# Breaking the Linear-Memory Barrier in Massively Parallel Computing

## MIS on Trees with Strongly Sublinear Memory

Sebastian Brandt, **Manuela Fischer**, Jara Uitto
ETH Zurich

## Model:

# Massively Parallel Computing (MPC)

Model:

# Massively Parallel Computing (MPC)

parallel computing framework

## Model:

# Massively Parallel Computing (MPC)

parallel computing framework
inspired by MapReduce

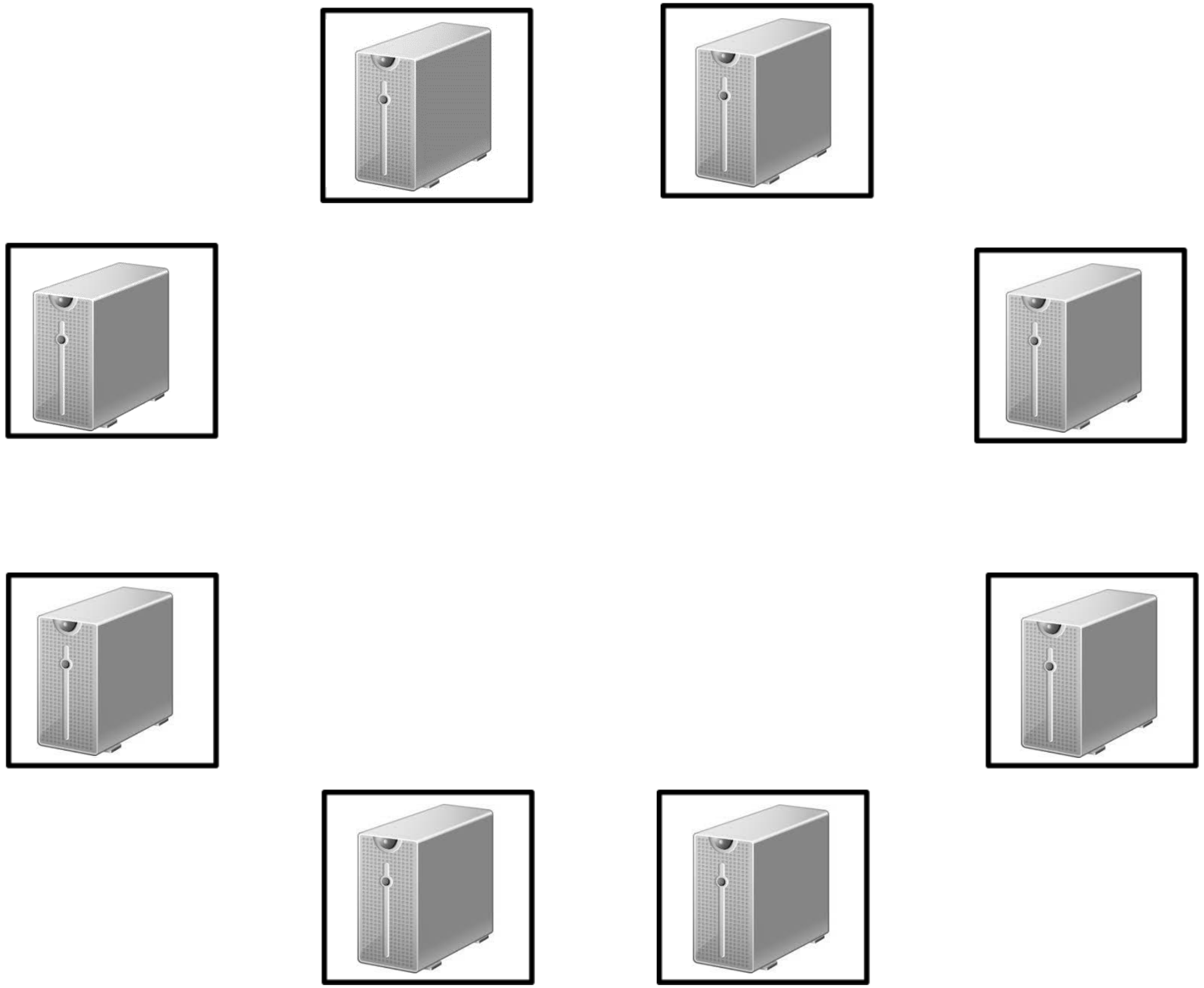
# Model:

# **Massively Parallel Computing (MPC)**

parallel computing framework
inspired by MapReduce

*Karloff, Suri, Vassilvitskii* [SODA'10]

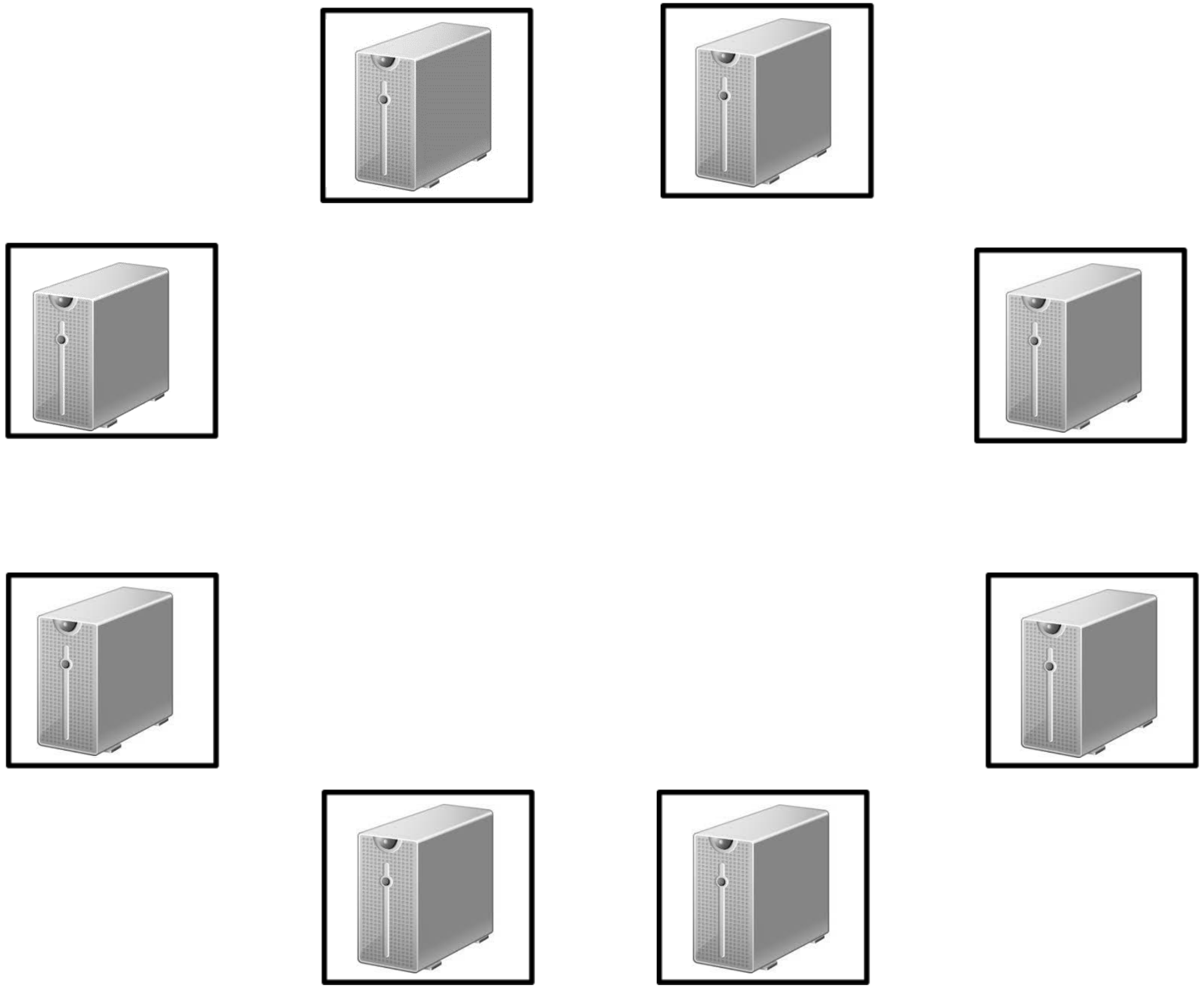
# Massively Parallel Computing (MPC) Model



$M$ machines

$S$ memory per machine

# Massively Parallel Computing (MPC) Model

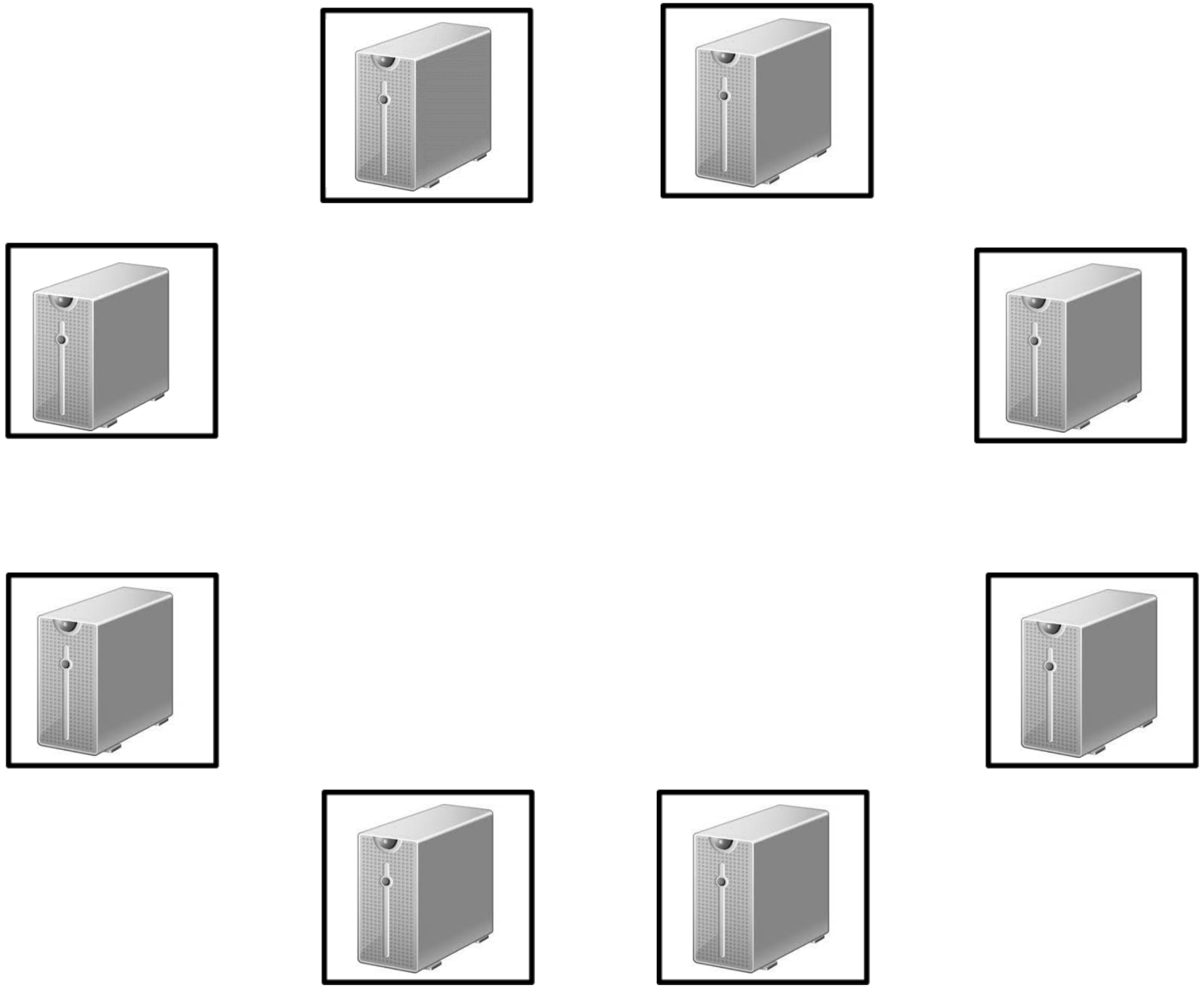


$M$ **machines**

$S$ **memory per machine**

**Synchronous Rounds**

# Massively Parallel Computing (MPC) Model



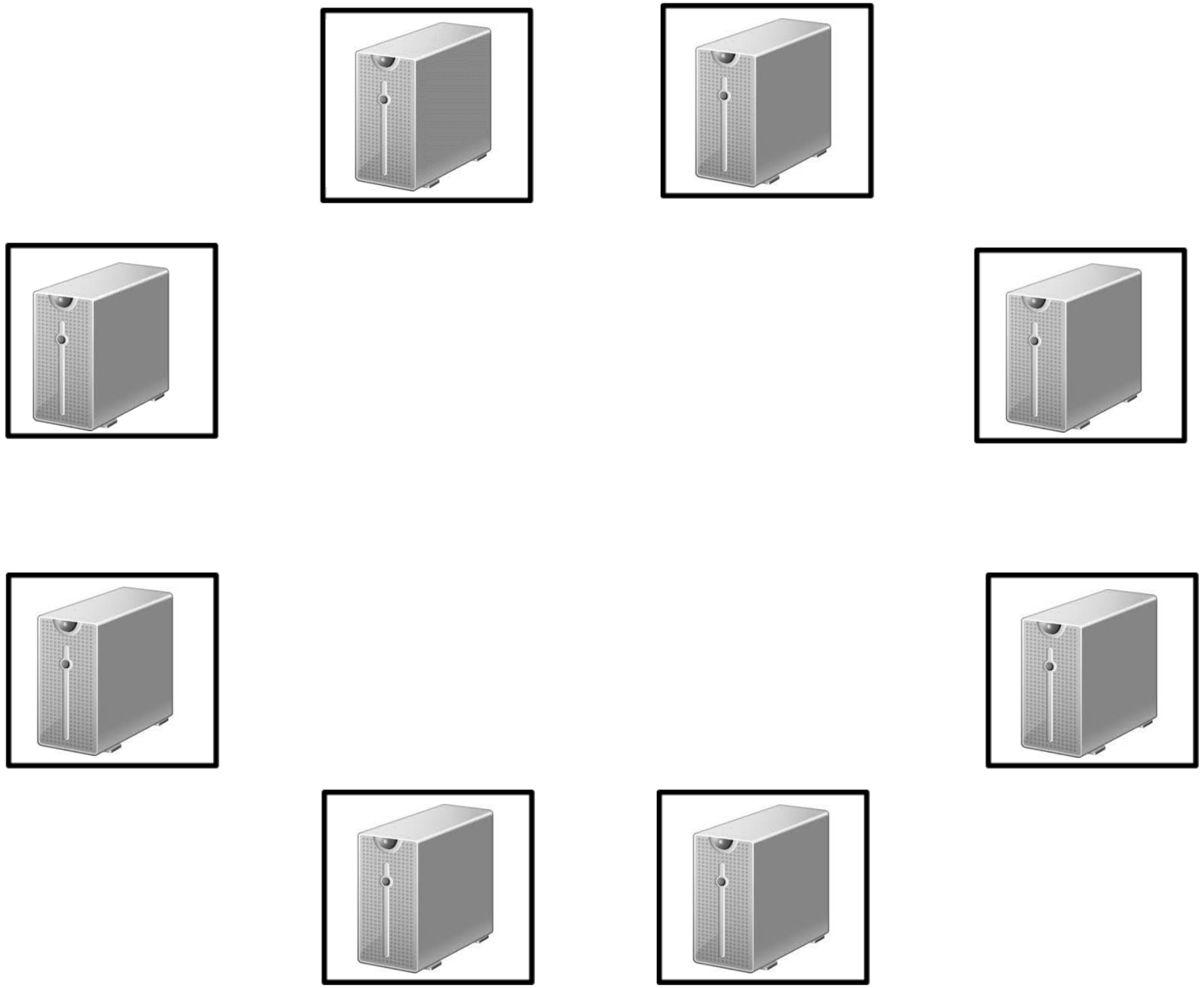
$M$ **machines**

$S$ **memory per machine**

**Synchronous Rounds**

1. **Local Computation**
   at every machine

# Massively Parallel Computing (MPC) Model



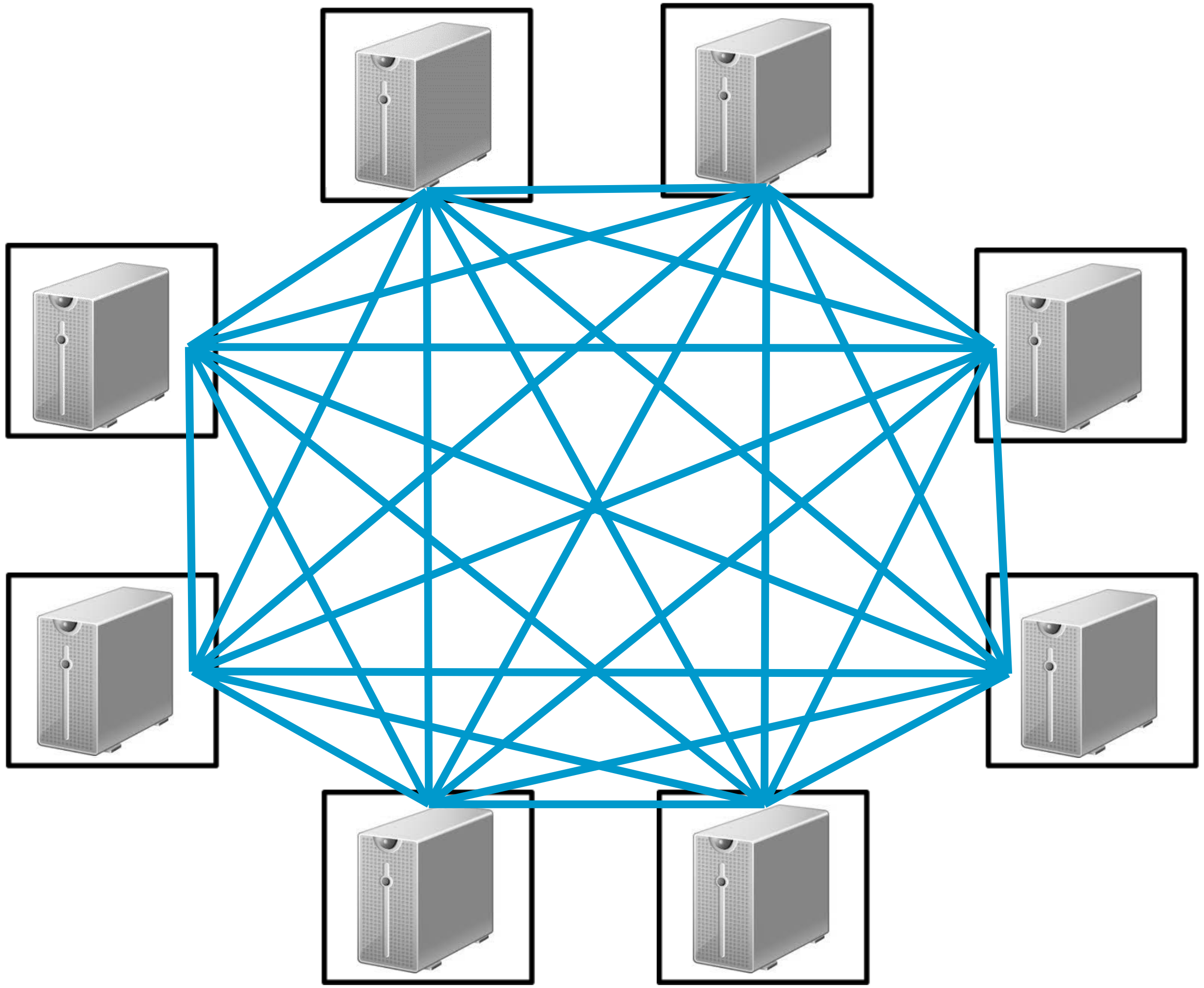
**$M$ machines**

**$S$ memory per machine**

**Synchronous Rounds**

1. **Local Computation**
   at every machine
2. **Global Communication**
   between machines

# Massively Parallel Computing (MPC) Model



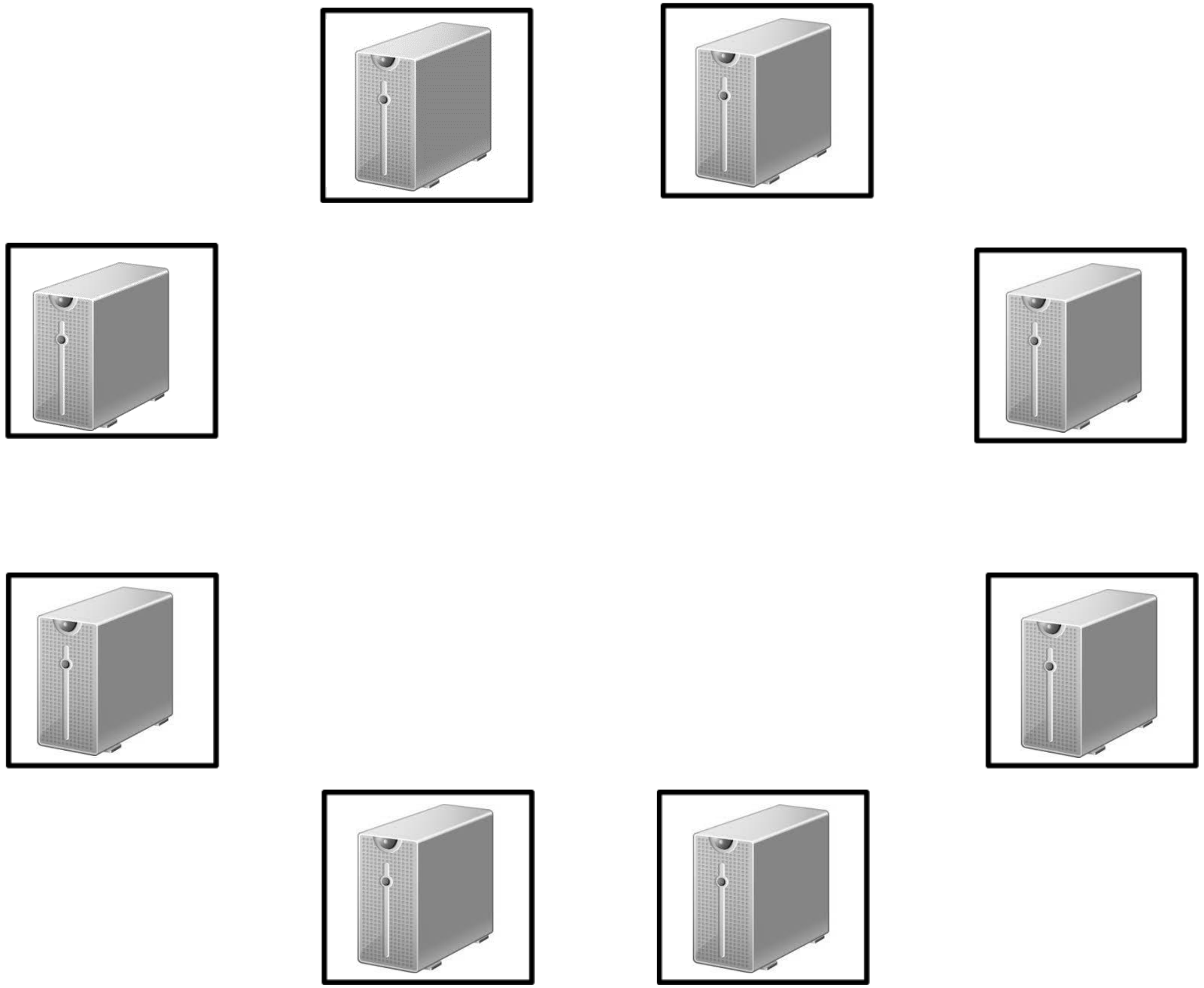
$M$ **machines**

$S$ **memory per machine**

**Synchronous Rounds**

1. **Local Computation**
   at every machine
2. **Global Communication**
   between machines

# Massively Parallel Computing (MPC) Model



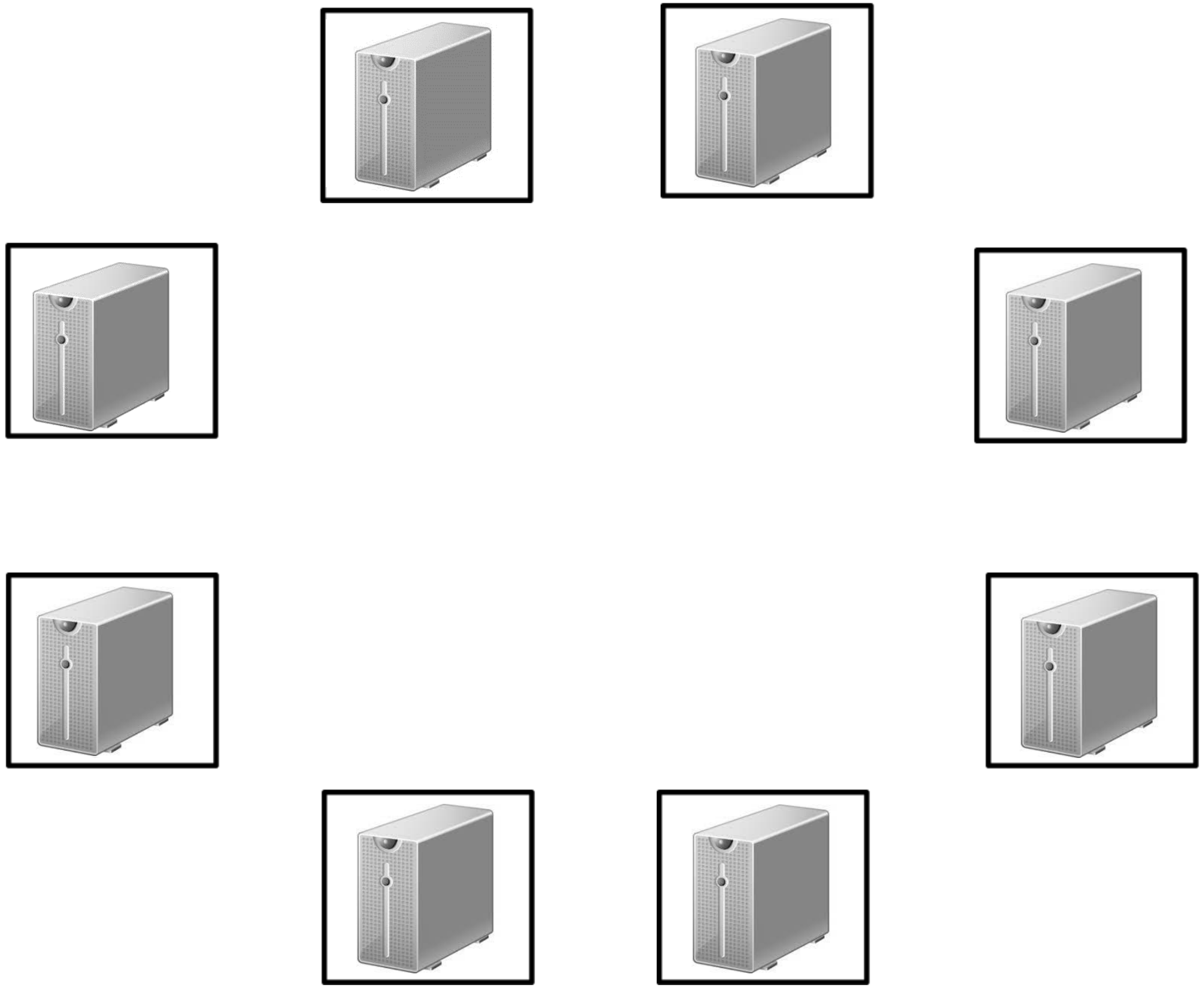
**$M$ machines**

**$S$ memory per machine**

**Synchronous Rounds**

1. **Local Computation**
   at every machine
2. **Global Communication**
   between machines

# Massively Parallel Computing (MPC) Model



$M$ **machines**

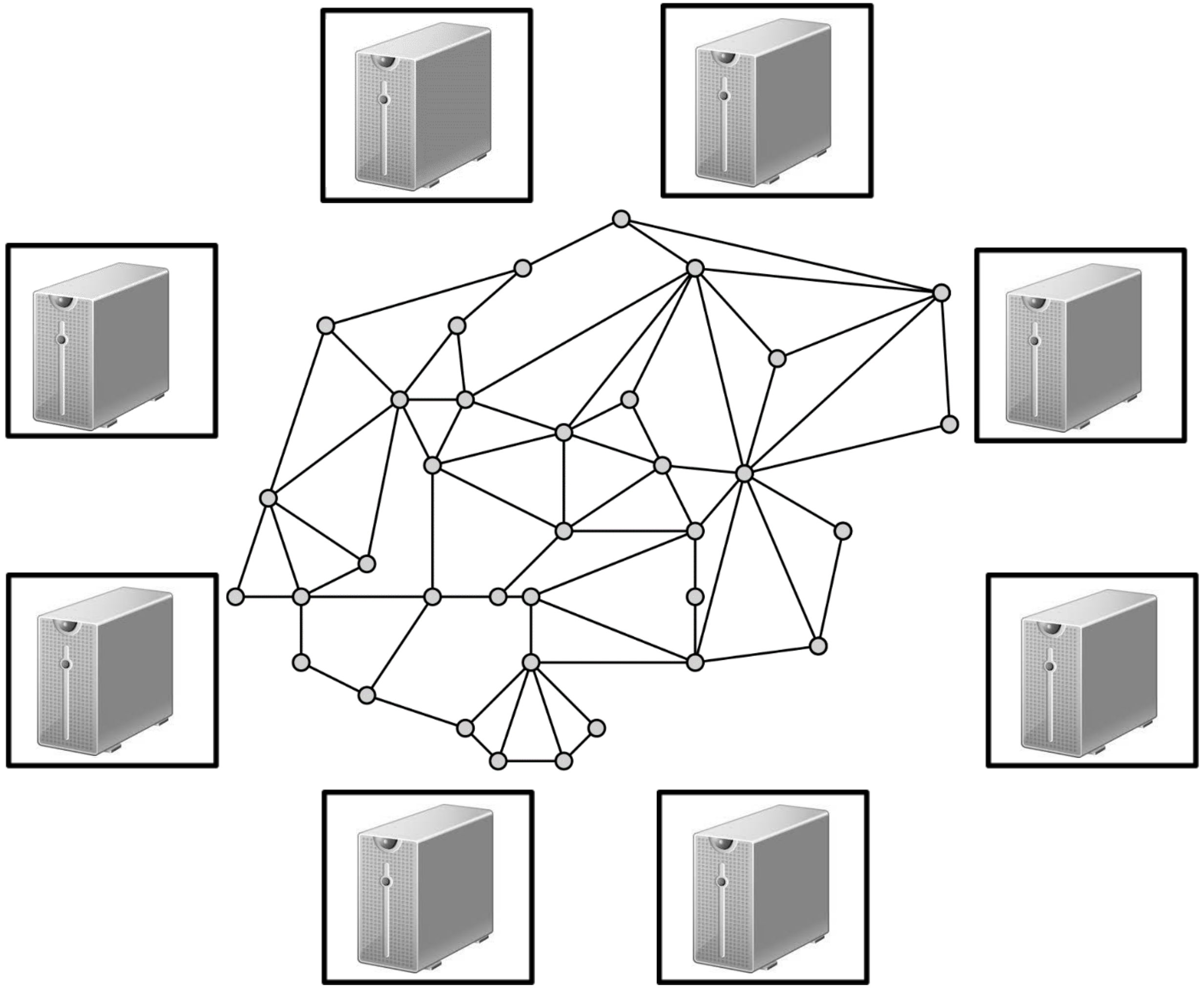
$S$ **memory per machine**

**Synchronous Rounds**

1. **Local Computation**
   at every machine
2. **Global Communication**
   between machines

**Complexity:**
number of rounds

# Massively Parallel Computing (MPC) Model



$M$ **machines**

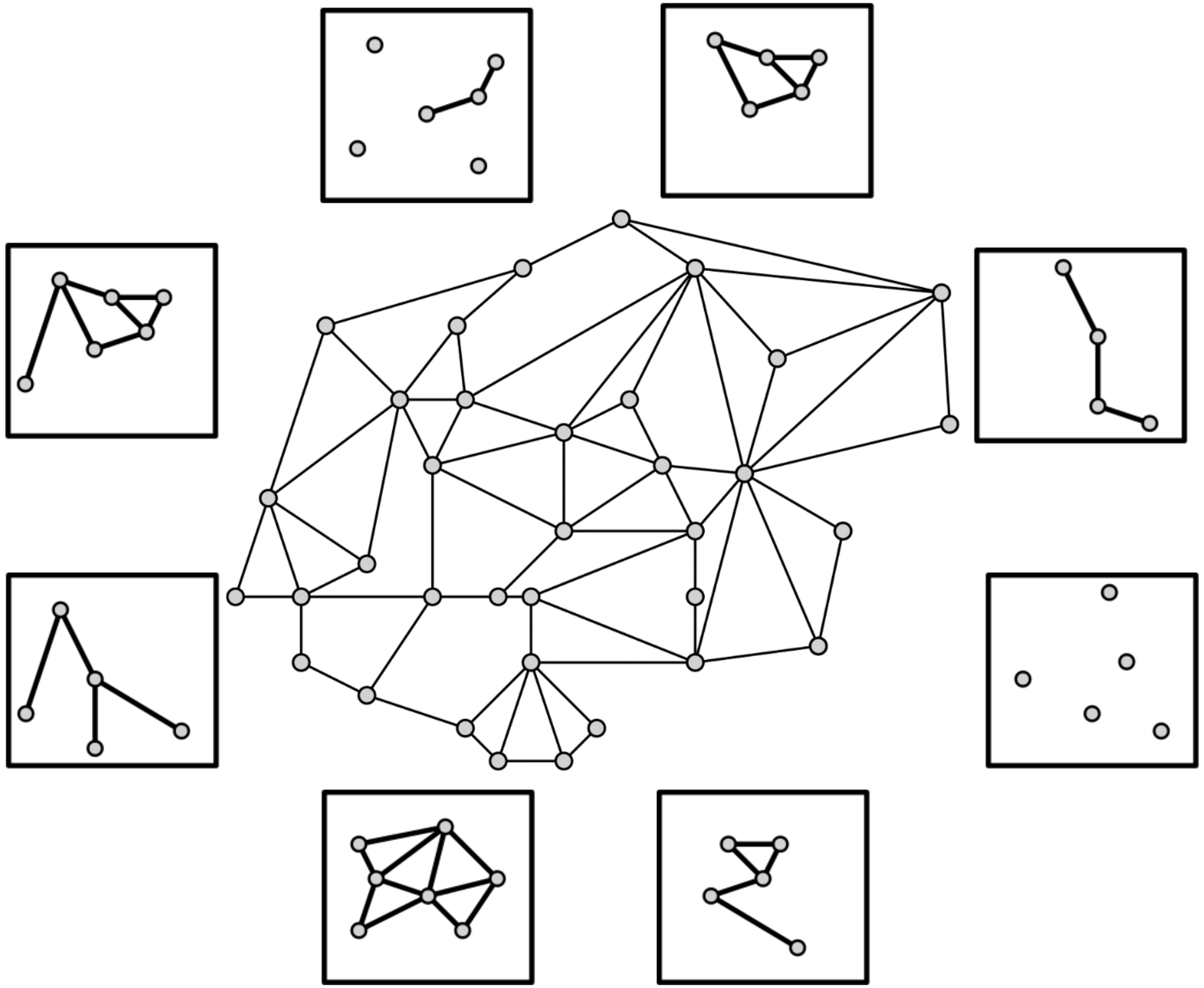
$S$ **memory per machine**

**Synchronous Rounds**

1. **Local Computation**
   at every machine
2. **Global Communication**
   between machines

**Complexity:**
number of rounds

# Massively Parallel Computing (MPC) Model



$M$ **machines**

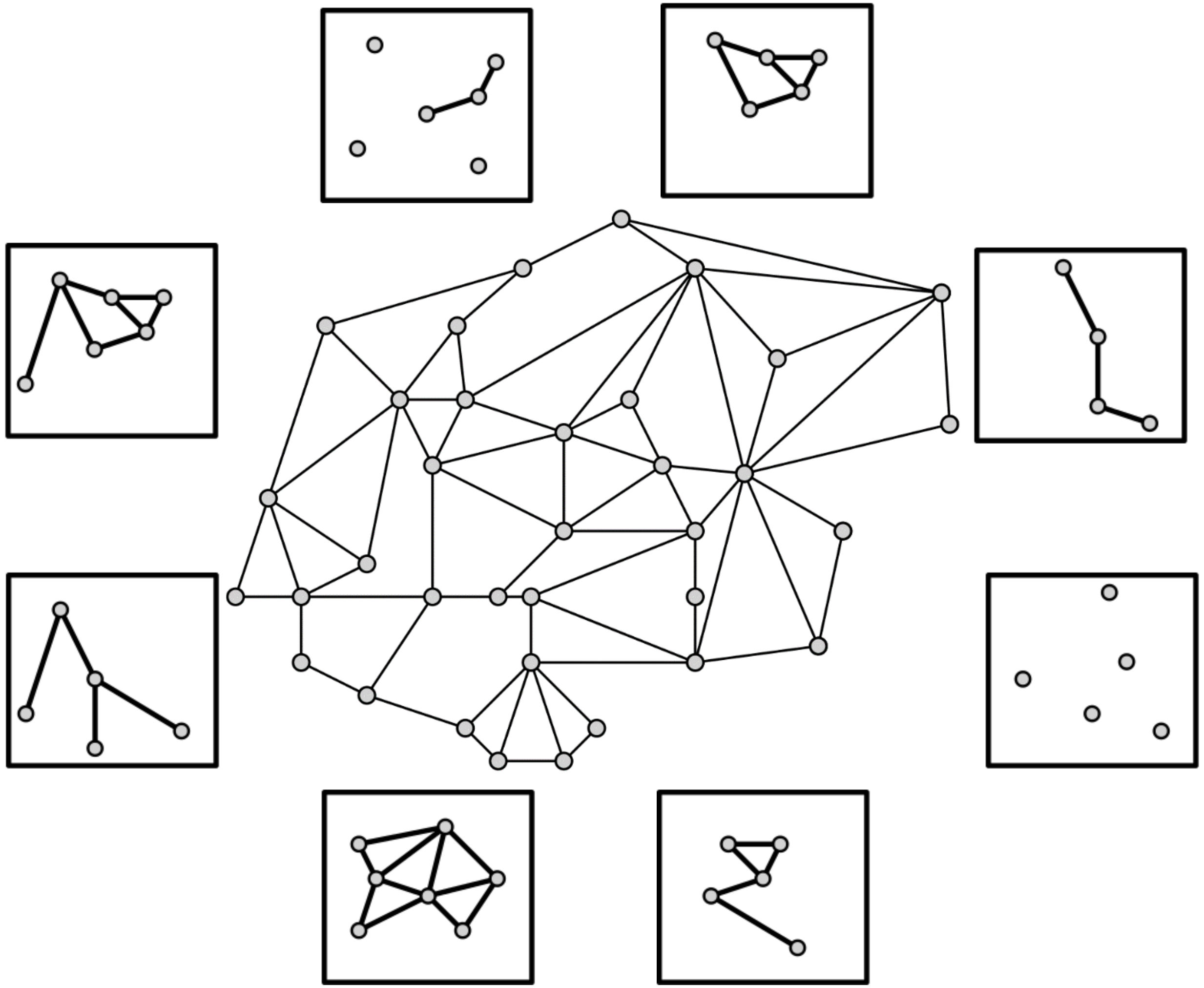
$S$ **memory per machine**

**Synchronous Rounds**

1. **Local Computation**
   at every machine
2. **Global Communication**
   between machines

**Complexity:**
number of rounds

# Massively Parallel Computing (MPC) Model



$M$ **machines**

$S$ **memory per machine**

$M \cdot S = \widetilde{O}\,(m + n)$

**Synchronous Rounds**

1. **Local Computation**
   at every machine
2. **Global Communication**
   between machines

**Complexity:**
number of rounds

# Local Memory in MPC

$M$ machines

$S$ memory per machine

$M \cdot S = \widetilde{O}\,(m + n)$

# Local Memory in MPC

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

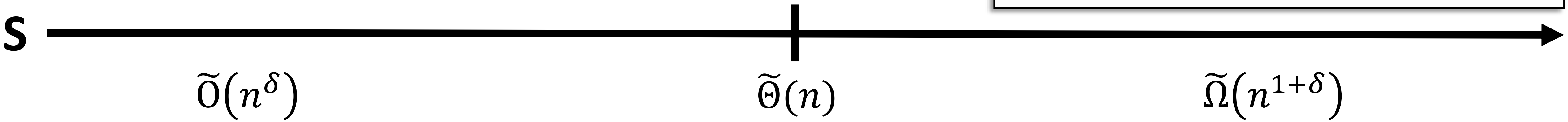
$\boldsymbol{M \cdot S = \widetilde{O}\ (m + n)}$

# Local Memory in MPC

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \widetilde{O}\ (m + n)}$



**S**

$\widetilde{\mathrm{O}}\left(n^{\delta}\right)$

$\widetilde{\Theta}(n)$

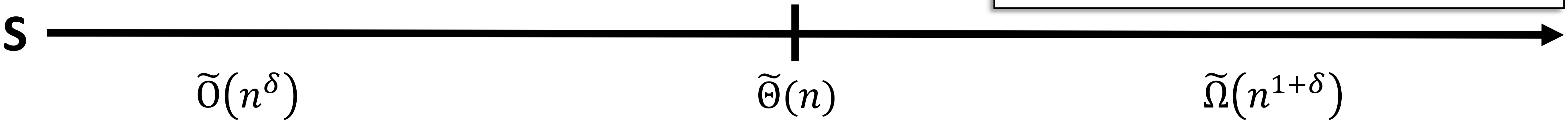
$\widetilde{\Omega}\left(n^{1+\delta}\right)$

# Local Memory in MPC

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$$\boldsymbol{M \cdot S = \tilde{O}\,(m + n)}$$



**Superlinear Memory:**

$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$

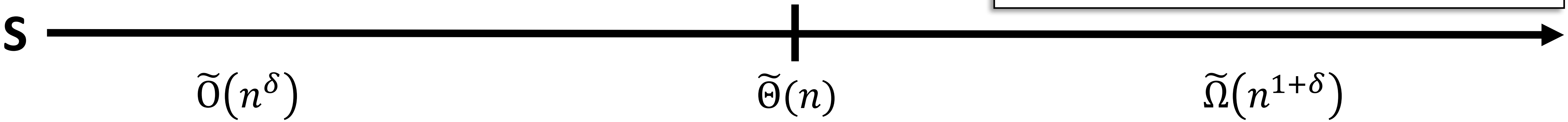
**Machines see all nodes.**

# Local Memory in MPC

**$M$ machines**

**$S$ memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\left(m + n\right)}$



**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

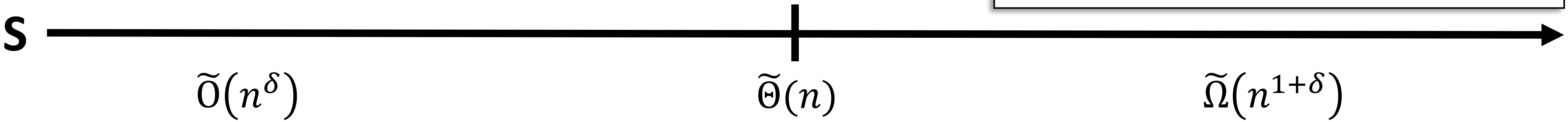
often trivial

# Local Memory in MPC

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\,(m + n)}$



**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

often trivial

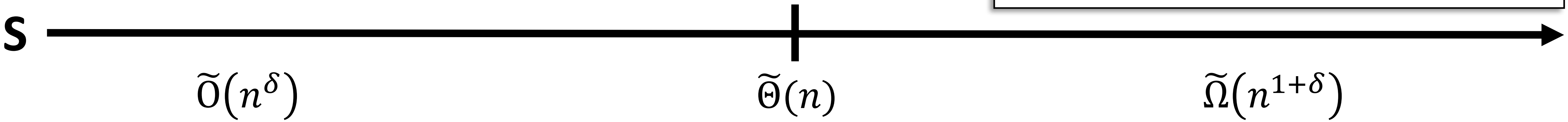
for many problems,
admits O(1)-round algorithms
based on very simple
sampling approach
*Lattanzi et al.* [SPAA'11]

# Local Memory in MPC

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\ (m + n)}$



S

$\tilde{O}(n^{\delta})$

$\tilde{\Theta}(n)$

$\tilde{\Omega}(n^{1+\delta})$

**Linear Memory:**

$S = \tilde{O}(n)$

**Machines see all nodes.**

**Superlinear Memory:**

$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$

**Machines see all nodes.**

often trivial

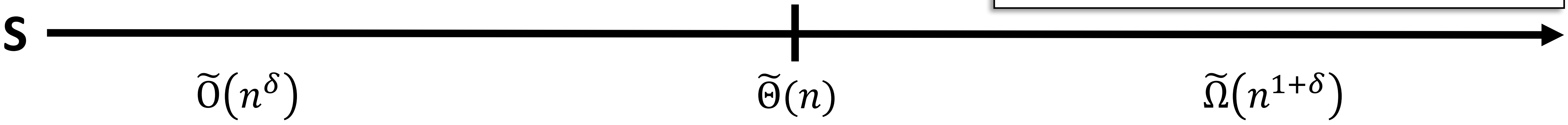
for many problems,
admits O(1)-round algorithms
based on very simple
sampling approach
*Lattanzi et al.* [SPAA’11]

# Local Memory in MPC

**$M$ machines**

**$S$ memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\,(m + n)}$



**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

usual assumption

**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

often trivial

for many problems,
admits O(1)-round algorithms
based on very simple
sampling approach
*Lattanzi et al.* [SPAA'11]

# Local Memory in MPC

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\,(m+n)}$

**S**

$\tilde{\mathrm{O}}(n^{\delta})$

$\tilde{\Theta}(n)$

$\tilde{\Omega}(n^{1+\delta})$

**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

usual assumption

often unrealistic

**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

often trivial

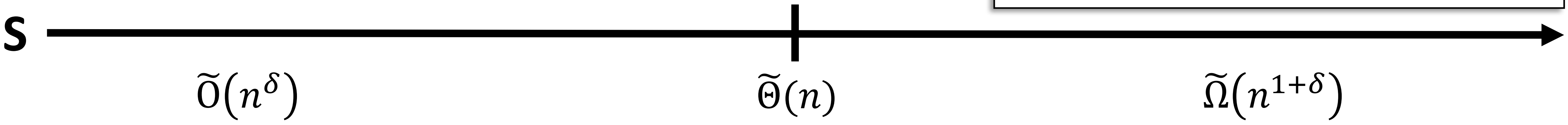
for many problems,
admits O(1)-round algorithms
based on very simple
sampling approach
*Lattanzi et al.* [SPAA’11]

# Local Memory in MPC

**$M$ machines**

**$S$ memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\ (m + n)}$



**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

**Superlinear Memory:**
$S = \tilde{O}\left(n^{1+\delta}\right), 0 < \delta \leq 1$
**Machines see all nodes.**

usual assumption

often unrealistic

- $\widetilde{\mathrm{O}}(n)$ prohibitively large

often trivial

for many problems,
admits O(1)-round algorithms
based on very simple
sampling approach
*Lattanzi et al.* [SPAA'11]

# Local Memory in MPC

**$M$ machines**

**$S$ memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\ (m + n)}$

S

$\widetilde{\mathrm{O}}\left(n^{\delta}\right)$ $\widetilde{\Theta}(n)$ $\widetilde{\Omega}\left(n^{1+\delta}\right)$

**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

usual assumption

often unrealistic
- $\widetilde{\mathrm{O}}(n)$ prohibitively large
- sparse graphs trivial

**Superlinear Memory:**
$S = \tilde{O}\left(n^{1+\delta}\right), 0 < \delta \leq 1$
**Machines see all nodes.**

often trivial

for many problems,
admits O(1)-round algorithms
based on very simple
sampling approach
*Lattanzi et al.* [SPAA’11]

# Local Memory in MPC

**$M$ machines**

**$S$ memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\,(m + n)}$

**S**

$\tilde{\mathrm{O}}(n^{\delta})$

$\tilde{\Theta}(n)$

$\tilde{\Omega}(n^{1+\delta})$

**Strongly Sublinear Memory:**
$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$
**No machine sees all nodes.**

**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

usual assumption

often unrealistic

- $\tilde{\mathrm{O}}(n)$ prohibitively large
- sparse graphs trivial

often trivial

for many problems,
admits O(1)-round algorithms
based on very simple
sampling approach
*Lattanzi et al.* [SPAA'11]

# Local Memory in MPC

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\,(m+n)}$

**S**

$\tilde{\mathrm{O}}(n^{\delta})$

$\tilde{\Theta}(n)$

$\tilde{\Omega}(n^{1+\delta})$

**Strongly Sublinear Memory:**
$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$
**No machine sees all nodes.**

for most problems,
only direct simulation of
LOCAL/PRAM algorithms
known

**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

usual assumption

often unrealistic

- $\tilde{\mathrm{O}}(n)$ prohibitively large
- sparse graphs trivial

**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

often trivial

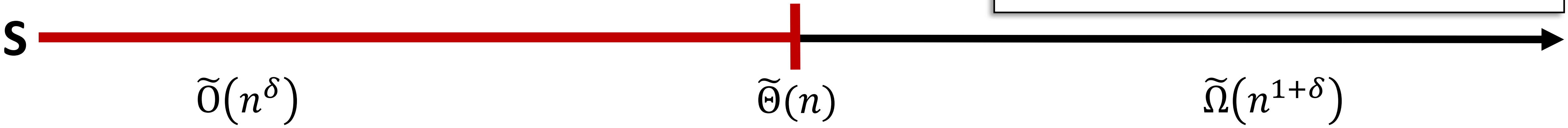
for many problems,
admits O(1)-round algorithms
based on very simple
sampling approach
*Lattanzi et al.* [SPAA’11]

# Local Memory in MPC

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\ (m + n)}$



**Strongly Sublinear Memory:**
$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$
**No machine sees all nodes.**

**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

for most problems, only direct simulation of LOCAL/PRAM algorithms known

usual assumption

often unrealistic

- $\tilde{\mathrm{O}}(n)$ prohibitively large
- sparse graphs trivial

often trivial

for many problems, admits O(1)-round algorithms based on very simple sampling approach
*Lattanzi et al.* [SPAA’11]

# Local Memory in MPC

$M$ **machines**

$S$ **memory per machine**

$M \cdot S = \tilde{O}\ (m + n)$

**S**

$\tilde{O}(n^{\delta})$ $\tilde{\Theta}(n)$ $\tilde{\Omega}(n^{1+\delta})$

**Strongly Sublinear Memory:**
$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$
**No machine sees all nodes.**

**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

for most problems, only direct simulation of LOCAL/PRAM algorithms known

usual assumption

often unrealistic
- $\tilde{O}(n)$ prohibitively large
- sparse graphs trivial

often trivial

for many problems, admits O(1)-round algorithms based on very simple sampling approach
*Lattanzi et al.* [SPAA'11]

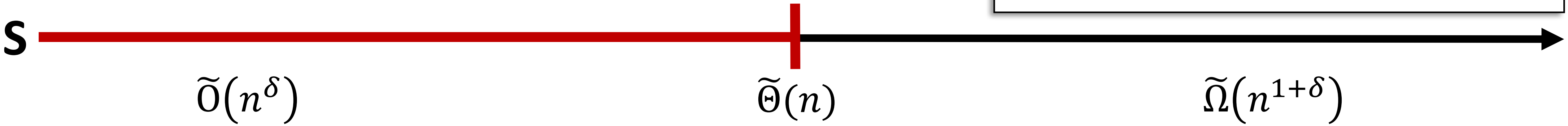
**Algorithms have been stuck at this linear-memory barrier!**

# Local Memory in MPC

**$M$ machines**

**$S$ memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\,(m + n)}$



**Strongly Sublinear Memory:**
$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$
**No machine sees all nodes.**

**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

for most problems,
only direct simulation of
LOCAL/PRAM algorithms
known

usual assumption

often unrealistic
- $\tilde{\mathrm{O}}(n)$ prohibitively large
- sparse graphs trivial

often trivial

for many problems,
admits O(1)-round algorithms
based on very simple
sampling approach
*Lattanzi et al.* [SPAA'11]

**Algorithms have been stuck at this linear-memory barrier!**

**Fundamentally?**

# Breaking the Linear-Memory Barrier:

## Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

## Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S = O(n^\delta)}$ local memory

$\boldsymbol{M = O(m/n^\delta)}$ machines

$\mathbf{poly}\,\mathbf{log}\,\mathbf{log}\,\boldsymbol{n}$ rounds

Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S = O(n^{\delta})}$ local memory

$\boldsymbol{M = O(m/n^{\delta})}$ machines

$\mathbf{poly\,log\,log}\,\boldsymbol{n}$ rounds

*Ghaffari, Kuhn, Uitto* [FOCS'19]

**Conditional Lower Bound**

$\Omega(\log\log n)$ rounds

## Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S = O(n^{\delta})}$ local memory

$\boldsymbol{M = O(m/n^{\delta})}$ machines

$\mathbf{poly\log\log} \boldsymbol{n}$ rounds

## Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S = O(n^{\delta})}$ local memory

$\boldsymbol{M = O(m/n^{\delta})}$ machines

$\mathbf{poly\,log\,log}\,\boldsymbol{n}$ rounds

**Imposed Locality:**
machines see only subset of nodes,
regardless of sparsity of graph

Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

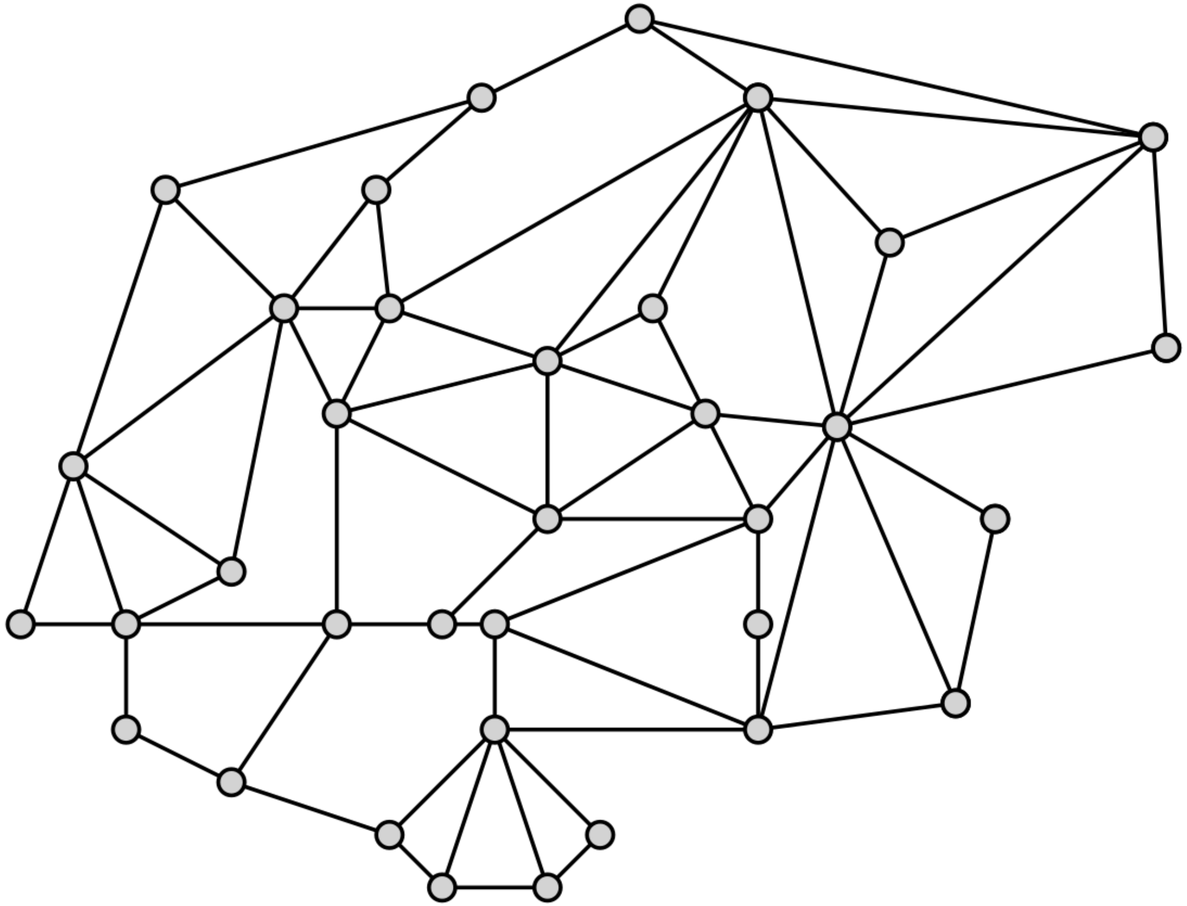
$\boldsymbol{S = O(n^{\delta})}$ local memory

$\boldsymbol{M = O(m/n^{\delta})}$ machines

$\mathbf{poly\,log\,log}\,\boldsymbol{n}$ rounds

**Imposed Locality:**

machines see only subset of nodes, regardless of sparsity of graph



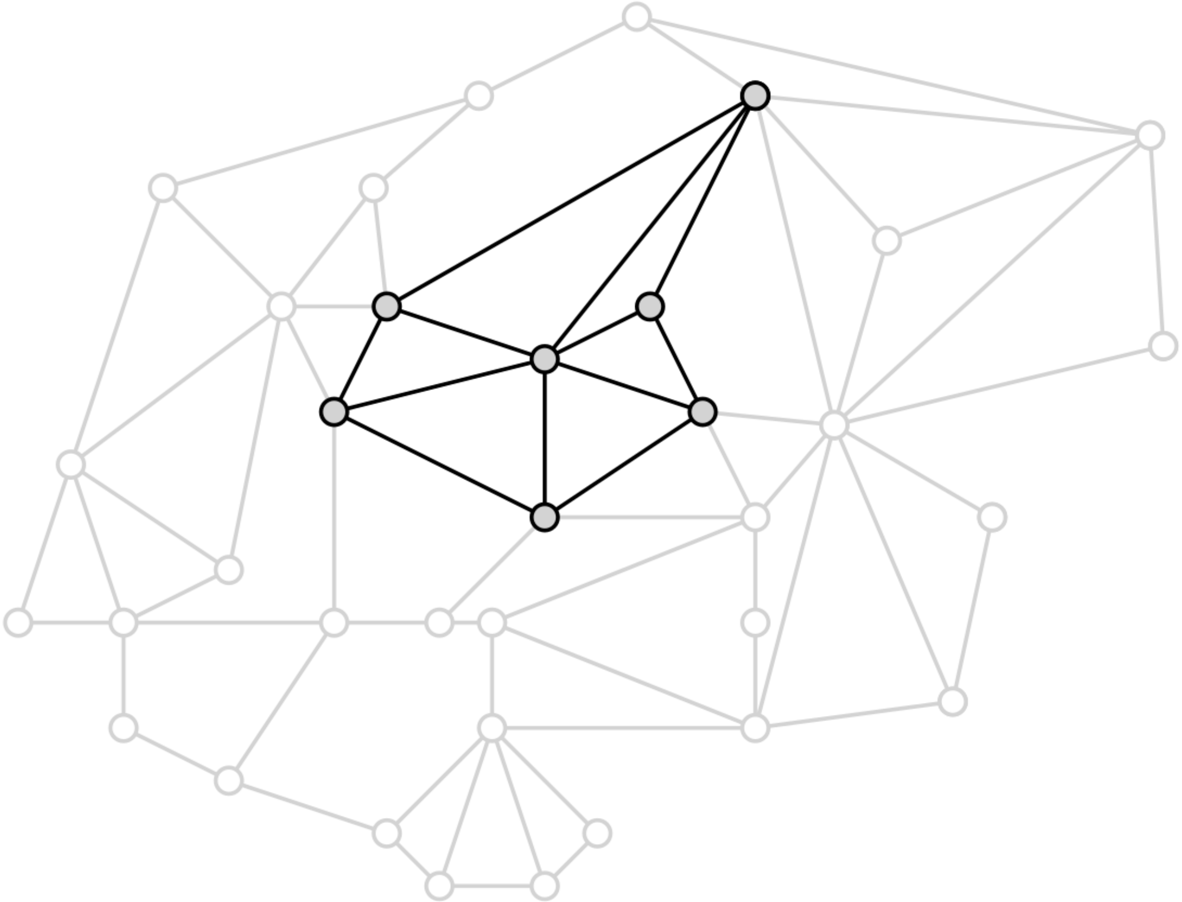
## Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S = O(n^{\delta})}$ local memory

$\boldsymbol{M = O(m/n^{\delta})}$ machines

$\textbf{poly}\log\log \boldsymbol{n}$ rounds



**Imposed Locality:**

machines see only subset of nodes, regardless of sparsity of graph

Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

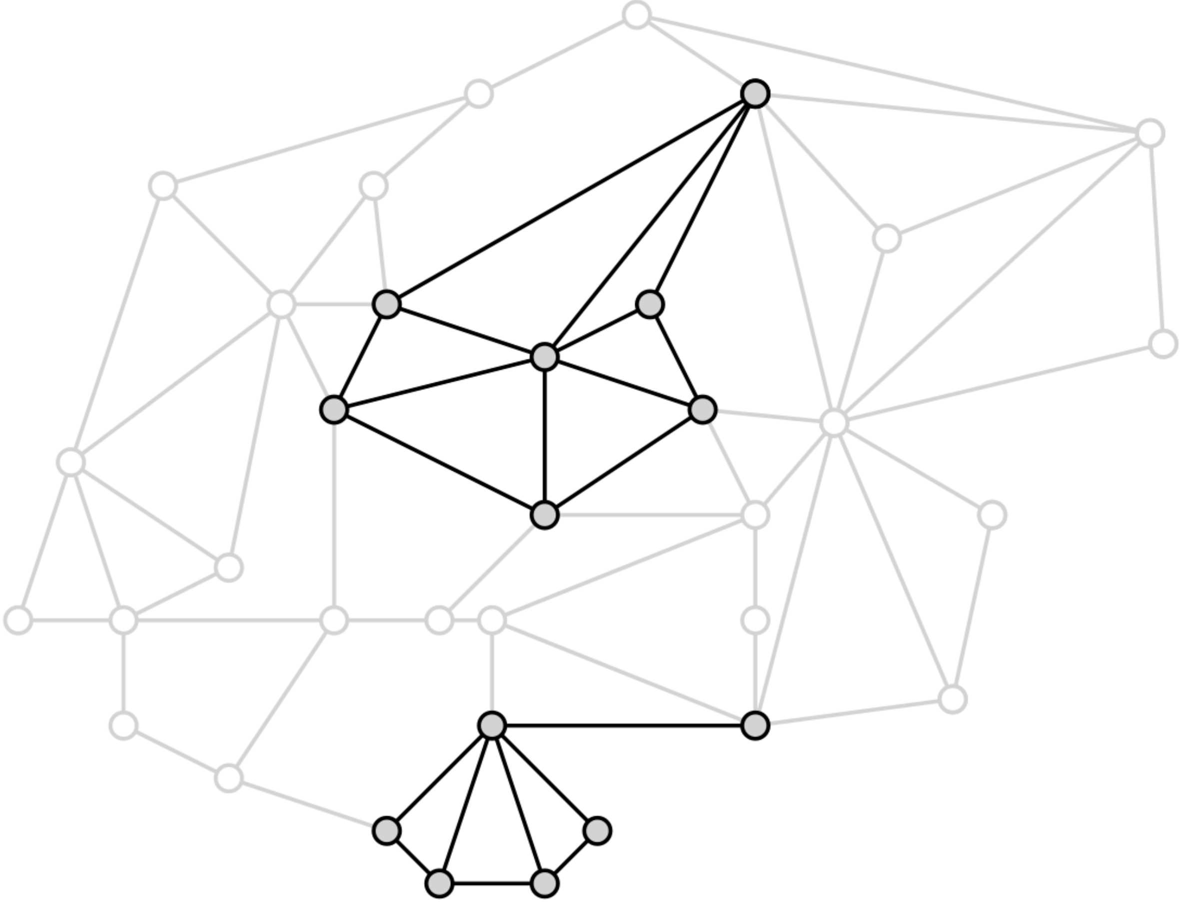
$\boldsymbol{S = O(n^{\delta})}$ local memory

$\boldsymbol{M = O(m/n^{\delta})}$ machines

$\mathbf{poly\,log\,log}\,\boldsymbol{n}$ rounds

**Imposed Locality:**

machines see only subset of nodes, regardless of sparsity of graph



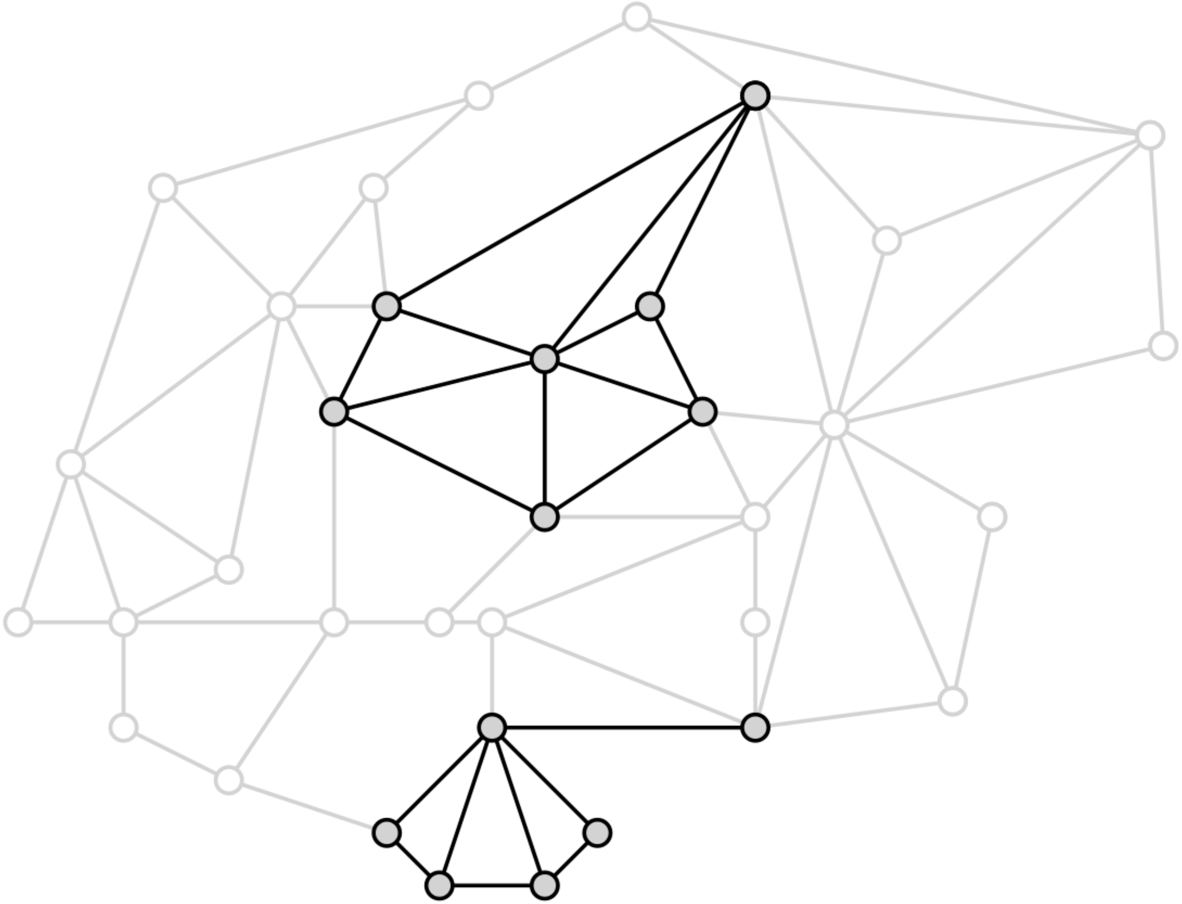
Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S = O(n^{\delta})}$ local memory

$\boldsymbol{M = O(m/n^{\delta})}$ machines

$\mathbf{poly\,log\,log}\,\boldsymbol{n}$ rounds



**Imposed Locality:**

machines see only subset of nodes,
regardless of sparsity of graph

**Our Approach to Cope with Locality:**

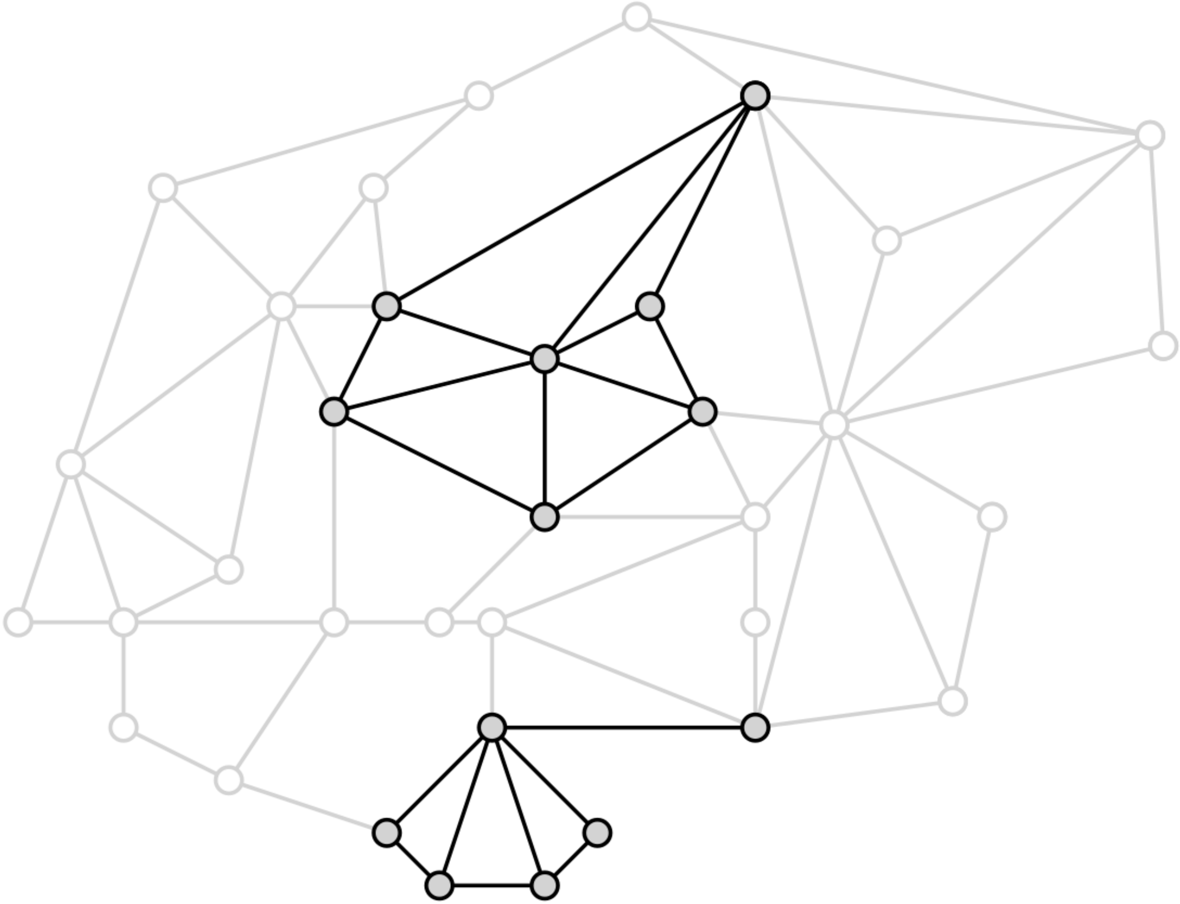
Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S} = \boldsymbol{O}(\boldsymbol{n}^{\boldsymbol{\delta}})$ local memory

$\boldsymbol{M} = \boldsymbol{O}(\boldsymbol{m}/\boldsymbol{n}^{\boldsymbol{\delta}})$ machines

$\mathbf{poly}\log\log \boldsymbol{n}$ rounds



**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph

**Our Approach to Cope with Locality:**
enhance **LOCAL algorithms** with **global communication**

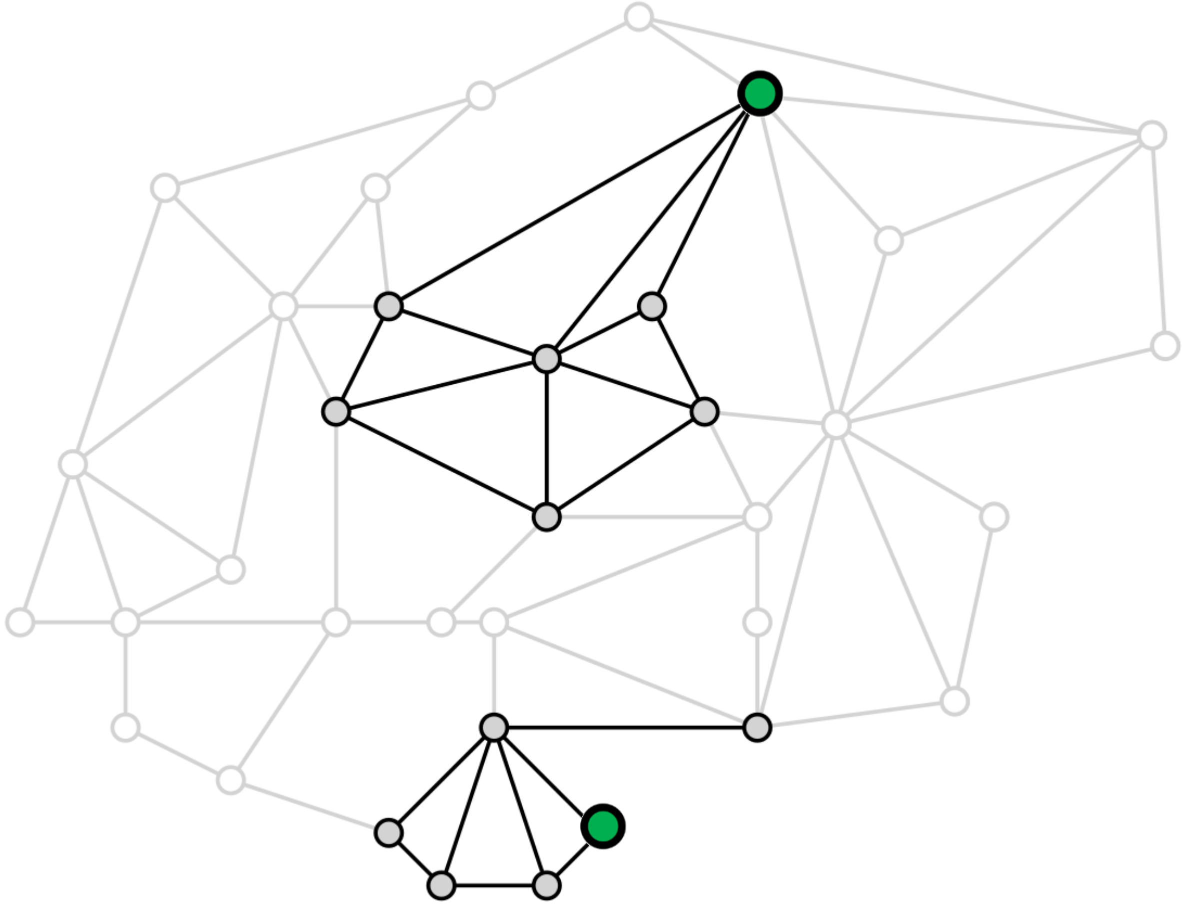
## Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S = O(n^{\delta})}$ local memory

$\boldsymbol{M = O(m/n^{\delta})}$ machines

$\mathbf{poly}\log\log \boldsymbol{n}$ rounds



**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph

**Our Approach to Cope with Locality:**
enhance **LOCAL algorithms** with **global communication**

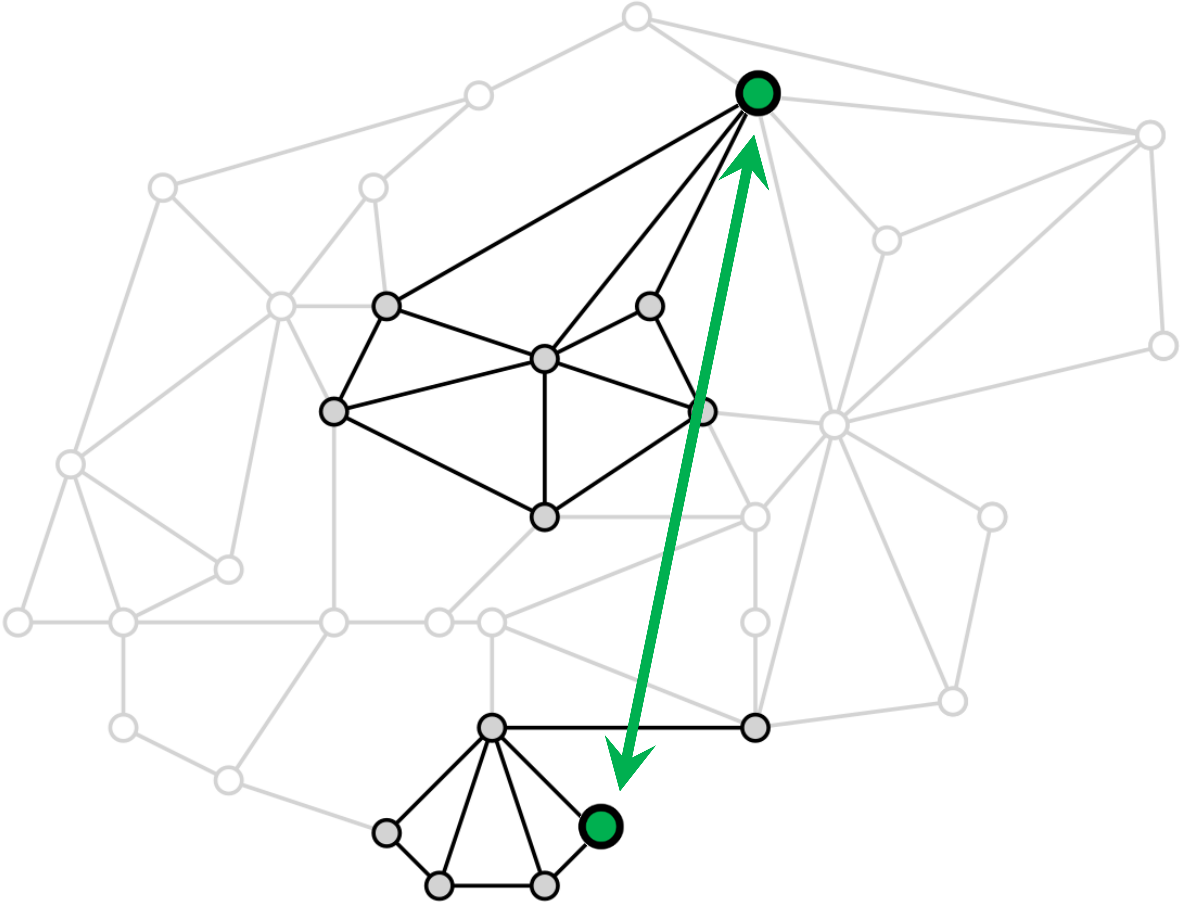
Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$S = O(n^{\delta})$ local memory

$M = O(m/n^{\delta})$ machines

$\text{poly}\log\log n$ rounds



**Imposed Locality:**

machines see only subset of nodes, regardless of sparsity of graph

**Our Approach to Cope with Locality:**

enhance **LOCAL algorithms** with **global communication**

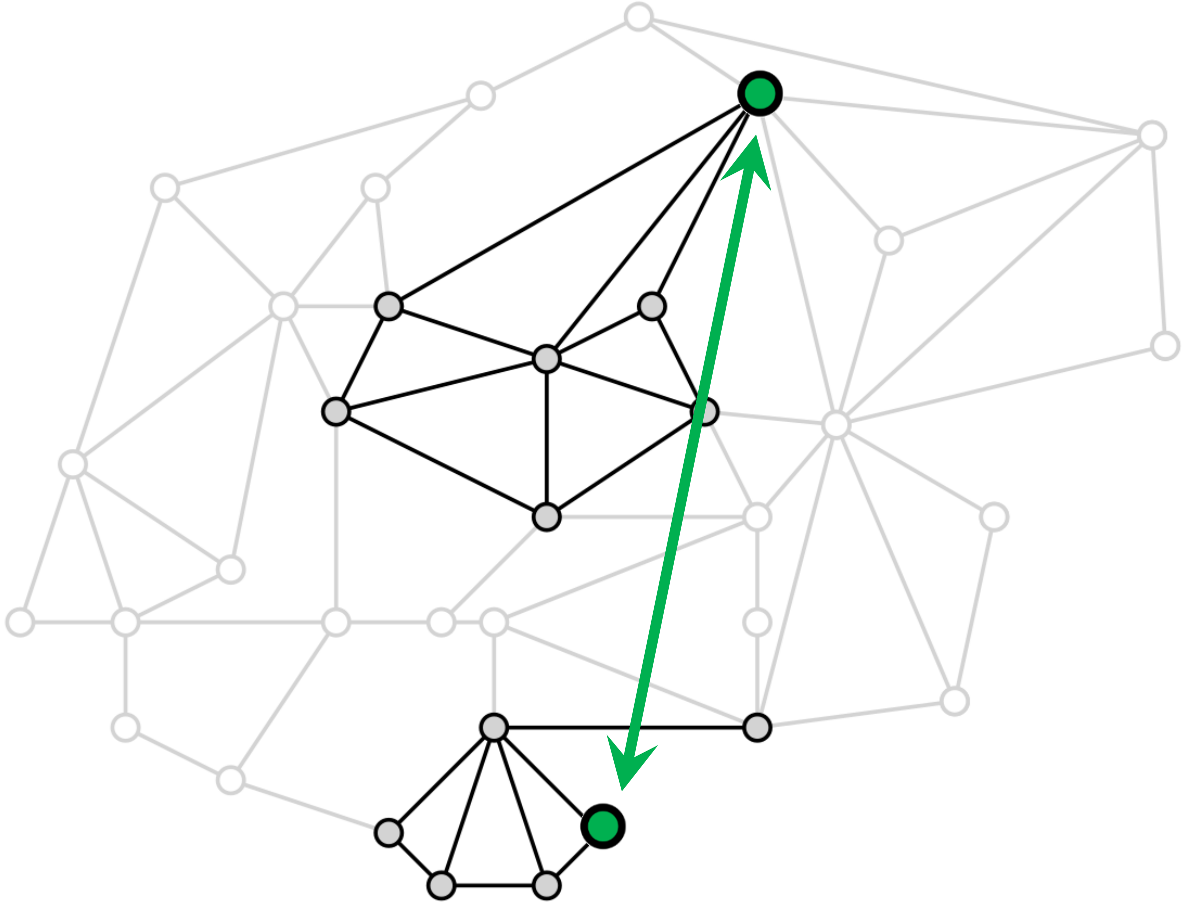
Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S = O(n^\delta)}$ local memory

$\boldsymbol{M = O(m/n^\delta)}$ machines

$\mathbf{poly\,log\,log}\,\boldsymbol{n}$ rounds



**Imposed Locality:**

machines see only subset of nodes, regardless of sparsity of graph

**Our Approach to Cope with Locality:**

enhance **LOCAL algorithms** with **global communication**

- exponentially faster than LOCAL algorithms due to shortcuts

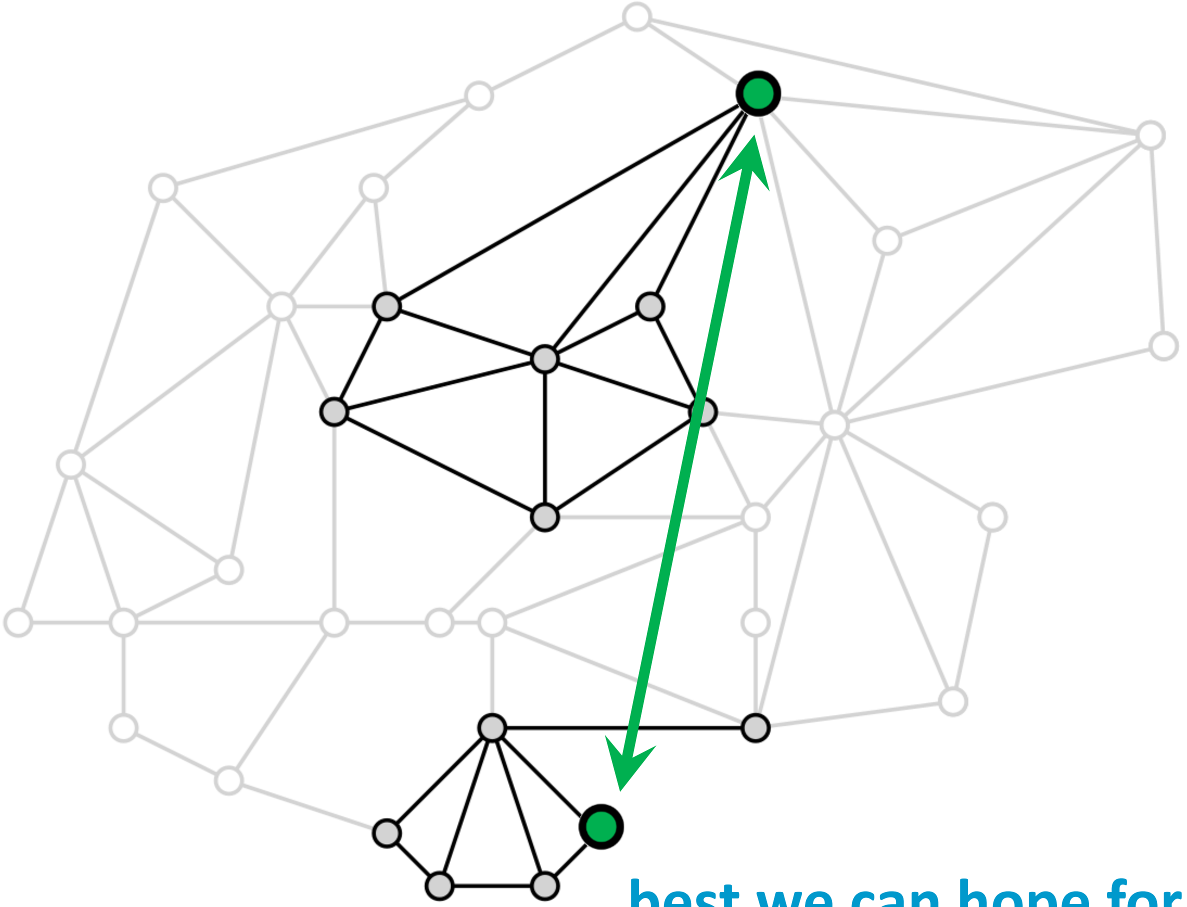
## Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S = O(n^{\delta})}$ local memory

$\boldsymbol{M = O(m/n^{\delta})}$ machines

$\mathbf{poly\,log\,log}\,\boldsymbol{n}$ rounds



**best we can hope for**

*GKU* [FOCS'19]

**Imposed Locality:**

machines see only subset of nodes, regardless of sparsity of graph

**Our Approach to Cope with Locality:**

enhance **LOCAL algorithms** with **global communication**

- exponentially faster than LOCAL algorithms due to shortcuts

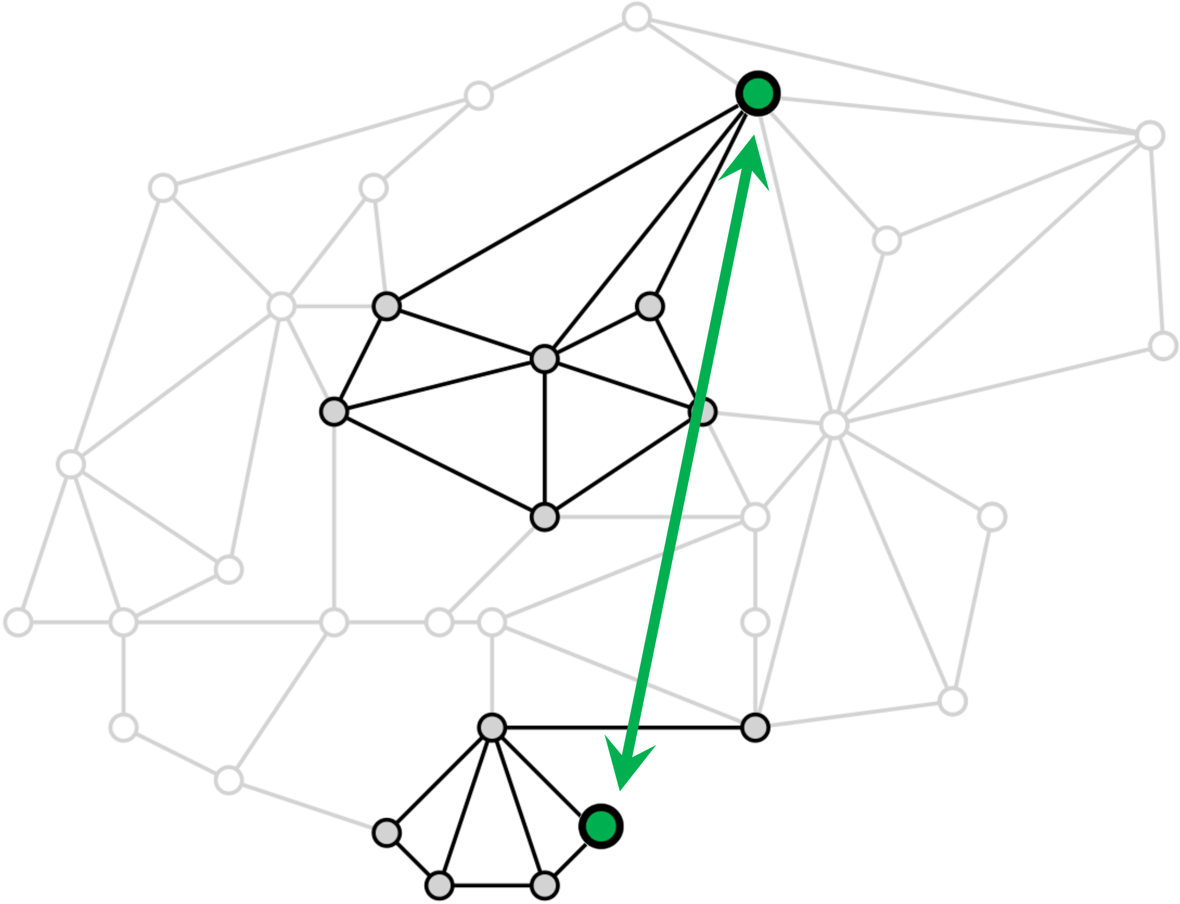
Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S = O(n^{\delta})}$ local memory

$\boldsymbol{M = O(m/n^{\delta})}$ machines

$\mathbf{poly\,log\,log}\,\boldsymbol{n}$ rounds



**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph

**Our Approach to Cope with Locality:**
enhance **LOCAL algorithms** with **global communication**

- exponentially faster than LOCAL algorithms due to shortcuts

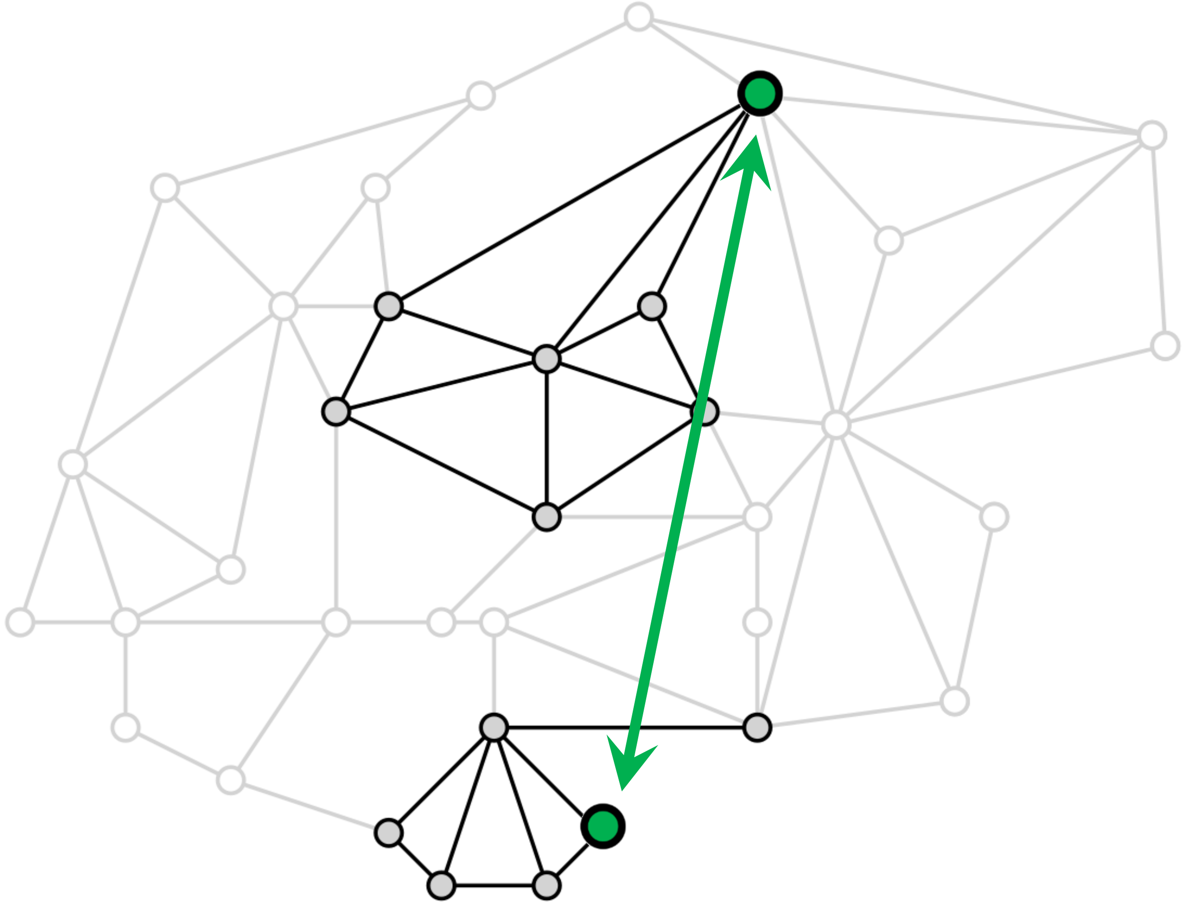
Breaking the Linear-Memory Barrier:

# Efficient MPC Graph Algorithms with Strongly Sublinear Memory

$\boldsymbol{S = O(n^{\delta})}$ local memory

$\boldsymbol{M = O(m/n^{\delta})}$ machines

$\mathbf{poly\,log\,log}\,\boldsymbol{n}$ rounds



**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph

**Our Approach to Cope with Locality:**
enhance **LOCAL algorithms** with **global communication**

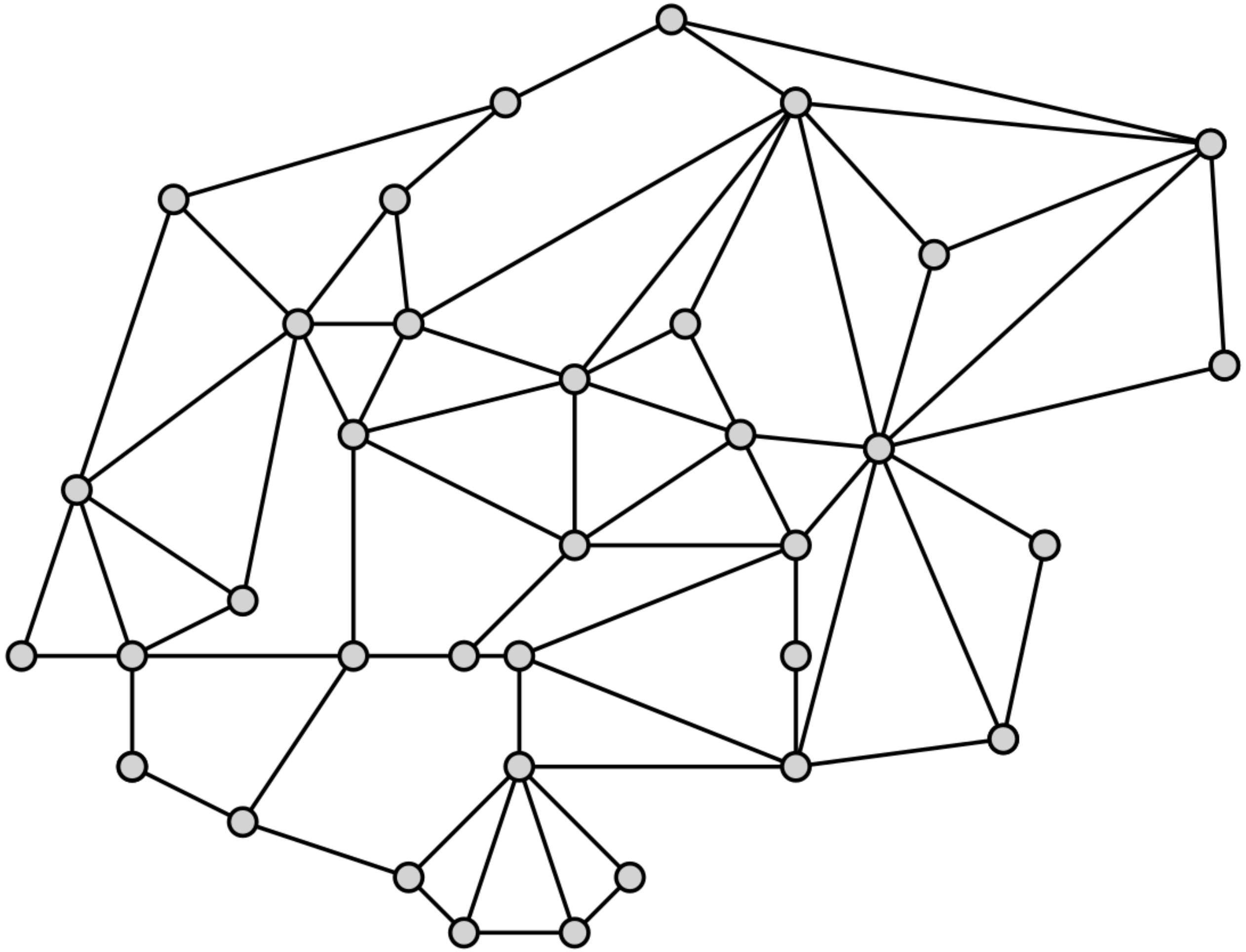
- exponentially faster than LOCAL algorithms due to shortcuts
- polynomially less memory than most MPC algorithms

Problem:

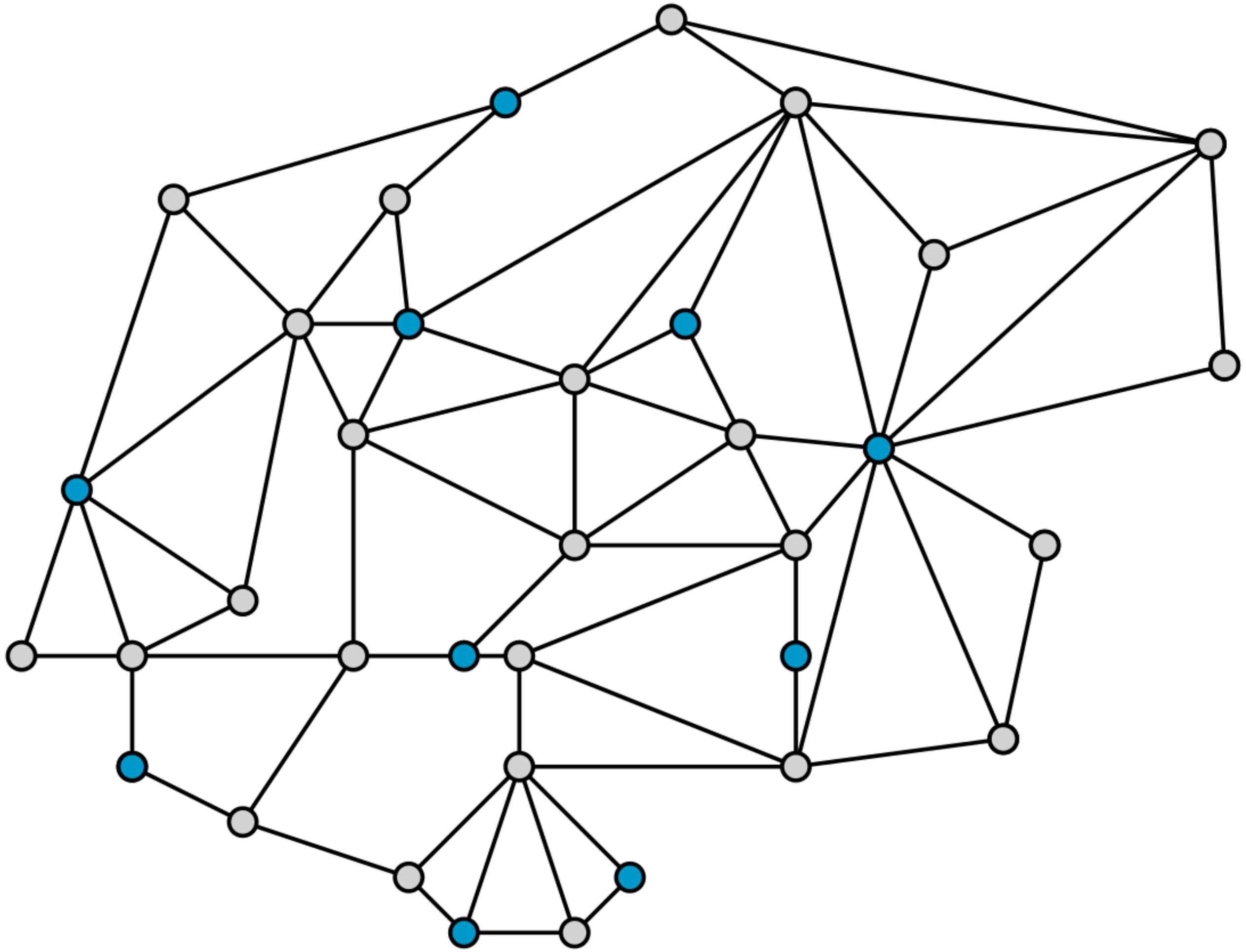
# Maximal Independent Set (MIS)

# Maximal Independent Set (MIS)

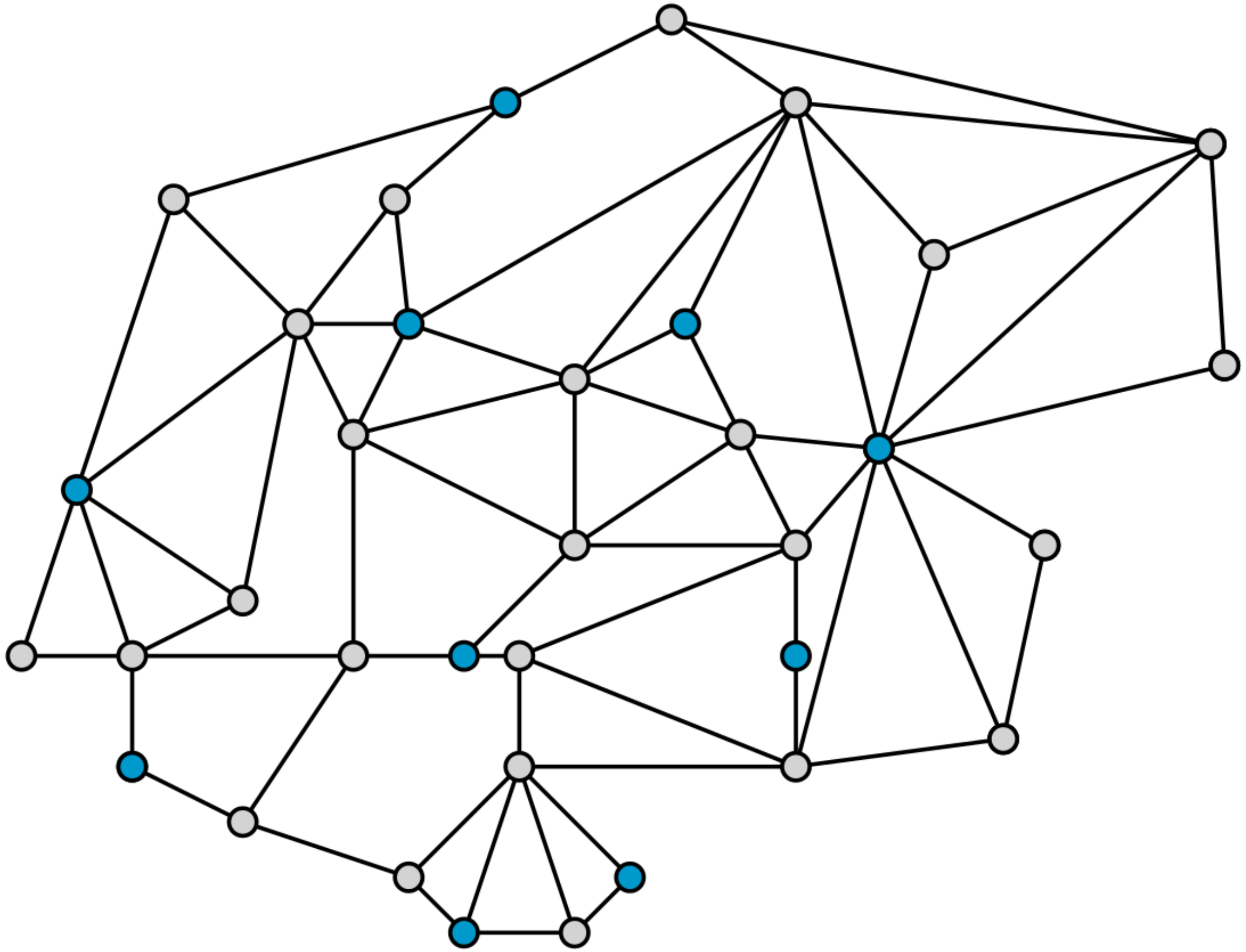
# Maximal Independent Set (MIS)



# Maximal Independent Set (MIS)

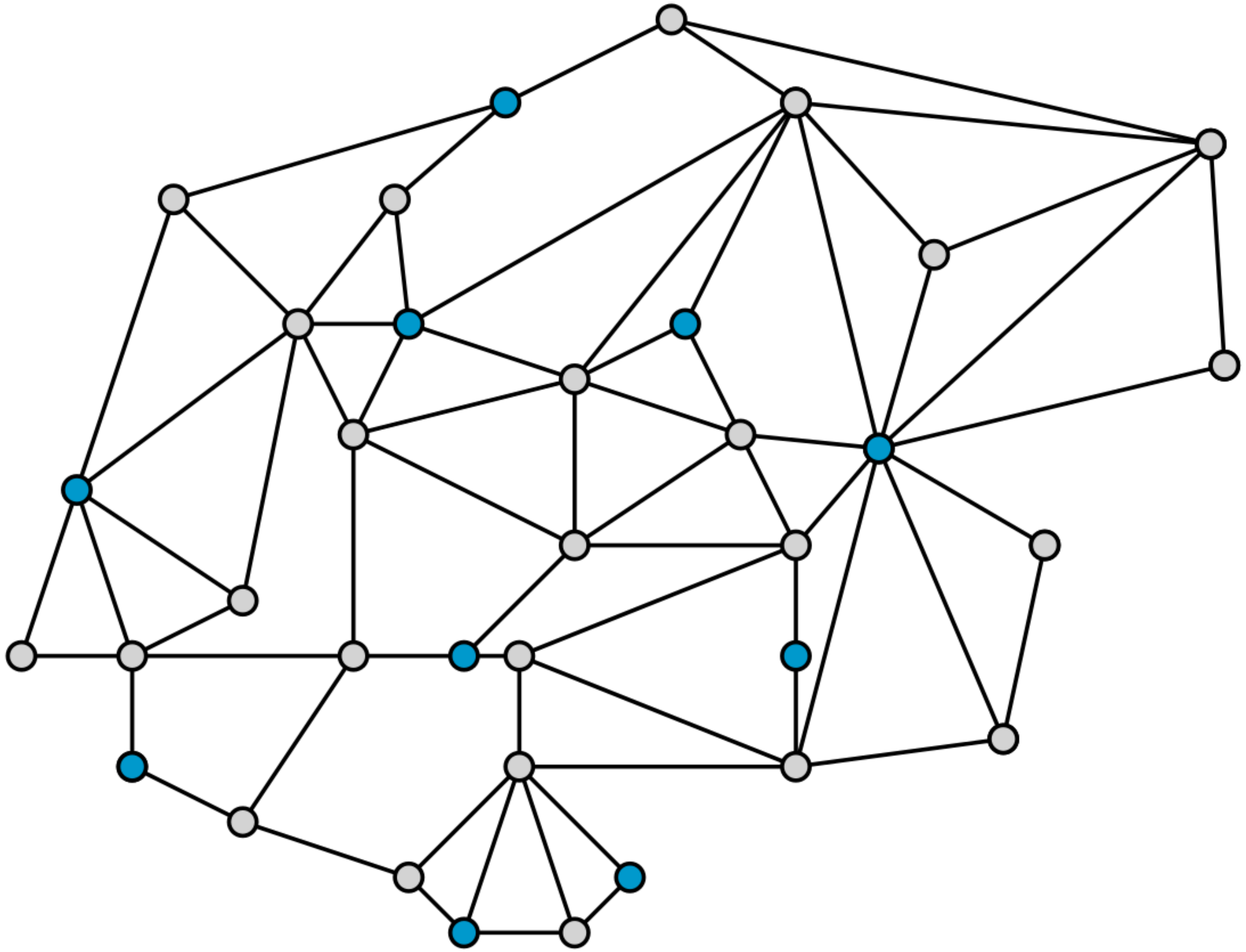


# Maximal Independent Set (MIS)



**Independent Set:**
set of non-adjacent nodes

# Maximal Independent Set (MIS)



**Independent Set:**
set of non-adjacent nodes

**Maximal:**
no node can be added
without violating independence

# MIS: State of the Art

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\,(m+n)}$

**S**

$\widetilde{\mathrm{O}}(n^{\delta})$

$\widetilde{\Theta}(n)$

$\widetilde{\Omega}(n^{1+\delta})$

**Strongly Sublinear Memory:**
$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$
No machine sees all nodes.

**Linear Memory:**
$S = \tilde{O}(n)$
Machines see all nodes.

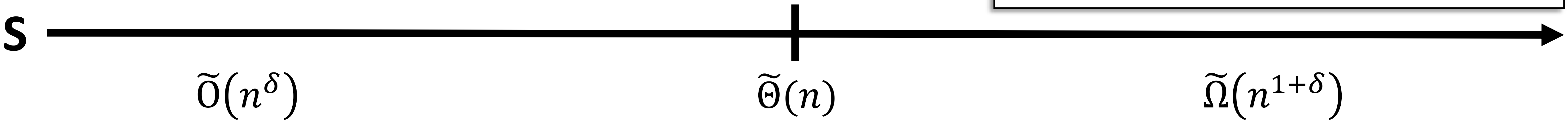
**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
Machines see all nodes.

# MIS: State of the Art

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\,(m + n)}$



**S**

$\tilde{\mathrm{O}}(n^{\delta})$

$\tilde{\Theta}(n)$

$\tilde{\Omega}(n^{1+\delta})$

**Strongly Sublinear Memory:**
$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$
**No machine sees all nodes.**

**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

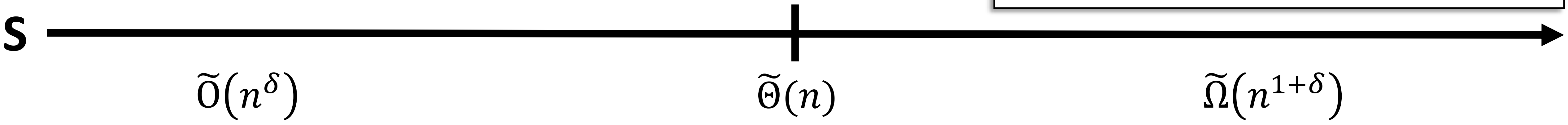
*Lattanzi et al.* [SPAA'11]
$O(1)$

# MIS: State of the Art

$M$ **machines**

$S$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\,(m + n)}$



| | | |
|---|---|---|
| **Strongly Sublinear Memory:**<br>$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$<br>No machine sees all nodes. | **Linear Memory:**<br>$S = \tilde{O}(n)$<br>Machines see all nodes. | **Superlinear Memory:**<br>$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$<br>Machines see all nodes. |
| | *Ghaffari et al.* [PODC'18]<br>$O(\log \log n)$ | *Lattanzi et al.* [SPAA'11]<br>$O(1)$ |

# MIS: State of the Art

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\ (m + n)}$

**S**

$\tilde{\mathrm{O}}(n^{\delta})$ $\tilde{\Theta}(n)$ $\tilde{\Omega}(n^{1+\delta})$

**Strongly Sublinear Memory:**
$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$
**No machine sees all nodes.**

*Luby's Algorithm*
$O(\log n)$

**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

*Ghaffari et al.* [PODC'18]
$O(\log \log n)$

**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

*Lattanzi et al.* [SPAA'11]
$O(1)$

# MIS: State of the Art

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\ (m + n)}$

**S**

$\widetilde{\mathrm{O}}(n^{\delta})$ $\widetilde{\Theta}(n)$ $\widetilde{\Omega}(n^{1+\delta})$

**Strongly Sublinear Memory:**
$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$
**No machine sees all nodes.**

**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

*Luby's Algorithm*
$O(\log n)$

*Ghaffari et al.* [PODC'18]
$O(\log \log n)$

*Lattanzi et al.* [SPAA'11]
$O(1)$

*Ghaffari and Uitto* [SODA'19]
$\tilde{O}(\sqrt{\log n})$

# MIS: State of the Art on Trees

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\,(m + n)}$

**S**

$\widetilde{\mathrm{O}}(n^{\delta})$ $\widetilde{\Theta}(n)$ $\widetilde{\Omega}(n^{1+\delta})$

| | | |
|---|---|---|
| **Strongly Sublinear Memory:**<br>$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$<br>**No machine sees all nodes.** | **Linear Memory:**<br>$S = \tilde{O}(n)$<br>**Machines see all nodes.** | **Superlinear Memory:**<br>$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$<br>**Machines see all nodes.** |
| *Luby's Algorithm*<br>$O(\log n)$ | *Ghaffari et al.* [PODC'18]<br>$O(\log\log n)$ | *Lattanzi et al.* [SPAA'11]<br>$O(1)$ |
| *Ghaffari and Uitto* [SODA'19]<br>$\tilde{O}(\sqrt{\log n})$ | | |

# MIS: State of the Art on Trees

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$\boldsymbol{M \cdot S = \tilde{O}\ (m + n)}$

**S**

$\tilde{O}(n^{\delta})$

$\tilde{\Theta}(n)$

$\tilde{\Omega}(n^{1+\delta})$

**Strongly Sublinear Memory:**
$S = \tilde{O}(n^{\delta}), 0 \leq \delta < 1$
**No machine sees all nodes.**

**Linear Memory:**
$S = \tilde{O}(n)$
**Machines see all nodes.**

**Superlinear Memory:**
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$
**Machines see all nodes.**

*Luby's Algorithm*
$O(\log n)$

*Trivial solution*
$O(1)$

*Trivial solution*
$O(1)$

*Ghaffari and Uitto* [SODA'19]
$\tilde{O}(\sqrt{\log n})$

# MIS: State of the Art on Trees

$\boldsymbol{M}$ **machines**

$\boldsymbol{S}$ **memory per machine**

$$\boldsymbol{M \cdot S = \tilde{O}\,(m+n)}$$

**S**

$\tilde{\mathrm{O}}\left(n^{\delta}\right)$ $\tilde{\Theta}(n)$ $\tilde{\Omega}\left(n^{1+\delta}\right)$

| | | |
|---|---|---|
| **Strongly Sublinear Memory:**<br>$S=\tilde{O}\left(n^{\delta}\right), 0 \leq \delta < 1$<br>**No machine sees all nodes.** | **Linear Memory:**<br>$S=\tilde{O}(n)$<br>**Machines see all nodes.** | **Superlinear Memory:**<br>$S=\tilde{O}\left(n^{1+\delta}\right), 0 < \delta \leq 1$<br>**Machines see all nodes.** |
| *Our Result*<br>$O(\log^3 \log n)$ | *Trivial solution*<br>$O(1)$ | *Trivial solution*<br>$O(1)$ |
| *Ghaffari and Uitto* [SODA'19]<br>$\tilde{O}\left(\sqrt{\log n}\right)$ | | |

# Our Result

$\boldsymbol{O(\log^3 \log n)}$-round MPC algorithm

with $\boldsymbol{S = \widetilde{O}(n^{\delta})}$ memory that w.h.p. computes MIS on trees.

# Our Result

$\widetilde{\boldsymbol{O}}(\sqrt{\log \boldsymbol{n}})$ rounds

$\boldsymbol{S} = \widetilde{\boldsymbol{O}}(\boldsymbol{n}^{\delta})$ memory

*Ghaffari and Uitto* [SODA’19]

$\boldsymbol{O}(\log^3 \log \boldsymbol{n})$-round MPC algorithm

with $\boldsymbol{S} = \widetilde{\boldsymbol{O}}(\boldsymbol{n}^{\delta})$ memory that w.h.p. computes MIS on trees.

# Our Result

$\widetilde{O}(\sqrt{\log n})$ rounds

$S = \widetilde{O}(n^{\delta})$ memory

*Ghaffari and Uitto* [SODA'19]

$O(\log \log n)$ rounds

$S = \widetilde{O}(n)$ memory

*Ghaffari et al.* [PODC'18]

$O(\log^3 \log n)$-round MPC algorithm

with $S = \widetilde{O}(n^{\delta})$ memory that w.h.p. computes MIS on trees.

# Our Result

$\widetilde{\boldsymbol{O}}\left(\sqrt{\log \boldsymbol{n}}\right)$ rounds

$\boldsymbol{S} = \widetilde{\boldsymbol{O}}\left(\boldsymbol{n}^{\delta}\right)$ memory

*Ghaffari and Uitto* [SODA’19]

$\boldsymbol{O}(\log \log \boldsymbol{n})$ rounds

$\boldsymbol{S} = \widetilde{\boldsymbol{O}}(\boldsymbol{n})$ memory

*Ghaffari et al.* [PODC’18]

$\boldsymbol{O}(\log^3 \log \boldsymbol{n})$-round MPC algorithm

with $\boldsymbol{S} = \widetilde{\boldsymbol{O}}\left(\boldsymbol{n}^{\delta}\right)$ memory that w.h.p. computes MIS on trees.

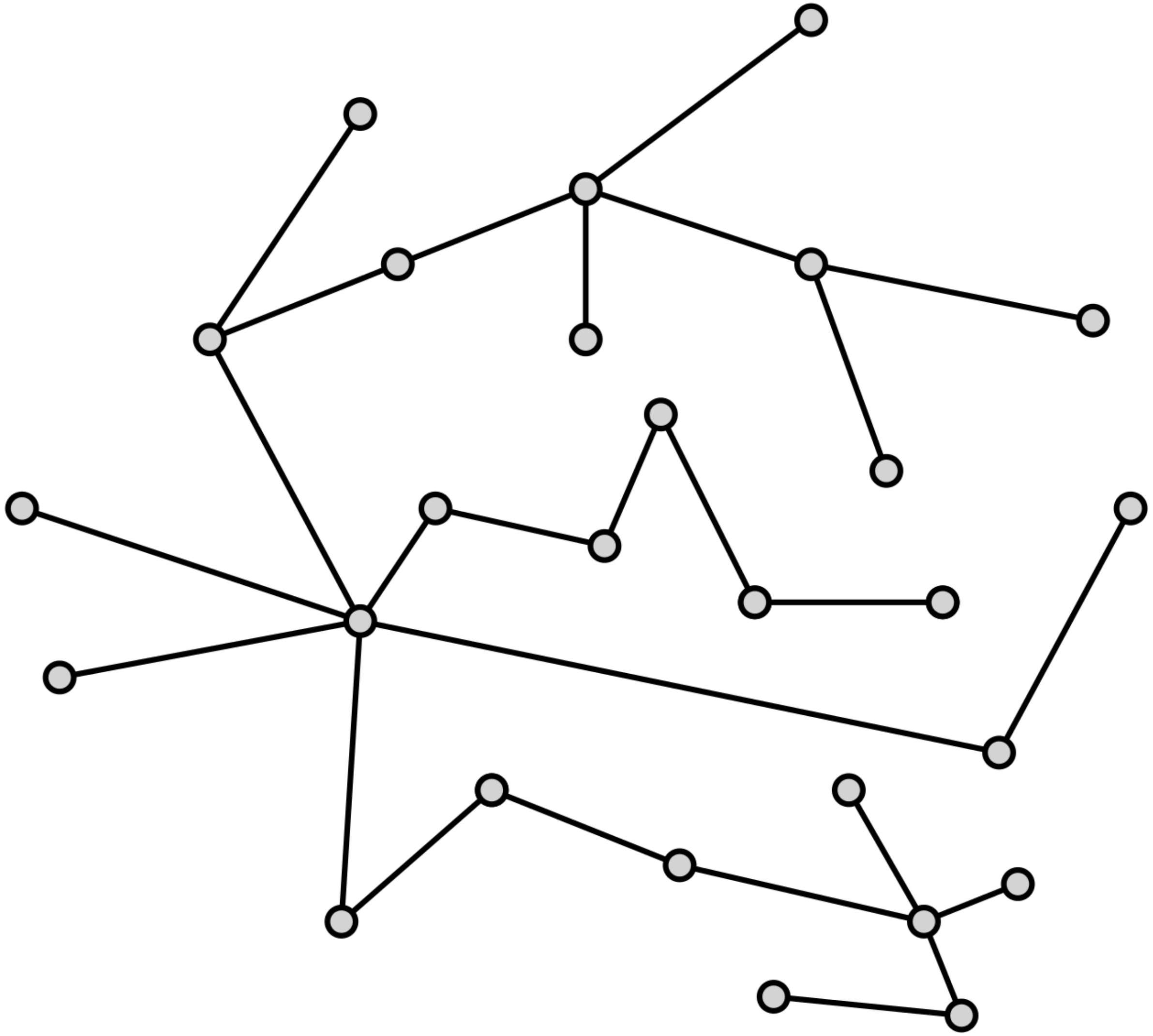
Conditional $\boldsymbol{\Omega}(\log\ \log \boldsymbol{n})$-round lower bound for $\boldsymbol{S} = \widetilde{\boldsymbol{O}}\left(\boldsymbol{n}^{\delta}\right)$

*Ghaffari, Kuhn, and Uitto* [FOCS’19]

# Algorithm

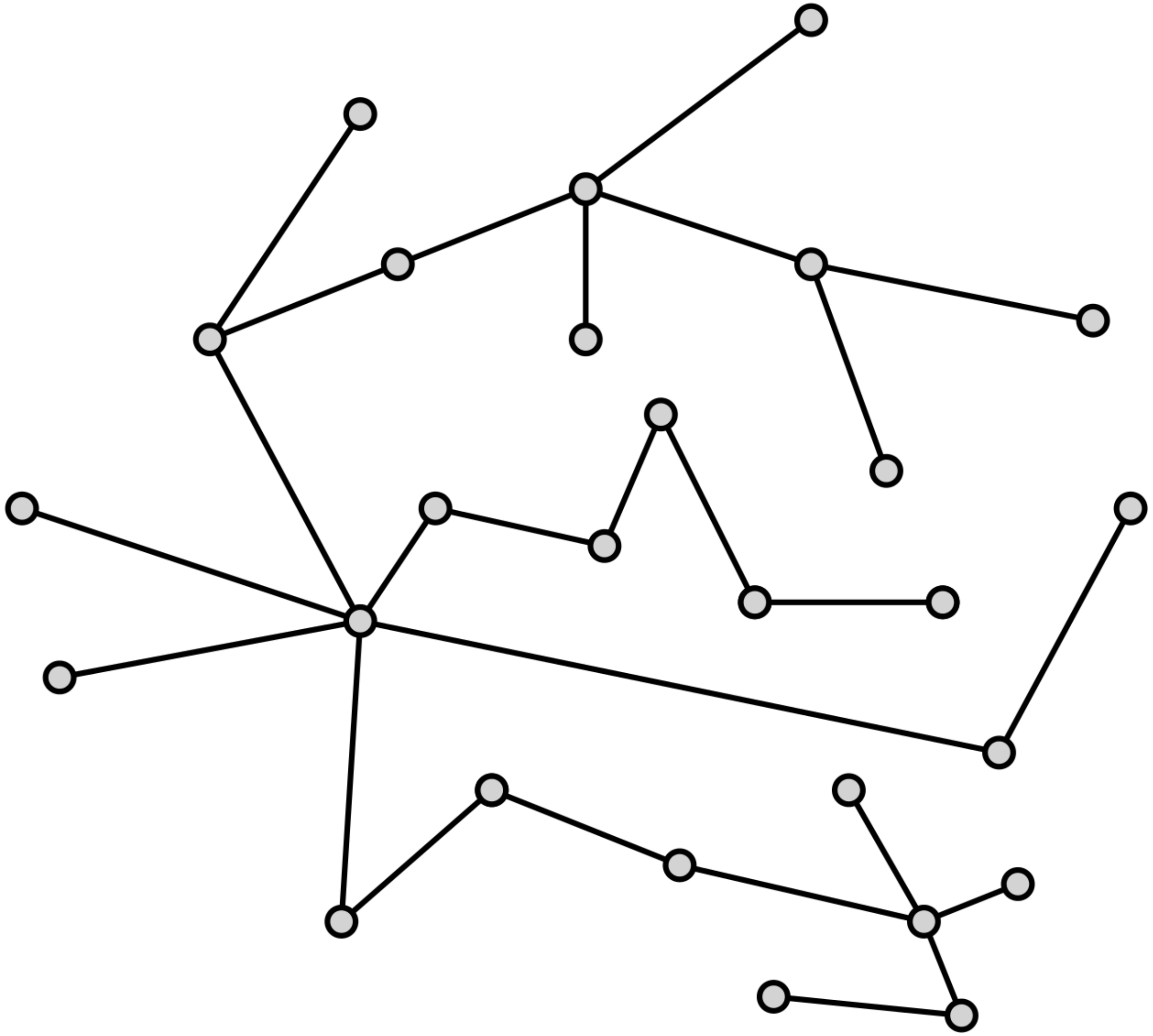
# Algorithm Outline

# Algorithm Outline



# Algorithm Outline

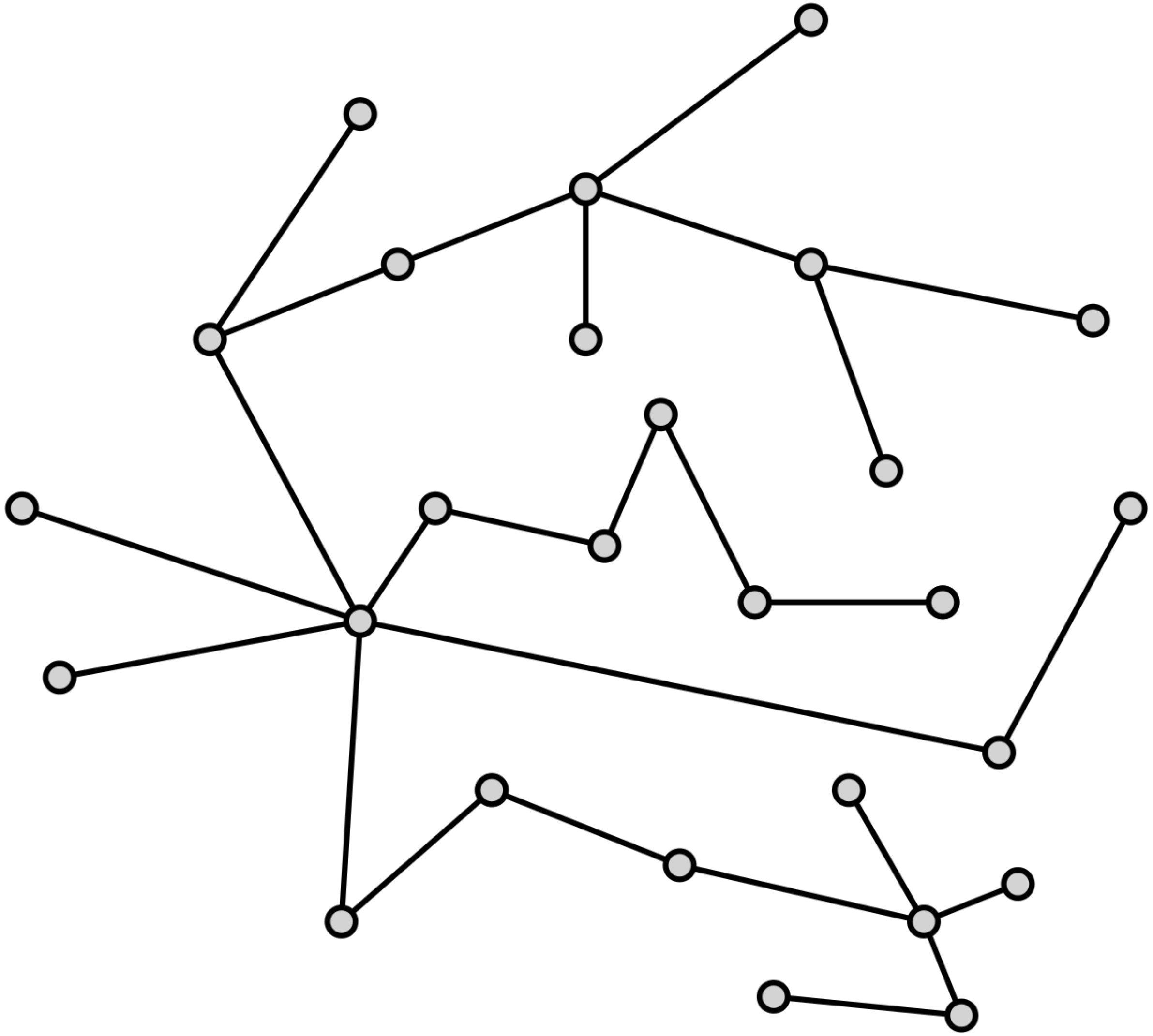
## 1) Shattering



# Algorithm Outline

## 1) Shattering

main LOCAL technique
*Beck* [RSA'91]



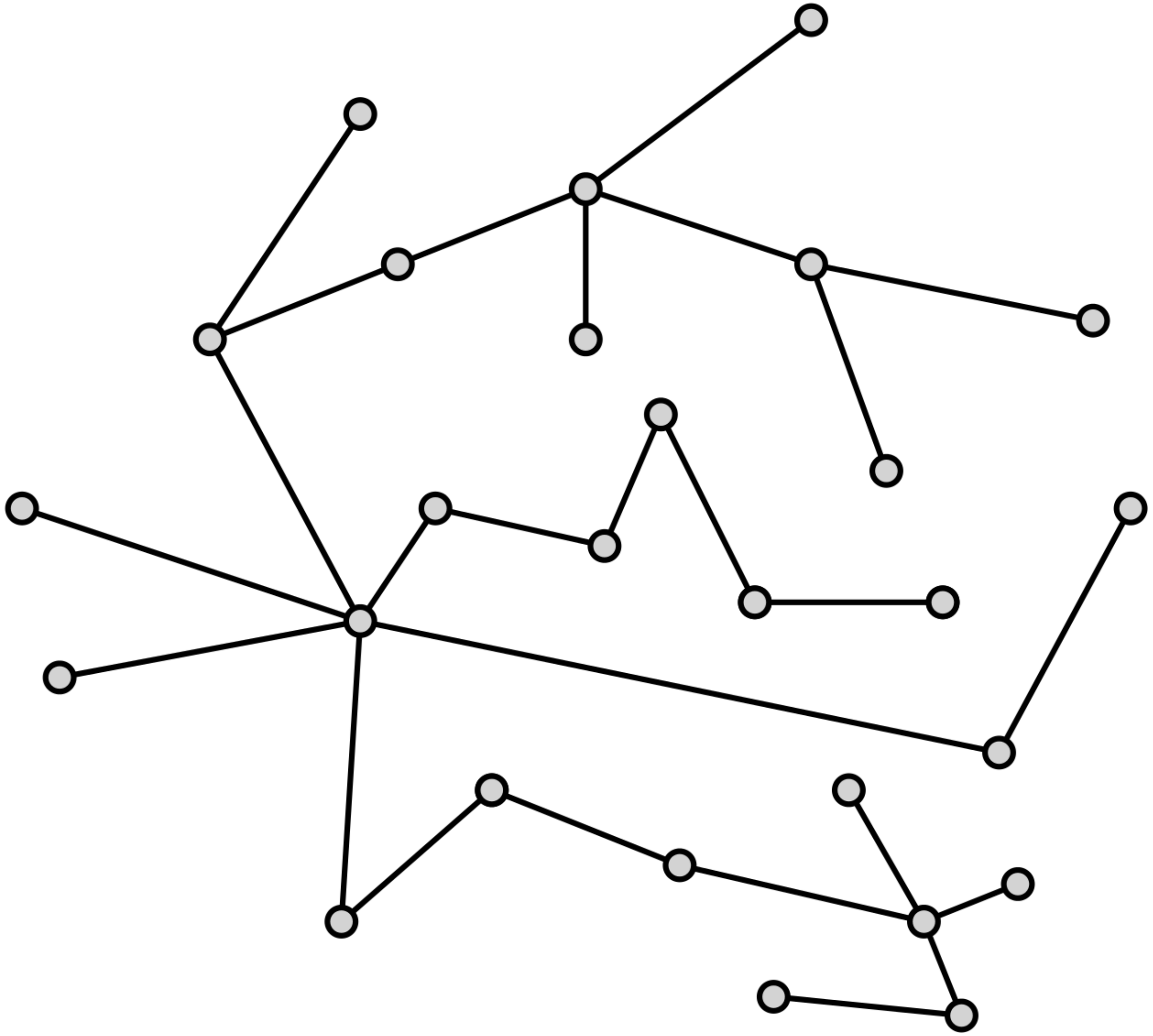
# Algorithm Outline

## 1) Shattering

break graph into small components

main LOCAL technique

*Beck* [RSA'91]



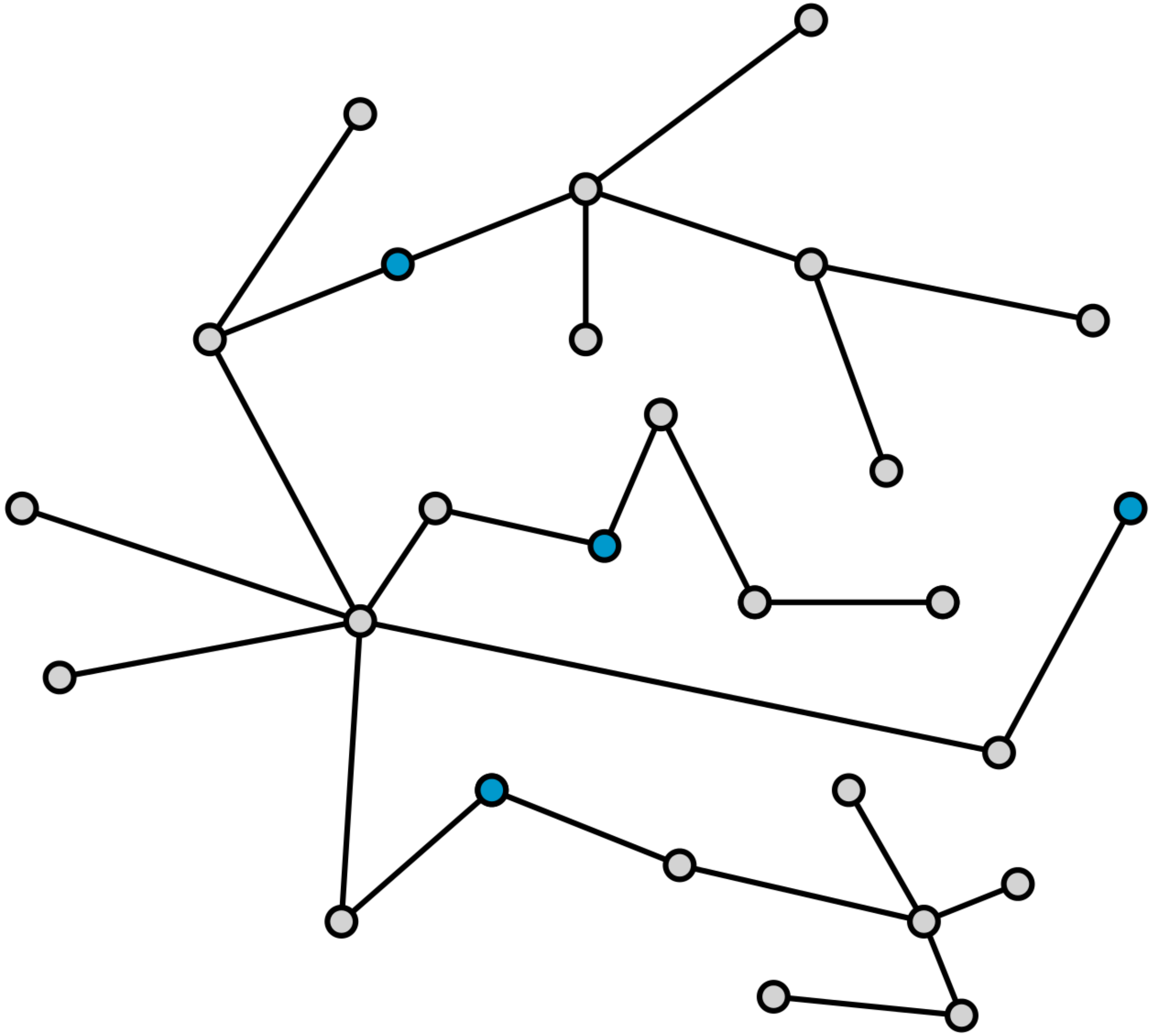
# Algorithm Outline

## 1) Shattering

break graph into small components

main LOCAL technique

*Beck* [RSA'91]



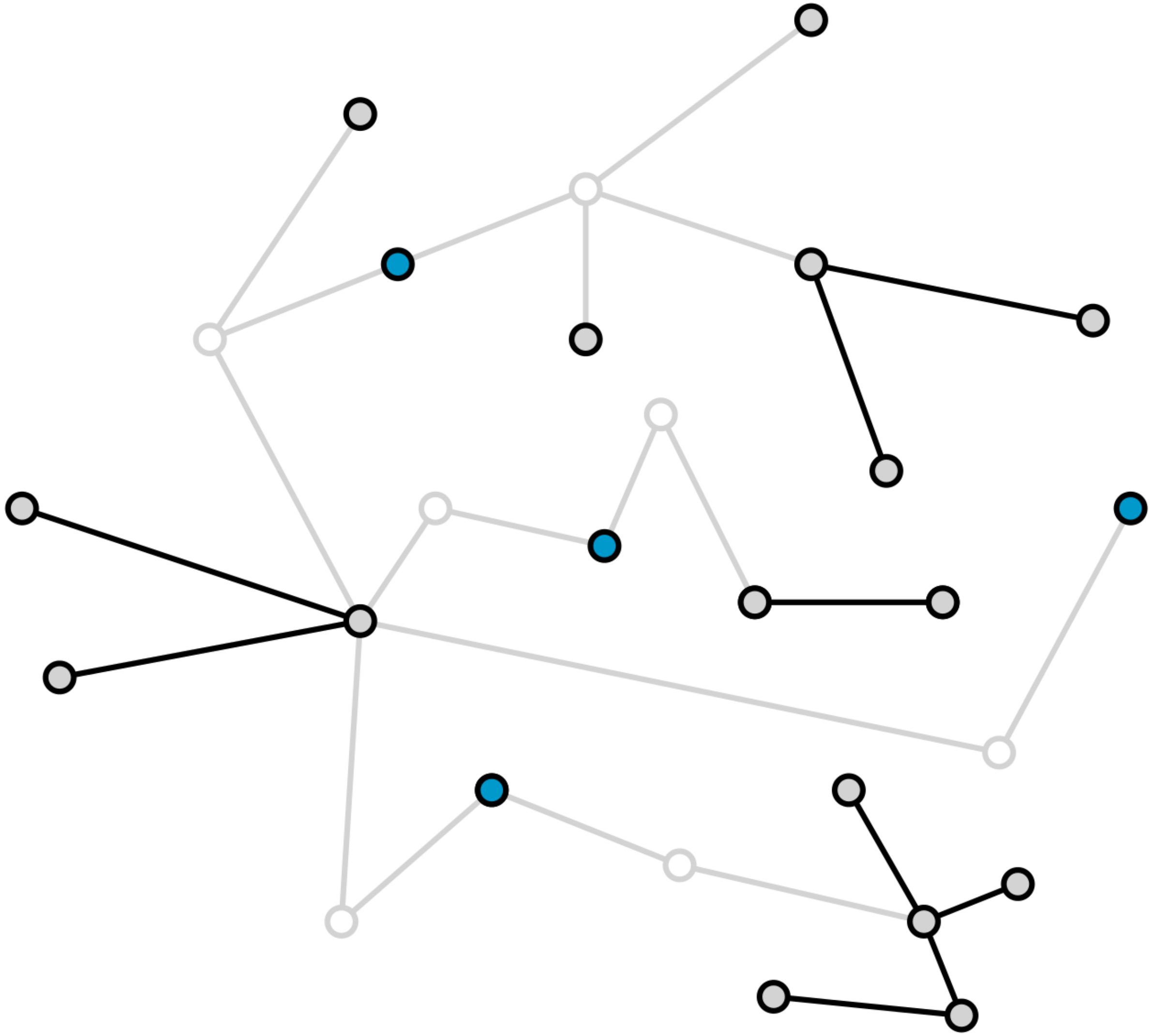
# Algorithm Outline

## 1) Shattering

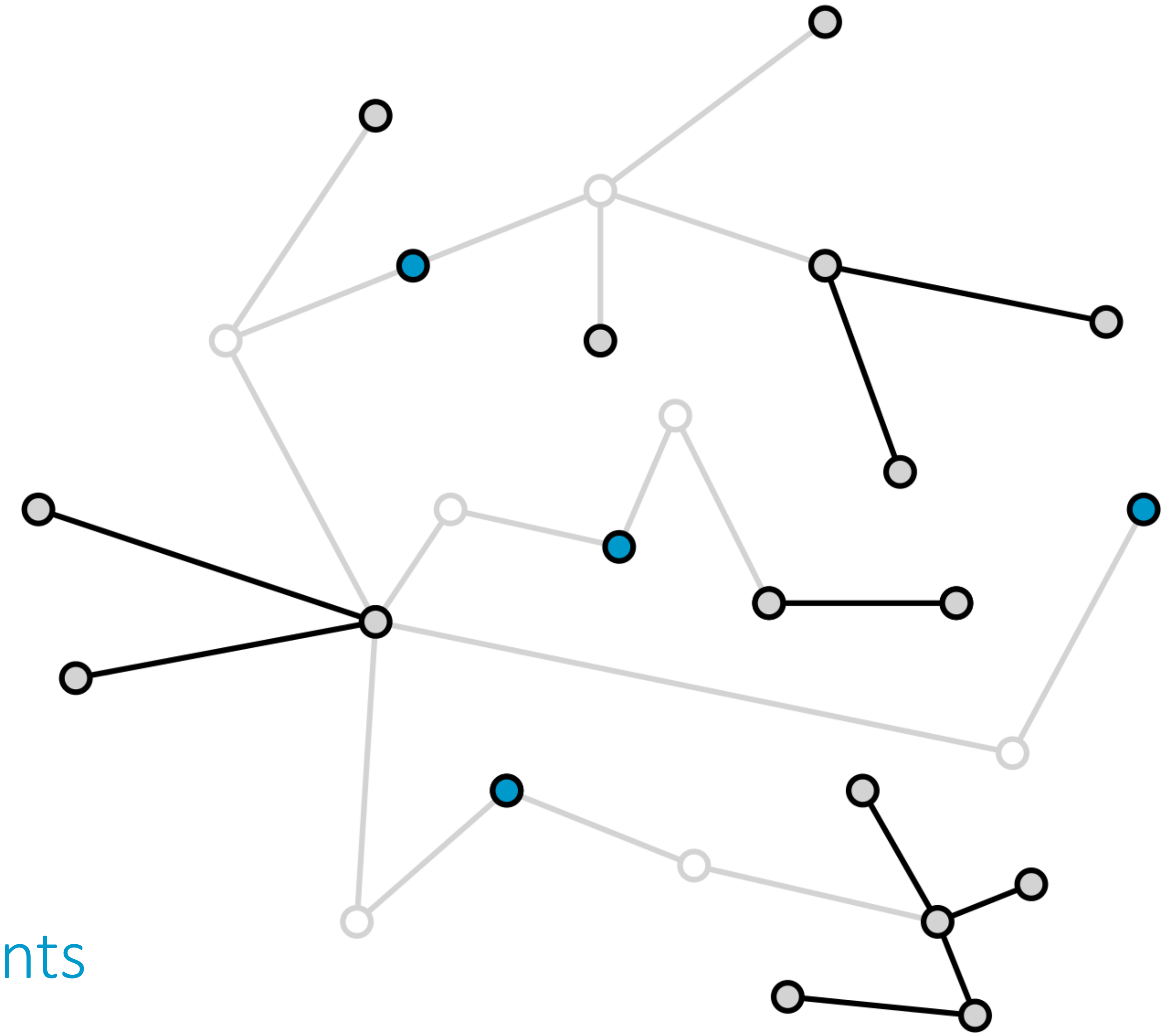
break graph into small components

main LOCAL technique

*Beck* [RSA'91]



# Algorithm Outline



## 1) Shattering

break graph into small components

main LOCAL technique

*Beck* [RSA'91]

## 2) Post-Shattering

solve problem on remaining components

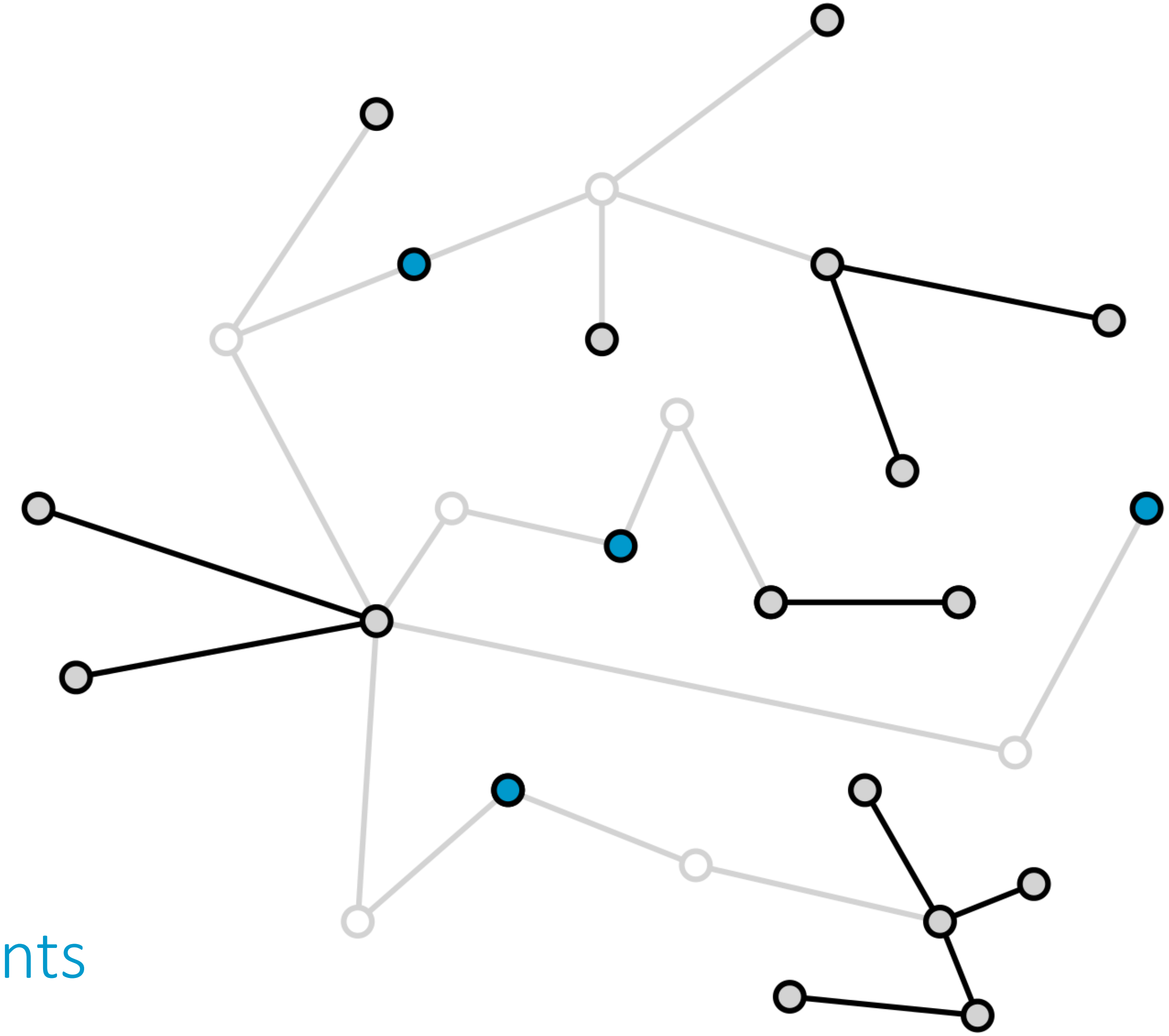
# Algorithm Outline

## 1) Shattering

break graph into small components

main LOCAL technique

*Beck* [RSA’91]



## 2) Post-Shattering

solve problem on remaining components

**i) Gathering of Components**

# Algorithm Outline

## 1) Shattering

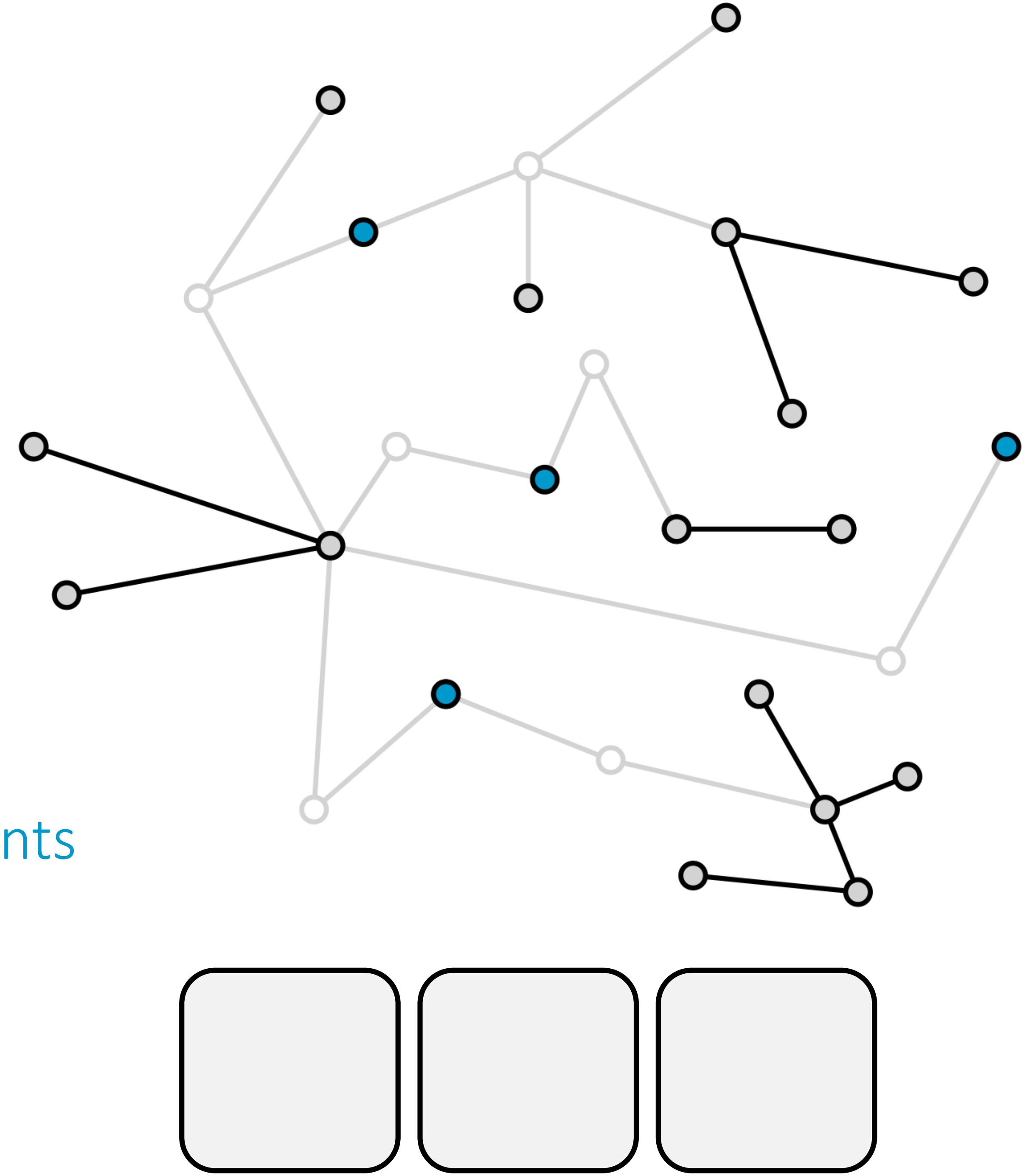
break graph into small components

main LOCAL technique
*Beck* [RSA’91]

## 2) Post-Shattering

solve problem on remaining components

i) **Gathering of Components**



# Algorithm Outline

## 1) Shattering

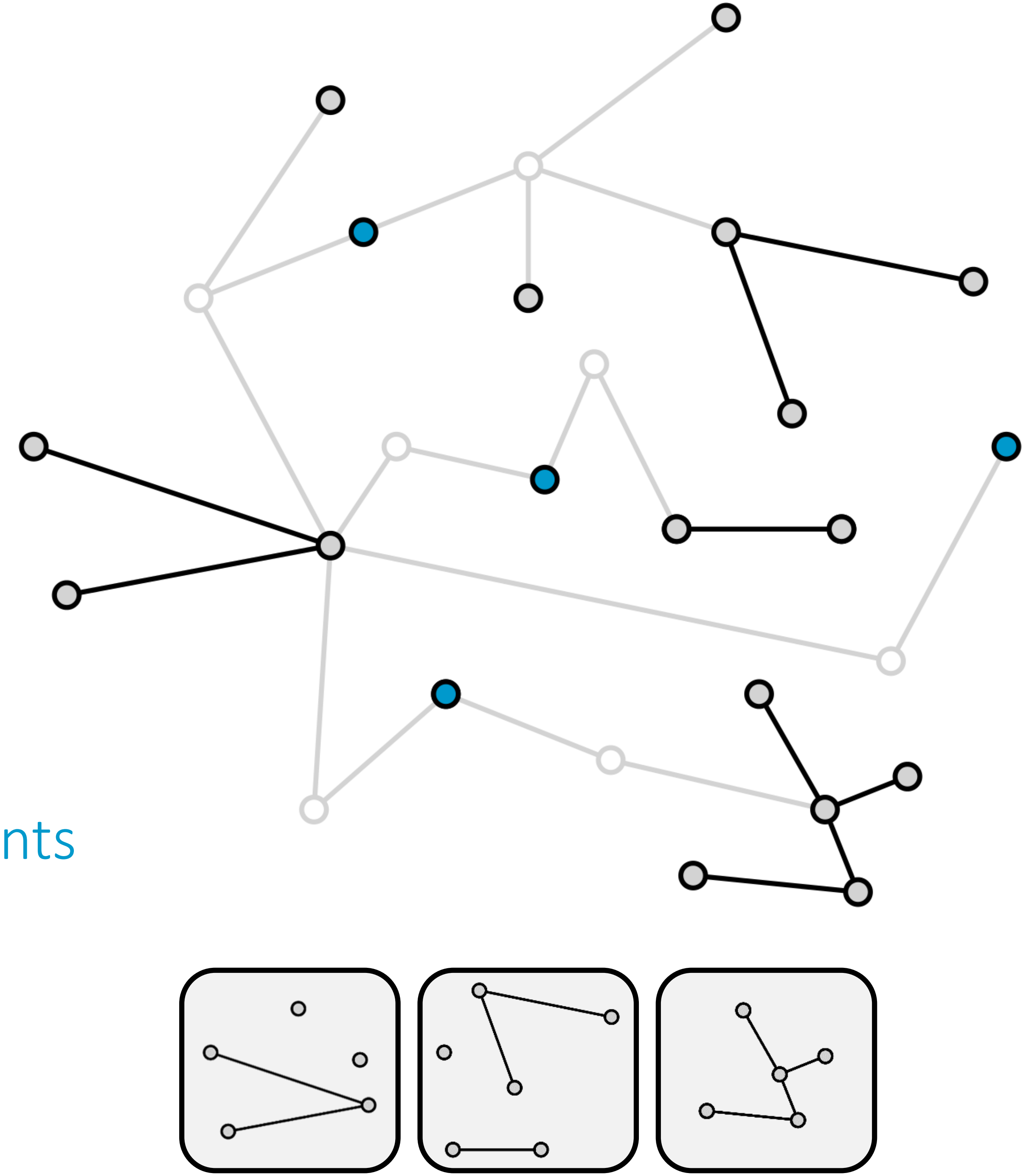
break graph into small components

main LOCAL technique
*Beck* [RSA'91]

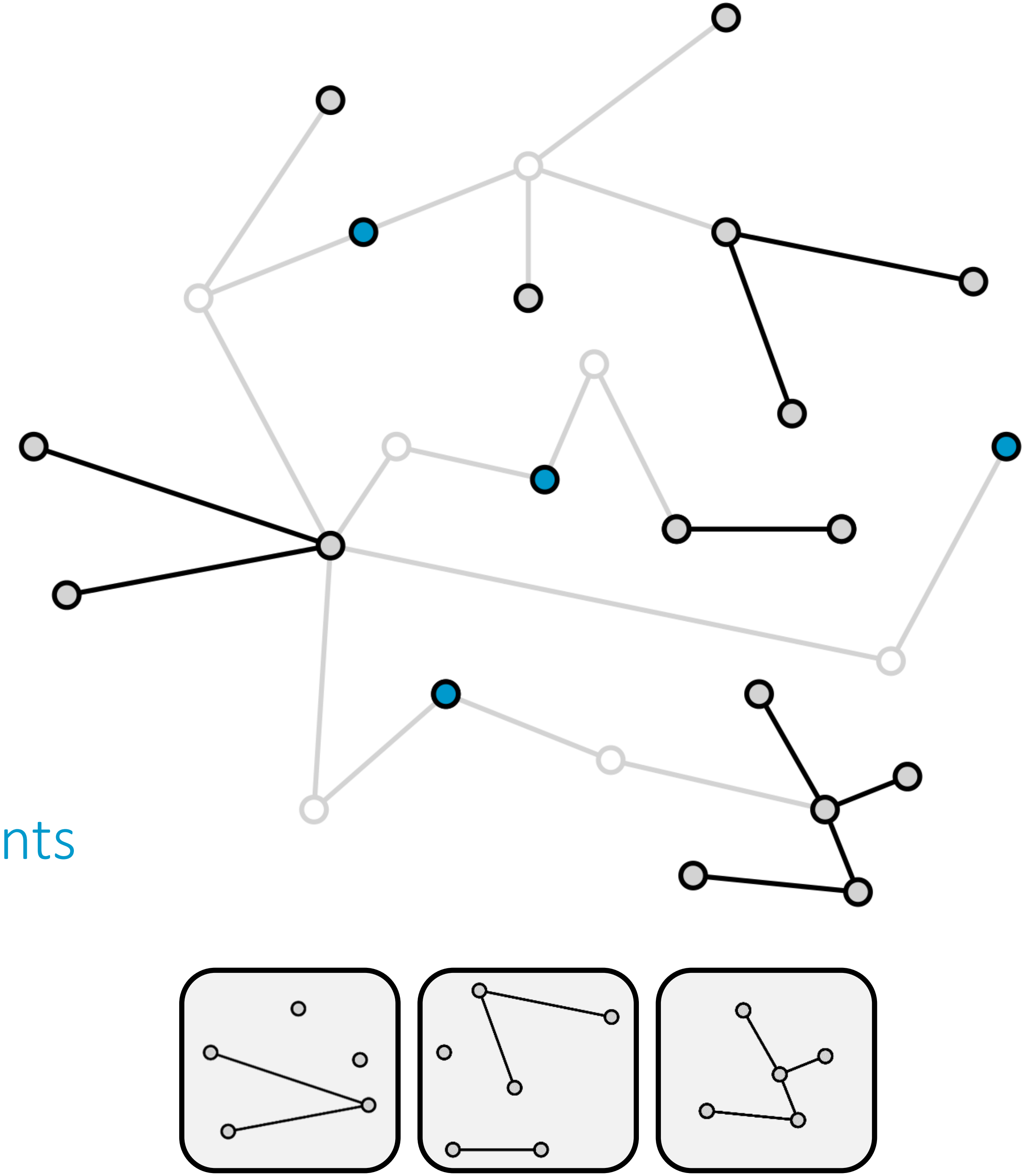
## 2) Post-Shattering

solve problem on remaining components

**i) Gathering of Components**



# Algorithm Outline



## 1) Shattering

break graph into small components

main LOCAL technique

*Beck* [RSA'91]

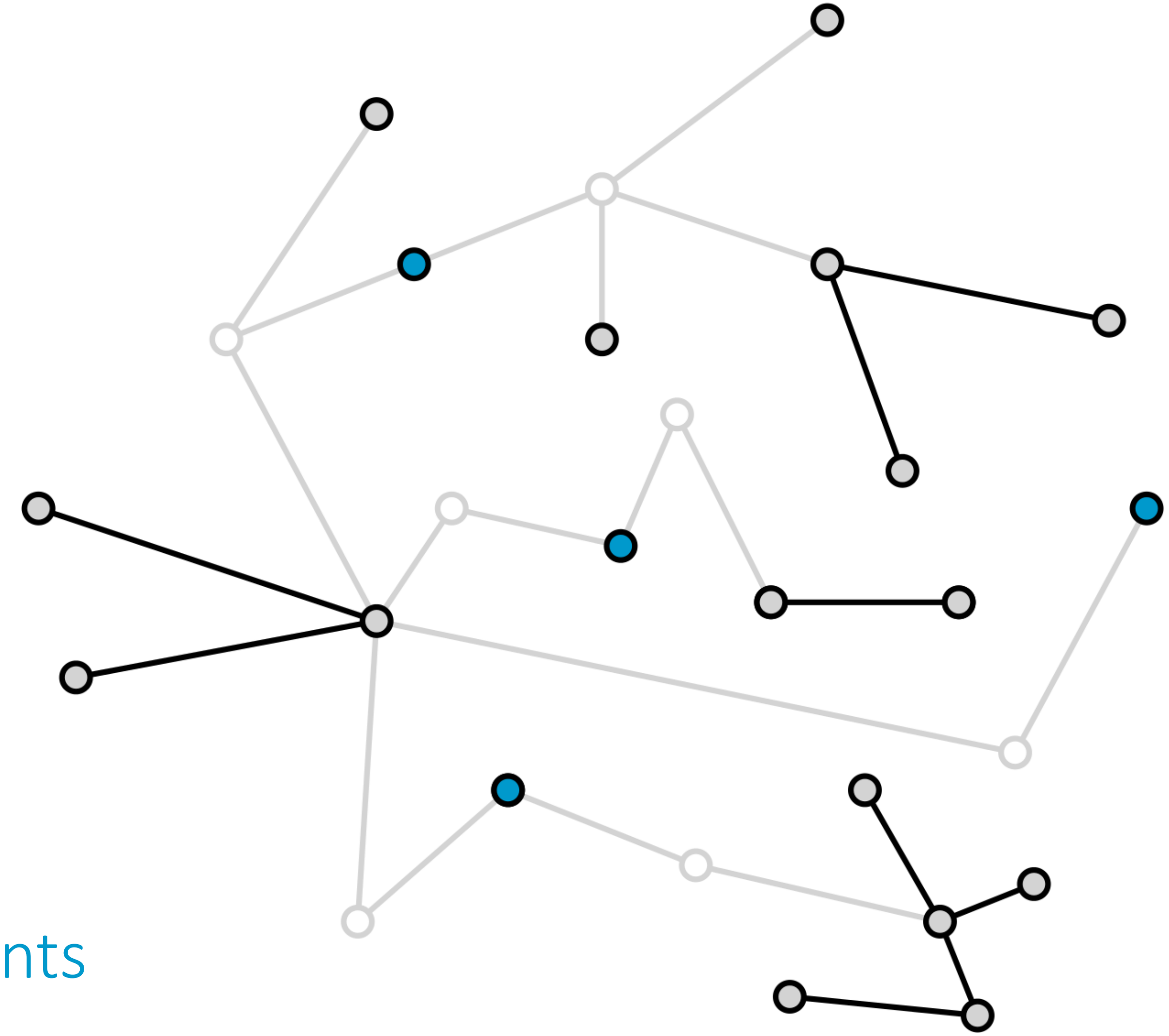
## 2) Post-Shattering

solve problem on remaining components

**i) Gathering of Components**

**ii) Local Computation**

# Algorithm Outline



## 1) Shattering

break graph into small components

main LOCAL technique

*Beck* [RSA'91]

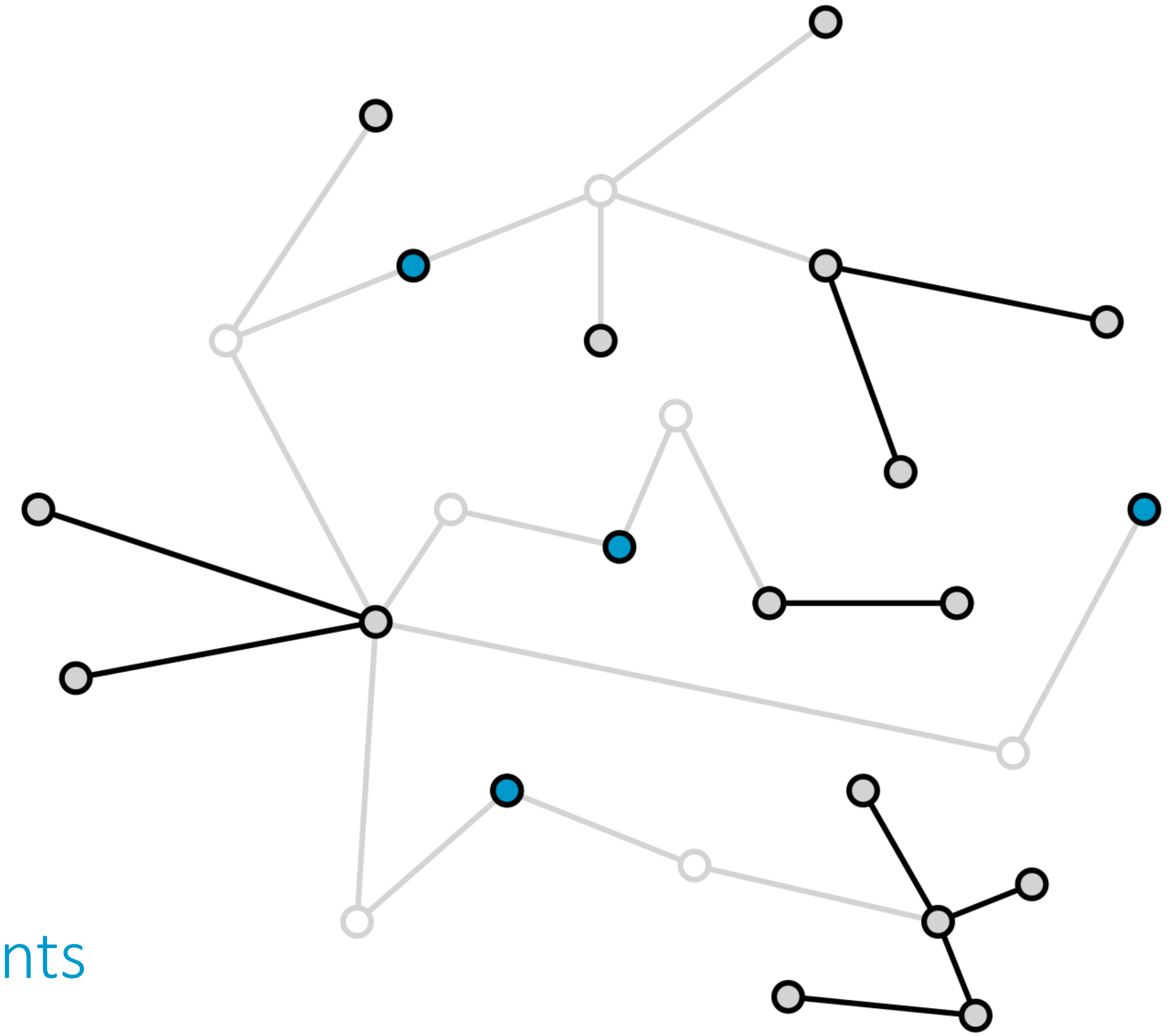
## 2) Post-Shattering

solve problem on remaining components

**i) Gathering of Components**

**ii) Local Computation**

# Algorithm Outline



## 1) Shattering

break graph into small components

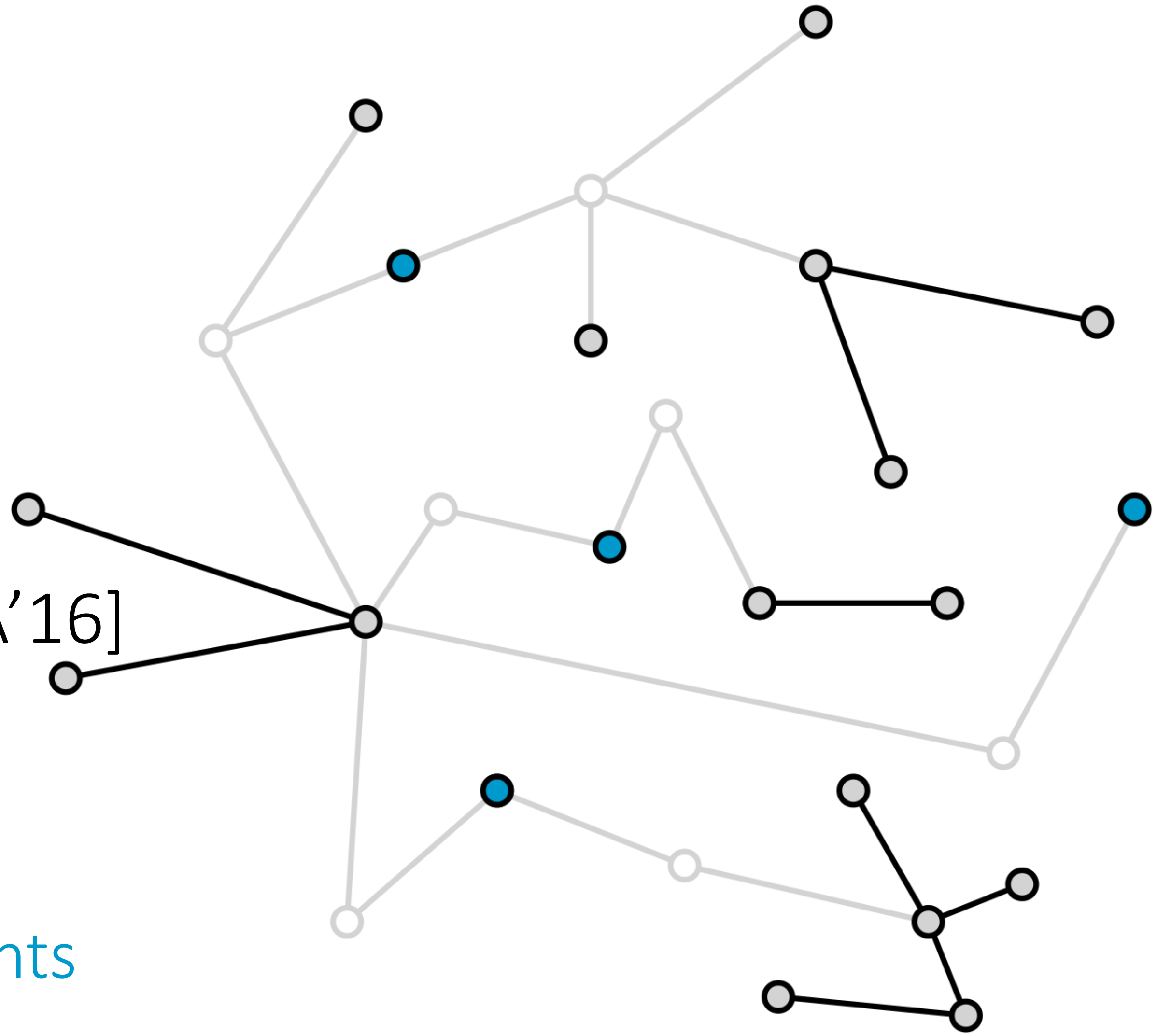
## 2) Post-Shattering

solve problem on remaining components

i) **Gathering of Components**

ii) **Local Computation**

# Algorithm Outline



## 1) Shattering

break graph into small components

**ii) LOCAL Shattering** *Ghaffari* [SODA'16]

## 2) Post-Shattering

solve problem on remaining components

**i) Gathering of Components**

**ii) Local Computation**

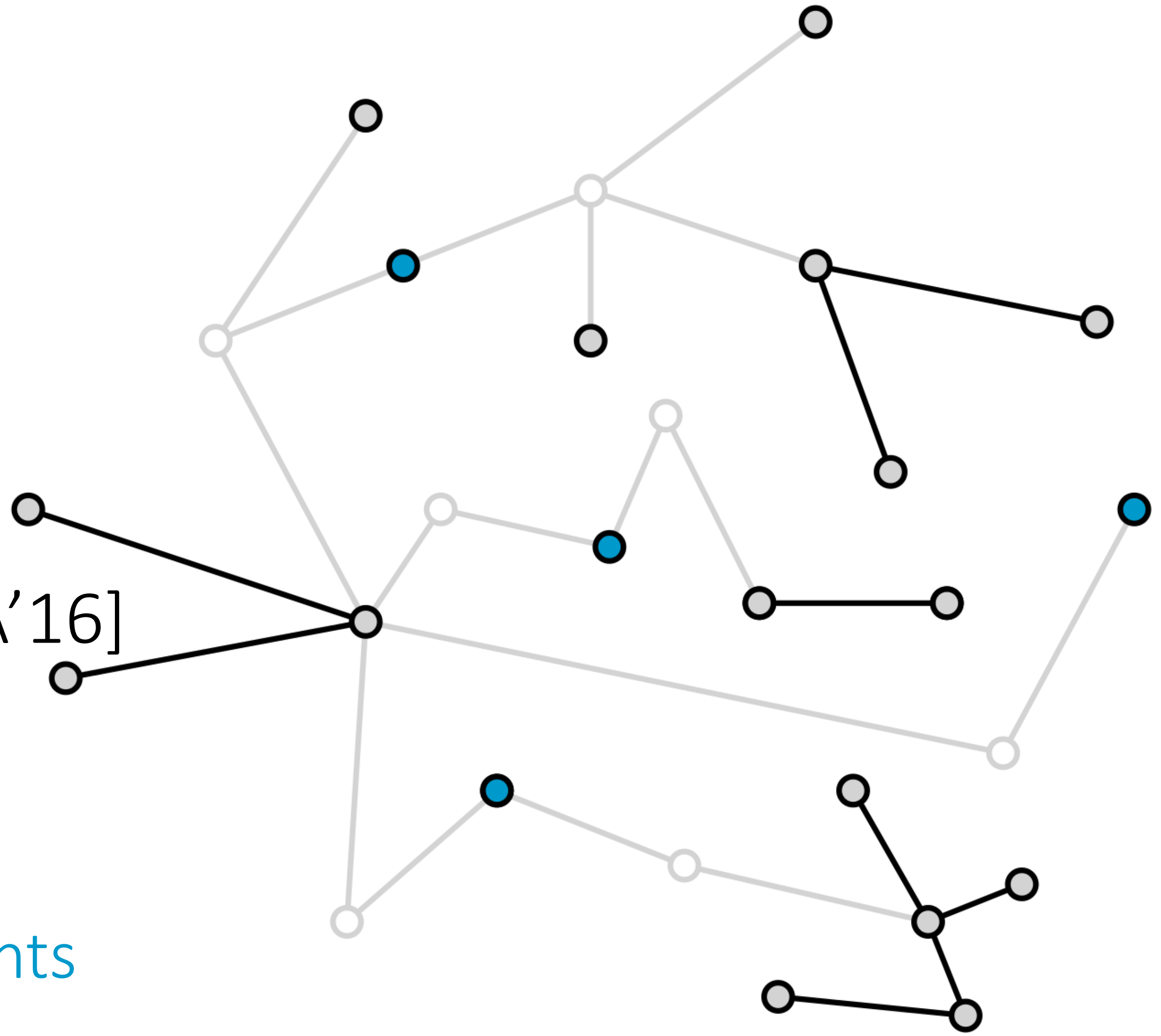
# Algorithm Outline

## 1) Shattering

break graph into small components

**i) Degree Reduction**

**ii) LOCAL Shattering** *Ghaffari* [SODA'16]



## 2) Post-Shattering

solve problem on remaining components

**i) Gathering of Components**

**ii) Local Computation**

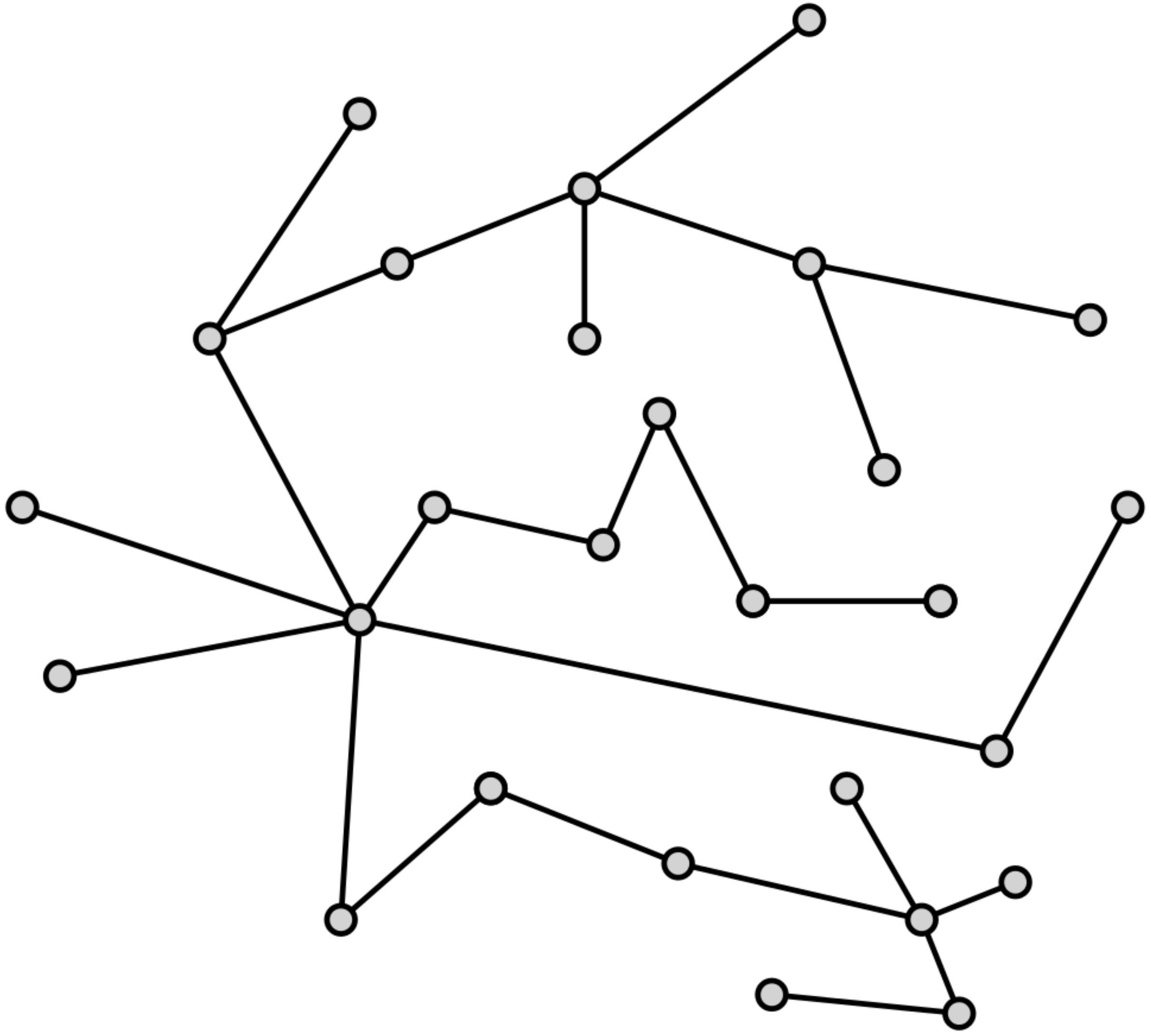
# Polynomial Degree Reduction:
# **Subsample-and-Conquer**

# Polynomial Degree Reduction:
# **Subsample-and-Conquer**

**Subsample**

**Conquer**

# Polynomial Degree Reduction:
# **Subsample-and-Conquer**



**Subsample**

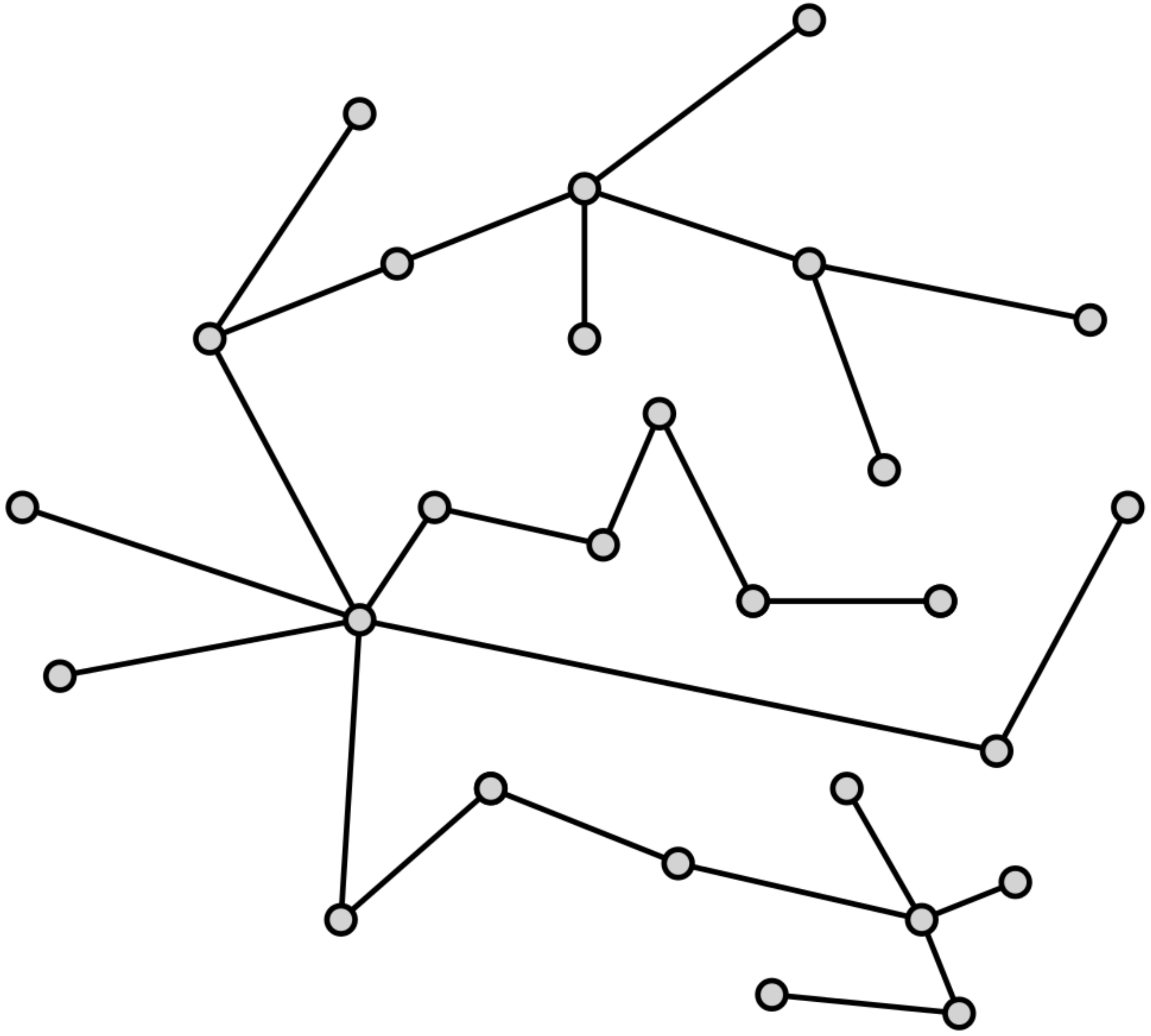
**Conquer**

## Polynomial Degree Reduction:
# Subsample-and-Conquer

**Subsample**

subsample **nodes** independently

**Conquer**



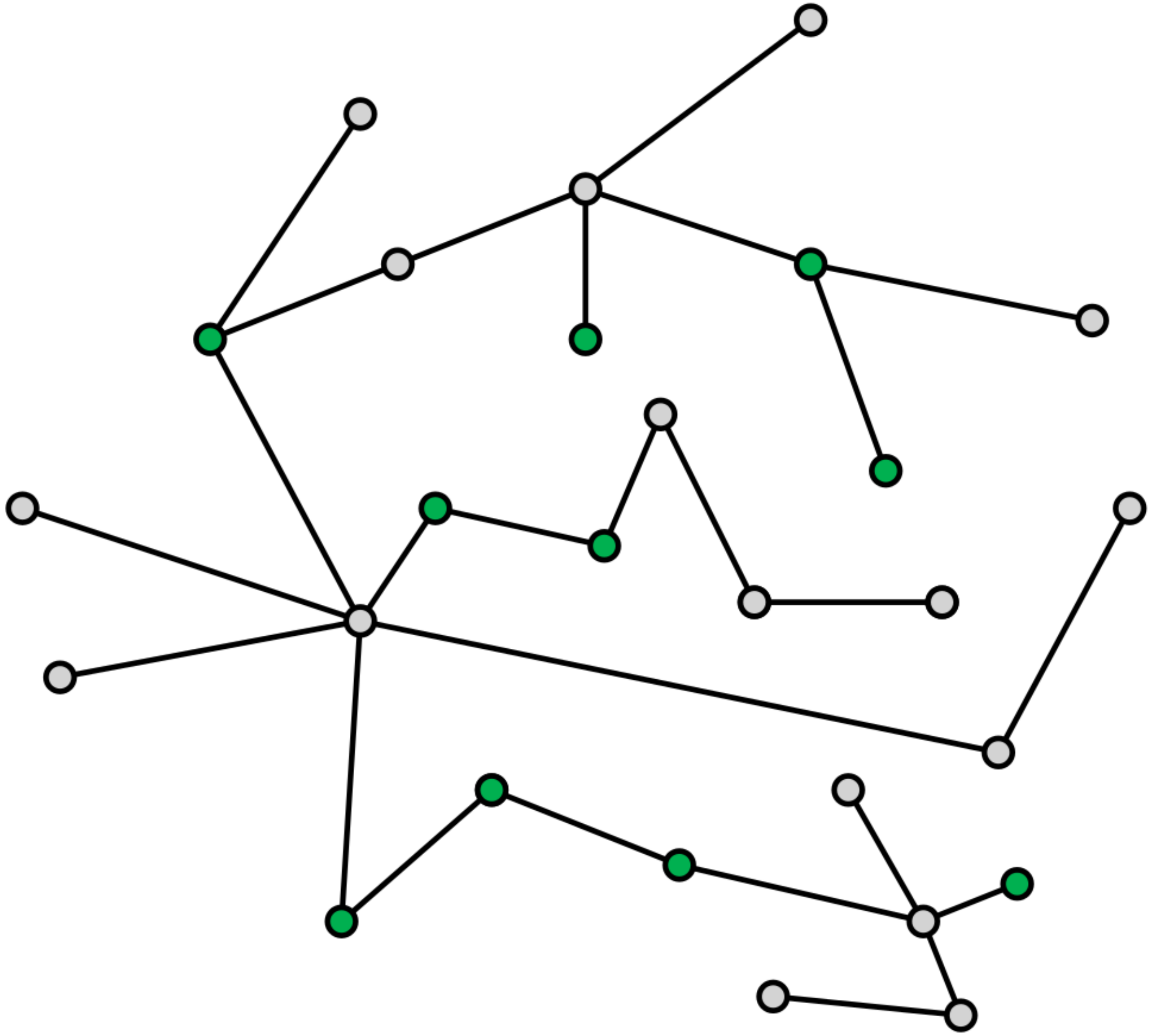
Polynomial Degree Reduction:

# Subsample-and-Conquer

**Subsample**

subsample **nodes** independently

**Conquer**



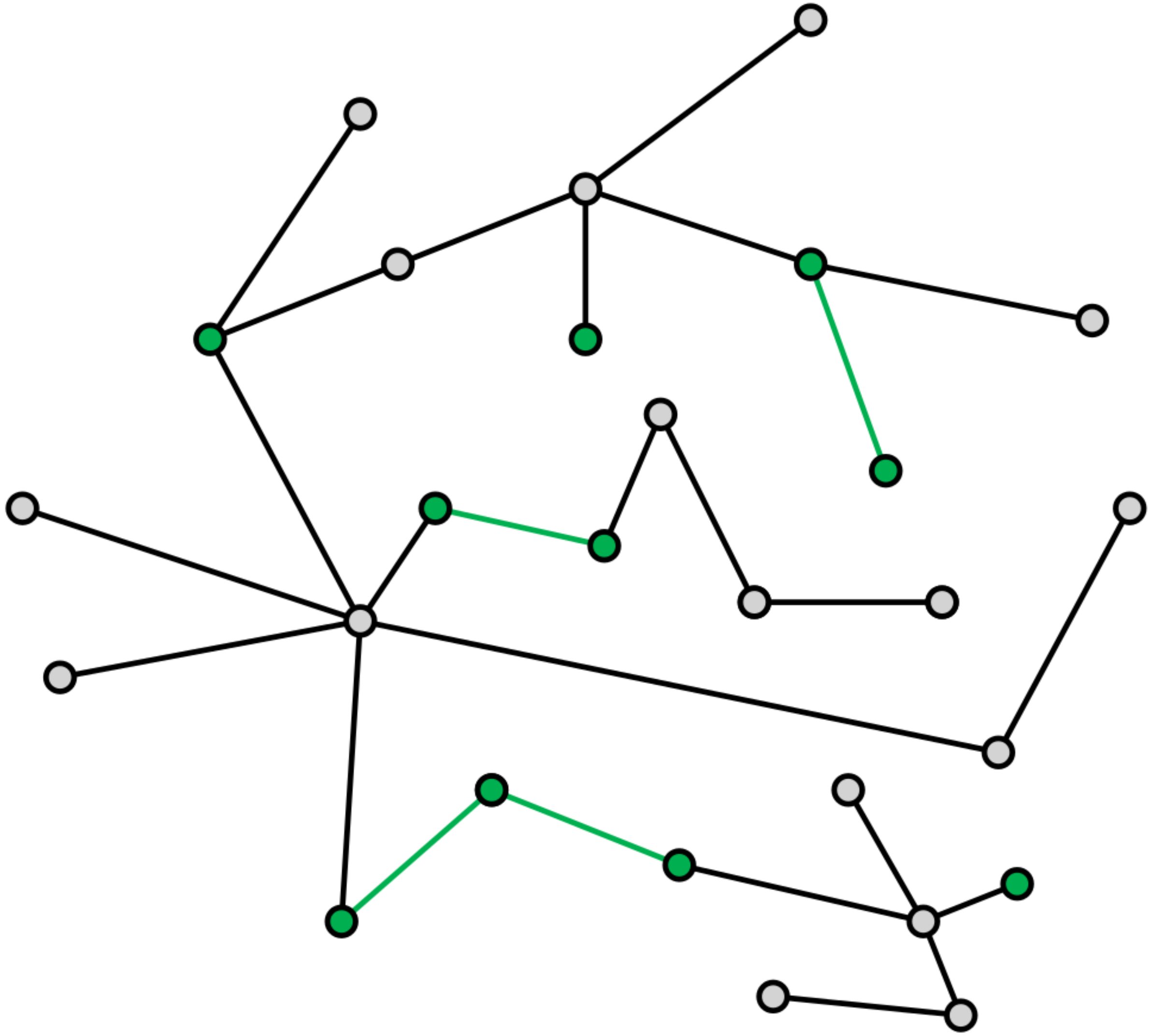
## Polynomial Degree Reduction:

# Subsample-and-Conquer

**Subsample**

subsample **nodes** independently

**Conquer**



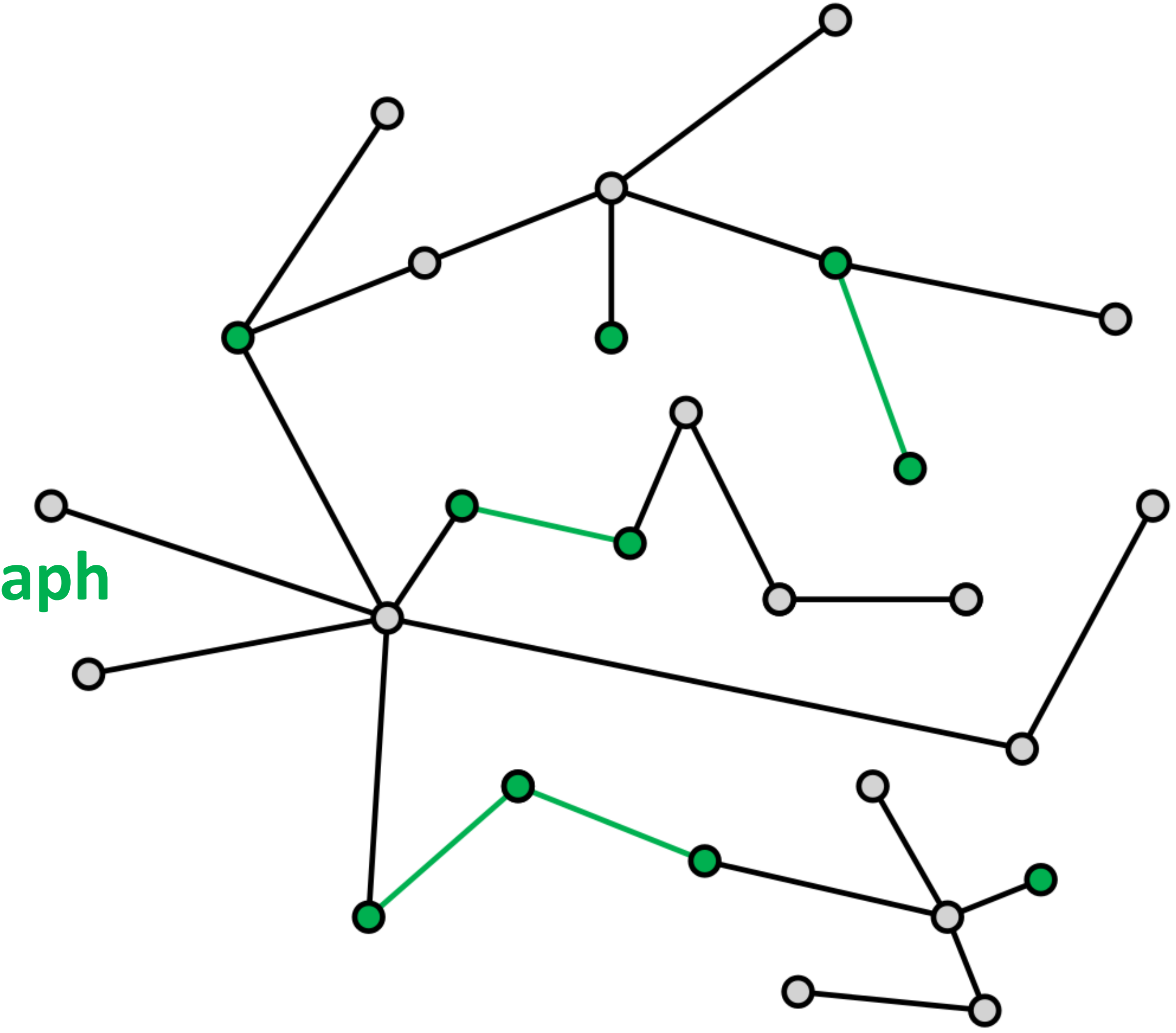
# Polynomial Degree Reduction:
# **Subsample-and-Conquer**

**Subsample**

subsample **nodes** independently

**Conquer**

compute **random MIS** in **subsampled graph**



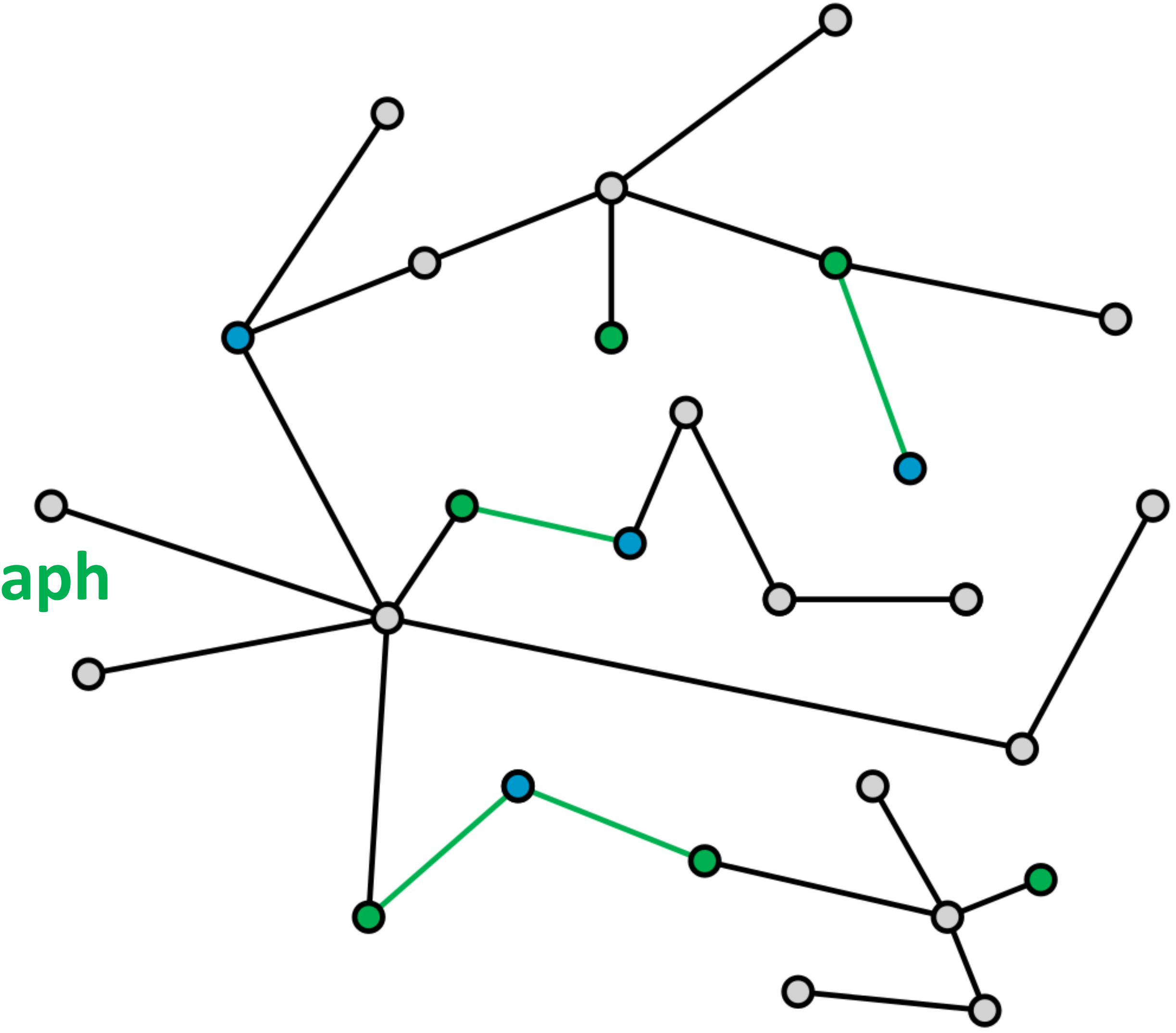
# Polynomial Degree Reduction:
# **Subsample-and-Conquer**

**Subsample**

subsample **nodes** independently

**Conquer**

compute **random MIS** in **subsampled graph**



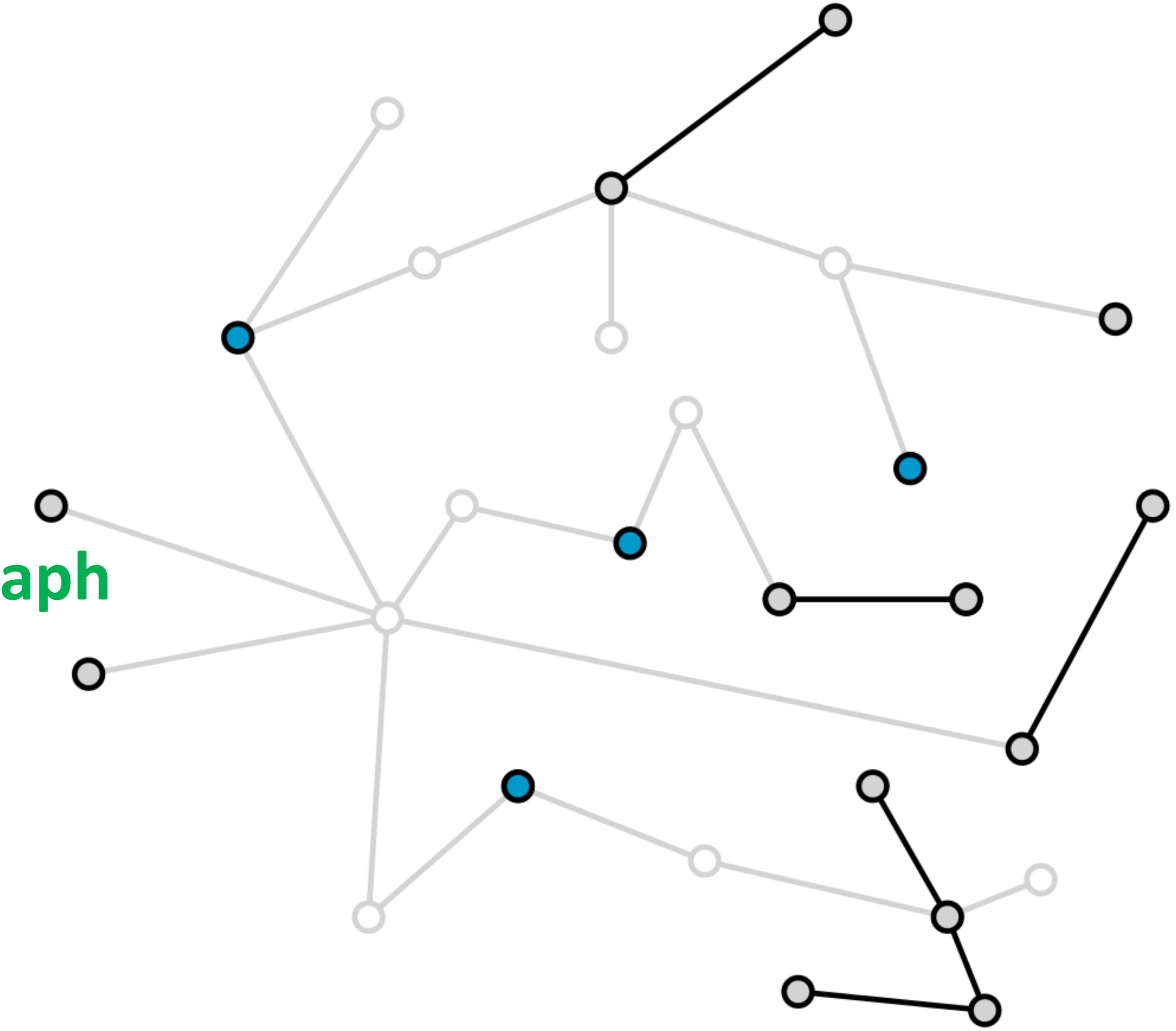
## Polynomial Degree Reduction:
# Subsample-and-Conquer

**Subsample**

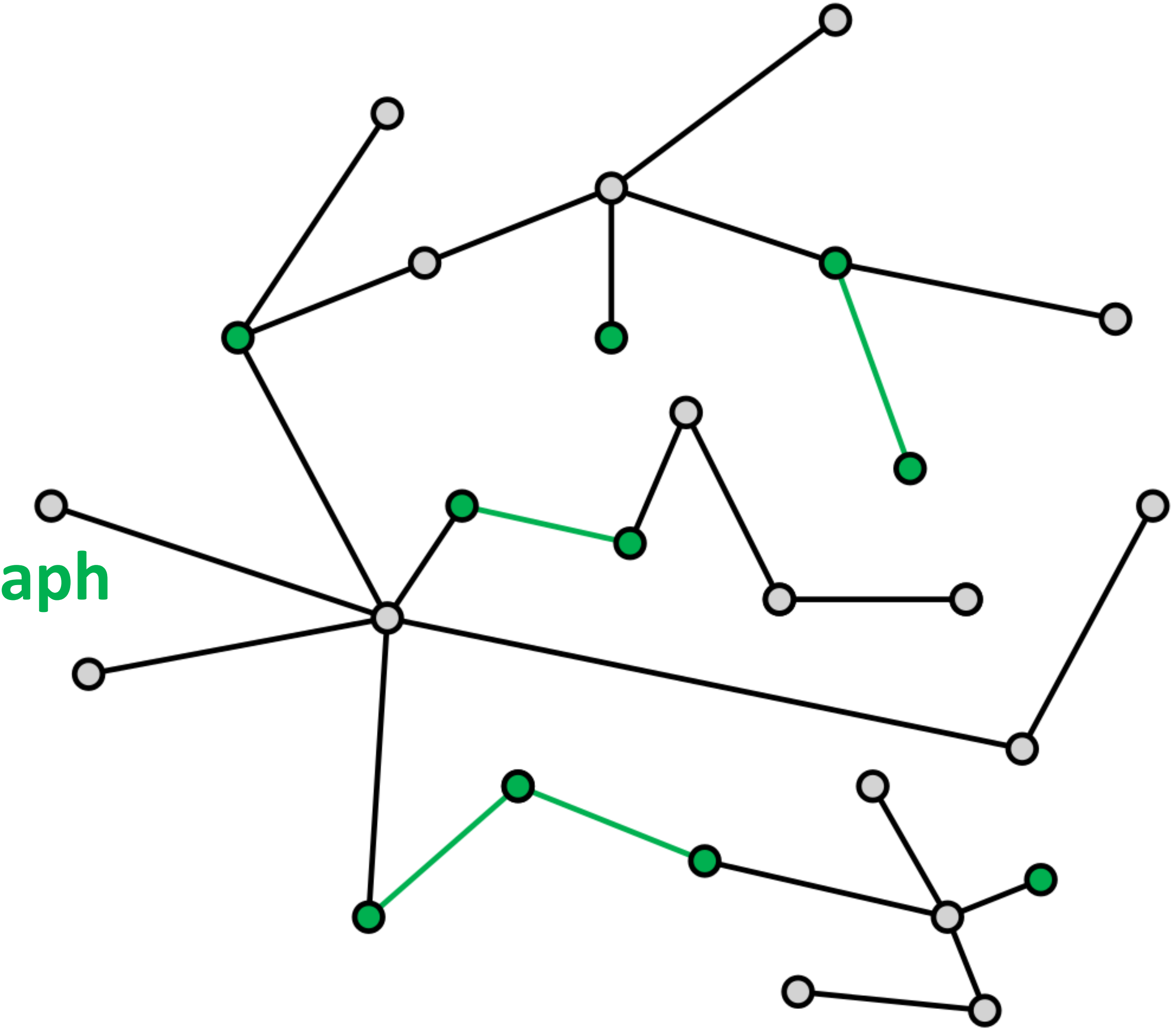
subsample **nodes** independently

**Conquer**

compute **random MIS** in **subsampled graph**



# Polynomial Degree Reduction:
# **Subsample-and-Conquer**



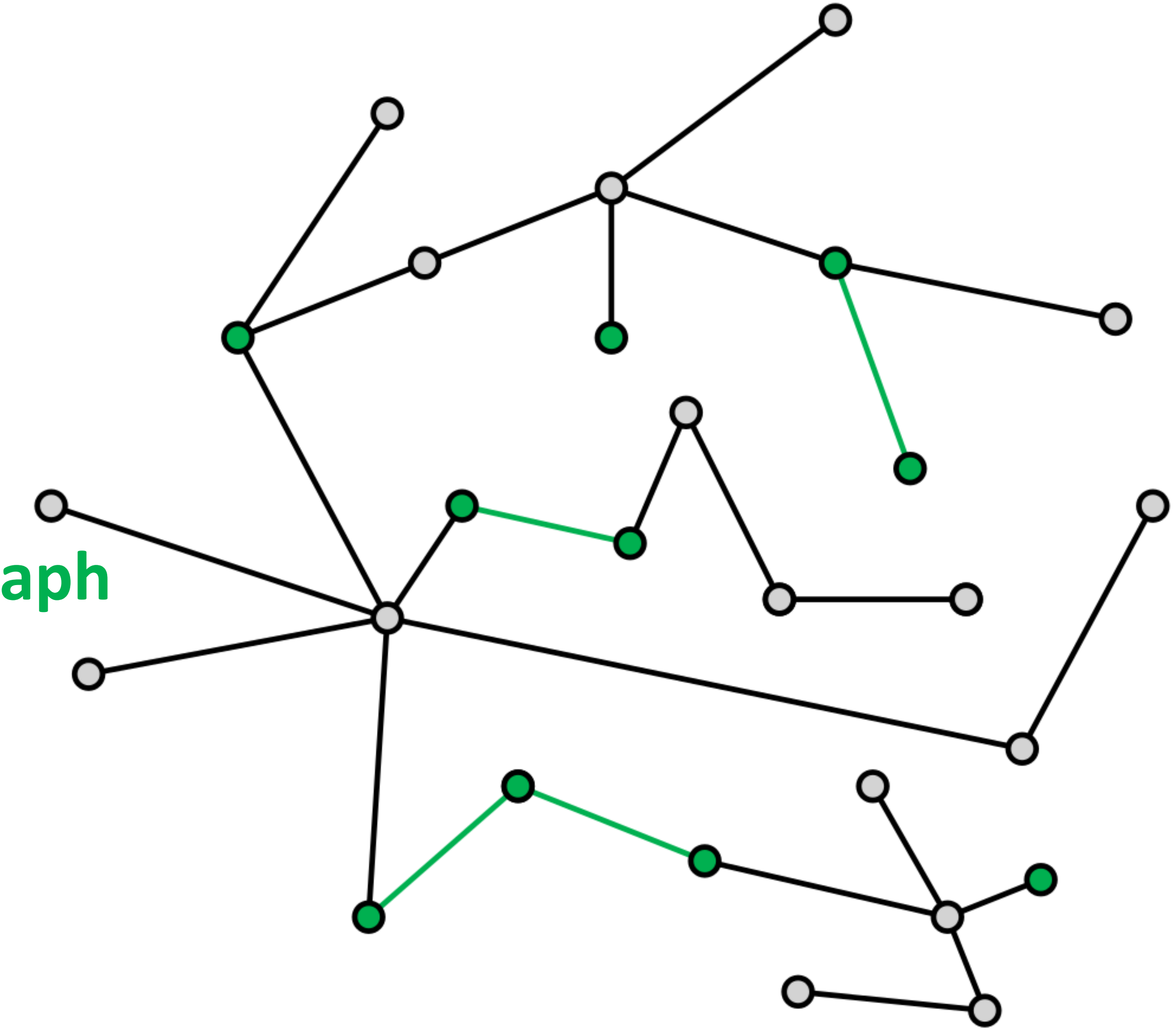
**Subsample**

subsample **nodes** independently

**Conquer**

compute **random MIS** in **subsampled graph**

# Polynomial Degree Reduction:
# **Subsample-and-Conquer**



**Subsample**

subsample **nodes** independently

**Conquer**

compute **random MIS** in **subsampled graph**

- gather connected components

# Polynomial Degree Reduction:
# Subsample-and-Conquer

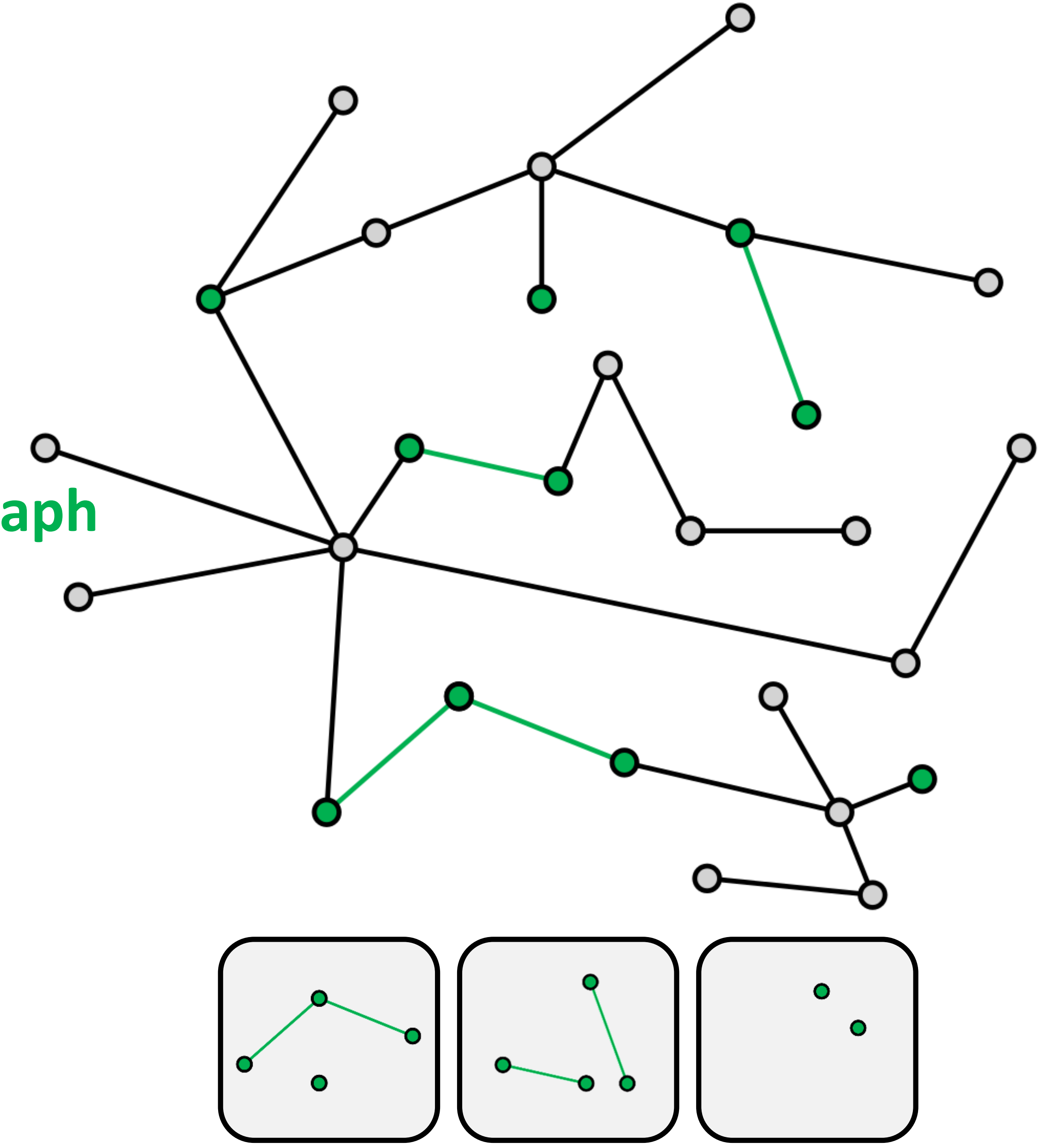
**Subsample**

subsample **nodes** independently

**Conquer**

compute **random MIS** in **subsampled graph**

- gather connected components



# Polynomial Degree Reduction:
# **Subsample-and-Conquer**

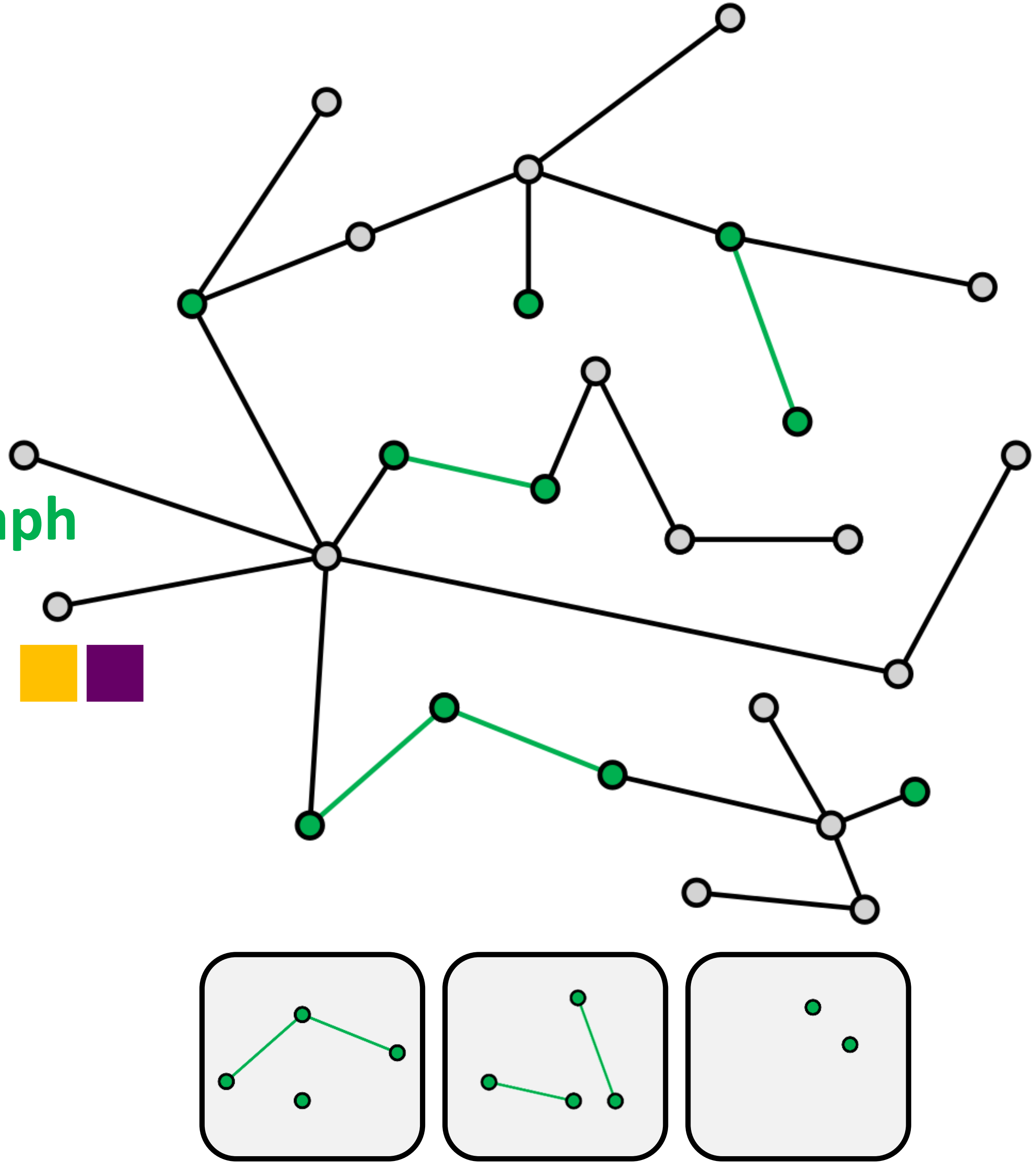
### Subsample

subsample **nodes** independently

### Conquer

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨🟪



# Polynomial Degree Reduction:
# **Subsample-and-Conquer**

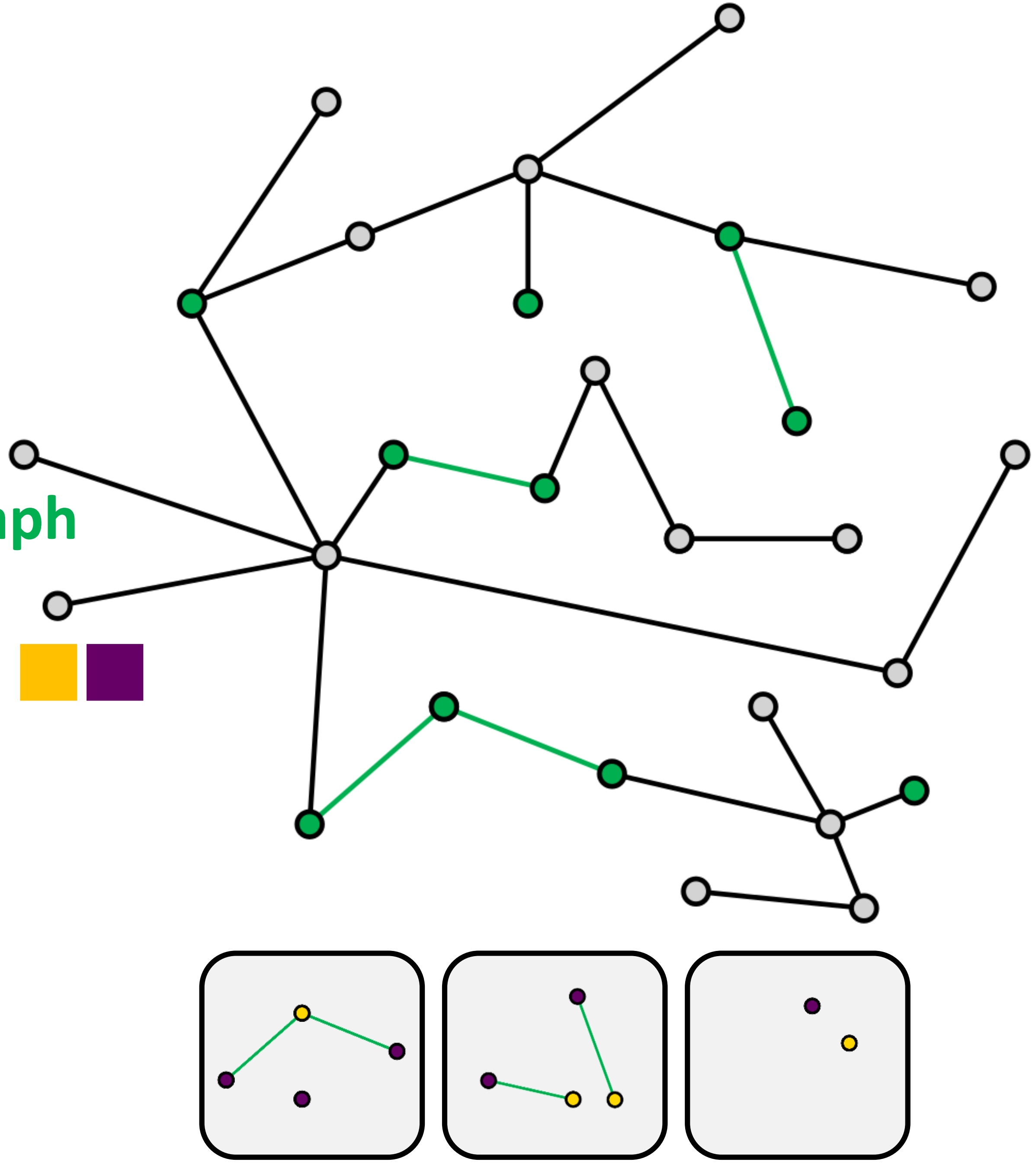
## **Subsample**

subsample **nodes** independently

## **Conquer**

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨 🟪



Polynomial Degree Reduction:

# Subsample-and-Conquer

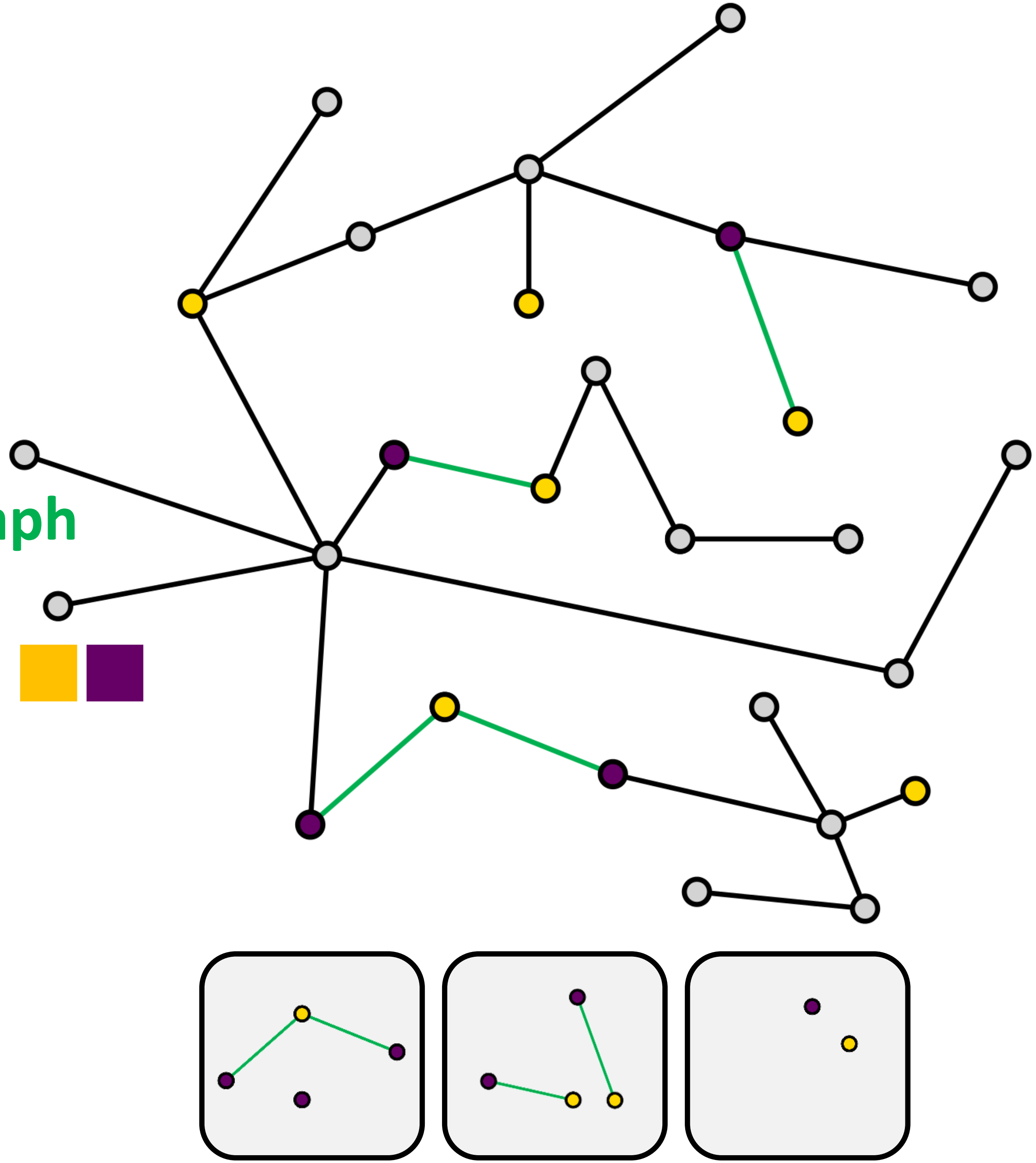
### Subsample

subsample **nodes** independently

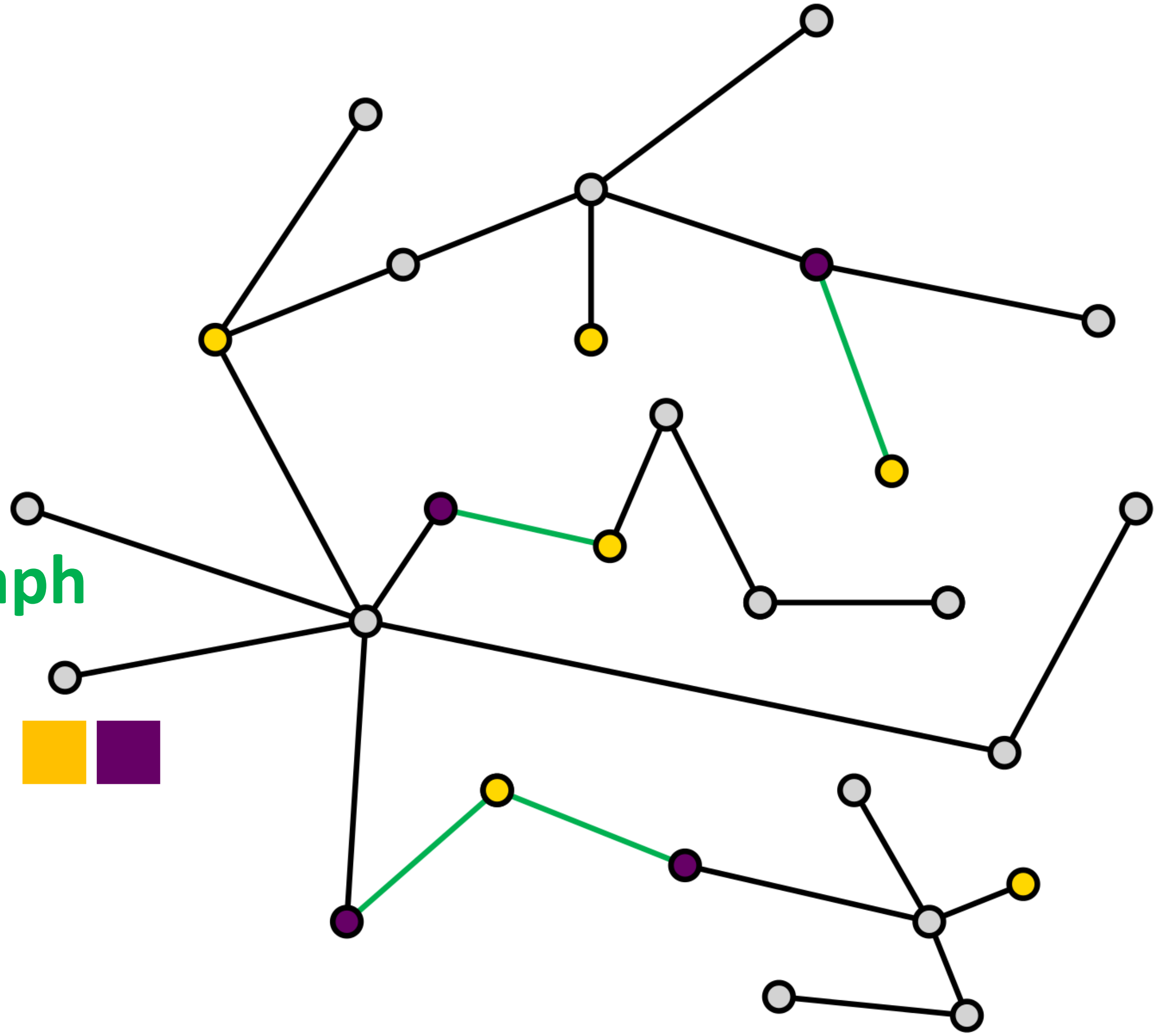
### Conquer

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨 🟪



# Polynomial Degree Reduction:
# **Subsample-and-Conquer**



**Subsample**

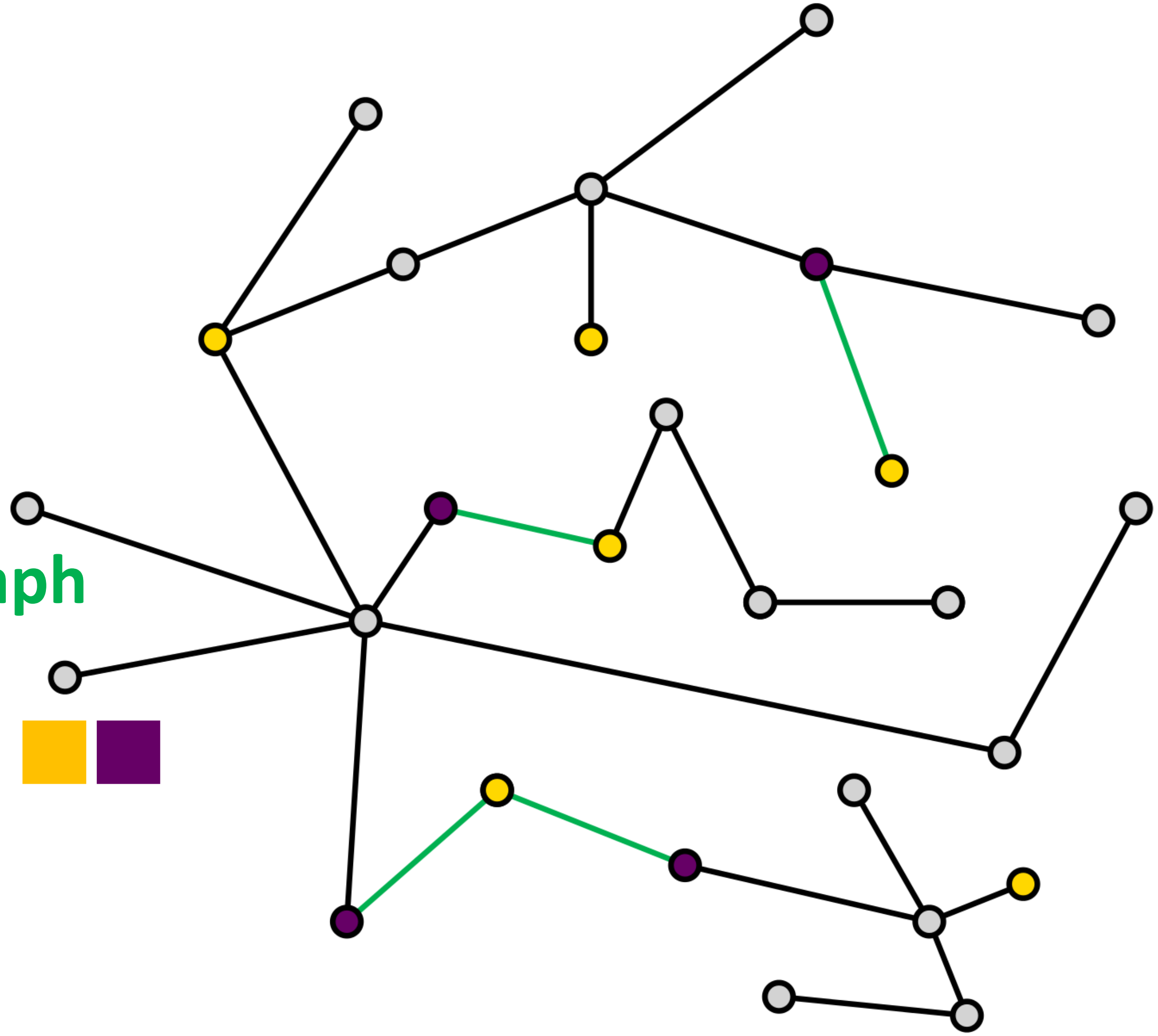
subsample **nodes** independently

**Conquer**

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨🟪

# Polynomial Degree Reduction:
# **Subsample-and-Conquer**



## **Subsample**

subsample **nodes** independently

## **Conquer**

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨 🟪
- add a color class to **MIS** 🟨

## Polynomial Degree Reduction:

# Subsample-and-Conquer

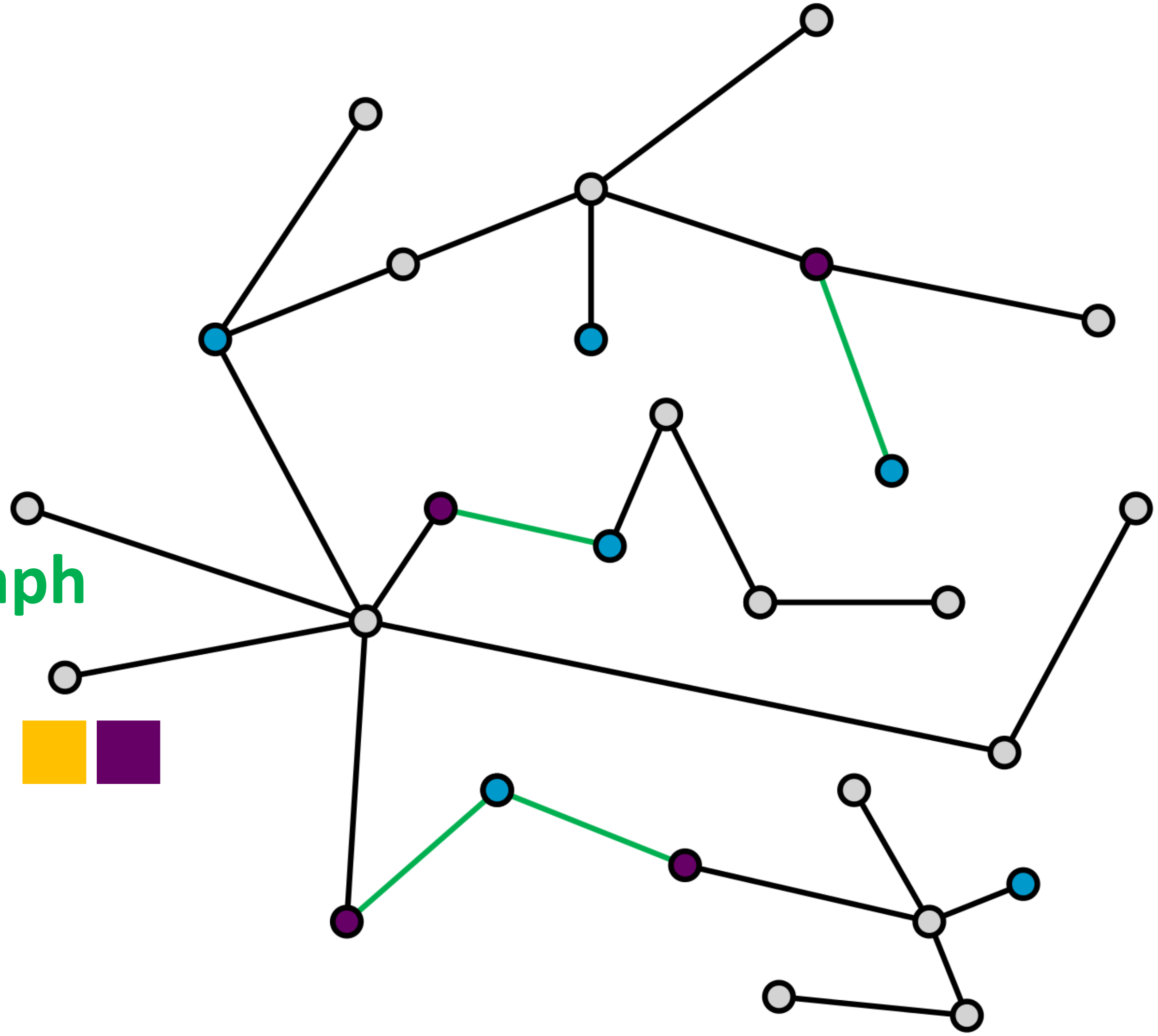
### Subsample

subsample **nodes** independently

### Conquer

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨 🟪
- add a color class to **MIS** 🟨



# Polynomial Degree Reduction:
# **Subsample-and-Conquer**

**Subsample**

subsample **nodes** independently

**Conquer**

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨 🟪
- add a color class to **MIS** 🟨

# Polynomial Degree Reduction:
# **Subsample-and-Conquer**

**Subsample**

subsample **nodes** independently

**Conquer**

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨 🟪
- add a color class to **MIS** 🟨

Non-subsampled **High-Degree Node**

# Polynomial Degree Reduction:
# Subsample-and-Conquer

## Subsample

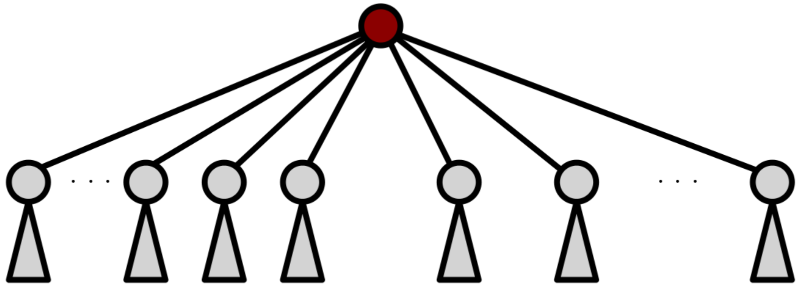
subsample **nodes** independently

## Conquer

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨 🟪
- add a color class to **MIS** 🟨

Non-subsampled **High-Degree Node**



# Polynomial Degree Reduction:
# **Subsample-and-Conquer**

## Subsample

subsample **nodes** independently

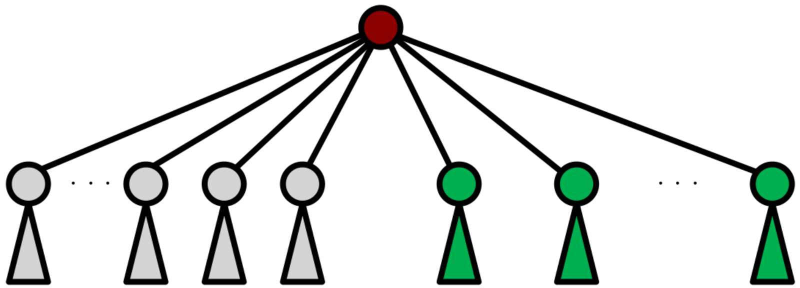
## Conquer

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨🟪
- add a color class to **MIS** 🟨

Non-subsampled **High-Degree Node**

- w.h.p. has many **subsampled neighbors**



# Polynomial Degree Reduction:
# Subsample-and-Conquer

**Subsample**

subsample **nodes** independently

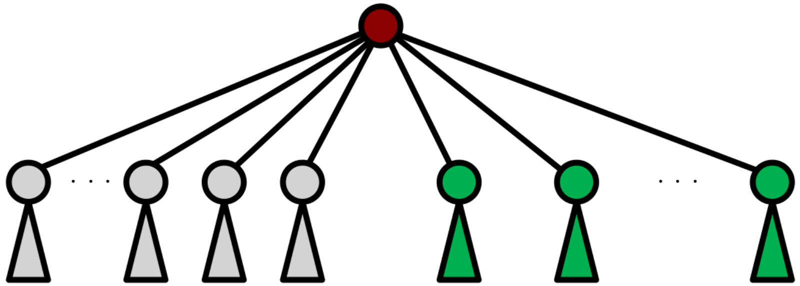
**Conquer**

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨 🟪
- add a color class to **MIS** 🟨

Non-subsampled **High-Degree Node**

- w.h.p. has many **subsampled neighbors**



**independence due to restriction to trees!**

# Polynomial Degree Reduction:
# **Subsample-and-Conquer**

**Subsample**

subsample **nodes** independently

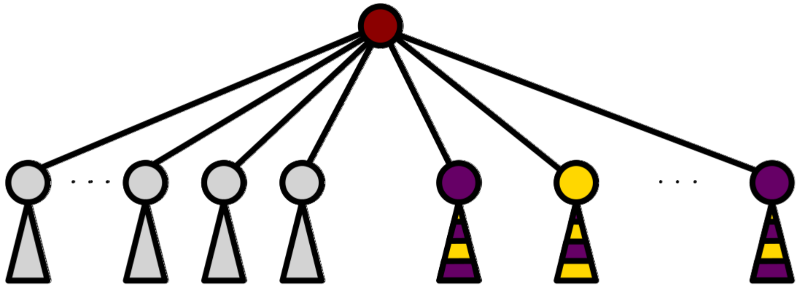
**Conquer**

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨 🟪
- add a color class to **MIS** 🟨

Non-subsampled **High-Degree Node**

- w.h.p. has many **subsampled neighbors**



**independence due to restriction to trees!**

## Polynomial Degree Reduction:

# Subsample-and-Conquer

### Subsample

subsample **nodes** independently

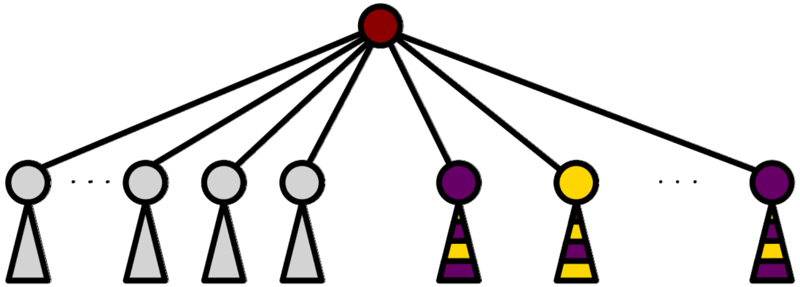
### Conquer

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨 🟪
- add a color class to **MIS** 🟨

Non-subsampled **High-Degree Node**

- w.h.p. has many **subsampled neighbors**



# Polynomial Degree Reduction:
# **Subsample-and-Conquer**

**Subsample**

subsample **nodes** independently

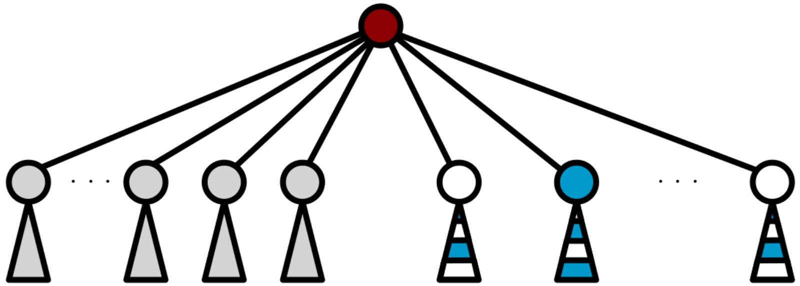
**Conquer**

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨🟪
- add a color class to **MIS** 🟨

Non-subsampled **High-Degree Node**

- w.h.p. has many **subsampled neighbors**
- thus w.h.p. has at least one **MIS neighbor**



# Polynomial Degree Reduction:
# Subsample-and-Conquer

## Subsample

subsample **nodes** independently

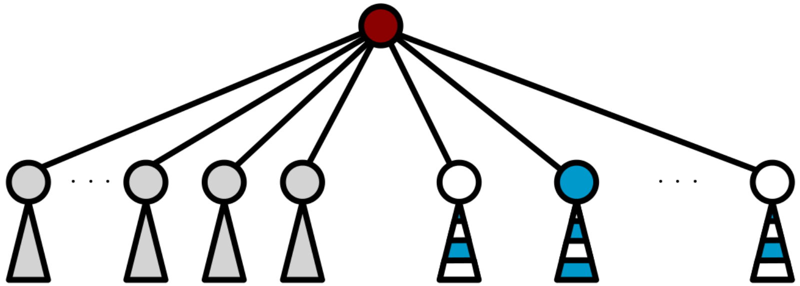
## Conquer

compute **random MIS** in **subsampled graph**

- gather connected components
- locally compute random 2-coloring 🟨 🟪
- add a color class to **MIS** 🟨

## Non-subsampled **High-Degree Node**

- w.h.p. has many **subsampled neighbors**
- thus w.h.p. has at least one **MIS neighbor**
- hence will be removed from the graph



# Algorithm Outline

## 1) Shattering

break graph into small components

i) **Degree Reduction** *Iterated Subsample-and-Conquer*

ii) **LOCAL Shattering** *Ghaffari* [SODA'16]

## 2) Post-Shattering

solve problem on remaining components

i) **Gathering of Components** *Distributed Union-Find*

ii) **Local Computation**

# Conclusion
and
Open Questions

MODEL: **Sublinear-Memory MPC**

$S = \tilde{O}(n^\delta)$ local memory

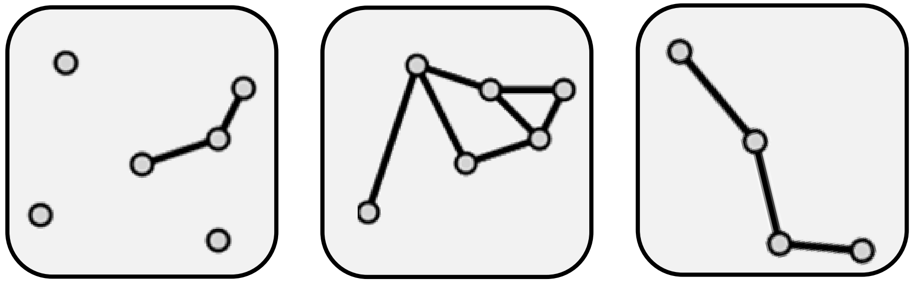
$\text{poly} \log \log n$ rounds

MODEL:

**Sublinear-Memory MPC**

$S = \tilde{O}(n^{\delta})$ local memory

$\text{poly} \log \log n$ rounds

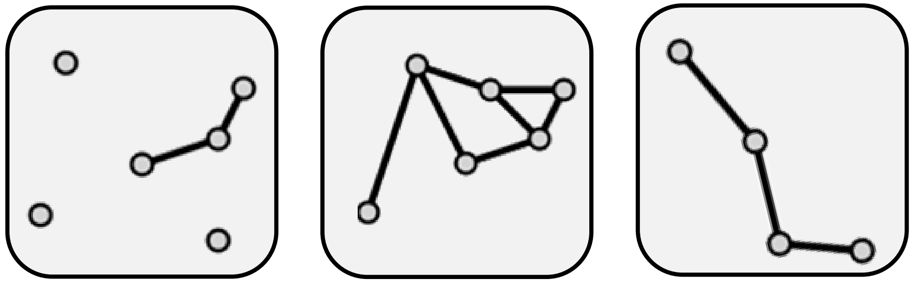


MODEL:

**Sublinear-Memory MPC**

$S = \tilde{O}(n^{\delta})$ local memory

$\text{poly} \log \log n$ rounds



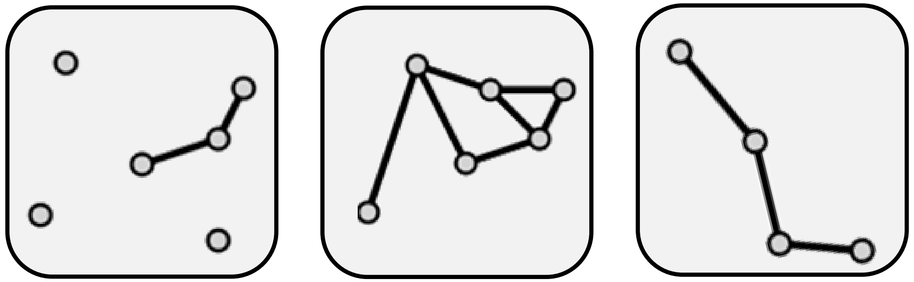
APPROACH:

LOCAL algorithms &
global communication

MODEL: **Sublinear-Memory MPC**

$S = \tilde{O}(n^{\delta})$ local memory

$\operatorname{poly} \log \log n$ rounds



APPROACH: LOCAL algorithms & global communication

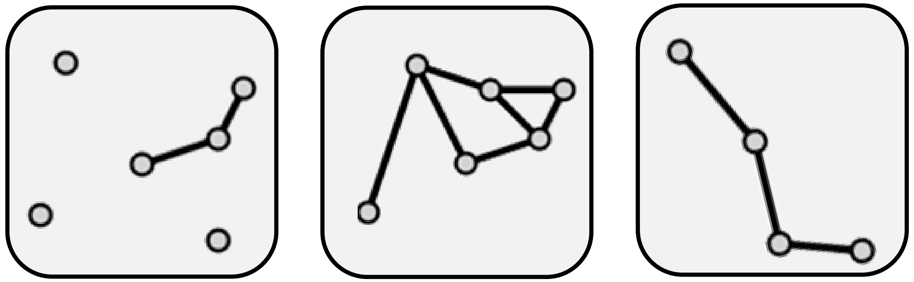
TECHNIQUE: Shattering

MODEL:

**Sublinear-Memory MPC**

$S = \tilde{O}(n^{\delta})$ local memory

$\text{poly} \log \log n$ rounds



APPROACH:

LOCAL algorithms &
global communication

TECHNIQUE:

Shattering

PROBLEM:

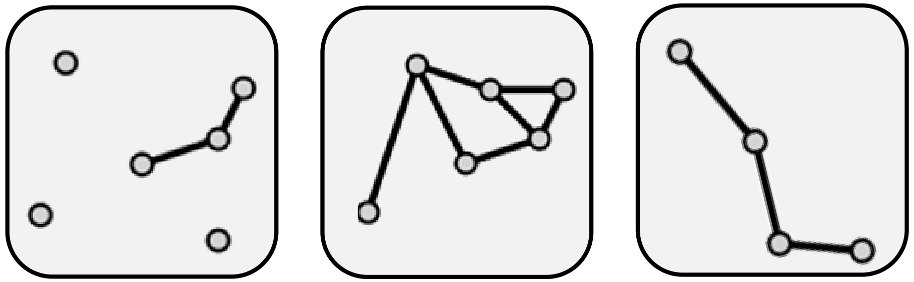
MIS
on trees

MODEL:

**Sublinear-Memory MPC**

$S = \tilde{O}(n^{\delta})$ local memory

$\text{poly} \log \log n$ rounds



APPROACH:

LOCAL algorithms &
global communication

TECHNIQUE:

Shattering

PROBLEM:

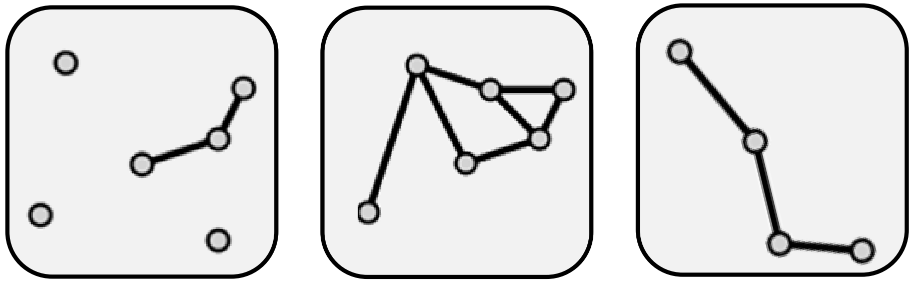
MIS
on trees

other graph problems?
more general graph families?

MODEL: **Sublinear-Memory MPC**

$S = \tilde{O}(n^{\delta})$ local memory

poly log log $n$ rounds



APPROACH: LOCAL algorithms &
global communication

TECHNIQUE: Shattering

PROBLEM: MIS
on trees

other graph problems?
more general graph families?

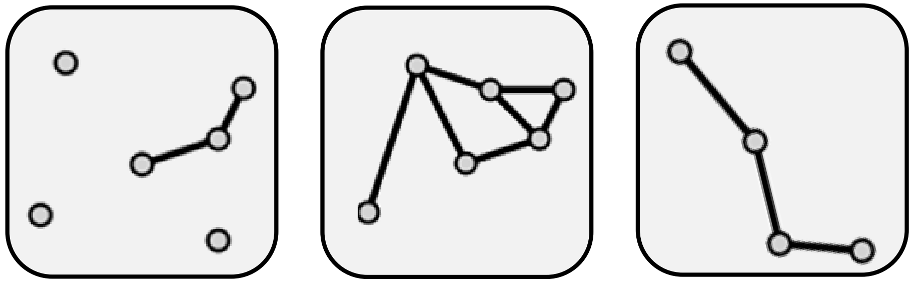
MIS & Matching for locally sparse graphs
in follow-up work [PODC’19]

MODEL:

**Sublinear-Memory MPC**

$S = \tilde{O}(n^{\delta})$ local memory

$\text{poly} \log \log n$ rounds



APPROACH:

LOCAL algorithms &
global communication

TECHNIQUE:

Shattering

other LOCAL techniques?

PROBLEM:

MIS
on trees

other graph problems?
more general graph families?

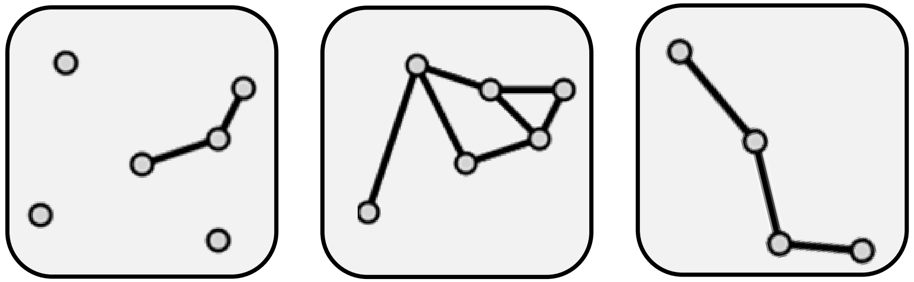
MIS & Matching for locally sparse graphs
in follow-up work [PODC'19]

MODEL:

**Sublinear-Memory MPC**

$S = \tilde{O}(n^\delta)$ local memory

poly log log $n$ rounds



APPROACH:

LOCAL algorithms &
global communication

other approaches?

TECHNIQUE:

Shattering

other LOCAL techniques?

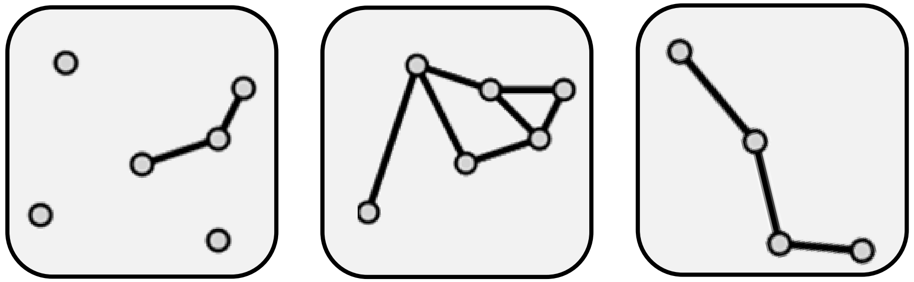
PROBLEM:

MIS
on trees

other graph problems?
more general graph families?

MIS & Matching for locally sparse graphs
in follow-up work [PODC'19]

## Thank you!



MODEL: **Sublinear-Memory MPC**

$S = \tilde{O}(n^{\delta})$ local memory

$\mathrm{poly}\log\log n$ rounds

APPROACH: LOCAL algorithms & global communication

other approaches?

TECHNIQUE: Shattering

other LOCAL techniques?

PROBLEM: MIS on trees

other graph problems?

more general graph families?

MIS & Matching for locally sparse graphs in follow-up work [PODC'19]