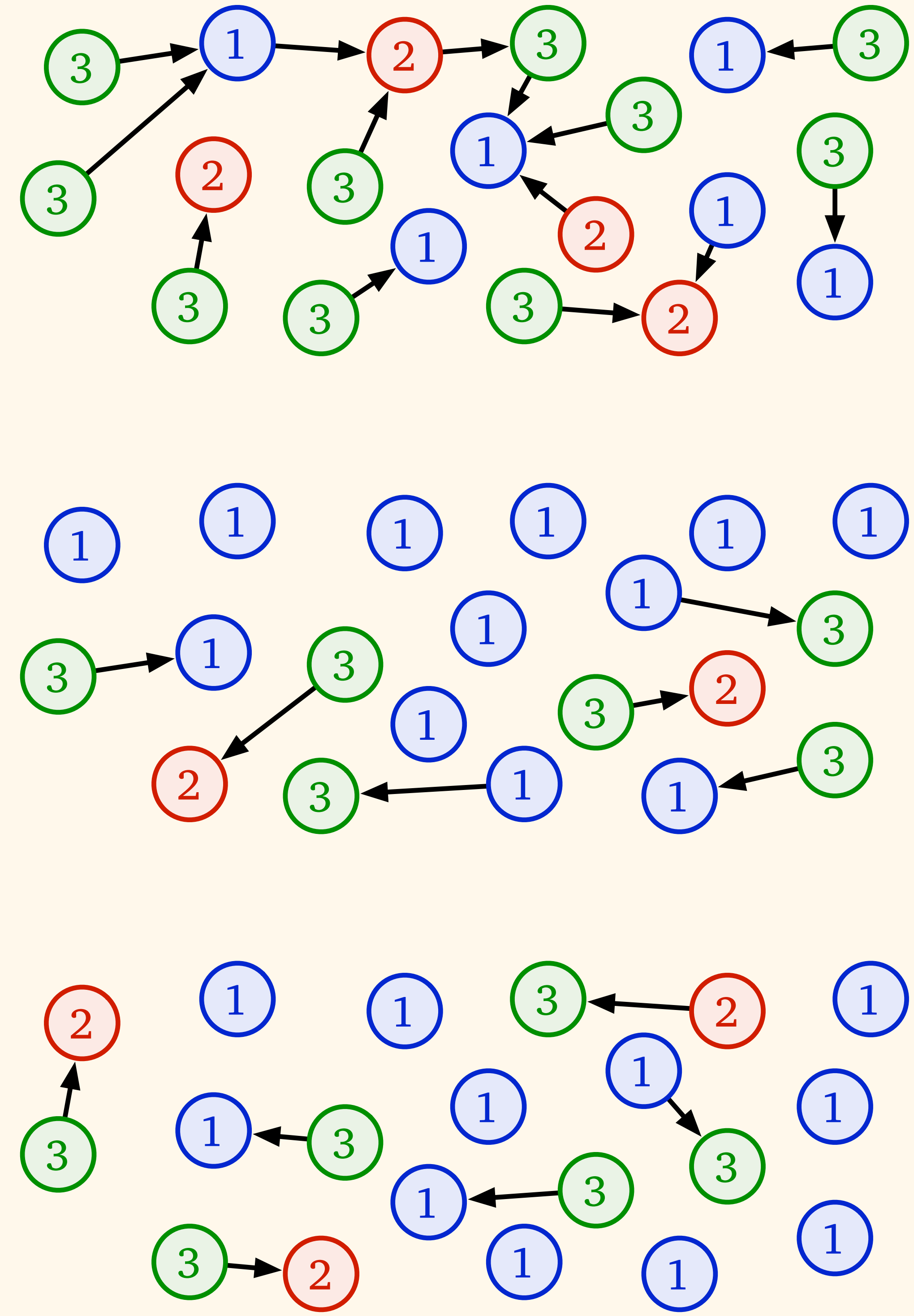
Computed in parallel,
simulate Δ instances of
the algorithm

Each node has Δ colours,
one for each forest



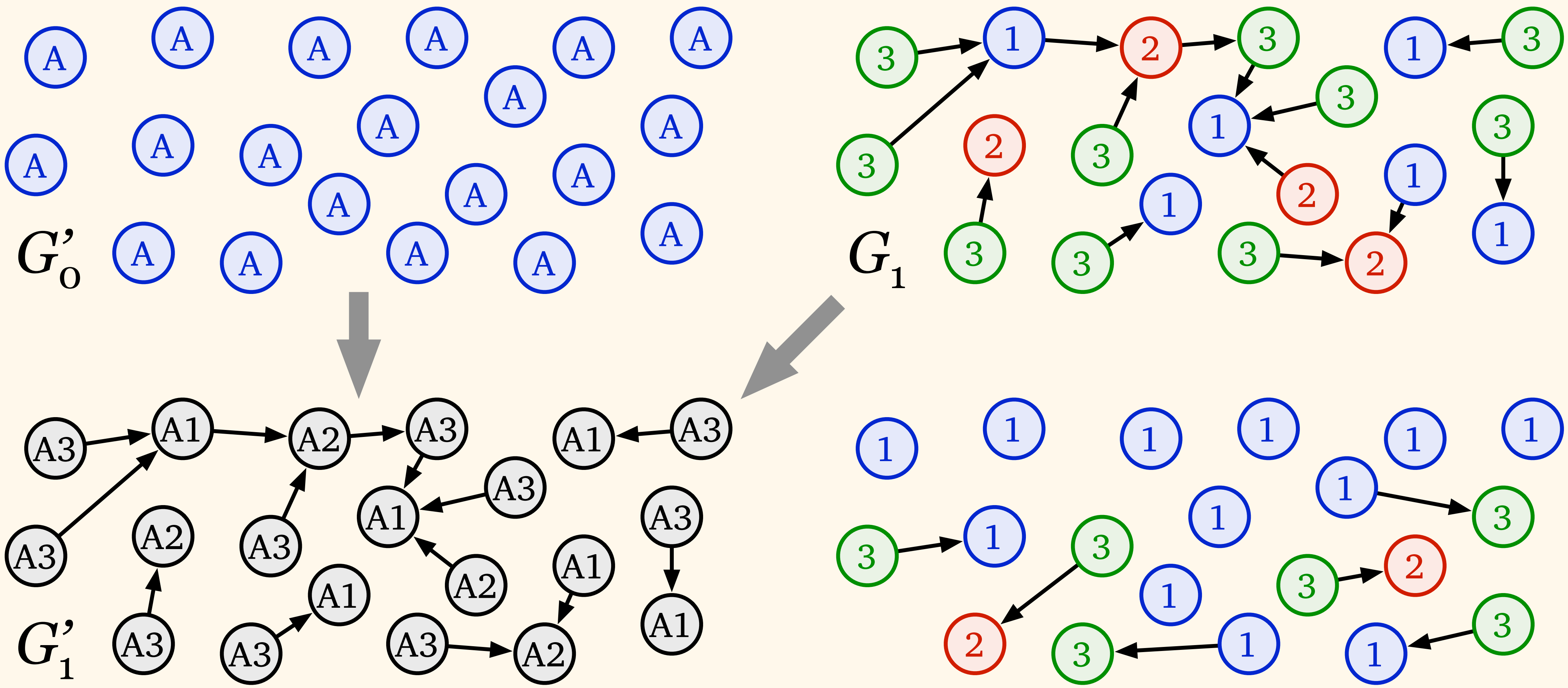


G'_0

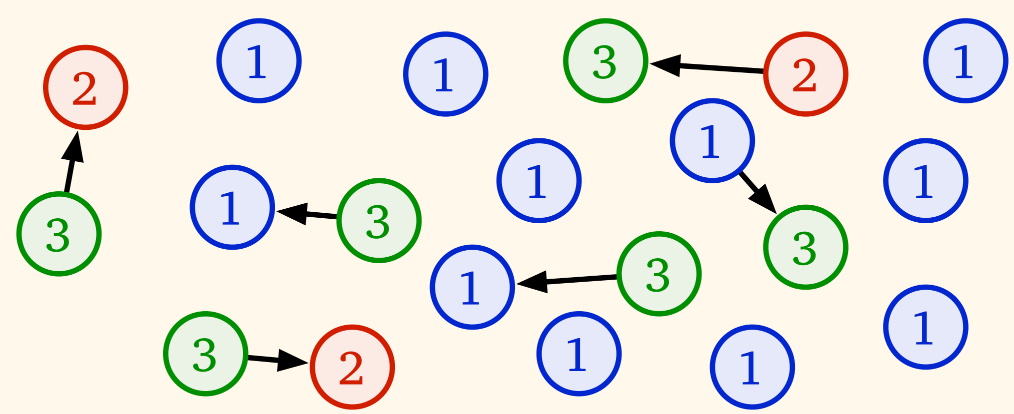


G'_0: (Δ+1)-coloured

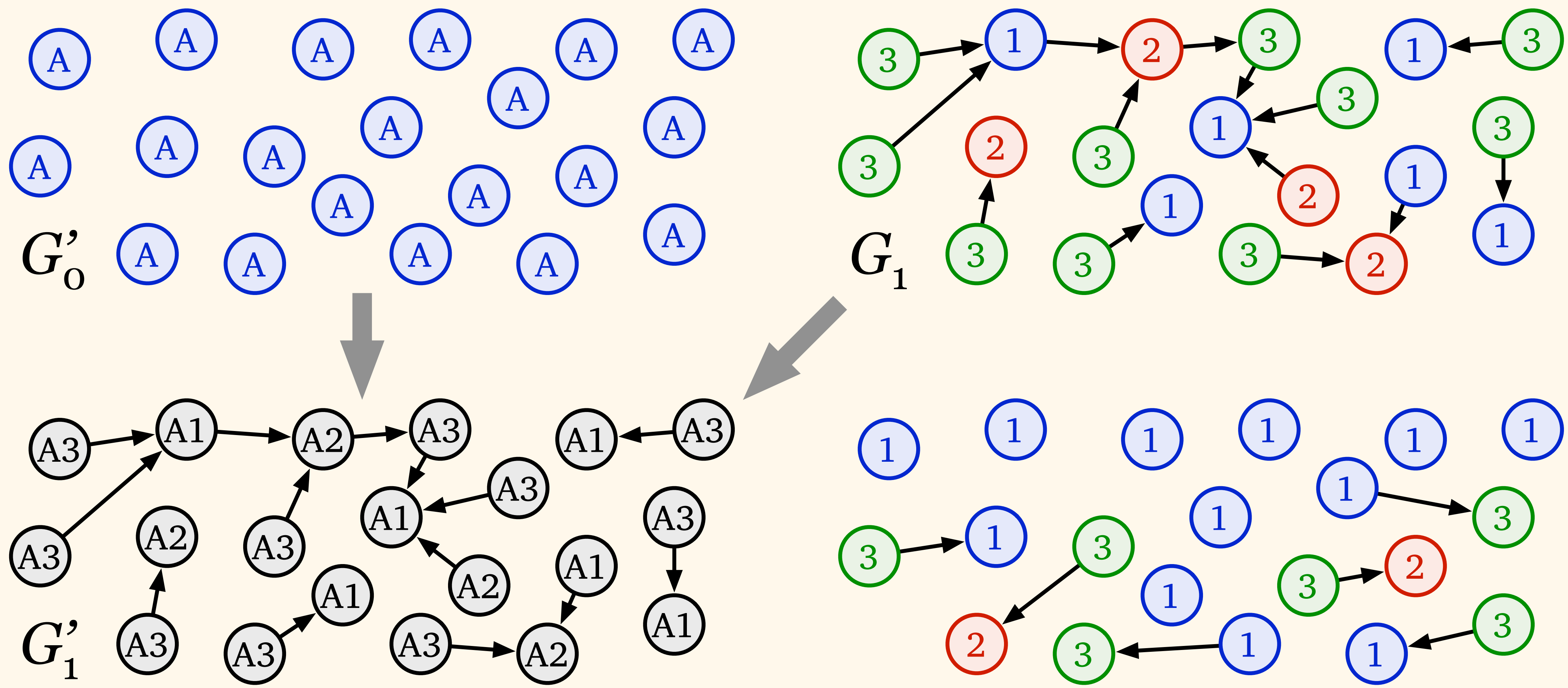
– trivial, no edges



union of edges,
combination of colours



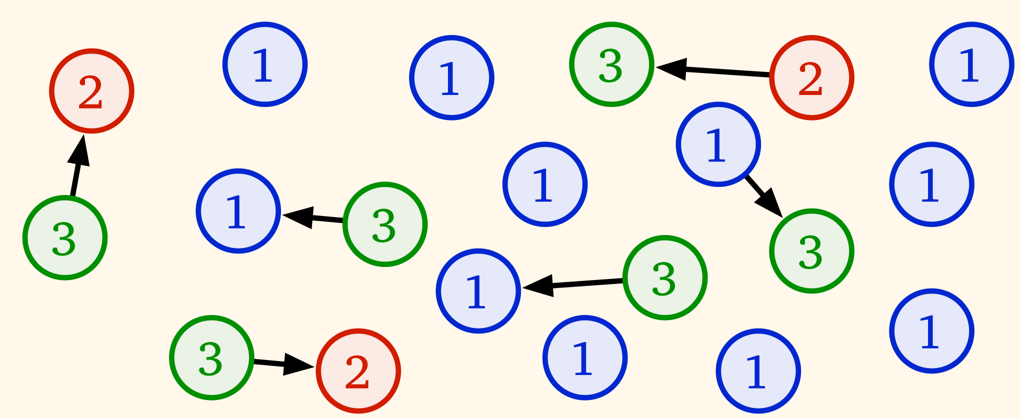
$$a + b \to (a, b)$$

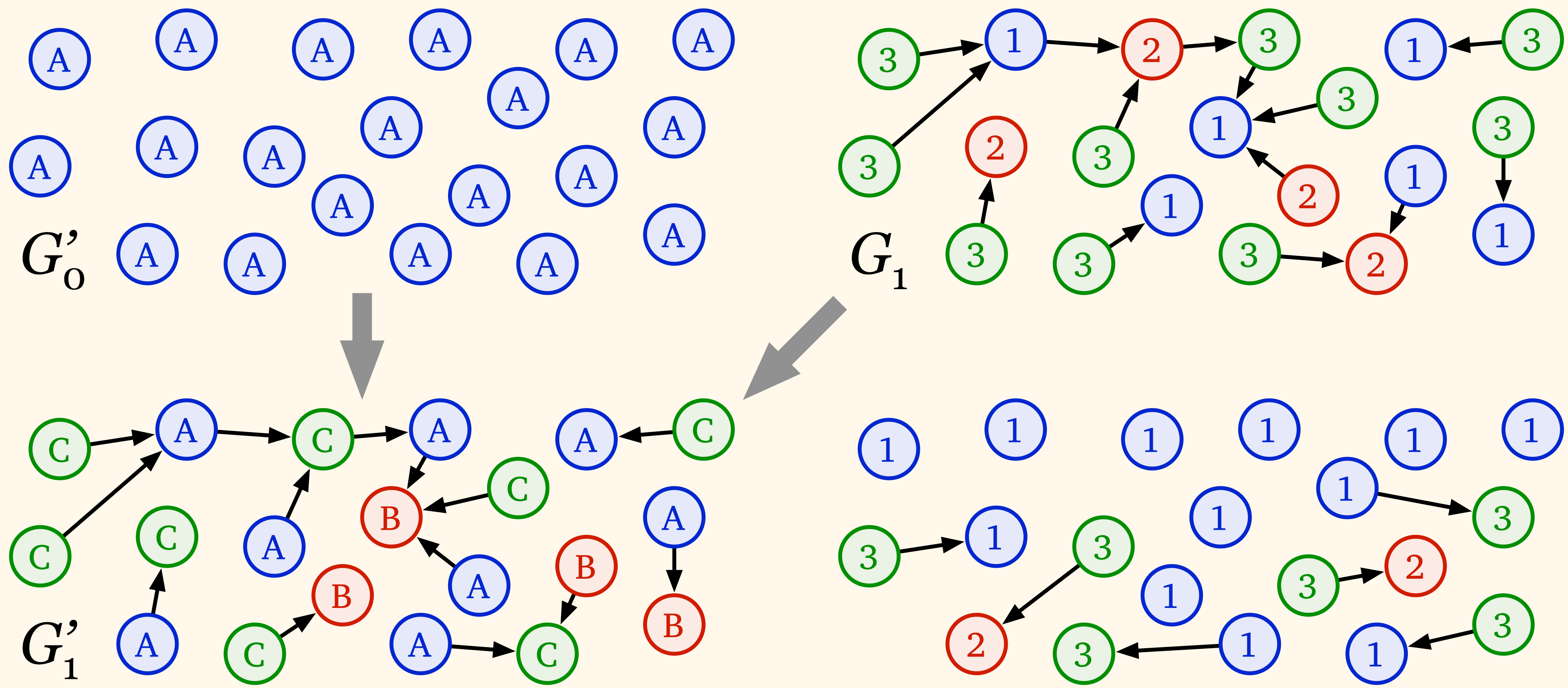


G'_0: $(\Delta+1)$-coloured

G_1: 3-coloured

G'_1: $3(\Delta+1)$-coloured



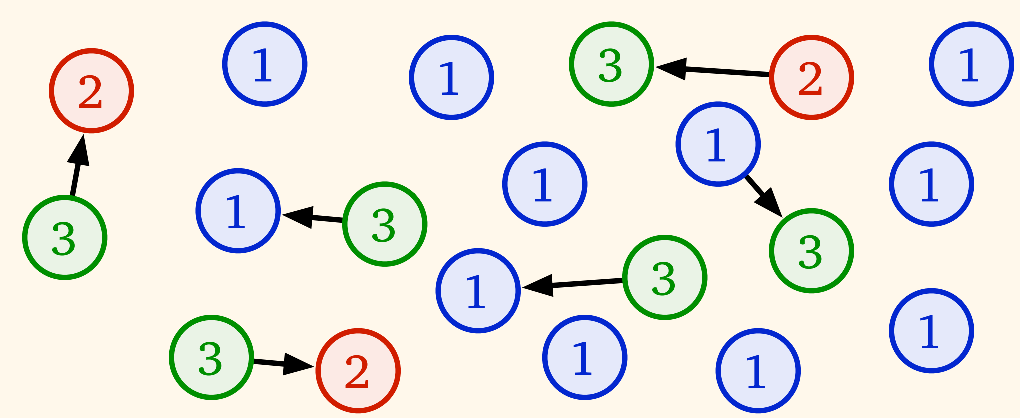


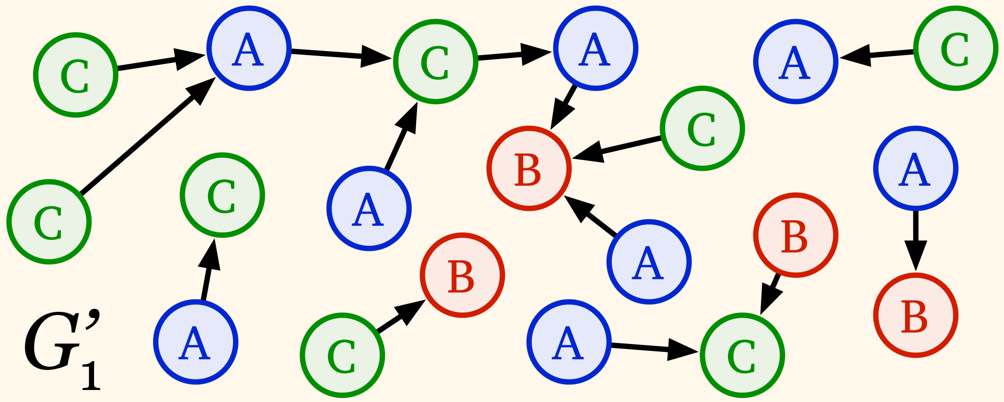
G'_0: $(\Delta+1)$-coloured

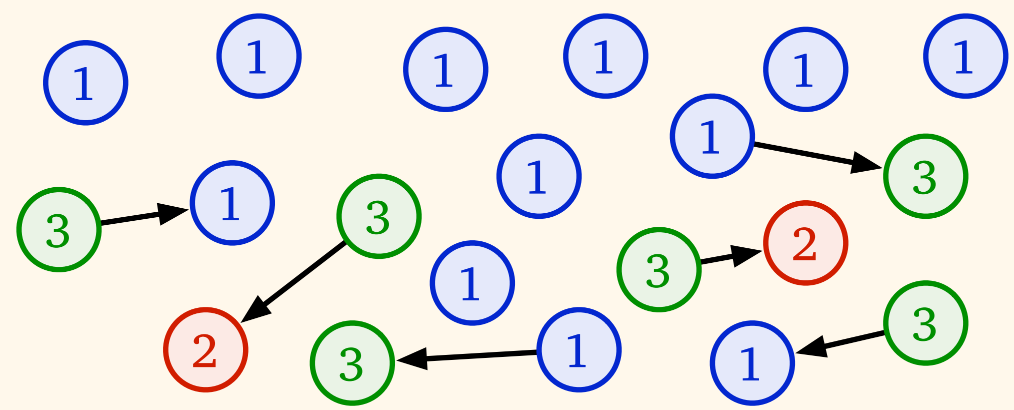
G_1: 3-coloured

G'_1: $3(\Delta+1)$-coloured,

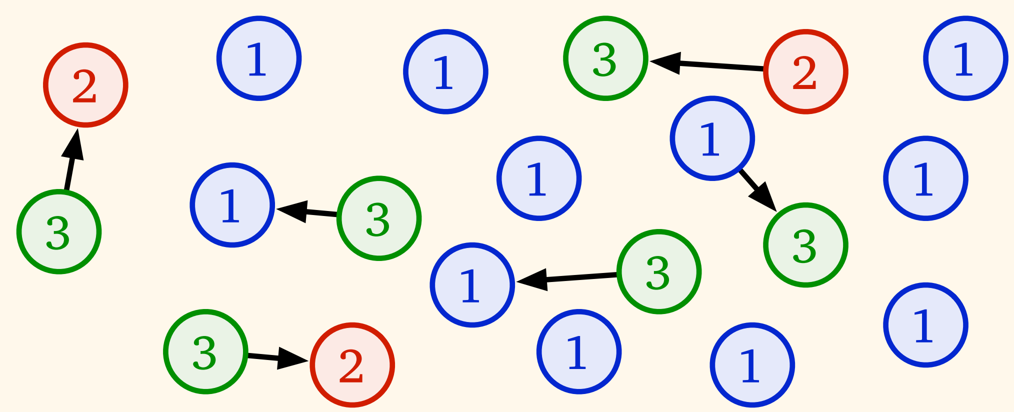
reduce to $\Delta+1$ greedily

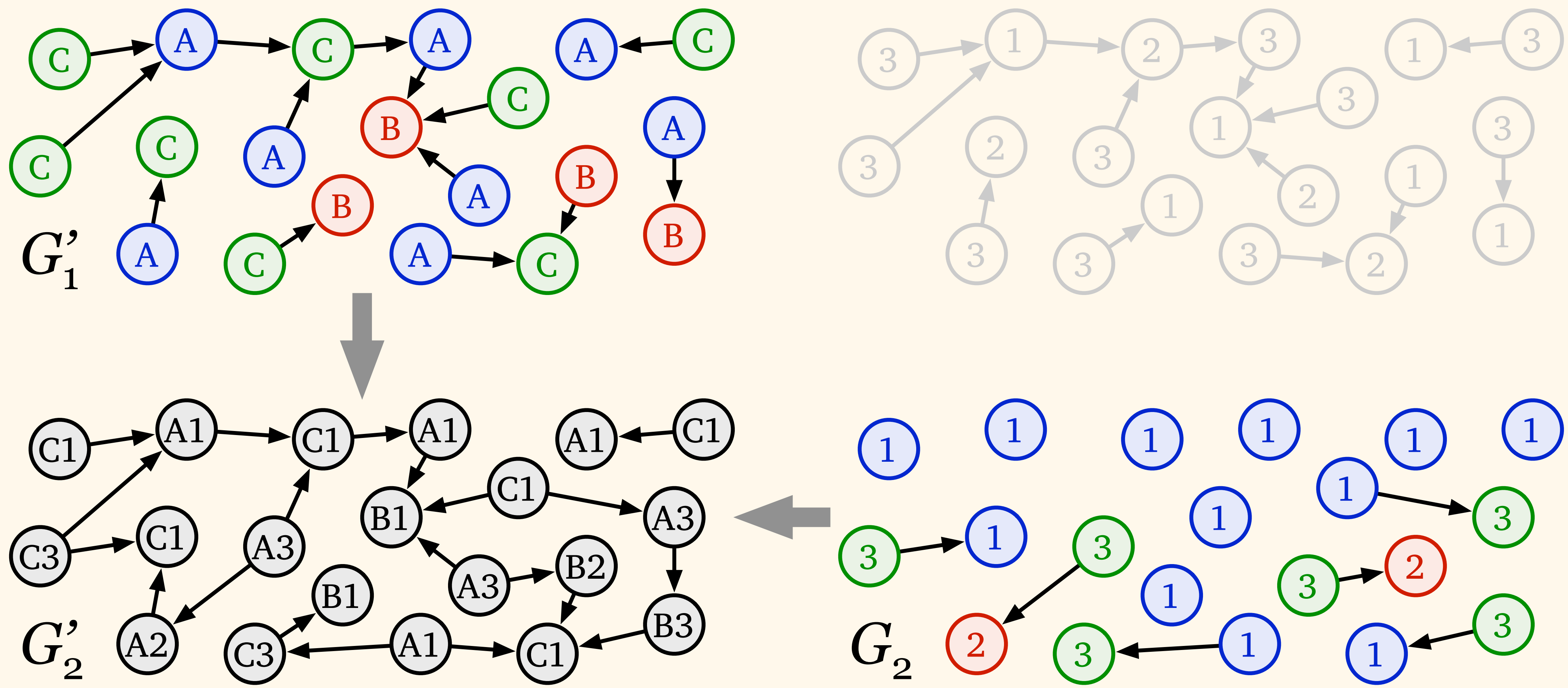






G'_1: $(\Delta+1)$-coloured

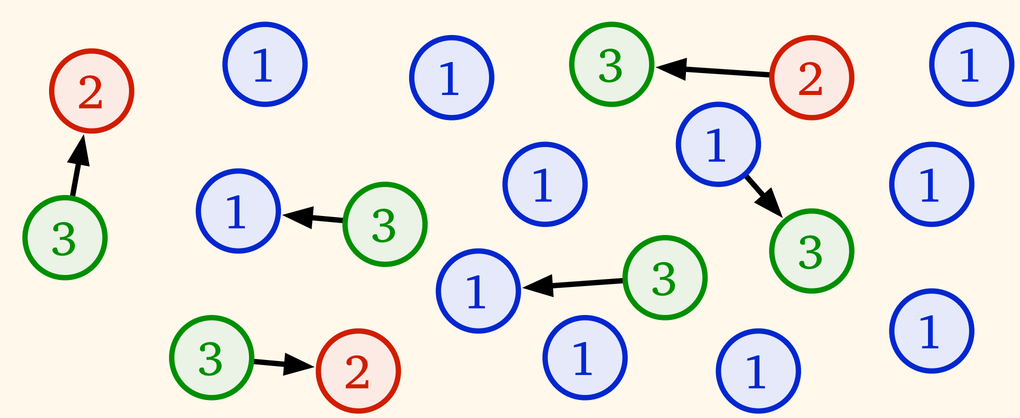


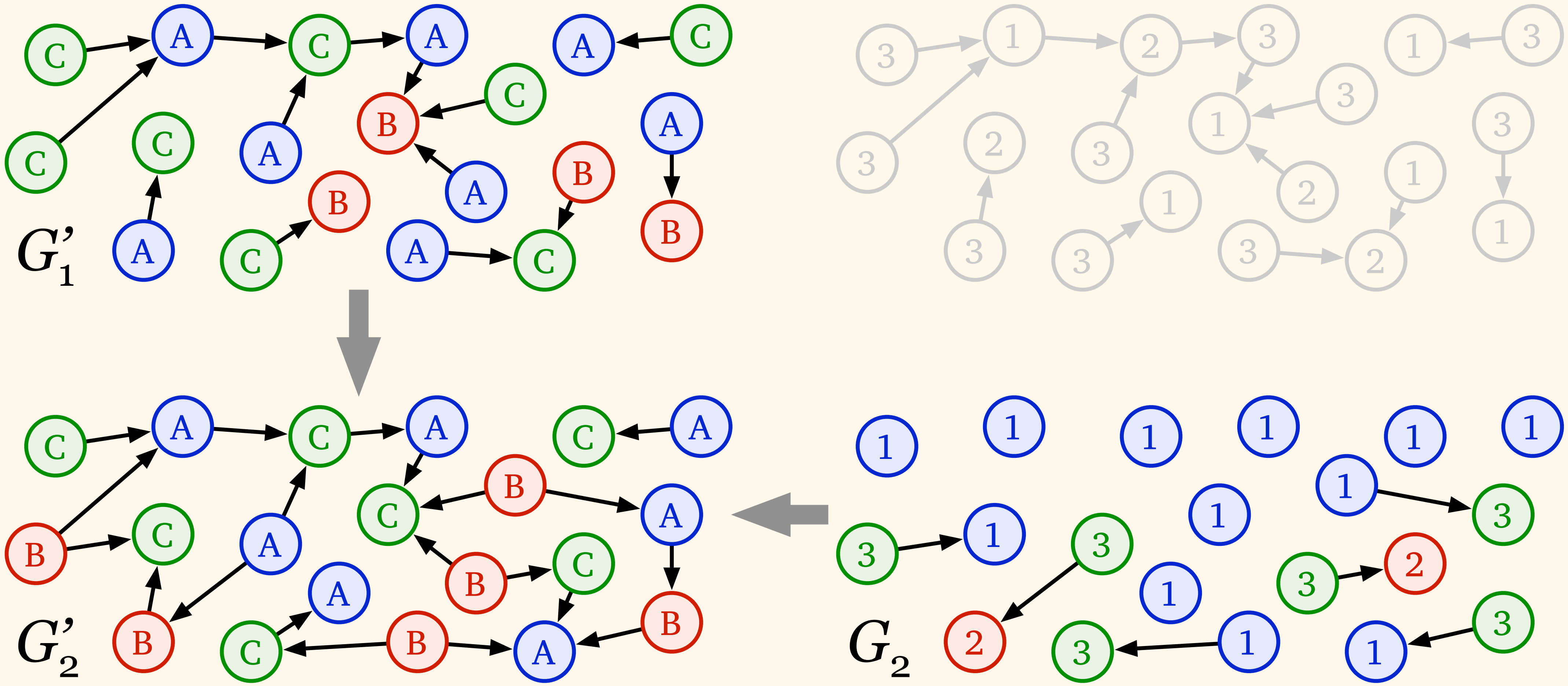


G'_1: $(\Delta+1)$-coloured

G_2: 3-coloured

G'_2: $3(\Delta+1)$-coloured



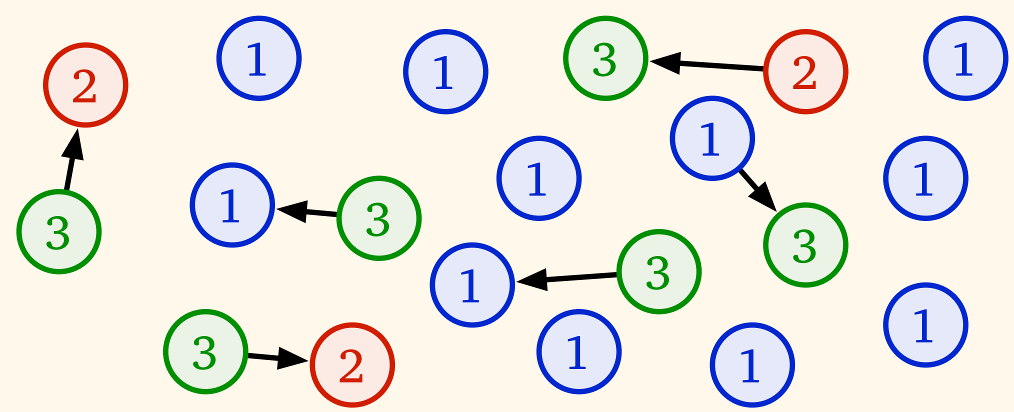


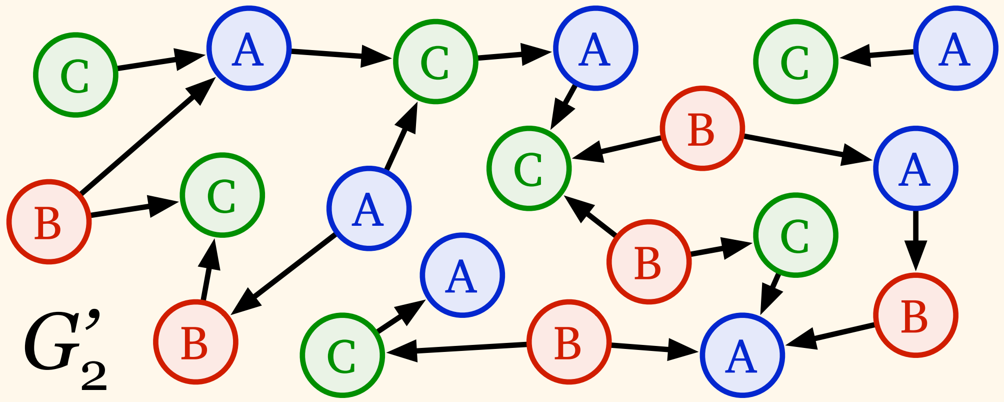
G'_1: $(\Delta+1)$-coloured

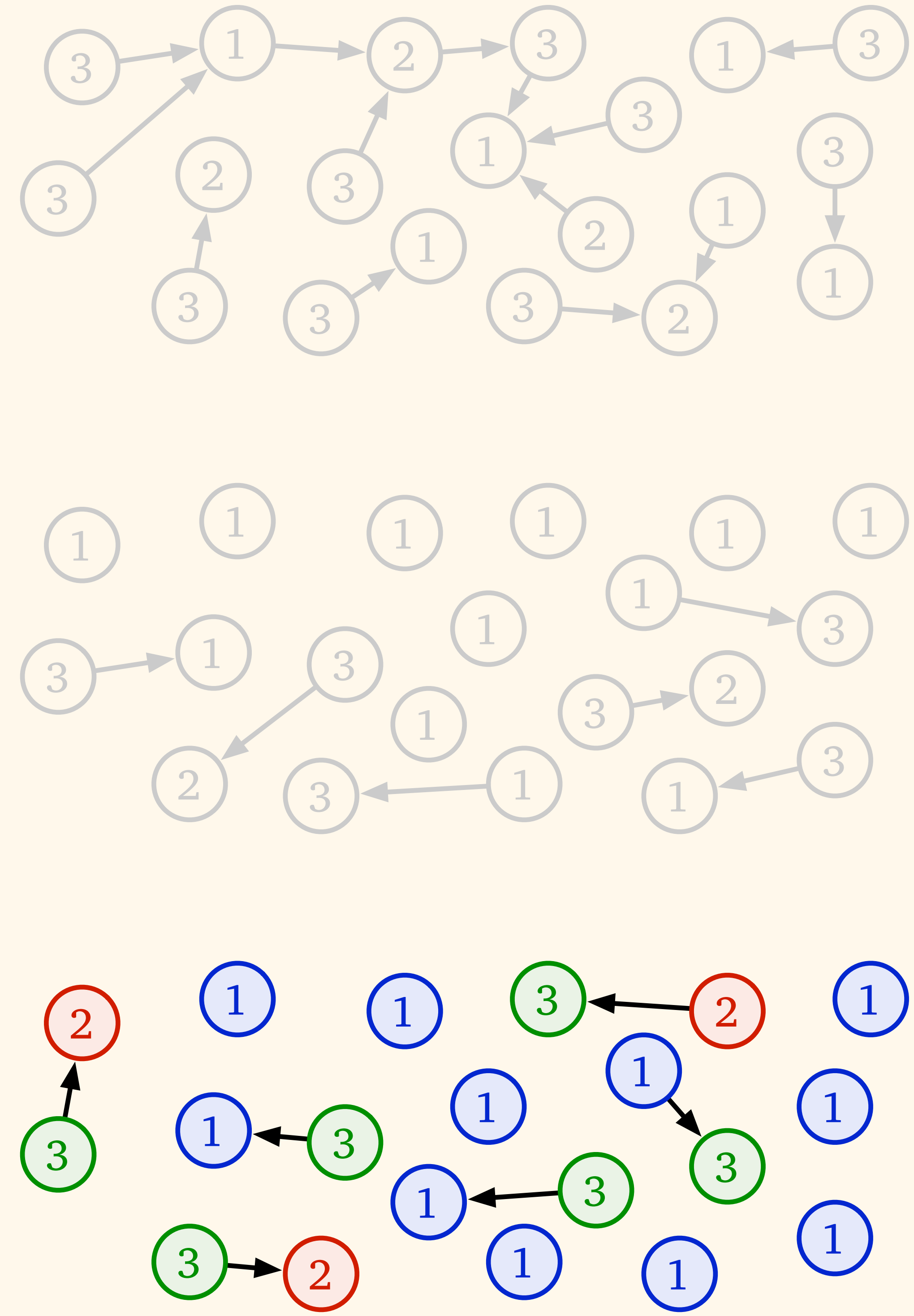
G_2: 3-coloured

G'_2: $3(\Delta+1)$-coloured,

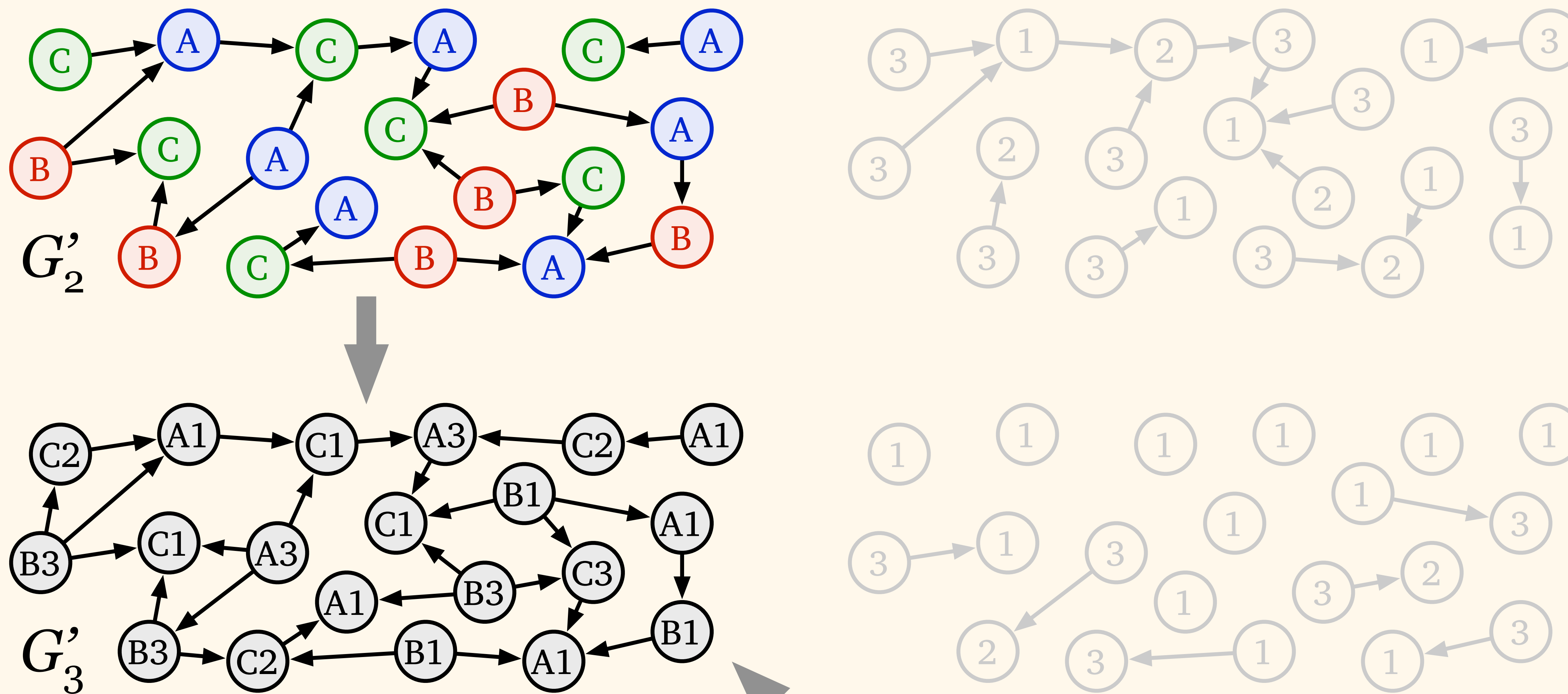
reduce to $\Delta+1$ greedily

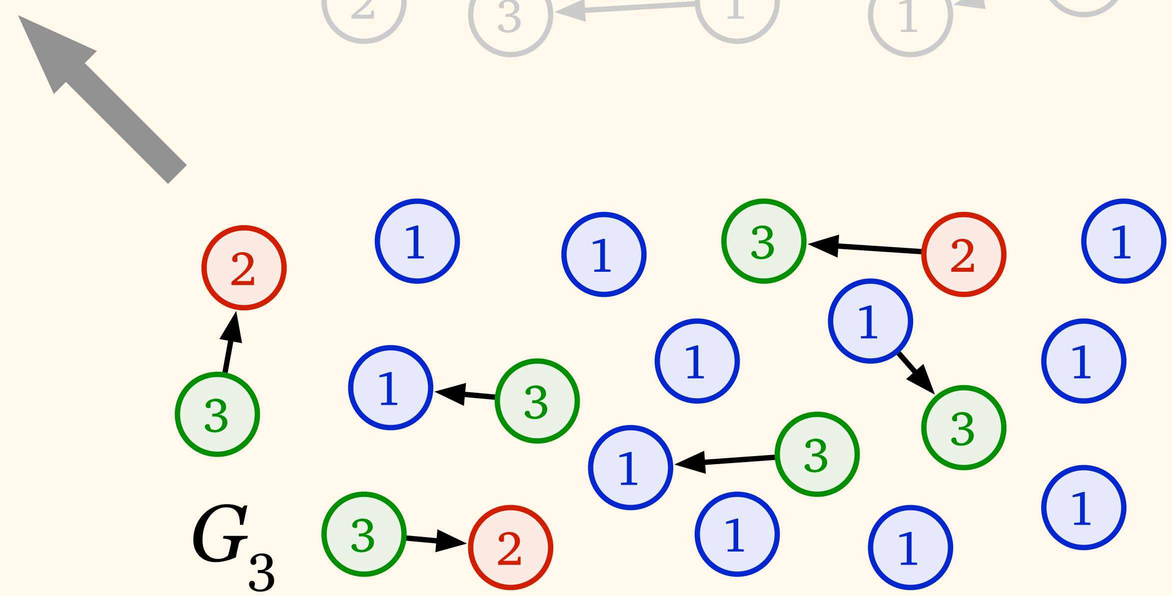






G_2': $(\Delta+1)$-coloured

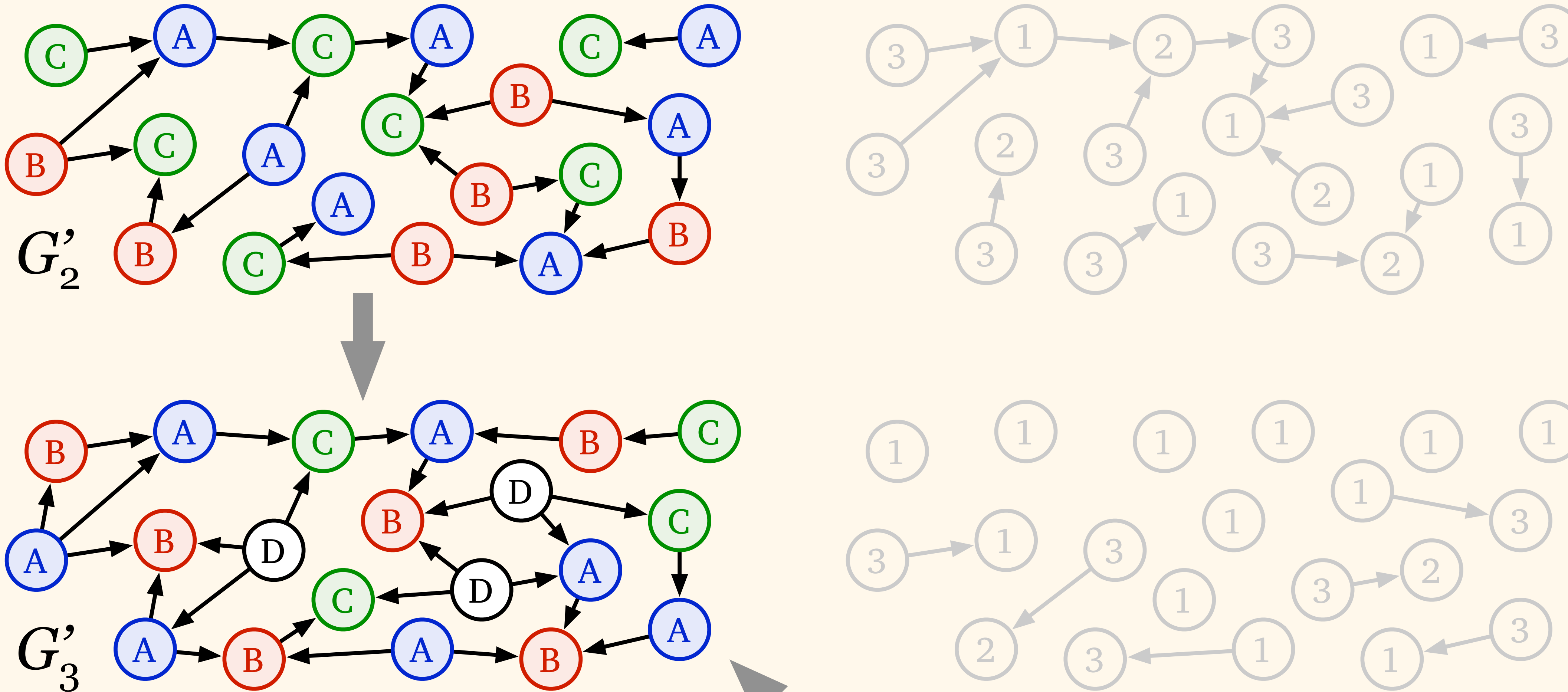




G'_2: $(\Delta+1)$-coloured

G_3: 3-coloured

G'_3: $3(\Delta+1)$-coloured

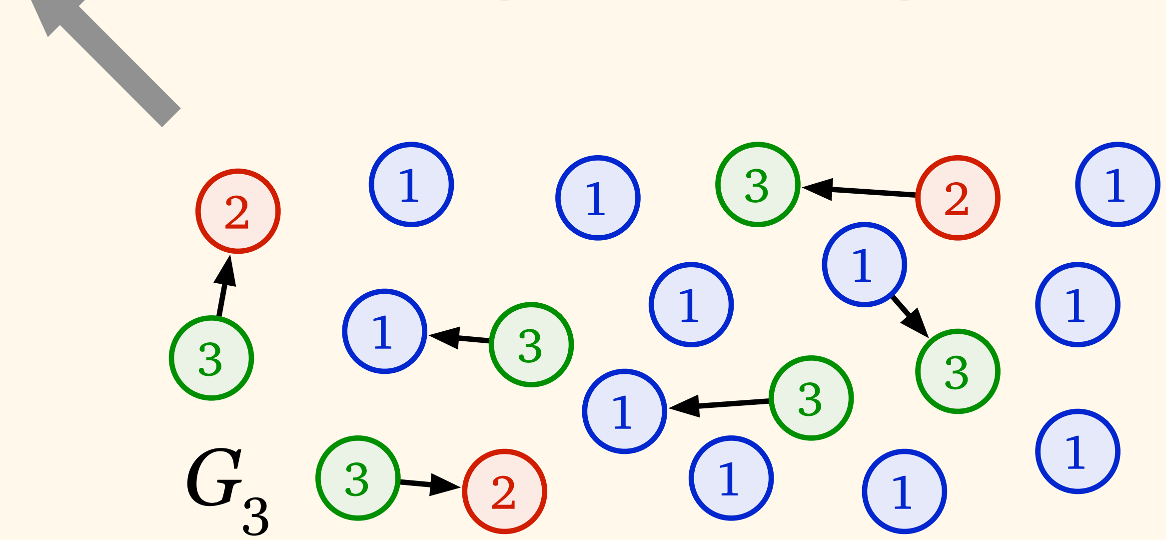


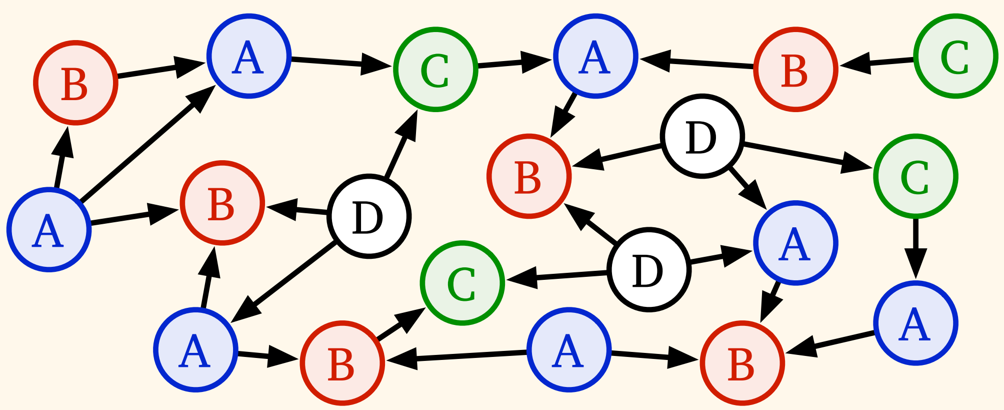
G'_2: $(\Delta+1)$-coloured

G_3: 3-coloured

G'_3: $3(\Delta+1)$-coloured,

reduce to $\Delta+1$ greedily





(Δ+1)-colouring of the original graph

Fast Graph Colouring

- Colour reduction from x to Δ+1
  - orientation: 1 round
  - partition: 0 rounds
  - 3-colouring: $O(\log^* x)$ rounds — see Exercise 5.4
  - Δ phases:
    - merge & reduce $3(\Delta+1) \to \Delta+1$: $2(\Delta+1)$ rounds
  - **total**: $O(\Delta^2 + \log^* x)$ rounds

Fast Graph Colouring

- Colour reduction from x to $\Delta+1$
  - $O(\Delta^2 + \log^* x)$ rounds
- Plenty of applications — see exercises
- Similar techniques can be used to solve other problems

Fast Graph Colouring

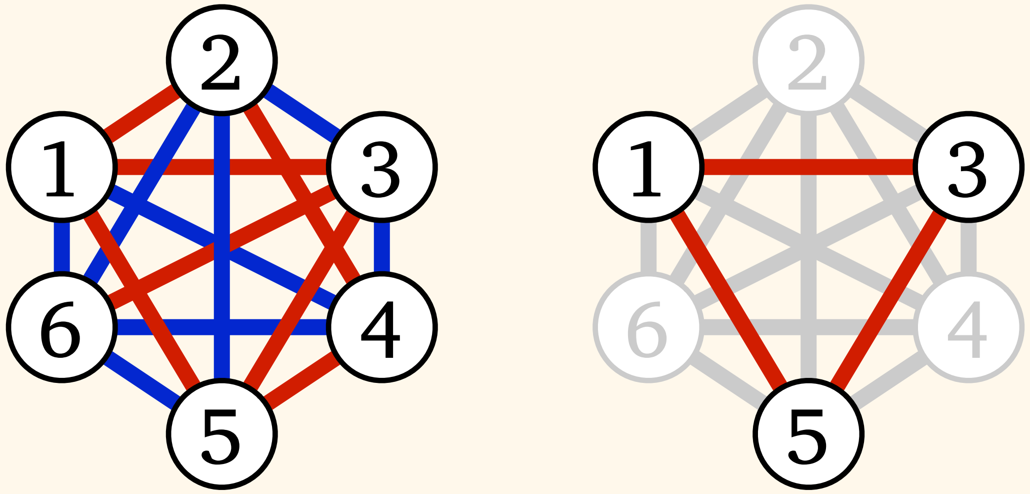
- Colour reduction from x to $\Delta+1$
  - $O(\Delta^2 + \log^* x)$ rounds
- Fast, but running time depends on x
- Next week:
  - dependence on x is necessary
  - even if $\Delta = 2$, we cannot reduce the number of colours from x to 3 in constant time, independently of x

Ramsey Theory

DDA Course

Lecture 6.1

24 April 2012



ON A PROBLEM OF FORMAL LOGIC

By F. P. RAMSEY.

[Received 28 November, 1928.—Read 13 December, 1928.]

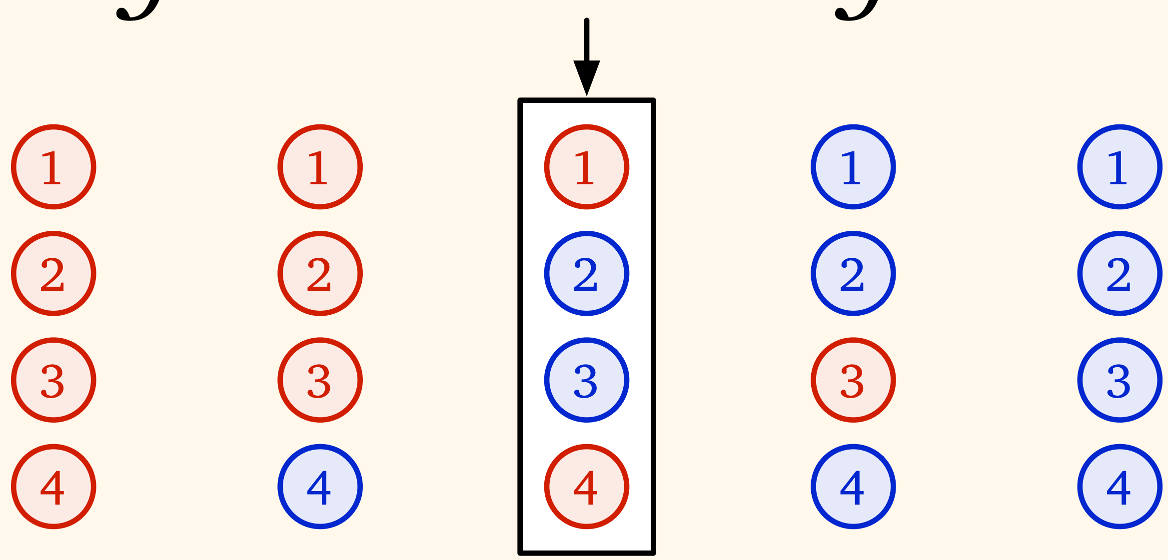
This paper is primarily concerned with a special case of one of the leading problems of mathematical logic, the problem of finding a regular procedure to determine the truth or falsity of any given logical formula*. But in the course of this investigation it is necessary to use certain theorems on combinations which have an independent interest and are most conveniently set out by themselves beforehand.

"… certain theorems on combinations which have an independent interest…"

Pigeonhole Principle

$N = 4$ items, colour each of them red or blue

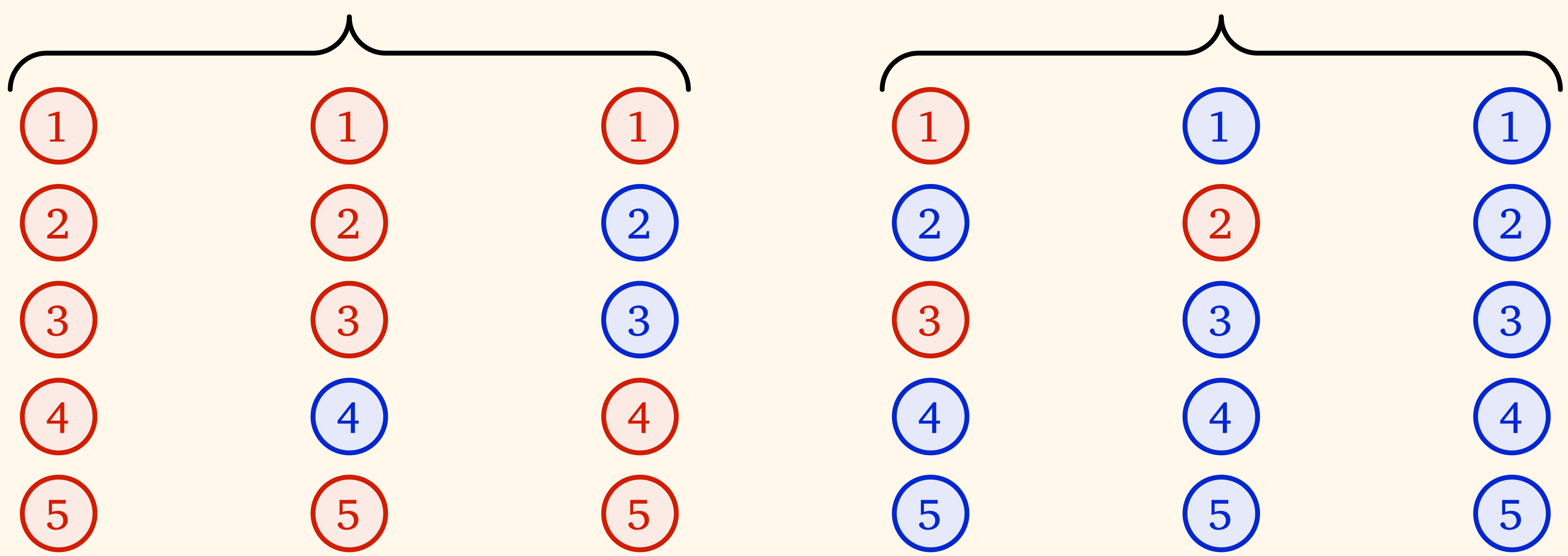
Possible: *only 2 red and only 2 blue*



Pigeonhole Principle

$N = 5$ items, colour each of them red or blue

Always: *at least 3 red* or *at least 3 blue*



Pigeonhole Principle

- Let $n = 3$
- N items, colour each of them red or blue
- If N is large enough, there are always
  - at least n red items or
  - at least n blue items
- Here $N \geq 5$ is sufficient, $N < 5$ is not

Pigeonhole Principle

- Let n be anything
- N items, colour each of them red or blue
- If N is large enough, there are always
  - at least n red items or
  - at least n blue items
- Here $N \geq 2n - 1$ is sufficient

Ramsey Theory

- Generalisation of pigeonhole principle
- Again, we have N items
- However, we will not colour items, we will colour ***sets*** of items
  - example: we colour all 2-subsets of items
  - "k-subset" = subset of size k

Ramsey Theory

- Y: set with N items
  - $N = 4$: $Y = \{1, 2, 3, 4\}$
- f: colouring of k-subsets of Y
  - $k = 2$: $f(\{1, 2\})$ = red, $f(\{1, 3\})$ = blue, ...
- $X \subseteq Y$ is ***monochromatic*** if all k-subsets of X have the same colour

$N = 4, \ Y = \{1, 2, \ldots, N\}, \ k = 2$

Colour each 2-subset of Y:

1, 2 1, 3 1, 4 2, 3 2, 4 3, 4

{1, 2, 3} is not monochromatic:

1, 2 1, 3 2, 3

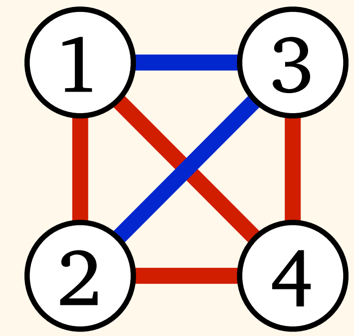
{1, 2, 4} is **monochromatic**:

1, 2 1, 4 2, 4

$N = 4,\ \ Y = \{1, 2, \ldots, N\},\ \ k = 2$

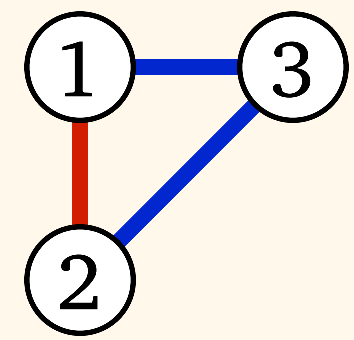
Colour each 2-subset of Y:

1, 2 | 1, 3 | 1, 4 | 2, 3 | 2, 4 | 3, 4



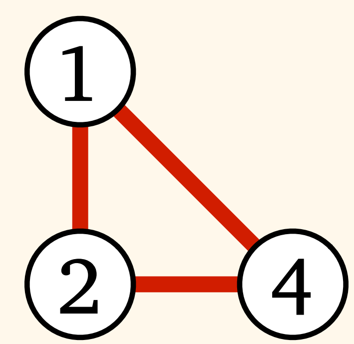
$\{1, 2, 3\}$ is not monochromatic:

1, 2 | 1, 3 | 2, 3



$\{1, 2, 4\}$ is **monochromatic**:

1, 2 | 1, 4 | 2, 4



Ramsey Theory

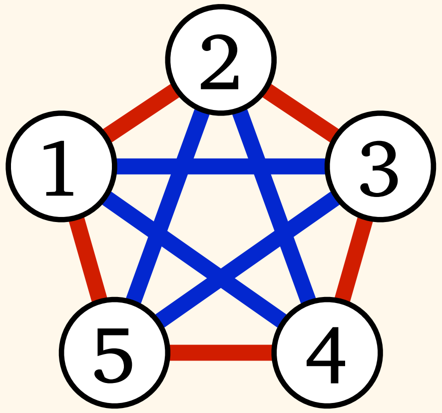
- Let $n = 3, \; k = 2$
- N items, colour each k-subset red or blue
- **Claim**: if N is sufficiently large, there is always a monochromatic subset of size n

$N = 5$, $Y = \{1, 2, \ldots, N\}$, $k = 2$

Colour each 2-subset of Y:

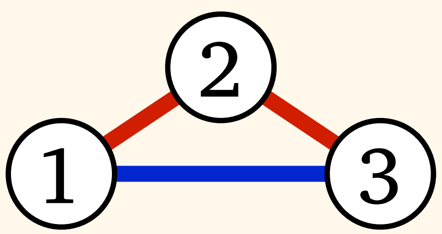
1, 2 | 1, 3 | 1, 4 | 1, 5 | 2, 3

2, 4 | 2, 5 | 3, 4 | 3, 5 | 4, 5



$\{1, 2, 3\}$ is not monochromatic:

1, 2 | 1, 3 | 2, 3



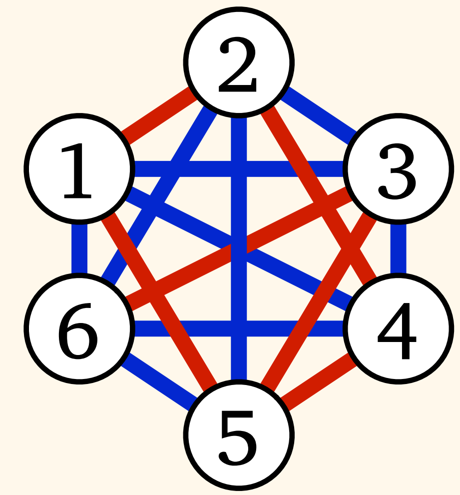
Check all possibilities:

there is no monochromatic subset of size 3

$N = 6,\ \ Y = \{1, 2, \ldots, N\},\ \ k = 2$

Colour each 2-subset of Y:

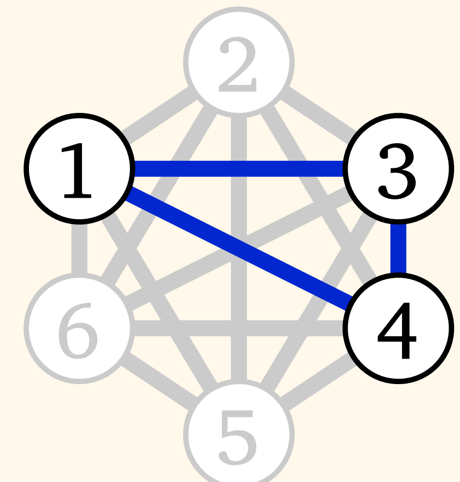
1, 2	1, 3	1, 4	1, 5	1, 6
	2, 3	2, 4	2, 5	2, 6
		3, 4	3, 5	3, 6
			4, 5	4, 6
				5, 6



$N = 6$, $Y = \{1, 2, \ldots, N\}$, $k = 2$

Colour each 2-subset of Y:

1, 2	**1, 3**	**1, 4**	1, 5	1, 6
	2, 3	2, 4	2, 5	2, 6
		3, 4	3, 5	3, 6
			4, 5	4, 6
				5, 6

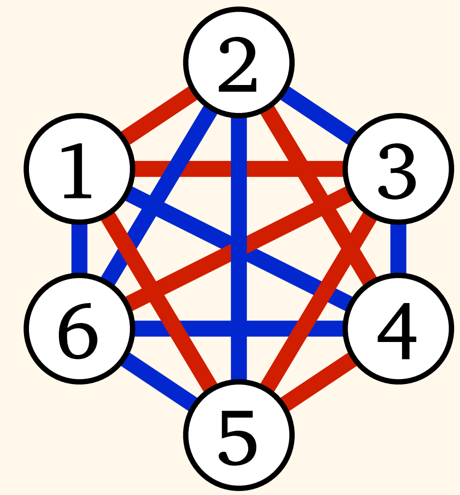


$\{1, 3, 4\}$ is monochromatic

$N = 6,\ \ Y = \{1, 2, \ldots, N\},\ \ k = 2$

Colour each 2-subset of Y:

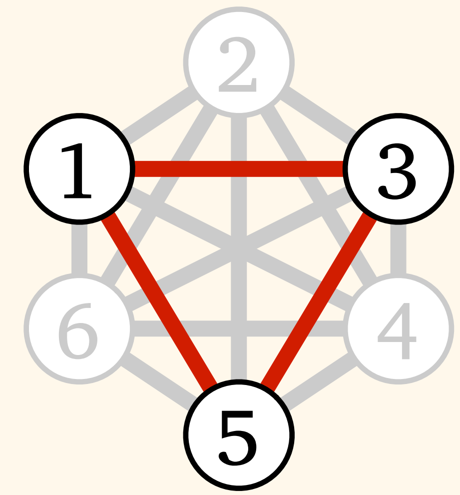
1, 2	1, 3	1, 4	1, 5	1, 6
	2, 3	2, 4	2, 5	2, 6
		3, 4	3, 5	3, 6
			4, 5	4, 6
				5, 6



$N = 6,\ \ Y = \{1, 2, \ldots, N\},\ \ k = 2$

Colour each 2-subset of Y:

1, 2	1, 3	1, 4	1, 5	1, 6
	2, 3	2, 4	2, 5	2, 6
		3, 4	3, 5	3, 6
			4, 5	4, 6
				5, 6



$\{1, 3, 5\}$ is monochromatic

Ramsey Theory

- Let $n = 3$, $k = 2$
- N items, colour each k-subset red or blue
- **Claim**: if N is sufficiently large, there is always a monochromatic subset of size n
  - $N = 5$ is not enough
  - it is possible to show that $N = 6$ is enough

Ramsey Theory

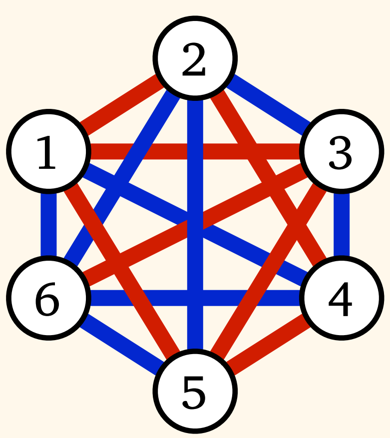
- Let n and k be any positive integers
- N items, colour each k-subset red or blue
- **Claim**: if N is sufficiently large, there is always a monochromatic subset of size n

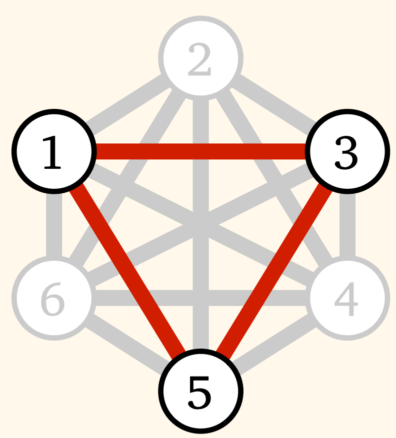
Ramsey Theory

- Let c, n, and k be any positive integers
- N items, colour each k-subset with a colour from $\{1, 2, \ldots, c\}$
- **Claim**: if N is sufficiently large, there is always a monochromatic subset of size n

Ramsey's Theorem

- **Theorem**: For all c, n, and k, there is a number $R_c(n; k)$ such that if you take $N \geq R_c(n; k)$ items, and colour each k-subset with one of c colours, there is always a monochromatic n-subset





$R_2(3; 2) = 6$

Ramsey's Theorem

- **Theorem**: For all c, n, and k, there is a number $R_c(n; k)$ such that if you take $N \geq R_c(n; k)$ items, and colour each k-subset with one of c colours, there is always a monochromatic n-subset
  - proof: see the course material
  - numbers $R_c(n; k)$ are called *Ramsey numbers*
  - examples: $R_2(3; 2) = 6$, $R_2(4; 2) = 18$

Ramsey's Theorem

- No matter how you colour subsets, if the base set is large enough, we can always find a monochromatic subset
- Our application: *no constant-time algorithm for 3-colouring directed cycles*
    - no matter how you design your algorithm, if the set of possible identifiers is large enough, we can always find a "bad input"

Colouring in Constant Time?

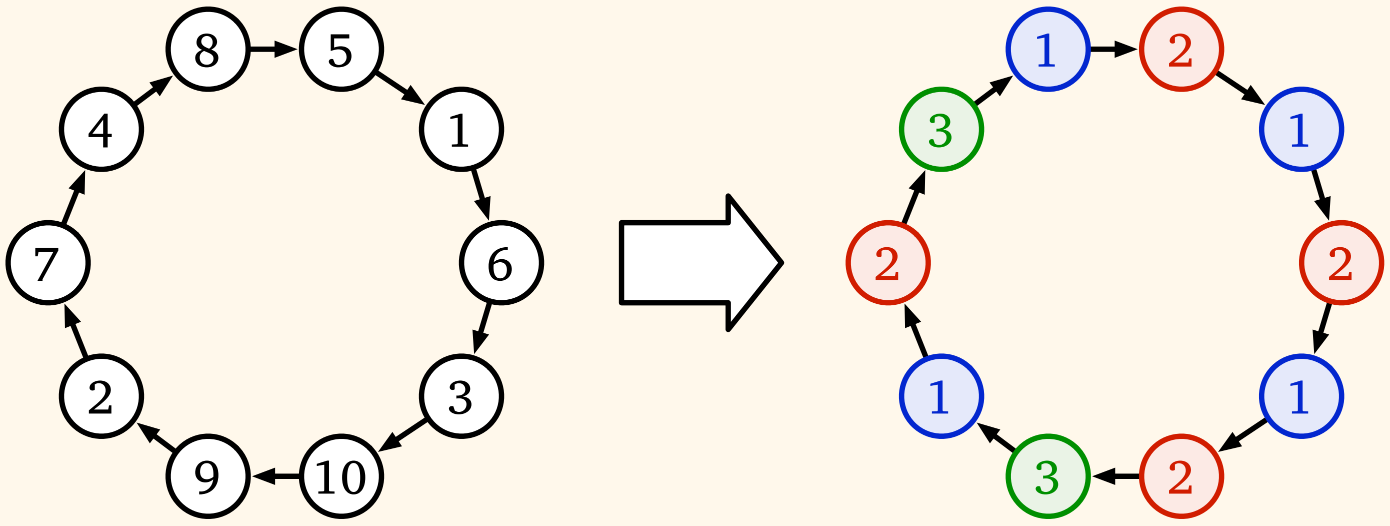
DDA Course

Lecture 6.2

26 April 2012

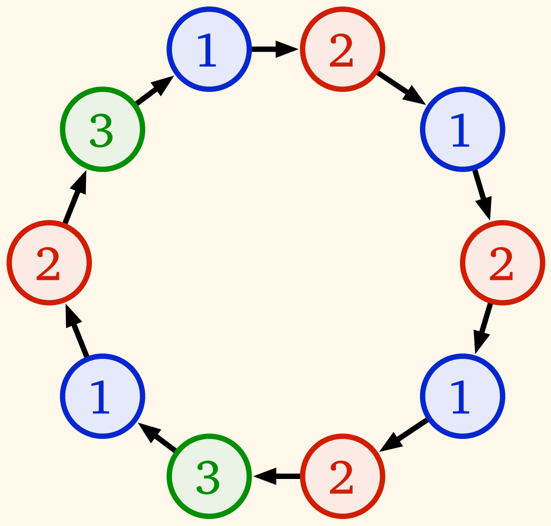
Colouring in Cycles

- Problem: 3-colouring in *directed cycles*
  - unique identifiers from {1, 2, … n}
  - outdegree = indegree = 1



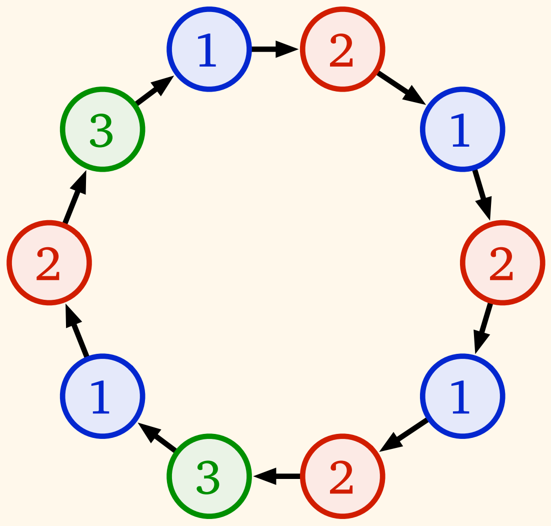
Colouring in Cycles

- Problem: 3-colouring in *directed cycles*
  - unique identifiers from $\{1, 2, \dots n\}$
  - outdegree = indegree = 1
- We know how to solve this problem in time $O(\log^* n)$
  - special case of directed pseudoforests



Colouring in Cycles

- Problem: 3-colouring in *directed cycles*
    - unique identifiers from $\{1, 2, \ldots n\}$
    - outdegree = indegree = 1
- We know how to solve this problem in time $O(\log^* n)$
- Can we do it in time $O(1)$?

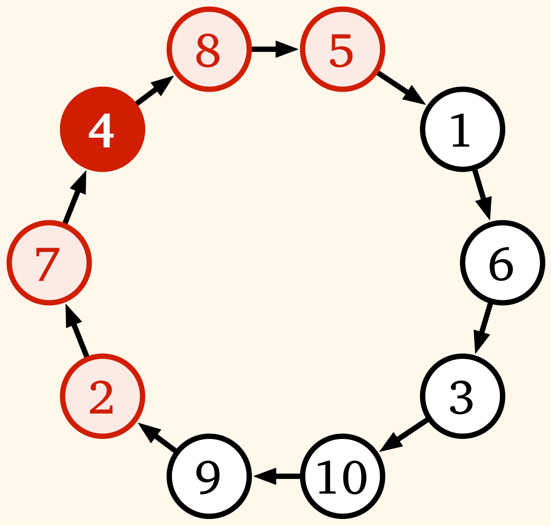


Ramsey Says No

- Assume that algorithm A:
    - in any directed cycle,
      stops in time T for some constant T
    - produces local outputs from {1, 2, 3}
- We will use Ramsey's theorem to show that there is a directed cycle in which A fails to produce a proper vertex colouring

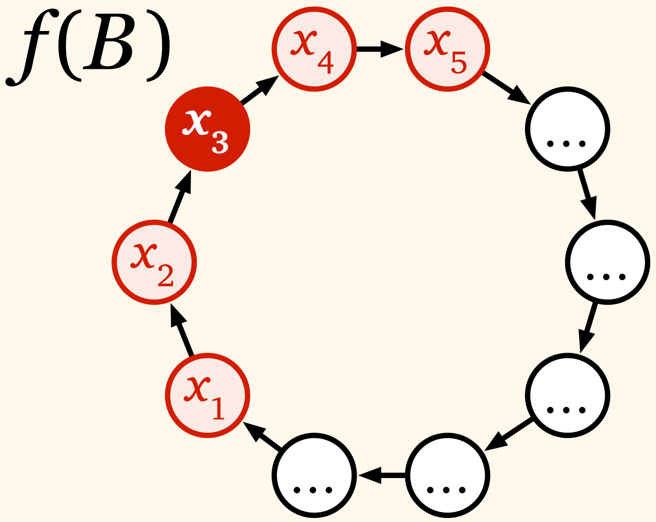
Ramsey Says No

- Example: algorithm runs in time $T = 2$
- Output of a node only depends on $k = 2T + 1 = 5$ nodes around it
  - choose $c = 3$, $n = k + 1 = 6$
  - choose $N \geq R_c(n; k)$
  - c-colour k-subsets of $\{1, 2, \ldots, N\}$: there is a monochromatic n-subset



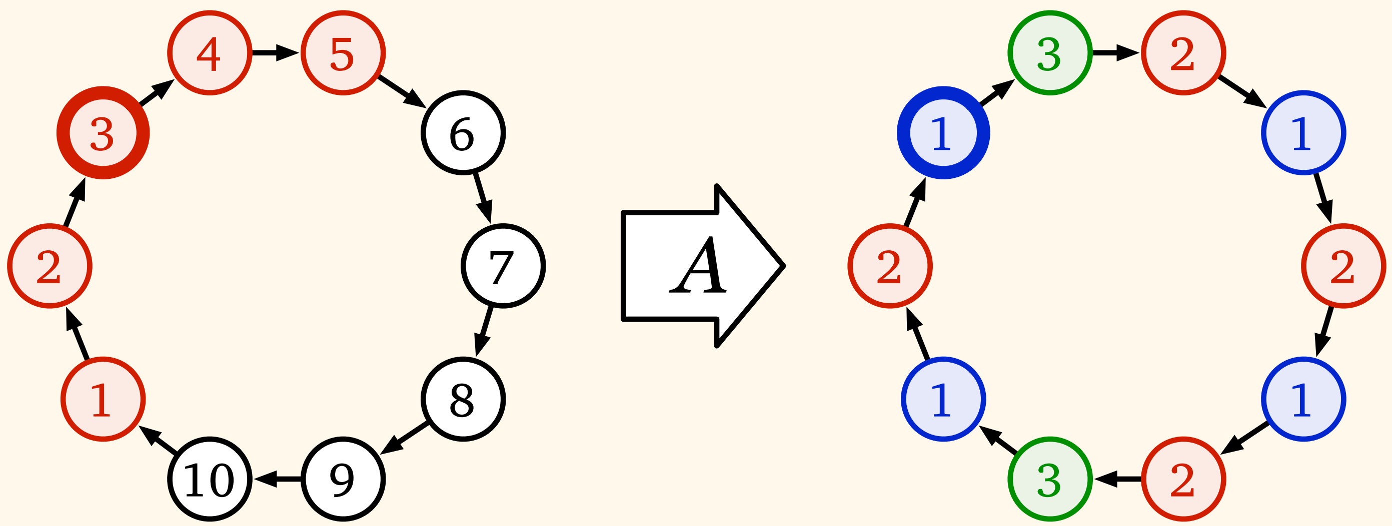
Ramsey Says No

- Set of identifiers: $Y = \{1, 2, \ldots N\}$
- We use algorithm A to colour k-subsets of Y
  - for each set $B = \{x_1, x_2, \ldots, x_k\} \subseteq Y$, $x_1 < x_2 < \ldots < x_k$
  - construct a cycle where nodes $x_1, x_2, \ldots, x_k$ are placed in this order
  - $f(B)$ = output of the middle node



Colour each k-subset of Y:

— what is the colour of $\{1, 2, 3, 4, 5\}$?



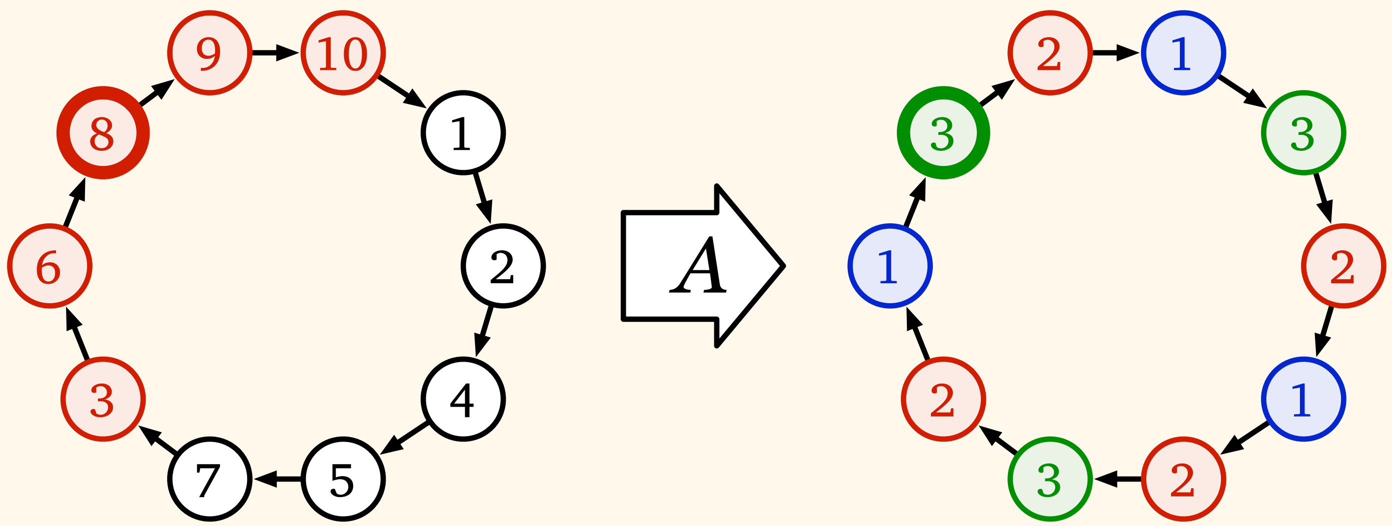
— middle node 3 outputs "blue"

— set $f(\{1, 2, 3, 4, 5\})$ = "blue"

1, 2, 3, 4, 5

Colour each k-subset of Y:

— what is the colour of {3, 6, 8, 9, 10}?



— middle node 8 outputs "green"

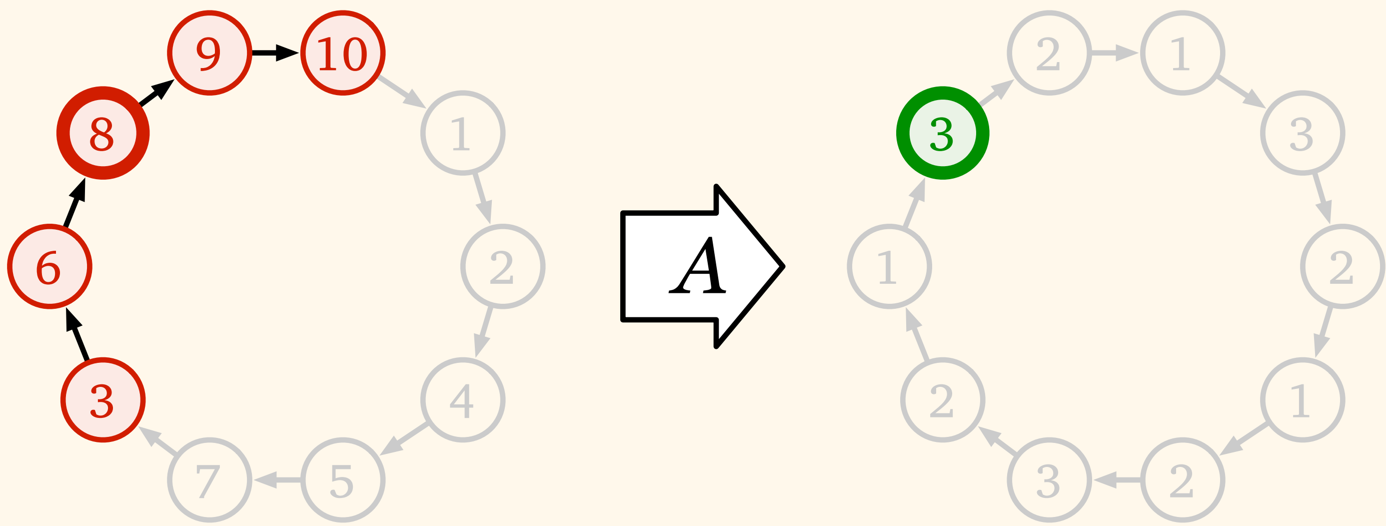
— set $f(\{3, 6, 8, 9, 10\})$ = "green"

1, 2, 3, 4, 5

3, 6, 8, 9, 10

Colour each k-subset of Y:

— what is the colour of {3, 6, 8, 9, 10}?



— middle node 8 outputs "green"

— set $f(\{3, 6, 8, 9, 10\})$ = "green"

1, 2, 3, 4, 5 | 3, 6, 8, 9, 10

Ramsey Says No

- We have assigned a colour $f(B) \in \{1, 2, 3\}$ to each k-subset B of Y

1, 2, 3, 4, 5	1, 2, 3, 4, 10	1, 2, 3, 5, 10
1, 2, 3, 4, 6	1, 2, 3, 5, 6	1, 2, 3, 6, 7
1, 2, 3, 4, 7	1, 2, 3, 5, 7	1, 2, 3, 6, 8
1, 2, 3, 4, 8	1, 2, 3, 5, 8	• • •
1, 2, 3, 4, 9	1, 2, 3, 5, 9	6, 7, 8, 9, 10

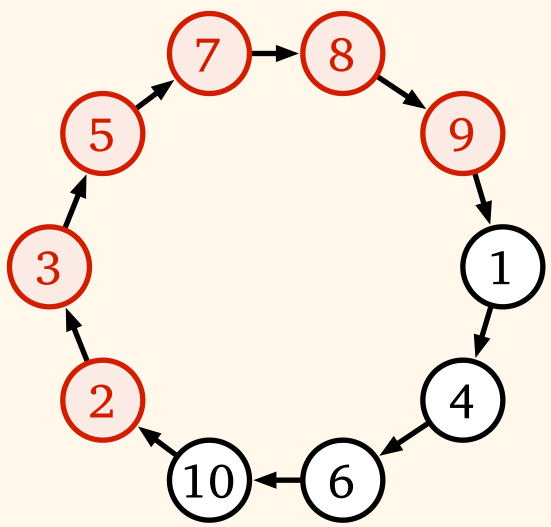
Ramsey Says No

- We have assigned a colour $f(B) \in \{1, 2, 3\}$ to each k-subset B of Y
- Ramsey: set Y was large enough, there is a monochromatic subset of size n
  - example: $\{2, 3, 5, 7, 8, 9\}$ is monochromatic

2, 3, 5, 7, 8	2, 3, 5, 8, 9	2, 3, 7, 8, 9
2, 3, 5, 7, 9	2, 5, 7, 8, 9	3, 5, 7, 8, 9

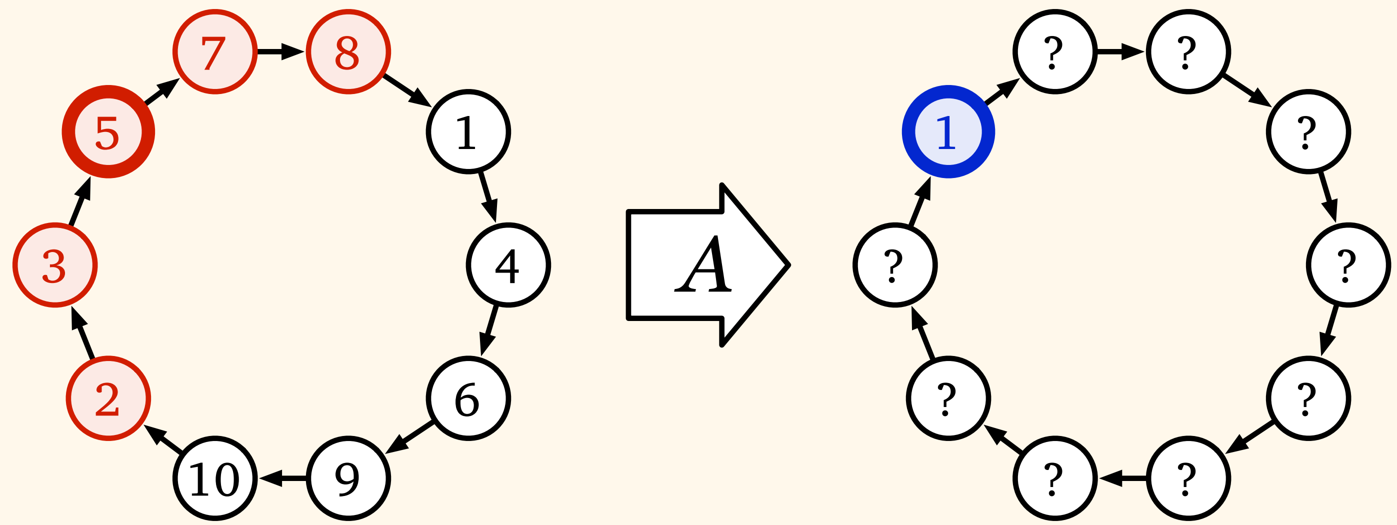
Ramsey Says No

What happens here?

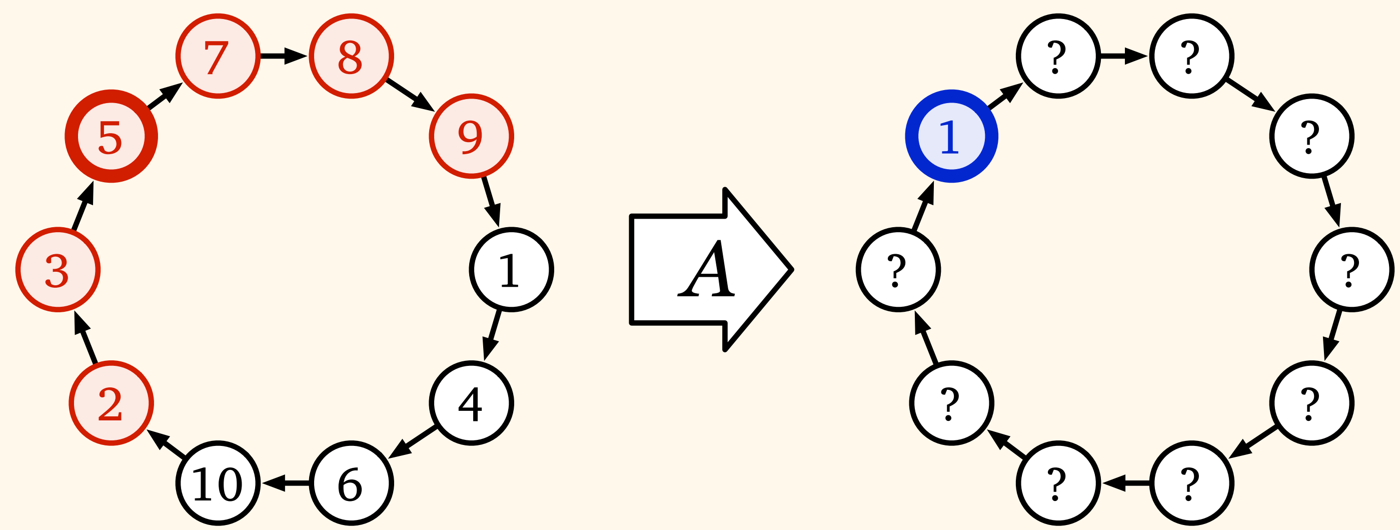


2, 3, 5, 7, 8 | 2, 3, 5, 8, 9 | 2, 3, 7, 8, 9

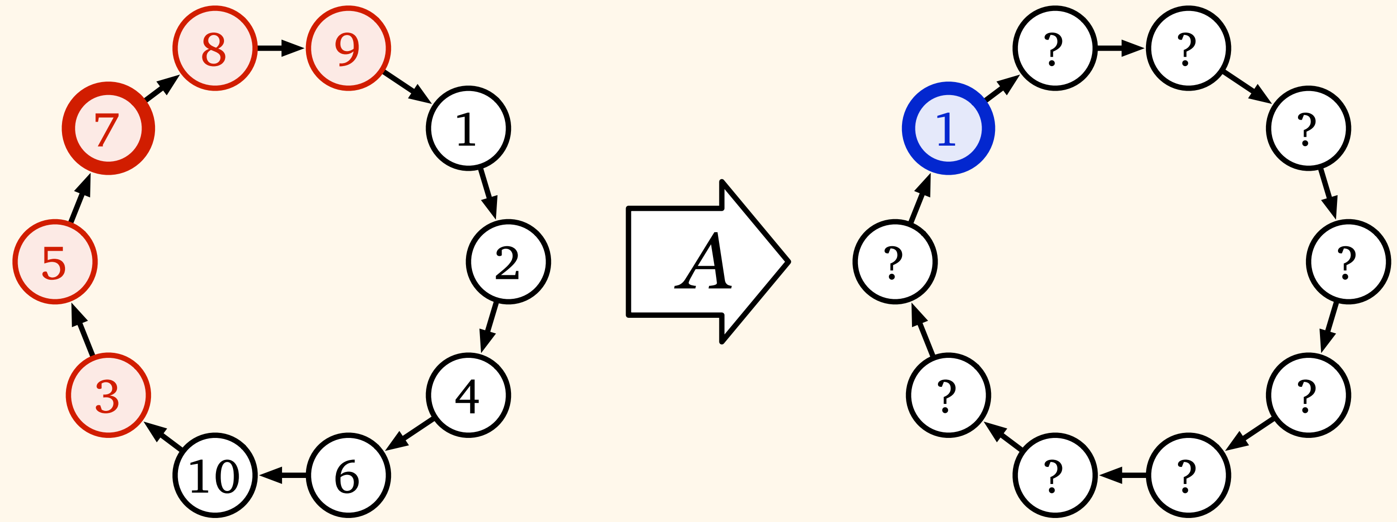
2, 3, 5, 7, 9 | 2, 5, 7, 8, 9 | 3, 5, 7, 8, 9



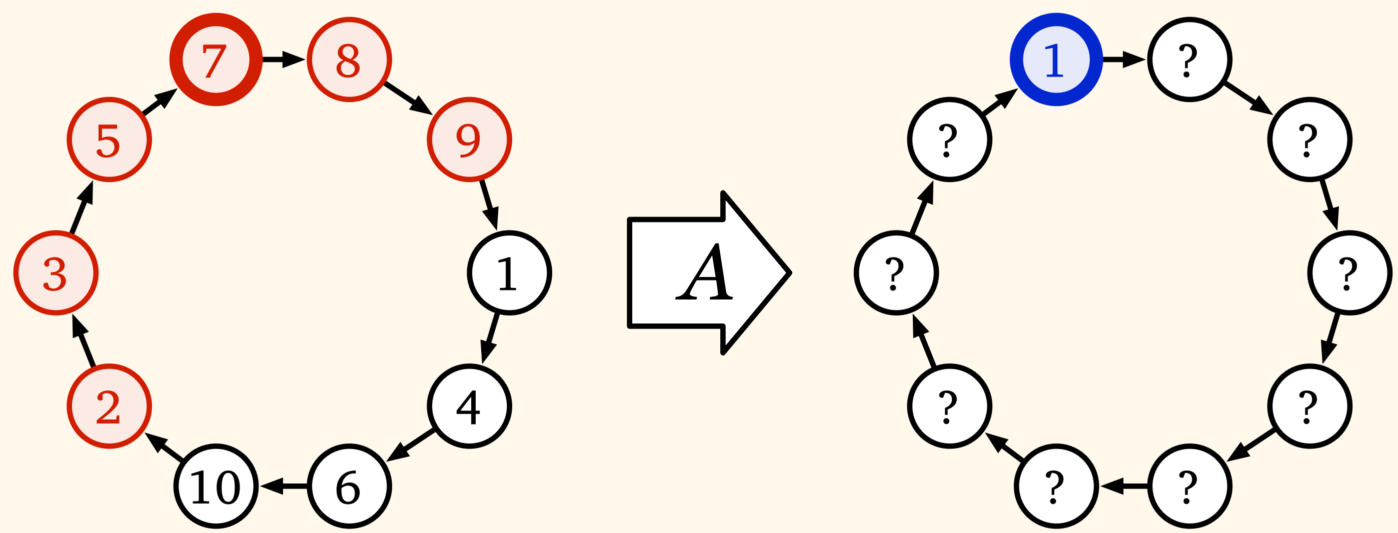
same local neighbourhood, same output



2, 3, 5, 7, 8	2, 3, 5, 8, 9	2, 3, 7, 8, 9
2, 3, 5, 7, 9	2, 5, 7, 8, 9	3, 5, 7, 8, 9



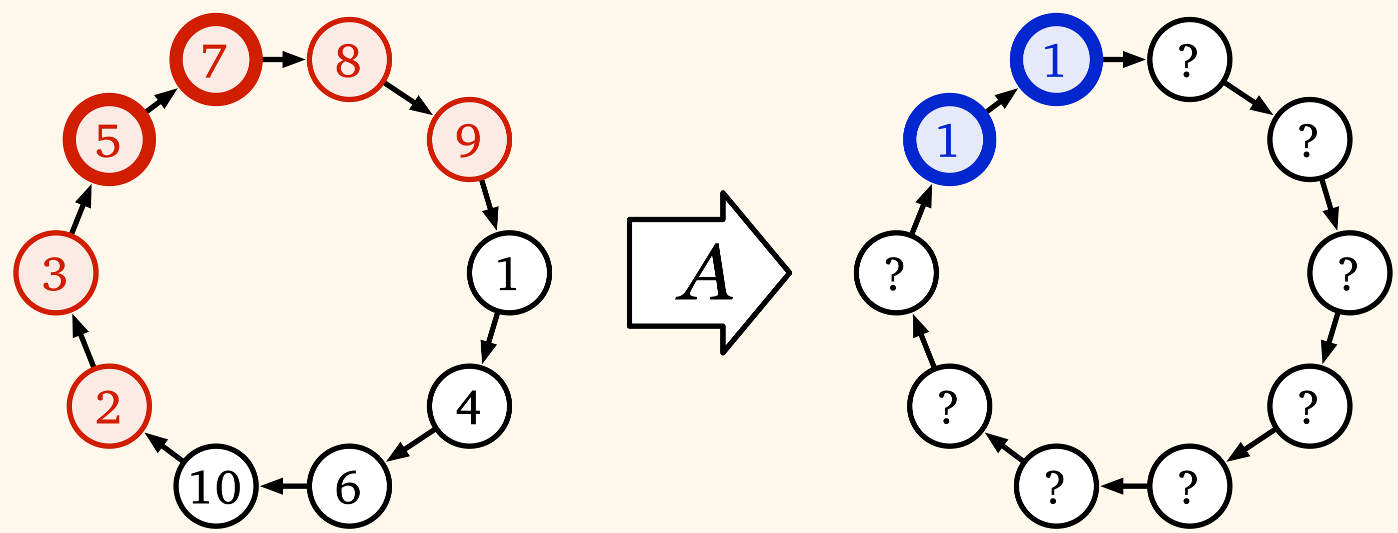
same local neighbourhood, same output



2, 3, 5, 7, 8	2, 3, 5, 8, 9	2, 3, 7, 8, 9
2, 3, 5, 7, 9	2, 5, 7, 8, 9	3, 5, 7, 8, 9

Ramsey Says No

Bad output!



2, 3, 5, 7, 8	2, 3, 5, 8, 9	2, 3, 7, 8, 9
2, 3, 5, 7, 9	2, 5, 7, 8, 9	3, 5, 7, 8, 9

Ramsey Says No

- There is no algorithm that finds a 3-colouring in time T
  - the proof holds for any constant T
  - larger $T \rightarrow$ need a (much) larger identifiers space Y

Summary

Distributed Algorithms

- Two models
- Port-numbering model
  - key question: what is computable?
- Unique identifiers
  - key question: what can be computed fast?

Algorithm Design

- *Colouring* is a powerful symmetry-breaking tool
- Port-numbering model
  - bipartite double covers → 2-colouring…
- Unique identifiers
  - identifiers → colouring → colour reduction…

Lower Bounds

- Port-numbering model
  - covering maps
  - local neighbourhoods
- Unique identifiers
  - Ramsey's theorem
  - local neighbourhoods

That's all.

- Exam: 4 May 2012
    - check the learning objectives!
- What next?
    - course feedback
    - seminar course, autumn 2012
    - Master's thesis topics available