CPSC 768:
Scalable and Private Graph Algorithms

Lecture 24: Distributed Graph Algorithms

Quanquan C. Liu
quanquan.liu@yale.edu

Announcements

- **Final project report and presentation: April 24th (last day of class)**
  - Final project presentation is a 30 min presentation
- **Last day of Open Problem Sessions: April 26th** (last week of classes)
  - Will be turned into a reading group/continue with OPS, stay tuned!

Traditional Distributed Graph Algorithms

- The input graph is not only the input but also represents the **communication graph**

Traditional Distributed Graph Algorithms

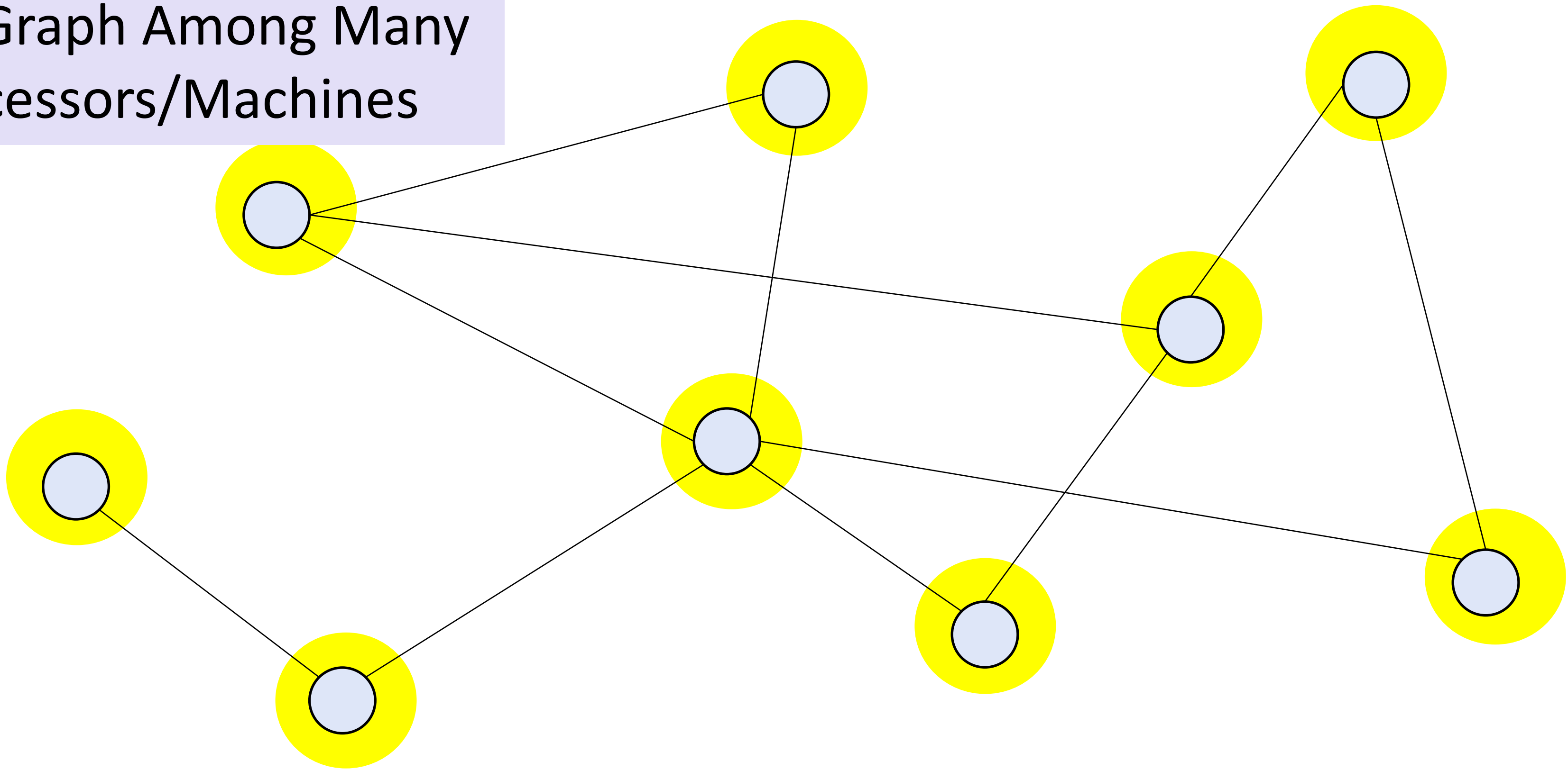
- The input graph is not only the input but also represents the **communication graph**
- Nodes can send messages along edges in **synchronous rounds**

Traditional Distributed Graph Algorithms

- The input graph is not only the input but also represents the **communication graph**
- Nodes can send messages along edges in **synchronous rounds**

Distributed Algorithms and Networks

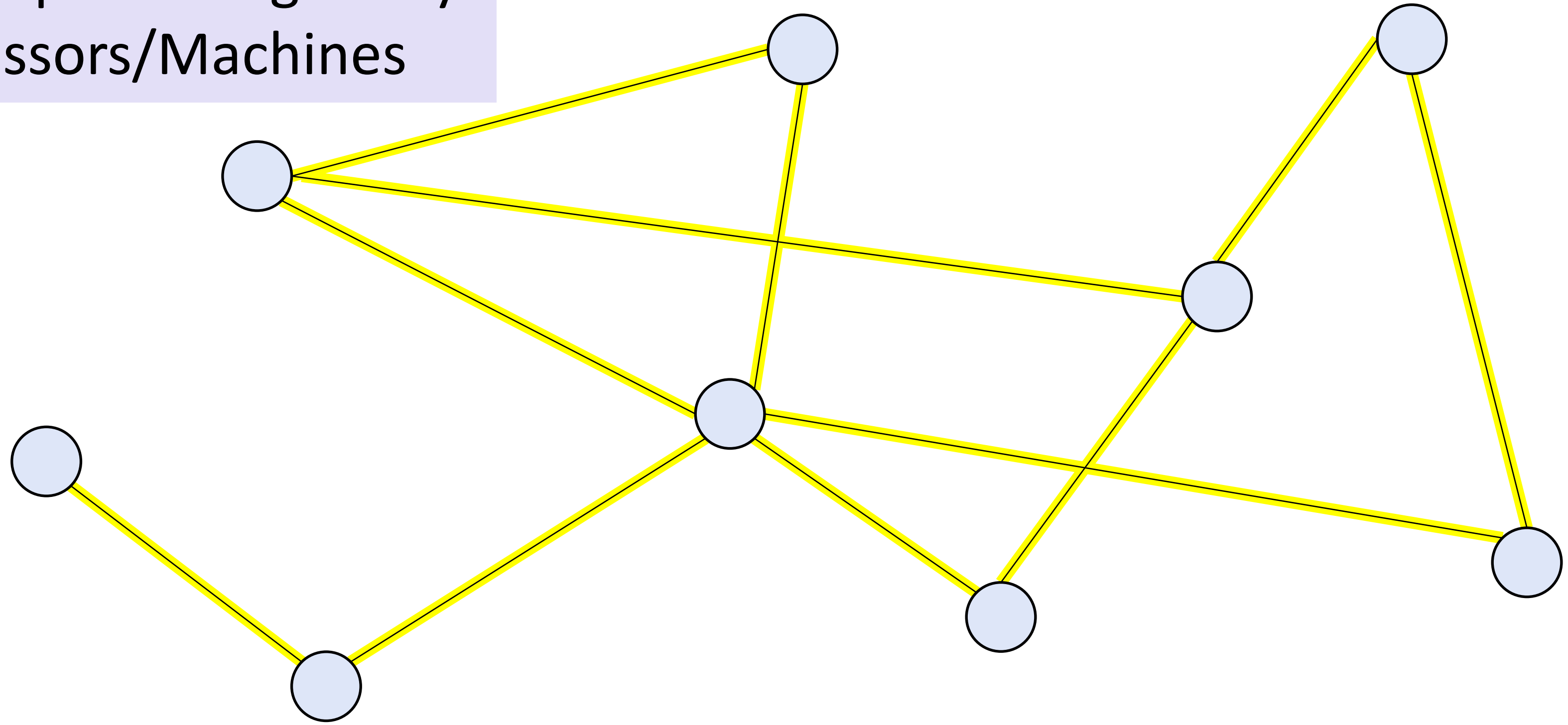
Split the Large Graph Among Many Different Processors/Machines



Each Node is a Processor/Machine

Distributed Algorithms and Networks

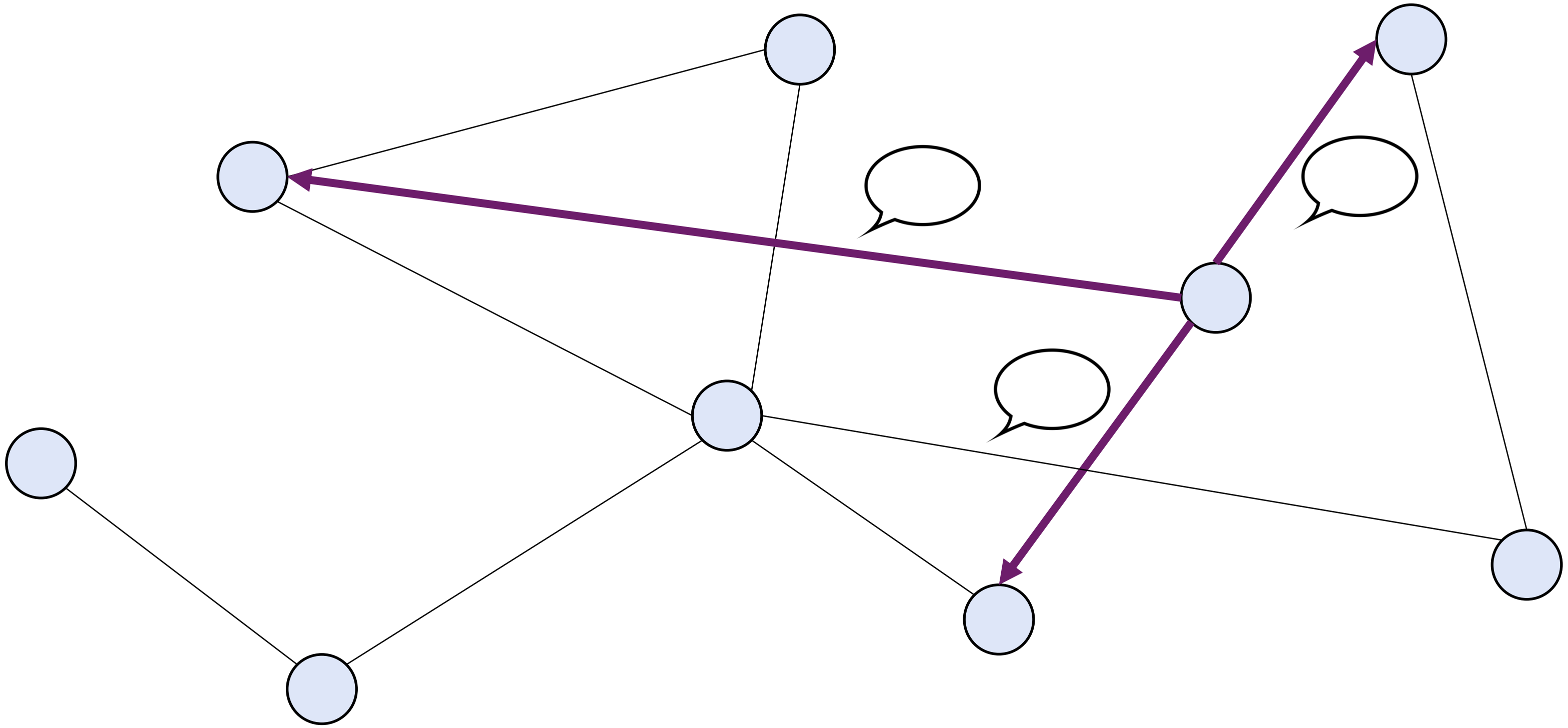
Split the Large Graph Among Many Different Processors/Machines



Each Node is a Processor/Machine

Edges are Communication Links

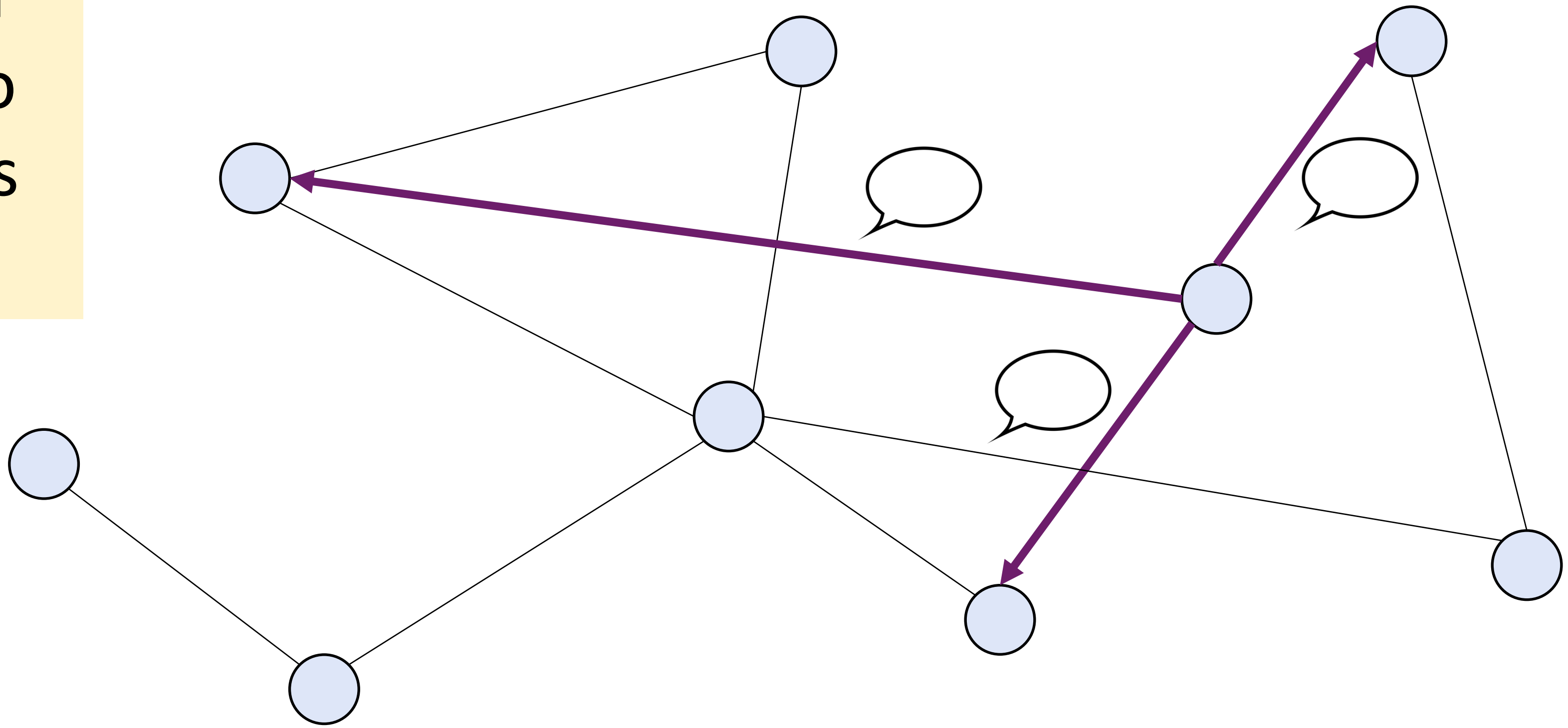
Distributed Algorithms and Networks



Broadcast model: if a node wants to send a message, it must send all the same message to all neighbors simultaneously in the round

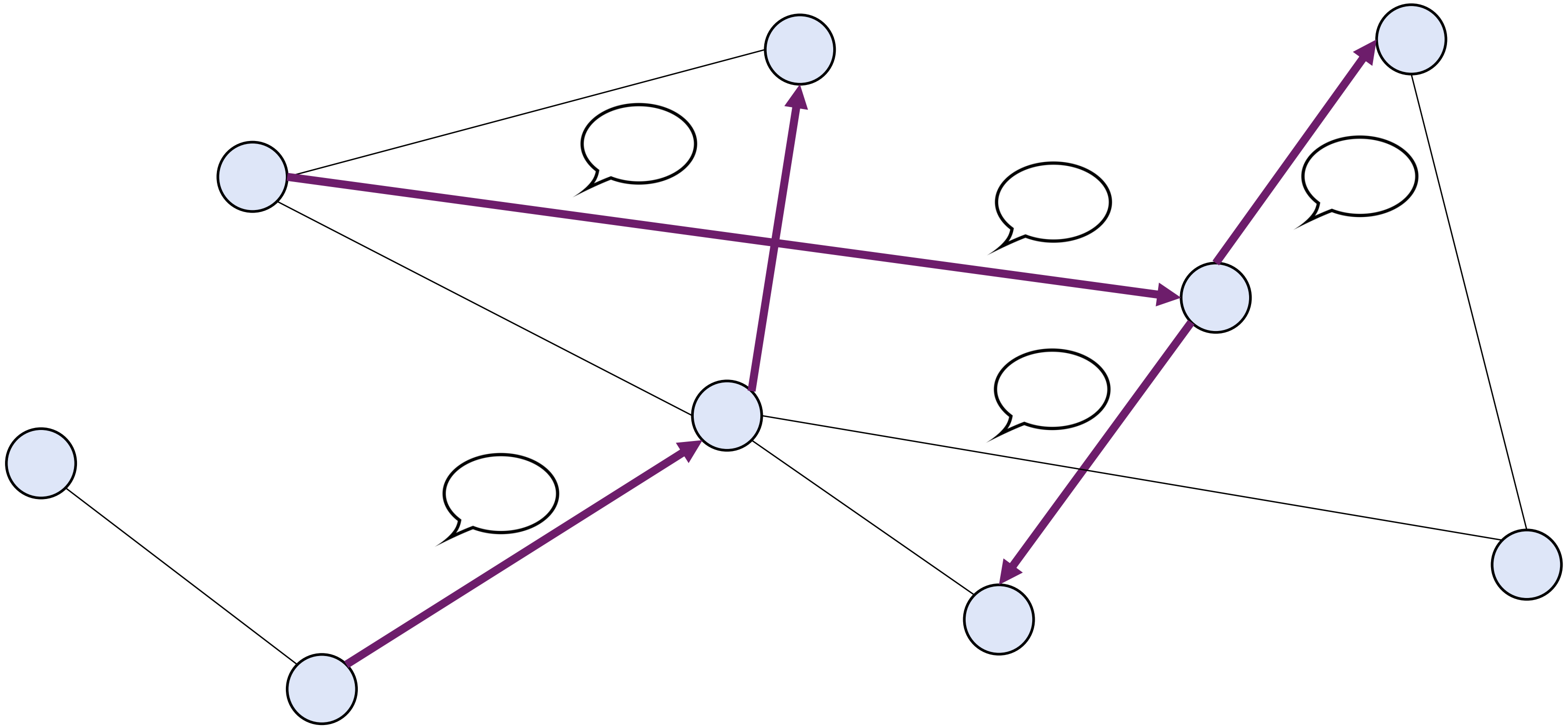
Distributed Algorithms and Networks

Nodes Send **Messages** to Other Nodes Via Edges



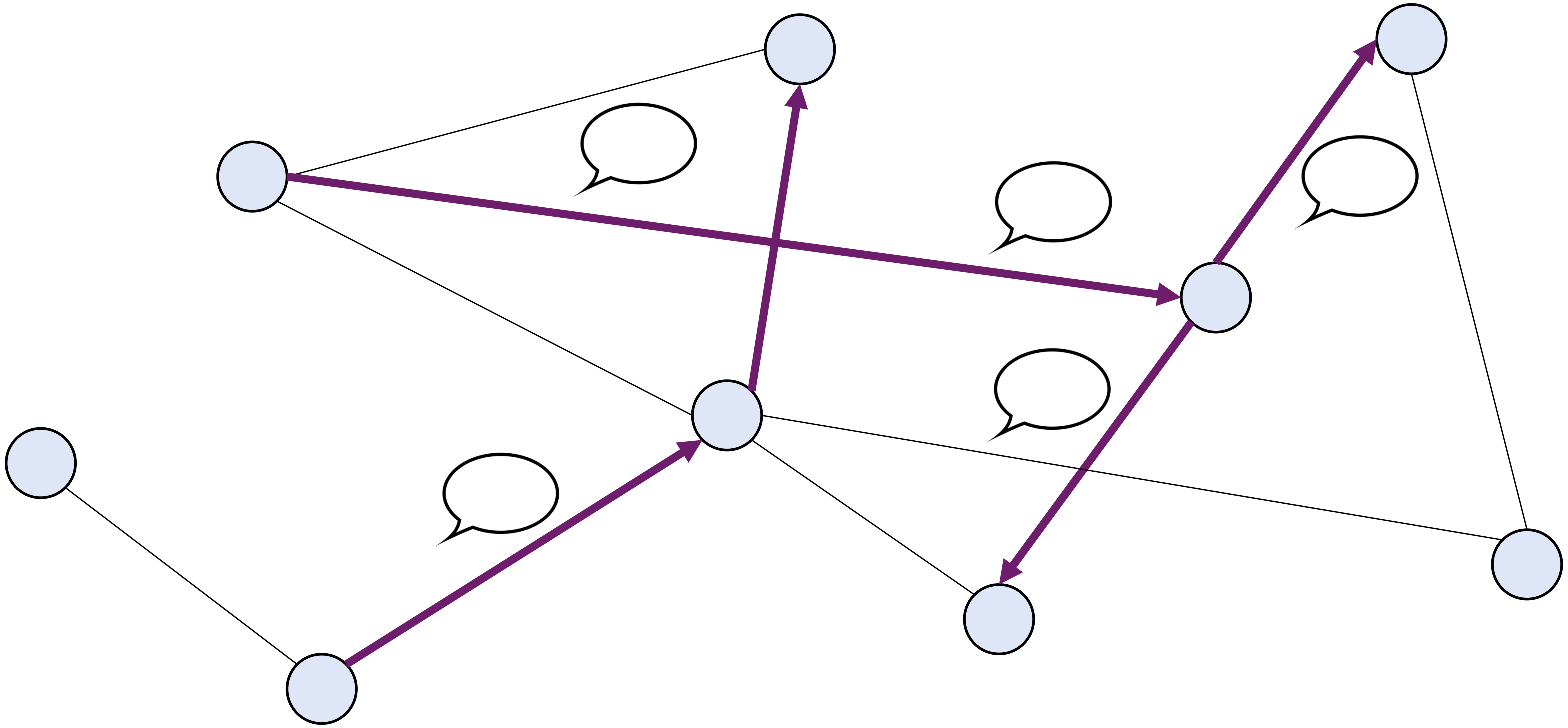
Broadcast model: if a node wants to send a message, it must send all the same message to all neighbors simultaneously in the round

Distributed Algorithms and Networks



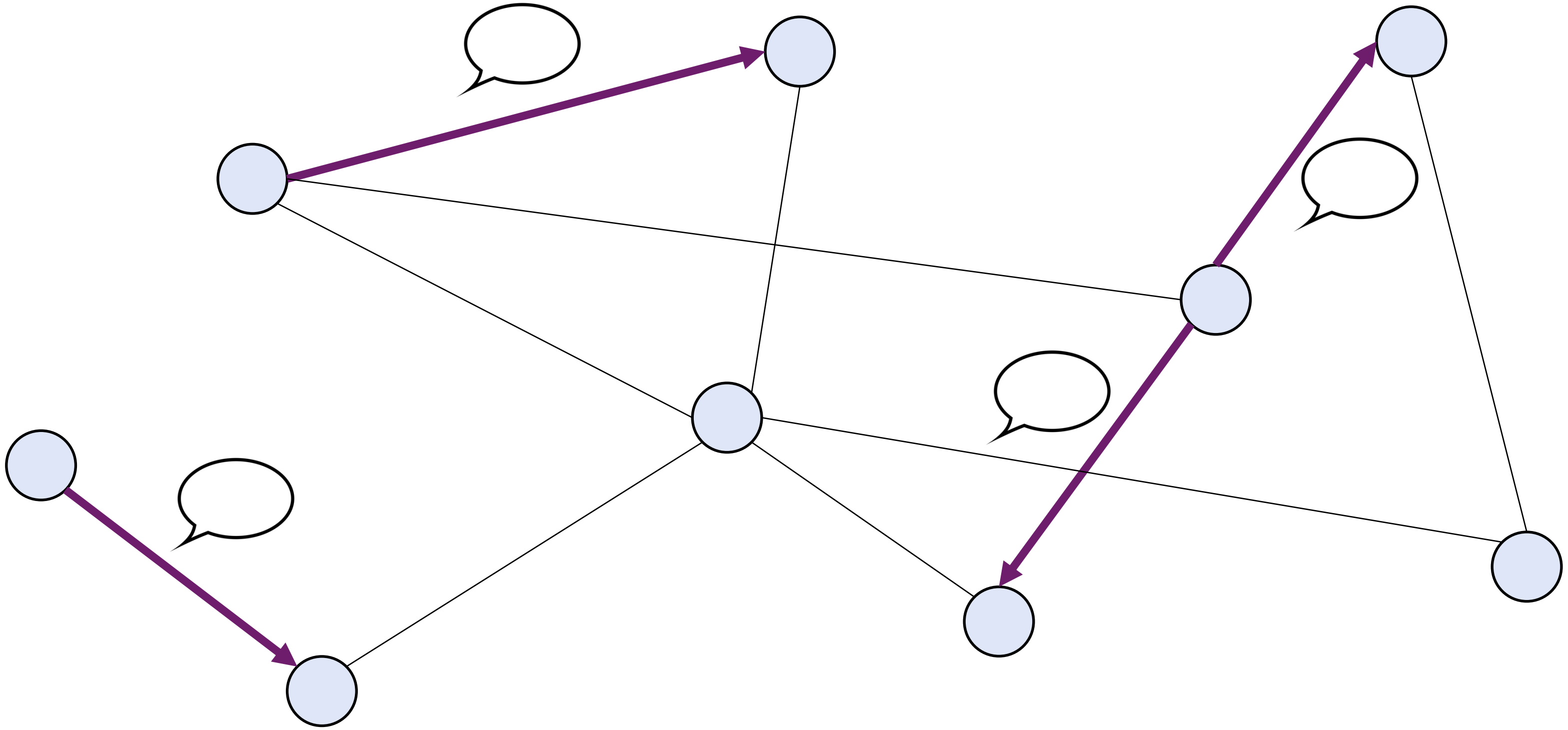
Point-to-point message passing:
Nodes Can Choose to Send to Some/All Neighbors

Distributed Algorithms and Networks



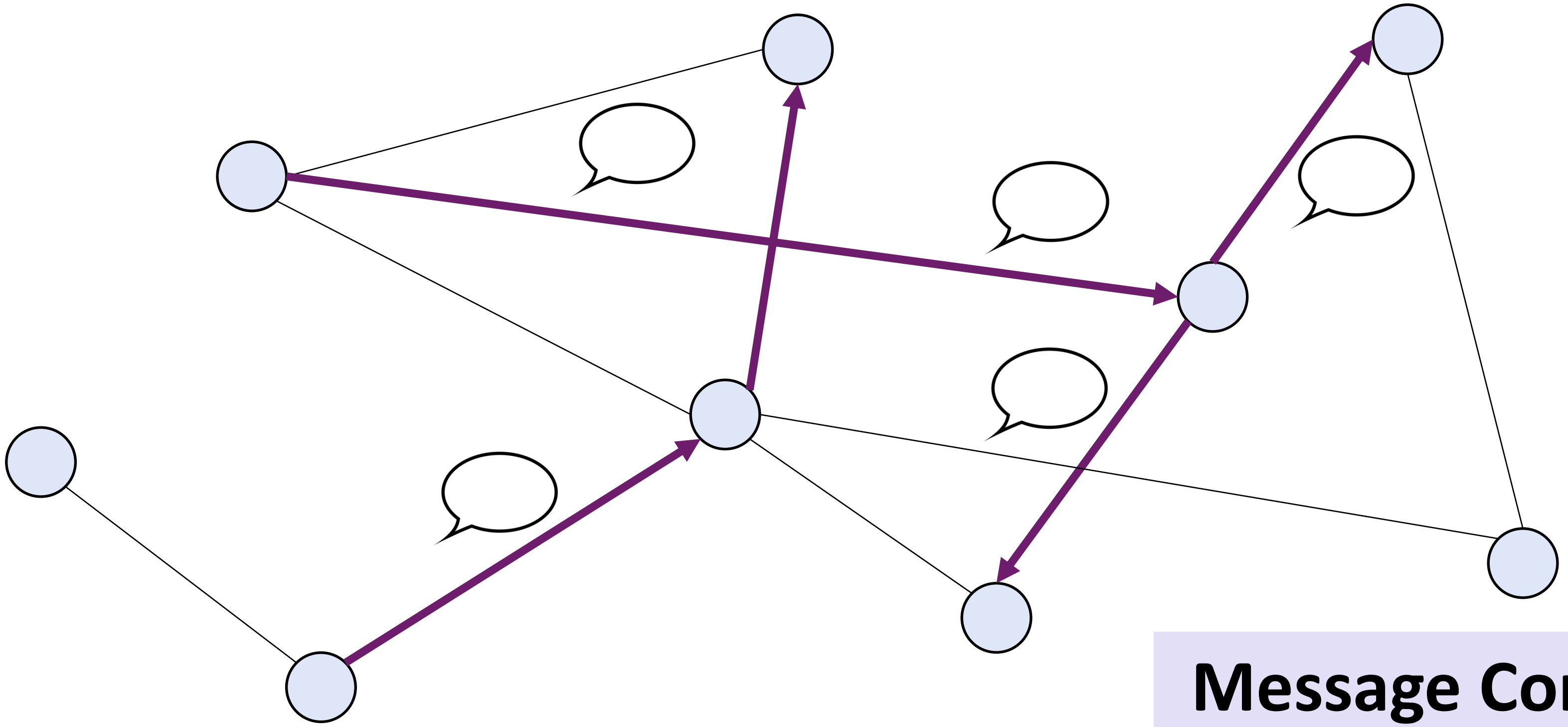
Nodes Use Multiple **Rounds** of Communication to Send Messages

Distributed Algorithms and Networks



Each **Round** Nodes Can Send to Same or Different Neighbors

Distributed Algorithms and Networks

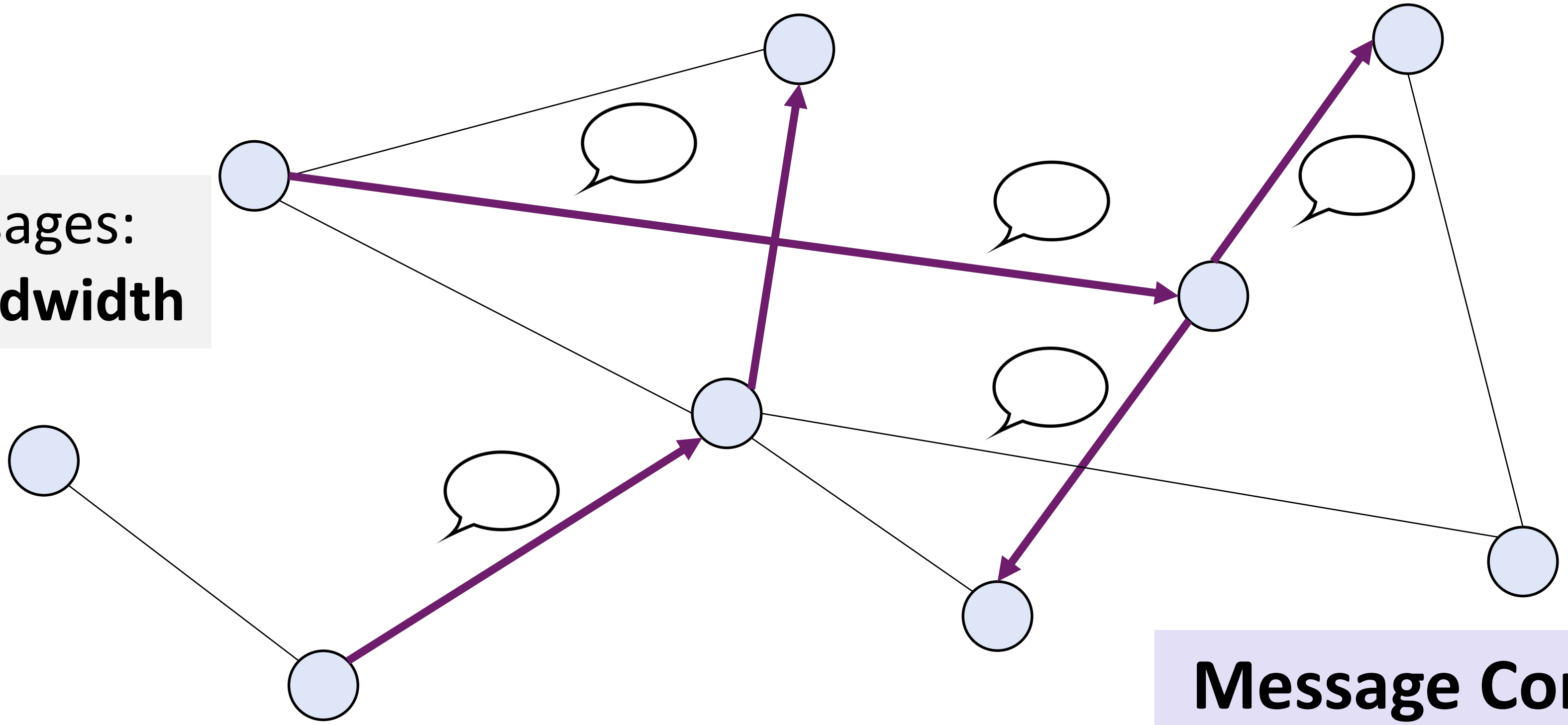


CONGEST Model:
Messages have $O(\log n)$ size

Message Complexity
Number of Messages
Sent in Total

Distributed Algorithms and Networks

Too many messages:
overwhelms bandwidth

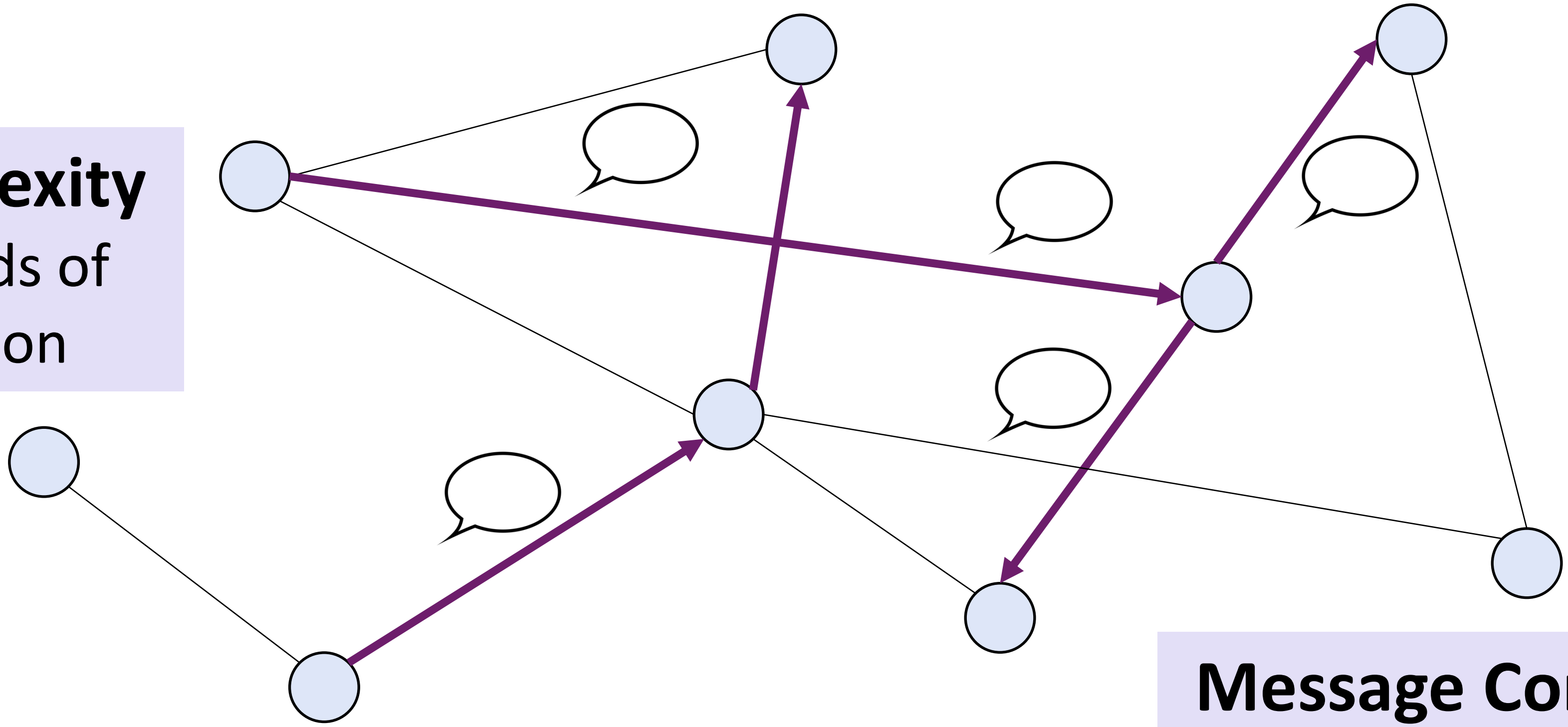


CONGEST Model:
Messages have $O(\log n)$ size

Message Complexity
Number of Messages
Sent in Total

Distributed Algorithms and Networks

Round Complexity
Multiple Rounds of
Communication

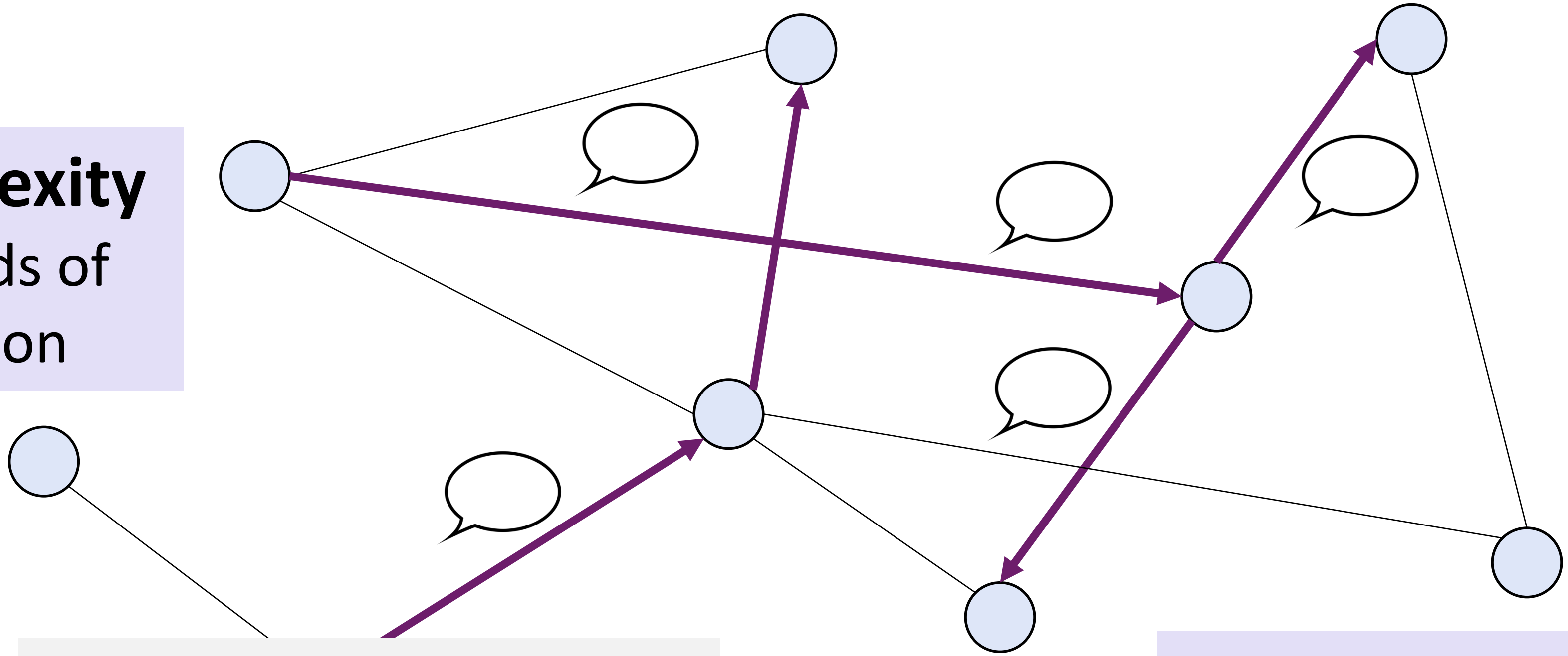


CONGEST Model:
Messages have $O(\log n)$ size

Message Complexity
Number of Messages
Sent in Total

Distributed Algorithms and Networks

Round Complexity
Multiple Rounds of
Communication



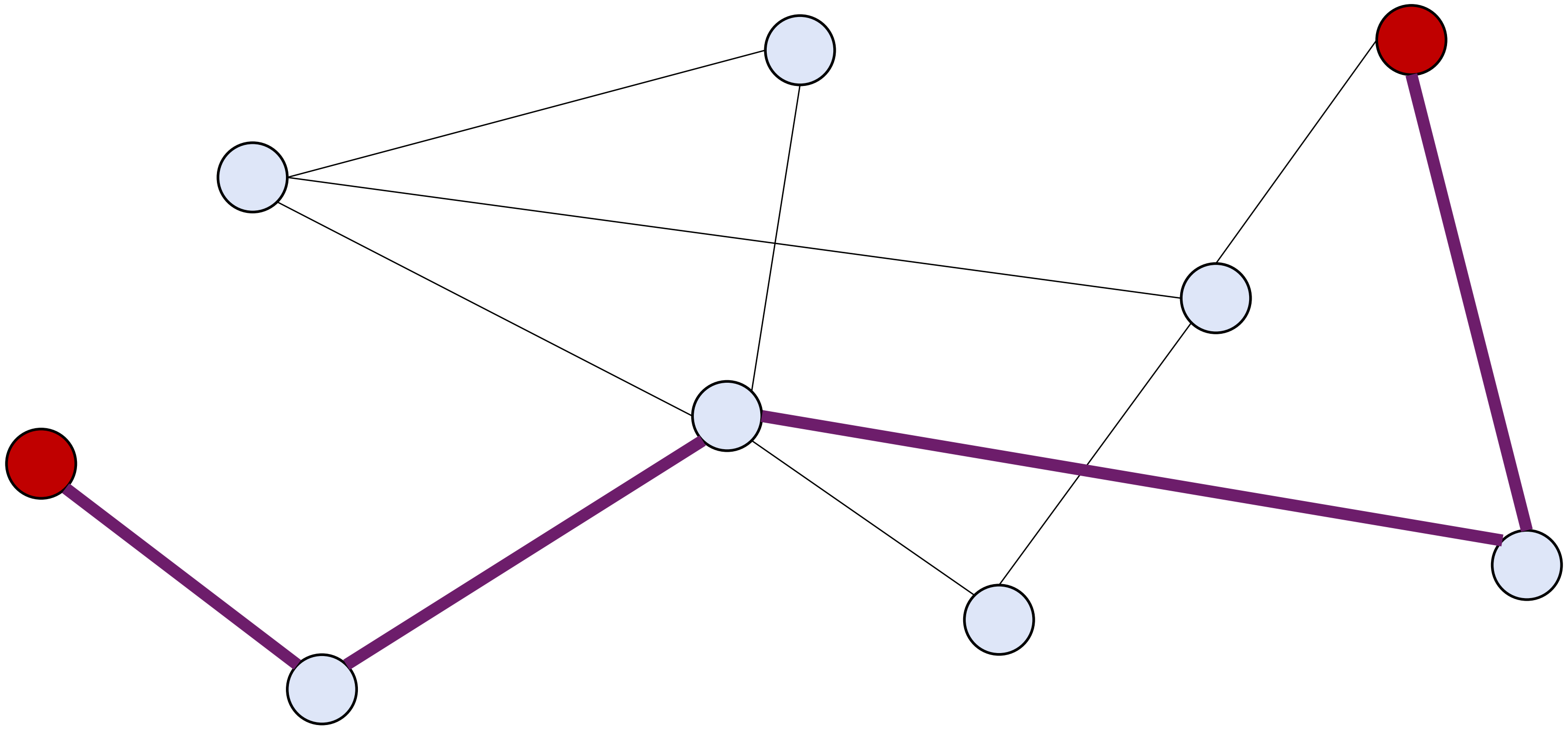
Too many rounds:
takes too long and sends too many messages

Message Complexity
Number of Messages
Sent in Total

Several Caveats

Diameter longest shortest path between any two nodes

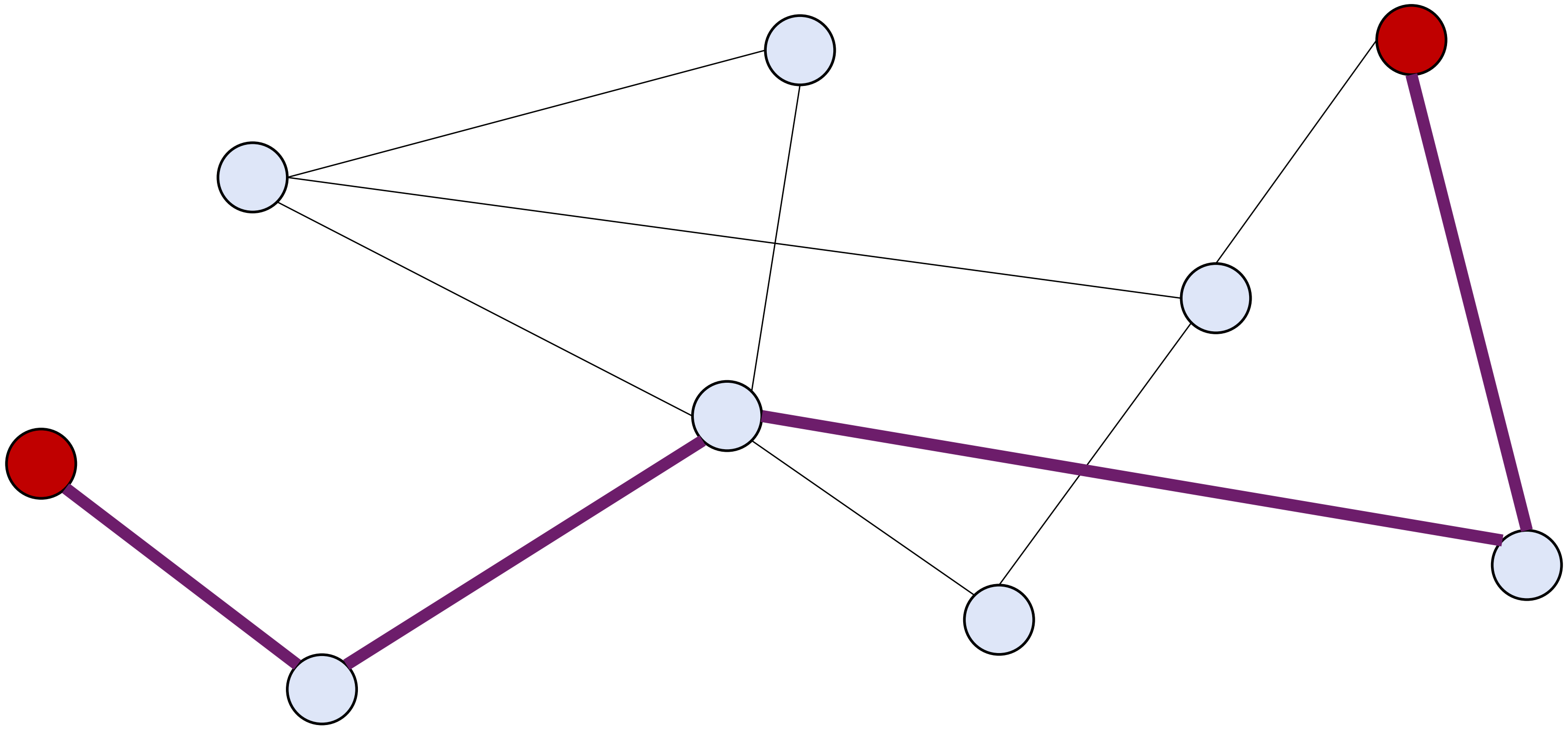
- Information propagation requires **diameter** number of rounds



Several Caveats

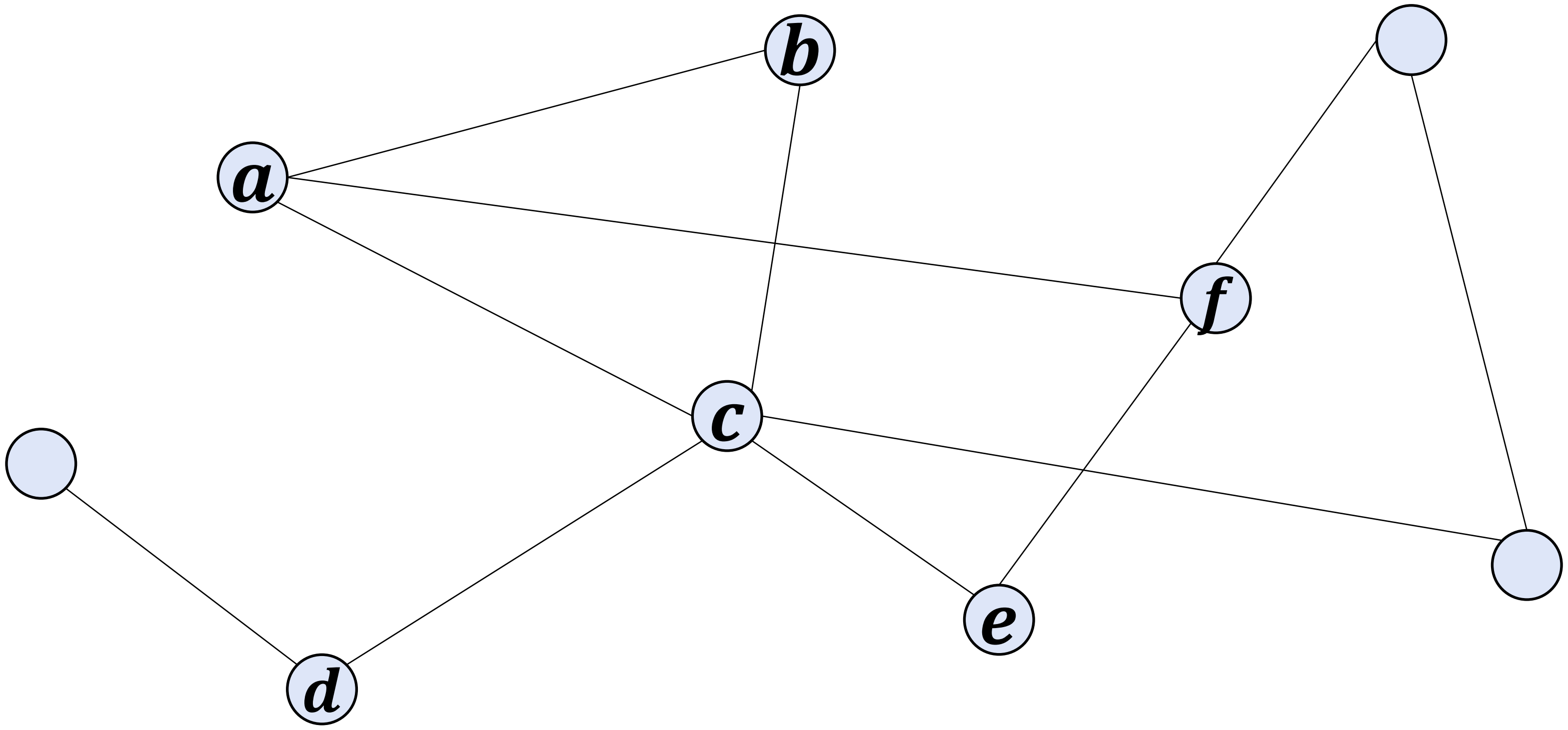
Diameter longest shortest path between any two nodes

- Can only model purely **decentralized networks**



Message Size Constraint for CONGEST

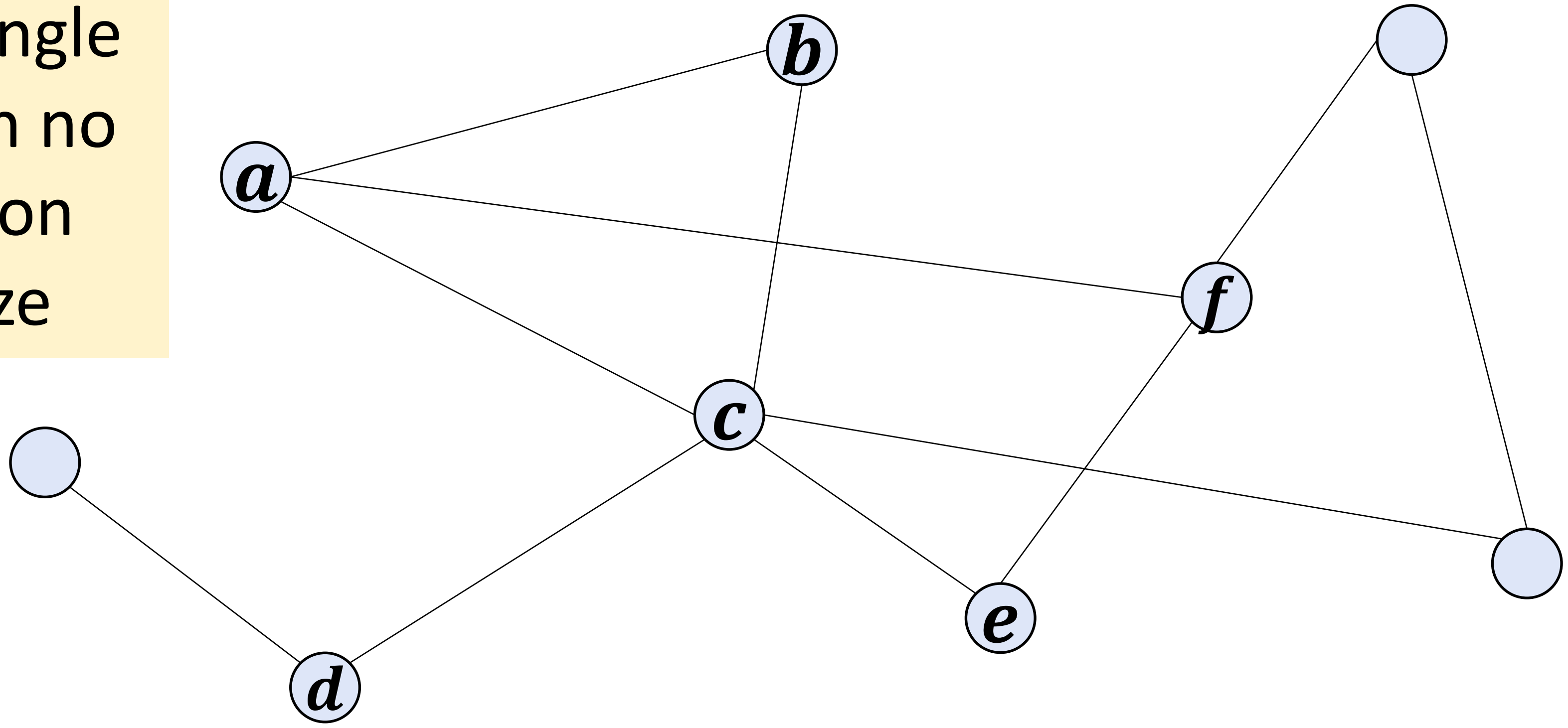
- **Can lead to very high round complexity**



Message Size Constraint for CONGEST

- **Can lead to very high round complexity**

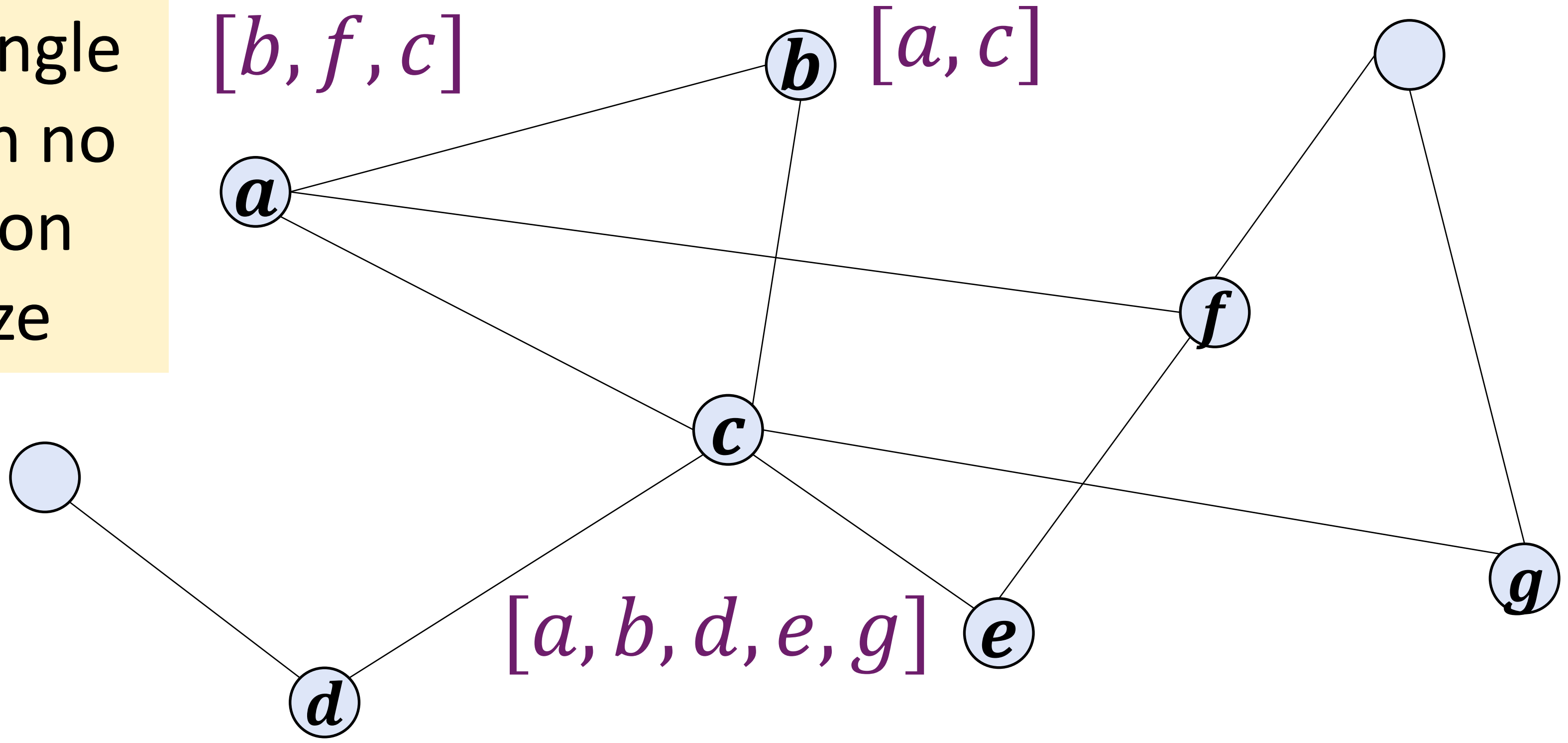
Example: Triangle Counting with no restrictions on message size



Message Size Constraint for CONGEST

- **Can lead to very high round complexity**

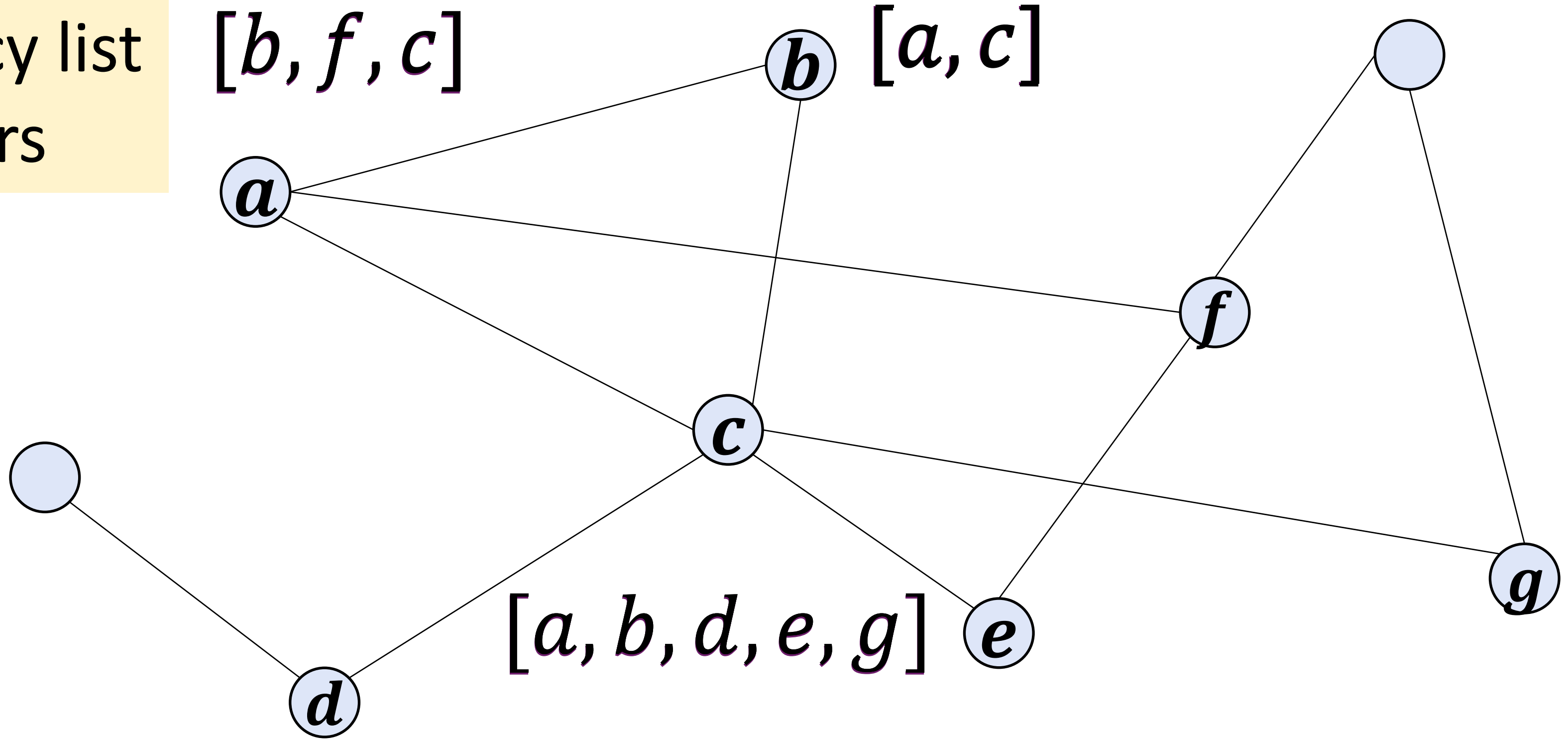
Example: Triangle Counting with no restrictions on message size



Message Size Constraint for CONGEST

- **Can lead to very high round complexity**

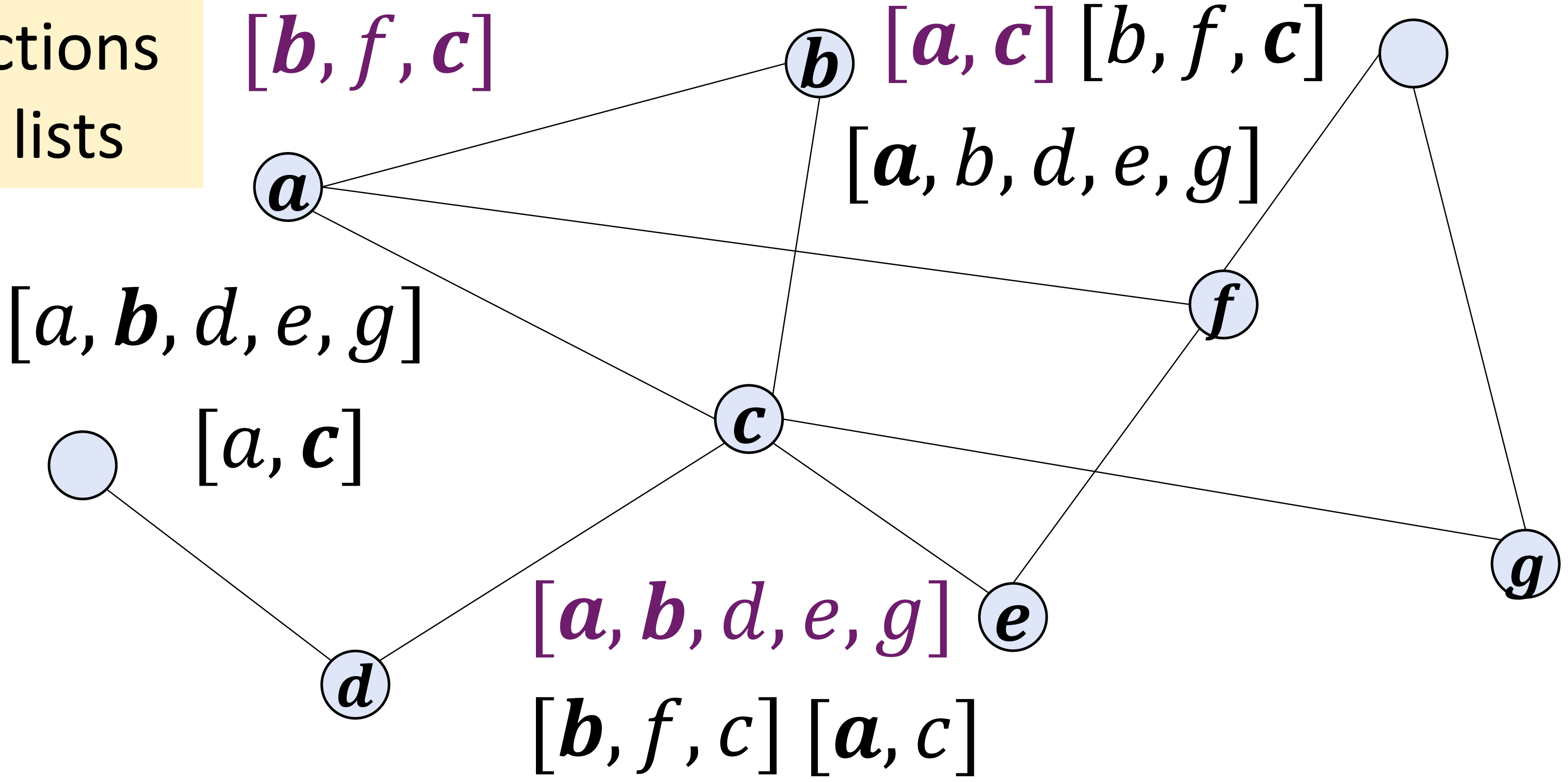
Send adjacency list to neighbors



Message Size Constraint for CONGEST

- **Can lead to very high round complexity**

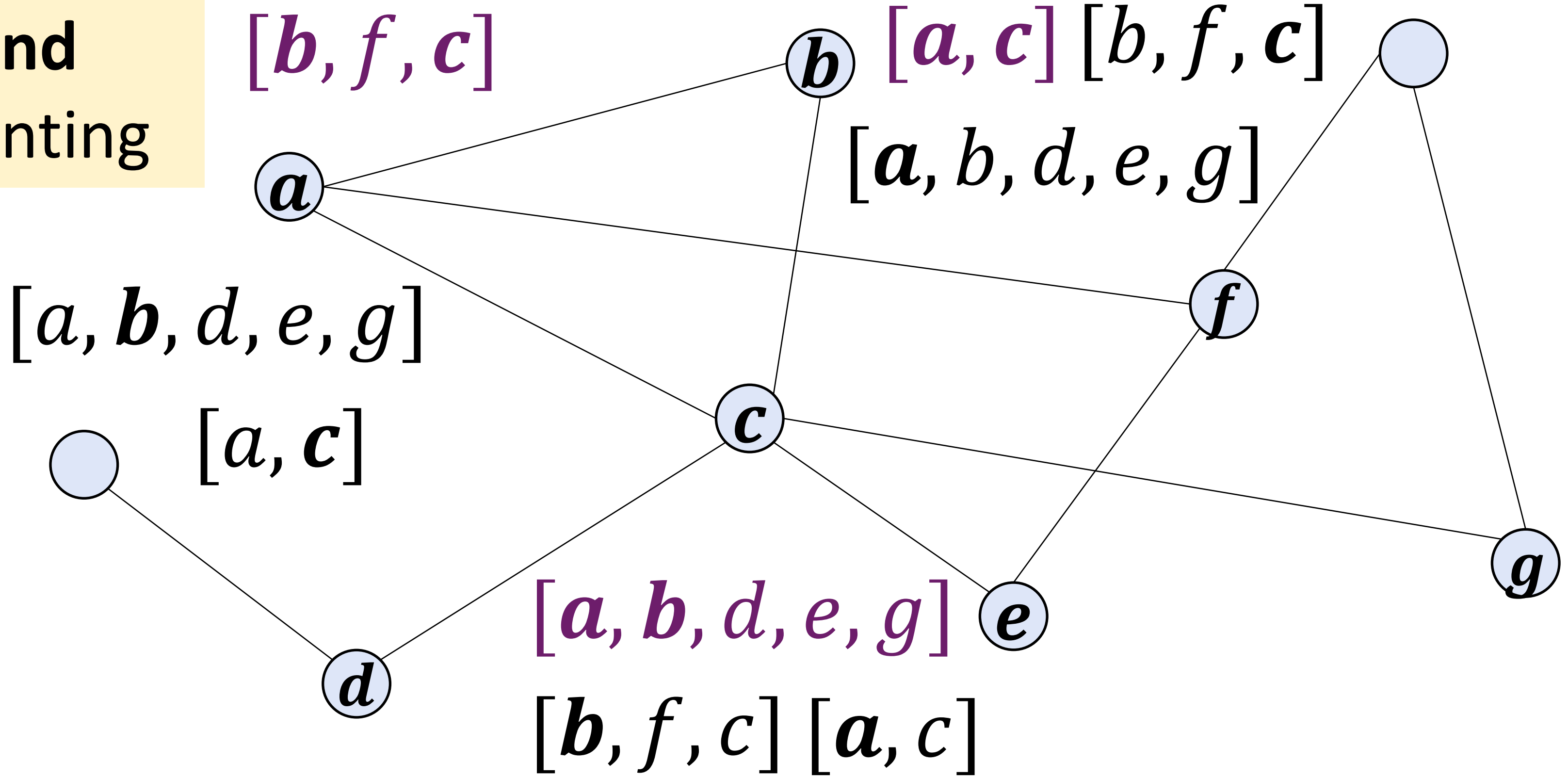
Find intersections in received lists



Message Size Constraint for CONGEST

- **Can lead to very high round complexity**

$O(1)$ round
triangle counting



Message Size Constraint for CONGEST

- **Can lead to very high round complexity**
- **Triangle counting in CONGEST:**

Message Size Constraint for CONGEST

- **Can lead to very high round complexity**
- **Triangle counting in CONGEST:**
  - $\tilde{O}\left(n^{\frac{1}{2}}\right)$ rounds [Chang, Pettie, Zhang SODA '19]
  - Large gap from LOCAL model (unrestricted message size)

Example Algorithm: Coloring Trees

- Classic $O\left(\log^*(n)\right)$ of Cole and Vishkin ‘86

Example Algorithm: Coloring Trees

- Classic $O\left(\log^*(n)\right)$ of Cole and Vishkin '86
  - Number of logarithms (base 2) to get down to 2

Example Algorithm: Coloring Trees

- Classic $O\left(\log^*(n)\right)$ of Cole and Vishkin '86
  - Number of logarithms (base 2) to get down to 2
  - $\forall x \leq 2\colon \log^*(x) \coloneqq 1;\ \ \forall x > 2\colon \log^*(x) \coloneqq 1 + \log^*\left(\log(x)\right)$

Example Algorithm: Coloring Trees

- Classic $O(\log^*(n))$ of Cole and Vishkin '86
  - Number of logarithms (base 2) to get down to 2
  - $\forall x \leq 2\colon \log^*(x) \coloneqq 1;\ \ \forall x > 2\colon \log^*(x) \coloneqq 1 + \log^*(\log(x))$
- Idea: Each node has **label of $\log(n)$ bits**
  - Each round compute label of **exponentially smaller size** that is still valid coloring

Example Algorithm: Coloring Trees

- Algorithm:
  - Initially each node has ID of color c_v of $\log n$ bits
  - Each node executes and repeats:

Example Algorithm: Coloring Trees

- Algorithm:
  - Initially each node has ID of color c_v of $\log n$ bits
  - Each node executes and repeats:
    - Send own color c_v to children

Example Algorithm: Coloring Trees

- Algorithm:
  - Initially each node has ID of color c_v of $\log n$ bits
  - Each node executes and repeats:
    - Send own color c_v to children
    - Receive parent color c_p

Example Algorithm: Coloring Trees

- Algorithm:
  - Initially each node has ID of color c_v of $\log n$ bits
  - Each node executes and repeats:
    - Send own color c_v to children
    - Receive parent color c_p
    - Write c_v and c_p in bits

Example Algorithm: Coloring Trees

- Algorithm:
  - Initially each node has ID of color c_v of $\log n$ bits
  - Each node executes and repeats:
    - Send own color c_v to children
    - Receive parent color c_p
    - Write c_v and c_p in bits
    - Let i be index of rightmost bit b where c_v and c_p differ

Example Algorithm: Coloring Trees

- Algorithm:
  - Initially each node has ID of color c_v of $\log n$ bits
  - Each node executes and repeats:
    - Send own color c_v to children
    - Receive parent color c_p
    - Write c_v and c_p in bits
    - Let i be index of rightmost bit b where c_v and c_p differ
    - **Node v's new color is $2i$ concatenated with b**

Example Algorithm: Coloring Trees

- Algorithm:
  - Initially each node has ID of color c_v of $\log n$ bits
  - Each node executes and repeats:
    - Send own color c_v to children
    - Receive parent color c_p
    - Write c_v and c_p in bits
    - Let i be index of rightmost bit b where c_v and c_p differ
    - **Node v's new color is $2i$ concatenated with b**
  - Stop when $c_v \in \{0, \ldots, 5\}$ for all nodes

Example Algorithm: Coloring Trees

- Example Run:

Example Algorithm: Coloring Trees

- Example Run:

Grandparent	0010101001
Parent	0010110001
Child	0001110001

Example Algorithm: Coloring Trees

- Example Run:

Grandparent	001010**1**001
Parent	001011**0**001
Child	0001110001

Example Algorithm: Coloring Trees

- Example Run:

Grandparent	0010101001	**1101**
Parent	0010110001	**1100**
Child	0001110001	

Example Algorithm: Coloring Trees

- Example Run:

Grandparent	0010101001	**01101**
Parent	0010110001	**01100**
Child	0001110001	**11001**

Example Algorithm: Coloring Trees

- Example Run:

Grandparent	0010101001	0110**1**	**01**
Parent	0010110001	0110**0**	**00**
Child	0001110001	1100**1**	**01**

Example Algorithm: Coloring Trees

- Why does it work?

Example Algorithm: Coloring Trees

- Why does it work?
  - Either parent/grandparent differ in a different **index** from parent/child

Example Algorithm: Coloring Trees

- Why does it work?
  - Either parent/grandparent differ in a different **index** from parent/child
    - First part is different

Example Algorithm: Coloring Trees

- Why does it work?
  - Either parent/grandparent differ in a different **index** from parent/child
    - First part is different
  - Or parent/grandparent and parent/child differ in **same index**

Example Algorithm: Coloring Trees

- Why does it work?
  - Either parent/grandparent differ in a different **index** from parent/child
    - First part is different
  - Or parent/grandparent and parent/child differ in **same index**
    - First part is same

Example Algorithm: Coloring Trees

- Why does it work?
  - Either parent/grandparent differ in a different **index** from parent/child
    - First part is different
  - Or parent/grandparent and parent/child differ in **same index**
    - First part is same
    - **Last bit differs—second part is different**

Runtime: $O(\log^*(n))$

Another Distributed Model (More Modern)

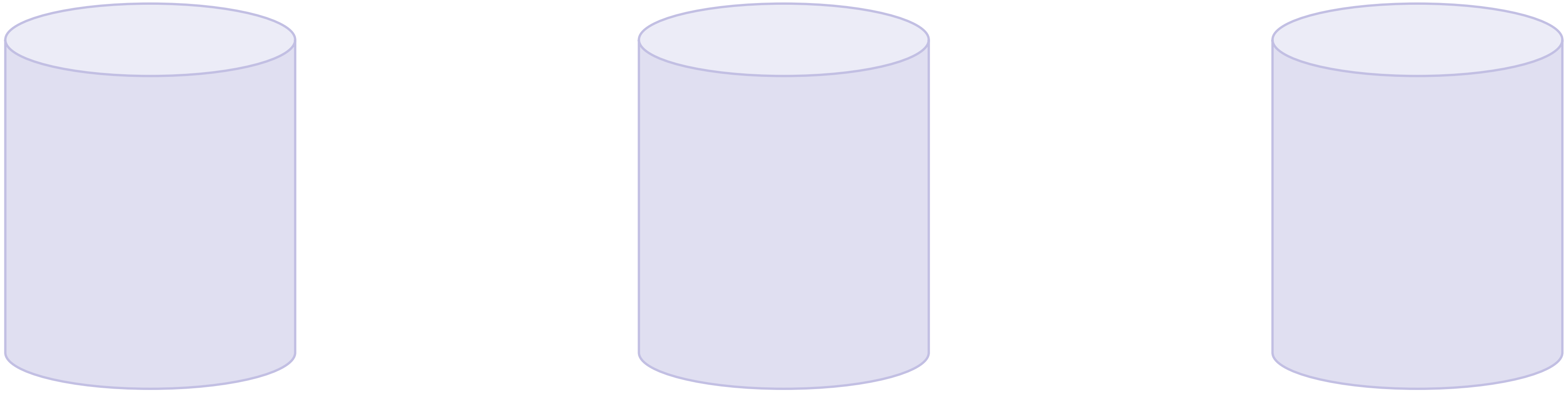
- Used by Google and other companies
- Massively parallel computation (**MPC Model**)

MPC Model Definition

- *M* machines
- Synchronous rounds

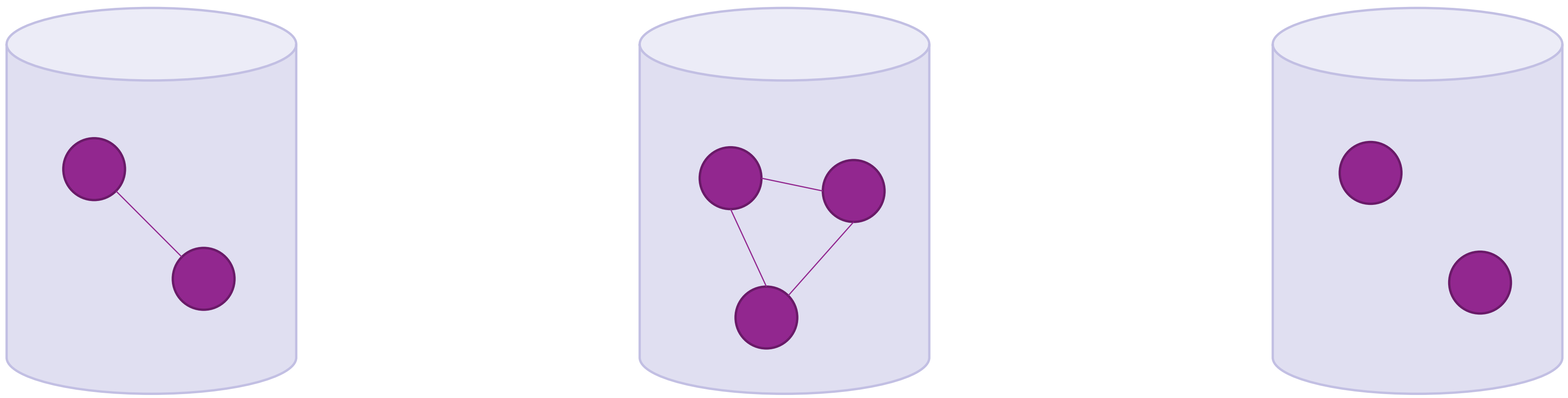
MPC Model Definition

- *M* machines
- Synchronous rounds



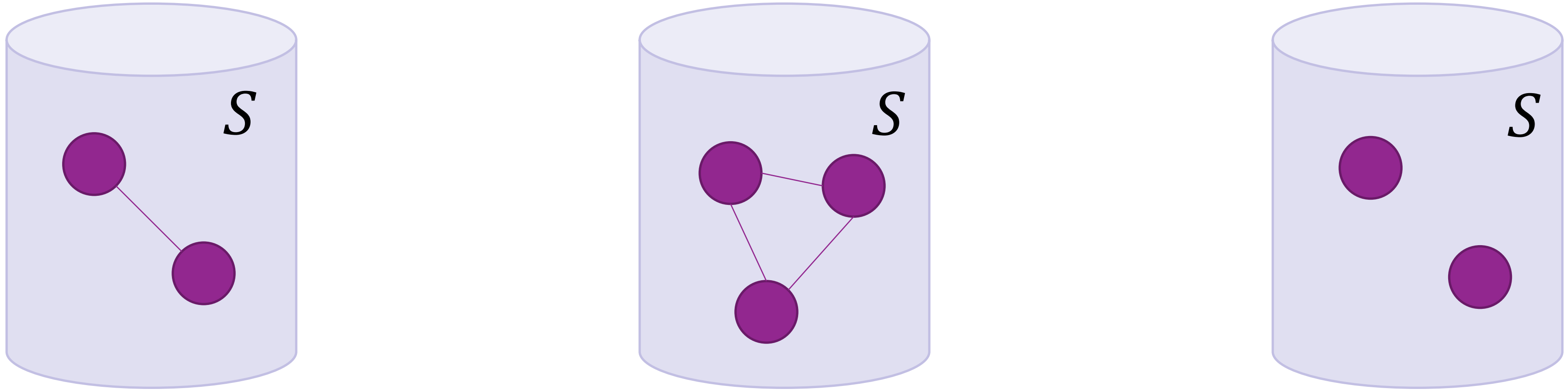
MPC Model Definition

- *M* machines
- Synchronous rounds



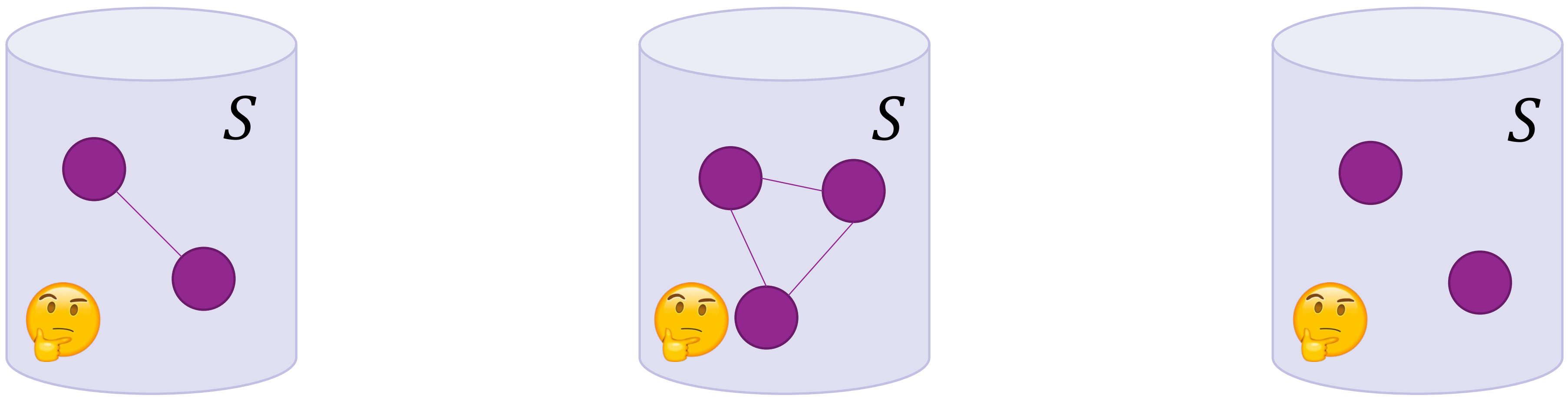
MPC Model Definition

- *M* machines
- Synchronous rounds



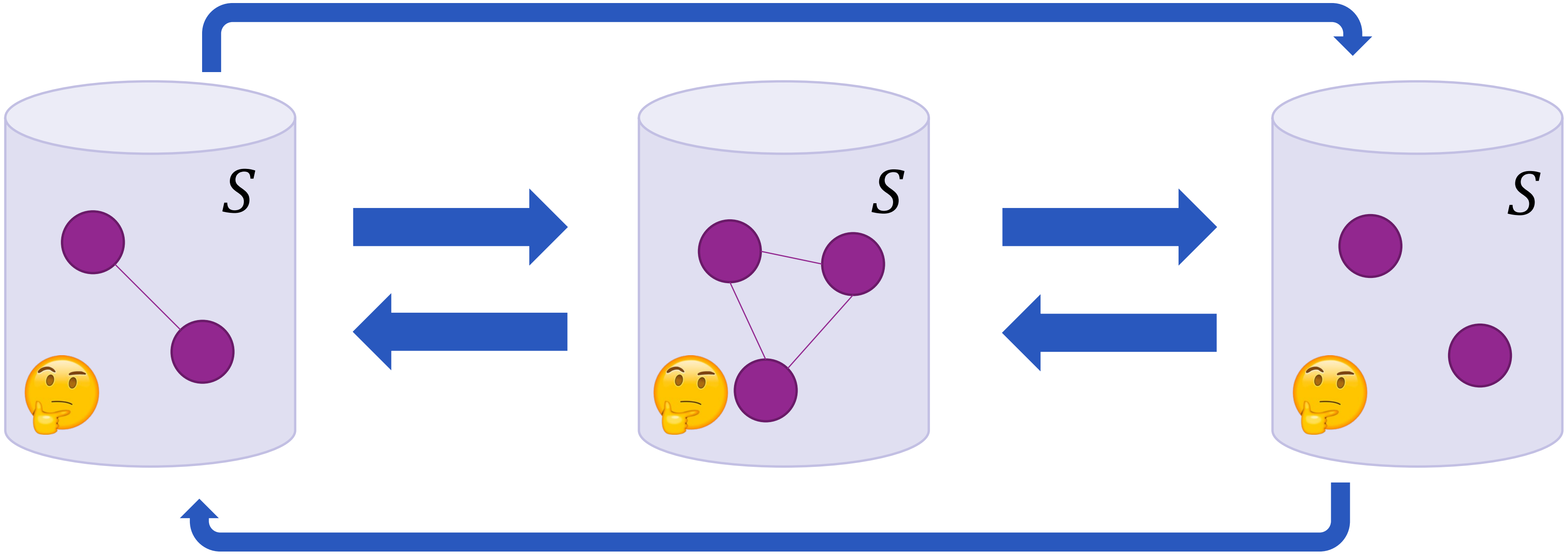
MPC Model Definition

- *M* machines
- Synchronous rounds



MPC Model Definition

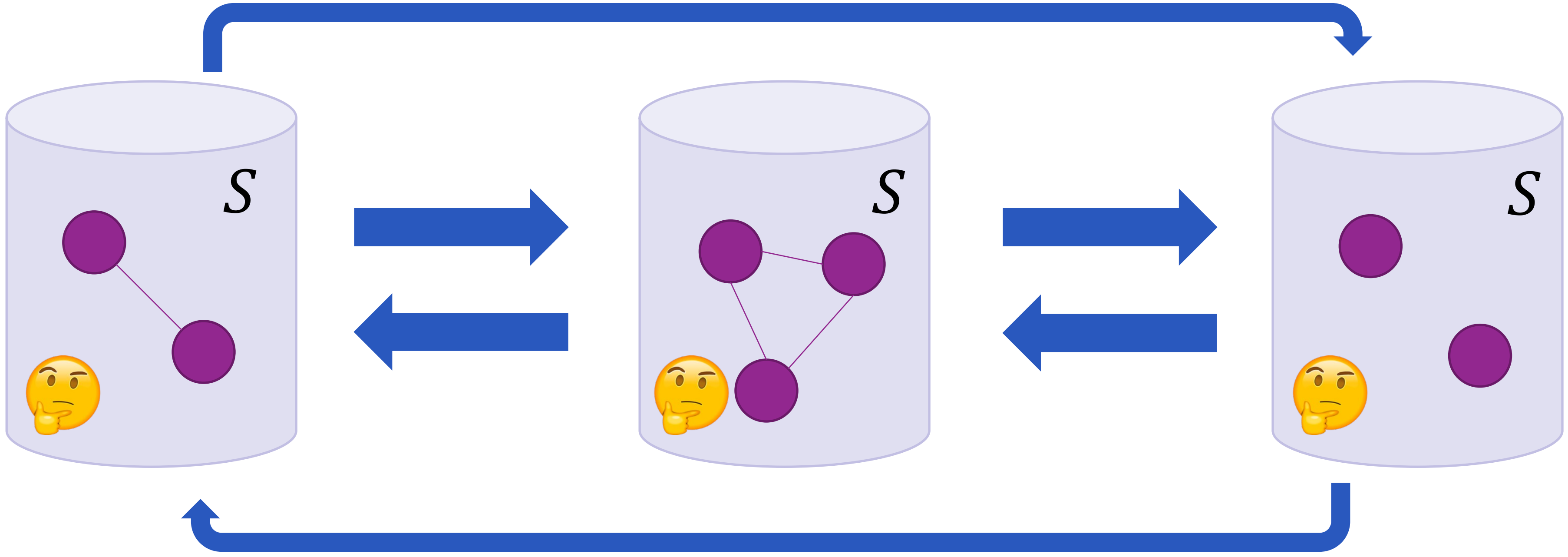
- *M* machines
- Synchronous rounds



MPC Model Definition

- M machines
- Synchronous rounds

Total Space: $\boldsymbol{M \cdot S}$

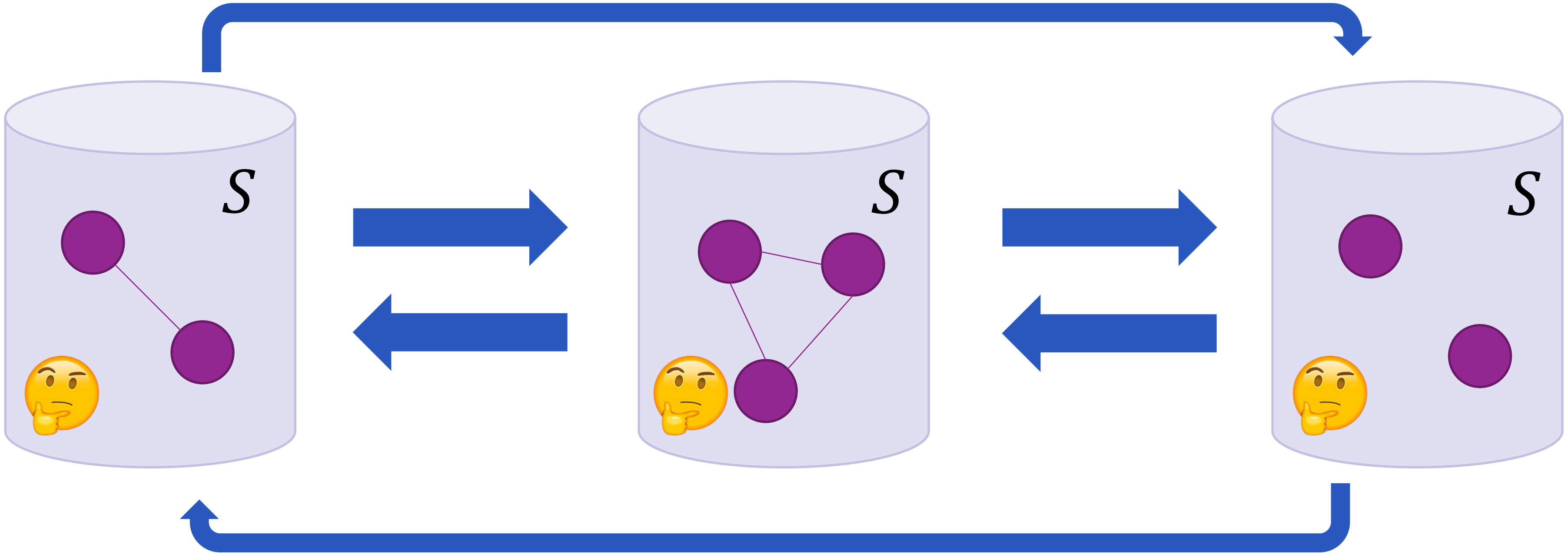


MPC Model Definition

Complexity measures:
- **Total Space**
- **Space Per Machine**
- **Rounds of communication**

- M machines
- Synchronous rounds

Total Space: $\boldsymbol{M \cdot S}$



Comparison of Models

n := number of vertices
m := number of edges

Measure	Database Theory	Algorithms
Load/Space per Machine	$L = N/p^{\frac{1}{c}}$	S
Total Space	$p \cdot L$	$T = \tilde{O}(n+m)$
Input	N	n, m, N
Rounds	r	r
# Machines	p	$M = T/S$

Space per Machine in MPC

- **Strongly sublinear memory**:
  - $S = n^{\delta}$ for some constant $\delta \in (0, 1)$

Space per Machine in MPC

- **Strongly sublinear memory**:
  - $S = n^{\delta}$ for some constant $\delta \in (0, 1)$
- **Near-linear memory**:
  - $S = \tilde{\Theta}(n)$ (ignoring $\text{poly}(\log(n))$ factors)

Space per Machine in MPC

- **Strongly sublinear memory**:
  - $S = n^{\delta}$ for some constant $\delta \in (0, 1)$
- **Near-linear memory**:
  - $S = \widetilde{\Theta}(n)$ (ignoring $\mathrm{poly}(\log(n))$ factors)
- **Strongly superlinear memory**:
  - $S = n^{1+\delta}$ for some constant $\delta > 0$

Space per Machine in MPC

- **Strongly sublinear memory**:
  - $S = n^\delta$ for some constant $\delta \in (0, 1)$
- **Near-linear memory**:
  - $S = \widetilde{\Theta}(n)$ (ignoring $\mathrm{poly}(\log(n))$ factors)
- **Strongly superlinear memory**:
  - $S = n^{1+\delta}$ for some constant $\delta > 0$

Also want: $\boldsymbol{O}(\log\log \boldsymbol{n})$ or $\boldsymbol{O}(\mathbf{1})$ rounds

Space per Machine in MPC

- **Strongly sublinear memory**:
  - $S = n^\delta$ for some constant $\delta \in (0, 1)$
- **Near-linear memory**:
  - $S = \widetilde{\Theta}(n)$ (ignoring $\text{poly}(\log(n))$ factors)
- **Strongly superlinear memory**:
  - $S = n^{1+\delta}$ for some constant $\delta > 0$

Also want: $\boldsymbol{O}(\log \log \boldsymbol{n})$ or $\boldsymbol{O}(\mathbf{1})$ rounds

Also want: $\widetilde{\boldsymbol{O}}(\boldsymbol{n} + \boldsymbol{m})$ total space

Space per Machine in MPC

- **Strongly sublinear memory**:
  - $S = n^\delta$ for some constant $\delta \in (0,1)$
- **Near-linear memory**:
  - $S = \tilde{\Theta}(n)$ (ignoring $\mathrm{poly}(\log(n))$ factors)
- **Strongly superlinear memory**:
  - $S = n^{1+\delta}$ for some constant $\delta > 0$

Also want: $\boldsymbol{O}(\log \log \boldsymbol{n})$ or $\boldsymbol{O}(\mathbf{1})$ rounds

Also want: $\tilde{\boldsymbol{O}}(\boldsymbol{n}+\boldsymbol{m})$ total space

All are **sublinear in number of edges *m*** in graph

Space per Machine in MPC

- **Strongly sublinear memory**:
  - $S = n^{\delta}$ for some constant $\delta \in (0, 1)$
- **Near-linear memory**:
  - $S = \widetilde{\Theta}(n)$ (ignoring $\text{poly}(\log(n))$ factors)
- **Strongly superlinear memory**:
  - $S = n^{1+\delta}$ for some

Also want: $O(\log \log n)$ or $O(1)$ rounds

Also want: $\widetilde{O}(n + m)$ total space

Often $\delta = \frac{1}{2}$

All are **sublinear in number of edges *m*** in graph

Graph Algorithms in MPC Model

- Matching and MIS [BBDFHKU19, BHH19, GGKMR19, CLMMOS18, NO21, FHO22, GGM22, ALT21, LKK23]
- Connectivity [ASSWZ18, BDELM19, DDKPSS19]
- Graph sparsification [GU19, CDP20]
- Vertex cover [Assadi17, GGKMR18, GJN20]
- MST and 2-edge connectivity [NO21, FHO22]
- Well-connected components [ASW18, ASW19]
- Coloring [BDHKS19, CFGUZ19]
- Subgraph counting [CC11, SV11, BELMR22]

Useful MPC Primitives in $\tilde{O}(\sqrt{N})$ **Space per Machine** and $O(1)$ **Rounds**

- **Sum of *N* integers**: given *N* integers, compute the sum

Useful MPC Primitives in $\tilde{O}(\sqrt{N})$ **Space per Machine** and $O(1)$ **Rounds**

- **Sum of *N* integers**: given *N* integers, compute the sum
- **Counting distinct elements**: given *N* integers over some universe *U* give the count of the number of distinct elements

Useful MPC Primitives in $\tilde{O}(\sqrt{N})$ **Space per Machine** and $O(1)$ **Rounds**

- **Sum of *N* integers**: given *N* integers, compute the sum
- **Counting distinct elements**: given *N* integers over some universe *U* give the count of the number of distinct elements
- **Prefix sum**: given *N* integers in order $I_1, I_2, \dots, I_n$, compute the prefix sums $\sigma_1, \sigma_2, \dots, \sigma_n$ where $\sigma_i = \sum_{j=1}^{i} I_j$

Useful MPC Primitives in $\tilde{O}(\sqrt{N})$ **Space per Machine** and $O(1)$ **Rounds**

- **Sum of *N* integers**: given *N* integers, compute the sum
- **Counting distinct elements**: given *N* integers over some universe *U* give the count of the number of distinct elements
- **Prefix sum**: given *N* integers in order $I_1, I_2, \ldots, I_n$, compute the prefix sums $\sigma_1, \sigma_2, \ldots, \sigma_n$ where $\sigma_i = \sum_{j=1}^{i} I_j$
- **Sorting**: given *N* integers, sort the integers

Round Compression

- **Goal**: Simulate **multiple rounds** of an iterative **LOCAL algorithm** with a **single MPC round**

LOCAL Model in Distributed Computing

- **Synchronous** distributed algorithm where each node is a processor/computer

LOCAL Model in Distributed Computing

- **Synchronous** distributed algorithm where each node is a processor/computer
- In **one synchronous round of communication**, each node, in order:

LOCAL Model in Distributed Computing

- **Synchronous** distributed algorithm where each node is a processor/computer
- In **one synchronous round of communication**, each node, in order:
  - Performs **local computation**

LOCAL Model in Distributed Computing

- **Synchronous** distributed algorithm where each node is a processor/computer
- In **one synchronous round of communication**, each node, in order:
  - Performs **local computation**
  - Sends a **point-to-point message** to each of its neighbors

LOCAL Model in Distributed Computing

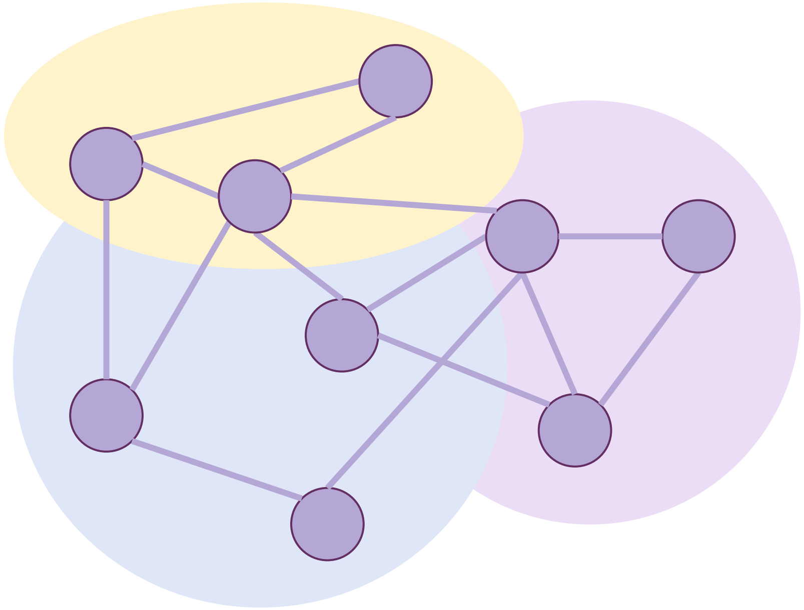
- **Synchronous** distributed algorithm where each node is a processor/computer
- In **one synchronous round of communication**, each node, in order:
  - Performs **local computation**
  - Sends a **point-to-point message** to each of its neighbors
  - **Receives a message** from each of its neighbors

Round Compression

- **Goal**: Simulate **multiple rounds** of an iterative **LOCAL algorithm** with a **single MPC round**

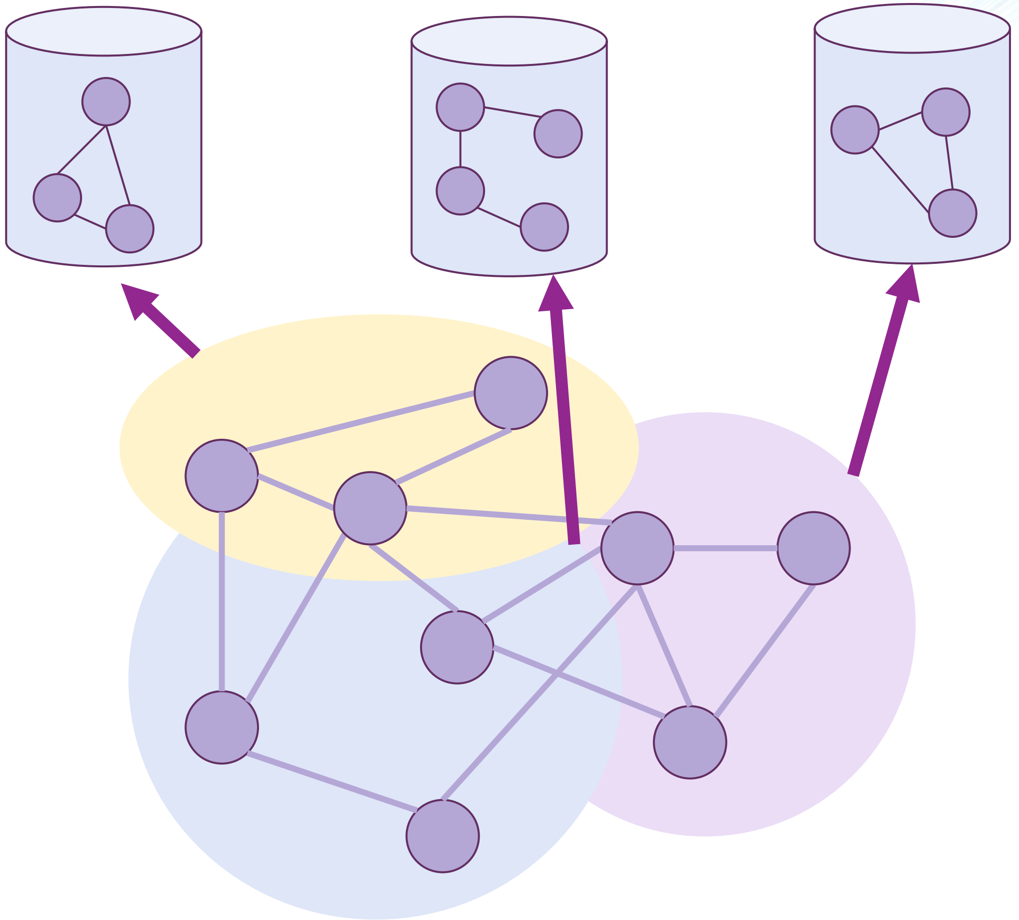
Round Compression

- **Goal**: Simulate **multiple rounds** of an iterative **LOCAL algorithm** with a **single MPC round**
- **Procedure**: Pick **appropriate subgraphs** of sufficiently small size



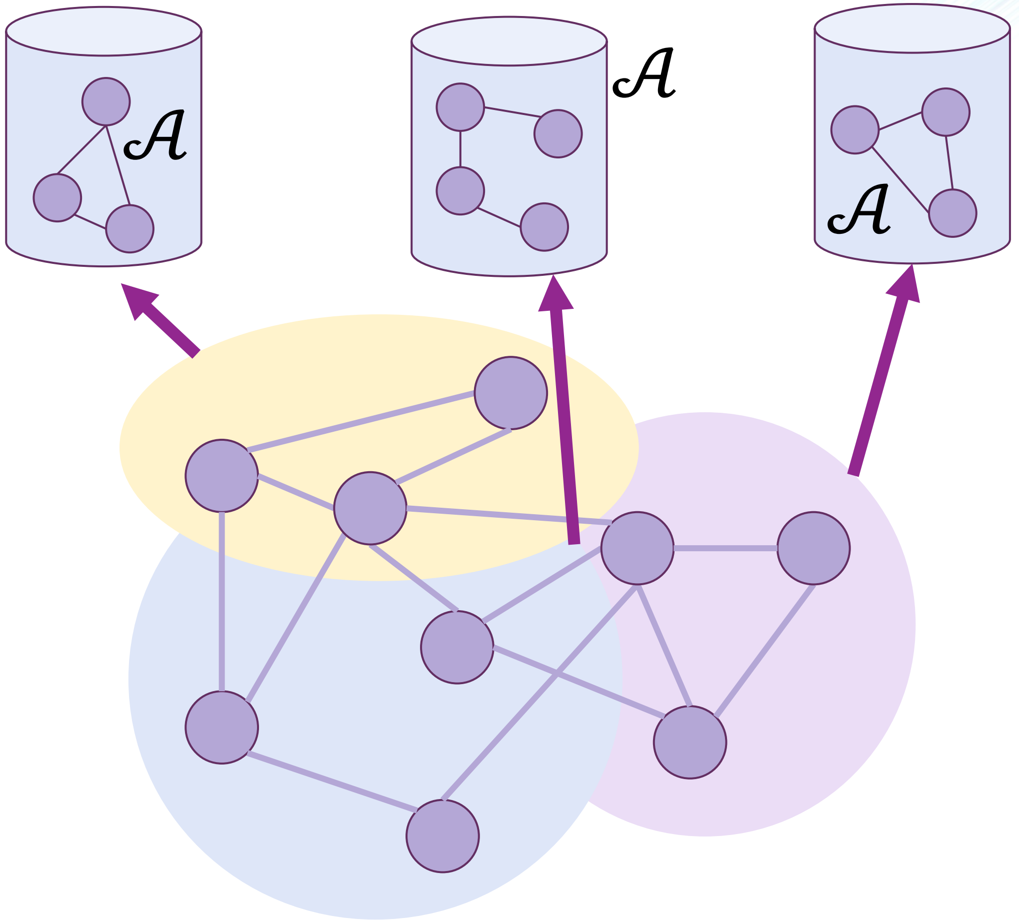
Round Compression

- **Goal**: Simulate **multiple rounds** of an iterative **LOCAL algorithm** with a **single MPC round**
- **<u>Procedure</u>**: Pick **appropriate subgraphs** of sufficiently small size
- Send each subgraph to **one machine**



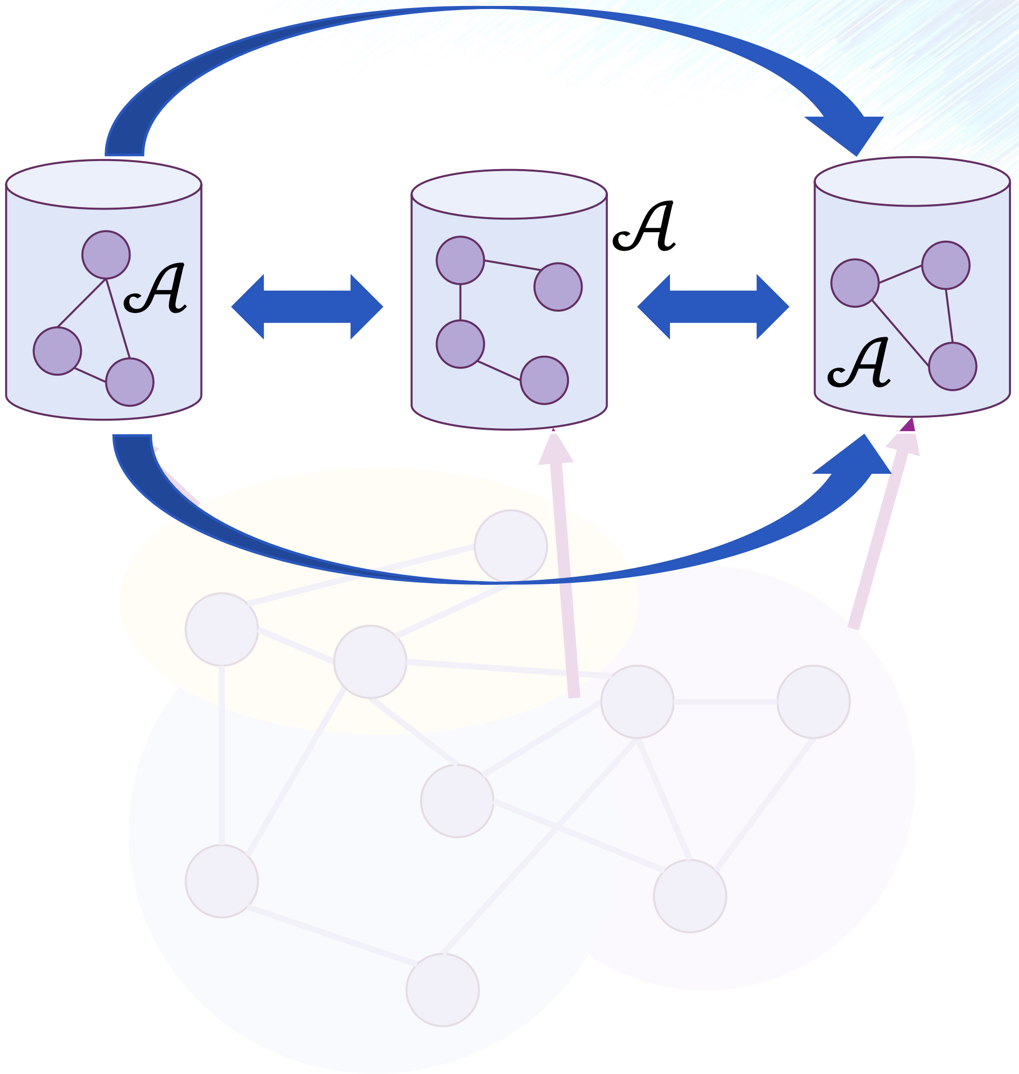
Round Compression

- **Goal**: Simulate **multiple rounds** of an iterative **LOCAL algorithm** with a **single MPC round**
- **Procedure**: Pick **appropriate subgraphs** of sufficiently small size
- Send each subgraph to **one machine**
- Simulate **LOCAL algorithm** $\mathcal{A}$ on each machine



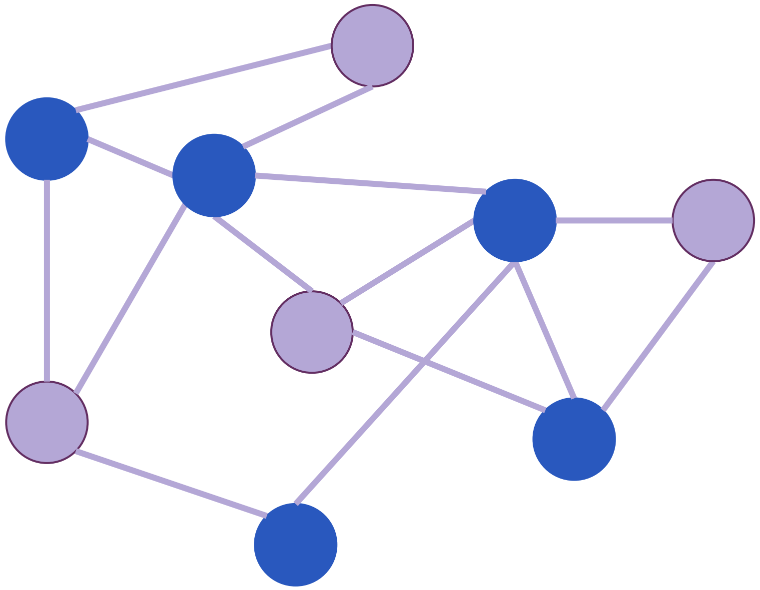
Round Compression

- **Goal**: Simulate **multiple rounds** of an iterative **LOCAL algorithm** with a **single MPC round**
- **Procedure**: Pick **appropriate subgraphs** of sufficiently small size
- Send each subgraph to **one machine**
- Simulate **LOCAL algorithm** $\mathcal{A}$ on each machine
- Each machine **sends results of simulation**



Minimum Vertex Cover

- Each edge in graph is **covered** by an endpoint
- Find the **minimum number of endpoints** that cover every edge



Simplified $(2 + \varepsilon)$-Approximate Vertex Cover

$(2 + \varepsilon)$-Approximate Vertex Cover
[Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC ’18]

Near-linear space per machine in $\boldsymbol{O(\log \log n)}$ **rounds**

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

$(2+\varepsilon)$-Approximate Vertex Cover
[Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Near-linear space per machine in $\boldsymbol{O(\log\log n)}$ **rounds**

Simplified version

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- LOCAL Algorithm based on Primal-Dual Method:

Primal

$$\min \sum_{v \in V} x_v$$

s.t. $\forall e = (u,v) \in E \quad x_u + x_v \geq 1$

$\forall v \in V \quad x_v \geq 0$

Dual

$$\max \sum_{e \in E} y_e$$

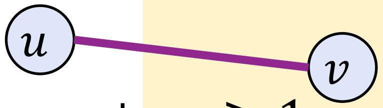
s.t. $\forall v \in V \quad \sum_{e: v \in e} y_e \leq 1$

$\forall e \in E \quad y_e \geq 0$

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC ’18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- LOCAL Algorithm based on Primal-Dual Method:



$x_u + x_v \geq 1$

Primal

$$\min \sum_{v \in V} x_v$$

s.t. $\forall e = (u, v) \in E \quad x_u + x_v \geq 1$

$\forall v \in V \quad x_v \geq 0$

Dual

$$\max \sum_{e \in E} y_e$$

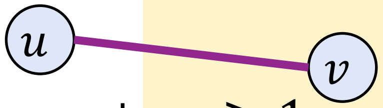
s.t. $\forall v \in V \quad \sum_{e: v \in e} y_e \leq 1$

$\forall e \in E \quad y_e \geq 0$

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC ’18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- LOCAL Algorithm based on Primal-Dual Method:



Primal

$$\min \sum_{v \in V} x_v$$

s.t. $\forall e = (u, v) \in E \quad x_u + x_v \geq 1$

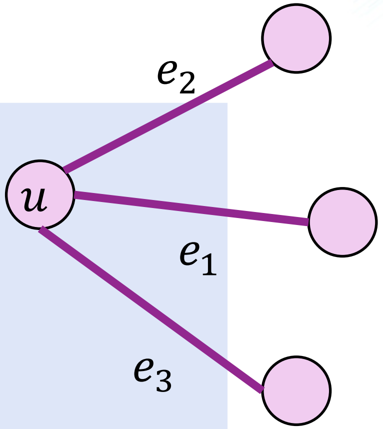
$\forall v \in V \quad x_v \geq 0$

Dual

$$\max \sum_{e \in E} y_e$$

s.t. $\forall v \in V \quad \sum_{e: v \in e} y_e \leq 1$

$\forall e \in E \quad y_e \geq 0$



Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC ’18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- LOCAL Algorithm based on Primal-Dual Method:

All nodes covered by at least one endpoint

Primal

$$\min \sum_{v \in V} x_v$$

s.t. $\forall e = (u, v) \in E \quad x_u + x_v \geq 1$

$\forall v \in V \quad x_v \geq 0$

Dual

$$\max \sum_{e \in E} y_e$$

s.t. $\forall v \in V \quad \sum_{e: v \in e} y_e \leq 1$

$\forall e \in E \quad y_e \geq 0$

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC ’18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- LOCAL Algorithm based on Primal-Dual Method:

All nodes covered by at least one endpoint

Primal

$$\min \sum_{v \in V} x_v$$

s.t. $\forall e = (u, v) \in E \quad x_u + x_v \geq 1$

$\forall v \in V \quad x_v \geq 0$

Dual

$$\max \sum_{e \in E} y_e$$

s.t. $\forall v \in V \quad \sum_{e: v \in e} y_e \leq 1$

$\forall e \in E \quad y_e \geq 0$

Fractional matching of the edges

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- LOCAL Algorithm based on Primal-Dual Method:
  - Initially **set $y_e = \frac{1}{\Delta}$**

Primal

$$\min \sum_{v \in V} x_v$$

s.t. $\forall e = (u, v) \in E \quad x_u + x_v \geq 1$

$\forall v \in V \quad x_v \geq 0$

Dual

$$\max \sum_{e \in E} y_e$$

s.t. $\forall v \in V \quad \sum_{e: v \in e} y_e \leq 1$

$\forall e \in E \quad y_e \geq 0$

n := number of vertices
m := number of edges
Δ := max degree

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC ’18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- LOCAL Algorithm based on Primal-Dual Method:
  - Initially **set $y_e = \frac{1}{\Delta}$**
  - Repeat for **iteration t** until all edges frozen:

Primal

$$\min \sum_{v \in V} x_v$$

s.t. $\forall e = (u, v) \in E \quad x_u + x_v \geq 1$

$\forall v \in V \quad x_v \geq 0$

Dual

$$\max \sum_{e \in E} y_e$$

s.t. $\forall v \in V \quad \sum_{e: v \in e} y_e \leq 1$

$\forall e \in E \quad y_e \geq 0$

n := number of vertices
m := number of edges
Δ := max degree

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- LOCAL Algorithm based on Primal-Dual Method:
  - Initially **set** $y_e = \frac{1}{\Delta}$
  - Repeat for **iteration** t until all edges frozen:
    - **Freeze vertex** v and all adjacent edges **if** $\sum_{v \in e} y_e \geq 1 - 2\varepsilon$

Primal

$$\min \sum_{v \in V} x_v$$

s.t. $\forall e = (u, v) \in E \quad x_u + x_v \geq 1$

$\forall v \in V \quad x_v \geq 0$

Dual

$$\max \sum_{e \in E} y_e$$

s.t. $\forall v \in V \quad \sum_{e: v \in e} y_e \leq 1$

$\forall e \in E \quad y_e \geq 0$

n := number of vertices
m := number of edges
Δ := max degree

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- LOCAL Algorithm based on Primal-Dual Method:
  - Initially **set $y_e = \frac{1}{\Delta}$**
  - Repeat for **iteration t** until all edges frozen:
    - **Freeze vertex v** and all adjacent edges **if $\sum_{v \in e} y_e \geq 1 - 2\varepsilon$**
    - For each **active** (non-frozen) edge, **set $y_e \leftarrow \frac{y_e}{1-\varepsilon}$**

<u>Primal</u>

$$\min \sum_{v \in V} x_v$$

s.t. $\forall e = (u,v) \in E \quad x_u + x_v \geq 1$

$\forall v \in V \quad x_v \geq 0$

<u>Dual</u>

$$\max \sum_{e \in E} y_e$$

s.t. $\forall v \in V \quad \sum_{e: v \in e} y_e \leq 1$

$\forall e \in E \quad y_e \geq 0$

n := number of vertices
m := number of edges
Δ := max degree

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- LOCAL Algorithm based on Primal-Dual Method:
  - Initially **set** $y_e = \frac{1}{\Delta}$
  - Repeat for **iteration** t until all edges frozen:
    - **Freeze vertex** v and all adjacent edges **if** $\sum_{v \in e} y_e \geq 1 - 2\varepsilon$
    - For each **active** (non-frozen) edge, **set** $y_e \leftarrow \frac{y_e}{1-\varepsilon}$
  - **Set of frozen vertices is cover**

<u>Primal</u>

$$\min \sum_{v \in V} x_v$$

s.t. $\forall e = (u, v) \in E \quad x_u + x_v \geq 1$

$\forall v \in V \quad x_v \geq 0$

<u>Dual</u>

$$\max \sum_{e \in E} y_e$$

s.t. $\forall v \in V \quad \sum_{e: v \in e} y_e \leq 1$

$\forall e \in E \quad y_e \geq 0$

n := number of vertices
m := number of edges
Δ := max degree

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- LOCAL Algorithm based on Primal-Dual Method:
  - Initially **set $y_e = \frac{1}{\Delta}$**
  - Repeat for **iteration t** until all edges frozen:
    - **Freeze vertex v** and all adjacent edges **if $\sum_{v \in e} y_e \geq 1 - 2\varepsilon$**
    - For each **active** (non-frozen) edge, **set $y_e \leftarrow \frac{y_e}{1-\varepsilon}$**
  - **Set of frozen vertices is cover**

Primal

$$\min \sum_{v \in V} x_v$$

s.t. $\forall e = (u, v) \in E \quad x_u + x_v \geq 1$

$\forall v \in V \quad x_v \geq 0$

Dual

$$\max \sum_{e \in E} y_e$$

s.t. $\forall v \in V \quad \sum_{e: v \in e} y_e \leq 1$

$\forall e \in E \quad y_e \geq 0$

$O(\log n)$ rounds

n := number of vertices
m := number of edges
Δ := max degree

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- Assume maximum degree $\boldsymbol{\Delta = O\left(n^{\frac{1}{9}}\right)}$

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- Assume maximum degree $\boldsymbol{\Delta = O\left(n^{\frac{1}{9}}\right)}$
    - Partition the graph into subgraphs of **radius 8**

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- Assume maximum degree $\boldsymbol{\Delta = O\left(n^{\frac{1}{9}}\right)}$
  - Partition the graph into subgraphs of **radius 8**
  - Give the **entirety of each subgraph** to a single machine

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC ’18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- Assume maximum degree $\boldsymbol{\Delta = O\left(n^{\frac{1}{9}}\right)}$
  - Partition the graph into subgraphs of **radius 8**
  - Give the **entirety of each subgraph** to a single machine
  - **Run the LOCAL algorithm** on each machine for $\boldsymbol{\frac{\log_{\frac{1}{1-\varepsilon}}(\Delta)}{10}}$ **rounds**

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- Assume maximum degree $\boldsymbol{\Delta = O\left(n^{\frac{1}{9}}\right)}$
  - Partition the graph into subgraphs of **radius 8**
  - Give the **entirety of each subgraph** to a single machine
  - **Run the LOCAL algorithm** on each machine for $\boldsymbol{\frac{\log_{\frac{1}{1-\varepsilon}}(\Delta)}{10}}$ **rounds**
  - Find new graph after **removing frozen vertices and edges**

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC ’18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- Assume maximum degree $\boldsymbol{\Delta = O\left(n^{\frac{1}{9}}\right)}$
  - Partition the graph into subgraphs of **radius 8**
  - Give the **entirety of each subgraph** to a single machine
  - **Run the LOCAL algorithm** on each machine for $\frac{\log_{\frac{1}{1-\varepsilon}}(\Delta)}{10}$ **rounds**
  - Find new graph after **removing frozen vertices and edges**
  - Set **new radius to 9** and repeat above until graph can fit into one machine

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2 + \varepsilon)$-Approximate Vertex Cover

- Assume maximum degree $\Delta = O\left(n^{\frac{1}{9}}\right)$
  - Partition the graph into subgraphs of **radius 8**
  - Give the **entiret** single machine
  - **Run the LOCAL** chine for $\frac{\log_{\frac{1}{1-\varepsilon}}(\Delta)}{10}$ **rounds**
  - Find new graph after **removing frozen vertices and edges**
  - Set **new radius to 9** and repeat above until graph can fit into one machine

Why does it work?

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- Assume maximum degree $\boldsymbol{\Delta = O\left(n^{\frac{1}{9}}\right)}$
  - Partition the graph into subgraphs of **radius 8**
  - Give the **entirety of each subgraph** to a single machine
  - **Run the LOCAL algorithm** on each machine for $\boldsymbol{\frac{\log_{\frac{1}{1-\varepsilon}}(\Delta)}{10}}$ **rounds**
  - Find new graph after **removing frozen vertices and edges**
  - Set **new radius to 9** and repeat above until graph can fit into one machine

In sublinear memory $\boldsymbol{O\left(n^{\frac{8}{9}}\right)}$

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- Assume maximum degree $\boldsymbol{\Delta = o\left(n^{\frac{1}{9}}\right)}$
  - Partition the graph into subgraphs of **radius 8**
  - Give the **entirety of each subgraph** to a single machine
  - **Run the LOCAL algorithm** on each machine for $\frac{\log_{\frac{1}{1-\varepsilon}}(\Delta)}{10}$ **rounds**
  - Find new graph after **removing frozen vertices and edges**
  - Set **new radius to 9** and repeat above until graph can fit into one machine

In sublinear memory $\boldsymbol{O\left(n^{\frac{8}{9}}\right)}$

Minimum weight on an edge becomes $\boldsymbol{\Delta^{-0.9}}$

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

Weight on each edge:

$$\frac{1}{\Delta} \cdot \left(\frac{1}{1-\varepsilon}\right)^{\log_{\frac{1}{1-\varepsilon}}(\Delta)/10} = \frac{1}{\Delta} \cdot \Delta^{\frac{1}{10}} = \Delta^{-0.9}$$

In sublinear memory $O\left(n^{\frac{8}{9}}\right)$

…ius 8

… single machine

- **Run the LOCAL algorithm** on each machine for $\frac{\log_{\frac{1}{1-\varepsilon}}(\Delta)}{10}$ rounds

Minimum weight on an edge becomes $\Delta^{-0.9}$

- Find new graph after **removing frozen vertices and edges**
- Set **new radius to 9** and repeat above until graph can fit into one machine

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

Weight on each edge after i-iteration:

$$\Delta^{-0.9^{i}}$$

In sublinear memory $O\left(n^{\frac{8}{9}}\right)$

ius 8

single machine

- **Run the LOCAL algorithm** on each machine for $\frac{\log_{\frac{1}{1-\varepsilon}}(\Delta)}{10}$ **rounds**

Minimum weight on an edge becomes $\Delta^{-0.9}$

- Find new graph after **removing frozen vertices and edges**
- Set **new radius to 9** and repeat above until graph can fit into one machine

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

Maximum degree after i-iteration:

$$1/(\Delta^{-0.9^i}) = \Delta^{0.9^i}$$

ius 8

single machine

In sublinear memory $O\left(n^{\frac{8}{9}}\right)$

- **Run the LOCAL algorithm** on each machine for $\frac{\log_{\frac{1}{1-\varepsilon}}(\Delta)}{10}$ rounds

Minimum weight on an edge becomes $\Delta^{-0.9}$

- Find new graph after **removing frozen vertices and edges**
- Set **new radius to 9** and repeat above until graph can fit into one machine

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

- Assu[...] $O\left(\frac{1}{\cdot}\right)$
  - Pa[...]f **radius 8**
  - Giv[...] **entirety of each subgraph** to a single machine
  - **Run the LOCAL algorithm** on each machine for $\frac{\log_{\frac{1}{1-\varepsilon}}(\Delta)}{10}$ rounds
- Find new graph after **removing frozen vertices and edges**
- Set **new radius to 9** and repeat above until graph can fit into one machine

$O(\log \log n)$ rounds

In sublinear memory $O\left(n^{\frac{8}{9}}\right)$

Minimum weight on an edge becomes $\Delta^{-0.9}$

Maximum degree of active vertices is $\Delta^{0.9}$

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC ’18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

Round compression: $O(\log n)$ LOCAL → $O(\log\log n)$ MPC

- Assume
  - Parti
  - rounds
  - Find new grap ... es and edges
  - Set **new radiu** ... raph can fit into one machine

Removing assumption requires **random partition of vertices** + other techniques

$$O\left(\log\log\left(\frac{m}{n}\right)\right) \text{ rounds}$$

[Ghaffari, Jin, Nilis SPAA '20]

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

• Assume

• Parti

Round compression: $O(\log n)$ LOCAL $\rightarrow$ $O(\log \log n)$ MPC

Removing assumption requires **random partition of vertices** + other techniques

rounds

es and edges

raph can fit into

Fine-grained lower bound for **sublinear space** and $o(\log \log n)$ rounds! [Ghaffari, Kuhn, Uitto FOCS '19]

$O\left(\log \log \left(\frac{m}{n}\right)\right)$ **rounds**

[Ghaffari, Jin, Nilis SPAA '20]

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]

Simplified $(2+\varepsilon)$-Approximate Vertex Cover

• Assume

• Parti

rounds

es and edges

raph can fit into

Round compression: $O(\log n)$ LOCAL → $O(\log\log n)$ MPC

Removing assumption requires **random partition of vertices** + other techniques

Fine-grained lower bound for **sublinear space** and $o(\log\log n)$ rounds! [Ghaffari, Kuhn, Uitto FOCS '19]

$O\left(\log\log\left(\frac{m}{n}\right)\right)$ rounds

[Ghaffari, Jin, Nilis SPAA '20]

Very Simplified Version of [Ghaffari, Gouleakis, Konrad, Mitrović, Rubinfeld PODC '18]