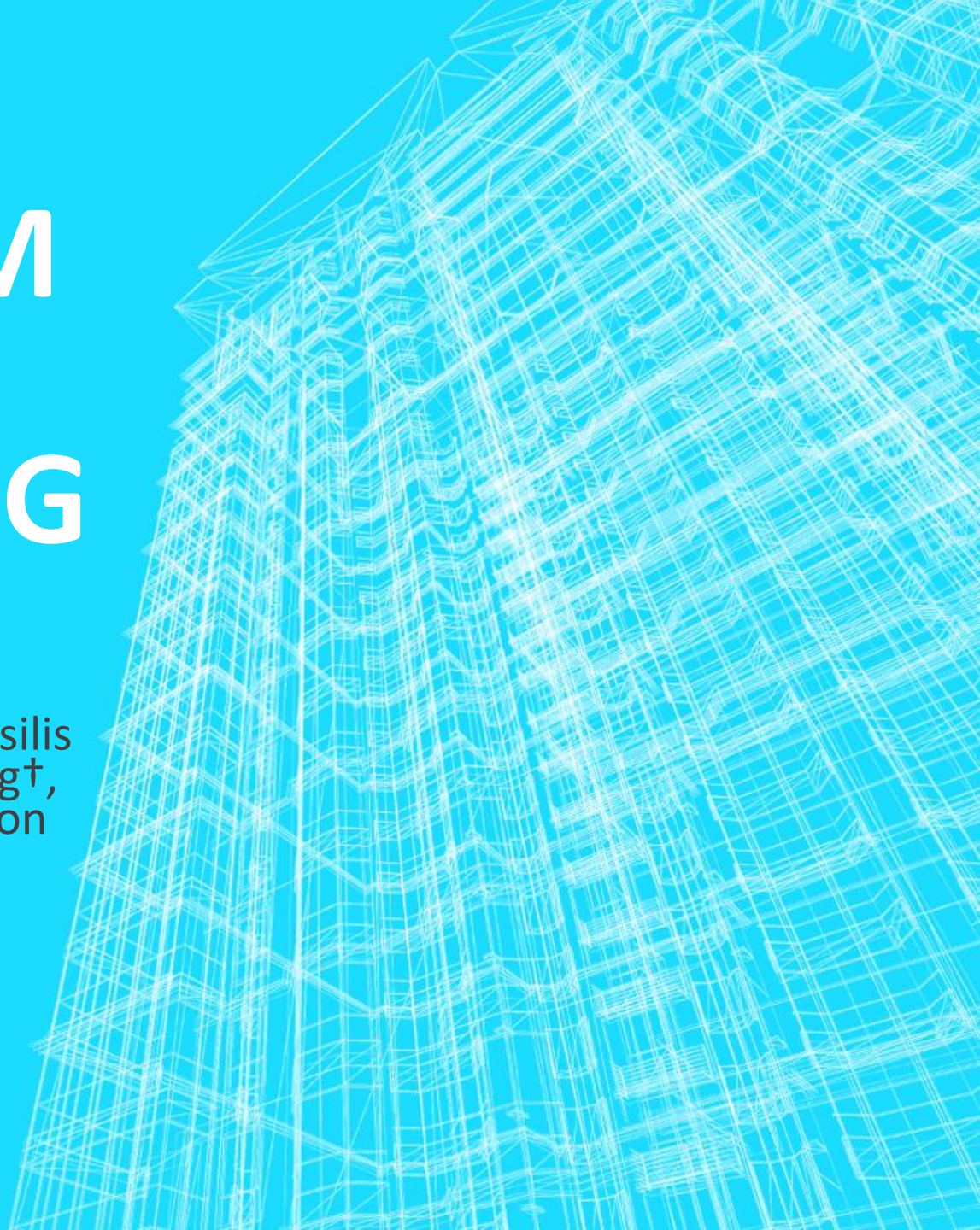
LITHOS - AN OPERATING SYSTEM FOR EFFICIENT MACHINE LEARNING ON GPUS

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

Presented by: Dimitris Nikolaidis



GPUS ARE UNDERUTILIZED

- GPUs are incredibly powerful, but they are being severely underutilized in modern data centers, wasting massive amounts of money and electricity.

Meta

Device: **30%**

SM: **15%**

Train: **40%**

Microsoft 52%

(Philly ATC 2019)

Alibaba 10%

ByteDance 26%

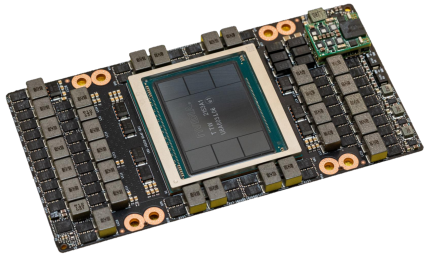
(MuxFlow, arXiv)

SOLUTION: AN OS FOR GPUS

The 1950s Mainframe: Early computers were insanely expensive, highly scarce, and single-tenant (only one person/program could use them at a time).

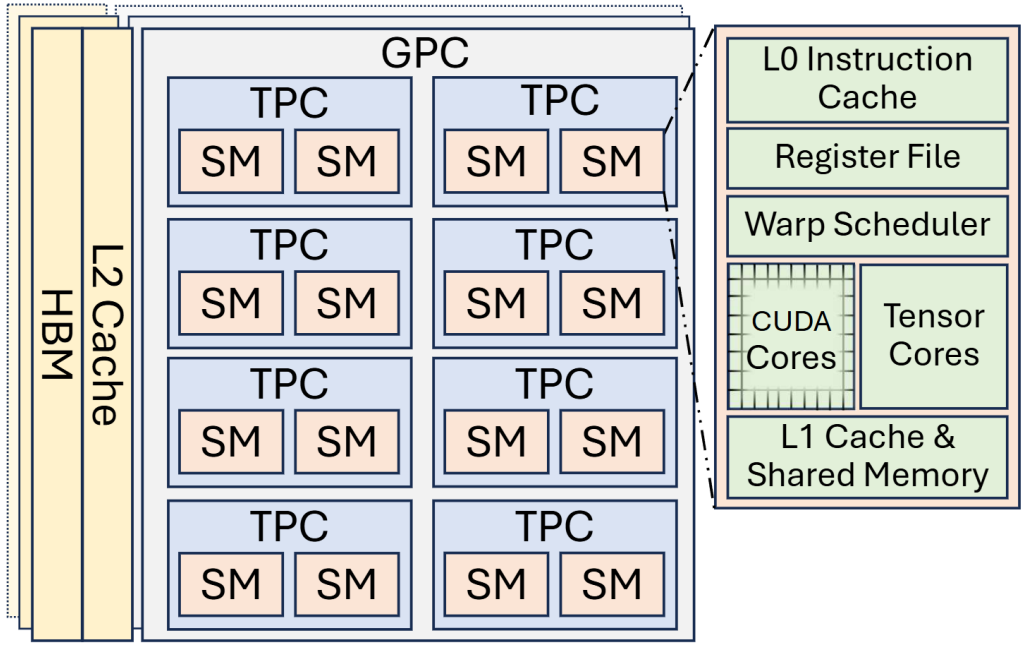
The CPU Solution: Operating Systems were invented to solve this exact problem. They introduced virtual memory, process scheduling, and context switching so multiple users could share the hardware transparently

Modern AI GPUs are scarce, cost tens of thousands of dollars, and are largely treated as single-tenant hardware. We need to bring OS-level management to GPUs.



GPU ARCHITECTURE 101

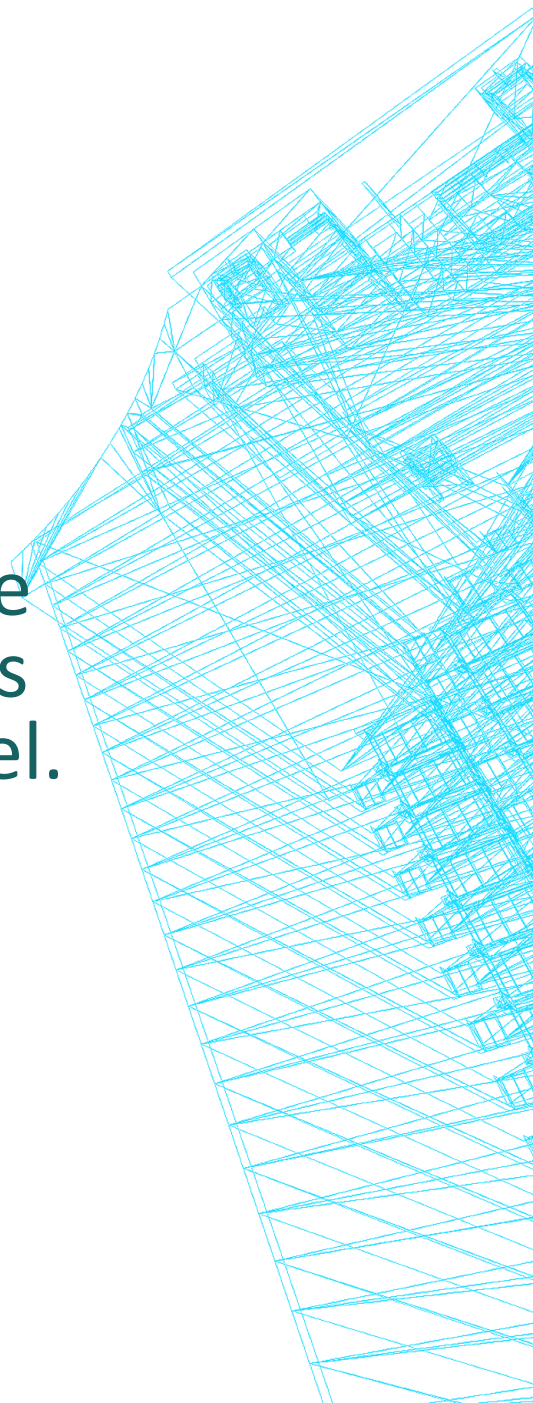
- **GPC (Graphics Processing Cluster):** The largest building block. A modern GPU might have 8 of these.
- **TPC (Texture Processing Cluster):** The next level down. There are typically multiple TPCs per GPC (e.g., 66 total on an H100).
- **SM (Streaming Multiprocessor):** The main compute engine. There are roughly 132 SMs per GPU.
- **CUDA Cores:** The smallest execution units (thousands per GPU).



Current hardware sharing methods partition the GPU at the massive GPC level. LithOS introduces software partitioning at the much finer TPC level.

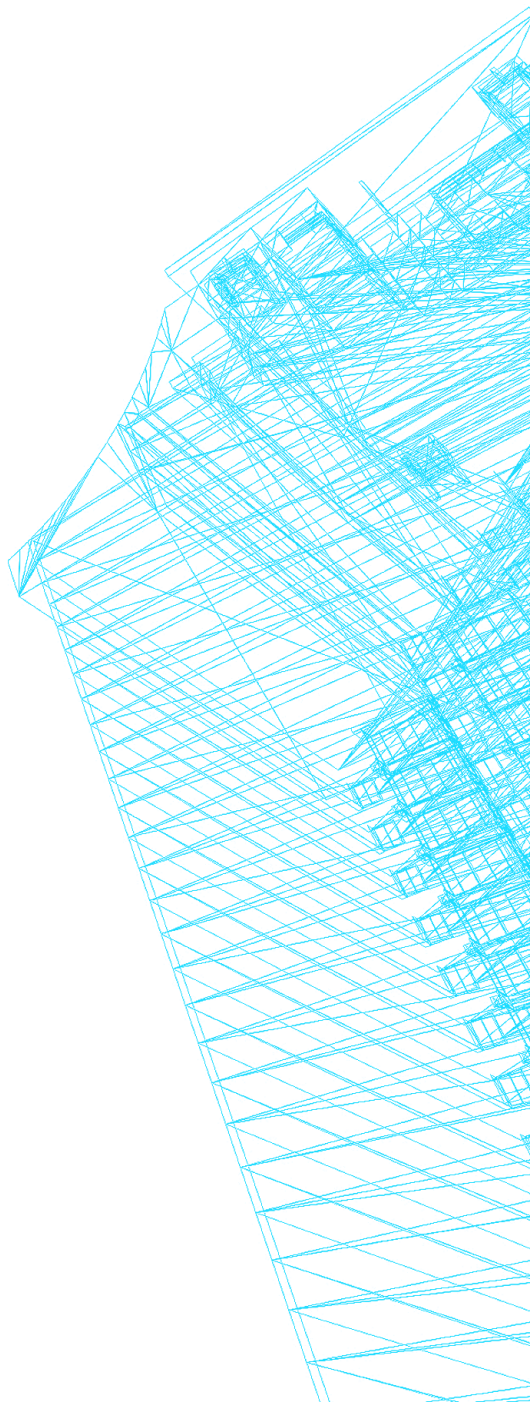

PCI Express 5.0 Host Interface
GigaThread Engine with MIG Control
GPC
TPC
SM
L2 Cache
L2 Cache
Memory Controller
HBM3
High-Speed Hub
NVLink

Current hardware sharing methods partition the GPU at the massive GPC level. LithOS introduces software partitioning at the much finer TPC level.



GPU SHARING IS HARD

CPUs: Sharing happens completely transparently. The OS schedules threads to specific cores and can "preempt" (pause) a running thread instantly if something more important comes along.
GPUs: No native preemption. Once a chunk of work (a "kernel") is sent to the GPU, you cannot pause it. You have no control over where kernels execute on the hardware. Fixing this usually requires invasive, manual modifications to ML frameworks and application code.



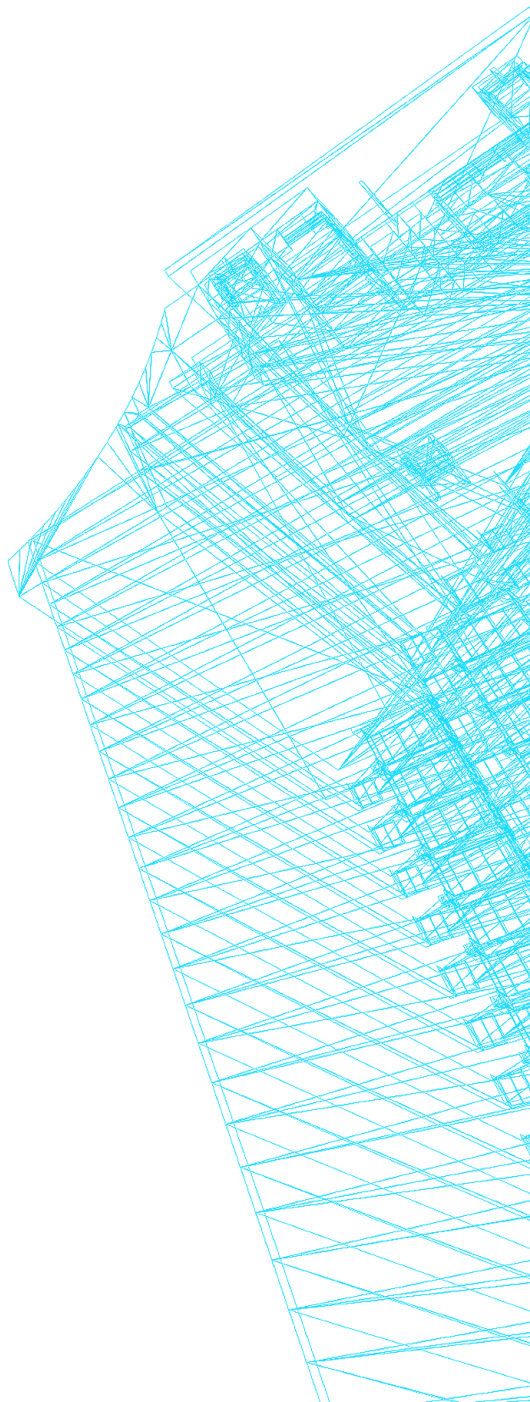
WHY CURRENT GPU SHARING METHODS FAIL

MIG (Multi-Instance GPU): Physically slices the GPU into separate partitions.

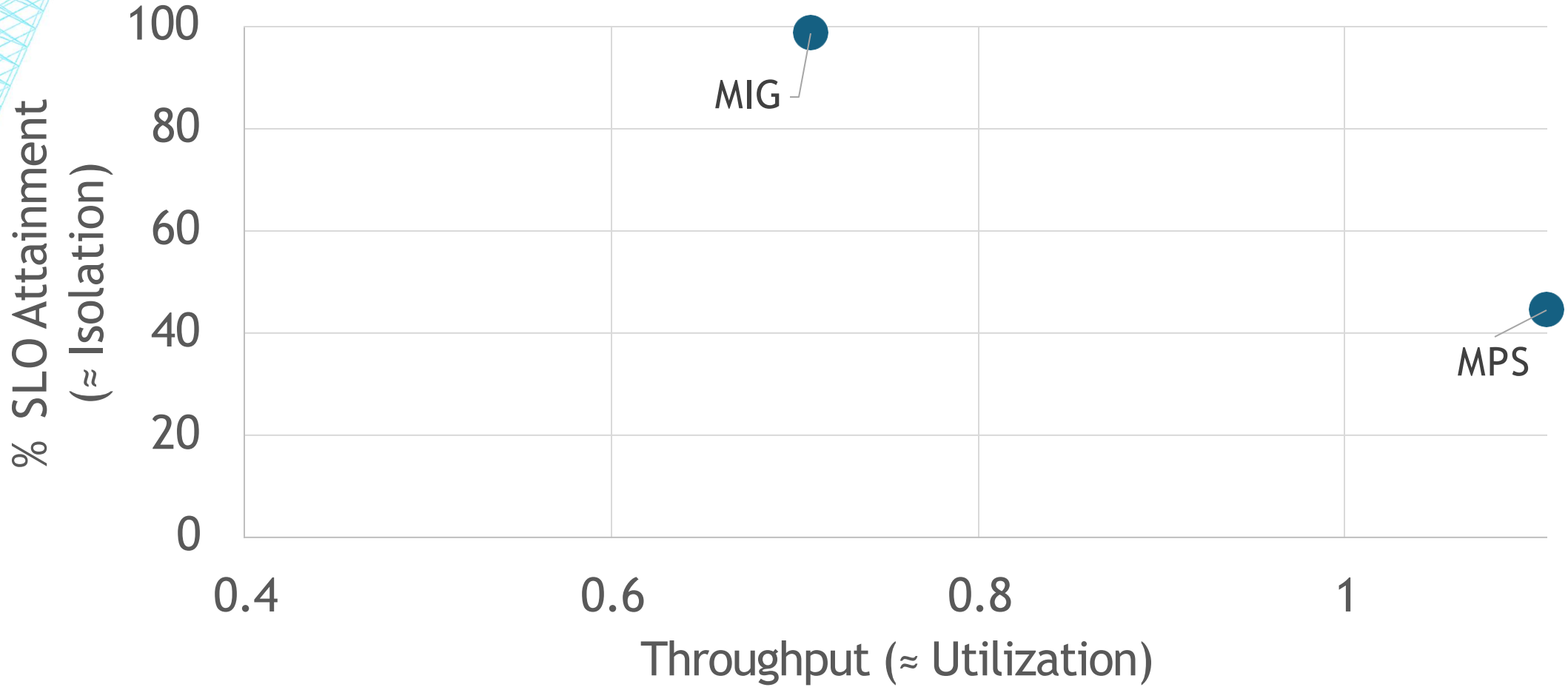
- Perfect isolation (no interference).
- Partitions are coarse-grained and fixed. If an app finishes early, its slice sits empty, leading to wasted capacity.

MPS (Multi-Process Service): Mashes all applications together on the same hardware.

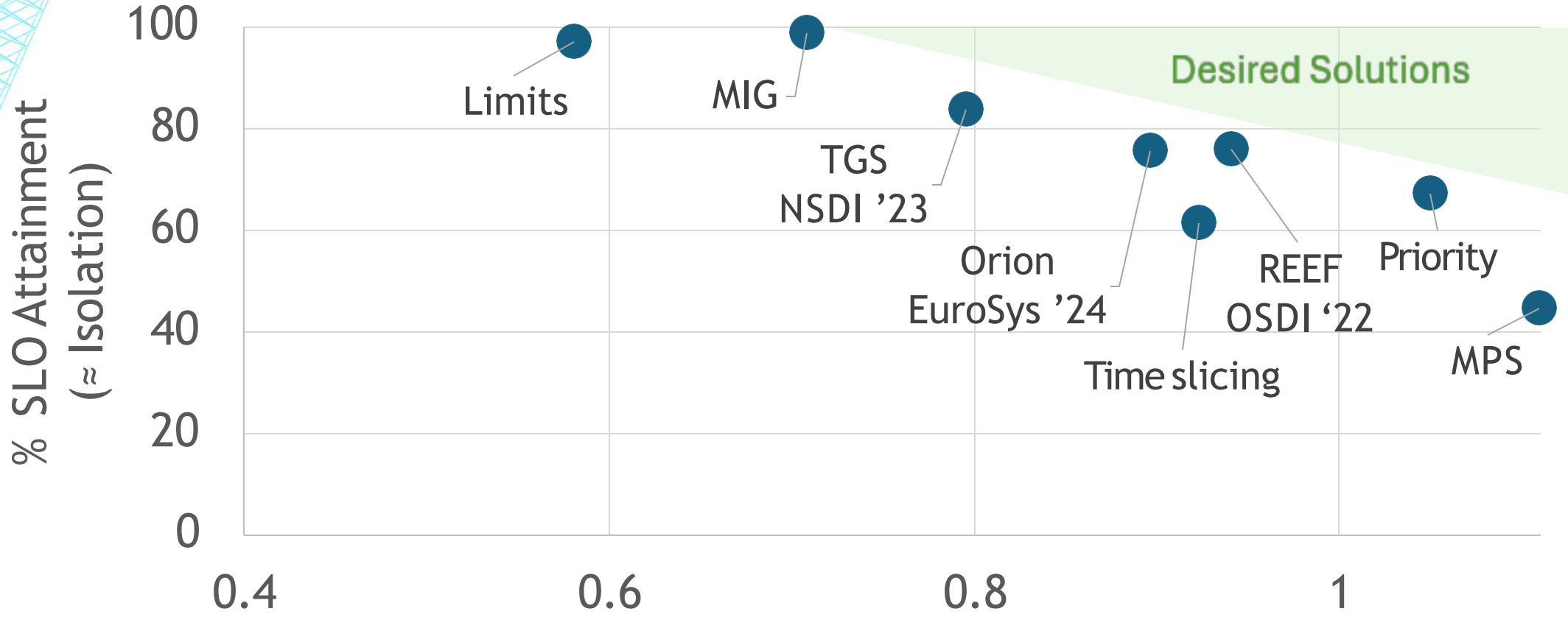
- High utilization and throughput.
- Time-sharing with no preemption causes heavy performance interference. If a big background job runs, a critical user-facing request gets stuck waiting (a massive latency hit)



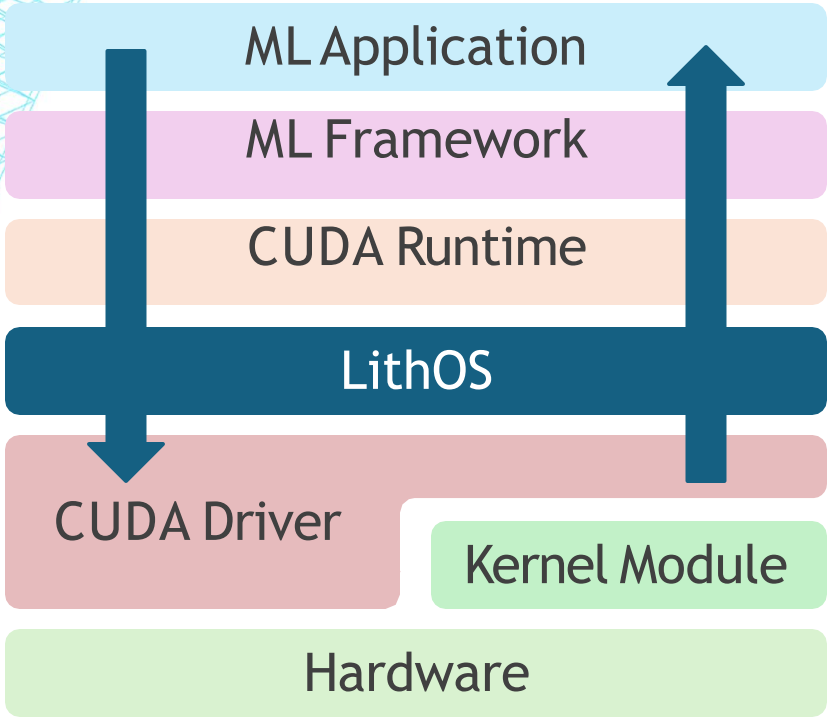
THE LATENCY SLO VS. THROUGHPUT TRADE-OFF



THE LATENCY SLO VS. THROUGHPUT TRADE-OFF



LITHOS ARCHITECTURE



TPC Masking

Prelude Kernels

Advantages

- No need to modify application software (CUDA C++, PTX, etc.)
- Pulls kernel scheduling logic out of closed-source software and hardware
- Easy to port across hardware and driver versions

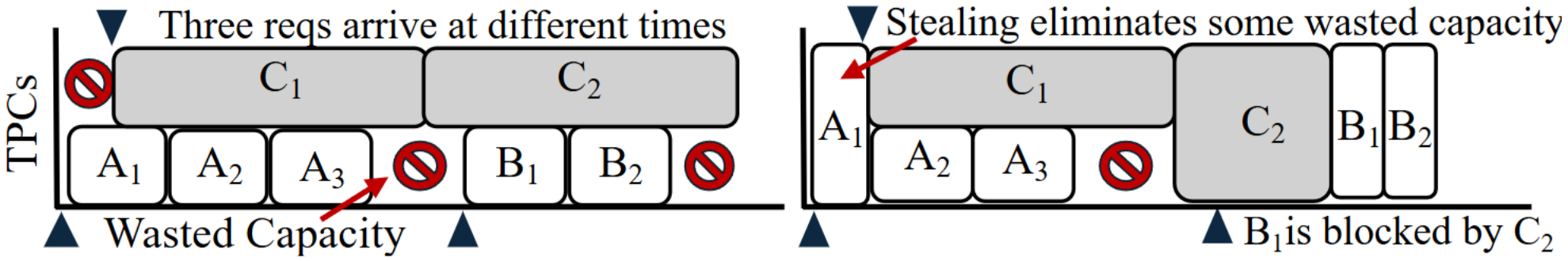
Vision

- Software from LithOS down is part of an integrated, open OS

TPC SCHEDULING & STEALING

Instead of dividing the GPU into massive, rigid chunks (like MIG), LithOS schedules workloads at the fine-grained TPC level.

TPC Stealing: If App A (e.g., LLaMA) finishes its immediate tasks and its TPCs go idle, LithOS dynamically lets App B "steal" those TPCs to keep the GPU fully utilized

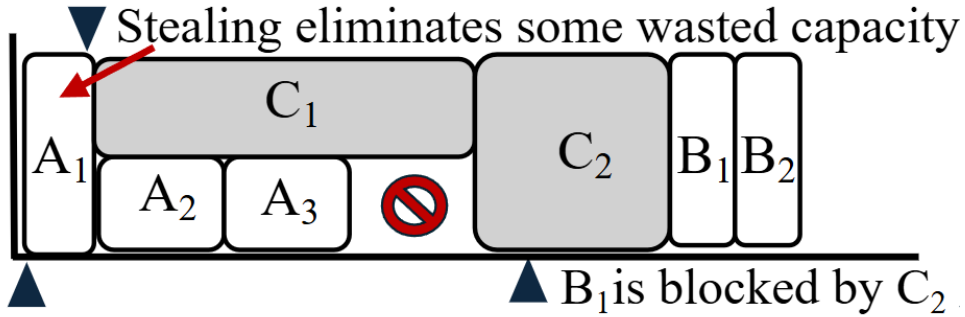


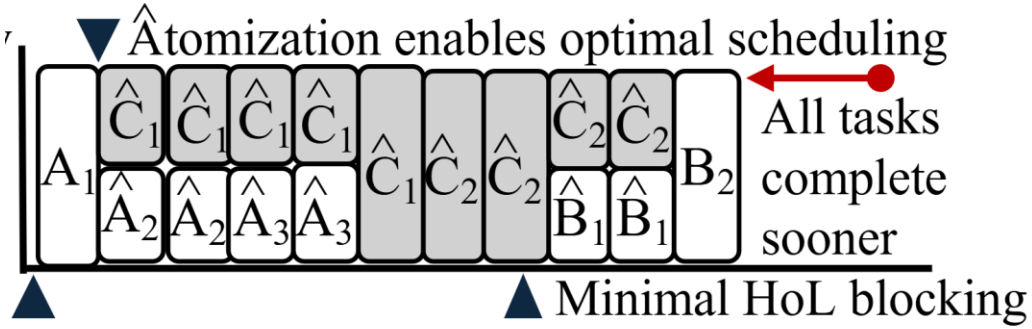
KERNEL ATOMIZATION

Stealing is not enough. Because GPUs lack preemption, if App B launches a massive, long-running kernel on those stolen TPCs, and App A suddenly wakes up with an urgent request, App A is blocked. This causes a major latency hit.

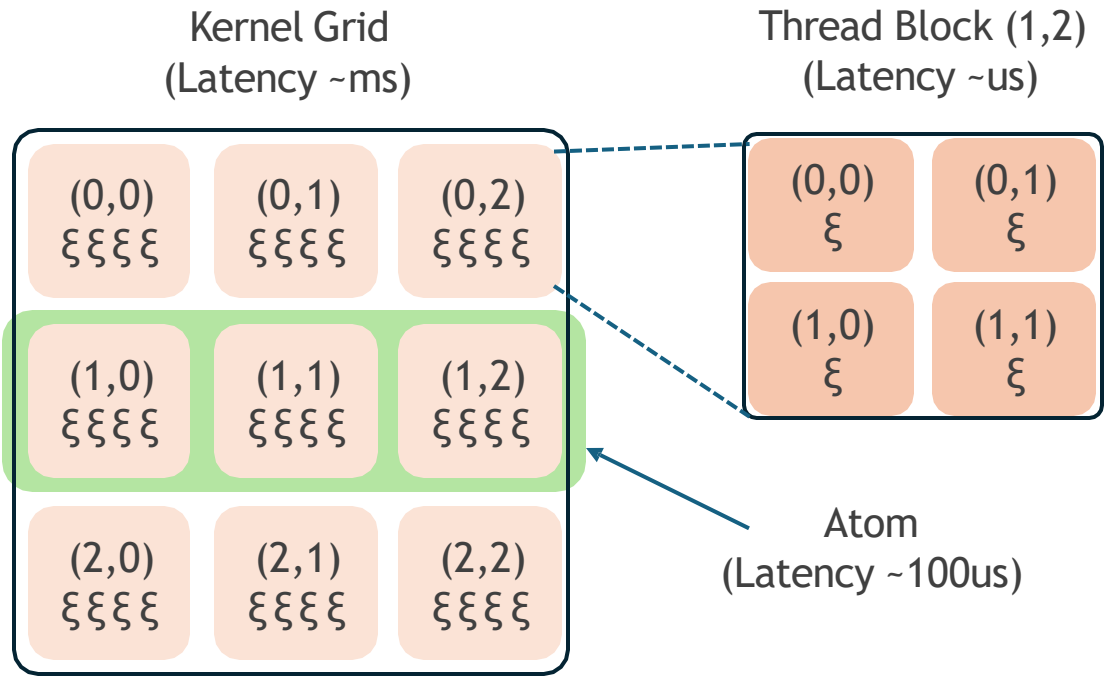
KERNEL ATOMIZATION

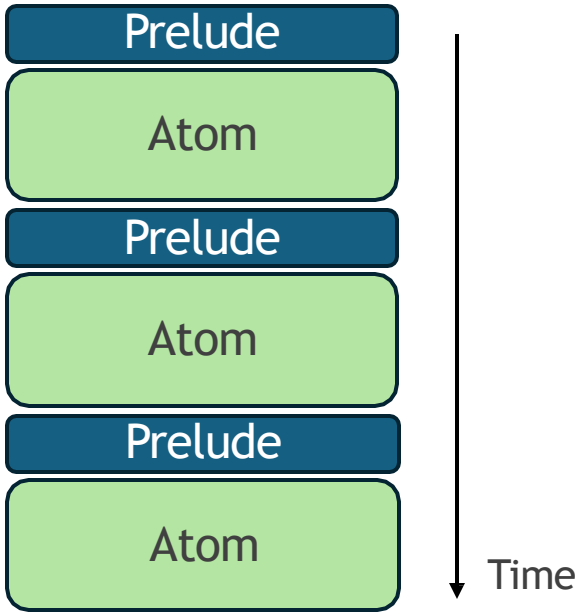
A standard GPU "Kernel Grid" takes several milliseconds to run. It is made up of smaller "Thread Blocks" which only take microseconds to run. LithOS intercepts the application's massive Kernel Grid and automatically slices it up into smaller chunks called **"Atoms "** (groupings of thread blocks). It does this using a "Prelude" wrapper, meaning it doesn't need to rewrite the application's actual math.





KERNEL ATOMIZATION

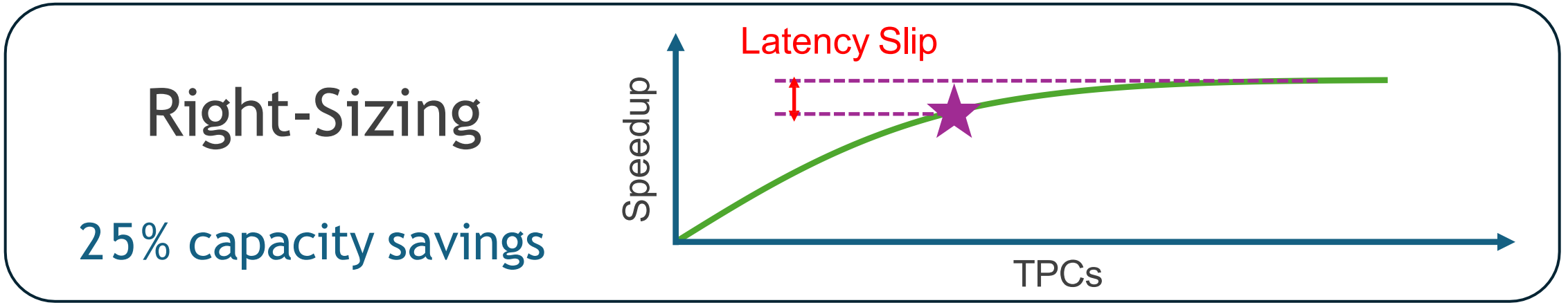




Thread-block-level scheduling in software!

HARDWARE RIGHT-SIZING

Not all ML operations need the entire GPU to run efficiently. LithOS calculates exactly how many TPCs a specific kernel actually needs. By allocating only what is necessary, LithOS yields an average of 25% capacity savings.

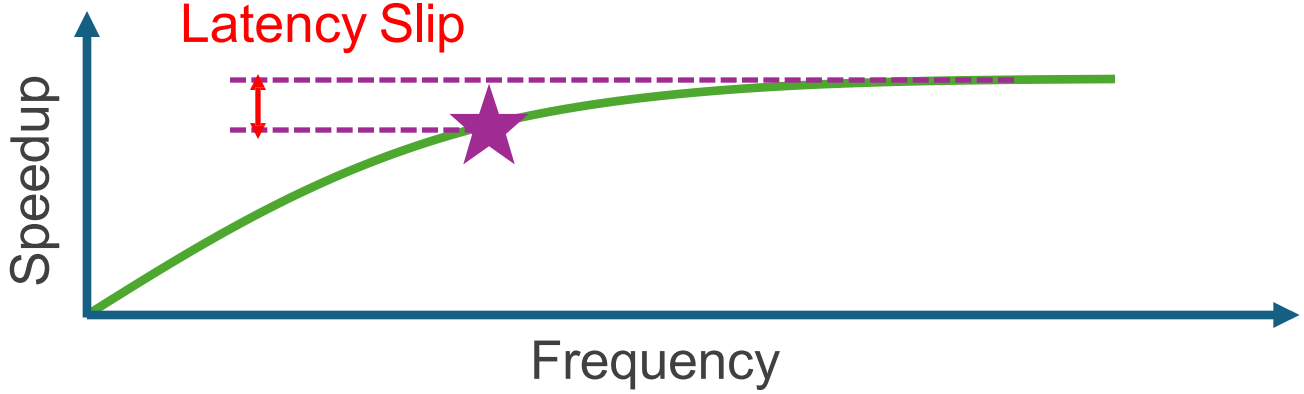


DYNAMIC VOLTAGE FREQUENCY SCALING

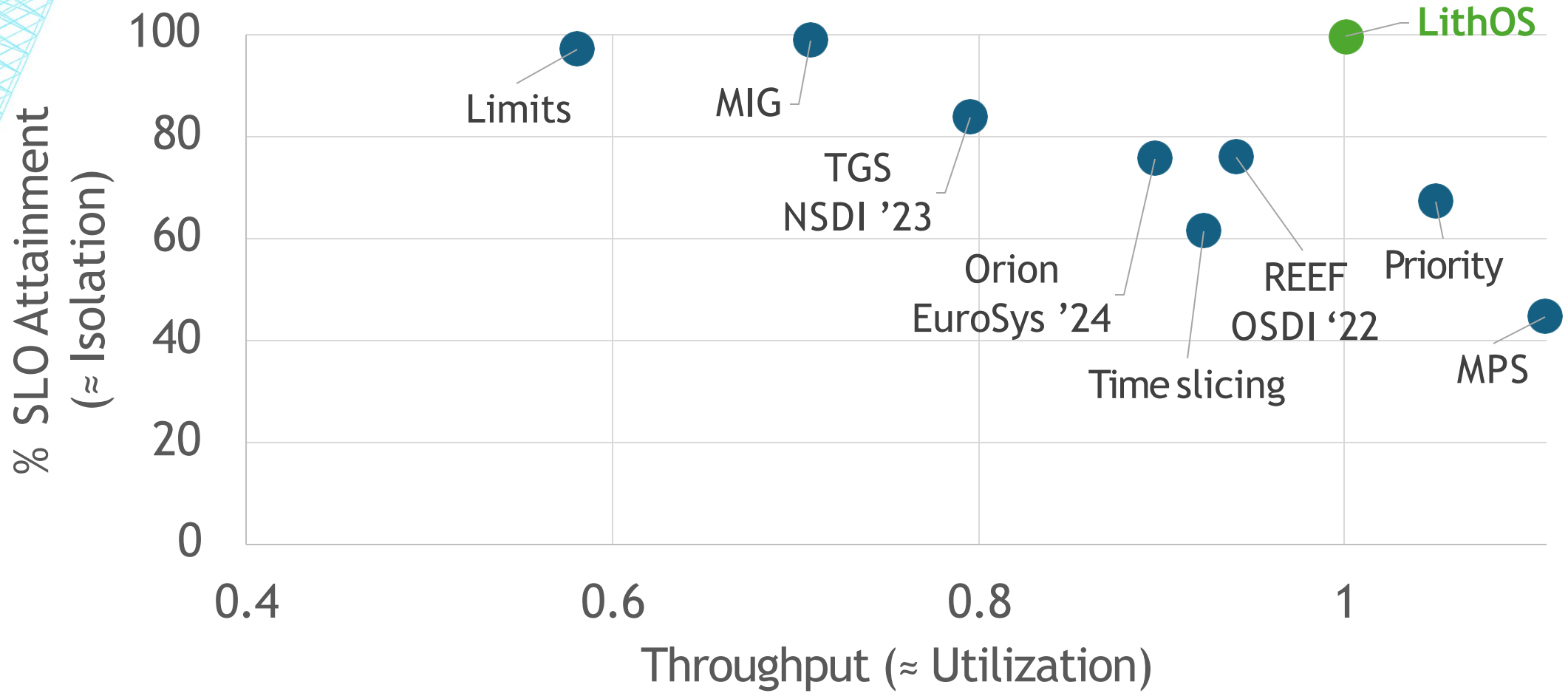
LithOS analyzes kernels in real-time to see if they are compute-bound or memory-bound. It dynamically slows down the GPU clock speed for kernels that don't benefit from high frequencies, yielding 25% energy savings.

Dynamic Voltage Frequency Scaling

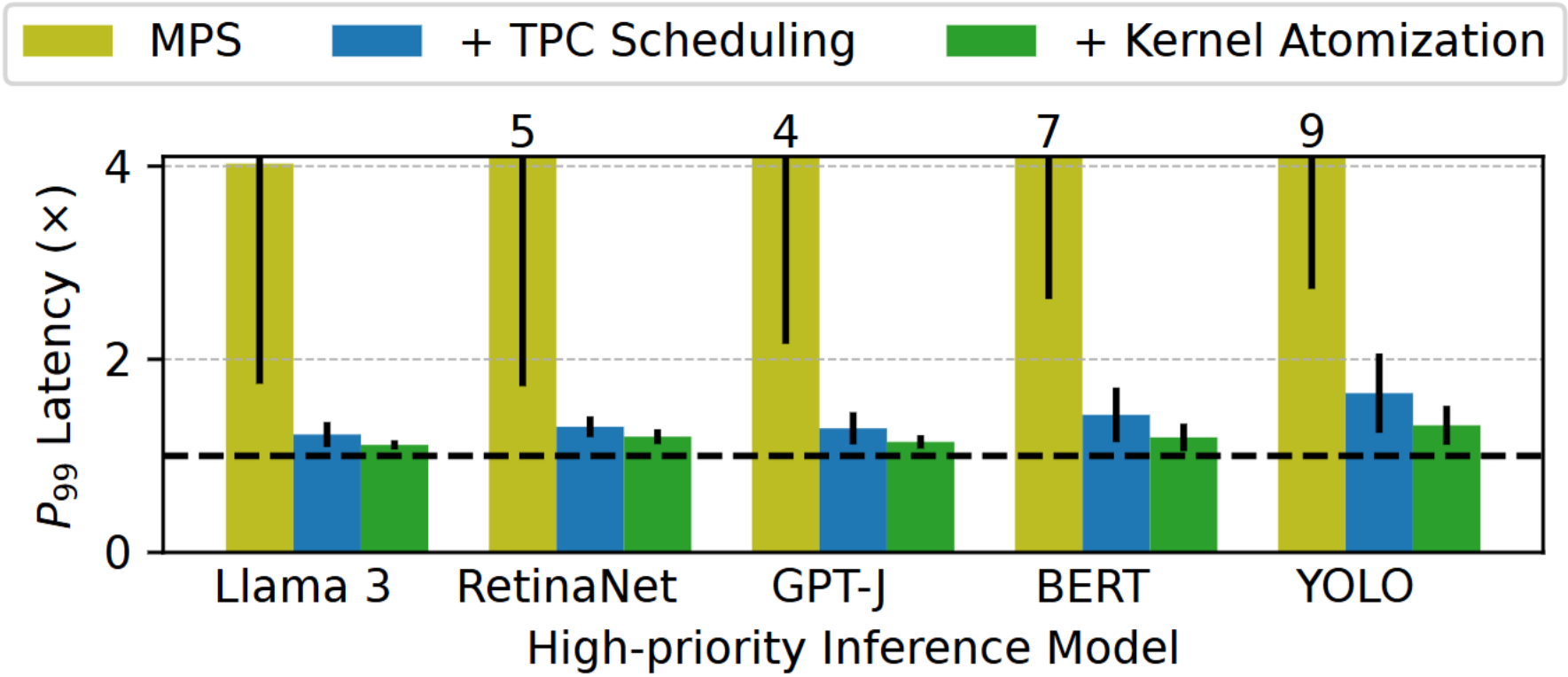
25% energy savings



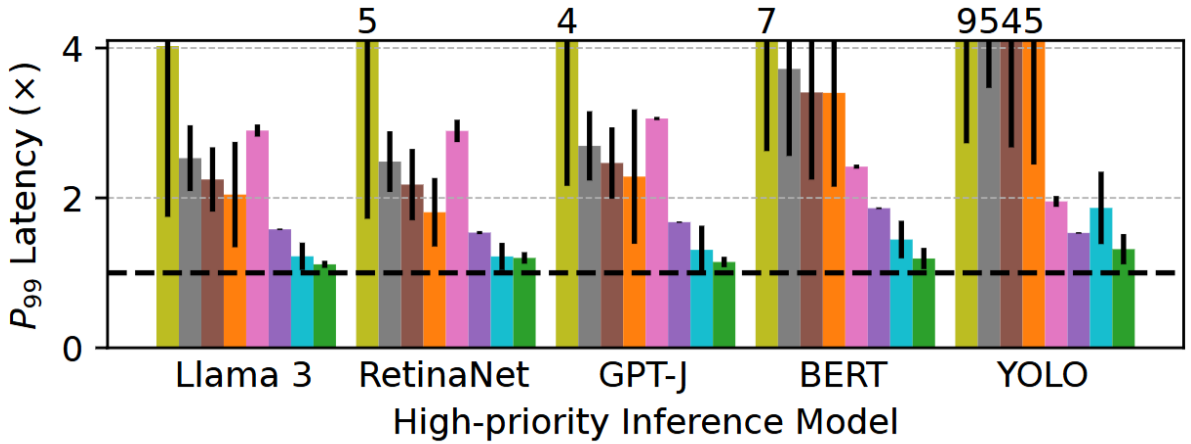
BREAKING THE TRADE-OFF

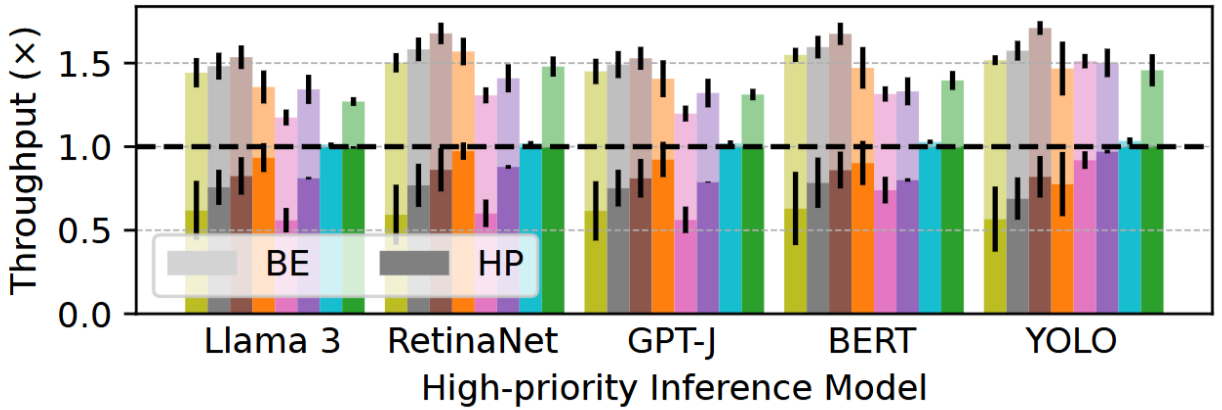


IT'S SUPER EFFECTIVE!



IT'S SUPER EFFECTIVE!





THANK YOU FOR YOUR ATTENTION 💯 💯