GriNNder:
Breaking the Memory Capacity Wall in Full-Graph GNN Training with Storage Offloading

Jaeyong Song, Seongyeon Park, Hongsun Jang, Jaewon Jung,

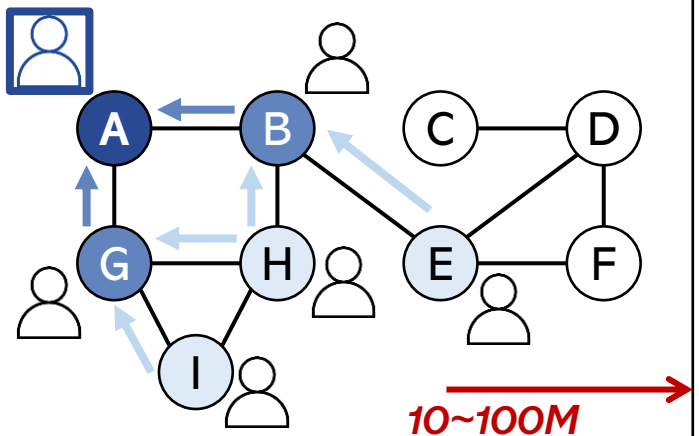
Hunseong Lim, Junguk Hong, and Jinho Lee

Electrical and Computer Engineering, Seoul National University

MLSys 2026

*Ninth Annual Conference on Machine Learning and Systems (**MLSys**)*

GriNNder

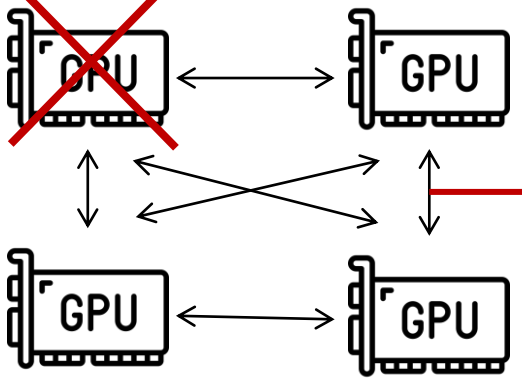


GNN effectively extracts A user embedding considering relationships

10~100M Full-Graph Training?

GPU/host memory capacity wall

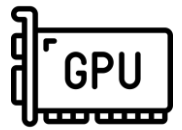
OOM



*Researchers report **OOM**, utilize expensive **distributed** setups, or need to co-design (search) complicated memory-saving sampling methods**

Cost, inter-comm. bottleneck

PCIe 5.0 NVMeSSD
10+ GB/s
+



= Cost effective, high-throughput full-graph GNN training

GriNNder achieves this goal with a holistic design of storage offloading in full-graph GNN training.

Up to 9.78x speedup over SOTA and even comparable throughput to distributed baselines with only a single GPU.

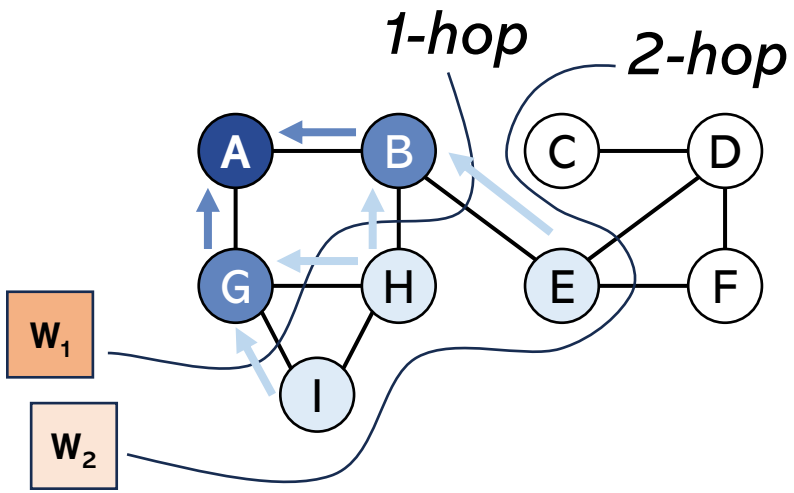
*: For our survey about recent GNN usages in conferences, please see the appendix.

Contents

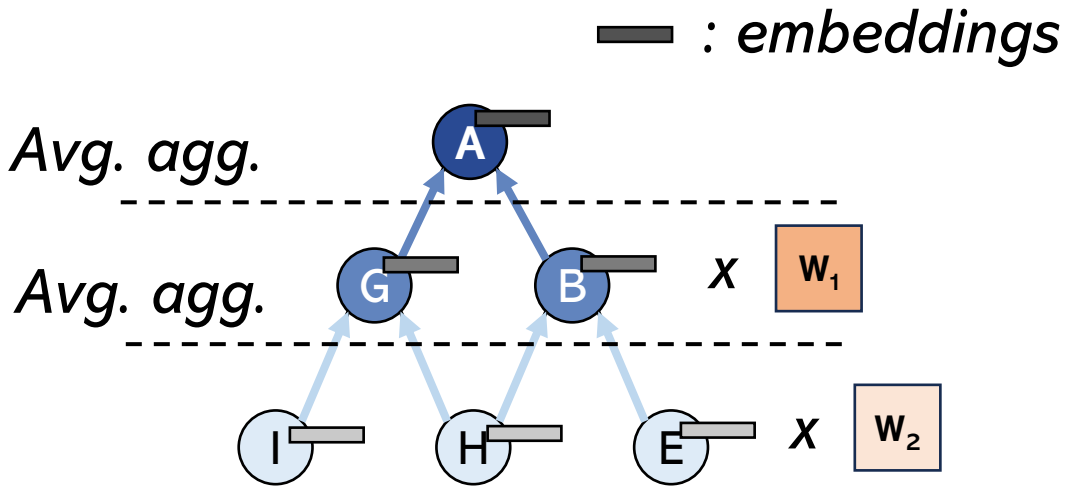
1. **Introduction**
   - Full-Graph GNN Training
   - Memory Capacity Wall of Full-Graph GNN Training
2. **Motivation**
   - Naïve Storage Offloading
   - Systemic Challenges
3. **GriNNder**
   - Cache-(Re)Gather-Bypass
   - Systemic Supports
4. **Evaluation**
5. **Conclusion**

Graph Neural Networks (GNNs)

- GNN aggregates information from neighbors and generates an embedding for a vertex
- MLPs are applied per hop and they are trained



(a) Graph neural networks

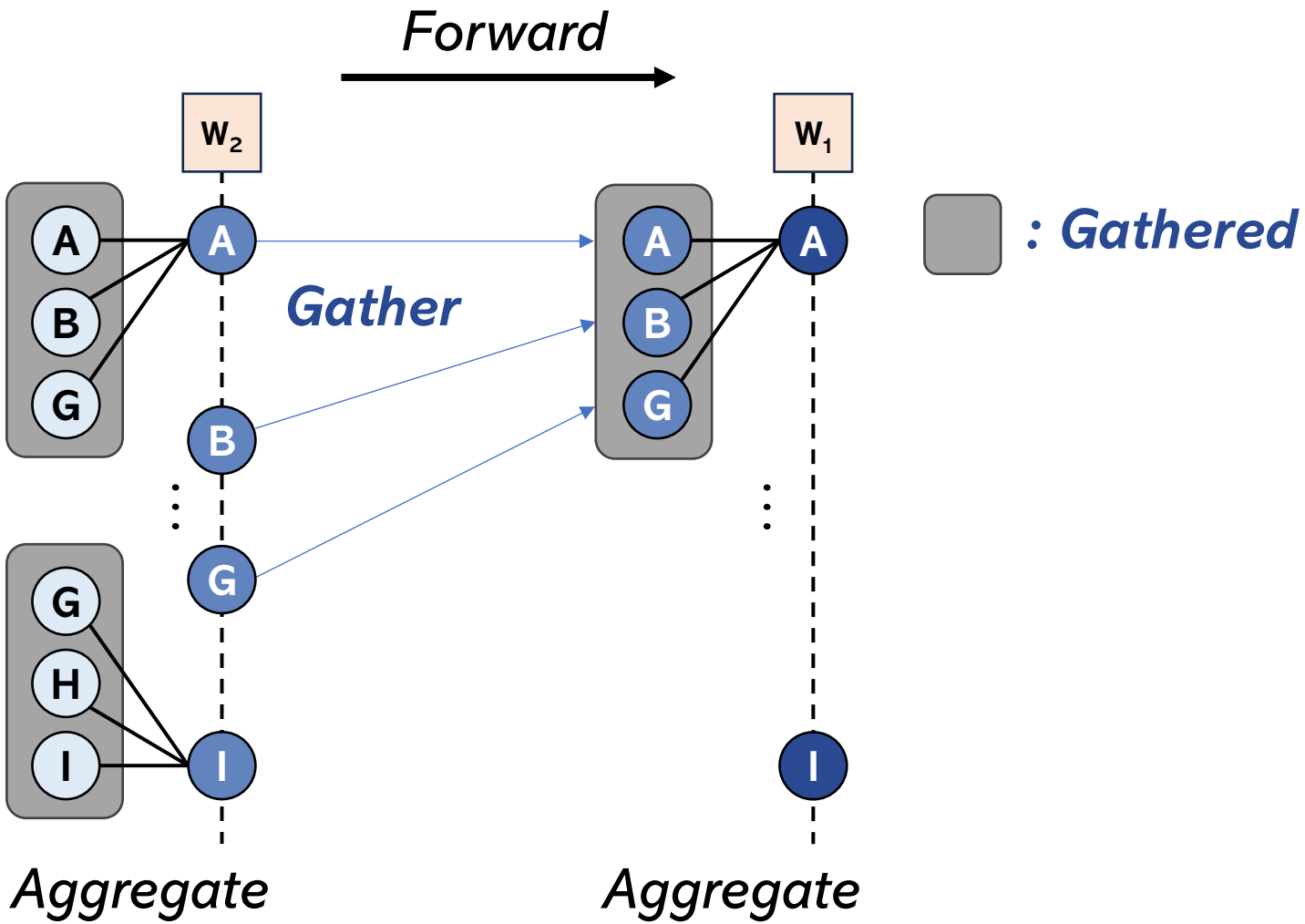


(b) Computation example

(= ***message passing***)

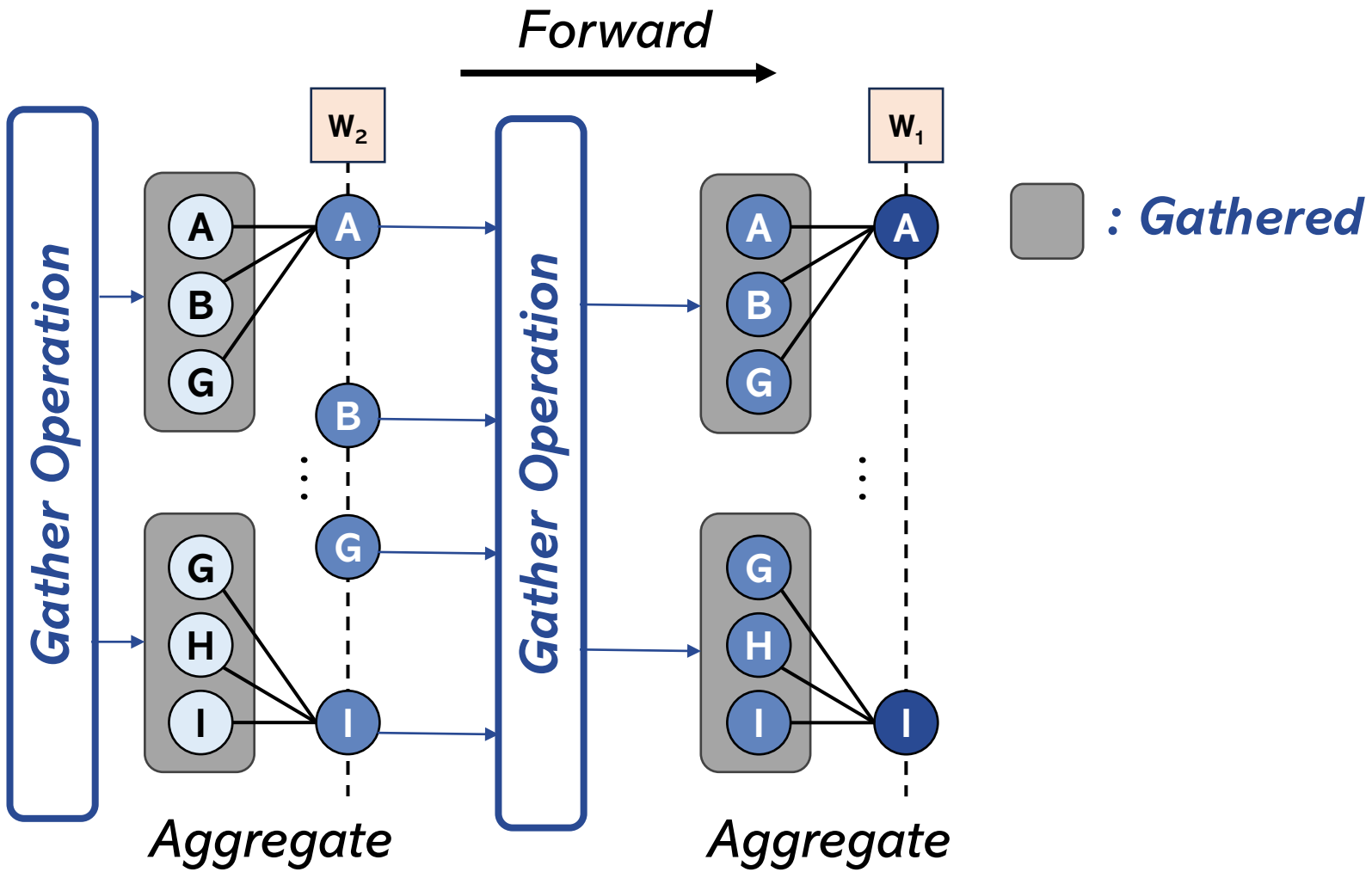
GNN Training: Forward

- During forward, we perform ***gather***, which collects embeddings of all its neighbors and aggregate them.



GNN Training: Forward

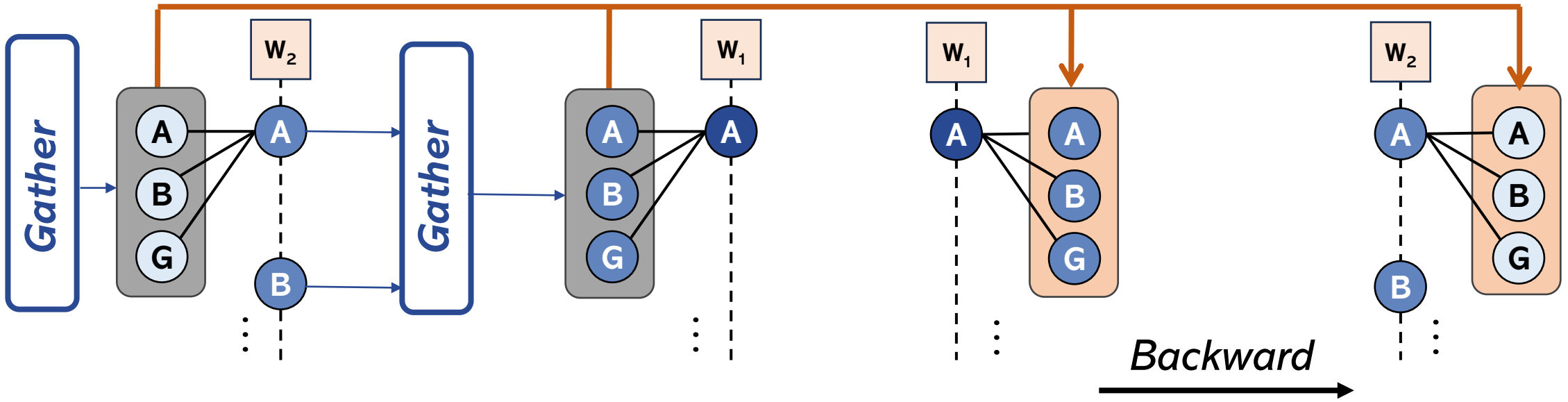
- During forward, we perform ***gather***, which collects embeddings of all its neighbors and aggregate them.



GNN Training: Backward

: Gathered : Snapshot

- During forward, we ***snapshot the gathered activations*** for the future use in the backward pass.
- Snapshots are required to calculate the gradients.



GNN Training: Backward

: Gathered : Snapshot

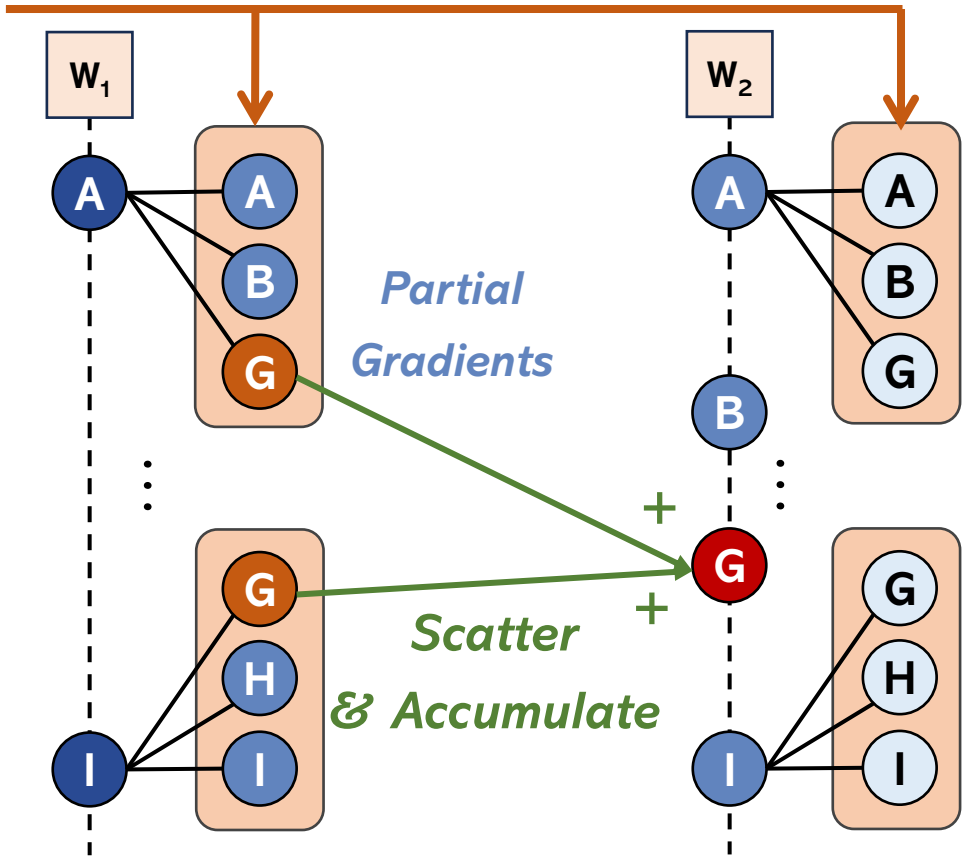
- After backward, each layer generates *partial gradients* of a vertex.
- Thus, we *scatter & accumulate* the gradients for the next layer BW

Backward regarding vertex A

Generates partial gradient for G

Backward regarding vertex I

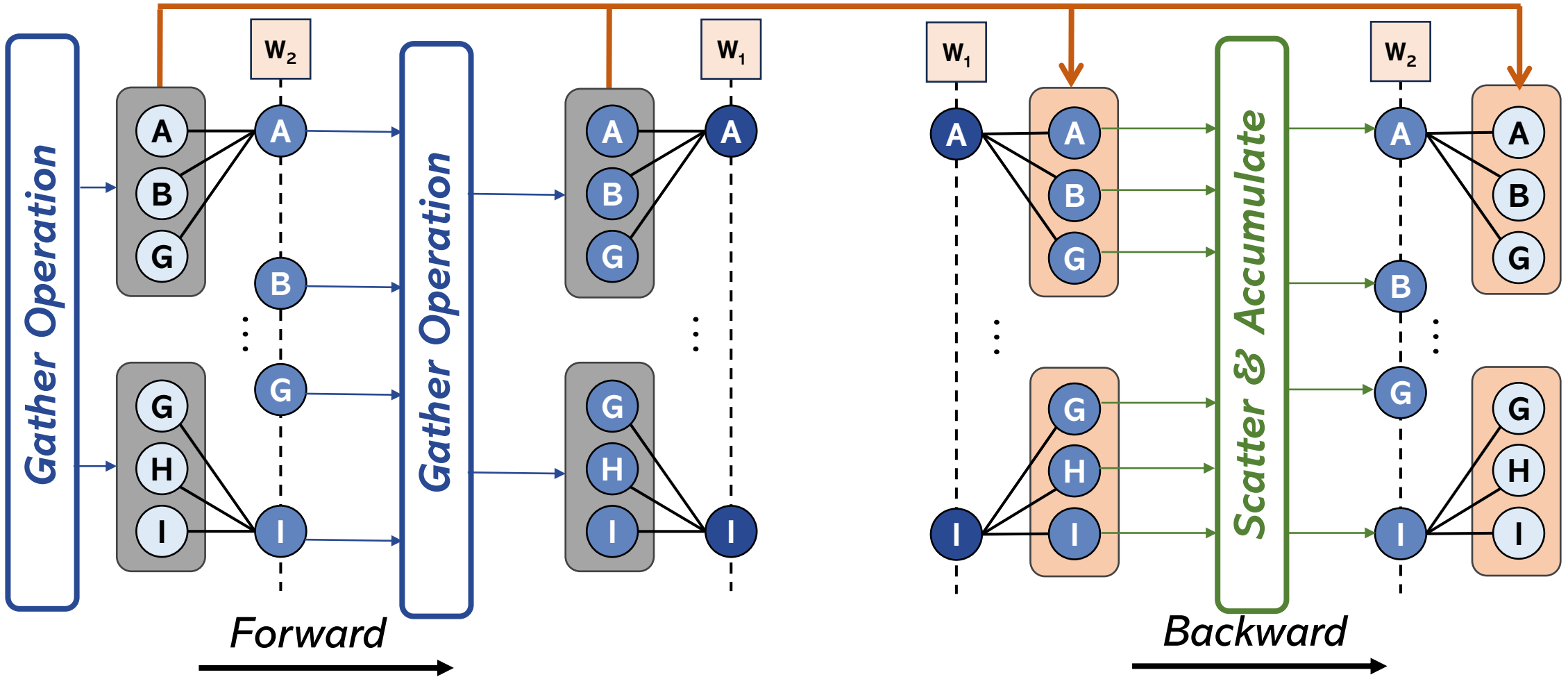
Generates partial gradient for G



GNN Training: Overall

: Gathered : Snapshot

- Forward propagates all vertices' information using ***gather***.
- Backward back-propagates all vertices' gradients using the ***snapshots*** from the forward and ***scatter & accumulation***.



GNN Training: Overall

- Forward propagates all vertices' information using *gather*.
- Backward back-propagates all vertices' gradients using the *snapshots* from the forward and *scatter & accumulation*.

Severe memory capacity pressure!

e.g., 100M-sized graph with 512 hidden size and 3 layers
➔ *1.2TB* GPU memory

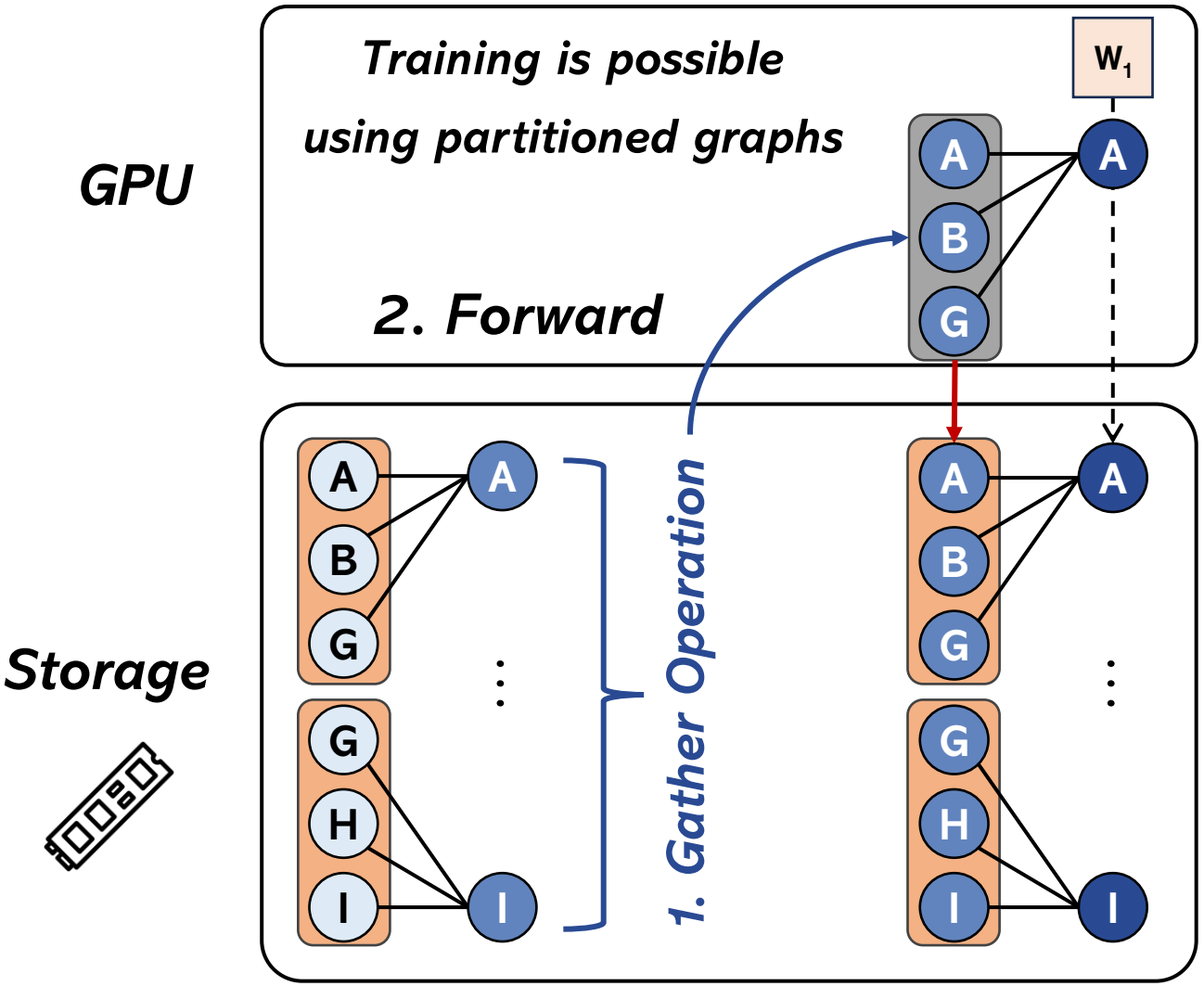
For existing approaches to address this issue, please see the paper.

Contents

1. Introduction
   - Full-Graph GNN Training
   - Memory Capacity Wall of Full-Graph GNN Training
2. **Motivation**
   - Naïve Storage Offloading
   - Systemic Challenges
3. GriNNder
   - Cache-(Re)Gather-Bypass
   - Systemic Supports
4. Evaluation
5. Conclusion

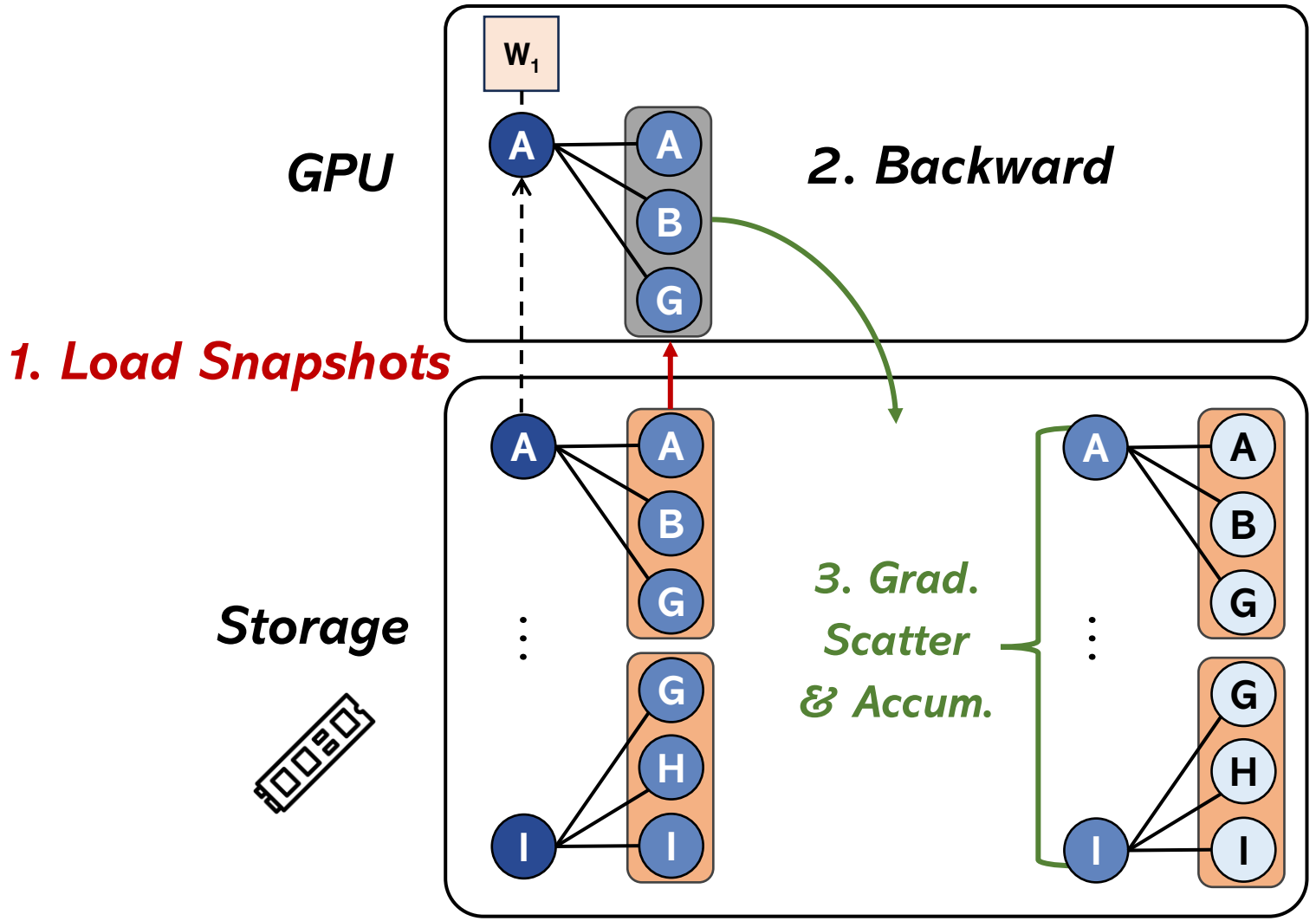
Naïve Storage Employment: Forward

- Offload activations on NVMe SSD and load ***only a single partition***

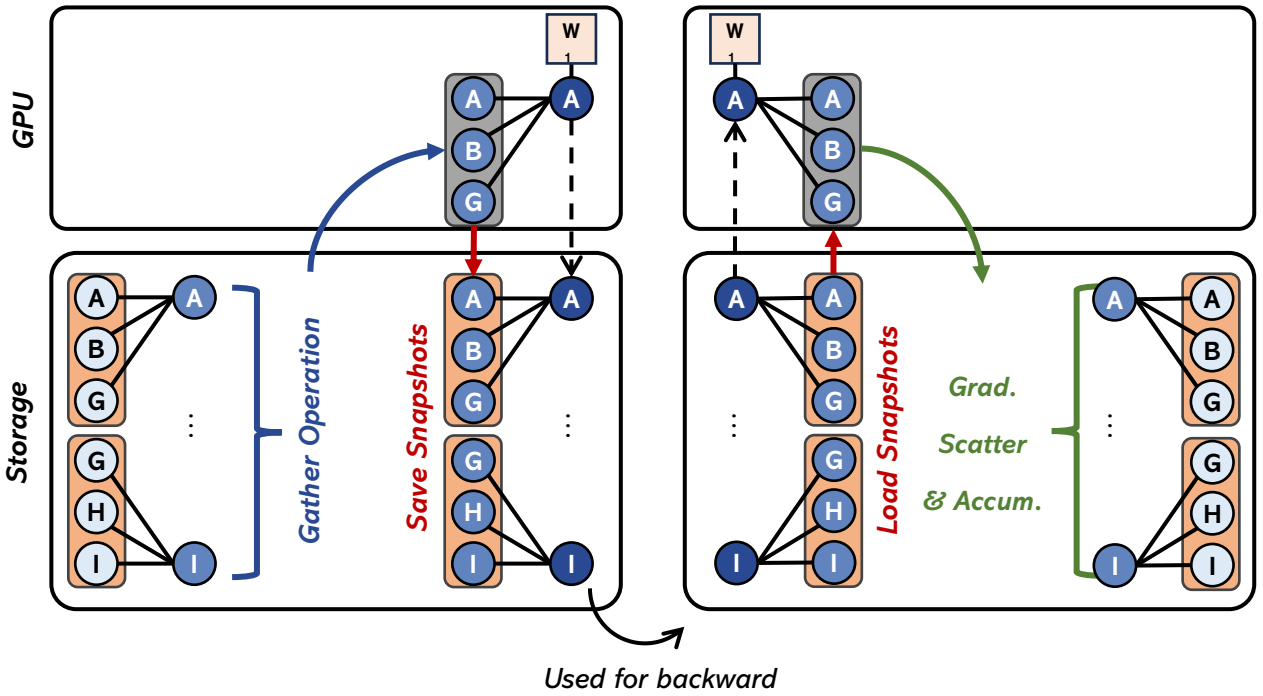


Naïve Storage Employment: Backward

- Offload activations on NVMe SSD and load ***only a single partition***



Naïve Storage Employment: Limitations



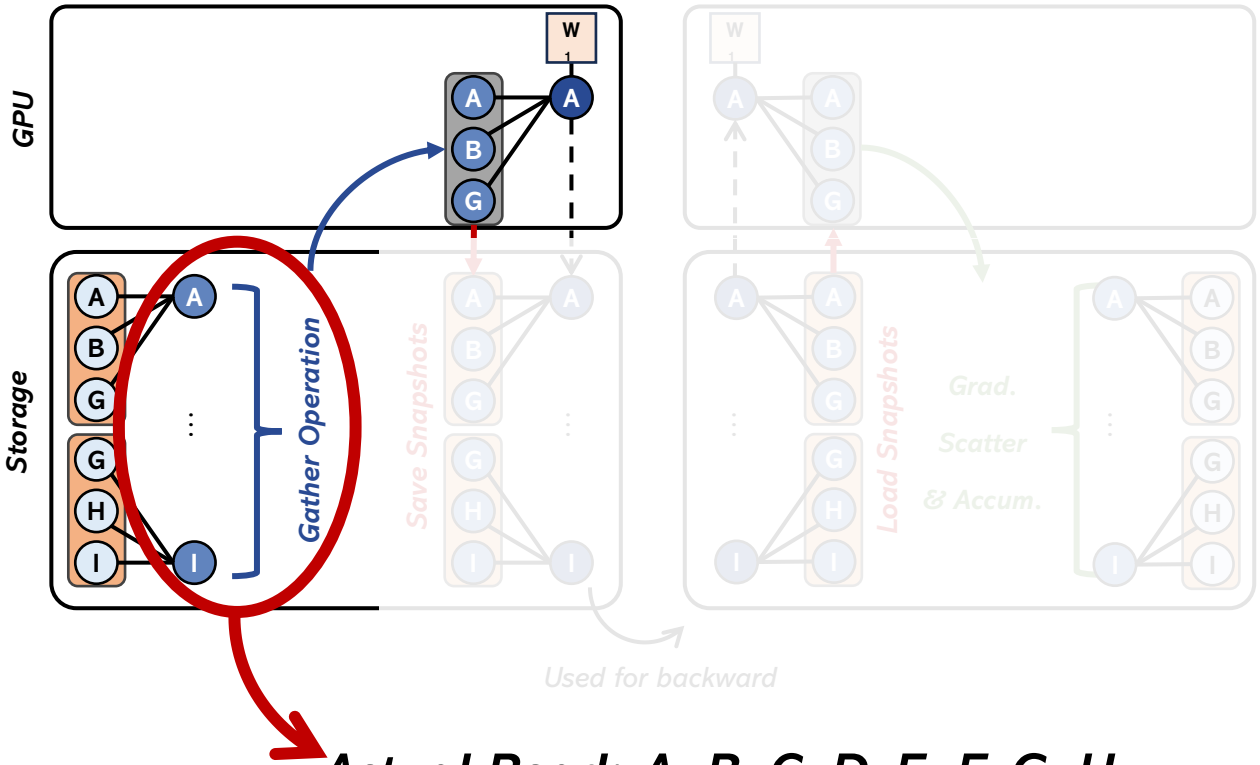
1. Random Fine-Grained Gather

- Read amplification and bandwidth saturation to storage

2. Redundancy in Snapshots

- Redundant store of activations due to snapshotting

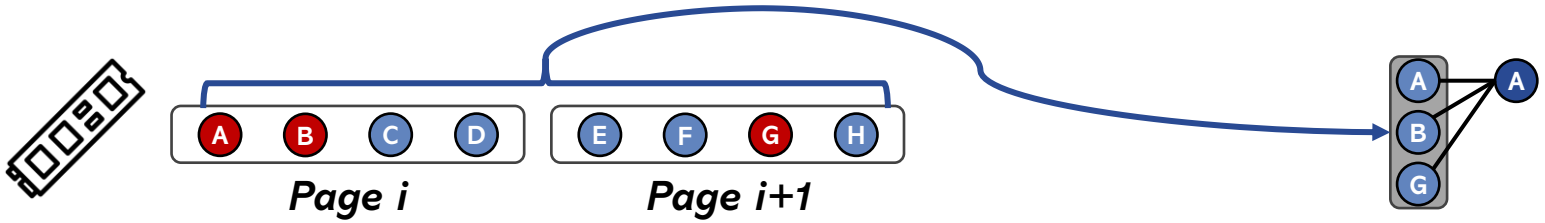
Naïve Storage Employment: Limitations



1. **Random Fine-Grained Gather**
   - Read amplification and bandwidth saturation to storage

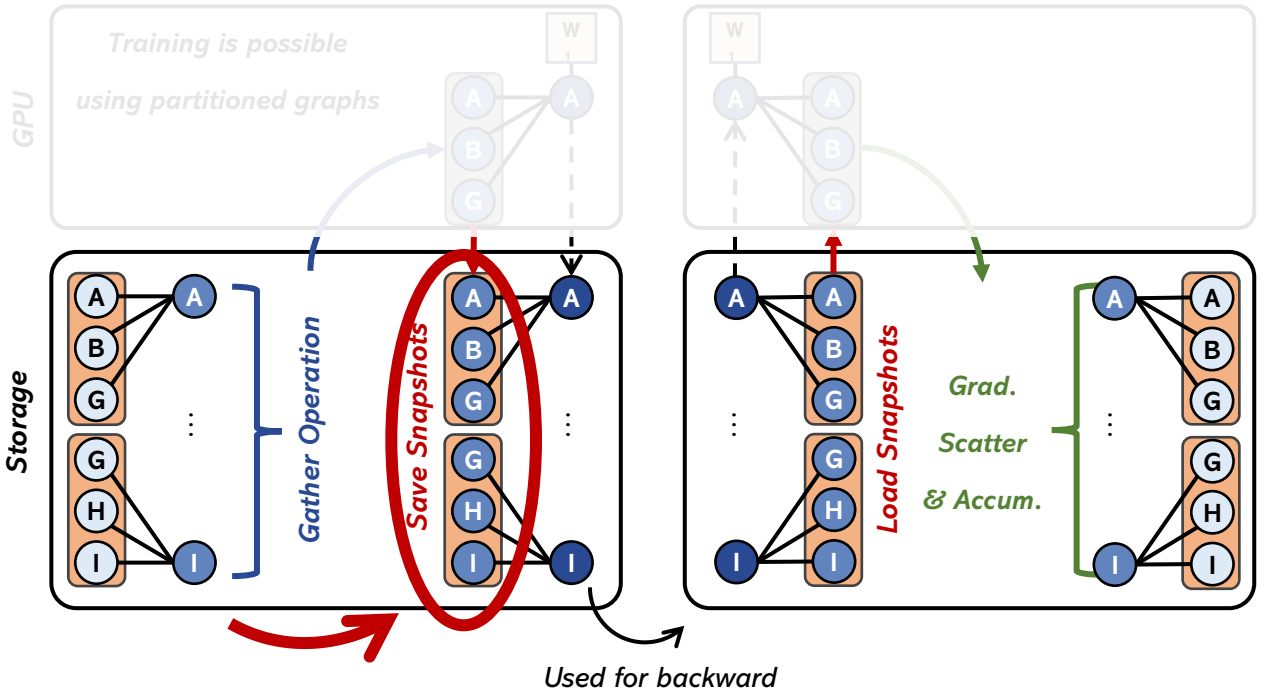
2. **Redundancy in Snapshots**
   - Redundant store of activations due to snapshotting

Actual Read: A, B, C, D, E, F, G, H ***Required: A, B, G***



Read amplification

Naïve Storage Employment: Limitations



Already exists, but redundantly stored!

1. Random Fine-Grained Gather

- Read amplification and bandwidth saturation to storage

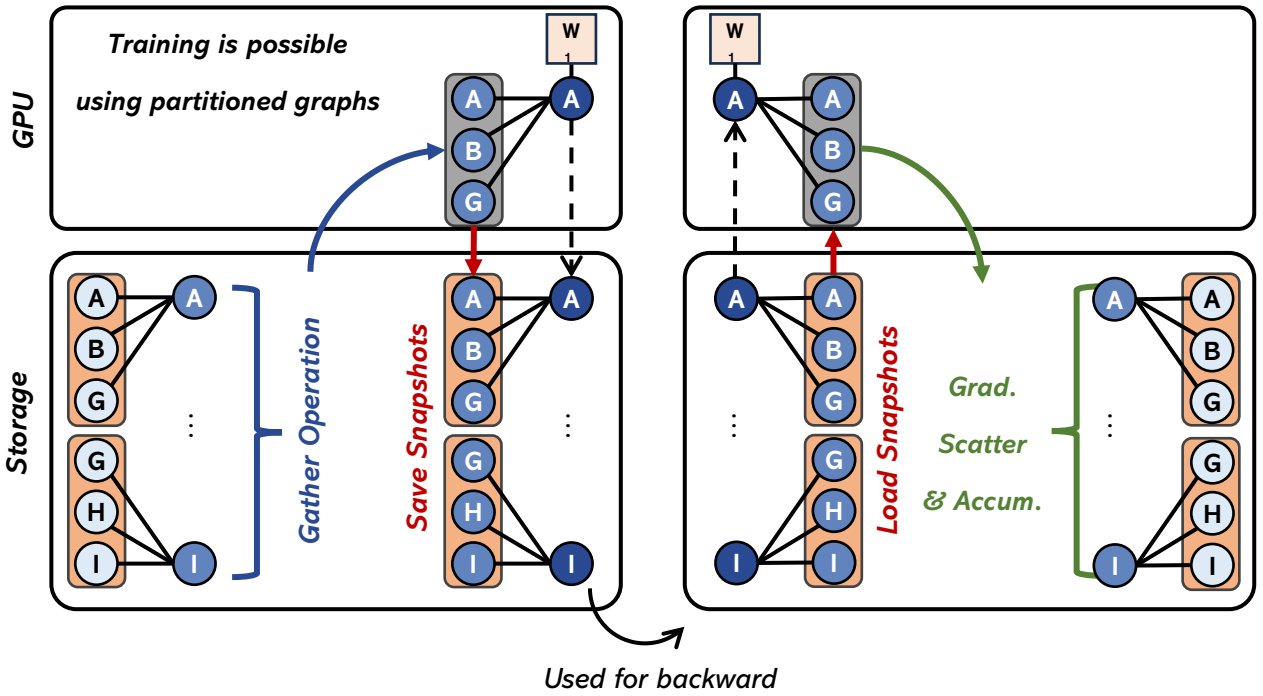
2. Redundancy in Snapshots

- Redundant store of activations due to snapshotting

Beneficial for sequential accesses

But incurs significant I/O for storage usage

Naïve Storage Employment: Limitations



1. Random Fine-Grained Gather

- Read amplification and bandwidth saturation to storage

2. Redundancy in Snapshots

- Redundant store of activations due to snapshotting

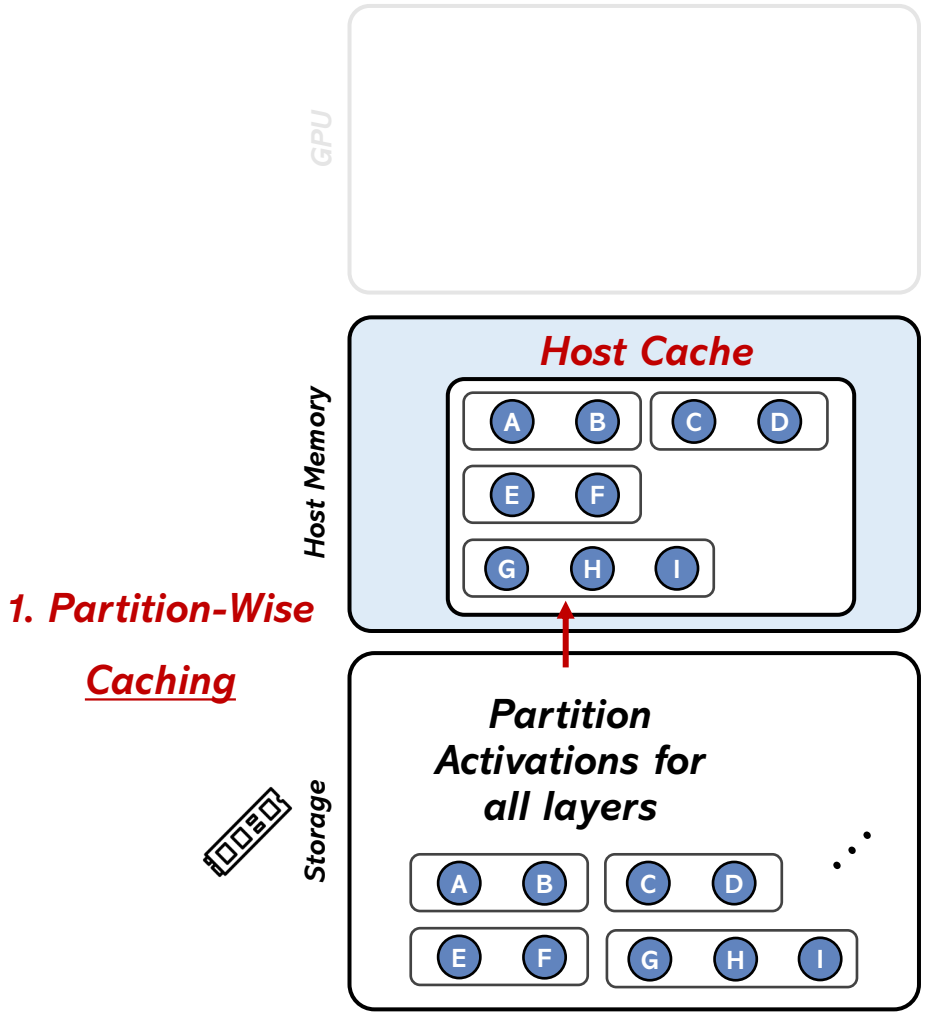
GriNNder addresses these limitations via *cache-(re)gather-bypass*

Contents

1. Introduction
   - Full-Graph GNN Training
   - Memory Capacity Wall of Full-Graph GNN Training
2. Motivation
   - Naïve Storage Offloading
   - Systemic Challenges
3. **GriNNder**
   - **Cache-(Re)Gather-Bypass**
   - Systemic Supports
4. Evaluation
5. Conclusion

GriNNder: Cache-(Re)Gather-Bypass

Cache-(Re)Gather-Bypass: Forward



Partition-Wise Caching

+ Activation Gather

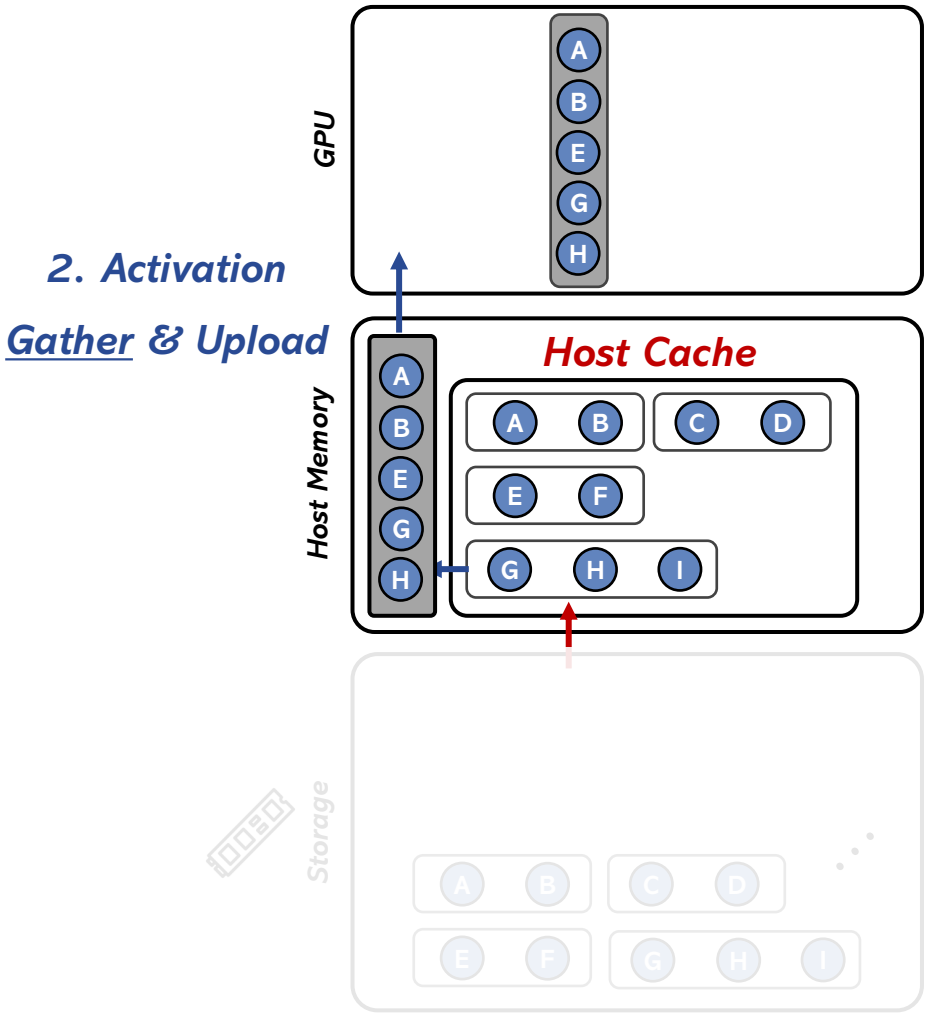
→ Reduces read amplification and bandwidth saturation to storage

Skip Snapshot & Bypass

→ Eliminate redundant store of activations from snapshotting

GriNNder: Cache-(Re)Gather-Bypass

Cache-(Re)Gather-Bypass: Forward



Partition-Wise Caching

\+ ***Activation Gather***

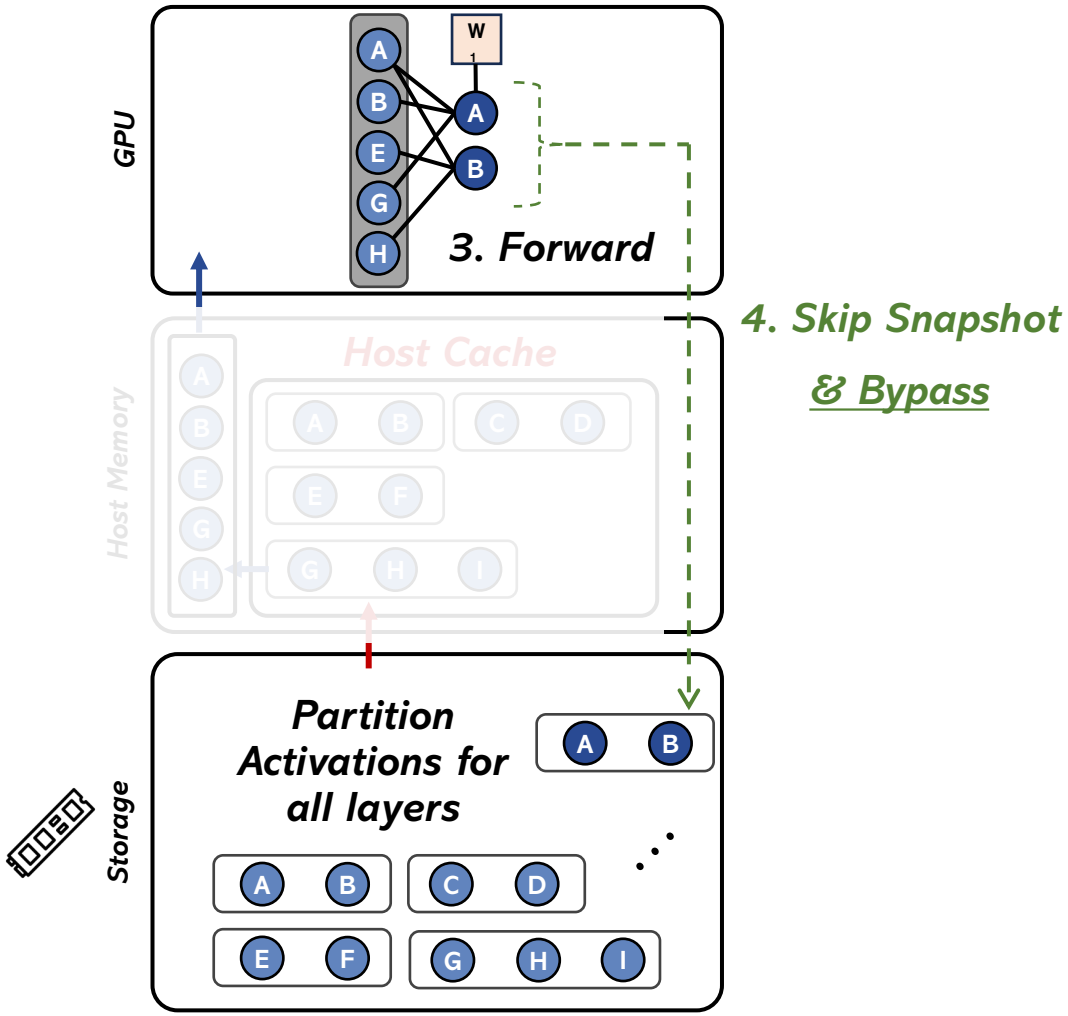
→ Reduces read amplification and bandwidth saturation to storage

Skip Snapshot & Bypass

→ Eliminate redundant store of activations from snapshotting

GriNNder: Cache-(Re)Gather-Bypass

Cache-(Re)Gather-Bypass: Forward



Partition-Wise Caching

+ Activation Gather

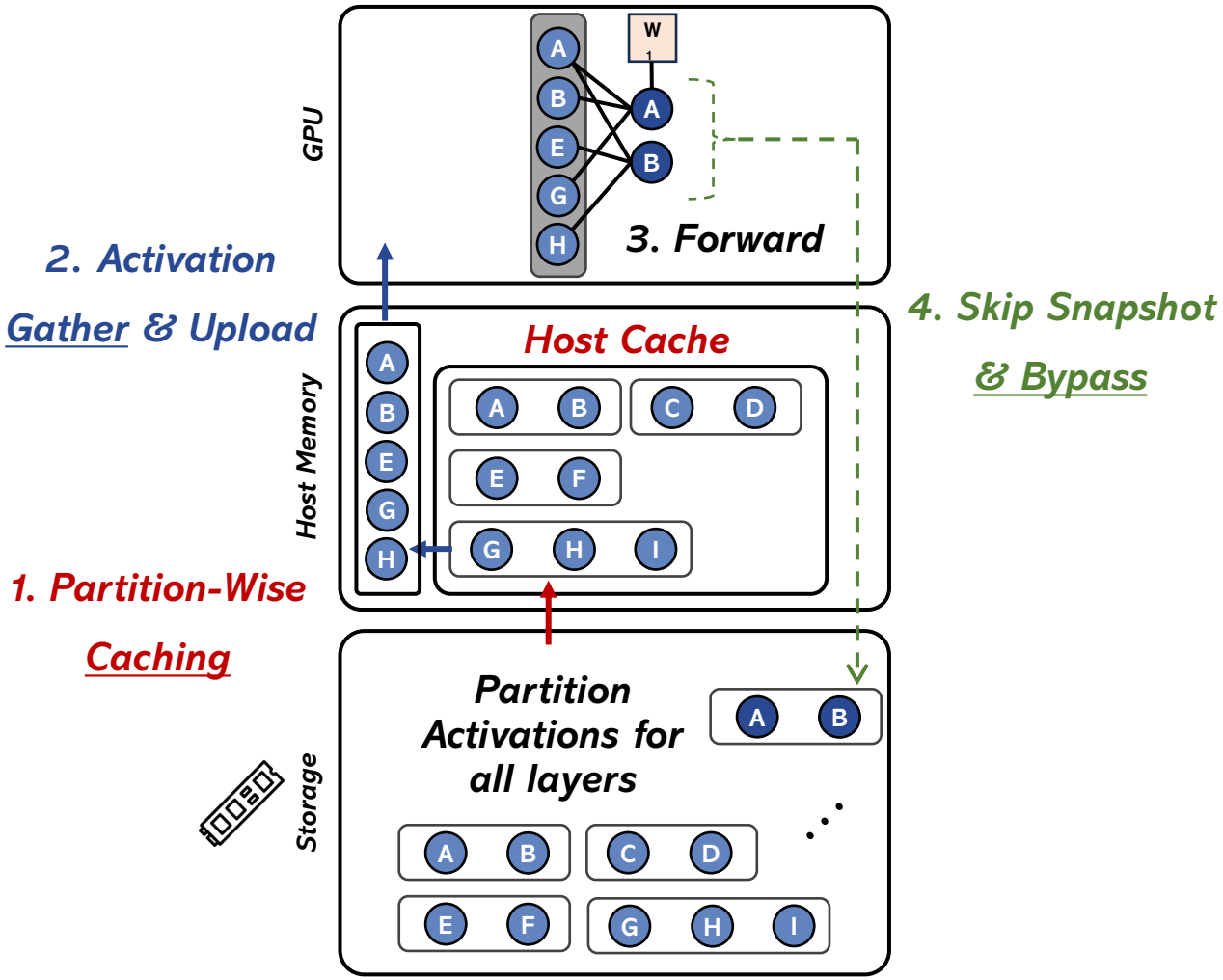
→ Reduces read amplification and bandwidth saturation to storage

Skip Snapshot & Bypass

→ Eliminate redundant store of activations from snapshotting

GriNNder: Cache-(Re)Gather-Bypass

Cache-(Re)Gather-Bypass: Forward



Partition-Wise Caching

+ *Activation Gather*

→ Reduces read amplification and bandwidth saturation to storage

Skip Snapshot & Bypass

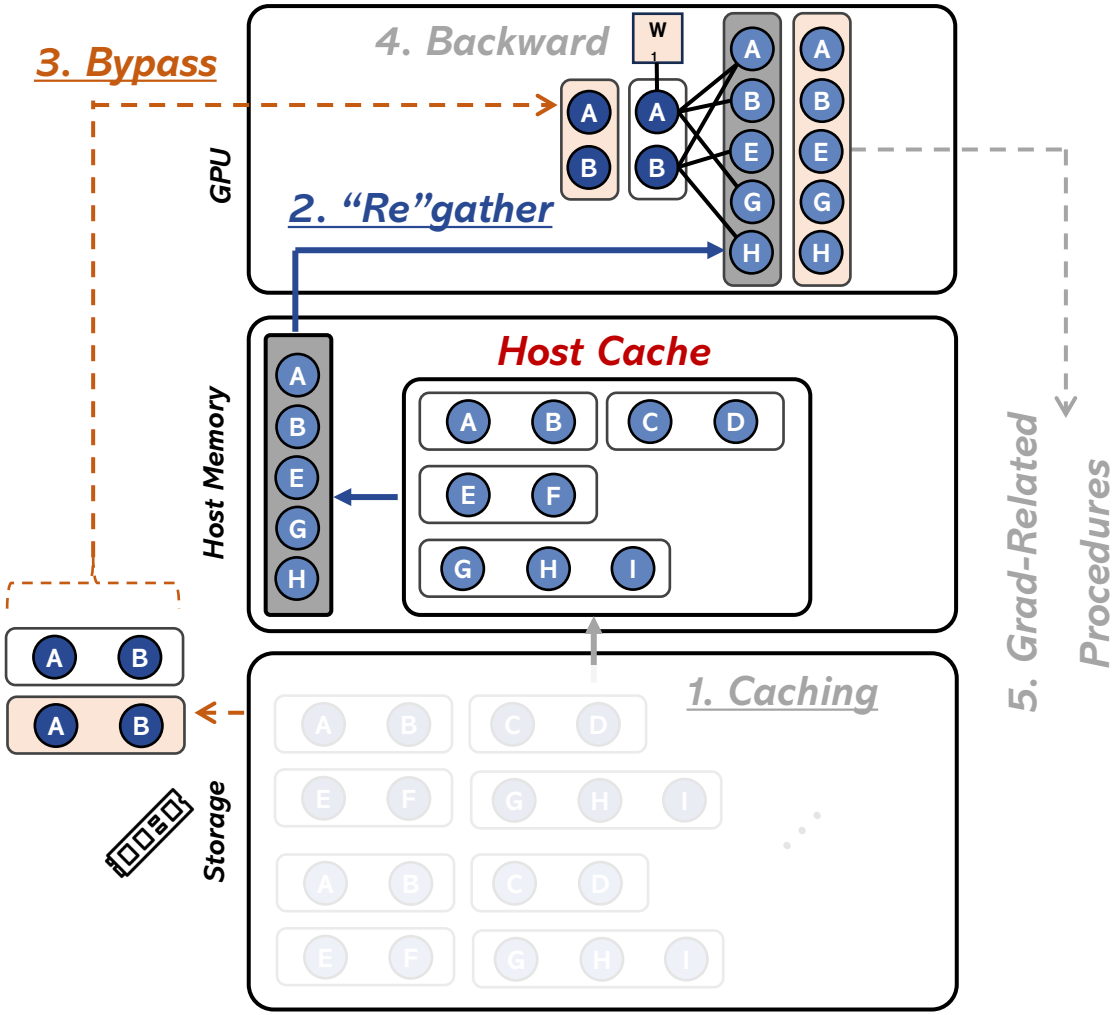
→ Eliminate redundant store of activations from snapshotting

No Snapshot
→ How to support BW?

GriNNder: Cache-(Re)Gather-Bypass

Cache-(Re)Gather-Bypass: Backward

: Gradients



No Snapshot
→ How to support BW?

Partition-Wise Caching
→ Reduces read amplification

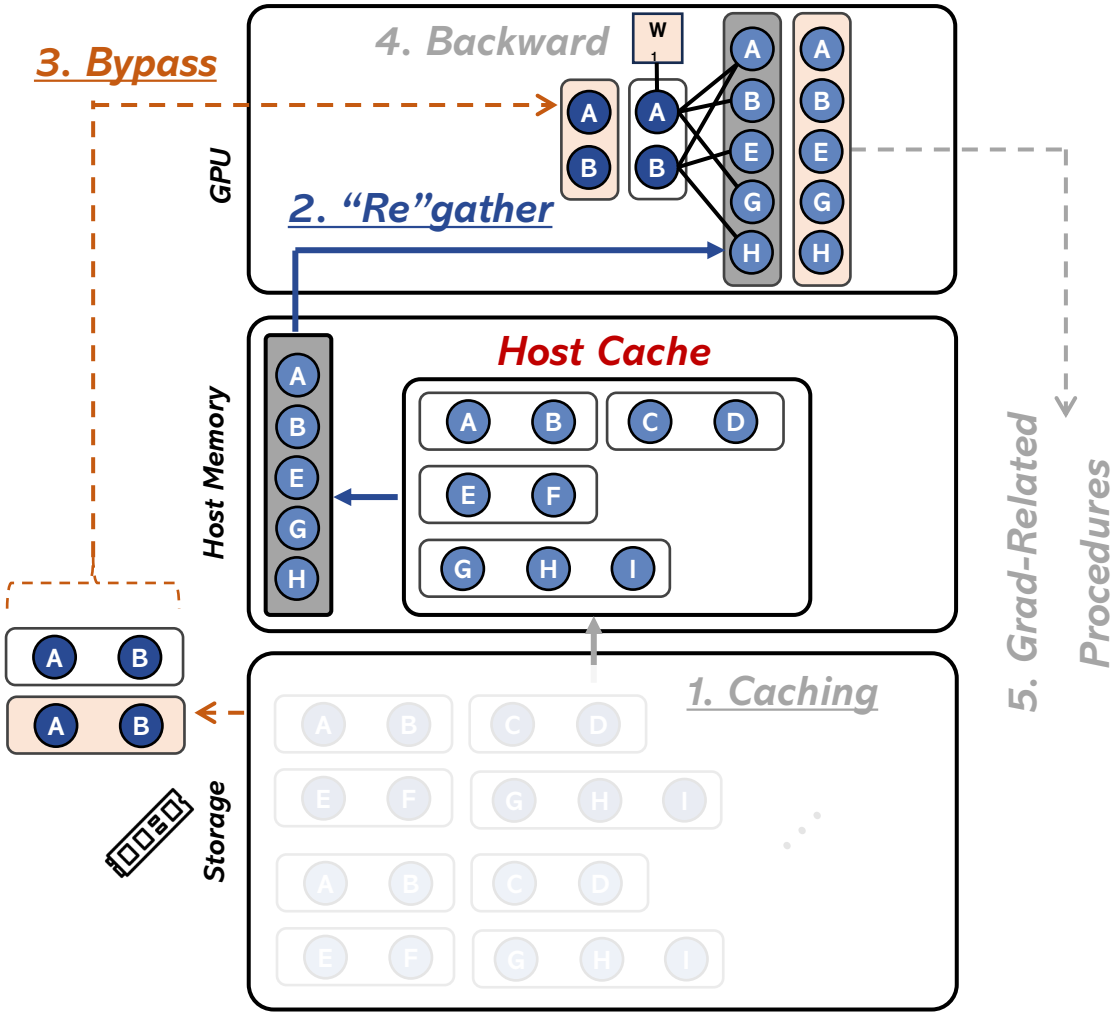
2. Activation "Re"gather
→ Instead of ***Snapshot!***

3. Bypass Input Act./Grad.
→ Do not cache low-reuse elements
(e.g., input activations & gradients)

GriNNder: Cache-(Re)Gather-Bypass

Cache-(Re)Gather-Bypass: Backward

: Gradients



No Shapshot
→ How to support BW?

1) How to cache?
2) How to support regathering in gradient engine?

3. Bypass Input Act./Grad.
→ Do not cache low-reuse elements (e.g., input activations & gradients)

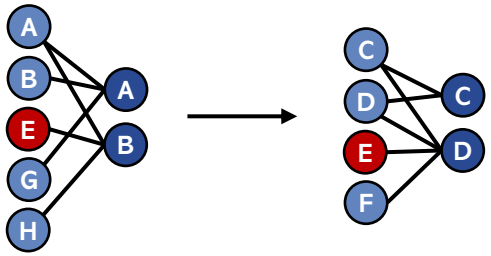
Contents

1. Introduction
   - Full-Graph GNN Training
   - Memory Capacity Wall of Full-Graph GNN Training
2. Motivation
   - Naïve Storage Offloading
   - Systemic Challenges
3. **GriNNder**
   - Cache-(Re)Gather-Bypass
   - **Systemic Supports**
4. Evaluation
5. Conclusion

GriNNder: How to cache?

Why caching is important?

- Intra-layer reuse!



Partition 0 *Partition 1*

e.g., vertex e is used multiple times

∴ Layer-Wise Caching Policy

If can handle all layers
- Cache all layers

Else if can handle subset of layers
- Cache layers in LRU order

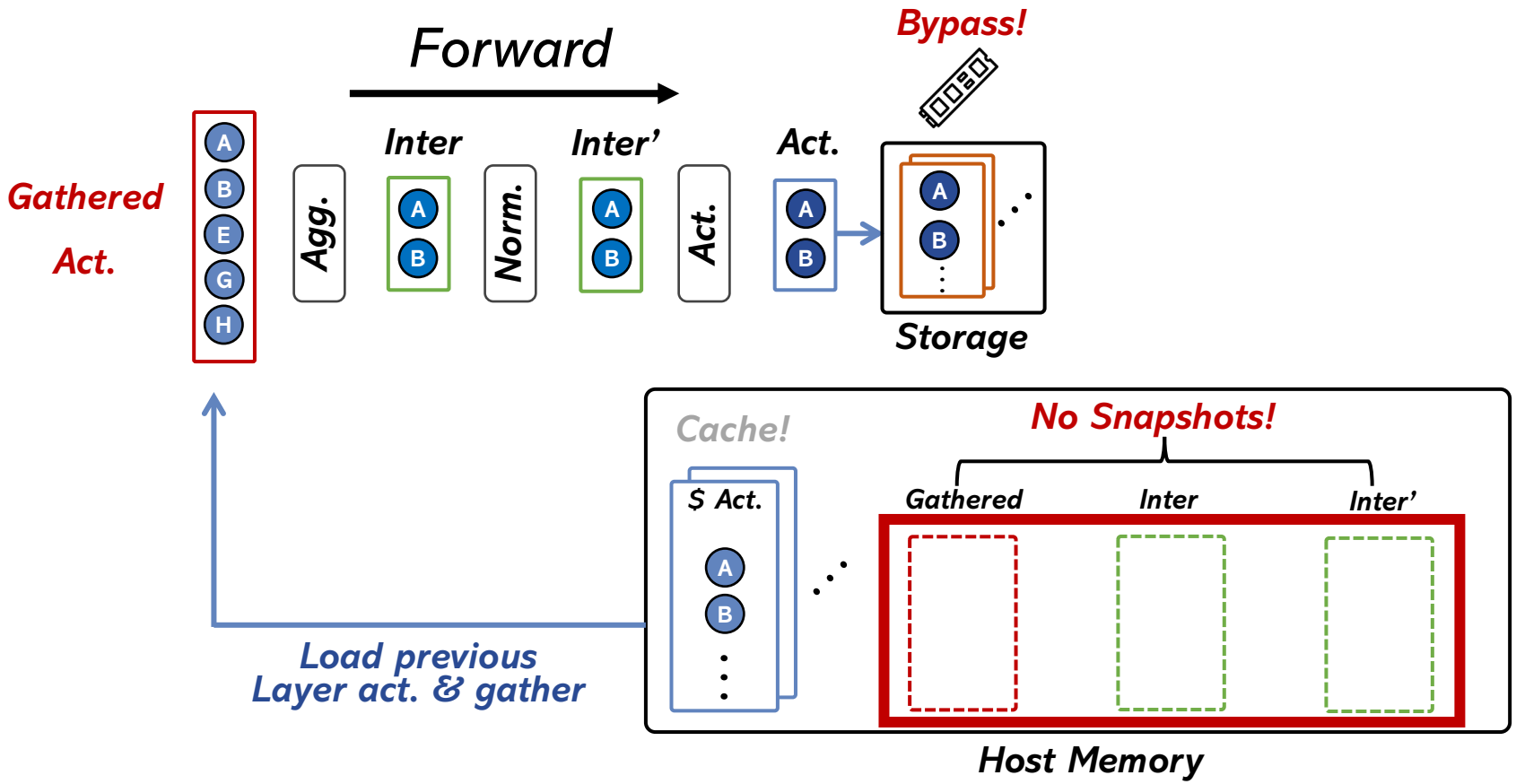
Else
- ***Partition-wise LRU replacement***

Why not vertex-wise replacement?

- Fine-grained accesses are unsuitable for storage pages
- Partition dependency already follows power law

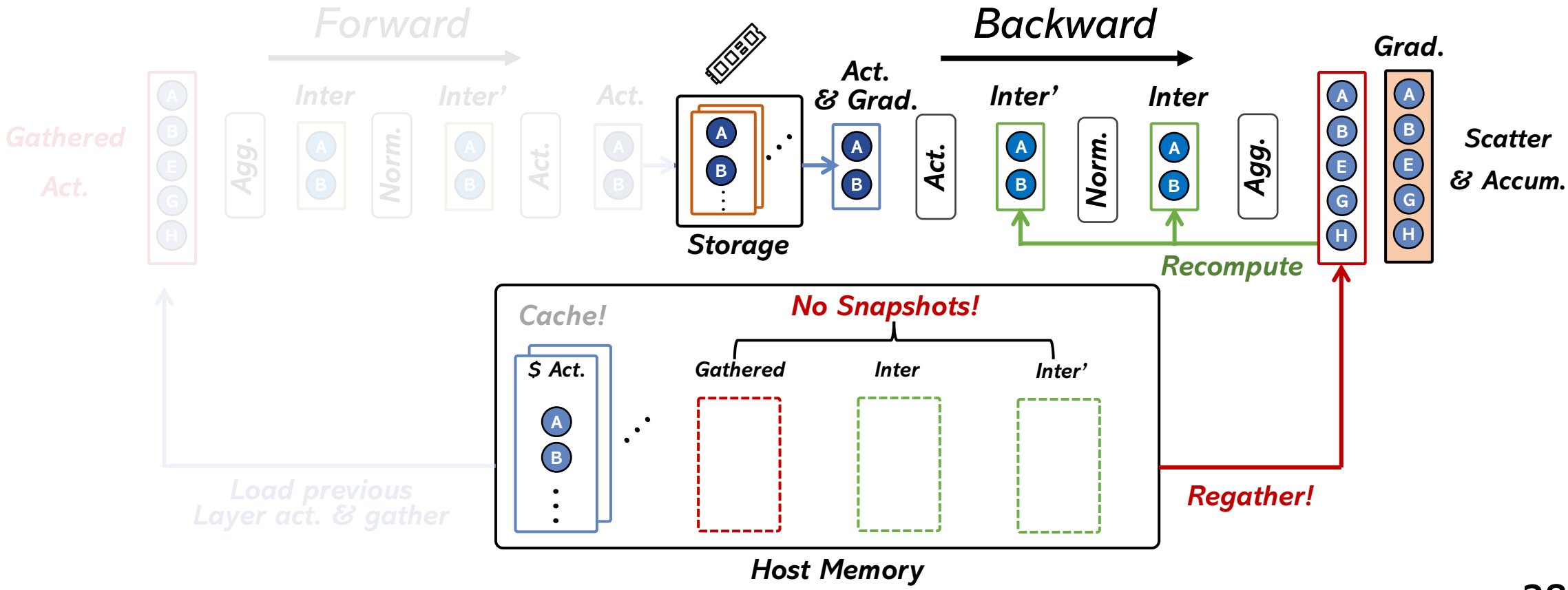
GriNNder: How to support regathering?

- ***Skip snapshots*** during forward.
- Directly write generated activations to storage through ***bypassing***
- Minimize the host memory usage and eliminate redundant I/O considering storage



GriNNder: How to support regathering?

- *"Regather"* instead of snapshots.
- *"Recompute"* intermediate activations.
- Minimize the host memory usage and eliminate redundant I/O considering storage



Contents

1. Introduction
   - Full-Graph GNN Training
   - Memory Capacity Wall of Full-Graph GNN Training
2. Motivation
   - Naïve Storage Offloading
   - Systemic Challenges
3. GriNNder
   - Cache-(Re)Gather-Bypass
   - Systemic Supports
4. Evaluation
5. Conclusion

Evaluation Setup

Workstation Setup

- CPU: AMD Ryzen9 7950X3D 16C 32T
- DRAM: 128GB DDR5-5600
- GPU: a single RTX A5000 24GB
- SSD: PCIe 5.0 NVMe SSD 4TB

Distributed Setup for distributed baselines

- For each server,
- CPU: 2x Intel Xeon Gold 6442Y
- DRAM: DDR5 512GB
- GPU: 4x RTX A6000 48GB
- Interconnect: NVLink Bridge, Infiniband SDR
- A total of 16 GPUs (four servers)

Datasets & Models

- Datasets: Products (2.4M), IGBM (10M), **Papers (100M)**
- Models: 3-layer GCN, hidden size=256

Baselines

- Distributed: CAGNET, SANCUS
- Single GPU: Betty, Ginex, HongTu
- **HongTu**: SOTA, uses swap memory when facing host OOM

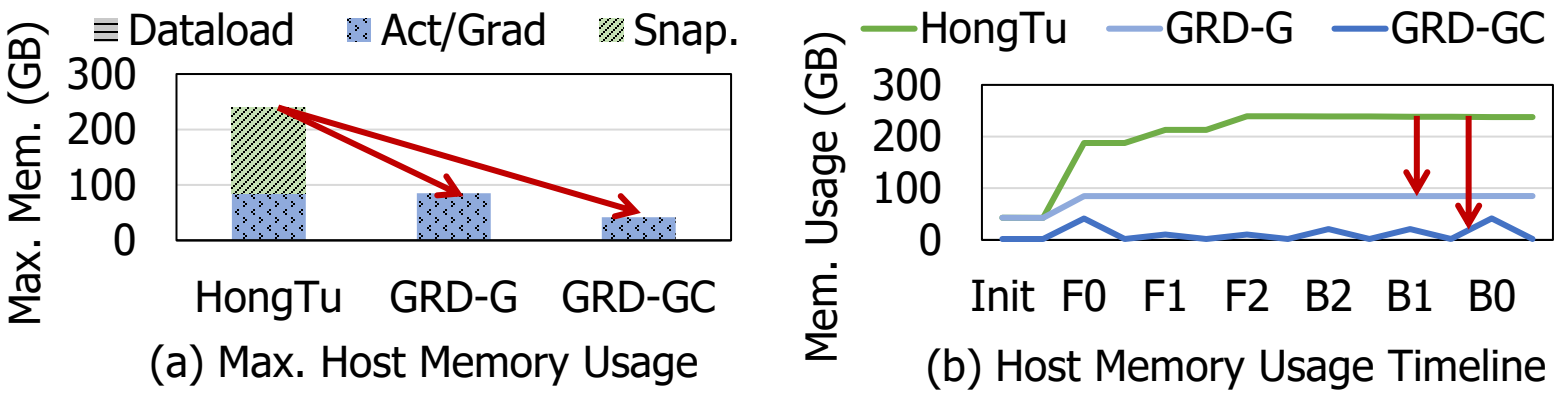
Main Speedup

3-Layer GCN (Time/epoch (min))		Products	IGBM	Papers
Single GPU	Betty	0.61	28.71	GPU OOM
	Ginex	9.00	GPU OOM	17.72
	HongTu	0.17	6.46	Host OOM
Distributed	CAGNET	0.21	1.41	10.01*
	SANCUS+	0.19	0.77*	GPU OOM
Ours (Single GPU)	**GriNNder**	**0.12**	**0.93**	**9.07**

**: Activation recomputation to avoid OOM, +: Non-exact full-graph (with staleness)*

- In 3-layer GCN, up to 6.95x speedup over SOTA (HongTu)
- Comparable to distributed baselines, which suffer from slow inter-server communication (>80%, 10G Infiniband)
- Does not suffer from GPU/host memory capacity wall

Host Memory Usage



GRD-G: GriNNder only with the new gradient engine

GRD-GC: GRD-G + layer-wise caching

(imposed an explicit cache cap of one layer's act/grads for demonstration)

- GriNNder's snapshot skipping significantly reduces the host memory usage
- Layer-/partition-wise caching can further cut the host memory usage
- Compared to HongTu, the host memory usage timeline is stable and under the host memory capacity

Conclusion

1. **Breaks the GPU/host memory capacity wall** of full-graph GNN training with storage offloading

2. Enables previously infeasible large-scale (100M+) full-graph GNN training even with a single GPU

3. Provides up to **9.78x speedup** over SOTA

4. Even comparable throughput to **distributed systems**

Thanks!

Jaeyong Song @ Seoul National University

Email: jaeyong.song@snu.ac.kr

Page: https://aisys.snu.ac.kr/members/JaeyongSong.html

: GriNNder additionally provides a lightweight partitioner for full-graph GNN training on limited environments. To see the details of that partitioner, please see the paper.

Project GitHub