A framework for boosting matching approximation: parallel, distributed, and dynamic

Slobodan Mitrović
(UC Davis)

Wen-Horng Sheu
(UC Davis)

Maximum matching problem

- Let $G = (V, E)$ be an unweighted graph
- Let $n = |V|$, $m = |E|$
- Matching: set of edges that do not share endpoints
- Maximum matching: the matching of maximum size
- c-approximate matching:
  matching of size at least $1/c$ times the maximum

Prior work

- The problem has been extensively studied:
- **Polynomial time:** [Berge ‘57] [Edmonds ‘65] [Hopcroft, Karp ‘73] [Micali, Vazirani ‘80] [Gabow ‘90] [Kalantari, Shokoufandeh ‘95] …
- **Dynamic:** [Bernstein, Stein ‘16] [Solomon ‘16] [Bhattacharya, Kulkarni ‘19] [Behnezhad, Łącki, Mirrokni ‘19] [Behnezhad, Khanna ‘22] …
- **Semi-streaming:** [McGregor ‘05] [Ahn, Guha, '11] [Ahn, Guha, '13] [Kapralov, '13] [Tirodkar, '18] [Assadi, Liu, Tarjan, '21] [Assadi, Jambulapati, Jin, Sidford, Tian, '22] [Fischer, Mitrović, Uitto, '22] [Huang, Su, '23] [Assadi, '24] ...
- **Distributed (CONGEST, MPC):** [Behnezhad, Hajiaghayi, Harris '19] [Ghaffari, Grunau, Jin '20] [Fischer, Mitrović, Uitto, '22]...

(1+ε)-approximate matching

- **Our focus:** Given ε > 0, find (1+ε)-approximation
- **Motivation:** finding exact maximum is inefficient in many settings
- **Approaches:**

  (1) stand-alone algorithm

  (2) boosting framework (reducing to constant approximation)

Boosting framework

- **Input:**
  - graph G
  - parameter ε
  - access to oracle A_{mat} for constant-approximate matching

- **What it does:**
  - Calls A_{mat} on adaptively chosen graphs
  - (May not be subgraphs of G)
  - Find (1+ε)-approximate matching for G

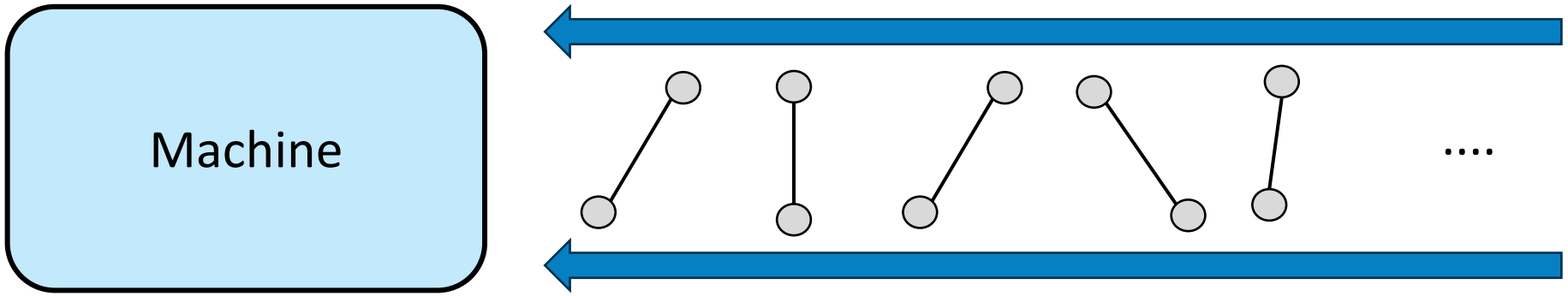
First framework

- By [McGregor '05]
- Was a semi-streaming algorithm
- Later adapted as a framework:
  - MPC [Onak, 2018]
  - Fully dynamic [Bhattacharya et al., 2023]
- **Number of calls to A_{mat} :** $(1/\varepsilon)^{O(1/\varepsilon)}$, independent of n!
- **Message:** (1+ε)-approx. reduces to constant approx. in many settings!

Semi-streaming setting

- No random access to G
- Edges are presented as a stream
- Algorithm can use $\tilde{O}(n)$ memory (sublinear)

- **Goal:** minimize number of passes

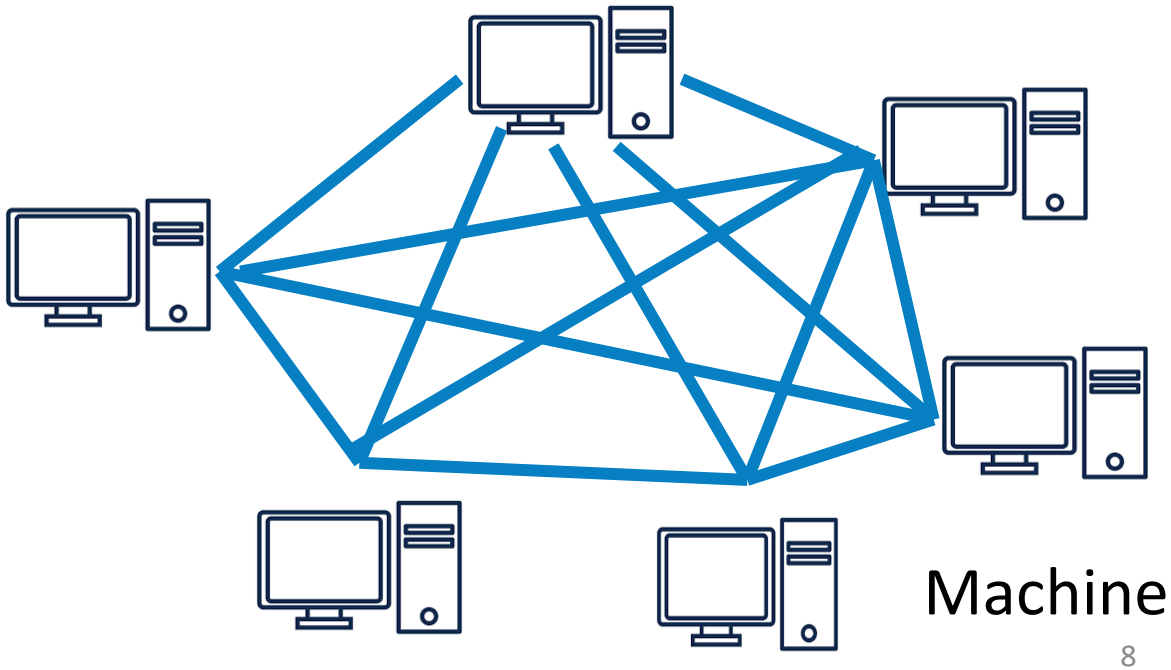


$O(n \text{ poly log } n)$ memory

data stream (edges)

MPC setting (informal)

- Input stored in M machines
- Each machine has $O(n^{\alpha})$ memory, $\alpha < 1$
- Machines communicate in synchronous rounds
- **Goal:** minimize number of rounds



Fully dynamic setting

- **Input:** Empty graph of *n* vertices
  Sequence of edge updates (add or remove edges)

- **Goal:** Maintain a (1+ε)-approximate matching

- **Goal:** minimize update time

- [McG05]'s framework works for all these settings!
- but with exp(1/ε) calls

Recent improvement

1. By [Fischer, Mitrović, Uitto, 2022]
   - Improved semi-streaming algorithm
   - Framework with poly(1/ε) calls
   - $1/\varepsilon^{19}$ for semi-streaming
   - $1/\varepsilon^{52}$ for MPC, and CONGEST

2. By [Mitrović, Mukherjee, Sankowski, Sheu, 2025]
   - Simplify semi-streaming algorithm
   - All complexities improved by $(1/\varepsilon)^{13}$

- Not clear if they work for dynamic

New result 1

- A new framework for CONGEST and MPC
- Adapted from [MMSS25]'s semi-streaming algorithm
- $\varepsilon^{-7} \log(1/\varepsilon)$ calls

Reference	# calls to A_{mat}
[McG05]	$(1/\varepsilon)^{O(1/\varepsilon)}$
[FMU22]	ε^{-52}
[MMSS25]	ε^{-39}
[this]	$\varepsilon^{-7} \log(1/\varepsilon)$

New result 2

- First framework (for dynamic) with poly(1/ε) calls in general graphs

Reference	Setting	Complexity in ε	Complexity in n
[AKK24]	dynamic	exp(1/ε)	$n^{o(1)}$ ORS(n, $\theta_\varepsilon(n)$)
[Liu24]	dynamic, bipartite	poly(1/ε)	$\frac{n}{2^{\Omega(\sqrt{\log n})}}$
[Liu24]	offline dynamic, bipartite	poly(1/ε)	$n^{0.58}$
[this]	dynamic	poly(1/ε)	$n^{o(1)}$ ORS(n, $\theta_\varepsilon(n)$)
[this]	dynamic	poly(1/ε)	$\frac{n}{2^{\Omega(\sqrt{\log n})}}$
[this]	offline dynamic	poly(1/ε)	$n^{0.58}$

Remark (technical details)

1. All frameworks require additional technical assumptions

   - Need simple procedures for preparing the inputs to A_{mat}

2. For bipartite graphs, better frameworks exist
   (See [Assadi, Khanna, Kiss, 2024] for a list)

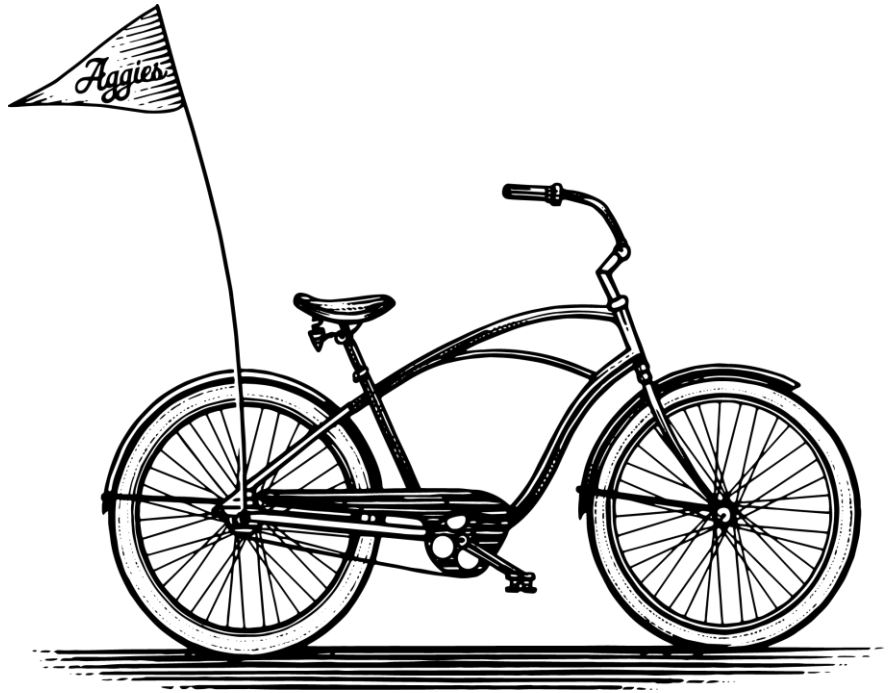
3. **Fun fact:** All frameworks above work by simulating semi-streaming algorithms
   (except [Liu24]'s algorithm)

Route map (technical part)

1. Review of [MMSS25]'s algorithm

2. Simulation in distributed settings
(no model-specific details)

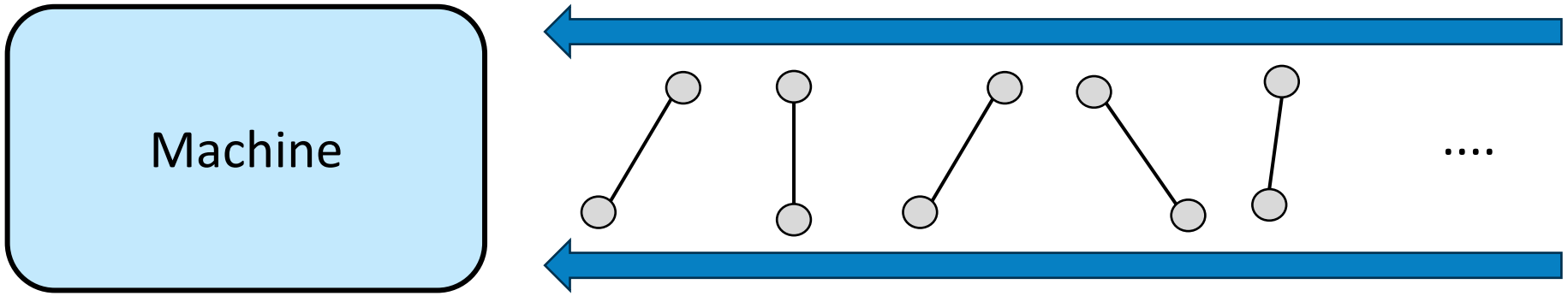
3. Challenges in dynamic settings
(using a weaker A_{mat})



Semi-streaming setting (review)

- No random access to G
- Edges presented as stream
- $\tilde{O}(n)$ memory
- Can make multiple passes

- **Goal:** minimize the number of passes



$O(n \text{ poly log } n)$ memory

data stream (edges)

Definition

- *Free node*: unmatched vertex
- *Alternating path*: path alternates between matched and unmatched edges
- *Augmenting path*: alternating path from a free node to another

Starting point - short augmenting paths

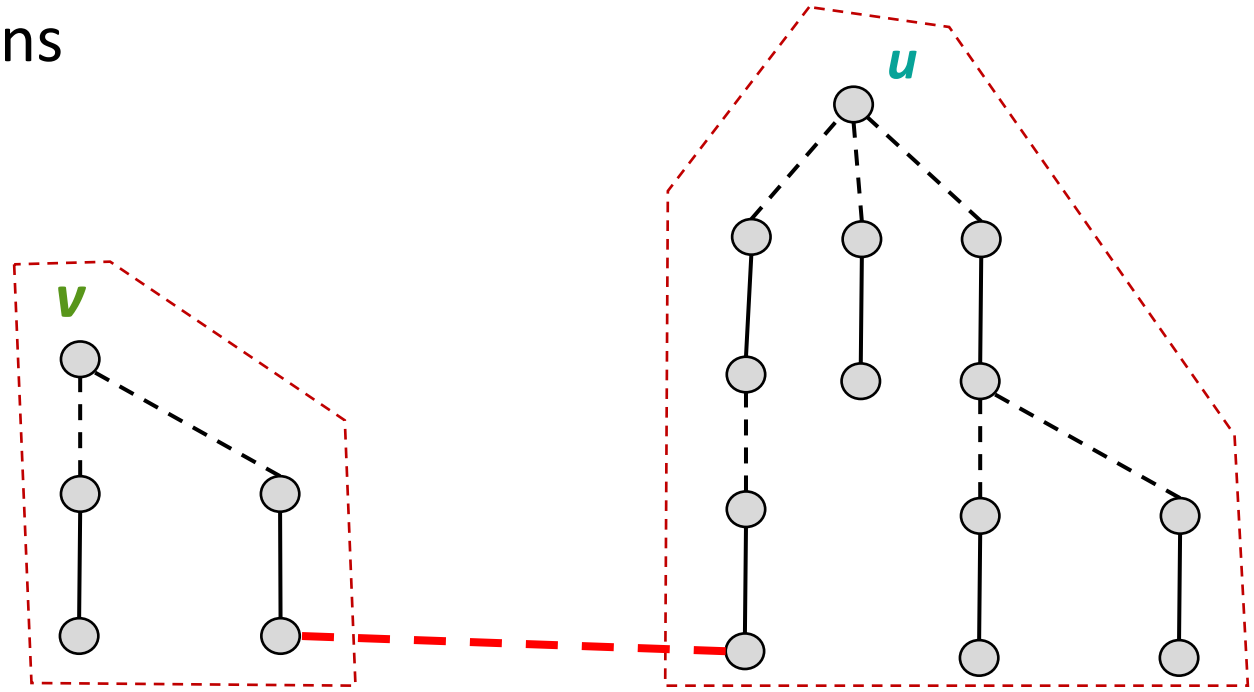
Lemma

Let M be a matching and Y be an inclusion-maximal set of $2/\varepsilon$-long vertex-disjoint augmenting paths. If $|Y| < \varepsilon^2|M|/6$, then M is a $(1+\varepsilon)$-approximate maximum matching.

[Kalantari, Shokoufandeh ‘95] [McGregor ‘05] [Eggert, Kliemann, Munstermann, Srivastav ‘12]

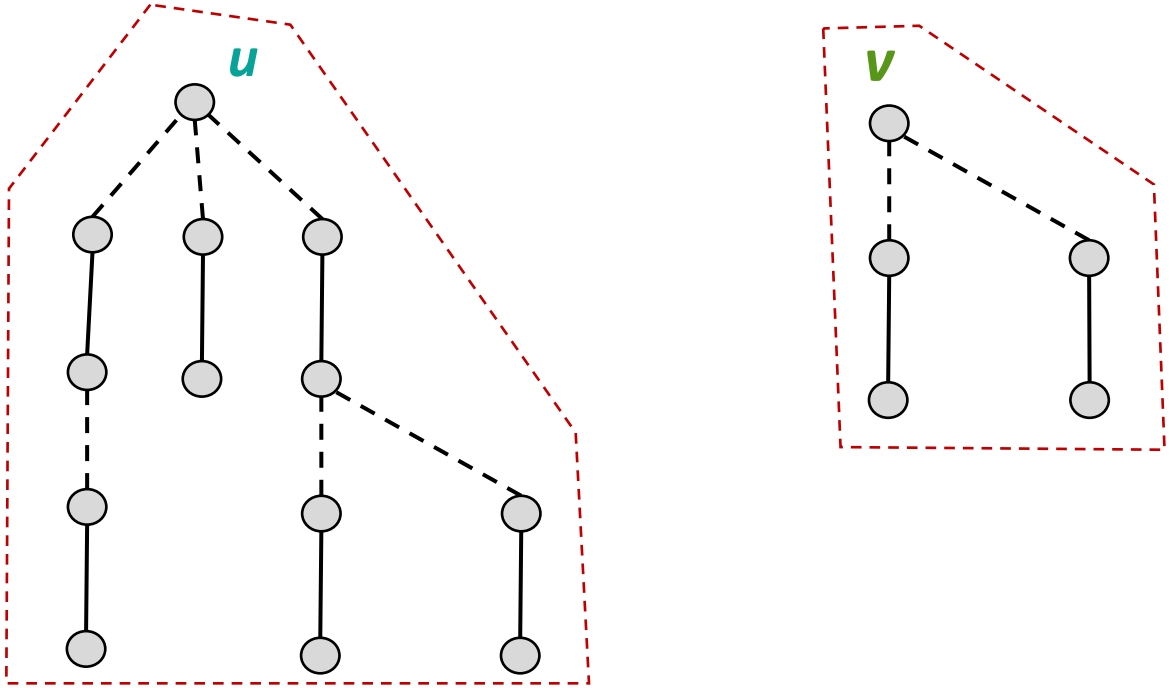
Idea of [MMSS25]

- Start from a maximal matching M
- Growing disjoint *alternating trees* of depth $O(1/\varepsilon)$
- Extend these trees to find augmentations



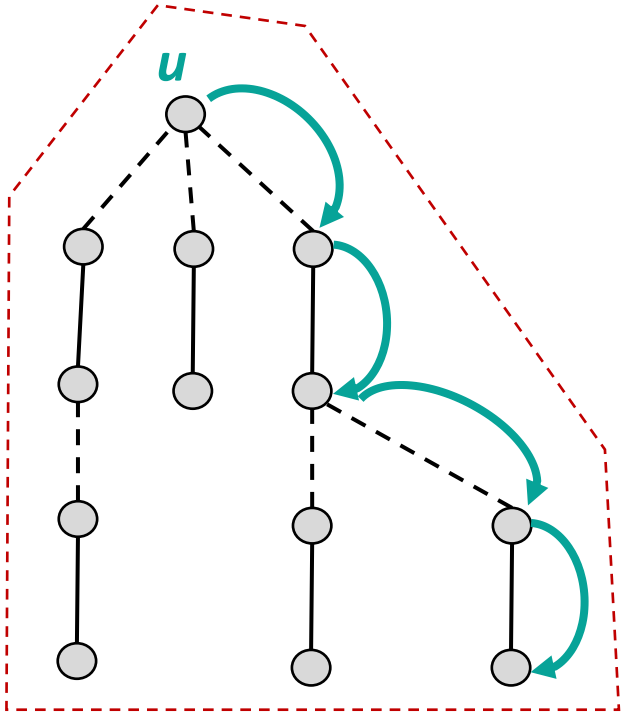
Alternating trees

- Each free node maintains an *alternating tree*
  - Root is the free node
  - Root-to-leaf paths are even-length alternating paths



Active paths

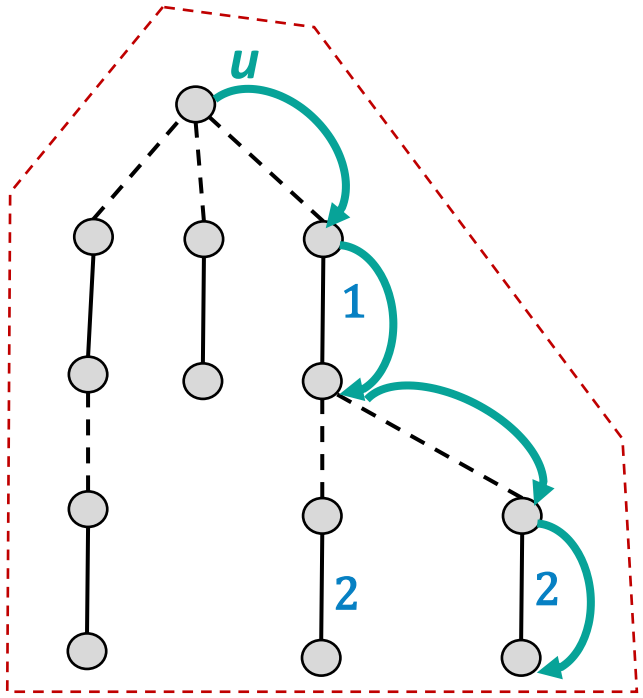
- Each tree has an *active path*
  - Starts from root
  - Even length



active path

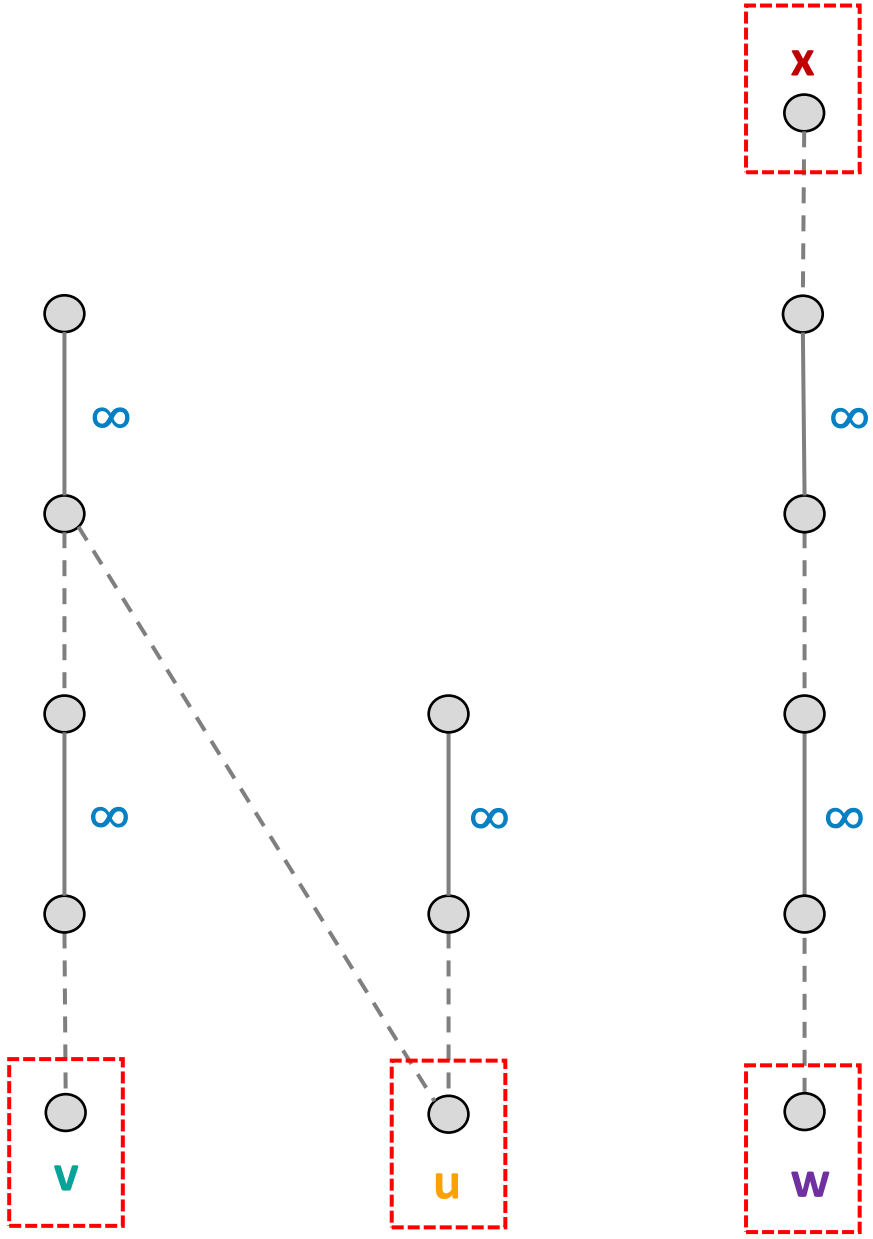
Edge label

- Each matched edge *e* maintains a *label L*(*e*)
- Represents the depth of *e* in the tree (informal)



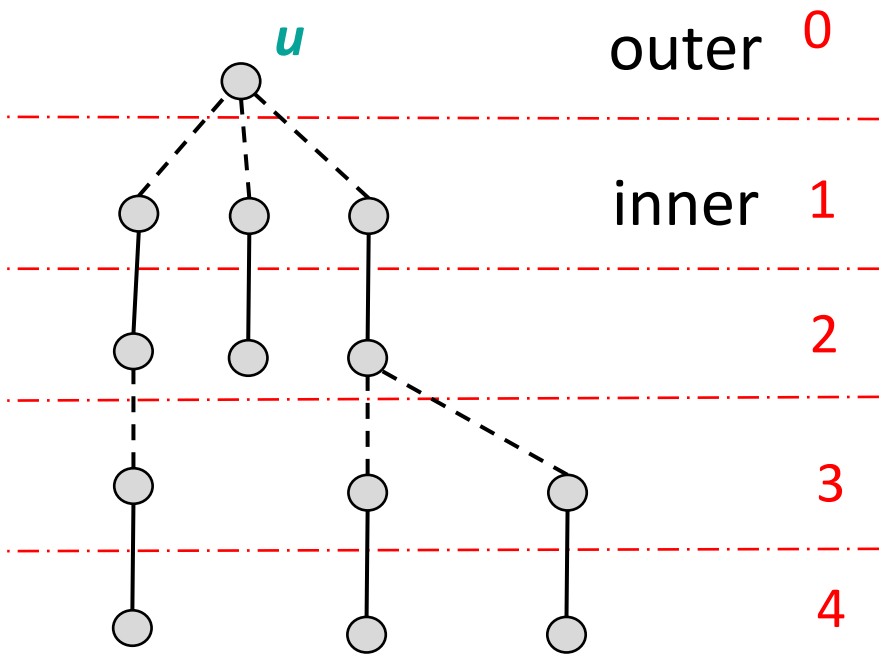
Initialization

- Each free node itself is a tree
- Active path is empty
- Label of each edge is ∞



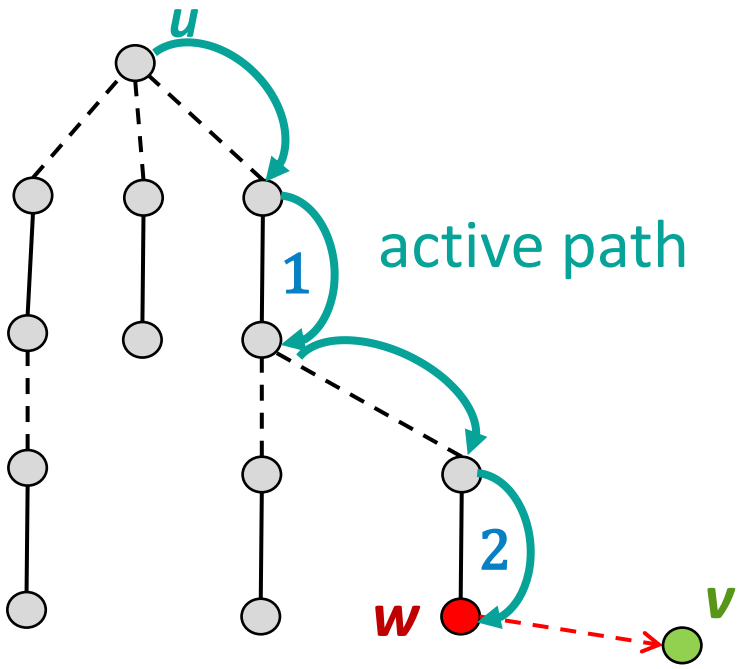
Outer / inner vertices

- Even layers: outer vertices
- Odd layers: inner vertices
- Root: outer vertex



Growing trees

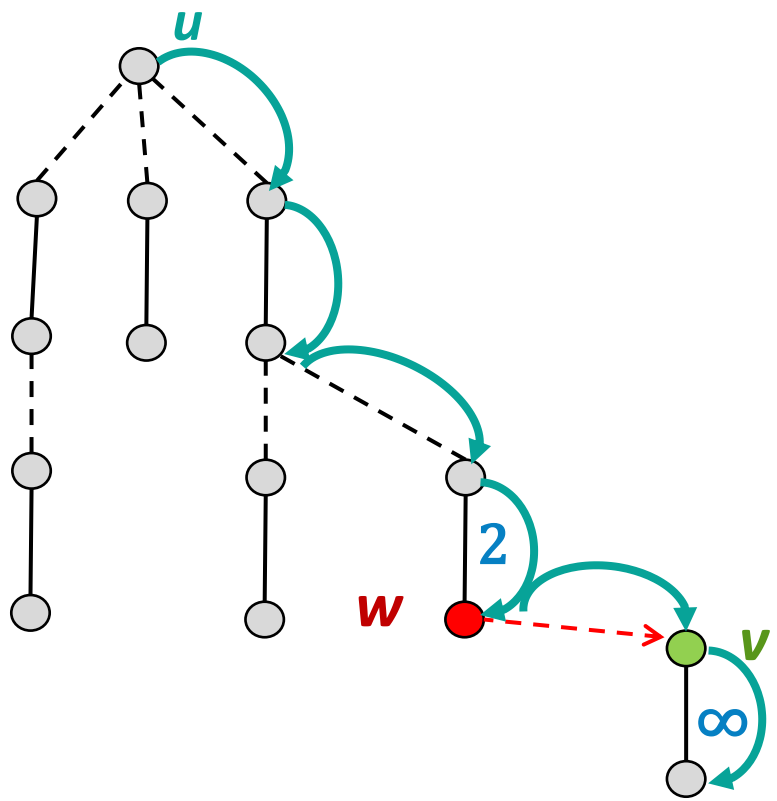
- Read edges (***w***, ***v***) from stream
- Focus on edges from an active path



Growing trees

- Read edges (***w***, ***v***) from stream
- Focus on edges from an active path

Case 1: ***v*** is not in any tree → **Extend**

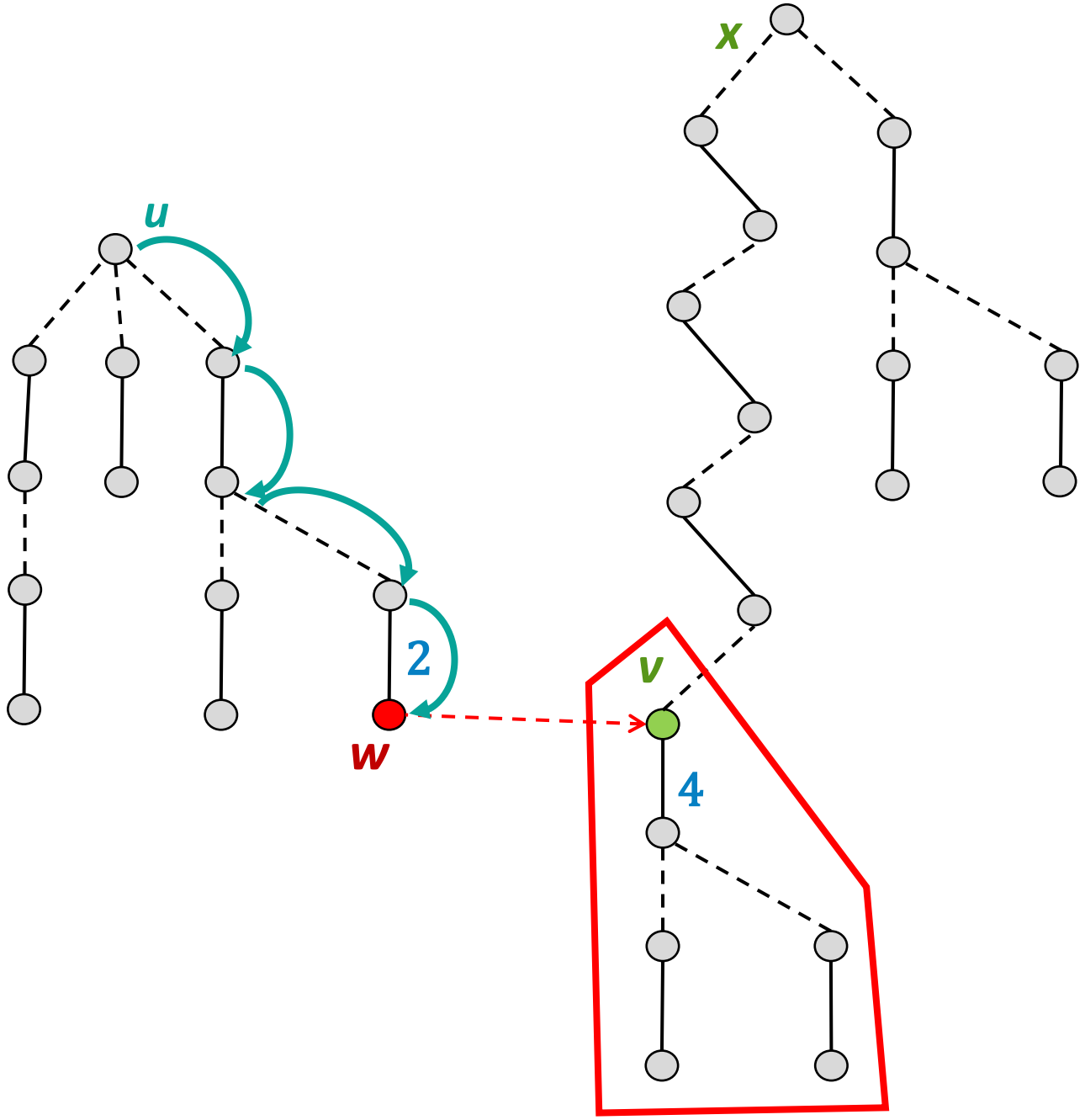


Growing trees

- Read edges (***w***, ***v***) from stream
- Focus on edges from an active path

Case 1: ***v*** is not in any tree → **Extend**

Case 2: ***v*** is an inner vertex → **Overtake**
(if label can be reduced)
(also take the subtree of ***v***)

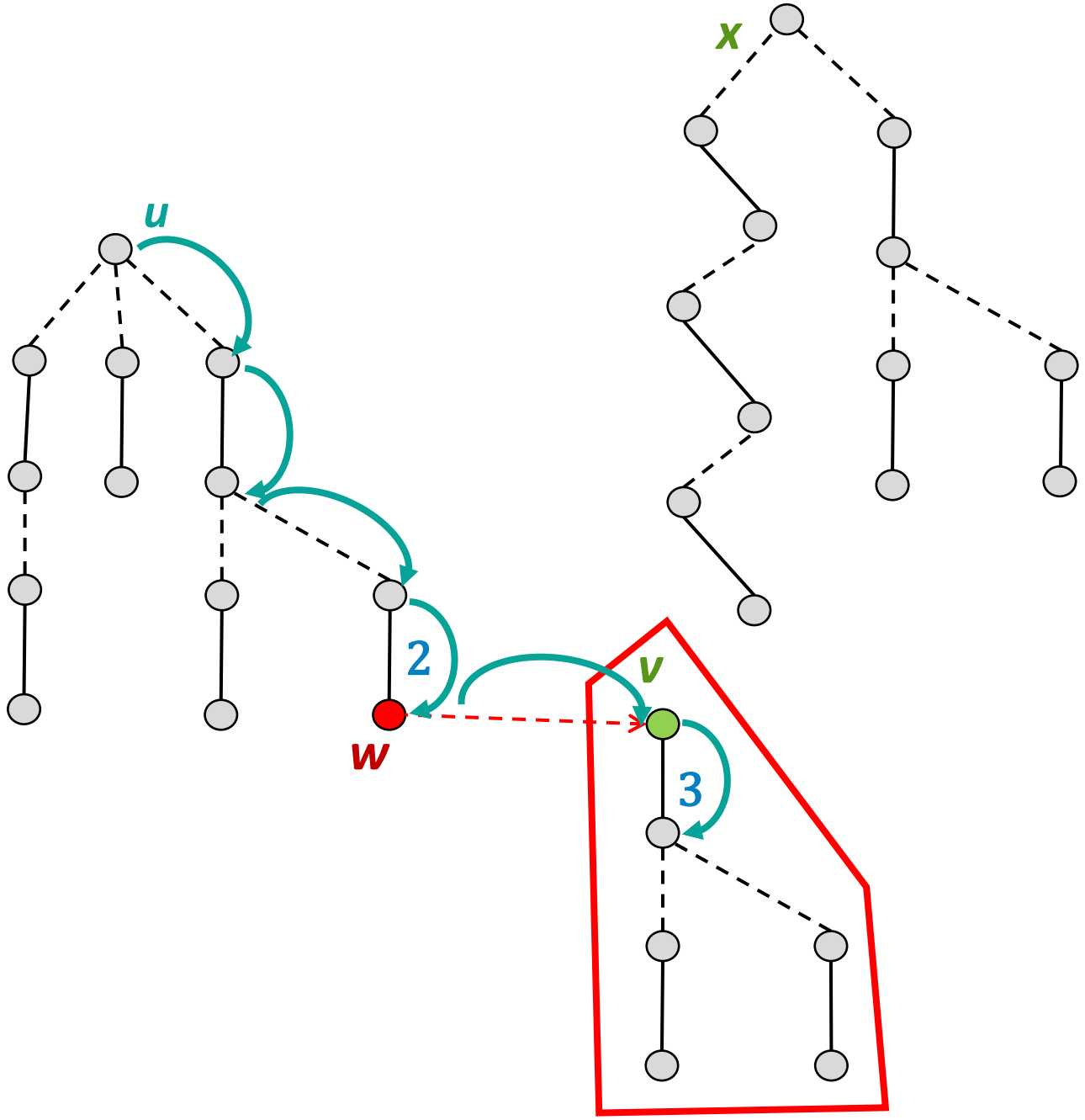


Growing trees

- Read edges (***w***, ***v***) from stream
- Focus on edges from an active path

Case 1: ***v*** is not in any tree → **Extend**

Case 2: ***v*** is an inner vertex → **Overtake**
(if label can be reduced)
(also take the subtree of ***v***)



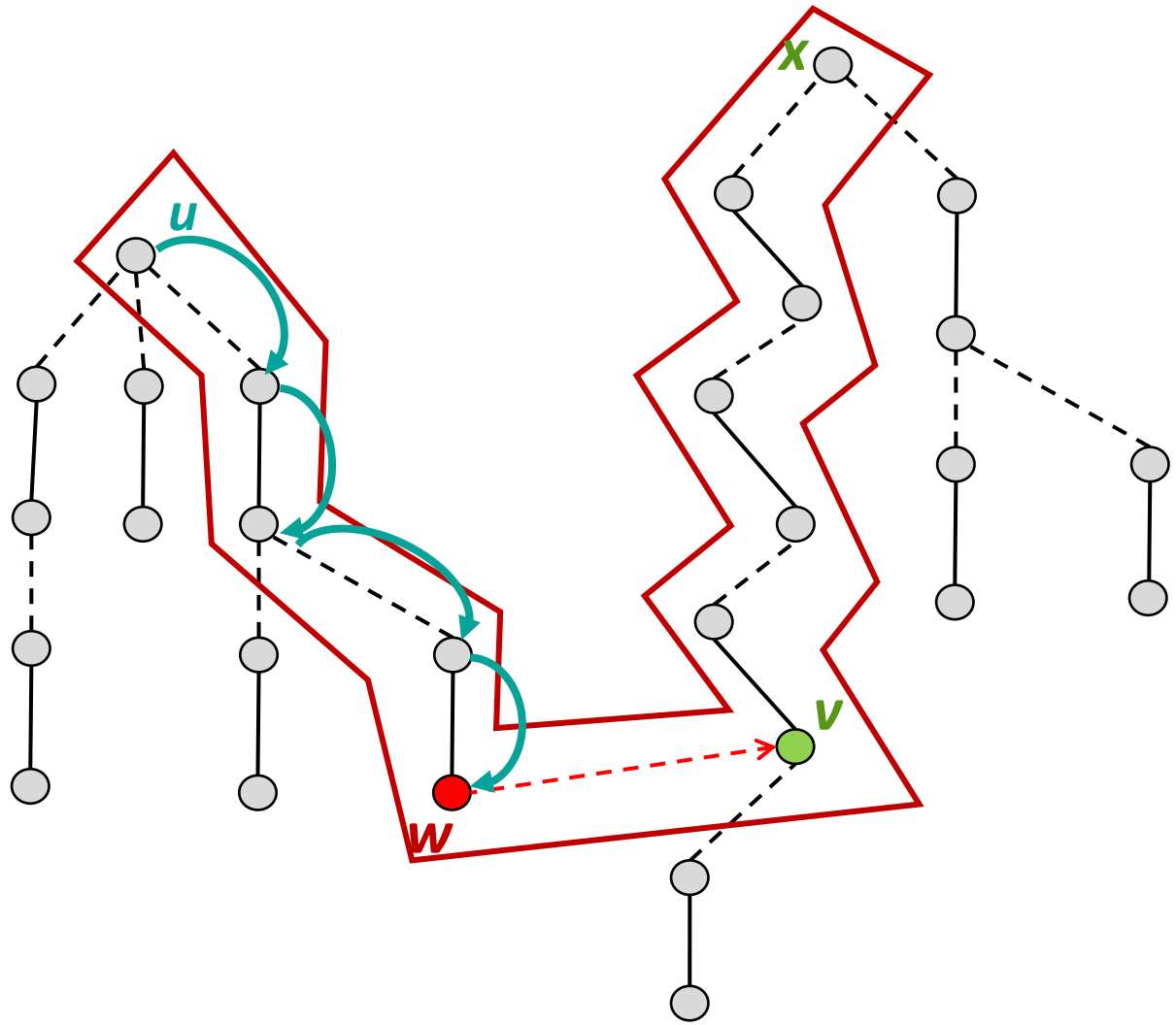
Growing trees

- Read edges (***w***, ***v***) from stream
- Focus on edges from an active path

Case 1: ***v*** is not in any tree → **Extend**

Case 2: ***v*** is an inner vertex → **Overtake**

Case 3: ***v*** is an outer vertex of another tree
→ **Augment**
(remove both trees)



Growing trees

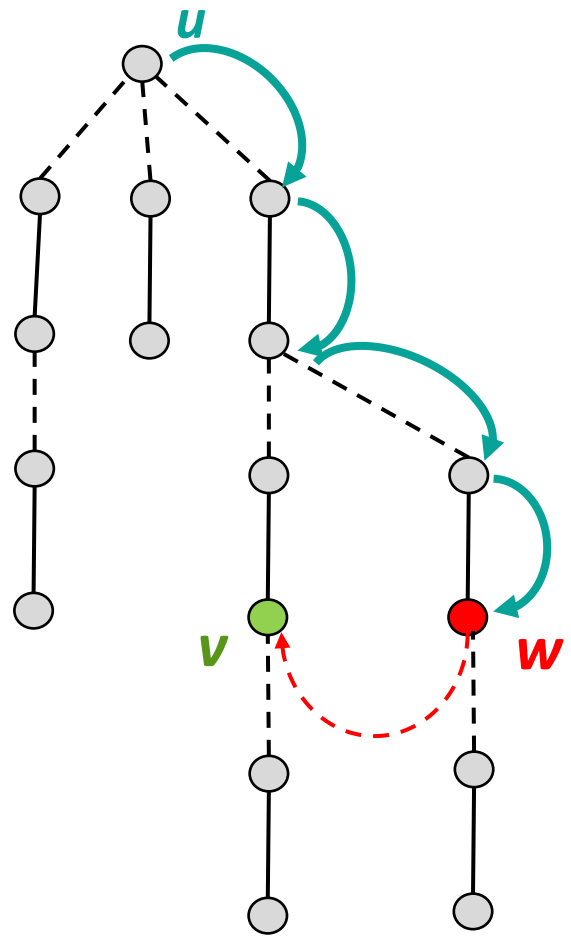
- Read edges (***w***, ***v***) from stream
- Focus on edges from an active path

Case 1: ***v*** is not in any tree → **Extend**

Case 2: ***v*** is an inner vertex → **Overtake**

Case 3: ***v*** is an outer vertex of another tree
→ **Augment**

Case 4: ***v*** is an outer vertex of the same tree
→ **Contract** (skipped)



Summary

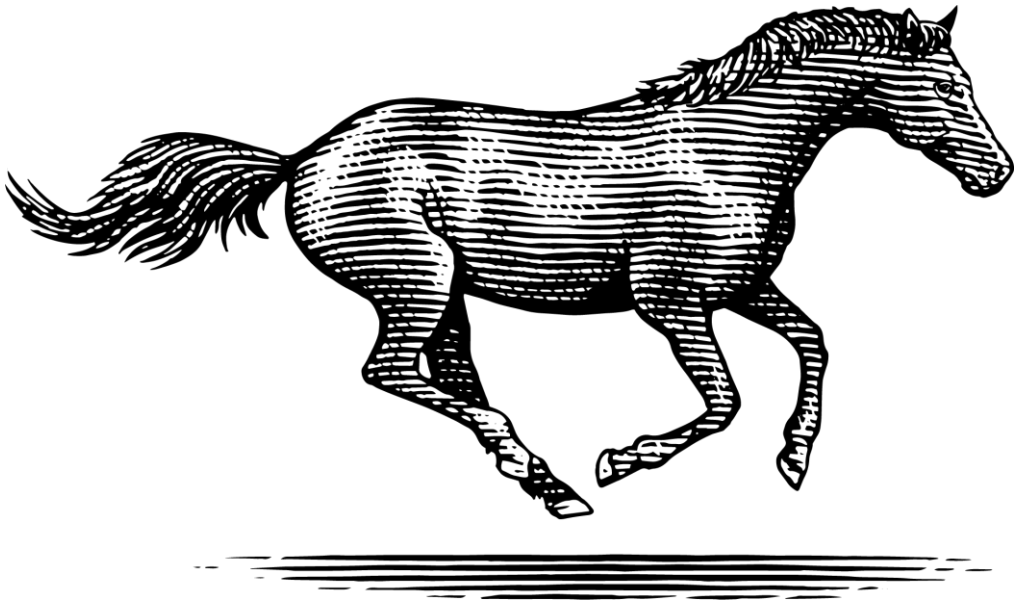
- In each pass, scan edges and perform Extend / Overtake / Augment / Contract
- Run and repeated for poly(1/ε) passes
- Finds (1+ε)-approx. matching
- **Properties:**
  - Tree size is always $1/\varepsilon^6$
  - Each tree can only do one operation in a pass
- **[FMU22, MMSS25]'s framework:** simulate each pass using $1/\varepsilon^{33}$ calls of A_{mat}

Route map (part 2)

1. Review of [MMSS25]'s algorithm

2. Simulation in distributed settings

3. Challenges in dynamic settings



Overview - framework

- **Goal:** Find (1+ε)-approx. matching using A_{mat} (constant approx.)
- **Approach:** Simulate each pass with $\varepsilon^{-1} \log(1/\varepsilon)$ calls to A_{mat}
- **Idea:** repeat two steps:
  1. Use A_{mat} to find a matching
  2. Perform basic operations on matched edges
- Focus on Augment and Overtake

Review - Augment

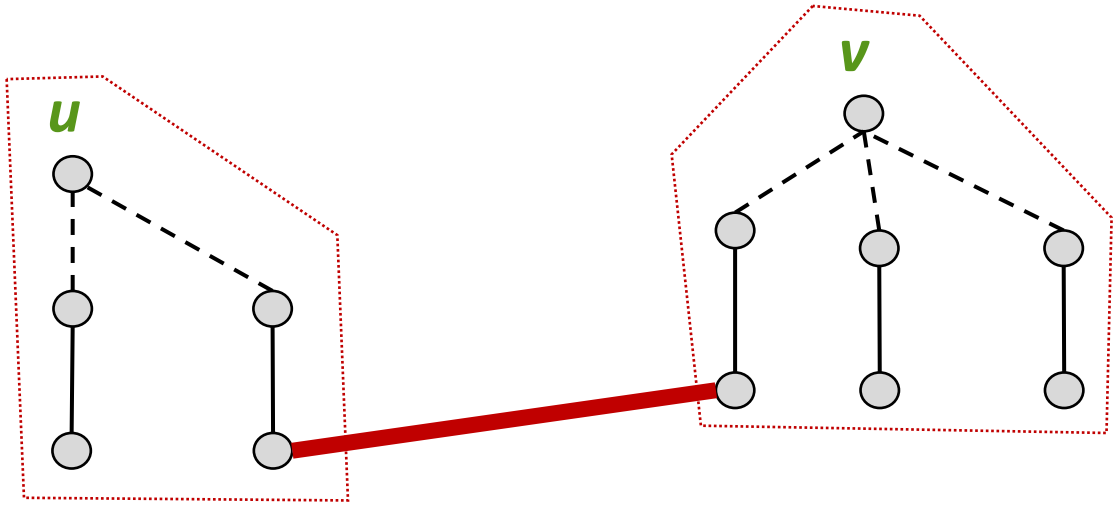
- **Applied on:**

  An edge between outer vertices of different trees

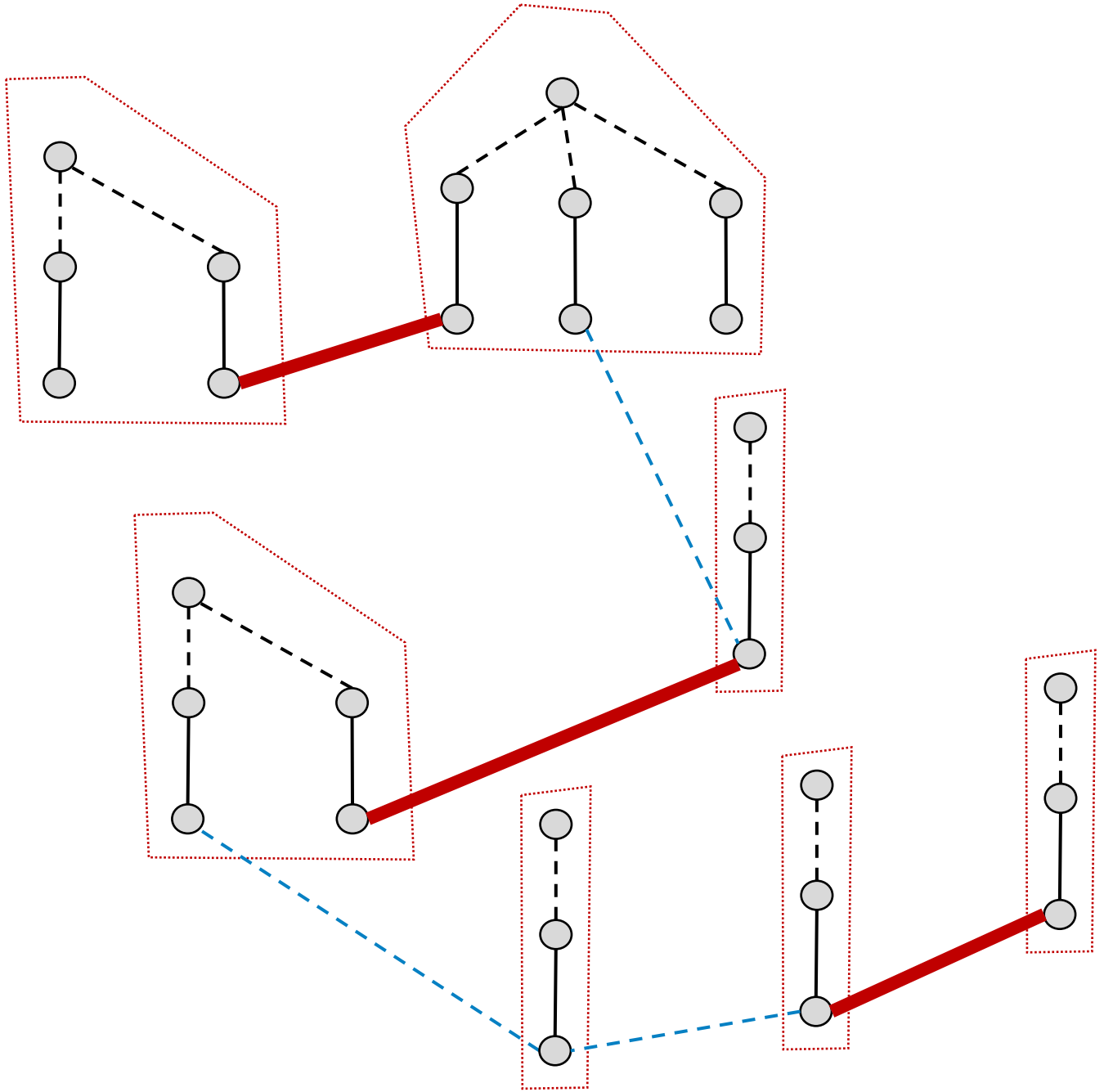
- **Result:**

  The two trees are removed

  Augmentation recorded

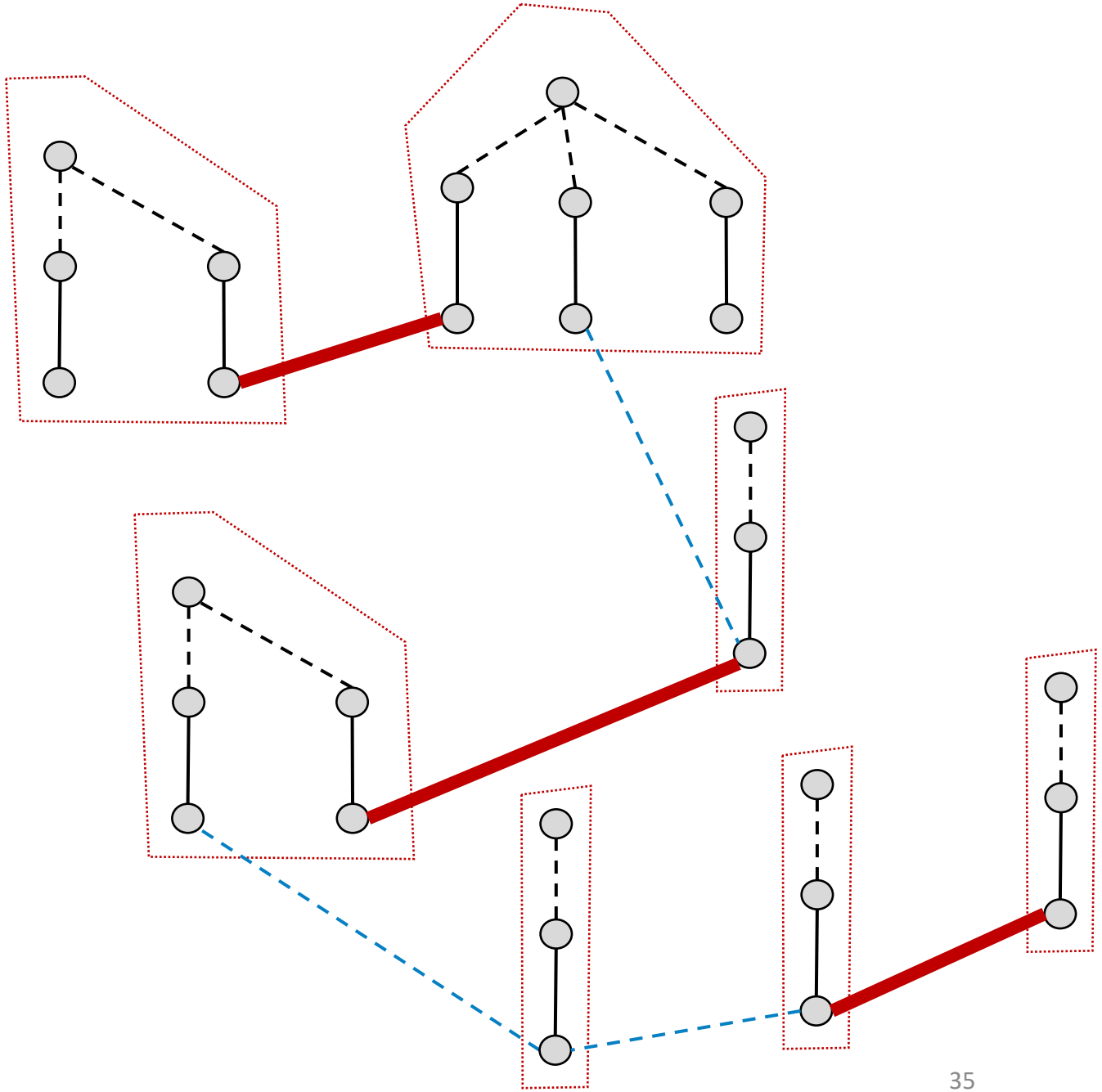


Example



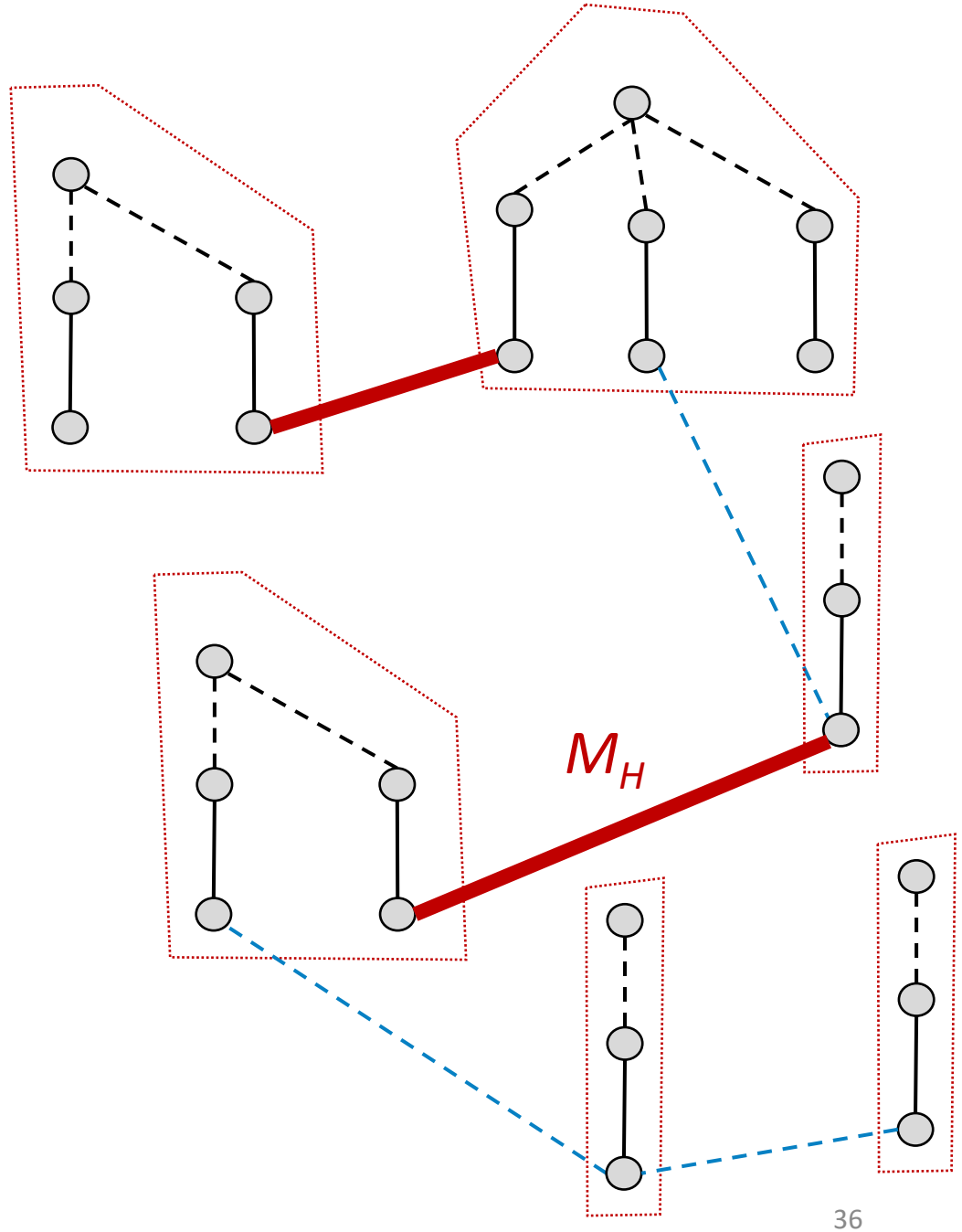
Observation

- Removed trees form a matching!



Simulation - Augment

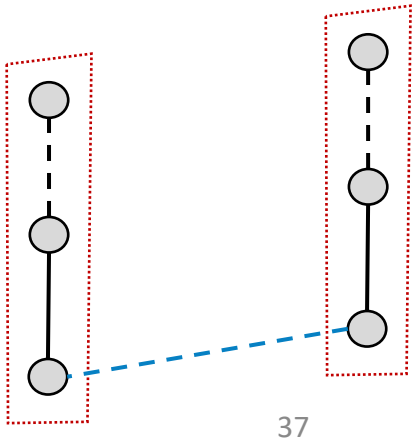
- Construct graph H
- Each tree is shrunk into a node
- Build edges between trees if Augment is possible
- Invoking A_{mat} on H to find matching M_H



Simulation - Augment

- Construct graph H
- Each tree is shrunk into a node
- Build edges between trees if Augment is possible
- Invoking A_{mat} on H to find matching M_H
- Perform Augment on returned matching
- Repeat the above for $O(\log(1/\varepsilon))$ calls

(Removed)



Analysis - Augment

- Let:
  - $OPT(H)$ = current maximum matching size of H
  - r = approx. factor of A_{mat}
- In each call, A_{mat} finds an r-approximate matching
- All matched vertices are removed from H

$\rightarrow$ $OPT(H)$ reduced by a $(1 - 1/r)$ factor

$\rightarrow$ After k calls, $OPT(H)$ reduced by $(1 - 1/r)^k \approx e^{k/r}$ times

$\rightarrow$ With $10\, r \log(1/\varepsilon)$ calls, $OPT(H)$ reduced by ε^{-10} times

Analysis - Augment

- But, $OPT(H)$ is not reduced to zero! (Some augments missed)
- We could miss $\varepsilon^{10}|M|$ augmentations
- **Claim:** it's ok to miss them
- Recall our starting point:

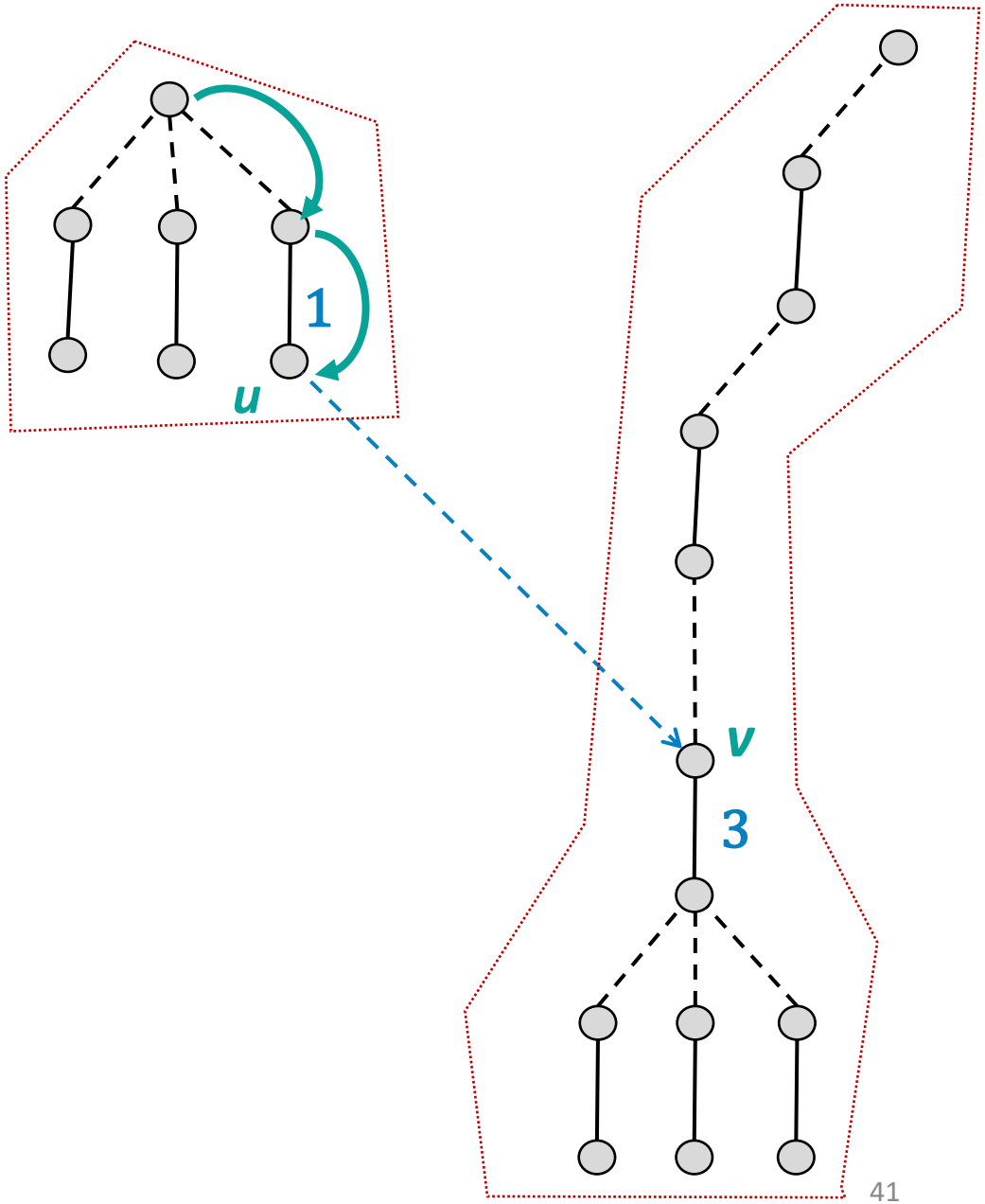
> Lemma:
> If M has at most $\varepsilon^2|M|/6$ short augmentations, then it is a $(1+\varepsilon)$-approx.

It's ok to miss $O(\varepsilon^2|M|)$ augmentations

Simulation - Overtake

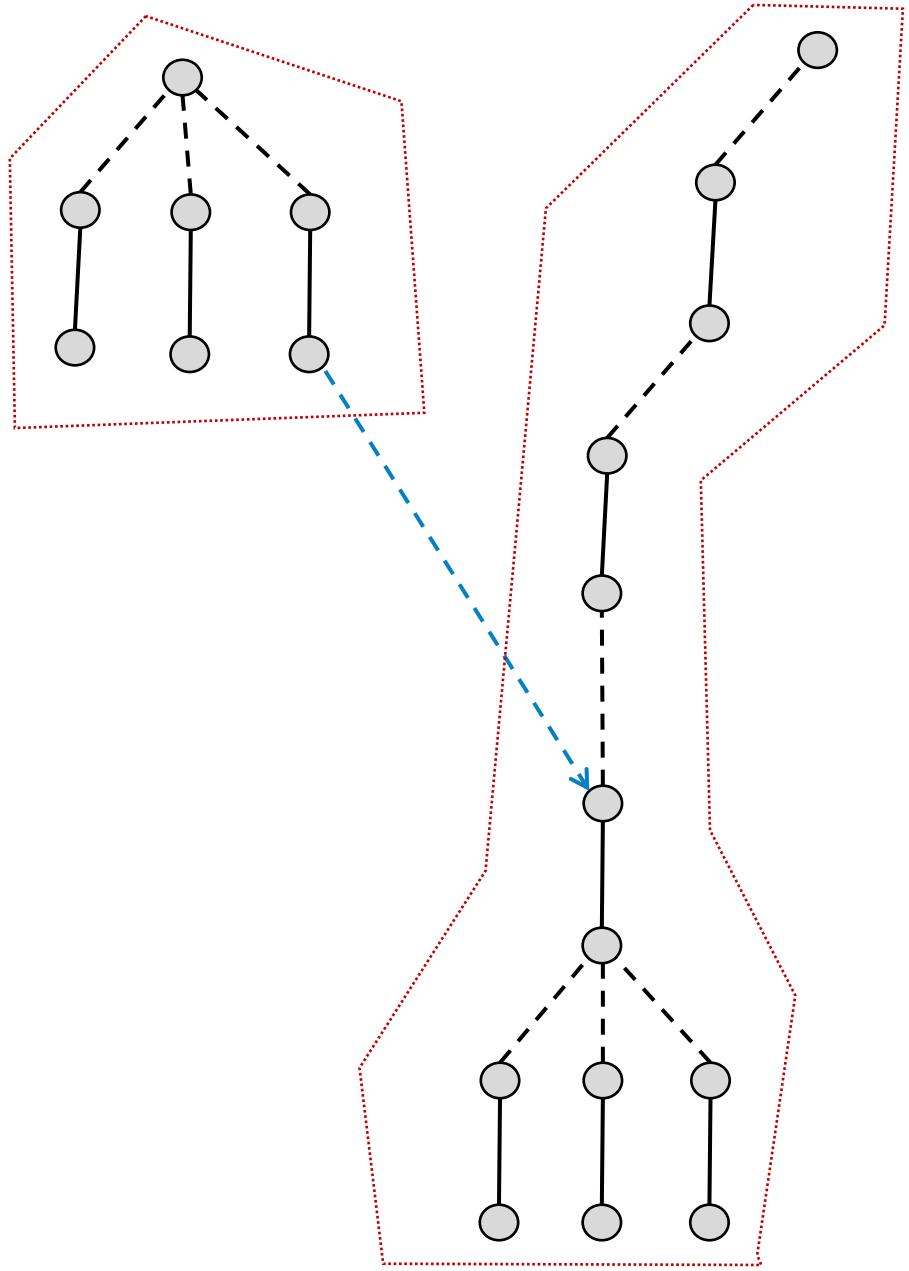
- **Applied on:**
  Edges (*u*, *v*)
  u is head of active path
  v is inner vertex
  label can be reduced
  the tree of *u* has not extended

- **Result:**
  u takes the subtree of *v*



Simulation - Overtake

- **Same idea?**
  - Build graph *H*
  - Contracted each tree
  - Each edge represents a possible Overtake
- **Problem:** analysis does not apply
- For Augment: Matched trees are removed
- For Overtake: Further overtake could happen

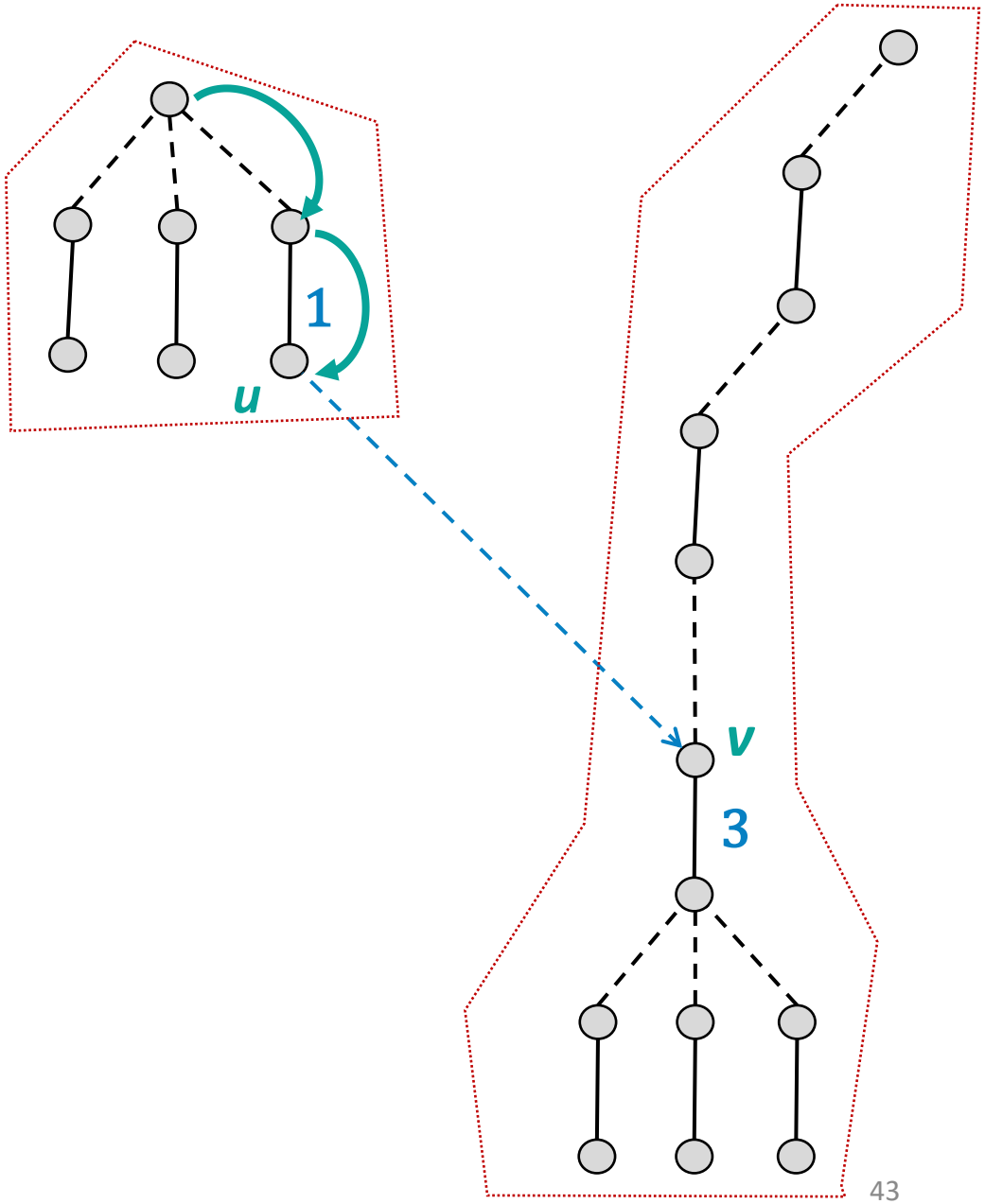


Idea

- Split into $O(1/\varepsilon)$ stages
- In stage s,
  only edges with label s can overtake
- **Goal:** In one stage, each tree can only
  overtake / taken once

→ After calling A_{mat},
can remove all matched trees

→ Previous analysis applies to one stage



Route map (part 3)

1. Review of [MMSS25]'s algorithm

2. Simulation in distributed settings

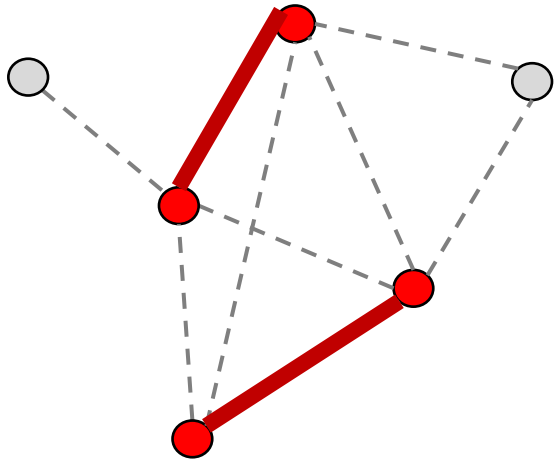
3. Challenges in dynamic settings

Dynamic setting (review)

- **Input:** Empty graph of *n* vertices
  Sequence of edge updates (add or remove edges)

- **Goal:** Maintain a (1+ε)-approx.

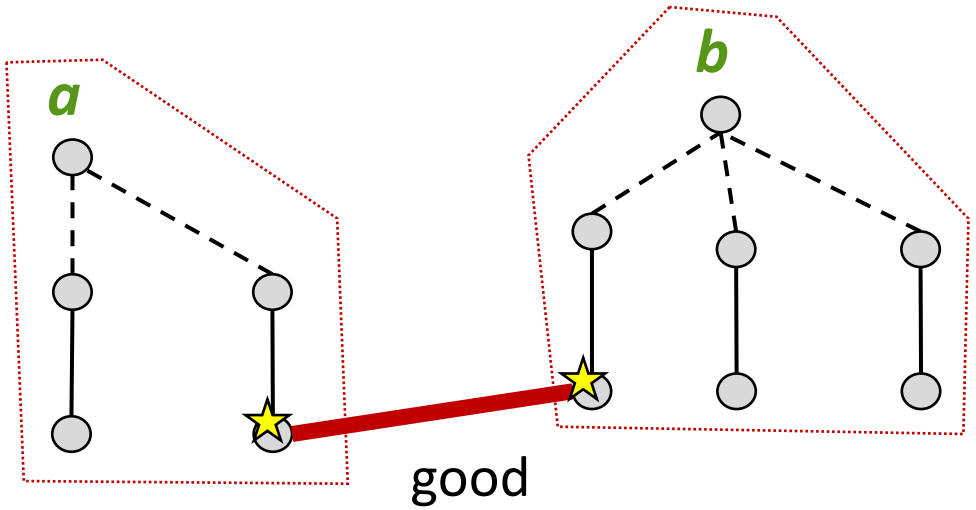
Known reduction to **static** problem

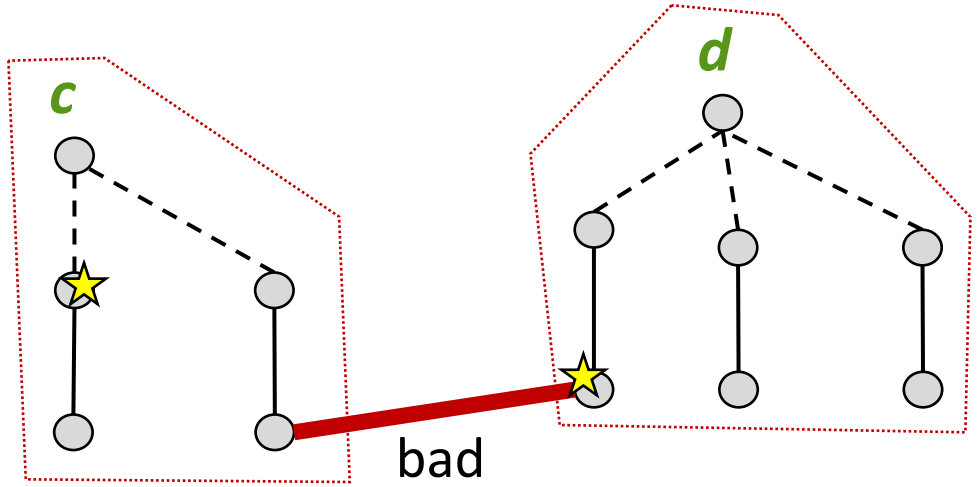


- We have a weaker A_{mat}, called A_{weak}
- Informal definition:
  - **Input:** graph G, vertex subset S
  - **Output:** Finds r-approximate matching of $G[S]$
  - **Constraint:** Input to A_{weak} must be prepared in $O_\varepsilon(n)$ time
- **Goal:** Call A_{weak} $poly(1/\varepsilon)$ times and finds a $(1+\varepsilon)$-approximate matching
- **Challenge:** A_{weak} only works on induced subgraphs!

Idea - vertex sampling

- **Recall, Augment operation:**
  Find edges between outer vertices of different trees
- Sample one outer vertex from each tree
- Suppose semi-streaming algorithm does Augment on (u, v)
- Our simulation works when u and v are both sampled



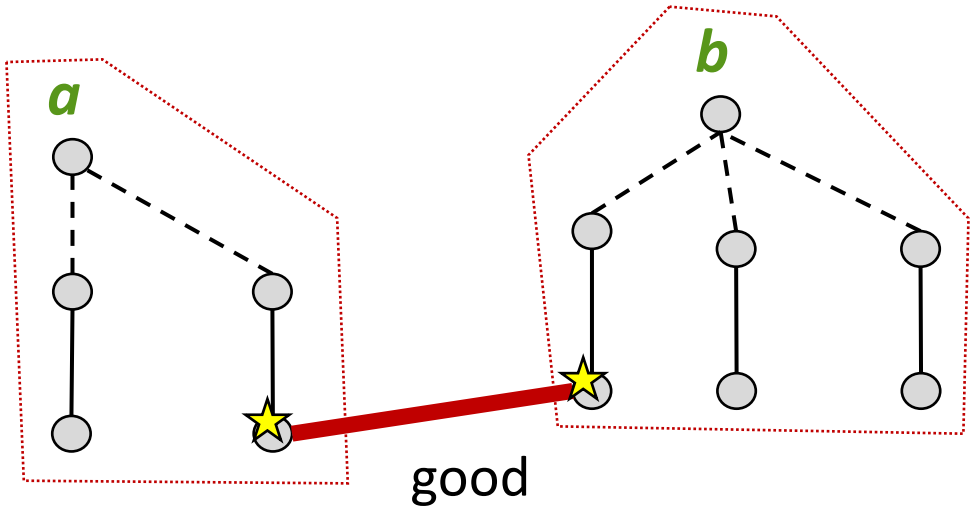


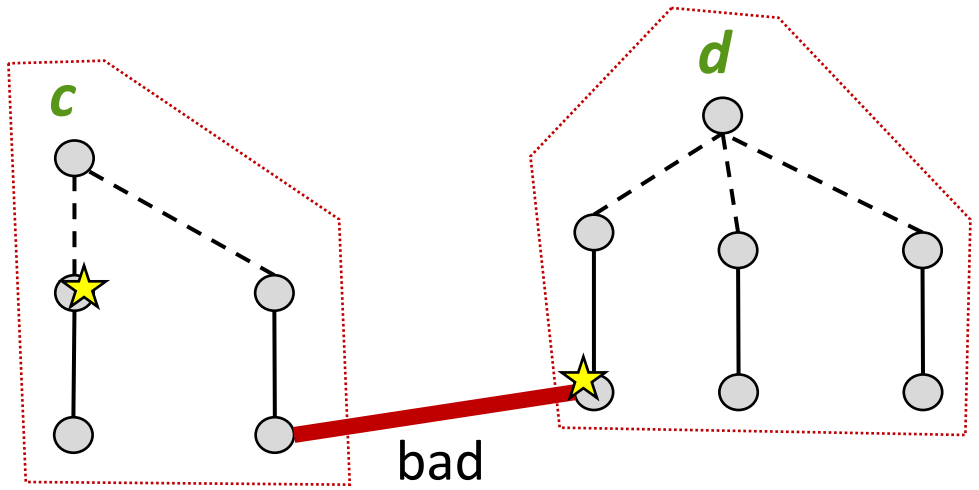
Analysis

- Size of each tree is poly(1/ε)

  → an edge preserved with probability poly(ε)

  → Use previous framework, with poly(1/ε) times more calls





Remark

- There are some additional challenges for overtake and contract
- But the overall idea is the same

Q&A

Contact
Wen-Horng Sheu
wsheu@ucdavis.edu