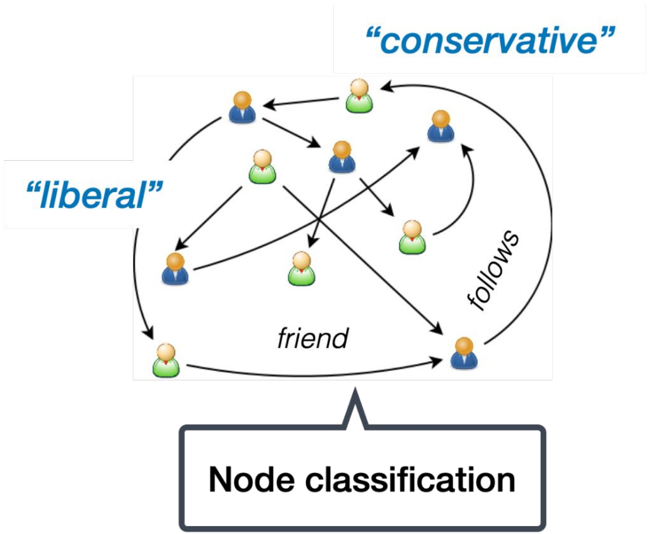
RingSampler: GNN Sampling on Large-Scale Graphs with io_uring

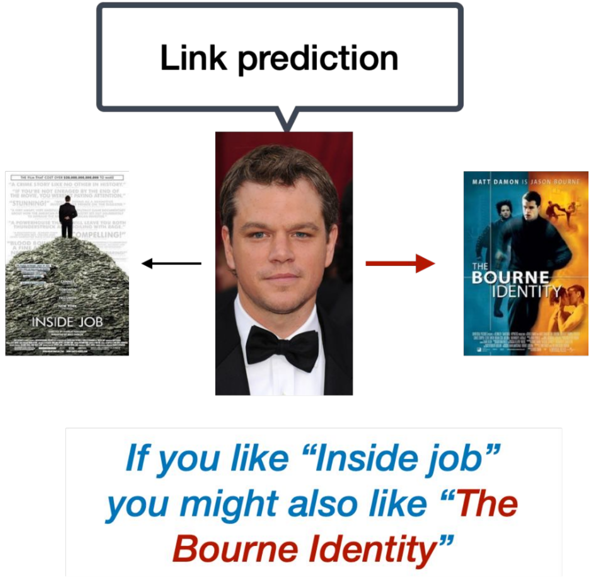
Qixuan (Talia) Chen, Yuhang Song, Melissa Martinez, Vasiliki Kalavri

Boston University

Graph Neural Networks (GNNs)

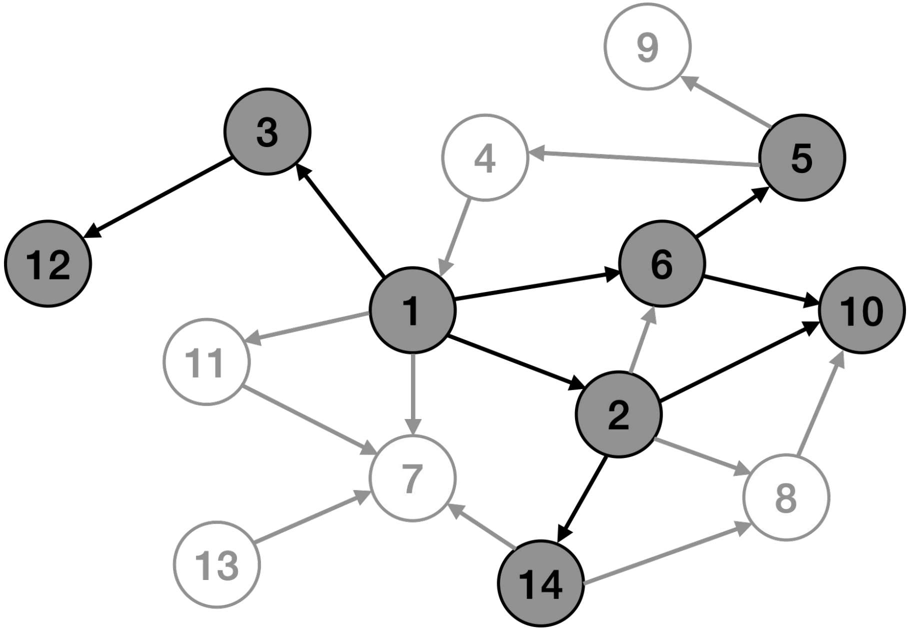
Graph Neural Networks (GNNs) are powerful models in graph learning that capture relationships and dependencies between entities in a network.





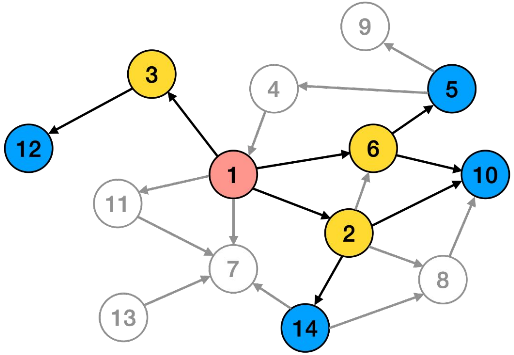
GNN Sampling

Training on the full graph can be computationally intensive and memory-heavy, so GNNs often sample a subset of the original graph that retains its key features and structure.

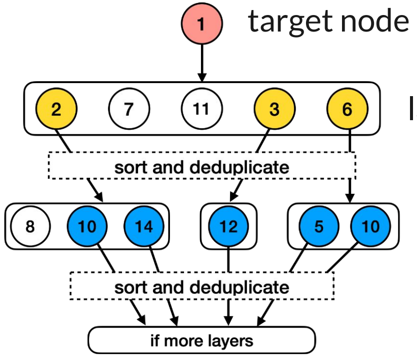


GraphSAGE Sampling

For each **target node** (i.e., a node to be trained), GraphSAGE samples a fixed number of neighbors (**fanout**) at each layer to construct the computation graph.



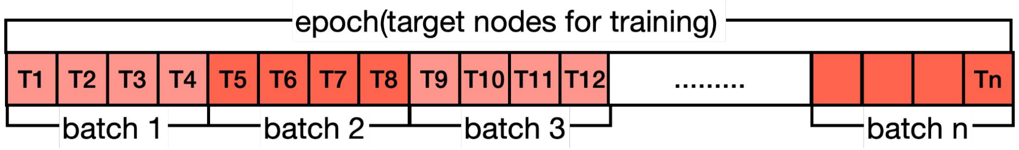
a. sample input graph



b. node-wise sampling workflow

GraphSAGE 2 layer sampling

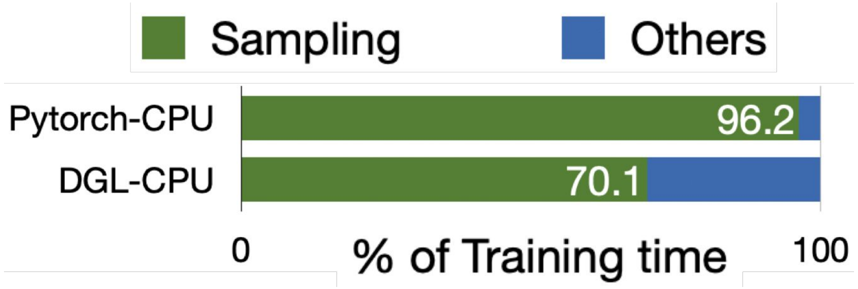
Epoch is split into mini-batches to efficiently process large datasets



GNN Sampling is a Critical Bottleneck

Neighborhood sampling can consume 50–90% of total training time

— gSampler (OSDI '23), GIDS (ICDE '23), Ginex (OSDI '22), NextDoor (VLDB '21)



How to support larger-than memory graphs?

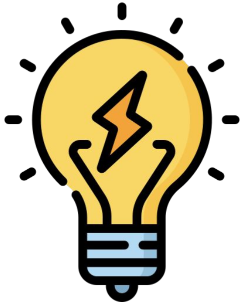
Datasets	#Nodes	#Edges	Size
Papers100M	111M	1.62B	70GB
Hyperlink	3.5B	128B	**3.4TB**
Facebook	1.4B	1T	**8.5TB**

Graph size may exceed main memory

Related Work

System Type	CPU-based	GPU-based	SSD-based
Sampling Device	CPU	GPU	SSD
Graph Stored In	Disk(SSD)	CPU/GPU	SSD
Examples	MariusGNN (EuroSys ‘23) Ginex (EuroSys ‘21)	gSampler (SOSP ‘23) NextDoor (EuroSys ‘21)	FlashGNN (HPCA ‘24) In situ SmartSSD (DaMoN ’24)
Advantages	Able to process larger-than-memory graph	Leverage high computational power of GPUs	Avoid data movement between storage and sampling device
Limitations	- High data movement overhead between SSD and main memory - Low computation power comparing to GPU-based	- Constrained GPU memory - Sampling competes with other tasks for GPU-resources - High computation Cost - Underutilized CPU resources	- Low data transfer bandwidth - Limited adoption due to hardware customizations

Key idea behind RingSampler



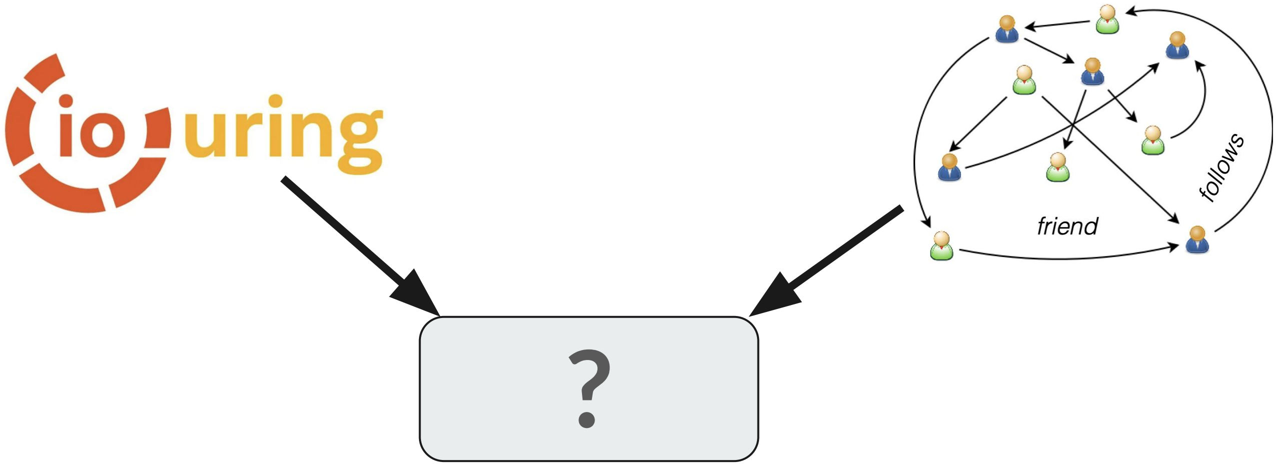
Leverage io_uring, a modern storage API, and high-bandwidth SSDs to perform sampling on larger-than-memory graphs in CPU

Why io_uring

	libaio	io_uring	SPDK
Performance (Scalability, IOPS)	❌	✅	✅
CPU Usage	✅	✅	✅
Ease of Use	✅	✅	❌
Flexibility	❌	✅	❌
Compatibility	✅	✅	❌

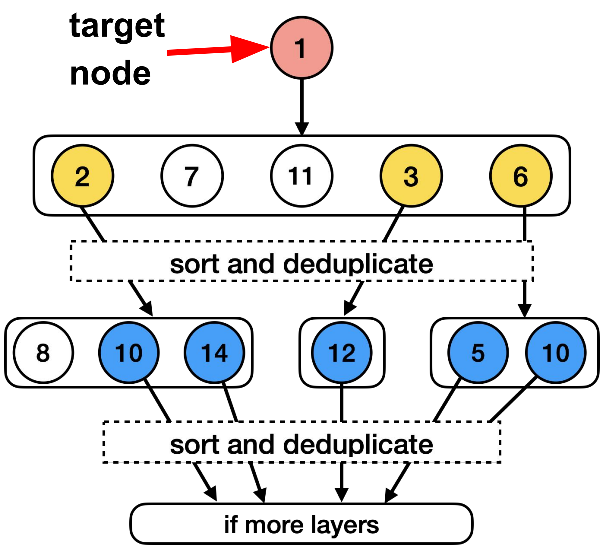
Technical Challenges

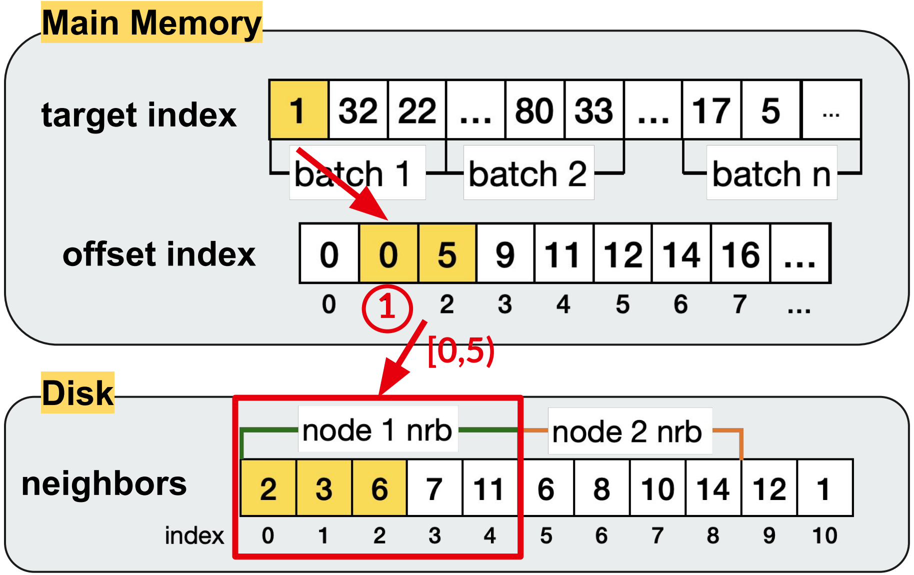
- **How to** minimize data movement between SSD and CPU?
- **How to** integrate and adjust io_uring to GNN Sampling?
- **How to** implement multi-threading to maximize CPU utilization?
- **How to** take the advantage of asynchronous I/O?



Minimize data movement

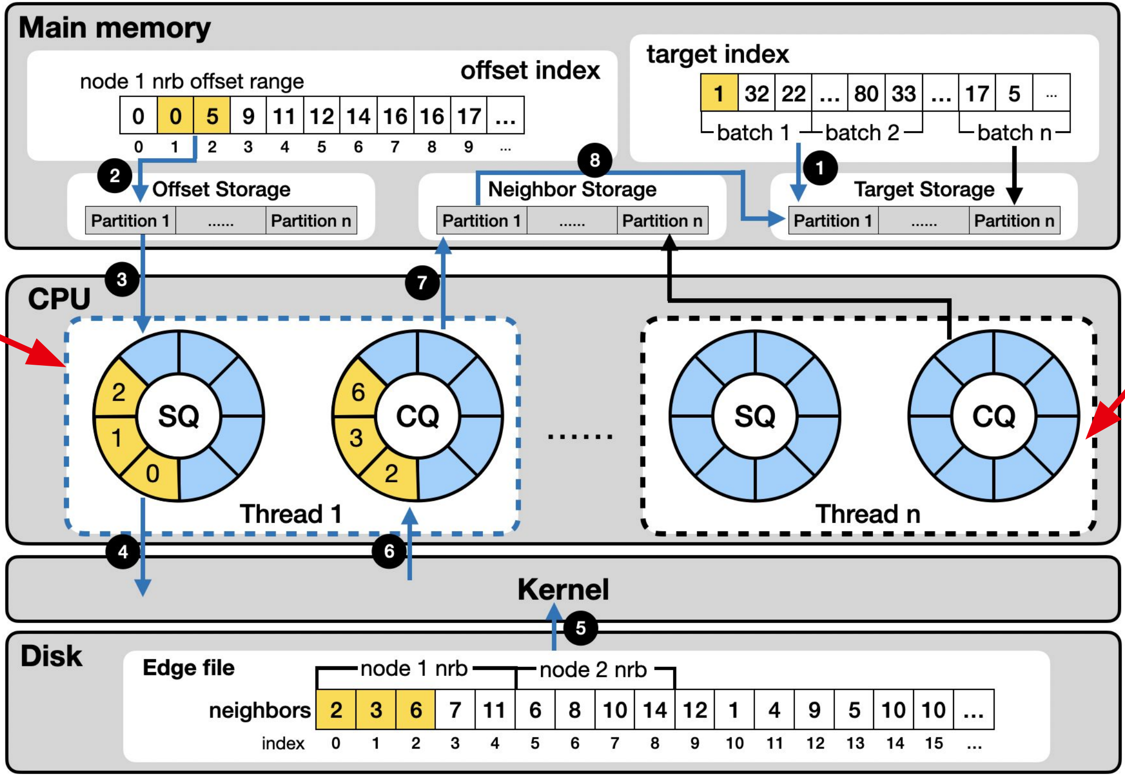
Create two indexes to efficiently select neighbor offsets, allowing direct access to sampled neighbors without loading all neighbors from disk.





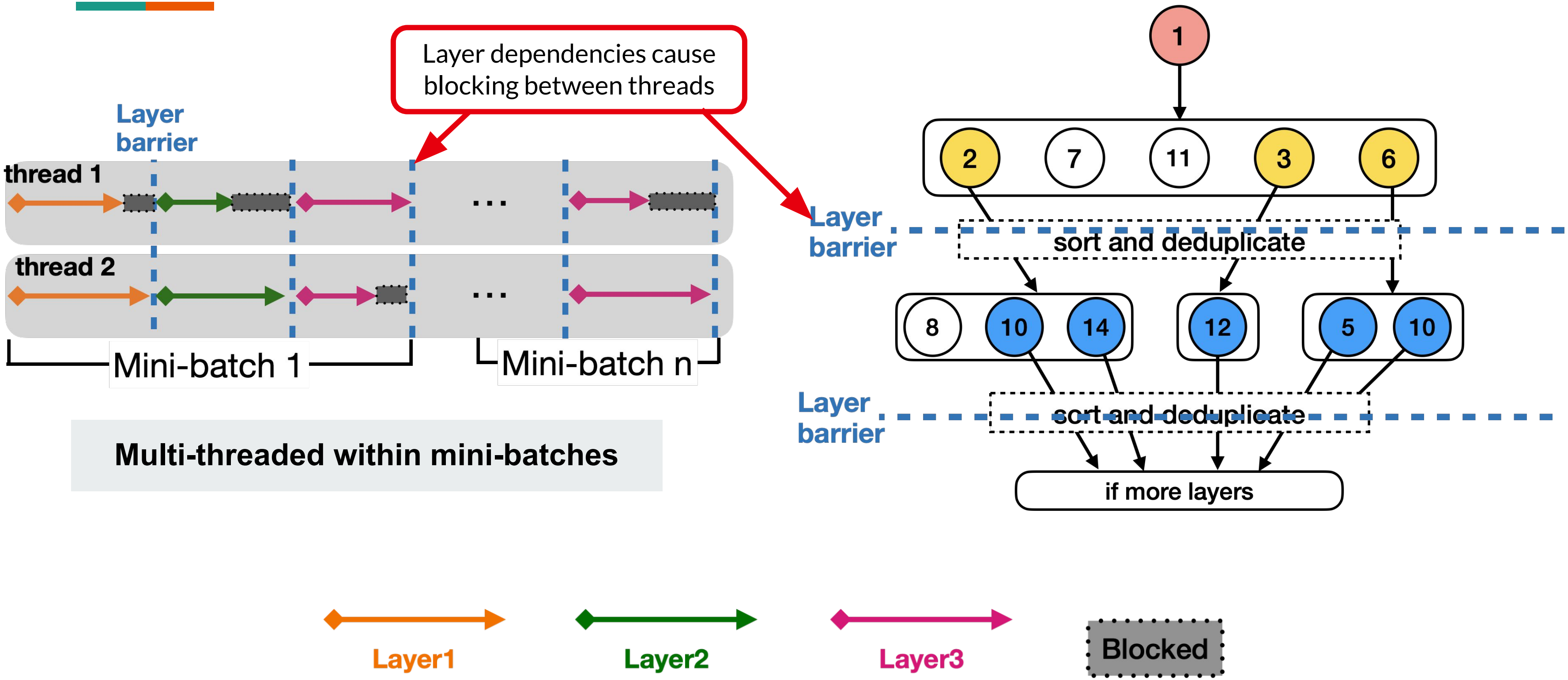
io_uring: Batched I/O Requests and High Parallelism

Submitting multiple requests in a single system call

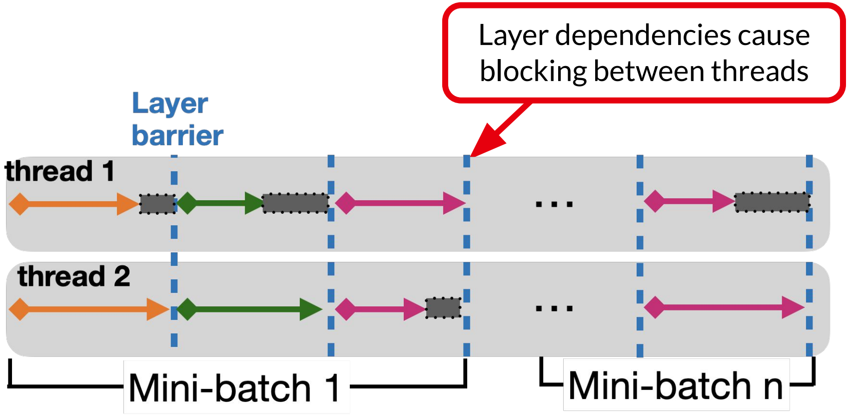


Each thread has its own pair of ring buffers

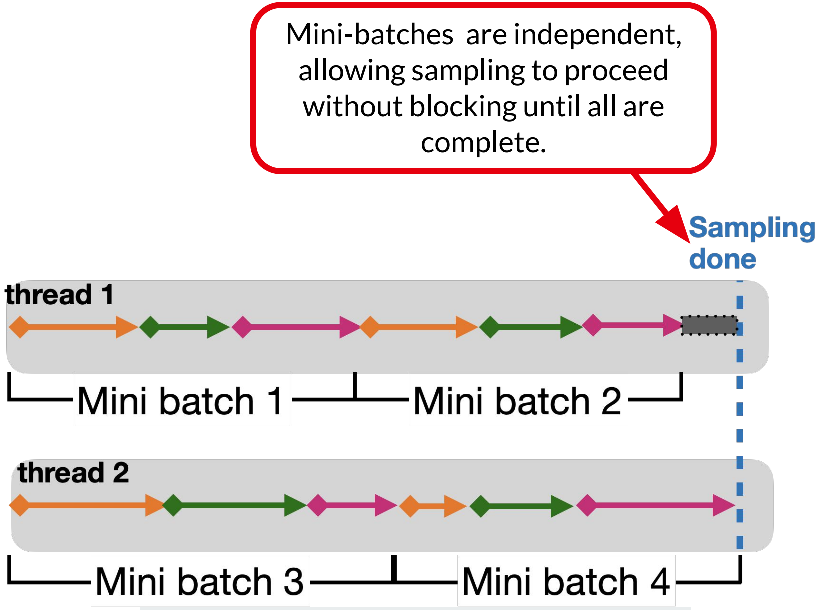
Parallel Design



Parallel Design

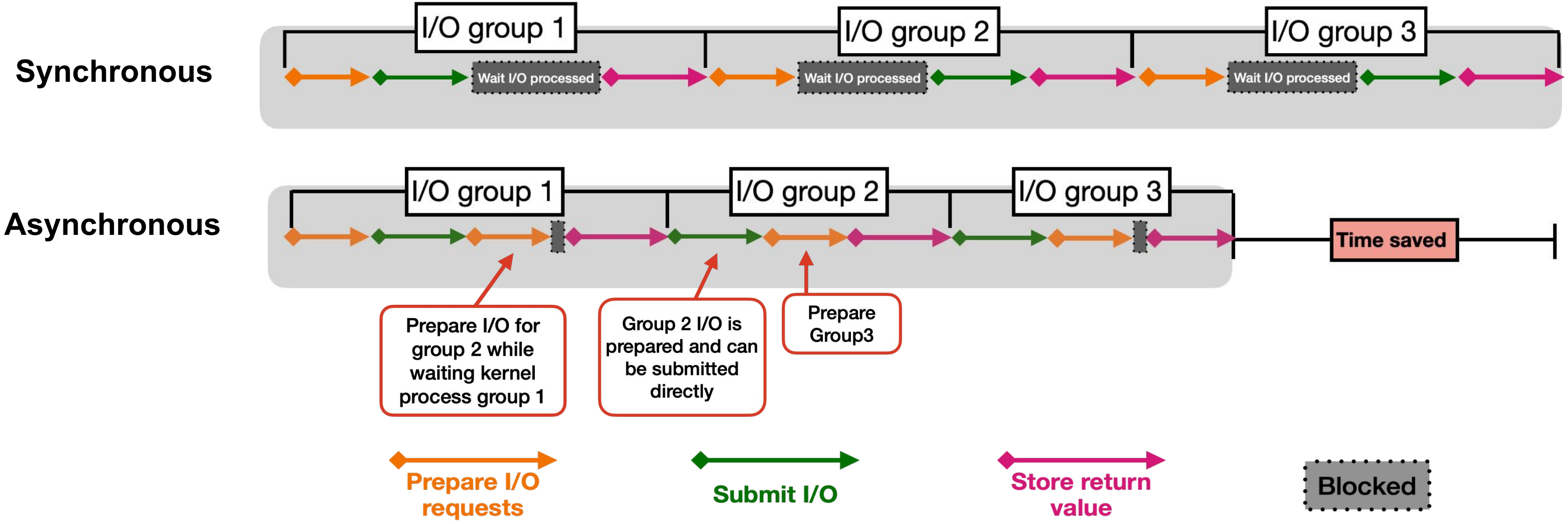


Multi-threaded within mini-batches

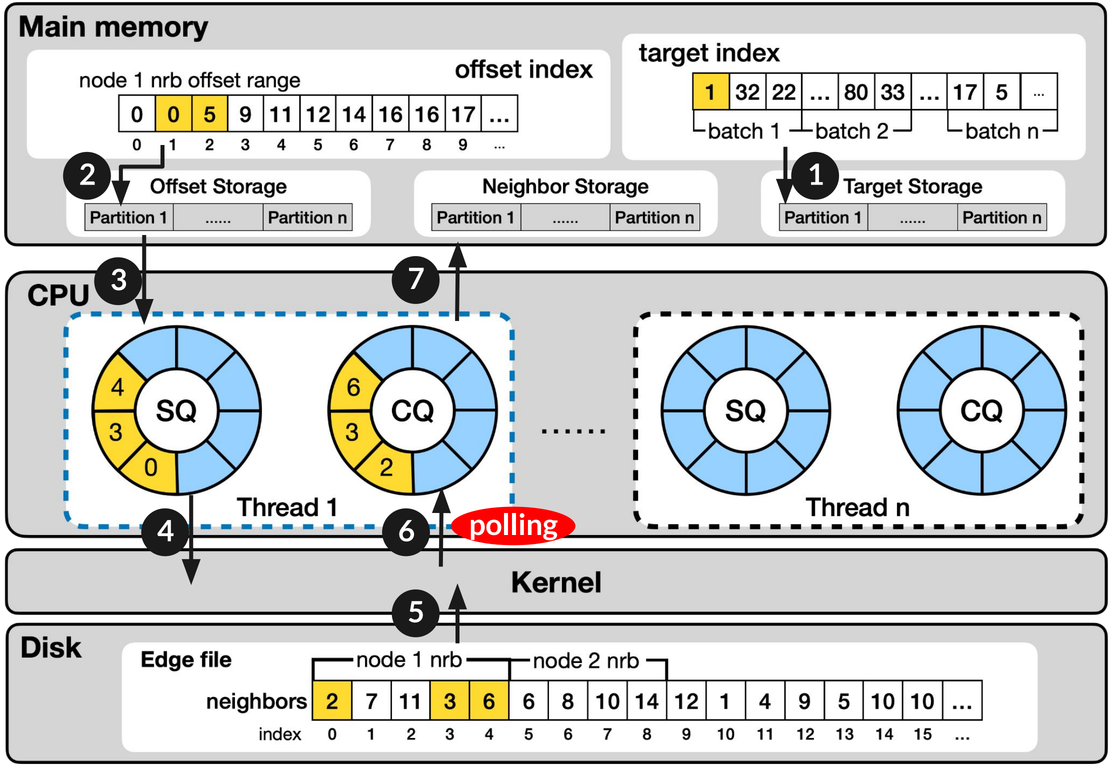


Equally distribute mini-batches across threads to process in parallel

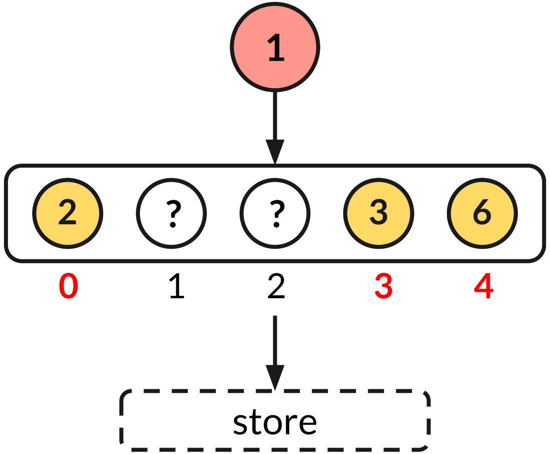
Asynchronous Design



Workflow



Main memory
node 1 nrb offset range
offset index
0 0 5 9 11 12 14 16 16 17 ...
0 1 2 3 4 5 6 7 8 9 ...
target index
1 32 22 ... 80 33 ... 17 5 ...
batch 1
batch 2
batch n
2
Offset Storage
Partition 1 ...... Partition n
Neighbor Storage
Partition 1 ...... Partition n
1
Target Storage
Partition 1 ...... Partition n
CPU
3
7
4 3 0
SQ
6 3 2
CQ
Thread 1
......
SQ
CQ
Thread n
4
6
polling
Kernel
5
Disk
Edge file
node 1 nrb
node 2 nrb
neighbors
2 7 11 3 6 6 8 10 14 12 1 4 9 5 10 10 ...
index
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ...



1
2 ? ? 3 6
0 1 2 3 4
store

Experimental Configuration

Machine

- Our setup includes an AMD EPYC 7713P **64C/128T CPU**, 252GB DRAM, a 4TB **Samsung SmartSSD**, and an **NVIDIA A100 80GB GPU**. The software environment consists of Ubuntu 20.04, CUDA 12.1, PyTorch 2.3.1+cu121, and DGL v2.3.0+cu121.

Model Configuration

- **3-layer GraphSAGE** model with a fanout of {20, 15, 10} and a mini-batch size of 1024.

Baselines

- **DGL v2.3 (Deep Graph Library)**: In-memory CPU/GPU-based
- **gSampler (SOSP '23):** In-memory GPU-based
- **MariusGNN (EuroSys '23)**: Out-of-memory CPU-based
- **In-situ SmartSSD (DaMoN '24)**: SSD-based

Datasets Used For Evaluation

Dataset	Vertices	Edges	Raw Size (GB)	Bin Size (GB)
ogbn-papers	111M	1.6B	24.1	6.8
Friendster	65M	3.6B	30.1	13.5
Yahoo	1.4B	6.6B	66.9	35.3
Synthetic	134M	8.2B	140.8	31.7

ogbn-papers and Friendster: Fit in memory; used to evaluate in-memory sampling performance.

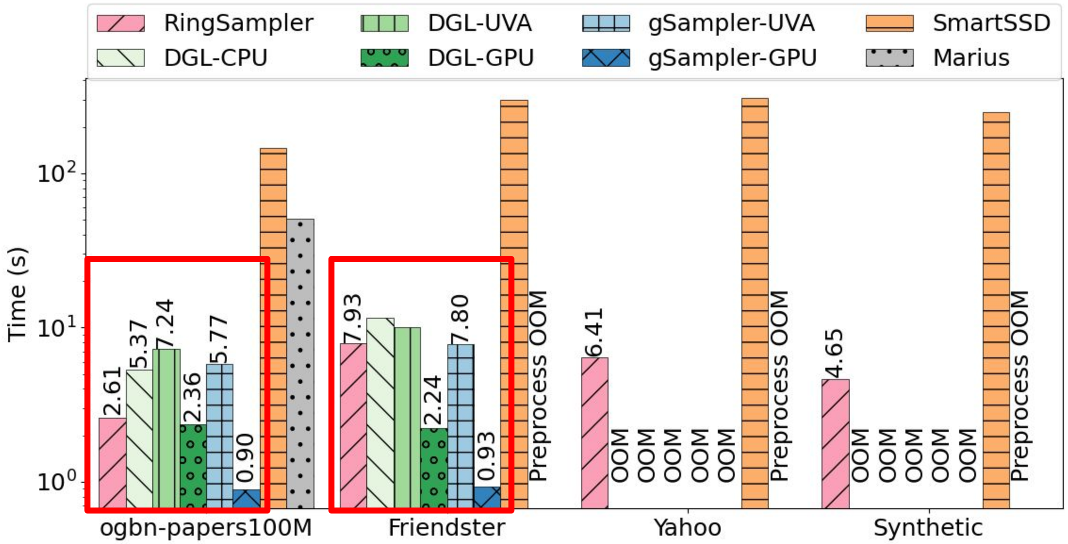
Yahoo and Synthetic: Extremely large graphs that do not fit in memory; used to test the system's ability to handle larger-than-memory graphs.

Faster Than In-Memory CPU Systems, Competitive with GPU Solutions

DGL-CPU: Graph stored and sampled on CPU

DGL-GPU, gSampler-GPU: Graph stored and sampled on GPU

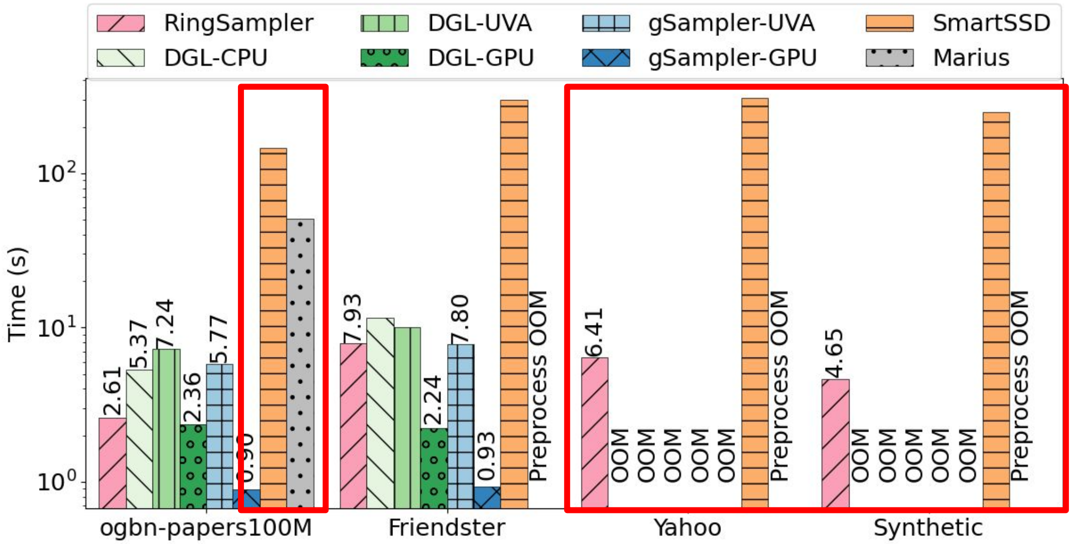
DGL-UVA, gSampler-UVA: Graph stored on CPU, sampled on GPU using Unified Virtual Addressing (UVA) to access data



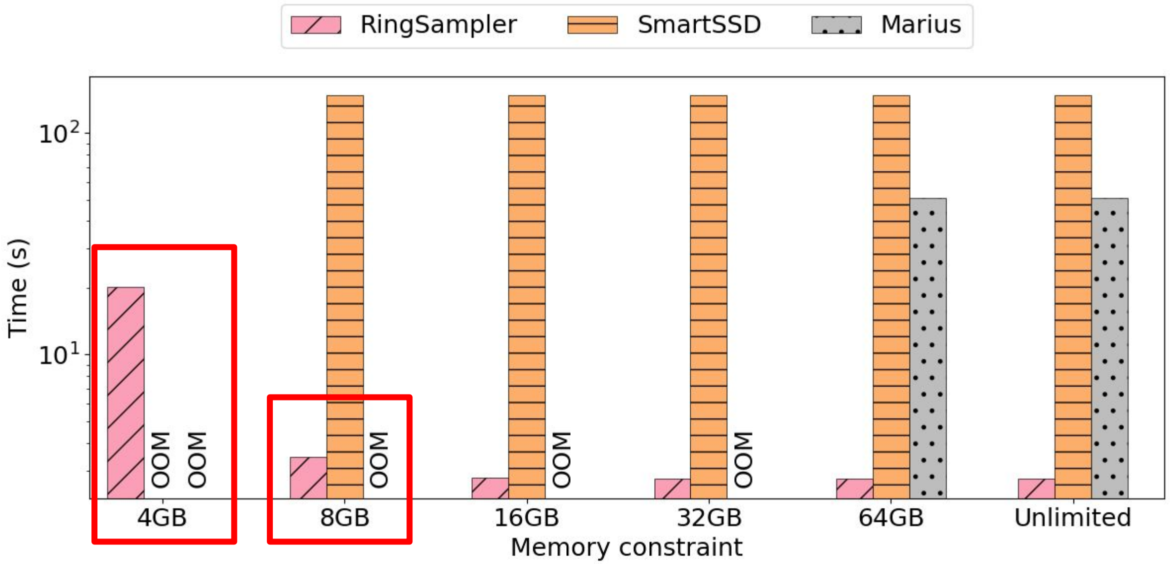
Scales to Large Graphs and Outperforms Baseline

MariusGNN: Graph stored on SSD and partially loaded into memory for CPU-based sampling

SmartSSD: Graph stored and sampled directly on SSD using FPGA

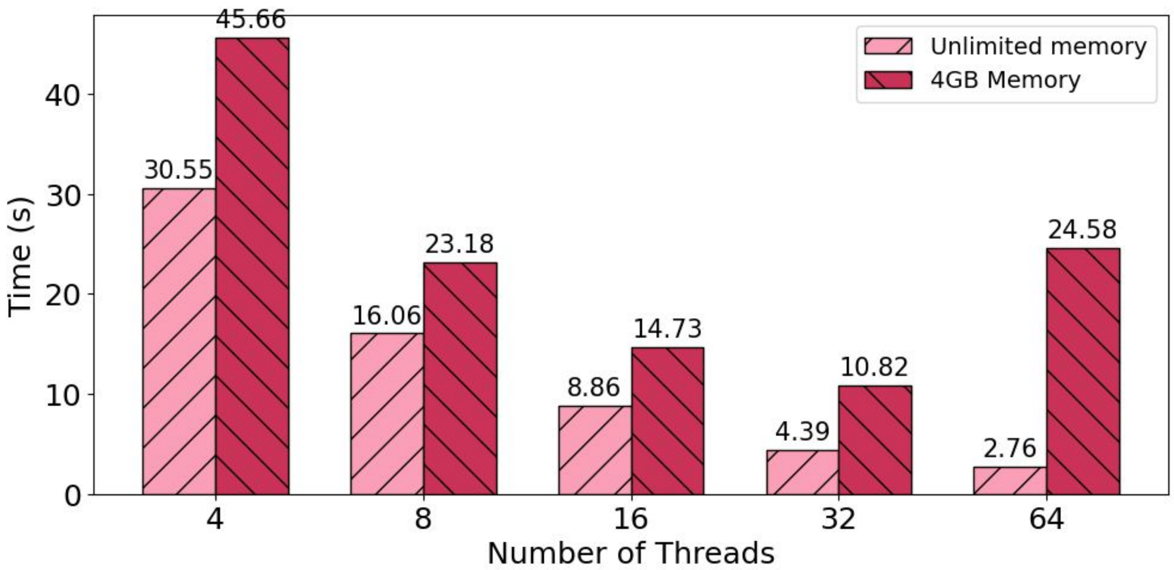


Low Memory Requirement for Large-scale Sampling



Only 4 GB of memory is needed to sample a billion-edge graph (ogbn-papers)

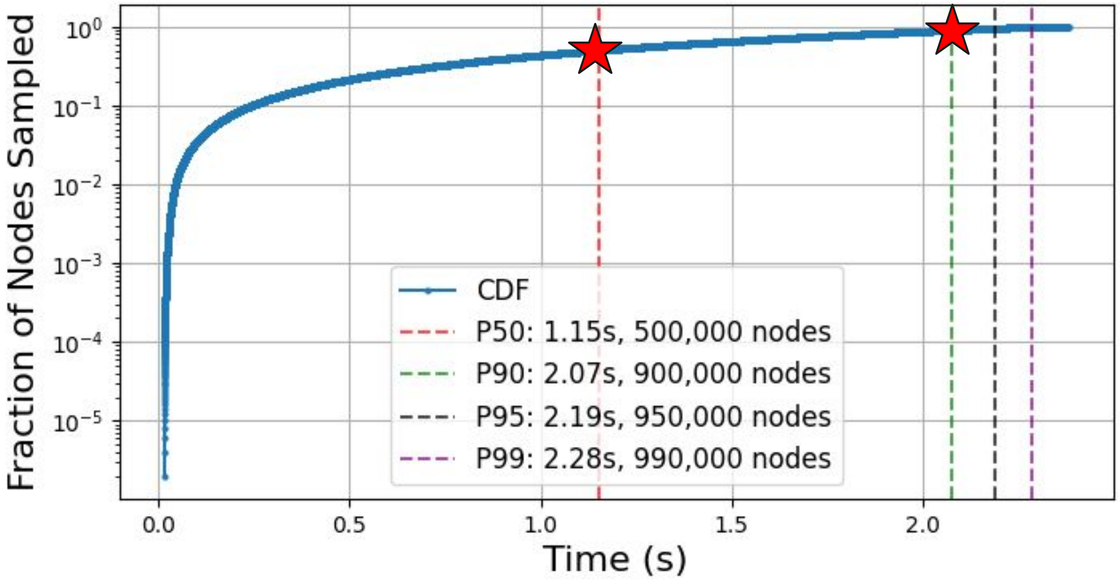
Linear Scalability with No of Threads



Sampling run time decreases almost linearly with the number of threads, up to the maximum number of available cores

(ogbn-papers)

Low Latency Sampling for Real-Time GNN Inference

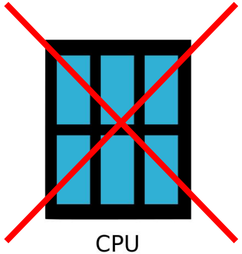


50% of sampling requests complete within 1.15s, and 90% within 2.07s

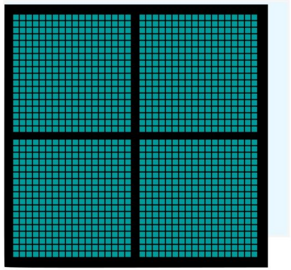
When RingSampler May Not Be the Best Choice

Large GPUs

- When GPU is large enough, GPU-based sampling Systems (gSampler) will be the best choice.



CPU
Multiple Cores



GPU
Thousands of Cores

Small Graphs

- For graphs that fit entirely in memory, the performance gains from io_uring-based sampling are minimal

Future Work

End-to-end implementation

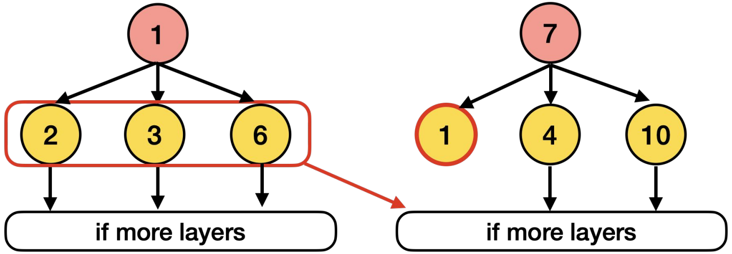
- Build an end-to-end system by integrating RingSampler into GNN frameworks like DGL to enable asynchronous CPU-based sampling alongside GPU-based feature aggregation.

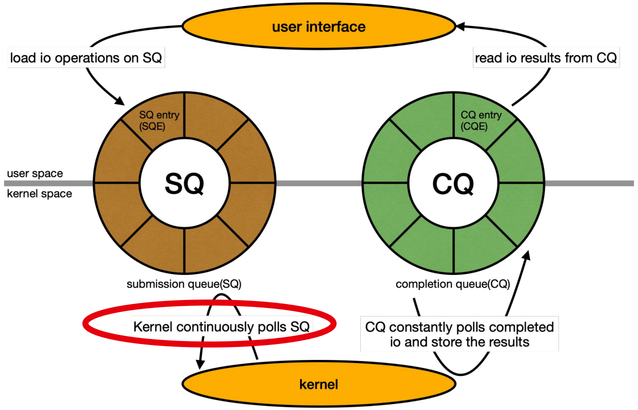
Exploiting caching strategy and sampling reuse

- Due to our parallel design, threads do not share states, which can result in reading the same nodes multiple times.

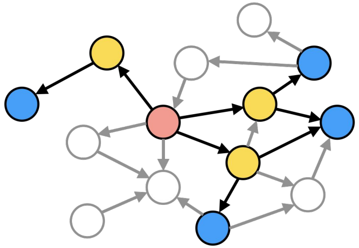
io_uring kernel polling mode

- Kernel polling mode auto-submits I/O when the submission queue is full.

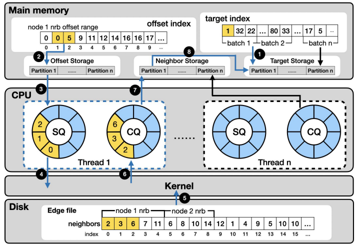




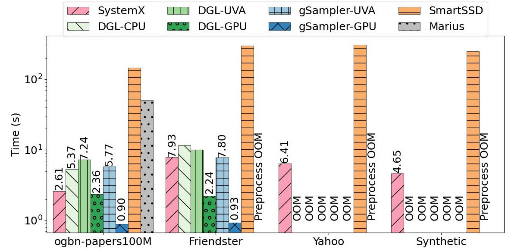
RingSampler



As graph data scales to terabyte sizes, out-of-memory challenges arise, making **efficient sampling** difficult



A CPU-based GNN system **leveraging io_uring** for asynchronous, batched I/O, and multi-threading to maximize CPU utilization



Efficiently handles larger-than-memory graphs and outperforms existing baselines.

Contact:

Qixuan(Talia) Chen
taliac@bu.edu

RingSampler

https://github.com/CASP-Systems-BU/RingSampler