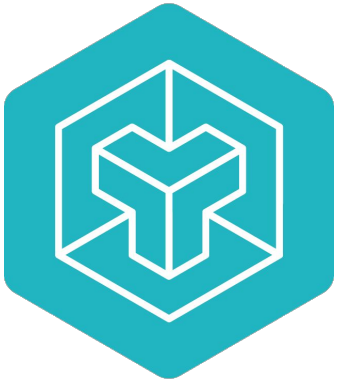
GPU Programming Fundamentals: A Hardware-First Perspective

William Brandon

August 2025

Who is this talk for?

- Anyone who might want to write a kernel at some point!
  - (In any language)



etc…

- Try to build foundations from the bottom up
- If you’re more experienced, I hope it still helps you see things a bit differently

Who am I?

Deep learning compilers

→

LLM efficiency research + creating **6.S894** (“Accelerated Computing”)

→

Making Claude faster

Who am I?

creating **6.S894** (“Accelerated Computing”)

Who am I?

Class co-created with…

Jonathan Ragan-Kelley

Nikita Lazarev

creating **6.S894** (“Accelerated Computing”)

Materials are online!

Recommended reading:

Lab 1: http://bit.ly/4lMkoBz
Lab 2: http://bit.ly/4mPB2Bu

What is a CUDA program?

```
__global__ void my_kernel(...) {
    ...
}
```

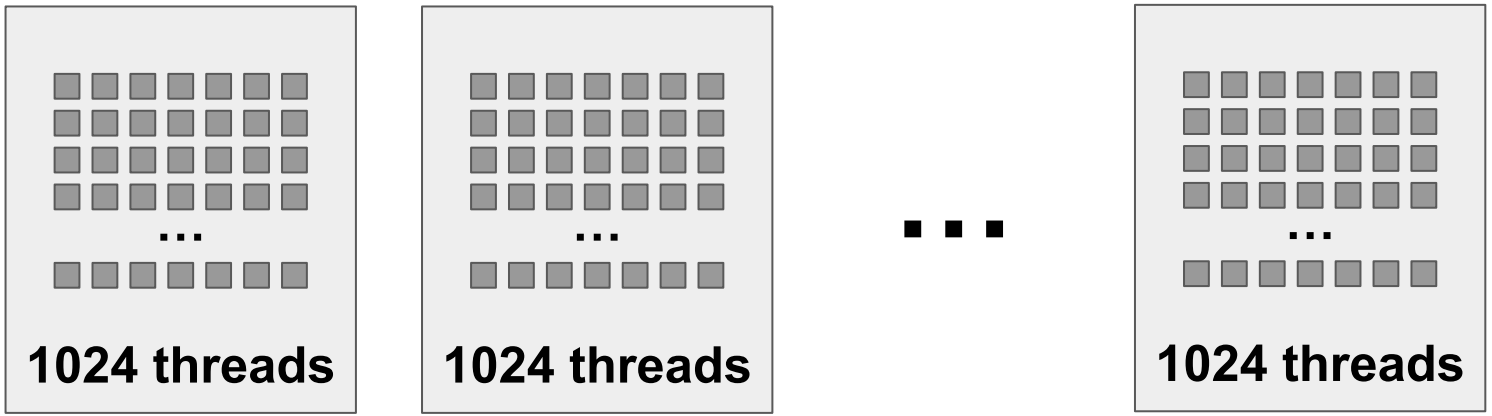
Number of “blocks”

Number of “threads per block”

```
my_kernel<<<128, 1024>>>(...);
```

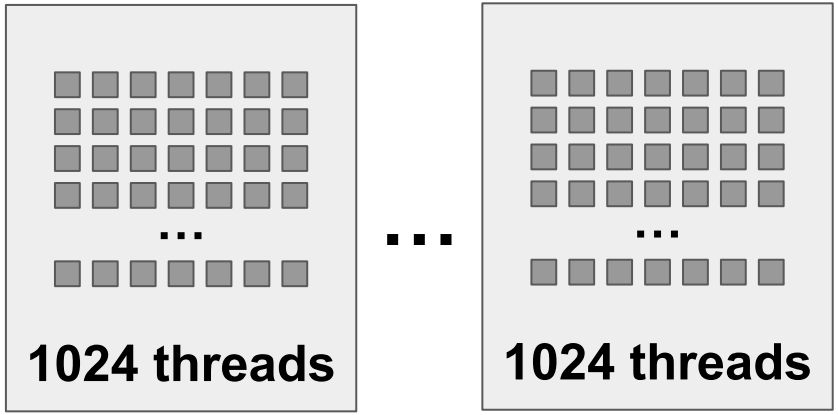
What is a CUDA program?

Your program, according to CUDA:



A Puzzle

A Puzzle

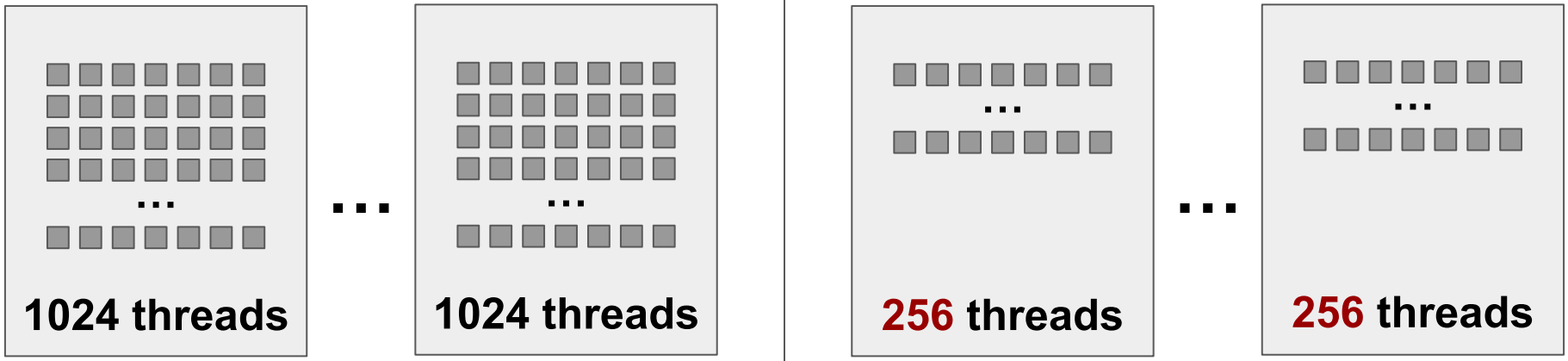


128 blocks

```
my_kernel<<<128, 1024>>>(...);
```

A Puzzle

How do these compare?



128 blocks

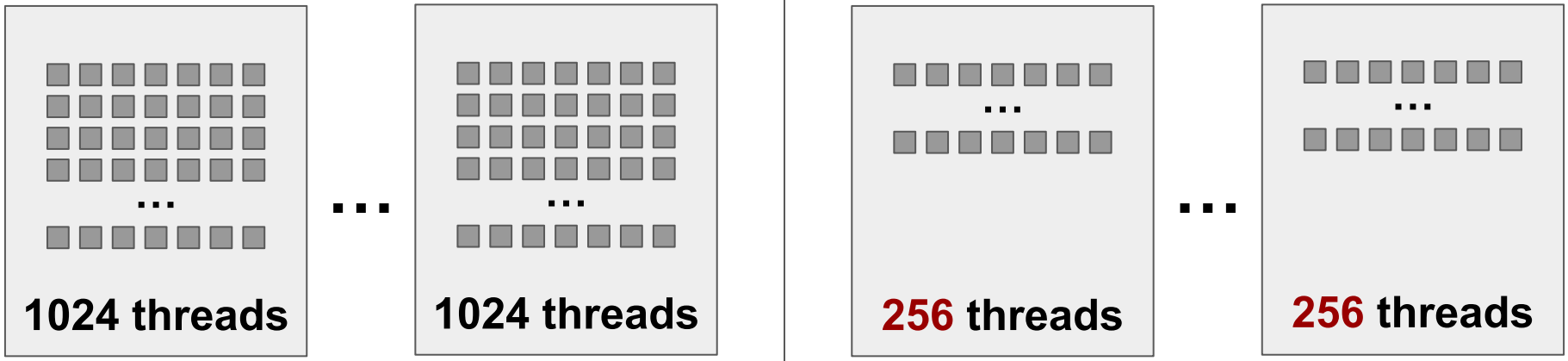
```
my_kernel<<<128, 1024>>>(...);
```

128 blocks

```
my_kernel<<<128, 256>>>(...);
```

A Puzzle

How do these compare?



128 blocks

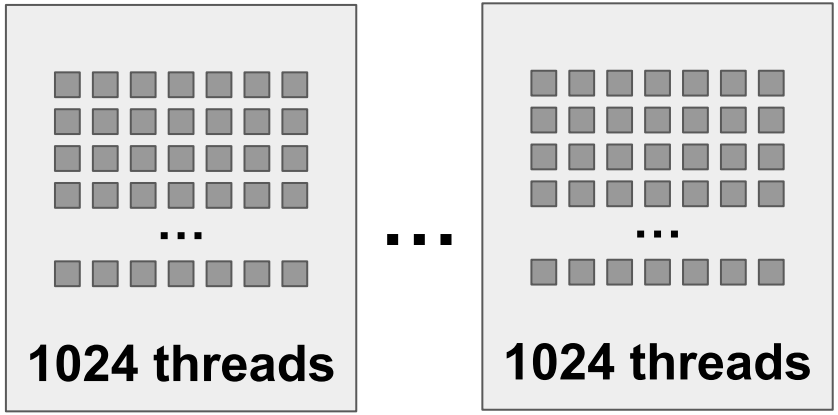
```
my_kernel<<<128, 1024>>>(...);
```

128 blocks

```
my_kernel<<<128, 256>>>(...);
```

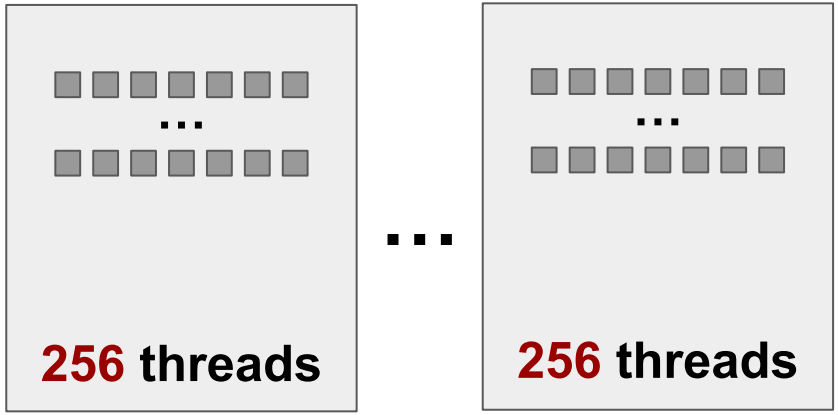
A Puzzle

Often, ~*same* speed! (why? 🤔)



128 blocks

```
my_kernel<<<128, 1024>>>(...);
```

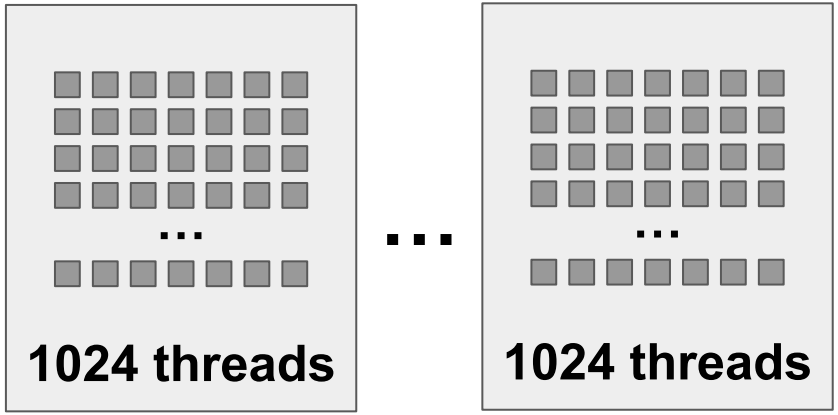


128 blocks

```
my_kernel<<<128, 256>>>(...);
```

Another Puzzle

Another Puzzle

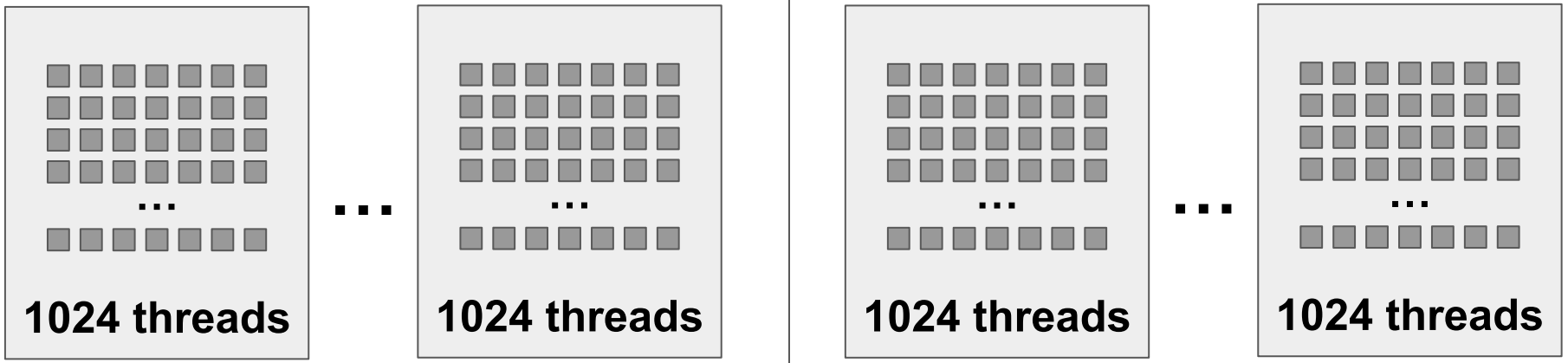


128 blocks

```
my_kernel<<<128, 1024>>>(...);
```

Another Puzzle

How do these compare?



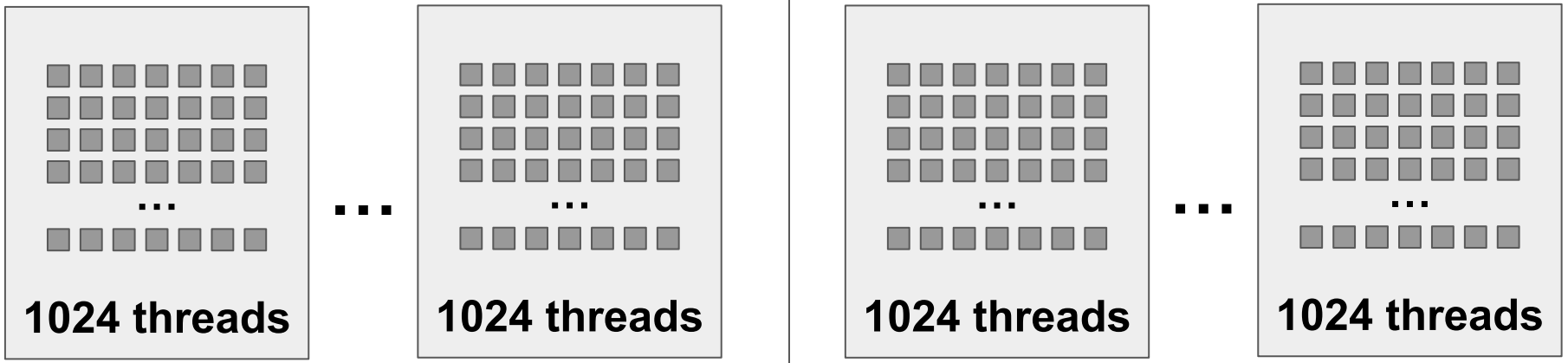
128 blocks ~4% increase **133 blocks**

```
my_kernel<<<128, 1024>>>(...);
```

```
my_kernel<<<133, 1024>>>(...);
```

Sometimes, ~*2x slower*! (why? 🤔)

Another Puzzle



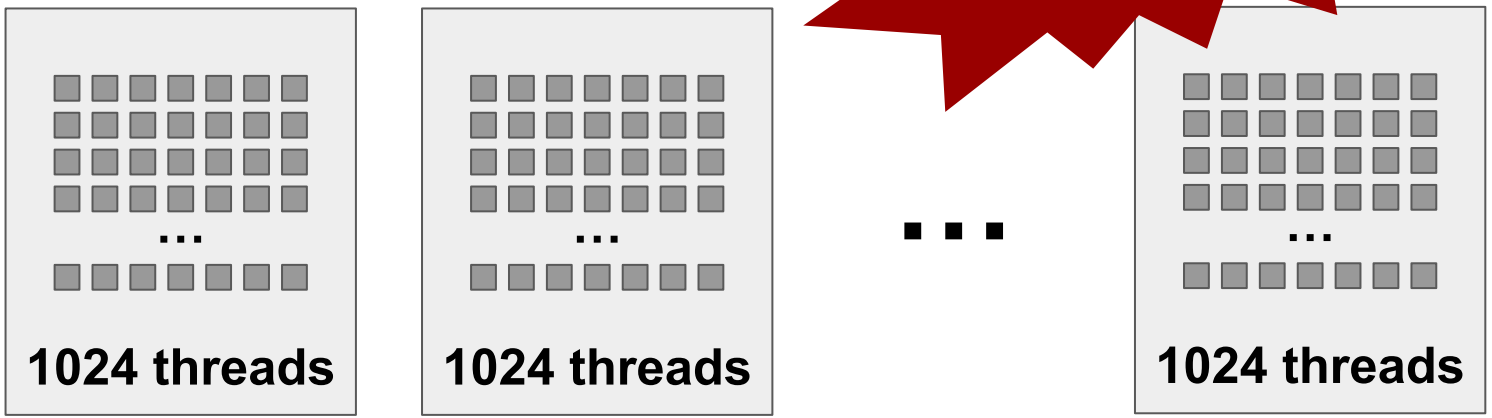
128 blocks → **~4% increase** → **133 blocks**

```
my_kernel<<<128, 1024>>>(...);
```

```
my_kernel<<<133, 1024>>>(...);
```

What is a CUDA program?

Your program, according to CUDA:



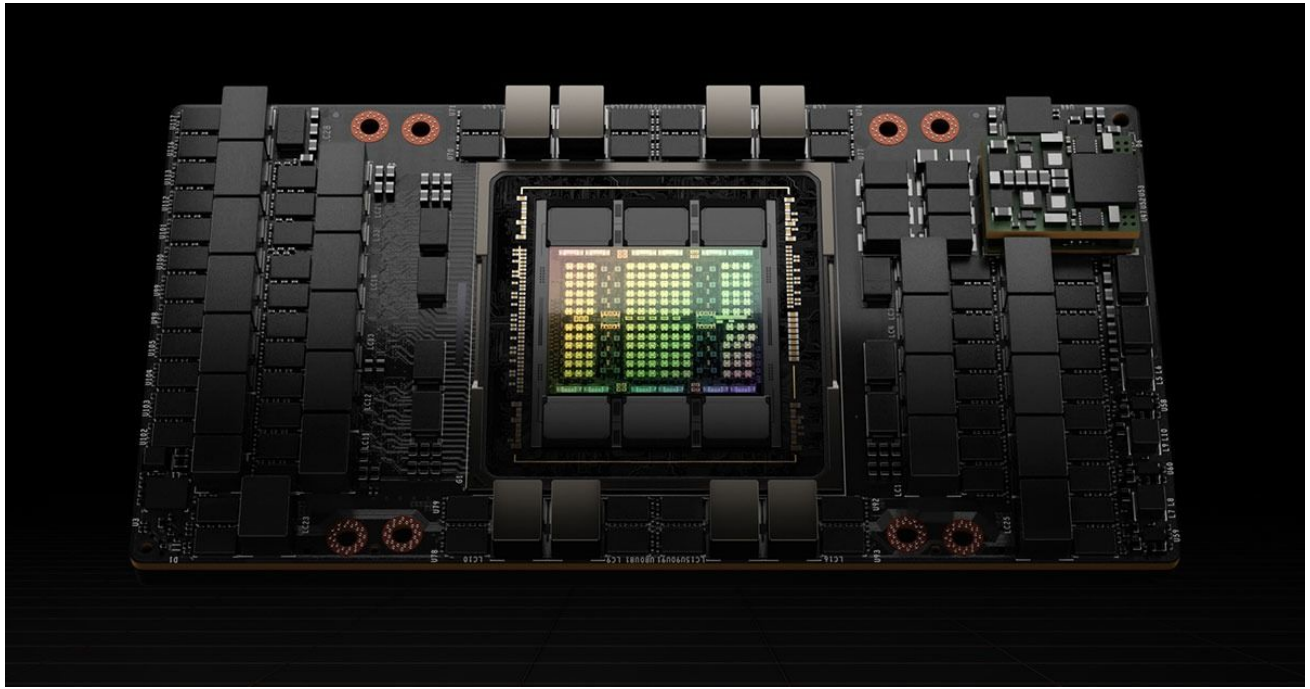
What is a CUDA program?

~~What is a CUDA program?~~

What is a GPU?

~~What is a CUDA program?~~

What is a GPU?

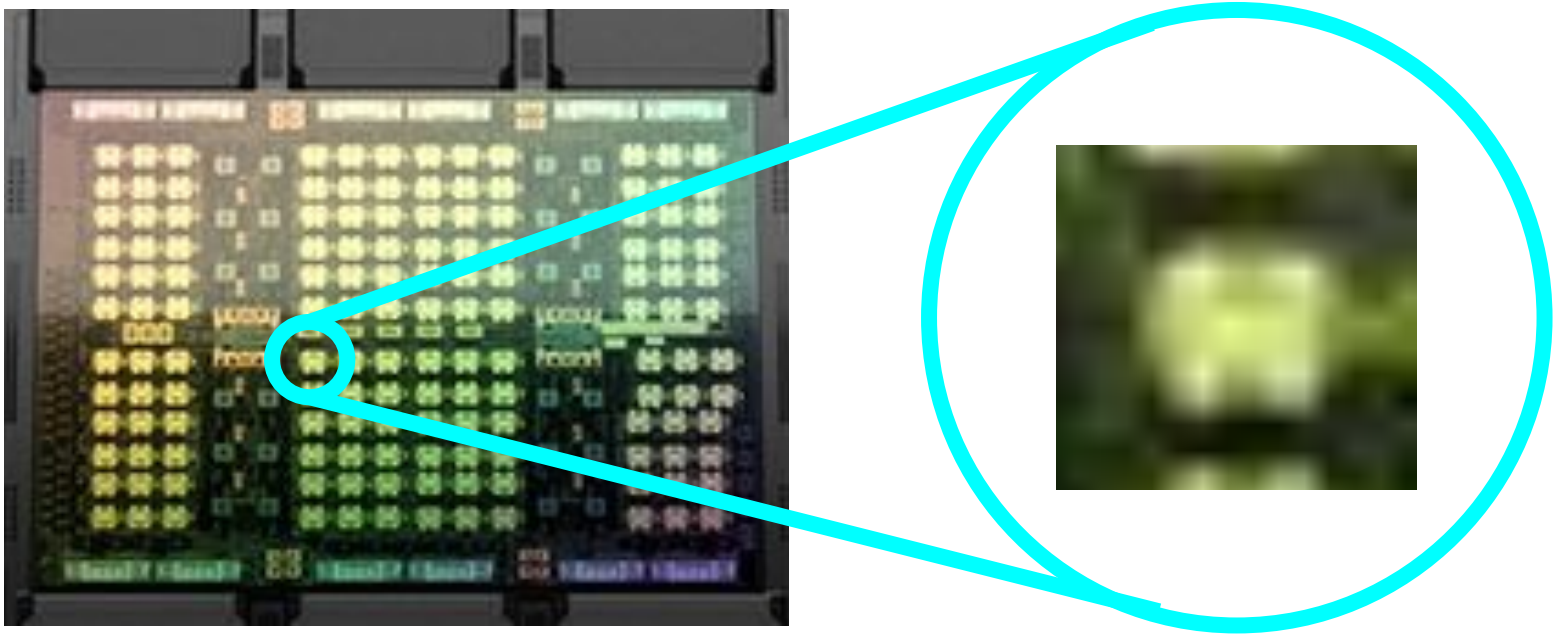


~~What is a CUDA program?~~

What is a GPU?

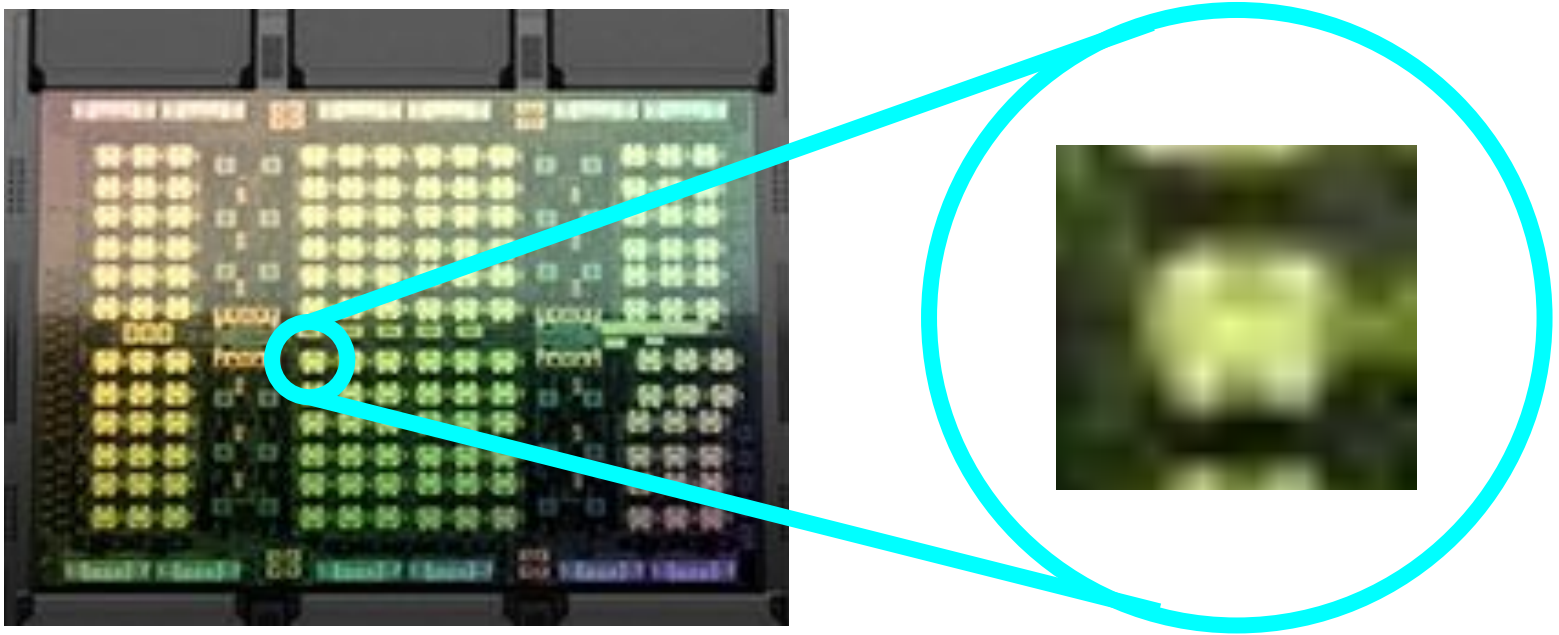
~~What is a CUDA program?~~

What is a GPU?



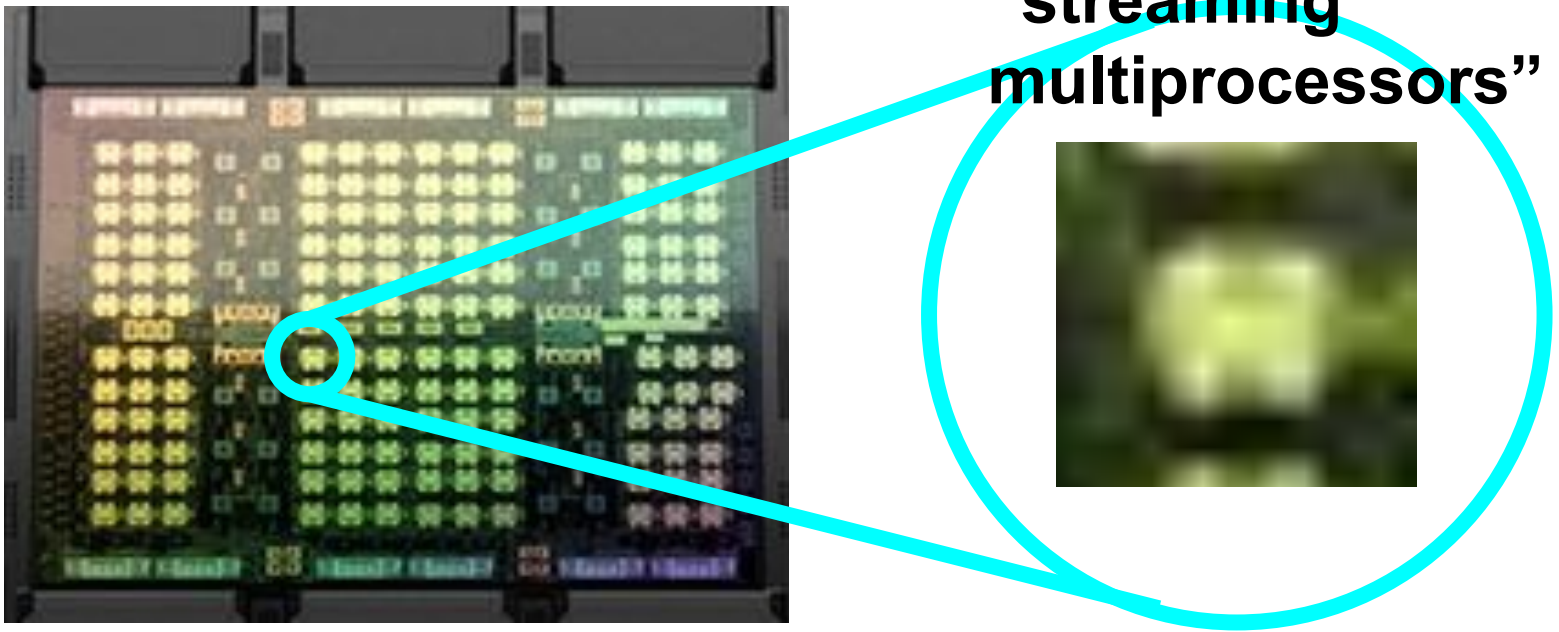
~~What is a CUDA program?~~

What is a GPU? **A GPU is a bunch of tiny squares.**



~~What is a CUDA program?~~

What is a GPU? **A GPU is a bunch of ~~tiny squares.~~ "streaming multiprocessors"**



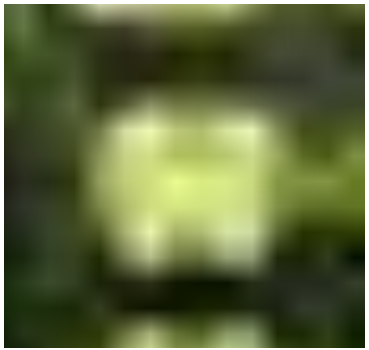
A GPU is a bunch of streaming multiprocessors.

How many? Order of magnitude: **~100**

A GPU is a bunch of streaming multiprocessors.

How many? Order of magnitude: **~100**

What is inside a streaming multiprocessor?

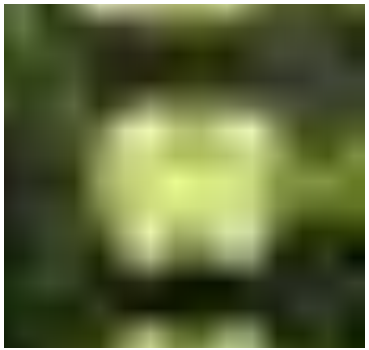


1 **SM** = 4 ***thingies***

A GPU is a bunch of streaming multiprocessors.

How many? Order of magnitude: **~100**

What's inside a streaming multiprocessor?



1 **SM** = 4 ***thingies***

("***quadrants***" or "***partitions***" or "***warp schedulers***")

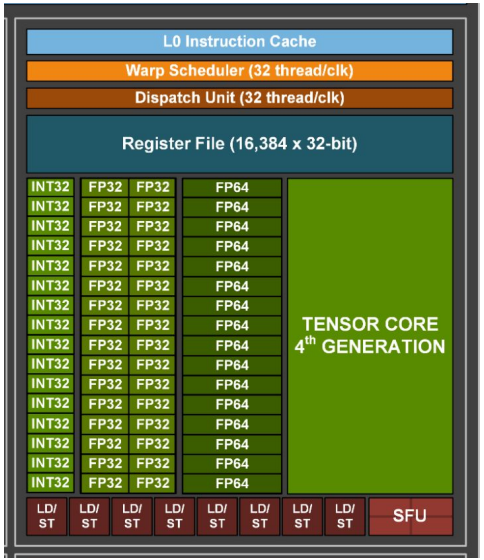
What's inside a streaming multiprocessor?

Four ***warp schedulers***

What's inside a streaming multiprocessor?

Four ***warp schedulers***

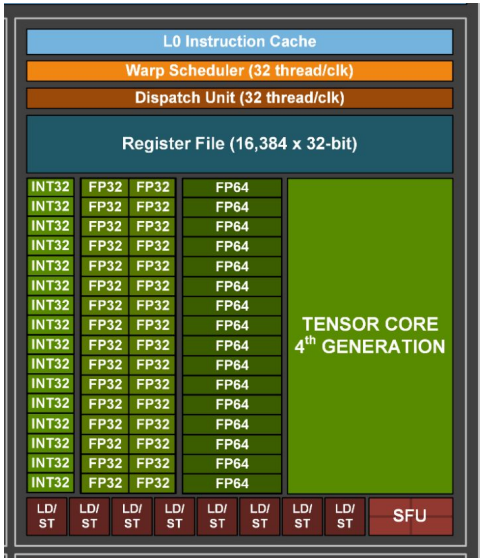
What is a warp scheduler? **It's like a CPU core.**



What's inside a streaming multiprocessor?

Four ***warp schedulers***

What is a warp scheduler? **It's like a CPU core*.**

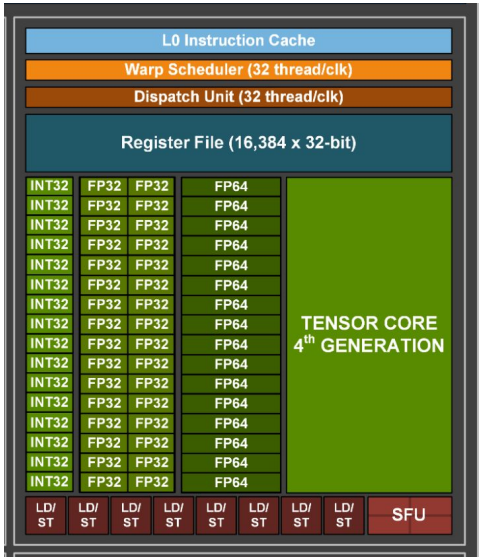


(*a CPU core specialized for running SIMD instructions)

What's inside a streaming multiprocessor?

Four ***warp schedulers***

What is a warp scheduler? **It's like a CPU core*.**

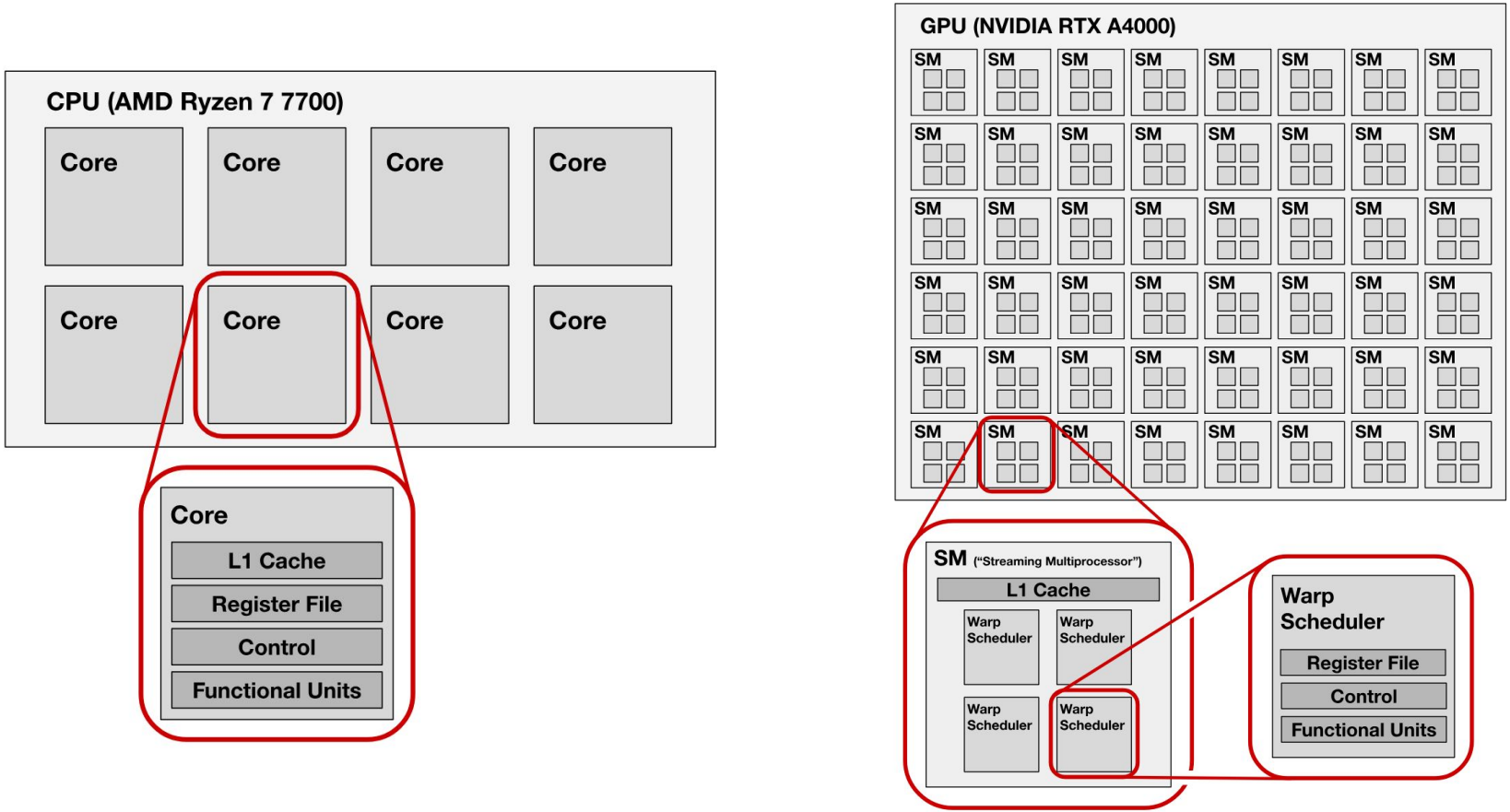


In what sense?

- Issues a stream of instructions *in sequence*
- Stores *execution state* for the running program (program counter, registers)
- Has *functional units* which can do math, talk to main memory, etc.

(*a CPU core specialized for running SIMD instructions)

What is a warp scheduler? **It's like a CPU core.**



What is a warp scheduler? **It's like a CPU core.**

Sanity check: does the math work out?

1) You may often run kernels with hundreds of thousands of threads

Example

```
kernel<<<128, 1024>>>(...);
```

128 * 1024 = 131,072

2) But a GPU only has hundreds of warp schedulers

Example

NVIDIA H100 GPU:

132 SMs * 4 = 528 warp schedulers

What gives?

What is a warp scheduler? **It's like a CPU core.**

Sanity check: does the math work out?

1) You may often run kernels with hundreds of thousands of threads

2) But a GPU only has hundreds of warp schedulers

What gives?

Three answers:

1. Warp scheduler instructions are **SIMD** (32 threads at a time)
2. Warp scheduler can **time multiplex** between different threads
3. Too many blocks → run them **serially**

Warp scheduler instructions are (implicitly) **SIMD**

CPU – explicit SIMD

```
fp32x16 x = {...};
fp32x16 y = {...};
fp32x16 z =
  vector_add_fp32x16(x, y);
```

Single instruction issued
→ 16 scalar additions in parallel

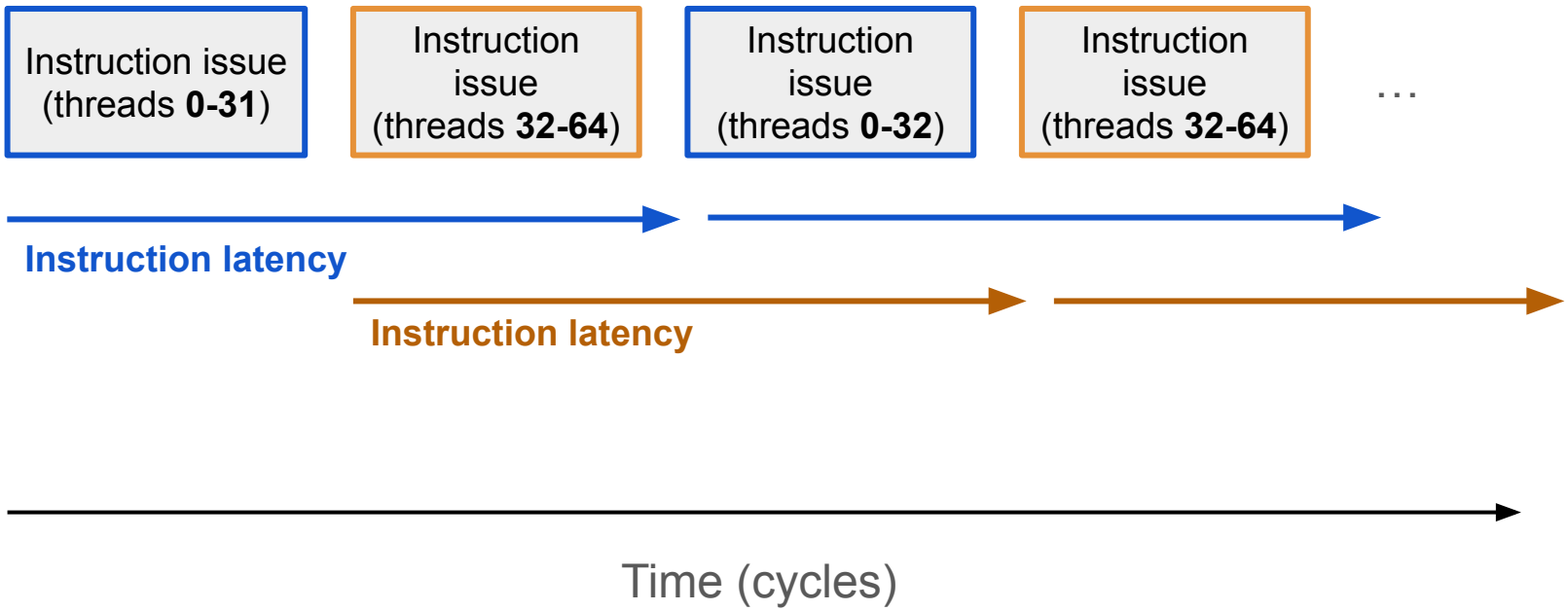
GPU – implicit SIMD

```
float x = ...;
float y = ...;
float z = x + y;
```

Single instruction issued
→ 32 scalar additions in parallel

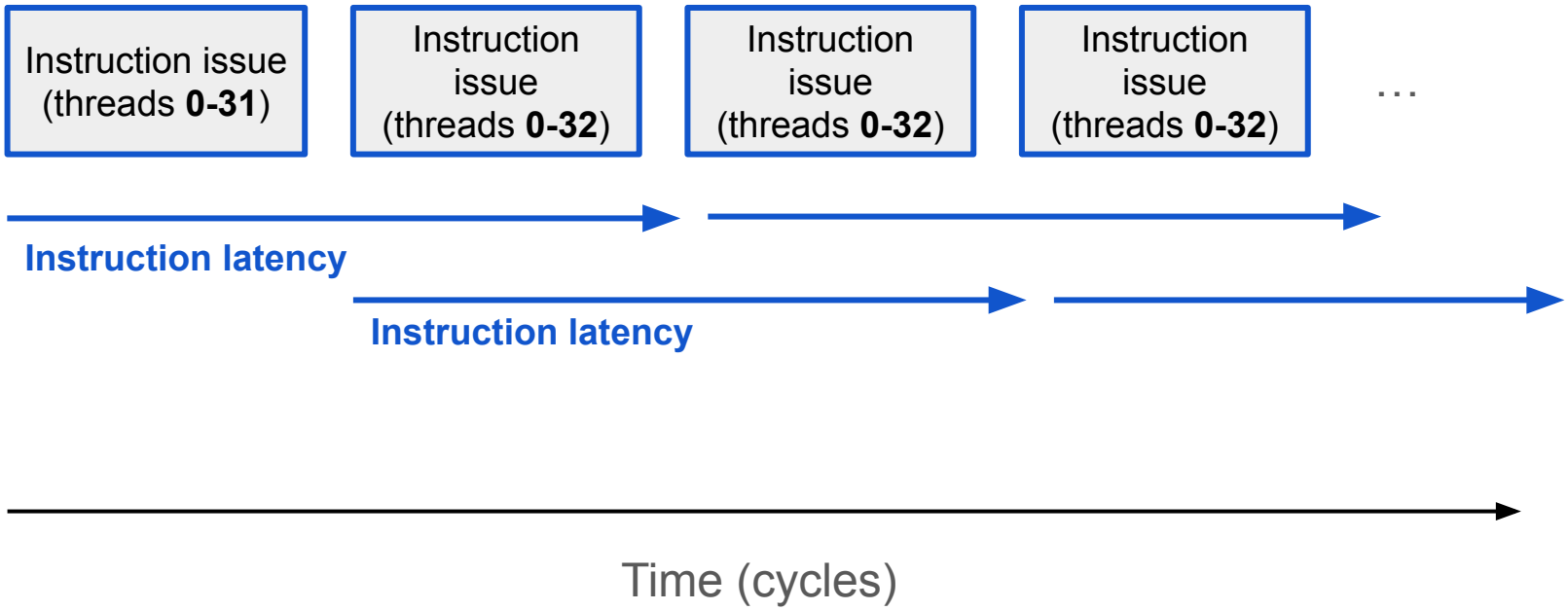
(serving 32 consecutive threads)

Warp scheduler can **time multiplex** between different threads



(CPU cores can do this too!)

Can also overlap instructions *without* multiple threads (if the pattern is right)



(CPU cores can do this too!)

How many threads do you need to keep an SM busy?
(at minimum)

4 warp schedulers * 32 = **128 threads**

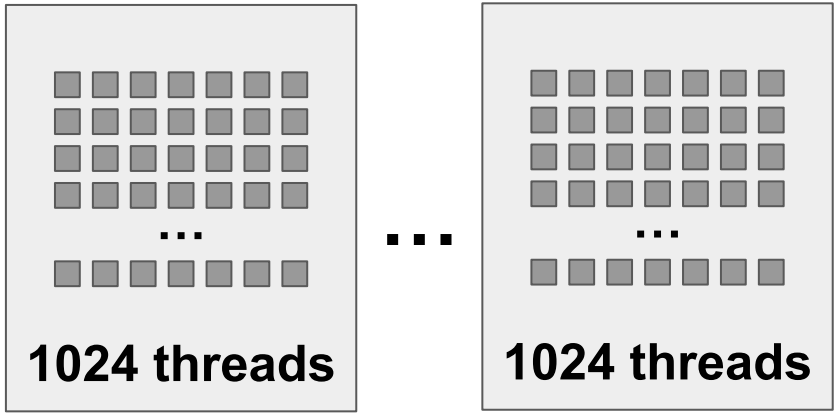
How many threads do you need to keep the GPU busy?
(e.g. H100)

132 SMs * 128 = **16,896 threads**

(but more threads often improves latency hiding!)

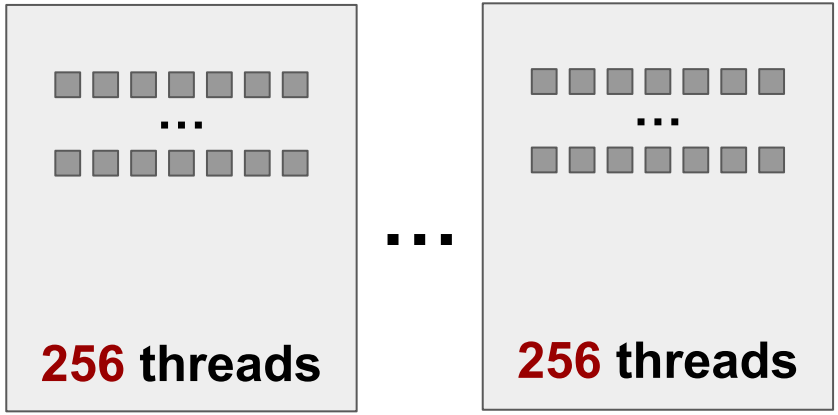
The First Puzzle

How do these compare?



128 blocks

```
my_kernel<<<128, 1024>>>(...);
```

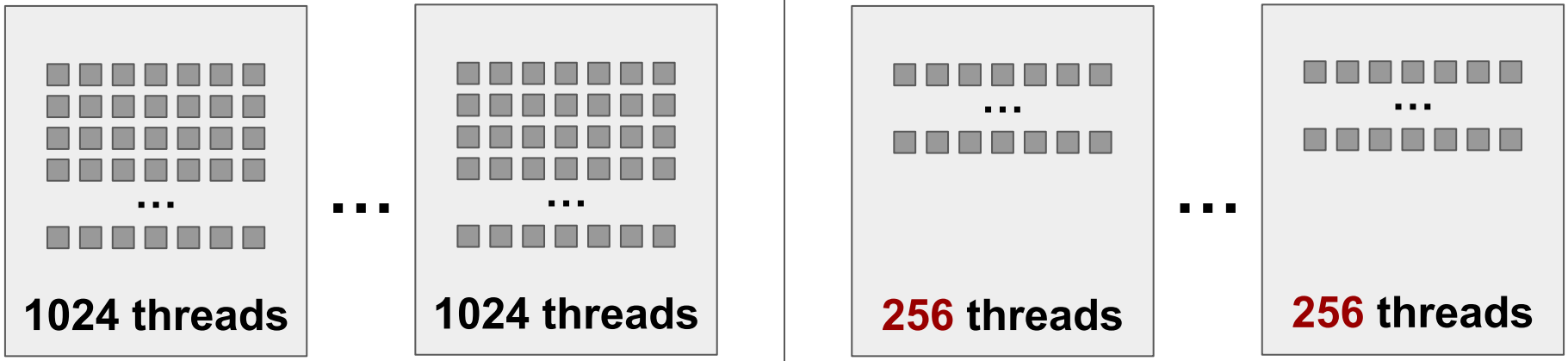


128 blocks

```
my_kernel<<<128, 256>>>(...);
```

The First Puzzle

Each one (might) have enough to saturate 128 SMs!



128 blocks

```
my_kernel<<<128, 1024>>>(...);
```

128 blocks

```
my_kernel<<<128, 256>>>(...);
```

Too many blocks → run them **serially**

Example:

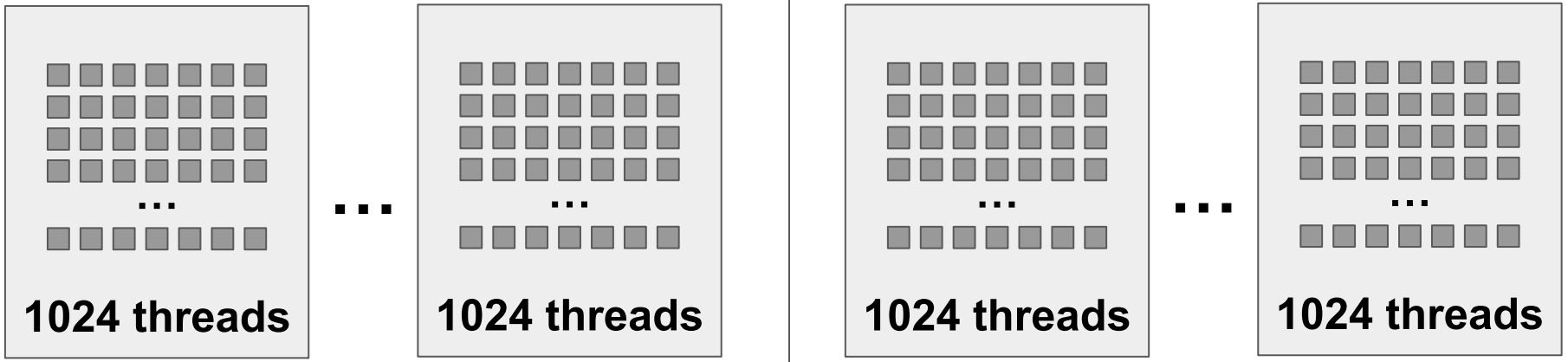
396 blocks

132 SMs

Each SM runs
3 blocks

How do these compare?

The Second Puzzle



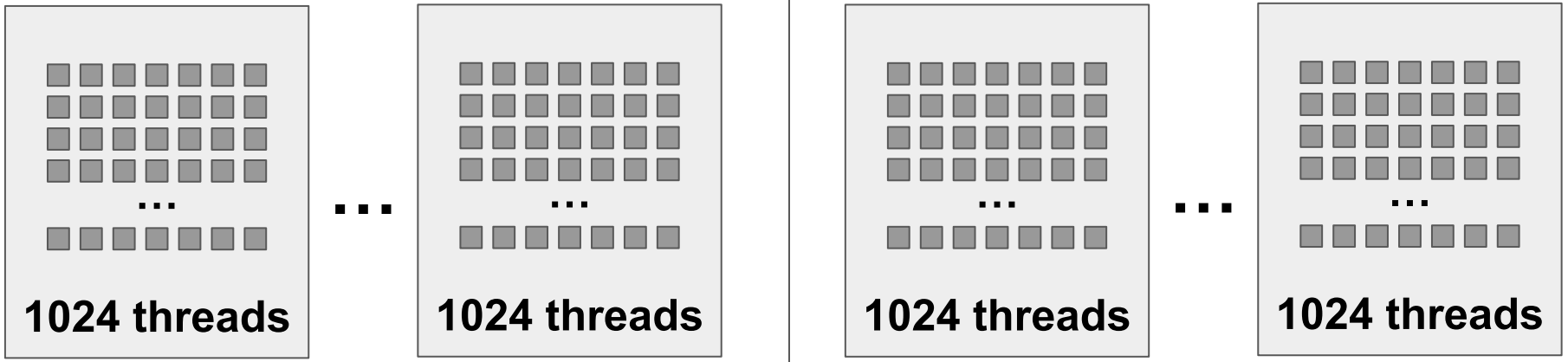
128 blocks ~4% increase **133 blocks**

```
my_kernel<<<128, 1024>>>(...);
```

```
my_kernel<<<133, 1024>>>(...);
```

133 blocks, 132 SMs → 1 SM has to run 2 blocks!

The Second Puzzle



128 blocks **< 4% increase** **133 blocks**

```
my_kernel<<<128, 1024>>>(...);
```

```
my_kernel<<<133, 1024>>>(...);
```

Bonus Chatter

- What if different threads have different **control flow**?
  - To a first approximation: **masking**
- Why do SMs exist at all? Why not only warp schedulers?
  - Shared **scratchpad memory**
- What about clusters / tensor cores / TMA / weirder features?
  - **Q&A!**

Q&A!

Materials are online!