Efficient Training of Graph Neural Networks on Large Graphs

Yanyan Shen, Lei Chen, Jingzhi Fang, Xin Zhang, Shihong Gao, Hongbo Yin

Shanghai Jiao Tong University
Hong Kong University of Science and Technology
Hong Kong University of Science and Technology (Guangzhou)

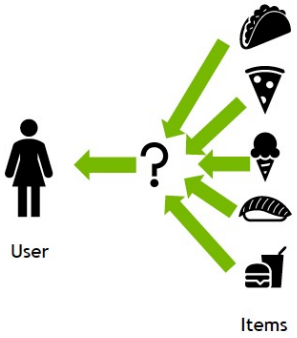
VLDB 2024 Tutorial

Tutorial Outline

- **Graph Neural Network Training Overview (15 min)**
  - Introduction to graph neural network
  - GNN training workflow
  - Challenges and opportunities
- **Data Management for GNN Training (65-70 min)**
  - Graph preprocessing
  - Batch preparation
  - Data transfer
  - Model computation
  - Training Temporal GNN
- **Future Research Directions (5-10 min)**

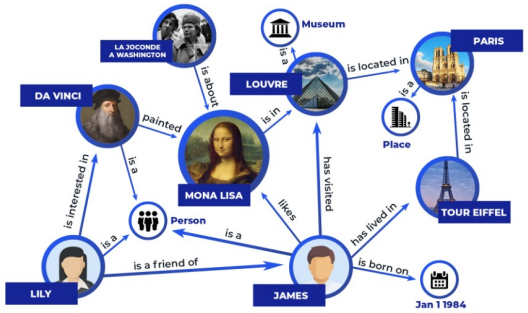
Graphs: Modeling Relationships

- **Graphs are foundation of many real-world applications**

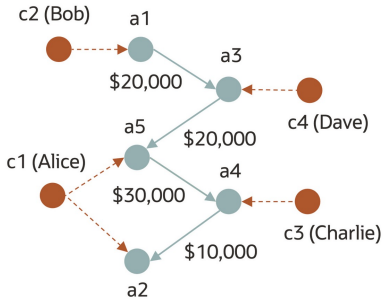


User-item Interaction

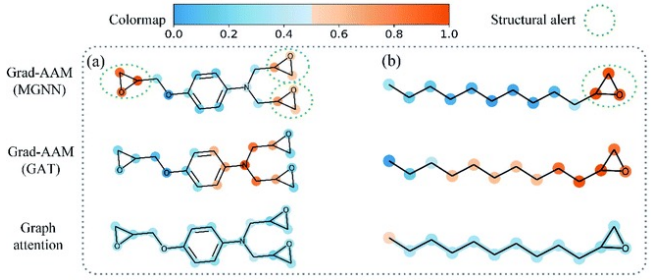
Social Network



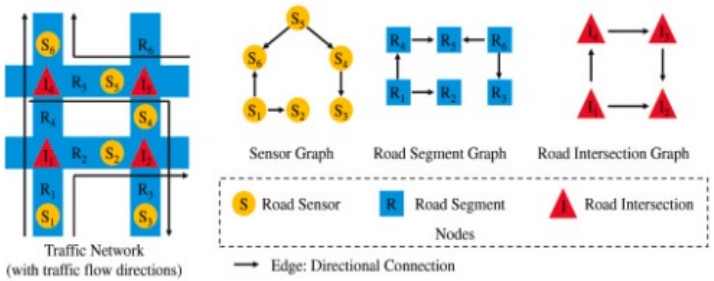
Knowledge Graph



Bank Transaction



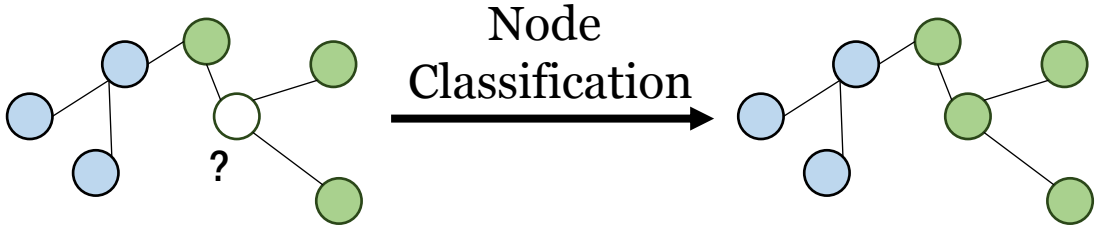
Molecule

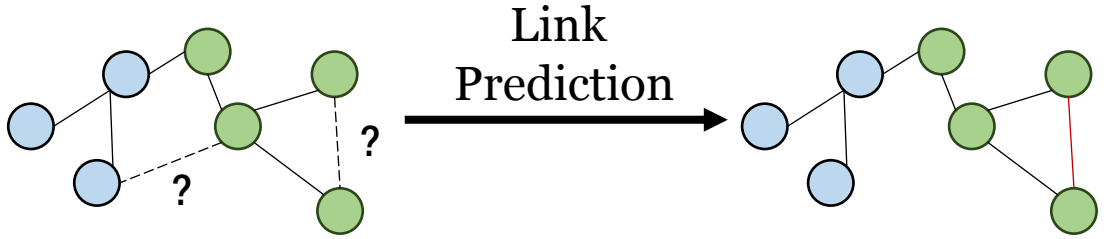


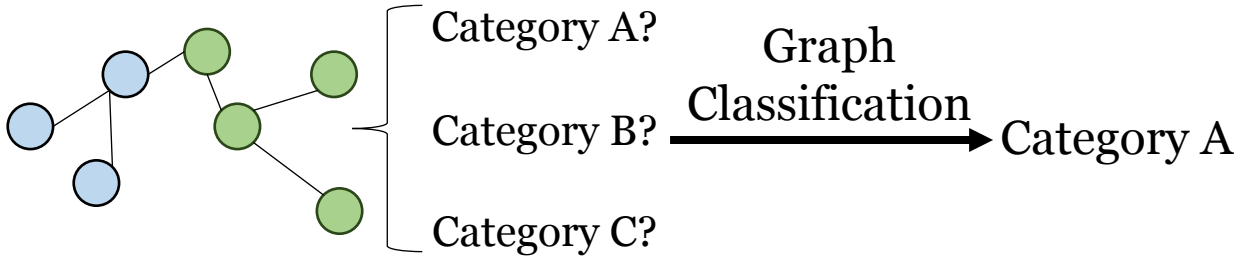
Traffic Network

Predictive Analytics Tasks on Graphs

- **Node-level tasks**
  - Node classification
  - Node regression
  - …
- **Edge-level tasks**
  - Link prediction
  - Edge classification
  - …
- **Graph-level tasks**
  - Graph classification
  - Graph regression
  - …







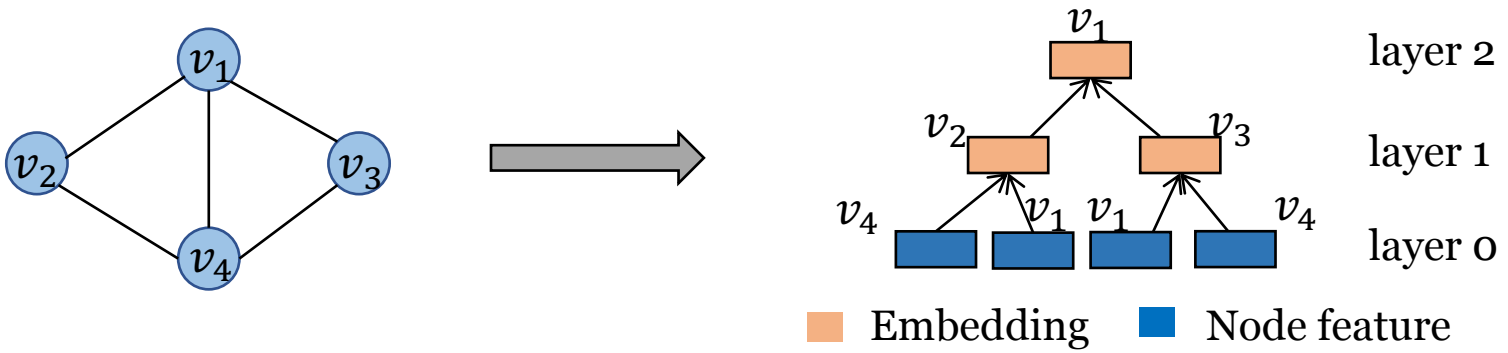
Yu et al. Towards graph foundation models (Tutorial). In TheWebConf, 2024.

Graph Neural Network (GNN) Model

- **Input graph data**
  - A: adjacency matrix X: node feature matrix

- **L-layered GNN via message passing**
  - H: node embedding matrix W: weight matrix σ: activation function
  - H^l : node embedding at layer l

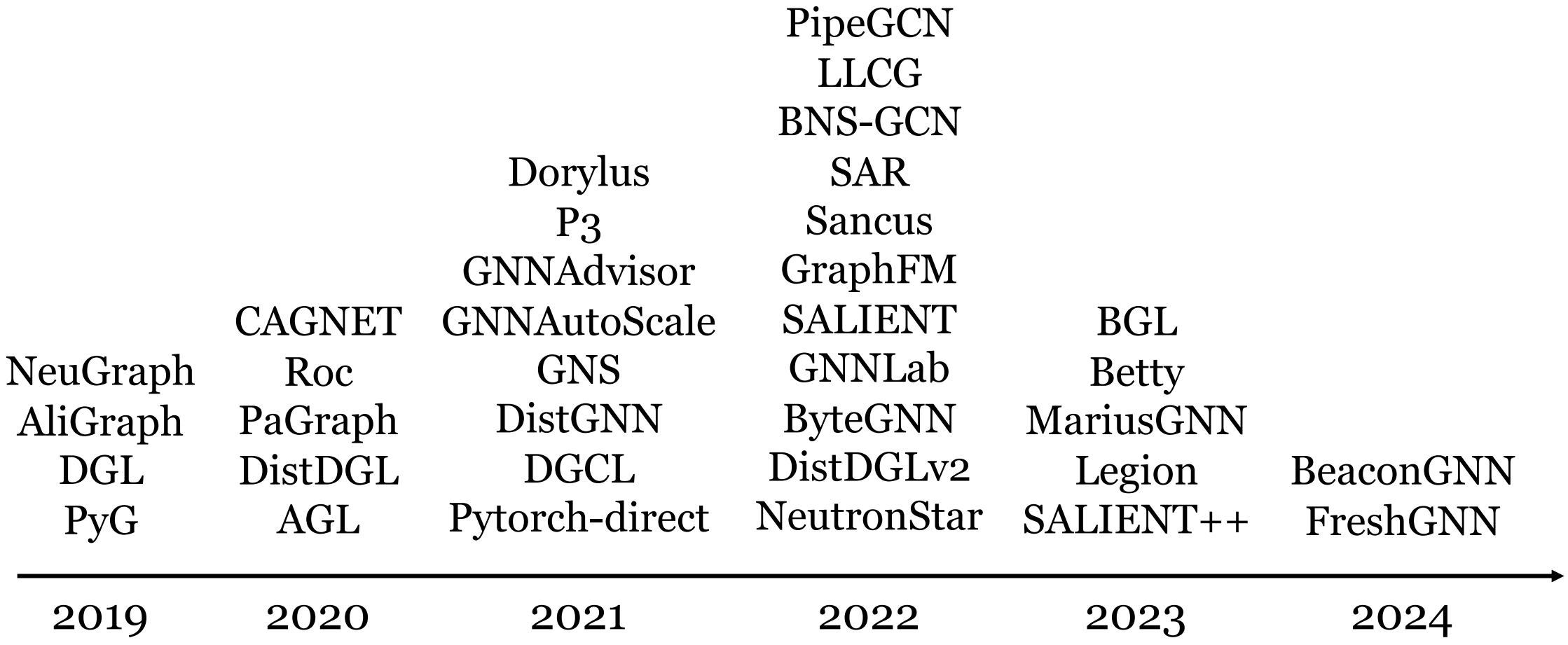
$$\boldsymbol{Z^l = AH^{l-1}W^l, H^l = \sigma(Z^l)}$$



Graph Computing v_1 embedding in a 2-layer GNN

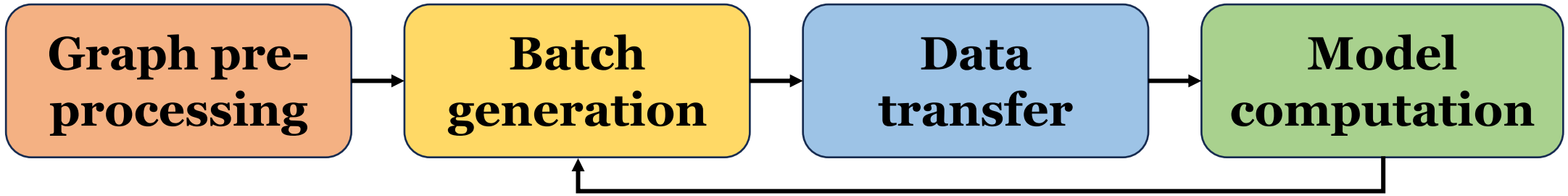
GNN Training Systems at a Glance

- **The evolution of various GNN training systems**



Yuan et al. Comprehensive evaluation of GNN training systems: a data management perspective. PVLDB, 2024.

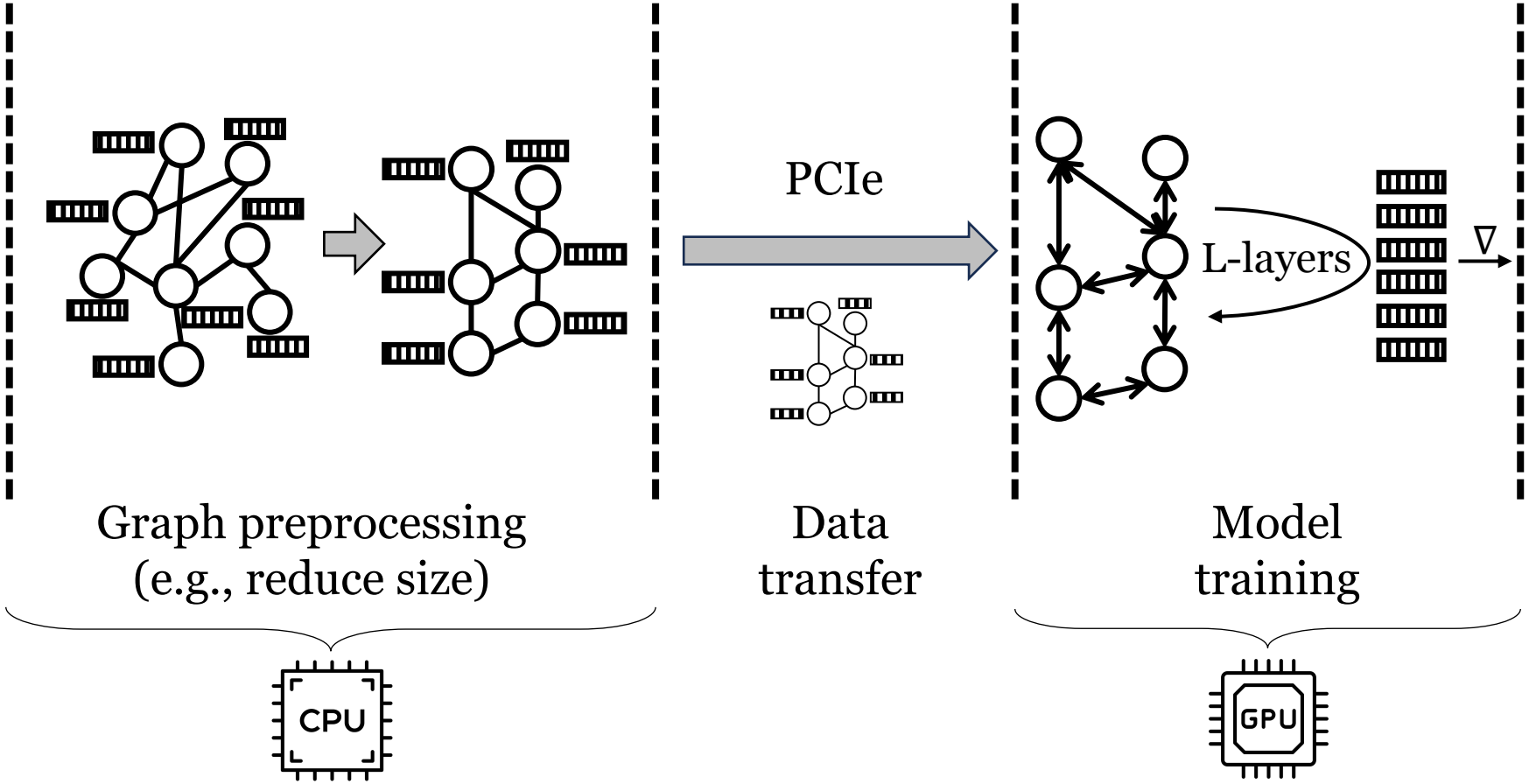
Four Major Stages in GNN Training



- **Graph preprocessing**
  - Modify the input graph before training
- **Batch generation**
  - Generate training batches when performing mini-batch training
- **Data transfer**
  - Transfer graph data / intermediate embeddings over CPU-GPU
- **Model computation**
  - Optimize GNN parameters based on transferred data

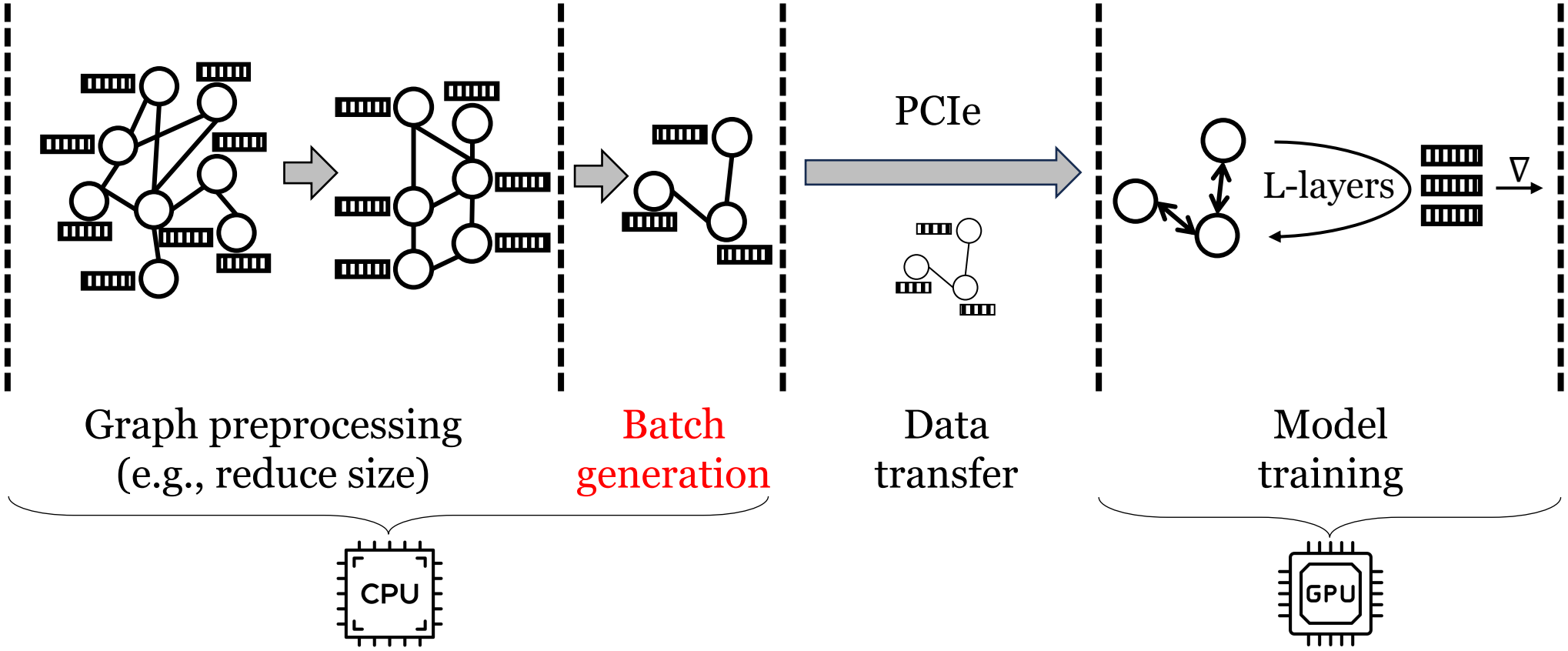
Full-graph Training on Single-GPU

- **Put graph data and model parameters in single-GPU memory**



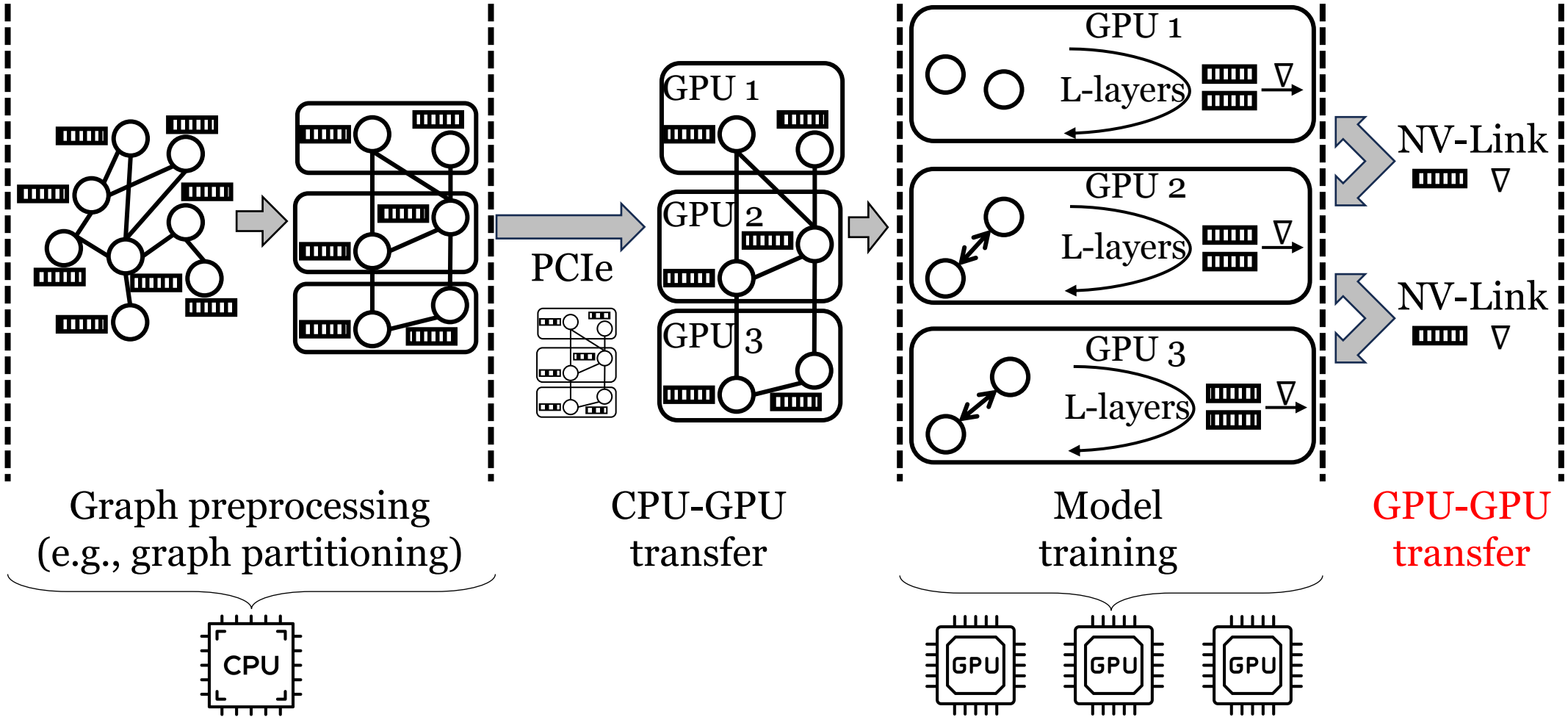
Mini-batch Training on Single GPU

- **Generate mini-batches for GNN training on large graph**
- Assume mini-batch sampling is done on CPU



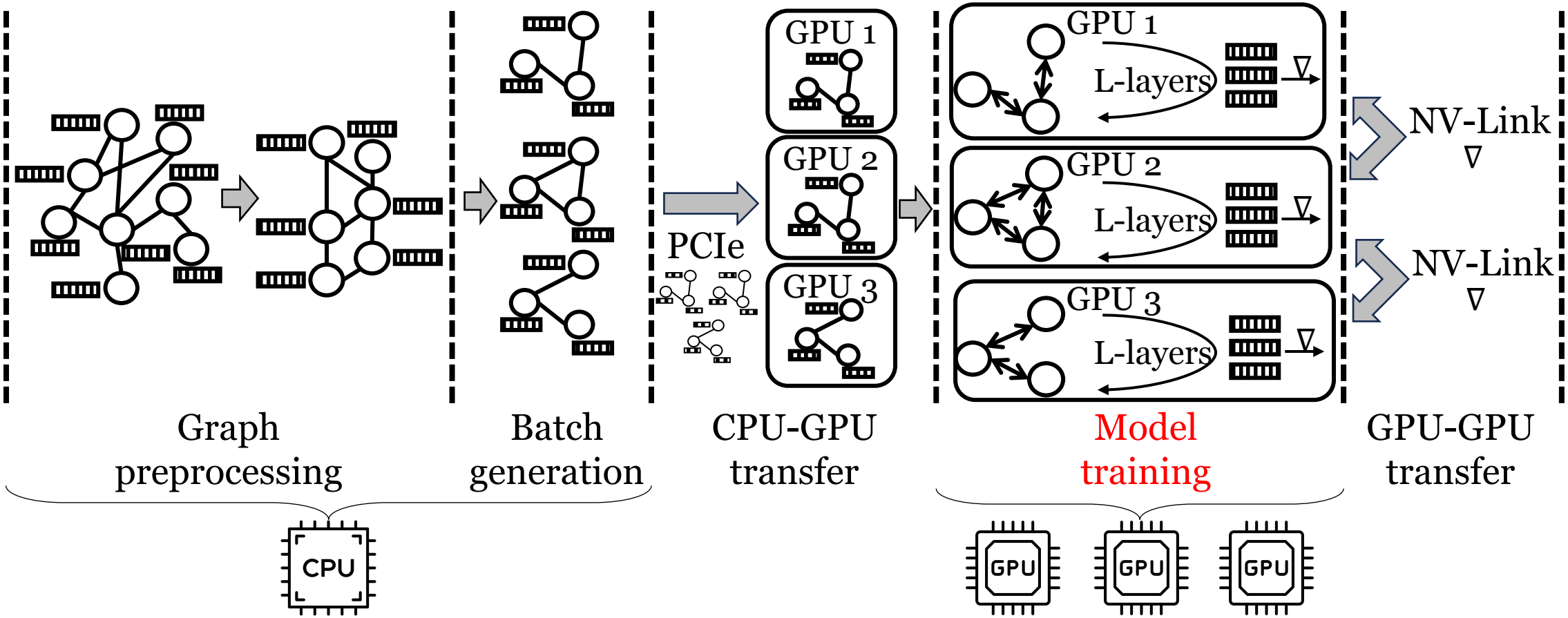
Full-graph Training on Multi-GPUs

- **Distribute graph partitions on multiple GPUs**



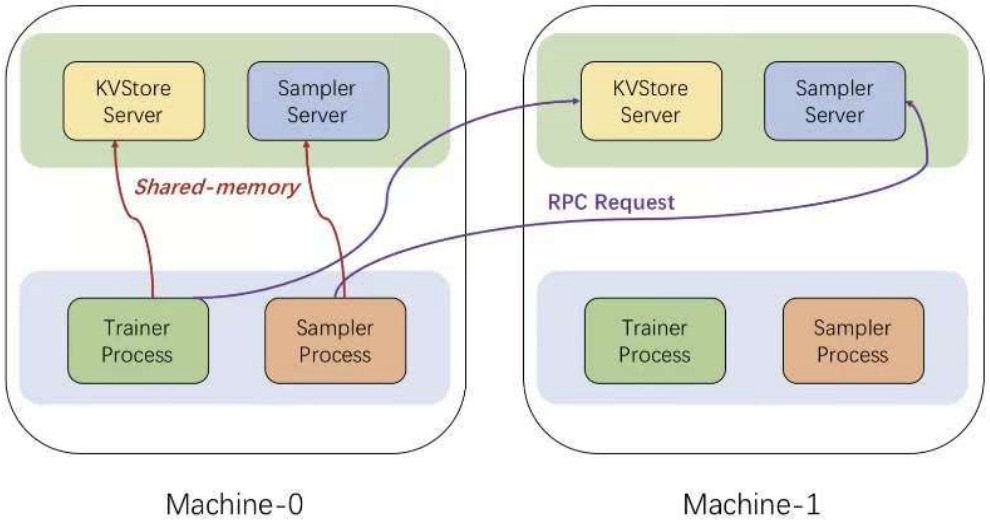
Mini-batch Training on Multi-GPUs

- Perform mini-batch training on multiple GPUs
- Assume CPU performs mini-batch sampling on the input graph

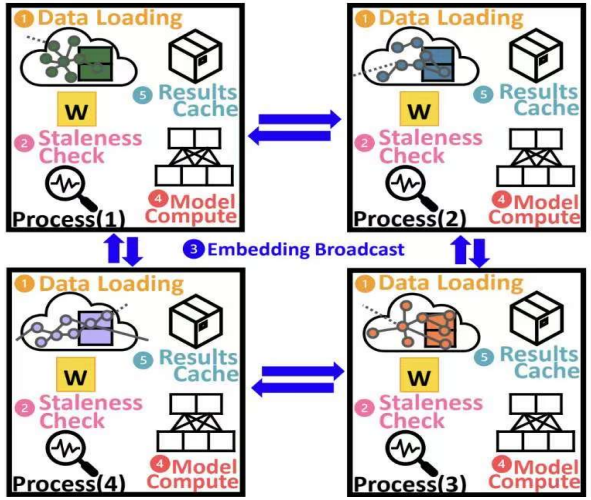


Distributed GNN Training

- CPU-cluster
  - AliGraph, DistDGL, DistGNN, ByteGNN …
- GPU-cluster
  - ROC, P3, DGCL, BNS-GCN, DistDGLv2, NeutronStar, Sancus …



DistDGL

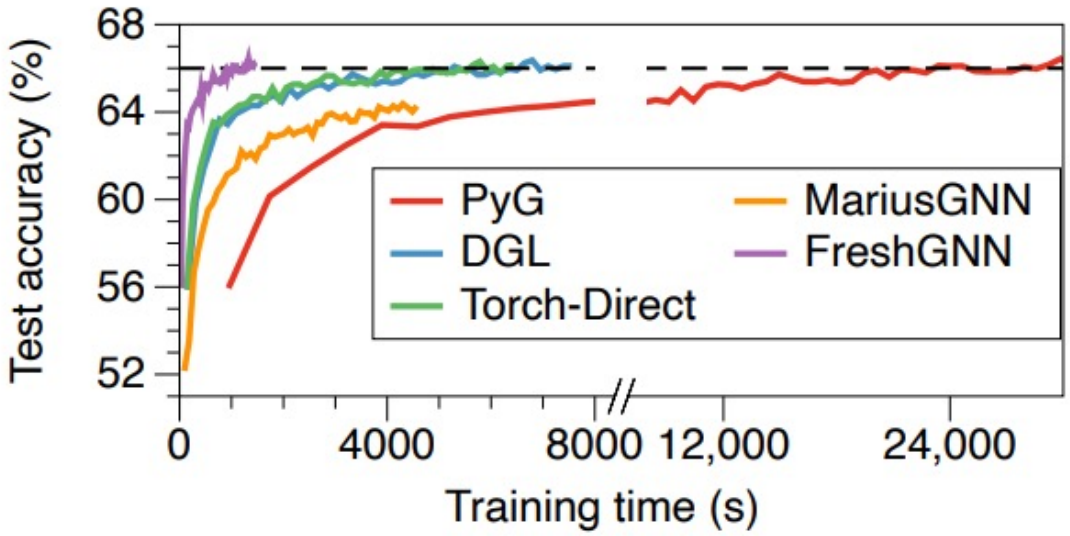


Sancus

Yuan et al. Comprehensive evaluation of gnn training systems: a data management perspective. PVLDB, 2024.

GNN Training Challenge

- **GNN training on large graphs is time-consuming**



>6 hours for PyG

The time-to-accuracy curve of different training systems across 50 epochs of GraphSAGE on ogbn-papers100M (|V| = 111M; |E|=1.6B; input node feature dimension=128) on NVIDIA A100 (40G).

Huang et al. FreshGNN: reducing memory access via stable historical embeddings for graph neural network training. PVLDB, 2024.

GNN Training Opportunities

- GNN training as **Complex Graph Processing on CPU-GPU**

- **Data** in the GNN training pipeline
  - Graph data
  - Batch data
  - Intermediate node embeddings
  - Model parameters and gradients

- **Data management** techniques in four stages
  - Graph preprocessing
  - Batch generation
  - Data transferring
  - Model computation

The Scope of Our Tutorial

- **Two types of graphs**
  - Static graph
  - Dynamic graph
- **Two types of GNN models**
  - GNNs: GraphSAGE, GCN, GAT …
  - Temporal GNNs: TGN …
- **Hardware setup**
  - CPU-GPU hybrid computing

What's Next

- **Efficient GNN Training from Data Management Perspective**
  - Stage 1: graph preprocessing (12 slides) → **Yanyan**
  - Stage 2: Batch preparation (8 slides)
  - Stage 3: Data transfer (19 slides) → **Xin**
  - Stage 4: Model computation (9 slides)
  - Training Temporal GNN (9 slides) → **Jingzhi**
- **Future Research Directions**

Stage 1: Graph Preprocessing

- **Graph Size Reduction**
  - Reduce the number of edges and nodes to reduce training cost

- **Graph Partitioning**
  - Computation balance
  - Per-partition communication balance
  - Global communication reduction

Graph Size Reduction

- **Large graph size is the root cause of GNN training cost**

- **Three ways to reduce graph size**
  - Graph Sparsification: remove unimportant edges
  - Graph Coarsening: group closely related nodes
  - Graph Condensation: synthesize similar but small graph

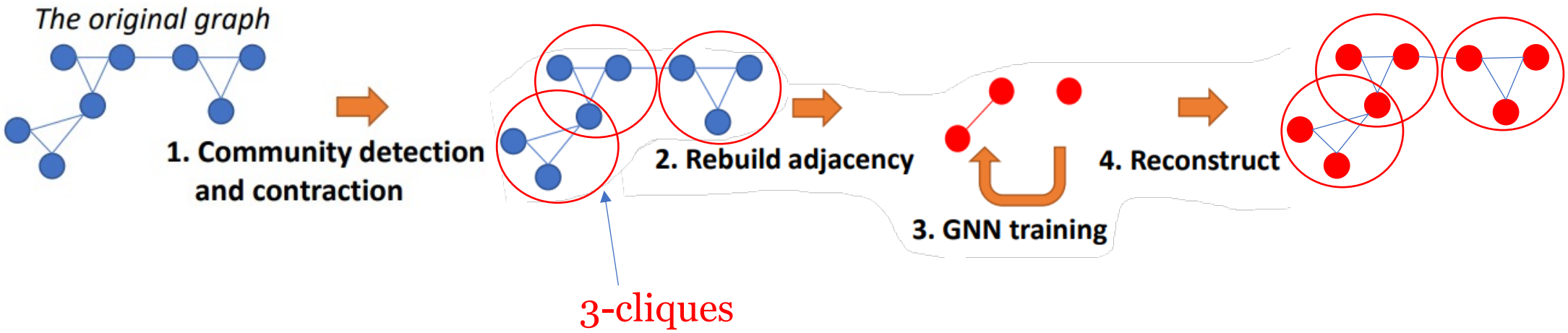
Graph Sparsification

- Compute scores of edges and remove unimportant ones
- E.g., use walk index WI_v^G
  - Consider a graph with two partitions $(\{v\}, V\backslash\{v\})$. Walk index WI_v^G is defined as # of (L-1)-walks from the neighbors of v that end at v
  - Compute walk indices if removing edge e: $\boldsymbol{S}^e = \left(\mathrm{WI}_{v_1}^{G\backslash\{e\}}, \ldots, \mathrm{WI}_{v_n}^{G\backslash\{e\}}\right)$, sorted in an ascending order
  - A large value of $\mathrm{WI}_{v_1}^{G\backslash\{e\}}$ means: removing e will hurt interactions between v's neighbor with v little
  - Remove edges according to $\mathrm{argmax}_{e\in E}\ \boldsymbol{S}^e$ using lexicographical order over $\boldsymbol{S}$

Razin et al. On the ability of graph neural networks to model interactions between vertices. In NeurIPS, 2023.

Graph Coarsening

- Group closely related nodes into super node
- E.g., use community detection
  - Community based on k-cliques (i.e., complete subgraphs with k nodes)
  - Graph contraction based on detected communities
  - Reconstruct node representations after training



Li et al. Cc-gnn: a community and contraction-based graph neural network. In ICDM, 2022.

Graph Condensation: the Basics

- Learn synthetic but smaller graphs
- The performance of GNN trained on the condensed graph should align with that on the original graph

- Graph condensation as a bi-level optimization problem
  - $\mathcal{T}$: original graph $\mathcal{S}$: synthetic graph
  - $\mathcal{L}$: training loss θ: GNN parameter

$$\min_{\mathcal{S}} \mathcal{L}^{\mathcal{T}}\left(\theta^{\mathcal{S}}\right) \text{ s.t. } \theta^{\mathcal{S}} = \arg\min_{\theta} \mathcal{L}^{\mathcal{S}}(\theta).$$

Update the synthetic graph $\mathcal{S}$

Train the GNN on $\mathcal{S}$

Improved Bi-level Optimization

- Naïve method: synthetic graph $\mathcal{S}$ has a large number of parameters to be optimized in the outer-loop
  - $N' \times N'$ adjacency matrix + $N' \times d$ feature matrix in $\mathcal{S}$

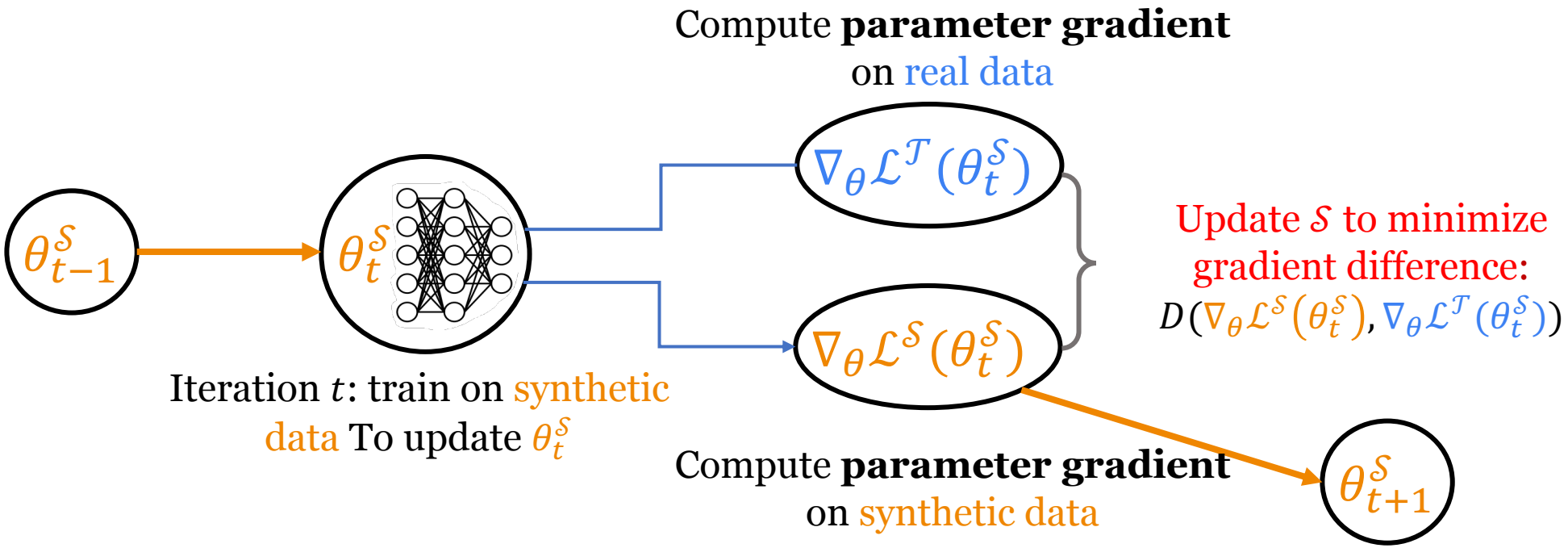
- Improved method: reduce learnable parameters in outer-loop
  - E.g., synthetic adjacency matrix as a function of synthetic features
  - $N' \times d$ feature matrix + Φ in MLP layers

$$\mathbf{A}' = g_\Phi(\mathbf{X}'), \qquad \text{with } \mathbf{A}'_{ij} = \text{Sigmoid}\left(\frac{\text{MLP}_\Phi([\mathbf{x}'_i; \mathbf{x}'_j]) + \text{MLP}_\Phi([\mathbf{x}'_j; \mathbf{x}'_i])}{2}\right)$$

Jin et al. Graph condensation for graph neural networks. In ICLR, 2022.

Bypassing Bi-Level Optimization

- Bi-level optimization is computationally expensive
- Make the training trajectory on the synthetic graph imitate that on the original graph, e.g., via gradient matching



Zhao et al. Dataset condensation with gradient matching. In ICLR, 2021.
Jin et al. Graph condensation for graph neural networks. In ICLR, 2022.

Stage 1: Graph Preprocessing

- **Graph Size Reduction**
  - Reduce number of edges and nodes to reduce training cost

- **Graph Partitioning**
  - Computation balance
  - Per-partition communication balance
  - Global communication reduction

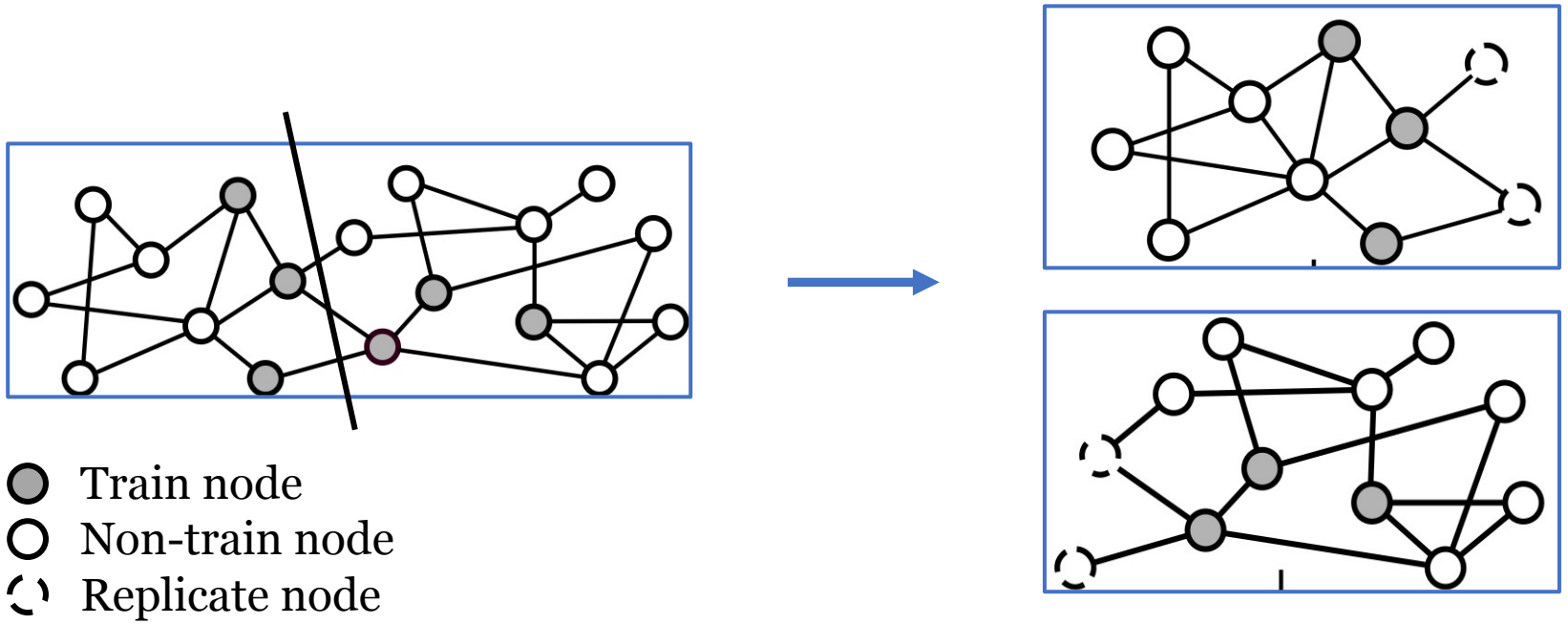
Graph Partitioning

- **Partition the node set and the node features**
  - Co-partitioning
    - Compute a mapping from the node set to the partition set, keeping complete features for each node
  - Independent partitioning
    - Partition the node sets and the node features independently

- **Three concerns**
  - Computation balance
    - Each partition have a similar number of train nodes
  - Per-partition communication balance
    - Each partition have similar communication cost during synchronized training
  - Global communication reduction
    - Reduce total communication in end to end GNN training

Co-Partitioning: Pagraph

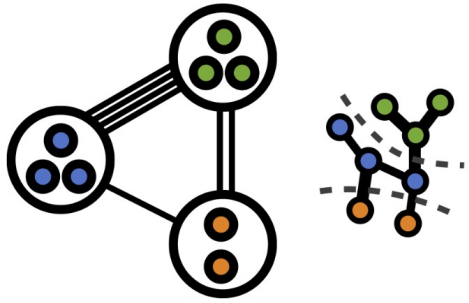
- **Computation balance**
  - Different partitions have similar numbers of target nodes
- **Reduce cross-partition communication**
  - Replicate each train node's *L*-hop reachable neighbors in its partition



Lin et al. Pagraph: scaling gnn training on large graphs via computation-aware caching. In SoCC. 2020.

Co-Partitioning: G3

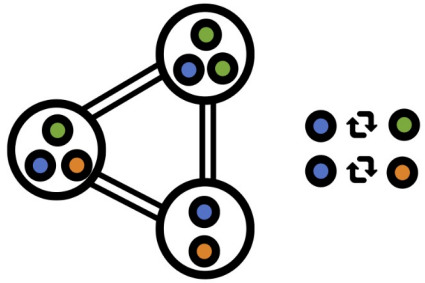
- Stage 1: weighted partitioning
- Stage 2: iterative swapping



Stage 1:
Weighted Partitioning

1) balance per-partition comp. cost

2) minimize global comm. cost



Stage 2:
Iterative Swapping

balance per-partition comm. cost

$$\text{minimize } T_{max\_nn} = \max_{V_p \in P} |V_p|,$$

$$\text{minimize } T_{max\_aggr} = \max_{V_p \in P} \sum_{v \in V_p} d_v,$$

Balance comp.
node number & degree sum

$$\text{minimize } T_{total\_comm} = \sum_{V_p \in P} |V_p^{remote}|$$

Min global comm.
edge-cut

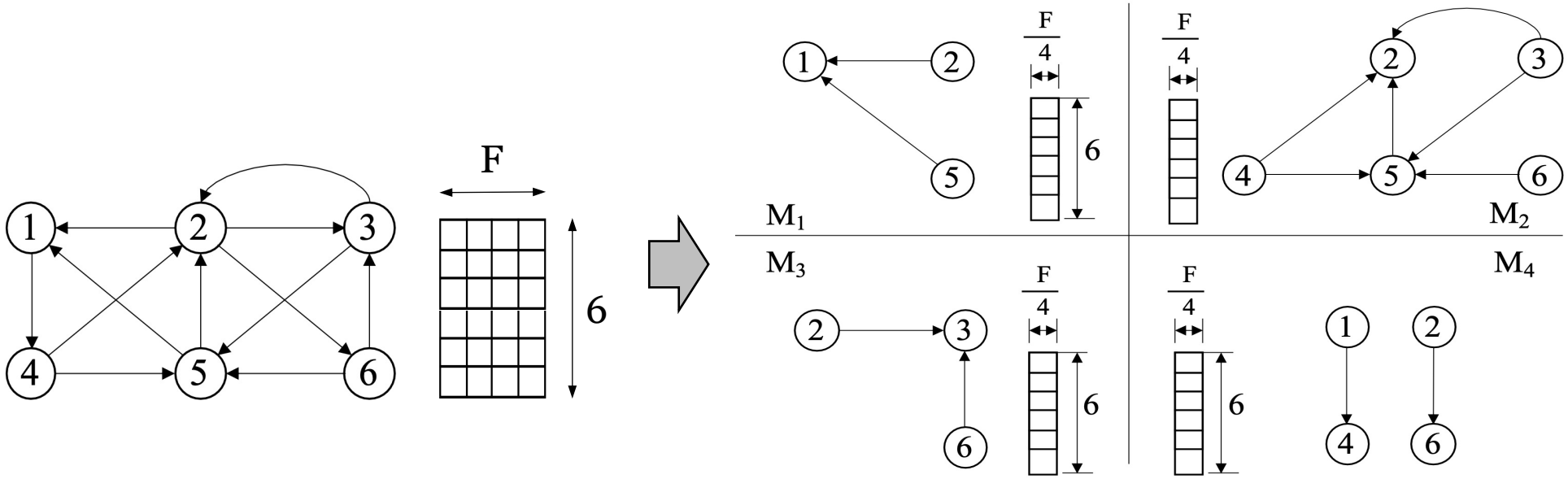
$$\text{minimize } T_{max\_comm.} = \max_{V_p \in P} |V_p^{remote}|$$

Balance comm.
edge-cut

Wan et al. Scalable and efficient full-graph gnn training for large graphs. In SIGMOD, 2023.

Independent Partitioning: P3

- Partition node feature matrix along the feature dimension
- Partition nodes with random partition
- Pros: low partitioning overhead, reduced communication…



E.g., partition a graph with 6 nodes to 4 machines (F is the node feature dimension; each partition has ¼ node features of all 6 nodes)

Gandhi et al. P3: Distributed deep graph learning at scale. In OSDI, 2021.

What's Next: Part II

- **Efficient GNN Training from Data Management Perspective**
  - Stage 1: graph preprocessing (12 slides) → **Yanyan**
  - Stage 2: Batch preparation (8 slides)
  - Stage 3: Data transfer (19 slides)

  → **Xin**
  - Stage 4: Model computation (9 slides)
  - Training Temporal GNN (9 slides)

  → **Jingzhi**
- **Future Research Directions**

Stage 2: Batch Generation

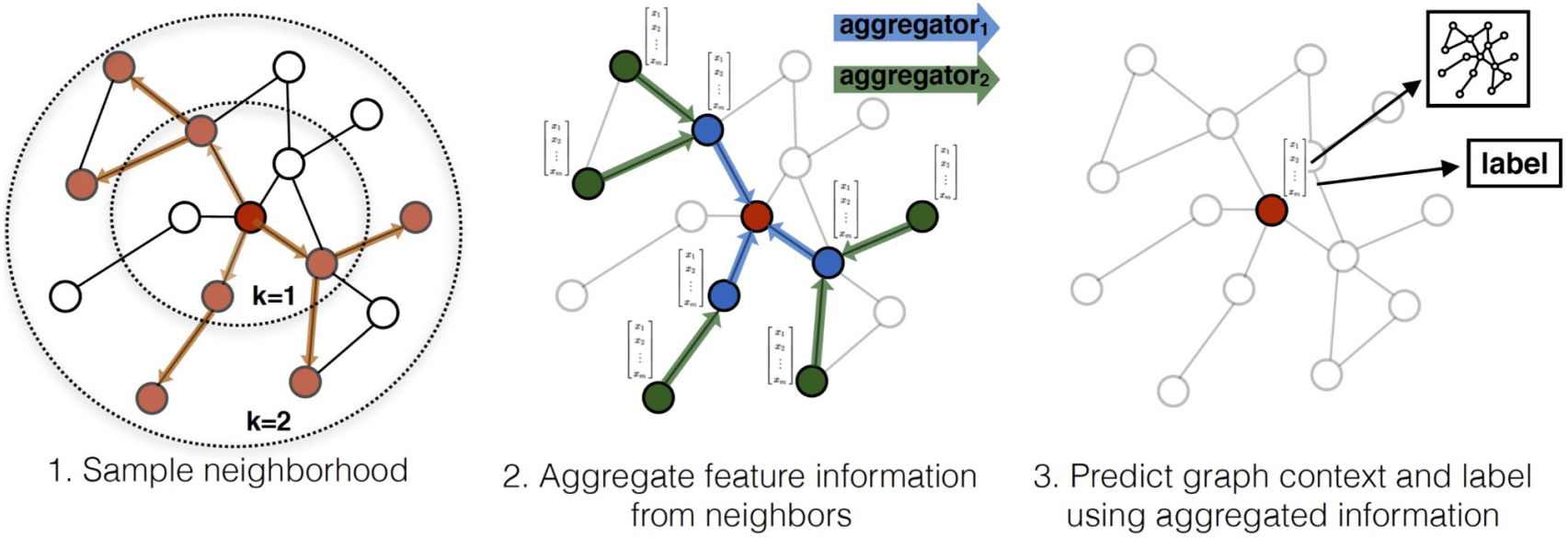
- For large-scale graphs, mini-batch training is useful to reduce computation and memory cost

- **Batch generation involves graph sampling**
  - Node-wise graph sampling
  - Layer-wise graph sampling
  - Subgraph-wise graph sampling
  - Feature-oriented graph sampling

- **Graph sampling on GPUs**
  - Efficient sampling kernel
  - Multi-GPU sampling

Node-wise Graph Sampling

- Start with any target node and recursively sample a node's neighbors based on certain probability
- Target node embedding is computed by aggregating messages from the sampled neighborhood (e.g., GraphSAGE)

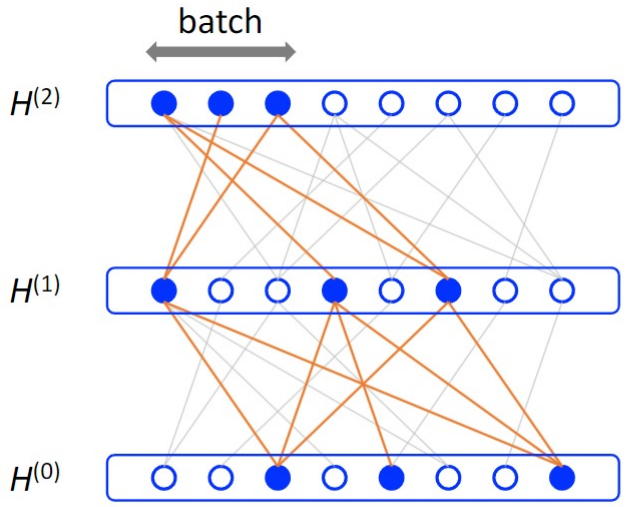


An example of node-wise sampling and the corresponding message passing in GraphSAGE.

Hamilton et al. Inductive representation learning on large graphs. In NIPS, 2017.

Layer-wise Graph Sampling

- Sample a fixed number of nodes in each GNN layer
  - E.g., for each batch, sample 3 nodes in each GNN layer and perform message passing following the associated edges
- Avoid the exponential neighborhood extension in node-wise sampling and reduce the computation cost

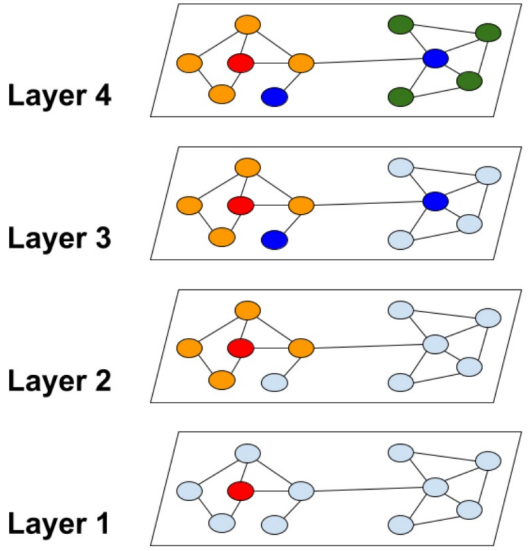


Graph convolution view

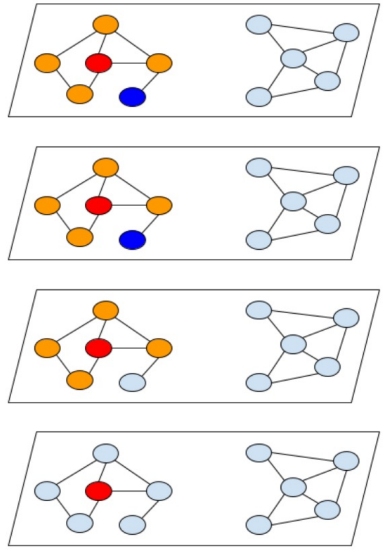
Chen et al. Fastgcn: fast learning with graph convolutional networks via importance sampling. In ICLR, 2018.

Subgraph-wise Graph Sampling

- Sample a subgraph for each batch
  - E.g., use clustering results
- Restrict the message passing within the subgraph
  - Improve node access locality in message passing



Message passing without sampling

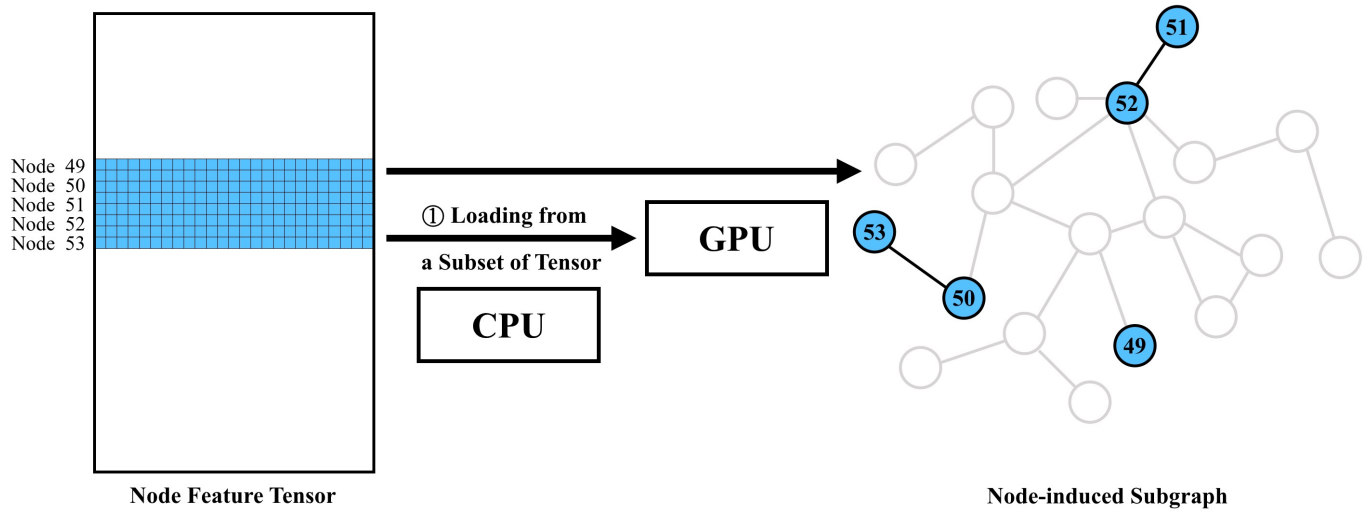


Partition the graph by clustering and do message passing over a subgraph

Chiang et al. Cluster-gcn: an efficient algorithm for training deep and large graph convolutional networks. In SIGKDD, 2019.

Feature-oriented Graph Sampling

- Sample subtensor(s) stored in consecutive memory from the node feature matrix
- Induce subgraph from the sampled nodes
  - Consecutive loading of node features accelerates CPU-based sampling

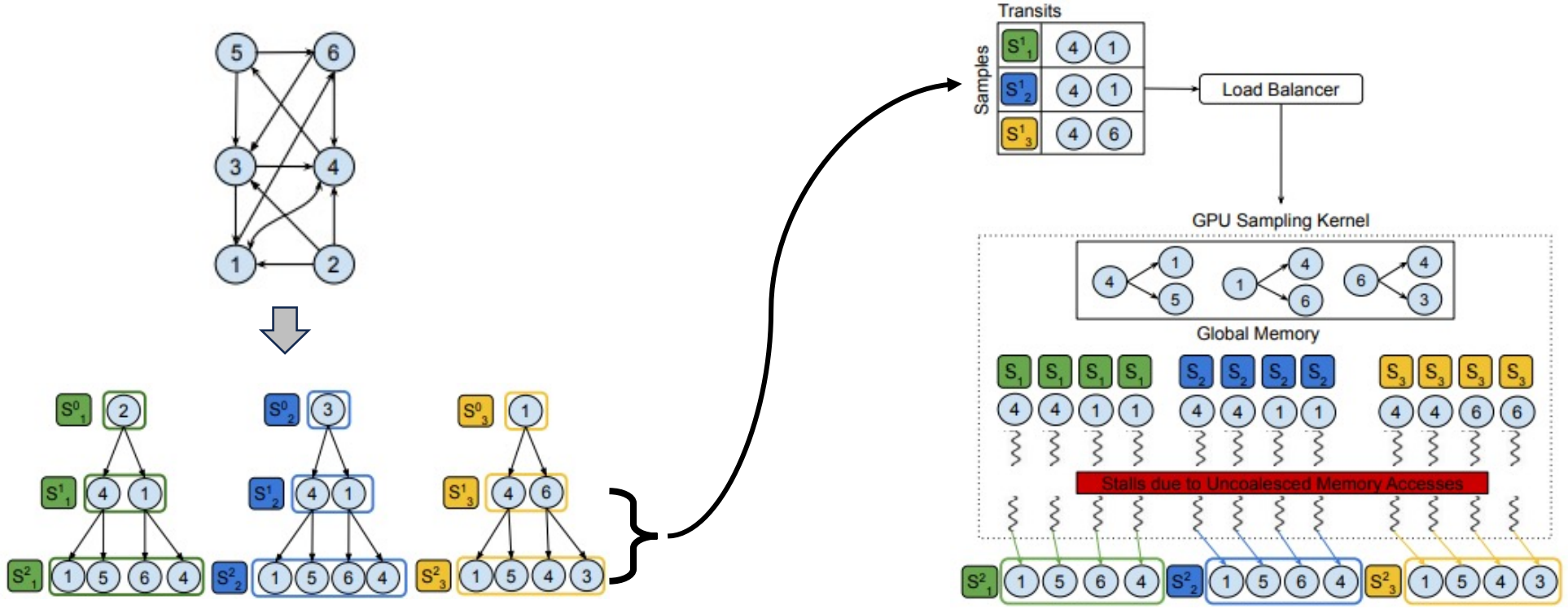


Memory I/O is reduced because data random access is avoided

Zhang et al. Feature-oriented sampling for fast and scalable gnn training. In ICDM, 2022.

Graph Sampling on GPU

- Since GPUs are extremely fast, graph sampling can also be performed on GPUs
  - E.g., make each thread samples a neighbor for a target node



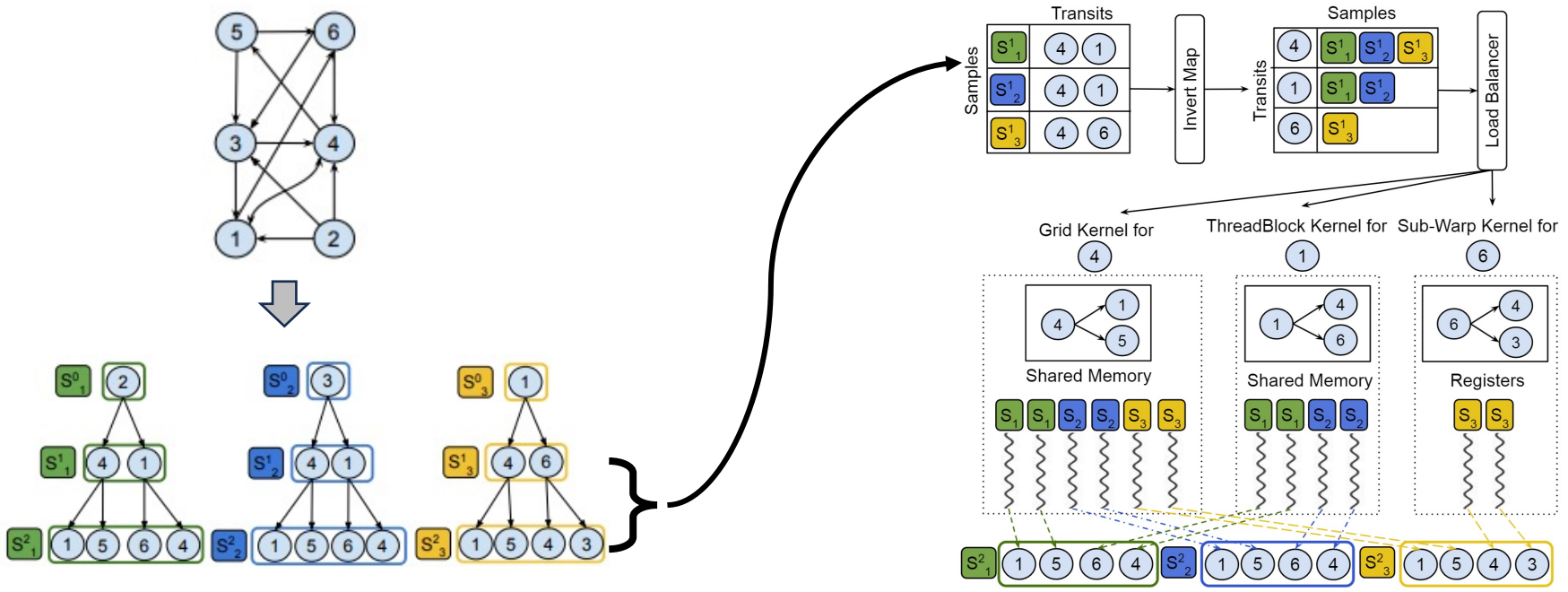
2-hop node-wise sampling

Execute sampling on GPU

Jangda et al. Accelerating graph sampling for graph machine learning using GPUs. In EuroSys, 2021.

Accelerating Graph Sampling on GPU

- Group threads that perform sampling for the same target node together for data reuse
  - E.g., the edge list of the common target node is shared in memory



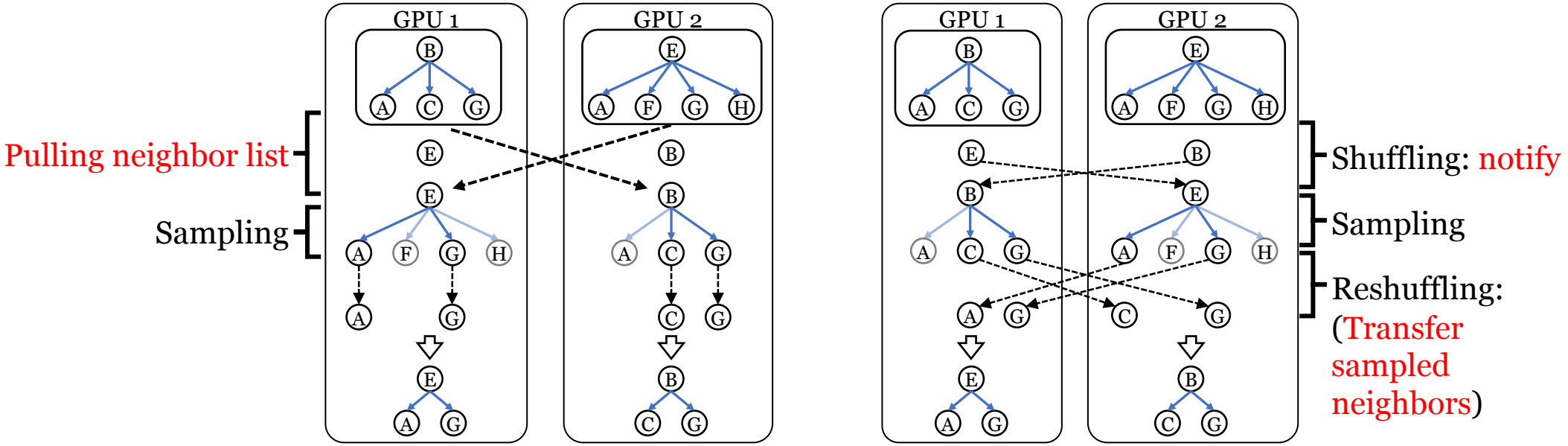
An example of 2-hop node-wise sampling.

Execute the second step of sampling on GPU

Jangda et al. Accelerating graph sampling for graph machine learning using GPUs. In EuroSys, 2021.

Multi-GPU Sampling

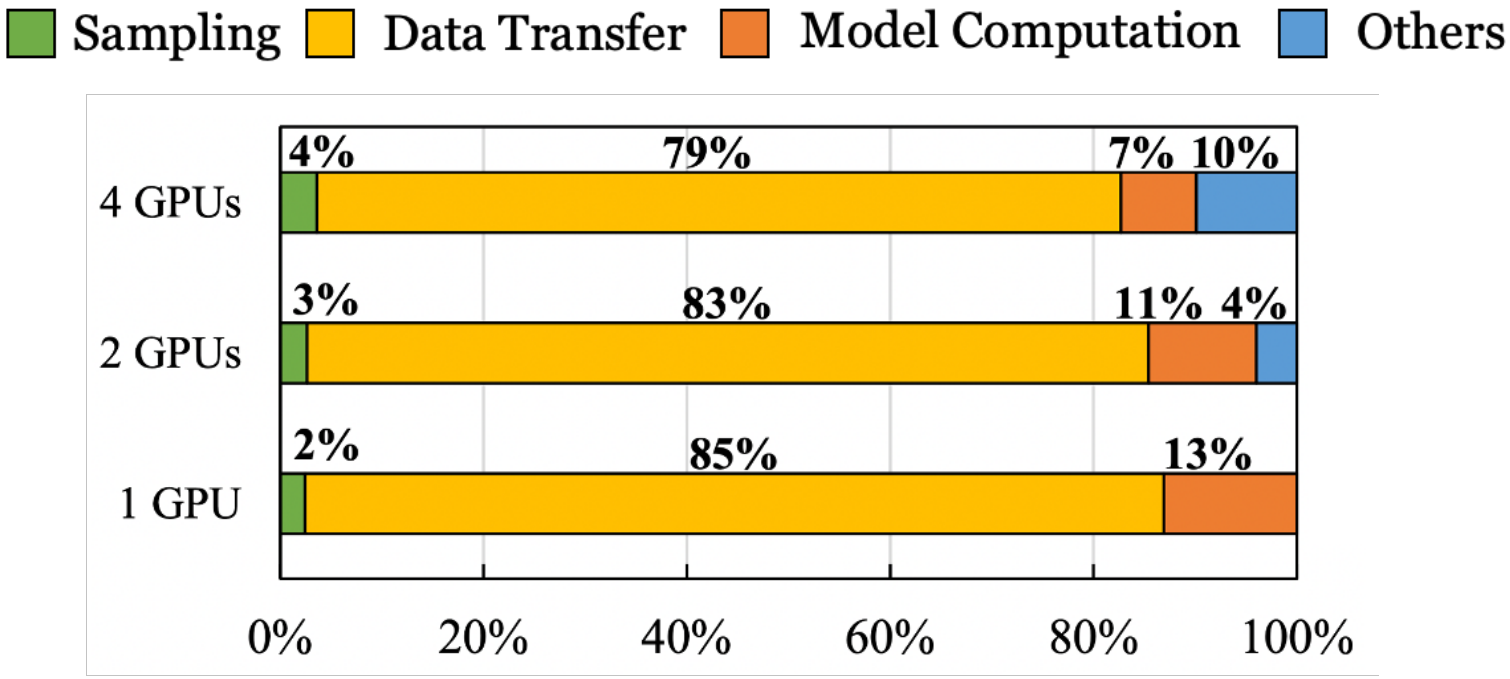
- Partition graph over GPUs
- Each GPU performs node-wise sampling in either way:
  - 1) Pull nodes' edge lists from remote GPUs and do sampling locally, Or
  - 2) Request remote GPUs to do sampling on their containing nodes and send sampling results to the requested GPU



Cai et al. DSP: efficient gnn training with multiple gpus. In PPoPP, 2023.

Stage 3: Data Transfer

- High data transfer cost when training GNN at scale
- Training time profiling

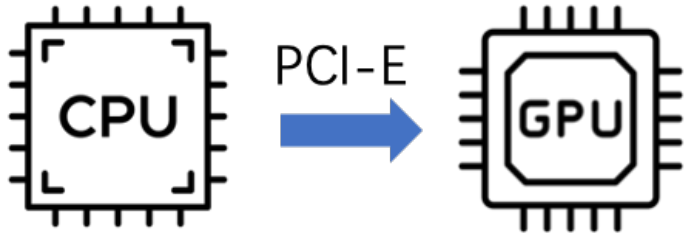


Training 2-layer GCN on Reddit Dataset

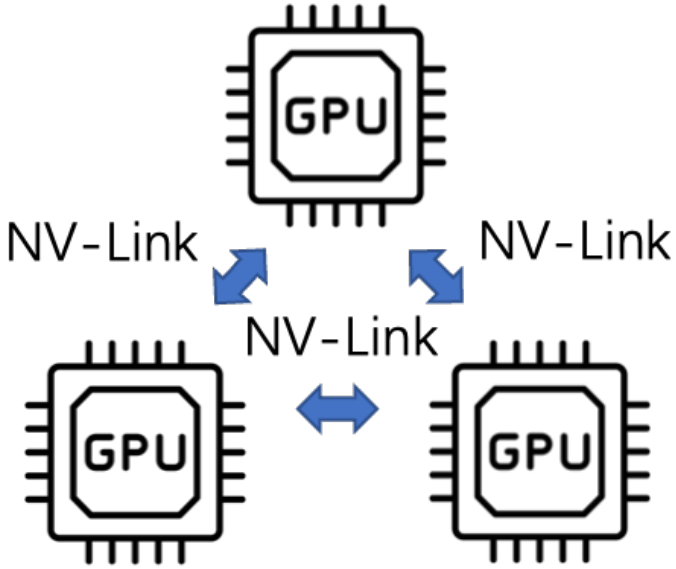
Lin et al. Characterizing and understanding distributed gnn training on gpus. CAL, 2022.

Transfer on CPU-GPU Architecture

- Transfer graph data from CPU to GPU(s)



- Transfer intermediate data among GPUs



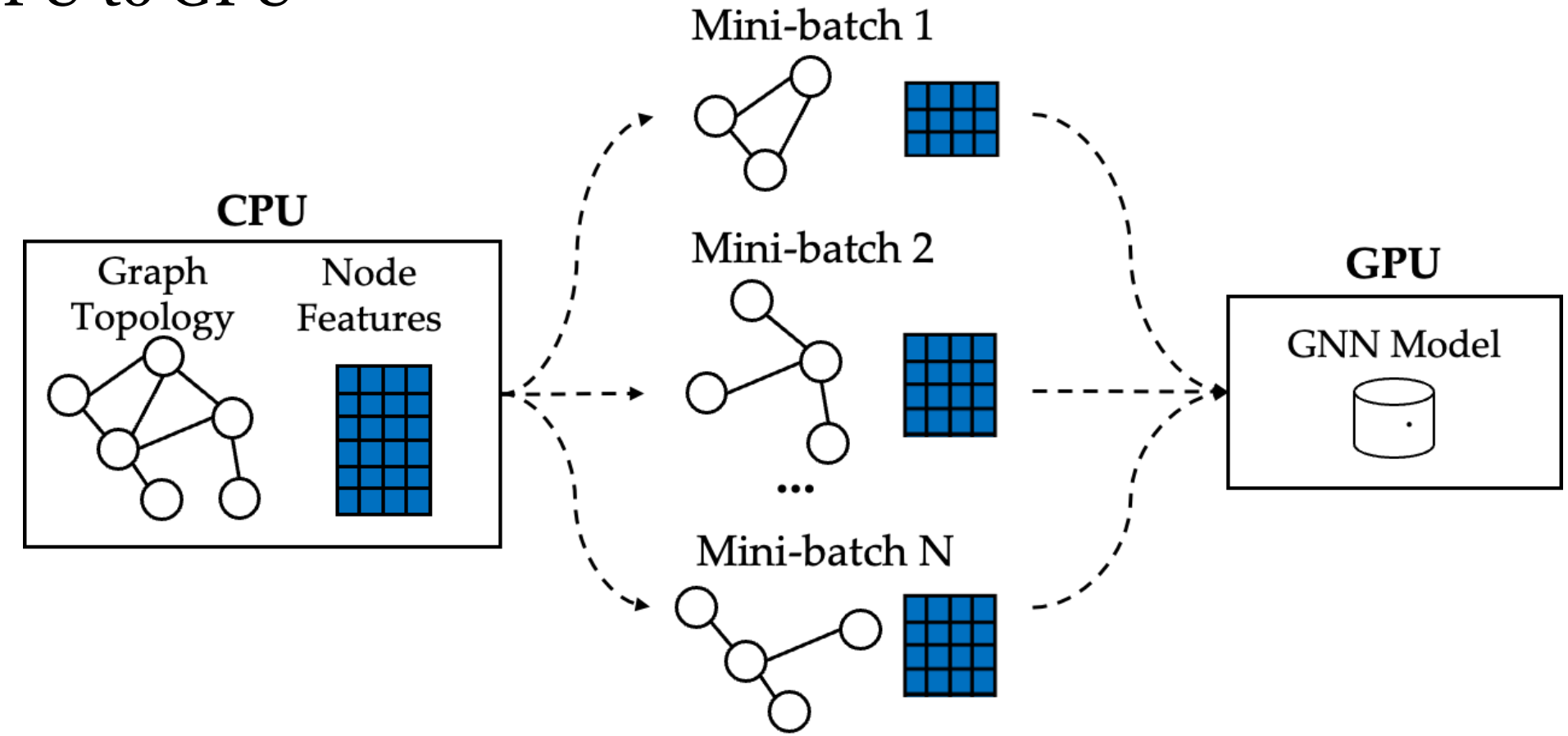
Optimized Data Transfer

- **Optimize graph data transfer: caching is the key**
  - Degree-based caching (for node features)
  - Pre-sampling-based caching (for node features)
  - Hybrid caching (for node features and adjacency matrix)

- **Optimize immediate data transfer**
  - Reduce inter-GPU transfer frequency
  - Reduce inter-GPU transfer volume
  - Select inter-GPU transfer route

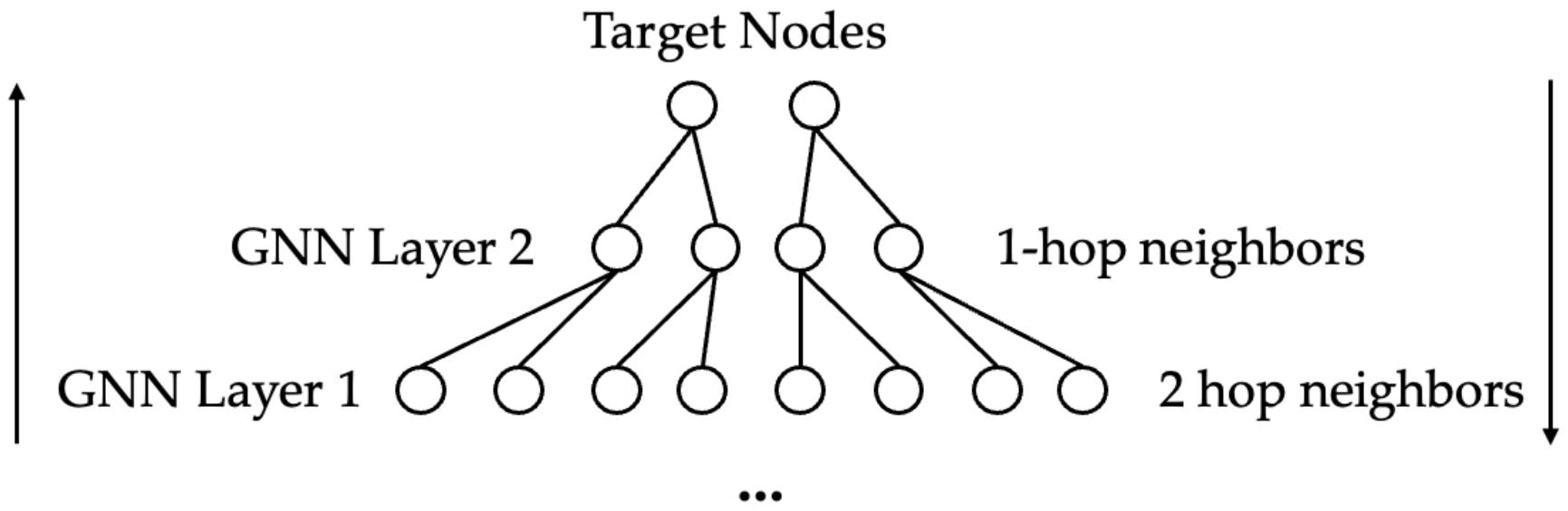
Recall: Transfer of Graph Data

- For mini-batch based GNN training, the sampled subgraphs and the corresponding node features are constantly transferred from CPU to GPU



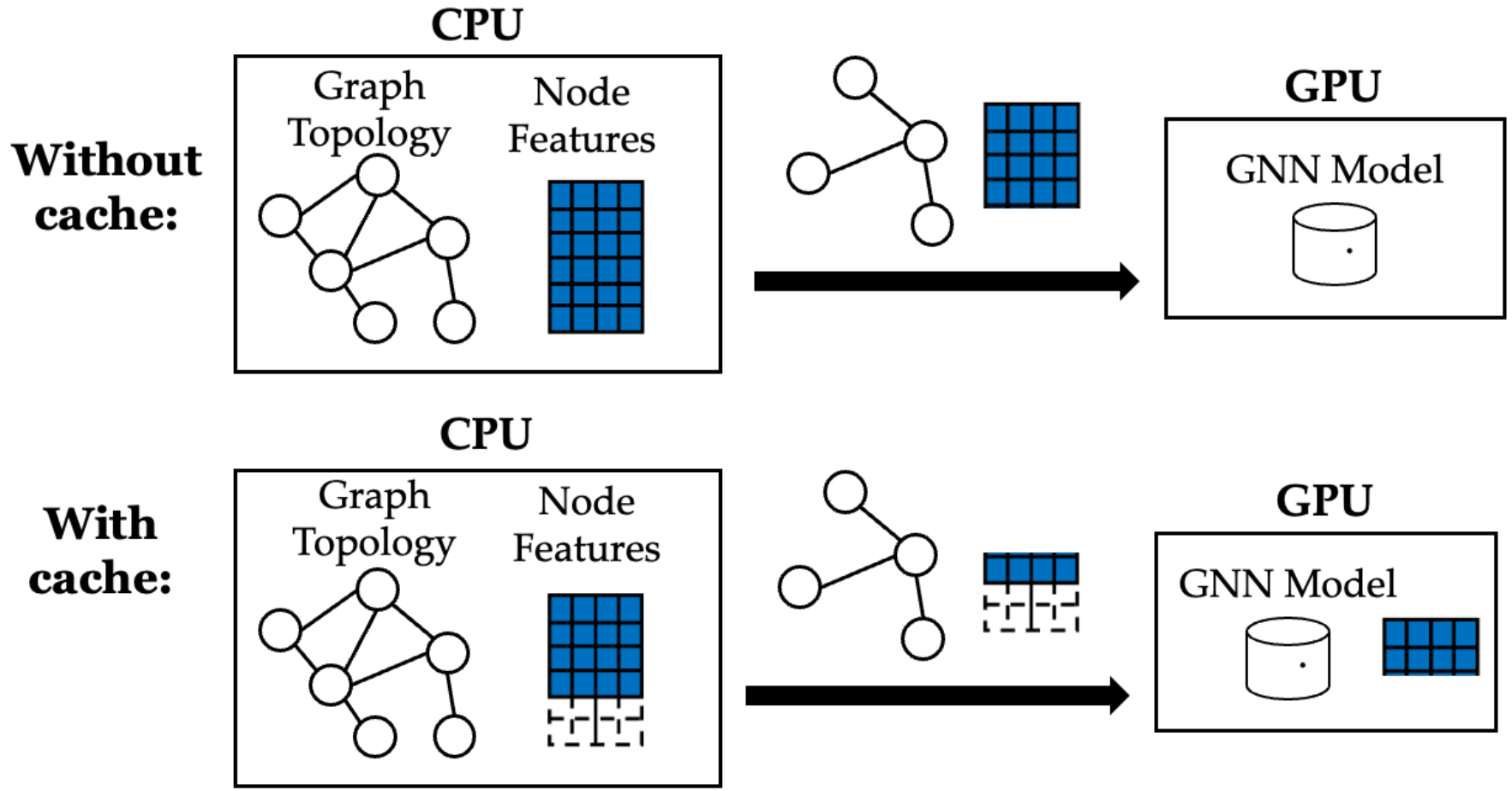
Graph Data Transfer Volume

- Volume of graph data transferred from CPU to GPU per mini-batch may grow exponentially with the depth of the GNN model (i.e., the number of layers)



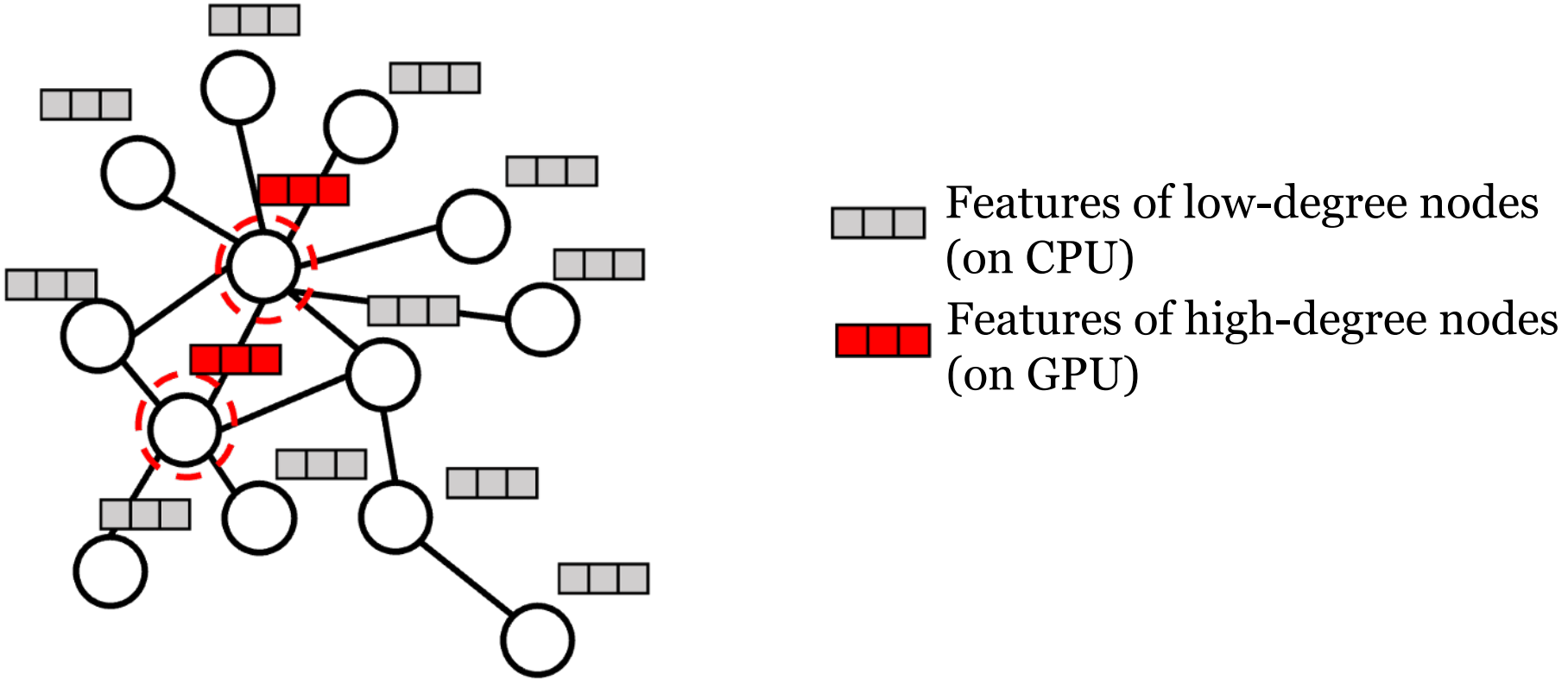
Caching to Reduce Transfer Volume

- Caching some graph data on GPU in advance to directly reduce CPU-GPU data transfer volume



Degree-based Caching

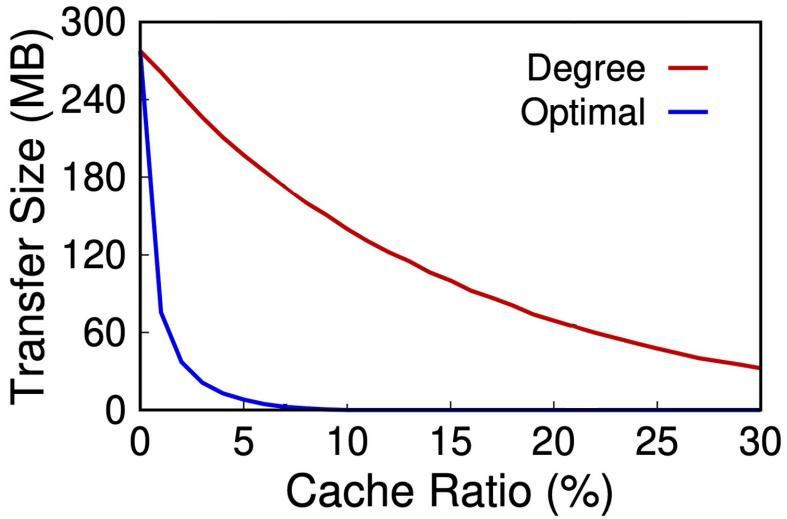
- Caching features of high-degree nodes
  - Sort the graph nodes based on their degrees in descending order
  - Cache as many features of high-degree nodes as possible



Lin et al. Pagraph: scaling gnn training on large graphs via computation-aware caching. In SoCC, 2020.

The Pitfall of Degree-based Caching

- Possible low cache hit ratio due to:
  - May not work well on graphs whose node degree does not follow the power-law distribution
  - Ignore the underlying mini-batch sampling methods

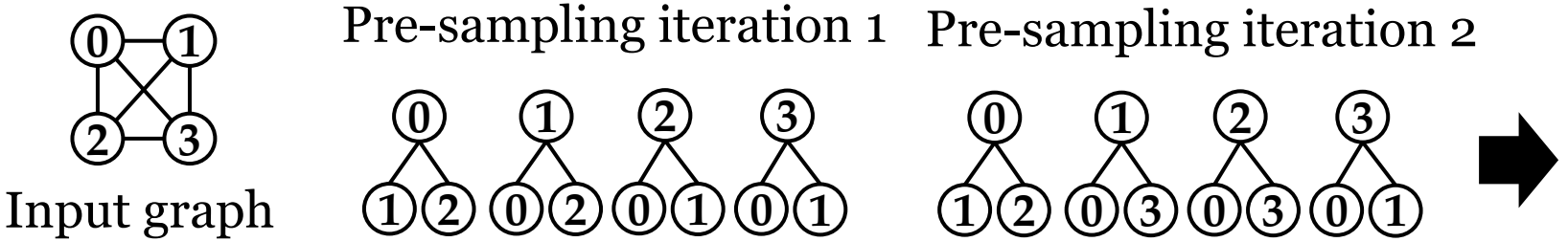


Data transfer size (MB) of degree-based caching versus that of optimal caching under different cache ratios (% of the total GPU memory used as cache)

Yang et al. GNNLab: a factored system for sample-based gnn training over gpus. In EuroSys, 2022.

Pre-sampling-based Caching

- Pre-sampling before training
  - Conduct a few sampling iterations for all graph nodes
  - Bookkeep the total occurrence of each node
- Sort all nodes by their occurrences in a descending order
- Cache features for the nodes with high occurrences



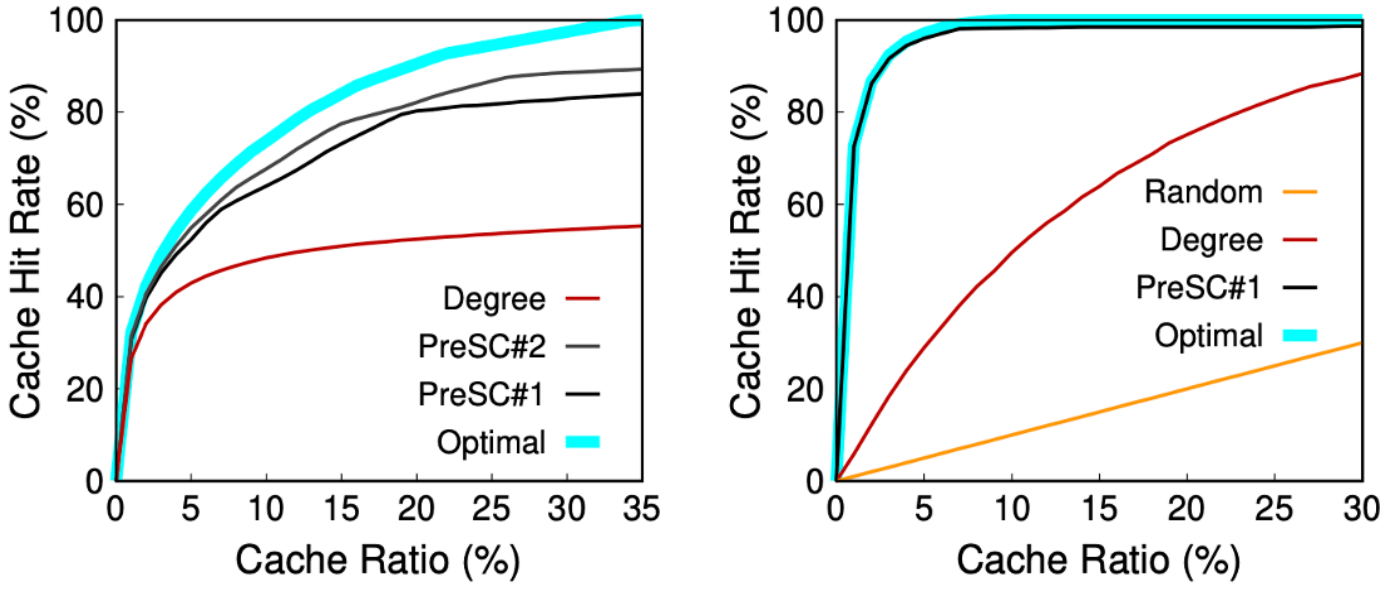
Node	#Occurrence
0	6
1	5
2	3
3	2

Two iterations in pre-sampling;
Sample 2 neighbors for each node in each iteration.

Yang et al. GNNLab: a factored system for sample-based gnn training over gpus. In EuroSys, 2022.

Pre-sampling-based VS Degree-based

- Pre-sampling-based caching presents a higher cache hit rate



(a) (b)

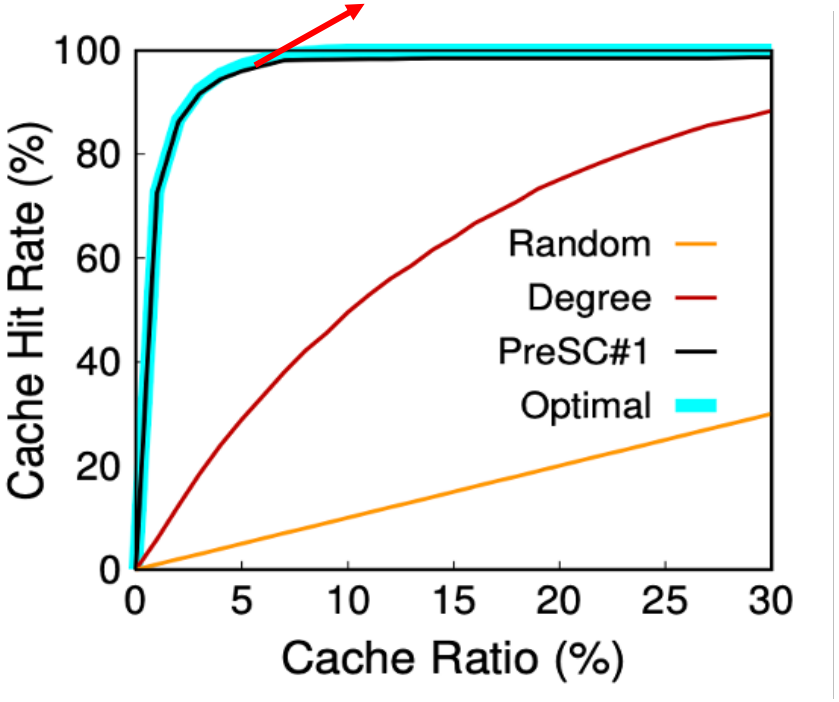
Cache hit rate versus cache ratio (% of the total GPU memory used as cache)
(a) on Twitter with weighted sampling;
(b) on OGB-Papers with 3-hop node-wise sampling

Yang et al. GNNLab: a factored system for sample-based gnn training over gpus. In EuroSys, 2022.

The Pitfall of Single Cache

- Diminishing return of caching node features only

Almost reaches 100% hit rate with a small node feature cache



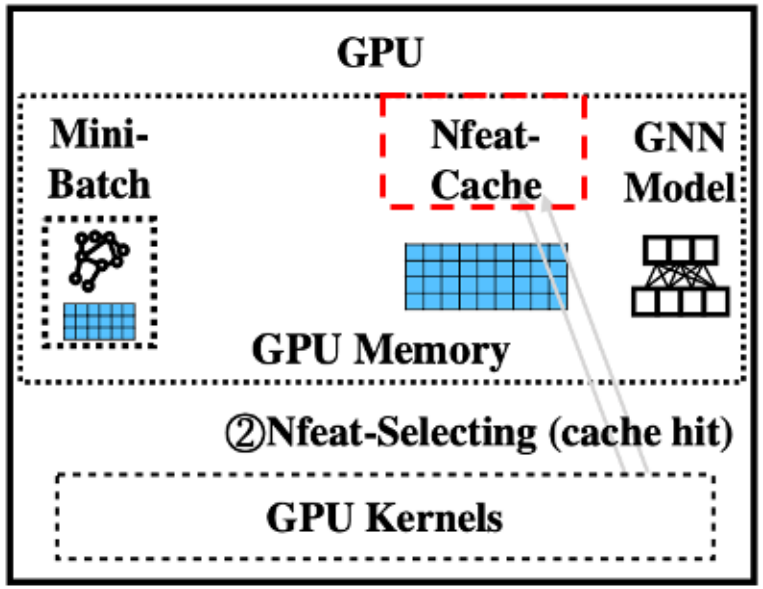
Cache hit rate versus cache ratio on OGB-Papers with 3-hop node-wise sampling

Yang et al. GNNLab: a factored system for sample-based gnn training over gpus. In EuroSys, 2022.

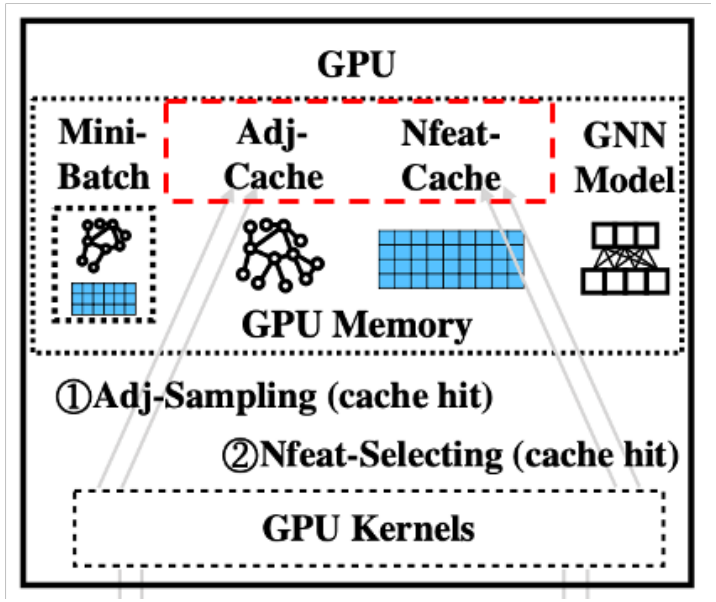
Hybrid Cache

- Cache both node features and graph topology
  - Pre-sampling to estimate the runtime reduction of caching each individual node feature and each individual adjacency list
  - Adaptively allocate two kinds of caches to reduce data transfer cost

Node feature cache only



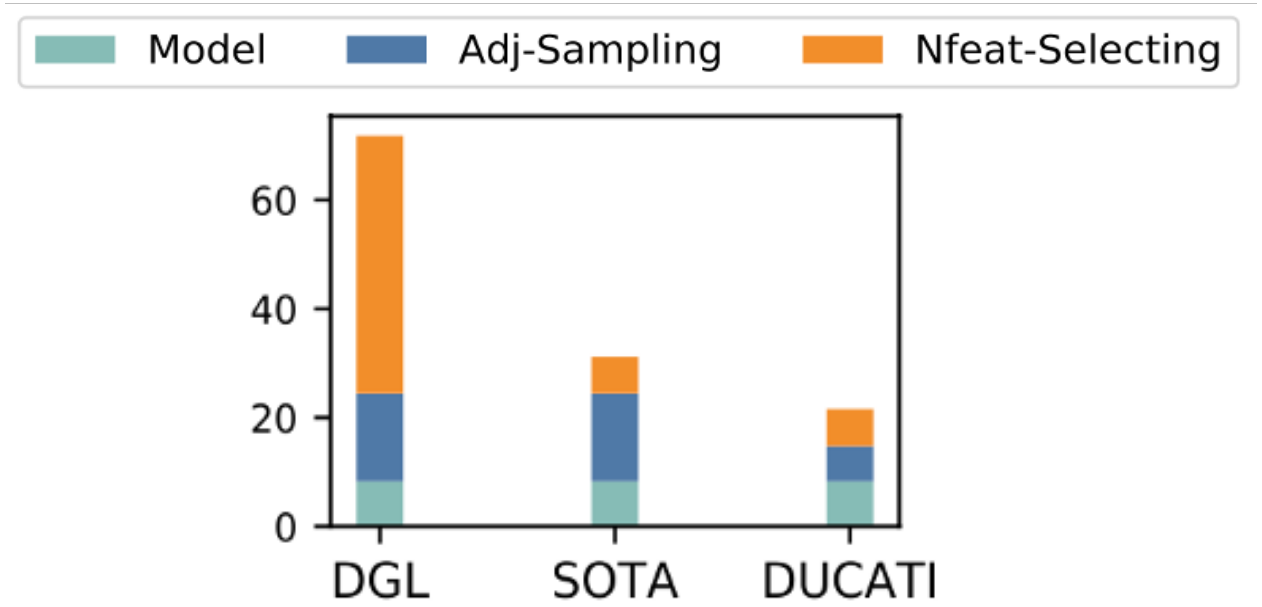
Node feature + graph topology cache



Zhang et al. DUCATI: a dual-cache training system for graph neural networks on giant graphs with the gpu. In SIGMOD, 2023.

Hybrid Cache VS Single Cache

- Hybrid cache achieves a higher reduction in data transfer cost, leading to higher end-to-end training efficiency



Epoch time breakdown of DGL (no cache), SOTA (Nfeat. cache), DUCATI (hybrid cache) on OGB-Papers100M (PA), with sampling fanout (15, 10, 5) and embedding dimension 256

Zhang et al. DUCATI: a dual-cache training system for graph neural networks on giant graphs with the gpu. In SIGMOD, 2023.

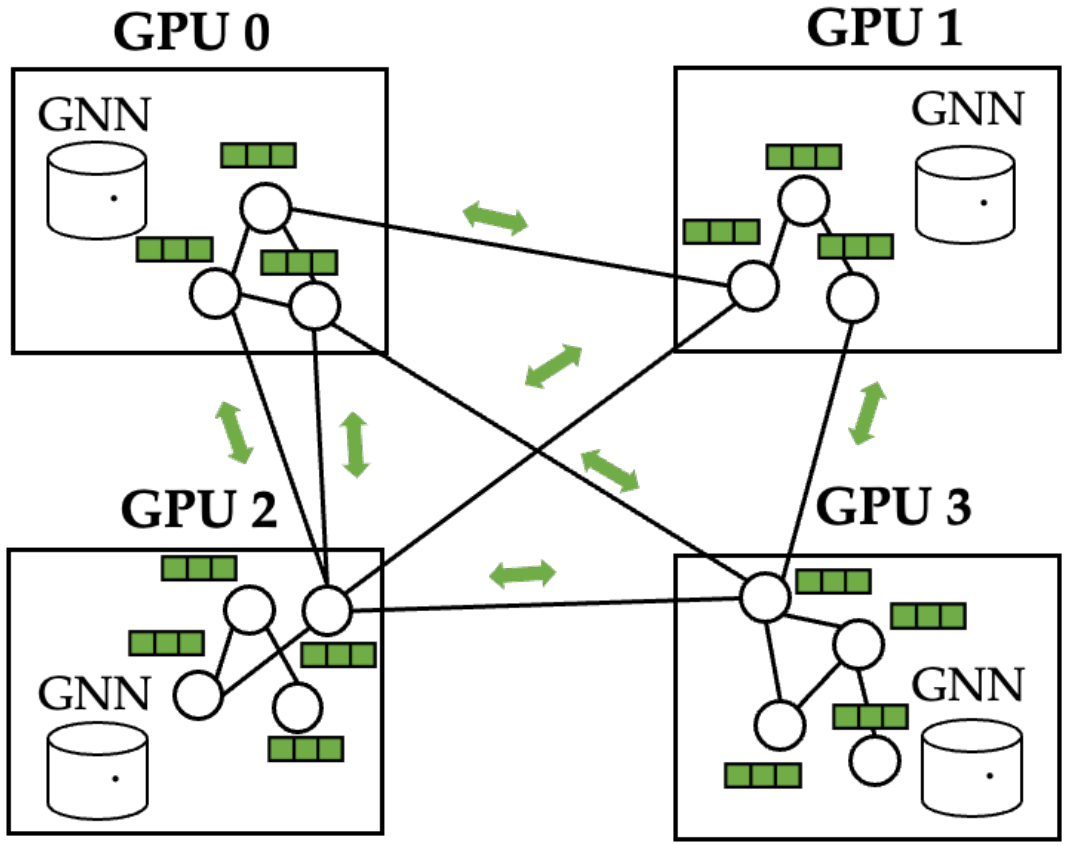
Optimized Data Transfer

- **Optimize graph data transfer: caching is the key**
  - Degree-based caching (for node features)
  - Pre-sampling-based caching (for node features)
  - Hybrid caching (for node features and adjacency matrix)

- **Optimize immediate data transfer**
  - Reduce inter-GPU transfer frequency
  - Reduce inter-GPU transfer volume
  - Select inter-GPU transfer route

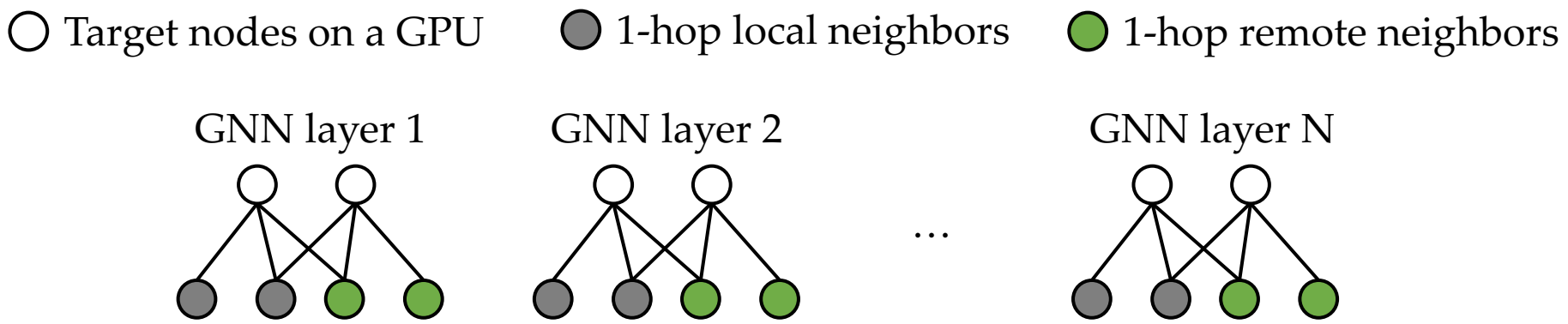
Recall: Transfer of Intermediate Data

- For full-graph training, the intermediate embeddings of remote neighbors are transferred among GPUs



Intermediate Data Transfer Volume

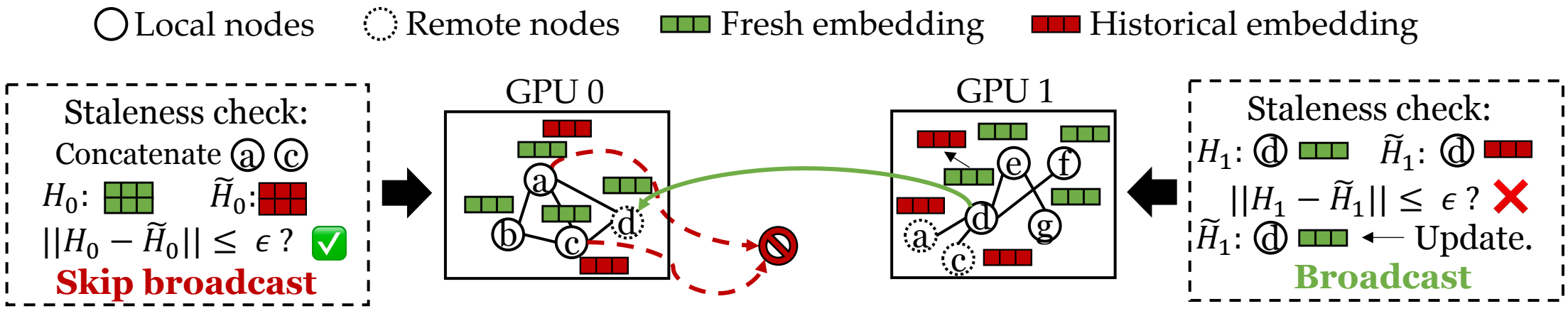
- In each epoch of full graph training, the volume of intermediate data that needs to be transferred from other GPUs to a target GPU grows linearly with the depth of the GNN



In each GNN layer, a target node needs to visit all its 1-hop neighbors

Reduce Transfer Frequency

- Each GPU keeps previously transferred intermediate embeddings
- If the staleness of embeddings is lower than a threshold:
  - Skip broadcasting fresh embeddings to the destination GPU(s)
  - Make destination GPU(s) use the historically received embeddings

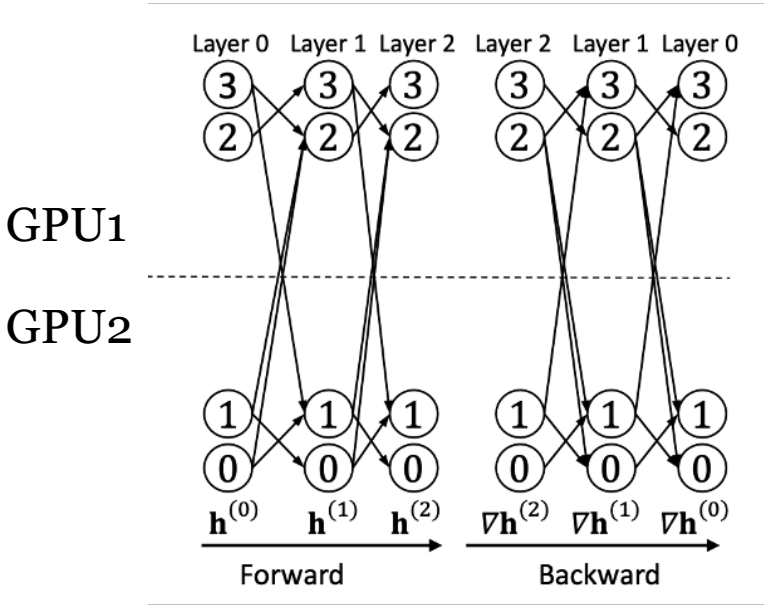


An illustration of inter-GPU skip broadcast.
After staleness check on each GPU, the GPU0 -> GPU1 data transfer is skipped.

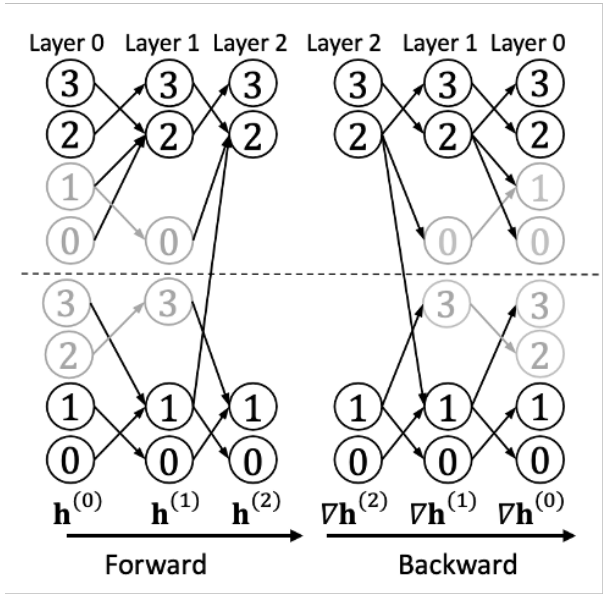
Peng et al. Sancus: staleness-aware communication-avoiding full-graph decentralized training in large-scale graph neural networks. PVLDB, 2022.

Reduce Inter-GPU Transfer Volume

- Duplicate k-hop neighbors of certain nodes and perform local computation without data transfer
- Tradeoff: increased computation versus reduced data transfer



(a)) full data transfer

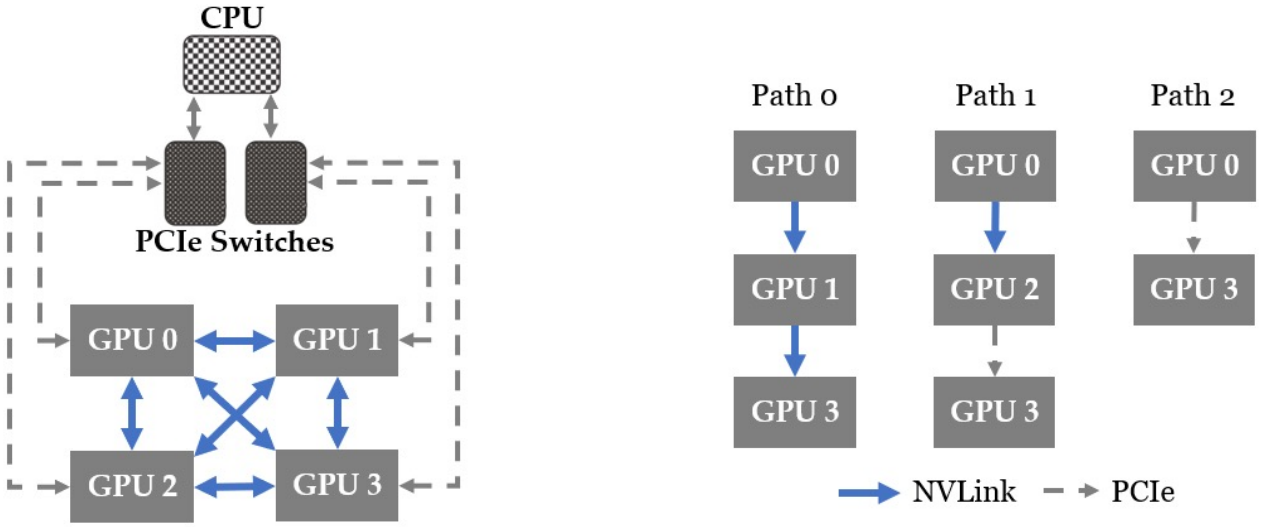


(b) partial data transfer + local computation

Wang et al. Neutronstar: distributed gnn training with hybrid dependency management. In SIGMOD, 2022.

Select Transfer Route

- Naïve peer-to-peer data transfer happens concurrently, causing network contention and load imbalance issue
- Find the optimal transfer routes for all embeddings to minimize the overall data transfer costs



Different embeddings transferred from source (GPU 0) to destination (GPU 3) can go by different transfer routes.

Cai et al. DGCL: an efficient communication library for distributed gnn training. In EuroSys, 2021.

What's Next: Part III

- **Efficient GNN Training from Data Management Perspective**
  - Stage 1: graph preprocessing (12 slides) → **Yanyan**
  - Stage 2: Batch preparation (8 slides)
  - Stage 3: Data transfer (19 slides)

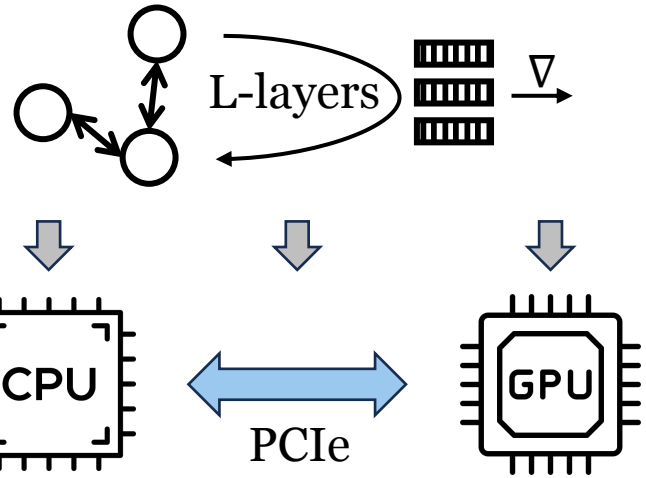
  → **Xin**
  - Stage 4: Model computation (9 slides)
  - Training Temporal GNN (9 slides)

  → **Jingzhi**
- **Future Research Directions**

Stage 4: Model Training

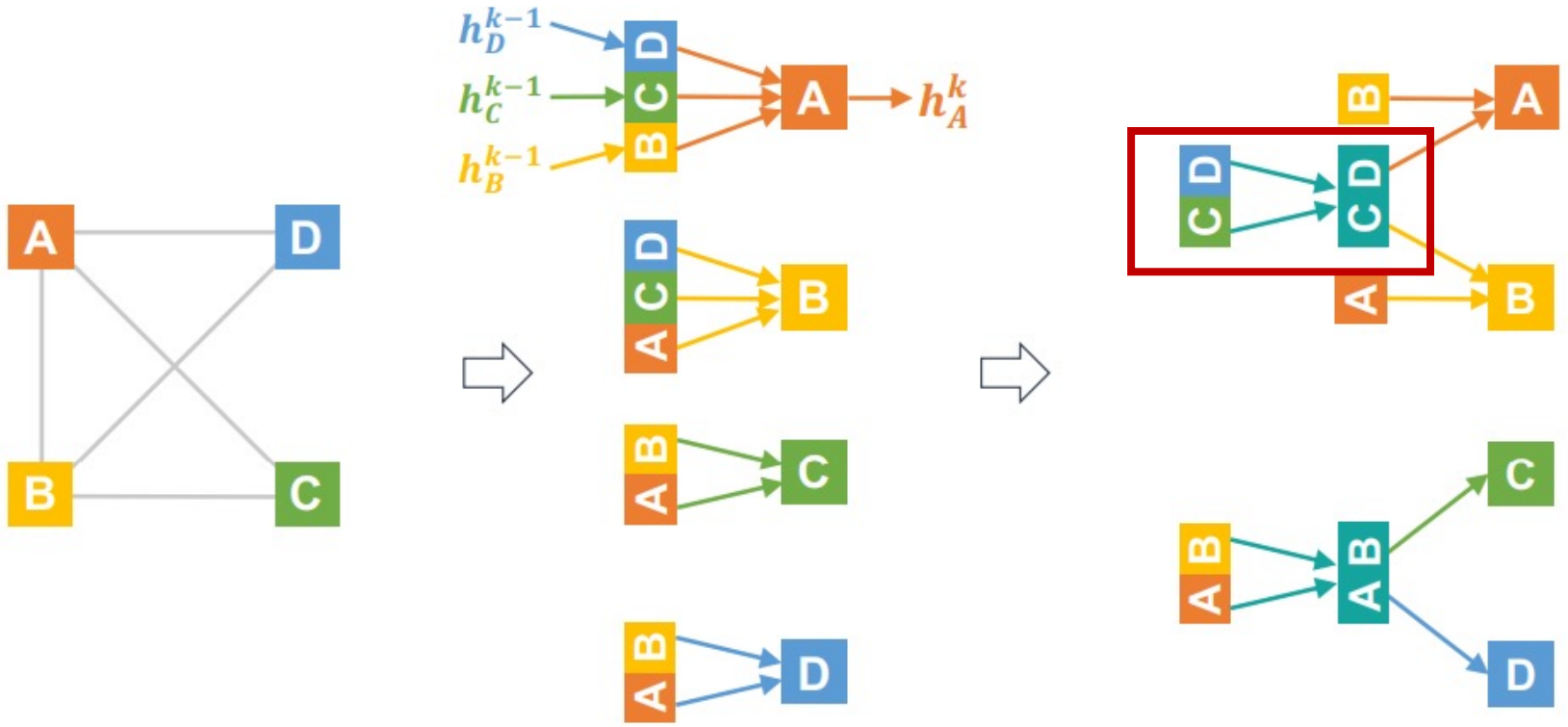
- **Goal: reduce training time when learning GNN parameters on GPU(s)**

- **Three ways to reduce GNN training time**
  - Computation graph optimization
  - Operator optimization
  - CPU-GPU collaborative computation



Computation Graph Optimization

- Avoid repeated computation by sharing common computations

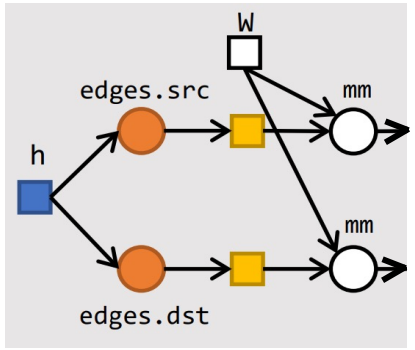


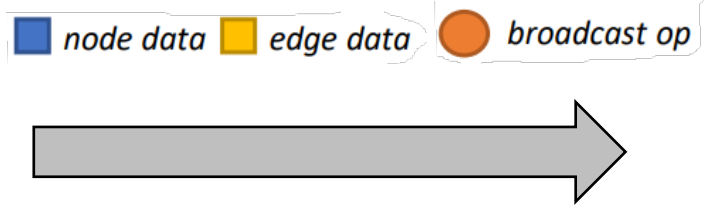
(a) Input graph (b) GNN-graph (c) HAG

Jia et al. Redundancy-free computation for graph neural networks. In SIGKDD, 2020.

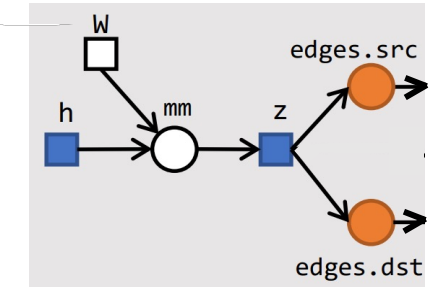
Computation Graph Optimization

- Broadcast reordering for lower computation complexity
- Original: fetch node embeddings for edges and transform them
- After reordering: transform node embeddings and fetch transformed node embeddings for edges





Reorder operators in the computation graph



$$z_{src} = edges.src['h'] \times W$$
$$z_{dst} = edges.dst['h'] \times W$$

$$z = h \times W$$
$$z_{src} = edges.src['z']$$
$$z_{dst} = edges.dst['z']$$

$O(|E| + |E|d_1d_2)$ → $O(|V|d_1d_2 + |E|)$

d_1: node feature size
W: $d_1 \times d_2$ weight matrix
$|E| \gg |V|$

Xie et al. Graphiler: Optimizing graph neural networks with message passing data flow graph. MLSys, 2022.

Operator Optimization

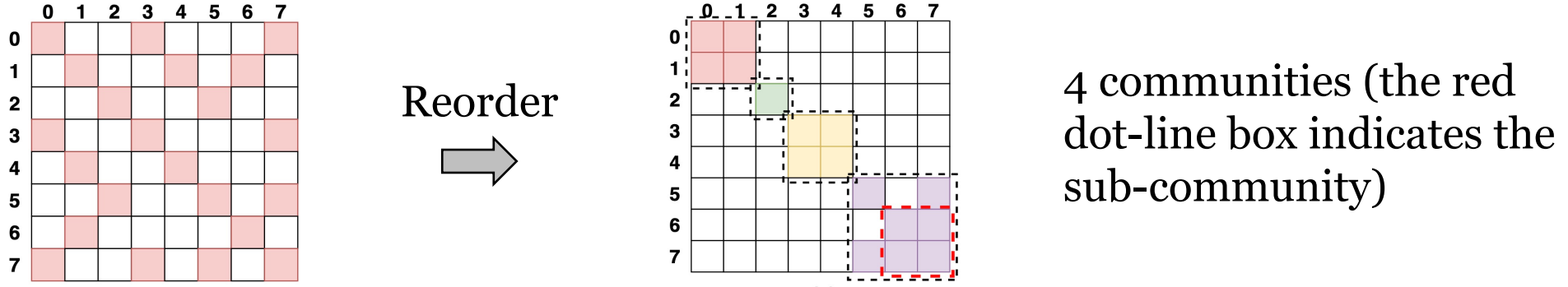
- Reduce execution time of some basic operators in GNN

- Sparse Matrix-Matrix Multiplication (SpMM): $O = SA$
  - S: sparse matrix, A: dense matrix, O: output matrix

- Sampled Dense-Dense Matrix Multiplication (SDDMM): $S_O = (A_1 A_2) \odot S$
  - A_1, A_2: dense matrices, S: sparse matrix, S_O: output sparse matrix
  - $\odot$: Hadamard product

Sparse Formats for Sparse Matrices

- Sparse matrices can be stored in different sparse formats
- Sparse format affects kernel implementation
- Rabbit reorder:
  - Detect communities and reorder nodes based on the communities they belong to
  - Improve locality for GNN aggregation



4 communities (the red dot-line box indicates the sub-community)

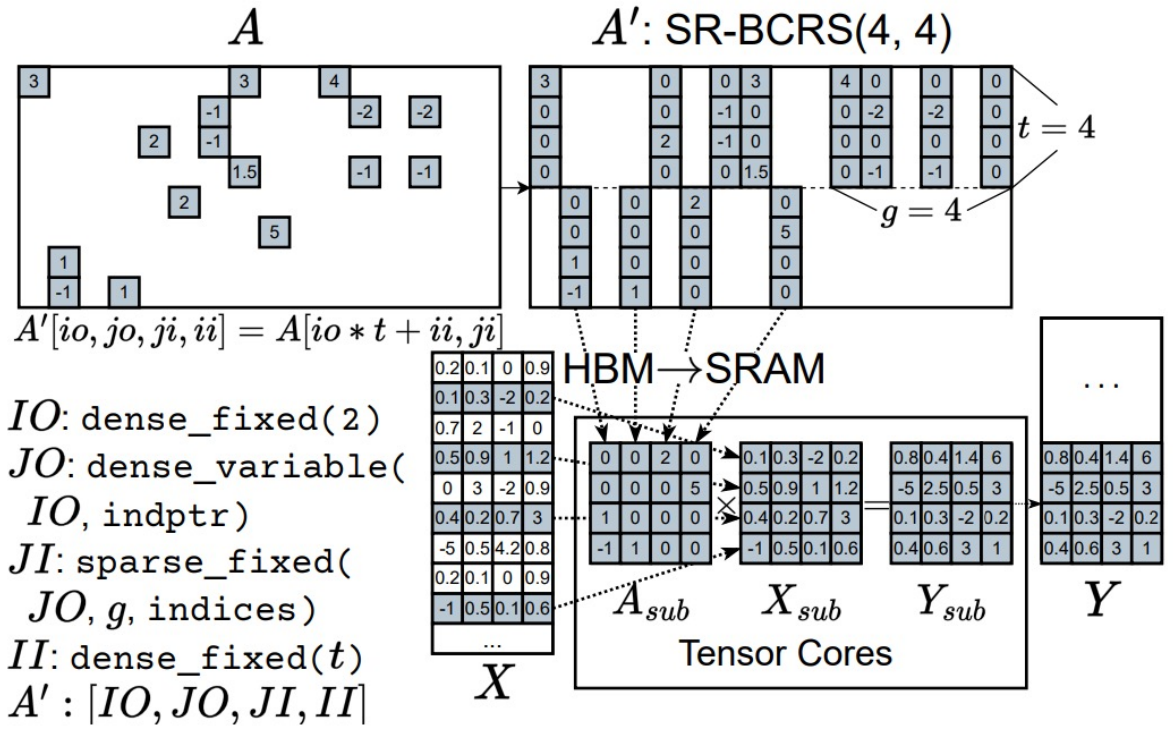
Original adjacency matrix

Adjacency matrix after reordering

Wang et al. GNNAdvisor: an adaptive and efficient runtime system for gnn acceleration on gpus. In OSDI, 2021.
Arai et al. Rabbit order: just-in-time parallel reordering for fast graph analysis. In IPDPS 2016.

Sparse Formats for Sparse Matrices

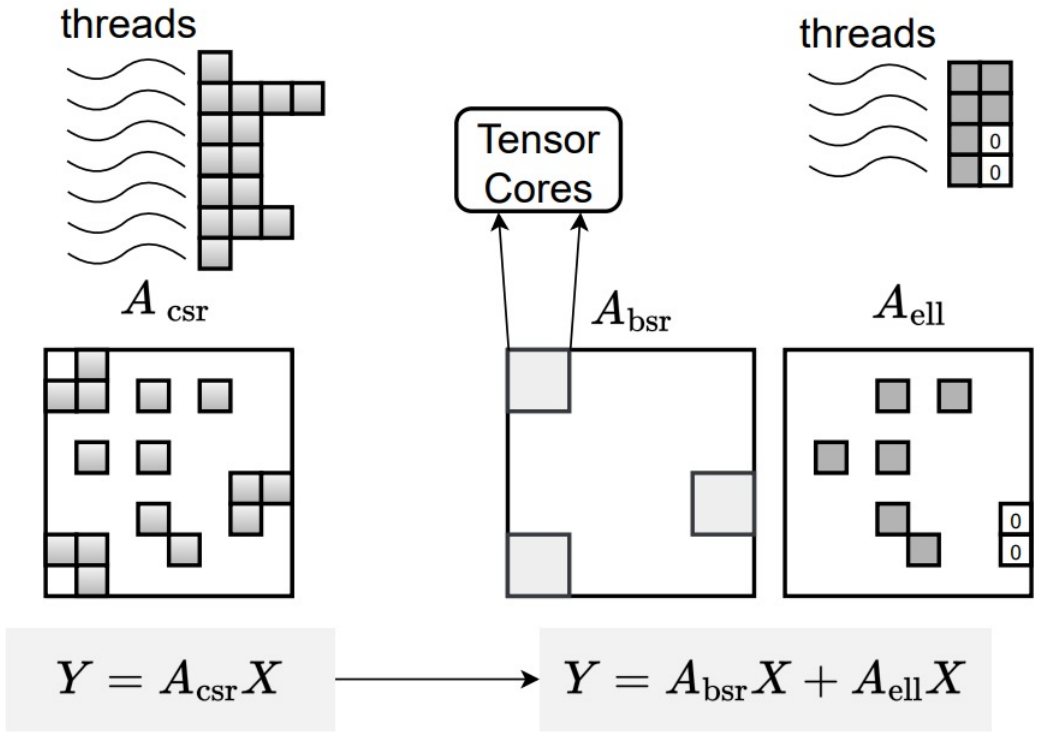
- Reorder columns and organize non-zeros into dense blocks
  - Run block matrix multiplication on efficient GPU Tensor Cores
- Cons: computation waste due to the zeros in the dense blocks



Ye et al: Composable abstractions for sparse compilation in deep learning. In ASPLOS, 2023.
Fang et al. STile: Searching hybrid sparse formats for sparse deep learning operators automatically. In SIGMOD, 2024.

Sparse Formats for Sparse Matrices

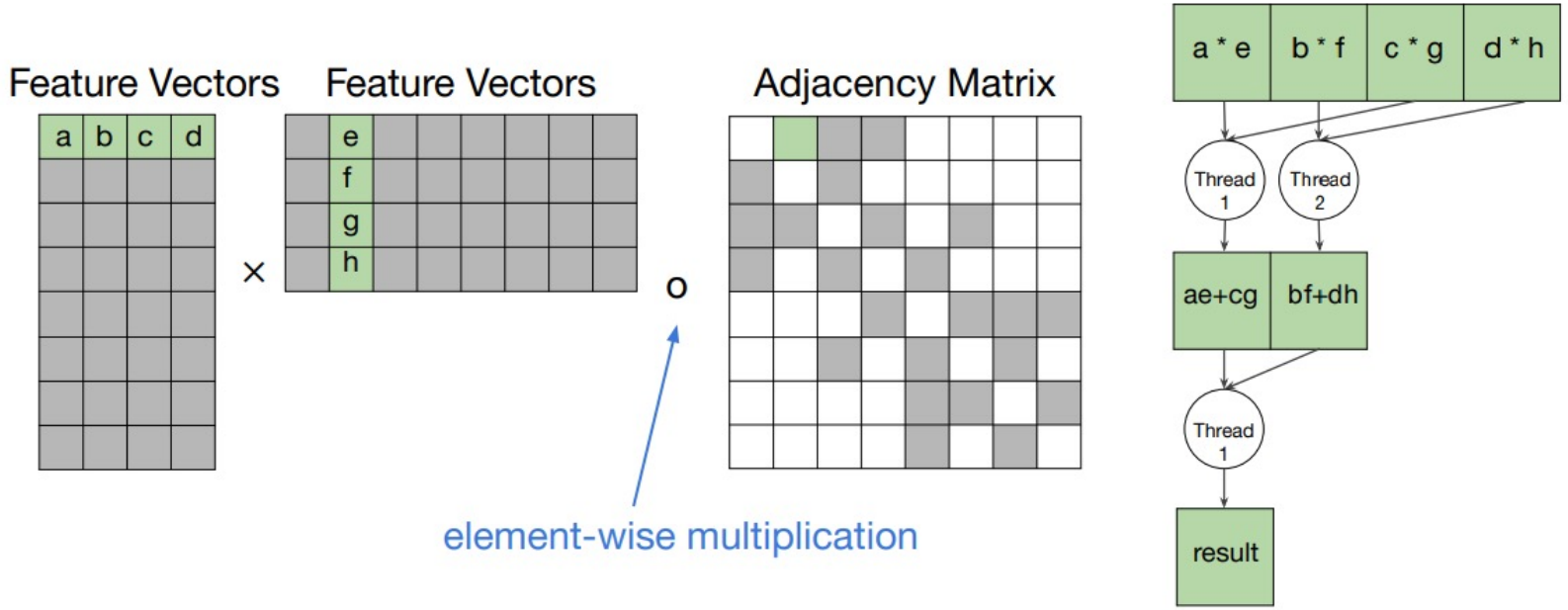
- Store dense regions into dense blocks and the remaining non-zeros compactly
  - Reduce computation waste while utilizing Tensor Cores



Ye et al: Composable abstractions for sparse compilation in deep learning. In ASPLOS, 2023.
Fang et al. STile: Searching hybrid sparse formats for sparse deep learning operators automatically. In SIGMOD, 2024.

Operator Kernel Implementation

- Workload balance: distribute workload among threads evenly
- Better resource utilization: GPU shared memory, Tensor cores, registers, memory bandwidth…

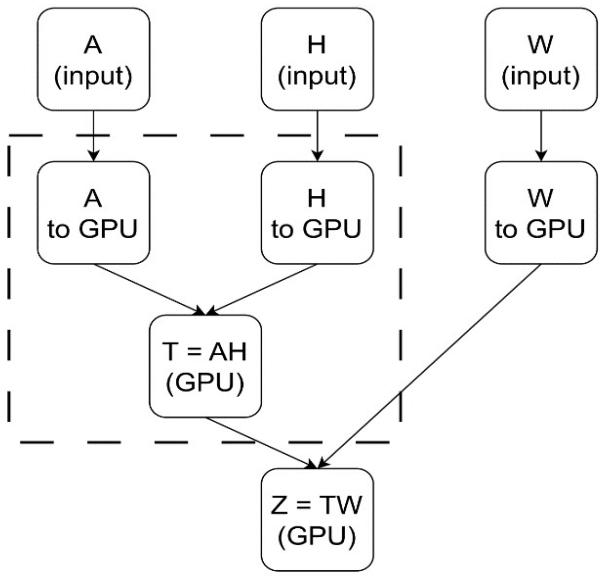


Each CUDA thread block processes a number of edges; all the threads in one block collectively process the dot-product operations on edges using tree reduction.

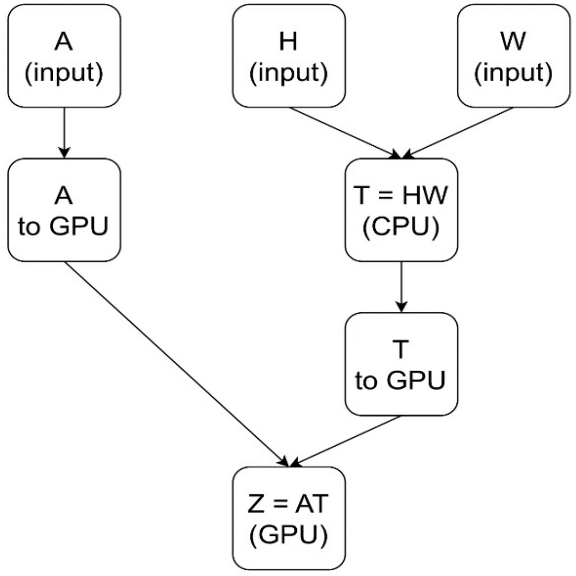
Hu et al. Featgraph: A flexible and efficient backend for graph neural network systems. In SC, 2020.

CPU-GPU Collaborative Computation

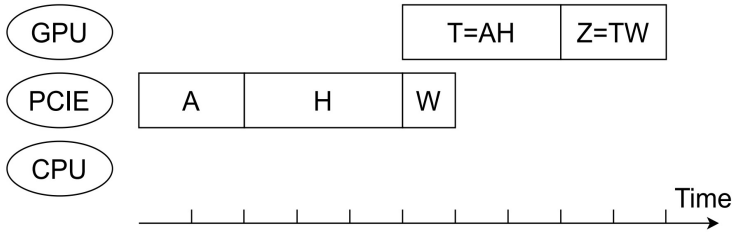
- Assign a part of operators in the computation graph to the CPU, thus saving data transfer time
- GNN training with CPU-GPU hybrid processing and more fine-grained pipeline parallelism



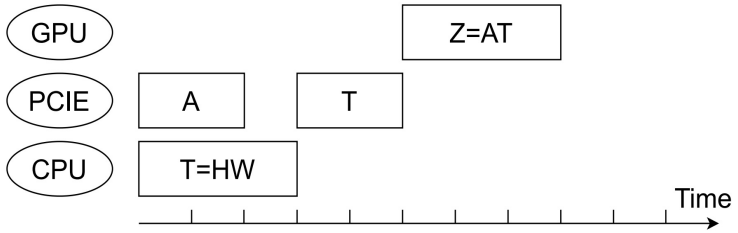
(a) EPG: pure GPU.



(b) EPG: HybridPU.



(c) Device: pure GPU.

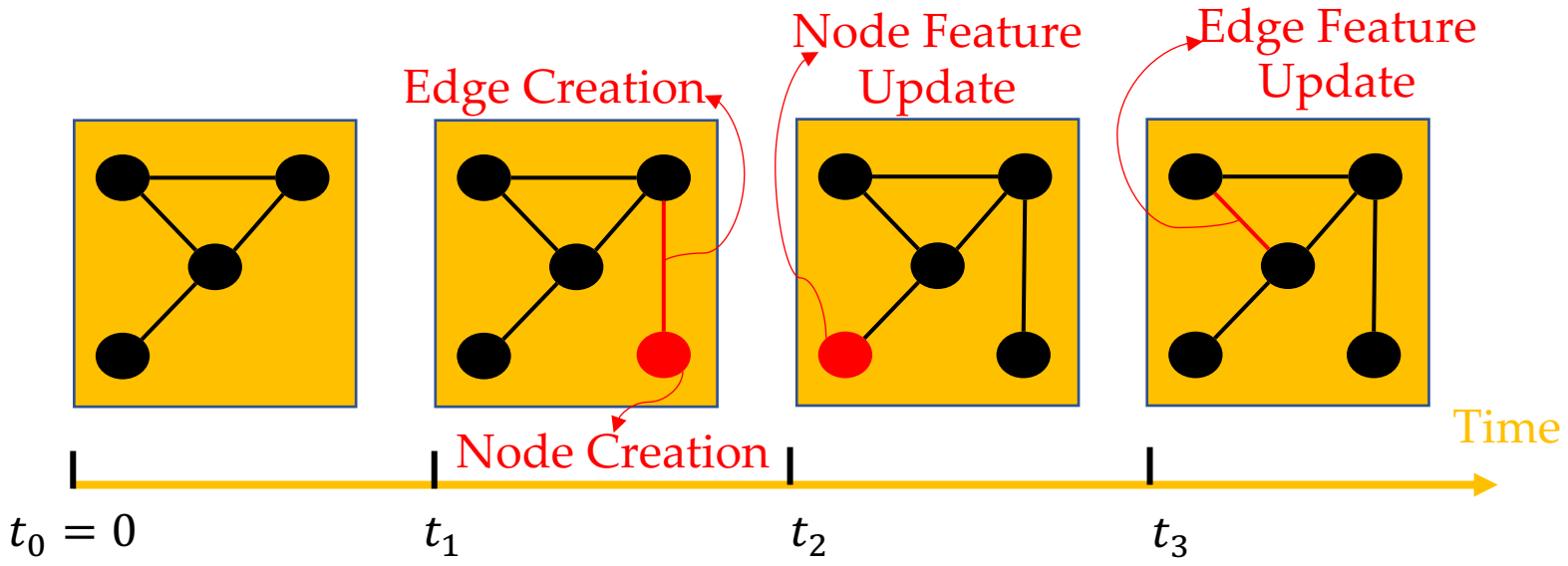


(d) Device: HybridPU.

Li et al. DAHA: accelerating gnn training with data and hardware aware execution planning. PVLDB, 2024.

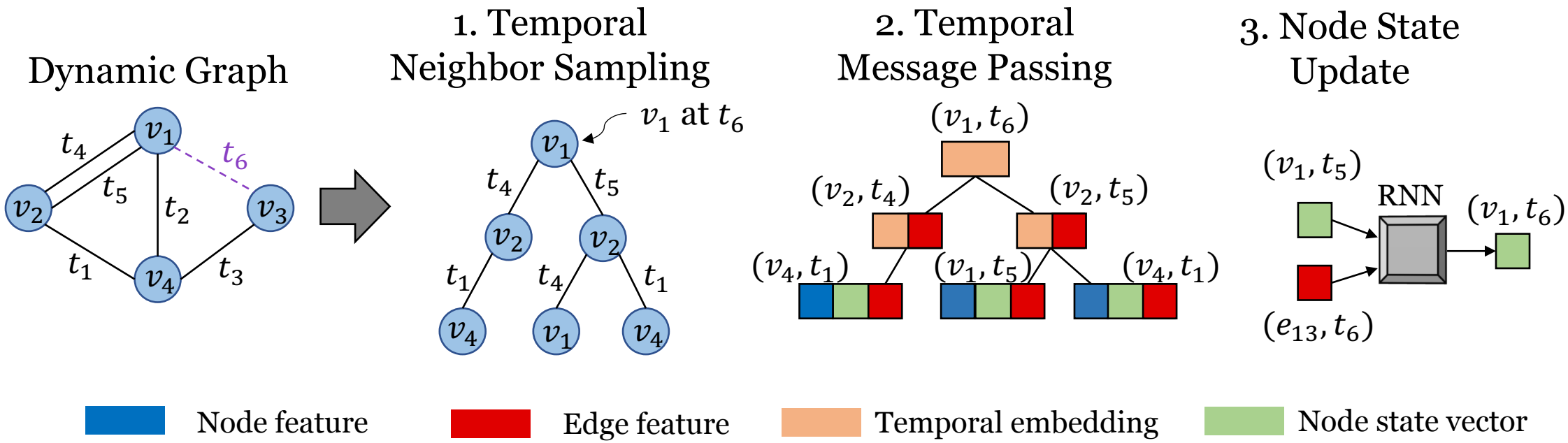
Graphs are Dynamically Changing

- **Continuous-time Dynamic Graphs (CTDGs)**
  - A sequence of observed temporal events
  - Formally, x_{t_i} denotes an event happened at timestamp t_i
  - CTDG at t_n is: $G_{t_n} = (x_{t_0}, x_{t_1}, \ldots, x_{t_n})$



Temporal GNNs (T-GNNs)

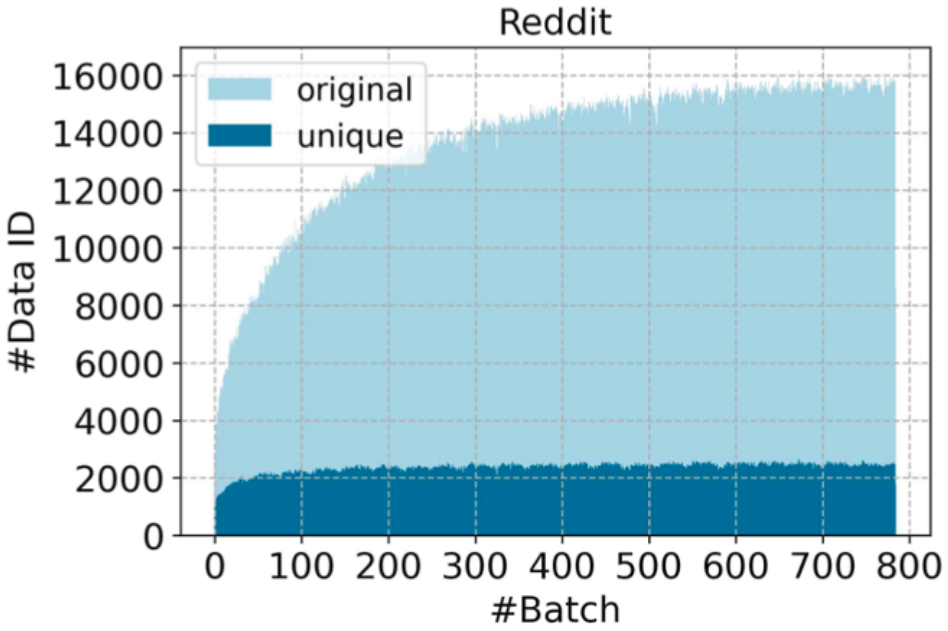
- T-GNNs mimic the message passing scheme of GNNs, except:
  - each node maintains a state vector summarizing historical events
  - 3 training steps given a new event



When a new edge arrive at t6, update the state vectors of v1 and v3. The example is about v1.

T-GNN Training Challenge I

- High Data Transfer Volume
  - Same neighboring nodes with different timestamps may be sampled in the same batch: $[(v_1, t_1), (v_1, t_2), \ldots, (v_1, t_n)]$
  - Transferring the sampling result as in training static GNNs leads to large data transfer volume (e.g., v_1's feature is transferred for each t_i)

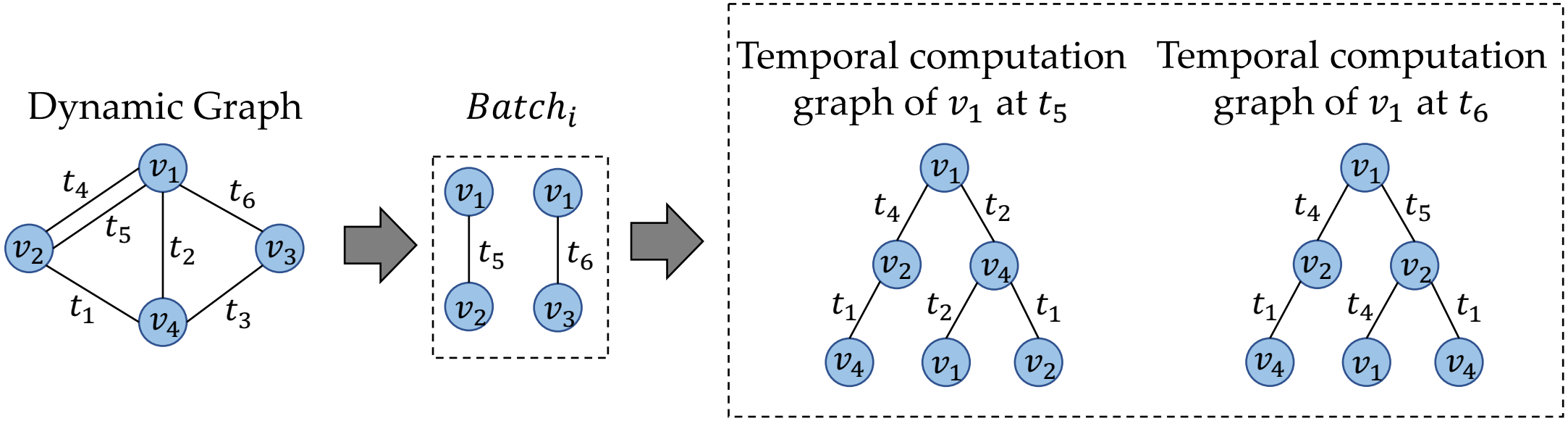


Temporal sampling result:
# of unique data << # of sampling data

Gao et al. ETC: efficient training of temporal graph neural networks over large-scale dynamic graphs. PVLDB, 2024.

T-GNN Training Challenge II

- **High Computation Complexity**
  - Temporal explosion: each timestamp has a unique computation graph
  - Neighbor explosion: computational cost increases exponentially with the model depth L



In $Batch_i$, 2 edges are sampled, and $(v_1, t_5), (v_1, t_6), (v_2, t_5), (v_3, t_6)$ are target nodes. The figure shows the temporal computation graphs for $(v_1, t_5), (v_1, t_6)$ as an example.

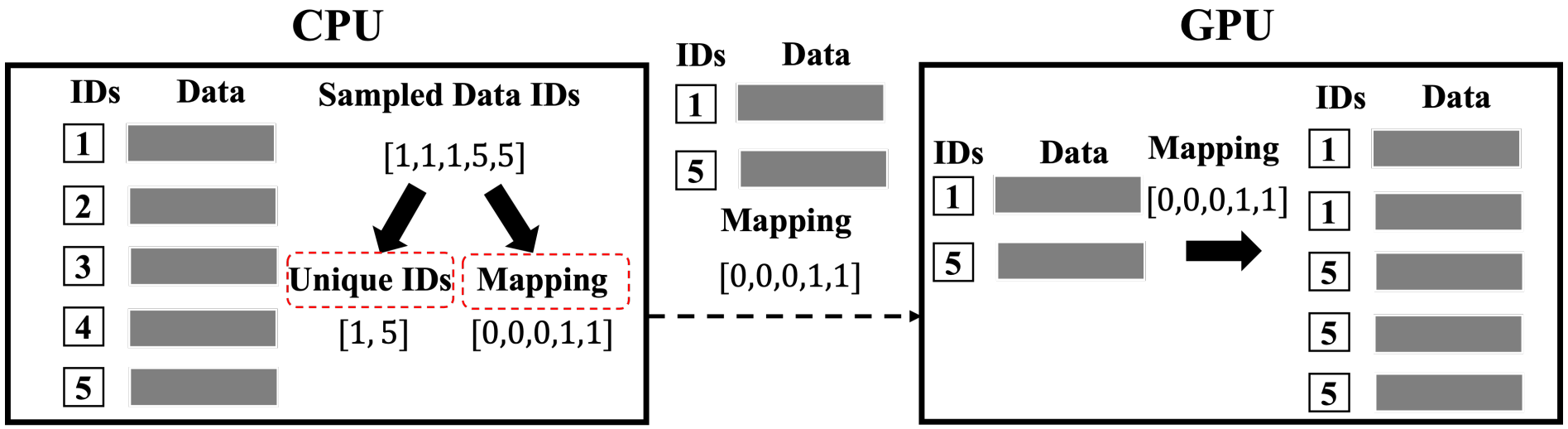
Towards Efficient T-GNN Training

- **Optimize data transfer**
  - Remove transfer redundancy
  - Buffer-based data transfer

- **Optimize model computation**
  - Influence-based single-layer aggregation
  - Historical embedding reuse

Remove Transfer Redundancy: ETC

- Transform the sampled data IDs into unique IDs and get a mapping for reconstruction on CPU
- Transfer unique graph data and the mapping from CPU to GPU
- Reconstruct the required data on GPU



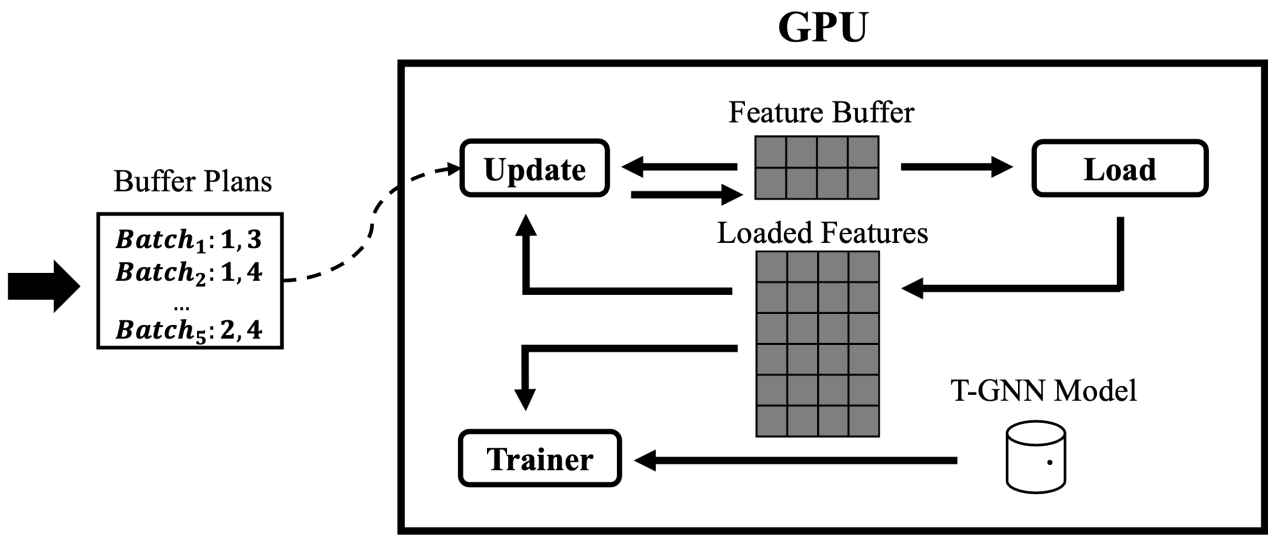
Gao et al. ETC: efficient training of temporal graph neural networks over large-scale dynamic graphs. PVLDB, 2024.

Buffer-based Data Transfer: SIMPLE

- Maintain a small GPU buffer for input graph data
  - Pre-sampling to get data occurrences in each temporal batch
  - Derive buffer plans for each batch to minimize the total data transfer volume of all batches
  - During training: update GPU buffer based on the buffer plans

Data Occurrence Patterns

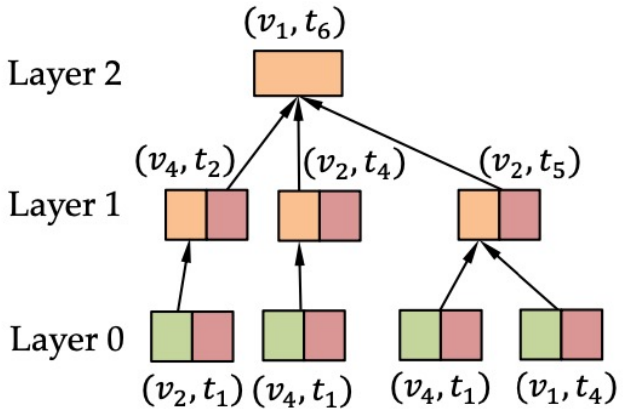
Data IDs	$Batch_1$	$Batch_2$	$Batch_3$	$Batch_4$	$Batch_5$
❶	5	5	0	0	2
❷	1	0	6	2	0
❸	2	4	0	4	2
❹	0	1	6	1	0



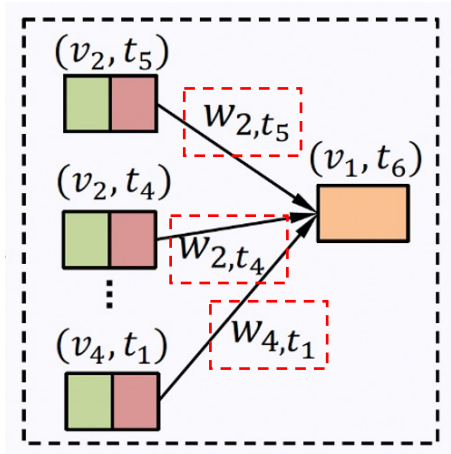
Gao S et al. SIMPLE: efficient temporal graph neural network training at scale with dynamic data placement. In SIGMOD, 2024.

Influenced-based Aggregation: Zebra

- Not all temporal neighbors are equally important
  - Use the Temporal Personalized PageRank (T-PPR) metric to quantify influence of multi-hop temporal neighbors for target nodes
  - Conduct single-layer, influence-based weighted aggregation instead of multi-hop temporal neighborhood aggregation



Multi-hop recursive temporal aggregation

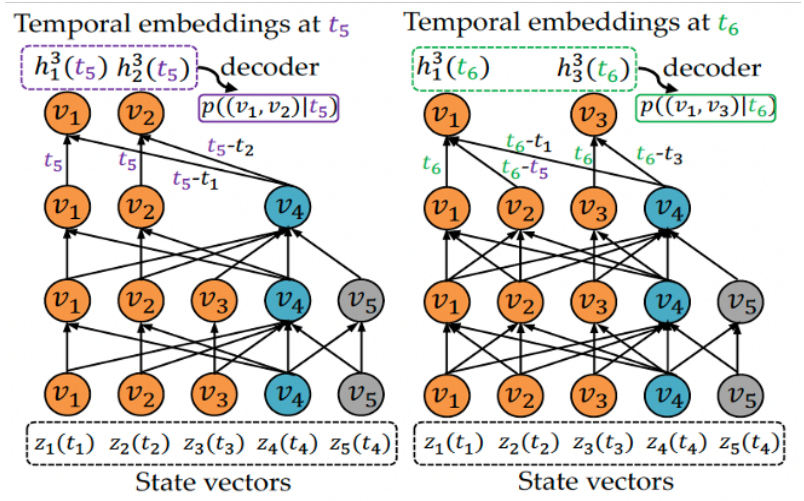


Single-layer weighted aggregation

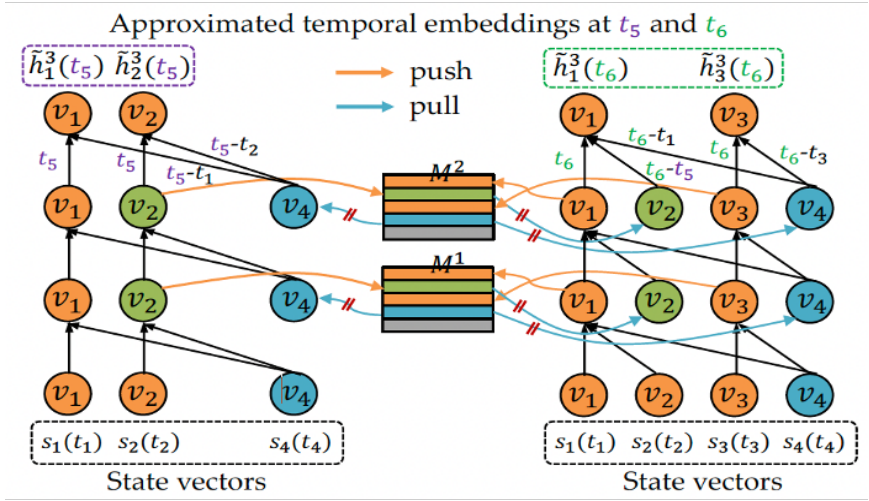
Li et al. Zebra: when temporal graph neural networks meet temporal personalized pagerank. PVLDB, 2023.

Historical Embedding Reuse: Orca

- When calculating embeddings for target temporal nodes:
  - Pull the embeddings of neighbors from cache
  - Push the newly computed embeddings of target nodes to cache
- Cache management if GPU memory is limited
  - Based on when an embedding will be used in the future



T-GNN without embedding reuse



T-GNN with embedding reuse

Li et al. Orca: Scalable temporal graph neural network training with theoretical guarantees. In SIGMOD, 2023.

So far…

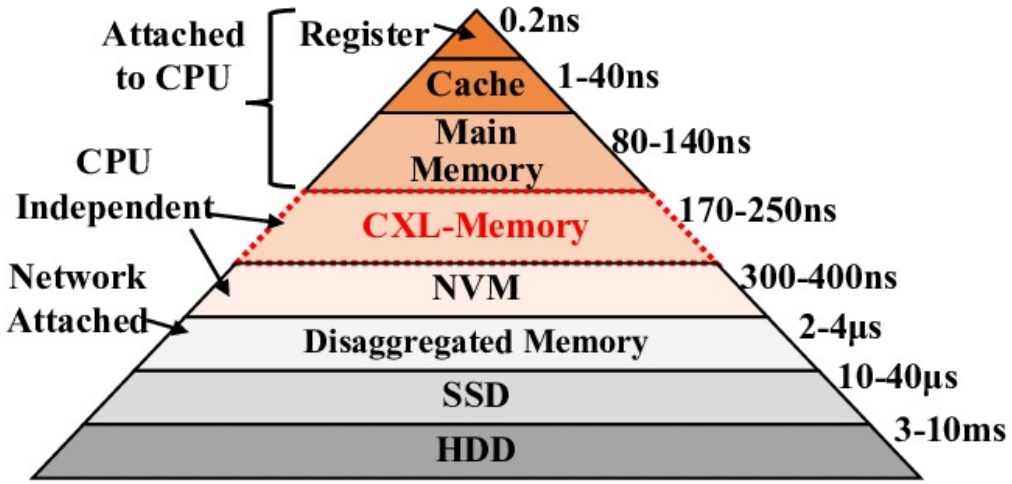
- **GNNs training as 4-stage graph processing procedure**
  - Graph preprocessing: reduce graph size, graph partitioning
  - Batch generation
  - Data transfer
  - Model training
- **Four stages are inter-dependent**
  - When optimizing one, the others might be affected
- **Ultimate goal: end-to-end training efficiency**
  - Three-factor consideration: data, model, hardware

Future Research Directions

- More tiers of memory hierarchy
- Hybrid CPU-GPU Data Transfer
- GNN Training on Dynamic Graphs
- Text-Attributed Graphs

More Tiers of Memory Hierarchy

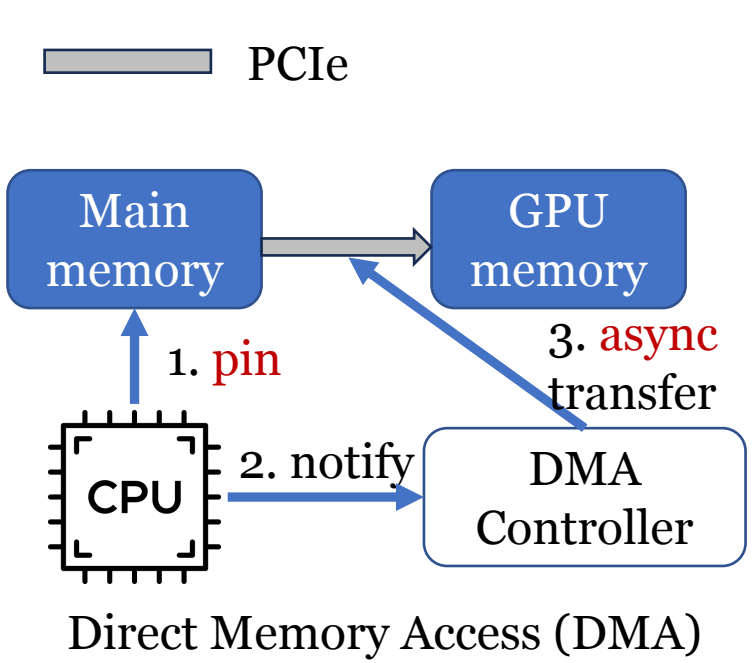
- Different memory tiers: bandwidth, speed, latency
- Opportunities
  - GNN training cost reduction with HDD/Network storage
  - Latency-expense tradeoff via computation offloading (to CPU, cloud service, …)



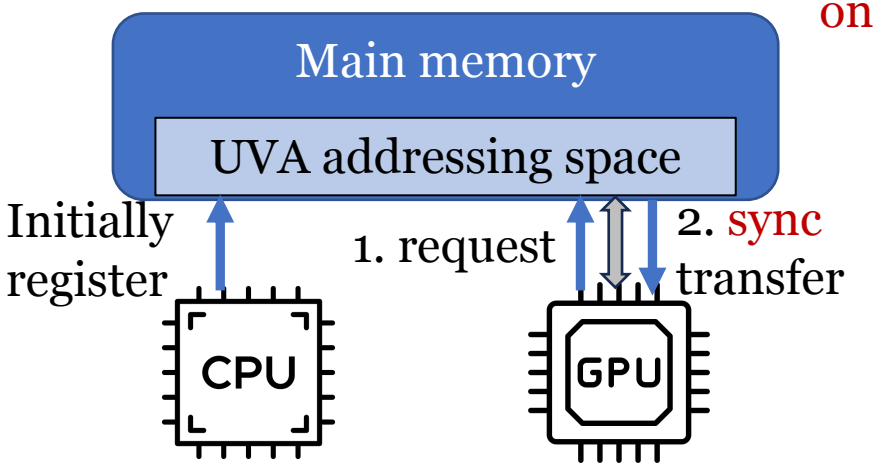
Hybrid CPU-GPU Data Transfer

- Three transfer techniques with different properties ➔ hybrid solution

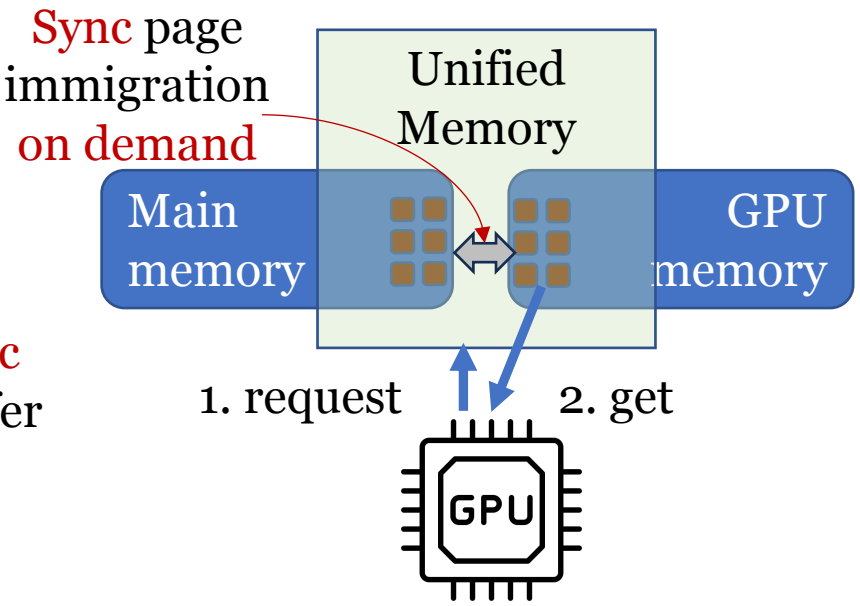
	Direct Memory Access	Unified Virtual Addressing	Unified Memory
Granularity	Any continuous data (≥ 1 B)	GPU Cacheline (128 B)	Page (4KB)
Major workload on	CPU	GPU	GPU



Direct Memory Access (DMA)



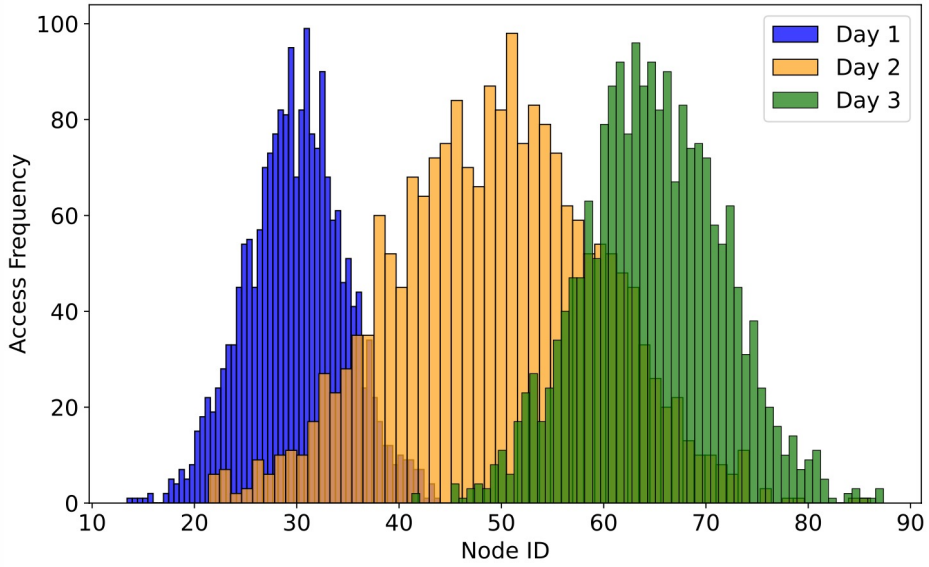
Unified Virtual Addressing (UVA)



Unified Memory (UM)

GNN Training on Dynamic Graphs

- Graph content and graph access pattern change with time and are unknown in advance
- Opportunities
  - Dynamic cache
  - Adaptive graph convolution kernels
  - Dynamic graph reordering

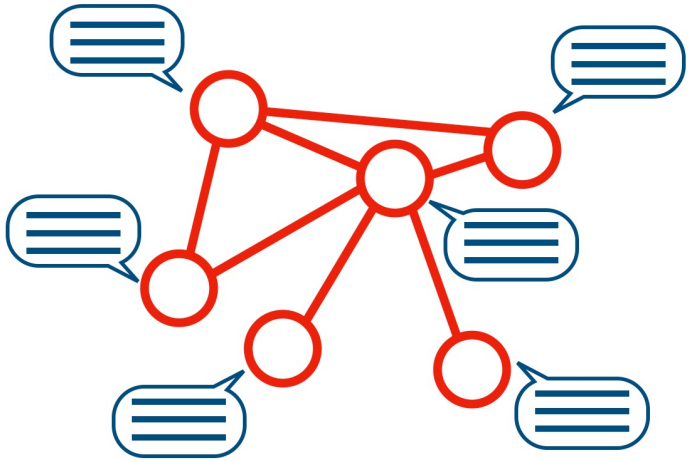


Graph access pattern changes over time

Text-Attributed Graphs (TAGs)

- TAGs: vertices or edges in the graph have rich text features
- Workload shifts: text modality requires language models
- Opportunities
  - Coordinate GNN's caching and LLM's KV-cache
  - Schedule computations in different modalities

Graph Space Learning



Text Space Learning

Q&A

THANK YOU!

Efficient Training of Graph Neural Networks on Large Graphs, prepared by Yanyan Shen, Lei Chen, Jingzhi Fang, Xin Zhang, Shihong Gao, Hongbo Yin**.**