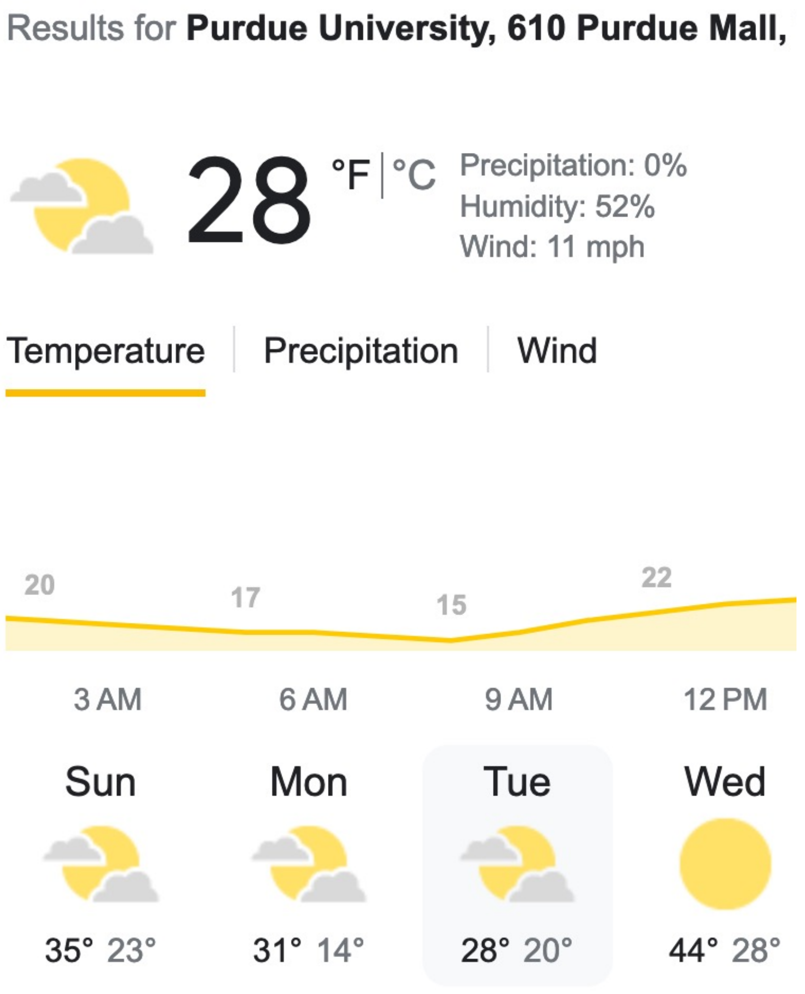
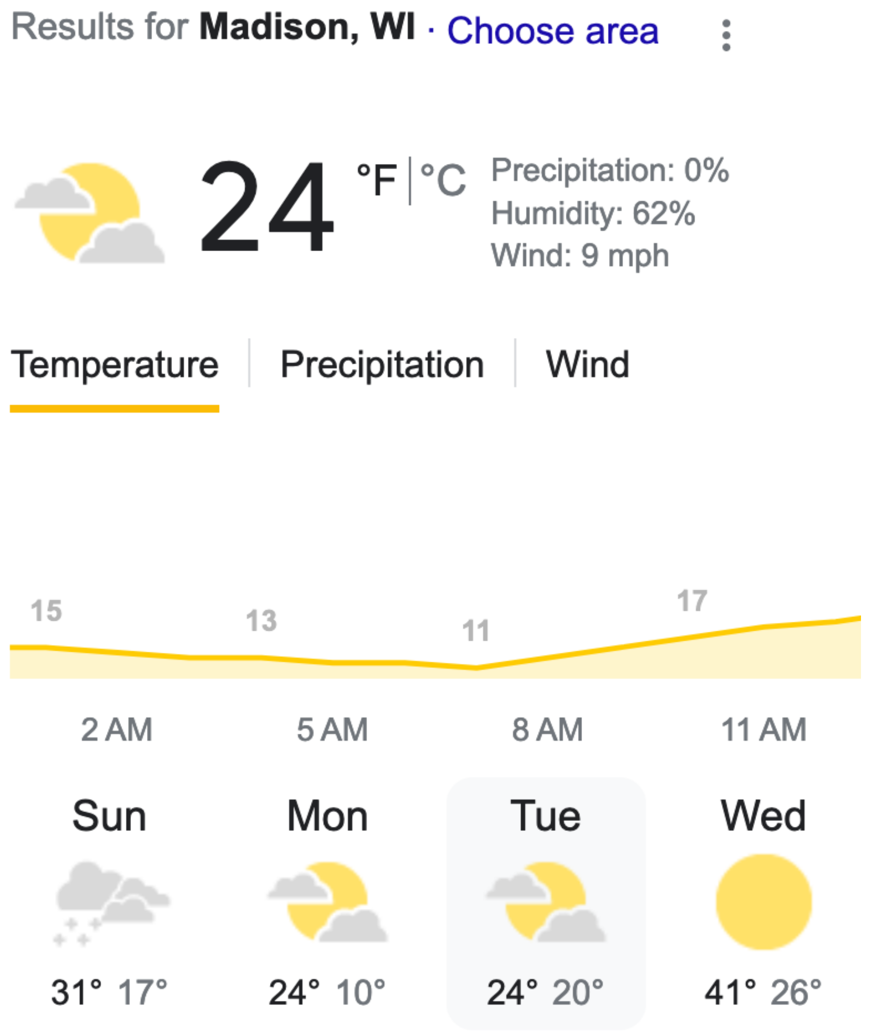
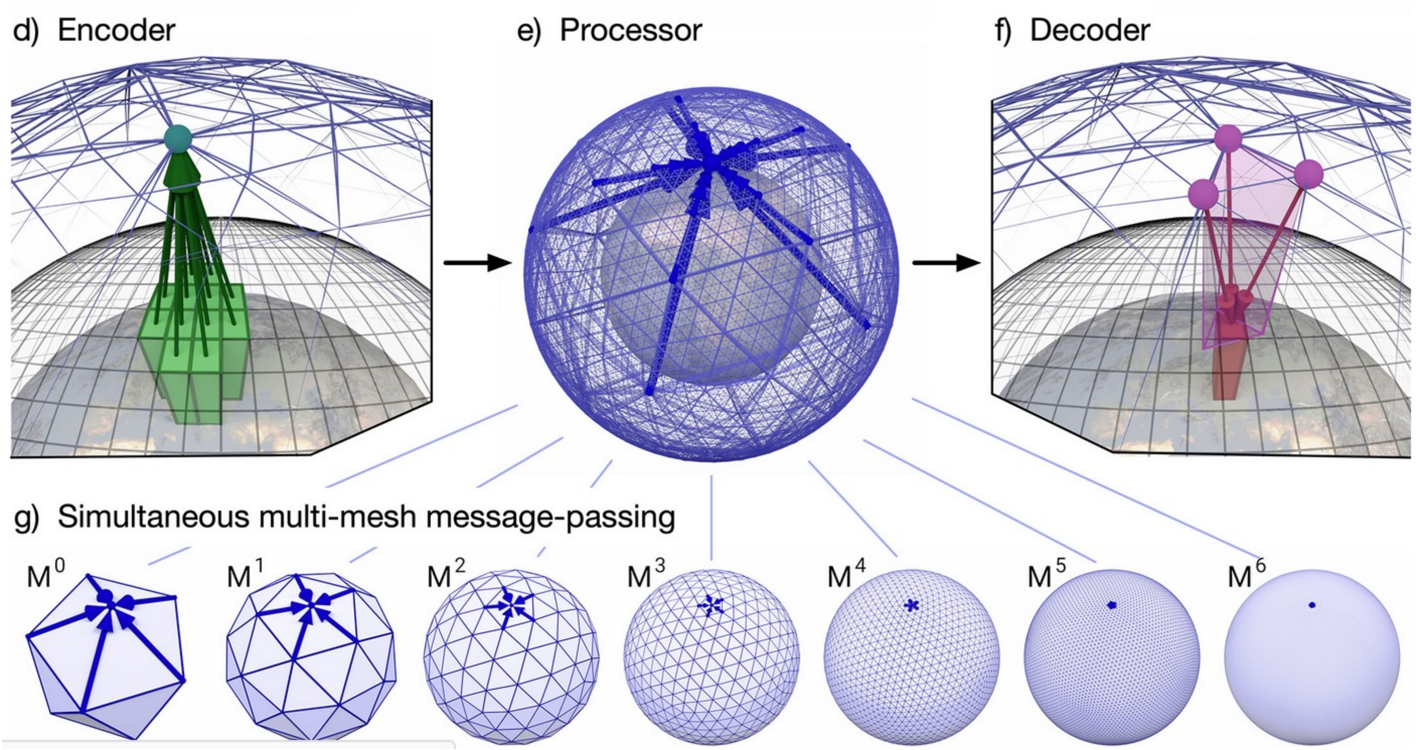
RESOURCE EFFICIENT LARGE SCALE ML: PLAN BEFORE YOU RUN

Shivaram Venkataraman
University of Wisconsin, Madison


Results for Purdue University, 610 Purdue Mall,
28 °F | °C
Precipitation: 0%
Humidity: 52%
Wind: 11 mph
Temperature
Precipitation
Wind
20
17
15
22
3 AM
6 AM
9 AM
12 PM
Sun
35° 23°
Mon
31° 14°
Tue
28° 20°
Wed
44° 28°


Results for Madison, WI · Choose area
24 °F | °C
Precipitation: 0%
Humidity: 62%
Wind: 9 mph
Temperature
Precipitation
Wind
15
13
11
17
2 AM
5 AM
8 AM
11 AM
Sun
31° 17°
Mon
24° 10°
Tue
24° 20°
Wed
41° 26°

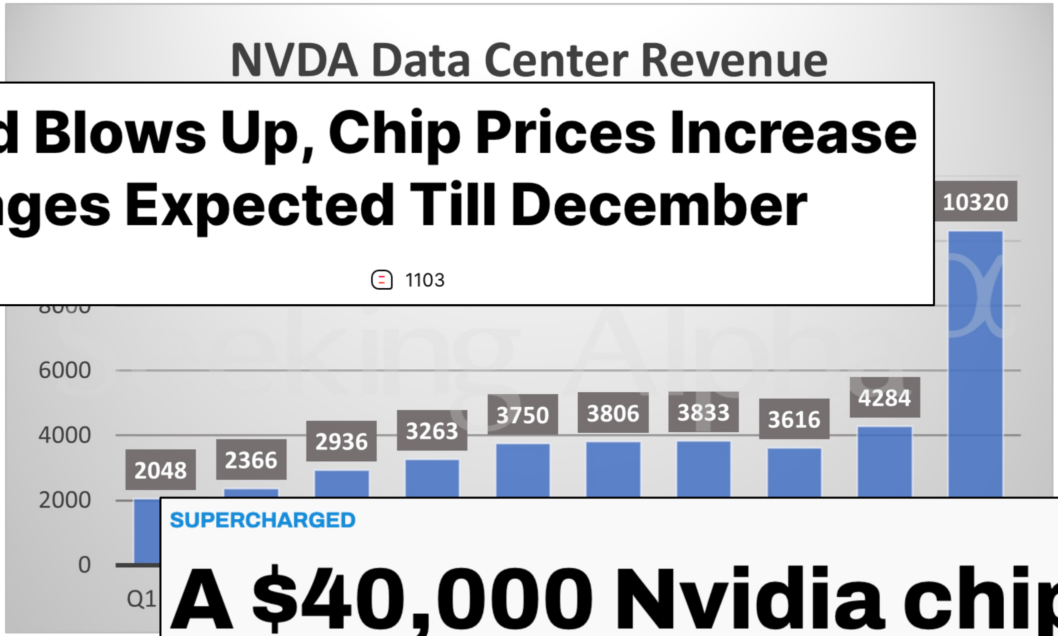
GraphCast: AI model for faster and more accurate global weather forecasting



While GraphCast's training was ***computationally intensive,*** *the resulting forecasting model is highly efficient…*

https://deepmind.google/discover/blog/graphcast-ai-model-for-faster-and-more-accurate-global-weather-forecasting/

INCREASE DEMAND, RISING COSTS



NVIDIA AI GPU Demand Blows Up, Chip Prices Increase By 40% & Stock Shortages Expected Till December

Hassan Mujtaba · May 22, 2023 02:10 PM EDT · Copy Shortlink 1103

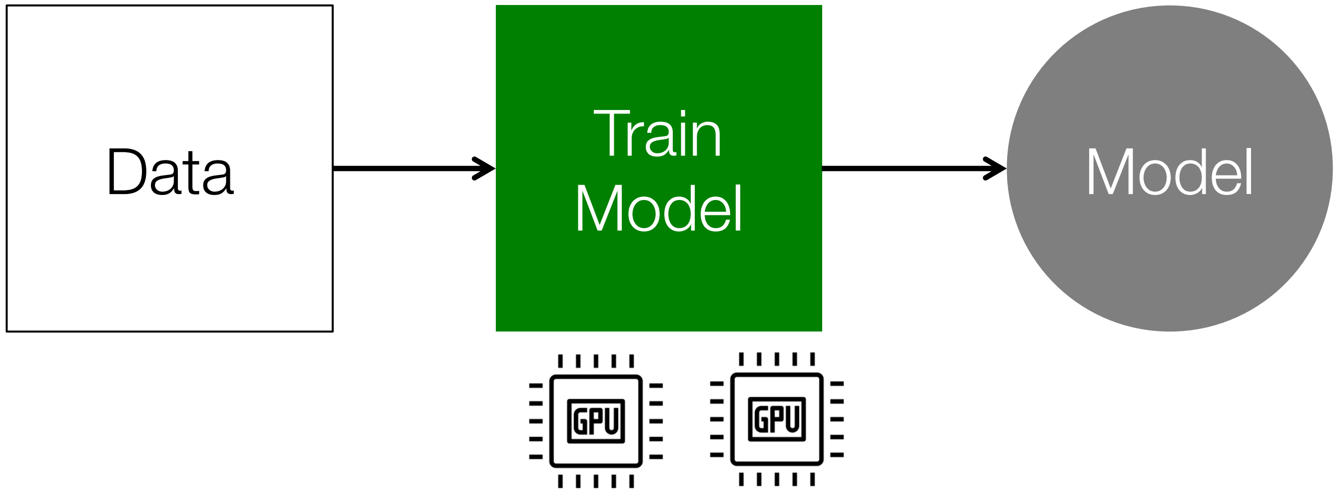
SUPERCHARGED

A $40,000 Nvidia chip has become the world's most sought-after hardware

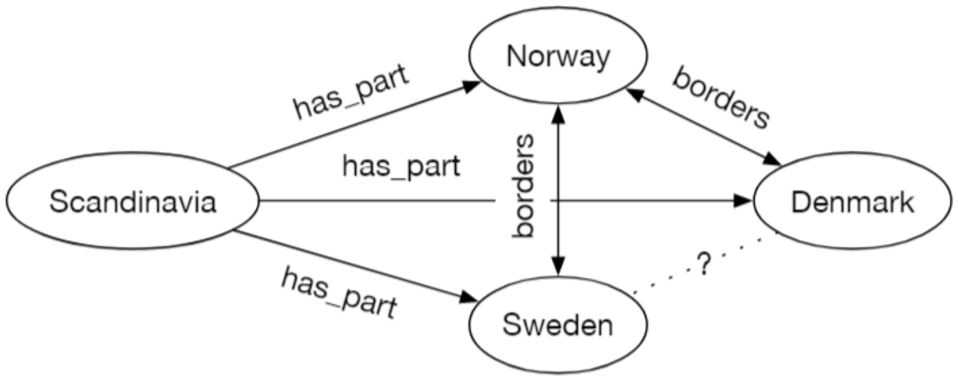
RESEARCH GOAL: RESOURCE EFFICIENT ML

Approach: Design efficient systems which can ***plan*** resource use given structure of ML workloads

MACHINE LEARNING WORKFLOW



Data access for training on large structured data



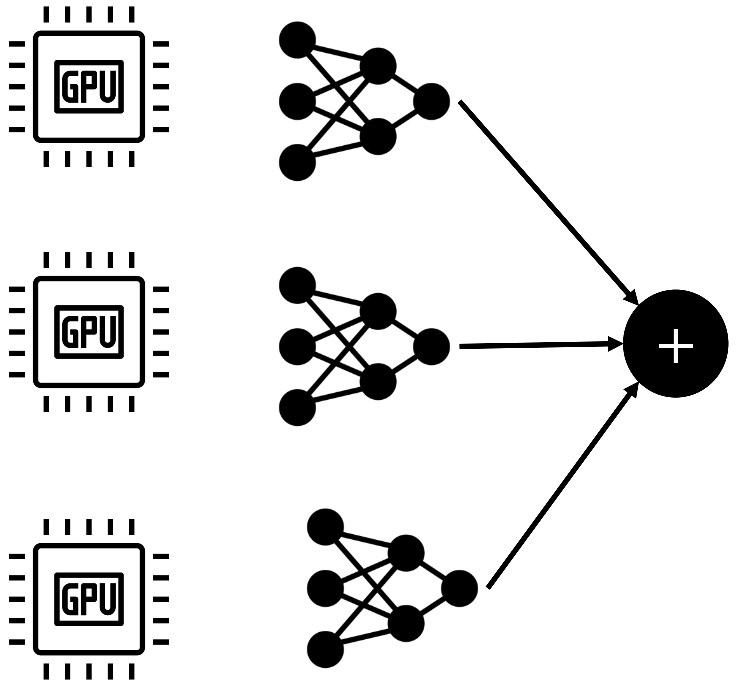
Marius [OSDI 2021, VLDB 2021], Marius GNN [Eurosys 2023], BagPipe [SOSP 2023]

Data access

Synchronization

Cluster Scheduling

Optimize communication during distributed training



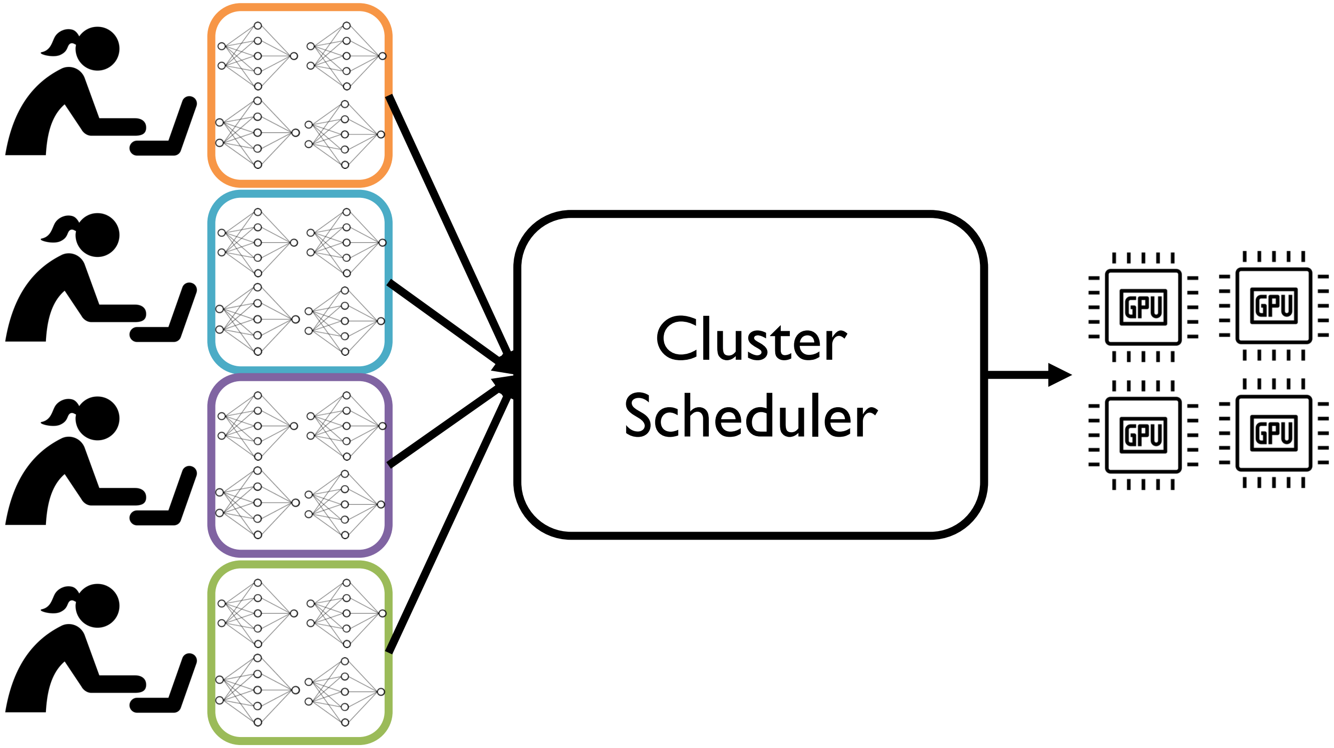
Blink [MLSys 2020], Accordion [MLSys 2021], KAISA [SC 2021],
Understanding Gradient Compression [MLSys 2022, arxiv 2301.02654]

Data access

Synchronization

Cluster Scheduling

Policies and mechanisms for scheduling on shared clusters



Philly [ATC 2019], Themis [NSDI 2020], Shockwave [NSDI 2023], Variability analysis [SC 2022], Mirage [SC 2023], Blox [Eurosys 2024]

THIS TALK

Data access

Data access for training on large graph structured data

Synchronization

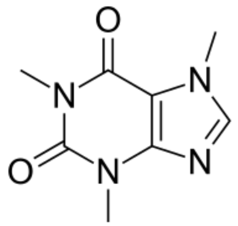
Optimize communication during distributed training

Cluster Scheduling

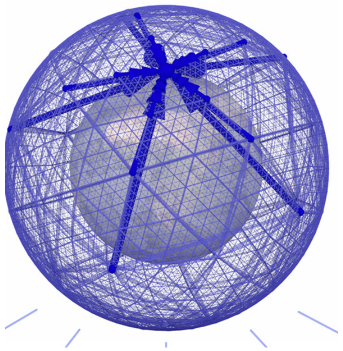
Policies and mechanisms for scheduling on shared clusters

ML ON GRAPH STRUCTURED DATA

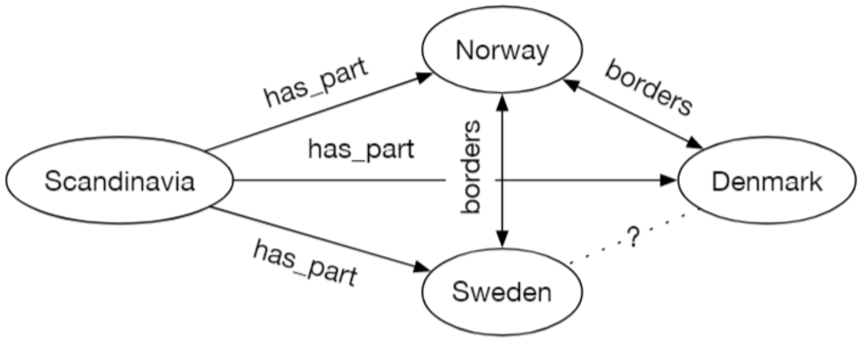
Social networks



Protein structure



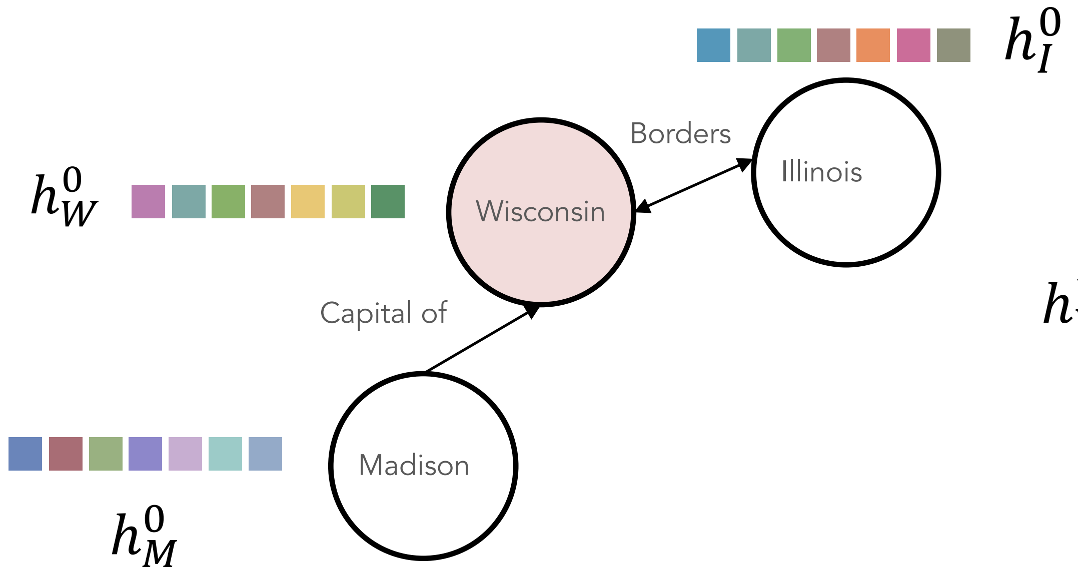
Weather forecast



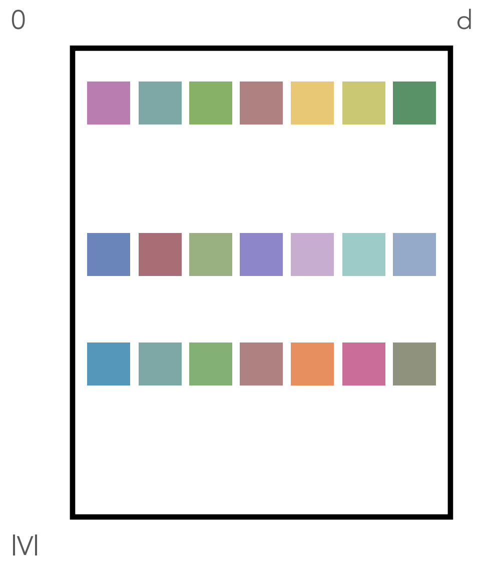
Knowledge graphs

EXAMPLE: GRAPH NEURAL NETWORKS (GNN)

Graph Neural Networks: Use NN to capture neighborhood structure



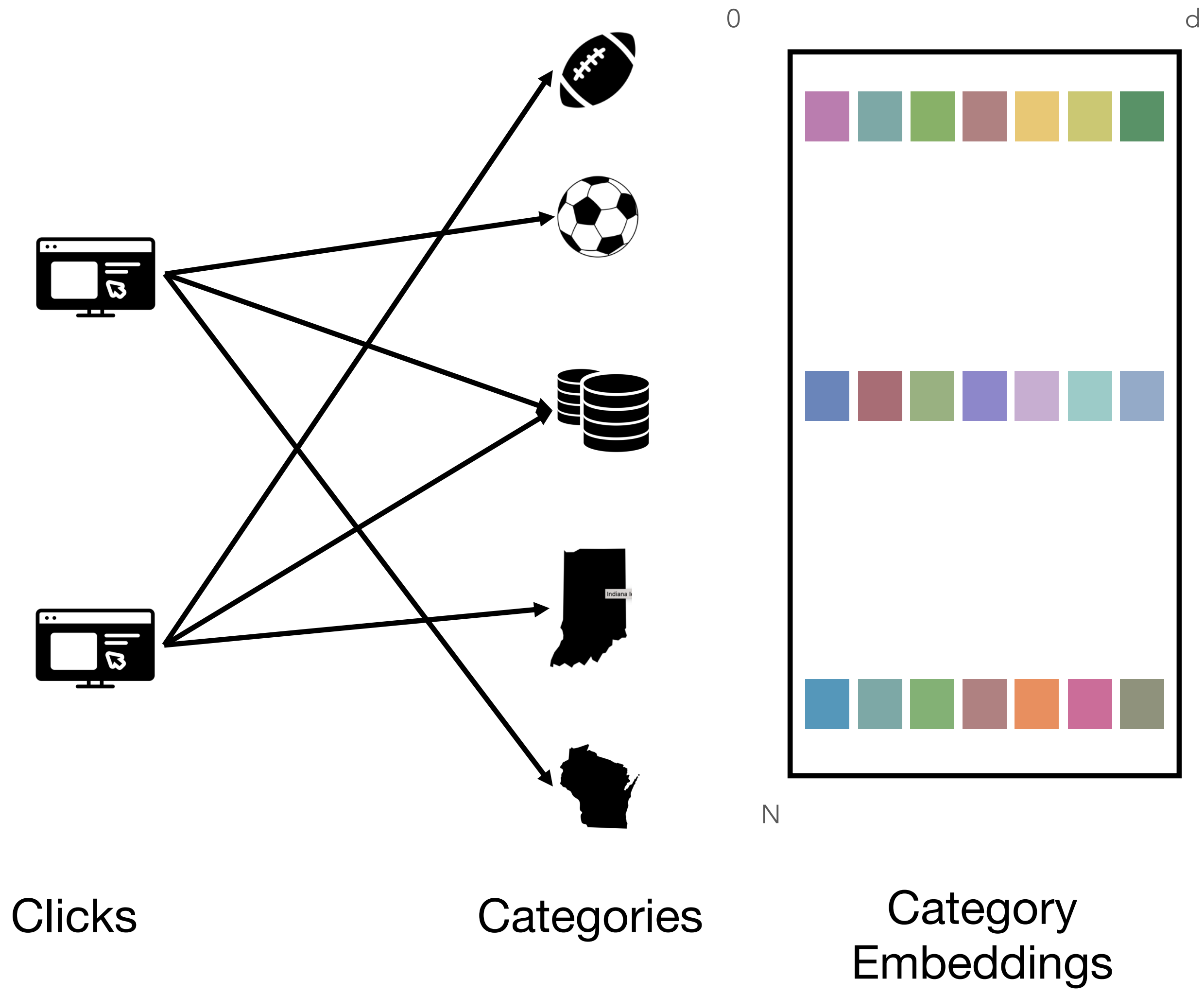
$$h_W^1 = AGG(h_W^0, \{h_I^0, h_M^0\})$$



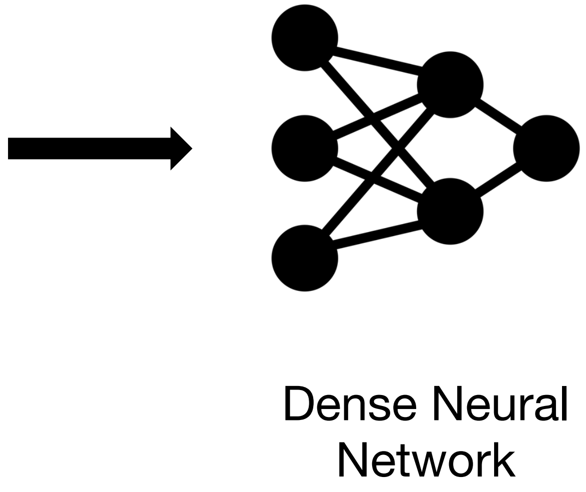
Vertex Embeddings

Example GNNs: GraphSage, Graph Convolution Network, GAT

EXAMPLE: RECOMMENDATION MODELS



Examples: DLRM, DeepFM,
Wide & Deep



LARGE GRAPHS

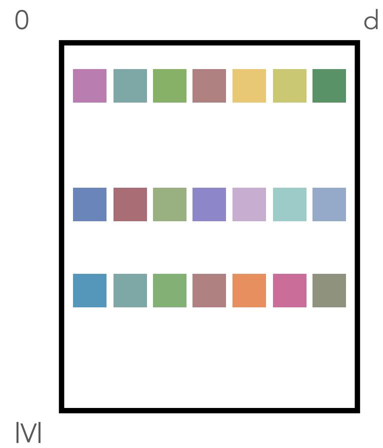
Large graphs → Large embedding models

Example

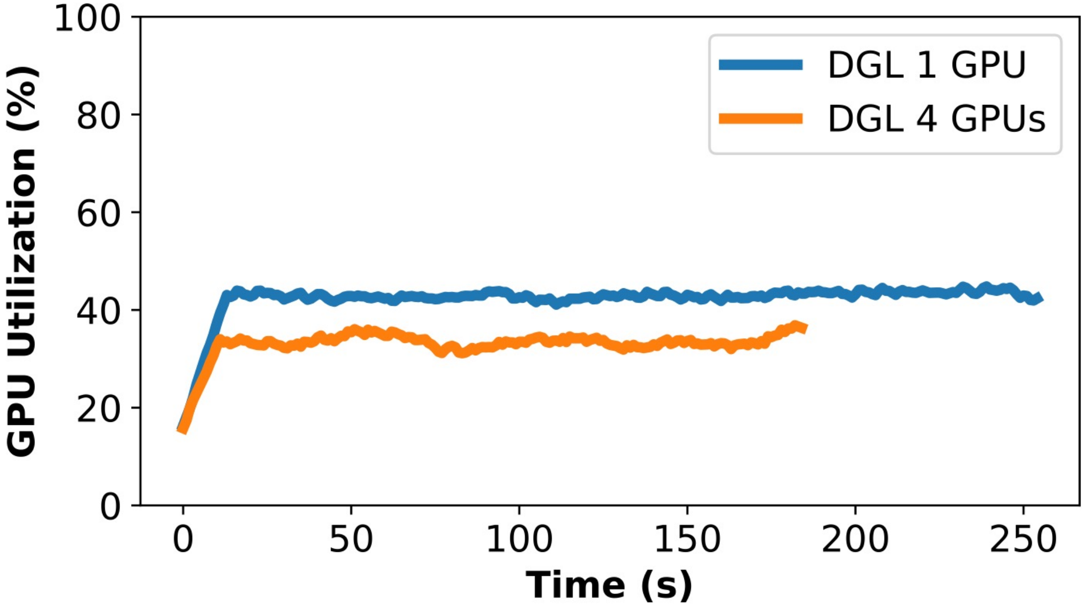
Embedding tables at Meta

3 Billion vertices, d = 400

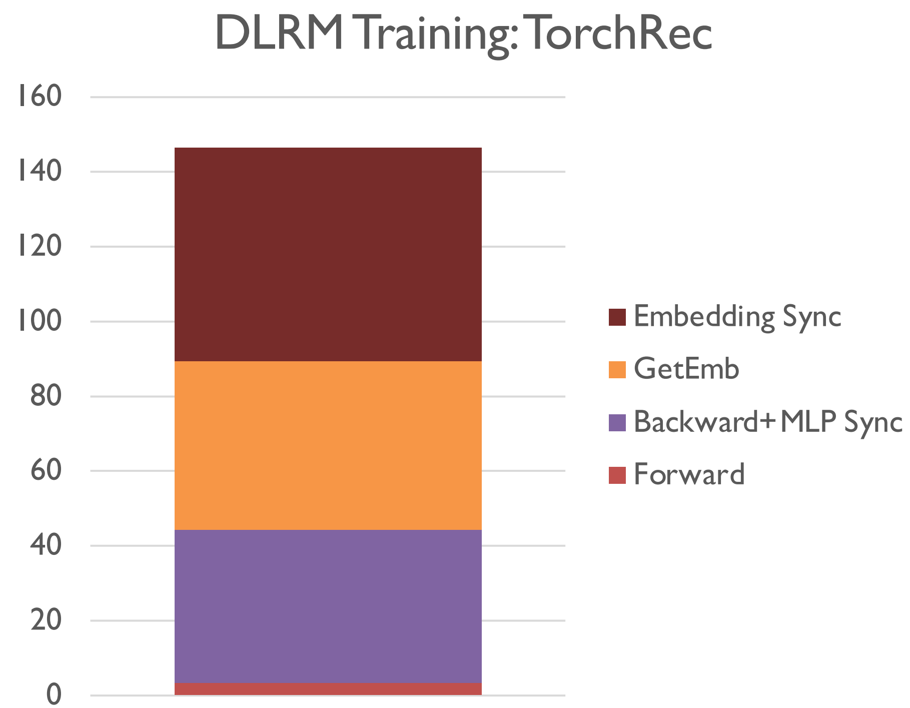
Model size = 3 billion * 400 * 4 = 4.8 TB!



CHALLENGE: DATA MOVEMENT



One epoch of GraphSage on Papers100M



One iteration of DLRM on Criteo dataset

Data movement overheads → low GPU util

MARIUS

I/O efficient system for learning on large graphs

Key Ideas

- Pipelined training
- New disk-based graph ordering
- Faster with 1 GPU than multi-GPU baselines

Learning Massive Graph Embeddings on a Single Machine
USENIX OSDI 2021

MariusGNN: Resource-Efficient Out-of-Core Training of Graph Neural Networks, ACM Eurosys 2023

BAGPIPE

System for training recommendation models

Key Ideas

- Lookahead-based caching
- Synchronous consistency guarantee
- Reduce communication by 60-70%

BagPipe: Accelerating Deep Recommendation Model Training, ACM SOSP 2023

MARIUS: TRAINING LOOP

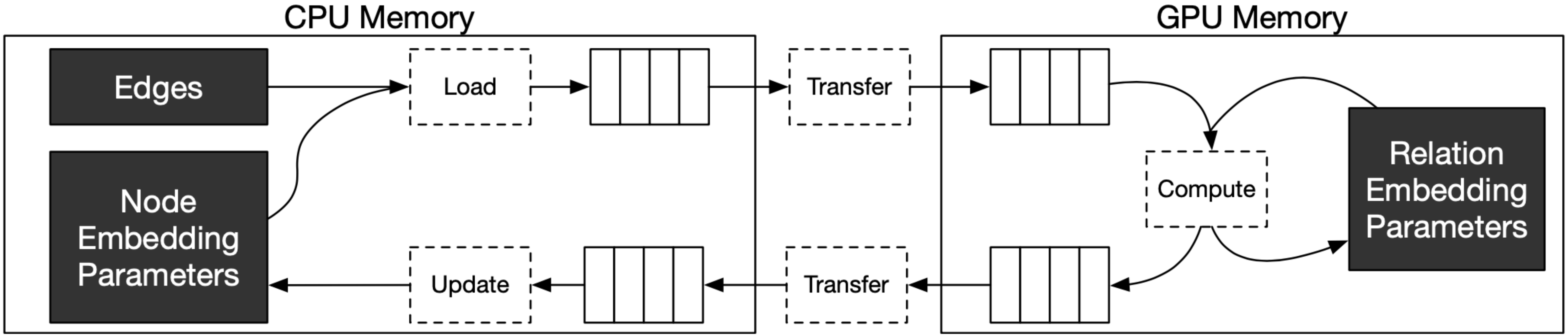
Sample vertices and neighbors

Access embeddings for each vertex

SGD/AdaGrad optimizer

```
for i in range(num_batches)
    B = getBatchEdges(i)
    N = sampleNbrs(B)
    E = getEmbeddingParams(B, N)
    G = computeGrad(E, B)
    updateEmbeddingParams(G)
```

MARIUS: PIPELINED TRAINING



OUT-OF-MEMORY GRAPH EMBEDDINGS

Prior work in out-of-memory graph algorithms

- GraphChi (OSDI 2012)
- Mosaic (Eurosys 2017)
- …

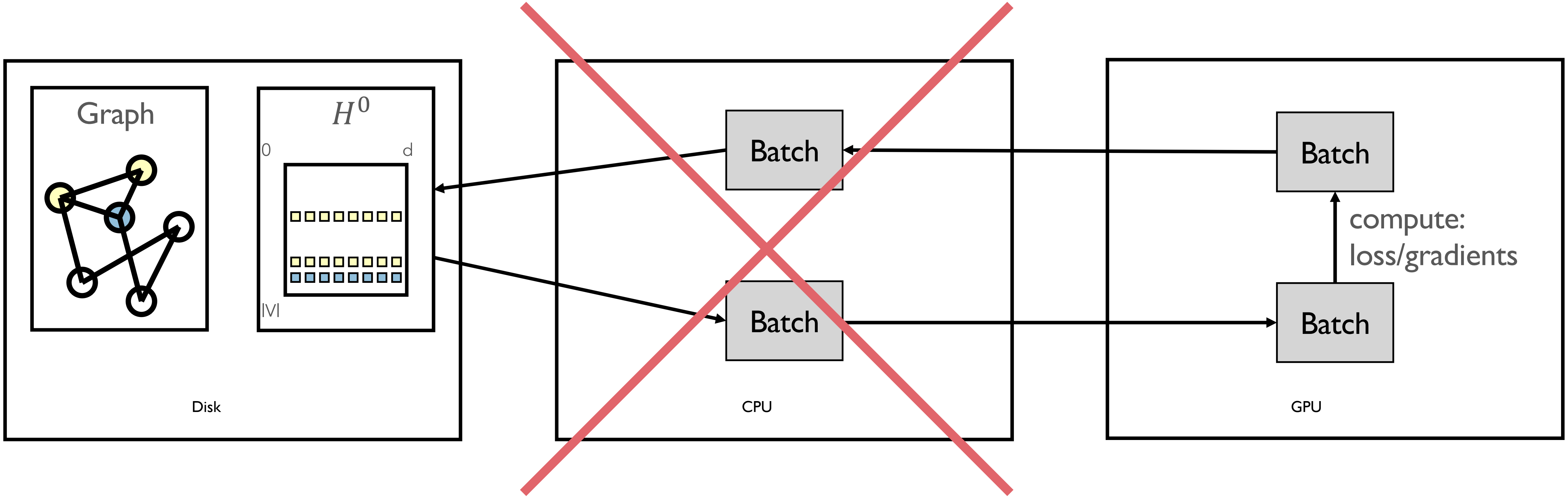
Graph Analytics (PageRank)

- Iterate over vertices, accessing state (scalar) of incoming edges

Graph Embeddings

- Iterate over edges, accessing vertex embeddings (vectors)

DISK-BASED TRAINING



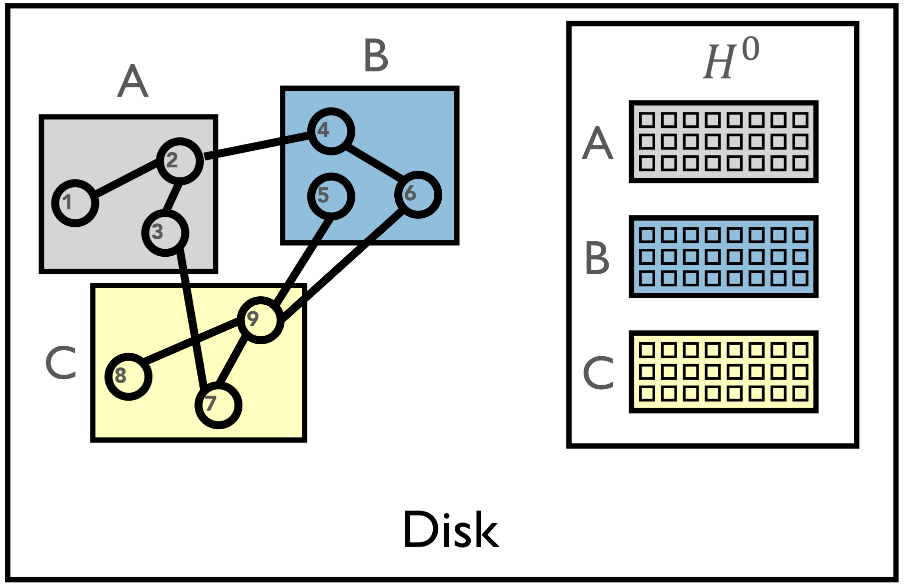
Prohibitively expensive to randomly access data on disk!

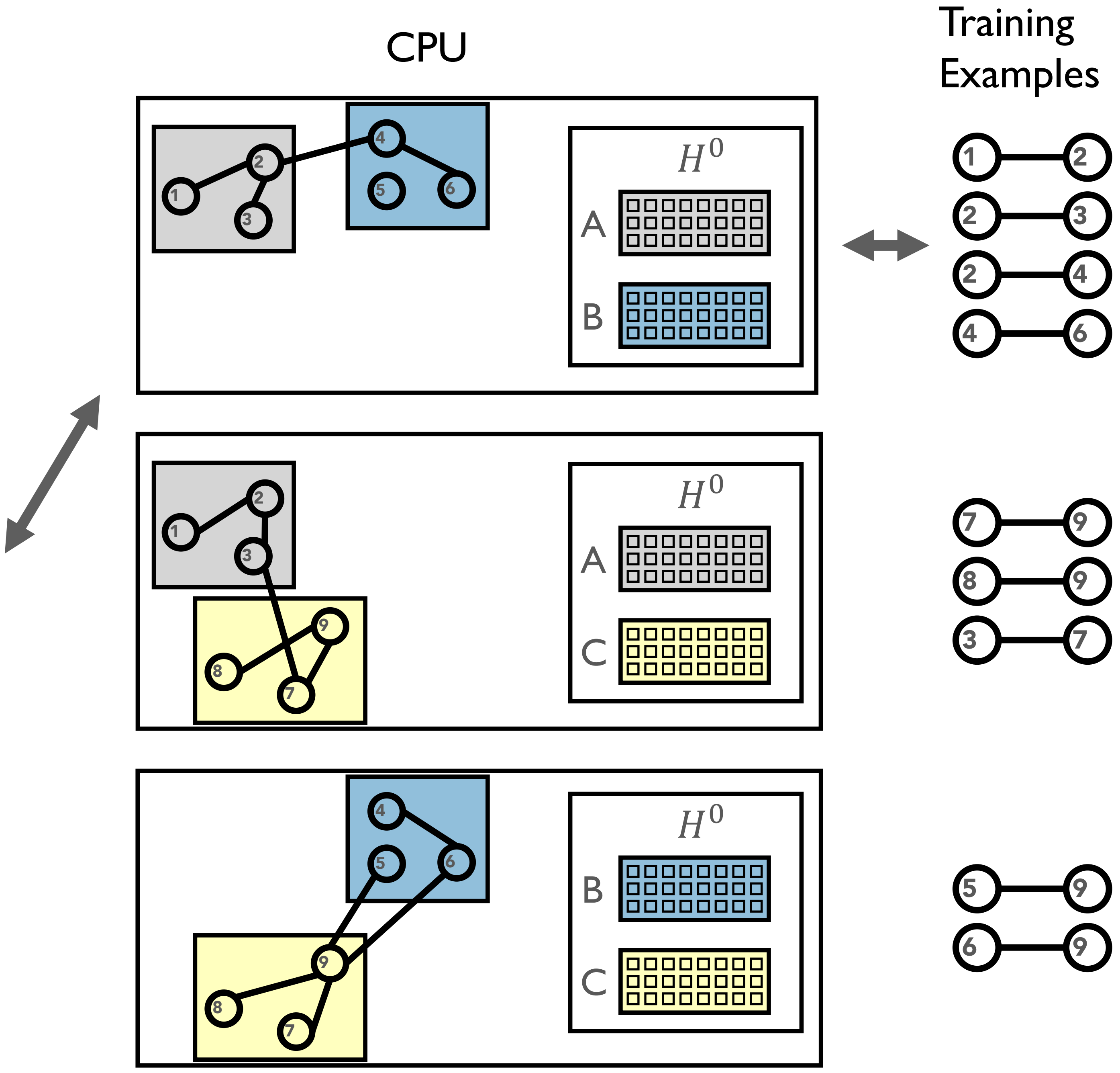
DISK-BASED TRAINING

Goal: Iterate over all examples (i.e., edges) on disk

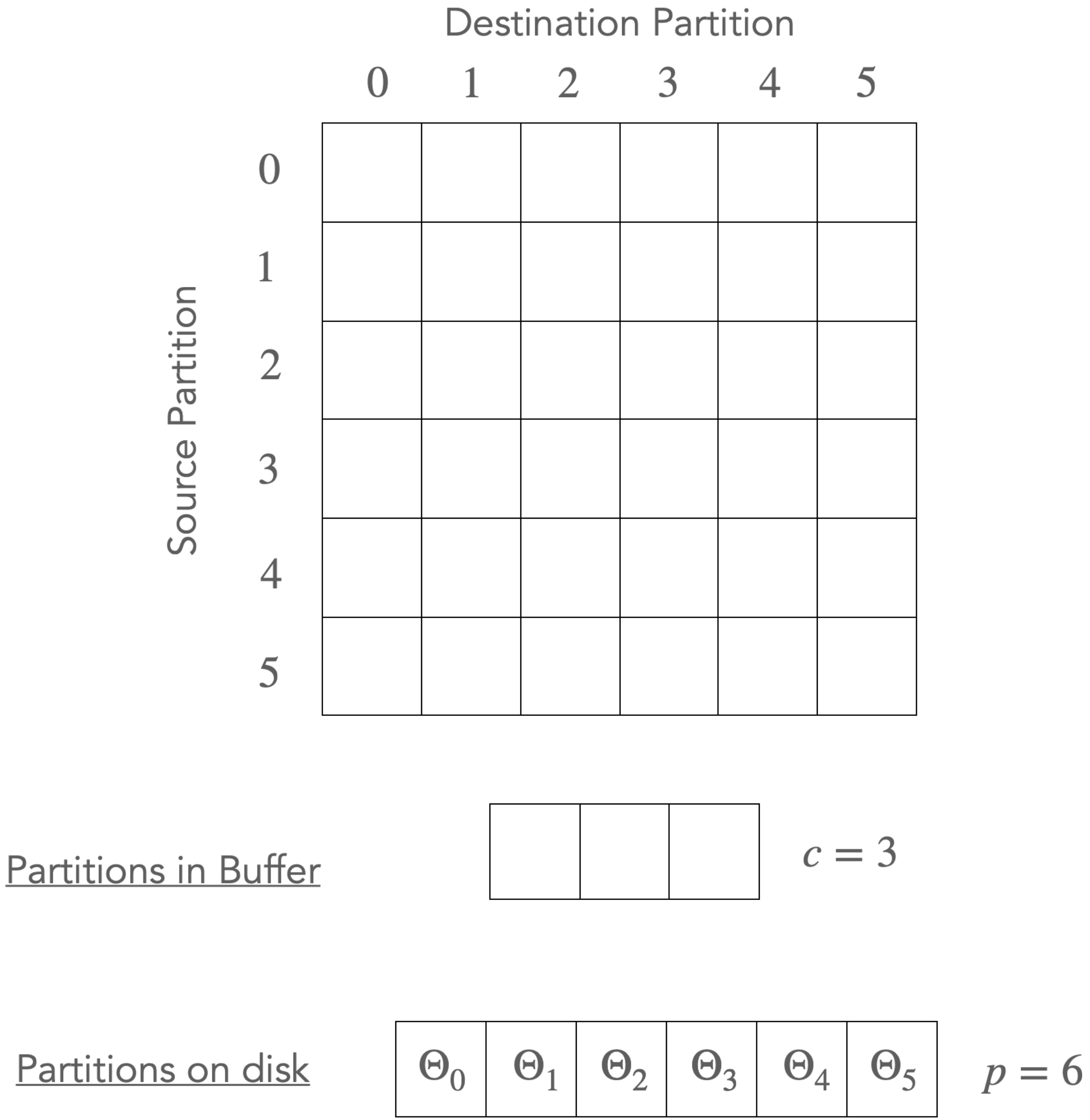
1. Randomly partition graph nodes

2. Load subsets into memory





PLANNING DATA ACCESS



Key idea: Maintain a *cache* of partitions in CPU memory

Questions

- Order of partition traversal?
- How to perform eviction?

EDGE BUCKET ORDERINGS

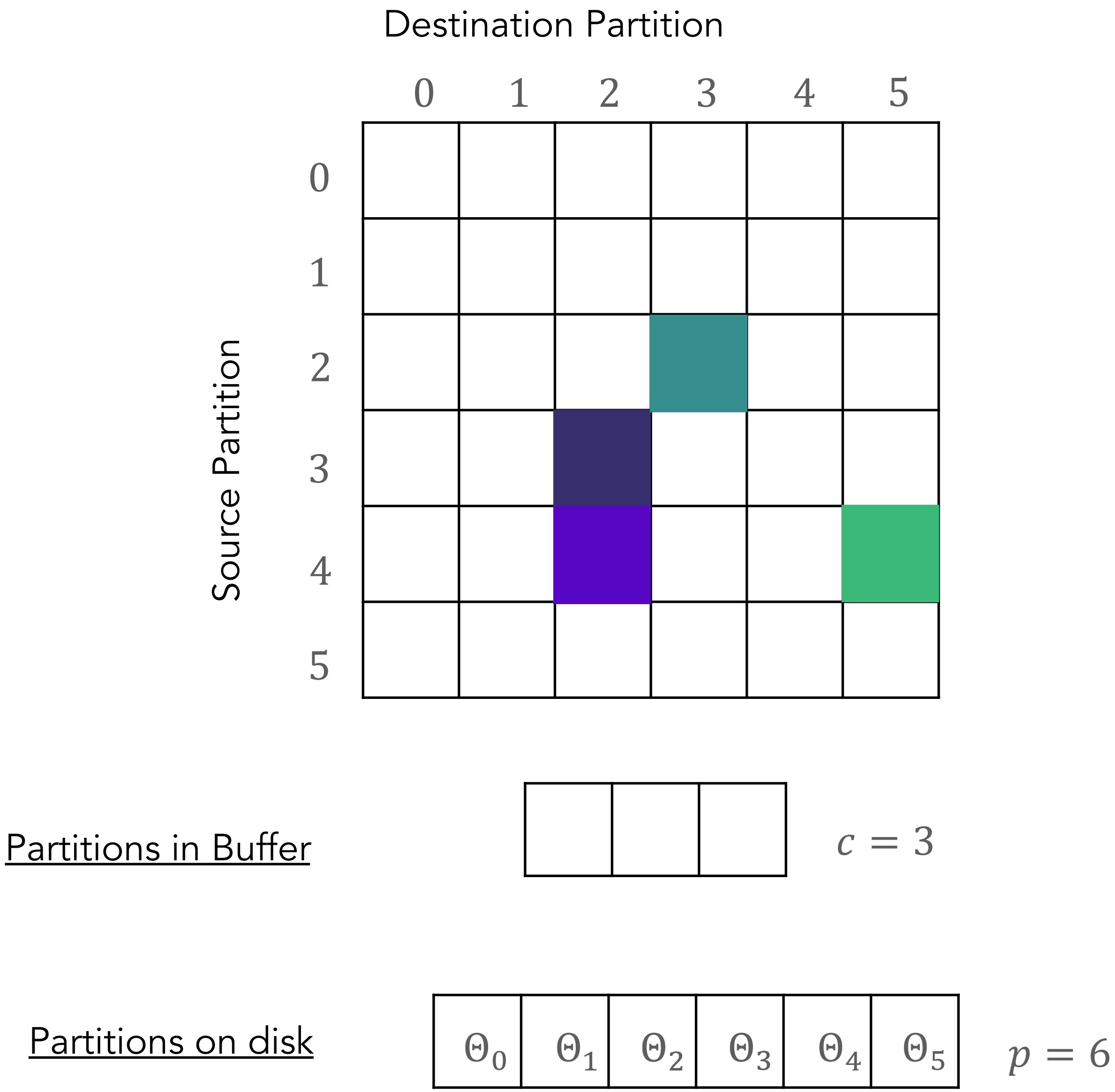
The order in which edge buckets are processed has an impact on IO

Example: After processing edge bucket (3, 2)

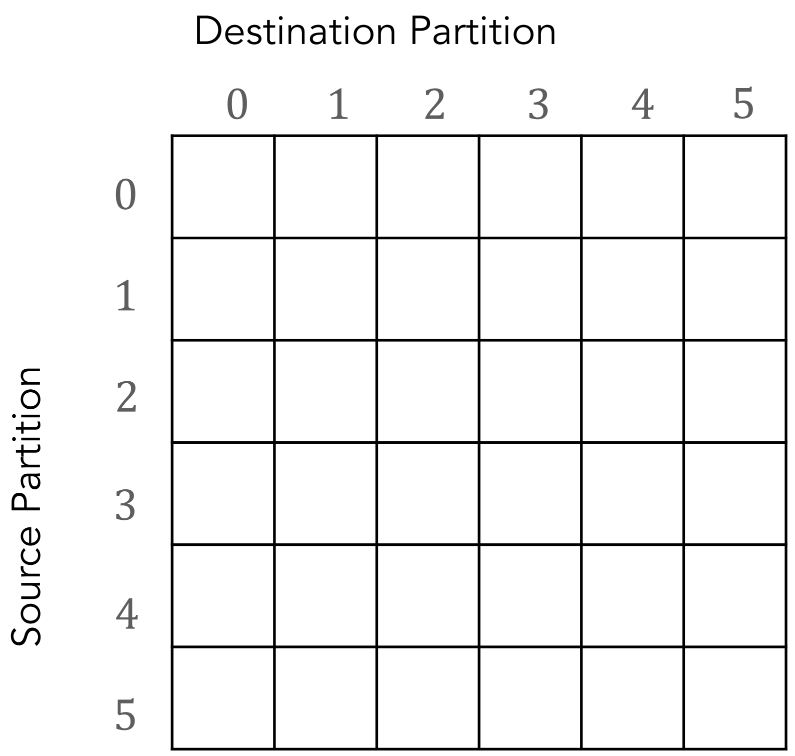
Processing (2, 3): Requires no extra swaps

Processing (2, 4): Requires one swap

Processing (4, 5): Requires two swaps



EDGE BUCKET ORDERINGS



A Lower Bound

Can never process more than 2c - 1 edge buckets per swap

$$\left\lceil \frac{p^2 - c^2}{2c - 1} \right\rceil = \left\lceil \frac{6^2 - 3^2}{2 * 3 - 1} \right\rceil = 6$$

Lower bound	6 swaps
Random Ordering	~23 swaps
Hilbert Curve Ordering	12 swaps
BETA Ordering	7 swaps

Partitions in Buffer

$c = 3$

Partitions on disk

Θ_0	Θ_1	Θ_2	Θ_3	Θ_4	Θ_5

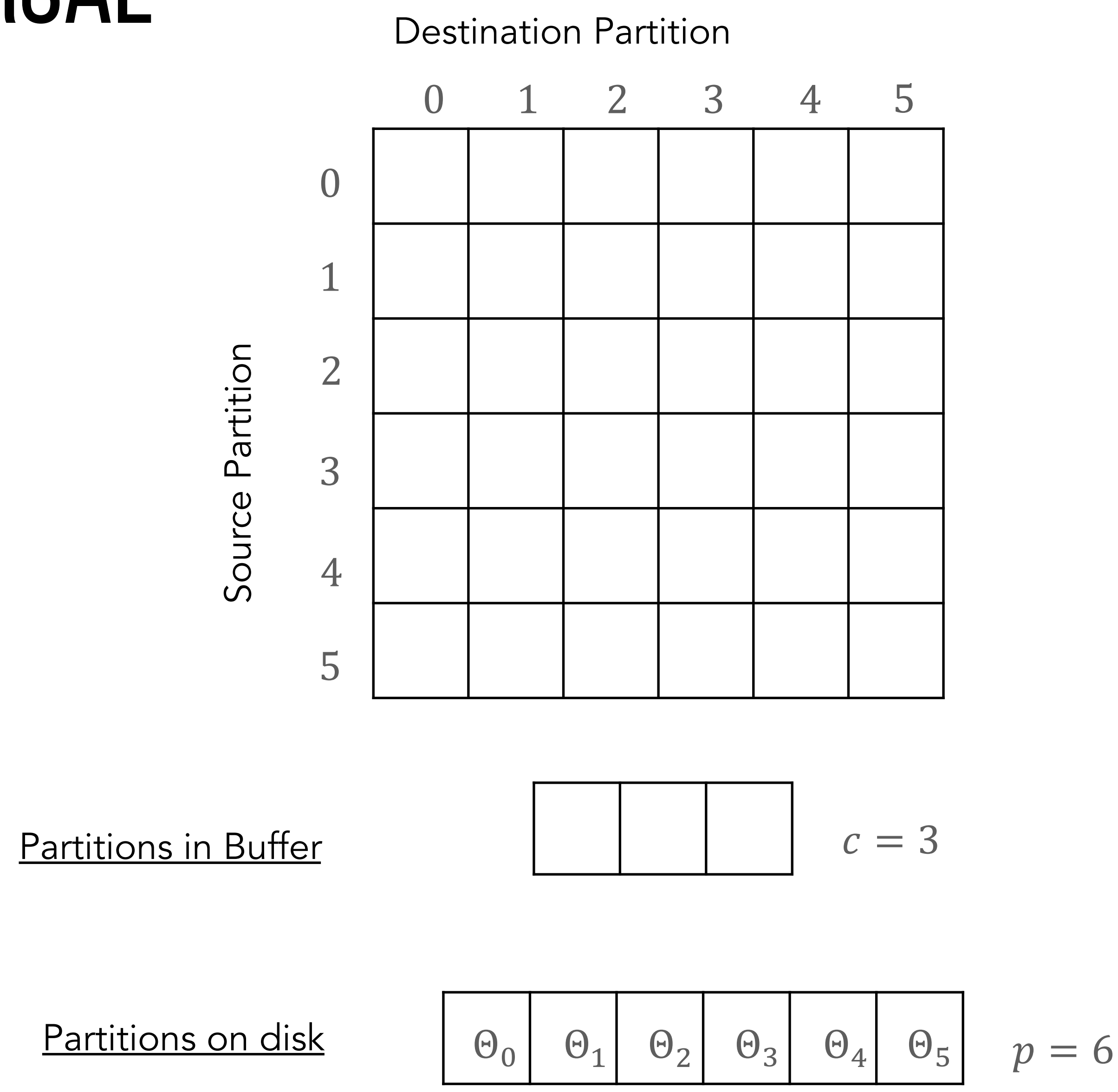
$p = 6$

BUFFER-AWARE EDGE TRAVERSAL ALGORITHM (BETA)

1. Randomly initialize buffer

2. Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets

3. Fix a new c - 1 partitions and repeat until all edge buckets have been processed



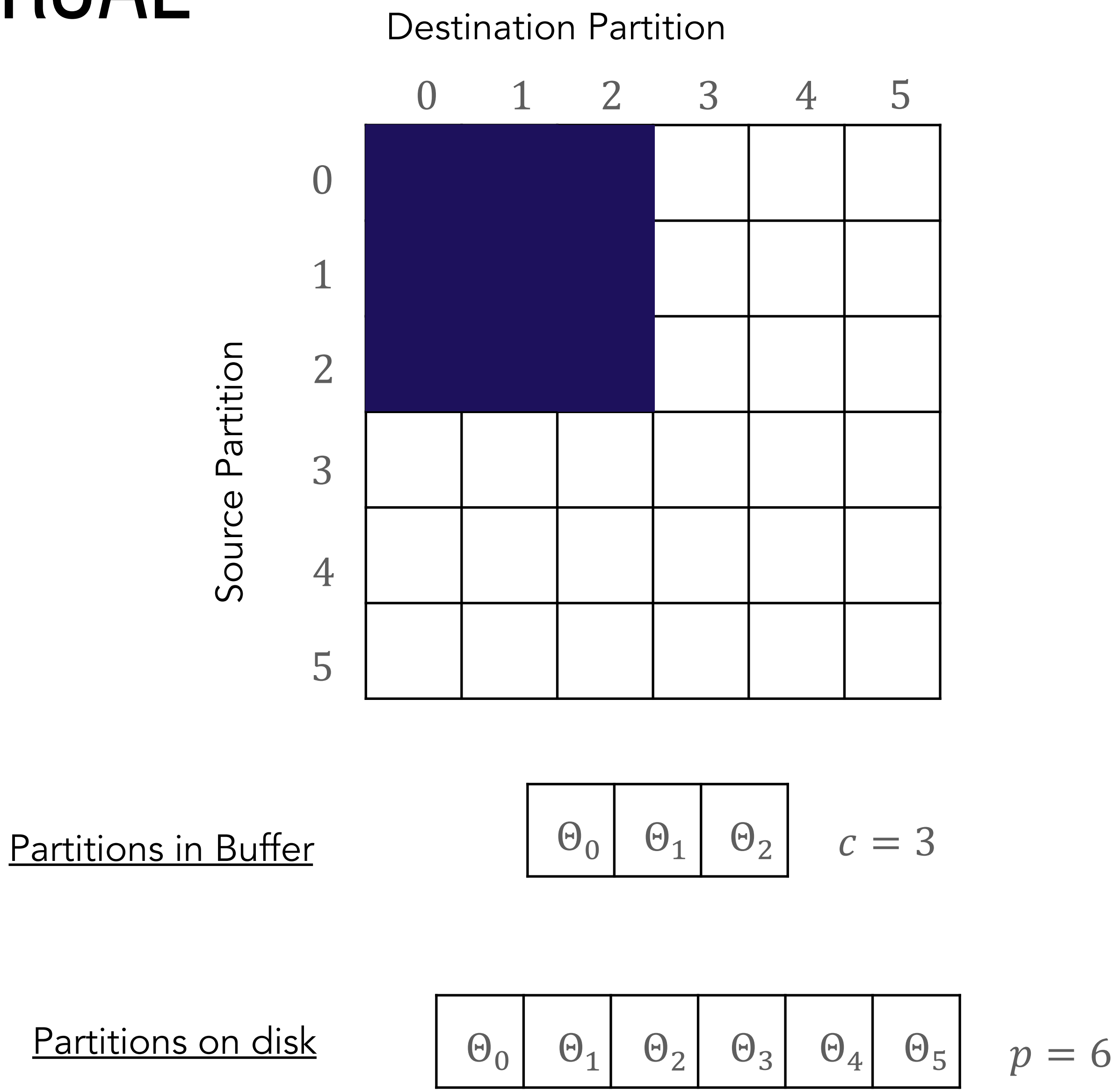
BUFFER-AWARE EDGE TRAVERSAL ALGORITHM (BETA)

1. Randomly initialize buffer

2. **Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets**

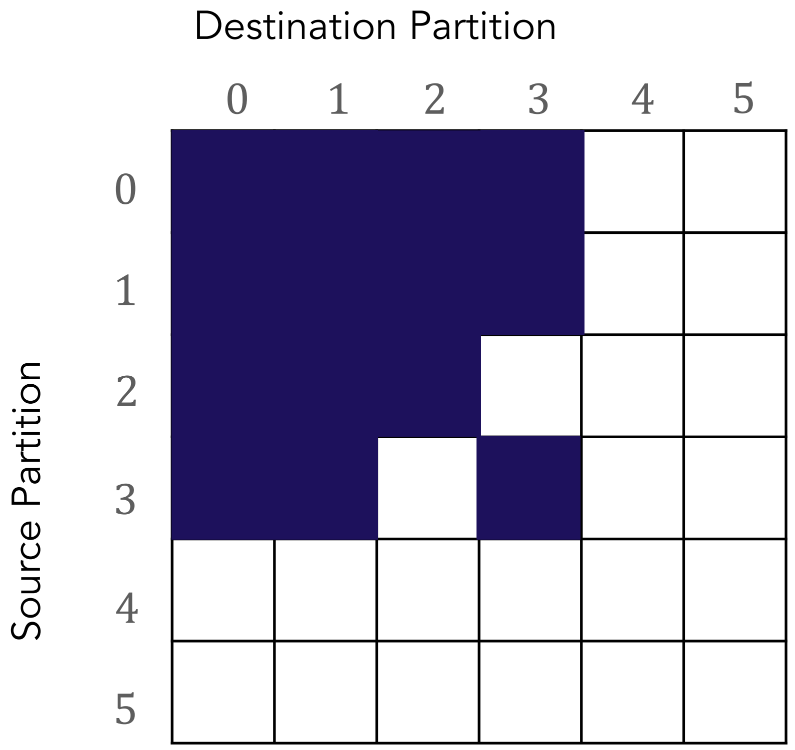
3. Fix a new c - 1 partitions and repeat until all edge buckets have been processed

0 swaps*



* Not counting initialized buffer, as with the previous orderings

Buffer-Aware Edge Traversal Algorithm (BETA)



1. Randomly initialize buffer
2. **Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets**
3. Fix a new c - 1 partitions and repeat until all edge buckets have been processed

<u>Partitions in Buffer</u>

Θ_0	Θ_1	Θ_2

$c = 3$

<u>Partitions on disk</u>

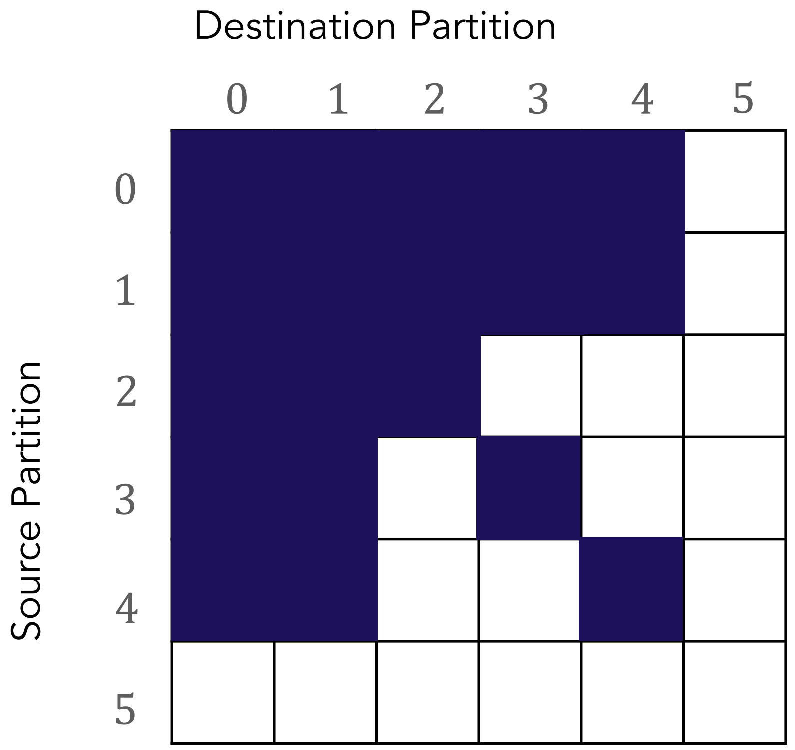
Θ_0	Θ_1	Θ_2	Θ_3	Θ_4	Θ_5

$p = 6$

0 swaps*

* Not counting initialized buffer, as with the previous orderings

BUFFER-AWARE EDGE TRAVERSAL ALGORITHM (BETA)



1. Randomly initialize buffer
2. **Use the last spot in the buffer to cycle through the rest of the partitions, processing their corresponding edge buckets**
3. Fix a new c - 1 partitions and repeat until all edge buckets have been processed

2 swaps

<u>Partitions in Buffer</u>

Θ_0	Θ_1	Θ_3

$c = 3$

<u>Partitions on disk</u>

Θ_0	Θ_1	Θ_2	Θ_3	Θ_4	Θ_5

$p = 6$

MARIUS: GPU UTILIZATION

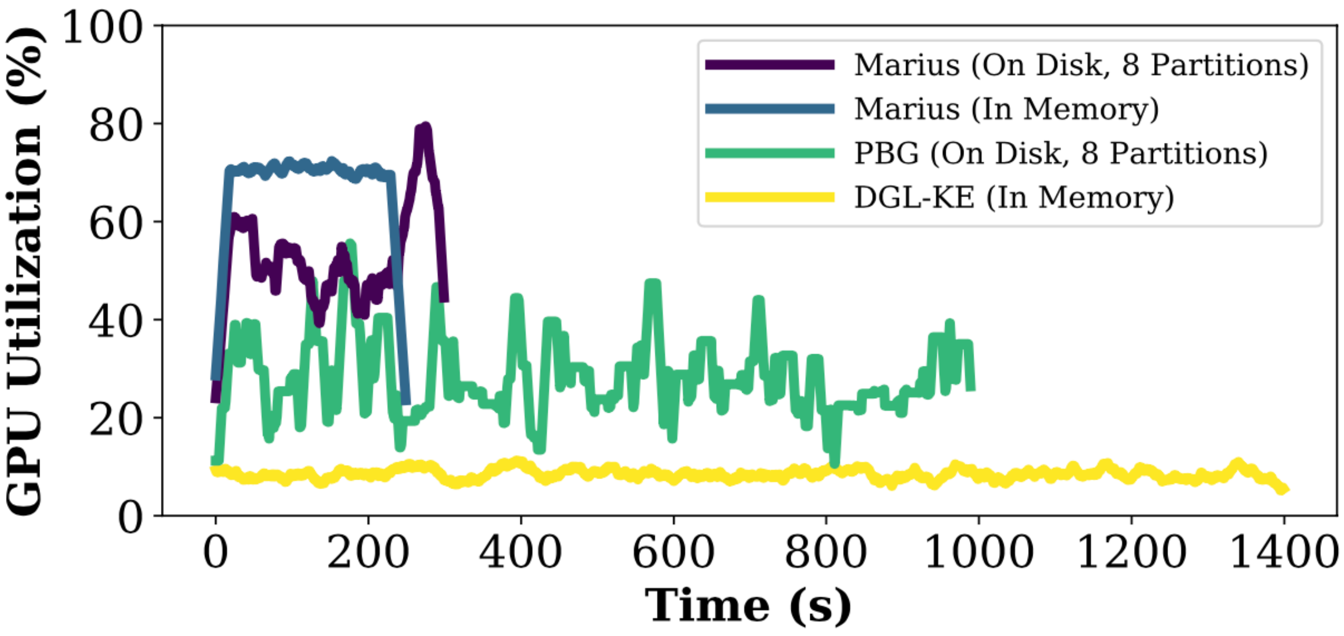
Freebase-86m

(338M vertices, 86.1M edges)

Single AWS p3.2xlarge

(one V100 GPU, 61GB DRAM)

3.8x speedup



One epoch on the Freebase86m knowledge graph
With d=50 embedding size

ACCURACY OF DISK-BASED TRAINING?

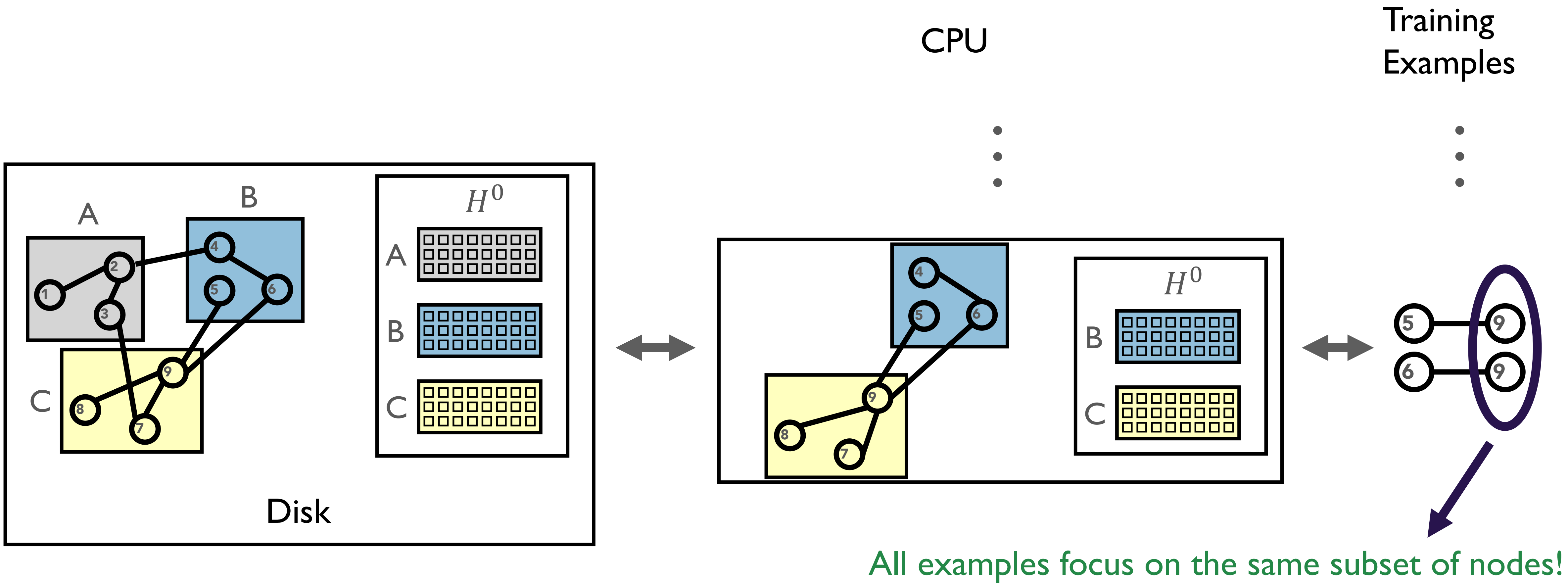
Problem: disk-based training can produce **low quality** GNN models

Graph	GNN Model	Mem Accuracy	Disk Accuracy
FB15k-237	GraphSage	0.2825	0.2369
FB15k-237	GAT	0.2869	0.2076
Freebase86m	GraphSage	0.7342	0.6976
Freebase86m	GAT	0.7418	0.6860

Using BETA policy to minimize IO (partition swapping)

Reason: using only on the in-memory subgraph for generating examples and neighborhoods leads to biased training

DISK-BASED TRAINING



Problem: correlated training examples (reduced mini batch randomness) → bad for SGD

DISK-BASED TRAINING: NO FREE LUNCH?

Challenge: develop policies that lead to **fast training** and yield **high accuracy** models

fast training

↓

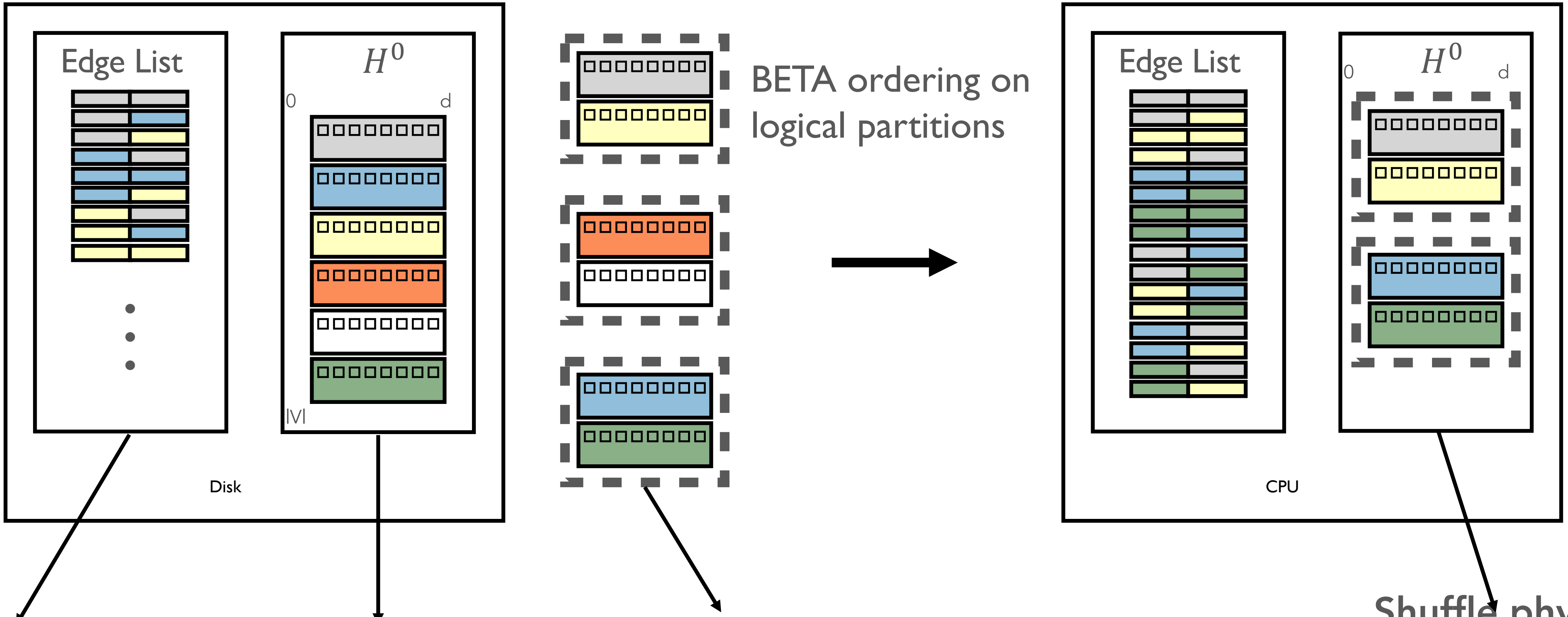
sequential access to disk data

high accuracy

↓

random access to graph data

CORRELATION MINIMIZING EDGE TRAVERSAL (COMET)



p^2 edge buckets

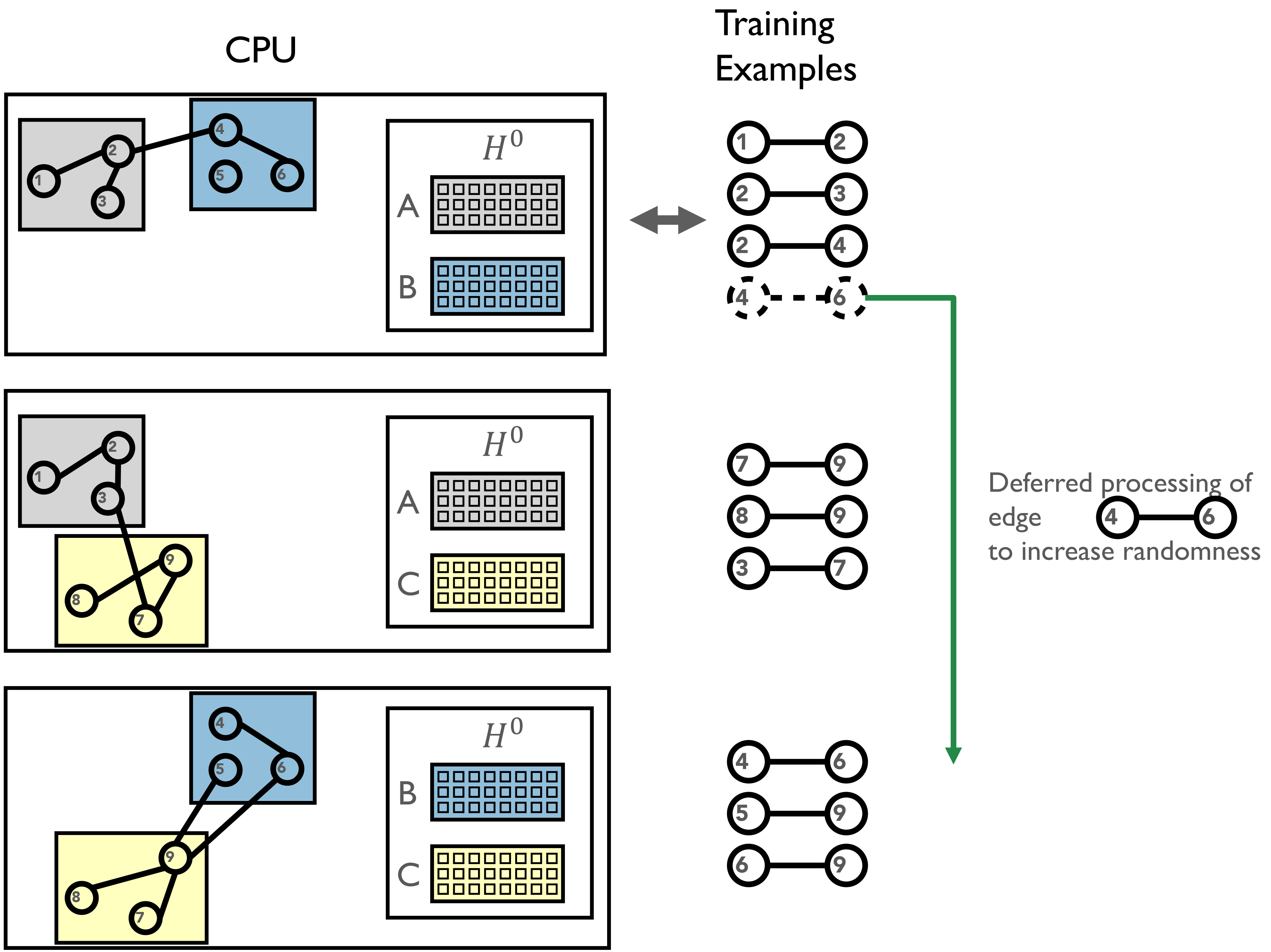
p physical partitions

1. virtually group into l logical partitions

Shuffle physical partitions in a logical partition

COMET

2. Randomize training examples used to create mini batches



DISK-BASED TRAINING WITH COMET

COMET: flexible two level partitioning and randomized training examples

Improves disk-based accuracy!

Graph	GNN Model	Mem Accuracy	Disk Accuracy (BETA)	Disk Accuracy (COMET)
FB15k-237	GraphSage	0.2825	0.2369	0.2736
FB15k-237	GAT	0.2869	0.2076	0.2341
Freebase86m	GraphSage	0.7342	0.6976	0.7123
Freebase86m	GAT	0.7418	0.6860	0.7053

MARIUS EVALUATION

Node Classification: 3-Layer GraphSage on OGB-Papers100M

System	GPUs	Epoch (min)	Accuracy	Cost ($/epoch)
PyG	4	8.01	66.93	1.63
DGL	4	3.07	66.98	0.63

Mixed CPU-GPU training - MariusGNN (Mem): reaches similar accuracy ~4x faster than multi-GPU DGL

→ 4x cheaper training cost

Disk-based training - MariusGNN (Disk): 4x cheaper machine yet also ~4x faster than baselines

→ **16x cheaper training cost**

MARIUS: SUMMARY

Open source, Python API

Deployed at Apple for training models on large knowledge graphs

https://github.com/marius-team/marius

Saga: A Platform for Continuous Construction and Serving of Knowledge at Scale, ACM SIGMOD 2022

MARIUS

I/O efficient system for learning on large graphs

Key Ideas

- Pipelined training
- New disk-based graph ordering
- Faster with 1 GPU than 8-GPU baselines

Learning Massive Graph Embeddings on a Single Machine, USENIX OSDI 2021

MariusGNN: Resource-Efficient Out-of-Core Training of Graph Neural Networks, ACM Eurosys 2023

BAGPIPE

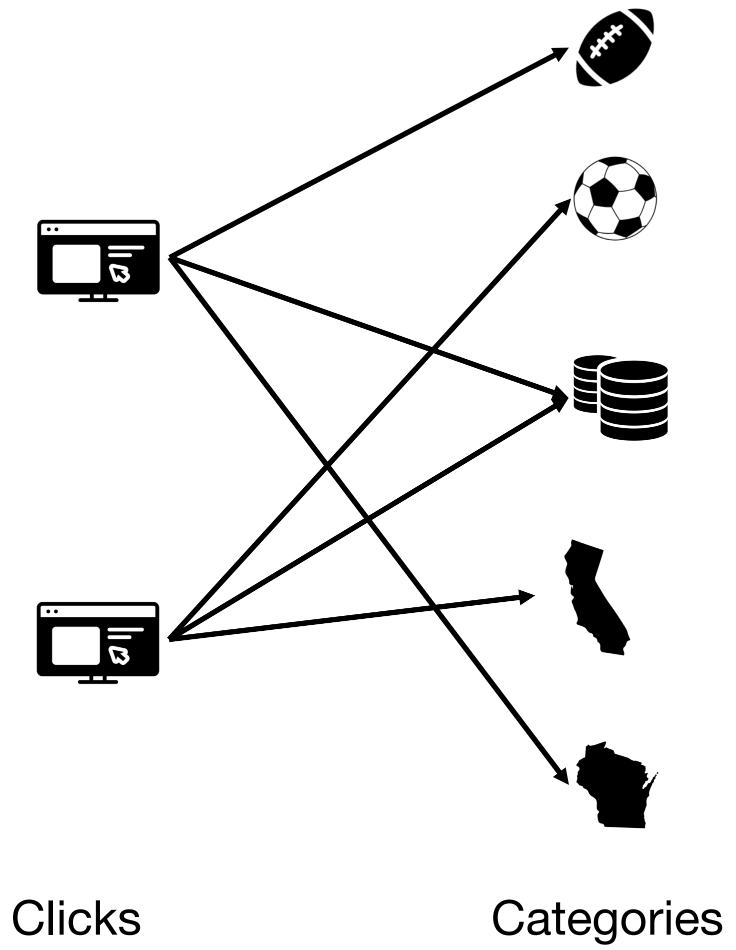
System for training recommendation models

Key Ideas

- Lookahead-based caching
- Distributed, disaggregated execution
- Reduce communication by 60-70%

BagPipe: Accelerating Deep Recommendation Model Training, ACM SOSP 2023

WHAT IS DIFFERENT?



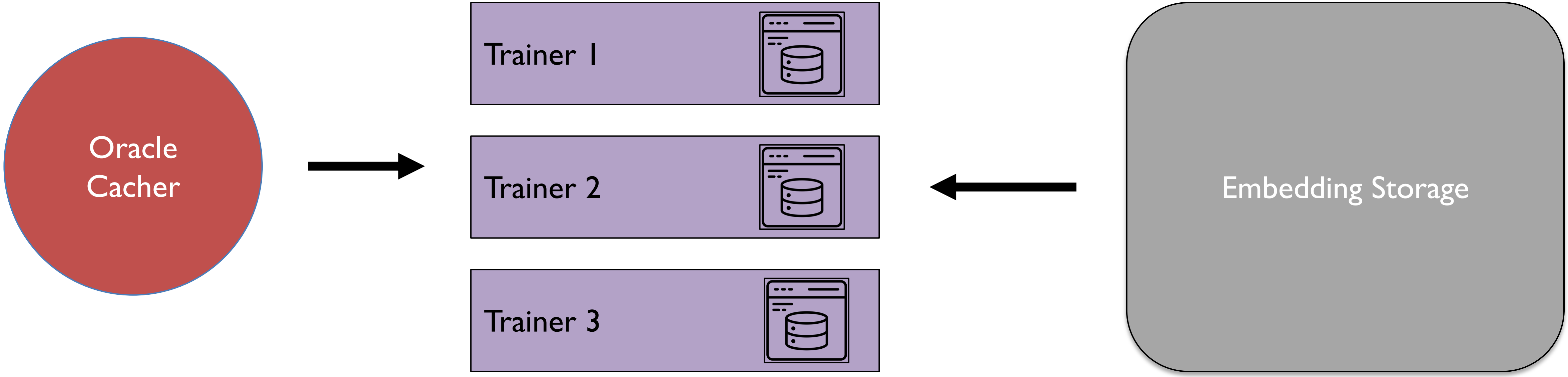
- Bi-partite graph with edges only between events and categories
- Only requires one-hop neighborhood

Can "lookahead" to determine embedding access pattern!

→

1. Prefetch embeddings before batch
2. Cache frequently used embeddings

BAGPIPE DESIGN



Lookahead; determine what to prefetch, cache

Prefetch embeddings and store in local cache

ORACLE CACHER ALGORITHM

Example Lookahead = 2

Batch 1: Prefetch embedding 3 and 9
Also cache embedding 3 (as it is used in Batch 2)

Batch 2: Prefetch embedding 4
Embedding 3 is already in cache!
Update its time-to-live (TTL) to 3 (used in Batch 3)

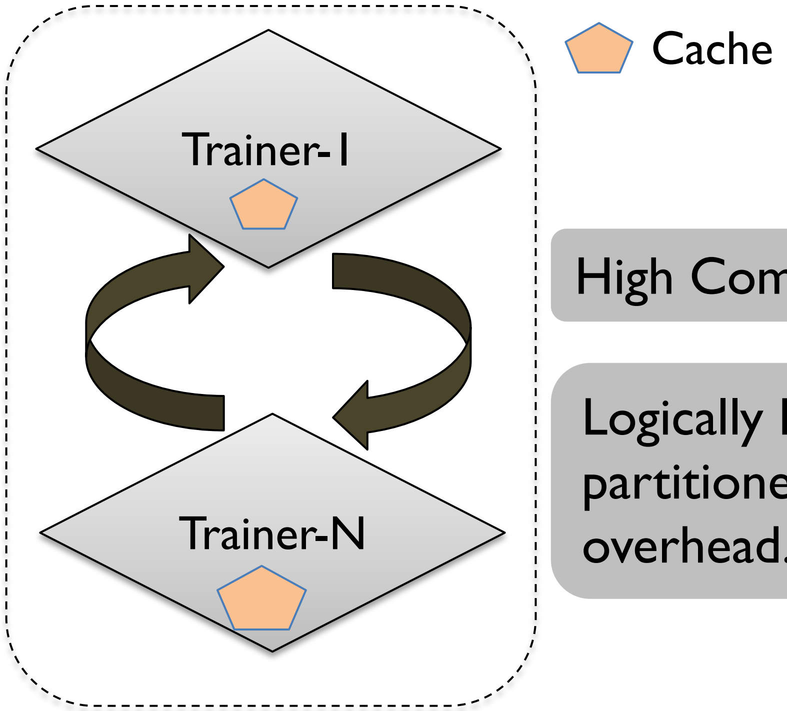
…

Guarantee: All workers can always access the latest value of the embeddings.

Maintains synchronous training semantics, no accuracy loss!

DISTRIBUTED CACHE SYNC

At the end of each iteration, need to synchronize the updated embeddings

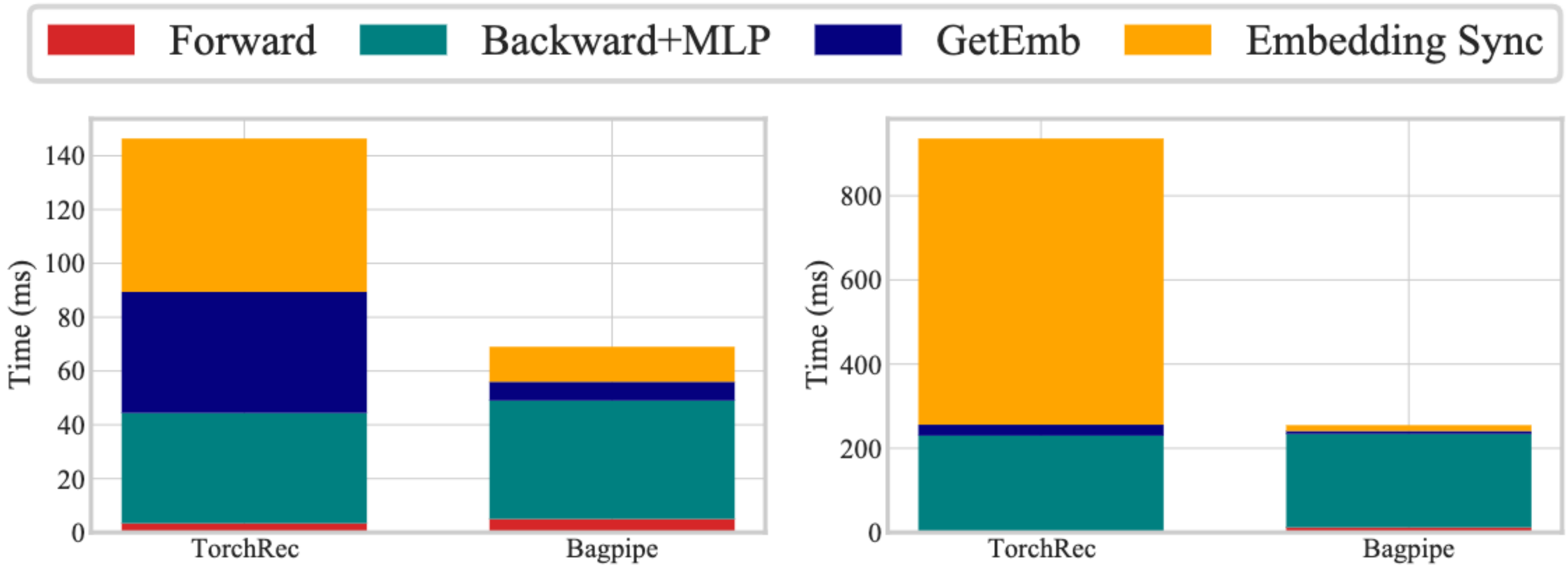


High Communication overhead

Logically Replicated and Physically partitioned caches to minimize overhead.

More Details in the paper!

BAGPIPE: EVALUATION



(a) DLRM

(b) DeepFM

Time per iteration of DLRM model using 8 p3.2xlarge EC2 instances (1 V100 each node).

Reduce communication overhead to 10% from 75%!

Data access

Data access for training on large graph structured data

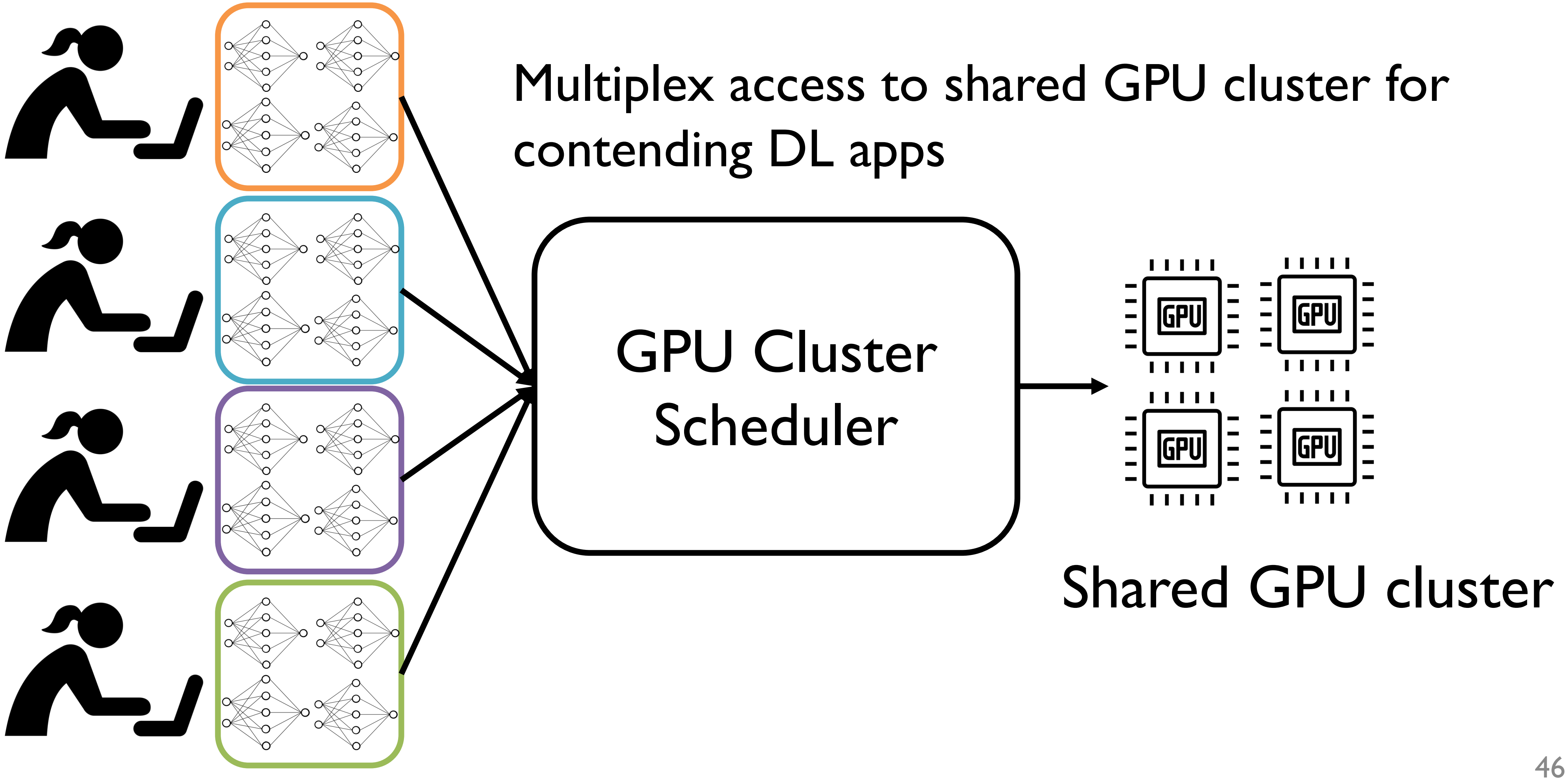
Synchronization

Optimize communication during distributed training

Cluster Scheduling

Policies and mechanisms for scheduling on shared clusters

GPU CLUSTER SCHEDULING



PHILLY STUDY DETAILS

Trace details

75-day period from Oct. 2017 to Dec. 2017

Total of 96,260 jobs over 14 virtual clusters

Logs details

Scheduler logs: job arrival time, num GPUs, finish status

stdout, stderr logs from ML frameworks

Per-minute statistics from Ganglia

Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads – USENIX ATC 2019

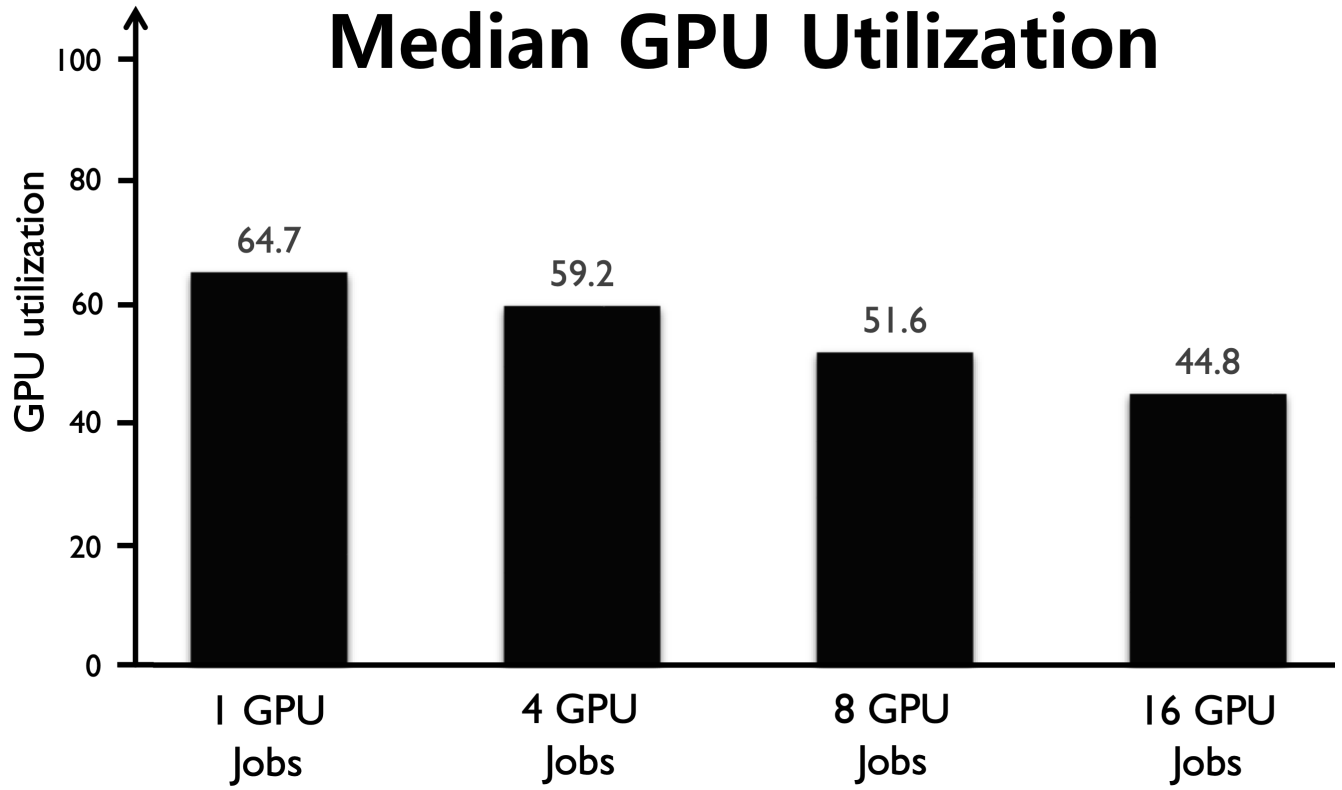
STUDY QUESTIONS

(1) What is effect of gang scheduling on queuing?

(2) What is impact of locality on GPU utilization?

(3) How frequent are failures during training ?

GPU UTILIZATION



How effectively are the GPUs utilized for DNN training?

Most GPUs are allocated but utilization is low!

Placement across servers decreases utilization

More details in our ATC 2019 paper!

DEEP LEARNING SCHEDULERS

Improve Cluster Utilization:

Gandiva (OSDI 2018), AntMan (OSDI 2020), HiveD (OSDI 2020)

Reduce Job-completion Time

Tiresias (NSDI 2019), Optimus (Eurosys 2019)

Optimize Goodput through Elasticity

SLAQ (SoCC 2017), Optimus (Eurosys 2019), Pollux (OSDI 2021), Sia (SOSP 2023)

Fair sharing across users:

Themis (NSDI 2020), $\text{Gandiva}_{\text{fair}}$ AlloX(Eurosys 2020), Gavel (OSDI 2020),

Shockwave (NSDI 23)

THEMIS: METRIC FOR FAIRNESS

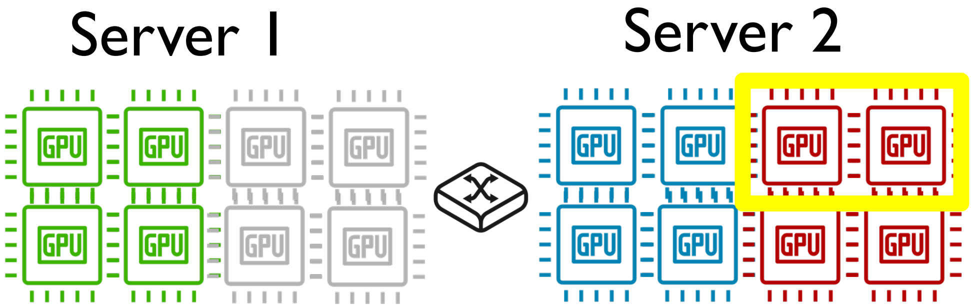
$\rho = T_{sh} / T_{id}$

- T_{sh}: finish-time of app in shared cluster
- T_{id}: finish-time of app in exclusive 1/N share of cluster
- N: Average contention during app lifetime

Sharing Incentive: for all apps, $\rho <= 1$

Used to evaluate fairness of many schedulers including Gavel, Pollux, Sia etc.

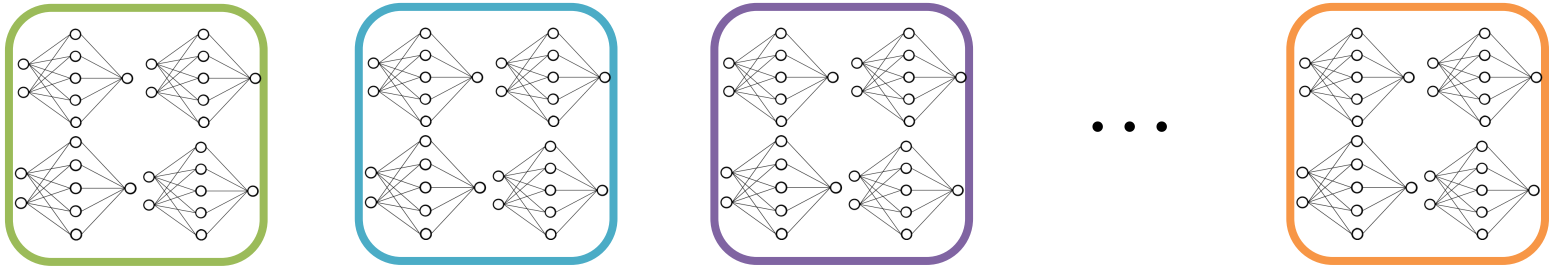
THEMIS: MECHANISM



Objective: min (max ρ)

Interface: Get ρ estimates from all jobs

Red GPUs become available



$$\underbrace{\rho_1 > \rho_2}_{1-f} > \underbrace{\rho_3 > \cdots > \rho_N}_{f}$$

1. Filter *1 – f* jobs with max ρ values
2. Allocate to one or more of *1 – f* jobs for *lease* duration using Partial Allocation Auctions

DYNAMIC ADAPTATION IN ML JOBS: GNS

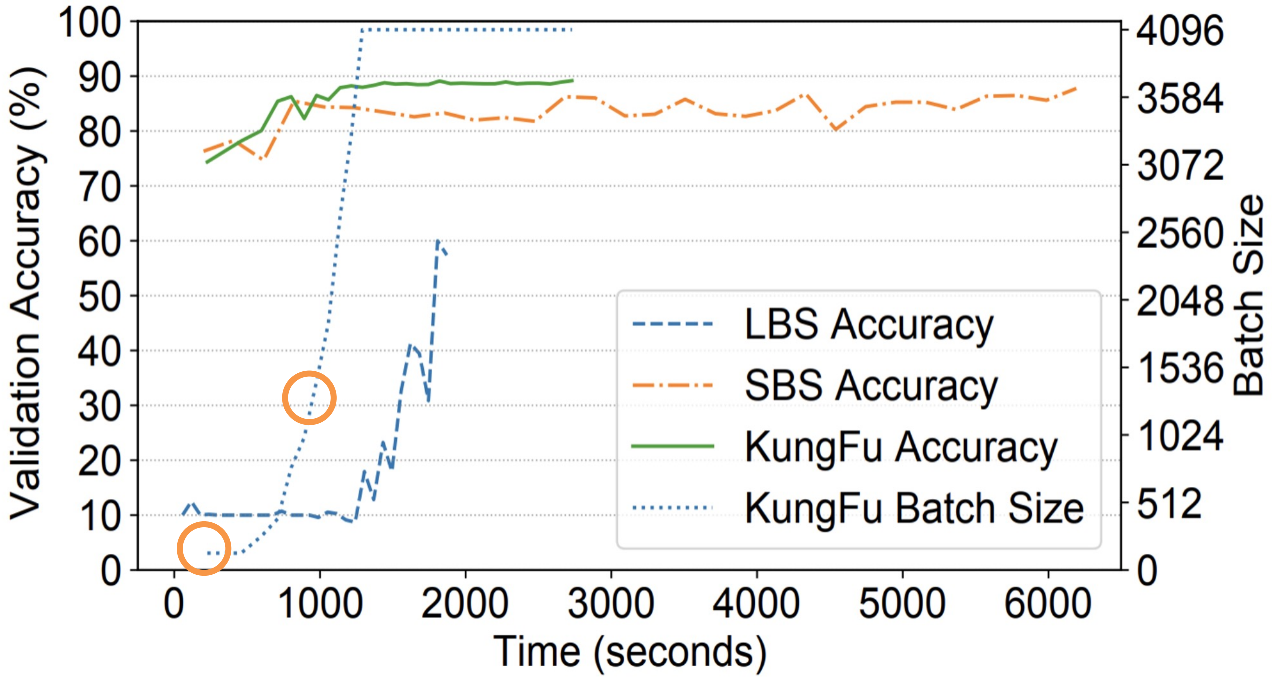
Gradient Noise Scaling (GNS)

Adaptively double batch size based on gradient noise

Small Batch Size (16),

Large Batch Size (4096)

Batch size 16 → 32 → … → 4096

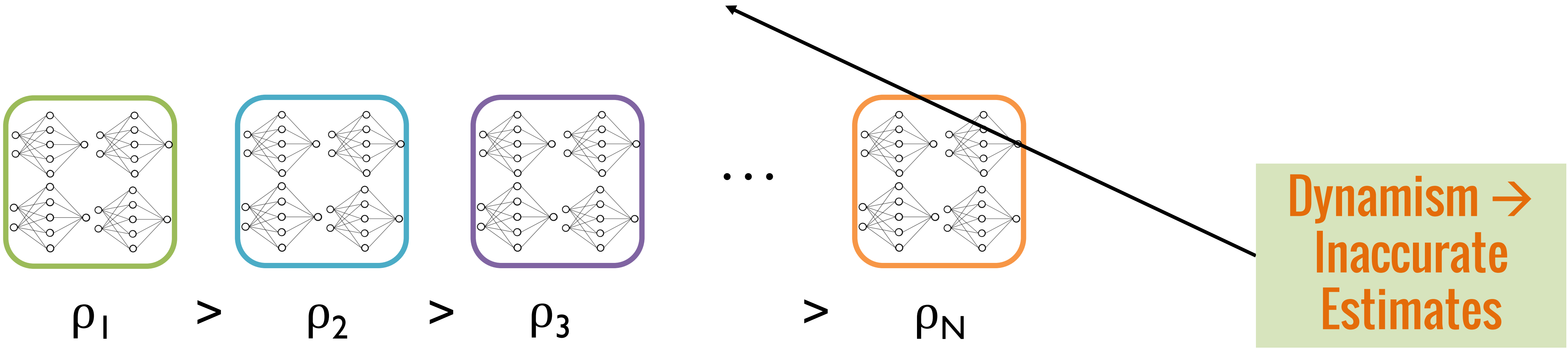


KungFu (OSDI 2020)

CHALLENGE: INACCURATE ESTIMATES

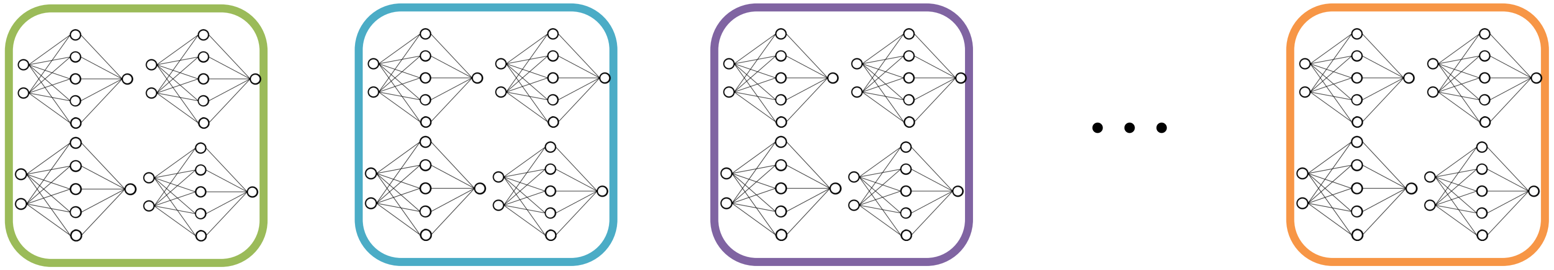
Themis Objective: min (max ρ)

Interface: Get finish time fairness (ρ) estimates from all apps



CHALLENGE: FILTERING ON PAST ALLOCATIONS

Themis objective – min (max ρ)



$$\underbrace{\rho_1 > \rho_2}_{1-f} > \underbrace{\rho_3 \quad > \quad \rho_N}_{f}$$

Filter only looks at past allocations

1. Filter $1 - f$ apps with worst ρ values
2. Allocate to one or more of $1 - f$ apps for *next round* using partial auctions

SHOCKWAVE: DYNAMIC MARKETS

Goal: Scheduling policy that accounts for the past and future utilities

Market theory: Provable guarantees for efficiency, fairness.

Static market: Every training job has a known, **time-invariant** utility *U(x)*

- Utility *U(x)*: map allocated GPU-time to training throughput for a job

Volatile Fisher Market (VFM) in Shockwave

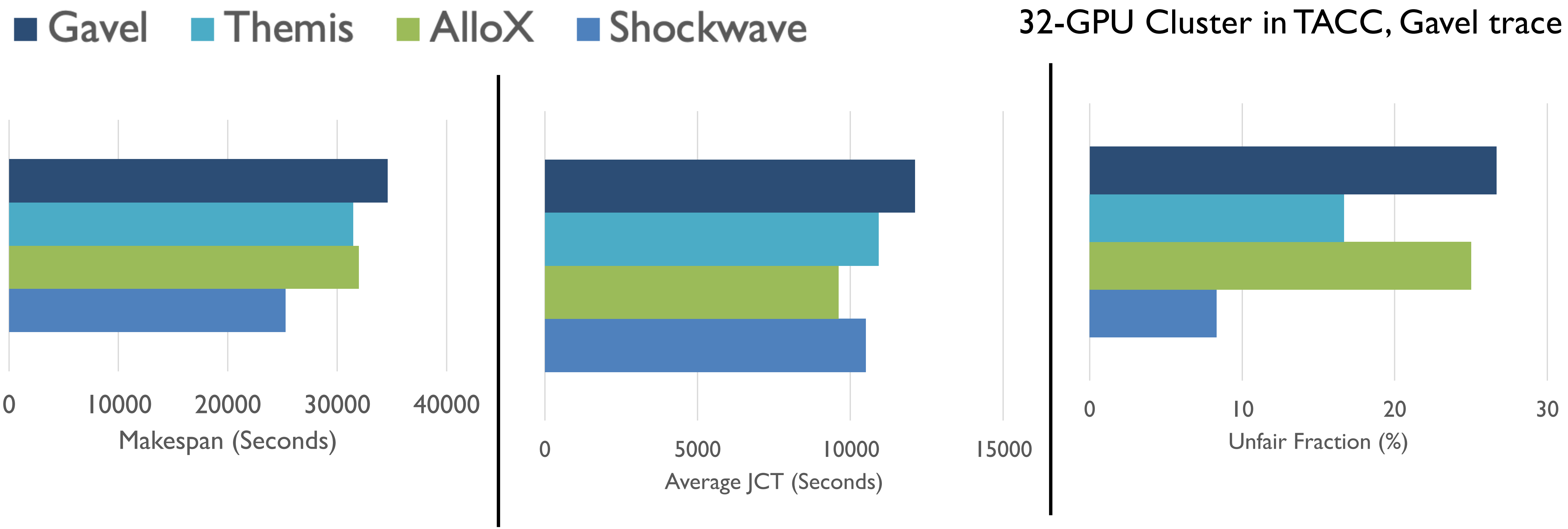
- Operate at discrete time intervals (rounds)
- Every training job has **a time-variant utility $U_t(x)$** for each round (t)
- Solve for allocation that leads to market equilibrium

END-TO-END COMPARISONS

■ Gavel ■ Themis ■ AlloX ■ Shockwave

32-GPU Cluster in TACC, Gavel trace

END-TO-END COMPARISONS



Reduces Makespan by ~1.3x, Unfair fraction by ~2x, maintains JCT

PUTTING IT ALL TOGETHER

Data access

Synchronization

Cluster Scheduling

BETA and COMET orderings for GNN training

Marius: https://github.com/marius-team/marius/

Lookahead algorithm for recommendation models

Bagpipe: https://github.com/uw-mad-dash/bagpipe

Market-theory based fair scheduling

Shockwave: https://github.com/uw-mad-dash/shockwave

THANK YOU!

Students: Jason Mohoney, Roger Waleffe, Saurabh Agarwal, Kshitij Mahajan, Arjun Singhvi, Pengfei Zheng, Rui Pan, Rutwik Jain, Prasoon Sinha, Brandon Tran, Hongyi Wang

Collaborators: Theo Rekatsinas, Aditya Akella, Amar Phanishayee, Matt Sinclair, Zhao Zhang

Shivaram Venkataraman
shivaram@cs.wisc.edu