USENIX
ATC '25

GPreempt: GPU Preemptive Scheduling Made General and Efficient

Ruwen Fan, Tingxu Ren, Minhui Xie,

Shiwei Gao, Jiwu Shu, Youyou Lu

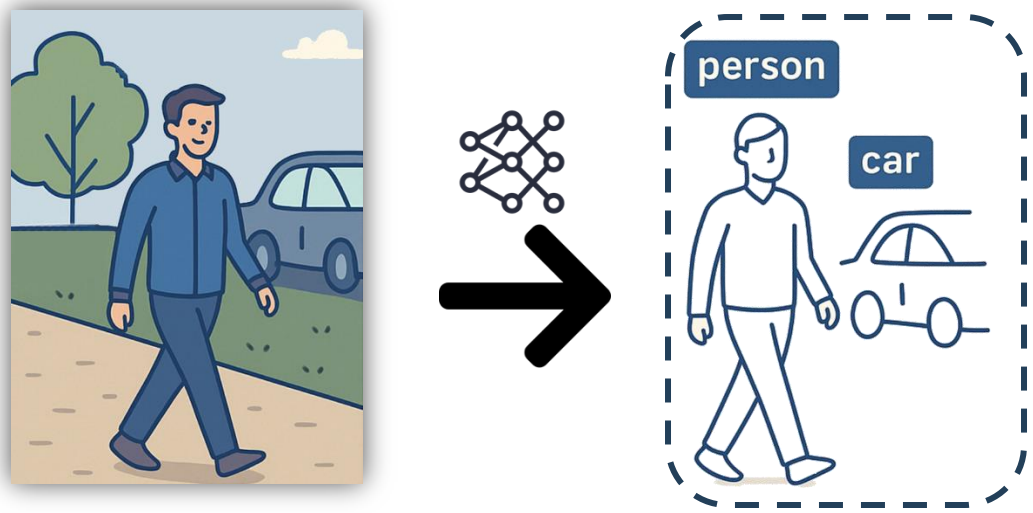
Tsinghua University

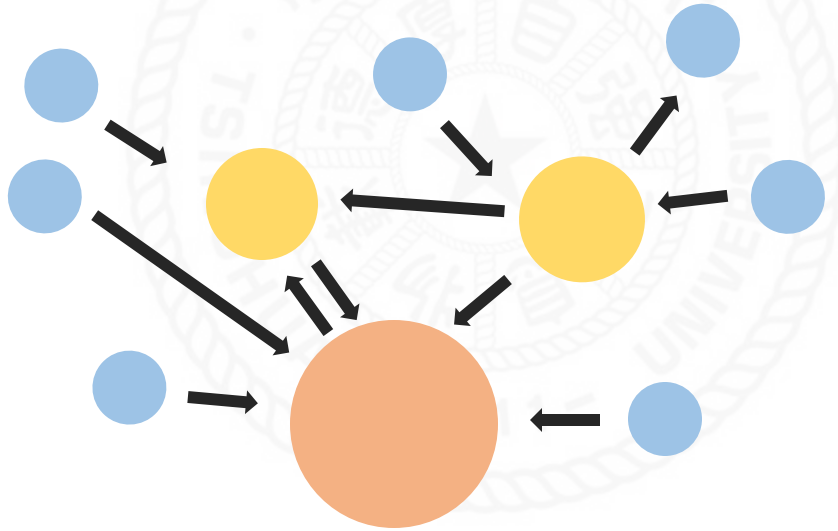


GPU Application

GPUs have become essential for a wide range of computing tasks

- ❖ Computer vision
- ❖ Machine learning
- ❖ Graph computing
- ❖ … …





GPU Sharing

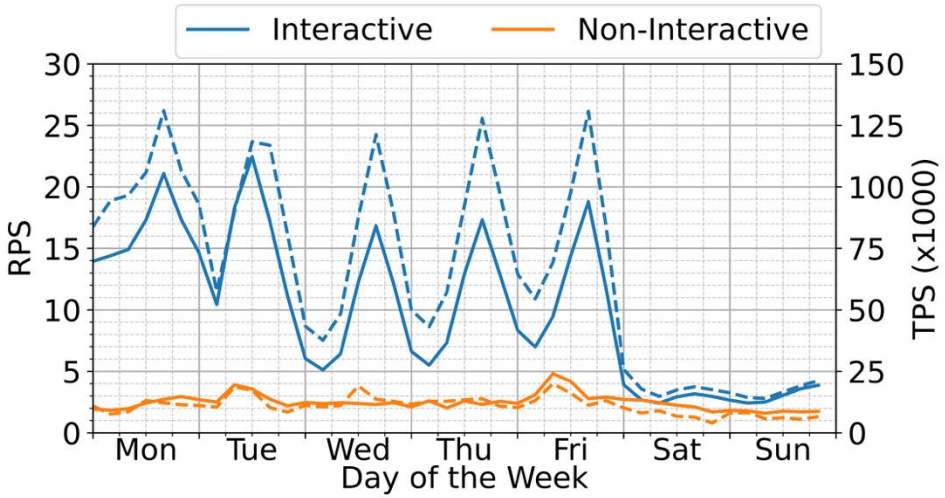
GPU applications can be divided into two categories

- Latency-Critical (LC) Applications
  - Real-time recommendation
  - Virtual reality
  - ……
- Best-Effort (BE) Applications
  - Scientific computing
  - Graph computing
  - Offline inference
  - …

GPU Sharing

GPU applications can be divided into two categories

- ❖ Latency-Critical (LC) Applications
  - ❖ Real-time recommendation
  - ❖ Virtual reality
  - ❖ ……
- ❖ Best-Effort (BE) Applications
  - ❖ Scientific computing
  - ❖ Graph computing
  - ❖ Offline inference
  - ❖ …

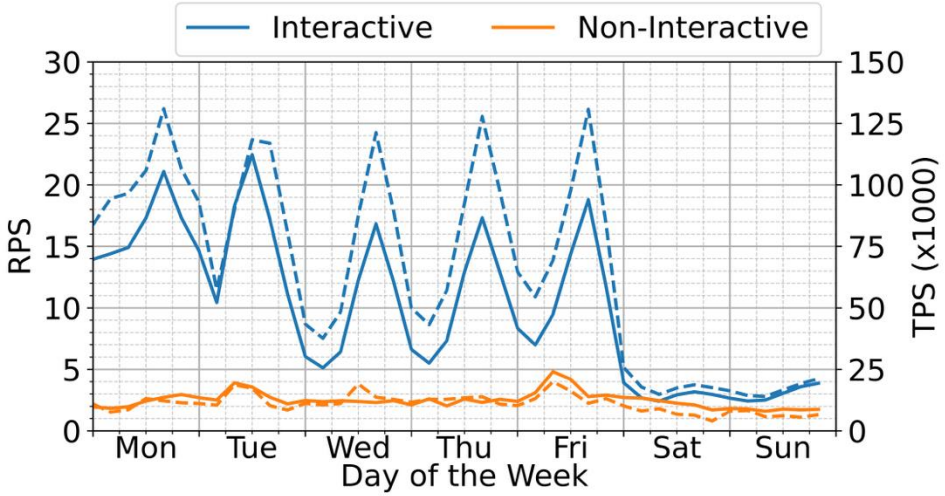


[1] Serving Models, Fast and Slow: Optimizing Heterogeneous LLM Inferencing Workloads at Scale, arXiv

GPU Sharing

GPU applications can be divided into two categories

- ❖ Latency-Critical (LC) Applications
  - ❖ Real-time recommendation
  - ❖ Virtual reality
  - ❖ ……
- ❖ Best-Effort (BE) Applications
  - ❖ Scientific computing
  - ❖ Graph computing
  - ❖ Offline inference
  - ❖ …



Co-locate different task
to improve GPU utilization

[1] Serving Models, Fast and Slow: Optimizing Heterogeneous LLM Inferencing Workloads at Scale, arXiv

GPU Sharing is not common

Interference between tasks, and potentially ***violating SLA guarantees***

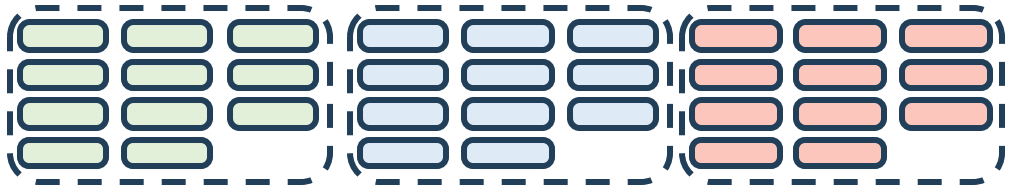
If we co-locate ***multiple task*** in….

GPU Sharing is not common

Interference between tasks, and potentially ***violating SLA guarantees***

If we co-locate ***multiple task*** in….

1: *M*ultiple streams of same context



BE kernels ***LC kernel***

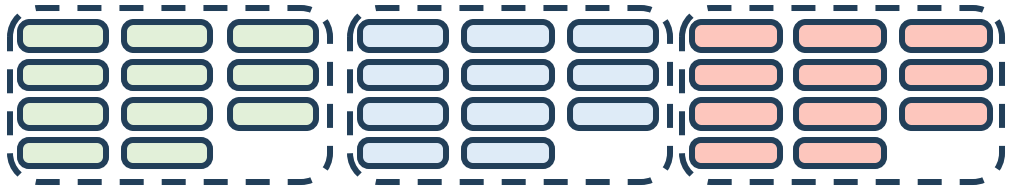
Streams provide no preemption
LC kernels will be blocked

GPU Sharing is not common

Interference between tasks, and potentially ***violating SLA guarantees***

If we co-locate ***multiple task*** in….

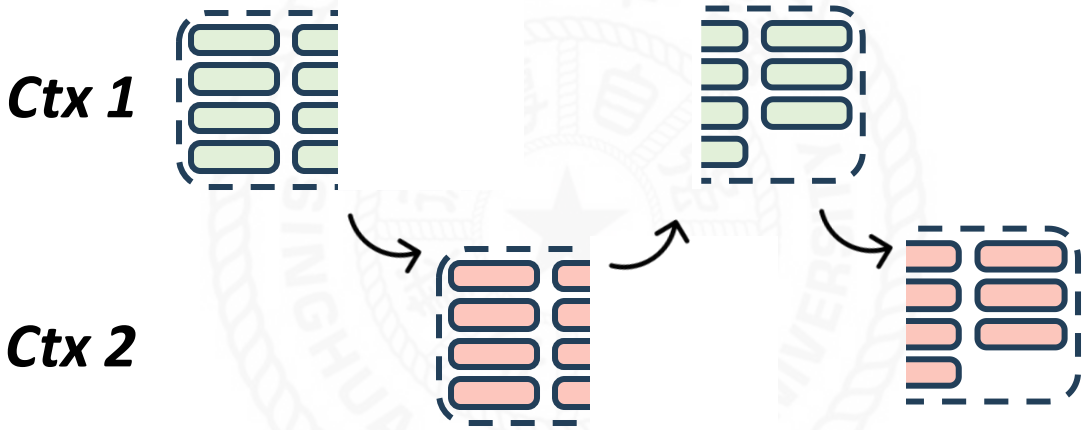
1*: M*ultiple streams of same context



BE kernels ***LC kernel***

Streams provide no preemption
LC kernels will be blocked

2*:* Different context

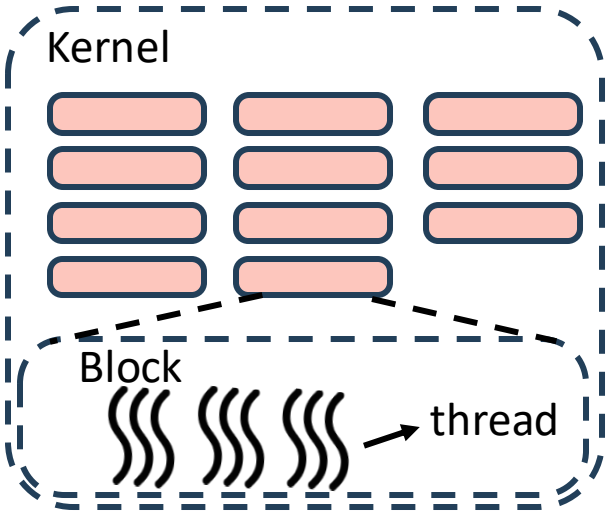


Task execution may be split into multiple fragments

GPU Sharing is not common

GPU programming paradigm is not suitable for scheduling

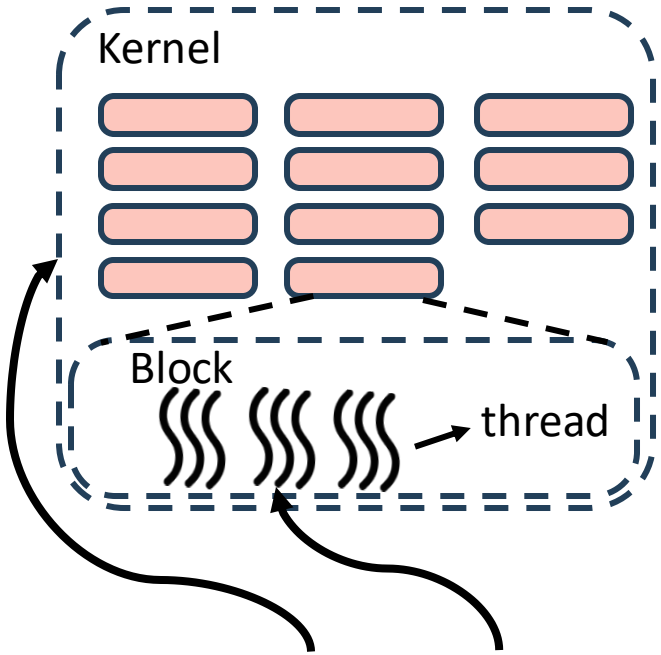
- ❖ Launch-and-wait model



GPU Sharing is not common

GPU programming paradigm is not suitable for scheduling

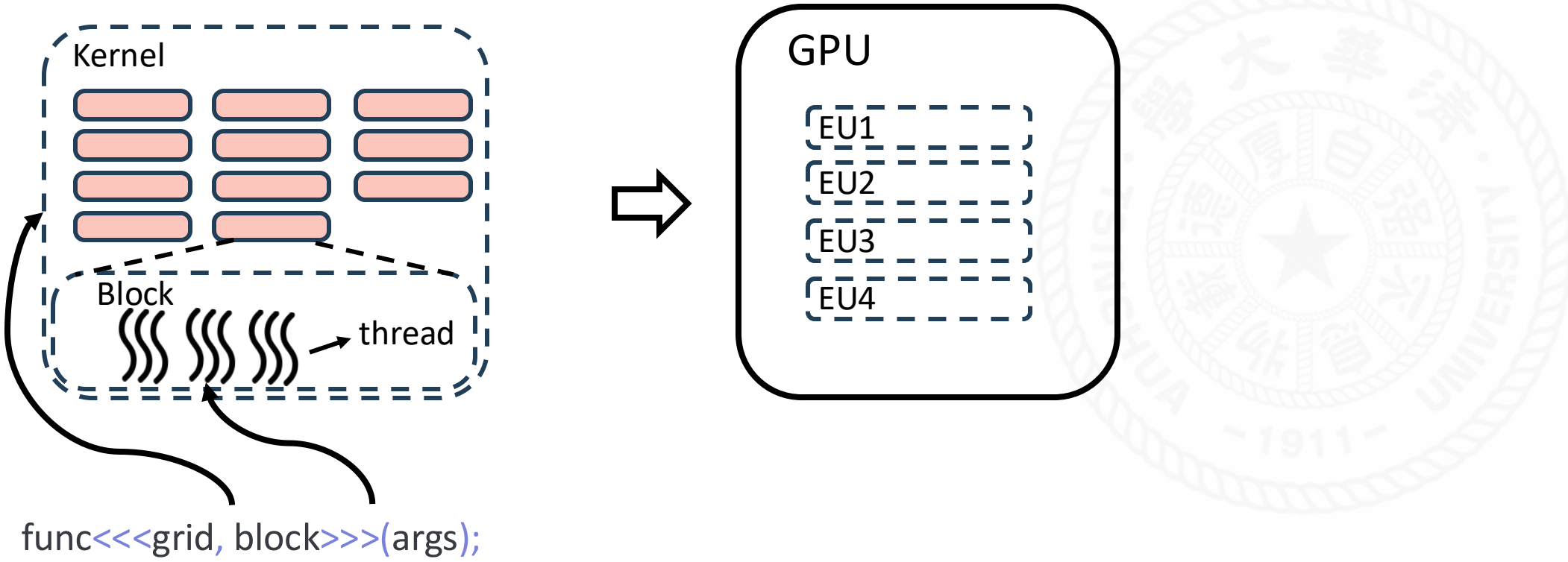
- ❖ Launch-and-wait model



GPU Sharing is not common

GPU programming paradigm is not suitable for scheduling

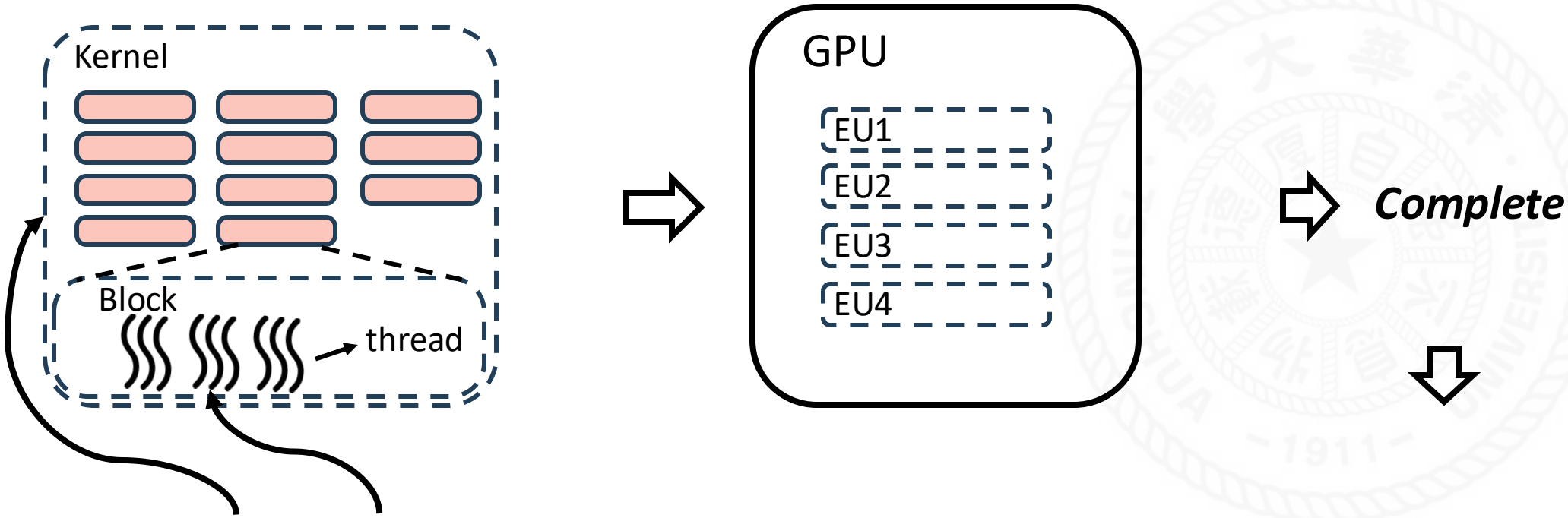
- ❖ Launch-and-wait model



GPU Sharing is not common

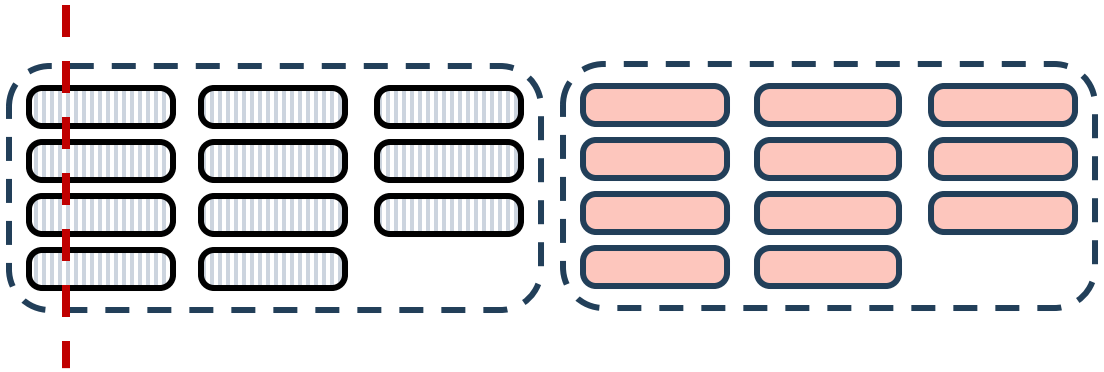
GPU programming paradigm is not suitable for scheduling

- ❖ Launch-and-wait model



Common paradigm: Wait-based preemption

❖ ***Check before launch*** a kernel (kernel level)



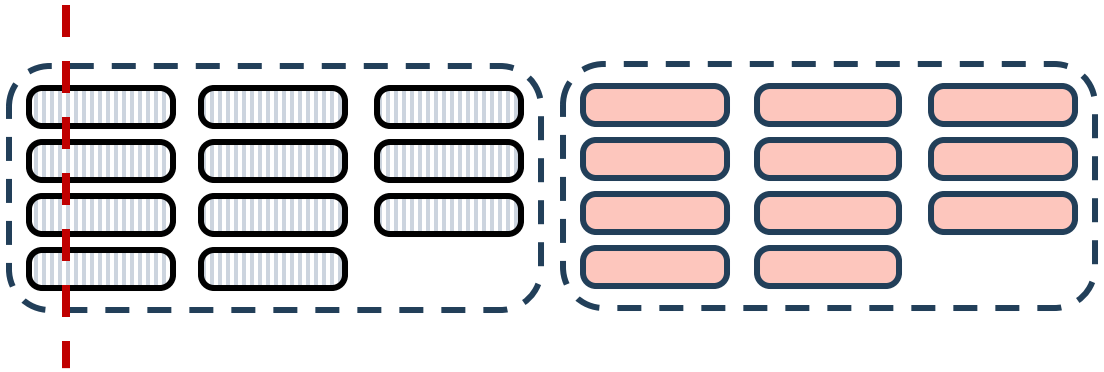
LC task arrival

Preempt when a kernel finishes

Latency: a kernel's lifetime

Common paradigm: Wait-based preemption

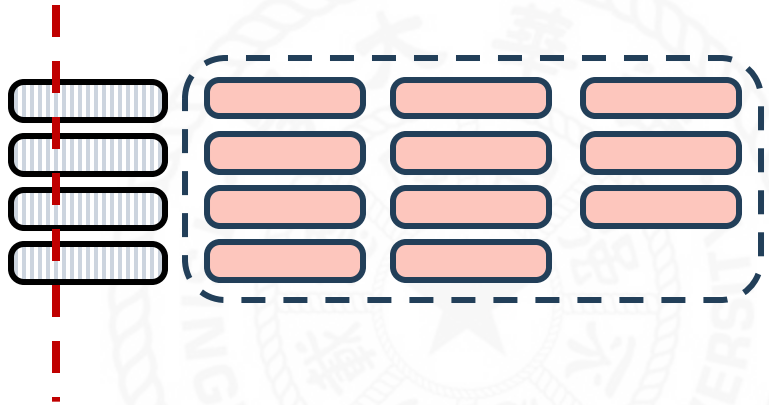
- ***Check before launch*** a kernel (kernel level)
- ***Check and exit*** if LC tasks arrive ***at the beginning*** of each block (block level)



LC task arrival

Preempt when a kernel finishes

Latency: a kernel's lifetime



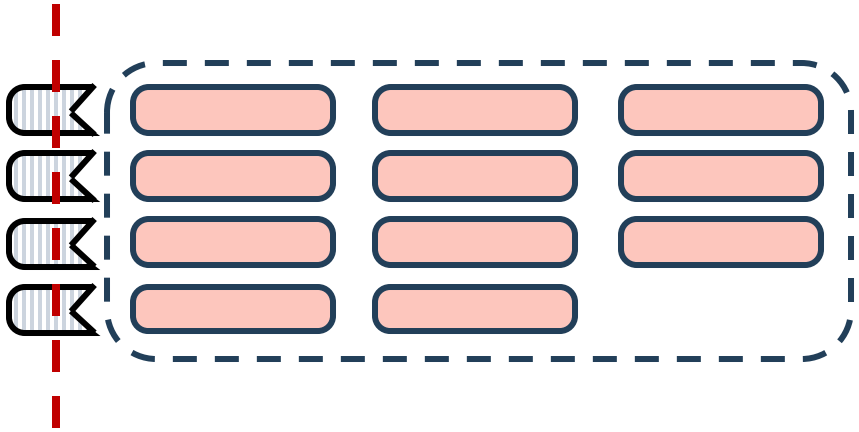
LC task arrival

Preempt when running blocks finish

Latency: a block's lifetime

Common paradigm: Reset-based preemption

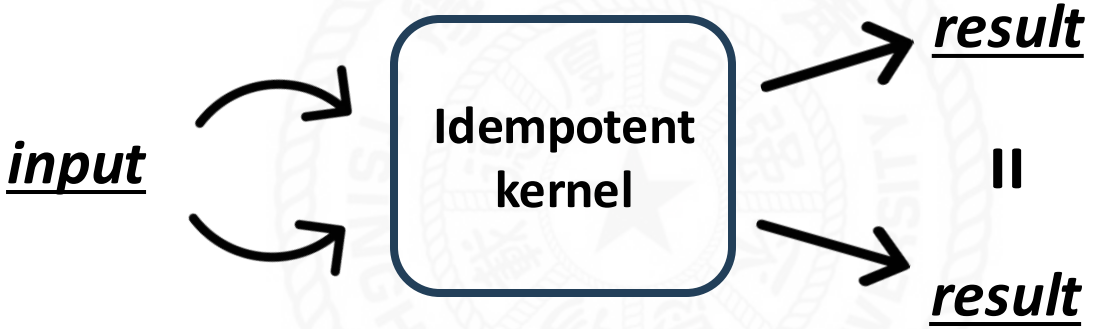
- Directly ***kill*** the BE kernel on the GPU
- Require ***Idempotent kernels***, which produce same result in repeated runs



LC task arrival

Exit anywhere

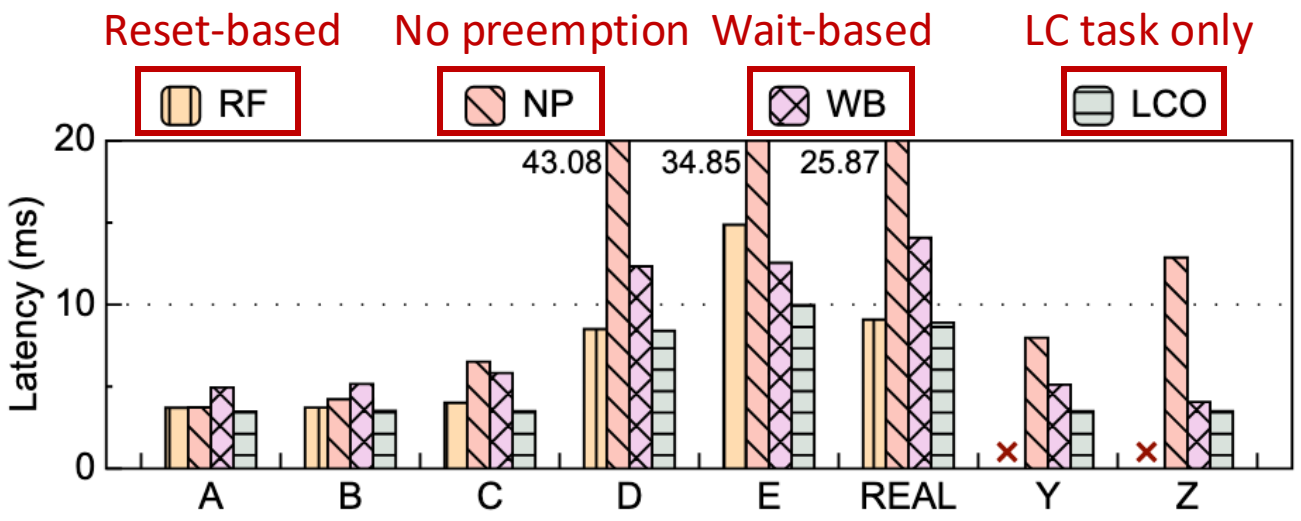
Latency: Near-zero



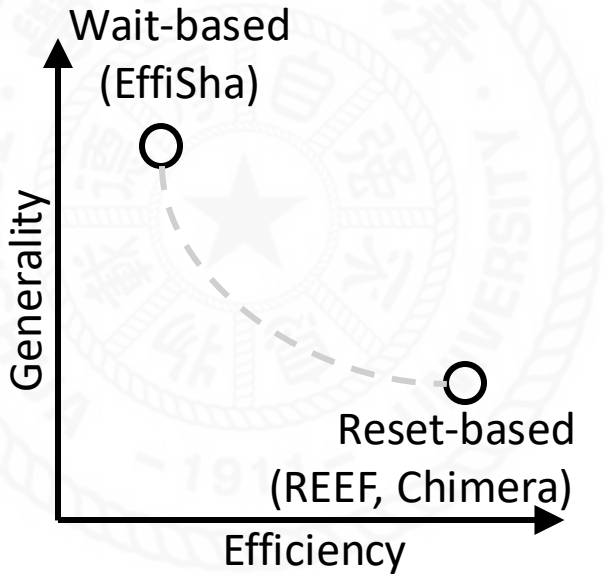
Tradeoff between existing approaches

Existing approaches make a tradeoff between ***generality*** and ***efficiency***

- ❖ Wait-based: Different kernel/block execution time, leading to unstable latency
- ❖ Reset-based: Limited generality due to idempotence requirement



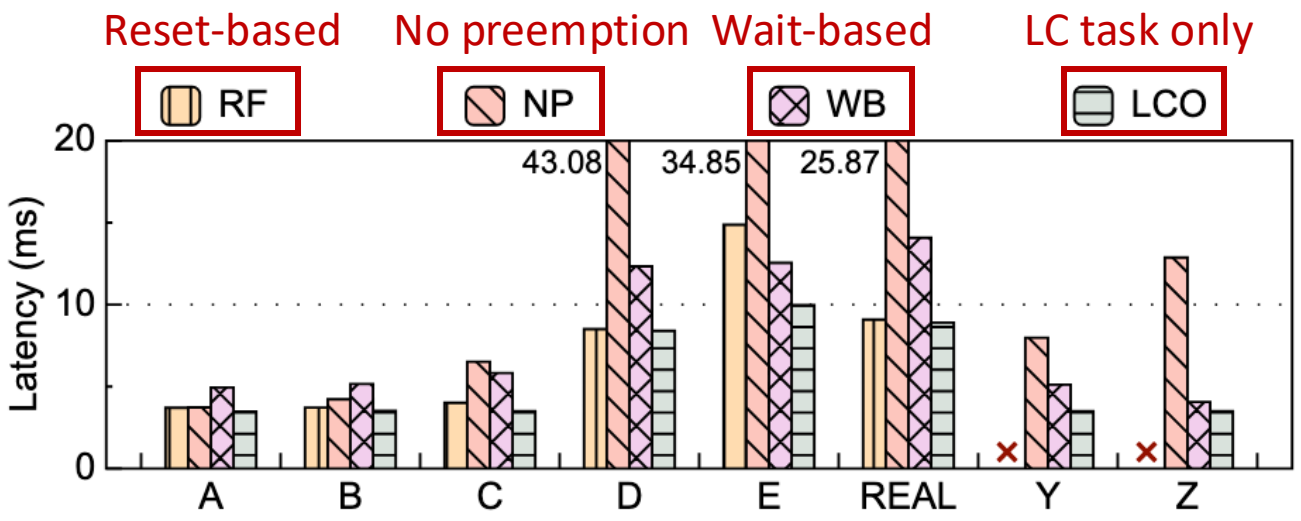
Preemption Latency of Existing approaches



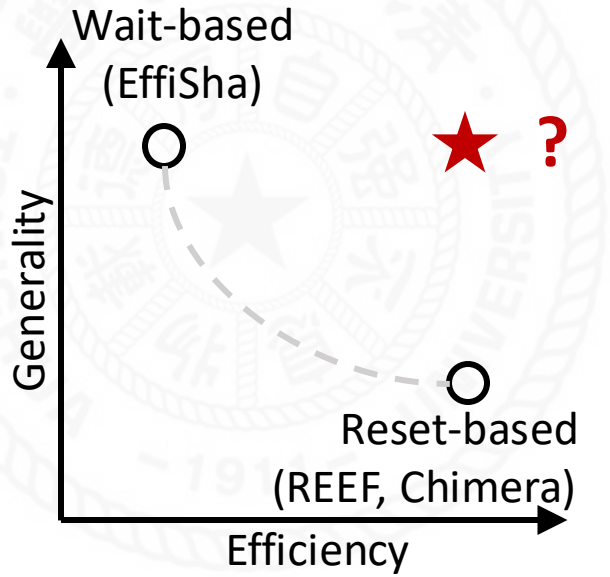
Tradeoff between existing approaches

Existing approaches make a tradeoff between ***generality*** and ***efficiency***

- ❖ Wait-based: Different kernel/block execution time, leading to unstable latency
- ❖ Reset-based: Limited generality due to idempotence requirement



Preemption Latency of Existing approaches



Key idea: Switch-based preemption

Can we break the tradeoff of generality and efficiency

- Generality: We shouldn't drop the context of existing task

Key idea: Switch-based preemption

Can we break the tradeoff of generality and efficiency

- Generality: We shouldn’t drop the context of existing task
- Efficiency: We shouldn’t wait for the kernel / blocks to finish

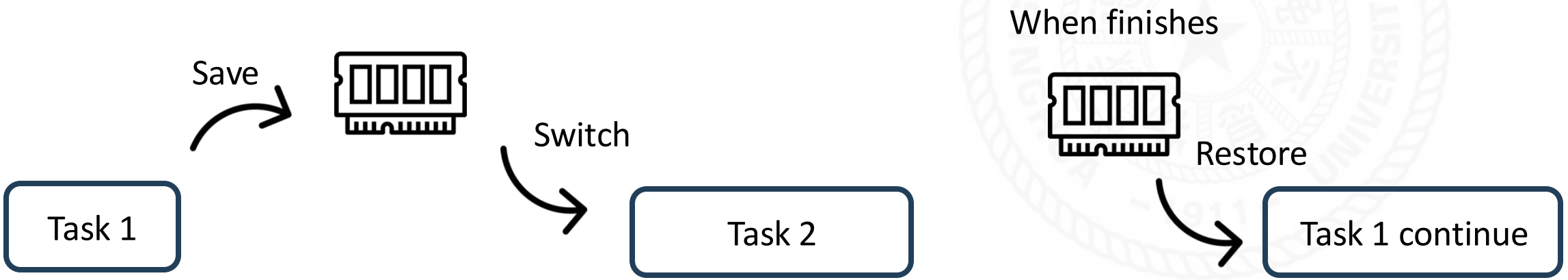


Key idea: Switch-based preemption

Can we break the tradeoff of generality and efficiency

- Generality: We shouldn't drop the context of existing task
- Efficiency: We shouldn't wait for the kernel / blocks to finish

Learn a lesson from the OS, ***context switch*** offers a solution

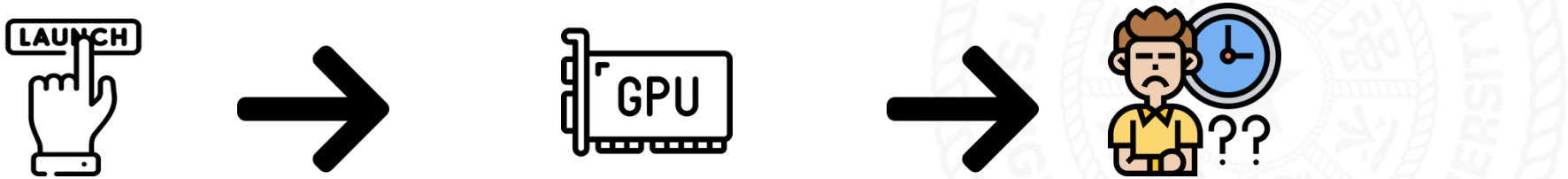


Challenge: GPU scheduling is opaque to users (1)

Challenge: GPU scheduling is opaque to users (1)

How we use the GPU?

- We just ***launch kernels*** and ***synchronize*** to complete



Challenge: GPU scheduling is opaque to users (1)

How we use the GPU?

- We just ***launch kernels*** and ***synchronize*** to complete
- GPU runtimes provide ***no scheduling interface***



yield() / schedule()

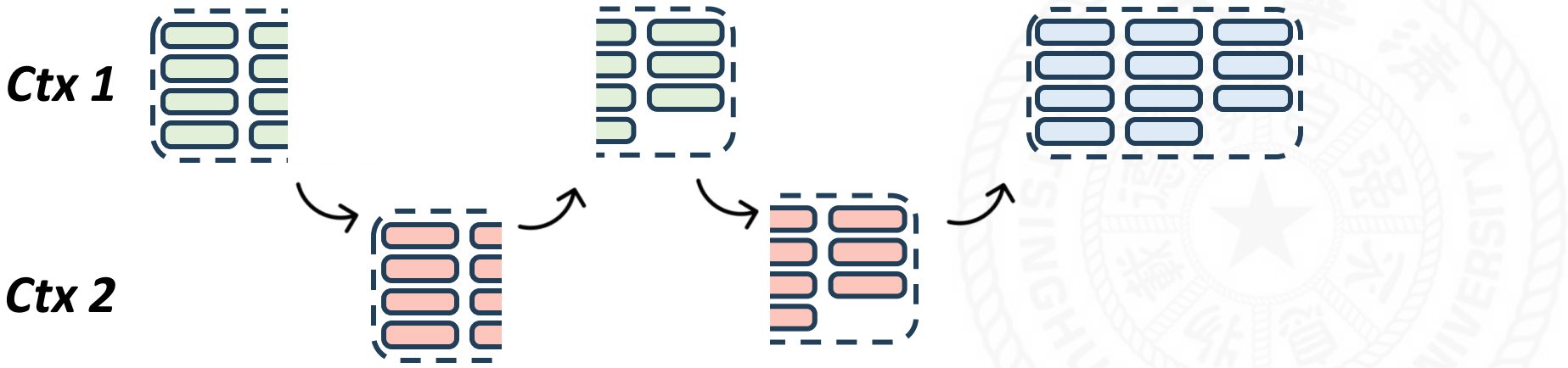
Observation: Context switch is inherent in GPU

When we directly run two tasks on the same GPU?

Observation: Context switch is inherent in GPU

When we directly run two tasks on the same GPU?

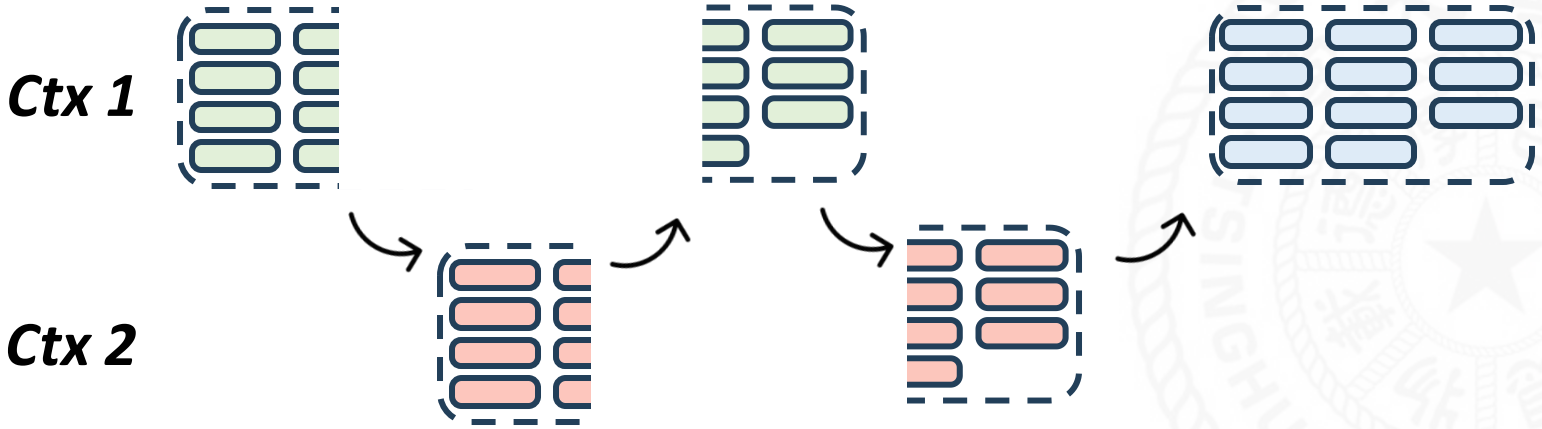
- The GPU runs two tasks alternately in a ***timeslice round-robin manner***



Observation: Context switch is inherent in GPU

When we directly run two tasks on the same GPU?

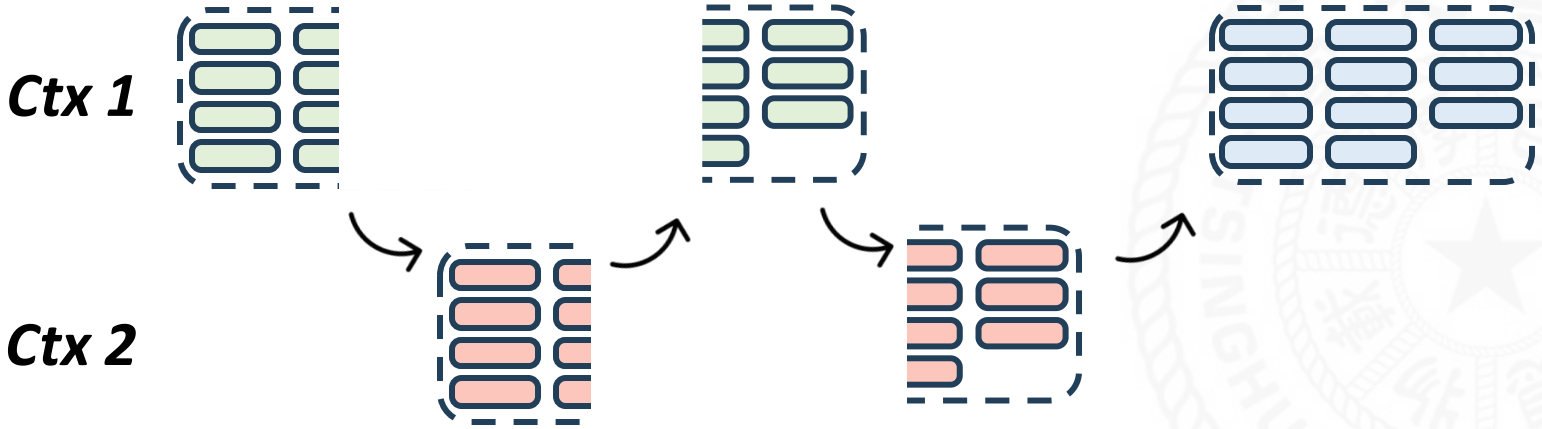
- The GPU runs two tasks alternately in a ***timeslice round-robin manner***
- When there are only one task, the GPU will continually run it



Observation: Context switch is inherent in GPU

When we directly run two tasks on the same GPU?

- The GPU runs two tasks alternately in a ***timeslice round-robin manner***
- When there are only one task, the GPU will continually run it



Context switch is an inherent feature in the GPU,
which can be leveraged to preempt the task

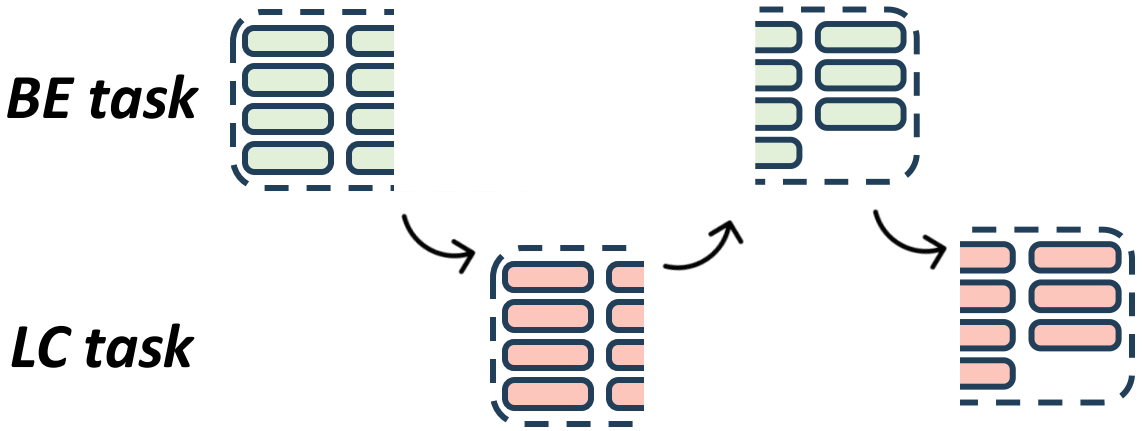
GPreempt: Timeslice-based Preemption

Repurpose the GPU task rotation mechanism to achieve preemption?

GPreempt: Timeslice-based Preemption

Repurpose the GPU task rotation mechanism to achieve preemption?

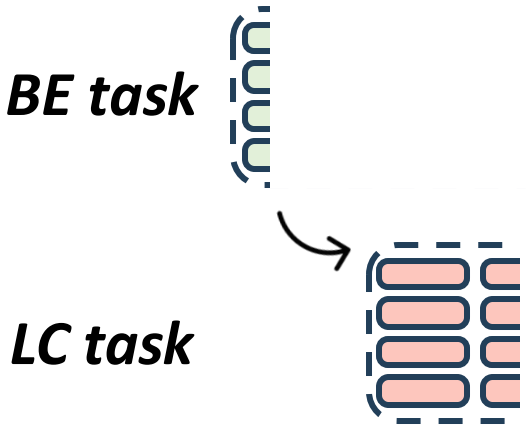
- ❖ GPU switch between task according to a predefined timeslice



GPreempt: Timeslice-based Preemption

Repurpose the GPU task rotation mechanism to achieve preemption?

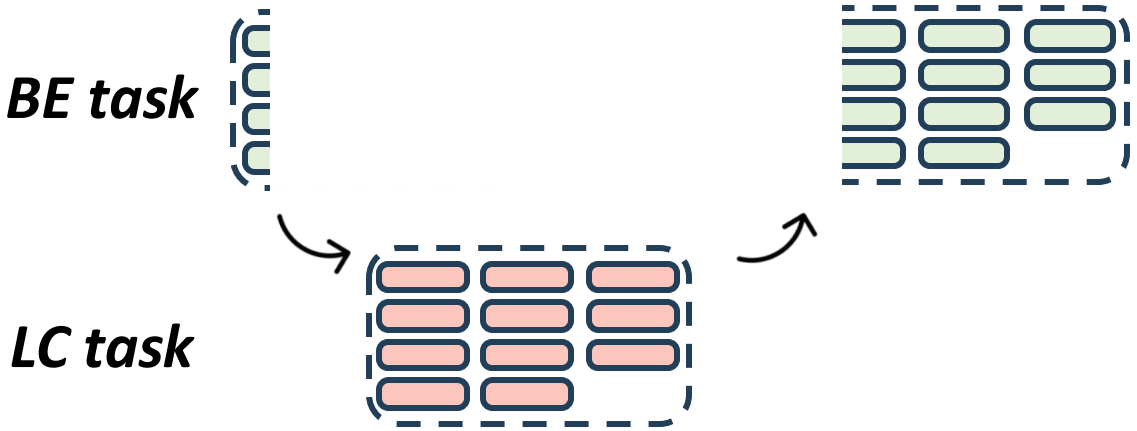
- GPU switch between task according to a predefined timeslice
  - Set the BE task timeslice to the minimum, almost 0



GPreempt: Timeslice-based Preemption

Repurpose the GPU task rotation mechanism to achieve preemption?

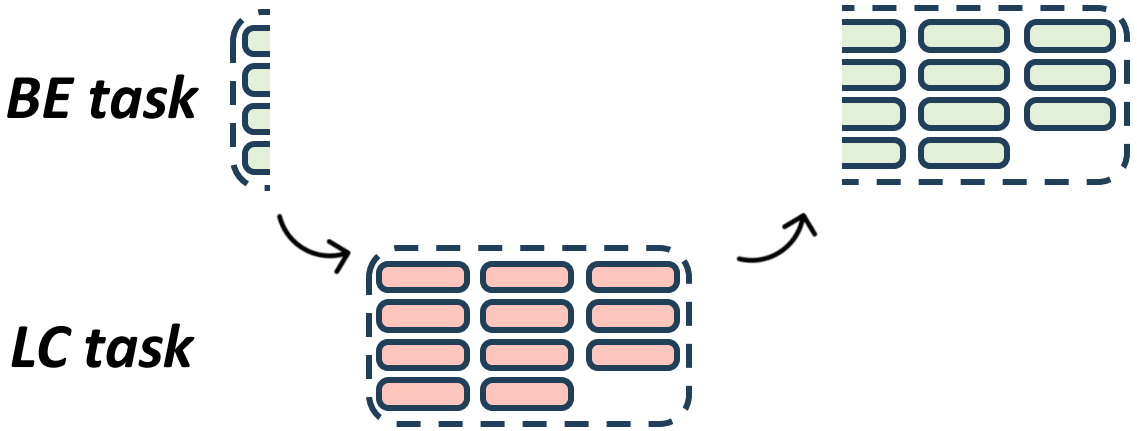
- GPU switch between task according to a predefined timeslice
  - Set the BE task timeslice to the minimum, almost 0
  - Expand the timeslice of the LC task beyond its lifetime



GPreempt: Timeslice-based Preemption

Repurpose the GPU task rotation mechanism to achieve preemption?

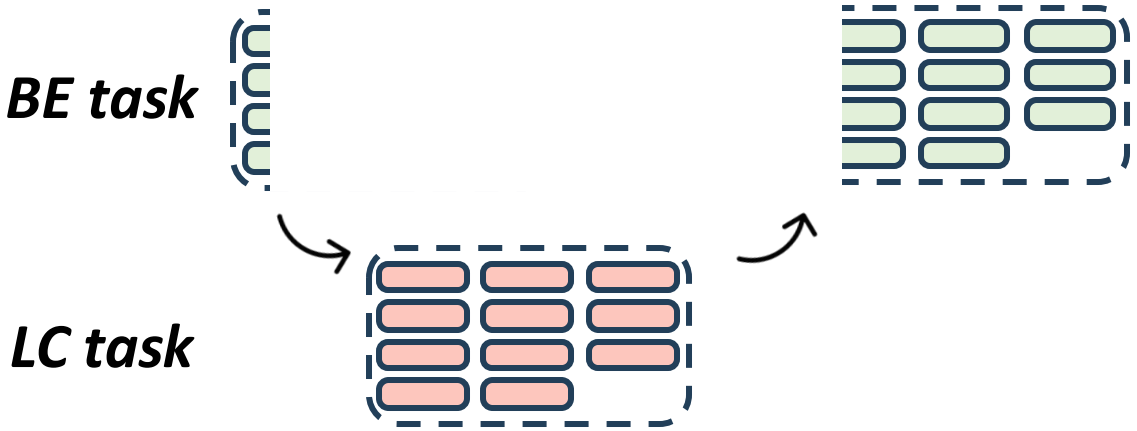
- GPU switch between task according to a predefined timeslice
  - Set the BE task timeslice to the minimum, almost 0
  - Expand the timeslice of the LC task beyond its lifetime
- By modifying the driver code, we are able to change the timeslice



GPreempt: Timeslice-based Preemption

Repurpose the GPU task rotation mechanism to achieve preemption?

- GPU switch between task according to a predefined timeslice
  - Set the BE task timeslice to the minimum, almost 0
  - Expand the timeslice of the LC task beyond its lifetime
- By modifying the driver code, we are able to change the timeslice

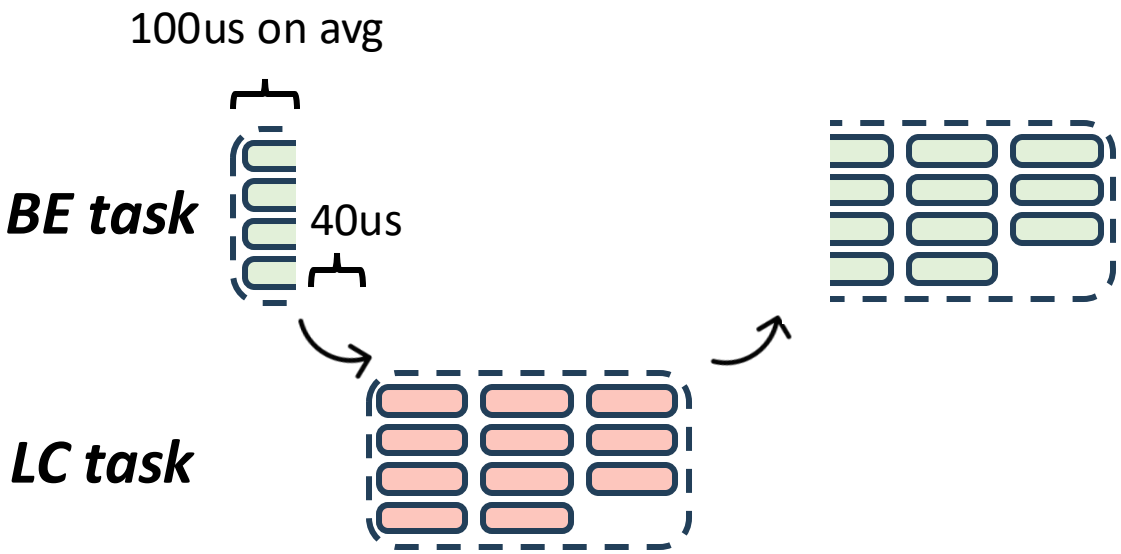


1. Switch is guaranteed by the hardware
2. Adjusting timeslice needs no modification to the userkernel code

Challenge: High preemption overhead (2)

Preemption overhead comes from

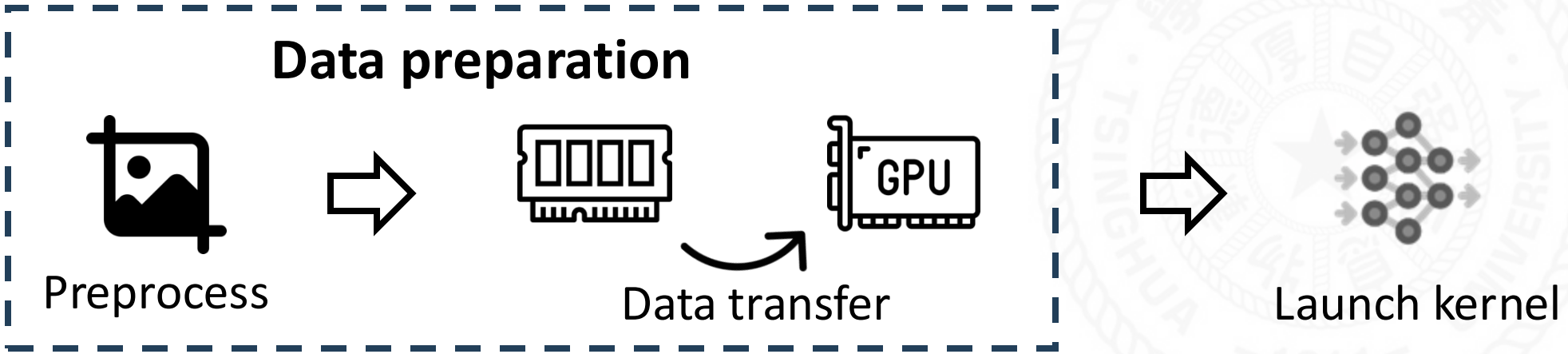
- Minimal timeslice for the BE task
  - The minimal GPU timeslice is about 200us, LC task will wait 100us on avg
- Massive compute contexts make switch expensive
  - E.g. 44MB context of NVIDIA A100 GPU take 40us to save



Observation: GPU computing paradigm

How we ***offload a task to GPU***?

- Preprocessing data (often on CPU)
- Transfer data to GPU before launch kernels (GPU DMA/copy engine)
- Launch kernels (GPU compute engine)

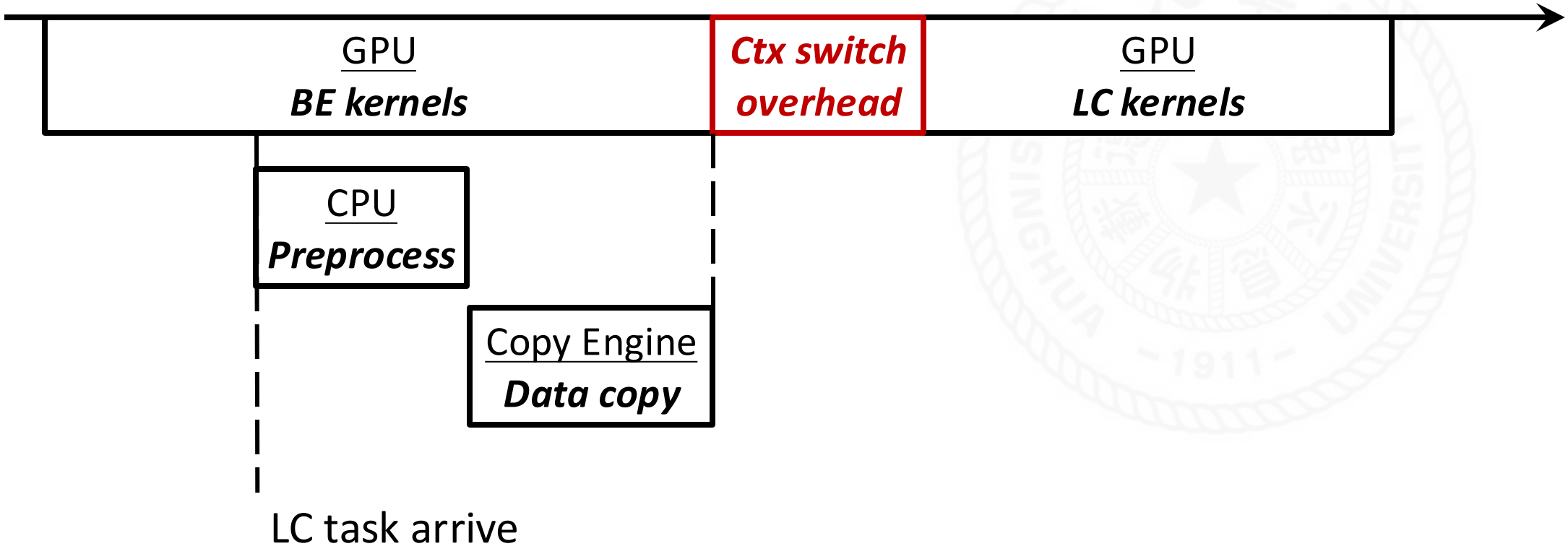


Not all the procedures are done on GPU

GPreempt: Hint-based Pre-preemption

Data preparation offers predictive signals for imminent LC kernel arrival

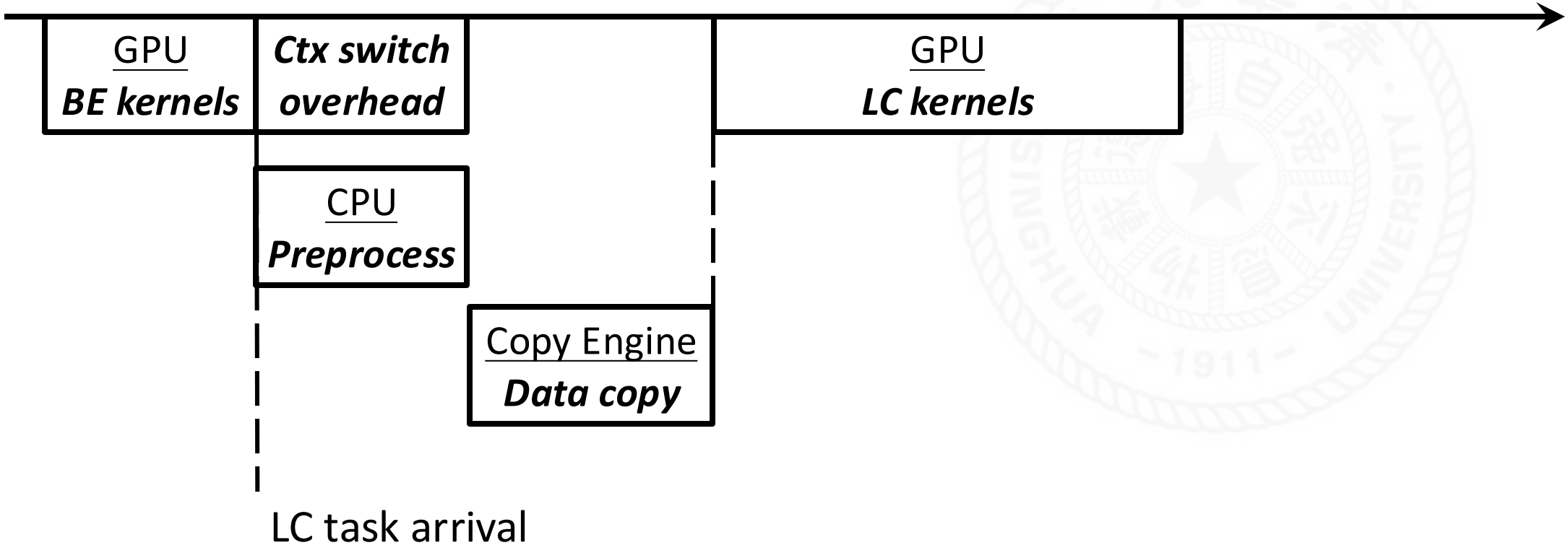
- How can we preempt BE before the LC kernels launch?



GPreempt: Hint-based Pre-preemption

Preparation offer predictive signals for imminent LC kernel arrival

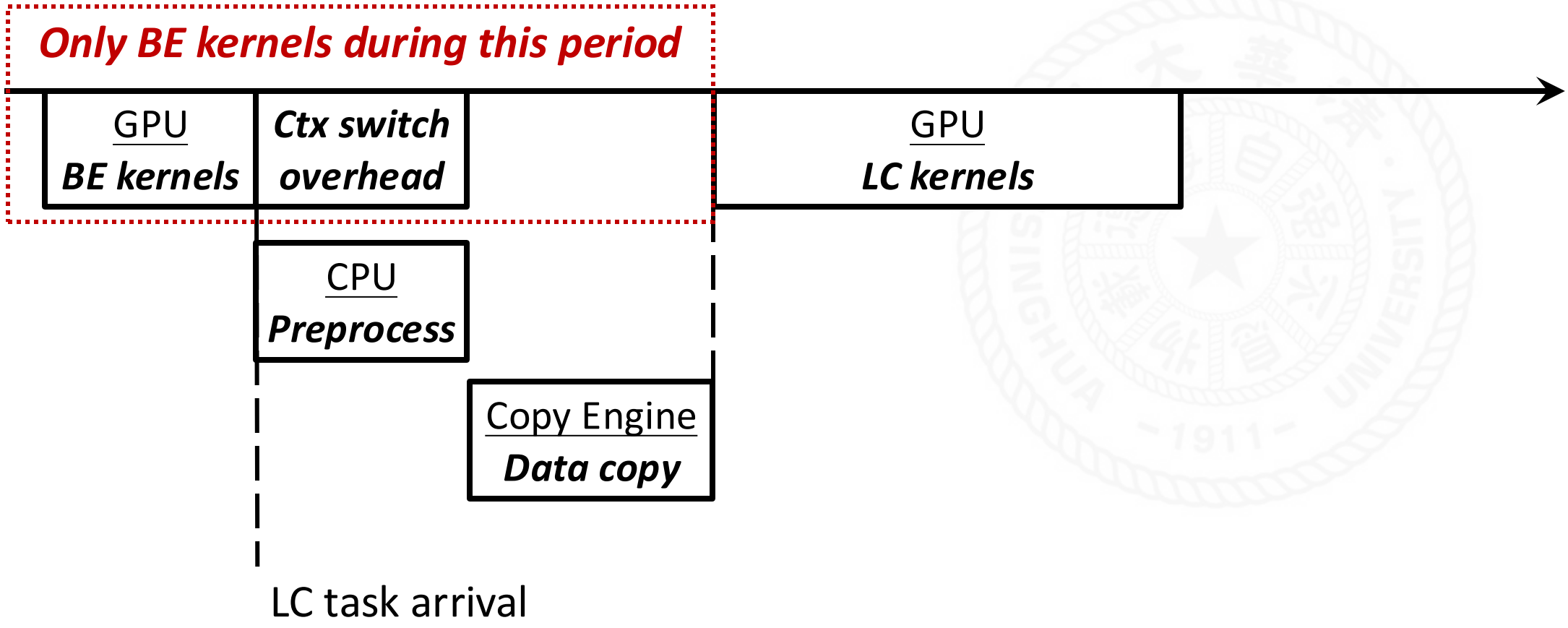
- How can we preempt BE before the LC kernels launch?



GPreempt: Hint-based Pre-preemption

Preparation offer predictive signals for imminent LC kernel arrival

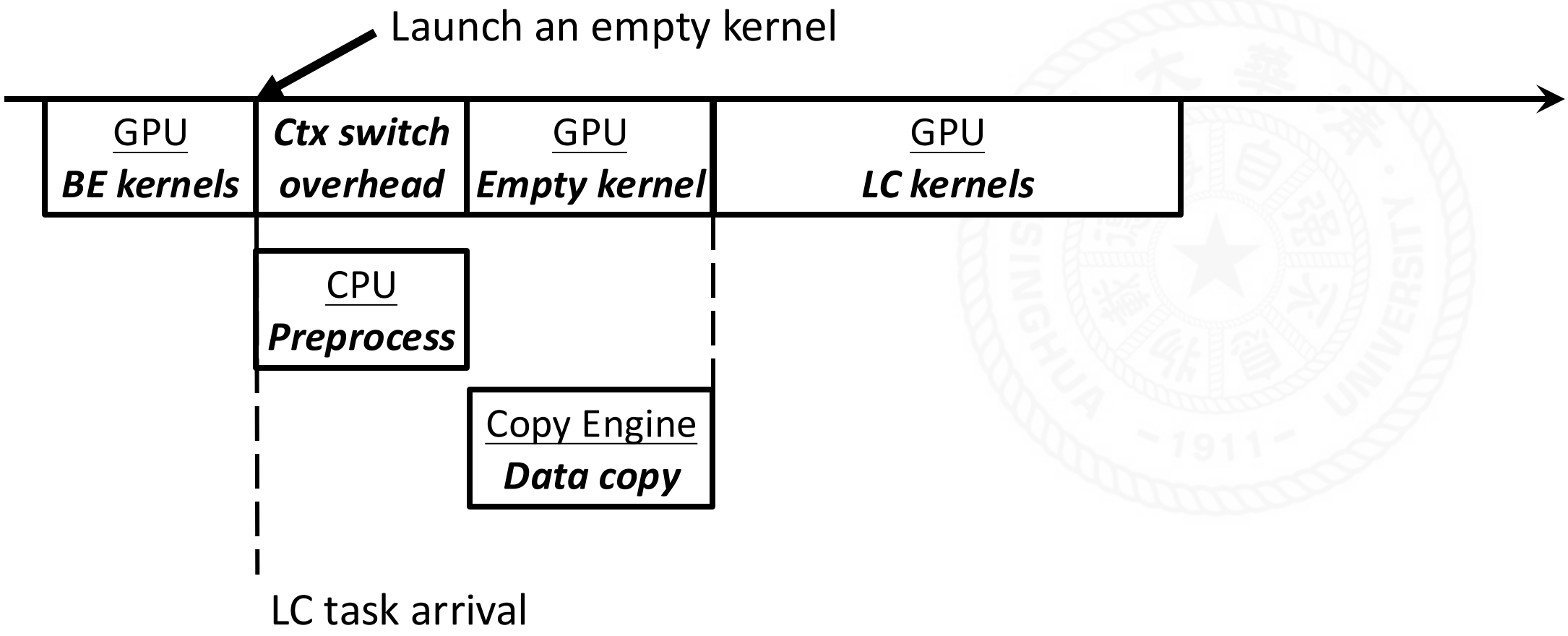
- How can we preempt BE before the LC kernels launch?



GPreempt: Hint-based Pre-preemption

Preparation offer predictive signals for imminent LC kernel arrival

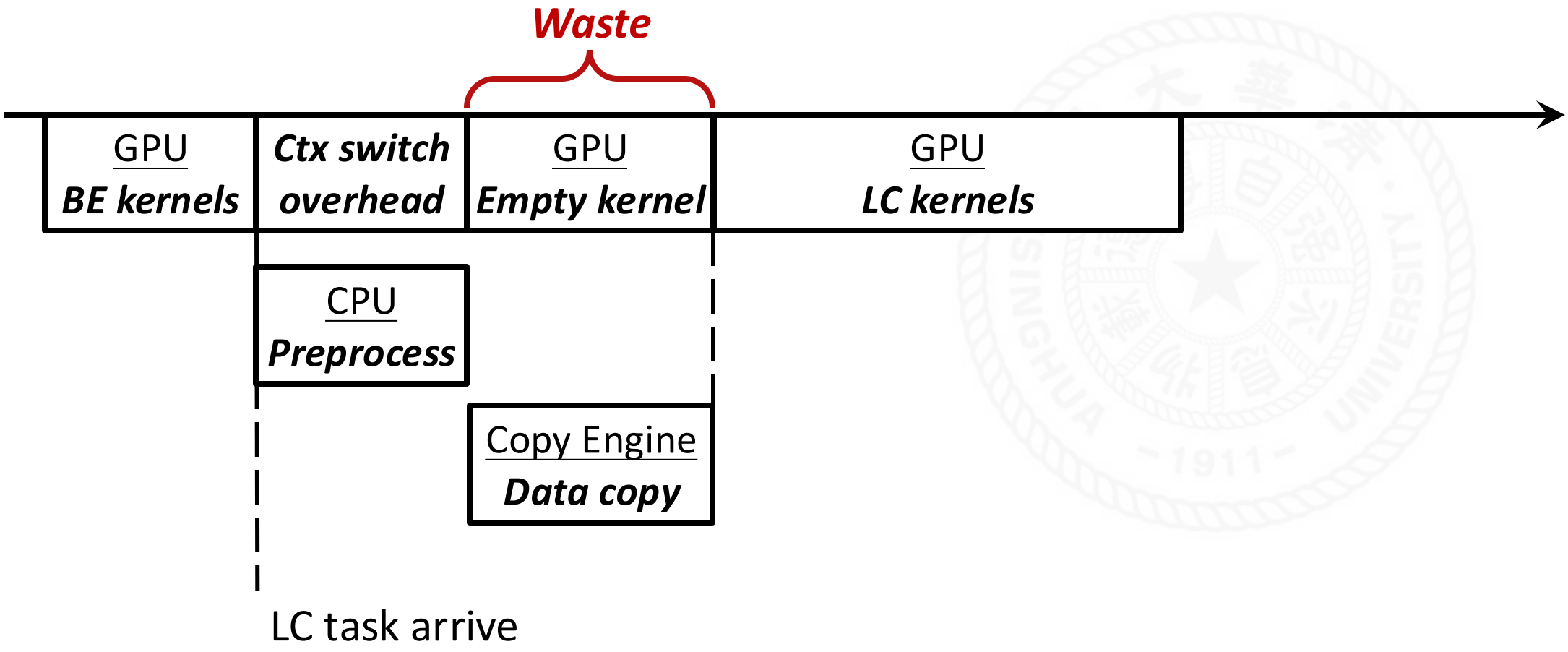
- How can we preempt BE before the LC kernels launch?



GPreempt: Hint-based Pre-preemption

Preempt in advance, but not too much

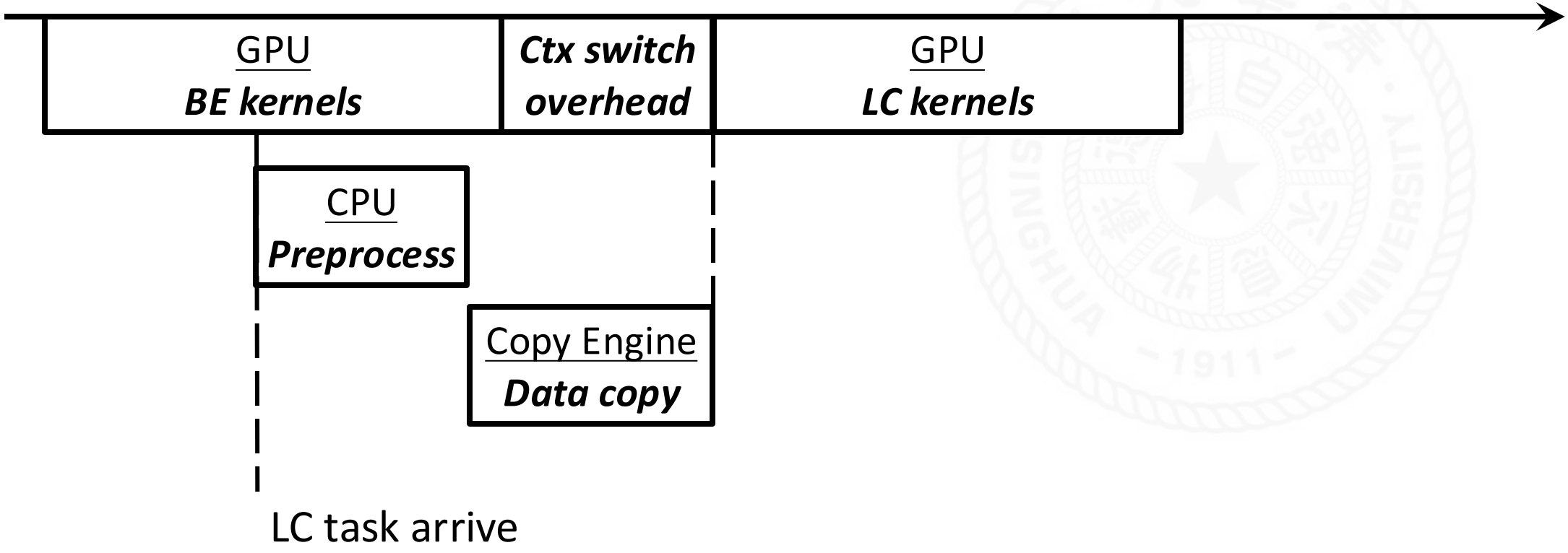
- The empty kernel will occupy the GPU and prevent BE task execution



GPreempt: Hint-based Pre-preemption

***Scheduled pre-preemption*:** Prepare in advance, but not too much

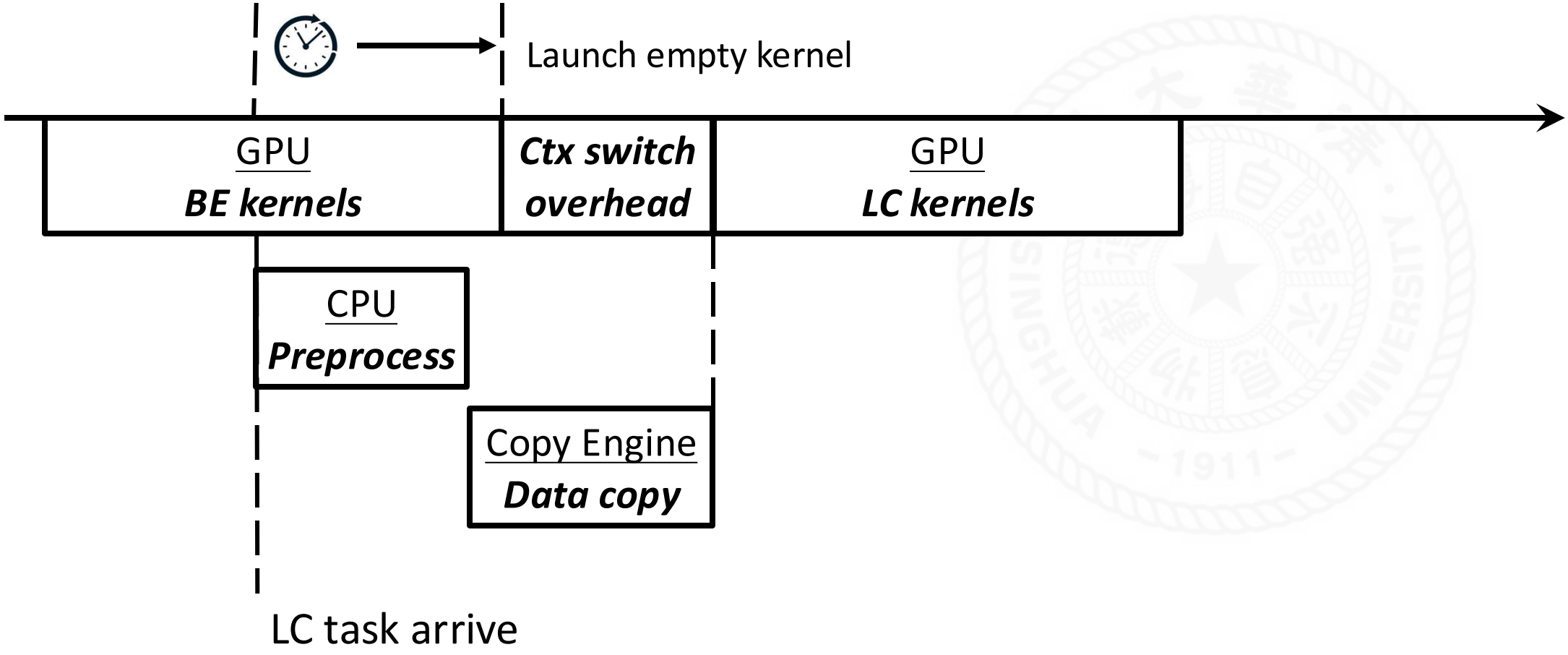
- Use a background thread to launch the preemption kernel



GPreempt: Hint-based Pre-preemption

***Scheduled pre-preemption*: Prepare in advance, but not too much**

- Use a background thread to launch the preemption kernel



Evaluation Setup

Hardware Platform

- Our server machine: Intel(R) Xeon(R) Gold 5420+ CPU, 256GB DRAM
  - NVIDIA A100-40GB GPU
  - AMD Instinct MI100 GPU

Target comparisons

- **LC-task only baseline:** The ideal latency boundary
- **No preemption:** The ideal throughput boundary
- **Wait-based preemption:** Including sequential execution and block level preemption
- **Reset-based preemption**

Evaluation Setup

Workloads

- **DISB:** 6 typical DNN inference workloads
- **Scientific computing:** Weather simulation
- **Graph computing:** 4 typical graph workloads

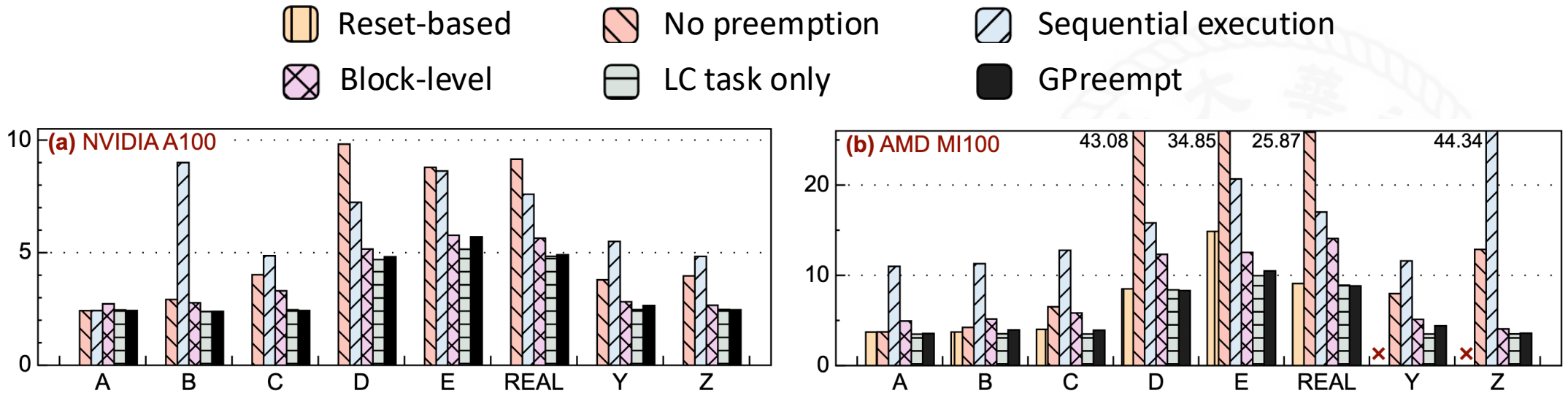
} Stateful, Non-idempotent

WL	LC Tasks	LC Rate	BE Tasks	BE Rate
A	VGG	100 (C)	ResNet	100 (C)
B	VGG	220 (C)	ResNet	220 (C)
C	VGG	100 (C)	DM*	100 (C)
D	DM*	20 (C)	DM*	100 (C)
E	DM*	20 (P)	DM*	100 (P)
REAL	DM*	Trace	DM*	Trace
Y	VGG	100 (C)	WS	150 (C)
Z	VGG	100 (C)	GM**	100 (C)

DM: {ResNet, DenseNet, VGG, Inception, BERT}
GM: {BFS, SSSP, PageRank, CC}.
(C): Constant rate, (P): Poisson rate. "100 (C)": 100 requests/sec per task.

Overall performance

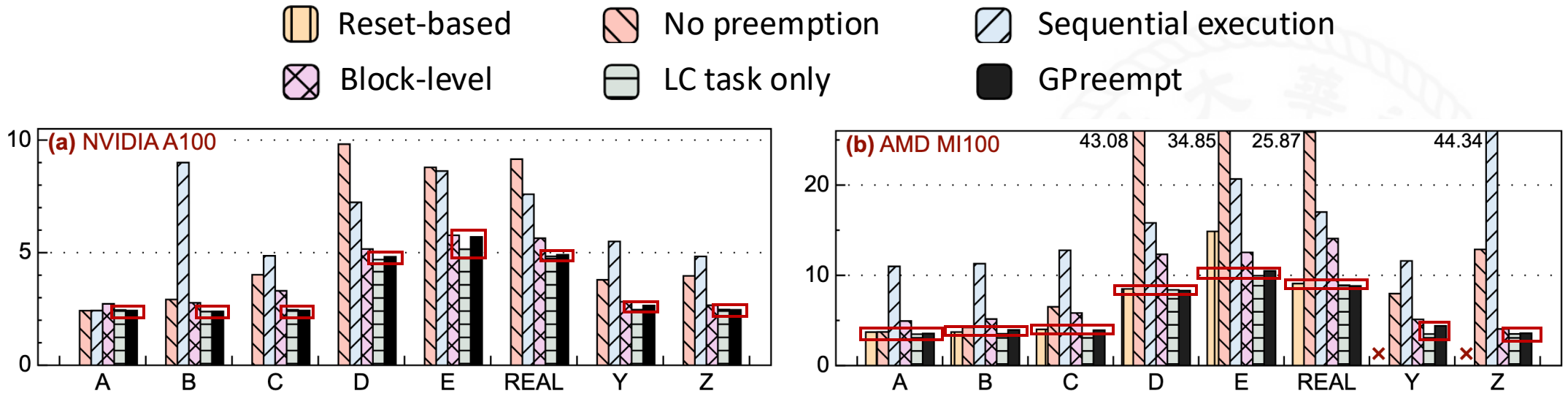
- Co-locate LC DNN inference task with different BE tasks
- Compare the latency of the LC task



End-to-end Latency of LC Tasks

Overall performance

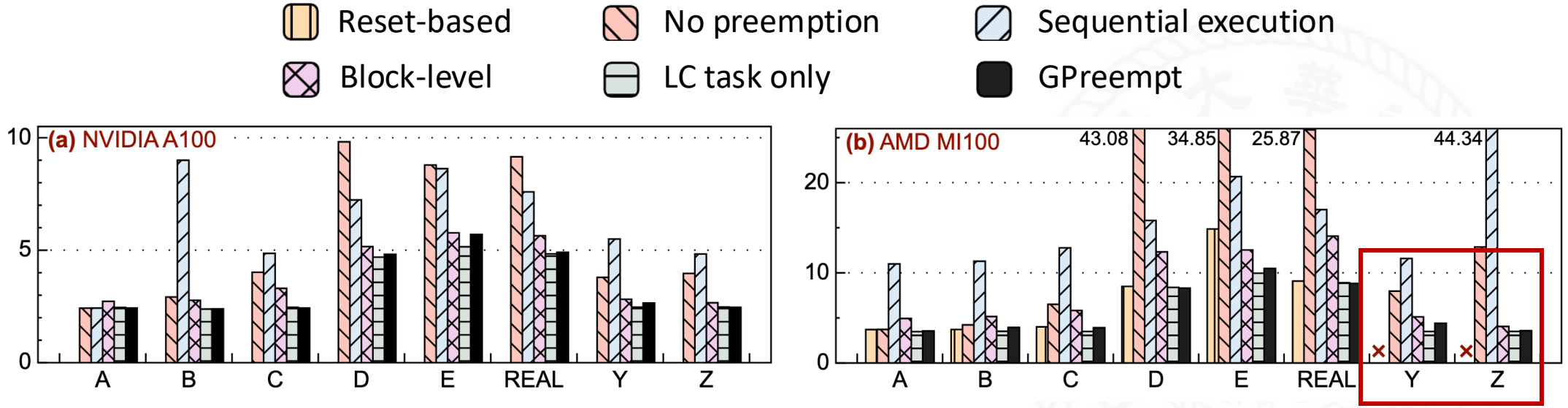
- Co-locate LC DNN inference task with different BE tasks
- Compare the latency of the LC task



End-to-end Latency of LC Tasks

Overall performance

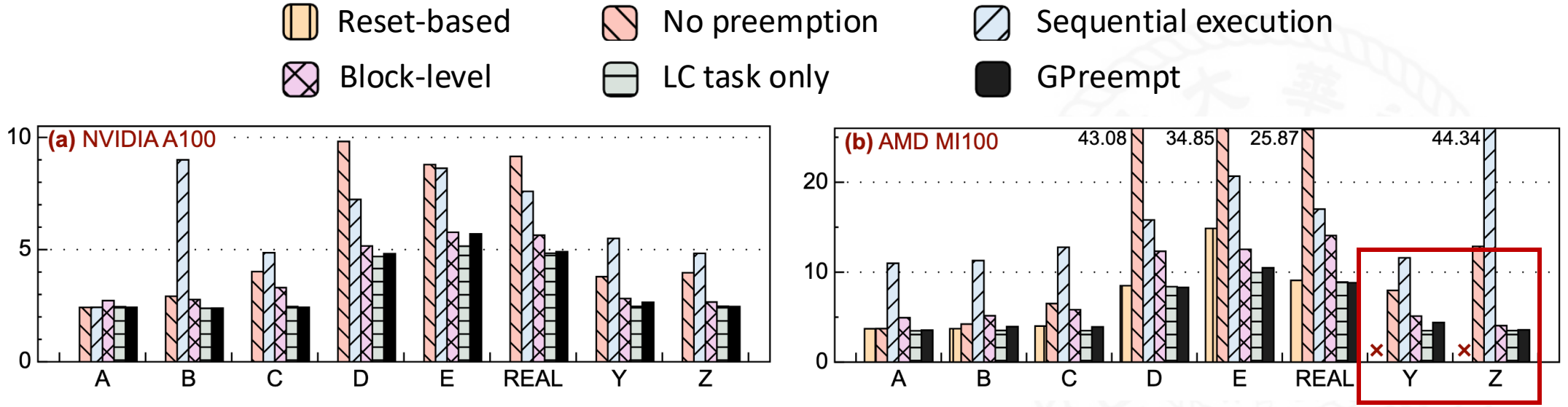
- Co-locate LC DNN inference task with different BE tasks
- Compare the latency of the LC task



End-to-end Latency of LC Tasks

Overall performance

- Co-locate LC DNN inference task with different BE tasks
- Compare the latency of the LC task

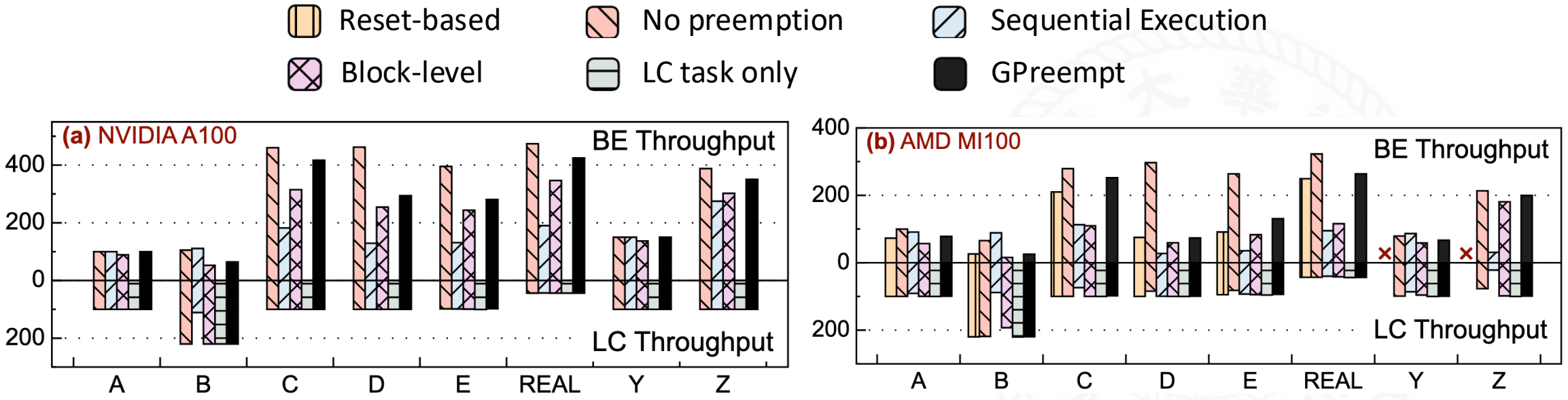


End-to-end Latency of LC Tasks

GPreempt increase only 2.4% latency on average for LC task when co-locating with BE tasks

Overall performance

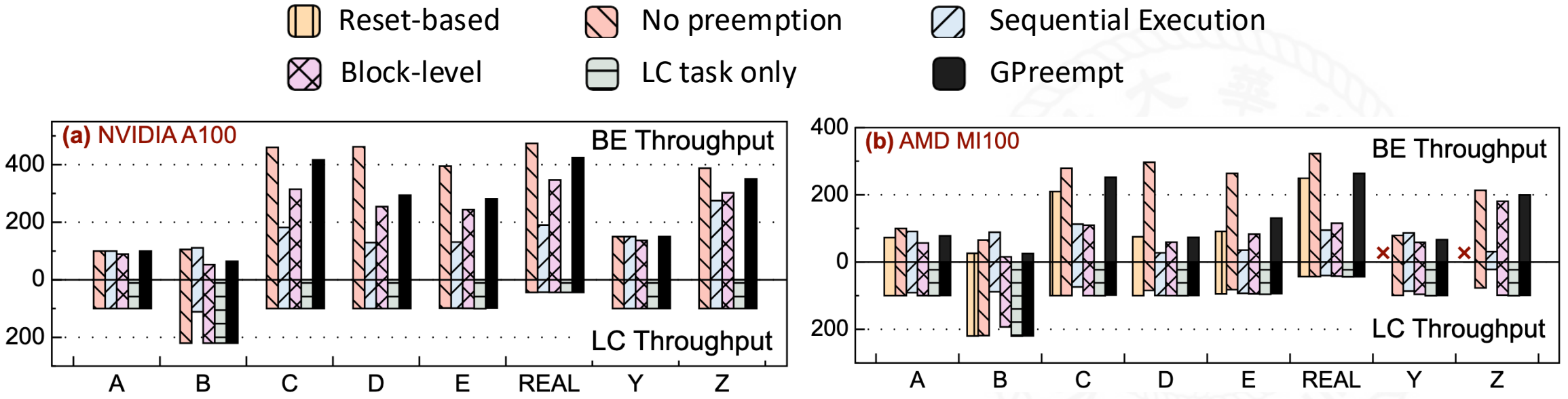
- Co-locate LC DNN inference task with different BE tasks
- Compare the throughput of LC and BE tasks



LC and BE task throughput (Above: BE tasks throughput, Blow: LC tasks throughput)

Overall performance

- Co-locate LC DNN inference task with different BE tasks
- Compare the throughput of LC and BE tasks



LC and BE task throughput (Above: BE tasks throughput, Blow: LC tasks throughput)

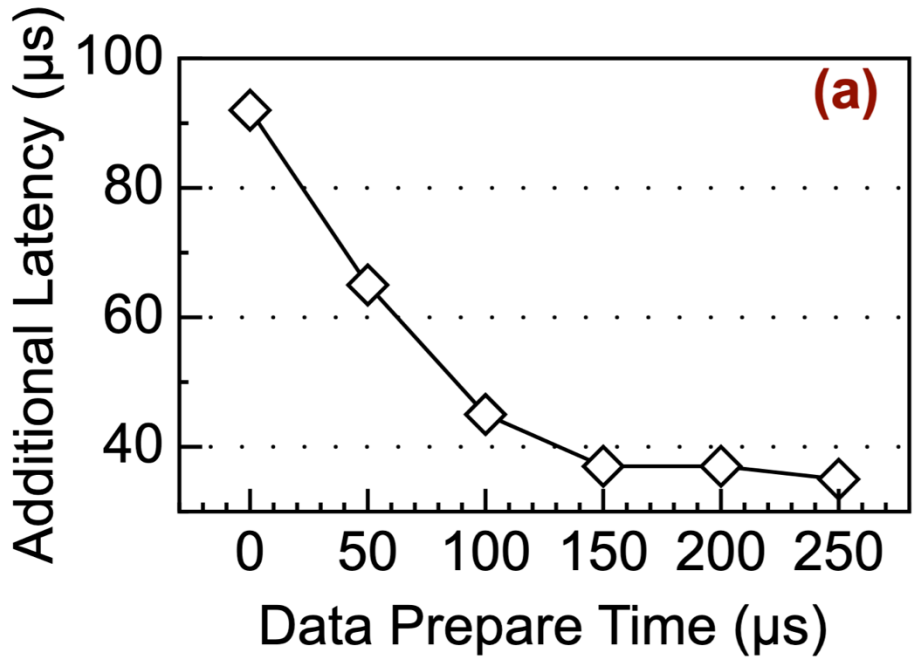
Gpreempt ensure high throughput of BE tasks (88.6% of the ideal throughput) while keeping the throughput of LC tasks

Technical Analysis

- Using workload A (LC: VGG + BE: ResNet)

Technical Analysis

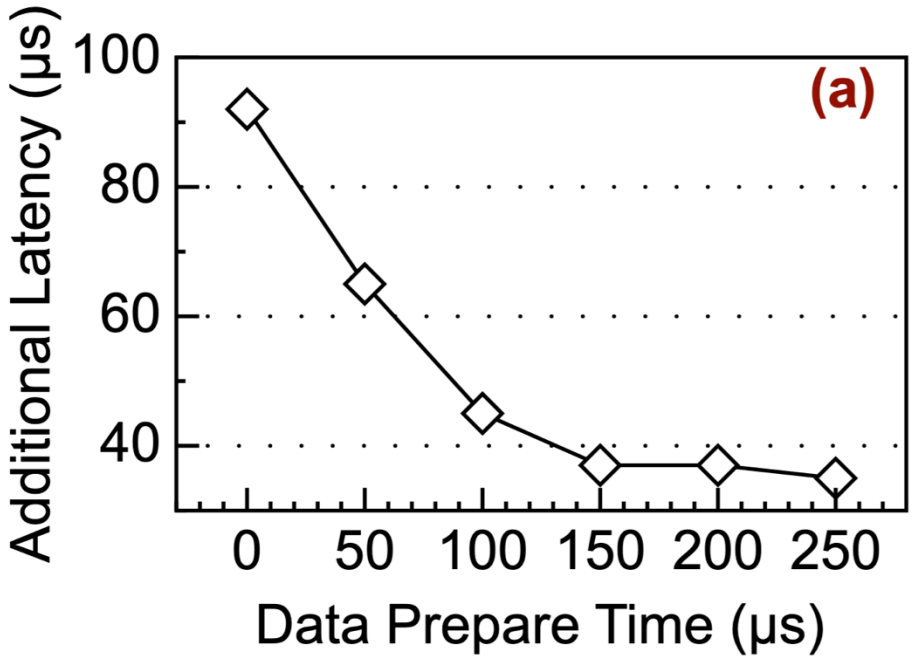
❖Using workload A (LC: VGG + BE: ResNet)



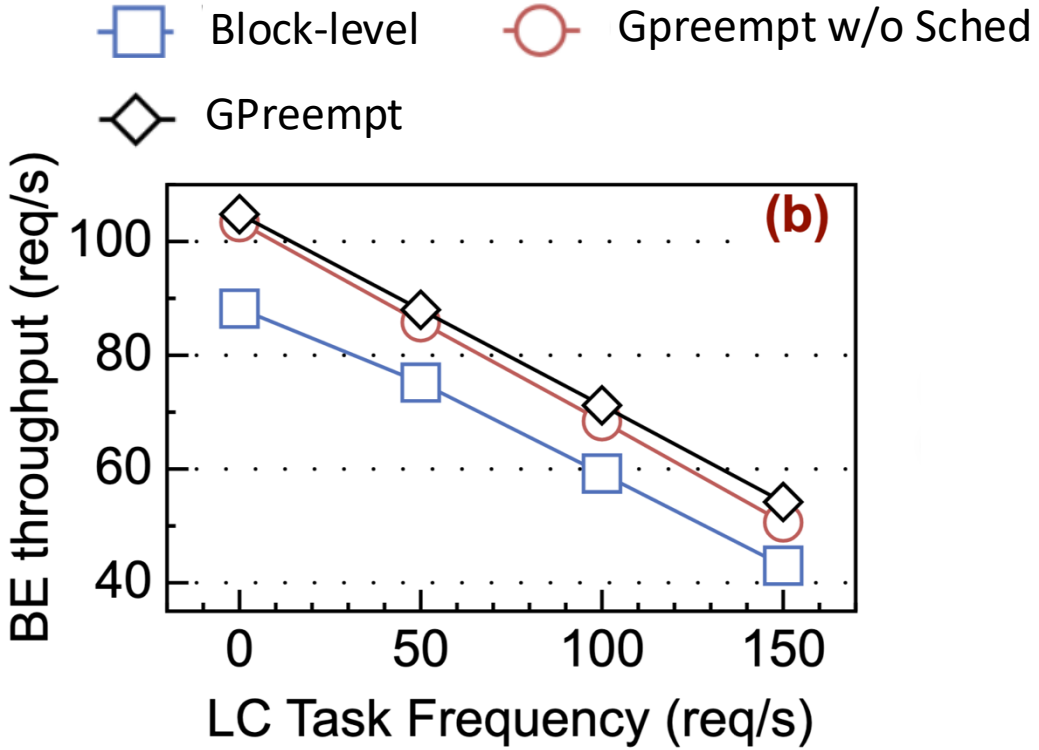
Additional latency on various data prepare time

Technical Analysis

❖Using workload A (LC: VGG + BE: ResNet)



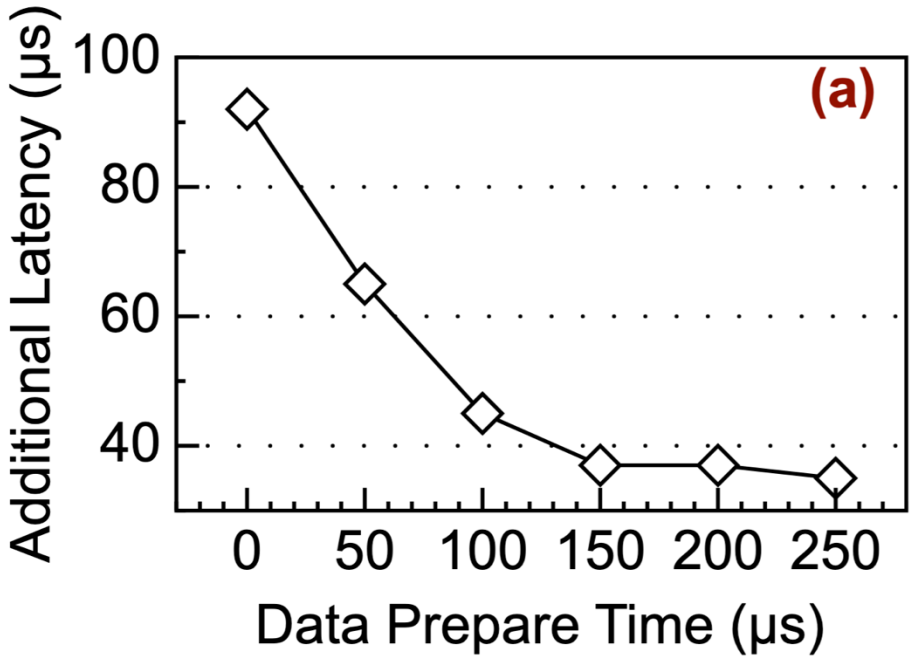
Additional latency on various data prepare time



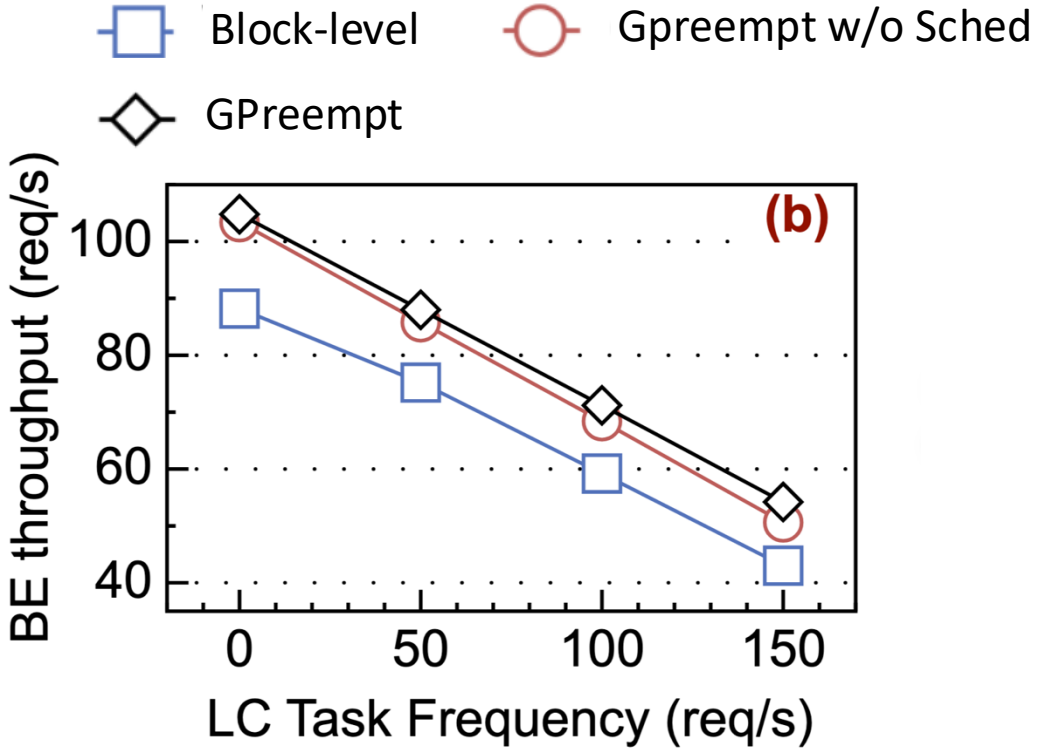
BE task throughput on different LC task frequency

Technical Analysis

❖Using workload A (LC: VGG + BE: ResNet)



Additional latency on various data prepare time



BE task throughput on different LC task frequency

Hint-based pre-preemption effectively overlap the latency of context switch and minimizes the impact on BE tasks.

Conclusion







Our Goal:

- Achieving general and fast GPU preemption

Our Idea:

- Leverage a switch-based approach to preempt the BE task

Our Design and Evaluation:

- Repurpose the timeslice mechanism to a passive preemption mechanism
- Overlap the data preparation with the preemption to hide the latency
- GPreempt can incur minimal overhead while keeping it general to various tasks

Takeaway:

- Leverage timeslice to achieve the same effect as active preemption
- Bringing more OS concepts to GPU make GPU more general and easier to use

GPreempt: GPU Preemptive Scheduling Made General and Efficient

Thanks & QA



Contact Information: frw23@mails.tsinghua.edu.cn