NEUTRON
STAR

NeutronStar: **Distributed** GNN Training with **Hybrid** Dependency Management

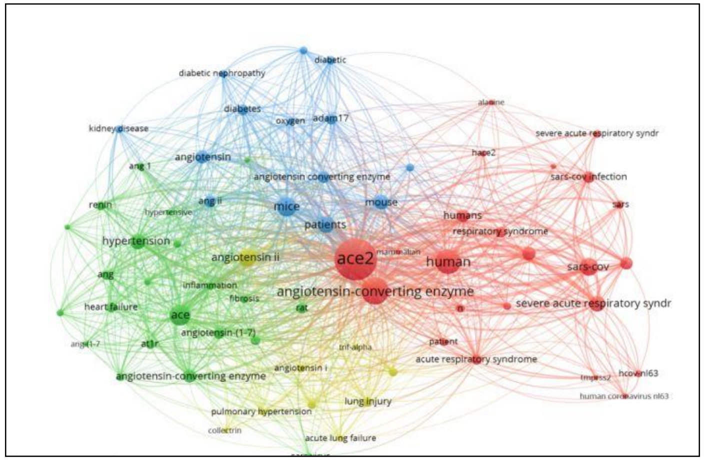
Qiange Wang, Yanfeng Zhang, Hao Wang, Chaoyi Chen,
Xiaodong Zhang, Ge Yu

Northeastern University, China
International Digital Economy Academy (IDEA), China
The Ohio State University, USA

SIGMOD 2022

Graph Neural Network

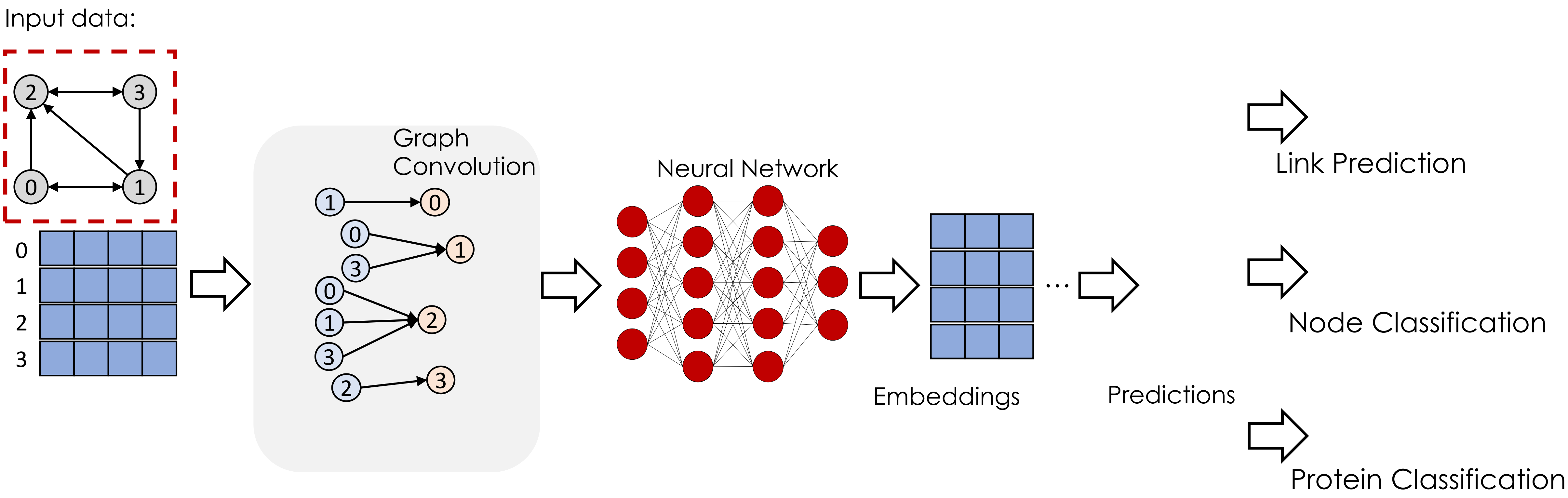
(a) Social Networks



(b) Knowledge Graph

(c) Biological networks

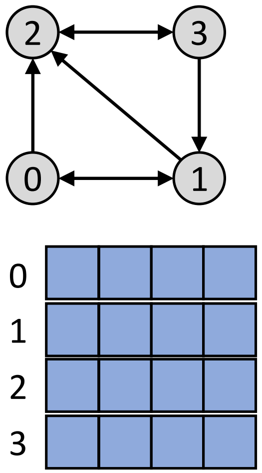
Graph Neural Network

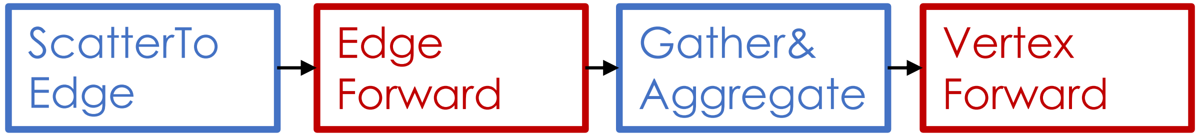


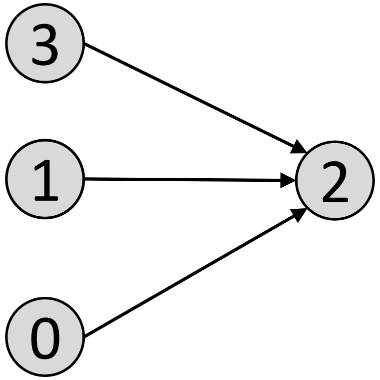
Execution Pattern of GNN

Forward computation (1-layer):

Input data:



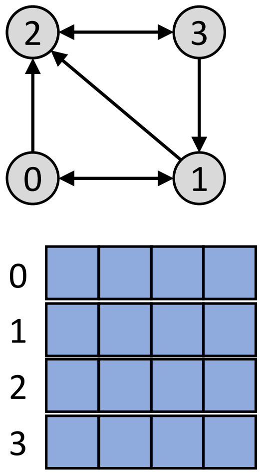


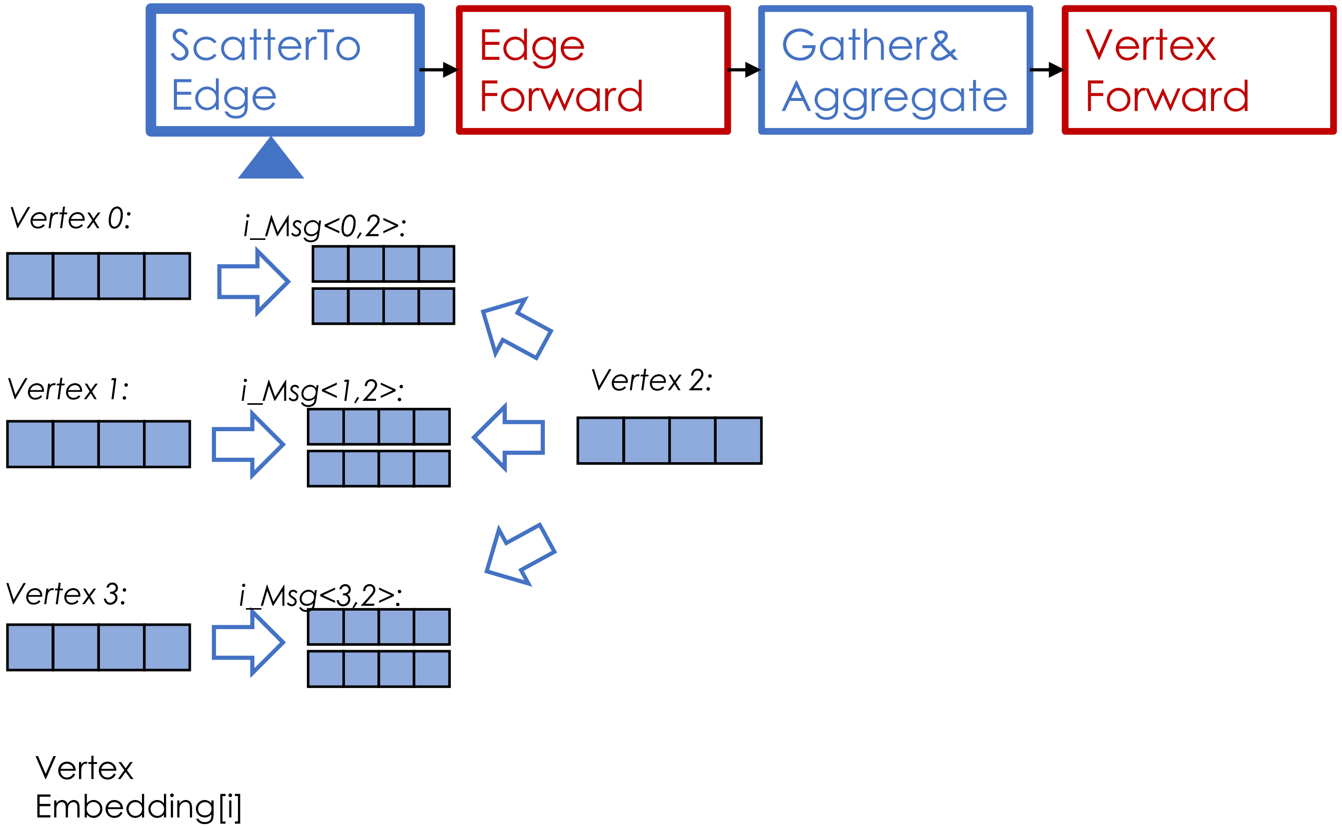


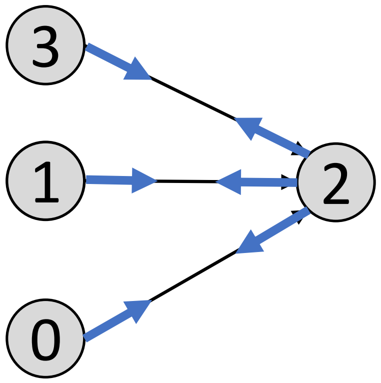
Execution Pattern of GNN

Forward computation (1-layer):

Input data:

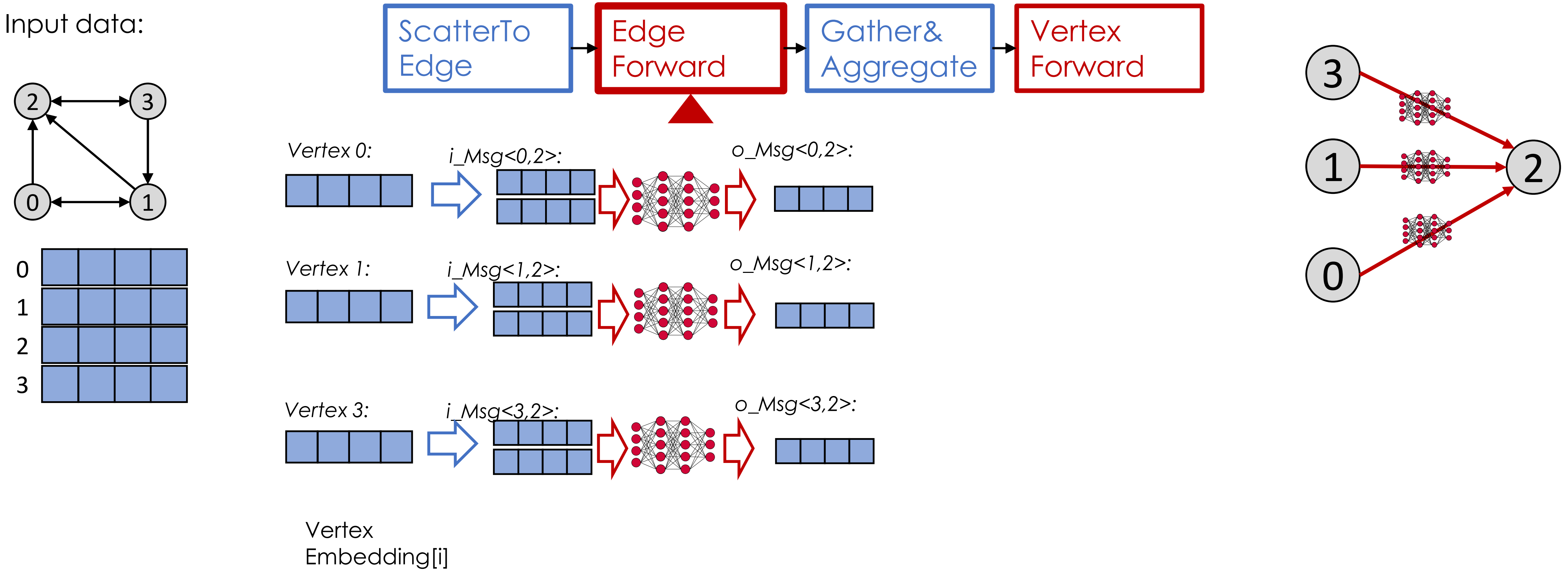






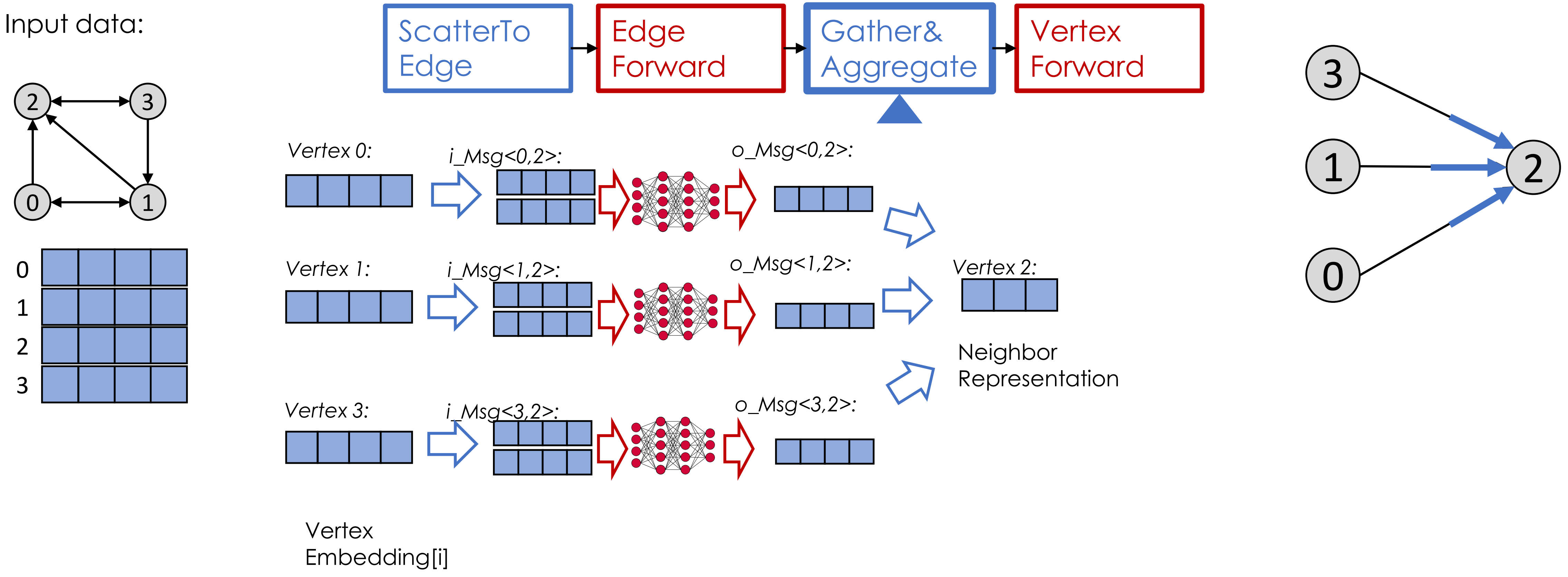
Execution Pattern of GNN

Forward computation (1-layer):



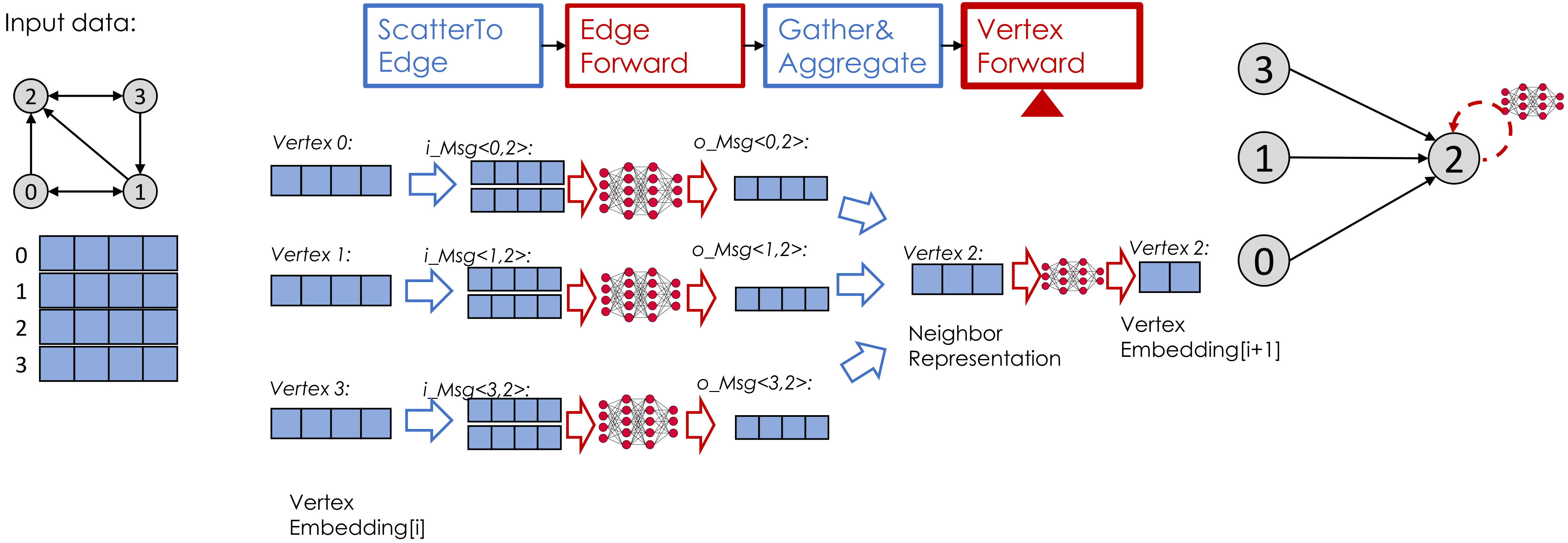
Execution Pattern of GNN

Forward computation (1-layer):



Execution Pattern of GNN

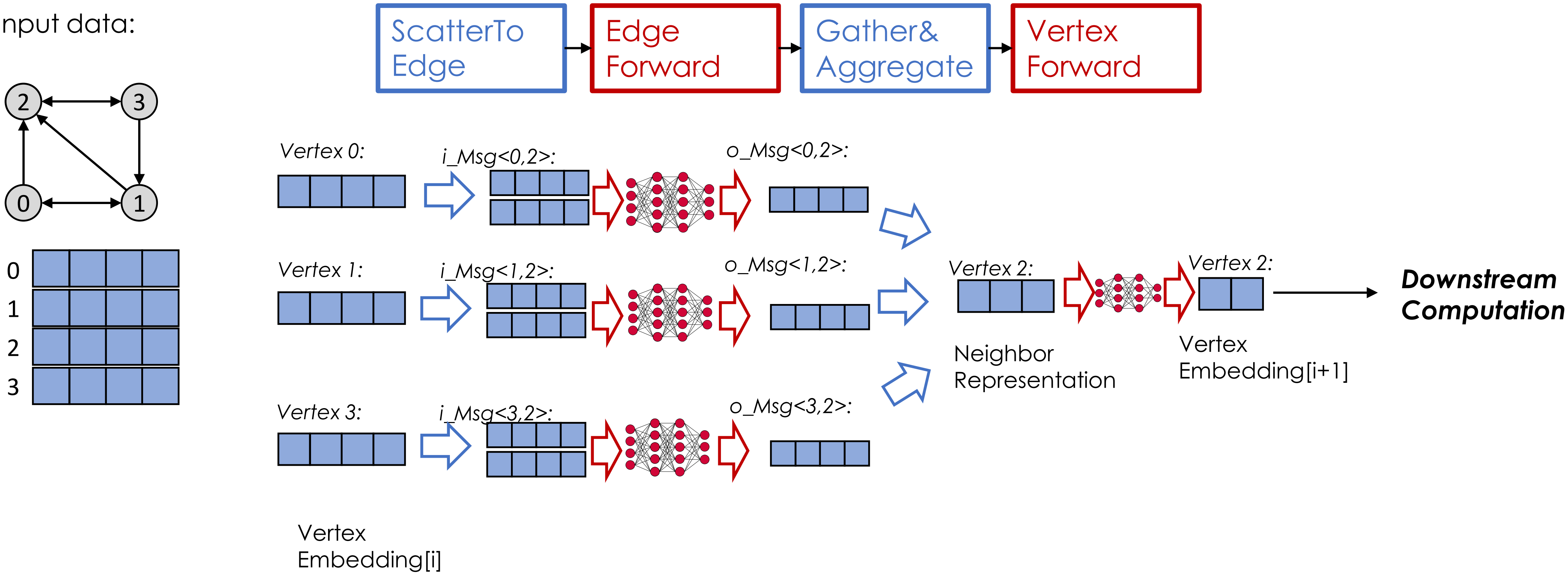
Forward computation (1-layer):



Execution Pattern of GNN

Forward computation (1-layer):

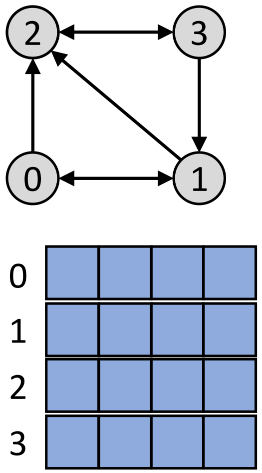
Input data:



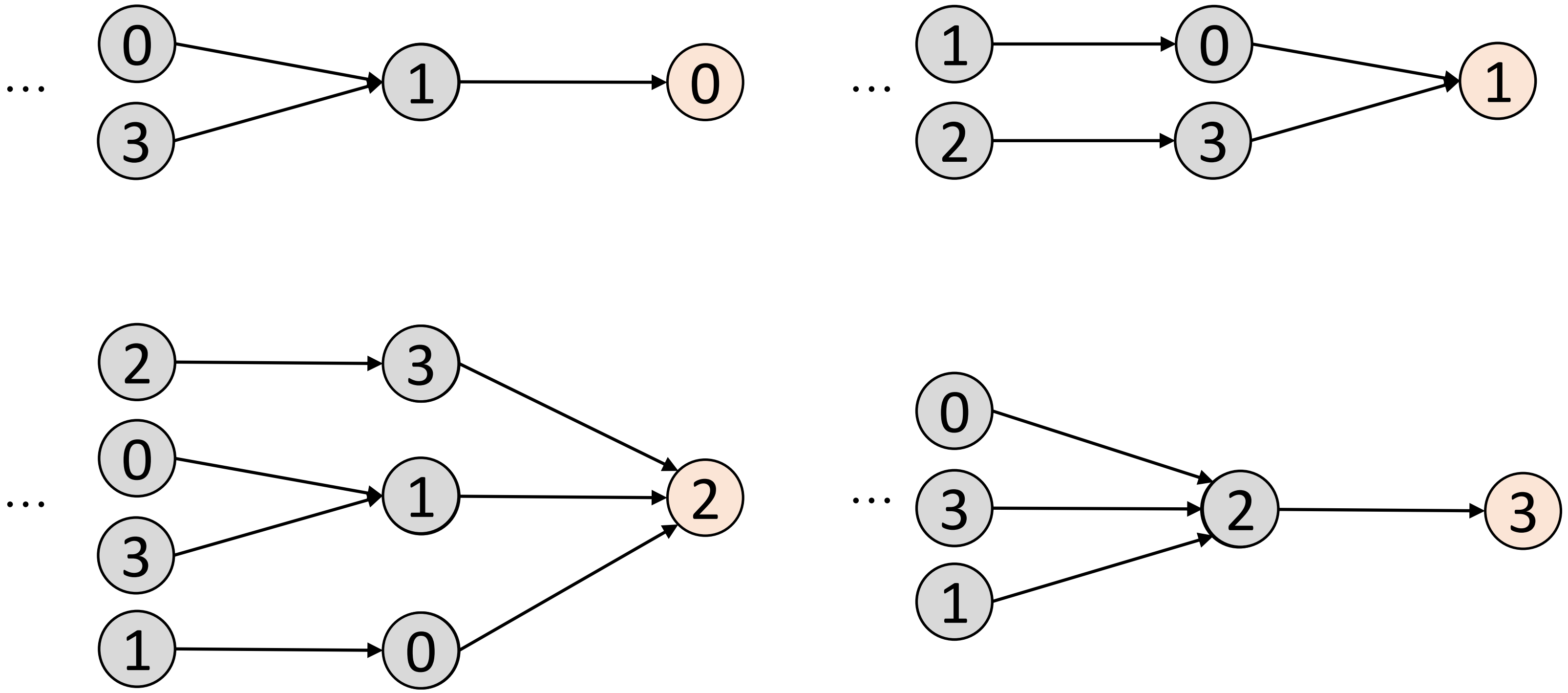
Execution Pattern of GNN

Forward computation:

Input data:



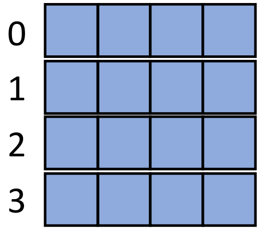
The computation of each vertex needs to gather information from its multi-hop neighbors



Distributed GNN Training

DNN training:

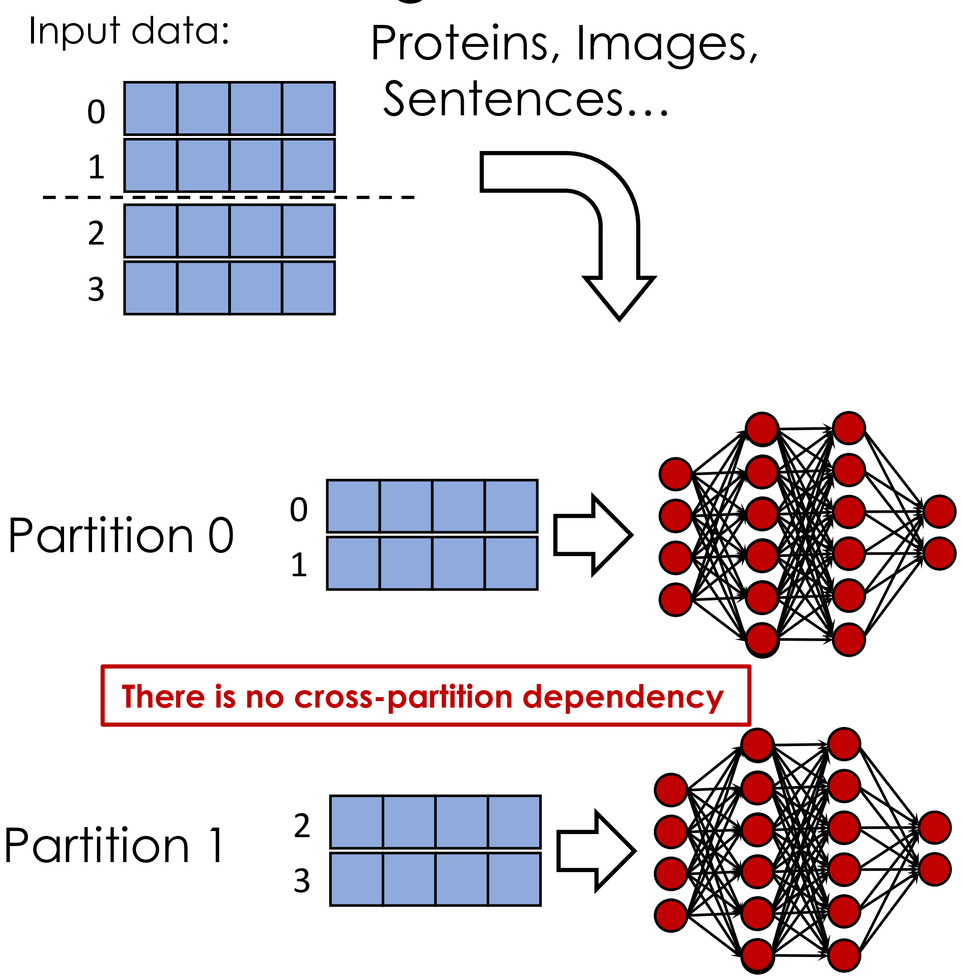
Input data:



Proteins, Images,
Sentences…

Distributed GNN Training

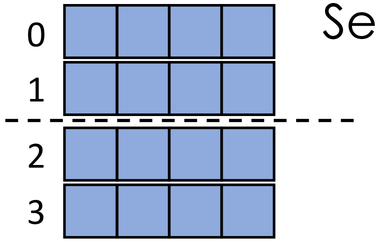
DNN training:



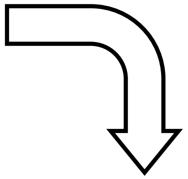
Distributed GNN Training

DNN training:

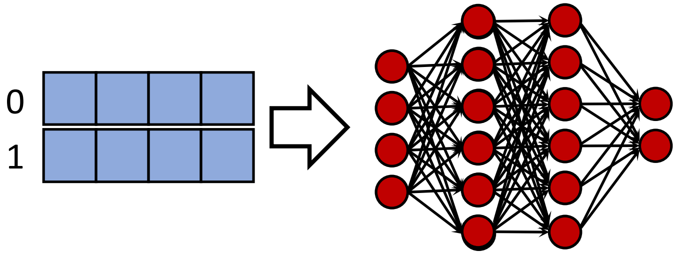
Input data:



Proteins, Images, Sentences…

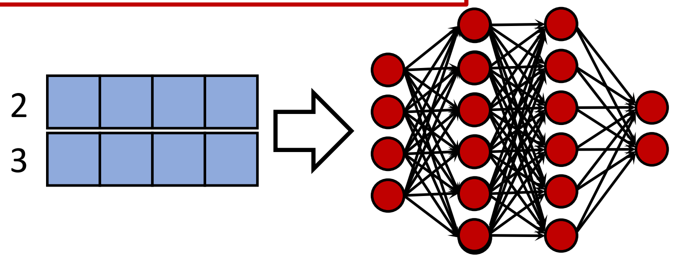


Partition 0



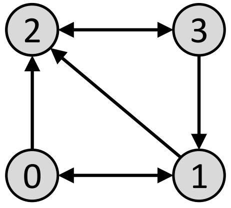
There is no cross-partition dependency

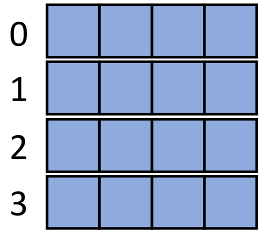
Partition 1



GNN training:

Input data:



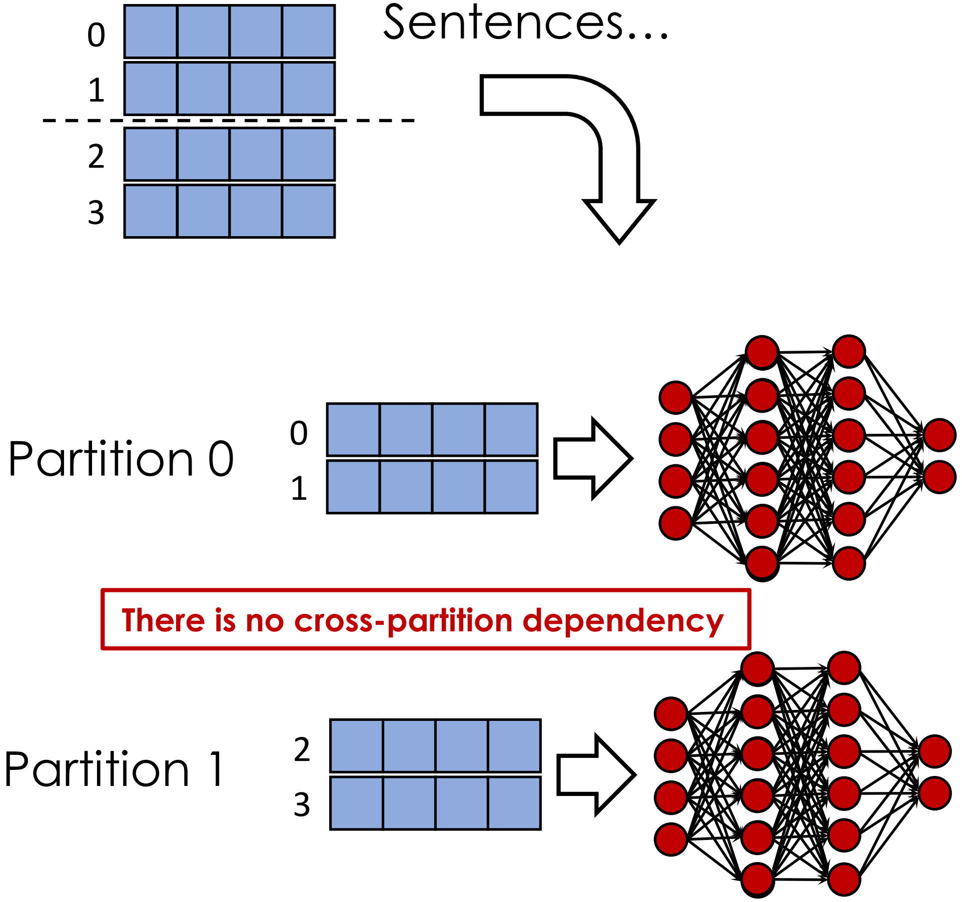


Distributed GNN Training

DNN training:

Input data:

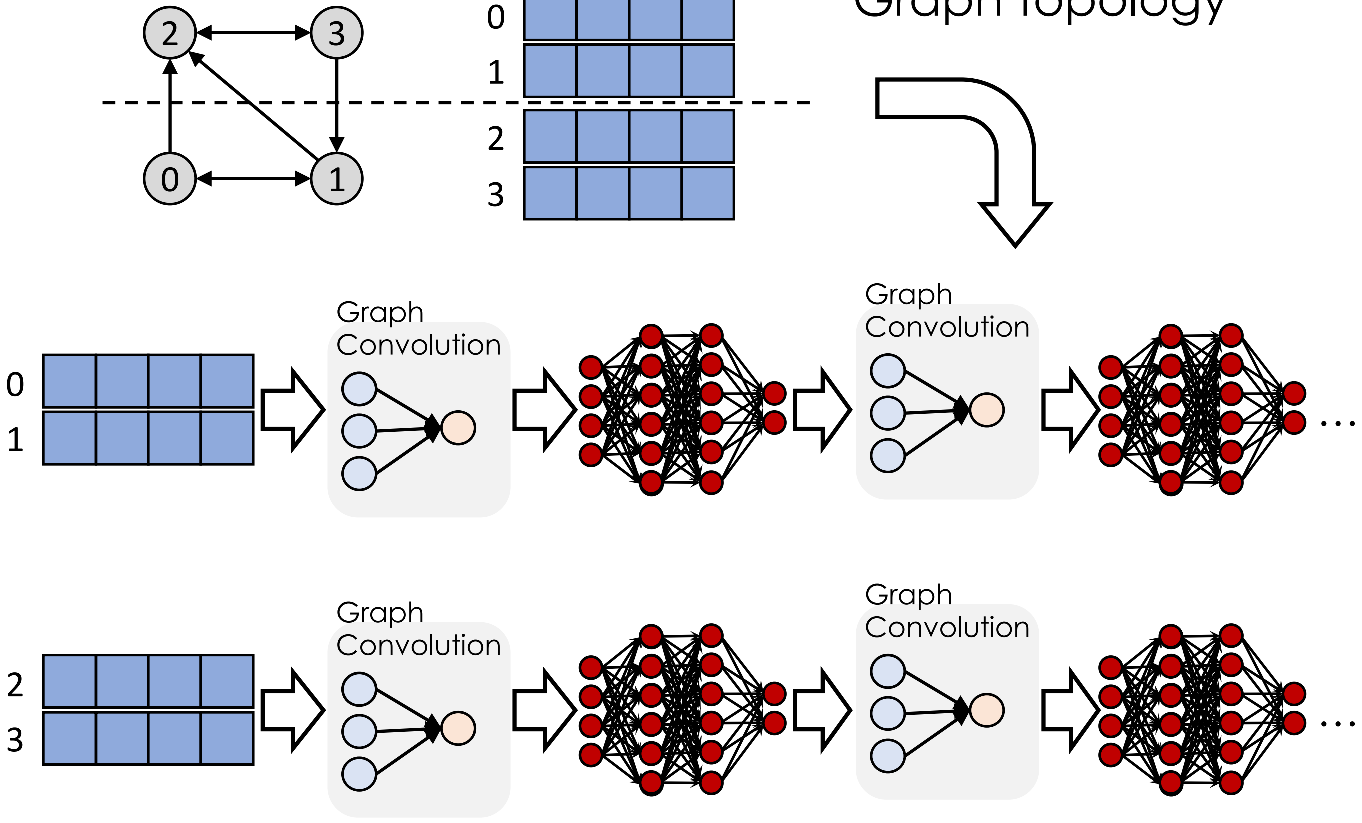
Proteins, Images, Sentences…



GNN training:

Input data:

Node features, Graph topology

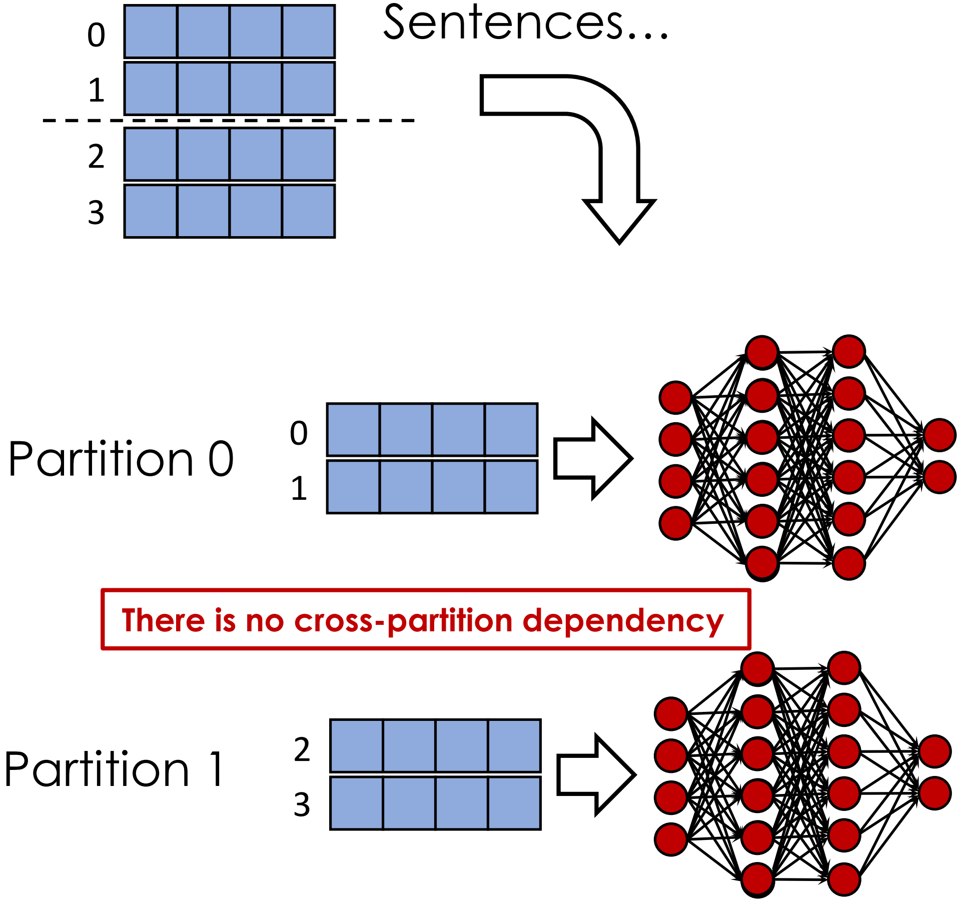


Distributed GNN Training

DNN training:

Input data:

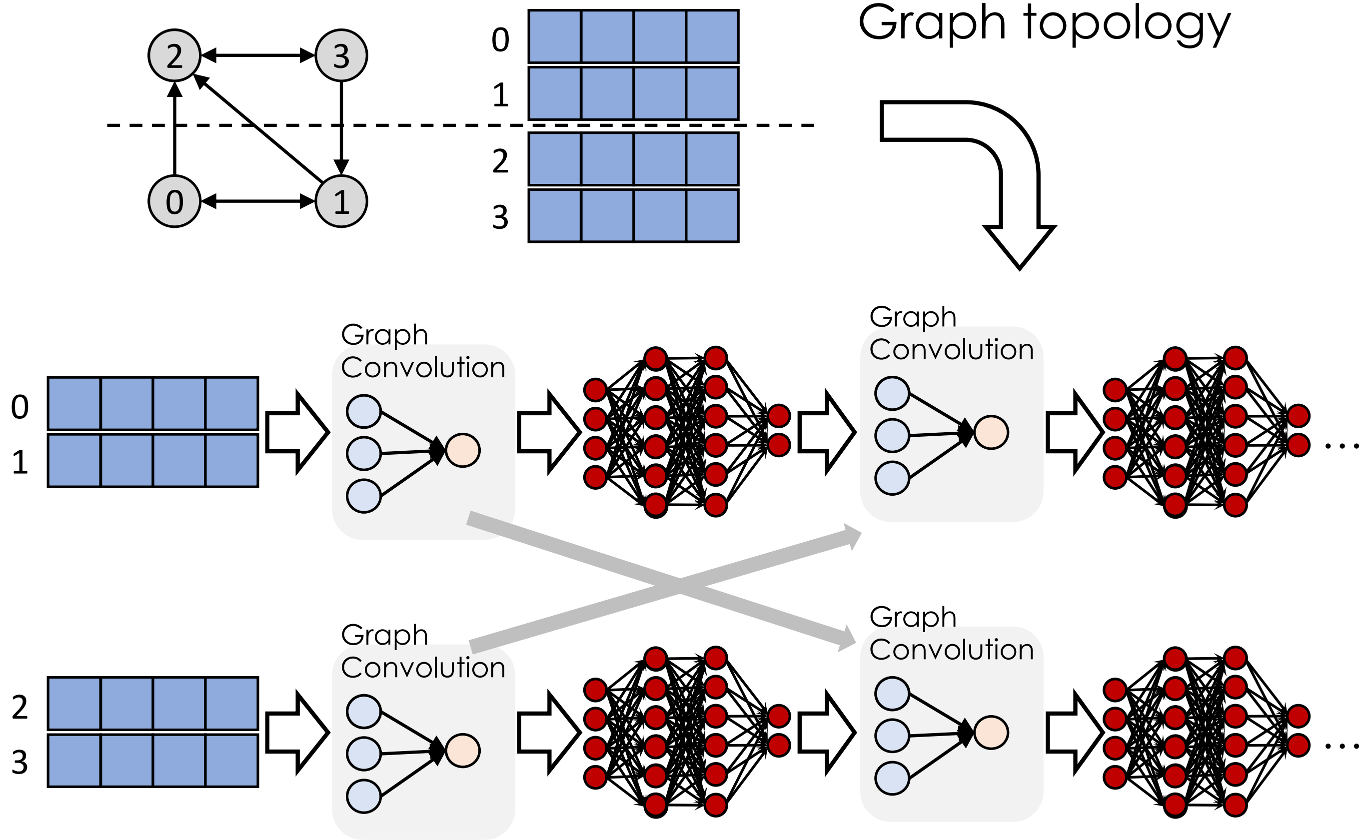
Proteins, Images, Sentences…



GNN training:

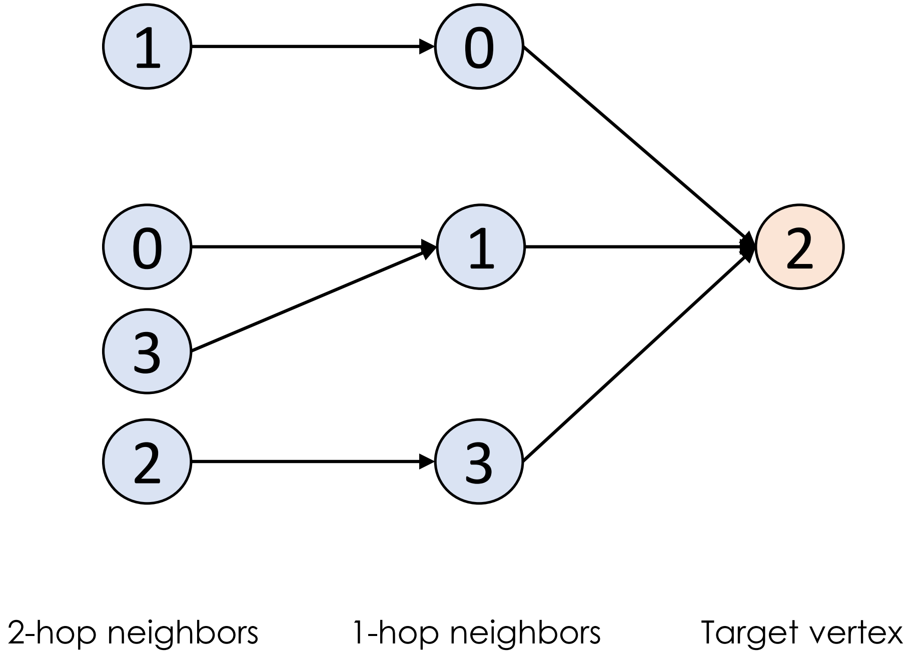
Input data:

Node features, Graph topology

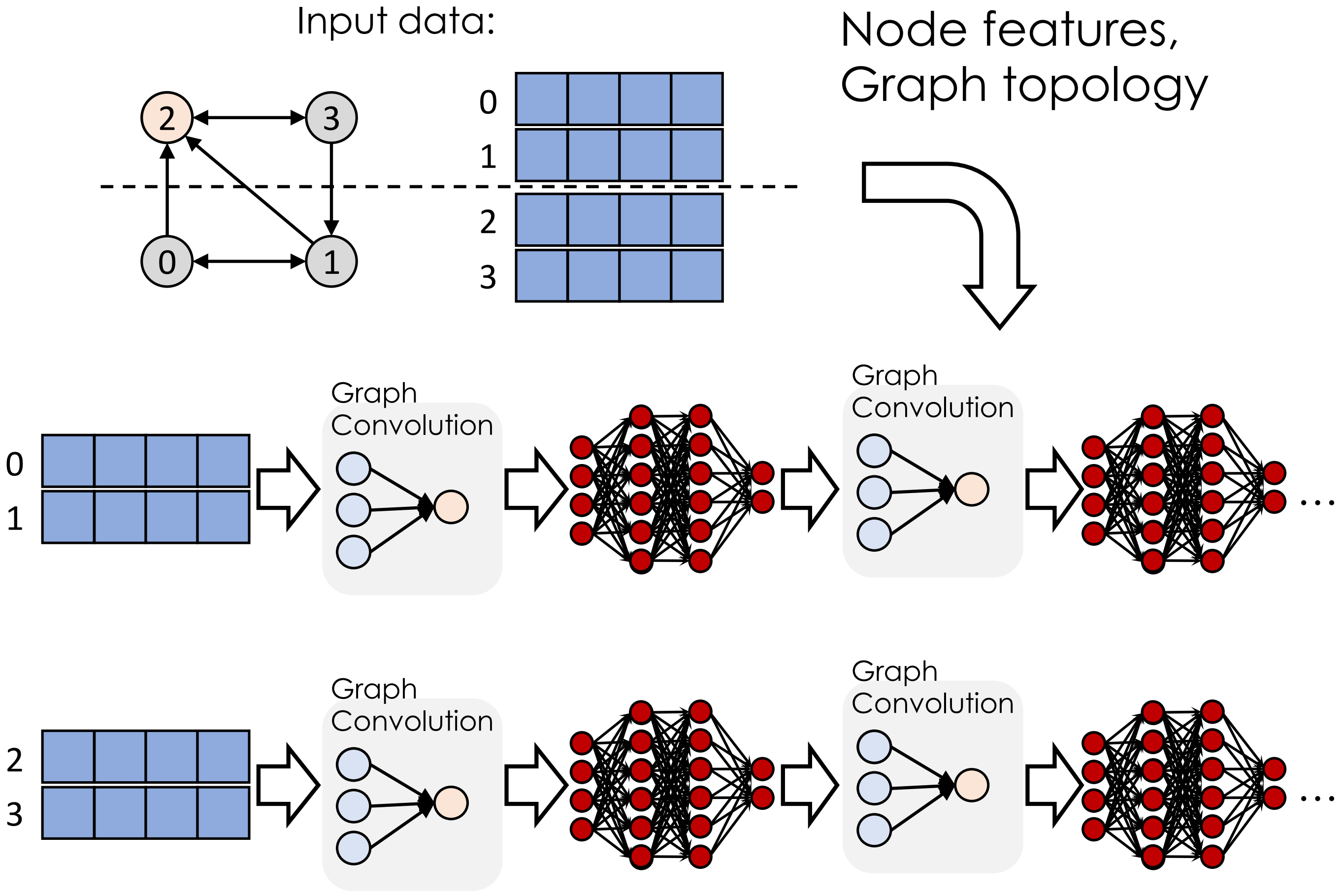


Distributed GNN Training

Dependency tree of node 2:

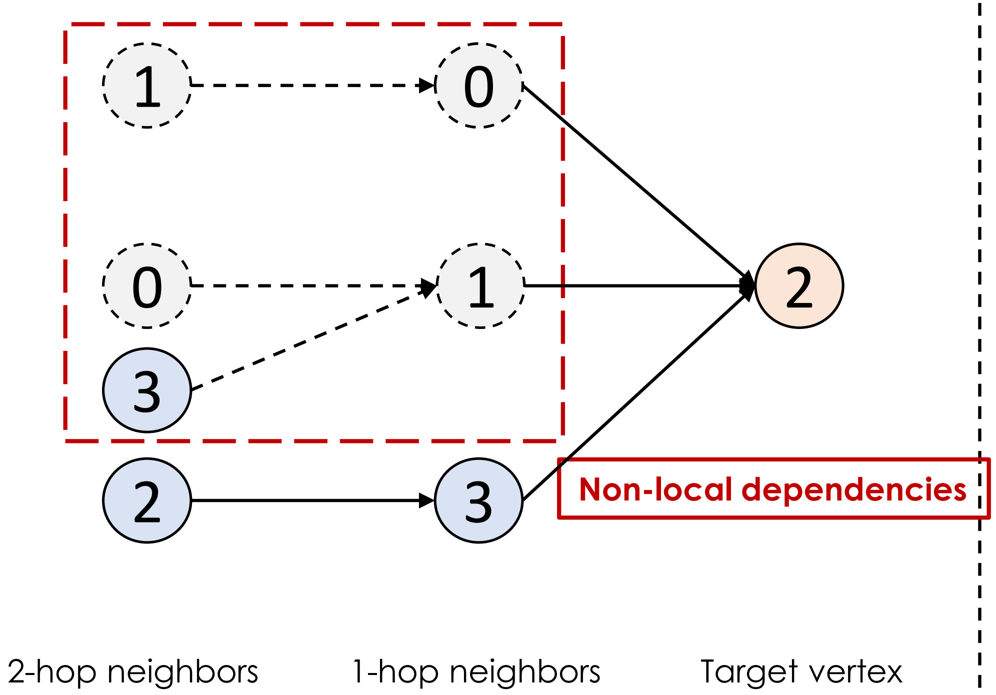


GNN training:

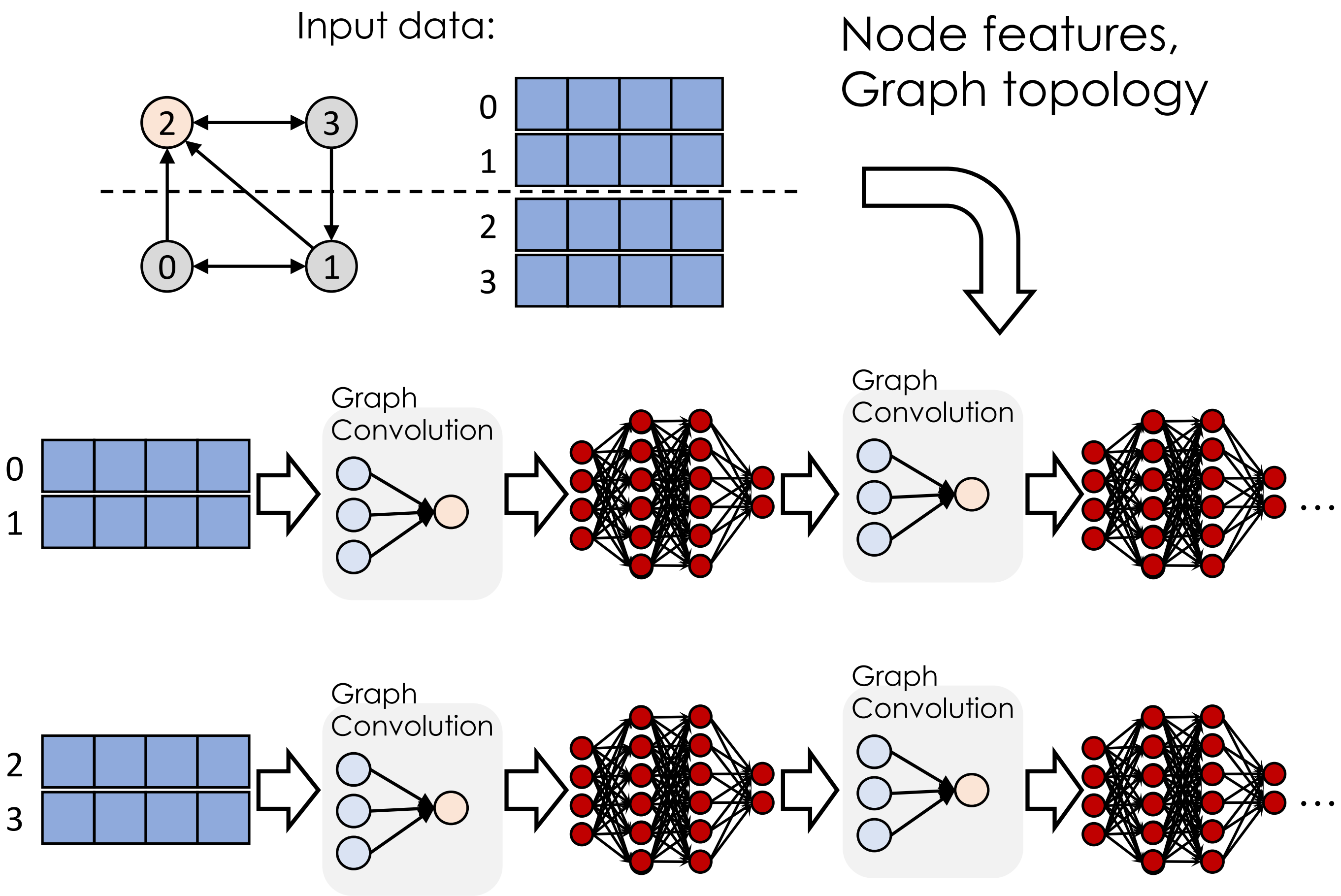


Distributed GNN Training

Dependency tree of node 2:

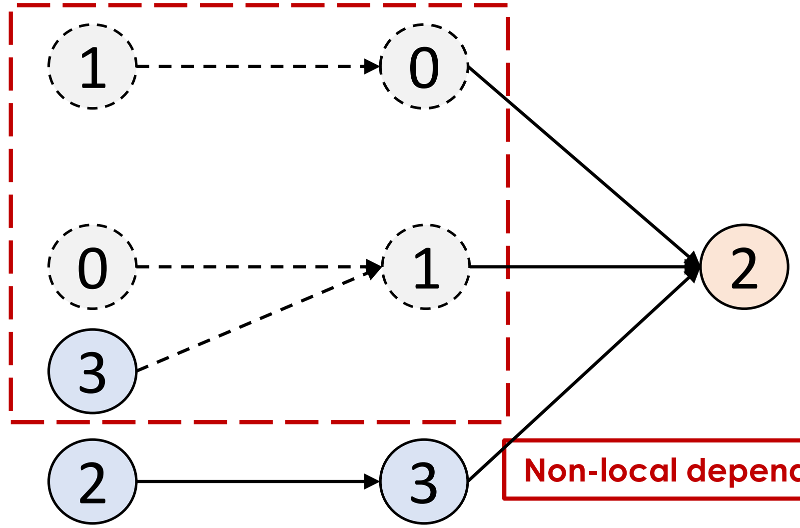


GNN training:



Distributed GNN Training

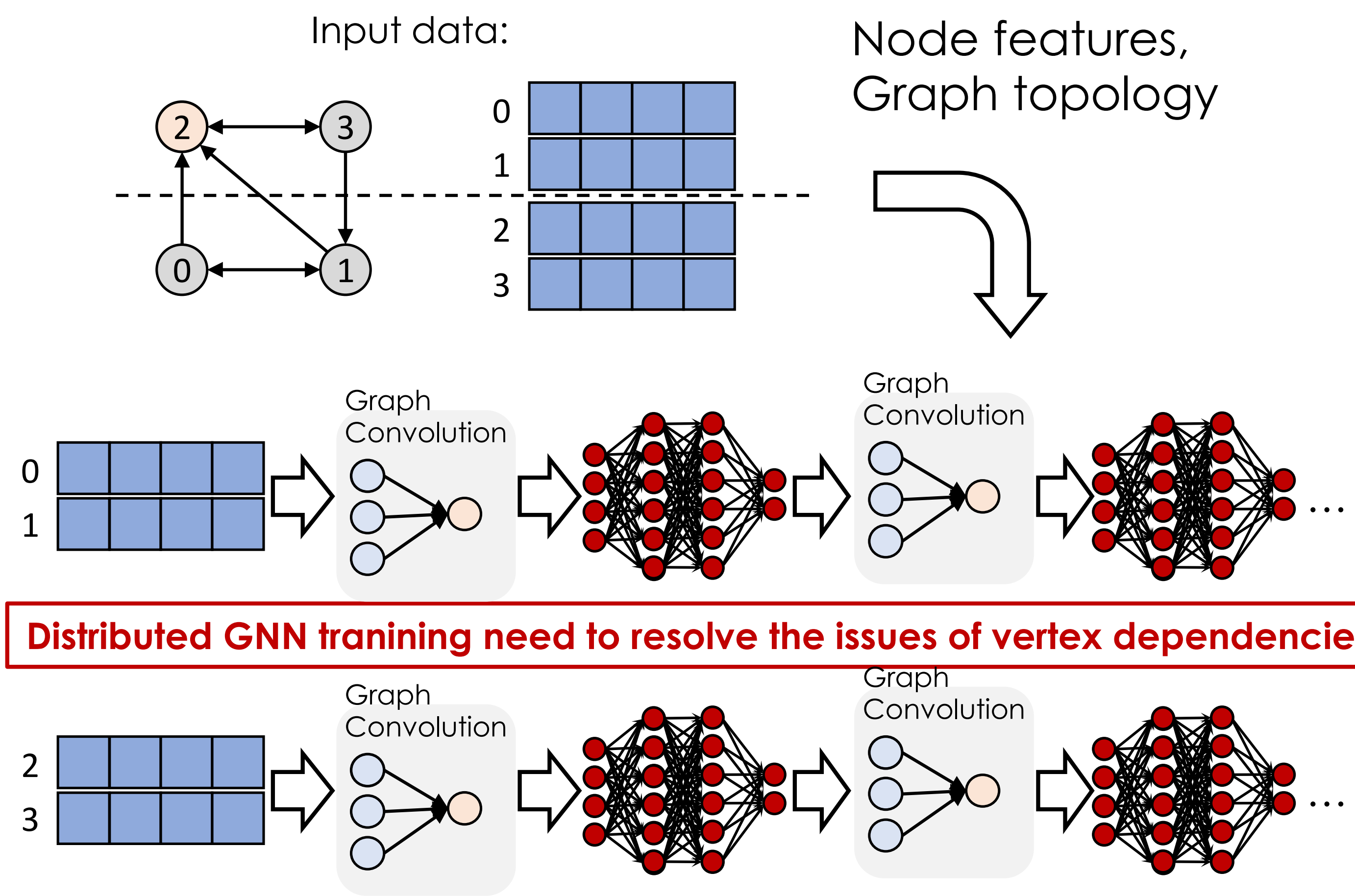
Dependency tree of node 2:



Non-local dependencies

2-hop neighbors 1-hop neighbors Target vertex

GNN training:

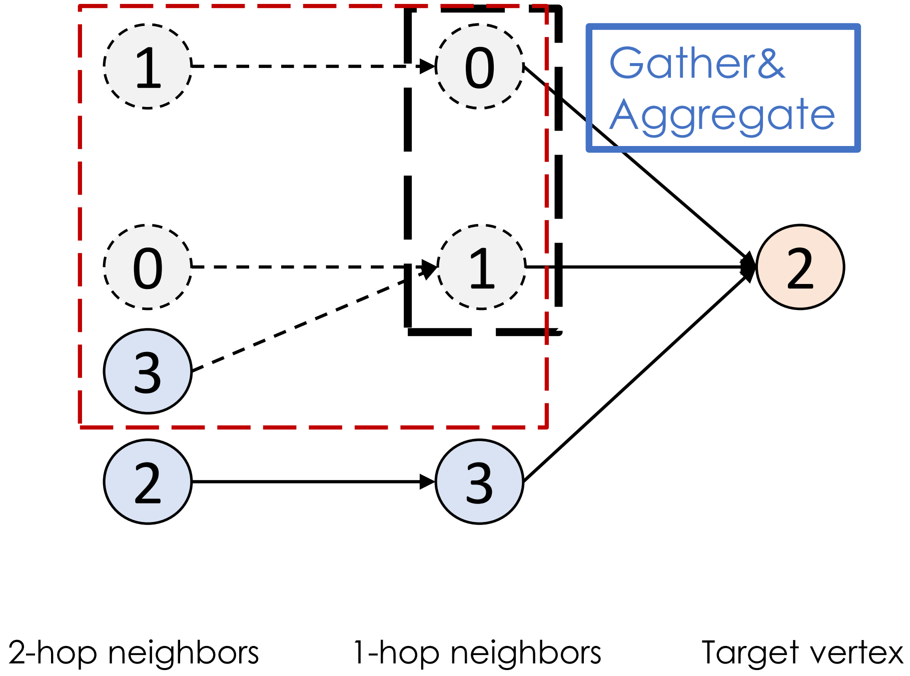


Distributed GNN tranining need to resolve the issues of vertex dependencies

Challenges in Distributed Training

Performance:

Efficiently managing the cross-partition vertex representation.



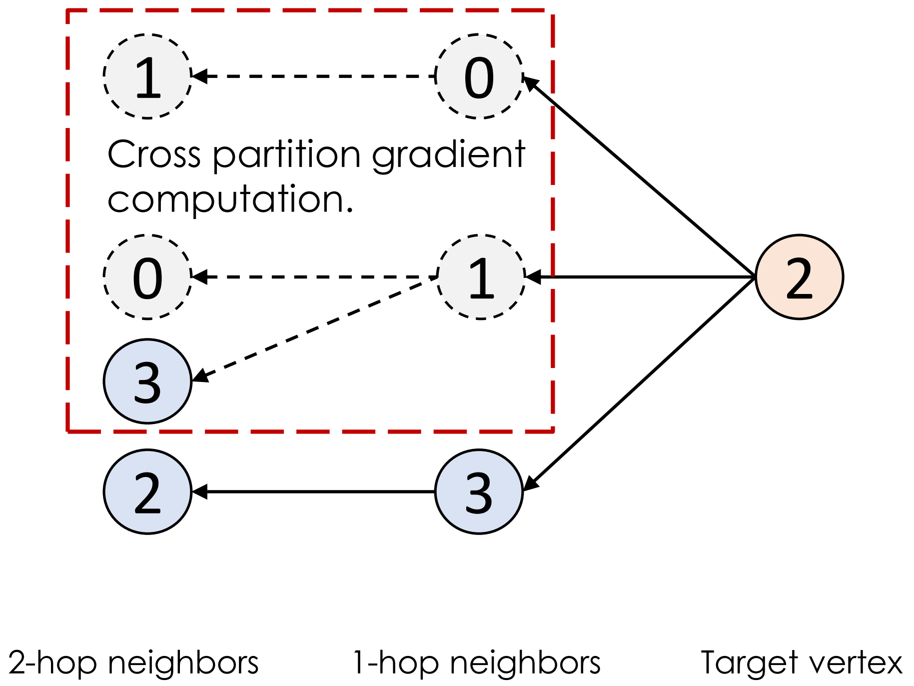
Challenges in Distributed Training

Performance:

Efficiently managing the cross-partition vertex representation.

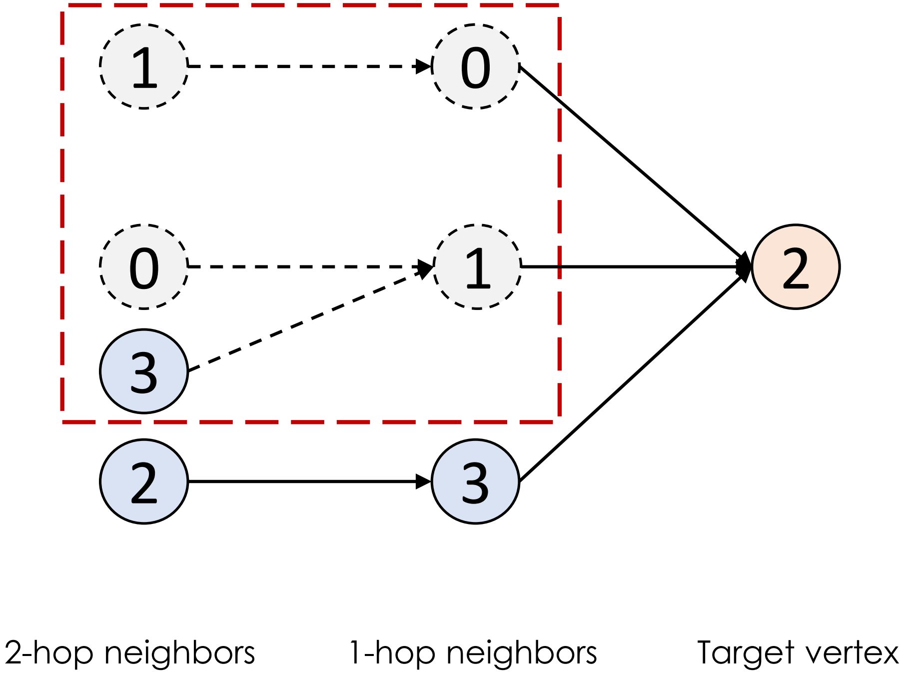
Usability:

Automated cross-partition gradient backward propagation.



Distributed GNN Training

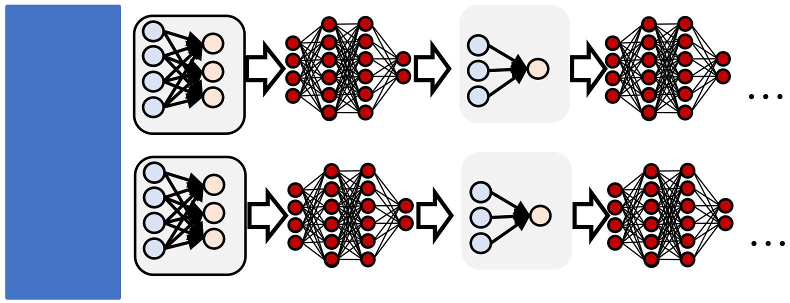
Dependency tree of node 2:



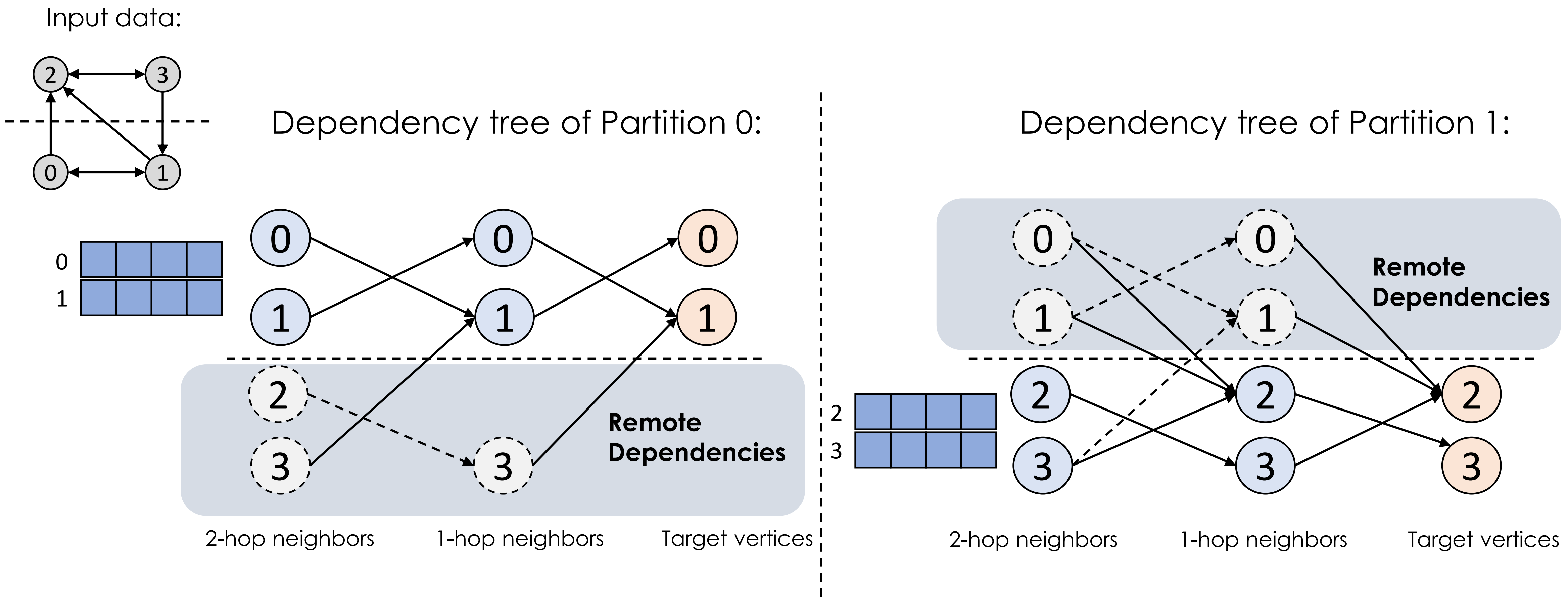
Existing Approaches:

Dependency Cached:

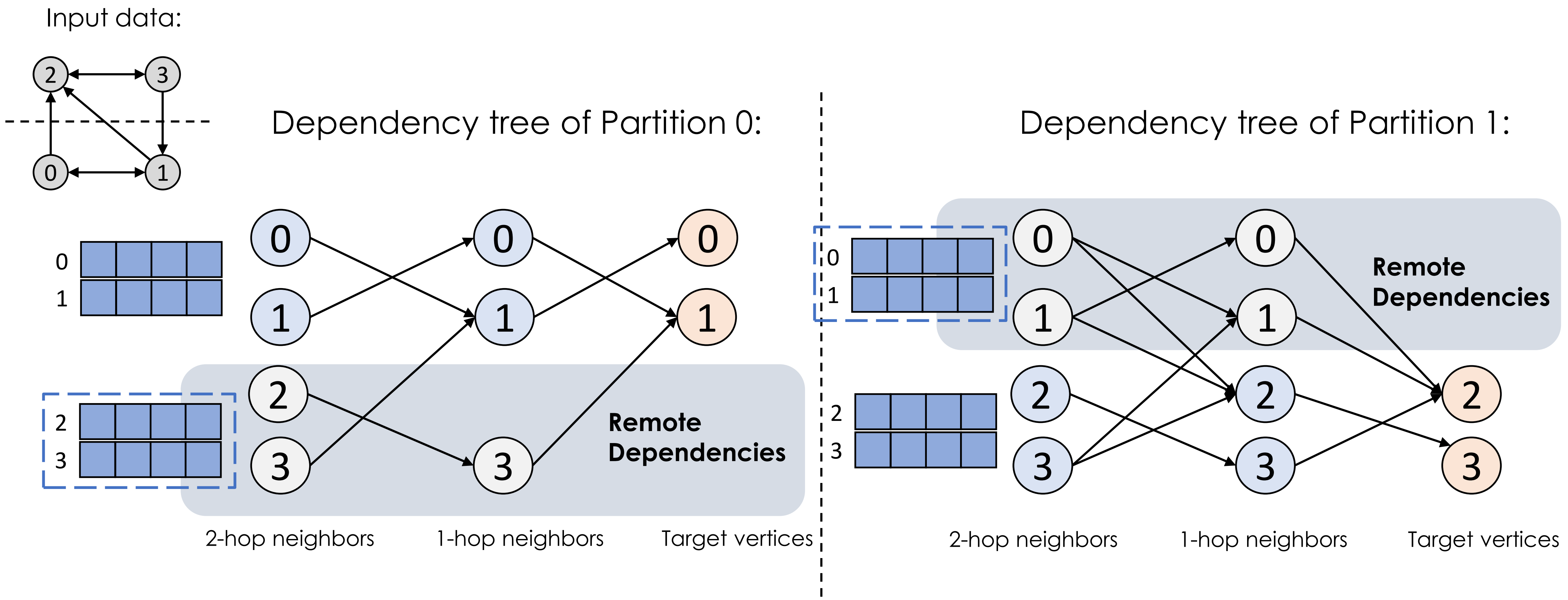
AliGraph[VLDB'20], Euler[arXiv'20], AGL[VLDB'20], DistDGL[arXiv'20]



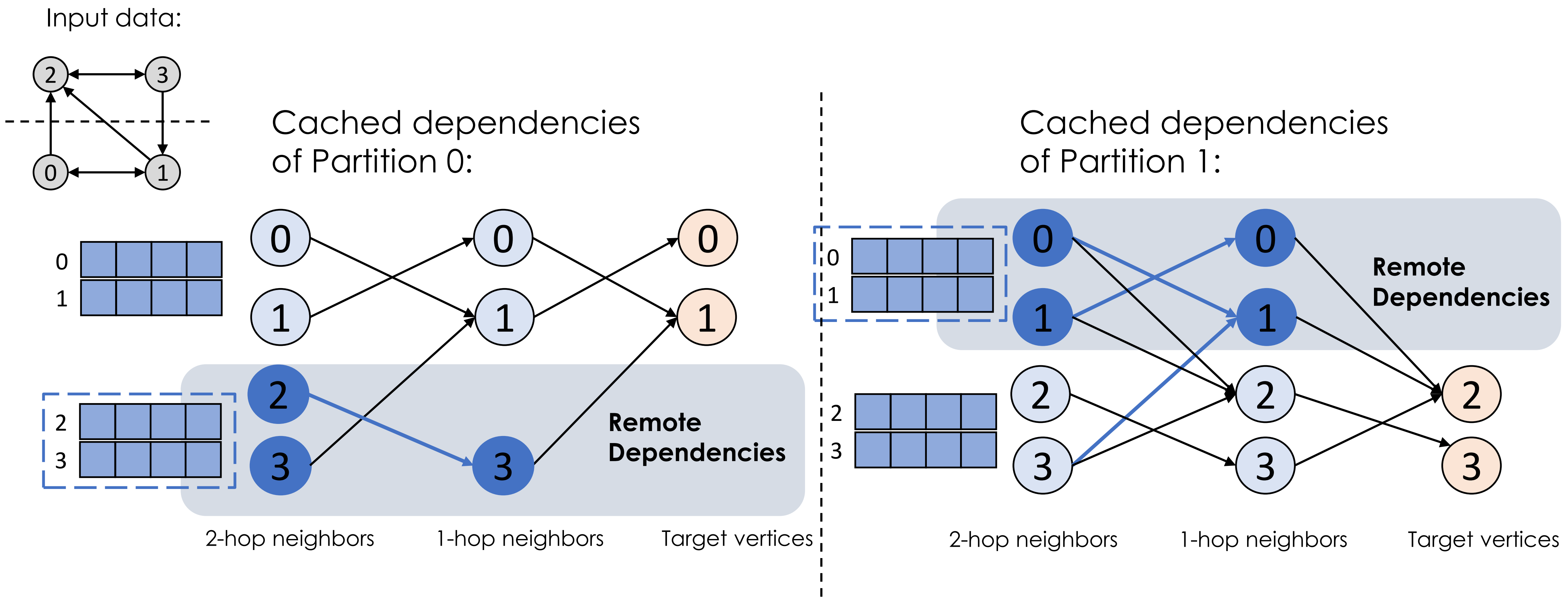
Dependency Cached Approach



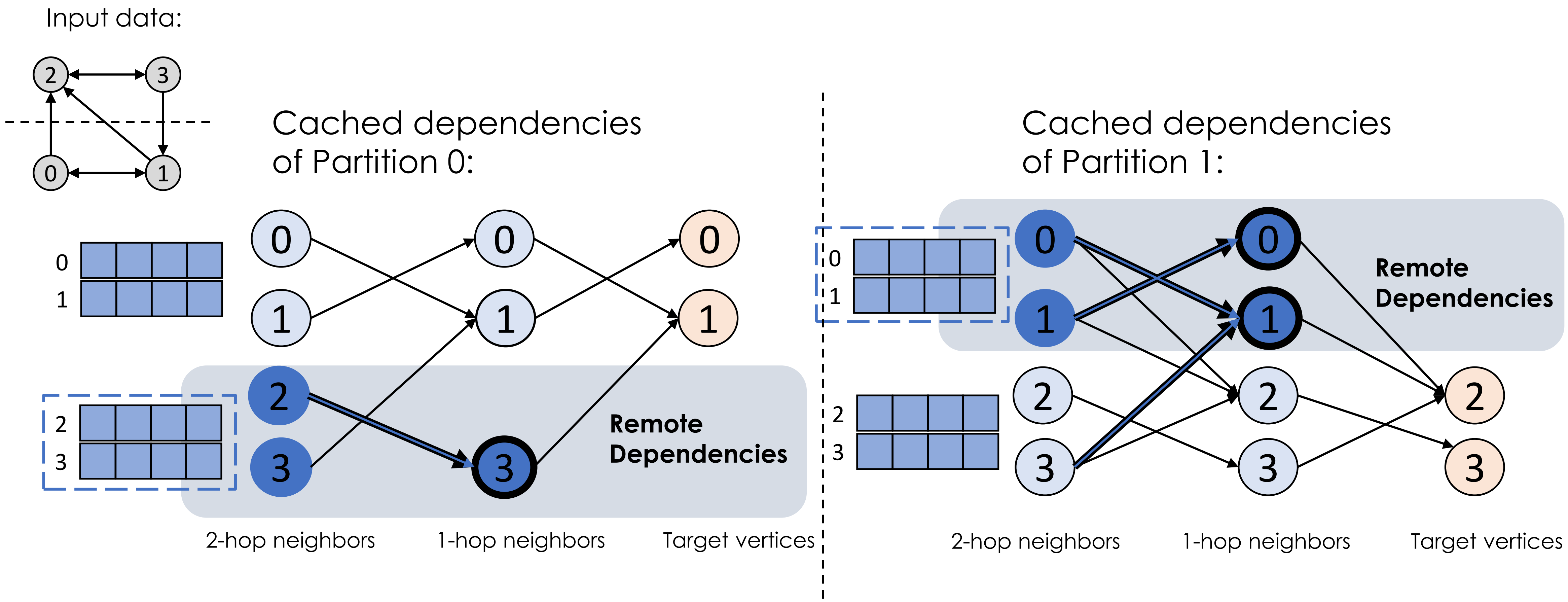
Dependency Cached Approach



Dependency Cached Approach



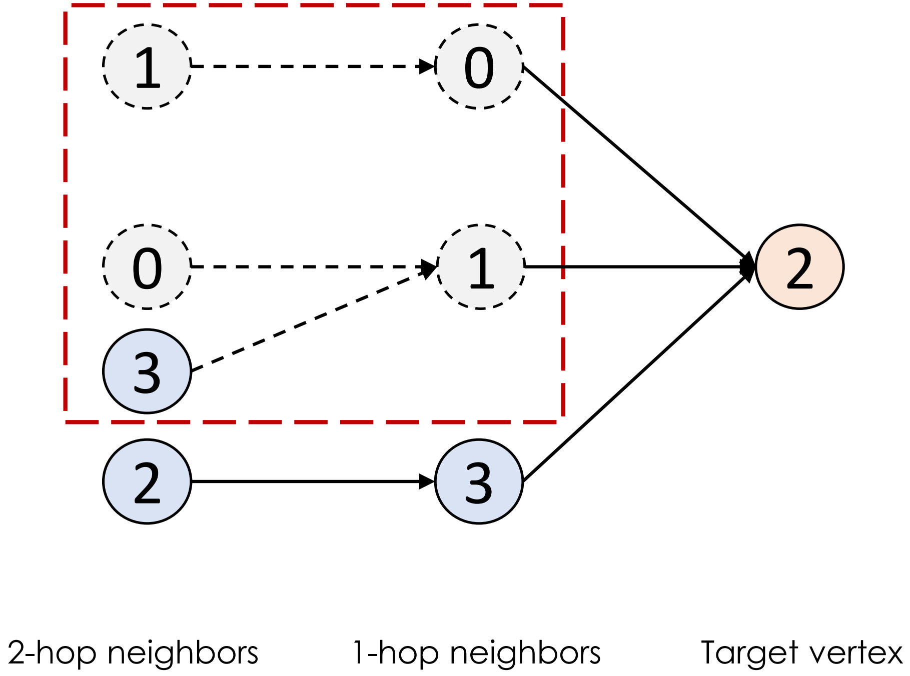
Dependency Cached Approach



Redundant computation problem

Distributed GNN Training

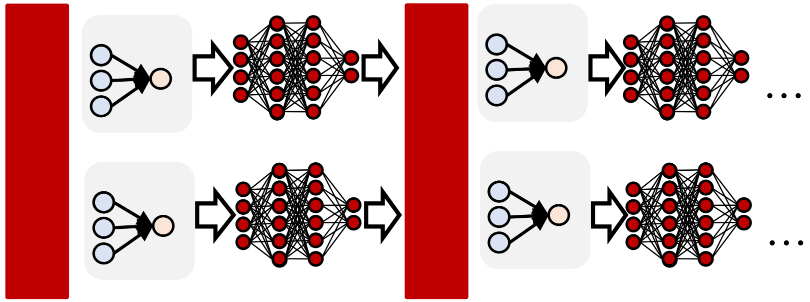
Dependency tree of node 2:



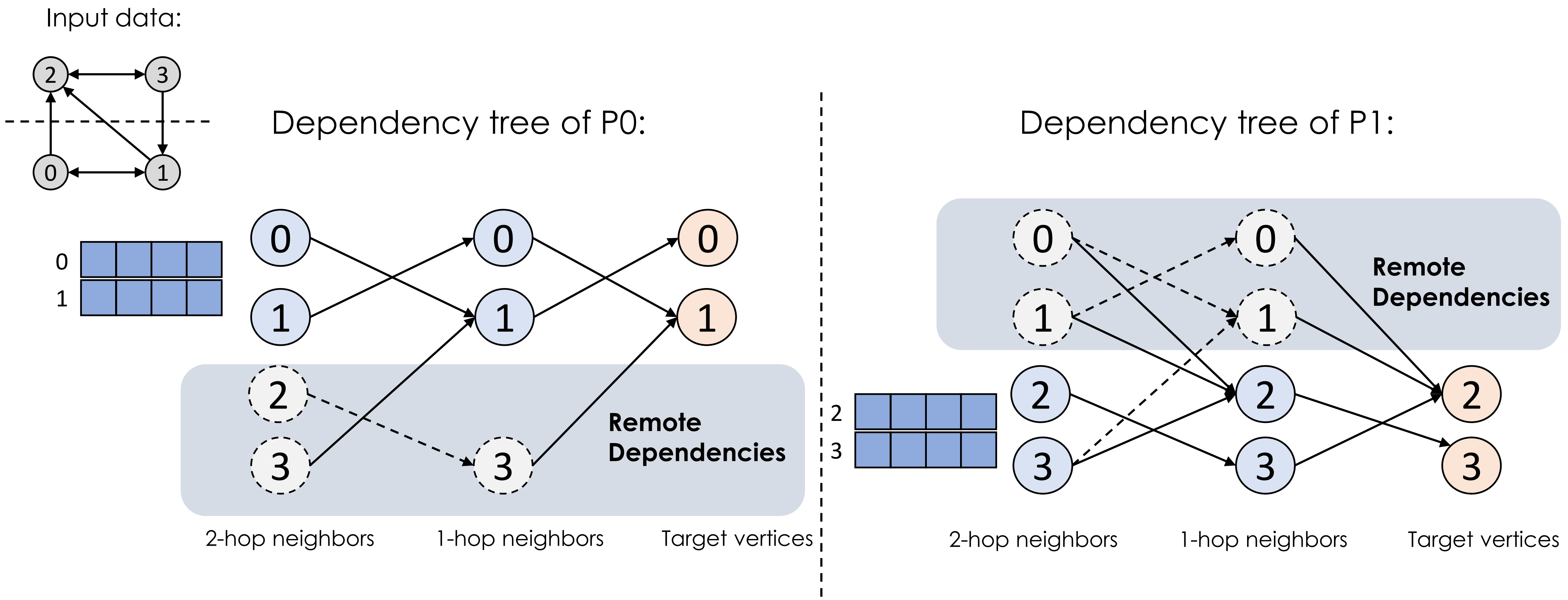
Existing Approaches:

Dependency Communicated:

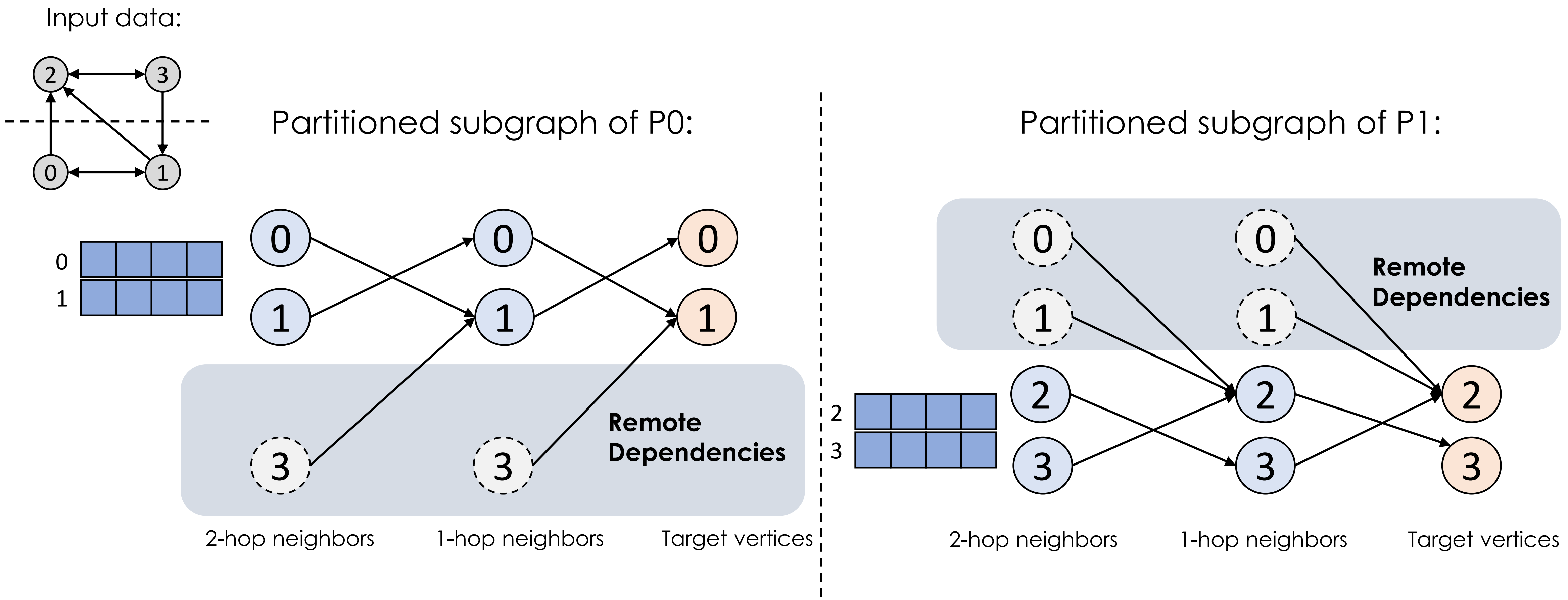
ROC[MLSYS'20], Dorylus[OSDI'21], CAGNET[SC'20], DistGNN[SC'21], DGCL[EUROSYS'21].



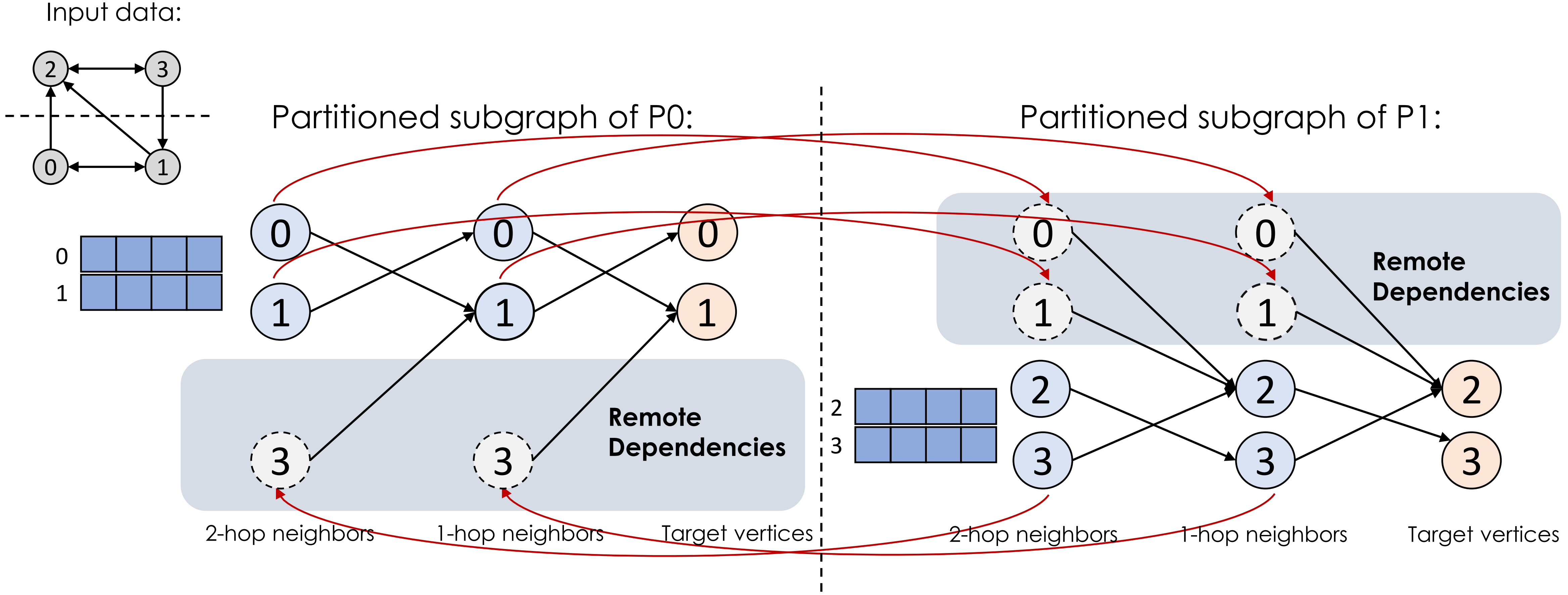
Dependency Communicated Approach



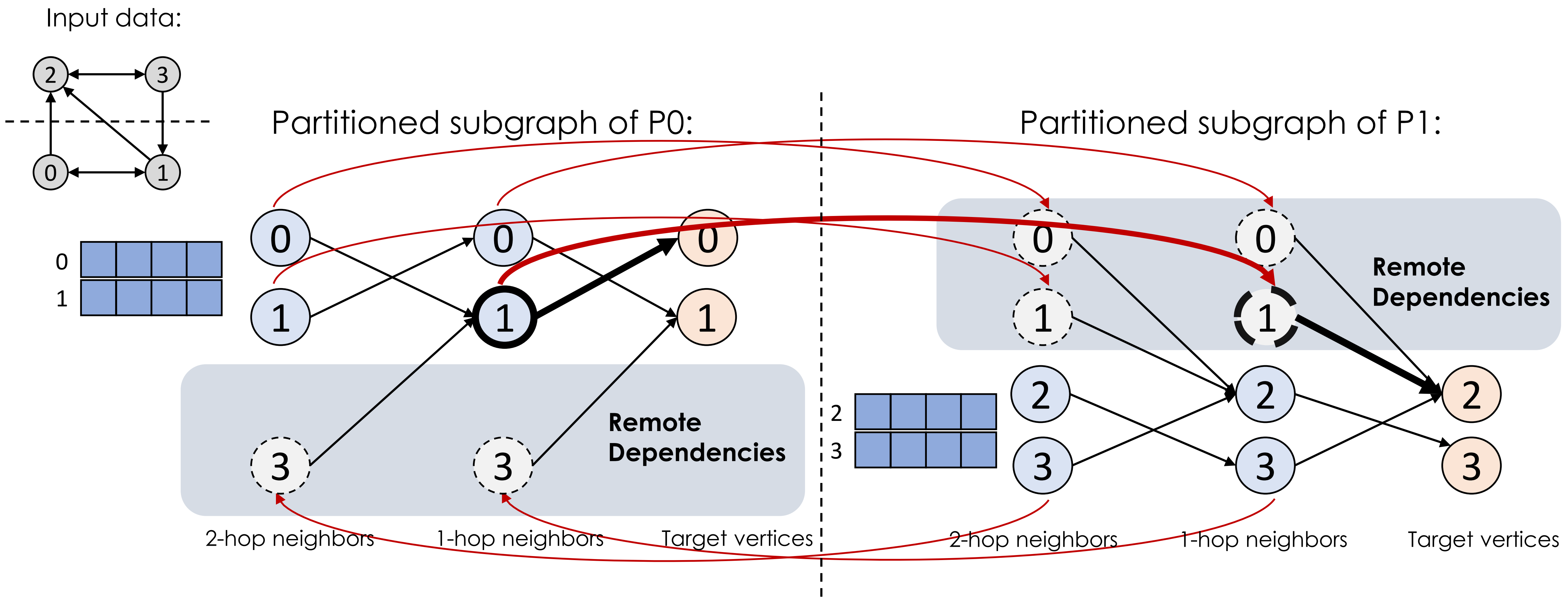
Dependency Communicated Approach



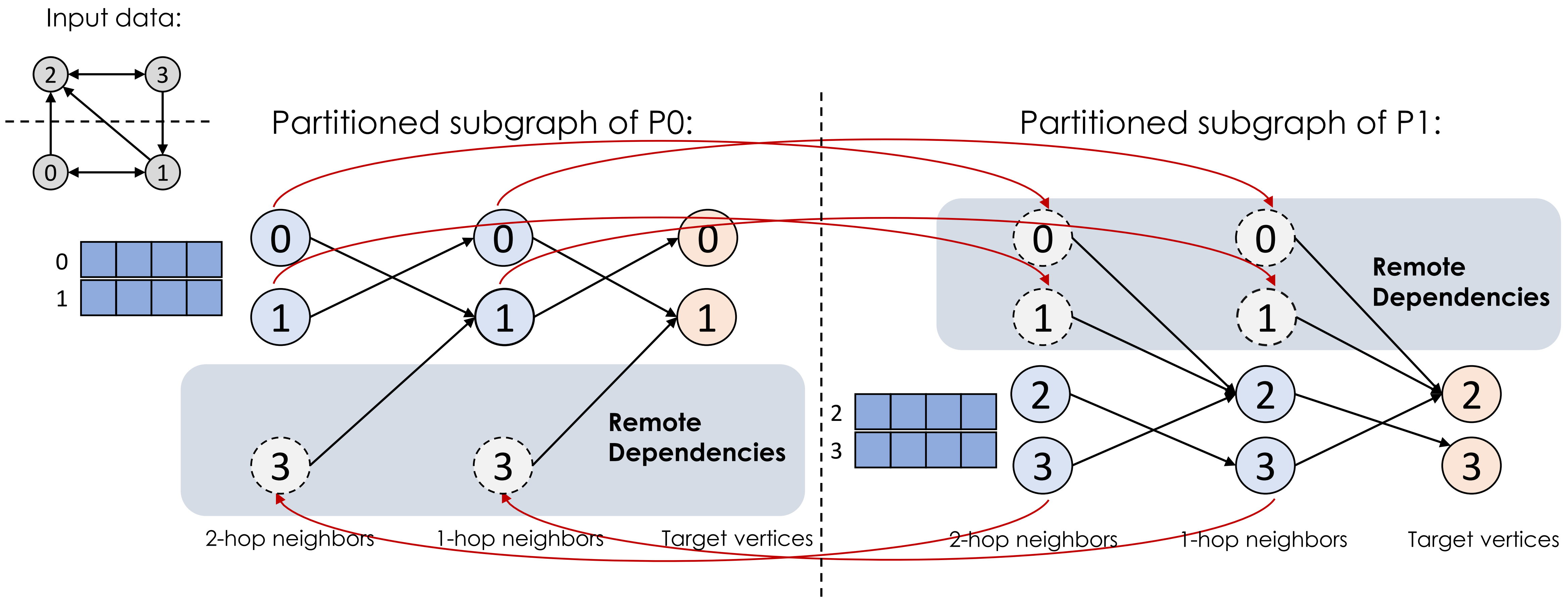
Dependency Communicated Approach



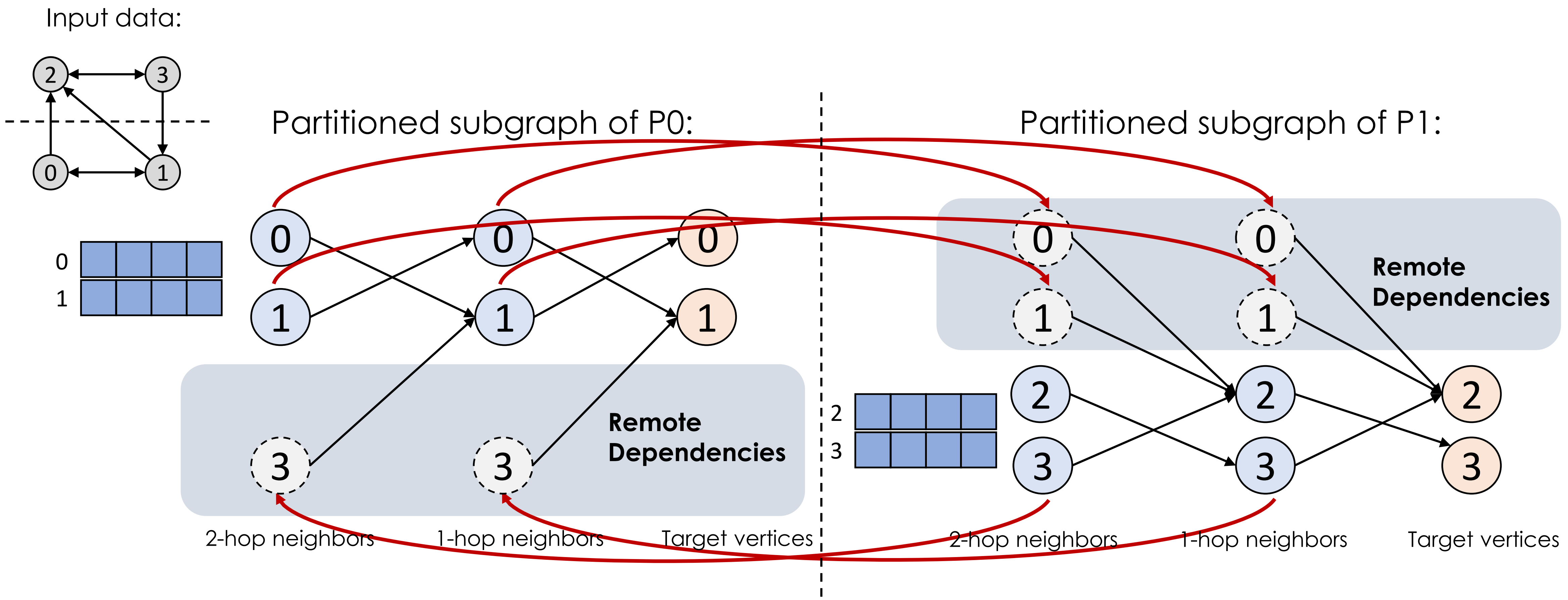
Dependency Communicated Approach



Dependency Communicated Approach

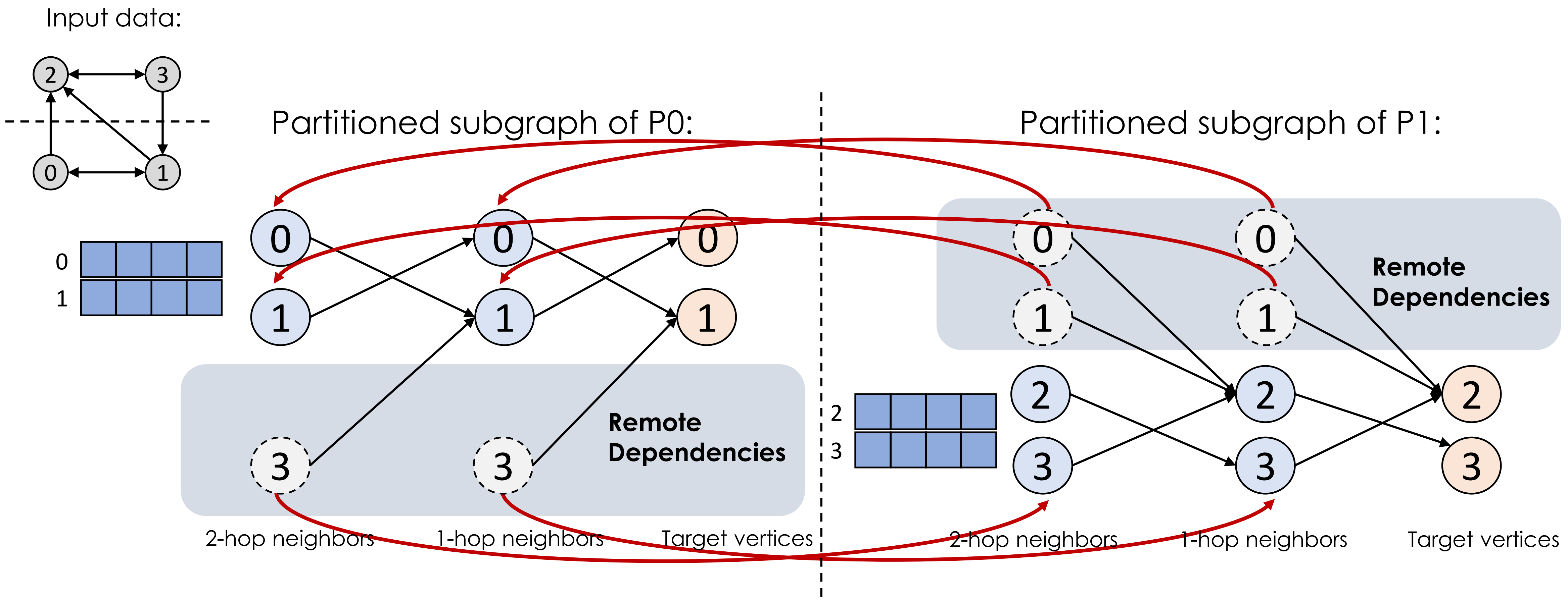


Dependency Communicated Approach



Frequent cross-worker communication

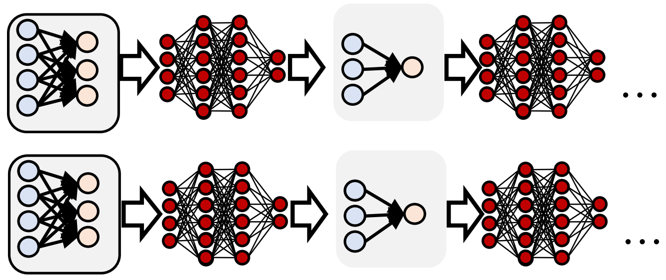
Dependency Communicated Approach



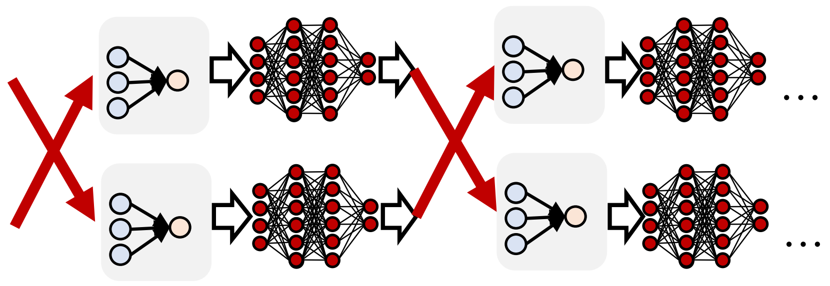
Frequent cross-worker communication

Comparison of the Two Approaches

Dependency Cached:



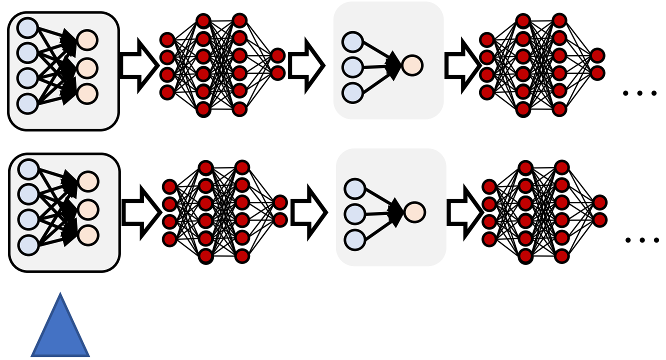
Dependency Communicated:



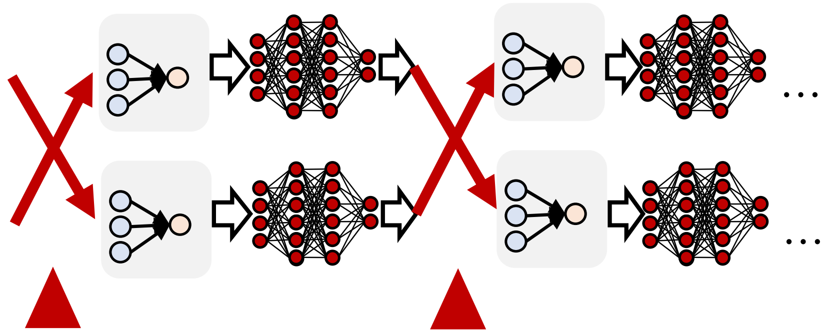
The performance of **DepCache** and **DepComm** is dominated by the cost of (1) **redundant computation** and (2) **communication**.

Comparison of the Two Approaches

Dependency Cached:

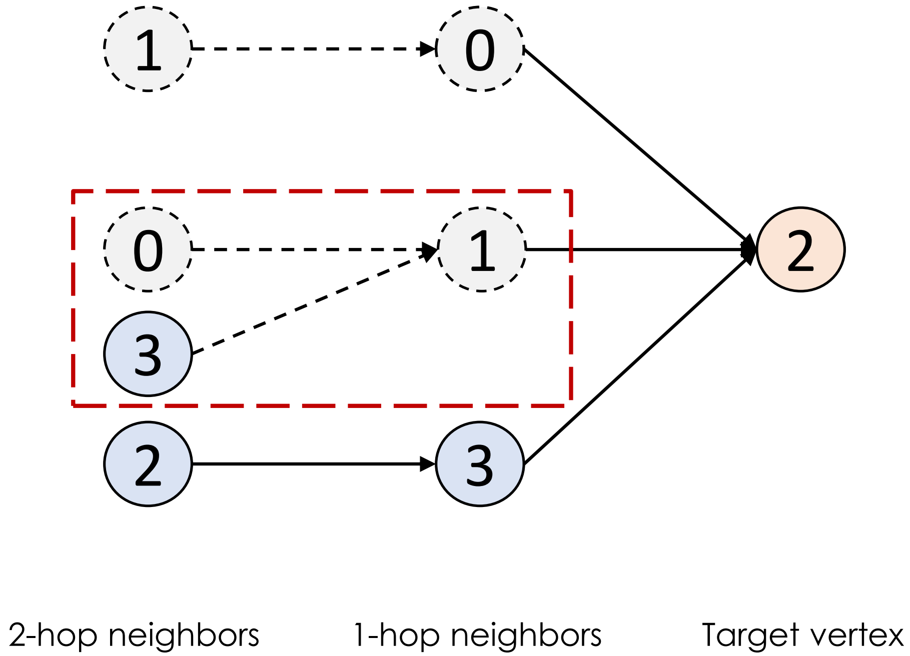


Dependency Communicated:



The performance of **DepCache** and **DepComm** is dominated by the cost of (1) **redundant computation** and (2) **communication**.

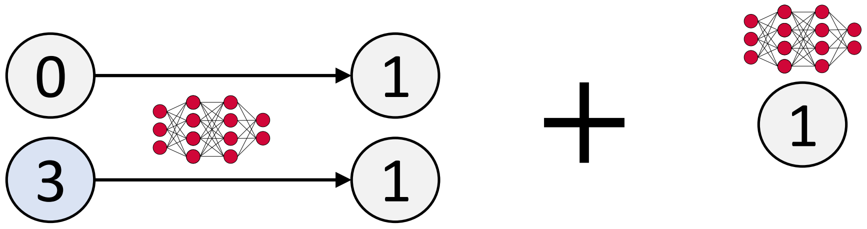
Cost of the Two Approaches



Cost of DepCache:

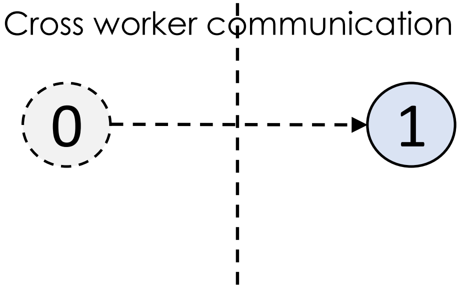
Graph convolution overhead:

Vertex computation overhead:

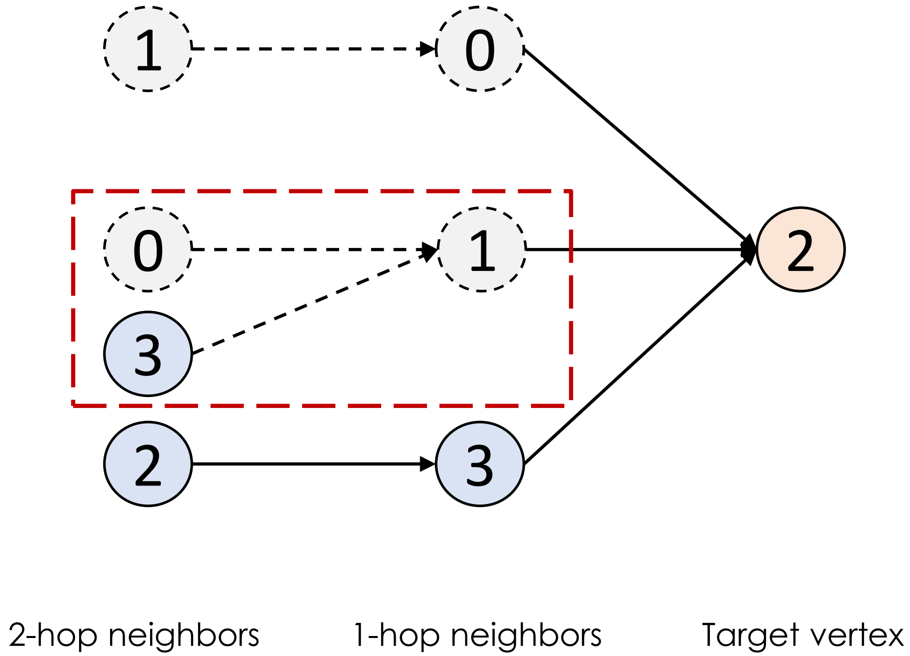


Cost of DepComm:

Cross worker communication overhead



Cost of the Two Approaches

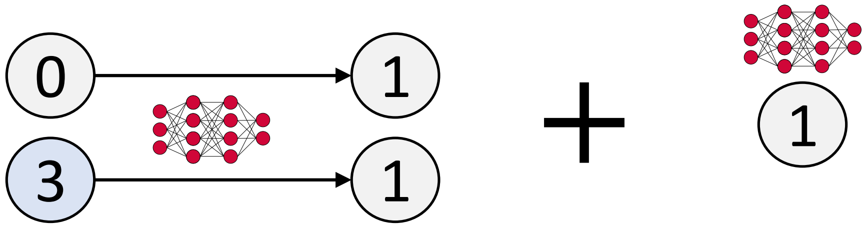


(1) Graph inputs
(2) Model configurations
(3) Environment configurations

Cost of DepCache:

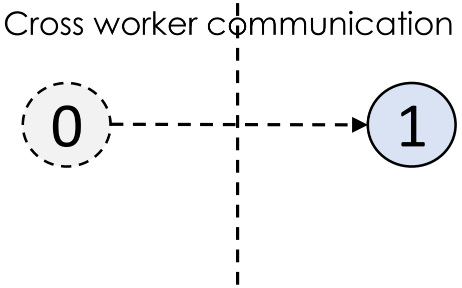
Graph convolution overhead:

Vertex computation overhead:



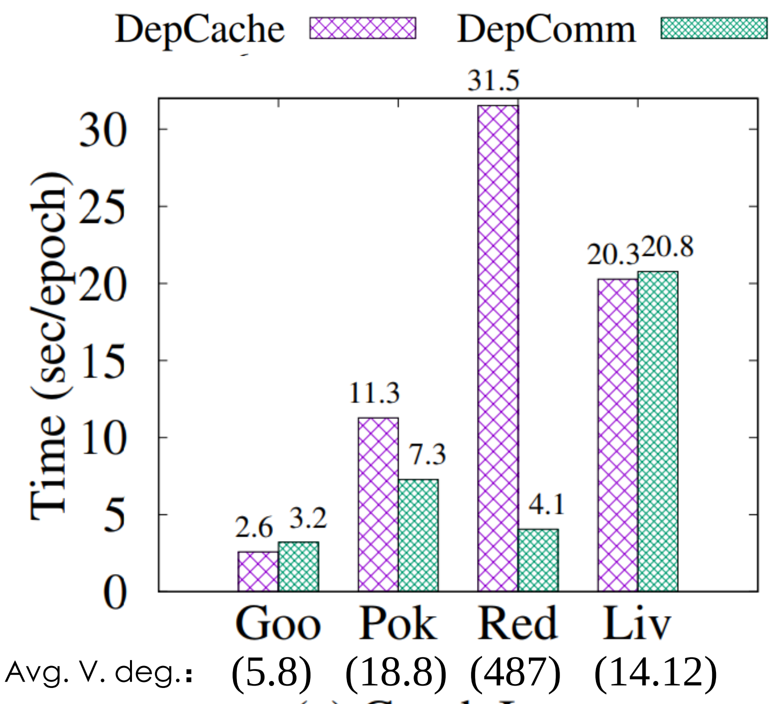
Cost of DepComm:

Cross worker communication overhead

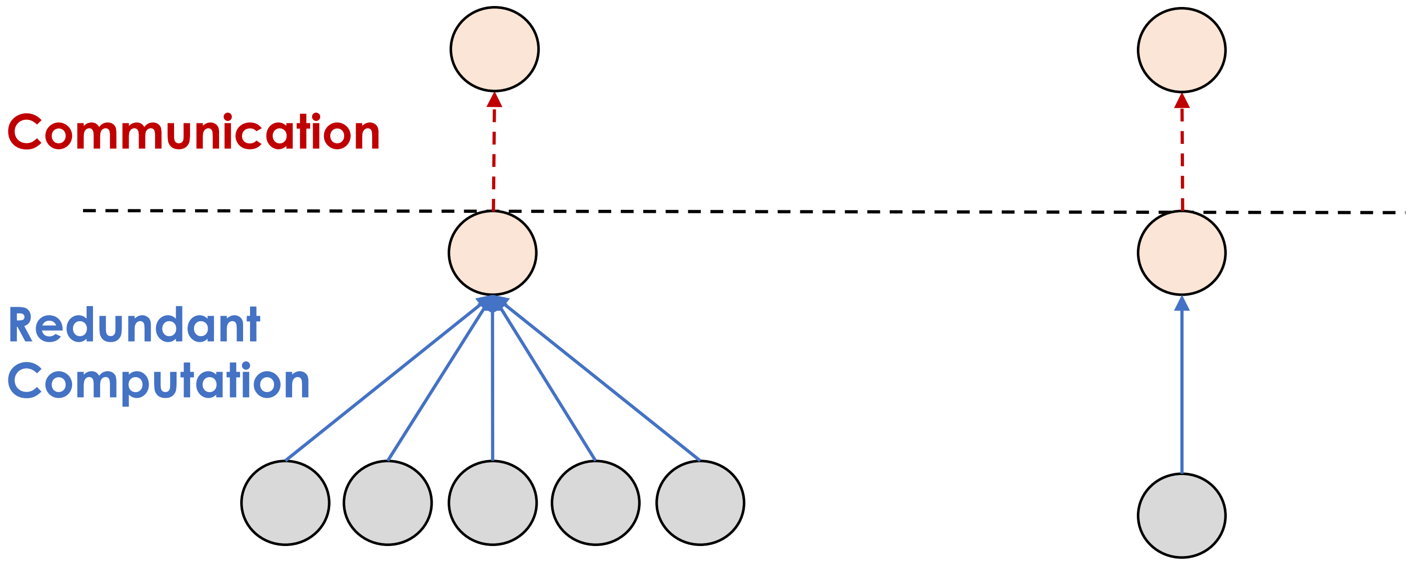


Comparison of the Two Approaches (1)

Graph inputs (the vertex degree):



(a) Graph Inputs

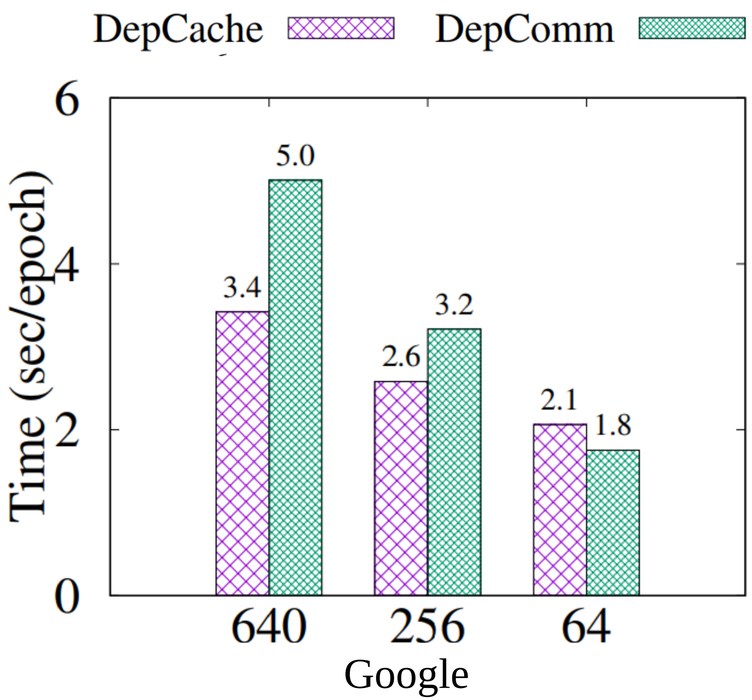


DepComm is effective to high-degree vertices

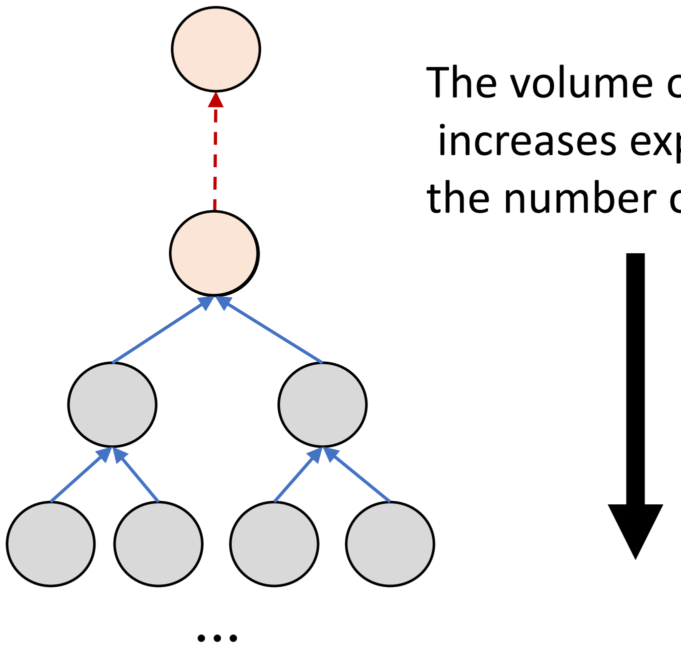
DepCache is effective to low-degree vertices

Comparison of the Two Approaches (2)

Model configurations (Hidden layer sizes):



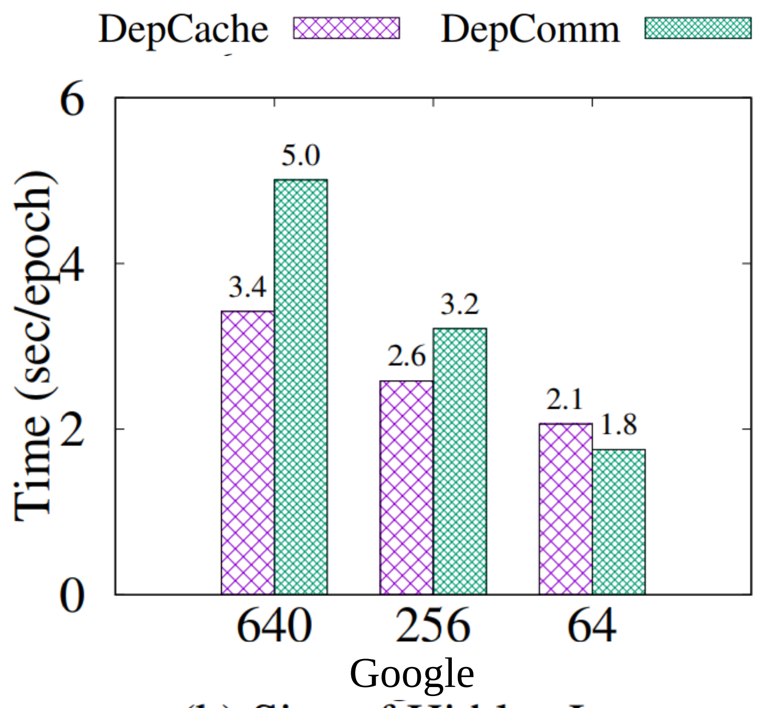
(b) Size of Hidden Layer



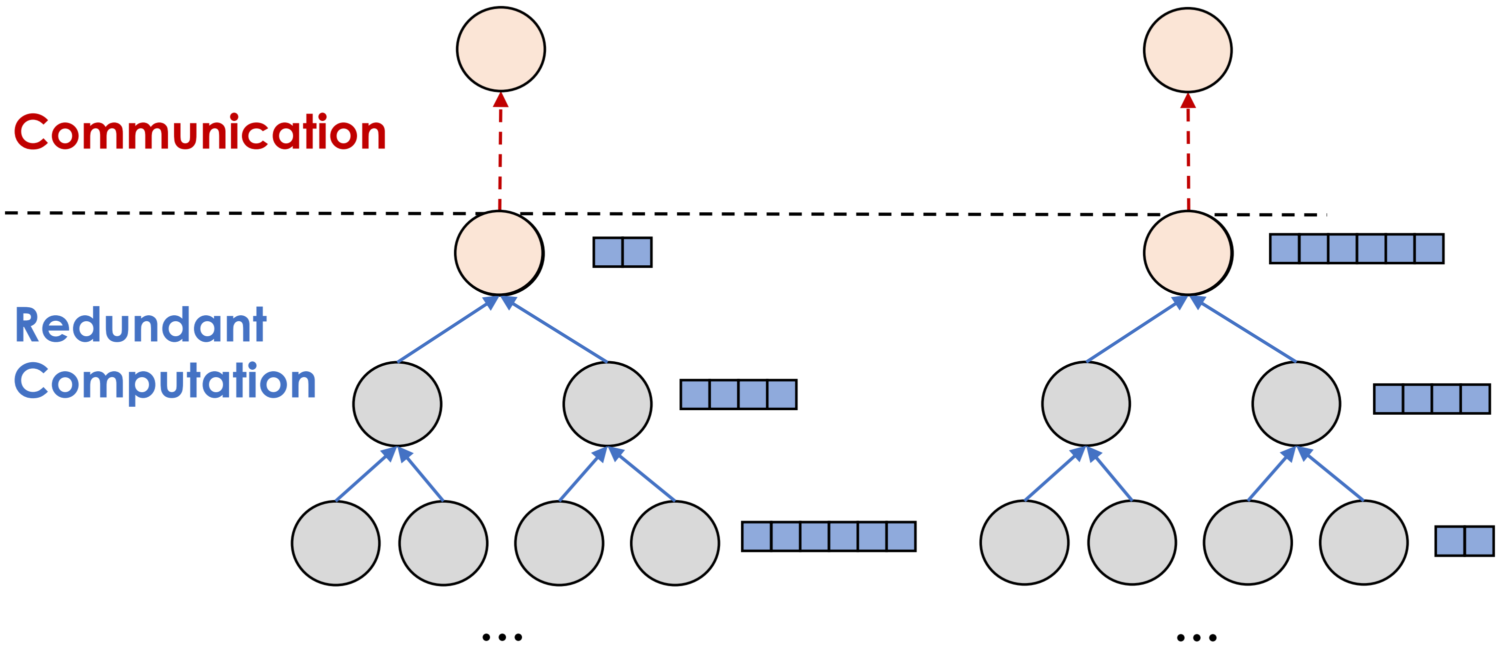
The volume of vertex dependencies increases exponentially with the number of hops

Comparison of the Two Approaches (2)

Model configurations (Hidden layer sizes):



(b) Size of Hidden Layer

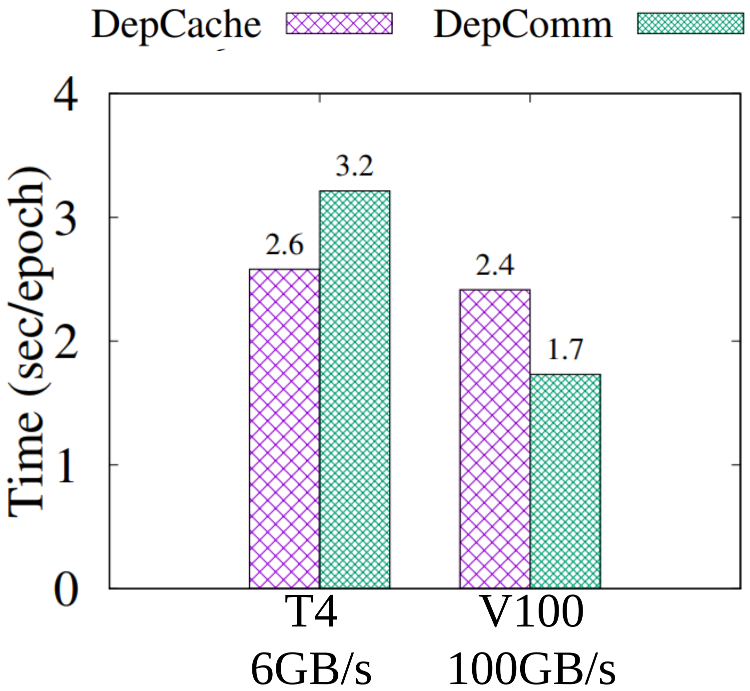


DepComm is effective to large hidden layer size

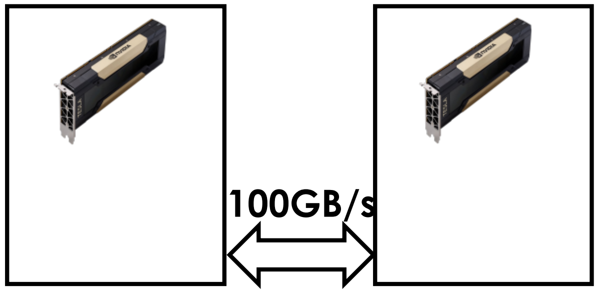
DepCache is effective to small hidden layer size

Comparison of the Two Approaches (3)

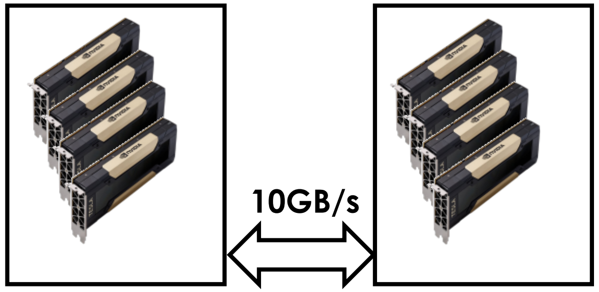
Cluster Configurations (Computing power and network bandwidth):



(c) Cluster Environments

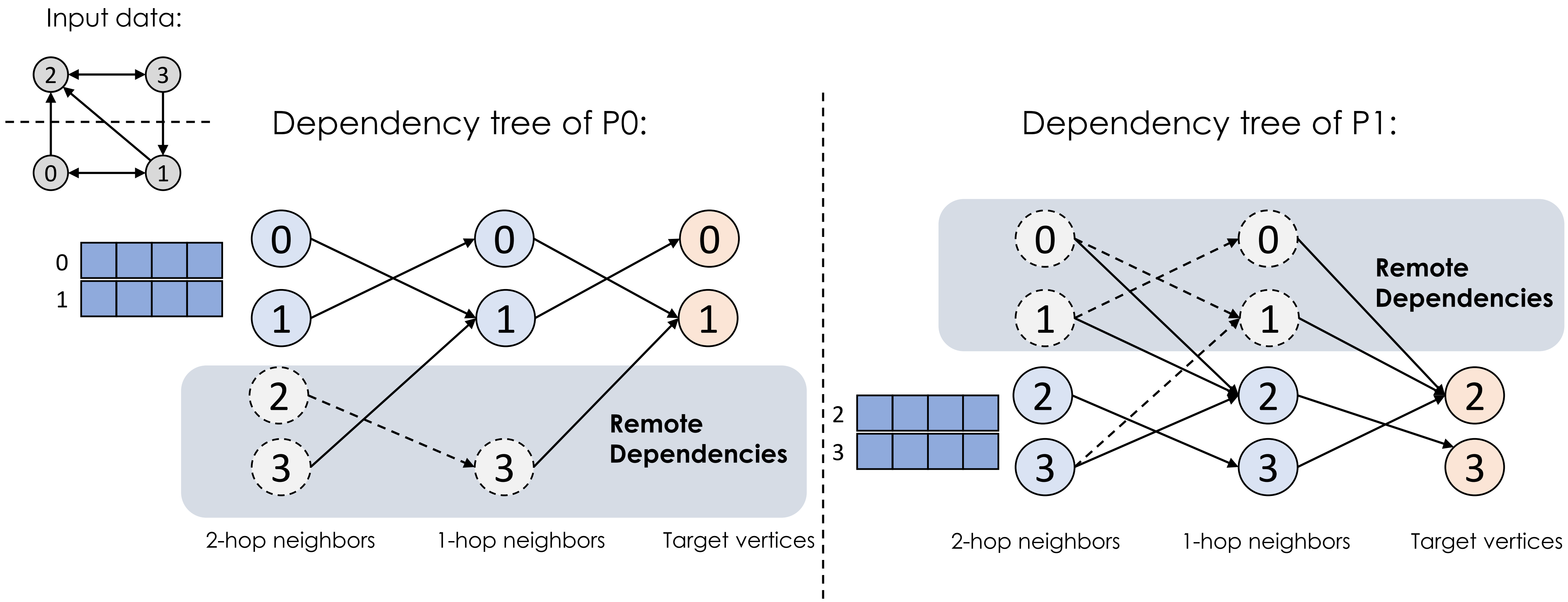


DepComm is effective to high network bandwidth clusters



DepCache is effective to high computing power clusters

Hybrid Dependency Management



We evaluating the cost of **DepCache** and **DepComm** for each dependent neighbor.

Hybrid Dependency Management

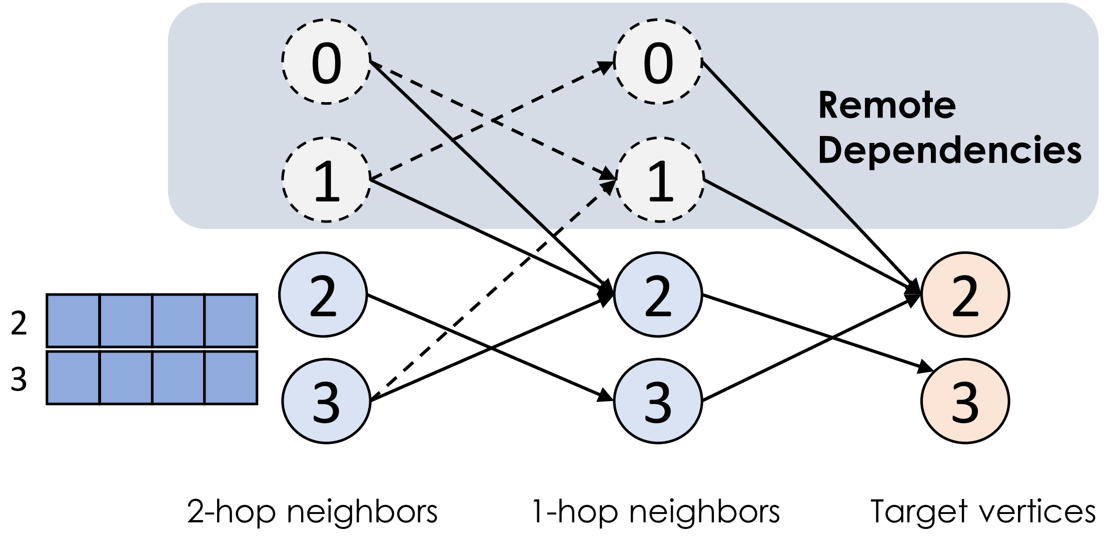
cost of
DepCache:

$$t_r(v) = \sum_{l=0}^{L} \left(|E^l(v)| * T_e + Nbr^l(v)| * T_v \right) * dim^l$$

cost of
DepComm:

$$t_c(v) = dim^l * T_c$$

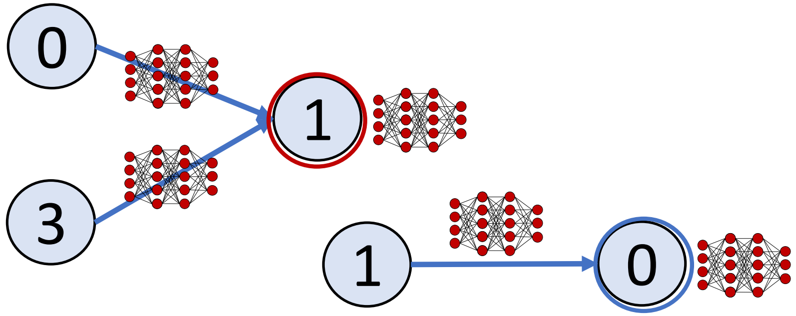
Dependency tree of P1:



We evaluating the cost of **DepCache** and **DepComm** for each dependent neighbor.

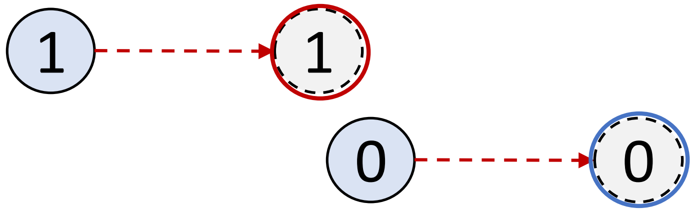
Hybrid Dependency Management

cost of
DepCache:



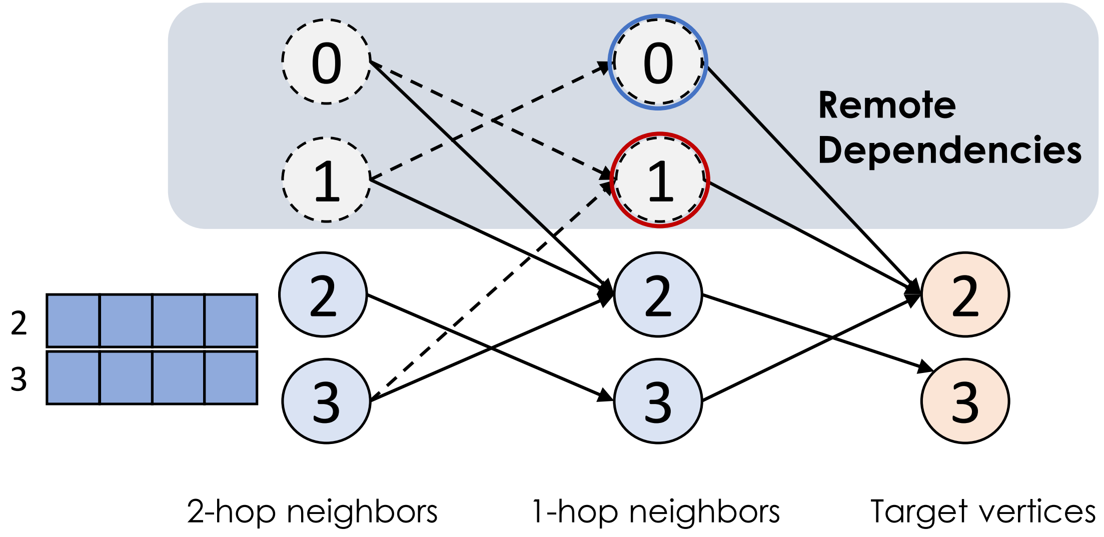
$$t_r(v) = \sum_{l=0}^{L}\left(|E^l(v)| * T_e + Nbr^l(v)| * T_v\right) * dim^l$$

cost of
DepComm:



$$t_c(v) = dim^l * T_c$$

Dependency tree of P1:



We evaluating the cost of **DepCache** and **DepComm** for each dependent neighbor.

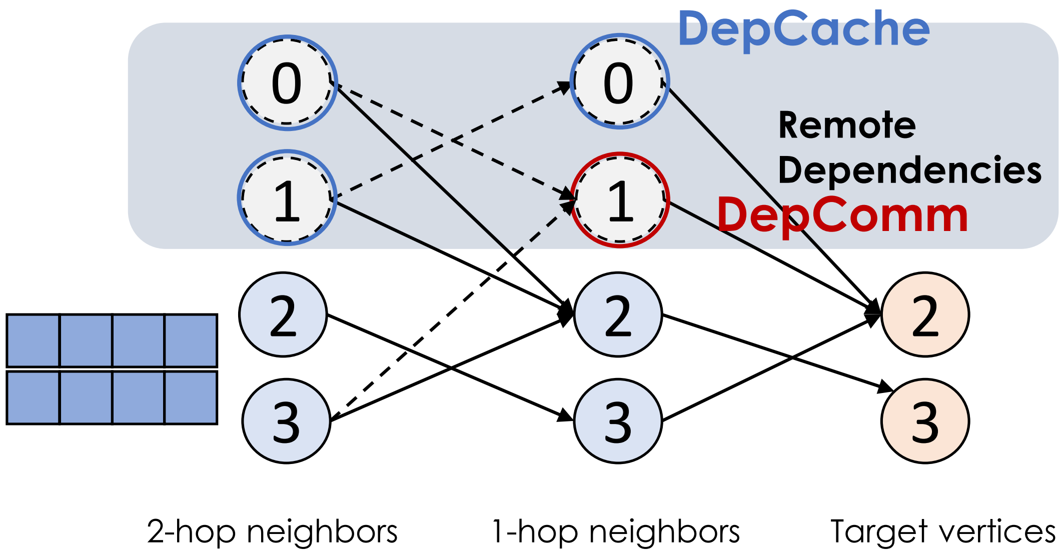
Hybrid Dependency Management

COST MODEL:

$$T(P_i) = \sum_{v \in Dep_{Cache}} t_r(v) + \sum_{v \in Dep_{Comm}} t_c(v)$$

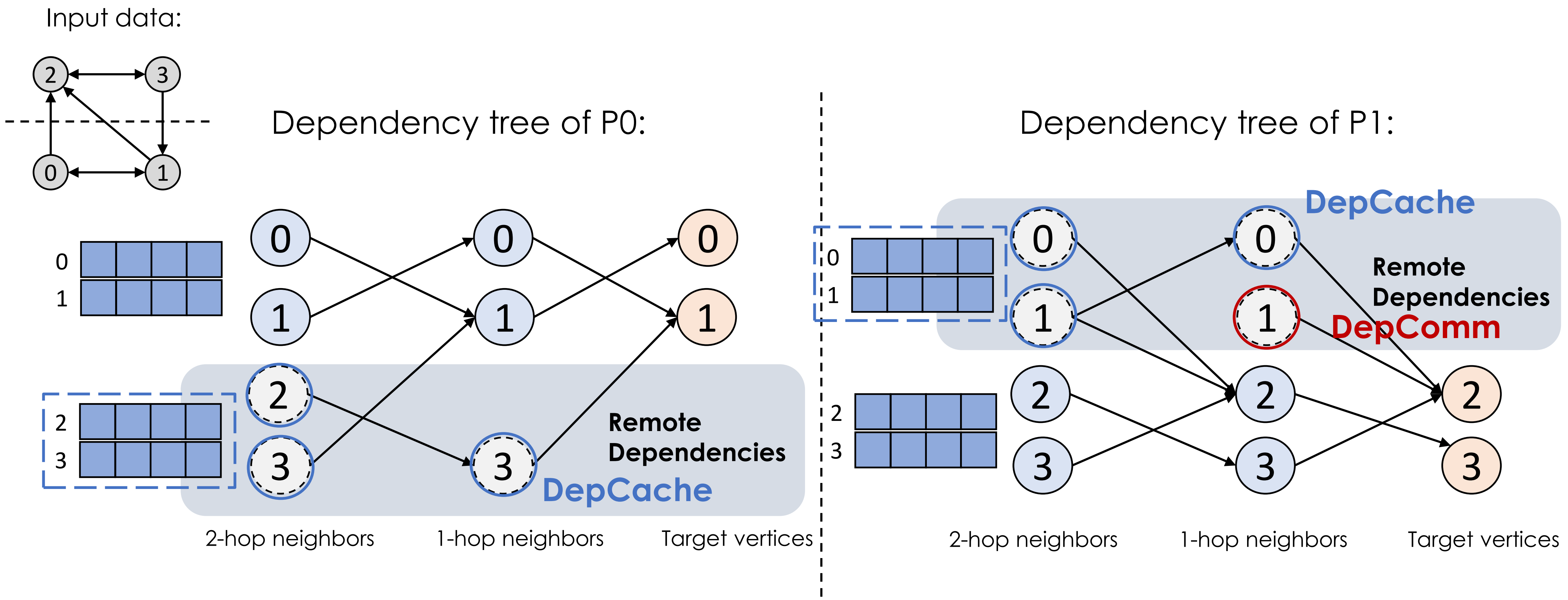
$$\mathbf{s.t.,}\quad Size(Dep_{Cache}) < \boldsymbol{S}$$

Dependency tree of P1:

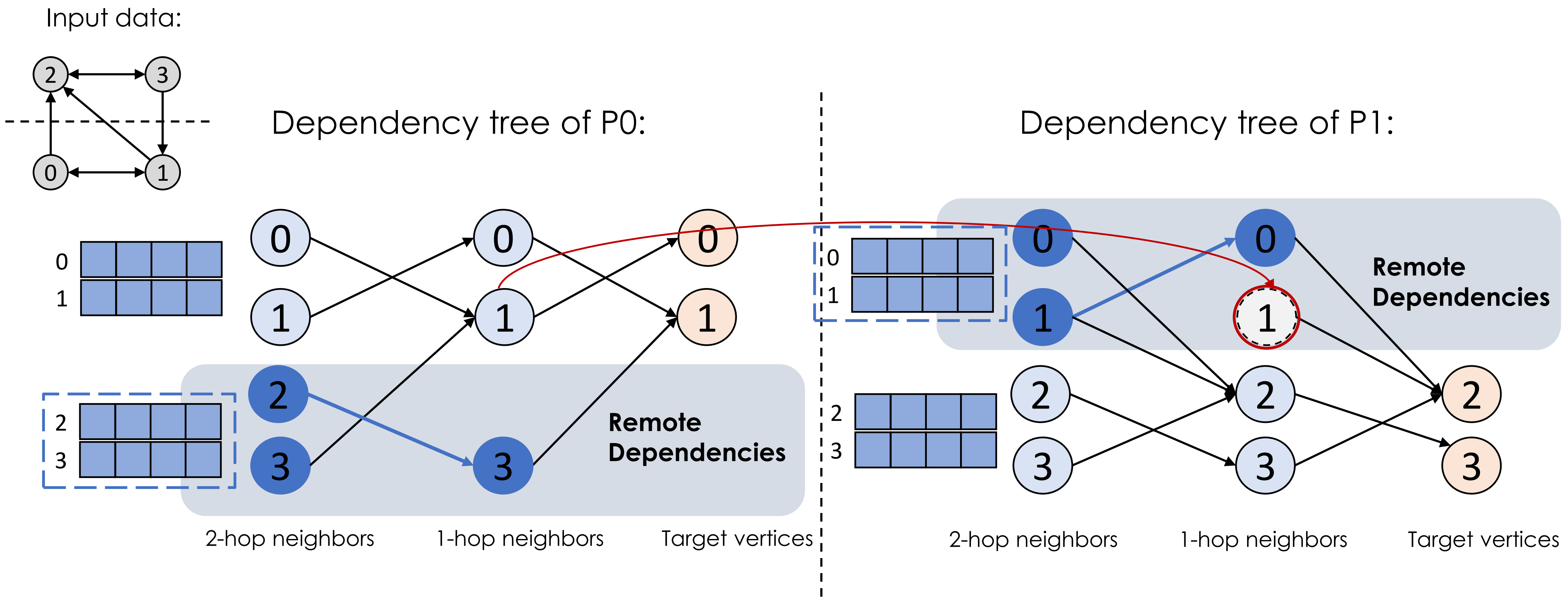


We evaluating the cost of **DepCache** and **DepComm** for each dependent neighbor.

Hybrid Dependency Management



Hybrid Dependency Management

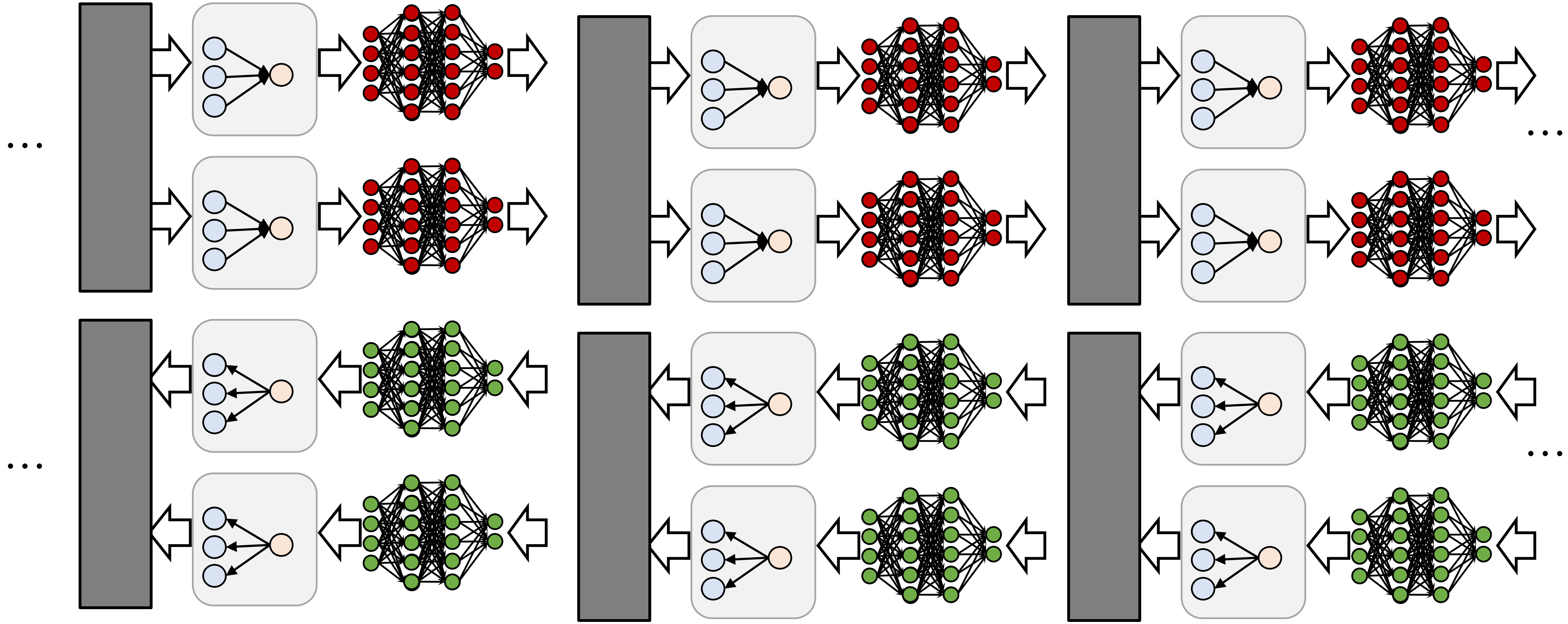


NeutronStar

We propose **NeutronStar**, a GPU-accelerated distributed GNN system with flexible automatic differentiation.

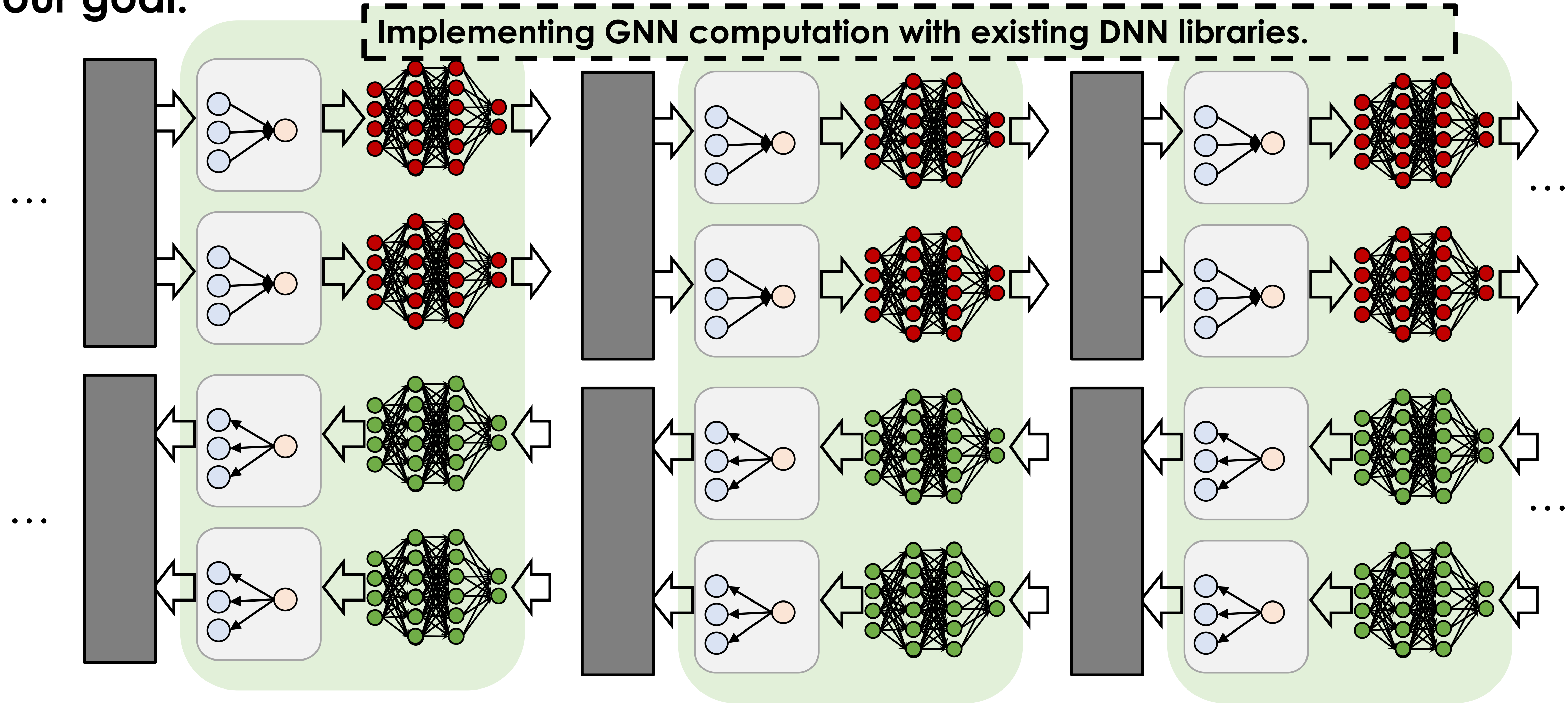
Flexible Auto Differentiation

Manually implementing the **cross-worker** operators is **challenging**



Flexible Auto Differentiation

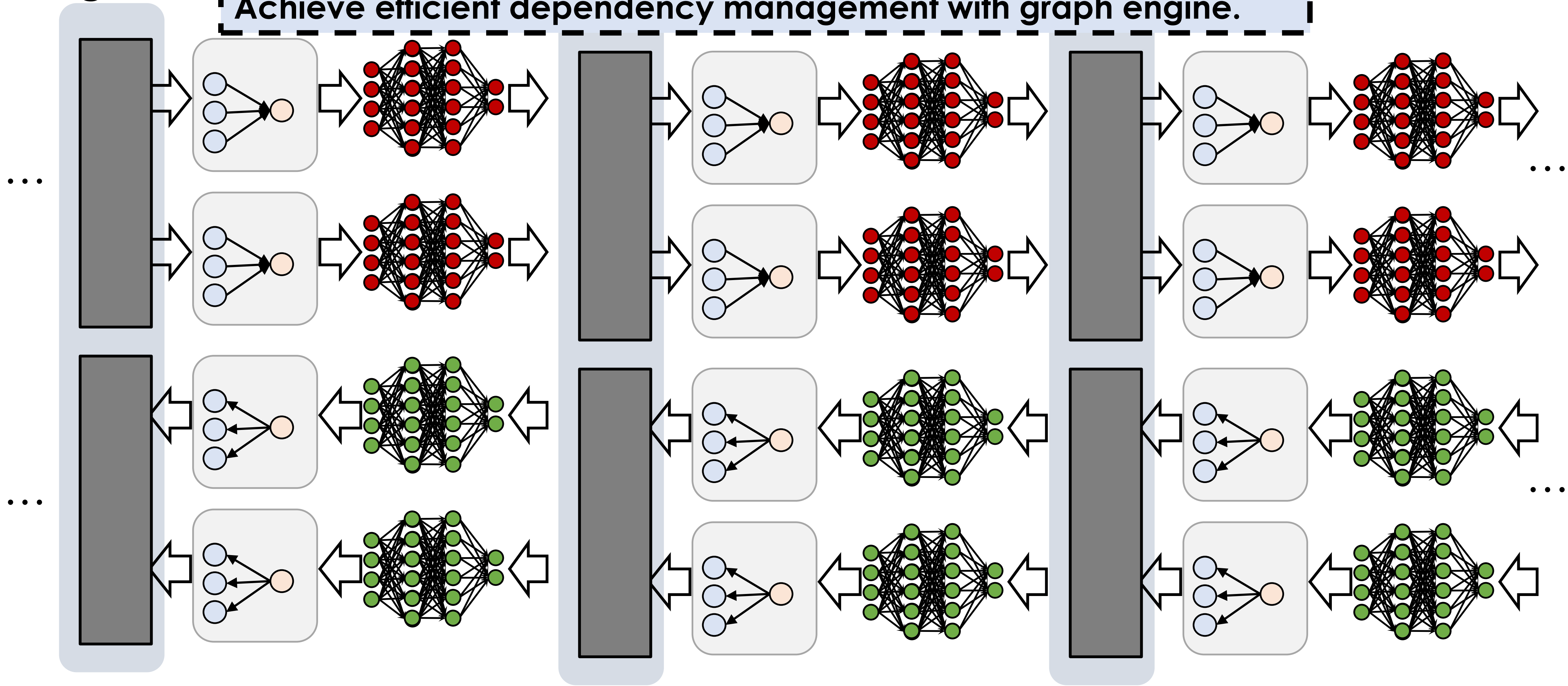
our goal:



Flexible Auto Differentiation

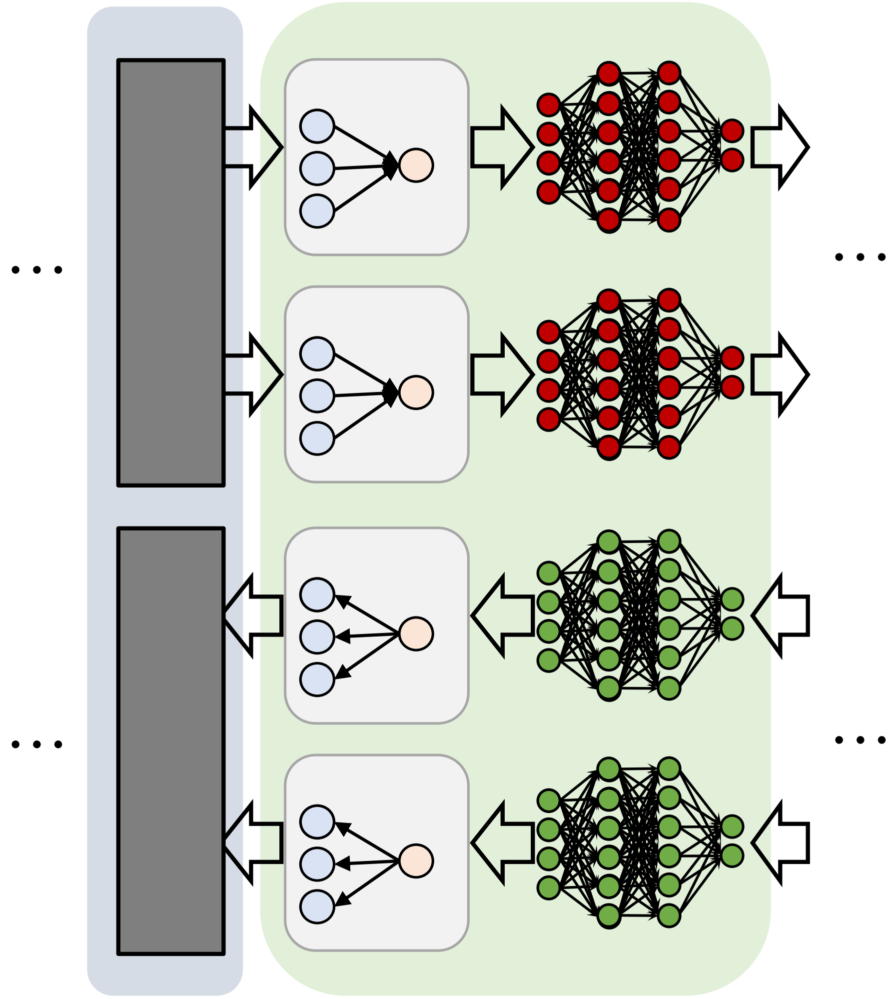
our goal:

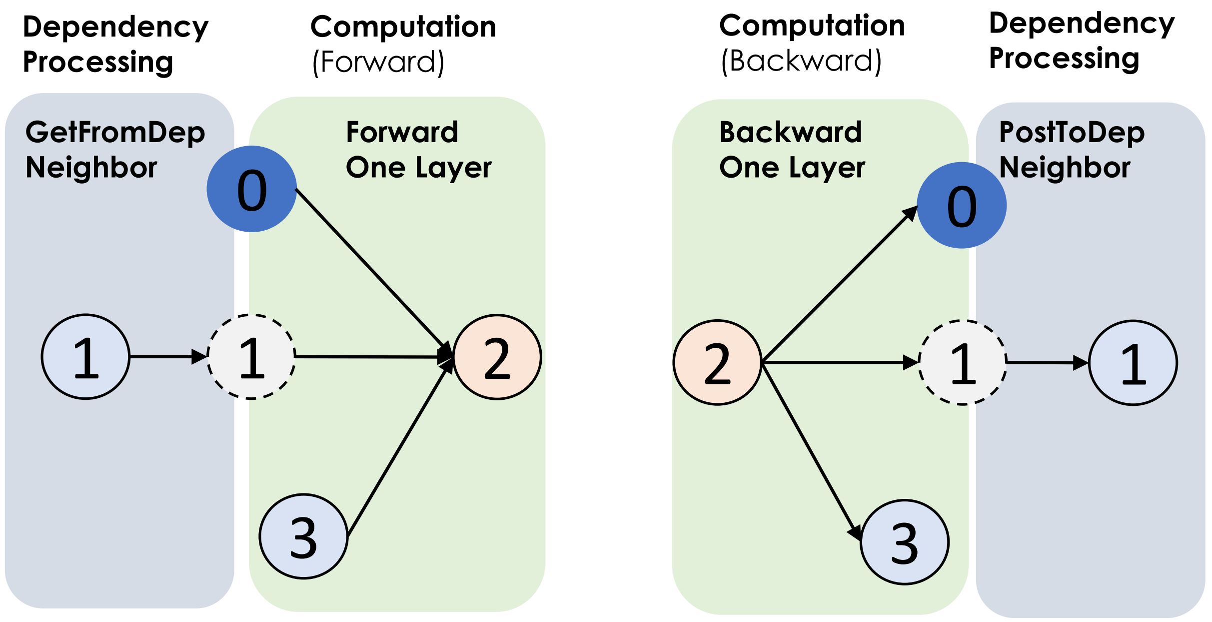
Achieve efficient dependency management with graph engine.



Flexible Auto Differentiation

Decoupling the **dependency management** and **GNN computation**



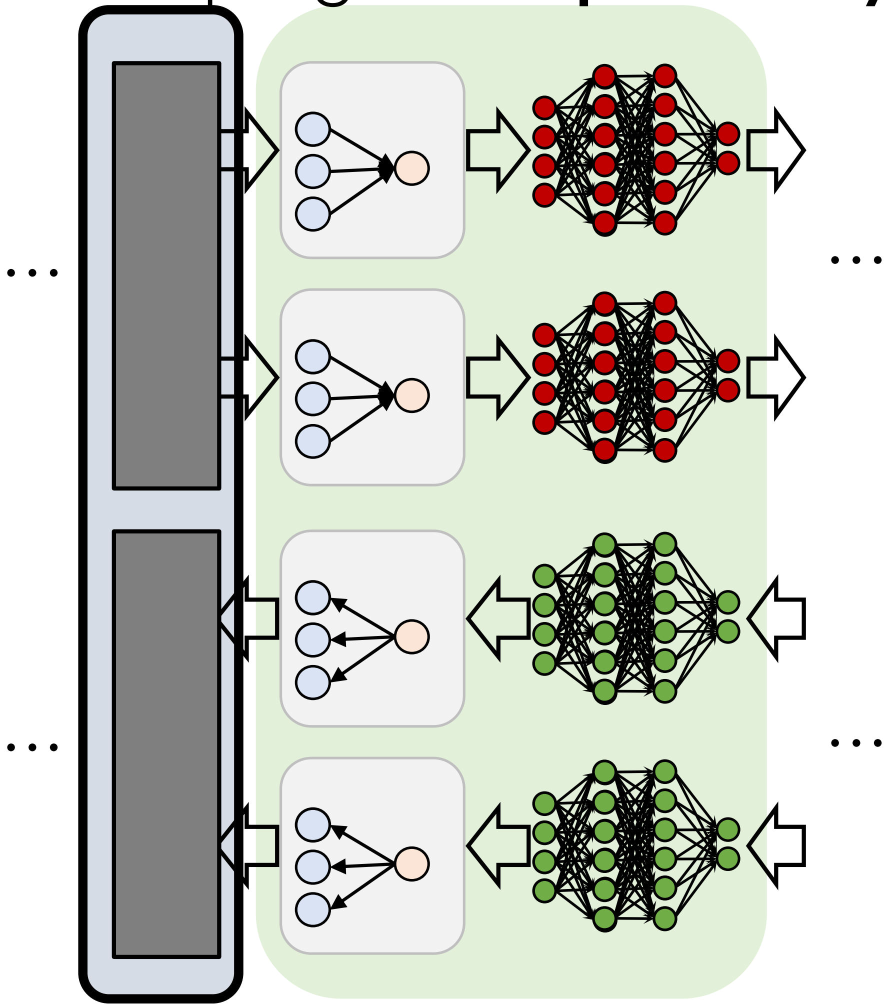


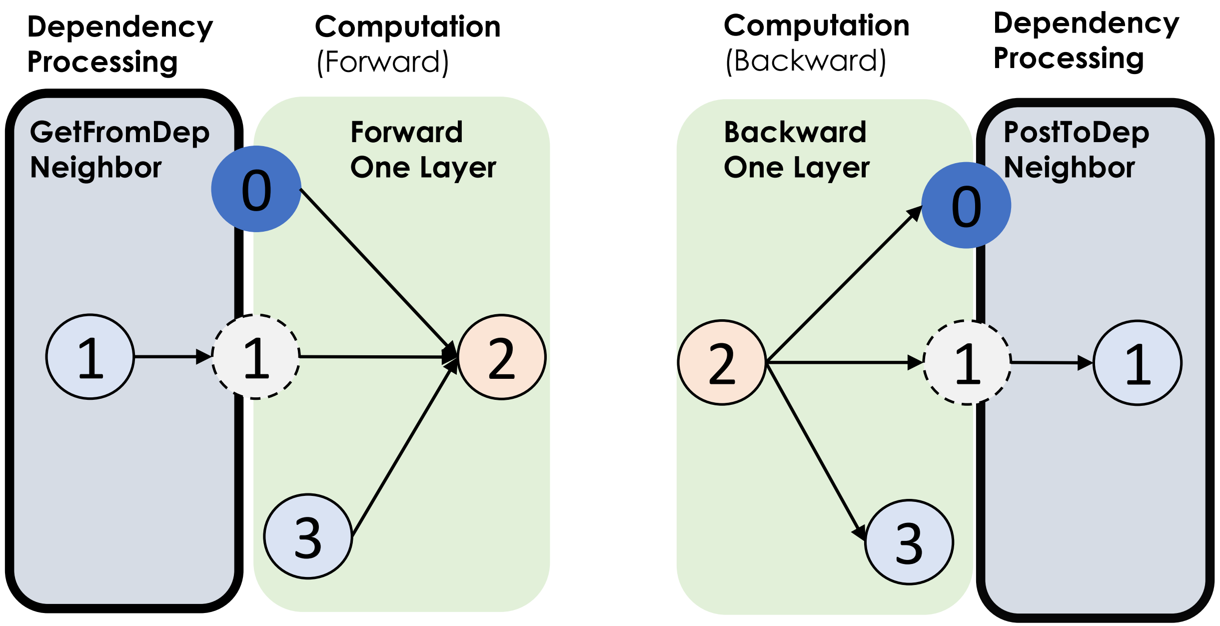
(a) Forward Computation

(b) Backward Computation

Flexible Auto Differentiation

Decoupling the **dependency management** and **GNN computation**

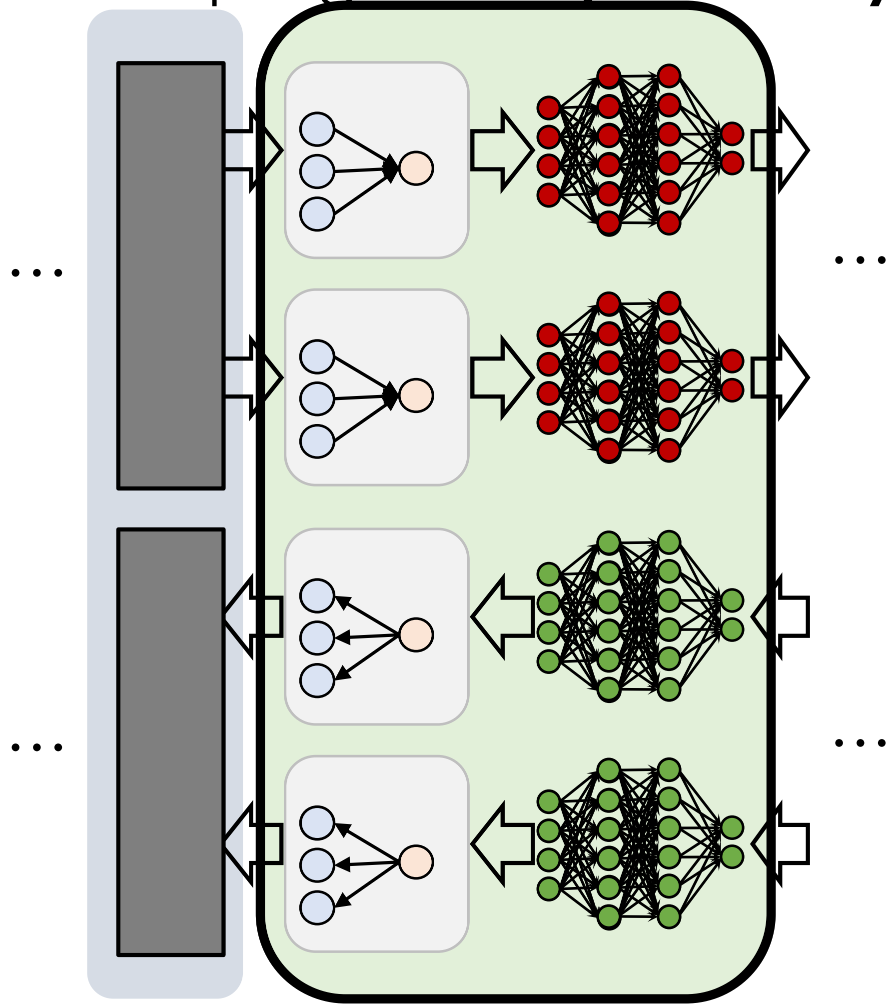


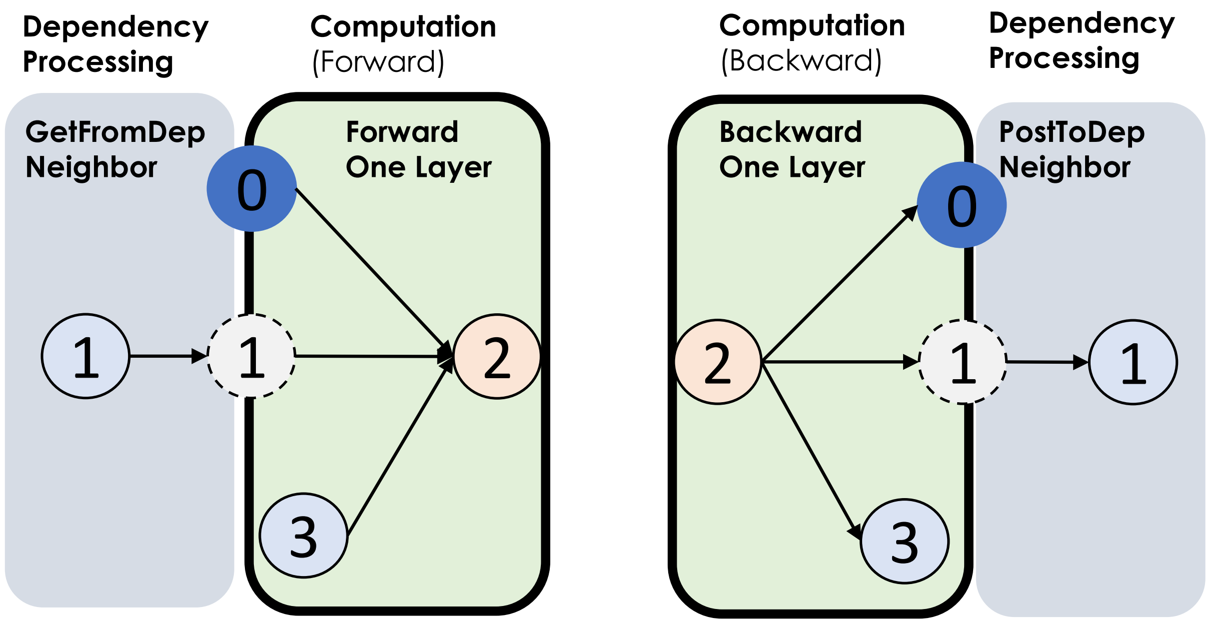


Master-mirror communication

Flexible Auto Differentiation

Decoupling the **dependency management** and **GNN computation**

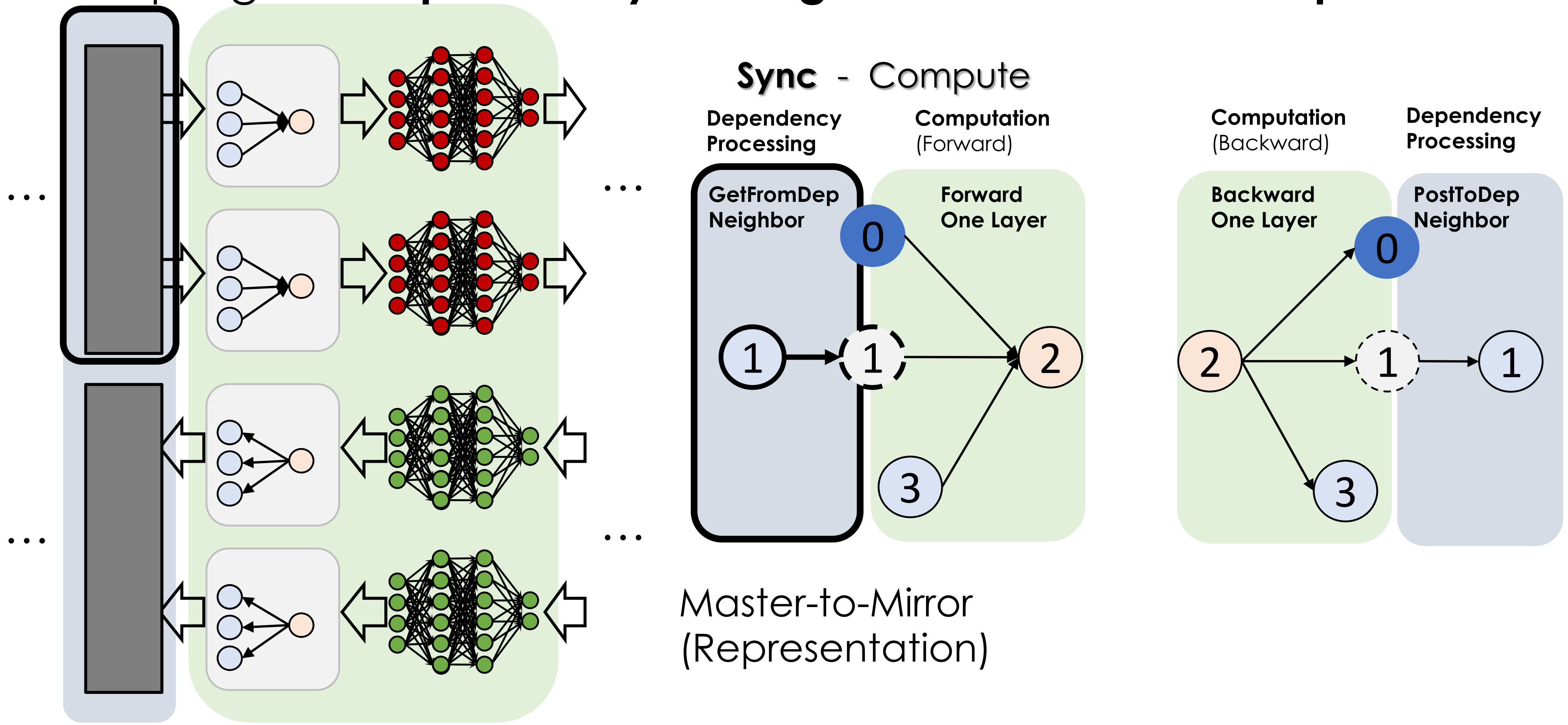




Existing DL library (Pytorch) and NeutronStar's built-in graph operations.

Flexible Auto Differentiation

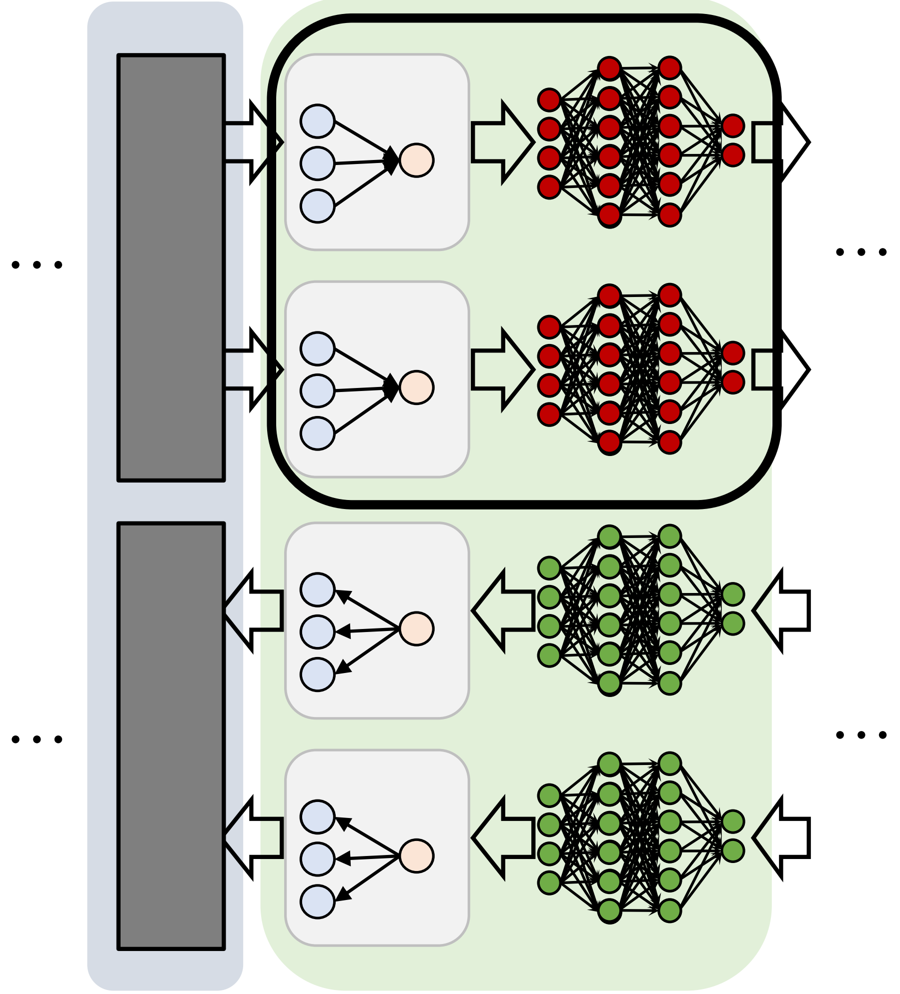
Decoupling the **dependency management** and **GNN computation**

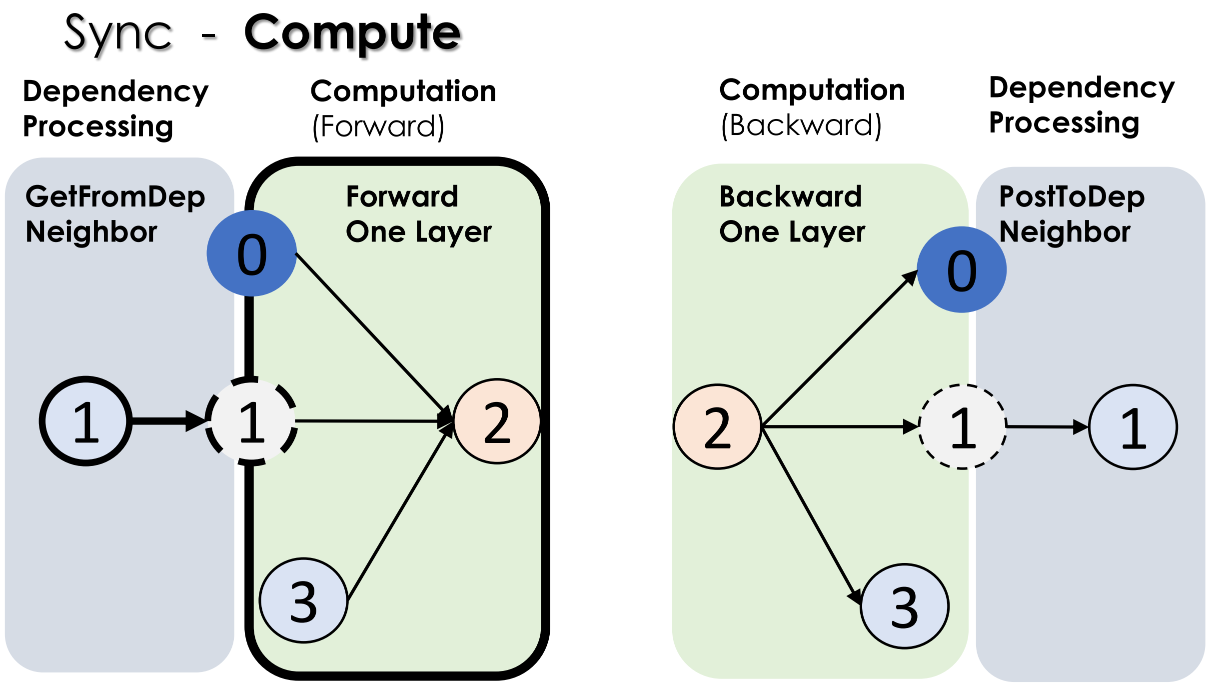


Master-to-Mirror
(Representation)

Flexible Auto Differentiation

Decoupling the **dependency management** and **GNN computation**

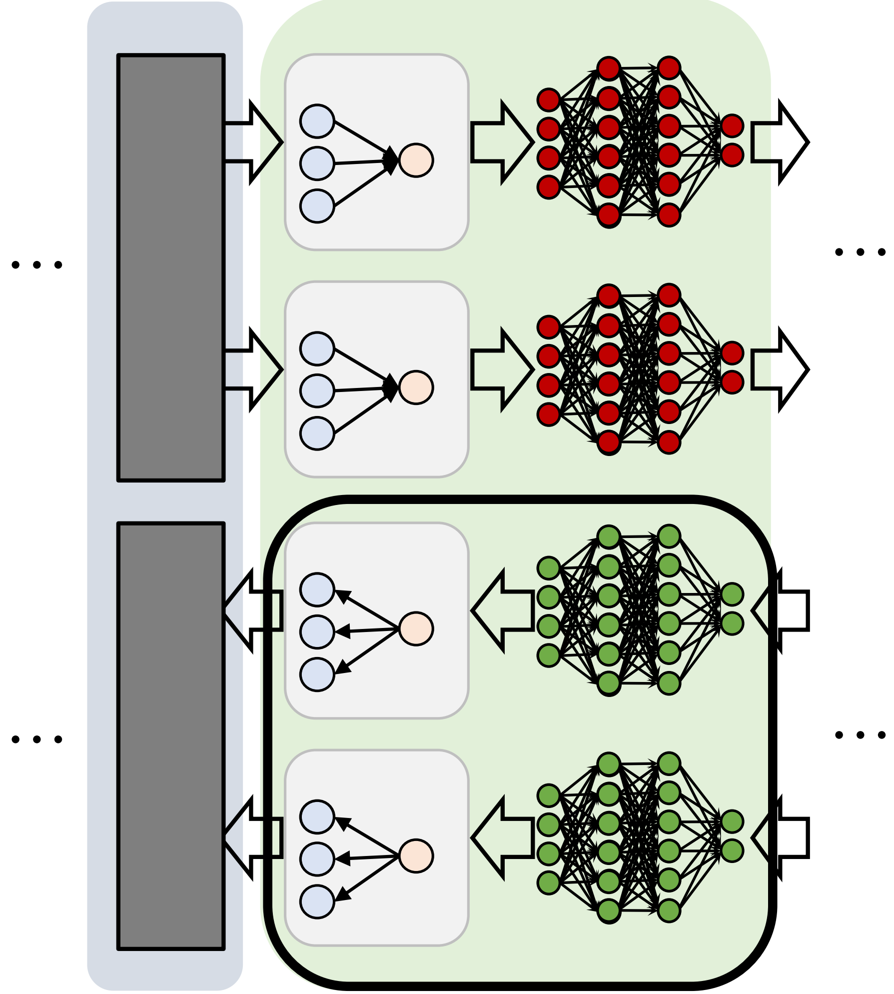


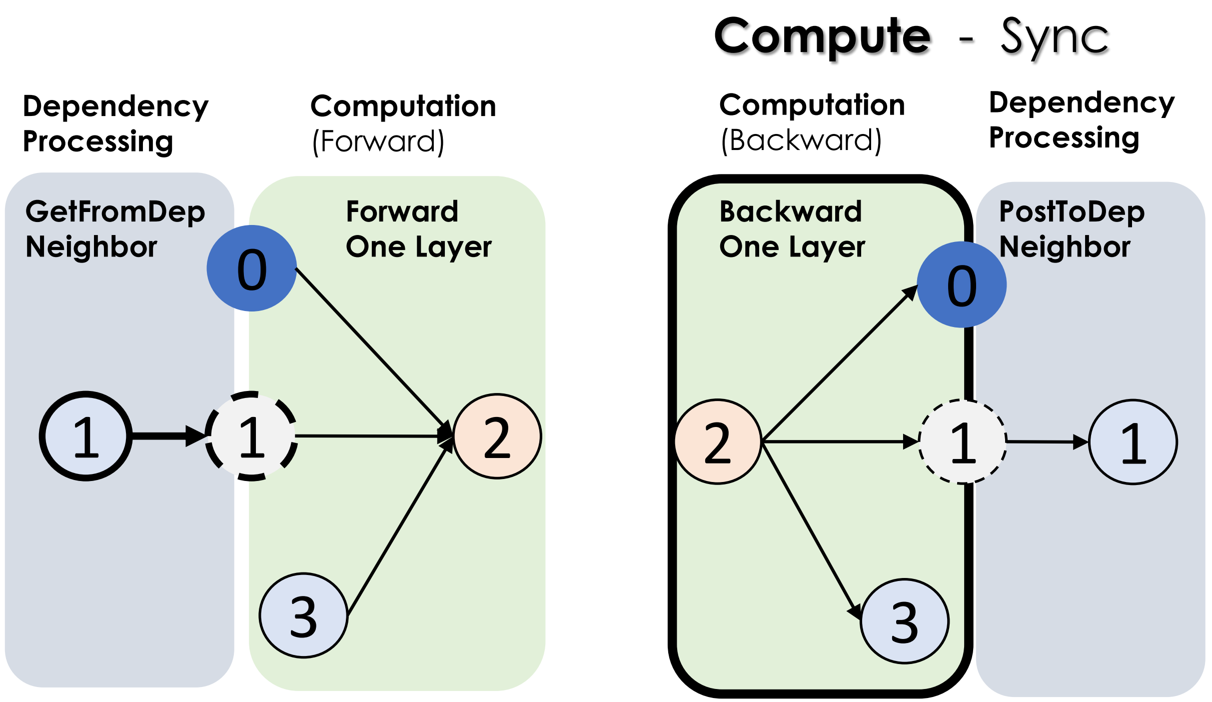


Forward Computation
With Pytorch

Flexible Auto Differentiation

Decoupling the **dependency management** and **GNN computation**

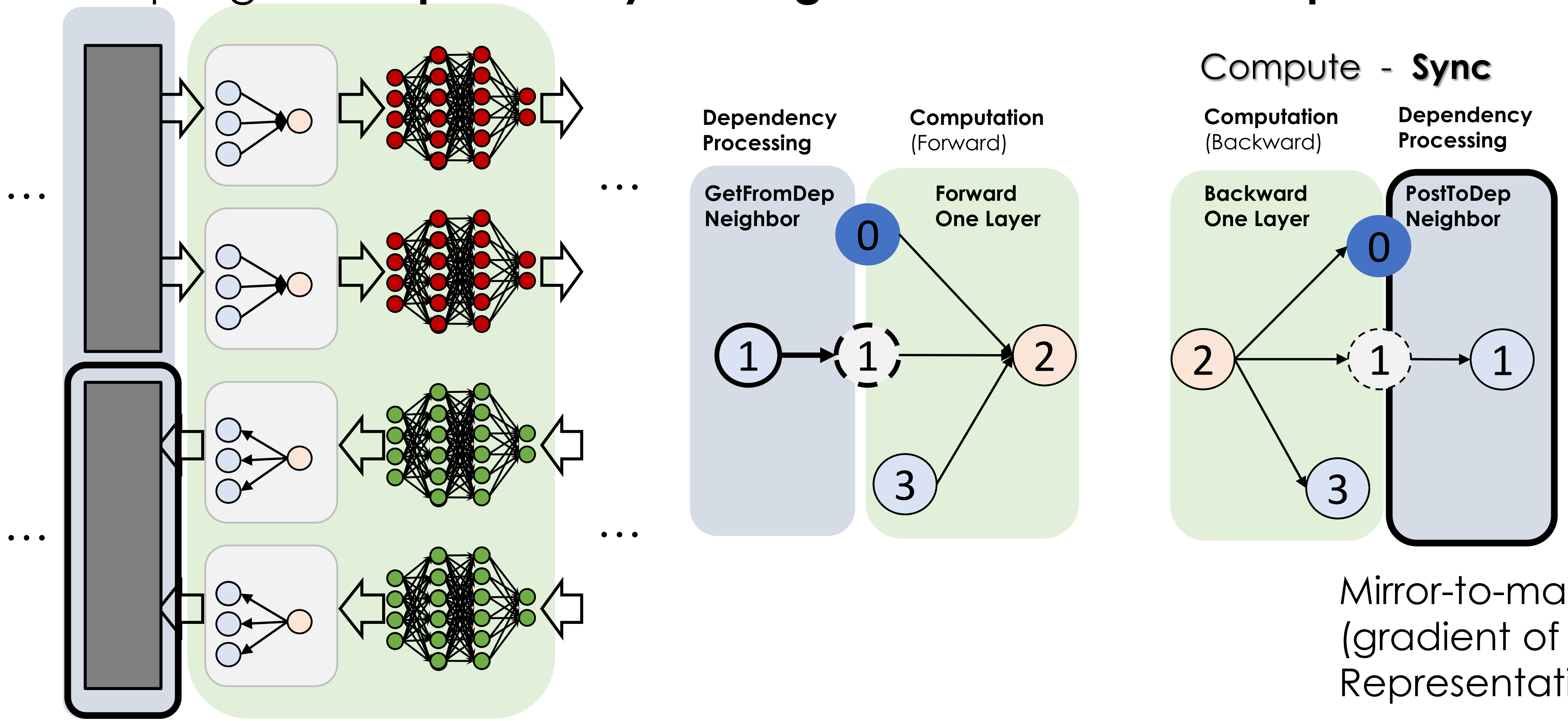




Backward Computation with Pytorch

Flexible Auto Differentiation

Decoupling the **dependency management** and **GNN computation**



Mirror-to-master (gradient of Representation)

Experimental Setups

Baseline: **ROC**, **DistDGL**, **DepCache** (NeutronStar), and **DepComm** (NeutronStar).

Platforms:

A 16-node Aliyun ECS cluster[1] (Each: 16 vCPUs, 62GB RAM, 1 NVIDIA-T4 GPU)

Algorithms and graphs:

- 3 Graph Neural Networks
  GCN, GIN, GAT
- 7 real-world graphs.

Environment

- Ubuntu 18.04 LTS
- CUDA 10.1

Table 2: Dataset description

Dataset	|V|	|E|	ftr. dim	#$\mathbb{L}$	avg. deg	hid. dim
Google	0.87M	5.1M	512	16	5.86	256
Pokec	1.6M	30M	512	16	18.75	256
LiveJournal	4.8M	68M	320	16	14.12	160
Reddit	0.23M	114M	602	41	487	256
Orkut	3.1M	117M	320	20	38.1	160
Wiki-link	12M	378M	256	16	31.12	128
Twitter	42M	1.5B	52	16	70.5	32

[1] Clusters are connected via 6GigE

Effectiveness of Hybrid Processing

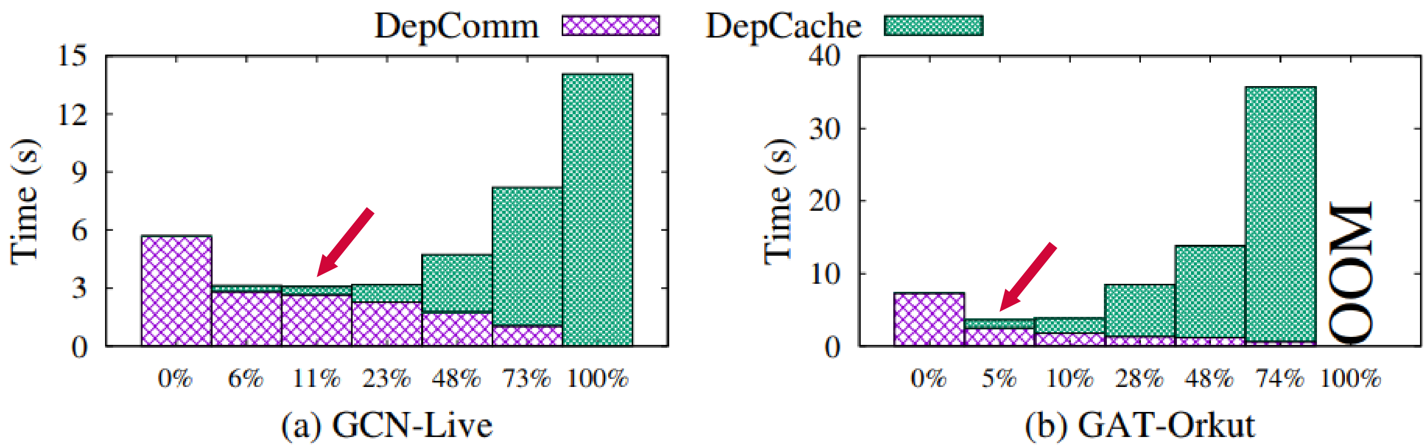


Figure 11: Runtime results when varying the ratios between cached dependencies and communicated dependencies.

(1) Neither **communicating** nor **caching** all dependencies will reach the optimal performance.

(2) The optimal performance is reached when **mixing** **DepCache** and **DepComm**.

Performance Comparison

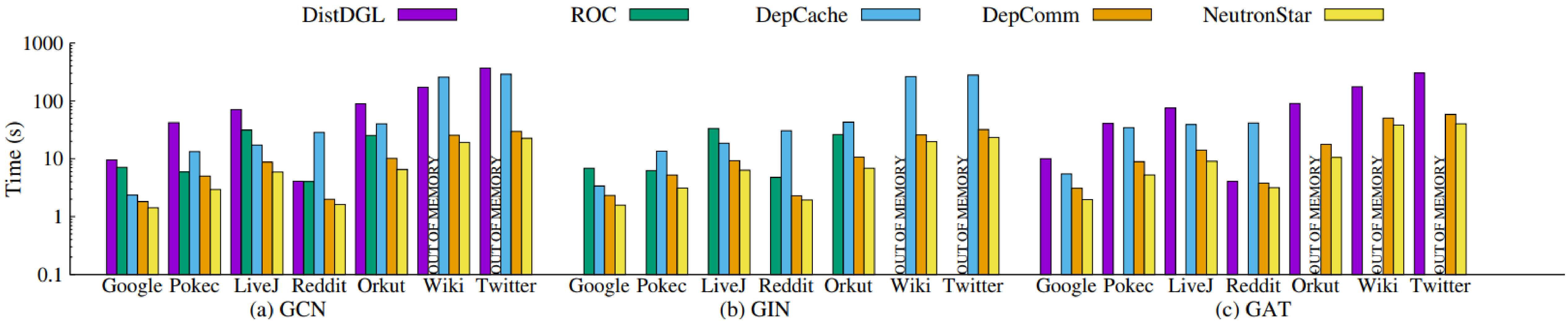


Figure 10: Overall performance comparison.

Compared with the two representative distributed GNN systems **(DistDGL, ROC)**, **NeutronStar** achieves **1.8x – 14.3x** and **1.8X-5.3X** speedups on 3 GNNs and several real datasets, respectively.

Performance Comparison

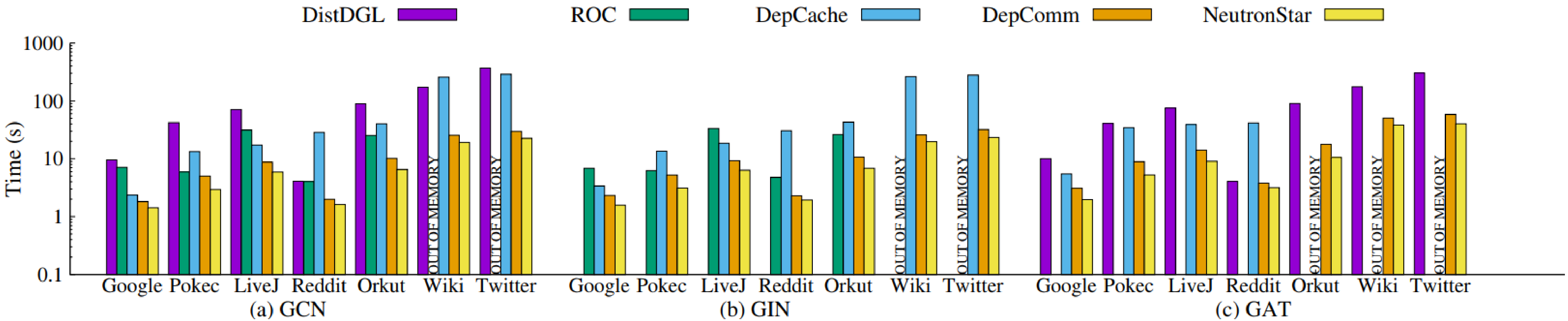


Figure 10: Overall performance comparison.

Compared with the **DepCache** and **DepComm**, **NeutronStar** achieves **2.0x – 15.0x** and **1.2X-1.7X** speedups on 3 GNNs and several real datasets, respectively.

Accuracy Comparison

Time-to-accuracy comparison

NeutronStar outperforms other approaches

1.20X faster than DepComm
1.96X faster than DepCache-Sampling
24.62X faster than DepCache

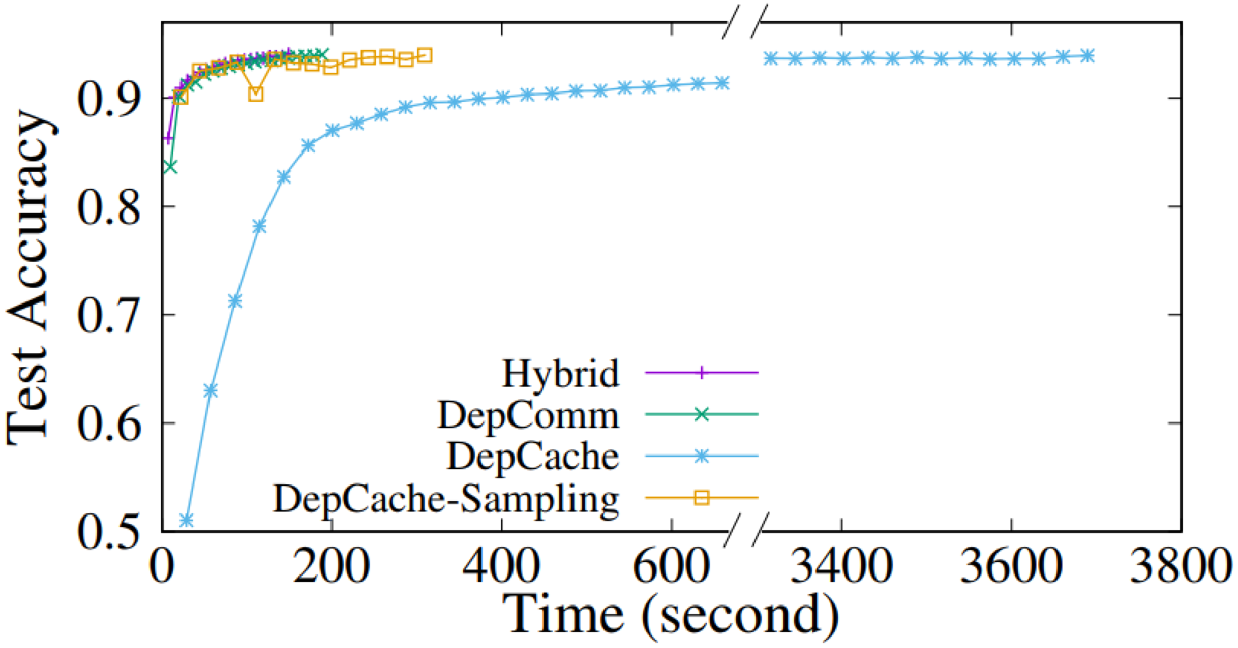


Figure 14: Accuracy comparisons between Hybrid, DepComm, DepCache, **and** DepCache-**sampling with GCN on the Reddit dataset. Each dot indicates five training epochs for** Hybrid **and** DepComm, **and one training epoch for** DepCache **and** DepCache-**sampling.**

Summary

NEUTRON STAR

NeutronStar: Distributed GNN training with hybrid dependency management.

- **Providing insight into the two existing approaches**

  We conduct a comprehensive study on the performance merits and limits of the two distributed GNN training approaches (**DepCache** and **DepComm**).

Summary

NEUTRON STAR

NeutronStar: Distributed GNN training with hybrid dependency management.

- ☐ **Providing insight into the two existing approaches**

  We conduct a comprehensive study on the performance merits and limits of the two distributed GNN training approaches (**DepCache** and **DepComm**).

- ☐ **Proposing a hybrid dependency management framework**

  We identify the key tradeoff between the two approaches and propose a hybrid dependency management approach.

Summary

NEUTRON STAR

NeutronStar: Distributed GNN training with hybrid dependency management.

- **Providing insight into the two existing approaches**

  We conduct a comprehensive study on the performance merits and limits of the two distributed GNN training approaches (**DepCache** and **DepComm**).

- **Proposing a hybrid dependency management framework**

  We identify the key tradeoff between the two approaches and propose a hybrid dependency management approach.

- **Delivering a fast distributed GNN system**

  We design and implement NeutronStar, a distributed GNN system with auto differentiation that achieves 1.8X-14.3X speedups over the existing GNN systems.

Summary

NEUTRON STAR

NeutronStar: Distributed GNN training with hybrid dependency management.

- **Providing insight into the two existing approaches**

  We conduct a comprehensive study on the performance merits and limits of the two distributed GNN training approaches (**DepCache** and **DepComm**).

- **Proposing a hybrid dependency management framework**

  We identify the key tradeoff between the two approaches and propose a hybrid dependency management approach.

- **Delivering a fast distributed GNN system**

  We design and implement NeutronStar, a distributed GNN system with auto differentiation that achieves 1.8X-14.3X speedups over the existing GNN systems

- **The codes are publicly available on github**

  https://github.com/Wangqge/NeutronStarLite

Questions