Space-Efficient Random Walks on Streaming Graphs

Serafeim Papadias, Zoi Kaoudi, Jorge-Arnulfo Quiane-Ruiz*, Volker Markl

VLDB 2023

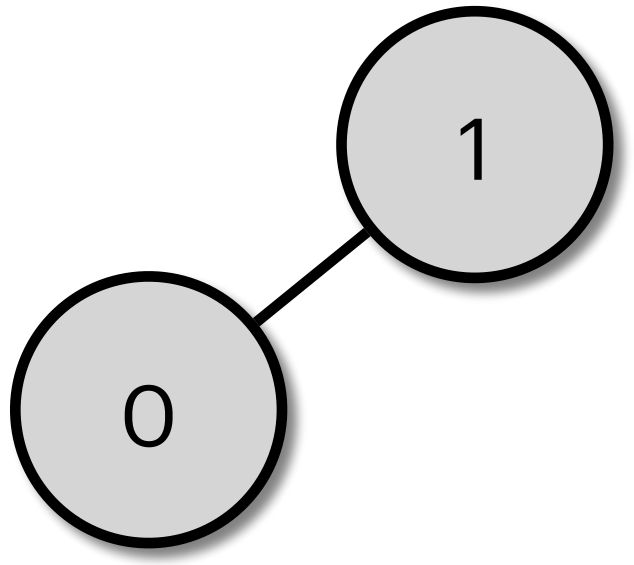
* Jorge will be remembered as bright mentor and his words will always nest both in my mind and in my heart — "Somos chingones"

Background: Streaming Graphs

A streaming graph is a sequence of discrete graph snapshots, $G_t = \{V_t, E_t\}$*, where* $V_t = \{v_1^t, \ldots, v_n^t\}$ *are the vertices,* $E_t = \{e_1^t, \ldots, e_m^t\}$ *are the edges, and* $t \in N$ *is a timestamp*

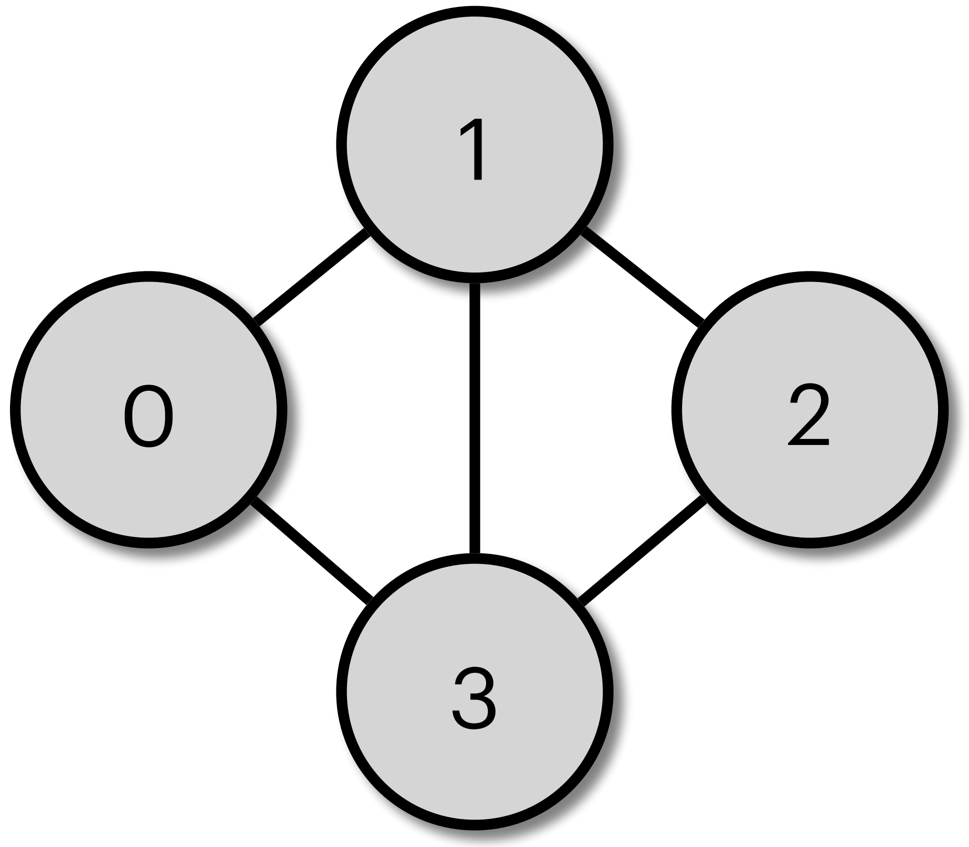
Background: Streaming Graphs

A streaming graph is a sequence of discrete graph snapshots, $G_t = \{V_t, E_t\}$*, where* $V_t = \{v_1^t, \ldots, v_n^t\}$ *are the vertices,* $E_t = \{e_1^t, \ldots, e_m^t\}$ *are the edges, and* $t \in N$ *is a timestamp*



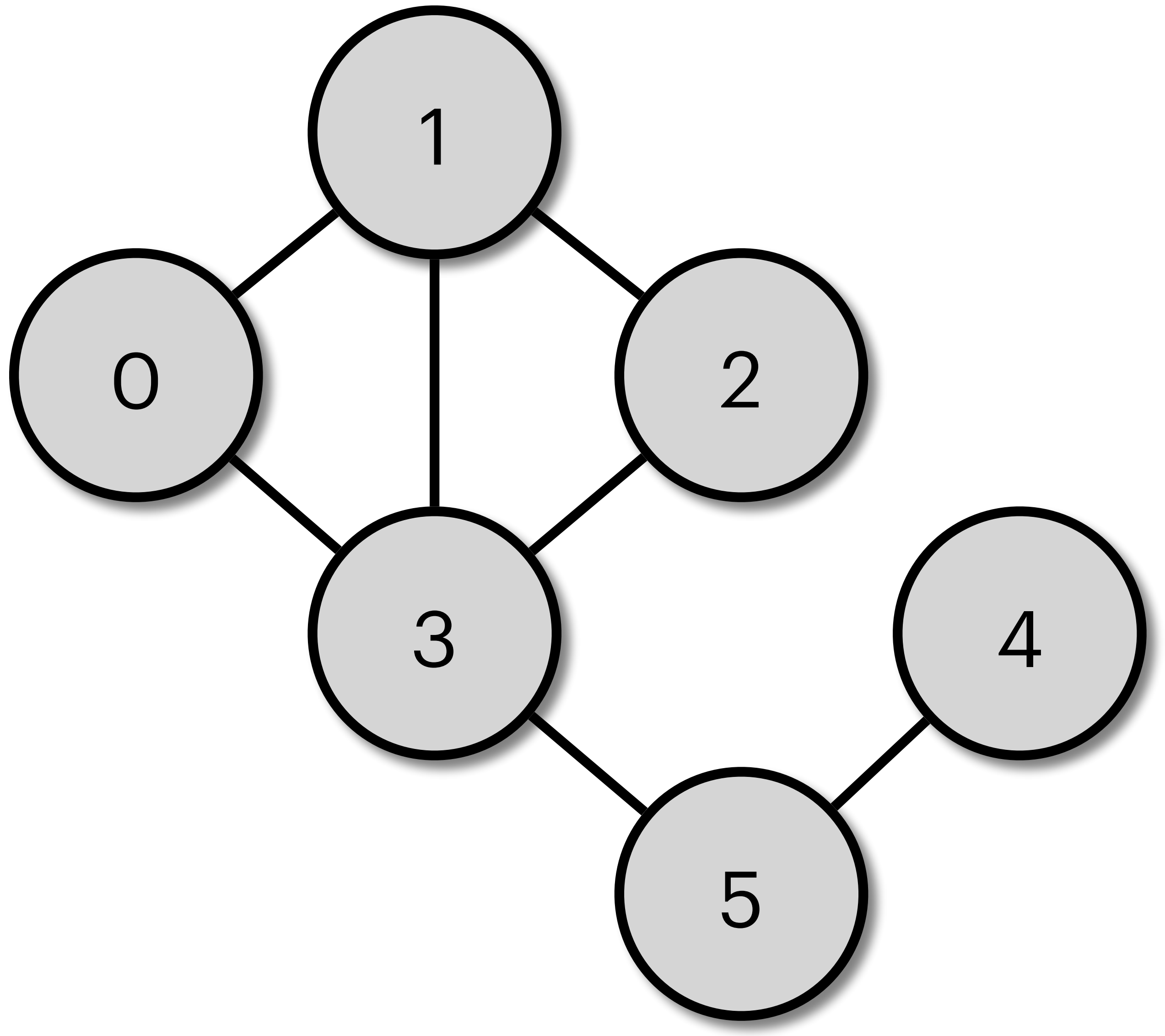
Background: Streaming Graphs

A streaming graph is a sequence of discrete graph snapshots, $G_t = \{V_t, E_t\}$*, where* $V_t = \{v_1^t, \ldots, v_n^t\}$ *are the vertices,* $E_t = \{e_1^t, \ldots, e_m^t\}$ *are the edges, and* $t \in N$ *is a timestamp*



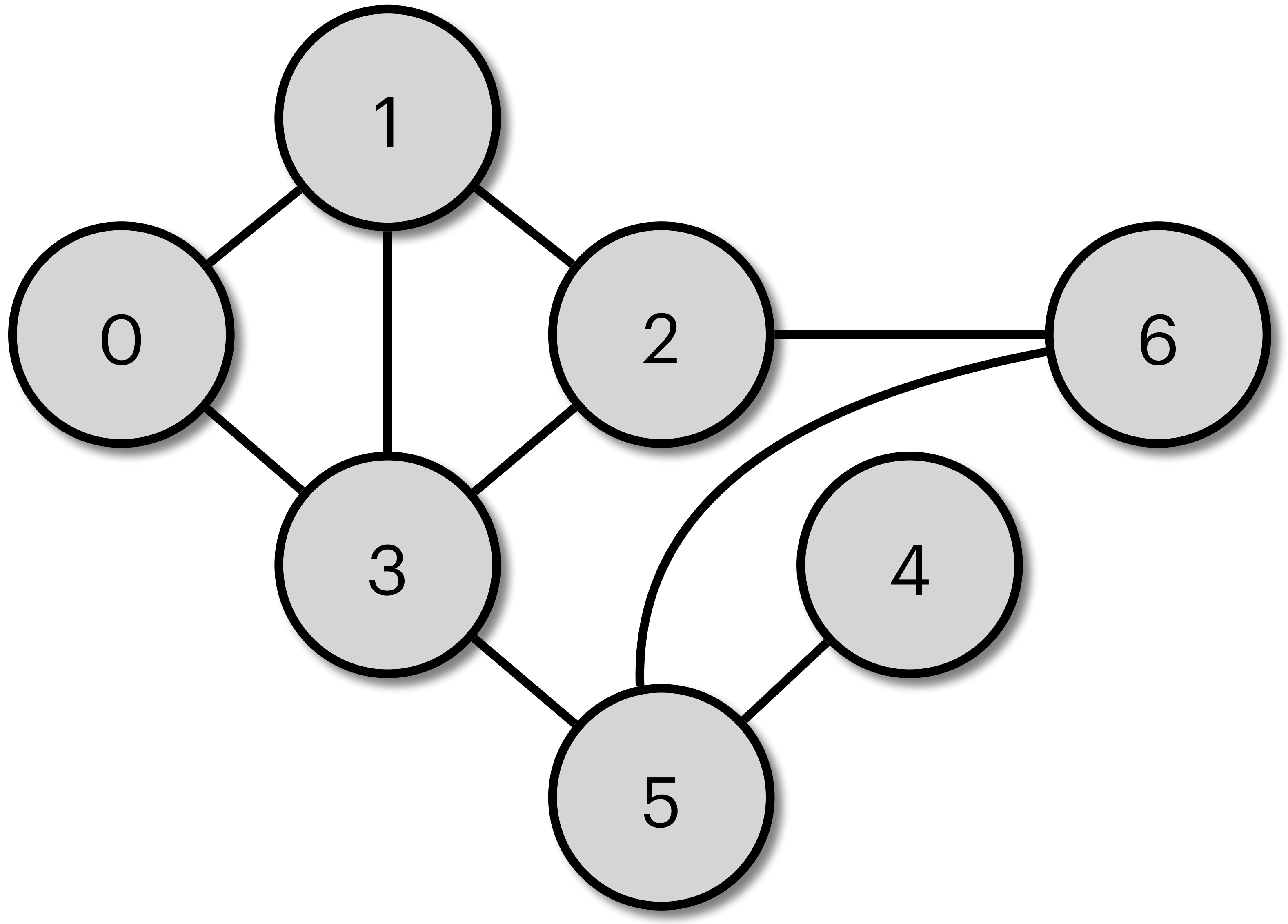
Background: Streaming Graphs

A streaming graph is a sequence of discrete graph snapshots, $G_t = \{V_t, E_t\}$, *where* $V_t = \{v_1^t, \ldots, v_n^t\}$ *are the vertices,* $E_t = \{e_1^t, \ldots, e_m^t\}$ *are the edges, and* $t \in N$ *is a timestamp*



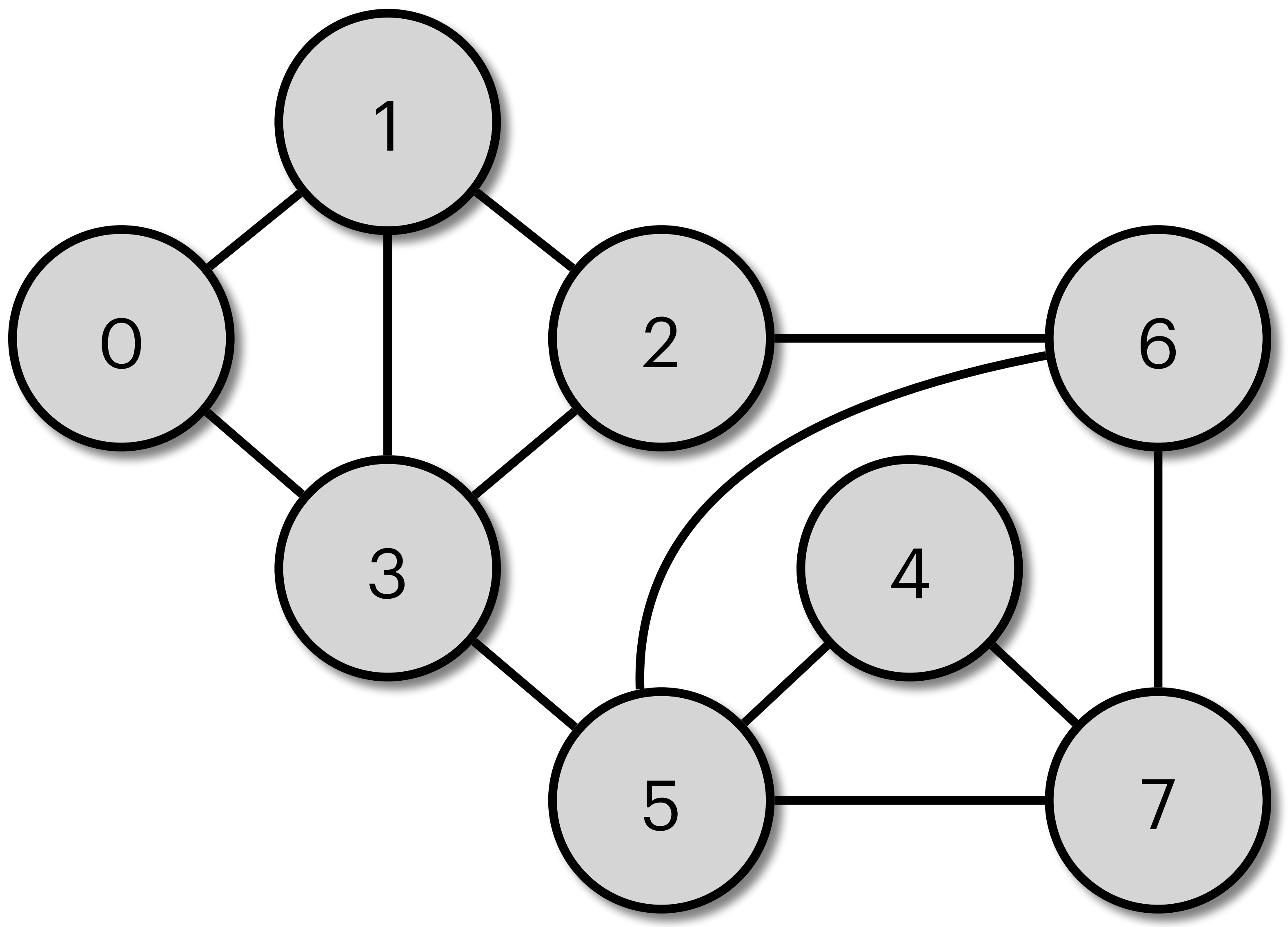
Background: Streaming Graphs

A streaming graph is a sequence of discrete graph snapshots, $G_t = \{V_t, E_t\}$*, where* $V_t = \{v_1^t, \ldots, v_n^t\}$ *are the vertices,* $E_t = \{e_1^t, \ldots, e_m^t\}$ *are the edges, and* $t \in N$ *is a timestamp*



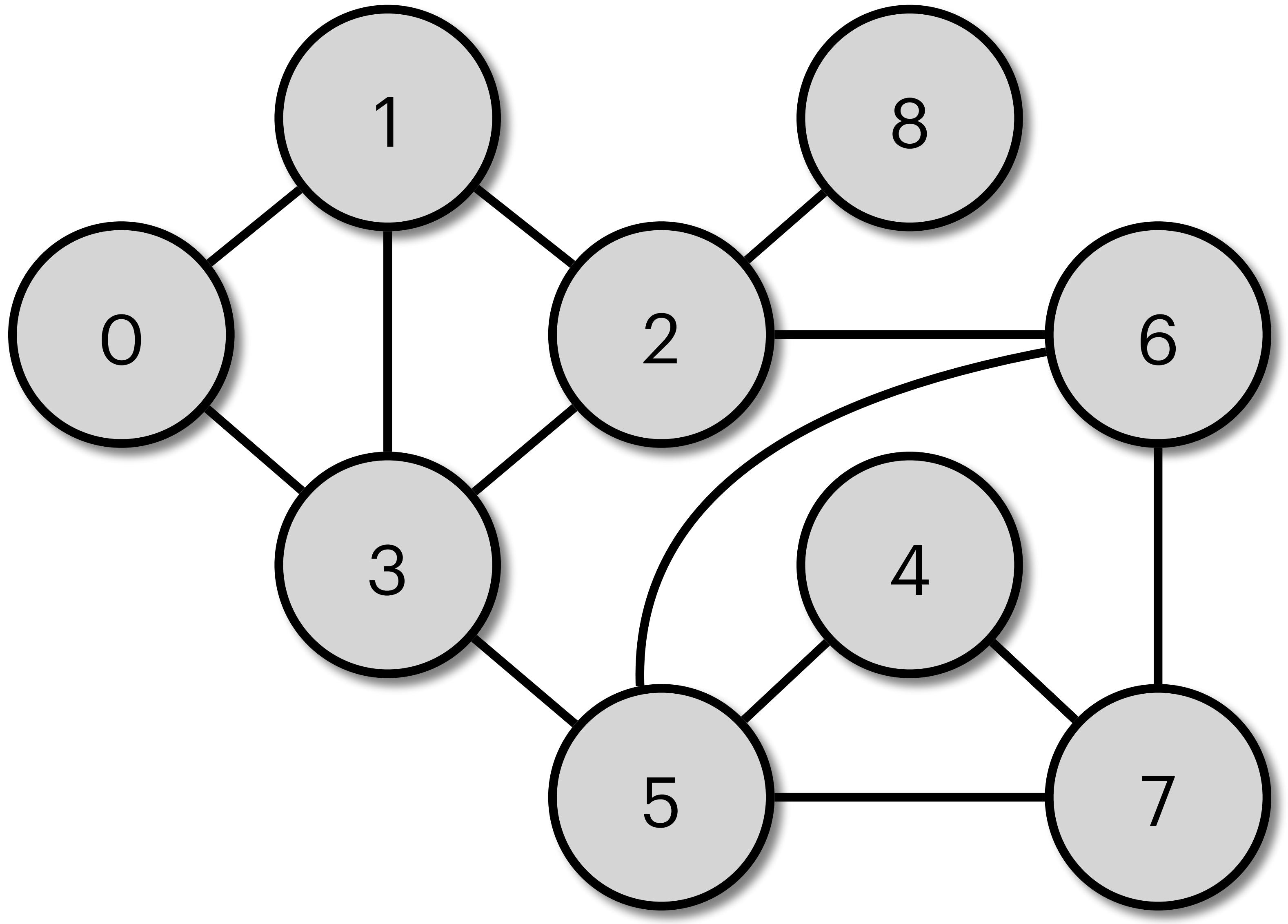
Background: Streaming Graphs

A streaming graph is a sequence of discrete graph snapshots, $G_t = \{V_t, E_t\}$*, where* $V_t = \{v_1^t, \ldots, v_n^t\}$ *are the vertices,* $E_t = \{e_1^t, \ldots, e_m^t\}$ *are the edges, and* $t \in N$ *is a timestamp*



Background: Streaming Graphs

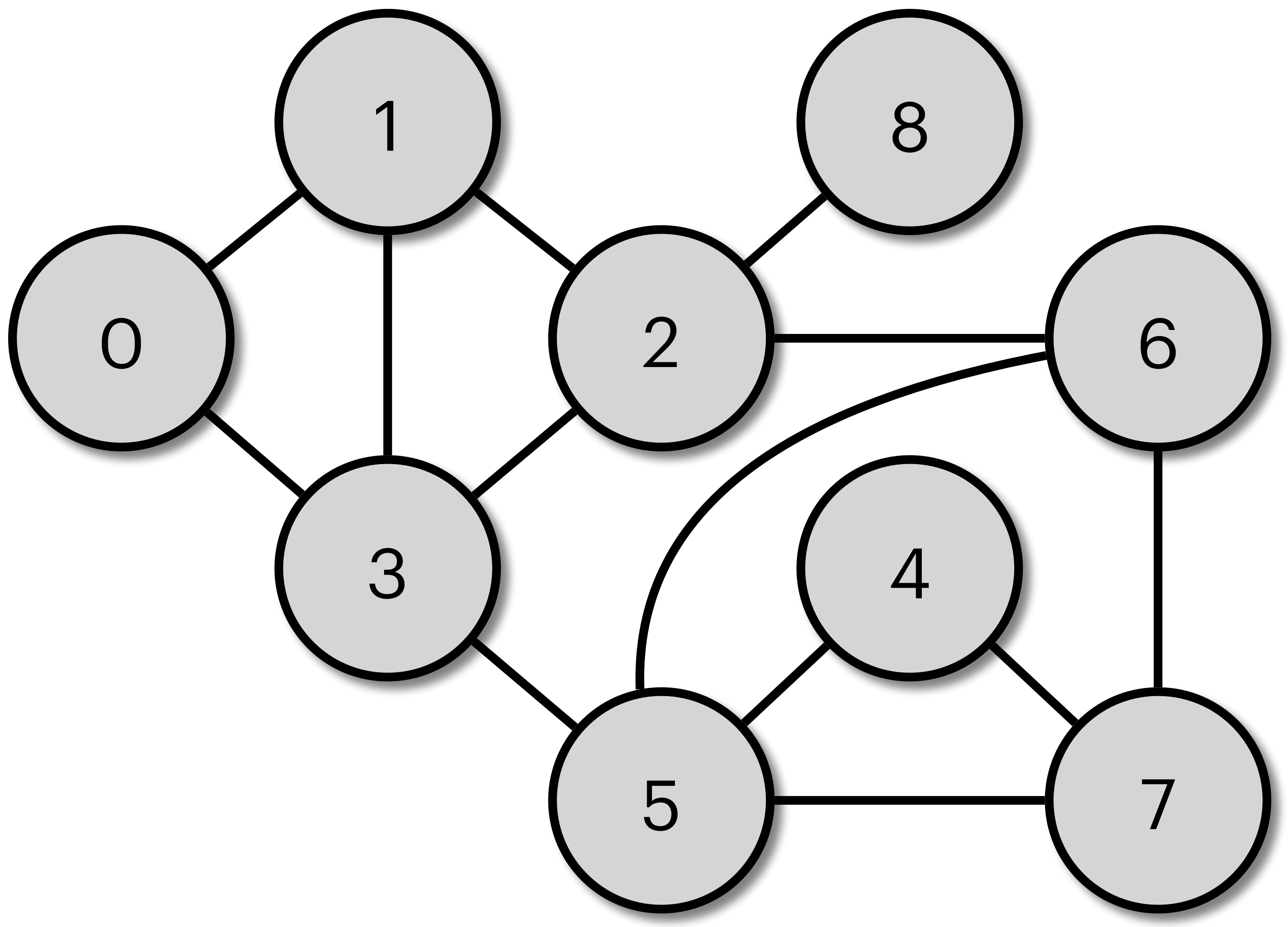
A streaming graph is a sequence of discrete graph snapshots, $G_t = \{V_t, E_t\}$*, where* $V_t = \{v_1^t, \ldots, v_n^t\}$ *are the vertices,* $E_t = \{e_1^t, \ldots, e_m^t\}$ *are the edges, and* $t \in N$ *is a timestamp*



Background: Random Walk

A random walk is a sequence of vertices that represent the graph

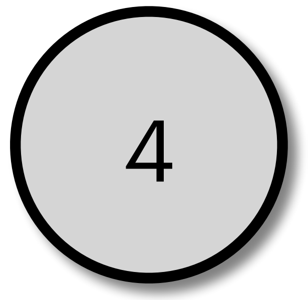
$w:$

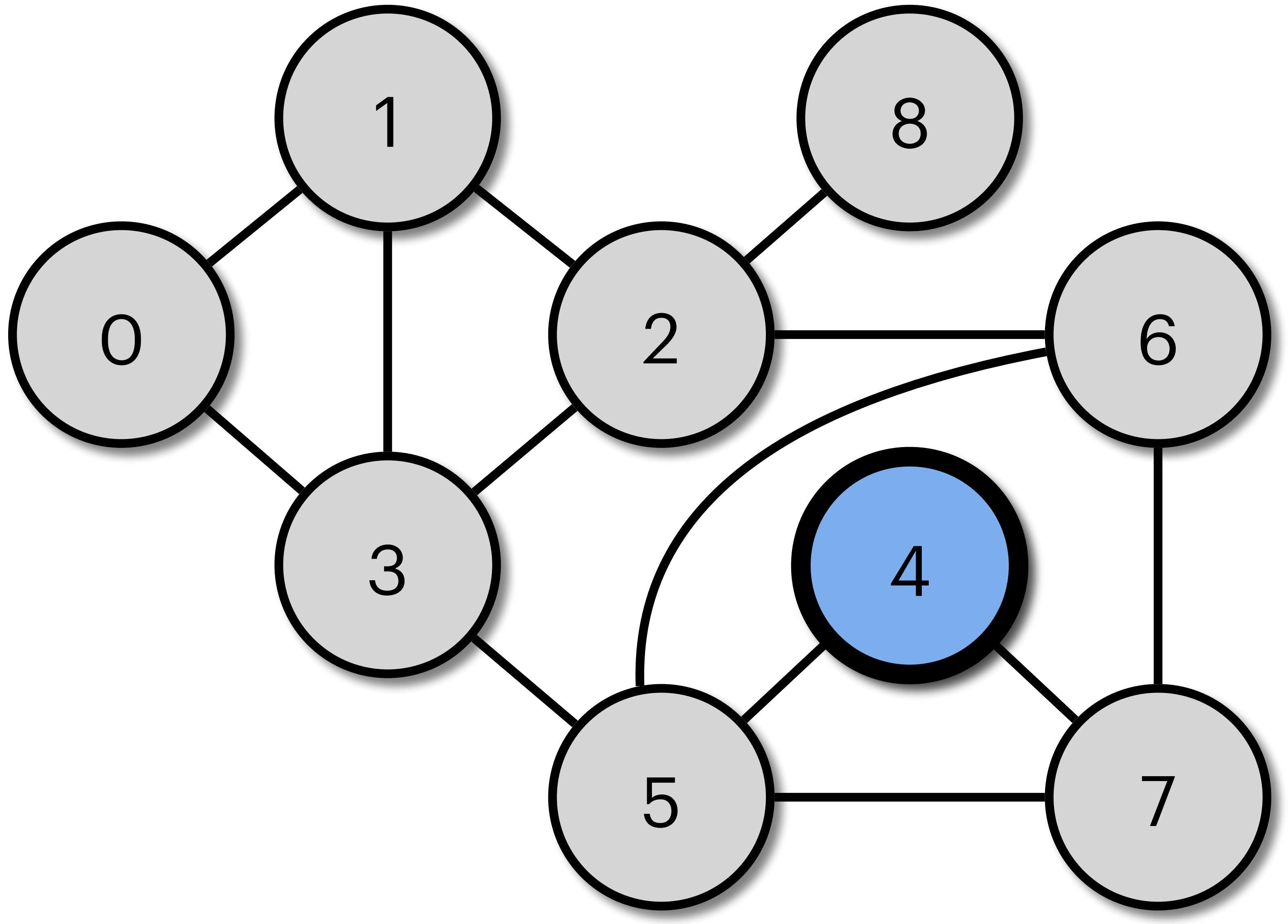


Background: Random Walk

A random walk is a sequence of vertices that represent the graph

w :

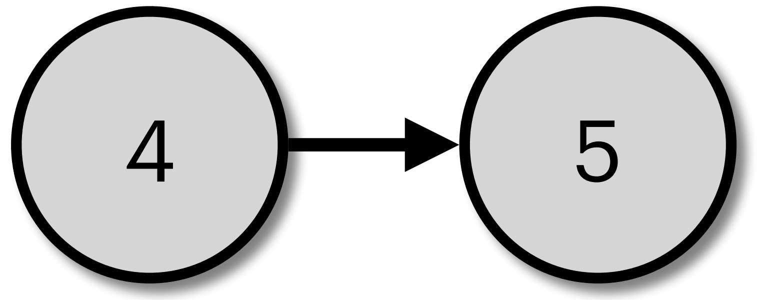


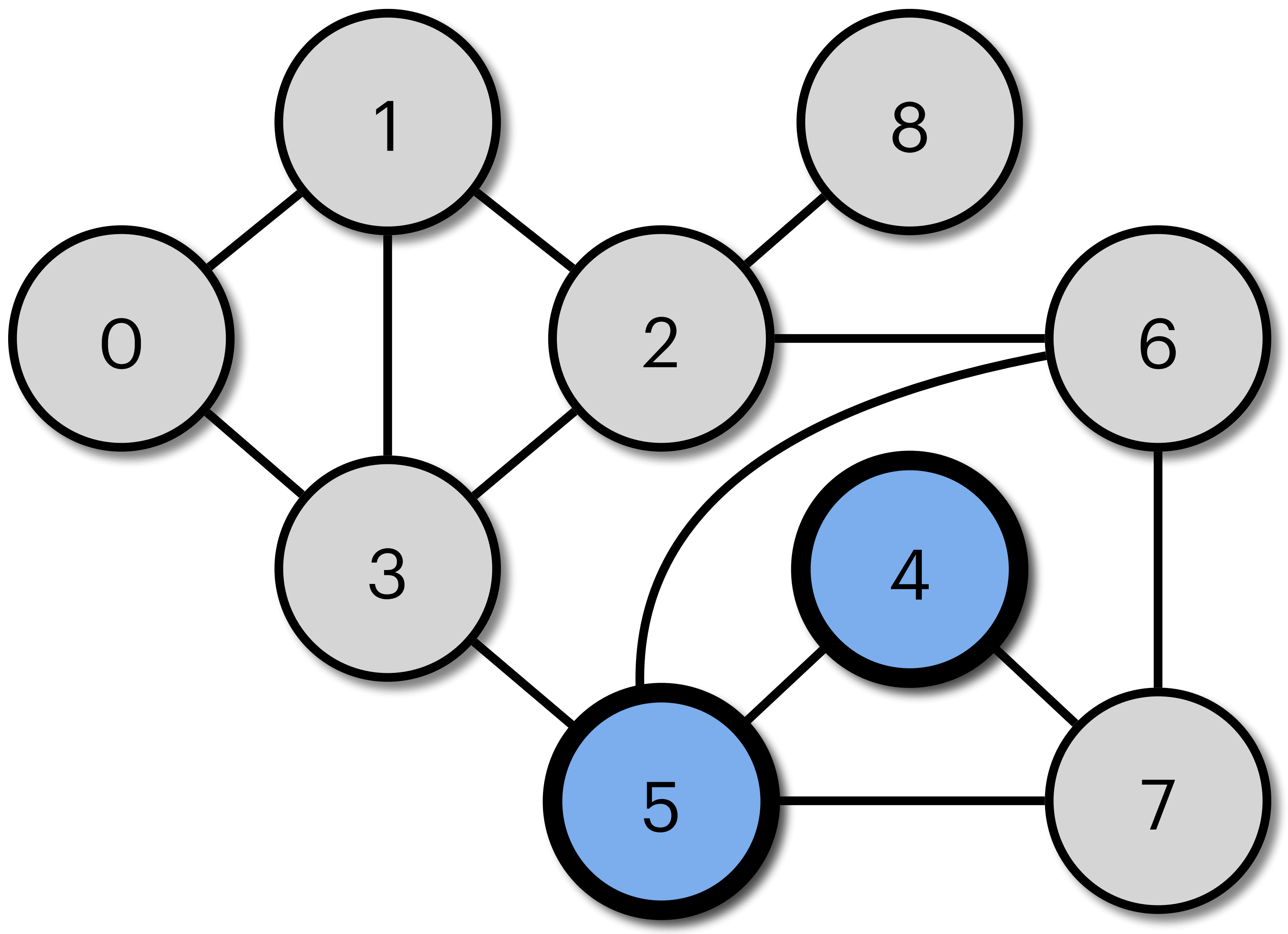


Background: Random Walk

A random walk is a sequence of vertices that represent the graph

w :

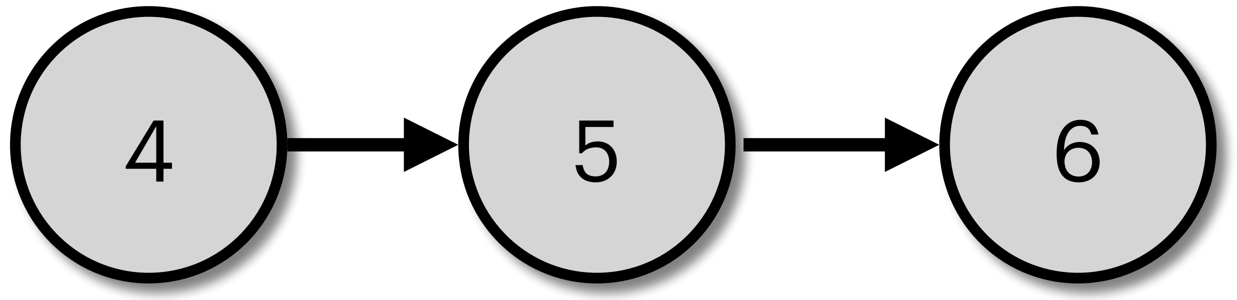


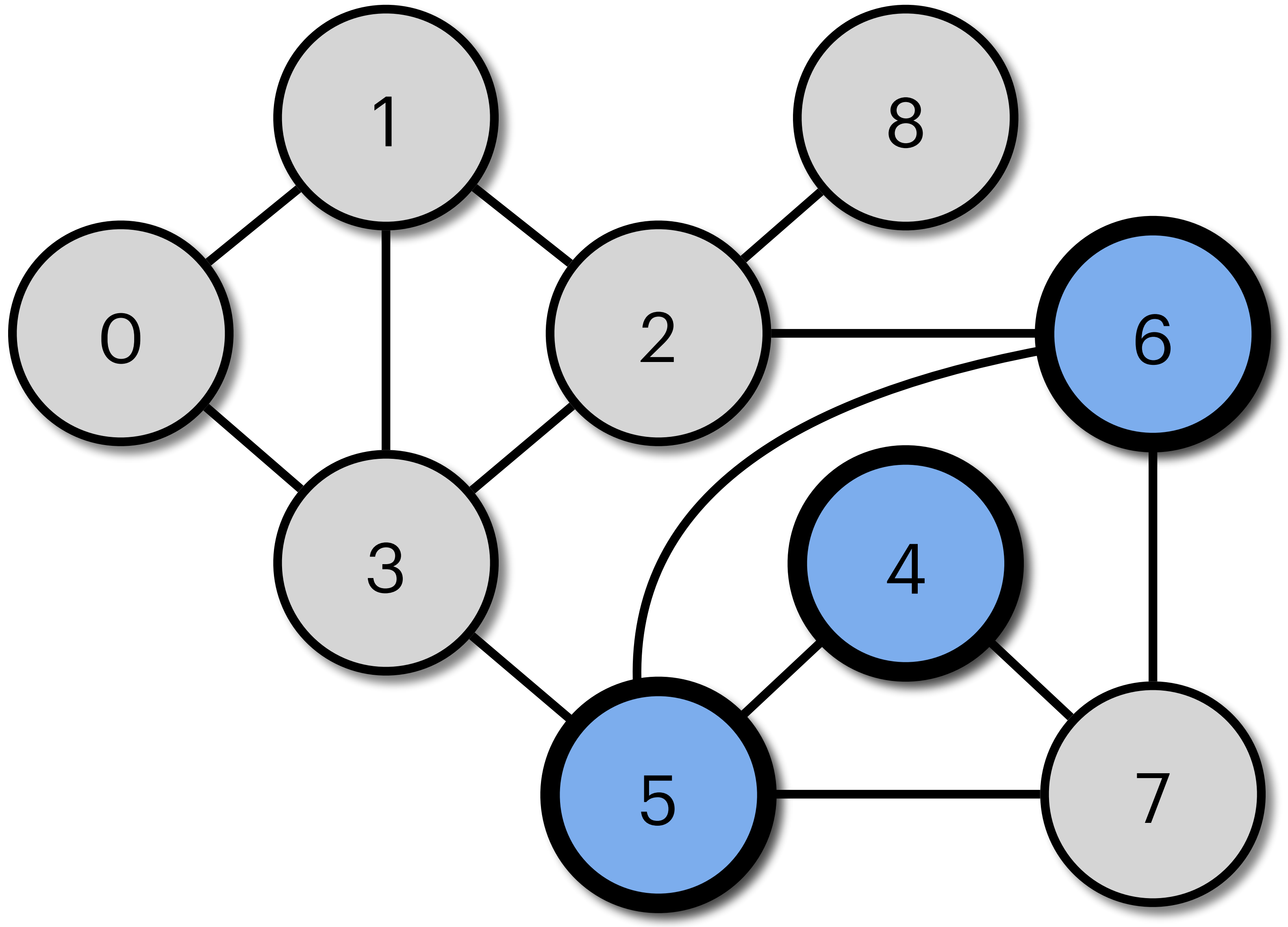


Background: Random Walk

A random walk is a sequence of vertices that represent the graph

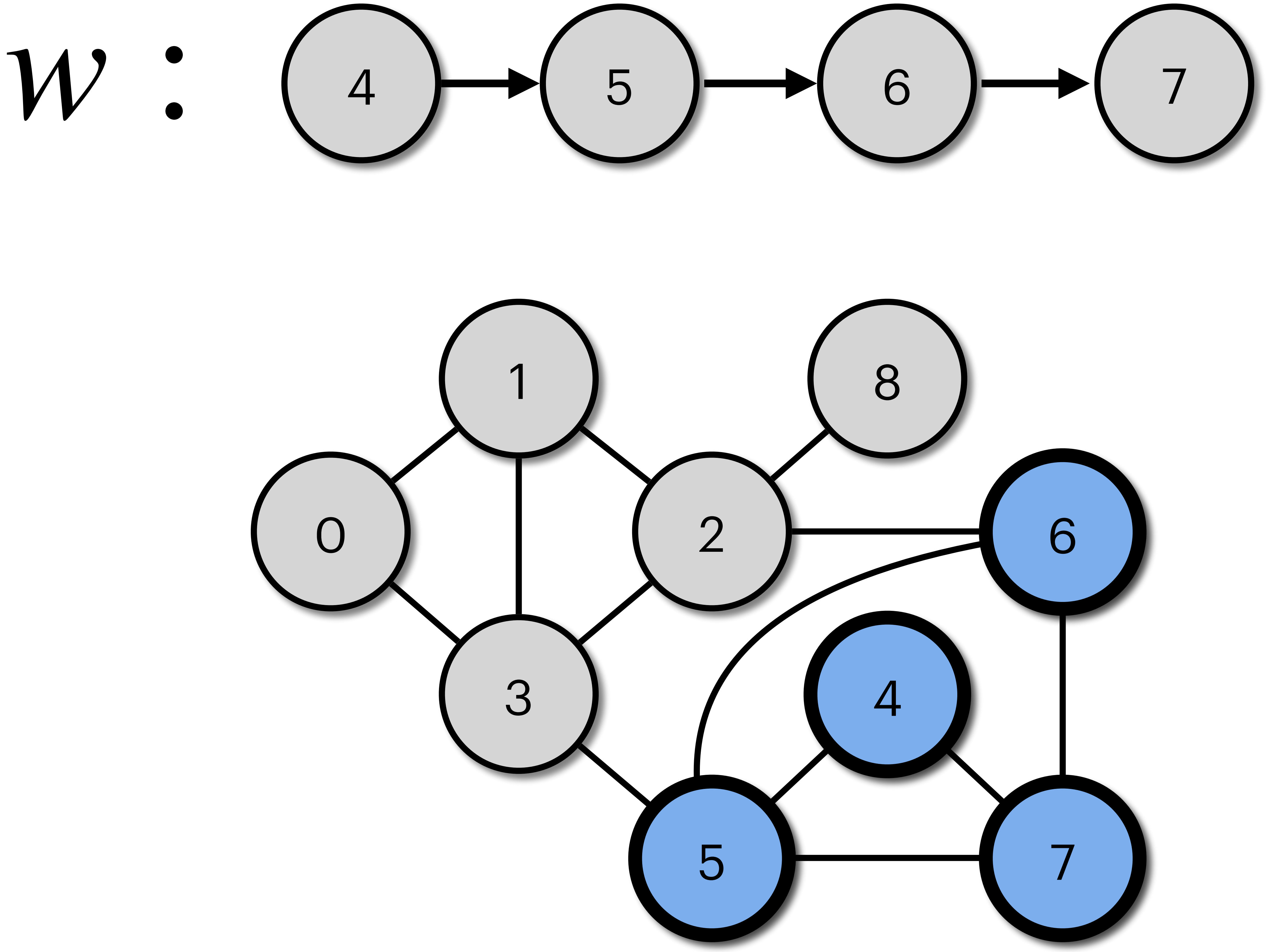
w :





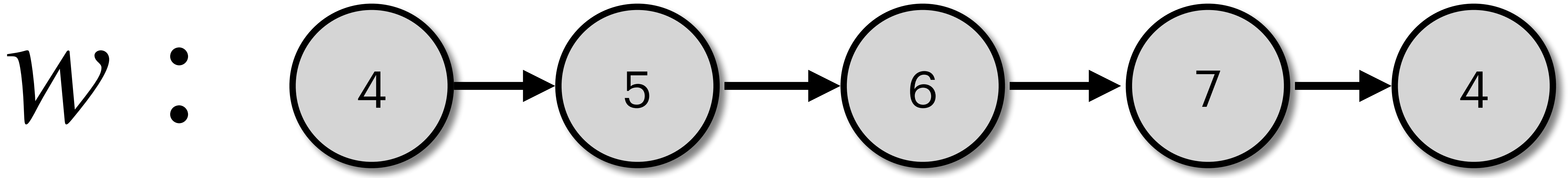
Background: Random Walk

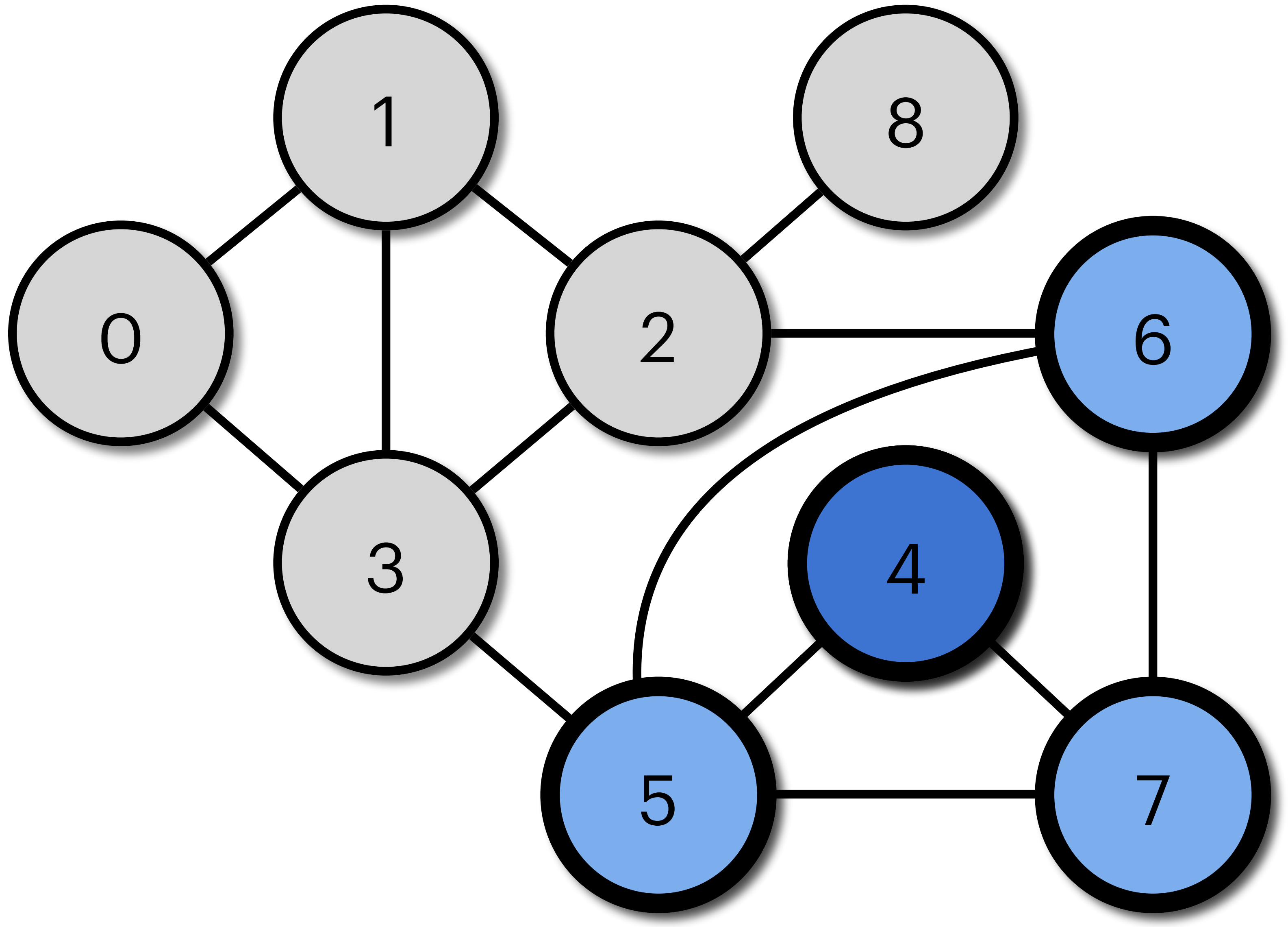
A random walk is a sequence of vertices that represent the graph



Background: Random Walk

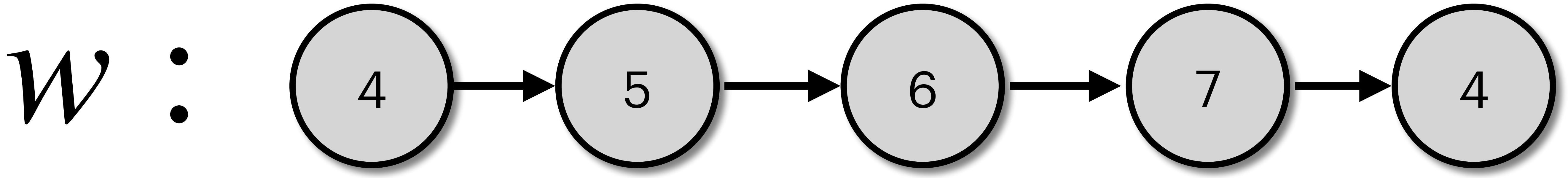
A random walk is a sequence of vertices that represent the graph

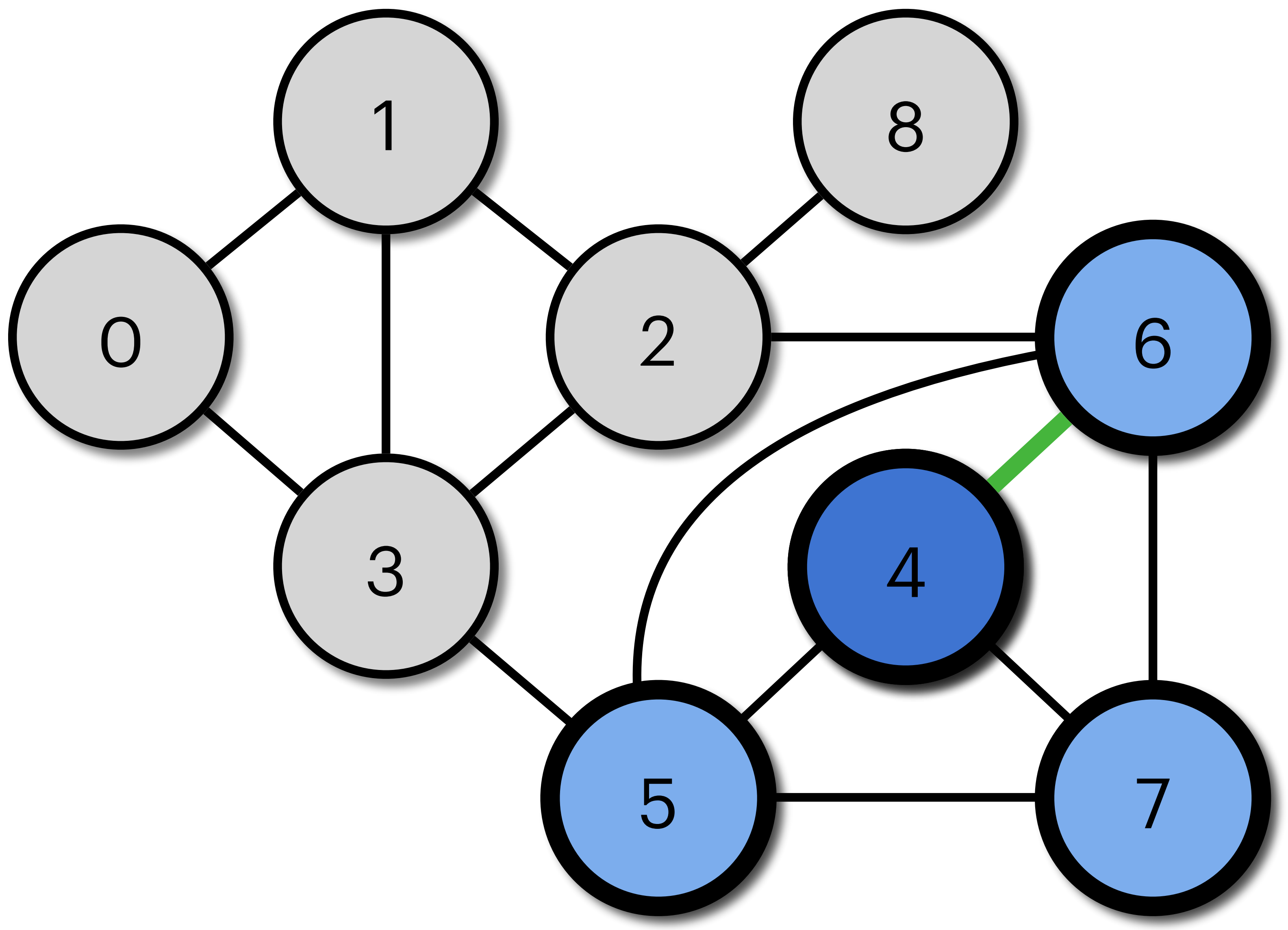




Background: Random Walk

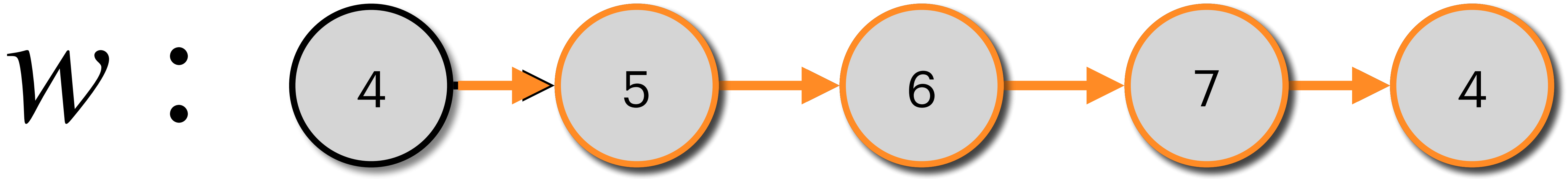
A random walk is a sequence of vertices that represent the graph



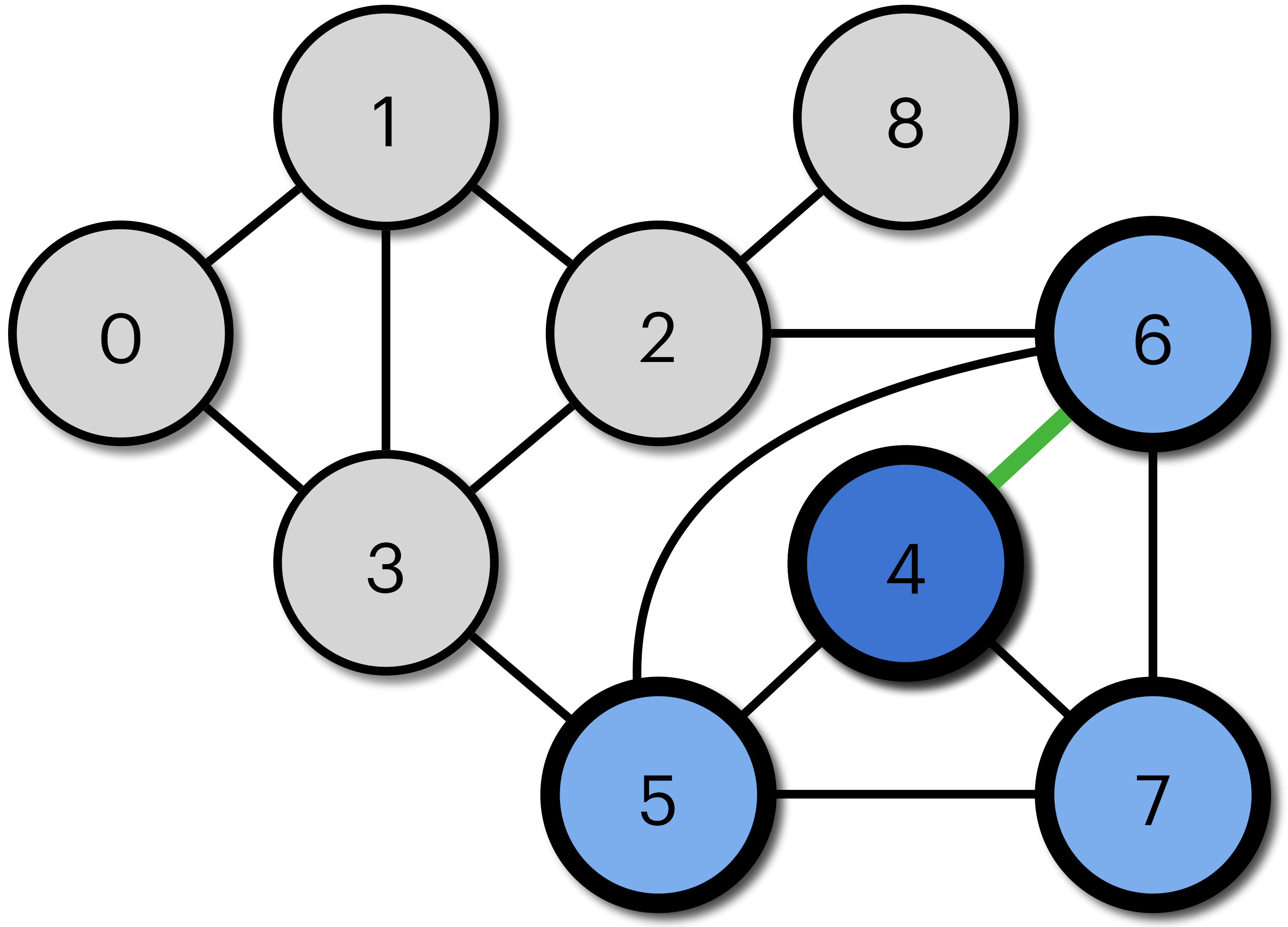


Background: Random Walk

A random walk is a sequence of vertices that represent the graph

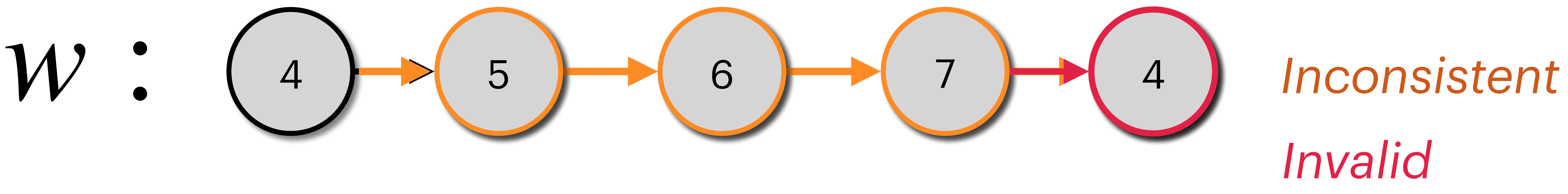


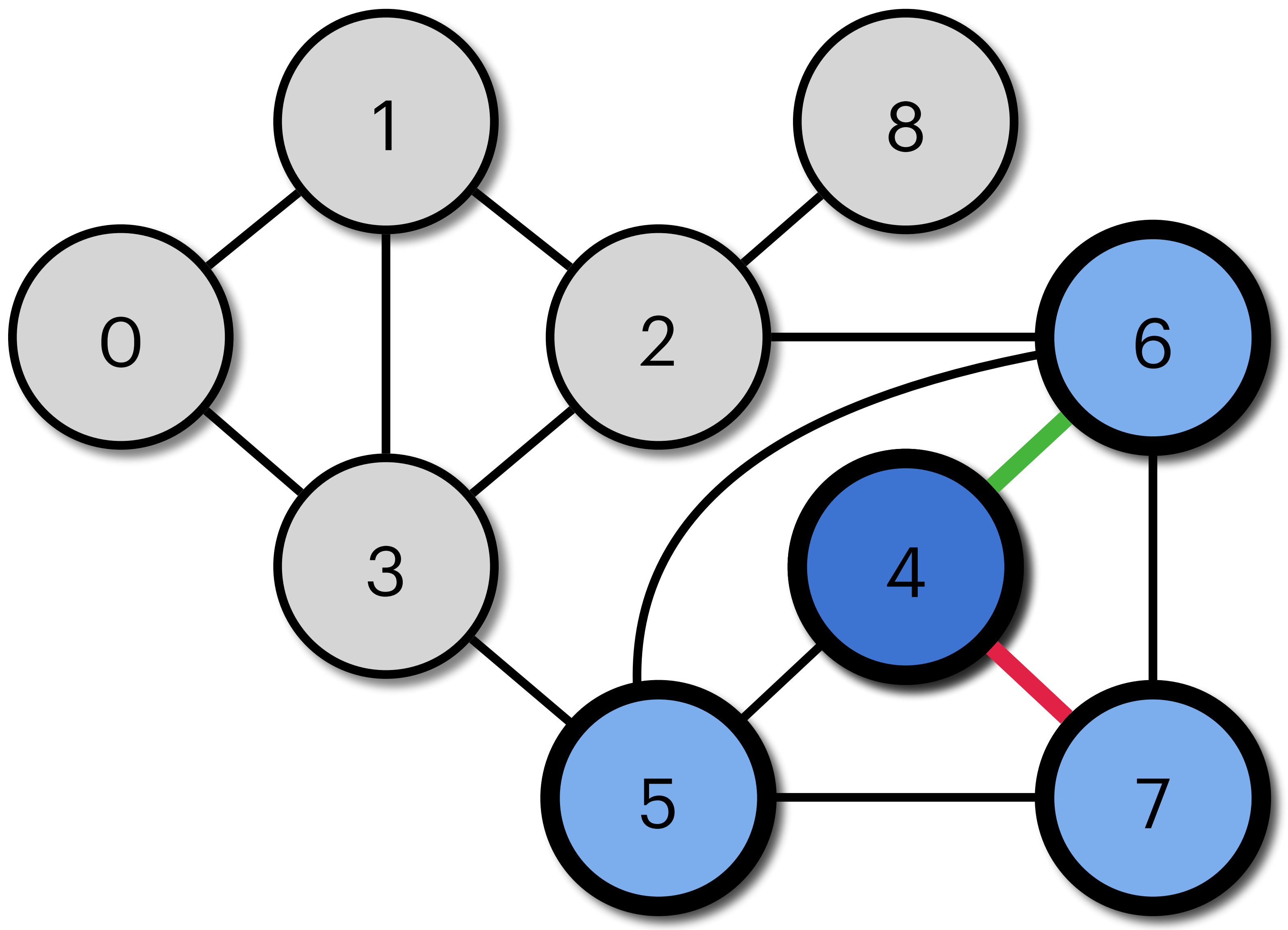
Inconsistent



Background: Random Walk

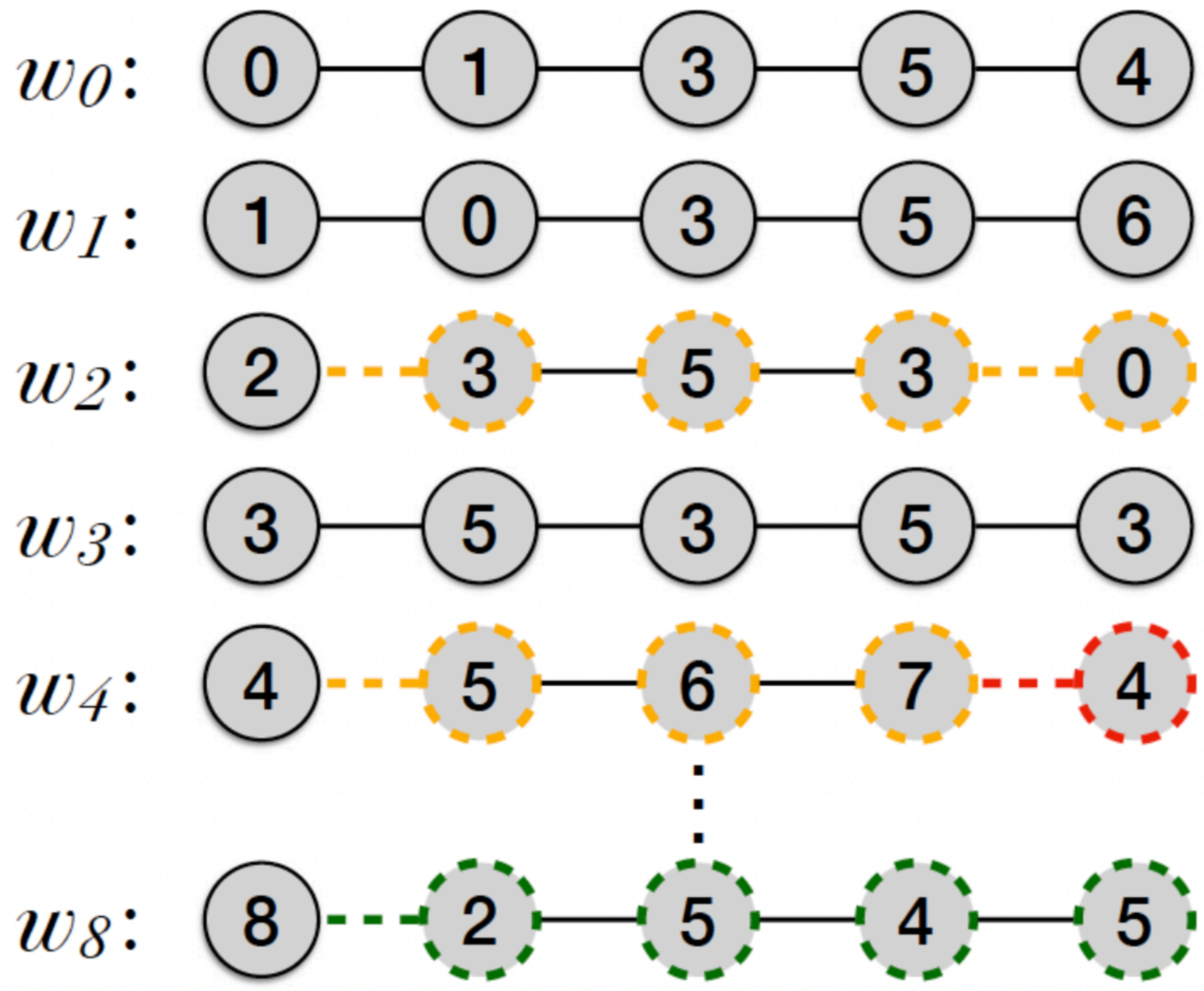
A random walk is a sequence of vertices that represent the graph





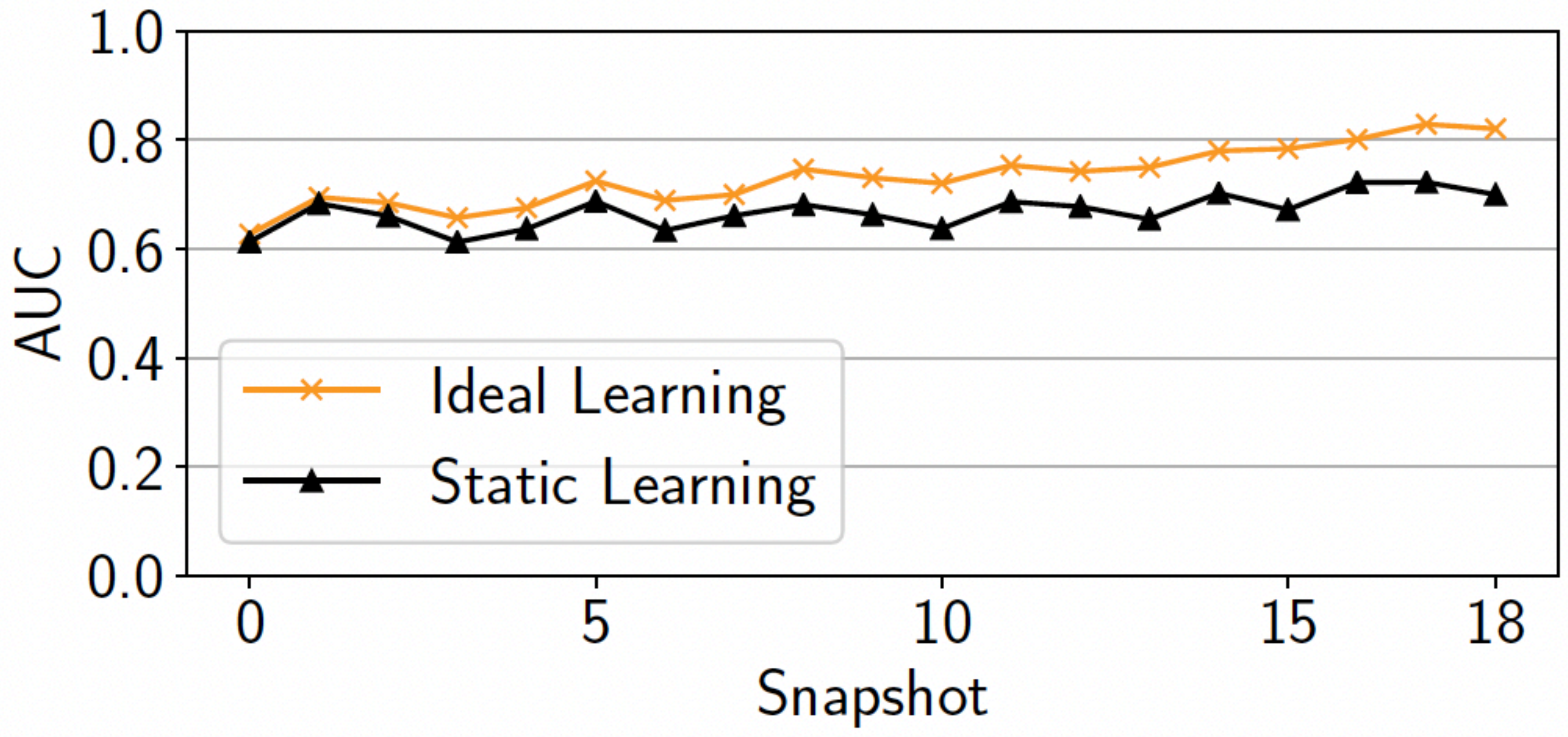
Graph ML Applications do not rely on a single walk!

Applications do not use a small number of random walks but huge corpuses

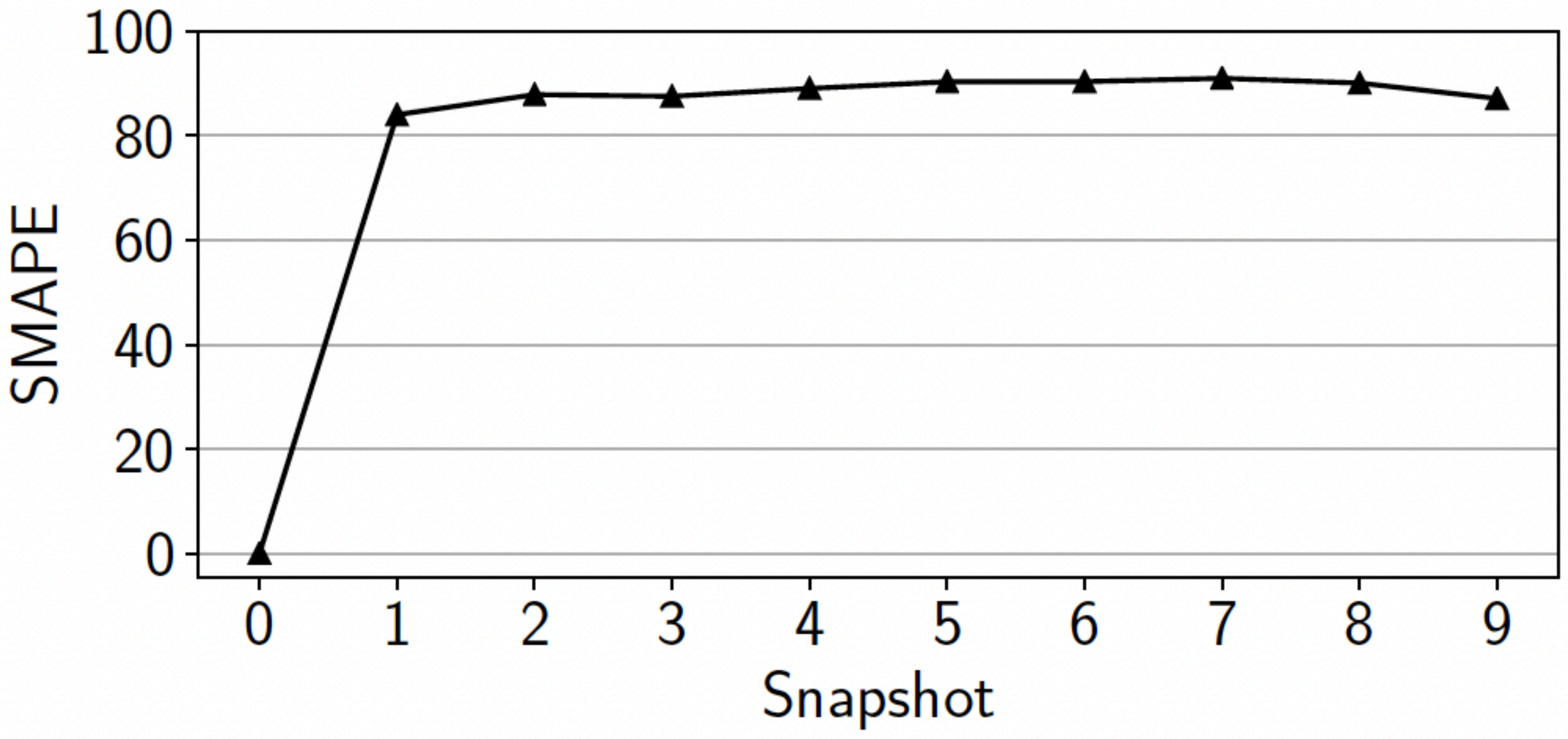


Update to date Walk Corpuses for Accuracy

The Graph ML models are computed on walk corpuses that must be up to date



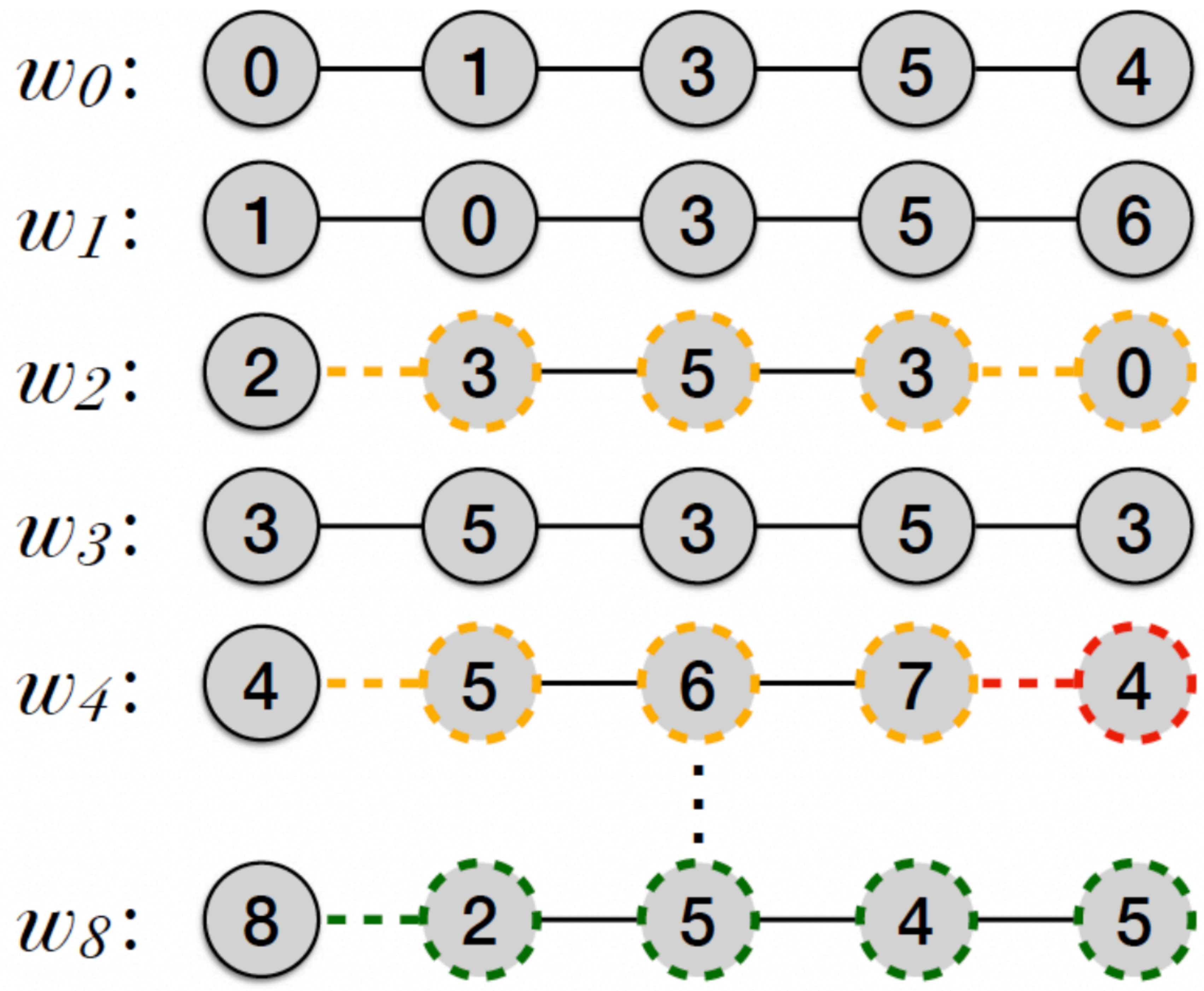
(a) Graph Embeddings



(b) Personalized PageRank

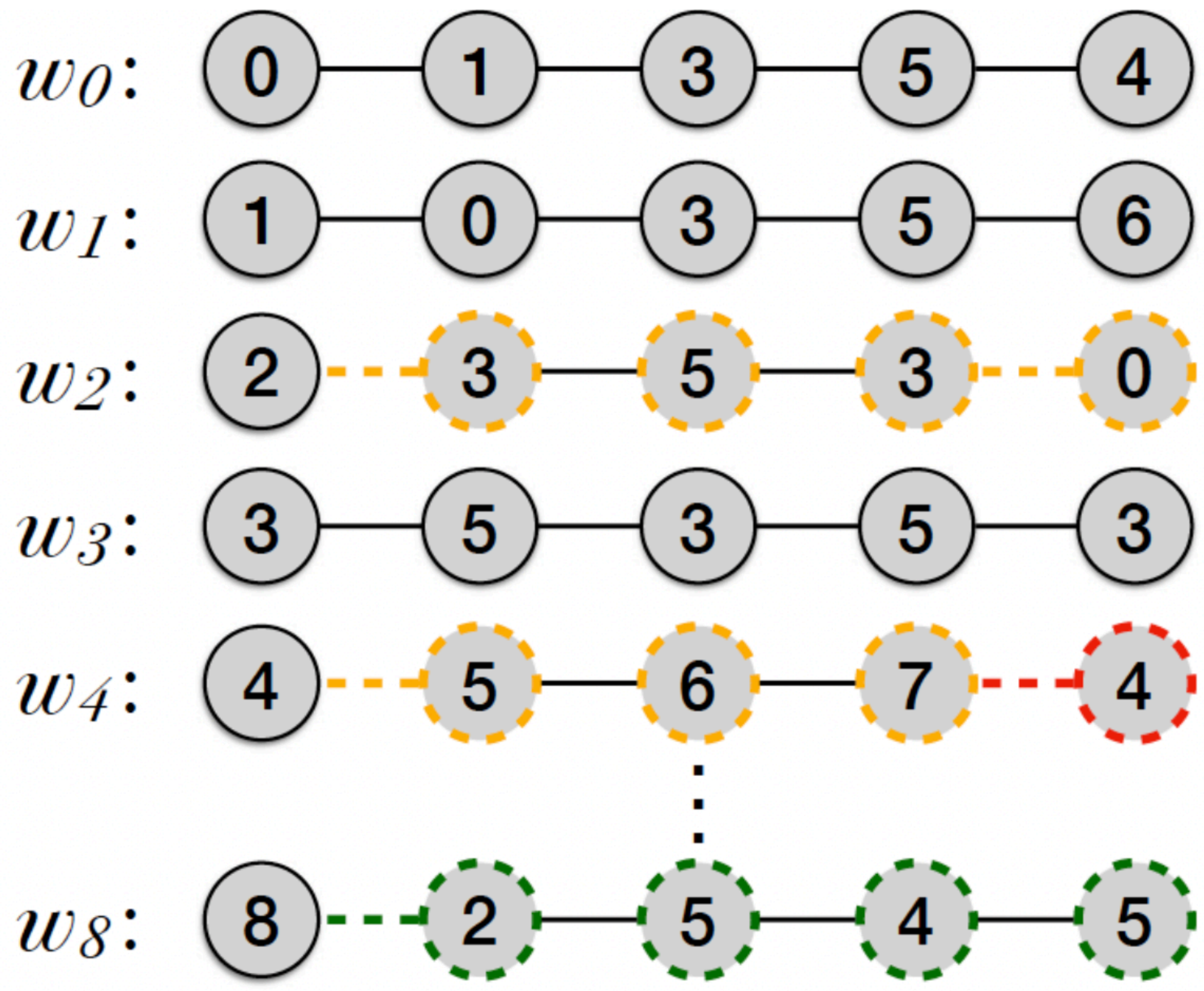
Problem: Efficiency and Space

- Efficient resampling of affected random walks (both *inconsistent* and *invalid*)



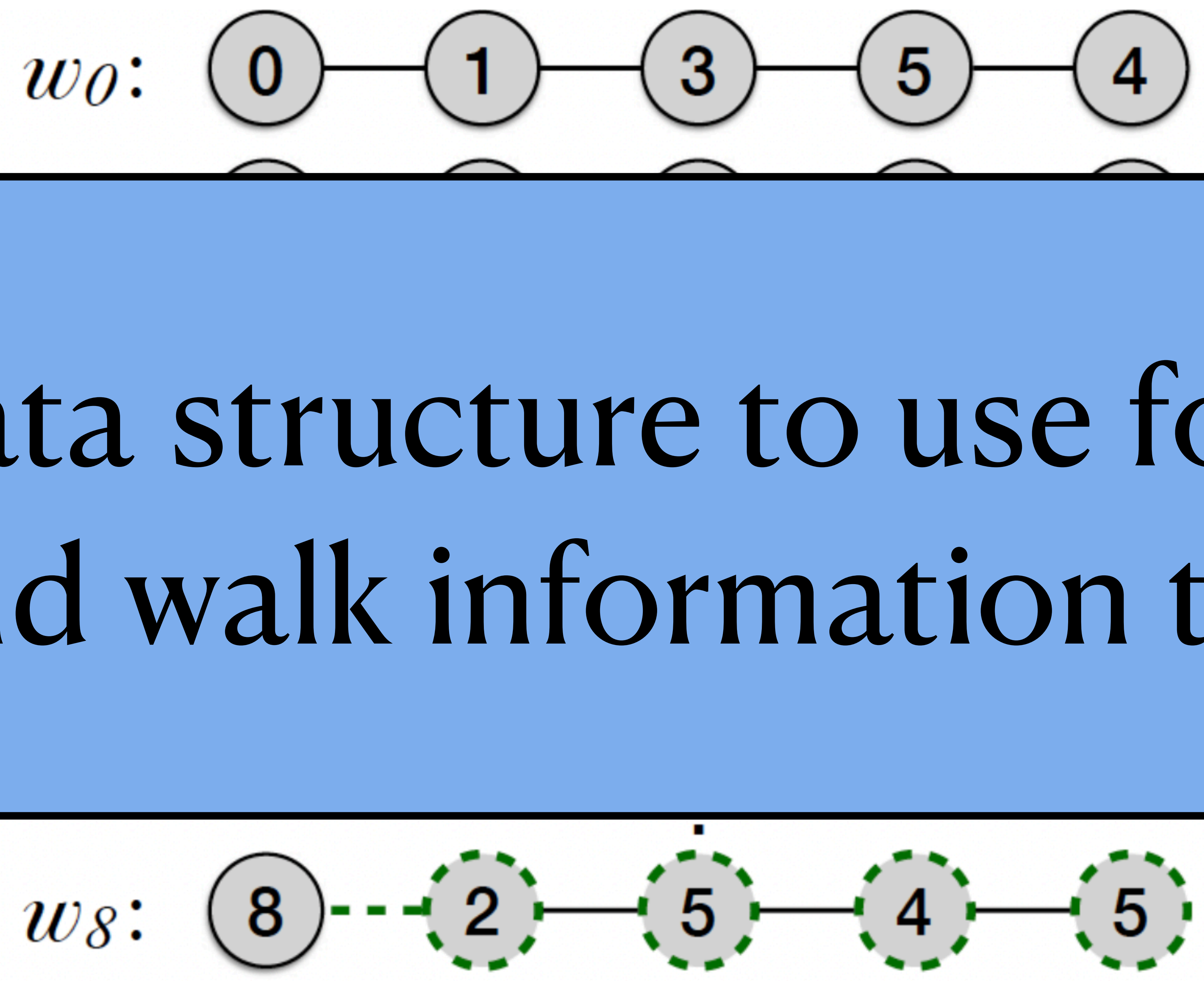
Problem: Efficiency and Space

- Efficient resampling of affected random walks (both *inconsistent* and *invalid*)
- Effective graph and random walk storage in main memory



Problem: Efficiency and Space

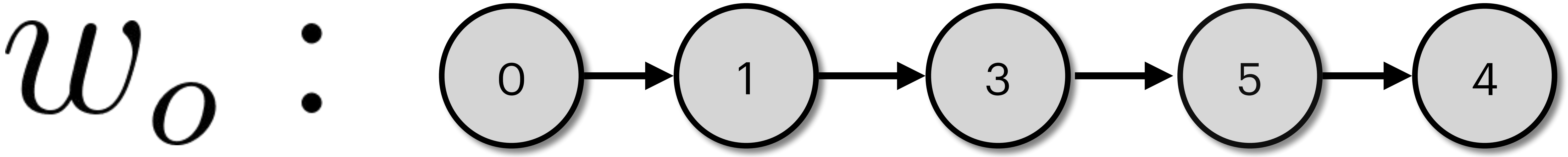
- Efficient resampling of affected random walks (both *inconsistent* and *invalid*)
- Effective graph and random walk storage in main memory



Which data structure to use for storing graph and walk information together?

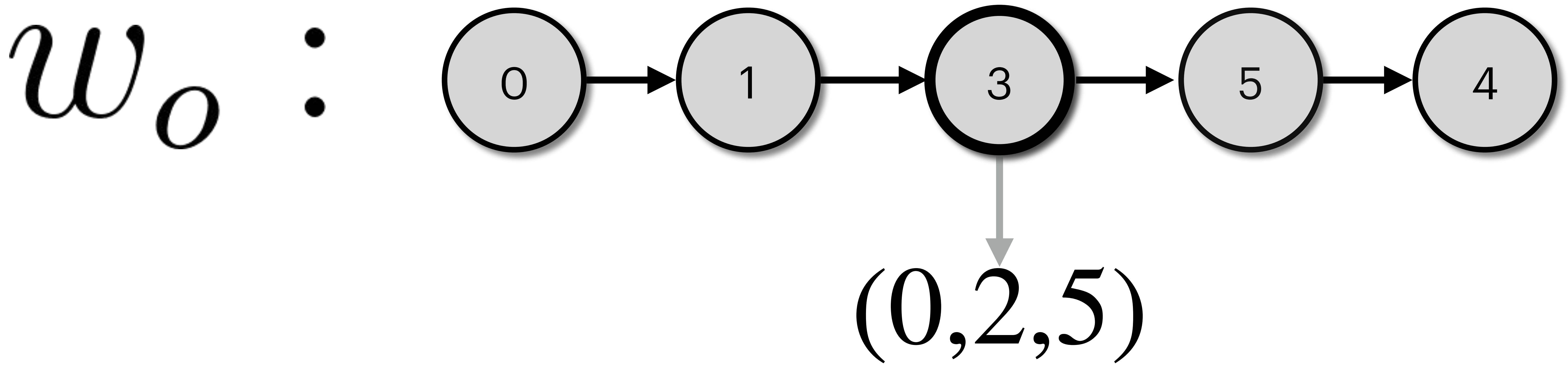
Random Walk Representation with Triplets

- A random walk is decomposed into l triplets: $(w_i, p_j, v_{w_i,p_{j+1}})$ where
  - w_i : walk identifier
  - p_j : position index
  - $v_{w_i,p_{j+1}}$: next vertex identifier



Random Walk Representation with Triplets

- A random walk is decomposed into l triplets: $(w_i, p_j, v_{w_i,p_{j+1}})$ where
  - w_i : walk identifier
  - p_j : position index
  - $v_{w_i,p_{j+1}}$: next vertex identifier



Encoding a Walk Triplet

- Encode a walk triplet into a single integer
  - Encode w_i, p_j into a single integer: $f(w_i, p_j) = w_i \times l + p_j$
  - Use the Szudzik pairing function to encode $f(w_i, p_j)$ and $v_{w_i, p_{j+1}}$: $< f(w_i, p_j), v_{w_i, p_{j+1}} >$
  - **Szudzik** Pairing Function

$$Szudzik(x, y) = \begin{cases} y^2 + x & \text{if } x < y \\ x^2 + x + y & \text{if } x \geq y \end{cases}$$

$$Szudzik^{-1}(z) = \begin{cases} \{z - \lfloor \sqrt{z} \rfloor^2, \lfloor \sqrt{z} \rfloor\} & \text{if } z - \lfloor \sqrt{z} \rfloor^2 < \lfloor \sqrt{z} \rfloor \\ \{\lfloor \sqrt{z} \rfloor, z - \lfloor \sqrt{z} \rfloor^2 - \lfloor \sqrt{z} \rfloor\} & \text{if } z - \lfloor \sqrt{z} \rfloor^2 \geq \lfloor \sqrt{z} \rfloor \end{cases}$$

Encoding a Walk Triplet

- Encode a walk triplet into a single integer
  - Encode w_i, p_j into a single integer: $f(w_i, p_j) = w_i \times l + p_j$
  - Use the Szudzik pairing function to encode $f(w_i, p_j)$ and $v_{w_i, p_{j+1}}$: $< f(w_i, p_j), v_{w_i, p_{j+1}} >$
  - **Szudzik** Pairing Function

$$Szudzik(x, y) = \begin{cases} y^2 + x & \text{if } x < y \\ x^2 + x + y & \text{if } x \geq y \end{cases}$$

$$Szudzik^{-1}(z) = \begin{cases} \{z - \lfloor\sqrt{z}\rfloor^2, \lfloor\sqrt{z}\rfloor\} & \text{if } z - \lfloor\sqrt{z}\rfloor^2 < \lfloor\sqrt{z}\rfloor \\ \{\lfloor\sqrt{z}\rfloor, z - \lfloor\sqrt{z}\rfloor^2 - \lfloor\sqrt{z}\rfloor\} & \text{if } z - \lfloor\sqrt{z}\rfloor^2 \geq \lfloor\sqrt{z}\rfloor \end{cases}$$

Unified Walk and Graph Storage

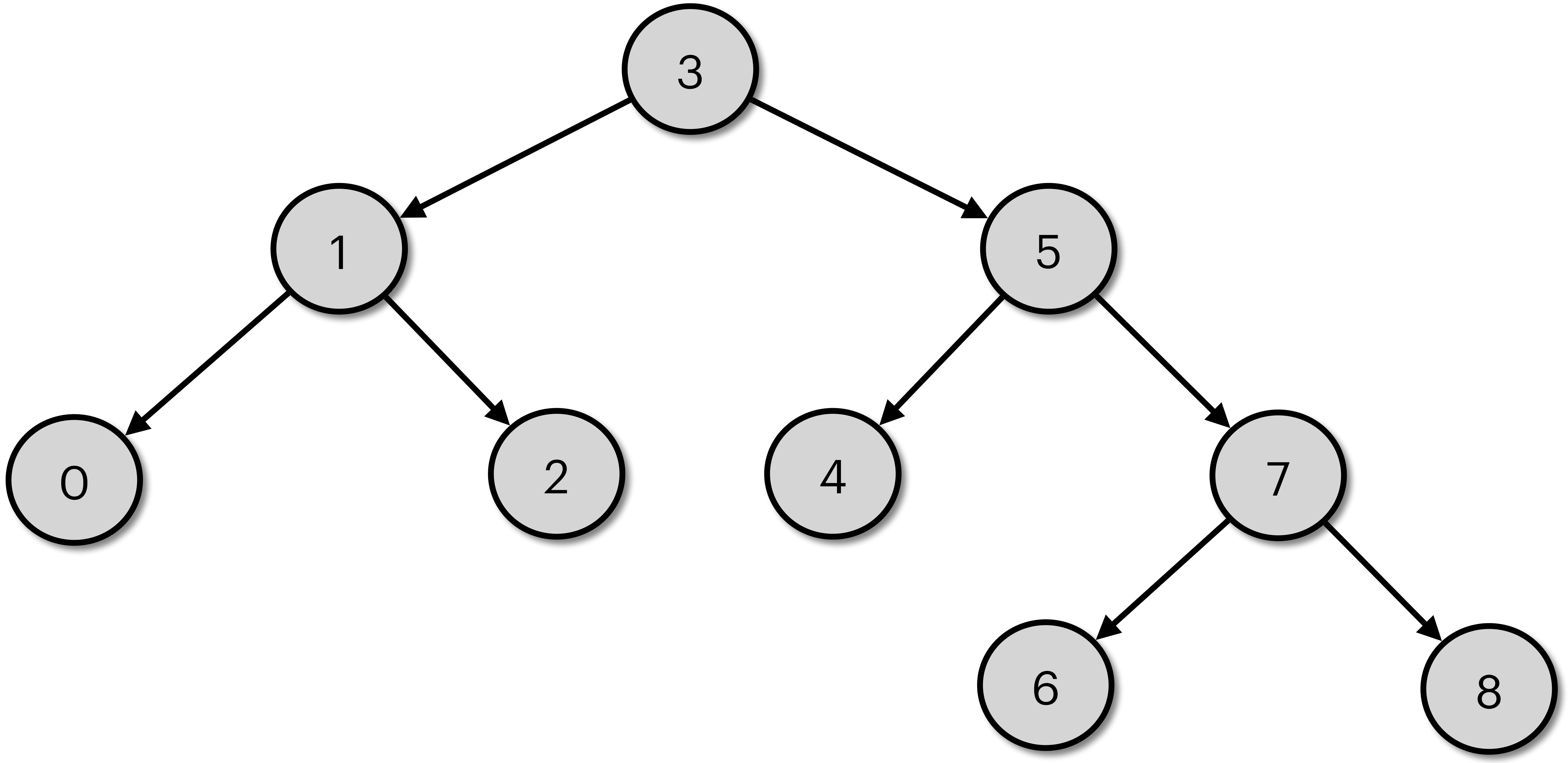
***Wharf**'s Tree of trees Data Structure*

- Trees of trees structure
  (Level 1) Vertex-tree (Purely-Functional Binary Tree)
  (Level 2) Walk-tree & Edge-tree (**Compressed** Purely-Functional Binary Tree [1])

[1] Laxman Dhulipala, Guy Blelloch, and Julian Shun. Low-Latency Graph Streaming Using Compressed Purely-Functional Trees. PLDI'19

Unified Walk and Graph Storage

Wharf*'s Tree of trees Data Structure*

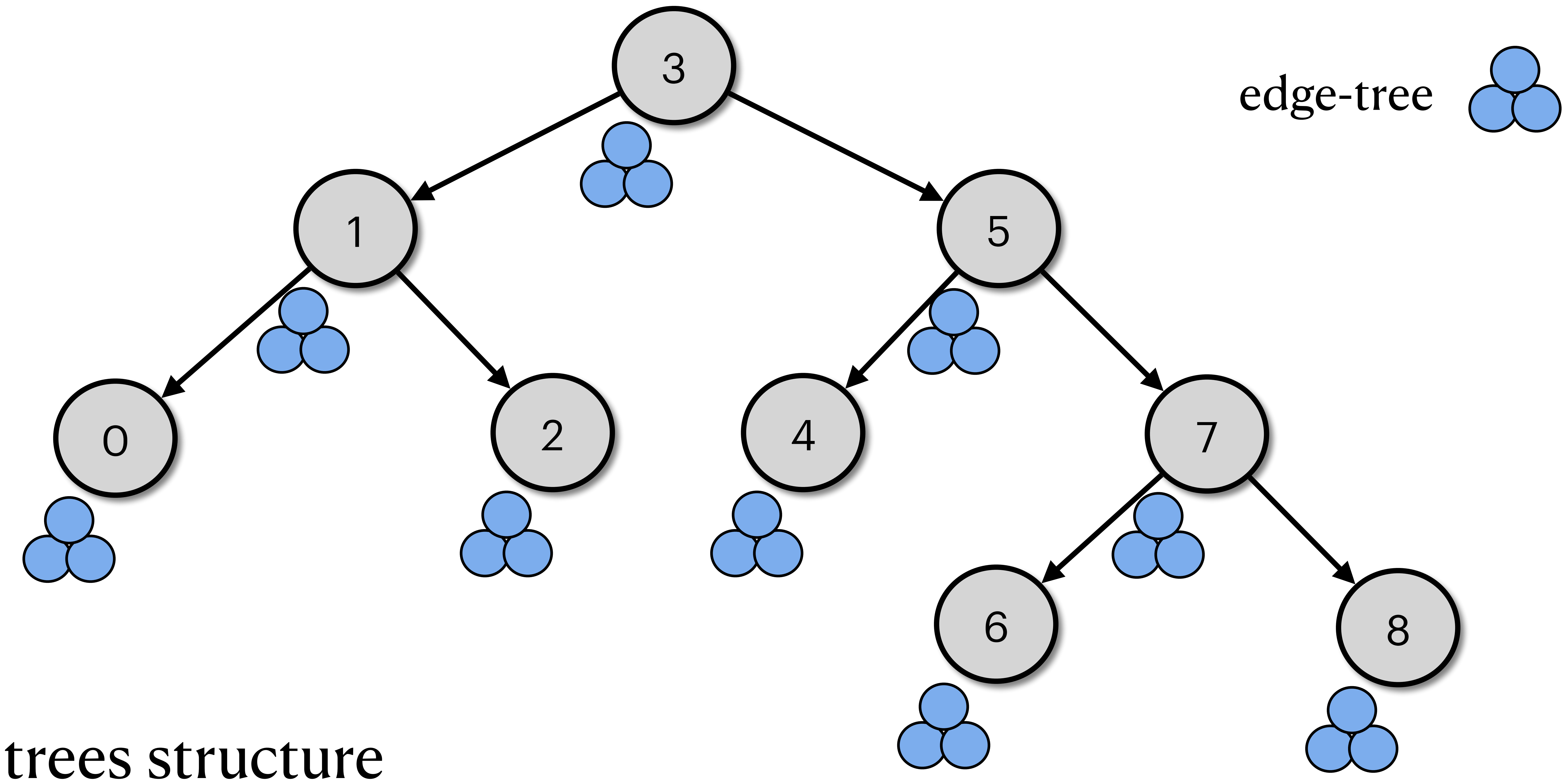


- Trees of trees structure
  (Level 1) Vertex-tree (Purely-Functional Binary Tree)
  (Level 2) Walk-tree & Edge-tree (**Compressed** Purely-Functional Binary Tree [1])

[1] Laxman Dhulipala, Guy Blelloch, and Julian Shun. Low-Latency Graph Streaming Using Compressed Purely-Functional Trees. PLDI'19

Unified Walk and Graph Storage

***Wharf**'s Tree of trees Data Structure*

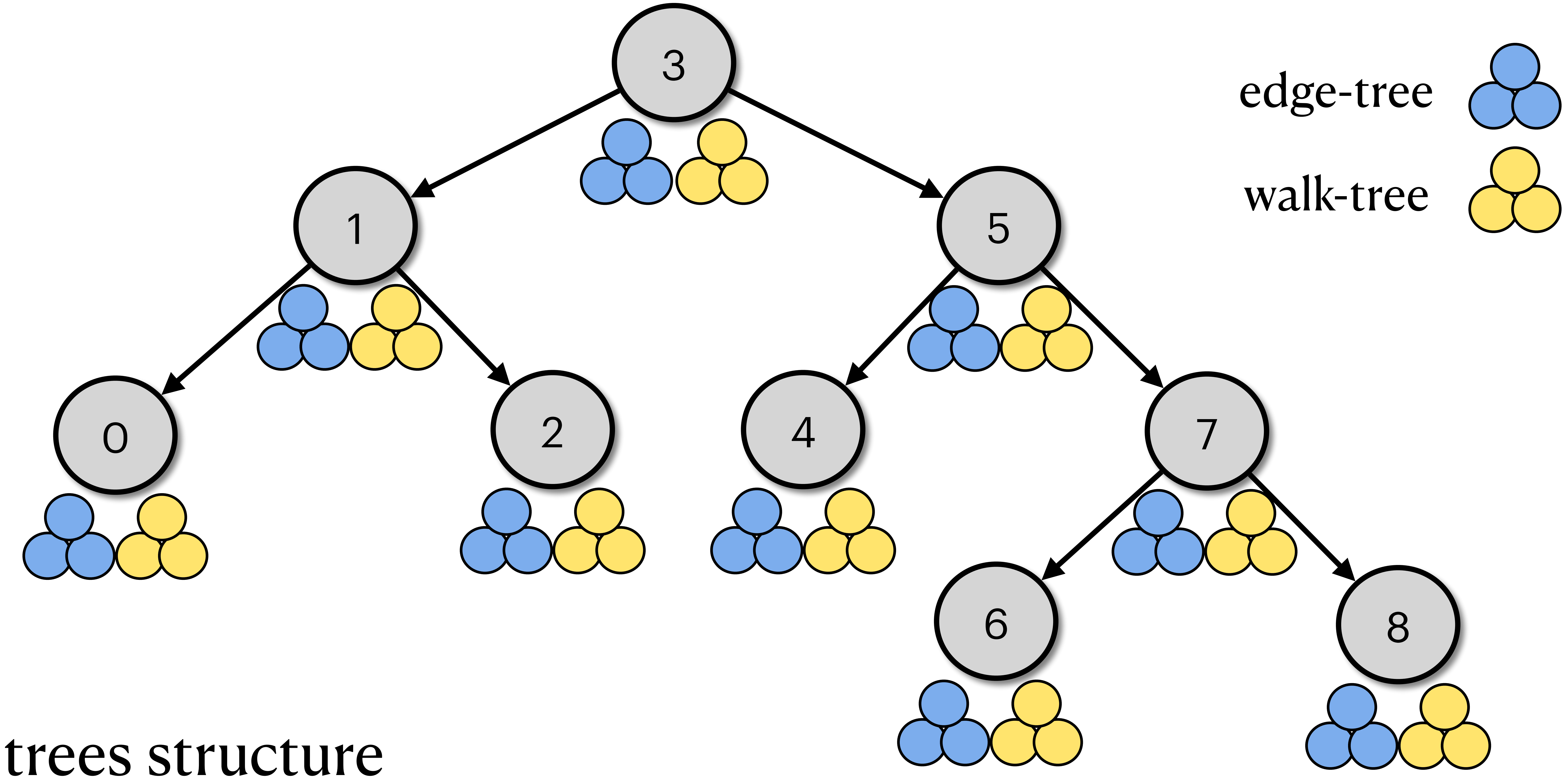


- Trees of trees structure
  (Level 1) Vertex-tree (Purely-Functional Binary Tree)
  (Level 2) Walk-tree & Edge-tree (**Compressed** Purely-Functional Binary Tree [1])

[1] Laxman Dhulipala, Guy Blelloch, and Julian Shun. Low-Latency Graph Streaming Using Compressed Purely-Functional Trees. PLDI'19

Unified Walk and Graph Storage

Wharf*'s Tree of trees Data Structure*



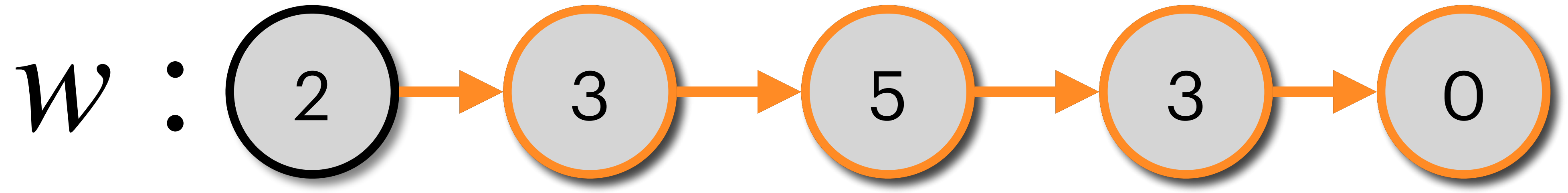
- Trees of trees structure
  (Level 1) Vertex-tree (Purely-Functional Binary Tree)
  (Level 2) Walk-tree & Edge-tree (**Compressed** Purely-Functional Binary Tree [1])

[1] Laxman Dhulipala, Guy Blelloch, and Julian Shun. Low-Latency Graph Streaming Using Compressed Purely-Functional Trees. PLDI'19

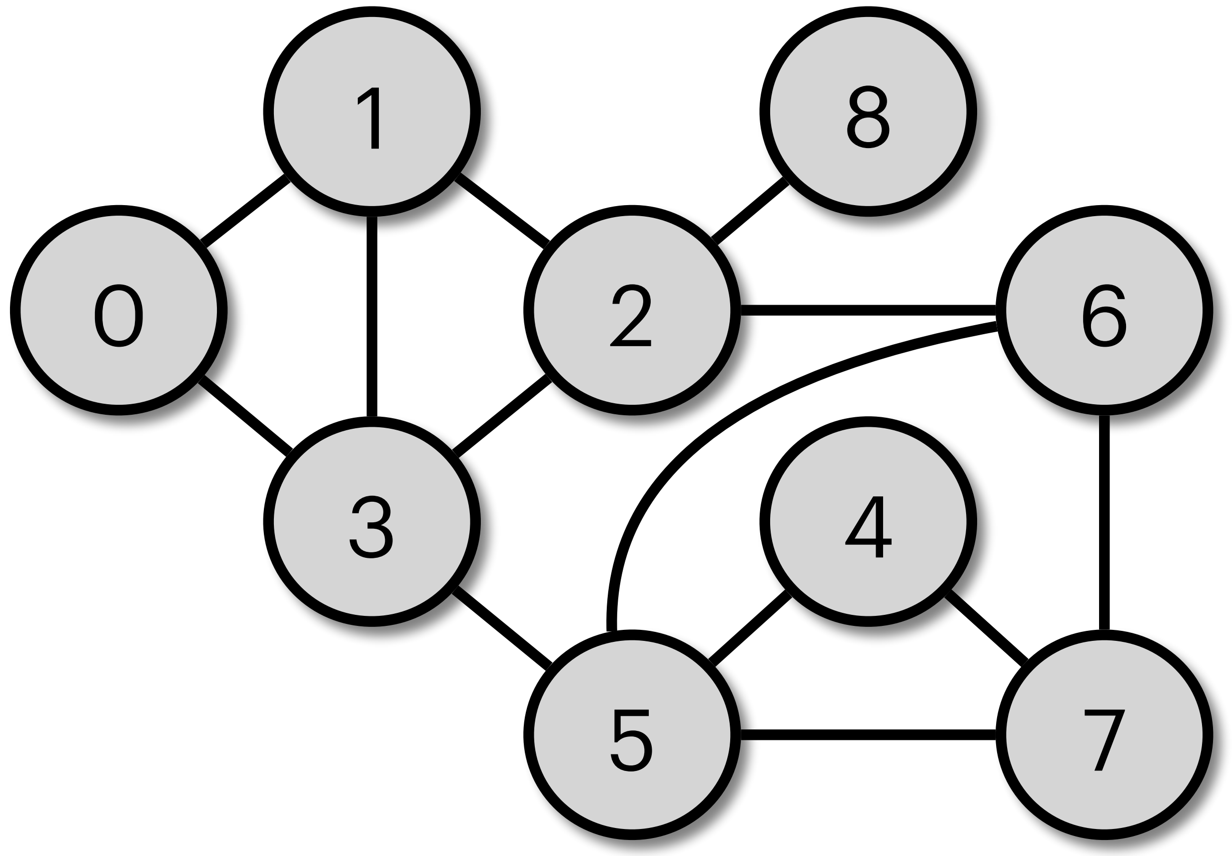
Map of Affected Vertices

Which subparts random walks need resampling?

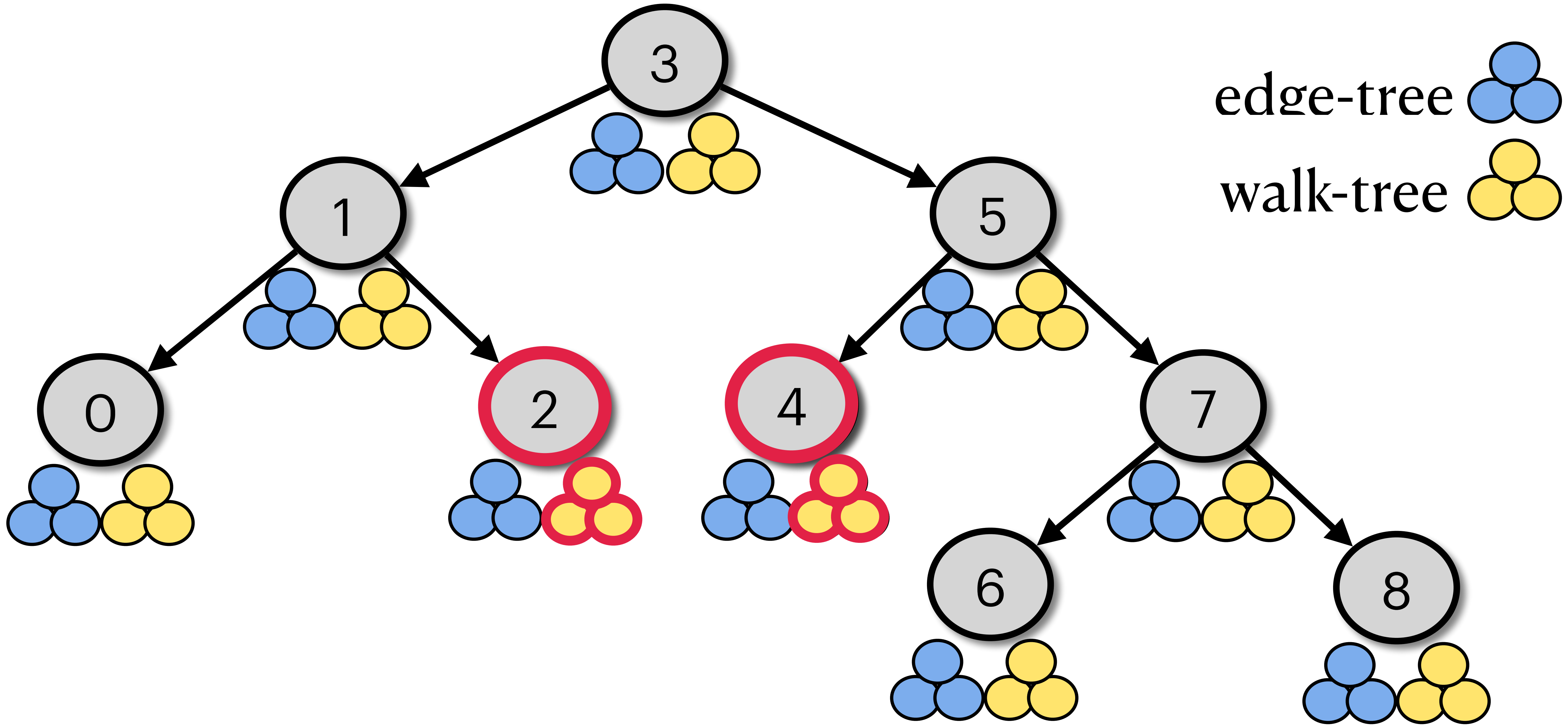
- Find the **earliest affected** position
- Save the Key-Value pair, K: w_i V: $\{v_{min}, p_{min}\}$
- Assume edge $\{4,2\}$ *gets inserted*



Graph (Visually)



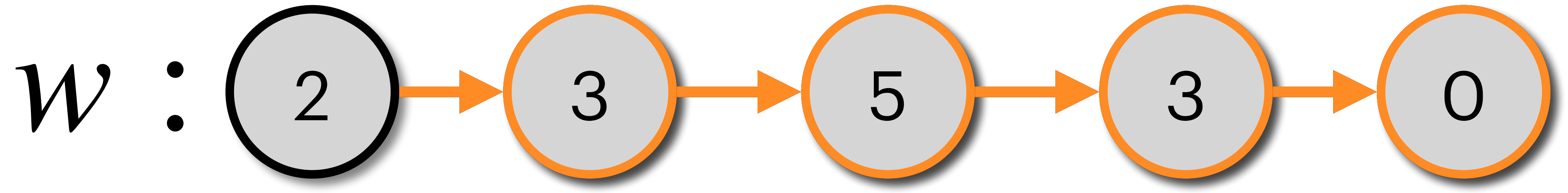
***Wharf**'s Tree of tree Structure*



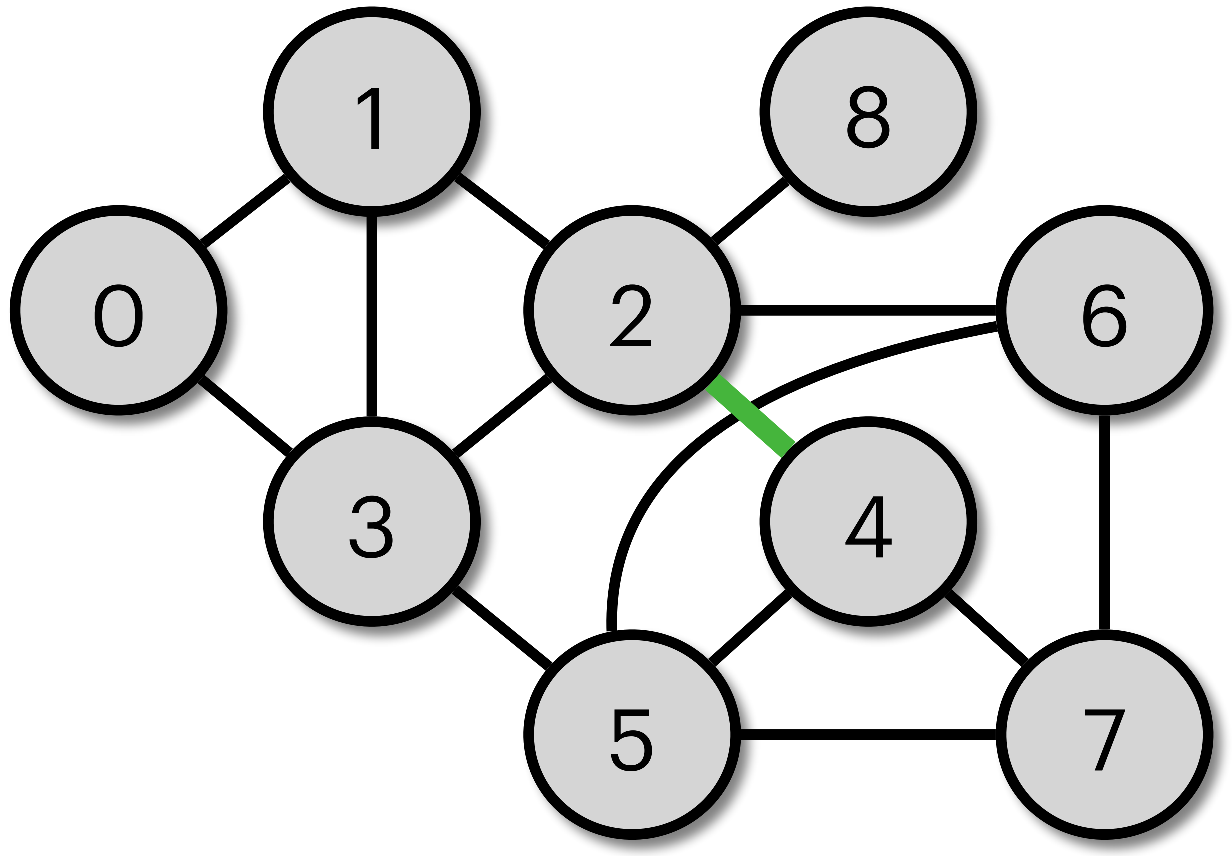
Map of Affected Vertices

Which subparts random walks need resampling?

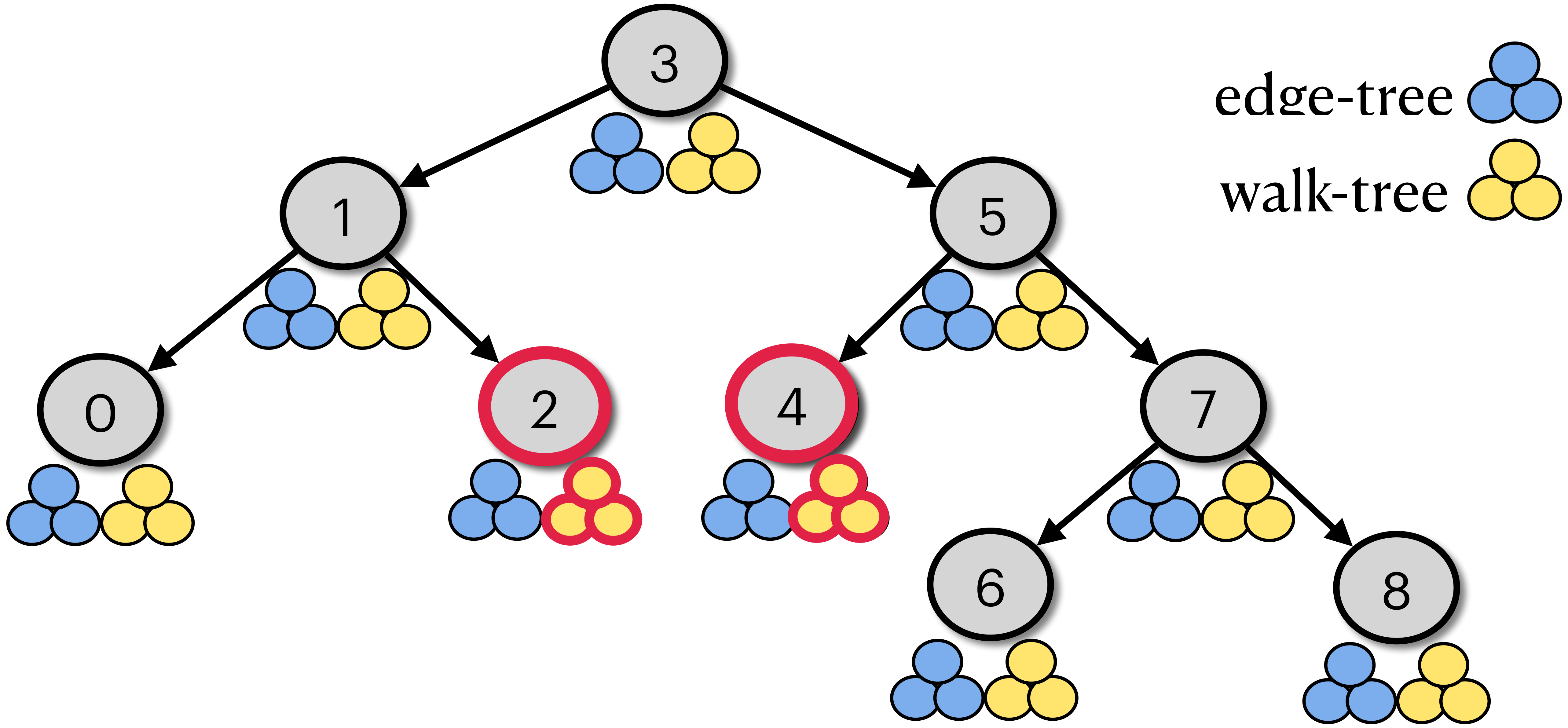
- Find the **earliest affected** position
- Save the Key-Value pair, K: w_i V: $\{v_{min}, p_{min}\}$
- Assume edge $\{4{,}2\}$ *gets inserted*



Graph (Visually)

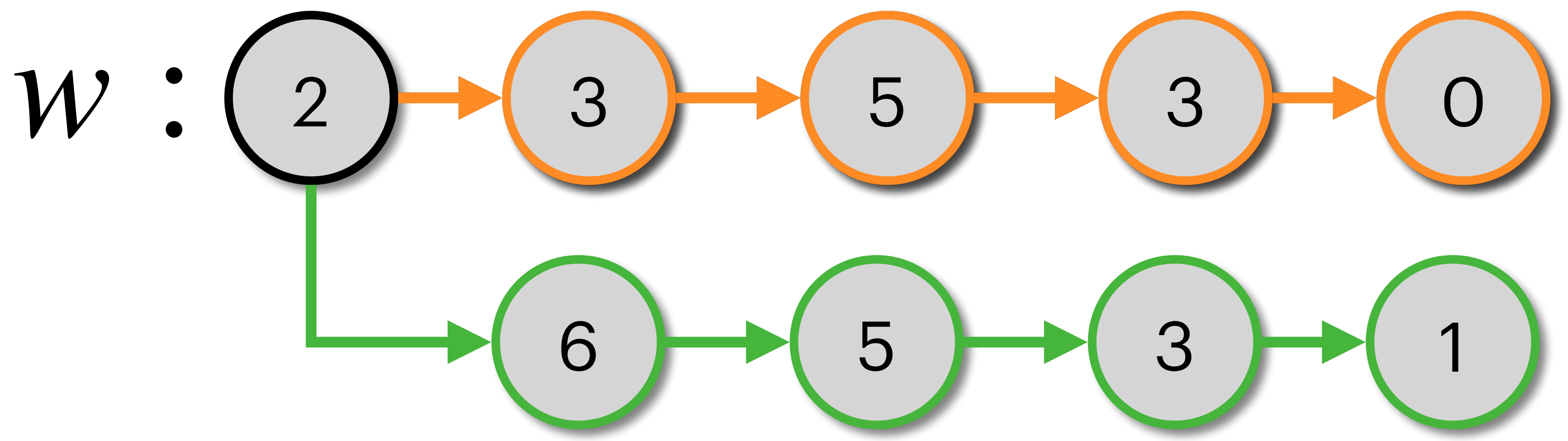


***Wharf**'s Tree of tree Structure*

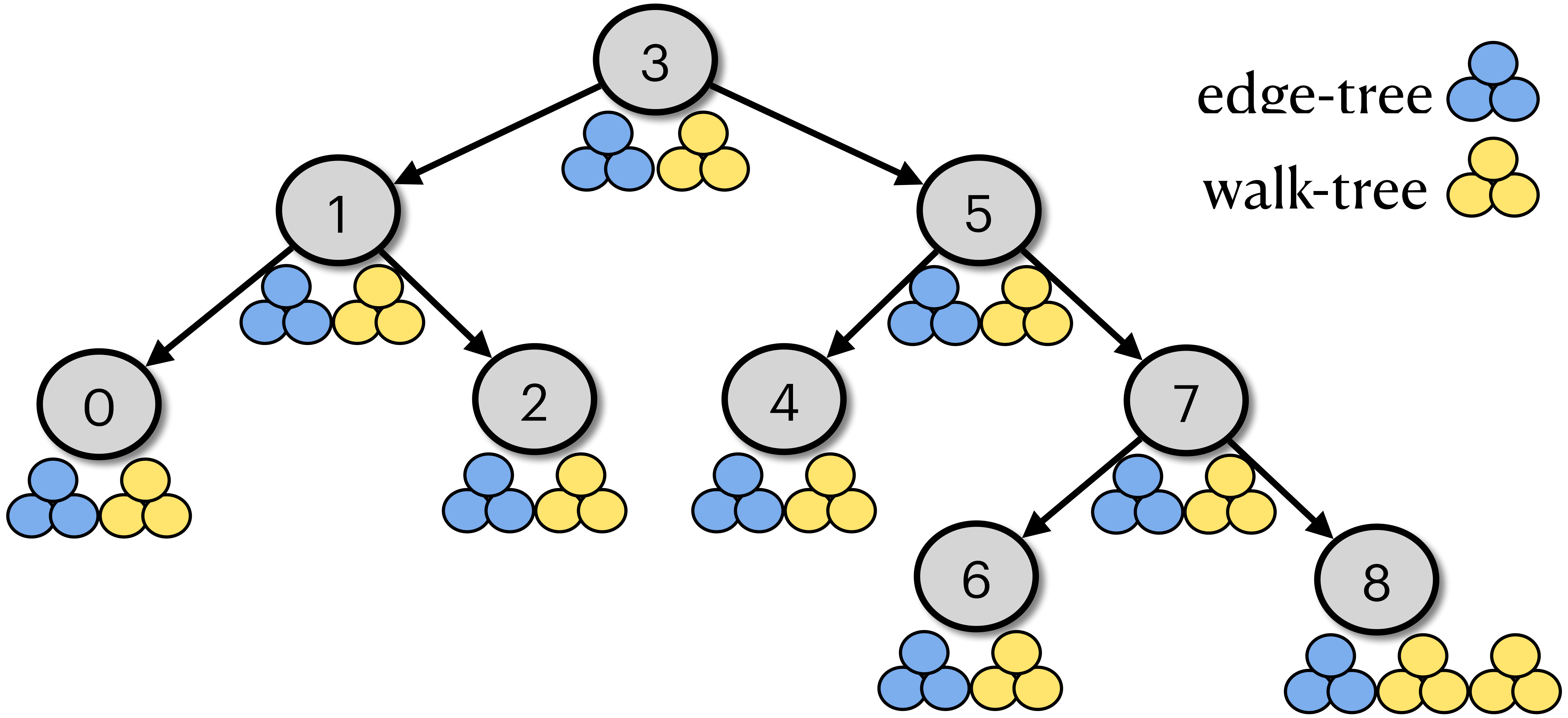


Updating Affected Random Walks

- Recreate a walk from its **earliest** affected position
  - Create new walk triplets and insert them into their corresponding walk-tree
  - Delete obsolete walk triplets and *merge* walk-trees under each vertex of the vertex-tree

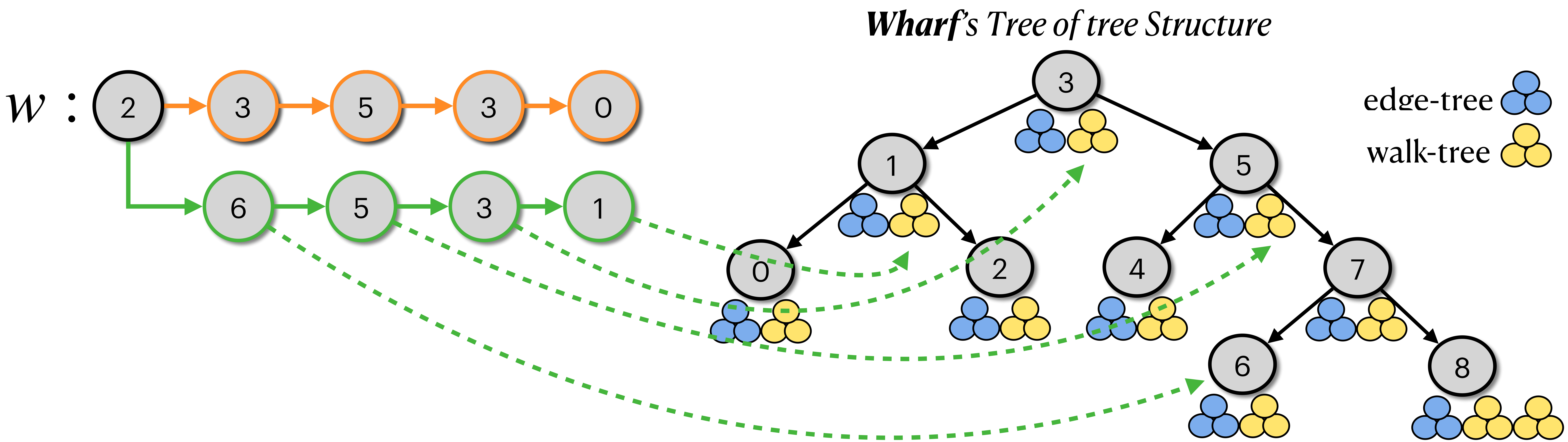


Wharf*'s *Tree of tree Structure



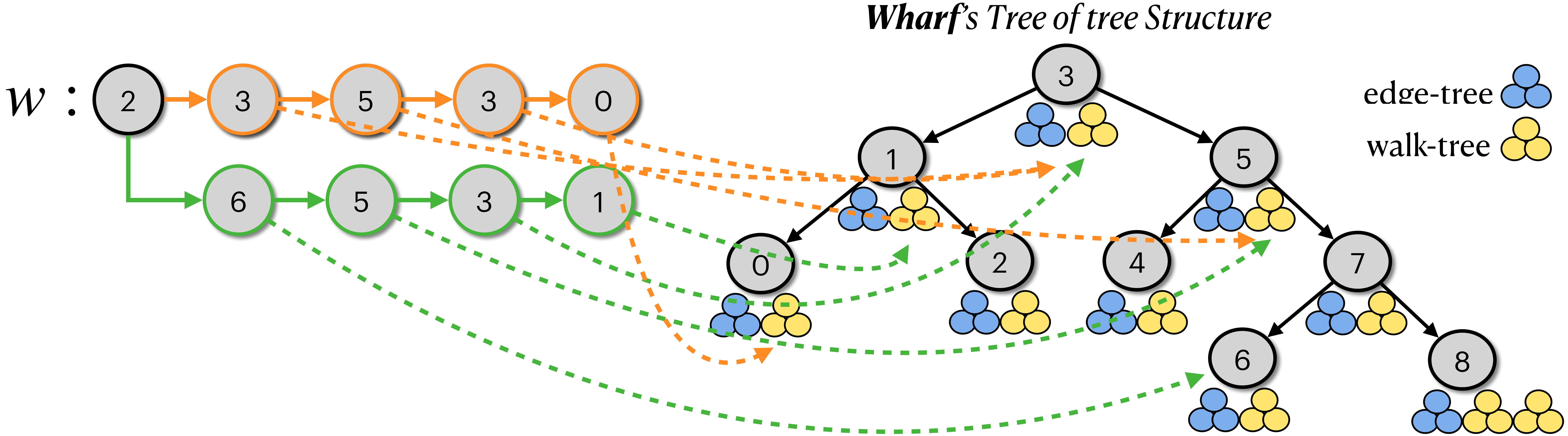
Updating Affected Random Walks

- Recreate a walk from its **earliest** affected position
  - Create new walk triplets and insert them into their corresponding walk-tree
  - Delete obsolete walk triplets and *merge* walk-trees under each vertex of the vertex-tree



Updating Affected Random Walks

- Recreate a walk from its **earliest** affected position
  - Create new walk triplets and insert them into their corresponding walk-tree
  - Delete obsolete walk triplets and *merge* walk-trees under each vertex of the vertex-tree



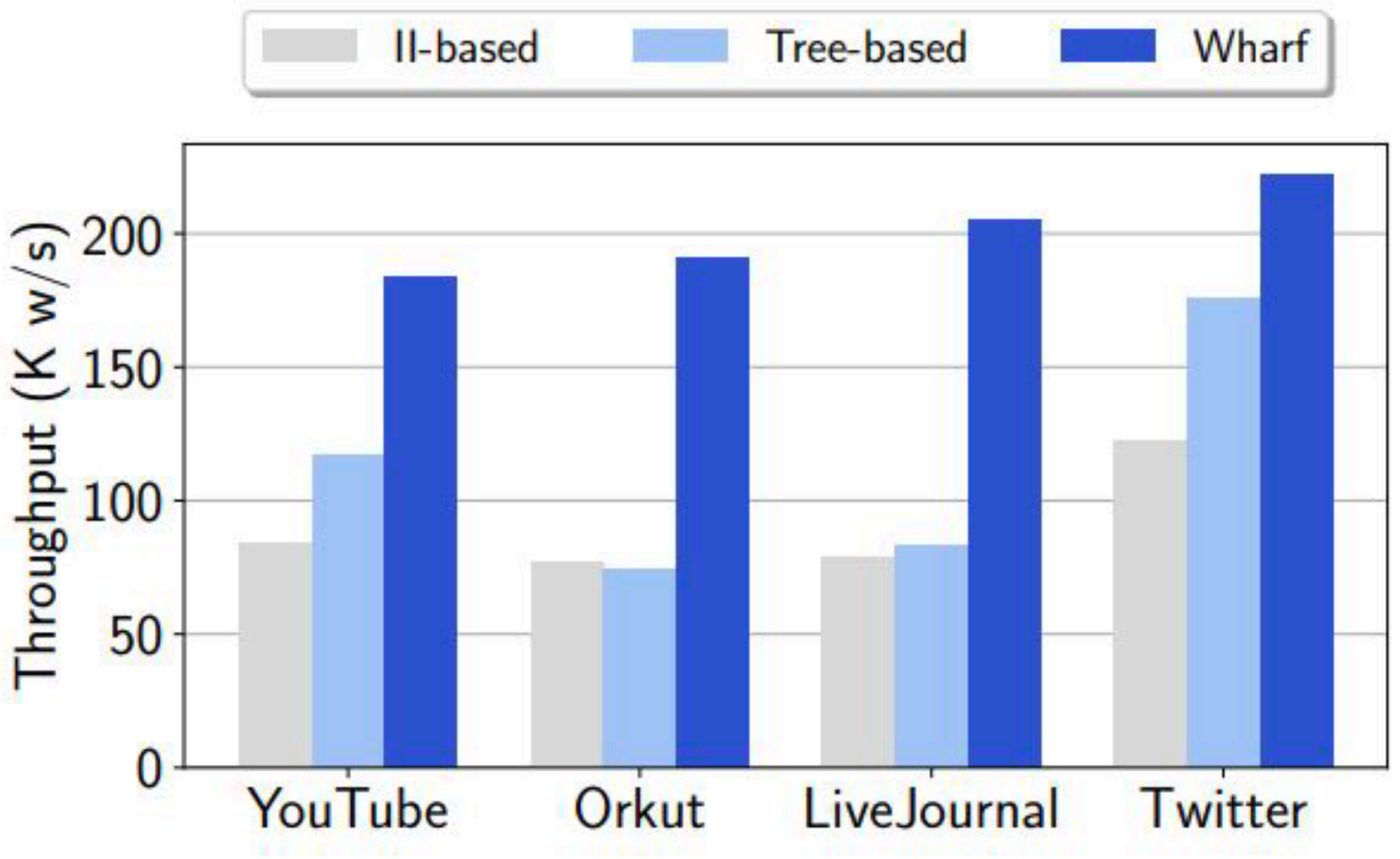
Optimised Search

How to enable fast search among encoding walk triplets?

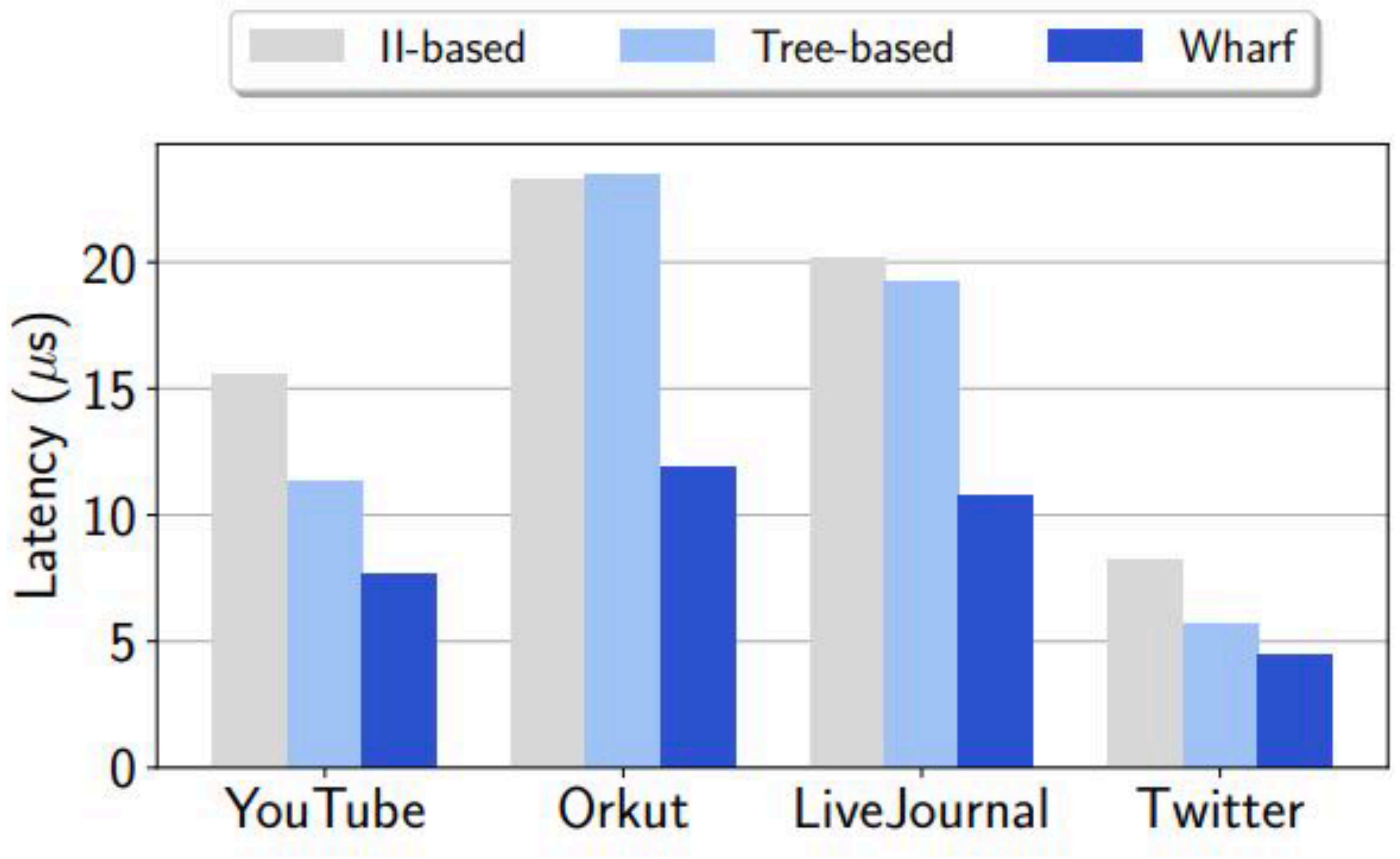
- Seek of a specific walk triplet, search for a specific integer
  - **Worst-case**: decode all triplets
  - **Better Solution**: use ordering properties of pairing functions
    - Restrict the search of a triplet-integers within a range of the form $\{lb, ub\}$ where
      $lb = < w \times l + p, v_{w,p+1}^{min} >$
      $ub = < w \times l + p, v_{w,p+1}^{max} >$
      while maintaining the min and max *next vertex identifier* in each walk-tree

Experimental Study: Throughput & Latency

Task: Random Walk Corpus Update



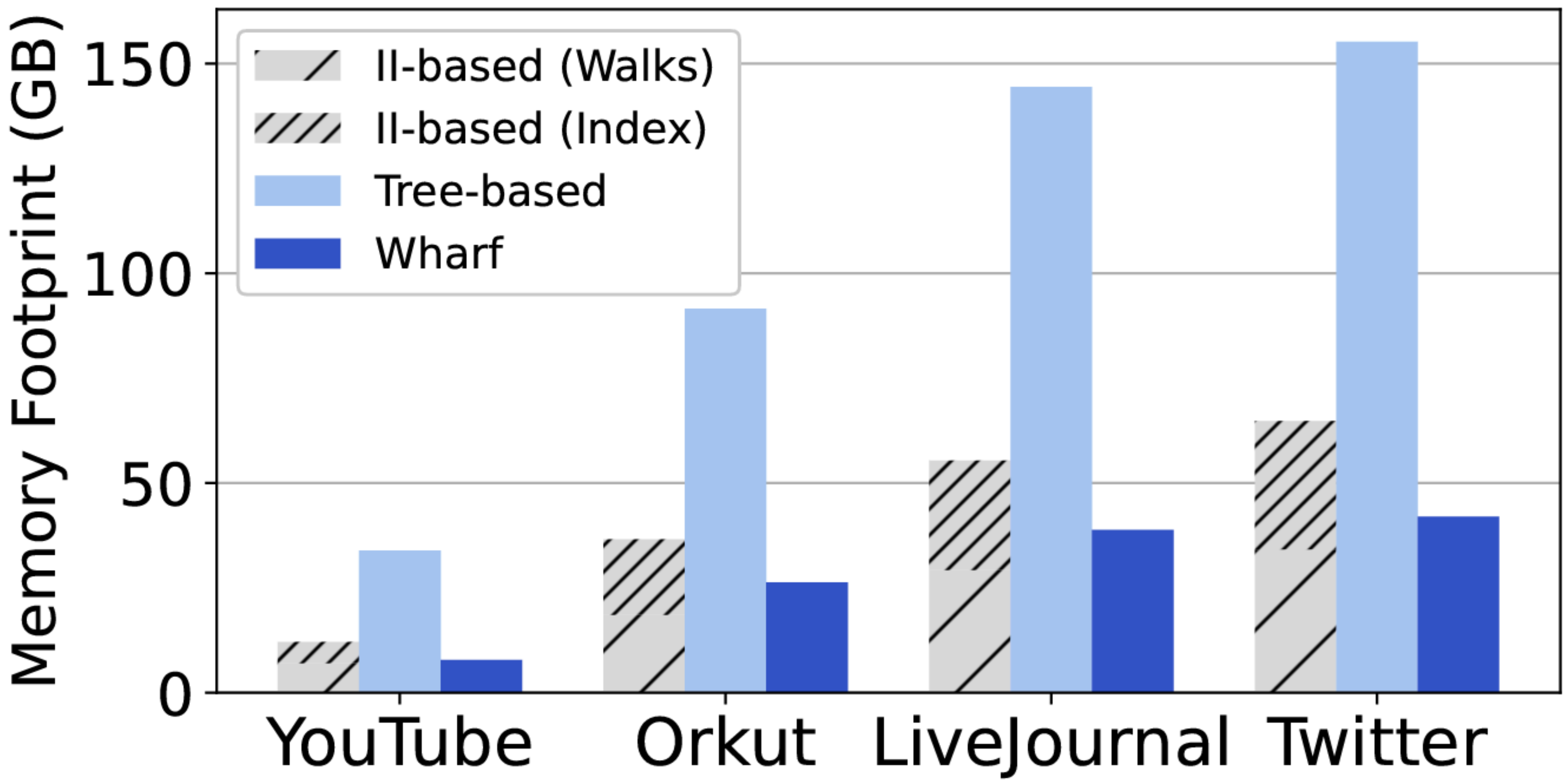
(a) Throughput



(b) Latency

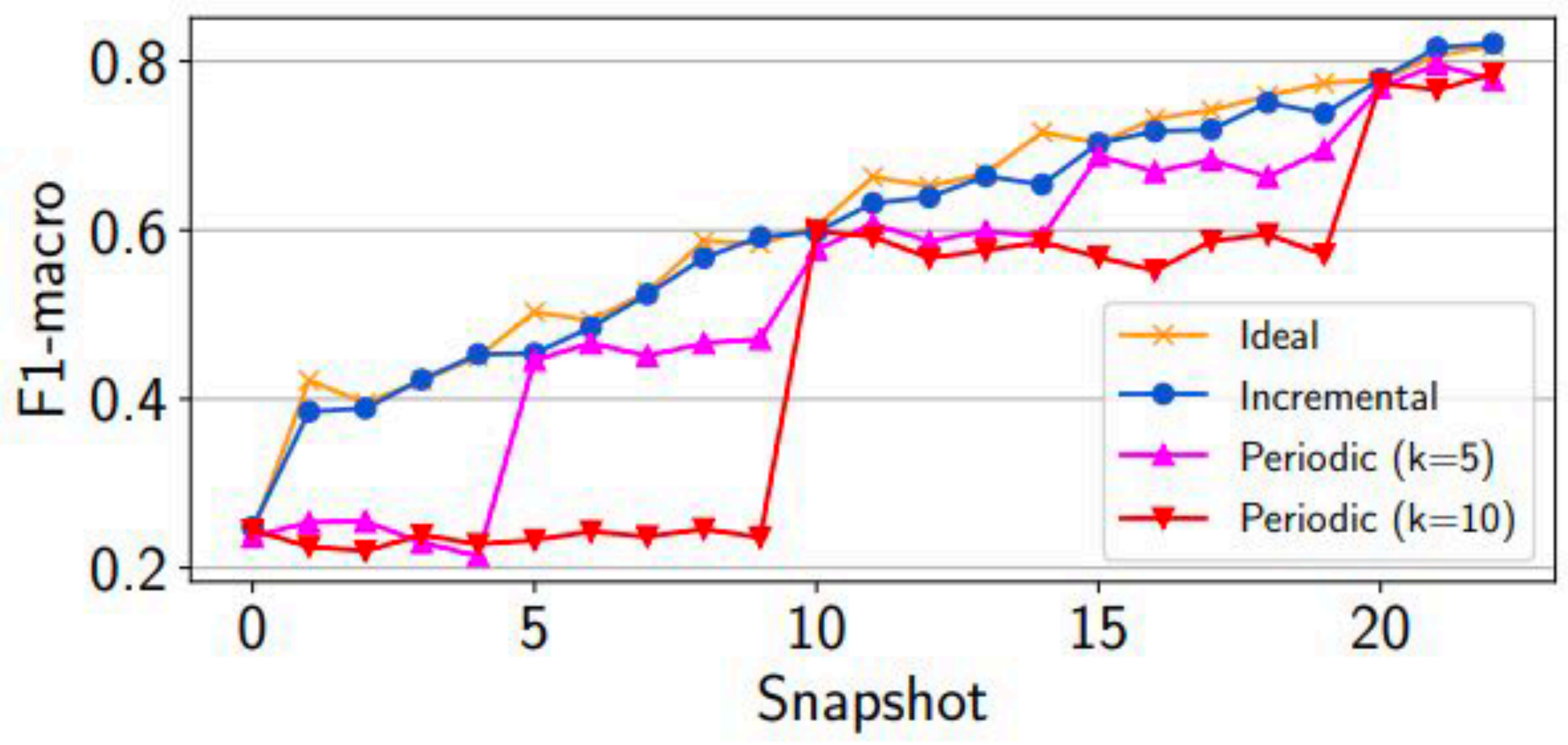
Experimental Study: Memory Footprint

Task: Random Walk Corpus Space Consumption

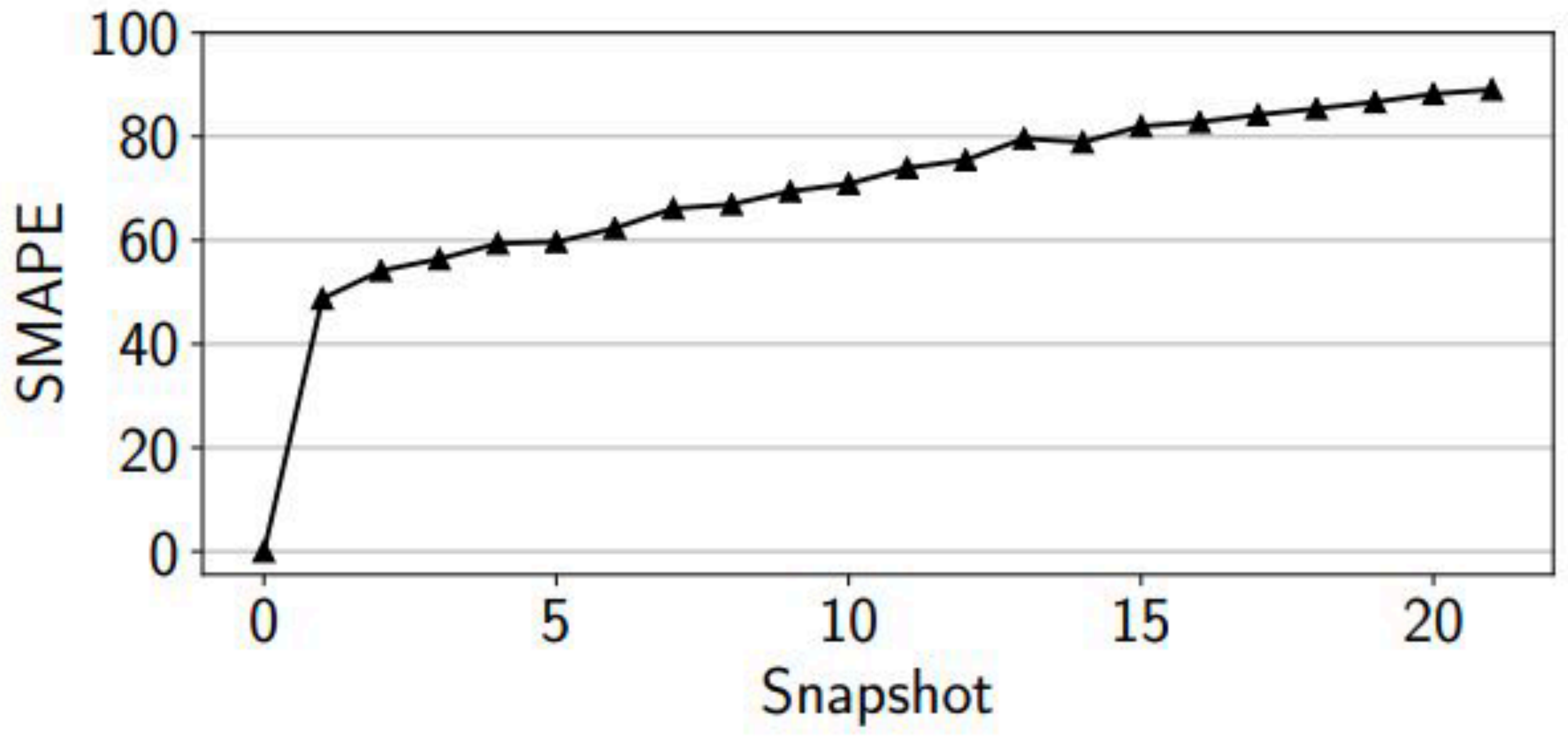


Experimental Study: Downstream Tasks

Tasks: Incremental Graph Embedding, Incremental Personalised PageRank



(a) Ver. Classification (DeepWalk)



(b) Personalized PageRank

Takeaways

- **Key challenges** in apps using whole *random walk corpuses* sampled from streaming graphs:

Efficiency + *Space*

- Our solution: **Wharf**
  - Efficient batch updates on whole *random walk corpuses*
  - Space-efficient walk representation by coupling C-trees with pairing functions

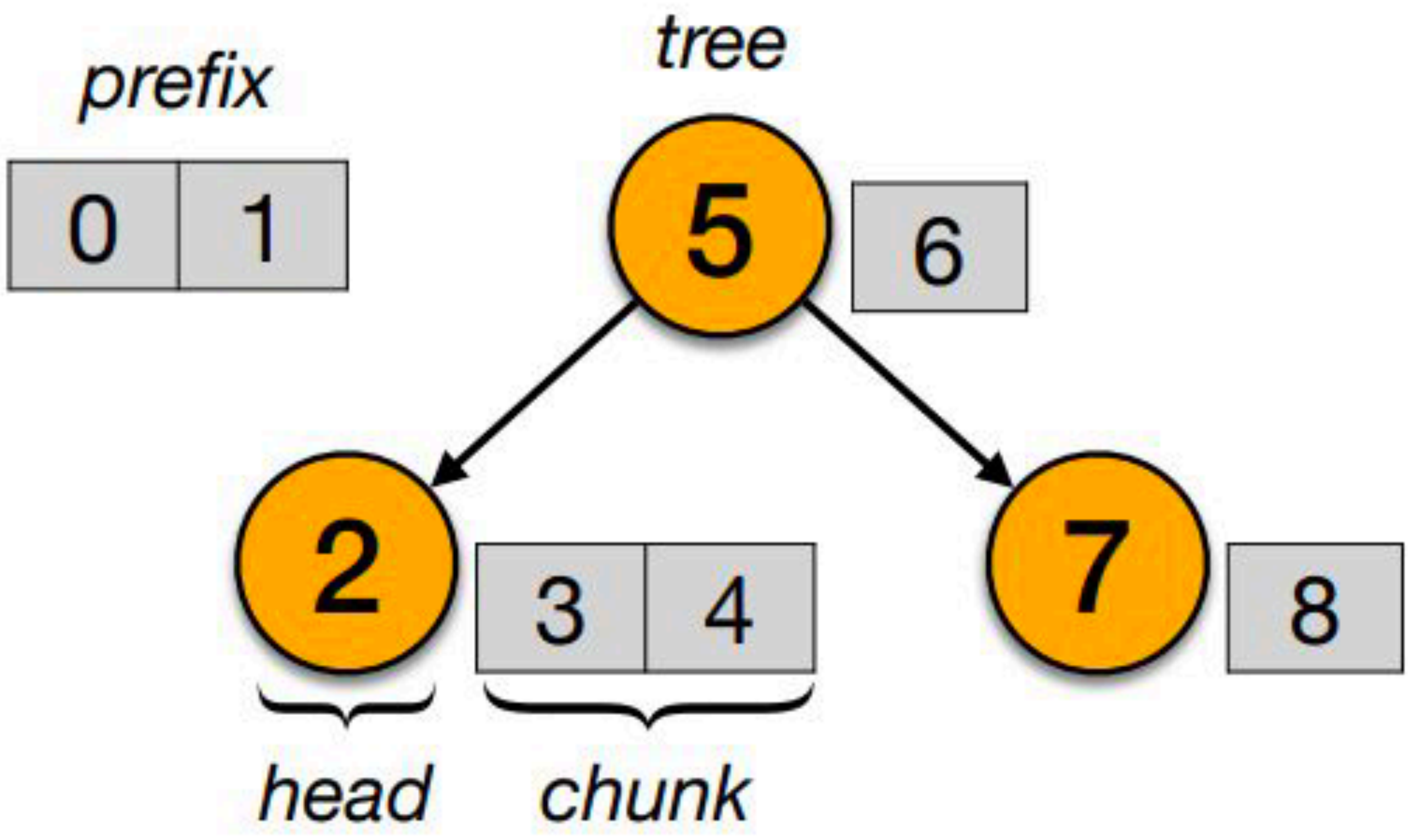
- **Wharf** achieves up to 2.6 × times higher throughput, up to 2 × lower latency, and is up to 4.4 × more space-efficient that the baselines

Space-Efficient Random Walks on Streaming Graphs

Thank you!

C-trees

Compressed Purely-Functional Trees



[1] Laxman Dhulipala, Guy Blelloch, and Julian Shun. Low-Latency Graph Streaming Using Compressed Purely-Functional Trees. PLDI'19

Additional Formulas

- Triplet Decoding ($f(w_i, p_j) = w_i \times + p_j$):
  $p_j = f(w_i, p_j) \mod l$ AND $w_i = \lfloor \frac{f}{l} \rfloor$
- Ordering Properties (Corollary 1)
  $x + y < x' + y' \rightarrow \langle x, y \rangle \leq \langle x', y' \rangle$
- **Szudzik** Pairing Function

$$Szudzik(x, y) = \begin{cases} y^2 + x & \text{if } x < y \\ x^2 + x + y & \text{if } x \geq y \end{cases}$$

$$Szudzik^{-1}(z) = \begin{cases} \{z - \lfloor \sqrt{z} \rfloor^2, \lfloor \sqrt{z} \rfloor\} & \text{if } z - \lfloor \sqrt{z} \rfloor^2 < \lfloor \sqrt{z} \rfloor \\ \{\lfloor \sqrt{z} \rfloor, z - \lfloor \sqrt{z} \rfloor^2 - \lfloor \sqrt{z} \rfloor\} & \text{if } z - \lfloor \sqrt{z} \rfloor^2 \geq \lfloor \sqrt{z} \rfloor \end{cases}$$

Additional Formulas

- Triplet Decoding ($f(w_i, p_j) = w_i \times + p_j$):

  $p_j = f(w_i, p_j) \mod l$ AND $w_i = \lfloor \frac{f}{l} \rfloor$

- Ordering Properties (Corollary 1)

  $x + y < x' + y' \rightarrow \, < x, y > \, \leq \, < x', y' >$

- **Szudzik** Pairing Function

$$Szudzik(x, y) = \begin{cases} y^2 + x & \text{if } x < y \\ x^2 + x + y & \text{if } x \geq y \end{cases}$$

$$Szudzik^{-1}(z) = \begin{cases} \{z - \lfloor \sqrt{z} \rfloor^2, \lfloor \sqrt{z} \rfloor\} & \text{if } z - \lfloor \sqrt{z} \rfloor^2 < \lfloor \sqrt{z} \rfloor \\ \{\lfloor \sqrt{z} \rfloor, z - \lfloor \sqrt{z} \rfloor^2 - \lfloor \sqrt{z} \rfloor\} & \text{if } z - \lfloor \sqrt{z} \rfloor^2 \geq \lfloor \sqrt{z} \rfloor \end{cases}$$

Datasets Statistics

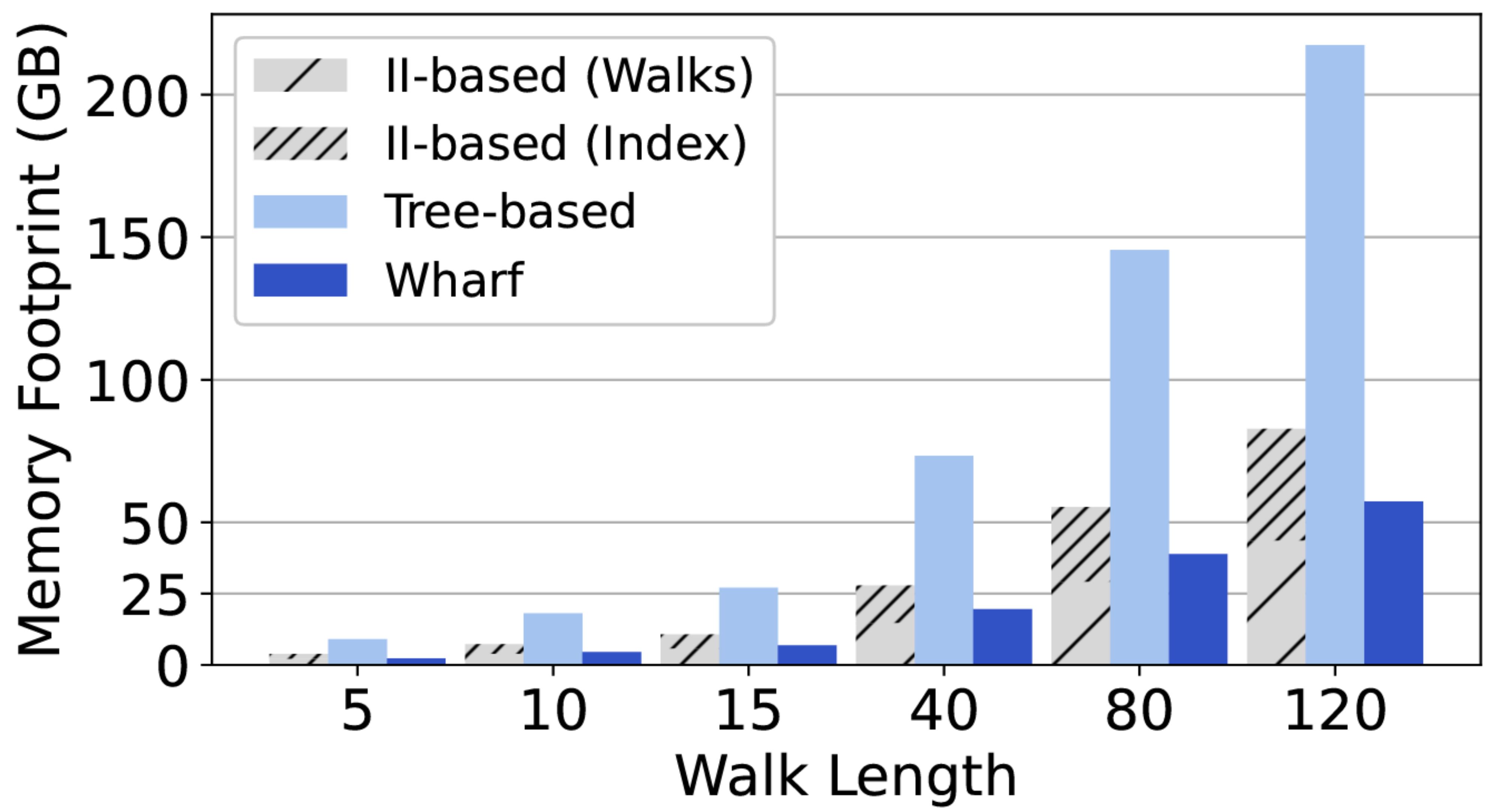
Graph	Num. Vertices	Num. Edges	Avg. Degree
com-YouTube	1,134,890	2,987,624	5.30
soc-LiveJournal	4,847,571	85,702,474	17.80
com-Orkut	3,072,627	234,370,166	76.20
Twitter	41,652,230	1,468,365,182	57.70

- Additionally, we generated *synthetic graphs* using TrillionG [2]. Specifically:
  - Erdos-Renyi (*uniform* vertex degree distribution)
  - Skewed (*skewed* vertex degree distribution)

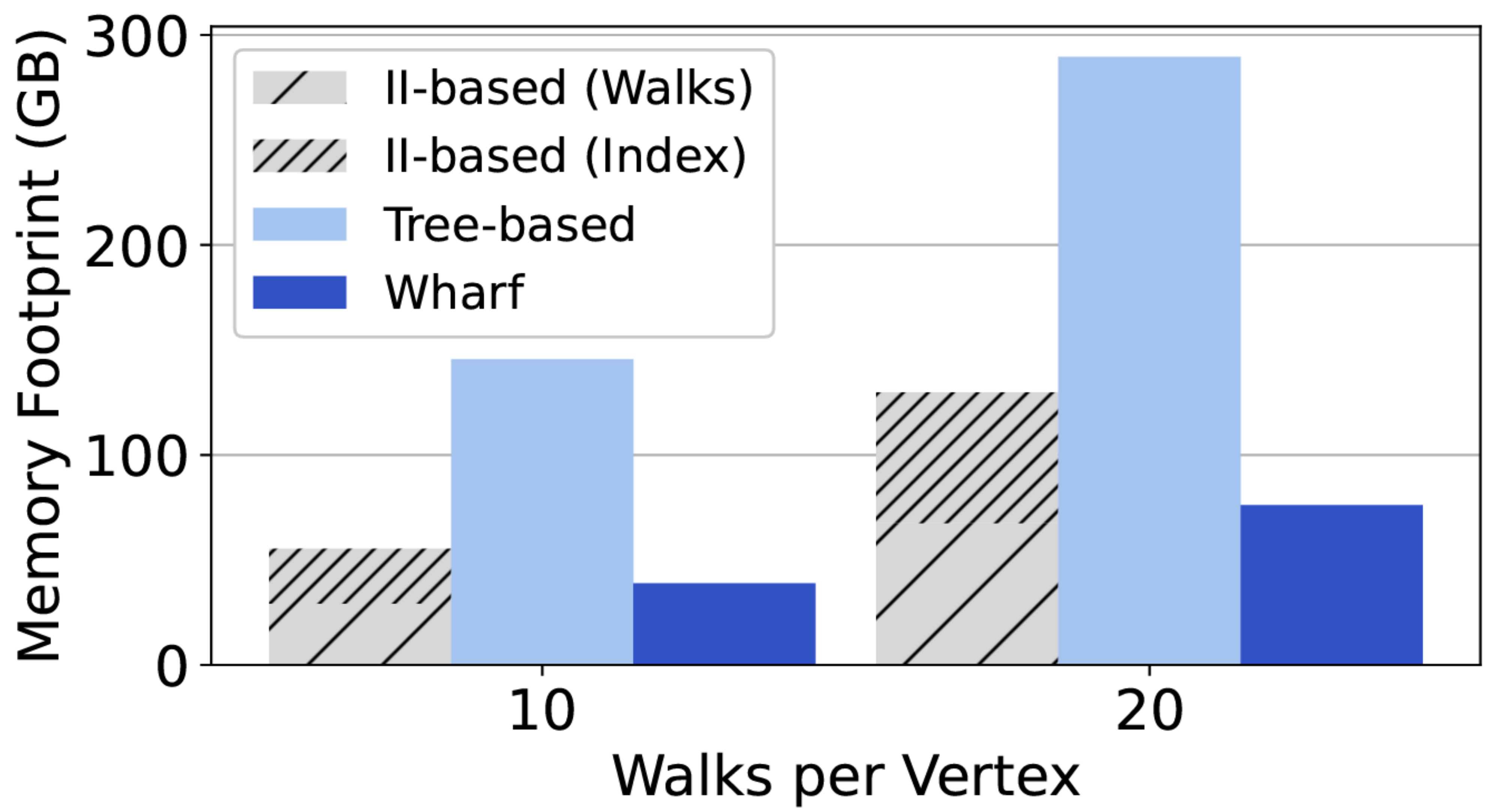
[2] Himchan Park, and Min-Soo Kim. TrillionG: A trillion-scale synthetic graph generator using a recursive vector model.SIGMOD'17

Exp. Study: Mem. Footprint varying n_w and l

Task: Random Walk Corpus Space Consumption



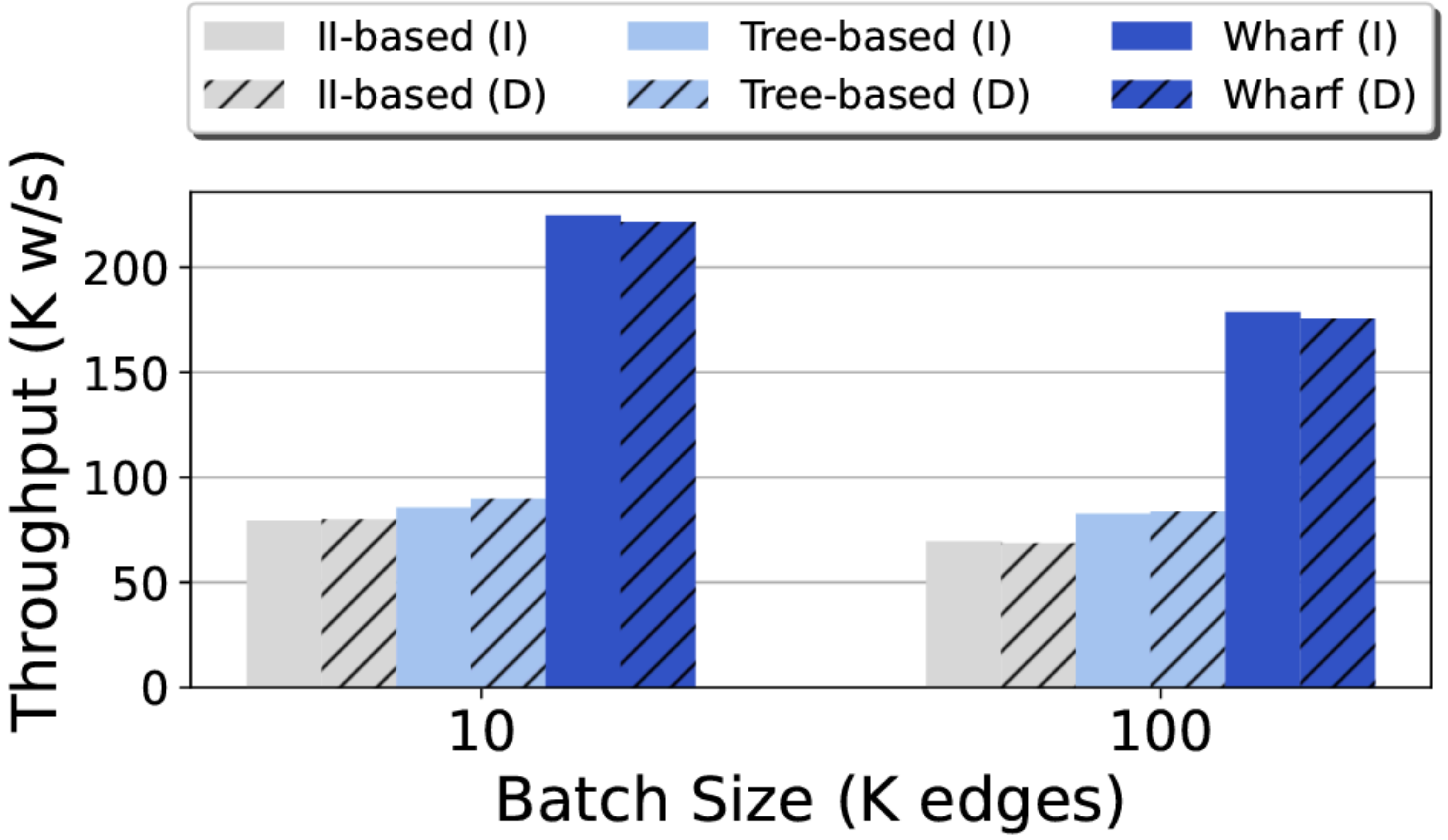
(b) *LiveJournal*, varying l, $n_w = 10$



(c) *LiveJournal*, varying n_w, $l = 80$

Experimental Study: Mixed Workload

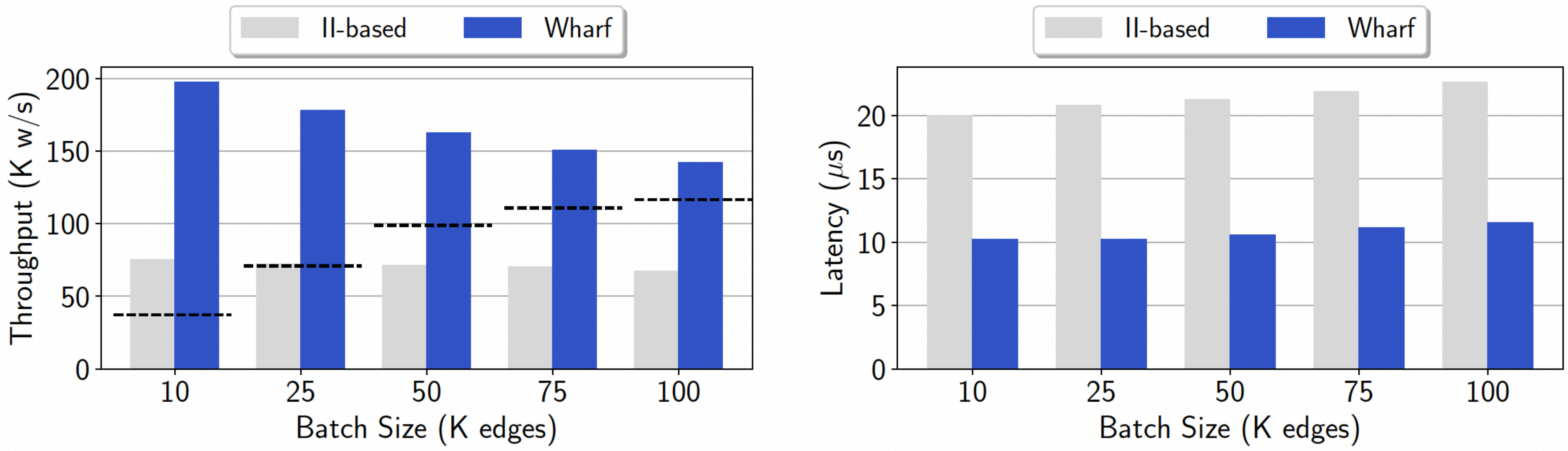
Task: Workload that contains alternate batches of edge insertions and deletions



(a) *LiveJournal*

Experimental Study: Scalability 1

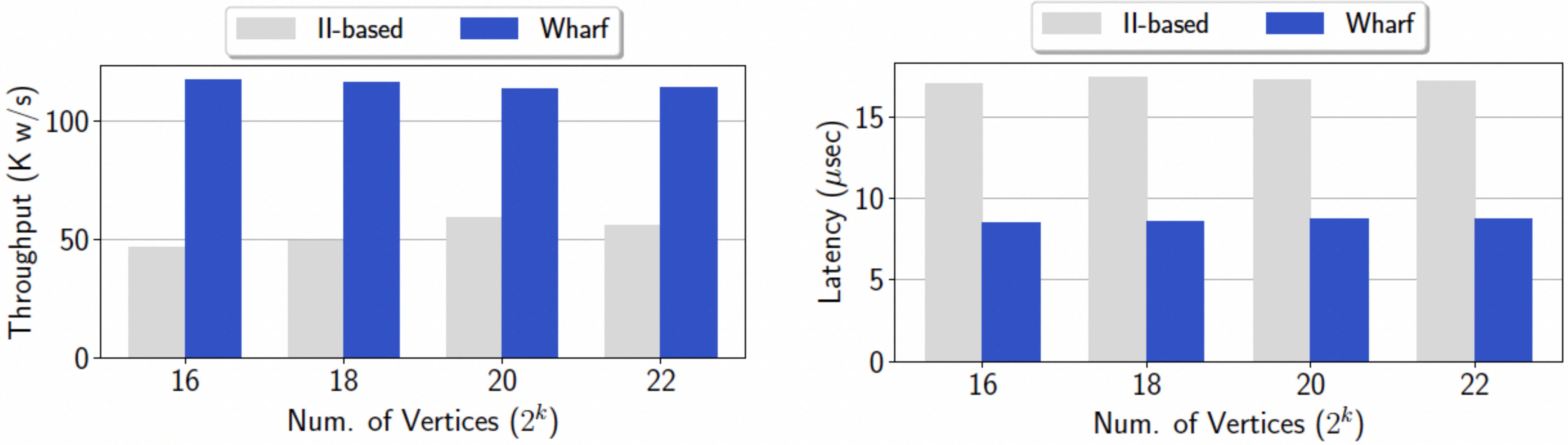
Task: Random Walk Corpus Update on *Orkut* w.r.t. batch size



(a) Throughput (b) Latency

Experimental Study: Scalability 2

Task: Random Walk Corpus Update on *Erdos Renyi* synthetic graphs

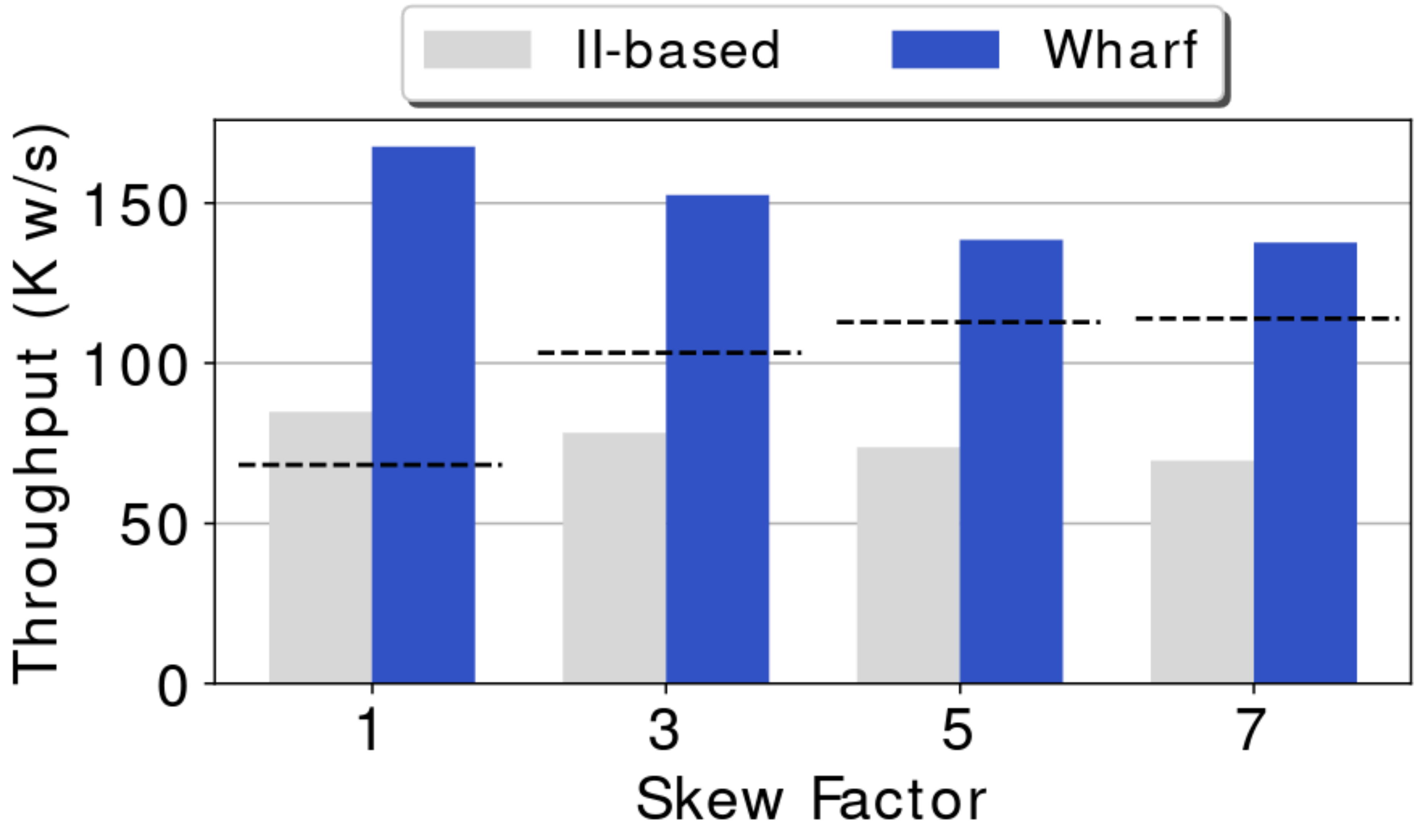


(a) Throughput

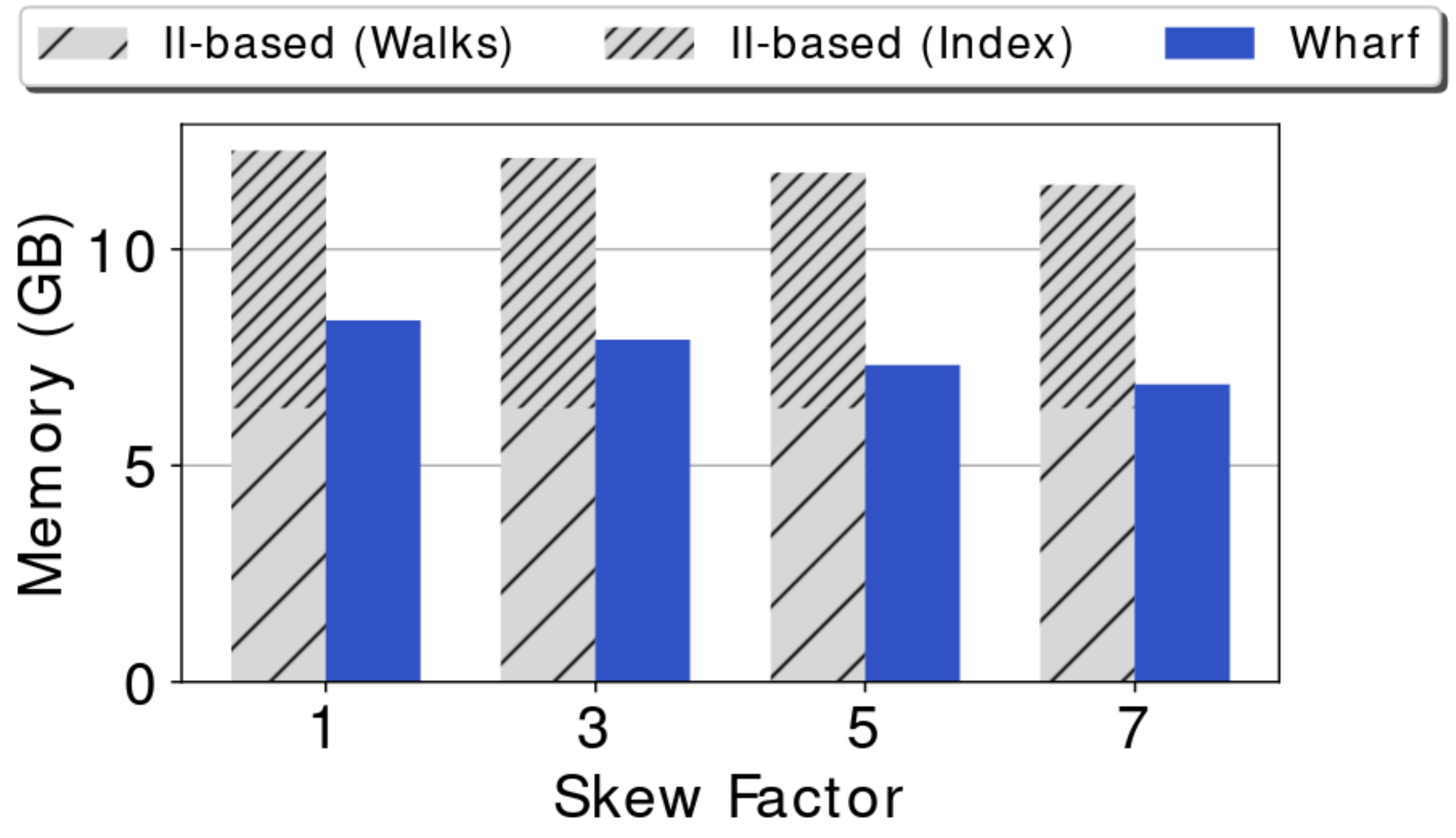
(b) Latency

Experimental Study: Skewness

Task: Random Walk Corpus Update on *skewed* synthetic graphs



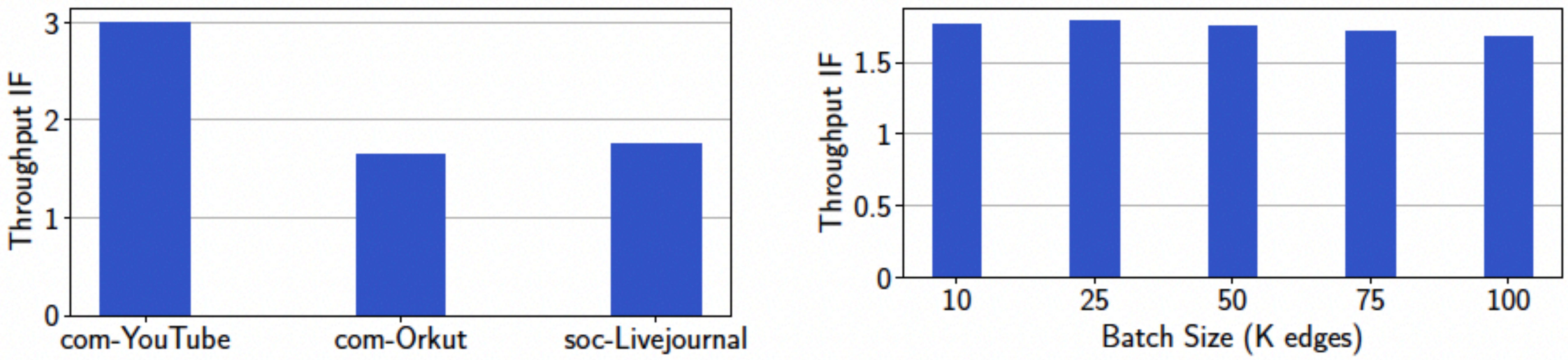
(a) Throughput



(b) Memory footprint

Experimental Study: Optimised Search

Task: Ablation Study on optimised search when updating random walk corpuses



(a) Real graphs, ins. $10K$ edges (b) *Livejournal*, vary batch size