PIE: Enabling Fast and Scalable Incremental Evolving Graph Analytics on Persistent Memory

Yunmo Zhang, Jiacheng Huang, Xizhe Yin[▲],
Junqiao Qiu, Hong Xu[◆], Chun Jason Xue[✦]

香港城市大學
City University of Hong Kong

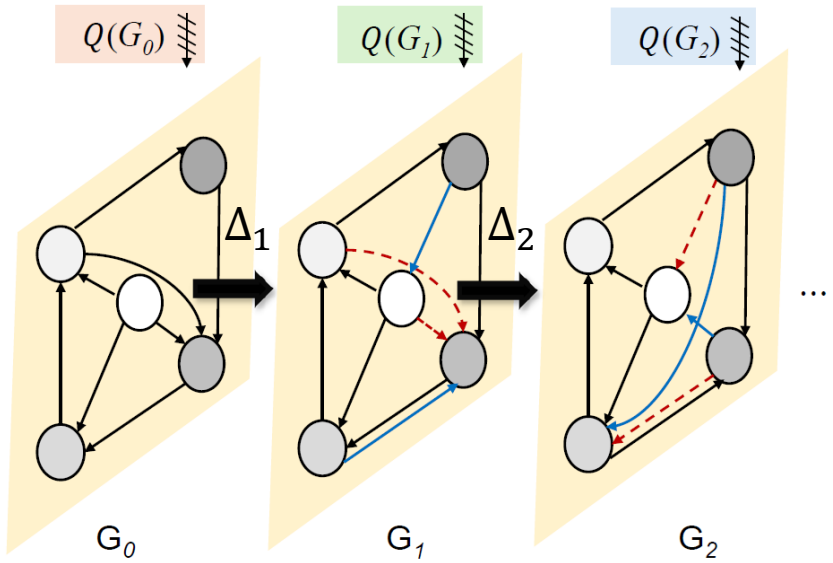
UC RIVERSIDE

香港中文大學
The Chinese University of Hong Kong

MOHAMED BIN ZAYED
UNIVERSITY OF
ARTIFICIAL INTELLIGENCE

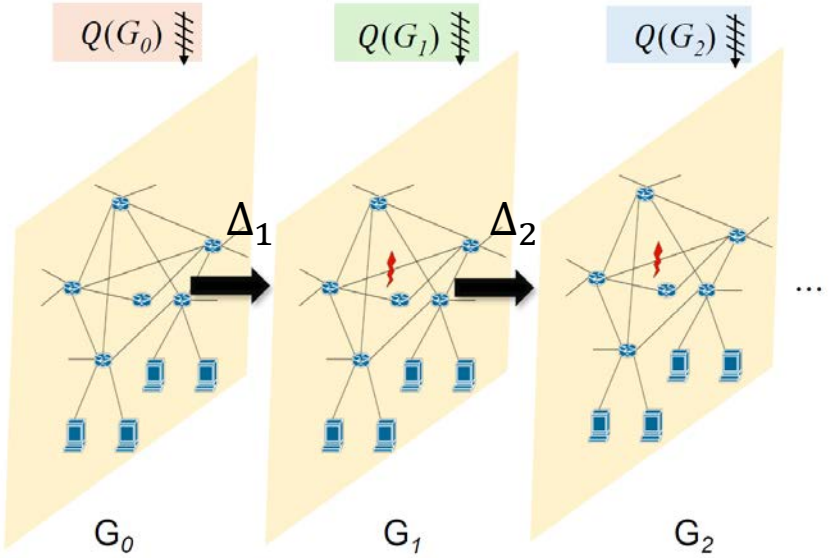
Evolving Graph Analytics

- Real-world graphs evolve continuously over time
  - With $\Delta_i = \{\Delta_i^+, \Delta_i^-\}$
- Track a graph property by evaluating the query on ***a sequence of snapshots*** within a time window
- Have many applications
  - In social network, fraud detection, bioinformatics, network management, etc.



Evolving Graph Analytics

- Real-world graphs evolve continuously over time
  - With $\Delta_i = \{\Delta_i^+, \Delta_i^-\}$
- Track a graph property by evaluating the query on ***a sequence of snapshots*** within a time window
- Have many applications
  - In social network, fraud detection, bioinformatics, network management, etc.

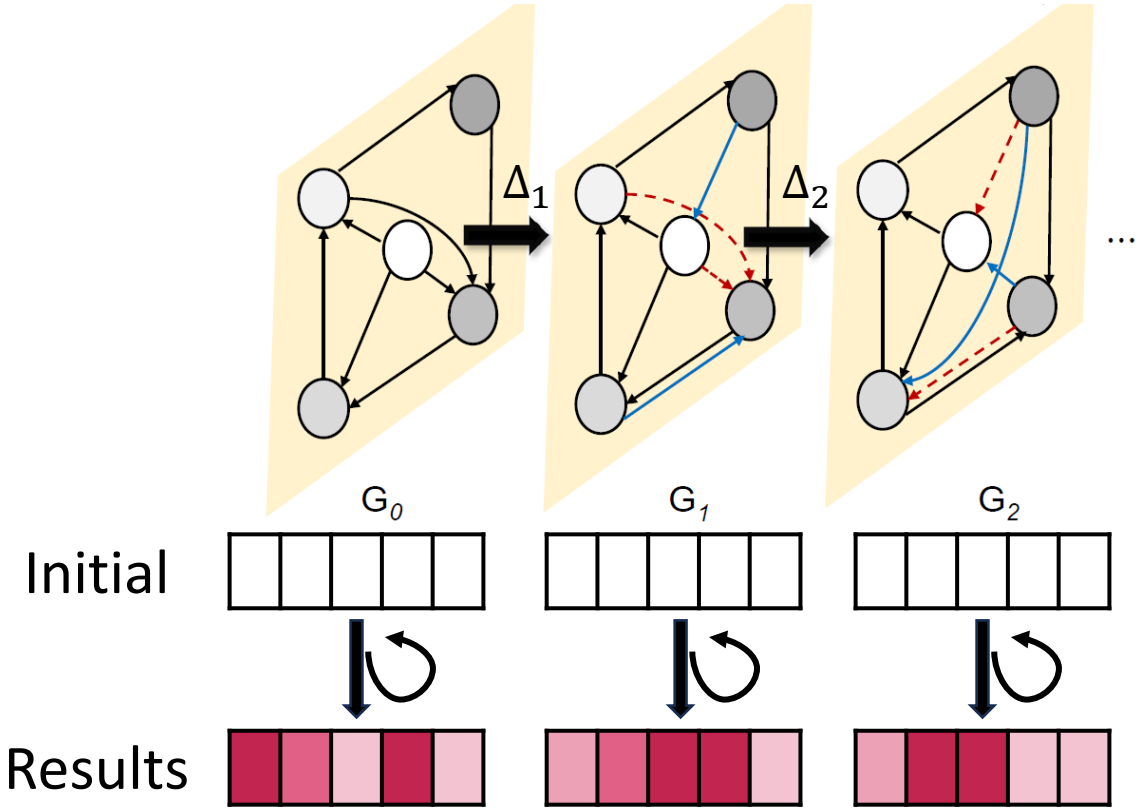


Network Administrator

For which period(s) is *u* unreachable from *v*?

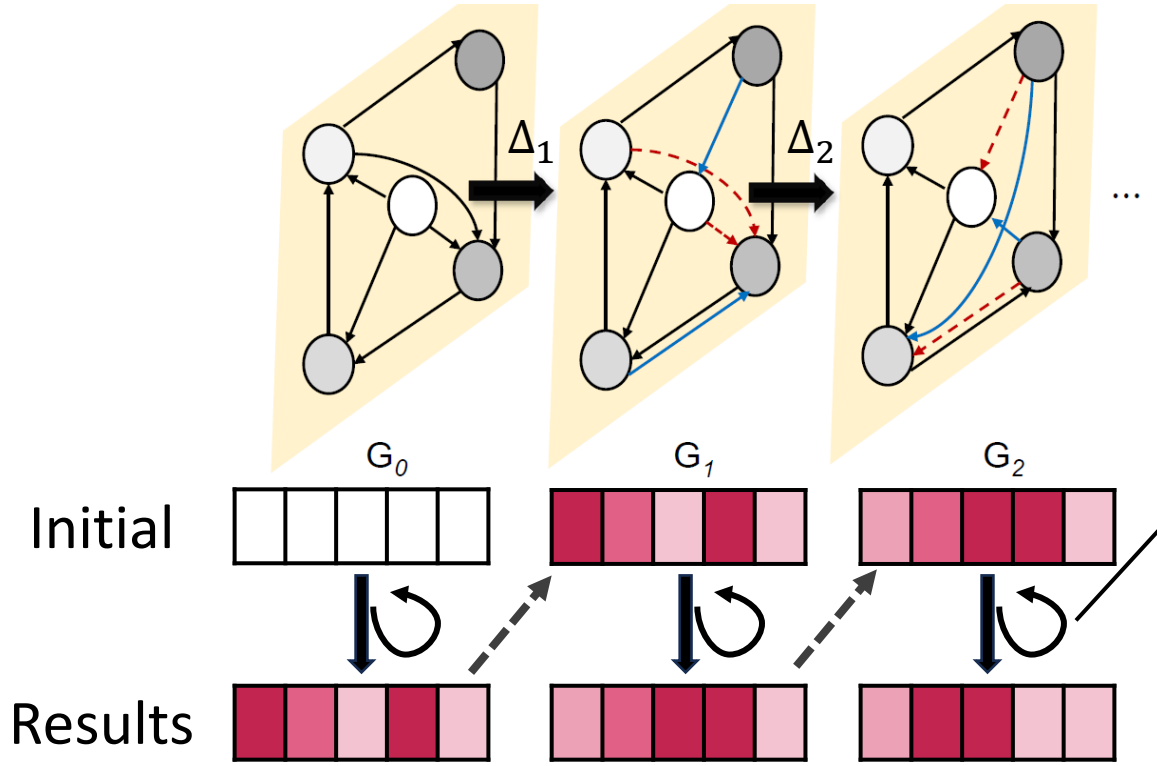
Evolving Graph Analytics Approaches

- A naïve method: *re-computing*
  - Full evaluation of each snapshot from scratch



Evolving Graph Analytics Approaches

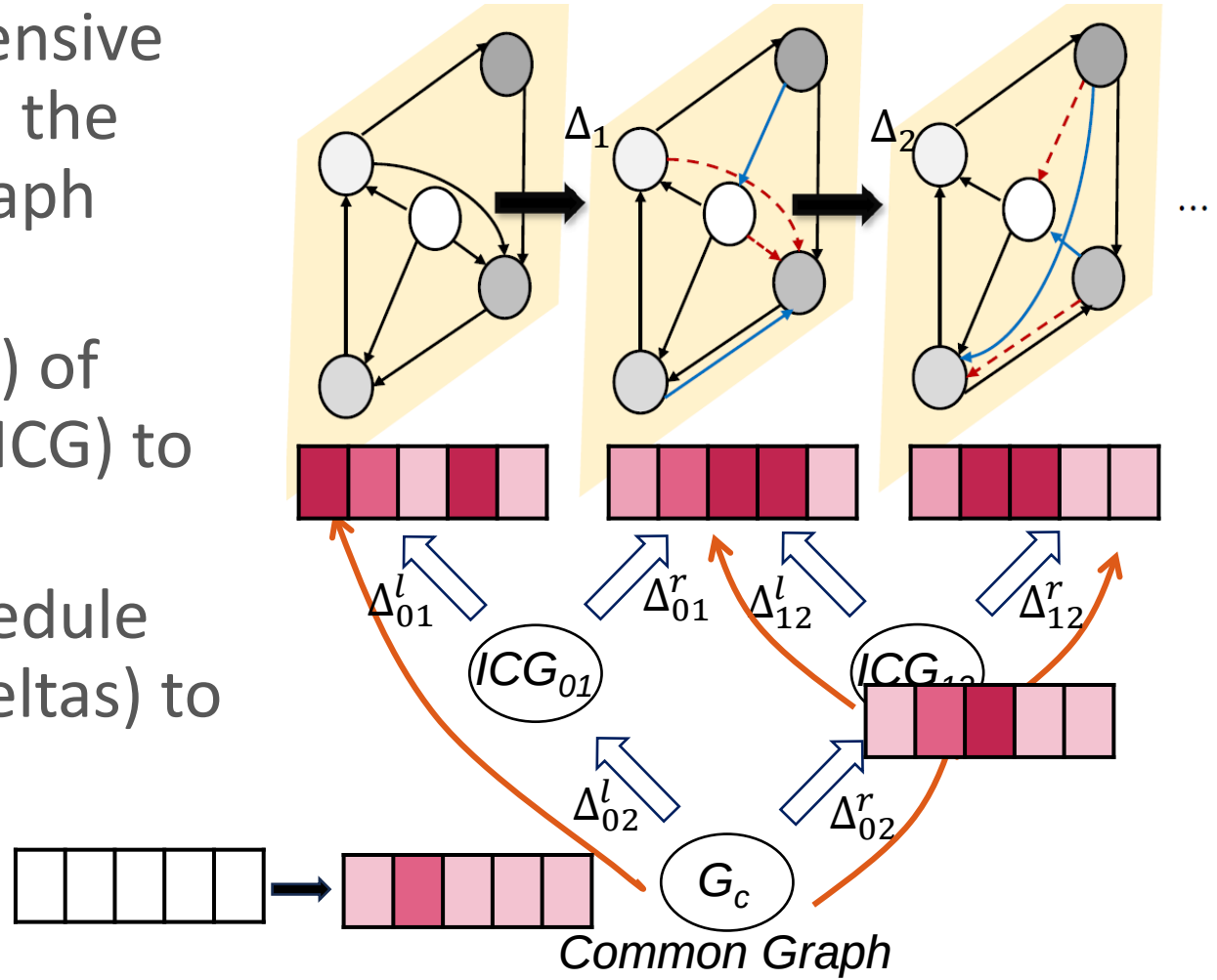
- **Incremental** Analysis
  - Reuse results from the previous snapshot.
  - Naiad [SOSP'13], Tornado [SIGMOD'16], Kickstarter [ASPLOS'17]



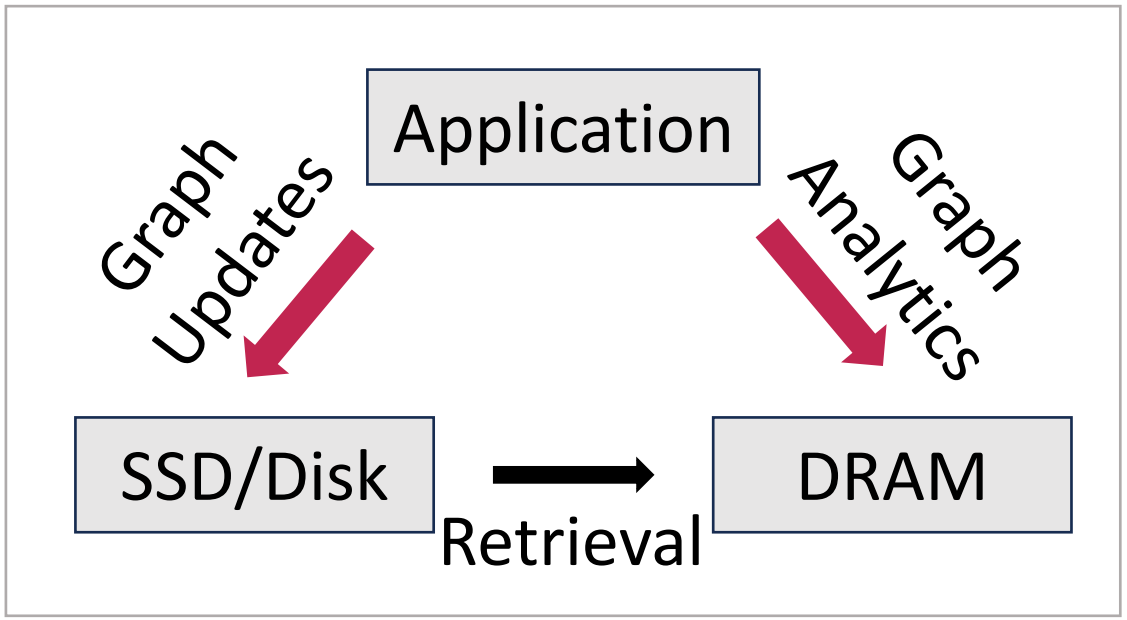
Step 1: Identify effects of Δ^{-}.
Step 2: Process Δ^{+} and re-converge on the new snapshot.

Evolving Graph Analytics Approaches

- **Deletion-free** Incremental Analysis: CommonGraph [ASPLOS'23]
  - Avoid processing the expensive deletions by starting from the results of the common graph across snapshots (G_c).
  - Build the triangle grid (TG) of interm. common graphs (ICG) to further shared results.
  - Follow the computed schedule (path along the interm. deltas) to maximize sharing.

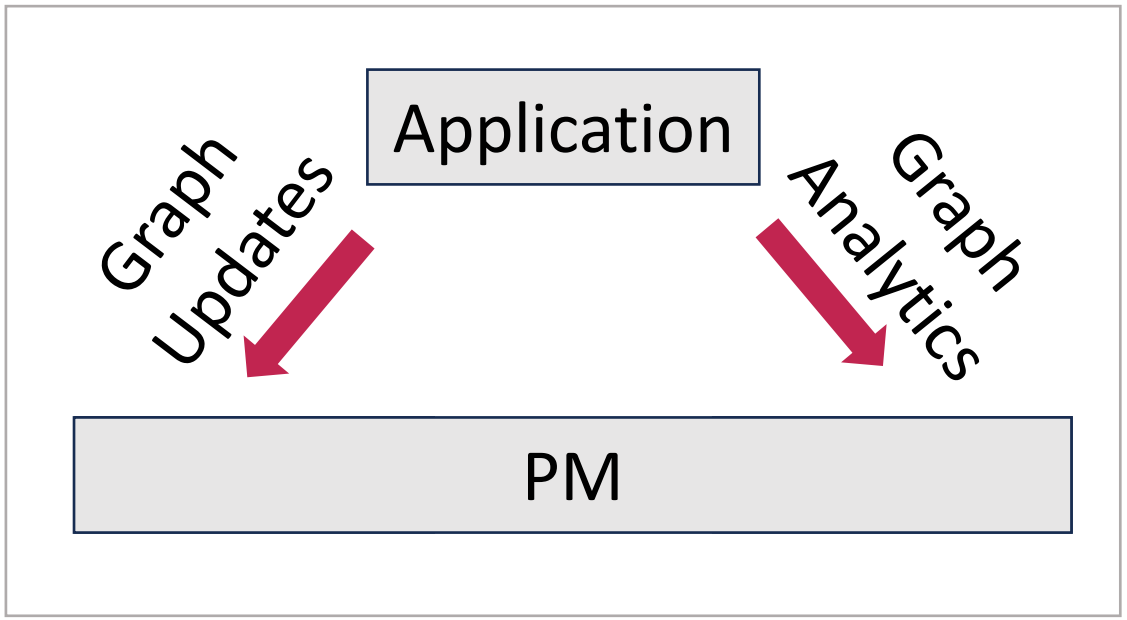


Large-scale Evolving Graph Systems



Traditional Evolving Graph System

Large-scale Evolving Graph Systems

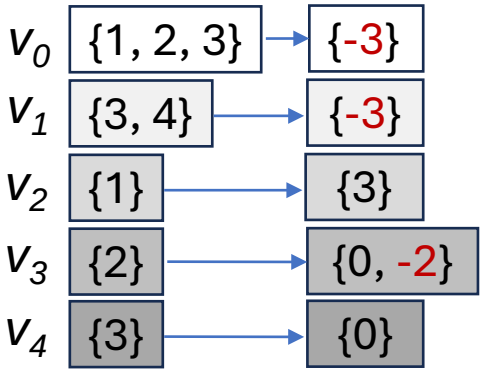


PM-based Evolving Graph System

- ✓ Data Persistence
- ✓ Low latency, high density
- ✓ Byte Addressable

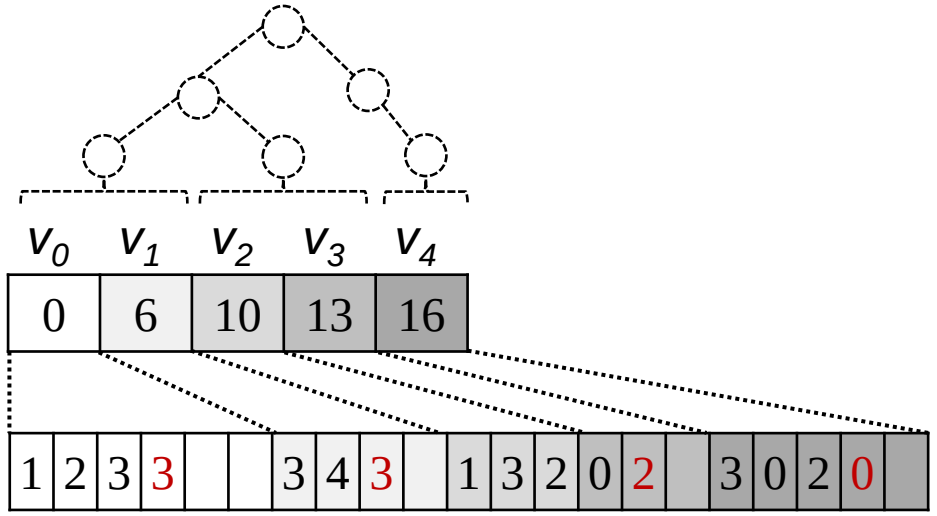
Existing PM-based Evolving Graph Systems

- Have explored porting different graph formats to PM



Adjacency Lists

XPGraph [MICRO '22]



PMA (Mutable CSR)

DGAP [SC '23]

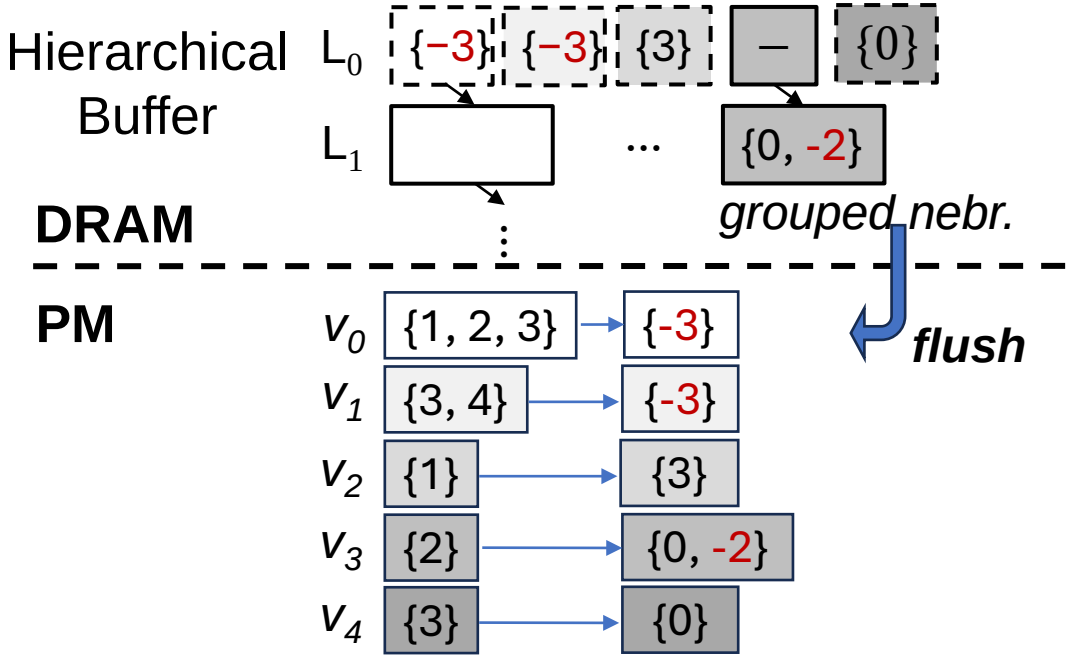
PM Performance Characteristics:

1. Performance gap with DRAM

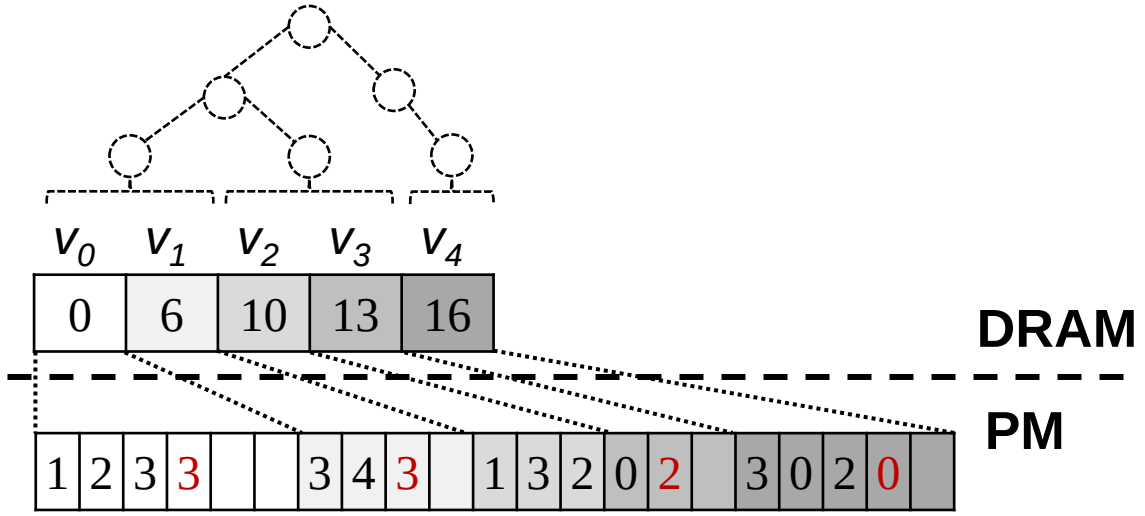
(2~3× lower max. READ bandwidth and 7~8× lower max. WRITE bandwidth)

2. Bandwidth drop for small random accesses

(e.g., <256 bytes in Optane PM)



Adjacency Lists



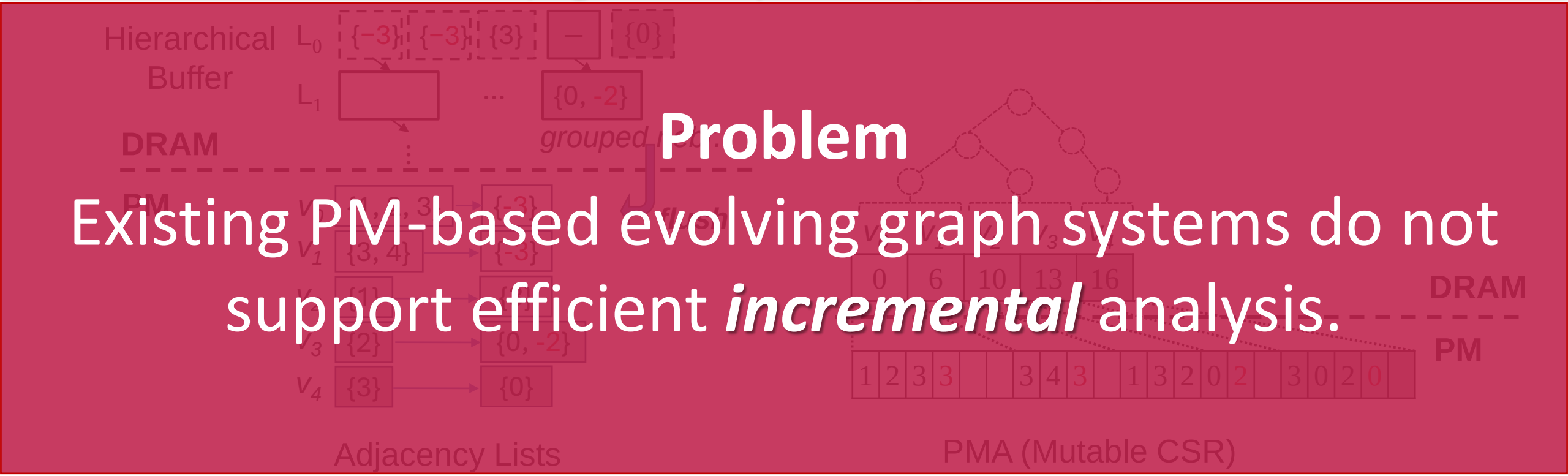
PMA (Mutable CSR)

XPGraph [MICRO '22]

DGAP [SC '23]

PM Performance Characteristics:

1. **Performance gap with DRAM**
   (2~3× lower max. READ bandwidth and 7~8× lower max. WRITE bandwidth)
2. **Bandwidth drop for small random accesses**
   (e.g., <256 bytes in Optane PM)



Problem

Existing PM-based evolving graph systems do not support efficient ***incremental*** analysis.

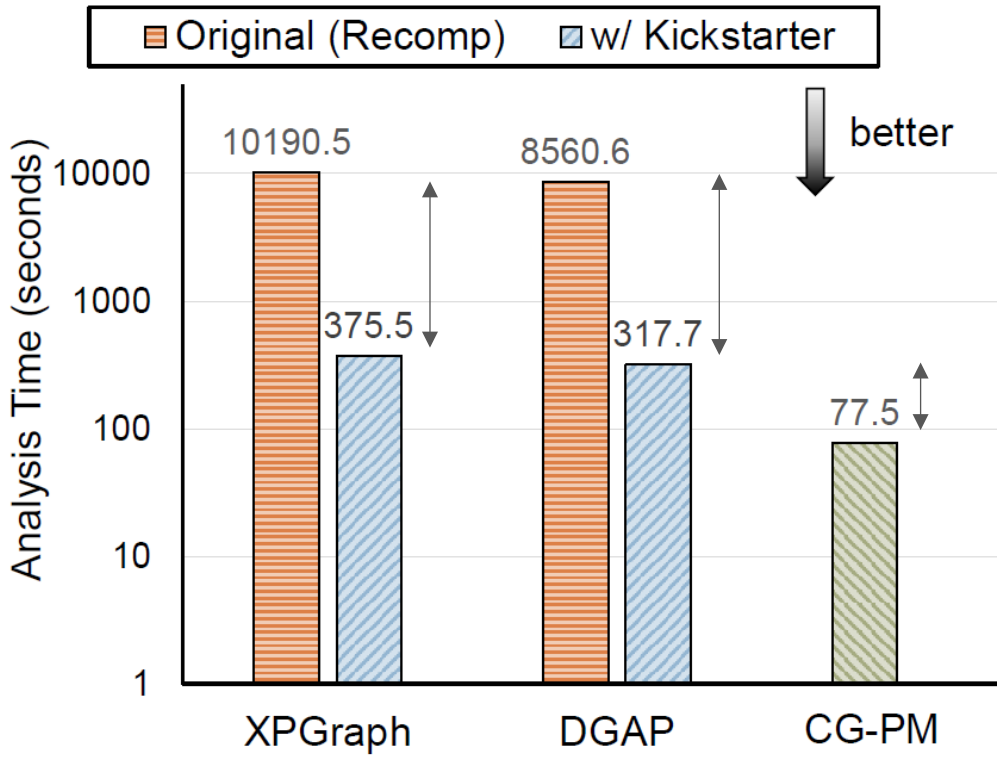
XPGraph [MICRO '22]

DGAP [SC '23]

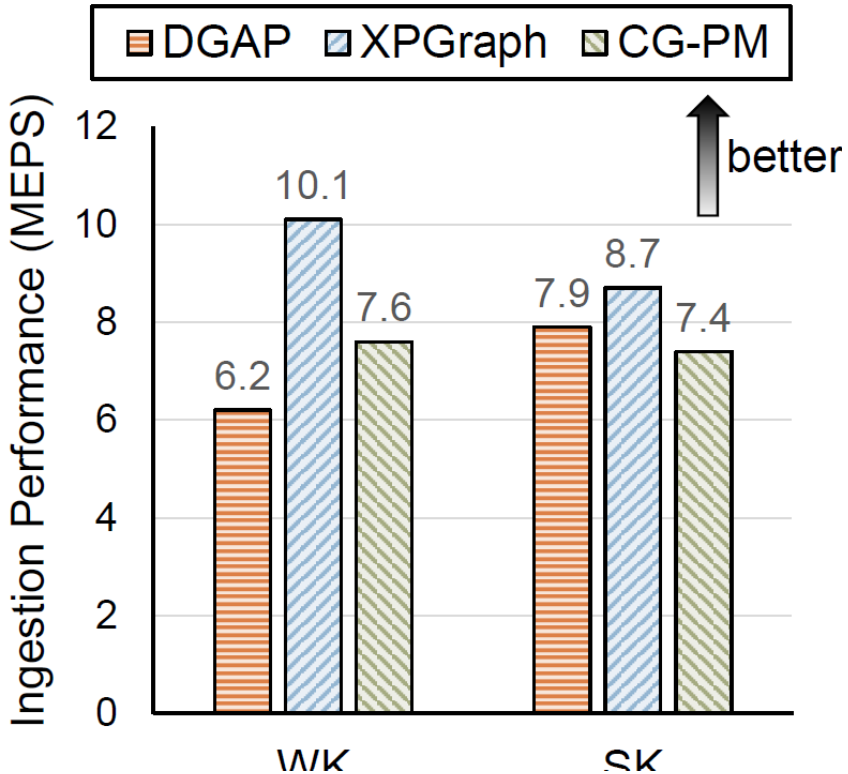
Motivation

- Integrat **Kickstarter** into SOTA PM-based graph systems.
- Port **CommonGraph** *directly* to PM (CG-PM)

(a) Analytics

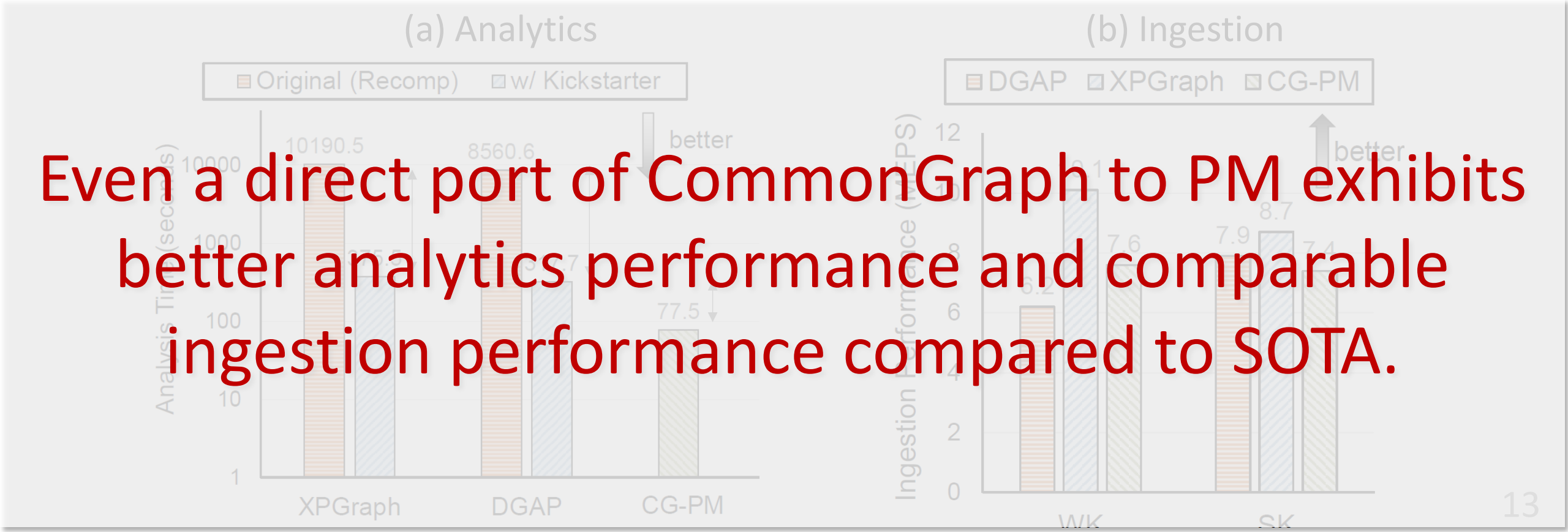


(b) Ingestion



Motivation

- Integrat **Kickstarter** into SOTA PM-based graph systems.
- Port **CommonGraph** *directly* to PM (CG-PM)



Even a direct port of CommonGraph to PM exhibits better analytics performance and comparable ingestion performance compared to SOTA.

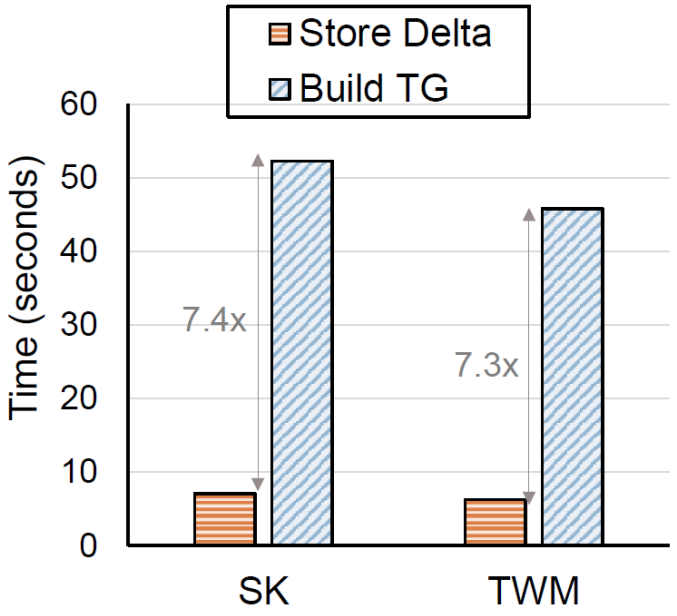
Motivation

- Integrat **Kickstarter** into SOTA PM-based graph systems.
- Port **CommonGraph** *directly* to PM (CG-PM)

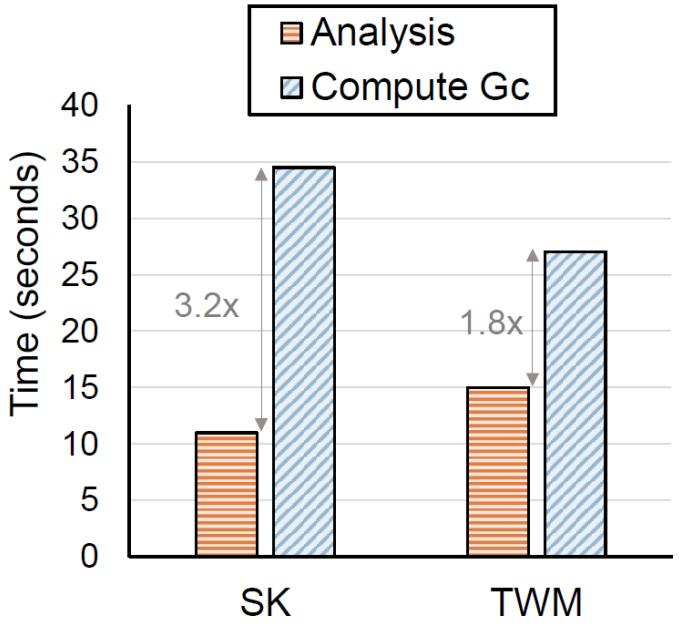
However, directly porting CommonGraph to PM suffers from **high read/write amplifications and extra computations**, making it far from optimal.

Issues of CommonGraph on PM

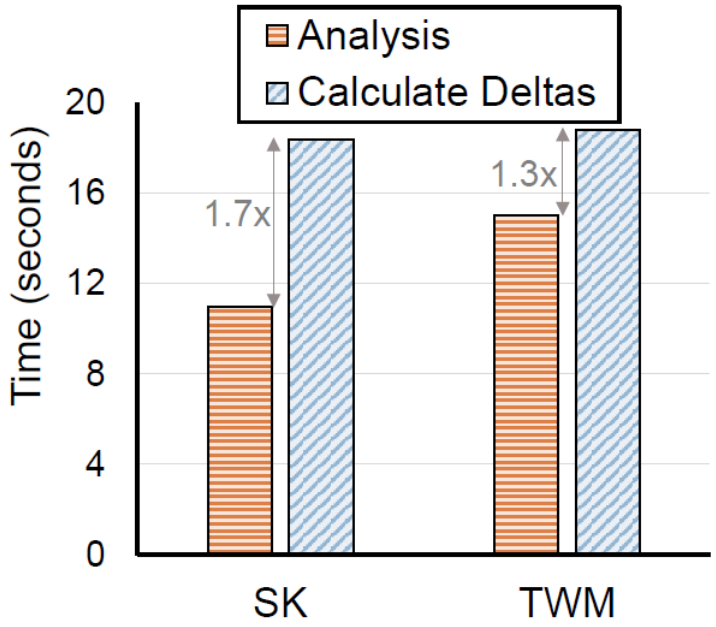
Issue 1: High Write Amplification



Issue 2: High Read Amplification

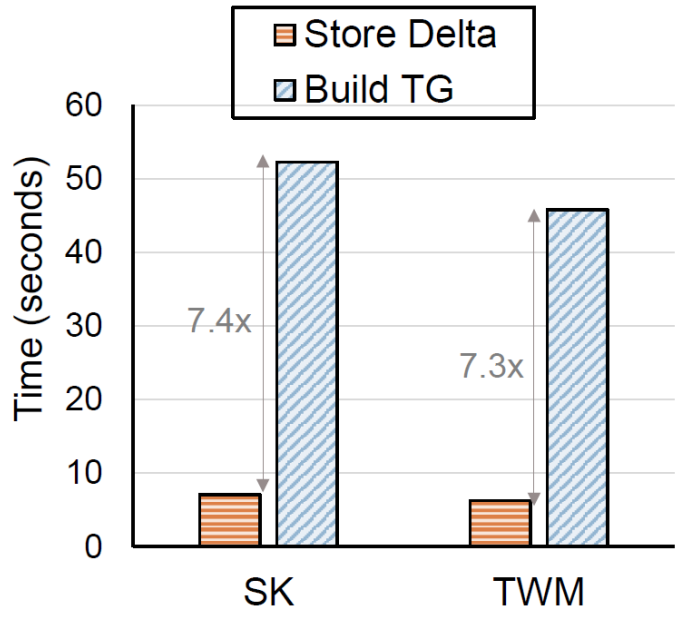


Issue 3: Extra Computations

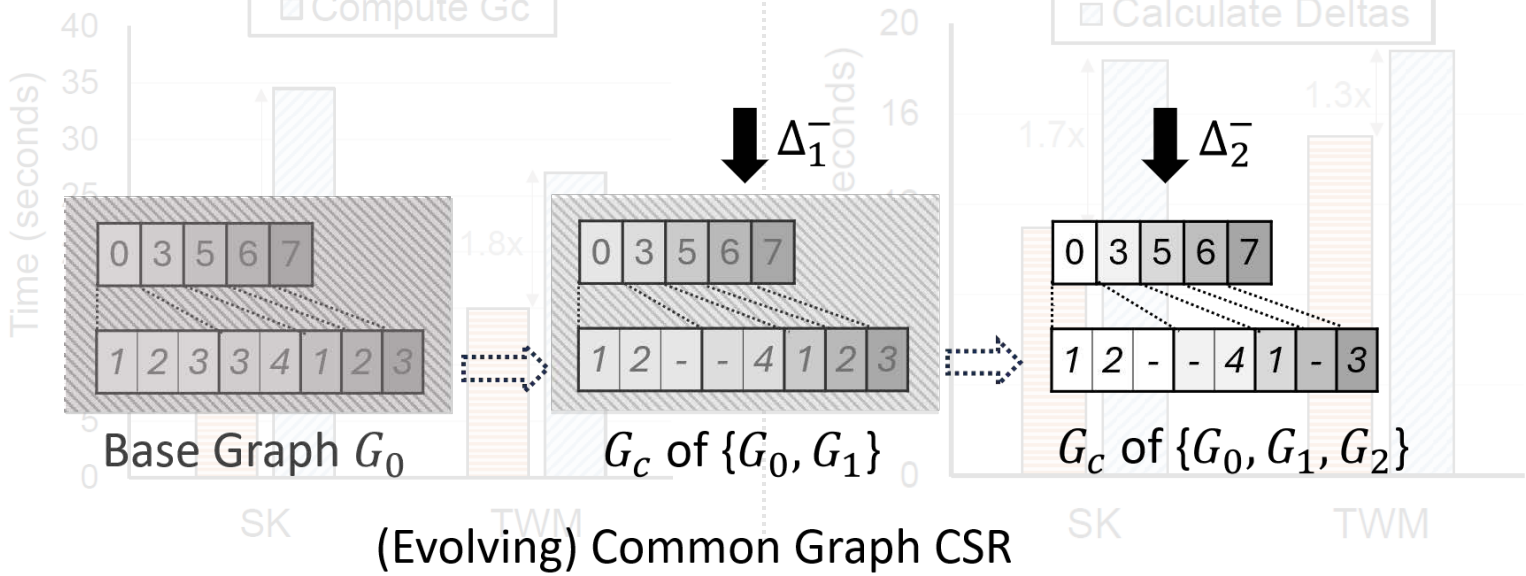


Issues of CommonGraph on PM

Issue 1: High Write Amplification



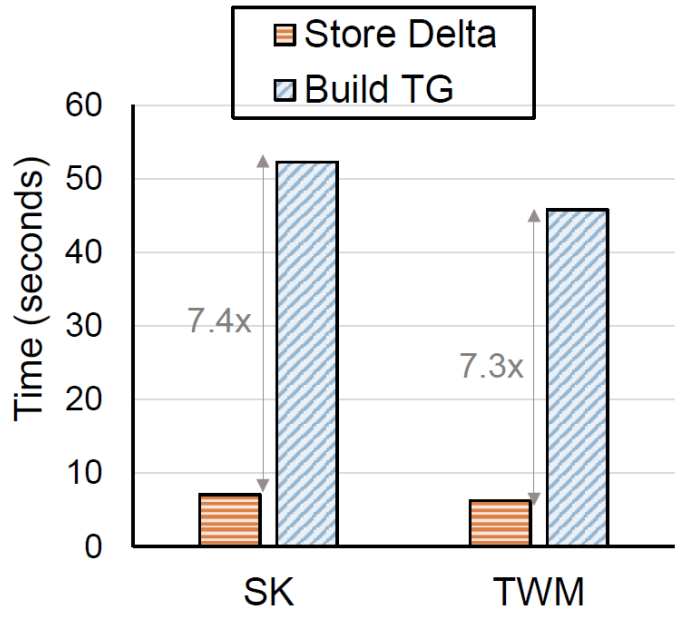
- Maintaining the common graph upon ingestion requires updating the base graph CSR stored in PM.
- It causes small random writes.



(Evolving) Common Graph CSR

Issues of CommonGraph on PM

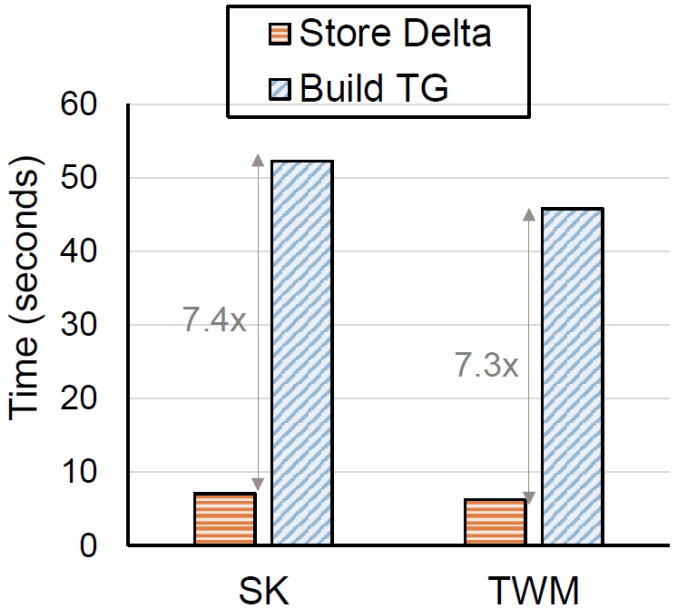
Issue 1: High Write Amplification



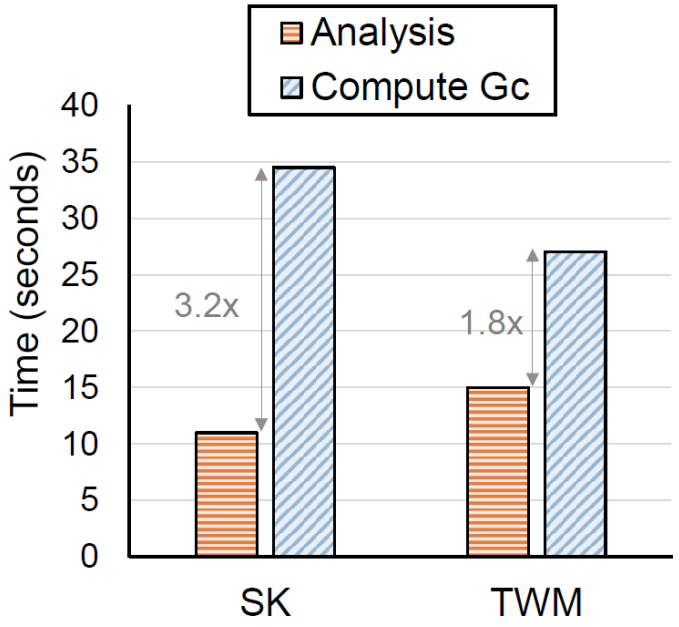
- Maintaining the common graph upon ingestion requires updating the base graph CSR stored in PM.
- It causes small random writes.
- Intermediate deltas $\Delta_{ij}^{l/r}$ are required for constructing TG (including snapshots), resulting in additional PM writes.

Issues of CommonGraph on PM

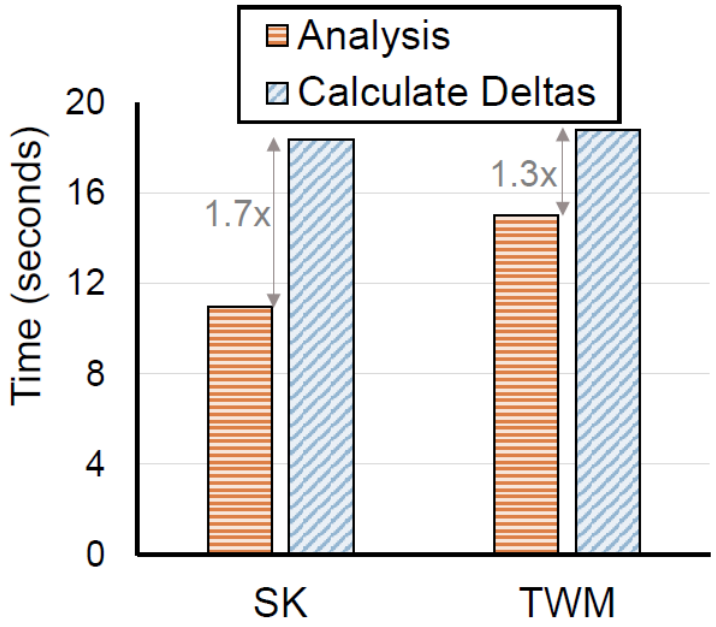
Issue 1: High Write Amplification



Issue 2: High Read Amplification

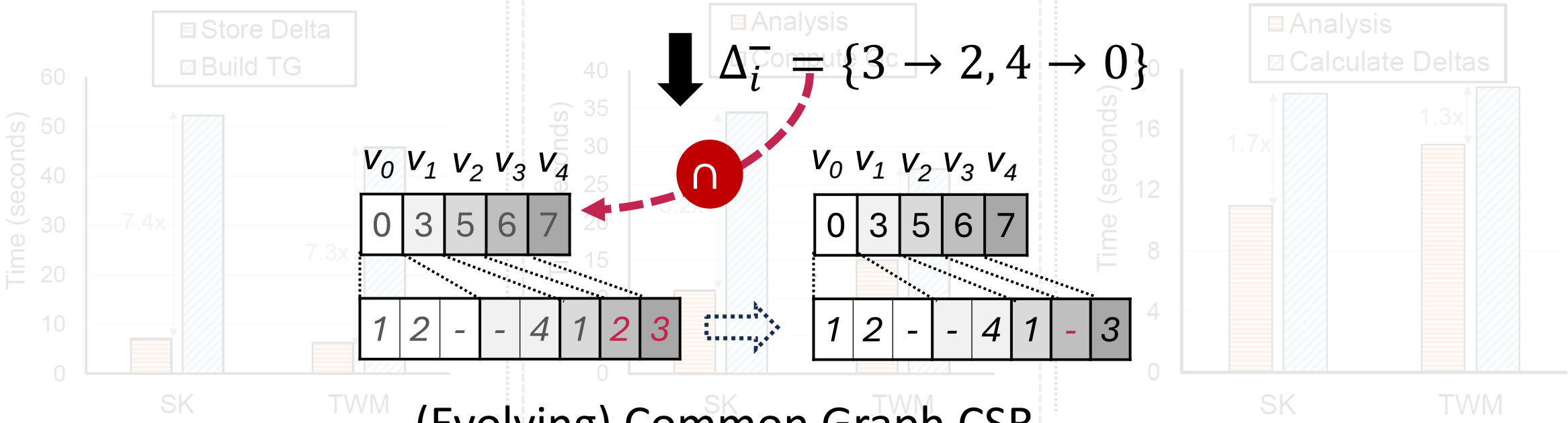


Issue 3: Extra Computations



Issues of CommonGraph on PM

- Calculating G_c requires expensive set intersection operations.
- It causes additional reads from PM, which are random reads when using binary search.



(Evolving) Common Graph CSR

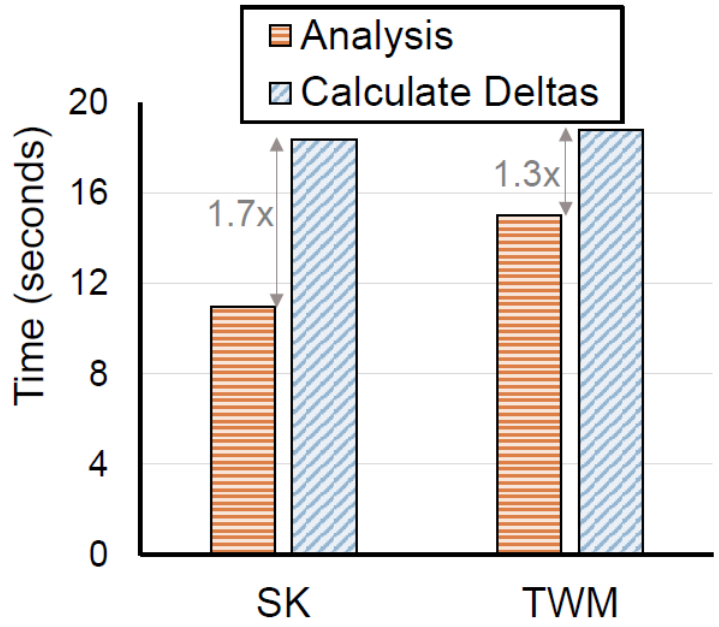
Issues of CommonGraph on PM

Issue 1: High Write Amplification

Issue 2: High Read Amplification

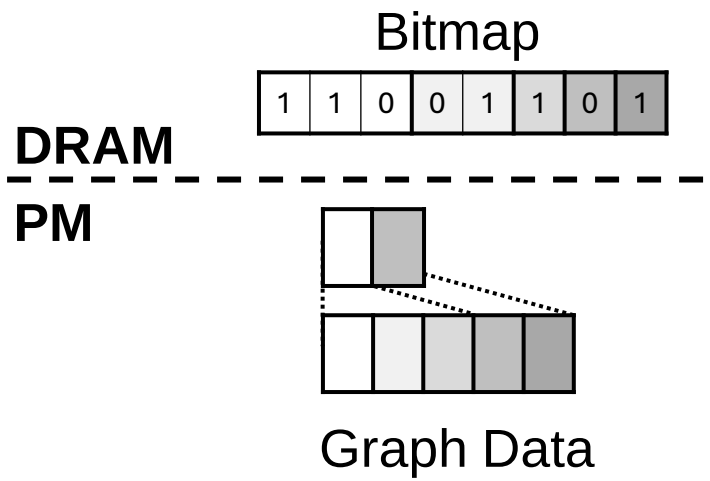
- Computing the optimal schedule requires building a complete TG.
- This involves $N(N-1)$ set intersections to calculate the IR deltas, for snapshots $\{G_0, \ldots, G_N\}$.

Issue 3: Extra Computations

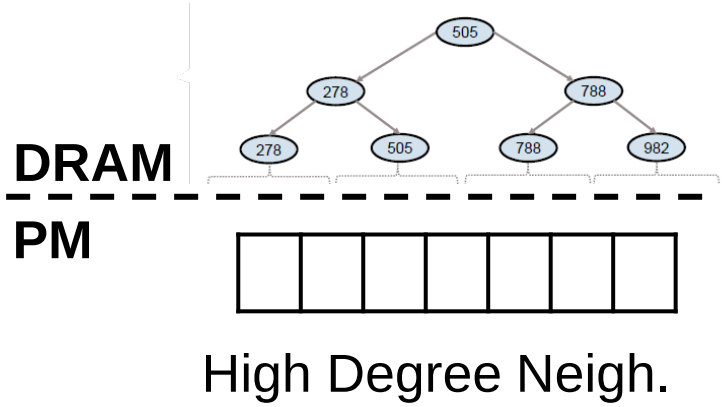


Solutions

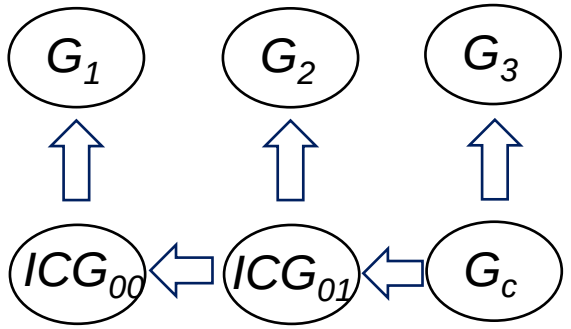
Issue 1: High Write Amplification



Issue 2: High Read Amplification



Issue 3: Extra Computations



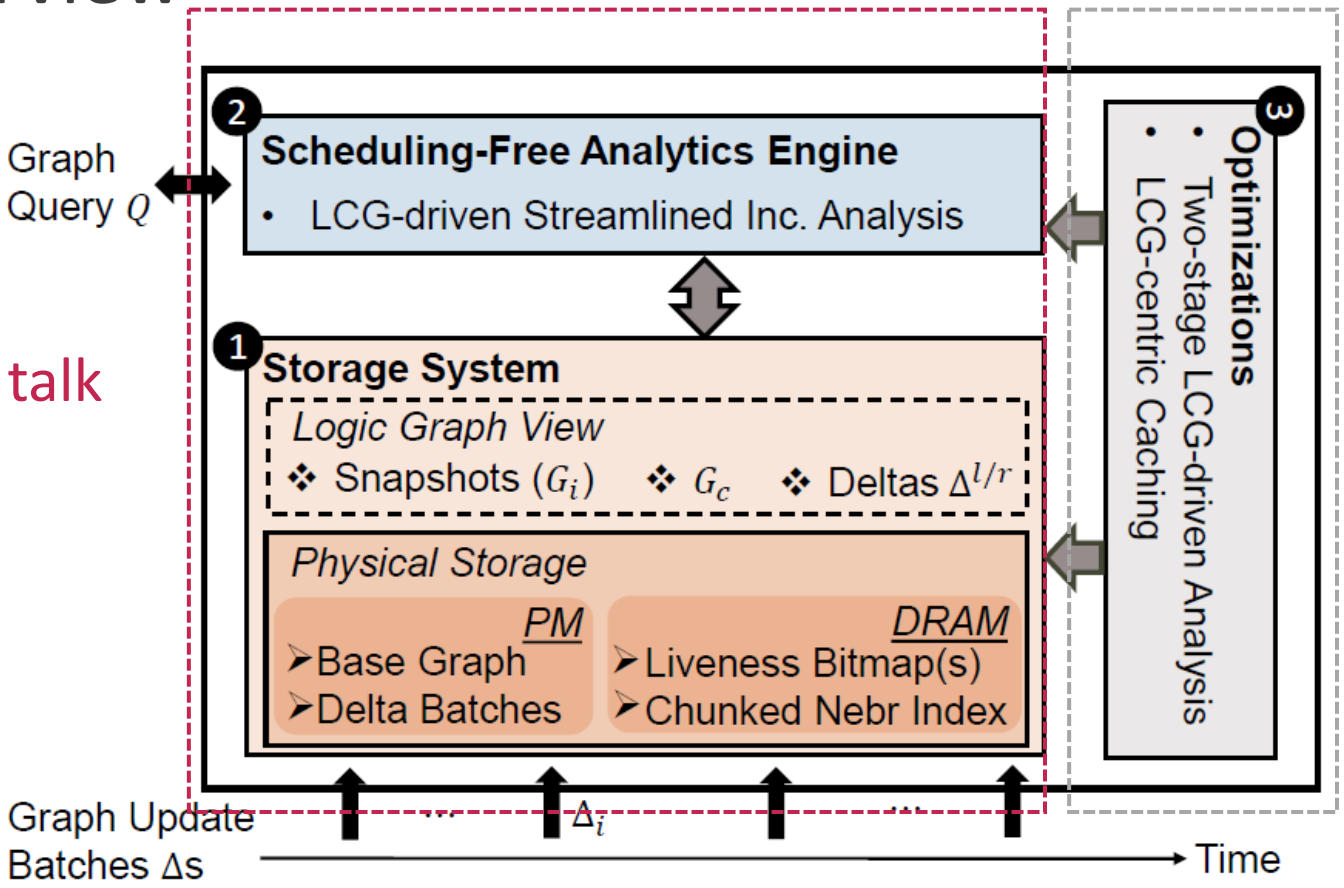
Solution 1: Detached Logical Graph View

Solution 2: Chunked Neighbor Index

Solution 3: Streamlined Incremental Analysis

PIE: PM-based Evolving Graph System

- An evolving graph storage and analytics system
- System Overview

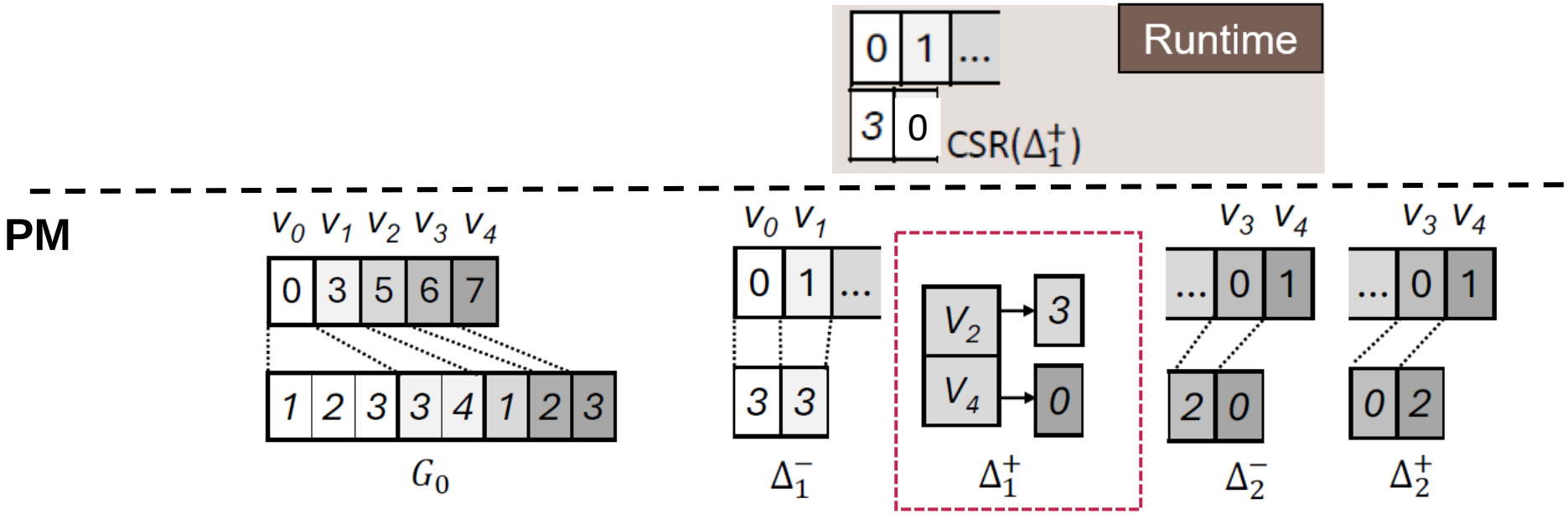


This talk

Refer to the paper

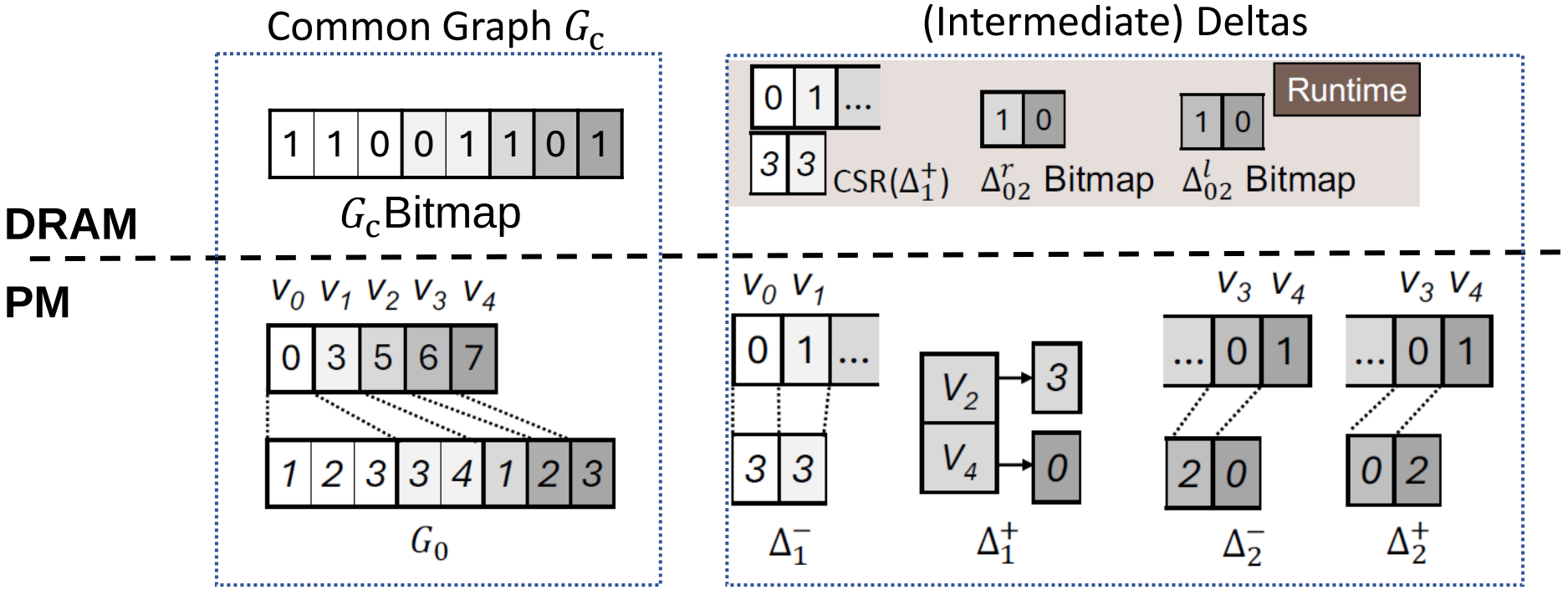
PIE: Storage System

- Graph data are totally stored in PM.
- Hybrid delta format
  - Deltas are represented by CSRs in most cases.
  - Deltas are stored as edge list in PM when vertex array >> edge array, its CSR is rebuilt in DRAM at runtime.



PIE: Detached Logical Graph View

- The graph view of G_c and components in TG are *logically* provided to the deletion-free incremental analysis.
  - The logical is separated from the graph storage by bitmaps.
  - It requires NO additional PM writes.



PIE: Detached Logical Graph View

- The logical graph view is feasible due to our observation for the relationship between TG components and graph data.
  - G_c is included by the base graph stored in PM ($G_c = \mathrm{ICG}_{0,N}$)

$$ICG_{0,j} = G_0 - \sum_{k=1}^{j} \Delta_k^-.$$

  - IR deltas are included by the ingested deltas in PM

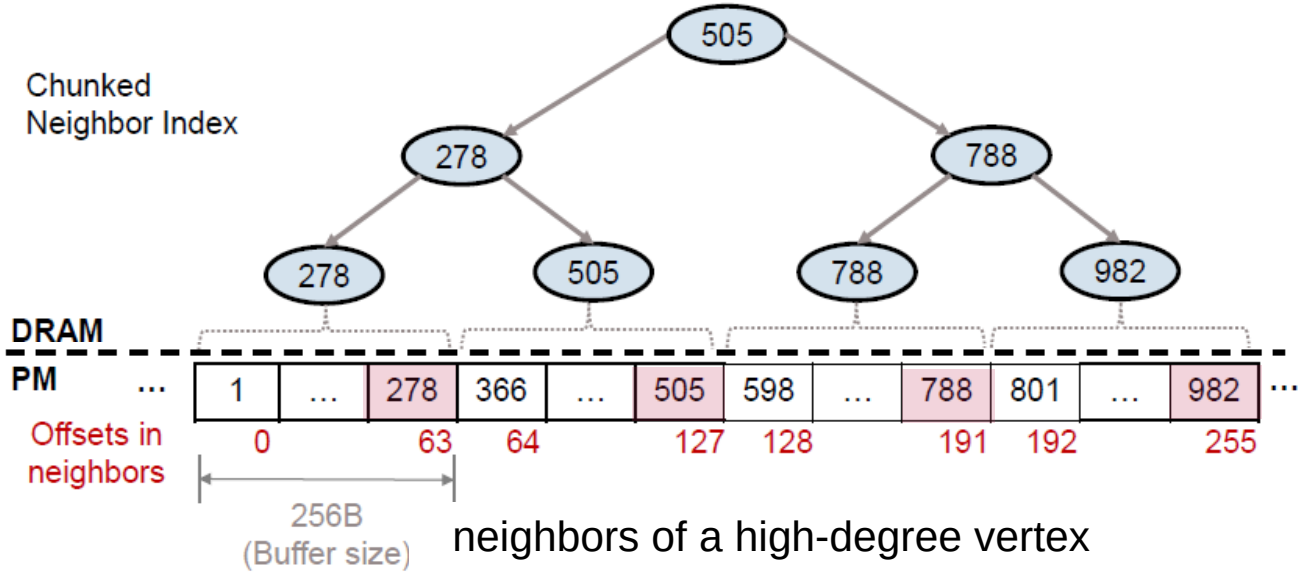
$$\Delta_{ij}^l = \Delta_j^- - \sum_{k=i}^{j-1} \Delta_k^+; \quad \Delta_{i,j}^r = \Delta_i^+ - \sum_{k=i+1}^{j} \Delta_j^-.$$

Graph Data

TG components

PIE: Chunked Neighbor Index

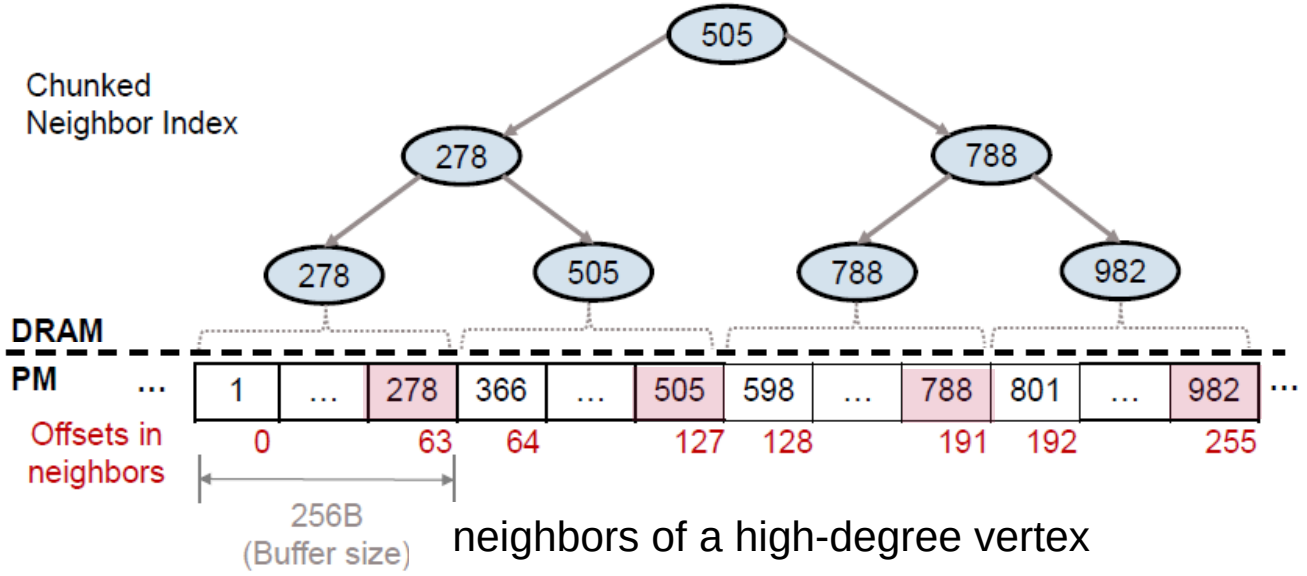
- Chunked neighbor index is built in DRAM for high-degree vertices in G_0.
  - The neighbors are segmented into chunks of the buffer size, e.g., 256B.
  - A binary tree (stored as an array) is built with the pivot value of each chunk as the leaf node to accelerate search.



neighbors of a high-degree vertex

PIE: Chunked Neighbor Index

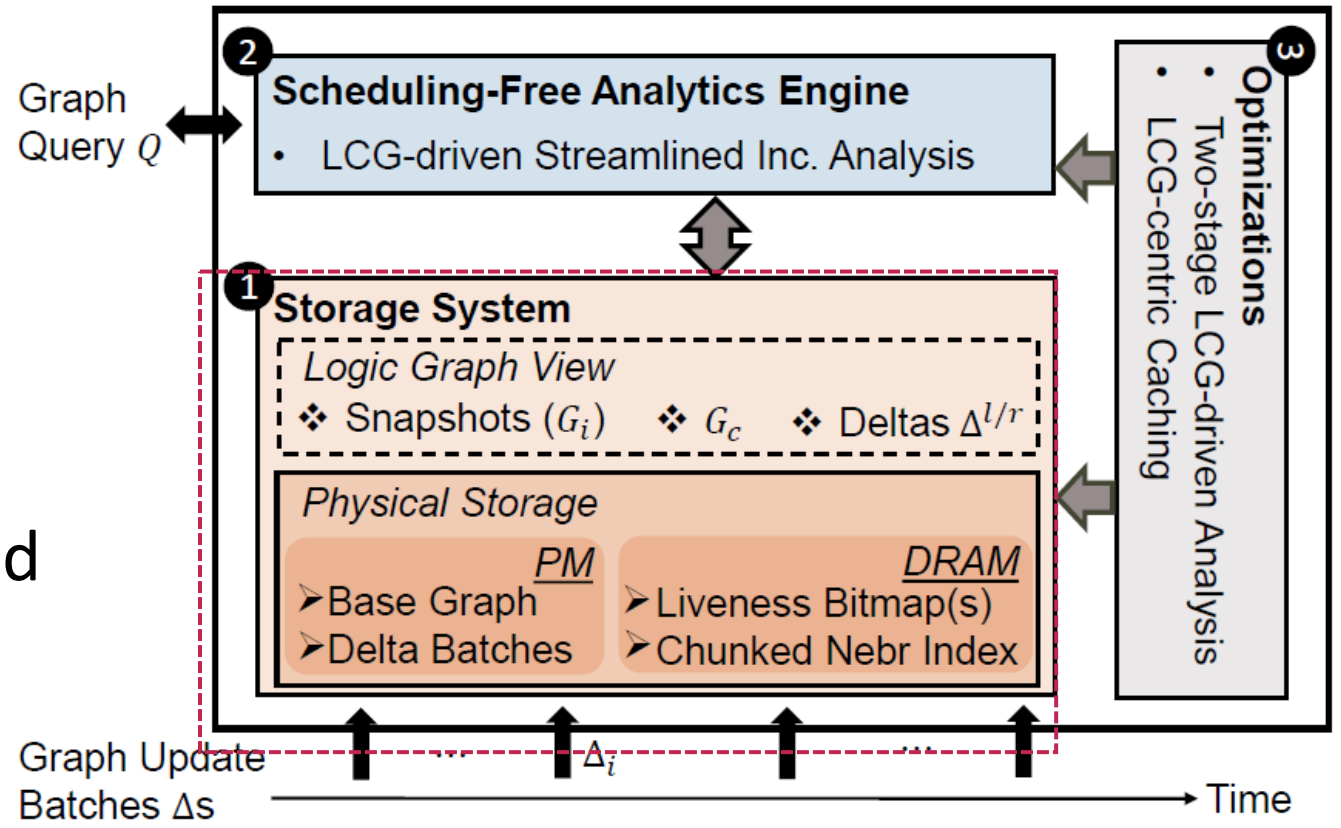
- Chunked neighbor index is built in DRAM for high-degree vertices in G_0.
  - A leaf node position in the array directly indicates the position of the chunk containing the target of a search.
  - A search on a neighbor in G_c requires one PM media access.



neighbors of a high-degree vertex

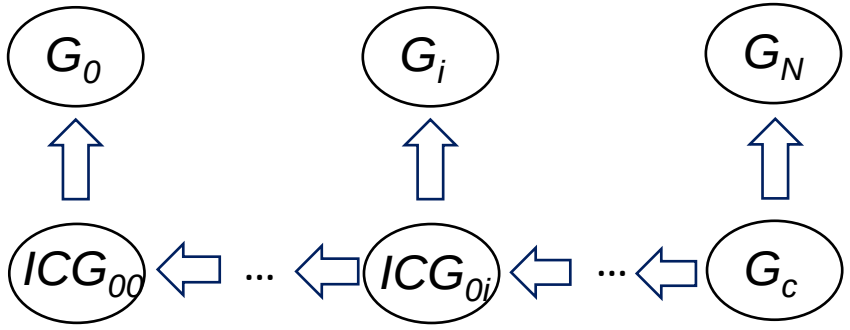
PIE: Storage System Summary

- ✓ No Write Amplification
- ✓ Low Read Amplification
- ✓ Crash Consistency--buffered durable linearizability



PIE: Streamlined Incremental Analysis

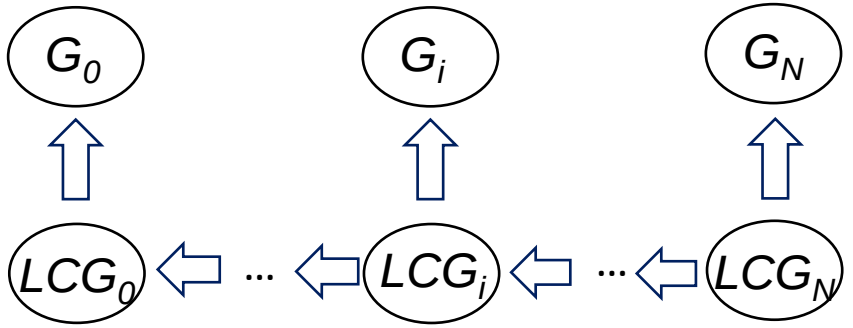
- A scheduling-free design
  - Instead of building a complete TG, we only only focus on the schedule path that goes through ICGs that are historical common graph.



(Historical) Common graph of snapshots in [0, i]

PIE: Streamlined Incremental Analysis

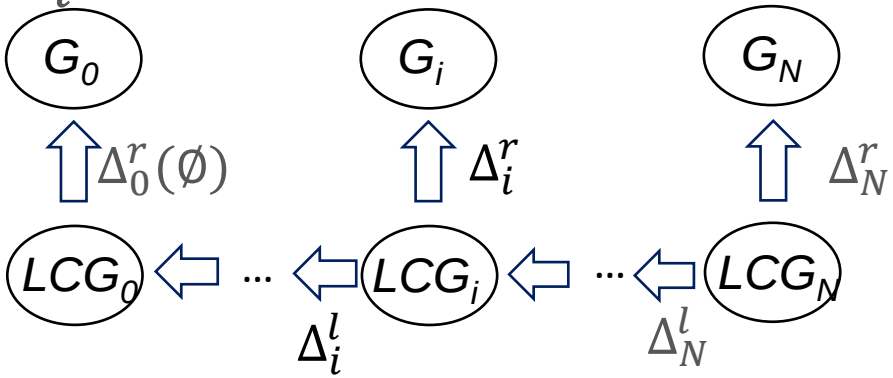
- A scheduling-free design
  - Lasting Common Graph (LCG)-driven deletion-free incremental analysis.
  - In this schedule, only half of IR deltas need to be calculated and these deltas could reuse the calculation of G_C.



(Historical) Common graph of snapshots in [0, i]

PIE: Streamlined Incremental Analysis

- LCG-driven deletion-free incremental analysis
  - Cal. right deltas $\Delta_i^r = \sum_{k=0}^{i-1} \Delta_{ki}^r$, including all right IR deltas in TG.
  - Cal. left deltas $\Delta_i^l = \Delta_{0i}^l$, including a few left IR deltas in TG. However, they could be acquired by directly reusing the calculation of LCG_i.



PIE: Streamlined Incremental Analysis

- LCG-driven deletion-free incremental analysis
  - Cal. right deltas $\Delta_i^r = \sum_{k=0}^{i-1} \Delta_{ki}^r$, including all right IR deltas in TG.
  - ~~Cal. left deltas $\Delta_i^l = \Delta_{0i}^l$, including a few left IR deltas in TG.~~
  - r, they could be acquired by reusing the calculation of

Half of deltas need to be calculated.

	PIE
LCG_i	$LCG_{i-1} - LCG_{i-1} \cap \Delta_i^-$
Δ_i^l	$LCG_{i-1} \cap \Delta_i^-$

PIE: Streamlined Incremental Analysis

- LCG-driven deletion-free incremental analysis
  - Cal. right deltas $\Delta_i^r = \sum_{k=0}^{i-1} \Delta_{ki}^r$, including all right IR deltas in TG.
  - The calculation of the right IR deltas could reuse the calculation of LCG_i and other right IR deltas.

	PIE
LCG_i	$LCG_{i-1} - LCG_{i-1} \cap \Delta_i^-$
Δ_i^l	$LCG_{i-1} \cap \Delta_i^-$
Δ_{ki}^r, $k = 0, \ldots, i-1$	Δ_i^+, $i - k = 1$ $\Delta_{k,i-1}^r - \neg LCG_{i-1} \cap \Delta_i^-$, $i - k > 1$

Evaluation Setup

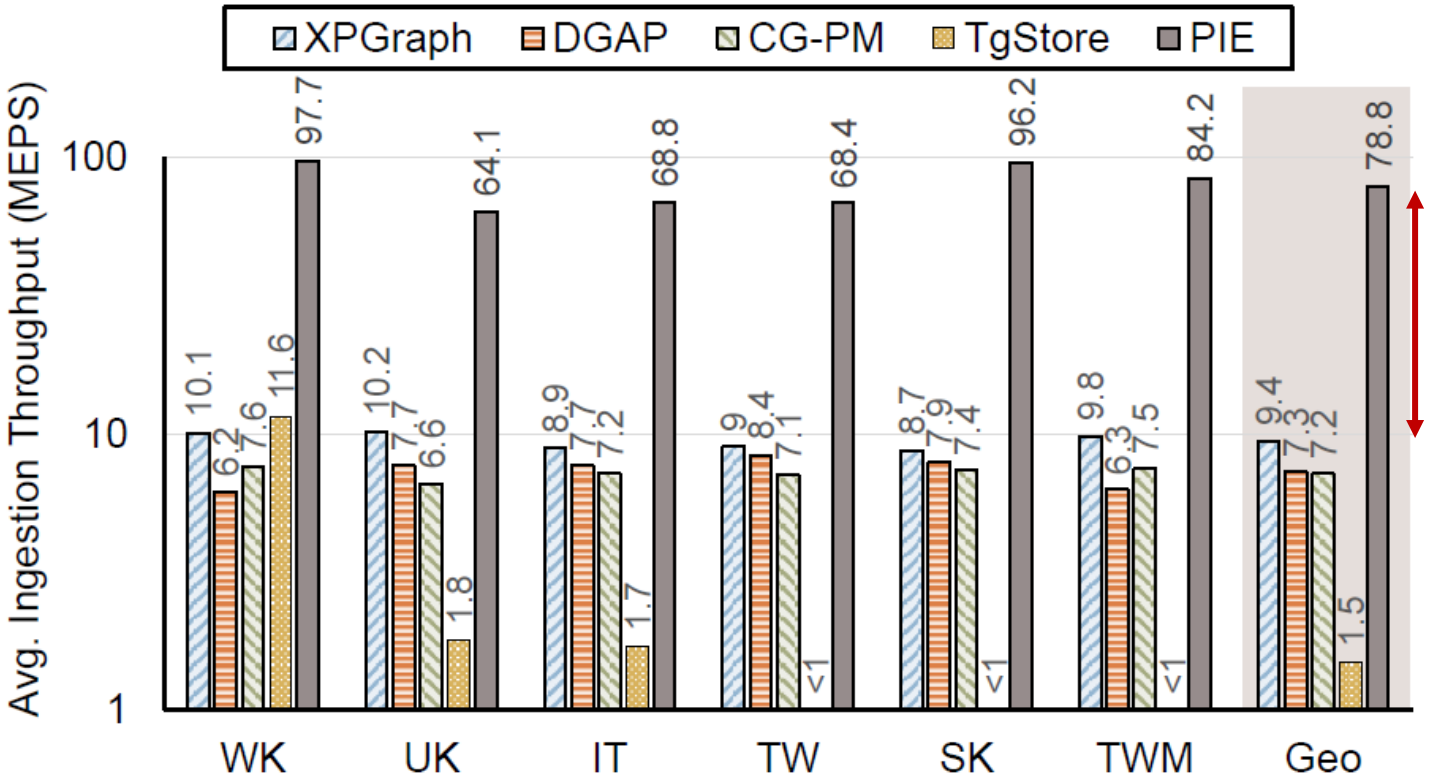
- Platform
  - A 12-core Linux server with Intel Xeon Gold 5317 CPU, 128 GB memory, and 1TB Optane Persistent Memory
- Graph queries
  - BFS, SSSP, SSWP, SSNP, Viterbi
- Graph datasets
  - 12 snapshots, with 0.5% Δ^+ and 0.5% Δ^- delta

Graph	V	E
WikiLinks (WK) [35]	13M	669M
uk-2005 (UK) [52]	39M	1.6B
It-2004 (IT) [52]	41M	2.1B
Twitter-2010 (TW) [36]	61M	2.4B
SK [52]	50M	3.7B
Twitter-MPI (TWM) [21]	53M	3.2B

- Compared Graph Storage
  - **XPGraph** [MICRO '22]
  - **DGAP** [SC '23]
  - CommonGraph on PM (**CG-PM**)
- Compared Graph Analytics
  - XPGraph + Kickstarter (**XPGraph+KS**)
  - DGAP + Kickstarter (**DGAP+KS**)
  - CommonGraph on PM (**CG-PM**)

Evaluation

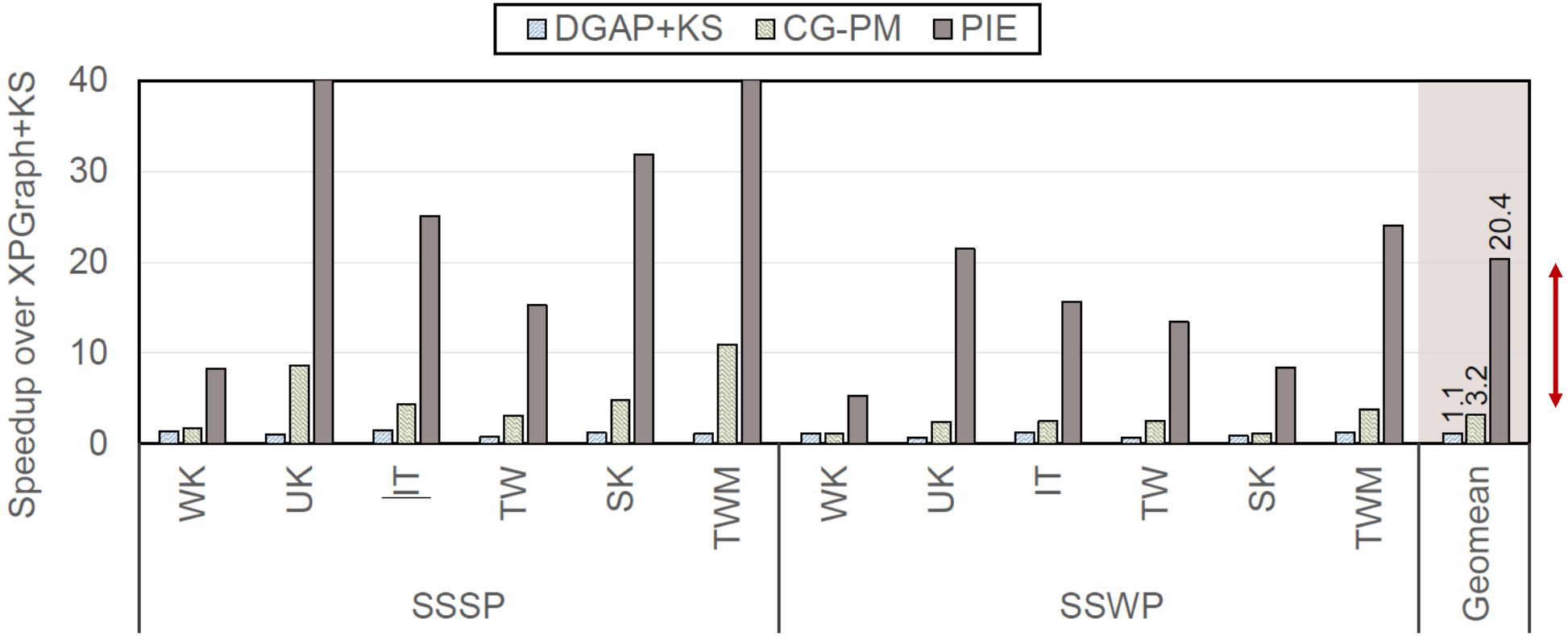
- Overall **Ingestion** Performance



Outperforms state-of-the-art by 8.4x

Evaluation

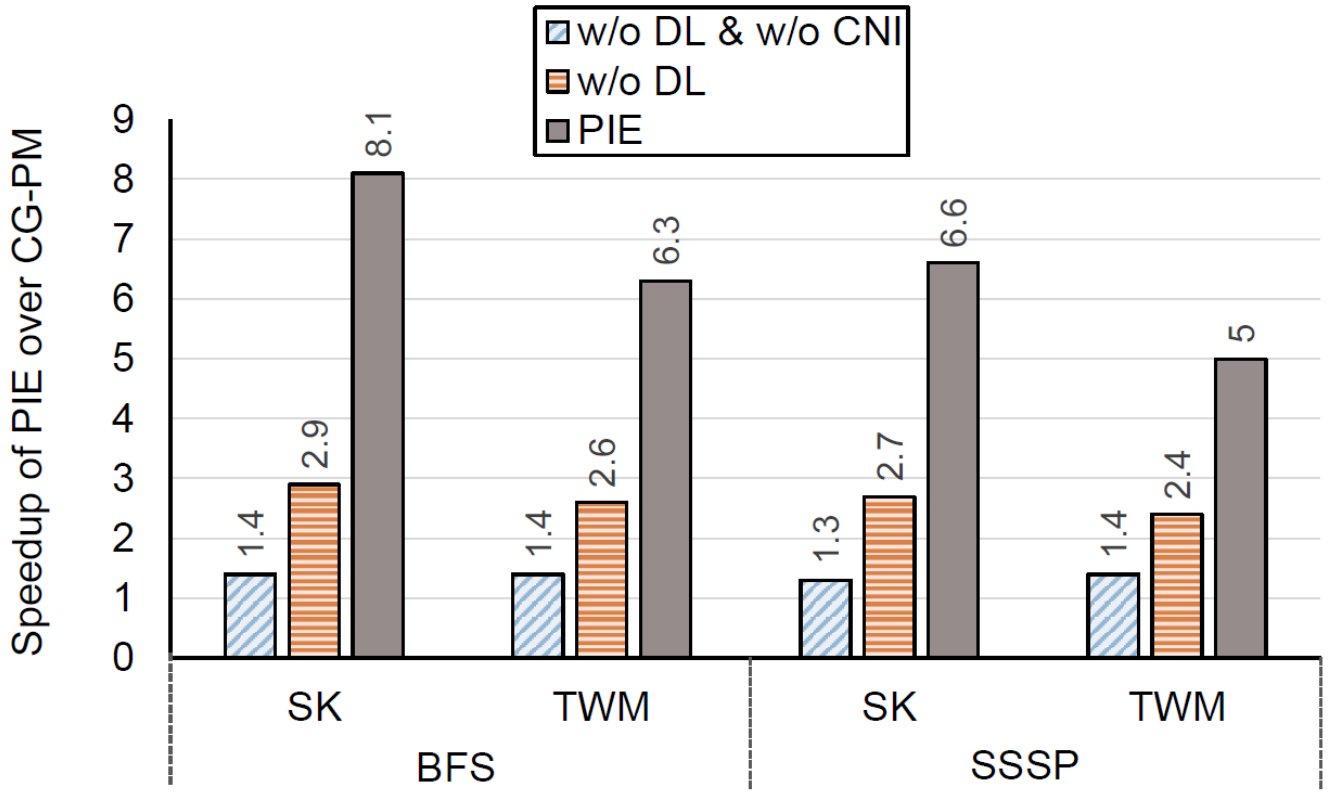
- Overall **Analytics** Performance



Outperforms state-of-the-art by 6.4x

Evaluation

- Breakdown Analysis
  - Detached Logical Graph View (DL)
  - Chunked Neighbor Index (CNI)



Evaluation

- Recovery Cost
  - in seconds

	WK	UK	IT	TW	SK	TWM
XPGraph	3.3	13.8	17.6	16.8	24.6	45.7
DGAP	1.6	4.2	5.0	5.8	8.3	7.7
PIE	0.8	2.2	2.4	3.1	4.5	3.9

- DRAM Usage (Peak)

	XPGraph	DGAP	PIE w/o Cache	PIE w/ Cache
WK	17.4GB	2.0GB	686.2MB	741.2MB
TW	21.8GB	9.4GB	2.8GB	3.2GB
SK	20.6GB	7.1GB	2.2GB	2.5GB

Best recovery speed and minimal DRAM requirement among PM-based systems.

More Experiments (in our paper)

- Ingestion Performance Sensitivity
  - to varying #snapshots
  - to varying the delta batch sizes
- Analysis Performance Sensitivity
  - to varying #snapshots
  - to varying the delta batch sizes
- Breakdown Analysis of optimizations

Summary

- PIE is an efficient PM-based evolving graph system that supports deletion-free incremental analysis.
  - Logical graph view for avoiding additional PM writes.
  - Chunked neighbor index for reducing read amplification.
  - Streamlined incre. analysis for less extra computations.

- PIE outperforms state-of-the-art solutions in ingestion, analytics, DRAM usage and recovery cost.

yunmo.zhang@my.cityu.edu.hk