CPSC 768:
Scalable and Private Graph Algorithms

Lecture 20: Dynamic Graph Algorithms

Quanquan C. Liu
quanquan.liu@yale.edu

Announcements

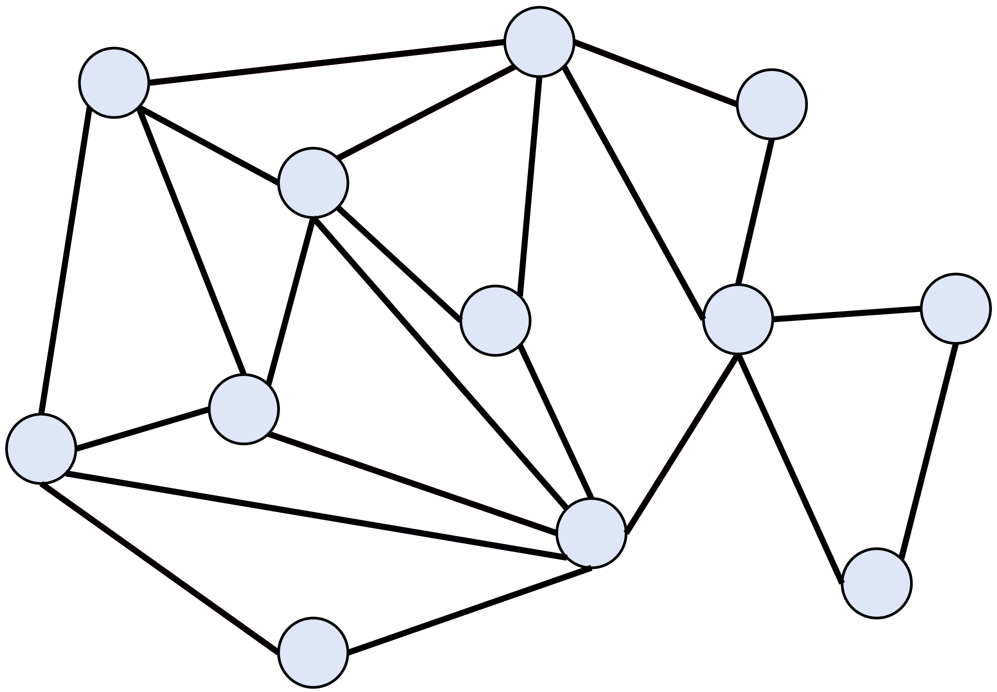
- **Final project report and presentation: April 24th (last day of class)**
  - Final project presentation is a 30 min presentation

Dynamic Graph Algorithms

- Updates to the graph occur where edges are added and deleted from the graph

Edge **insertions/deletions** arrive **sequentially**

Maintain **graph property** after each update



Minimize Update Time

- **Want**: minimize the update time between updates
  - Amortized or worst-case (often a gap)

Sublinear Runtime:
strive for $\text{poly}(\log n)$

Minimize Update Time

- **Want**: minimize the update time between updates
  - Amortized or worst-case (often a gap)
- Sometimes need to do **preprocessing**
  - Small polynomial in the input graph

Sublinear Runtime: strive for $\text{poly}(\log n)$

Minimize Update Time

- **Want**: minimize the update time between updates
  - Amortized or worst-case (often a gap)
- Sometimes need to do **preprocessing**
  - Small polynomial in the input graph
- Sometimes have **queries** (e.g. connectivity queries)

Sublinear Runtime: strive for $\mathrm{poly}(\log n)$

Many Recent Results in Dynamic Graph Algorithms

- Dynamic maximum matching (find a matching of maximum size):
  - **Best known:** $(1.973 + \varepsilon)$-approximation in $\mathrm{poly}(\log n)$ update time [BKSW SODA `23]

Many Recent Results in Dynamic Graph Algorithms

- Dynamic maximum matching (find a matching of maximum size):
  - **Best known:** $(1.973 + \varepsilon)$-approximation in $\text{poly}(\log n)$ update time [BKSW SODA `23]
- Dynamic $(\Delta + 1)$-coloring (find a valid coloring with $\Delta + 1$ colors):
  - **Best known:** $O(1)$ update time [HP, BGK**L**S TALG `22]

Many Recent Results in Dynamic Graph Algorithms

- Dynamic maximum matching (find a matching of maximum size):
  - **Best known:** $(1.973 + \varepsilon)$-approximation in $\mathrm{poly}(\log n)$ update time [BKSW SODA `23]
- Dynamic $(\Delta + 1)$-coloring (find a valid coloring with $\Delta + 1$ colors):
  - **Best known:** $O(1)$ update time [HP, BGK**L**S TALG `22]
- Approximate Densest Subgraph:
  - **Best known:** $\mathrm{poly}(\log n)$ update time [SW STOC ‘20, CCHHQRS SODA ‘24]

Dynamic Algorithms + Other Models

- **Dynamic meets distributed**:
  - Dynamic updates in a distributed graph; very recent, nascent field
    - Count # of rounds/messages sent in the graph

Dynamic Algorithms + Other Models

- **Dynamic meets distributed**:
  - Dynamic updates in a distributed graph; very recent, nascent field
    - Count # of rounds/messages sent in the graph
  - Clique counting [BC ICALP ‘19, **L** IPL ‘23]
  - Maximal Independent Set [AOSS STOC ‘18, A**L**S ITCS ‘22]

Dynamic Algorithms + Other Models

- **Dynamic meets distributed**:
  - Dynamic updates in a distributed graph; very recent, nascent field
    - Count # of rounds/messages sent in the graph
  - Clique counting [BC ICALP '19, **L** IPL '23]
  - Maximal Independent Set [AOSS STOC '18, A**L**S ITCS '22]
- **Learning-augmented** Dynamic Algorithms [**L**S '23, BFNP '23, HSSY '23]

Types of Dynamic Algorithms

- **Incremental/Decremental** vs. **Fully Dynamic**
  - Incremental/decremental algorithms:
    - Only edge insertions/deletions, respectively

Types of Dynamic Algorithms

- **Incremental/Decremental** vs. **Fully Dynamic**
  - Incremental/decremental algorithms:
    - Only edge insertions/deletions, respectively
- Sometimes **large gap in runtimes**

Types of Dynamic Algorithms

- **Incremental/Decremental** vs. **Fully Dynamic**
  - Incremental/decremental algorithms:
    - Only edge insertions/deletions, respectively
- Sometimes **large gap in runtimes**
  - **Polynomial** or **exponential gaps** in runtimes

Types of Dynamic Algorithms

	Best Fully Dynamic		Best Partially Dynamic	
Planar Digraph APSP	$\widetilde{O}\left(n^{2/3}\right)$	[FR06, Kle05]	$\widetilde{O}(\sqrt{n})$	[DGWN22]
Triconnectivity	$O(n^{2/3})$	[GIS99]	$\widetilde{O}(1)$	[HR20, PSS17]
k-Edge Connectivity	$n^{o(1)}$	[JS22]	$\widetilde{O}(1)$	[CDK+21]
Dynamic DFS Tree	$\widetilde{O}\left(\sqrt{mn}\right)$	[BCCK19]	$\widetilde{O}(n)$	[BCCK19, CDW+18]
Minimum Spanning Forest	$\widetilde{O}(1)$	[HDLT01]	$\widetilde{O}(1)$	[Epp94]
APSP	$\left(\frac{256}{k^2}\right)^{4/k}$-Approx $\widetilde{O}\left(n^k\right)$ update $\widetilde{O}(n^{k/8})$ query	[FGNS23]	$(2r-1)^k$-Approx $\widetilde{O}\left(m^{1/(k+1)}n^{k/r}\right)$	[CGH+20]
AP Maxflow/Mincut	$O(\log(n)\log\log n)$-Approx $\widetilde{O}\left(n^{2/3+o(1)}\right)$	[CGH+20]	$O\left(\log^{8k}(n)\right)$-Approx. $\widetilde{O}\left(n^{2/(k+1)}\right)$	[Gor19, GHS19]
MCF	$(1+\varepsilon)$-Approx $\widetilde{O}(1)$ update $\widetilde{O}(n)$ query	[CGH+20]	$O(\log^{8k}(n))$-Approx. $\widetilde{O}\left(n^{2/(k+1)}\right)$ update $\widetilde{O}(P^2)$ query	[Gor19, GHS19]
Strongly Connected Components	$\Omega(m^{1-\varepsilon})$ query or update	[AW14]	$\widetilde{O}(m)$	[Rod13]
Uniform Sparsest Cut	$2^{O(\log^{5/6}(n))}$-Approx $2^{O(\log^{5/6}(n))}$ update $O(\log^{1/6}(n))$ query	[GRST21]	$O\left(\log^{8k}(n)\right)$-Approx $\widetilde{O}\left(n^{2/(k+1)}\right)$ $O(1)$ query	[Gor19, GHS19]
Submodular Max	1/4-Approx $\widetilde{O}(k^2)$	[DFL+23]	0.3178-Approx $\widetilde{O}\left(\text{poly}(k)\right)$	[FLN+22]

[LS '23]

Types of Dynamic Algorithms

- **Worst-case** vs. **amortized runtimes**

Types of Dynamic Algorithms

- **Worst-case** vs. **amortized runtimes**
  - Worst-case runtimes:
    - Monte Carlo (whp solution is correct; runtime always small)

Types of Dynamic Algorithms

- **Worst-case** vs. **amortized runtimes**
  - Worst-case runtimes:
    - Monte Carlo (whp solution is correct; runtime always small)
    - Las Vegas: runtime is whp; solution is always correct

Types of Dynamic Algorithms

- **Worst-case** vs. **amortized runtimes**
  - Worst-case runtimes:
    - Monte Carlo (whp solution is correct; runtime always small)
    - Las Vegas: runtime is whp; solution is always correct
  - Amortized runtimes:
    - *Lazy updates* strategy where updates are delayed and processed all at once

Types of Dynamic Algorithms

- Easy example of **lazy updates**:
  - $(2 + \varepsilon)$-Approximate Maximum matching with large $\Theta(n)$ size

Types of Dynamic Algorithms

- Easy example of **lazy updates**:
  - $(2+\varepsilon)$-Approximate Maximum matching with large $\Theta(n)$ size
    - Each update adds **at most one edge** to maximum matching

Types of Dynamic Algorithms

- Easy example of **lazy updates**:
  - $(2+\varepsilon)$-Approximate Maximum matching with large $\Theta(n)$ size
    - Each update adds **at most one edge** to maximum matching
    - Can afford to wait for $\varepsilon \cdot n$ updates

Types of Dynamic Algorithms

- Easy example of **lazy updates**:
  - $(2+\varepsilon)$-Approximate Maximum matching with large $\Theta(n)$ size
    - Each update adds **at most one edge** to maximum matching
    - Can afford to wait for $\varepsilon \cdot n$ updates
    - Rerun maximal matching static algorithm after $\varepsilon \cdot n$ updates

Types of Dynamic Algorithms

- Easy example of **lazy updates**:
  - $(2+\varepsilon)$-Approximate Maximum matching with large $\Theta(n)$ size
    - Each update adds **at most one edge** to maximum matching
    - Can afford to wait for $\varepsilon \cdot n$ updates
    - Rerun maximal matching static algorithm after $\varepsilon \cdot n$ updates
    - Amortized update time: $O\left(\frac{m}{\varepsilon n}\right) = o(n)$ when $m = o(n)$

Types of Dynamic Algorithms

- **Adaptive** vs. **Oblivious** vs. **Offline-Dynamic** Adversaries
  - Offline-Dynamic:
    - Sequence of updates occurs offline
    - Produce a valid solution after every update, minimize amortized update time

Types of Dynamic Algorithms

- **Adaptive** vs. **Oblivious** vs. **Offline-Dynamic** Adversaries
  - Offline-Dynamic:
    - Sequence of updates occurs offline
    - Produce a valid solution after every update, minimize amortized update time
  - Oblivious:
    - Sequence of updates determined before algorithm starts
    - Updates come one at a time online

Types of Dynamic Algorithms

- **Adaptive** vs. **Oblivious** vs. **Offline-Dynamic** Adversaries
  - Adaptive:
    - Can see algorithm output and determine next update based on output
    - Can see **everything** including internal randomness
    - **Deterministic algorithms** always robust against adaptive adversaries

Types of Dynamic Algorithms

- **Adaptive** vs. **Oblivious** vs. **Offline-Dynamic** Adversaries
  - Adaptive:
    - Can see algorithm output and determine next update based on output
    - Can see **everything** including internal randomness
    - **Deterministic algorithms** always robust against adaptive adversaries
- **Large gap between oblivious and adaptive adversaries:**
  - Example: dynamic connectivity, polynomial deterministic worst-case, polylog oblivious worst-case

Dynamic Connectivity

- Offline Dynamic [Eppstein ‘92]
  - Including offline dynamic minimum spanning tree
- Oblivious [Kapron-King-Mountjoy SODA ‘13]
- Deterministic [Frederickson’s Algorithm ‘85]

Dynamic Connectivity

- Offline Dynamic [Eppstein ‘92]
  - Including offline dynamic minimum spanning tree
- Oblivious [Kapron-King-Mountjoy SODA ‘13]
- ~~Deterministic [Frederickson’s Algorithm ‘85]~~ (classic, won’t discuss today—similar theme to newer algorithms: http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15850-f20/www/notes/lec3.pdf)

Offline-Dynamic Connectivity

- Receive an offline sequence of edge insertion/deletion updates and queries

Offline-Dynamic Connectivity

- Receive an offline sequence of edge insertion/deletion updates and queries
  - Insert or delete an edge

Offline-Dynamic Connectivity

- Receive an offline sequence of edge insertion/deletion updates and queries
  - Insert or delete an edge
  - Query(s, t) queries whether s and t are connected

Offline-Dynamic Connectivity

- Receive an offline sequence of edge insertion/deletion updates and queries
  - Insert or delete an edge
  - Query(s, t) queries whether s and t are connected
- We will use the offline dynamic minimum spanning tree algorithm of Eppstein '92

Offline-Dynamic Connectivity

- We will use the offline dynamic minimum spanning tree algorithm of Eppstein '92

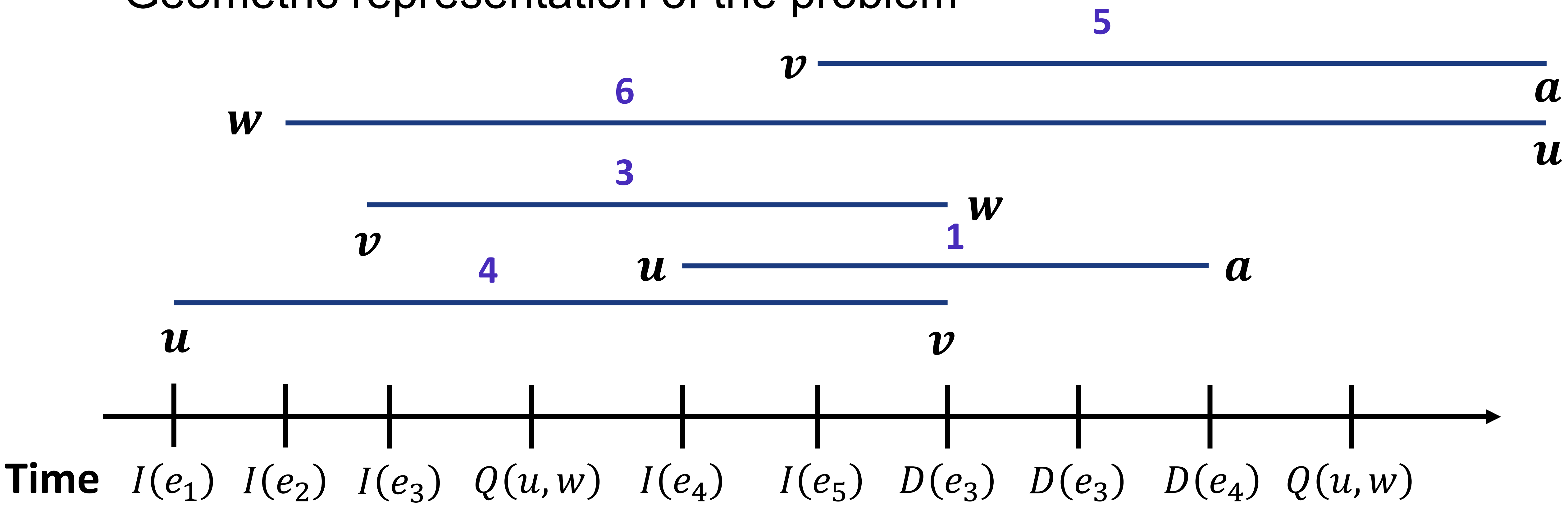
Offline-Dynamic Connectivity

- We will use the offline dynamic minimum spanning tree algorithm of Eppstein ‘92
- Geometric representation of the problem

Time

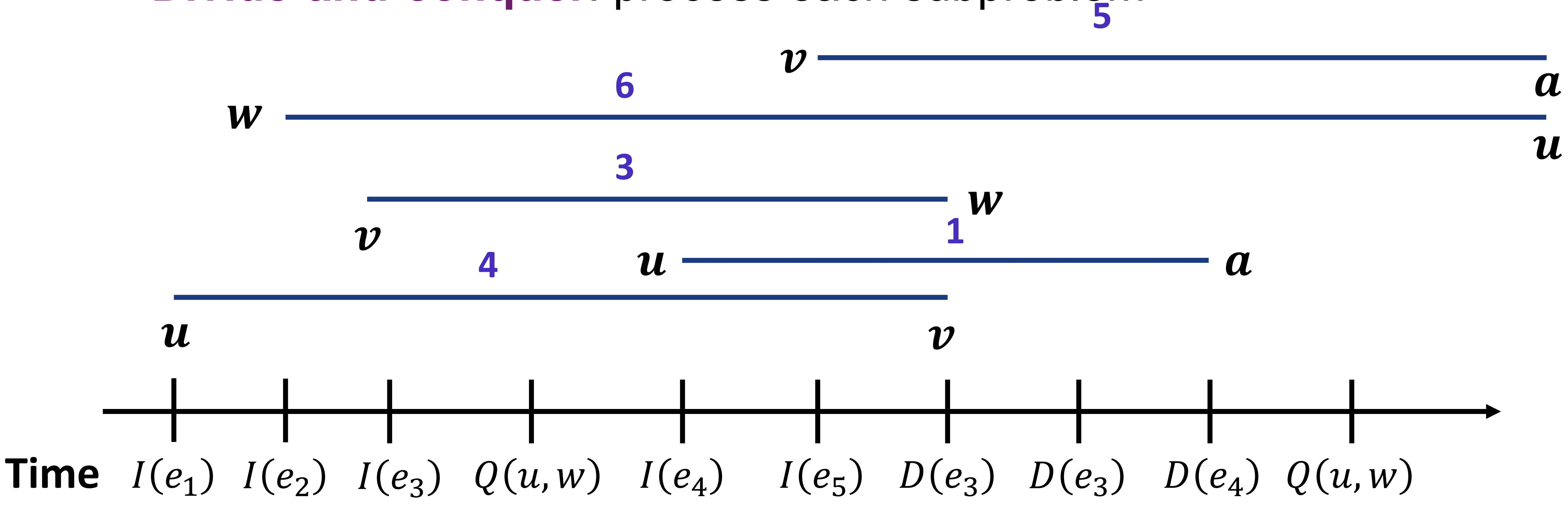
Offline-Dynamic Connectivity

- We will use the offline dynamic minimum spanning tree algorithm of Eppstein '92
- Geometric representation of the problem



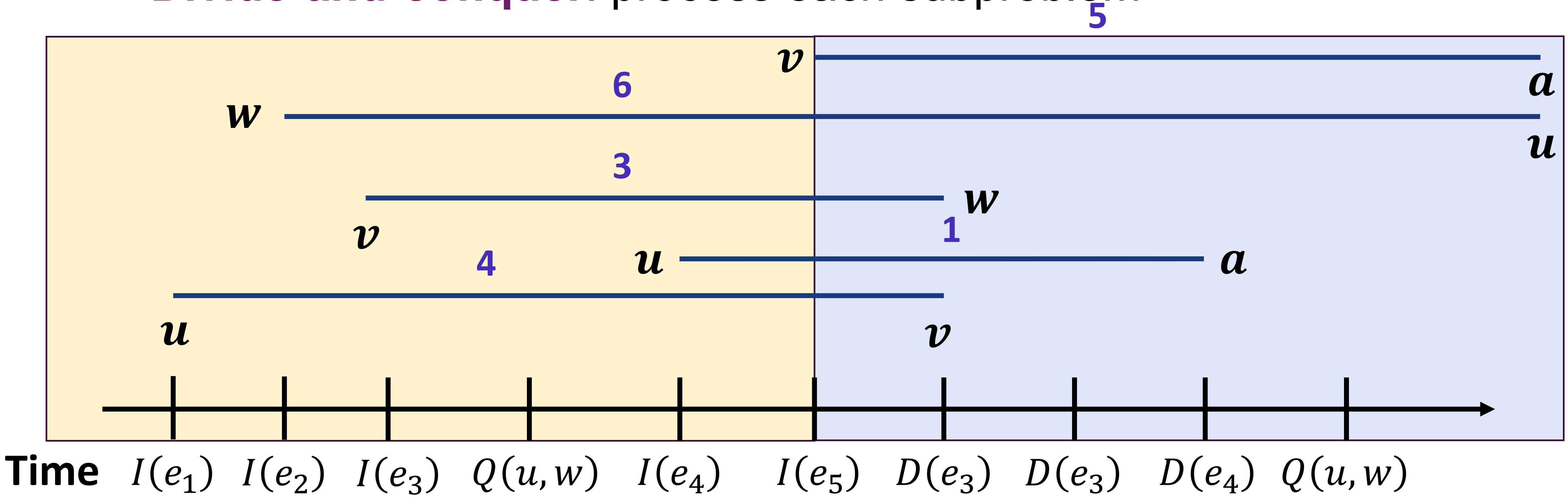
Offline-Dynamic Connectivity

- Geometric representation of the problem
- **Divide-and-conquer**: process each subproblem



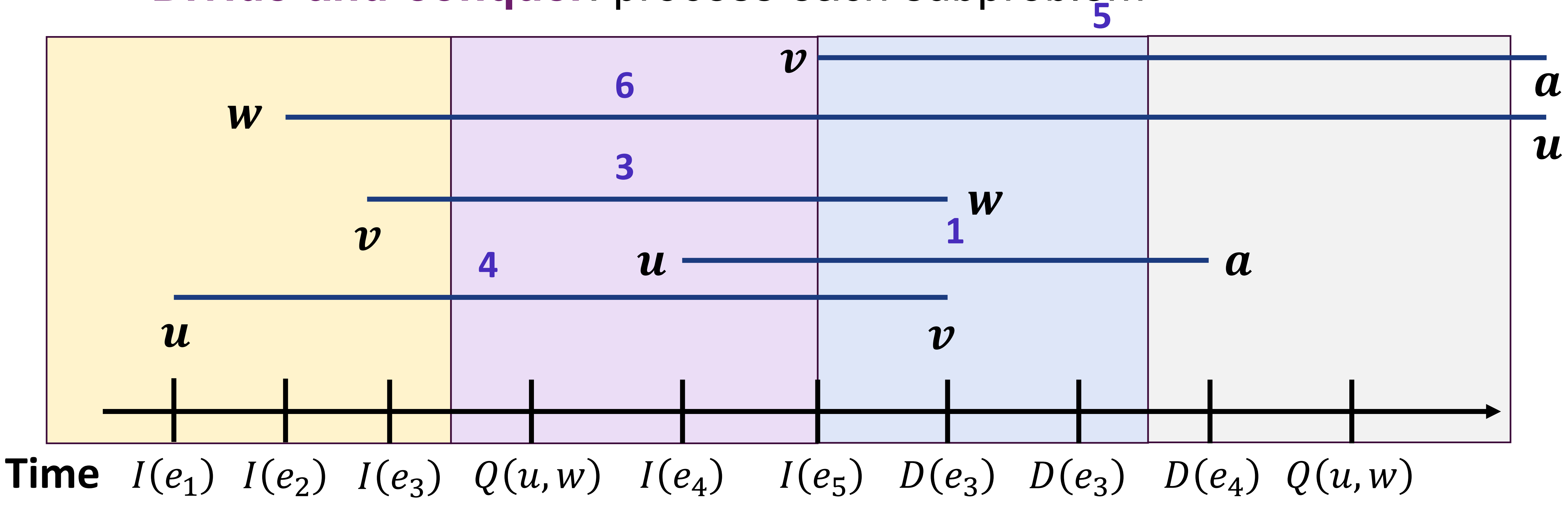
Offline-Dynamic Connectivity

- Geometric representation of the problem
- **Divide-and-conquer**: process each subproblem



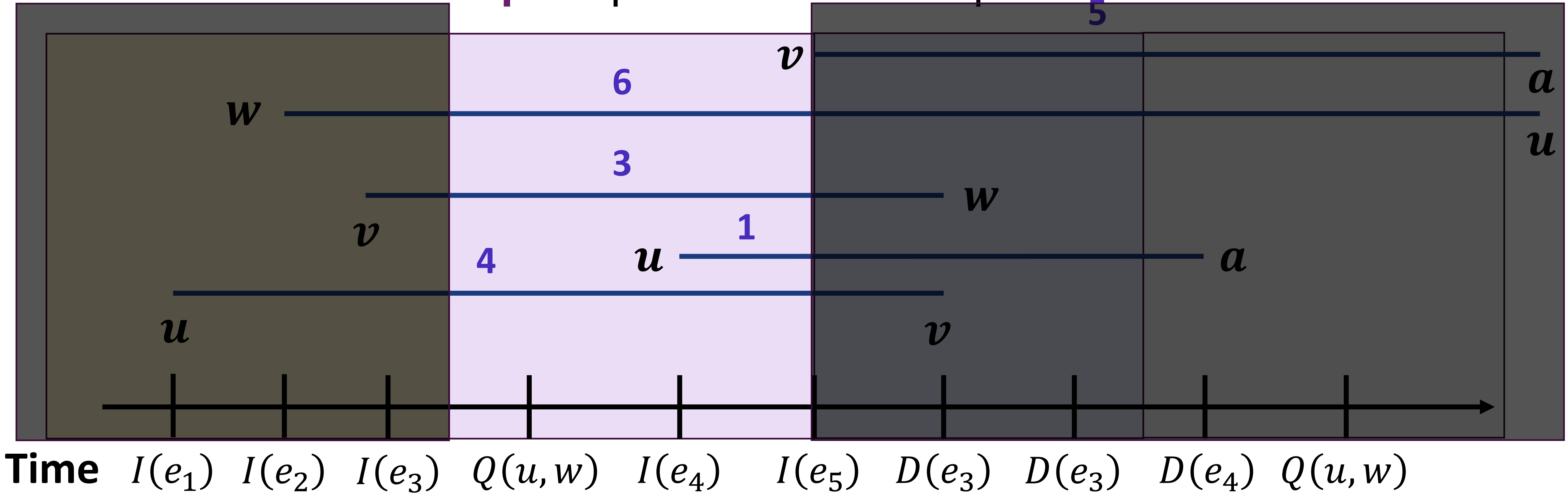
Offline-Dynamic Connectivity

- Geometric representation of the problem
- **Divide-and-conquer**: process each subproblem



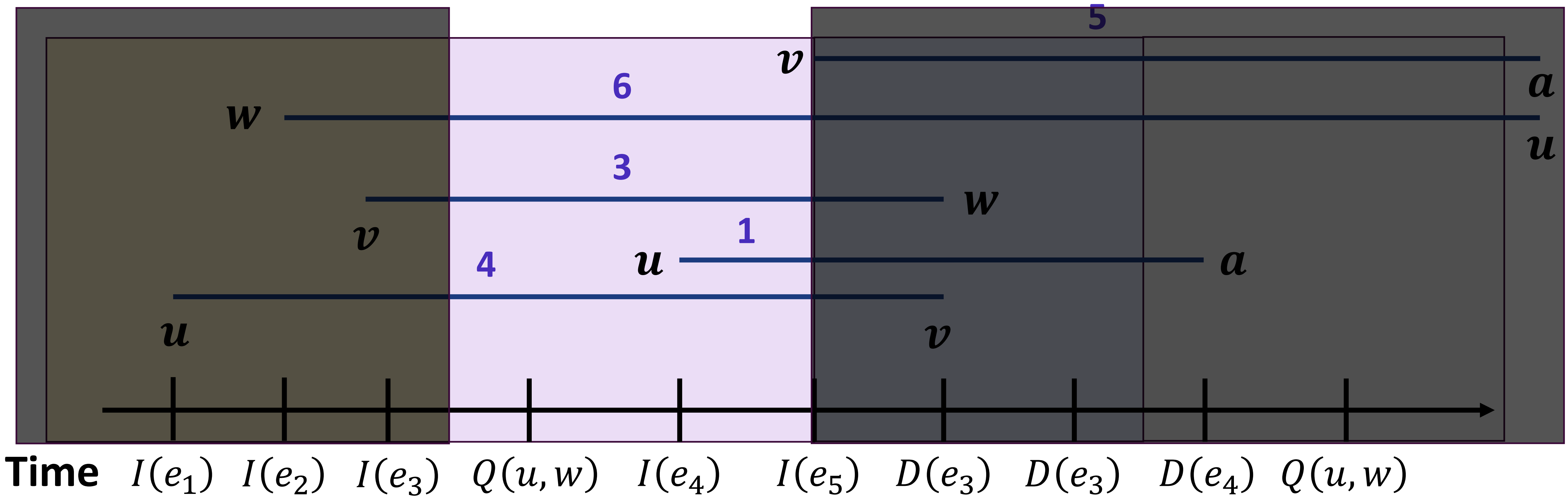
Offline-Dynamic Connectivity

- Geometric representation of the problem
- **Divide-and-conquer**: process each subproblem



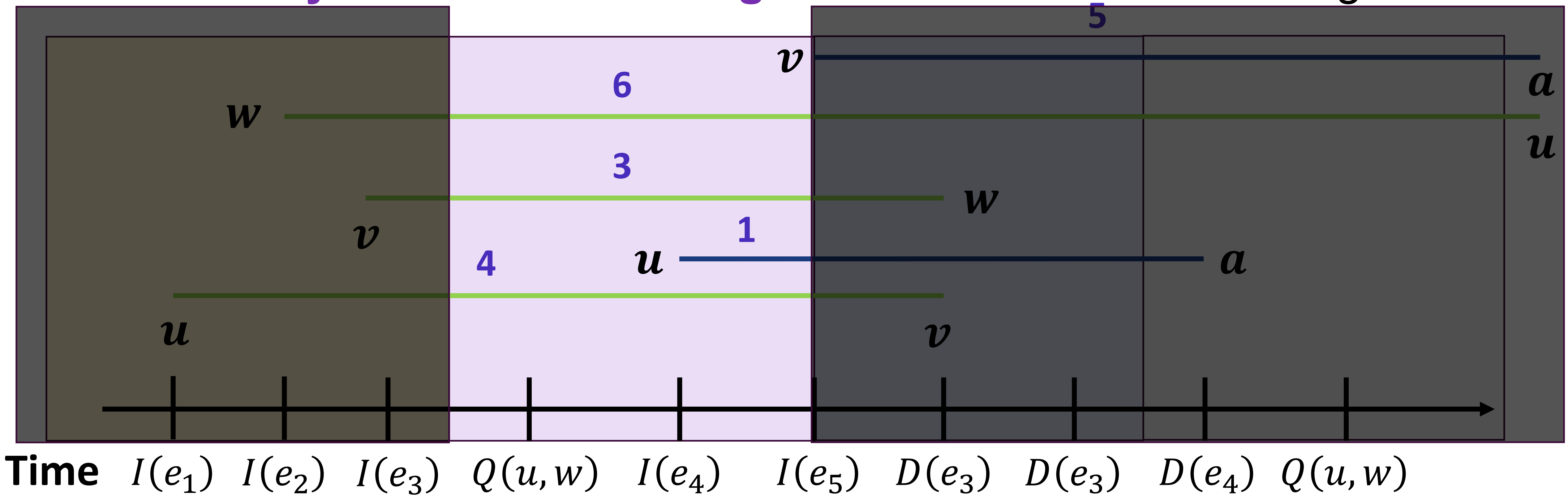
Offline-Dynamic Connectivity

- First, consider all **permanent edges** (edges that go across the subproblem)



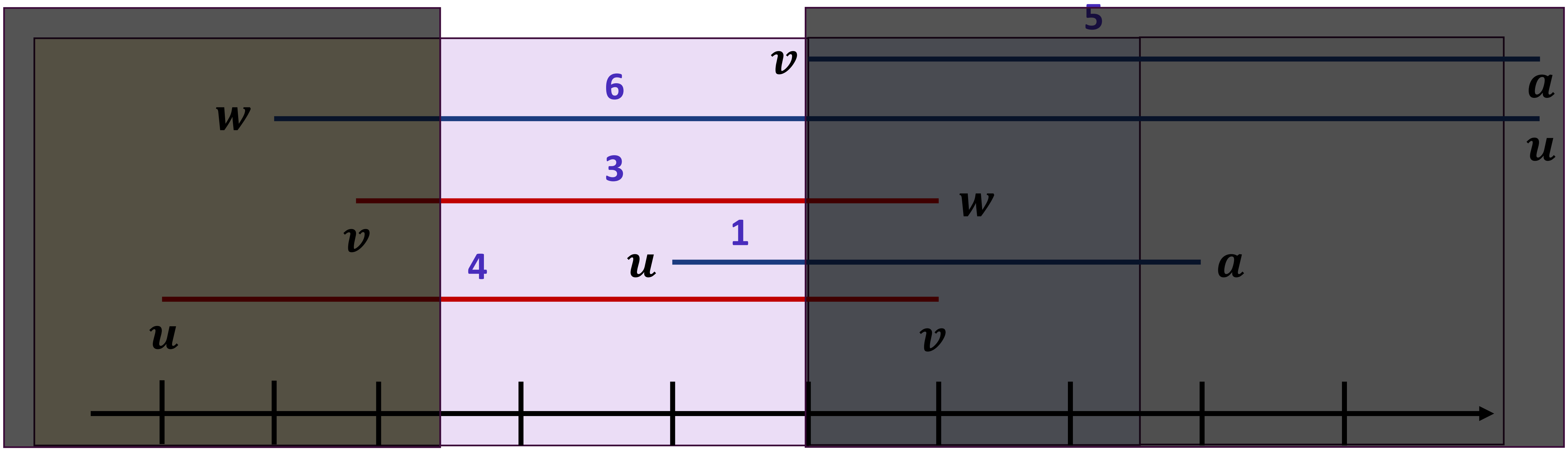
Offline-Dynamic Connectivity

- First, consider all **permanent edges** (edges that go across the subproblem)
- Run **any linear time MST algorithm** on all considered edges



Offline-Dynamic Connectivity

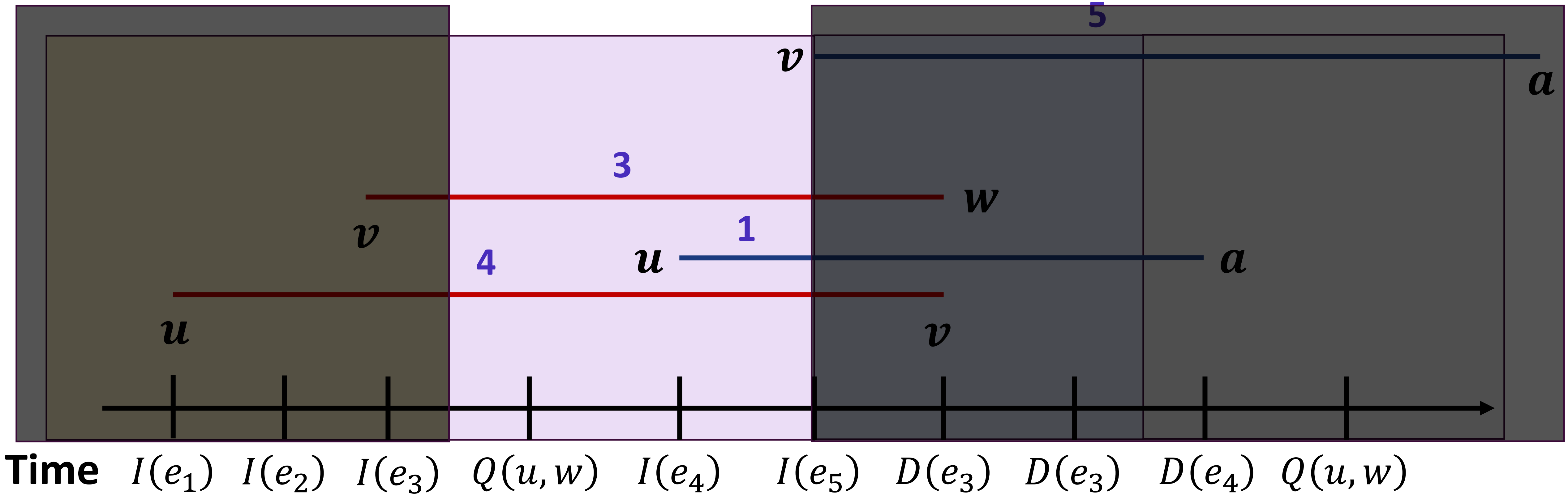
- Run **any linear time MST algorithm** on all considered edges
- **Red edges** are in the MST



Time $I(e_1)$ $I(e_2)$ $I(e_3)$ $Q(u,w)$ $I(e_4)$ $I(e_5)$ $D(e_3)$ $D(e_3)$ $D(e_4)$ $Q(u,w)$

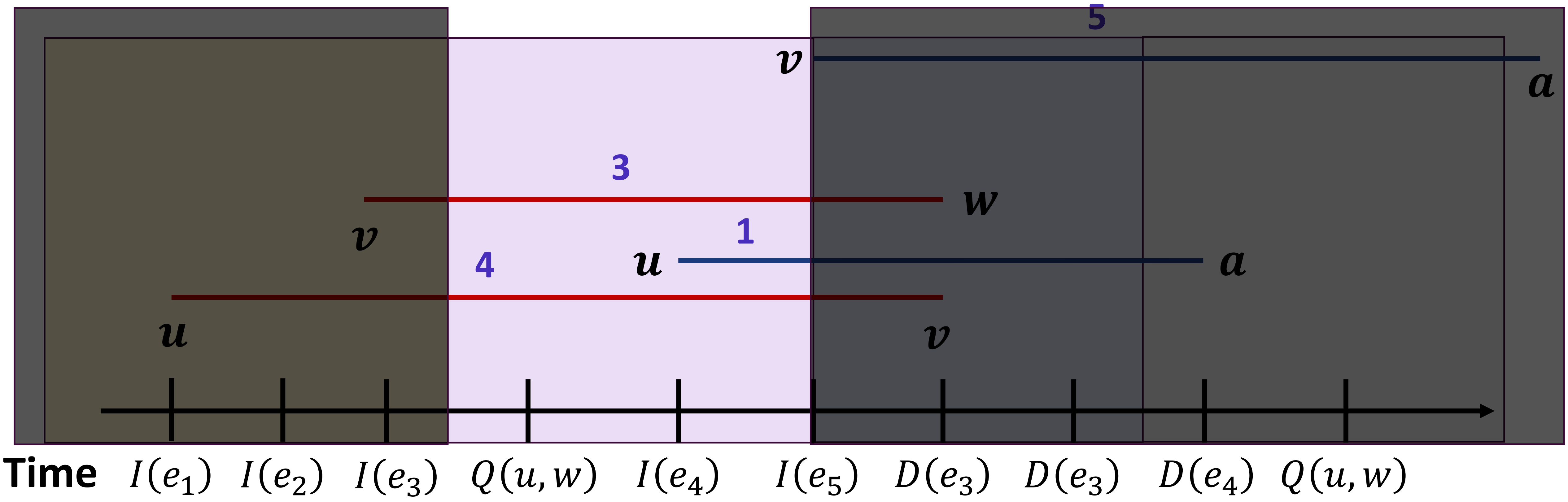
Offline-Dynamic Connectivity

- Run **any linear time MST algorithm** on all considered edges
- **Red edges** are in the MST; **Delete permanent edges not red**



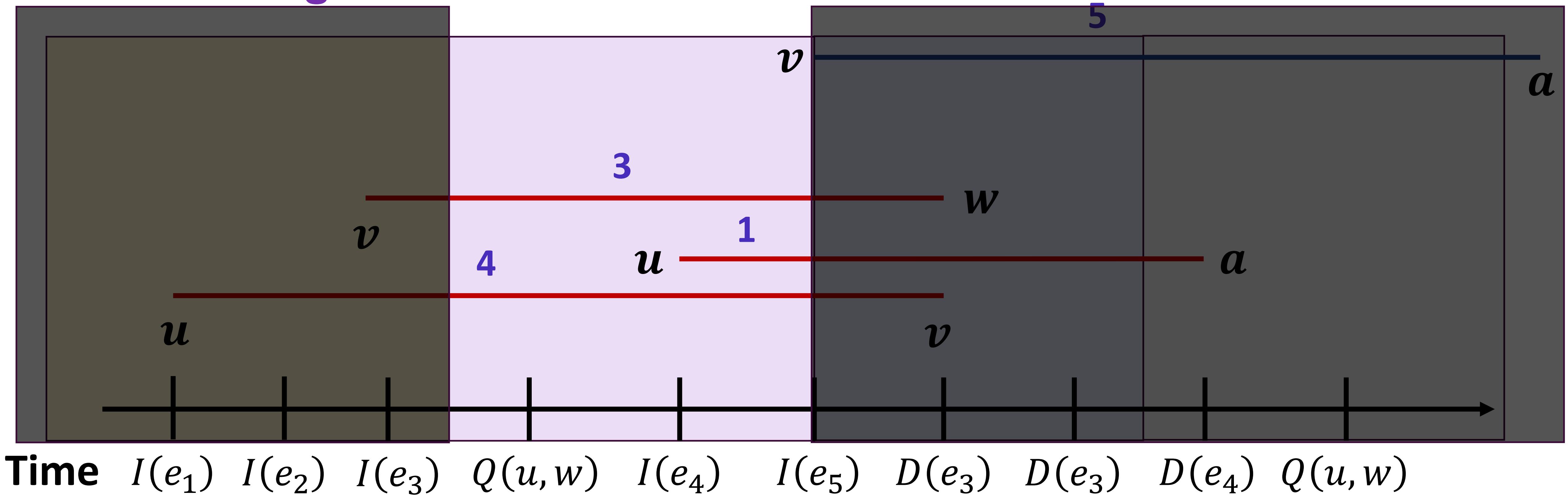
Offline-Dynamic Connectivity

- **Red edges** are in the MST; **Delete permanent edges not red**
- **Now consider all edges in subproblem**



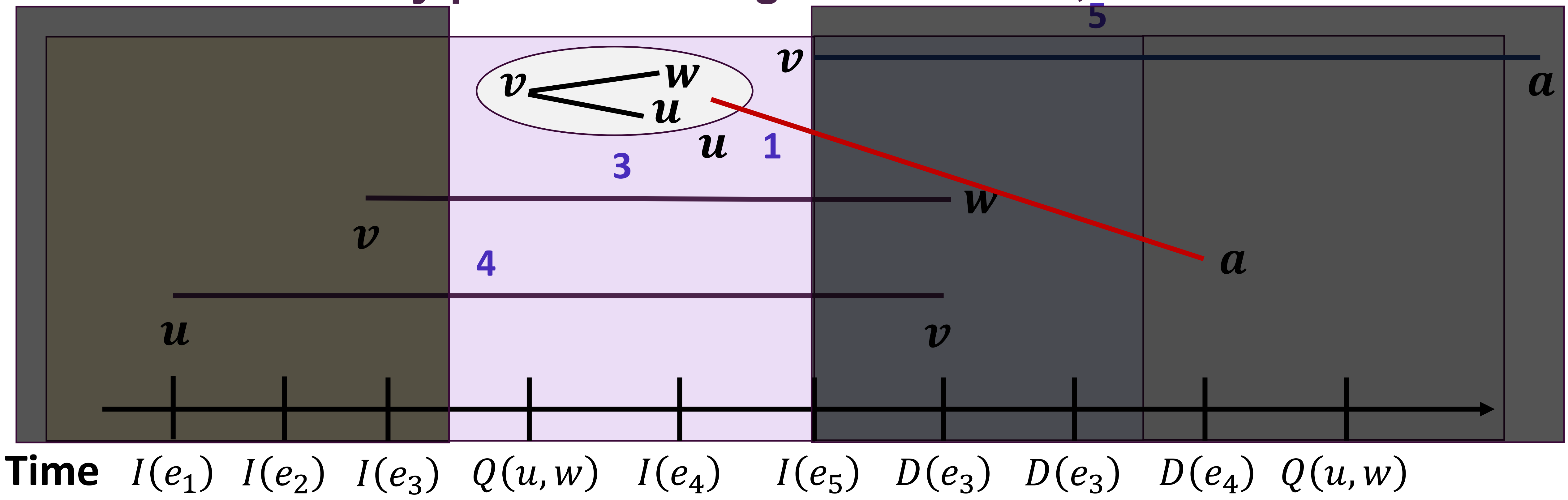
Offline-Dynamic Connectivity

- **Red edges** are in the MST; **Delete permanent edges not red**
- **Now consider all edges in subproblem;** Run **any linear time MST algorithm**



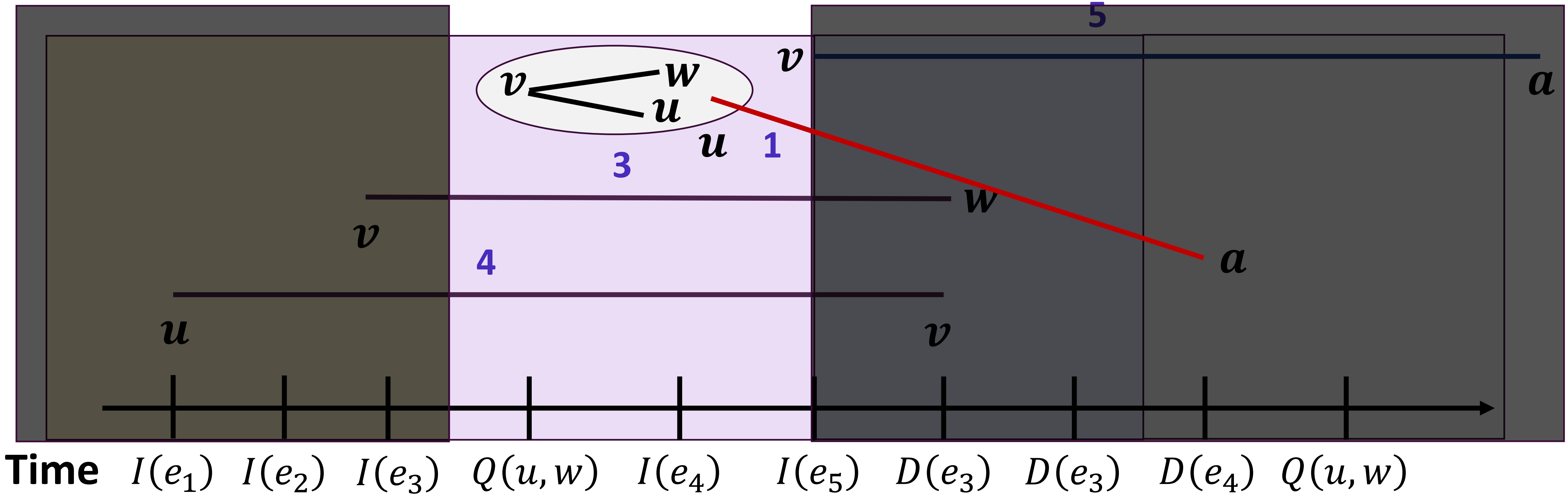
Offline-Dynamic Connectivity

- **Now consider all edges in subproblem;** Run **any linear time MST algorithm**
- **Contract any permanent edges in the MST;** link-cut tree



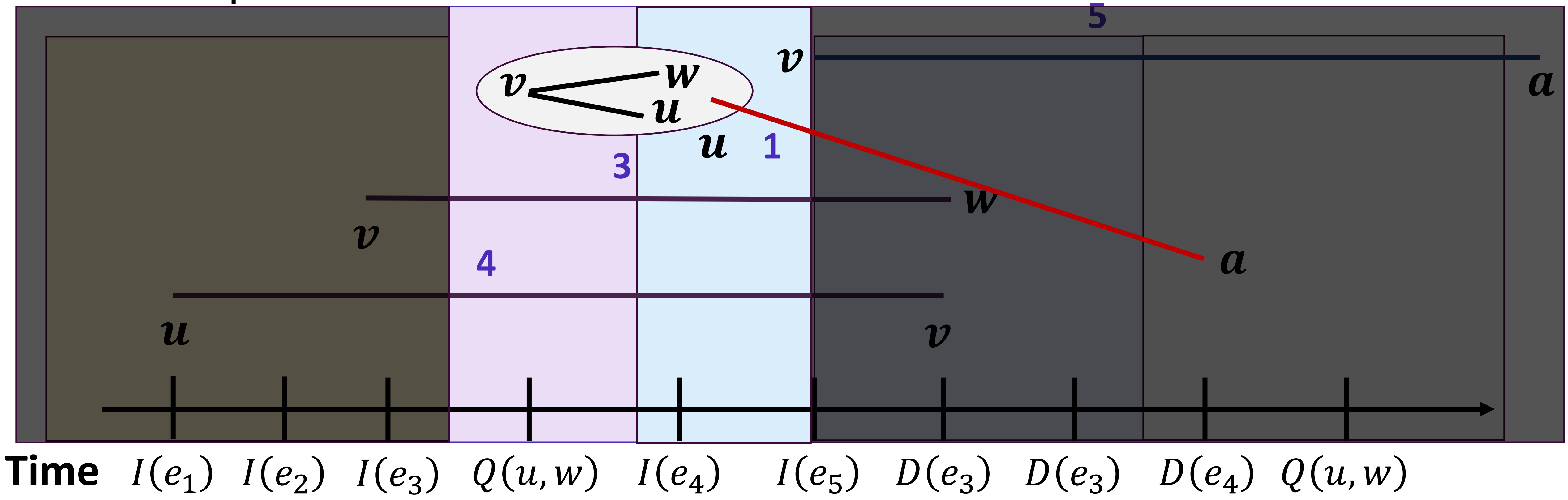
Offline-Dynamic Connectivity

- **Pass data structure to next smaller subproblem (persistence)**



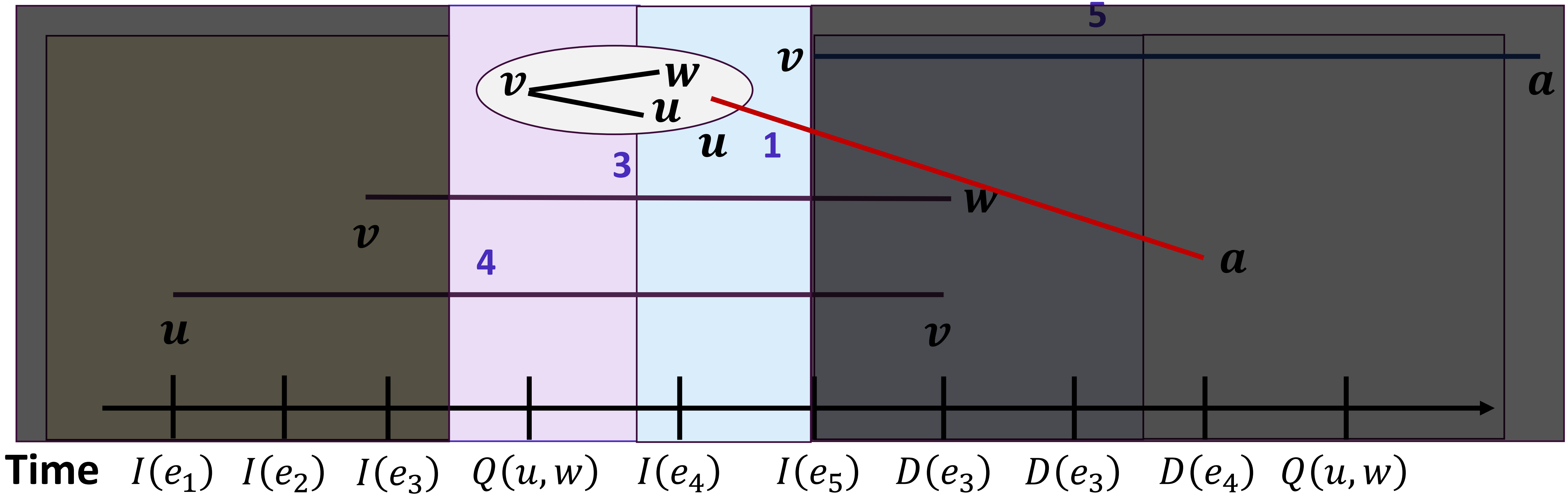
Offline-Dynamic Connectivity

- Pass data structure to next smaller subproblem (persistence)
- Consider **non-contracted** and **not deleted edges** in subproblem



Offline-Dynamic Connectivity

- For queries, look at the data structure and edges of **smallest subproblem containing the query**



Offline-Dynamic Connectivity

- Assume link-cut tree and persistence such that each subproblem with $\boldsymbol{T}$ total permanent and non-permanent edges takes $\boldsymbol{O(T)}$ **time**

Offline-Dynamic Connectivity

- Assume link-cut tree and persistence such that each subproblem with $\boldsymbol{T}$ total permanent and non-permanent edges takes $\boldsymbol{O(T)}$ **time**
- First, consider permanent edges, what happens to them?

Offline-Dynamic Connectivity

- Assume link-cut tree and persistence such that each subproblem with $\boldsymbol{T}$ total permanent and non-permanent edges takes $\boldsymbol{O(T)}$ **time**
- First, consider permanent edges, what happens to them?
  - They are either **deleted**

Offline-Dynamic Connectivity

- Assume link-cut tree and persistence such that each subproblem with $\boldsymbol{T}$ total permanent and non-permanent edges takes $\boldsymbol{O(T)}$ **time**
- First, consider permanent edges, what happens to them?
  - They are either **deleted, contracted,**

Offline-Dynamic Connectivity

- Assume link-cut tree and persistence such that each subproblem with $\boldsymbol{T}$ total permanent and non-permanent edges takes $\boldsymbol{O(T)}$ **time**
- First, consider permanent edges, what happens to them?
  - They are either **deleted, contracted,** or **neither**

Offline-Dynamic Connectivity

- Assume link-cut tree and persistence such that each subproblem with $\boldsymbol{T}$ total permanent and non-permanent edges takes $\boldsymbol{O(T)}$ **time**
- First, consider permanent edges, what happens to them?
  - They are either **deleted, contracted,** or **neither**
  - **Deleted and contracted edges charged to previous level**

Offline-Dynamic Connectivity

- Assume link-cut tree and persistence such that each subproblem with T total permanent and non-permanent edges takes $O(T)$ **time**
- First, consider permanent edges, what happens to them?
    - They are either **deleted, contracted,** or **neither**
    - **Deleted and contracted edges charged to previous level**
    - **Neither edges are charged to a non-permanent edge in window**

Offline-Dynamic Connectivity

- Assume link-cut tree and persistence such that each subproblem with $\boldsymbol{T}$ total permanent and non-permanent edges takes **$\boldsymbol{O(T)}$ time**
- First, consider permanent edges, what happens to them?
  - They are either **deleted, contracted,** or **neither**
  - **Deleted and contracted edges charged to previous level**
  - **Neither edges are charged to a non-permanent edge in window**
- Thus, $3T$ operations in window with T updates

Offline-Dynamic Connectivity

- Assume link-cut tree and persistence such that each subproblem with $\boldsymbol{T}$ total permanent and non-permanent edges takes $\boldsymbol{O(T)}$ **time**
- First, consider permanent edges, what happens to them?
  - They are either **deleted, contracted,** or **neither**
  - **Deleted and contracted edges charged to previous level**
  - **Neither edges are charged to a non-permanent edge in window**
- Thus, $3T$ operations in window with T updates

Total Runtime: $\boldsymbol{O\left(T\log(T)\right)}$ by Master Theorem

Monte Carlo Oblivious Adversary Dynamic Connectivity

- Level Data Structure Algorithm of [Kapron-King-Mountjoy SODA '13]

Monte Carlo Oblivious Adversary Dynamic Connectivity

- Level Data Structure Algorithm of [Kapron-King-Mountjoy SODA '13]
- High-Level Idea:
  - Data structure for quickly determining: given a cut if there's an edge (whp) going in between the cut

Monte Carlo Oblivious Adversary Dynamic Connectivity

- Level Data Structure Algorithm of [Kapron-King-Mountjoy SODA '13]
- High-Level Idea:
  - Data structure for quickly determining: given a cut if there's an edge (whp) going in between the cut
  - Data structure for maintaining connected vertices

Monte Carlo Oblivious Adversary Dynamic Connectivity

- Level Data Structure Algorithm of [Kapron-King-Mountjoy SODA '13]
- High-Level Idea:
  - Data structure for quickly determining: given a cut if there's an edge (whp) going in between the cut
  - Data structure for maintaining connected vertices
  - Easy access to determine if vertices are in the same connected component

Monte Carlo Oblivious Adversary Dynamic Connectivity

- Data structure:
  - **Euler tour tree**: Operations and runtimes:
    - Check whether two vertices u and v are in the same tree: $O(\log n)$ time

Monte Carlo Oblivious Adversary Dynamic Connectivity

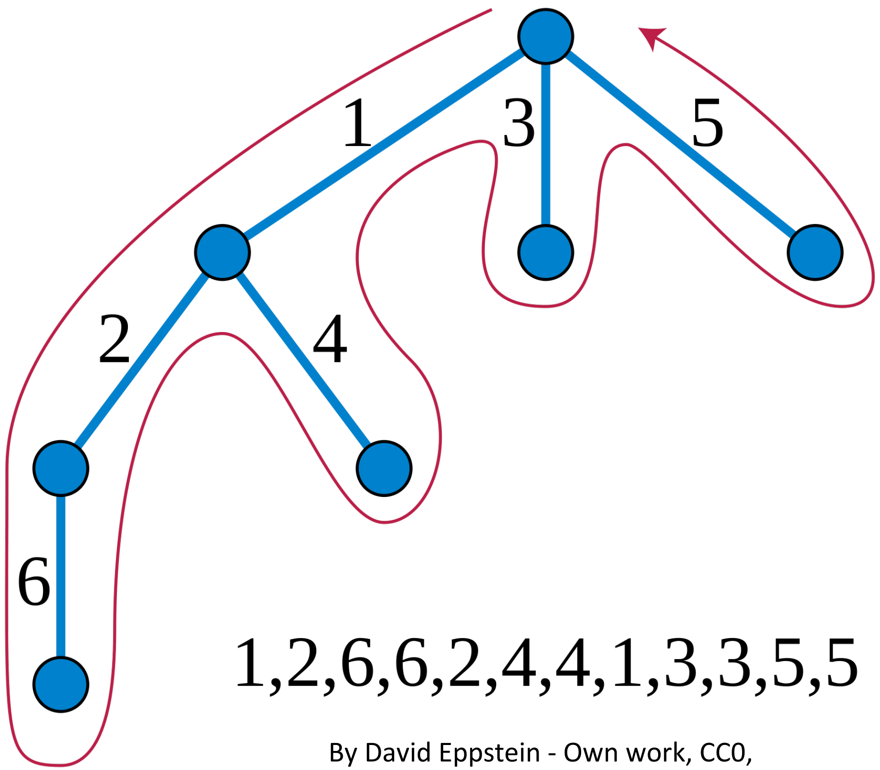
- Data structure:
  - **Euler tour tree**: Operations and runtimes:
    - Check whether two vertices u and v are in the same tree: $O(\log n)$ time
    - Break a cycle in $O(\log n)$ time

Monte Carlo Oblivious Adversary Dynamic Connectivity

- Data structure:
  - **Euler tour tree**: Operations and runtimes:
    - Check whether two vertices u and v are in the same tree: $O(\log n)$ time
    - Break a cycle in $O(\log n)$ time
    - Find SUM or XOR (any commutative, associative operation) of subtree in $O(\log n)$ time

Monte Carlo Oblivious Adversary Dynamic Connectivity

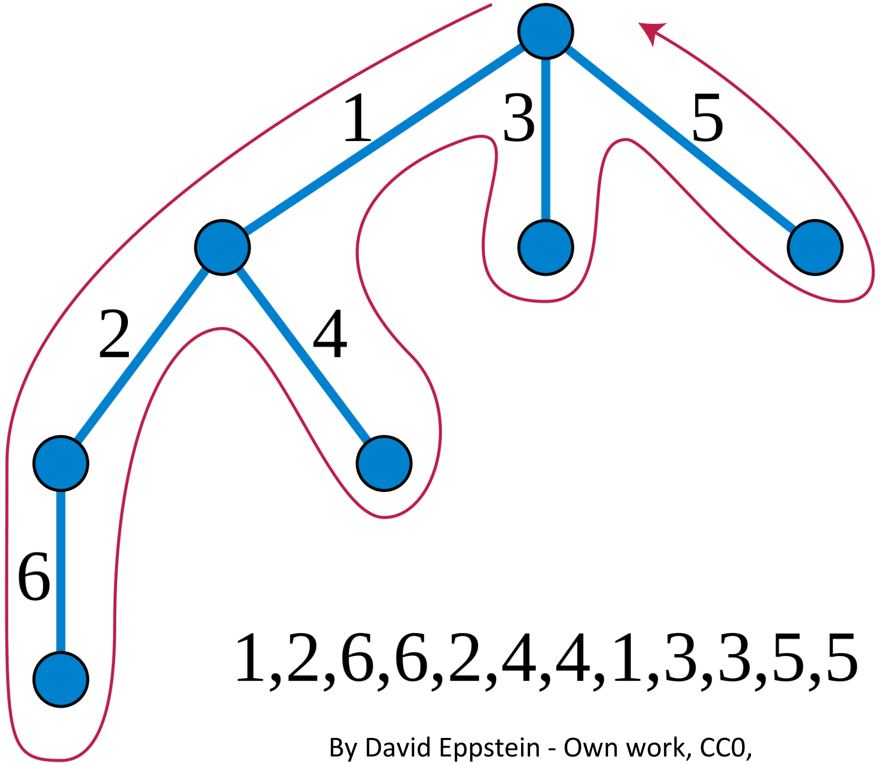
- **Euler tour tree** (Tarjan-Vishkin ‘94) high level description:



1,2,6,6,2,4,4,1,3,3,5,5

Monte Carlo Oblivious Adversary Dynamic Connectivity

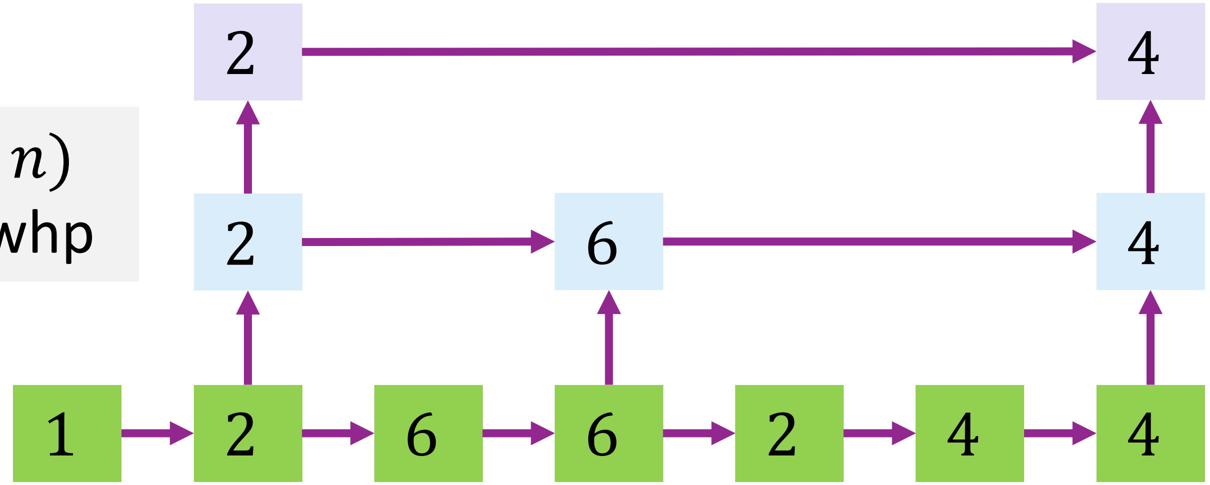
- **Euler tour tree** (Tarjan-Vishkin ‘94) high level description:



By David Eppstein - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=25178326

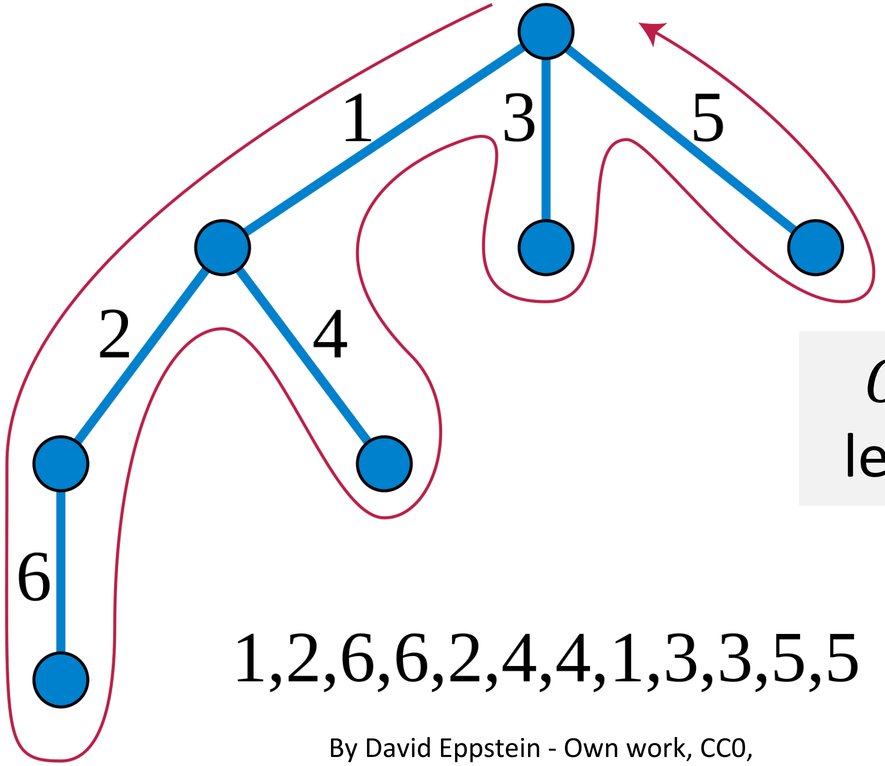
Can be implemented using Skip-List

$O(\log n)$ levels whp



Monte Carlo Oblivious Adversary Dynamic Connectivity

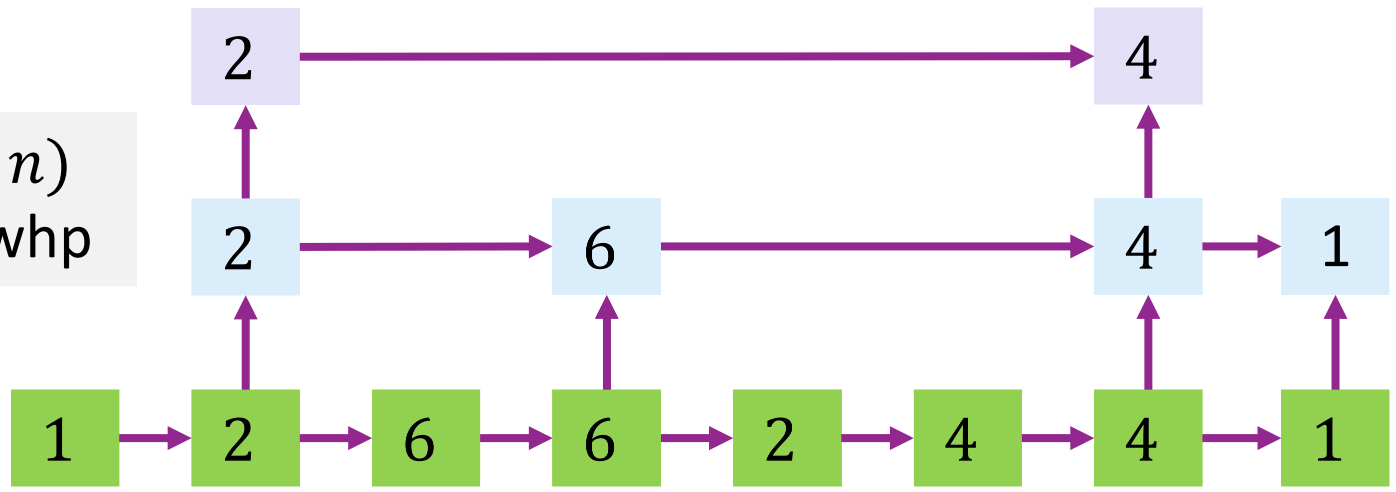
- **Euler tour tree allows you to remove a subtree very easily**



1,2,6,6,2,4,4,1,3,3,5,5

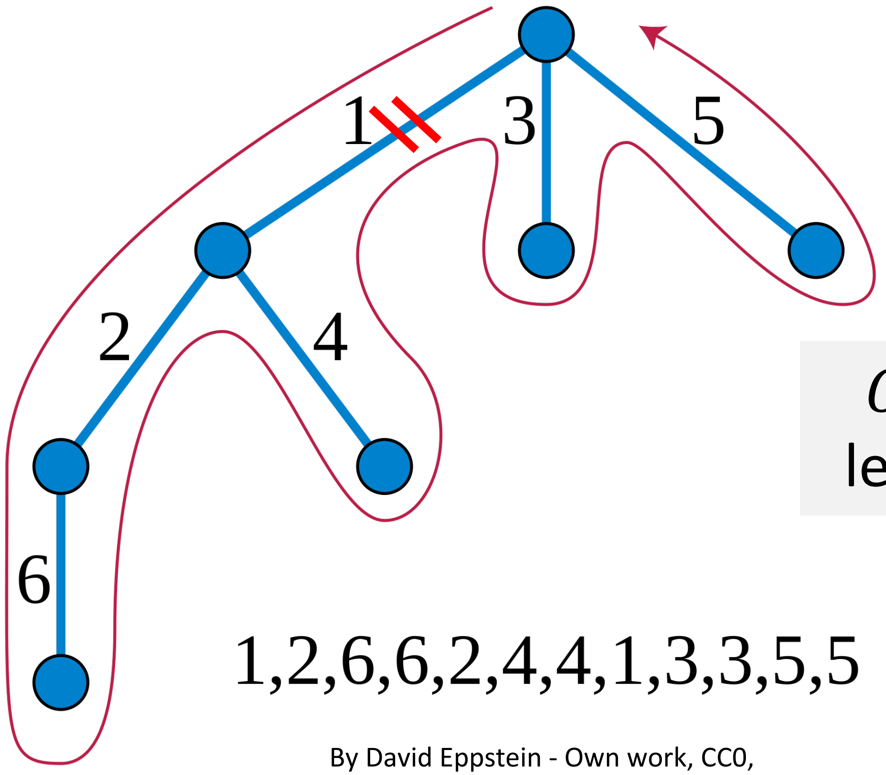
Can be implemented using Skip-List

$O(\log n)$
levels whp



Monte Carlo Oblivious Adversary Dynamic Connectivity

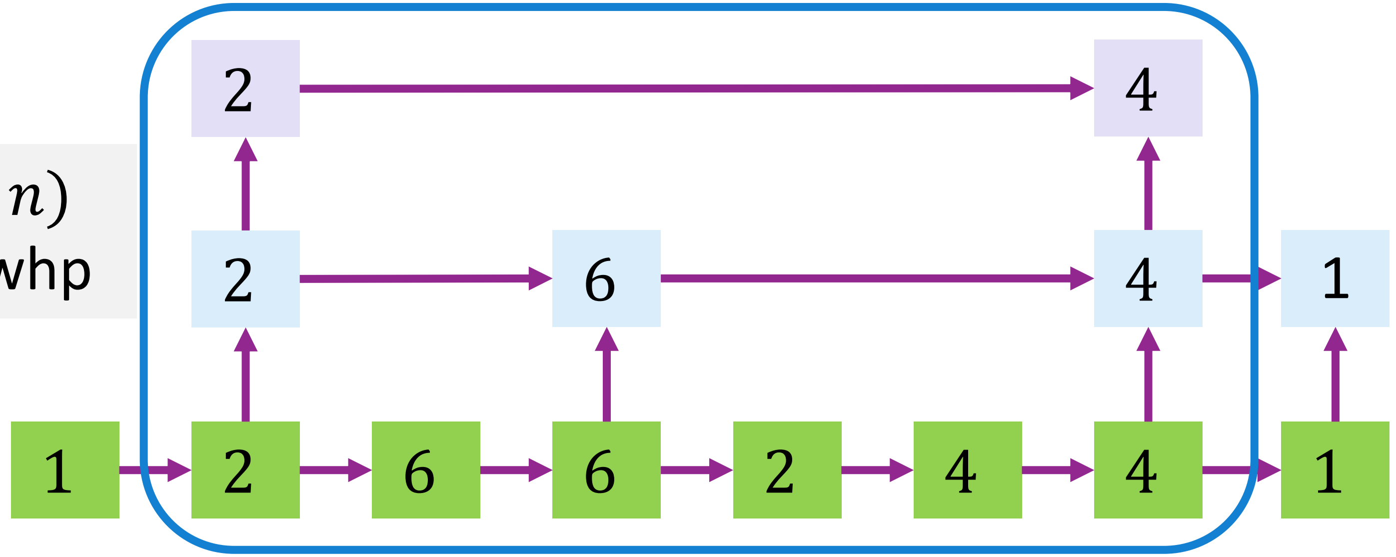
- **Delete edge 1**



1,2,6,6,2,4,4,1,3,3,5,5

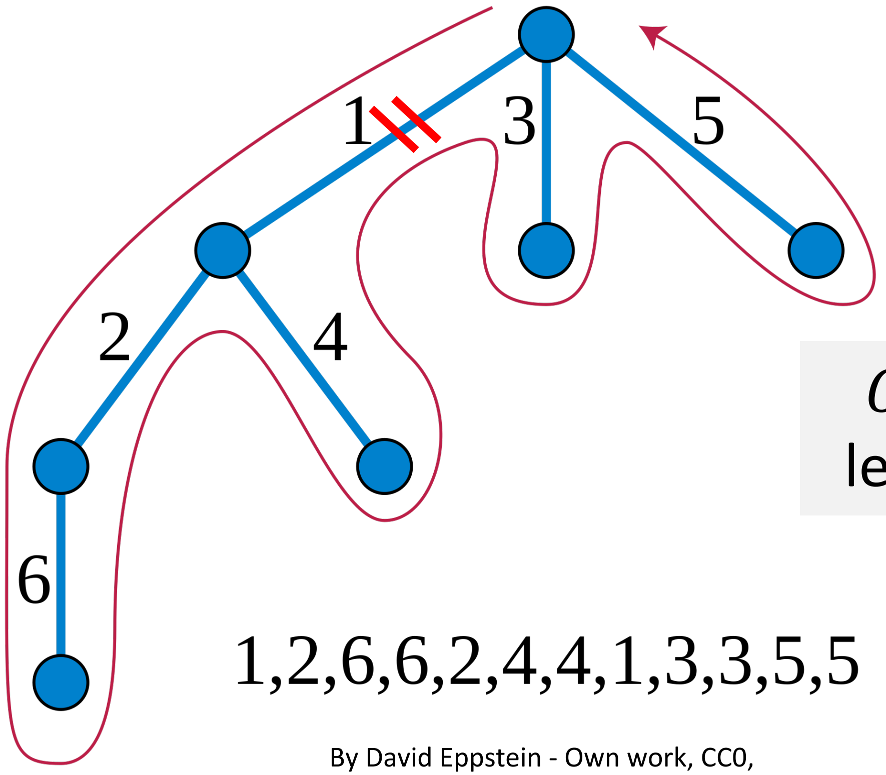
Can be implemented using Skip-List

$O(\log n)$
levels whp



Monte Carlo Oblivious Adversary Dynamic Connectivity

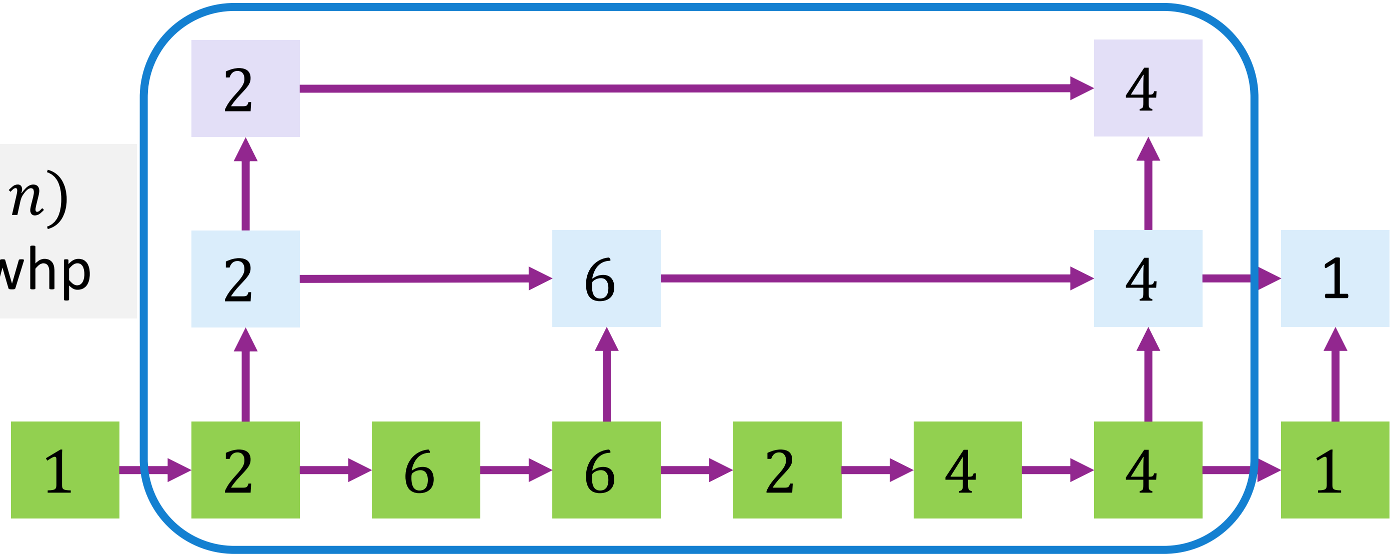
- **Delete edge 1**



1,2,6,6,2,4,4,1,3,3,5,5

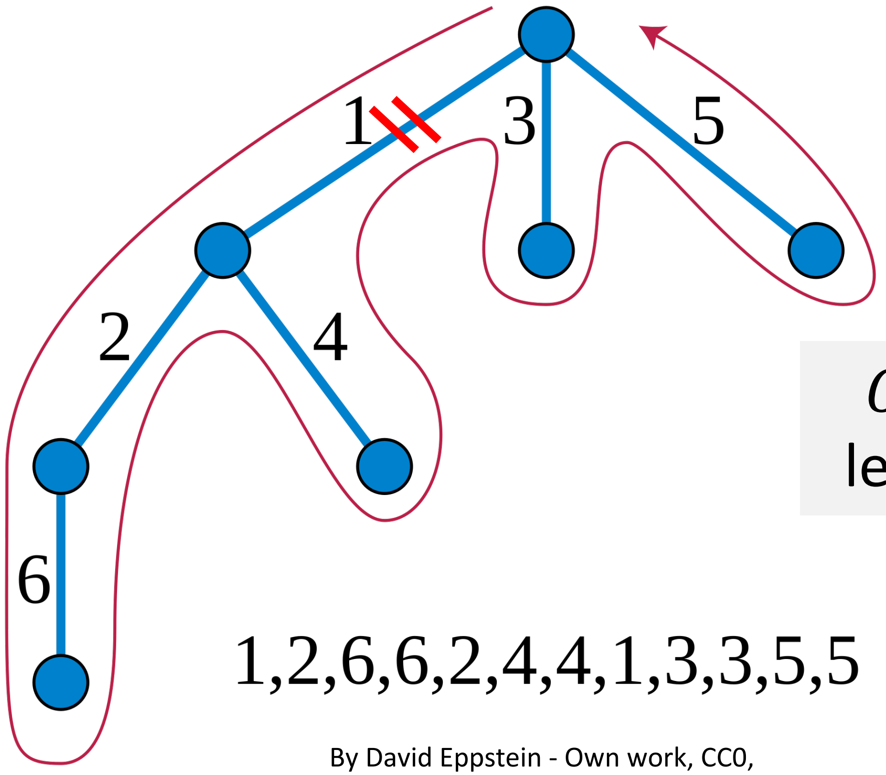
Remove relevant contiguous section of skip-list and link together the ends

$O(\log n)$
levels whp

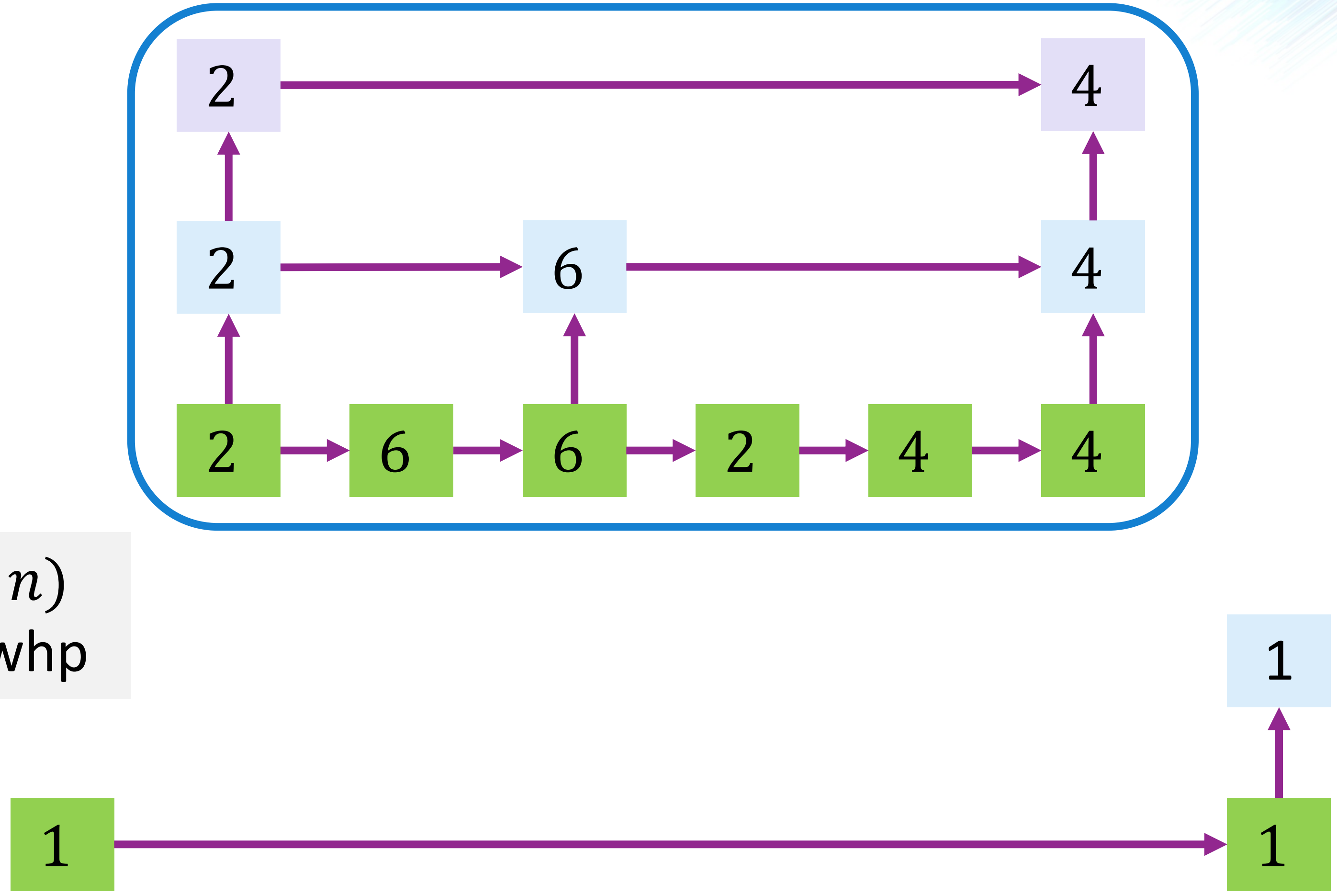


Monte Carlo Oblivious Adversary Dynamic Connectivity

- **Delete edge 1**



$O(\log n)$
levels whp



Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- Start from initially **empty graph**

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

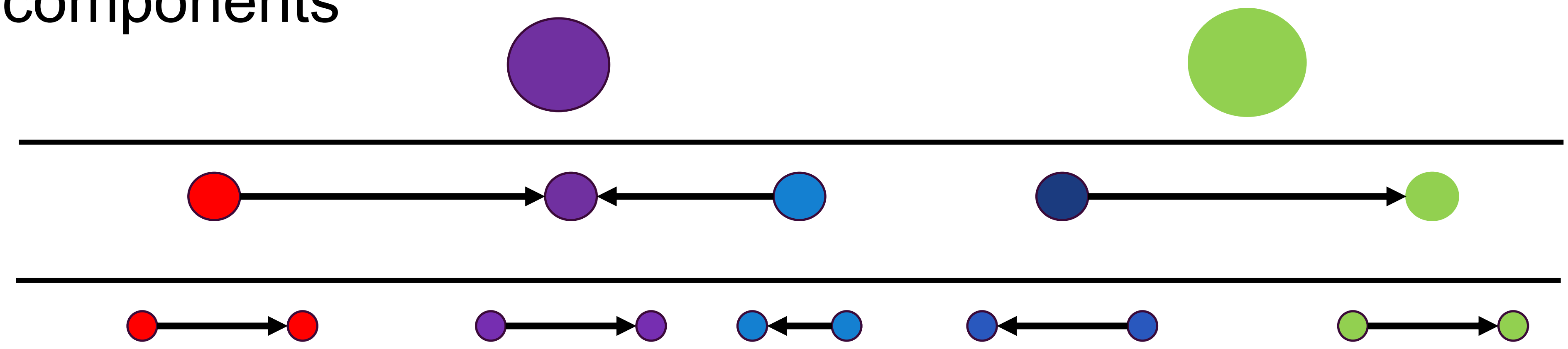
- Start from initially **empty graph**
- Maintain spanning trees of connected components using **Euler Tour Trees**

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- Start from initially **empty graph**
- Maintain spanning trees of connected components using **Euler Tour Trees**
- Maintain Monte Carlo Boruvka tree (MST) data structure where in each level, you add a new edge between not connected components

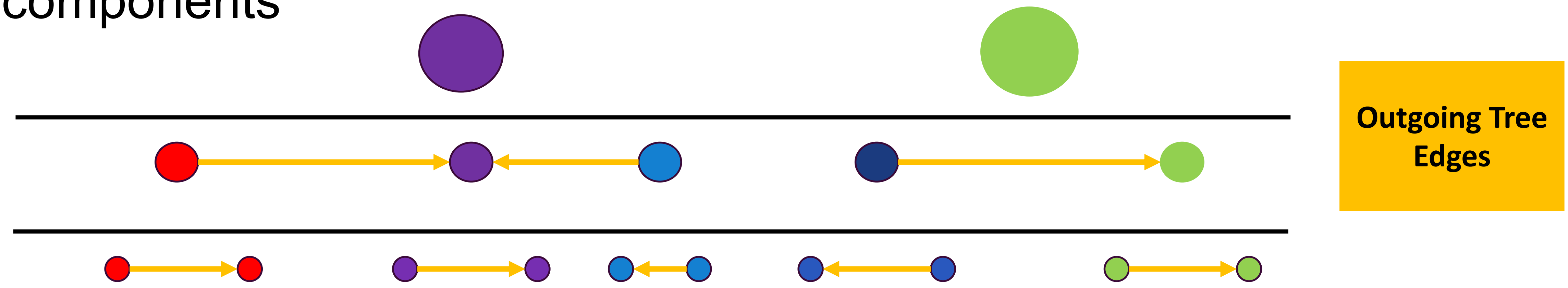
Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- Start from initially **empty graph**
- Maintain spanning trees of connected components using **Euler Tour Trees**
- Maintain Monte Carlo Boruvka tree (MST) data structure where in each level, you add a new edge between not connected components



Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

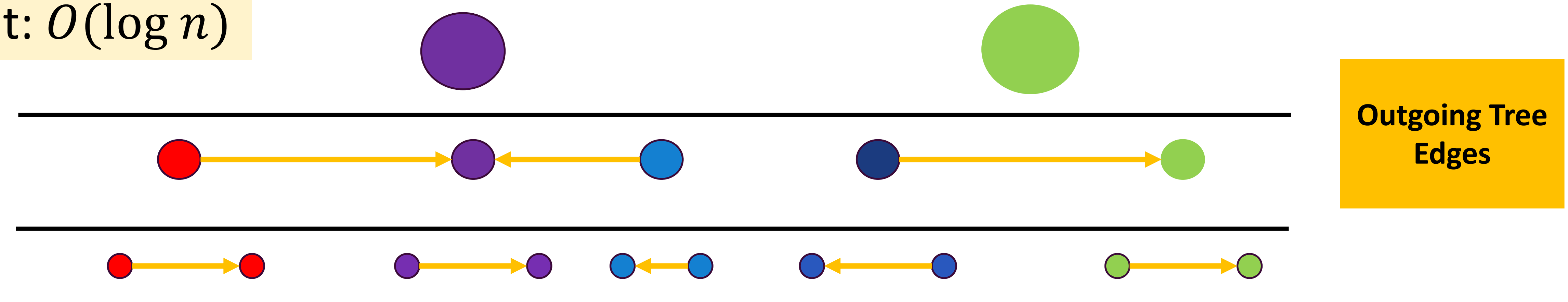
- Start from initially **empty graph**
- Maintain spanning trees of connected components using **Euler Tour Trees**
- Maintain Monte Carlo Boruvka tree (MST) data structure where in each level, you add a new edge between not connected components



Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- Want: $\text{poly}(\log n)$ runtime

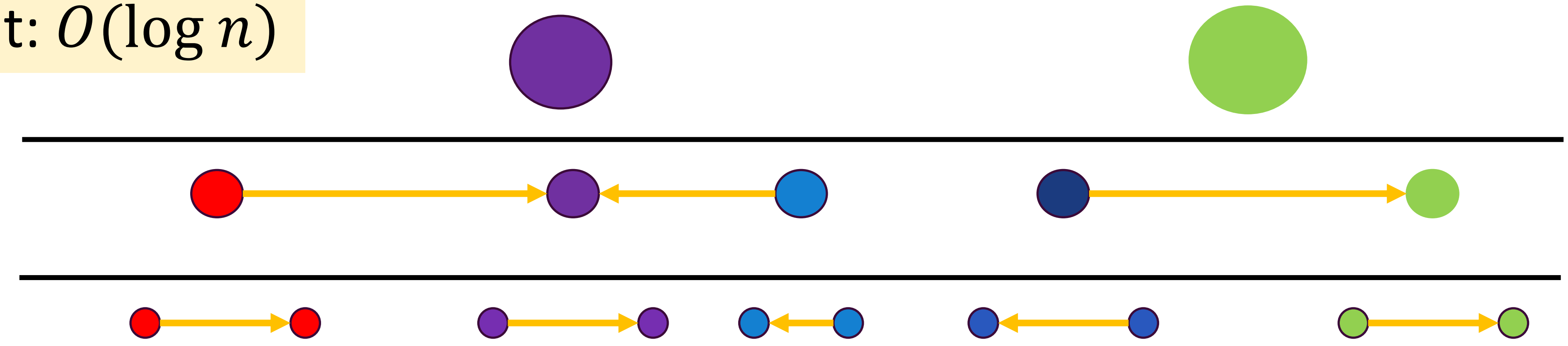
Height: $O(\log n)$



Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- Want: $\text{poly}(\log n)$ runtime
- On edge insertion: if between two disconnected components on **top level**
  - Insert into **every level**

Height: $O(\log n)$

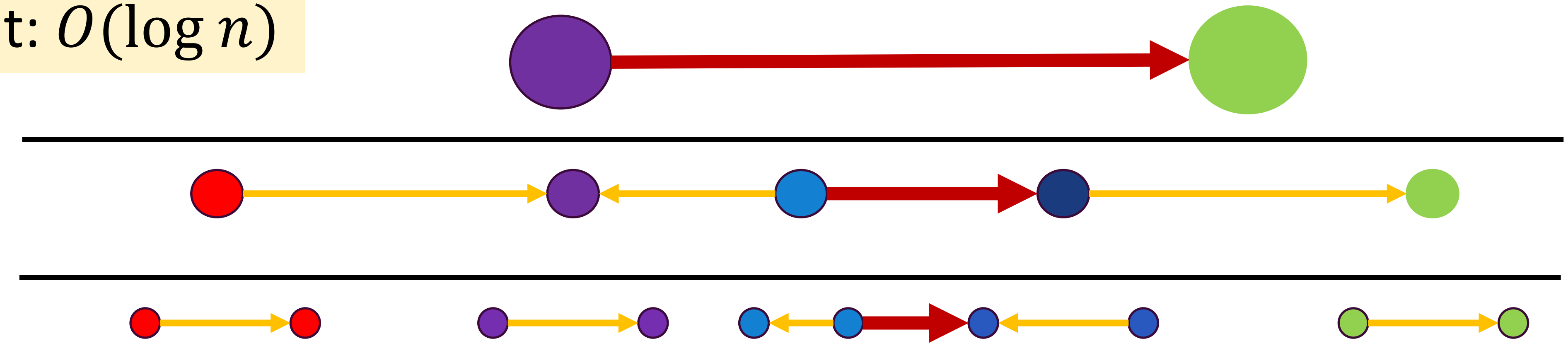


Outgoing Tree Edges

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- Want: $\mathrm{poly}(\log n)$ runtime
- On edge insertion: if between two disconnected components on **top level**
  - Insert into **every level (still an MST)**

Height: $O(\log n)$

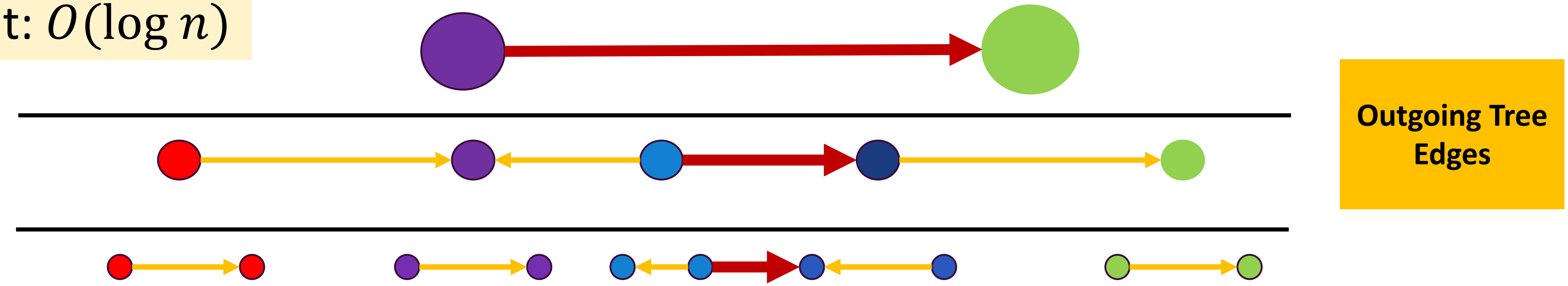


Outgoing Tree Edges

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- Want: $\mathrm{poly}(\log n)$ runtime
- On edge insertion: if between two disconnected components on **top level**
  - Insert into **every level (still an MST)**
- Update **XOR data structure**

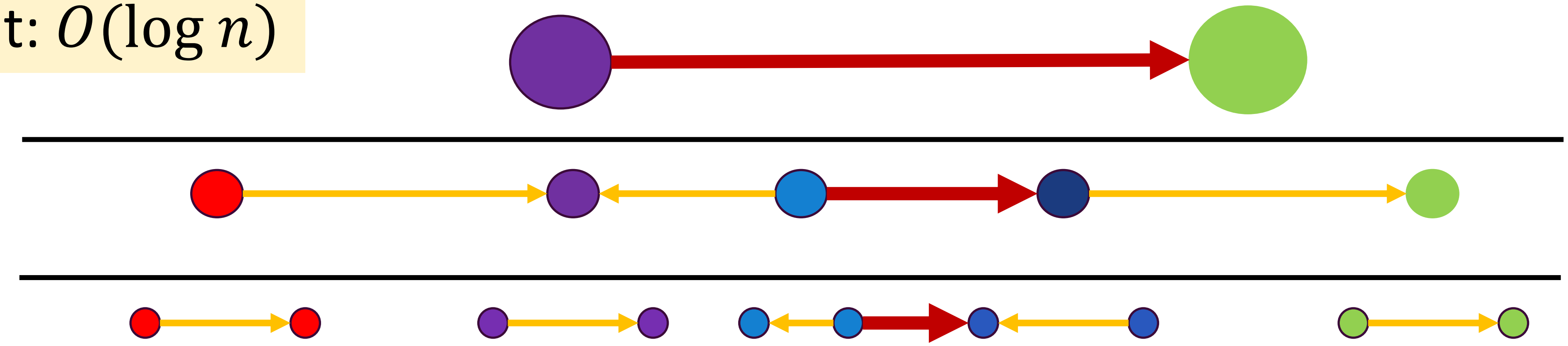
Height: $O(\log n)$



Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- On edge deletion: if deletion of an **outgoing tree edge**:
  - Delete from every level

Height: $O(\log n)$

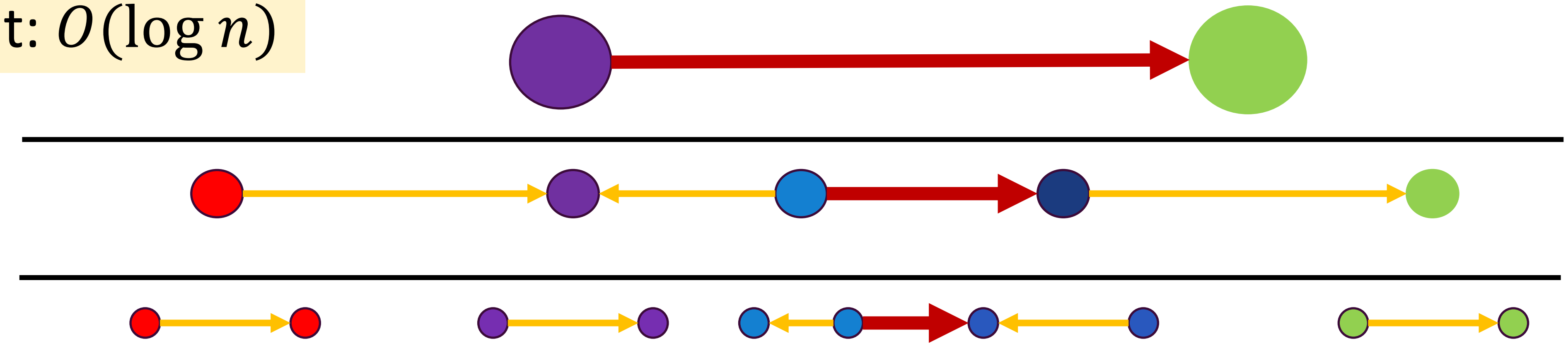


Outgoing Tree Edges

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- On edge deletion: if deletion of an **outgoing tree edge**:
  - Delete from every level
  - **Search for replacement edge from XOR data structure**

Height: $O(\log n)$

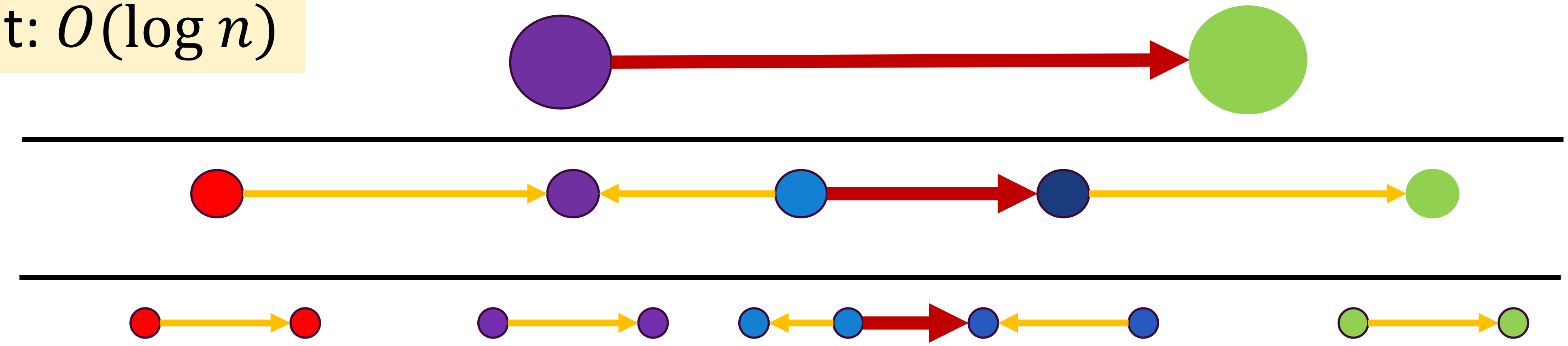


Outgoing Tree Edges

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- On edge deletion: if deletion of an **outgoing tree edge**:
  - Delete from every level
  - **Search for replacement edge from XOR data structure**
- Update **XOR data structure**

Height: $O(\log n)$

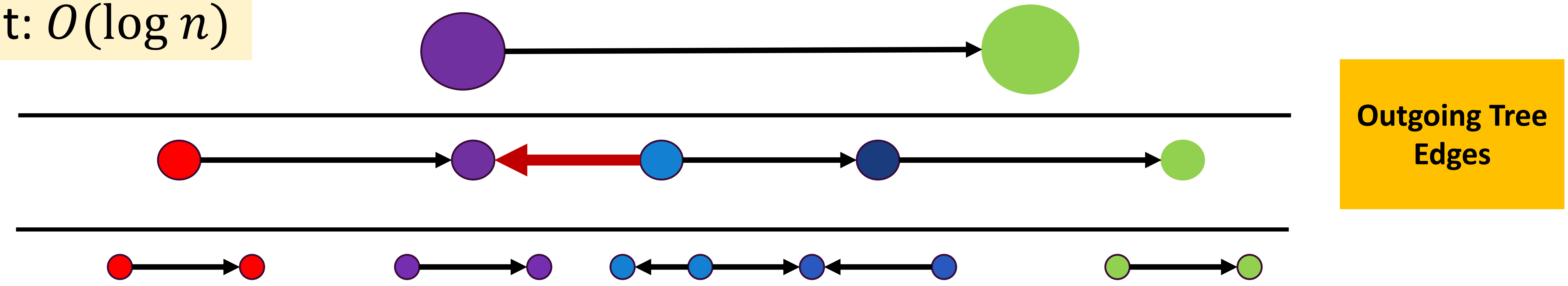


Outgoing Tree Edges

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- On edge deletion: if deletion of an **outgoing tree edge**:
  - Delete from every level
  - **Search for replacement edge from XOR data structure**
- Update **XOR data structure**

Height: $O(\log n)$



Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- Searching efficiently for **new outgoing edges**:
  - **XOR data structure**
    - Each vertex has an ID

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- Searching efficiently for **new outgoing edges**:
  - **XOR data structure**
    - Each vertex has an ID
    - Each vertex stores XOR of IDs of **sampled edges** adjacent to it

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- Searching efficiently for **new outgoing edges**:
  - **XOR data structure**
    - Each vertex has an ID
    - Each vertex stores XOR of IDs of **sampled edges** adjacent to it
    - How do we store sampled edges?

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- Searching efficiently for **new outgoing edges**:
  - **XOR data structure**
    - Each vertex has an ID
    - Each vertex stores XOR of IDs of **sampled edges** adjacent to it
    - How do we store sampled edges?
      - In an array where **sample probability depends on index of array**

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- Searching efficiently for **new outgoing edges**:
  - **XOR data structure**
    - Each vertex has an ID
    - Each vertex stores XOR of IDs of **sampled edges** adjacent to it
    - How do we store sampled edges?
      - In an array where **sample probability depends on index of array**
      - Each node stores such an array

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- **XOR data structure**
  - Each vertex has an ID; each vertex stores XOR of IDs of **sampled edges** adjacent to it

Vertex v array

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- **XOR data structure**
  - Each vertex has an ID; each vertex stores XOR of IDs of **sampled edges** adjacent to it
  - Store in an array where **sample probability depends on index of array**
    - Each node stores such an array

Vertex v array

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- **XOR data structure**
  - Each vertex has an ID; each vertex stores XOR of IDs of **sampled edges** adjacent to it
  - Store in an array where **sample probability depends on index of array**
    - Each node stores such an array

Vertex v array

Store edge (ID_v, ID_u) in index i with probability $\frac{1}{2^i}$

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- **XOR data structure**
  - Each vertex has an ID; each vertex stores XOR of IDs of **sampled edges** adjacent to it
  - Store in an array where **sample probability depends on index of array**
    - Each node stores such an array

Vertex v array

				Index i				

By store we mean XOR the edge with whatever is stored there

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- **XOR data structure**
  - Duplicate each index $O(\log n)$ times; with high probability, at least one index stores **exactly one edge**

Vertex v array

By store we mean XOR the edge with whatever is stored there

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

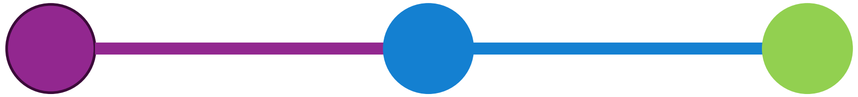
- **XOR data structure**
  - Duplicate each index $O(\log n)$ times; with high probability, at least one index stores **exactly one edge**
  - Use this data structure to find an edge across a cut quickly

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- **XOR data structure**
  - Duplicate each index $O(\log n)$ times; with high probability, at least one index stores **exactly one edge**
  - Use this data structure to find an edge across a cut quickly
- Compute XOR of values stored in **every index of every node in Euler Tour Tree with tree edges**—every tree edge stored in every index

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- **XOR data structure**
  - Duplicate each index $O(\log n)$ times; with high probability, at least one index stores **exactly one edge**
  - Use this data structure to find an edge across a cut quickly
- Compute XOR of values stored in **every index of every node in Euler Tour Tree with tree edges**—every tree edge stored in every index



If XOR data structure of tree only contains tree edges, returns 0

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- **XOR data structure**
  - Duplicate each index $O(\log n)$ times; with high probability, at least one index stores **exactly one edge**
  - Use this data structure to find an edge across a cut quickly
- Compute XOR of values stored in **every index of every node in Euler Tour Tree with tree edges**—every tree edge stored in every index

Otherwise, if contains one outgoing edge, returns ID_e

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- **Why have different probabilities of sampling:**
  - Due to XOR data structure, return **exactly one edge** in cut whp

Otherwise, if contains one outgoing edge, returns ID_e

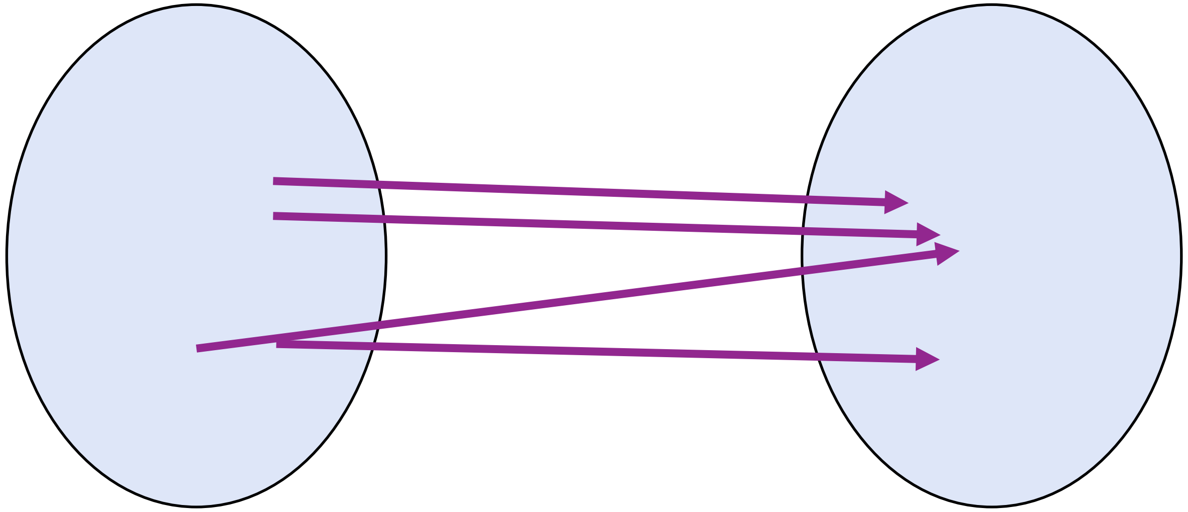
Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- **Why have different probabilities of sampling:**
  - Due to XOR data structure, return **exactly one edge** in cut whp
  - **Cutset data structure**

Otherwise, if contains one outgoing edge, returns ID_e

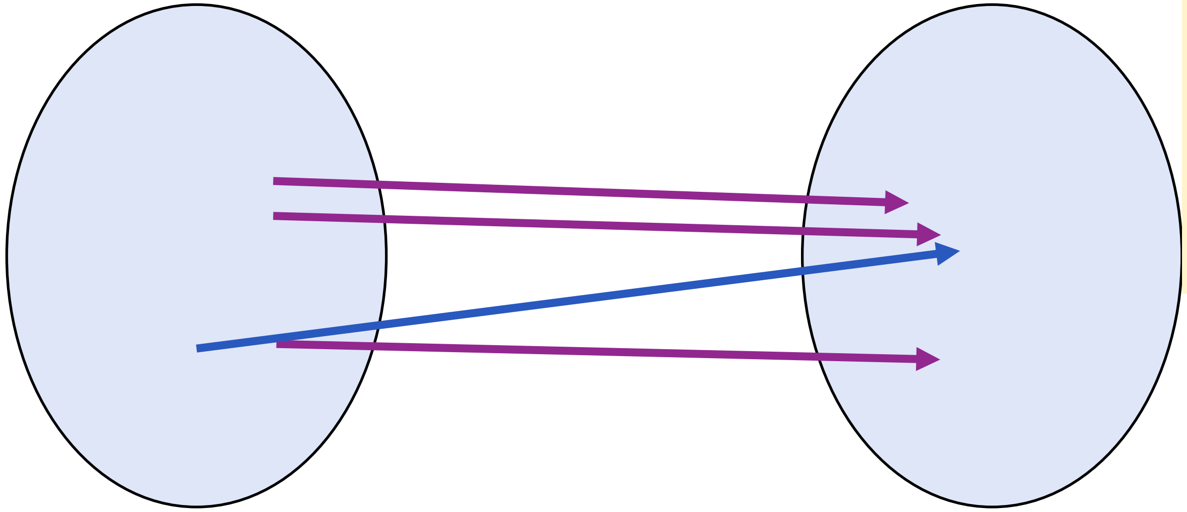
Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- **Why have different probabilities of sampling:**
  - Due to XOR data structure, return **exactly one edge** in cut whp
  - **Cutset data structure**



Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- **Why have different probabilities of sampling:**
  - Due to XOR data structure, return **exactly one edge** in cut whp
  - **Cutset data structure**

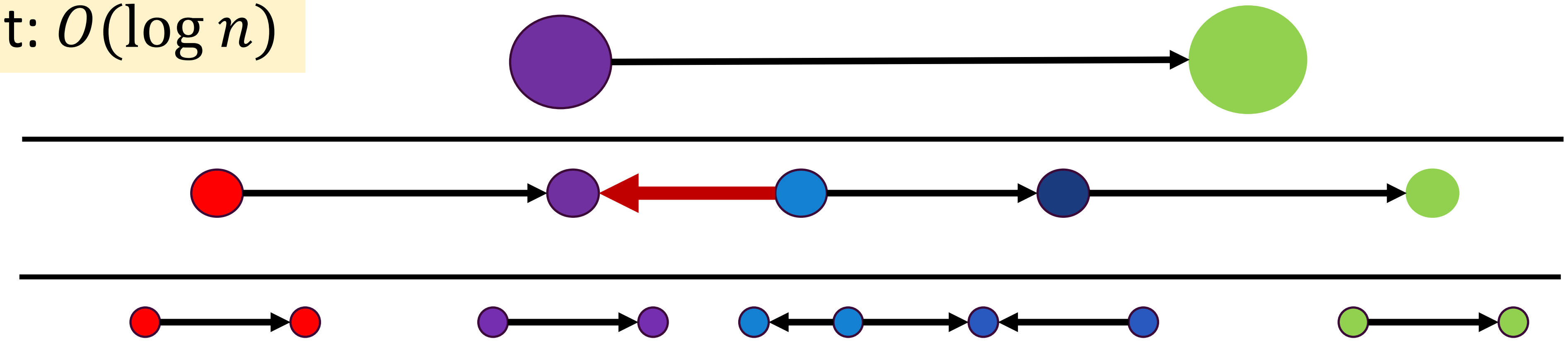


Probability $\frac{1}{4}$ returns 1 edge in cut in expectation

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- On edge deletion: if deletion of an **outgoing tree edge**:
  - Delete from every level
  - **Search for replacement edge from XOR data structure**
- Update **XOR data structure**

Height: $O(\log n)$

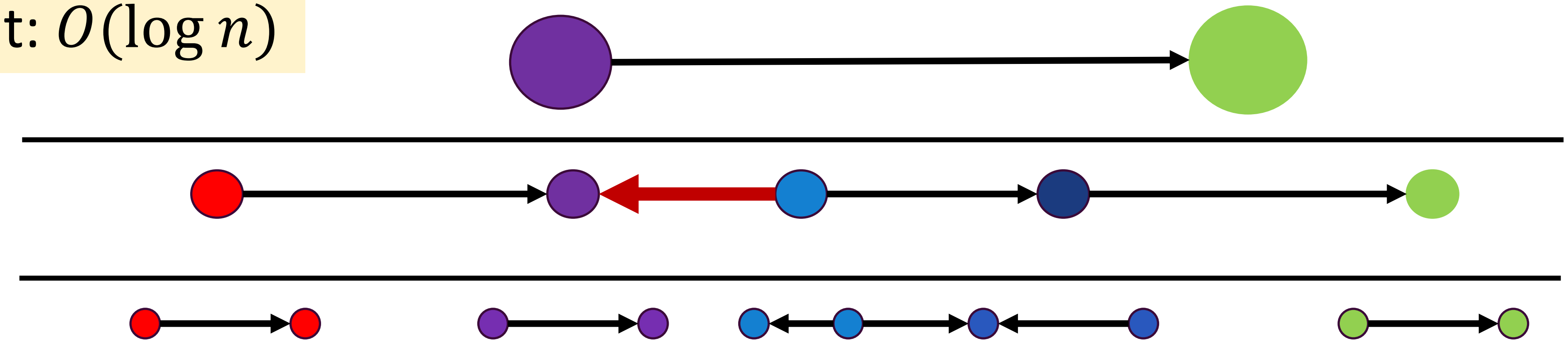


Outgoing Tree Edges

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- Additional details: newly inserted replacement edge can **cause cycles in higher levels**

Height: $O(\log n)$

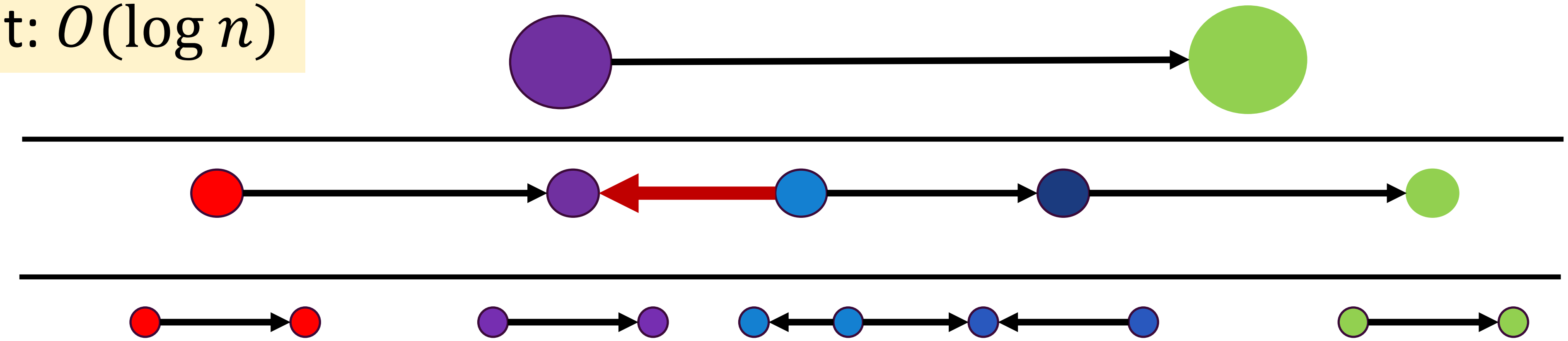


Outgoing Tree Edges

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- Additional details: newly inserted replacement edge can **cause cycles in higher levels**
  - **Break cycles using binary search** in Euler Tour tree

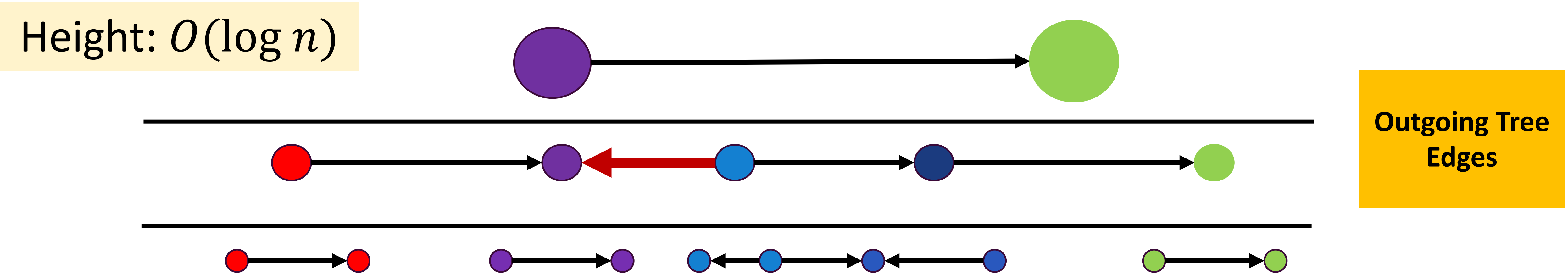
Height: $O(\log n)$



Outgoing Tree Edges

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- Additional details: newly inserted replacement edge can **cause cycles in higher levels**
  - **Break cycles using binary search** in Euler Tour tree
  - **Total:** $\mathrm{poly}(\log \boldsymbol{n})$ **time per operation**



Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- Correctness with High Probability:
  - Each level's randomness is **independent of previous level**

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn ‘15]

- Correctness with High Probability:
  - Each level’s randomness is **independent of previous level**
  - $O(\log n)$ levels success with high probability:

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- Correctness with High Probability:
  - Each level's randomness is **independent of previous level**
  - $O(\log n)$ levels success with high probability:
    - Count # of connected components decrease using cutset

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- Correctness with High Probability:
  - Each level's randomness is **independent of previous level**
  - $O(\log n)$ levels success with high probability:
    - Count # of connected components decrease using cutset
    - Each level expected decrease number of connected components by $\frac{1}{8}$

Monte Carlo Oblivious Adversary Dynamic Connectivity [Gibbs-Kapron-King-Thorn '15]

- Correctness with High Probability:
  - Each level's randomness is **independent of previous level**
  - $O(\log n)$ levels success with high probability:
    - Count # of connected components decrease using cutset
    - Each level expected decrease number of connected components by $\frac{1}{8}$
    - Thus, by Chernoff, $O(\log n)$ levels suffice