LithOS: An Operating System for Efficient Machine Learning on GPUs

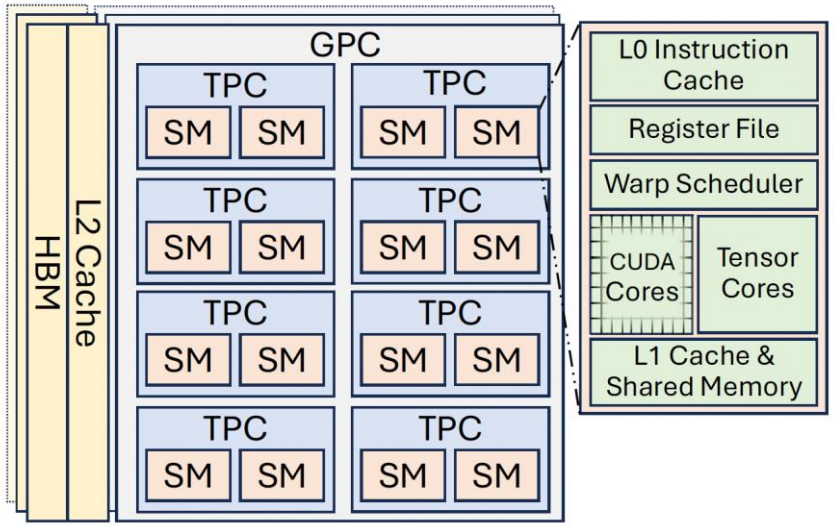
Patrick H. Coppock, Brian Zhang, Eliot H. Solomon, Vasilis Kypriotis, Leon Yang†, Bikash Sharma†,
Dan Schatzberg†, Todd C. Mowry, and Dimitrios Skarlatos
Carnegie Mellon University †Meta

Presented by Qingyuan Chen

Contents

GPU architecture

- GPU Architecture



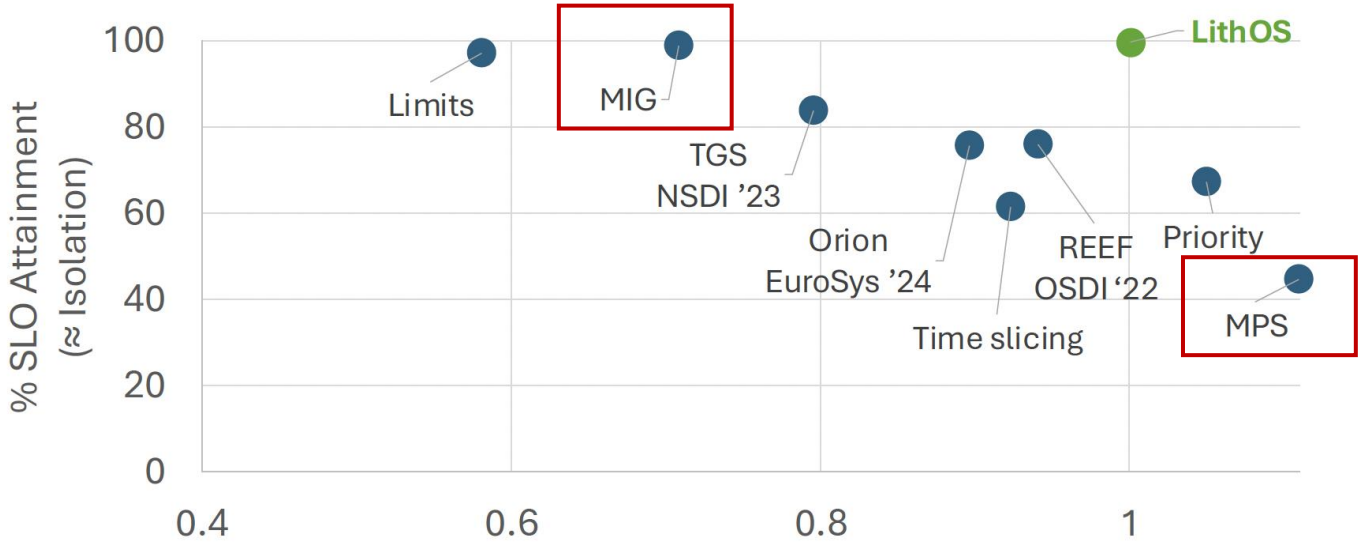
- GPU Programming
  - kernels
- GPU Streams

Current GPU sharing solution problem

- Coarse-grained scheduling
- Temporal-only
- Non-transparent

SOTA GPU sharing solution

- TGS (NSDI ’23) Orion (EuroSys ’24) REEF (OSDI ‘22)
- MIG, MPS, Priority, Time slicing



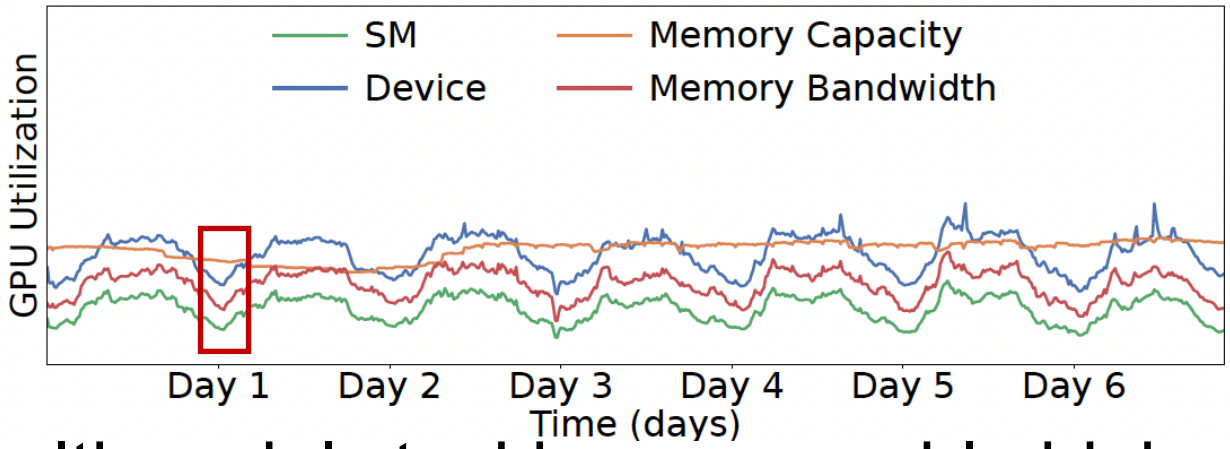
SOTA GPU sharing solution

- Multi-Process Service (MPS)
  - allowing multiple tasks to use the GPU concurrently
  - No isolation, can’t meet SLO
- Multi-Instance GPU (MIG)
  - Static partitions the GPU (in GPC granularity)
  - Reconfiguration overheads

Contents

GPU utilization is low

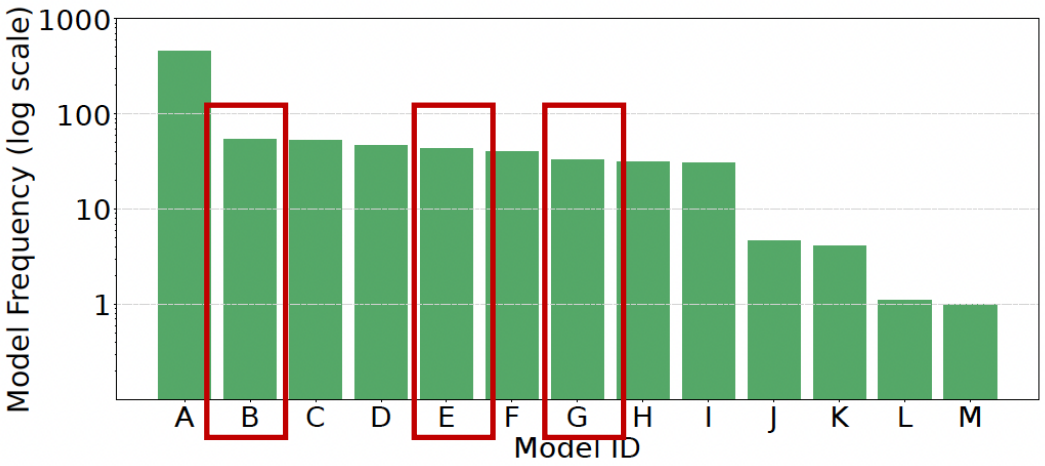
- Analyze a subset of inference services at Meta
  - Device utilization: 25%
  - SM utilization: 15%
  - Memory bandwidth: 20%



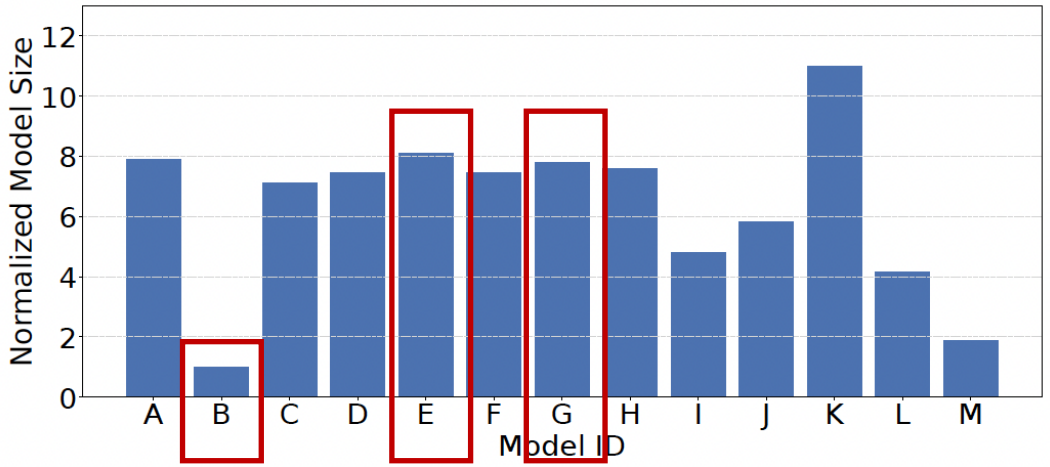
- multi-model stacking can enable high utilization

Inference Traffic

- Sample 13 of the most commonly used models



Model frequency distribution



Model size distribution

- This highlights the opportunity to collocate models of different sizes without violating their SLAs

GPU can’t be shared efficiently

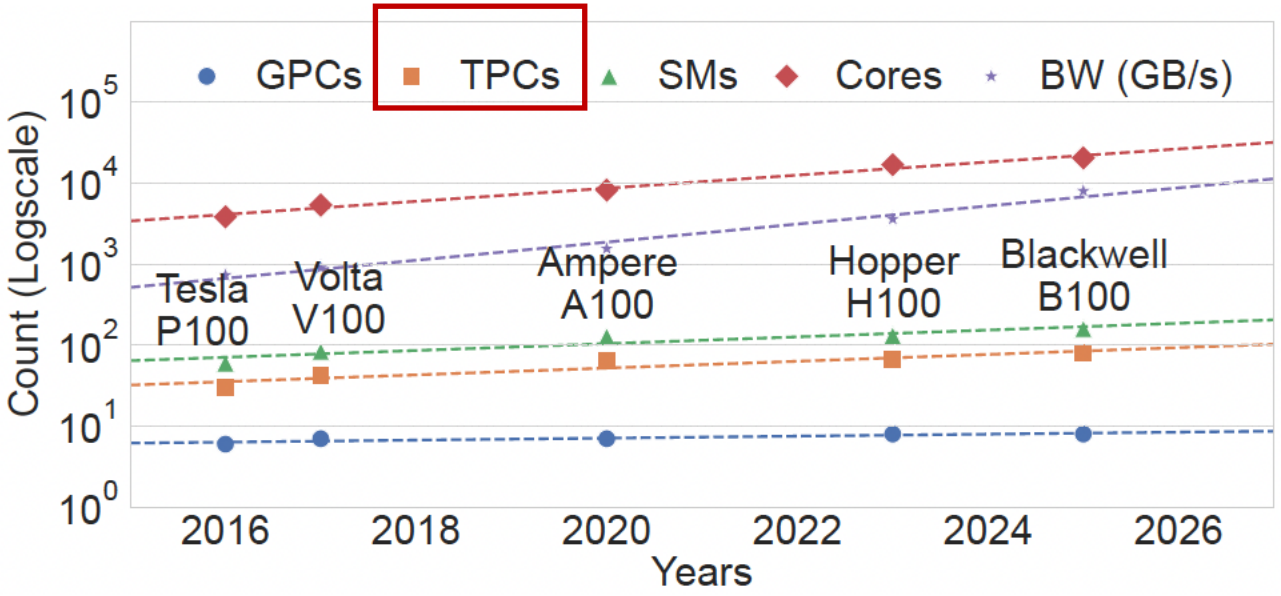
CPU	GPU
Threads scheduled into cores	No-control of kernel resource allocation
Preemption	No Preemption

LithOS
TPC scheduler
Kernel atomizer

Contents

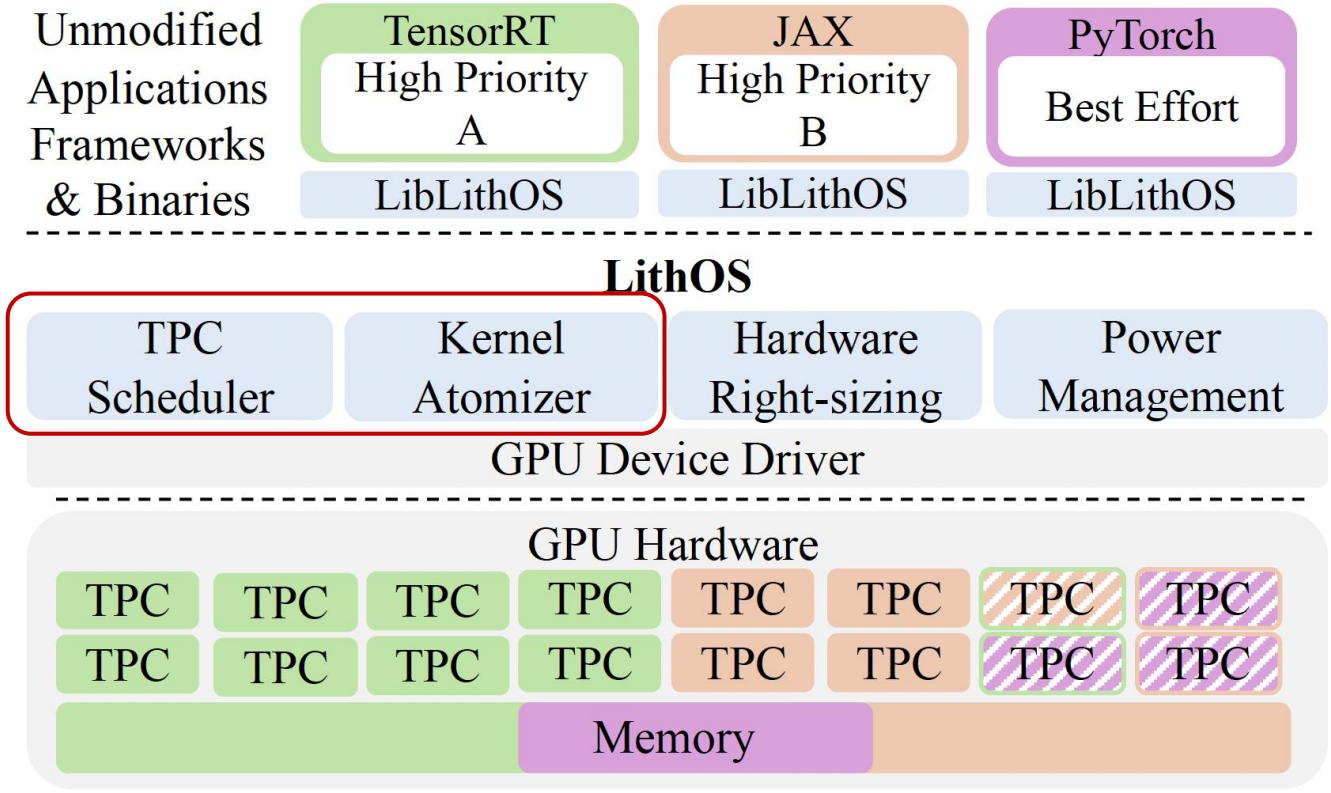
LithOS scheduling granularity

- Coarse GPC-level partitioning (e.g., MIG) will waste even more resources
- Intra-SM control is best handled by hardware and compilers

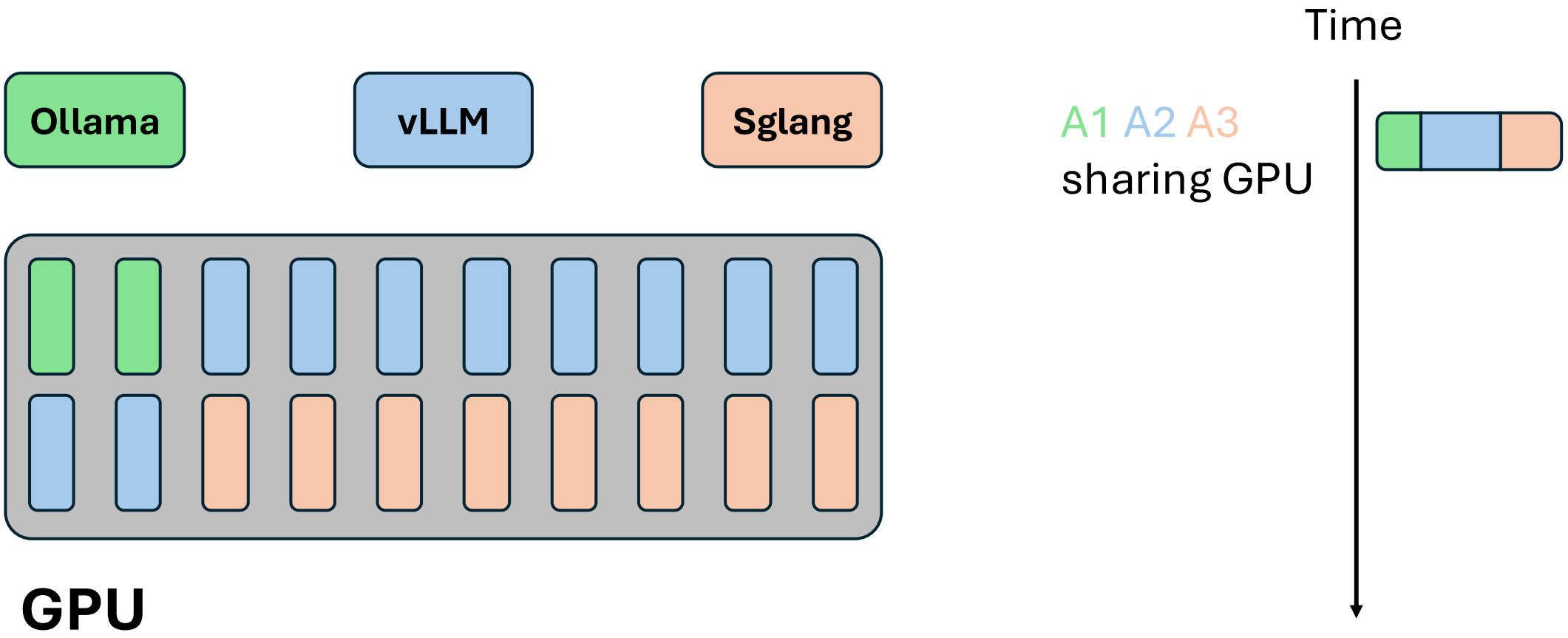


H100:
8 GPC
66 TPC
132 SM

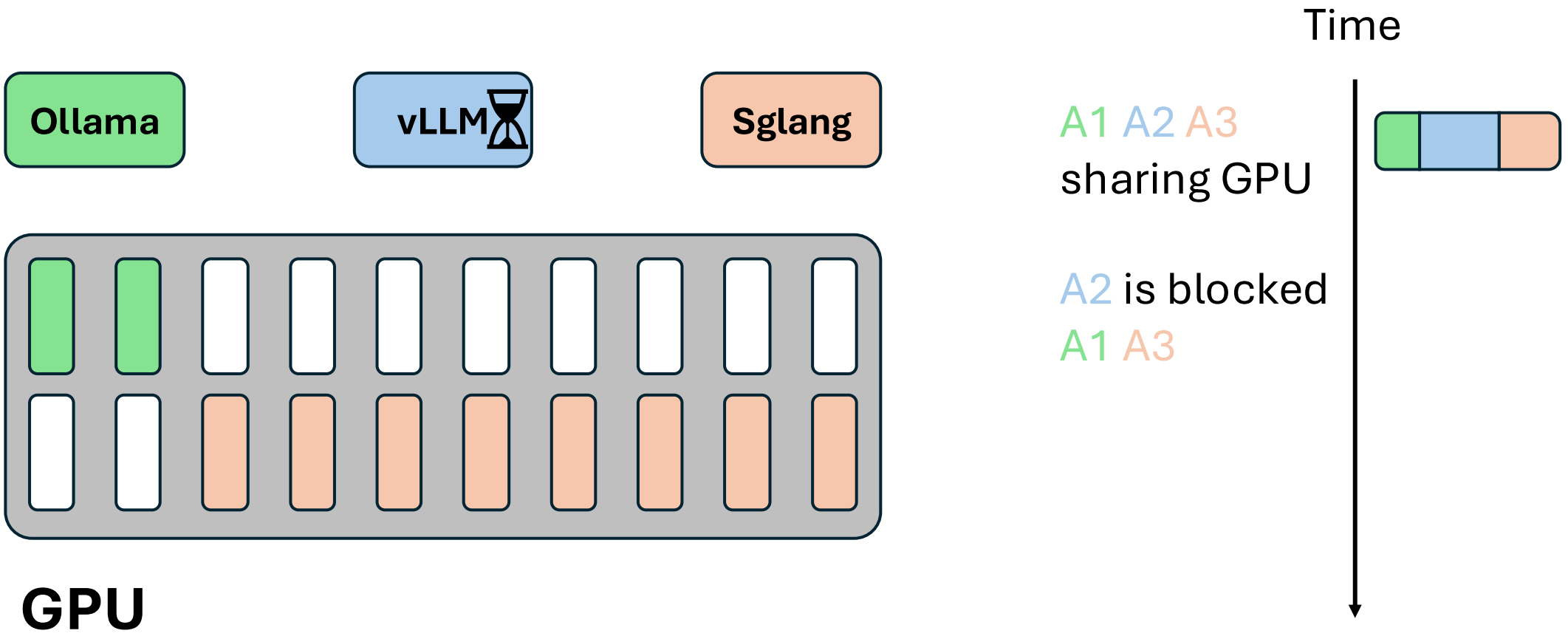
LithOS architecture overview



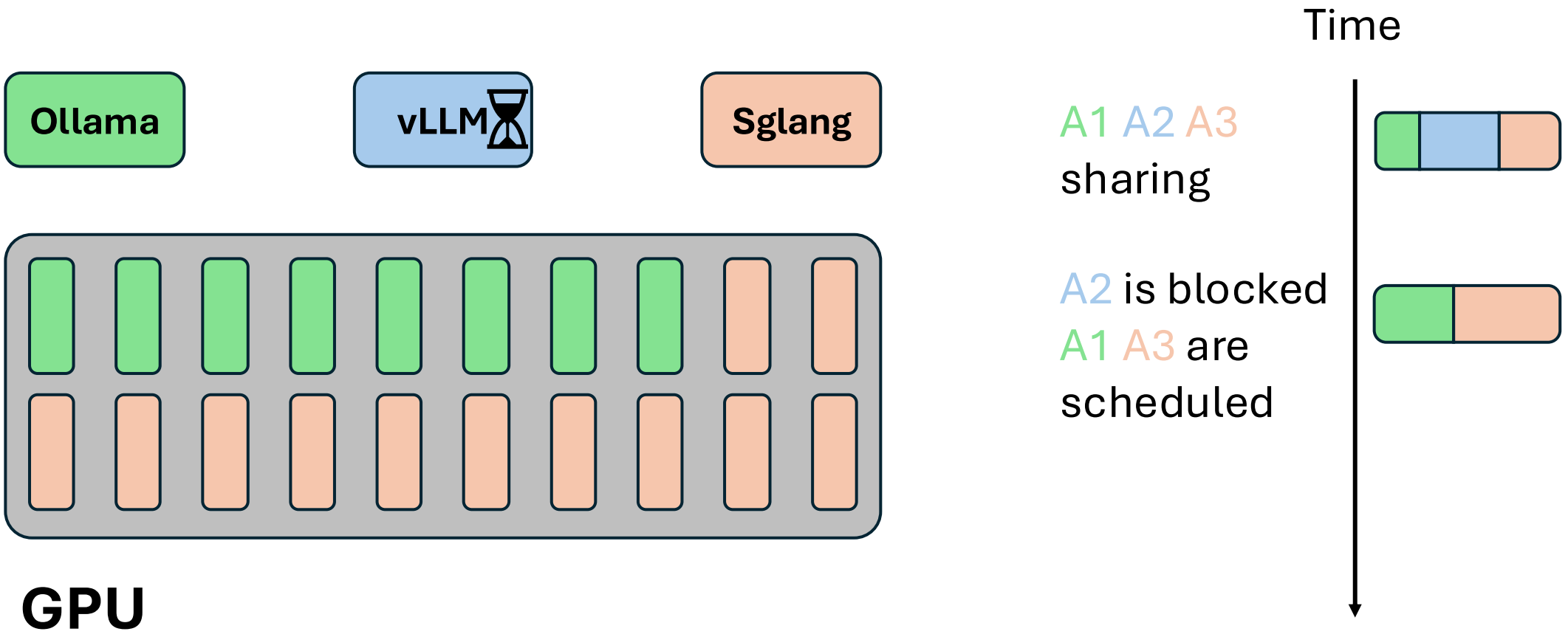
TPC Scheduler



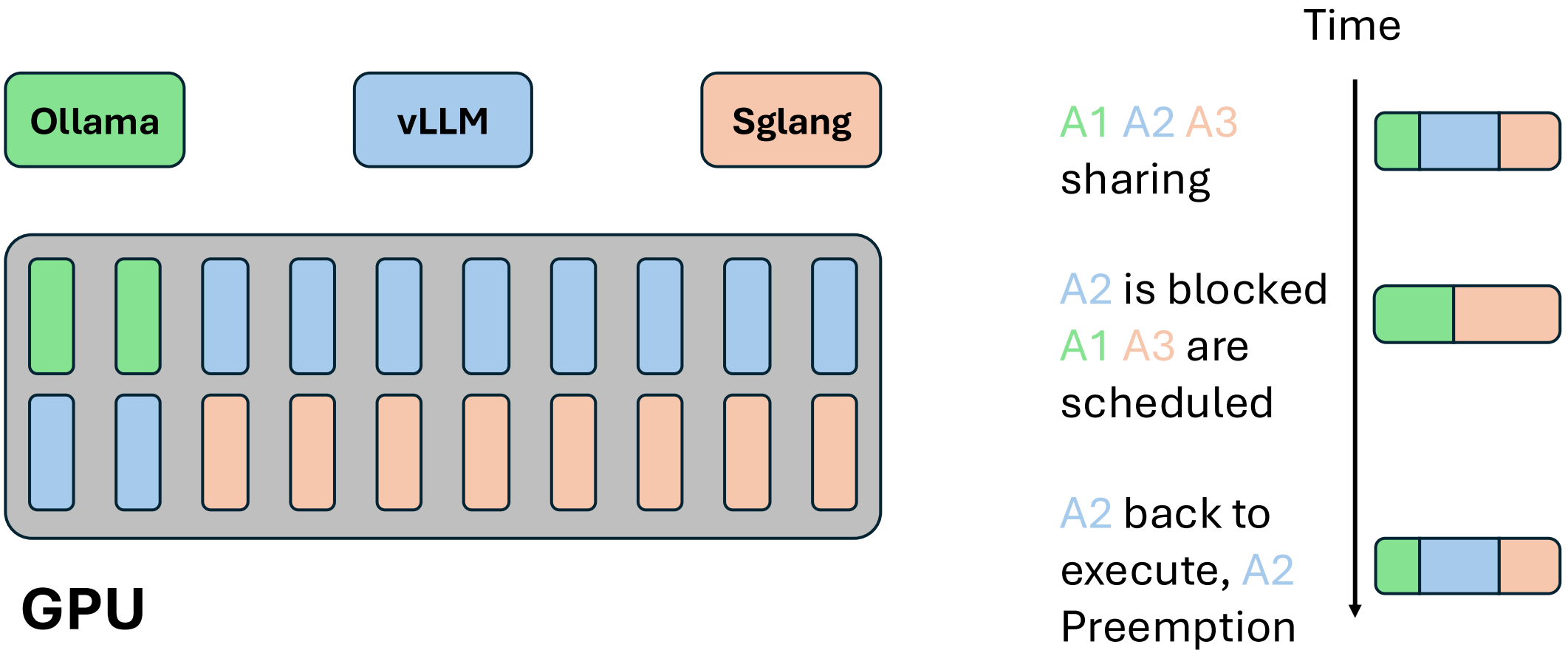
TPC Scheduler



TPC Scheduler

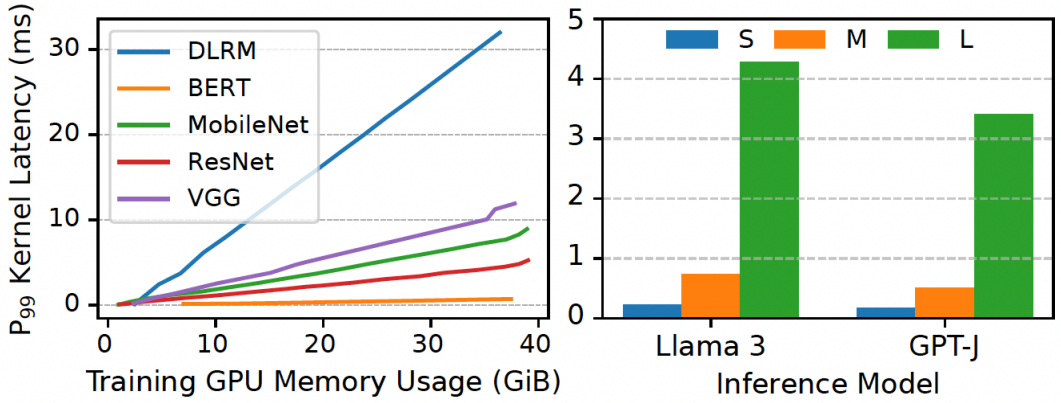


TPC Scheduler

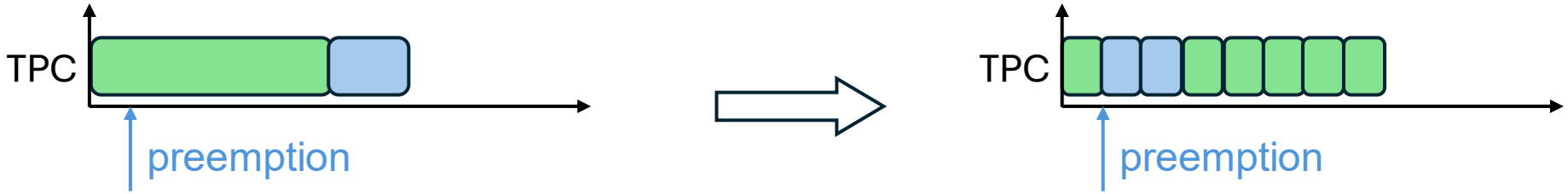


Kernel Atomizer for preemption

- Kernel runtime varies a lot



- Kernel Atomizer: split the kernel by dividing the predicted kernel duration



Atomization implementation details

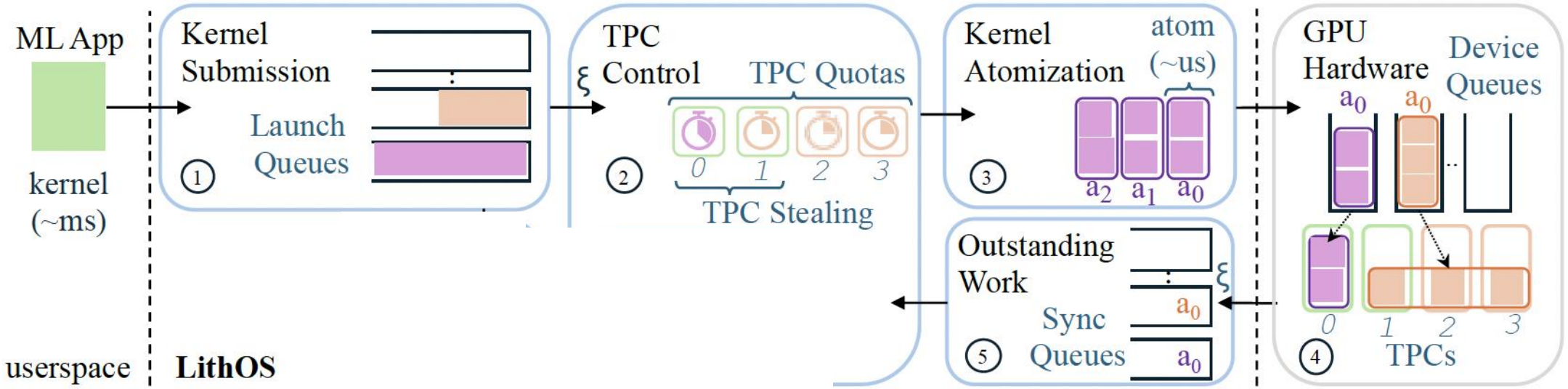
- Prelude kernel

Algorithm 1 Prelude Kernel Pseudocode.

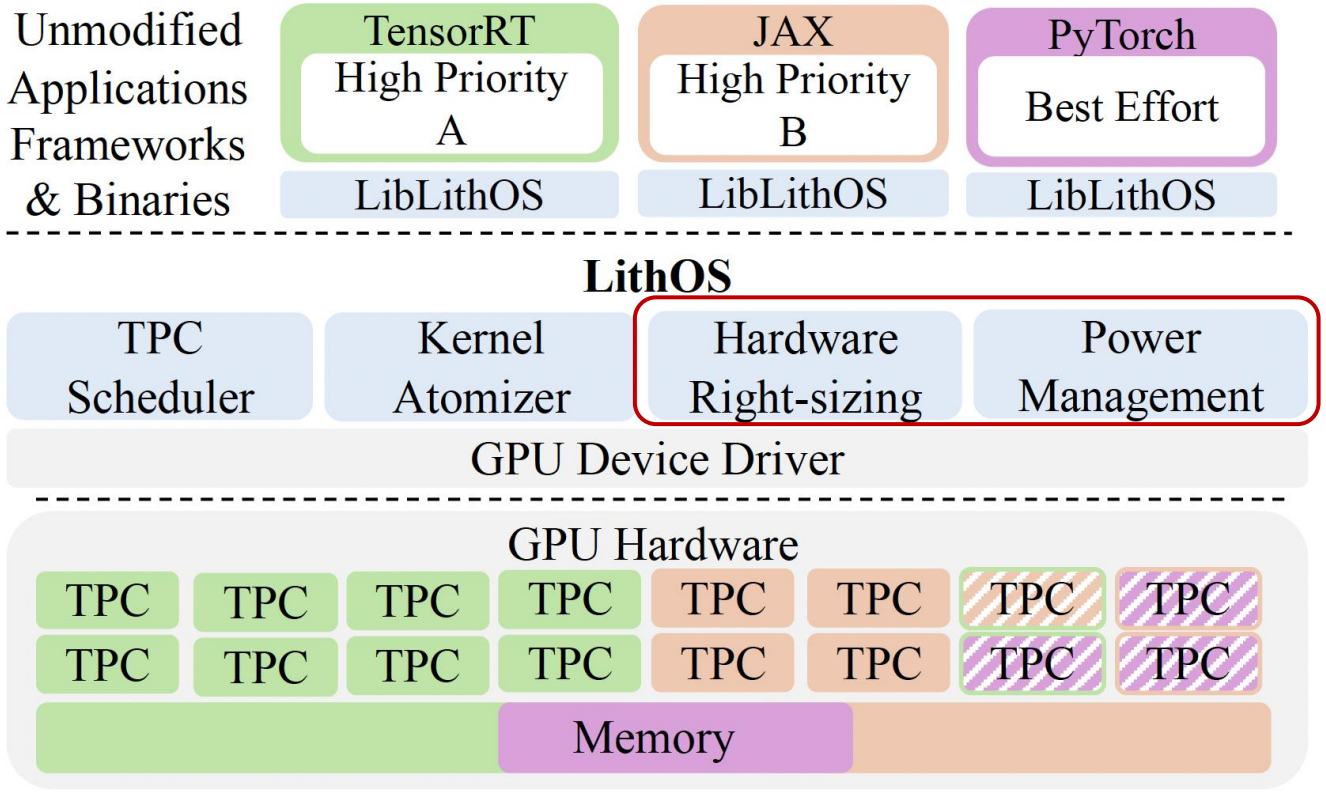
```
1  kernel fn prelude(*args):
2      let atom : *const AtomMetadata = AtomMetadataAddr as _
3      let block_idx = blockIdx.z * gridDim.y * gridDim.x
4                    + blockIdx.y * gridDim.x
5                    + blockIdx.x
6      if atom->block_idx_lo <= block_idx < atom->block_idx_hi:
7          atom->kernel_entrypoint(*args)
```

- Inject Prelude logic by modifying the Queue MetaData (QMD) struct used to launch kernels, patch the QMD's program address to point to the Prelude

LithOS operations overview

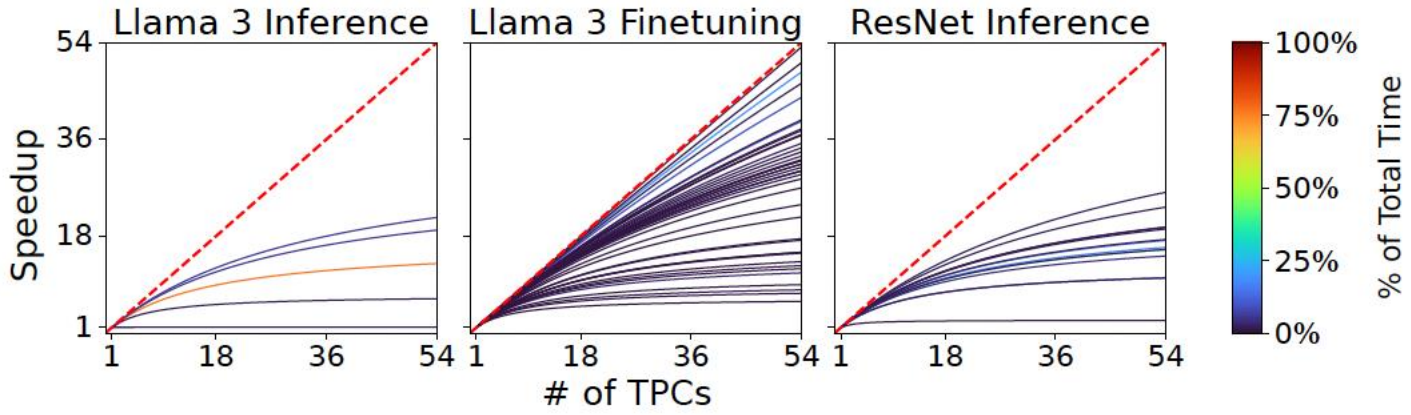


LithOS architecture overview



Right-Sizing Hardware Resources

- Adaptive, per-kernel scheduling
  - Individual kernels exhibit diverse scaling behaviors
  - The distribution of execution time across kernels is workload-dependent



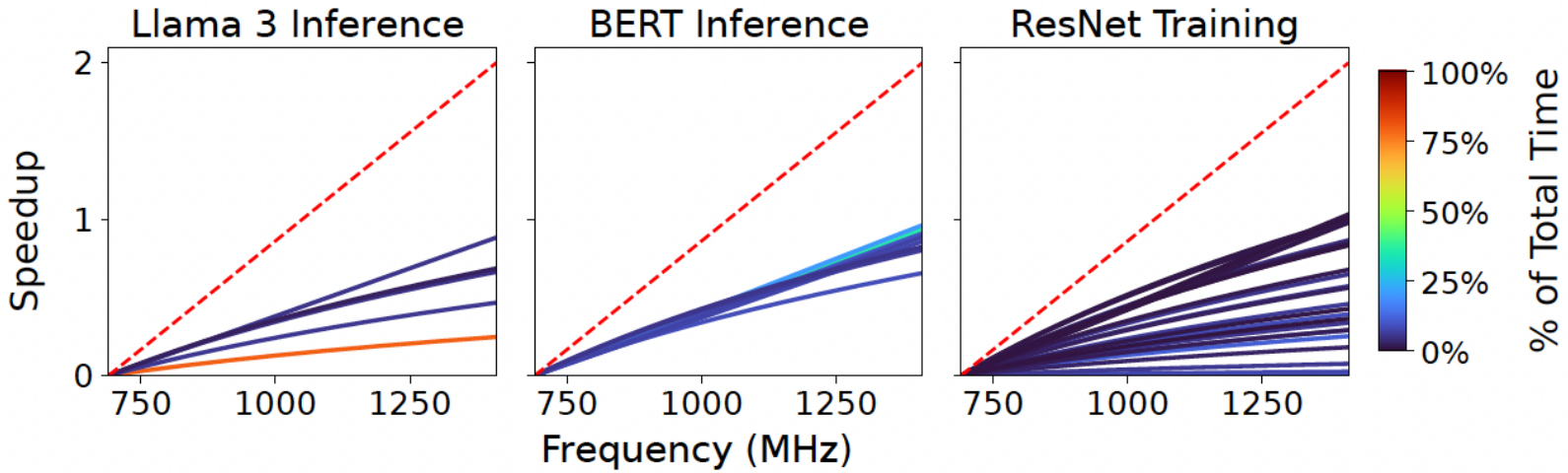
Right-Sizing Hardware Resources

- Modeling Kernel Scaling
  - Latency = m / TPC_num + b
  - based on two points: the latencies of a kernel running with all TPCs and 1 TPC

- Latency slip parameter *k (means increase the kernel's latency by, at most k)*
  - Choose the minimum TPCs using the scaling model

Power Management

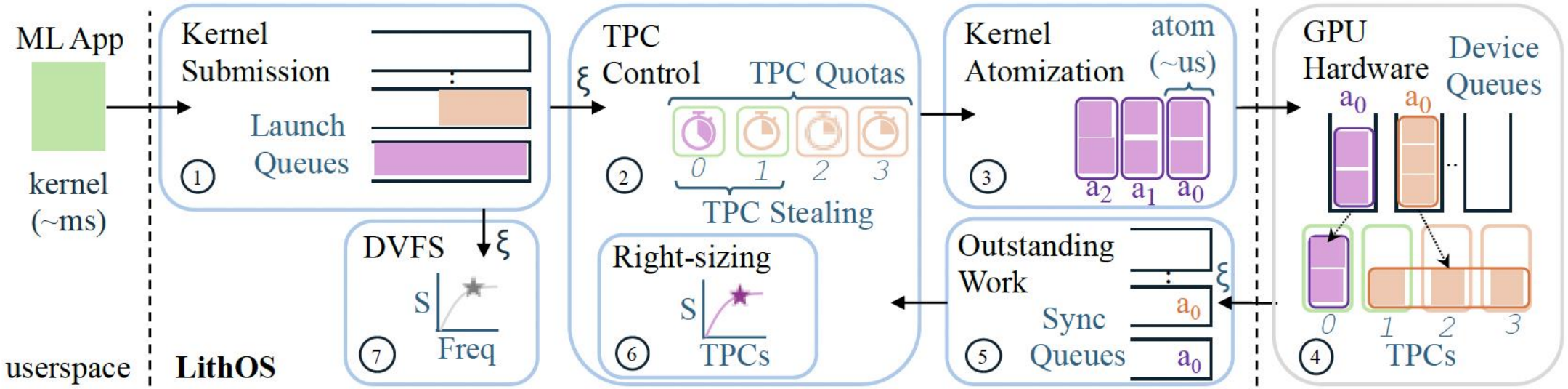
- Dynamic Voltage Frequency Scaling (DVFS)



- Modeling Frequency Scaling

$$k = \frac{lat(f_{th})}{lat(f_{max})} - 1 = s \cdot (\frac{f_{max}}{f_{th}} - 1) \implies \sum w * s \cdot (\frac{f_{max}}{f_{final}} - 1) \leq k$$

LithOS operations overview



LithOS implementation details

- LithOS build on top of MPS
- LithOS TPC mapping
  - LithOS extend prior reverse-engineering work libsmctrl
- Kernel atomization

Contents

Evaluation setup

- Testbed
  - GPU: A100 (108 SM 40 GB HBM)
  - CPU: 30 core
  - DRAM: 216 GB
- Baselines:
  - TGS (NSDI ’23) Orion (EuroSys ’24) REEF (OSDI ‘22):
  - MIG, MPS, Priority, Time slicing

Evaluation setup

- Models and configurations

Model	Mem. (GiB)	Batch Size	Latency (ms)
VGG-19 [56]	17.4	120	291
ResNet-50 [28]	18.4	184	281
MobileNetV2 [53]	18.4	216	254
DLRM [40]	6.7	32768	74
BERT-Large [20]	17.3	20	159
Llama 3 Finetuning	32.0	4	690

Table 1. *Training model parameters.*

Training batch size is adjusted to use half HBM

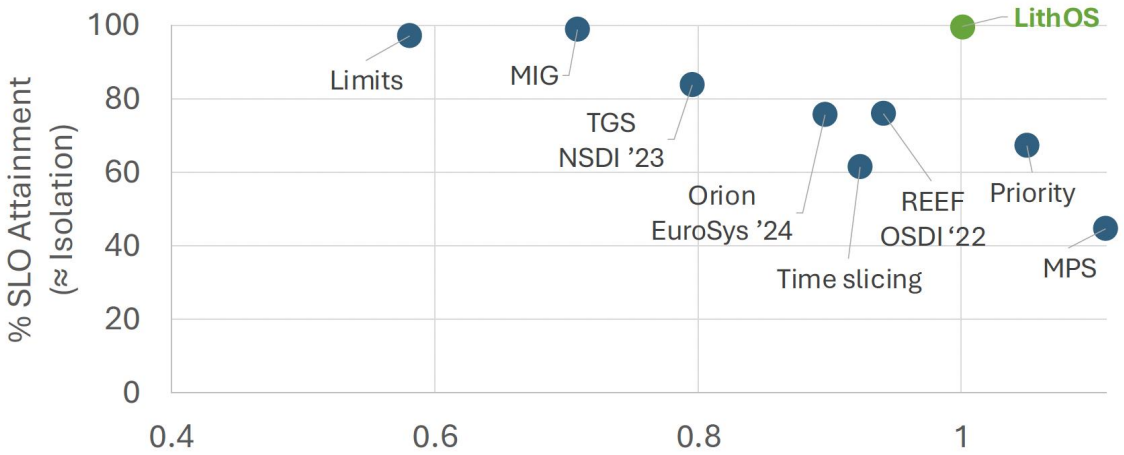
Model	Framework	Load (rps)	Constraint (ms)
ResNet [28]	TensorRT	1000	15
RetinaNet [38]	ONNX Runtime	9	100
Llama 3 [25]	TensorRT-LLM	0.5	2000
GPT-J [62]	TensorRT-LLM	0.5	2000
BERT [20]	TensorRT	30	130

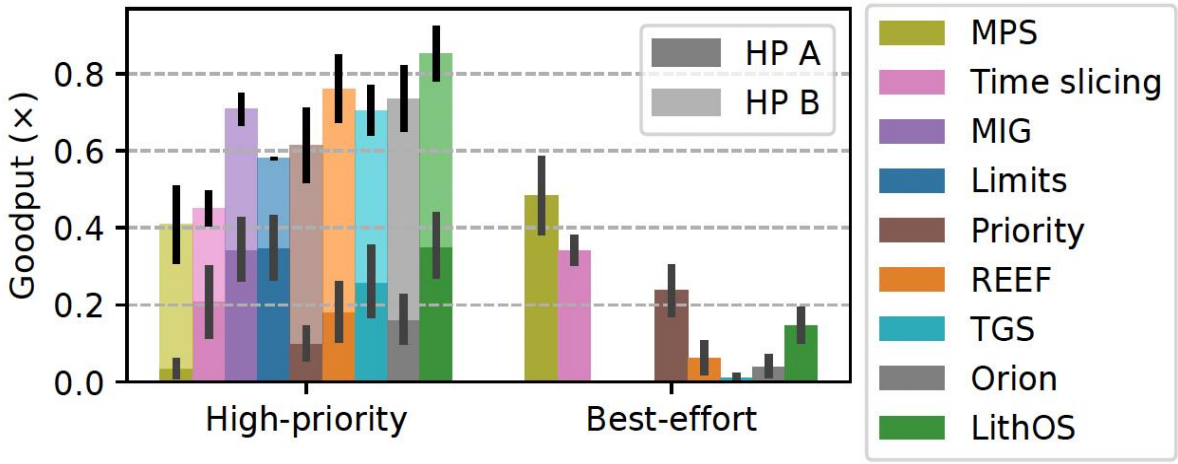
Table 2. *Inference services for inference-only multitenancy.*

use latency constraints from the MLPerf datacenter inference benchmark

Inference-only multitenancy

- 2 high-priority (HP) + 1 best-effort (BE)
  - HP-A latency-oriented SLO (ResNet and RetinaNet)
  - HP-B throughput-oriented SLO (Llama 3, GPT-J, and BERT)
  - BE (Llama 3, GPT-J, and BERT)





Inference-only multitenancy

- P99 latency

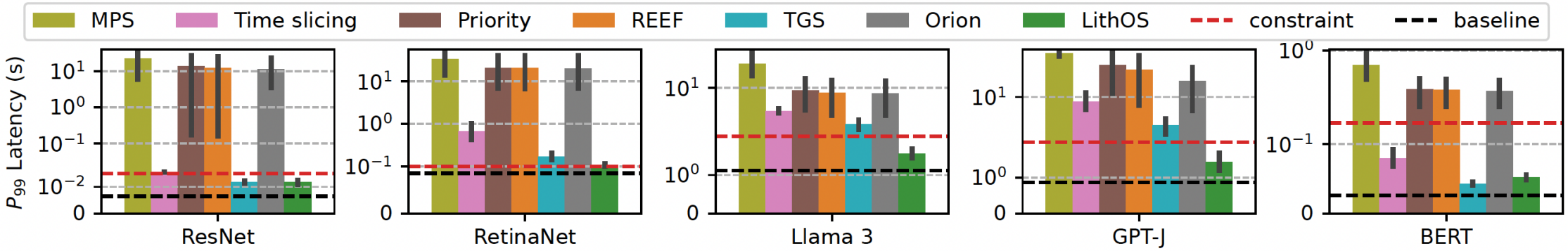
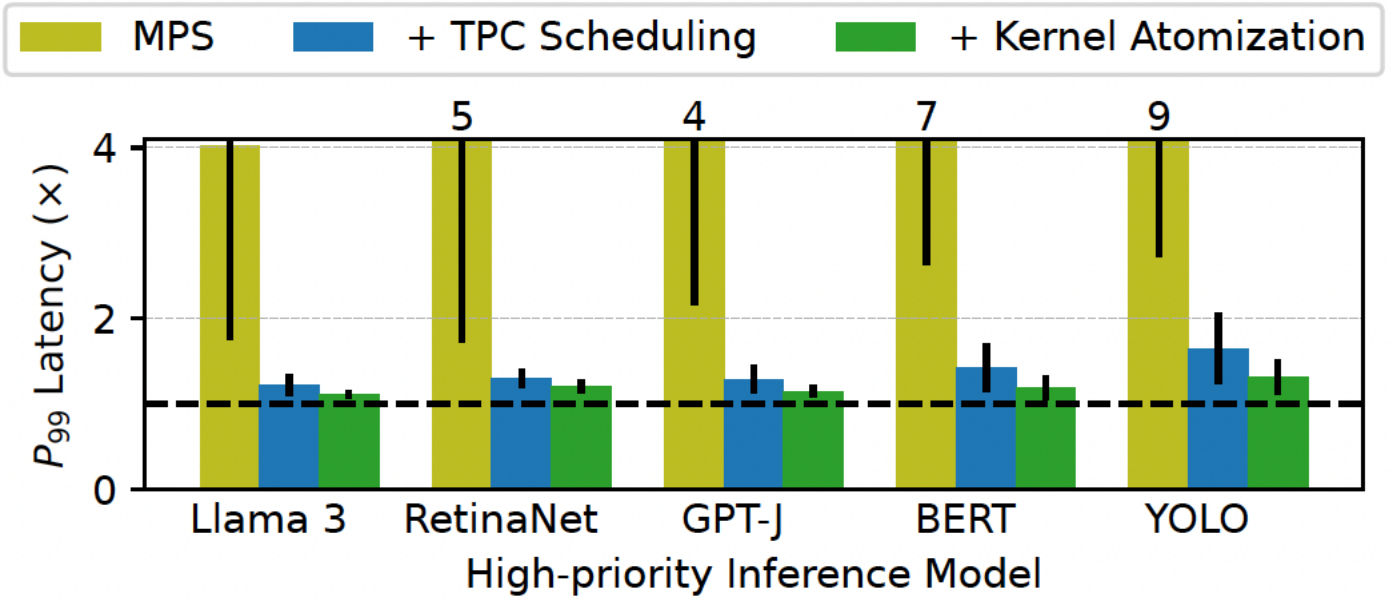


Figure 16. *Inference stacking multitenancy: HP A tail latencies by model.*

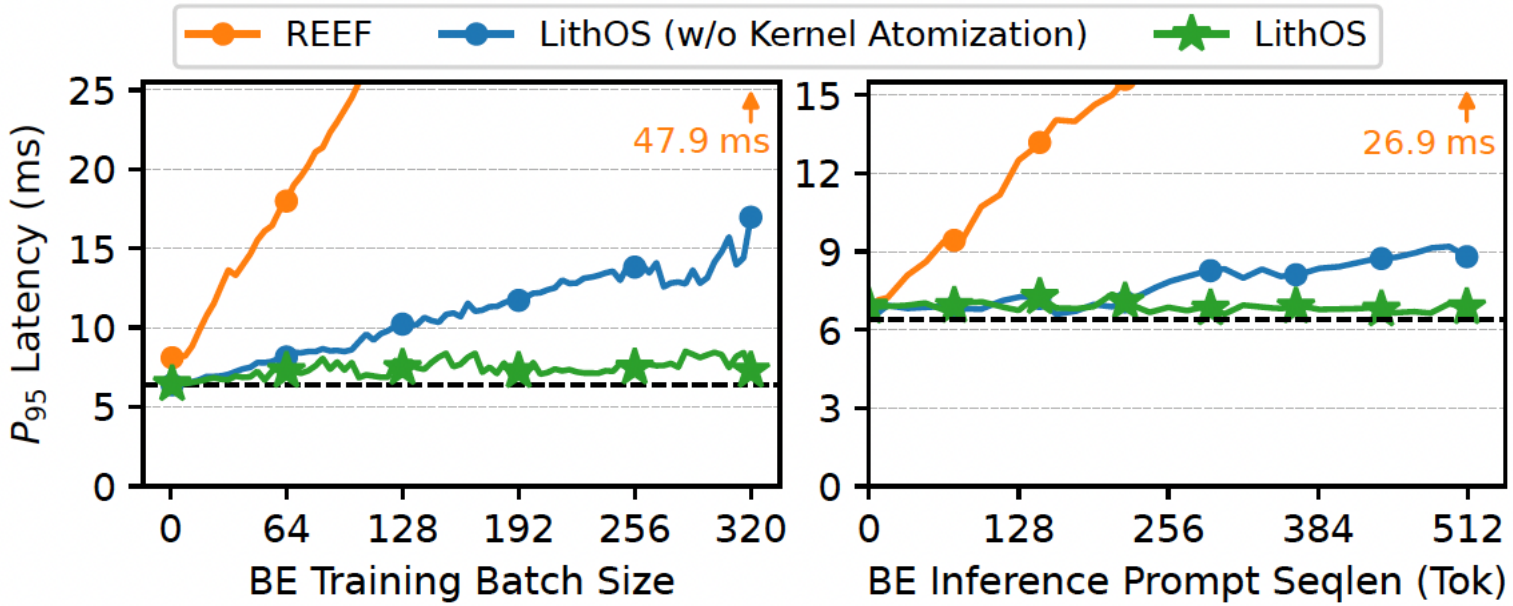
Multi-tenancy Breakdown

- Hybrid Inference/Training Multitenancy



Kernel Atomization

- Vary (a) the batch size of the BE training job and (b) the sequence length of the BE inference



- Kernel atomization mitigates HoL blocking

Thanks