Tally: Non-Intrusive Performance Isolation for Concurrent Deep Learning Workloads

Wei Zhao[1,4], Anand Jayarajan[2,3,4], Gennady Pekhimenko[2,3,4]

Presenter: Jiaqi Ruan, Jia He

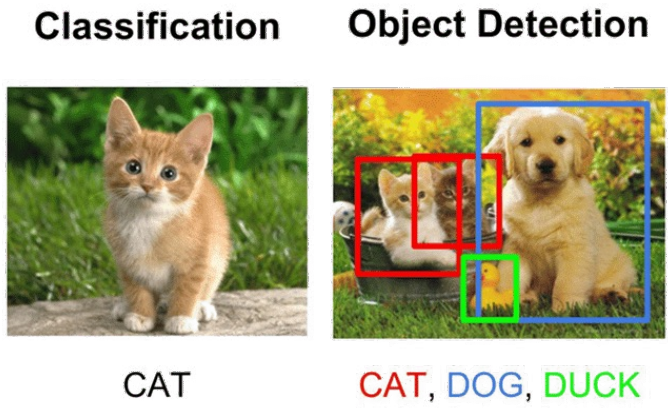
Stanford University [1]

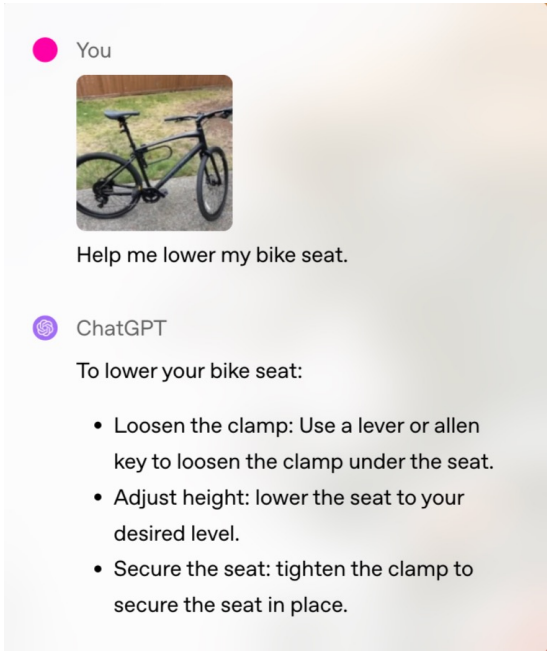
UNIVERSITY OF TORONTO [2]

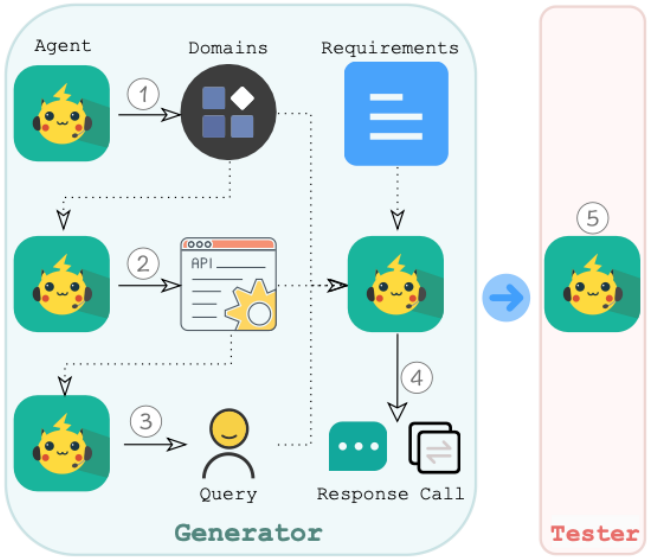
VECTOR INSTITUTE [3]

CentML [4]

DL is Gaining Unprecedented Popularity







Soaring Investments in GPU Clusters

NVIDIA H100 GPU Units By Company

Company/Entity	H100 GPU Count
Meta	350,000
XAI/X	100,000
Tesla	
Lambda	
Google A3	
Oracle Cloud	

Ensuring high GPU utilization is crucial for cost-efficiency!

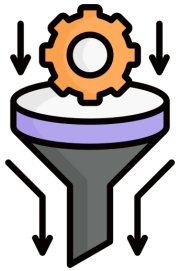
While we don't know the precise mix of GPUs sold, each Nvidia H100 80GB HBM2E compute GPU add-in-card (14,592 CUDA cores, 26 FP64 TFLOPS, 1,513 FP16 TFLOPS) retails for around $30,000 in the U.S. However, this is not the company's highest-performing Hopper architecture-based part. In fact, this is the cheapest one, at least for now. Meanwhile in China, one such card can cost as much as $70,000.

350,000 * 30,000 ≈ $10 billion

https://www.visualcapitalist.com/which-companies-own-the-most-nvidia-h100-gpus/
https://www.tomshardware.com/news/nvidia-to-sell-550000-h100-compute-gpus-in-2023-report

Meanwhile - GPUs are Severely Underutilized

- 2022 Alibaba Study:
  - median GPU utilization in 6000-GPU cluster is only **4.2%**.
- Sources of Inefficiency:

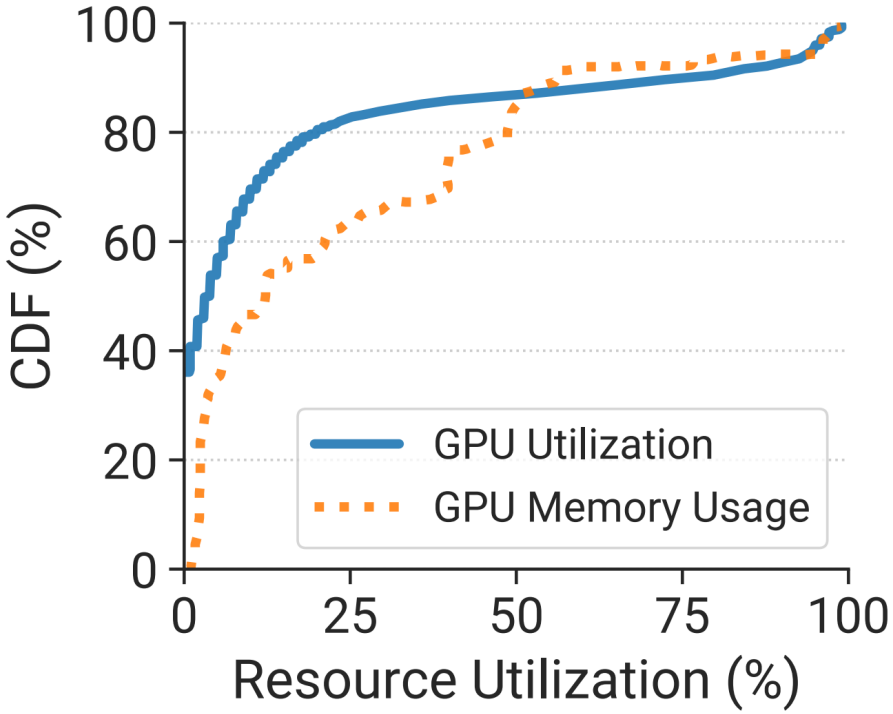


Training:

bottlenecks in CPU execution

Inference:

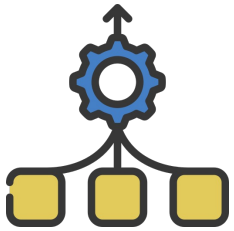
fluctuating user traffic



GPU utilization distribution in an Alibaba cluster

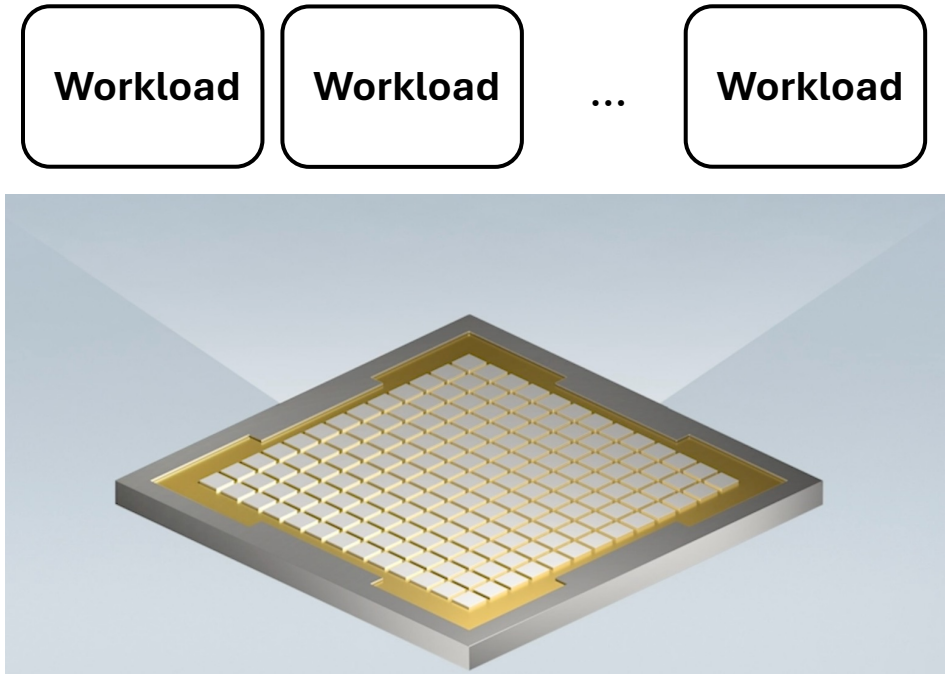
Weng et al., "MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters," NSDI 2022.

GPU Sharing to the Rescue



Consolidating multiple workloads to share a single GPU's resources.

The same Alibaba study shows GPU sharing can cut resource requirements by up to **73%**.



https://developer.nvidia.com/techdemos/video/disc03

Weng et al., " MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters," NSDI 2022.

Yet, GPU Sharing is Rarely Used in Clusters

Limitations of Current GPU Sharing Solutions:

High integration and maintenance cost due to intrusive code modifications

Lack of performance isolation guarantees leads to violations of task SLAs

Limited application compatibility due to reliance on workload-specific characteristics

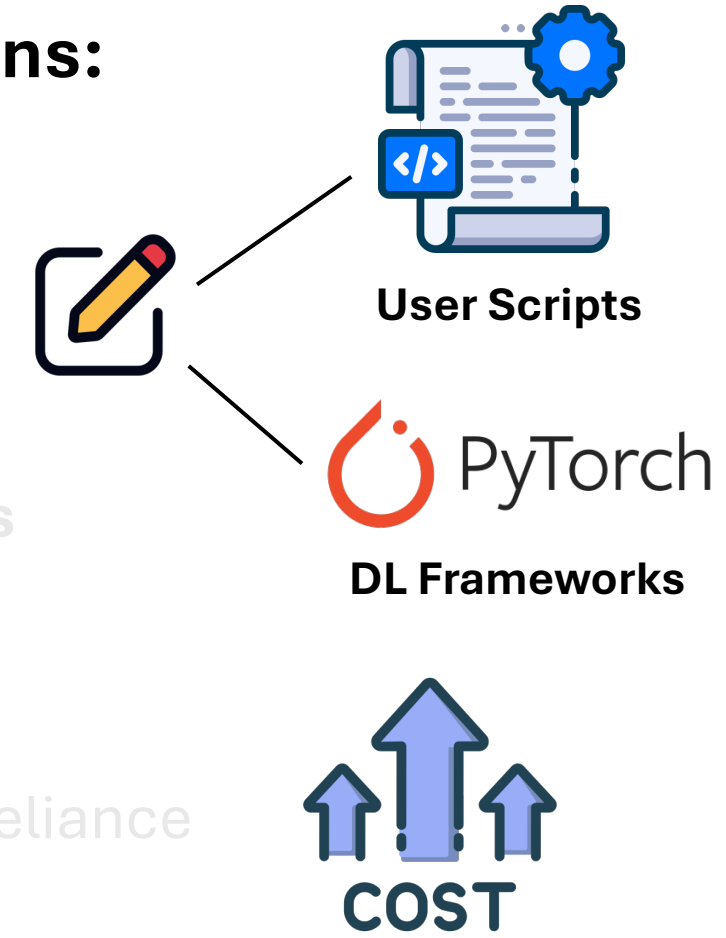
Yet, GPU Sharing is Rarely Used in Clusters

Limitations of Current GPU Sharing Solutions:

High integration and maintenance cost due to intrusive code modifications

Lack of performance isolation guarantees leads to violations of task SLAs

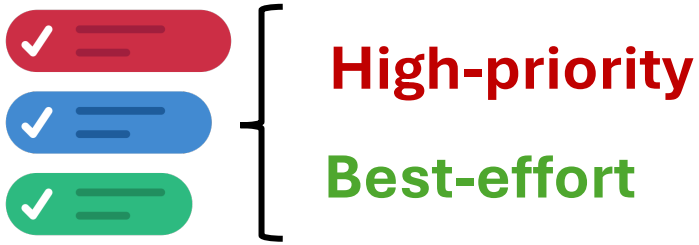
Limited application compatibility due to reliance on workload-specific characteristics



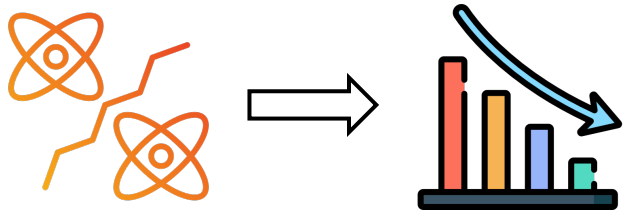
Yet, GPU Sharing is Rarely Used in Clusters

Limitations of Current GPU Sharing Solutions:

High integration and maintenance cost due to intrusive code modifications



Lack of performance isolation guarantees leads to violations of task SLAs



Interference causes performance degradation

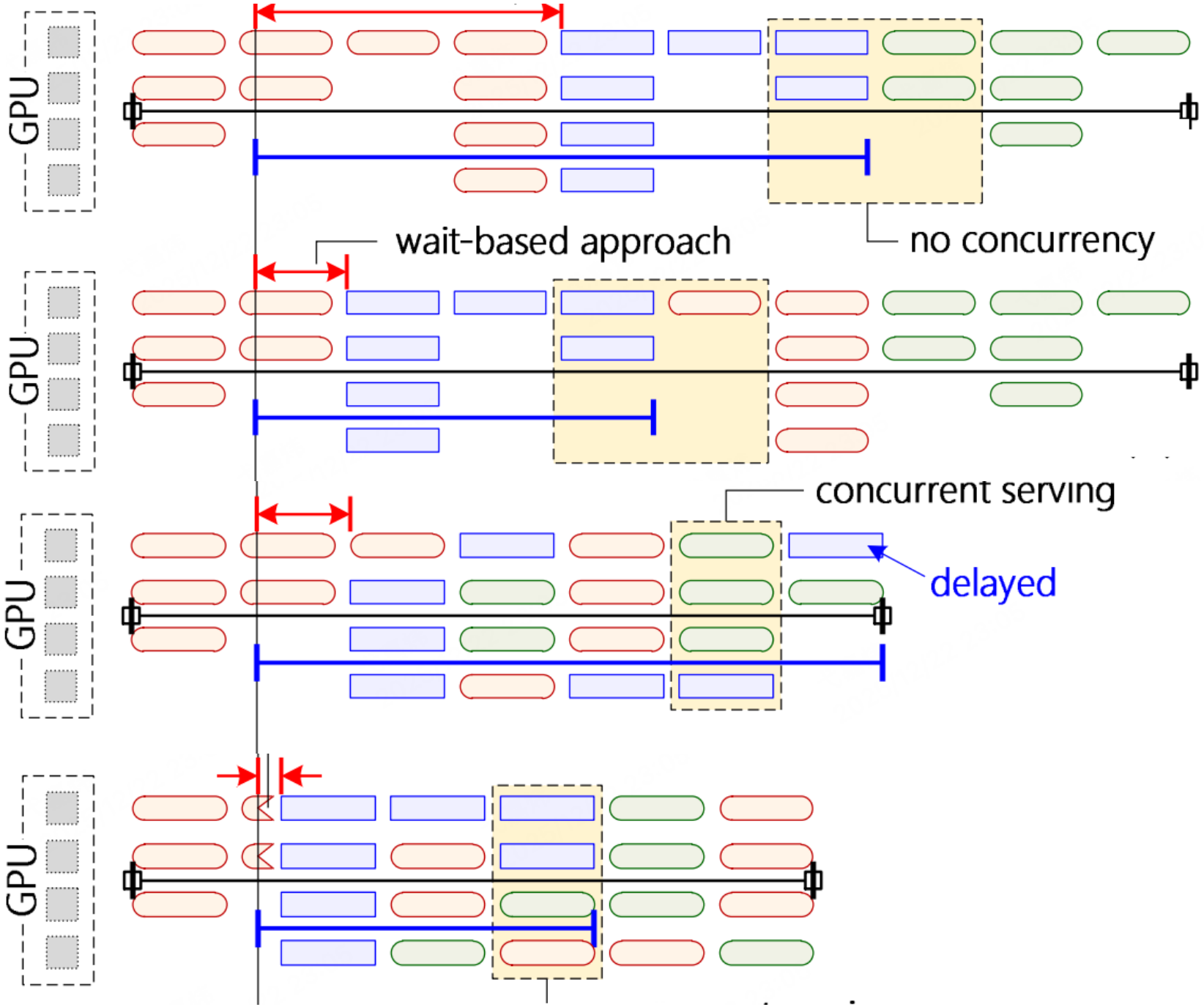
Limited application compatibility due to reliance on workload-specific characteristics

Iteration

Kernel

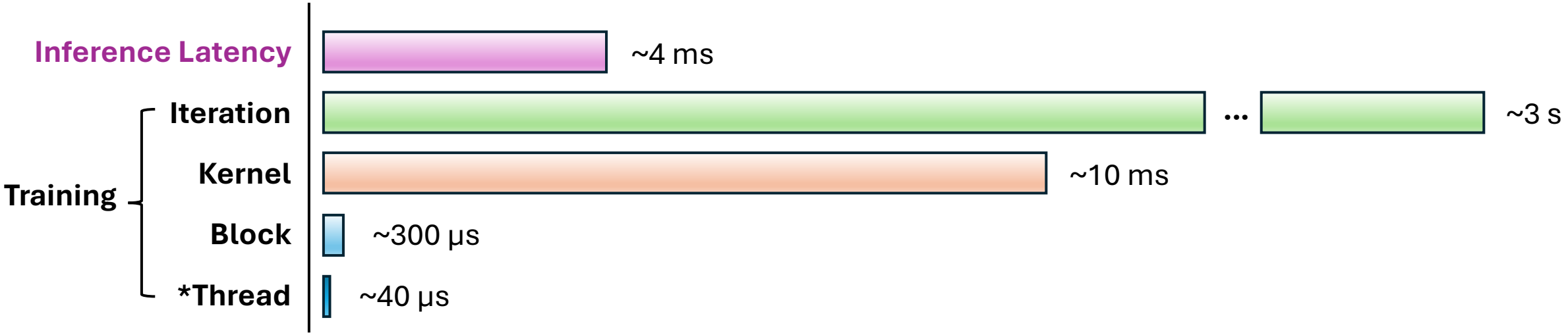
Stream

Block



Scheduling Granularity for DL Workloads

- High-priority: **BERT Inference**
- Best-effort: **Whisper Training**
- Co-executed on a NVIDIA A100 GPU



*: the thread-level latency is taken from the REEF paper as it is not supported in NVIDIA GPUs.

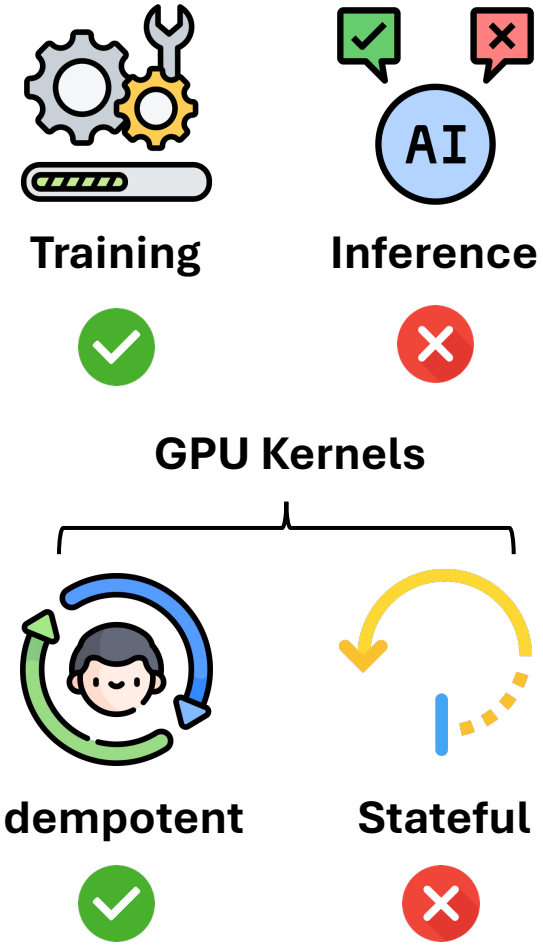
Yet, GPU Sharing is Rarely Used in Clusters

Limitations of Current GPU Sharing Solutions:

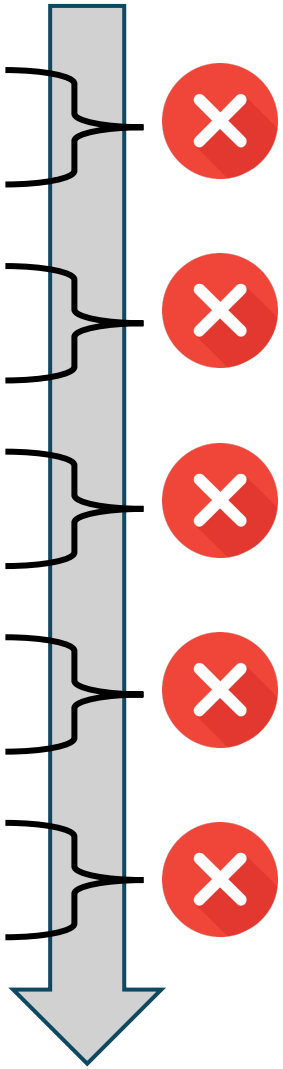
High integration and maintenance cost due to intrusive code modifications

Lack of performance isolation guarantees leads to violations of task SLAs

Limited application compatibility due to reliance on workload-specific characteristics



Yet, GPU Sharing is Rarely Used in Clusters



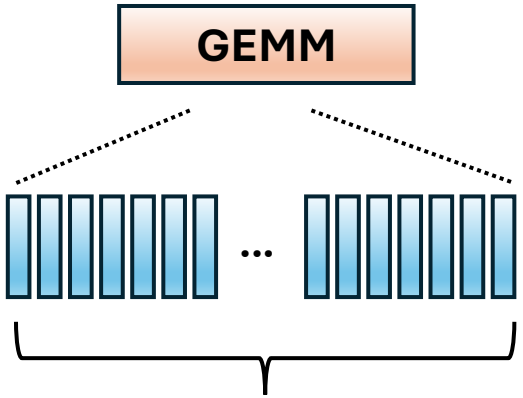
		Intrusiveness	Granularity	Generality
2017	EffiSha [PPoPP'17]	Intrusive code	Block level	All
2020	Salus [MLSys'20]	Non-Intrusive	Iteration level	All
2022	REEF [OSDI'22]	Non-Intrusive	thread level	Indempotent
2023	TGS [NSDI'23]	Non-Intrusive	kernel level	Static
2024	Orion [EuroSys'24]	Intrusive code	stream level	All

Choosing the Right Scheduling Granularity

Adopted by Tally

	Block-level	*Thread-level*
Latency	~100s μs 	~10s μs 
Generality	Universally applicable 	Limited to idempotent kernels 

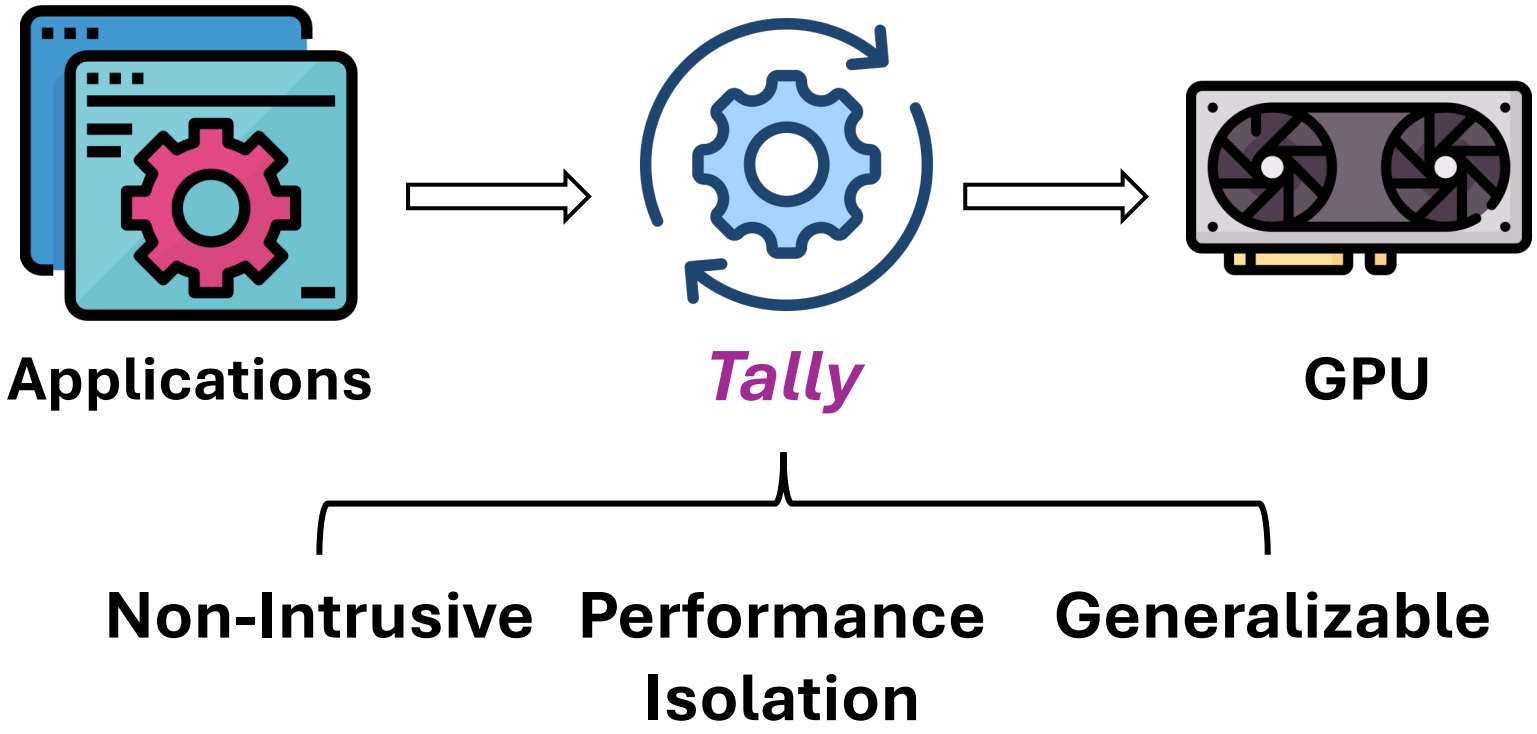
Block-level:



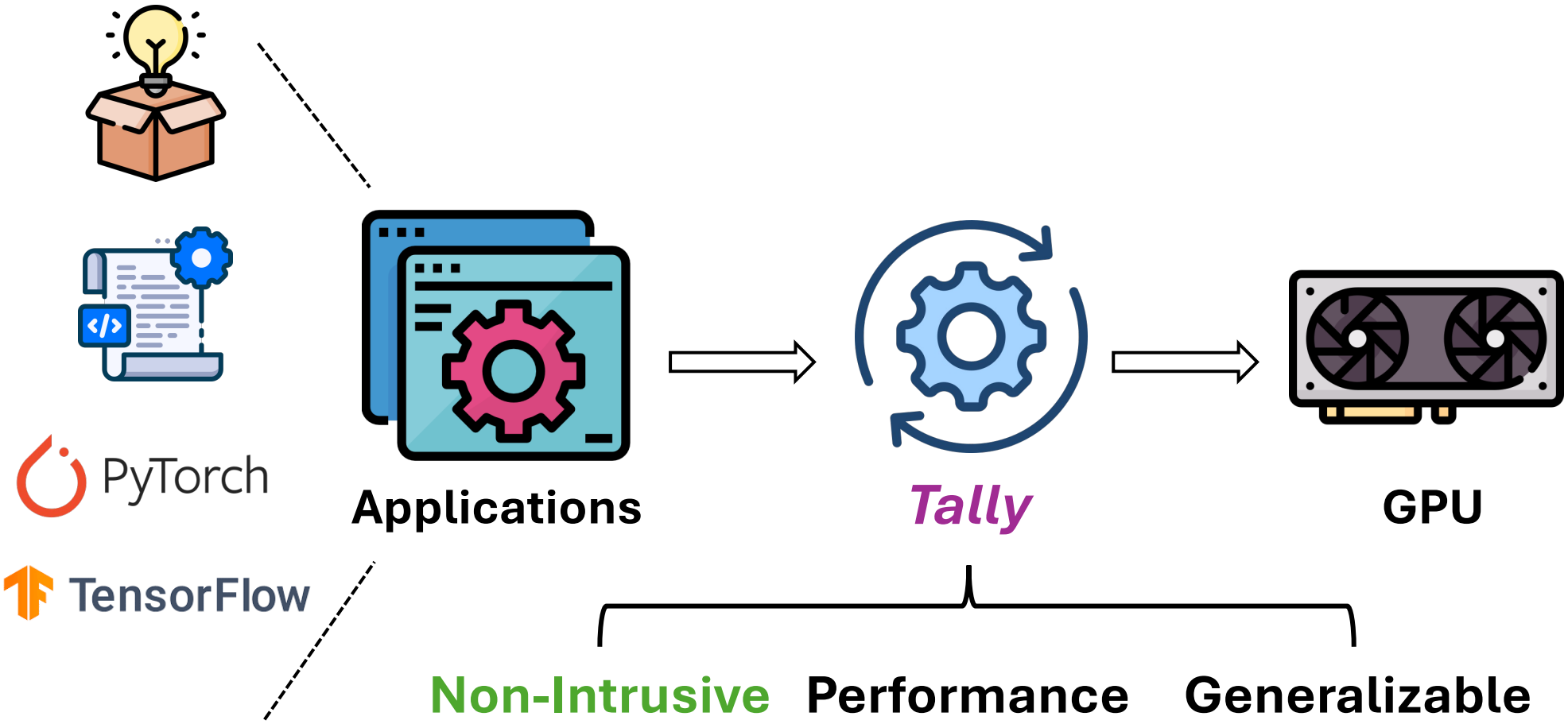
Thread-level:

Disruptive abort and restart of a block require idempotency

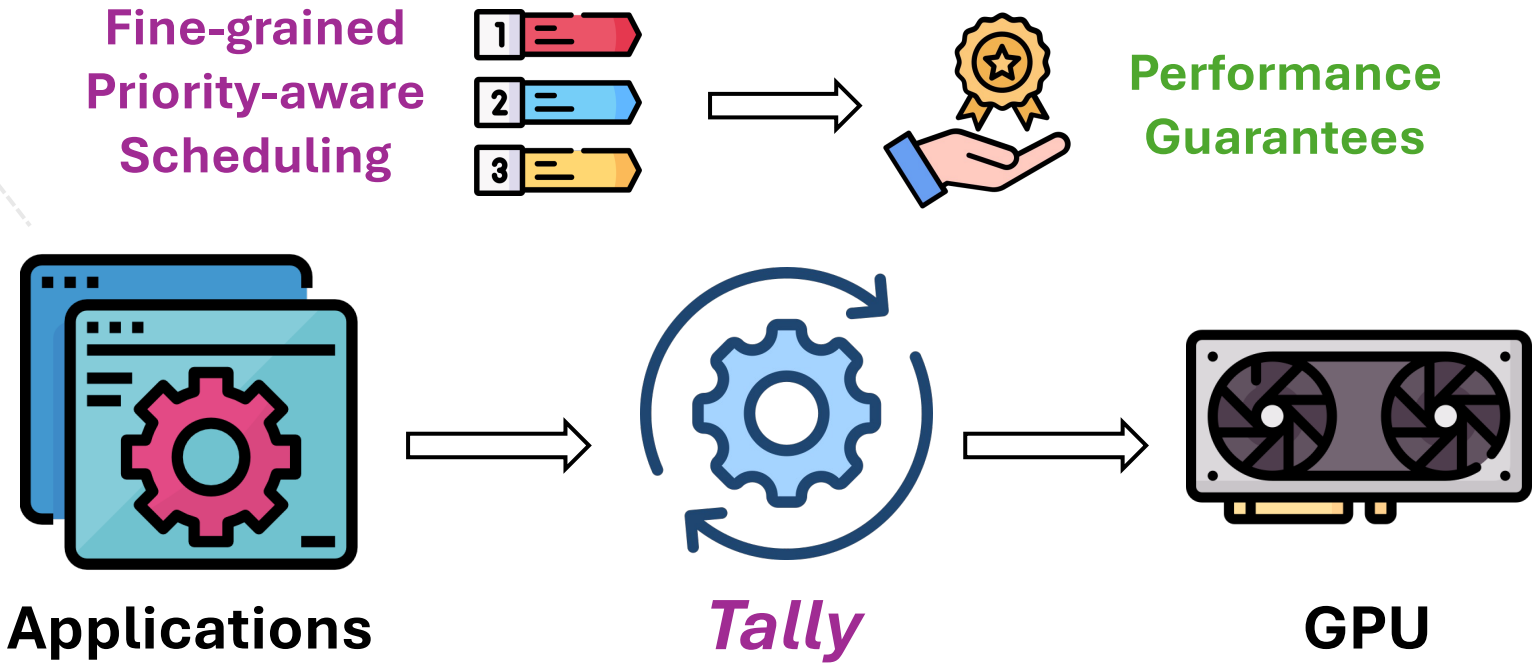
Tally: Enabling Practical GPU Sharing



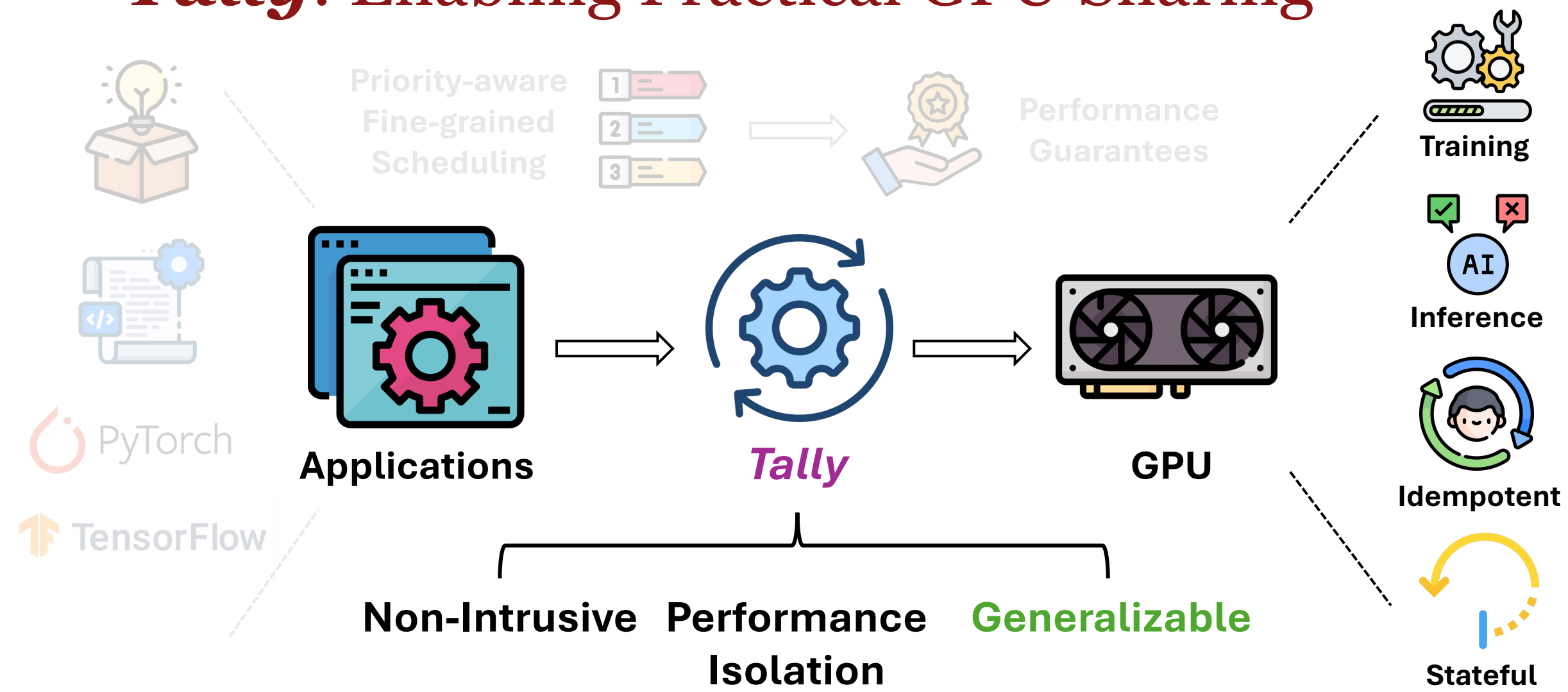
Tally: Enabling Practical GPU Sharing



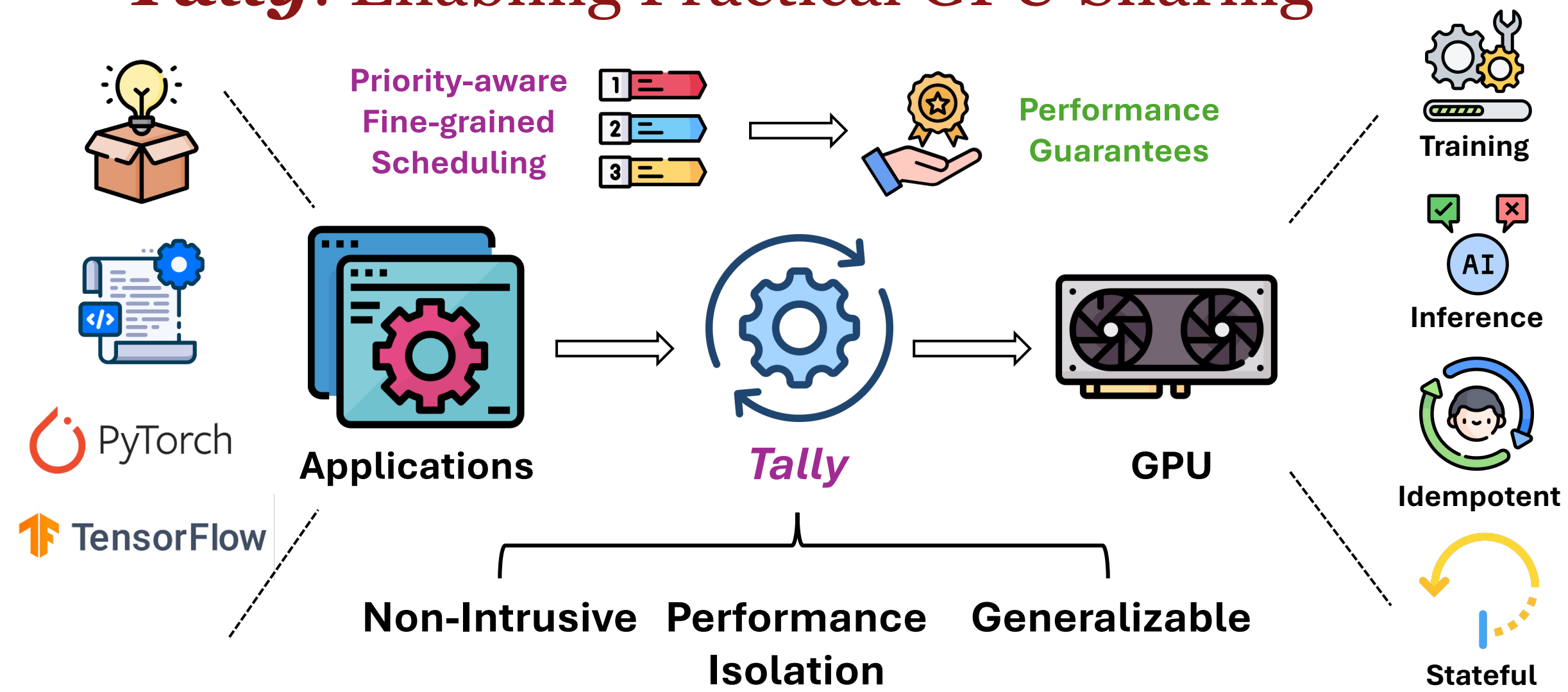
Tally: Enabling Practical GPU Sharing



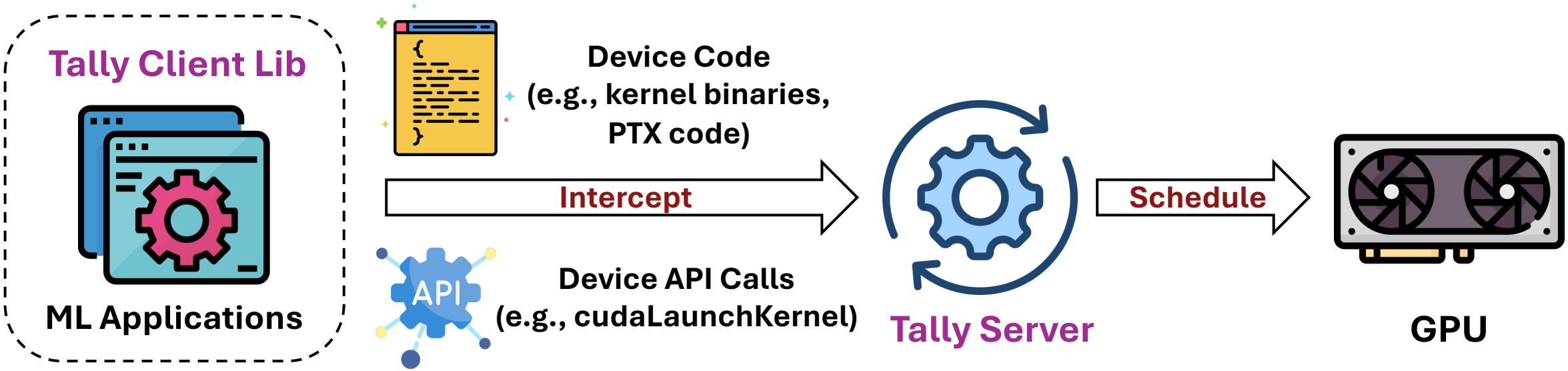
Tally: Enabling Practical GPU Sharing



Tally: Enabling Practical GPU Sharing

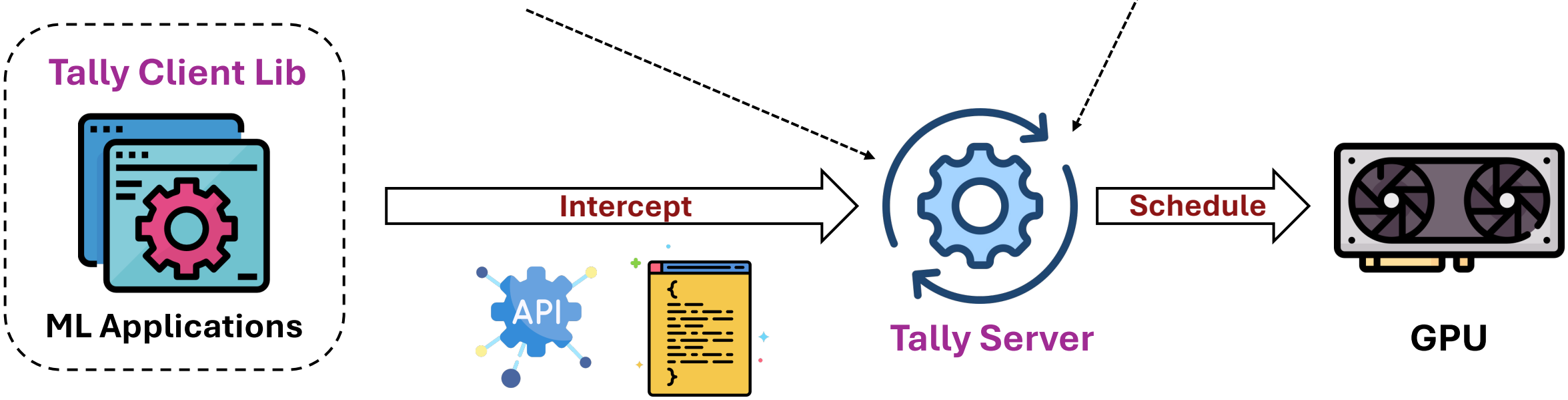


Overview of Tally — Act as Virtualization Layer



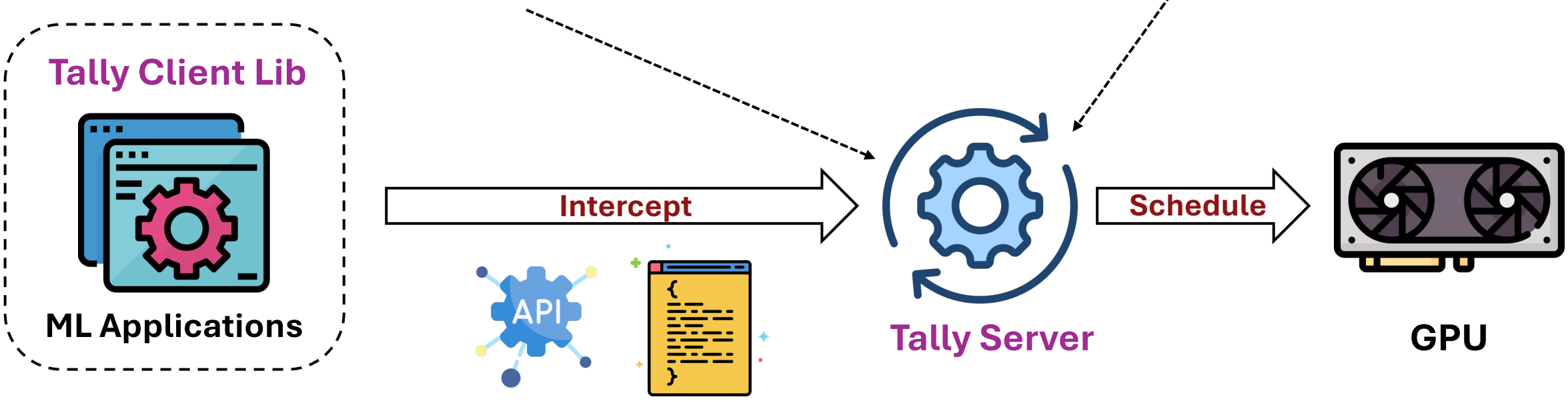
Overview of Tally

- Core mechanisms
  - Block-level scheduling primitives for best-effort kernels
  - Transparent profiler
  - Priority-aware scheduler



Overview of Tally

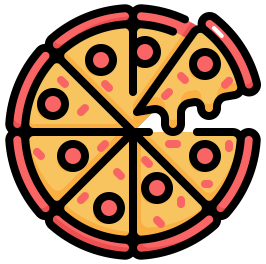
- Core mechanisms
  - **Block-level scheduling primitives for best-effort kernels**
  - Transparent profiler
  - Priority-aware scheduler



Tally's Block-level Scheduling

- Key Insights:
  - Block-level scheduling is **non-intrusive** and **generalizable** for all ML tasks
  - Previous results are **kept** instead of simply discarding
  - Turnaround latency (~100us) is **tolerable**

- Tally's Scheduling Strategy towards Different Tasks
  - High-priority: keep it **as-is**
  - Best-effort: **transform** and schedule kernels at block-level **via two primitives**



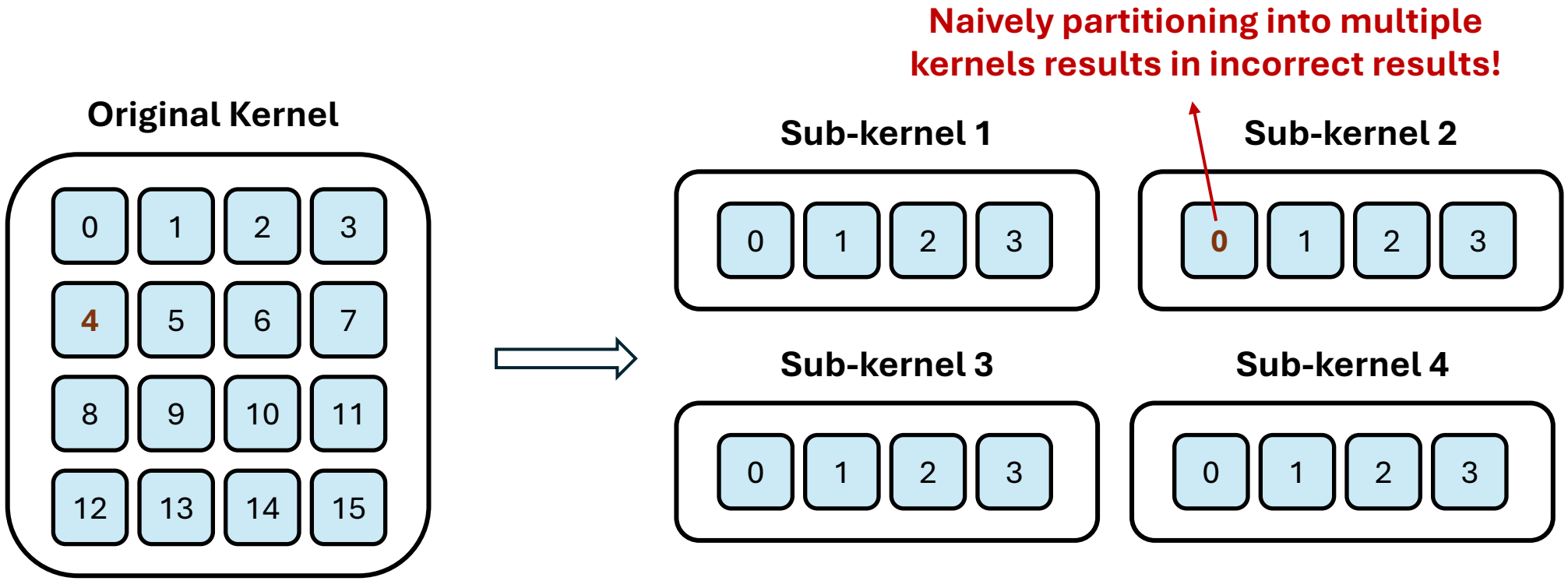
Slicing

or

Preemption

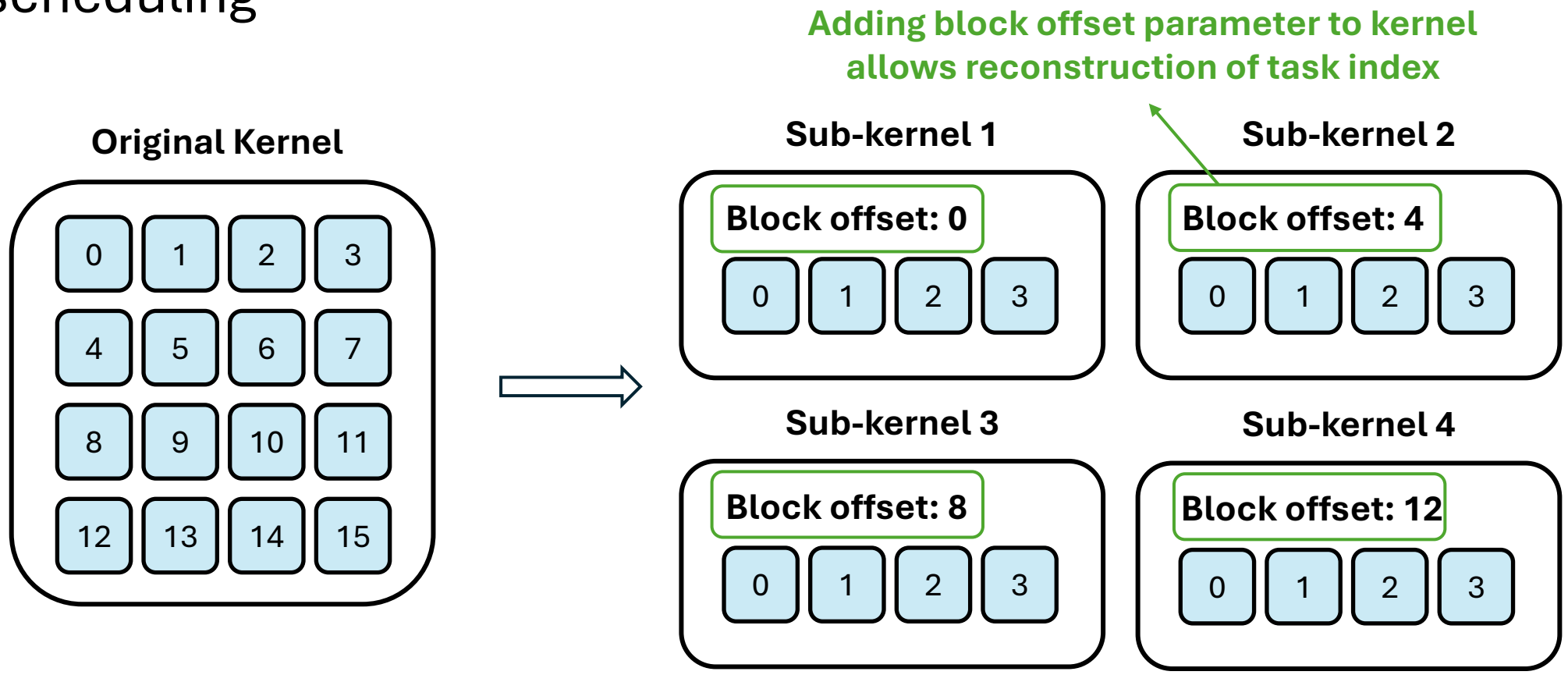
Slicing Transformation

- Divide a kernel into multiple sub-kernels to allow for more fine-grained scheduling



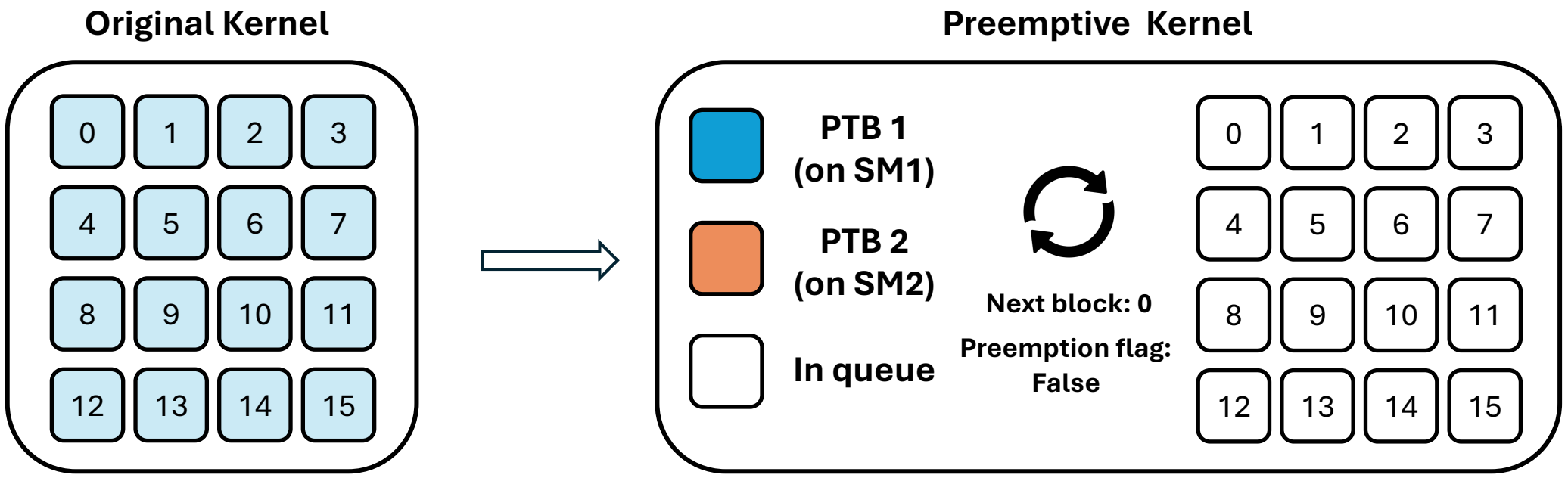
Slicing Transformation

- Divide a kernel into multiple sub-kernels to allow for more fine-grained scheduling



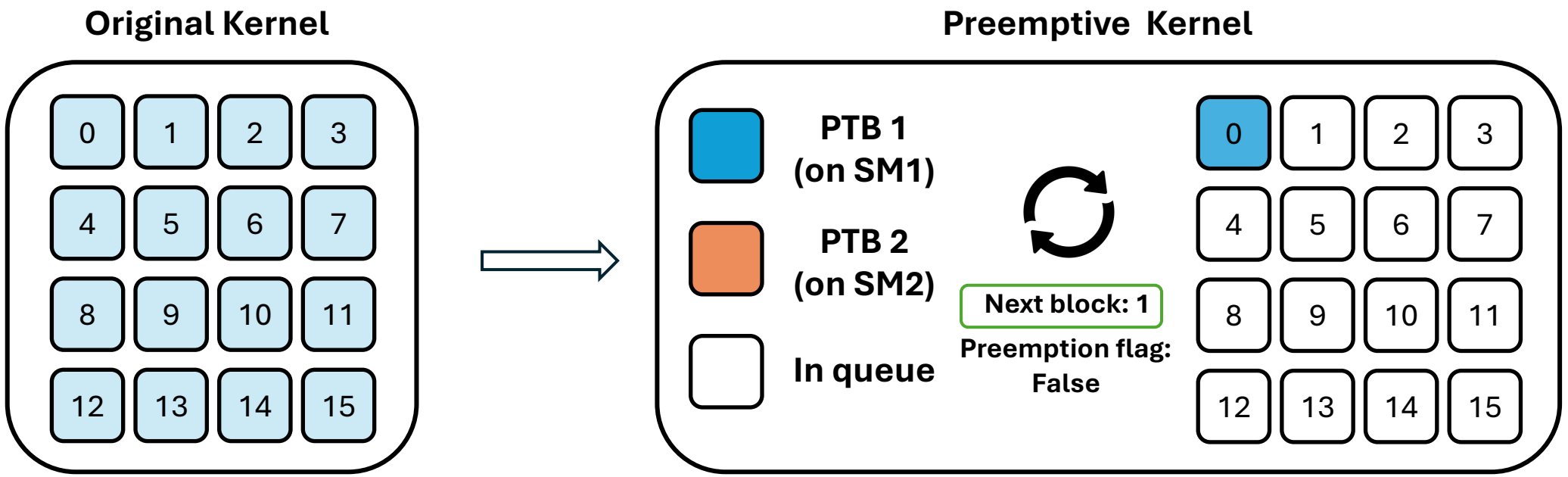
Preemption Transformation

- Modify the kernel execution mode to **Persistent Thread Block** (PTB)



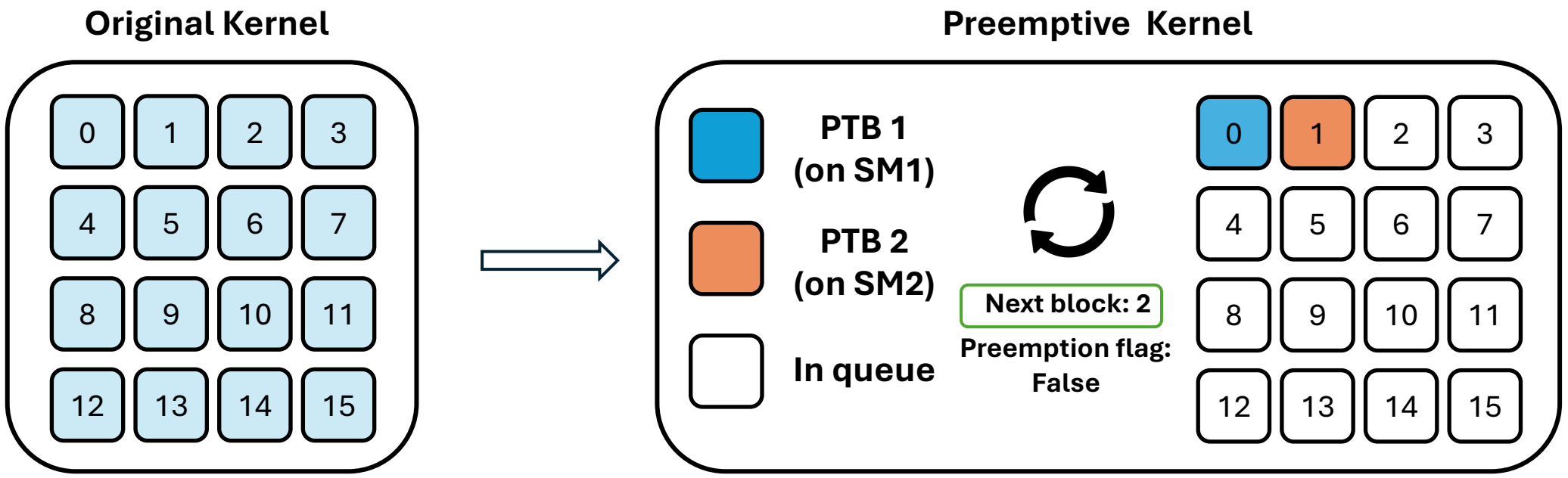
Preemption Transformation

- Modify the kernel execution mode to Persistent Thread Block (PTB)



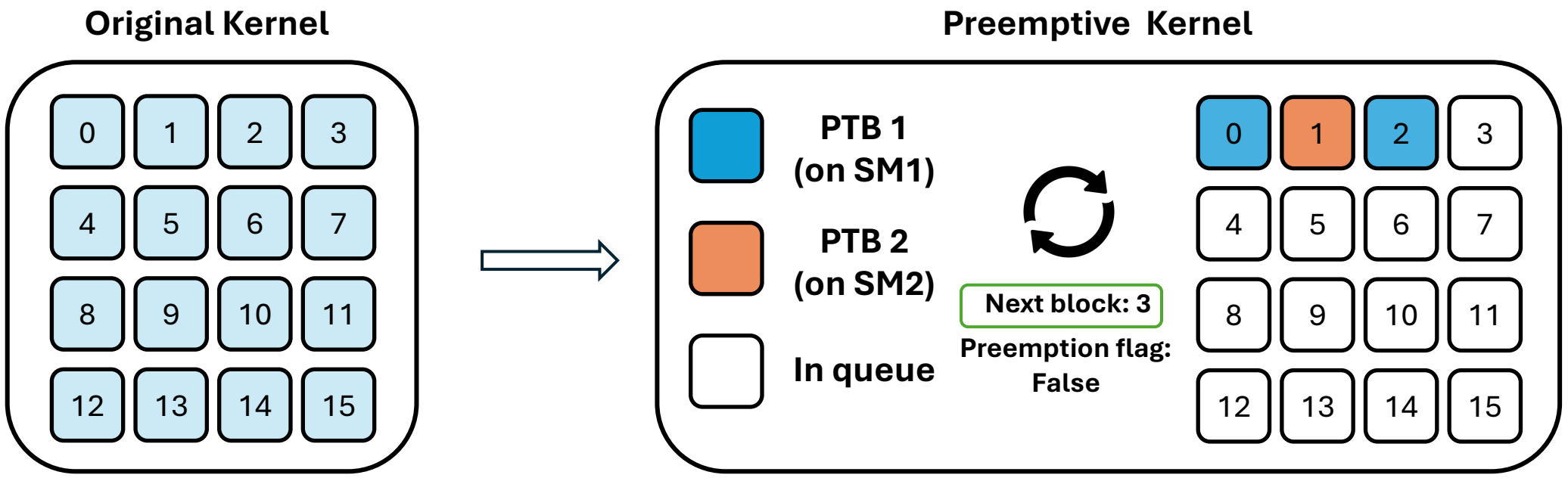
Preemption Transformation

- Modify the kernel execution mode to Persistent Thread Block (PTB)



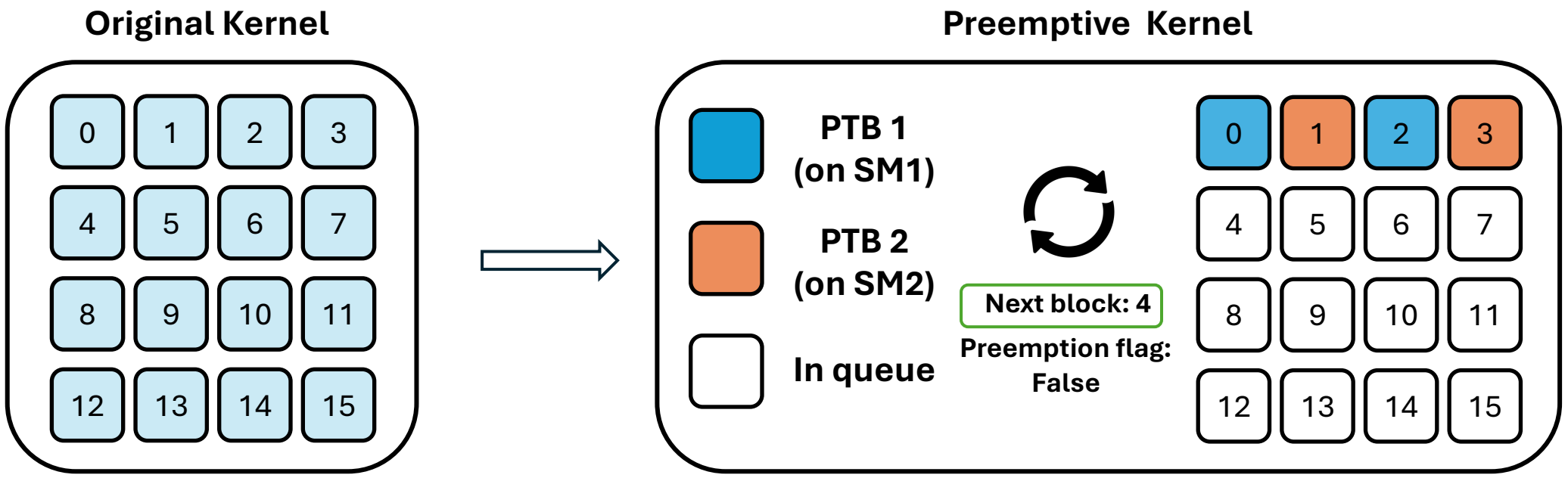
Preemption Transformation

- Modify the kernel execution mode to Persistent Thread Block (PTB)



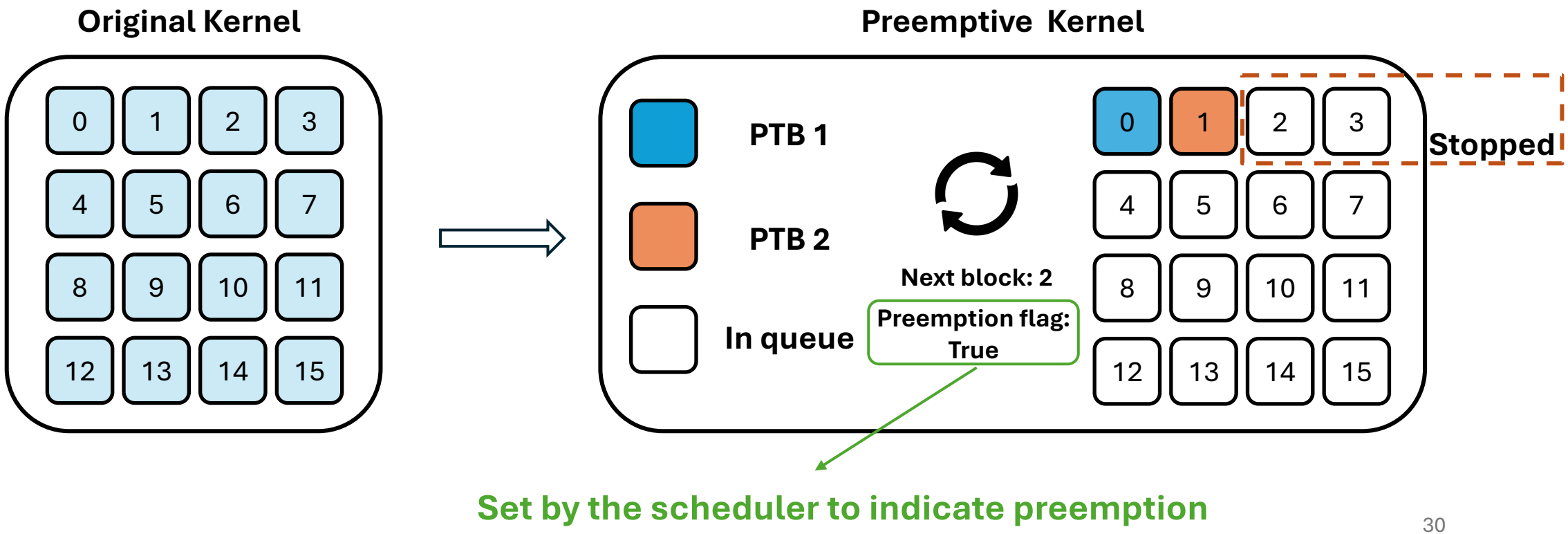
Preemption Transformation

- Modify the kernel execution mode to Persistent Thread Block (PTB)
  - **DeepGEMM also uses PTB to control SM number**



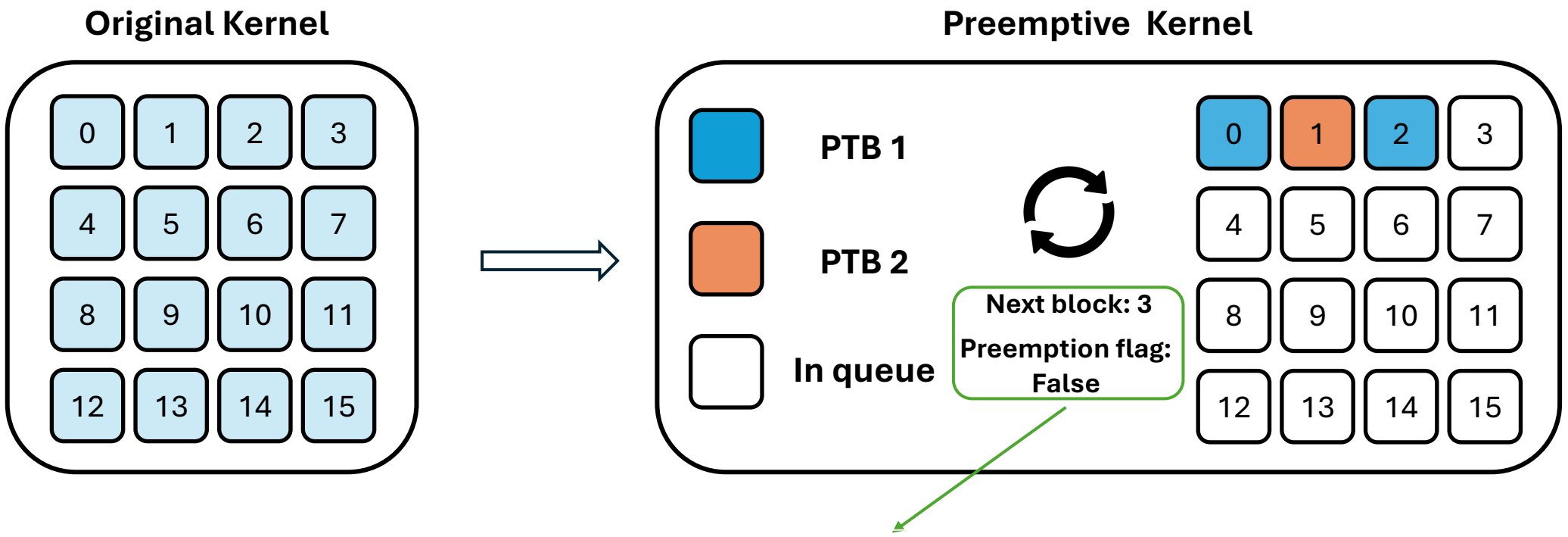
Preemption Transformation

- Allow the interruption of active kernels to accelerate resource reallocation



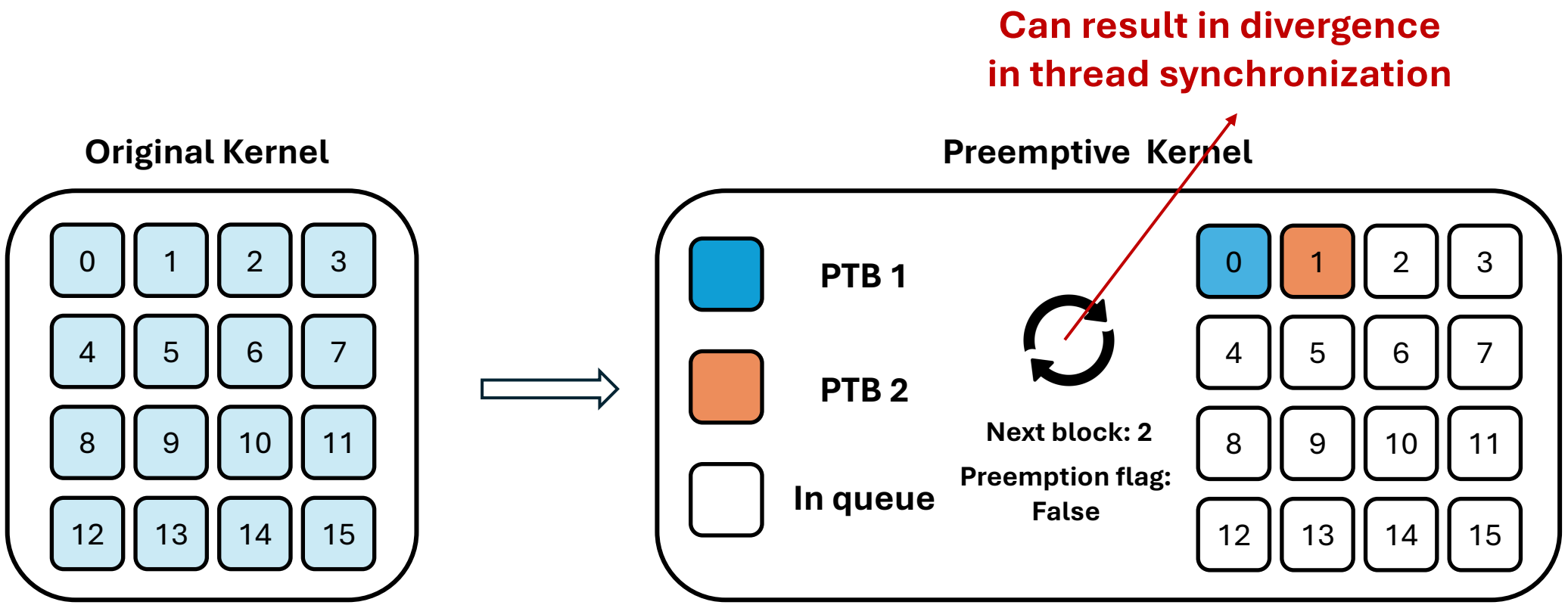
Preemption Transformation

- Allow the interruption of active kernels to accelerate resource reallocation



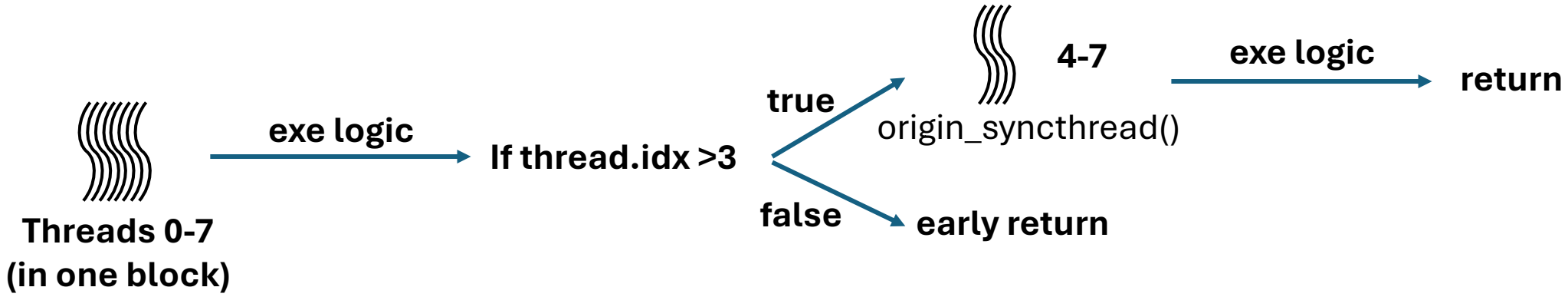
Preemption Transformation

- Transform kernels to PTB by wrapping it with an **outer control loop**



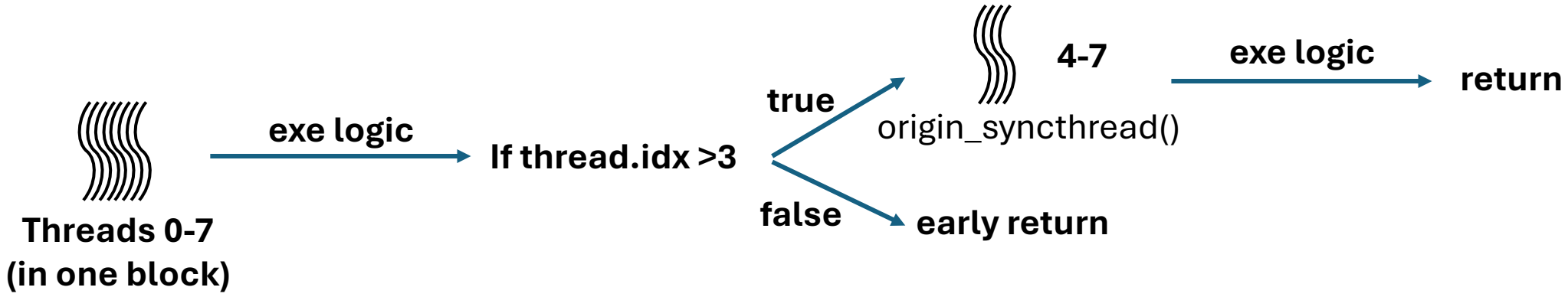
Unified Synchronization Transformation (UST)

- Original code before preemption transformation

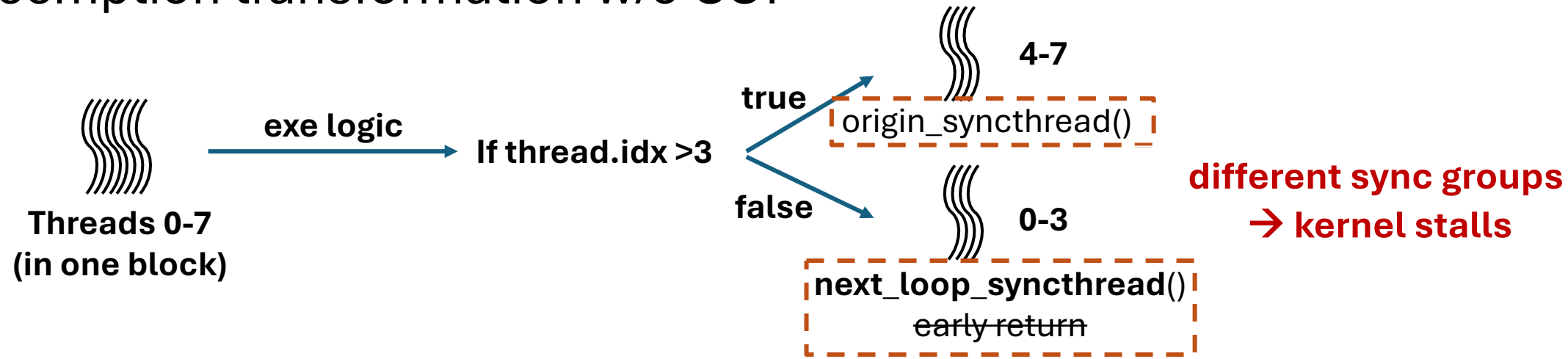


Unified Synchronization Transformation (UST)

- Original code before preemption transformation

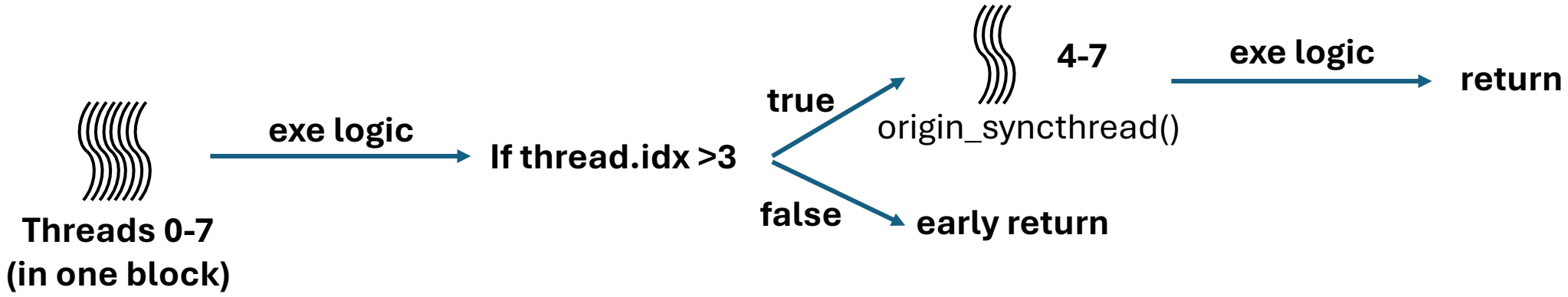


- Preemption transformation w/o UST

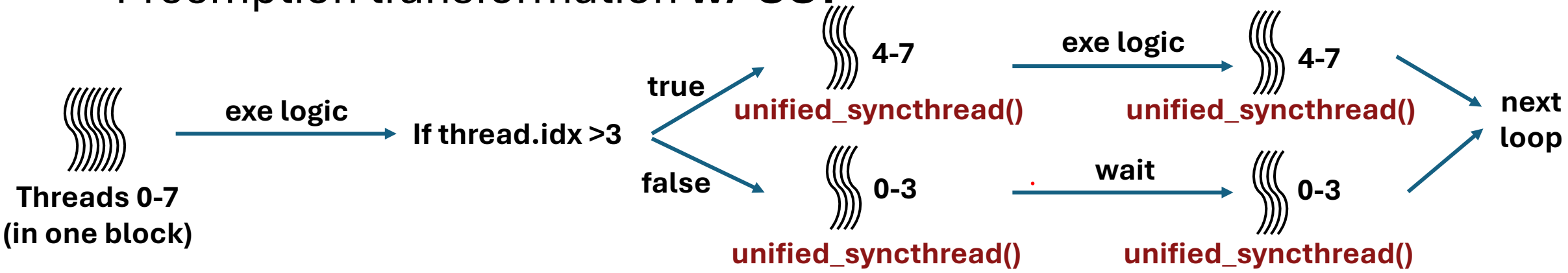


Unified Synchronization Transformation (UST)

- Original code before preemption transformation

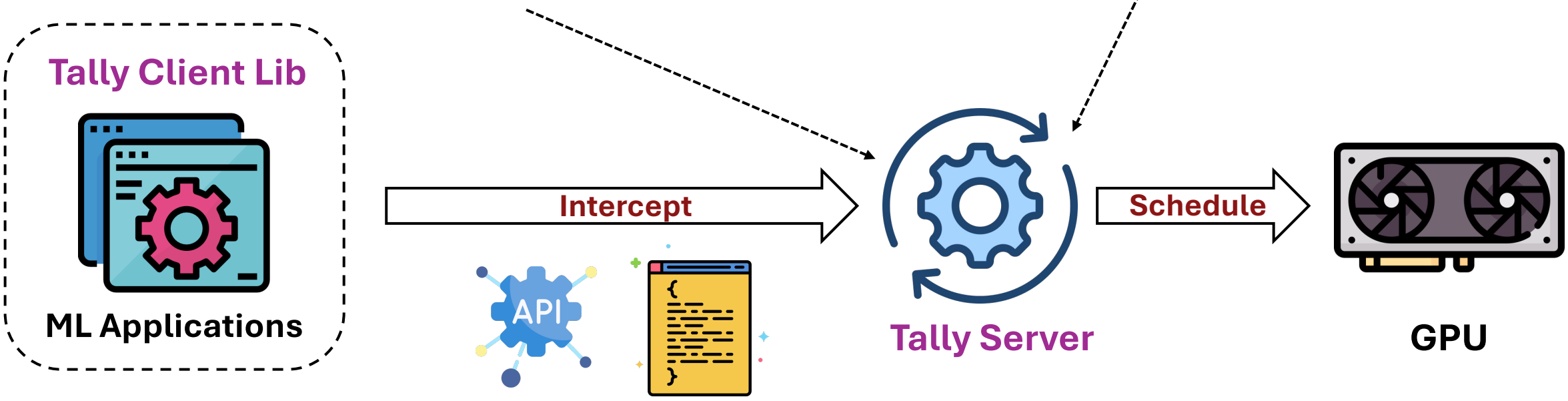


- Preemption transformation **w/ UST**



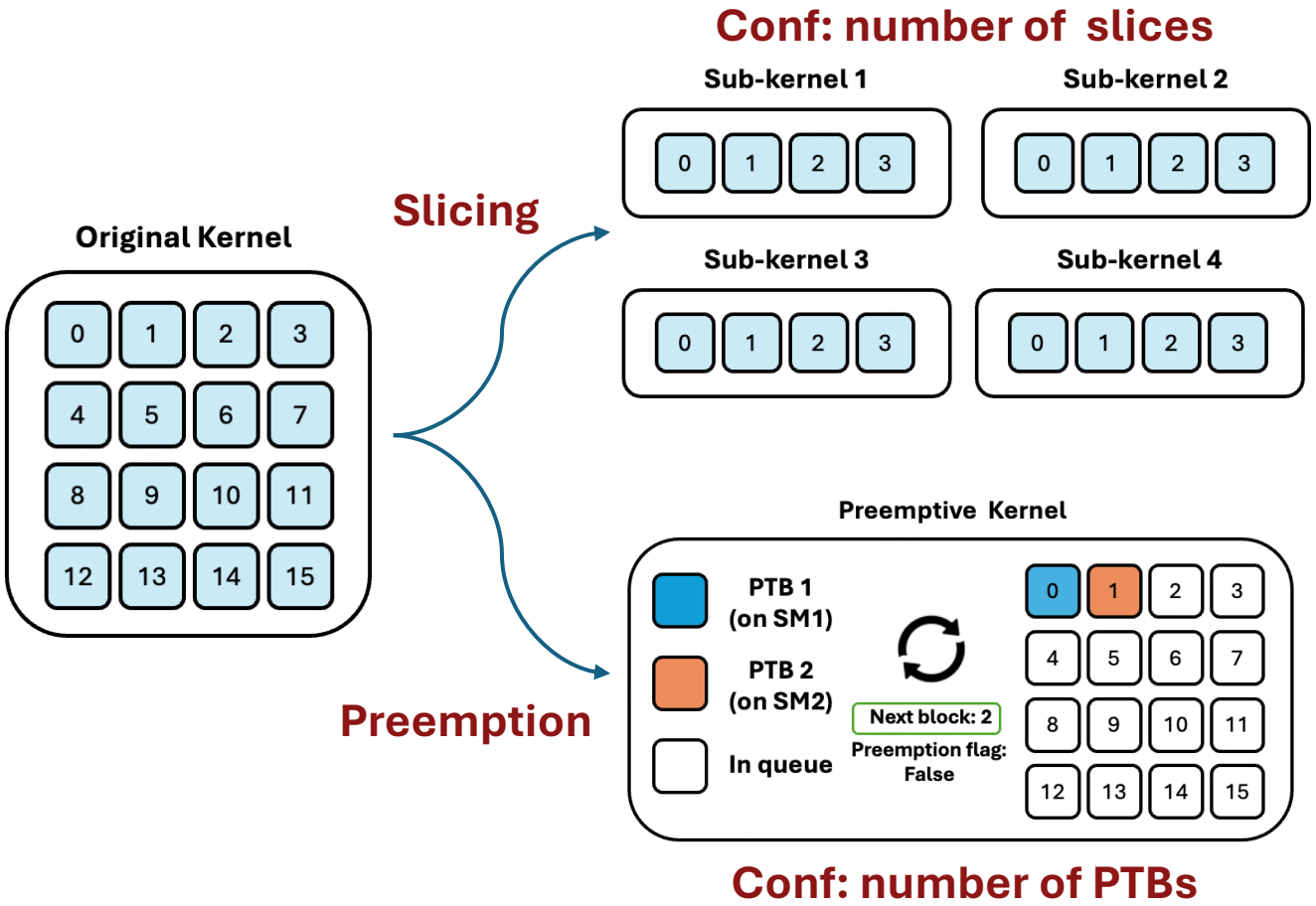
Overview of Tally

- Core mechanisms
  - Block-level scheduling primitives for best-effort kernels
  - **Transparent profiler**
  - Priority-aware scheduler



Pros and Cons of Different Primitives

- **Turnaround latency** and **throughput** vary when using different block-level primitives and their launch configurations



Pros: lightweight, no sync overhead
Cons: launch overhead of sub-kernels

Pros: one-time kernel launch
Cons: sync overhead by UST

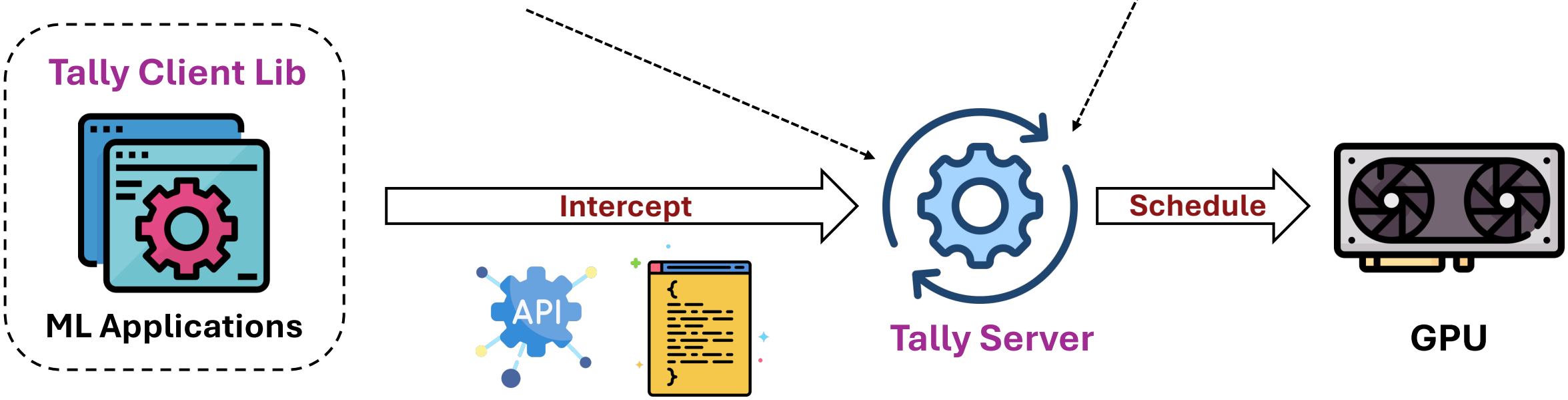
Transparent Profiler

- Profile and estimate the performance of transformed kernels in **best-effort** tasks **at runtime**
  - Generate candidate configurations encompassing both slicing and preemption
  - Estimates **turnaround latency** of different configurations
    - Slicing: execution time of one sub-kernel
    - Preemption: heuristic approximation → $$turnaround\_latency = \frac{kernel\_latency \times PTB\_blocks}{total\_blocks}$$

- **Scheduler** selects the one that achieves the optimal performance while complying with a predefined turnaround latency threshold

Overview of Tally

- Core mechanisms
  - Block-level scheduling primitives for best-effort kernels
  - Transparent profiler
  - **Priority-aware scheduler**



Priority-aware Scheduler

- Wisely schedules kernel to guarantee **SLA** and improve **throughput** based on block-level primitives and transparent profiler

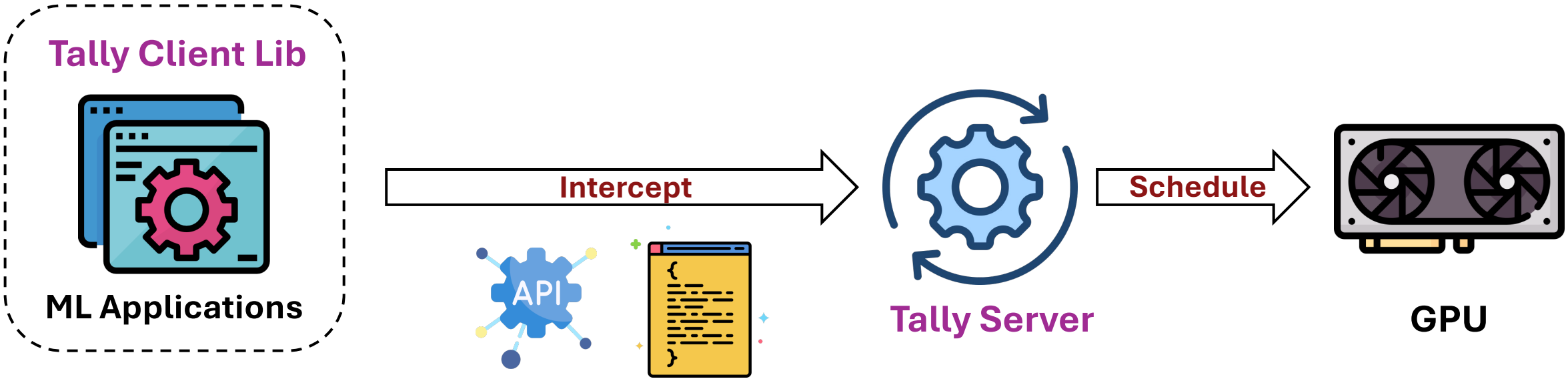
- Scheduling strategy **in a nutshell**
  - If meets high-priority kernel
    1. **Preempt** block-level best-effort kernels (slicing/preemption)
    2. Schedule high-priority kernel instantly
  - If doesn't have high-priority kernel
    1. Choose best-effort kernels to schedule
    2. If kernel doesn't have block-level launch configuration, use **transparent profiler** to find optimal configuration **on-the-fly**

Overview of Tally

- Core mechanisms
  - Block-level scheduling primitives for best-effort kernels
  - Transparent profiler
  - Priority-aware scheduler

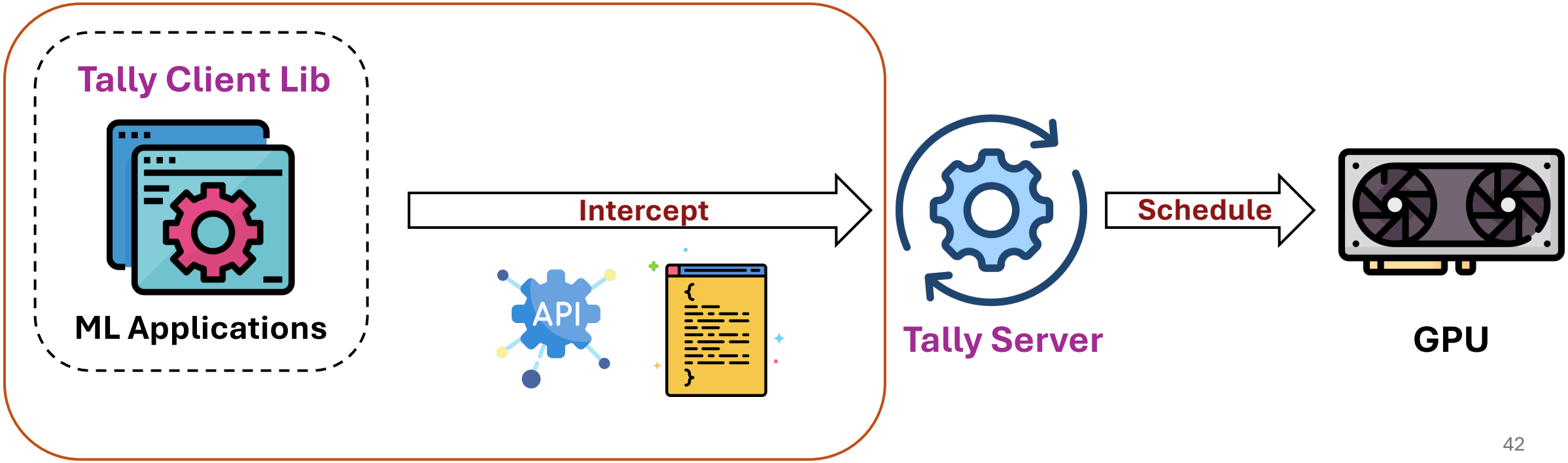
Non-Intrusive
Generalizable

Performance
Isolation



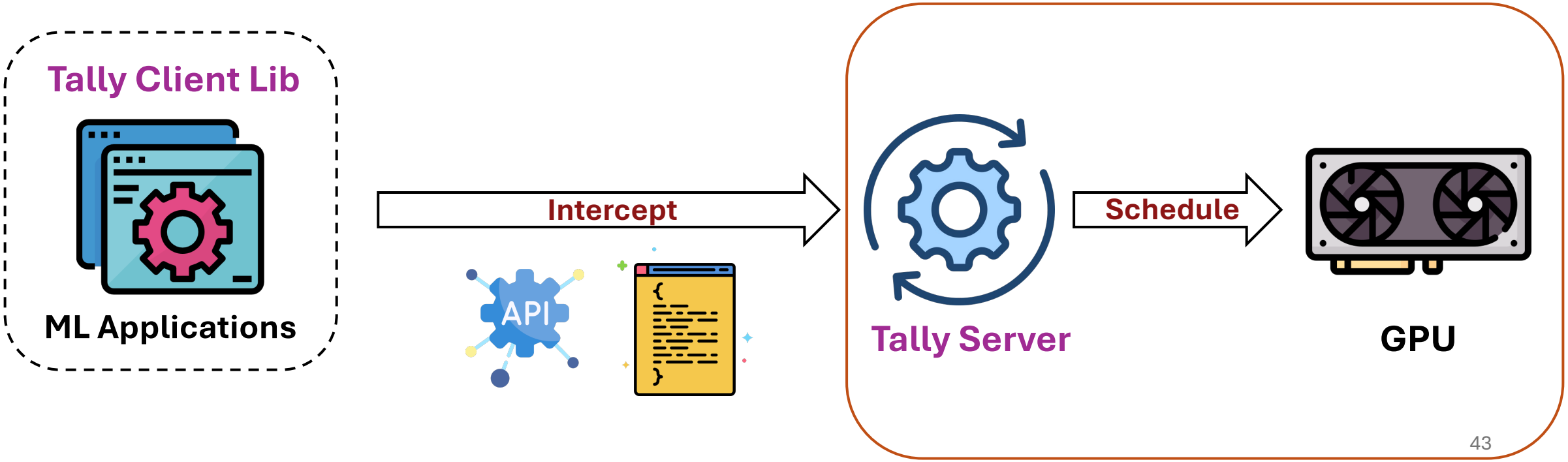
Implementation Details

- Tally's client-side
  1. Use LD_PRELOAD in Linux to **intercept** API calls
  2. Intercepted API calls are **sent** to the server for actual execution
     - **Shared memory** and **local state caching** are used to reduce communication overhead



Implementation Details

- Tally's server-side
  1. Apply kernel transformations for best-effort kernels at **PTX level**, then **recompile** to executable GPU kernels
  2. Schedule kernels based on priority-aware scheduler

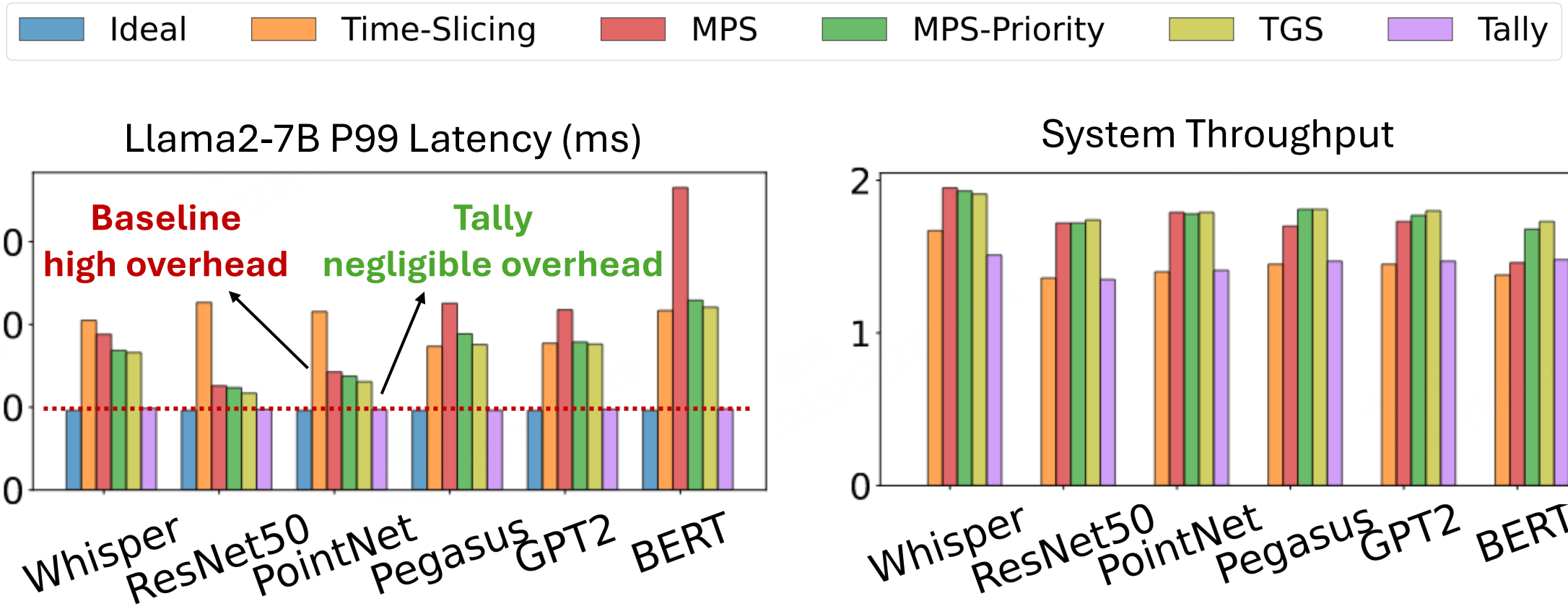


Evaluation

- Hardware:
  - NVIDIA A100 GPU
- Workloads:
  - High-priority Inference + Best-effort Training
- Metrics:
  - 99th Percentile Latency
  - System Throughput
- Baselines:
  - Ideal (no sharing, evaluate **latency**)
  - Time-Slicing (default sharing mechanism)
  - NVIDIA MPS (with and w/o priority mode)
  - TGS (state-of-the-art GPU sharing)

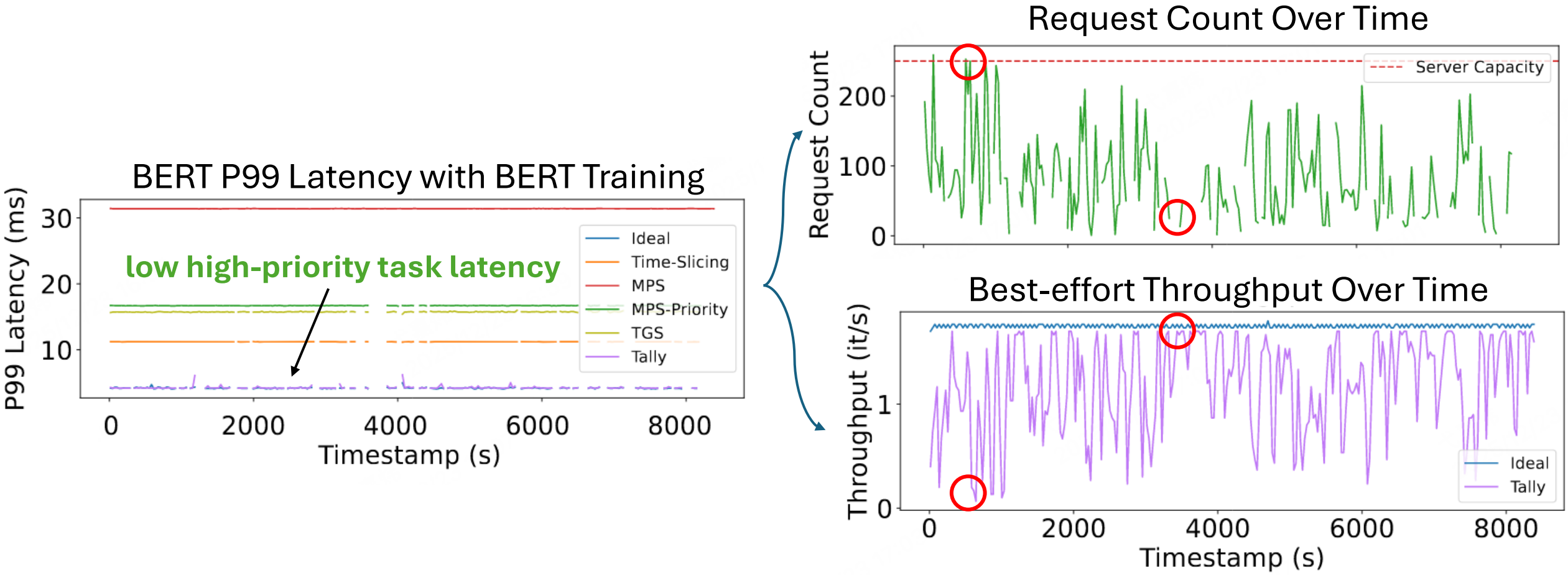
Inference	Training
ResNet50	ResNet50
BERT	PointNet
YOLOv6m	BERT
Llama2-7b	GPT2-Large
Stable Diffusion	PEGASUS
GPT-Neo	Whisper-v3

End-to-end Results



On average, Tally incurs only a **7% overhead** on the P99 latency of high-priority inference tasks, compared to **188% overhead** of the best baseline, while attaining more than **80% throughput**.

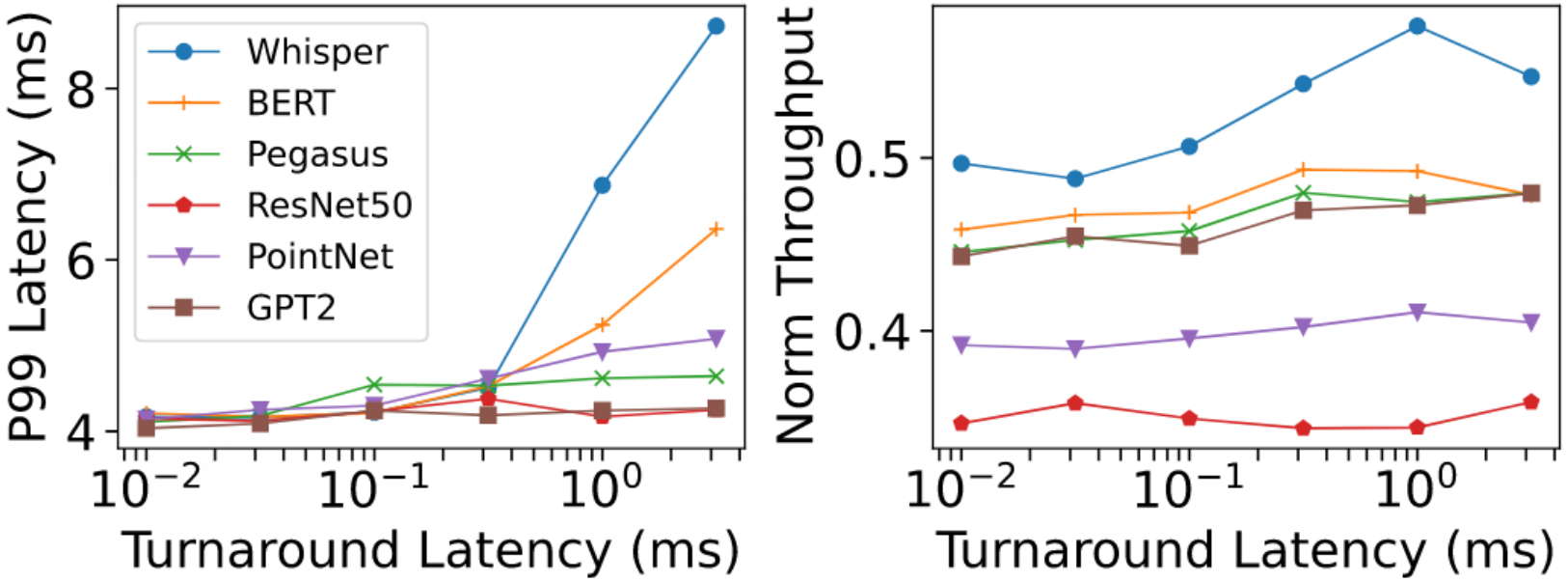
Responsiveness of Tally



Tally can **adaptively adjust the throughput** of the best-effort task in correspondence with the fluctuating traffic load to the high-priority inference task.

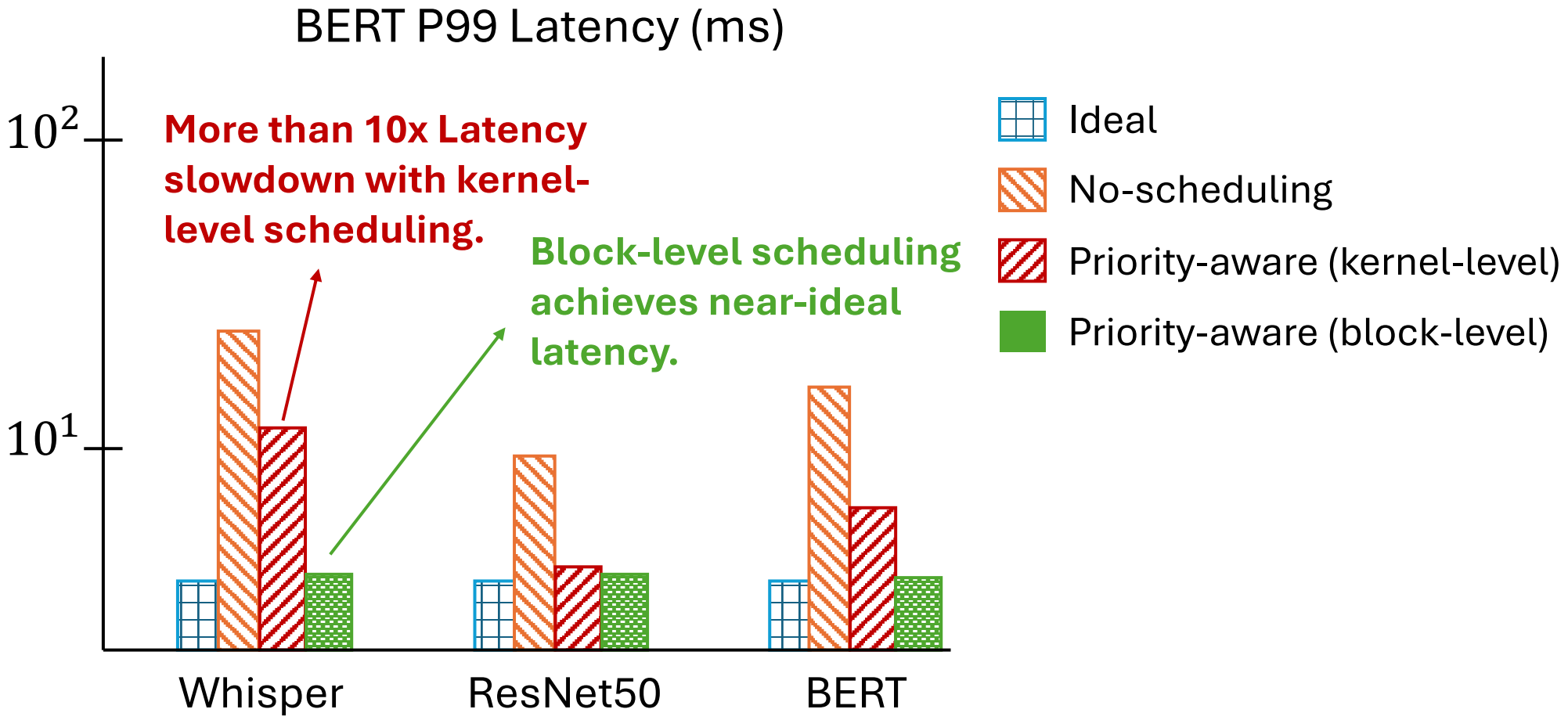
Selection of Turnaround Latency Threshold

P99 Latency and Throughput during BERT Inference Varying Turnaround Latency



Based on **empirical results**, Tally chooses threshold of **0.0316 ms** that provides the best balance between latency and throughput.

Importance of Block-level Scheduling (Ablation)



Overhead Analysis

- Virtualization (client-server)
  - Incurs an average overhead pf only **1%**, proving minimal performance impact
- Transformed kernel execution
  - Execution of block-level best-effort kernels causes **25%** average overhead
- Profiling
  - One-time online profiling for each kernel in one best-effort task complete within **minutes**, negligible given training workloads often run for hours/days

Summary

- Challenges of current GPU sharing solutions:
  - **High Integration costs** due to intrusive code modifications (**Intrusive**)
  - **Performance degradation** leading to SLA violations (**Bad Performance Isolation**)
  - **Limited workload compatibility** across applications (**Non-generalizable**)
- Method:
  - Implements **block-level scheduling primitives** for GPU virtualization
  - Ensures **effective performance isolation** in a **non-intrusive, task-agnostic** manner
- Results:
  - **7% overhead** vs **188%** in state-of-the-art GPU sharing
  - Maintains over **80% throughput** of the best-performing baseline

Remaining Problems

- Selection of **turnaround latency threshold** at block-level launch configuration is very **ad-hoc**
- Doesn’t compare with **REEF**
- Fundamentally **incompatible** with **CUDA Graph**, may encounter launch bubble issues on high-end GPUs (e.g., NVIDIA H100)