Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing

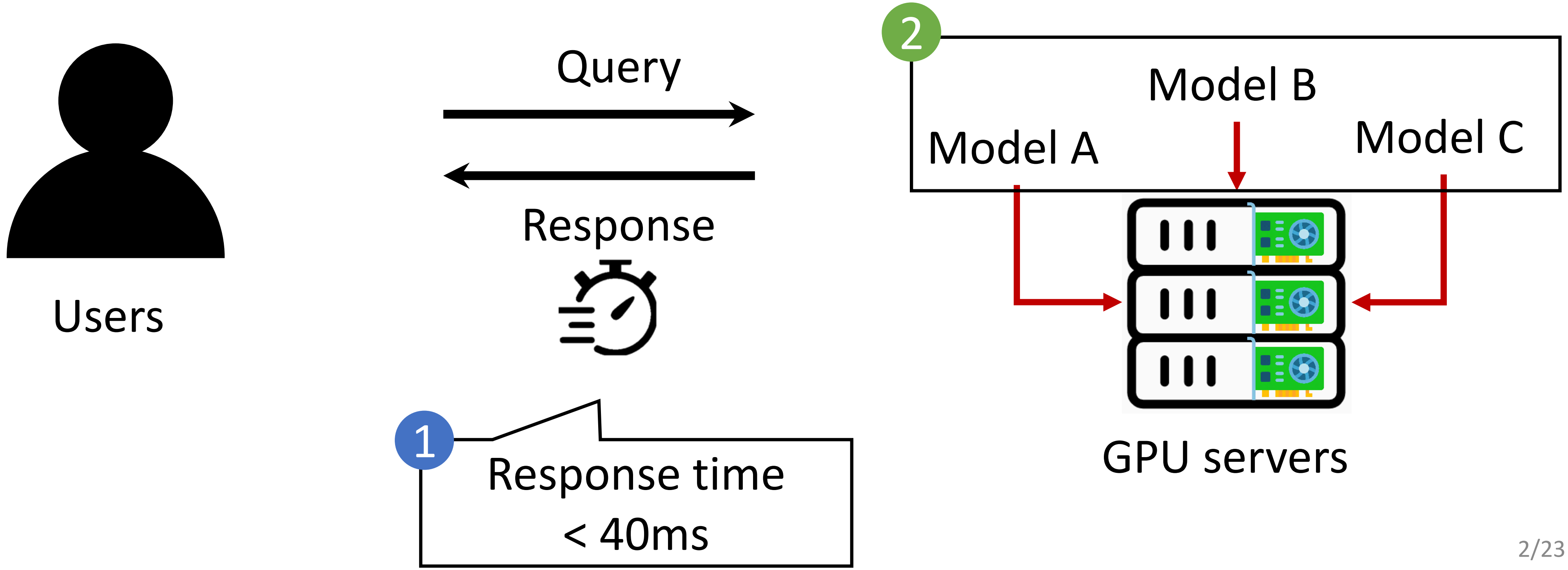
Seungbeom Choi, Sunho Lee, Yeonjae Kim,

Jongse Park, Youngjin Kwon, Jaehyuk Huh

Machine Learning (ML) Inference in GPUs

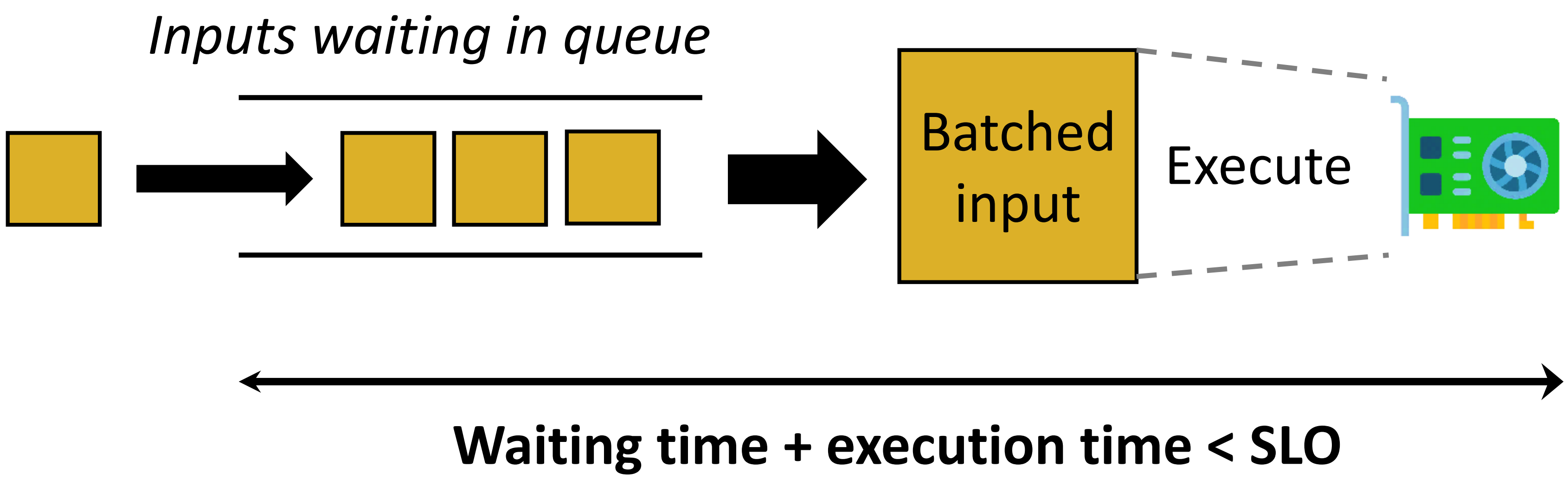
- GPUs are widely adopted as inference accelerator
- Following **requirements** must be satisfied:

1 Serve queries in a bounded time, *service-level objective* (SLO)
2 Serve multiple-heterogeneous ML models



Prior Approach: Batching

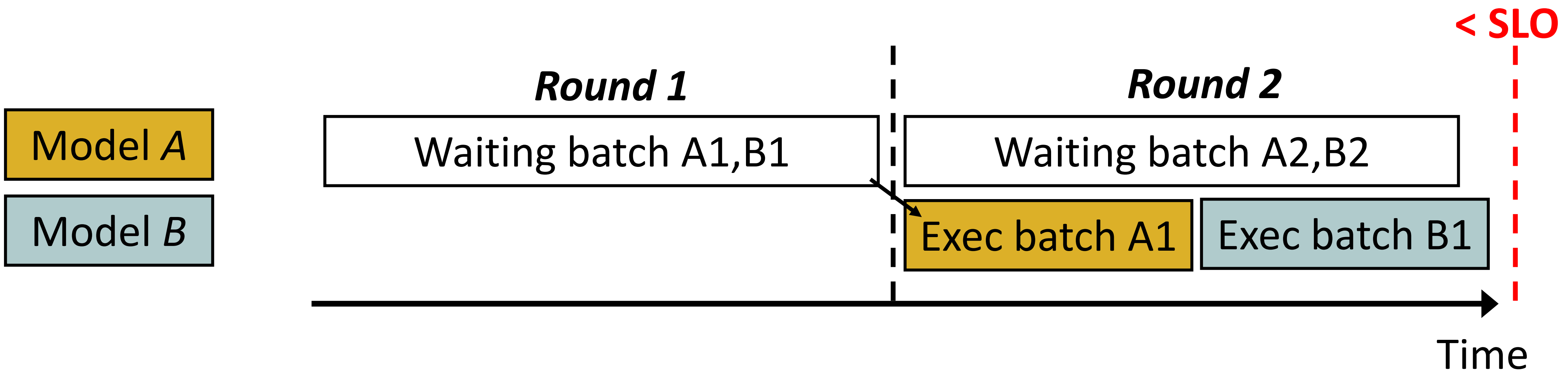
- **Batching:** Merge inputs to a single large input [1], [2], [3]
  - Improves throughput and utilization of GPU
  - **Batch size could not be huge due to SLO**



[1] Clipper [ATC'17]
[2] Clockwork [OSDI'20]
[3] Nexus [SOSP'19]

Prior Approach: Time-Sharing

- **Time-sharing:** Round-based interleaved execution of batches [1]
  - Increase utilization by reducing idle time on GPU
  - **Guarantee 2 rounds < SLO**



[1] Nexus [SOSP'19]

Problem with prior approaches

Batching and time-sharing inference,
underutilize GPUs

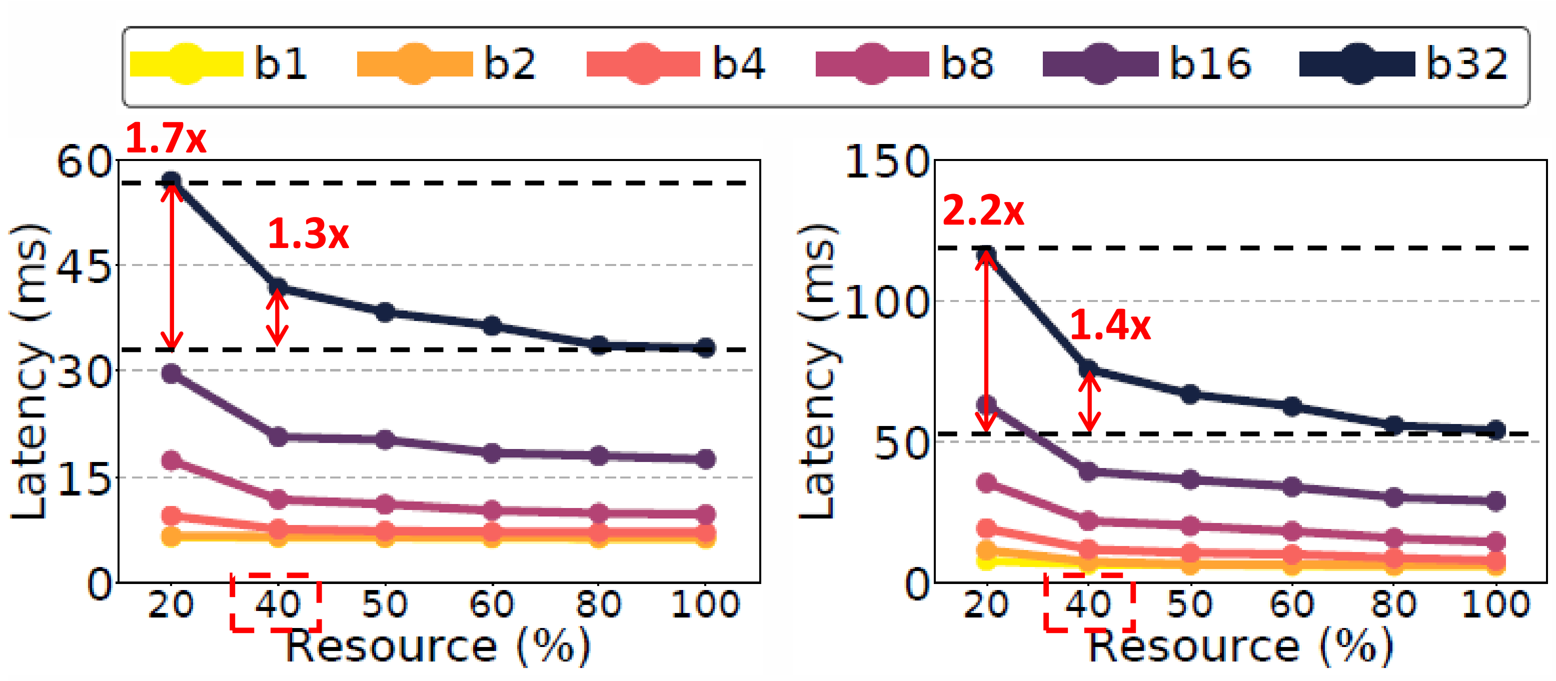
Prior Approach: Time-Sharing

- **Time-sharing:** Round-based interleaved execution of batches [1]
  - Increas…
  - **Guarantee 2 rounds < SLO**

[1] Nexus [SOSP'19]

Underutilized Resources

- Measured latency vs. computing resources w/ varying batch size



(a) GoogLeNet (b) ResNet50

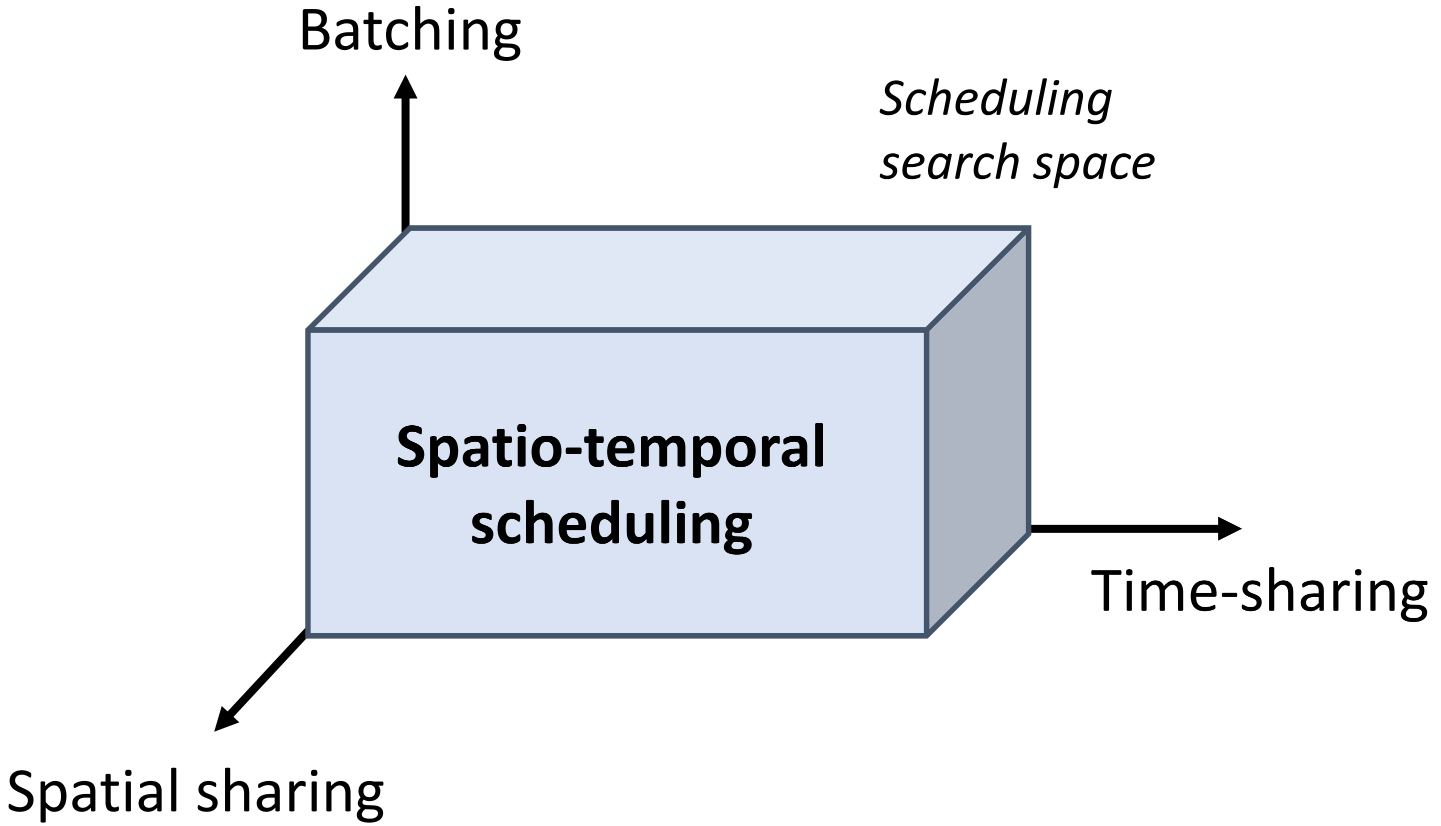
Diminishing return **beyond 40%**

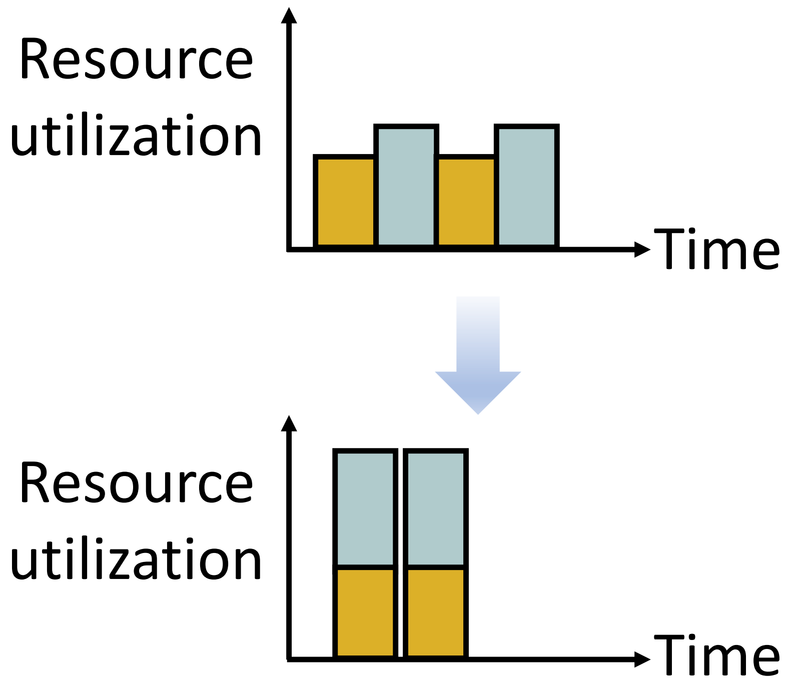
Little improvement in **smaller batch** sizes

Opportunities for improving performance with better resource utilization

New Opportunity: Spatio-temporal Scheduling

- **Spatio-temporal scheduling**:
  - Schedule tasks with batching, time-sharing, **and spatial sharing**

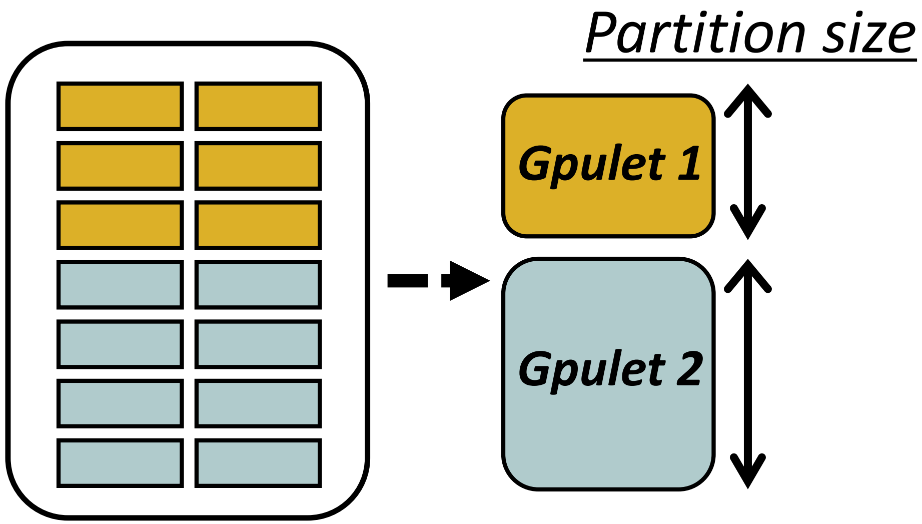




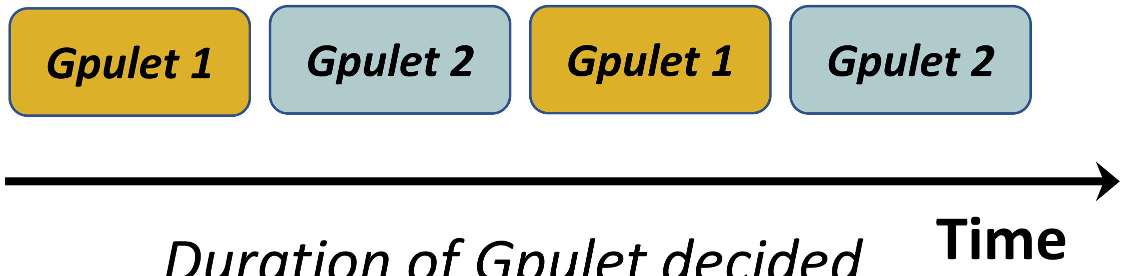
Better utilization → Improved throughput

New Abstraction: Gpulet

- Need an abstraction of spatial/temporal resource
- **Gpulet**: A share of spatial/temporal partition of GPU resource



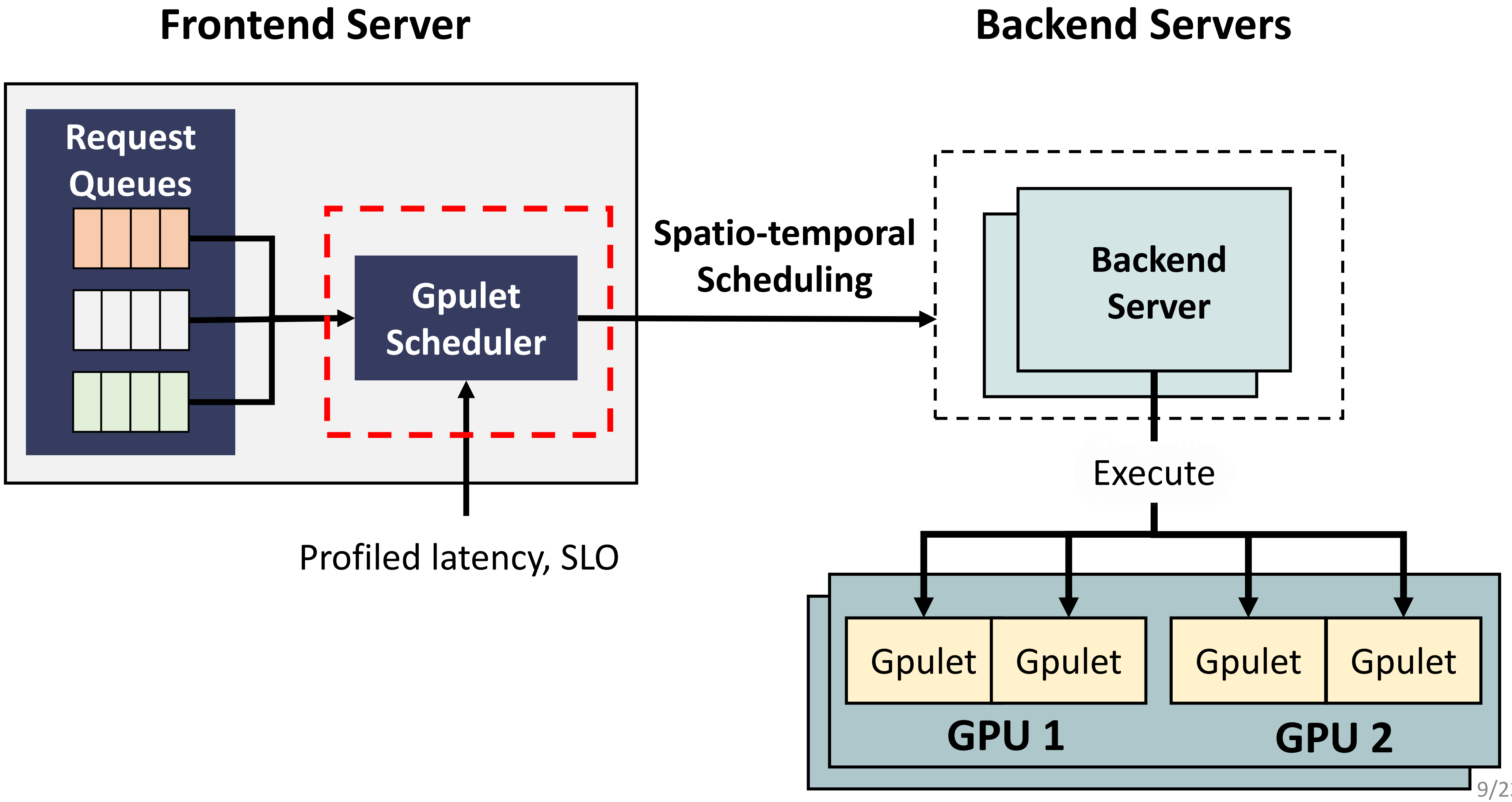
Reserved computational resource



Duration of Gpulet decided by squishy bin-packing [1]

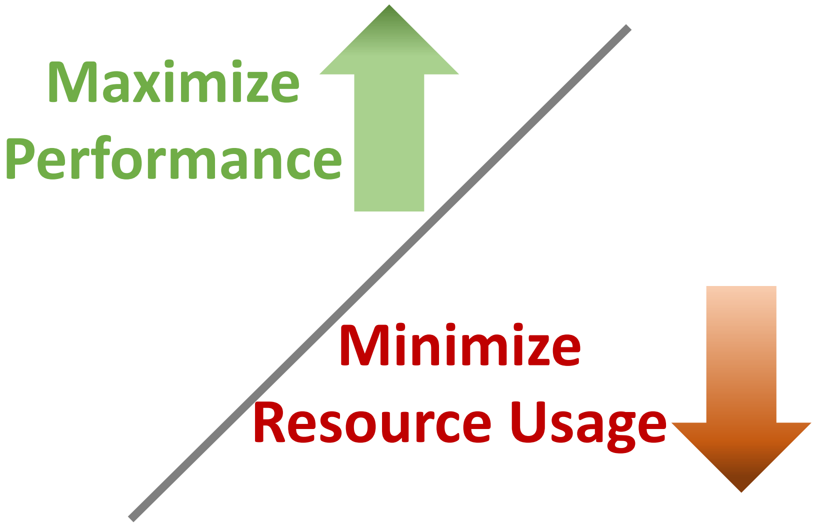
[1] Nexus [SOSP'19]

Overview of Gpulet Scheduling Framework

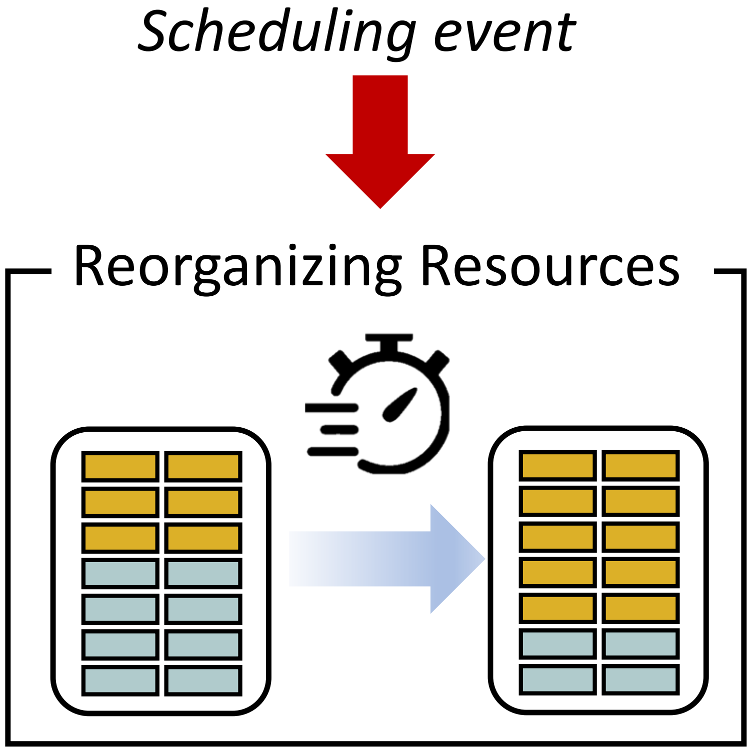


Design Overview of Gpulet Scheduler

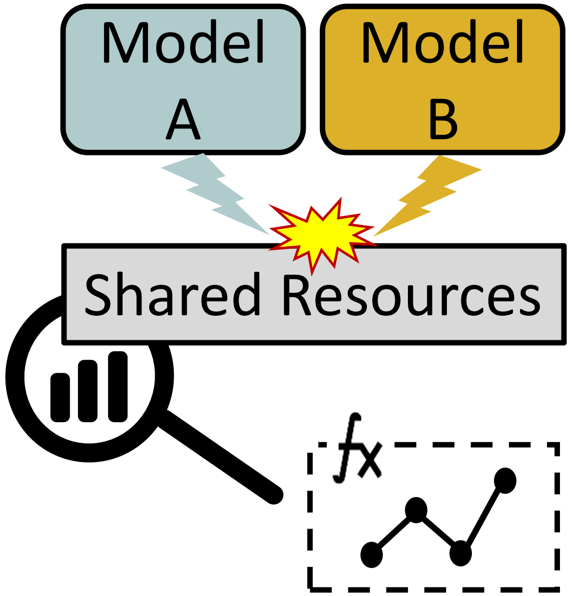
Cost-effective scheduling



Dynamic reorganization



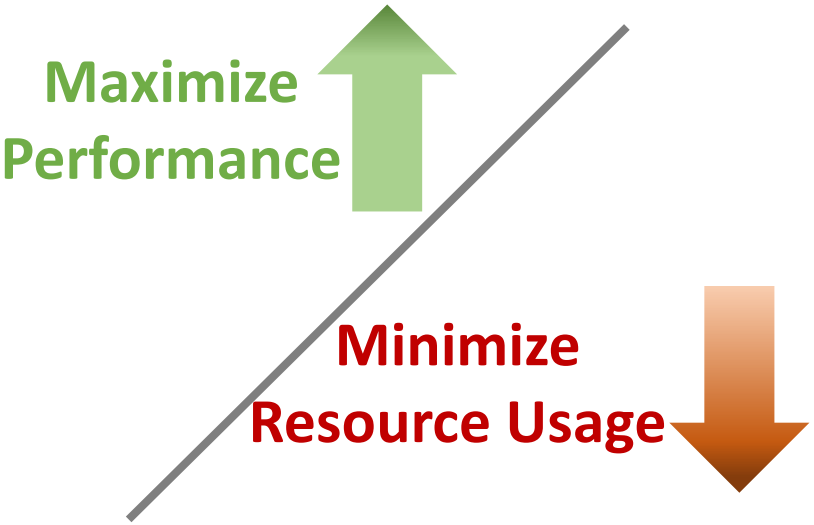
Interference prediction



Please refer to the paper!

Design Overview of Gpulet Scheduler

Cost-effective scheduling



Dynamic reorganization

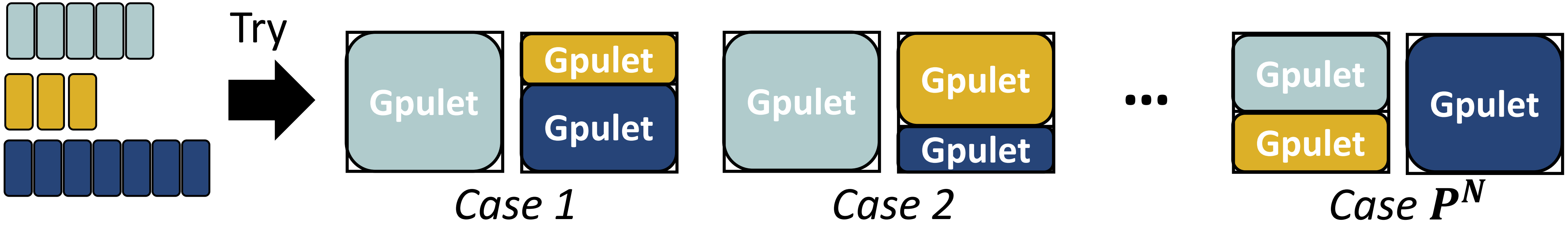
Scheduling event

Interference prediction

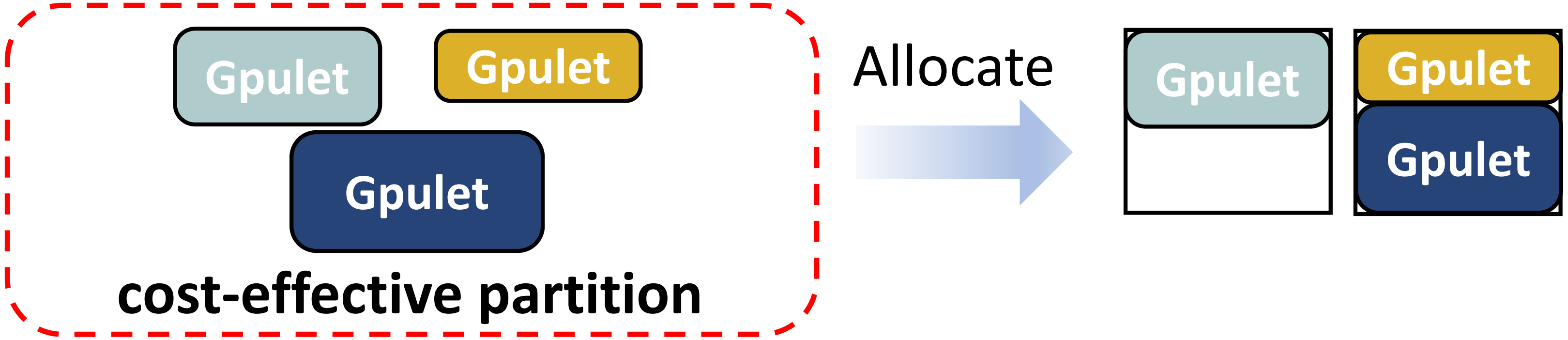
Please refer to the paper!

Scheduling Gpulets

- **Challenge:** Large search space for spatial scheduling
  - *P* spatial partitioning choices for *N* GPUs: P^N **cases to search exhaustively**



- **Main idea**: Allocate partitions to GPUs <u>**incrementally**</u>



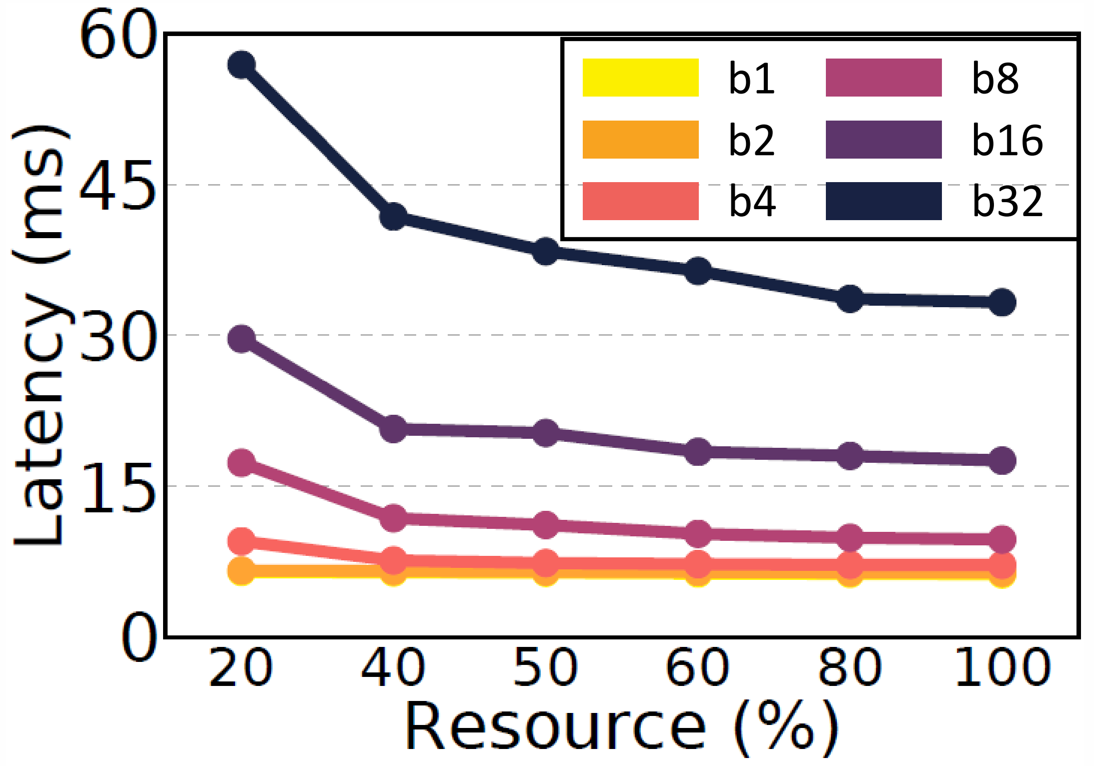
Efficient than exhaustive search

Q) How to find cost-effective partitions?

Cost-effective Partition

- **Cost-effective**: Maximum performance / resource
  - Cost-effective partition size (resource) = starting point of diminishing return
  - Performance is **not** linearly proportional to resource
  - Example) GoogLeNet

Thr



%

Resource
(Partition size)

For each partition size

Step ①
Get maximum batch size *b* *with Latency(b) < SLO*

Step ②
Get throughput
= *b / Latency(b)*

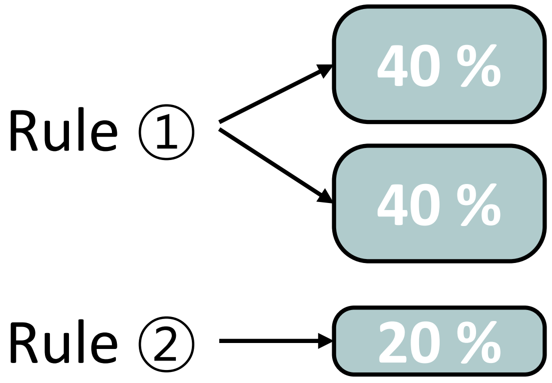
Allocating Partitions to Input Rate

- **Rules** for allocating minimum sum of partitions
  - ① ***As much as many*** **cost-effective partitions** within rate
  - ② **One minimum partition** for remaining rate
- Example: 900 requests per second (rps)

Partition Size	Throughput
*40 %	400 rps
20 %	100 rps

* Cost-effective partition size

Resulting Partition Sizes



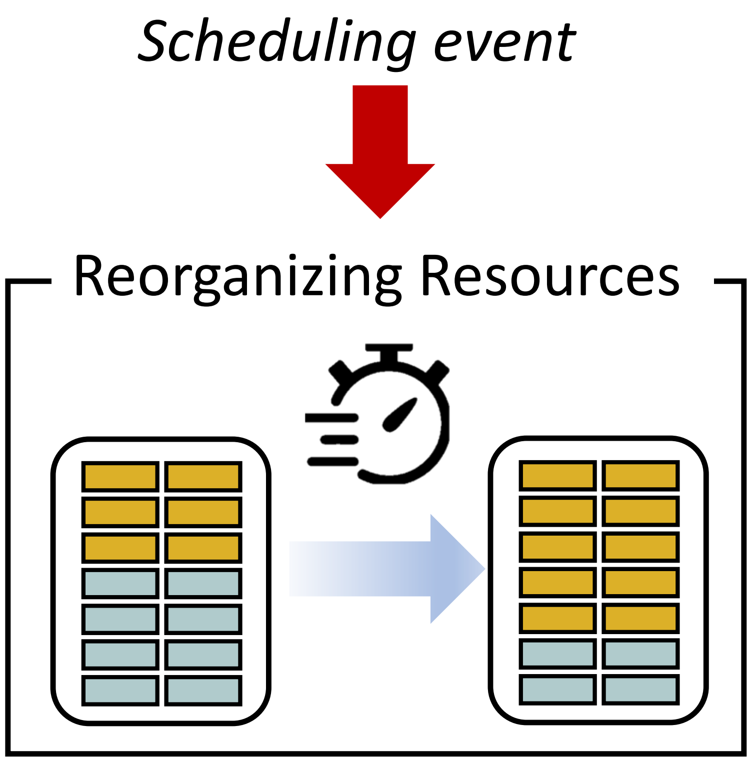
Σ = 900 rps

100 rps still remains!

Design Overview of Gpulet Scheduler

Cost-effective scheduling

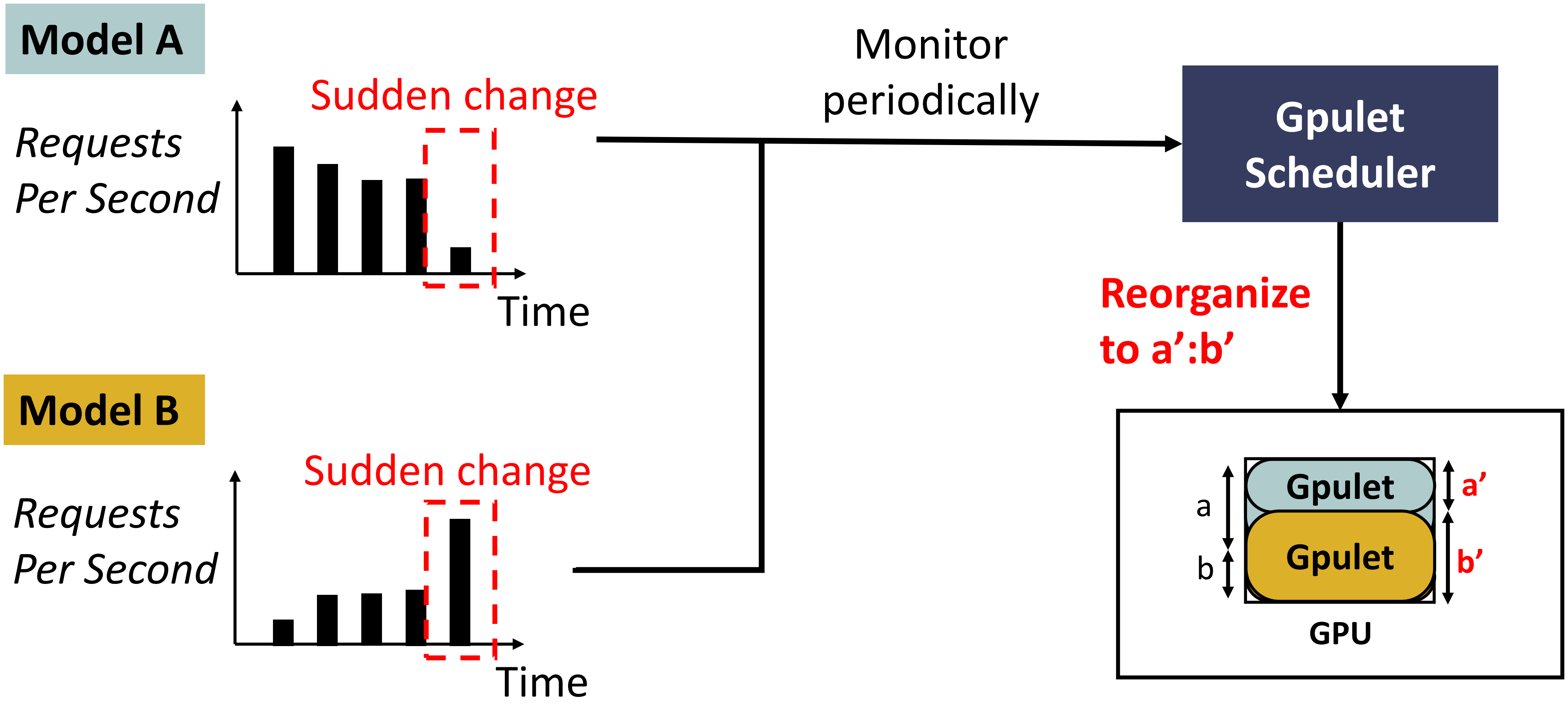
Dynamic reorganization



Interference prediction

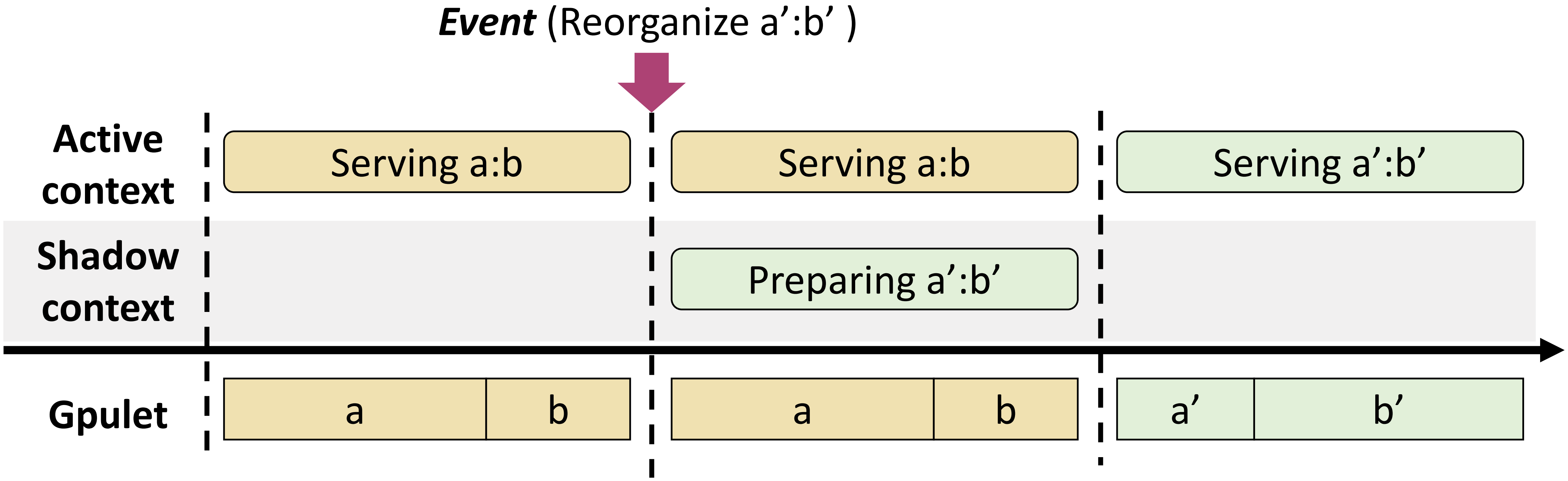
Please refer to the paper!

Scheduling Event for Reorganization



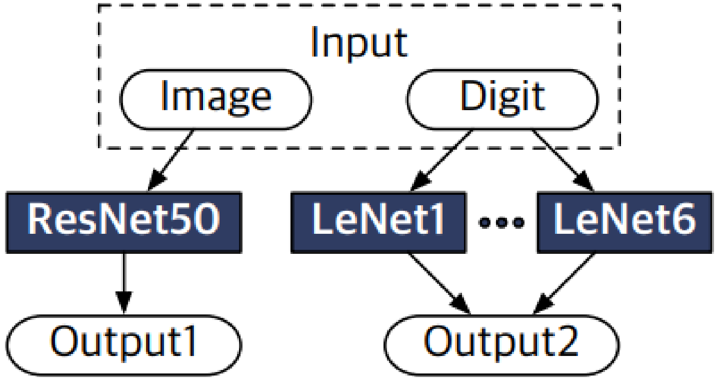
Dynamic Partition Reorganization

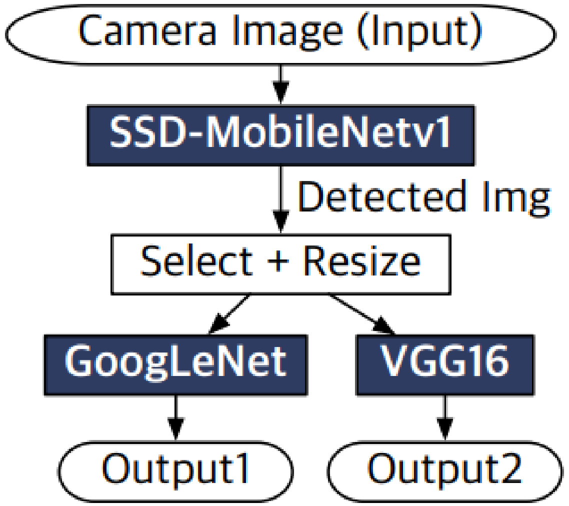
- **Challenge:** Large overhead exists for preparing a new Gpulet
  - Overhead: *Loading kernels, warming up models*
- **Solution:** Hide overhead by **shadowing** in the background



Evaluated Benchmarks

- Two multi-model applications
  - game: image/digit recognition
  - traffic: camera footage analysis





- Five multi-model scenarios
  - Composed 5 group of models by memory footprint size

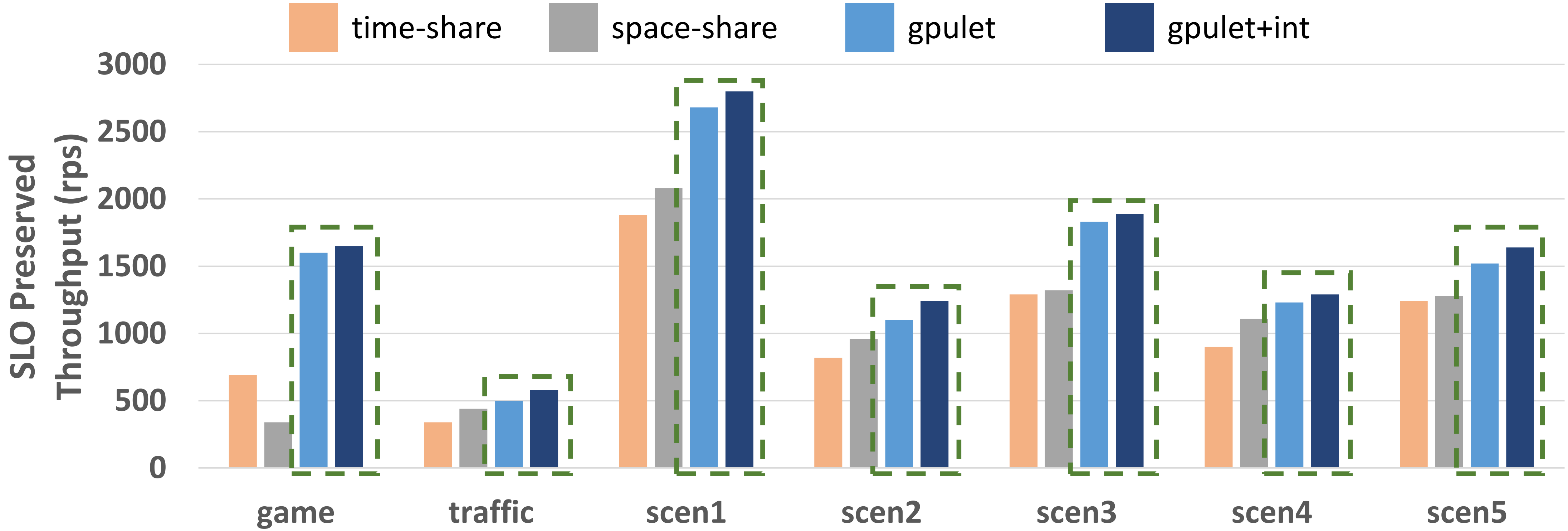
Name	Number of models by size (small : medium : large)
scen1	2 : 2 : 0
scen2	0 : 1 : 1
scen3	1 : 1 : 1
scen4	1 : 2 : 0
scen5	1 : 2: 1

Evaluation Methodology

- Environment:
  - 2 multi-GPU servers, *each with 2x* RTX 2080Ti
  - Connected with 10G Ethernet network
- Metric: SLO preserved throughput
  - Maximum throughput w/ SLO violate rate < 1%
- Schedulers:

Name	Time Sharing	Spatial Sharing	Interference Prediction
time-share	YES	NO	NO
space-share	NO	Greedy	NO
gpulet	YES	Cost-effective	NO
gpulet + int	**YES**	**Cost-effective**	**YES**

SLO Preserved Throughput Comparison

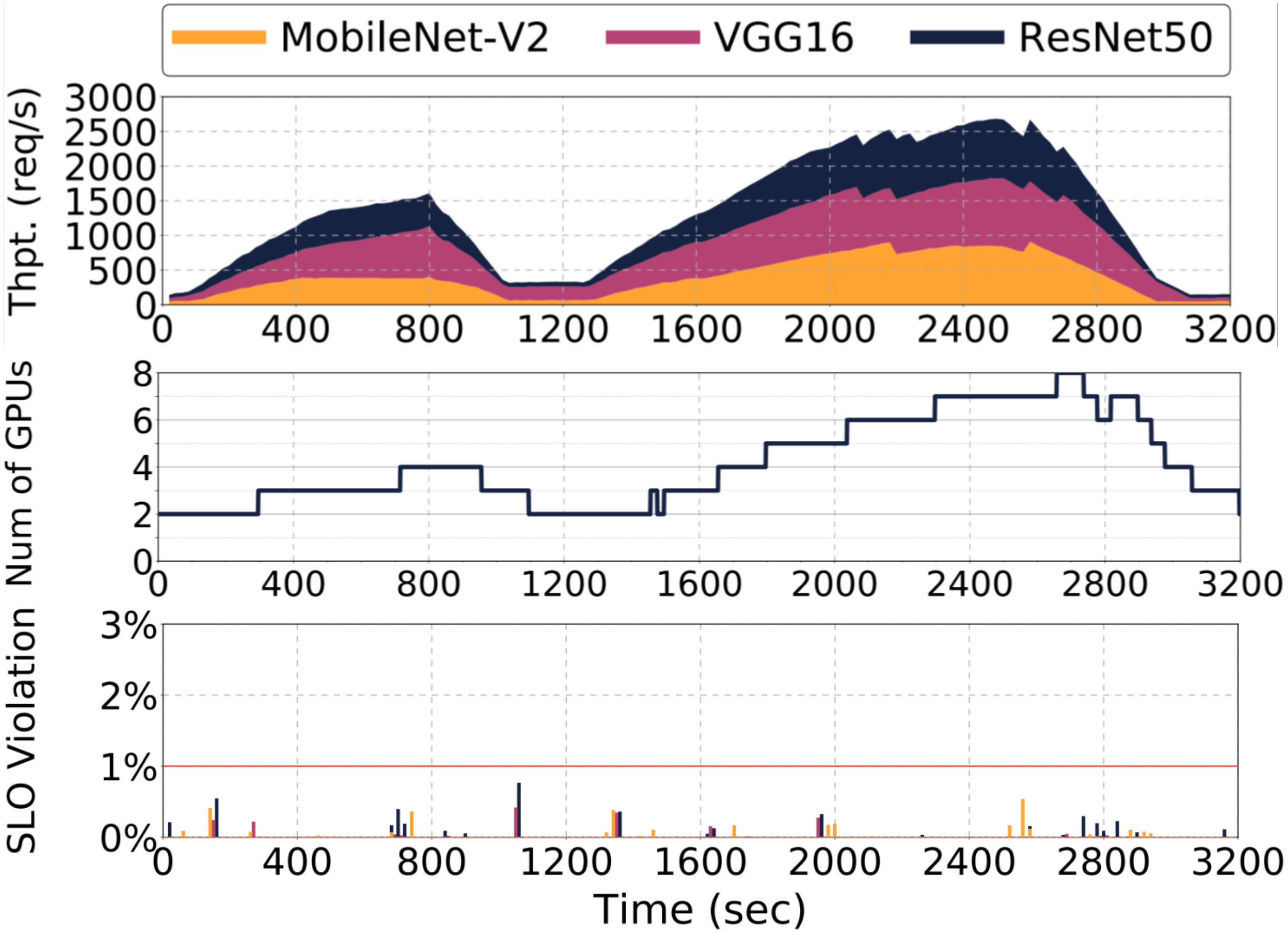


Best performance when **both time and spatial** scheduling enabled throughput increased by an average **61.7%** than *time-share*

Considering interference boosts throughput by **7.5%**

Evaluation of Scaling GPUs

- Extended environment: 4x RTX 2080Ti ➡ ***8x RTX 2080Ti***



➡ **Throughput**

GPUs successfully scaled with SLO violation rate < 1%

➡ **SLO violation rate**

More Results in the Paper

- Comparison of spatial partitioning vs. non-partitioning
- Comparison of proposed scheduler vs. ideal scheduler
- Evaluation of meeting SLO without interference prediction

Conclusion

- ML inference performance enhanced by spatio-temporal scheduling

- Spatio-temporal scheduling further enhanced by
  - **Minimizing wasted resources** with spatial sharing
  - **Scaling resources efficiently** by hiding overheads for preparing resources
  - **Predicting interference effect** when scheduling

- Outperformed time-sharing scheduler’s throughput by **61.7%**