DDA 2010, lecture 1: Introduction

- Synchronous deterministic distributed algorithms
- Two models:
  - Port-numbering model
  - Unique identifiers

Some notational conventions

- Graphs:
  - unless otherwise mentioned,
    graphs are **undirected** and **simple**
  - graphs are pairs: $G = (V, E)$,
    V set of nodes, E set of edges
  - undirected edges are unordered pairs: if there is an edge between $u \in V$ and $v \in V$, we have $\{u, v\} \in E$
  - directed edges are ordered pairs, e.g. $(u, v) \in E$
  - $\deg(v)$ = degree of $v \in V$

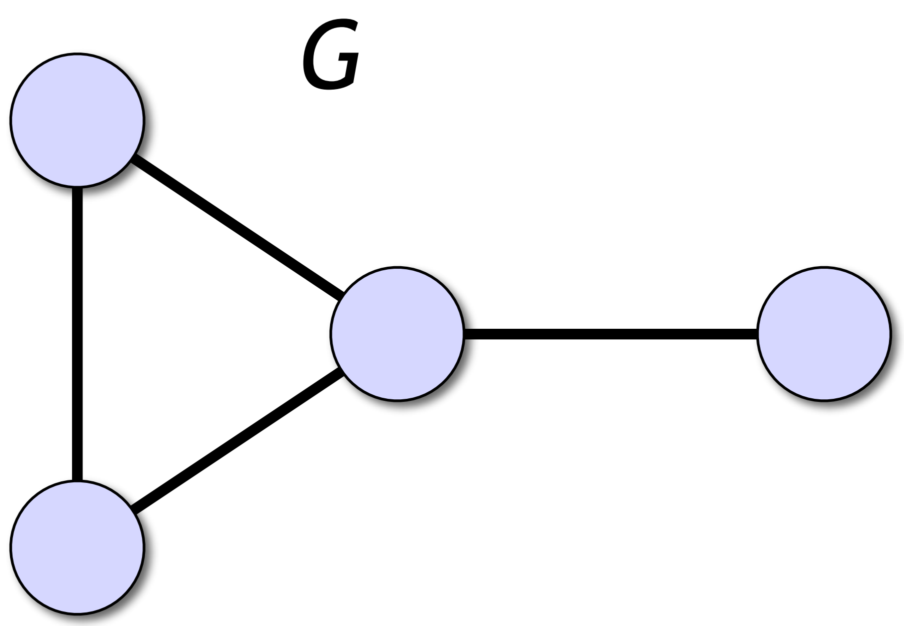
Some notational conventions

- Parameters:
  - $n = |V|$, number of nodes
  - Δ is an upper bound on degrees: $\deg(v) \le \Delta$ for all $v \in V$
- These are often used in algorithm analysis
  - e.g., "running time $O(\Delta + \log n)$"
- Sometimes we assume that Δ is a global constant
  - "bounded-degree graphs", $\Delta = O(1)$

DDA 2010, lecture 1a: Port-numbering model

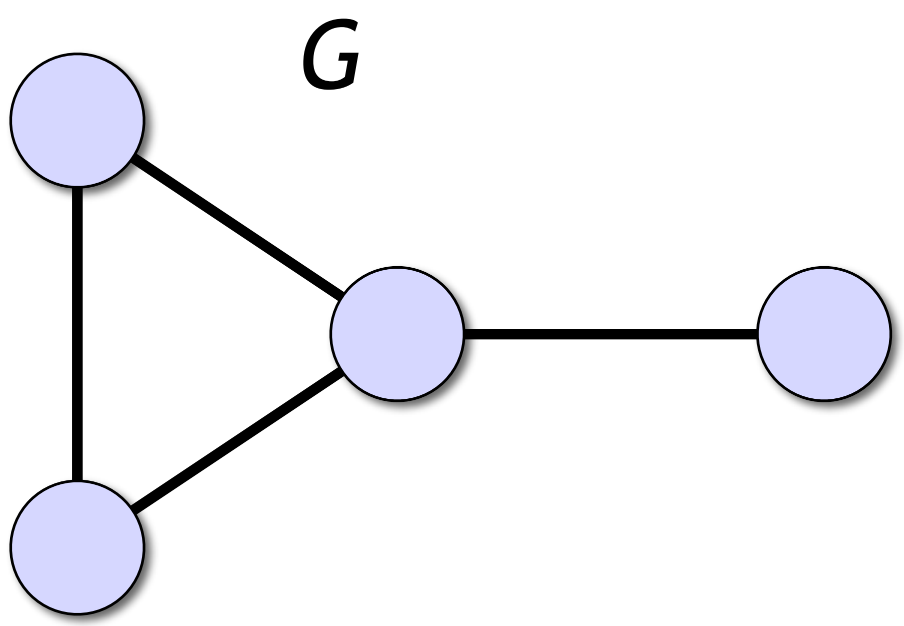
- Synchronous deterministic distributed algorithms in the port-numbering model
- Limited model, we will study extensions later

Distributed algorithms



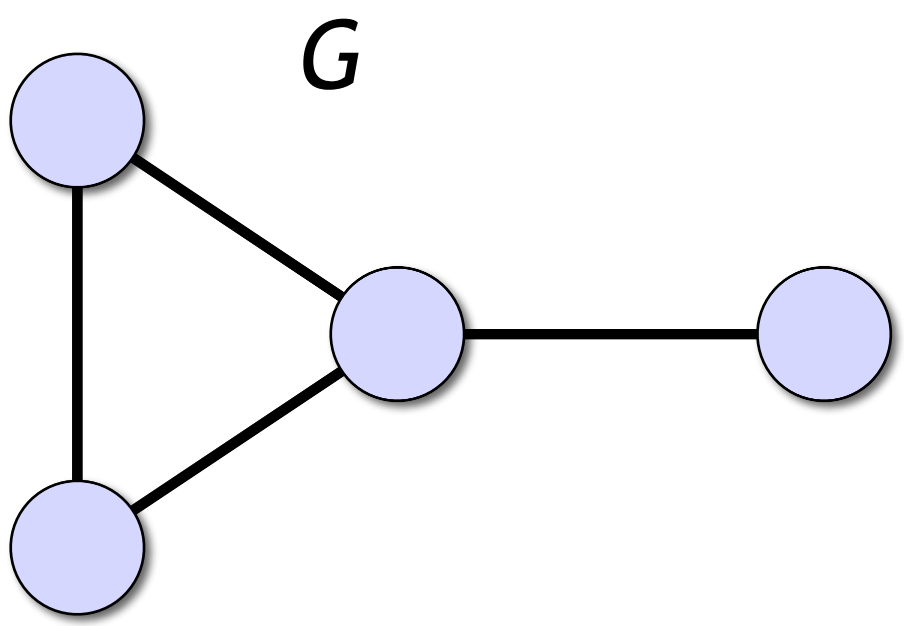
- Communication graph G
- Node = computer
  - e.g., Turing machine, finite state machine
- Edge = communication link
  - computers can exchange messages

Distributed algorithms



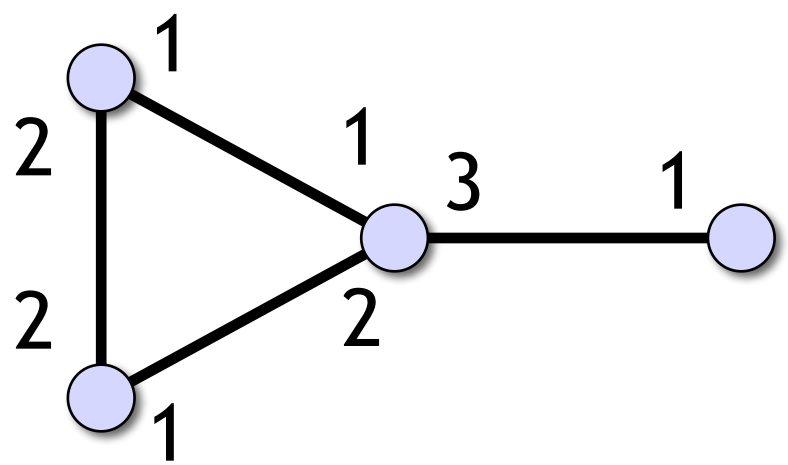
- All nodes are identical, run the **same algorithm**
- ***We*** can choose the algorithm
- An ***adversary*** chooses the structure of G
- Our algorithm must produce a correct output in any graph G

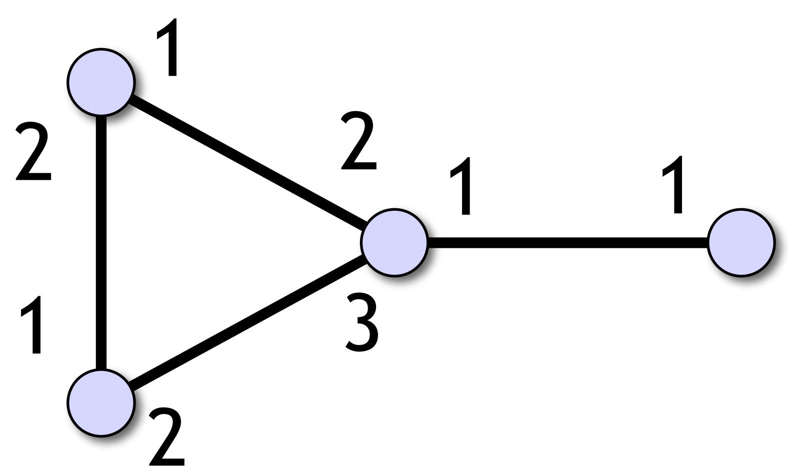
Distributed algorithms



- Usually, computational problems are related to the structure of the communication graph *G*
  - example: find a maximal independent set for *G*
  - the same graph is both the input and the system that tries to solve the problem...

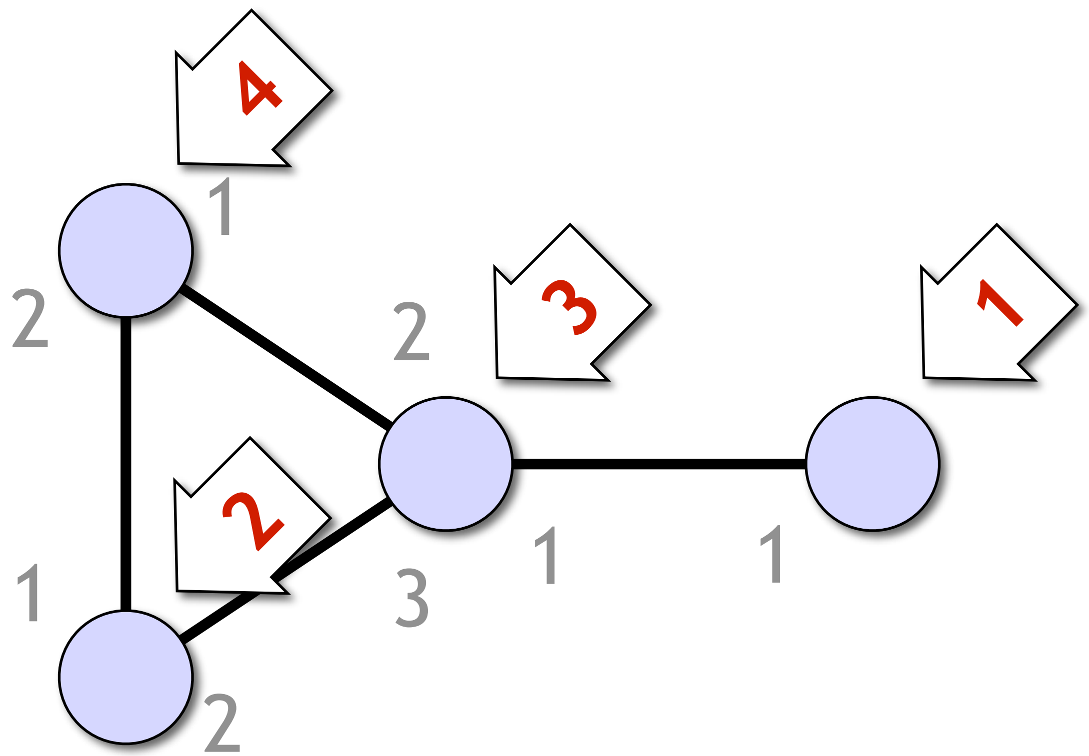
Port-numbering model





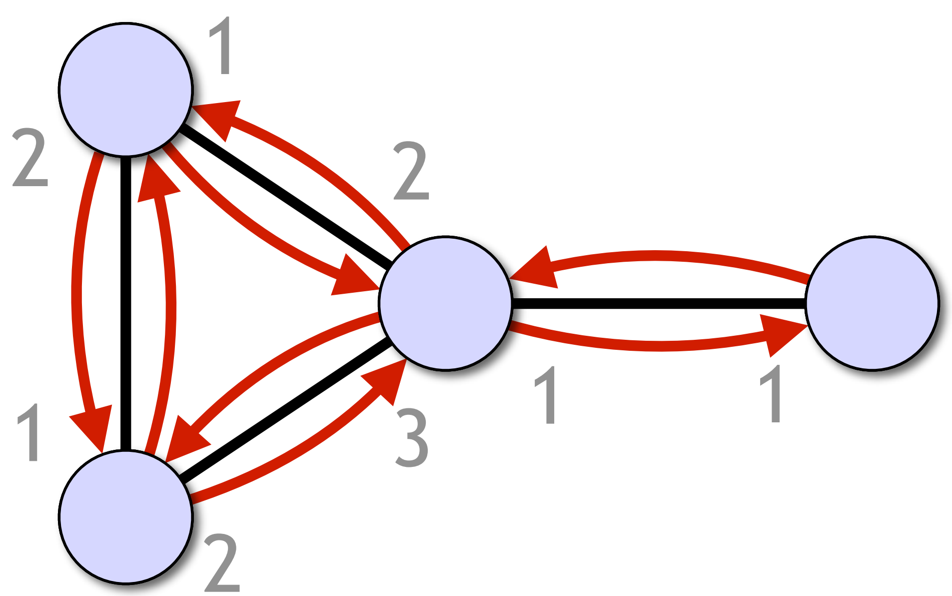
- A node of degree *d* can refer to its neighbours by integers 1, 2, ..., *d*
- Port-numbering chosen by adversary

Synchronous distributed algorithms



1. Each node reads its own **local input**
   - Depends on the problem, for example:
     - node weight
     - weights of incident edges
   - May be empty

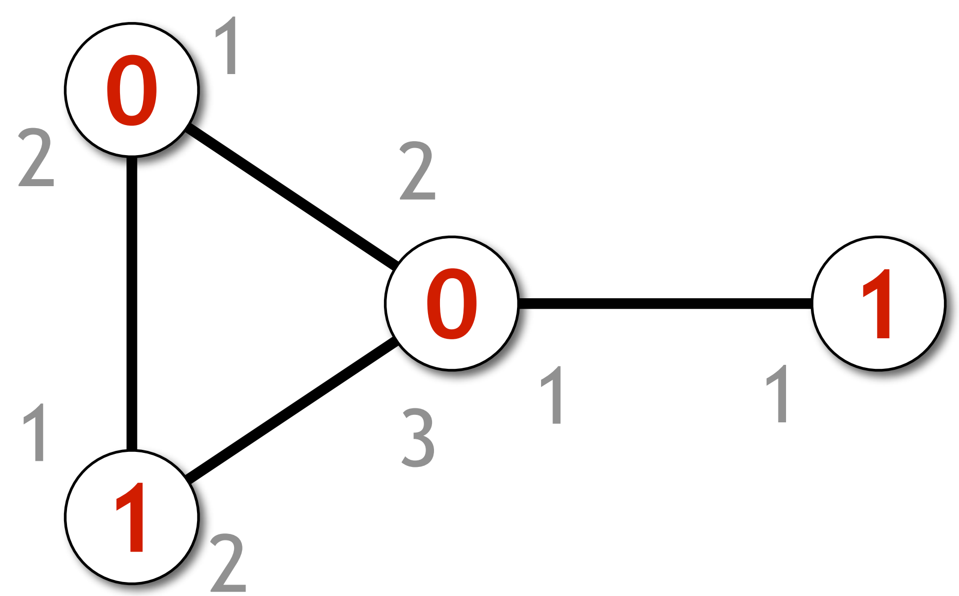
Synchronous distributed algorithms



1. Each node reads its own **local input**
2. Repeat synchronous **communication rounds**

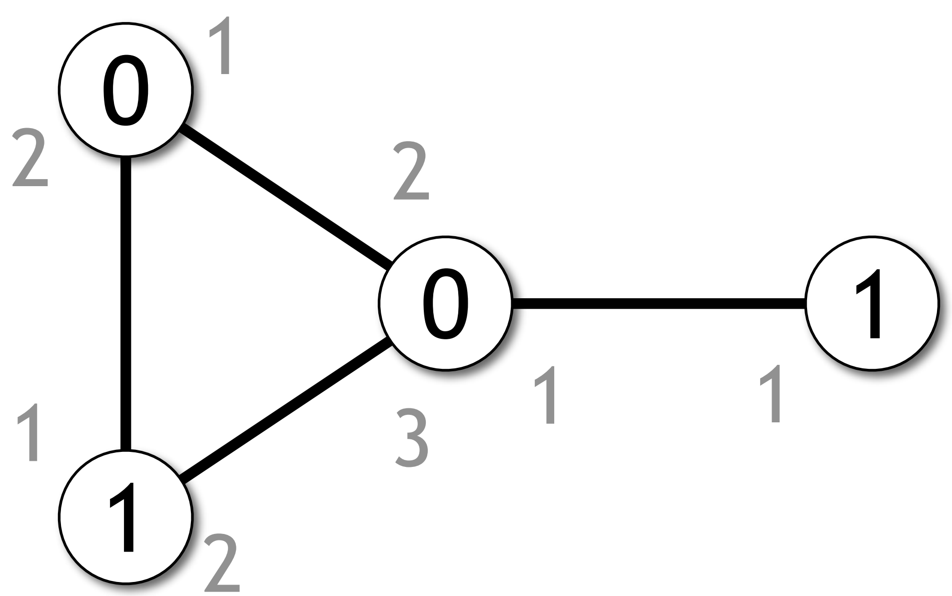
   …

Synchronous distributed algorithms



1. Each node reads its own **local input**
2. Repeat synchronous **communication rounds** until all nodes have announced their **local outputs**
   - Solution of the problem

Synchronous distributed algorithms

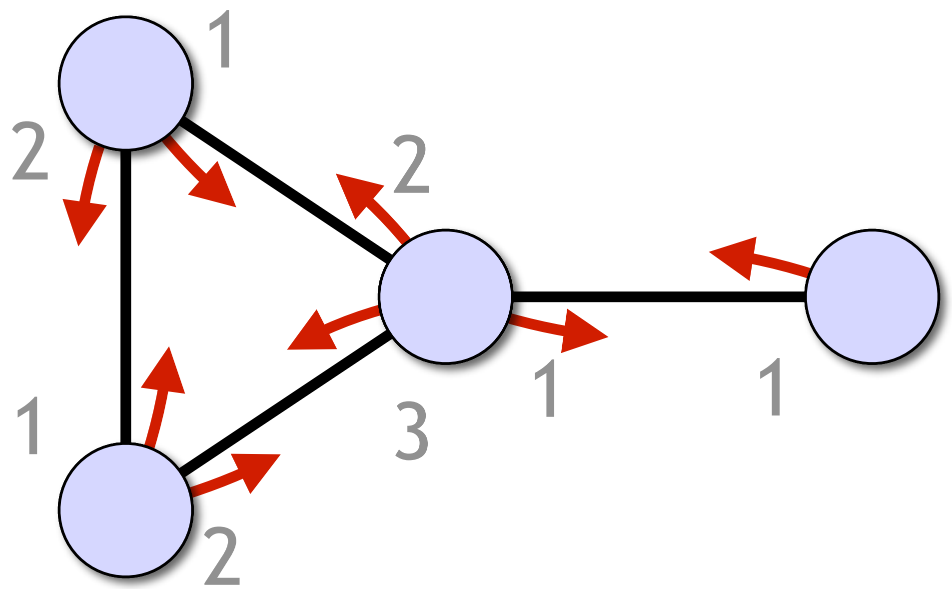


1. Each node reads its own **local input**
2. Repeat synchronous **communication rounds** until all nodes have announced their **local outputs**

Example: Find a maximal independent set *I*

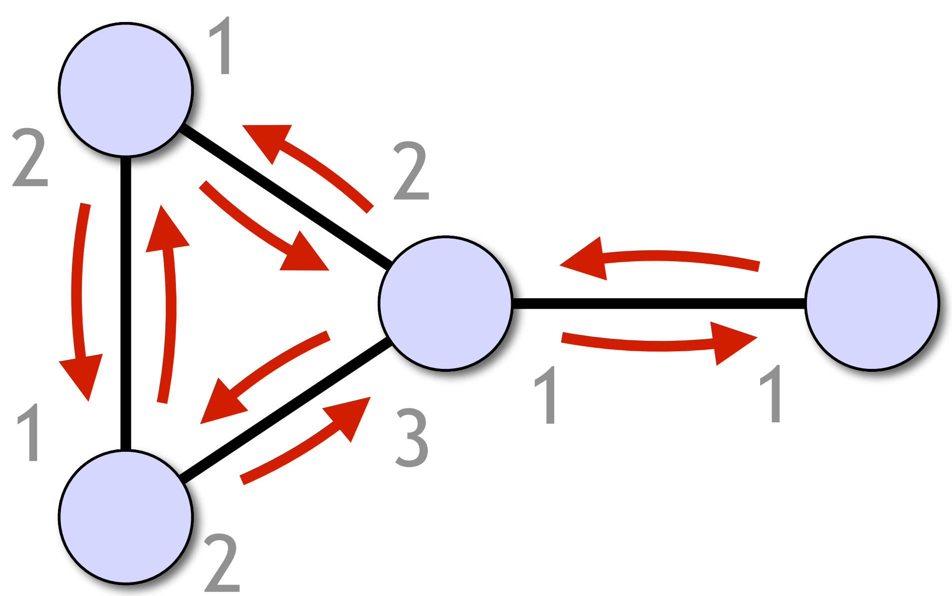
Local output of a node *v* indicates whether $v \in I$

Synchronous distributed algorithms



- Communication round: each node
  1. sends a message to each port

Synchronous distributed algorithms

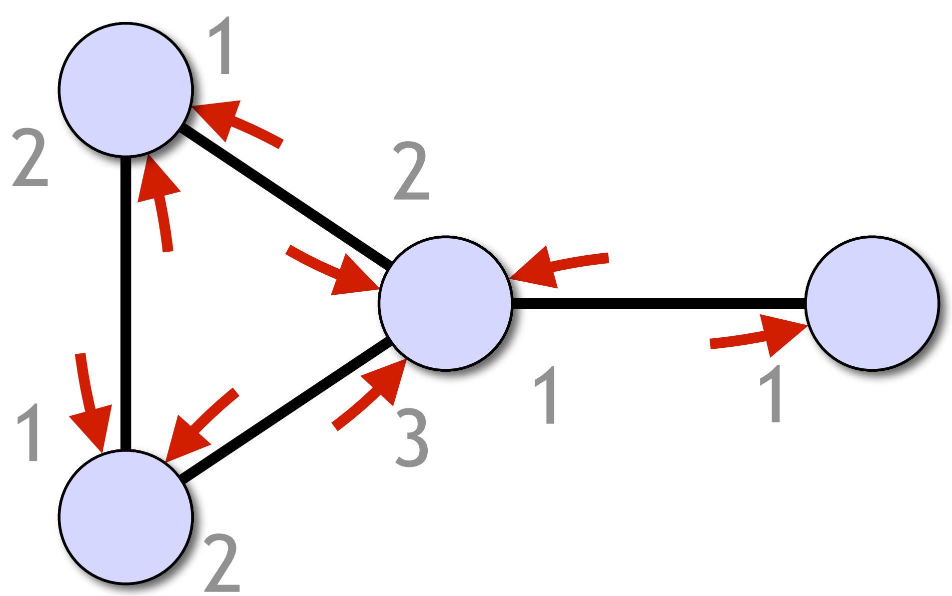


- Communication round: each node

  1. sends a message to each port

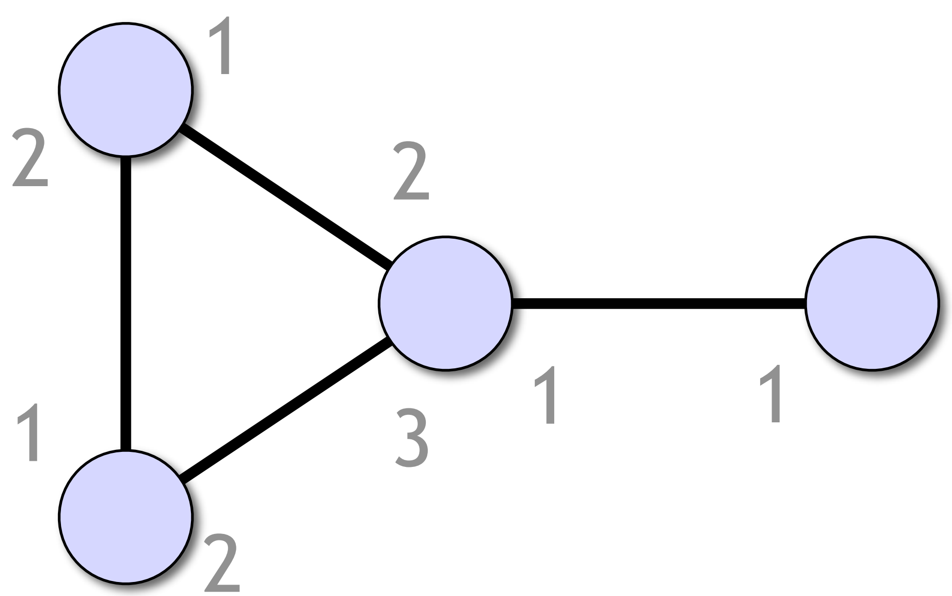
  (message propagation...)

Synchronous distributed algorithms



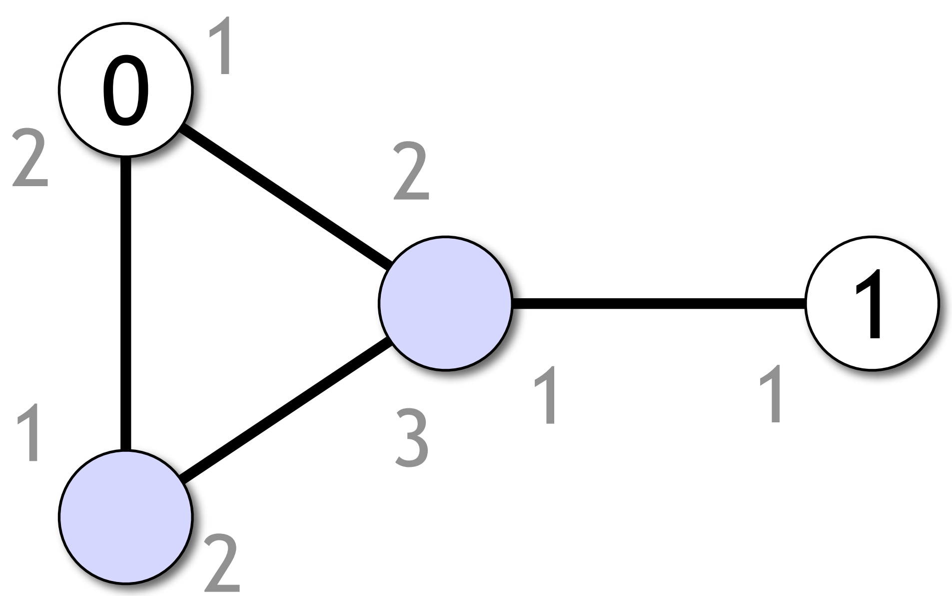
- Communication round: each node
  1. sends a message to each port
  2. receives a message from each port

Synchronous distributed algorithms



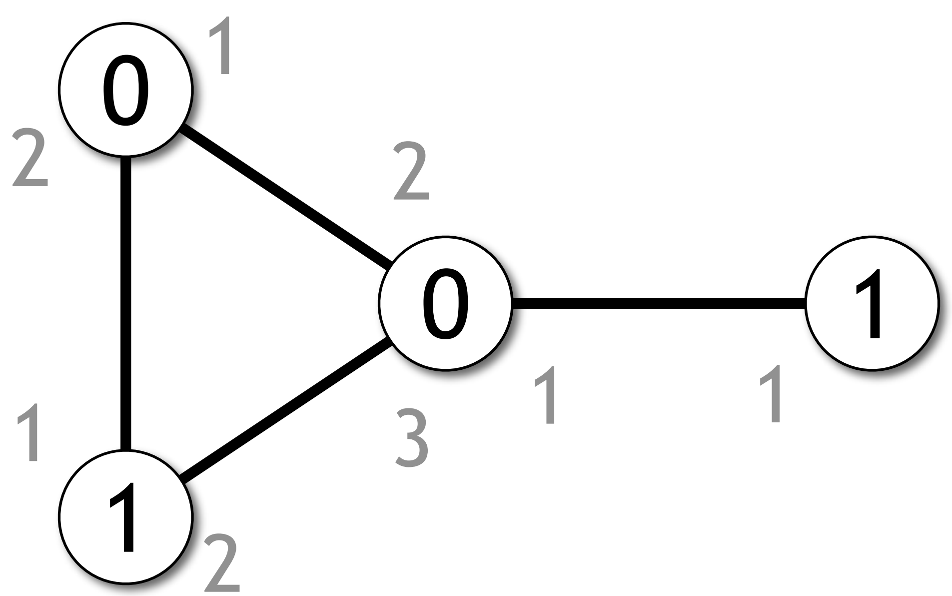
- Communication round: each node
  1. sends a message to each port
  2. receives a message from each port
  3. updates its own state

Synchronous distributed algorithms



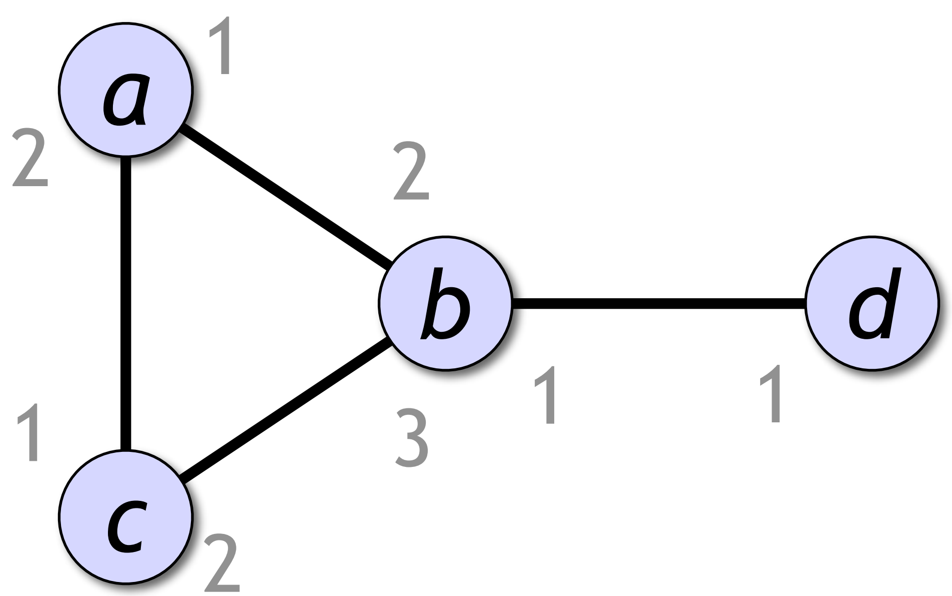
- Communication round: each node
  1. sends a message to each port
  2. receives a message from each port
  3. updates its own state
  4. possibly stops and announces its output

Synchronous distributed algorithms



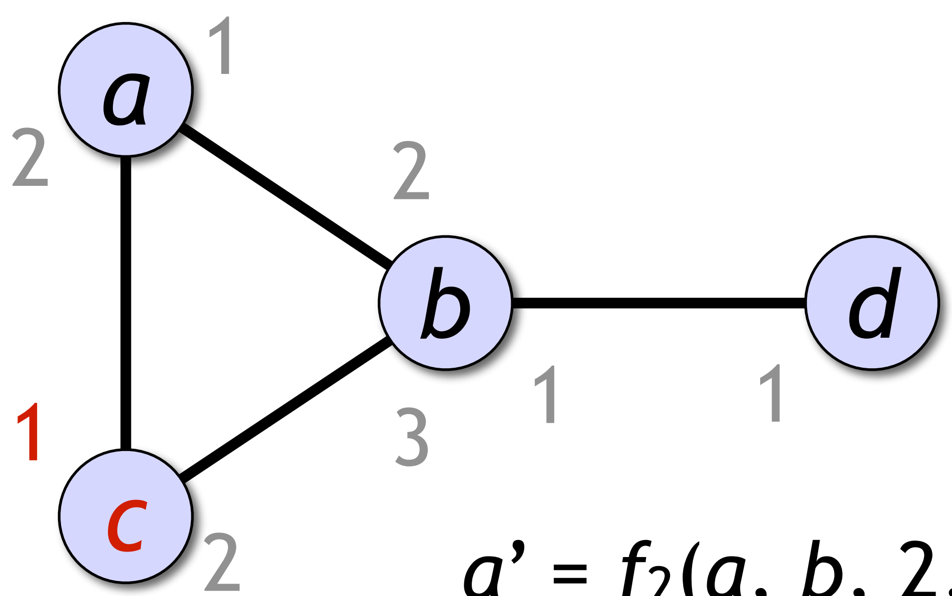
- Communication rounds are repeated until all nodes have stopped and announced their outputs
- Running time = **number of rounds**
- Worst-case analysis

Synchronous distributed algorithms: networks of state machines



- Equivalently:
  - Node = **state machine** (not necessarily finite)
  - All nodes update their states simultaneously

Synchronous distributed algorithms: networks of state machines



- Equivalently:
  - Node = **state machine** (not necessarily finite)
  - All nodes update their states simultaneously

$a' = f_2(a, b, 2, c, 1)$

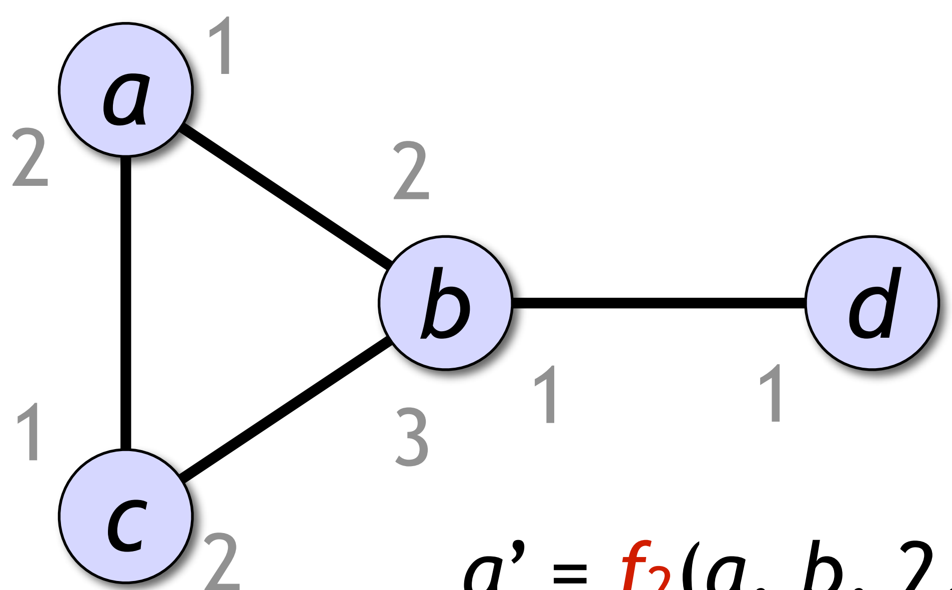
$b' = f_3(b, d, 1, a, 1, c, 2)$

$c' = f_2(c, a, 2, b, 3)$

$d' = f_1(d, b, 1)$

Current state + port number: we can reconstruct the outgoing message

Synchronous distributed algorithms: networks of state machines



- Equivalently:
  - Node = **state machine** (not necessarily finite)
  - All nodes update their states simultaneously

$a' = f_2(a, b, 2, c, 1)$

$b' = f_3(b, d, 1, a, 1, c, 2)$

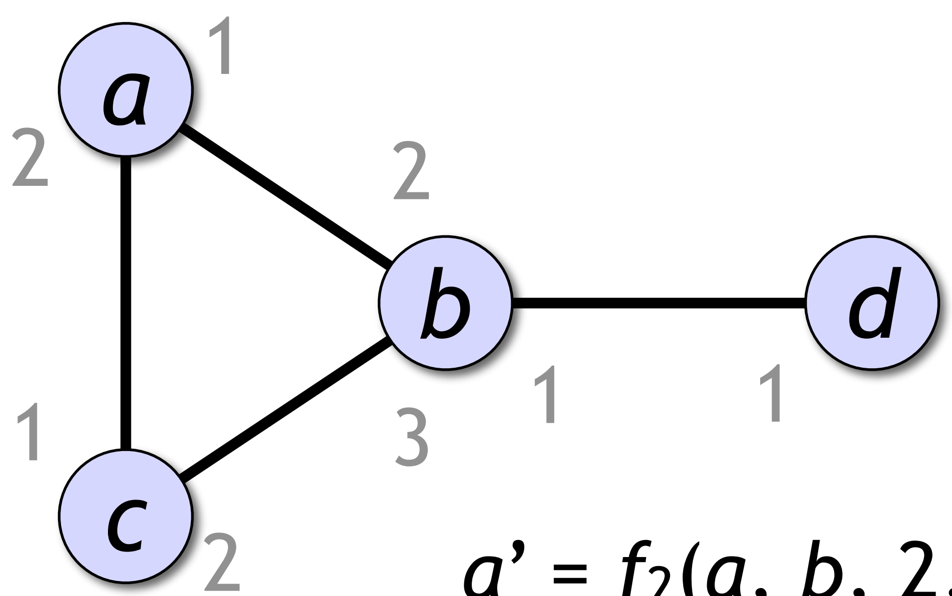
$c' = f_2(c, a, 2, b, 3)$

$d' = f_1(d, b, 1)$

Same function

f_2 = algorithm for degree-2 nodes

Synchronous distributed algorithms: networks of state machines



$a' = f_2(a, b, 2, \ldots)$

$b' = f_3(b, d, 1, \ldots)$

$c' = f_2(c, a, 2, \ldots)$

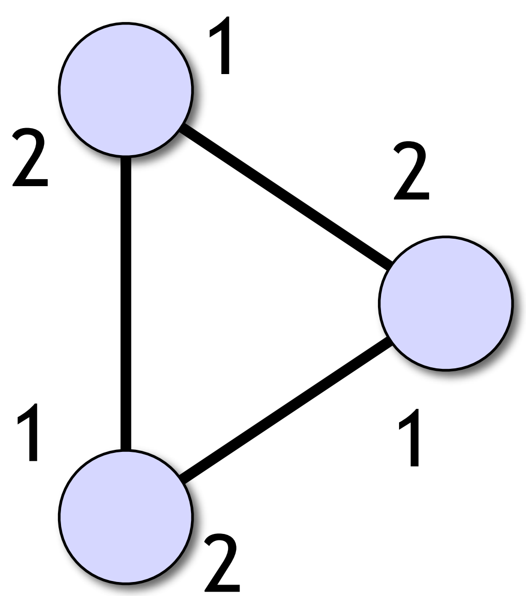
$d' = f_1(d, b, 1)$

- Equivalently:
  - Node = **state machine** (not necessarily finite)
  - All nodes update their states simultaneously
  - Initial state = local input (incl. degree of the node)
  - Final state = local output

DDA 2010, lecture 1b: Computability in port-numbering model

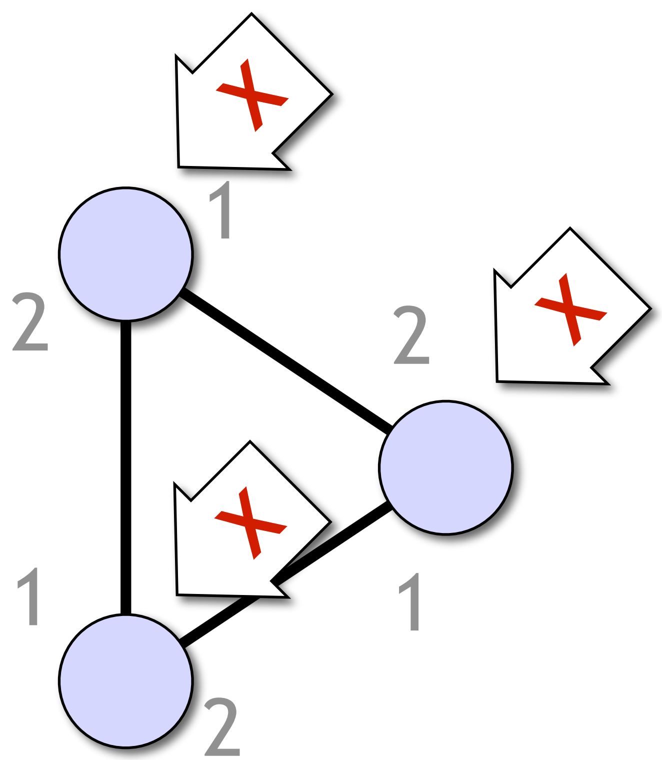
- Impossibility of symmetry breaking
- Covering maps and covering graphs: tools for proving more impossibility results

Symmetry can't be broken



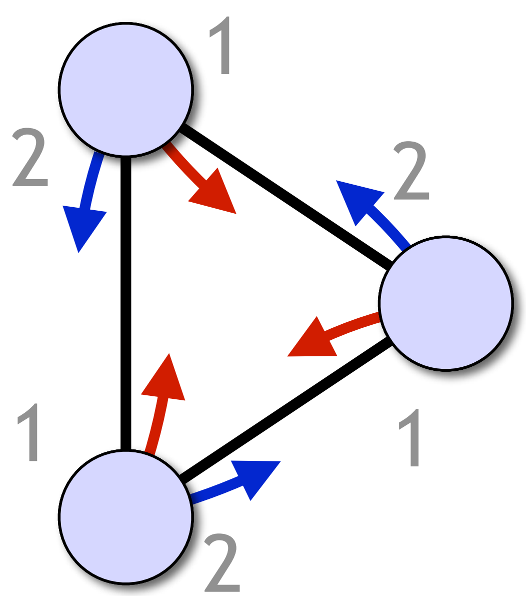
- Input may be symmetric
  - symmetric graph
  - symmetric port numbering
  - identical local inputs

Symmetry can’t be broken



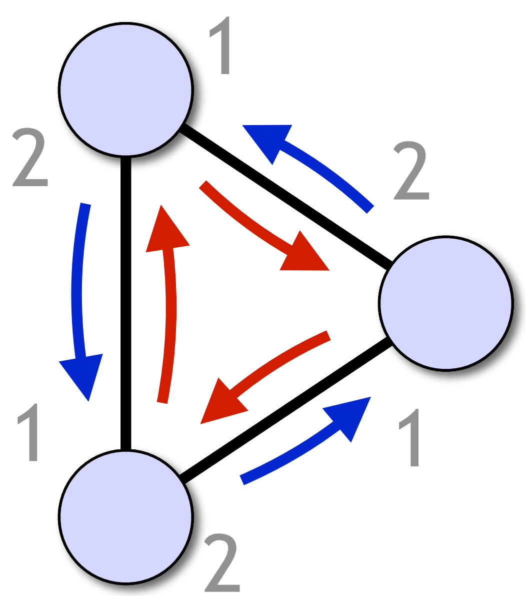
- Same input
- Same algorithm
- Same initial state

Symmetry can't be broken

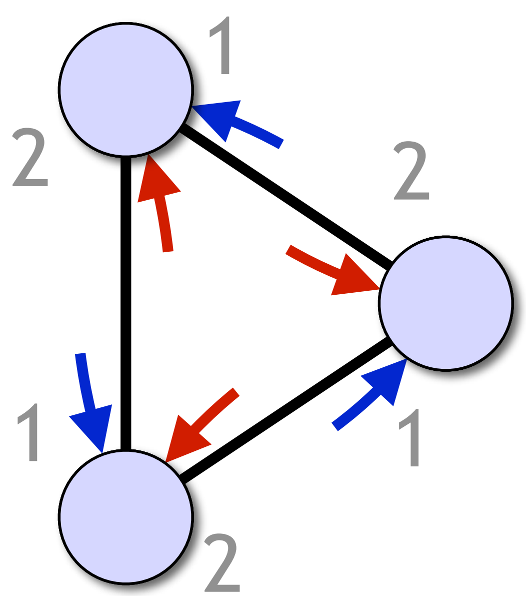


- Same current state
- Messages sent to port 1 are identical to each other
- Messages sent to port 2 are identical to each other

Symmetry can't be broken

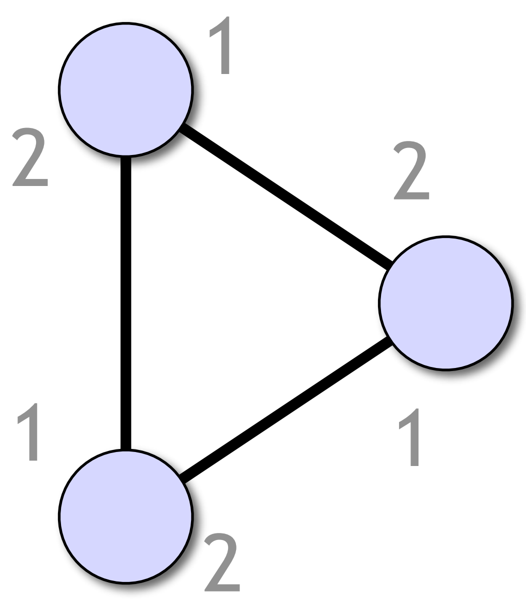


Symmetry can’t be broken



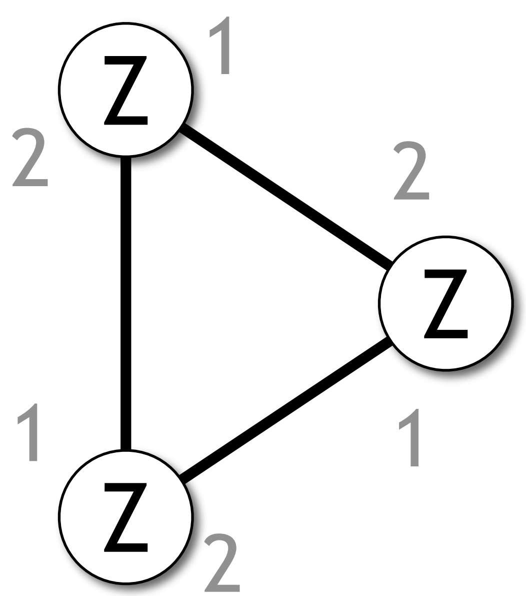
- Messages received from port 1 are identical to each other
- Messages received from port 2 are identical to each other

Symmetry can't be broken



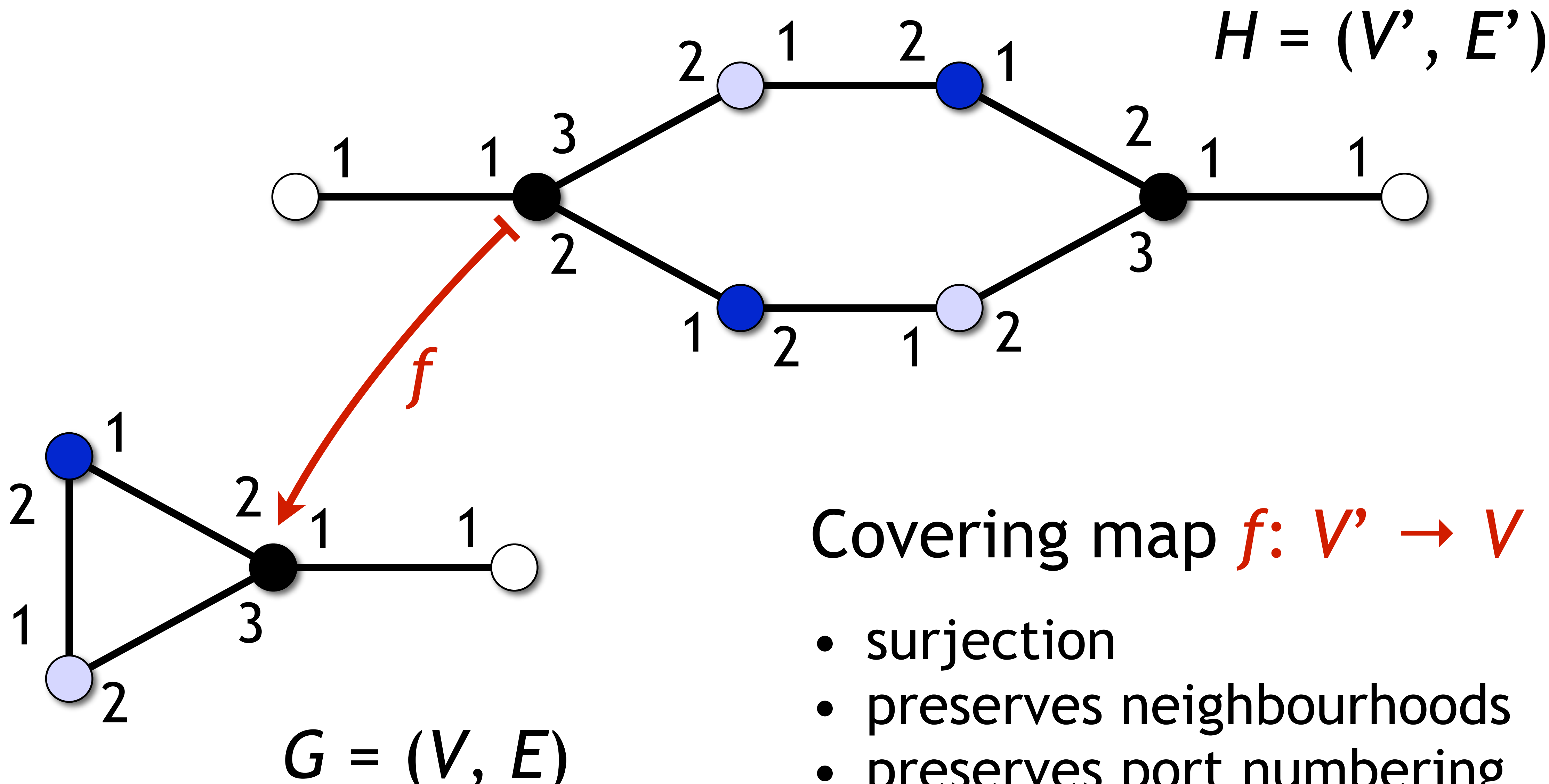
- Same old state
- Same set of received messages
- Same deterministic algorithm
- **Same new state**

Symmetry can’t be broken



- Same new state
- Either none of the nodes stops — or all of them stop and produce **identical outputs**
- Symmetry can’t be broken!
    - let’s formalise this...

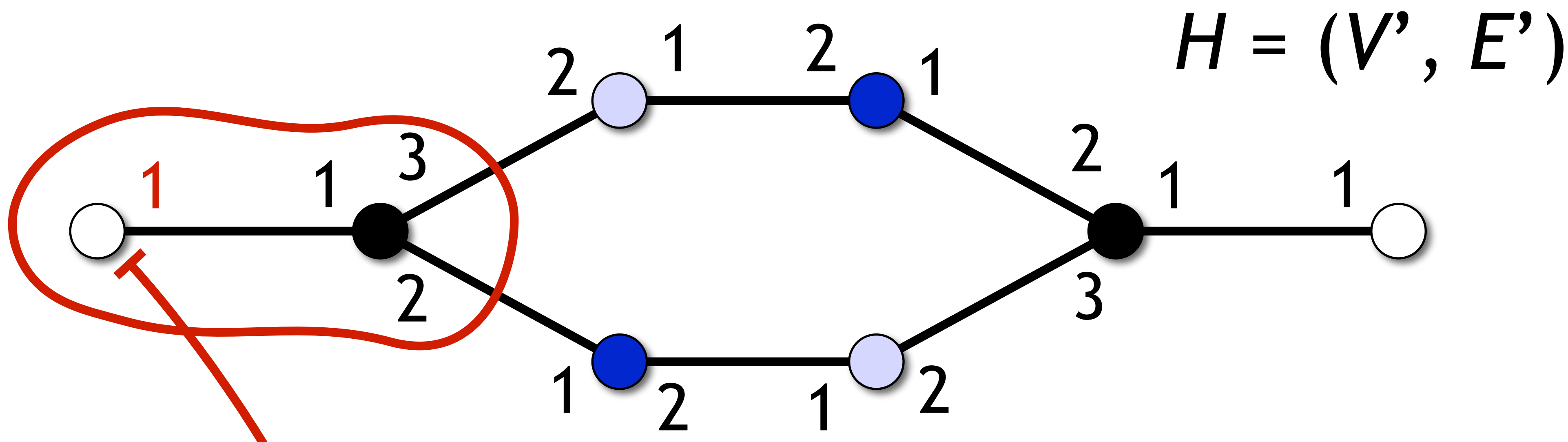
Covering maps

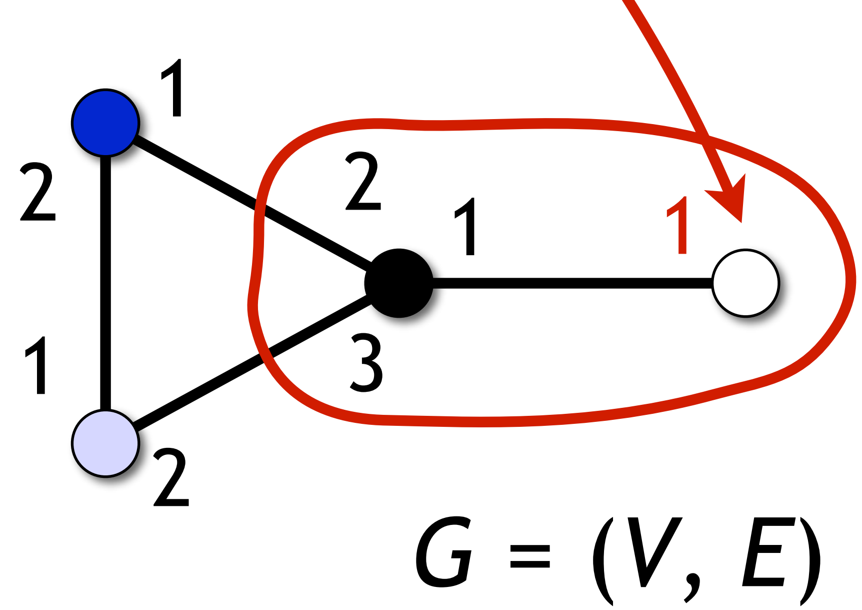


Covering map f: $V' \to V$

- surjection
- preserves neighbourhoods
- preserves port numbering

Covering maps

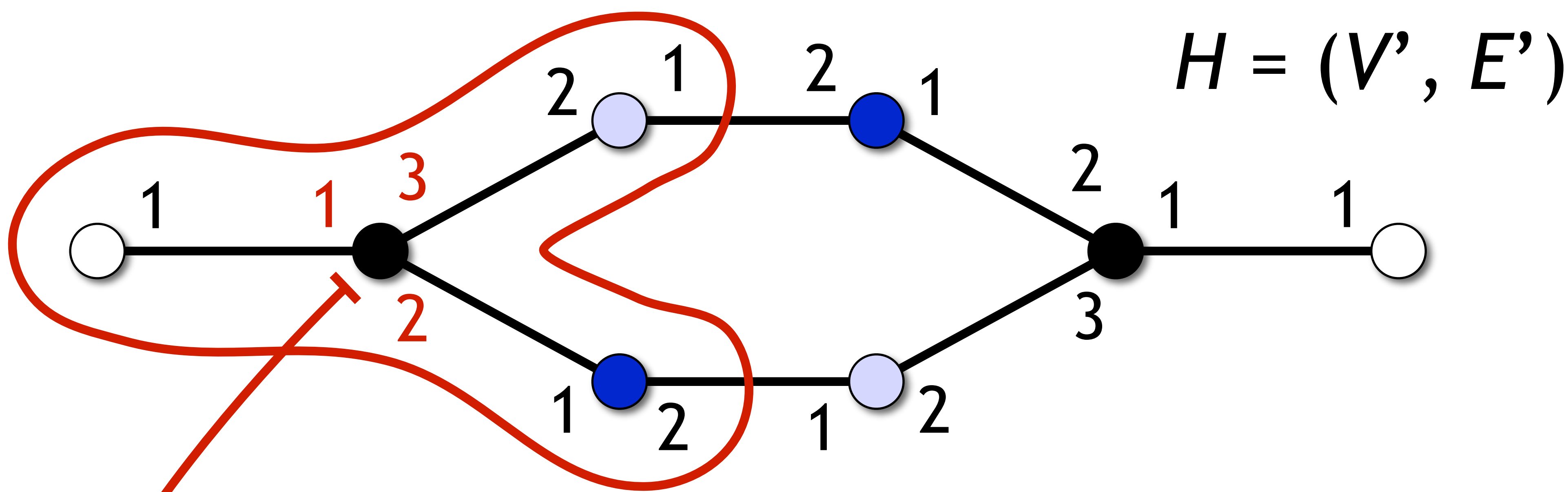


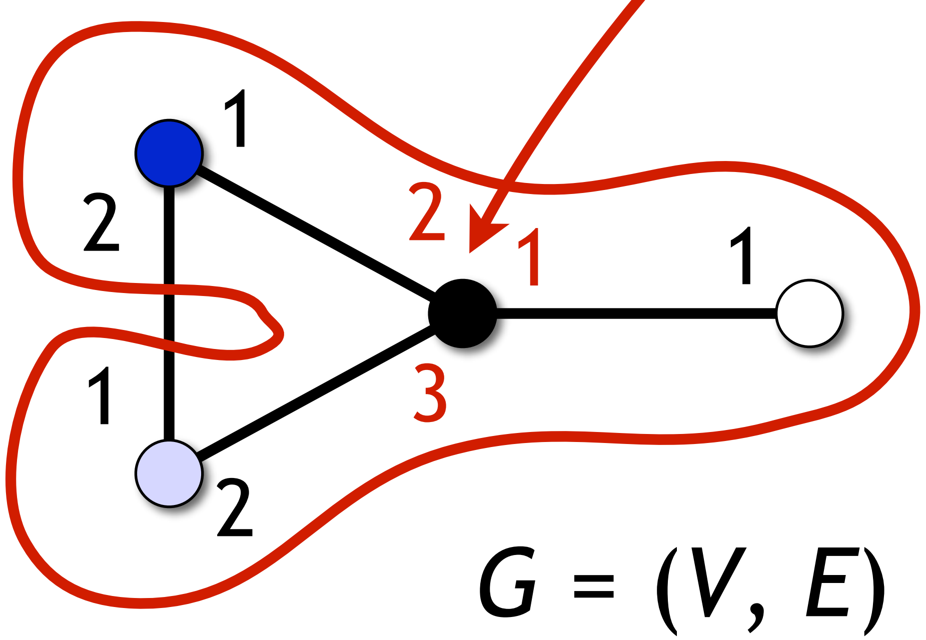


Covering map f: $V' \to V$

- surjection
- preserves neighbourhoods
- preserves port numbering

Covering maps

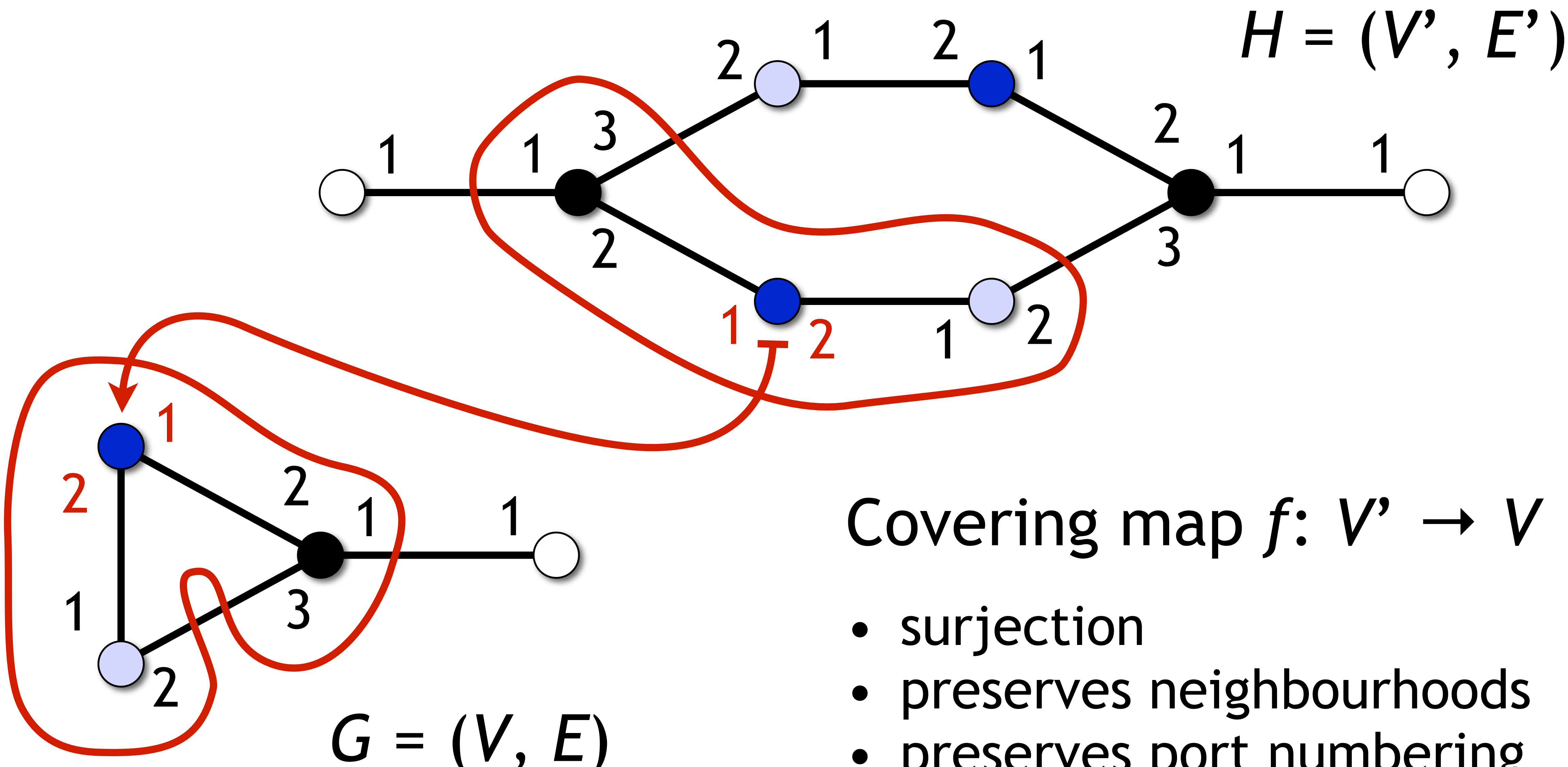




Covering map f: $V' \rightarrow V$

- surjection
- preserves neighbourhoods
- preserves port numbering

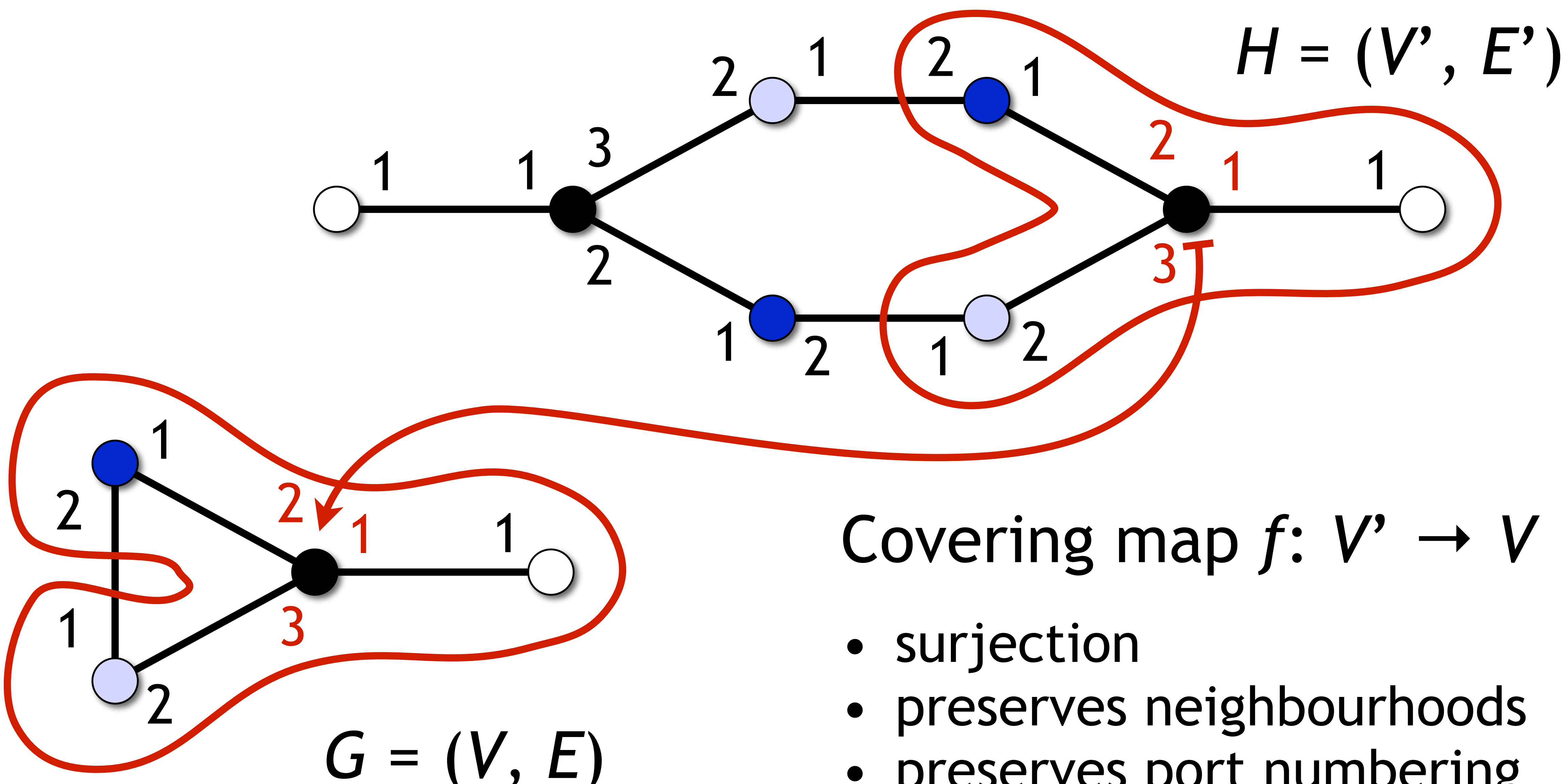
Covering maps



Covering map f: $V' \to V$

- surjection
- preserves neighbourhoods
- preserves port numbering

Covering maps

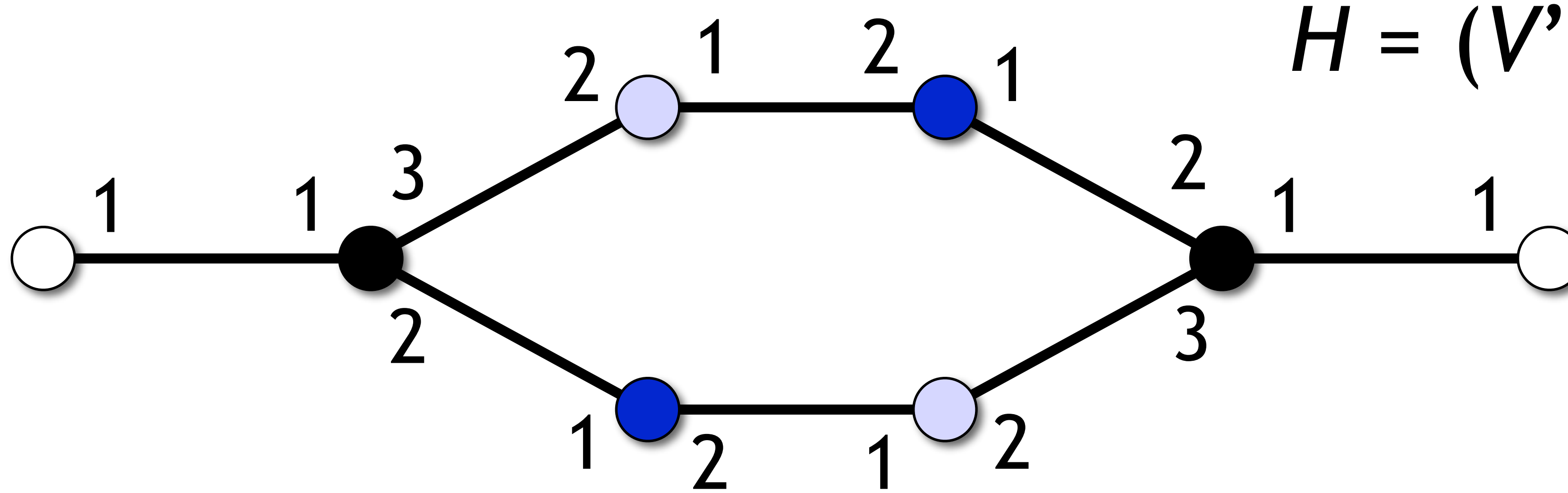


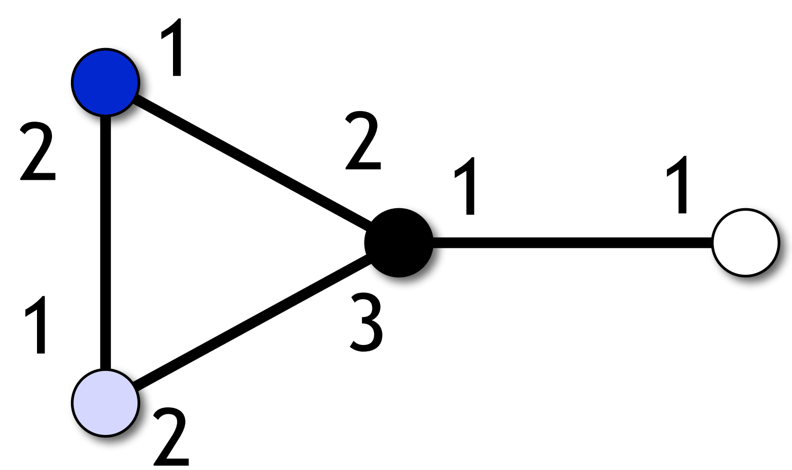
Covering map $f\colon V' \to V$

- surjection
- preserves neighbourhoods
- preserves port numbering

Covering maps and covering graphs

$H = (V', E')$



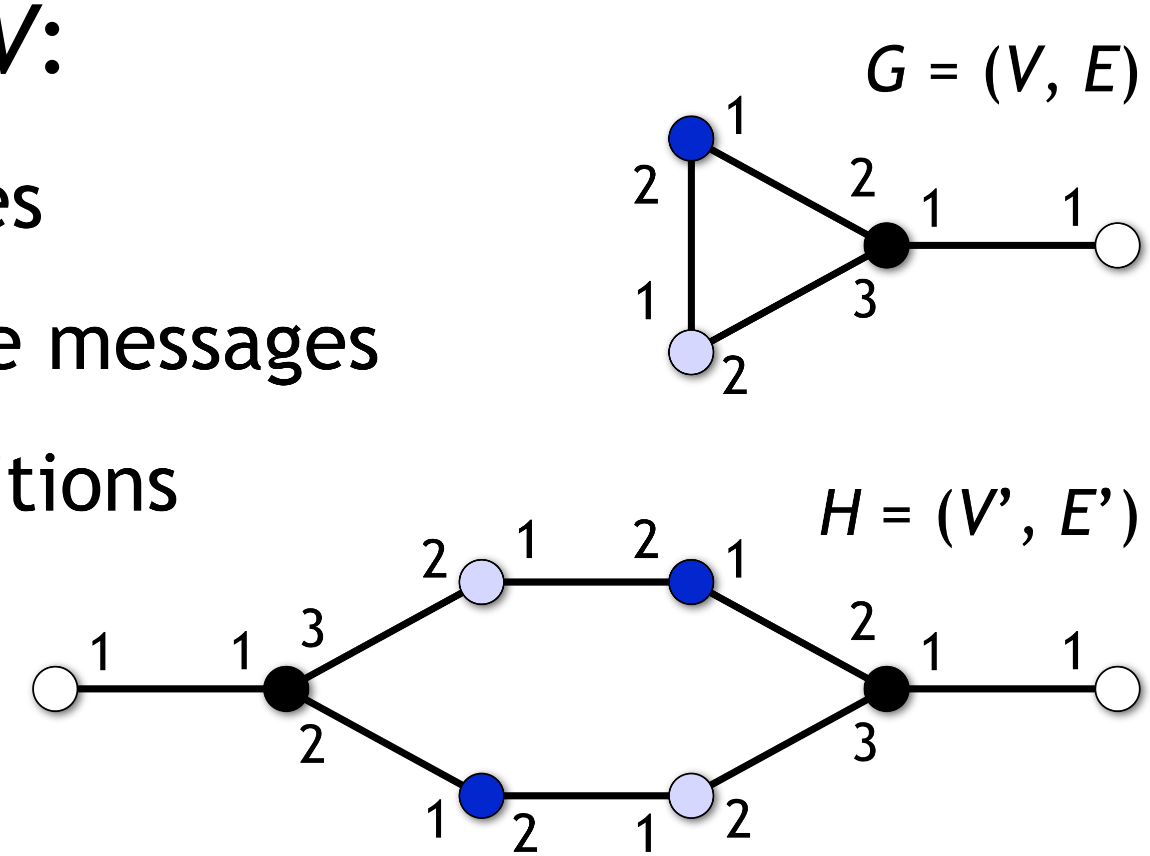


$G = (V, E)$

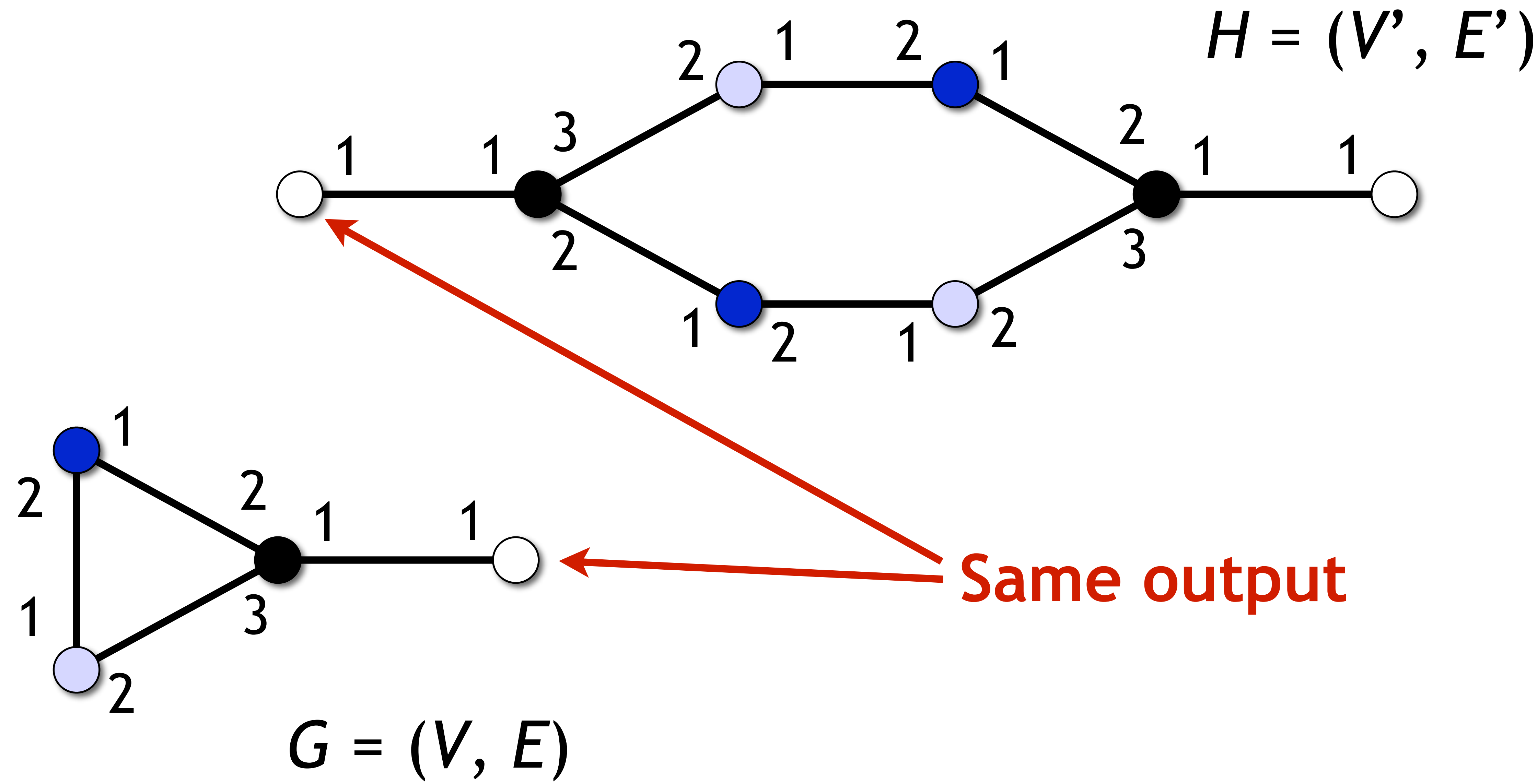
H is a **covering graph** of *G* if there is a covering map $f\colon V' \to V$

Covering maps and covering graphs

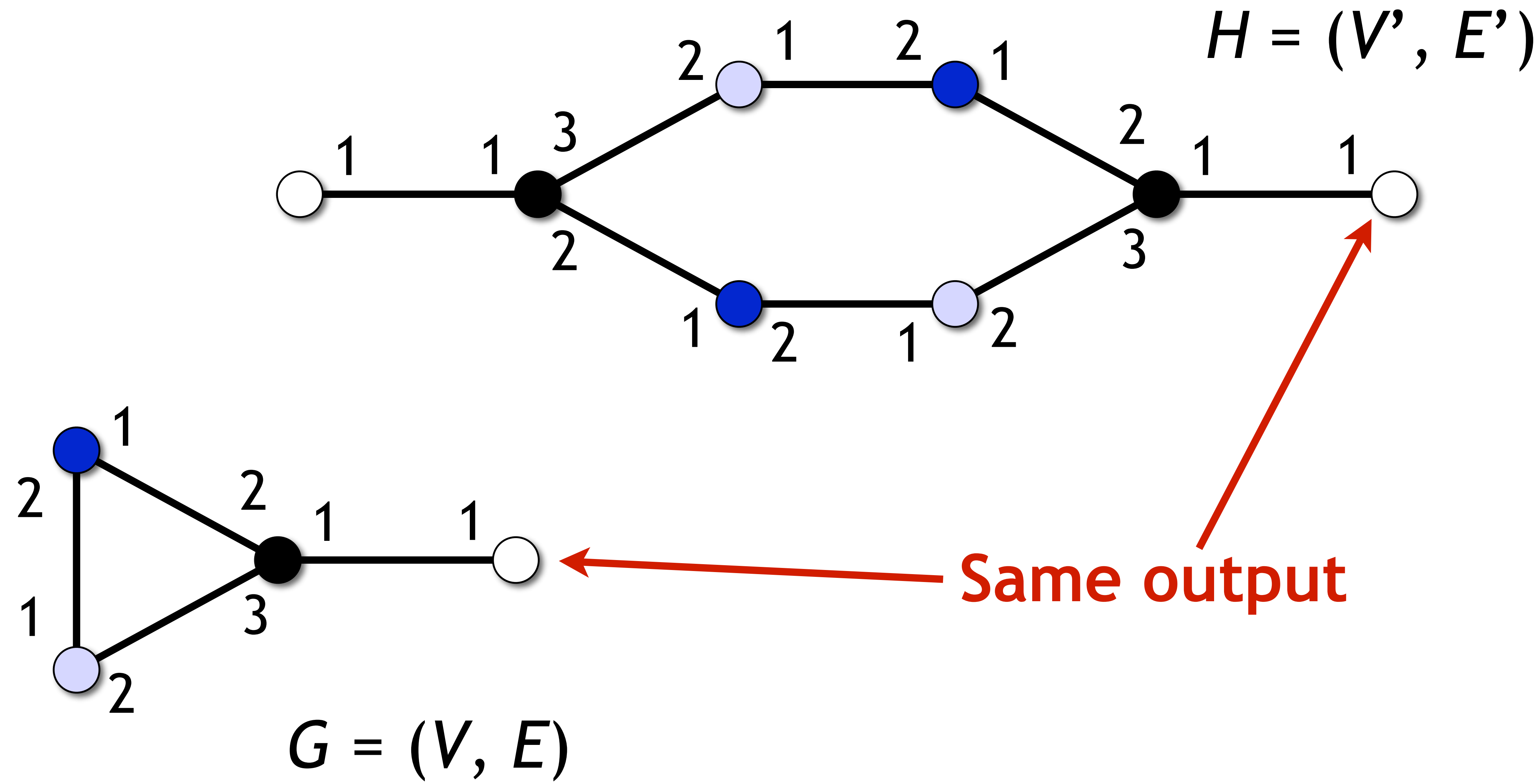
- Run the **same algorithm** in G and H
  - $v' \in V'$ and $f(v') \in V$ have the same input for all v'
- Then $v' \in V'$ and $f(v') \in V$:
  - have identical initial states
  - send and receive the same messages
  - have identical state transitions
  - produce **identical local outputs**!



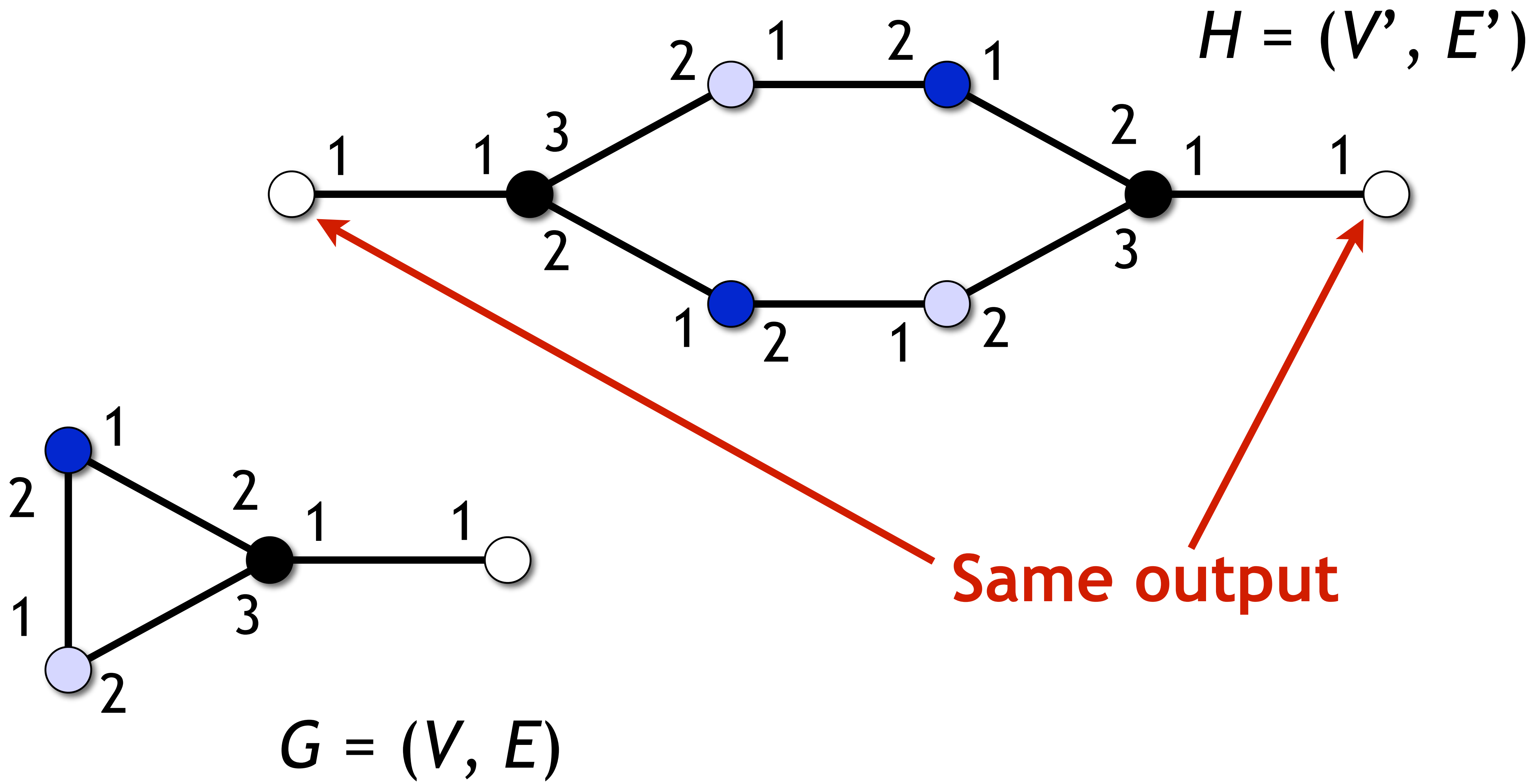
Covering maps and covering graphs



Covering maps and covering graphs



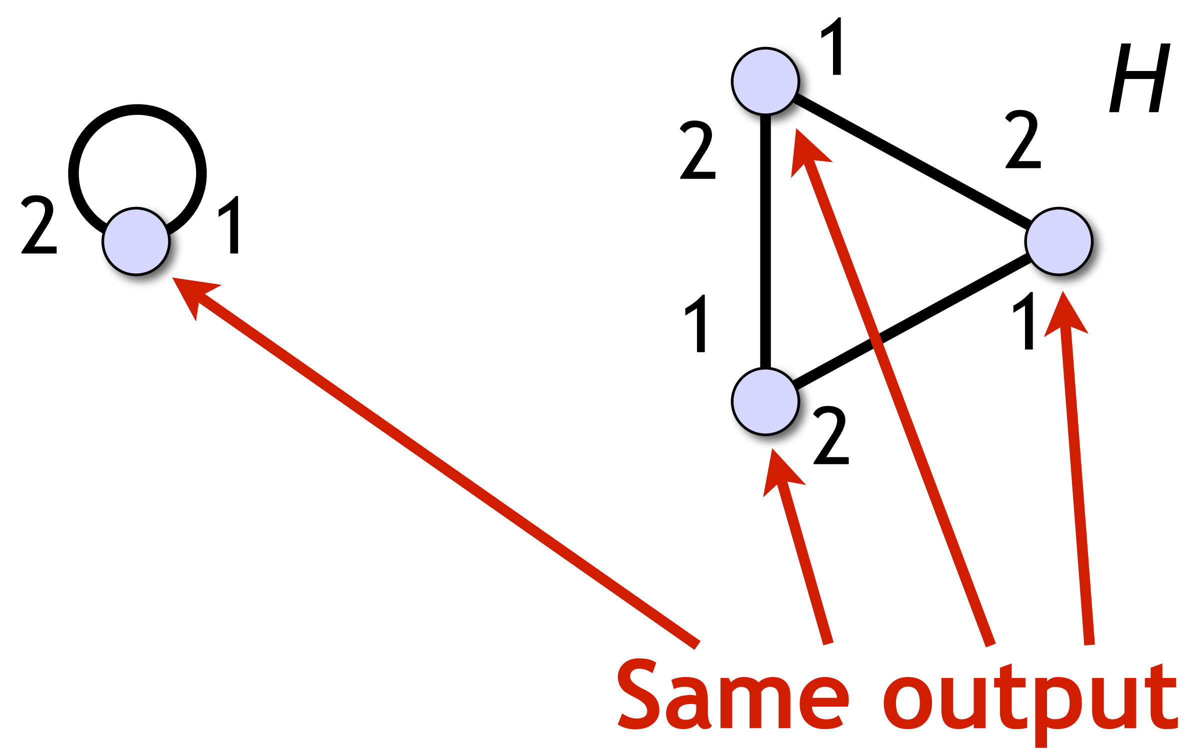
Covering maps and covering graphs



Covering maps and covering graphs

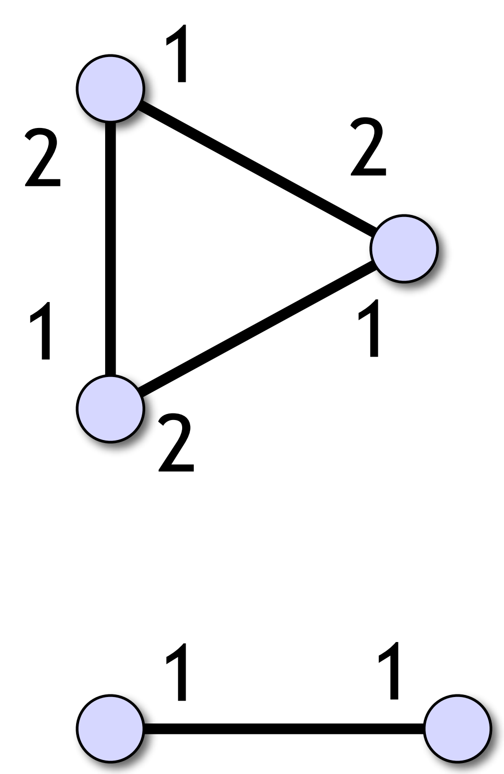
- Symmetric cycles are a simple special case of covering maps

$G = (V, E)$



$H = (V', E')$

Computability in the port-numbering model

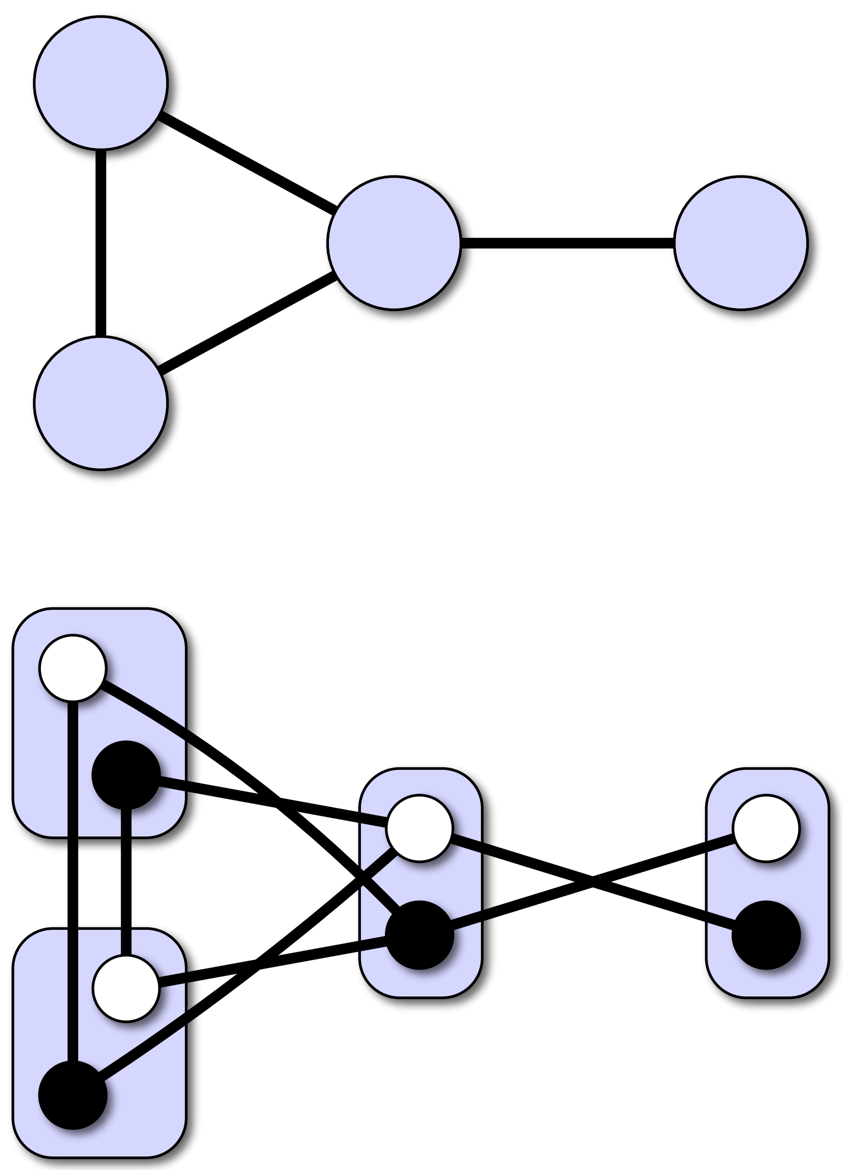


- Very limited model
  - in a cycle, we can only find a trivial solution: empty set, all nodes, ...
  - we can't even break symmetry in a 2-node network!
- **What can be solved**?

DDA 2010, lecture 1c: Algorithms in port-numbering model

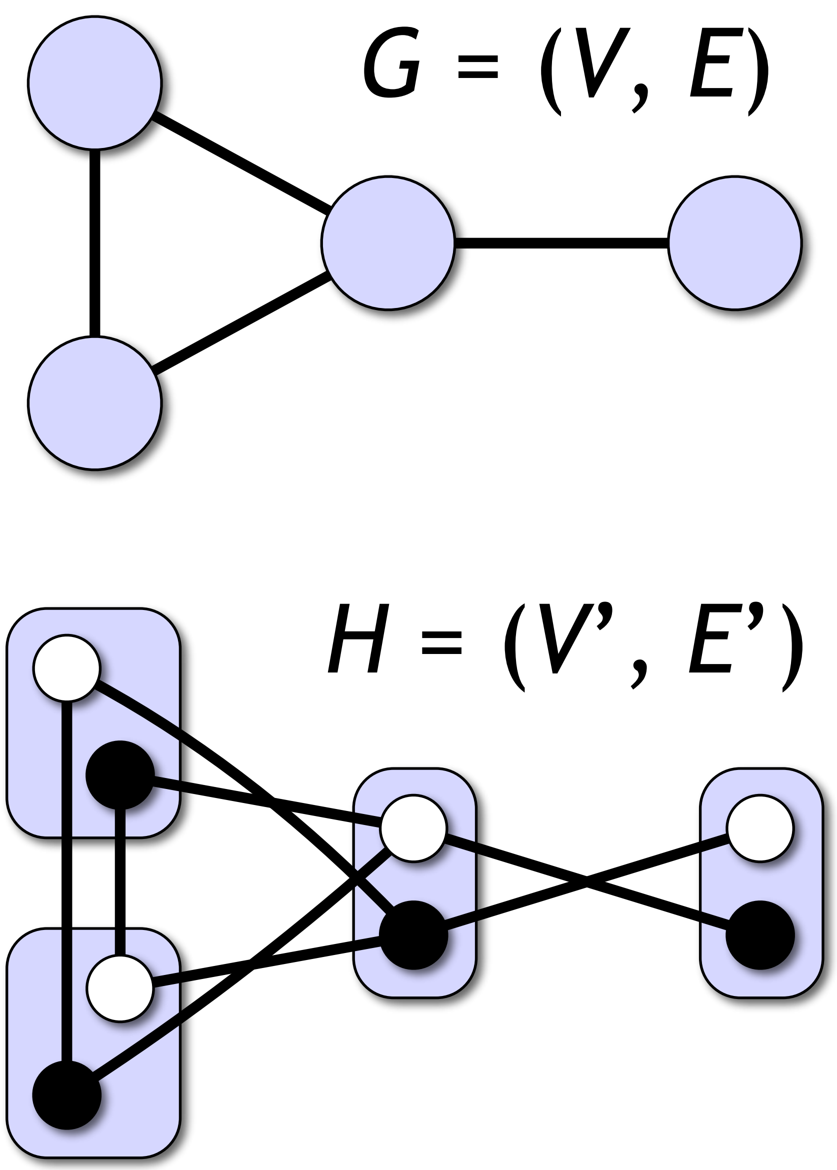
- Some problems *can* be solved in the port-numbering model...
  - and covering graphs can be used as an algorithm design technique, too!
- Example: vertex cover approximation

Symmetry breaking out of thin air: bipartite double covers



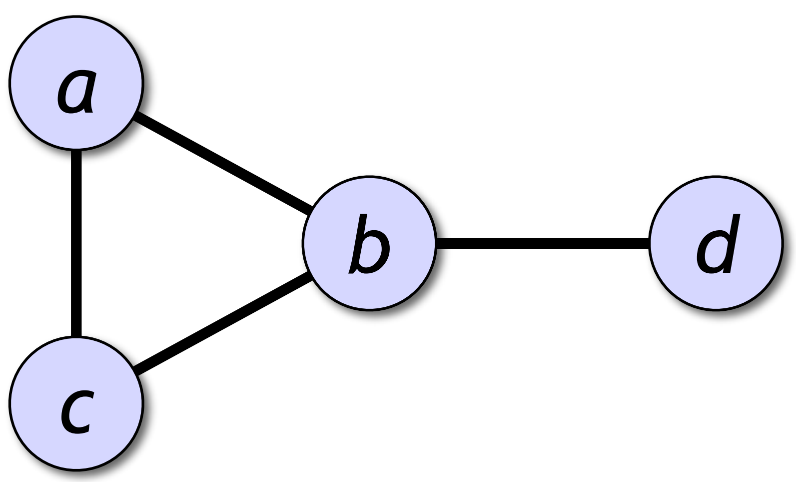
- Replace each node by **two virtual nodes**: black and white
  - original nodes **simulate** virtual nodes
  - each computers runs two programs in parallel: “black program” and “white program”
- Edges: **black-to-white**

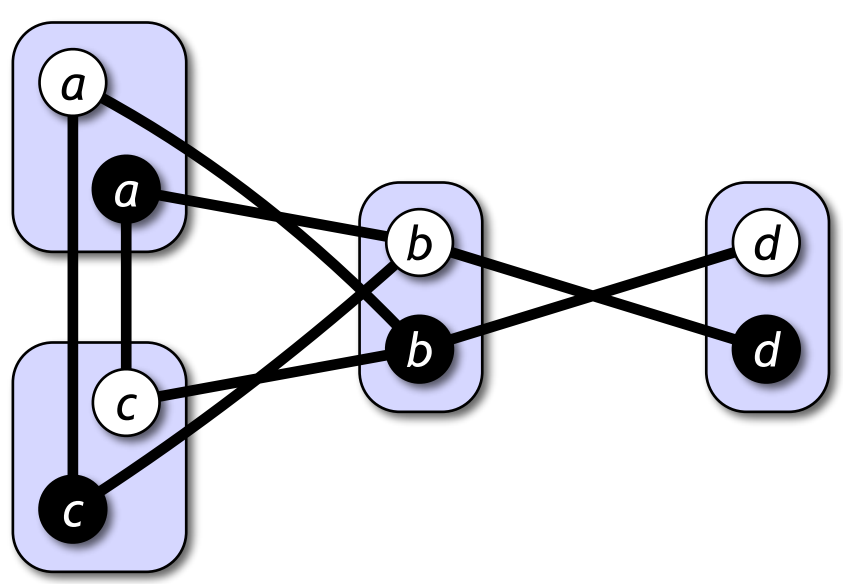
Symmetry breaking out of thin air: bipartite double covers



- Virtual graph H is a **covering graph** of G
- It is a **double** cover: 2 nodes of H map to each node of G
- It is **bipartite**
  - and we have already coloured its two parts: black and white!

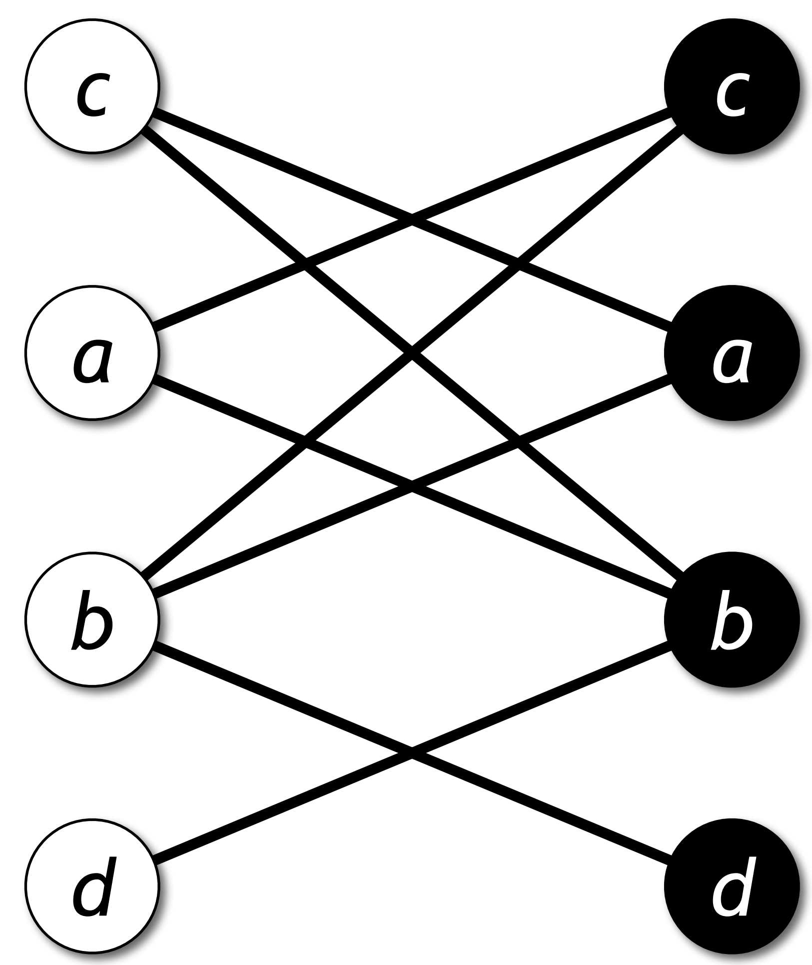
Symmetry breaking out of thin air: bipartite double covers



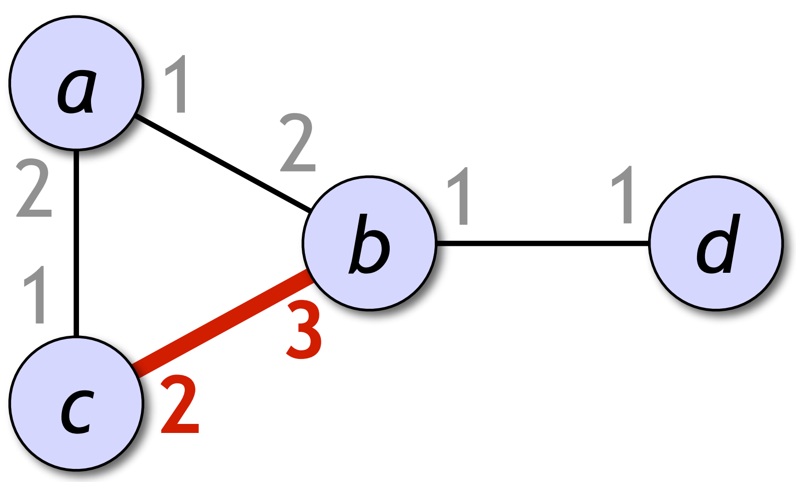


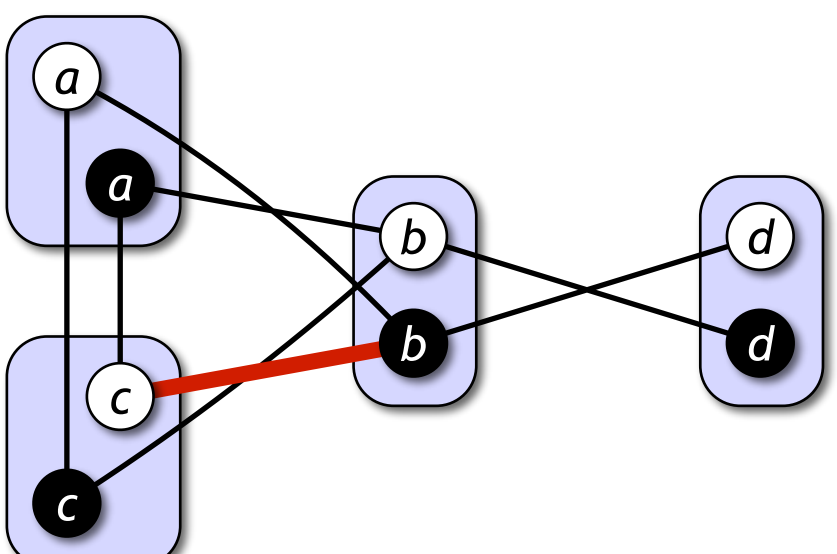
=

2-coloured graph



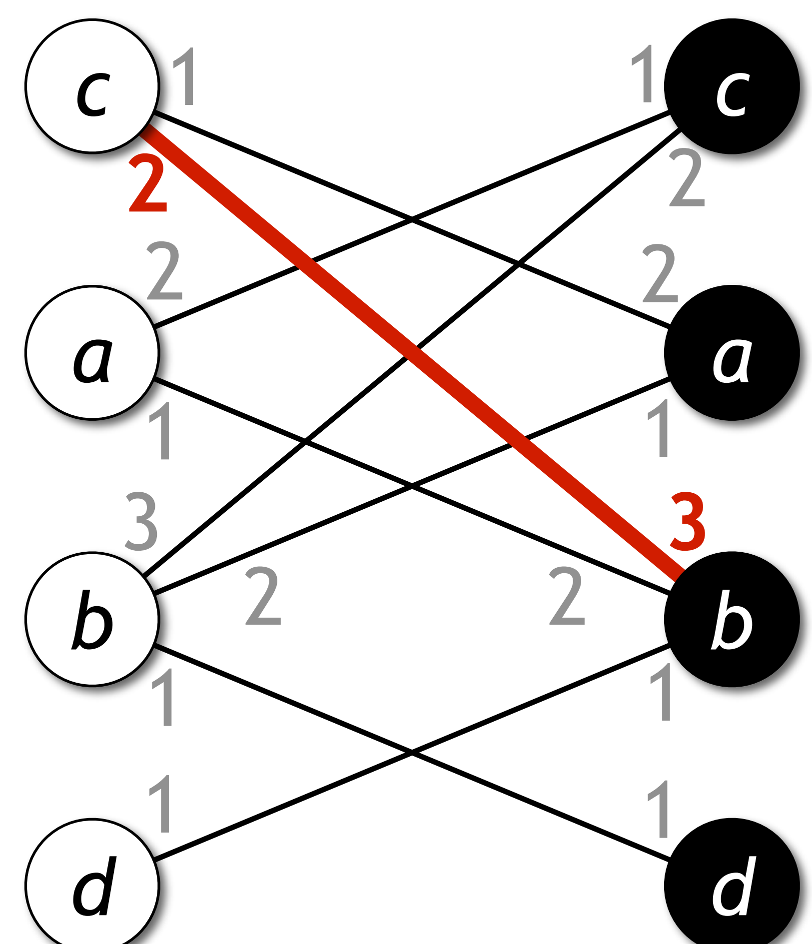
Symmetry breaking out of thin air: bipartite double covers



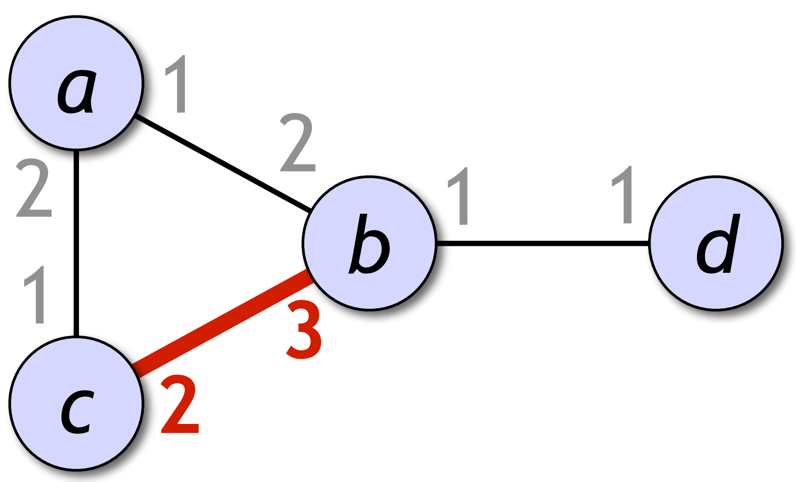


=

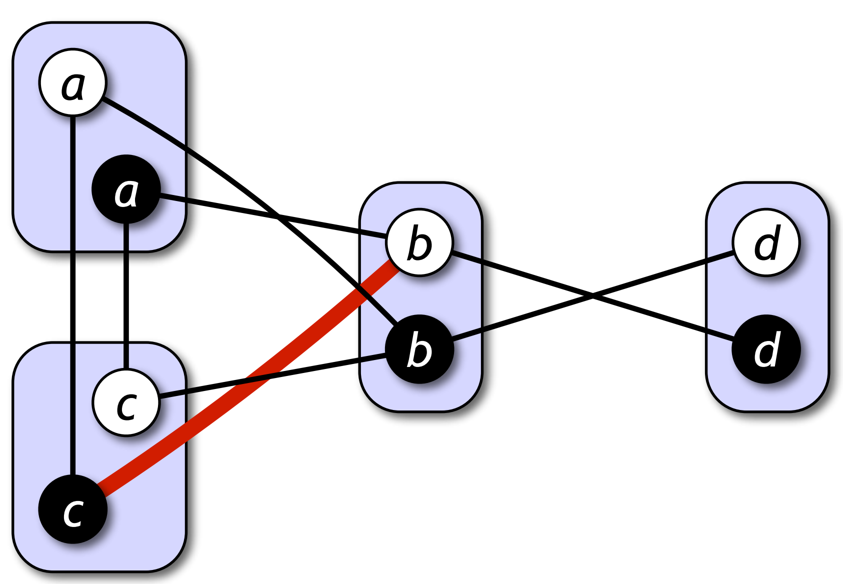
Port-numbering inherited



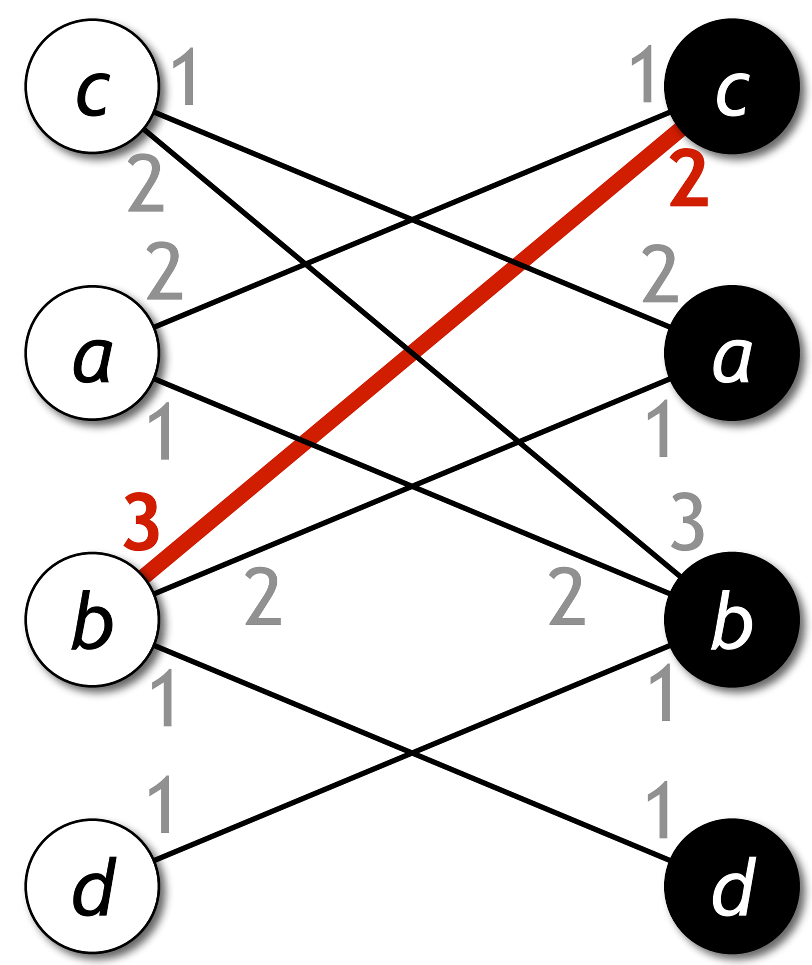
Symmetry breaking out of thin air: bipartite double covers



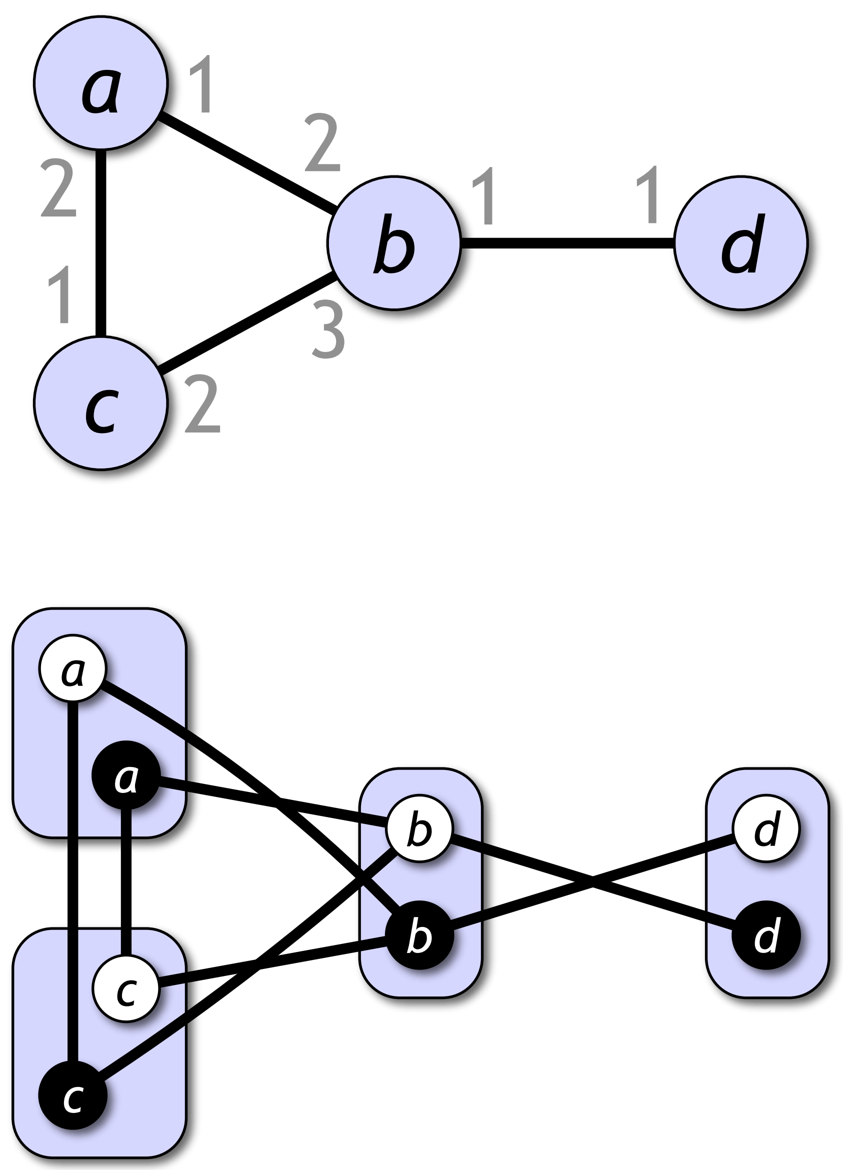
Port-numbering inherited



=

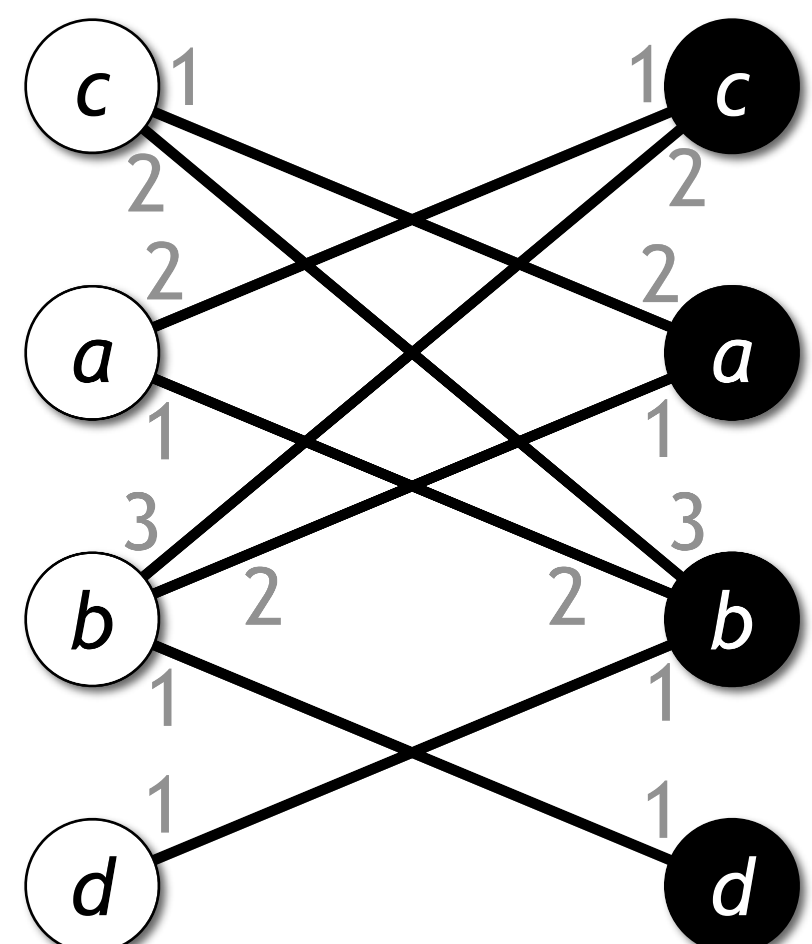


Symmetry breaking out of thin air: bipartite double covers



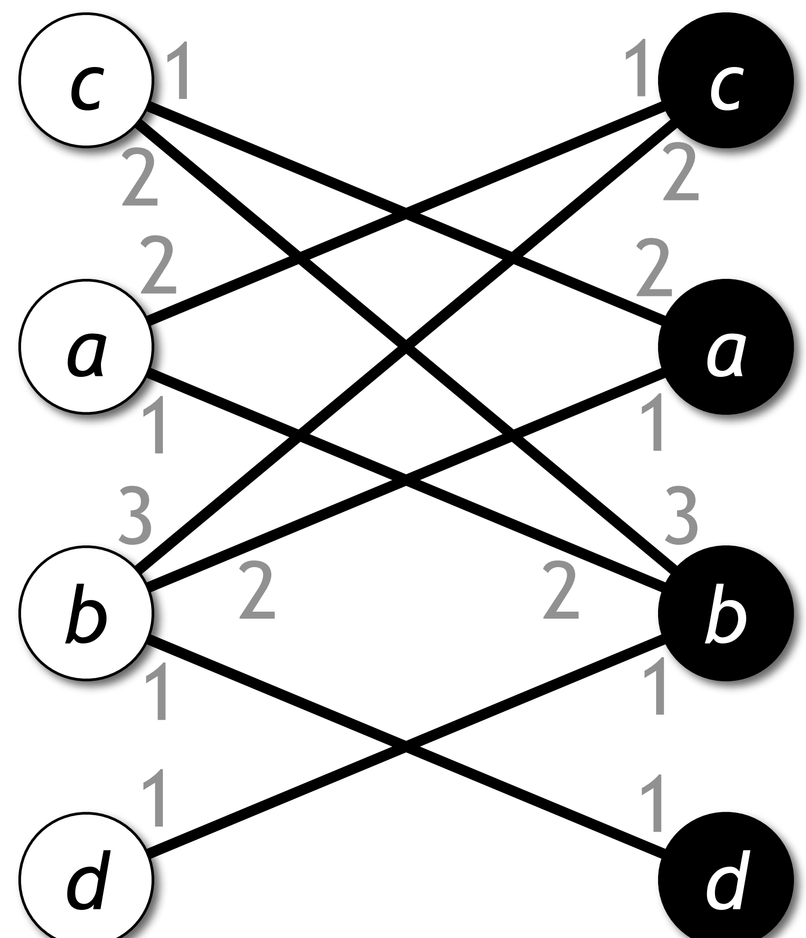
=

Port-numbering inherited



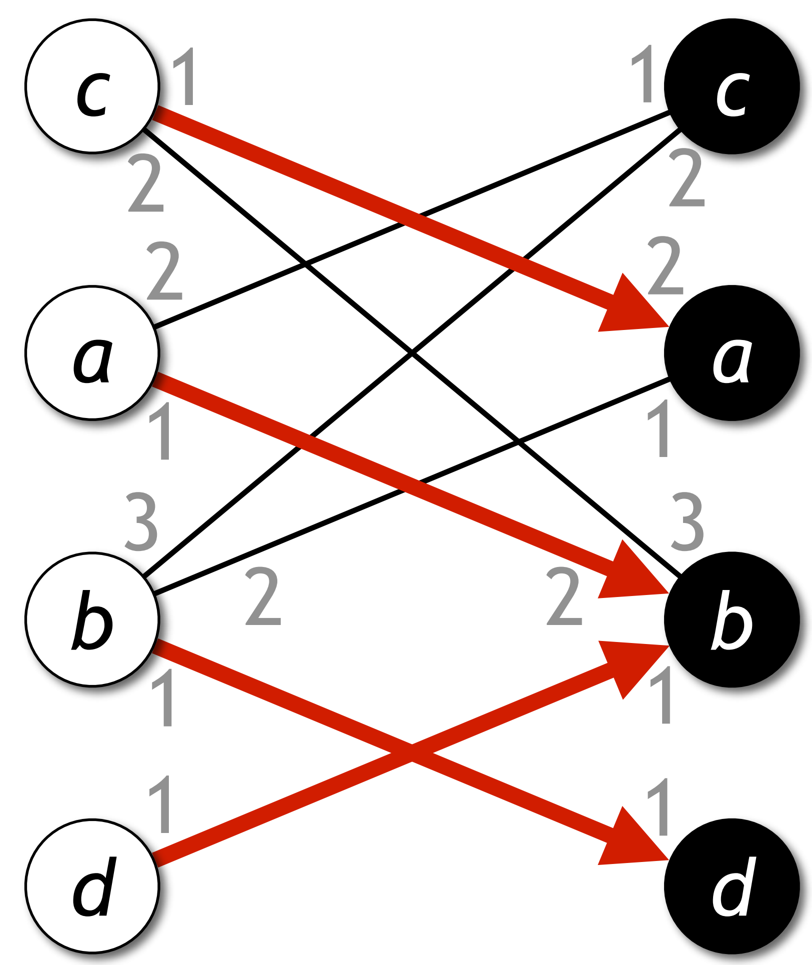
Symmetry breaking out of thin air: bipartite double covers

- Port-numbered graphs without colouring:
  - not possible to find a maximal matching (consider an even cycle)
- Port-numbered graphs with 2-colouring:
  - very easy to find a maximal matching!



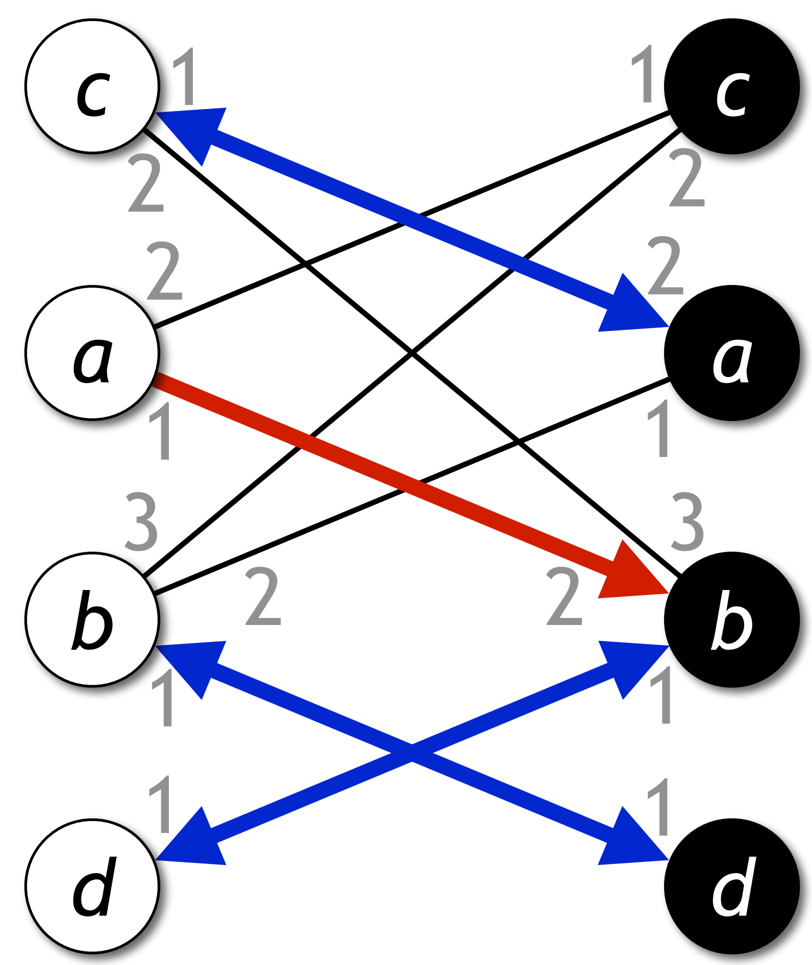
Maximal matching in 2-coloured graphs

- Each white node sends **proposals** to its black neighbours
    - one by one,
      order by port numbers



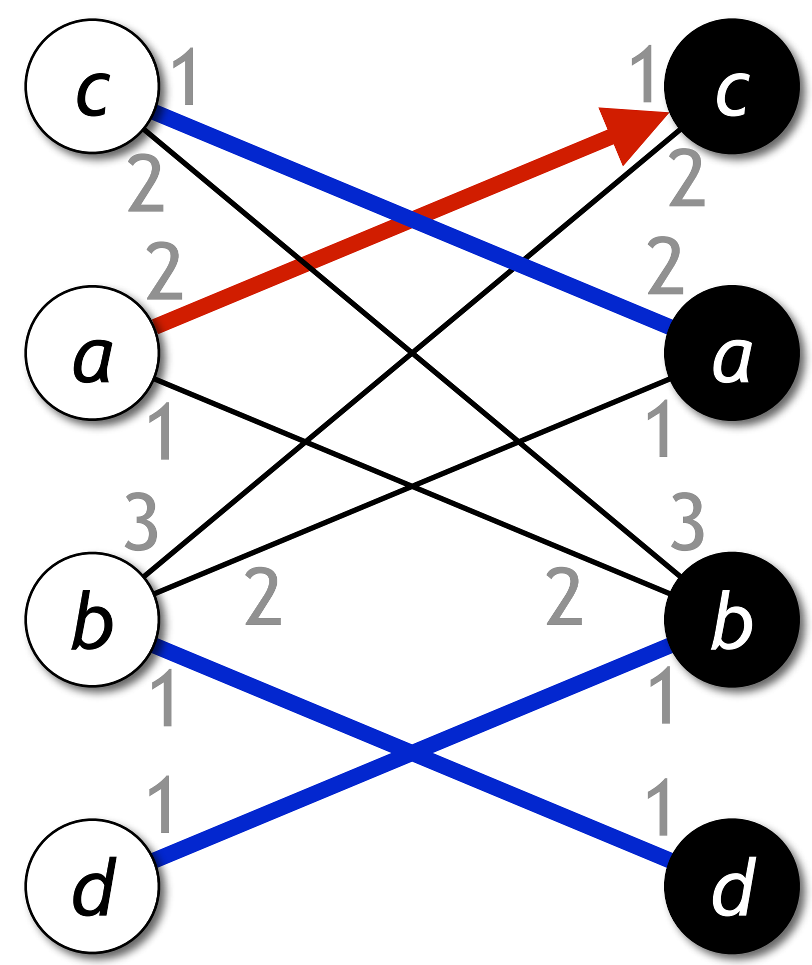
Maximal matching in 2-coloured graphs

- Each white node sends **proposals** to its black neighbours
  - one by one,
    order by port numbers
- Each black node **accepts** the first proposal it gets
  - break ties using
    port numbers



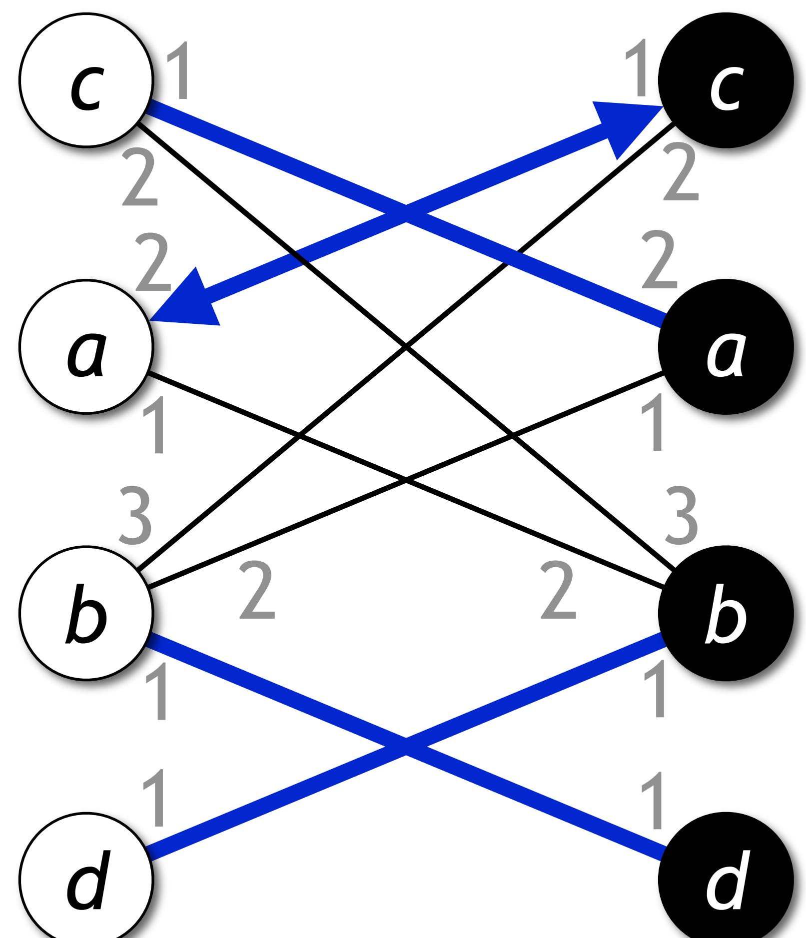
Maximal matching in 2-coloured graphs

- Each white node sends **proposals** to its black neighbours
  - one by one, order by port numbers
  - until its proposal is accepted, or all neighbours have rejected



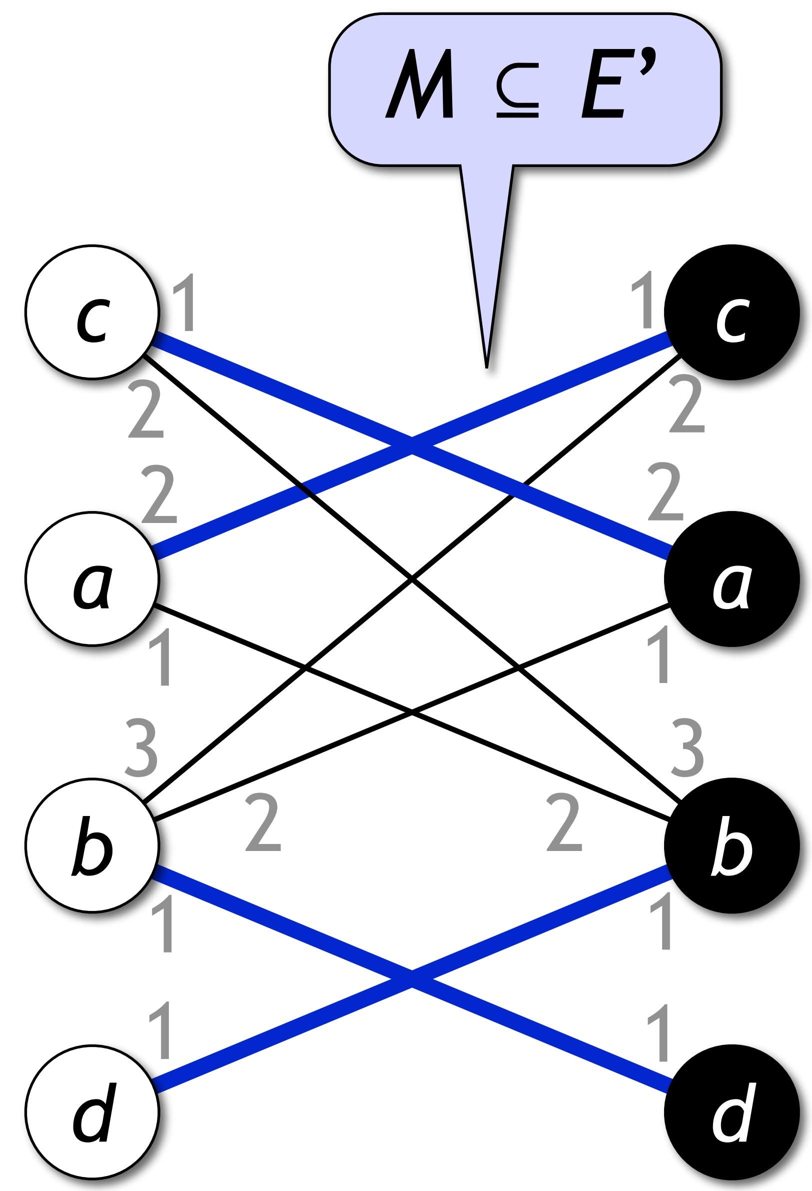
Maximal matching in 2-coloured graphs

- Each white node sends **proposals** to its black neighbours
  - one by one,
    order by port numbers
- Each black node **accepts** the first proposal it gets
  - break ties using
    port numbers



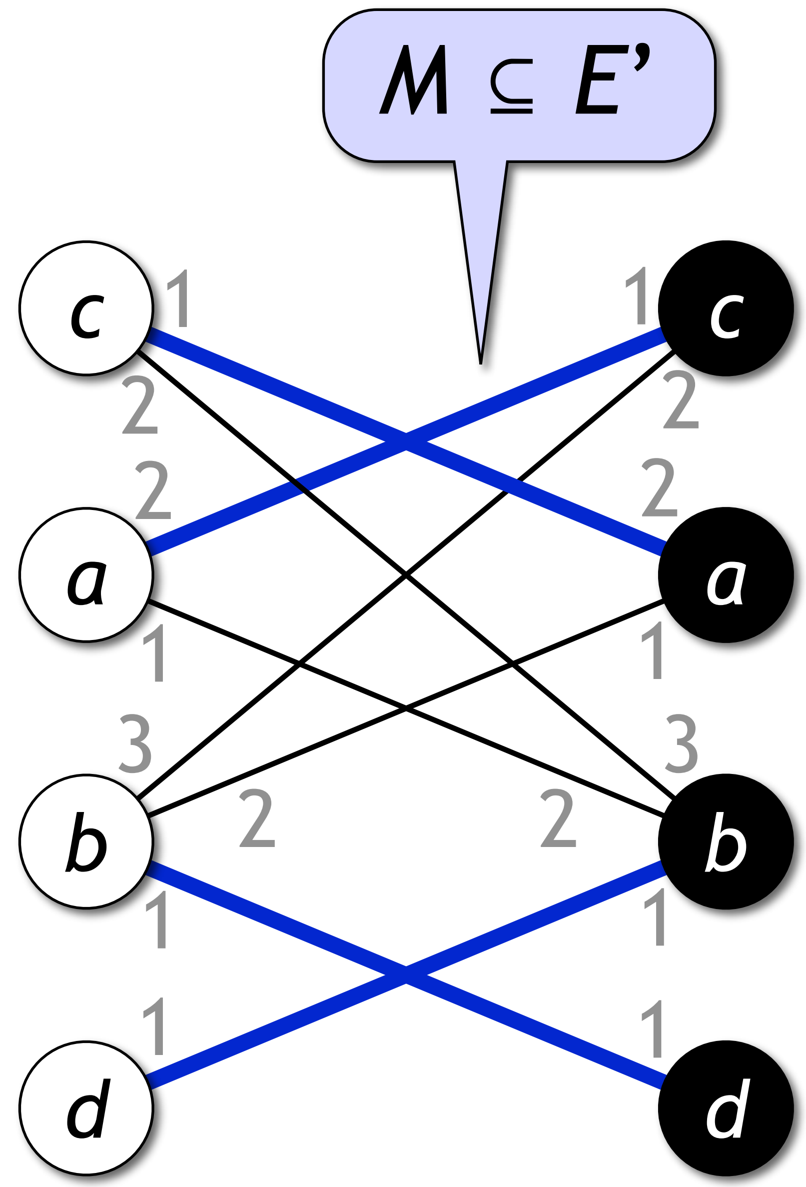
Maximal matching in 2-coloured graphs

- Accepted proposals M: **matching**
  - white nodes don't propose after acceptance
  - black nodes don't accept more than once
  - all nodes incident to at most one edge

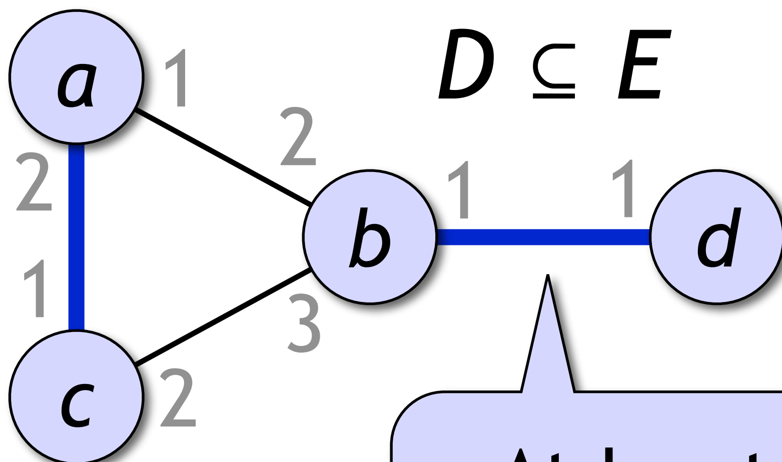


Maximal matching in 2-coloured graphs

- Accepted proposals *M*:
  maximal matching!
  - assume $\{u, v\} \in E \setminus M$
    u unmatched
  - then *u* has sent a proposal to *v* and *v* has rejected it
  - therefore *v* had already received another proposal, v is matched
  - can't add $\{u, v\}$ to *M*

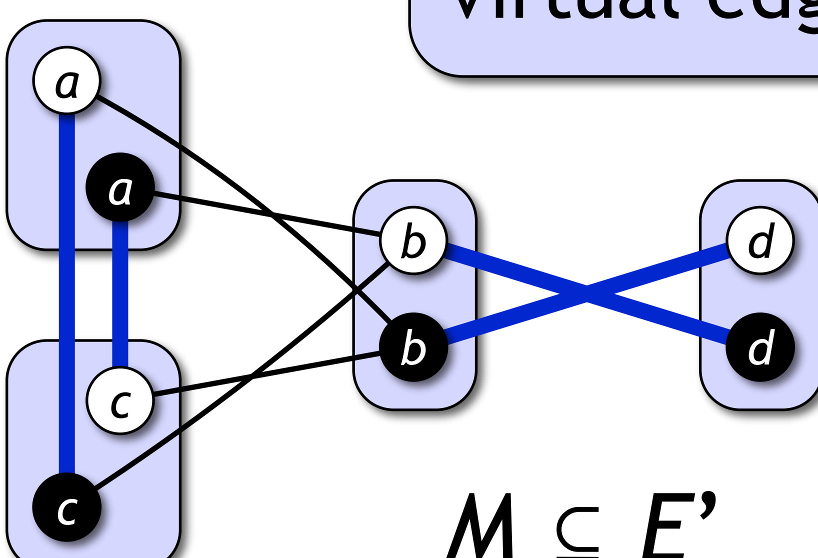


Maximal matching in bipartite double cover



$D \subseteq E$

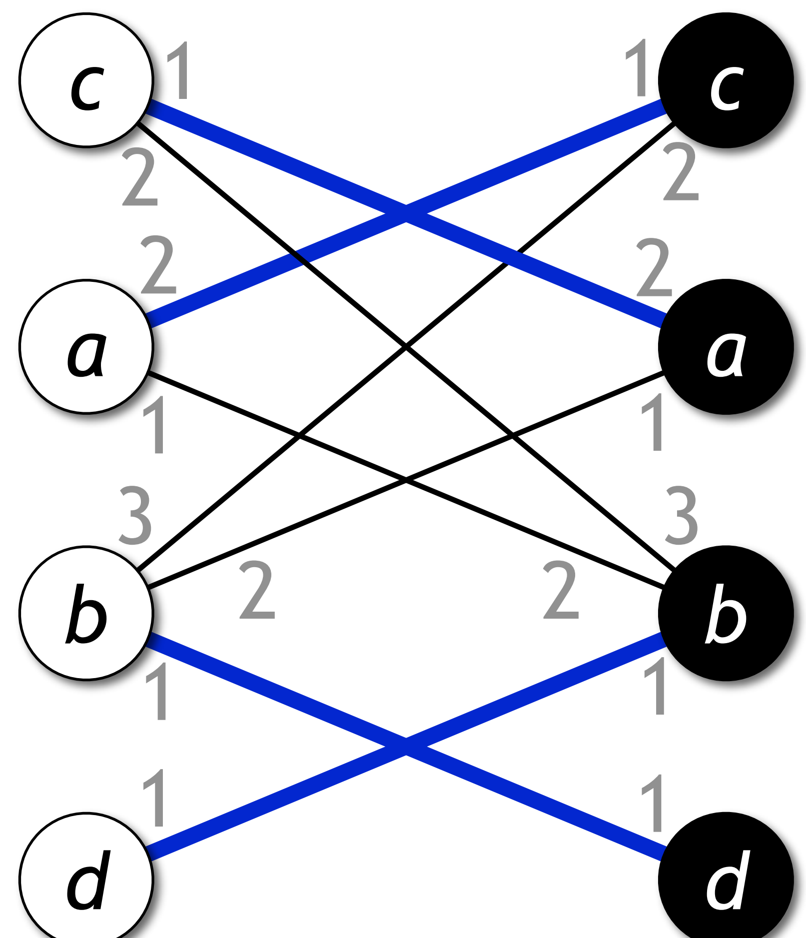
At least 1 of 2 virtual edges in M



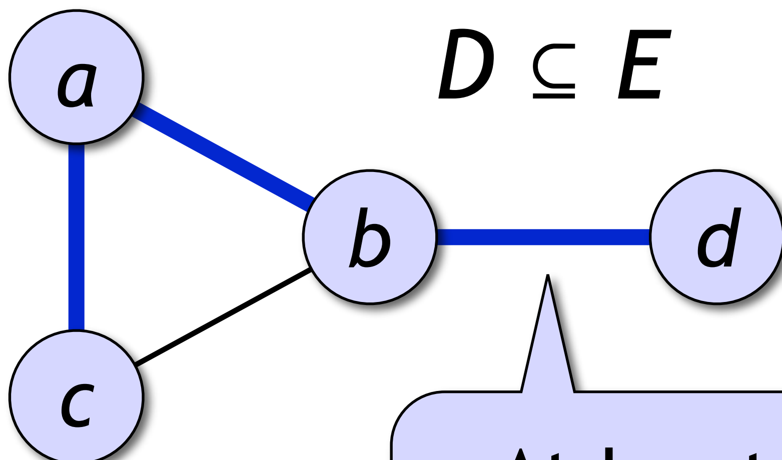
$M \subseteq E'$

=

Map back to original graph

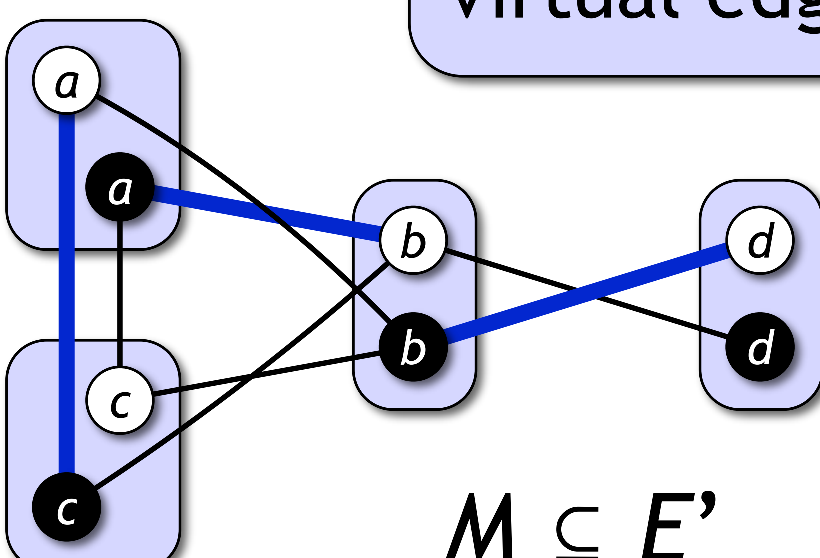


Maximal matching in bipartite double cover



$D \subseteq E$

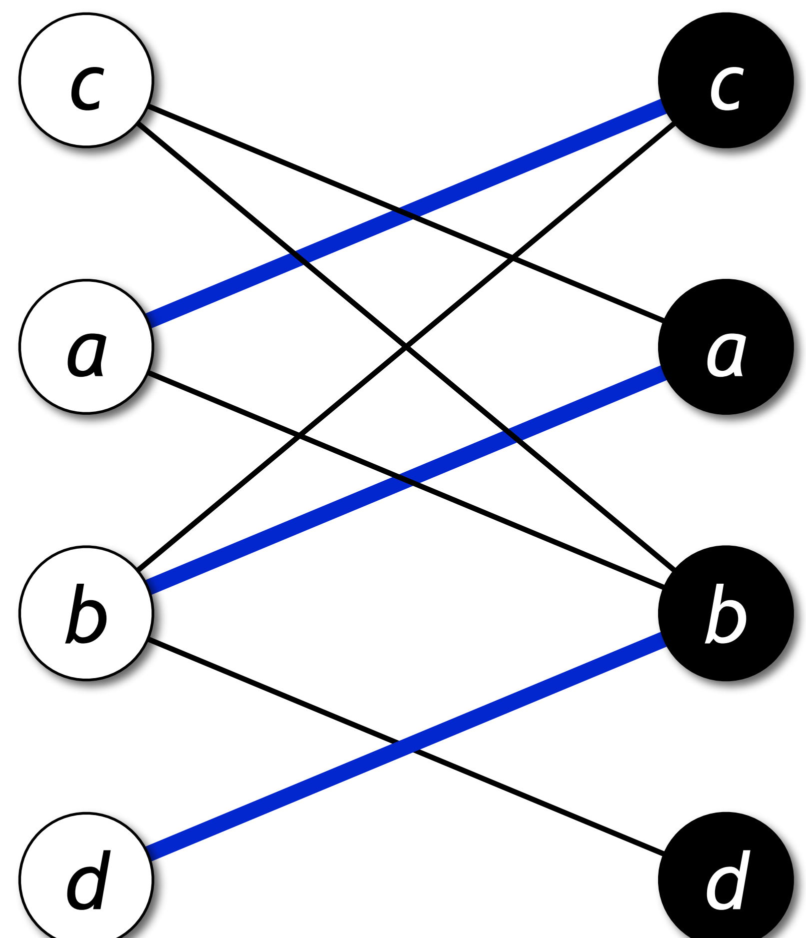
At least 1 of 2 virtual edges in M



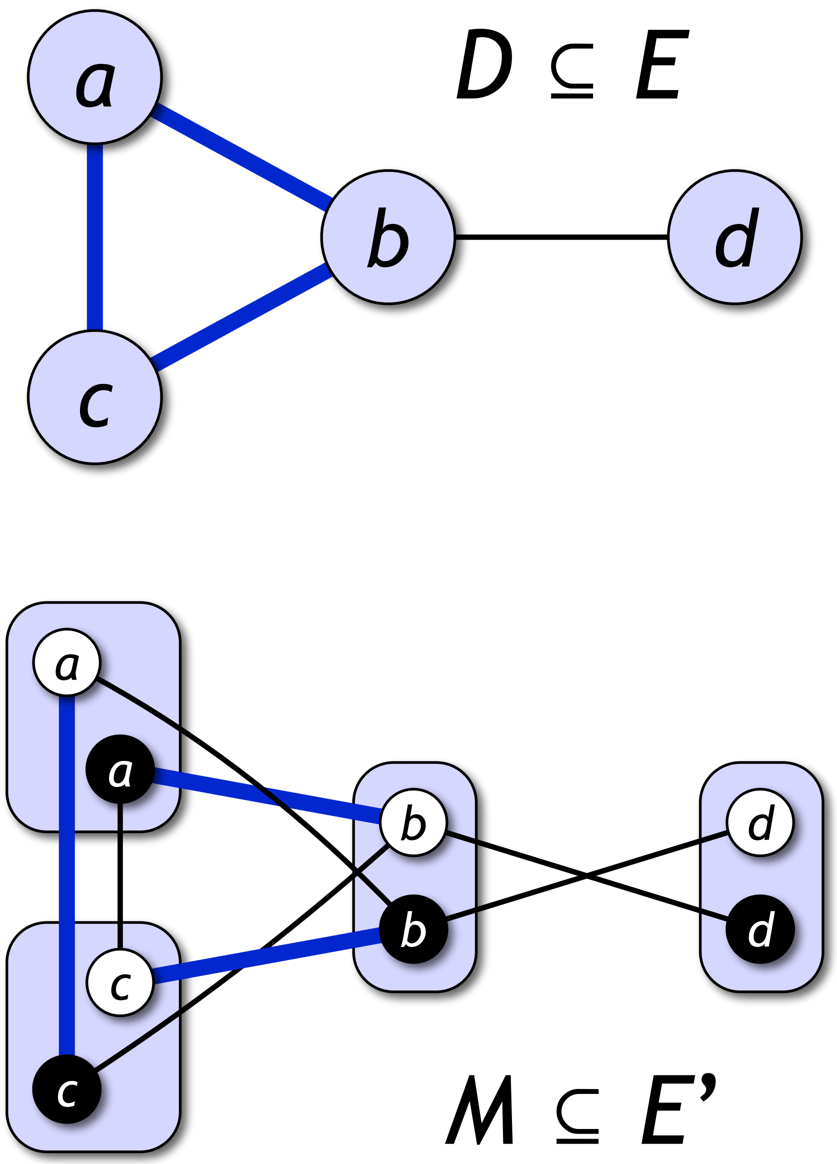
$M \subseteq E'$

Different possibilities...

=

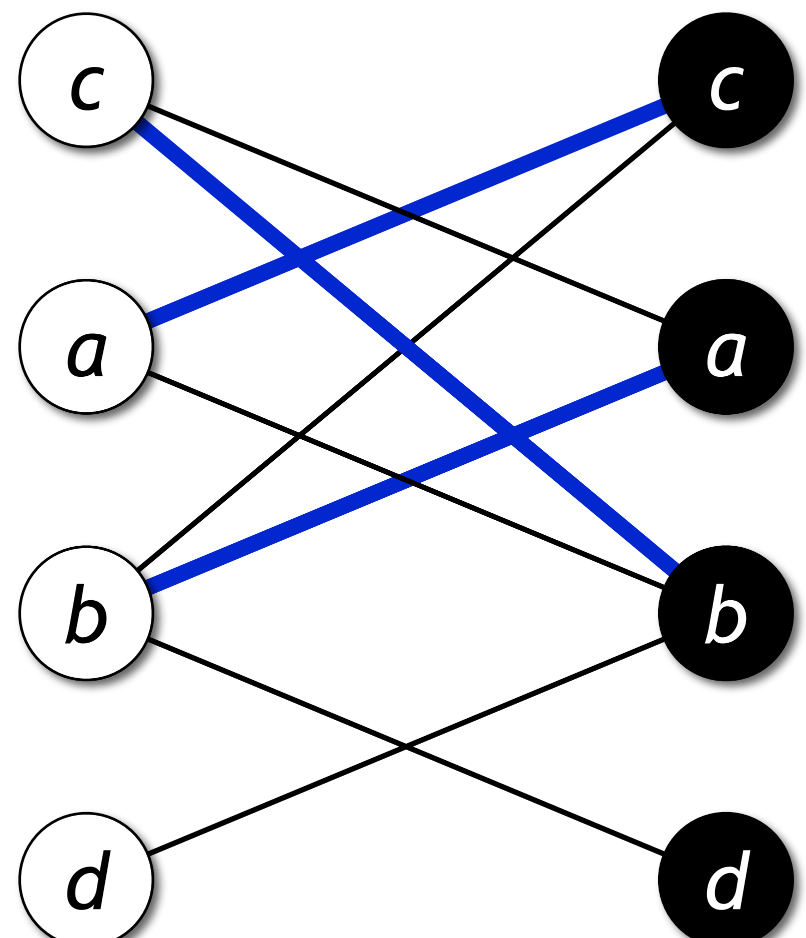


Maximal matching in bipartite double cover

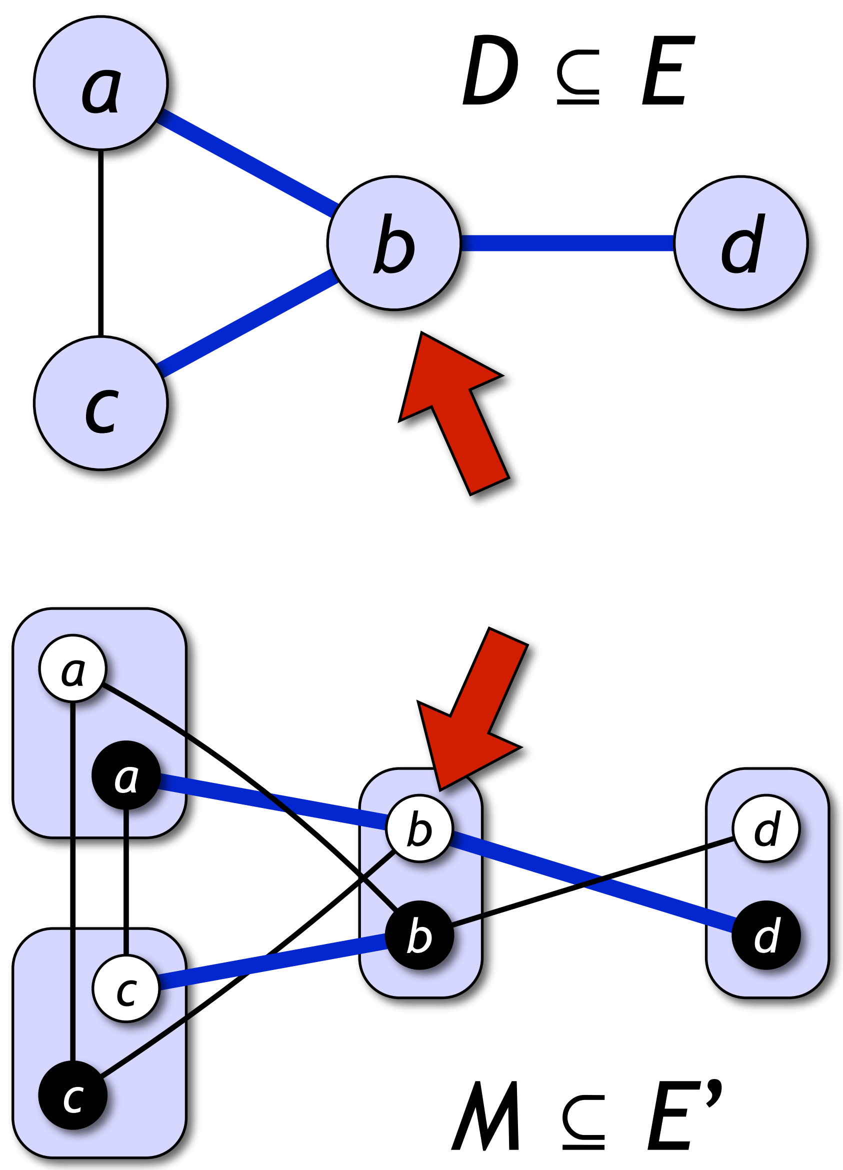


=

Different possibilities...

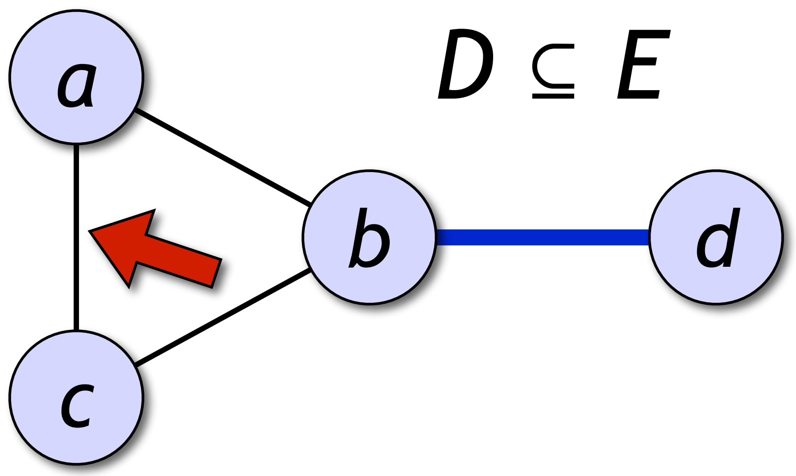


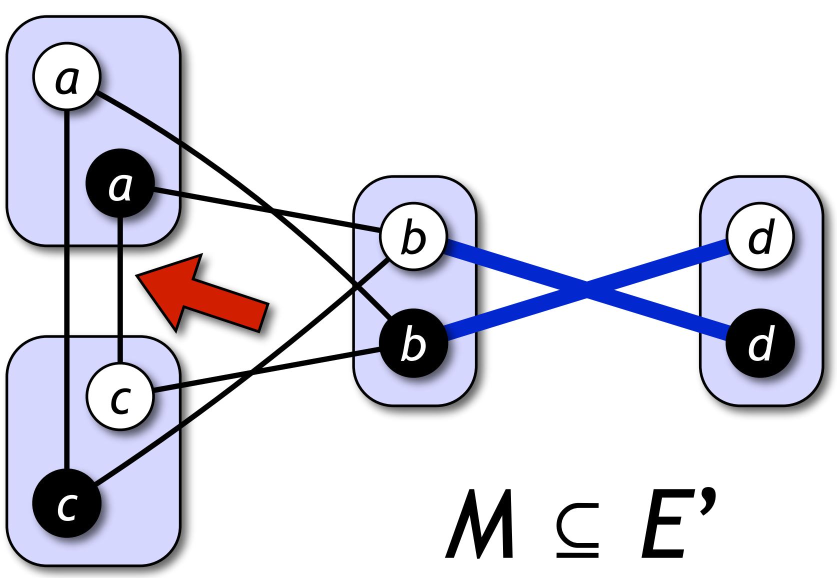
Maximal matching in bipartite double cover



- However, this is not possible, because *M* is a matching
  - *M* induces a subgraph of *H* with max. degree **1**
  - therefore: *D* induces a subgraph of *G* with max. degree **2**

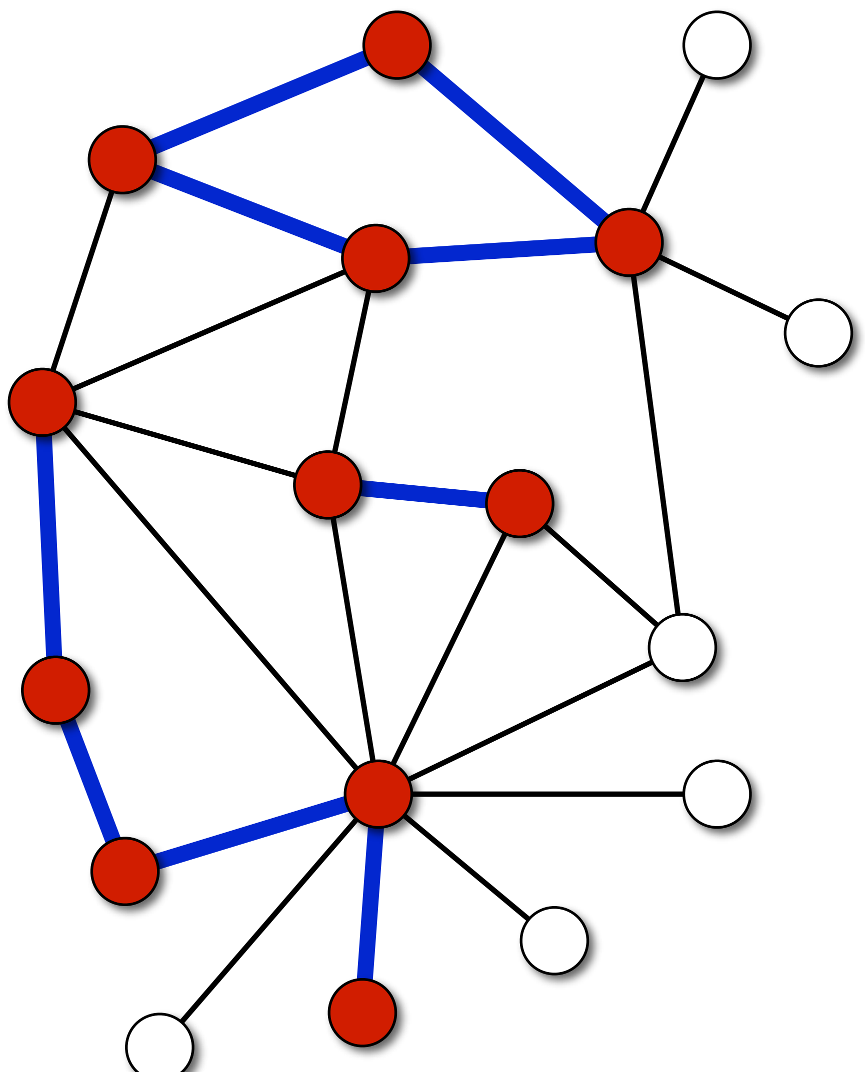
Maximal matching in bipartite double cover





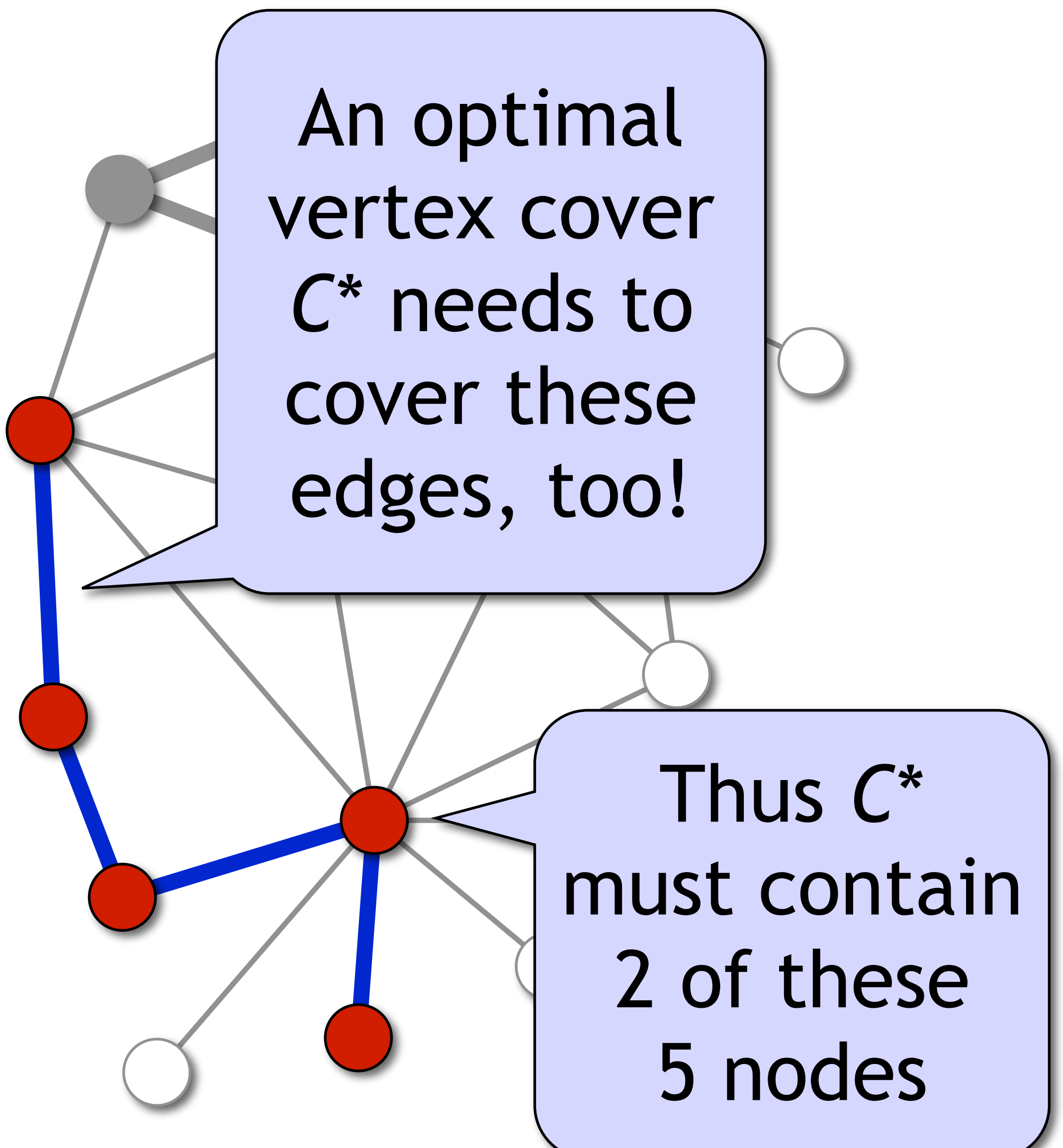
- And this is not possible, because M is maximal
  - each edge of H is in M or shares at least one endpoint with M
  - endpoints of M form a **vertex cover** in H
  - endpoints of D form a **vertex cover** in G!

Finding a vertex cover



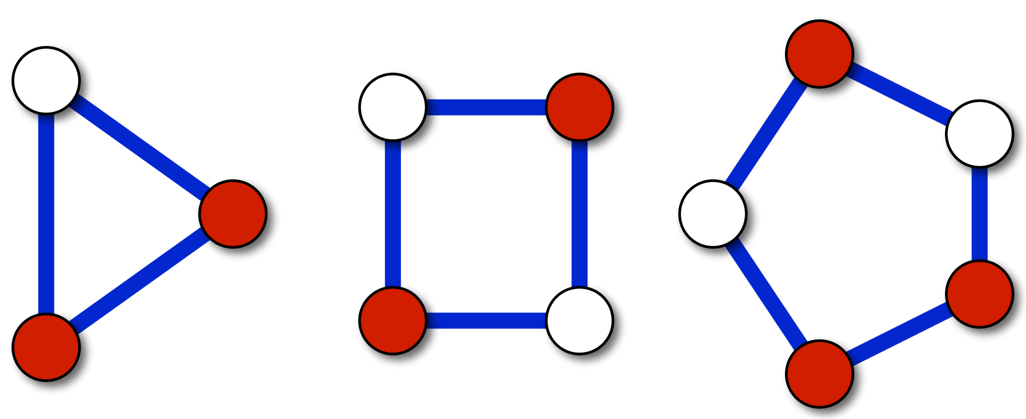
- So we will find a set *D* of edges such that:
  - *D* induces a subgraph of maximum degree 2
  - *D* must consist of **paths** and **cycles**
  - endpoints of *D* form a **vertex cover** *C*
  - is it a small vertex cover?

Finding a vertex cover



- So we will find a set D of edges such that:
  - D induces a subgraph of maximum degree 2
  - D must consist of **paths** and **cycles**
  - endpoints of D form a **vertex cover** C
  - is it a small vertex cover?

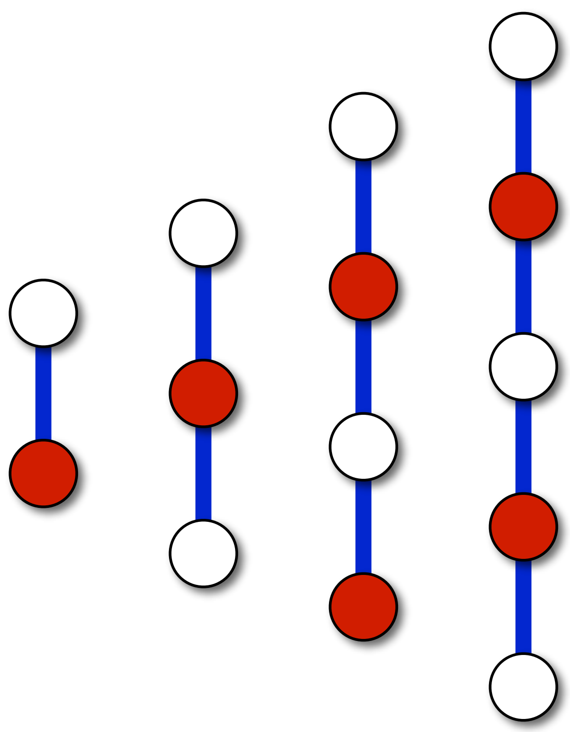
Finding a vertex cover



- Different cases:
  - Cycle with 3 edges:
    3 nodes in C, ≥ 2 in C^*
  - Cycle with 4 edges:
    4 nodes in C, ≥ 2 in C^*
  - Cycle with 5 edges:
    5 nodes in C, ≥ 3 in C^*
    ...

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

Finding a vertex cover

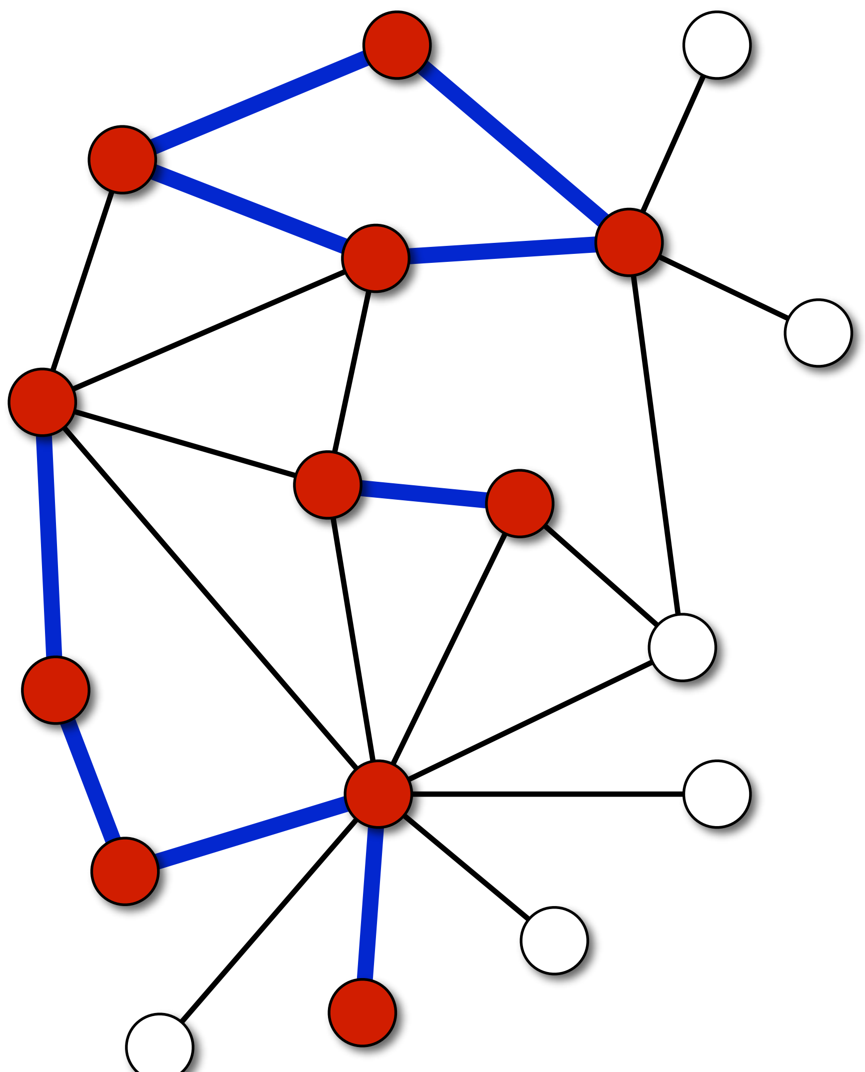


$|C| \leq 3|C^*|$

- Different cases:
  - Path with 1 edge:
    2 nodes in C, ≥ 1 in C^*
  - Path with 2 edges:
    3 nodes in C, $\geq$ **1** in C^*
  - Path with 3 edges:
    4 nodes in C, ≥ 2 in C^*
  - Path with 4 edges:
    5 nodes in C, ≥ 2 in C^*

    ...

Finding a vertex cover



- In each path or cycle:
  - C has at most **3** times as many nodes as C^*
- Summing over all paths and cycles:
  - $|C| \leq \mathbf{3}|C^*|$
- The algorithm finds a **3-approximation** of minimum vertex cover!

Finding a vertex cover: summary

- Vertex cover is a graph problem that *can* be solved reasonably well in the port-numbering model with a deterministic distributed algorithm
  - And the algorithm was simple and fast: $O(\Delta)$ rounds!

Finding a vertex cover: two very different worlds

- Centralised setting, polynomial-time algorithms:
  - **trivial** to find a *minimal vertex cover*: greedy algorithm
  - it requires more thought to find a good *approximation of minimum vertex cover*
- Distributed setting, port-numbering model:
  - **impossible** to find a *minimal vertex cover*: symmetry breaking issues
  - but we have seen that it is possible to find a good *approximation of minimum vertex cover*

Finding a vertex cover: symmetry breaking

- Vertex cover approximation does not require symmetry breaking
  - Proof: algorithm in the port-numbering model
- However, many interesting problems do...
- Let’s study a stronger model of distributed computing