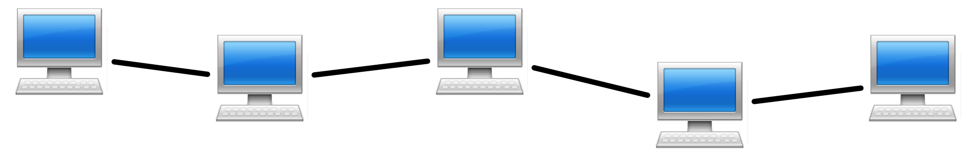
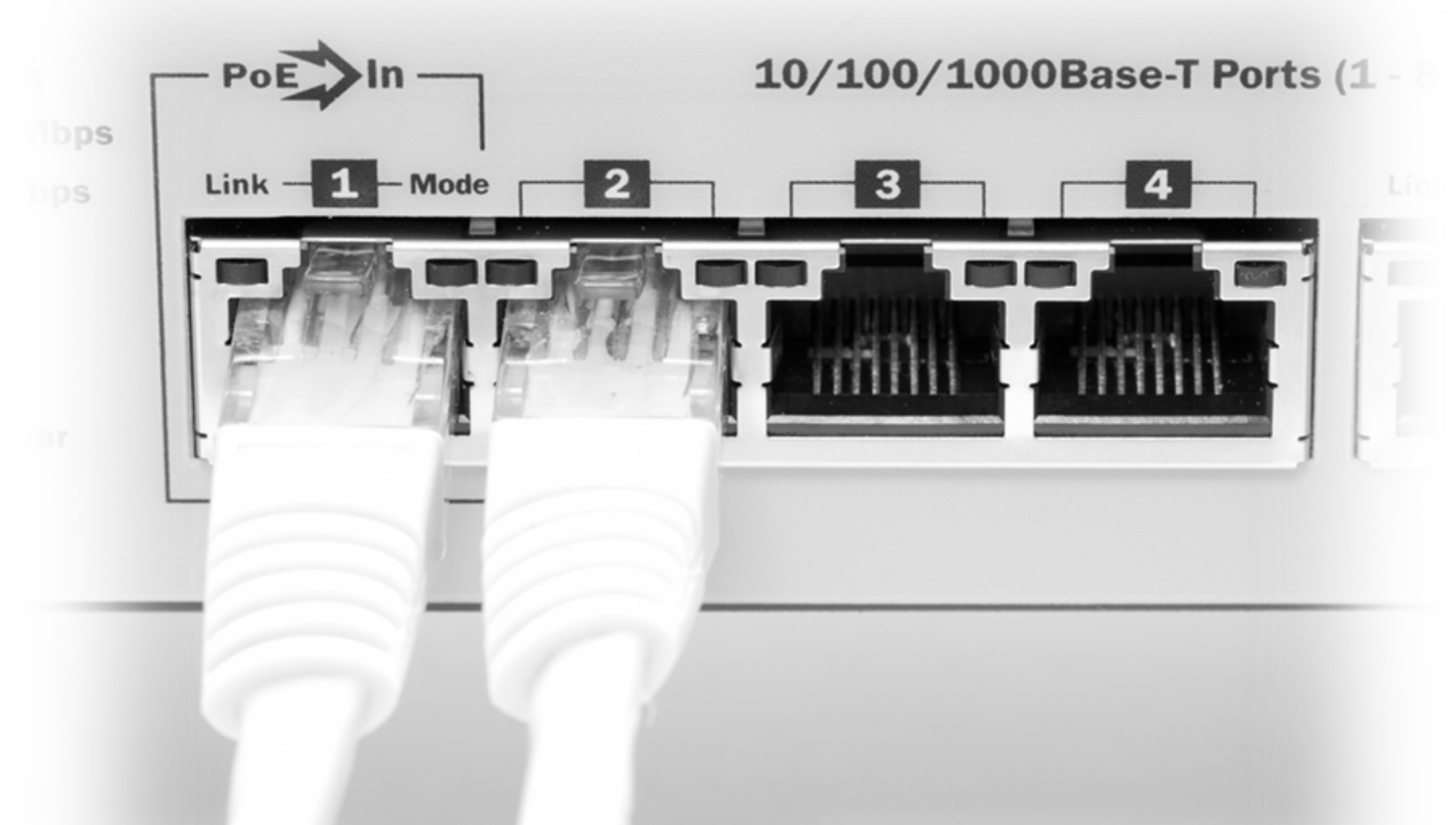
- **Weeks 1–2: informal introduction**
  - network = path 
- **Week 3: graph theory**
- **Weeks 4–7: models of computing**
  - what can be computed (efficiently)?
- **Weeks 8–11: lower bounds**
  - what cannot be computed (efficiently)?
- **Week 12: recap**

Week 4

– PN model: port numbering

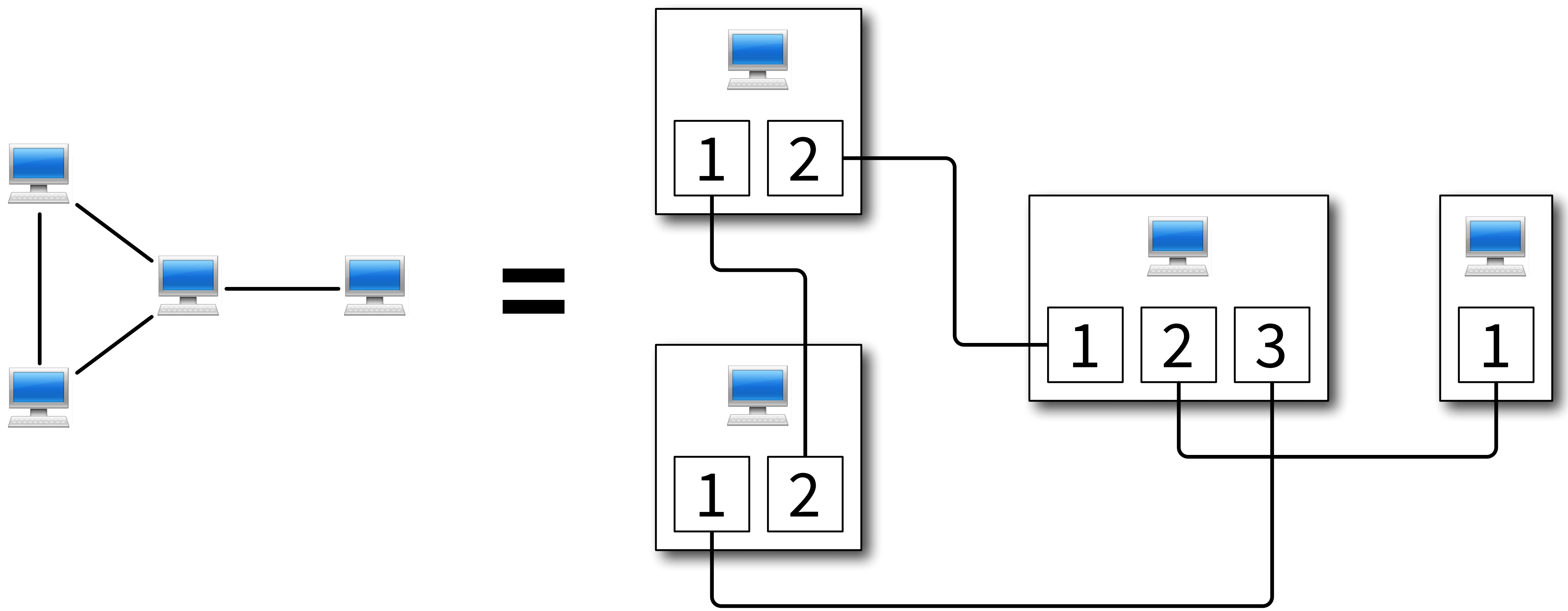
Port-numbering model



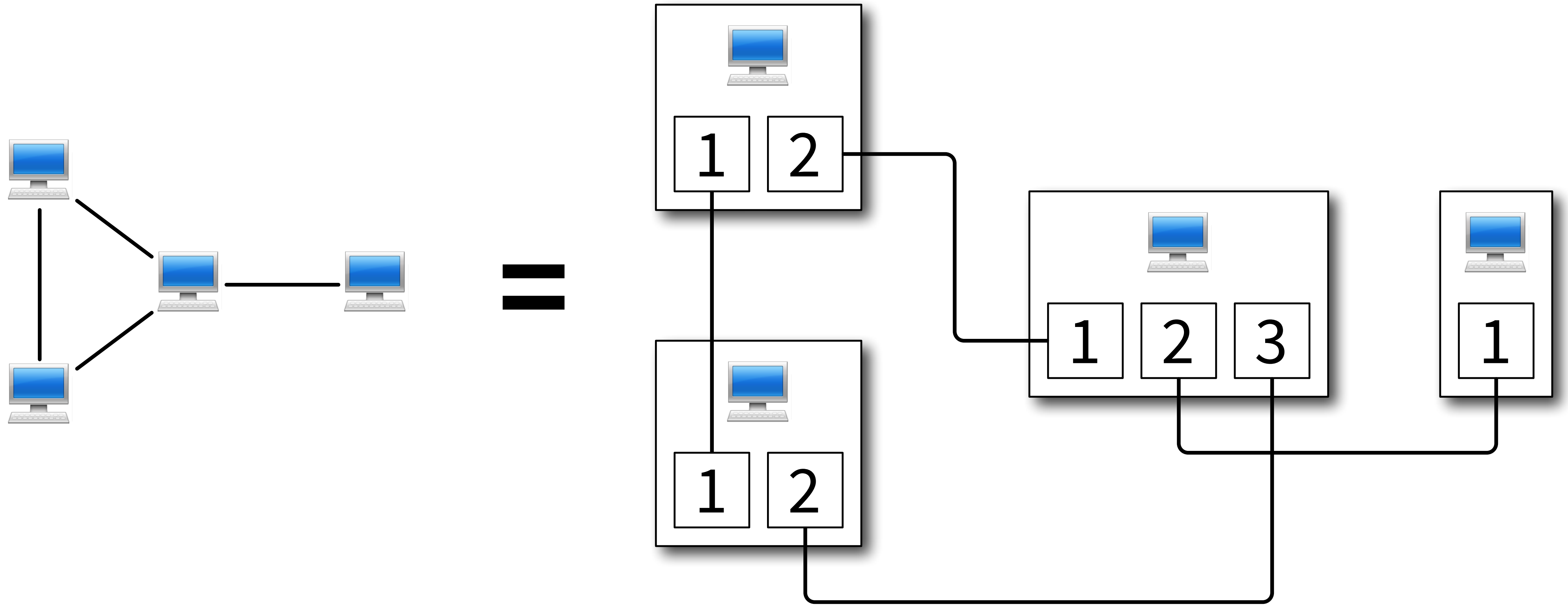
Port-numbering model

- **Simple and restrictive**
  - anonymous nodes, deterministic algorithms
- **All other models are extensions of PN model:**
  - Chapter 5: add unique identifiers
  - Chapter 6: add bandwidth restrictions
  - Chapter 7: add randomness

Port-numbered network

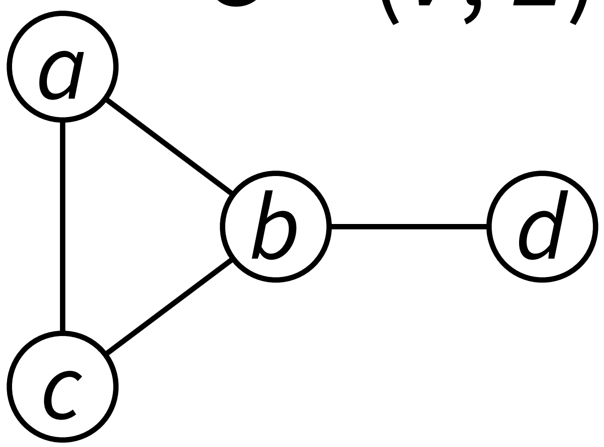


Port-numbered network



Underlying graph

$G = (V, E)$

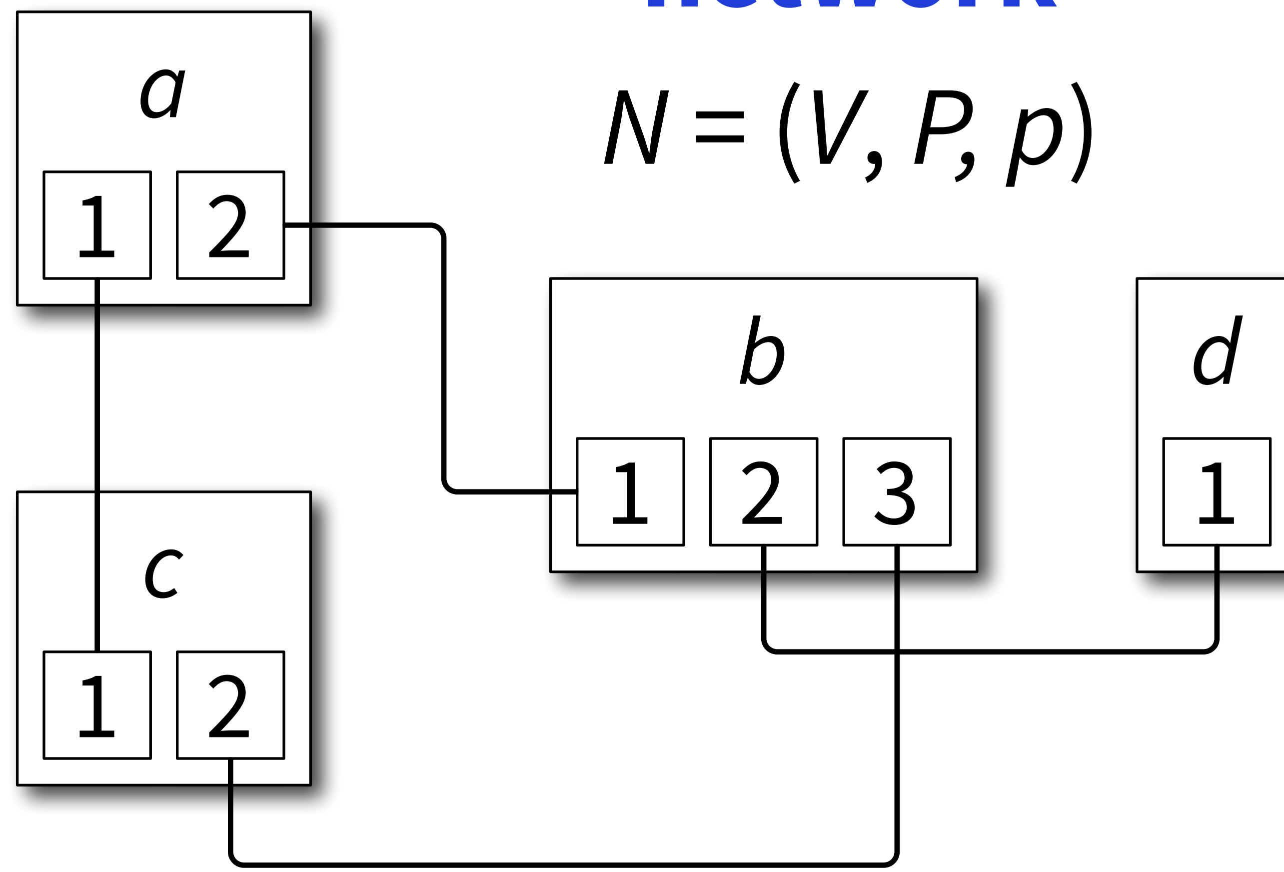


$V = \{a, b, c, d\}$

$E = \{\{a,b\}, \{a,c\}, \{b,c\}, \{b,d\}\}$

Port-numbered network

$N = (V, P, p)$



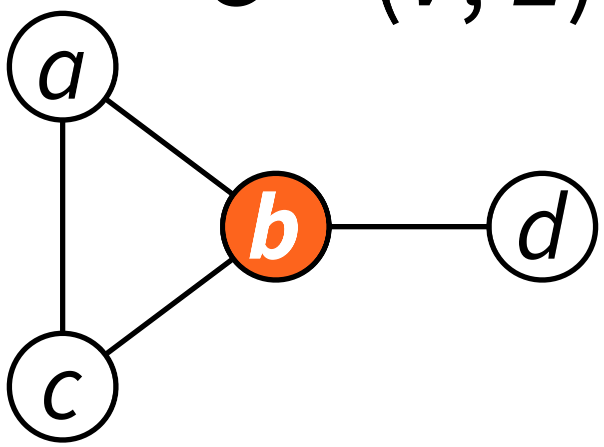
$V = \{a, b, c, d\}$

$P = \{(a,1), (a,2), (b,1), (b,2), (b,3), (c,1), (c,2), (d,1)\}$

$p(a,1) = (c,1),\ p(a,2) = (b,1),\ \ldots$

Underlying graph

$G = (V, E)$

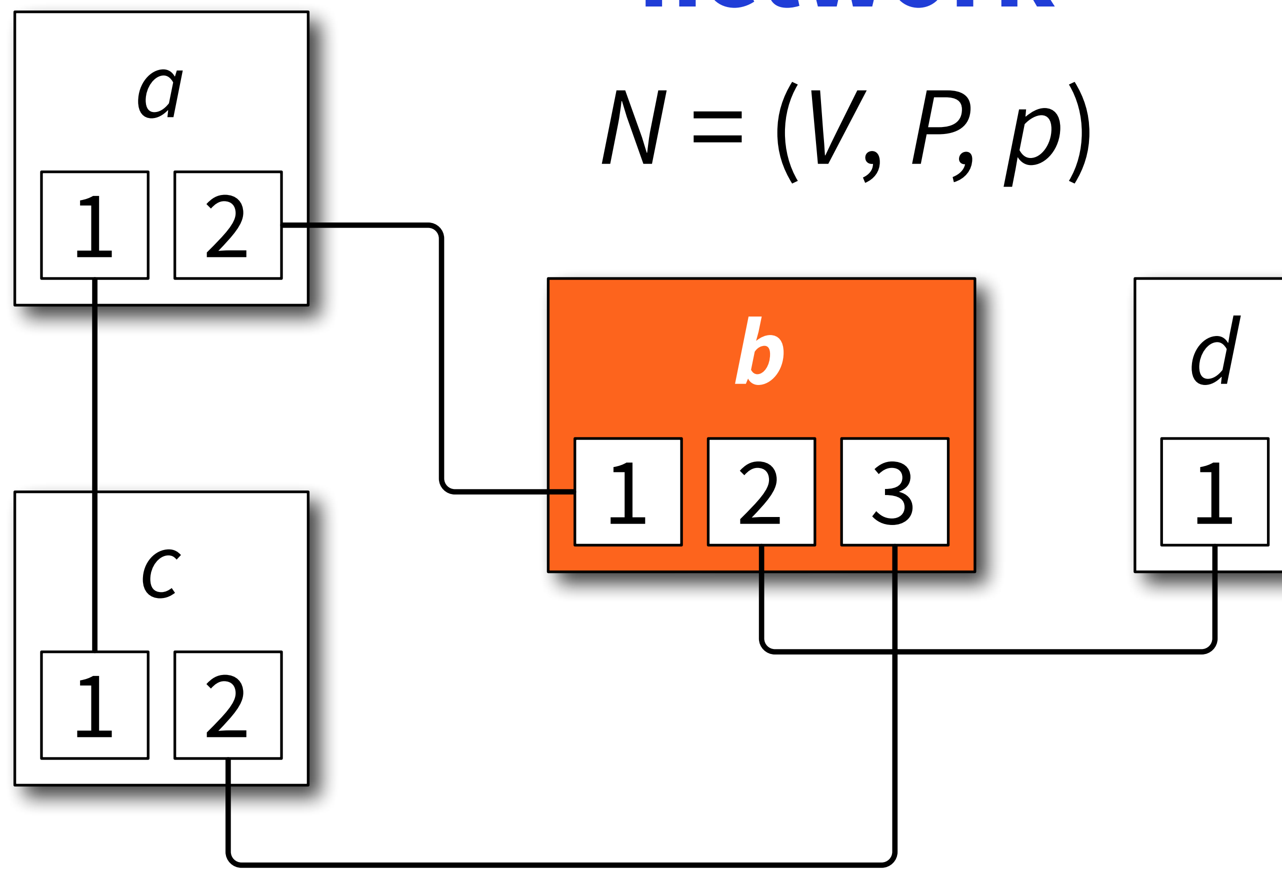


$V = \{a, \mathbf{b}, c, d\}$

$E = \{\{a,b\}, \{a,c\},$
$\{b,c\}, \{b,d\}\}$

Port-numbered network

$N = (V, P, p)$



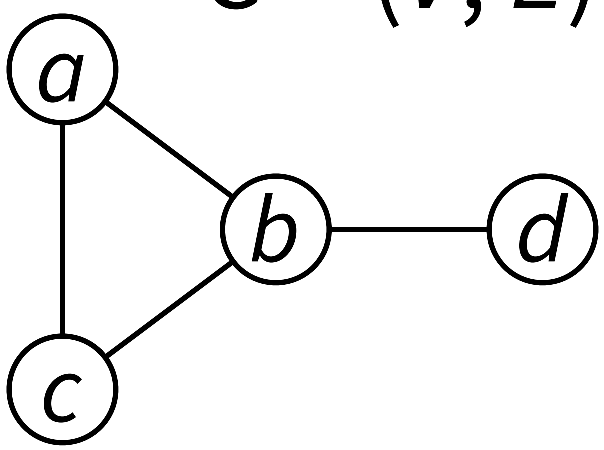
$V = \{a, \mathbf{b}, c, d\}$

$P = \{(a,1), (a,2), (b,1), (b,2),$
$(b,3), (c,1), (c,2), (d,1)\}$

$p(a,1) = (c,1),\ \ p(a,2) = (b,1),\ \ \ldots$

Underlying graph

$G = (V, E)$

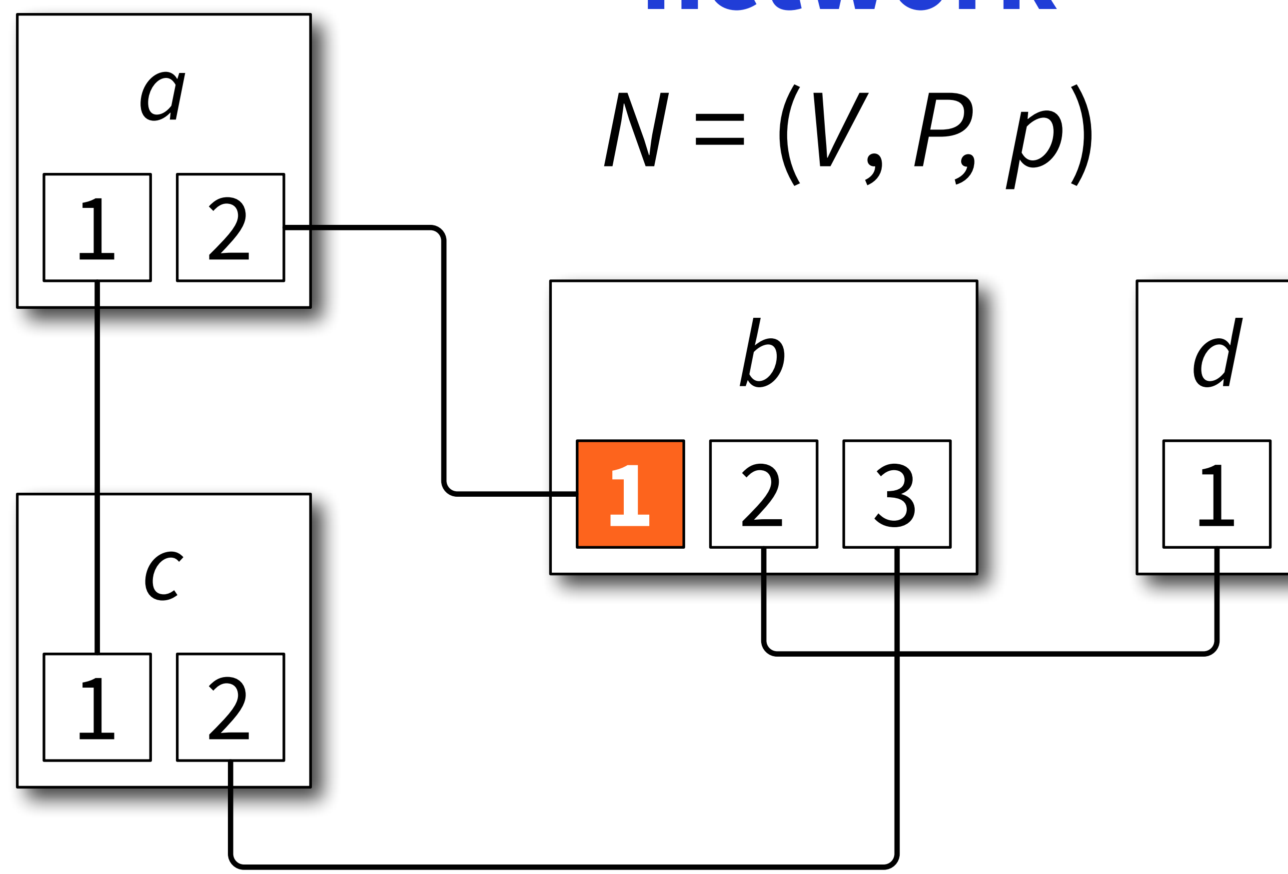


$V = \{a, b, c, d\}$

$E = \{\{a,b\}, \{a,c\}, \{b,c\}, \{b,d\}\}$

Port-numbered network

$N = (V, P, p)$



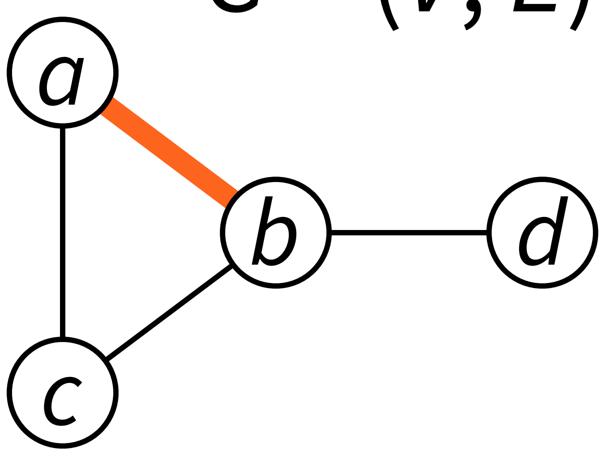
$V = \{a, b, c, d\}$

$P = \{(a,1), (a,2), \mathbf{(b,1)}, (b,2), (b,3), (c,1), (c,2), (d,1)\}$

$p(a,1) = (c,1),\ p(a,2) = (b,1),\ \ldots$

Underlying graph

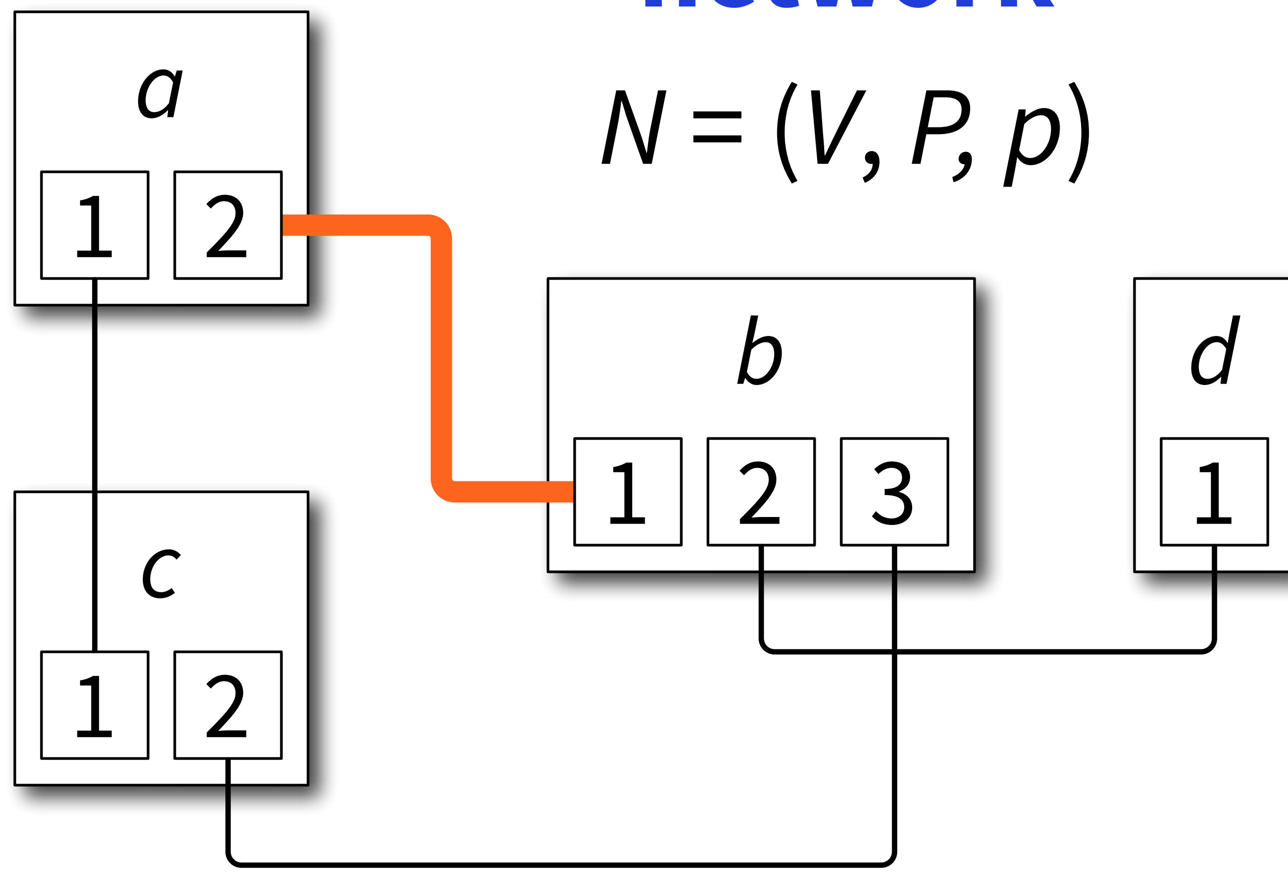
$G = (V, E)$



$V = \{a, b, c, d\}$
$E = \{\{a,b\}, \{a,c\}, \{b,c\}, \{b,d\}\}$

Port-numbered network

$N = (V, P, p)$



$V = \{a, b, c, d\}$
$P = \{(a,1), (a,2), (b,1), (b,2), (b,3), (c,1), (c,2), (d,1)\}$
$p(a,1) = (c,1)$, $\mathbf{p(a,2) = (b,1)}$, …

Distributed algorithm in PN model

- **Algorithm = state machine**
  (can have infinitely many states)

- **Input, States, Output, Msg:** sets

- **init$_d$, send$_d$, receive$_d$:**
  functions for each degree d = 0, 1, 2, …

Distributed algorithm in PN model

- **Input** = set of local inputs
- **States** = set of states
- **Output** = set of stopping states
- **Msg** = set of possible messages

Distributed algorithm in PN model

- **init_d: Input → States**

  how to initialise the state machine

- **send_d: States → Msg^d**

  how to construct outgoing messages

- **receive_d: States × Msg^d → States**

  how to process incoming messages

Distributed algorithm in PN model

- $\mathbf{init}_d(x) = y$

  local state at time 0 if local input is x

- $\mathbf{send}_d(x) = (m_1, m_2, \ldots, m_d)$

  what messages to send if local state is x

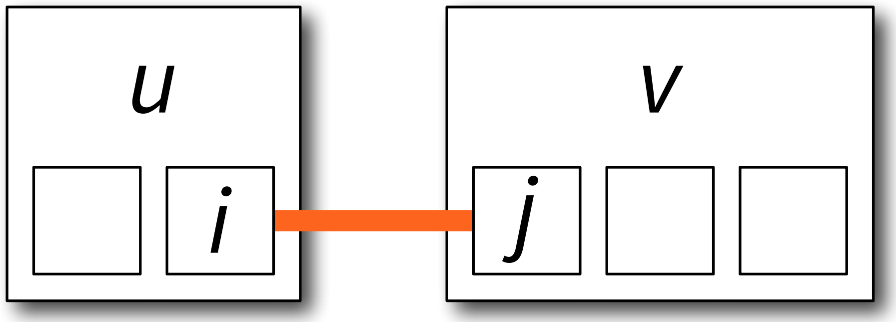
- $\mathbf{receive}_d(x, m_1, m_2, \ldots, m_d) = y$

  new state after receiving these messages

Distributed algorithm in PN model

- **Execution = sequence of state vectors** $\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$
    - $x_t(u)$ = state of node u at time t
- $\boldsymbol{x}_0(\boldsymbol{u}) = \mathbf{init}_d(\boldsymbol{f}(\boldsymbol{u}))$
    - $f(u)$ is the local input of u
    - d = degree of u

Distributed algorithm in PN model



- **Assume $p(u, i) = (v, j)$**
- $\boldsymbol{m_t(u, i)}$ = message received by u from port i
  = message sent by v to port j
  = component j of vector $\text{send}_d(x_{t-1}(v))$
- $\boldsymbol{x_t(u) = \text{receive}_d(x_{t-1}(u), m_t(u, 1), \ldots, m_t(u, d))}$

Distributed algorithm in PN model

- Current state + send → outgoing messages
- Outgoing messages + *p* → incoming messages
- Incoming messages + receive → new state

Distributed algorithm in PN model

- **For any algorithm A and any network N: execution $x_0, x_1, x_2, \ldots$ of A in N**

- **Stops in time T if $x_T(v) \in$ Output for all v**
    - $x_T(v)$ is the local output of v

"*A* solves problem *X* on graph family *F*"

- Take any graph *G* from graph family *F*
- Take any port-numbered network *N* such that *G* is the underlying graph of *N*
- If we run *A* in *N*, then *A* stops and outputs a valid solution of problem *X*

“A solves problem X on family F in time T”

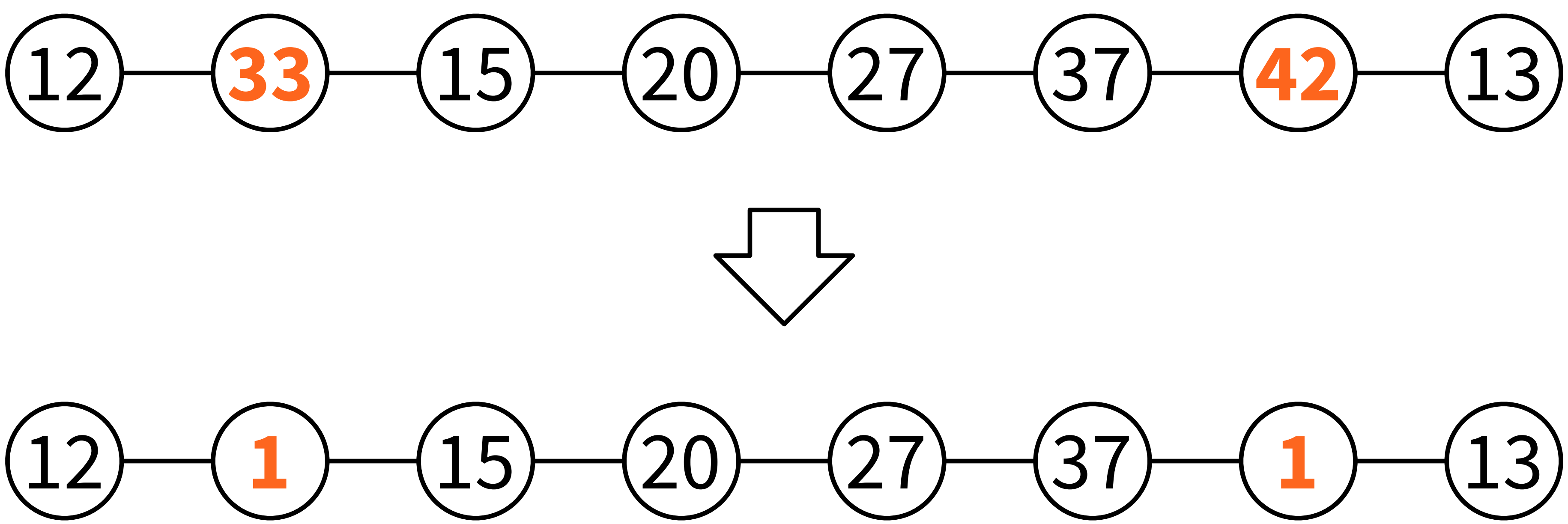
- Take any graph G from graph family F
- Take any port-numbered network N such that G is the underlying graph of N
- If we run A in N, then A stops in time T and outputs a valid solution of problem X

“*A* solves *X* given *Y* on family *F*”

- Take any graph *G* from graph family *F*
- Take any port-numbered network *N* such that *G* is the underlying graph of *N*
- If we run *A* in *N* with any valid input *f* then *A* stops and outputs a valid solution of problem *X*

Algorithm P3C:
3-colouring paths

- **Local maxima pick a new colour from {1,2,3}**



Algorithm P3C:
3-colouring paths

- "**Algorithm P3C solves problem X given Y on graph family F in time $O(|V|)$**"
- X = **3-colouring**
- Y = **colouring** (with any number of colours)
- F = **path graphs**

Algorithm P3C:
3-colouring paths

- **Input = {1, 2, . . .}**
- **States = {1, 2, . . .}**
- **Output = {1, 2, 3}**
- **Msg = {1, 2, . . .}**

Algorithm P3C:
3-colouring paths

- **$\mathrm{init}_0(x) = x$**
- **$\mathrm{init}_1(x) = x$**
- **$\mathrm{init}_2(x) = x$**

Algorithm P3C:
3-colouring paths

- $\mathbf{send}_0(x) = ()$
- $\mathbf{send}_1(x) = (x)$
- $\mathbf{send}_2(x) = (x, x)$

Algorithm P3C:
3-colouring paths

- $\mathbf{receive}_0(\boldsymbol{x}) = \mathbf{1}$ if $x \notin$ Output
- $\mathbf{receive}_0(\boldsymbol{x}) = \boldsymbol{x}$ otherwise

Algorithm P3C:
3-colouring paths

- **$\mathbf{receive}_1(x, y)$ = $\min(\{1, 2\} \setminus \{y\})$**
  if $x \notin$ Output and $x > y$
- **$\mathbf{receive}_1(x, y) = x$** otherwise

Algorithm P3C: **3-colouring paths**

- $\mathbf{receive}_2(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \mathbf{min}(\{1, 2, 3\} \setminus \{\boldsymbol{y}, \boldsymbol{z}\})$
  if $x \notin$ Output and $x > y$ and $x > z$

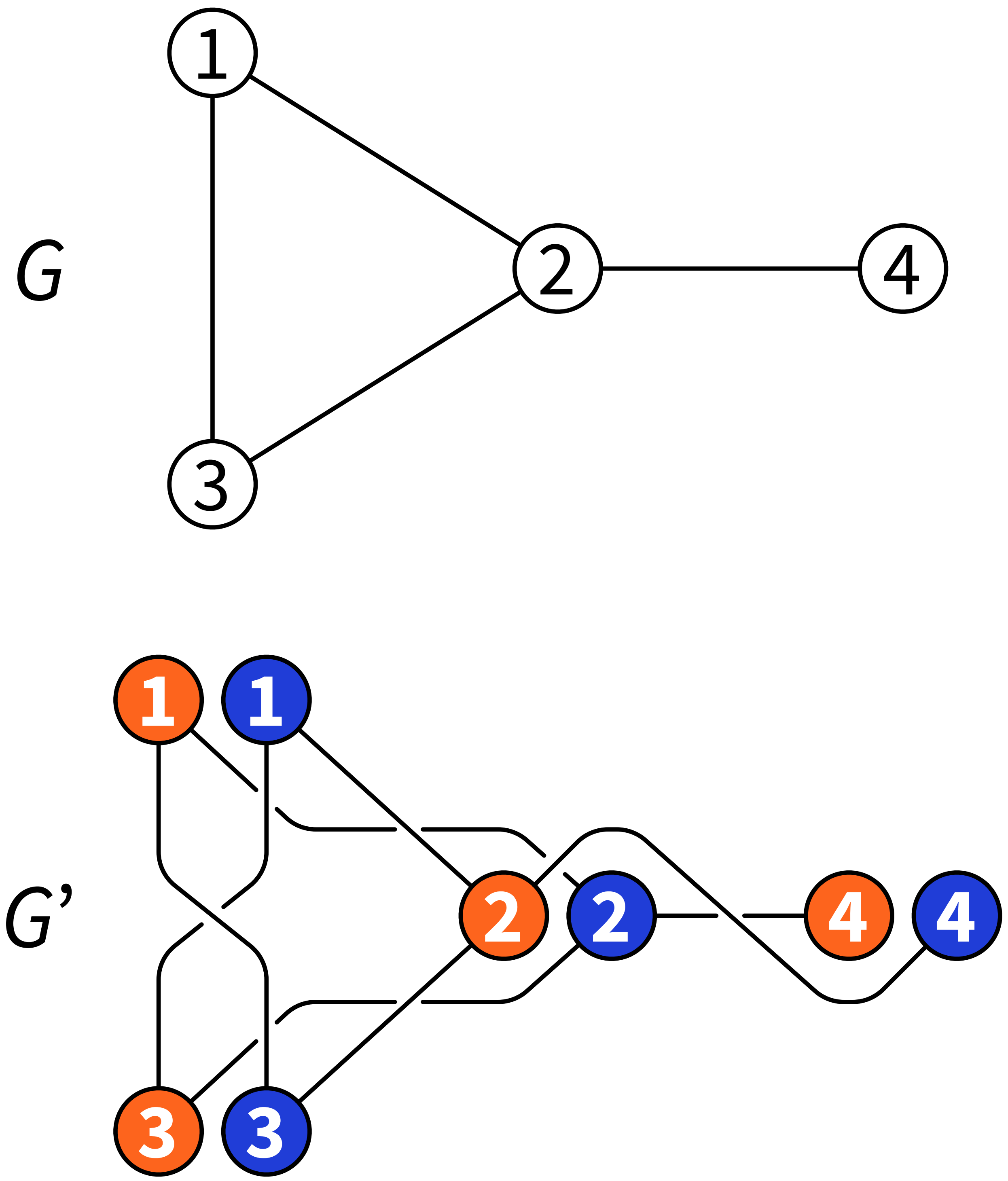
- $\mathbf{receive}_2(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \boldsymbol{x}$ otherwise

Key question

- **What can be solved in PN model without any additional input?**
  - no colouring, unique identifiers, etc.
  - no randomness
- **Example: 3-approximation of minimum vertex cover**

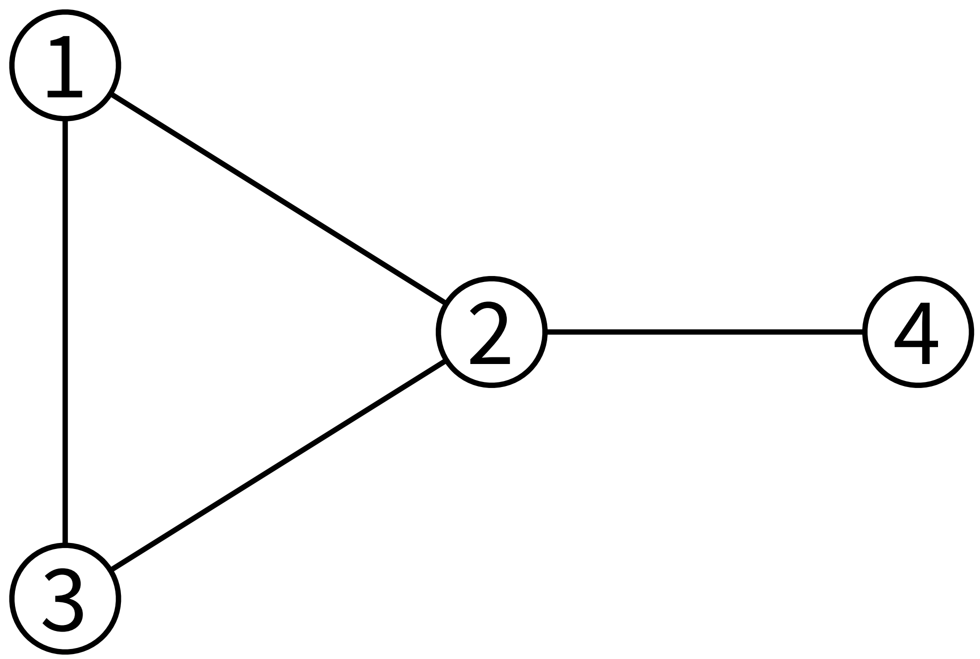
Algorithm VC3:
Small vertex covers

- **Original graph *G*: without any colouring**
- **Virtual graph *G'*: 2-coloured**
- **Find a maximal matching *M'* in *G'***
- **Use *M'* to find a 3-approximation of a minimum vertex cover in *G***



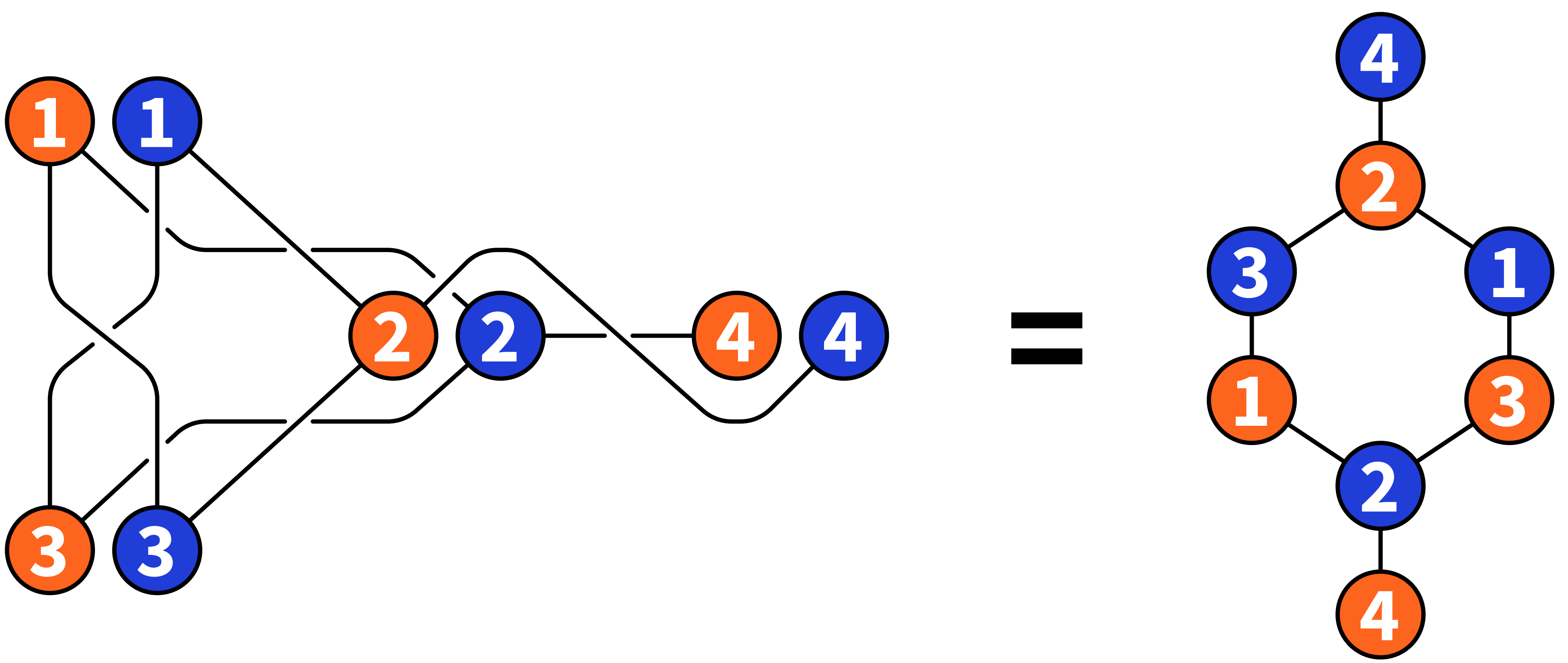
Construct virtual graph *G'*

G

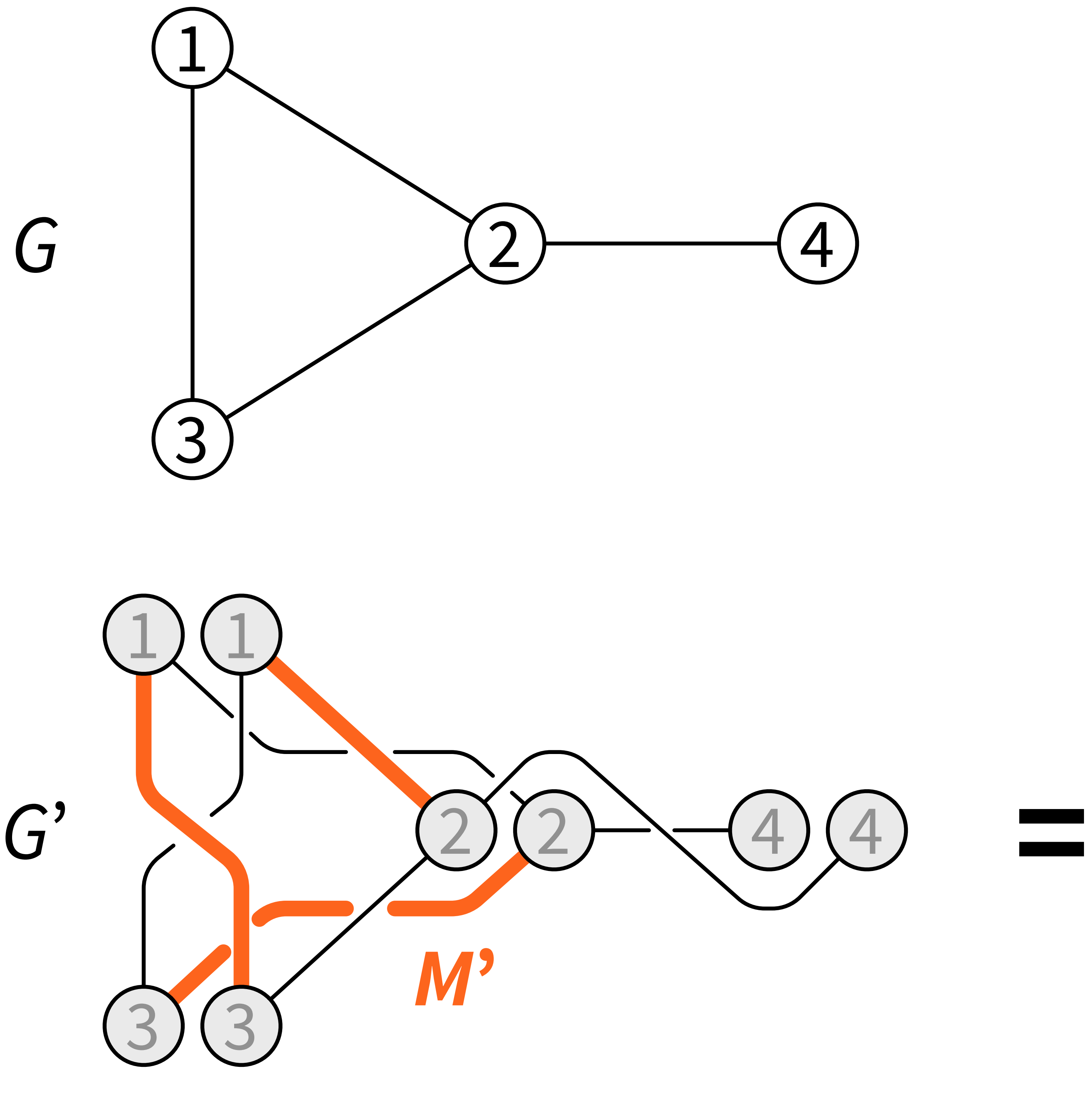


Construct virtual graph *G'*

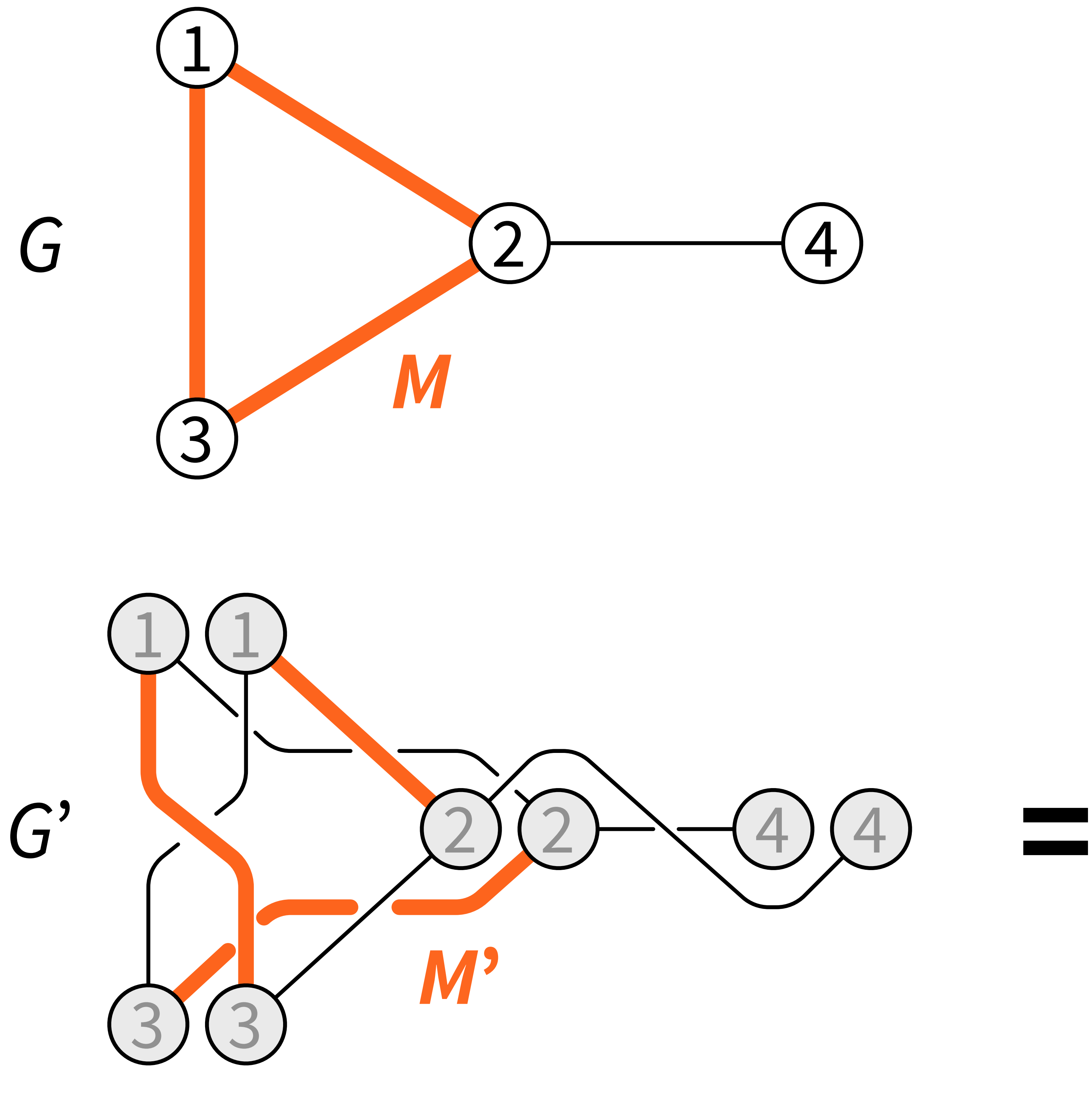
G'

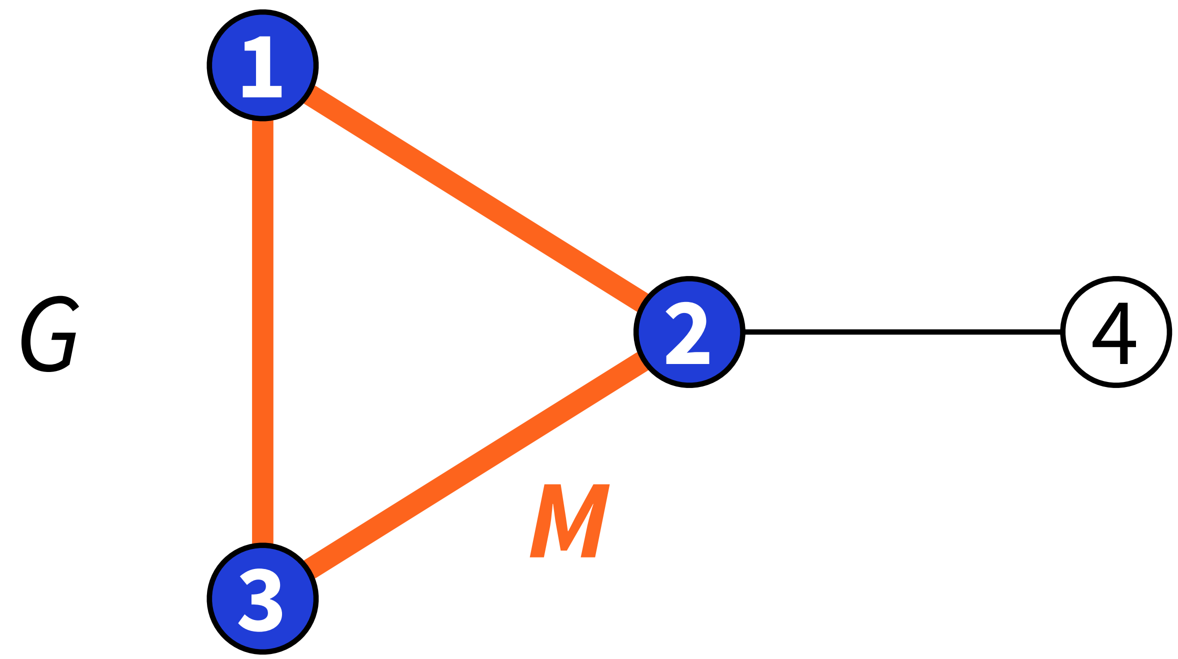


Find maximal matching *M’* in graph *G’*

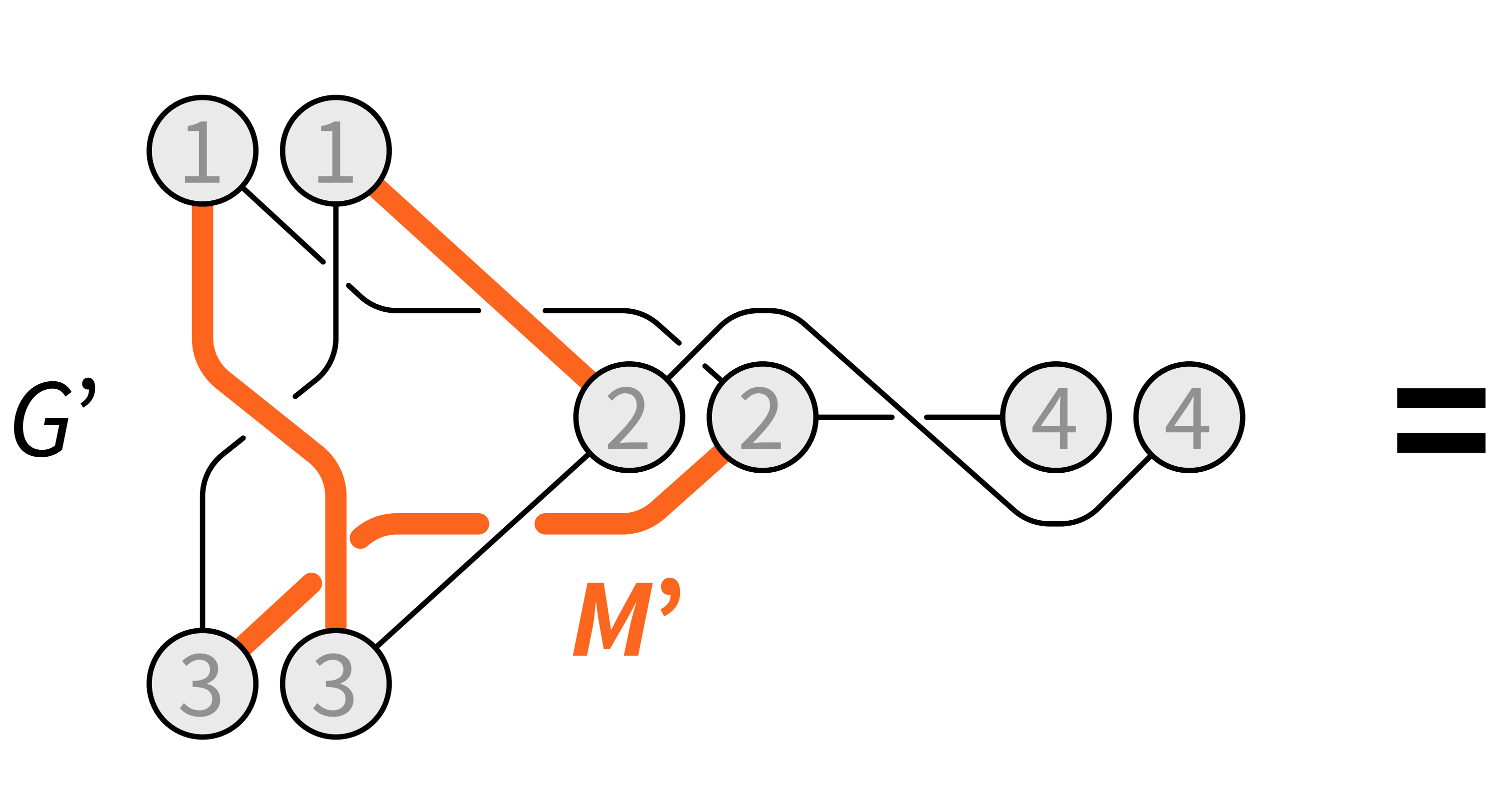


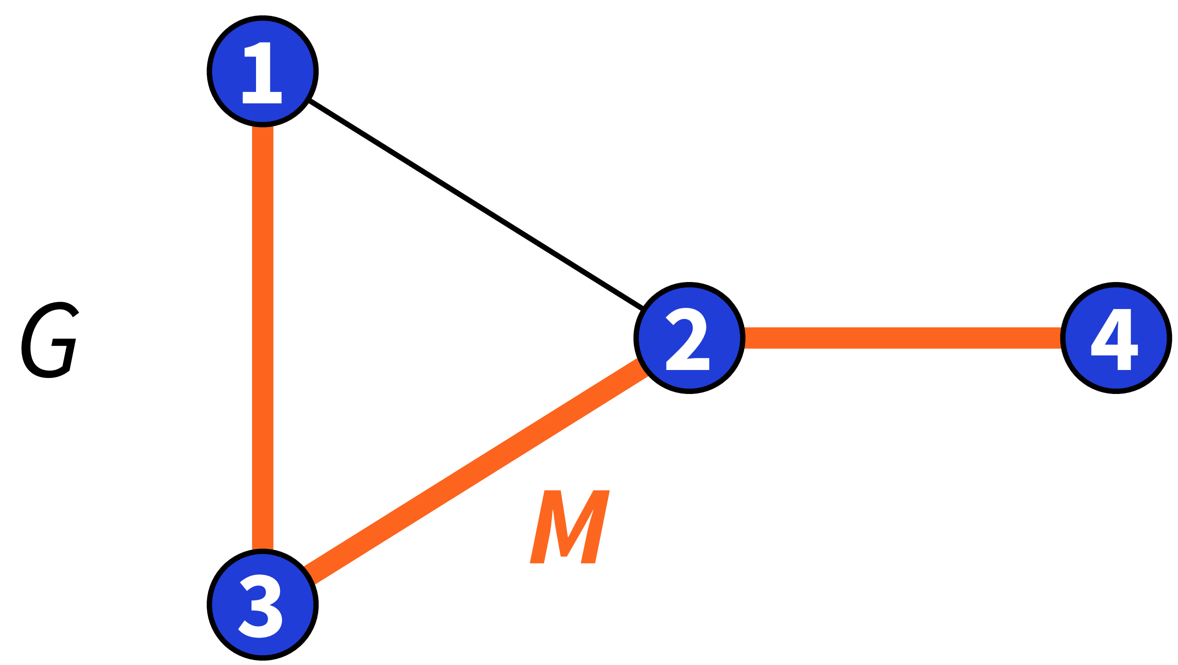
Map back to original graph



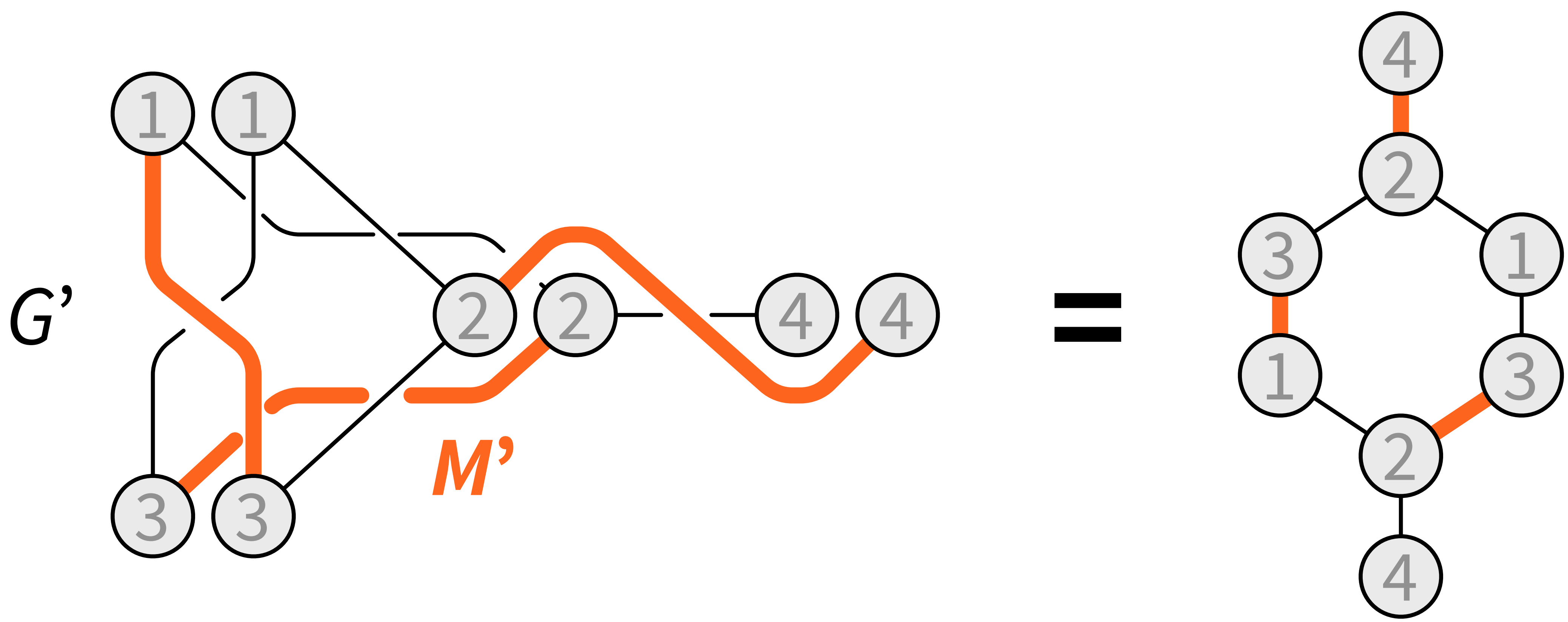


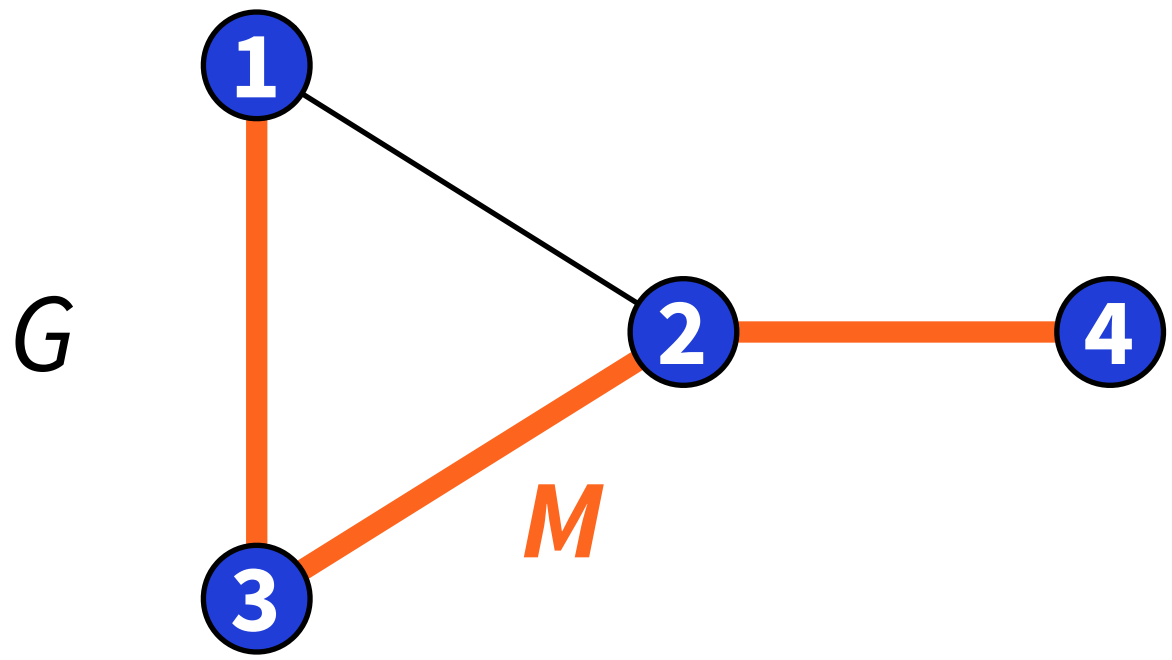
Vertex cover = **all nodes incident to *M***

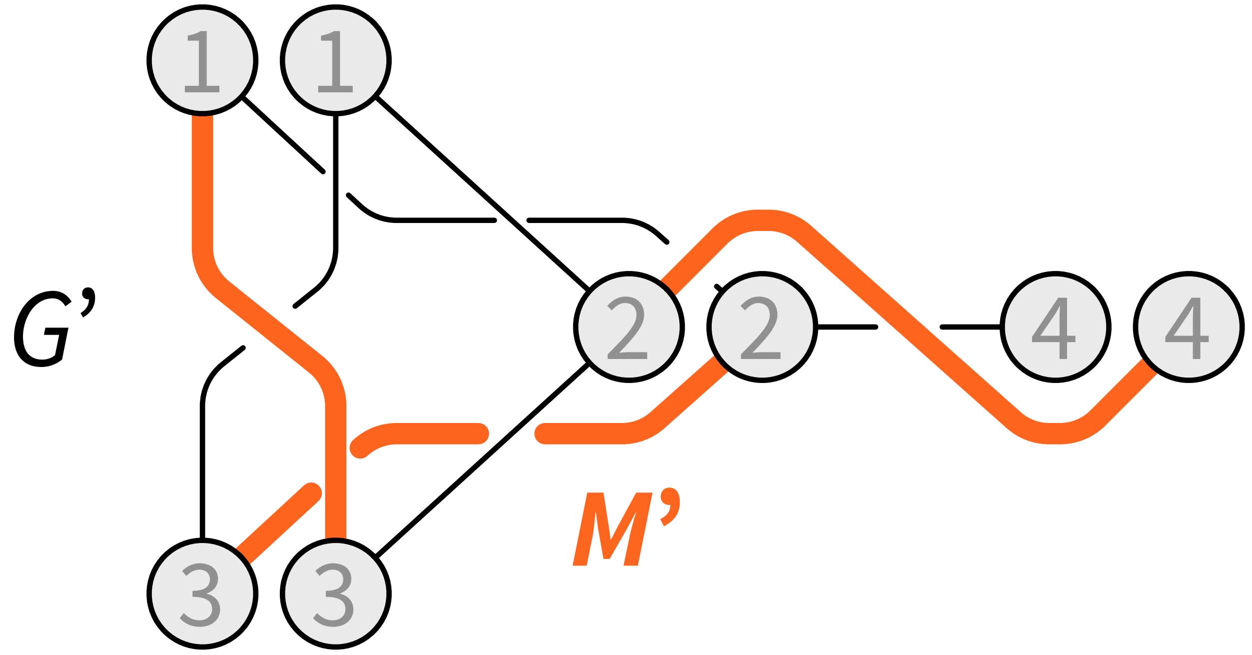




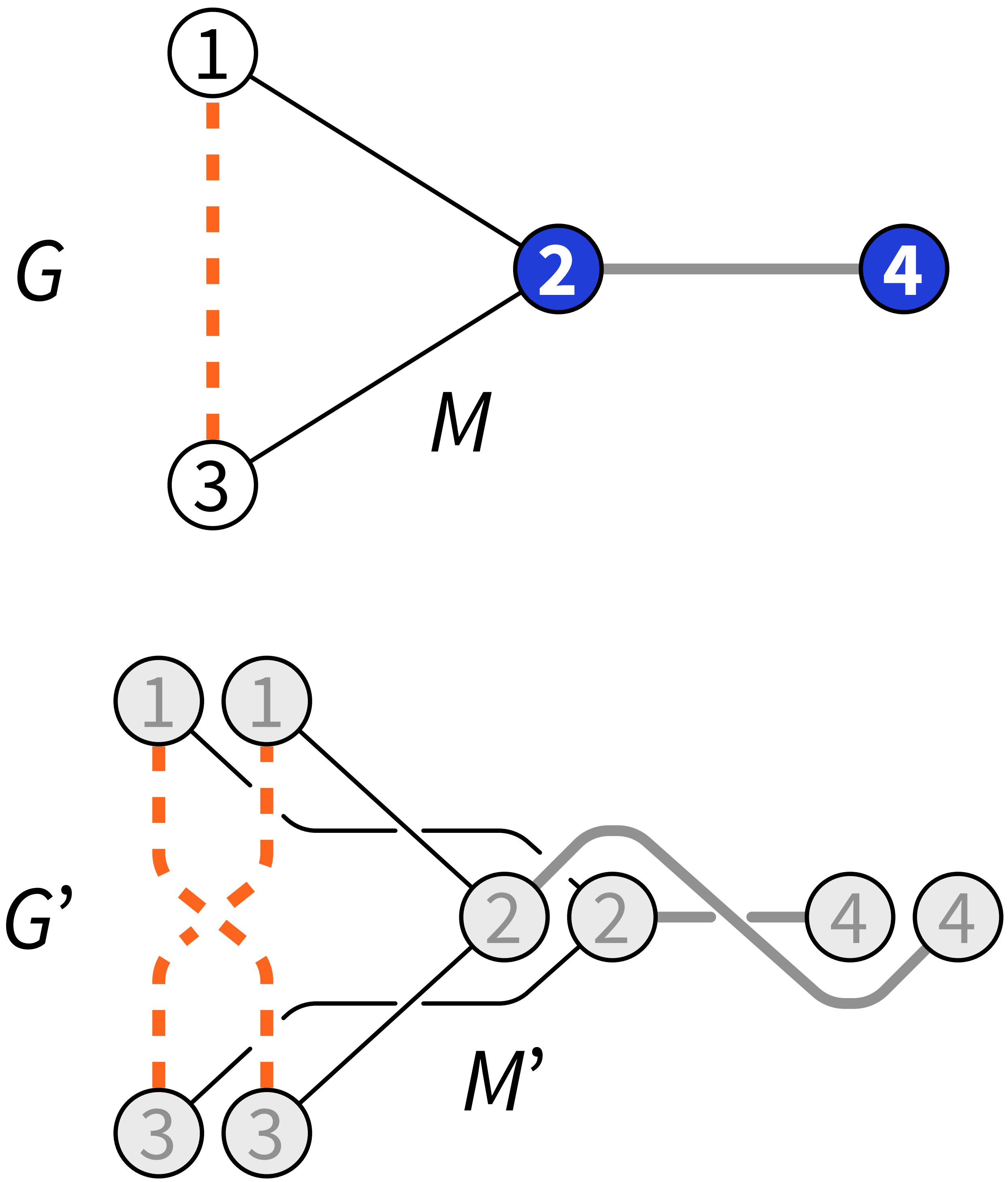
Vertex cover = **all nodes incident to *M***





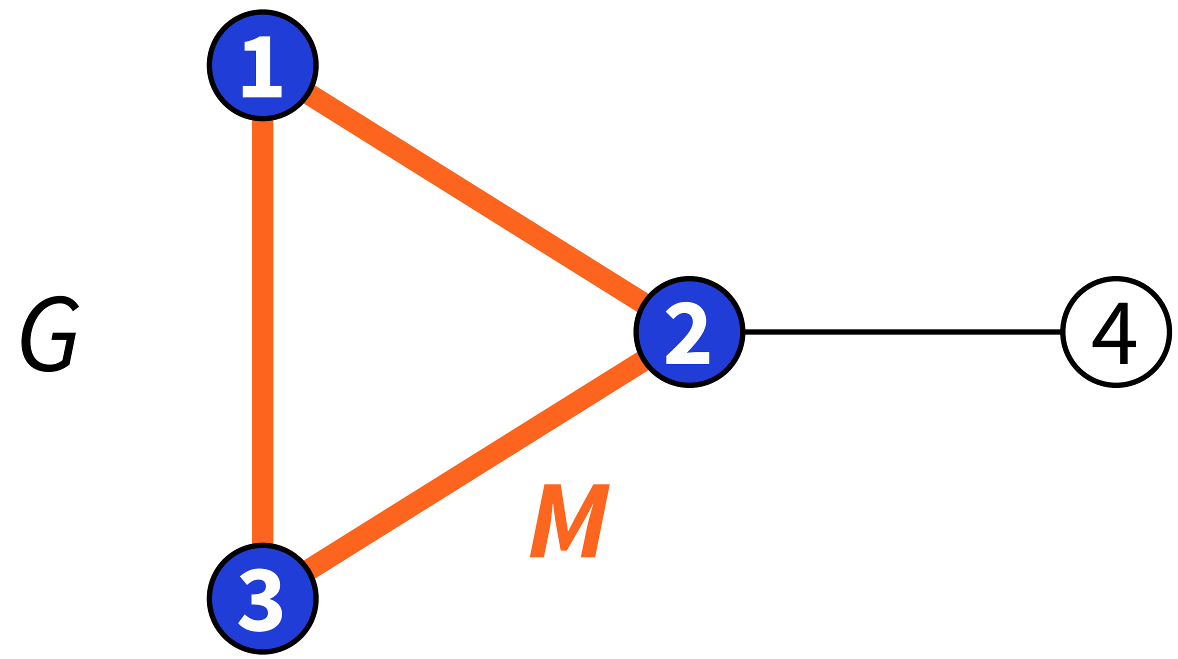


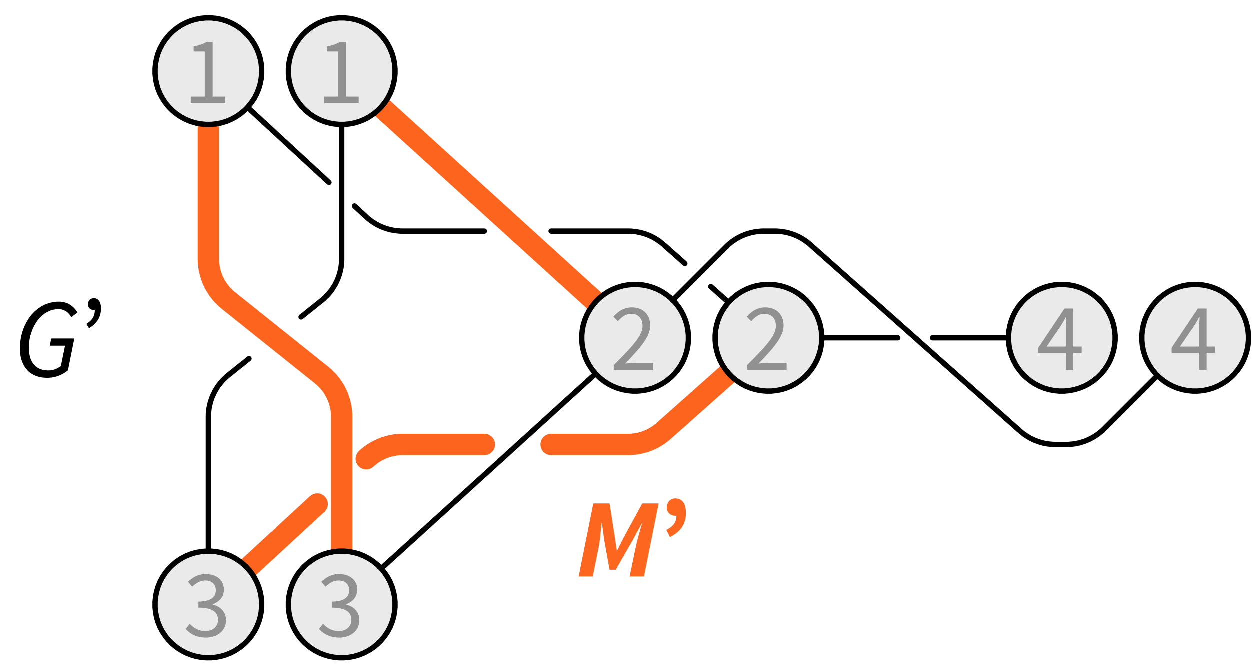
Why vertex cover?



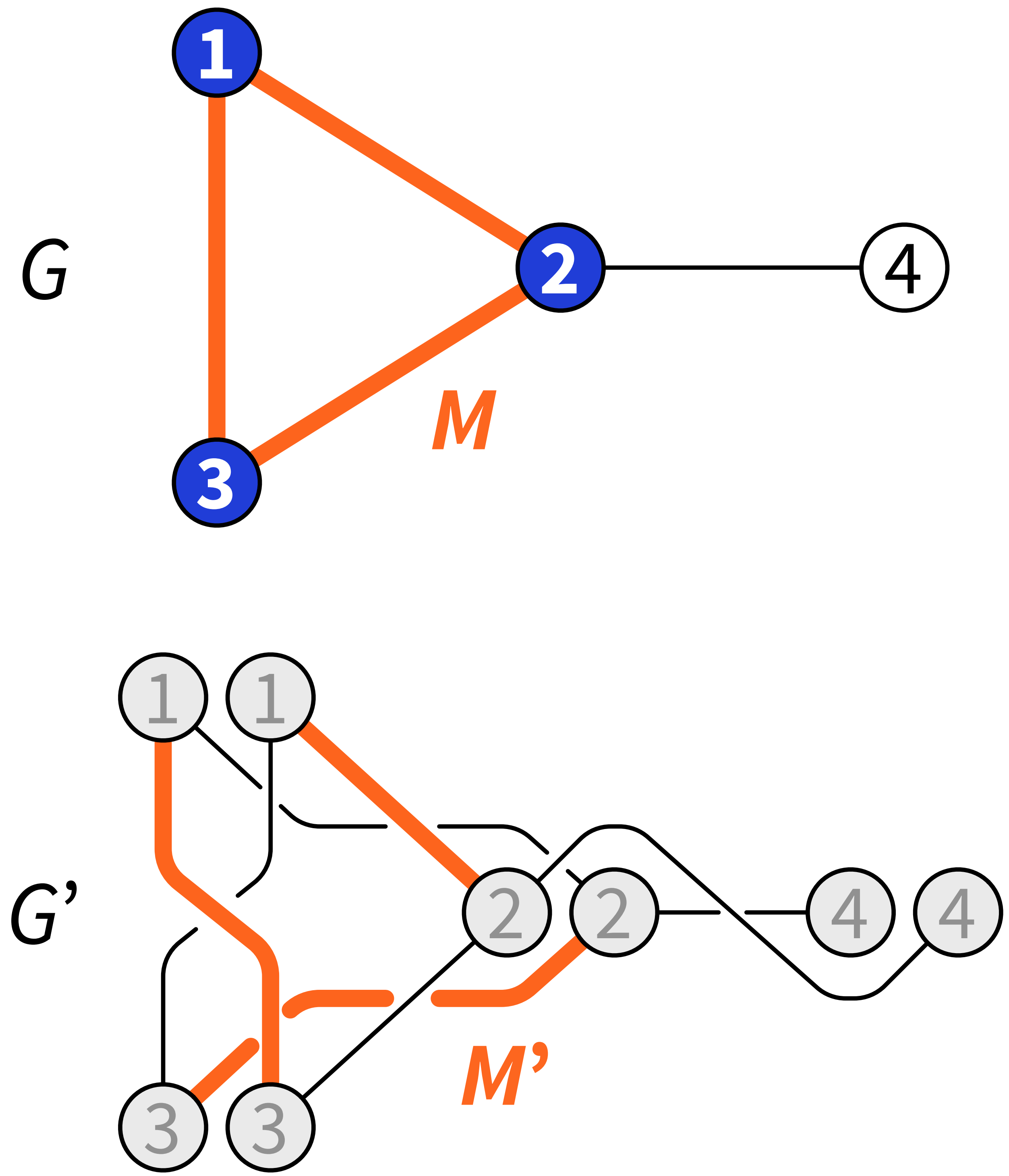
Edge not covered

→ *M'* not maximal

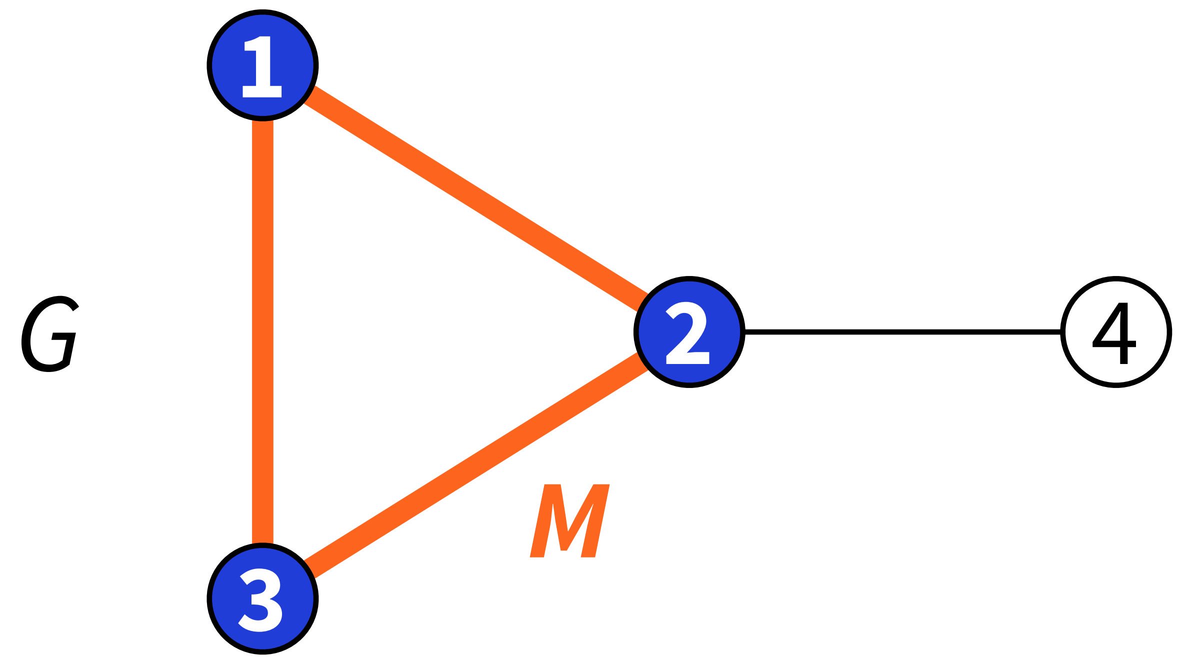




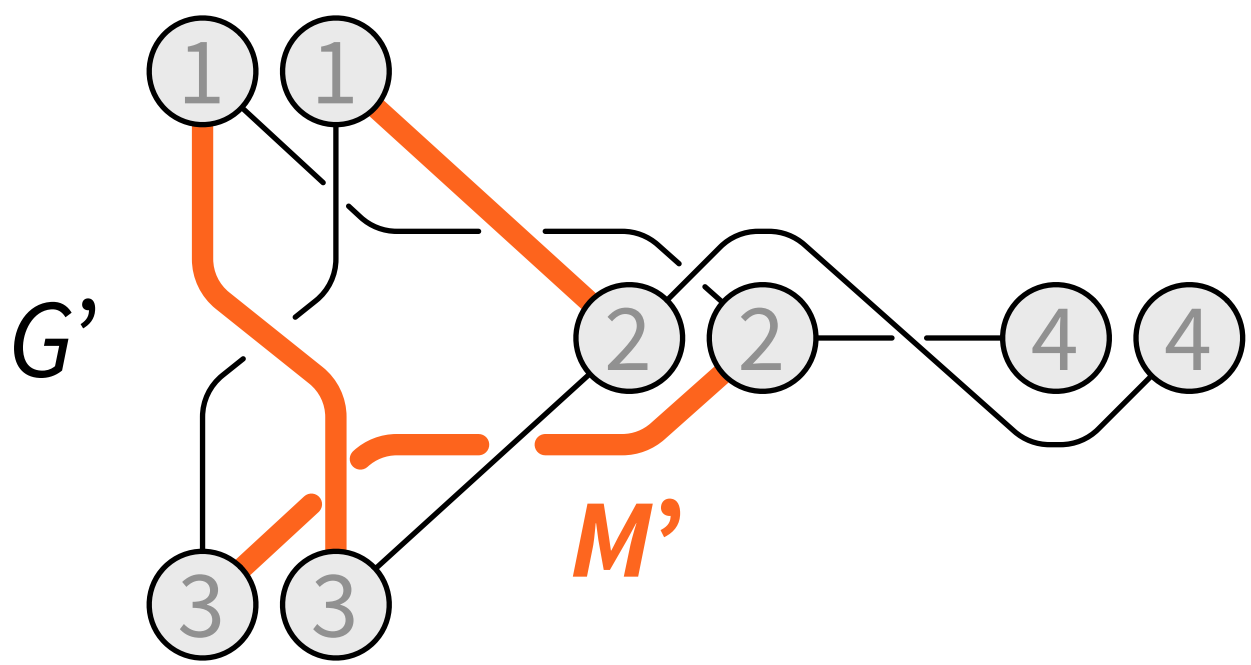
Why within factor 3 of minimum vertex cover?



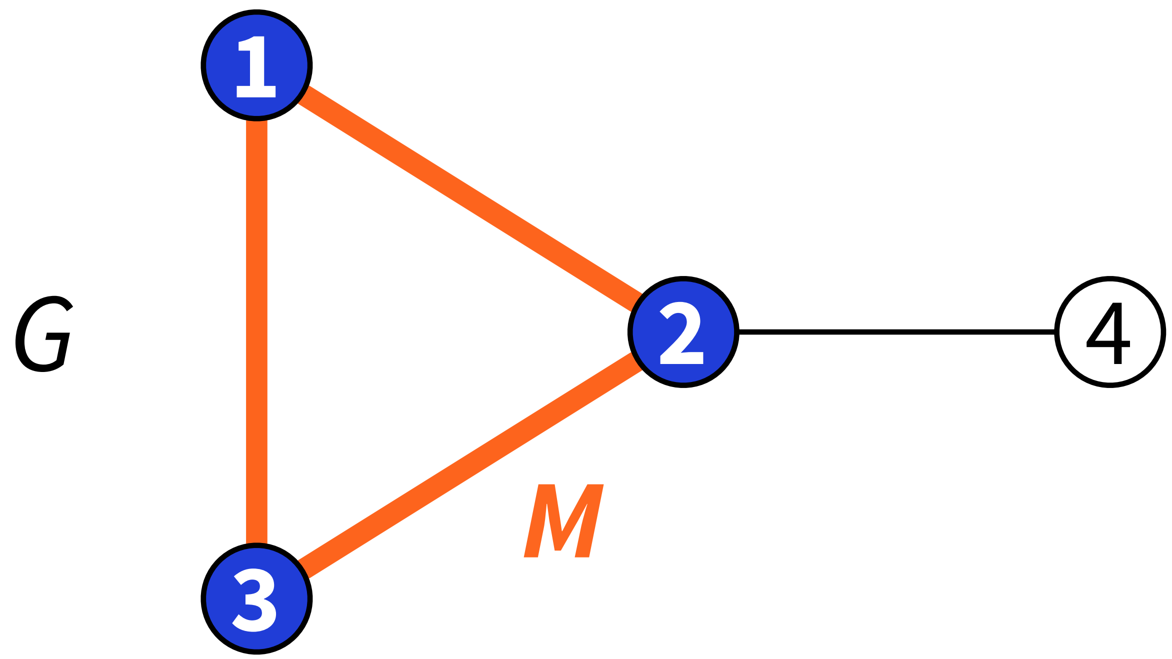
Virtual node: incident to at most 1 edge of *M'*



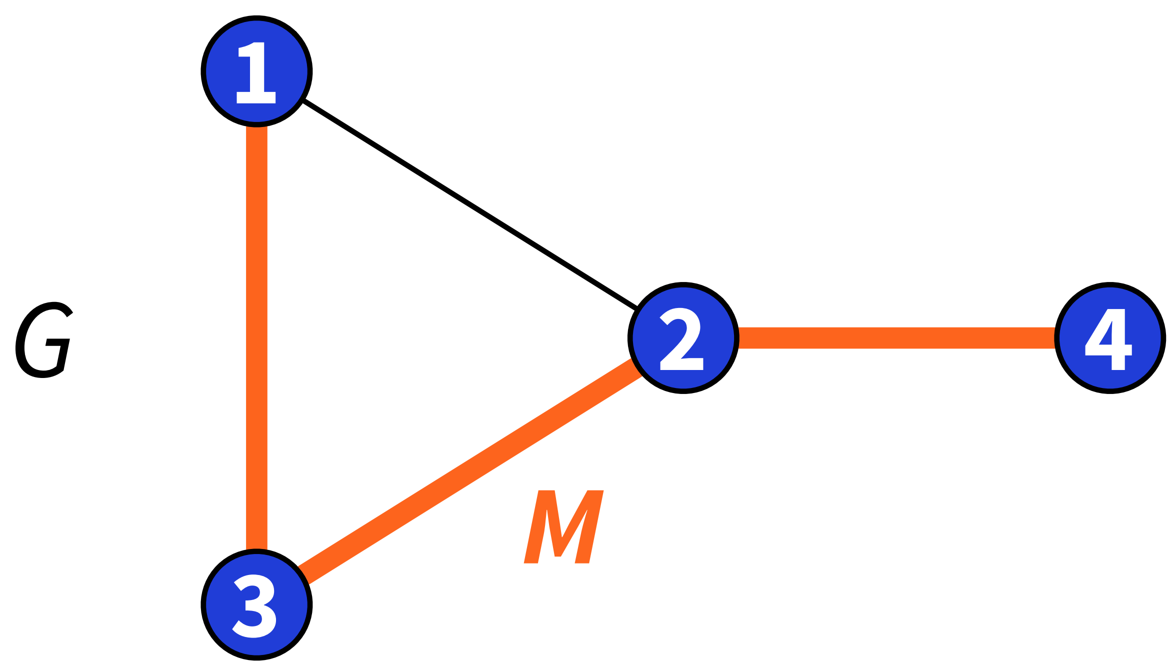
Original node: incident to at most 2 edges of *M*



Virtual node: incident to at most 1 edge of *M’*

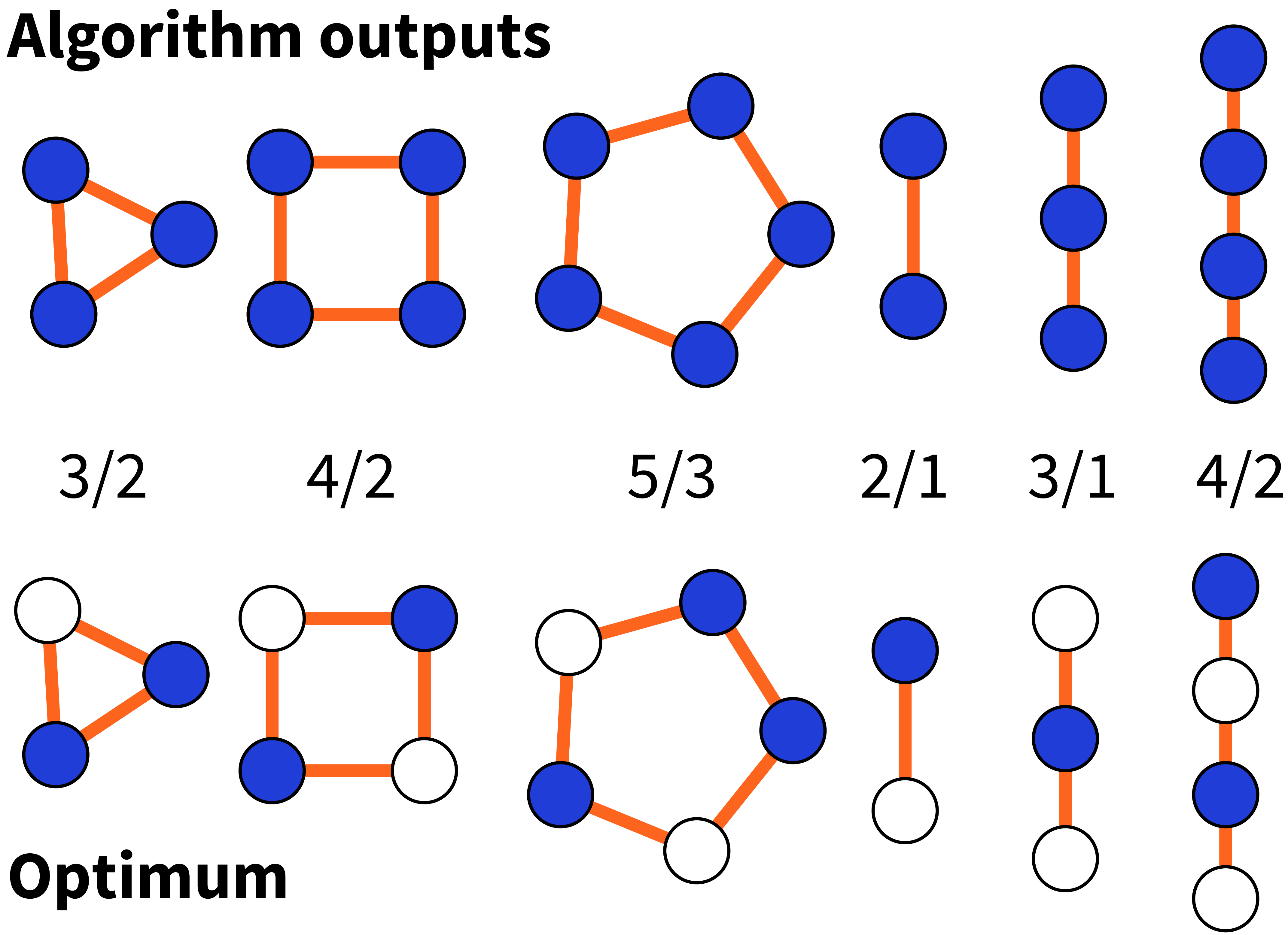


Original node: incident to at most 2 edges of *M*



***M* = paths and/or cycles**

OPT has to cover these!


Algorithm outputs
3/2
4/2
5/3
2/1
3/1
4/2
Optimum

Approximation ratio

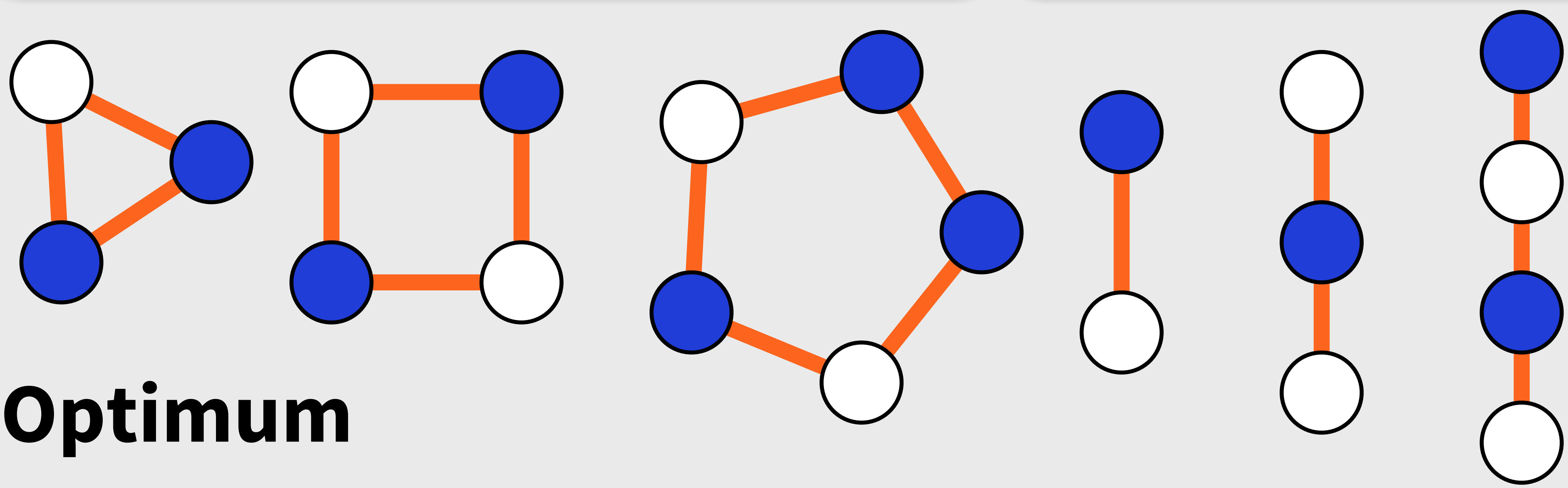
Sum over all cycles & paths of *M*

≤ 2·OPT for cycles

3/2 4/2 5/3

≤ 3·OPT for paths

2/1 3/1 4/2



Optimum

Algorithm VC3:
Small vertex covers

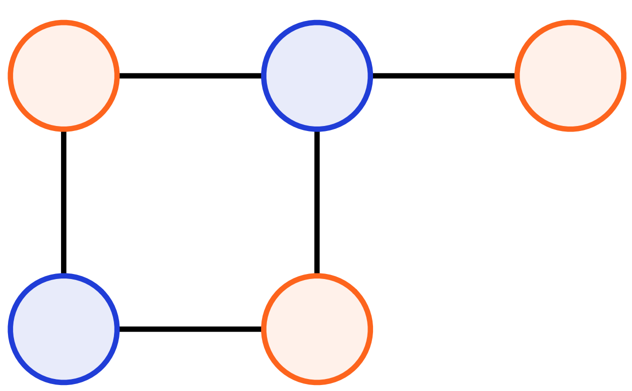
- We can find 3-approximation of a minimum vertex cover in any graph
- … assuming that we can find a maximal matching in 2-coloured graphs!
- Easy to solve: algorithm BMM

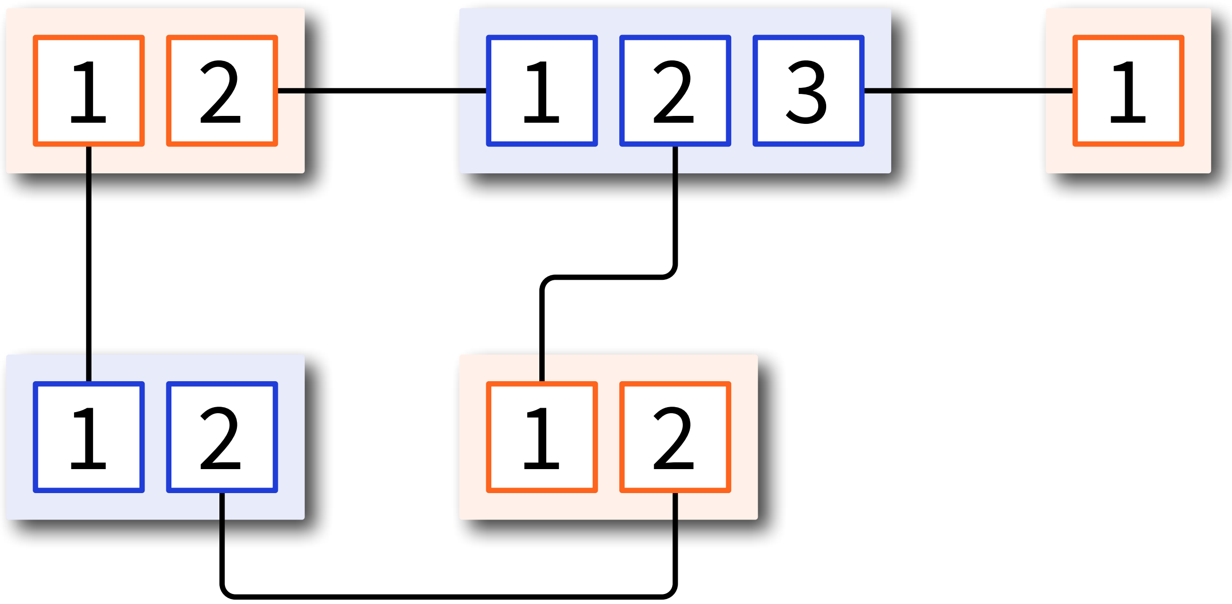
Algorithm BMM: **Maximal matching**

- **Blue nodes send proposals to their orange neighbours one by one**
  - using port numbers
- **Orange nodes accept the first proposal that they get**
  - using port numbers to break ties

Algorithm BMM:
Maximal matching

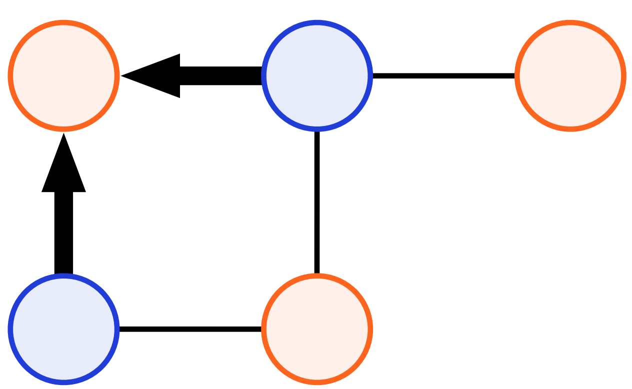
- **Input: 2-coloured graph**

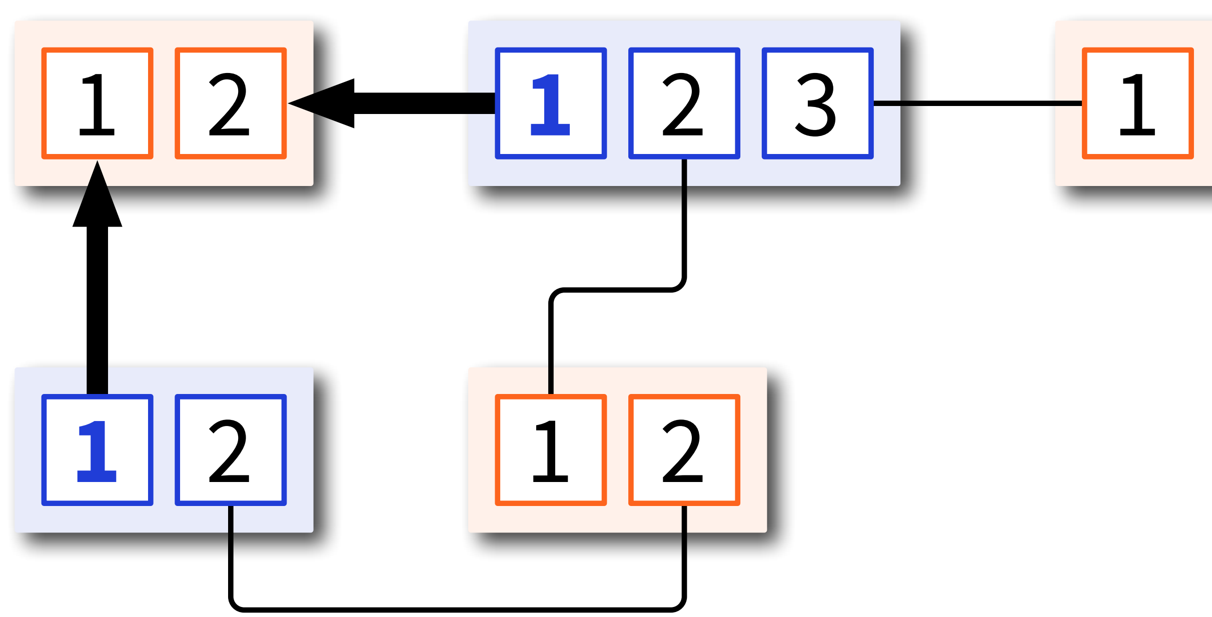




Algorithm BMM:
Maximal matching

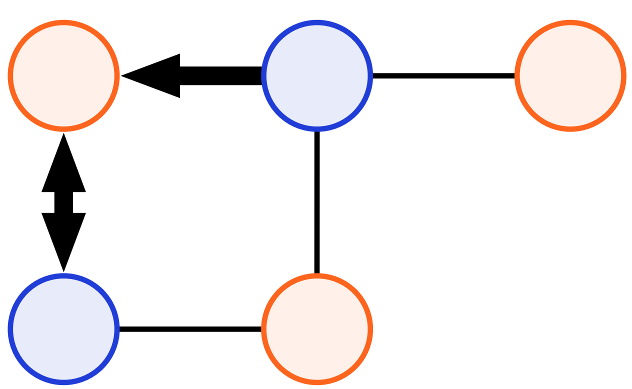
- **Unmatched blue nodes send proposals to port 1**

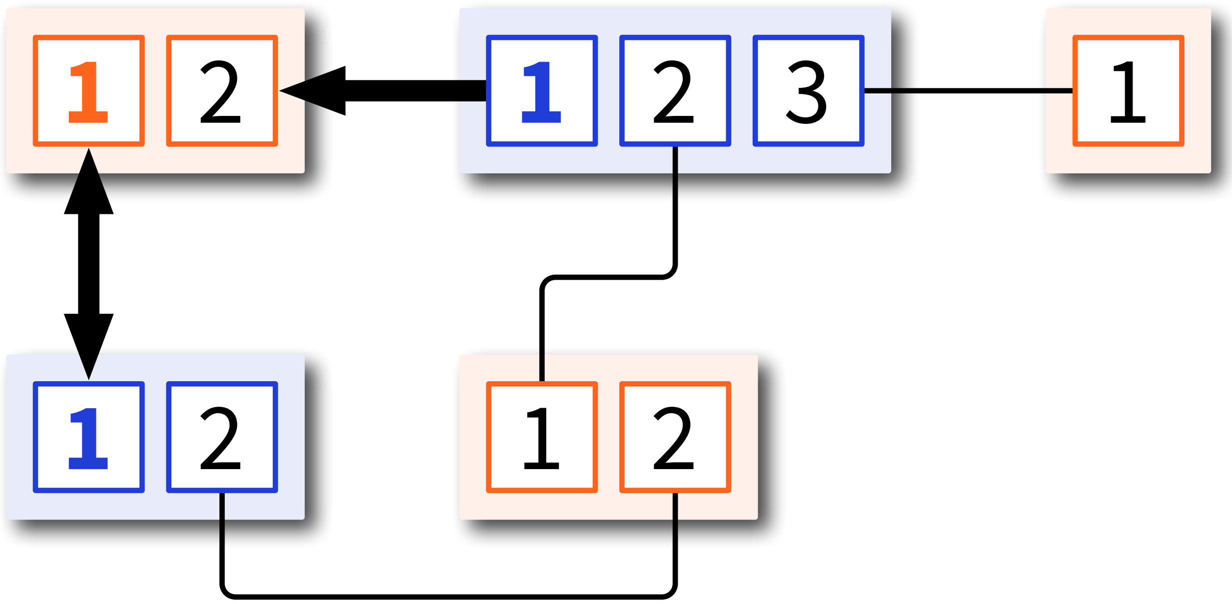




Algorithm BMM:
Maximal matching

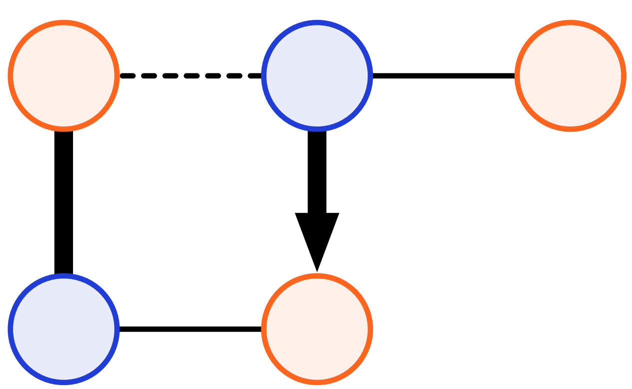
- **Orange nodes accept the first proposal that they get** (giving priority to small ports)

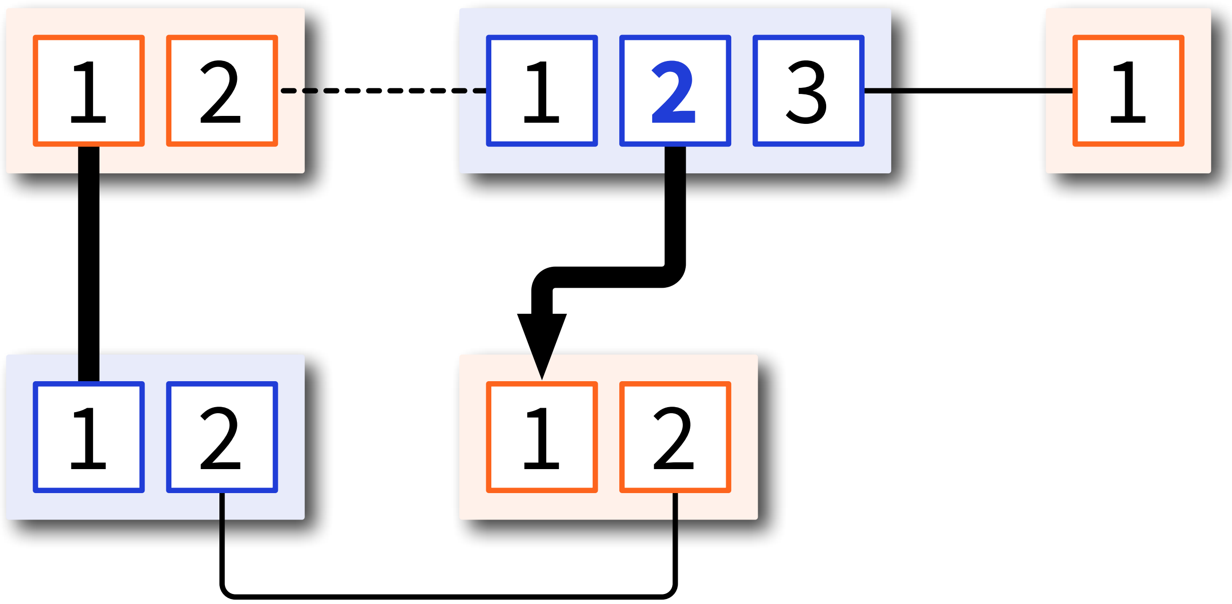




Algorithm BMM: **Maximal matching**

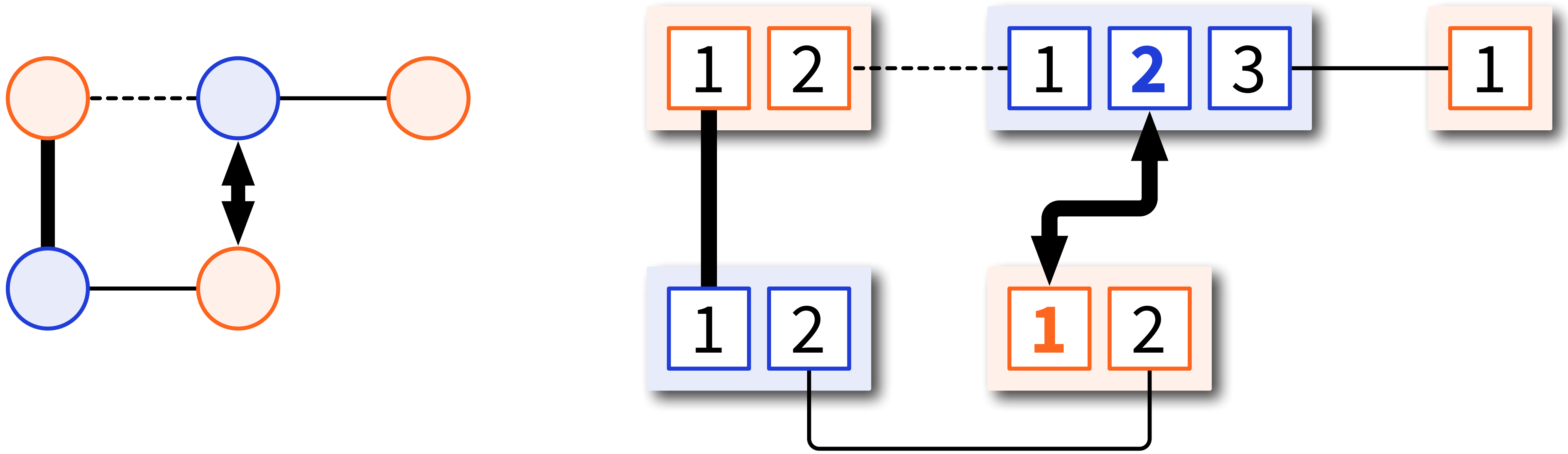
- **Unmatched blue nodes send proposals to port 2**





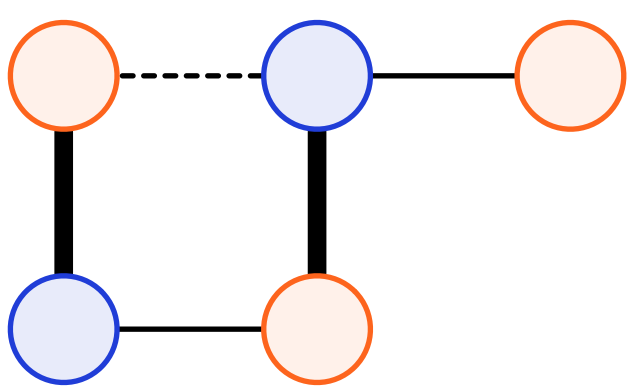
Algorithm BMM:
Maximal matching

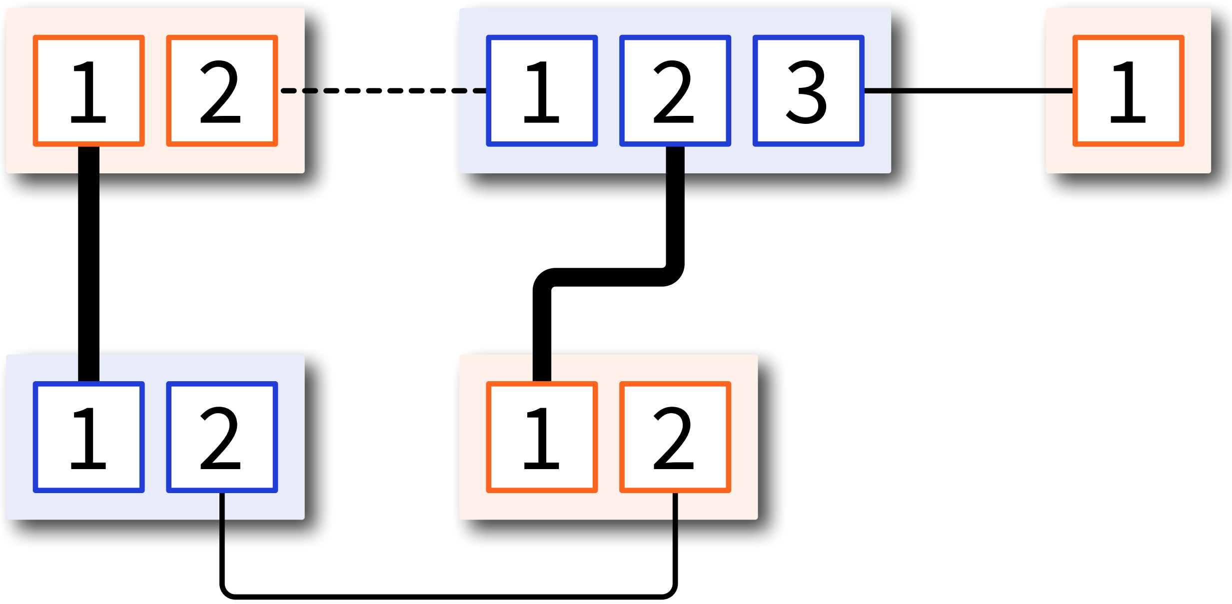
- **Orange nodes accept the first proposal that they get** (giving priority to small ports)



Algorithm BMM:
Maximal matching

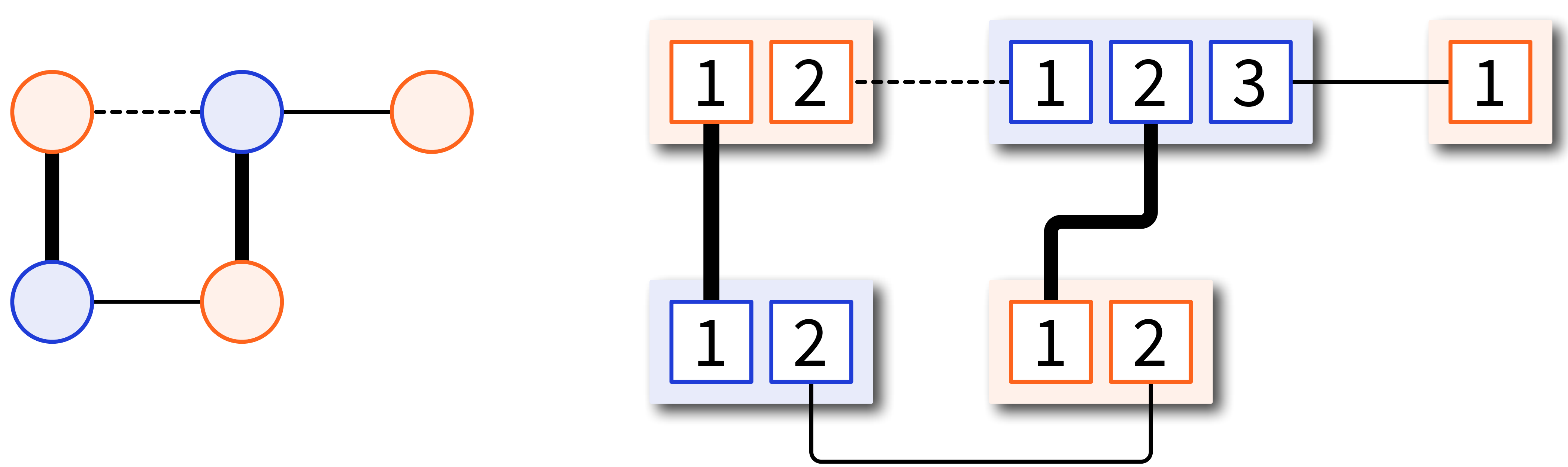
- **Continue until all blue nodes matched or rejected**





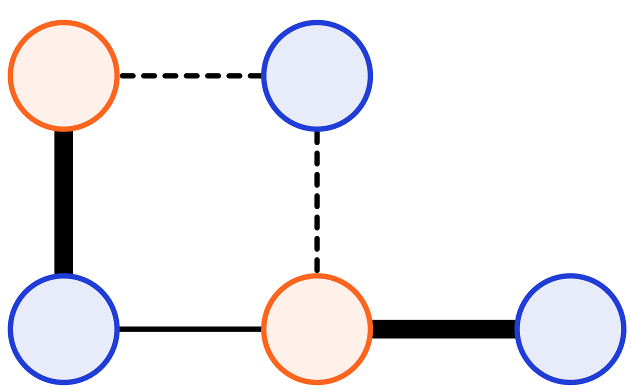
Algorithm BMM:
Maximal matching

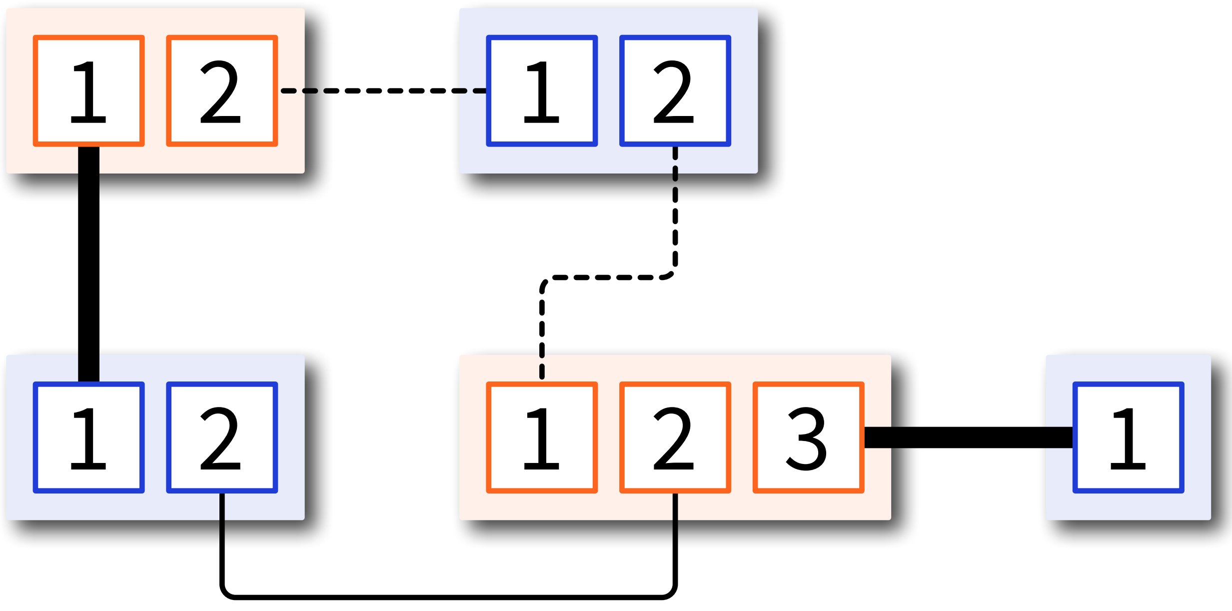
- **All nodes get ≤ 1 partners → matching**



Algorithm BMM:
Maximal matching

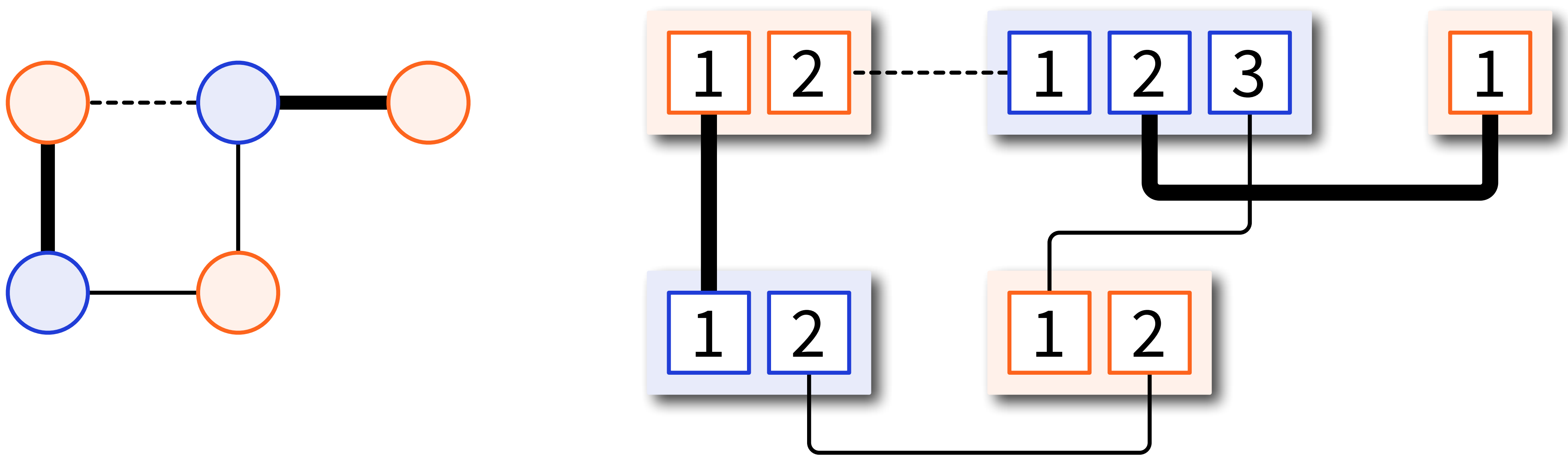
- **Maximality: blue node unmatched only if all orange neighbours reject** (= already matched)





Algorithm BMM:
Maximal matching

- **Maximality: orange node unmatched only if no proposals** (= blue neighbours are matched)

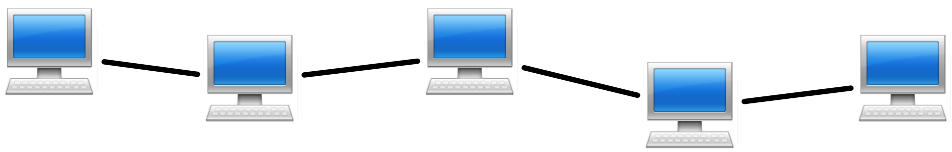


Summary

- **Algorithm BMM:** maximal matching in 2-coloured graphs
- **Algorithm VC3:** 3-approximation of minimum vertex covering in any graph
- **VC3 uses BMM as a subroutine:** virtual 2-coloured graph

Summary

- **There are non-trivial problems that can be solved in the PN model**
  - without unique identifiers, colouring, etc.
- **However, algorithm design much easier if we assume unique IDs**
  - our topic next week

- **Weeks 1–2: informal introduction**
  - network = path 
- **Week 3: graph theory**
- **Weeks 4–7: models of computing**
  - what can be computed (efficiently)?
- **Weeks 8–11: lower bounds**
  - what cannot be computed (efficiently)?
- **Week 12: recap**