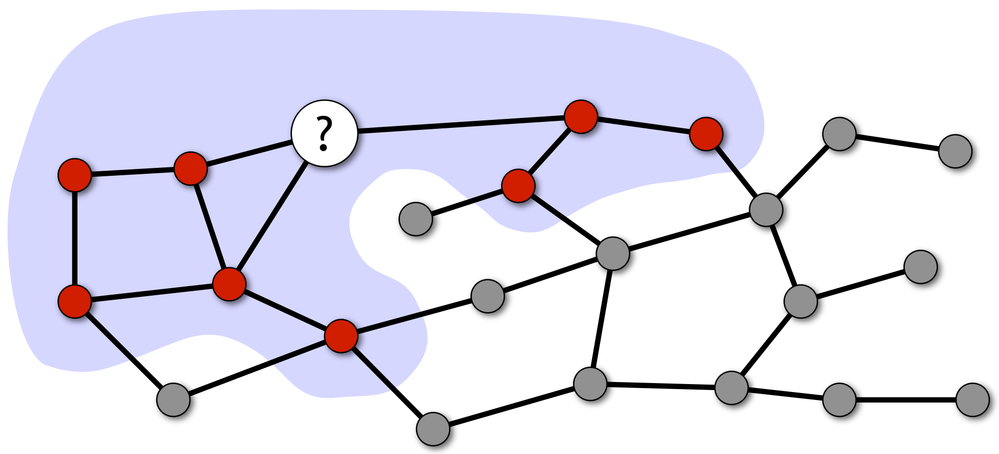
Models of distributed computing: port numbering and local algorithms

Jukka Suomela

Adaptive Computing Group
Helsinki Institute for Information Technology HIIT
University of Helsinki

FMT seminar, 26 February 2010



Our research focus

- Very restrictive models of distributed computing
  - Local algorithms (constant-time distributed algorithms)
  - Algorithms in anonymous networks
  - Deterministic algorithms
- Graph problems
  - Vertex covers, dominating sets, ...
  - Linear programs in graphs
- Approximability

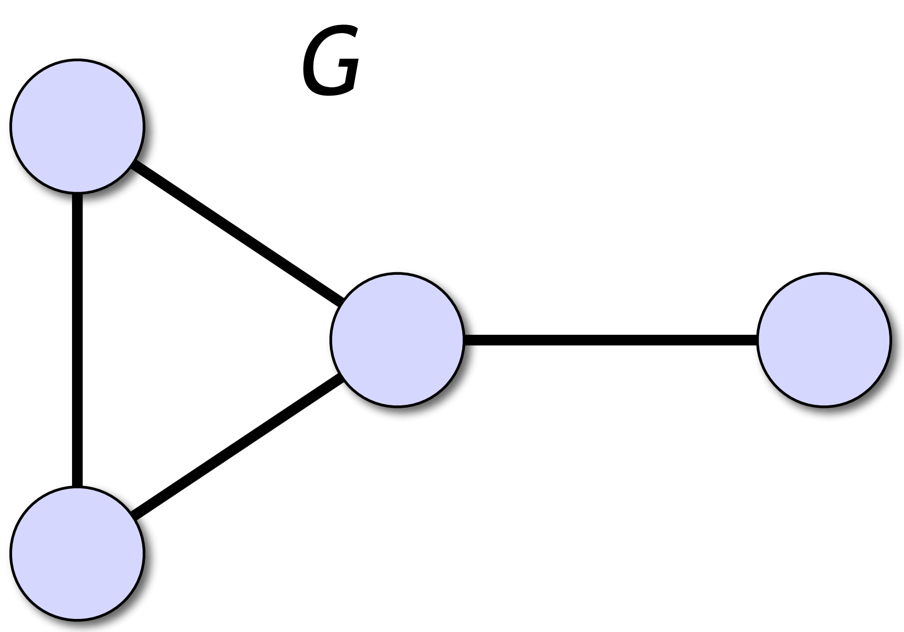
Outline of today's talk

- Models of computation
  - Local algorithms
  - Port-numbering model
- Observations and results
  - What is known about these models?
  - Case study: vertex cover problem
- Connections to other models of computation
  - Constant-depth bounded-fan-in circuits, NC^0

Part I: Models of computation

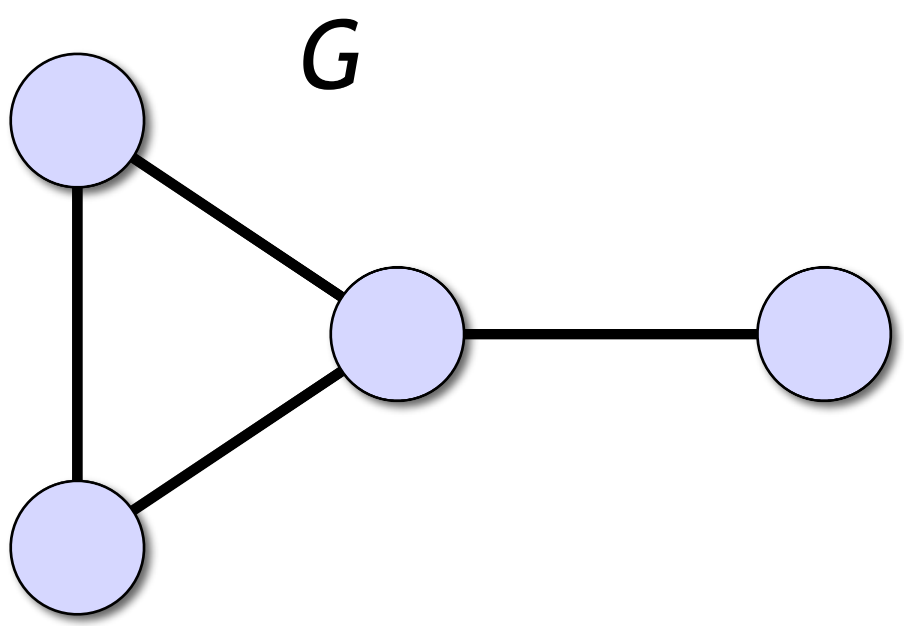
- Distributed algorithms in general
- Two very limited special cases:
  - Local algorithms
  - Port-numbering model

Distributed algorithms



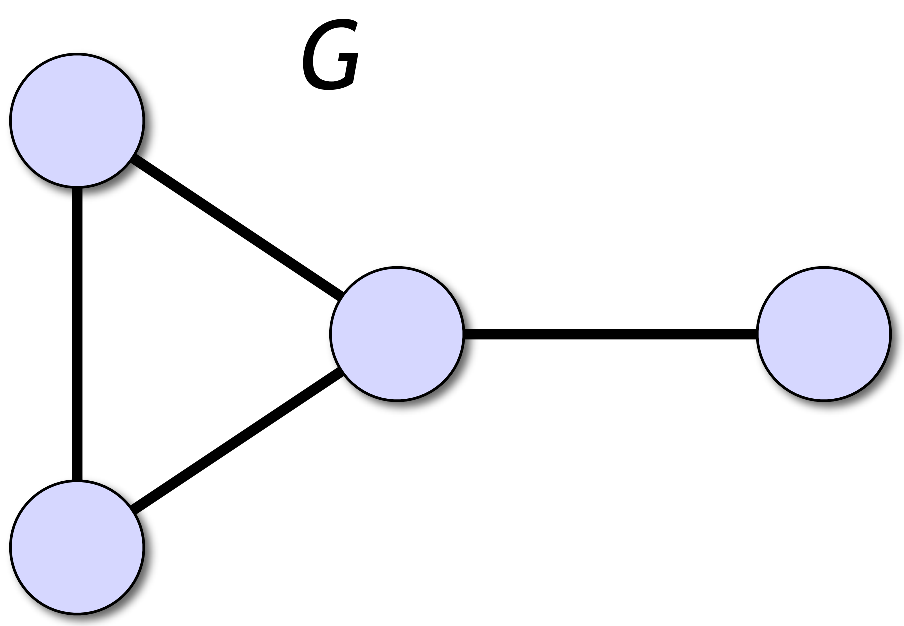
- Communication graph *G*
- Node = computer
  - e.g., Turing machine, finite state machine
- Edge = communication link
  - computers can exchange messages

Distributed algorithms



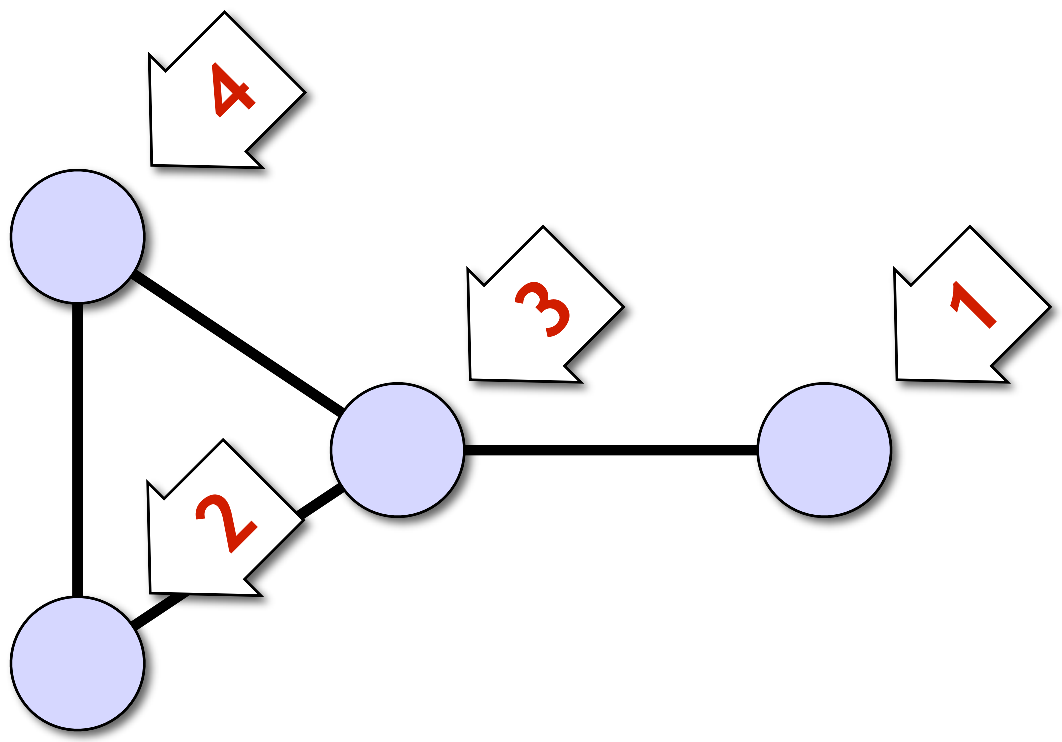
- All nodes are identical, run the same algorithm
- ***We*** can choose the algorithm
- An ***adversary*** chooses the structure of G
- Our algorithm must produce a correct output in any graph G

Distributed algorithms



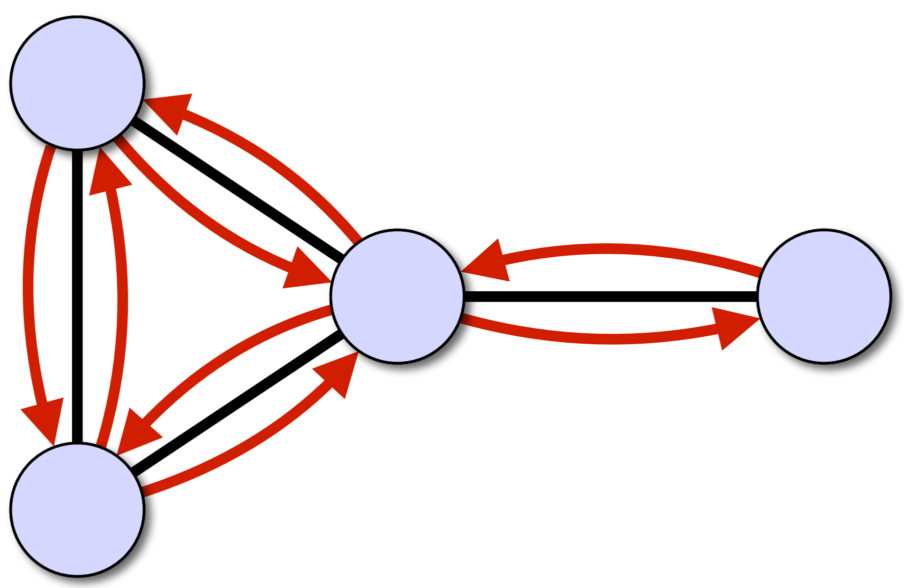
- Usually, computational problems are related to the structure of the communication graph G
  - Example: find a maximal independent set for G
  - The same graph is both the input and the system that tries to solve the problem...

Synchronous distributed algorithms



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

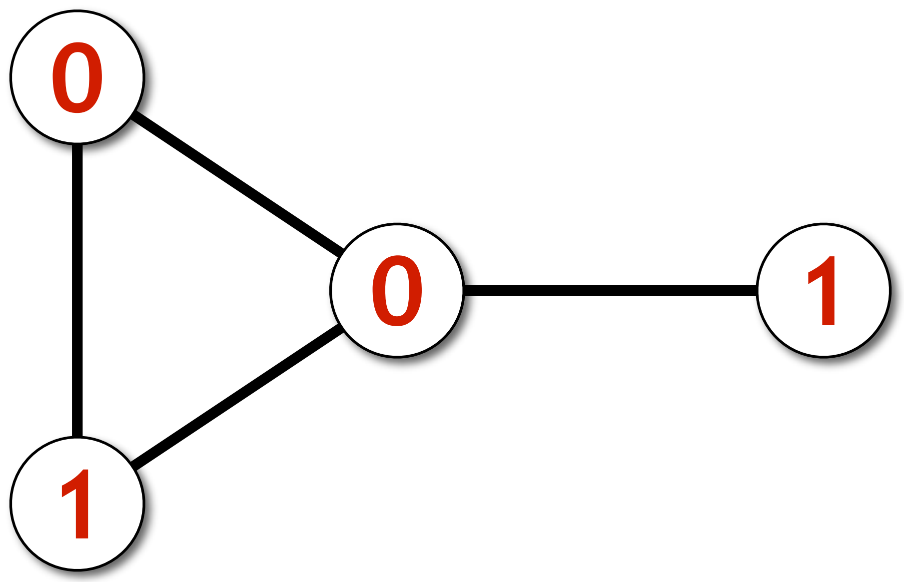
Synchronous distributed algorithms



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

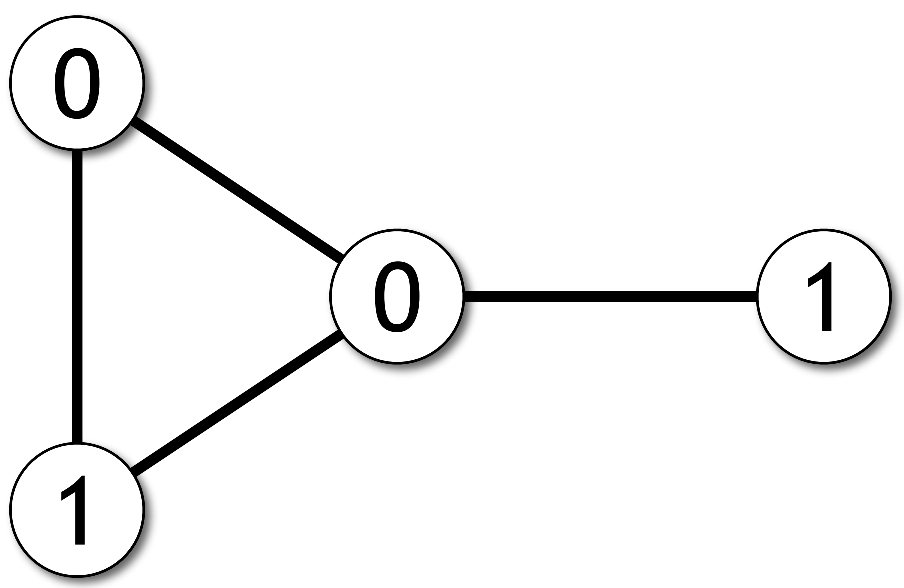
   ...

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

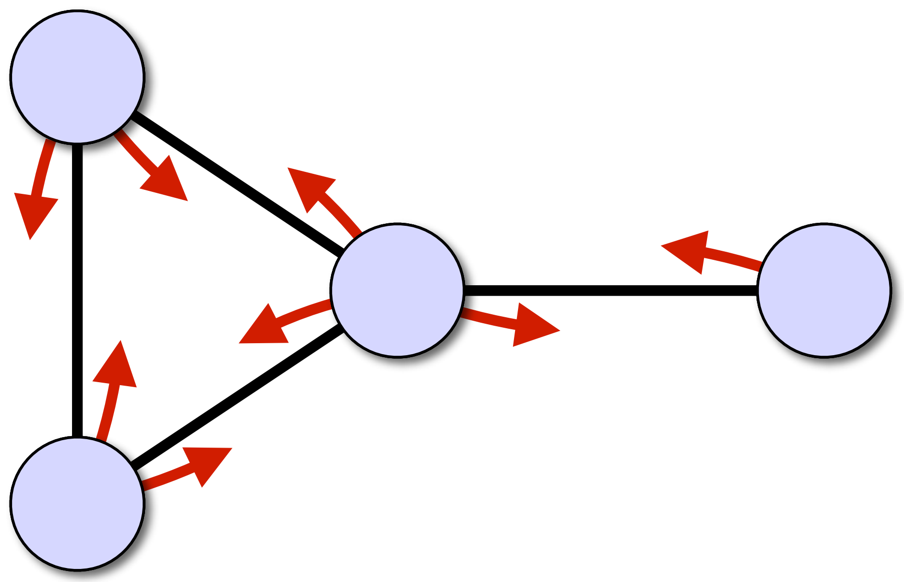


1. Each node reads its own **local input**
2. Repeat synchronous **communication rounds** until all nodes have announced their **local outputs**

Example: Find a maximal independent set I

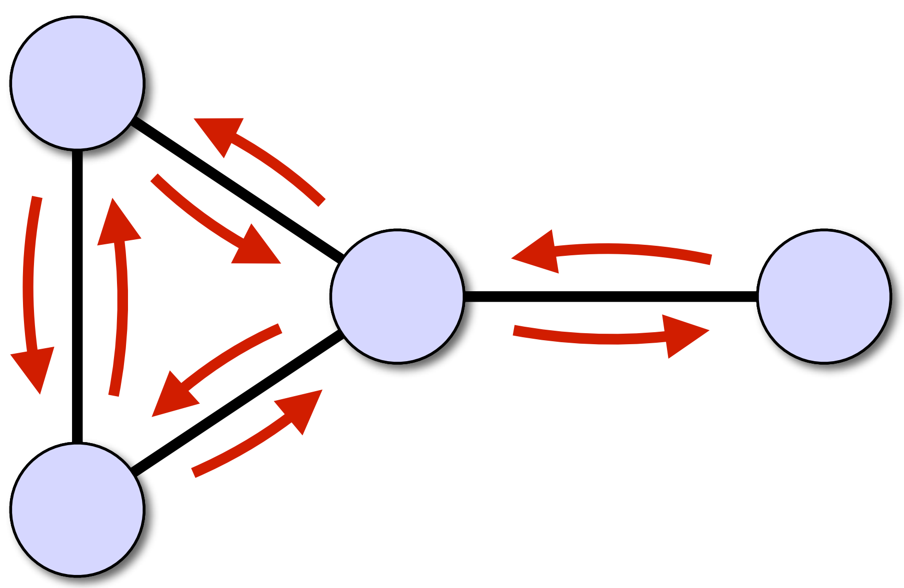
Local output of a node v indicates whether $v \in I$

Synchronous distributed algorithms



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

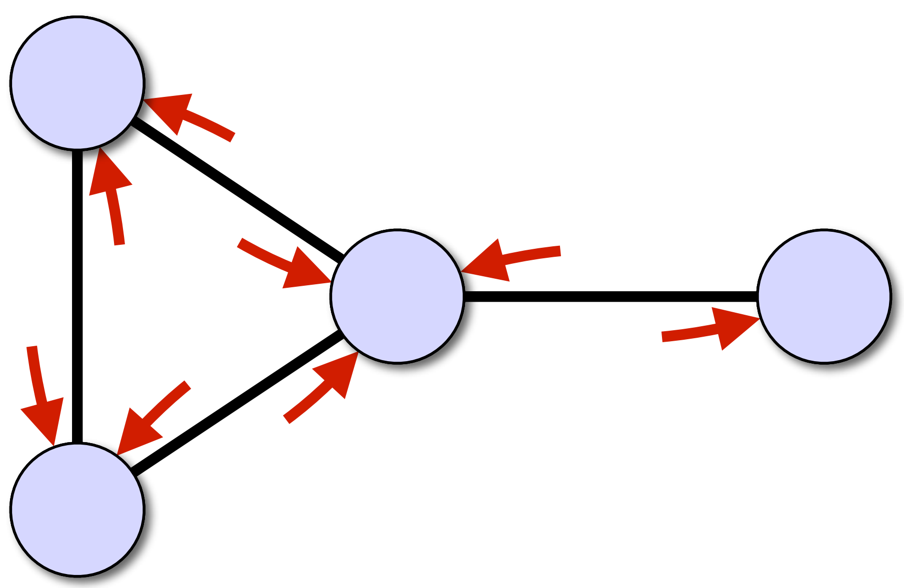
Synchronous distributed algorithms



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

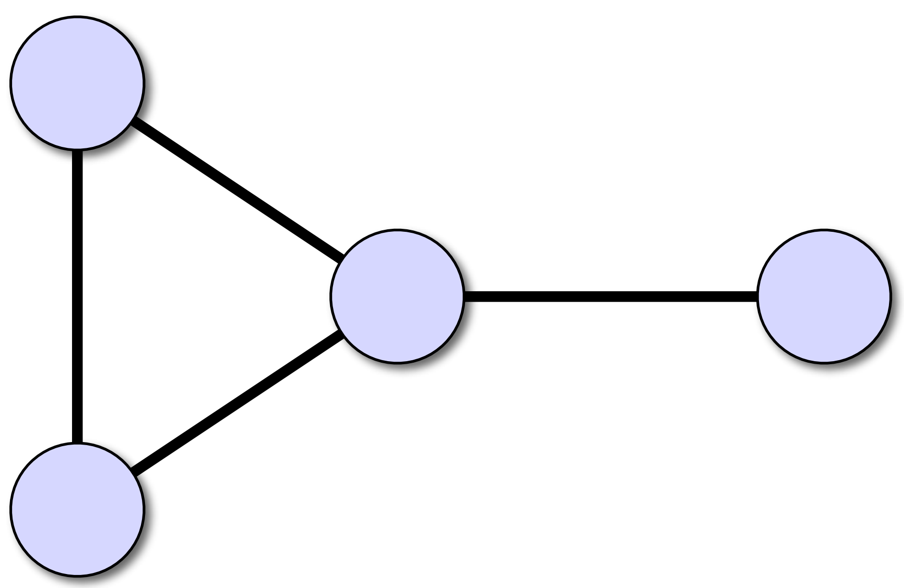
  (message propagation...)

Synchronous distributed algorithms



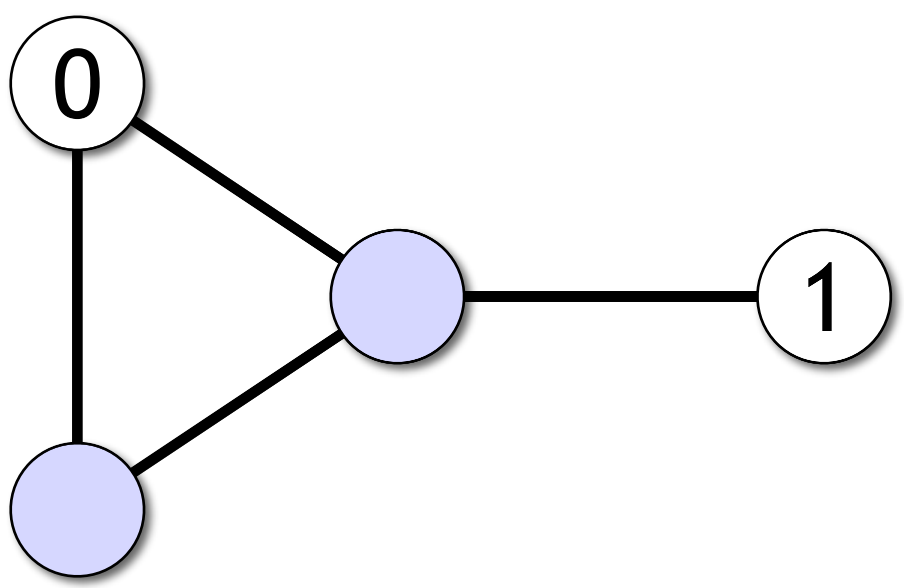
- Communication round: each node
    1. sends a message to each neighbour
    2. receives a message from each neighbour

Synchronous distributed algorithms



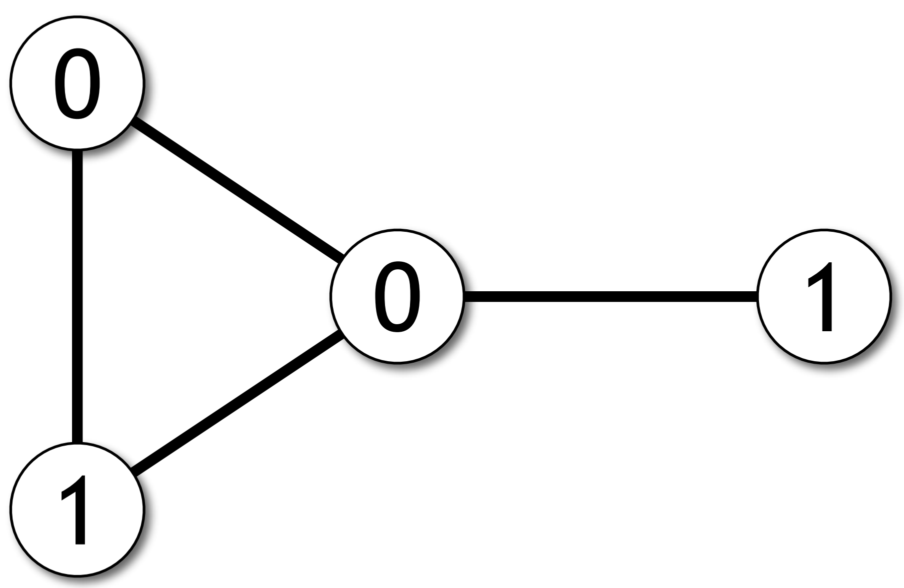
- Communication round: each node
  1. sends a message to each neighbour
  2. receives a message from each neighbour
  3. updates its own state

Synchronous distributed algorithms



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

- If the nodes have unique identifiers, “everything” can be solved in diameter(G) + 1 rounds
- Algorithm: each node
    1. gathers full information about G (including all local inputs)
    2. solves the graph problem by brute force
    3. chooses its local output accordingly

Synchronous distributed algorithms

- If the nodes have unique identifiers, "everything" can be solved in diameter(G) + 1 rounds
- Natural research problems:
  - What can be solved in $o(\text{diam}(G))$ rounds?
    - Focus: **local algorithms**
  - What if we do not have unique identifiers?
    - Focus: **port-numbering model**

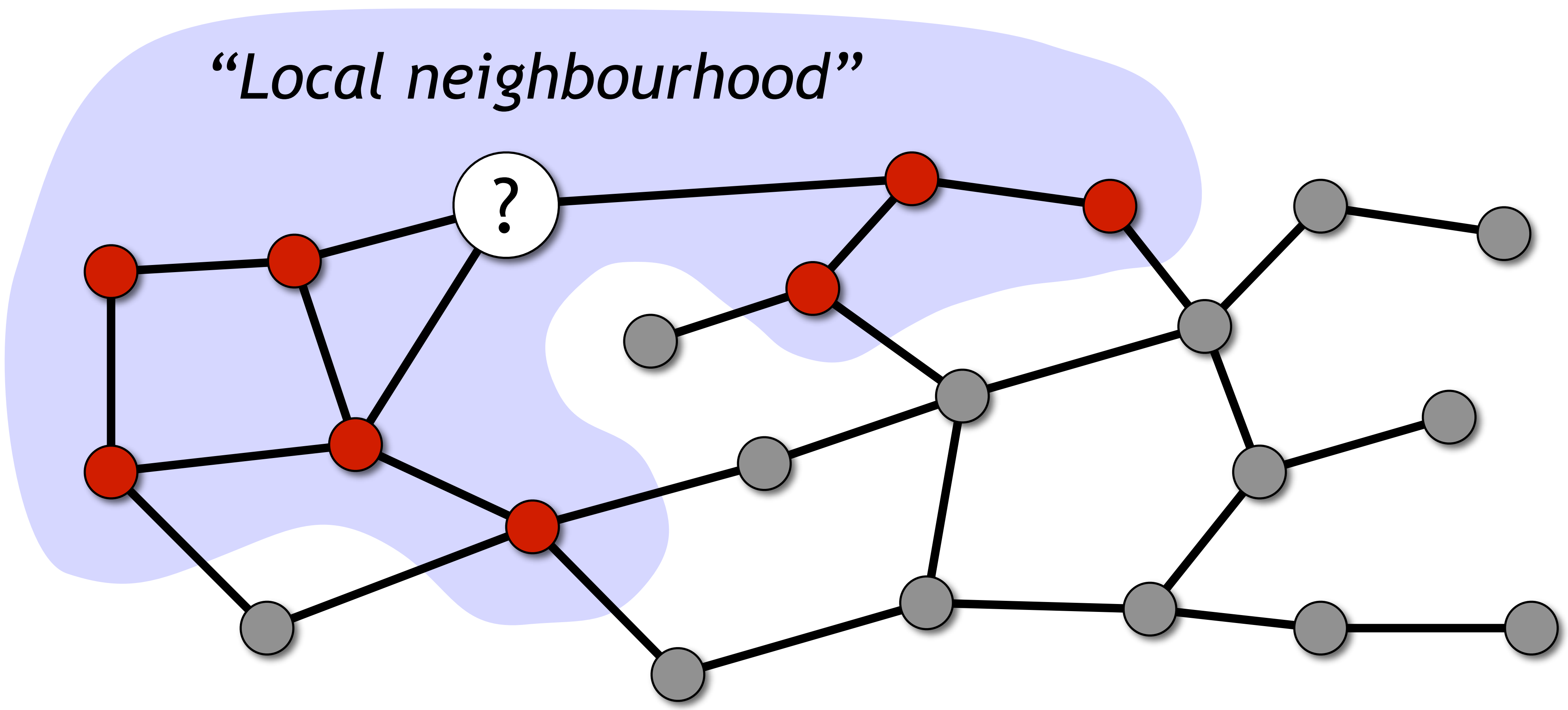
Model 1: Local algorithms

- An extreme version of sublinear-time algorithms: running time **independent** of the number of nodes
- Examples:
  - running time 100 rounds in any graph
  - running time $f(\Delta)$ in graphs with maximum degree $\leq \Delta$
- Our focus: deterministic local algorithms

Deterministic local algorithms

- Running time is $T \iff$
  output is a function of input within distance T

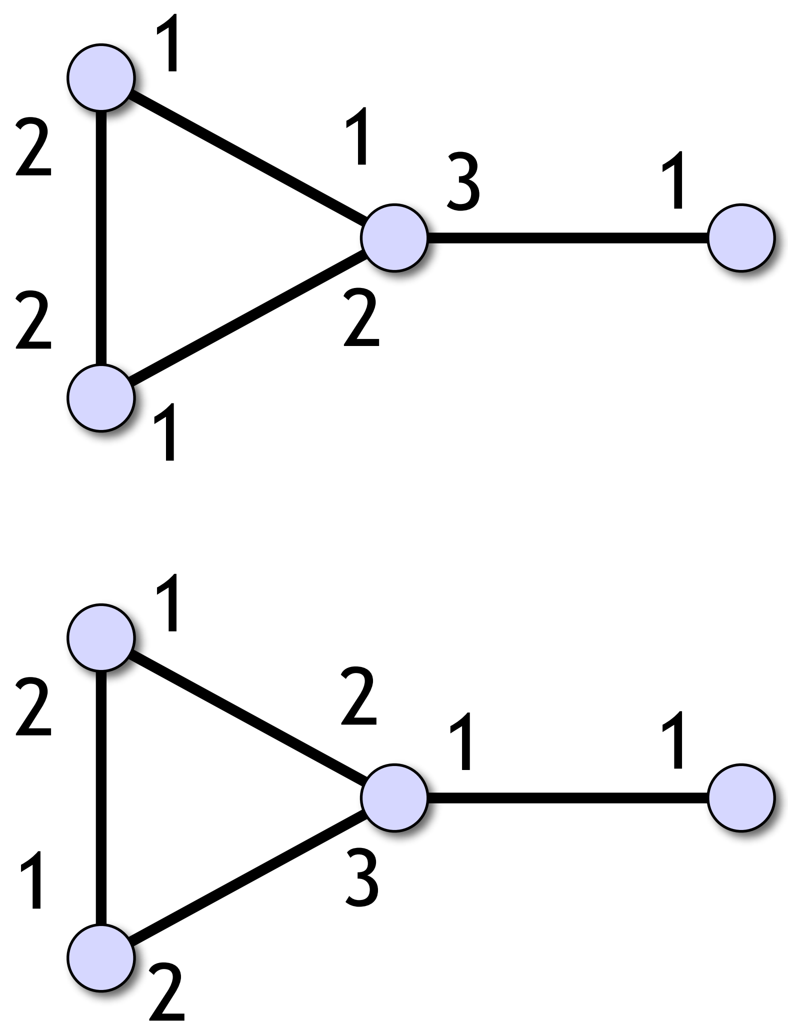
$T = 2$:



Deterministic local algorithms

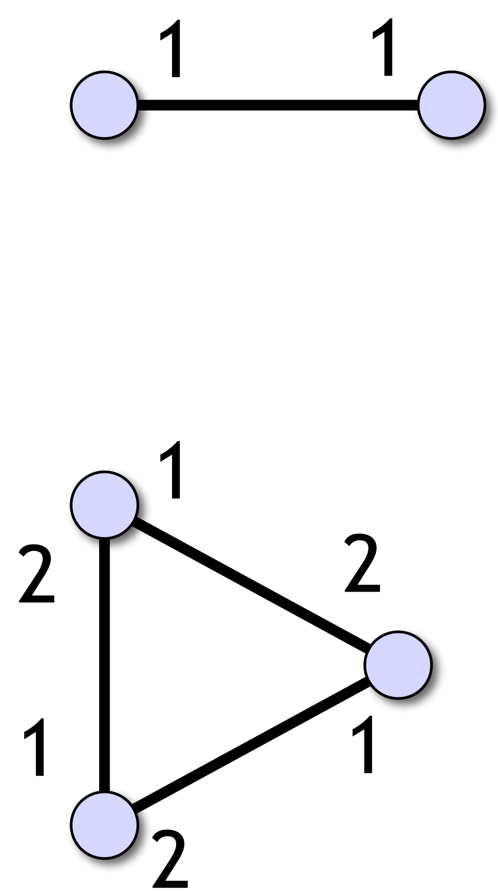
- Scalability:
  - Can be used in infinitely large (but locally finite) graphs
- Fault tolerance:
  - Output can be re-computed repeatedly
  - Efficient self-stabilising algorithm,
    recovers from any initial configuration,
    can be used in dynamic graphs
- Very limited model: **what can be computed**?

Model 2: Port-numbering model



- No unique identifiers
- A node of degree *d* can refer to its neighbours by integers 1, 2, ..., *d*
- Port-numbering chosen by adversary
- Focus: deterministic algorithms

Deterministic algorithms in the port-numbering model



- Graph + port numbering may be symmetric
- Nodes indistinguishable
  - Identical inputs, deterministic computation, identical outputs
- Very limited model: **what can be computed**?

Local algorithms and the port-numbering model

- Very limited models of distributed computing
  - Local algorithms: constant time
  - Port-numbering model: anonymous nodes
- Seemingly unrelated
  - Why did I choose to introduce both?
- What can be said about these models?
  - Certainly plenty of negative results,
    but do we have anything positive?

Part II: Observations and results

- Similarities between local algorithms and the port-numbering model
- Case study: vertex cover problem
  - Joint work with Matti Åstrand
- Examples of other positive results

Local algorithms and the port-numbering model

- Orthogonal models
- All 4 combinations are reasonable
- All 4 combinations are distinct
  - Simple (contrived) examples...

Any running time		
Local algorithms		
	Port numbering	Unique IDs

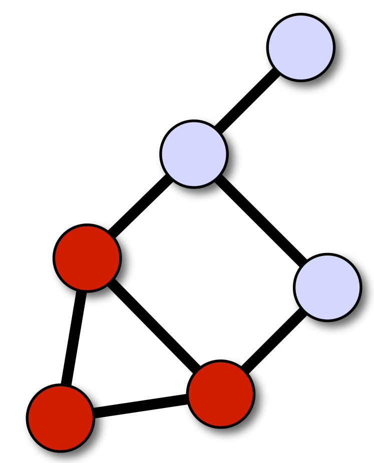
Local algorithms and the port-numbering model

- All 4 combinations are distinct
- Trivial problems can be solved in any model

Any running time		
Local algorithms	Constant function	
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

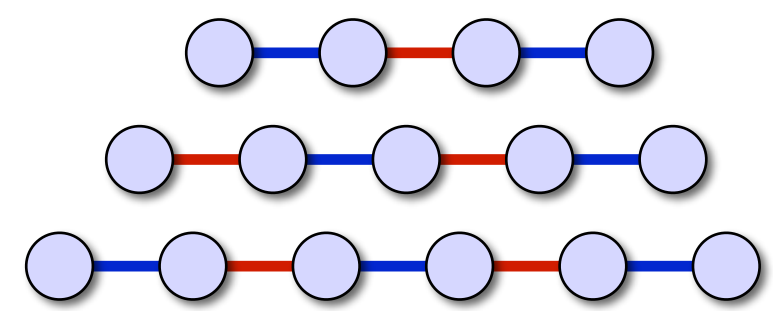
- All 4 combinations are distinct
- Identifying all triangles (3-cycles):
  - Local information is sufficient, but unique IDs are needed to distinguish between a cycle and a long path



Any running time		
Local algorithms	Constant function	Find triangles
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

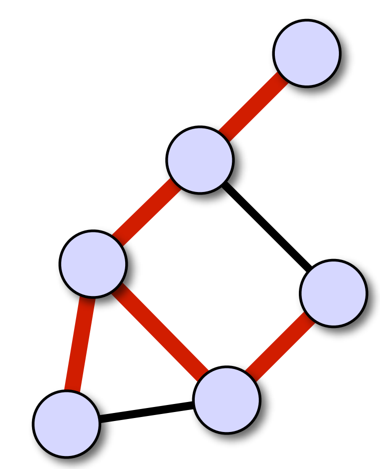
- All 4 combinations are distinct
- 2-colouring edges of paths:
  - Port numbering is sufficient, but the worst-case running time is necessarily $\theta(\mathrm{diam}(G))$



Any running time	Path colouring	
Local algorithms	Constant function	Find triangles
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

- All 4 combinations are distinct
- Spanning tree construction:
  - Non-local problem
  - Unique IDs needed to detect cycles



Any running time	Path colouring	Spanning trees
Local algorithms	Constant function	Find triangles
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

- All 4 combinations are distinct
- However, there are surprising similarities between local algorithms and the port-numbering model
  - Not fully understood yet!

Any running time		
Local algorithms		
	Port numbering	Unique IDs

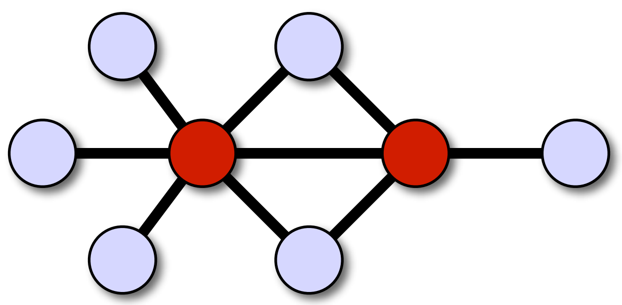
Local algorithms and the port-numbering model

- There are problems where both models seem to be equally strong
    - Best algorithm in port-numbering model is local
    - Best local algorithm uses the port-numbering model

Any running time		
Local algorithms		
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

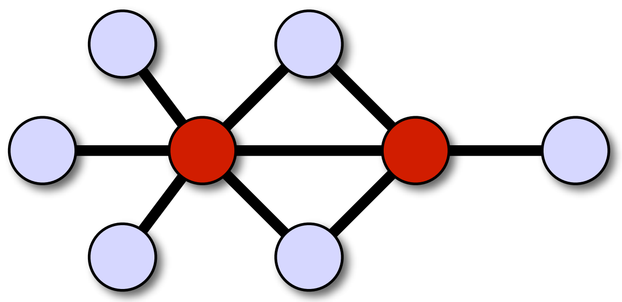
- Example: minimum vertex cover
  - Find a minimum-size subset *C* of nodes that “covers” all edges: each edge incident to at least one node in *C*
  - Classical NP-hard optimisation problem



Any running time		
Local algorithms		
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

- Example: minimum vertex cover
- Best possible approximation ratio?
  - Focus on bounded-degree graphs



Any running time		
Local algorithms		
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

- Example: minimum vertex cover
- Trivial lower bound
  - Cycles, optimum $n/2$
  - Solution with < n nodes requires symmetry-breaking

Any running time	≥ 2	
Local algorithms		
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

- Example: minimum vertex cover
- Non-trivial lower bound
  - Cycles
  - Czygrinow et al. 2008, Lenzen & Wattenhofer 2008

Any running time	≥ 2	
Local algorithms		≥ 2
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

- Example: minimum vertex cover
- Matching positive result
  - Bounded-degree graphs
  - One algorithm for both models

Any running time	≥ 2	
Local algorithms	**≤ 2**	≥ 2
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

- Example: minimum vertex cover
- Best possible approximation ratios in bounded-degree graphs

Any running time	**2**	1
Local algorithms	**2**	**2**
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

- Naturally, we can study running time with a finer granularity than $O(1)$ vs. arbitrary...
- However, anything larger-than-constant seems to lead to a very different model

Any running time		
Local algorithms		
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

- Slightly non-constant running time together with unique IDs already makes a huge difference

$O(n)$		
$O(\log n)$		
$O(\log^* n)$		
$O(1)$		
	Port numbering	Unique IDs

Iterated logarithm
≈ inverse of tetration
(power tower)

Local algorithms and the port-numbering model

- Slightly non-constant running time together with unique IDs already makes a huge difference

$O(n)$		
$O(\log n)$		
$O(\log^* n)$		Cole–Vishkin 1986
$O(1)$		Linial 1992
	Port numbering	Unique IDs

Deterministic symmetry breaking in cycles

Negative result

Local algorithms and the port-numbering model

- E.g., vertex cover in cycles becomes easier to approximate

$O(n)$	2	
$O(\log n)$	2	Greedy
$O(\log^* n)$	2	≤ 4/3
$O(1)$	2	2
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

- E.g., vertex cover in cycles becomes much easier to approximate

$O(n)$	2	
$O(\log n)$	2	Clustering
$O(\log^* n)$	2	$\leq 1 + \varepsilon$
$O(1)$	2	2
	Port numbering	Unique IDs

Local algorithms and the port-numbering model

- Hence the focus: strictly constant time and/or anonymous nodes

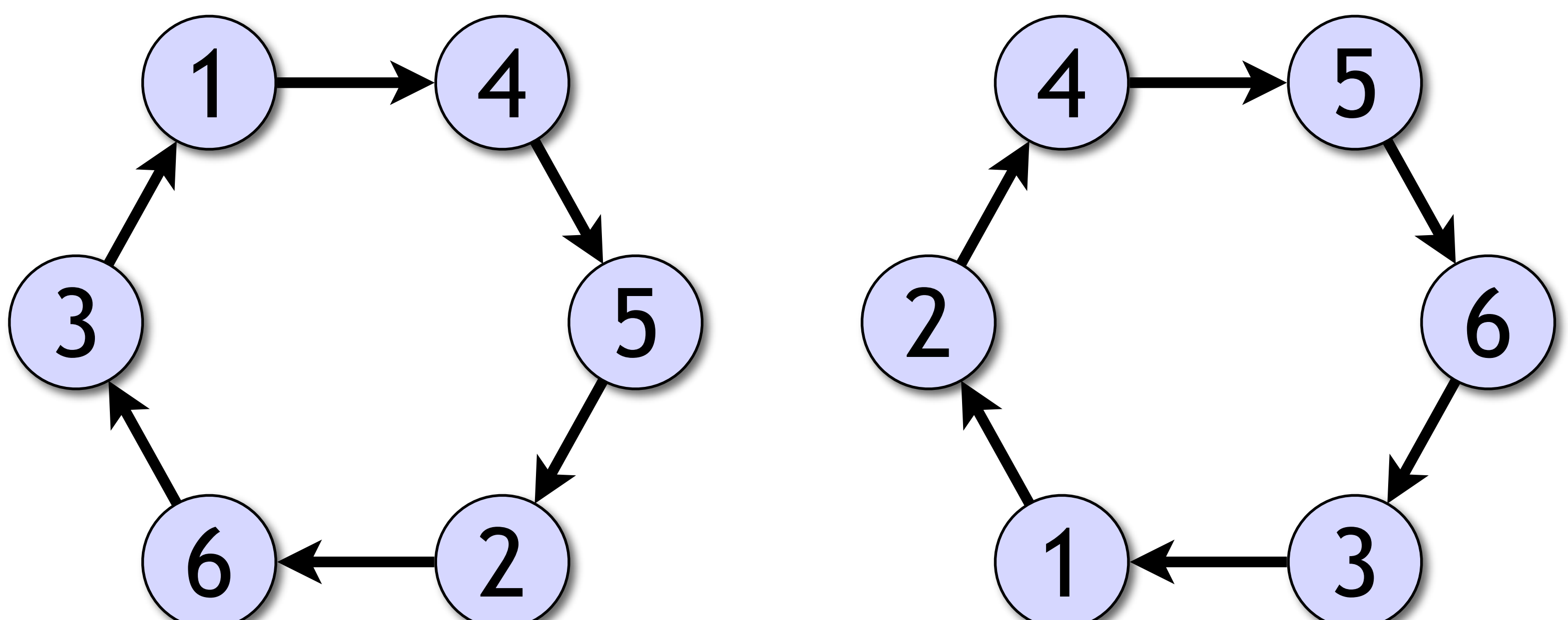
$O(n)$		
$O(\log n)$		
$O(\log^* n)$		
$O(1)$		
	Port numbering	Unique IDs

Case study:
2-approximation of vertex cover

- Lower bound result (for cycles):
  - There is no local algorithm with approximation factor $2 - \varepsilon$ for any $\varepsilon > 0$
  - I'll sketch Czygrinow et al.'s (2008) proof, which is a nice application of Ramsey's theorem
- Fast local algorithm (for bounded-degree graphs):
  - 2-approximation in $O(\Delta)$ time in unweighted graphs
  - Uses LP duality; finds a maximal dual solution using a combination of greedy increments and graph colouring

Lower-bound result for vertex cover approximation

- Numbered directed *n*-cycle:
  - directed *n*-cycle, each node has outdegree = indegree = 1
  - node identifiers are a permutation of {1, 2, ..., *n*}



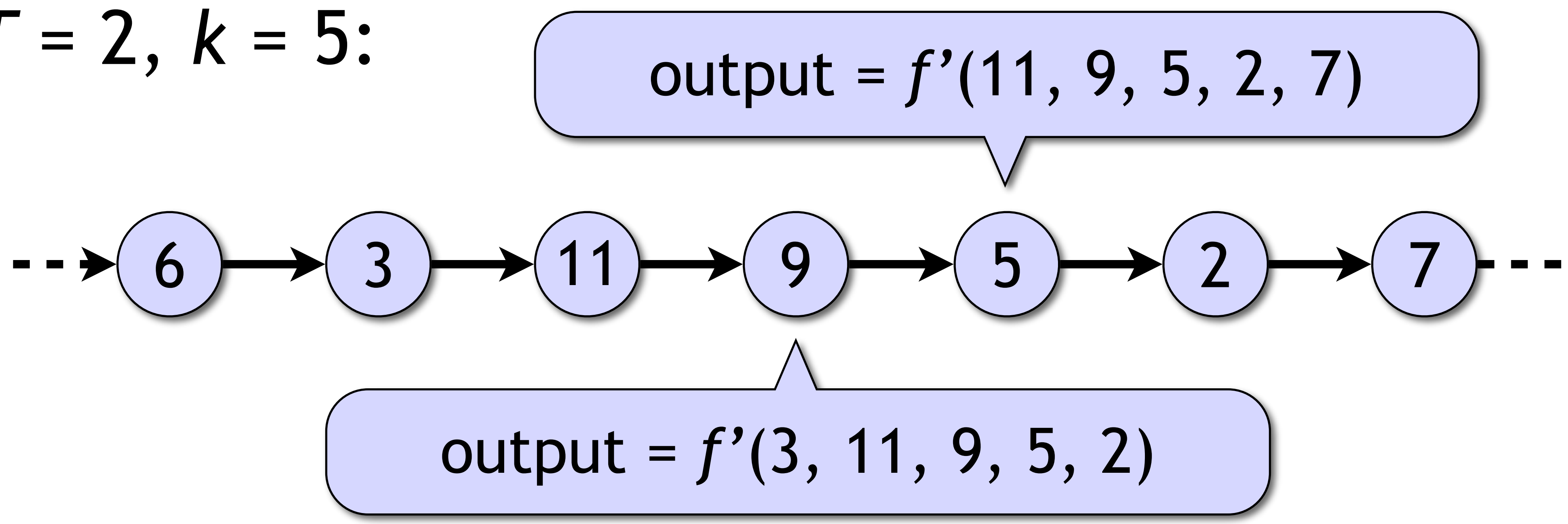
Lower-bound result for vertex cover approximation

- Fix any $\varepsilon > 0$ and a deterministic local algorithm A
  - Assumption: A finds a feasible vertex cover (at least in any numbered directed cycle)
- **Theorem:** For a sufficiently large n there is a numbered directed n-cycle C in which A outputs a vertex cover with $\geq (1 - \varepsilon)n$ nodes
- **Corollary:** Approximation ratio of A is at least $2 - 2\varepsilon$

Lower-bound result for vertex cover approximation

- Let T be the running time of A, let $k = 2T + 1$
- The output of a node is a function f' of a sequence of k integers (unique IDs)

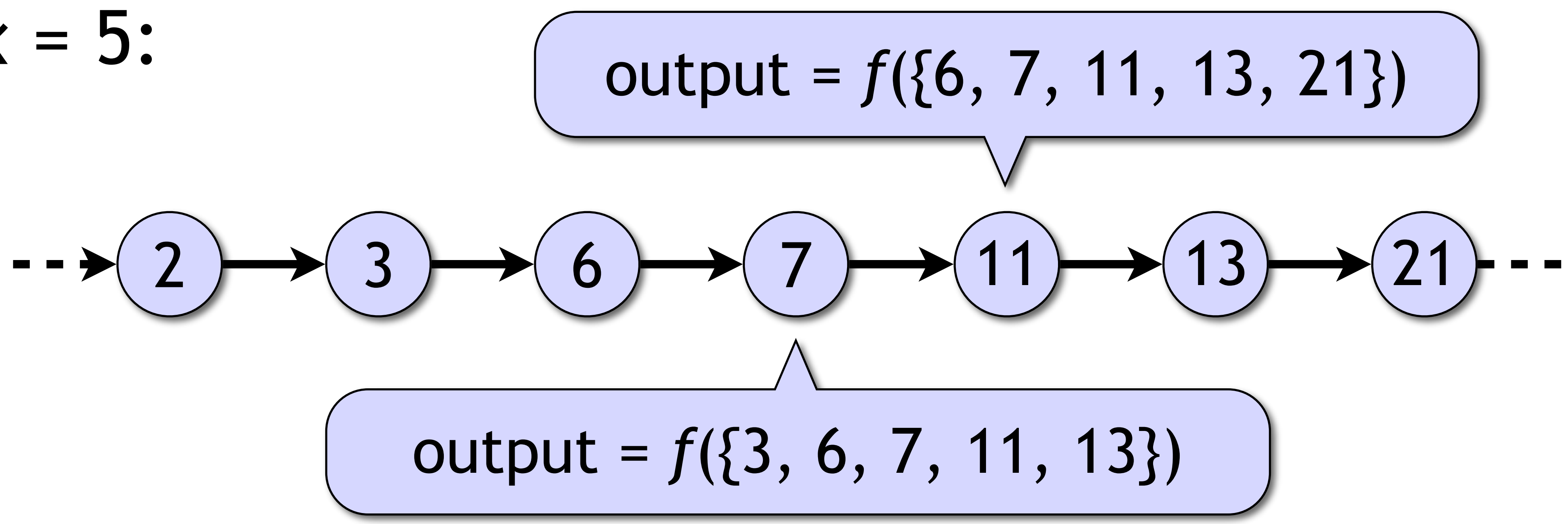
$T = 2$, $k = 5$:



Lower-bound result for vertex cover approximation

- Lets focus on **increasing** sequences of IDs
- Then the output of a node is a function f of a **set** of k integers

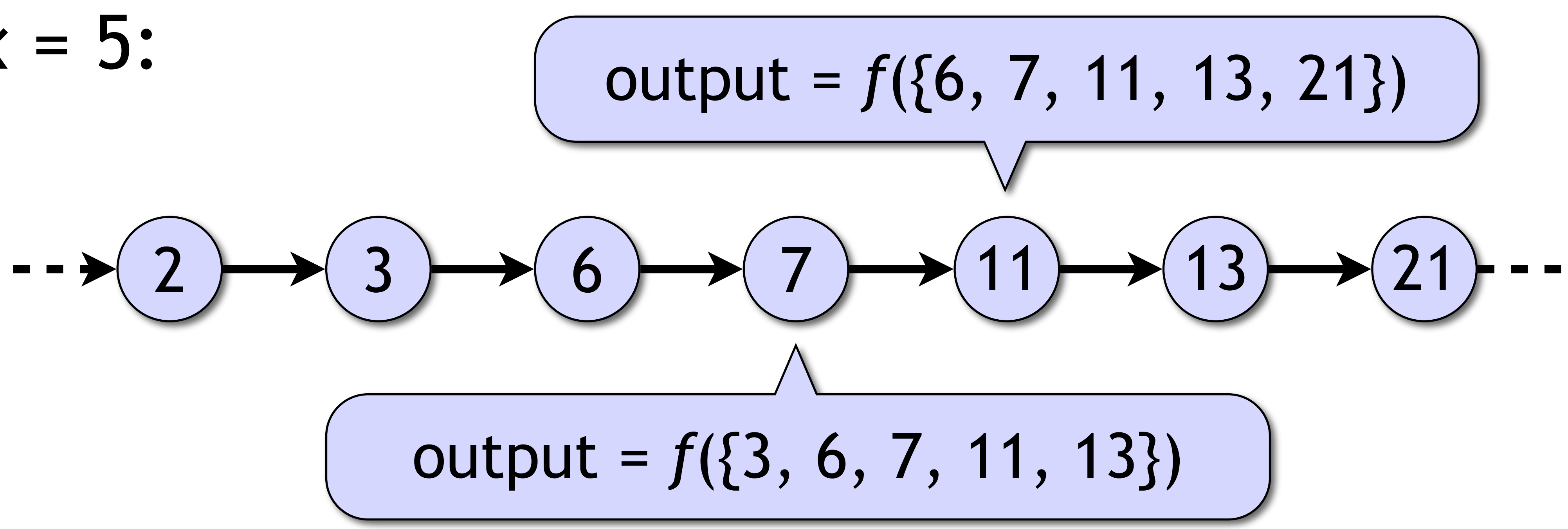
$k = 5$:



Lower-bound result for vertex cover approximation

- Hence we have assigned a colour $f(X) \in \{0, 1\}$ to each k-subset $X \subset \{1, 2, \ldots, n\}$

$k = 5$:

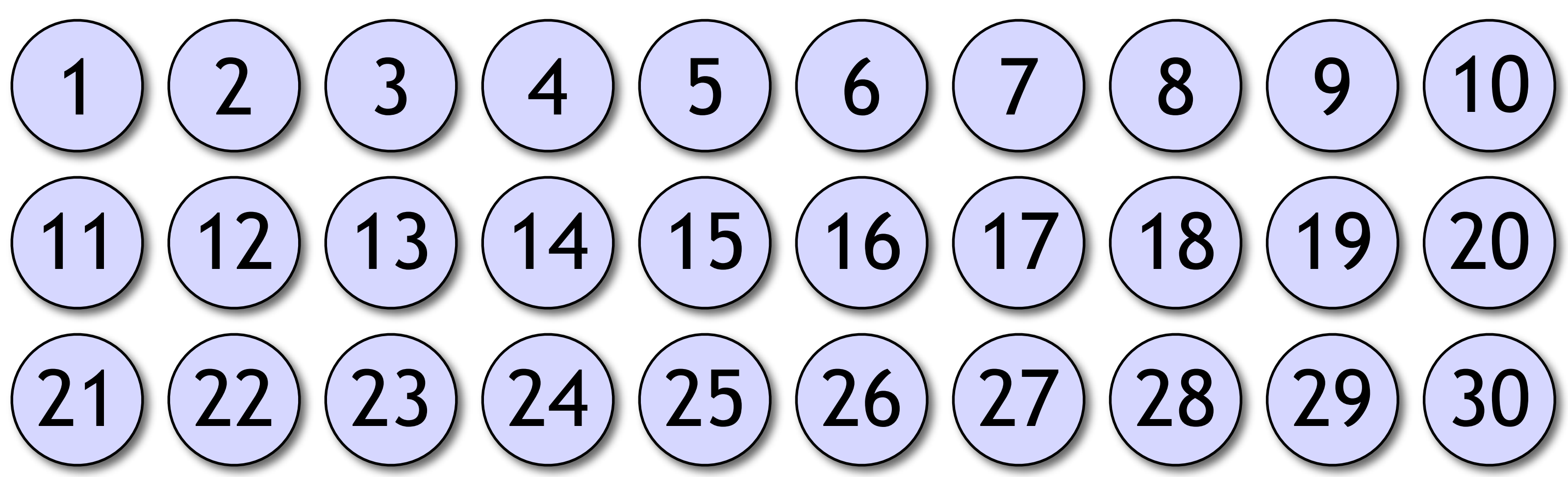


Lower-bound result for vertex cover approximation

- Hence we have assigned a colour $f(X) \in \{0, 1\}$ to each k-subset $X \subset \{1, 2, \ldots, n\}$
- Fix a large m (depends on k and ε)
- Ramsey: If n is sufficiently large, we can find an m-subset $A \subset \{1, 2, \ldots, n\}$ s.t. all k-subset $X \subset A$ have the same colour

Lower-bound result
for vertex cover approximation

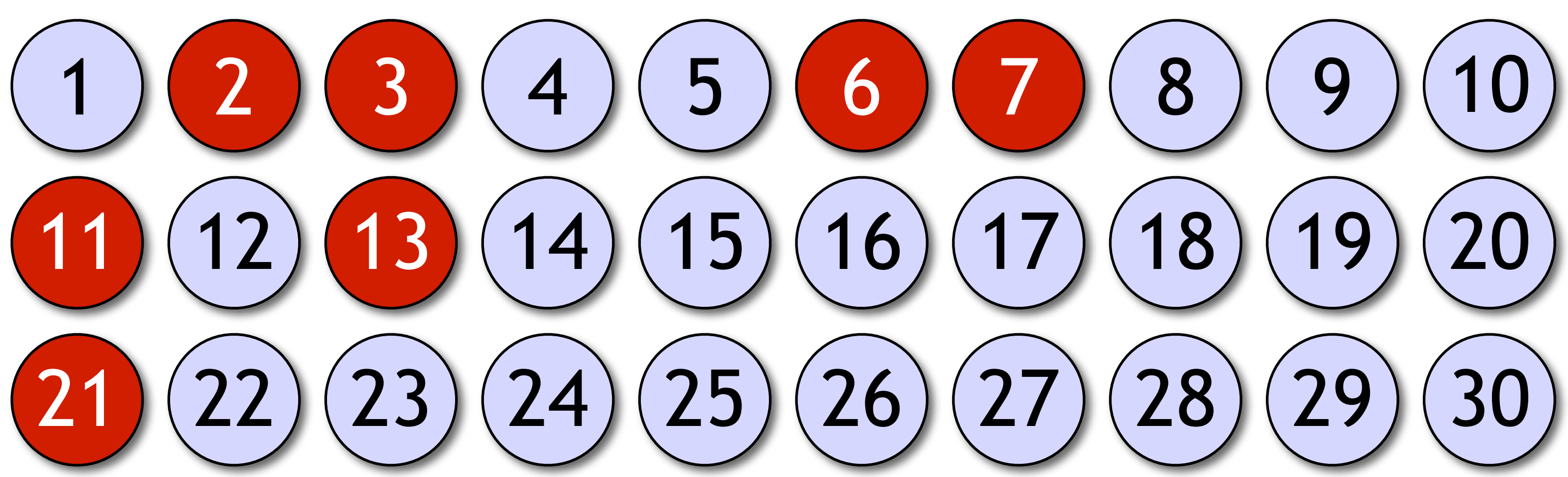
- That is, if the ID space is sufficiently large...



Lower-bound result for vertex cover approximation

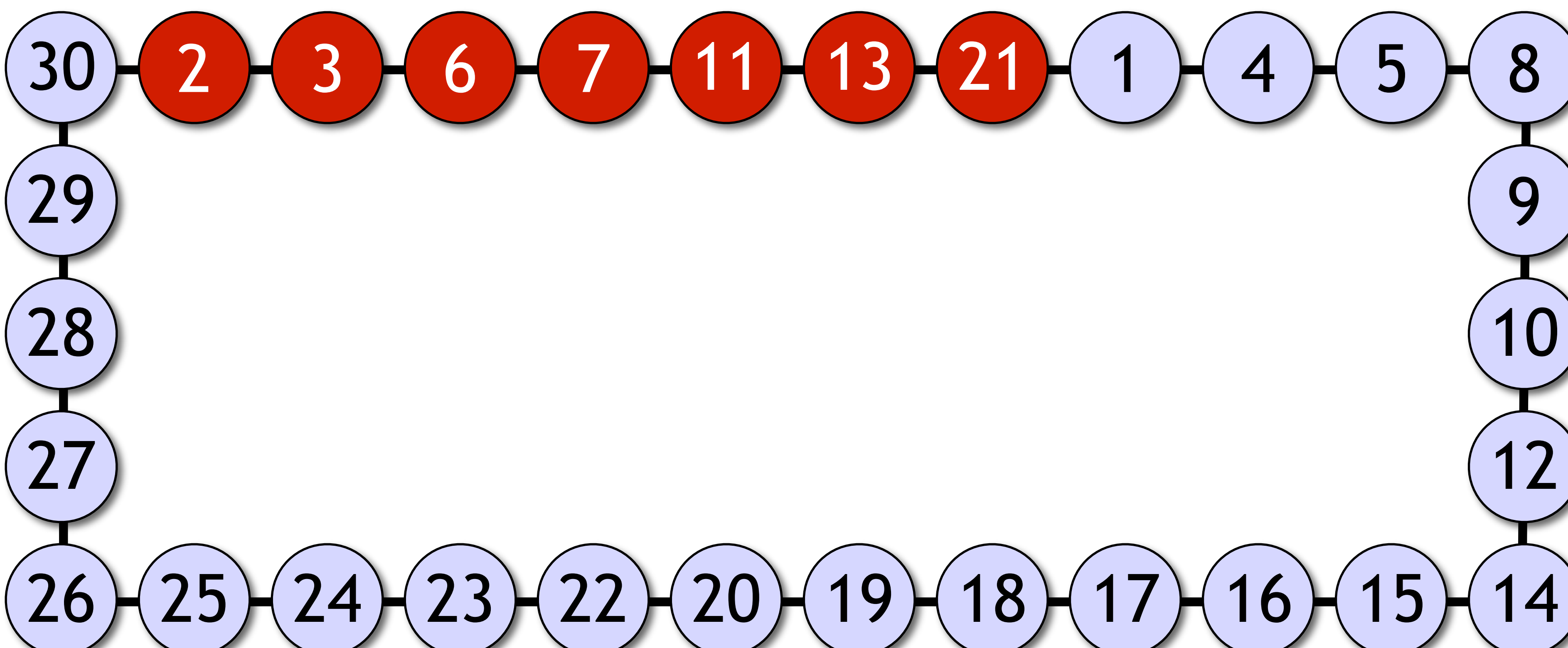
- That is, if the ID space is sufficiently large, we can find a **monochromatic** subset of m IDs...

$f(\{2, 3, 6, 7, 11\}) = f(\{2, 3, 6, 7, 13\}) =$
$f(\{2, 3, 6, 7, 21\}) = f(\{2, 3, 6, 11, 13\}) =$
$\ldots = f(\{6, 7, 11, 13, 21\})$



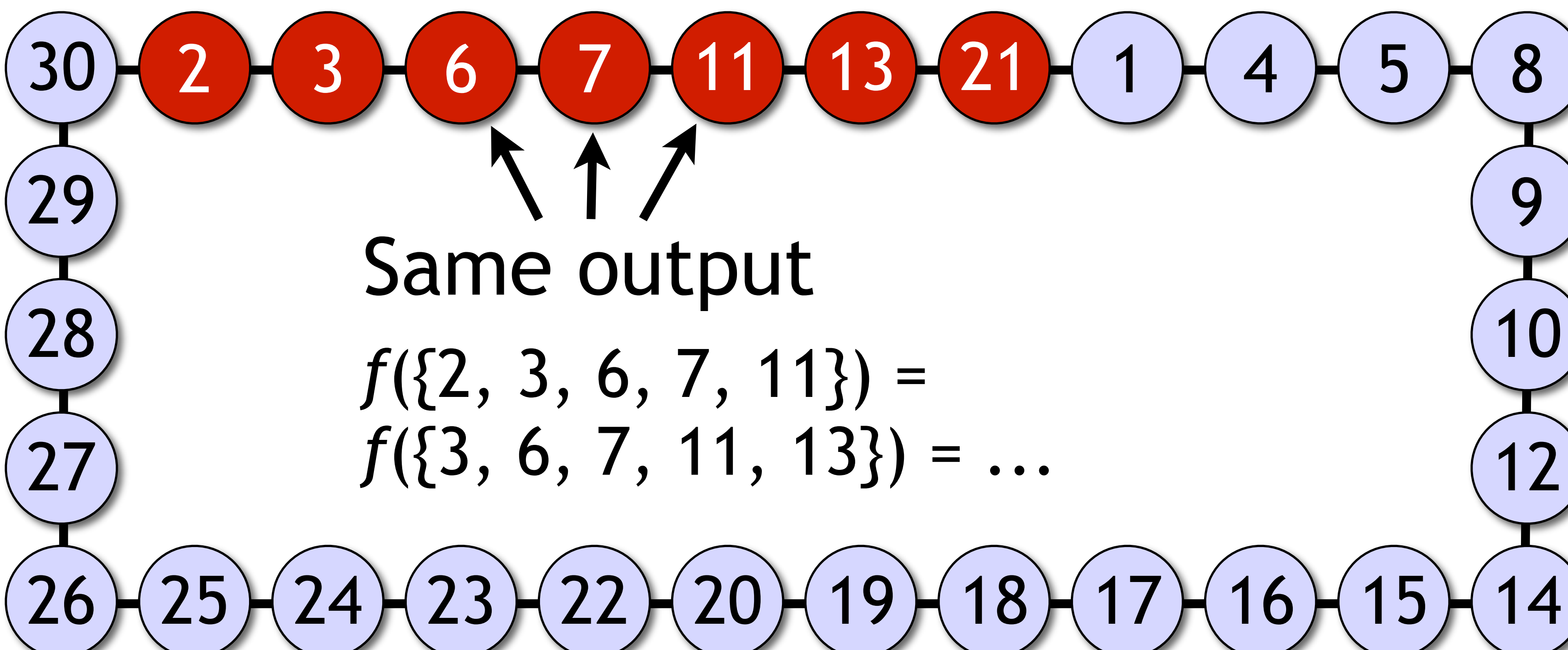
Lower-bound result for vertex cover approximation

- Construct a numbered directed cycle: monochromatic subset as consecutive nodes



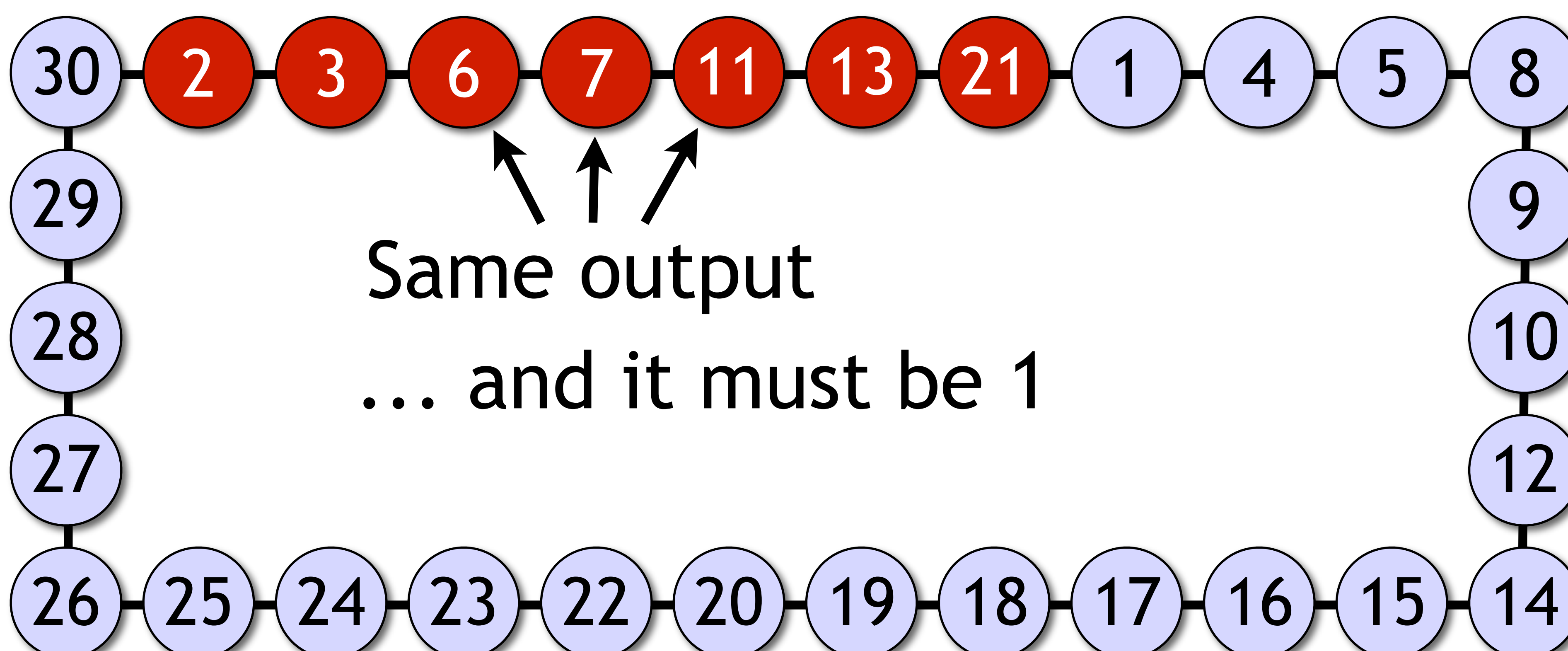
Lower-bound result for vertex cover approximation

- Construct a numbered directed cycle: monochromatic subset as consecutive nodes



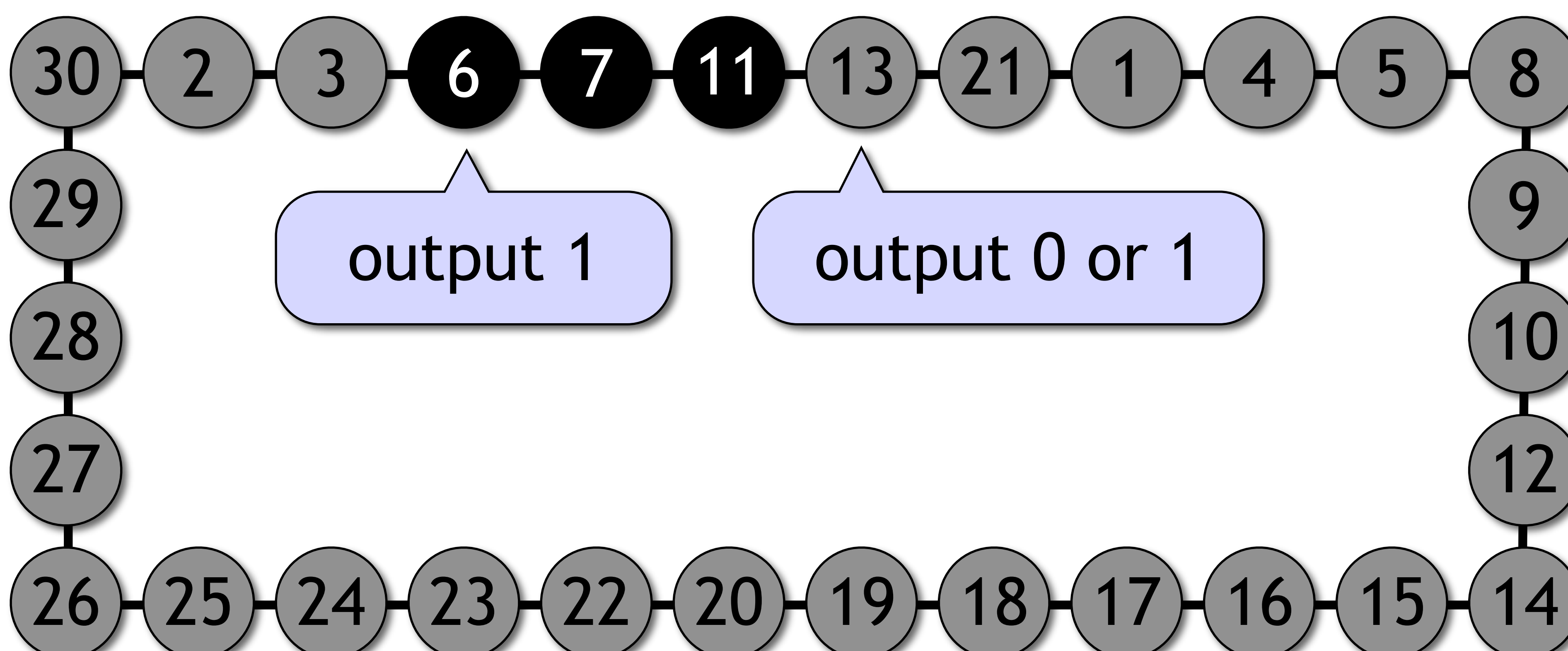
Lower-bound result for vertex cover approximation

- Construct a numbered directed cycle: monochromatic subset as consecutive nodes



Lower-bound result for vertex cover approximation

- Hence there is an *n*-cycle with a chain of $m - 2T$ nodes that output 1

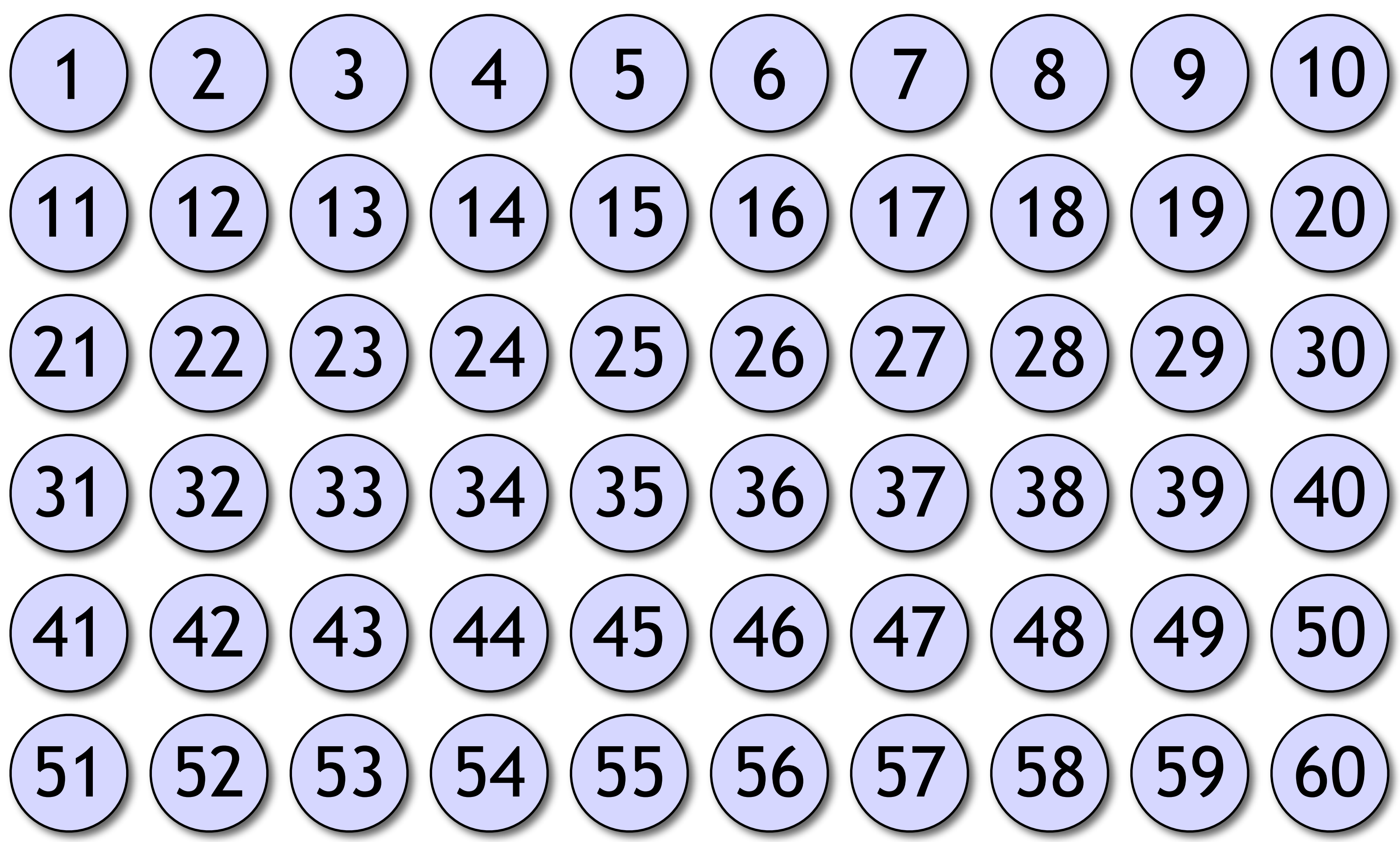


Lower-bound result for vertex cover approximation

- Hence there is an n-cycle with a chain of $m - 2T$ nodes that output 1
- We can choose as large m as we want
  - Good, more “black” nodes that output 1
- However, n increases rapidly if we increase m
  - Bad, more “grey” nodes that might output 0
- Trick: choose “unnecessarily large” n so that we can apply Ramsey’s theorem repeatedly

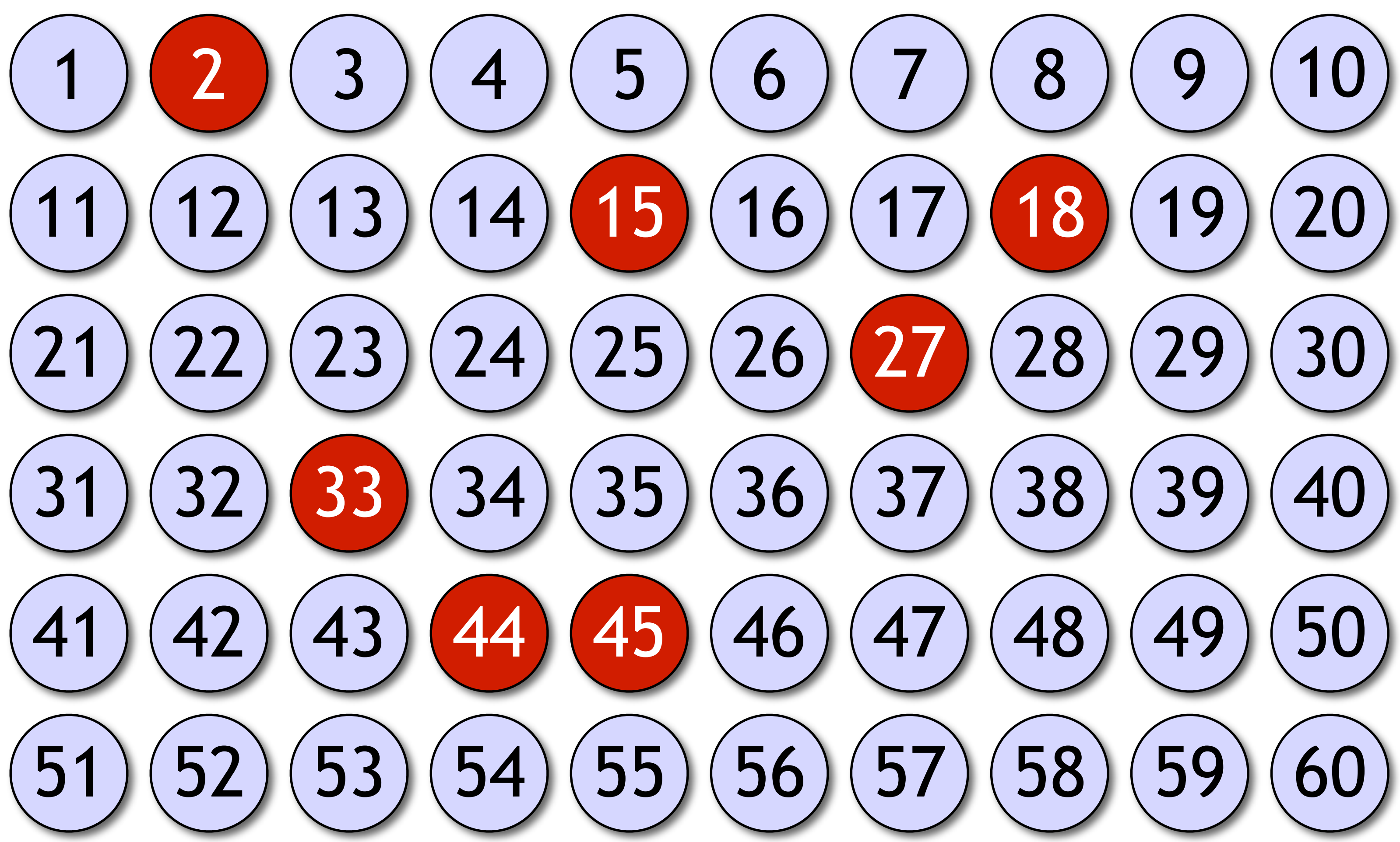
Lower-bound result for vertex cover approximation

- Huge ID space...



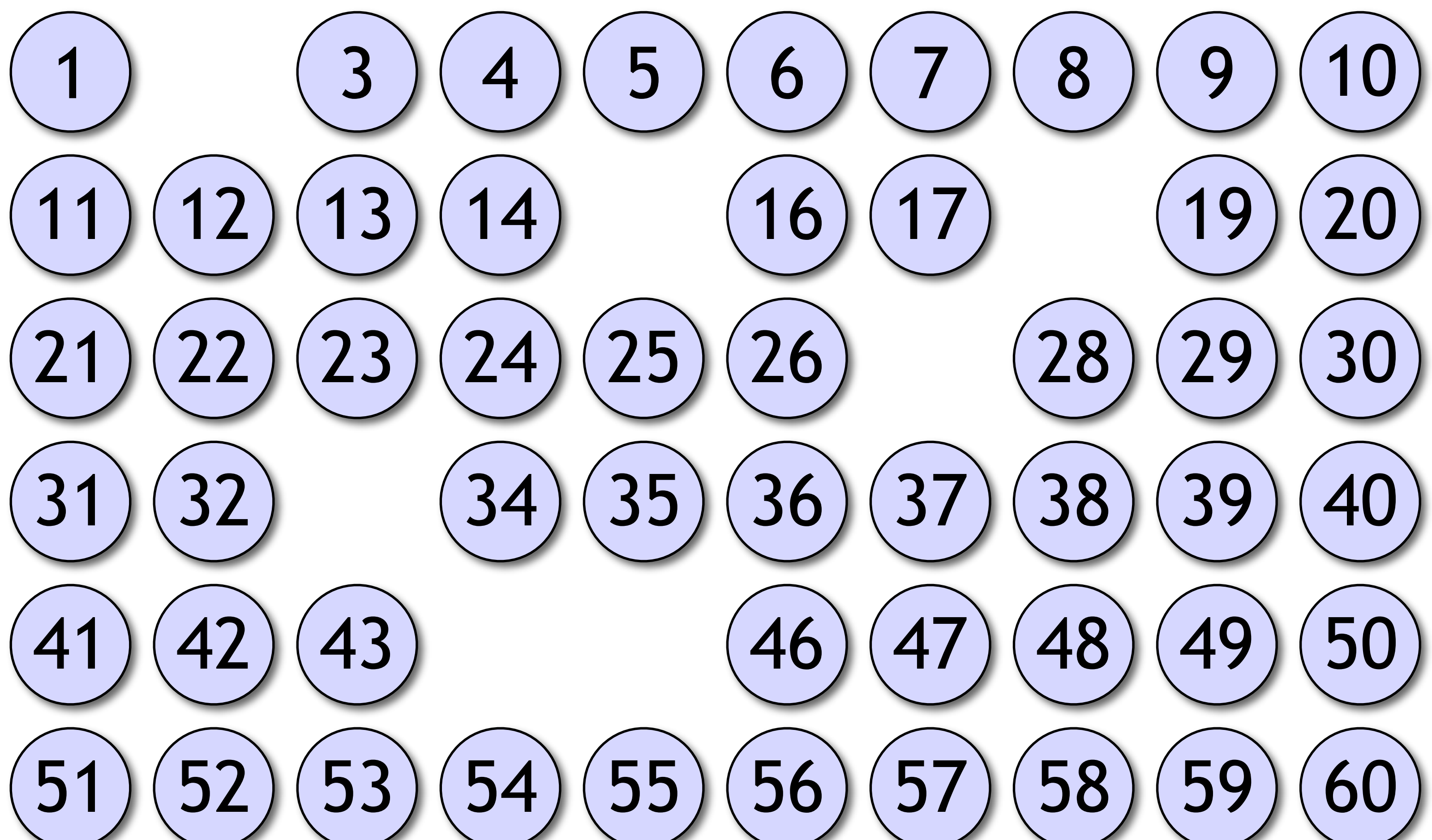
Lower-bound result for vertex cover approximation

- Find a monochromatic subset of size *m*...



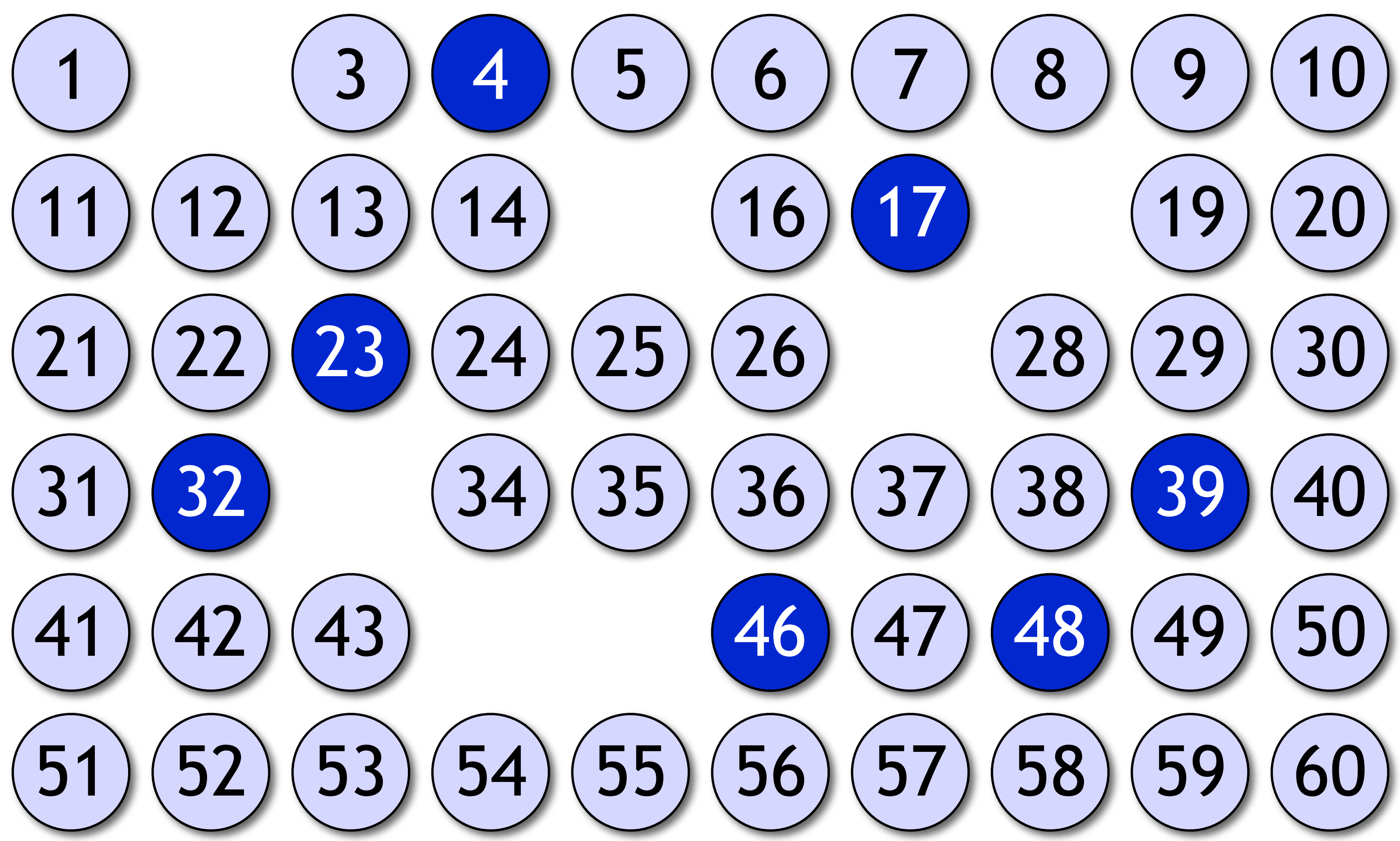
Lower-bound result for vertex cover approximation

- Delete these IDs...



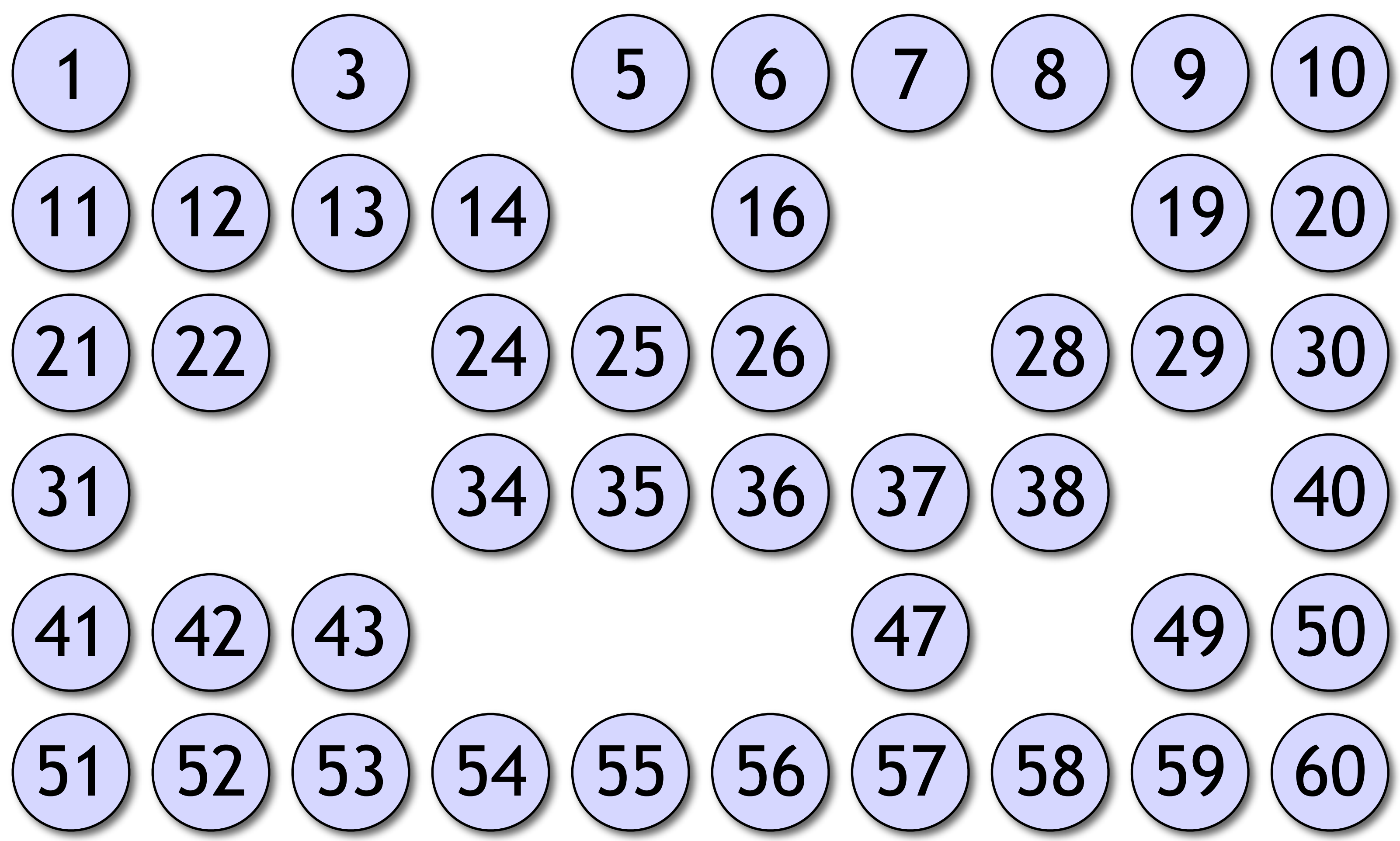
Lower-bound result for vertex cover approximation

- Still sufficiently many IDs to apply Ramsey...



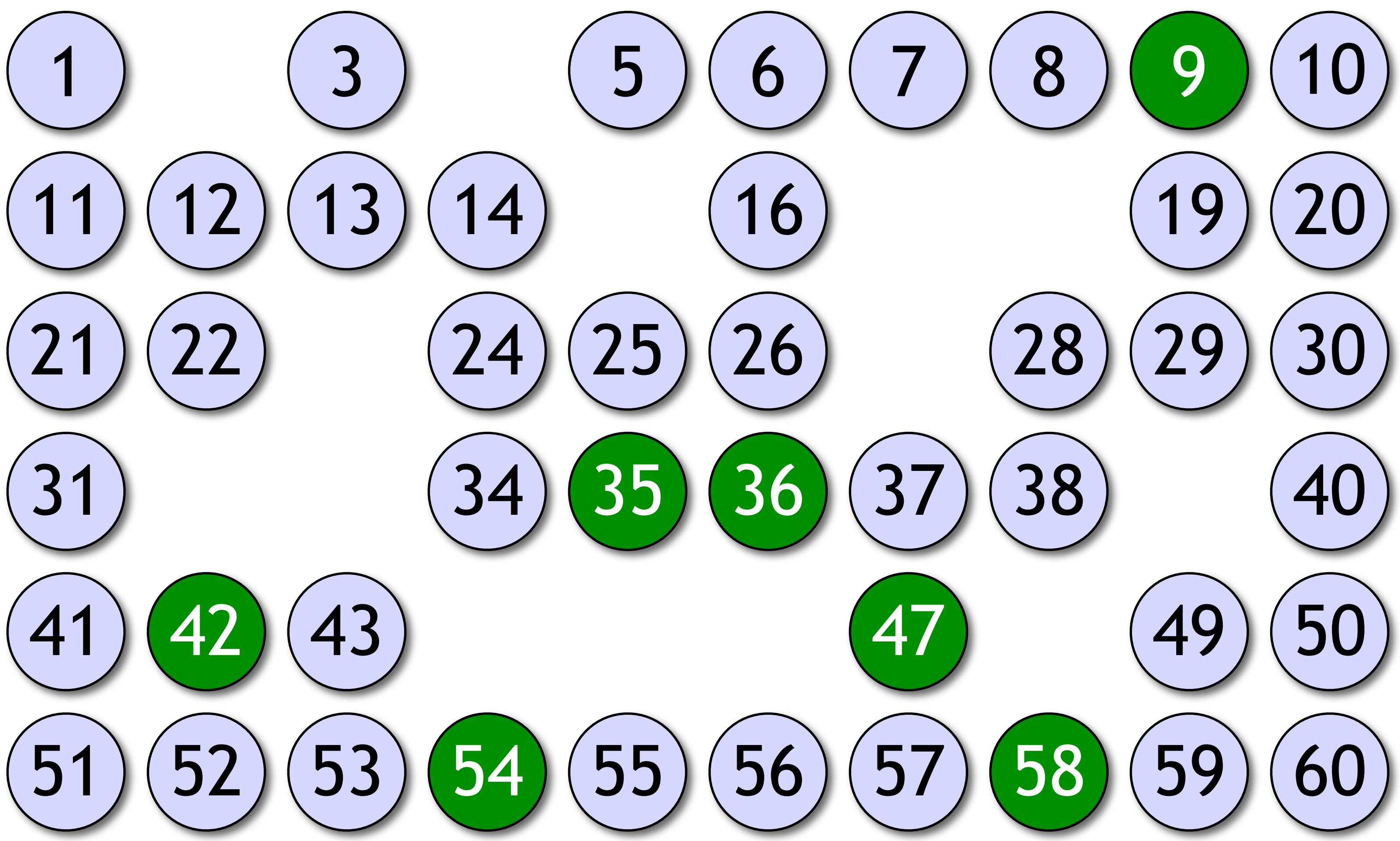
Lower-bound result for vertex cover approximation

- Repeat...



Lower-bound result for vertex cover approximation

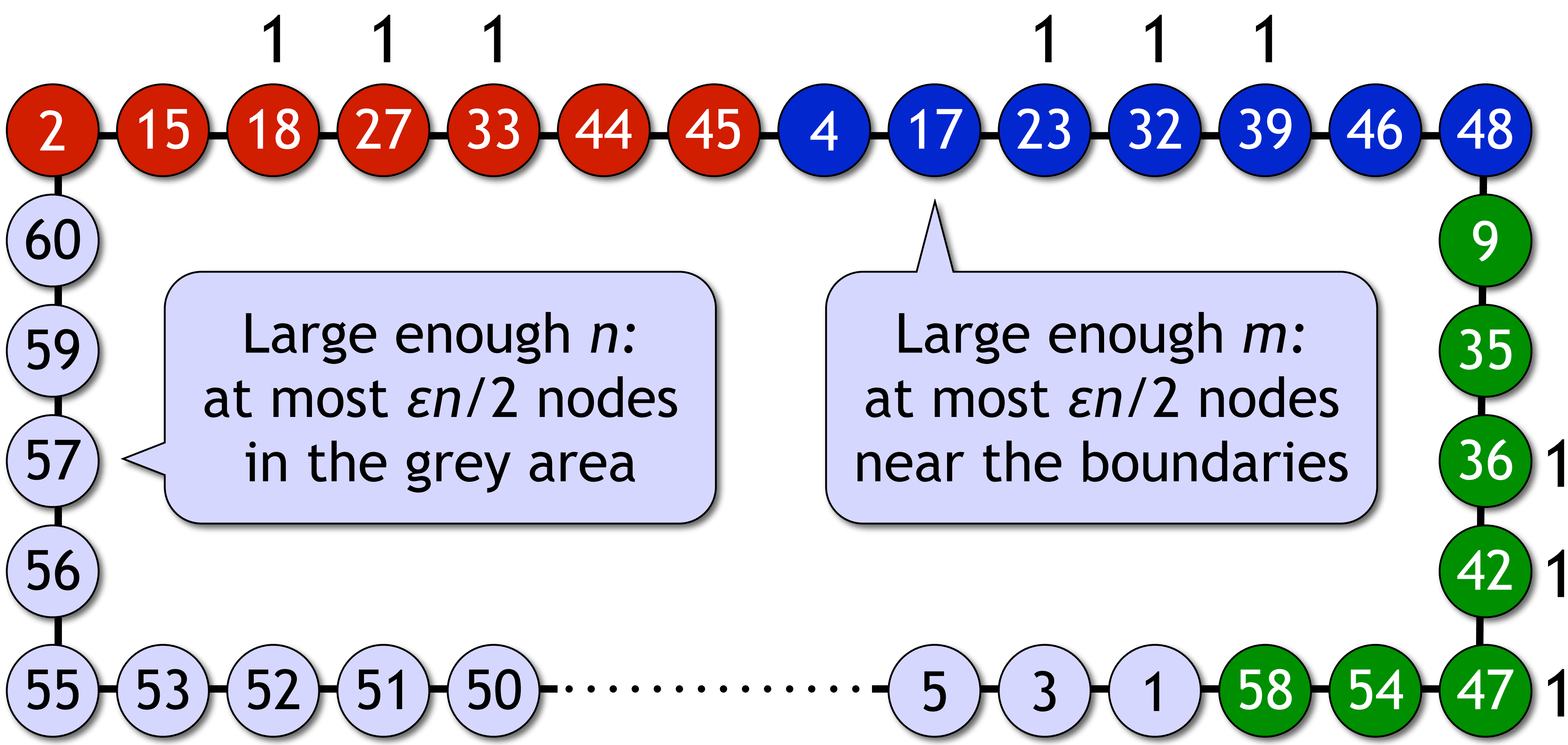
- Repeat until stuck



Lower-bound result for vertex cover approximation

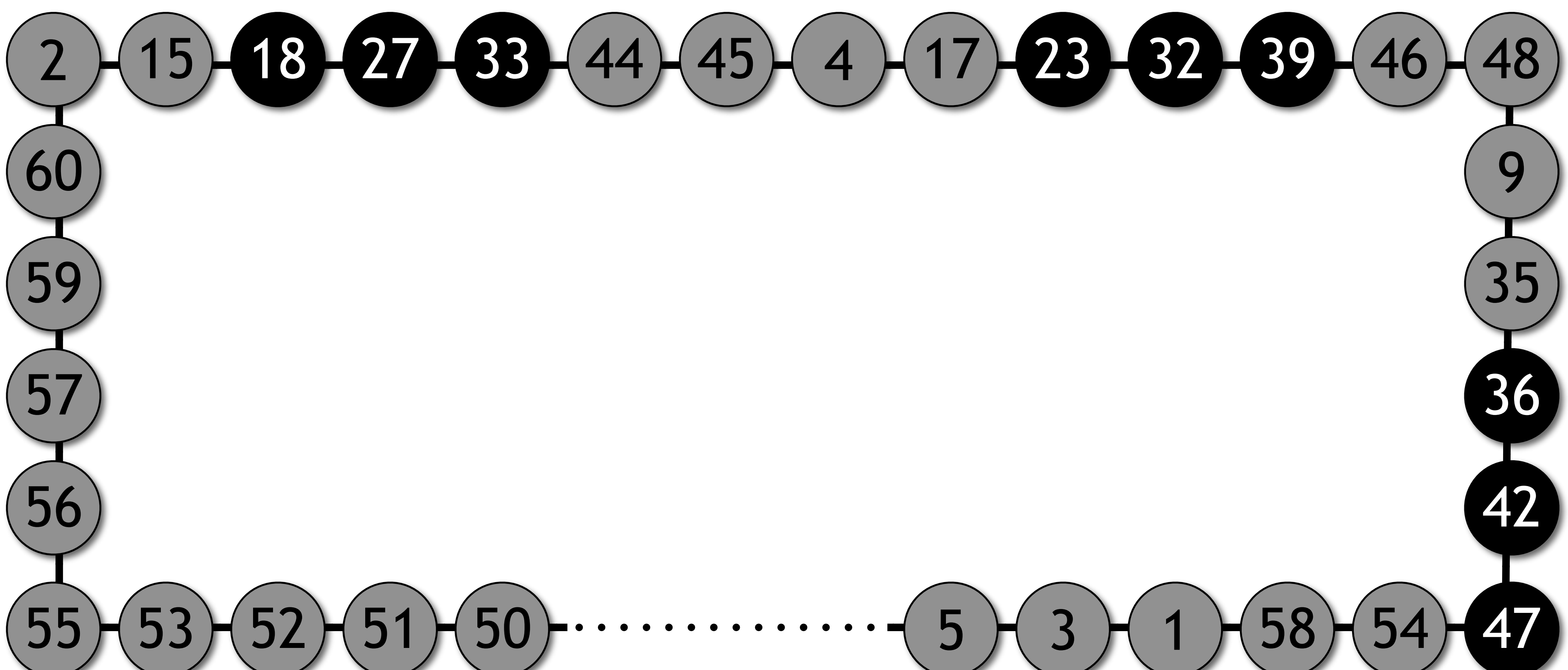
- Several monochromatic subsets + some leftovers

Lower-bound result for vertex cover approximation



Lower-bound result for vertex cover approximation

- Thus A outputs a vertex cover with $\geq (1 - \varepsilon)n$ nodes

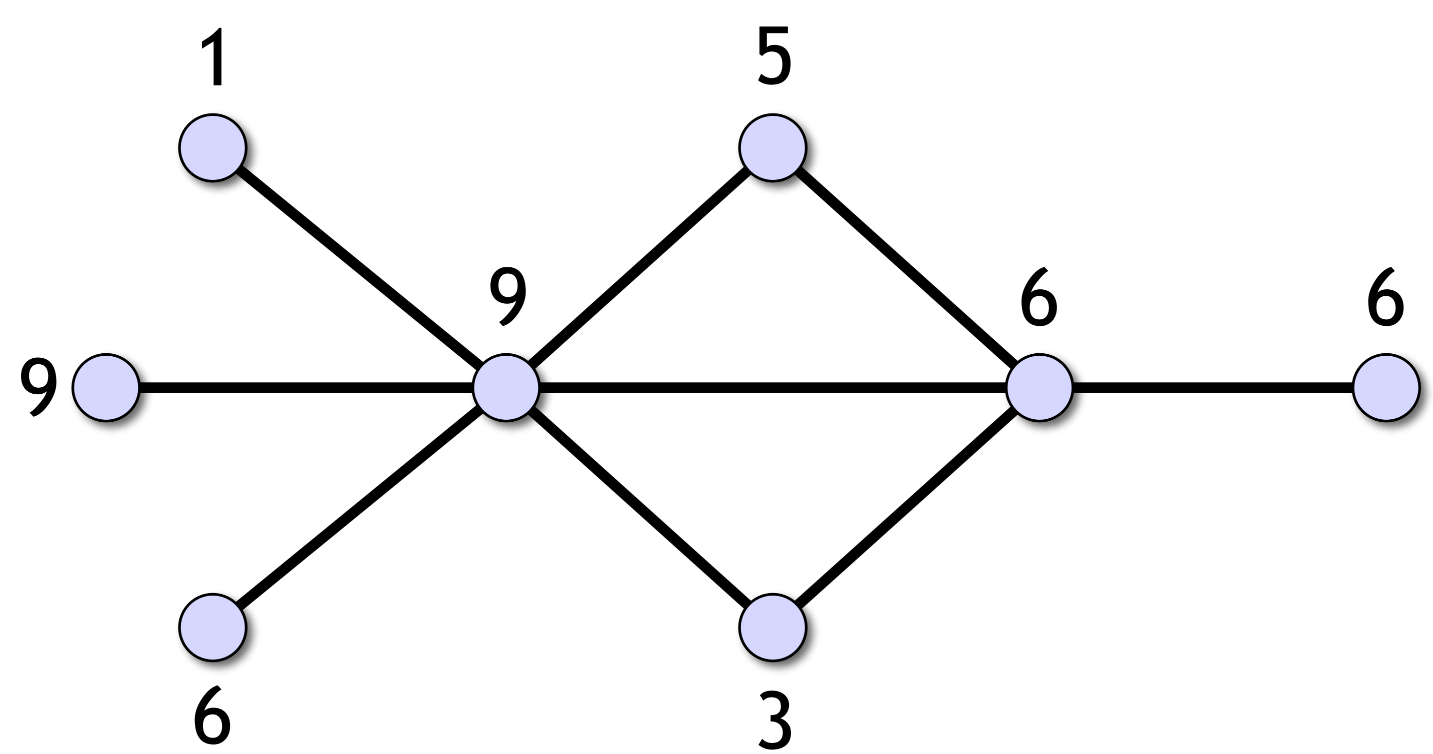


Lower-bound result for vertex cover approximation

- Thus *A* outputs a vertex cover with $\geq (1 - \varepsilon)n$ nodes
- In the proof, n is huge — and this is necessary
  - Using an upper bound on Ramsey numbers, the same proof would give a negative result for $T = o(\log^* n)$
  - With $T = \Theta(\log^* n)$, we could do better!
- We have seen that $(2 - \varepsilon)$-approximation is not possible in time independent of n
- Now let's see how to find a 2-approximation

Local 2-approximation algorithm for vertex cover

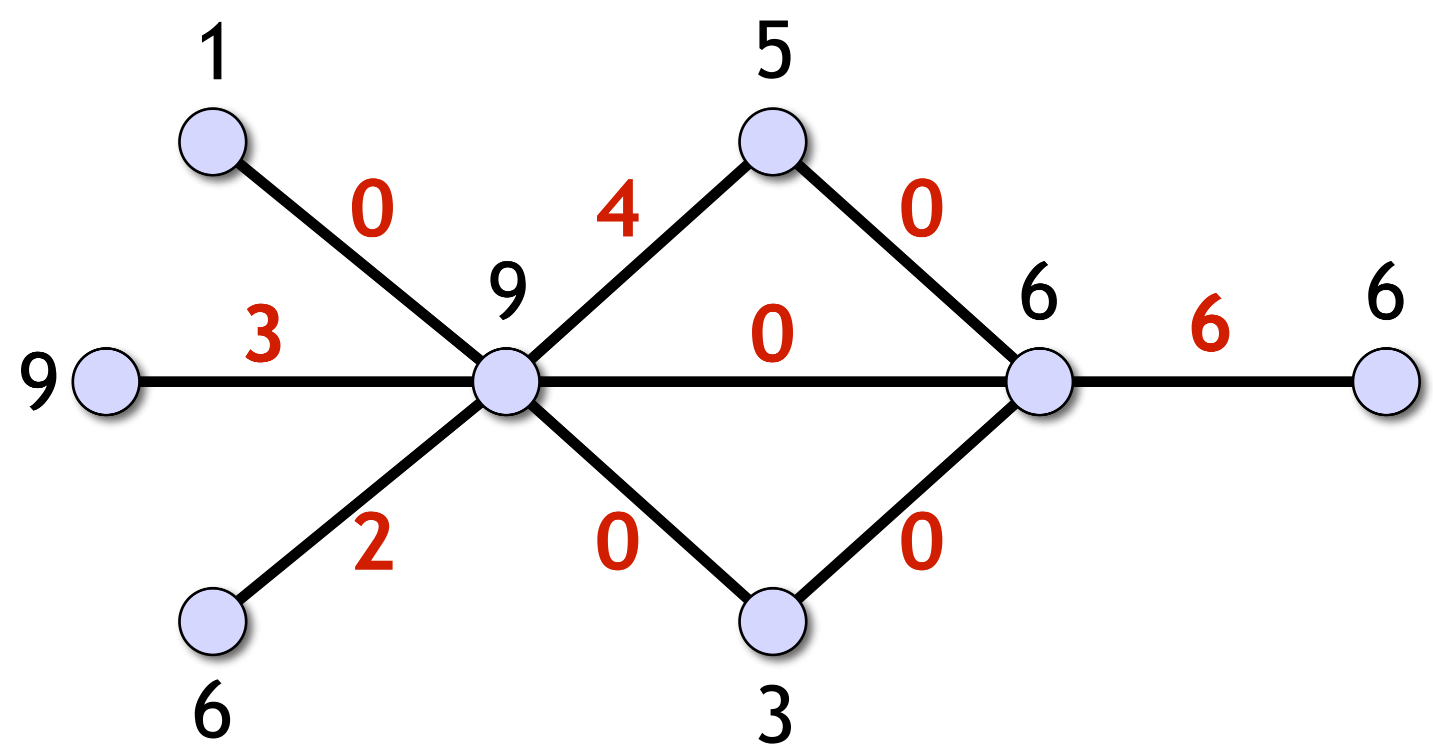
- Convenient to study a more general problem: minimum-weight vertex cover
  - Minimum-cardinality vertex cover: all weights = 1



Notation:
$w(v)$ = weight of v

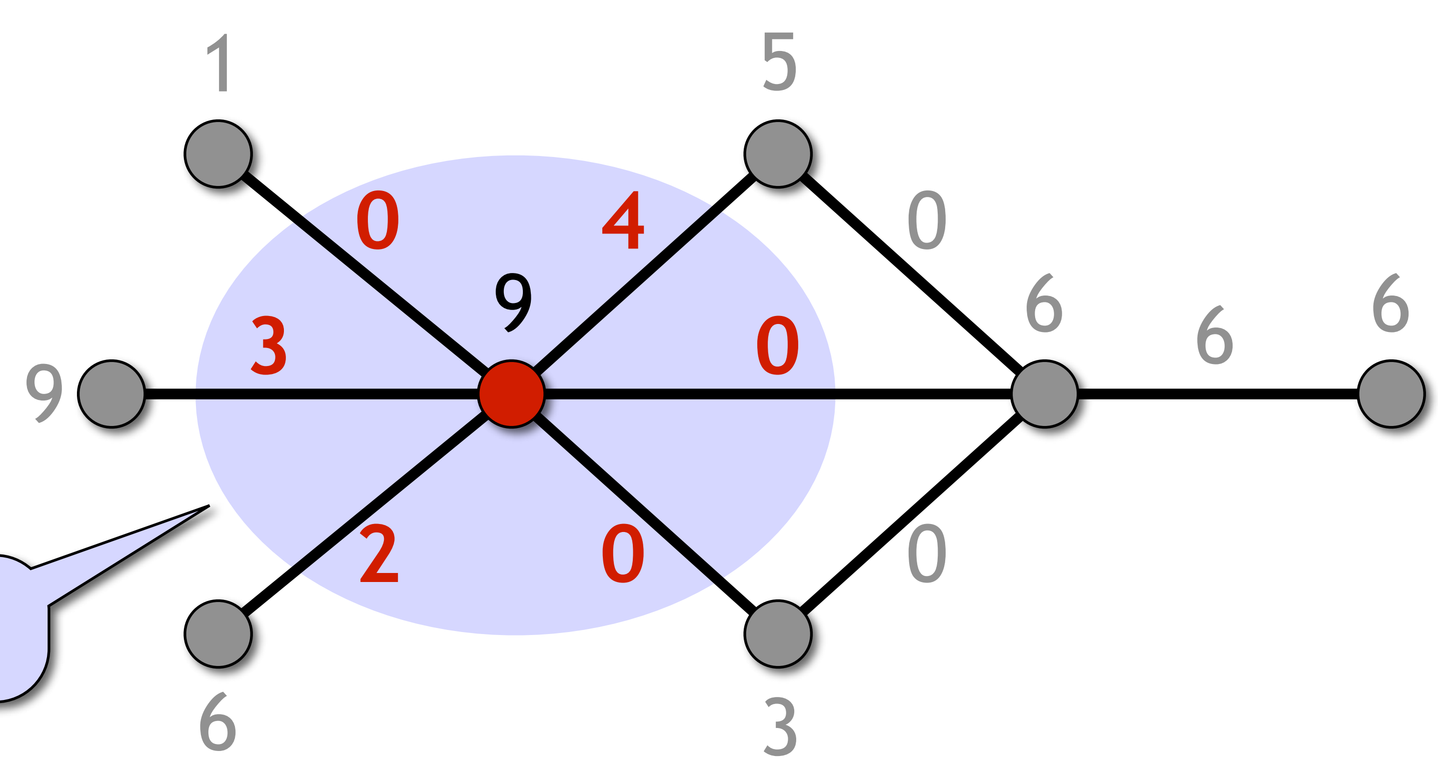
Local 2-approximation algorithm for vertex cover: background

- **Edge packing**: weight $y(e) \geq 0$ for each edge e
  - Packing constraint: for each node v, the total weight of edges incident to v is at most $w(v)$



Local 2-approximation algorithm for vertex cover: background

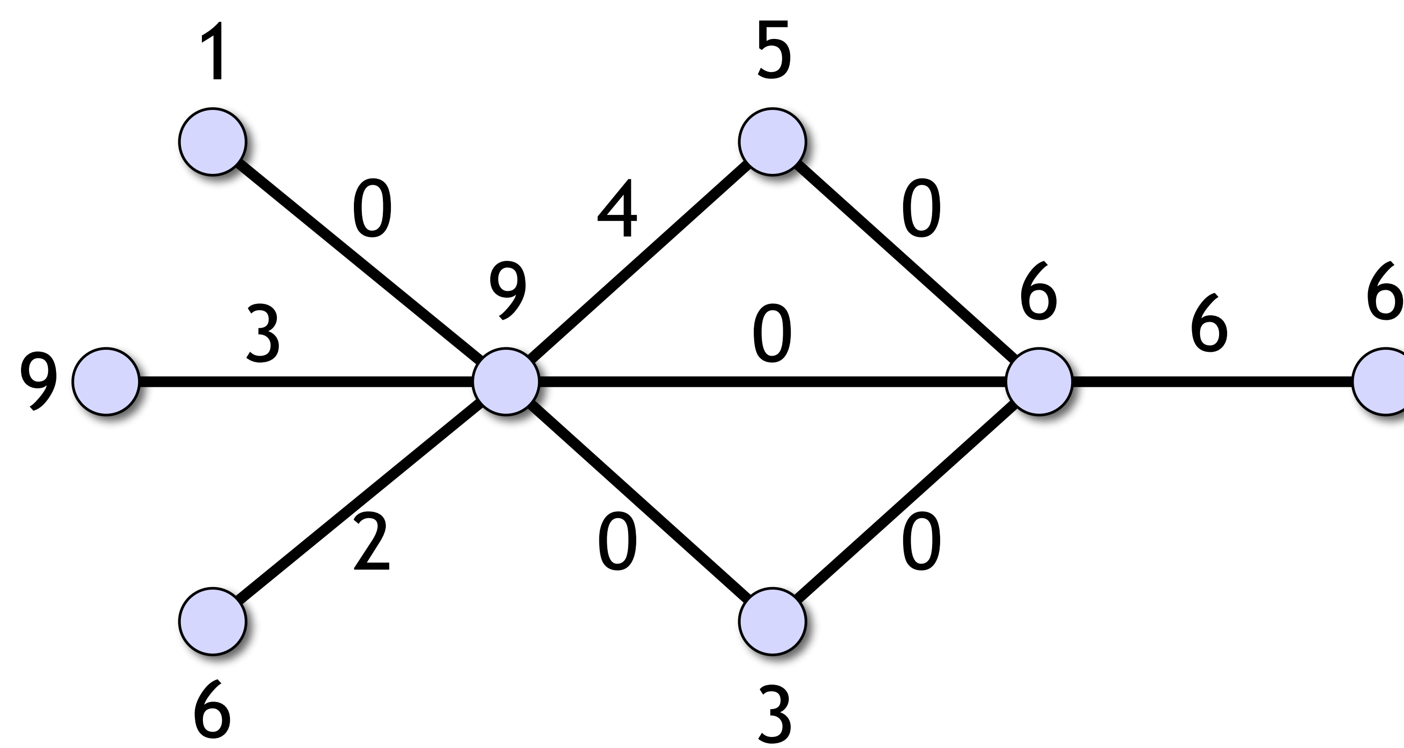
- **Edge packing**: weight $y(e) \geq 0$ for each edge e
  - Packing constraint: for each node v,
    the total weight of edges incident to v is at most $w(v)$



3 + 0 + 4 + 0 + 0 + 2 ≤ 9

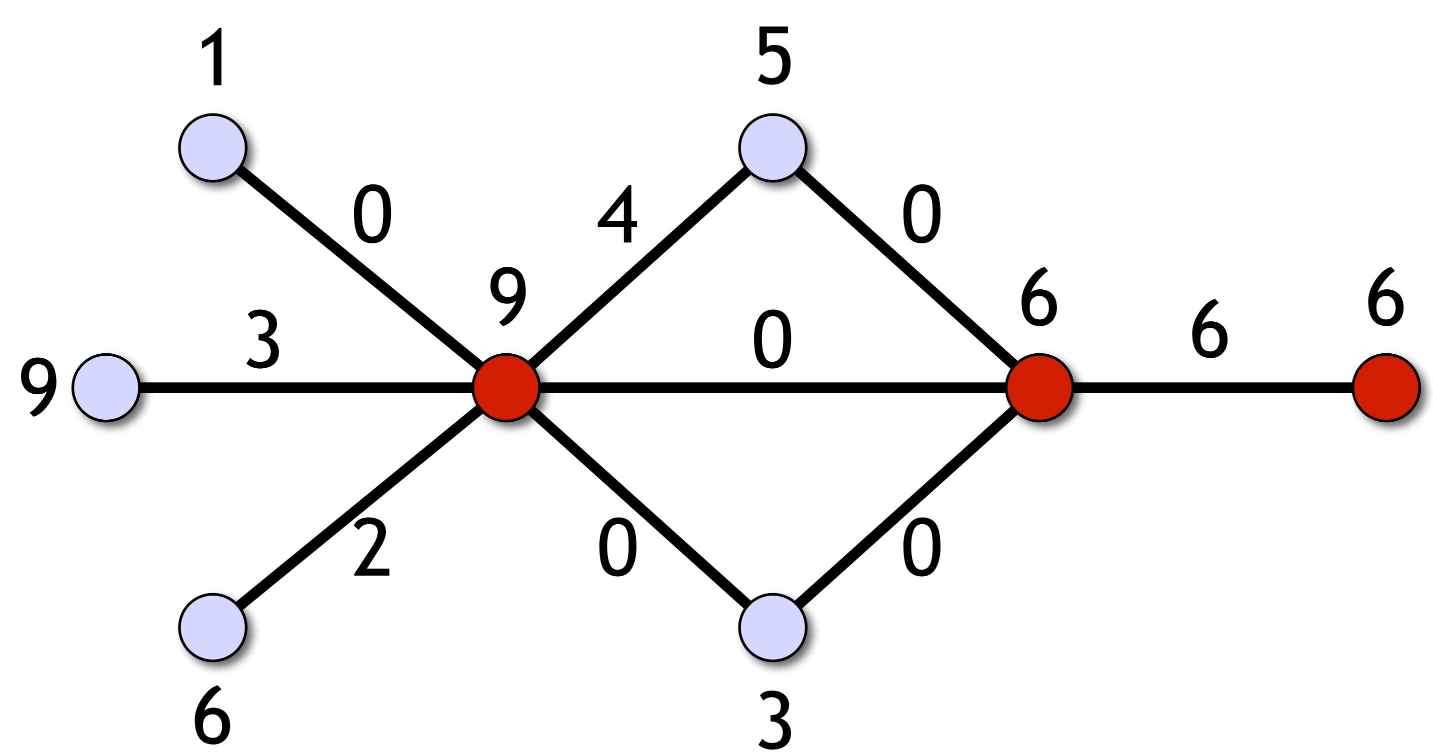
Local 2-approximation algorithm for vertex cover: background

- In linear programming, these are dual problems:
  - minimum-weight (fractional) vertex cover
  - maximum-weight edge packing



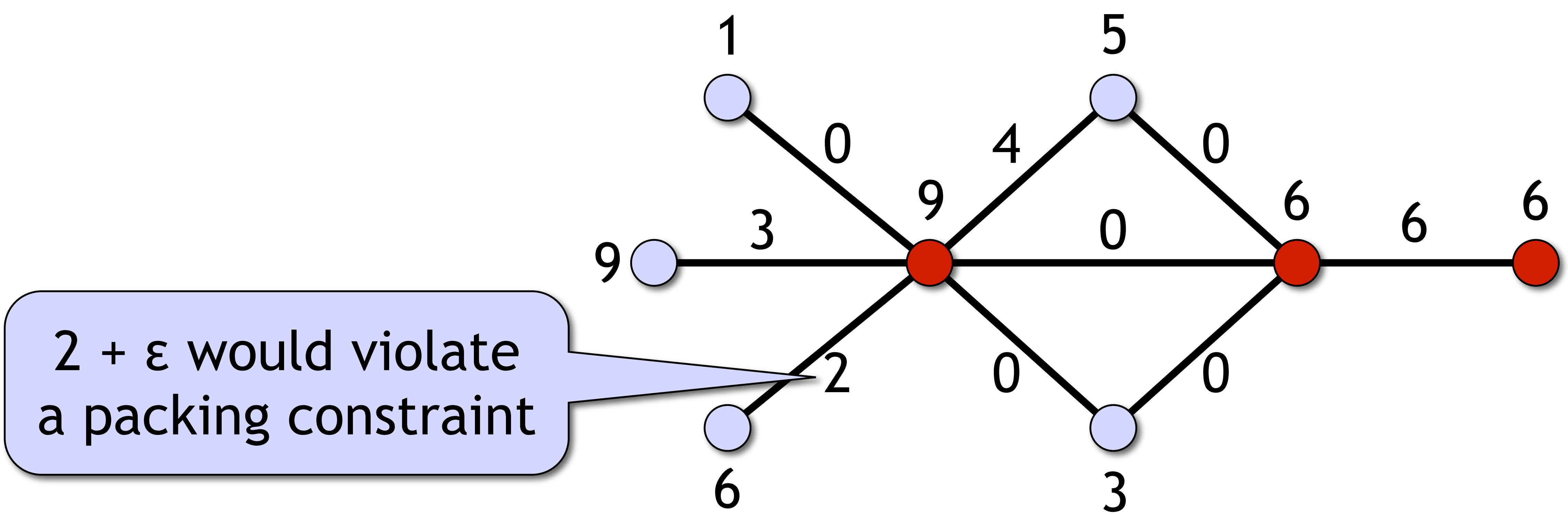
Local 2-approximation algorithm for vertex cover: background

- **Saturated node** *v*: the total weight on edges incident to *v* is equal to *w*(*v*)



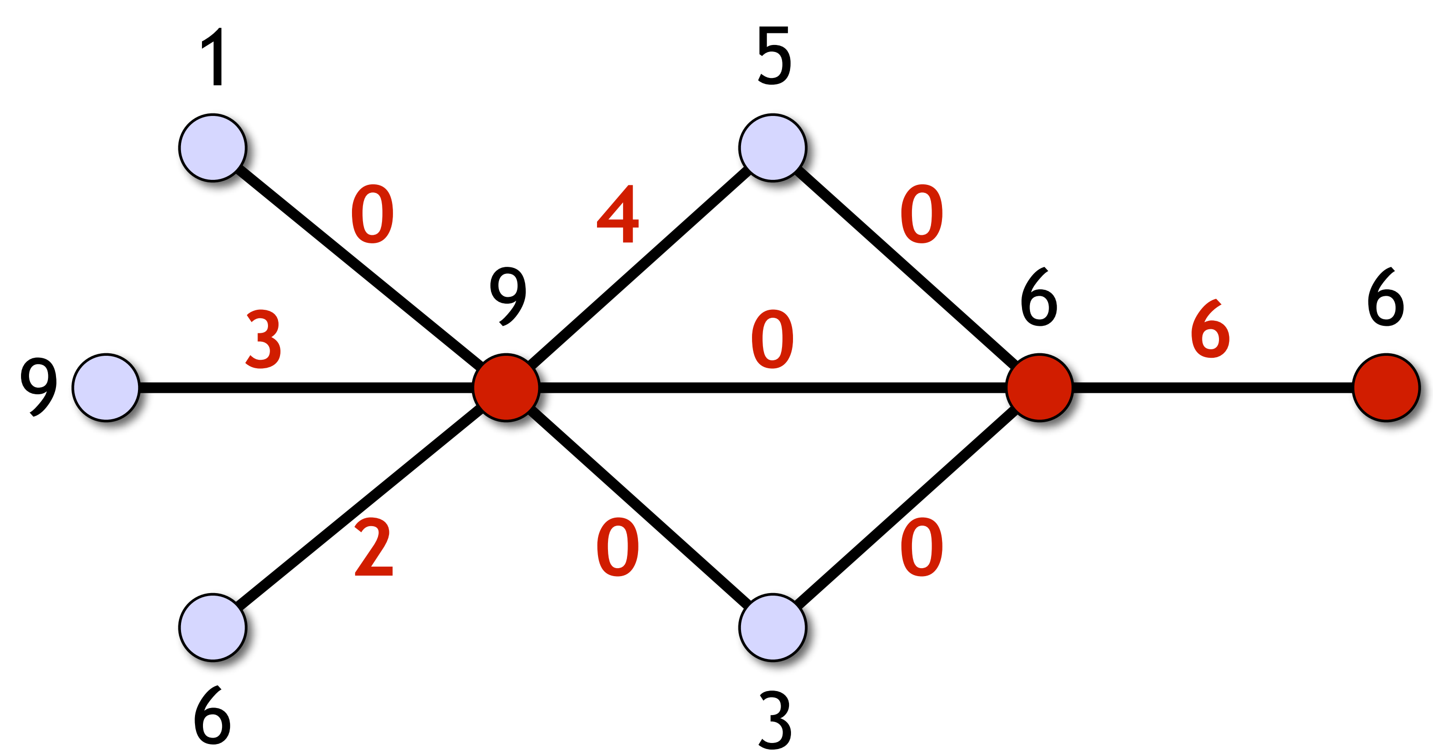
Local 2-approximation algorithm for vertex cover: background

- **Saturated edge** e:
  at least one endpoint of e is saturated
  $\Leftrightarrow$ edge weight $y(e)$ can't be increased



Local 2-approximation algorithm for vertex cover: background

- **Maximal edge packing**: all edges saturated
  $\Leftrightarrow$ none of the edge weights $y(e)$ can be increased
  $\Leftrightarrow$ saturated nodes form a vertex cover



Local 2-approximation algorithm for vertex cover: background

- Minimum-weight vertex cover C^* difficult to find:
  - Centralised setting: NP-hard
  - Distributed setting: integer problem, symmetry-breaking issues
- Maximal edge packing y easy to find:
  - Centralised setting: trivial greedy algorithm
  - Distributed setting: linear problem, no symmetry-breaking issues (?)

Local 2-approximation algorithm for vertex cover: background

- Minimum-weight vertex cover C^* difficult to find
- Maximal edge packing y easy to find?
- Saturated nodes $C(y)$ in y: **2-approximation** of C^*
    - $w(C(y)) \leq 2w(C^*)$
    - Notation: $w(C)$ = total weight of the nodes $v \in C$
    - Proof: LP-duality, relaxed complementary slackness

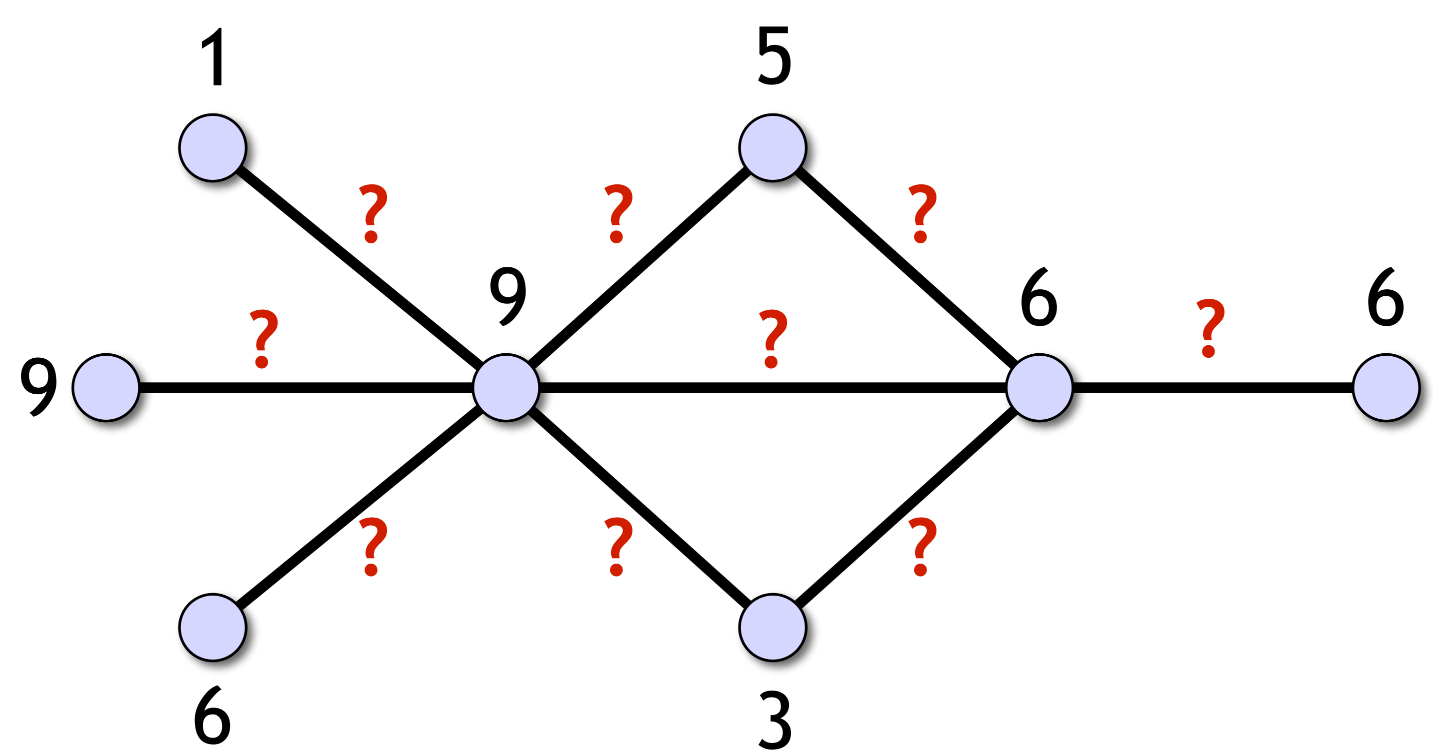
Local 2-approximation algorithm for vertex cover: background

- Minimum-weight vertex cover C^* difficult to find
- Maximal edge packing y easy to find?
- Saturated nodes $C(y)$ in y: **2-approximation** of C^*
  - $w(C(y)) \leq 2w(C^*)$
  - Constant 2: $C(y)$ covers edges at most **twice**, C^* at least **once**
  - Immediate generalisation to hypergraphs

$$w(C(y)) = \sum_{v \in C(y)} y[v] = \sum_{e \in E} y(e)\,|e \cap C(y)| \leq 2 \sum_{e \in E} y(e)\,|e \cap C^*| = 2 \sum_{v \in C^*} y[v] \leq 2w(C^*)$$

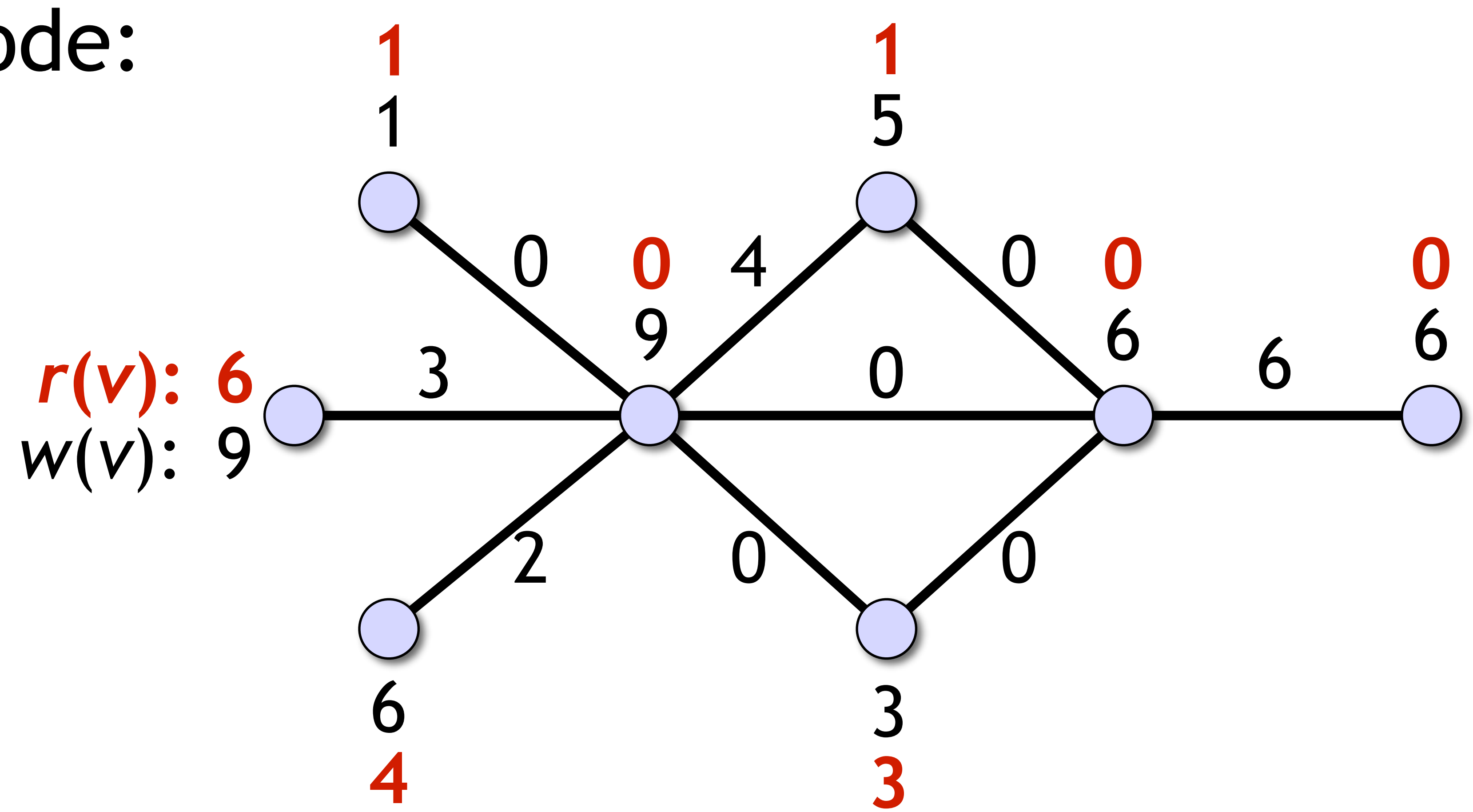
Local 2-approximation algorithm for vertex cover

- Finding a maximal edge packing?
  - Basic idea from Khuller et al. (1994) and Papadimitriou and Yannakakis (1993)



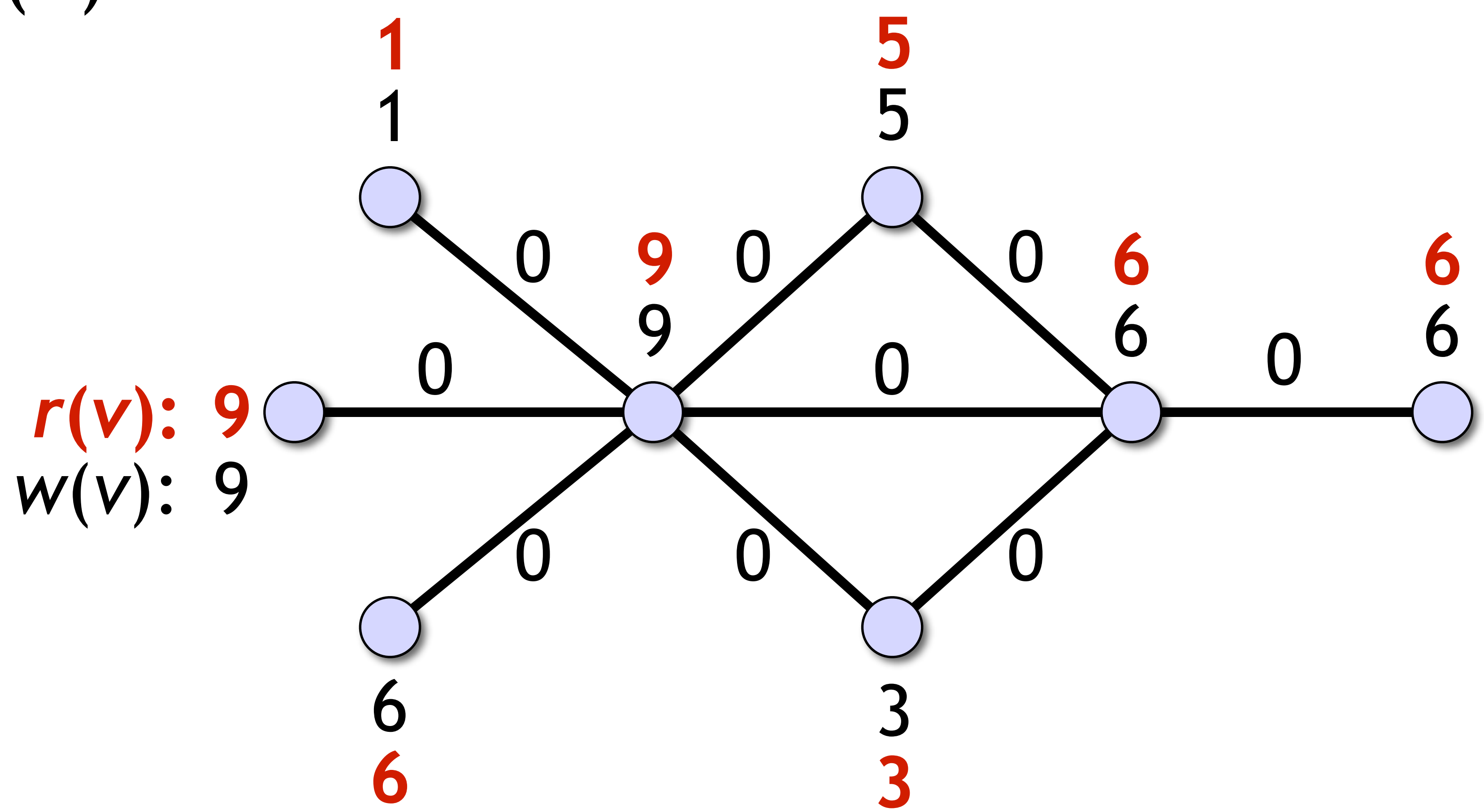
Local 2-approximation algorithm for vertex cover: basic idea

- $y[v]$ = total weight of edges incident to node v
- **Residual capacity** of node v: $r(v) = w(v) - y[v]$
- Saturated node: $r(v) = 0$



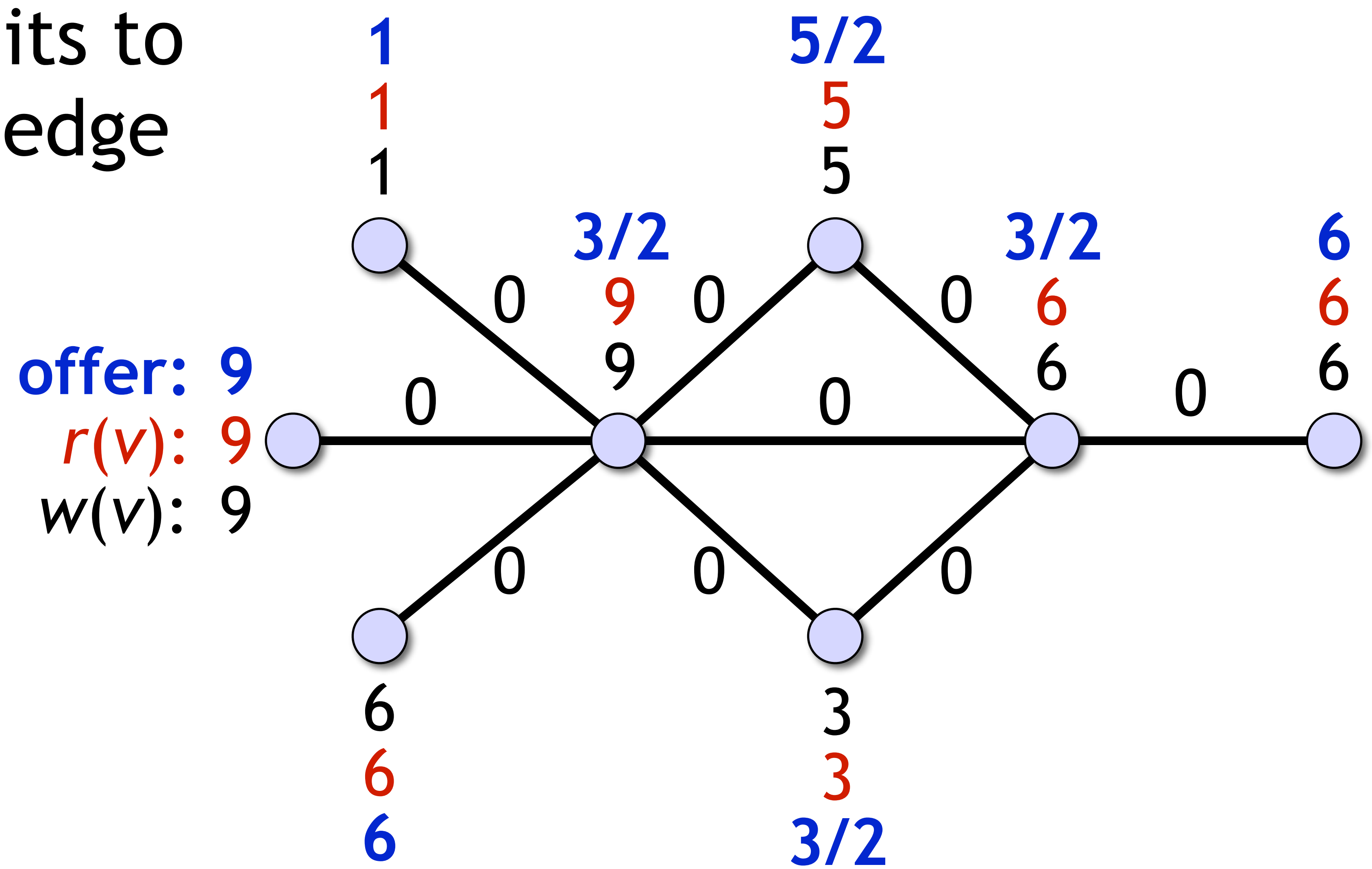
Local 2-approximation algorithm for vertex cover: basic idea

Start with a trivial edge packing $y(e) = 0$



Local 2-approximation algorithm for vertex cover: basic idea

Each node *v* **offers** $r(v)/\deg(v)$ units to each incident edge

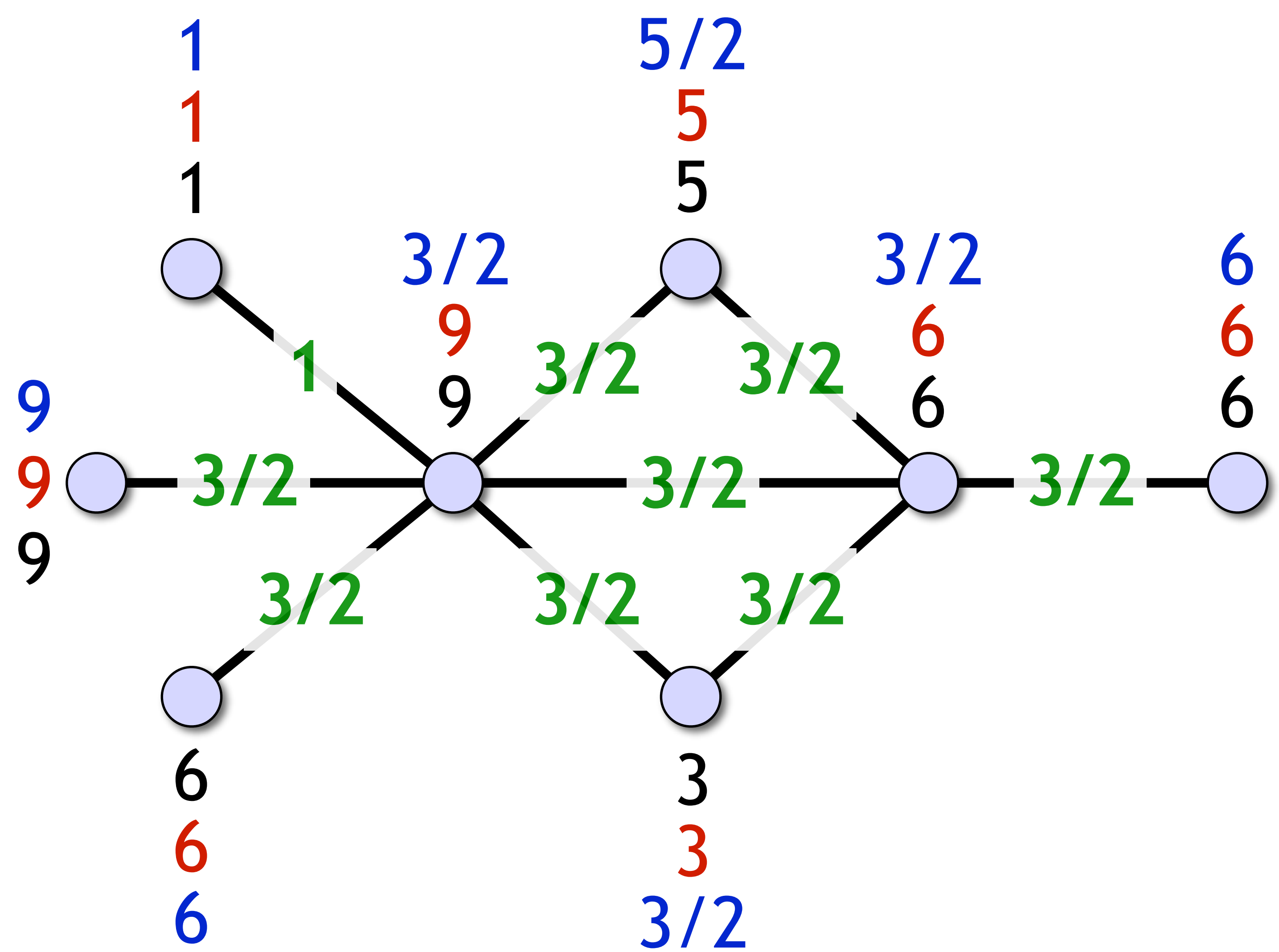


Local 2-approximation algorithm for vertex cover: basic idea

Each edge **accepts** the smallest of the 2 offers it received

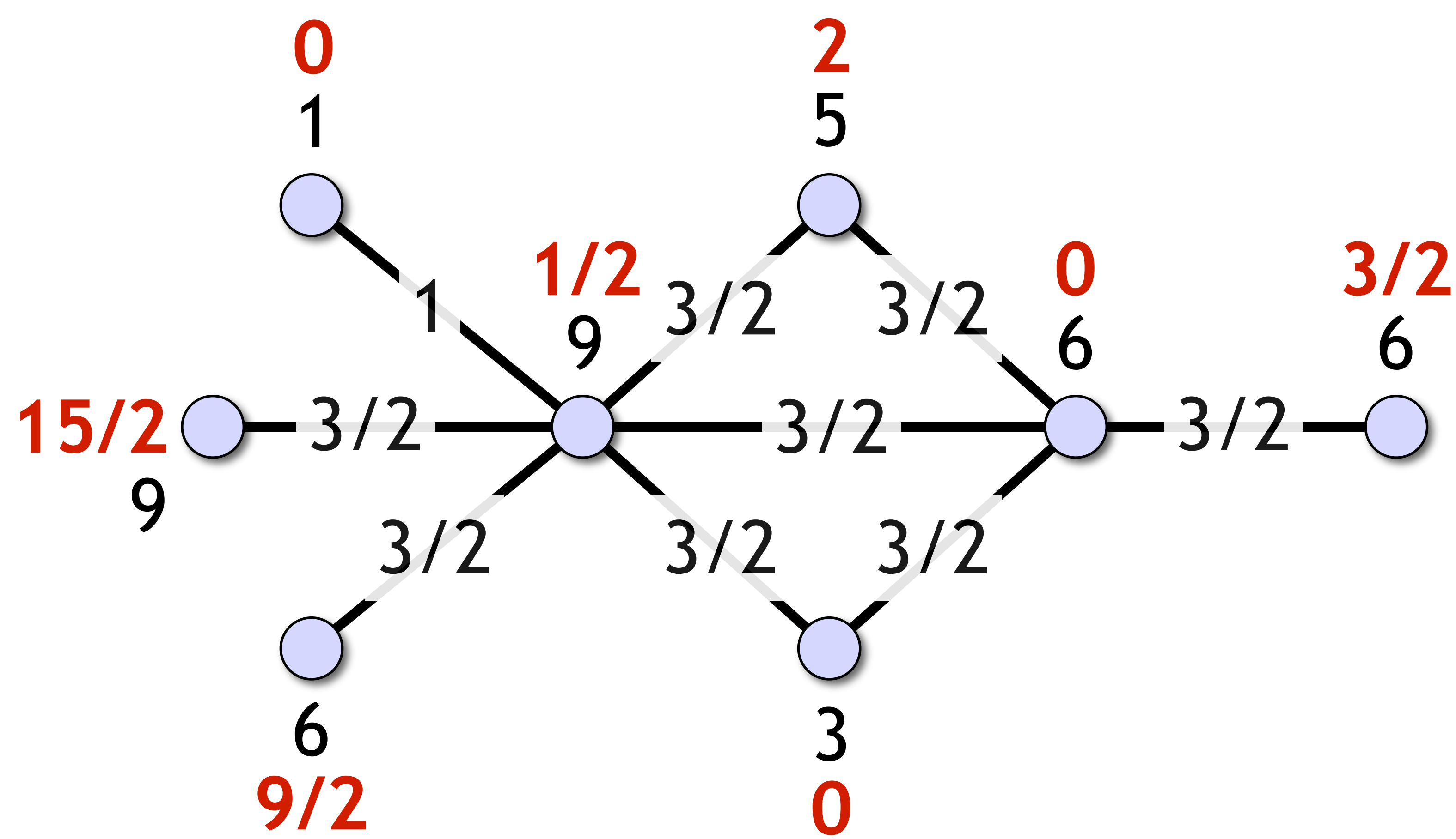
Increase $y(e)$ by this amount

- Safe, can't violate packing constraints



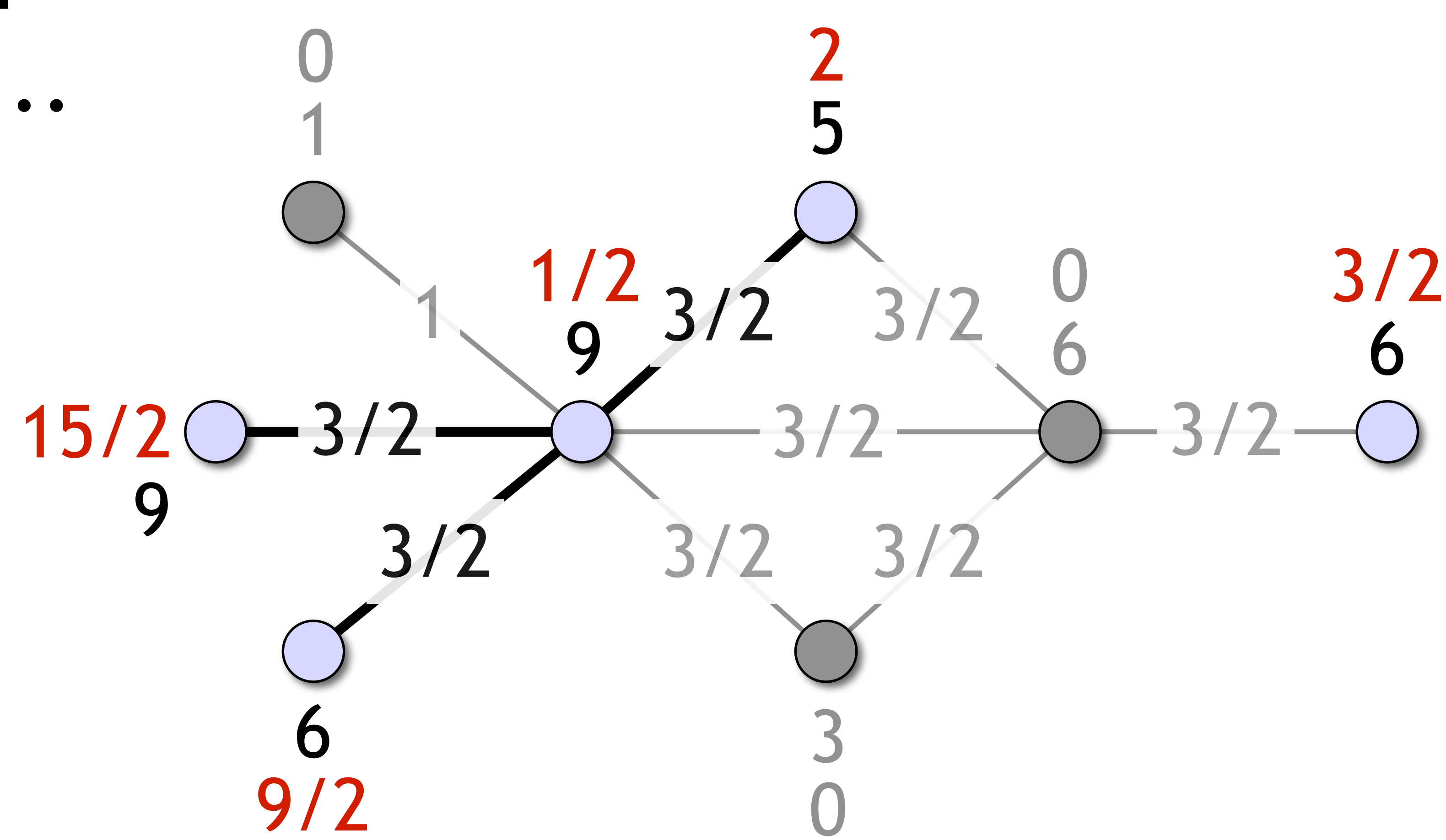
Local 2-approximation algorithm for vertex cover: basic idea

Update **residuals**...



Local 2-approximation algorithm for vertex cover: basic idea

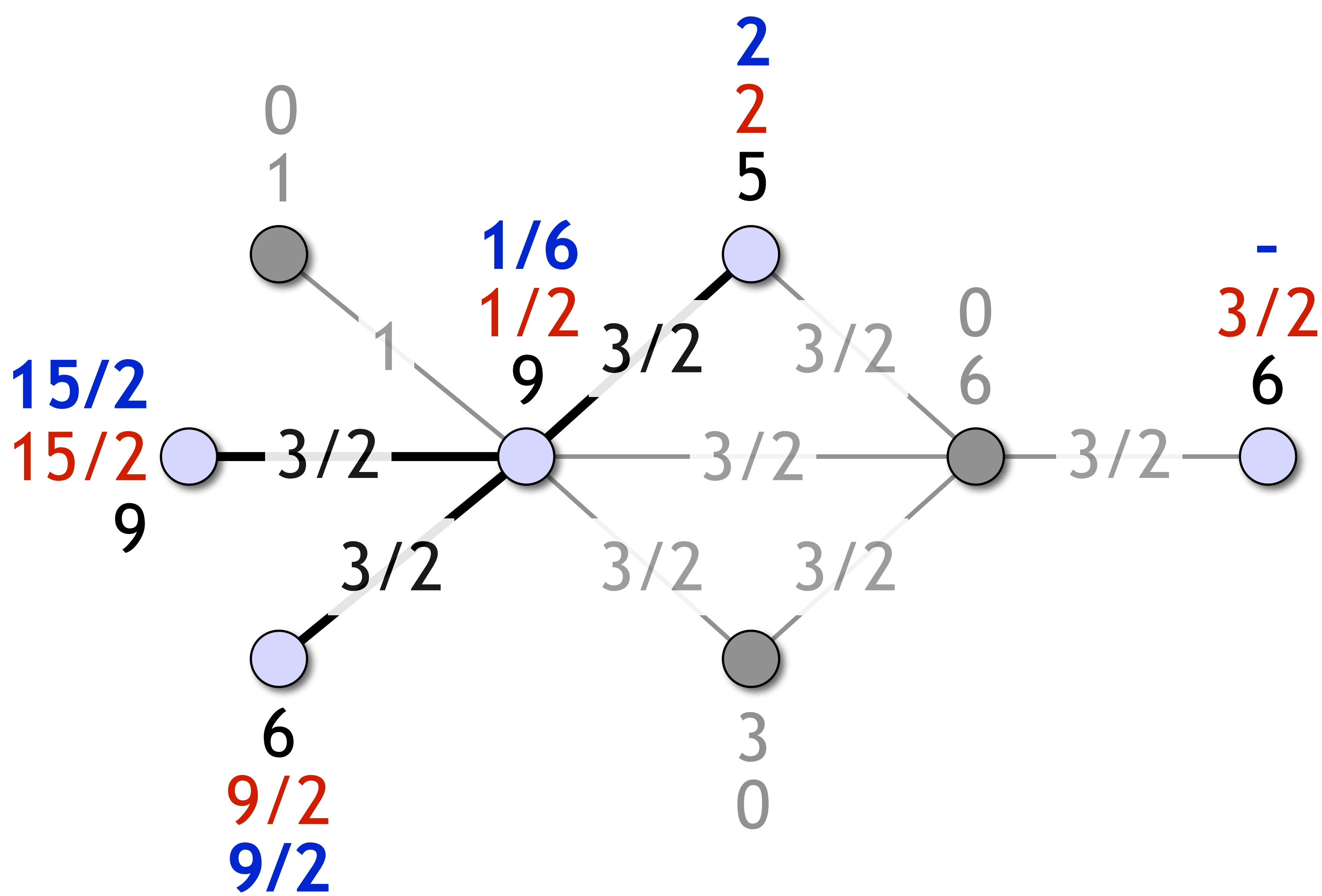
Update residuals, discard saturated nodes and edges...



Local 2-approximation algorithm for vertex cover: basic idea

Update residuals, discard saturated nodes and edges, repeat...

Offers...

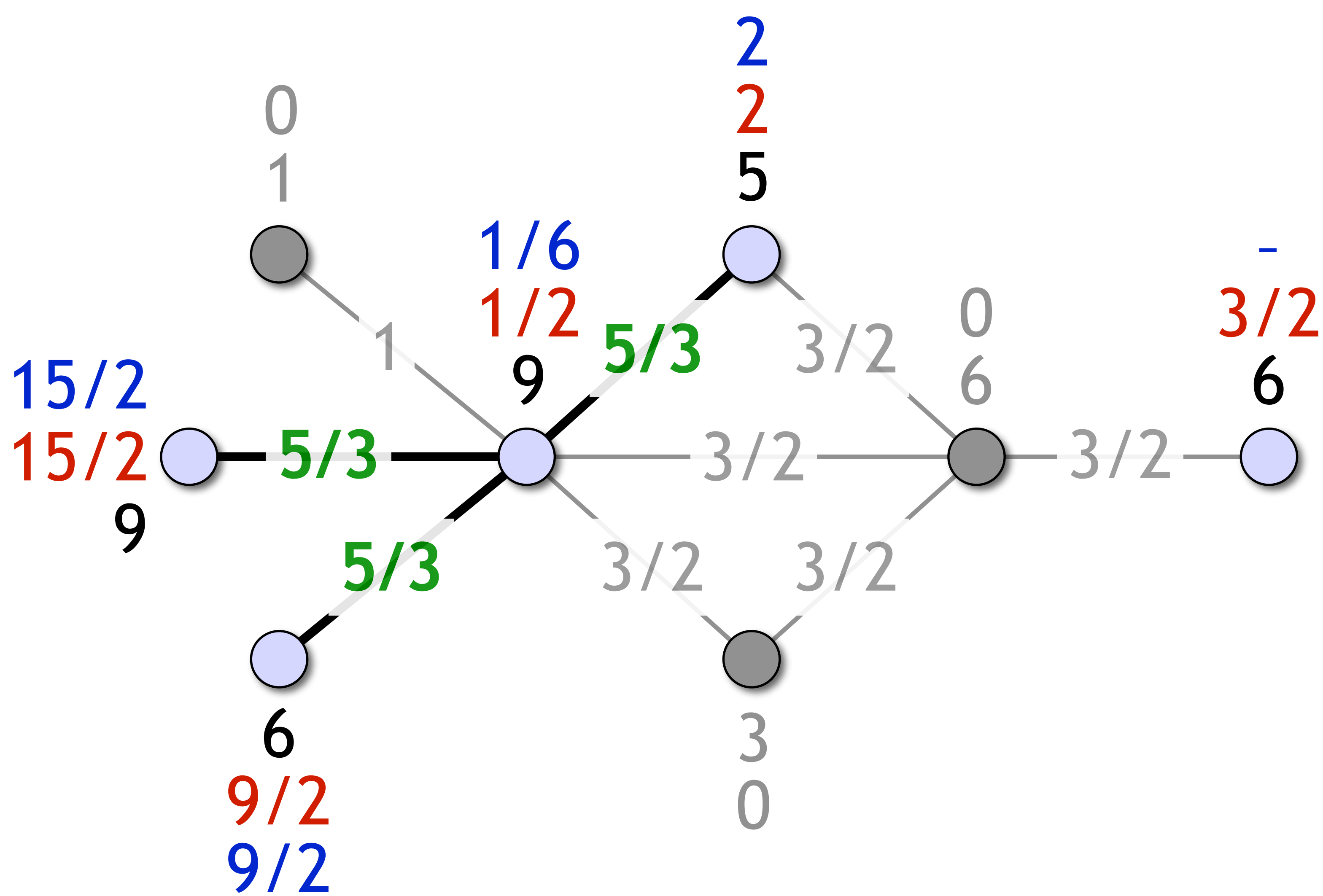


Local 2-approximation algorithm for vertex cover: basic idea

Update residuals, discard saturated nodes and edges, repeat...

Offers...

Increase weights...



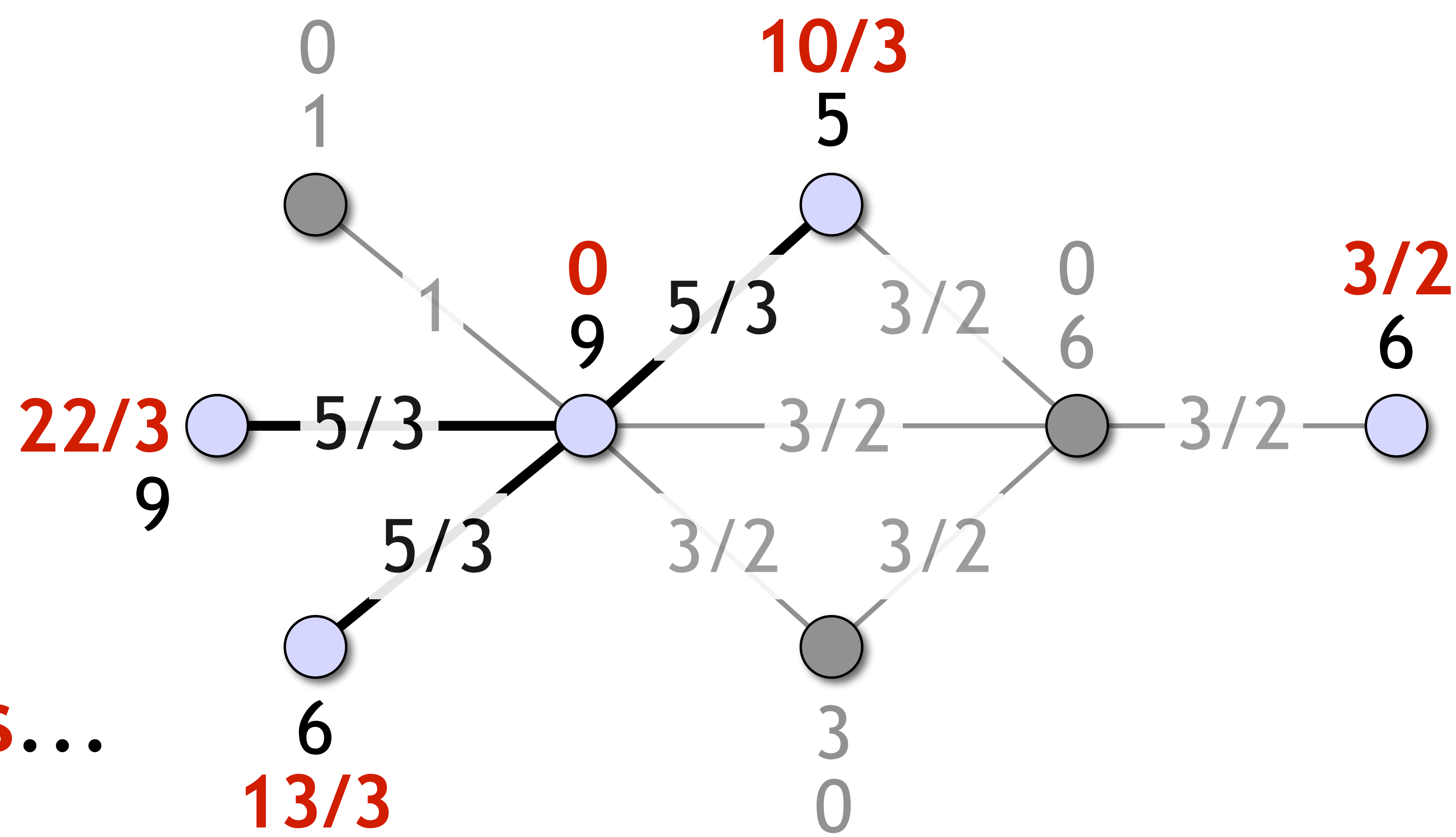
Local 2-approximation algorithm for vertex cover: basic idea

Update residuals, discard saturated nodes and edges, repeat...

Offers...

Increase weights...

Update residuals...



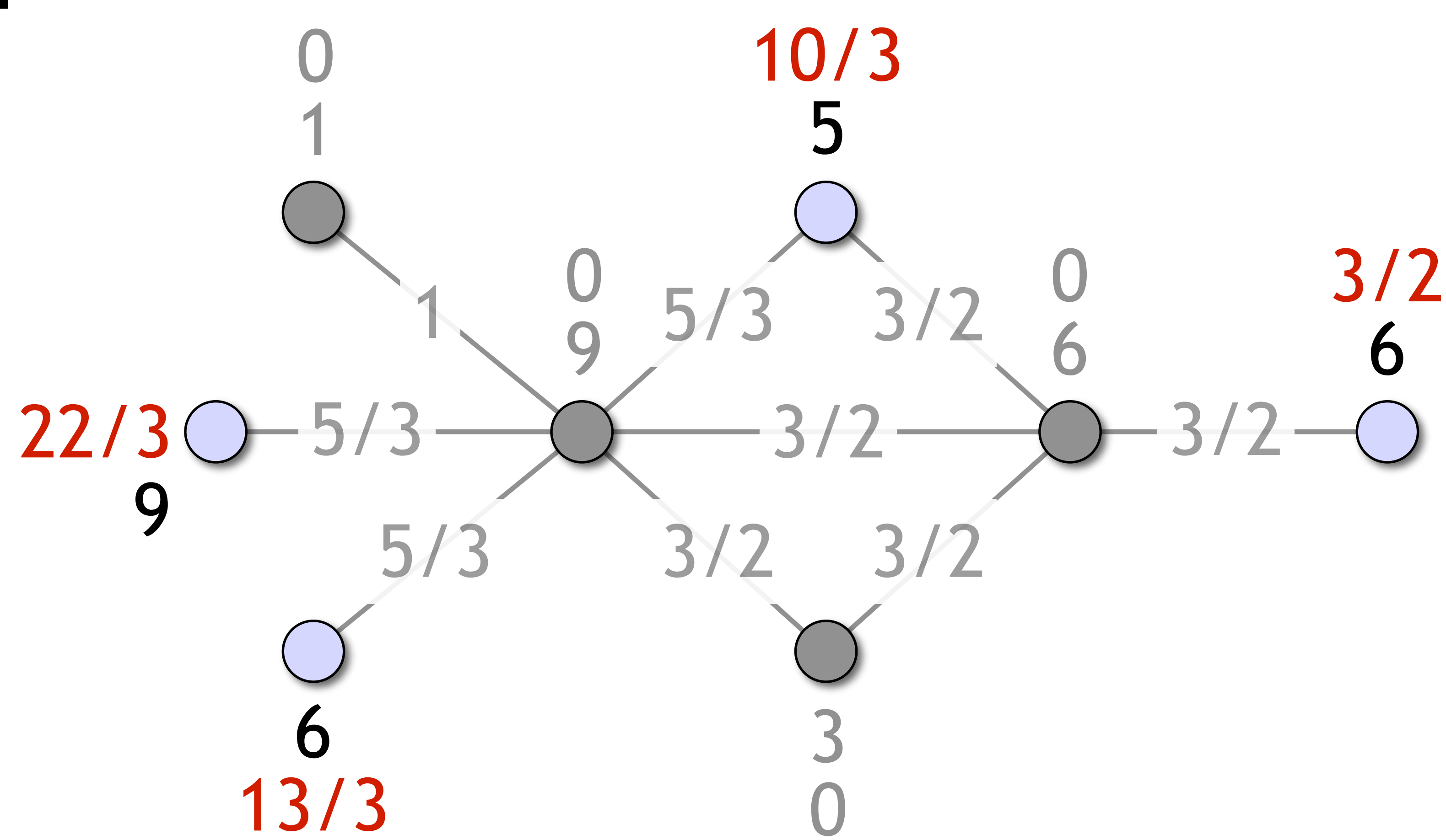
Local 2-approximation algorithm for vertex cover: basic idea

Update residuals, discard saturated nodes and edges, repeat...

Offers...

Increase weights...

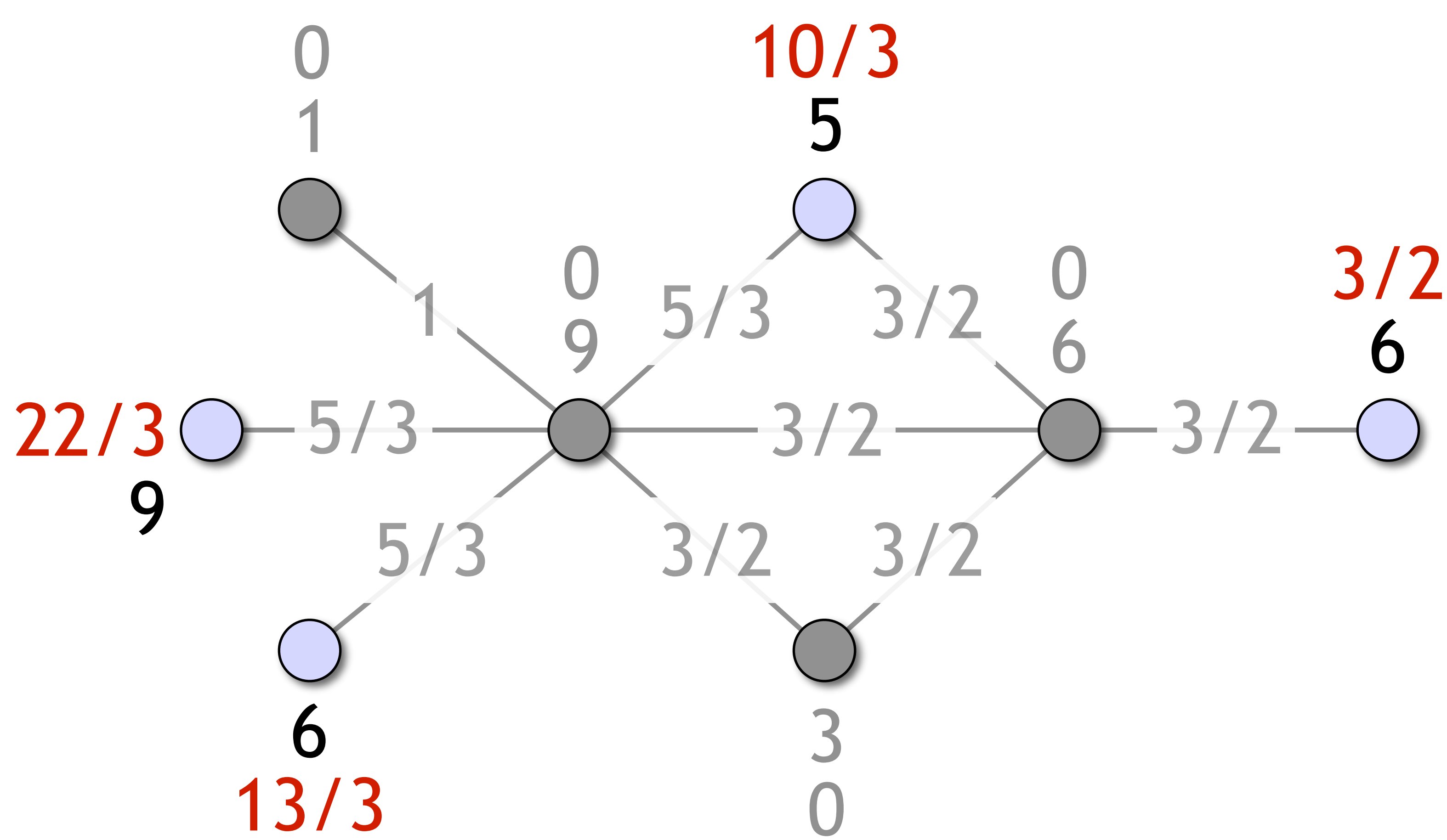
Update residuals and graph, etc.



Local 2-approximation algorithm for vertex cover: basic idea

This is a simple deterministic distributed algorithm

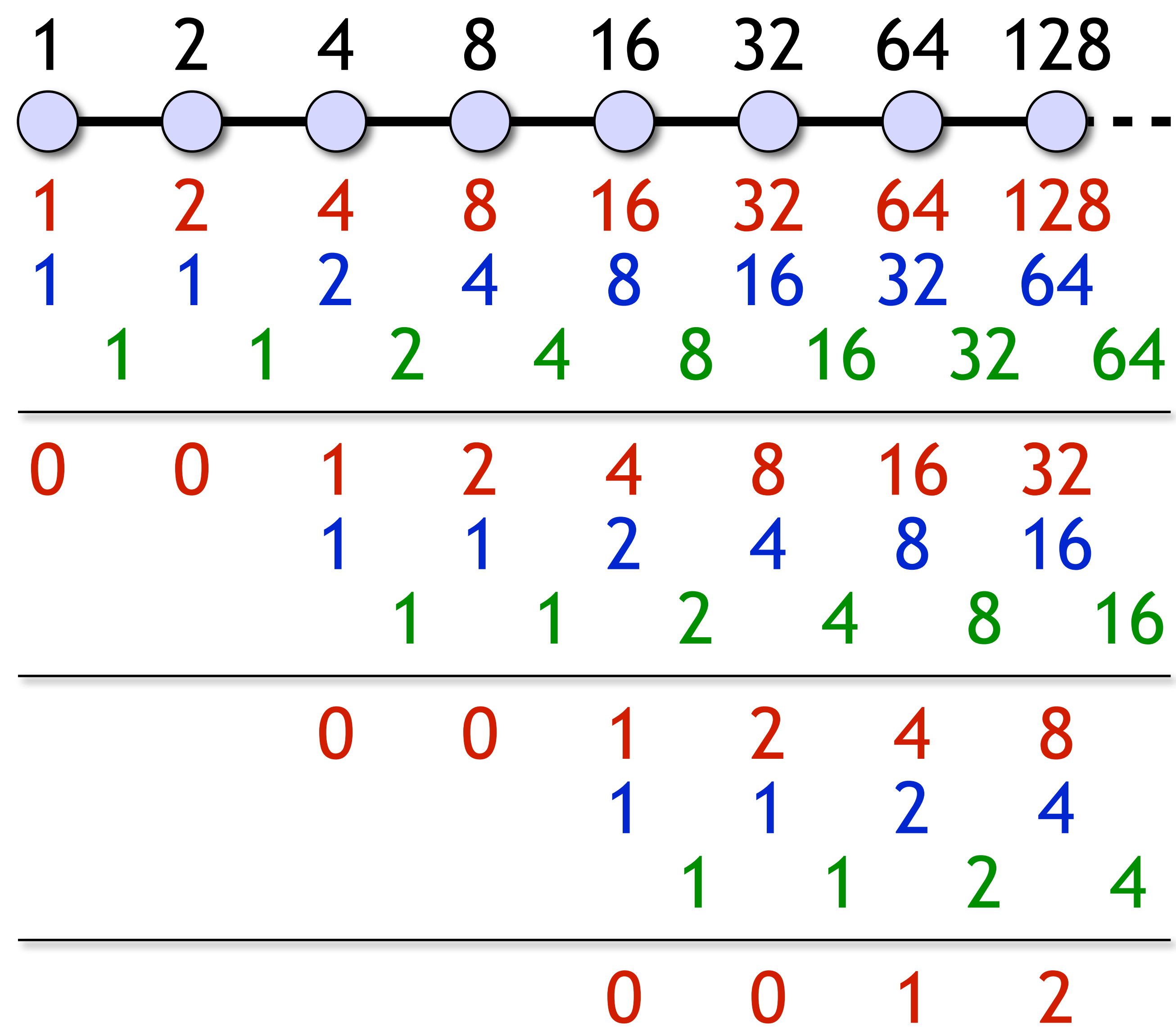
We are making some progress towards finding a maximal edge packing — but...



Local 2-approximation algorithm for vertex cover: basic idea

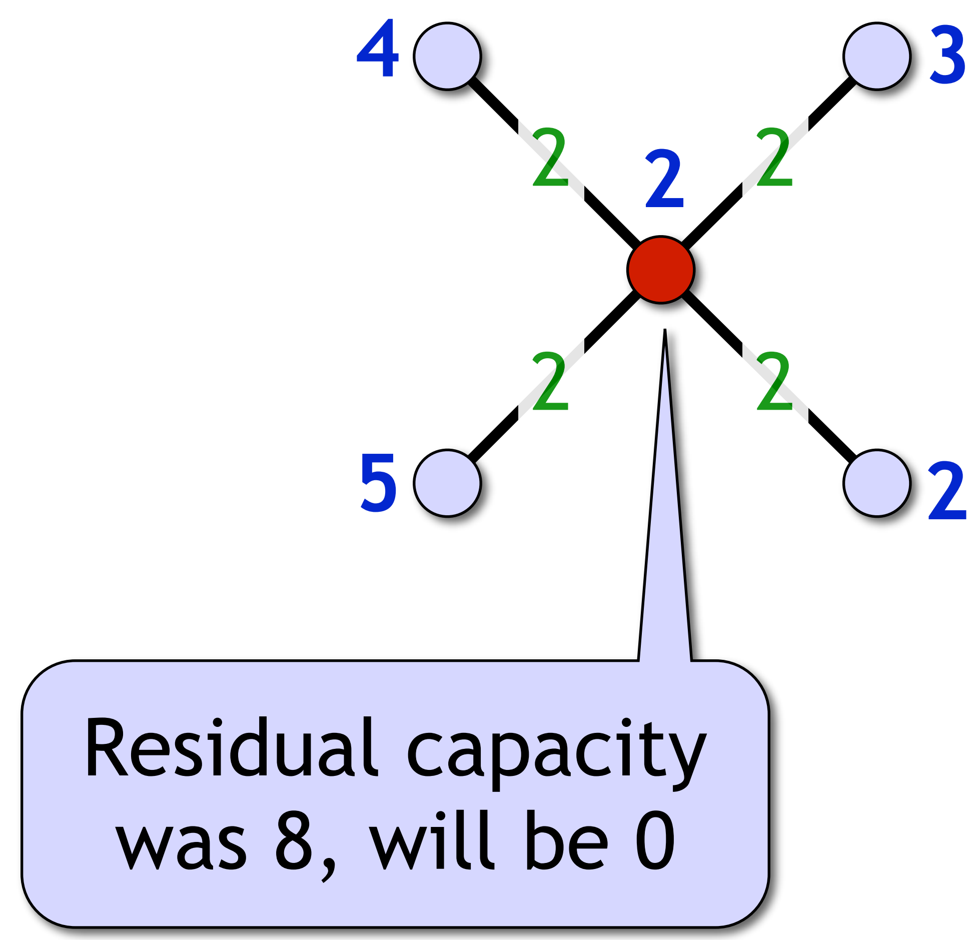
This is a simple deterministic distributed algorithm

We are making some progress towards finding a maximal edge packing — but this is **too slow**!



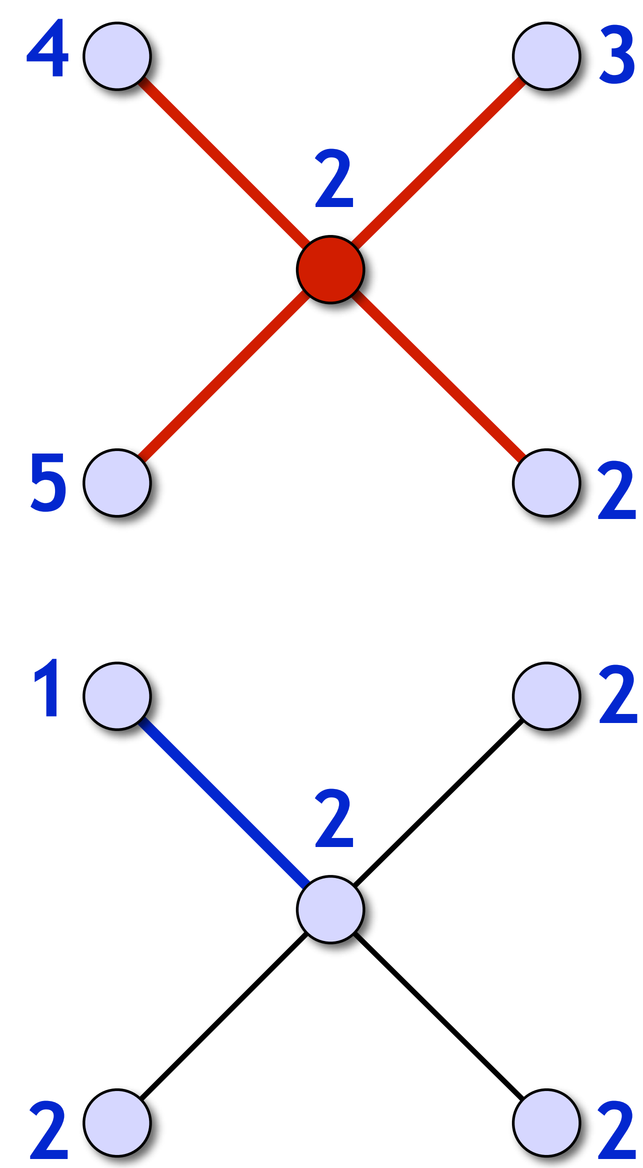
Local 2-approximation algorithm for vertex cover

- Offer is a local minimum:
  - Node will be **saturated**
  - And all edges incident to it will be saturated as well



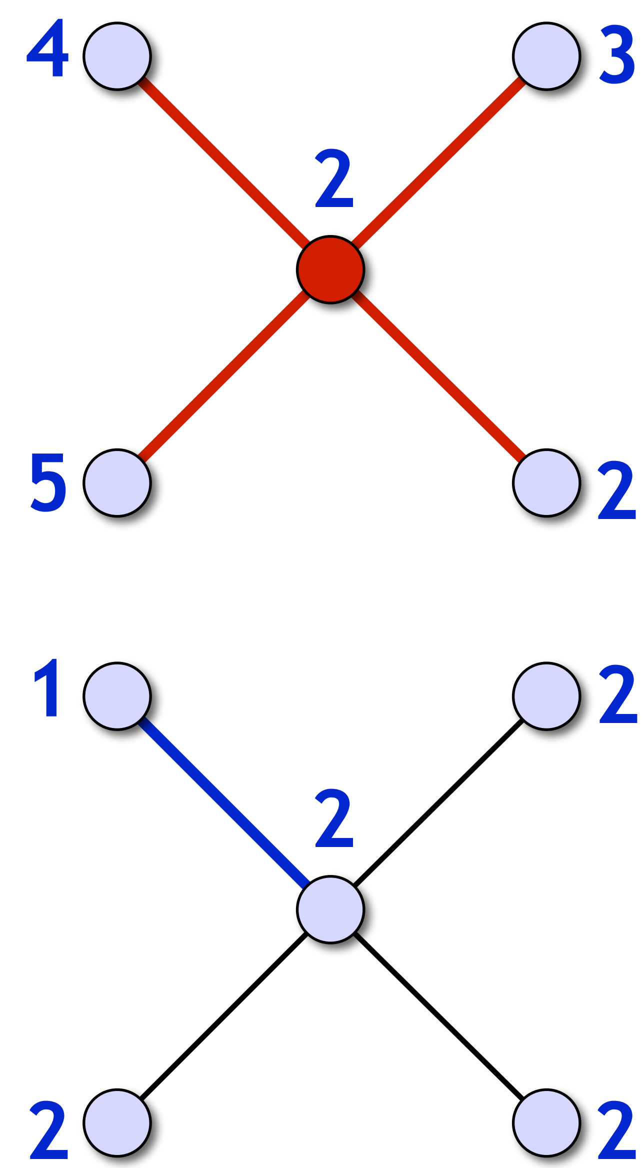
Local 2-approximation algorithm for vertex cover

- Offer is a local minimum:
  - Node will be **saturated**
- Otherwise there is a neighbour with a different offer:
  - Interpret the offer sequences as colours
  - Nodes *u* and *v* have different colours: $\{u, v\}$ is **multicoloured**



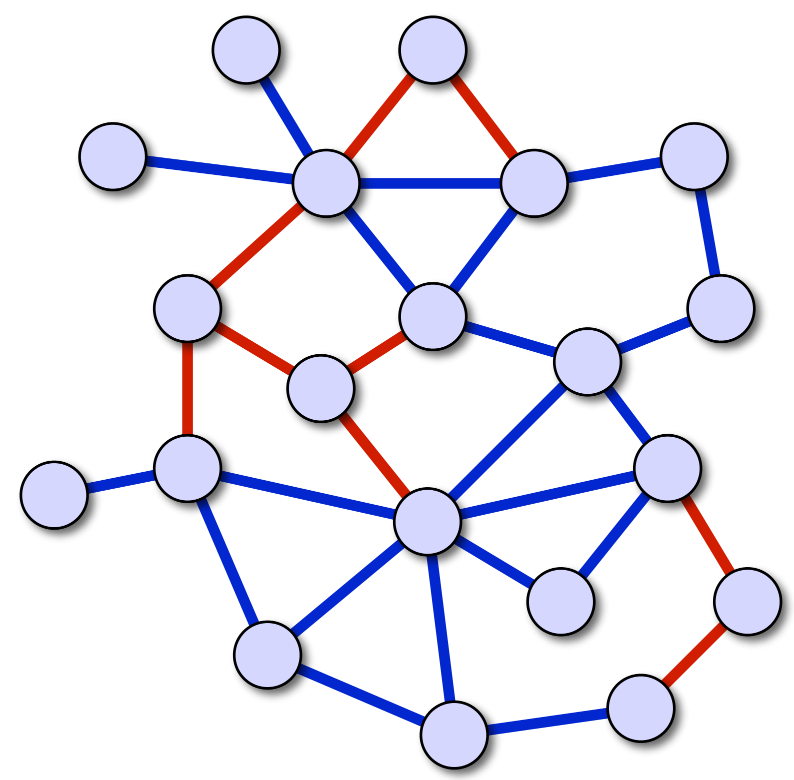
Local 2-approximation algorithm for vertex cover

- Progress guaranteed:
  - On each iteration, for each node, at least one incident edge becomes **saturated** or **multicoloured**
  - Such edges are be discarded; maximum degree Δ decreases by at least one
  - Hence in Δ rounds all edges are saturated or multicoloured



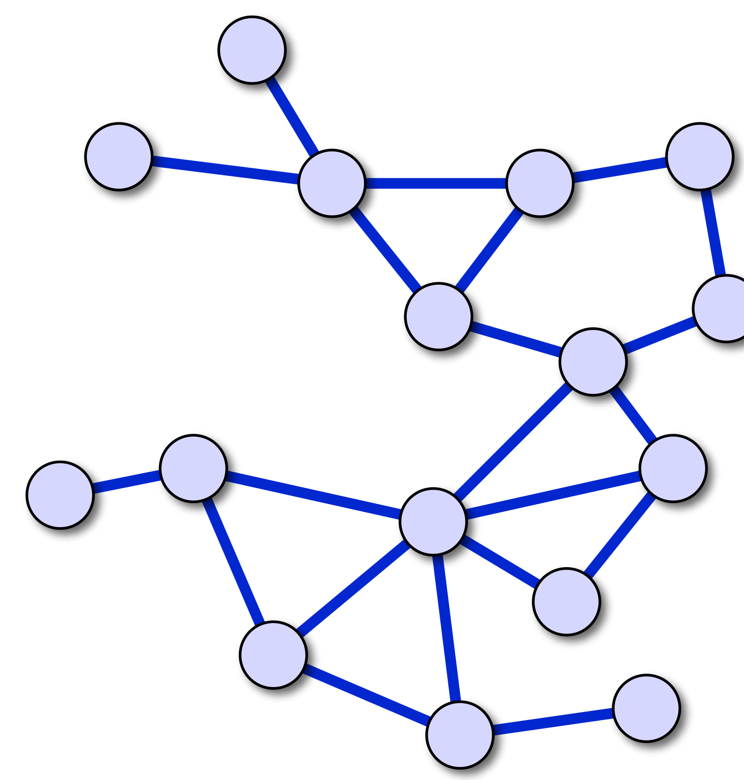
Local 2-approximation algorithm for vertex cover

- In Δ rounds all edges are **saturated** or **multicoloured**
  - Saturated edges are good — we're trying to construct a maximal edge packing
  - Why are the multicoloured edges useful?



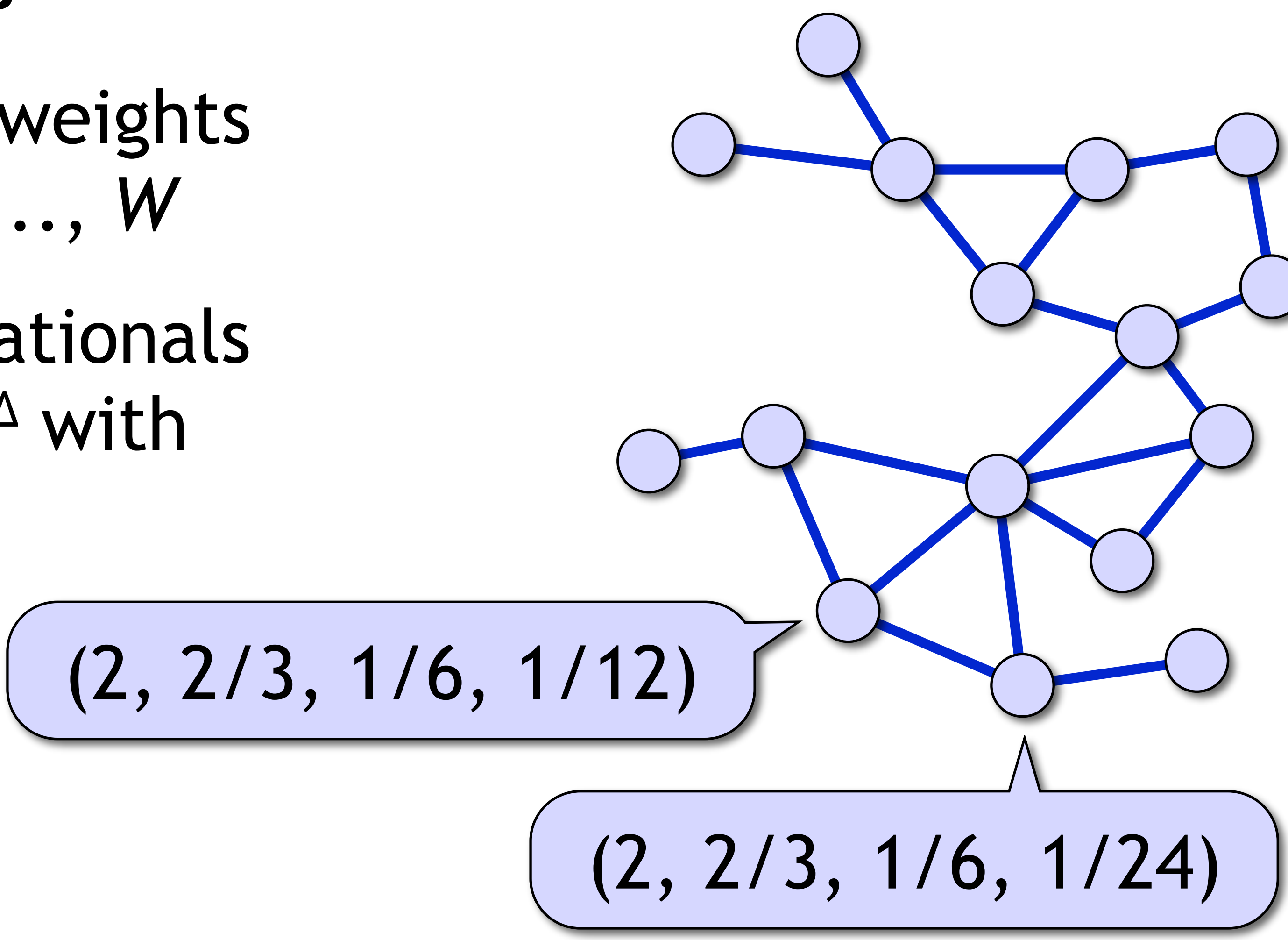
Local 2-approximation algorithm for vertex cover

- In Δ rounds all edges are **saturated** or **multicoloured**
  - Saturated edges are good — we're trying to construct a maximal edge packing
  - Why are the multicoloured edges useful?
  - Let's focus on unsaturated nodes and edges



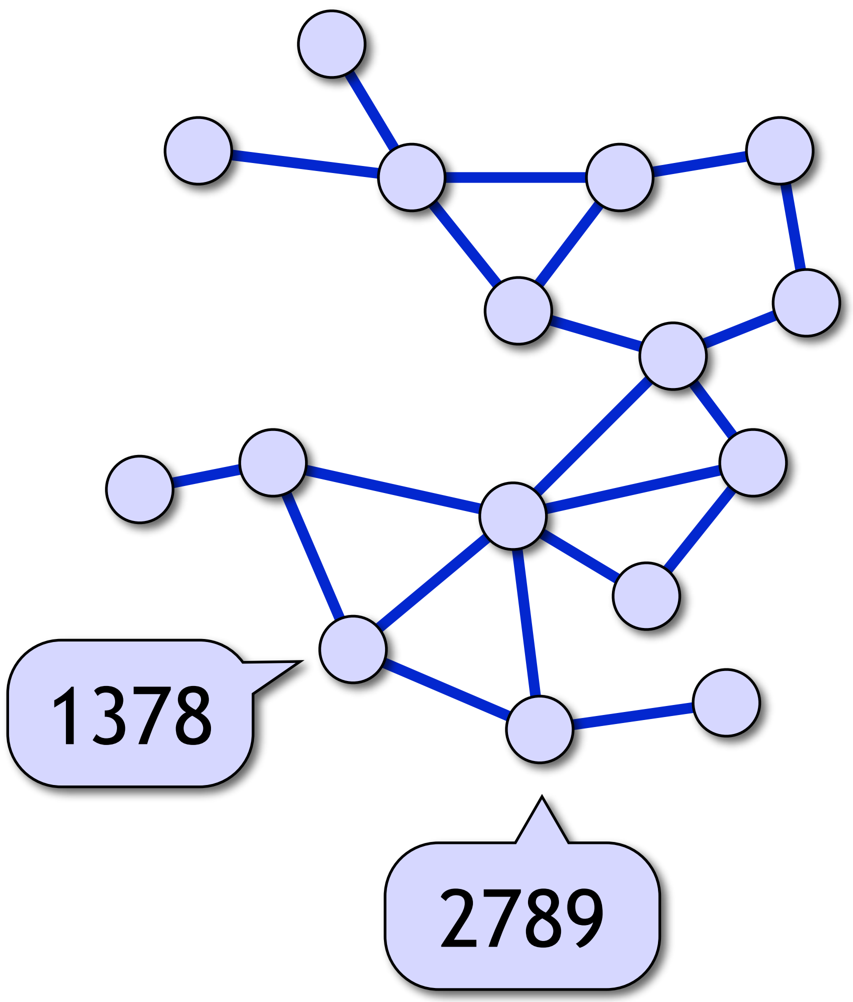
Local 2-approximation algorithm for vertex cover: multicoloured edges

- Colours are sequences of Δ rational numbers
  - Assume that node weights are integers 1, 2, ..., W
  - Then colours are rationals of the form $q/(\Delta!)^\Delta$ with $q \in \{1, 2, ..., W\}$



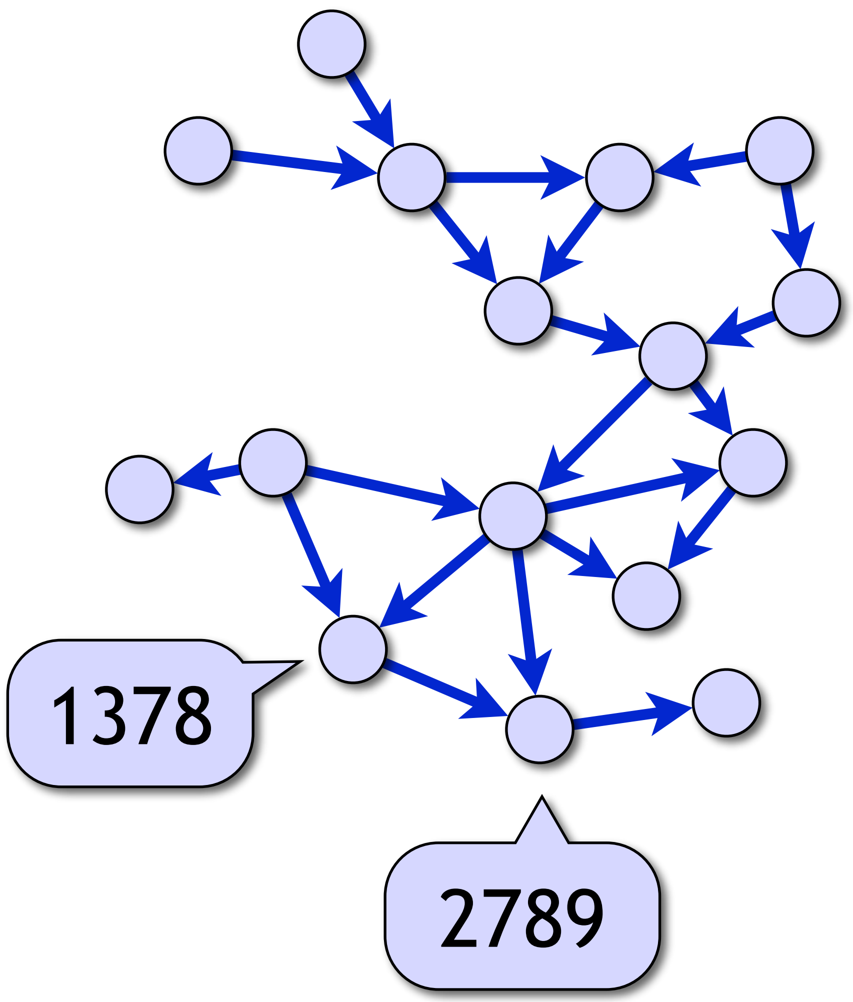
Local 2-approximation algorithm for vertex cover: multicoloured edges

- Colours are sequences of Δ rational numbers
  - Assume that node weights are integers 1, 2, ..., W
  - Then colours are rationals of the form $q/(\Delta!)^\Delta$ with $q \in \{1, 2, \ldots, W\}$
  - $k = (W(\Delta!)^\Delta)^\Delta$ possible colours, replace with integers 1, 2, ..., k



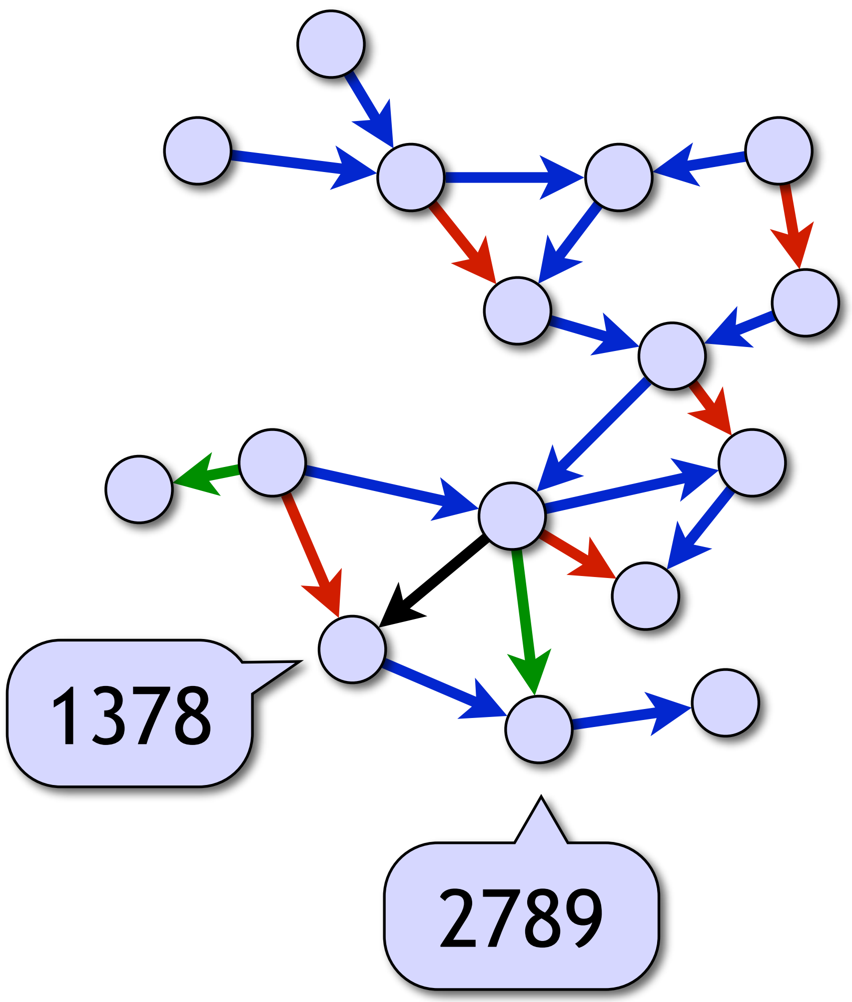
Local 2-approximation algorithm for vertex cover: multicoloured edges

- We have a proper *k*-colouring of the unsaturated subgraph
- Orient from lower to higher colour (acyclic directed graph)



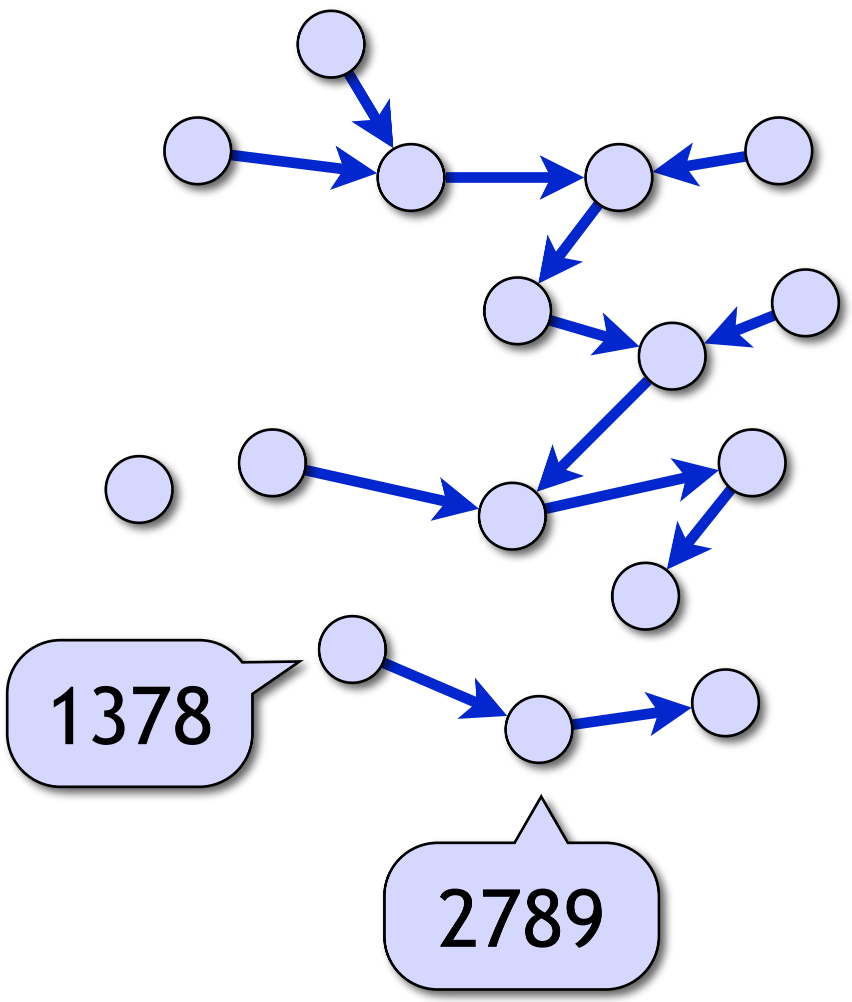
Local 2-approximation algorithm for vertex cover: multicoloured edges

- We have a proper k-colouring of the unsaturated subgraph
- Orient from lower to higher colour (acyclic directed graph)
- Partition in Δ forests
  - Each node assigns its outgoing edges to different forests



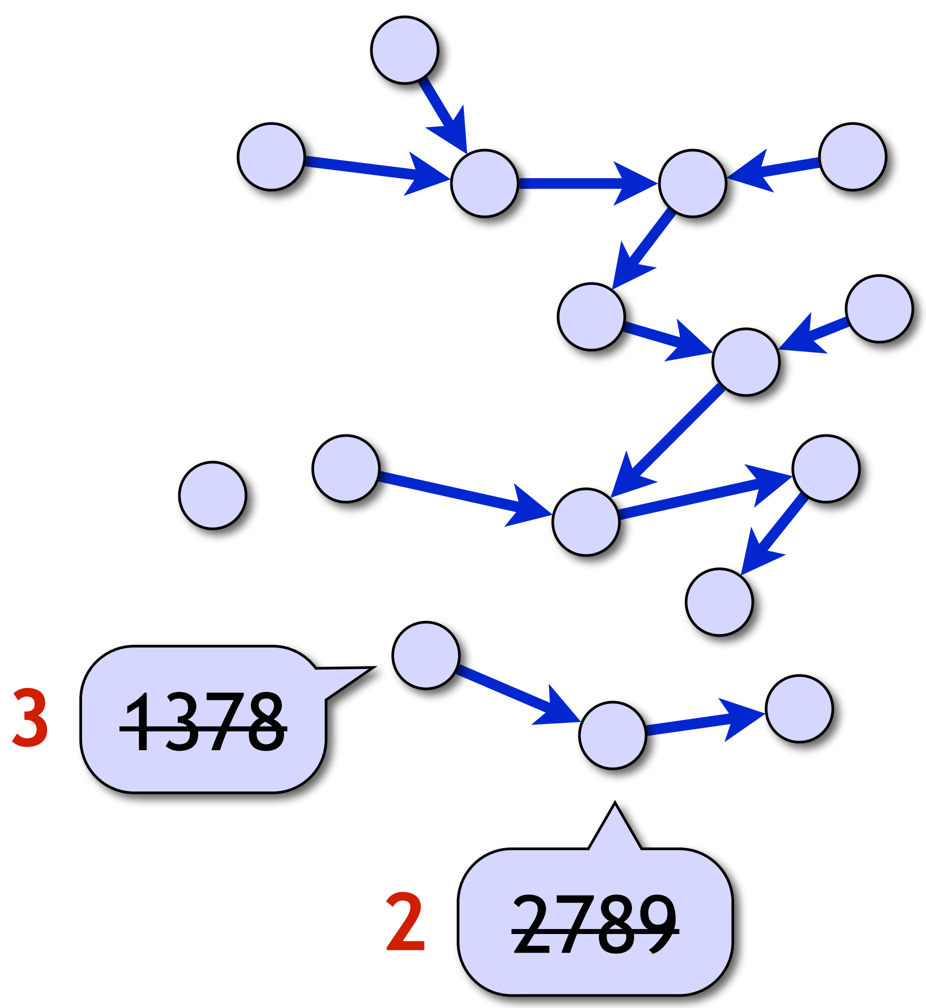
Local 2-approximation algorithm for vertex cover: multicoloured edges

- For each forest in parallel...



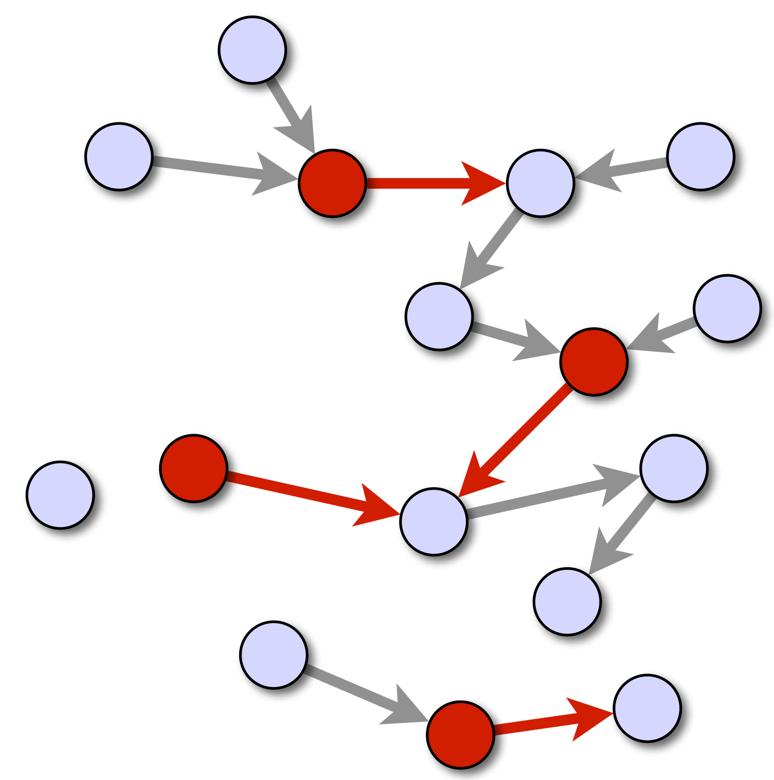
Local 2-approximation algorithm for vertex cover: multicoloured edges

- For each forest in parallel:
  - Use Cole–Vishkin (1986) style colour reduction algorithm
  - Given a k-colouring, finds a **3-colouring** in time $O(\log^* k)$
  - Bit manipulation trick: each step replaces a k-colouring with an $O(\log k)$-colouring



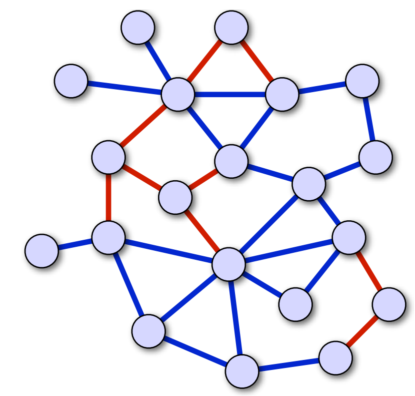
Local 2-approximation algorithm for vertex cover: multicoloured edges

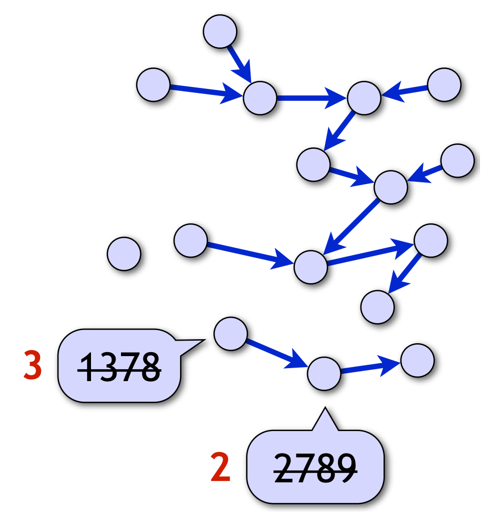
- For each forest and each colour $j = 1, 2, 3$ in sequence:
  - Saturate all outgoing edges of colour-j nodes
  - Node-disjoint stars, easy to saturate in parallel
- In $O(\Delta)$ rounds we have saturated all edges

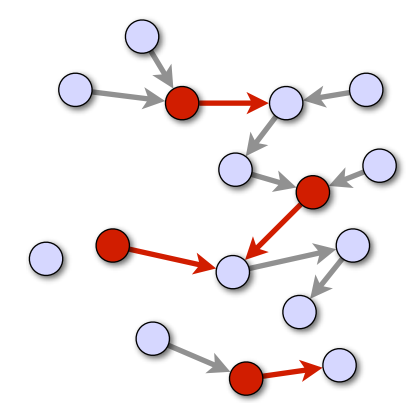


Local 2-approximation algorithm for vertex cover: summary

- Total running time:
  - All edges are saturated or multicoloured: $O(\Delta)$
  - Multicoloured forests are 3-coloured: $O(\log^* k)$
  - 3-coloured forests are saturated: $O(\Delta)$
- $O(\Delta + \log^* k) = O(\Delta + \log^* W)$
  - k is huge, but $\log^*$ grows slowly

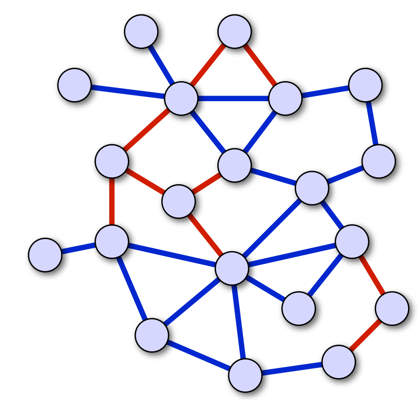


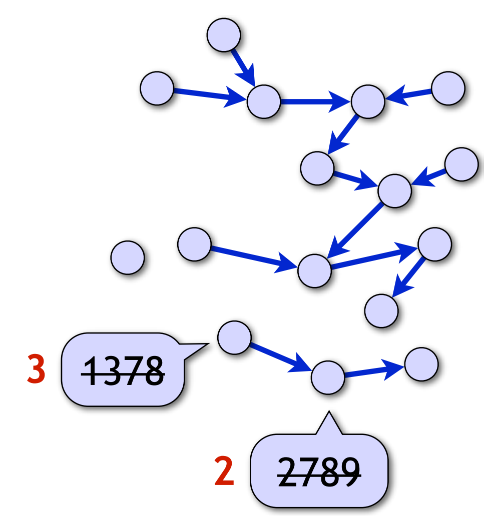


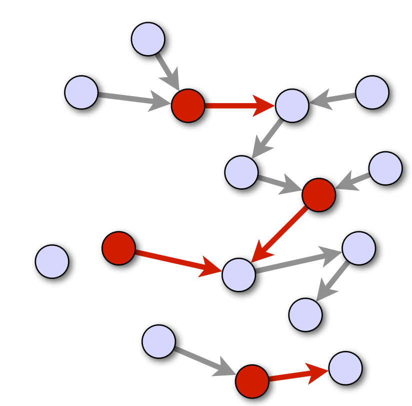


Local 2-approximation algorithm for vertex cover: summary

- Maximal edge packing and 2-approximation of vertex cover in time $O(\Delta + \log^* W)$
  - W = maximum node weight
- Unweighted graphs: running time simply $O(\Delta)$, independent of n
- Can be implemented in the **port-numbering model**

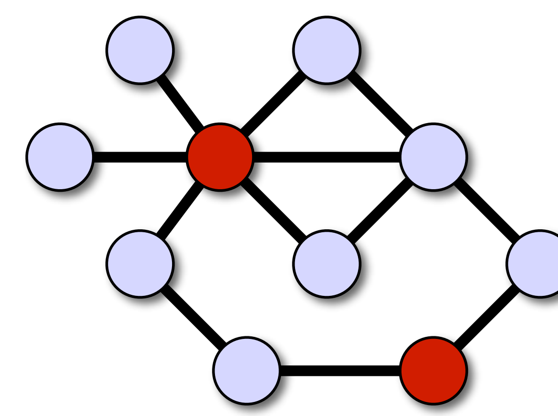






Other examples of positive results

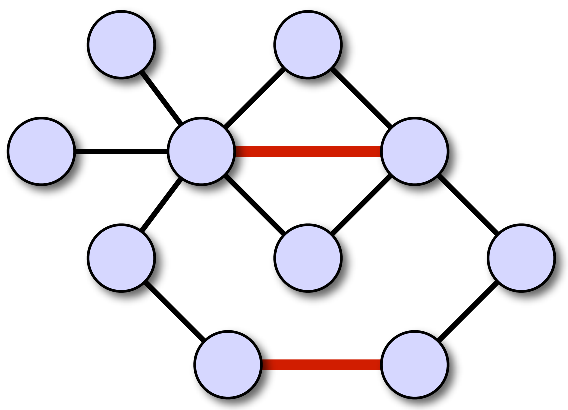
- Local algorithms for dominating sets: only trivial $(\Delta + 1)$-approximation possible in general graphs



- However, there is an approximation scheme for **fractional** dominating sets (Kuhn et al. 2006)
- And constant-factor approximation algorithms for dominating sets in **planar** graphs (Czygrinow et al. 2008, Lenzen et al. 2008)

Other examples of positive results

- Edge dominating sets in the port-numbering model
- **Best possible** approximation ratios:



Graph family		Approximation ratio
d-regular graphs	d = 1, 3, …	$4 - 6/(d + 1)$
	d = 2, 4, …	$4 - 2/d$
graphs with degree ≤ Δ	Δ = 3, 5, …	$4 - 2/(\Delta - 1)$
	Δ = 2, 4, …	$4 - 2/\Delta$

Other examples of positive results

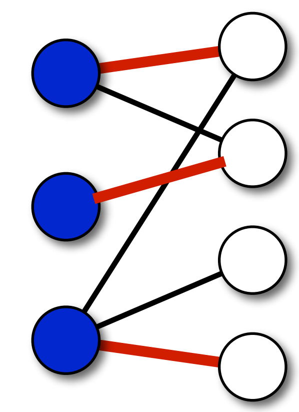
- Edge dominating sets in the port-numbering model
- Best possible approximation ratios:

Local algorithms!

Graph family		Approximation ratio	Time
d-regular graphs	d = 1, 3, ...	$4 - 6/(d + 1)$	$O(d^2)$
	d = 2, 4, ...	$4 - 2/d$	$O(1)$
graphs with degree $\leq \Delta$	Δ = 3, 5, ...	$4 - 2/(\Delta - 1)$	$O(\Delta^2)$
	Δ = 2, 4, ...	$4 - 2/\Delta$	$O(\Delta^2)$

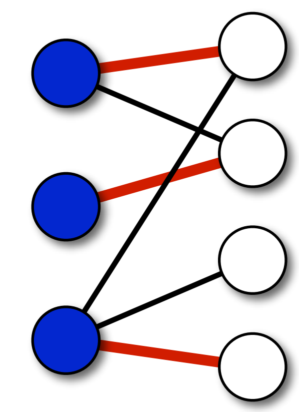
Other examples of positive results

- Matchings in 2-coloured graphs, max degree ≤ Δ
- Time Ω(*n*):
  - maximum matching
  - stable matching
- Time *f*(Δ, *ε*):
  - (1 + *ε*)-approximation of maximum matching
  - “almost stable” matching
    (fraction *ε* of unstable edges)



Other examples of positive results

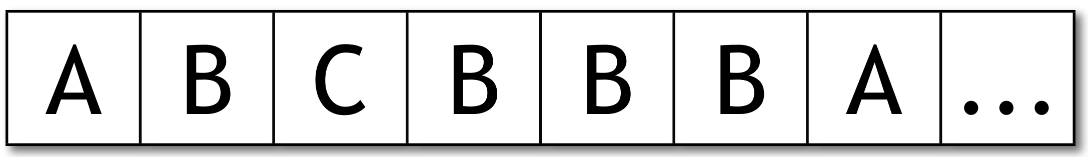
- Matchings in 2-coloured graphs, max degree ≤ Δ
- Time $\Omega(n)$, even with **unique IDs**:
  - maximum matching
  - stable matching
- Time $f(\Delta, \varepsilon)$, in **port-numbering model**:
  - $(1 + \varepsilon)$-approximation of maximum matching
  - “almost stable” matching
    (fraction ε of unstable edges)



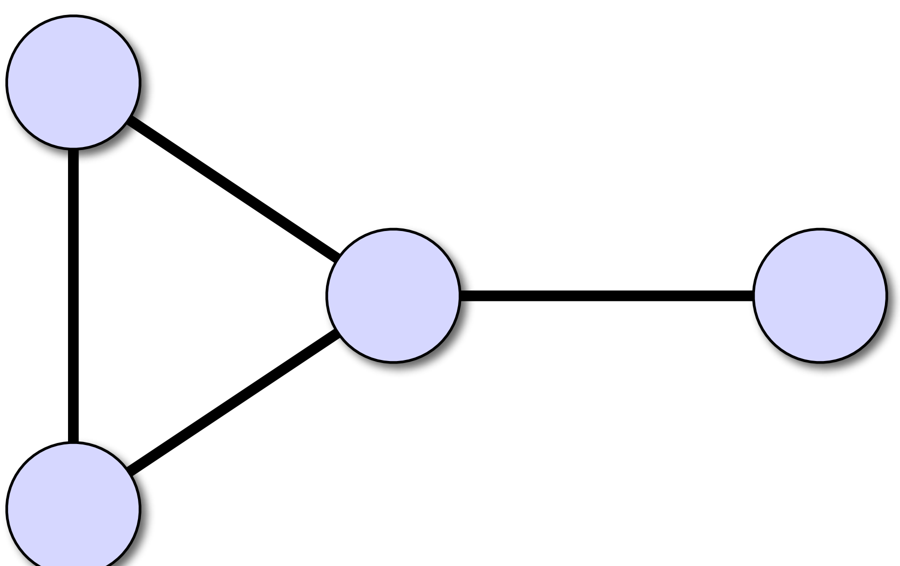
Part III: Other models of computation

- Can we relate local algorithms to traditional complexity classes such as NC^0?

Distributed algorithms vs. traditional computational complexity

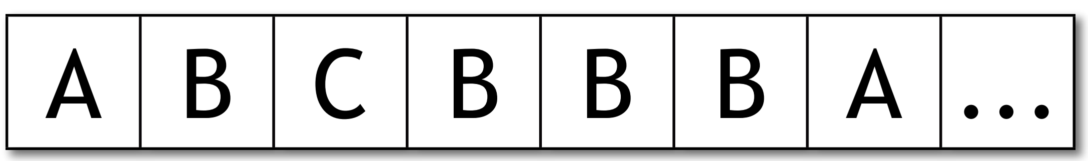


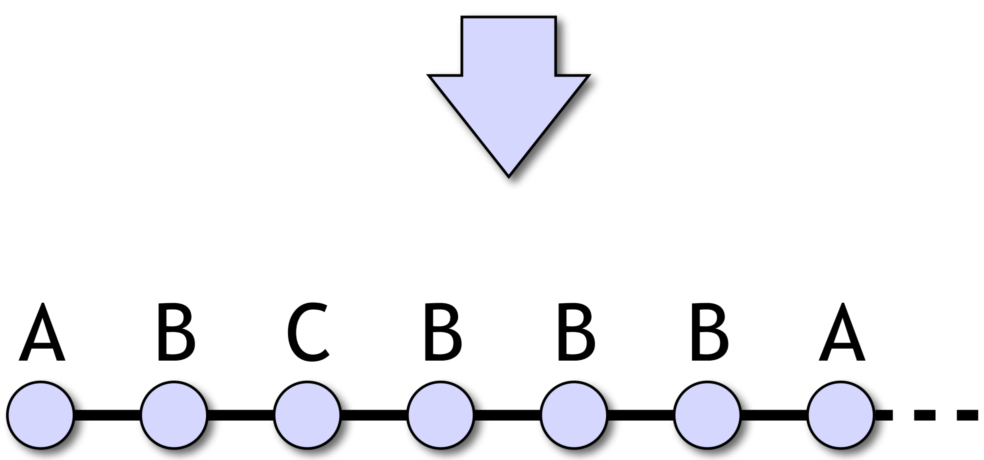
- Traditional view:
  - Problem instance encoded as a string



- Distributed algorithms:
  - Problem instance = structure of the system (graph)

Distributed algorithms vs. traditional computational complexity

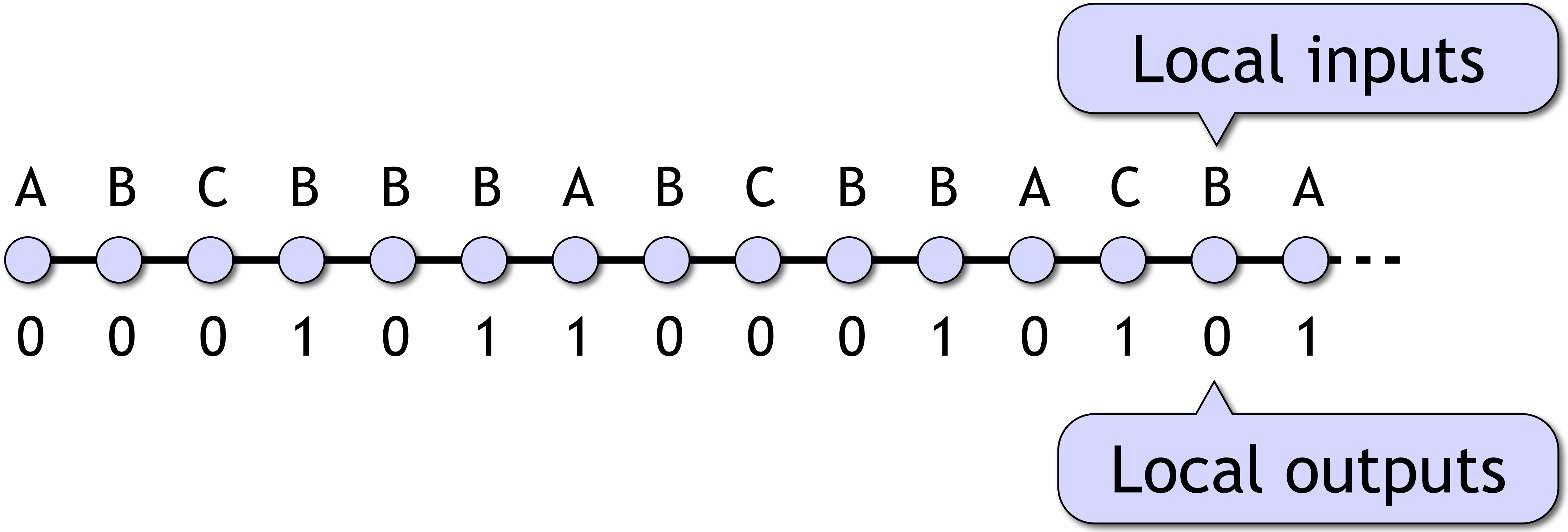




- Traditional view:
  - Problem instance encoded as a string
  - Can be interpreted as a path graph with local inputs
- Everything is a graph
- Let's study a simple model of computation...

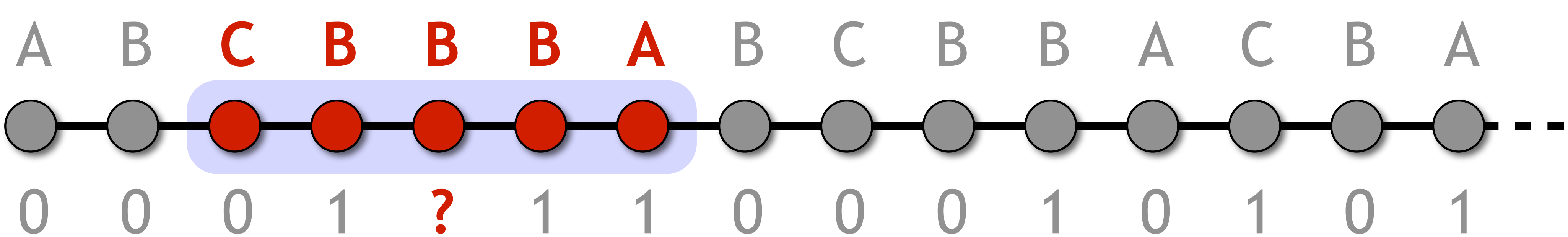
Distributed algorithms vs. traditional computational complexity

- Distributed algorithms on path graphs
- Constant-size local input
    - Hence no unique IDs



Distributed algorithms vs. traditional computational complexity

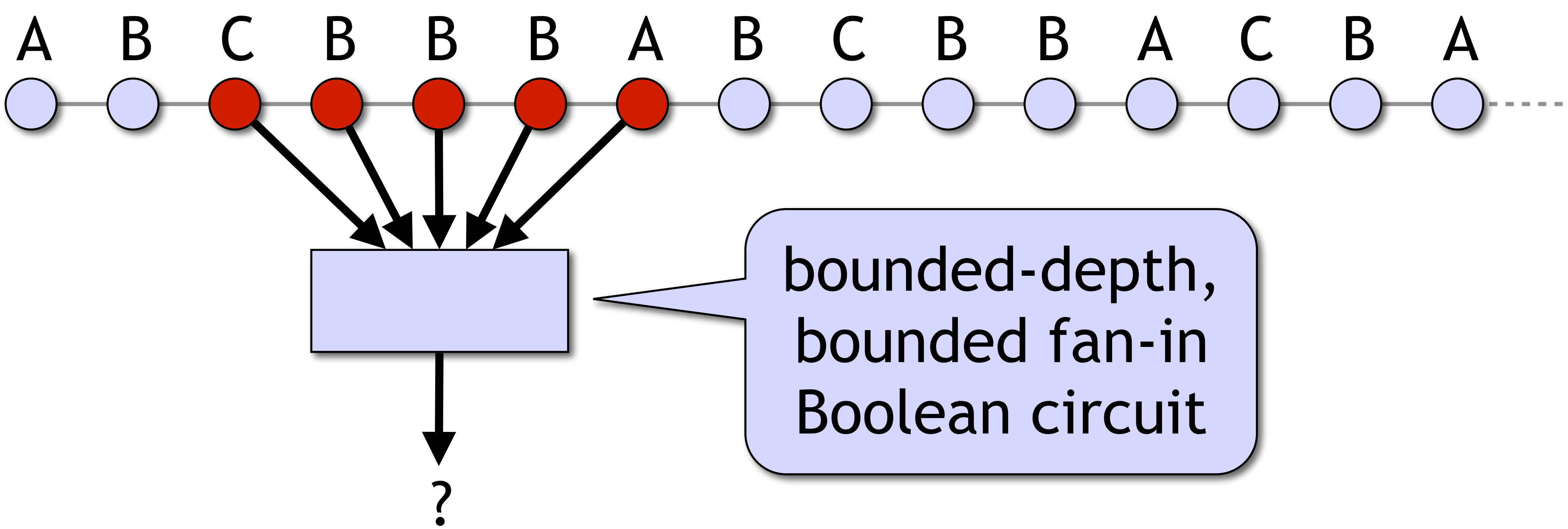
- **Deterministic local** algorithms on path graphs
- Constant-size local input



(here T = 2)

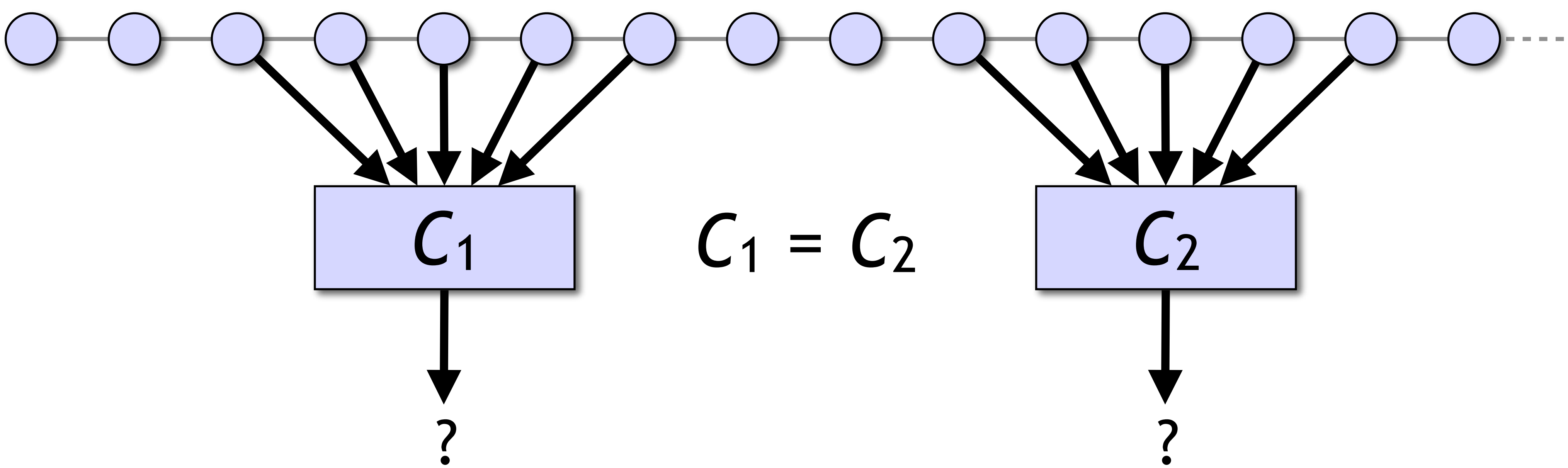
Distributed algorithms vs. traditional computational complexity

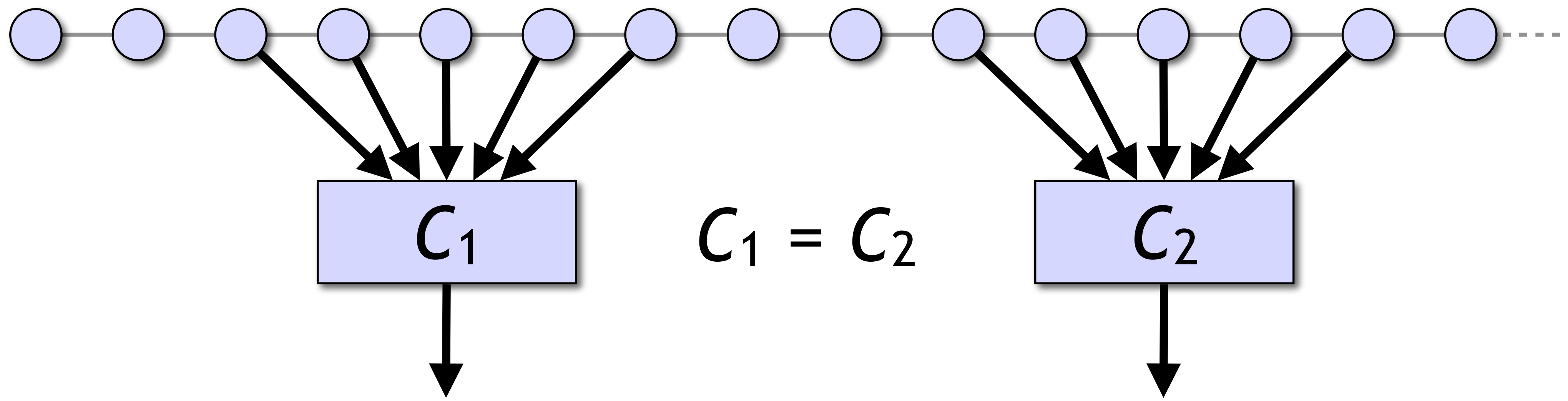
- Deterministic local algorithms on path graphs
- Constant-size local input



Distributed algorithms vs. traditional computational complexity

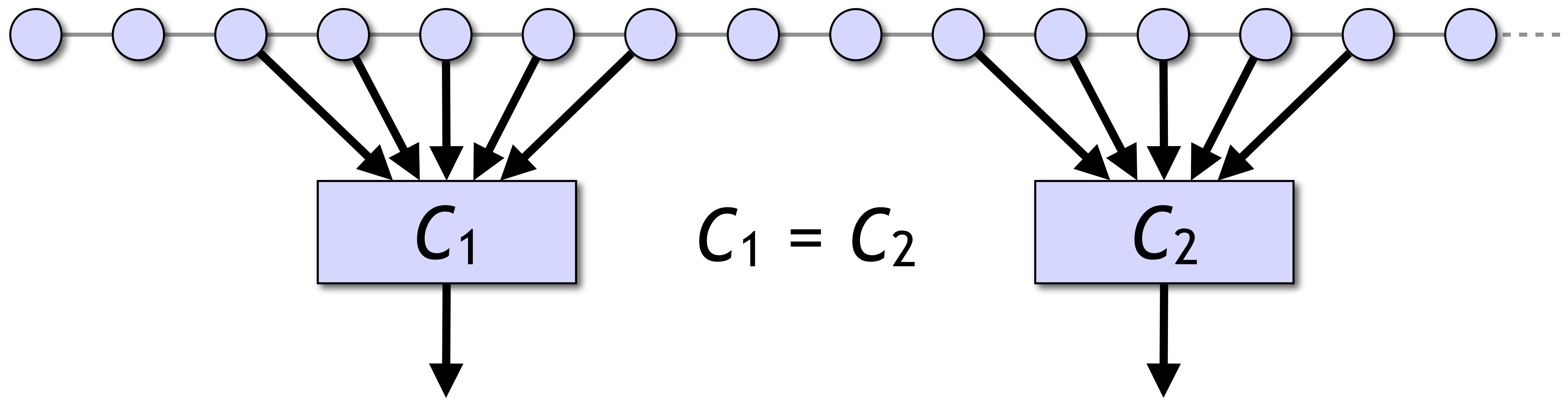
- Deterministic local algorithms on path graphs
- Constant-size local input



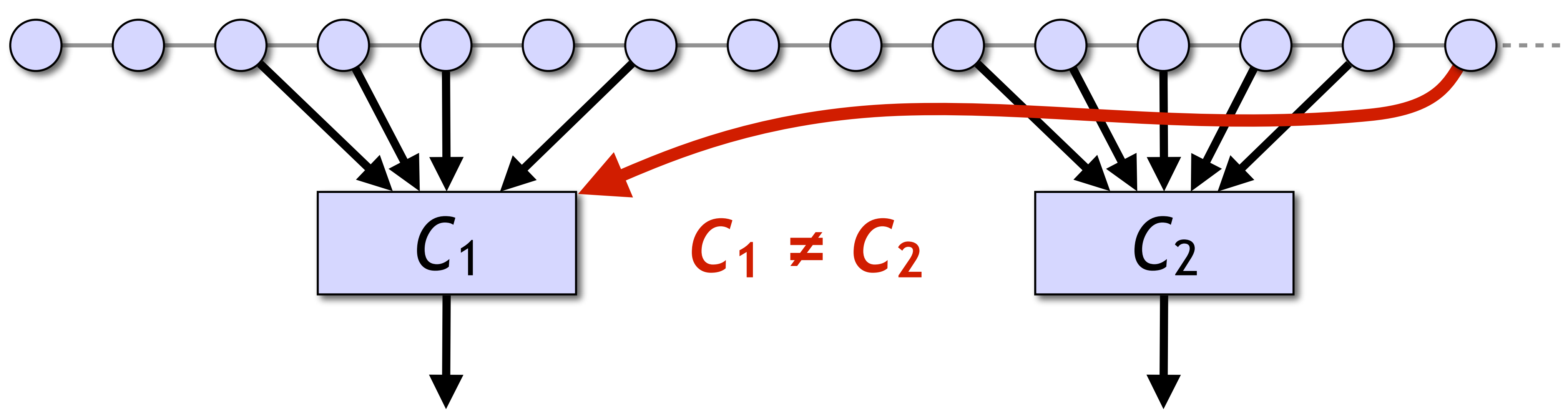


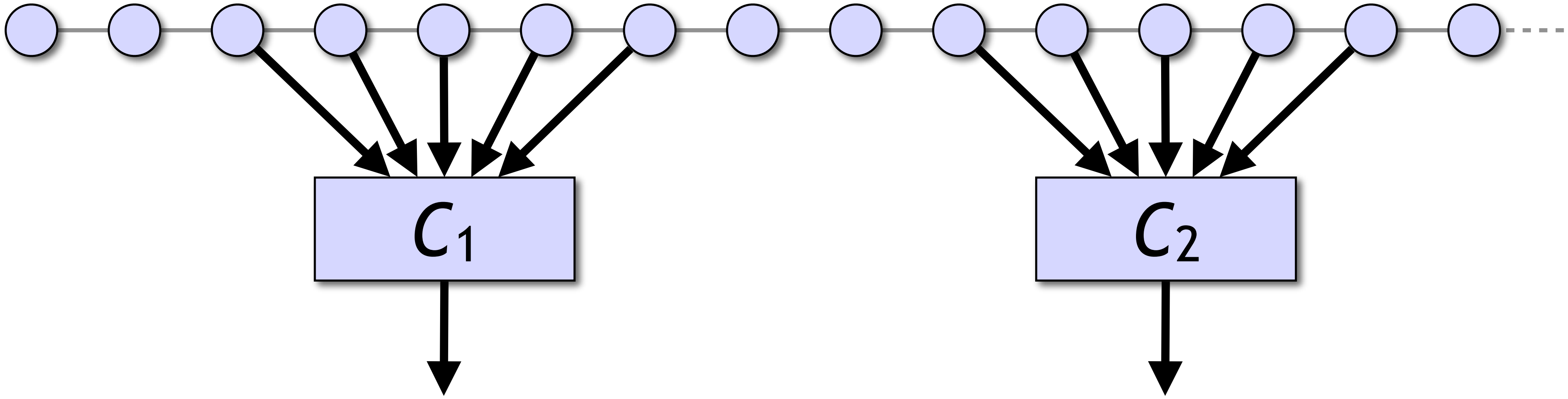
Ridiculously restrictive model —

let's consider two different extensions...

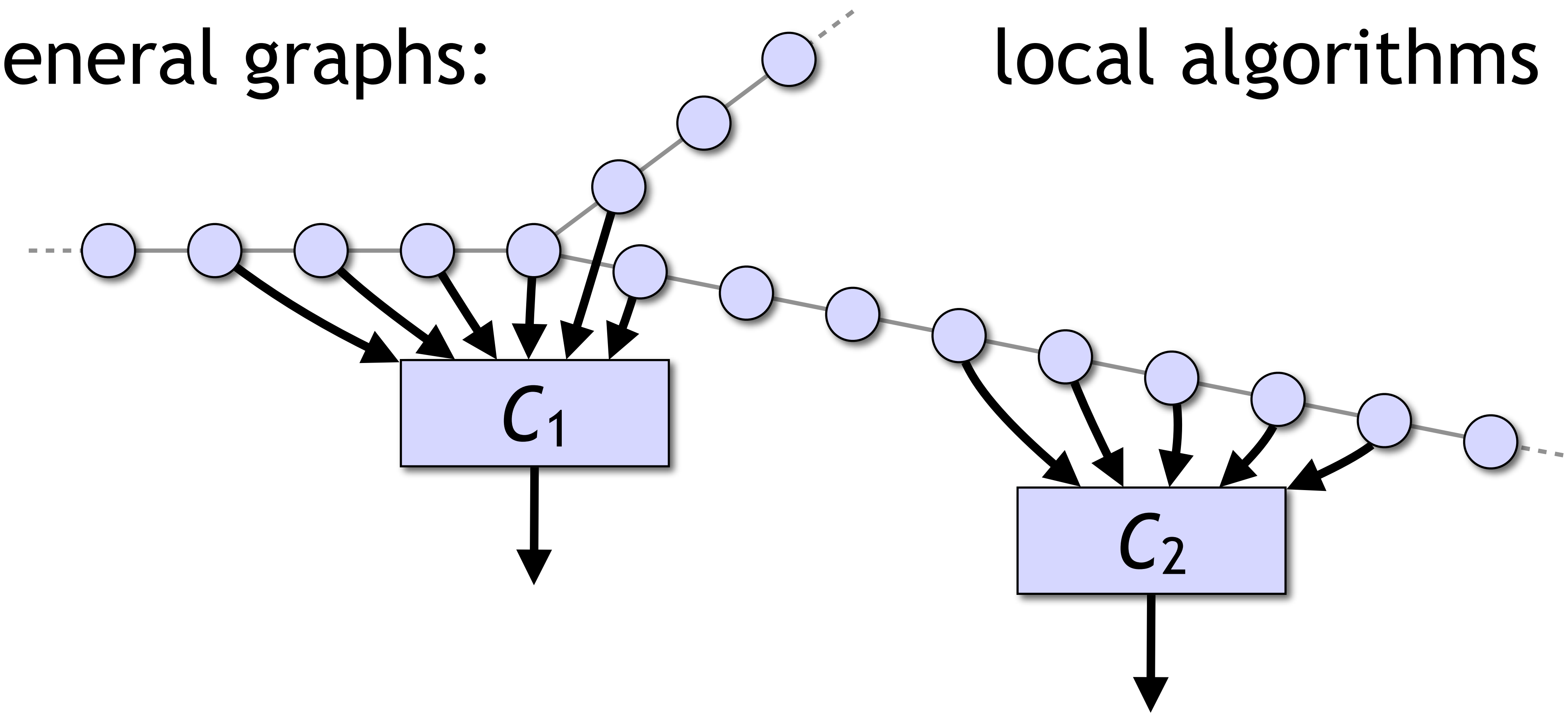


Non-local connections, different circuits: NC^0

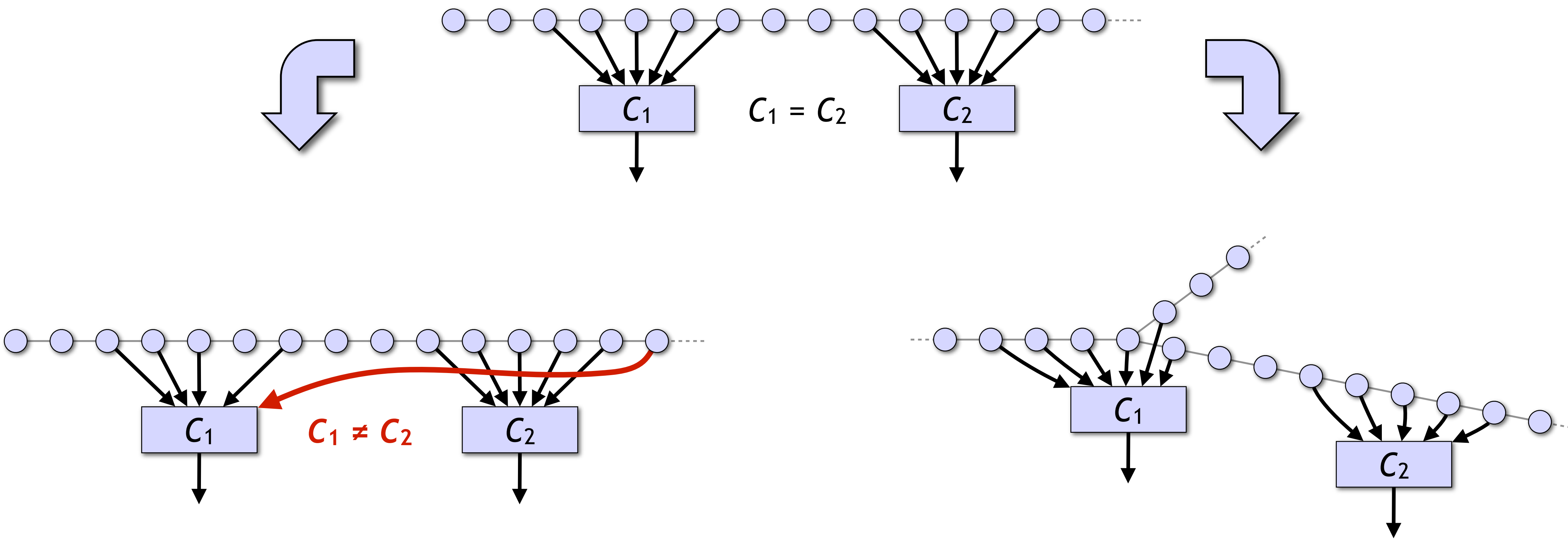




General graphs: local algorithms

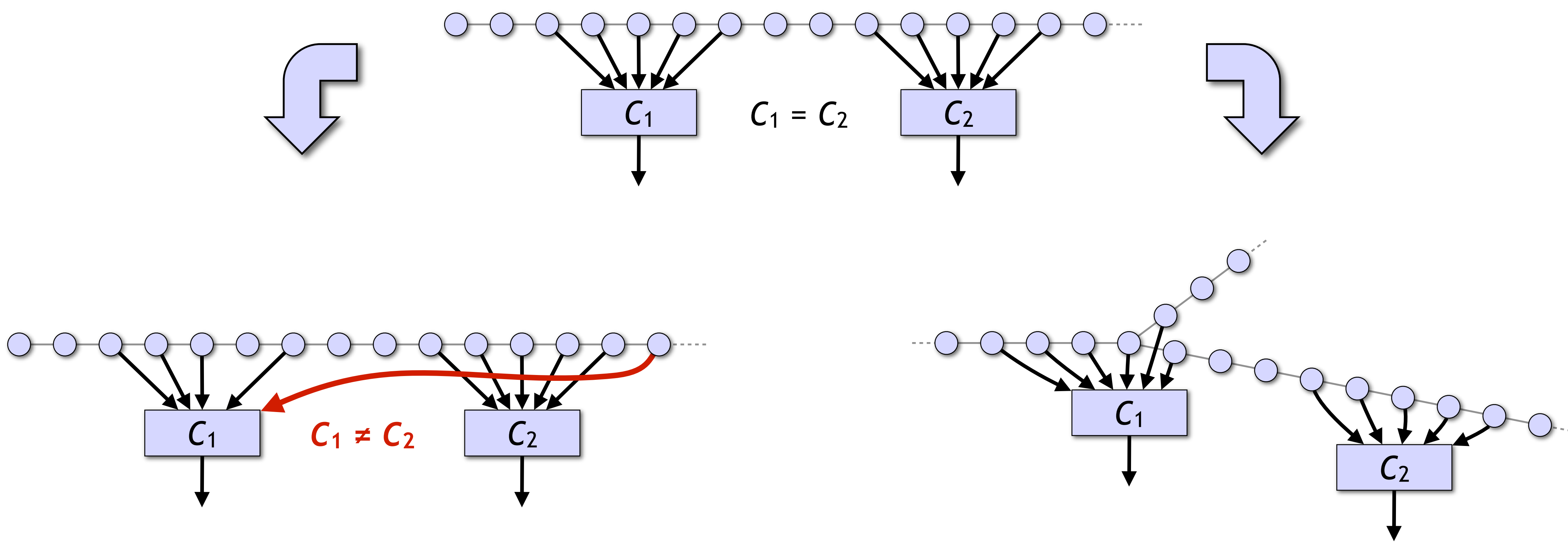


Distributed algorithms vs. traditional computational complexity



- NC^0
- Deterministic local algorithms, port numbering

Distributed algorithms vs. traditional computational complexity



- NC^0, NC^1, NC, RNC, ...
- Traditional computational complexity

- Deterministic local algorithms, port numbering
- Distributed algorithms

Conclusions

- Local algorithms & port-numbering model
  - Non-trivial problems can be solved in very simple models of distributed computing
  - Tight, unconditional lower bounds can be proven
- Research directions
  - Better understand the similarities between the two models?
  - Traditional computational complexity studies strings (= path graphs), consider more general graphs?