Near-Optimal Distributed Implementations of Dynamic Algorithms for Symmetry-Breaking Problems

Shiri Antaki
Tel Aviv University

Quanquan C. Liu
MIT

Shay Solomon
Tel Aviv University

Graph Algorithms

- Traditionally in **sequential, centralized** setting

Graph Algorithms

- Traditionally in **sequential, centralized** setting
- **Static** algorithms recompute the solution each time

Graph Algorithms

- Traditionally in **sequential, centralized** setting
- **Static** algorithms recompute the solution each time

facebook	**~ 92.5 million edges**
Twitter	**~ 2 billion edges**
Common Crawl	**~ 128 billion edges**

Example Sizes of Publicly Available Datasets

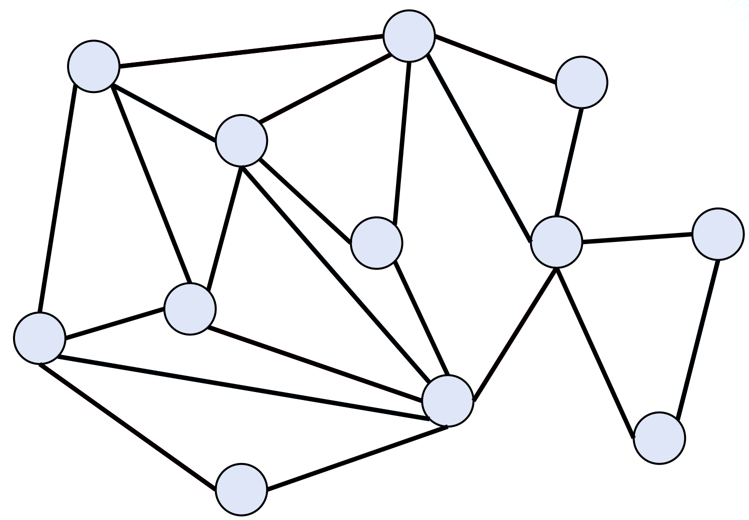
Graph Algorithms

- Traditionally in **sequential, centralized** setting

Large Graphs Too Expensive to **Rerun Even Linear Time Static Algorithms** After Updates

facebook	**~ 92.5 million edges**
Twitter	**~ 2 billion edges**
Common Crawl	**~ 128 billion edges**

Example Sizes of Publicly Available Datasets



Graphs Topology Dynamically Changing with Edge Insertions and Deletions

Dynamic Graph Algorithms

- Traditionally in **sequential, centralized** setting

Dynamic Graph Algorithms

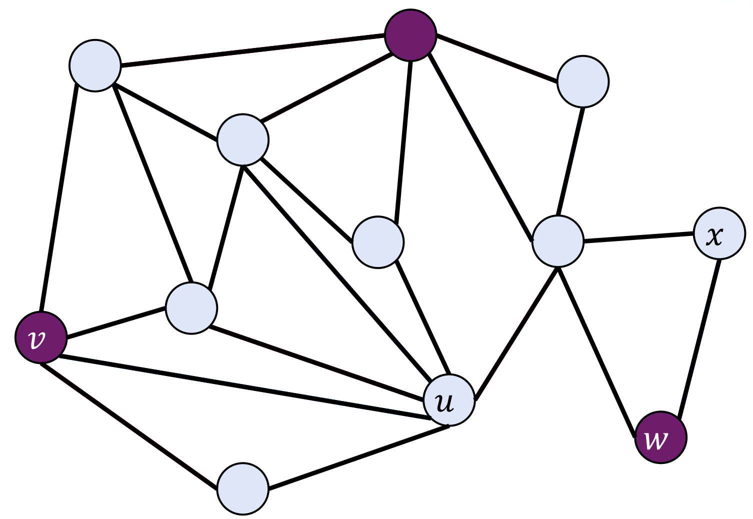
- Traditionally in **sequential, centralized** setting
- **Dynamic** algorithms recompute part of the solution after each update

Dynamic Graph Algorithms

- Traditionally in **sequential, centralized** setting
- **Dynamic** algorithms recompute part of the solution after each update
- Quality measure is **update time**, time to recompute solution

Dynamic Graph Algorithms

- Traditionally in **sequential, centralized** setting
- **Dynamic** algorithms recompute part of the solution after each update
- Quality measure is **update time**, time to recompute solution



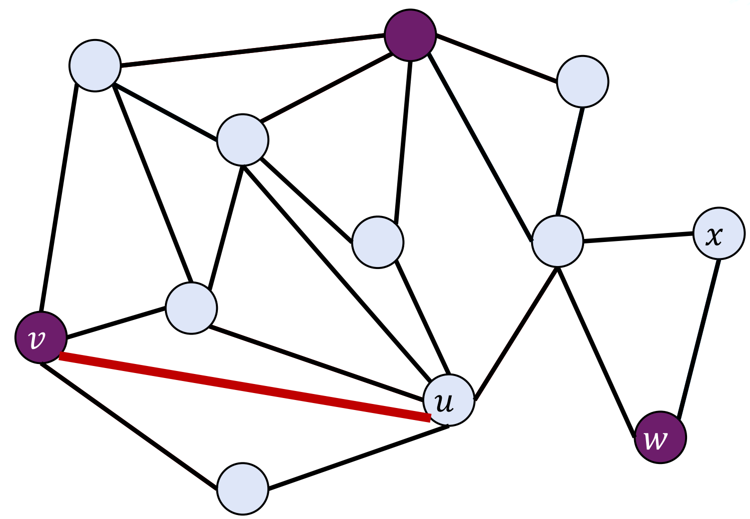
Goal (**Sequential, Centralized** Setting):
Minimize Update Time

Example Maximal Independent Set Updated After Edge Insertions/Deletions

Dynamic Graph Algorithms

- Traditionally in **sequential, centralized** setting
- **Dynamic** algorithms recompute part of the solution after each update
- Quality measure is **update time**, time to recompute solution

Look at Direct Neighbors to Update MIS



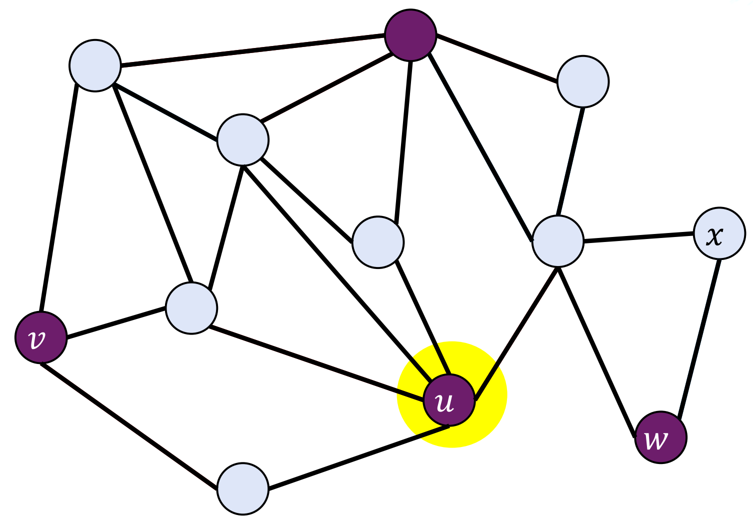
Goal (Sequential, Centralized Setting): Minimize Update Time

Example Maximal Independent Set Updated After Edge Insertions/Deletions

Dynamic Graph Algorithms

- Traditionally in **sequential, centralized** setting
- **Dynamic** algorithms recompute part of the solution after each update
- Quality measure is **update time**, time to recompute solution

Look at Direct Neighbors to Update MIS



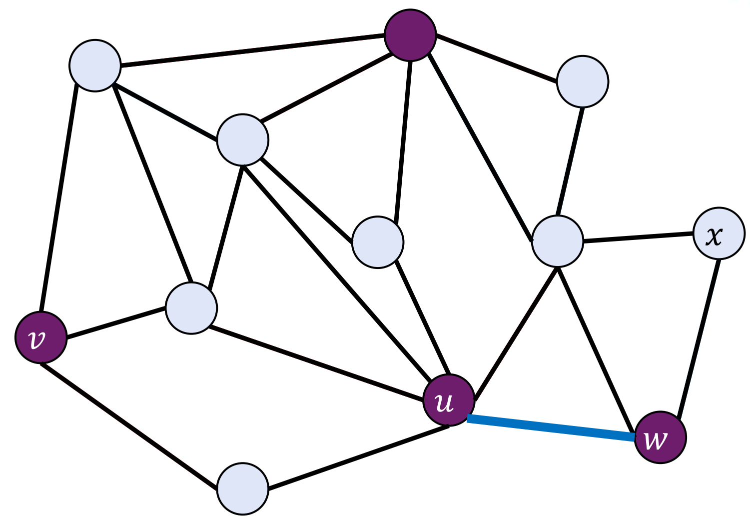
Goal (**Sequential, Centralized** Setting): **Minimize Update Time**

Example Maximal Independent Set Updated After Edge Insertions/Deletions

Dynamic Graph Algorithms

- Traditionally in **sequential, centralized** setting
- **Dynamic** algorithms recompute part of the solution after each update
- Quality measure is **update time**, time to recompute solution

Look at Direct Neighbors to Update MIS

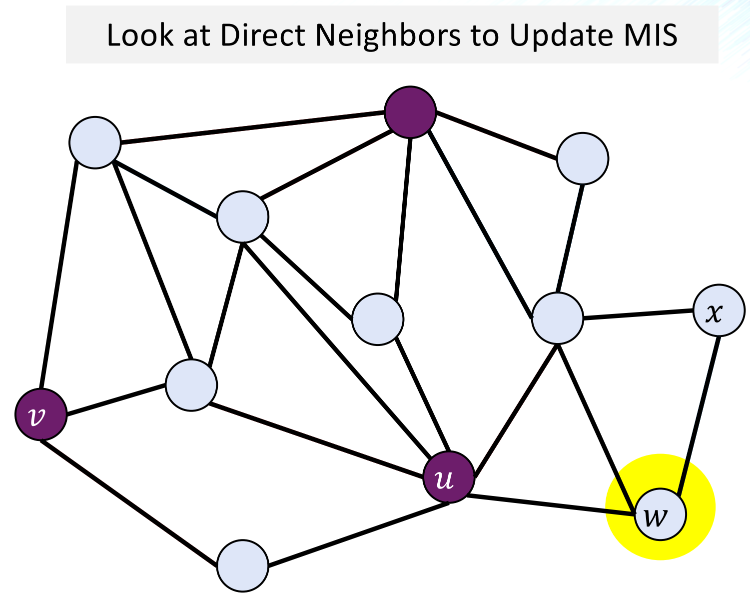


Goal (**Sequential, Centralized** Setting):
Minimize Update Time

Example Maximal Independent Set Updated After Edge Insertions/Deletions

Dynamic Graph Algorithms

- Traditionally in **sequential, centralized** setting
- **Dynamic** algorithms recompute part of the solution after each update
- Quality measure is **update time**, time to recompute solution



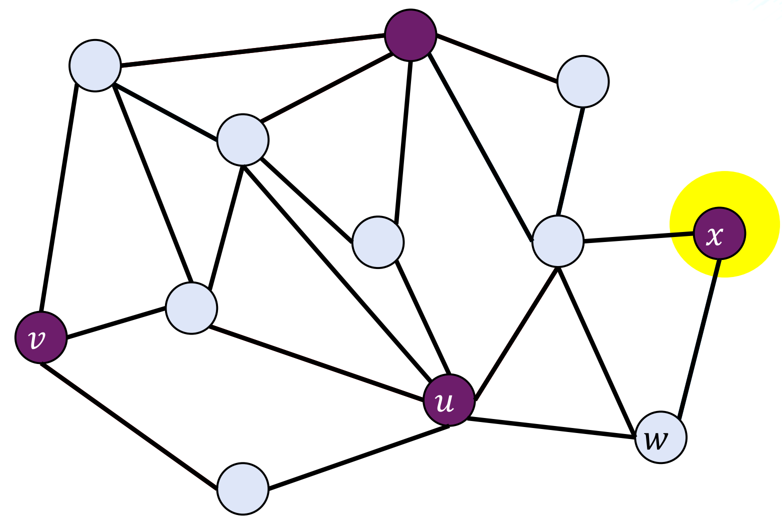
Goal (**Sequential, Centralized** Setting):
Minimize Update Time

Example Maximal Independent Set Updated After Edge Insertions/Deletions

Dynamic Graph Algorithms

- Traditionally in **sequential, centralized** setting
- **Dynamic** algorithms recompute part of the solution after each update
- Quality measure is **update time**, time to recompute solution

Look at Direct Neighbors to Update MIS



Goal (**Sequential, Centralized** Setting): **Minimize Update Time**

Example Maximal Independent Set Updated After Edge Insertions/Deletions

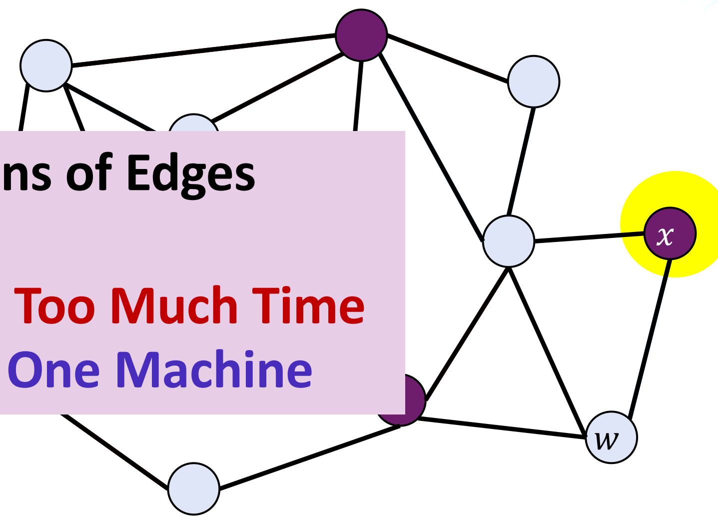
Dynamic Graph Algorithms

- Traditionally in **sequential, centralized** setting
- **Dynamic** algo
  part of the sol
  update
- Quality measu
  time to recompute solution

Billions or Even Trillions of Edges

Graph Too Large to Fit and Too Much Time Process Sequentially on One Machine

Look at Direct Neighbors to Update MIS

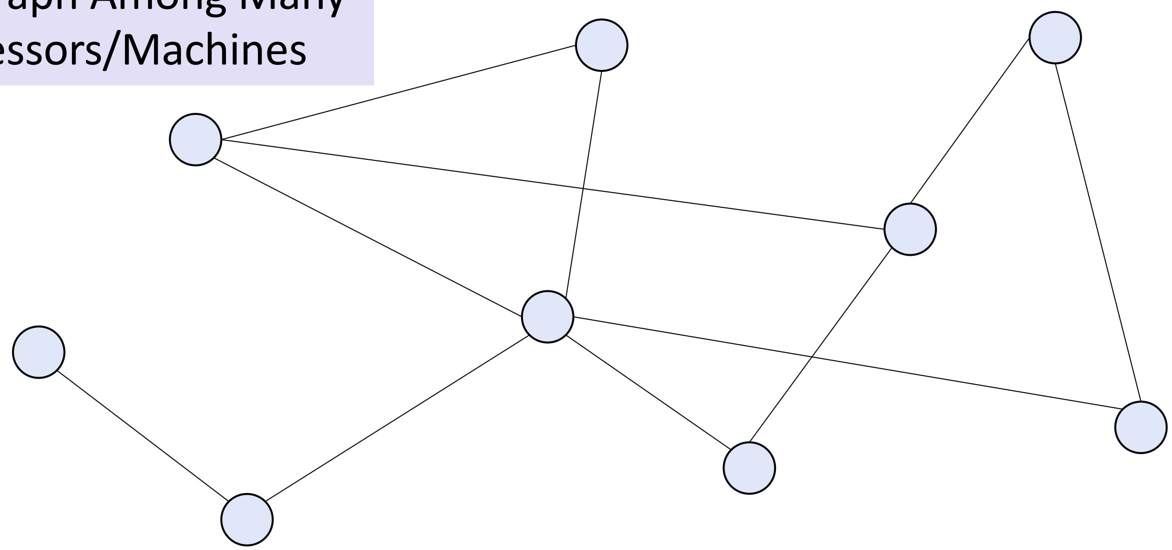


Goal (Sequential, Centralized Setting): **Minimize Update Time**

Example Maximal Independent Set Updated After Edge Insertions/Deletions

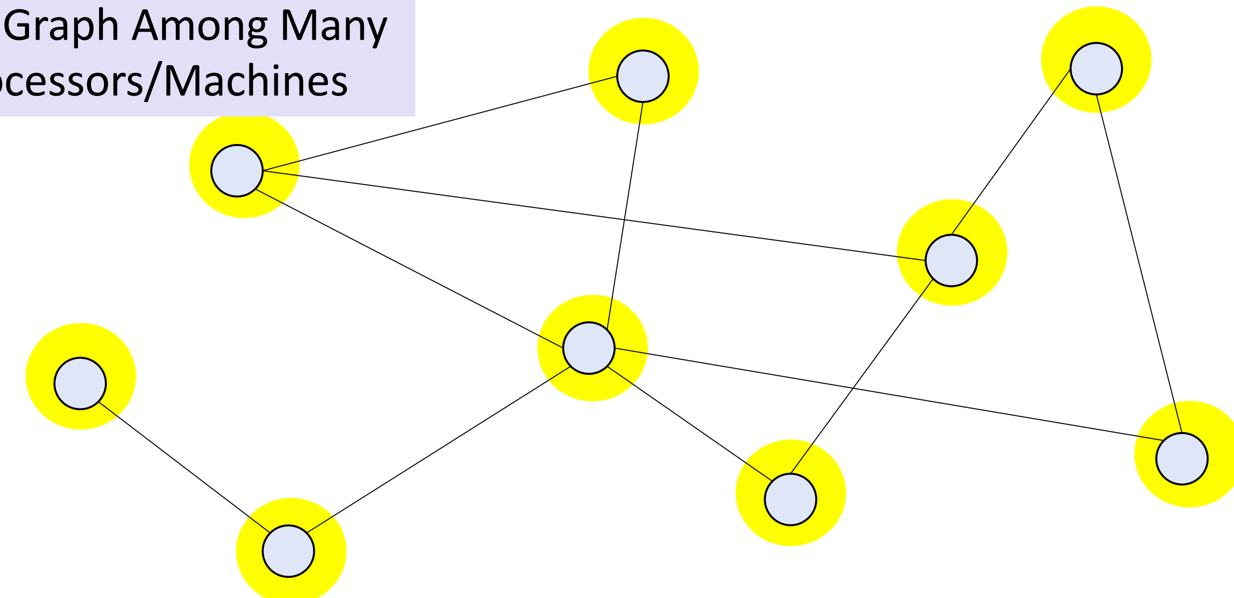
Distributed Algorithms and Networks

Split the Large Graph Among Many Different Processors/Machines



Distributed Algorithms and Networks

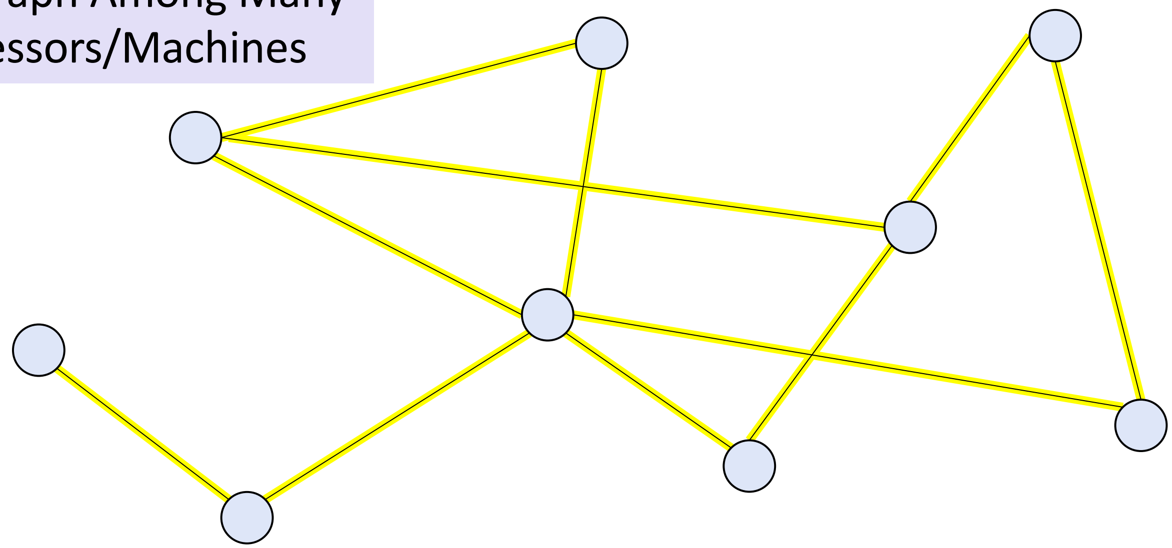
Split the Large Graph Among Many Different Processors/Machines



Each Node is a Processor/Machine

Distributed Algorithms and Networks

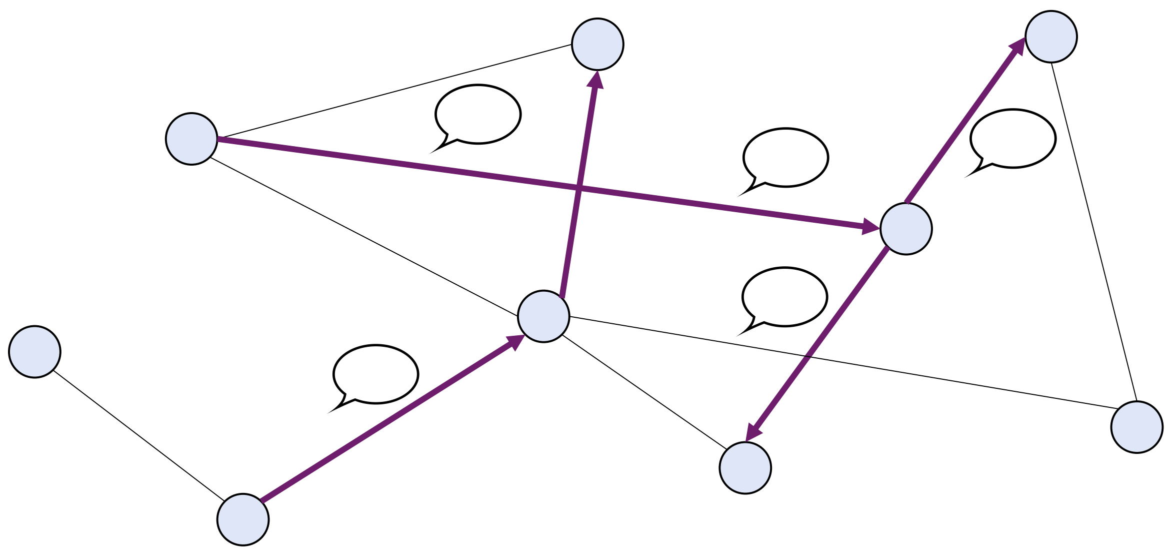
Split the Large Graph Among Many Different Processors/Machines



Each Node is a Processor/Machine

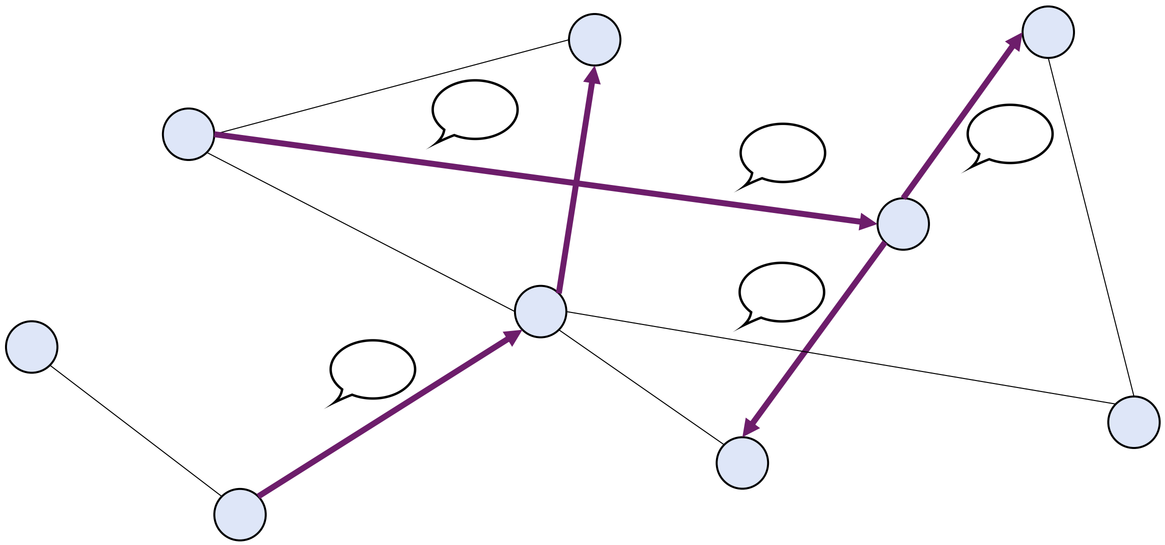
Edges are Communication Links

Distributed Algorithms and Networks



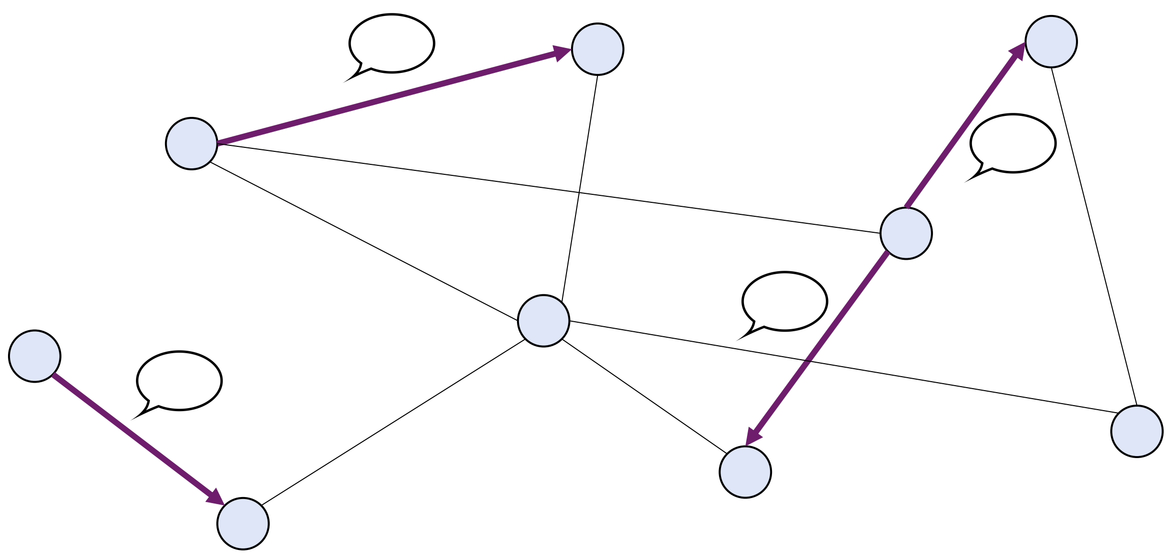
Nodes Send **Messages** to Other Nodes Via Edges
Nodes Can Choose to Send to Some/All Neighbors

Distributed Algorithms and Networks



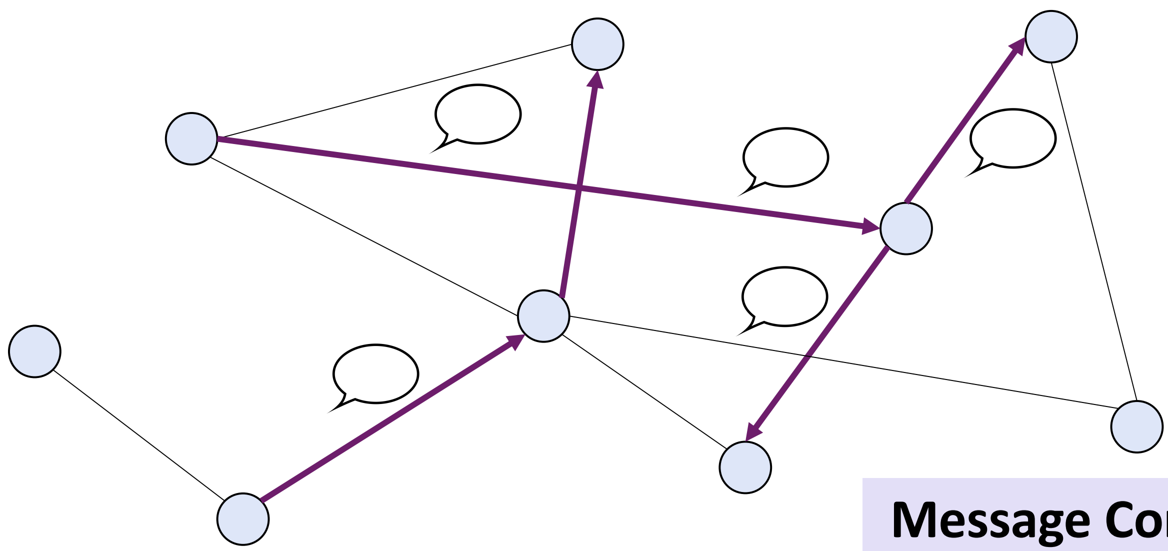
Nodes Use Multiple **Rounds** of Communication to Send Messages

Distributed Algorithms and Networks



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

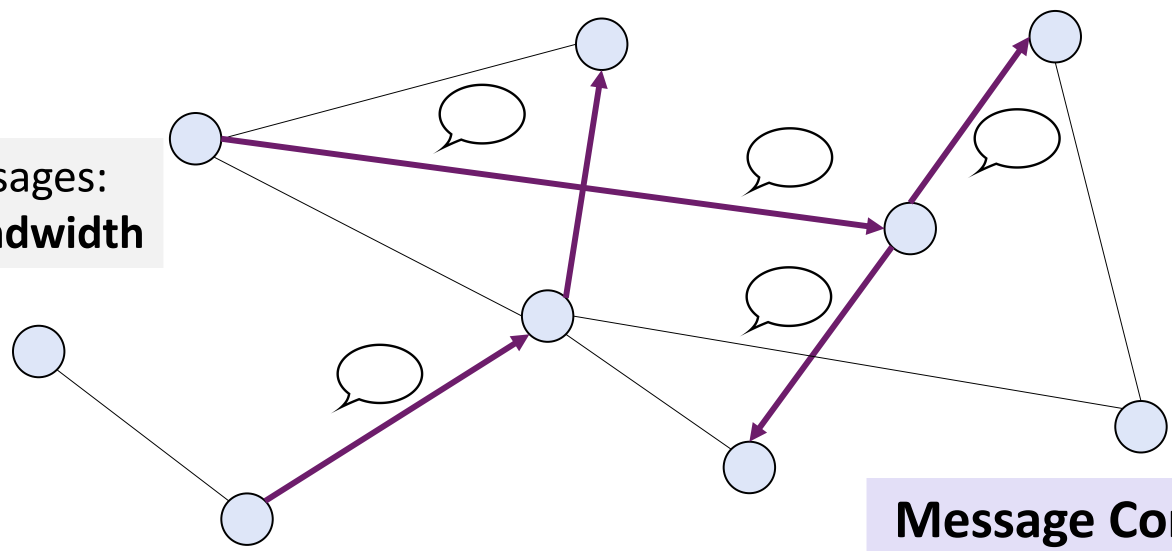
Distributed Algorithms and Networks



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

Message Complexity
Number of Messages
Sent in Total

Distributed Algorithms and Networks



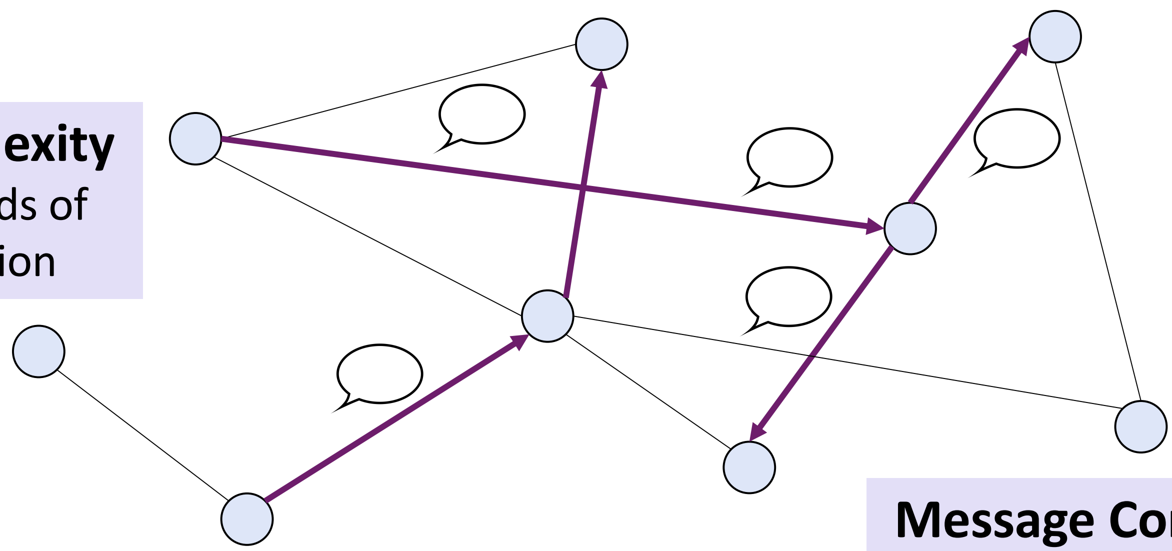
Too many messages:
overwhelms bandwidth

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

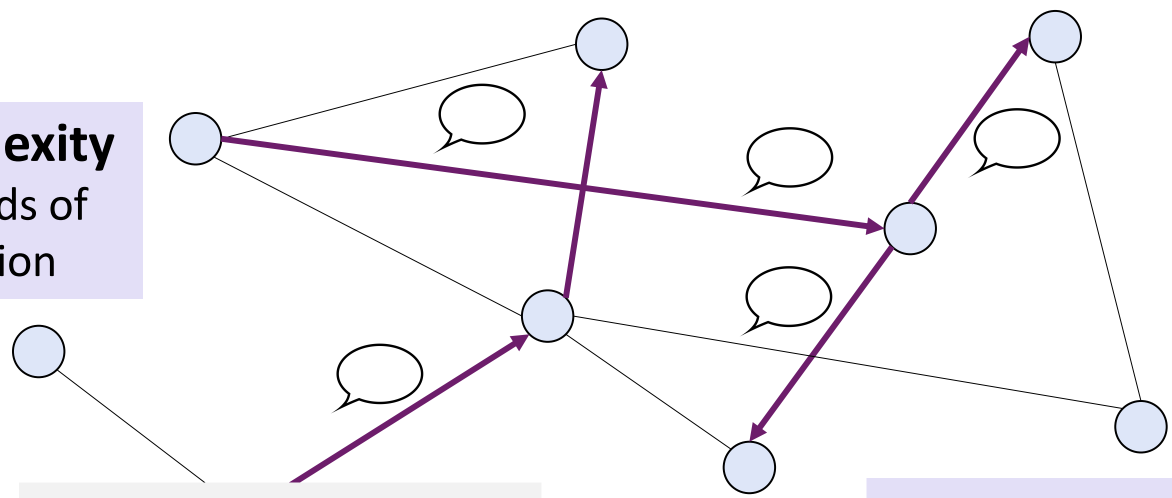


CONGEST Model:
Messages have $O(\log n)$ size

Message Complexity
Number of Messages Sent in Total

Distributed Algorithms and Networks

Round Complexity
Multiple Rounds of
Communication

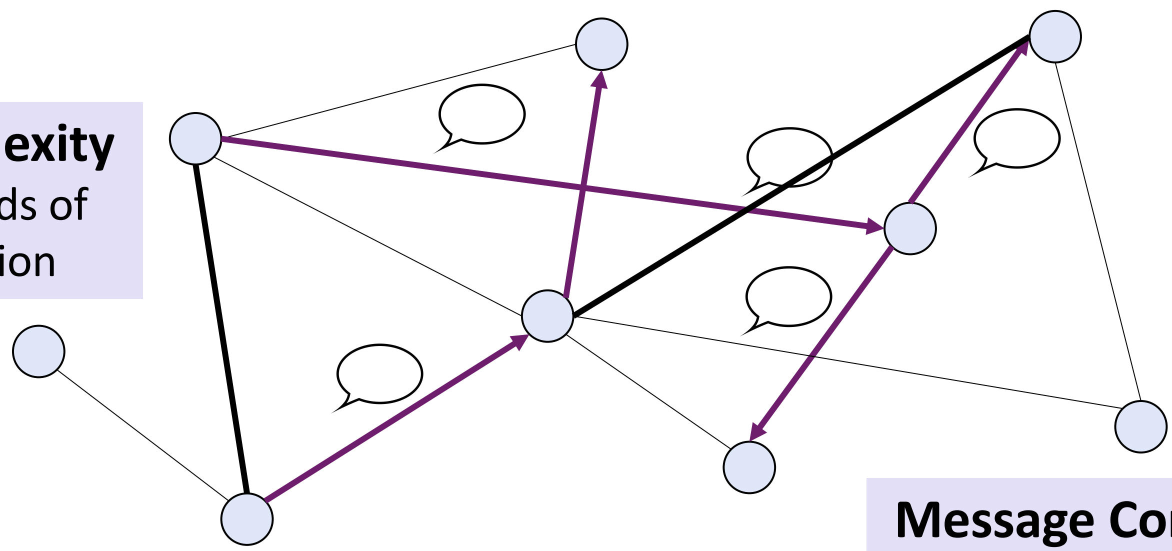


Too many rounds:
takes too long and sends
too many messages

Message Complexity
Number of Messages
Sent in Total

Dynamic Distributed Networks

Round Complexity
Multiple Rounds of
Communication

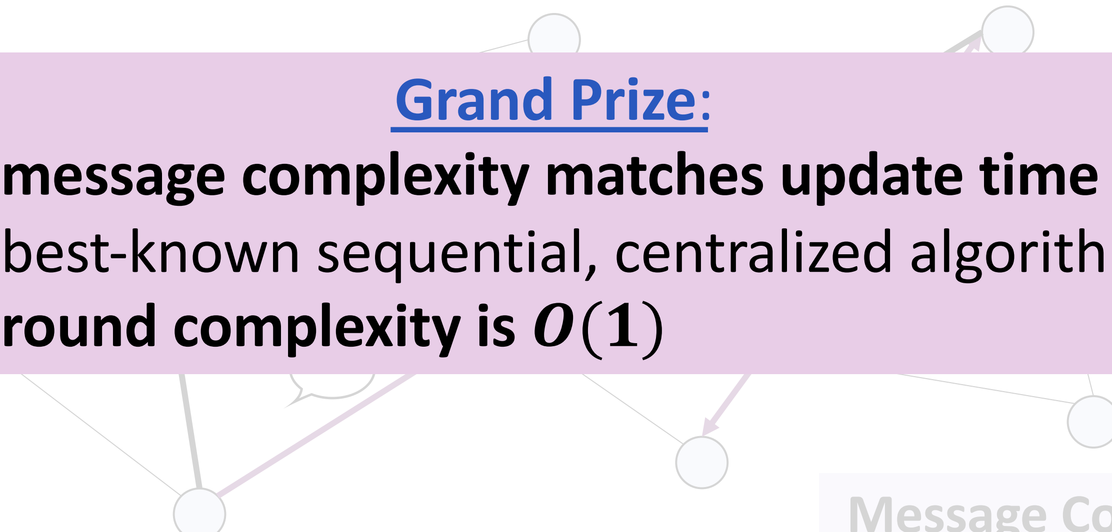


Edges Can be Added and Deleted from the Network
Changes Network Communication Topology

Message Complexity
Number of Messages
Sent in Total

Dynamic Distributed Networks

Round Com
Multiple Rou
Communic



Grand Prize:

- **message complexity matches update time** of best-known sequential, centralized algorithm
- **round complexity is $O(1)$**

Edges Can be Added and Deleted from the Network
Changes Network Communication Topology

Message Complexity
Number of Messages
Sent in Total

Dynamic Distributed Networks

Round Com
Multiple Rou
Communic

Grand Prize:

- **message complexity matches update time** of best-known sequential, centralized algorithm
- **round complexity is $O(1)$**

Robust against adaptive adversaries

Message Complexity
Number of Messages
Sent in Total

Edges Can be Added and Deleted from the Network
Changes Network Communication Topology

Previous Work: Dynamic Distributed Algorithms

- Most previous work focused on minimizing **round complexity** [BEG18, BKM19, CDKPS20, CHK16, KW13, LPR09, PPS16]

Previous Work: Dynamic Distributed Algorithms

- Most previous work focused on minimizing **round complexity** [BEG18, BKM19, CDKPS20, CHK16, KW13, LPR09, PPS16]
  - Dynamically changing distributed networks

Previous Work: Dynamic Distributed Algorithms

- Most previous work focused on minimizing **round complexity** [BEG18, BKM19, CDKPS20, CHK16, KW13, LPR09, PPS16]
  - Dynamically changing distributed networks
  - Very recently, [BKM19] and [CDKPS20] also studied **simultaneously** handling **many concurrent updates**

Previous Work: Dynamic Distributed Algorithms

- Most previous work focused on minimizing **round complexity** [BEG18, BKM19, CDKPS20, CHK16, KW13, LPR09, PPS16]
  - Dynamically changing distributed networks
  - Very recently, [BKM19] and [CDKPS20] also studied **simultaneously** handling **many concurrent updates**
- Previous algorithms send messages to **all neighbors (broadcast)**

Previous Work: Dynamic Distributed Algorithms

- Most previous work focused on minimizing **round complexity** [BEG18, BKM19, CDKPS20, CHK16, KW13, LPR09, PPS16]
  - Dynamically changing distributed networks
  - Very recently, [BKM19] and [CDKPS20] also studied **simultaneously** handling **many concurrent updates**
- Previous algorithms send messages to **all neighbors (broadcast)**
  - Results in $\Omega(\Delta)$ messages for $\Delta =$ max degree

Previous Work: Dynamic Distributed Algorithms

- Most previous work focused on minimizing **round complexity** [BEG18, BKM19, CDKPS20, CHK16, KW13, LPR09, PPS16]
  - Dynamically changing distributed networks
  - Very recently, [BKM19] and [CDKPS20] also studied **simultaneously** handling **many concurrent updates**
- Previous algorithms send messages to **all neighbors (broadcast)**
  - Results in $\Omega(\Delta)$ messages for $\Delta =$ max degree
  - Can be as large as $\Omega(m)$ for sparse graphs

Previous Work: Dynamic Distributed Algorithms

Very few previous works consider number of messages sent

- Most previou … **plexity** [BEG18, BK … PS16]
  - Dynamically changing distributed networks
  - Very recently, [BKM19] and [CDKPS20] also studied **simultaneously** handling **many concurrent updates**
- Previous algorithms send messages to **all neighbors (broadcast)**
  - Results in $\Omega(\Delta)$ messages for $\Delta =$ max degree
  - Can be as large as $\Omega(m)$ for sparse graphs

Previous Work: Dynamic Distributed Algorithms

Very few previous works consider number of messages sent

Many practical real-world situations require few messages:

• Most previou … plexity

[BEG18, BK … PS16]

• Dynamica

• Very rece

simultan … es

• Previous alg … **broadcast)**

• Results ir

• Can be a

Previous Work: Dynamic Distributed Algorithms

Very few previous works consider number of messages sent

Many practical real-world situations require few messages:

- **Systems with poor wireless connections**

Previous Work: Dynamic Distributed Algorithms

Very few previous works consider number of messages sent

Many practical real-world situations require few messages:

- **Systems with poor wireless connections**
- **Over-saturated network with many independent agents**

Previous Work: Dynamic Distributed Algorithms

• Most previou plexity
[BEG18, BK PS16]

• Dynamica

• Very rece
simultan es

• Previous alg **broadcast)**

• Results i

• Can be a

Very few previous works consider number of messages sent

Many practical real-world situations require few messages:

- **Systems with poor wireless connections**
- **Over-saturated network with many independent agents**
- **Mobile data network in poorly connected area**

Message Complexity for Dynamic Distributed Algorithms

- Send messages to specific **subsets of neighbors, not all (multicast)**

Message Complexity for Dynamic Distributed Algorithms

- Send messages to specific **subsets of neighbors, not all (multicast)**
- [AOSS18, PS16, KS18] studied message complexity for certain problems

Message Complexity for Dynamic Distributed Algorithms

- Send messages to specific **subsets of neighbors, not all (multicast)**
- [AOSS18, PS16, KS18] studied message complexity for certain problems
  - [AOSS18] gave $O(m^{3/4})$ amortized messages and $O(1)$ round algorithm for MIS

Message Complexity for Dynamic Distributed Algorithms

- Send messages to specific **subsets of neighbors, not all (multicast)**
- [AOSS18, PS16, KS18] studied message complexity for certain problems
  - [AOSS18] gave $O\left(m^{3/4}\right)$ amortized messages and $O(1)$ round algorithm for MIS
  - [PS16] gave an $O(\alpha/\epsilon)$ worst-case messages and $O(1)$ round algorithm for $(1+\epsilon)$-maximum cardinality matching

Message Complexity for Dynamic Distributed Algorithms

- Send messages to specific **subsets of neighbors, not all (multicast)**
- [AOSS18, PS16, KS18] studied message complexity for certain problems
    - [AOSS18] gave $O(m^{3/4})$ amortized messages and $O(1)$ round algorithm for MIS
    - [PS16] gave an $O(\alpha/\epsilon)$ worst-case messages and $O(1)$ round algorithm for $(1+\epsilon)$-maximum cardinality matching
        - α is a graph property, arboricity

Message Complexity for Dynamic Distributed Algorithms

- Send messages to specific **subsets of neighbors, not all (multicast)**
- [AOSS18, PS16, KS18] studied message complexity for certain problems
  - [AOSS18] gave $O\left(m^{3/4}\right)$ amortized messages and $O(1)$ round algorithm for MIS
  - [PS16] gave an $O(\alpha/\epsilon)$ worst-case messages and $O(1)$ round algorithm for $(1+\epsilon)$-maximum cardinality matching
    - α is a graph property, arboricity
    - α could be as large as $\sqrt{m}$

Message Complexity for Dynamic Distributed Algorithms

- Send messages to specific **subsets of neighbors, not all (multicast)**
- [AOSS18, PS16, KS18] studied message complexity for certain problems
  - [AOSS18] gave $O(m^{3/4})$ amortized messages and $O(1)$ round algorithm for MIS
  - [PS16] gave an $O(\alpha/\epsilon)$ worst-case messages and $O(1)$ round algorithm for $(1+\epsilon)$-maximum cardinality matching
    - α is a graph property, arboricity
    - α could be as large as $\sqrt{m}$
  - [KS18] gave an $O(\log n)$ amortized messages low out-degree orientation algorithm in $O(1)$ amortized rounds for constant α

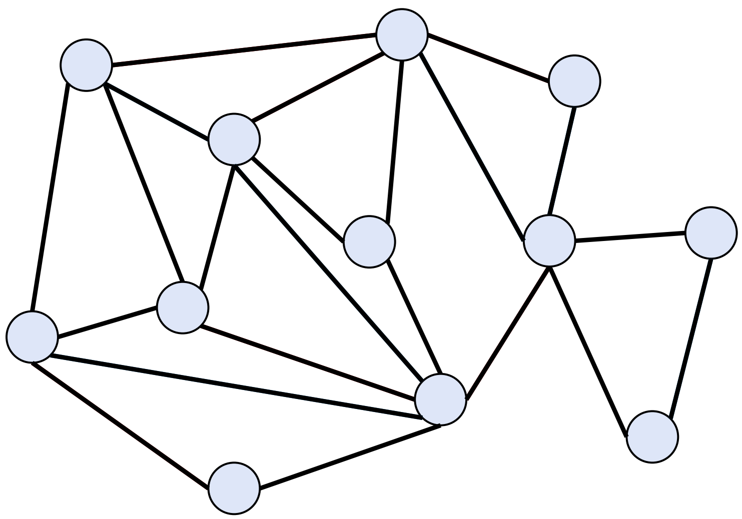
Message Complexity for Dynamic Distributed Algorithms

- Send messages to specific **subsets of neighbors, not all (multicast)**
- [AOSS18, PS16, KS18] studied message complexity for certain problems
  - [AO … algo…
  - [PS … orithm for …
    - α is a graph property, arboricity
    - α could be as large as $\sqrt{m}$
  - [KS18] gave an $O(\log n)$ amortized messages low out-degree orientation algorithm in $O(1)$ amortized rounds for constant α

Grand Prize:

- **message complexity matches update time** of best-known sequential, centralized algorithm
- **round complexity is $O(1)$**

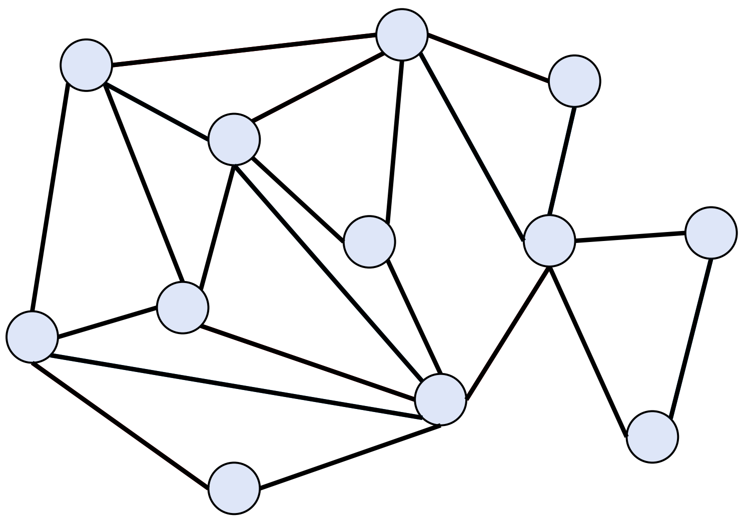
Challenges with Adapting Centralized Algorithms



Determining Number of Edges After Insertions and Deletions

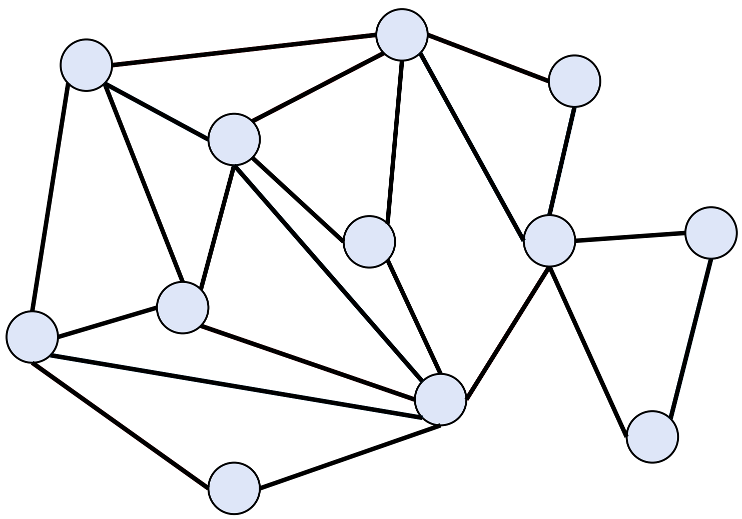
$m = 16$

Challenges with Adapting Centralized Algorithms



Many Centralized Algorithms use **Global Restarts** (Large Part of Graph Restarts)

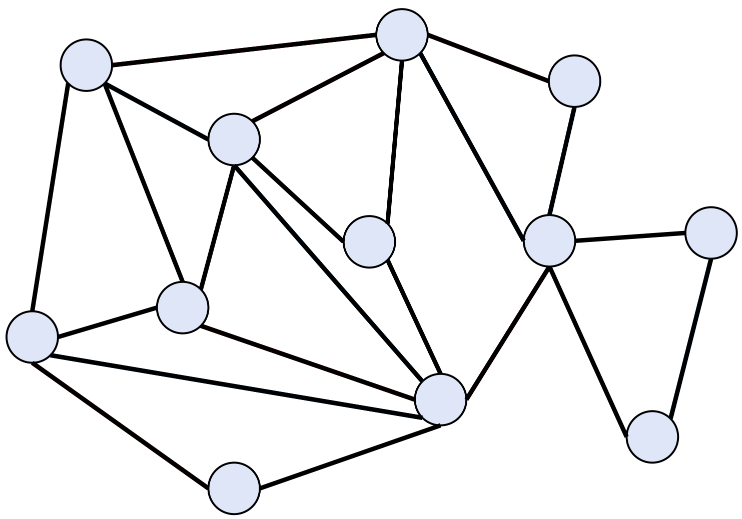
Challenges with Adapting Centralized Algorithms



Many Centralized Algorithms use **Global Restarts** (Large Part of Graph Restarts)

Global Restarts Must Be **Propagated to a Large Portion of Network**

Challenges with Adapting Centralized Algorithms

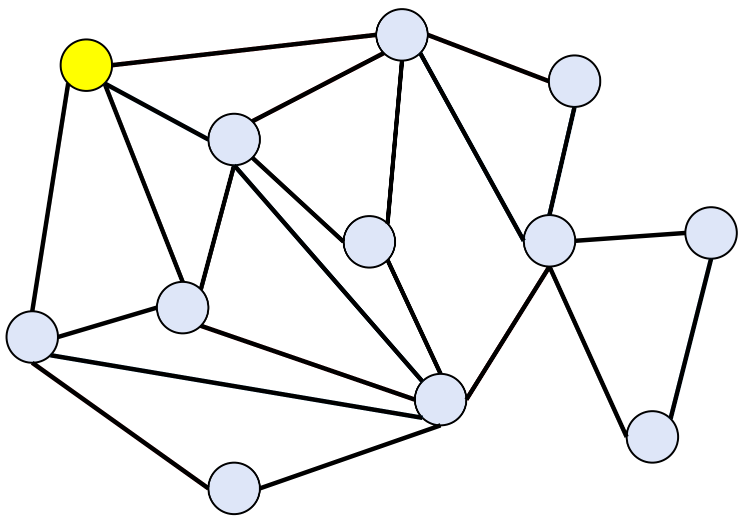


Many Centralized Algorithms use **Global Restarts** (Large Part of Graph Restarts)

Global Restarts Must Be **Propagated to a Large Portion of Network**

High Round and **High Message Complexity**

Challenges with Adapting Centralized Algorithms

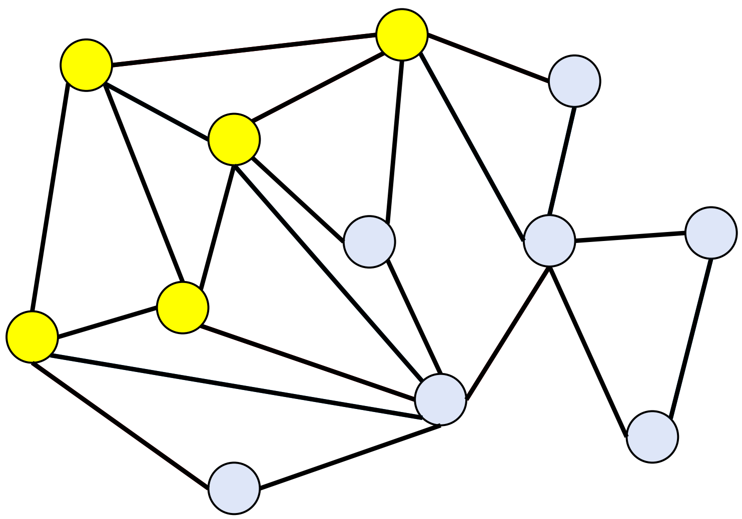


Many Centralized Algorithms use **Global Restarts** (Large Part of Graph Restarts)

Global Restarts Must Be **Propagated to a Large Portion of Network**

High Round and **High Message Complexity**

Challenges with Adapting Centralized Algorithms

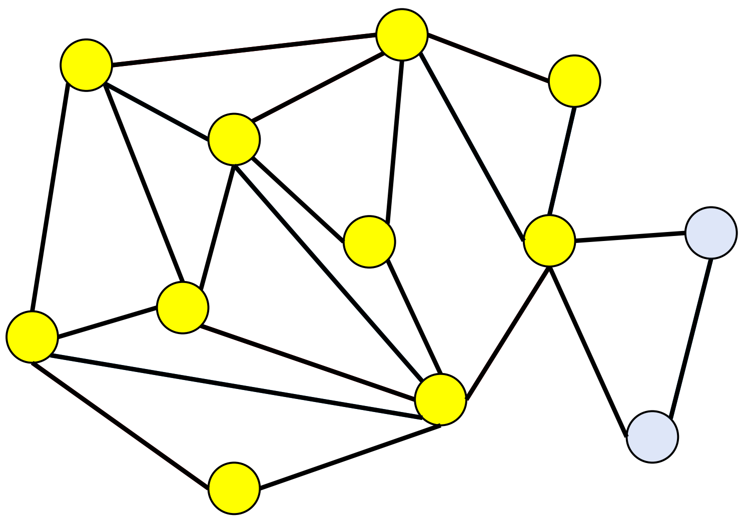


Many Centralized Algorithms use **Global Restarts** (Large Part of Graph Restarts)

Global Restarts Must Be **Propagated to a Large Portion of Network**

High Round and **High Message Complexity**

Challenges with Adapting Centralized Algorithms

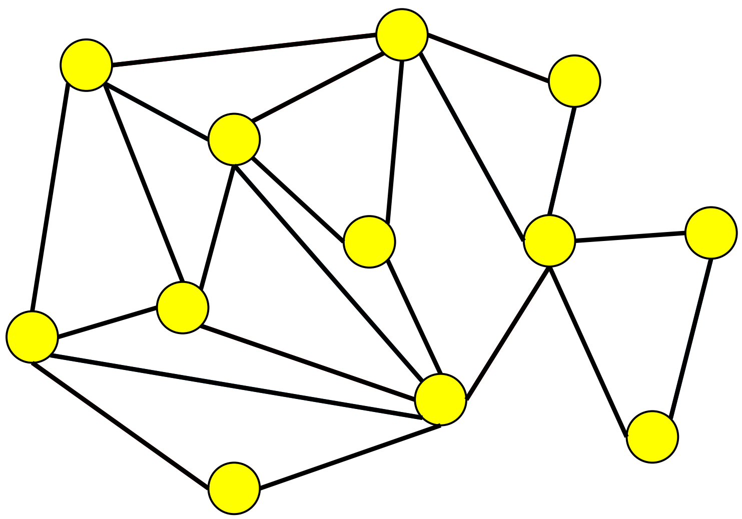


Many Centralized Algorithms use **Global Restarts** (Large Part of Graph Restarts)

Global Restarts Must Be **Propagated to a Large Portion of Network**

High Round and **High Message Complexity**

Challenges with Adapting Centralized Algorithms

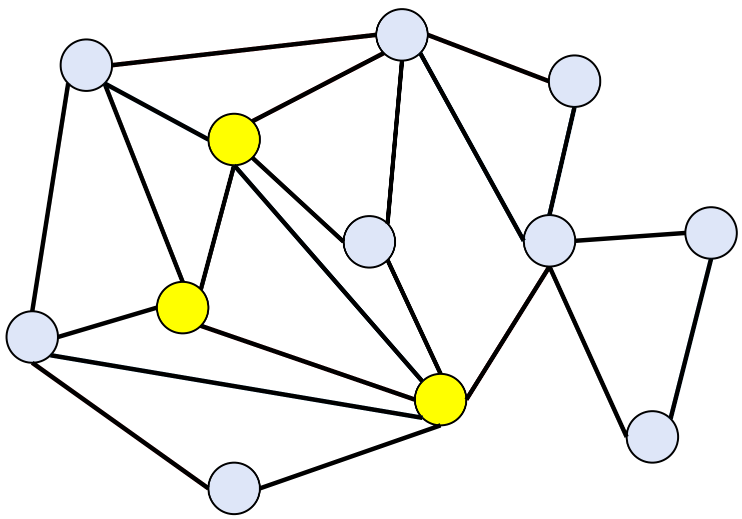


Many Centralized Algorithms use **Global Restarts** (Large Part of Graph Restarts)

Global Restarts Must Be **Propagated to a Large Portion of Network**

High Round and **High Message Complexity**

Challenges with Adapting Centralized Algorithms



Solution: Consider **Partial Local Neighborhood**

Local Neighborhood: reduces **round** complexity

Partial Neighborhood: reduces **message** complexity

Classic Symmetry-Breaking Problems

- $(\Delta + 1)$-Coloring
- Maximal Matching and 3/2-Approximate Maximum Matching
- Maximal Independent Set

Classic Symmetry-Breaking Problems

- $(\Delta + 1)$-Coloring
- Maximal Matching and 3/2-Approximate Maximum Matching
- Maximal Independent Set

Grand Prize:

- **message complexity matches update time** of best-known sequential, centralized algorithm
- **round complexity is $\boldsymbol{O(1)}$**

Our Deterministic Algorithm Results

$(\Delta + 1)$-Vertex Coloring

- $O(\sqrt{m})$ messages and $O(1)$ rounds, both worst-case
- Dynamic Edge Orientation Technique
- Matches best-known sequential, centralized algorithm of [KNNP20]

Maximal Matching

- $O(\sqrt{m})$ messages and $O(1)$ rounds, both worst-case
- Dynamic Edge Orientation Technique
- Matches best-known sequential, centralized algorithm of [NS13]

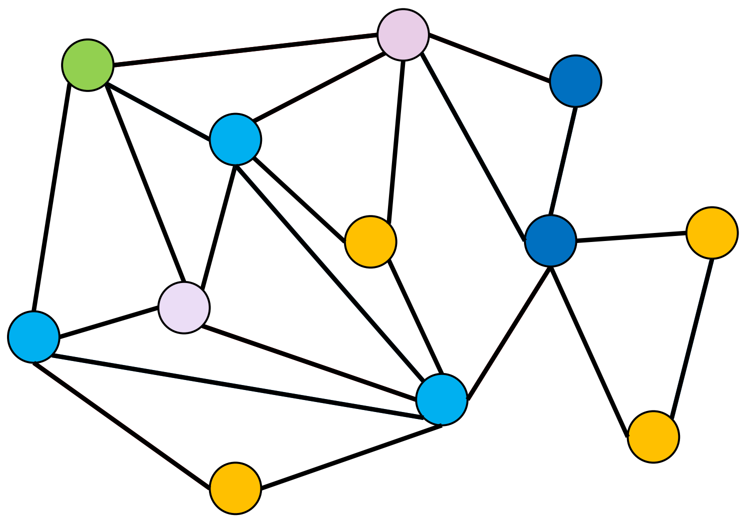
(3/2)-Maximum Matching

- $O(\sqrt{m})$ amortized messages and $O(\log \Delta)$ rounds, worst case
- High-Degree/Low-Degree Partitioning Using Surrogates
- Matches best-known sequential, centralized algorithm of [NS13]

Maximal Independent Set

- $O\left(m^{2/3} \log^2 n\right)$ messages and $O(\log^2 n)$ rounds, amortized
- High-Degree/Low-Degree Partitioning Using 6-Hop Neighborhood
- Use a small-diameter *static* algorithm to obtain MIS in high-degree and *dynamic* MIS for low-degree
- Matches best-known sequential, centralized [GK21] up to $\tilde{O}(1)$ factor

$(\Delta + 1)$-Coloring



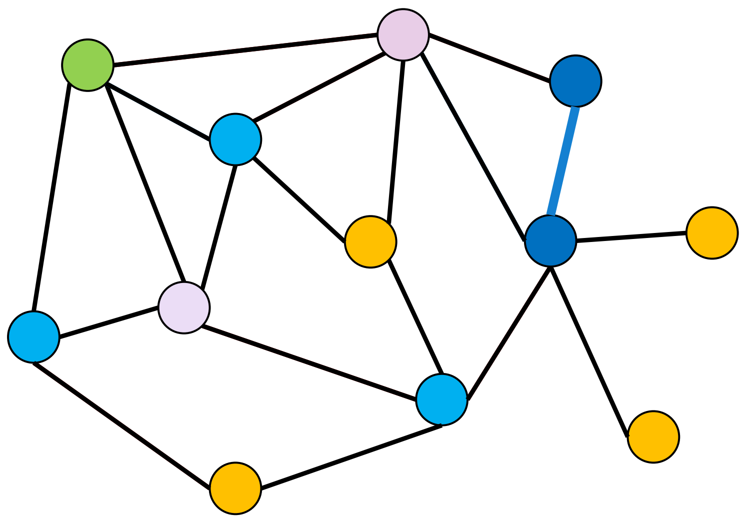
No two adjacent nodes have same color

Uses at most $(\Delta + 1)$ colors

Δ: maximum degree

$(\Delta + 1)$-Coloring

Edge Insertions May Result in Conflicts



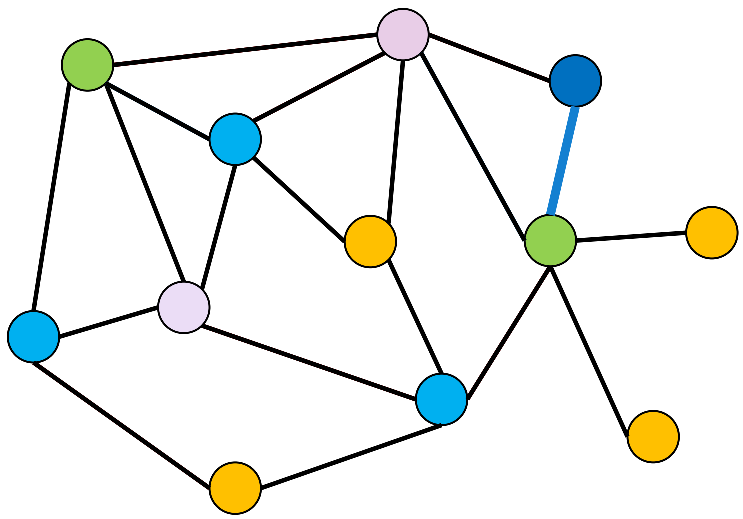
No two adjacent nodes have same color

Uses at most $(\Delta + 1)$ colors

Δ: maximum degree

$(\Delta + 1)$-Coloring

Edge Insertions May Result in Conflicts



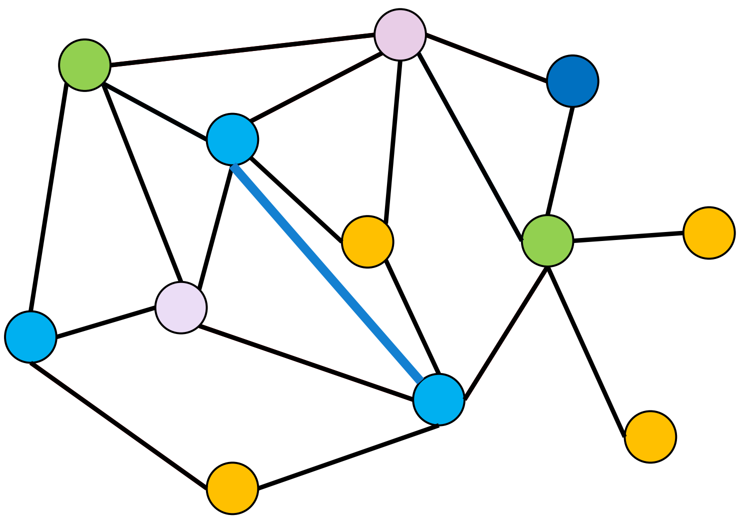
No two adjacent nodes have same color

Uses at most $(\Delta + 1)$ colors

Δ: maximum degree

$(\Delta + 1)$-Coloring

Edge Insertions May Result in Conflicts



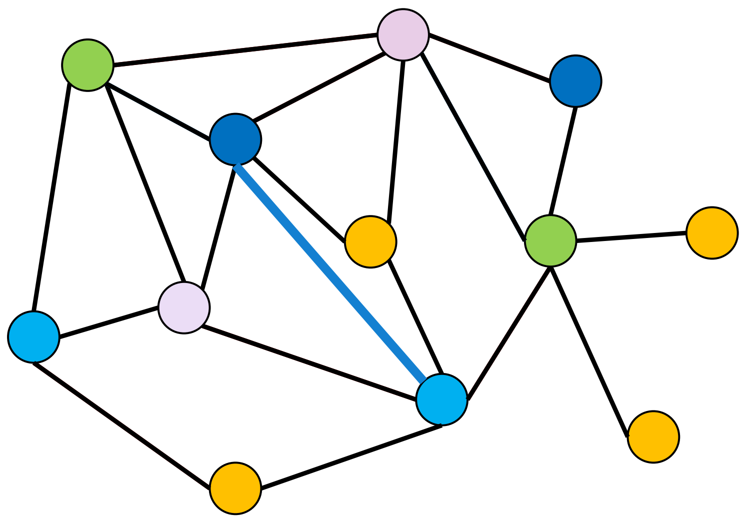
No two adjacent nodes have same color

Uses at most $(\Delta + 1)$ colors

Δ: maximum degree

$(\Delta + 1)$-Coloring

Edge Insertions May Result in Conflicts

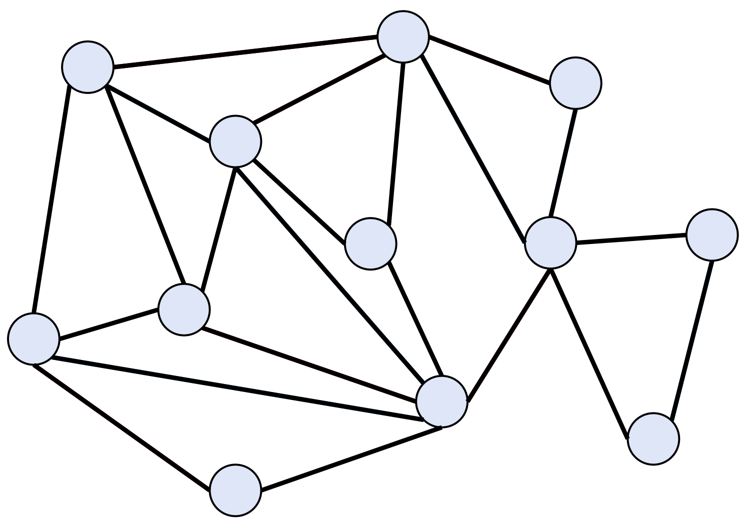


No two adjacent nodes have same color

Uses at most $(\Delta + 1)$ colors

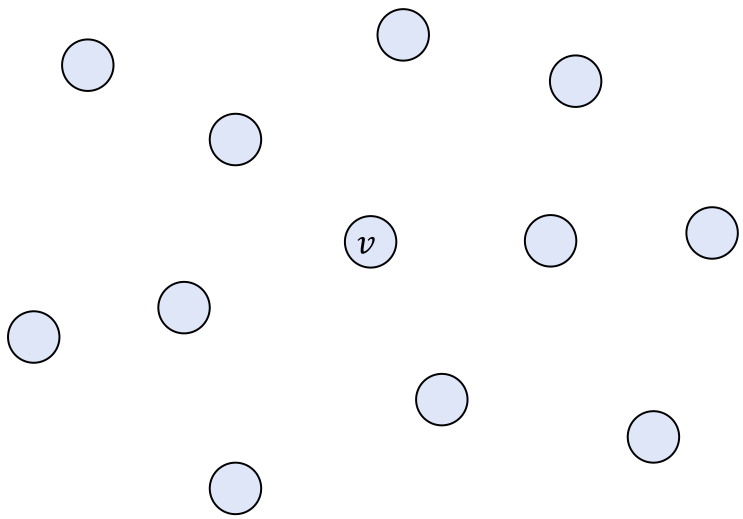
Δ: maximum degree

Dynamic Edge Orientation Technique



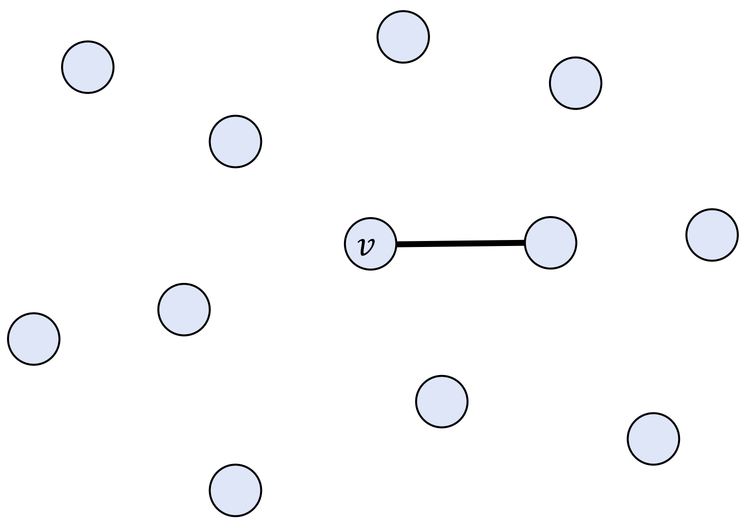
- Each vertex maintains counter $p_v \leftarrow \deg(v)$

Dynamic Edge Orientation Technique



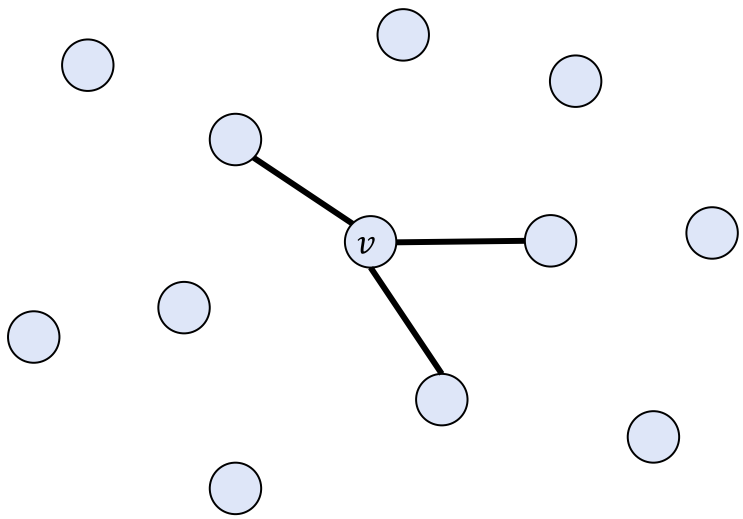
- Each vertex maintains counter $p_v \leftarrow 1$
- After degree of a vertex falls outside $\left[\frac{p_v}{2}, 2p_v\right]$, ask neighbors for degree

Dynamic Edge Orientation Technique



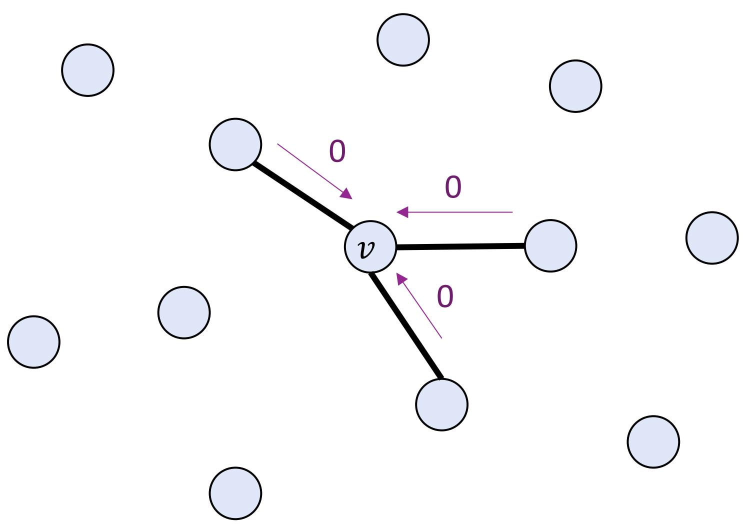
- Each vertex maintains counter $p_v \leftarrow 1$
- After degree of a vertex falls outside $\left[\frac{p_v}{2}, 2p_v\right]$, ask neighbors for degree

Dynamic Edge Orientation Technique



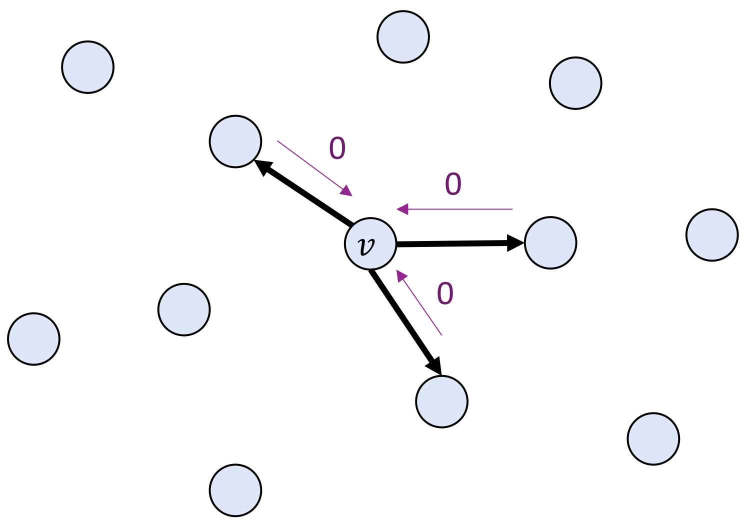
- Each vertex maintains counter $p_v \leftarrow 1$
- After degree of a vertex falls outside $\left[\frac{p_v}{2}, 2p_v\right]$, ask neighbors for degree

Dynamic Edge Orientation Technique



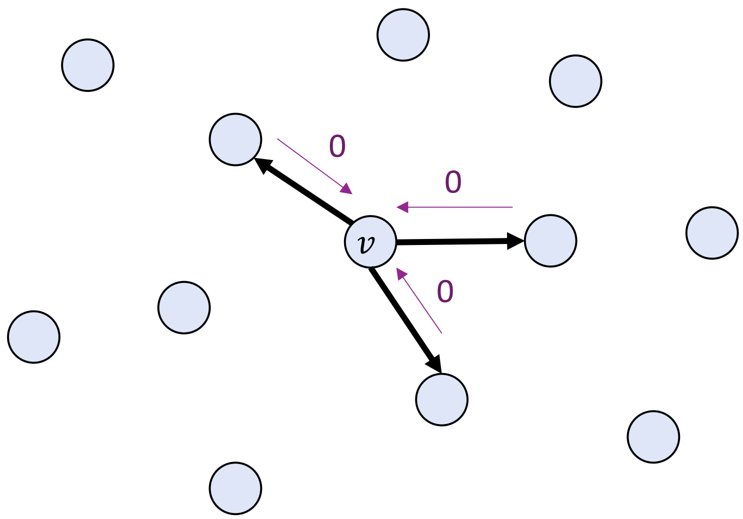
- Each vertex maintains counter $p_v \leftarrow 1$
- After degree of a vertex falls outside $\left[\frac{p_v}{2}, 2p_v\right]$, ask neighbors for degree

Dynamic Edge Orientation Technique



- Each vertex maintains counter $p_v \leftarrow 1$
- After degree of a vertex falls outside $\left[\frac{p_v}{2}, 2p_v\right]$, ask neighbors for degree
- Orient edges towards smaller degree endpoint

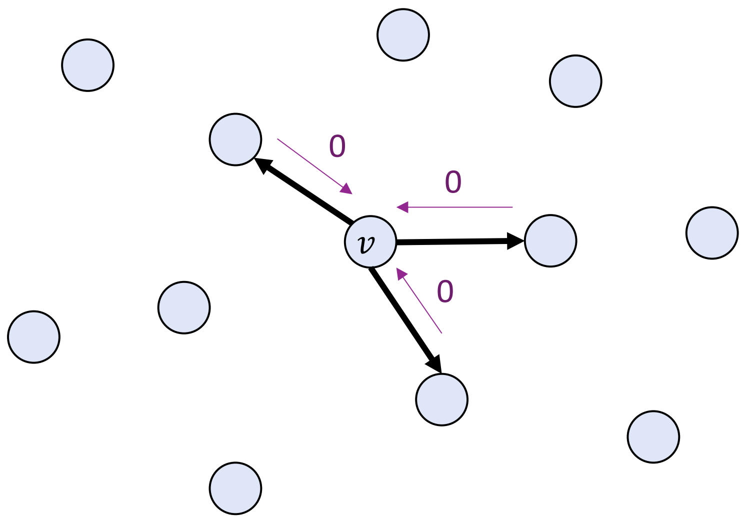
Dynamic Edge Orientation Technique



- Each vertex maintains counter $p_v \leftarrow 1$
- After degree of a vertex falls outside $\left[\frac{p_v}{2}, 2p_v\right]$, ask neighbors for degree
- Orient edges towards smaller degree endpoint
- Reset counter $p_v \leftarrow \deg(v)$
- Repeat under future updates

Dynamic Edge Orientation Technique

Invariant: at most $4\sqrt{m}$ **outgoing**

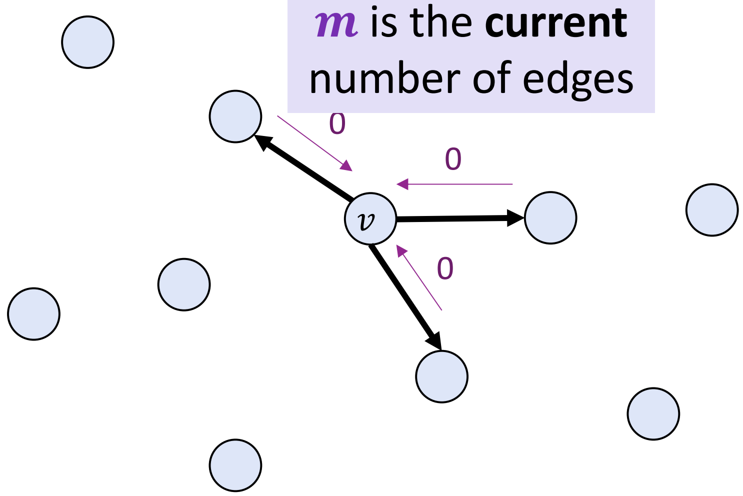


- Each vertex maintains counter $p_v \leftarrow 1$
- After degree of a vertex falls outside $\left[\frac{p_v}{2}, 2p_v\right]$, ask neighbors for degree
- Orient edges towards smaller degree endpoint
- Reset counter $p_v \leftarrow \deg(v)$
- Repeat under future updates

Dynamic Edge Orientation Technique

Invariant: at most $\mathbf{4\sqrt{m}}$ **outgoing**

m is the **current** number of edges

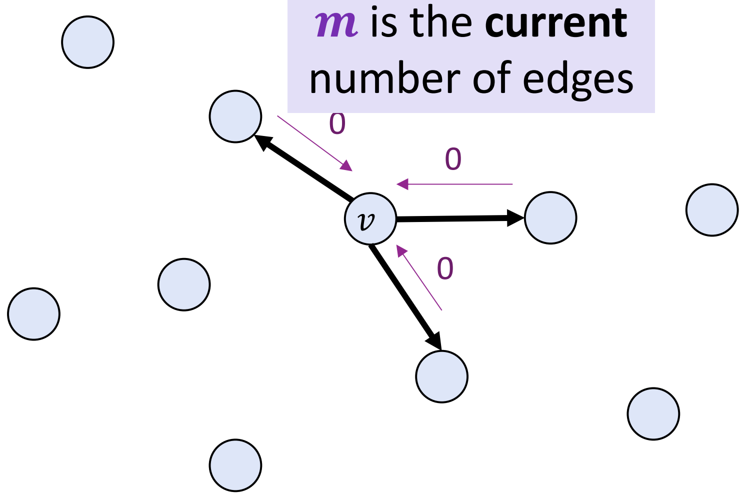


- Each vertex maintains counter $p_v \leftarrow 1$
- After degree of a vertex falls outside $\left[\frac{p_v}{2}, 2p_v\right]$, ask neighbors for degree
- Orient edges towards smaller degree endpoint
- Reset counter $p_v \leftarrow \deg(v)$
- Repeat under future updates

Dynamic Edge Orientation Technique

Invariant: at most $4\sqrt{m}$ **outgoing**

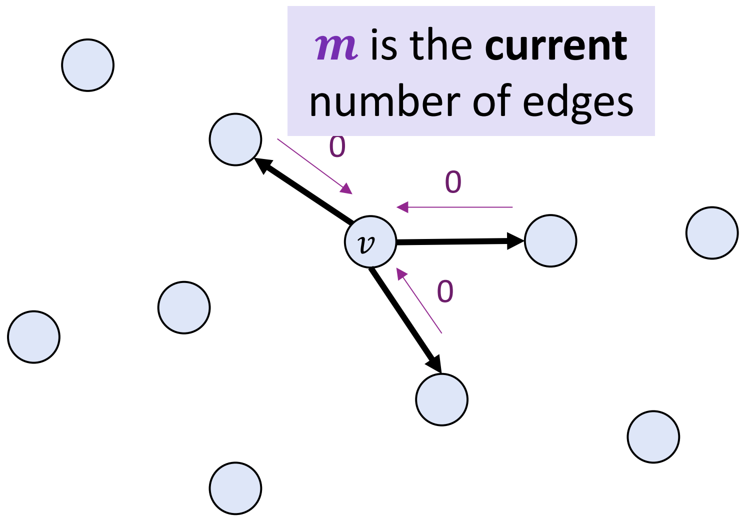
m is the **current** number of edges



- Round complexity: $O(1)$ worst-case
- Message complexity: $O(1)$ amortized

Dynamic Edge Orientation Technique

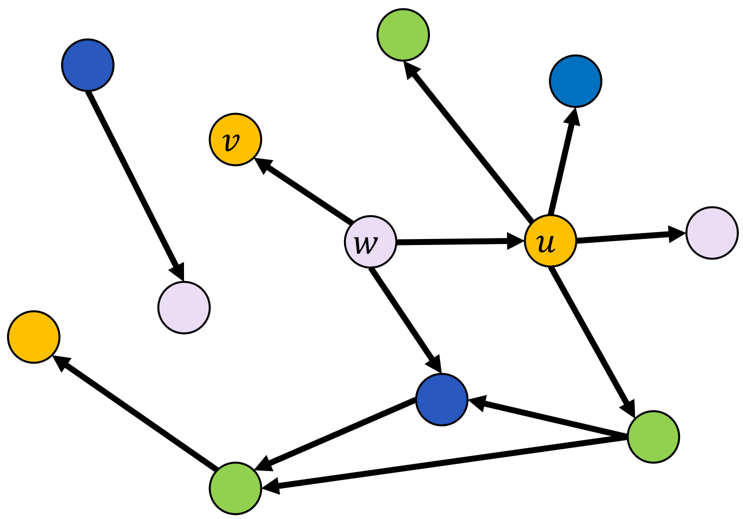
Invariant: at most $4\sqrt{m}$ **outgoing**



- Round complexity: $O(1)$ worst-case
- Message complexity: $\cancel{O(1)}$ ~~amortized~~
  - $O(1)$ **worst-case**
  - Gradually 20 reorientations per update for the next $p_v/10$ updates

Dynamic Distributed $(\Delta + 1)$-Coloring

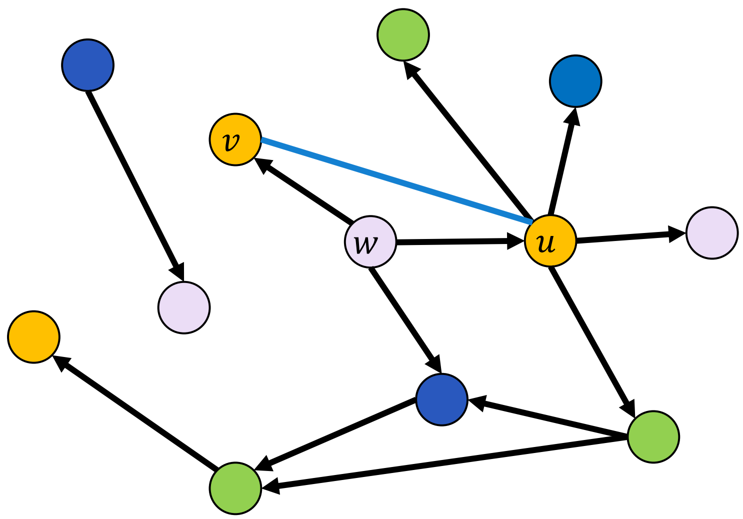
Invariant: at most $4\sqrt{m}$ **outgoing**



- Hard case: edge insertions

Dynamic Distributed $(\Delta + 1)$-Coloring

Invariant: at most $4\sqrt{m}$ **outgoing**

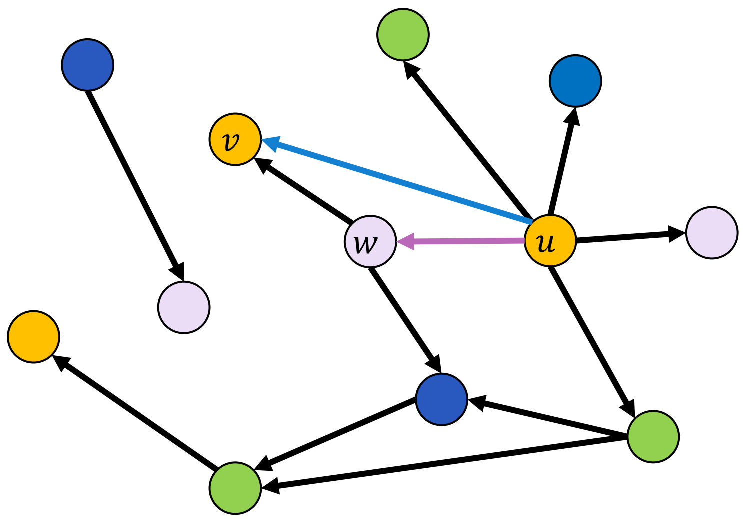


(u, v) edge insertion

- Hard case: edge insertions
- Perform edge orientation algorithm; reorient if necessary

Dynamic Distributed $(\Delta + 1)$-Coloring

Invariant: at most $4\sqrt{m}$ **outgoing**

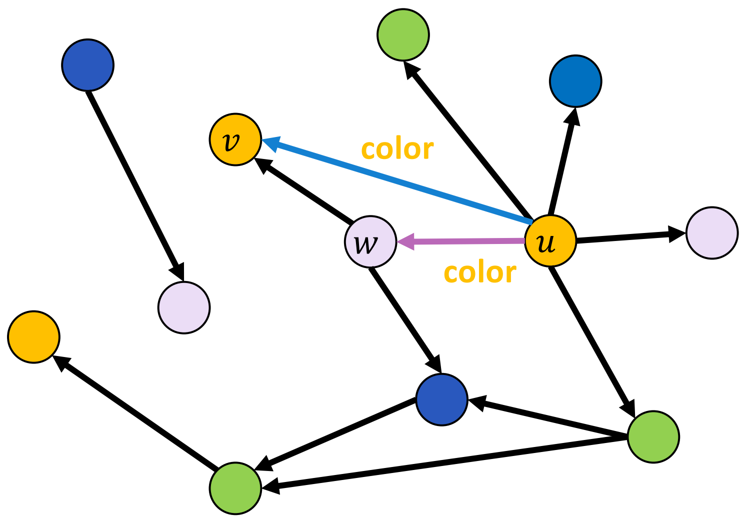


(u, v) edge insertion

- Hard case: edge insertions
- Perform edge orientation algorithm; reorient if necessary

Dynamic Distributed $(\Delta + 1)$-Coloring

Invariant: at most **$4\sqrt{m}$ outgoing**

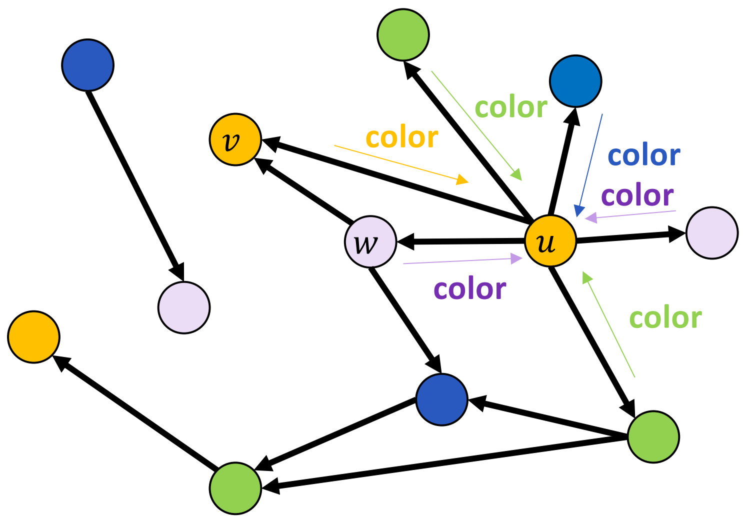


u sends color to v and w

- Hard case: edge insertions
- Perform edge orientation algorithm; reorient if necessary
- Each flipped edge, update neighbor about color

Dynamic Distributed $(\Delta + 1)$-Coloring

Invariant: at most $4\sqrt{m}$ **outgoing**

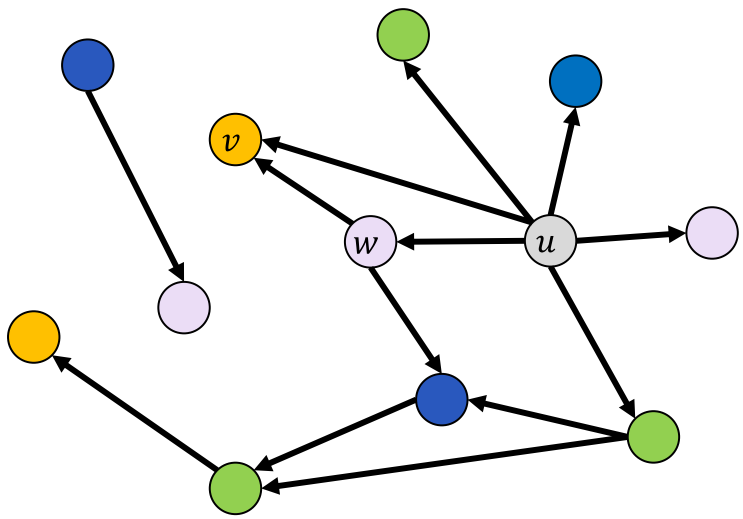


outgoing send colors to u

- Hard case: edge insertions
- Perform edge orientation algorithm; reorient if necessary
- Each flipped edge, update neighbor about color
- Ask outgoing neighbors their colors

Dynamic Distributed $(\Delta + 1)$-Coloring

Invariant: at most $\mathbf{4\sqrt{m}}$ **outgoing**

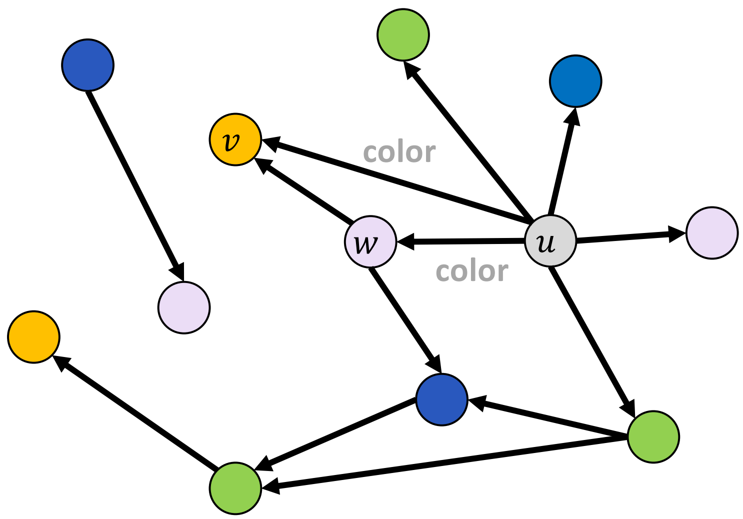


u recolors itself

- Hard case: edge insertions
- Perform edge orientation algorithm; reorient if necessary
- Each flipped edge, update neighbor about color
- Ask outgoing neighbors their colors
- Arbitrarily pick vertex recolor

Dynamic Distributed $(\Delta + 1)$-Coloring

Invariant: at most $4\sqrt{m}$ **outgoing**

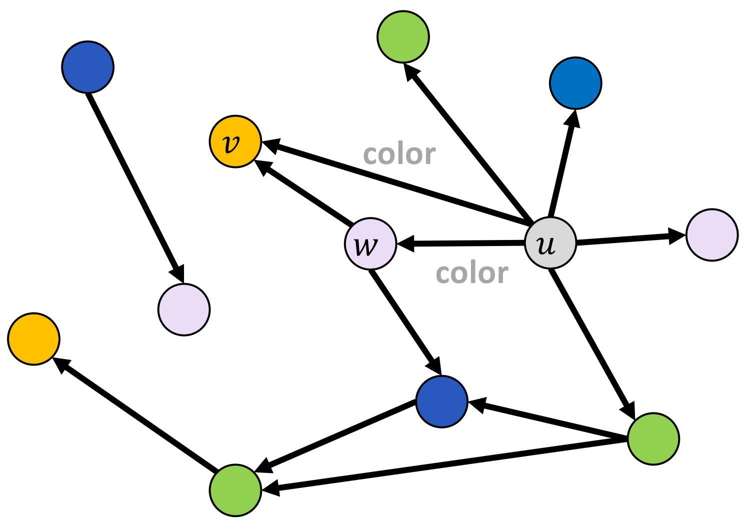


u recolors itself

- Hard case: edge insertions
- Perform edge orientation algorithm; reorient if necessary
- Each flipped edge, update neighbor about color
- Ask outgoing neighbors their colors
- Arbitrarily pick vertex recolor
- Send new color to outgoing

Dynamic Distributed $(\Delta + 1)$-Coloring

Invariant: at most $4\sqrt{m}$ **outgoing**

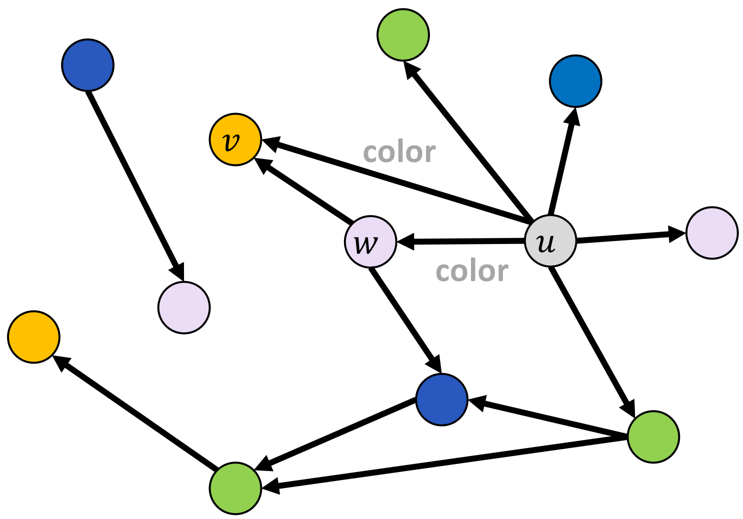


u recolors itself

- Correctness: u knows **all neighbor colors**
  - Can pick free color

Dynamic Distributed $(\Delta + 1)$-Coloring

Invariant: at most **$4\sqrt{m}$ outgoing**

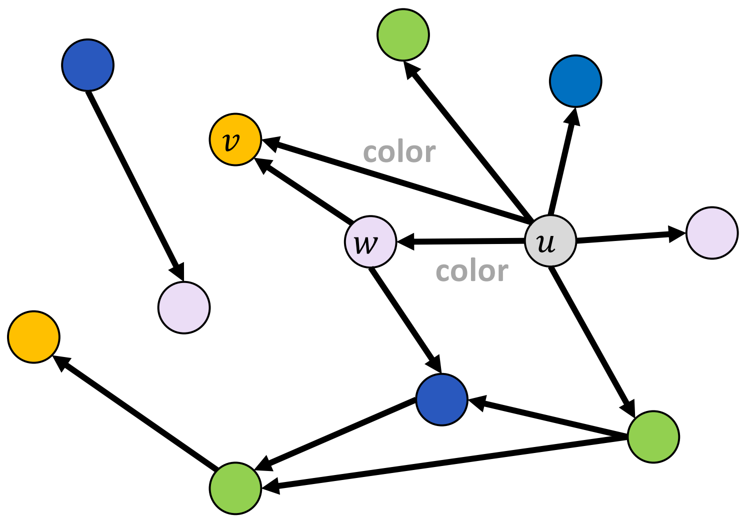


u recolors itself

- Correctness: u knows **all neighbor colors**
  - Can pick free color
- Message Complexity: $O(\sqrt{m})$ worst-case
  - Due to edge-orientation

Dynamic Distributed $(\Delta + 1)$-Coloring

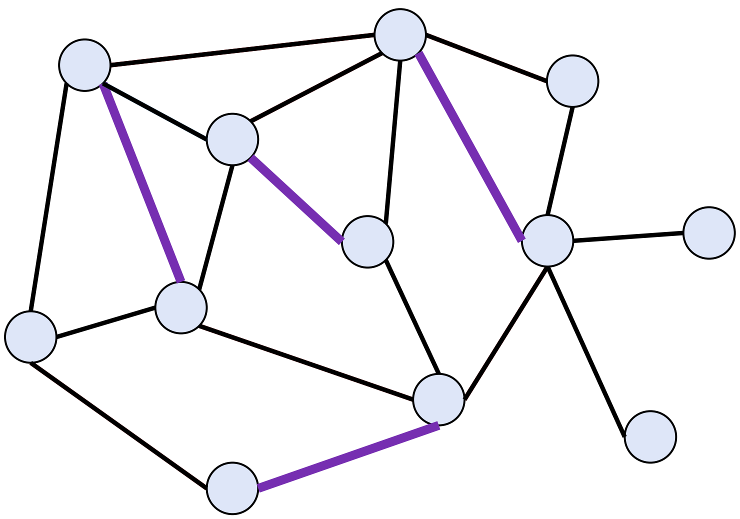
Invariant: at most $4\sqrt{m}$ **outgoing**



u recolors itself

- Correctness: u knows **all neighbor colors**
  - Can pick free color
- Message Complexity: $O(\sqrt{m})$ worst-case
  - Due to edge-orientation
- Round Complexity: $O(1)$ worst-case

Maximal Matching

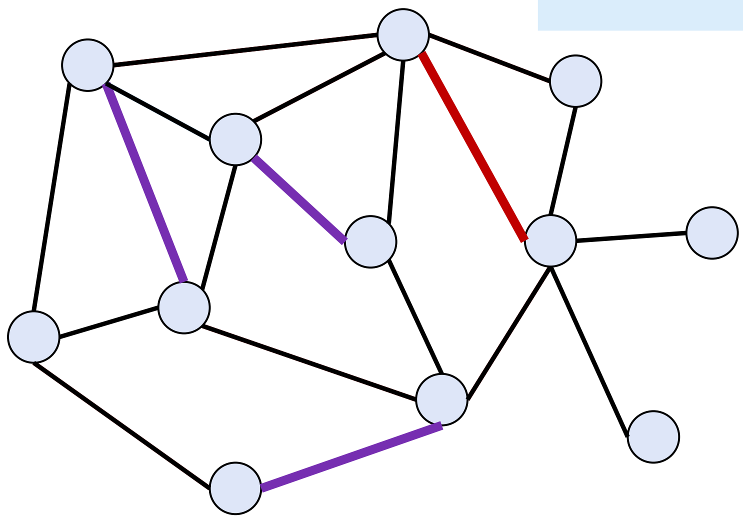


Each vertex matched to at most one neighbor

All vertices which can be matched are matched

Maximal Matching

Edge Insertions and Deletions May Violating Matching Maximality

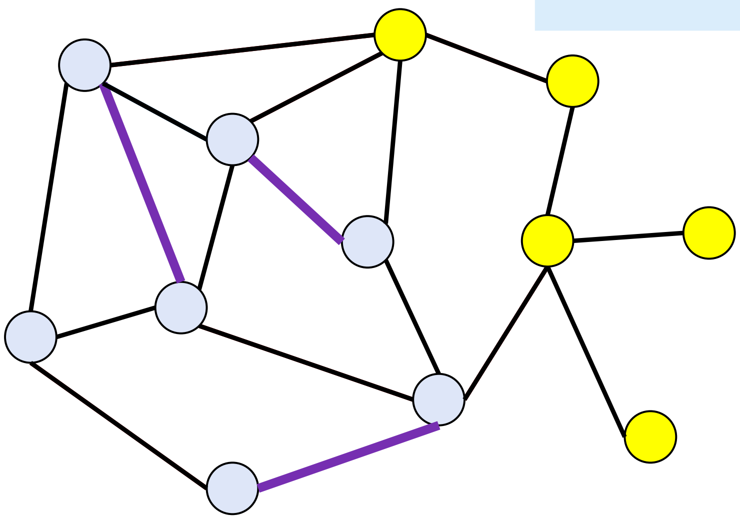


Each vertex matched to at most one neighbor

All vertices which can be matched are matched

Maximal Matching

Edge Insertions and Deletions May Violating Matching Maximality

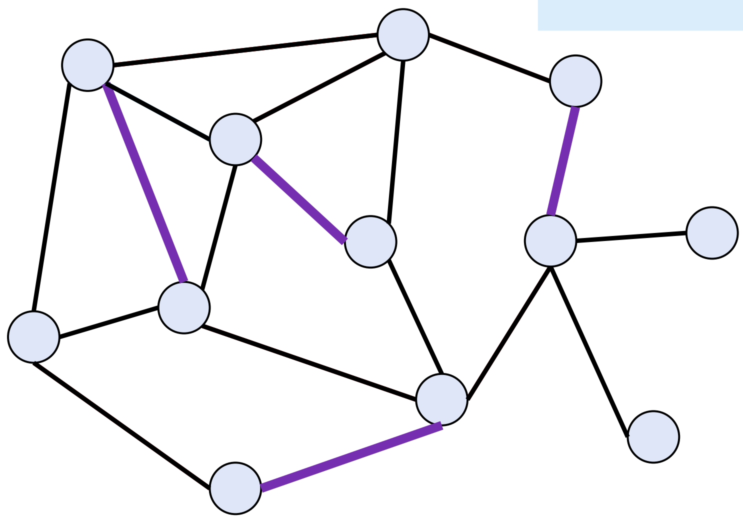


Each vertex matched to at most one neighbor

All vertices which can be matched are matched

Maximal Matching

Edge Insertions and Deletions May Violating Matching Maximality

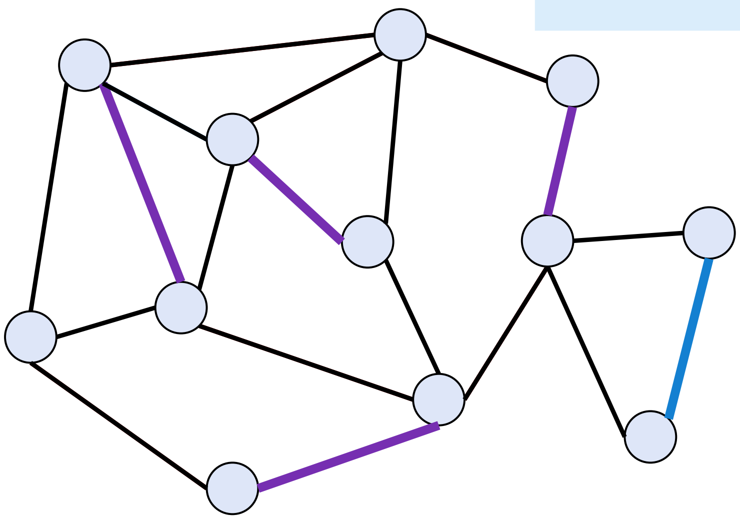


Each vertex matched to at most one neighbor

All vertices which can be matched are matched

Maximal Matching

Edge Insertions and Deletions May Violating Matching Maximality

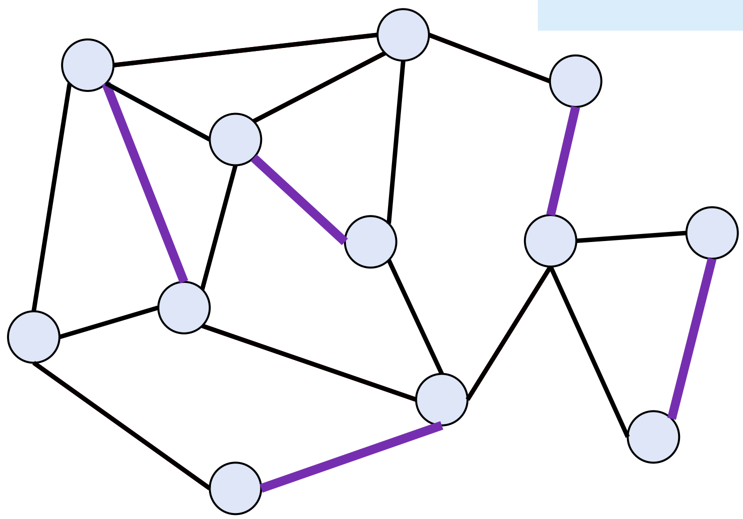


Each vertex matched to at most one neighbor

All vertices which can be matched are matched

Maximal Matching

Edge Insertions and Deletions May Violating Matching Maximality

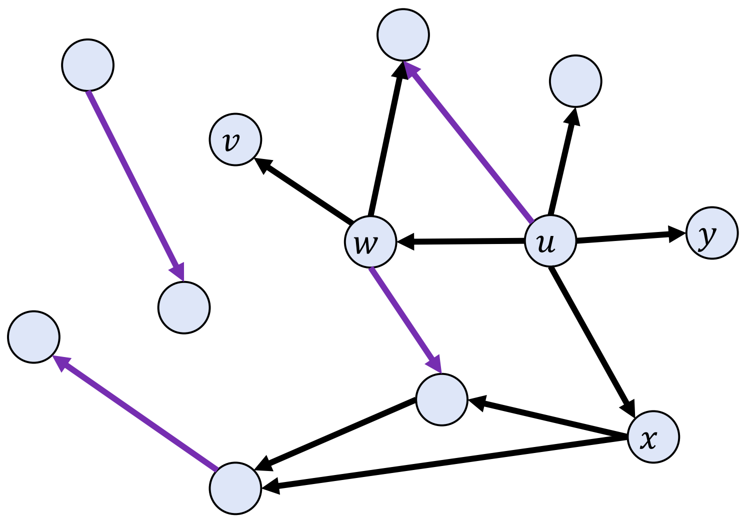


Each vertex matched to at most one neighbor

All vertices which can be matched are matched

Dynamic Distributed Maximal Matching

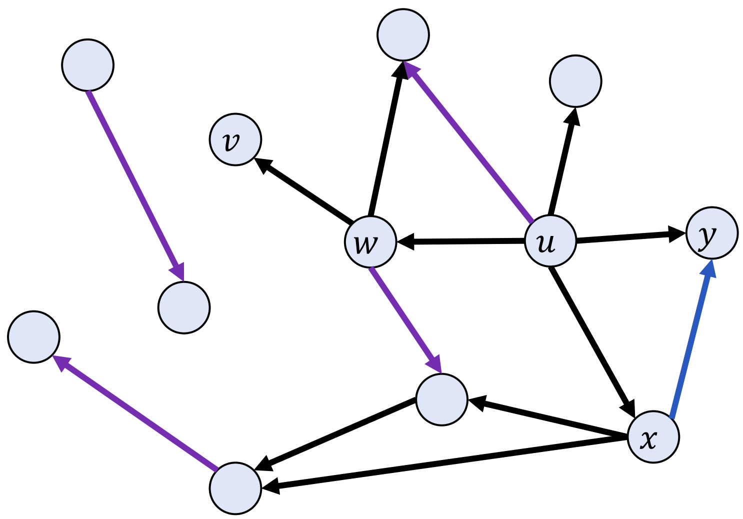
Invariant: at most $4\sqrt{m}$ **outgoing**



- Easy case: edge insertions

Dynamic Distributed Maximal Matching

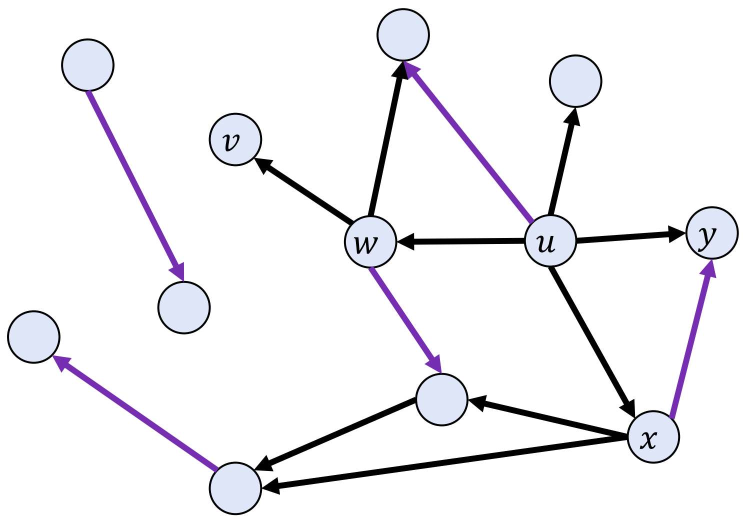
Invariant: at most $4\sqrt{m}$ **outgoing**



- Easy case: edge insertions
  - Orient edges as needed

Dynamic Distributed Maximal Matching

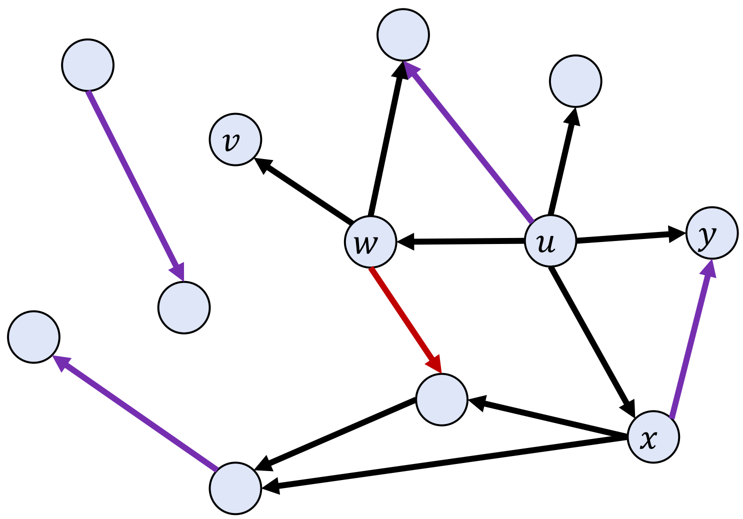
Invariant: at most $\mathbf{4\sqrt{m}}$ **outgoing**



- Easy case: edge insertions
  - Orient edges as needed
  - Match if both endpoints not matched

Dynamic Distributed Maximal Matching

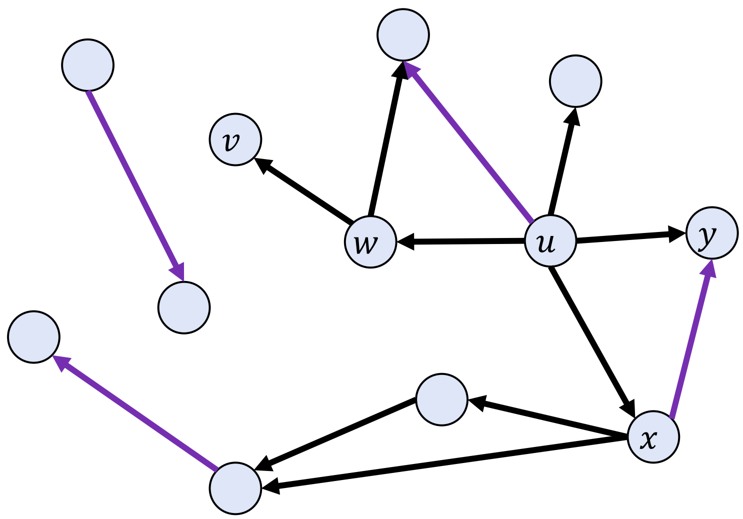
Invariant: at most $4\sqrt{m}$ **outgoing**



- Harder case: edge deletions

Dynamic Distributed Maximal Matching

Invariant: at most **$4\sqrt{m}$ outgoing**

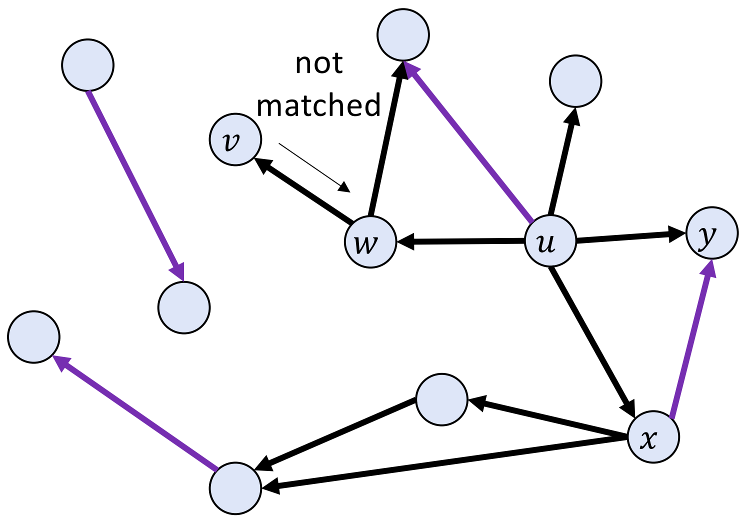


u is matched

- Harder case: edge deletions
- Match to incoming neighbor if any unmatched

Dynamic Distributed Maximal Matching

Invariant: at most $4\sqrt{m}$ **outgoing**

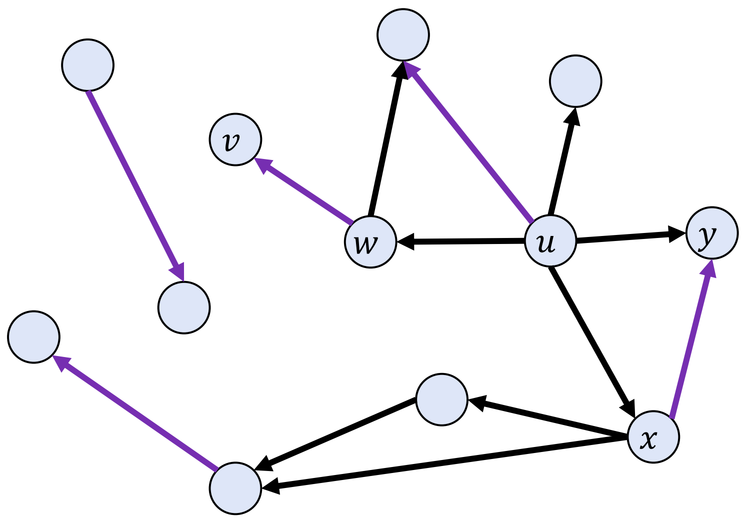


v is **not** matched

- Harder case: edge deletions
- Match to incoming neighbor if any unmatched
- Otherwise ask outgoing neighbors if they are matched

Dynamic Distributed Maximal Matching

Invariant: at most $4\sqrt{m}$ **outgoing**

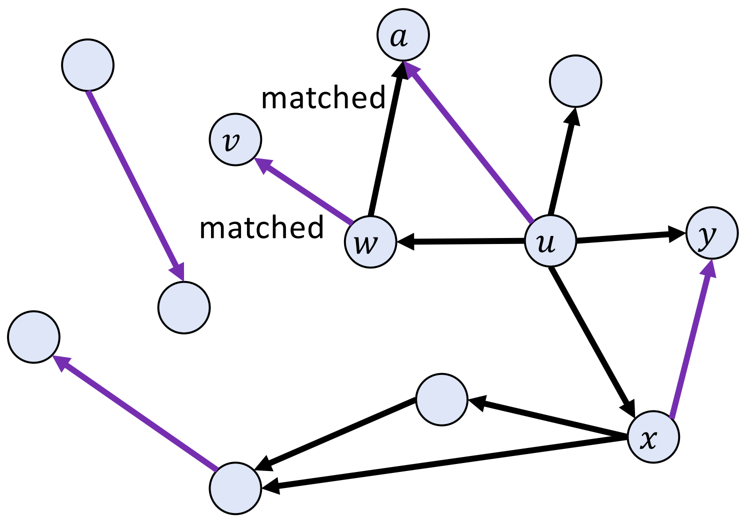


(v, w) are now **matched**

- Harder case: edge deletions
- Match to incoming neighbor if any unmatched
- Otherwise ask outgoing neighbors if they are matched
- Match to unmatched outgoing neighbor

Dynamic Distributed Maximal Matching

Invariant: at most $4\sqrt{m}$ **outgoing**



w tells v and a it is matched

- Harder case: edge deletions
- Match to incoming neighbor if any unmatched
- Otherwise ask outgoing neighbors if they are matched
- Match to unmatched outgoing neighbor
- Inform outgoing neighbors

Our Deterministic Algorithm Results

$(\Delta + 1)$-Vertex Coloring

- $O(\sqrt{m})$ messages and $O(1)$ rounds, both worst-case
- Dynamic Edge Orientation Technique
- Matches best-known sequential, centralized algorithm of [KNNP20]

(3/2)-Maximum Matching

- $O(\sqrt{m})$ amortized messages and $O(\log \Delta)$ rounds, worst case
- High-Degree/Low-Degree Partitioning Using Surrogates
- Matches best-known sequential, centralized algorithm of [NS13]

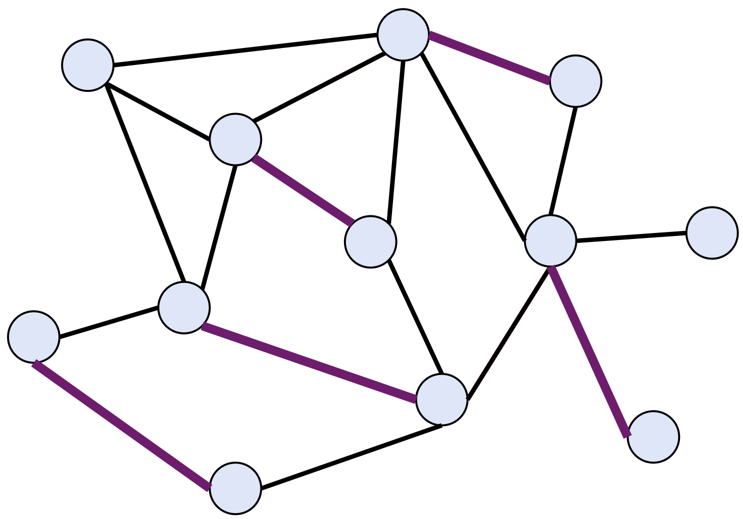
Maximal Matching

- $O(\sqrt{m})$ messages and $O(1)$ rounds, both worst-case
- Dynamic Edge Orientation Technique
- Matches best-known sequential, centralized algorithm of [NS13]

Maximal Independent Set

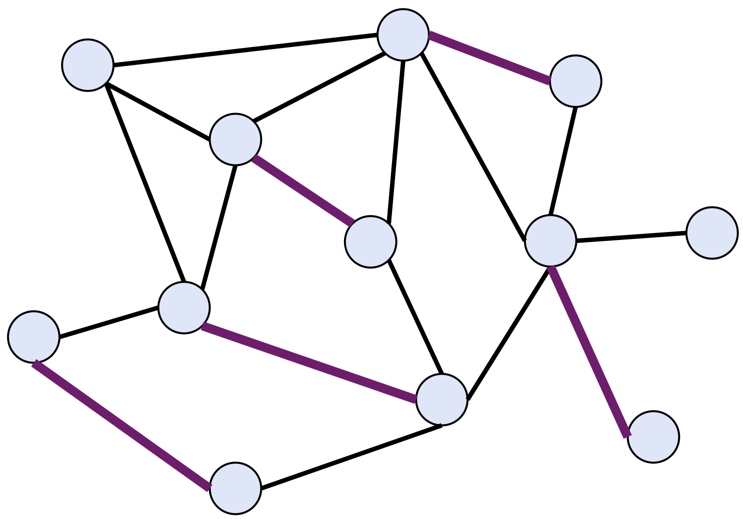
- $O\left(m^{2/3} \log^2 n\right)$ messages and $O(\log^2 n)$ rounds, amortized
- High-Degree/Low-Degree Partitioning Using 6-Hop Neighborhood
- Use a small-diameter *static* algorithm to obtain MIS in high-degree and *dynamic* MIS for low-degree
- Matches best-known sequential, centralized [GK21] up to $\tilde{O}(1)$ factor

(3/2)-Approximate Maximum Matching



(3/2)-Approximation of Maximum Matching

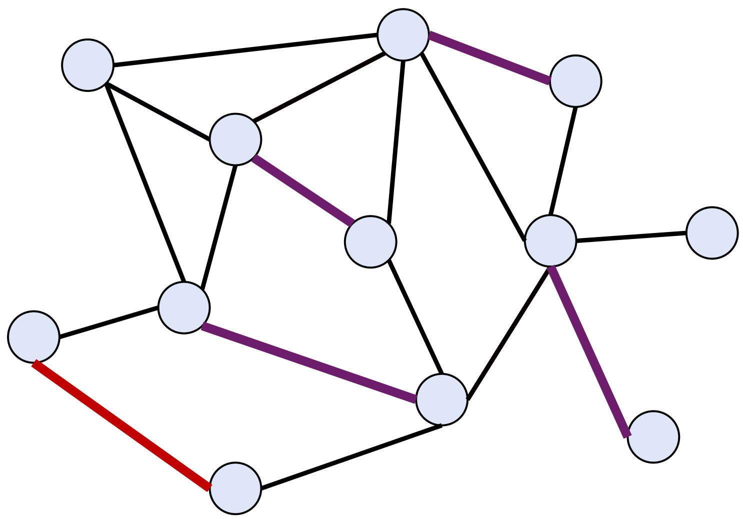
(3/2)-Approximate Maximum Matching



(3/2)-Approximation of Maximum Matching

Edge Insertions and Deletions May Change Size of Maximum Matching

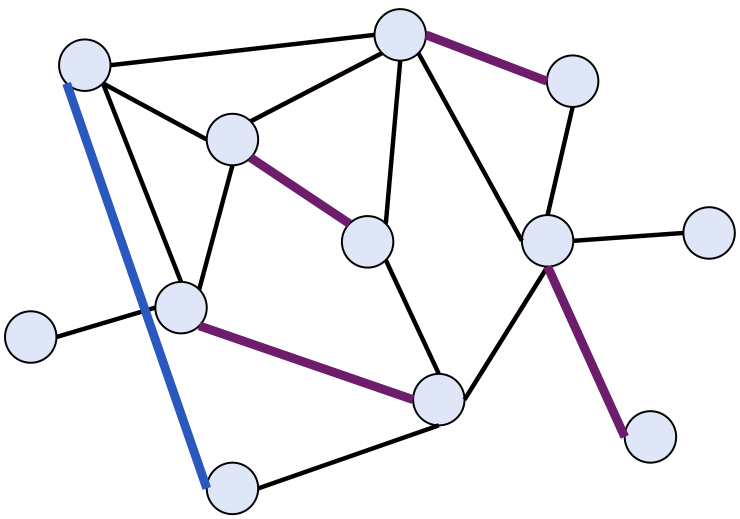
(3/2)-Approximate Maximum Matching



(3/2)-Approximation of Maximum Matching

Edge Insertions and Deletions May Change Size of Maximum Matching

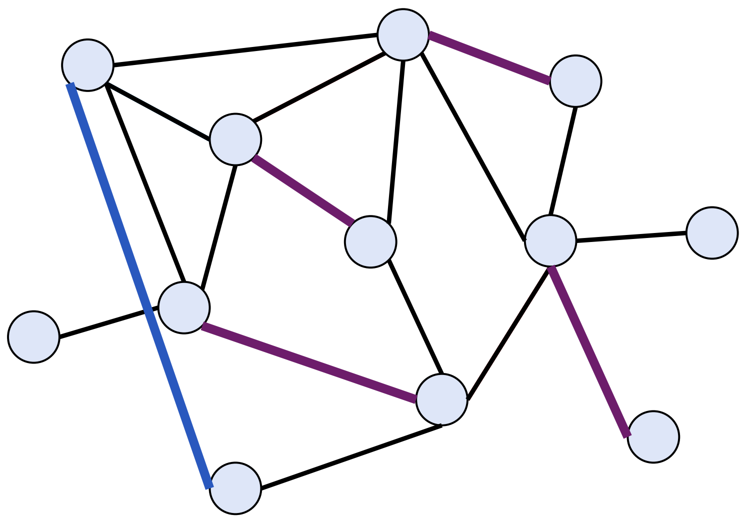
(3/2)-Approximate Maximum Matching



(3/2)-Approximation of Maximum Matching

Edge Insertions and Deletions May Change Size of Maximum Matching

(3/2)-Approximate Maximum Matching



(3/2)-Approximation of Maximum Matching

Edge Insertions and Deletions May Change Size of Maximum Matching

Maximum matching increased by 1

Sequential, Centralized Dynamic (3/2)-Maximum Matching

- Neiman and Solomon (STOC 2013):
  - Any (3/2)-maximum matching does not have an **augmenting path** of length 3 or longer

Sequential, Centralized Dynamic (3/2)-Maximum Matching

- Neiman and Solomon (STOC 2013):
  - Any (3/2)-maximum matching does not have an **augmenting path** of length 3 or longer
    - Path that starts and ends on unmatched vertices and alternate between edges in matching and not

Sequential, Centralized Dynamic (3/2)-Maximum Matching

- Neiman and Solomon (STOC 2013):
  - Any (3/2)-maximum matching does not have an **augmenting path** of length 3 or longer
    - Path that starts and ends on unmatched vertices and alternate between edges in matching and not
  - Partition vertices into high-degree ($\geq \sqrt{m}$) and low-degree
  - **Always match high-degree vertices**

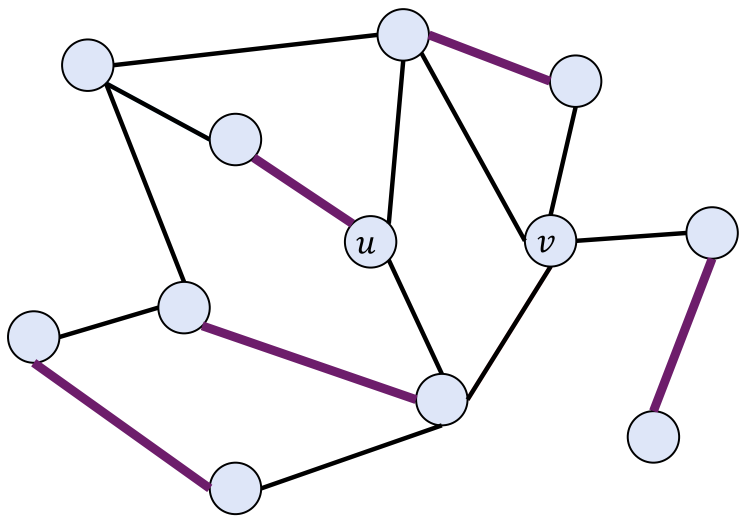
Sequential, Centralized Dynamic (3/2)-Maximum Matching

- Neiman and Solomon (STOC 2013):
  - Any (3/2)-maximum matching does not have an **augmenting path** of length 3 or longer
    - Path that starts and ends on unmatched vertices and alternate between edges in matching and not
  - Partition vertices into high-degree ($\geq \sqrt{m}$) and low-degree
  - **Always match high-degree vertices**
  - Look for augmenting paths through **surrogates**

Sequential, Centralized Dynamic (3/2)-Maximum Matching

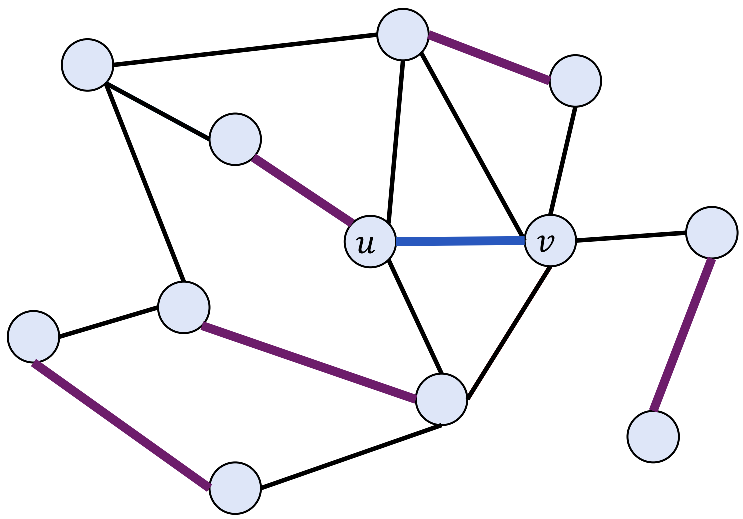
- Neiman and Solomon (STOC 2013):
  - Any (3/2)-maximum matching does not have an **augmenting path** of length 3 or longer
    - Path that starts and ends on unmatched vertices and alternate between edges in matching and not
  - Partition vertices into high-degree ($\geq \sqrt{m}$) and low-degree
  - **Always match high-degree vertices**
  - Look for augmenting paths through **surrogates**

Distributed, Dynamic (3/2)-Maximum Matching



- Key Idea: **Degree doubling** to find augmenting paths

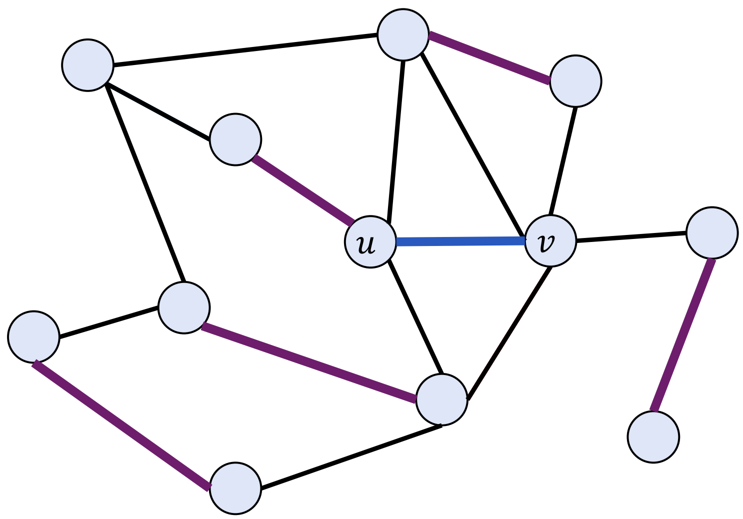
Distributed, Dynamic (3/2)-Maximum Matching



Edge Insertion (u, v)

- **Key Idea: Degree doubling** to find augmenting paths
- On update, search neighbors for free vertex or surrogate
  - Start with 1 neighbor

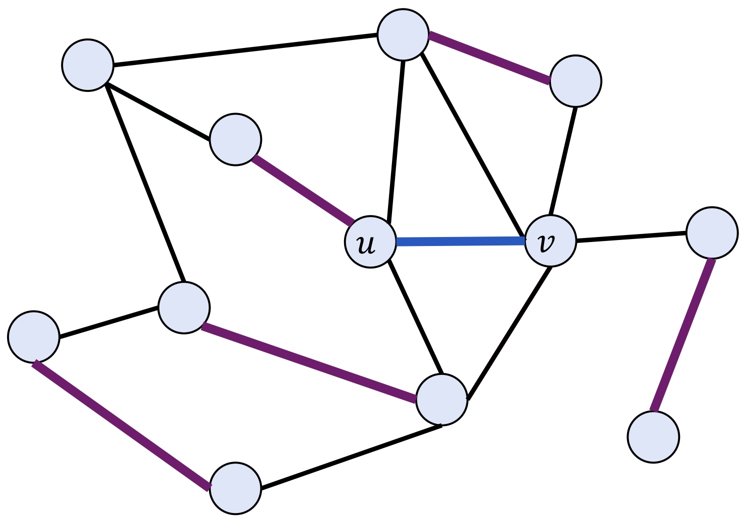
Distributed, Dynamic (3/2)-Maximum Matching



Edge Insertion (u, v)

- **Key Idea: Degree doubling** to find augmenting paths
- On update, search neighbors for free vertex or surrogate
  - Start with 1 neighbor
  - **Successively double neighbors searched, 2^i**

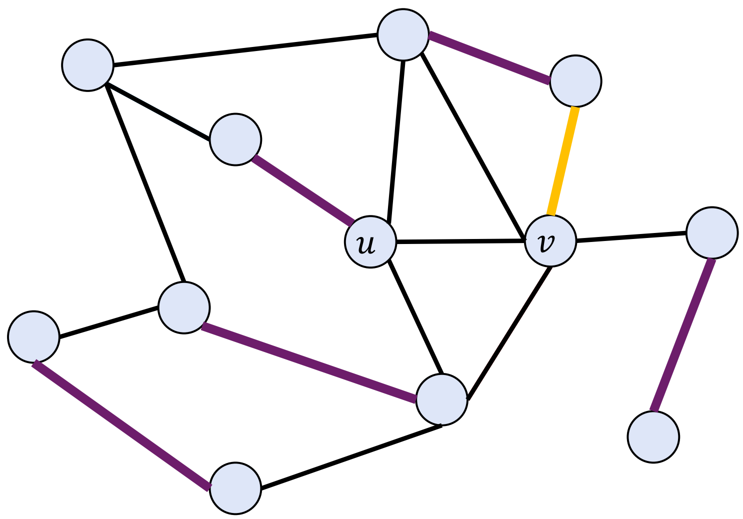
Distributed, Dynamic (3/2)-Maximum Matching



Edge Insertion (u, v)

- **Key Idea: Degree doubling** to find augmenting paths
- On update, search neighbors for free vertex or surrogate
  - Start with 1 neighbor
  - **Successively double neighbors searched, 2^i**
- **Surrogate:** matched neighbor whose mate has degree $\leq \sqrt{2^i}$

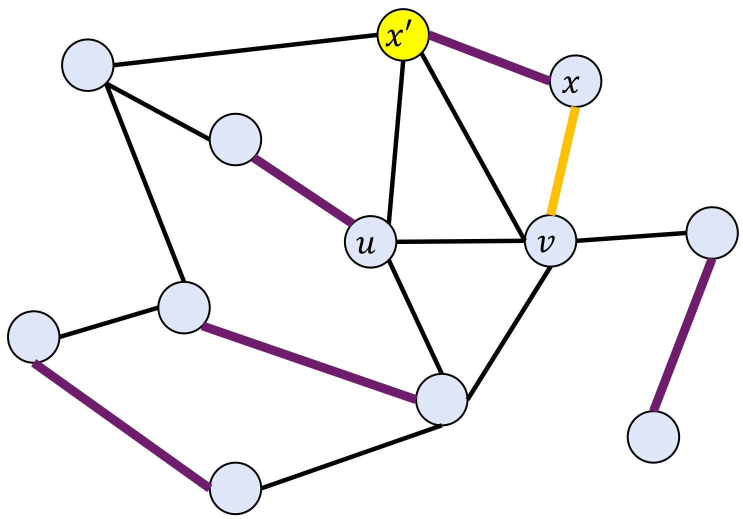
Distributed, Dynamic (3/2)-Maximum Matching



v searches 1 neighbor

- **Key Idea: Degree doubling** to find augmenting paths
- On update, search neighbors for free vertex or surrogate
  - Start with 1 neighbor
  - **Successively double neighbors searched, 2^i**
- **Surrogate:** matched neighbor whose mate has degree $\leq \sqrt{2^i}$

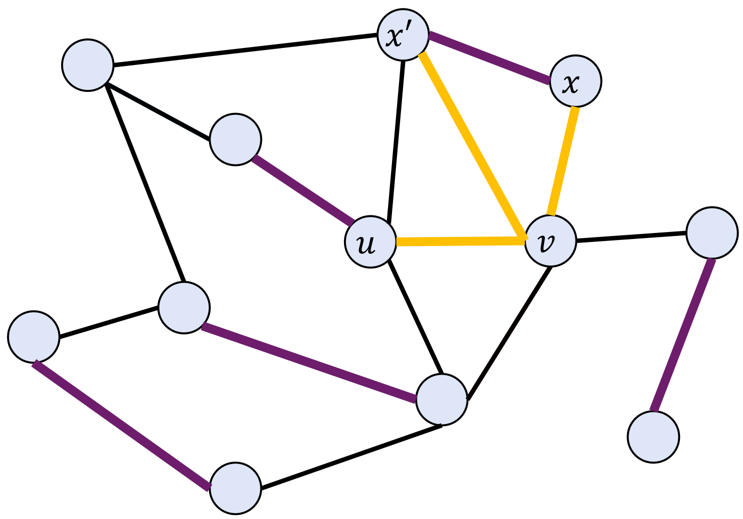
Distributed, Dynamic (3/2)-Maximum Matching



x does not have degree 1

- **Key Idea: Degree doubling** to find augmenting paths
- On update, search neighbors for free vertex or surrogate
  - Start with 1 neighbor
  - **Successively double neighbors searched, 2^i**
- **Surrogate:** matched neighbor whose mate has degree $\leq \sqrt{2^i}$

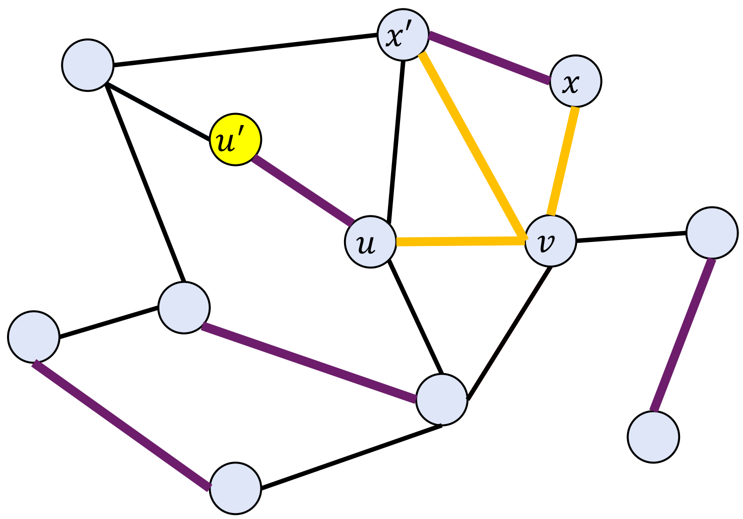
Distributed, Dynamic (3/2)-Maximum Matching



v searches two additional neighbors

- **Key Idea: Degree doubling** to find augmenting paths
- On update, search neighbors for free vertex or surrogate
  - Start with 1 neighbor
  - **Successively double neighbors searched, 2^i**
- **Surrogate:** matched neighbor whose mate has degree $\leq \sqrt{2^i}$

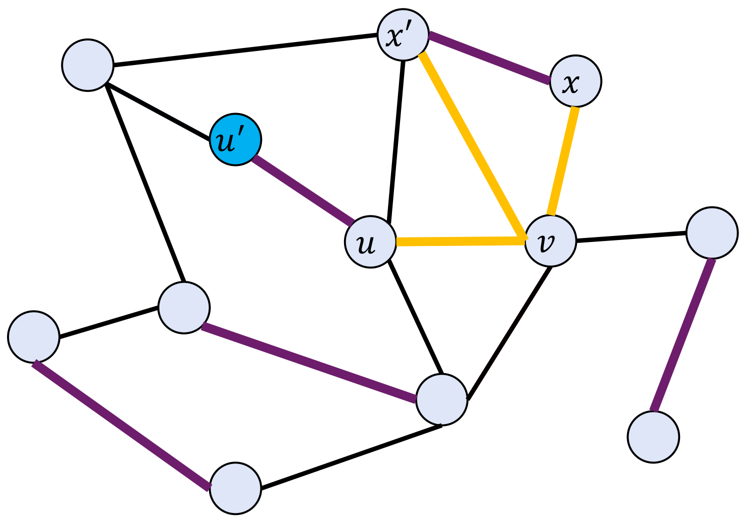
Distributed, Dynamic (3/2)-Maximum Matching



u has a mate u' with degree $\leq \sqrt{2}$

- **Key Idea: Degree doubling** to find augmenting paths
- On update, search neighbors for free vertex or surrogate
  - Start with 1 neighbor
  - **Successively double neighbors searched, 2^i**
- **Surrogate:** matched neighbor whose mate has degree $\leq \sqrt{2^i}$

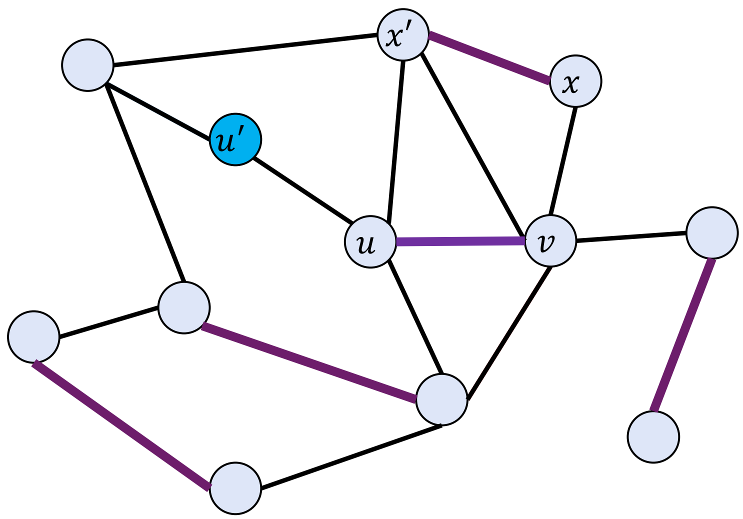
Distributed, Dynamic (3/2)-Maximum Matching



u' is a surrogate

- **Key Idea: Degree doubling** to find augmenting paths
- On update, search neighbors for free vertex or surrogate
  - Start with 1 neighbor
  - **Successively double neighbors searched, 2^i**
- **Surrogate:** matched neighbor whose mate has degree $\leq \sqrt{2^i}$

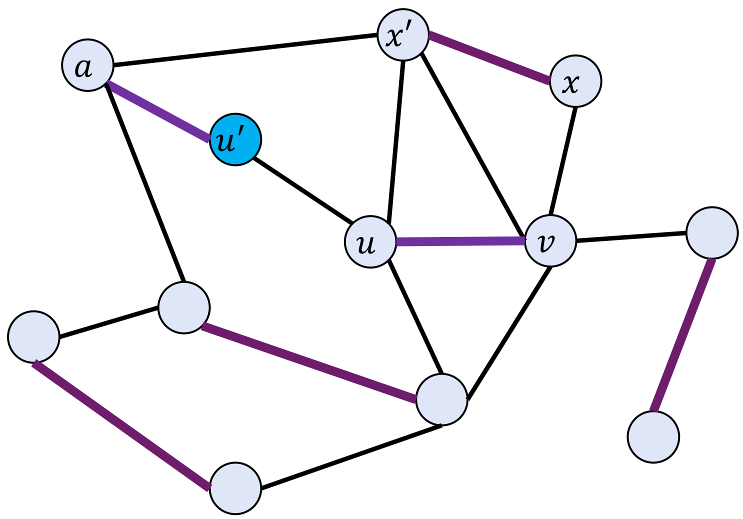
Distributed, Dynamic (3/2)-Maximum Matching



v matches with u

- **Match with neighbor if surrogate found**

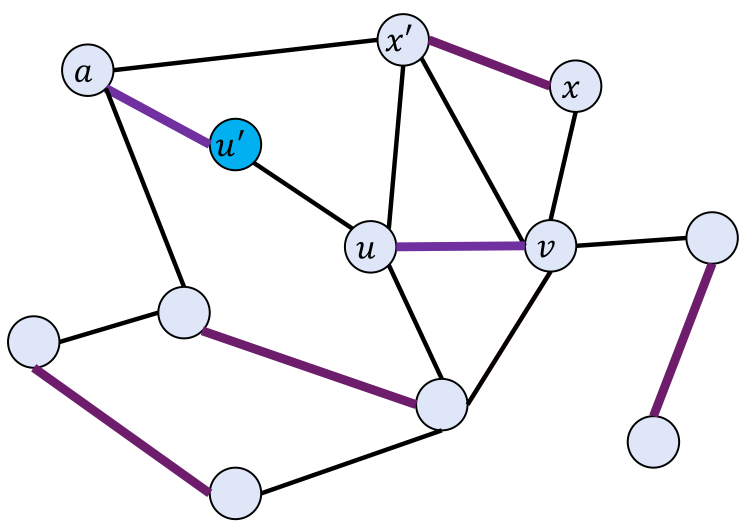
Distributed, Dynamic (3/2)-Maximum Matching



u' matches with a

- **Match with neighbor if surrogate found**
  - Surrogate matches with free neighbor if exists

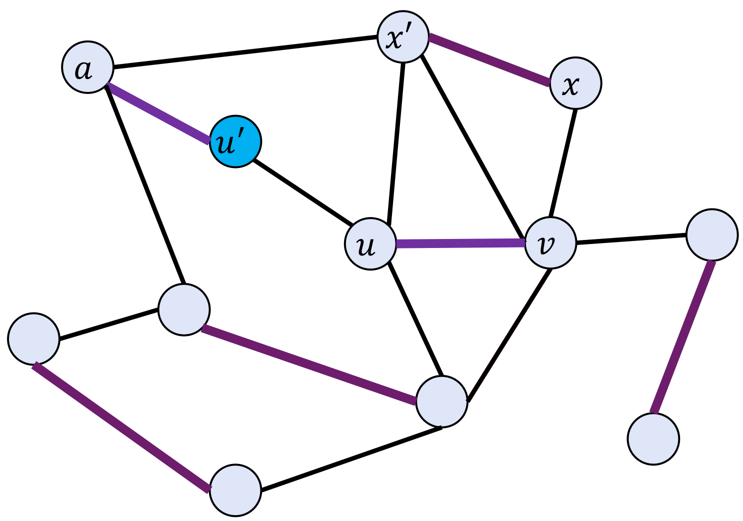
Distributed, Dynamic (3/2)-Maximum Matching



u' matches with a

- **Match with neighbor if surrogate found**
  - Surrogate matches with free neighbor if exists
- Similar procedure for deletions

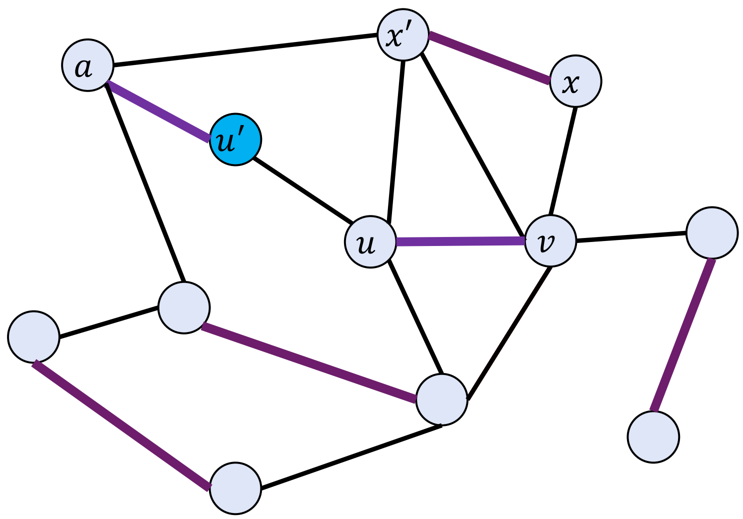
Distributed, Dynamic (3/2)-Maximum Matching



u' matches with a

- Round complexity: $O(\log \Delta)$ worst-case

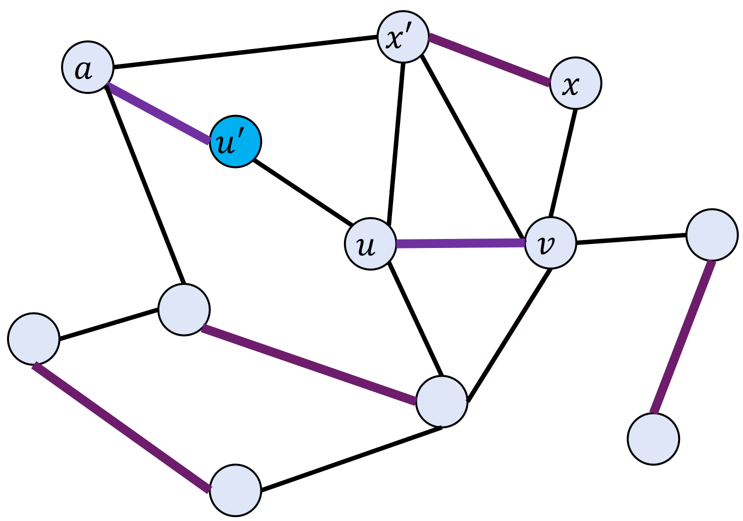
Distributed, Dynamic (3/2)-Maximum Matching



u' matches with a

- Round complexity: $O(\log \Delta)$ worst-case
  - Search at most Δ neighbors, doubling

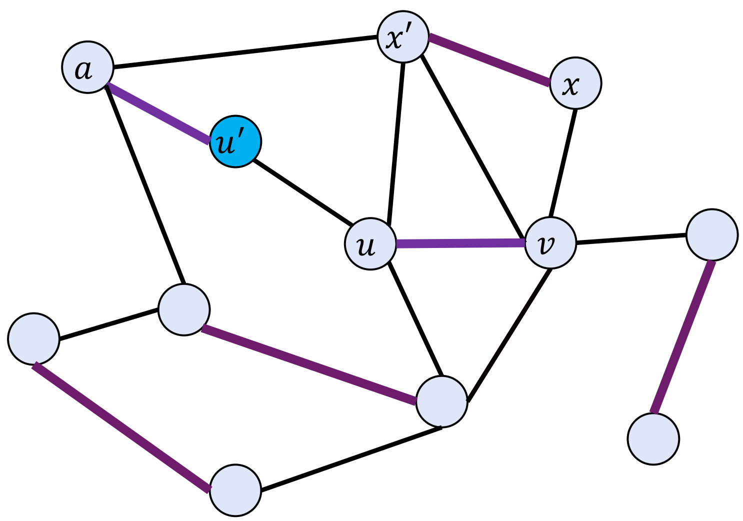
Distributed, Dynamic (3/2)-Maximum Matching



u' matches with a

- Round complexity: $O(\log \Delta)$ worst-case
  - Search at most Δ neighbors, doubling
- Message complexity: $O(\sqrt{m})$ amortized

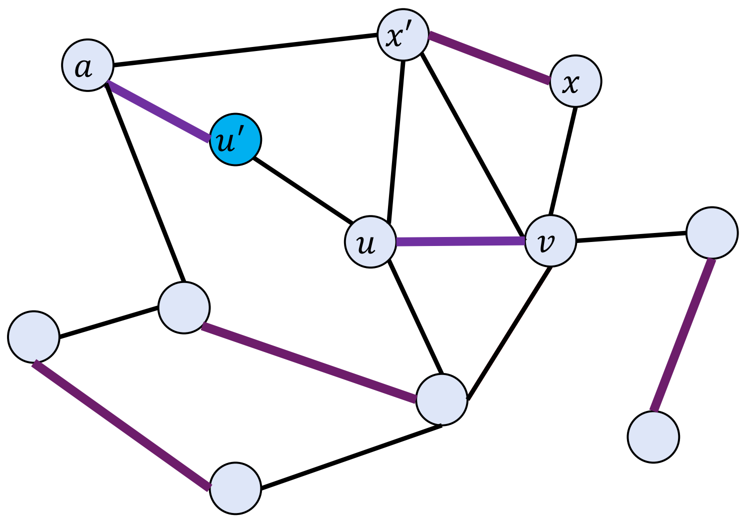
Distributed, Dynamic (3/2)-Maximum Matching



u' matches with a

- Round complexity: $O(\log \Delta)$ worst-case
  - Search at most Δ neighbors, doubling
- Message complexity: $O(\sqrt{m})$ amortized
  - **At most $\sqrt{m}$ matched neighbors with mates $\geq \sqrt{m}$ degree**

Distributed, Dynamic (3/2)-Maximum Matching



u' matches with a

- Round complexity: $O(\log \Delta)$ worst-case
  - Search at most Δ neighbors, doubling
- Message complexity: $O(\sqrt{m})$ amortized
  - **At most $\sqrt{m}$ matched neighbors with mates $\geq \sqrt{m}$ degree**
  - Need to search at most $2\sqrt{m}$ neighbors

Our Deterministic Algorithm Results

$(\Delta + 1)$-Vertex Coloring

- $O(\sqrt{m})$ messages and $O(1)$ rounds, both worst-case
- Dynamic Edge Orientation Technique
- Matches best-known sequential, centralized algorithm of [KNNP20]

(3/2)-Maximum Matching

- $O(\sqrt{m})$ amortized messages and $O(\log \Delta)$ rounds, worst case
- High-Degree/Low-Degree Partitioning Using Surrogates
- Matches best-known sequential, centralized algorithm of [NS13]

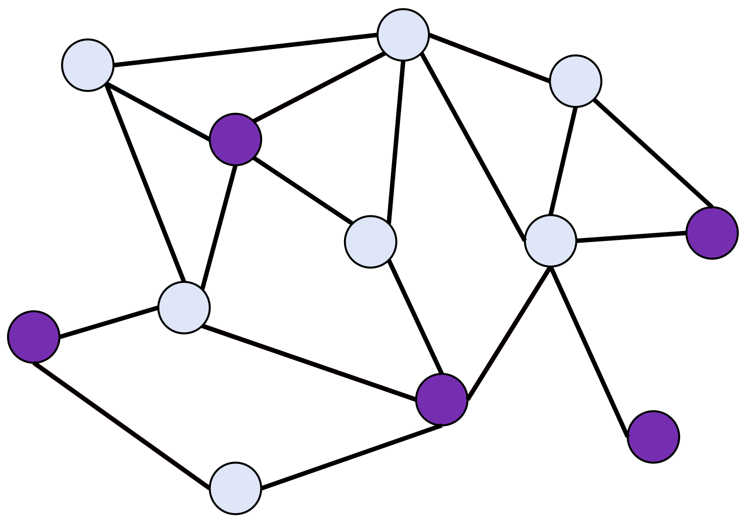
Maximal Matching

- $O(\sqrt{m})$ messages and $O(1)$ rounds, both worst-case
- Dynamic Edge Orientation Technique
- Matches best-known sequential, centralized algorithm of [NS13]

Maximal Independent Set

- $O\left(m^{2/3} \log^2 n\right)$ messages and $O(\log^2 n)$ rounds, amortized
- High-Degree/Low-Degree Partitioning Using 6-Hop Neighborhood
- Use a small-diameter *static* algorithm to obtain MIS in high-degree and *dynamic* MIS for low-degree
- Matches best-known sequential, centralized [GK21] up to $\tilde{O}(1)$ factor

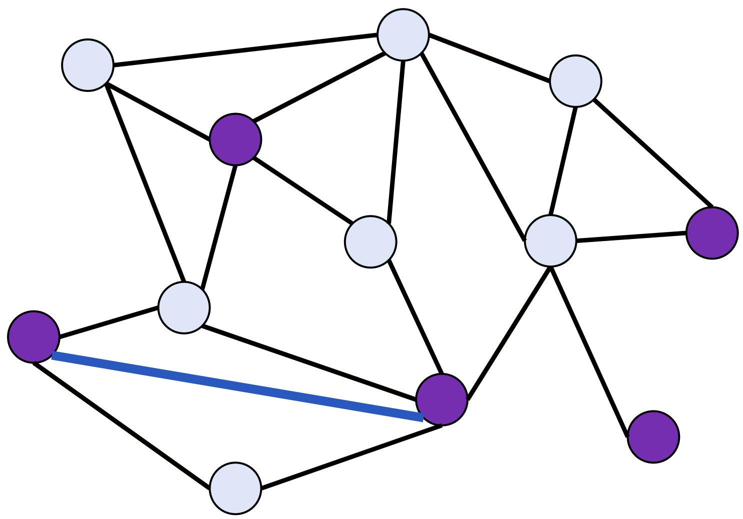
Maximal Independent Set (MIS)



No two vertices in the independent set are neighbors

All vertices that can be added to the independent set are added

Maximal Independent Set (MIS)

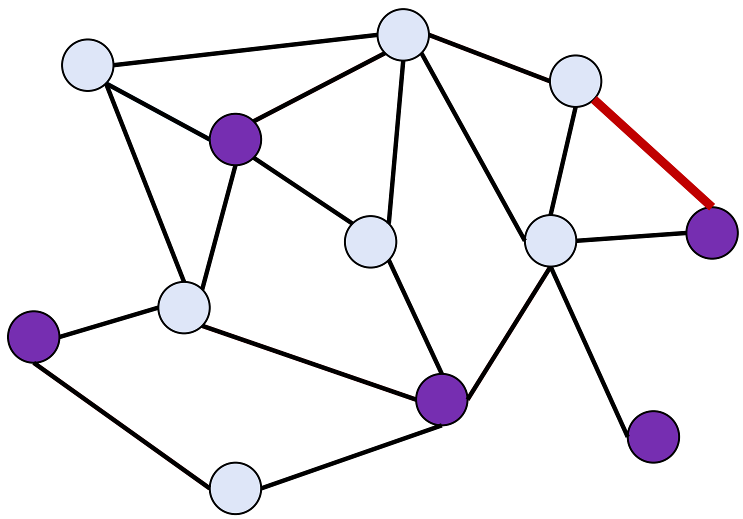


No two vertices in the independent set are neighbors

All vertices that can be added to the independent set are added

Edge Insertions May Violate Independence

Maximal Independent Set (MIS)

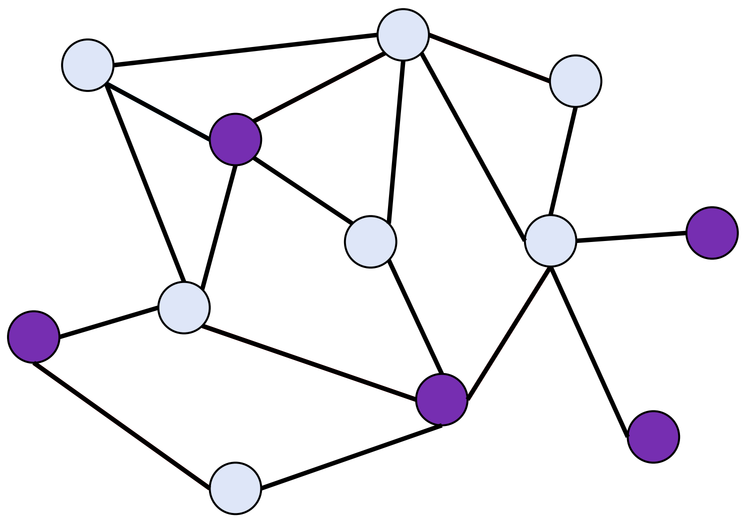


No two vertices in the independent set are neighbors

All vertices that can be added to the independent set are added

Edge Deletions May Violate Maximality

Maximal Independent Set (MIS)

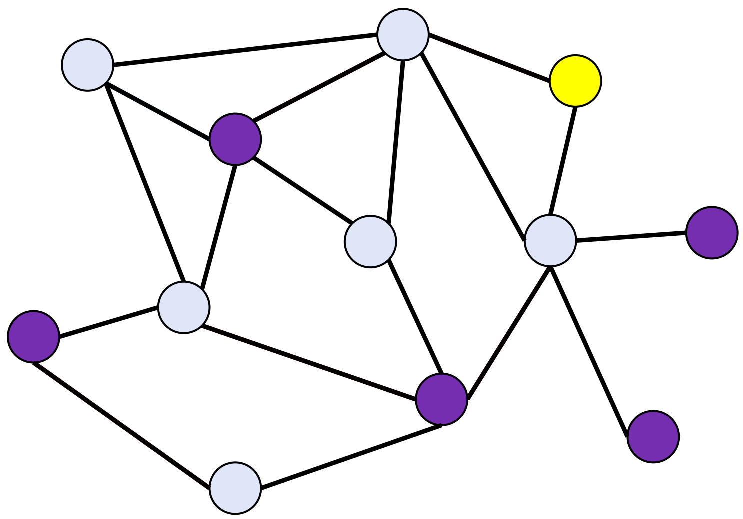


No two vertices in the independent set are neighbors

All vertices that can be added to the independent set are added

Edge Deletions May Violate Maximality

Maximal Independent Set (MIS)



No two vertices in the independent set are neighbors

All vertices that can be added to the independent set are added

Edge Deletions May Violate Maximality

Previous Deterministic Dynamic Distributed MIS

- Assadi, Onak, Schieber, and Solomon (STOC ‘18) provides a deterministic, dynamic, distributed MIS algorithm

Previous Deterministic Dynamic Distributed MIS

- Assadi, Onak, Schieber, and Solomon (STOC ‘18) provides a deterministic, dynamic, distributed MIS algorithm
  - $O(m^{3/4})$ amortized messages, $O(1)$ amortized rounds

Previous Deterministic Dynamic Distributed MIS

- Assadi, Onak, Schieber, and Solomon (STOC ‘18) provides a deterministic, dynamic, distributed MIS algorithm
  - $O(m^{3/4})$ amortized messages, $O(1)$ amortized rounds
  - Assumes graph remains connected throughout updates

Previous Deterministic Dynamic Distributed MIS

- Assadi, Onak, Schieber, and Solomon (STOC '18) provides a deterministic, dynamic, distributed MIS algorithm
  - $O(m^{3/4})$ amortized messages, $O(1)$ amortized rounds
  - Assumes graph remains connected throughout updates

Our Result: $O(m^{2/3} \log^2 n)$ amortized messages, $O(\log^2 n)$ amortized rounds

Does **not need connectivity assumption**

Sequential, Centralized, Dynamic MIS

- Gupta and Khan (SOSA 2021):
  - Partition nodes into **high-degree ($\geq m^{2/3}$)** and **low-degree**

Sequential, Centralized, Dynamic MIS

- Gupta and Khan (SOSA 2021):
  - Partition nodes into **high-degree ($\geq m^{2/3}$)** and **low-degree**
  - Low-degree nodes **prioritize membership in MIS**

Sequential, Centralized, Dynamic MIS

- Gupta and Khan (SOSA 2021):
  - Partition nodes into **high-degree ($\geq m^{2/3}$)** and **low-degree**
  - Low-degree nodes **prioritize membership in MIS**
    - If no low-degree neighbor in MIS, add self to MIS

Sequential, Centralized, Dynamic MIS

- Gupta and Khan (SOSA 2021):
  - Partition nodes into **high-degree ($\geq m^{2/3}$)** and **low-degree**
  - Low-degree nodes **prioritize membership in MIS**
    - If no low-degree neighbor in MIS, add self to MIS
  - High-degree nodes with no neighbors in MIS are added to MIS **after** processing all **low-degree nodes**

Sequential, Centralized, Dynamic MIS

- Gupta and Khan (SOSA 2021):
  - Partition nodes into **high-degree ($\geq m^{2/3}$)** and **low-degree**
  - Low-degree nodes **prioritize membership in MIS**
    - If no low-degree neighbor in MIS, add self to MIS
  - High-degree nodes with no neighbors in MIS are added to MIS **after** processing all **low-degree nodes**
  - Low-degree node entering/exiting MIS causes **all high-degree nodes** to find new MIS in induced subgraph (this is a **global restart**)

Distributing Challenges

Challenge 1: How do nodes determine if they're **high-degree/low-degree as m changes** with updates (for unknown m)?

Easy to achieve if the graph remains **connected throughout updates**

Distributing Challenges

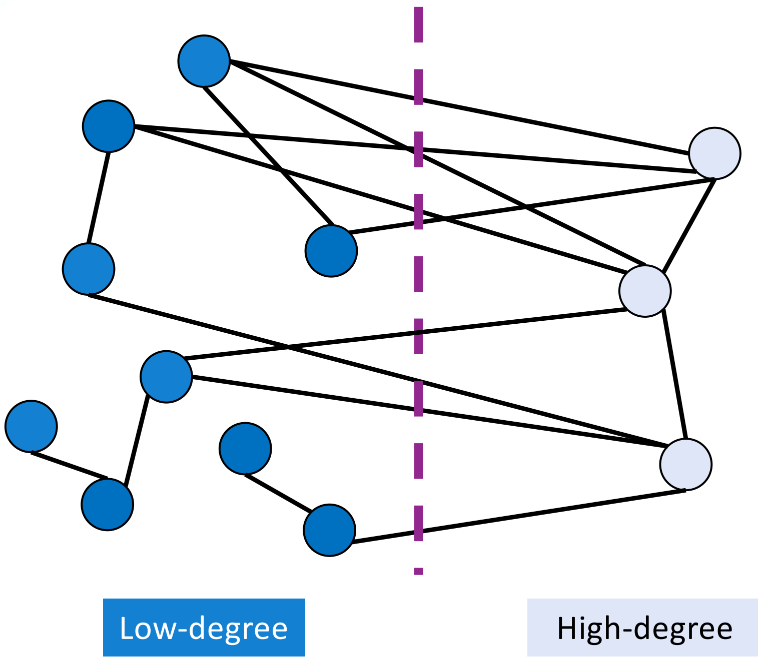
Challenge 1: How do nodes determine if they're **high-degree/low-degree as m changes** with updates (for unknown m)?

Easy to achieve if the graph remains **connected throughout updates**

Challenge 2: How do high-degree nodes compute maximal independent set in **small number of rounds** and **few messages**?

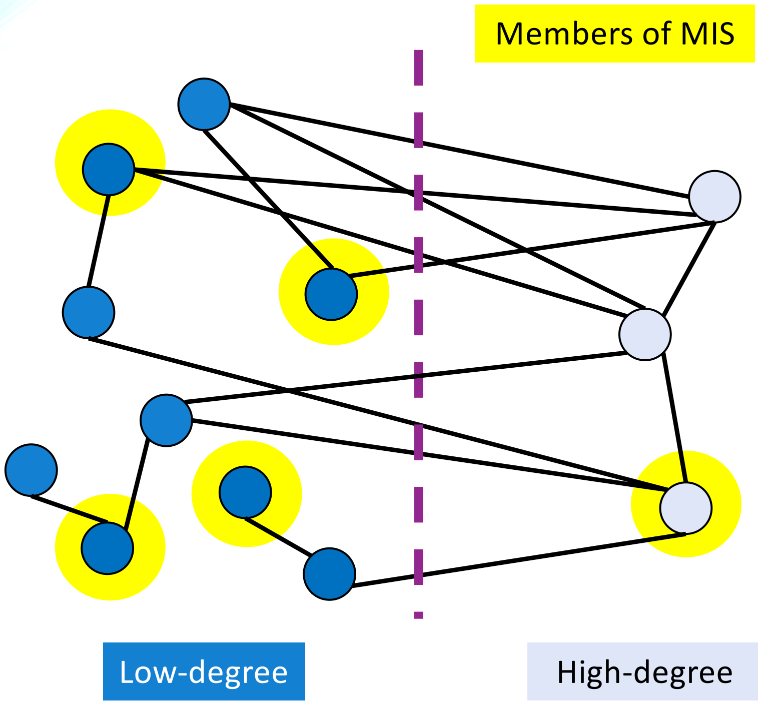
Global restarts are expensive

Distributed Dynamic MIS



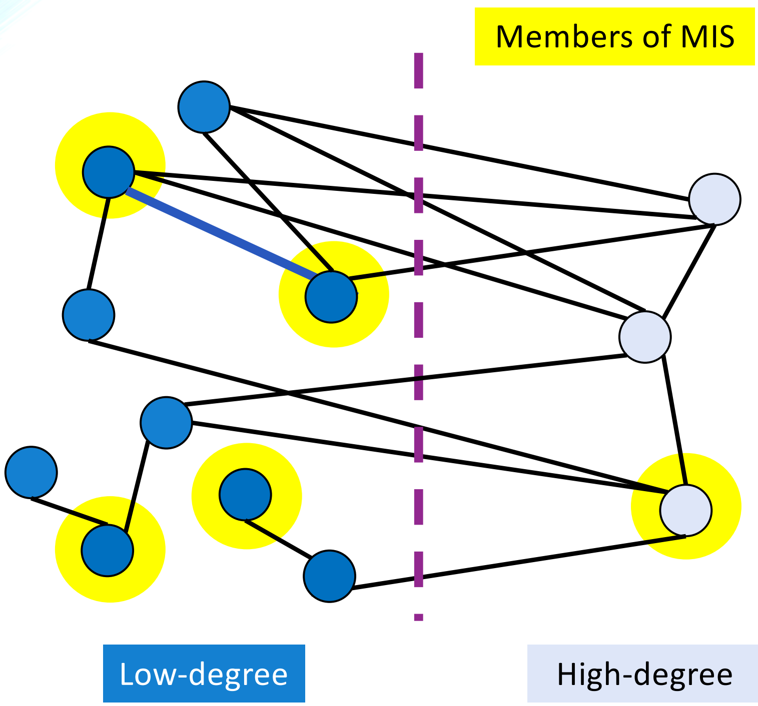
- **Algorithm:**

Distributed Dynamic MIS



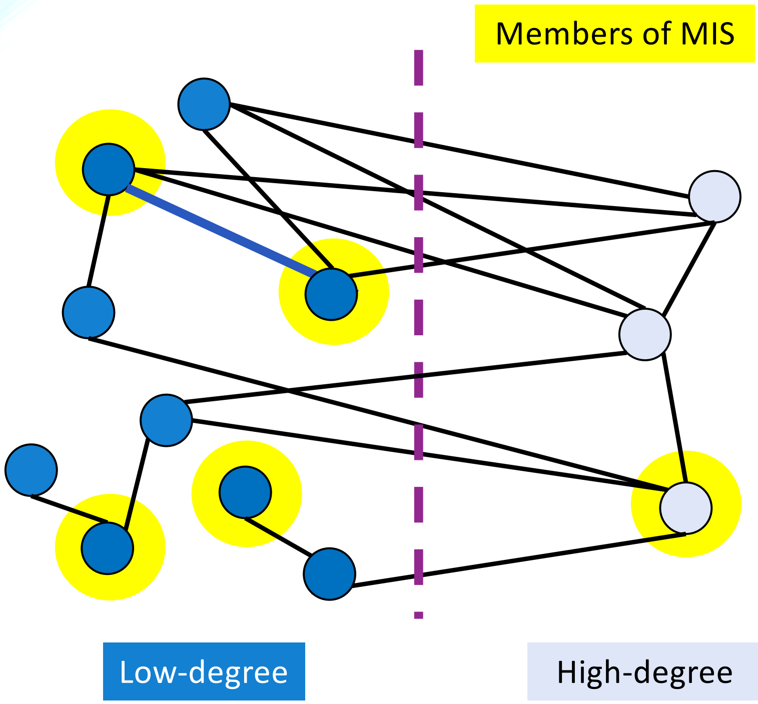
- **Algorithm:**
  - Low-degree vertices prioritize in MIS

Distributed Dynamic MIS



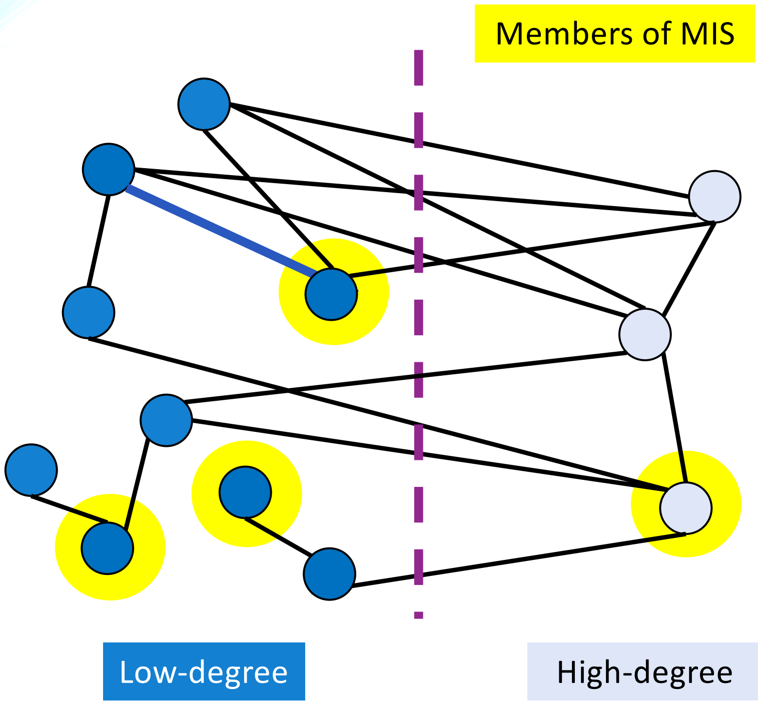
- **Algorithm:**
  - Low-degree vertices prioritize in MIS
  - On **edge insertion:**

Distributed Dynamic MIS



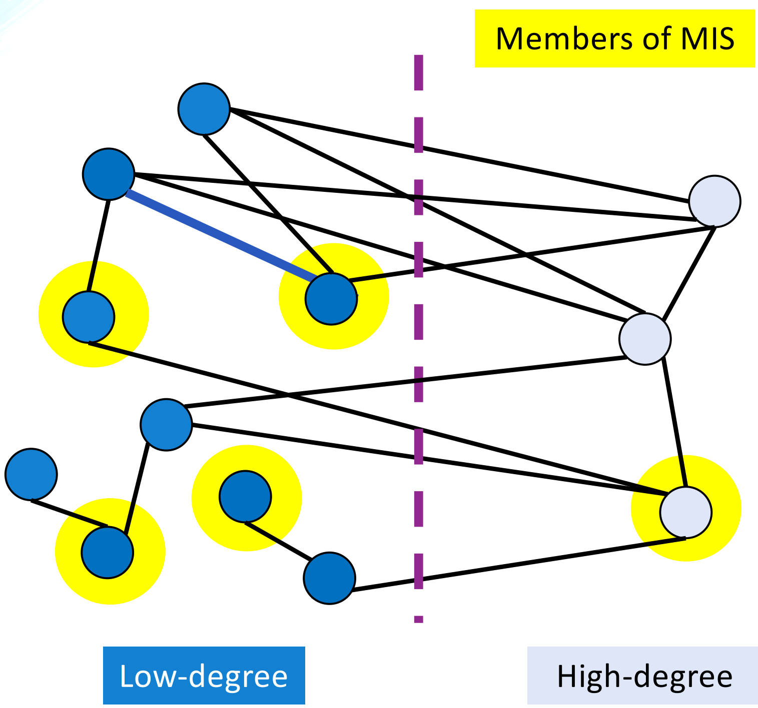
- **Algorithm:**
  - Low-degree vertices prioritize in MIS
  - On **edge insertion:**
    - Remove vertices from MIS if needed

Distributed Dynamic MIS



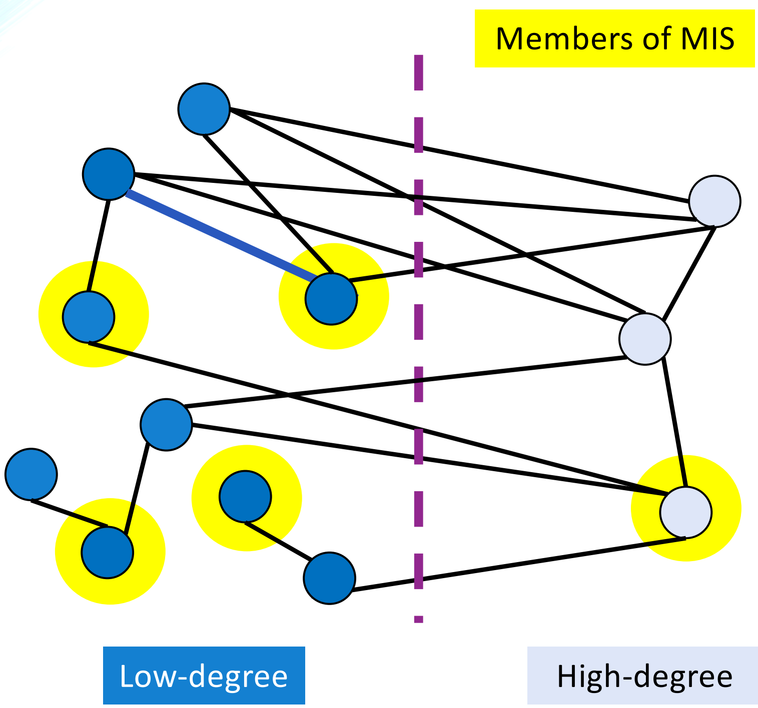
- **Algorithm:**
  - Low-degree vertices prioritize in MIS
  - On **edge insertion:**
    - Remove vertices from MIS if needed

Distributed Dynamic MIS



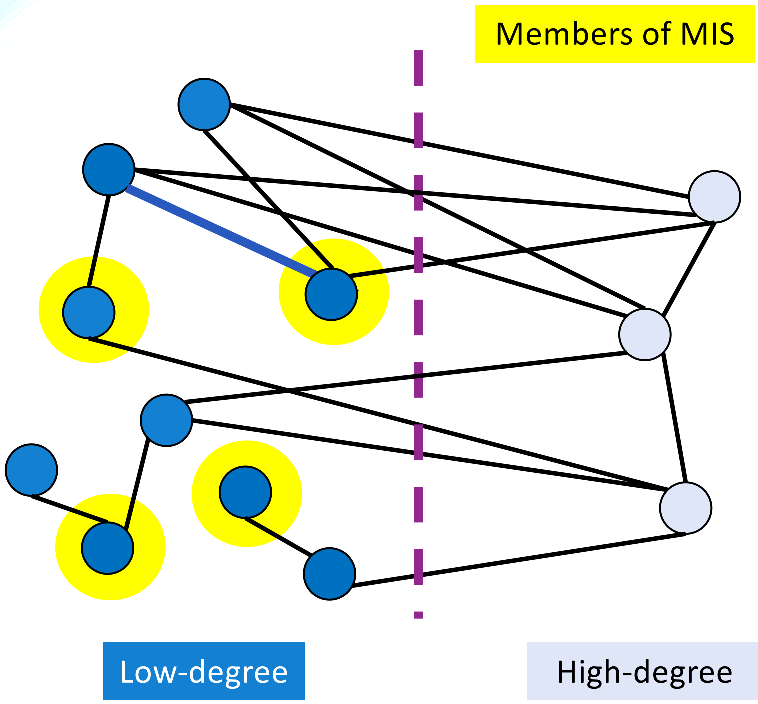
- **Algorithm:**
  - Low-degree vertices prioritize in MIS
  - On **edge insertion:**
    - Remove vertices from MIS if needed
    - Add neighbors into MIS if possible, prioritizing low-degree, then high-degree, potentially many

Distributed Dynamic MIS



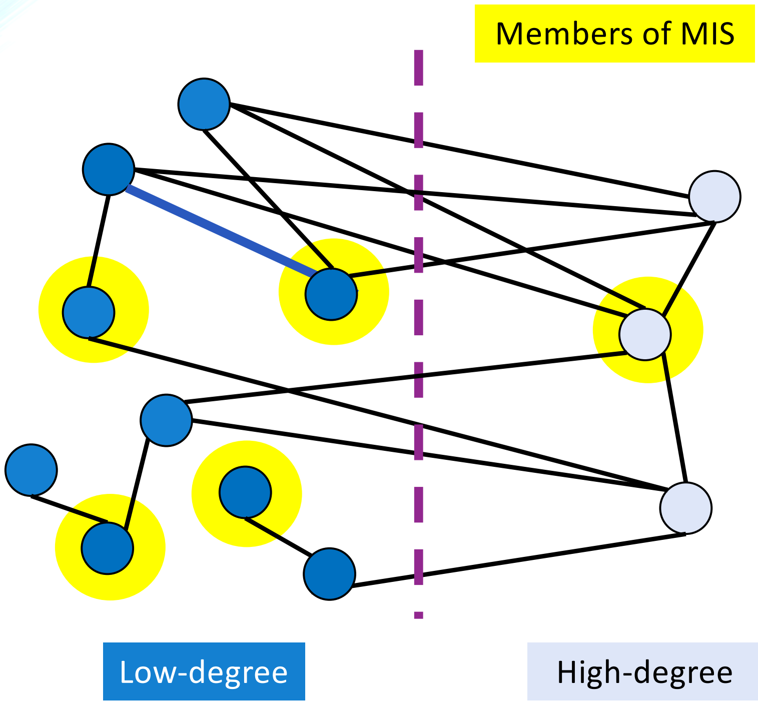
- **Algorithm:**
  - Low-degree vertices prioritize in MIS
  - On **edge insertion:**
    - Remove vertices from MIS if needed
    - Add neighbors into MIS if possible, prioritizing low-degree, then high-degree, potentially many

Distributed Dynamic MIS



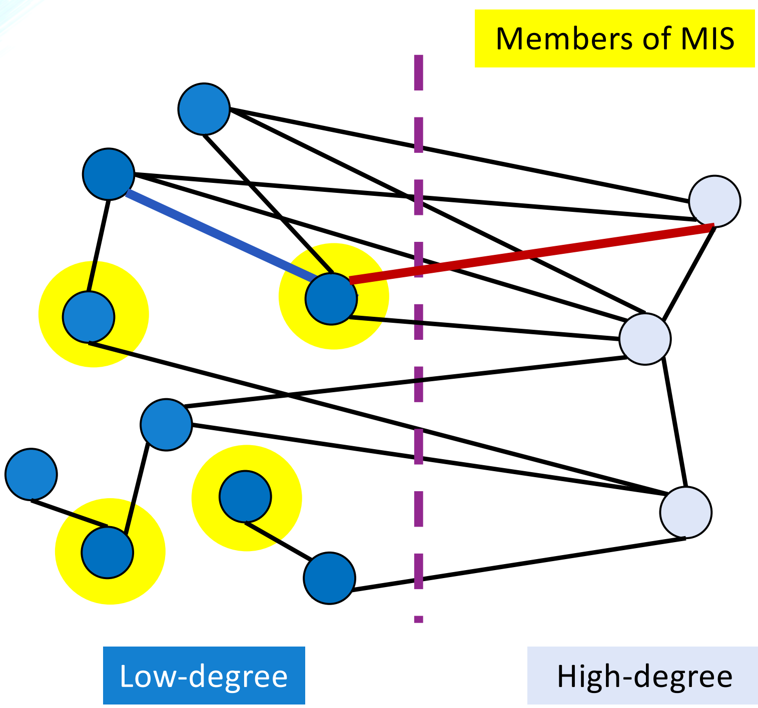
- **Algorithm:**
  - Low-degree vertices prioritize in MIS
  - On **edge insertion:**
    - Remove vertices from MIS if needed
    - Add neighbors into MIS if possible, prioritizing low-degree, then high-degree, potentially many

Distributed Dynamic MIS



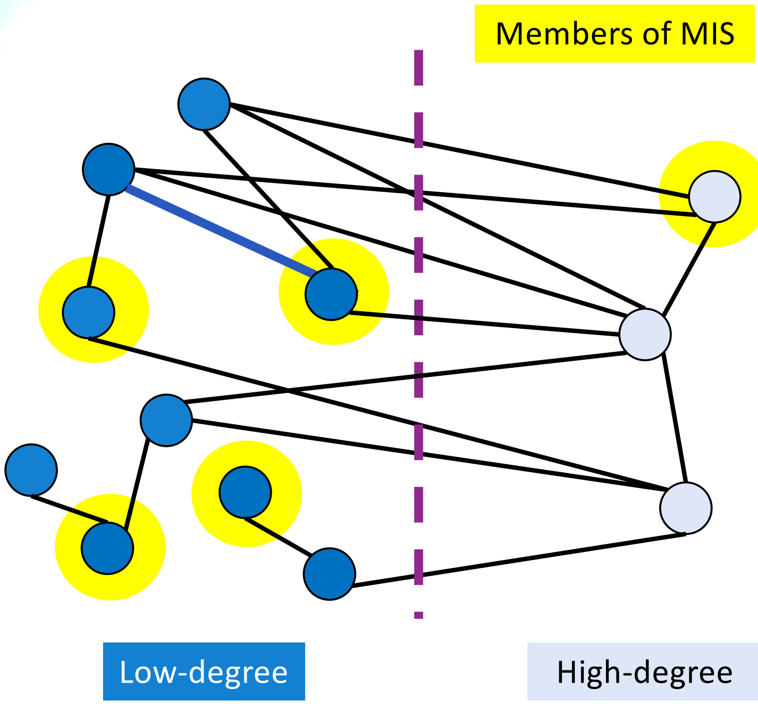
- **Algorithm:**
  - Low-degree vertices prioritize in MIS
  - On **edge insertion:**
    - Remove vertices from MIS if needed
    - Add neighbors into MIS if possible, prioritizing low-degree, then high-degree, potentially many

Distributed Dynamic MIS



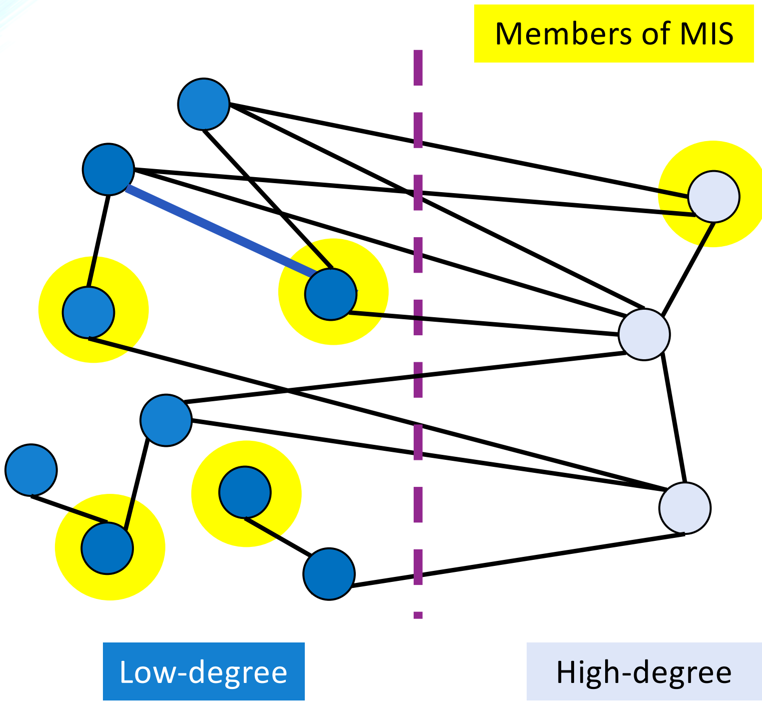
- **Algorithm:**
  - Low-degree vertices prioritize in MIS
  - On **edge deletion:**

Distributed Dynamic MIS



- **Algorithm:**
  - Low-degree vertices prioritize in MIS
  - On **edge deletion:**
    - Prioritize low-degree nodes, then high-degree
    - Add additional nodes to MIS if possible

Distributed Dynamic MIS



- **Algorithm:**
  - Low-degree vertices prioritize in MIS
  - On **edge deletion:**
    - Prioritize low-degree nodes, then high-degree
    - Add additional nodes to MIS if possible
    - Potentially many high-degree nodes added/removed

Distributed Dynamic MIS

Important Notes:

- Edge insertion/deletion could cause nodes to switch low/high-degree

Distributed Dynamic MIS

Important Notes:

- Edge insertion/deletion could cause nodes to switch low/high-degree
- Edge insertion/deletion could cause low degree node to enter or leave MIS

Distributed Dynamic MIS

Important Notes:

- Edge insertion/deletion could cause nodes to switch low/high-degree
- Edge insertion/deletion could cause low-degree node to enter or leave MIS
- Run [AOSS18] on the low-degree node to determine set of low-degree neighbors add to MIS; edge insertion removes at most 1 low-degree node

Distributed Dynamic MIS

Important Notes:

- Edge insertion/deletion could cause nodes to switch low/high-degree
- Edge insertion/deletion could cause low-degree node to enter or leave MIS
- Run [AOSS18] on the low-degree node to determine set of low-degree neighbors add to MIS; edge insertion removes at most 1 low-degree node
- Low-degree nodes entering MIS can cause many high-degree nodes leave

Distributed Dynamic MIS

Important Notes:

- Edge insertion/deletion could cause nodes to switch low/high-degree
- Edge insertion/deletion could cause low-degree node to enter or leave MIS
- Run [AOSS18] on the low-degree node to determine set of low-degree neighbors add to MIS; edge insertion removes at most 1 low-degree node
- Low-degree nodes entering MIS can cause many high-degree nodes leave
- Low-degree nodes leaving MIS can cause many high-degree nodes to enter

Distributed Dynamic MIS

Important Notes:

- Edge insertion/deletion could cause nodes to switch low/high-degree
- Edge insertion/deletion could cause low-degree node to enter or leave MIS
- Run [AOSS18] on the low-degree node to determine set of low-degree neighbors add to MIS; edge insertion removes at most 1 low-degree node
- Low-degree nodes entering MIS can cause many high-degree nodes leave
- Low-degree nodes leaving MIS can cause many high-degree nodes to enter
- [GK21] **global restart all** high-degree nodes to determine set of high-degree nodes that enter/leave at every step

Distributed Dynamic MIS

Important Notes:

- Edge insertion/deletion could cause nodes to switch low/high-degree
- Edge insertion/deletion could cause low degree node to enter or leave MIS
- Run [AOSS18] on the low-degree node to determine set of low-degree neighbors add to MIS; edge insertion removes at most 1 low-degree node
- Low-degree nodes entering MIS can cause many high-degree nodes leave
- Low-degree nodes leaving MIS can cause many high-degree nodes to enter
- [GK21] **global restart all** high-degree nodes to determine set of high-degree nodes that enter/leave at every step
- **Instead, do high-degree restart in local neighborhood only when needed**

Distributed Dynamic MIS

Important Notes:

- Edge insertion/deletion could cause nodes to switch low/high-degree
- Edge insertion/deletion could cause low degree nodes to enter or leave MIS

But need to know which vertices are low-degree and high-degree (unknown m and potentially disconnected graph)**!**

- Low-degree nodes entering MIS can cause many high-degree nodes to le
- Low-degree nodes leaving MIS can cause many high-degree nodes to enter

Challenge 1: How do nodes determine if they're **high-degree/low-degree as m changes** with updates (for unknown m)?

- **Instead, we use high-degree nodes in local neighborhood (details later)**

Distributed Dynamic MIS

Important Notes:

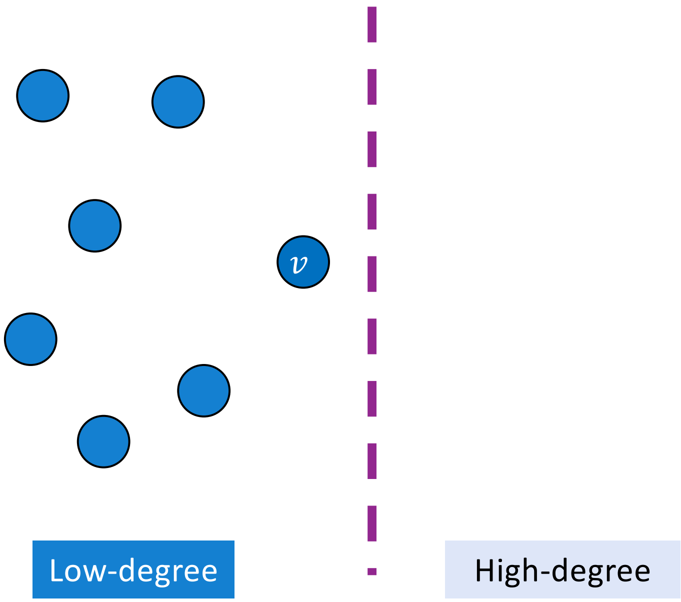
- Edge insertion/deletion could cause low degree nodes to enter or leave MIS
- Run [AOSS18] on the low-degree node to determine set of low-degree

- leave

Challenge 2: How do high-degree nodes find maximal independent set in **small number of rounds** and **few messages**?

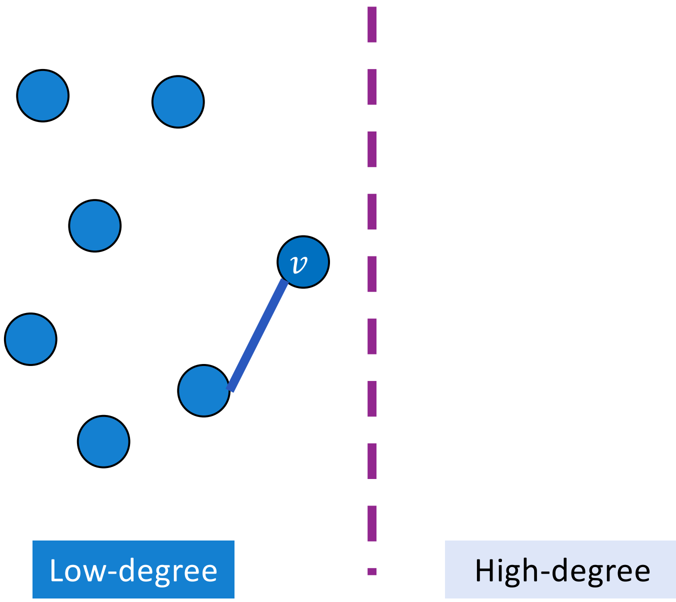
- Low-degree nodes leaving MIS can cause many high-degree nodes to enter
- [GK21] **global restart all** high-degree nodes to determine set of high-degree nodes that enter/leave at every step
- **Instead, do high-degree restart in local neighborhood only when needed**

Distributed Dynamic MIS



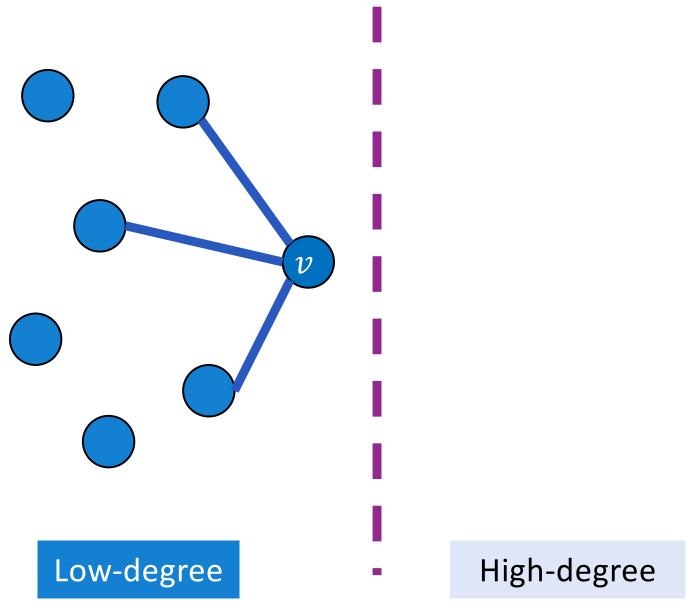
- **Solving Challenge 1:** how to determine low/high-degree
  - Initialize counter $p_v \leftarrow 1$

Distributed Dynamic MIS



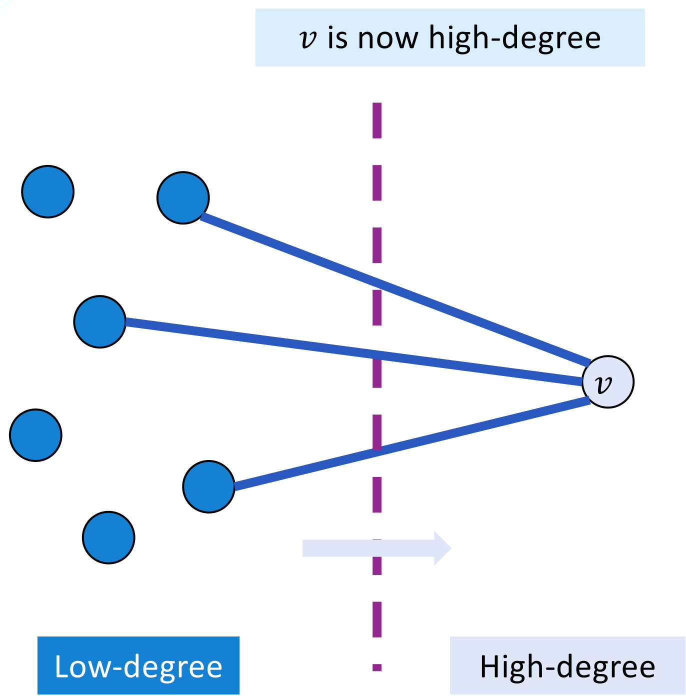
- **Solving Challenge 1:** how to determine low/high-degree
  - Initialize counter $p_v \leftarrow 1$
  - All vertices initially low-degree
  - On edge insertions where degree **exceeds** $2p_v$

Distributed Dynamic MIS



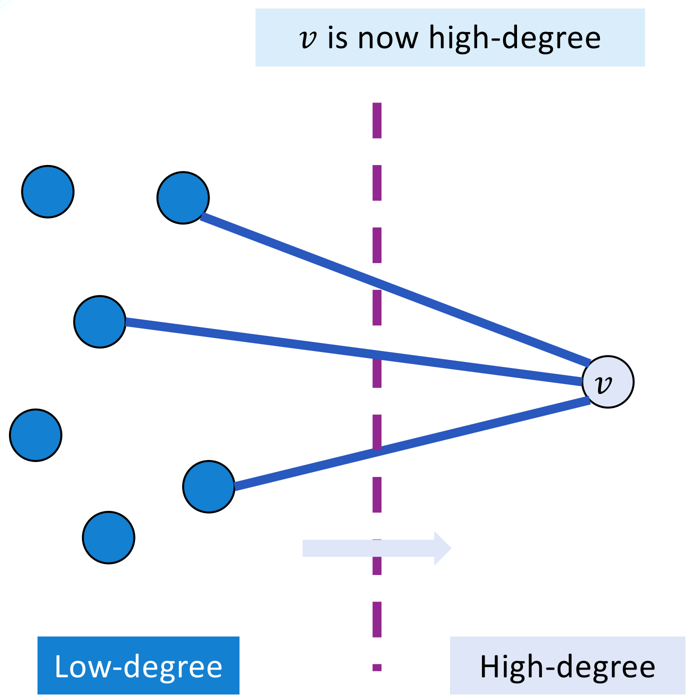
- **Solving Challenge 1:** how to determine low/high-degree
  - Initialize counter $p_v \leftarrow 1$
  - All vertices initially low-degree
  - On edge insertions where degree **exceeds** $2p_v$

Distributed Dynamic MIS



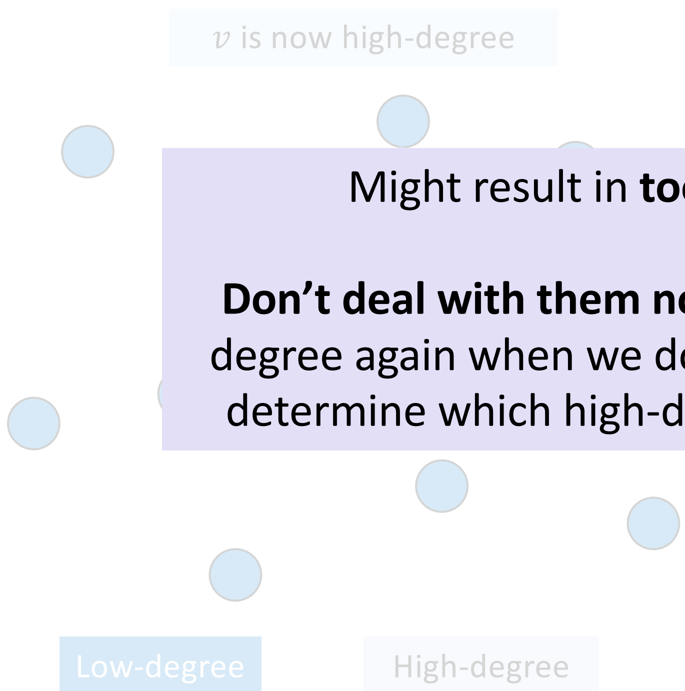
- **Solving Challenge 1:** how to determine low/high-degree
  - Initialize counter $p_v \leftarrow 1$
  - All vertices initially low-degree
  - On edge insertions where degree **exceeds** $2p_v$
  - Make node high-degree

Distributed Dynamic MIS



- **Solving Challenge 1:** how to determine low/high-degree
  - Initialize counter $p_v \leftarrow 1$
  - All vertices initially low-degree
  - On edge insertions where degree **exceeds** $2p_v$
  - Make node high-degree
  - p_v updates to the **current degree** when low-degree

Distributed Dynamic MIS

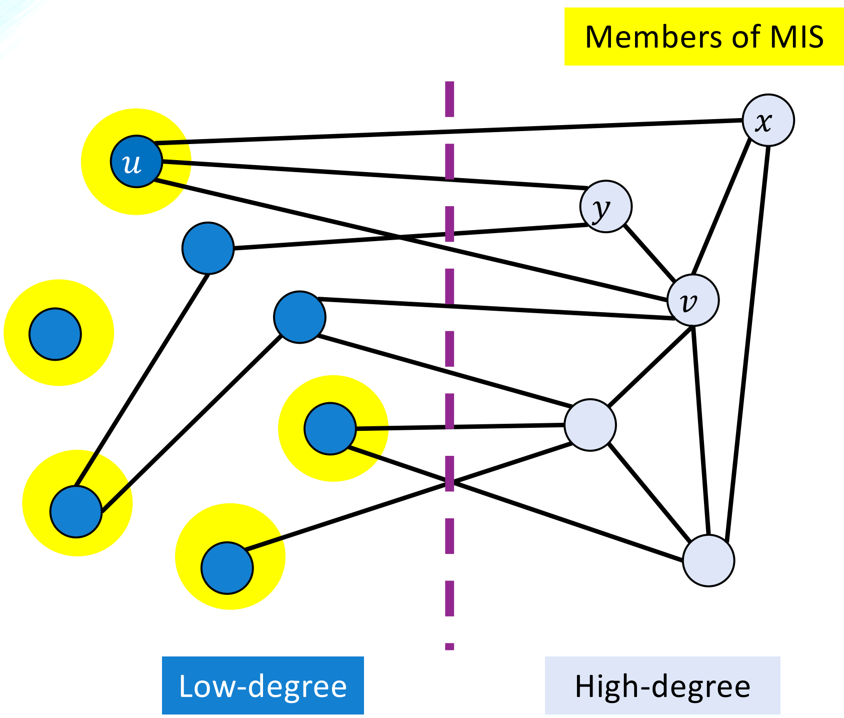


Might result in **too many high-degree nodes**

Don’t deal with them now, make high-degree nodes low-degree again when we do a **local restart** (when we need to determine which high-degree nodes need to go into MIS)

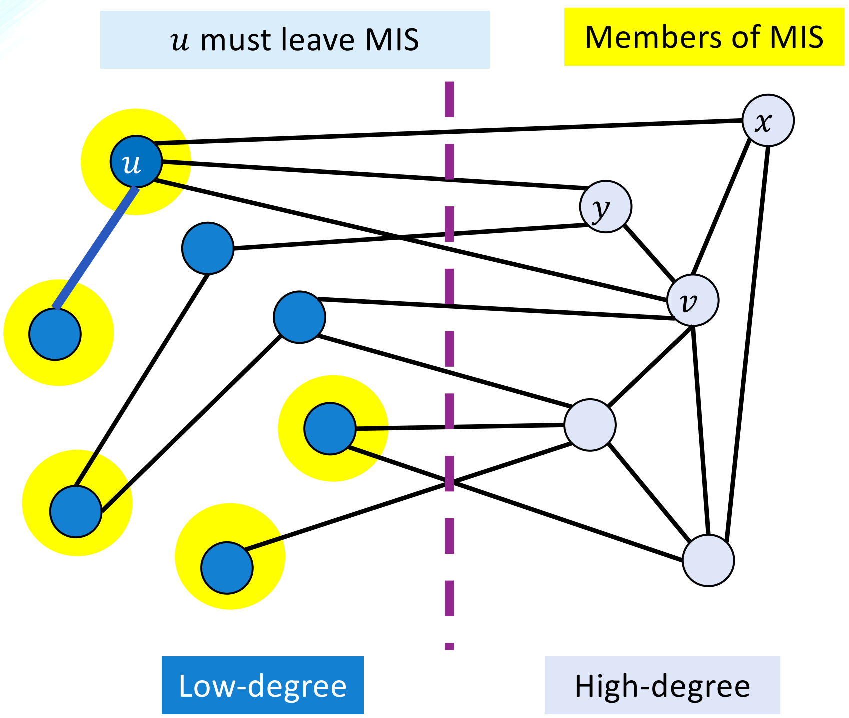
- **Solving Challenge 1:** how to determine low/high-
  - On edge insertions where degree **exceeds** $2p_v$
  - Make node high-degree

Distributed Dynamic MIS



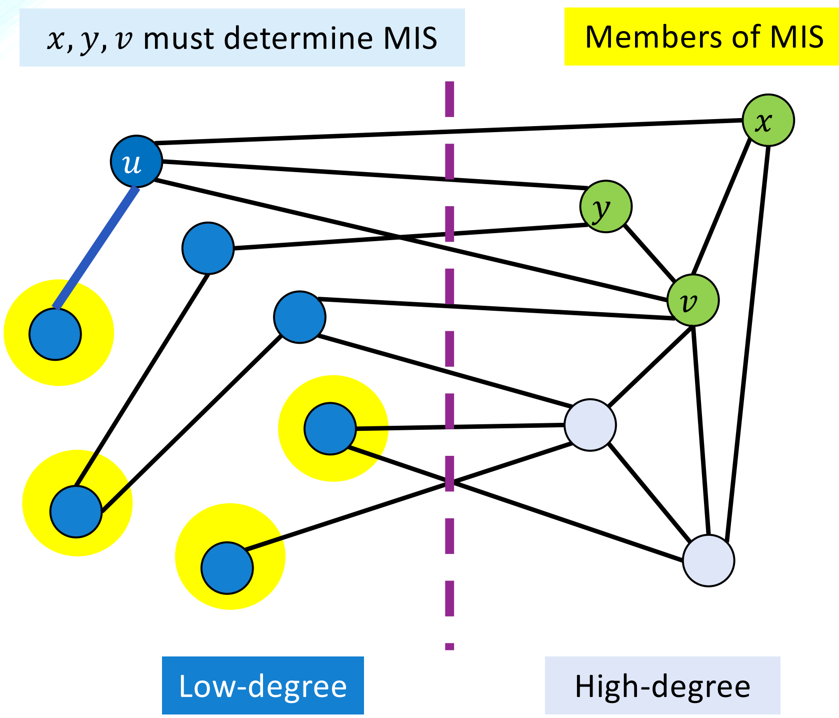
- **Solving Challenge 2:** how to determine MIS among high-degree neighbors

Distributed Dynamic MIS



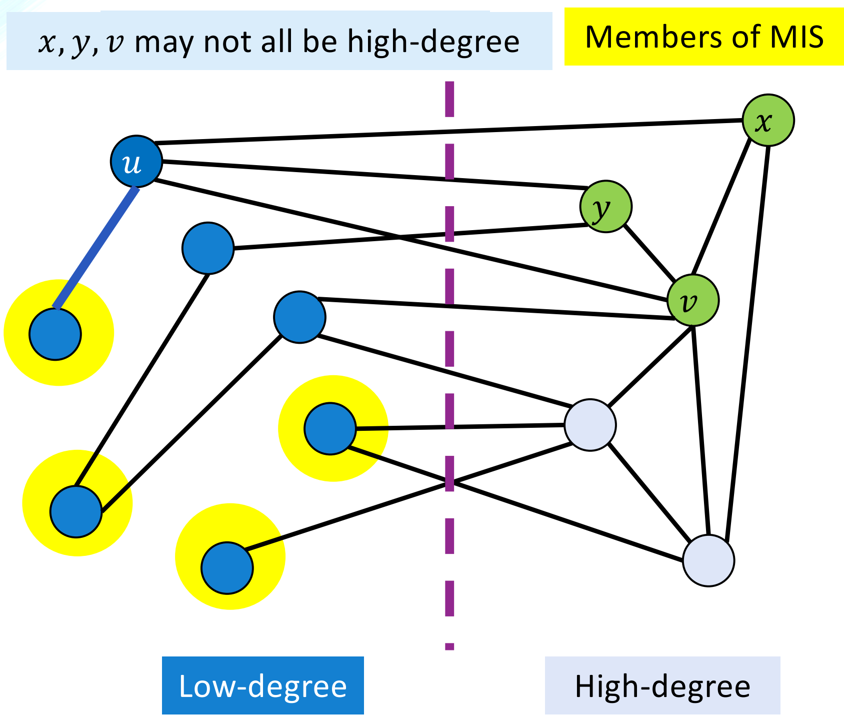
- **Solving Challenge 2:** how to determine MIS among high-degree neighbors
  - On edge insertion, when a low-degree neighbor leaves the MIS:

Distributed Dynamic MIS



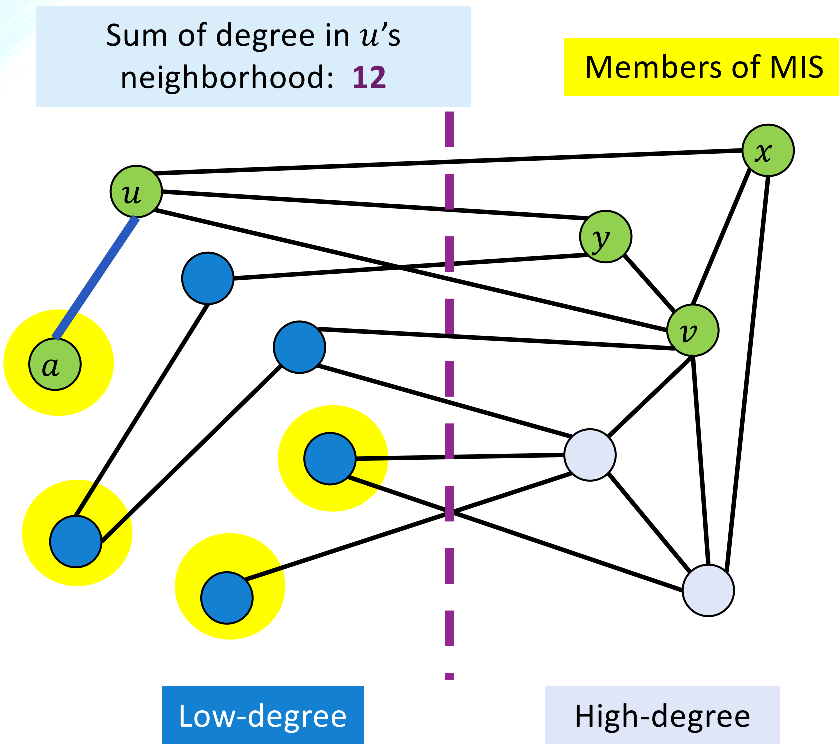
- **Solving Challenge 2:** how to determine MIS among high-degree neighbors
  - On edge insertion, when a low-degree neighbor leaves the MIS:
    - High-degree nodes must determine MIS in induced neighborhood

Distributed Dynamic MIS



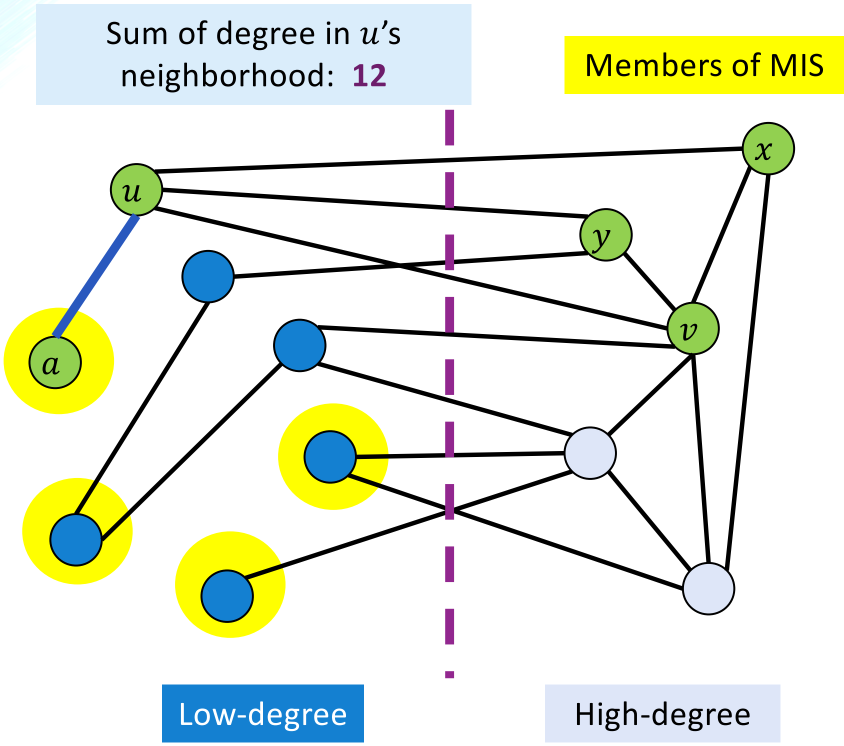
- **Solving Challenge 2:** how to determine MIS among high-degree neighbors
  - On edge insertion, when a low-degree neighbor leaves the MIS:
    - High-degree nodes must determine MIS in induced neighborhood
  - First solve **Challenge 1** again – Some are low-degree

Distributed Dynamic MIS



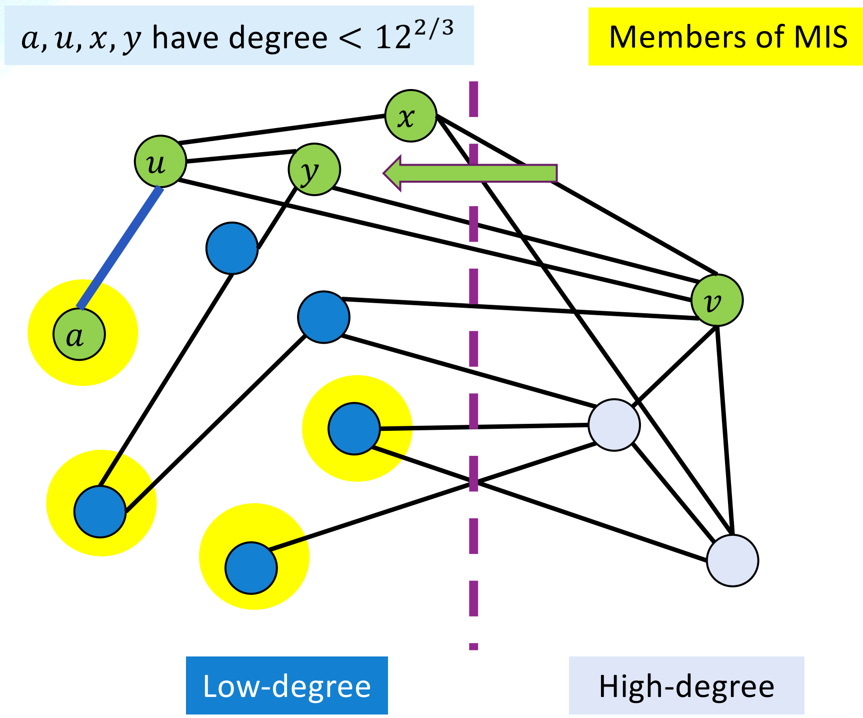
- First solve **Challenge 1** again
  - Some are low-degree
    - Determine sum of degree in 1-hop neighborhood (S) of low-degree node

Distributed Dynamic MIS



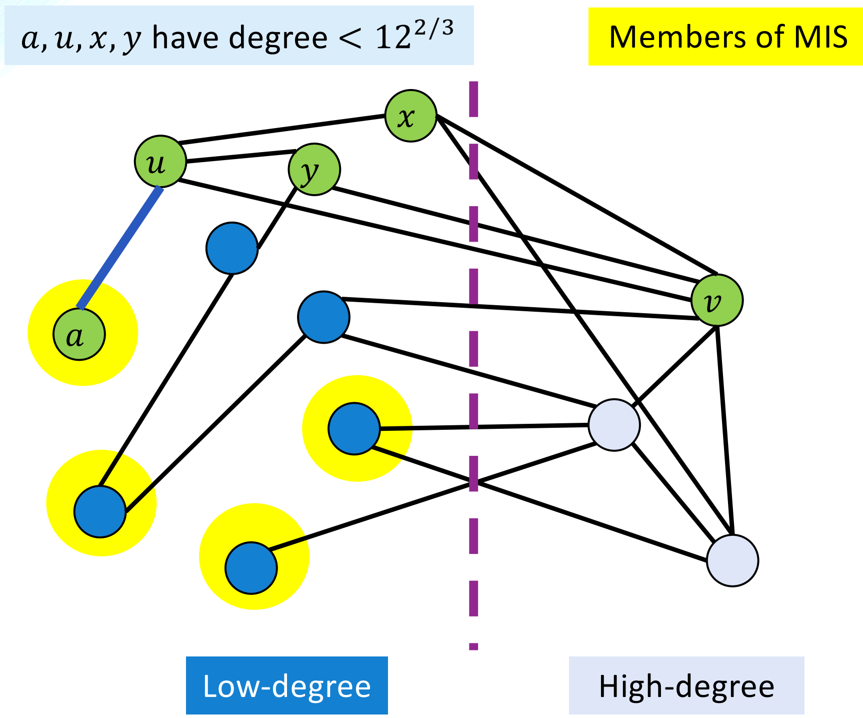
- First solve **Challenge 1** again
  - Some are low-degree
    - Determine sum of degree in 1-hop neighborhood (S) of low-degree node
    - Any vertex with degree < $S^{2/3}$ becomes low-degree

Distributed Dynamic MIS



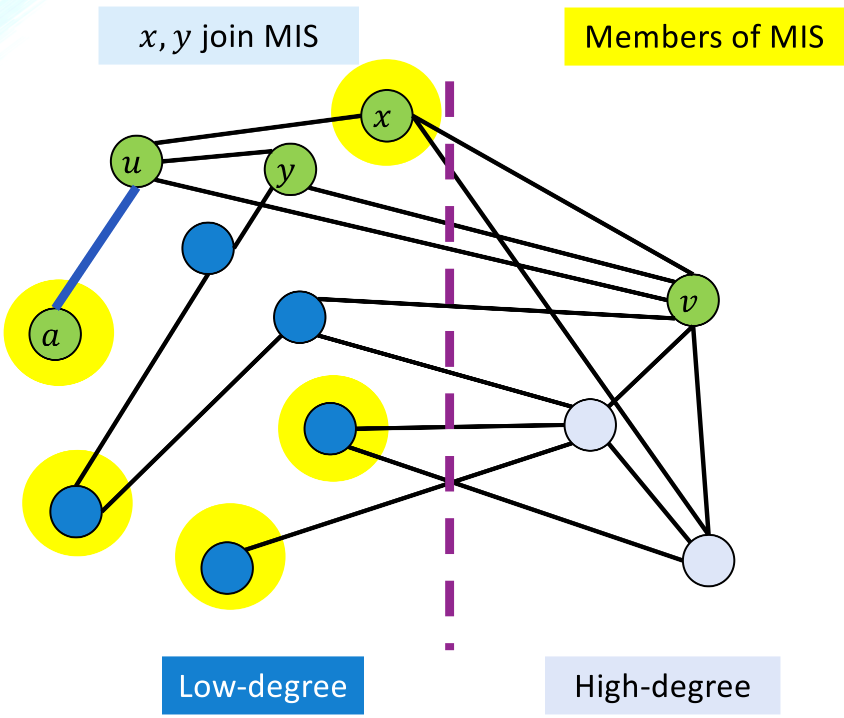
- First solve **Challenge 1** again
  - Some are low-degree
    - Determine sum of degree in 1-hop neighborhood (S) of low-degree node
    - Any vertex with degree $< S^{2/3}$ becomes low-degree

Distributed Dynamic MIS



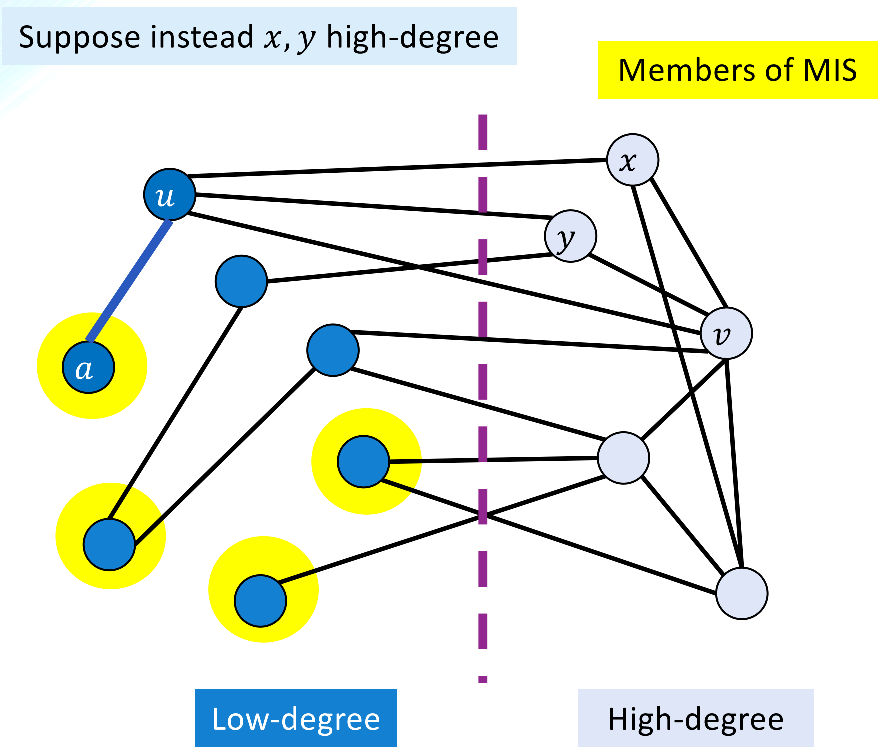
- First solve **Challenge 1** again
  – Some are low-degree
    - Determine sum of degree in 1-hop neighborhood (S) of low-degree node
    - Any vertex with degree $< S^{2/3}$ becomes low-degree
    - Vertices which became low-degree, priority in joining MIS

Distributed Dynamic MIS



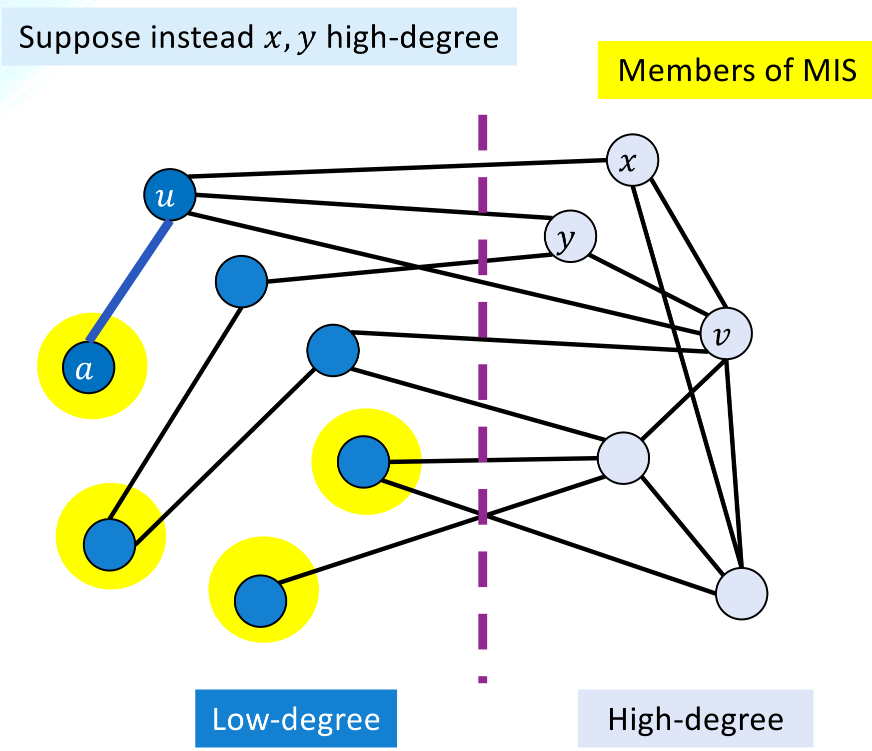
- First solve **Challenge 1** again
  – Some are low-degree
    - Determine sum of degree in 1-hop neighborhood (S) of low-degree node
    - Any vertex with degree $< S^{2/3}$ becomes low-degree
    - Vertices which became low-degree, priority in joining MIS

Distributed Dynamic MIS



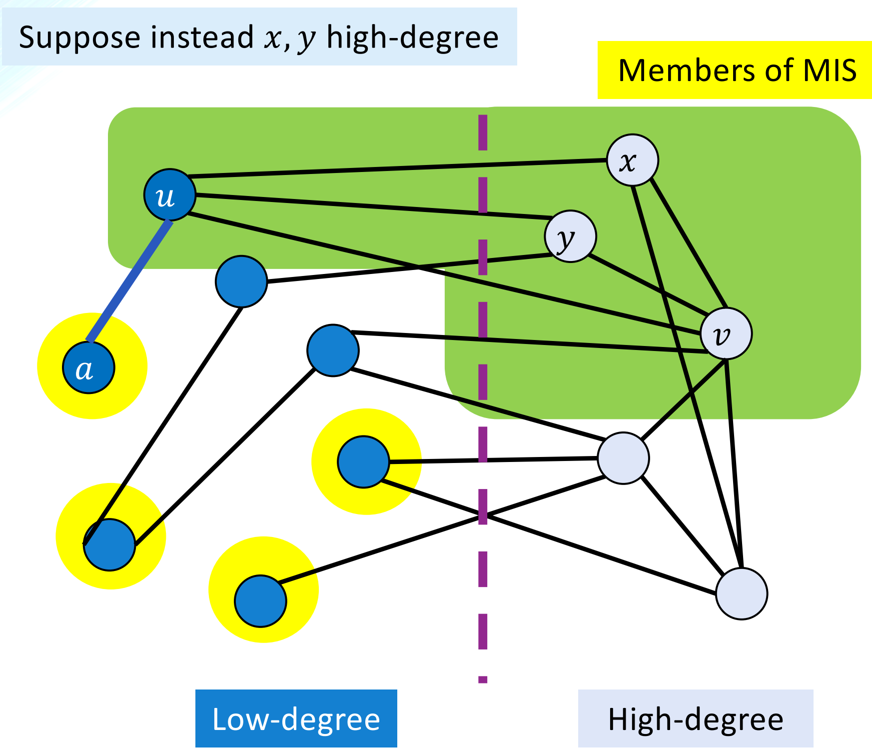
- **Finish Solving Challenge 2:**

Distributed Dynamic MIS



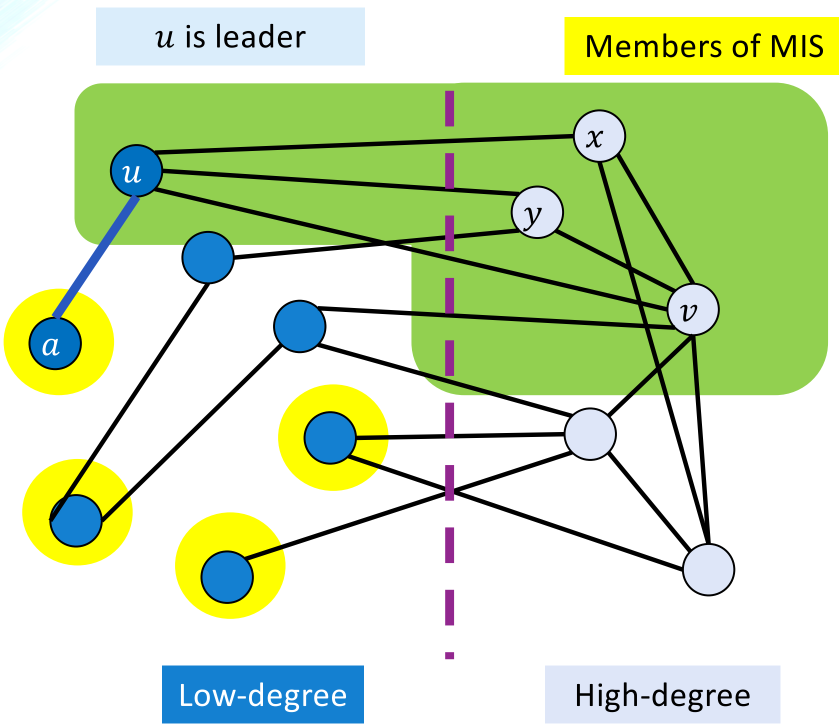
- **Finish Solving Challenge 2:**
  - Run static, distributed MIS algorithm on induced subgraph of high-degree nodes in local neighborhood

Distributed Dynamic MIS



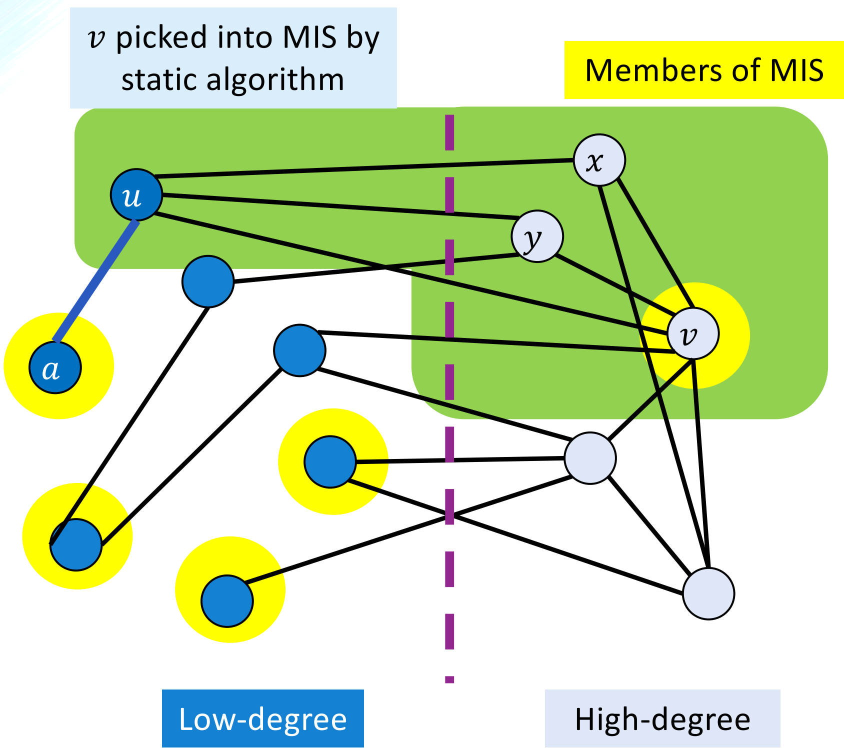
- **Finish Solving Challenge 2:**
  - Run static, distributed MIS algorithm on induced subgraph of high-degree nodes in local neighborhood
  - Run the small diameter algorithm of Censor-Hillel, Parter, and Schwartzman (2020) on induced subgraph

Distributed Dynamic MIS



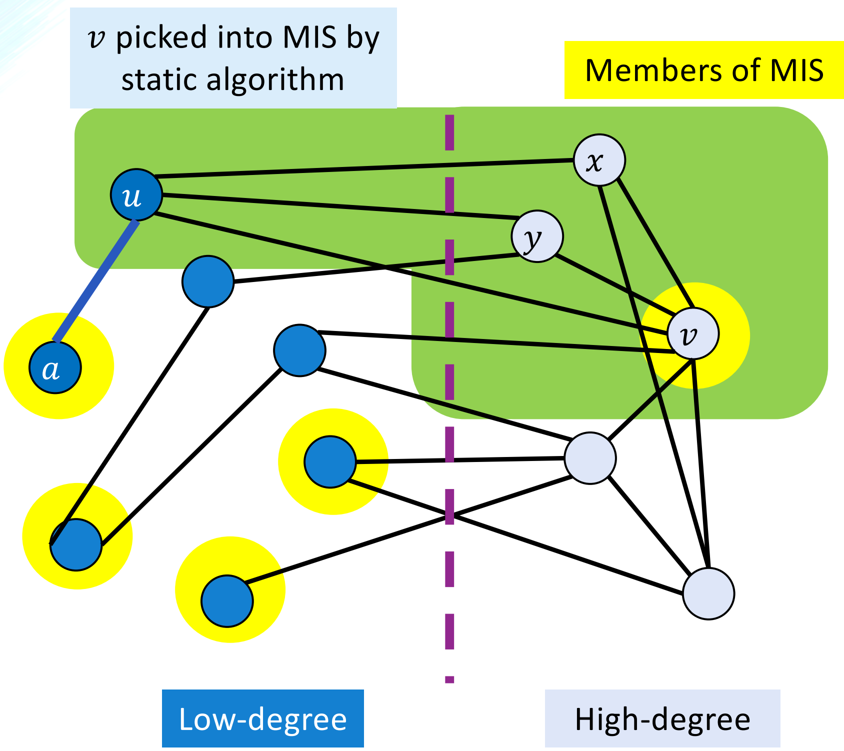
- **Finish Solving Challenge 2:**
  - Run static, distributed MIS algorithm on induced subgraph of high-degree nodes in local neighborhood
  - Run the small diameter algorithm of Censor-Hillel, Parter, and Schwartzman (2020) on induced subgraph

Distributed Dynamic MIS



- **Finish Solving Challenge 2:**
  - Run static, distributed MIS algorithm on induced subgraph of high-degree nodes in local neighborhood
  - Run the small diameter algorithm of Censor-Hillel, Parter, and Schwartzman (2020) on induced subgraph

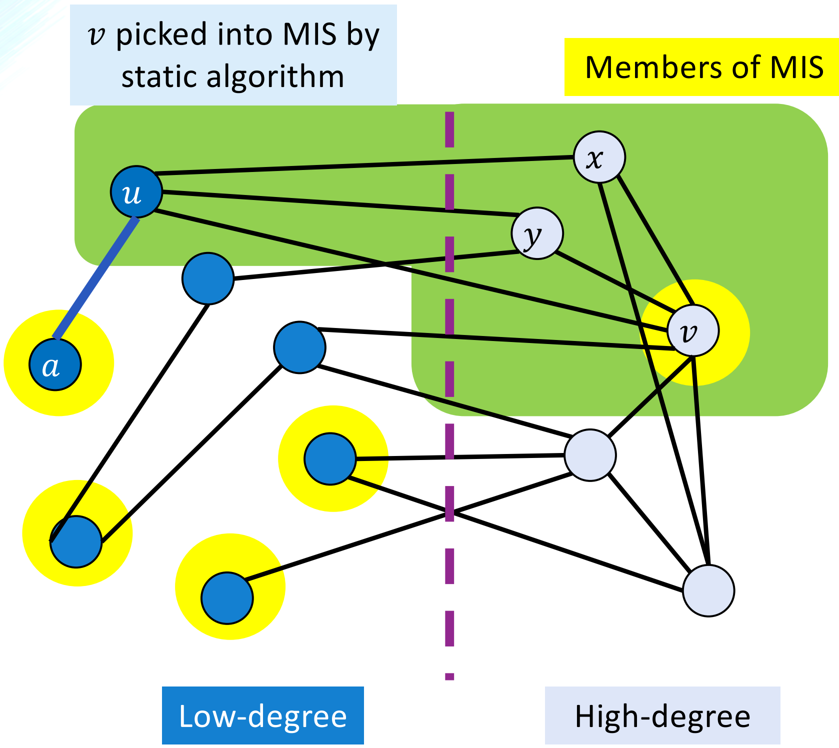
Distributed Dynamic MIS



- **Overall complexity:**
  - **Message complexity:**
    - $O(m^{2/3} \log^2 n)$ amortized

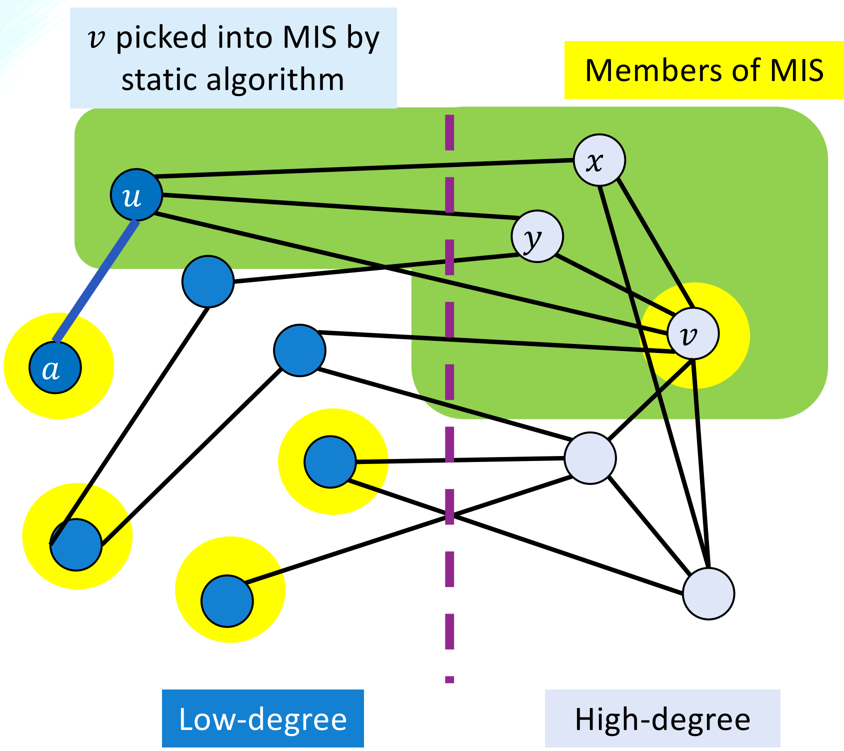
Distributed Dynamic MIS

m is average number of edges over all updates



- **Overall complexity:**
  - **Message complexity:**
    - $O(m^{2/3} \log^2 n)$ amortized

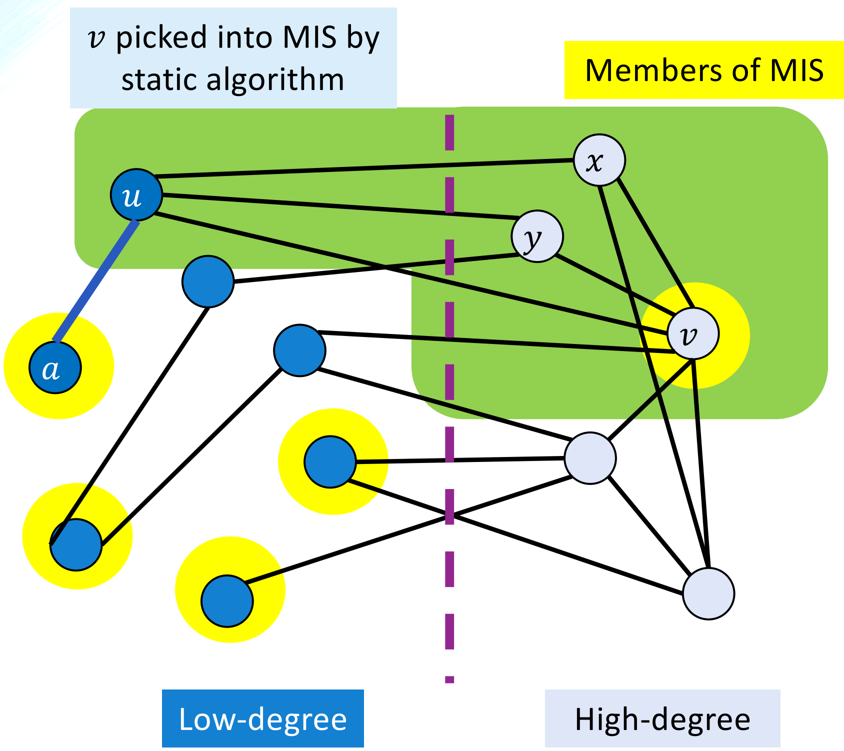
Distributed Dynamic MIS



m is average number of edges over all updates

- **Overall complexity:**
  - **Message complexity:**
    - $O\left(m^{2/3}\log^2 n\right)$ amortized
    - Low-degree vertices have degree $O\left(m^{2/3}\right)$

Distributed Dynamic MIS

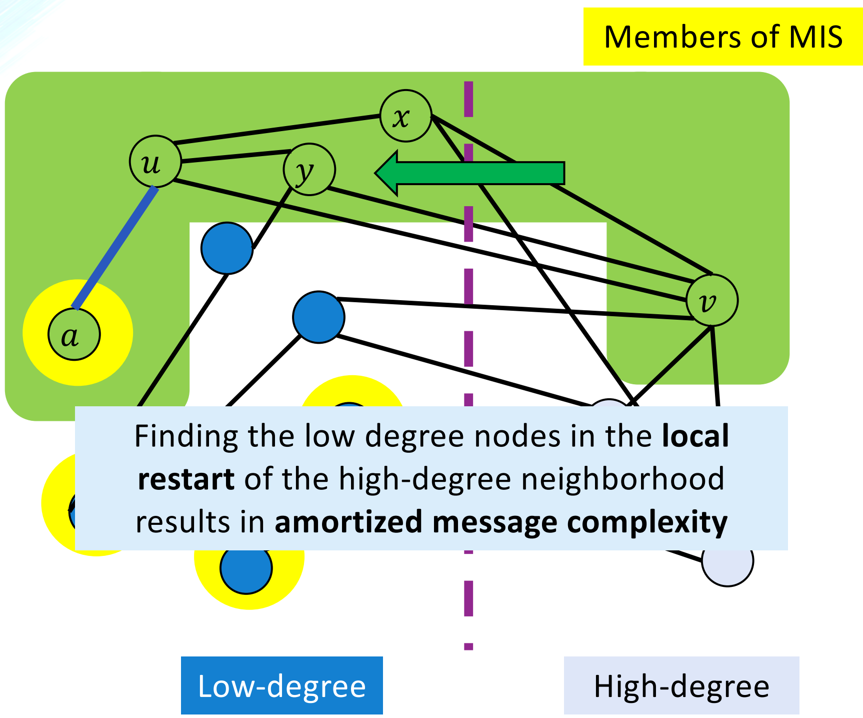


m is average number of edges over all updates

- **Overall complexity:**
  - **Message complexity:**
    - $O(m^{2/3} \log^2 n)$ amortized
    - Low-degree vertices have degree $O(m^{2/3})$
    - At most $O(m^{2/3})$ edges in local high-degree neighborhood

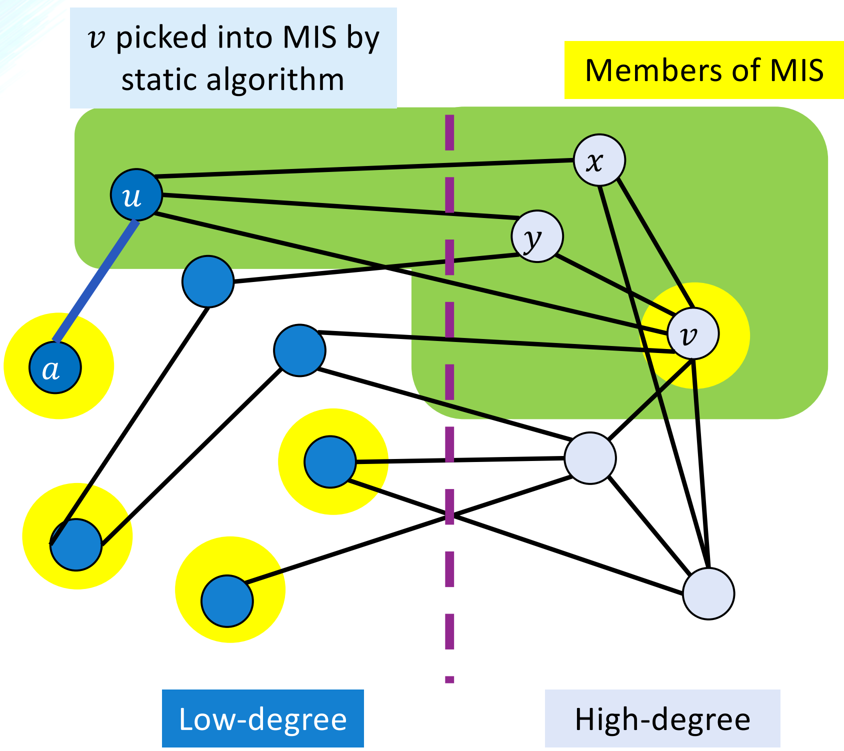
Distributed Dynamic MIS

m is average number of edges over all updates



- **Overall complexity:**
  - **Message complexity:**
    - $O\left(m^{2/3} \log^2 n\right)$ amortized
    - Low-degree vertices have degree $O\left(m^{2/3}\right)$
    - At most $O\left(m^{2/3}\right)$ edges in local high-degree neighborhood
    - Amortization due to local restarts

Distributed Dynamic MIS

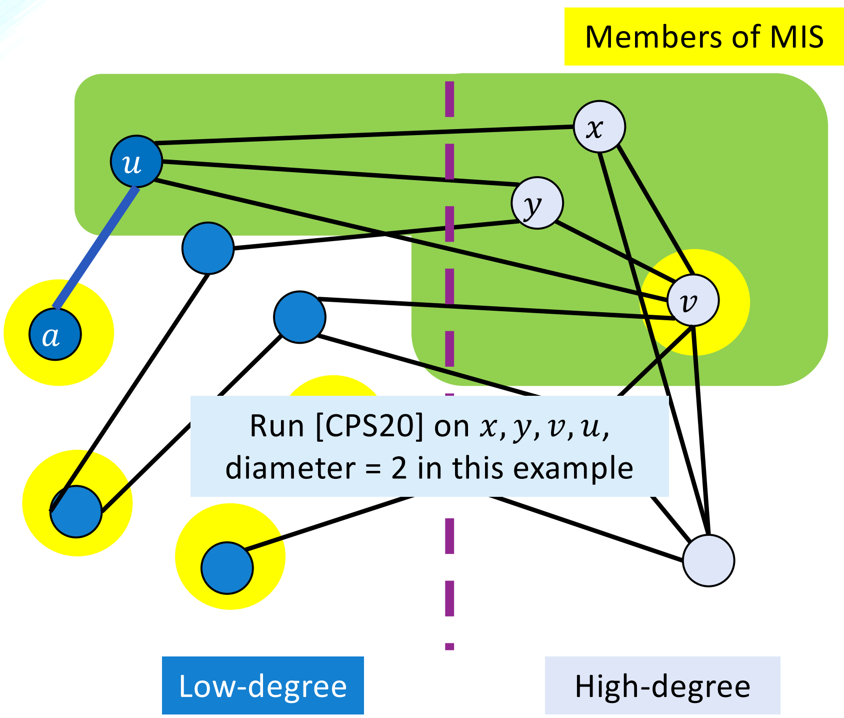


m is average number of edges over all updates

- **Overall complexity:**
  - **Round complexity:**
    - $O(\log^2 n)$ amortized
    - Running [CPS20] requires $O(\log^2 n)$ rounds for **constant** diameter graphs

Distributed Dynamic MIS

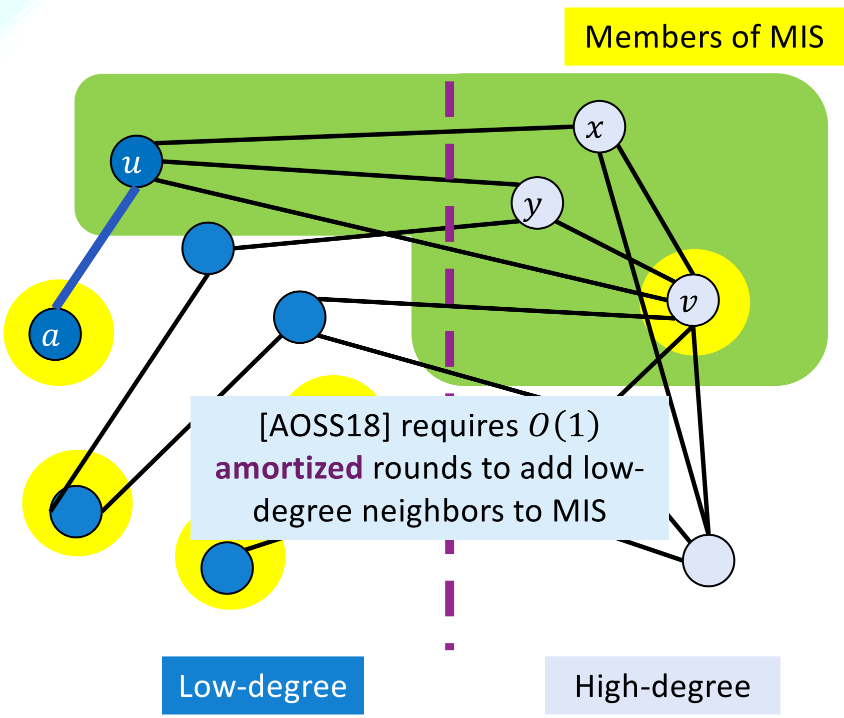
m is average number of edges over all updates



- **Overall complexity:**
  - **Round complexity:**
    - $O(\log^2 n)$ amortized
    - Running [CPS20] requires $O(\log^2 n)$ rounds for **constant** diameter graphs
    - We run the algorithm on local subgraphs with diameter at most 6

Distributed Dynamic MIS

m is average number of edges over all updates



- **Overall complexity:**
  - **Round complexity:**
    - $O(\log^2 n)$ amortized
    - Running [CPS20] requires $O(\log^2 n)$ rounds for **constant** diameter graphs
    - We run the algorithm on local subgraphs with diameter at most 6
    - Amortization from running [AOSS18] to add low-neighbors to MIS

Conclusion and Open Questions

- Initialize formal study of message-efficient dynamic algorithms in distributed networks

Grand Prize:

- **message complexity matches update time** of best-known sequential, centralized algorithm
- **round complexity is $O(1)$**

- Achieved for several fundamental symmetry breaking problems (up to $O(\log^2 n)$ factors for MIS, and smaller for other problems)
- Solve several general challenges—unknown m and global restarts

Conclusion and Open Questions

- Initialize formal study of message-efficient dynamic algorithms in distributed networks

Grand Prize:

- **message complexity matches update time** of best-known sequential, centralized algorithm
- **round complexity is $\boldsymbol{O(1)}$**

- Achieved for several fundamental symmetry breaking problems (up to $O(\log^2 n)$ factors for MIS, and smaller for other problems)
- Solve several general challenges—unknown m and global restarts

Question 1: Can our techniques be generalized for a wide class of dynamic distributed algorithms?

Question 2: Can we achieve worst-case bounds (esp. rounds) for MIS?

Question 3: Can we get rid of the $O(\log^2 n)$ factors especially in round complexity of MIS?

Question 4: Can our algorithms be modified to handle multiple concurrent updates, while maintaining low message complexity?

Question 5: Is there a general purpose compiler which takes a centralized dynamic algorithm and outputs a message-efficient distributed dynamic algorithm?