# Multitasking on GPU: Preemption

Yijia Diao

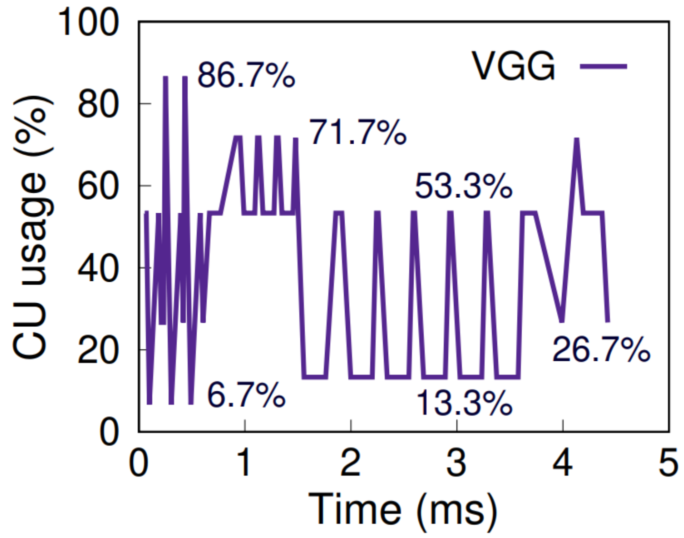
2023.10.19

## Outline

- Background
- Basic Preemption Techniques
- Advanced Preemption Techniques
- Complementary Techniques
- Summary

# Background: Scenario

- One GPU application could not fully utilize GPU resources.
- High priority task could not always immediately scheduled to GPU.



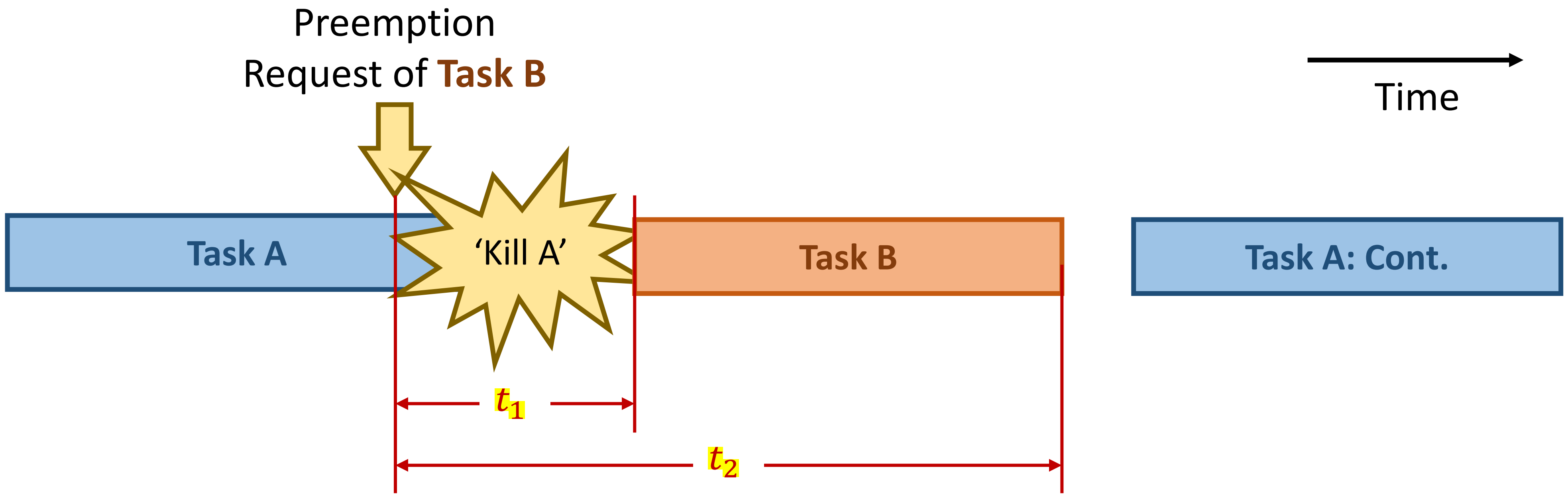
Obstacle Detection

Fatigue Detection

**Higher Priority Task**: Latency Critical

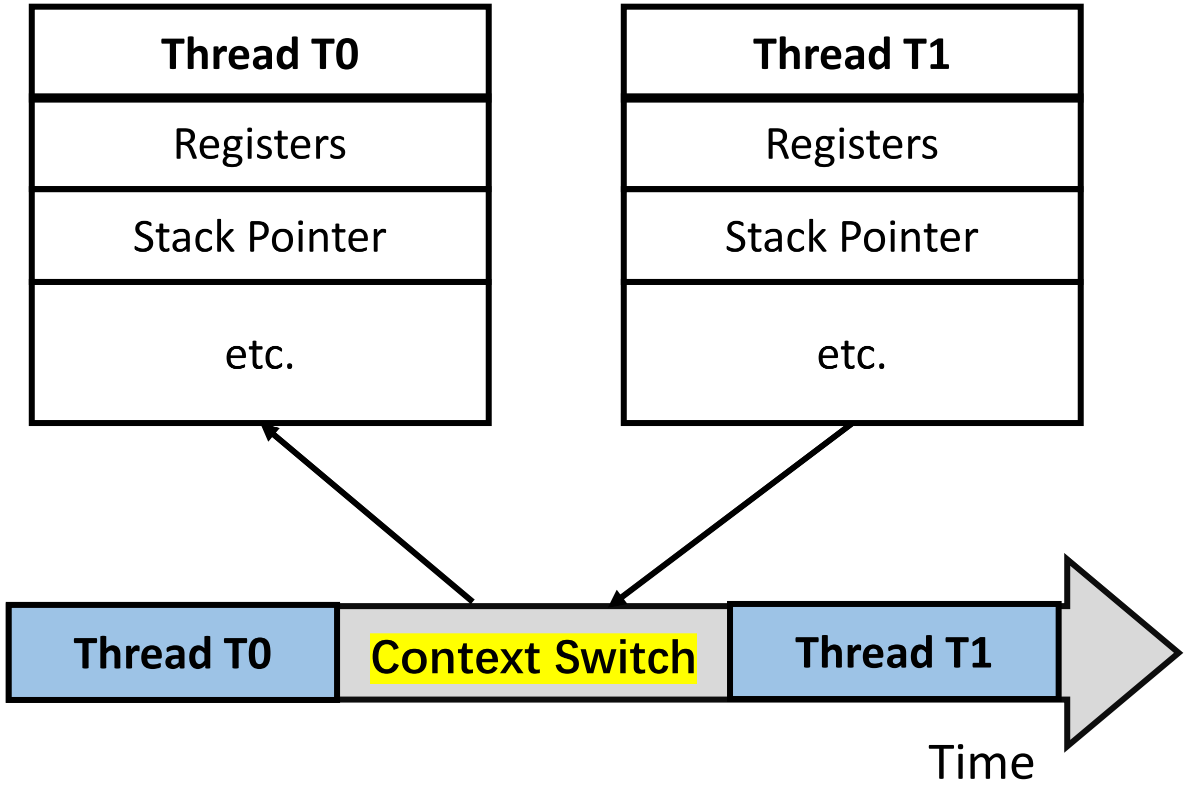
**Lower Priority Task:** No hard real-time requirement

# Background: Preemption Concept



- $t_1$: Preemption Time
- $t_2$: Turnaround Time
- System Throughput
- Hardware Utilization

## Background: CPU preemption



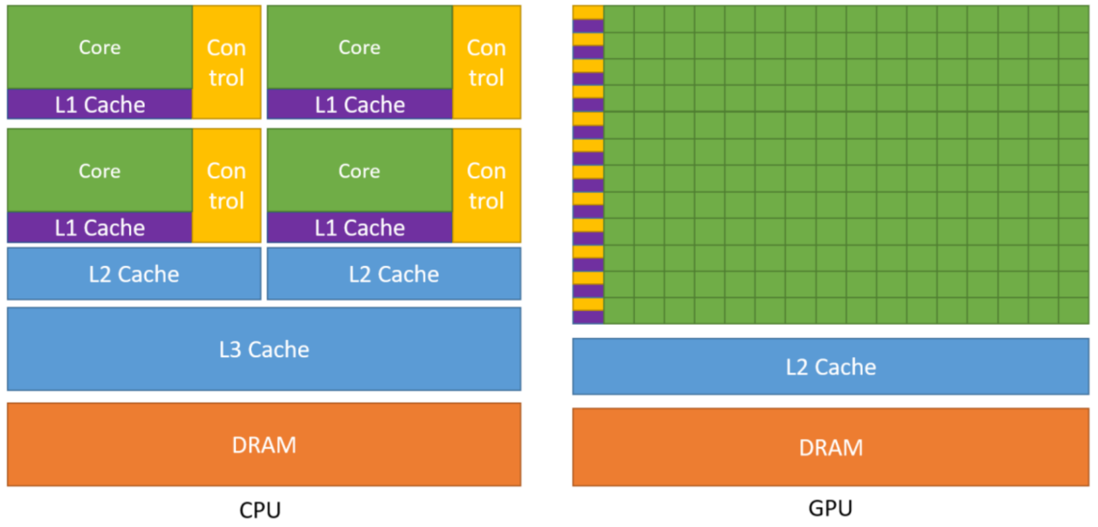
https://zhuanlan.zhihu.com/p/657205626

## Basic Preemption Techniques

- Context Switch
- Draining

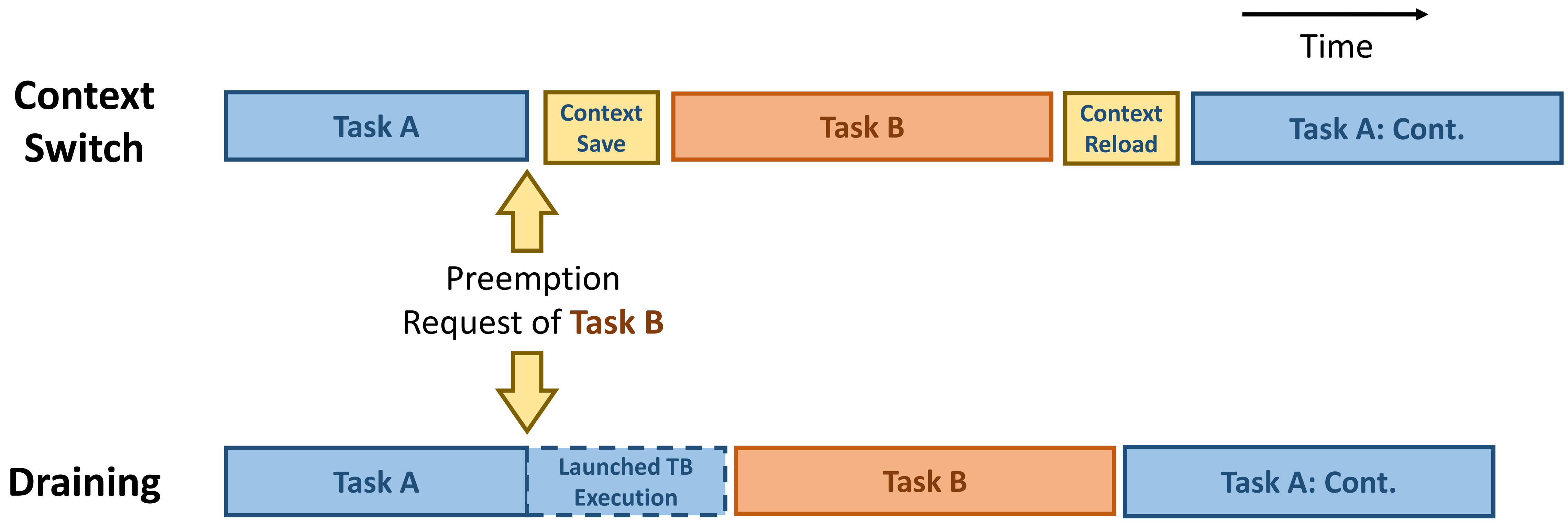
# Context Switch on GPU

- GPU has a large amount of on-chip memory.
  - CPU : 40 regs / core = 320 B / core
  - GPU: 64K regs / SM + 192KB smem / SM = 448KB / SM
  - #SM=108; max memory bandwidth = 1.5TB/s
- Minimum Context Saving Latency = 30us

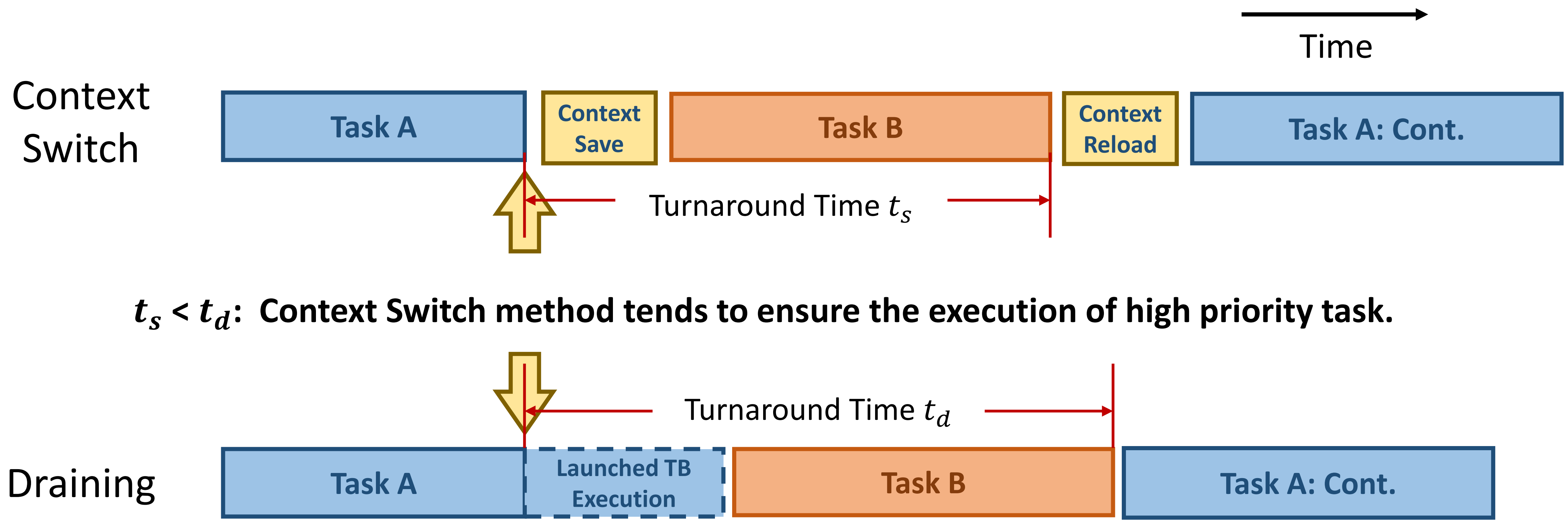


https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

# Context Switch vs. Draining

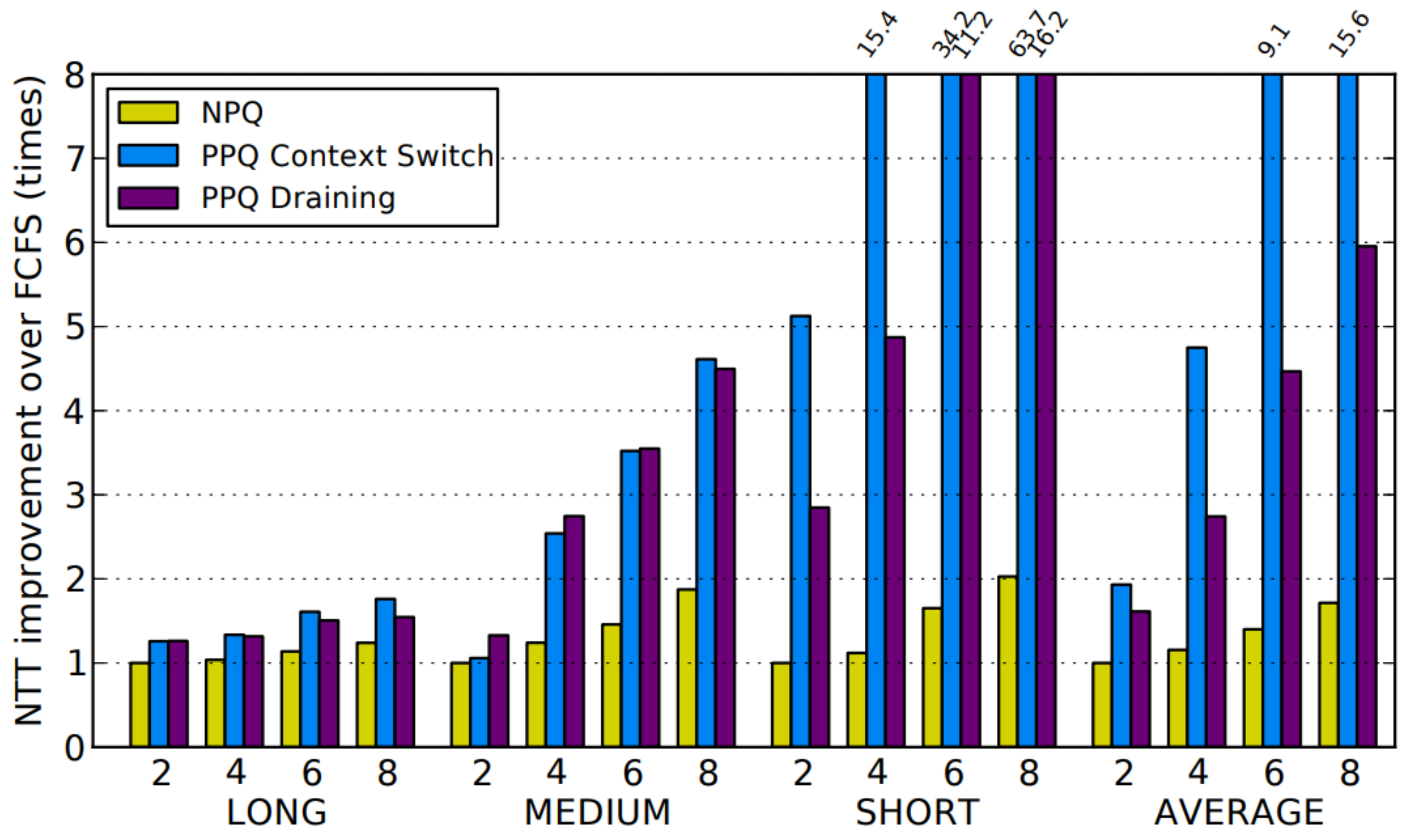


# Context Switch vs Draining: Turnaround Time

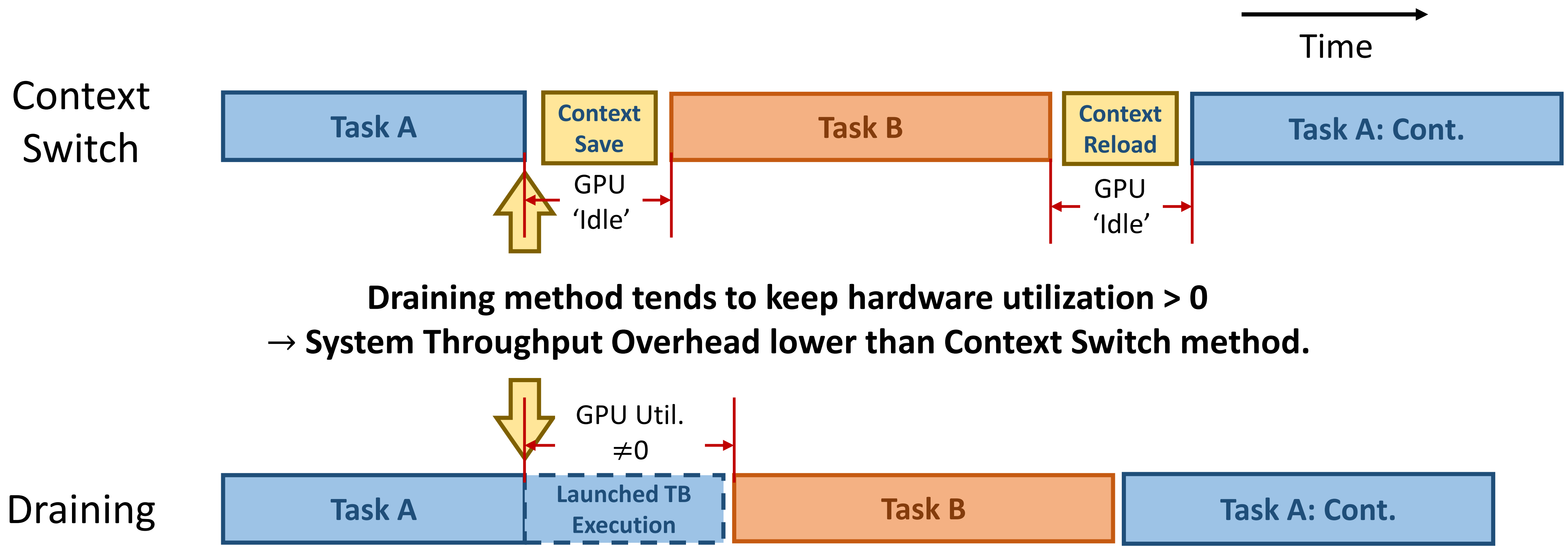


**$t_s < t_d$: Context Switch method tends to ensure the execution of high priority task.**

# Context Switch vs Draining: Turnaround Time



# Context Switch vs Draining: Throughput



**Draining method tends to keep hardware utilization > 0**
**→ System Throughput Overhead lower than Context Switch method.**

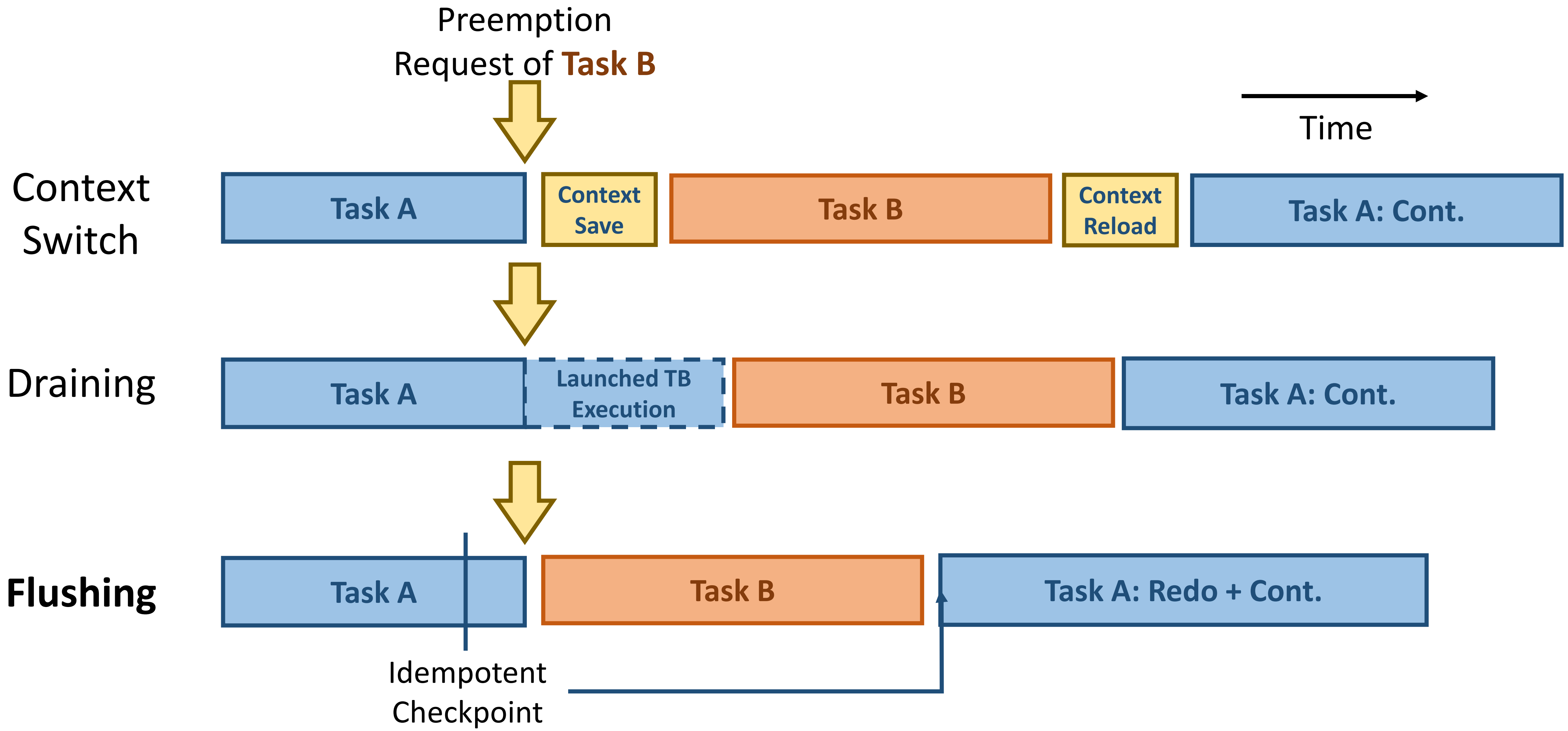
## Advanced Preemption Techniques

- Flushing: based on Idempotency
- Prediction: based on Kernel Launch Interval
- Fine-grained Preemption:  SM-level

## Idempotency: Definition

- Initial Definition:
  - A GPU kernel is idempotent if it produces the same result regardless of the number of times it is executed.

# Flushing: based on Idempotency



## Idempotency: Relaxed Definition

- Relaxed Definition:
  - A GPU thread block is idempotent at a given time if it neither
  1. has executed any atomic operations yet, nor
  2. has overwritten a global memory location that is read by the thread block.
- Atomic operations or global memory overwrites tend to be performed **at the end of a thread block execution!**

# Idempotency: Relaxed Definition (Cont.)

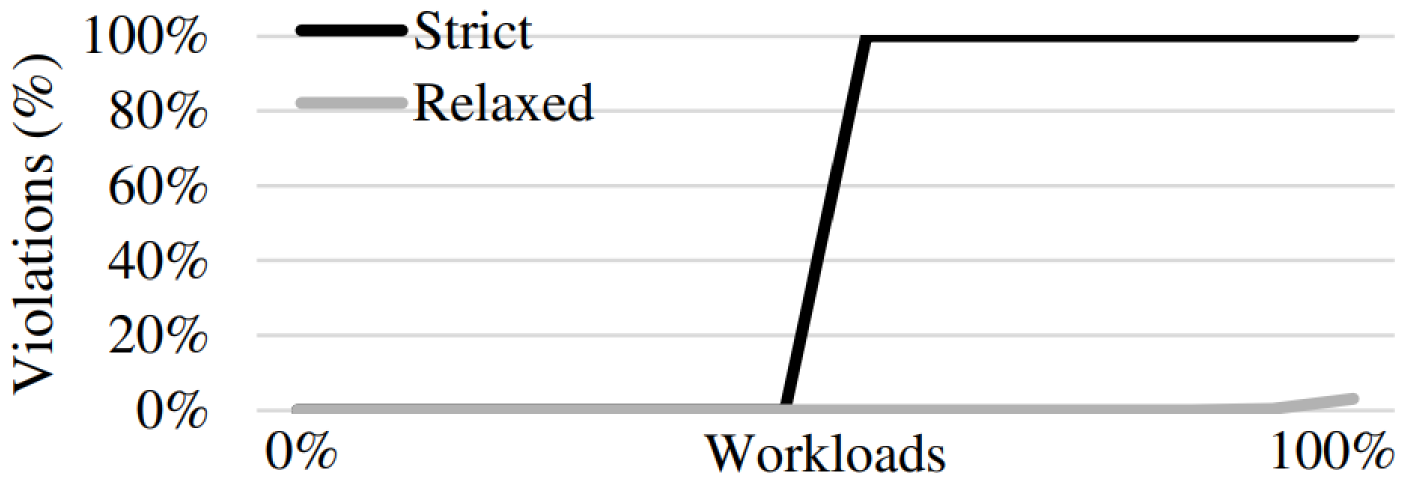
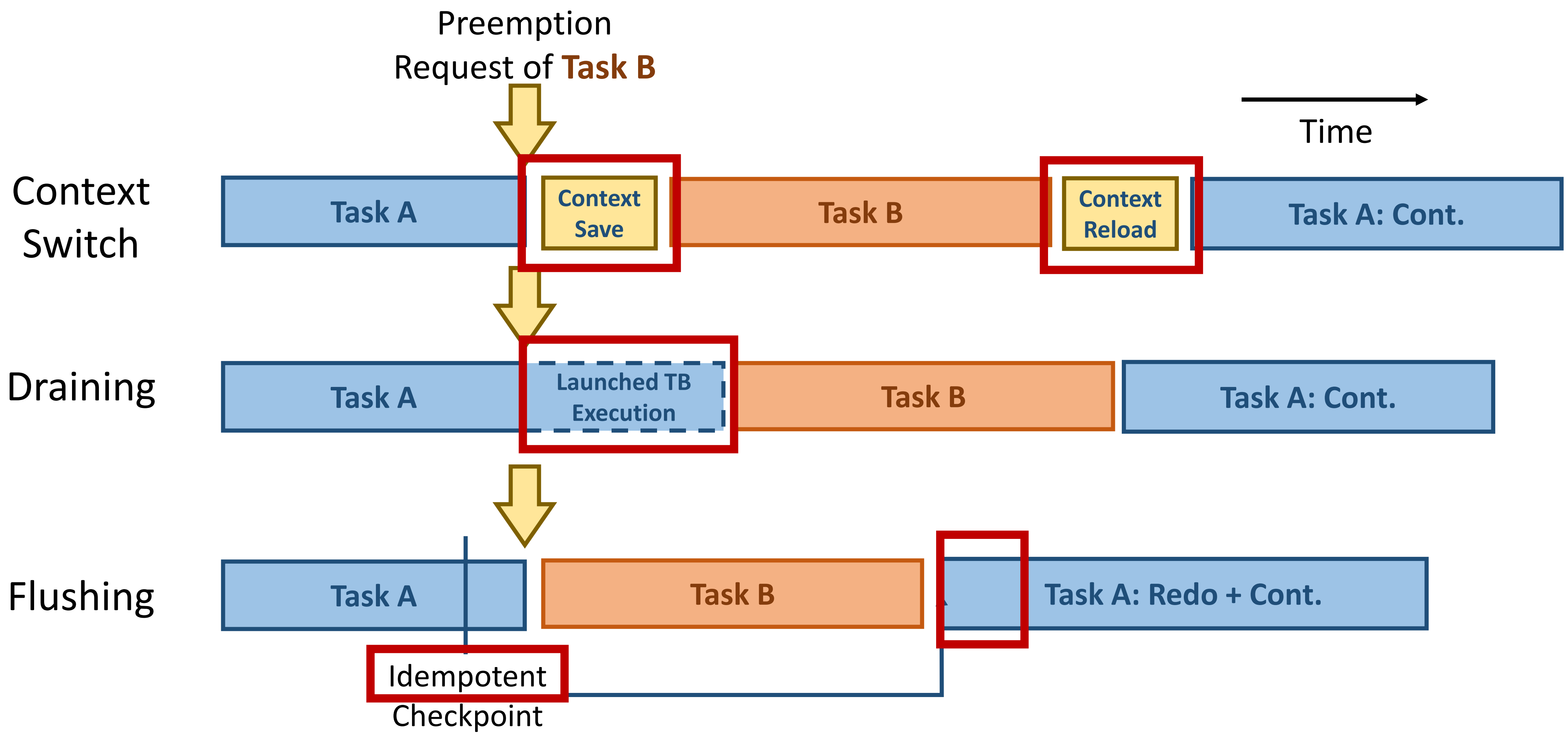


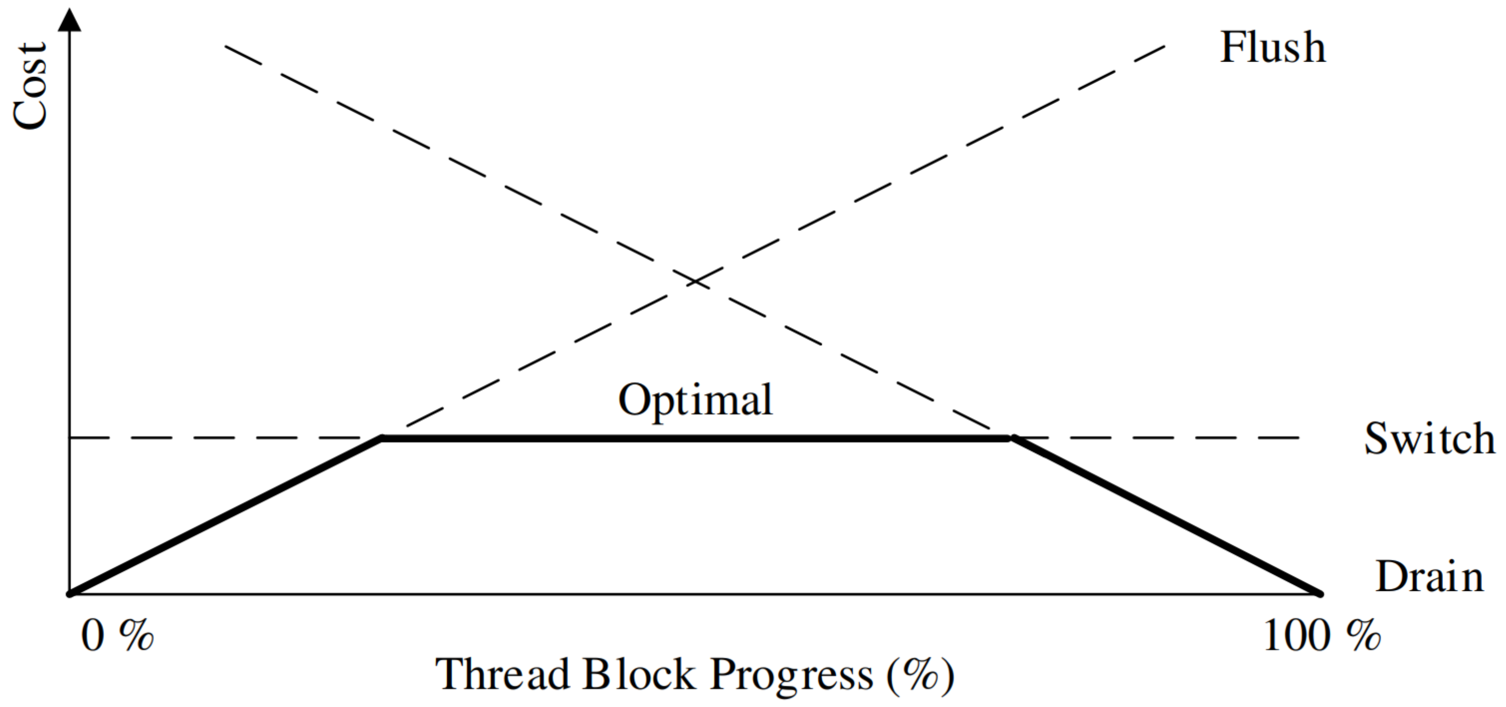
**Figure 9.** The percentage of preemptions that violate 15$\mu$s preemption latency constraint when SM flushing uses strict or relaxed idempotence condition.

# Combine three methods?

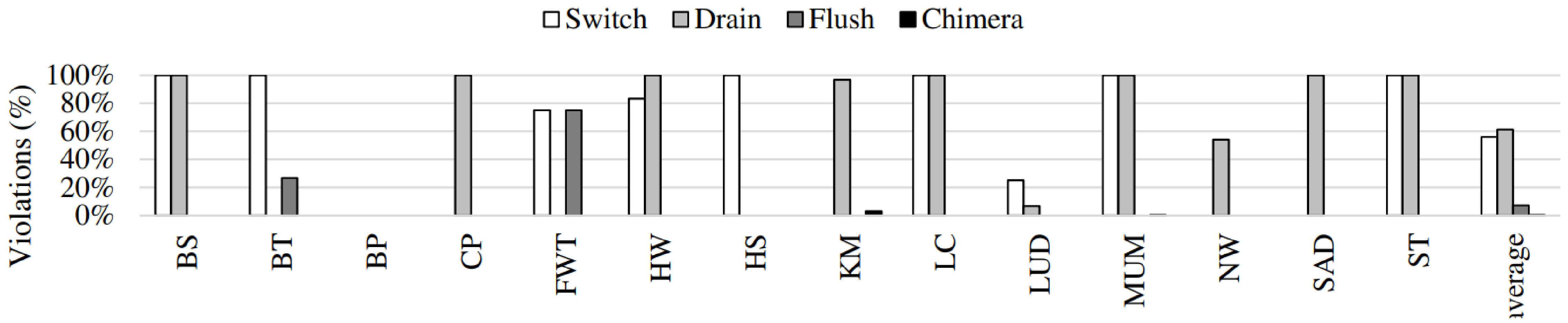


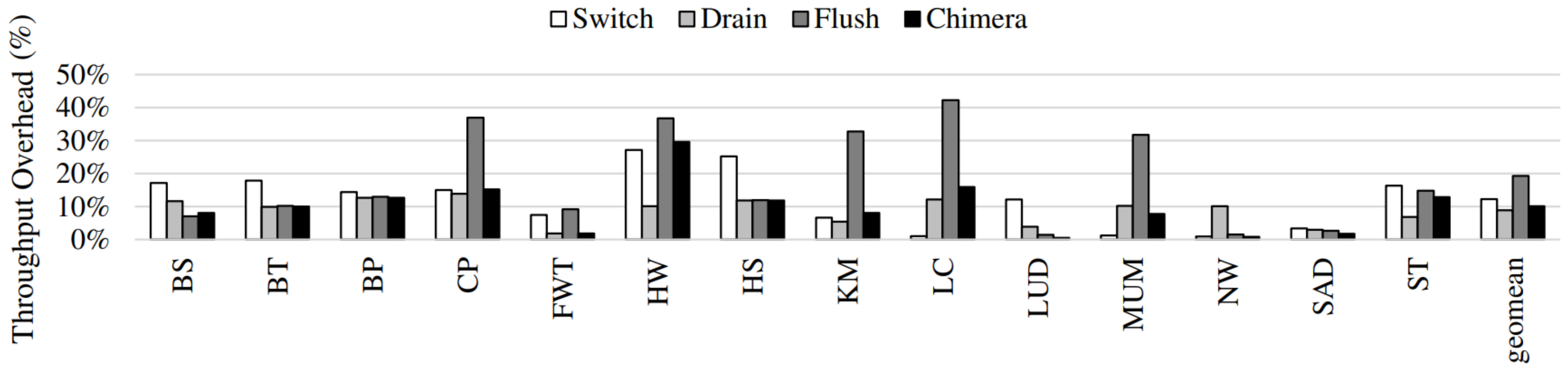
# Combine three methods (Cont.)

- Cost: Consider both preemption latency & system throughput



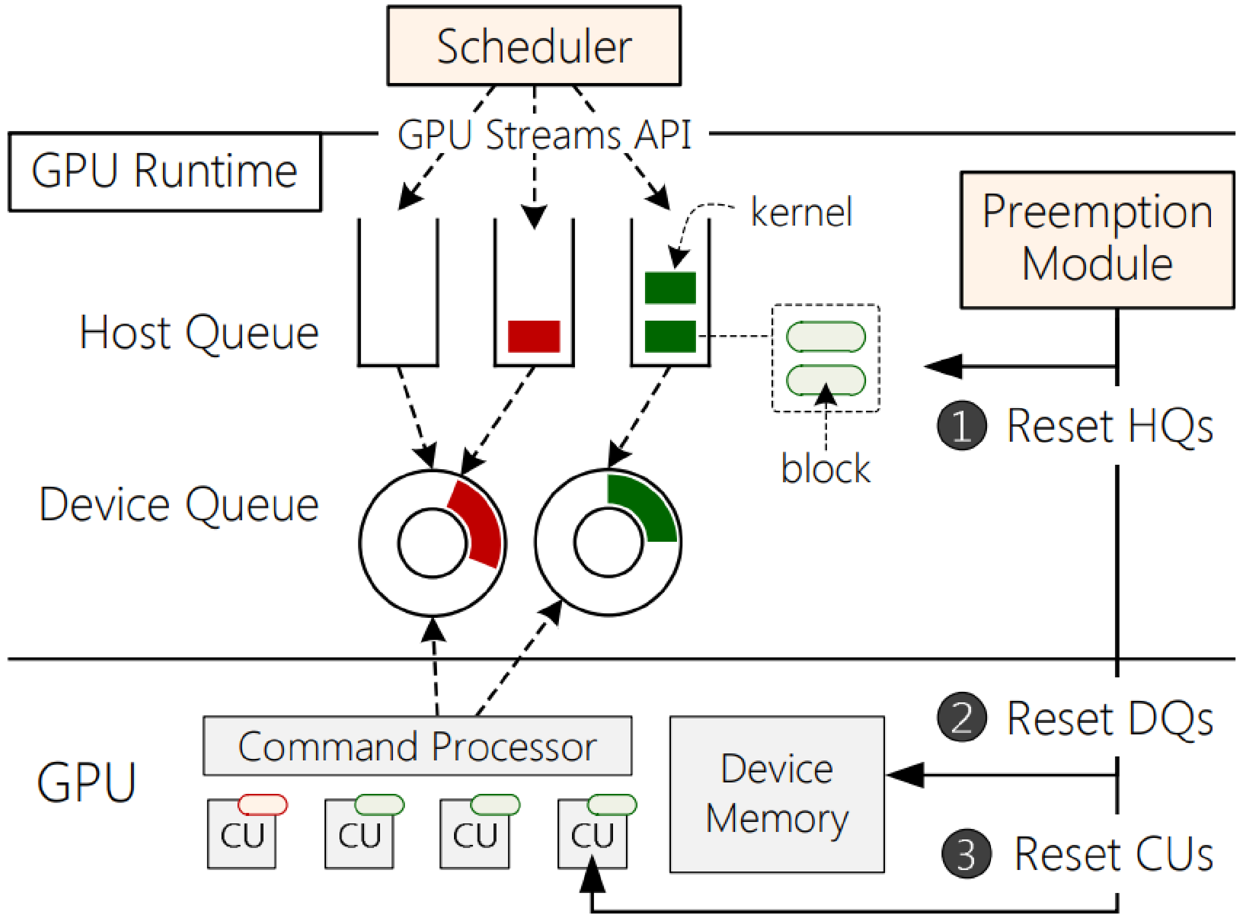
# Combine three methods: Experiment





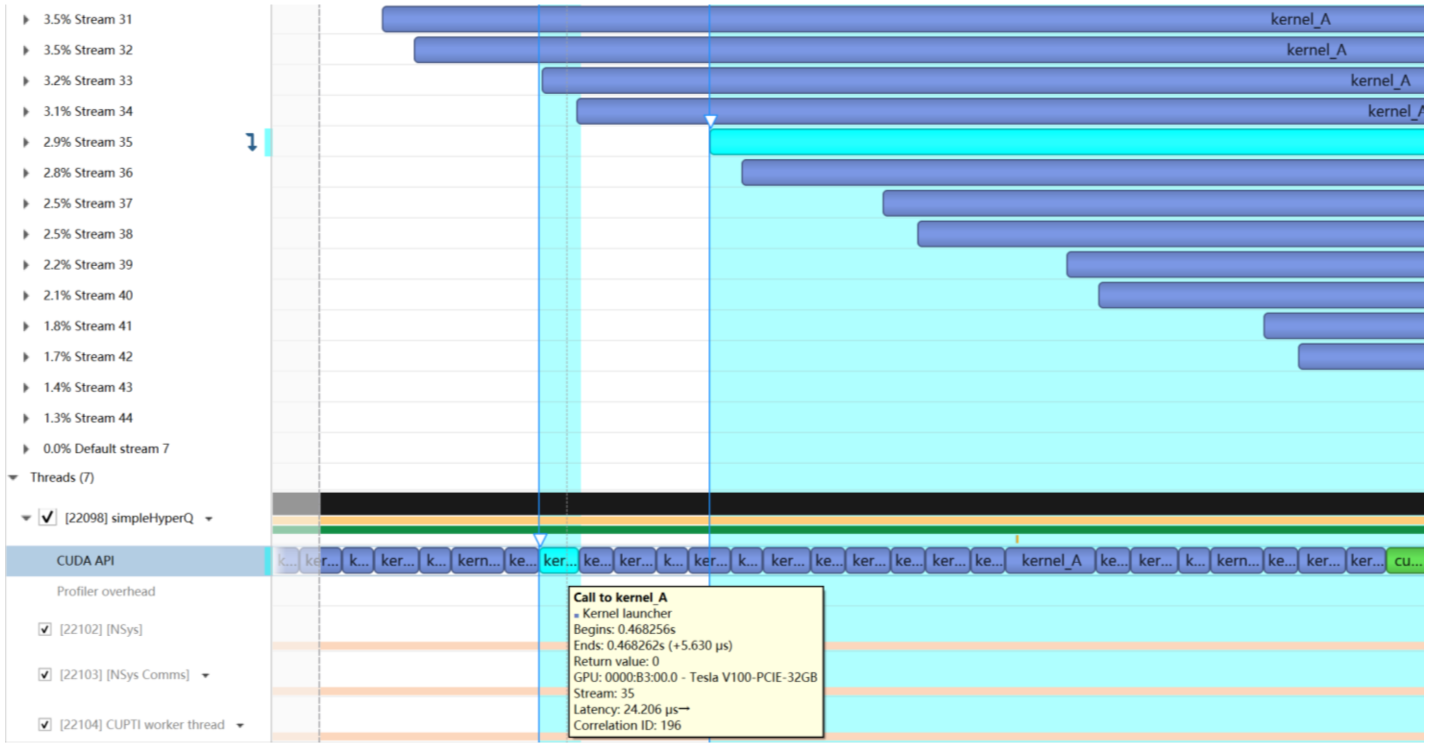
# Idempotency in DNN: REEF

- DNN inference is mostly idempotent.
- So we can only use Flushing method! : )
- They use 'Reset' instead of 'Flushing' in paper.



## Prediction: based on Kernel Launch Interval

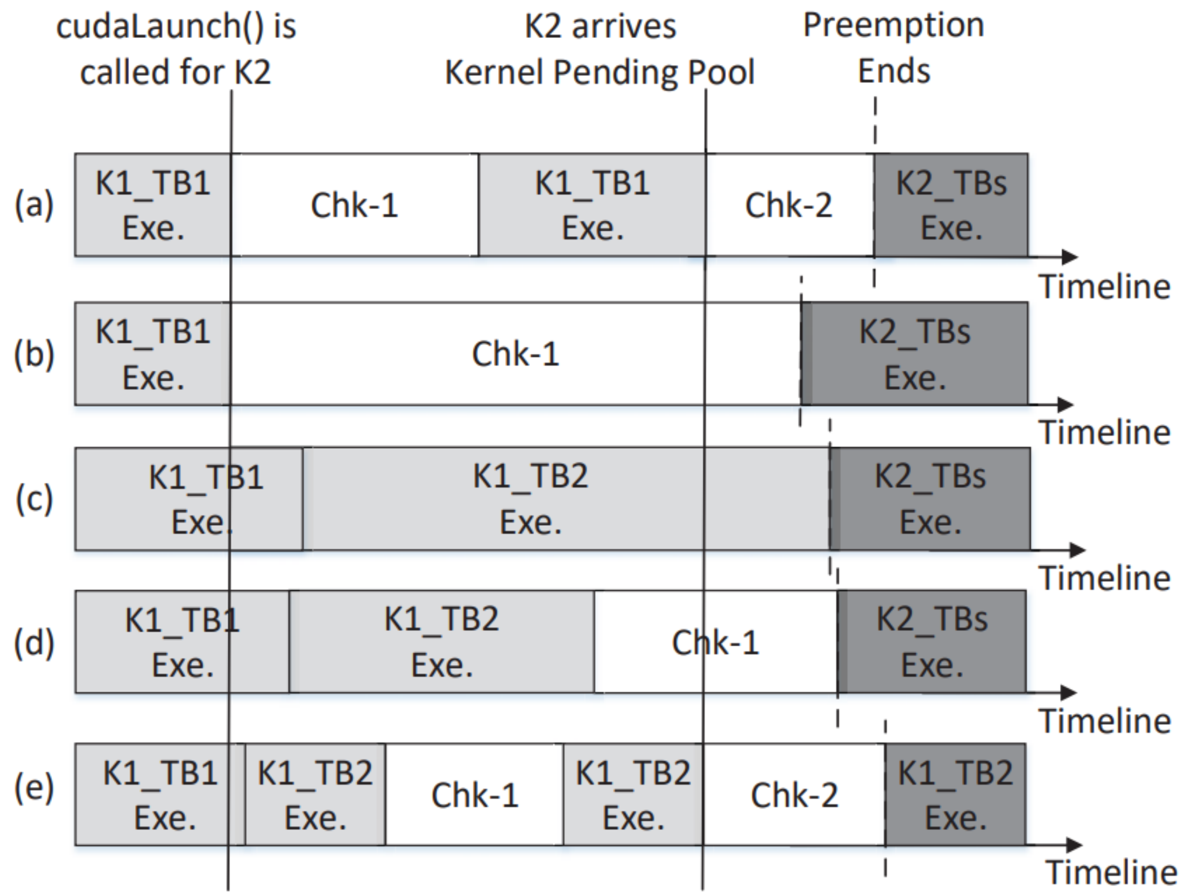
- Call to kernel launch ~ Kernel successful launch has time interval.
- We could do checkpointing at the time of preemption kernel call.



Call to kernel_A
▪ Kernel launcher
Begins: 0.468256s
Ends: 0.468262s (+5.630 μs)
Return value: 0
GPU: 0000:B3:00.0 - Tesla V100-PCIE-32GB
Stream: 35
Latency: 24.206 μs→
Correlation ID: 196

C. Li et al. PEP: Proactive Checkpointing for Efficient Preemption on GPUs. DAC 18'

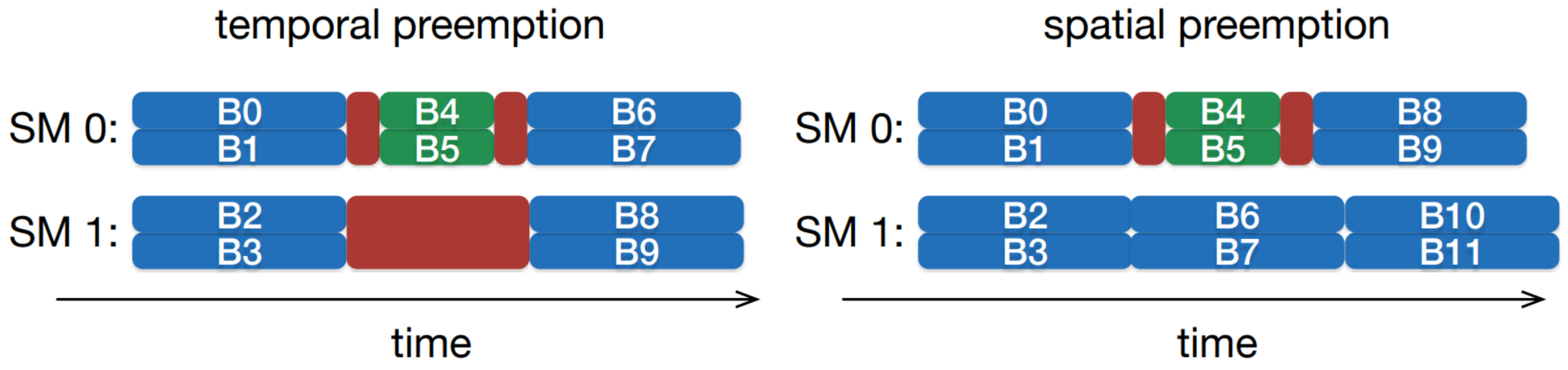
# Prediction: based on Kernel Launch Interval



C. Li et al. PEP: Proactive Checkpointing for Efficient Preemption on GPUs. DAC 18'

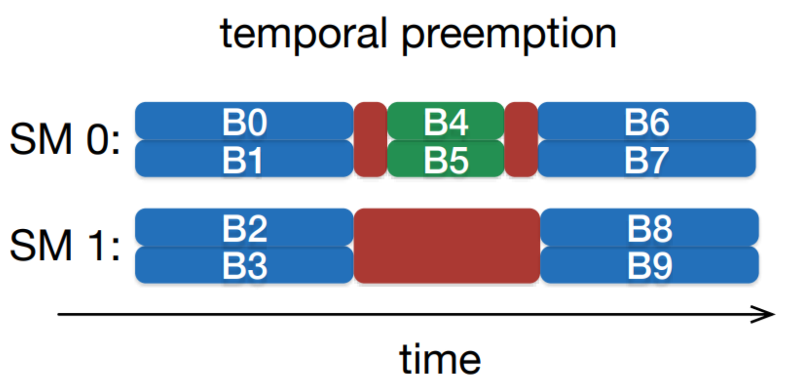
# Fine-grained Preemption:  SM-level

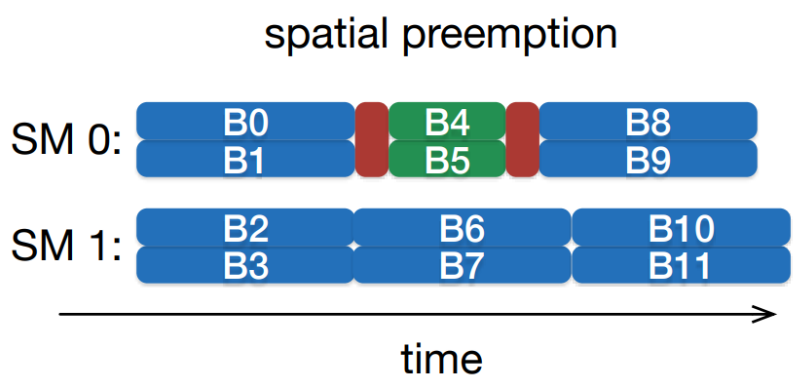
- Sometimes, high priority task could not fully utilize GPU.
- Preempt necessary SMs is enough.



B. Wu et al. FLEP: Enabling Flexible and Efficient Preemption on GPUs. ASPLOS 17’

# Fine-grained Preemption:  FLEP





```
kernel_temporal_preemption_enhanced(
              …  //original parameters
              volatile unsigned int *temp_P,
              unsigned int L)
{
  while(1) {
    if(*temp_P==true) return;

    for(int i=0; i<L; ++i) {
      if((task = pull_task()) == NULL)
        return;

      process(task); }
  }
}
```

```
kernel_spatial_preemption(
              …  //original parameters
              volatile unsigned int *spa_P,
              unsigned int L)
{
  while(1) {
    if(hostSM_ID < *spa_P) return;

    for(int i=0; i<L; ++i) {
      if((task = pull_task()) == NULL)
        return;

      process(task); }
  }
}
```

B. Wu et al. FLEP: Enabling Flexible and Efficient Preemption on GPUs. ASPLOS 17'

## Complementary Techniques

- Duration Estimation
- Dynamic Kernel Padding

# Why Complementary Techniques?

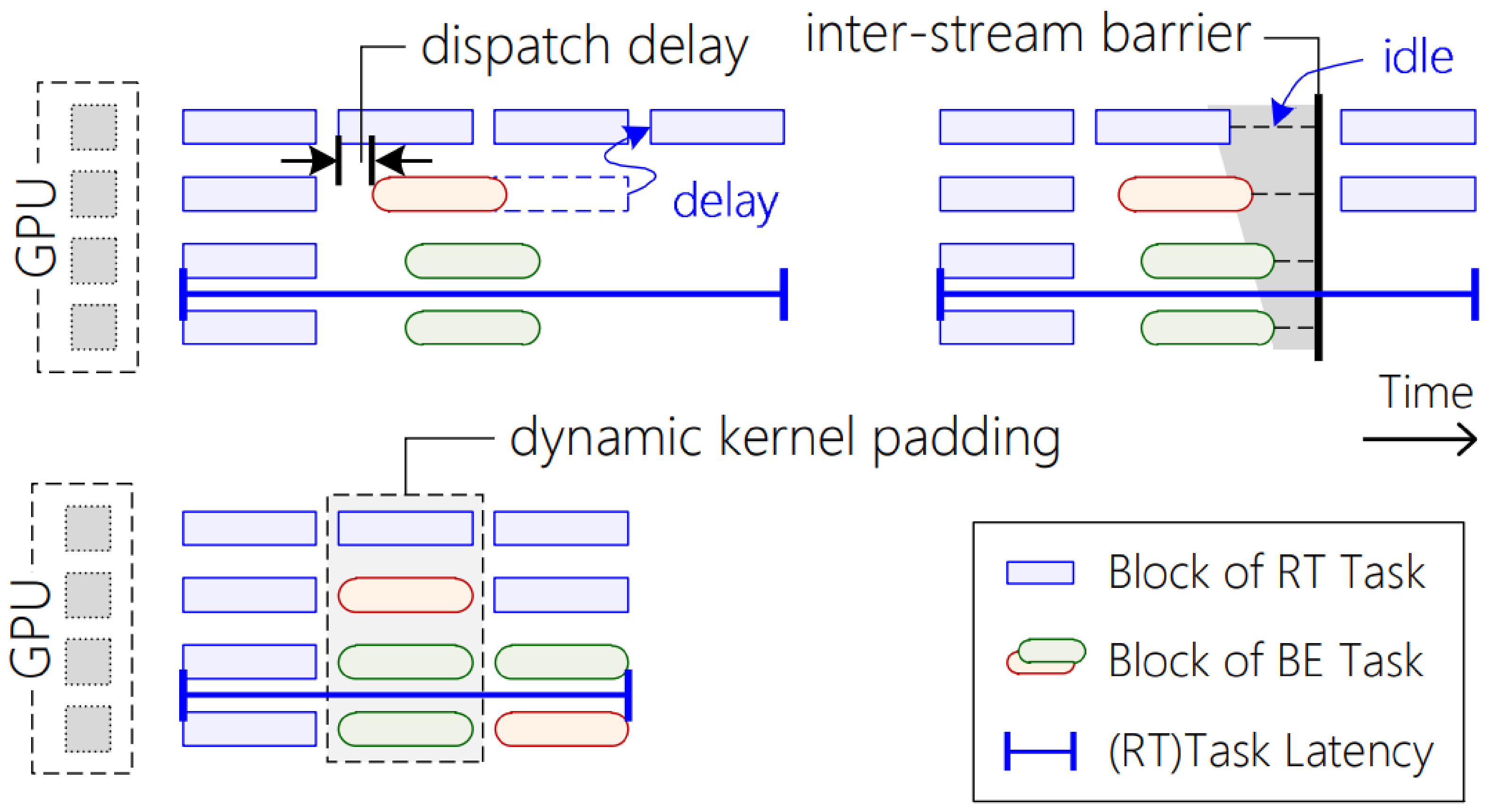
- Preemption Time:
  - We could not definitely say, Context switch method preemption time must be shorter than that of Draining.
  - Kernel & Context switch Duration estimation is needed.
- System Throughput:
  - Only guarantee high priority task’s execution is not enough. Still waste hardware resources.
  - Try to use the remaining resources.

# Duration Estimation: Chimera's Policy

- Preemption method selection depends on the estimation below.

| | **Throughput Overhead** | **Preemption Latency** |
|---|---|---|
| **Context Switch** | IPC(preempted kernel) * 2 | Assume fixed |
| **Draining** | $\sum \max(\#inst) - \#inst$ | #(remain insts.) * IPC(preempted kernel) |
| **Flushing** | $\sum \#inst$ | Assume fixed(lower than switch) |

# Dynamic Kernel Padding: REEF



## Summary

- Three important Preemption methods:
  - Context Switch
  - Draining
  - Flushing(idempotency)
- Complementary methods should be supplied to reduce the preemption latency and ensure system throughput.

Thanks!