Hardware Compute Partitioning on NVIDIA GPUs

Joshua Bakita and James H. Anderson

Department of Computer Science
University of North Carolina, Chapel Hill

How can we do more, with less?

How can we do more, with less, on the CPU?

Processor Cores

➔ Concurrency, with hardware partitioning

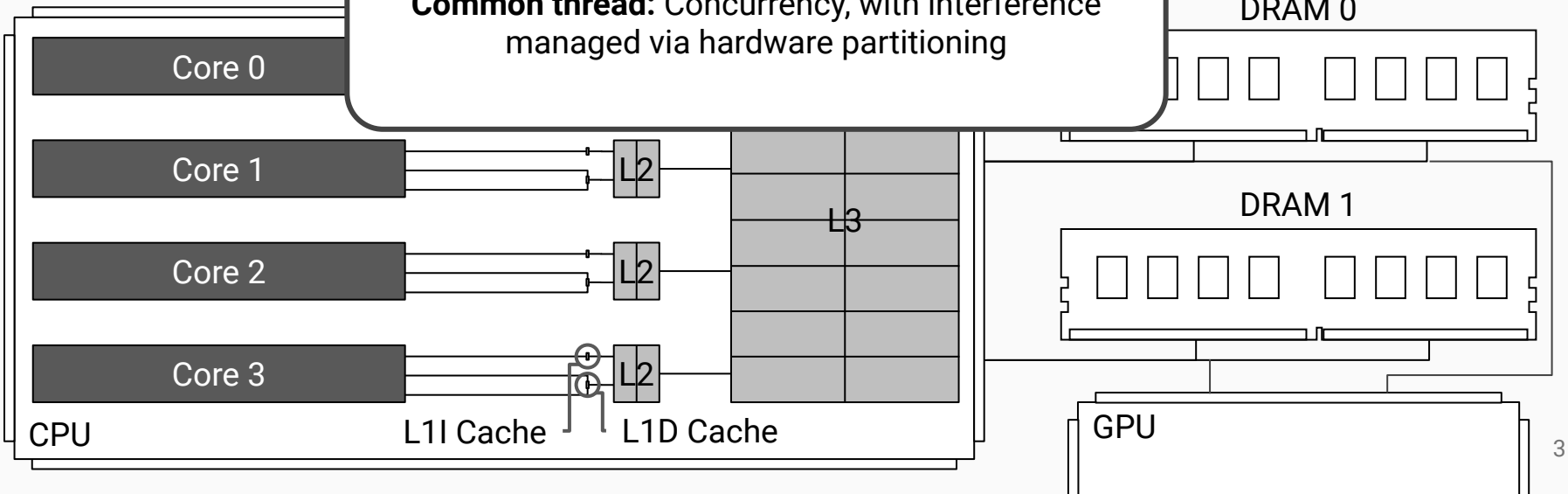
Memory Caches and Interconnects

➔ Concurrency, with hardware partitioning

Main Memory (DRAM)

➔ Concurrency, with hardware partitioning

Common thread: Concurrency, with interference managed via hardware partitioning



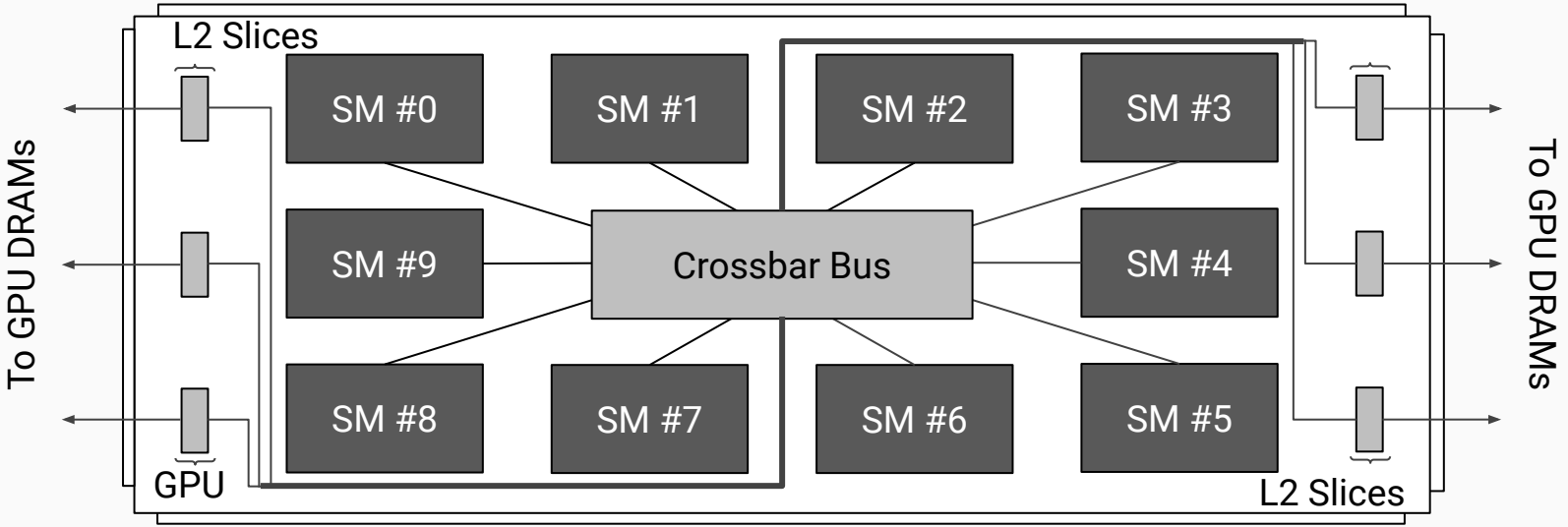
How can we do more, with less, on the GPU?

Compute Units

➔ **No hardware partitioning available**

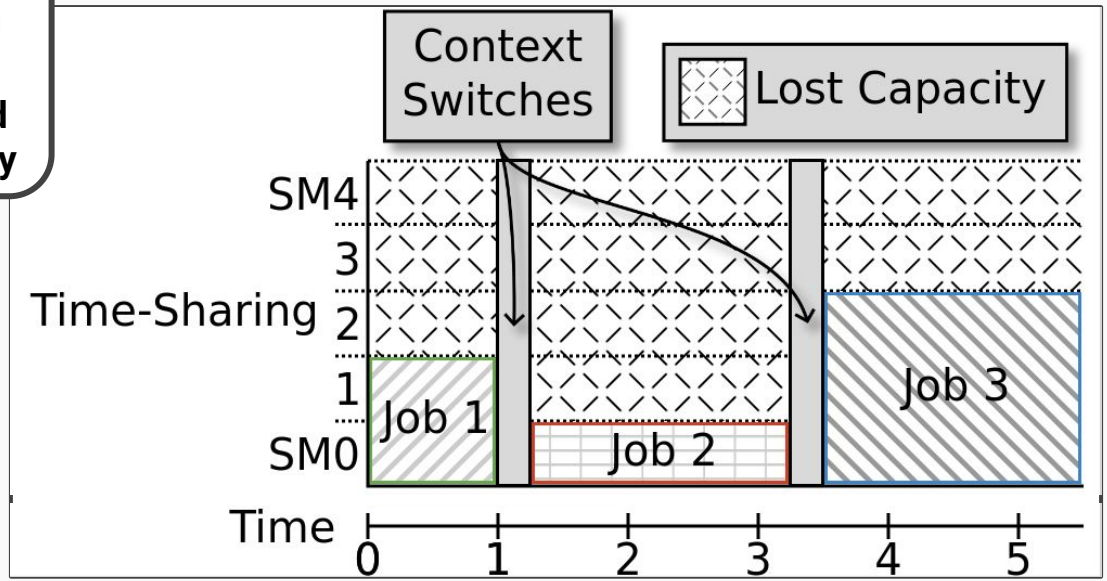
Memory Caches and Interconnects

➔ Concurrency, with hardware partitioning [1]

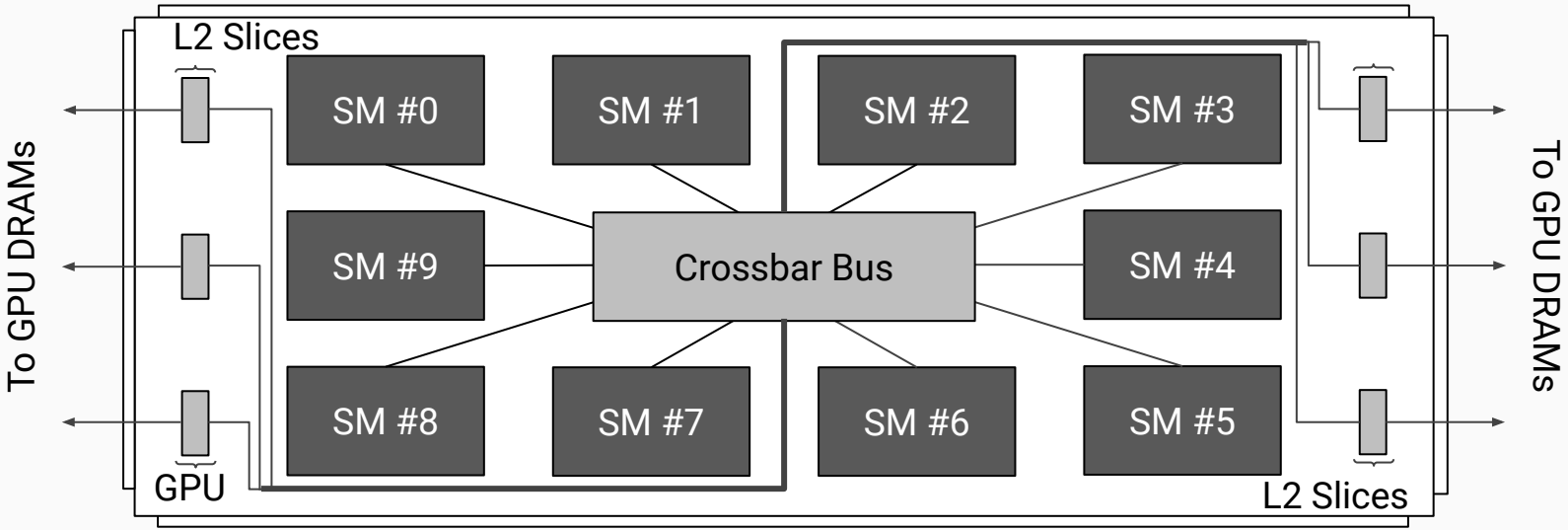


Assumption: No Capacity To Reclaim

Reality: Trends in GPU architecture have increased wasted capacity

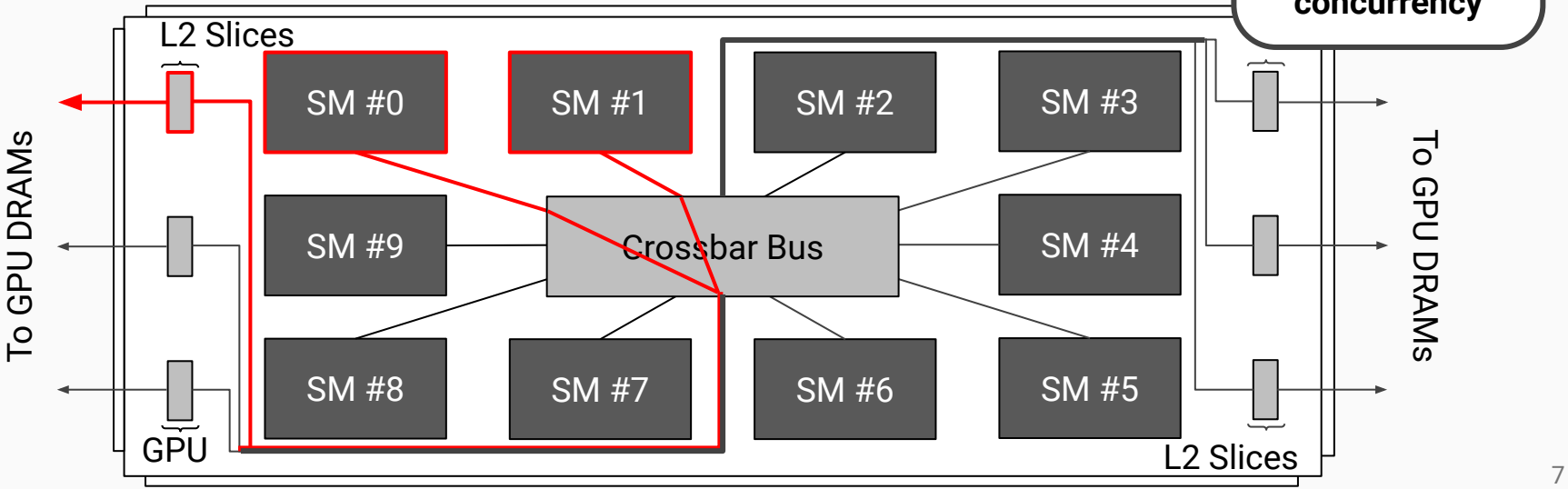


Assumption: Interference Worse than On-CPU



Assumption: Interference Worse than On-CPU

Reality: GPUs are highly capable of interference-free concurrency



Key Goals

Prior work:
NVIDIA MIG [9]

Prior work:
Fractional GPUs [1]

Spatial partitioning for GPU compute units that is:

Hardware-Enforced

Flexible

Easily Applicable

With key insights drawn from **GPU architectural norms** and **native GPU scheduling systems**, we achieve all three for *any* NVIDIA GPU from the past 10 years.

Closest prior work:
AMD Compute Unit Masking [14, 15]

Enabling Hardware-Enforced Compute Partitioning

Goal 1 of 3

Why Hardware Enforcement?

Tasks may misbehave due to:

- Engineering Oversight
- Malfunction
- Malice

and these are fatal to cooperation-based software partitioning.

Hardware-Enforced Part.

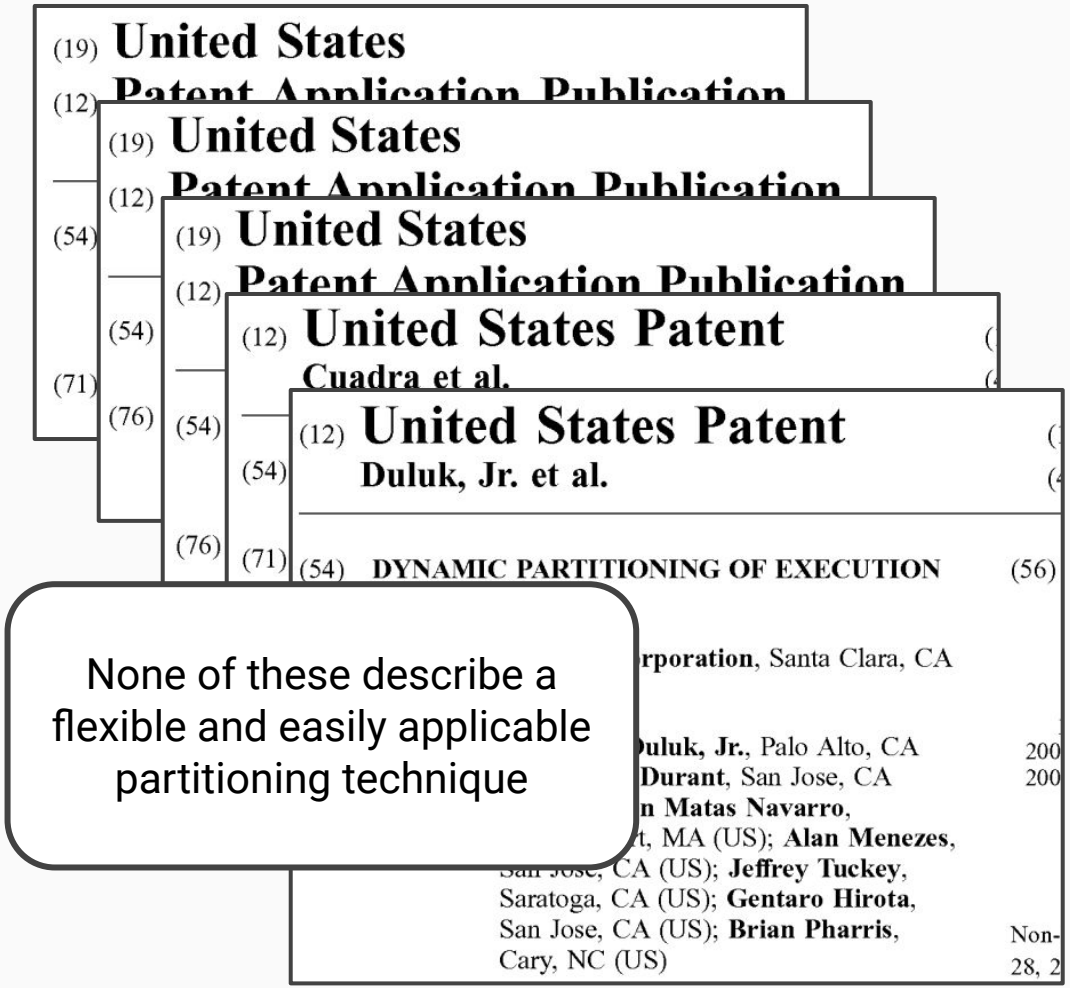
Elucidating GPU Capability

Untapped documentation:

- ➔ Patent Applications
- ➔ Granted Patents

Patents may describe non-existent inventions. We verify by cross-referencing:

- ➔ Open-Source Headers
- ➔ Open-Source Drivers
- ➔ NVIDIA Documentation
- ➔ Experiments
- ➔ ...



None of these describe a flexible and easily applicable partitioning technique

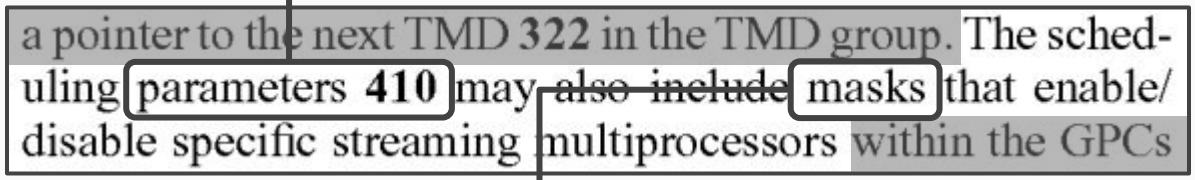
Hardware-Enforced Partitioning

Can these fields control kernel-to-SM assignment?

One Sentence of Documentation

Part of a Task MetaData (TMD) structure

A TMD describes a single program (a kernel) to be run on the GPU until completion.


a pointer to the next TMD **322** in the TMD group. The scheduling parameters **410** may ~~also include~~ masks that enable/disable specific streaming multiprocessors within the GPCs

From US 2013/0152094A1

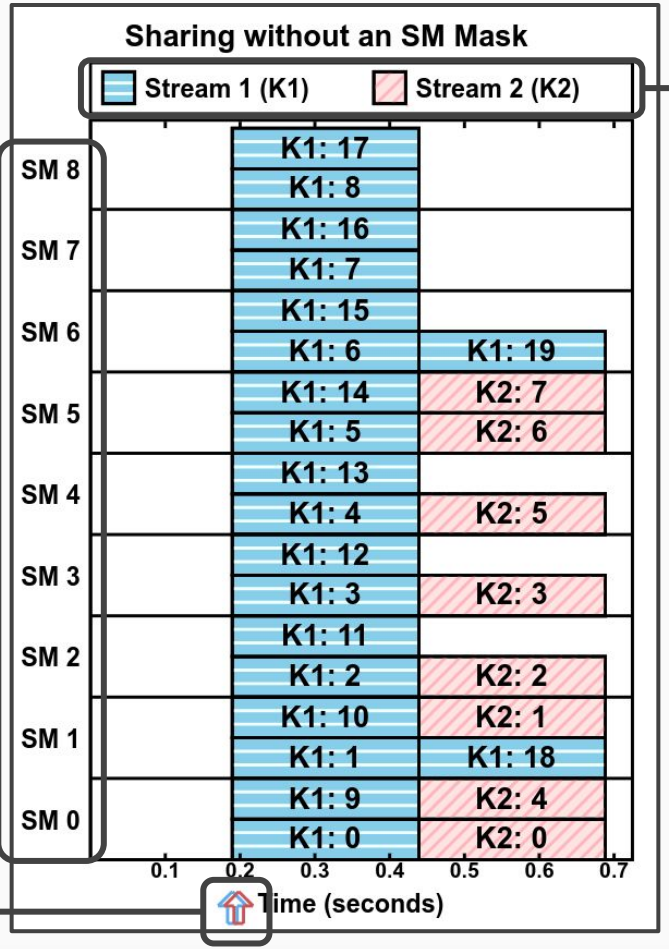
```
#define NVB0C0_QMDV01_07_SM_DISABLE_MASK_LOWER                MW(703:672)
#define NVB0C0_QMDV01_07_SM_DISABLE_MASK_UPPER                MW(735:704)
```

From `clb0c0qmd.h`

A kernel is broken into *blocks*, and then the blocks are scheduled concurrently on the GPU

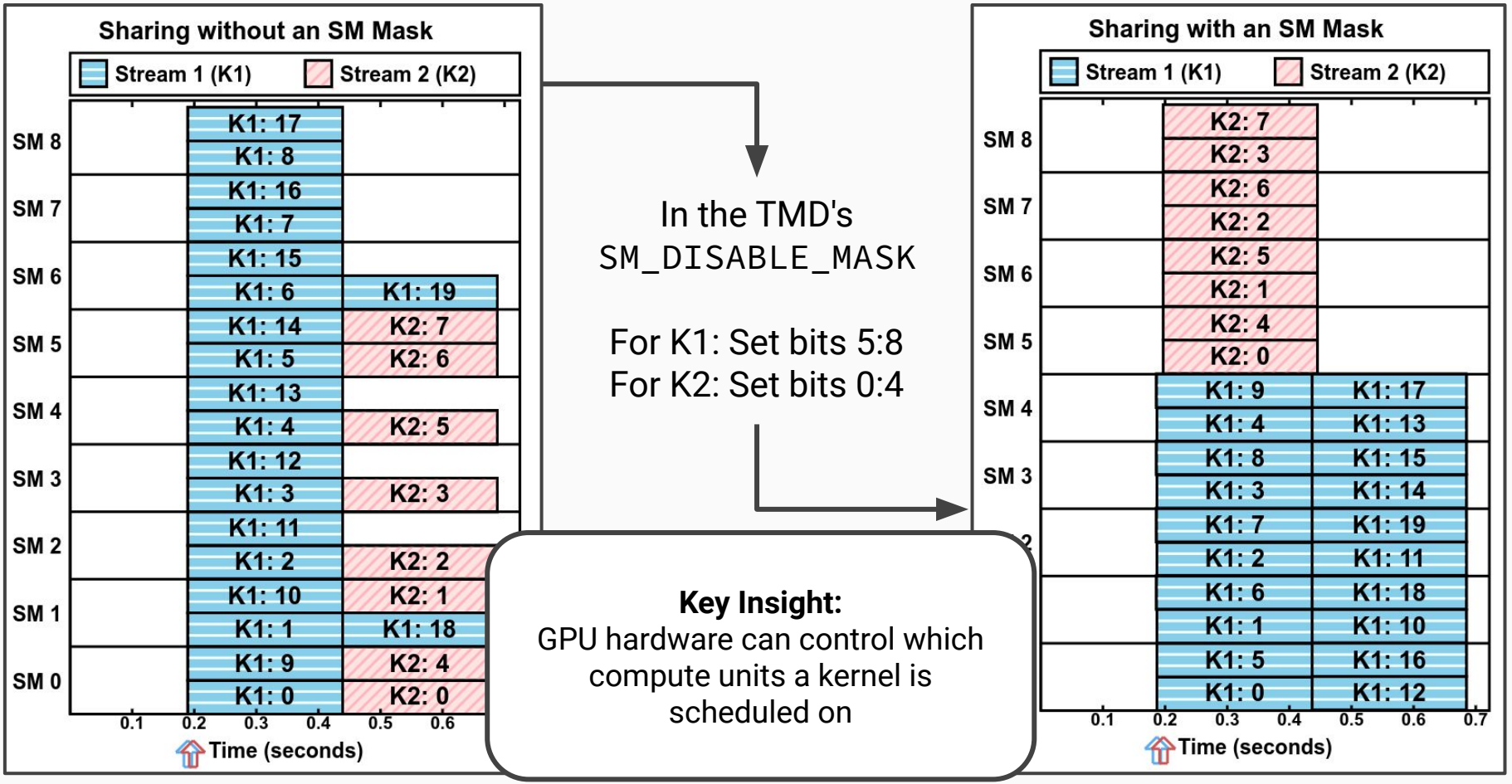
Each compute unit

Kernel launch times



Each *stream* is a FIFO of GPU kernels. Kernels must be in separate streams to execute concurrently

A timeline of which kernels (K_), execute which blocks (: __) on which GPU compute unit (SM)


Sharing without an SM Mask
Stream 1 (K1)
Stream 2 (K2)
SM 8
SM 7
SM 6
SM 5
SM 4
SM 3
SM 2
SM 1
SM 0
K1: 17
K1: 8
K1: 16
K1: 7
K1: 15
K1: 6
K1: 19
K1: 14
K2: 7
K1: 5
K2: 6
K1: 13
K1: 4
K2: 5
K1: 12
K1: 3
K2: 3
K1: 11
K1: 2
K2: 2
K1: 10
K2: 1
K1: 1
K1: 18
K1: 9
K2: 4
K1: 0
K2: 0
0.1
0.2
0.3
0.4
0.5
0.6
Time (seconds)
In the TMD's
SM_DISABLE_MASK
For K1: Set bits 5:8
For K2: Set bits 0:4
Key Insight:
GPU hardware can control which compute units a kernel is scheduled on
Sharing with an SM Mask
Stream 1 (K1)
Stream 2 (K2)
SM 8
SM 7
SM 6
SM 5
SM 4
SM 3
K2: 7
K2: 3
K2: 6
K2: 2
K2: 5
K2: 1
K2: 4
K2: 0
K1: 9
K1: 17
K1: 4
K1: 13
K1: 8
K1: 15
K1: 3
K1: 14
K1: 7
K1: 19
K1: 2
K1: 11
K1: 6
K1: 18
K1: 1
K1: 10
K1: 5
K1: 16
K1: 0
K1: 12
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Time (seconds)

Enabling Flexible Compute Partitioning

Goal 2 of 3

Flexible Partitioning

Given working partitioning, is it flexible and reliable enough to be useful?

Means of answering:

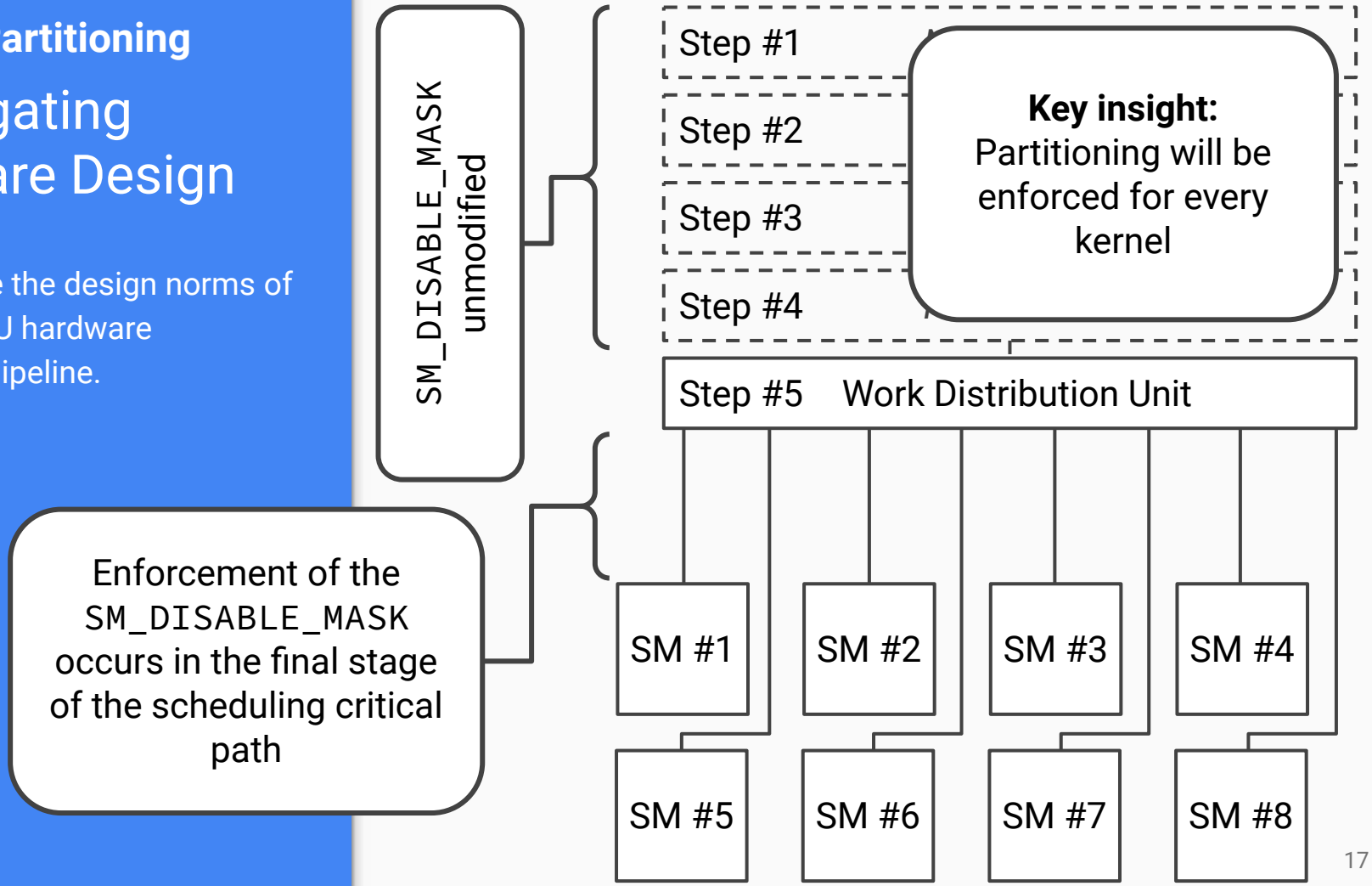
1. Investigate hardware design
2. Test with benchmarks

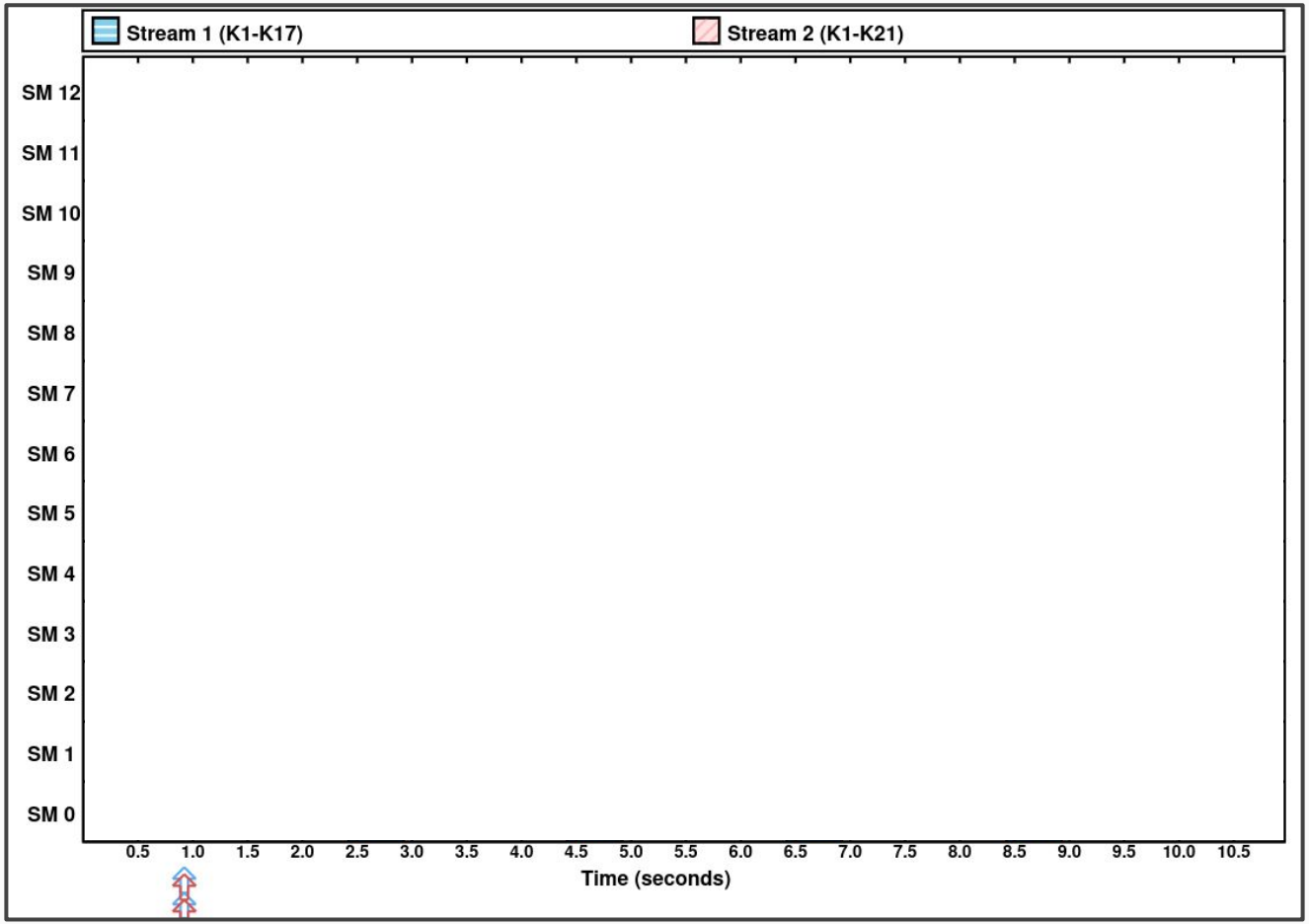
We do both.

Flexible Partitioning

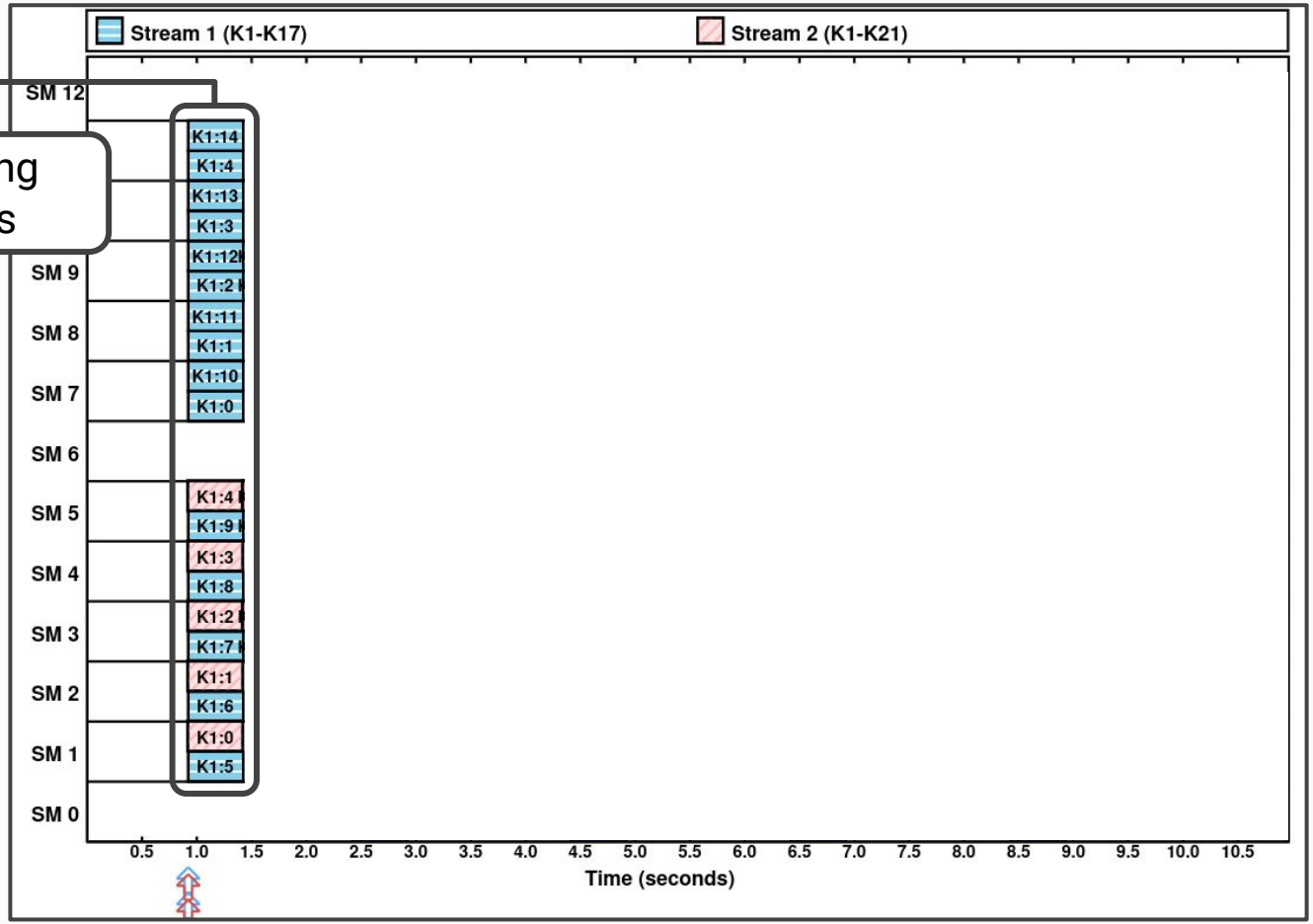
Investigating Hardware Design

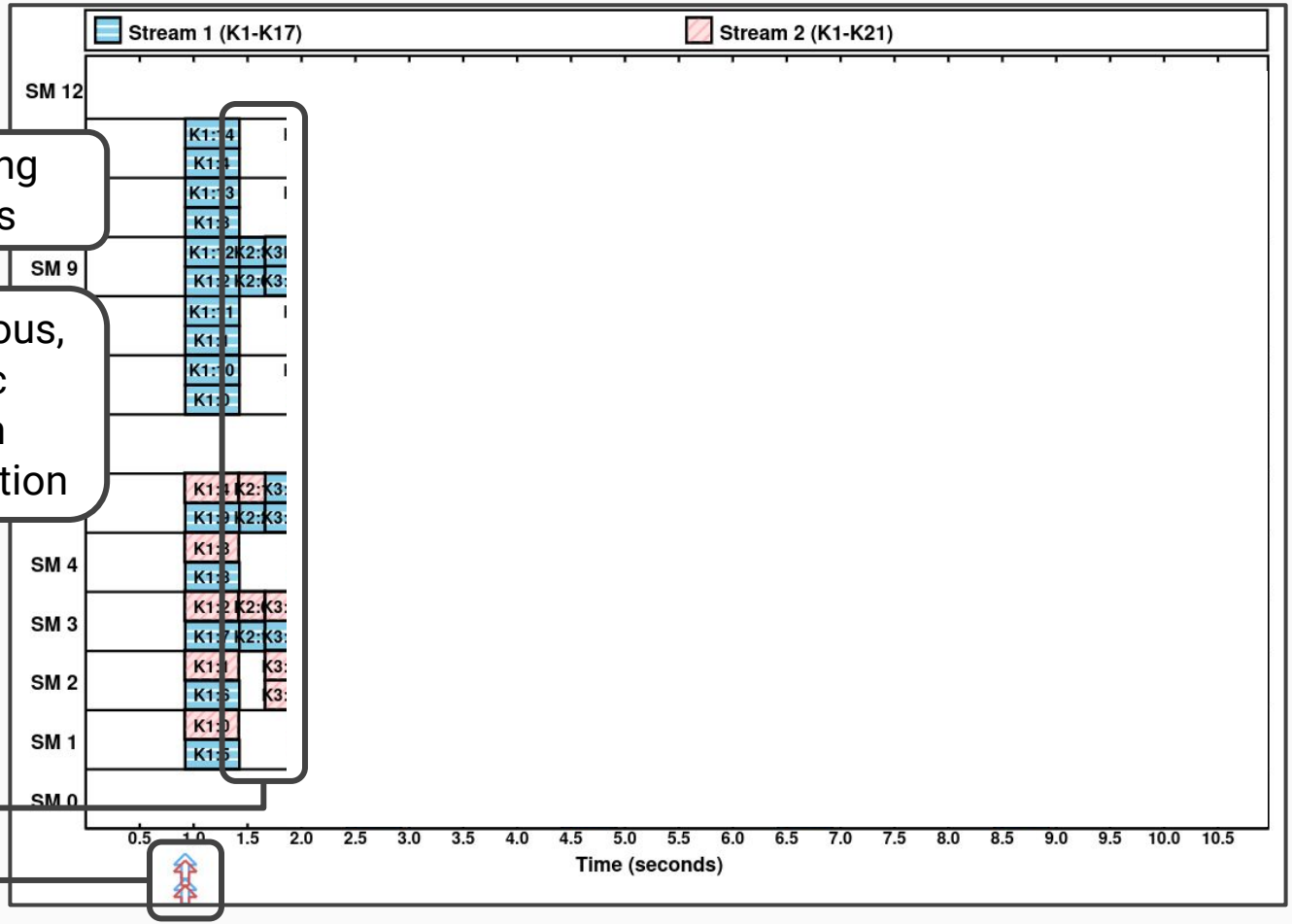
We elucidate the design norms of NVIDIA's GPU hardware scheduling pipeline.




Stream 1 (K1-K17)
Stream 2 (K1-K21)
SM 12
SM 11
SM 10
SM 9
SM 8
SM 7
SM 6
SM 5
SM 4
SM 3
SM 2
SM 1
SM 0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0
10.5
Time (seconds)

Overlapping partitions





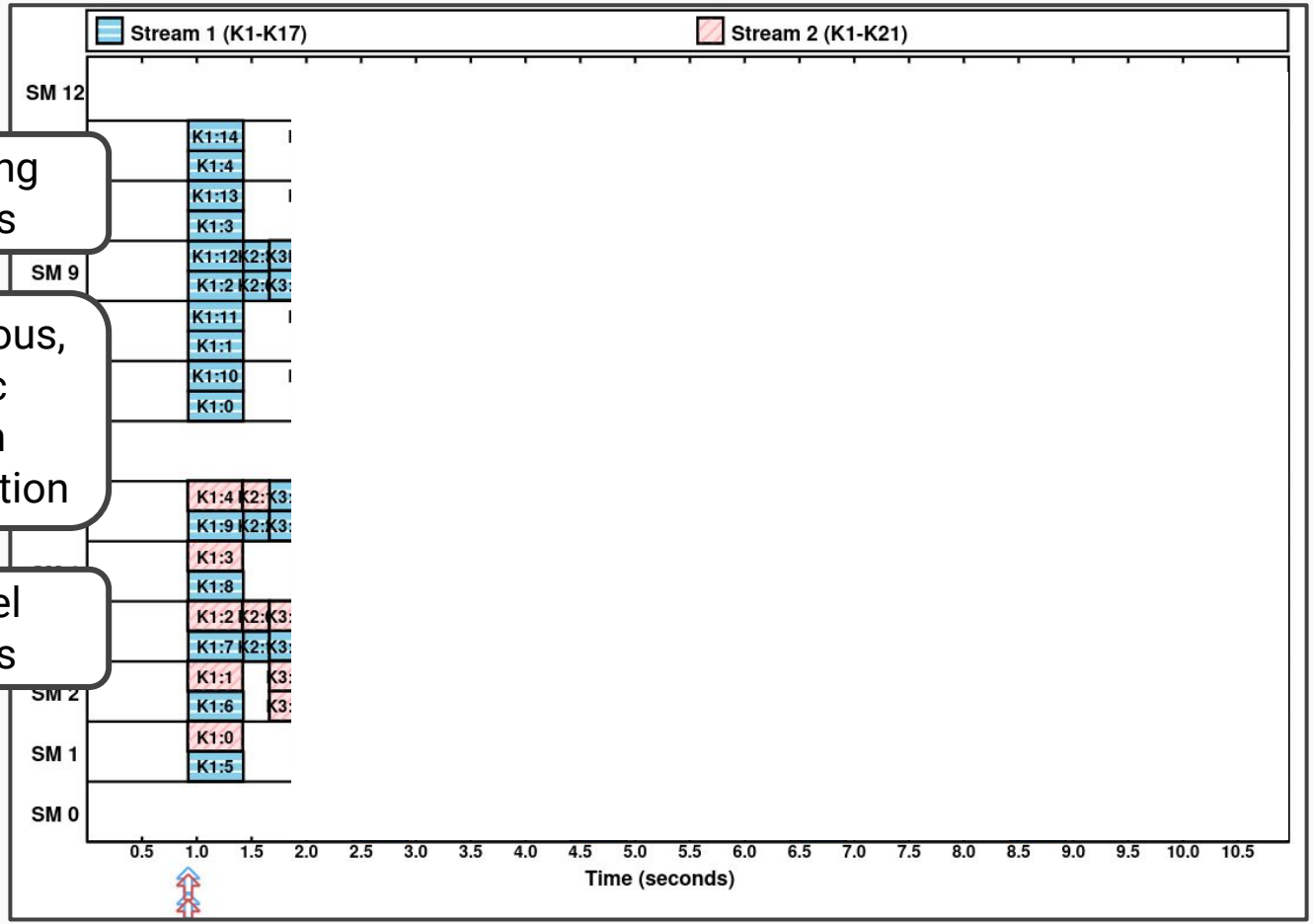
Overlapping partitions

Asynchronous, dynamic partition reconfiguration

Overlapping partitions

Asynchronous, dynamic partition reconfiguration

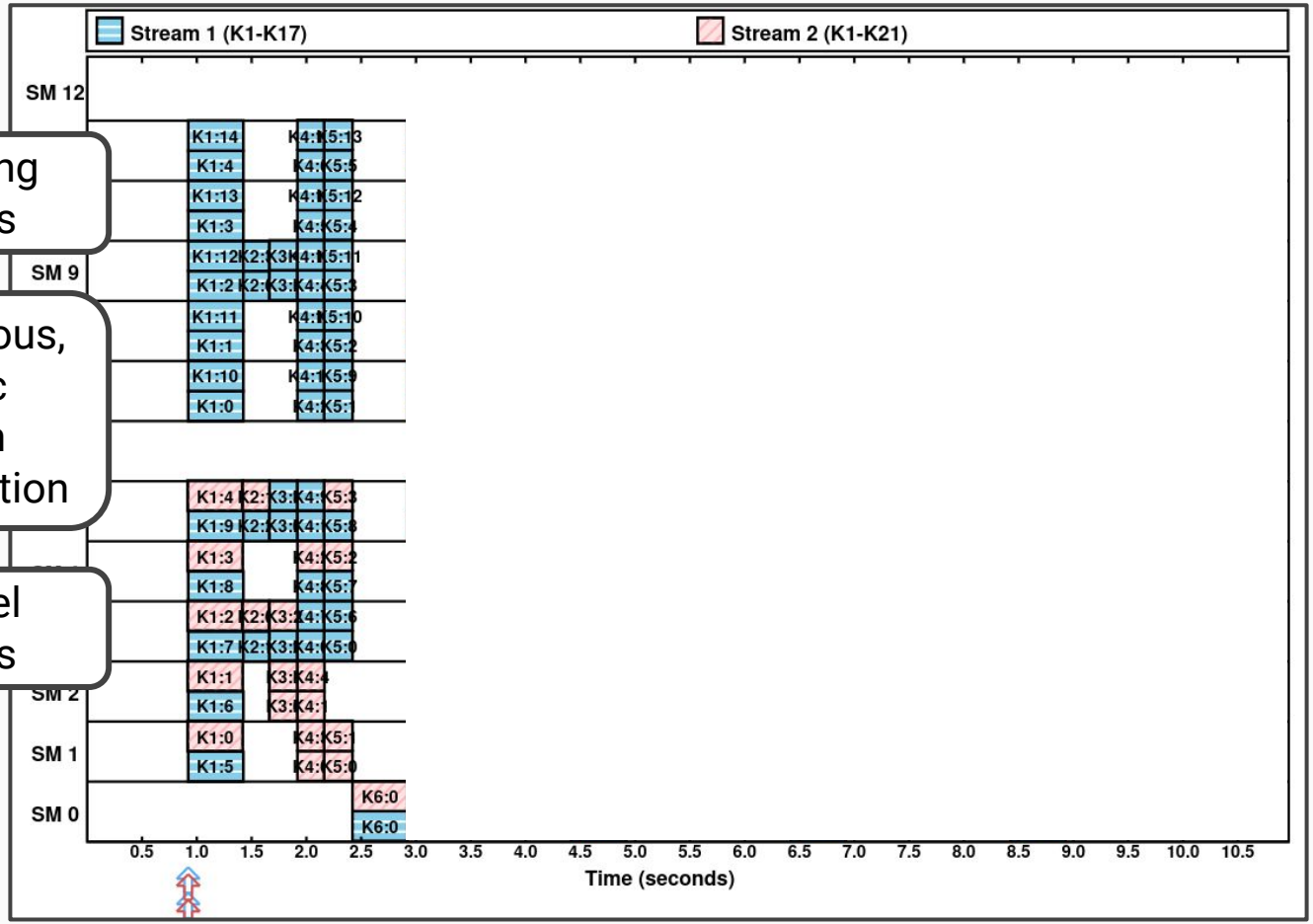
Per-kernel partitions



Overlapping partitions

Asynchronous, dynamic partition reconfiguration

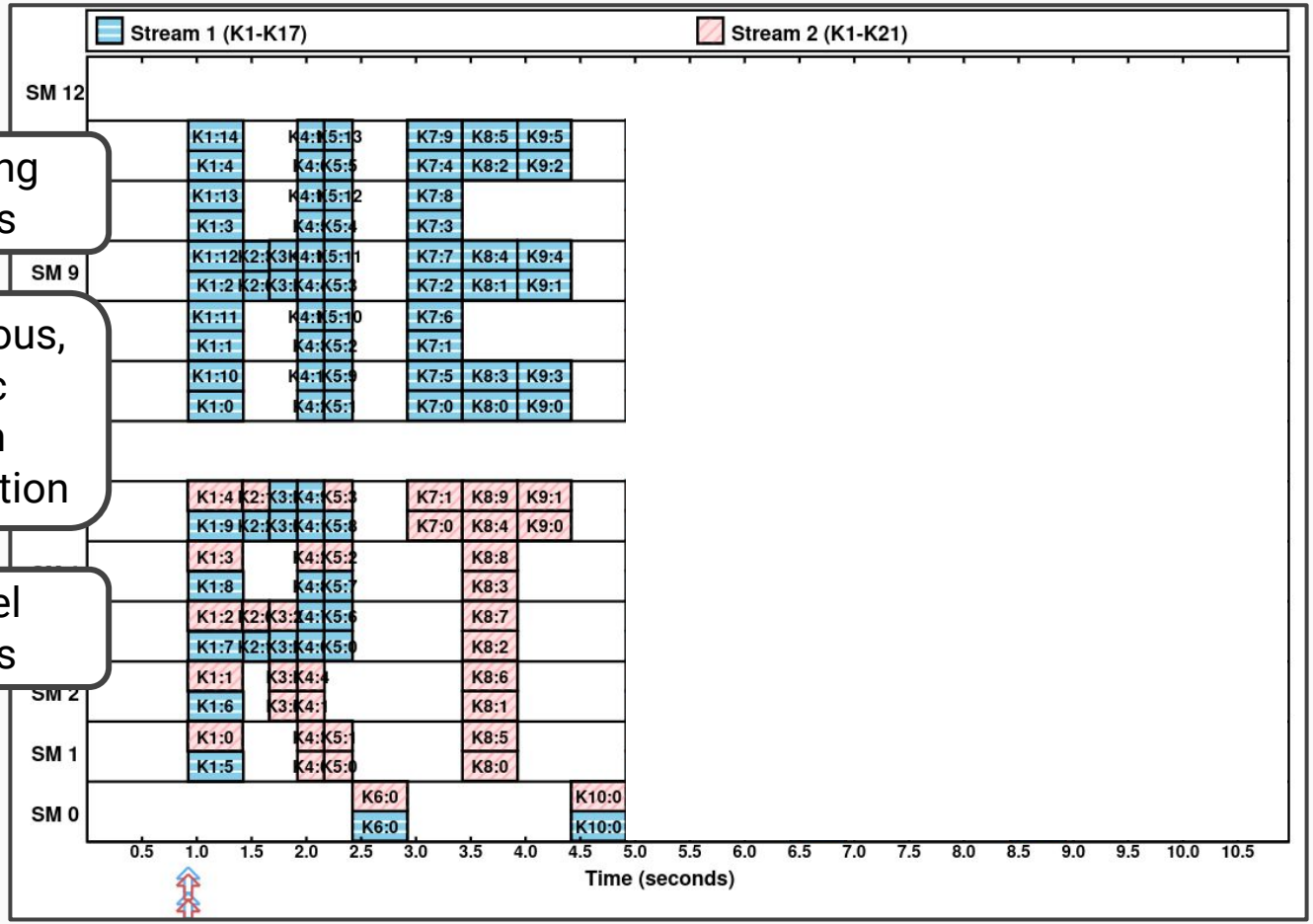
Per-kernel partitions



Overlapping partitions

Asynchronous, dynamic partition reconfiguration

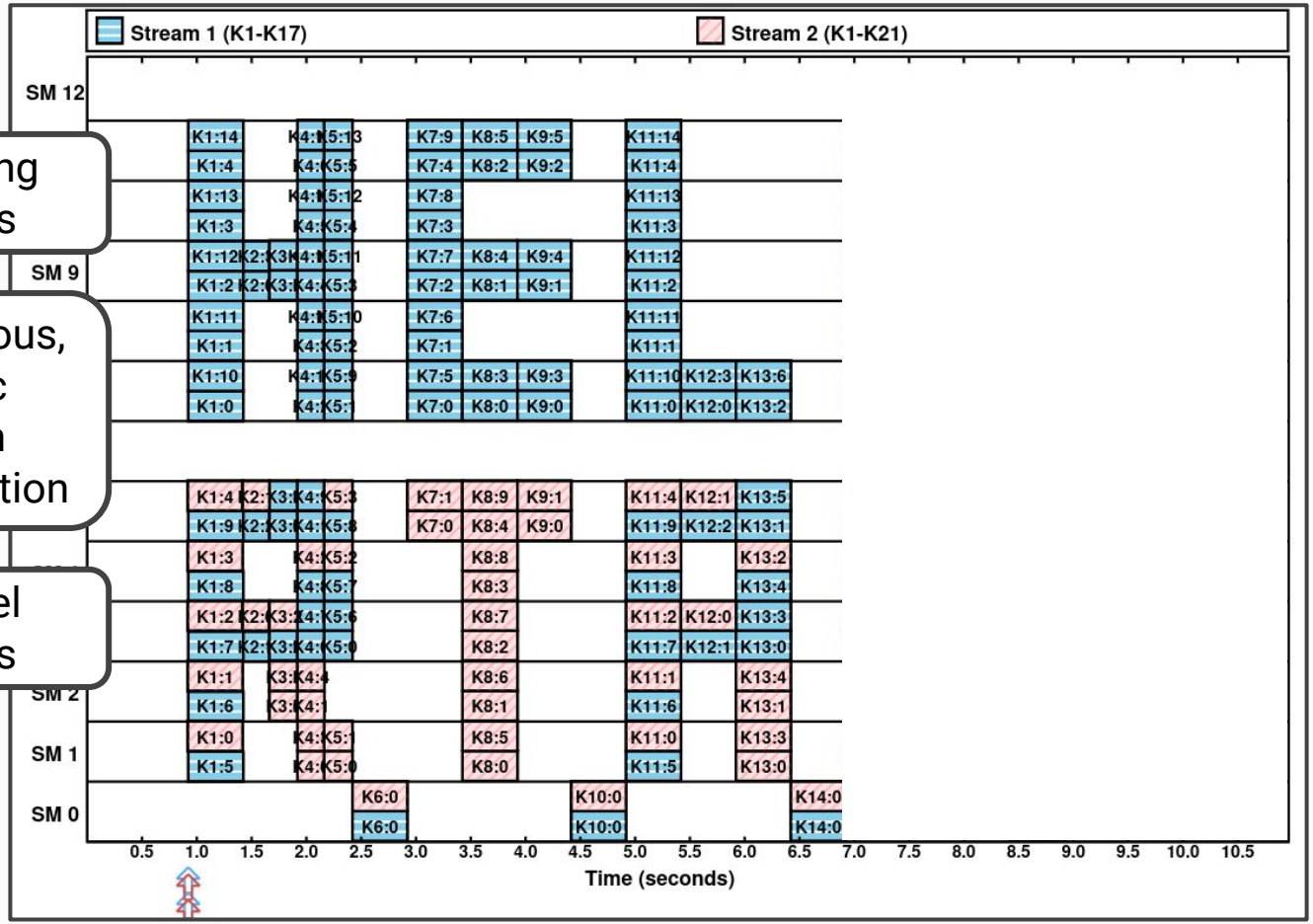
Per-kernel partitions



Overlapping partitions

Asynchronous, dynamic partition reconfiguration

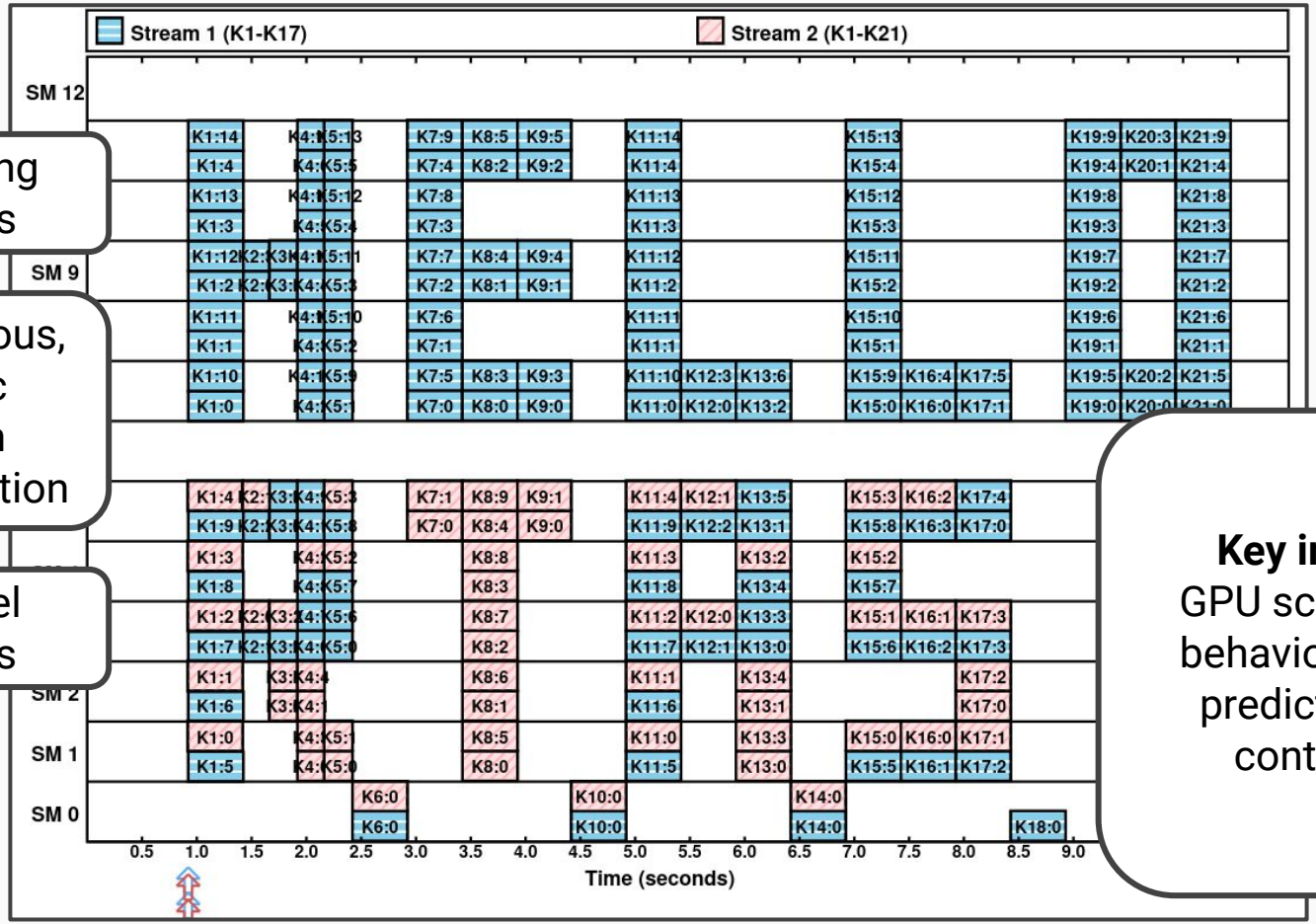
Per-kernel partitions



Overlapping partitions

Asynchronous, dynamic partition reconfiguration

Per-kernel partitions



Key insight:
GPU scheduling behavior can be predicted and controlled

Enabling Easily Applicable Compute Partitioning

Goal 3 of 3

Easily Applicable Partitioning

A simple API

Very portable: Works on any NVIDIA GPU of compute capability >3.5 (2013) with CUDA >10.2 (2019)

Key insight: GPU scheduling hardware changes little generation-to-generation

On Linux:

0. Download `libsmctrl.h` and `libsmctrl.so`
1. `#include "libsmctrl.h"` and add `-lsmctrl`
2. `libsmctrl_set_global_mask(uint64_t default_mask)`
3. `libsmctrl_set_stream_mask(cudaStream_t, uint64_t mask);`

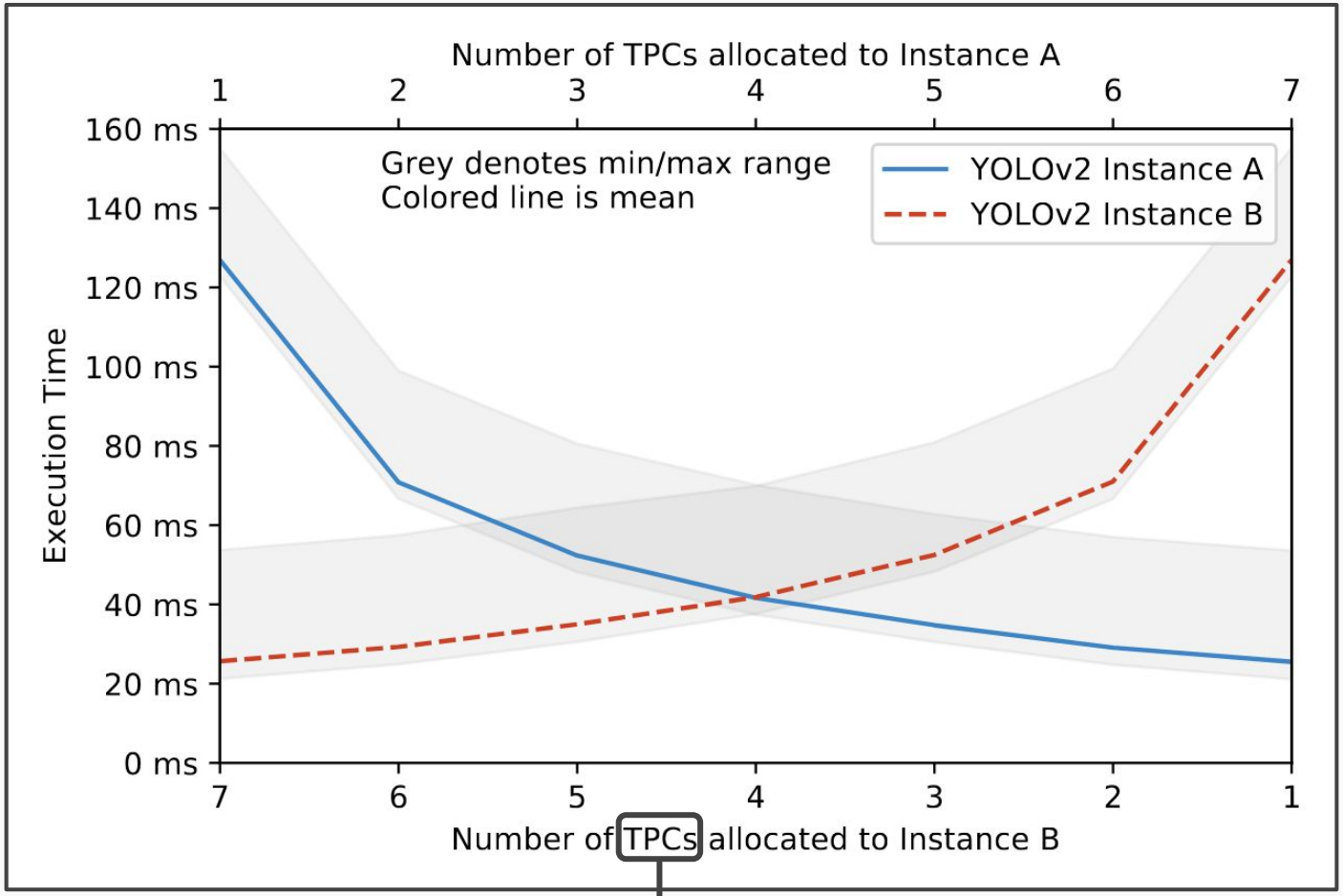
No kernel configuration, no driver configuration, and no superuser permissions.

Code is open source and documented. See https://www.cs.unc.edu/~jbakita/rtas23-ae.html to get started.

Consider 2 co-running instances of YOLOv2 in DarkNet

Only required three lines of changes to enable partitioning

1 TPC = 2 SMs



Conclusions

We build spatial partitioning for GPU compute units that is:

Hardware-Enforced	**Flexible**	**Easily Applicable**
Is there a hardware capability?	How can we be confident this will work widely?	Can we make GPU spatial-partitioning easy?
Yes, the `SM_DISABLE_MASK`	Hardware norms, benchmarks, and real-world software support it	Yes, via our 1-line, no-install Linux API

What you have to read the paper for…

Evaluation:

- Adversarial tests
- How GPU pitfalls noted in prior work still effect partitioned GPUs
- Hazards of overlapping partitions
- Comparison to prior work
- Full details on our system setup and configuration

API:

- Details on *how* we modify the TMD
- Full details on our supported API calls, with examples
- Details on our API to query GPU silicon configuration
- List of every GPU, CUDA version, and CPU architecture we tested portability on

Regarding GPUs:

- Distinction
- Extensive details on the NVIDIA GPU hardware scheduling pipeline, including:
  - The Host Interface
  - The Compute Front End
  - The Task Management Unit
  - The Work Distribution Unit
  - CPU-to-GPU Buffer Design
- GPU cache hierarchy and bus interconnect layout

\+ More details and background on everything covered in this presentation

Thanks!
Questions?

Future work:

- ➔ Cross-context partitioning
- ➔ Criticality-Aware time-slice scheduling

Contact:
Email: jbakita@cs.unc.edu
Twitter: @JJBakita
Web: https://cs.unc.edu/~jbakita

Old Well, University of North Carolina at Chapel Hill, Winter 2017