WHAT THE PROFILER IS TELLING YOU: OPTIMIZING YOUR CUDA APPLICATION

XAVIER SIMMONS | 2023

xsimmons@nvidia.com

AGENDA

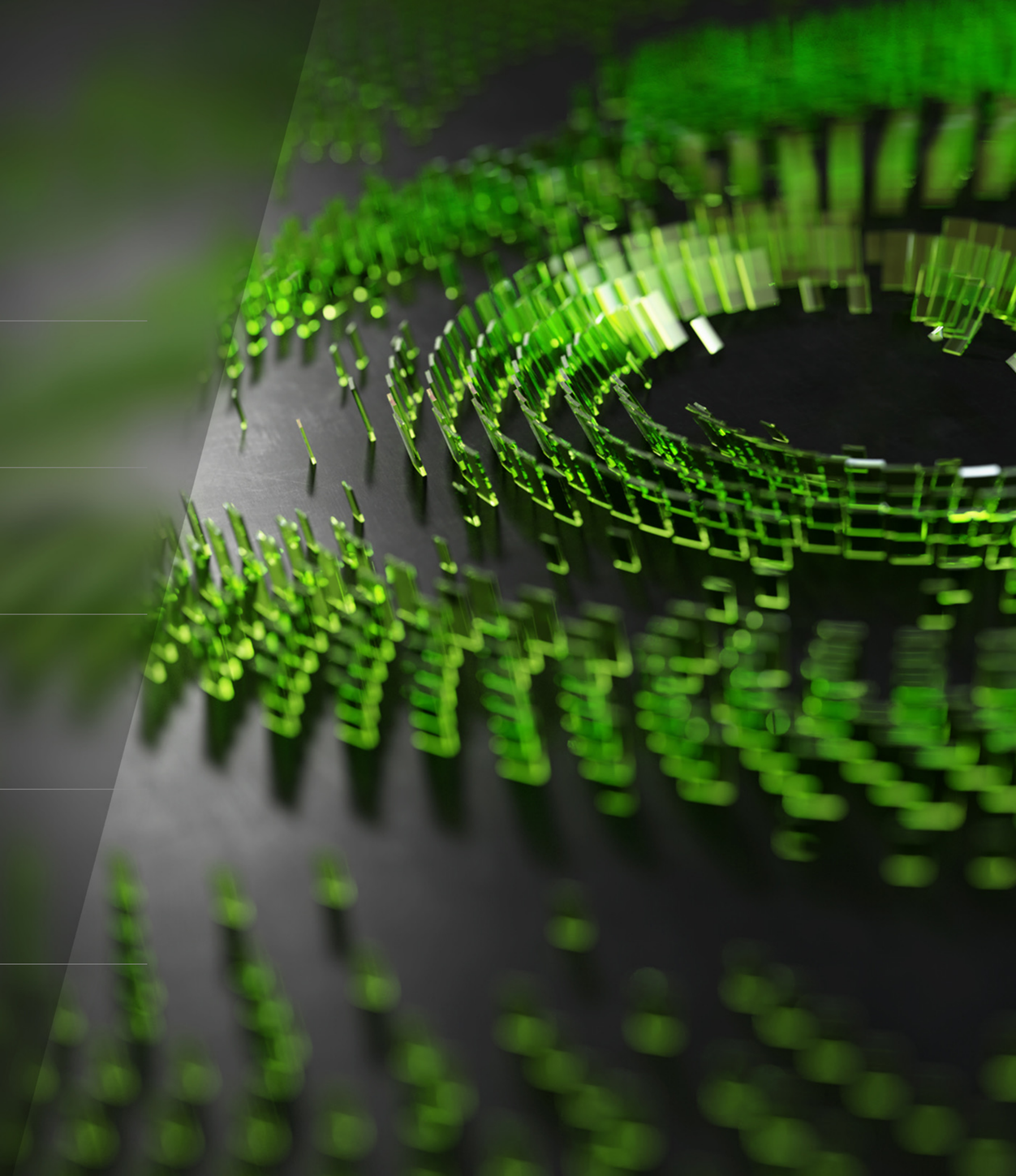
Introduction
Guiding the Optimization Effort

Understand Your Hardware
Ampere Overview

Understand Your Application – Identify Hot Spot
Nsight Systems

Optimizing Your CUDA Kernel
Nsight Compute – Understanding Latency, Compute and Memory

Summary
Closing comments, Extra Learning Resources



BEFORE YOU START

Steps to Enlightenment

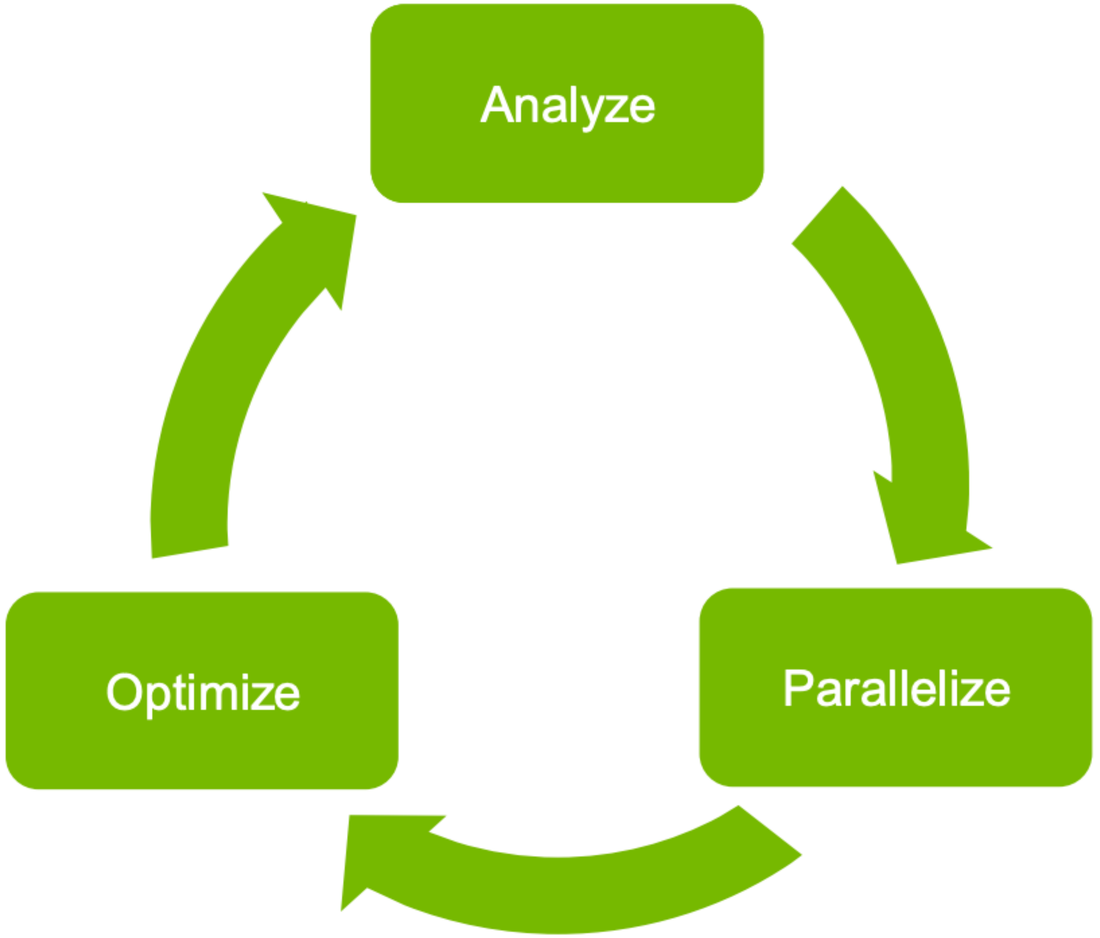
- Know your **application**
  - What does it compute? How can it be parallelized? What final performance is expected?
- Know your **hardware**
  - What are your target machines? How many GPUs or Nodes? Hardware-specific optimizations okay?
- Know your **profiling tools**
  - Strengths and weaknesses of each tool - learn how to use them (learn a couple and learn them well!)
- Know your **process**
  - Performance optimization is a constant cyclic learning process

DEVELOPMENT CYCLE

Optimization Workflow

- **Analyze** your code to determine most likely places that need parallelization or optimization
- **Parallelize** your code by starting with the most time-consuming parts and check for correctness
- **Optimize** your code to improve observed speed-up from parallelization. Make **iterative** changes!

This is an iterative learning process!



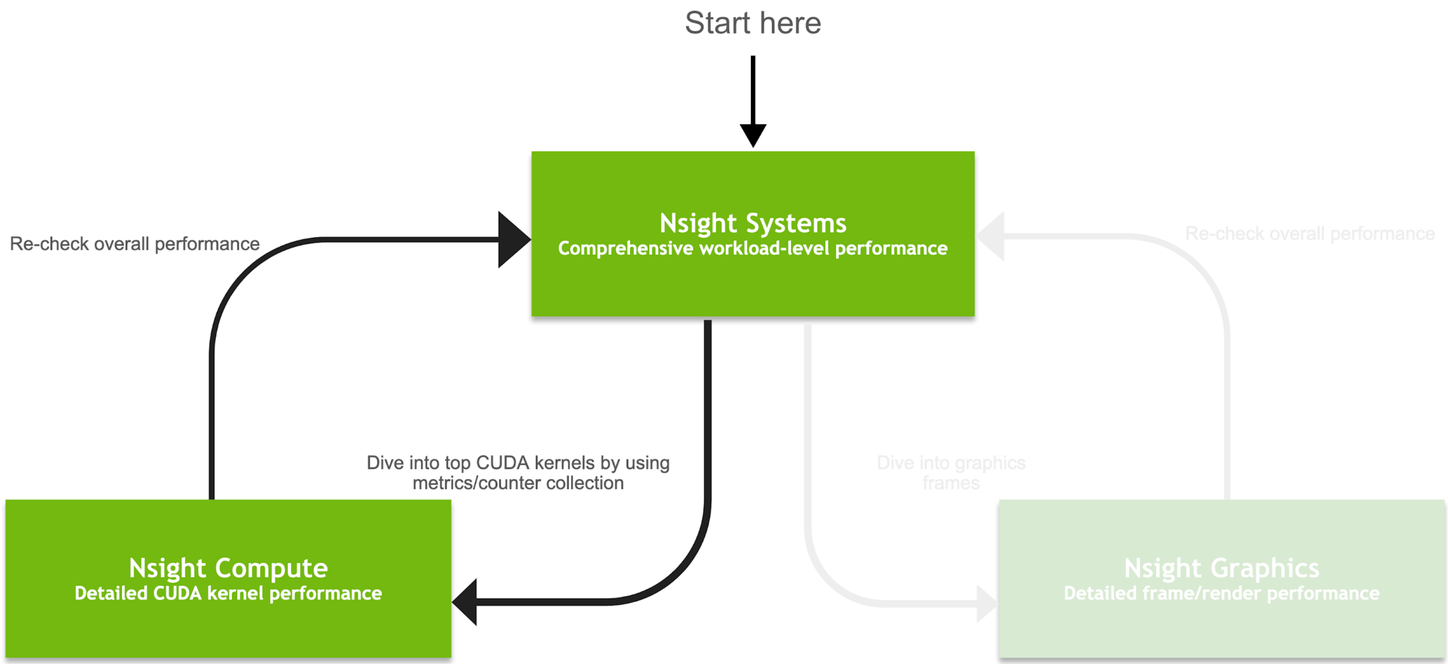
NSIGHT PRODUCT FAMILY

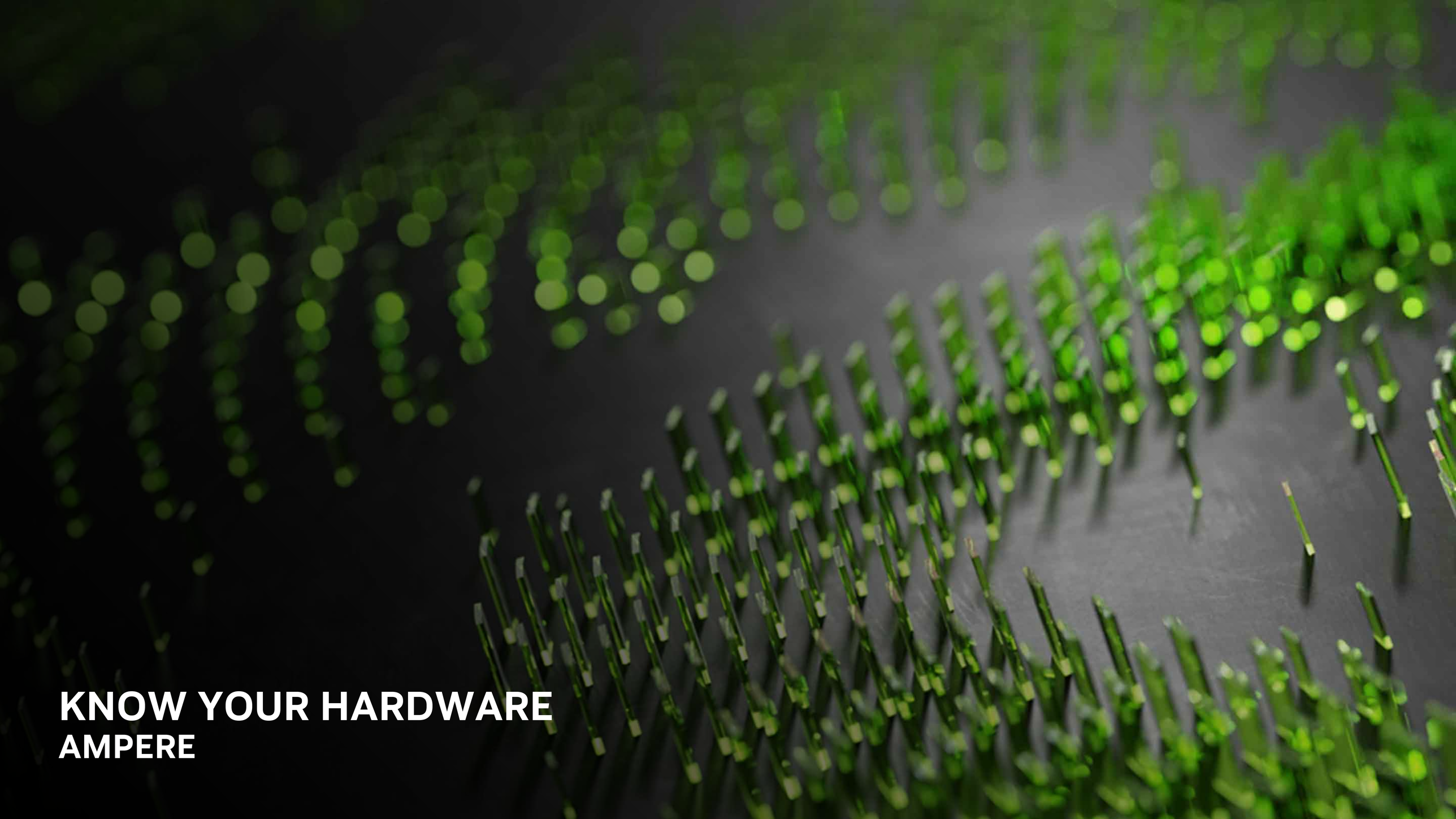
Optimization Workflow

Nsight Systems
Analyze application algorithm system-wide

Nsight Compute
Debug/optimize CUDA kernel

Nsight Graphics
Debug/optimize graphics workloads

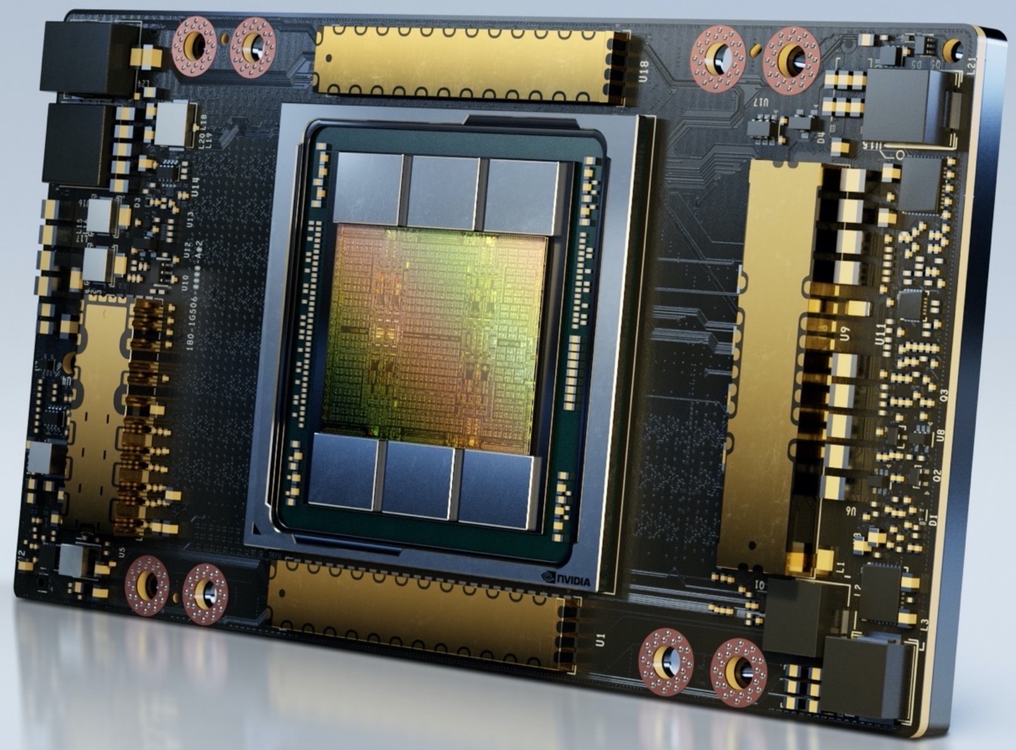



KNOW YOUR HARDWARE
AMPERE

NVIDIA A100 GPU

Generational Leap from Volta

	V100	A100
Double | Single TFLOP/s	7.8 | 15.7	9.7 | 19.5
FP32 Tensor Core TFLOPS	125	156 | 312 sparsity
Memory Bandwidth	900 GB/s	2 TB/s
Memory Size	32 GB	80 GB
L2 Cache Size	6144 KB	40 MB
Base/Boost Clock (Mhz)	1312 / 1530	765 / 1410
TDP (Watts)	300	400



AMPERE RESOURCES | STREAMING MULTIPROCESSOR (SM)

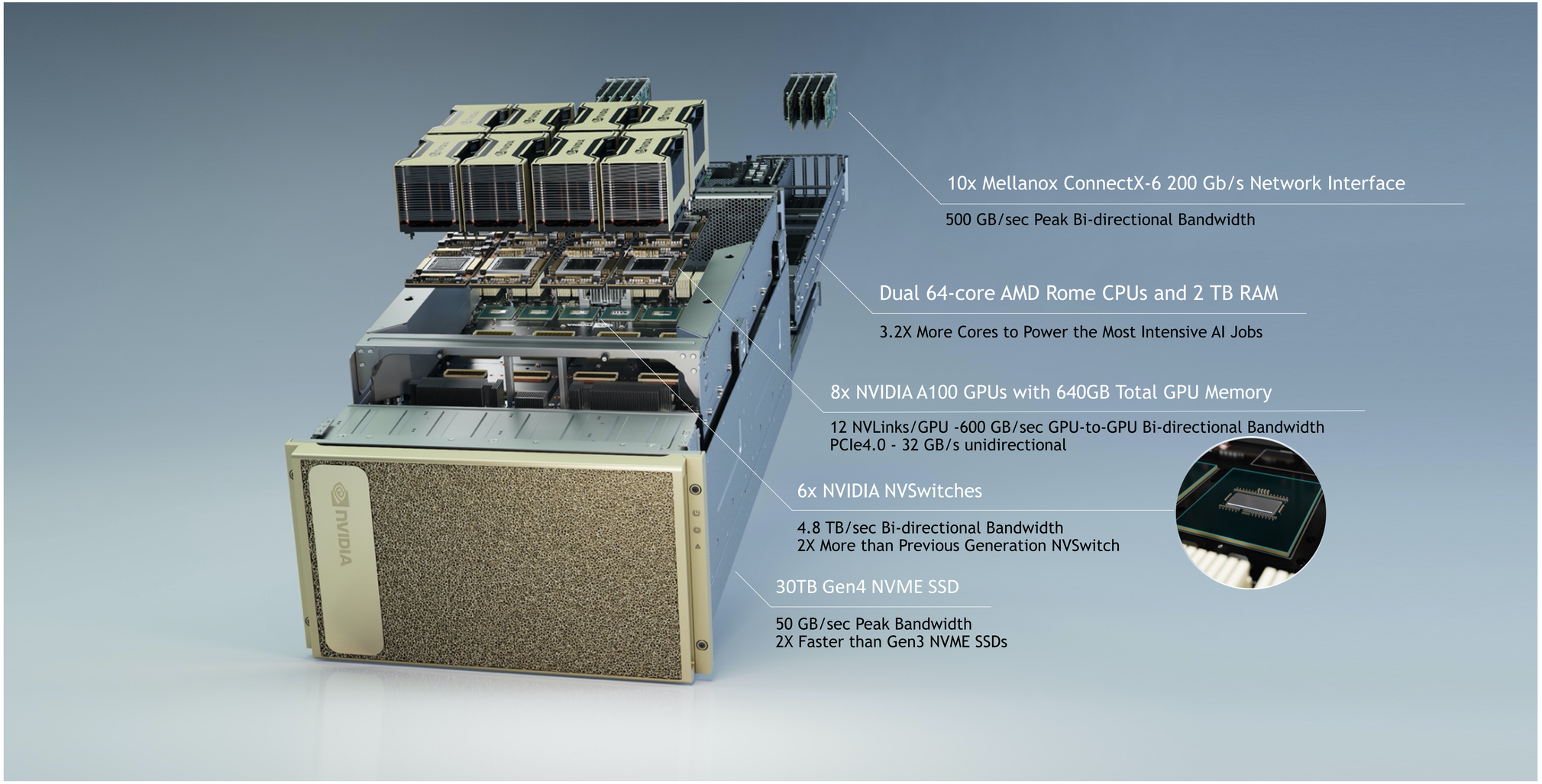
Know your GPU

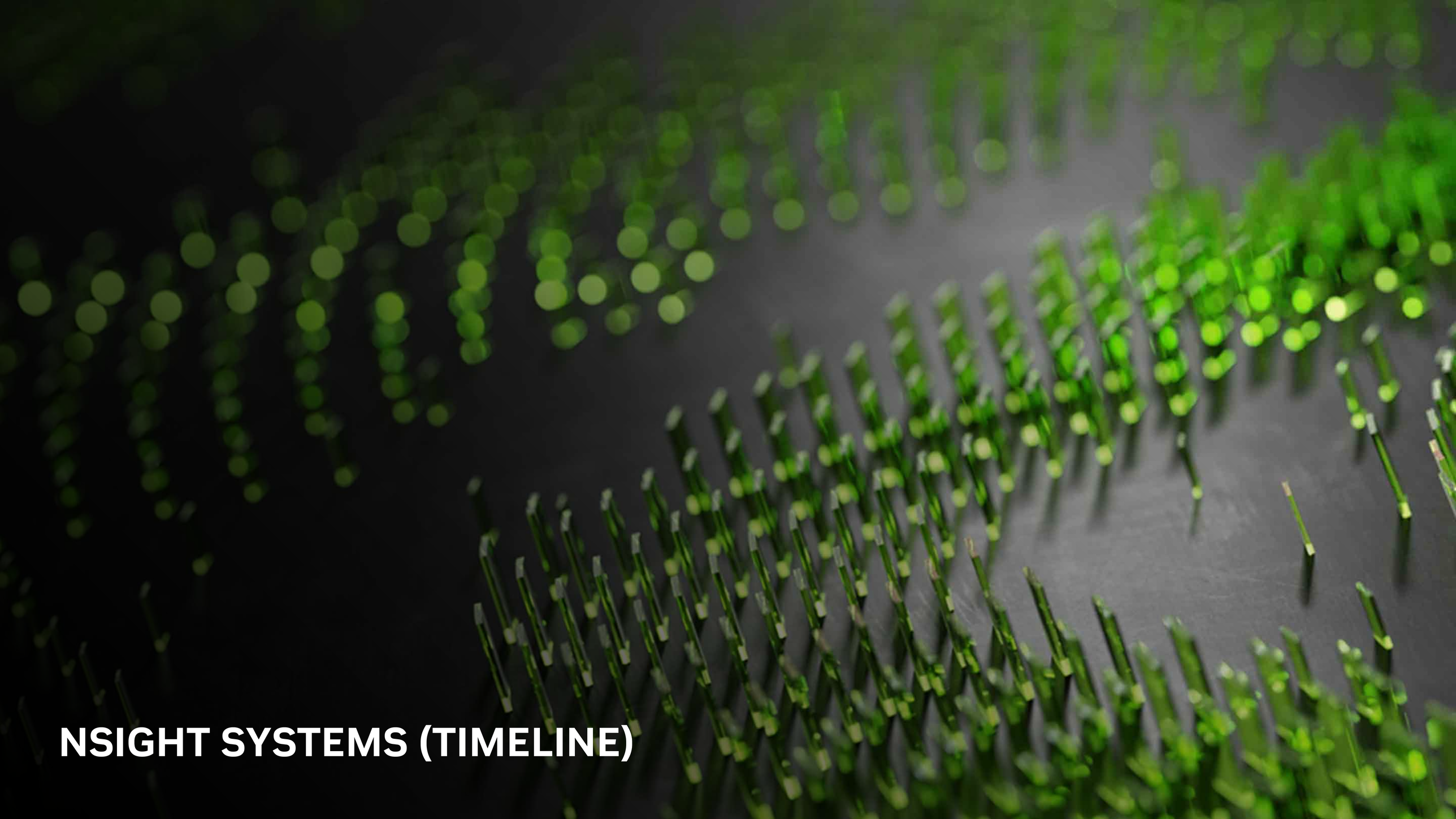
- Each thread block needs:
  - **Registers** (#registers/thread x #threads)
  - **Shared memory** (0 - 164KB)

	V100	A100
No. SMs	80	108
INT32 cores	64	64
FP32 cores	64	64
FP64 cores	32	32
Register File	256 KB	256 KB
Shared Memory	96 KB	164 KB
Active Blocks/SM	32	32
Max Threads/SM	2048	2048

SYSTEM CONFIGURATION

NVIDIA DGX A100 640GB System - Know your System





NSIGHT SYSTEMS (TIMELINE)

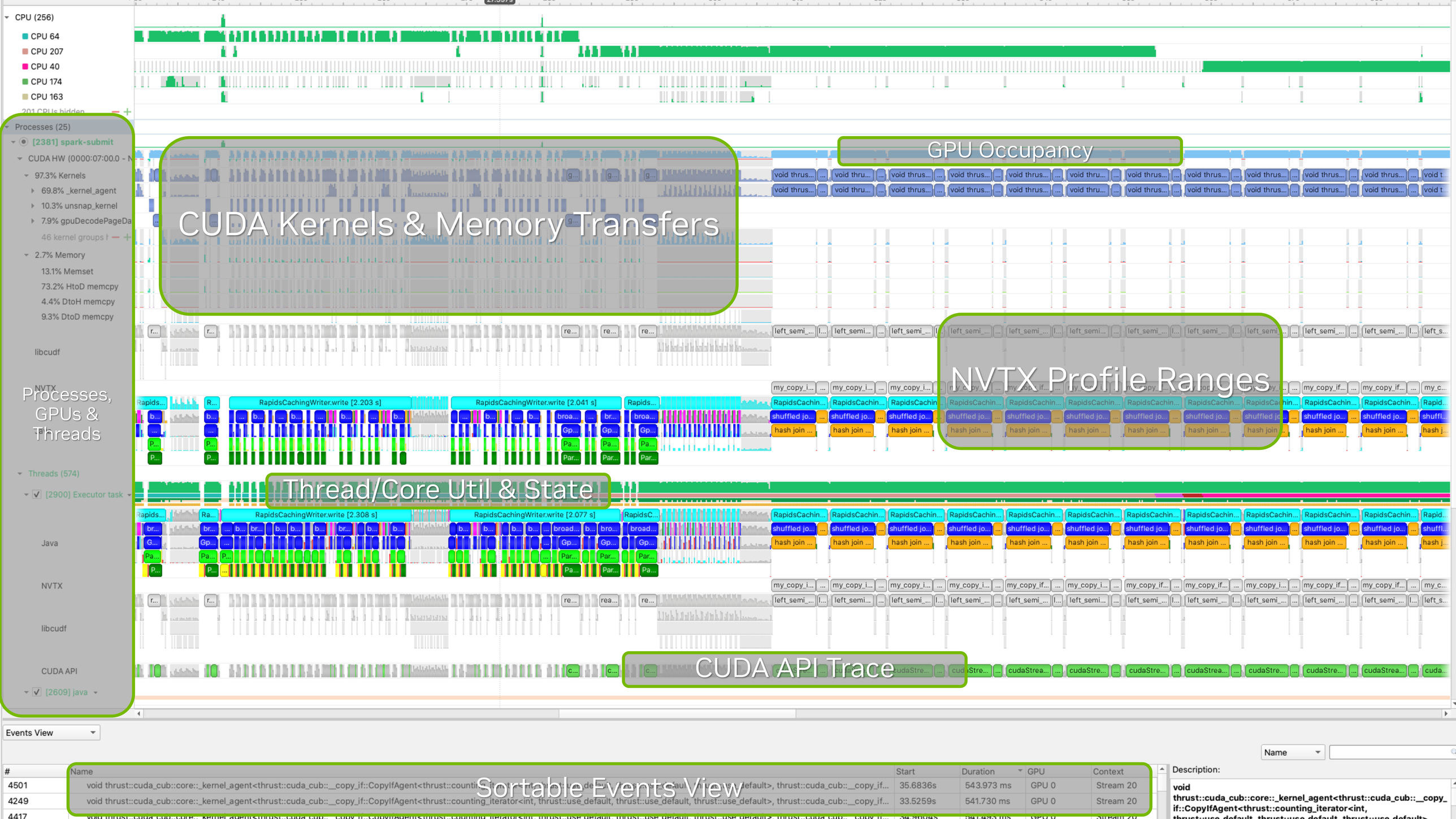
NSIGHT SYSTEMS PROFILE

Profile with CLI example

```
$ nsys profile -t cuda,osrt,nvtx,cudnn,cublas \   ← APIs to be traced
               -y 60 \                            ← Delayed profile (sec)
               -d 20 \                            ← Profiling duration (sec)
               -o baseline \                      ← Output filename
               -f true \                          ← Overwrite when it's true
               -w true \                          ← Display
               python my_app.py                   ← Execution command
```

cuda	- GPU kernel
osrt	- OS runtime (e.g pread, pthread, etc)
nvtx	- NVIDIA Tools Extension
cudnn	- CUDA Deep NN library
cublas	- CUDA BLAS library

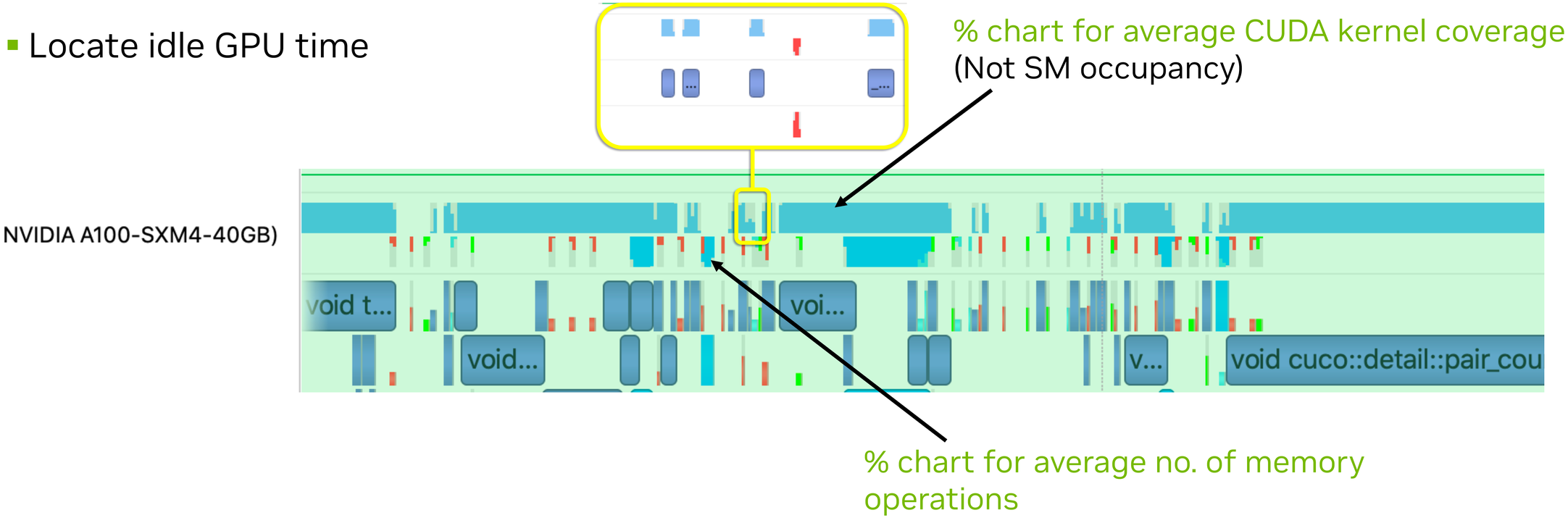
https://docs.nvidia.com/nsight-systems/UserGuide/#cli-profiling


CPU (256)
CPU 64
CPU 207
CPU 40
CPU 174
CPU 163
Processes (25)
[2381] spark-submit
CUDA HW (0000:07:00.0 - N
97.3% Kernels
69.8% _kernel_agent
10.3% unsnap_kernel
7.9% gpuDecodePageDa
46 kernel groups t
2.7% Memory
13.1% Memset
73.2% HtoD memcpy
4.4% DtoH memcpy
9.3% DtoD memcpy
libcudf
NVTX
Threads (574)
[2900] Executor task
Java
NVTX
libcudf
CUDA API
[2609] java
CUDA Kernels & Memory Transfers
GPU Occupancy
NVTX Profile Ranges
Processes, GPUs & Threads
Thread/Core Util & State
CUDA API Trace
RapidsCachingWriter.write [2.203 s]
RapidsCachingWriter.write [2.041 s]
RapidsCachingWriter.write [2.308 s]
RapidsCachingWriter.write [2.077 s]
Events View
Name
Start
Duration
GPU
Context
4501
4249
35.6836s
543.973 ms
33.5259s
541.730 ms
GPU 0
Stream 20
Description:
Sortable Events View

GPU WORKLOAD

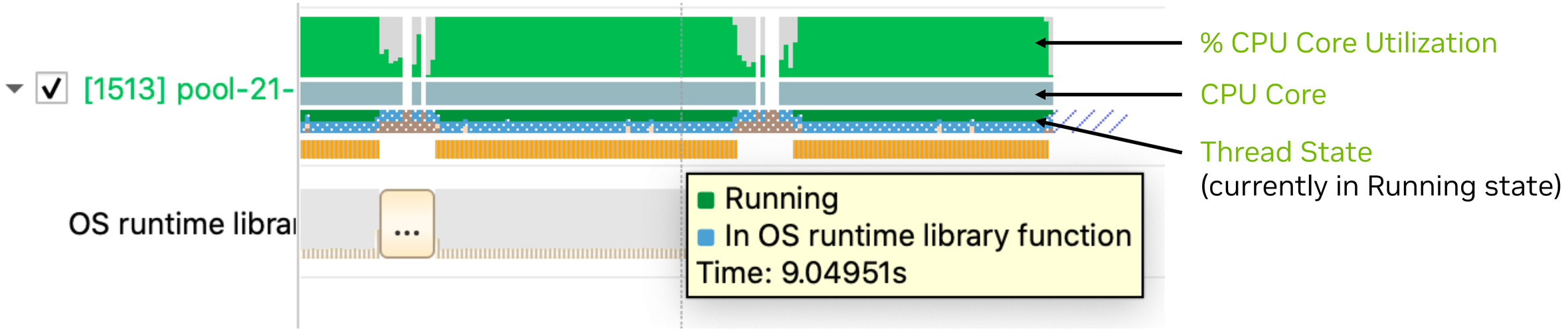
A Closer Look

- See trace of GPU activity
- Locate idle GPU time



CPU THREADS

Thread Activities



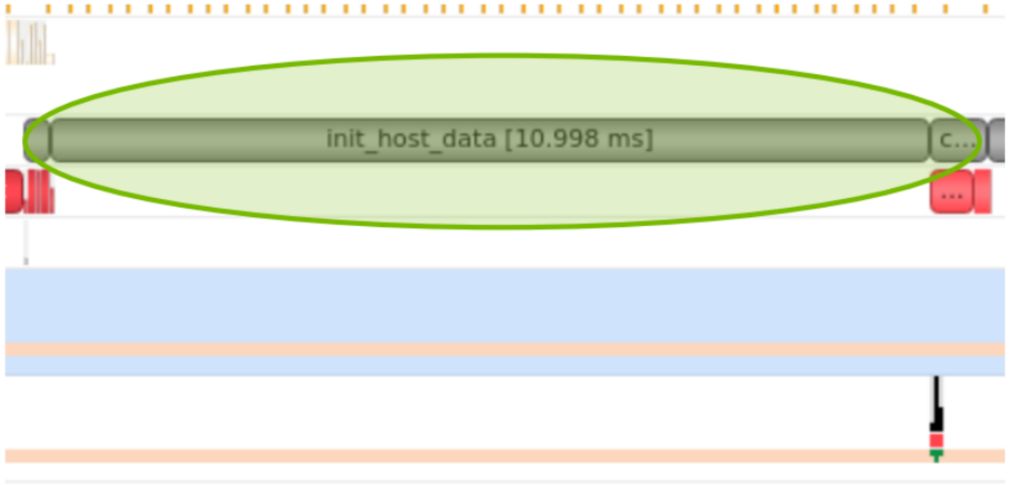
NVTX INSTRUMENTATION

Profile Ranges of Code

- NVIDIA Tools Extension (NVTX) is a library installed with CUDA to annotate events and ranges

- Include the header “nvToolsExt.h” (CUDA < 10.0 also link to shared library –lnvToolsExt)

```
#include "nvToolsExt.h"
...
void myfunction( int n, double * x )
{
    nvtxRangePushA("init_host_data");
    //initialize x on host
    init_host_data(n,x,x_d,y_d);
    nvtxRangePop();
}
...
```



NSIGHT SYSTEMS

Don’t want GUI? Use ‘**stats**’

```
$ nsys stats baseline.nsys-rep
```

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
63.3	6,529,893,517	19,849	328,978.5	9,799.0	2,314	176,542,667	3,253,041.5	cudaStreamSynchronize_ptsz
12.4	1,274,207,866	16,420	77,601.0	15,369.0	201	12,112,333	539,137.9	cudaMemcpyAsync_ptsz
9.7	1,003,014,674	1	1,003,014,674.0	1,003,014,674.0	1,003,014,674	1,003,014,674	0.0	cudaMallocHost
8.4	862,187,549	3	287,395,849.7	4,389.0	3,276	862,179,884	497,777,575.4	cudaFree
2.5	253,168,144	22,343	11,331.0	2,073.0	791	167,495,583	1,138,314.7	cudaMallocFromPoolAsync_v11020
1.4	143,342,402	1	143,342,402.0	143,342,402.0	143,342,402	143,342,402	0.0	cudaMemPoolDestroy_v11020
1.1	111,011,459	15,767	7,040.7	6,011.0	4,448	183,584	4,639.2	cudaLaunchKernel_ptsz
0.6	57,396,670	547	104,929.9	6,672.0	3,897	3,696,682	280,680.0	cudaHostAlloc
0.3	34,059,794	22,343	1,524.4	1,323.0	902	95,198	1,155.9	cudaFreeAsync_v11020
0.2	24,649,124	547	45,062.4	6,172.0	4,839	471,714	97,252.7	cudaFreeHost
0.2	21,114,478	8,227	2,566.5	2,825.0	220	76,984	2,879.2	cudaMemsetAsync_ptsz
0.0	143,709	373	385.3	331.0	160	2,114	254.3	cuGetProcAddress
0.0	104,997	2	52,498.5	52,498.5	26,700	78,297	36,484.6	cudaMemGetInfo
0.0	40,646	1	40,646.0	40,646.0	40,646	40,646	0.0	cudaMemPoolCreate_v11020
0.0	2,204	1	2,204.0	2,204.0	2,204	2,204	0.0	cudaMemPoolSetAttribute_v11020
0.0	1,773	1	1,773.0	1,773.0	1,773	1,773	0.0	cuInit

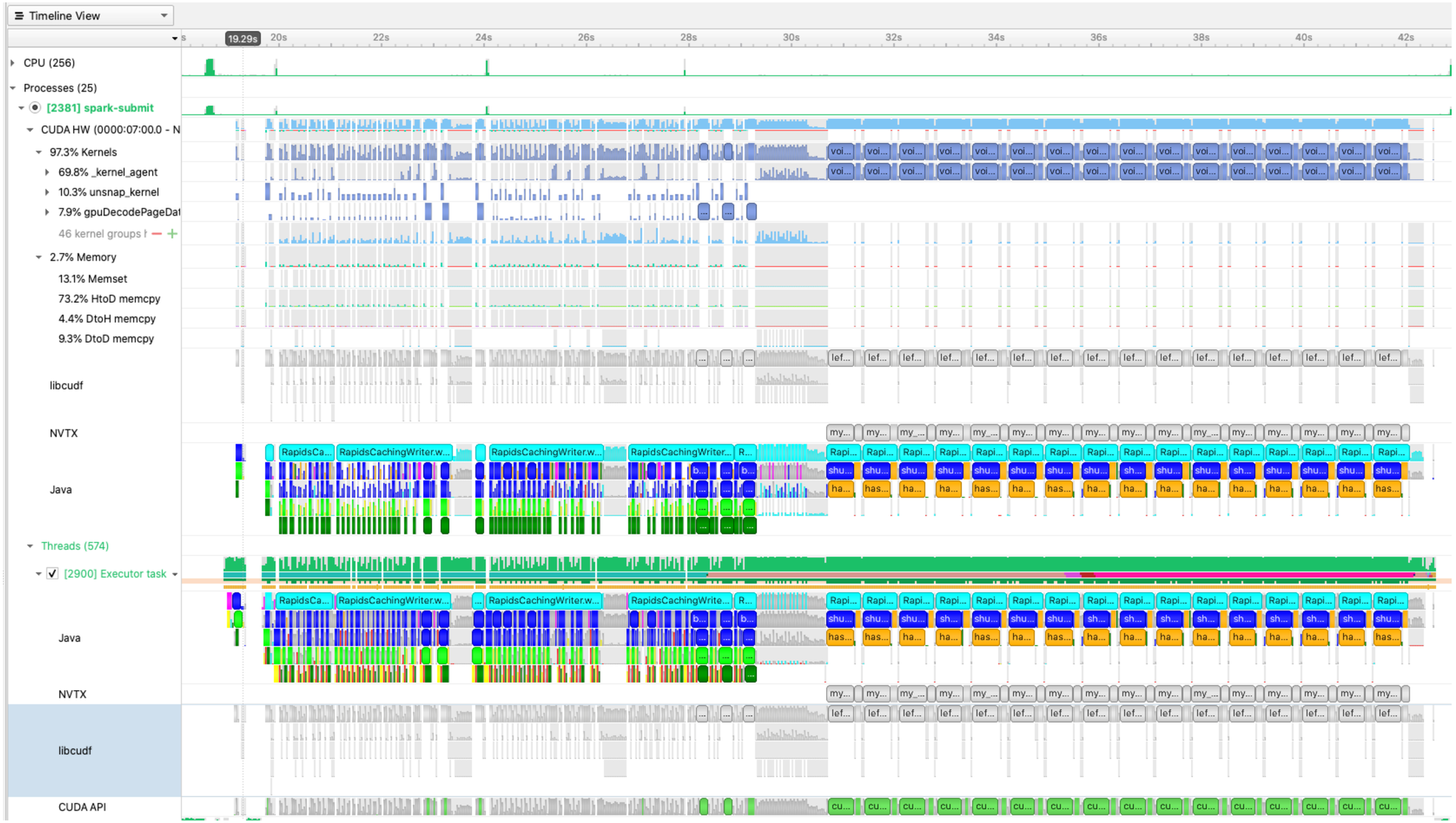
Running [/opt/nvidia/nsight-systems/2021.5.1/target-linux-x64/reports/gpukernsum.py q38_STABLE_none_thrust_flagged.sqlite]...

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
25.1	1,794,781,687	68	26,393,848.3	22,942,445.5	21,832,010	62,763,398	7,[illegible]5.3	void snappy::unsnap_kernel(const void *const *, const unsigned long *, void *const *, const unsigned lon…
19.2	1,372,425,557	68	20,182,728.8	9,045,509.5	2,833,783	175,824,084	38,380,711.6	gpuDecodePageData
9.7	695,997,164	118	5,898,281.1	396,854.0	22,944	41,137,654	10,842,102.5	void thrust::cuda_cub::core::_kernel_agent<thrust::cuda_cub::__parallel_for::ParallelForAgent<thrus…
6.4	455,679,966	98	4,649,795.6	4,810,227.0	1,332,670	9,623,687	2,337,559.2	void cuco::detail::pair_count<(unsigned int)128, (unsigned int)2, (bool)0, thrust::transform_iterat…
5.9	422,018,594	32	13,188,081.1	13,146,538.5	5,039,741	21,400,821	8,113,171.8	void thrust::cuda_cub::core::_kernel_agent<thrust::cuda_cub::__parallel_for::ParallelForAgent<thrus…
5.8	411,338,721	136	3,024,549.4	3,011,698.0	13,759	3,269,931	520,273.1	gpuDecodePageHeaders
4.3	306,115,435	65	4,709,468.2	5,358,037.0	1,580,087	12,099,591	2,781,472.7	void cuco::detail::pair_retrieve<(unsigned int)128, (unsigned int)32, (unsigned int)2, (unsigned in…
3.0	211,903,501	160	1,324,396.9	832,666.0	313,751	3,147,119	977,844.1	void cudf::strings::detail::gather_chars_fn_char_parallel<(int)32, thrust::transform_iterator<cudf:…
2.7	190,685,801	365	522,426.9	134,813.0	4,928	3,307,146	764,936.8	void cudf::<unnamed>::copy_partitions<(int)256>(const unsigned char **, unsigned char **, cudf::<un…
2.0	142,303,707	134	1,061,968.0	978,038.5	7,360	1,710,867	245,072.2	void cudf::detail::scatter_kernel<long, <unnamed>::boolean_mask_filter<(bool)0>, (int)256, (bool)1>…
1.6	111,595,352	214	521,473.6	141,628.0	27,904	3,231,980	872,577.8	void thrust::cuda_cub::core::_kernel_agent<thrust::cuda_cub::__parallel_for::ParallelForAgent<thrus…
1.4	102,050,971	194	526,035.9	94,237.5	8,896	2,881,686	1,023,355.2	void thrust::cuda_cub::core::_kernel_agent<thrust::cuda_cub::__parallel_for::ParallelForAgent<thrus…
1.2	83,996,258	194	432,970.4	110,525.5	9,279	2,247,046	793,430.0	void thrust::cuda_cub::core::_kernel_agent<thrust::cuda_cub::__parallel_for::ParallelForAgent<thrus…
1.1	80,745,583	224	360,471.4	63,262.0	28,416	1,090,404	441,796.1	void thrust::cuda_cub::core::_kernel_agent<thrust::cuda_cub::__parallel_for::ParallelForAgent<thrus…
0.9	62,816,704	159	395,073.6	363,030.0	86,749	1,119,299	281,481.8	void thrust::cuda_cub::core::_kernel_agent<thrust::cuda_cub::__parallel_for::ParallelForAgent<thrus…

Hot Spot

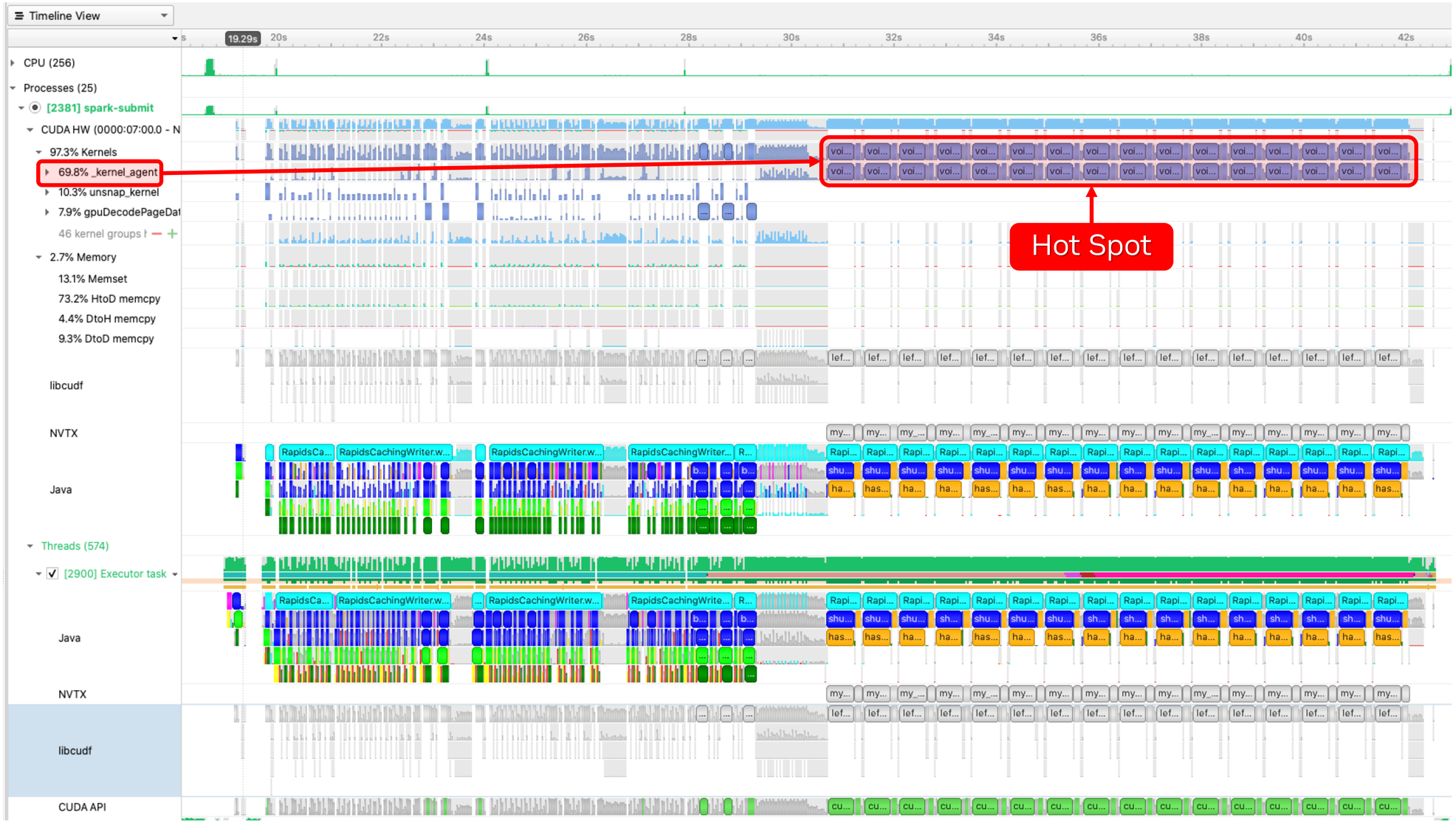
IDENTIFY HOTSPOT

Example #1



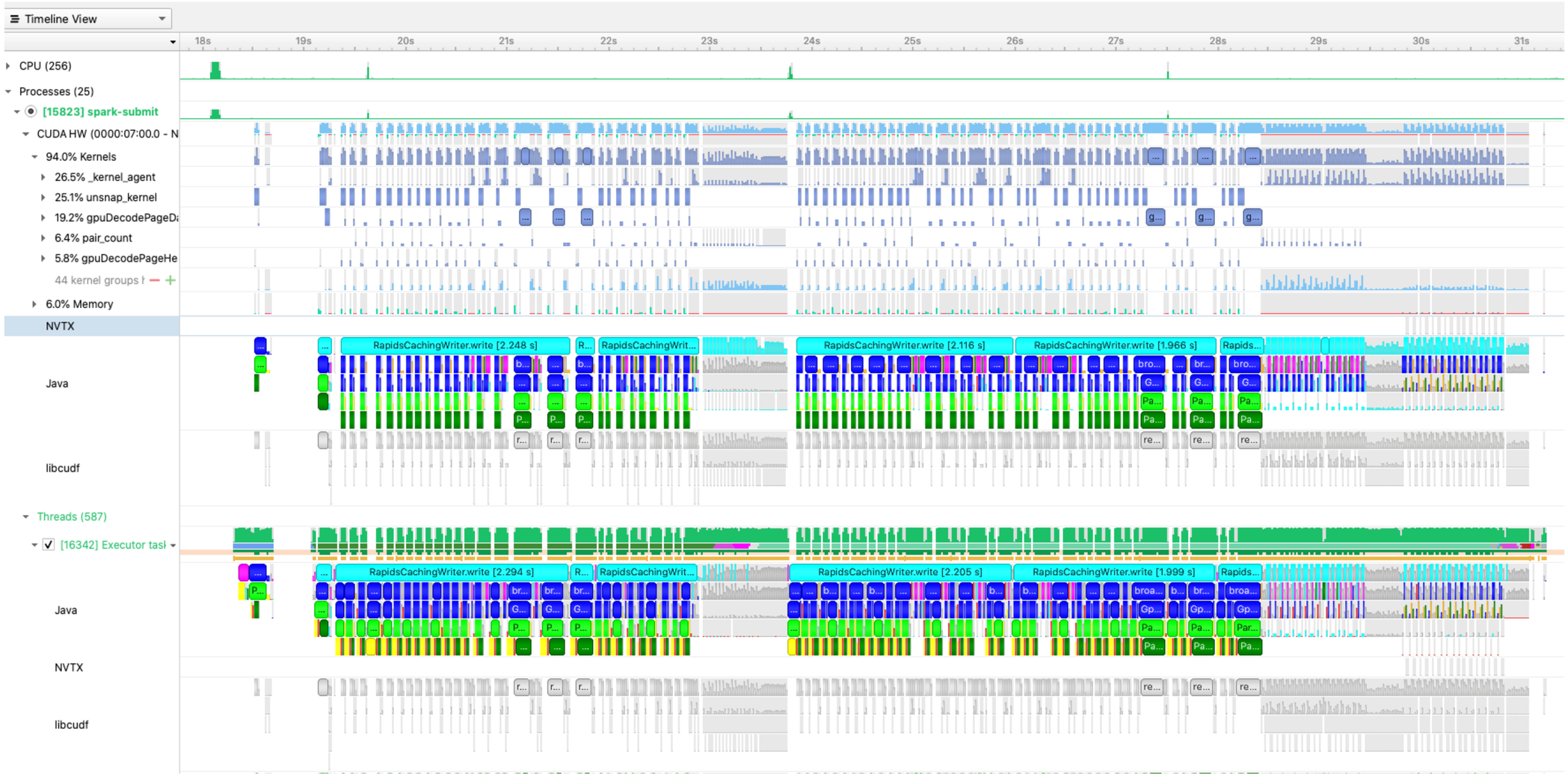
IDENTIFY HOTSPOT

Example #1



IDENTIFY HOTSPOT

Example 2



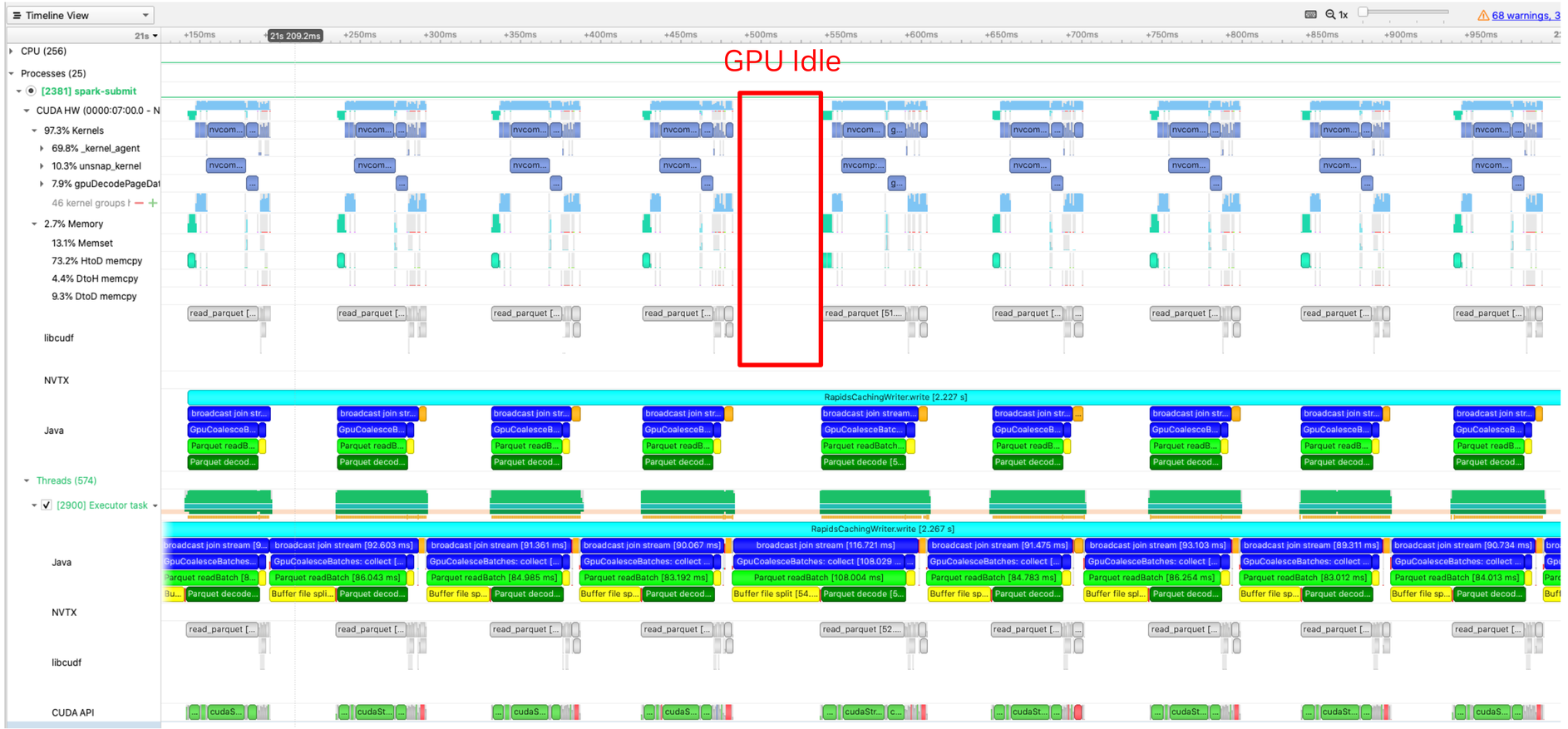
IDENTIFY HOTSPOT

Example 2

IDENTIFY HOTSPOT

Example 2

Problem: GPU starvation is caused by waiting for data to be read from disk



LATENCY BOUND

Solution: Hide Latency with more parallel work

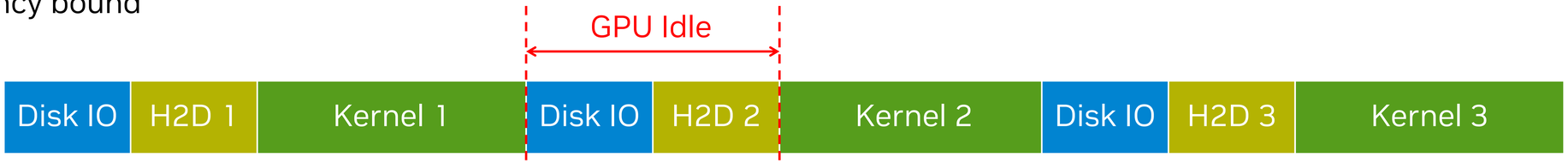
- For each table partition, we need to read it from disk, copy it to GPU, and compute on it.

LATENCY BOUND

Solution: Hide Latency with more parallel work

- For each table partition, we need to read it from disk, copy it to GPU, and compute on it.

- Serial – latency bound



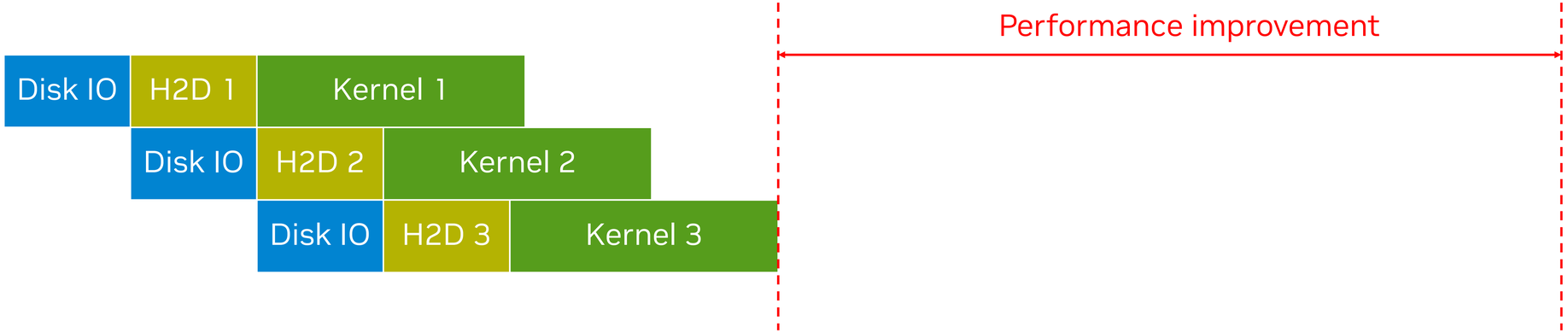
LATENCY BOUND

Solution: Hide Latency with more parallel work

- For each table partition, we need to read it from disk, copy it to GPU, and compute on it.

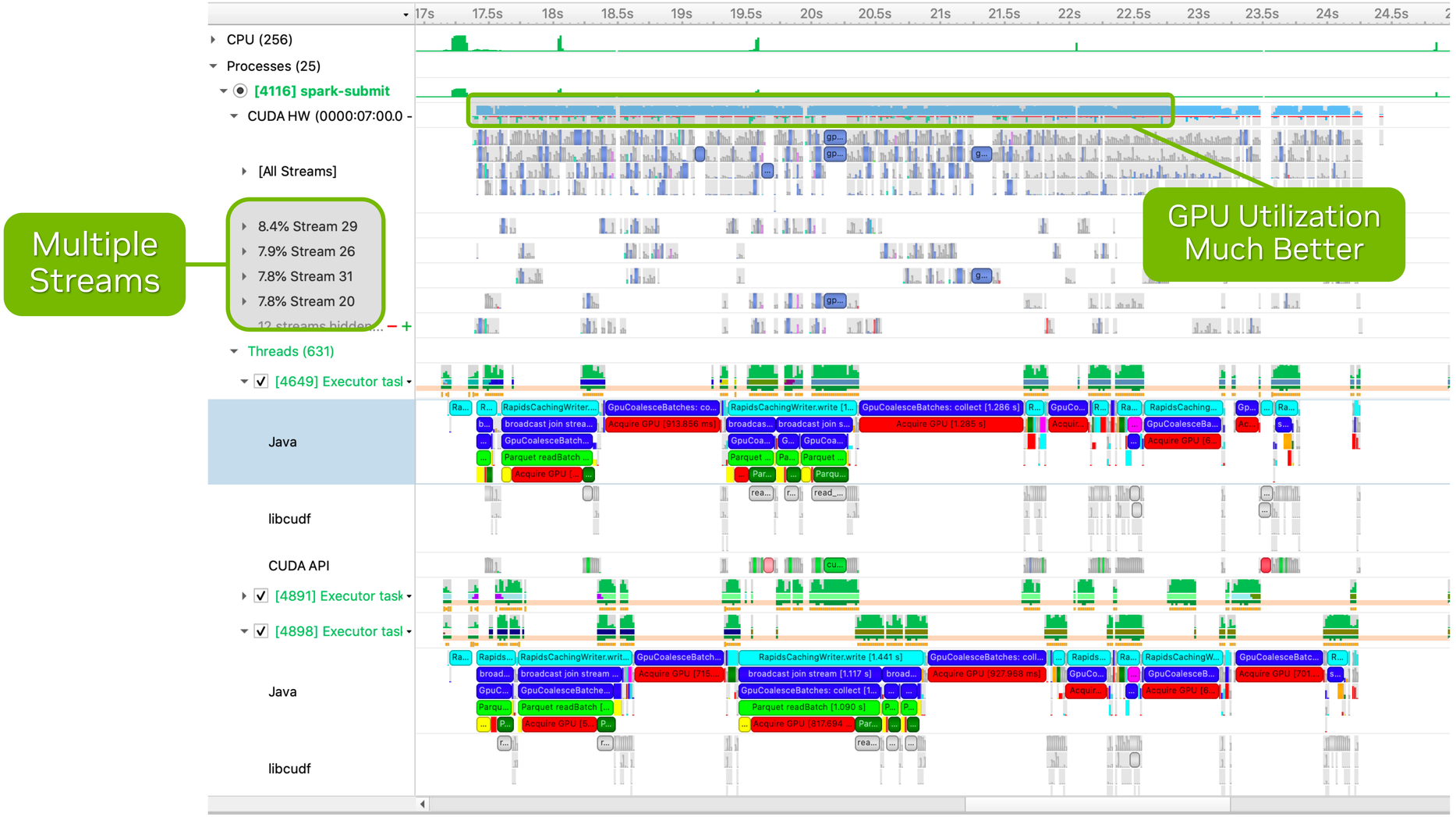
- Serial – latency bound

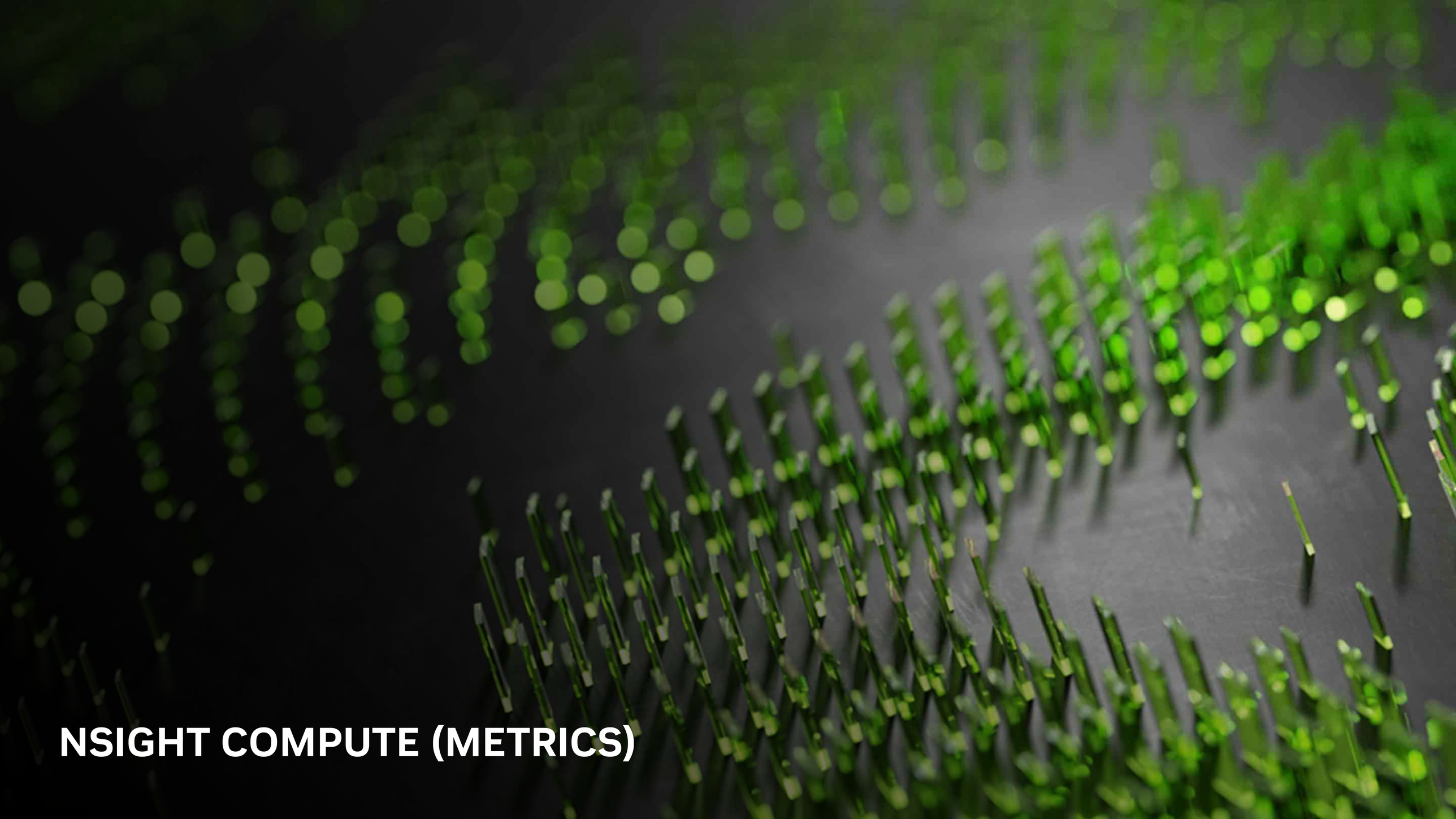
- Concurrent (CUDA Streams) – Hide cost of data transfers



CUDA STREAMS

GPU Fully Utilized





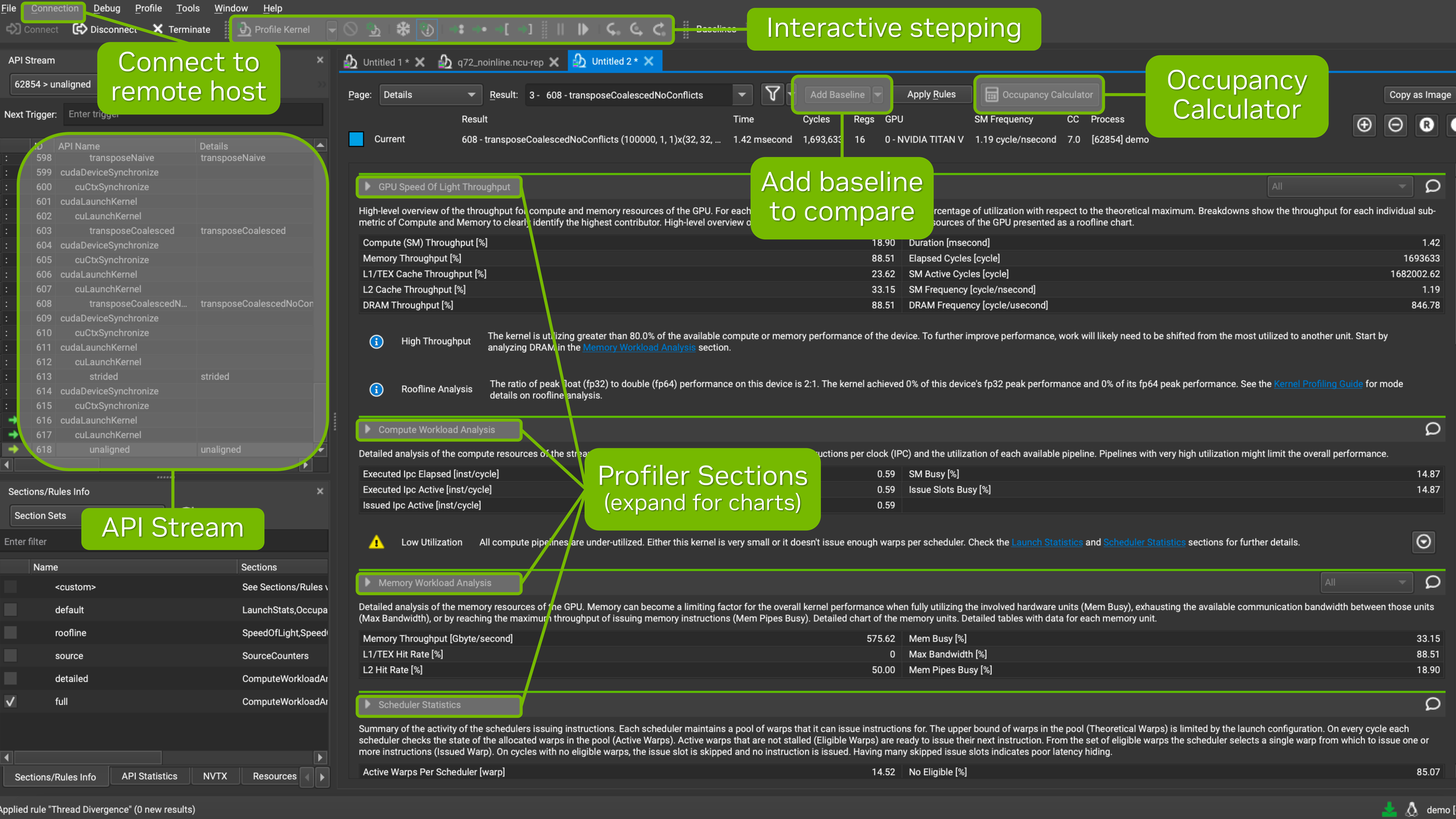
NSIGHT COMPUTE (METRICS)

NSIGHT COMPUTE PROFILE

Profile with CLI example

```
$ ncu -o kernel_prof \          ← Output filename
      -f \                      ← Overwrite file if exists
      -k my_kernel \            ← kernel name to profile
      -c 1 \                    ← Launch count
      -s 100 \                  ← Launches to skip which match kernel filter (name)
      --set full \              ← Collect metrics for all sections
      python my_app.py          ← Execution command
```

https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html



File Connection Debug Profile Tools Window Help

Connect Disconnect Terminate Profile Kernel Baselines

API Stream

62854 > unaligned

Next Trigger: Enter trigger

ID	API Name	Details
598	transposeNaive	transposeNaive
599	cudaDeviceSynchronize	
600	cuCtxSynchronize	
601	cudaLaunchKernel	
602	cuLaunchKernel	
603	transposeCoalesced	transposeCoalesced
604	cudaDeviceSynchronize	
605	cuCtxSynchronize	
606	cudaLaunchKernel	
607	cuLaunchKernel	
608	transposeCoalescedN...	transposeCoalescedNoCon
609	cudaDeviceSynchronize	
610	cuCtxSynchronize	
611	cudaLaunchKernel	
612	cuLaunchKernel	
613	strided	strided
614	cudaDeviceSynchronize	
615	cuCtxSynchronize	
616	cudaLaunchKernel	
617	cuLaunchKernel	
618	unaligned	unaligned

Sections/Rules Info

Section Sets

Enter filter

Name	Sections
<custom>	See Sections/Rules v
default	LaunchStats,Occupa
roofline	SpeedOfLight,Speed
source	SourceCounters
detailed	ComputeWorkloadAr
full	ComputeWorkloadAr

Sections/Rules Info | API Statistics | NVTX | Resources

Untitled 1 * | q72_noinline.ncu-rep | Untitled 2 *

Page: Details Result: 3 - 608 - transposeCoalescedNoConflicts Add Baseline Apply Rules Occupancy Calculator Copy as Image

	Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	608 - transposeCoalescedNoConflicts (100000, 1, 1)x(32, 32, ...	1.42 msecond	1,693,633	16	0 - NVIDIA TITAN V	1.19 cycle/nsecond	7.0	[62854] demo

GPU Speed Of Light Throughput

High-level overview of the throughput for compute and memory resources of the GPU. For each ...rcentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor. High-level overview o... ...sources of the GPU presented as a roofline chart.

Compute (SM) Throughput [%]	18.90	Duration [msecond]	1.42
Memory Throughput [%]	88.51	Elapsed Cycles [cycle]	1693633
L1/TEX Cache Throughput [%]	23.62	SM Active Cycles [cycle]	1682002.62
L2 Cache Throughput [%]	33.15	SM Frequency [cycle/nsecond]	1.19
DRAM Throughput [%]	88.51	DRAM Frequency [cycle/usecond]	846.78

High Throughput: The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing DRAM in the Memory Workload Analysis section.

Roofline Analysis: The ratio of peak float (fp32) to double (fp64) performance on this device is 2:1. The kernel achieved 0% of this device's fp32 peak performance and 0% of its fp64 peak performance. See the Kernel Profiling Guide for mode details on roofline analysis.

Compute Workload Analysis

Detailed analysis of the compute resources of the stre... ...uctions per clock (IPC) and the utilization of each available pipeline. Pipelines with very high utilization might limit the overall performance.

Executed Ipc Elapsed [inst/cycle]	0.59	SM Busy [%]	14.87
Executed Ipc Active [inst/cycle]	0.59	Issue Slots Busy [%]	14.87
Issued Ipc Active [inst/cycle]	0.59		

Low Utilization: All compute pipelines are under-utilized. Either this kernel is very small or it doesn't issue enough warps per scheduler. Check the Launch Statistics and Scheduler Statistics sections for further details.

Memory Workload Analysis

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy). Detailed chart of the memory units. Detailed tables with data for each memory unit.

Memory Throughput [Gbyte/second]	575.62	Mem Busy [%]	33.15
L1/TEX Hit Rate [%]	0	Max Bandwidth [%]	88.51
L2 Hit Rate [%]	50.00	Mem Pipes Busy [%]	18.90

Scheduler Statistics

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

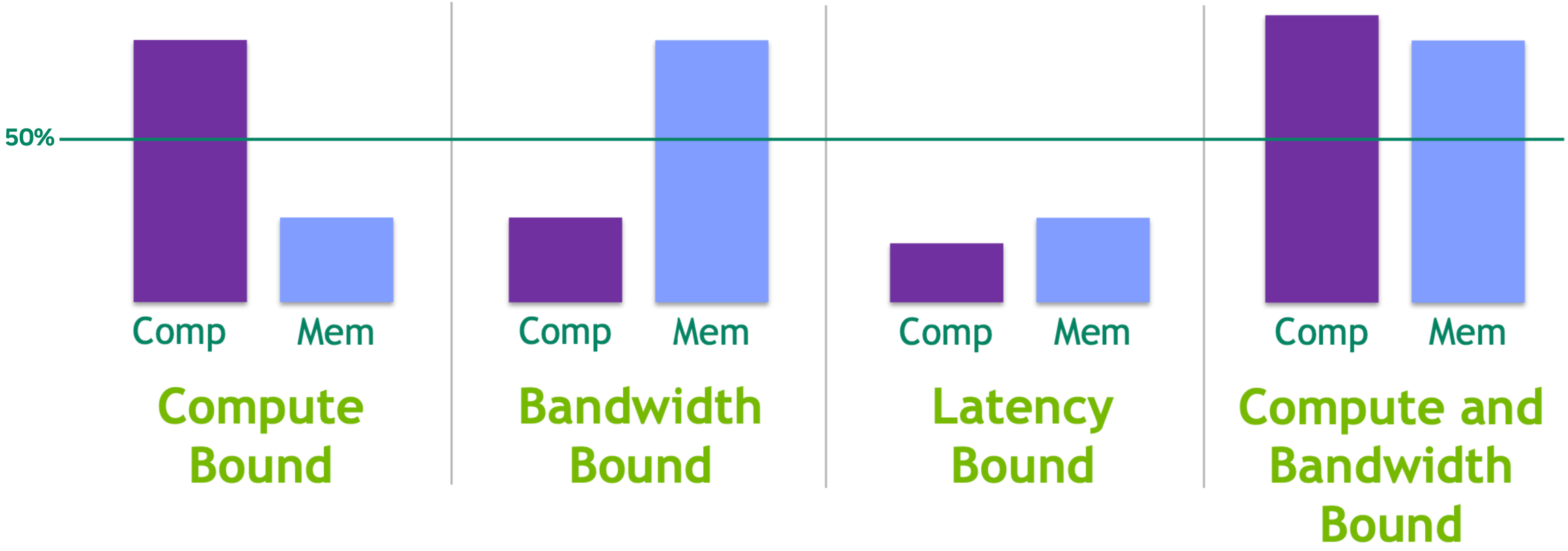
Active Warps Per Scheduler [warp]	14.52	No Eligible [%]	85.07

Applied rule "Thread Divergence" (0 new results)

demo [

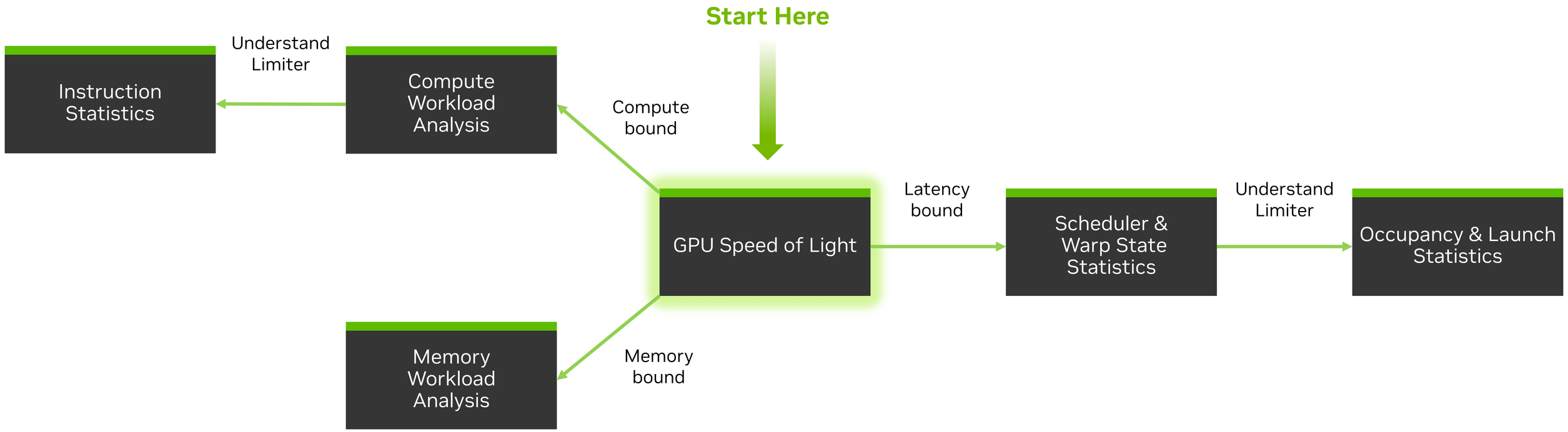
GPU SPEED OF LIGHT THROUGHPUT

Performance Limiter Categories



NSIGHT COMPUTE

Analysis Workflow



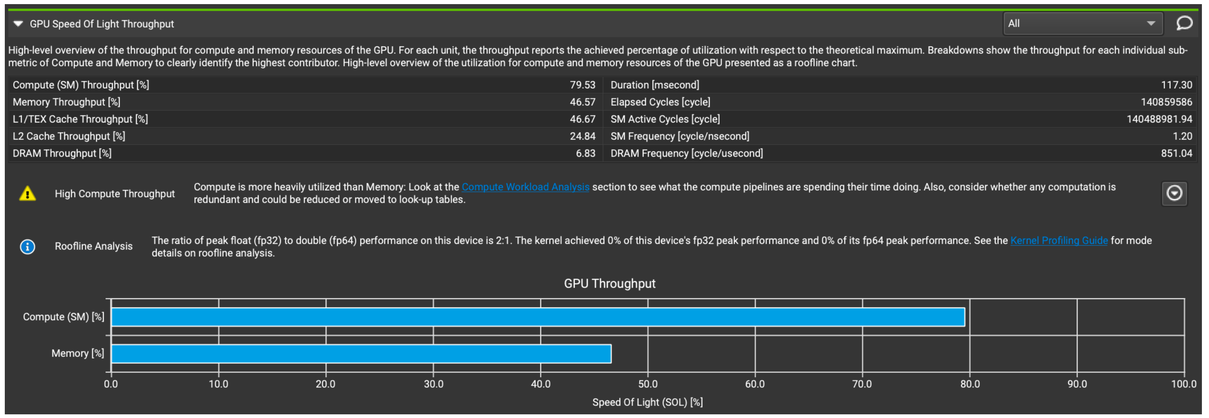
SOL SECTION

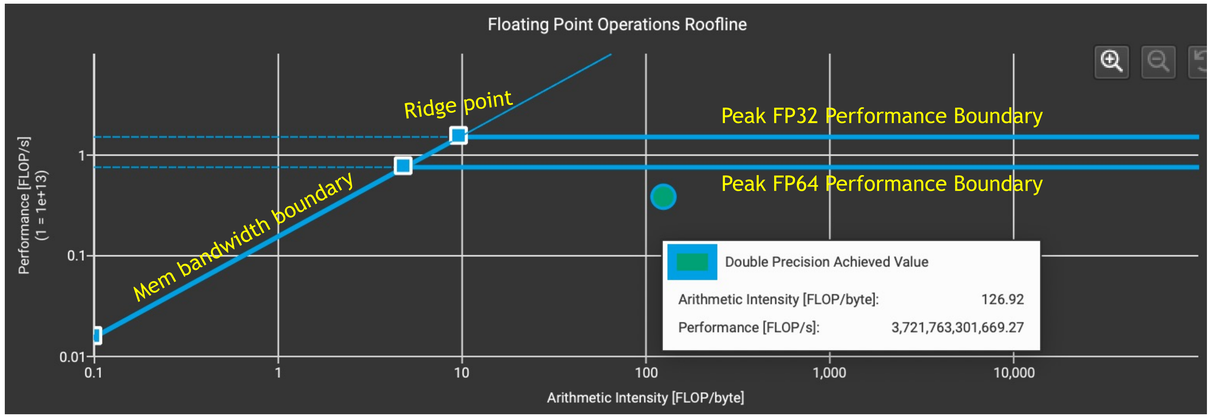
Start here

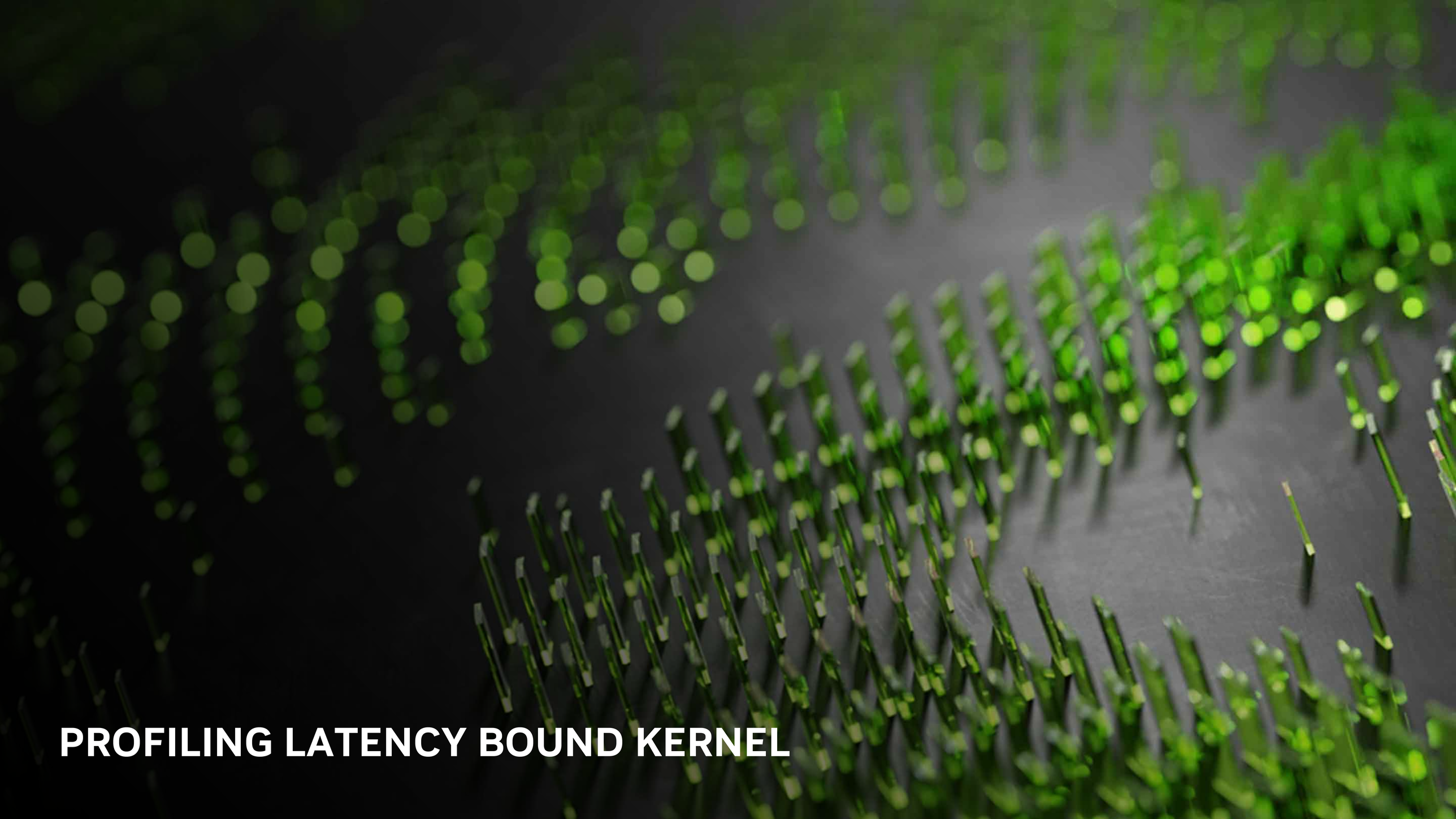
- **Speed Of Light (SOL)** chart gives you high-level overview of compute and memory utilization of the GPU. For each unit, it reports the achieved % of utilization with respect to its theoretical maximum.

- **Roofline** chart shows your achieved FLOPs with respect to the theoretical maximum single-precision or double-precision FLOPS.

Arithmetic Intensity = Compute/Memory
= FLOPS/byte



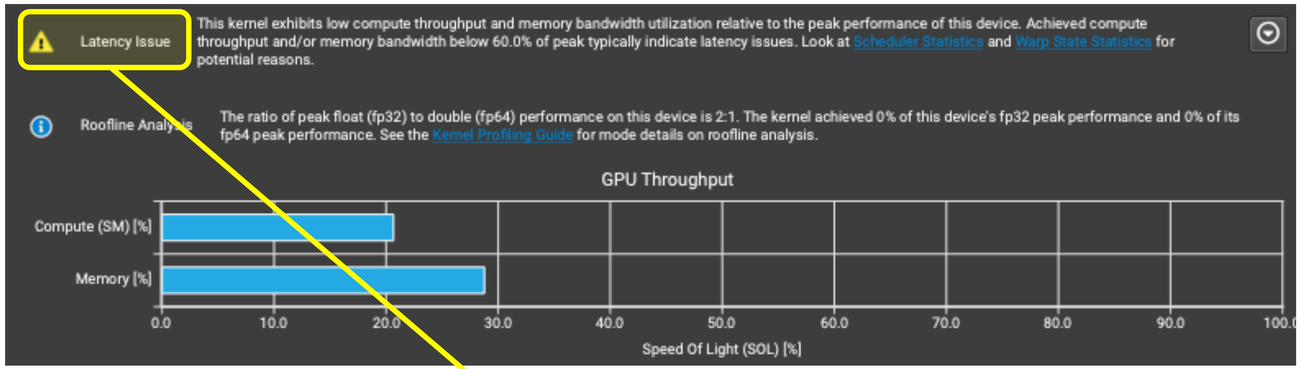




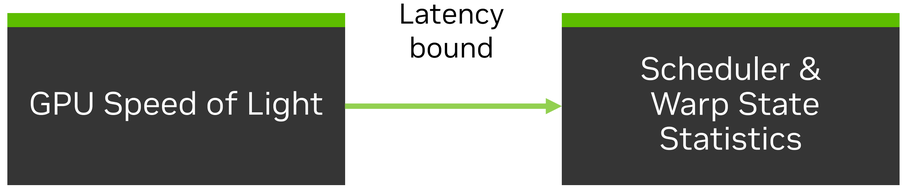
PROFILING LATENCY BOUND KERNEL

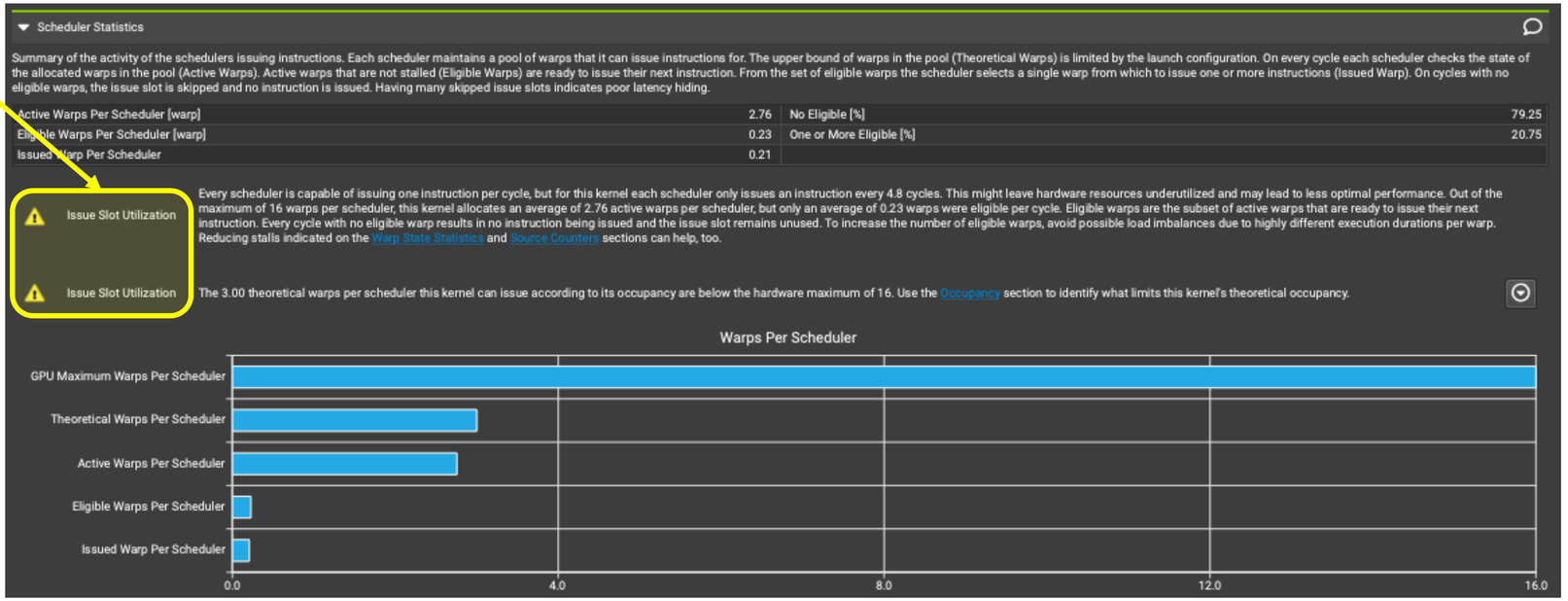
LATENCY BOUND KERNELS

Warp Scheduler Statistics



Start Here





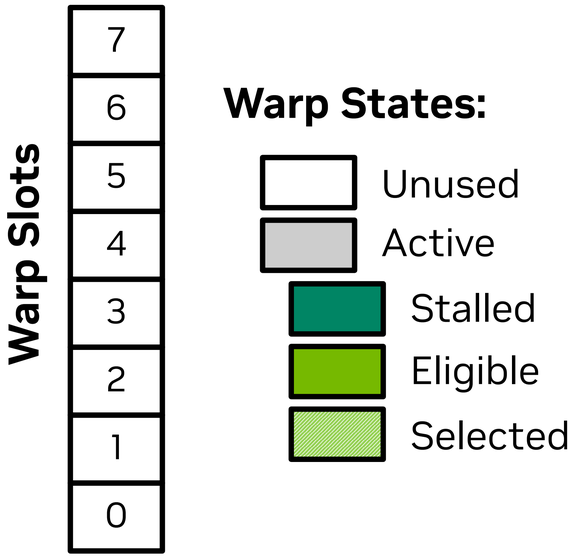
WARP SCHEDULER STATISTICS

Ampere Architecture

- 4 warp schedulers per SM
- Each scheduler managers a pool of warps:
  - Ampere: 16 warp slots
  - Volta: 16 warp slots
  - Turing: 8 warp slots
- Each scheduler can issue 1 instruction/warp/cycle
  - *Exception: can issue 2 instructions/warp/cycle if instructions independent and going to different functional units.
- Scheduler issues instructions to pipelines

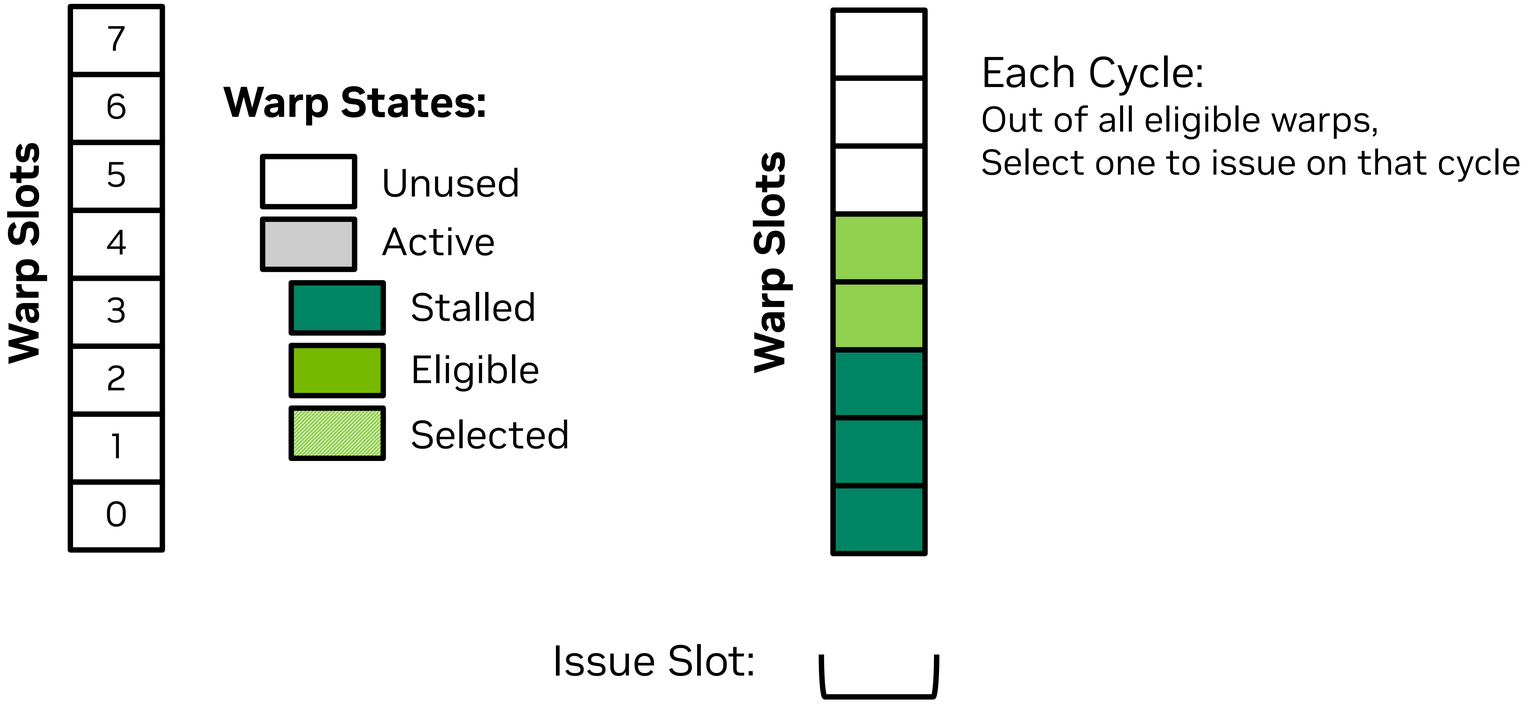
WARP SCHEDULER STATISTICS

Mental Model for Profiling



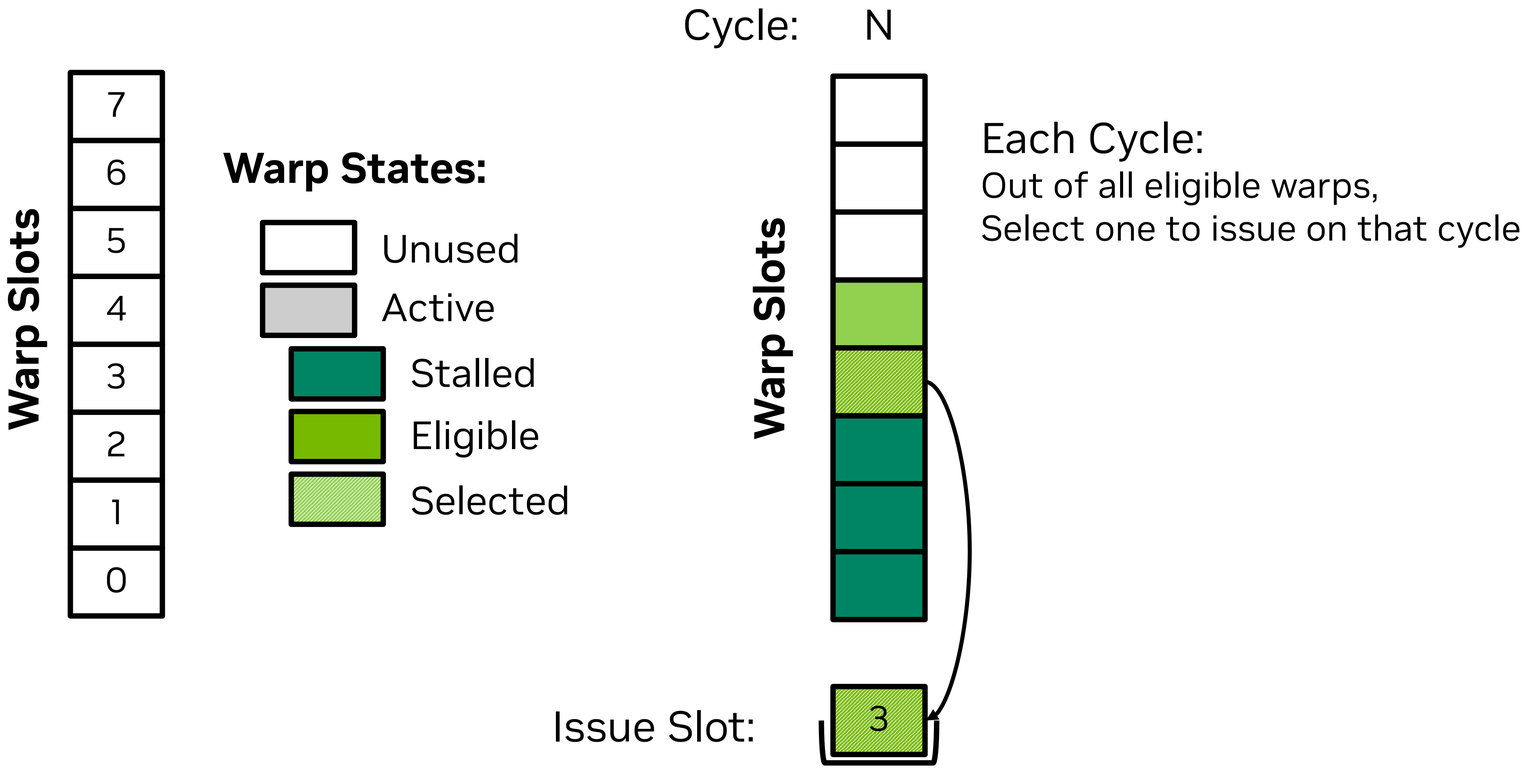
WARP SCHEDULER STATISTICS

Mental Model for Profiling



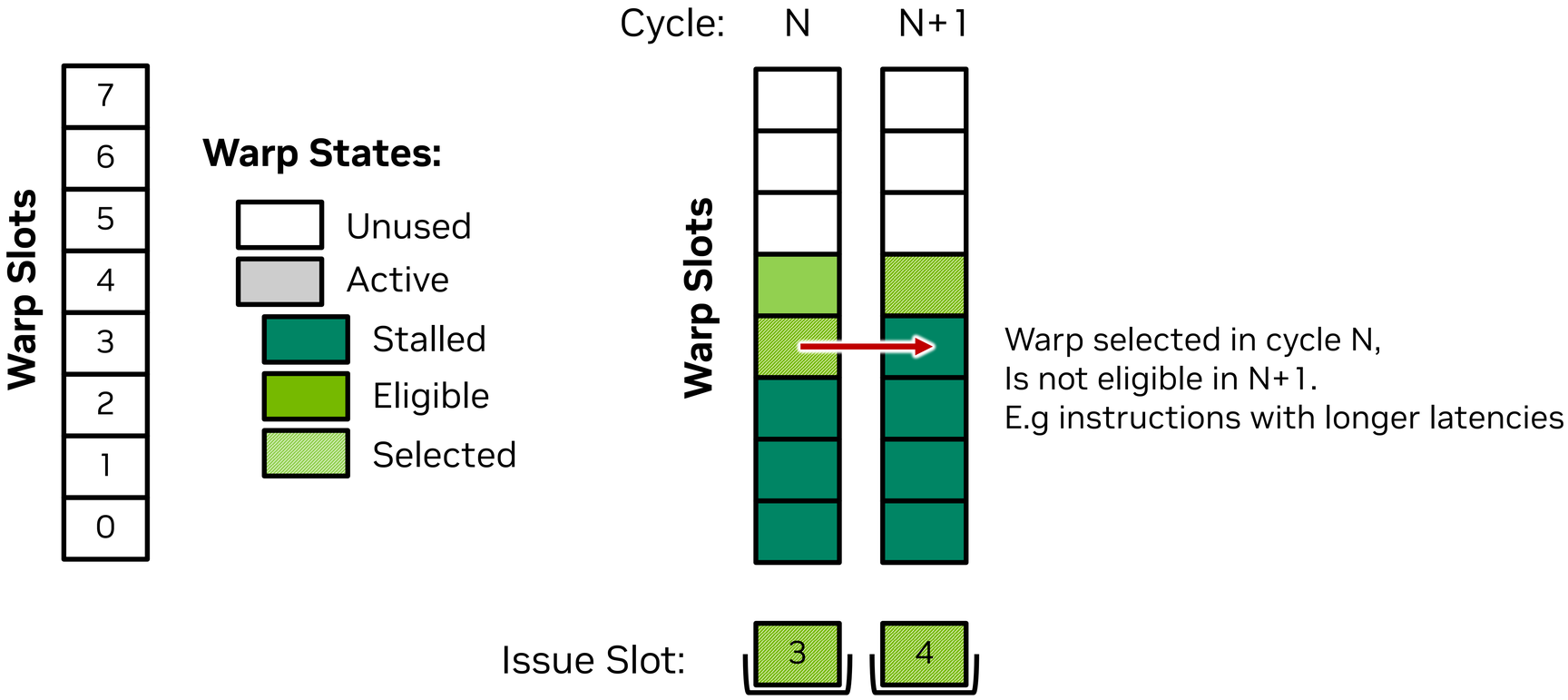
WARP SCHEDULER STATISTICS

Mental Model for Profiling



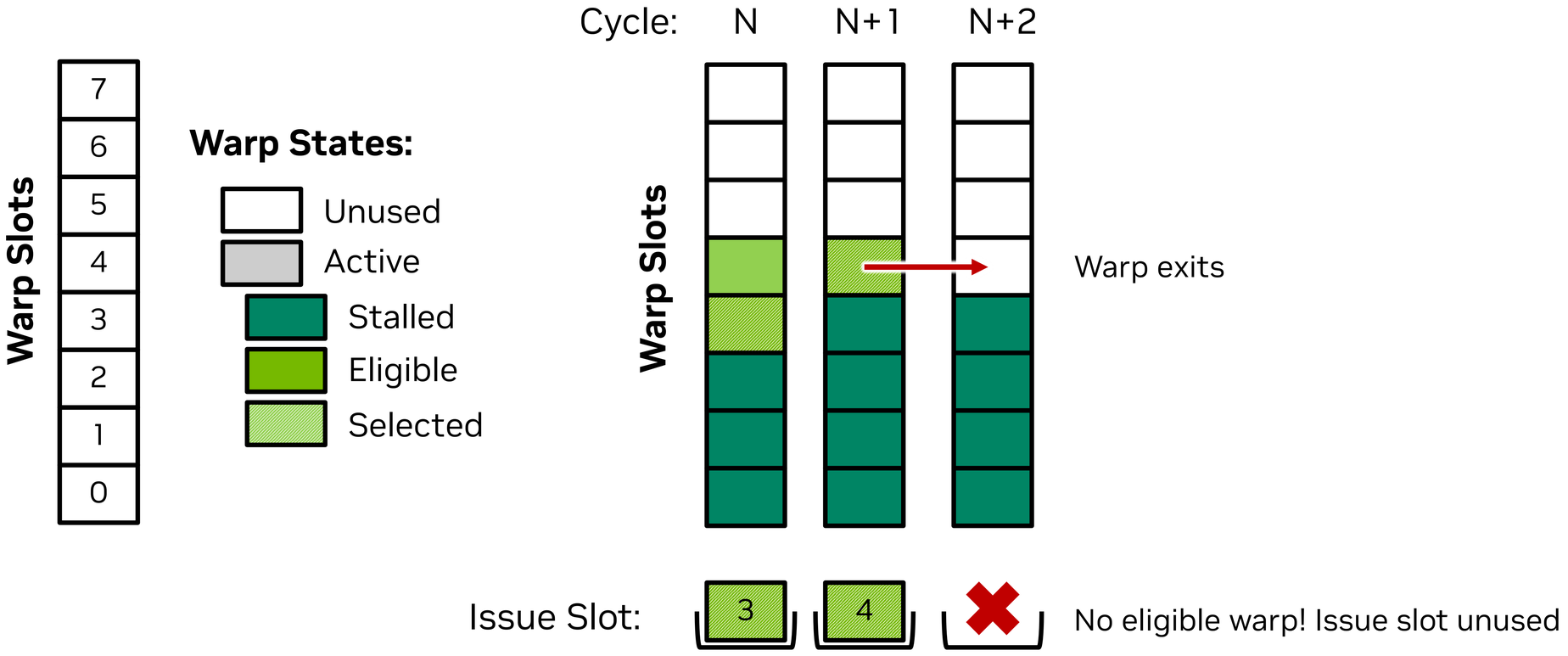
WARP SCHEDULER STATISTICS

Mental Model for Profiling



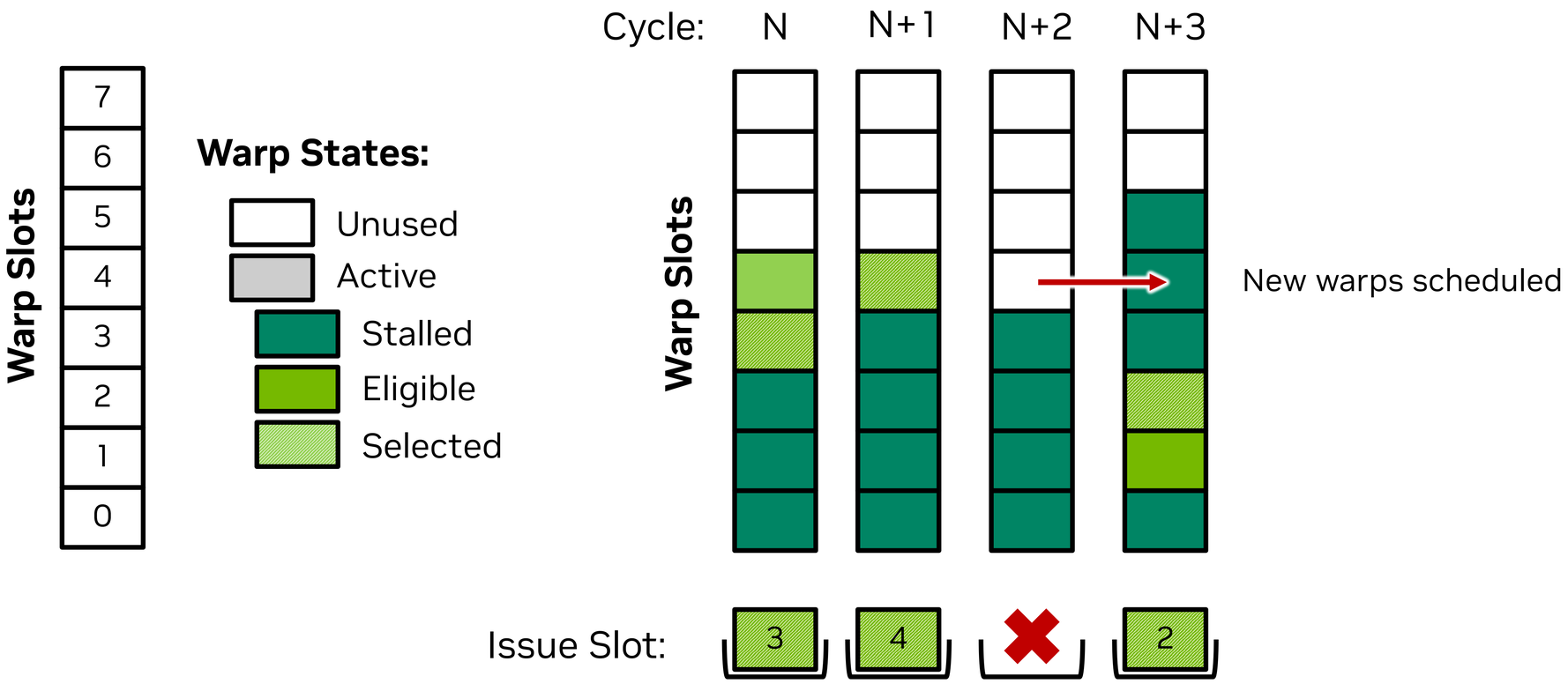
WARP SCHEDULER STATISTICS

Mental Model for Profiling



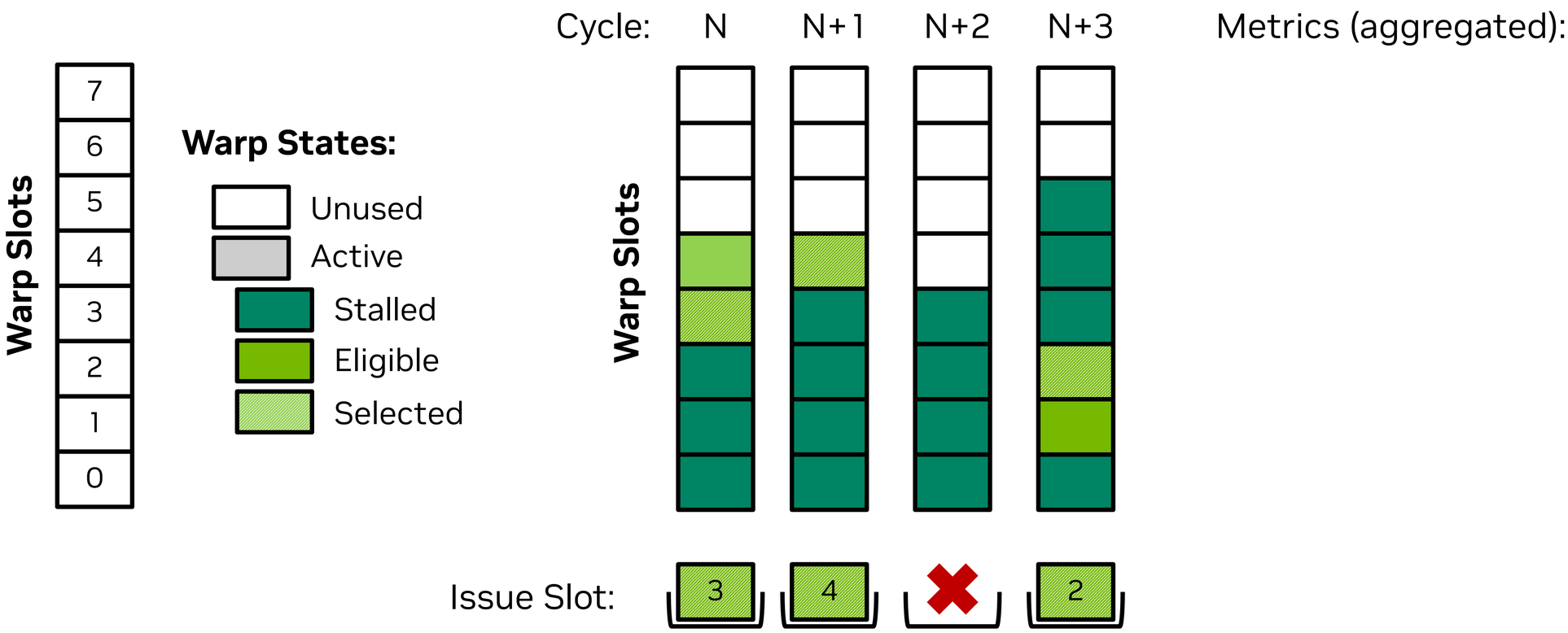
WARP SCHEDULER STATISTICS

Mental Model for Profiling



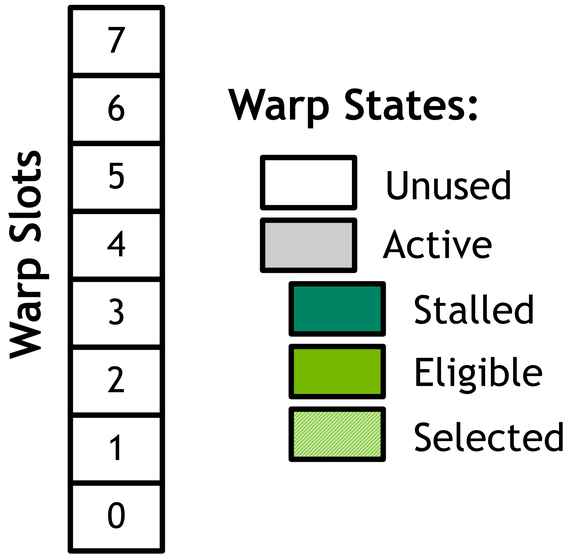
WARP SCHEDULER STATISTICS

Mental Model for Profiling



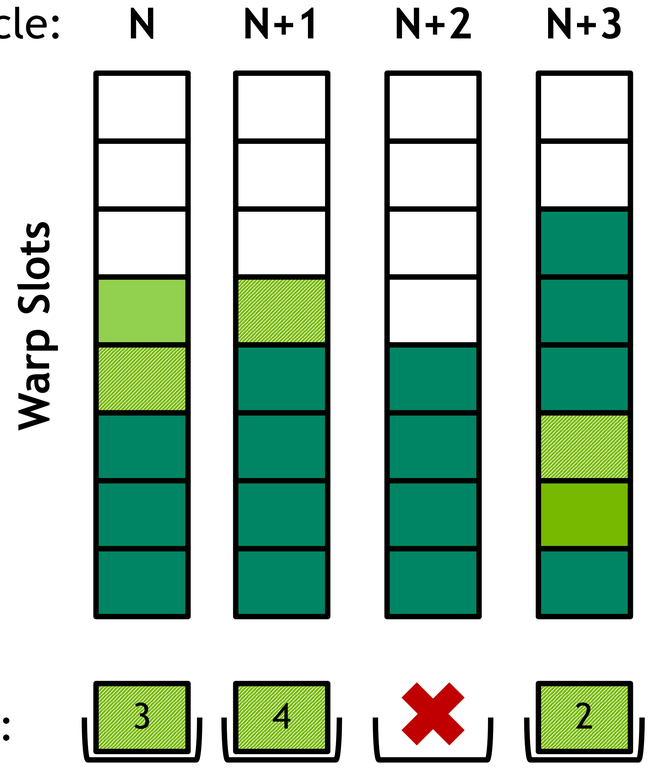
WARP SCHEDULER STATISTICS

Mental Model for Profiling



Warp States:

- Unused
- Active
- Stalled
- Eligible
- Selected

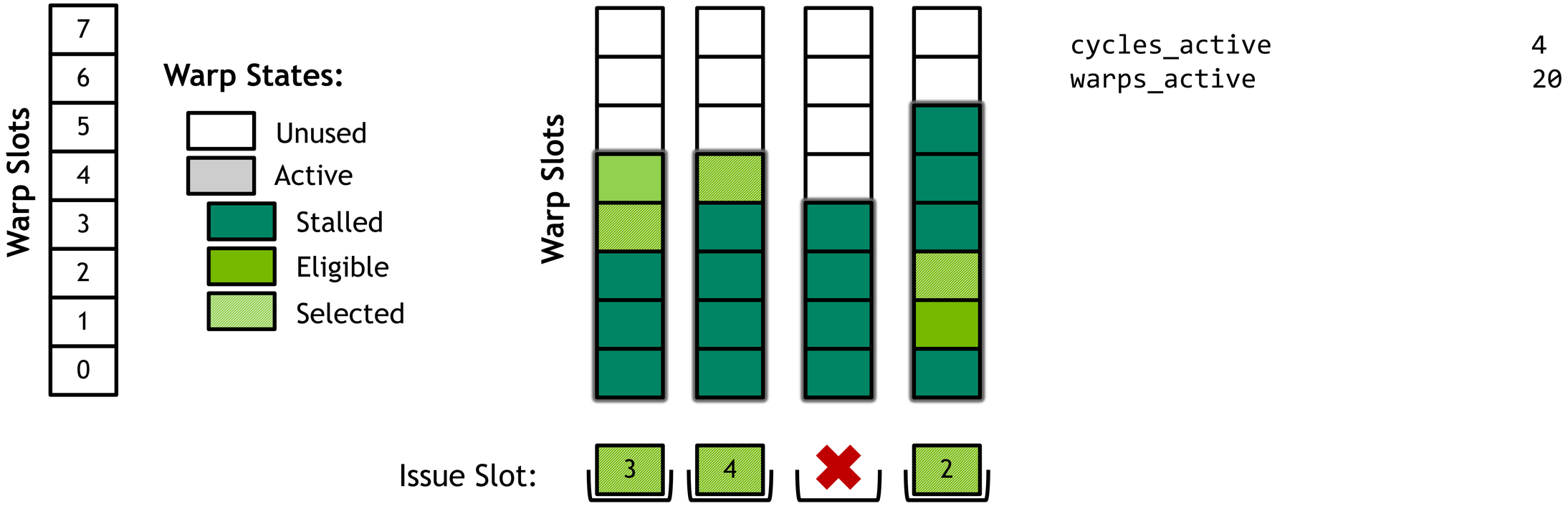


Metrics (aggregated):

`cycles_active`	4

WARP SCHEDULER STATISTICS

Mental Model for Profiling

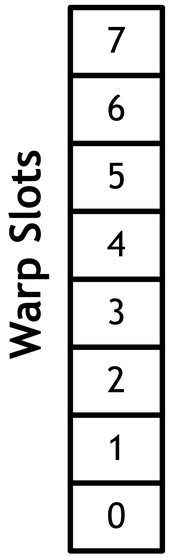


Metrics (aggregated):

`cycles_active`	4
`warps_active`	20

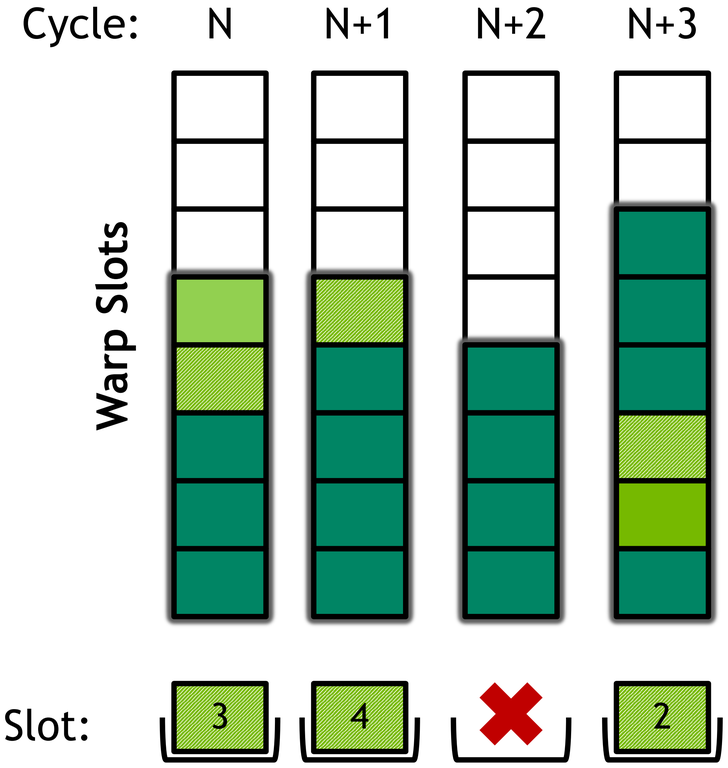
WARP SCHEDULER STATISTICS

Mental Model for Profiling



Warp States:

- Unused
- Active
- Stalled
- Eligible
- Selected

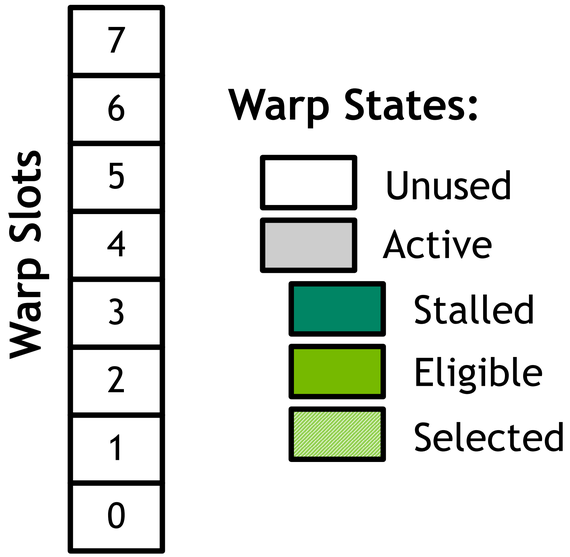


Metrics (aggregated):

`cycles_active`	`4`
`warps_active`	`20`
`achieved_occupancy`	`62.5%`

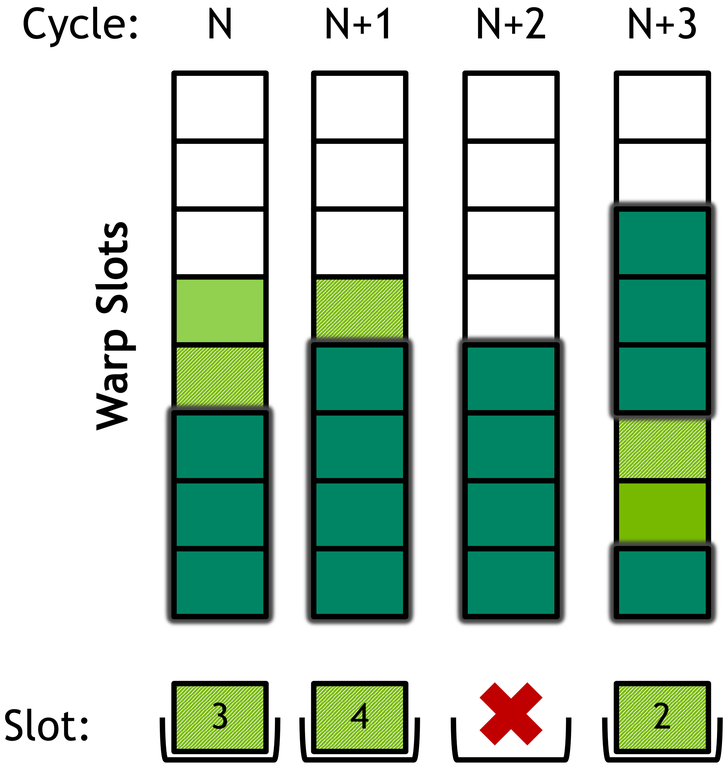
WARP SCHEDULER STATISTICS

Mental Model for Profiling



Warp States:

- Unused
- Active
- Stalled
- Eligible
- Selected

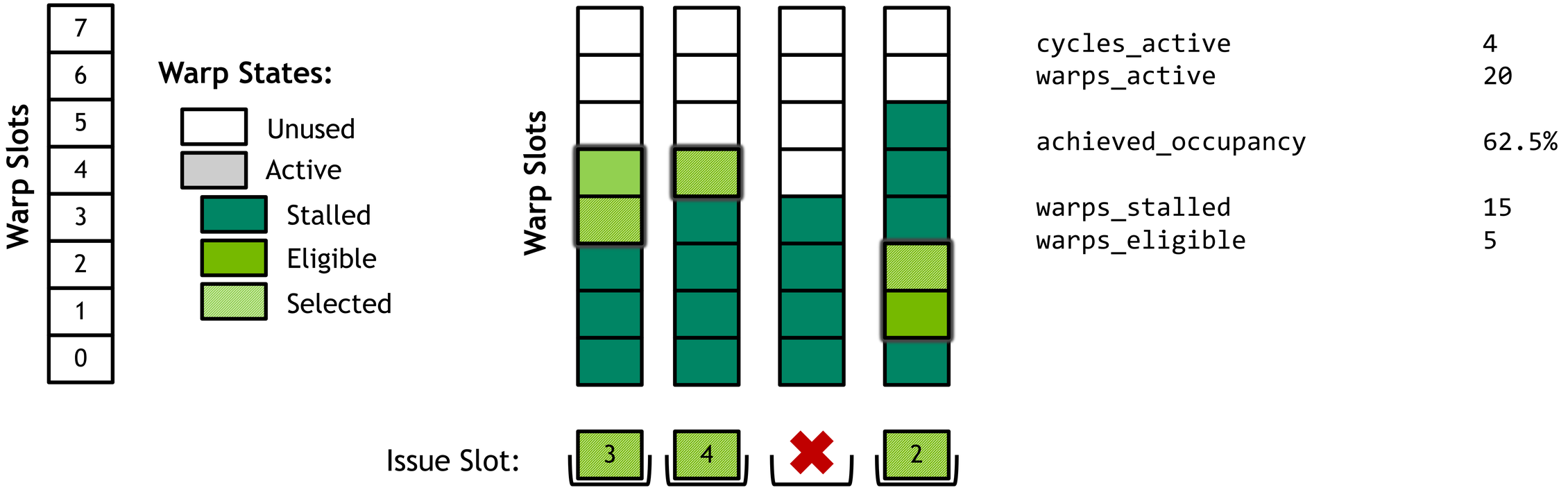


Metrics (aggregated):

cycles_active	4
warps_active	20
achieved_occupancy	62.5%
warps_stalled	15

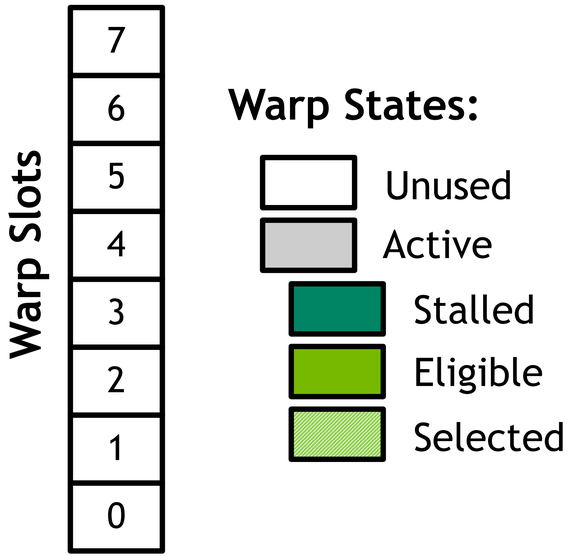
WARP SCHEDULER STATISTICS

Mental Model for Profiling



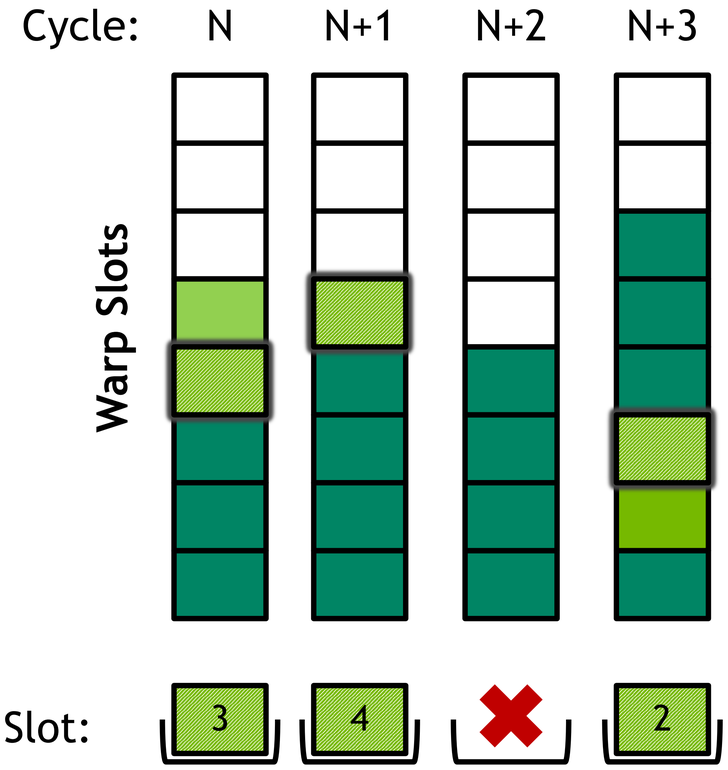
WARP SCHEDULER STATISTICS

Mental Model for Profiling



Warp States:

- Unused
- Active
- Stalled
- Eligible
- Selected

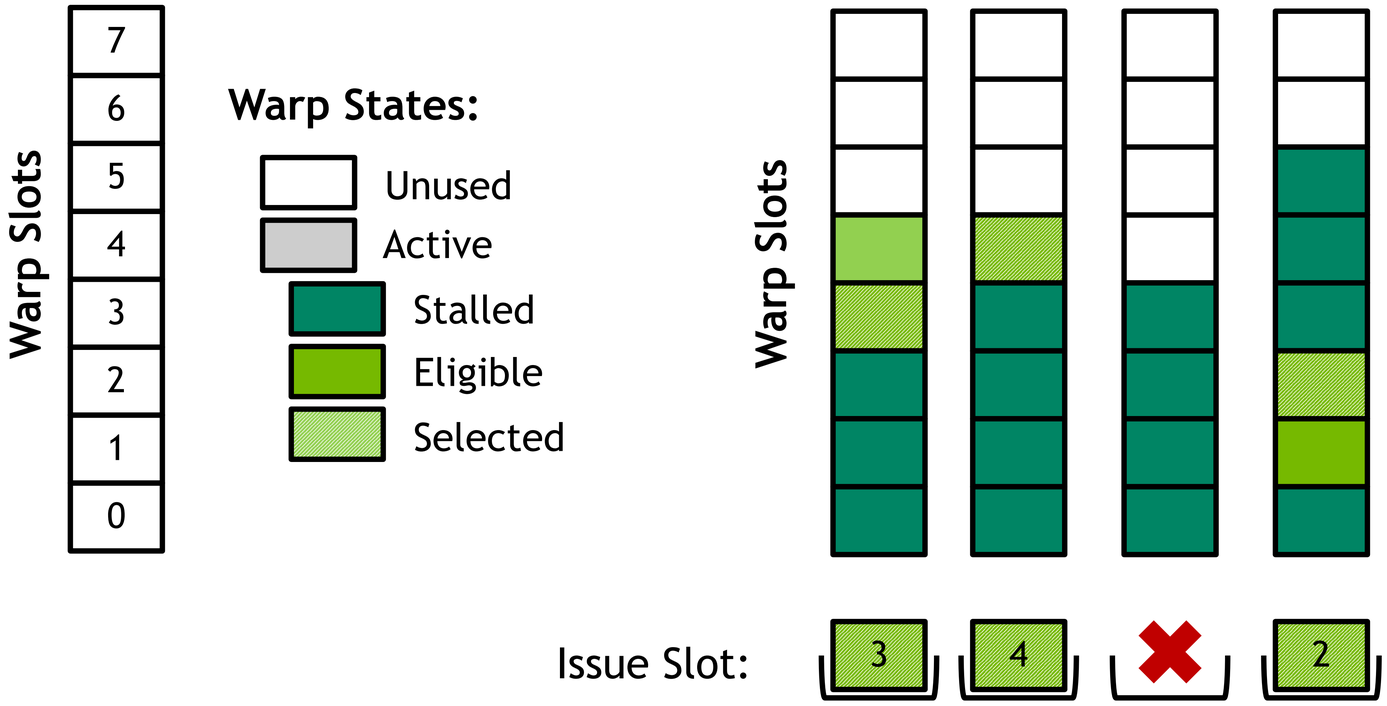


Metrics (aggregated):

cycles_active	4
warps_active	20
achieved_occupancy	62.5%
warps_stalled	15
warps_eligible	5
warps_issued	3

WARP SCHEDULER STATISTICS

Mental Model for Profiling



Metrics (aggregated):

cycles_active	4
warps_active	20
achieved_occupancy	62.5%
warps_stalled	15
warps_eligible	5
warps_issued	3
warps_issued/cycles_active	0.75
issue_slot_utilization	75%

LATENCY BOUND KERNEL

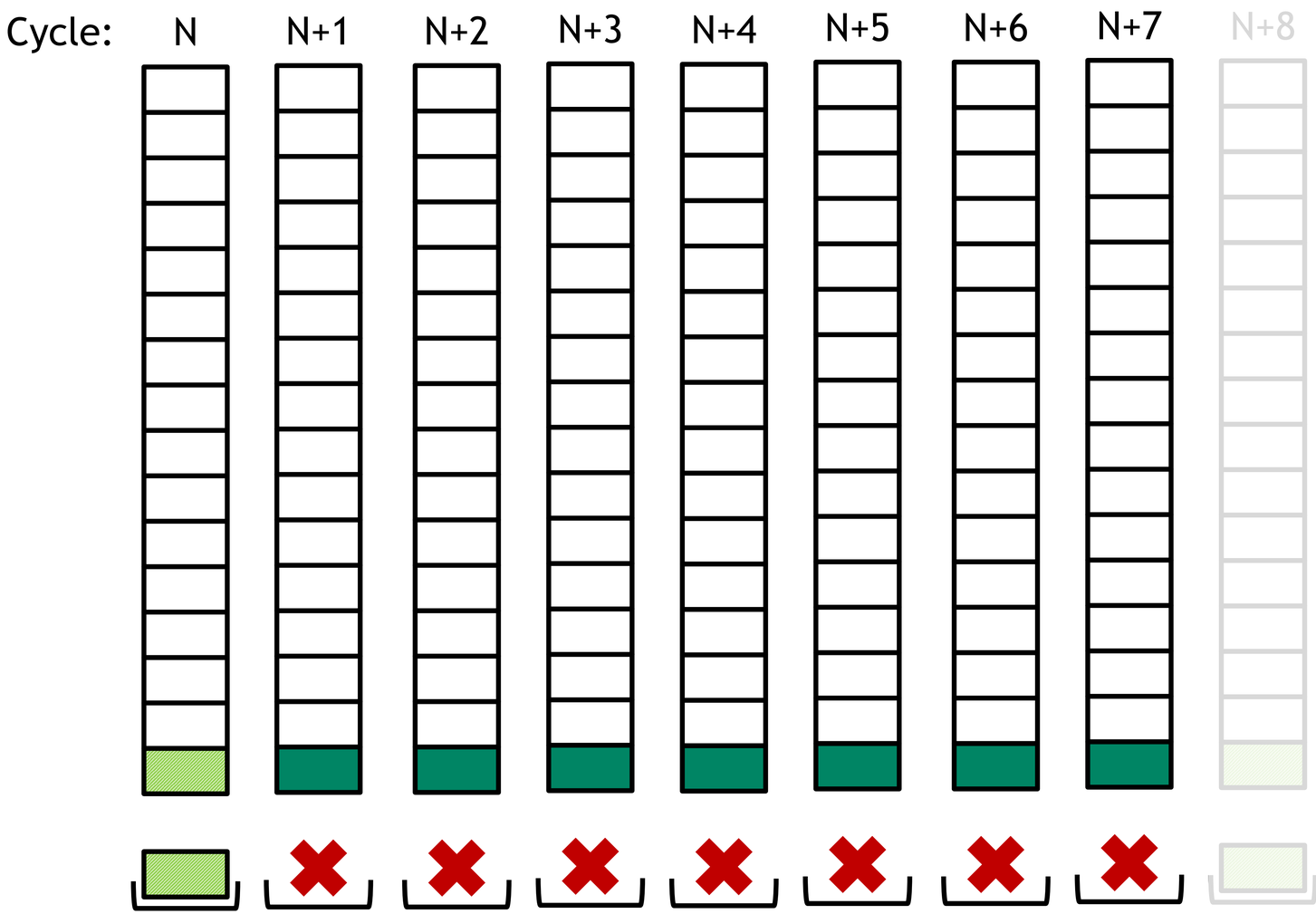
One active warp per scheduler

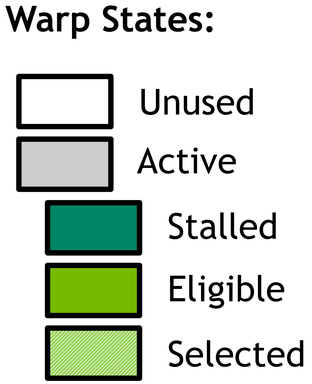
Each thread accumulates PI 1000 times using double-precision

```
__global__ void kernel_A(double* A, int K)
{
  double result = 0.0;
  int i = blockIdx.x * blockDim.x + threadIdx.x;

#pragma unroll (100)
  for (int j = 0; j < K; ++j) {
    result += 3.14;
  }
  A[i] = result;
}


…
kernel_A<<<1, 32>>>(d_f64, 1000);
cudaDeviceSynchronize();
```

WARP SCHEDULER STATISTICS

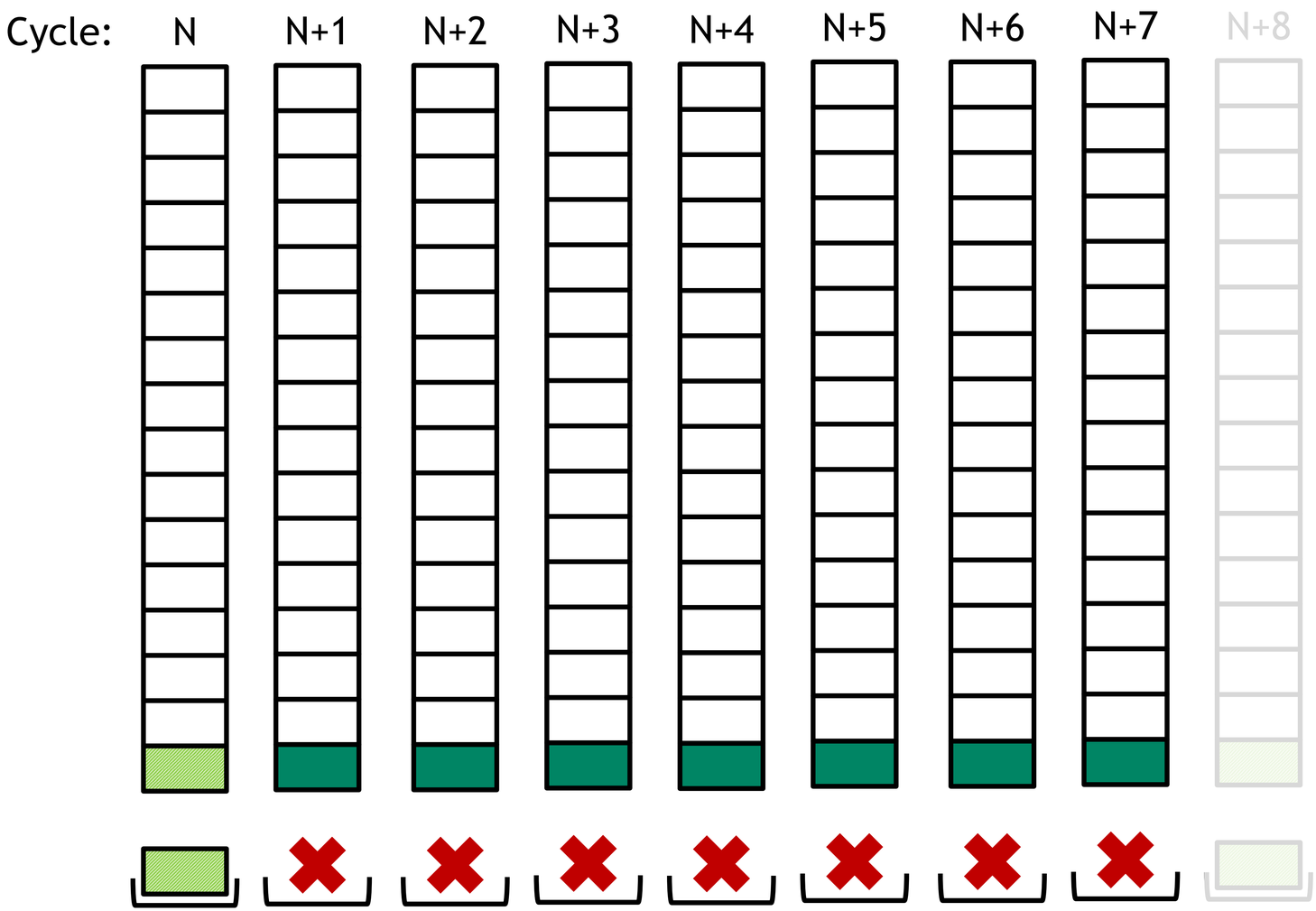
One active warp per scheduler





WARP SCHEDULER STATISTICS

One active warp per scheduler



Warp States:

- Unused
- Active
- Stalled
- Eligible
- Selected

Metrics (theoretical; every 8 cycles):

warps_active	8	1
warps_stalled	7	7/8 = 0.87
warps_eligible	1	1/8 = 0.125
warps_selected	1	1/8 = 0.125

SCHEDULER STATISTICS

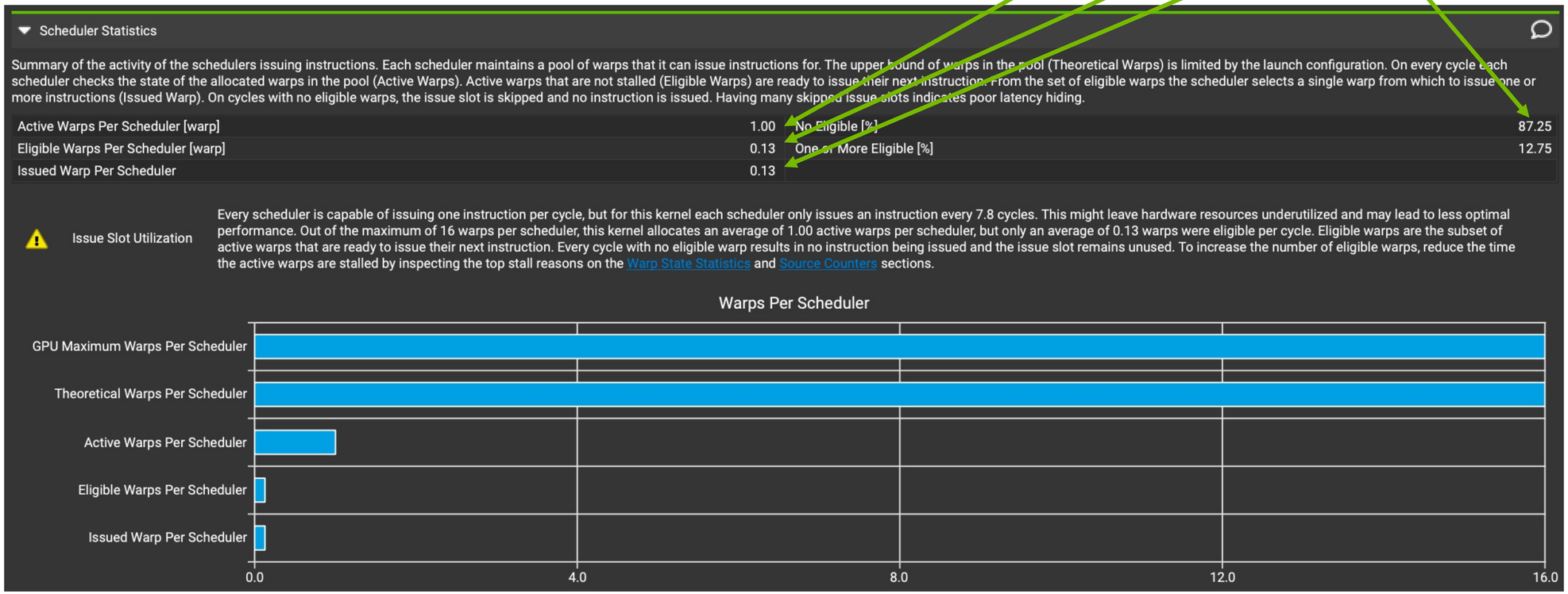
One warp per scheduler

Metrics (theoretical; every 8 cycles):

warps_active	8	1
warps_stalled	7	7/8 = 0.87
warps_eligible	1	1/8 = 0.125
warps_selected	1	1/8 = 0.125

Metrics (from report; rounded):

warps_active	1.00
warps_stalled	0.87
warps_eligible	0.13
warps_selected	0.13



SCHEDULER STATISTICS

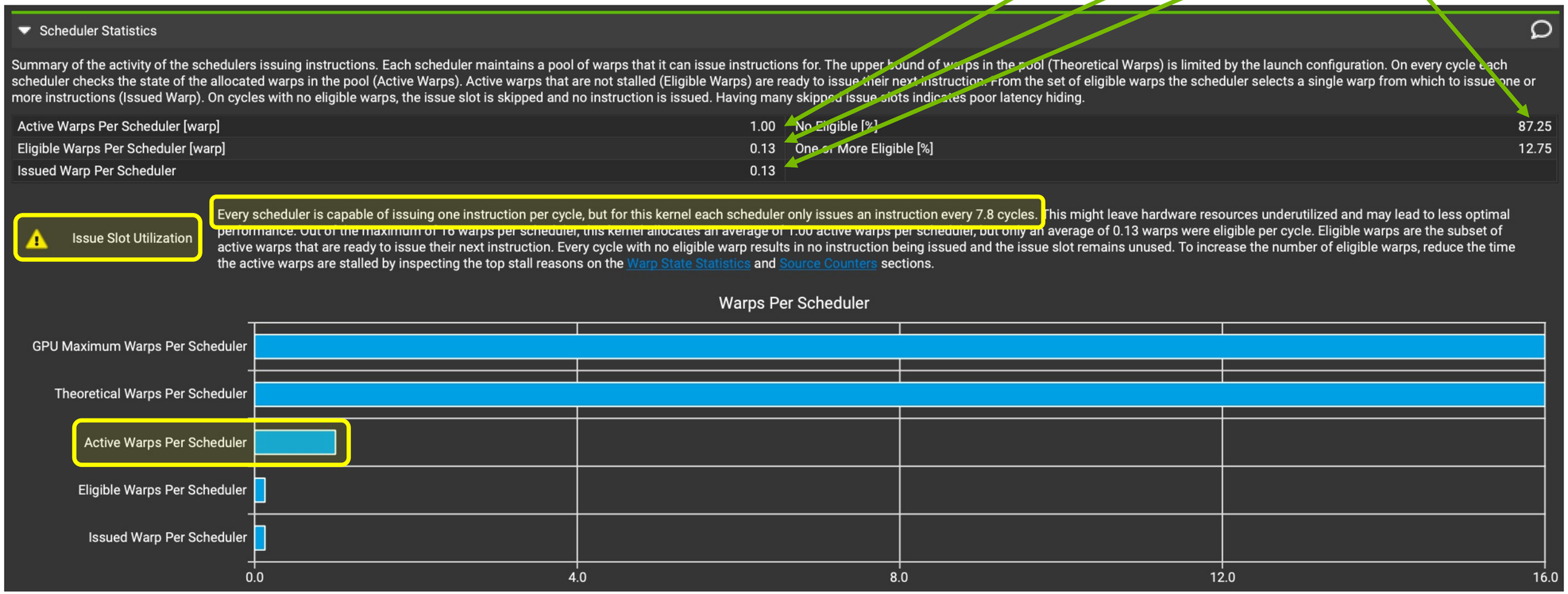
One warp per scheduler

Metrics (theoretical; every 8 cycles):

warps_active	8	1
warps_stalled	7	7/8 = 0.87
warps_eligible	1	1/8 = 0.125
warps_selected	1	1/8 = 0.125

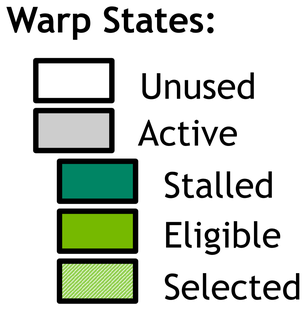
Metrics (from report; rounded):

warps_active	1.00
warps_stalled	0.87
warps_eligible	0.13
warps_selected	0.13

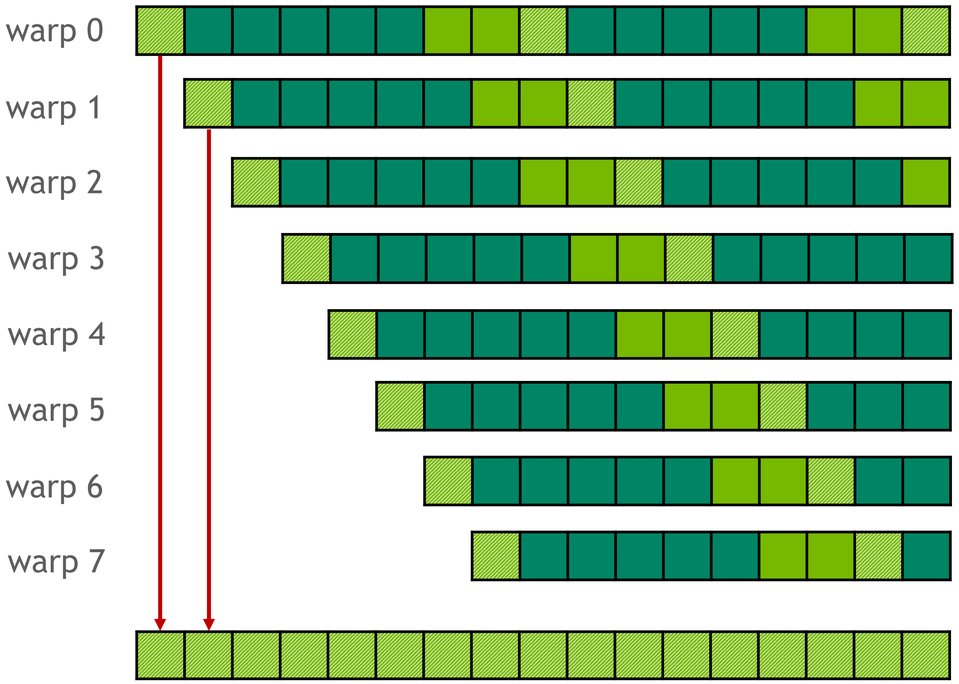


LATENCY

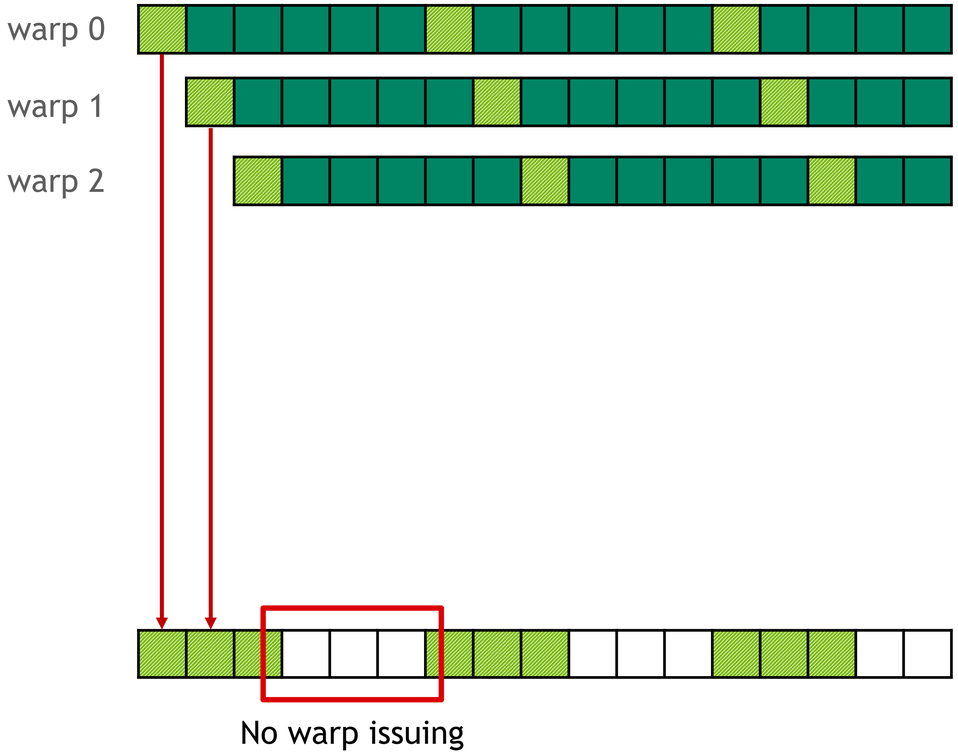
GPUs cover latency by having lots of work in flight



Fully covered latency



Exposed latency, not enough warps

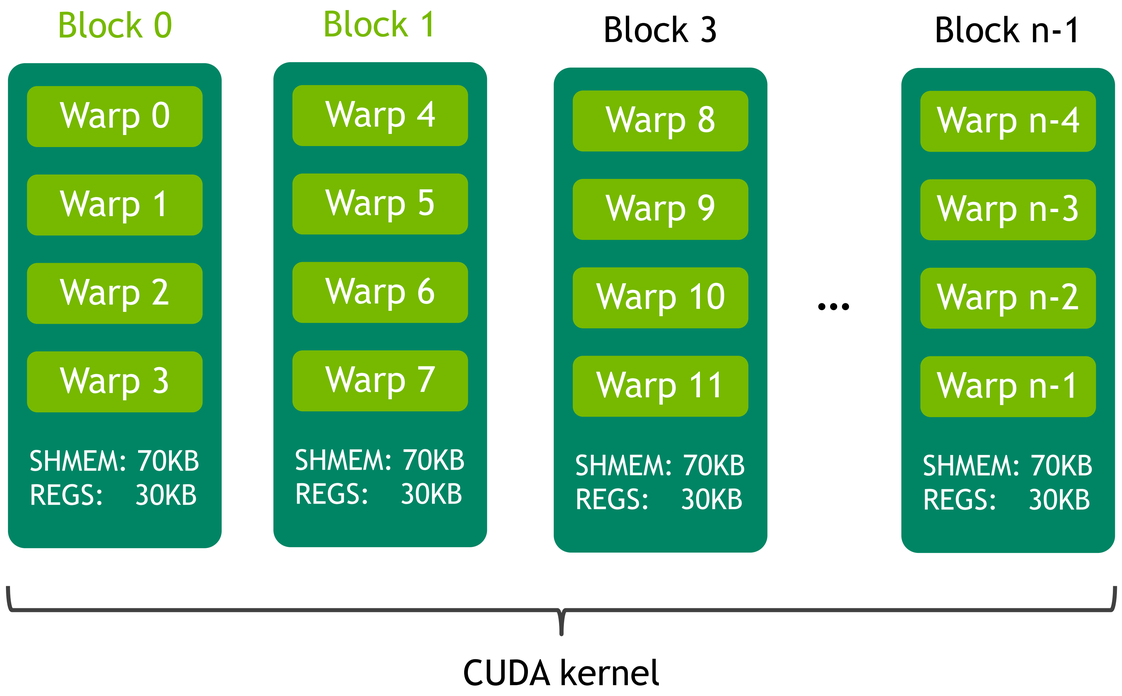


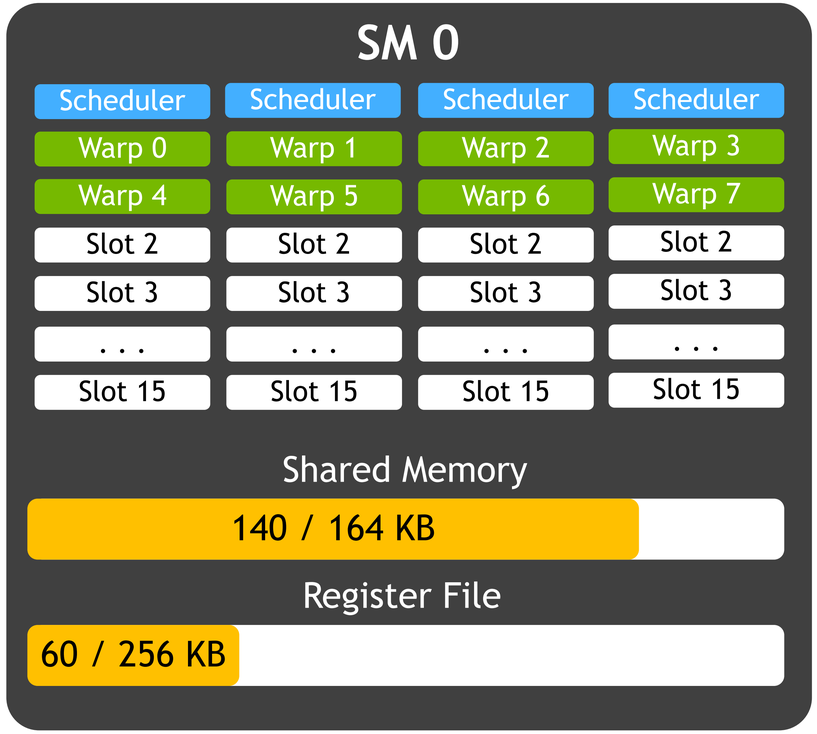
OCCUPANCY

Hide Latency with Higher Occupancy

Occupancy = Active warps per SM / Max warps per SM (64)

Only achieve 12.5% (8/64) occupancy: limiter is shared memory

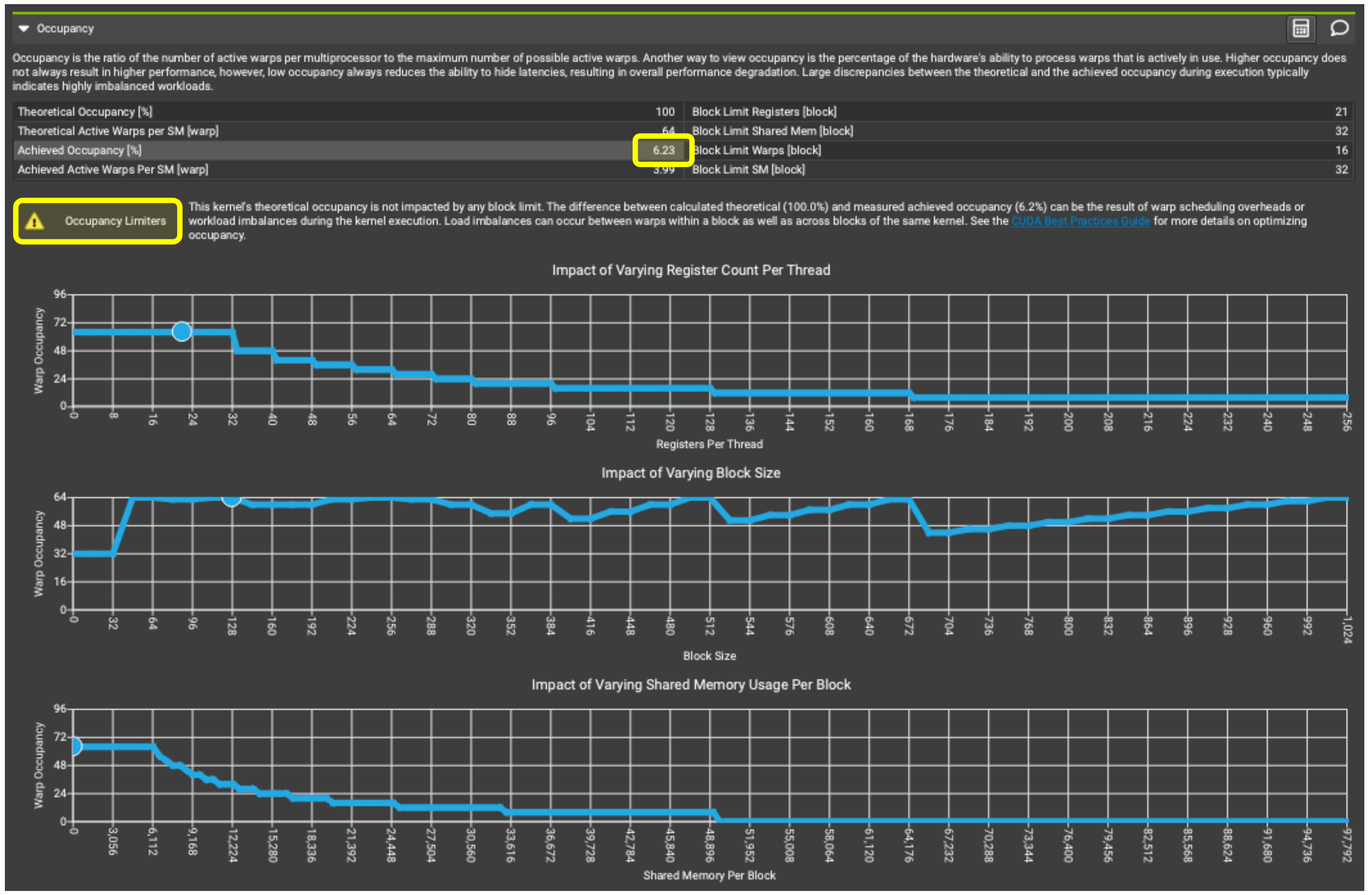




HIDE LATENCY WITH MORE PARALLEL WORK

Improve performance by increasing occupancy

- **Problem:**
  - Warp scheduler can't execute instructions every cycle because it doesn't have enough warps to select from.
- **Indicators:**
  - Occupancy low (< 60%)
  - High execution dependency (low Issue Slot Utilization)
- **Goal:**
  - Hide latency of instructions behind more parallel work
- **Strategy (Limiters that affect occupancy):**
  - Varying grid & block size (not enough work)
  - Reduce shared usage
  - Reduce register count (use `__launch_bounds__`)



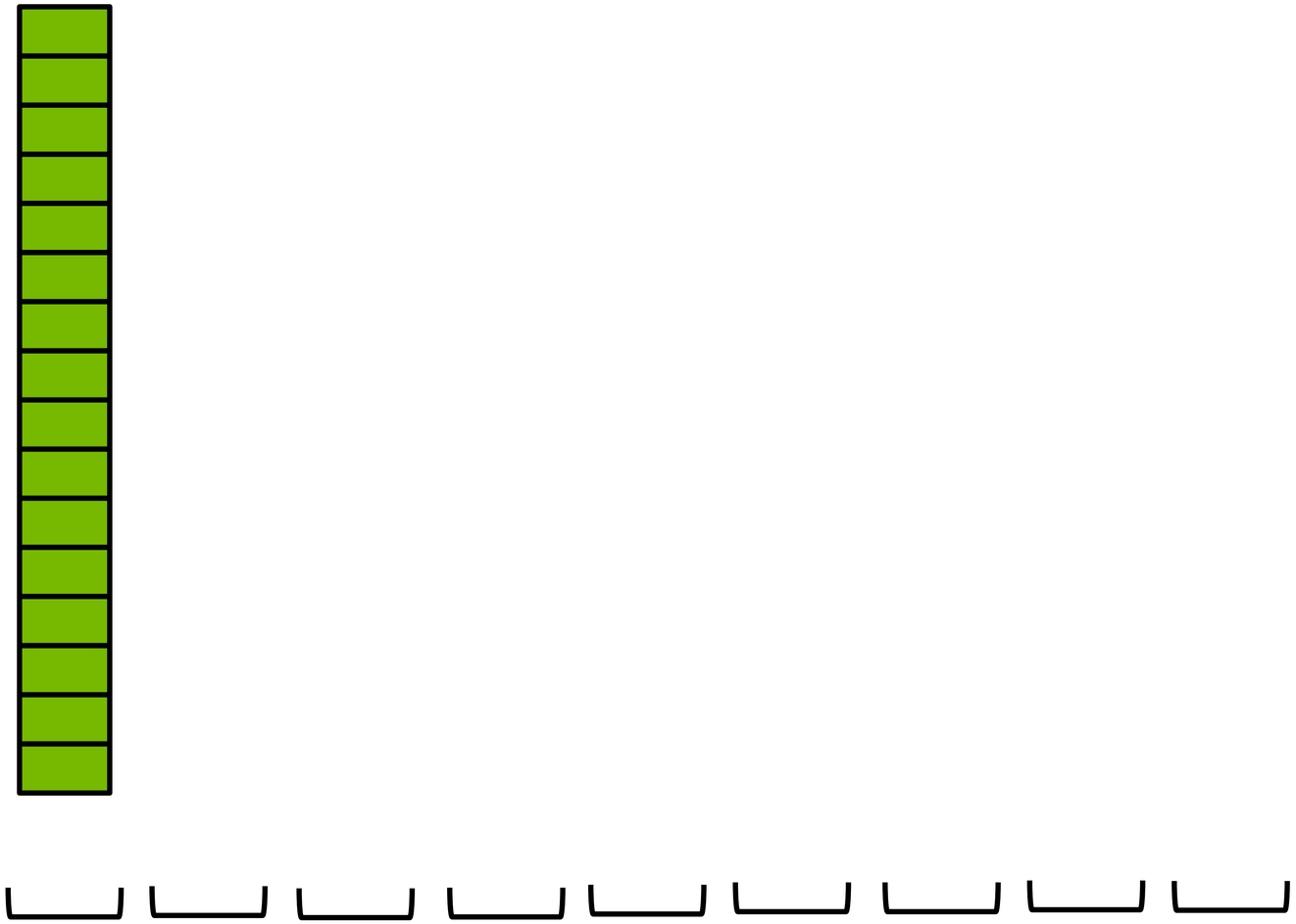
WARP SCHEDULER STATISTICS

kernel_A - bigger grid to saturate all slots

Warp States:

- Unused
- Active
- Stalled
- Eligible
- Selected

Cycle: N N+1 N+2 N+3 N+4 N+5 N+6 N+7 N+8



FP64 Pipeline on Ampere

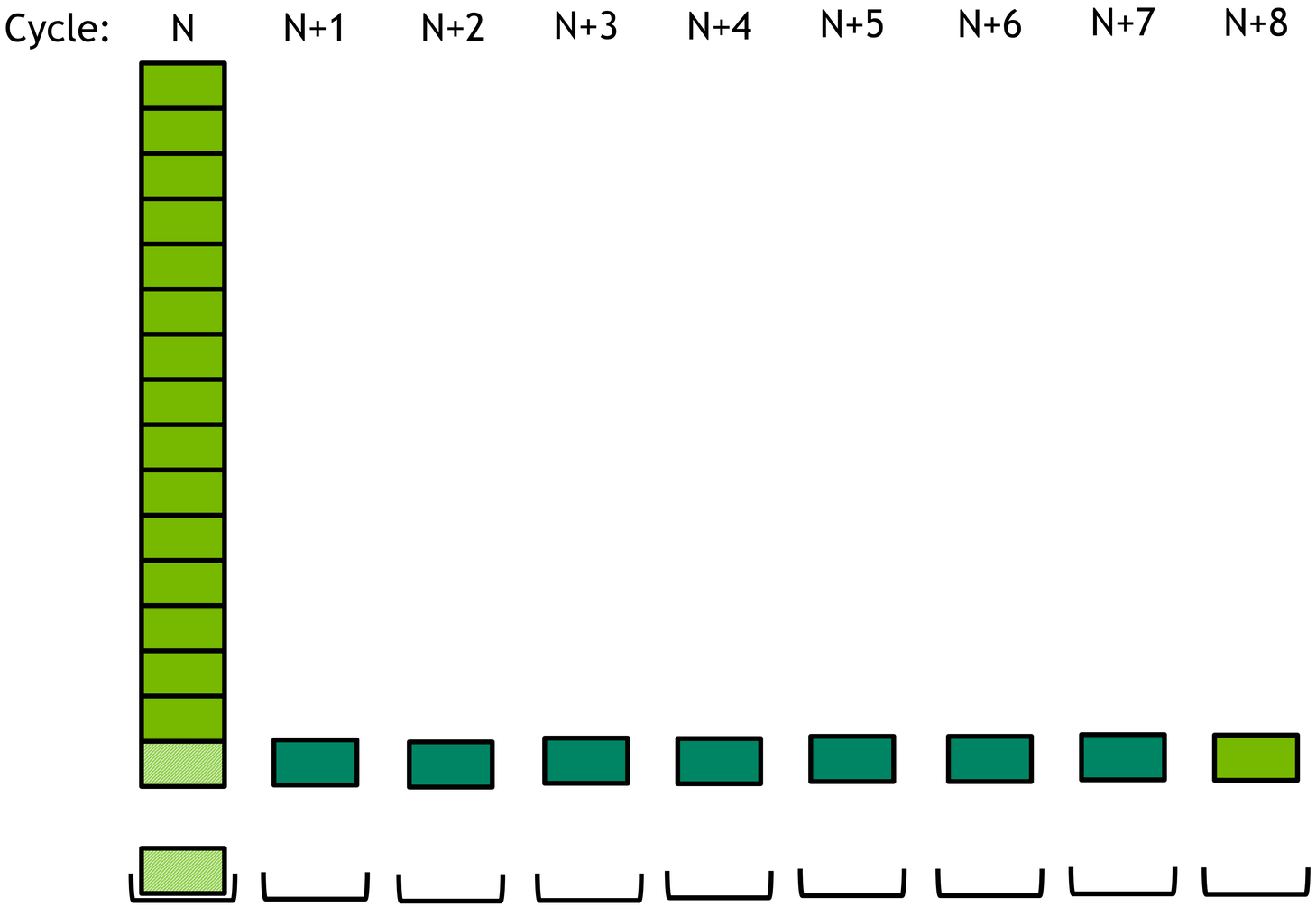
Dependent Issue Rate: 8 cycles
Issue Rate: 4 cycles

WARP SCHEDULER STATISTICS

kernel_A - bigger grid to saturate all slots

Warp States:

- Unused
- Active
- Stalled
- Eligible
- Selected

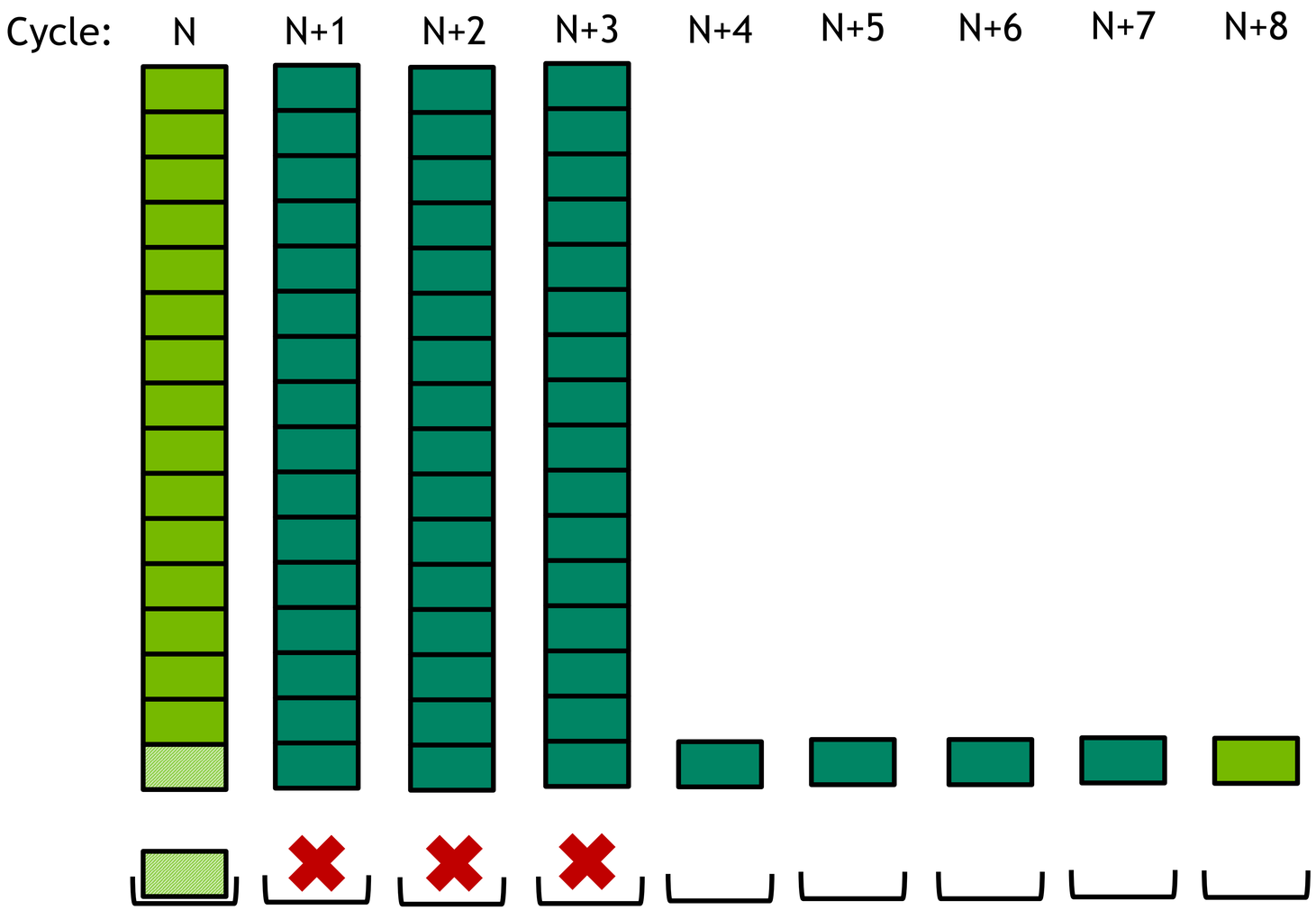


FP64 Pipeline on Ampere

Dependent Issue Rate: 8 cycles
Issue Rate: 4 cycles

WARP SCHEDULER STATISTICS

kernel_A - bigger grid to saturate all slots



Warp States:

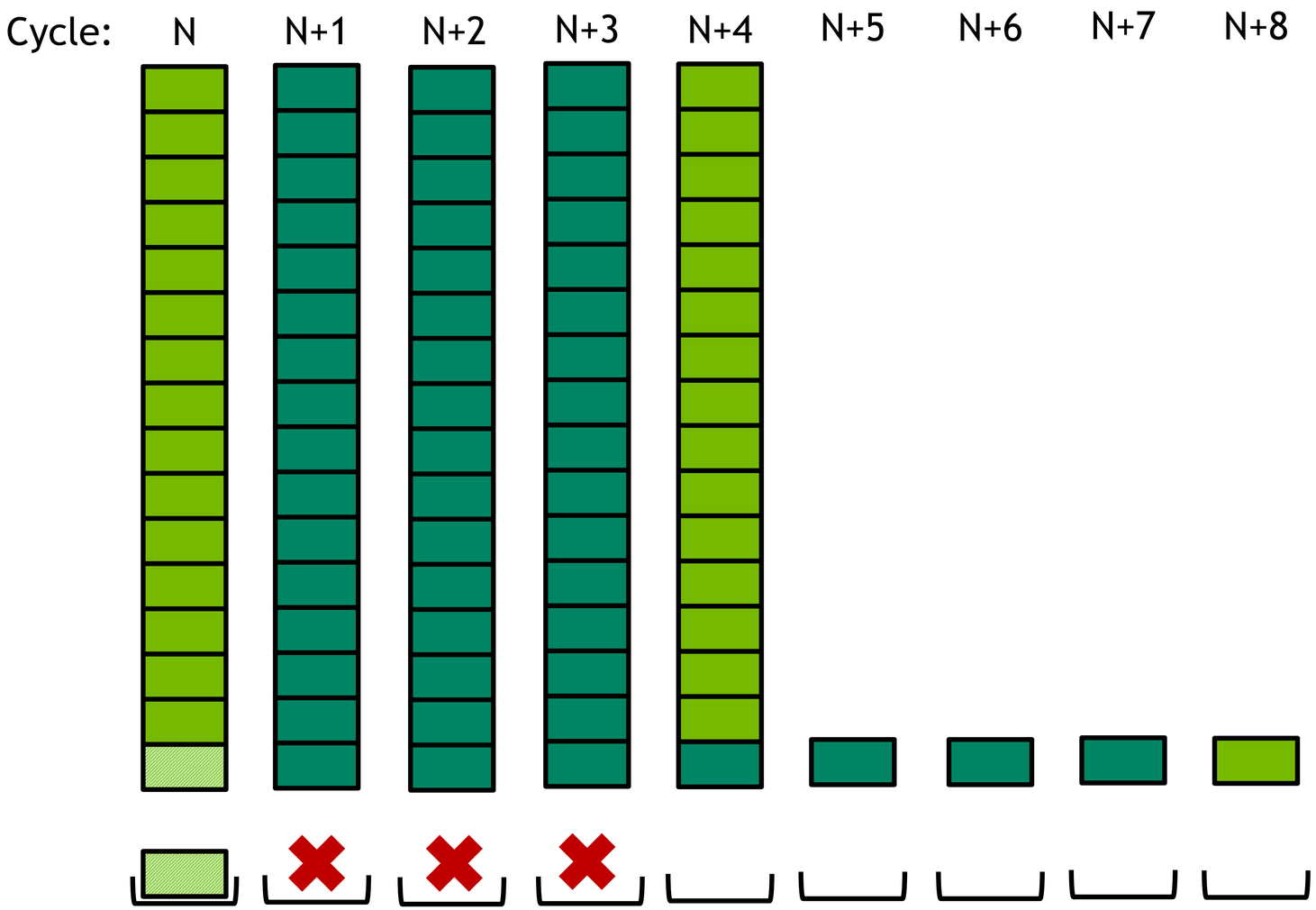
- Unused
- Active
- Stalled
- Eligible
- Selected

FP64 Pipeline on Ampere

Dependent Issue Rate: 8 cycles
Issue Rate: 4 cycles

WARP SCHEDULER STATISTICS

kernel_A - bigger grid to saturate all slots



Warp States:

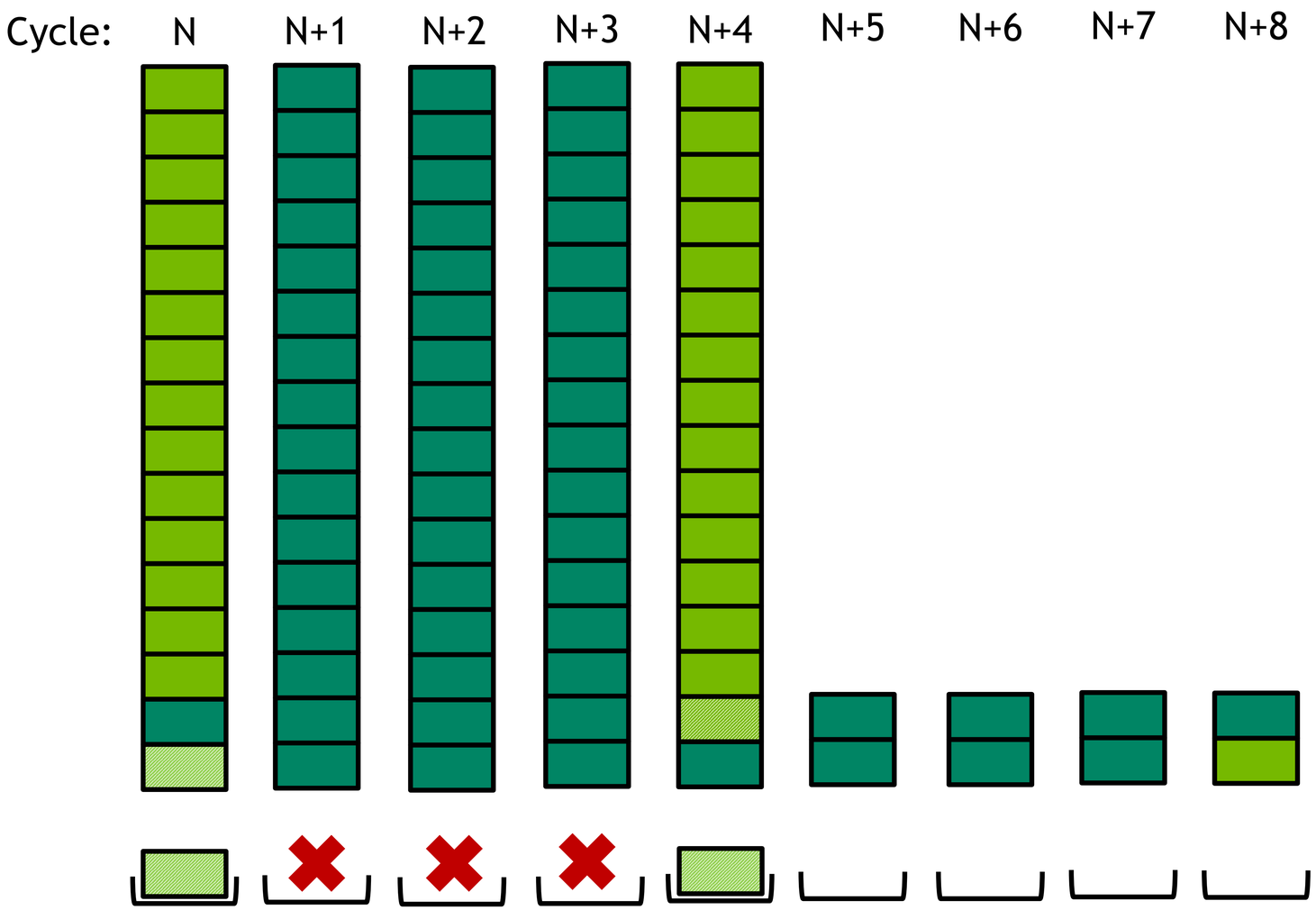
- Unused
- Active
- Stalled
- Eligible
- Selected

FP64 Pipeline on Ampere

Dependent Issue Rate: 8 cycles
Issue Rate: 4 cycles

WARP SCHEDULER STATISTICS

kernel_A - bigger grid to saturate all slots



Warp States:

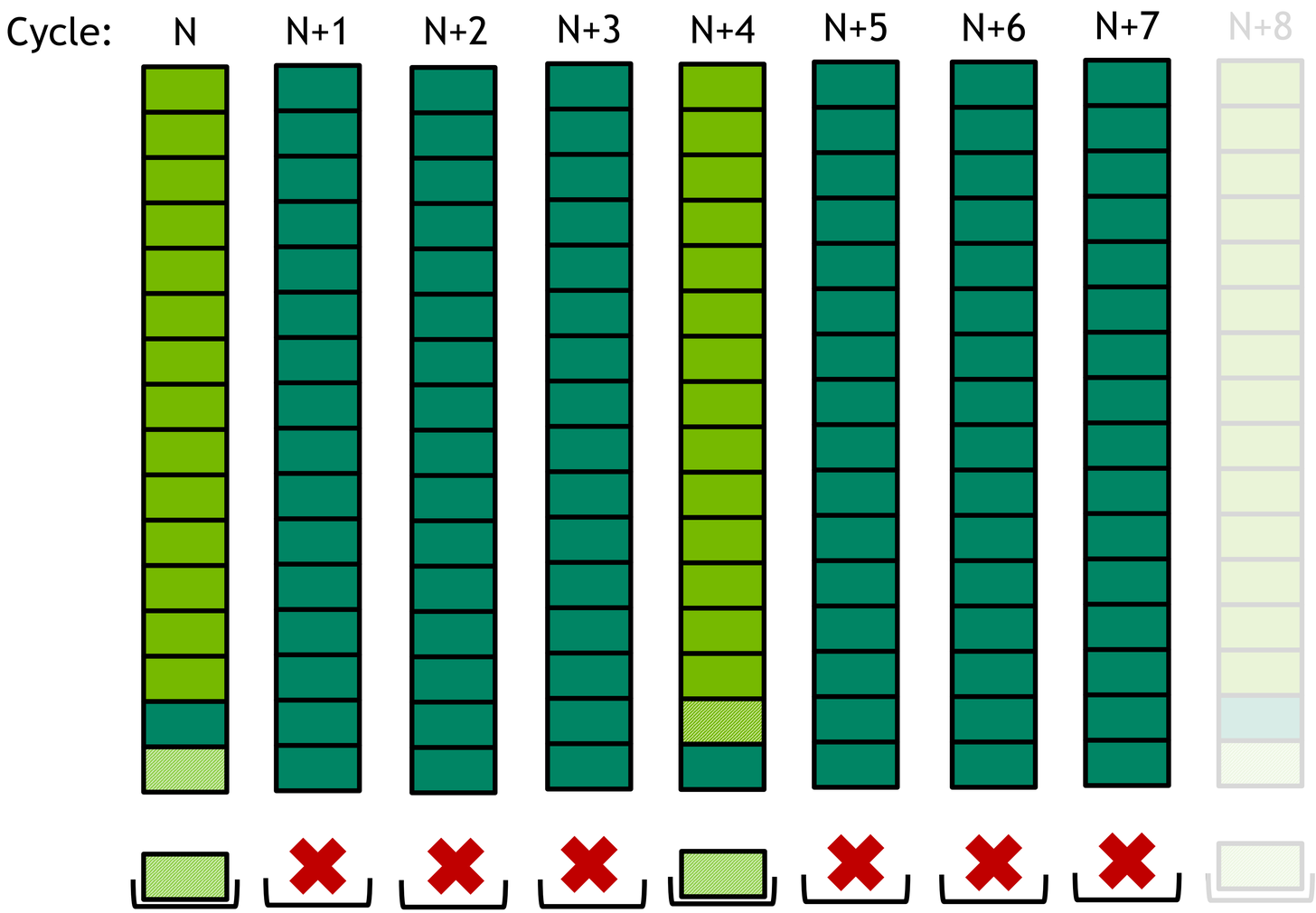
- Unused
- Active
- Stalled
- Eligible
- Selected

FP64 Pipeline on Ampere

Dependent Issue Rate: 8 cycles
Issue Rate: 4 cycles

WARP SCHEDULER STATISTICS

kernel_A - bigger grid to saturate all slots



Warp States:

- Unused
- Active
- Stalled
- Eligible
- Selected

Metrics (theoretical; every 8 cycles):

warps_active	128	128/8 =16
warps_stalled	98	98/128=0.76
warps_eligible	30	30/8 =3.75
warps_selected	2	2/8 =0.25

WARP SCHEDULER STATISTICS

Verifying the Metrics

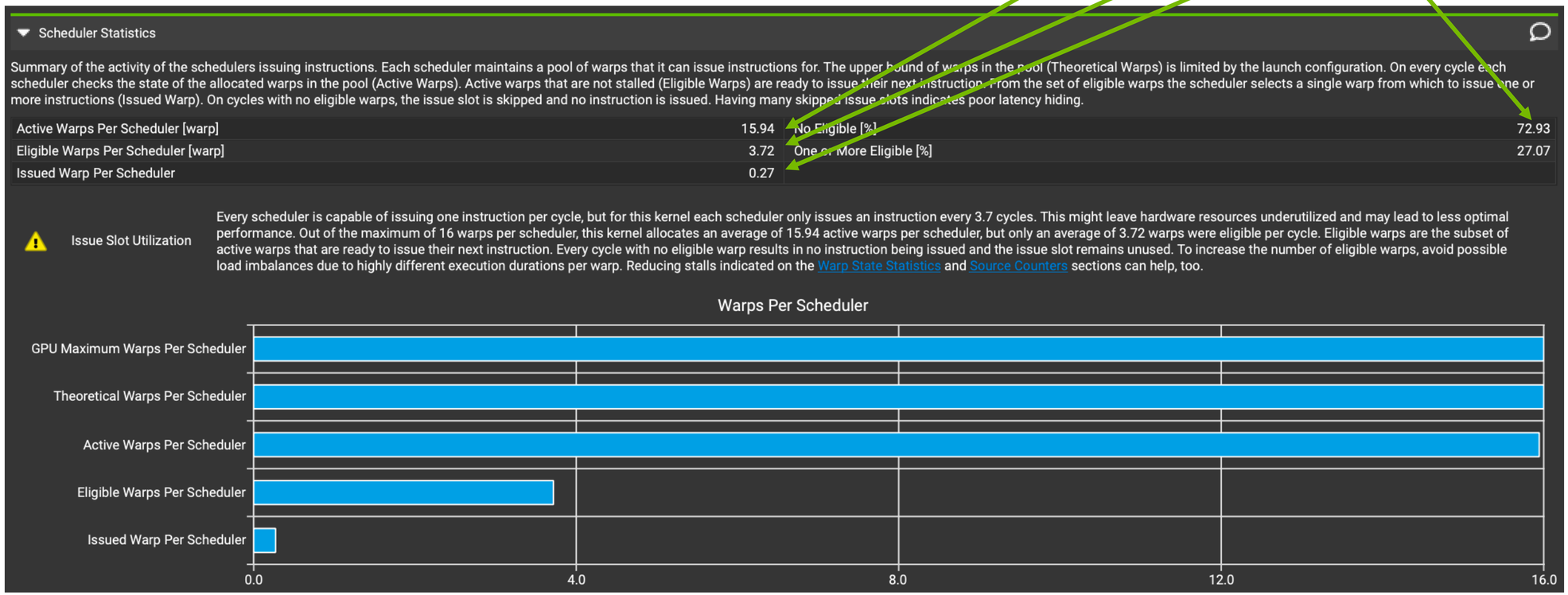
Metrics (theoretical; every 8 cycles):

warps_active	128	128/8 =16
warps_stalled	98	98/128=0.76
warps_eligible	30	30/8 =3.75
warps_selected	2	2/8 =0.25

Metrics (from report; rounded):

warps_active	15.94
warps_stalled	0.73
warps_eligible	3.72
warps_selected	0.27

Improved by 2x



WARP SCHEDULER STATISTICS

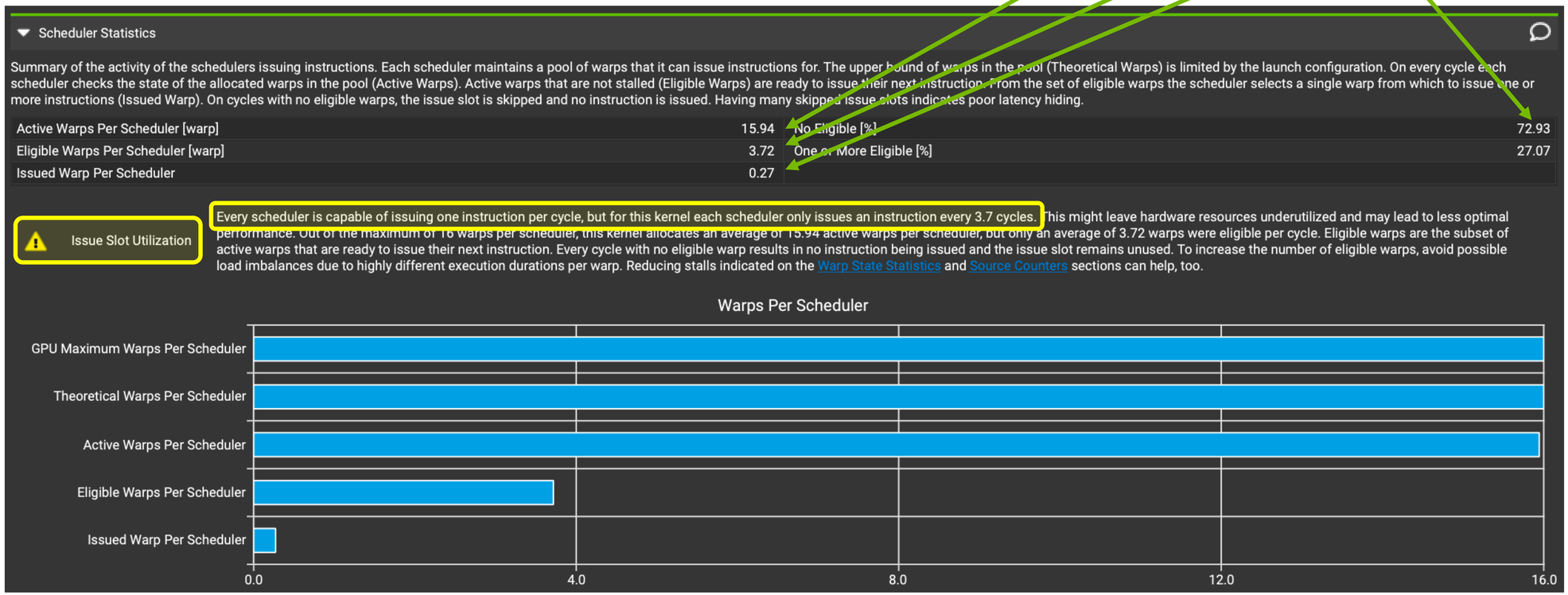
Still a Issue Slot Utilization Problem

Metrics (theoretical; every 8 cycles):

```
warps_active          128    128/8  =16
warps_stalled         98      98/128=0.76
warps_eligible        30       30/8 =3.75
warps_selected        2         2/8 =0.25
```

Metrics (from report; rounded):

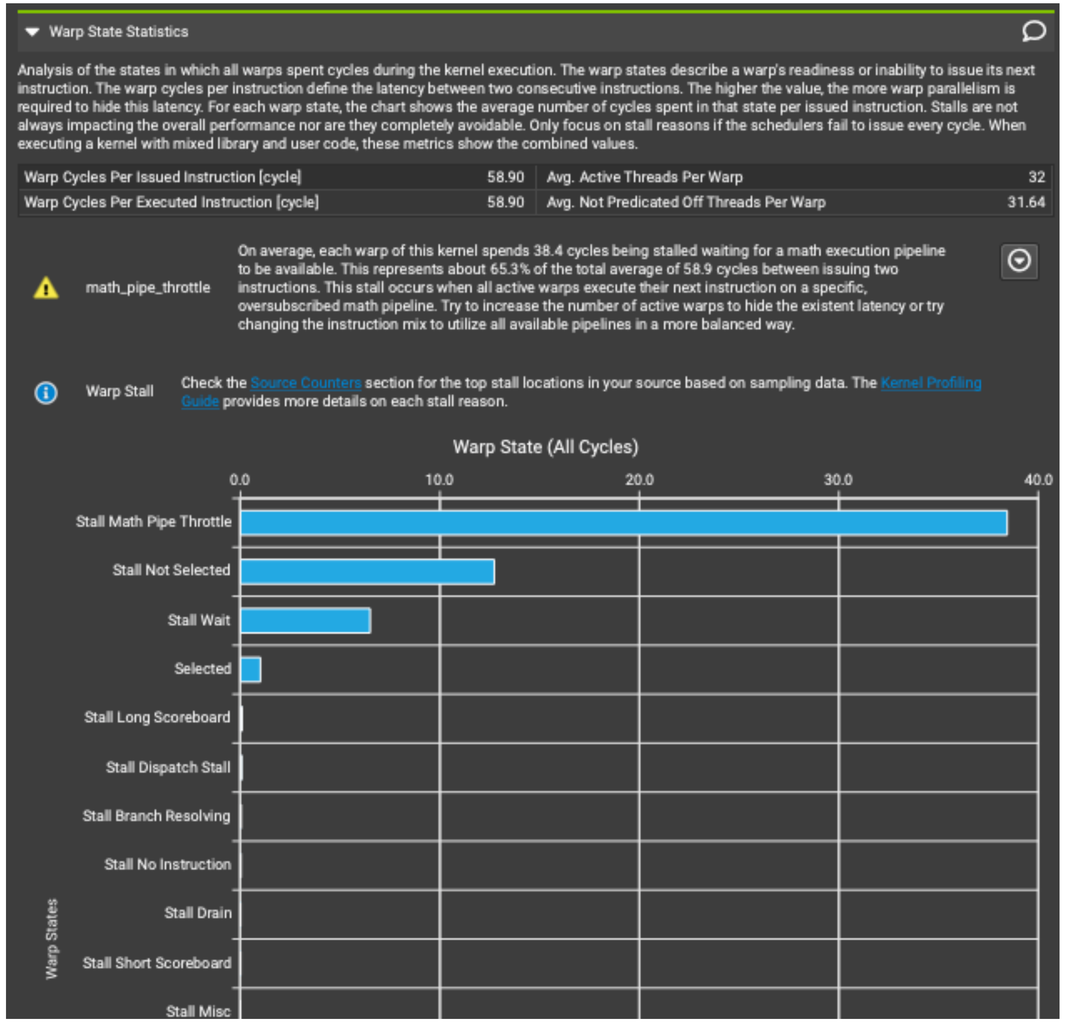
```
warps_active          15.94
warps_stalled          0.73
warps_eligible        3.72
warps_selected        0.27
```



WARP STATE STATISTICS

Sections

- **Warp State** chart shows you the average number of cycles a warp is stalled in each state across all warps.
- Gives you analysis on what is preventing your warp from issuing instructions.
- Stalls do not always impact overall performance, nor are they avoidable.
- Only focus on stall reasons if scheduler has a low issue slot utilization.

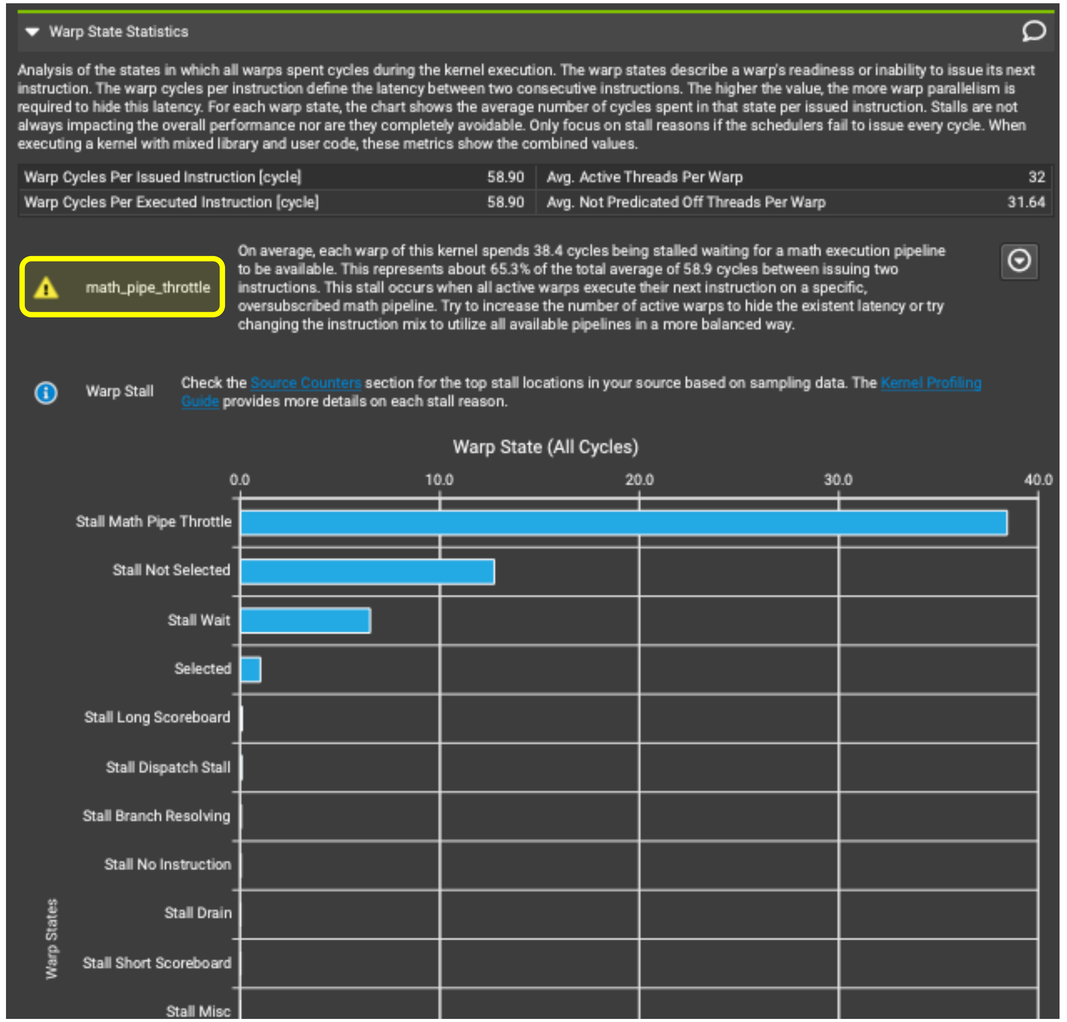


WARP STATE STATISTICS

Stall Reasons

Key stall reasons to watch out for:

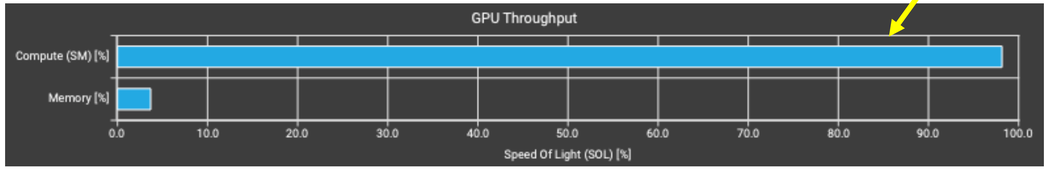
- **Stall Math Pipe Throttle**
  Waiting for ALU, FP32, FP64 pipelines to be free.
- **Stall Long Scoreboard**
  Waiting on L1TEX memory request (local, global, texture).
- **Stall Short Scoreboard**
  Typically waiting on shared memory request
- **Stall Not Selected**
  Waiting for warp scheduler to select and issue instruction for this warp. Reduce active warps to possibly increase cache coherence.
- **Stall Branch Resolving**
  Waiting for target branches to resolve (i.e lots of thread divergence).

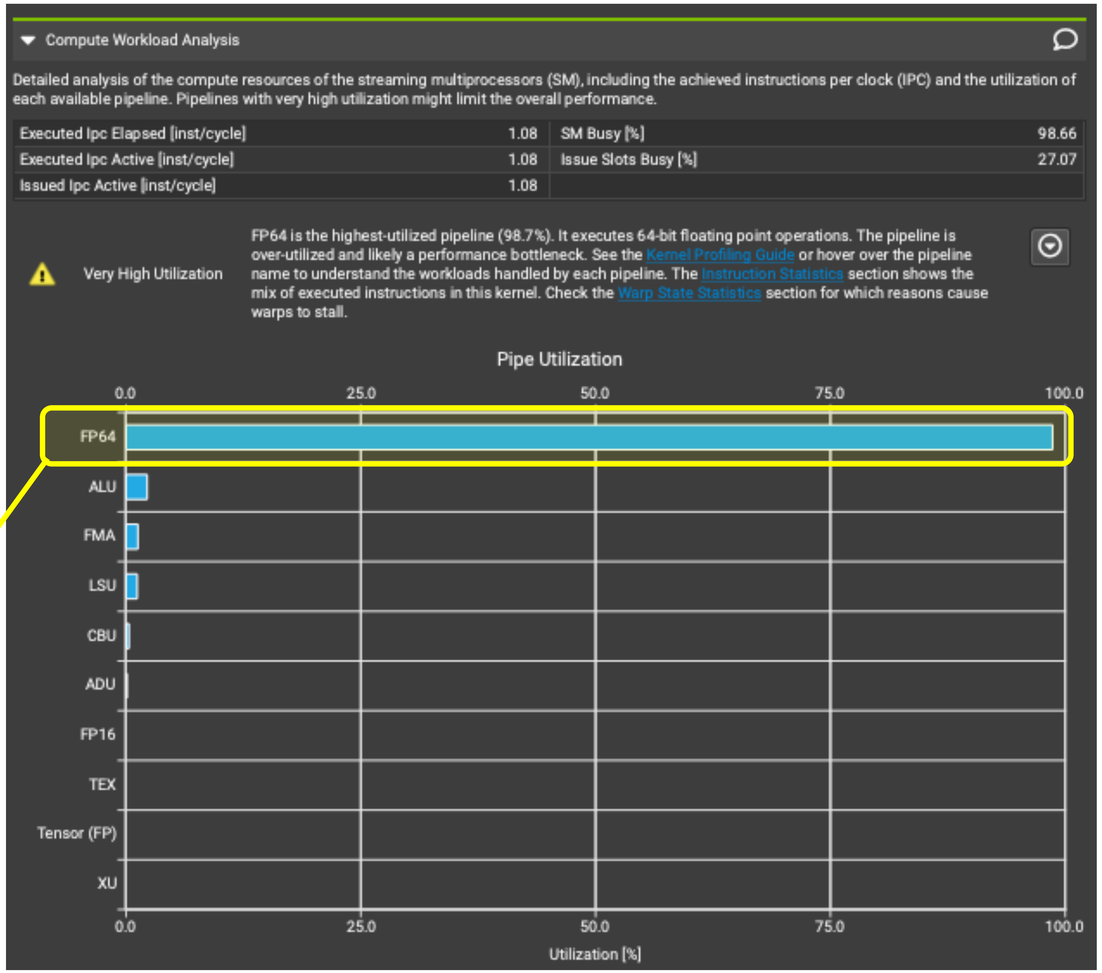


COMPUTE WORKLOAD ANALYSIS

Sections

- **Pipe Utilization** gives you detailed analysis of compute resources on the SM, including achieved instructions per clock (IPC), and the utilization of each pipeline.
- Pipelines with very high utilization might limit overall performance.
- You should look at this section when you are **Compute Bound**



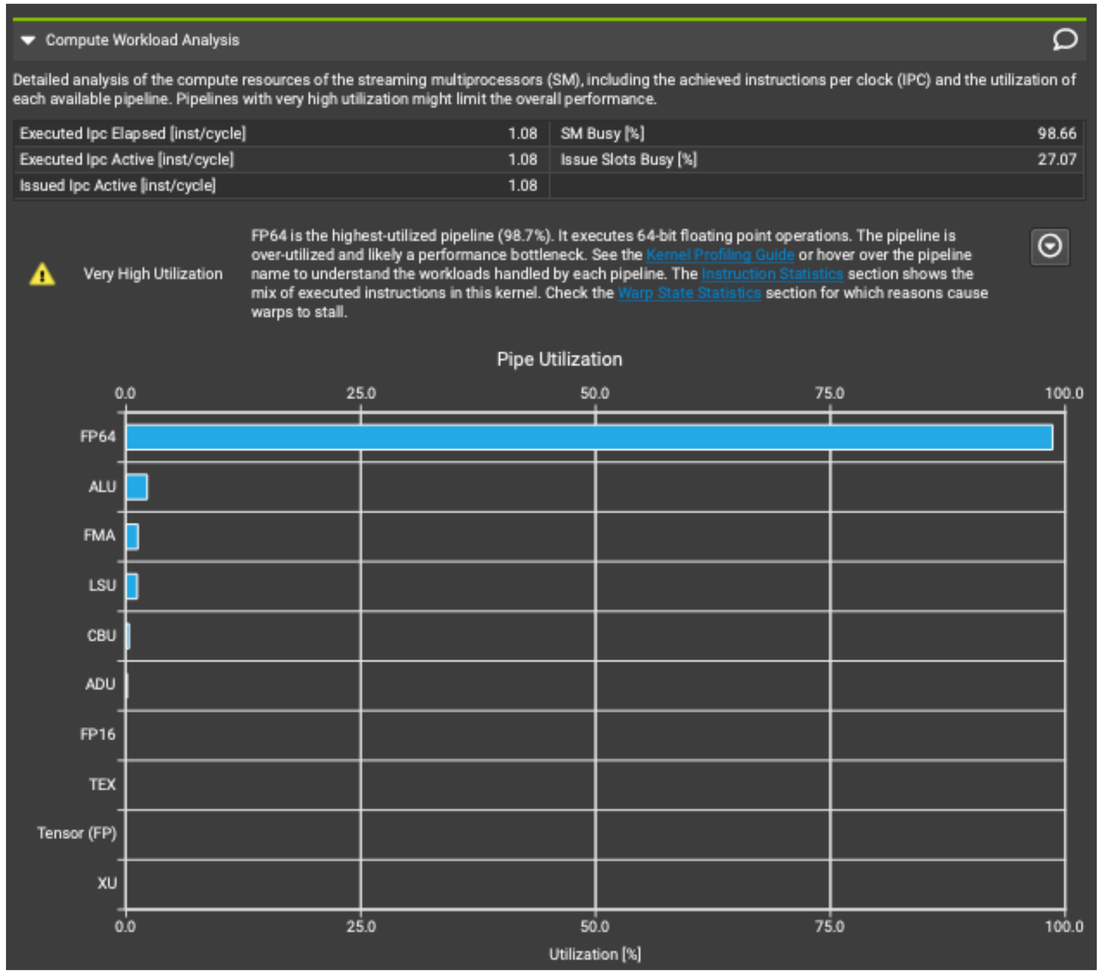


COMPUTE WORKLOAD ANALYSIS

Pipelines

Key pipelines to look out for:

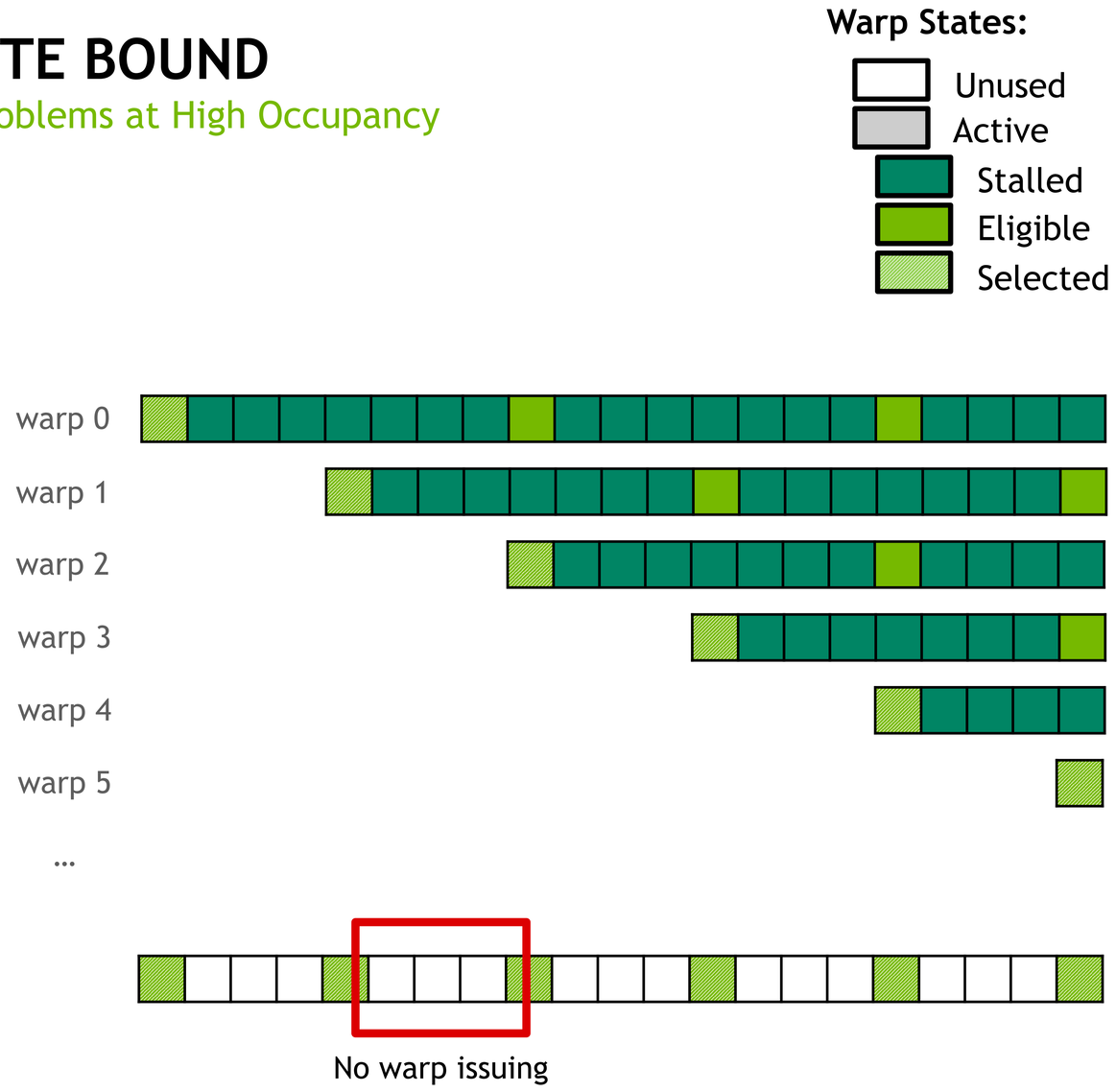
- FMA
  Fused-multiply-add. FP32 (single-precision) arithmetic.
- FP64
  FP64 (double-precision) arithmetic.
- ALU
  Integer arithmetic, bit manipulation & logic instructions.
- LSU
  Load Store Unit. Issues load, store & atomic instructions to L1TEX unit for global, local & shared memory.

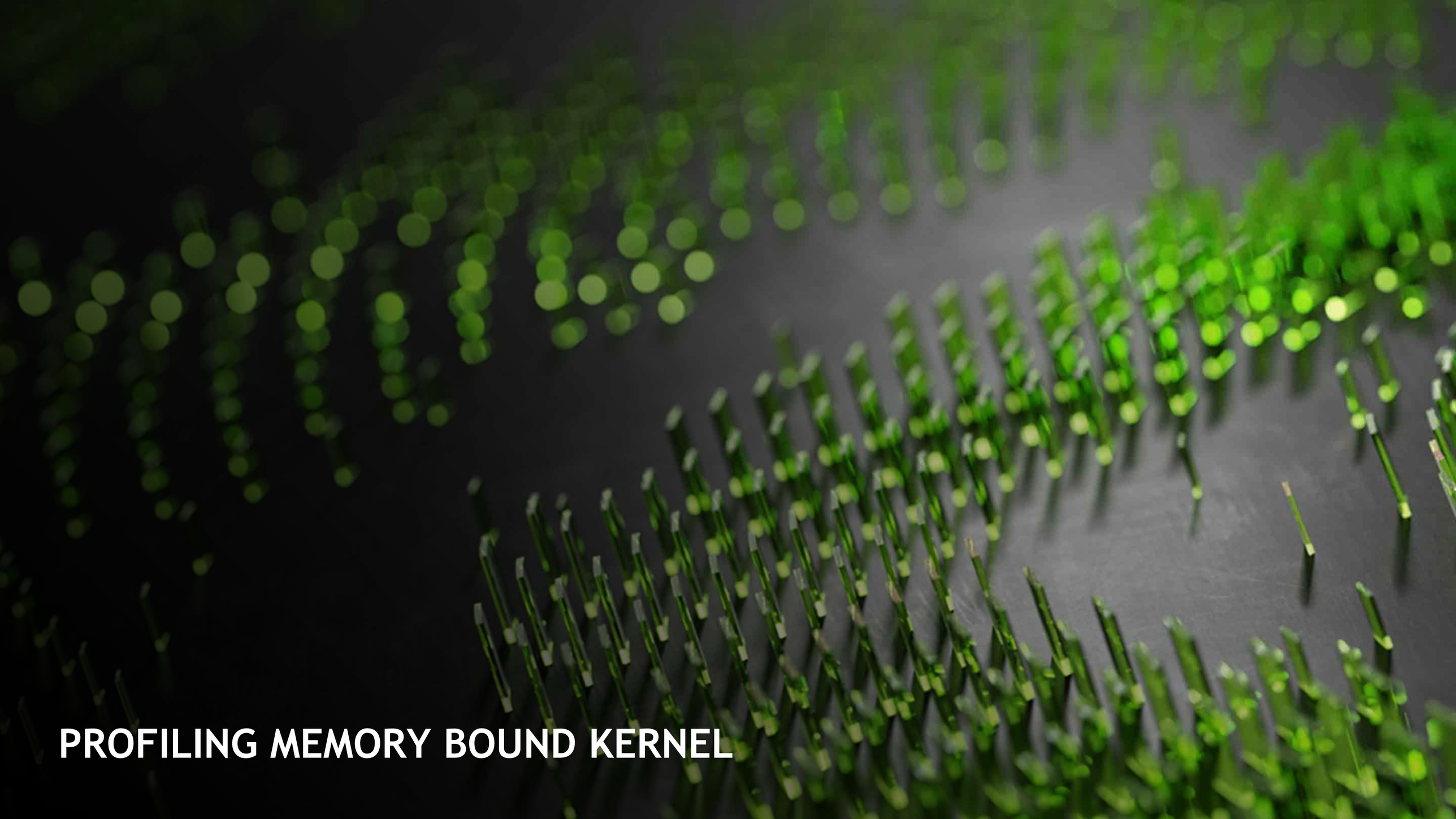


COMPUTE BOUND

Kernel_A: Latency Problems at High Occupancy

- **Problem:**
  - Even with higher occupancy, the latency of the fp64 pipeline is still exposing 4 cycles where the warp is idle and could be doing other work.
- **Goal:**
  - Issue a warp every cycle (or close to it)
- **Indicators:**
  - High compute utilization
  - High Math Throttle Stall
  - High execution dependency (low Issue Slot Utilization)
- **Strategy:**
  - Use instructions that take fewer cycles (FP32, FP16, Tensor Cores)
  - Instruction Level Parallelism (ILP): Hide latency of these instructions with other instructions that use **different** pipelines (ALU, FP32, LDS, etc)



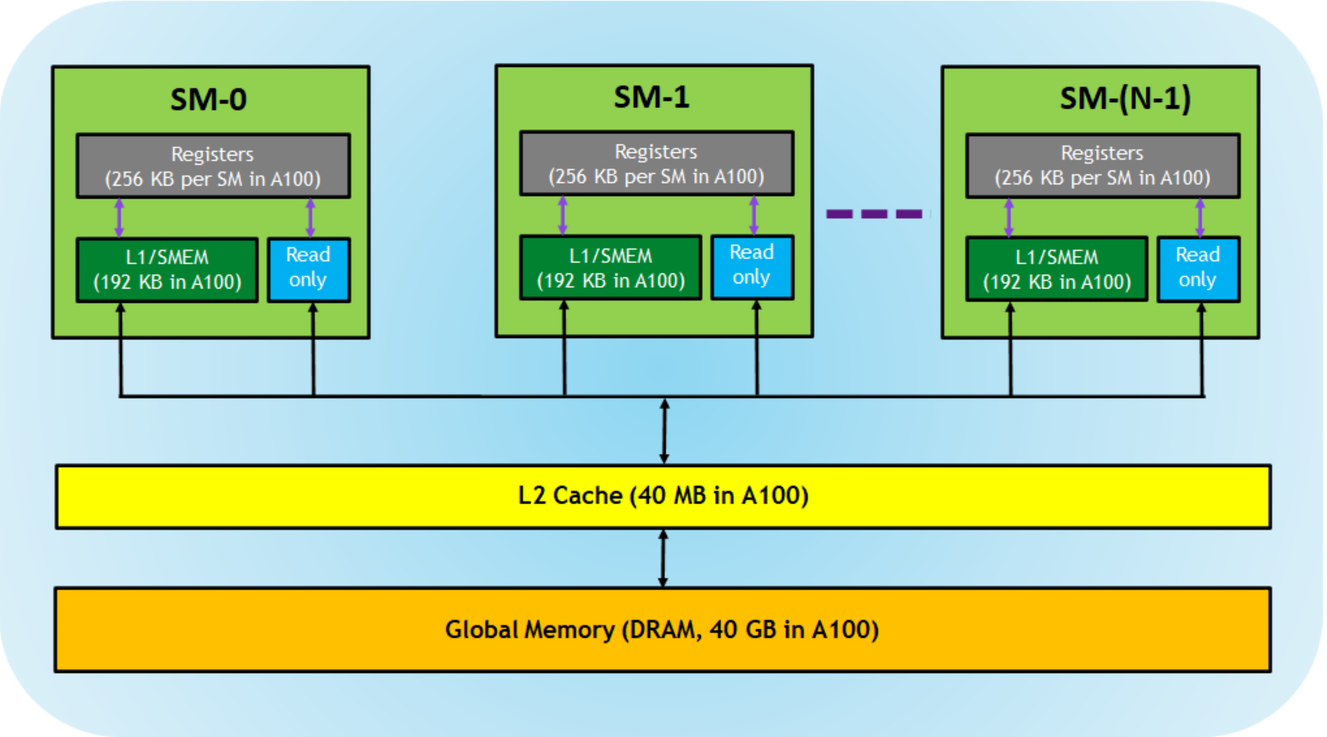


PROFILING MEMORY BOUND KERNEL

AMPERE'S MEMORY SUBSYSTEM

A100

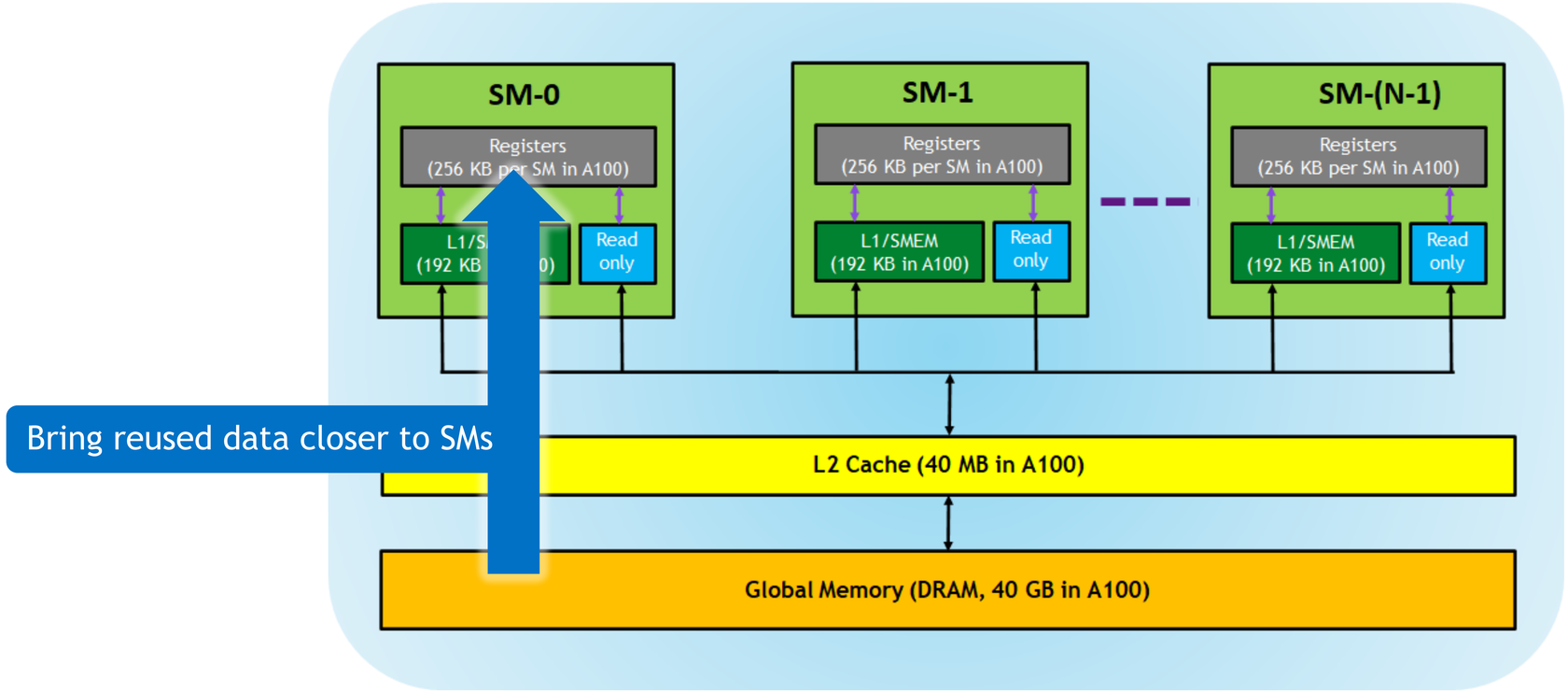
- 108 Symmetric Streaming Multiprocessors (SMs)
- 256KB register file per SM
- **Unified** Shared Mem/ L1 Cache
  192KB, Variable split
- 40MB L2 Cache
- 40/80 GB HMB2 DRAM
  2 TB/s



AMPERE'S MEMORY SUBSYSTEM

A100

- 108 Symmetric Streaming Multiprocessors (SMs)
- 256KB register file per SM
- **Unified** Shared Mem/ L1 Cache
  192KB, Variable split
- 40MB L2 Cache
- 40/80 GB HMB2 DRAM
  1555 TB/s

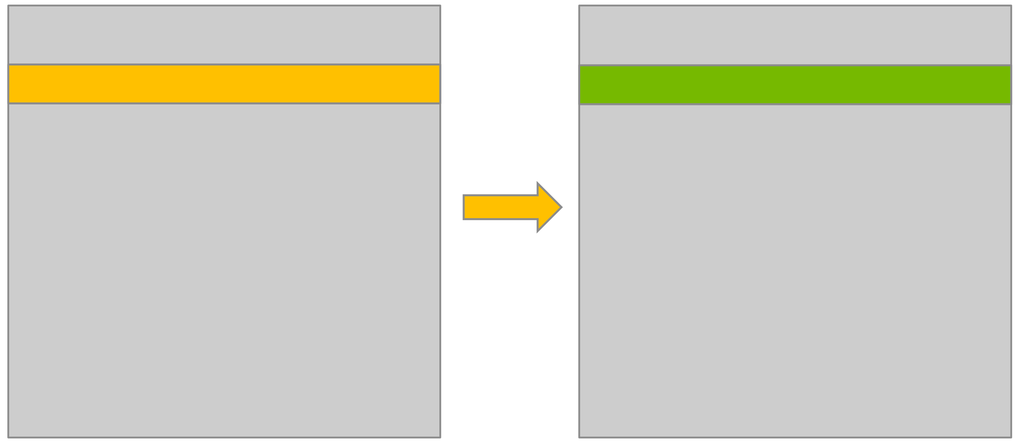


MEMORY BOUND KERNEL

Matrix copy

```
#define DIM 32

__global__ void copy(float* idata,
                     float* odata) {
  int matrix_id = blockIdx.x;
  int row = threadIdx.y;
  int col = threadIdx.x;

  float* imat = &idata[matrix_id * DIM * DIM];
  float* omat = &odata[matrix_id * DIM * DIM];

  omat[row * DIM + col] = imat[row * DIM + col];
}
```

Matrix Copy



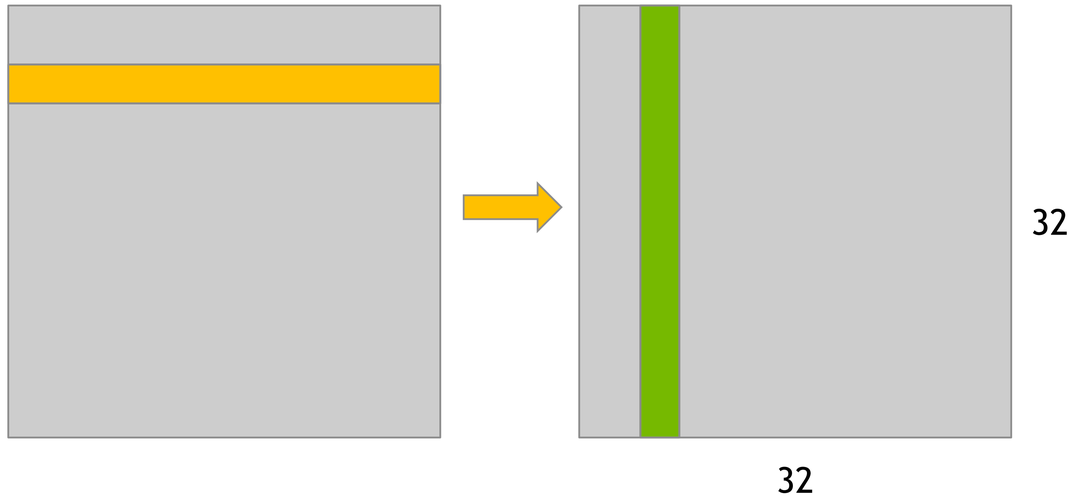
	Effective Bandwidth (GB/s)
GPU	NVIDIA Titan V (650 GB/s)
Copy	580 (90% peak)

MEMORY BOUND KERNEL

Matrix Transpose

```
#define DIM 32

__global__ void transposeNaive(float* idata,
                               float* odata) {
  int matrix_id = blockIdx.x;
  int row = threadIdx.y;
  int col = threadIdx.x;

  float* imat = &idata[matrix_id * DIM * DIM];
  float* omat = &odata[matrix_id * DIM * DIM];

  omat[col * DIM + row] = imat[row * DIM + col];
}
```

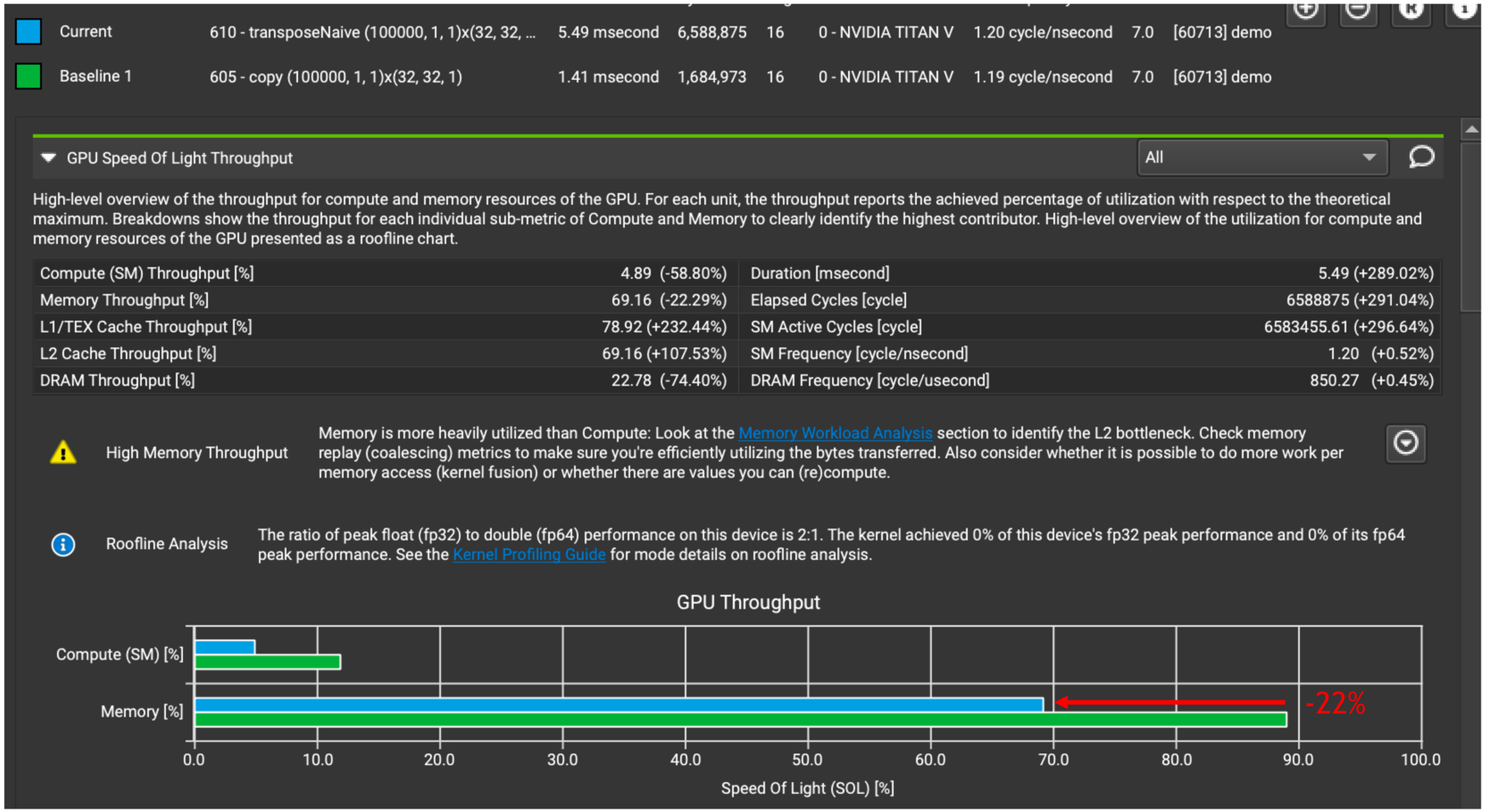
Matrix Transpose



	Effective Bandwidth (GB/s)
GPU	NVIDIA Titan V (650 GB/s)
copy	580 (90% peak)
transposeNaive	149

LOOKING FOR INDICATORS

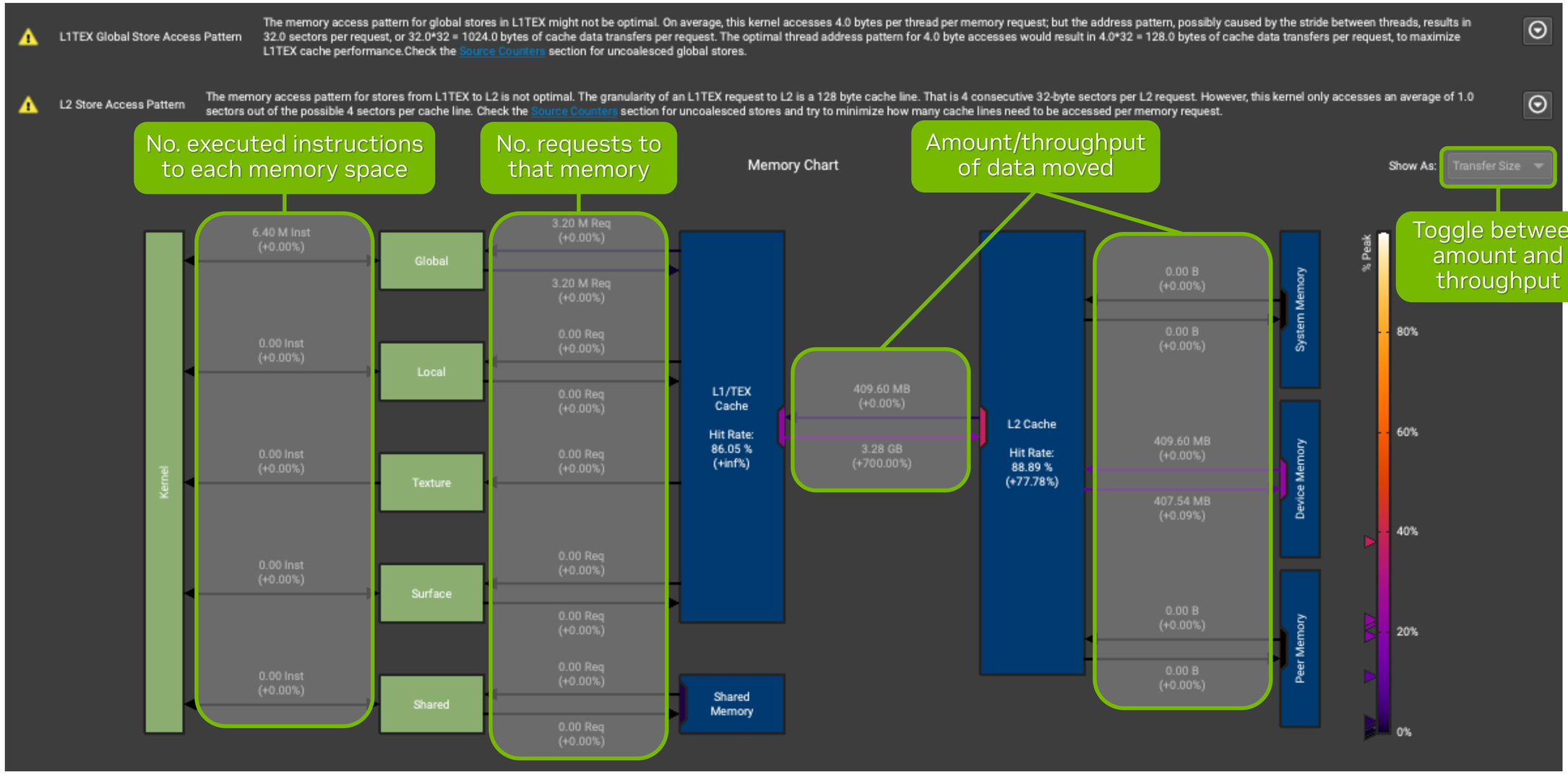
Start with Speed Of Light



	Effective Bandwidth (GB/s)	Speedup
copy	580 (90% peak)	
transposeNaive	149	1.00x

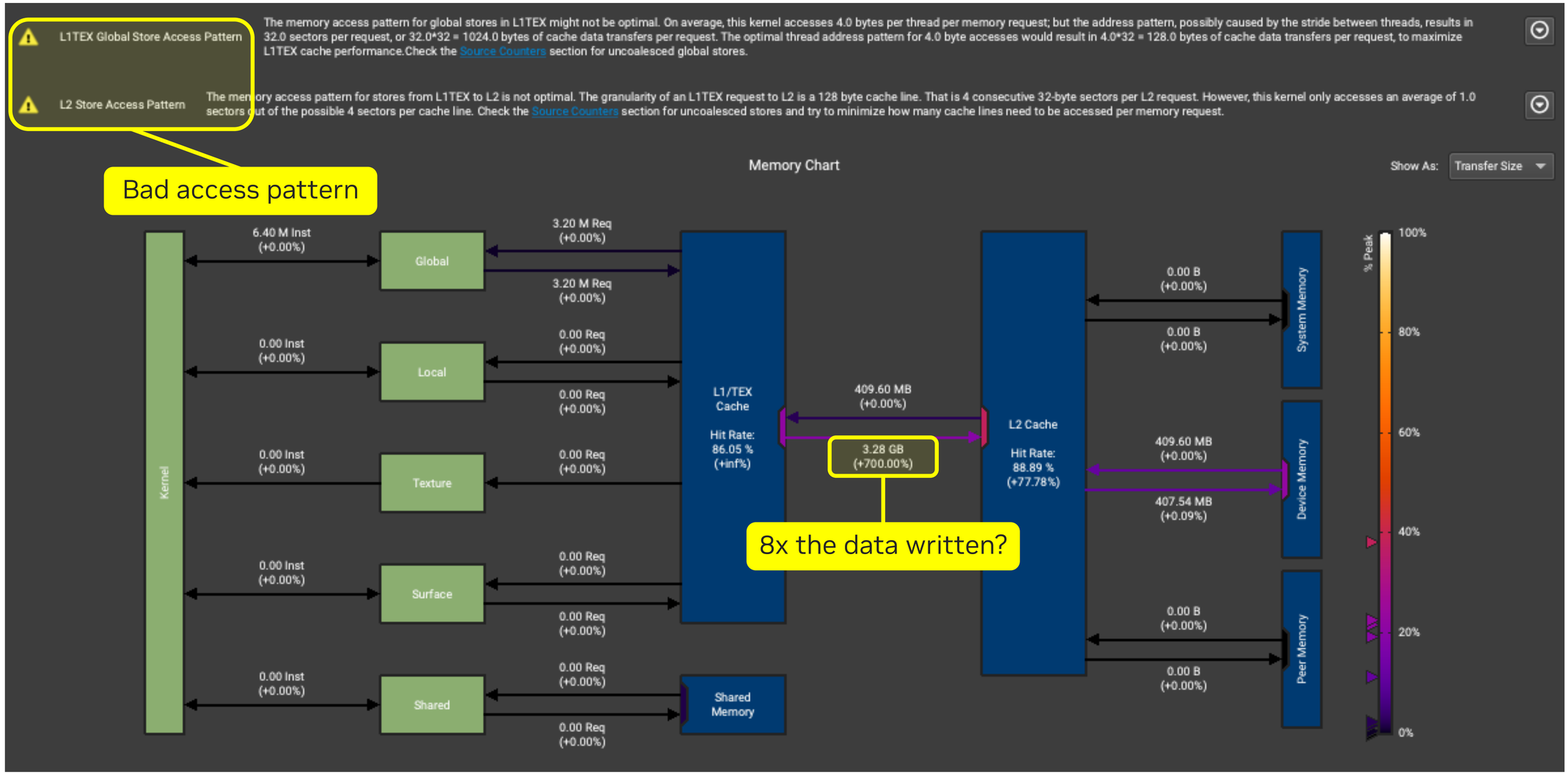
MEMORY WORKLOAD ANALYSIS

Logical Units vs Physical Units



MEMORY WORKLOAD ANALYSIS

Look for Indicators

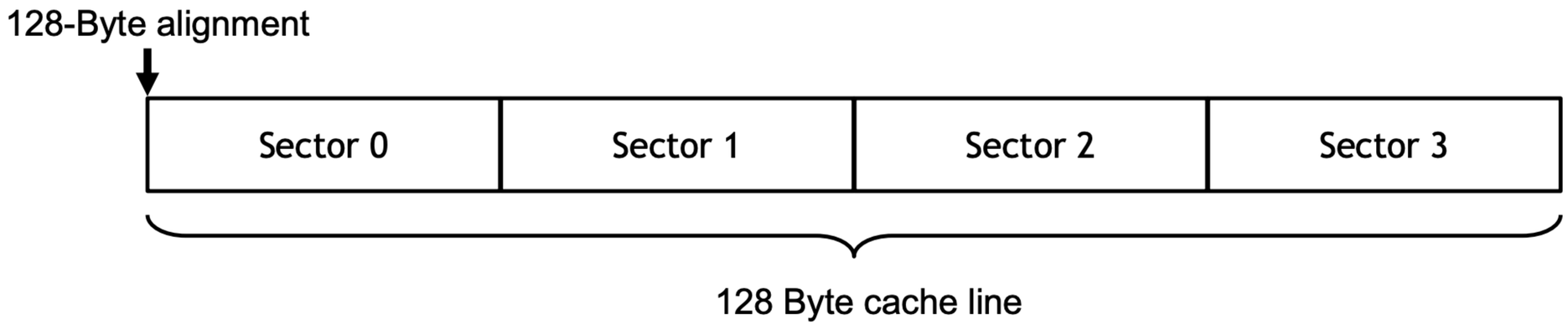


REFRESHER: CACHE LINES AND SECTORS

Moving data between L1, L2 & DRAM

- Memory access granularity = **32 Bytes = 1 sector**
  (32B for Maxwell, Pascal, Volta, Ampere. Kepler and before: variable, 32B or 128B, depending on architecture, access type, caching / non-caching options)

- A **cache line is 128 Bytes**, made of **4 sectors**. Cache ”management” granularity = 1 cache line



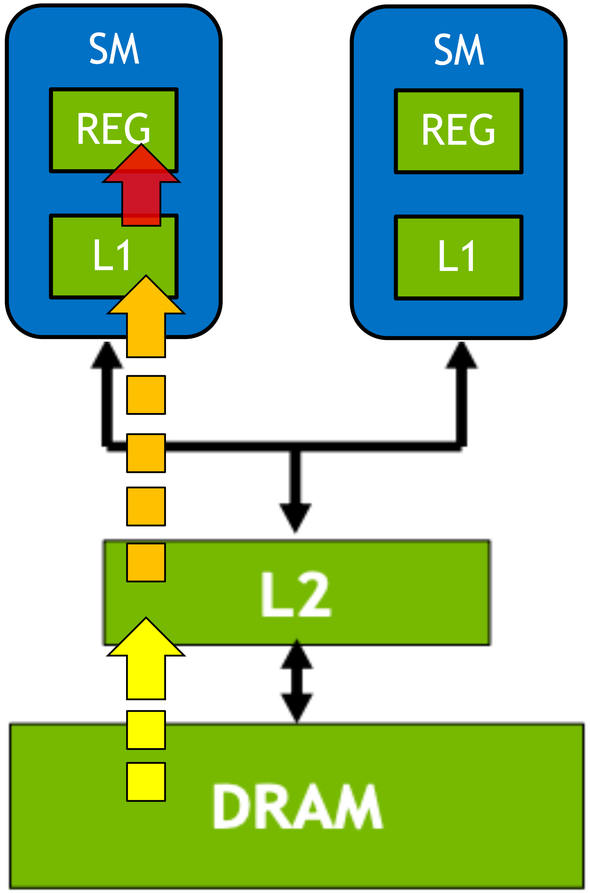
MEMORY READS

Getting data from Global Memory

Check if the data is in L1 (if not, check L2)

Check if the data is in L2 (if not, get from DRAM)

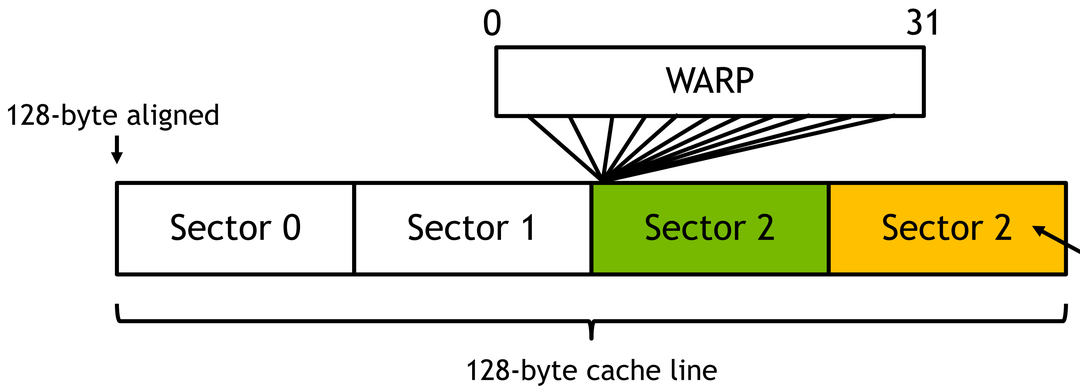
Unit of data moved: **Sectors**

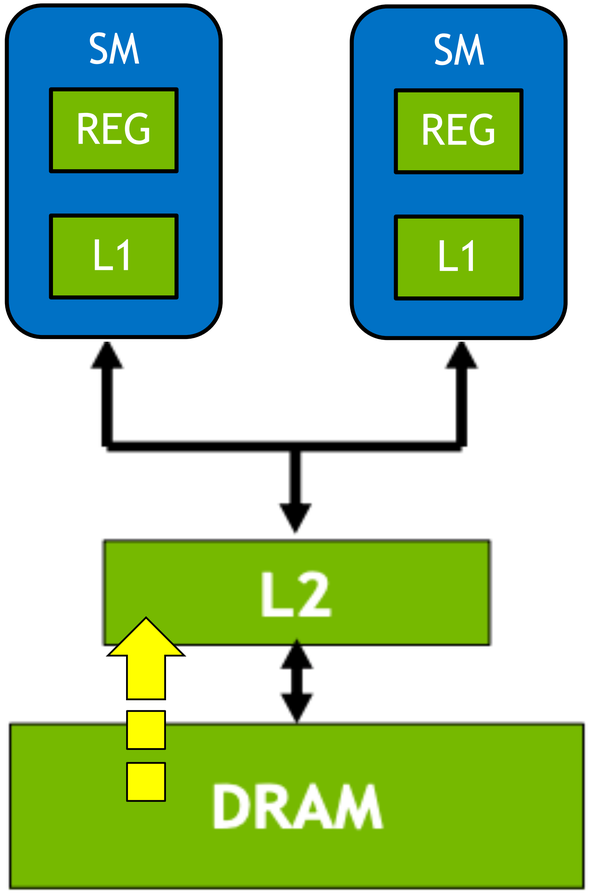


MEMORY READS

Moving Sectors from DRAM

- Since V100: default transaction size from DRAM -> L2 = 64 bytes = 2 sectors.
  - A100 granularity can be set to 32, 64, or 128 bytes
- Random access might prefer smaller granularity (minimize over fetch)
- Larger granularity can act as prefetch
  - E.g cudaDeviceSetLimit(cudaLimitMaxL2FetchGranularity, 32)



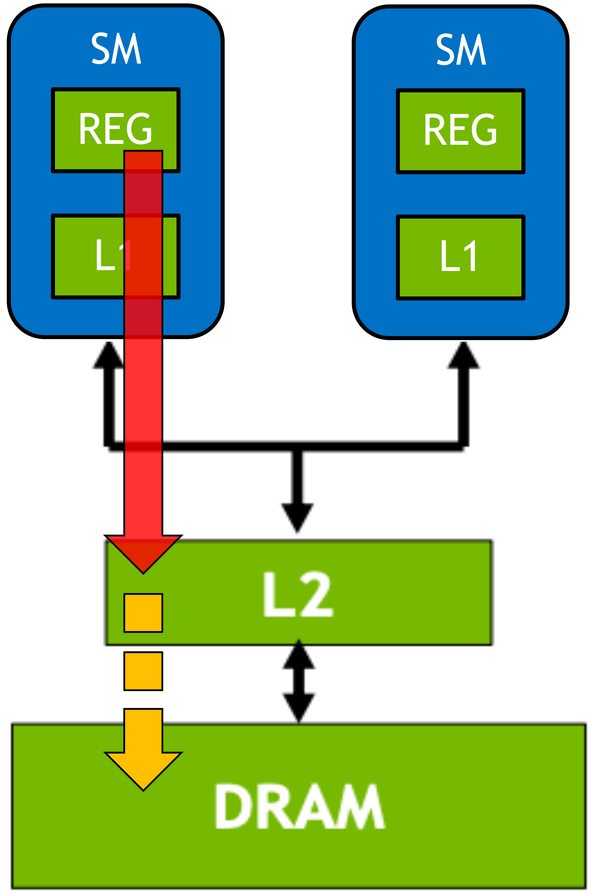


64B granularity:
“prefetched” sector

MEMORY WRITES

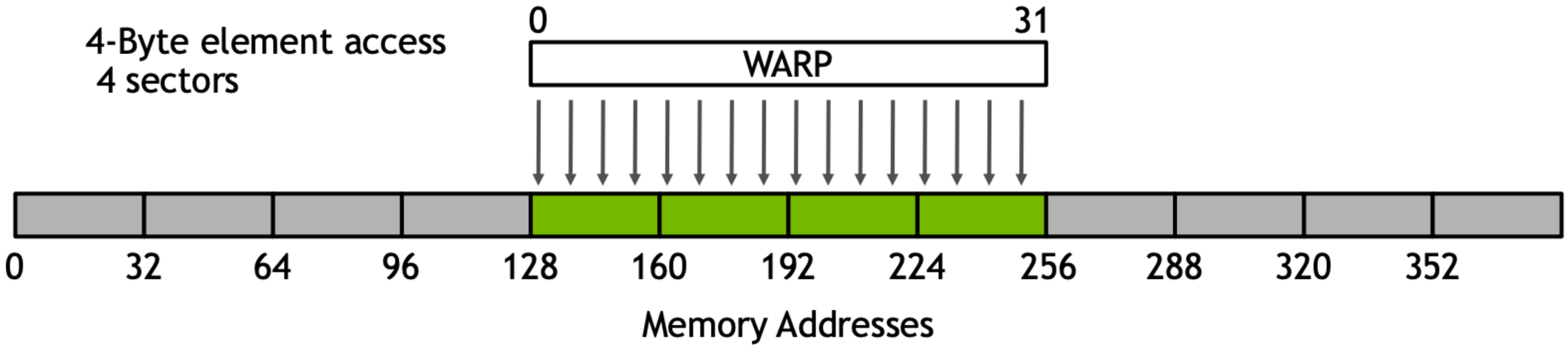
Writing data to Global Memory

- L1 is write-through
  - All writes to L1 will “write-through” to L2 immediately.
  - Before Volta: Writes were not cached in L1.
- L2 is write-back
  - Writes to L2 will be flushed to DRAM only when Needed.
  - Writes will always reach at least L2, Read After Write can hit in L2 (e.g register spills).



ACCESS PATTERNS (BEST CASE)

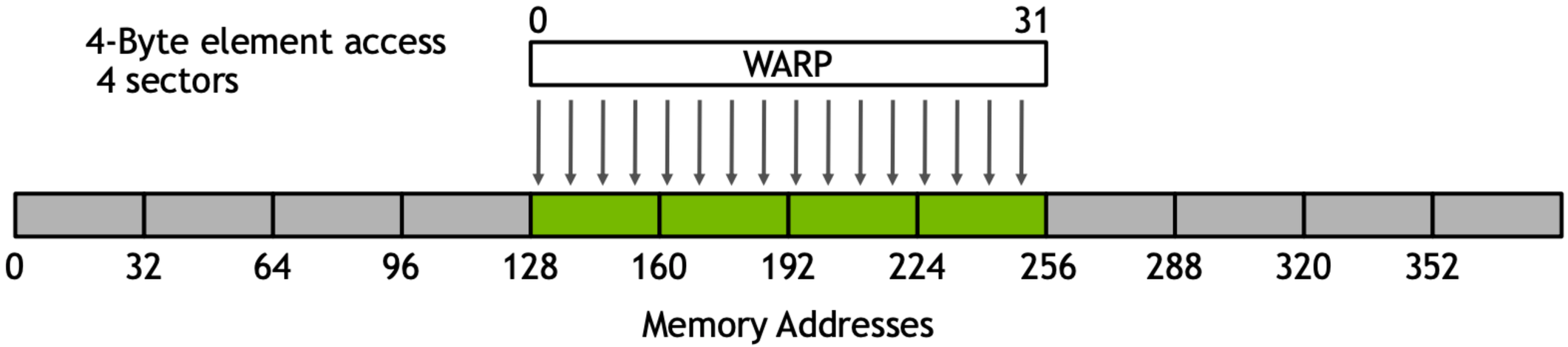
Mental Model for Profiling



128 bytes requested, 128 bytes transferred (**100% bus utilization**)

ACCESS PATTERNS (BEST CASE)

Mental Model for Profiling



128 bytes requested, 128 bytes transferred (**100% bus utilization**)

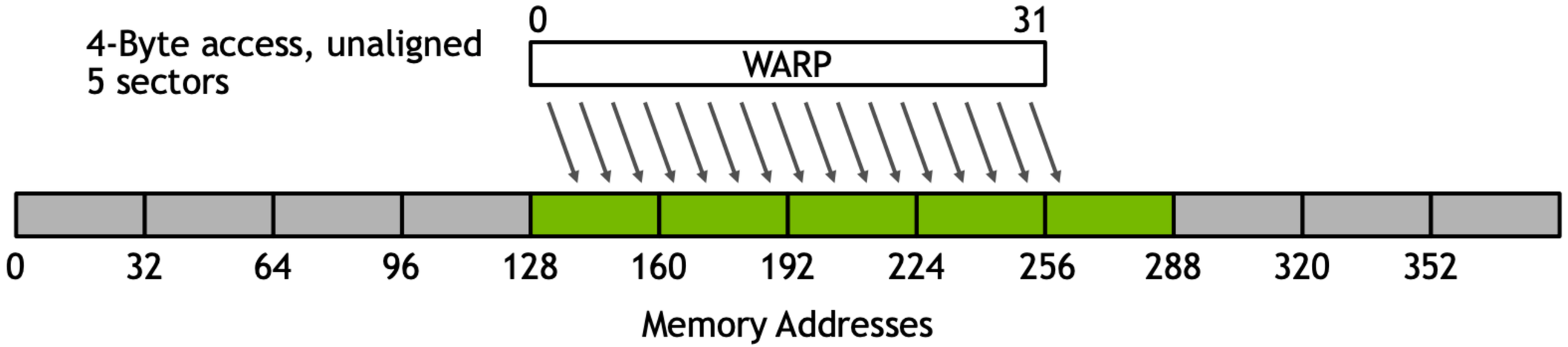
Metrics (summed):

L1_requests	1
L1_sectors	4
L2_requests	1
L2_sectors	4
L2_bytes_transferred	128
L1_bytes_transferred	128

Assumes data not found in L1

ACCESS PATTERN (UNALIGNED CASE)

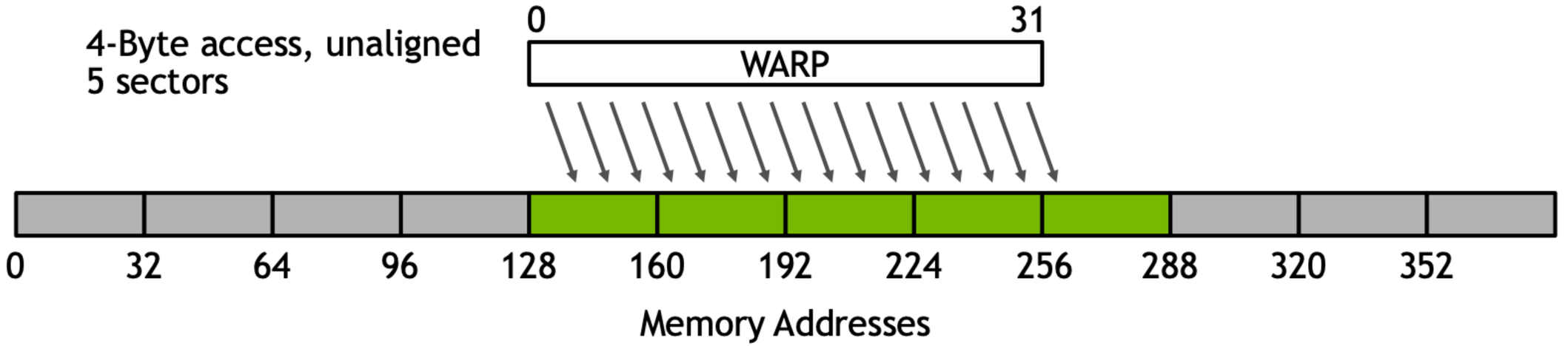
Mental Model for Profiling



128 bytes requested, 160 bytes transferred (**80% bus utilization**)

ACCESS PATTERN (UNALIGNED CASE)

Mental Model for Profiling



128 bytes requested, 160 bytes transferred (**80% bus utilization**)

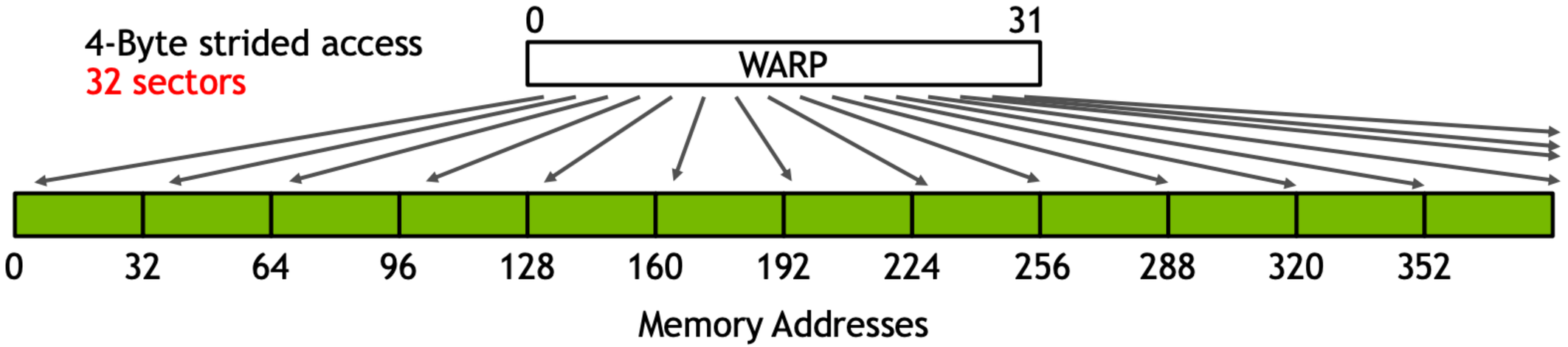
Metrics (summed):

L1_requests	1
L1_sectors	5
L2_requests	2
L2_sectors	5
L2_bytes_transferred	160
L1_bytes_transferred	160

Assumes data not found in L1

ACCESS PATTERN (WORST CASE)

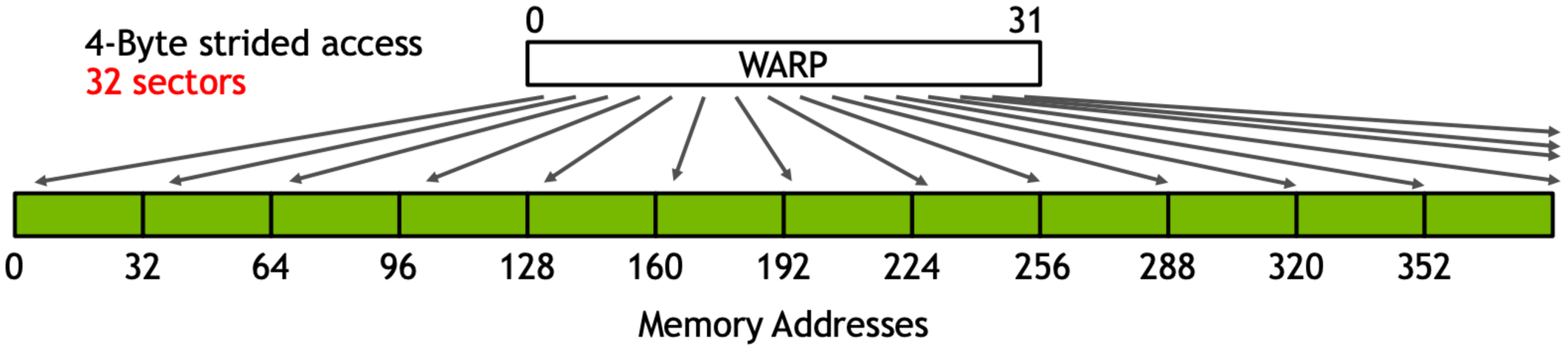
Mental Model for Profiling



128 bytes requested, 1024 bytes transferred (**12.5% bus utilization – 8X THE DATA TRANSFERRED**)

ACCESS PATTERN (WORST CASE)

Mental Model for Profiling



128 bytes requested, 1024 bytes transferred (**12.5% bus utilization – 8X THE DATA TRANSFERRED**)

Metrics (summed):

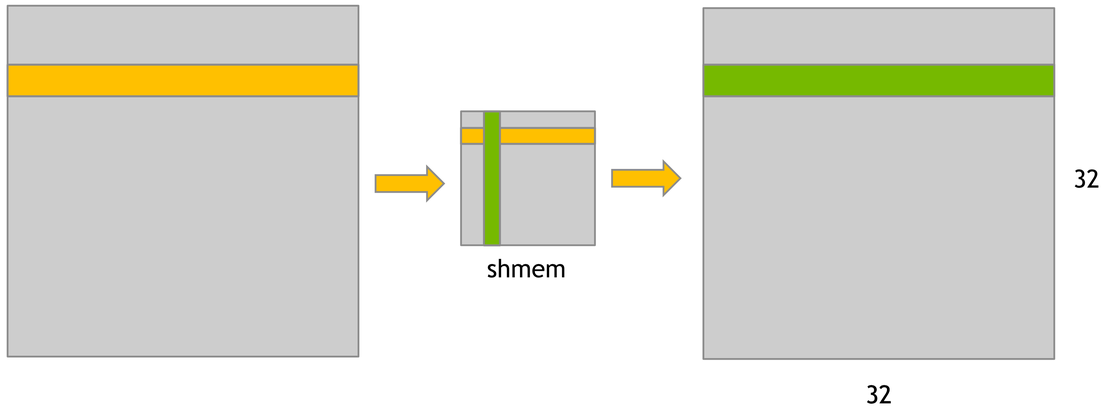
L1_requests	1
L1_sectors	32
L2_requests	8
L2_sectors	32
L2_bytes_transferred	1024
L1_bytes_transferred	1024

Assumes data not found in L1

FIX: COALESCE WRITES

Using shared memory

```
#define DIM 32

__global__ void transposeCoalesced(float* idata,
                                   float* odata)
{
  int matrix_id = blockIdx.x;
  int row = threadIdx.y;
  int col = threadIdx.x;

  __shared__ float* s_mat[DIM][DIM];

  float* imat = &idata[matrix_id * DIM * DIM];
  float* omat = &odata[matrix_id * DIM * DIM];

  s_mat[row][col] = imat[row * DIM + col];
  __syncthreads();

  omat[row * DIM + col] = s_mat[col][row];
}
```

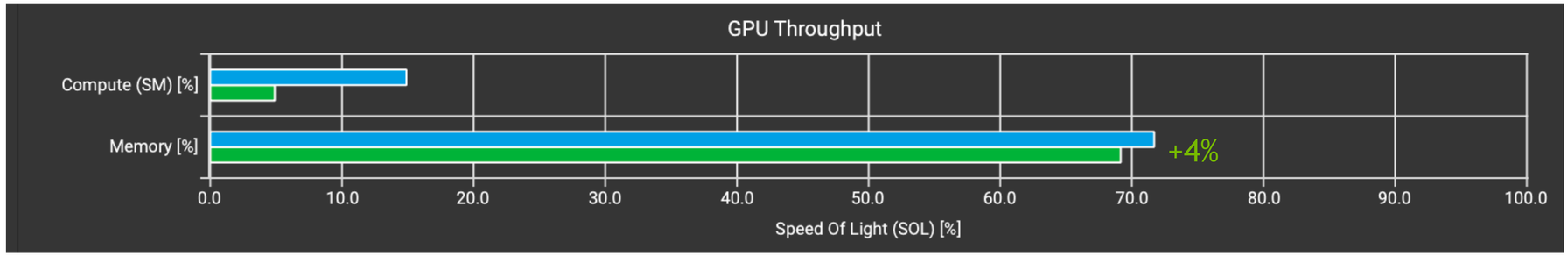


	Effective Bandwidth (GB/s)	Speedup
copy	580 (90% peak)	
transposeNaive	149	1.00x
transposeCoalesced	457	3.07x

FIX: COALESCE WRITES

Profile Analysis

Current	616 - transposeCoalesced (100000, 1, 1)x(32, 32,...	1.78 msecond	2,149,158	16	0 - NVIDIA TITAN V	1.20 cycle/nsecond	7.0	[49943] demo
Baseline 1	611 - transposeNaive (100000, 1, 1)x(32, 32, ...	5.46 msecond	6,587,245	16	0 - NVIDIA TITAN V	1.21 cycle/nsecond	7.0	[49943] demo



Before

L2 Cache

	Requests	Sectors	Sectors/Req	% Peak	Hit Rate	Bytes
L1/TEX Load	3200000	12800000	4	4.26	0	409600000
L1/TEX Store	102400000	102400000	1	34.09	100	3276800000
L1/TEX Atomic ALU	0	0	0	0	0	0

After

-32x +4x -8x

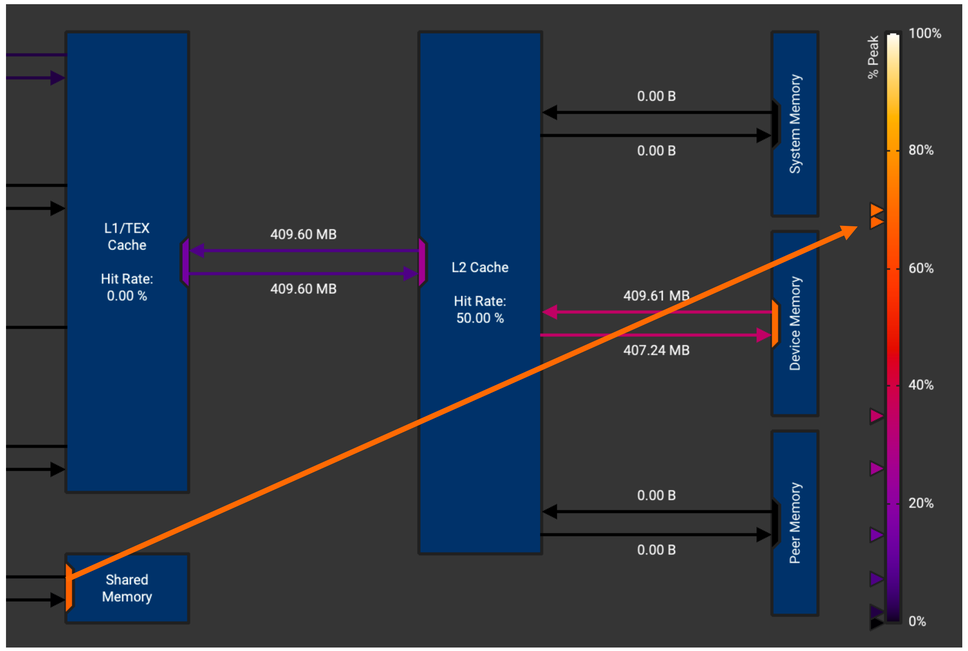
L2 Cache

	Requests	Sectors	Sectors/Req	% Peak	Hit Rate	Bytes
L1/TEX Load	3200000	12800000	4	13.13	0	409600000
L1/TEX Store	3200000	12800000	4	13.13	100	409600000
L1/TEX Atomic ALU	0	0	0	0	0	0

STILL NOT PEAK, LOOK FOR MORE INDICATORS

32-way bank conflicts

- High shared memory peak utilization
  - *Should* be read/writing same amount as DRAM
- 100,000,000 bank conflicts
  - 3,200,000 Requests:
    100,000,000 / 3,200,000 = 32 conflicts per request



Shared Memory

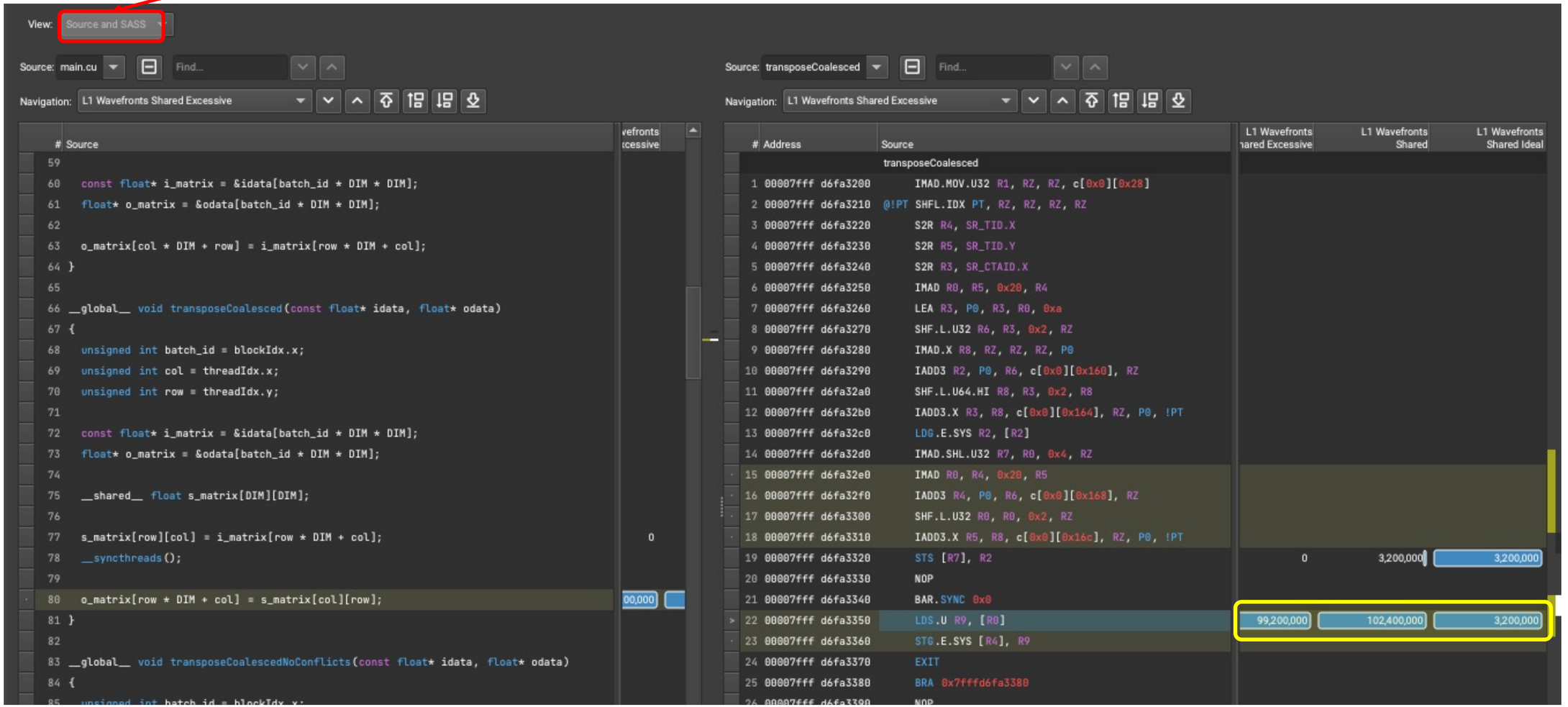
	Instructions	Requests	Wavefronts	% Peak	Bank Conflicts
Shared Load	3200000	3200000	~~109737397~~	~~60.52~~	100513901
Shared Store	3200000	3200000	3200000	1.87	0
Shared Atomic	0	0	0	0	0
Other	-	-	9702336	5.66	765
Total	6400000	6400000	116639733	68.05	100514666

32x

SOURCE CODE CORRELATION

32-way bank conflicts

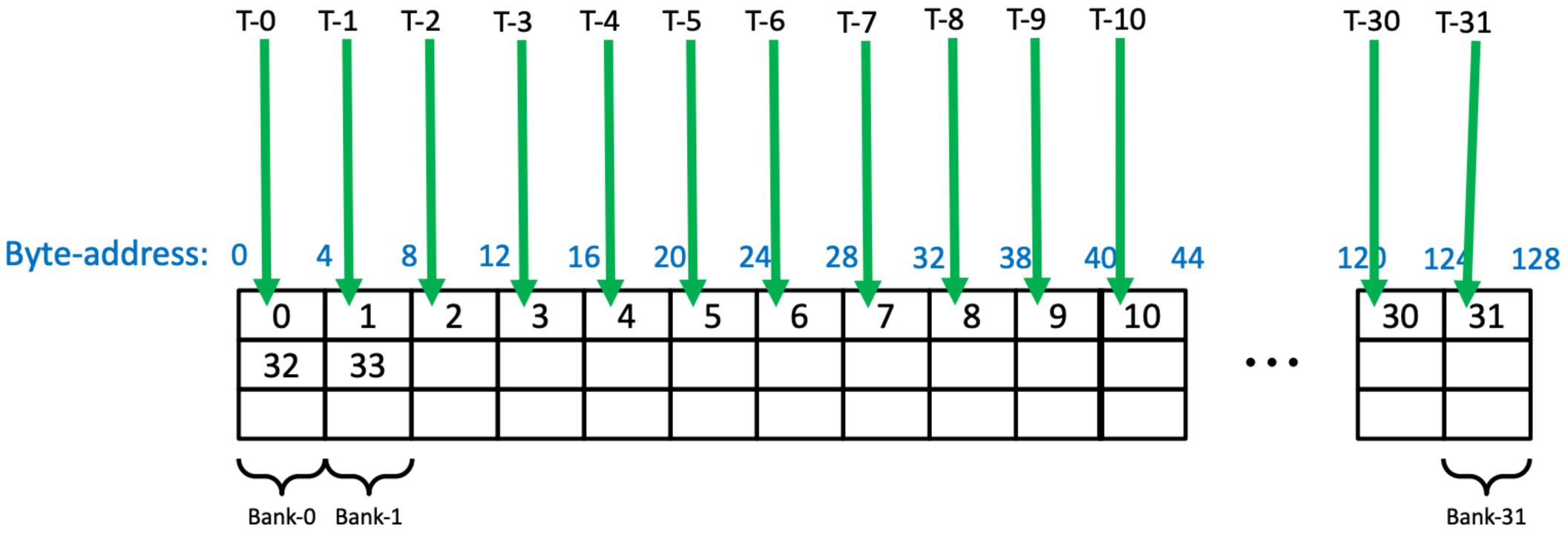
Switch page to “Source”



SHARED MEMORY: NO BANK CONFLICTS

Mental Model for Profiling

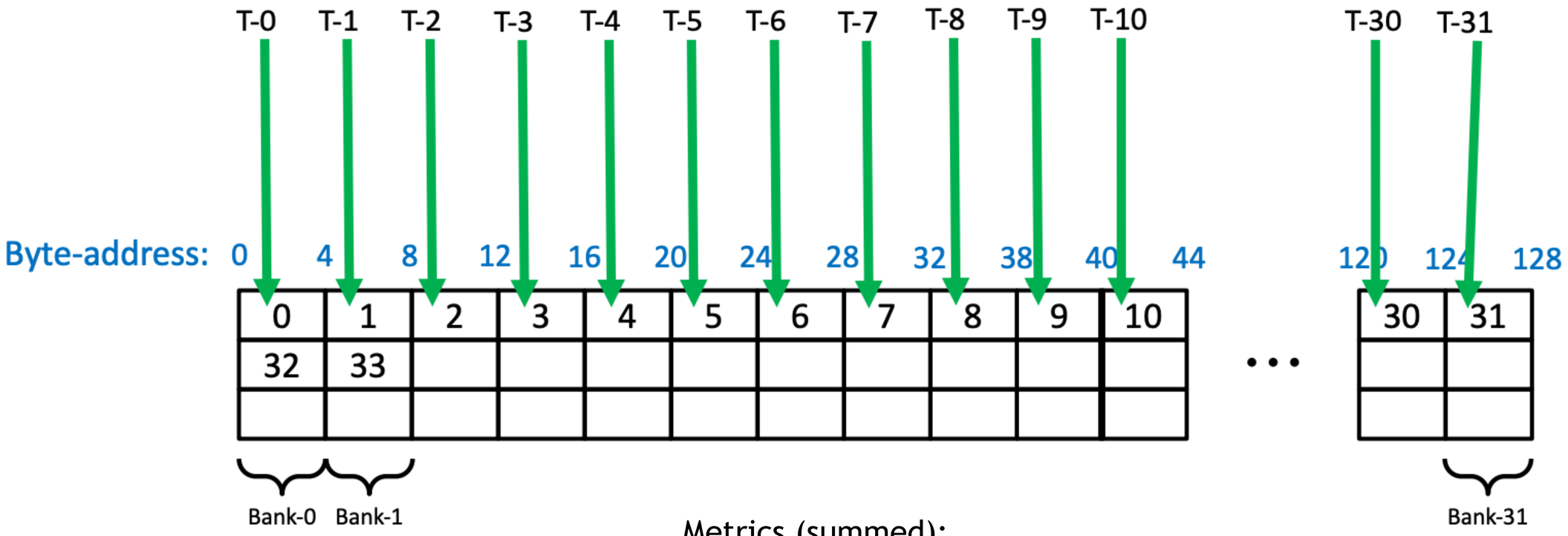
Shared memory divided into 32 banks, bandwidth = 32-bits/bank/cycle



SHARED MEMORY: NO BANK CONFLICTS

Mental Model for Profiling

Shared memory divided into 32 banks, bandwidth = 32-bits/bank/cycle



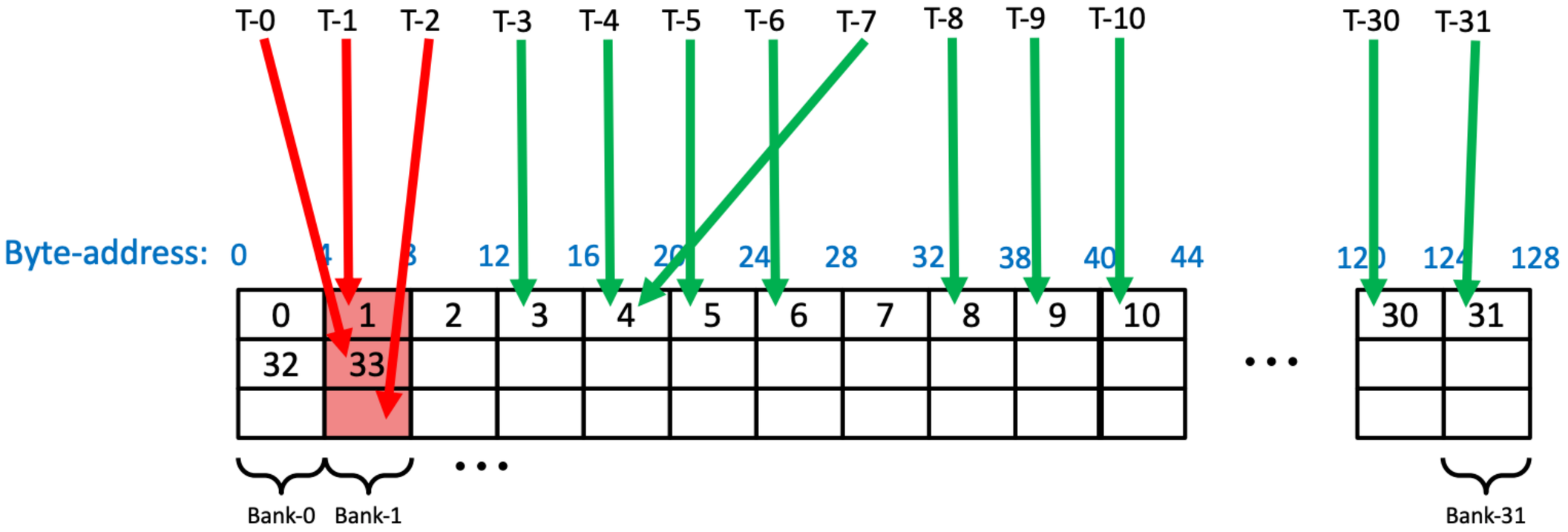
Metrics (summed):

```
memory_l1_wavefronts_shared_ideal       1
memory_l1_wavefronts_shared             1
memory_l1_wavefronts_shared_excessive   0
```

SHARED MEMORY: 3-WAY BANK CONFLICT

Mental Model for Profiling

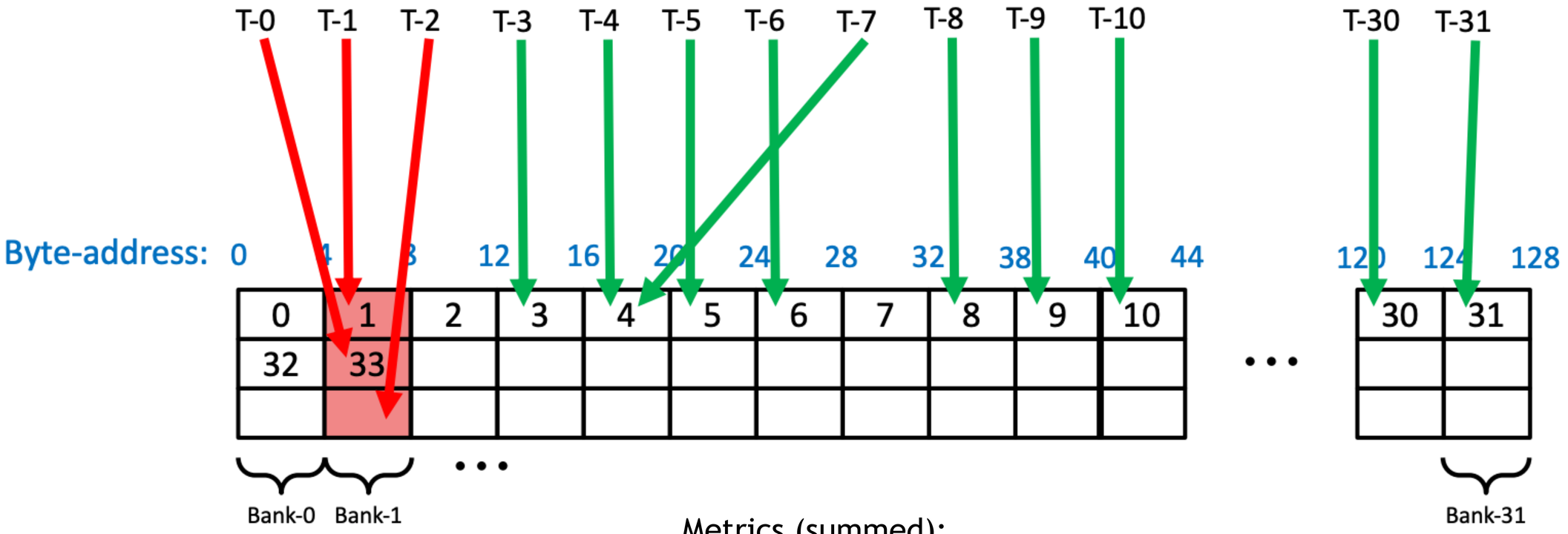
Shared memory divided into 32 banks, bandwidth = 32-bits/bank/cycle, conflicts serialize requests



SHARED MEMORY: 3-WAY BANK CONFLICT

Mental Model for Profiling

Shared memory divided into 32 banks, bandwidth = 32-bits/bank/cycle, conflicts serialize requests



Metrics (summed):

```
memory_l1_wavefronts_shared_ideal       1
memory_l1_wavefronts_shared             3
memory_l1_wavefronts_shared_excessive   2
```

MATRIX TRANSPOSE

Bank Conflicts

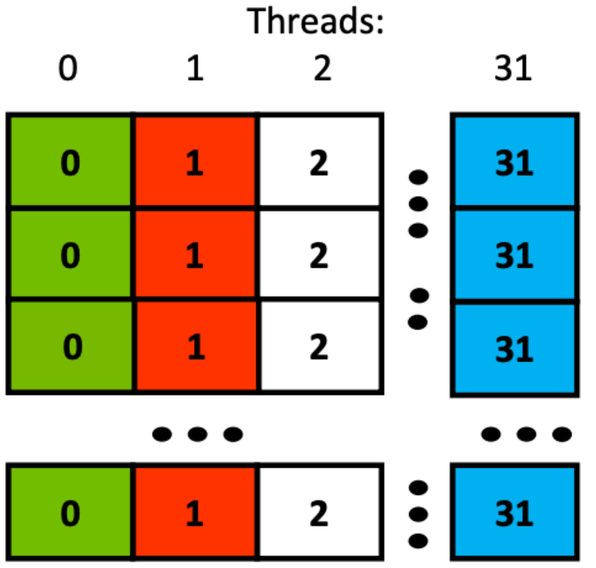
32x32 shared memory array (e.g __shared__ float sm[32][32])

Warp accesses a row: **No conflict**

Warp accesses a column: **32-way conflict**

Number indentifies which warp is accessing data
Color indicates in which bank data resides

Bank 0
Bank 1
...
Bank 31



MATRIX TRANSPOSE

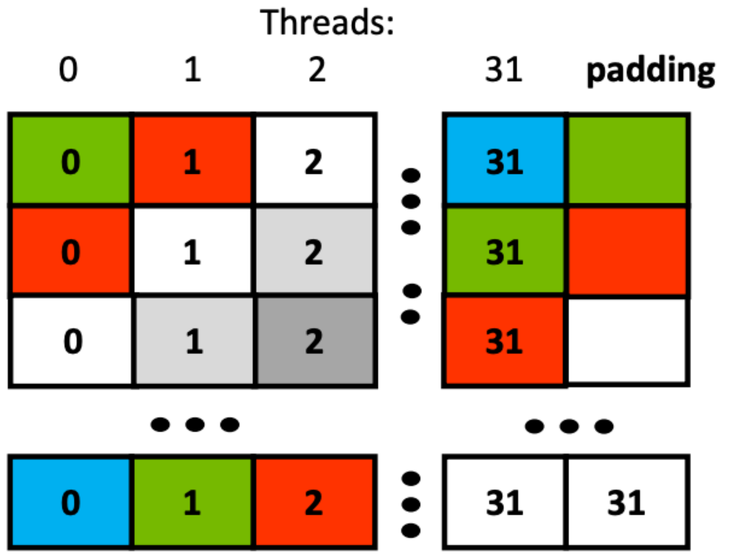
No Bank Conflicts

Solution: Add a column for **padding**: 32x33
(e.g. __shared__ float sm[32][33]

Warp accesses a column: **no conflict**

Number indentifies which warp is accessing data
Color indicates in which bank data resides

Bank 0
Bank 1
...
Bank 31



NO BANK CONFLICTS

Read column using all 32 banks

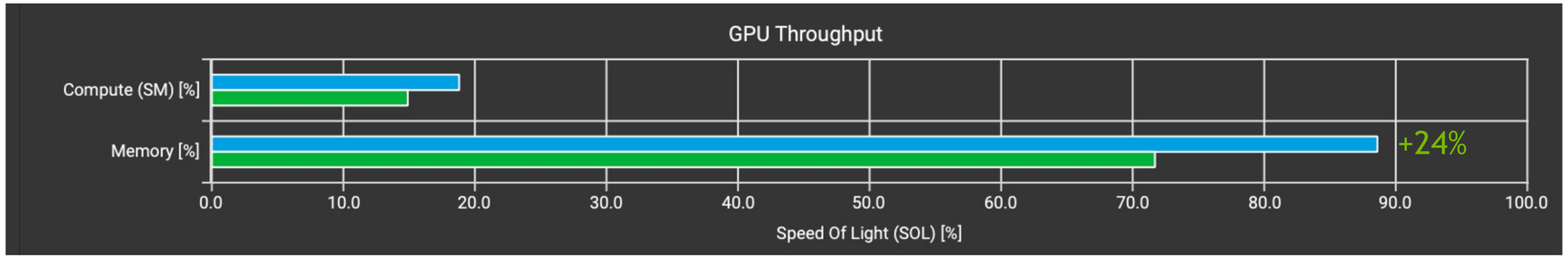
```
#define DIM 32

__global__ void transposeNoBankConflicts(
  float* idata, float* odata)
{
  int matrix_id = blockIdx.x;
  int row = threadIdx.y;
  int col = threadIdx.x;

   __shared__ float* s_mat[DIM][DIM + 1];

  float* imat = &idata[matrix_id * DIM * DIM];
  float* omat = &odata[matrix_id * DIM * DIM];

  s_mat[row][col] = imat[row * DIM + col];
  __syncthreads();

  omat[row * DIM + col] = s_mat[col][row];
}
```

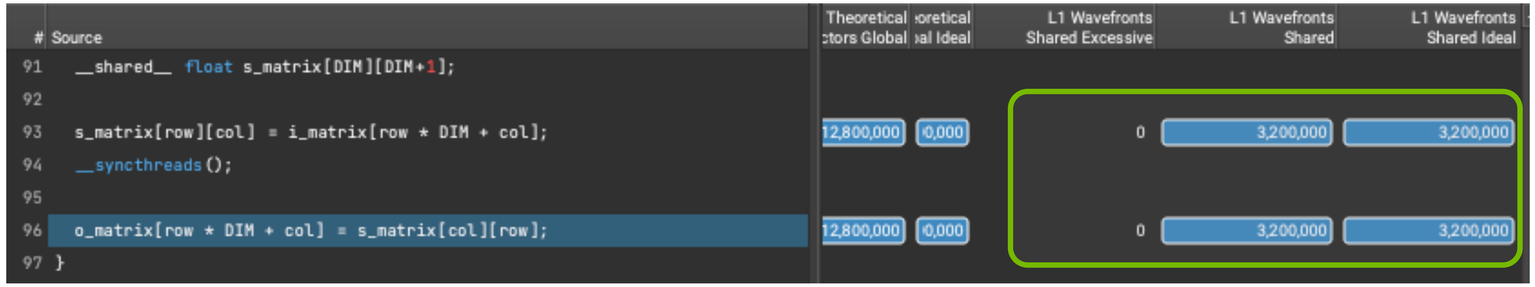
	Effective Bandwidth (GB/s)	Speedup
copy	580 (90% peak)	
transposeNaive	149	1.00x
transposeCoalesced	457	3.07x
transposeNoBankConflicts	577 (90% peak)	3.87x

NO BANK CONFLICTS

Profile Performance

Current	621 - transposeCoalescedNoConflicts (100000, 1, 1)x(...	1.42 msecond	1,702,523	16	0 - NVIDIA TITAN V	1.20 cycle/nsecond	7.0	[49943] demo
Baseline 1	616 - transposeCoalesced (100000, 1, 1)x(32, 32, ...	1.78 msecond	2,149,158	16	0 - NVIDIA TITAN V	1.20 cycle/nsecond	7.0	[49943] demo



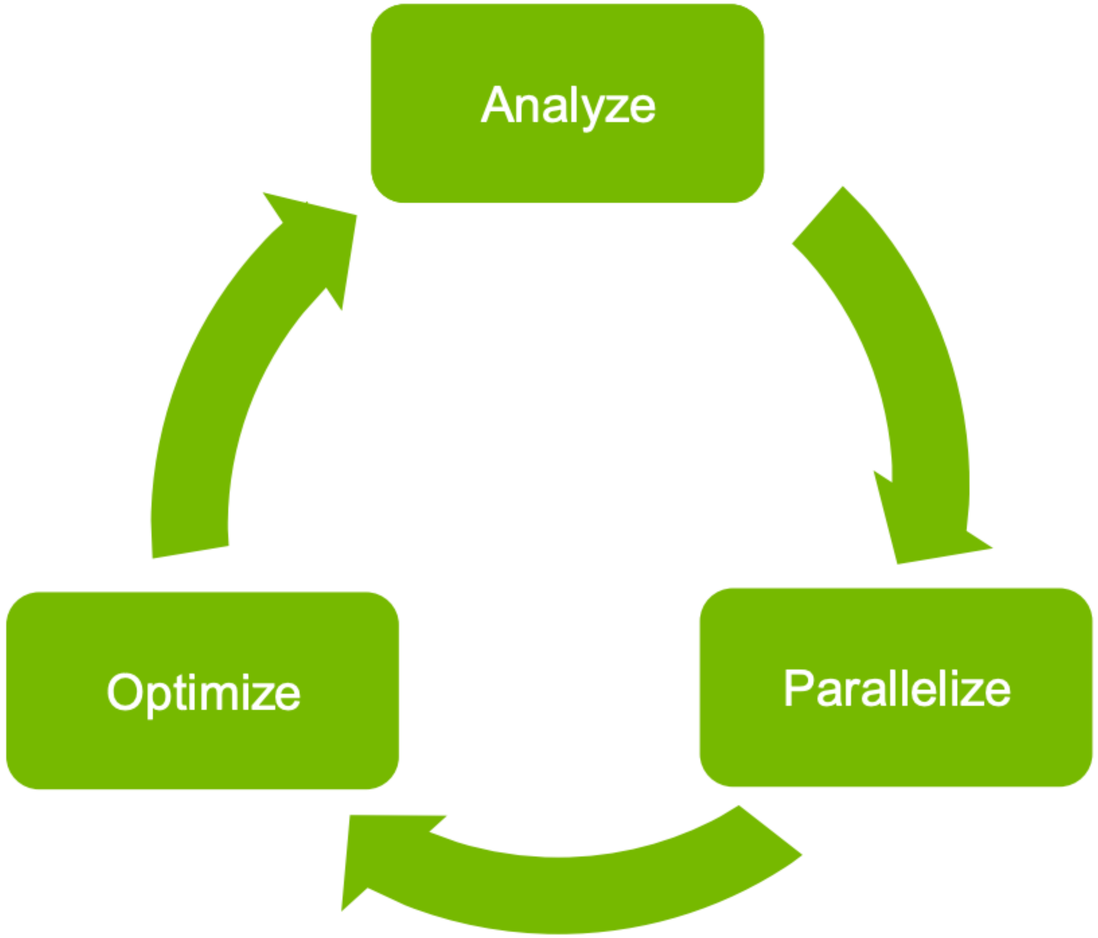


	Effective Bandwidth (GB/s)	Speedup
copy	580 (90% peak)	
transposeNaive	149	1.00x
transposeCoalesced	457	3.07x
transposeNoBankConflicts	577 (90% peak)	3.87x

SUMMARY

Optimization Process

- Trace application with **Nsight Systems** to understand system-wide hot-spots.
- Analyze CUDA kernels with **Nsight Compute**
  - Follow the warning signs! ⚠
  - Identify performance limiters
    - Memory
    - Compute
    - Latency
- Optimize your code following best practices.
- Iterate, iterate, iterate!



CUDA Documentation

Best Practices: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/

Volta Tuning Guide: https://docs.nvidia.com/cuda/volta-tuning-guide/

Ampere Tuning Guide: https://docs.nvidia.com/cuda/ampere-tuning-guide/

Nsight Tools

Nsight Systems Docs: https://docs.nvidia.com/nsight-systems/UserGuide/index.html

Nsight Systems Download: https://developer.nvidia.com/nsight-systems

Nsight Compute Docs: https://docs.nvidia.com/nsight-compute/NsightCompute/index.html

Nsight Compute Download: https://developer.nvidia.com/nsight-compute