UGACHE: A Unified GPU Cache for Embedding-based Deep Learning

Xiaoniu Song[1,2], Yiwen Zhang[1], Rong Chen[1,2] and Haibo Chen[1]

[1]Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University,

[2]Shanghai Artificial Intelligence Laboratory

SOSP 23

Presented by **Zheng Yang** and Yicheng Zhang

2024-11-19

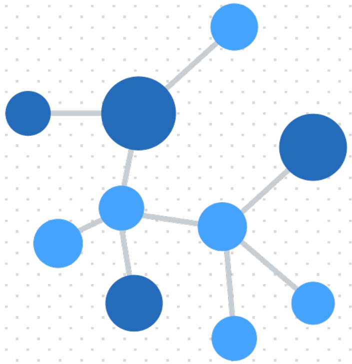
Outline

- Introduction
- Background and Motivation
- UGache
  - Extractor
  - Solver
- Evaluation

Embedding in Deep Learning

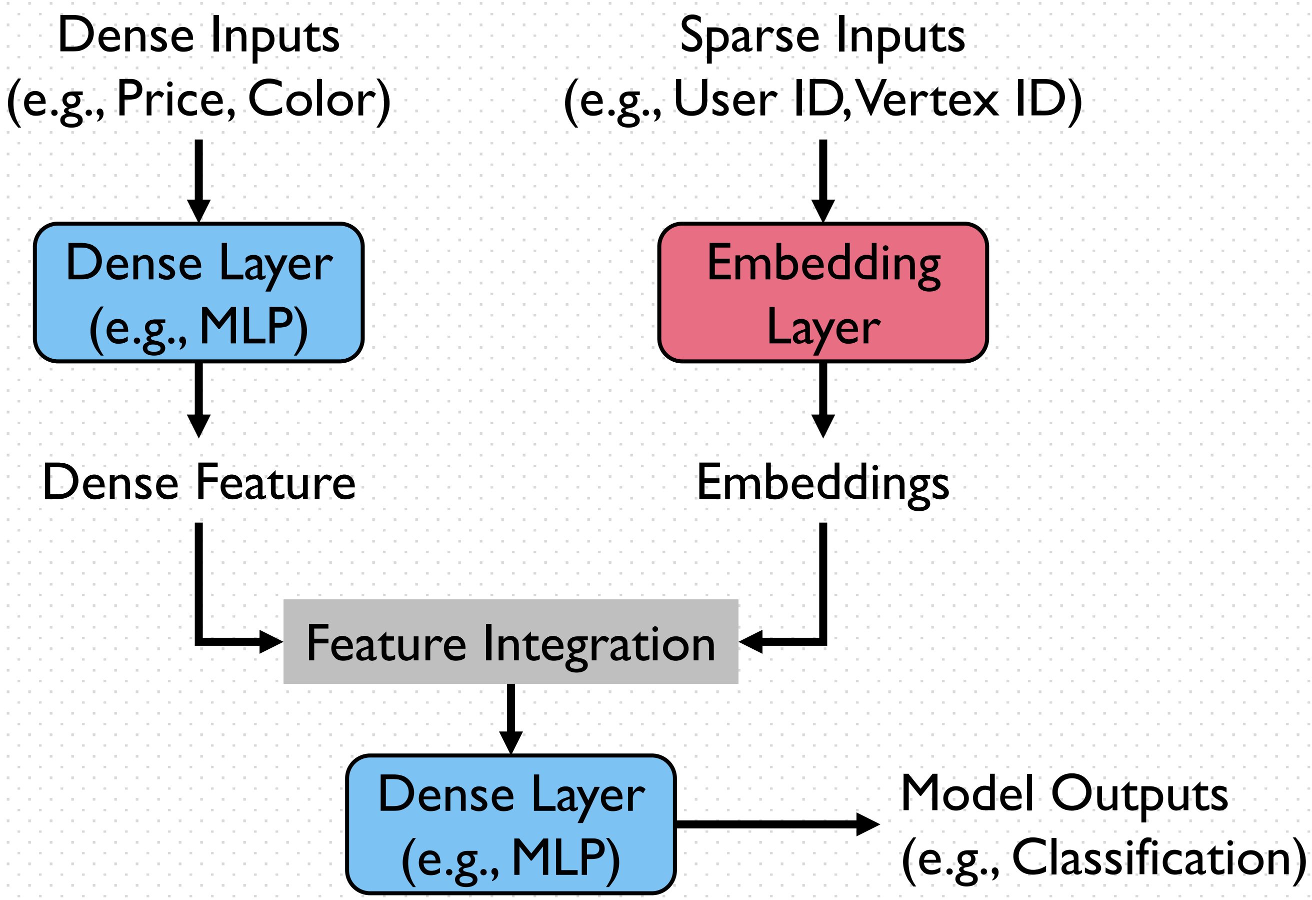
- Dense Inputs: continuous value

- Sparse Inputs: list of IDs (e.g., User ID, Vertex ID)

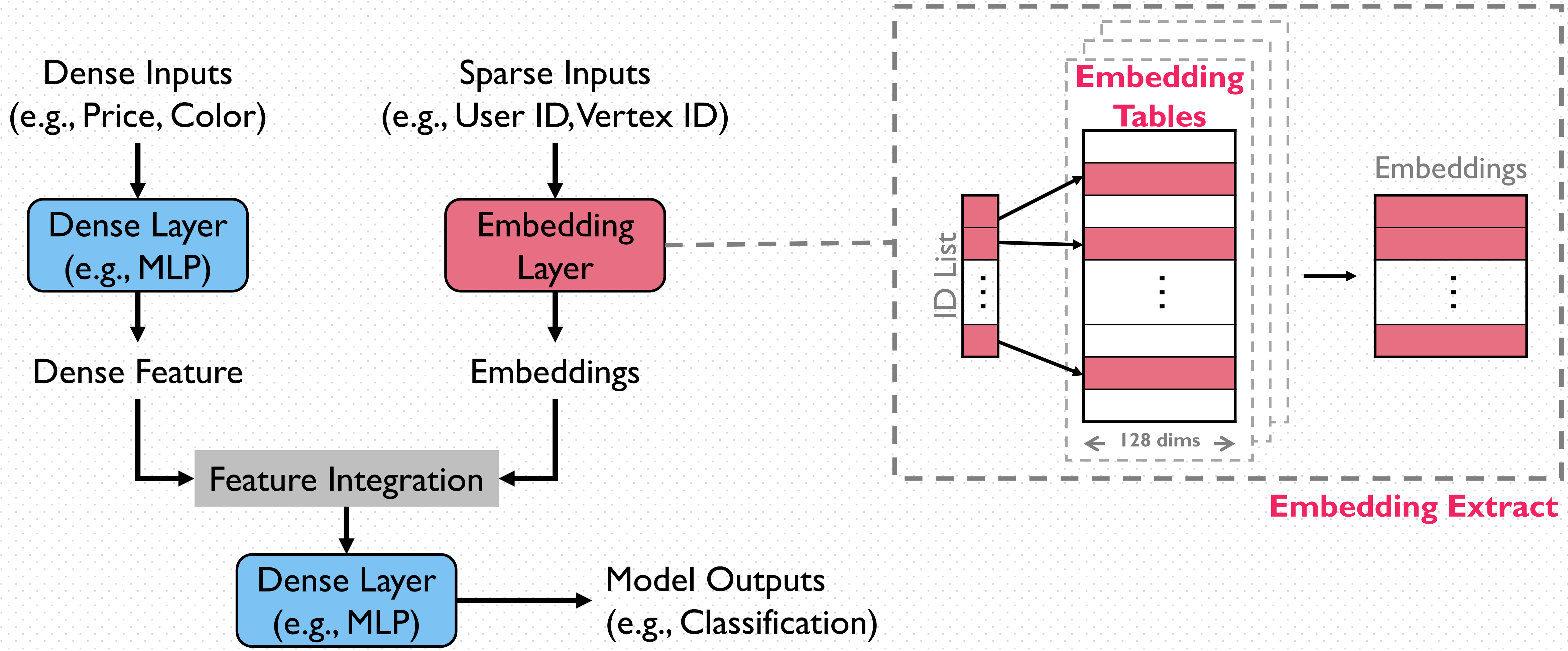


Poor support in traditional DL

Embedding in Deep Learning



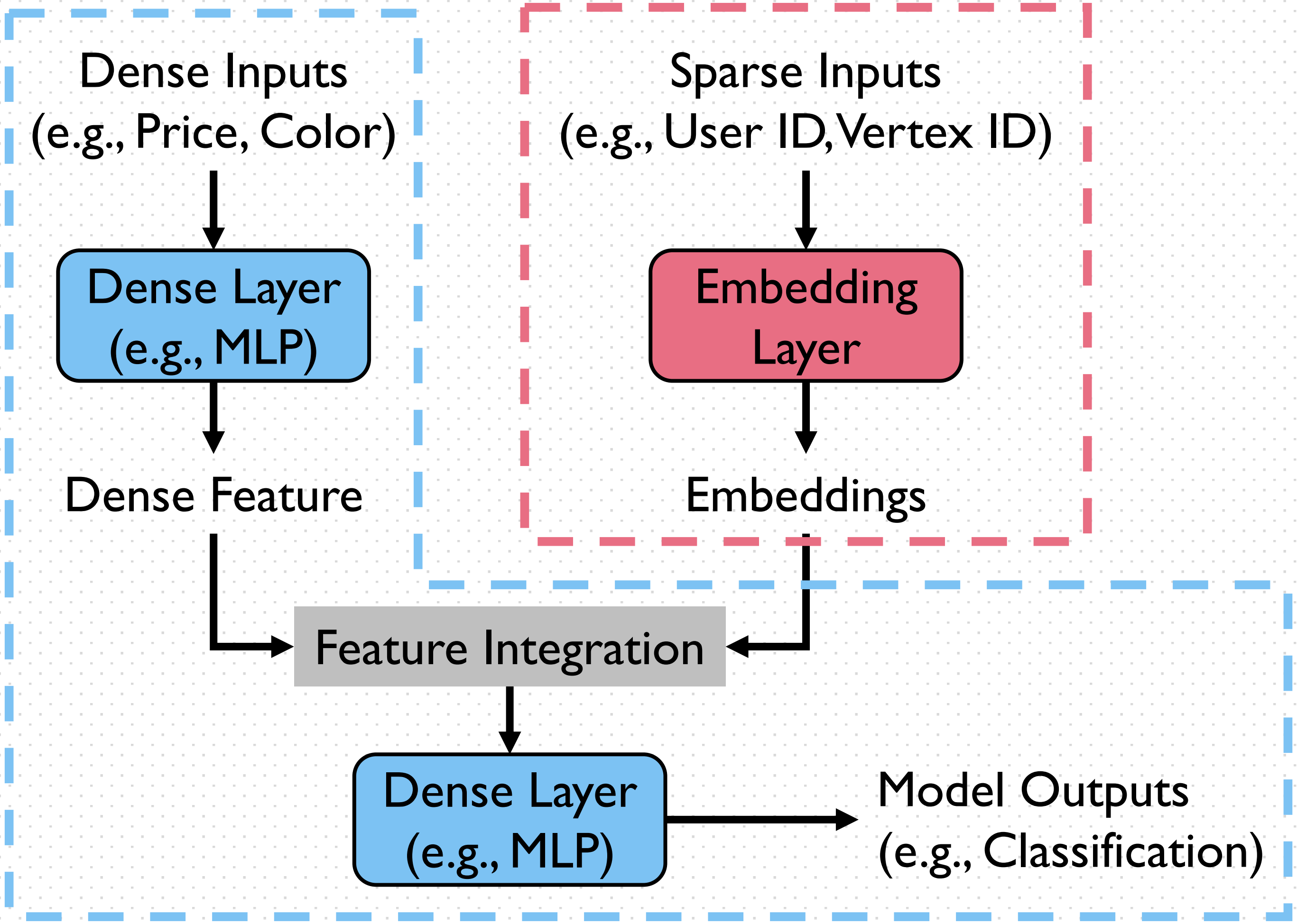
Embedding in Deep Learning



Outline

- Introduction
- Background and Motivation
- UGache
    - Extractor
    - Solver
- Evaluation

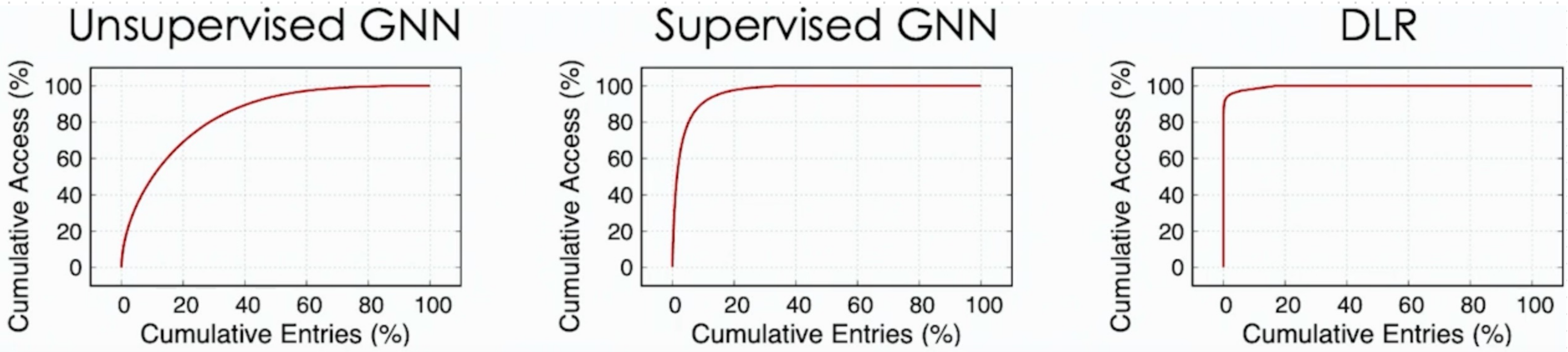
Embedding Bottleneck



Dense	Embedding
~100MB	~100GB
GPU	Host
~10ms	~100ms

Skewed Embedding Access

(Deep Learning Recommendation)

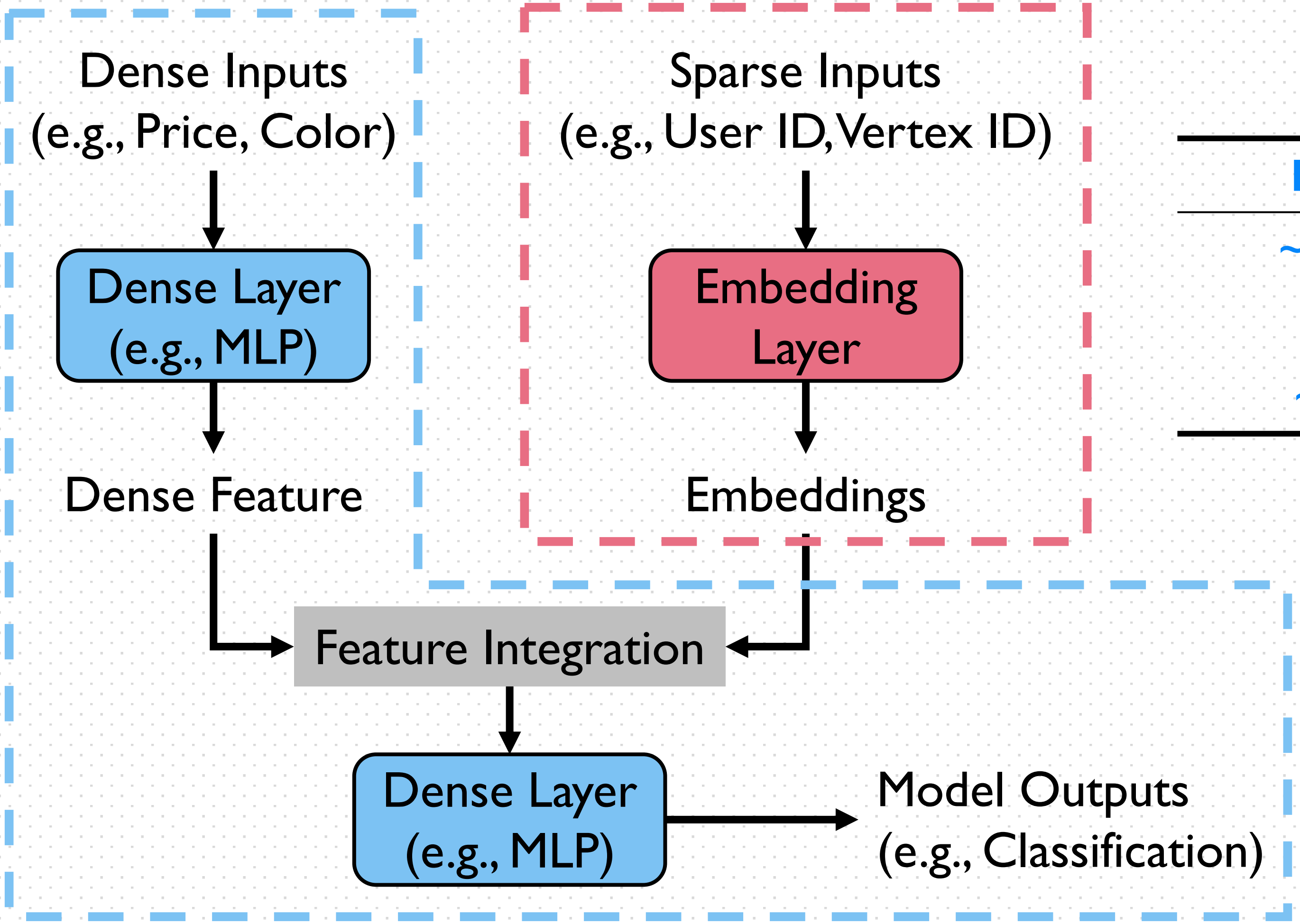


Source of skewness:

- Preferences in user choice
- Power-law in graph

Skewness remains ***relatively constant*** over an extended period

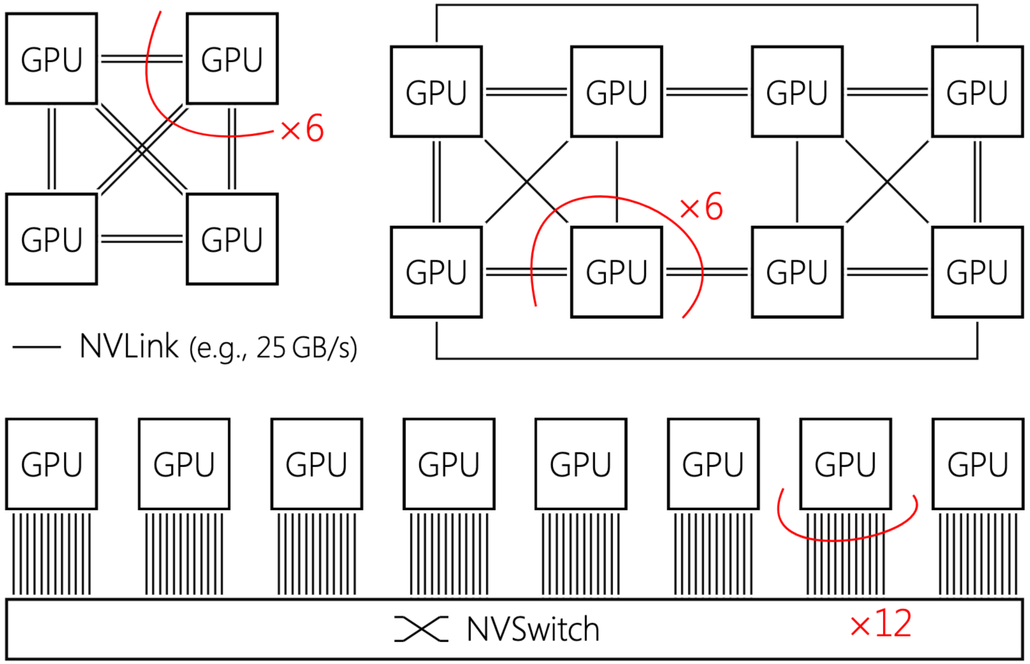
Enable a single-GPU Cache?



Dense	Embedding	+Cache
~100MB	~100GB	
GPU	Host	20% in GPU
~10ms	~100ms →	~20ms

Still a bottleneck!!!

Opportunity: GPU *Fast Interconnect*



Bandwidth (GB/s)		
	V100	A100
Local	900	1900
Remote	**300**	**600**
Host	**32**	**64**

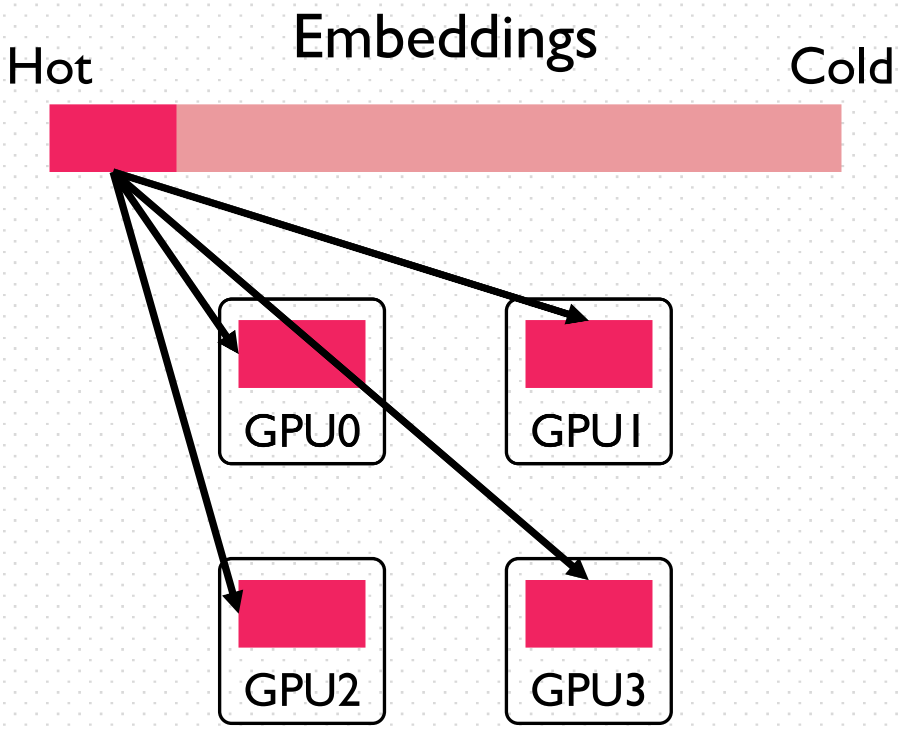
Enabling a faster and larger multi-GPU Cache?

Towards Fast and Large Multi-GPU Cache

- Cache Policy
  - How to ***place*** embeddings
- Extraction Mechanism
  - How to ***fetch*** embeddings

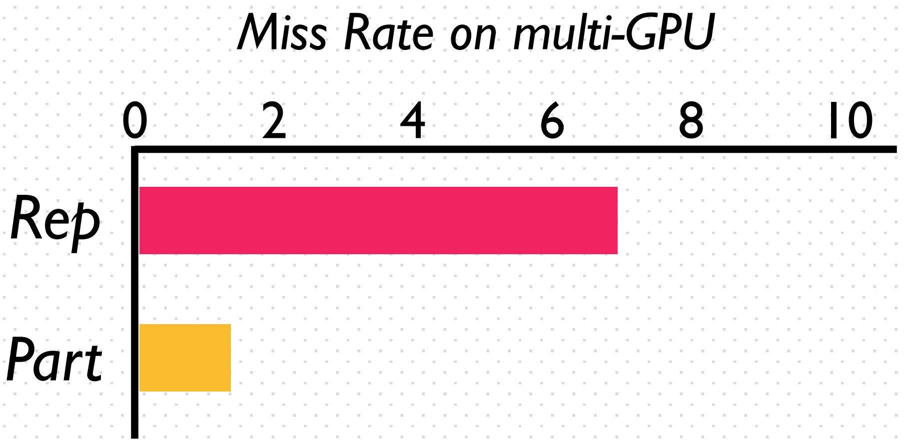
Multi-GPU Cache Policy

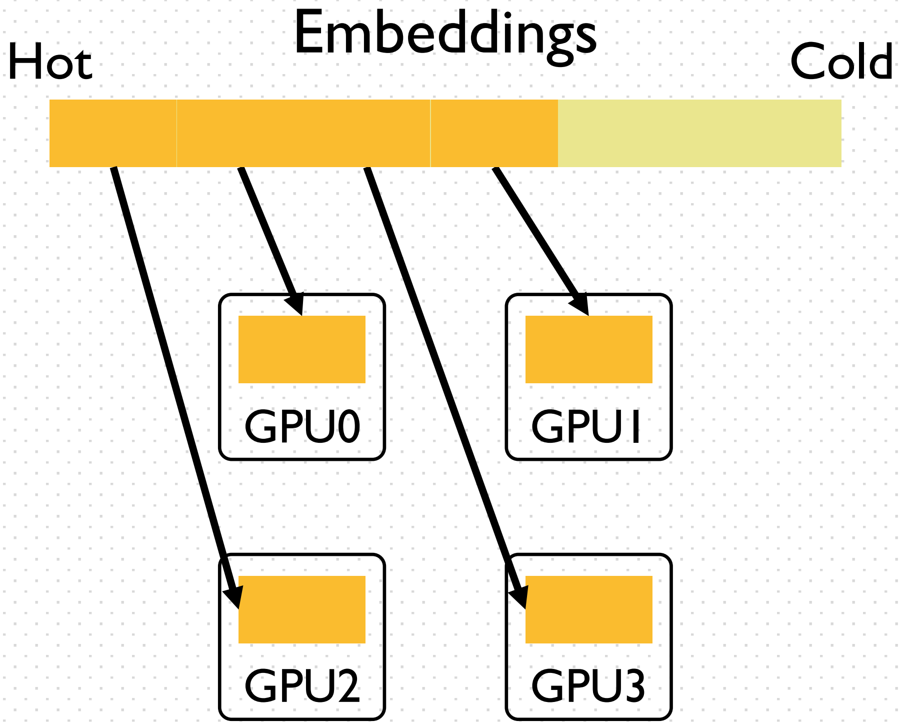
- ***Replication*** cache
  - Port single GPU solution
  - Independently cache hot entry
  - 🙁Ignore fast interconnect
  - 🙁>99% overlap in cache hit requests



Multi-GPU Cache Policy

- *Partition* cache
  - Cache more distinct entry
  - Reduced miss rate on multi-GPU



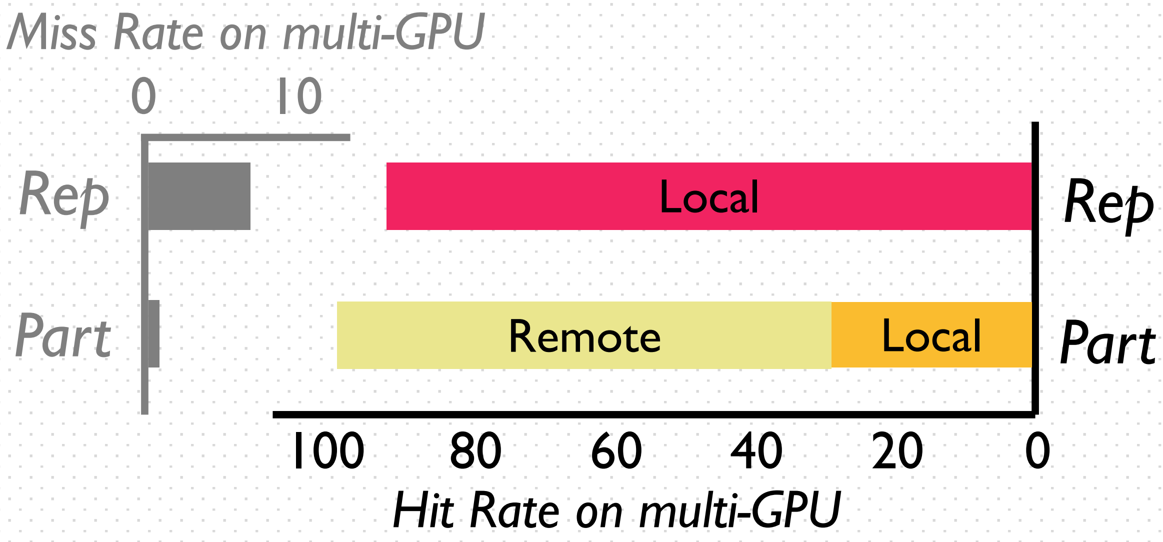


Multi-GPU Cache Policy

- ***Partition*** cache
  - 🙁 Poor local hit rate
    - Remote is 3x slower than local

Bandwidth (GB/s)

	V100	A100
Local	**900**	**1900**
Remote	**300**	**600**
Host	32	64

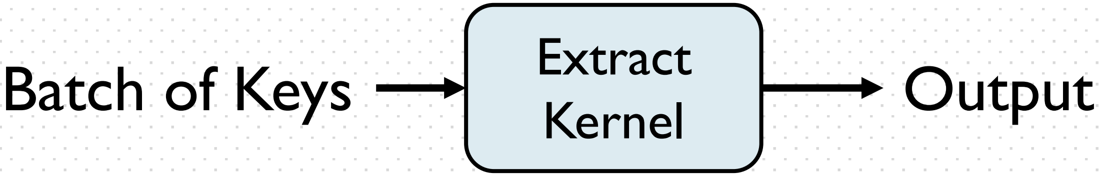


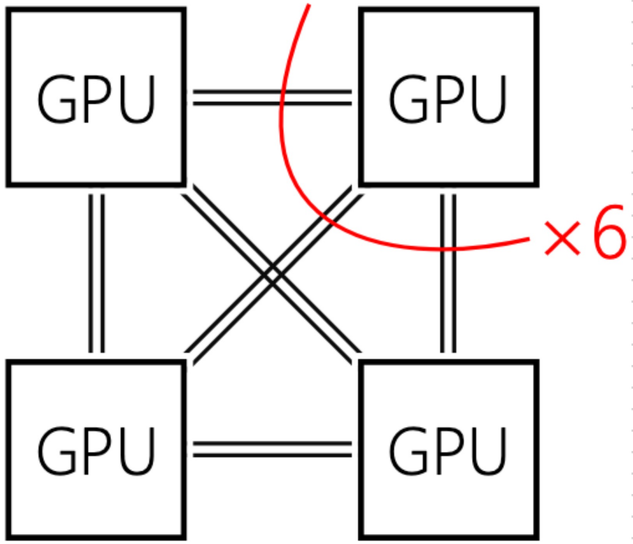
Multi-GPU Cache Challenges

- #1: Cache Policy
  - Reduce miss rate while preserve local hit rate

Multi-GPU Extraction Mechanism

- Peer-based
  - Unified address space for multi-GPU

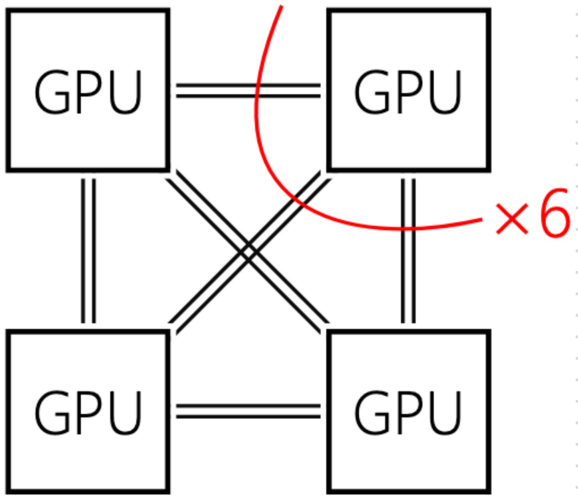


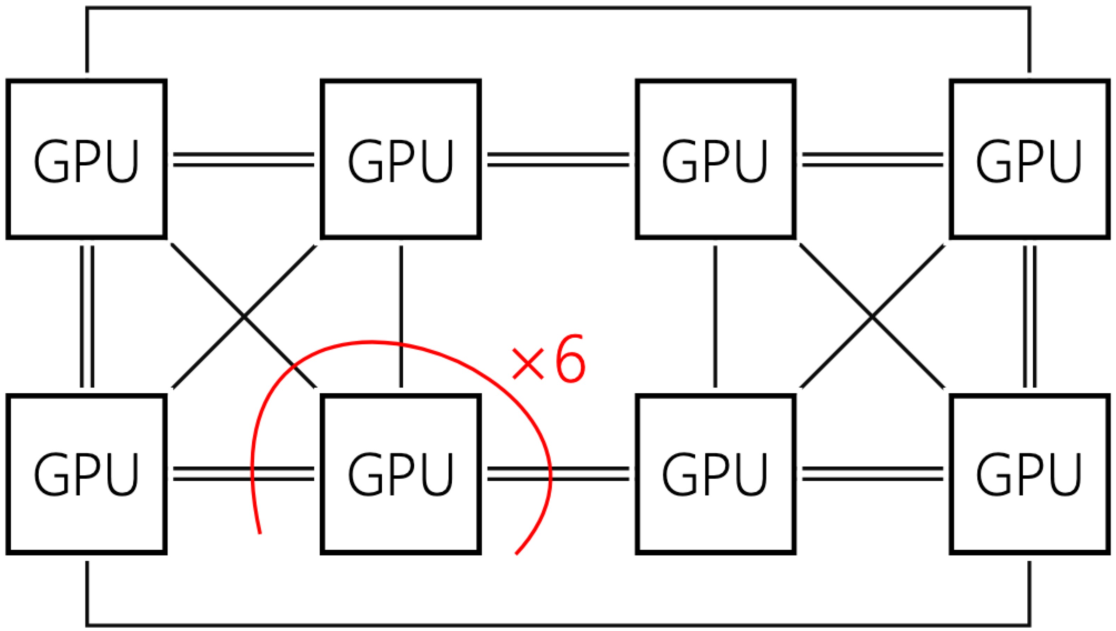


*Low bandwidth utilization: ~**30%***

Multi-GPU Extraction Mechanism

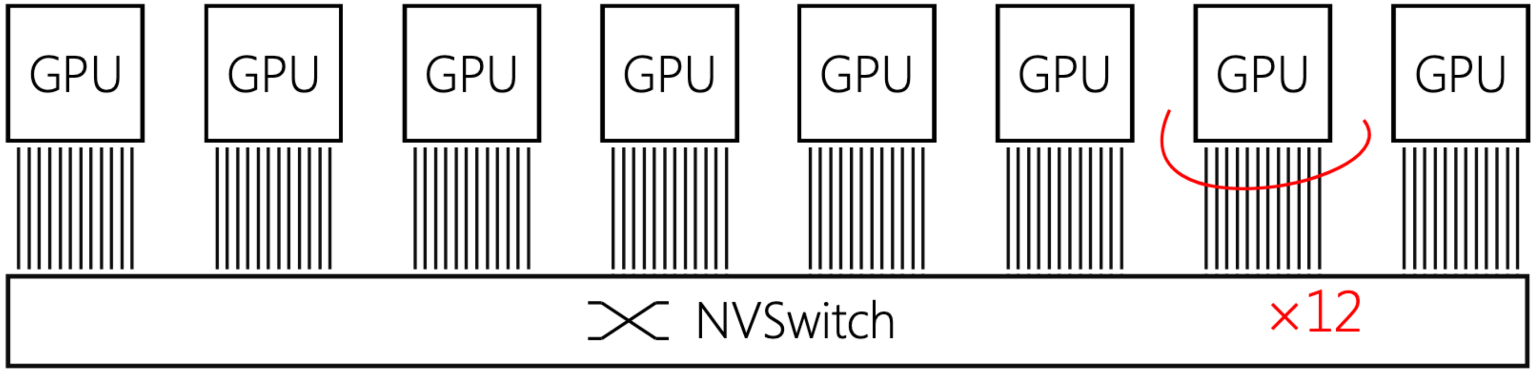
- Topology #1: Hard-wired
  - Static bandwidth partition





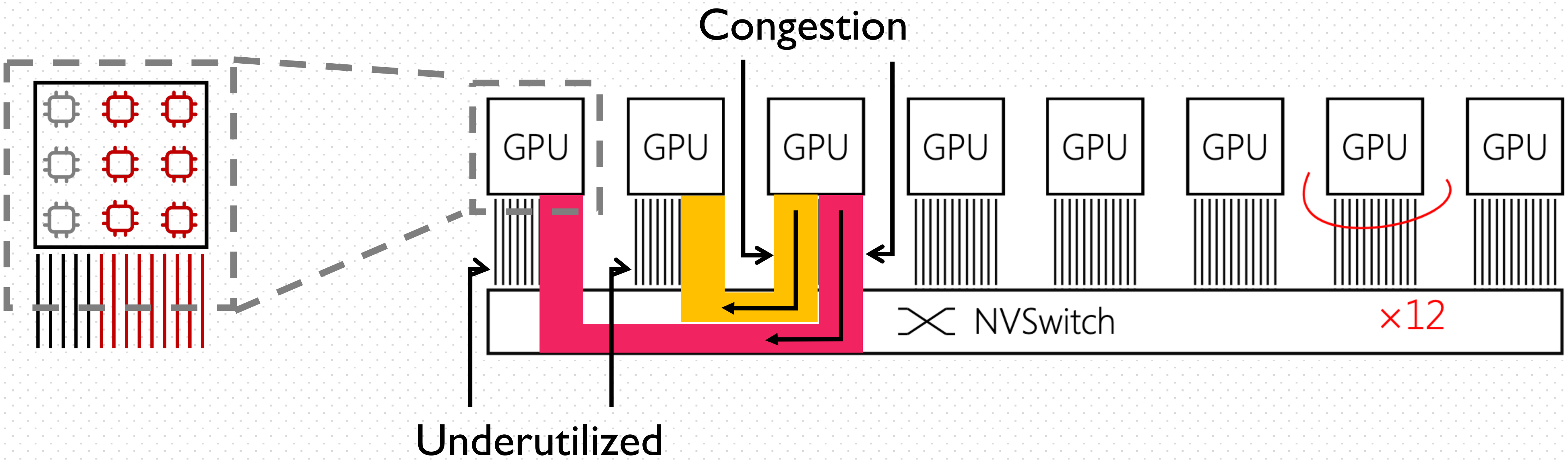
Multi-GPU Extraction Mechanism

- Topology #2: Switch-based
  - Dynamically allocates bandwidth



Multi-GPU Extraction Mechanism

- Bandwidth collision on switch-based platform



Multi-GPU Cache Challenges

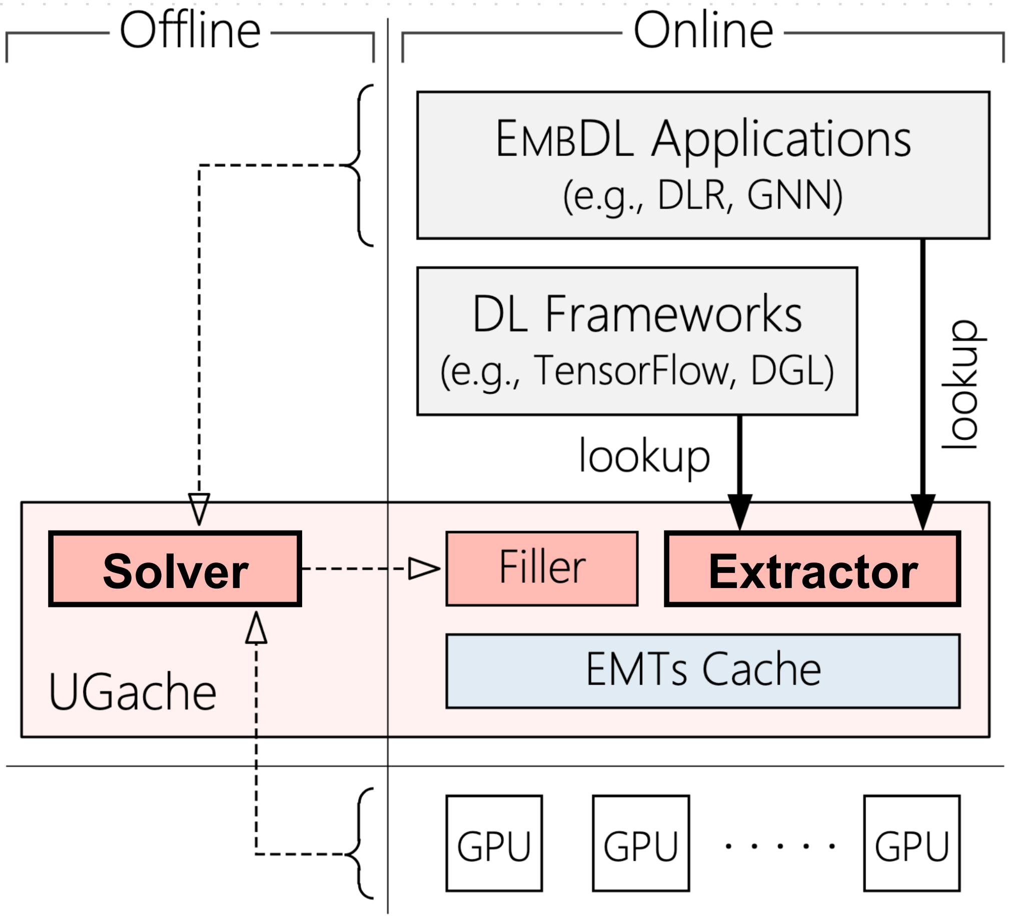
- #1: Cache Policy
  - Reduce miss rate while preserve local hit rate
- #2: Extraction Mechanism
  - Avoid congestion and improve bandwidth utilization

Outline

- Introduction
- Background and Motivation
- UGache
  - Extractor
  - Solver
- Evaluation

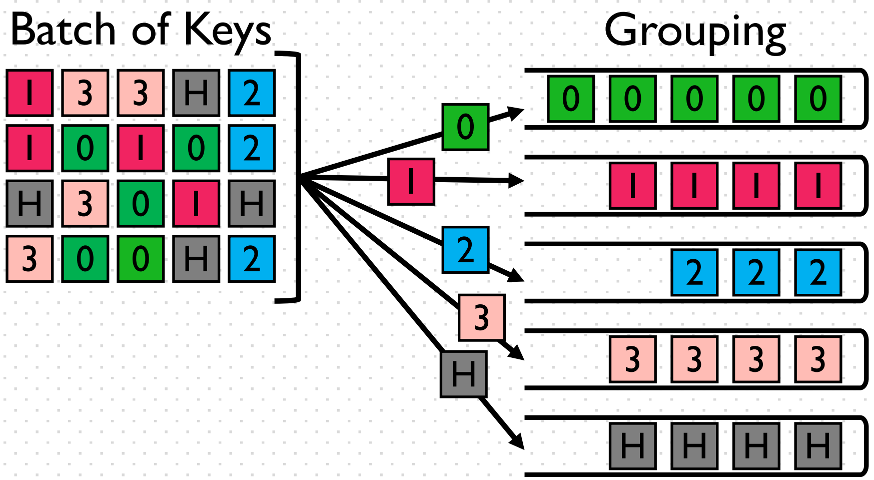
UGache

- A static embedding cache unifying multi-GPU
- Extractor (online)
    - Serve embedding extraction
- Solver (offline)
    - Provide cache policy



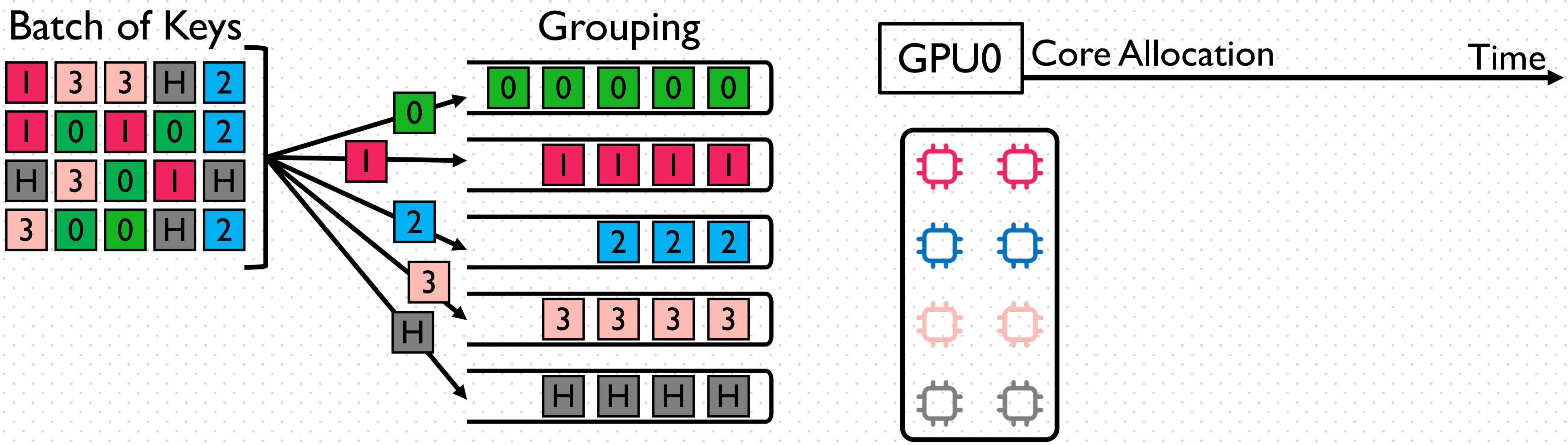
Extractor: Dedication

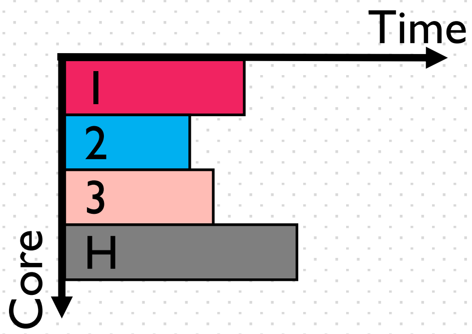
- ***Dedicate*** GPU cores to access different link



Extractor: Dedication

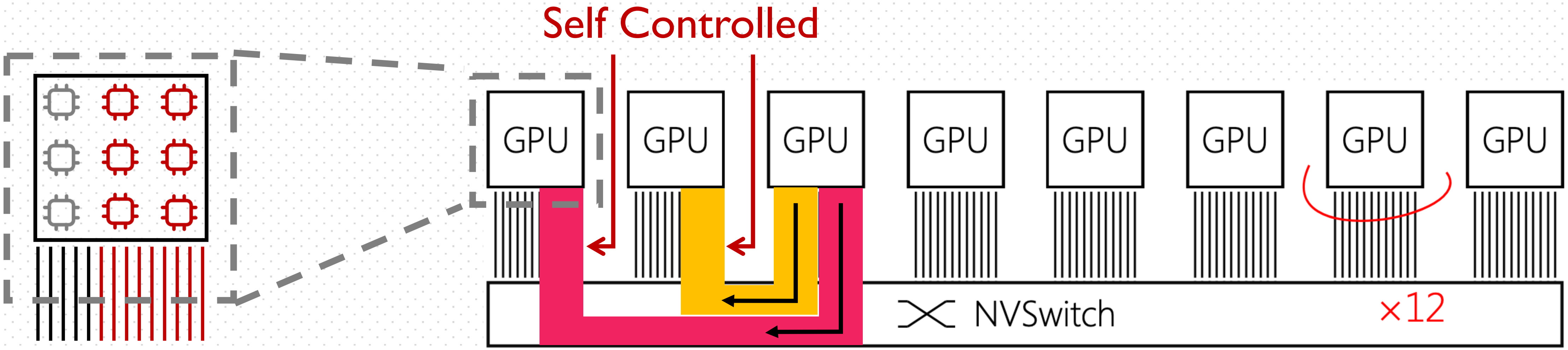
- ***Dedicate*** GPU cores to access different link





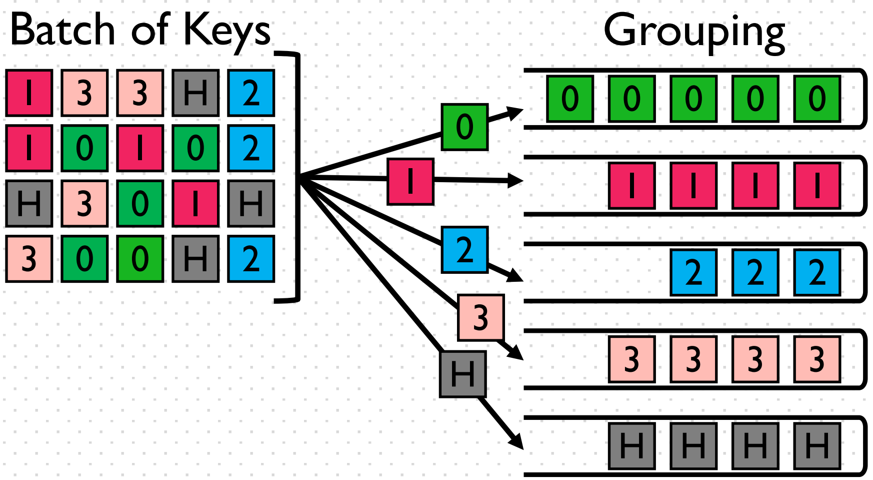
Extractor: Dedication

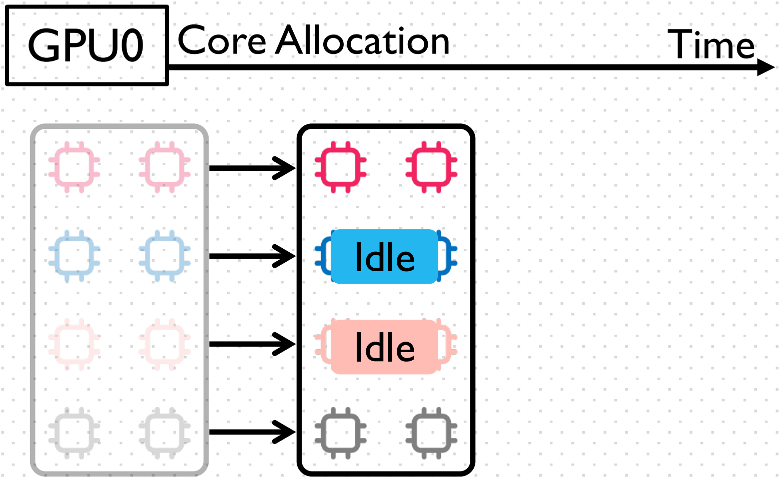
- Self controlled collision avoiding
  - No explicit coordination required

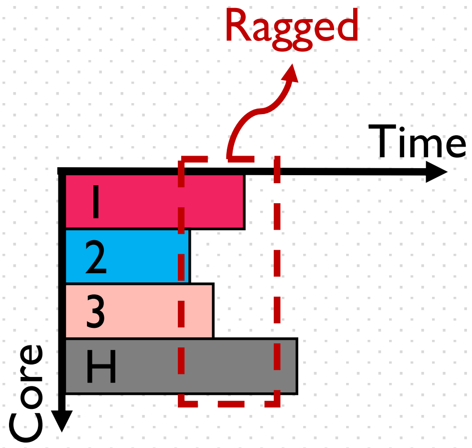


Extractor: Ragged Time

- *Dedicate* GPU cores to access different link

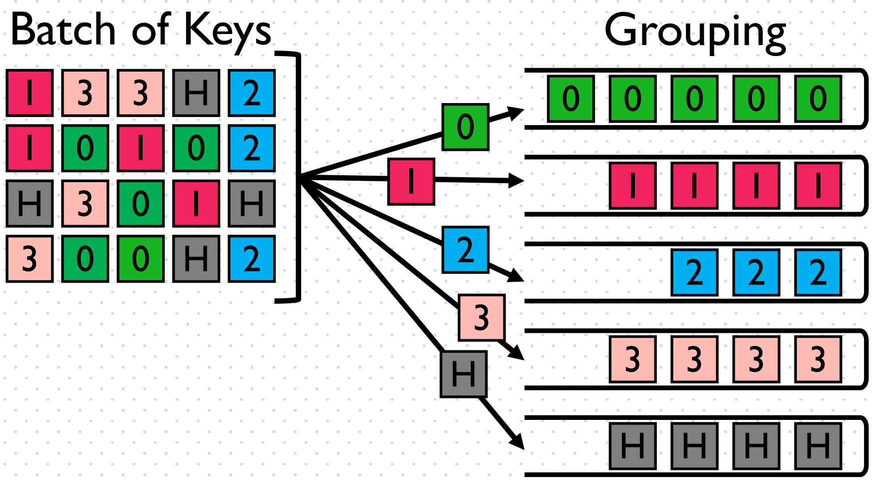


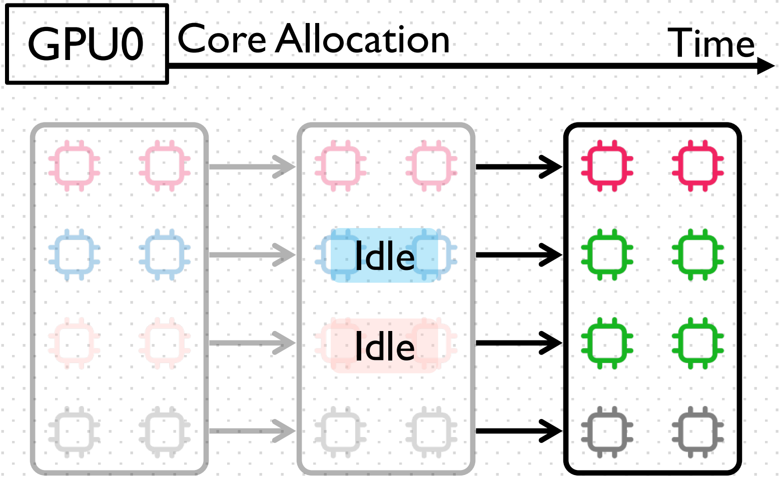


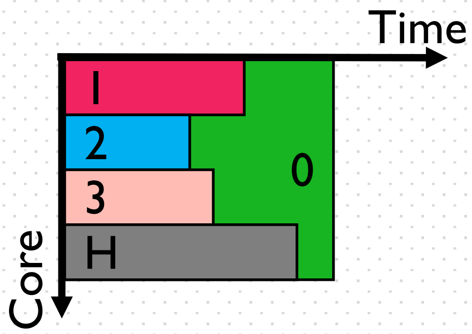


Extractor: Local Padding

- ***Dedicate*** GPU cores to access different link





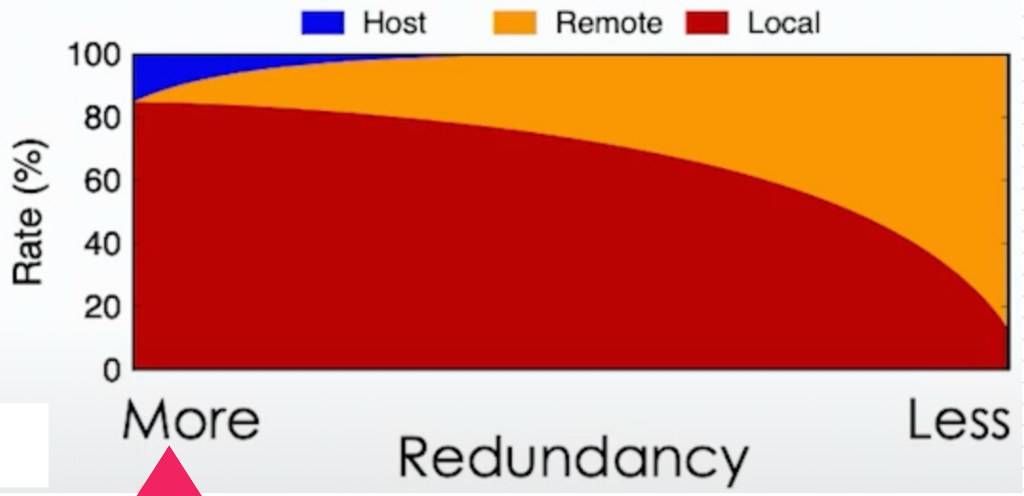


Solver: Sweet Spot of Redundancy

More
Less

Rep *Level of* ***redundancy*** Part

Duplicate more hot embedding



👍Improve *local* hit rate

🙁high *miss* rate

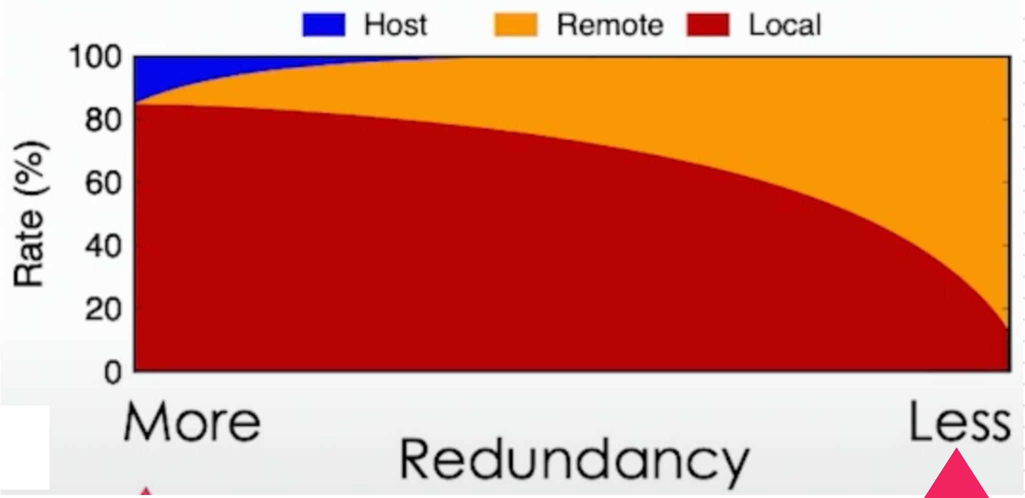
Solver: Sweet Spot of Redundancy

More | Less

Rep — *Level of **redundancy*** — Part

Duplicate more hot embedding

Cache more *distinct* embedding



👍Improve *local* hit rate

👍Reduce *host* hit rate

🙁high *miss* rate

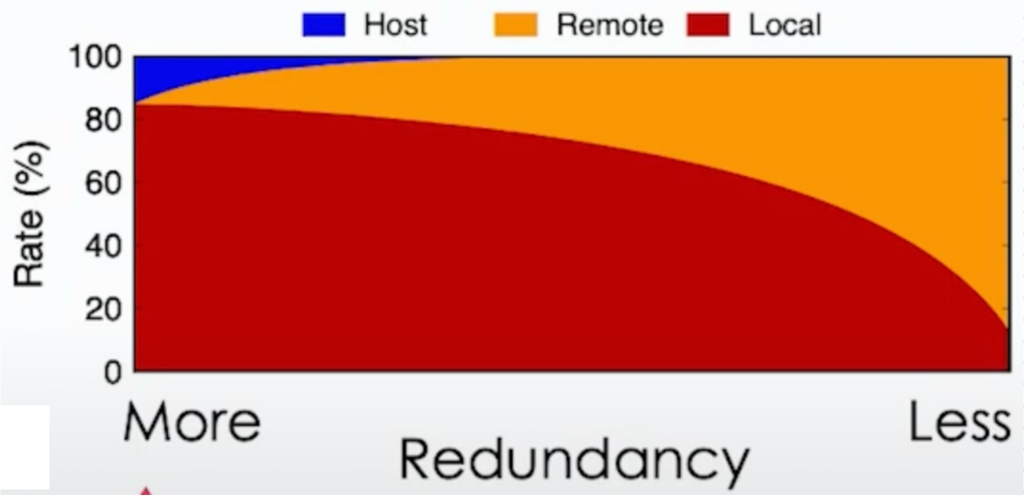
🙁Low local *hit* rate

Solver: Sweet Spot of Redundancy

More ... Less

Rep — *Level of **redundancy*** — Part

Duplicate more hot embedding

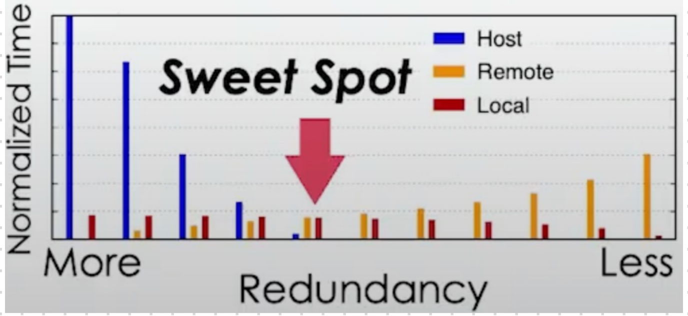


Cache more distinct embedding

👍Improve *local* hit rate

👍Reduce *host* hit rate

🙁high *miss* rate

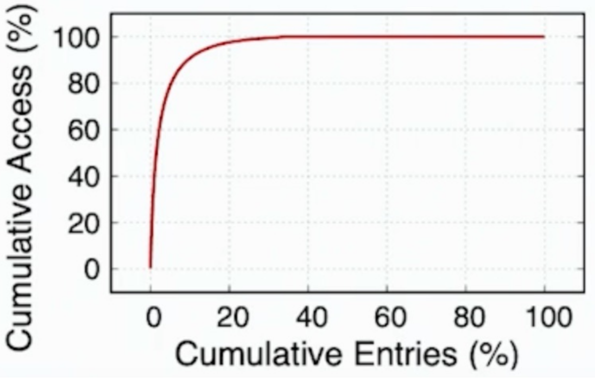


🙁Low local *hit* rate

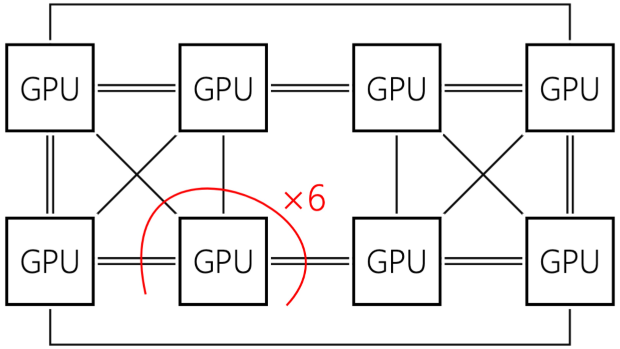
Solver: MILP-based Policy

- UGache uses *Mixed Integer Linear Programming*

Workload



Hardware



Target Function: minimize the extraction time of all GPU

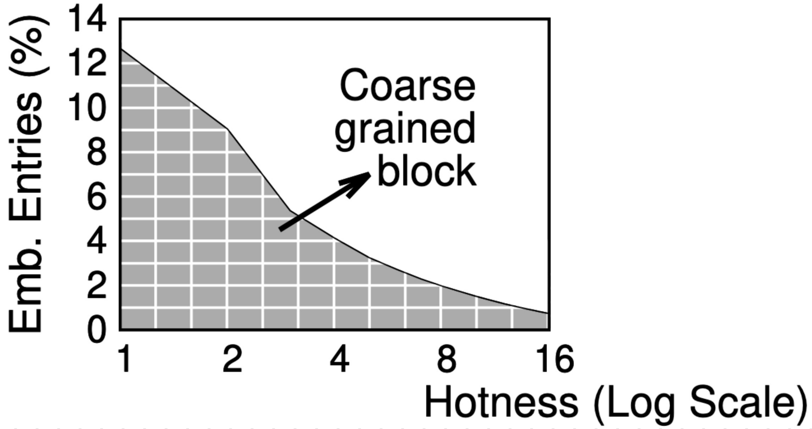
Solver

Plan for **Placement** and **Access** of embeddings

Offline

Solver: Cost Reducing

- High solving cost of MILP: $O(2^E)$
  - Entry-level decision: E is billion scale
- Batch similar embeddings
  - Billion to kilo: solve in 10s
- Hybrid batch granularity
  - Preserve accuracy: >95%

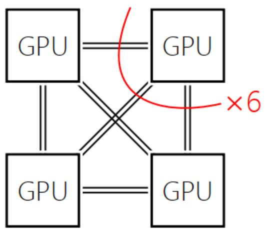


Outline

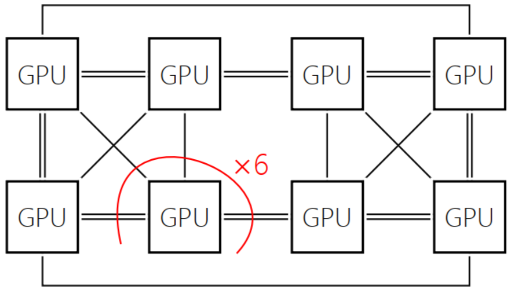
Evaluation Setup: Testbeds

- 3 servers with different topologies

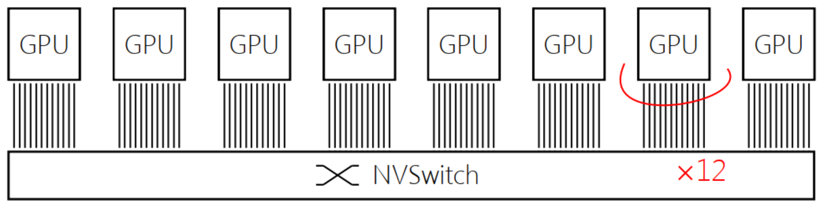
Server	GPU	Total CPU	Host memory
A	4 *V100 (16 GB)	40 cores	384 GB
B	8 *V100 (32 GB)	48 cores	724 GB
C	8 * A100 (80 GB)	56 cores	1 TB



Server A



Server B



Server C

Evaluation Setup: Applications and Datasets

- Models
  - GNN (GCN and GraphSAGE, Supervised)
  - GNN (GraphSAGE, Unsupervised)
  - DLR: DLRM and DCN
- Datasets

GNN training

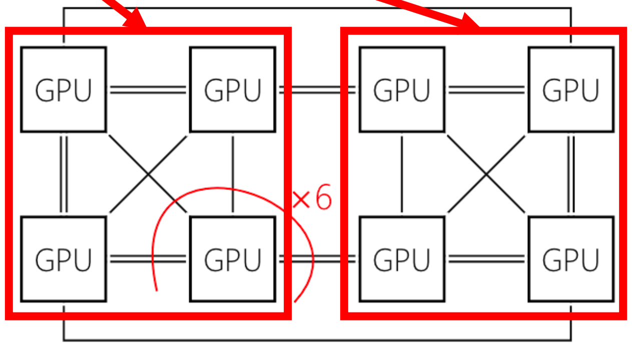
Dataset	#Vertex	#Edge	Dim.	VolumeG	VolumeE
PA	111 M	3.2 B	128	12.8 GB	53 GB
CF	65.6 M	3.6 B	256	14 GB	62 GB
MAG	232 M	3.2 B	768	13.8 GB	349 GB

DLR inference

Dataset	#Entry	#Table	Dim.	Skewness	VolumeE
CR	882 M	26	128	N/A	420.9 GB
SYN-A	800 M	100	128	1.2	381.5 GB
SYN-B	800 M	100	128	1.4	381.5 GB

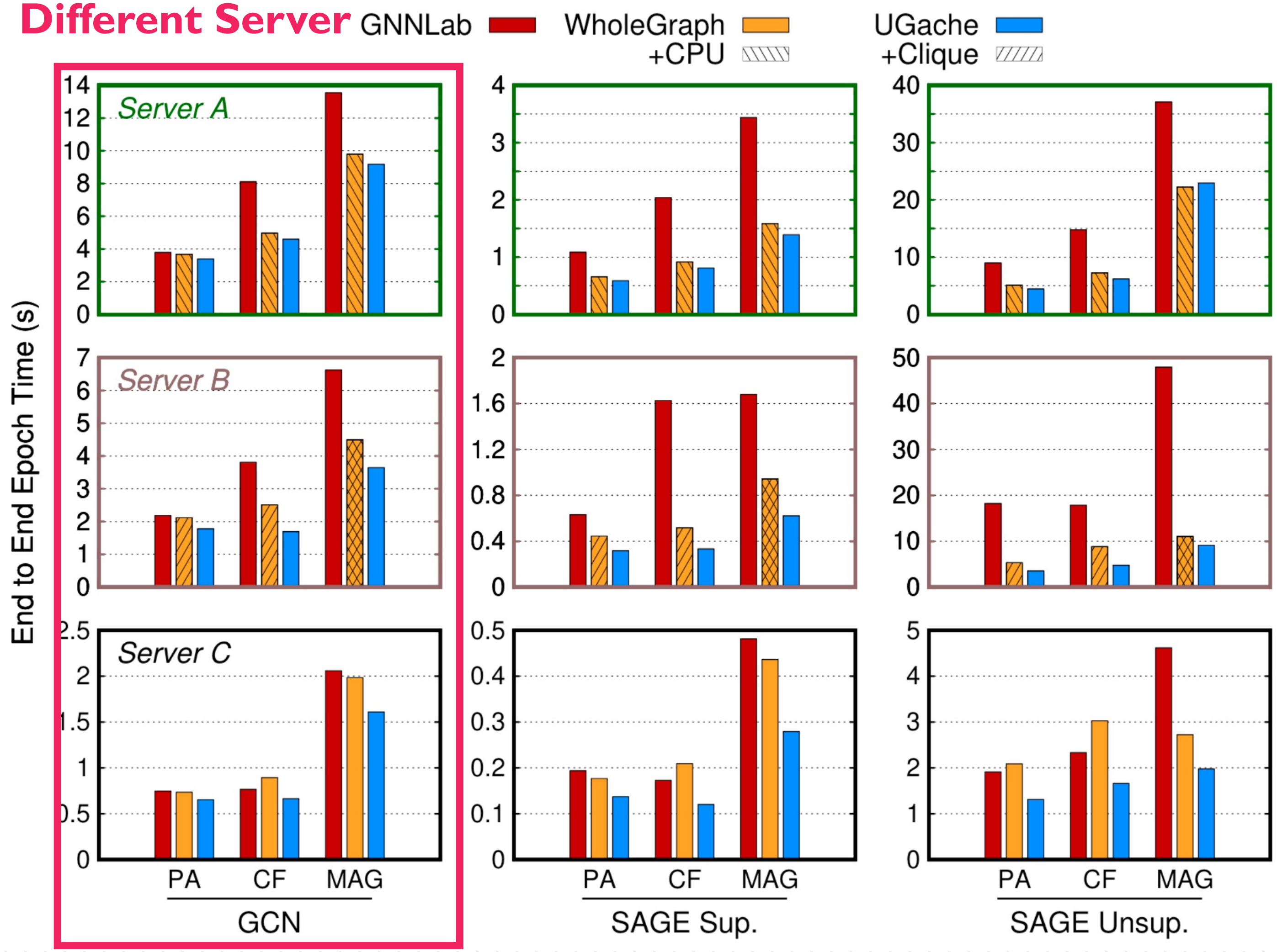
Evaluation Setup: Baselines

- GNN training
  - GNNLab: [EuroSys'22], replication approach
  - Part_U: extended from WholeGraph [SC'22], partition approach
    - Store cold embeddings in CPU (+cpu)
    - Separate Server B's 8 GPUs into two fully connect cliques (+clique)
  - Rep_U: Part_U with replication approach
- DLR inference
  - HPS: [RecSys'22], replication approach
    - Use LRU to update cache dynamically
  - SOK: by NVIDIA, partition approach
    - Conduct message-based embedding extraction



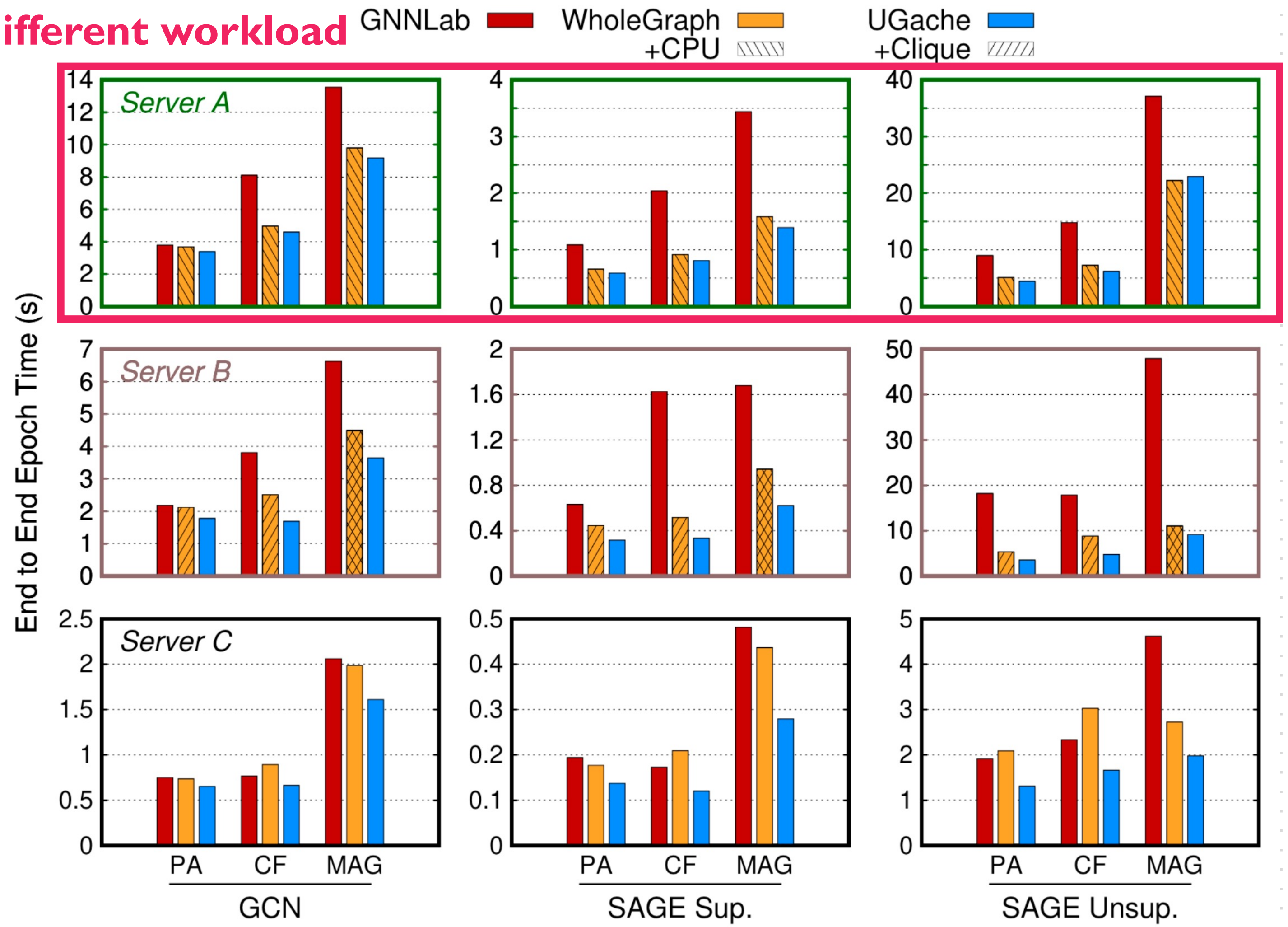
Evaluation Overall Results: GNN

Different Server

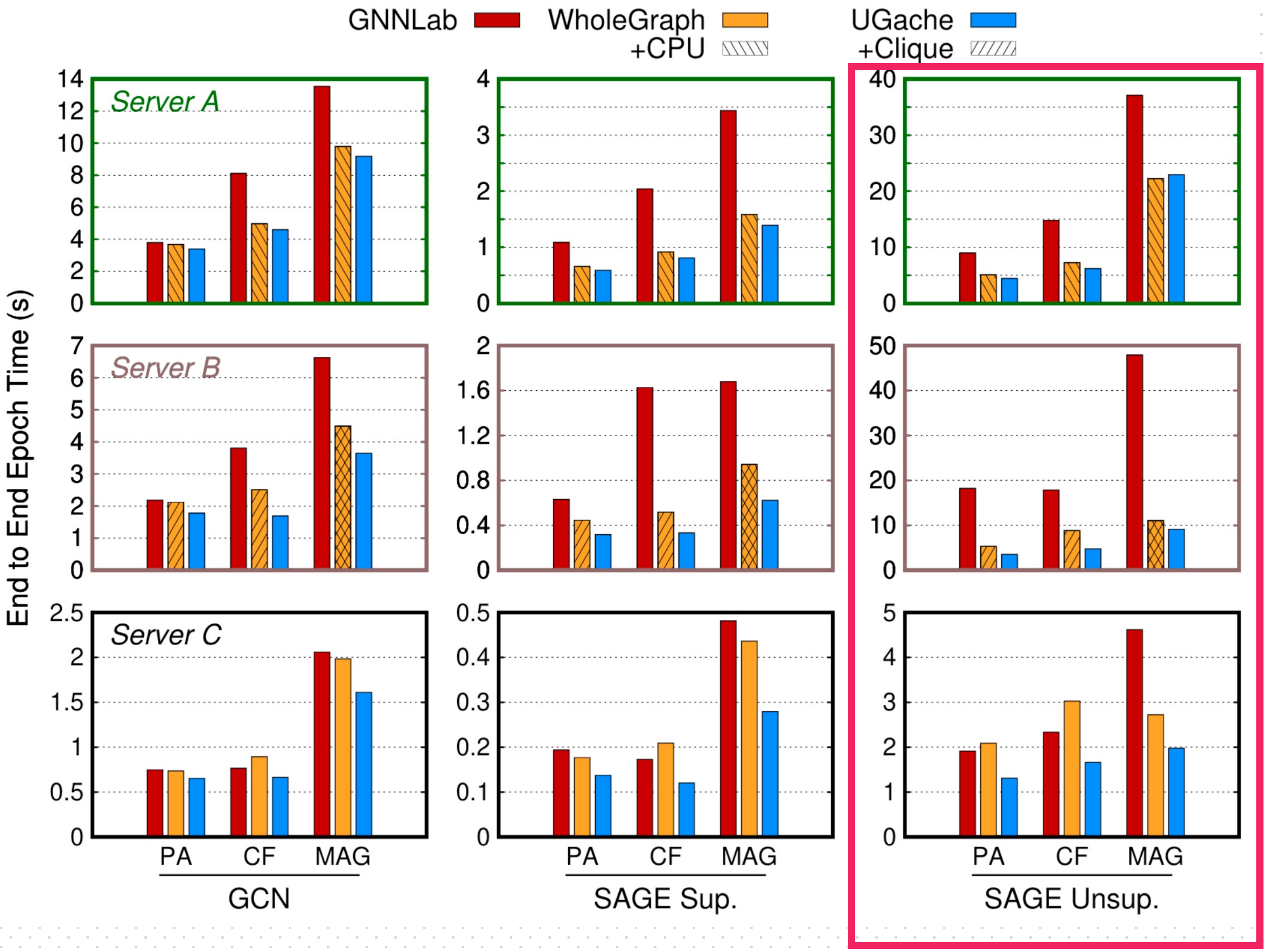


Evaluation Overall Results: GNN

Different workload



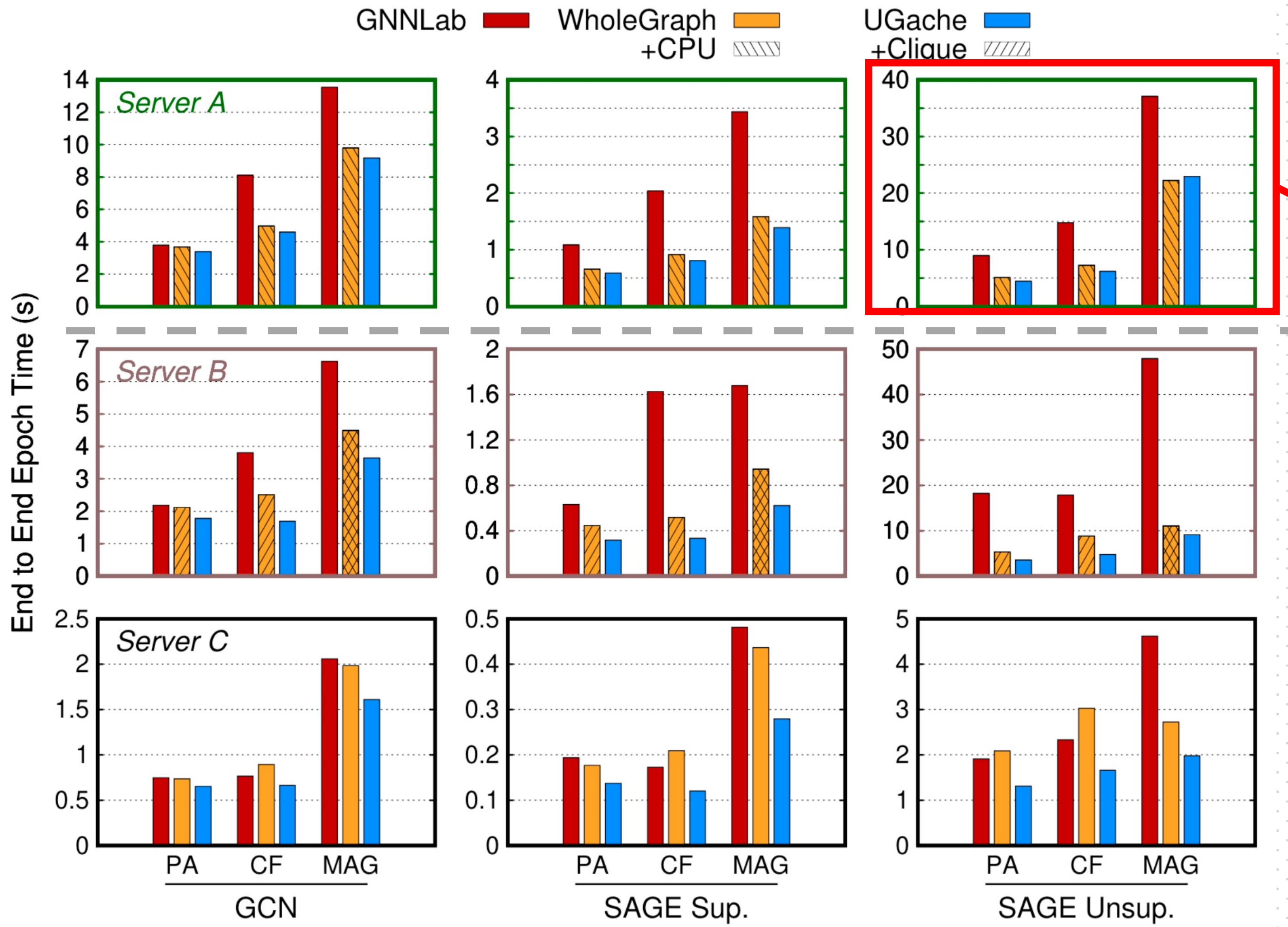
Evaluation Overall Results: GNN



- GNNLab (Replication)
  - A larger cache
  - Unsupervised SAGE leads to less skewness, so the improvement is higher

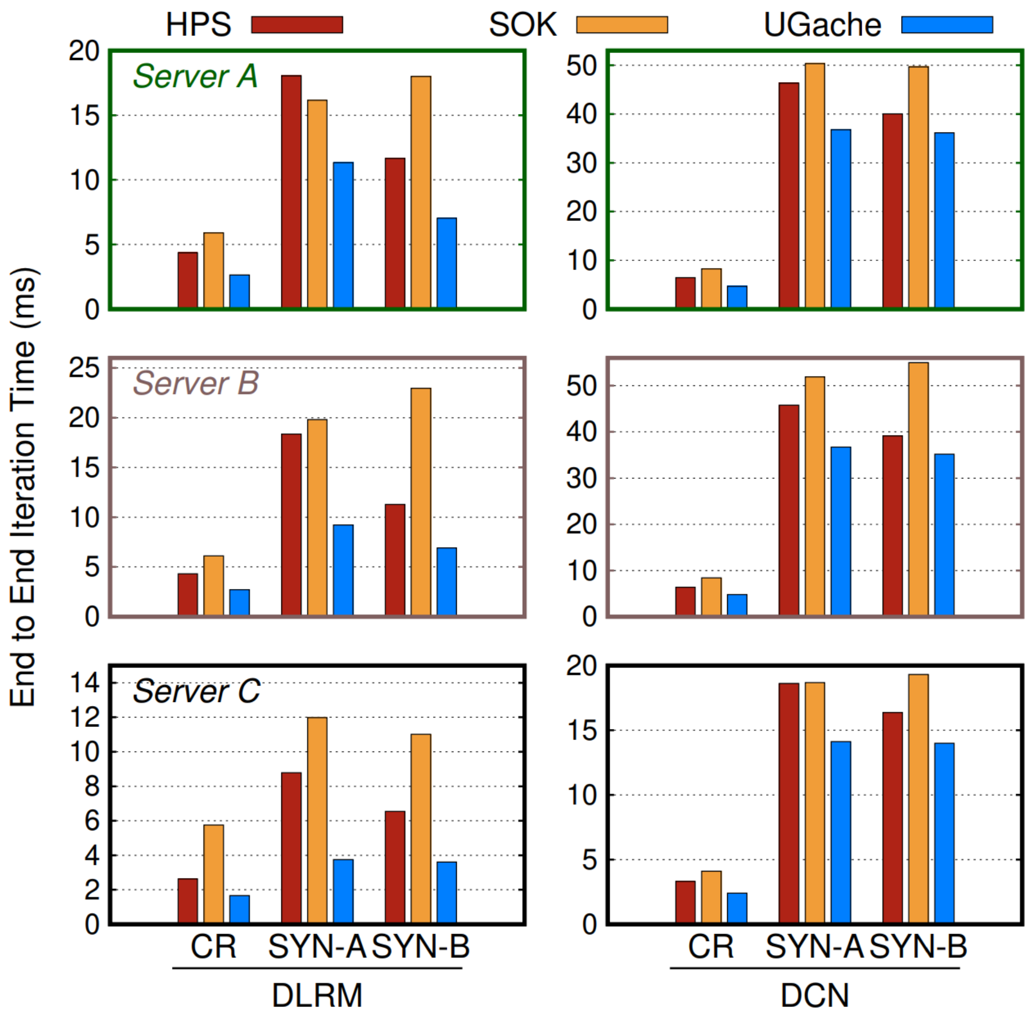
Less Skewness → Higer Improvement

Evaluation Overall Results: GNN



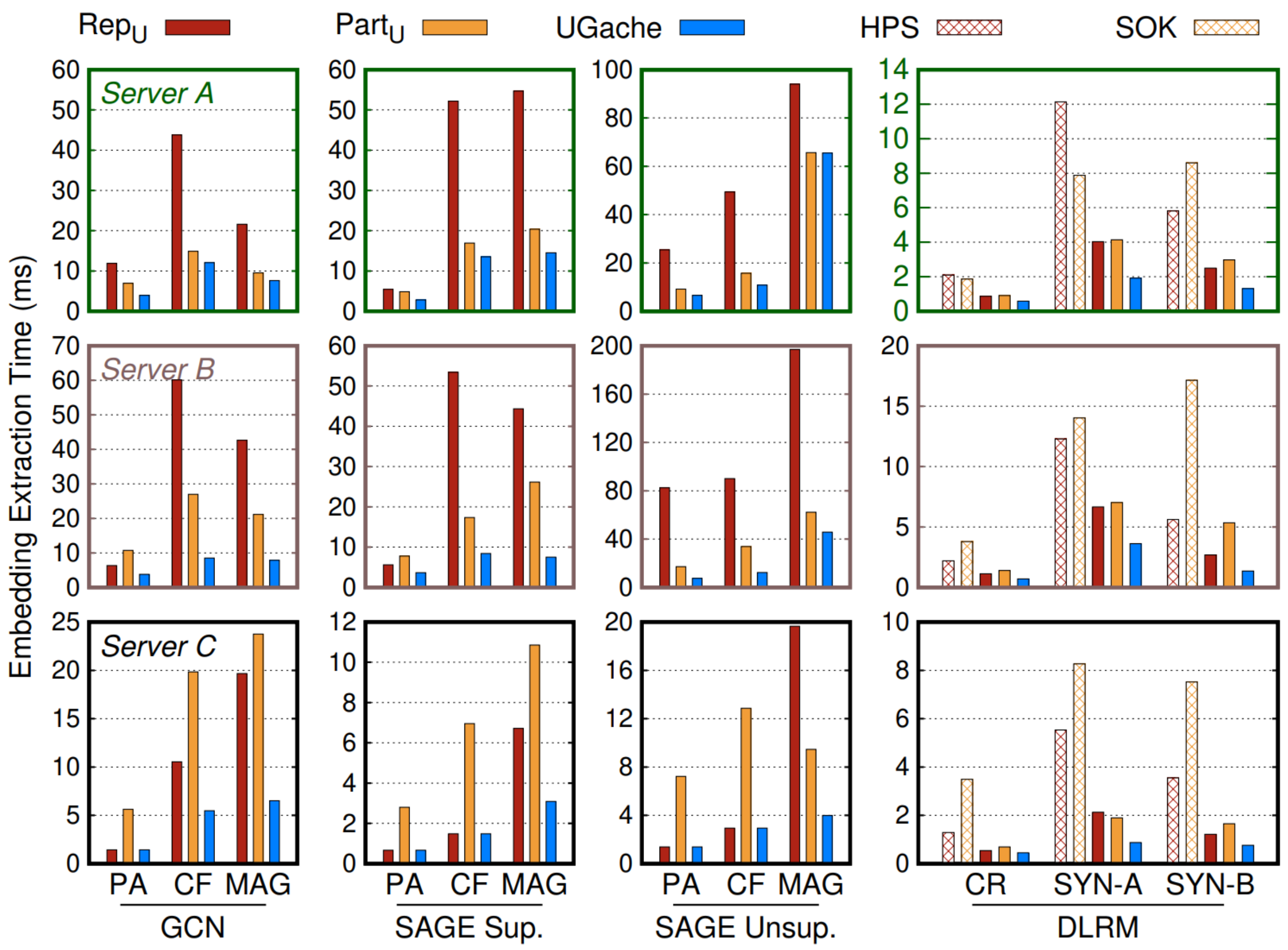
- WholeGraph (Partition)
  - In Server A
    - Host extraction dominates
    - Cost of approximate cache policy
  - In Server B,C
    - Efficiently use cache capacity
    - Fully utilize bandwidth

Evaluation Overall Results: DLR



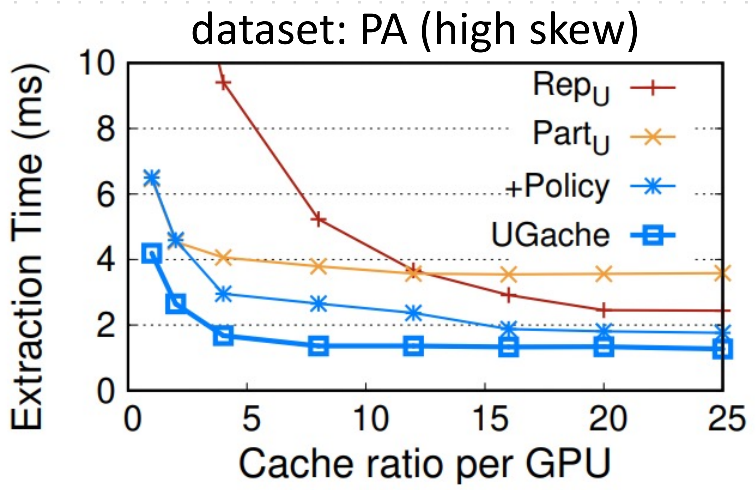
- ➢VS HPS (Replication)
  - Static cache policy is faster than LRU
- ➢VS SOK (Partition)
  - Peer-based embedding extraction is faster than message-based embedding extraction

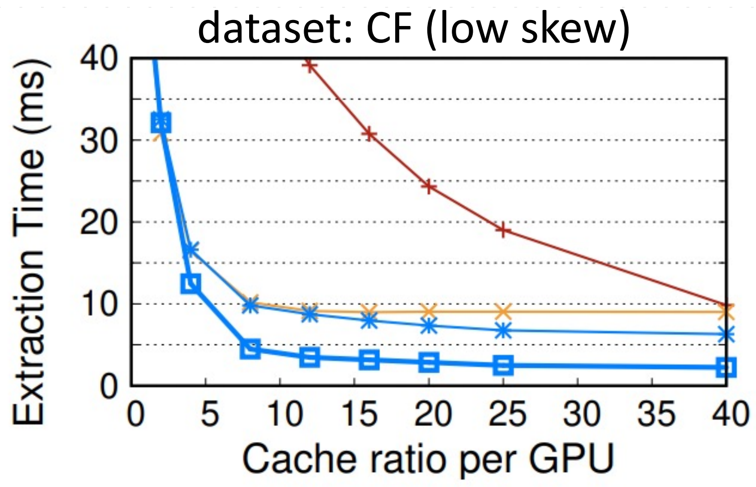
Evaluation Overall Results: Extraction Time



- GNN training
  - Similar to e2e comparison
    - Red stands for GNNLab
    - Yellow stands for WholeGraph
- DLR
  - HPS vs Rep_U
    - Rep_U avoids online eviction
  - SOK vs $Part_U$
    - $Part_U$ uses peer-based extraction
  - UGache vs Rep_U and $Part_U$
    - Ugache is near optimal

Performance Breakdown: Cache Ratio

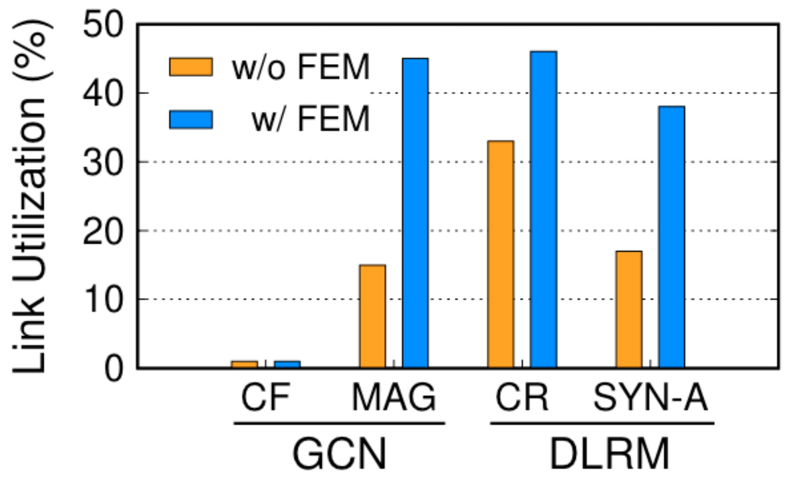




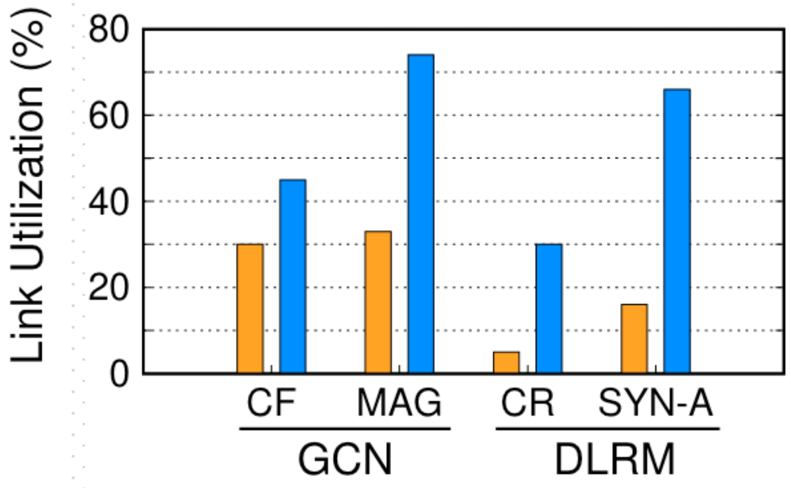
GraphSAGE sup. on 8xA100

- Small cache ratio
  - Rep$_U$ is inefficient
  - Part$_U$ and +Policy are close
    - Hottest data should be cached
  - UGache further improves
    - Factored extraction mechanism
- Increased cache ratio
  - +Policy outperforms Part$_U$
    - Balance local and global hit rate
- Dataset influence
  - Divergence point is affected by skewness

Evaluation: Bandwidth Utilization



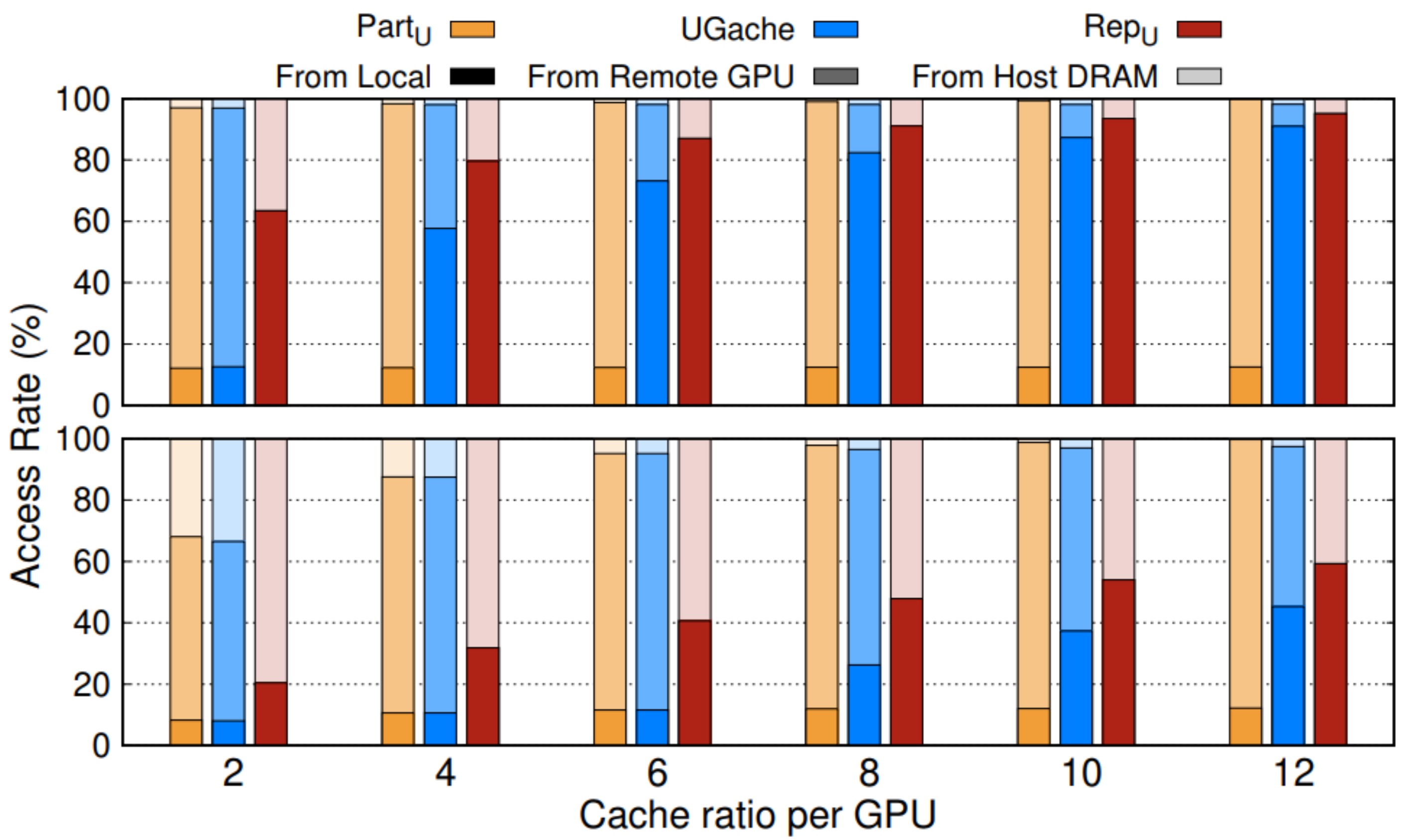
Utilization of PCIe



Utilization of NVLink

- ➢ Setting
  - Remove local hits
  - Only remote and host
- ➢ FEM improves utilization
  - Avoid congestion
- ➢ In GCN + CF
  - Small dataset, high cache ratio
  - Not much non-local access
  - Slight improvement

Evaluation: Cache Access Distribution

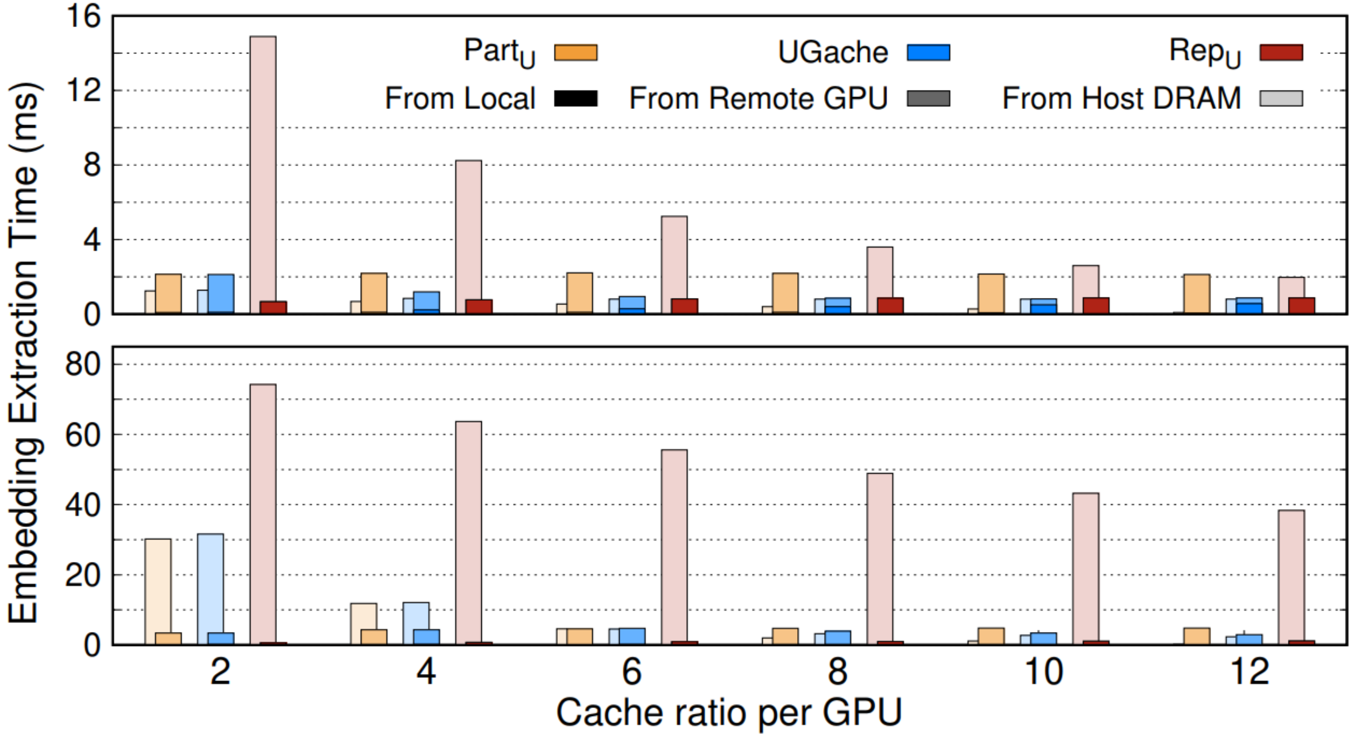


Hit rate of local GPU, remote GPU and host, in PA (top, high skew) and CF (bottom, low skew)

➢Hit Rate

- Low cache ratio (2%)
  - Rep_U frequently access host
  - $Part_U$ and UGache are similar
    - Cache hottest entries first
- Increased cache ratio (8%)
  - Rep_U still needs host access
  - $Part_U$ doesn't change much
  - UGache improves local hit rate
    - Slightly lower global hit rate

Evaluation: Cache Policy



Extraction time of local GPU, remote GPU and host, in PA (top, high skew) and CF (bottom, low skew)

- Extraction time
  - Rep$_U$: Suffers from host access
  - Part$_U$: Avoids host access in small cache and remote hits take a long time
- UGache
  - Balances global and local hits
  - Scales well in low skew CF

Summary

- A study of multi-GPU embedding cache
- UGache:
  - Factored extraction mechanism
  - MILP-based Cache policy with low-cost solving