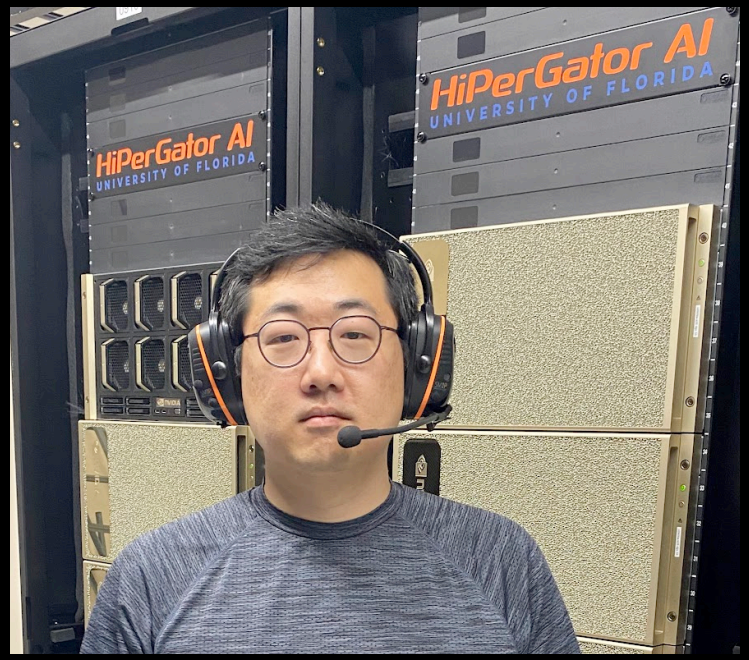
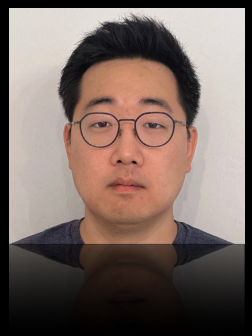
GPU Programming

HSF-India HEP Software Workshop
May 21, 2024





Philip Chang
University of Florida

Goal

Goal

Goal

Understand the guiding principles of parallel programming

Goal

Understand the guiding principles of parallel programming

Goal

Understand the guiding principles of parallel programming

Why do we need parallel programming?

Goal

Understand the guiding principles of parallel programming

Why do we need parallel programming?

Goal

Understand the guiding principles of parallel programming

Why do we need parallel programming?

Where is it going to shine?

Goal

Understand the guiding principles of parallel programming

Why do we need parallel programming?

Where is it going to shine?

Goal

Understand the guiding principles of parallel programming

Why do we need parallel programming?

Where is it going to shine?

What are the obstacles?

Goal

Understand the guiding principles of parallel programming

Why do we need parallel programming?

Where is it going to shine?

What are the obstacles?

How to think about parallel programming

Graphics Processing Units

UF
Chang
Florida

https://www.pcworld.com/article/416006/the-best-graphics-cards-for-pc-gaming.html

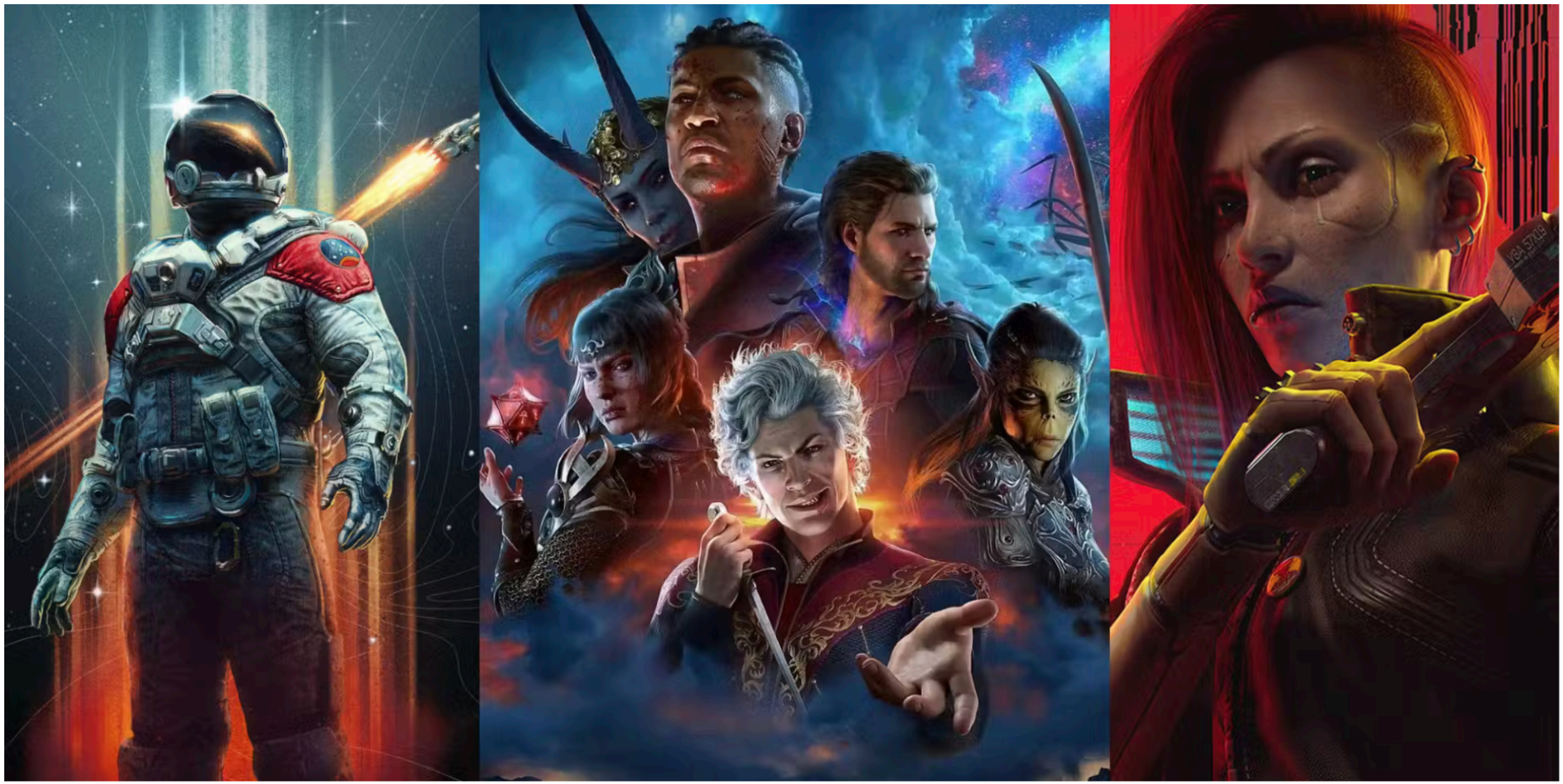
GPU

UF Chang Florida

The 20 Most Graphically Demanding PC Games

They're gorgeous, but these games have some high-spec GPU requirements. These are the most graphically demanding PC games to date.

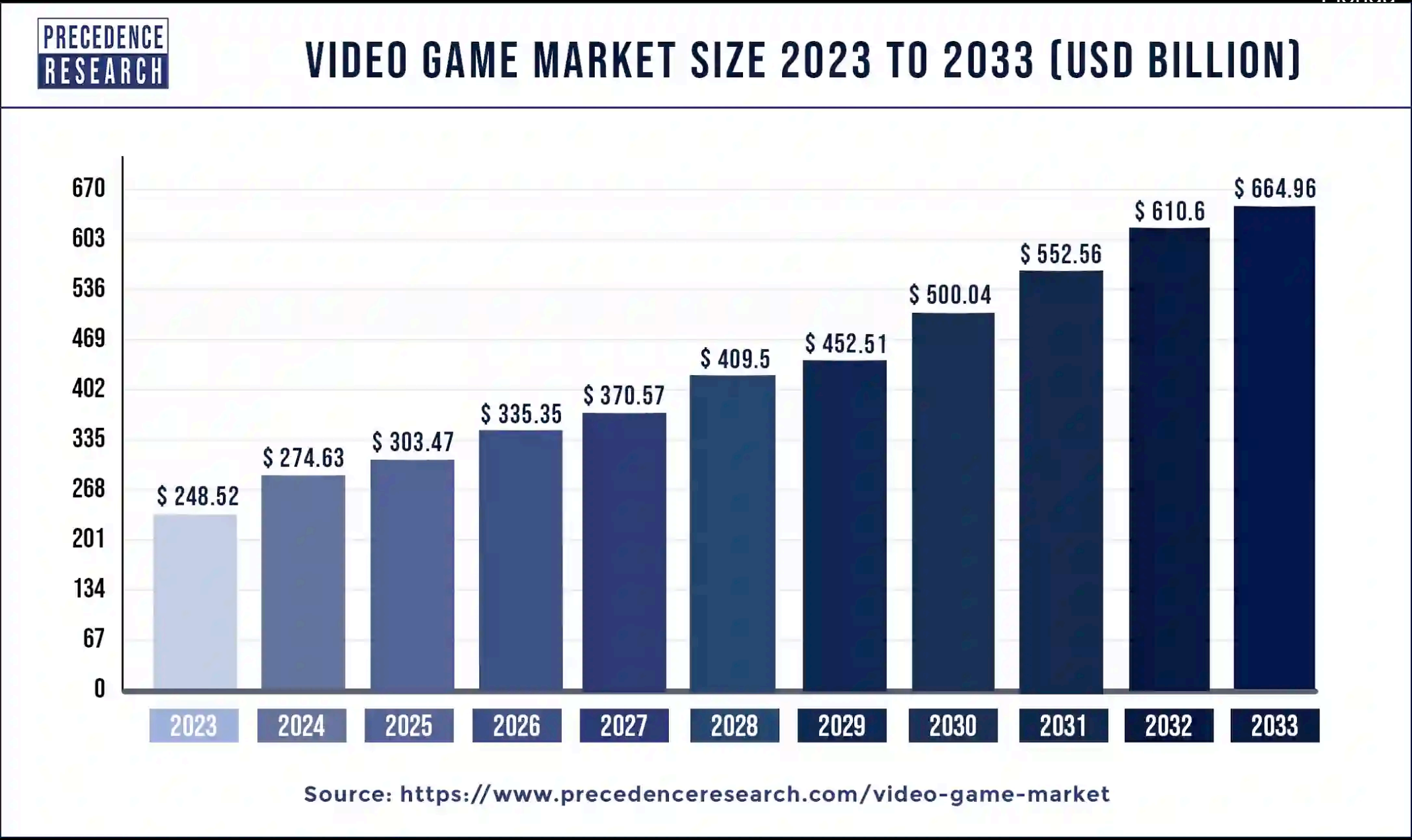
BY **CHARLES BURGAR**
UPDATED MAY 12, 2024



https://www.thegamer.com/pc-games-best-intense-graphics/

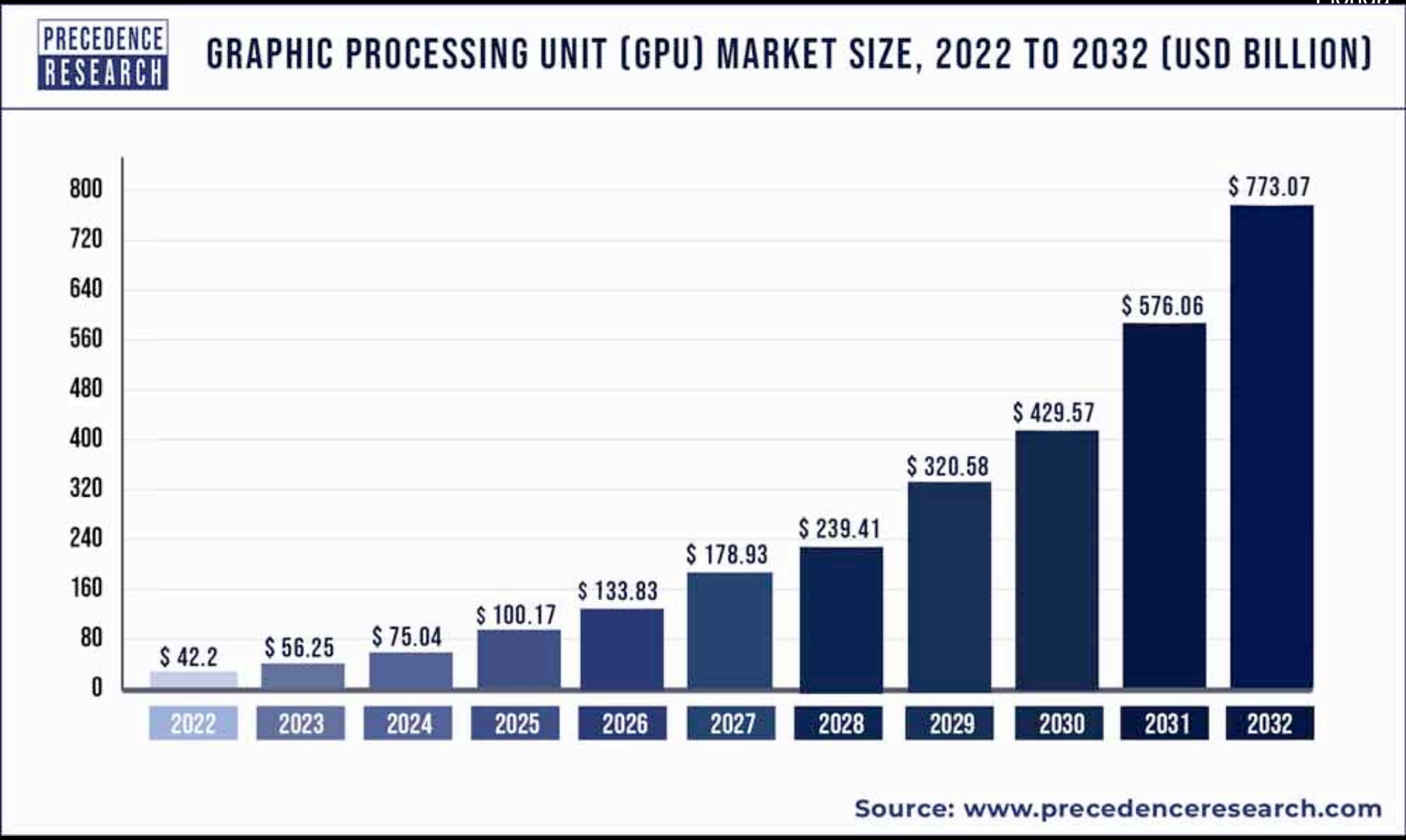
GPU is more than just gaming

UF Chang Florida



GPU is more than just gaming

UF Chang Florida




GPT-4o

SG **You**

Is GPU important for your model?

ChatGPT

Sure! GPUs are super important for models like mine. Here's why:

1. **Speeding Up Training**: Training a model like ChatGPT involves crunching through tons of data and tweaking billions of parameters. GPUs can handle many tasks at once, making this process way faster.

2. **Quick Responses**: When you use a model like this, you want quick answers. GPUs help deliver fast responses because they can process multiple things at the same time.

3. **Handling Big Models**: Models like ChatGPT are huge, and GPUs have the memory and power to manage these large models efficiently.

4. **Managing Multiple Requests**: GPUs can handle lots of requests simultaneously, which is great for providing smooth and efficient service, even when many people are using it at once.

5. **Energy Efficiency**: Even though they use a lot of power, GPUs are more efficient than CPUs for the kind of parallel tasks needed in AI.

In short, GPUs are essential for making sure models like mine run fast, efficiently, and effectively.

So GPUs are important

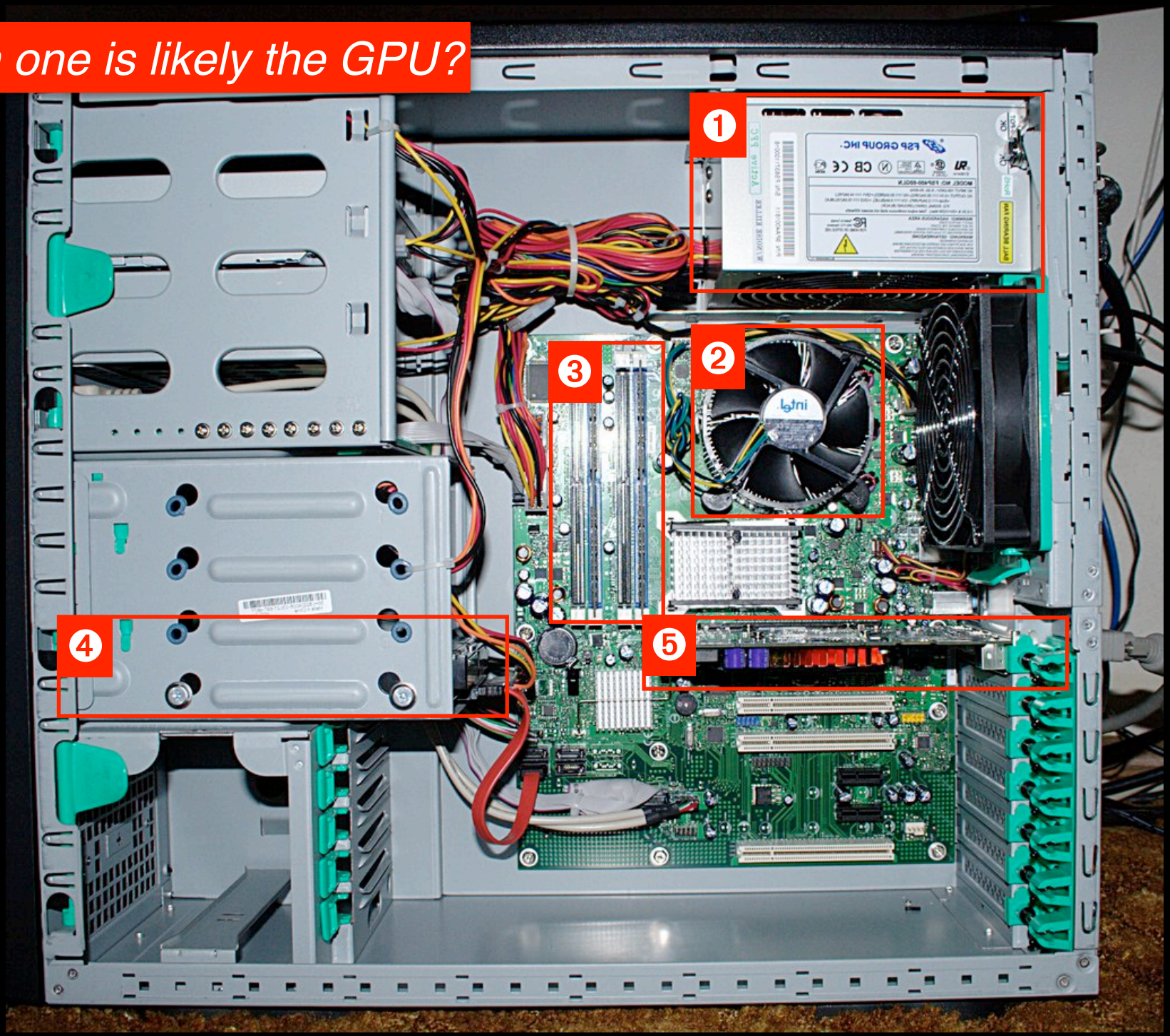
So GPUs are important

So GPUs are important

So how does this fit into our HEP computing?

https://brittanywashburn.com/2017/08/inside-of-computer-lesson-and-activity/

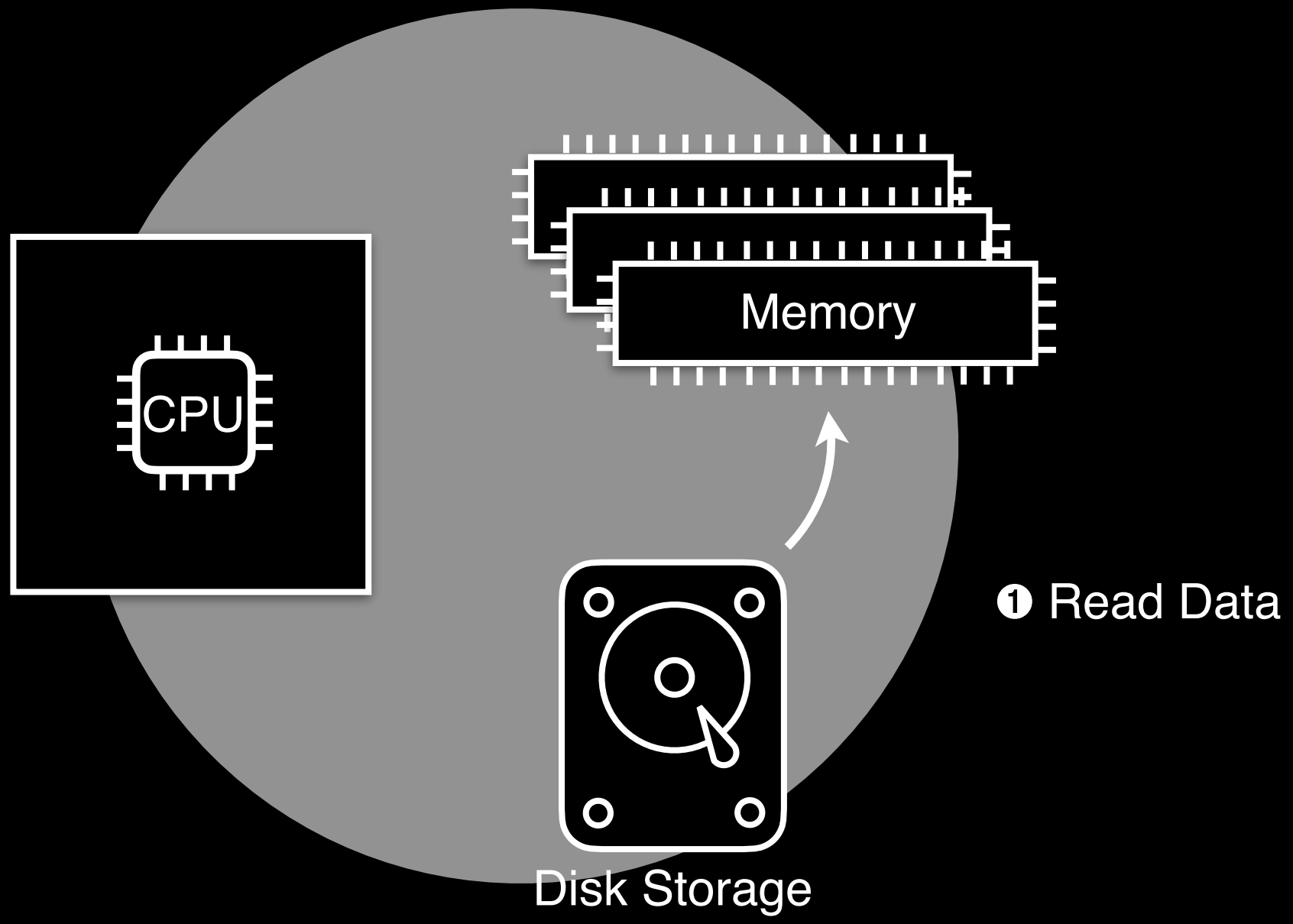
Which one is likely the GPU?



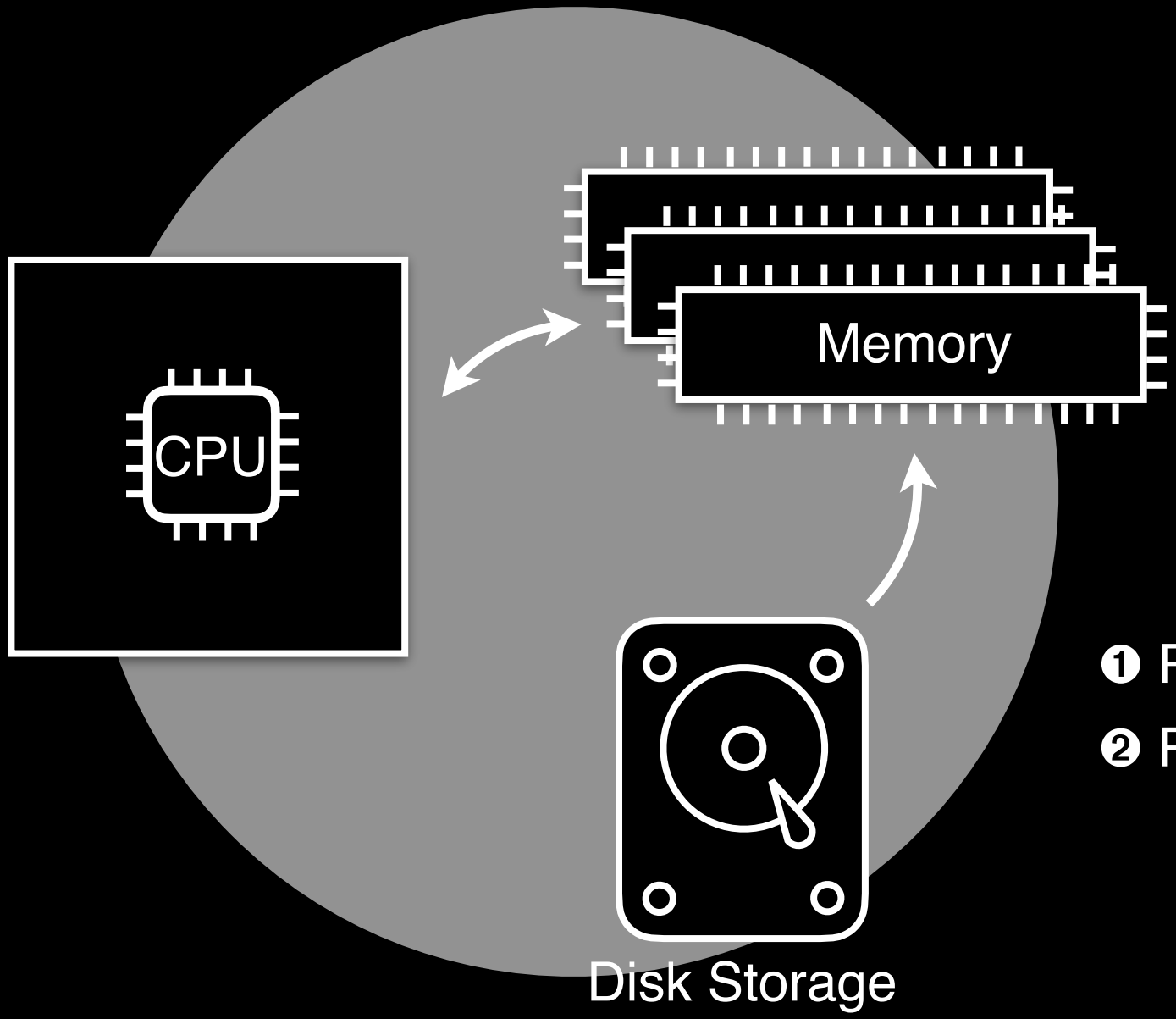
UF
Chang
Florida

https://brittanywashburn.com/2017/08/inside-of-computer-lesson-and-activity/

Computing



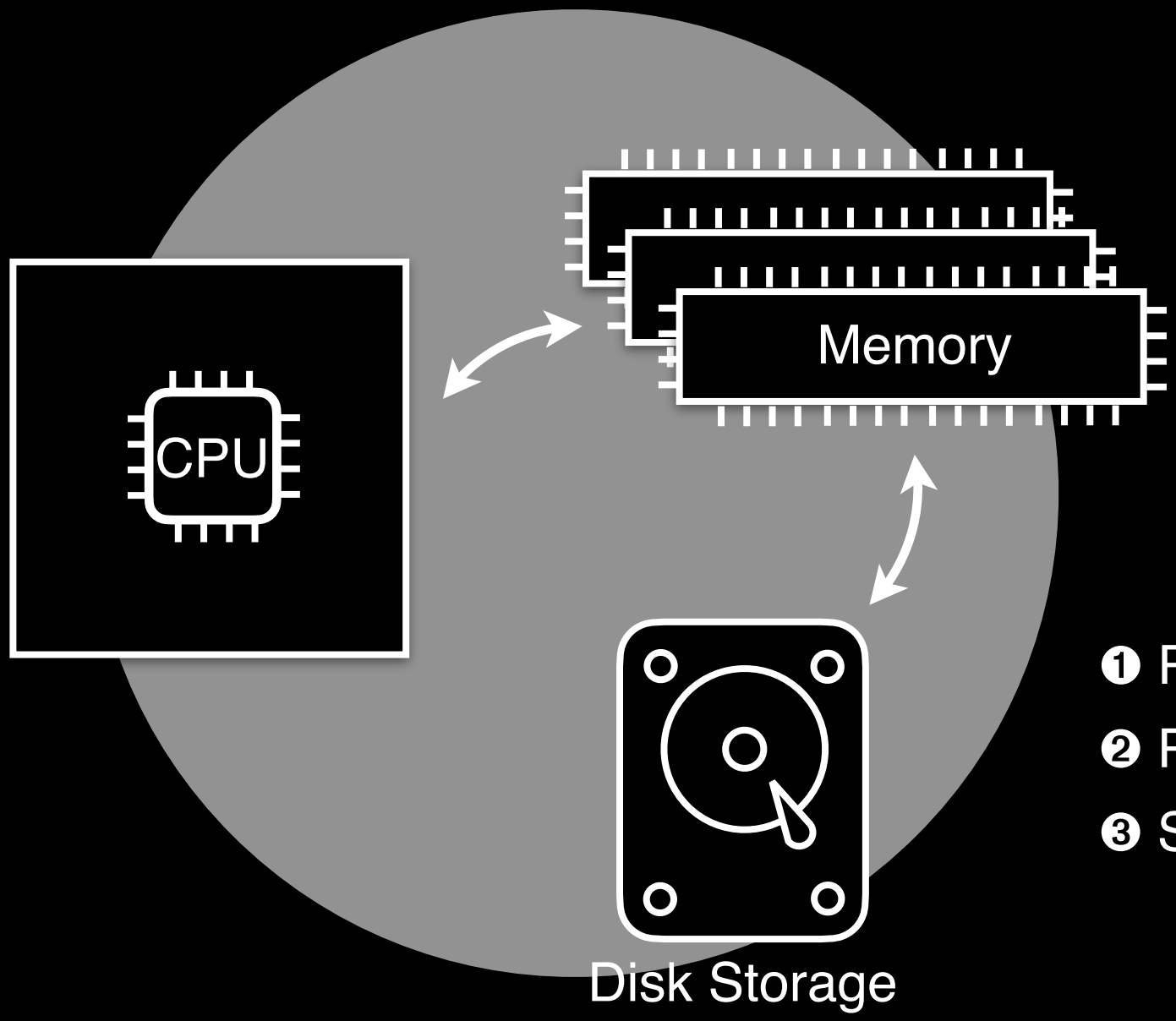
Computing



❶ Read Data

❷ Process Data

Computing



❶ Read Data

❷ Process Data

❸ Store Data

Computing Challenges

❶ Read Data
From where do we read? (near? far? network?)
How fast can we read? (network bottleneck? spinning disk?)

❷ Process Data
How do we process? (which algorithm? which workflow?)
Where do we process? (laptop? data center? supercomputing center?)
Which architecture? (CPU? GPU? FPGA? ARM?)
Which software? (Excel?? ROOT? Columnar?)

❸ Store Data
Where do we store? (near? far?)
What format? (Disk? Tape?)
What schema? (Various data tier? split up? object storage?)

Computing Challenges

UF Chang Florida

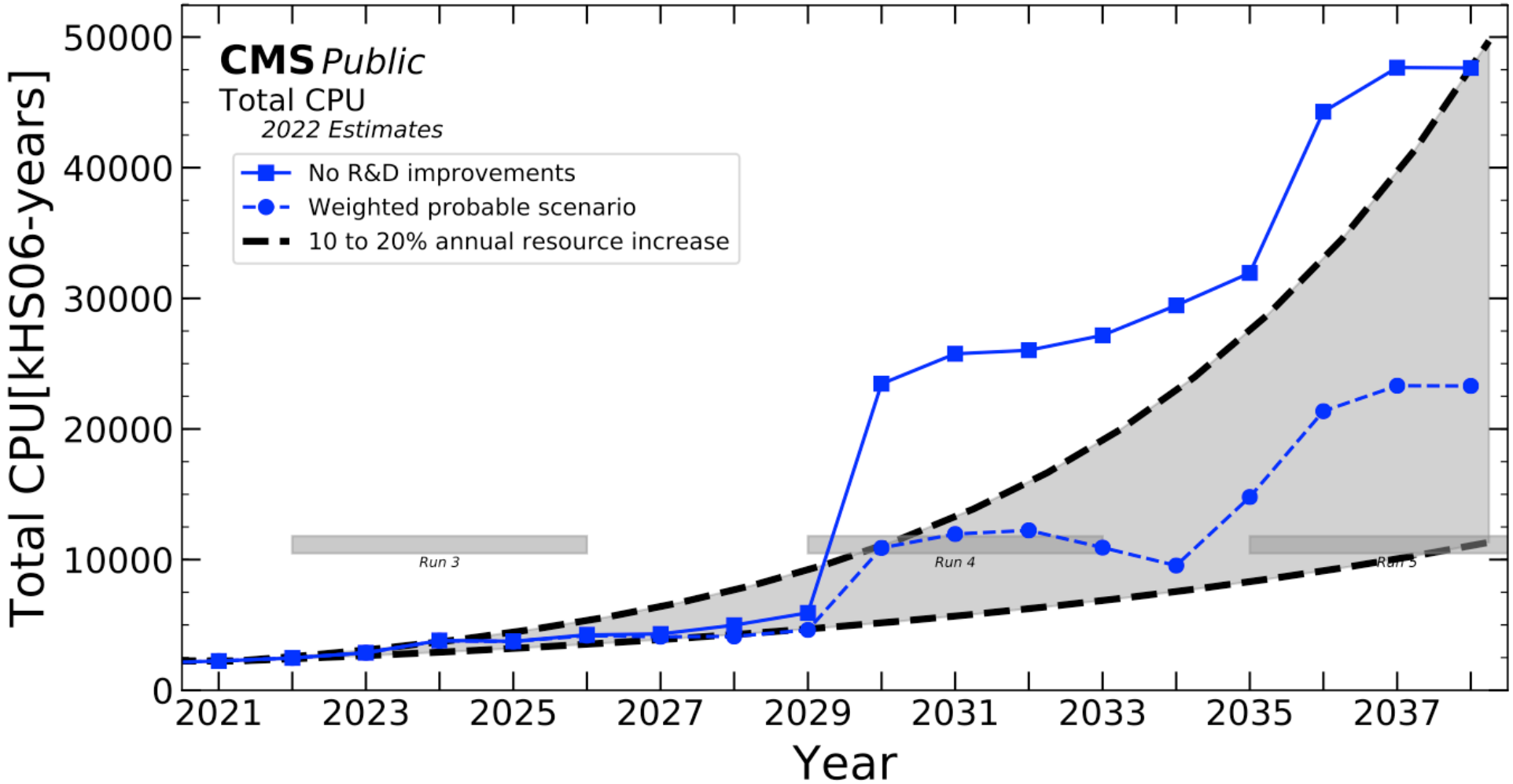
❶ Read Data
From where do we read? (near? far? network?)
How fast can we read? (network bottleneck? spinning disk?)

❷ Process Data
How do we process? (which algorithm? which workflow?)
Where do we process? (laptop? data center? supercomputing center?)
Which architecture? (CPU? GPU? FPGA? ARM?)
Which software? (Excel?? ROOT? Columnar?)

❸ Store Data
Where do we store? (near? far?)
What format? (Disk? Tape?)
What schema? (Various data tier? split up? object storage?)

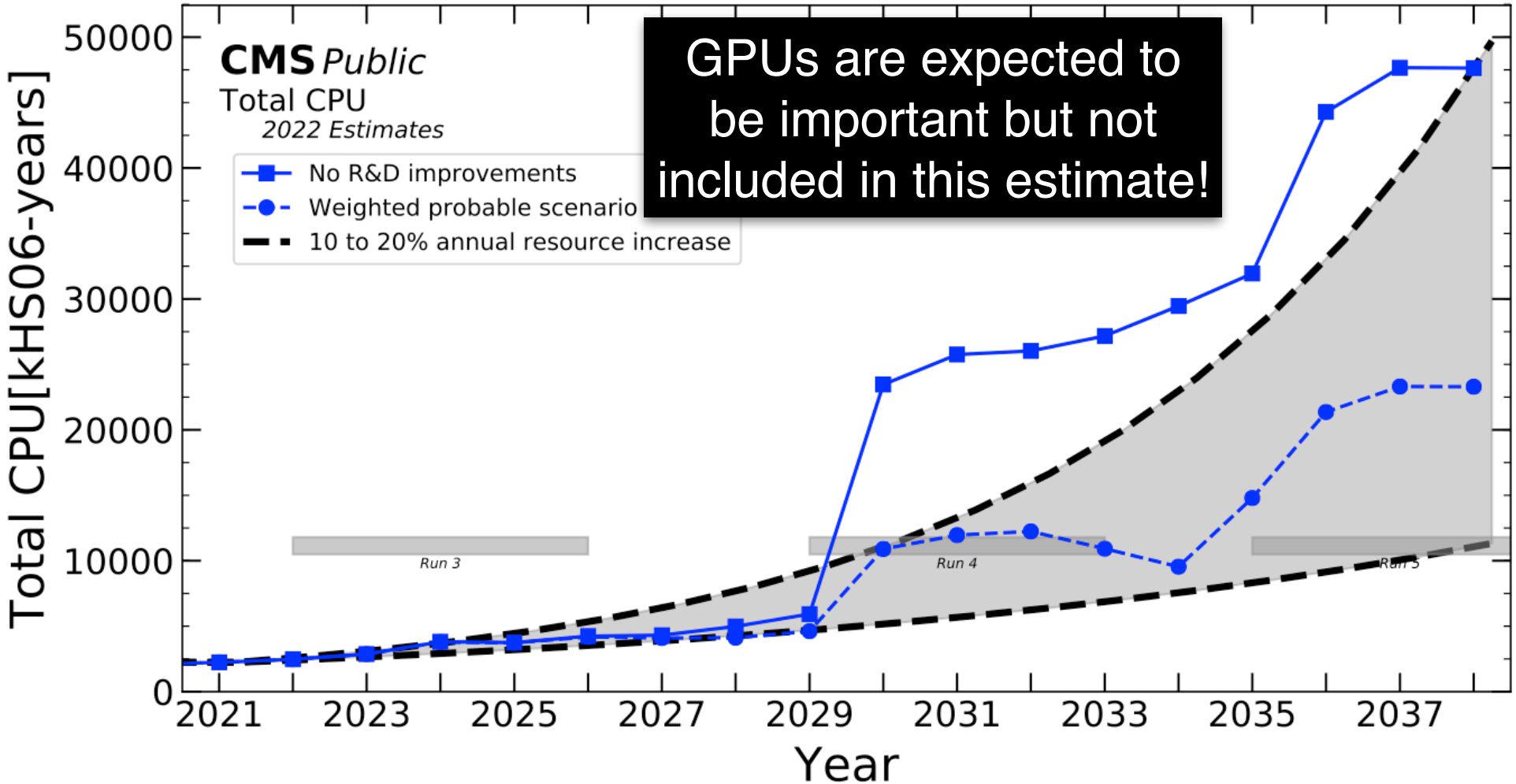
Big picture

UF
Chang
Florida



https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults

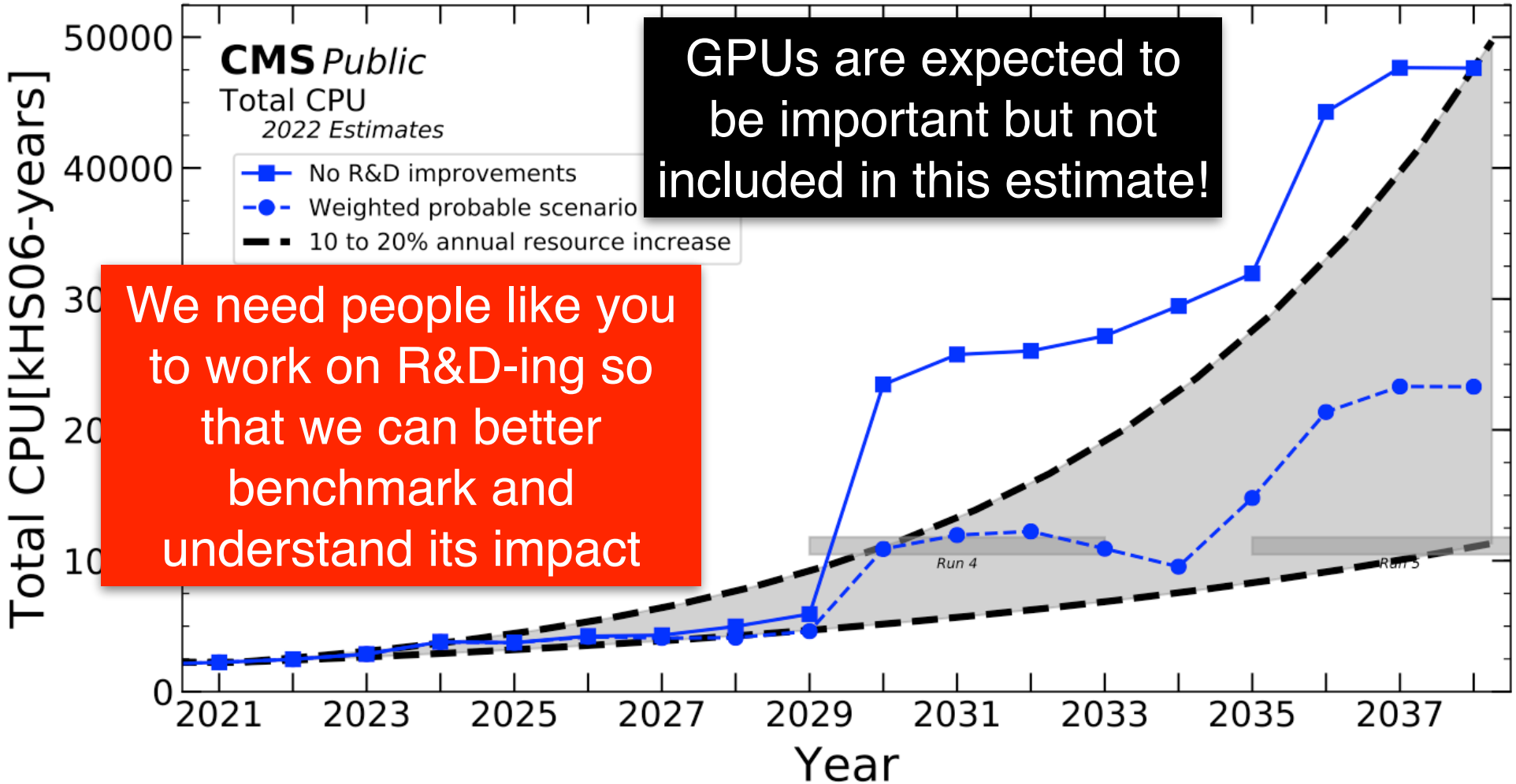
Big picture



GPUs are expected to be important but not included in this estimate!

https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults

Big picture



GPUs are expected to be important but not included in this estimate!

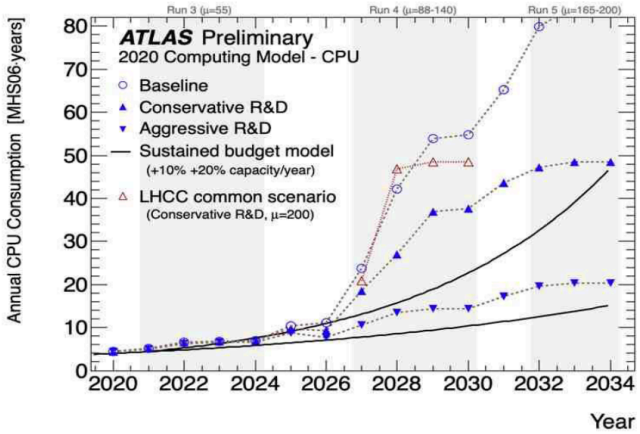
We need people like you to work on R&D-ing so that we can better benchmark and understand its impact

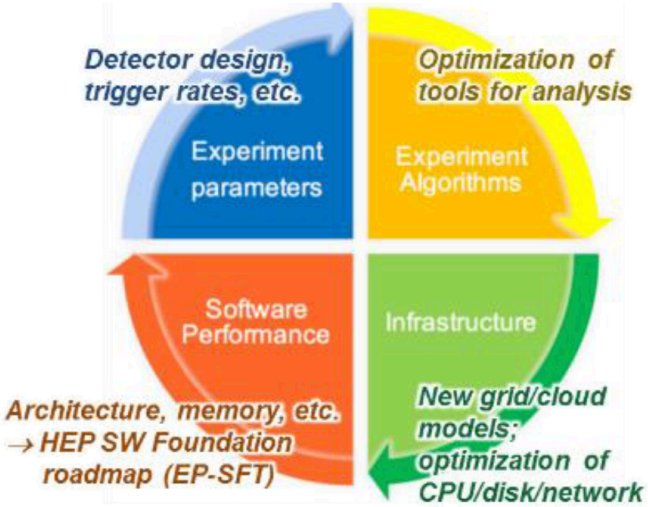
https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults

Computing in the HL-LHC Era

U.S. DEPARTMENT OF ENERGY | Office of Science

- **Simple extrapolation leads to an unsustainable place**
  - If the current software and computing approach is applied, costs can quickly exceed the entire U.S. HEP budget ("$1B problem")
- **Our goal is to match demonstrable experiment needs with a realistic funding profile — we want the science to succeed**
  - How do the software and computing models evolve?
    - much was developed beginning 15 years ago
    - they need to function 15 years from now
  - To what extent can we leverage HPC capabilities?
  - What is the optimum balance between CPU, disk, and networking?
  - R&D investments: what activities are being done or planned to address the HL-LHC software and computing challenges?
- **What is the optimum balance between people and hardware?**
  - Goal: assess computing resources and needs early enough to help inform experiments and funding agencies for successful operations during the HL-LHC era
- **For efforts towards a strategic plan, HEP Software Foundation prepared Community White Paper:** *https://arxiv.org/pdf/1712.06982.pdf* *(Dec. 2017)*
  - Additional documentation prepared by the LHC experiments during last few years





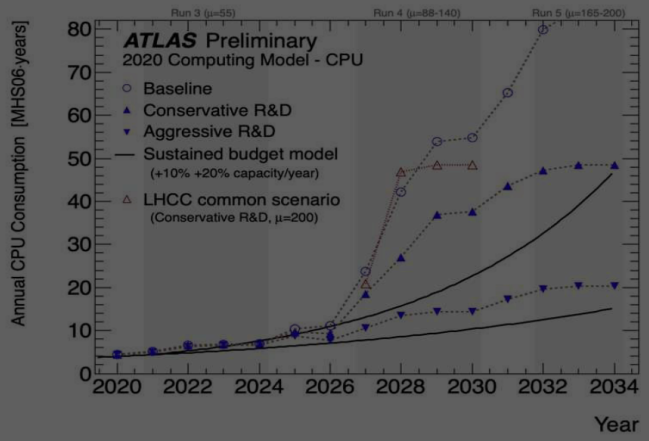
Big picture

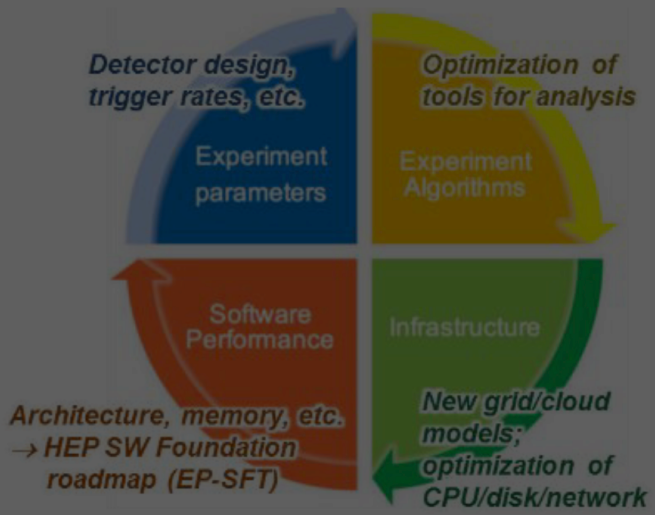
UF Chang Florida

Computing in the HL-LHC Era

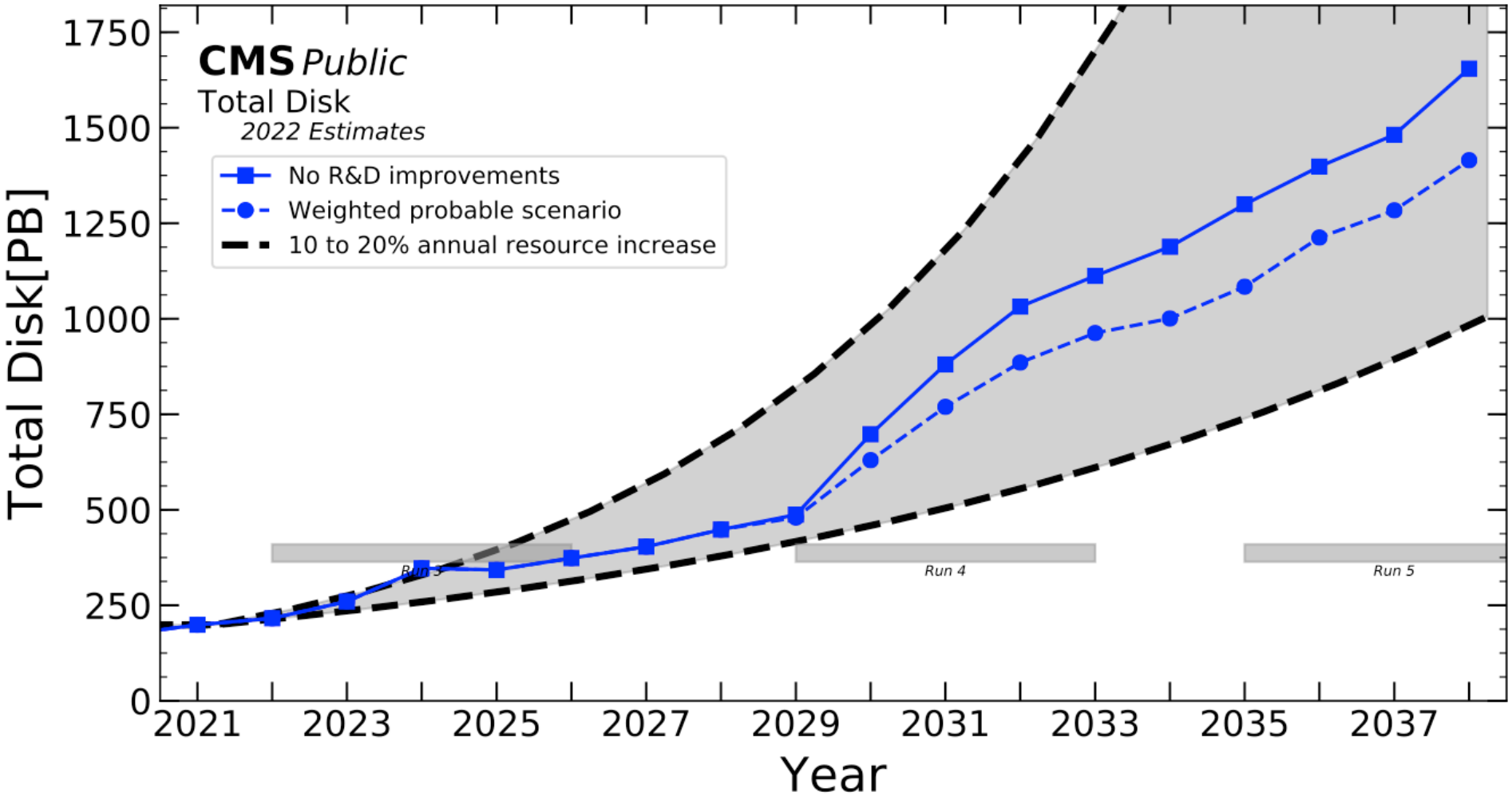
U.S. DEPARTMENT OF ENERGY | Office of Science

- **Simple extrapolation leads to an unsustainable place**
  - If the current software and computing approach is applied, costs can quickly exceed the entire U.S. HEP budget ("$1B problem")
- **Our goal is to match demonstrable experiment needs with a realistic funding profile — we want the science to succeed**
  - How do the software and computing models evolve?
    - much was developed beginning 15 years ago
    - they need to function 15 years from now
  - To what extent can we leverage HPC capabilities?
  - What is the optimum balance between CPU, disk, and networking?
  - R&D investments: what activities are being done or planned to address the HL-LHC software and computing challenges?
- **What is the optimum balance between people and hardware?**
  - Goal: assess computing resources and needs early enough to help inform experiments and funding agencies for successful operations during the HL-LHC era
- **For efforts towards a strategic plan, HEP Software Foundation prepared Community White Paper:** *https://arxiv.org/pdf/1712.06982.pdf* *(Dec. 2017)*
  - Additional documentation prepared by the LHC experiments during last few years





Big picture


CMS Public
Total Disk
2022 Estimates
No R&D improvements
Weighted probable scenario
10 to 20% annual resource increase
Run 3
Run 4
Run 5
Total Disk[PB]
Year
0
250
500
750
1000
1250
1500
1750
2021
2023
2025
2027
2029
2031
2033
2035
2037

CMS Phase-2 Computing Model: Update Document (CMS-NOTE-2022-008)

UF Chang Florida

https://cds.cern.ch/record/2815292?ln=en

8.3 Generation, simulation and reconstruction on GPUs

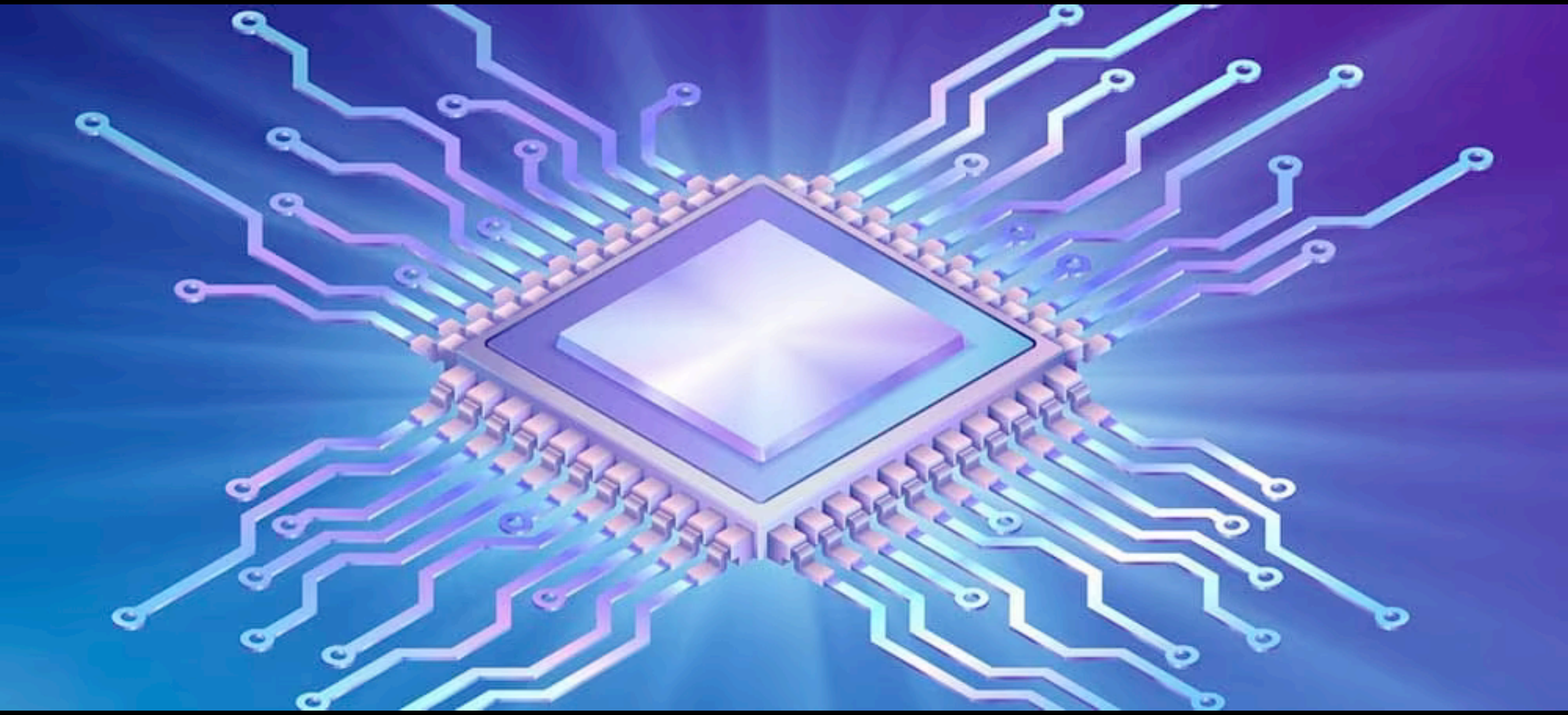
The solid architecture and robust implementation of the CMSSW framework, and its future planned developments, allow us to focus on what work can be offloaded on GPUs in the best possible way. Algorithms do not simply need to be ported, but rather re-invented to run on GPUs, taking advantage of both traditional and Machine Learning approaches. In the following we present a selection of the most prominent ongoing efforts in CMS, anticipating that

Read it!

So let’s talk about GPUs!

Central Processing Unit (CPU)

UF Chang Florida

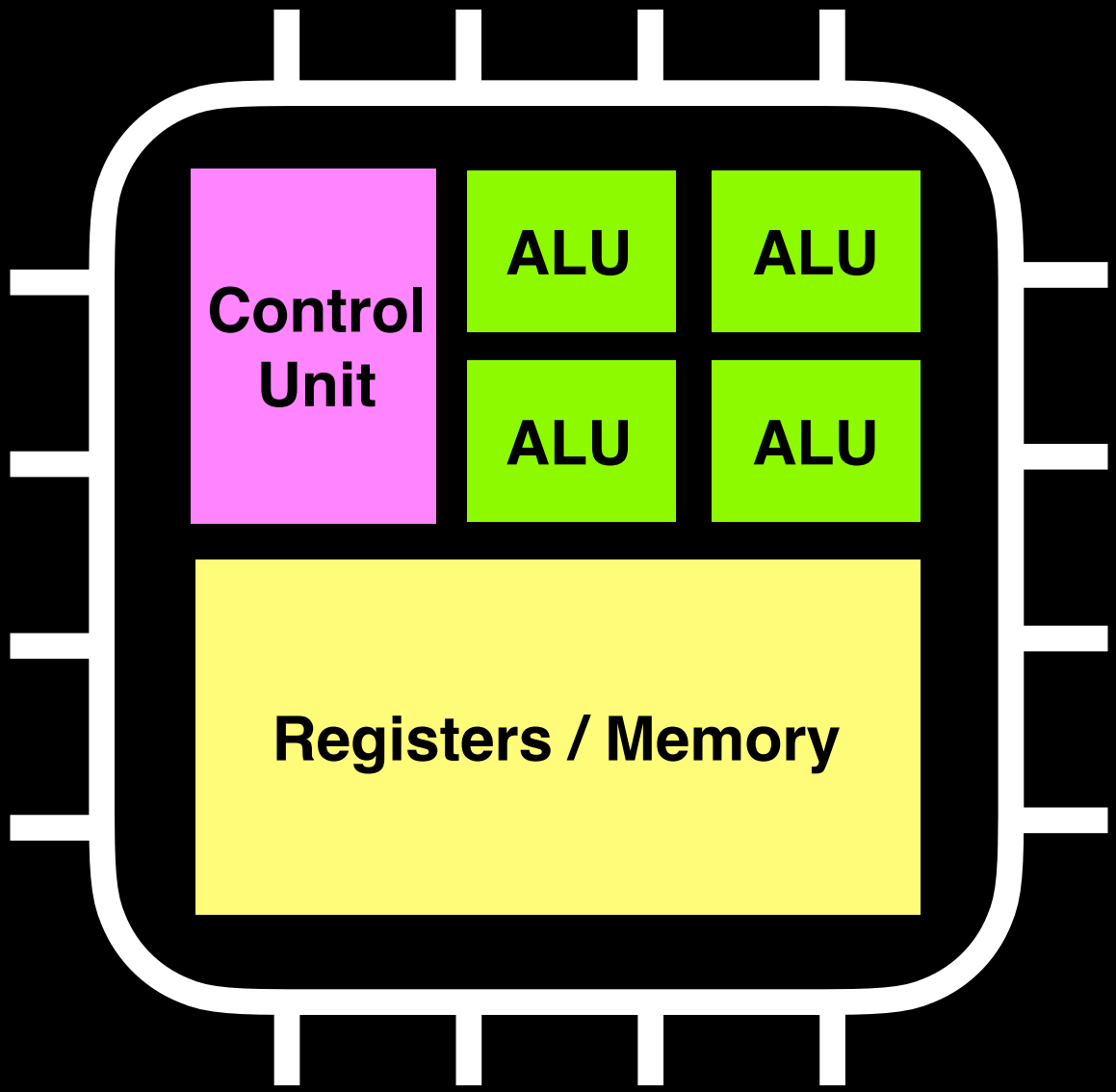


Used in most of our computers
Takes various instructions *serially* in performing tasks

https://www.hp.com/us-en/shop/tech-takes/what-is-cpu

Central Processing Units (CPUs)

UF Chang Florida



Arithmetic Logic Unit (ALU):
To perform arithmetic and logic operations

Registers / Memory:
Fast memory for Input and output of ALUs

Control Unit:
Directs operation of the processor; moving memory, executing ALUs, etc.

Fundamentally, "Get Numbers" and "Calculate"
Memory *Logic*

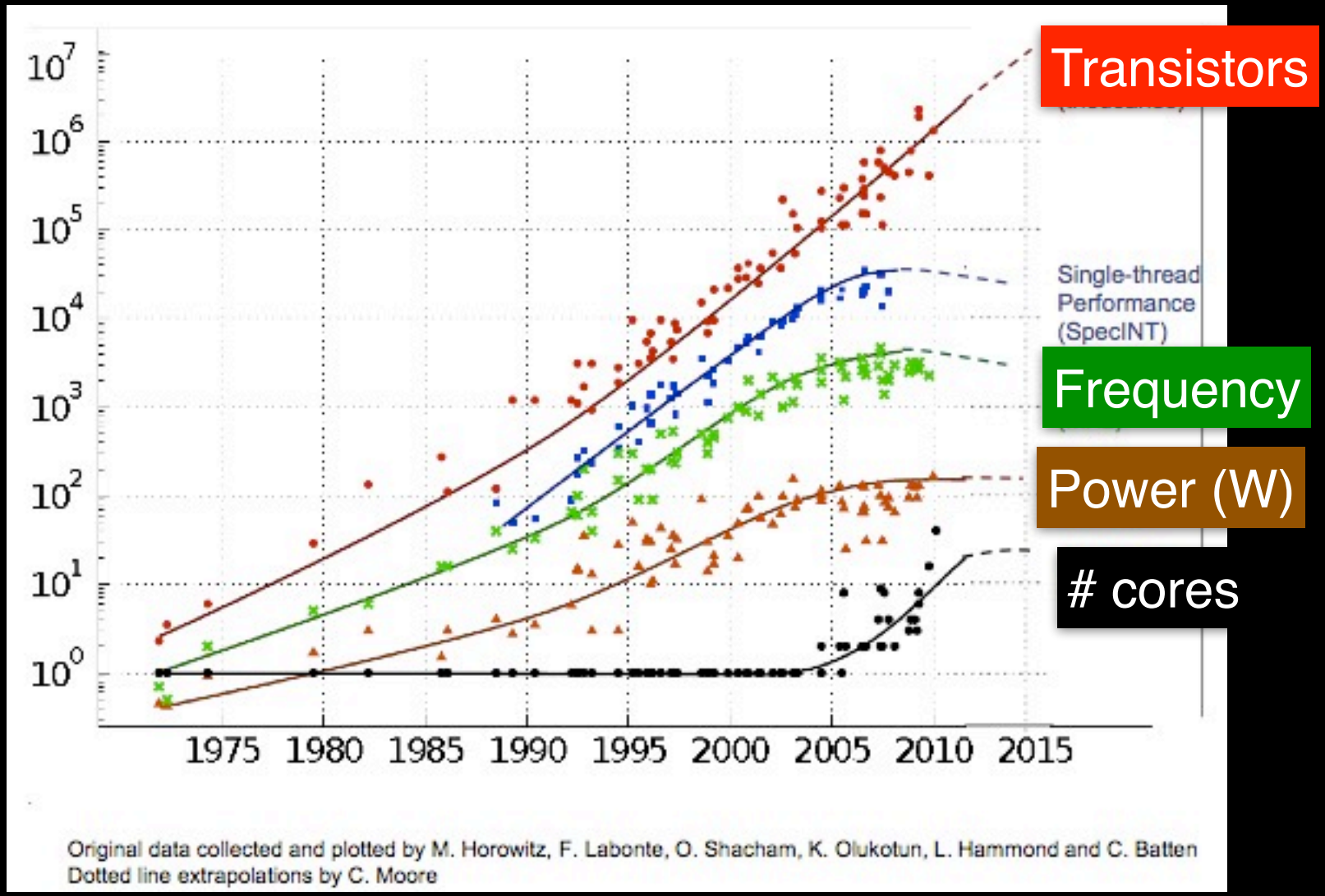
Ways to increase performance

UF
Chang
Florida

More logic units in same space = more transistors

Faster clocking = higher frequency

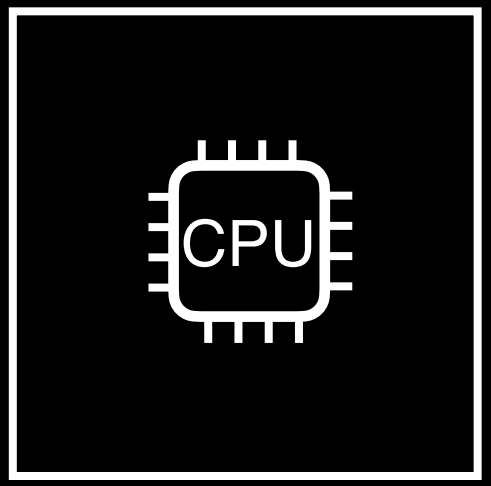
(Also need to catch up with how to push in the data)



Multicore was needed

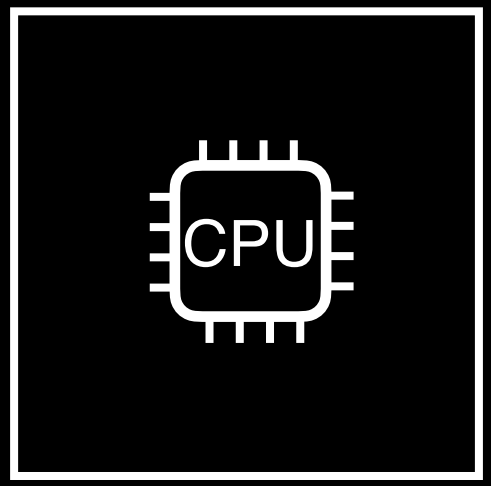
https://www.nextplatform.com/2015/08/04/future-systems-pitting-fewer-fat-nodes-against-many-skinny-ones/

Single core 1 GHz



Power ~ 1W

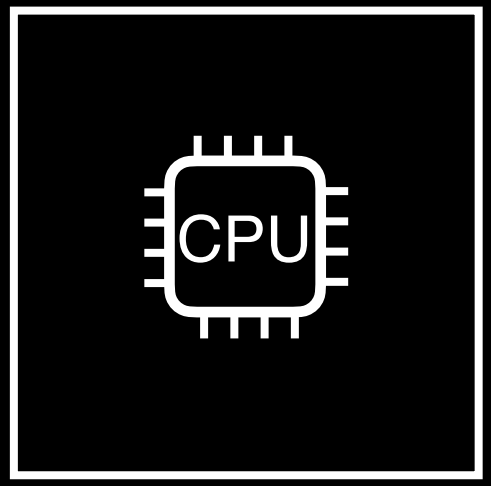
Single core 4 GHz



Power ~ 64W

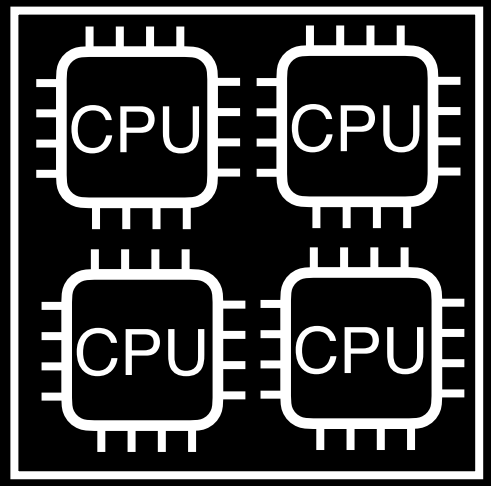
Power ~ (freq)^3

Single core 1 GHz



Power ~ 1W

Four cores 1 GHz

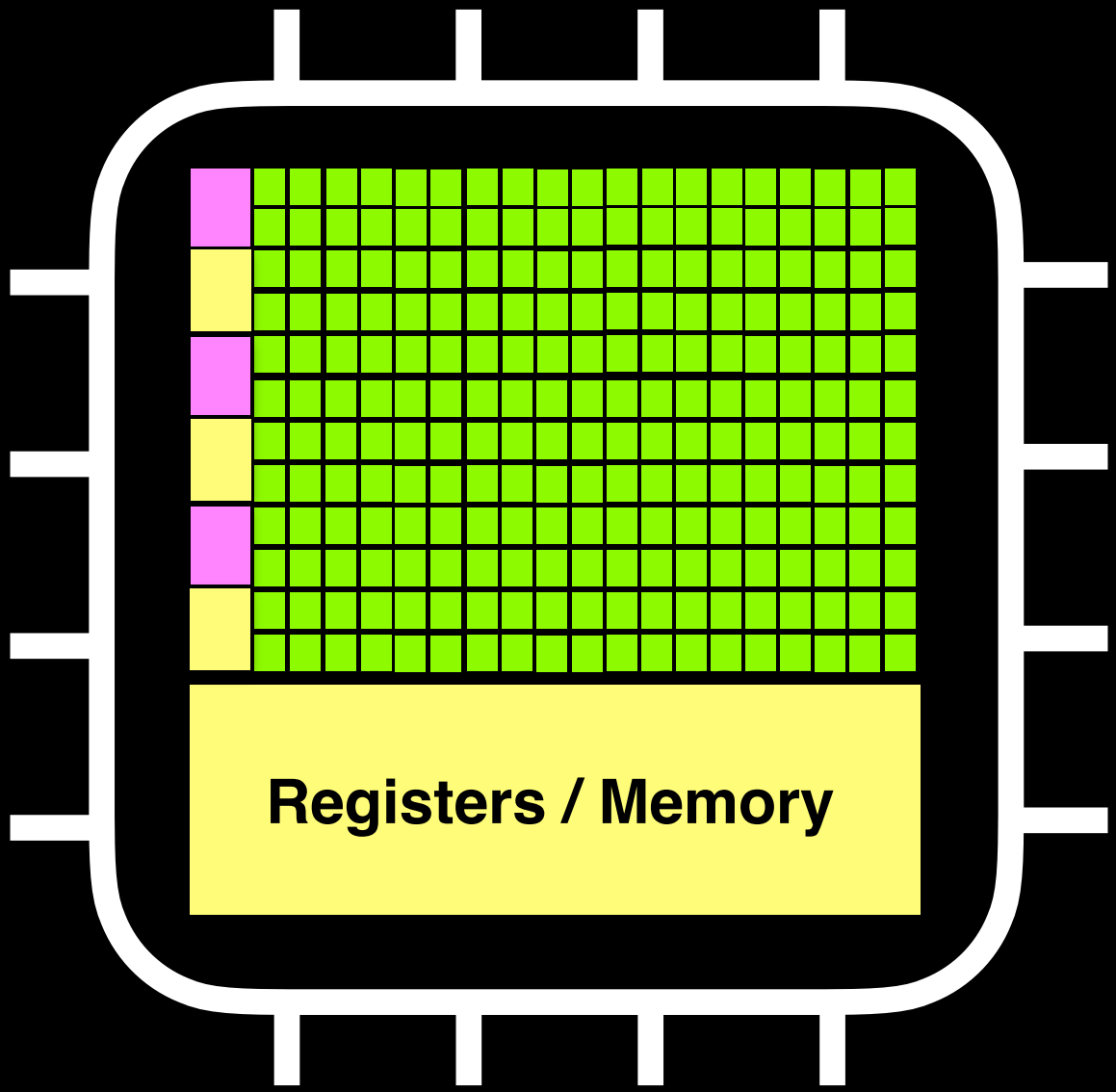


Power ~ 4W

Scaling of serial performance have reached its peak

Processors are not getting faster in clock cycle but getting diverse

Graphics Processing Units (GPUs)



Arithmetic Logic Unit (ALU):
To perform arithmetic and logic operations

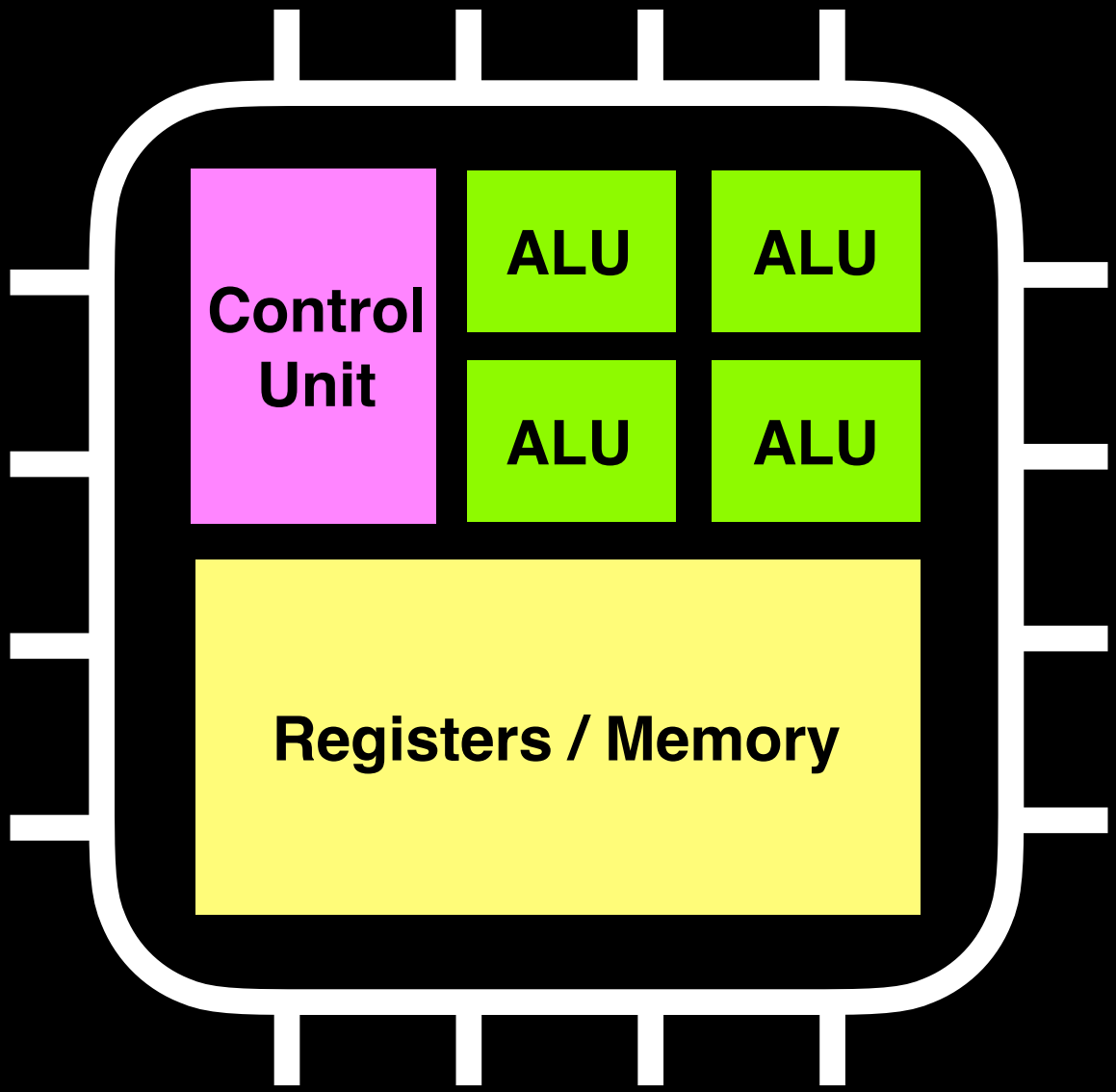
Registers / Memory:
Fast memory for Input and output of ALUs

Control Unit:
Directs operation of the processor; moving memory, executing ALUs, etc.

GPU devotes more silicons to computing

Central Processing Units (CPUs)

UF Chang Florida



Arithmetic Logic Unit (ALU):
To perform arithmetic and logic operations

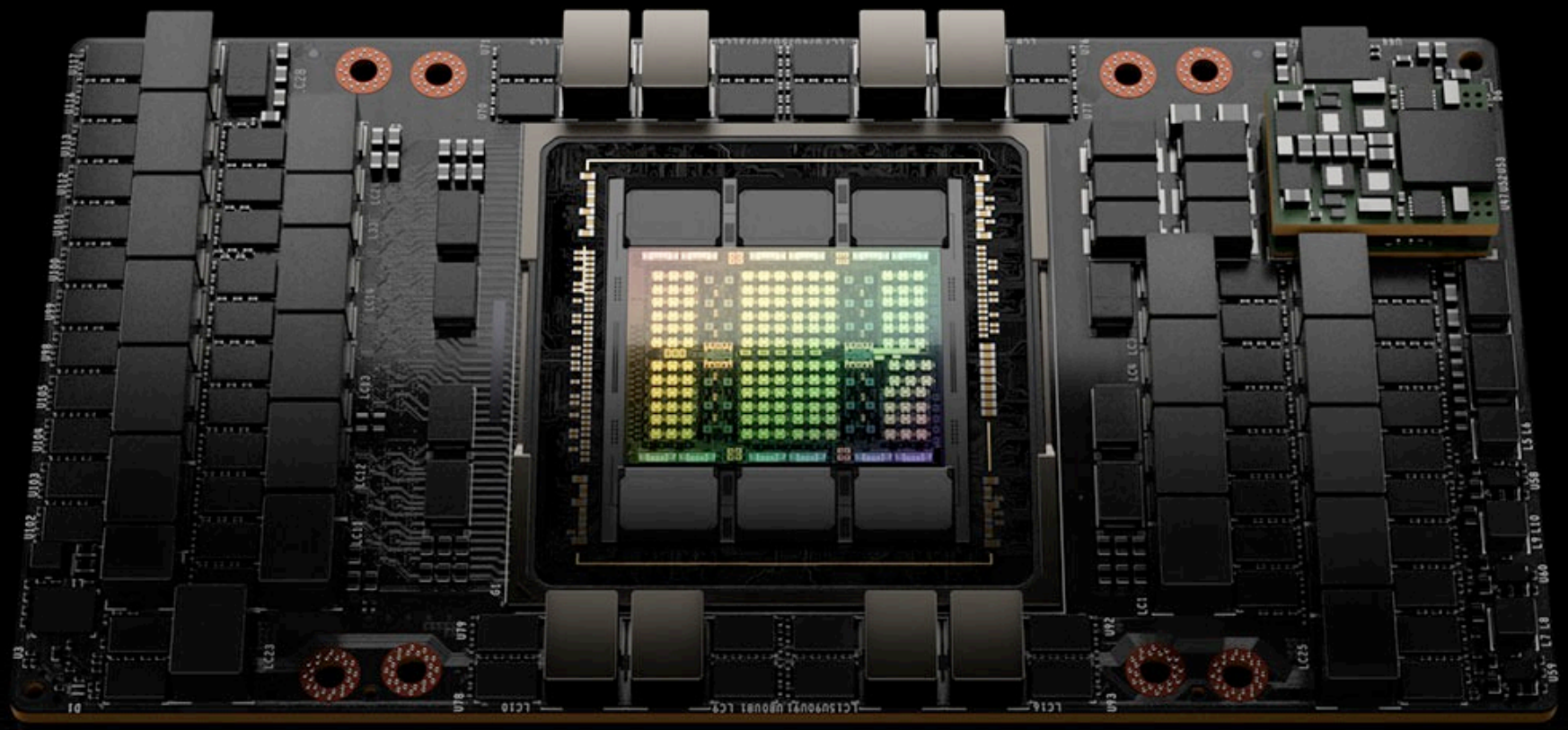
Registers / Memory:
Fast memory for Input and output of ALUs

Control Unit:
Directs operation of the processor; moving memory, executing ALUs, etc.

Fundamentally, "Get Numbers" and "Calculate"
Memory *Logic*

One of the latest GPUs

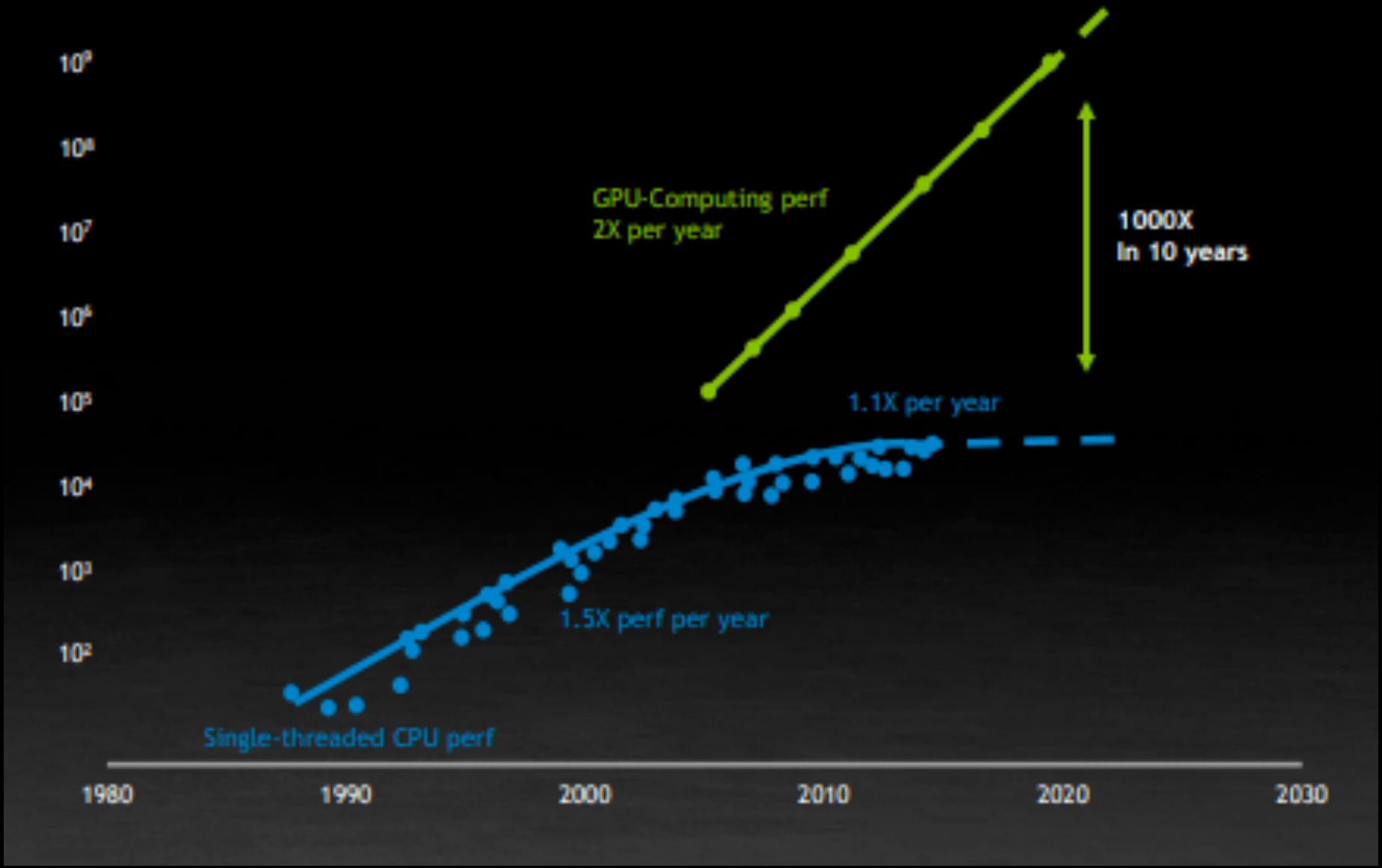
UF Chang Florida

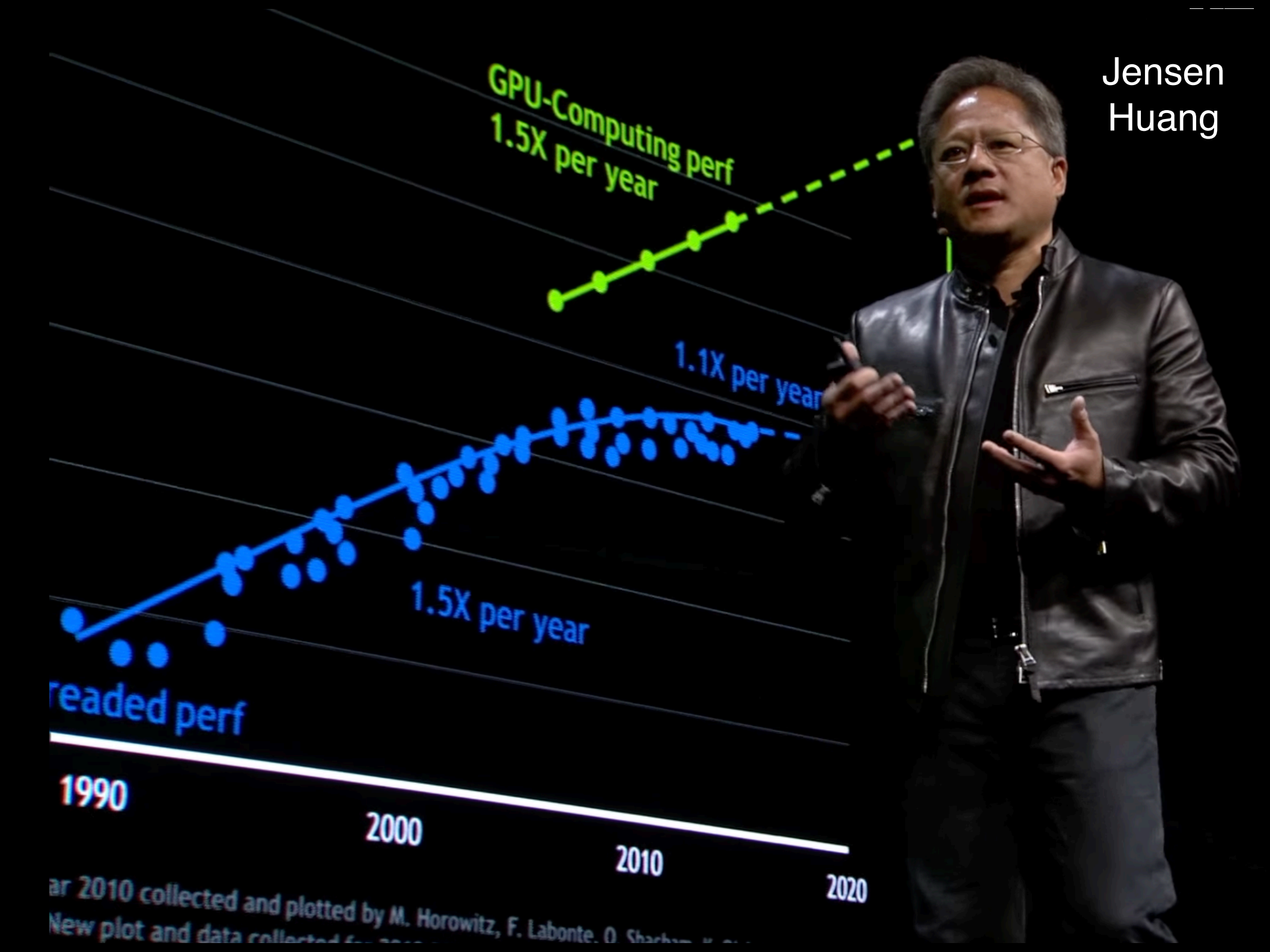


H100

14592 cores

Death of Moore's Law




Jensen Huang
GPU-Computing perf
1.5X per year
1.1X per year
1.5X per year
readed perf
1990
2000
2010
2020
ar 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham
New plot and data collected

Huang's law

3 languages

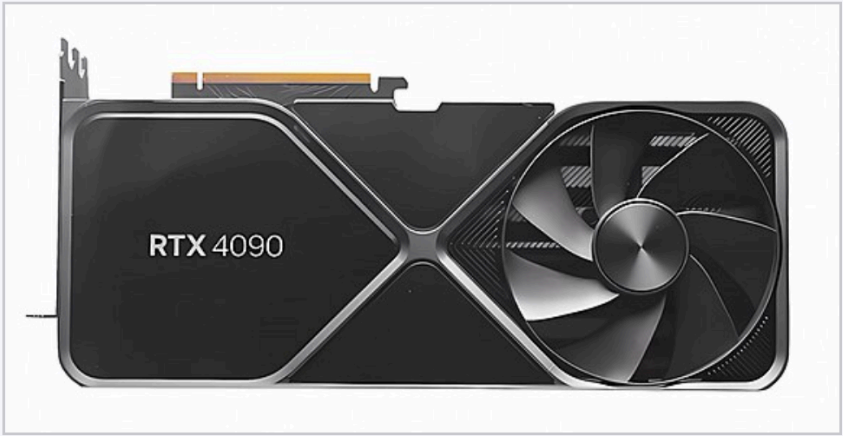
Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

Huang's law is an observation in computer science and engineering that advancements in graphics processing units (GPUs) are growing at a rate much faster than with traditional central processing units (CPUs). The observation is in contrast to Moore's law that predicted the number of transistors in a dense integrated circuit (IC) doubles about every two years.[1] Huang's law states that the performance of GPUs will more than double every two years.[2] The hypothesis is subject to questions about its validity.

History [edit]

The observation was made by Jensen Huang, the chief executive officer of Nvidia, at its 2018 GPU Technology Conference (GTC) held in San Jose, California.[3] He observed that Nvidia's GPUs were "25 times faster than five years ago" whereas Moore's law would have expected only a ten-fold increase.[2] As microchip components become smaller, it became harder for chip advancement to meet the speed of Moore's law.[4]



An RTX 4090, the most recent flagship card in Nvidia's GeForce series, with 82.58 TFLOPS at

In 2006 Nvidia's GPU had a 4x performance advantage over

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.8	
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.0	
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.8	
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.2	
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107
6	**Alps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	1,305,600	270.00	353.75	5,194
7	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, EVIDEN EuroHPC/CINECA Italy	1,824,768	241.20	306.31	7,494
8	**MareNostrum 5 ACC** - BullSequana XH3000, Xeon Platinum 8460Y+ 32C 2.3GHz, NVIDIA H100 64GB, Infiniband NDR, EVIDEN EuroHPC/BSC Spain	663,040	175.30	249.44	4,159
9	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
10	**Eos NVIDIA DGX SuperPOD** - NVIDIA DGX H100, Xeon Platinum 8480C 56C 3.8GHz, NVIDIA H100, Infiniband NDR400, Nvidia NVIDIA Corporation United States	485,888	121.40	188.65	

Rank	System
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States

one non-GPU supercomputer

Top 500 list "June" of 2024

So how does having many cores help?

H100 example

UF
Chang
Florida

https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

H100 example

UF
Chang
Florida

https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

H100 example

UF Chang Florida

https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

UF
Chang
Florida

Stream Multi-processor (SM)

These are "**SIMD**" Processor

Single Instruction Multiple Data

A [0] + B [0] = C [0]

A [1] + B [1] = C [1]

A [2] + B [2] = C [2]

A [3] + B [3] = C [3]

UF
Chang
Florida

Stream Multi-processor (SM)

These are "**SIMD**" Processor

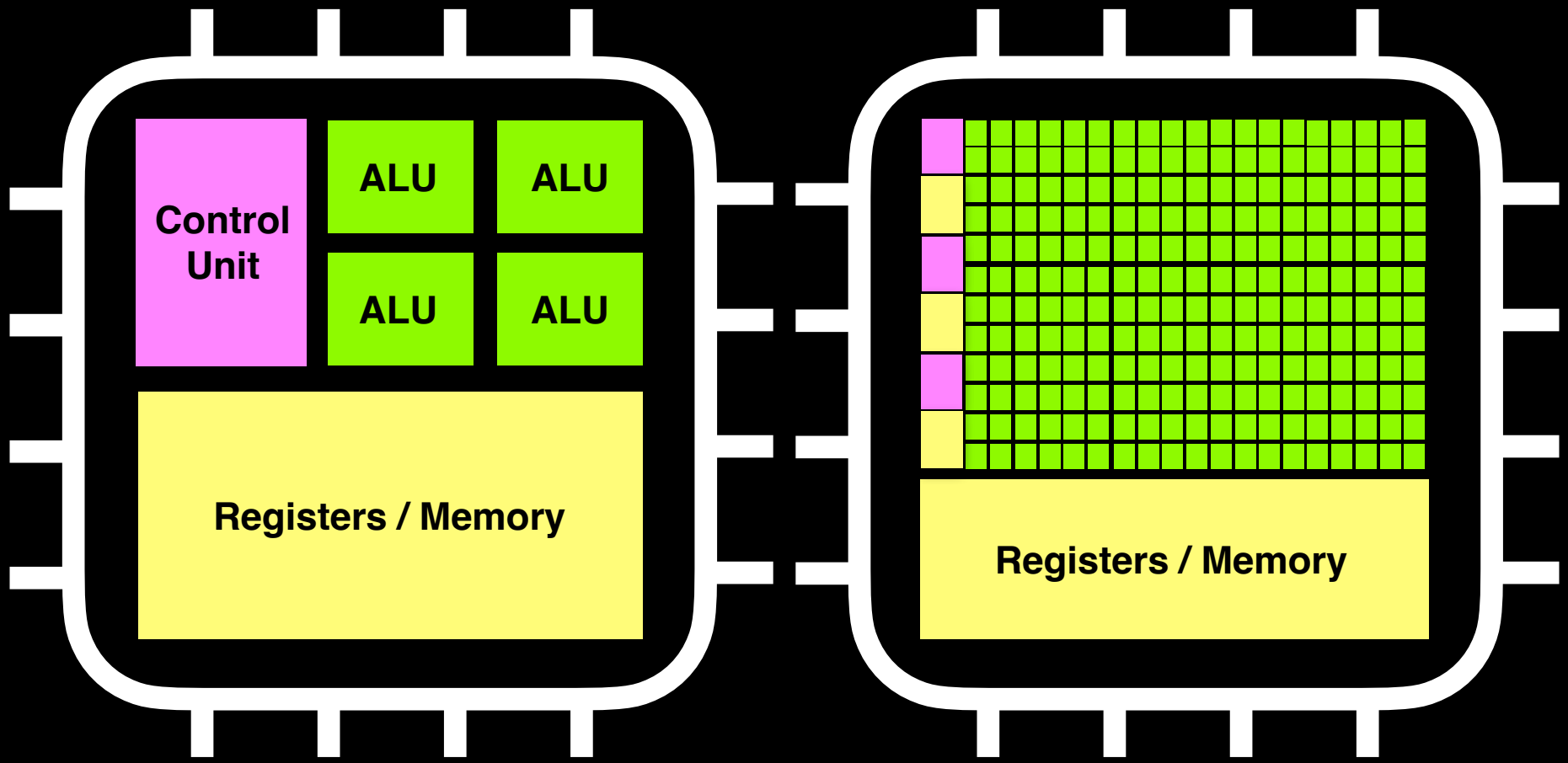
Single Instruction Multiple Data

A [0]	B [0]	C [0]
A [1]	B [1]	C [1]
+		=
A [2]	B [2]	C [2]
A [3]	B [3]	C [3]

Differences

GPUs have less CU ⇒ latency is higher

But GPUs have more "simple" cores ⇒ throughput is higher

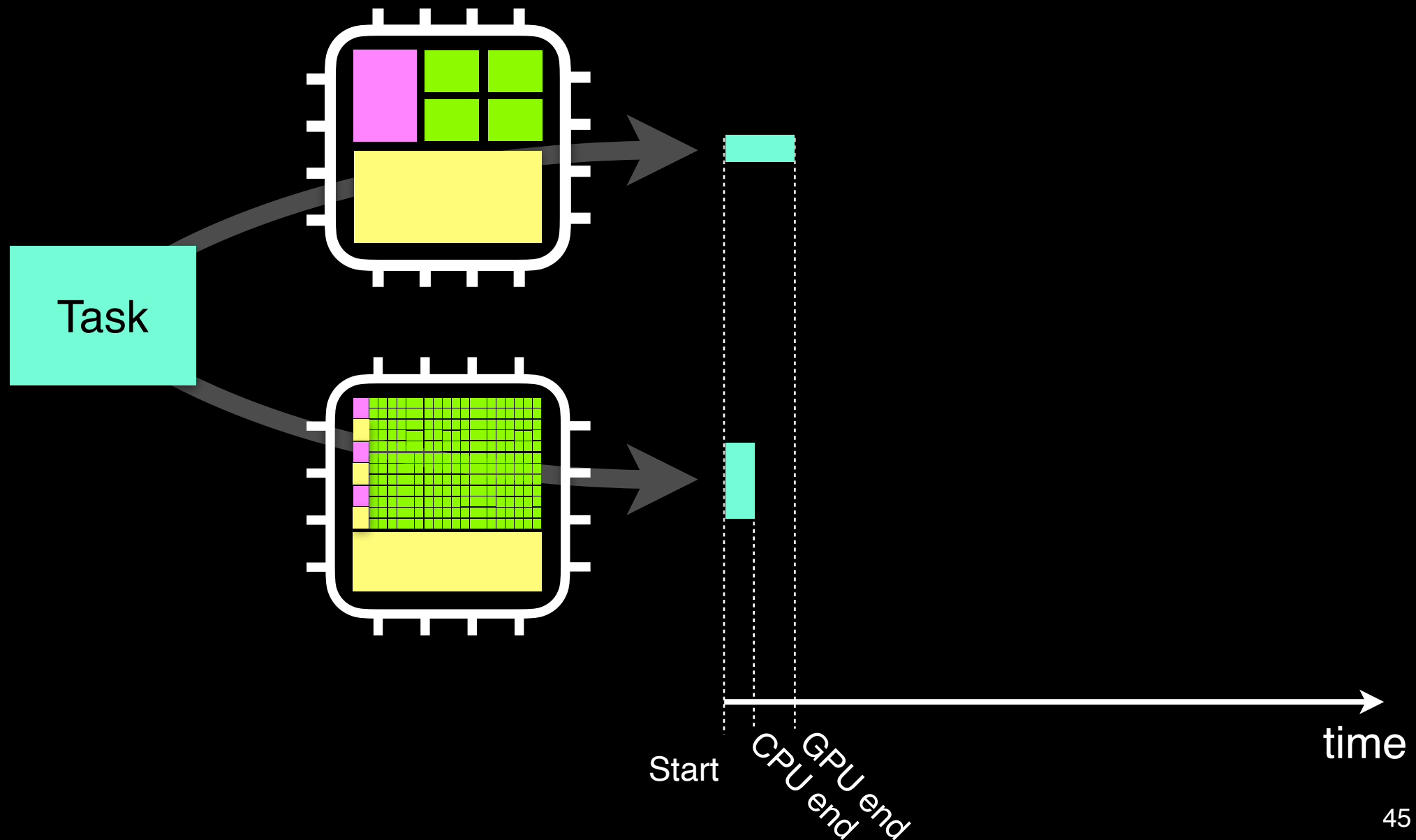


GPUs ⇒ maximize throughput of all threads

CPUs ⇒ maximize latency of single thread

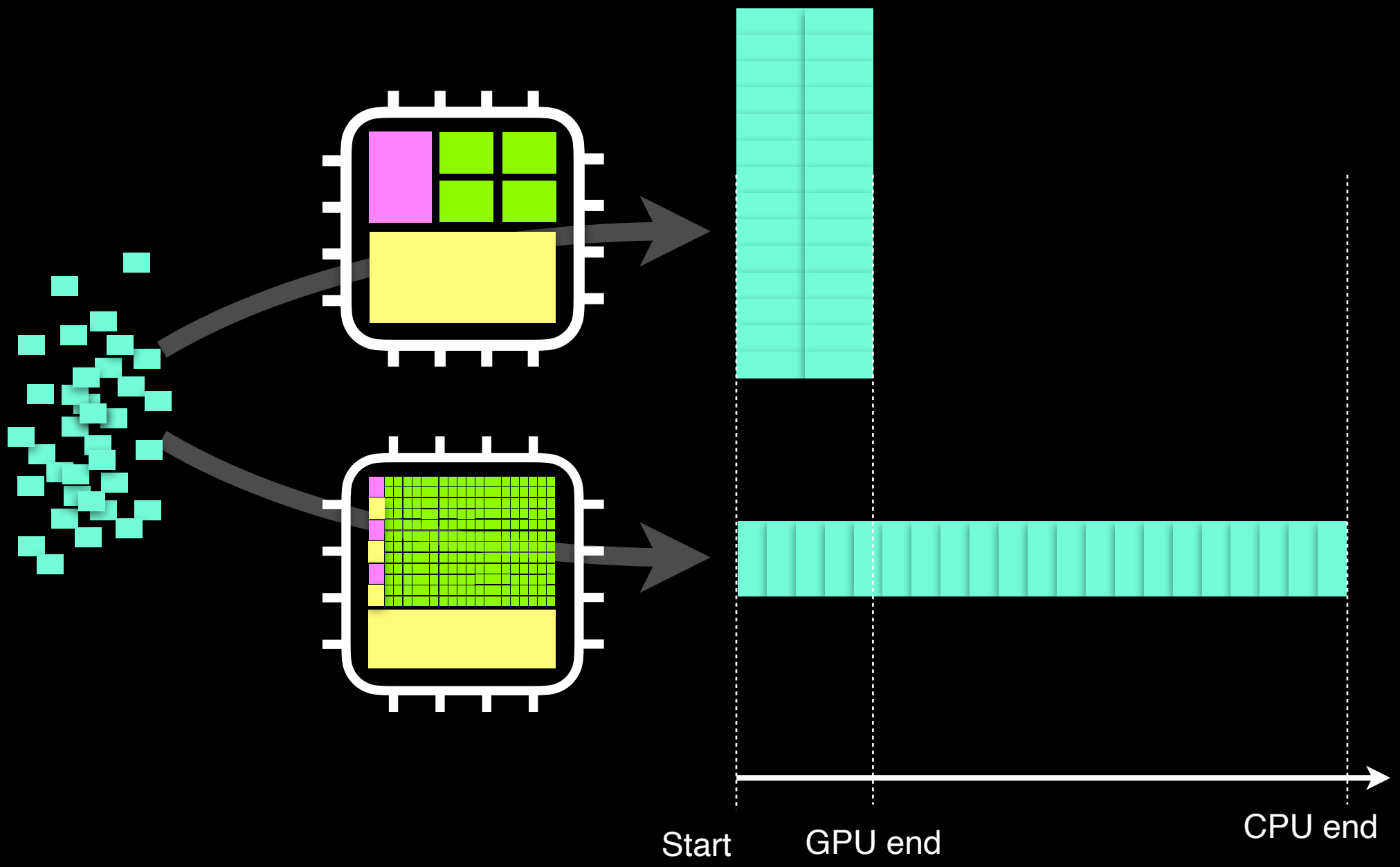
Differences

UF
Chang
Florida



Differences

UF
Chang
Florida

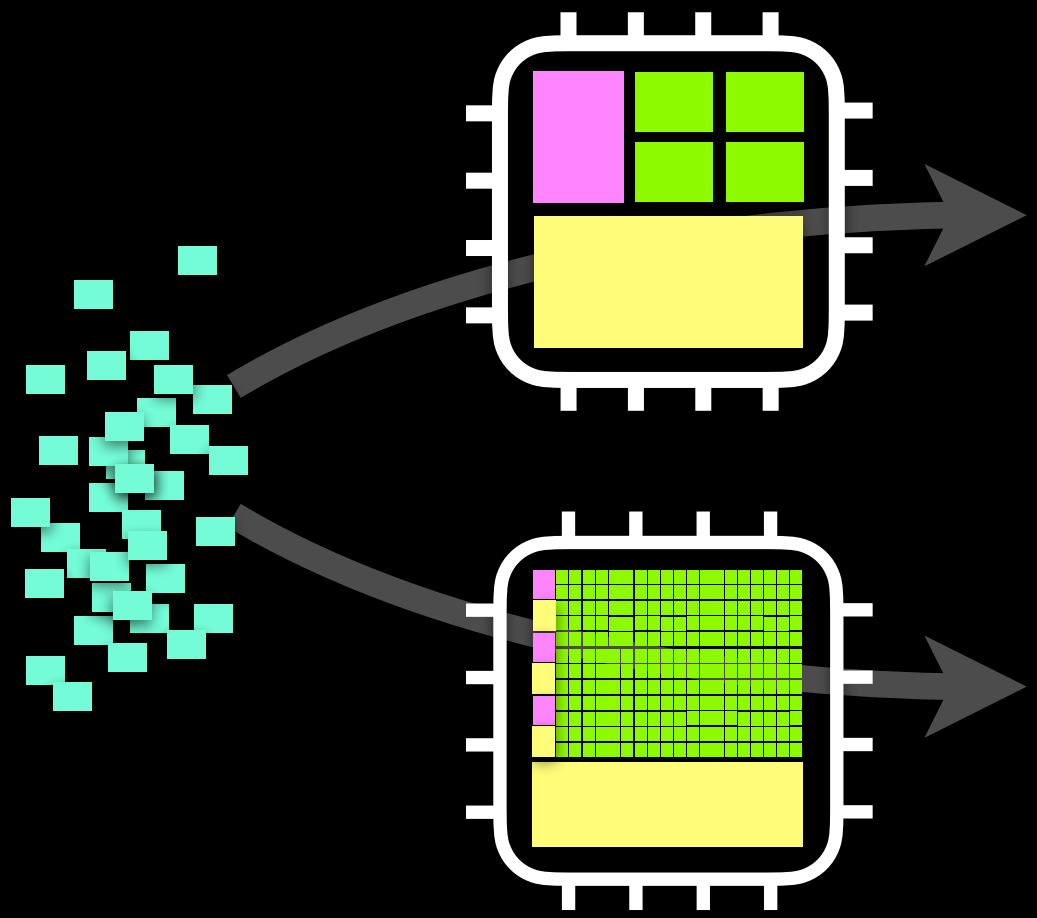


Parallelism

Some problems are worth parallelizing

Some problems may not even be possible to parallelize (serial only algorithm)

Amdahl's law

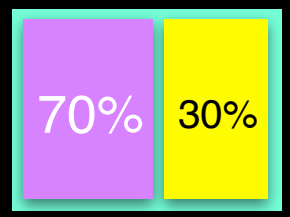


Amdahl's law

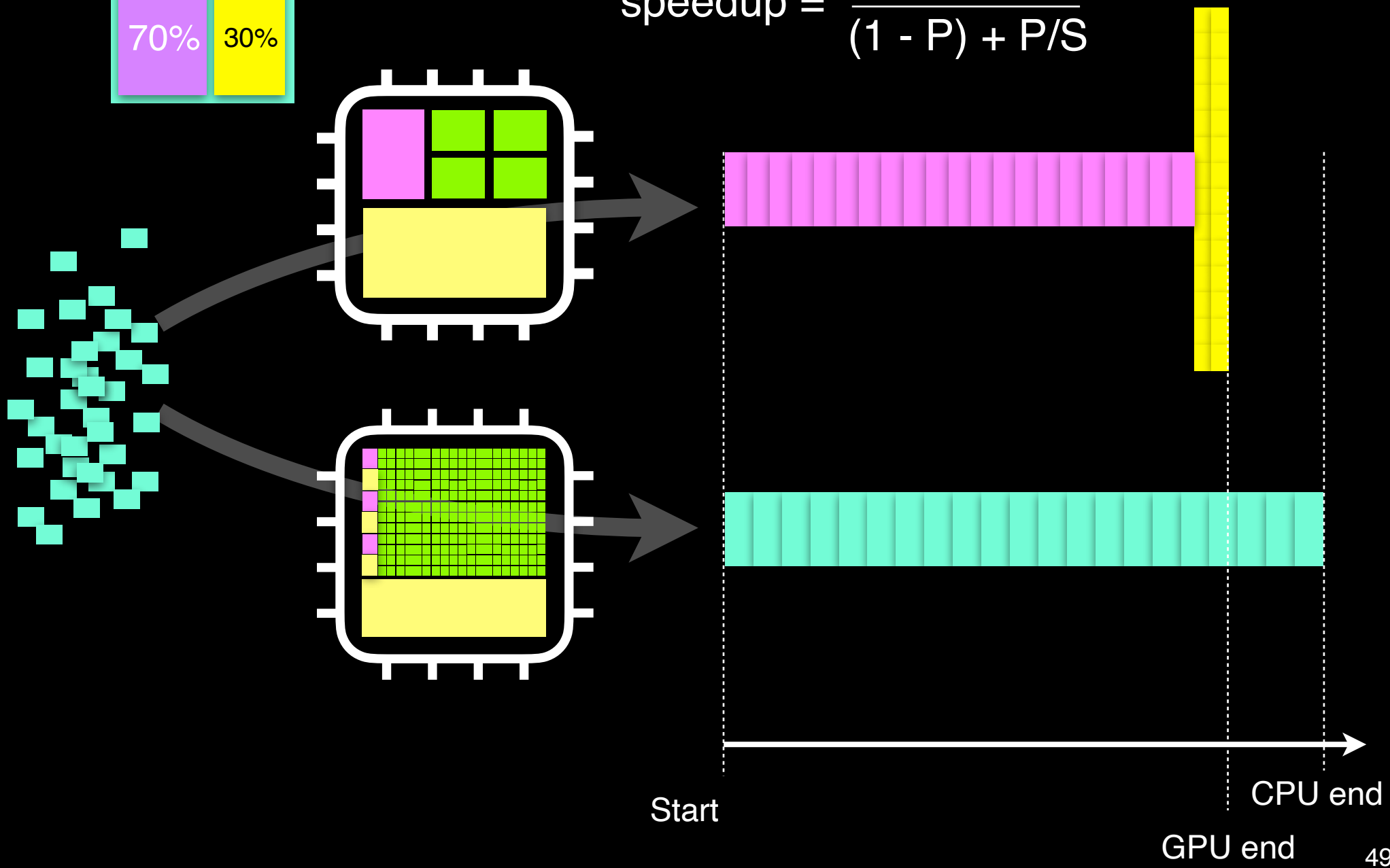
UF
Chang
Florida

Sequential
(not parallelizable)

Parallelizable



$$\text{speedup} = \frac{1}{(1 - P) + P/S}$$

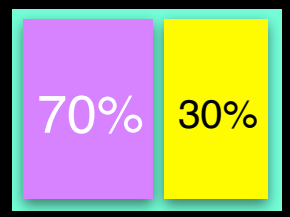


Amdahl's law

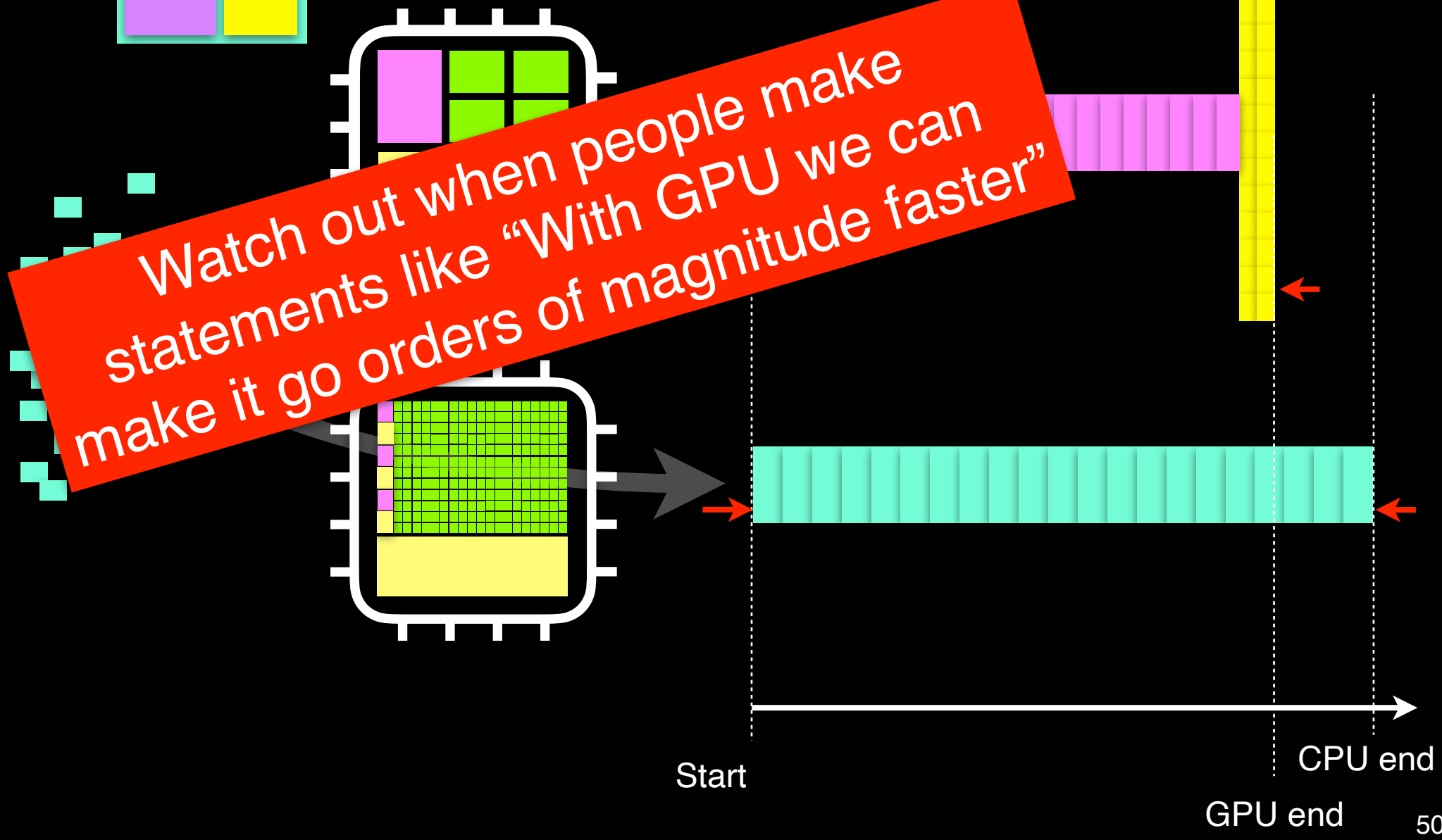
UF Chang Florida

Sequential (not parallelizable)

Parallelizable



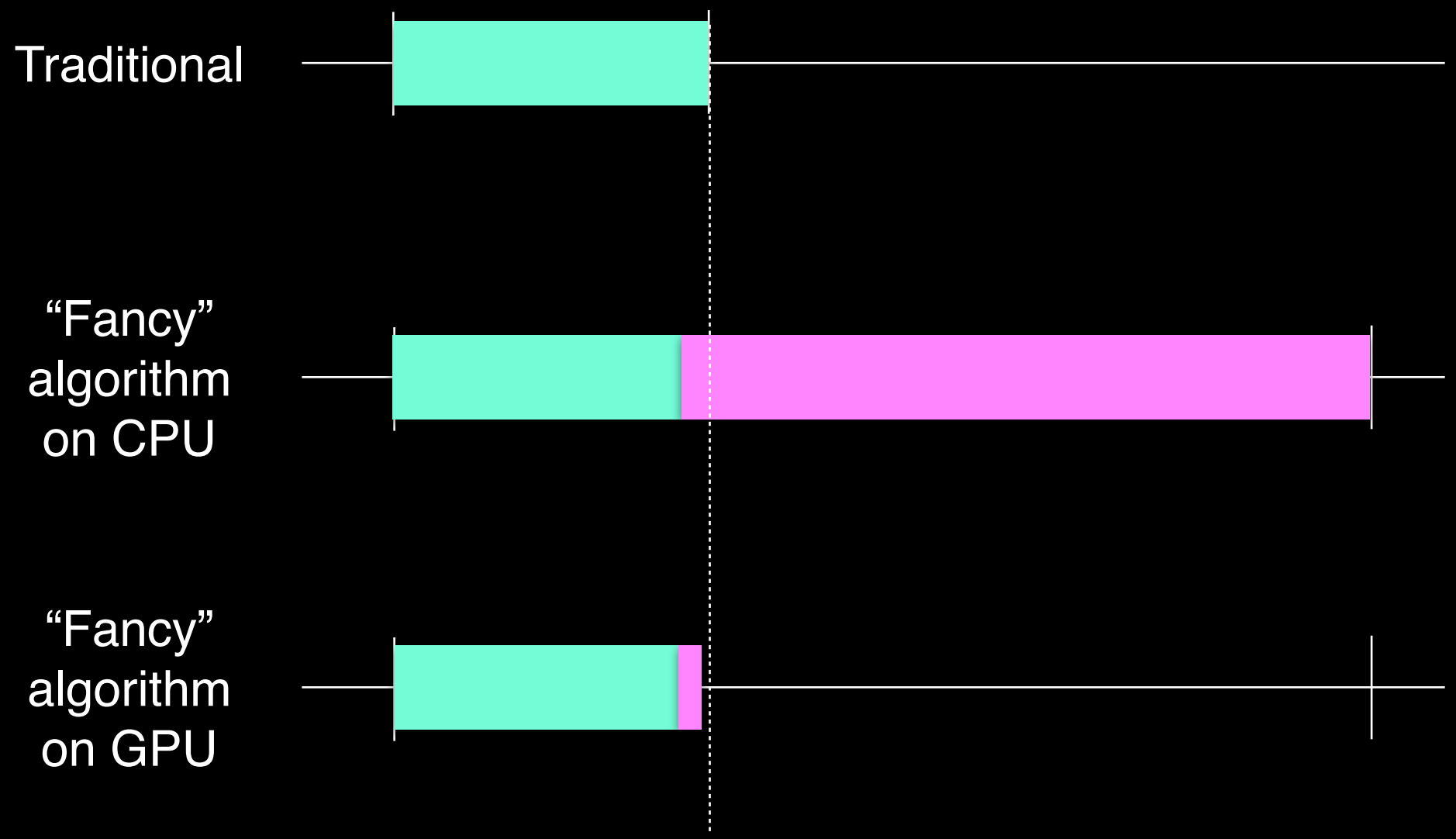
$$\text{speedup} = \frac{1}{(1 - P) + P/S}$$



Watch out when people make statements like "With GPU we can make it go orders of magnitude faster"

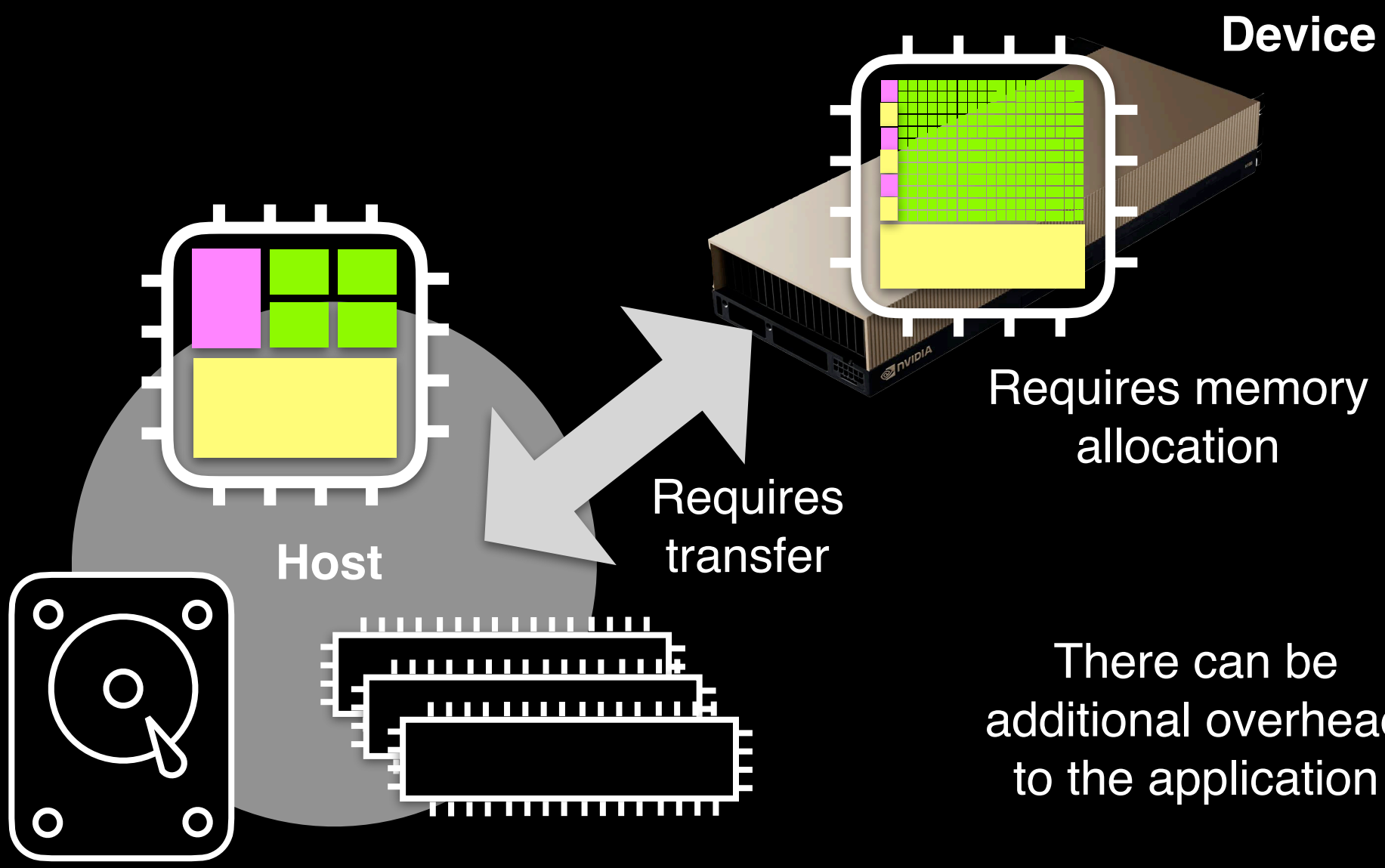
Moving the goal post

UF
Chang
Florida



Overhead

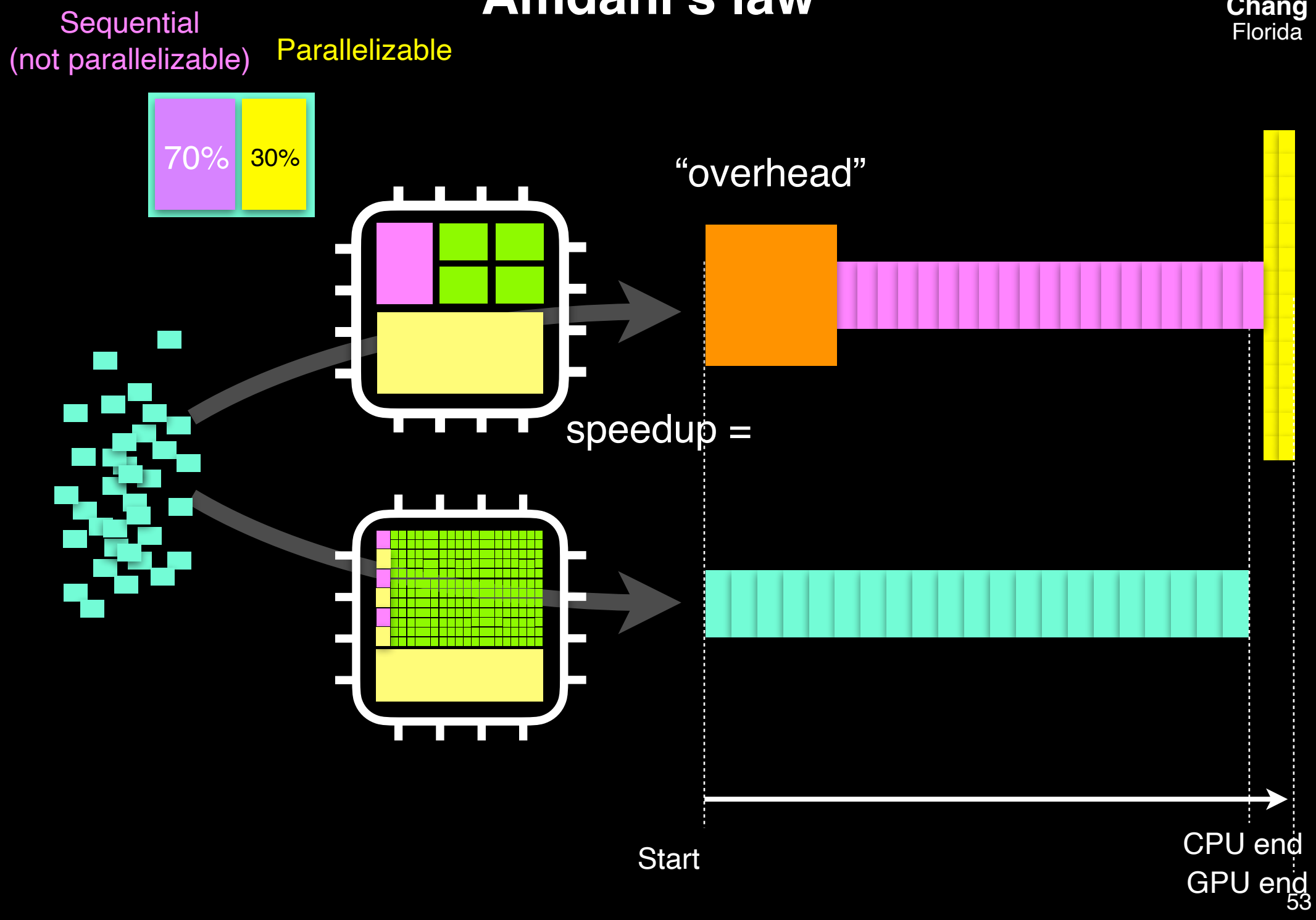
UF
Chang
Florida



There can be additional overhead to the application

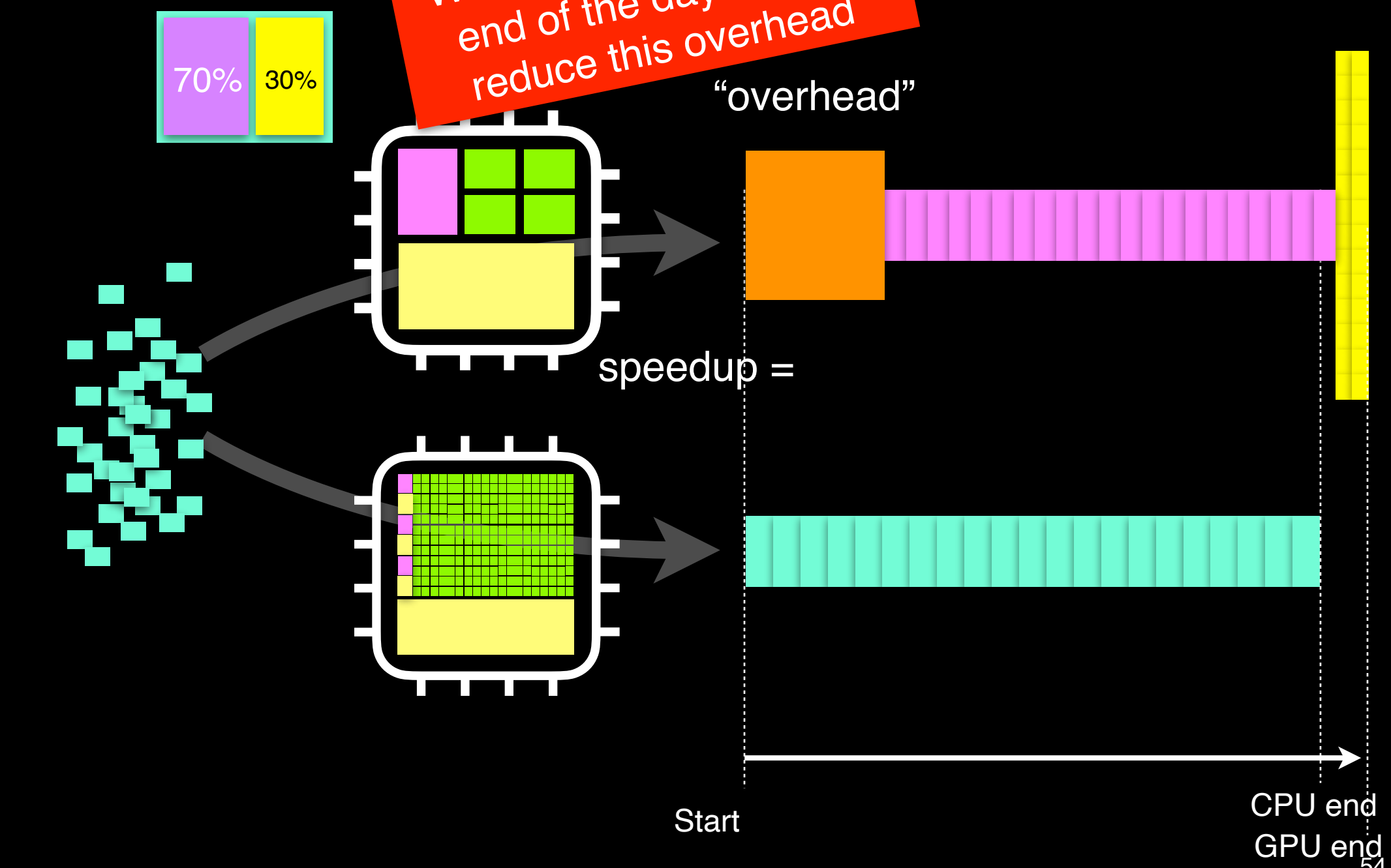
Amdahl's law

UF
Chang
Florida



Amdahl

We'll come back at then end of the day how to reduce this overhead



Some problems are worth parallelizing

Some problems may not even be possible to parallelize (serial only algorithm)

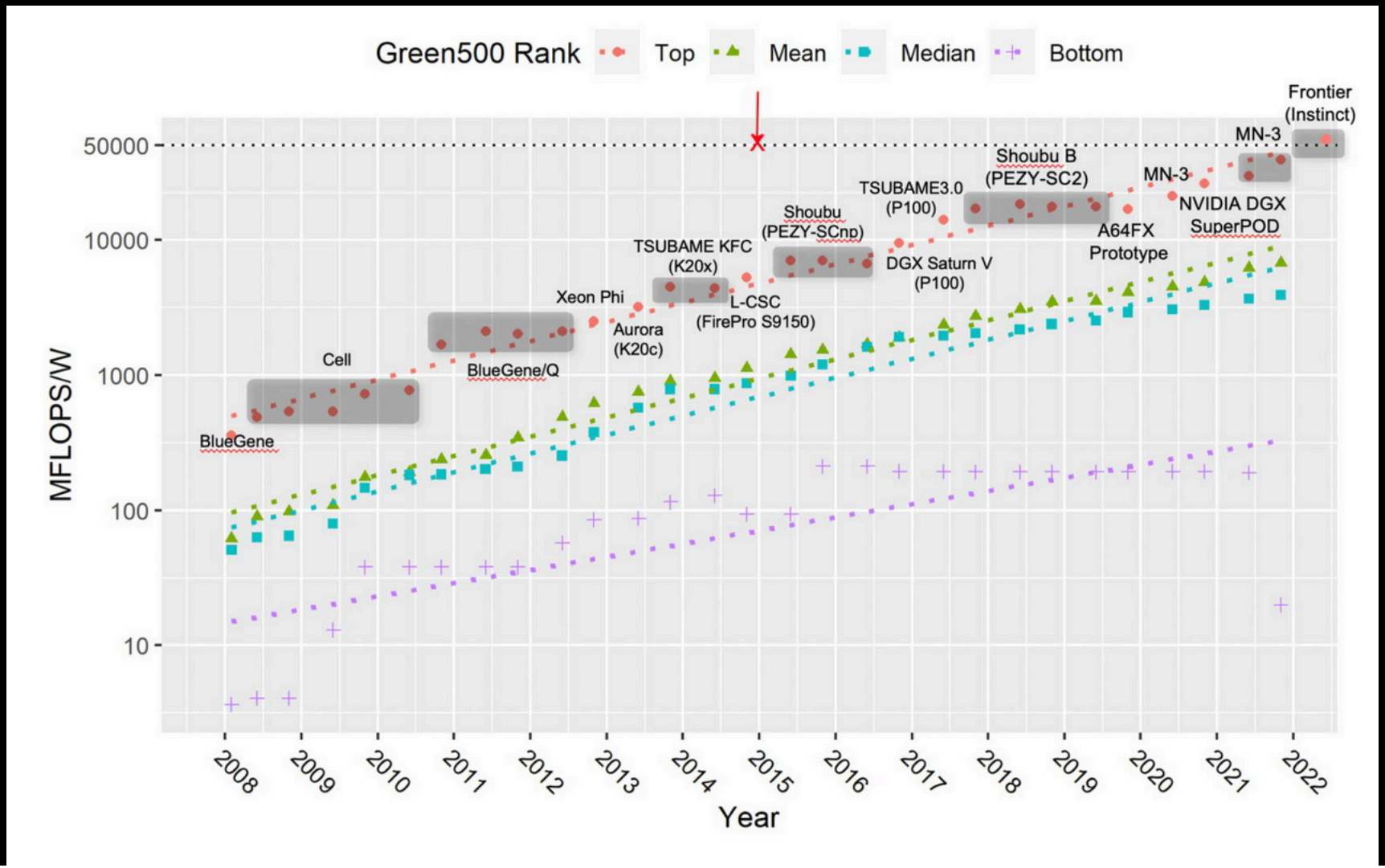
Some problems require “new algorithm” and then accelerating via GPU (prediction of how good this will be is not easy to predict!)

GPUs are going to be more and more ubiquitous

But if we don't create algorithms for GPU we would not be able to use them at all

(Chance of winning a lottery is 1 / trillion
But if you don't buy, chance is exactly 0)

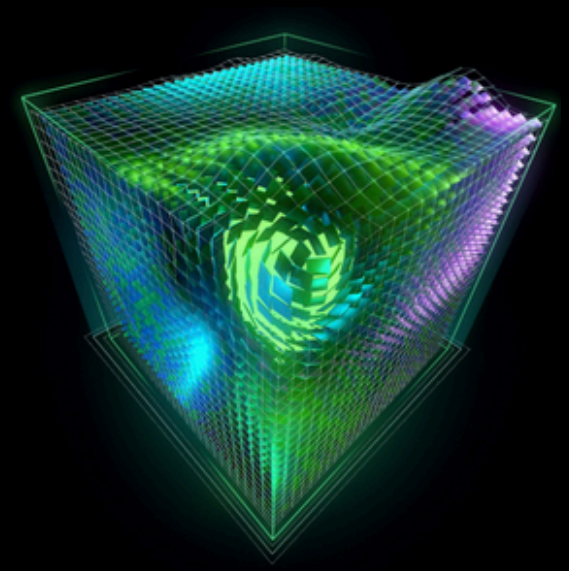
Green computing



The Green500 showed that heterogeneous systems — those with accelerators like GPUs in addition to CPUs — are consistently the most energy-efficient ones.

Alright fine! So how do I code?

UF
Chang
Florida



NVIDIA®
CUDA®

Computer Unified Device Architecture (CUDA)

Compute Unified Device Architecture (CUDA) is a proprietary[1] parallel computing platform and application programming interface (API) that allows software to use certain types of graphics processing units (GPUs) for accelerated general-purpose processing, an approach called general-purpose computing on GPUs (GPGPU). CUDA API and its runtime: The CUDA API is an extension of the C programming

We use CUDA

CUDA enabled GPUs

UF Chang Florida

CUDA-Enabled Datacenter Products

Tesla Workstation Products

GPU	Compute Capability
Tesla K80	3.7
Tesla K40	3.5
Tesla K20	3.5
Tesla C2075	2.0
Tesla C2050/C2070	2.0

NVIDIA Data Center Products

GPU	Compute Capability
NVIDIA H100	9.0
NVIDIA L4	8.9
NVIDIA L40	8.9
NVIDIA A100	8.0
NVIDIA A40	8.6
NVIDIA A30	8.0
NVIDIA A10	8.6
NVIDIA A16	8.6
NVIDIA A2	8.6
NVIDIA T4	7.5
NVIDIA V100	7.0
Tesla P100	6.0
Tesla P40	6.1
Tesla P4	6.1
Tesla M60	5.2
Tesla M40	5.2
Tesla K80	3.7
Tesla K40	3.5
Tesla K20	3.5
Tesla K10	3.0

https://developer.nvidia.com/cuda-gpus

Introduction of CUDA programming

What you write:

It is an extension of C/C++ programming

Very minimal difference from C/C++ programming

What you do:

Most of the time the steps are very similar

❶ Copy input data to GPU from the host

❷ Execute the code on GPU

❸ Retrieve the output data from GPU back to host

First let's try to do the vector addition

A [0]		B [0]		C [0]
A [1]		B [1]		C [1]
	+		=	
A [2]		B [2]		C [2]
A [3]		B [3]		C [3]

~~First let's try to do the vector addition~~

A [0]	B [0]	C [0]
A [1]	B [1]	C [1]
+		=
A [2]	B [2]	C [2]
A [3]	B [3]	C [3]

~~First let's try to do the vector addition~~

Repeat many times

A [0]	B [0]	C [0]
A [1]	B [1]	C [1]
+		=
A [2]	B [2]	C [2]
A [3]	B [3]	C [3]

Our approach today

We will first approach the CPU example and then GPU

Focusing on the bottomline i.e. *speed up*

Computing setup

UF Chang Florida

binder

Turn a Git repo into a collection of interactive notebooks

Have a repository full of Jupyter notebooks? With Binder, open those notebooks in an executable environment, making your code immediately reproducible by anyone, anywhere.

New to Binder? Get started with a Zero-to-Binder tutorial in Julia, Python, or R.

Build and launch a repository

GitHub repository name or URL

GitHub | https://github.com/davidlange6/courses-hsf-india-december2023

Git ref (branch, tag, or commit)

gpu

Path to a notebook file (optional)

Path to a notebook file (optional) | File

Advanced setting - sites and resource customizations

launch

Take me to Jupyterhub

Copy the URL below and share your Binder with others:

https://binderhub.ssl-hep.org/v2/gh/davidlange6/courses-hsf-india-december2023/gpu

Expand to see the text below, paste it into your README to show a binder badge: launch binder

Already built! Launching

nvidia-smi

UF Chang Florida

nvidia-smi: NVIDIA System Management Interface program

- Command line utility
- Aids in the management and monitoring of NVIDIA GPU devices

jovyan@jupyter-p-2echang

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce GTX 1080 Ti     Off |   00000000:DB:00.0 Off |                  N/A |
| 28%   30C    P8              8W /  250W |       0MiB /  11264MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
jovyan@jupyter-p-2echang-40ufl-2eedu--research-2ds
jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2d:~$
```

Simple 1 0 Mem: 156.00 ... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindi... 0

nvcc / .cu file

Compiling a CUDA program is similar to compiling a C/C++ program.

Cuda code should be typically stored in a file with extension .cu

NVIDIA provides a CUDA compiler called **nvcc**

nvcc is called for CUDA parts

gcc is called for c++ parts

nvcc converts .cu files into C++ for the host system and

CUDA assembly or binary instructions for the device

Our first program will be….

We will take a vector of size 10 million!
and add them

And we will just repeat this 1000 times
(for no good reason)

Vim

UF Chang Florida

I use Vim extensively so I am going to start with installing Vim

```
$ conda create -n env
$ conda activate env
$ conda install vim
```

Set up vim basic settings

```
$ vim ~/.vimrc
```

press `i` to edit

press `esc` to quit edit

type `:wq` to quit editor

```
:imap jk <Esc>
syntax on
set tabstop=8
set softtabstop=4
set shiftwidth=4
set expandtab
set cmdheight=2
set ruler
set hlsearch
set wildmenu
set number
set scrolloff=40
set nocursorcolumn
set nowrap
set list
set listchars=tab:>-
colorscheme delek
```

Create workdir

UF
Chang
Florida

```
$ mkdir workdir
$ cd workdir
```

First we will create an empty main function

UF Chang Florida

```
$ vim vadd_host.cu
```

```
i
```

```
#include <iostream>

int main()
{
    // banner
    std::cout << "##########################" << std::endl;
    std::cout << "#                        #" << std::endl;
    std::cout << "# Vector Addition Prog. #" << std::endl;
    std::cout << "#        (CPU)           #" << std::endl;
    std::cout << "#                        #" << std::endl;
    std::cout << "##########################" << std::endl;

    return 0;
}
```

```
esc
:wq
```

Save and compile the program

UF Chang Florida

Save to `vadd_host.cu`

Compile via

```
$ nvcc vadd_host.cu -o vadd_host
```

Execute via

```
$ ./vadd
```

```
jovyan@jupyter-p-2echang

(env) jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2d:~/workdir$ ./vadd_host
#########################
#                       #
# Vector Addition Prog. #
#       (CPU)           #
#                       #
#########################
(env) jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2d:~/workdir$
```

We will first try the following coding

UF Chang Florida

Initialize two size-N vectors in host (Allocate and Set)

```
float* A_host = new float[n_data];
float* B_host = new float[n_data];
float* C_host = new float[n_data];
```

```
for (int i = 0; i < n_data; ++i)
{
    A_host[i] = i;
    B_host[i] = i * pow(-1, i);
}
```

For loop and sum them to compute the sum of two vectors

```
for (int i_data = 0; i_data < n_data; ++i_data)
{
    C_host[i_data] = A_host[i_data] + B_host[i_data];
}
```

We will first try the following coding

UF Chang Florida

Initialize two size-N vectors in host (Allocate and Set)

```
float* A_host = new float[n_data];
float* B_host = new float[n_data];
float* C_host = new float[n_data];
```

```
for (int i = 0; i < n_data; ++i)
{
    A_host[i] = i;
    B_host[i] = i * pow(-1, i);
}
```

For loop and sum them to compute the sum of two vectors

```
for (int i_data = 0; i_data < n_data; ++i_data)
{
    for (int iop = 0; iop < n_ops; ++iop)
        C_host[i_data] = A_host[i_data] + B_host[i_data];
}
```

Some tools

UF Chang Florida

C++ based timing tool

```
#include <chrono>
using namespace std::chrono;

…

auto start = high_resolution_clock::now();

…

auto end = high_resolution_clock::now();

…

float time = duration_cast<microseconds>(end - start).count() / 1000.;
```

measures time between

Some tools

UF Chang Florida

CUDA based Timing tool

```
// create event
cudaEvent_t startEvent, stopEvent;
cudaEventCreate(&startEvent);
cudaEventCreate(&stopEvent);

// float to read out time
float ms;

// before start
cudaEventRecord(startEvent, 0);
…
…
…
// end start
cudaEventRecord(stopEvent, 0);
cudaEventSynchronize(stopEvent);

// get the time
cudaEventElapsedTime(&ms, startEvent, stopEvent);
```

vadd_host.cu

UF Chang Florida

```
#include <iostream>
#include <chrono>
using namespace std::chrono;


int main()
{
    // banner
    std::cout << "#########################" << std::endl;
    std::cout << "#                       #" << std::endl;
    std::cout << "# Vector Addition Prog. #" << std::endl;
    std::cout << "#        (CPU)          #" << std::endl;
    std::cout << "#                       #" << std::endl;
    std::cout << "#########################" << std::endl;

    int n_data = 10000000;
    int n_ops = 1000;

    auto start = high_resolution_clock::now();

    float* A_host = new float[n_data];
    float* B_host = new float[n_data];
    float* C_host = new float[n_data];

    for (unsigned int i = 0; i < n_data; ++i)
    {
        A_host[i] = i;
        B_host[i] = i * pow(-1, i);
    }

    for (int i_data = 0; i_data < n_data; ++i_data)
    {
        for (int iop = 0; iop < n_ops; ++iop)
            C_host[i_data] = A_host[i_data] + B_host[i_data];
    }

    auto end = high_resolution_clock::now();

    float time = duration_cast<microseconds>(end - start).count() / 1000.;

    std::cout << "time: " << time << std::endl;

    return 0;
}
```

Title

UF
Chang
Florida

More refined version here:

https://raw.githubusercontent.com/sgnoohc/hsf-india-examples/main/vadd_host.cu

Result

UF Chang Florida

jovyan@jupyter-p--researc

```
(myenv) jovyan@jupyter-p--research-2dsoftwa-2---dindia-2dmay2024-2df6chacf8:~/hsf-india-examples$ ./vadd_host
################################
#                              #
#    Vector Addition Program   #
#            (CPU)             #
#                              #
################################
 --- Input data ---
 n_data = 10000000
 n_ops  = 1000
 --- Sanity Check ---
 Printing last 10 result
 i: 9999990 C_host[i]: 2e+07
 i: 9999991 C_host[i]: 0
 i: 9999992 C_host[i]: 2e+07
 i: 9999993 C_host[i]: 0
 i: 9999994 C_host[i]: 2e+07
 i: 9999995 C_host[i]: 0
 i: 9999996 C_host[i]: 2e+07
 i: 9999997 C_host[i]: 0
 i: 9999998 C_host[i]: 2e+07
 i: 9999999 C_host[i]: 0

 --- Timing information ---
 time inititalizing       : 148.694 ms
 time executing on CPU    : 15461 ms
 -------------------------:
 time total               : 15609.7 ms
 speedup_ceiling: 104.978
```

We are adding a vector of size 10M

We are performing addition 1000 times (but we take final result of adding once)

it is an unrealistic situation…

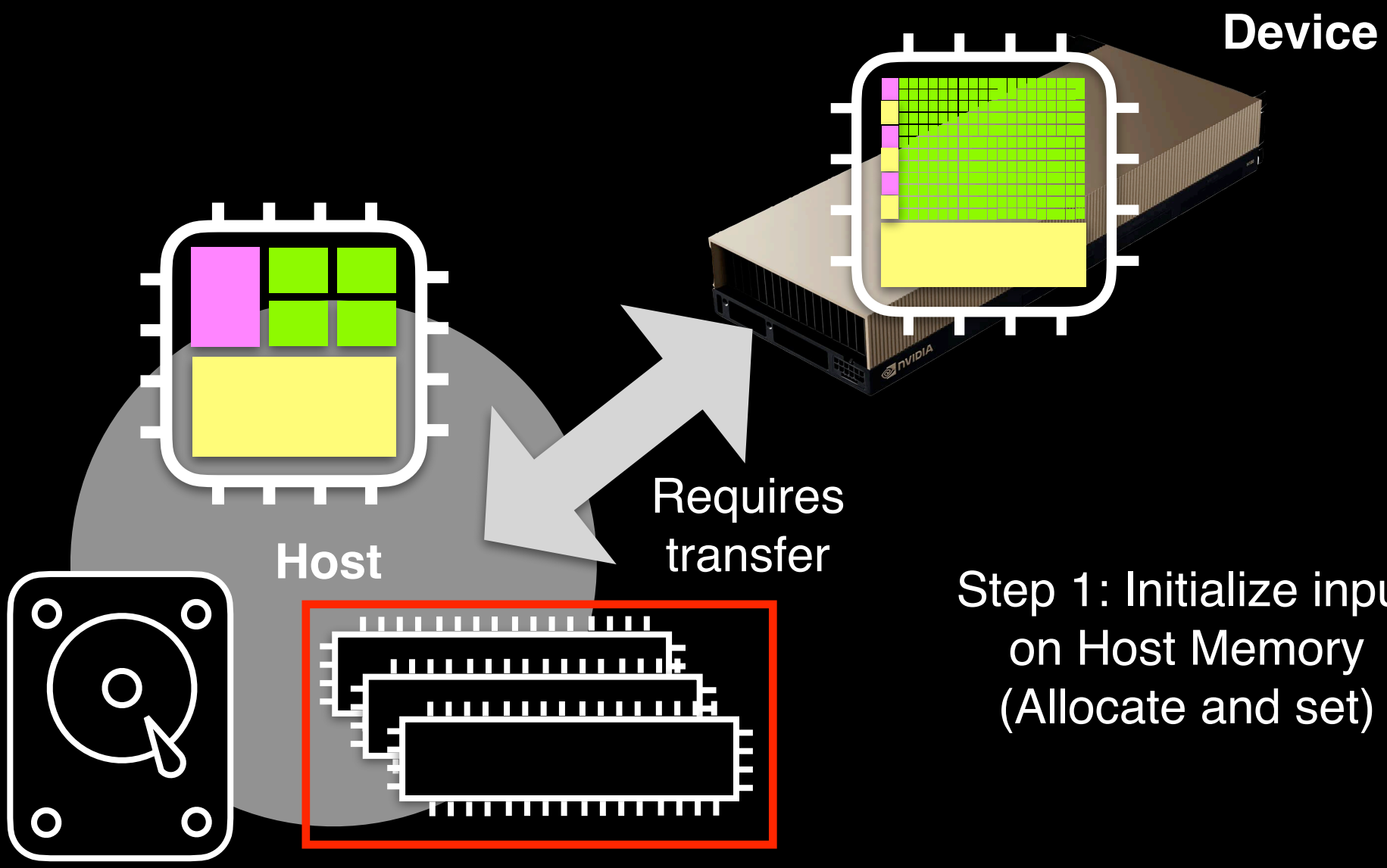
Time it took to create 10M length vectors

Time it took to perform addition 1000 times

OK let's do the same thing on GPU

First let's try to do the vector addition

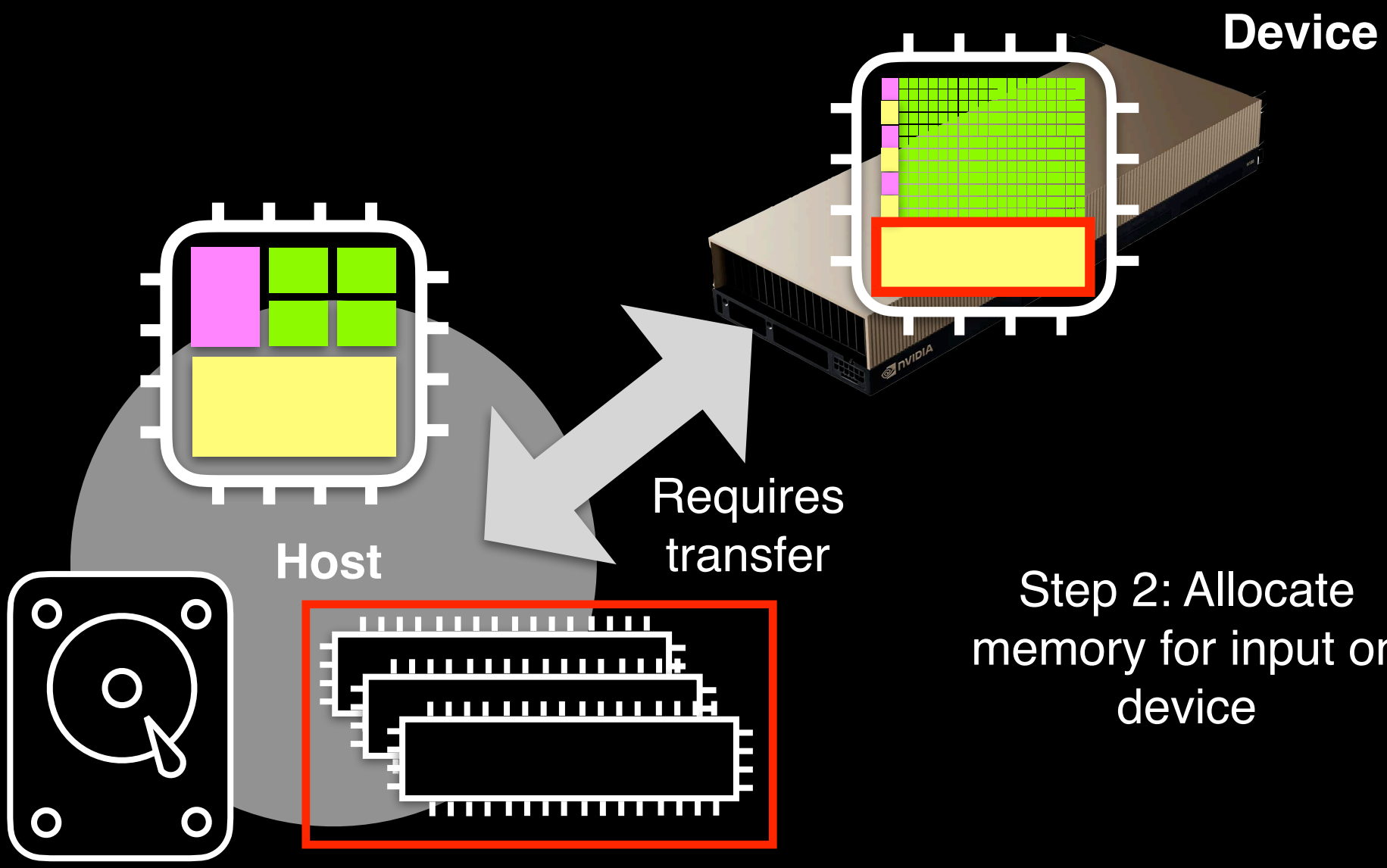
UF
Chang
Florida



Step 1: Initialize input on Host Memory (Allocate and set)

First let's try to do the vector addition

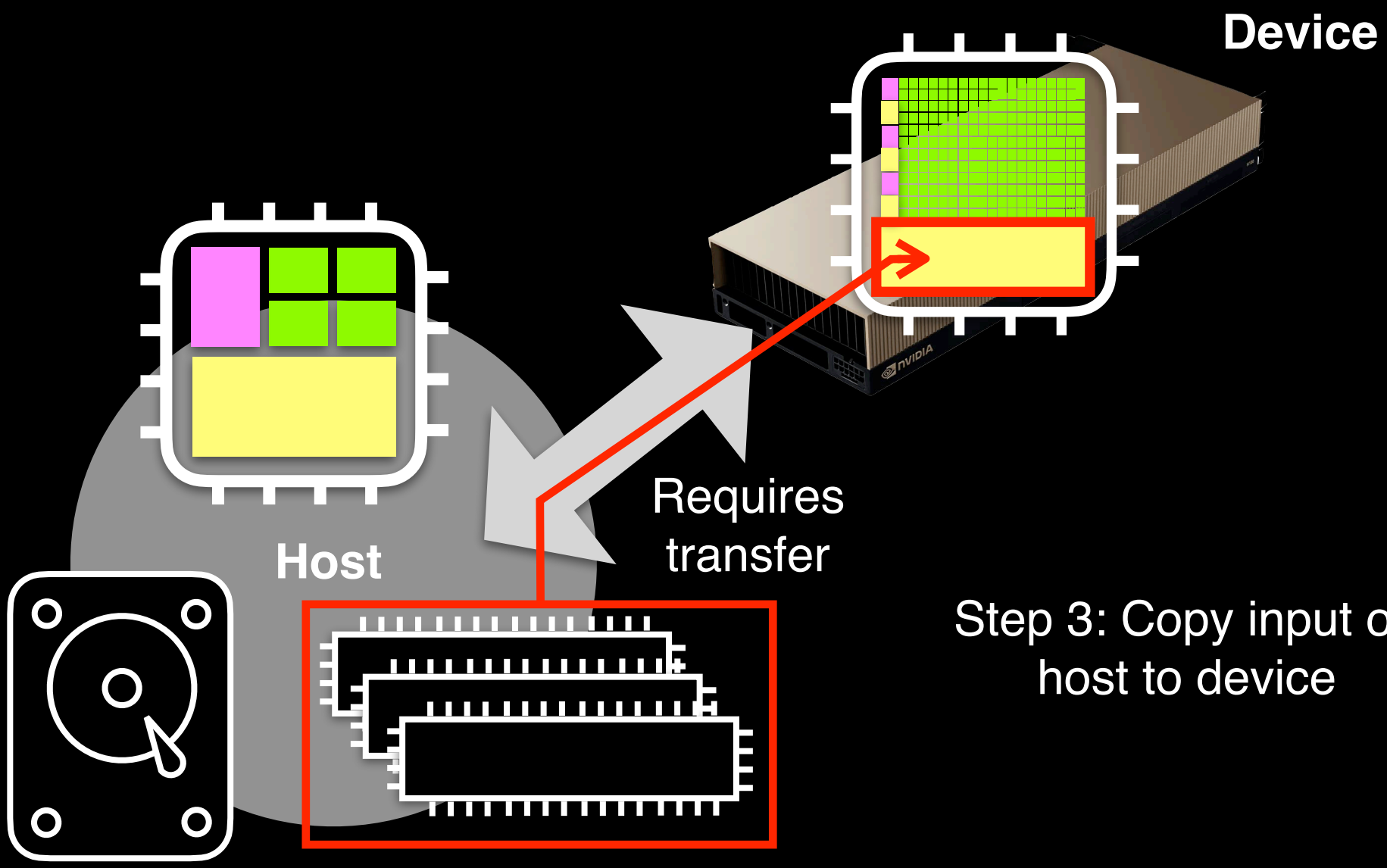
UF
Chang
Florida



Step 2: Allocate memory for input on device

First let's try to do the vector addition

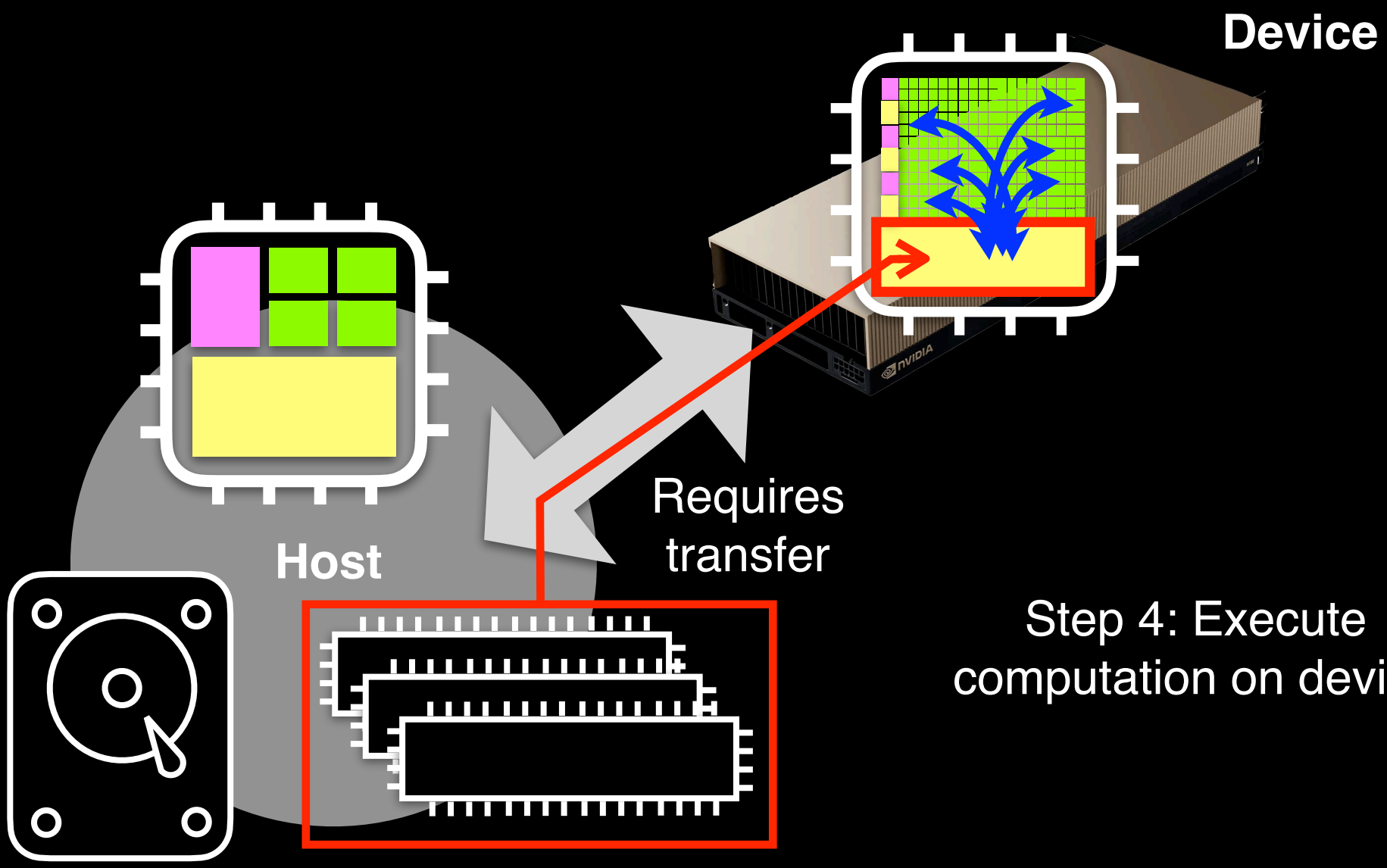
UF
Chang
Florida



Step 3: Copy input on host to device

First let's try to do the vector addition

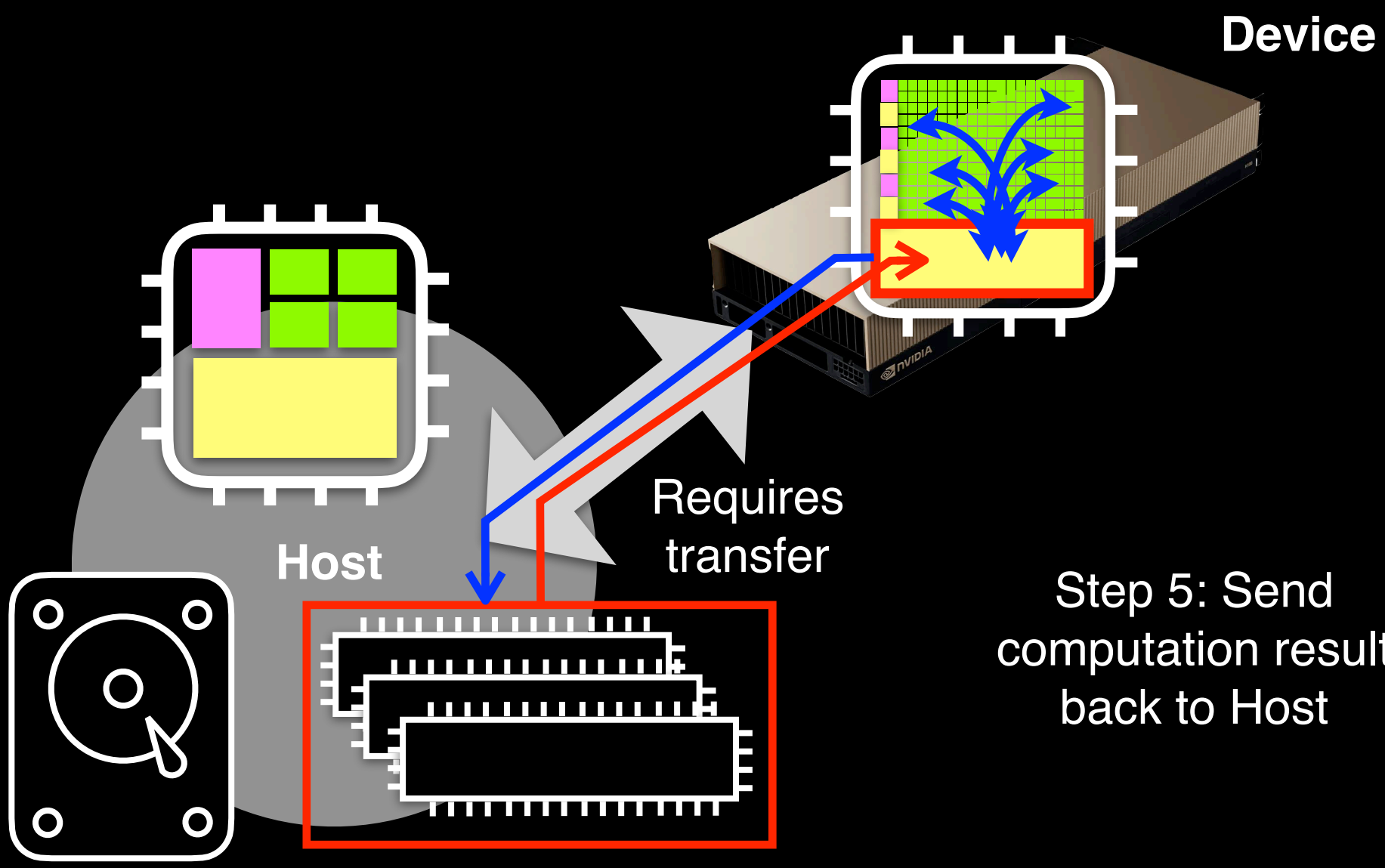
UF
Chang
Florida



Step 4: Execute computation on device

First let's try to do the vector addition

UF
Chang
Florida



Step 5: Send computation result back to Host

We will start with the same file but add

UF Chang Florida

```
#include <iostream>
#include <chrono>
using namespace std::chrono;


int main()
{
    // banner
    std::cout << "#########################" << std::endl;
    std::cout << "#                       #" << std::endl;
    std::cout << "# Vector Addition Prog. #" << std::endl;
    std::cout << "#        (CPU)          #" << std::endl;
    std::cout << "#                       #" << std::endl;
    std::cout << "#########################" << std::endl;

    int n_data = 10000000;
    int n_ops = 1000;

    auto start = high_resolution_clock::now();

    float* A_host = new float[n_data];
    float* B_host = new float[n_data];
    float* C_host = new float[n_data];

    for (unsigned int i = 0; i < n_data; ++i)
    {
        A_host[i] = i;
        B_host[i] = i * pow(-1, i);
    }

    for (int i_data = 0; i_data < n_data; ++i_data)
    {
        for (int iop = 0; iop < n_ops; ++iop)
            C_host[i_data] = A_host[i_data] + B_host[i_data];
    }

    auto end = high_resolution_clock::now();

    float time = duration_cast<microseconds>(end - start).count() / 1000.;

    std::cout << "time: " << time << std::endl;

    return 0;
}
```

I will add below the same thing but in GPU version

Allocating memory in GPU

UF Chang Florida

Create pointers to the memory on device GPU

```
float* A_device;
float* B_device;
float* C_device;
```

Allocate memory on device GPU (the pointer points to GPU memory)

```
cudaMalloc((void**) &A_device, n_data * sizeof(float));
cudaMalloc((void**) &B_device, n_data * sizeof(float));
cudaMalloc((void**) &C_device, n_data * sizeof(float));
```

This is to pass it as generic pointer

Define the details

Above looks confusing but it's nothing more than following in GPU

" float A_device = new float[N_data] "*

UF Chang Florida

tests - JupyterLab | raw.githubusercontent.com/sgnooh

https://jupyterhub.ssl-hep.org/user/p.chang@ufl.

File Edit View Run Kernel Tabs Settings Help

Filter files by name

/ tests /

Name	Last Modified
cudnn_tes...	2 days ago
hello.cu	2 days ago
run_tests.sh	2 days ago
vadd.cu	2 days ago

jovyan@jupyter-p-2echang

```
26      {
27          A_host[i] = i;
28          B_host[i] = i * pow(-1, i);
29      }
30
31      for (int i_data = 0; i_data < n_data; ++i_data)
32      {
33          for (int iop = 0; iop < n_ops; ++iop)
34              C_host[i_data] = A_host[i_data] + B_host[i_data];
35      }
36
37      auto end = high_resolution_clock::now();
38
39      float time = duration_cast<microseconds>(end - start).count
40
41      std::cout << "time: " << time << std::endl;
42
43      //~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*
44      //==============================================================
45      //~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*
46
47      // GPU VERSION
48
49      // First declare some pointers
50      float* A_device;
51      float* B_device;
52      float* C_device;
53
54      cudaMalloc((void**) &A_device, n_data * sizeof(float));
55      cudaMalloc((void**) &B_device, n_data * sizeof(float));
56      cudaMalloc((void**) &C_device, n_data * sizeof(float));
57
58      return 0;
59  }
                                                    59,1          Bot
```

Simple 1 0 Mem: 166.40... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2d... 0

Check whether it compiles

Now we set the memory

UF Chang Florida

As any other memory once you create them we need to use it

```
Normally, we'd do:

      *A_device[0] = 1.0;
      *A_device[1] = 2.0;
      …
      …
      *A_device[N] = var;
```

But we do not have a way to access them from the host directly
So we use following CUDA API to set the memory via copying content from host to device

set

```
cudaMemcpy(A_device, A_host, n_data * sizeof(float), cudaMemcpyHostToDevice);
```

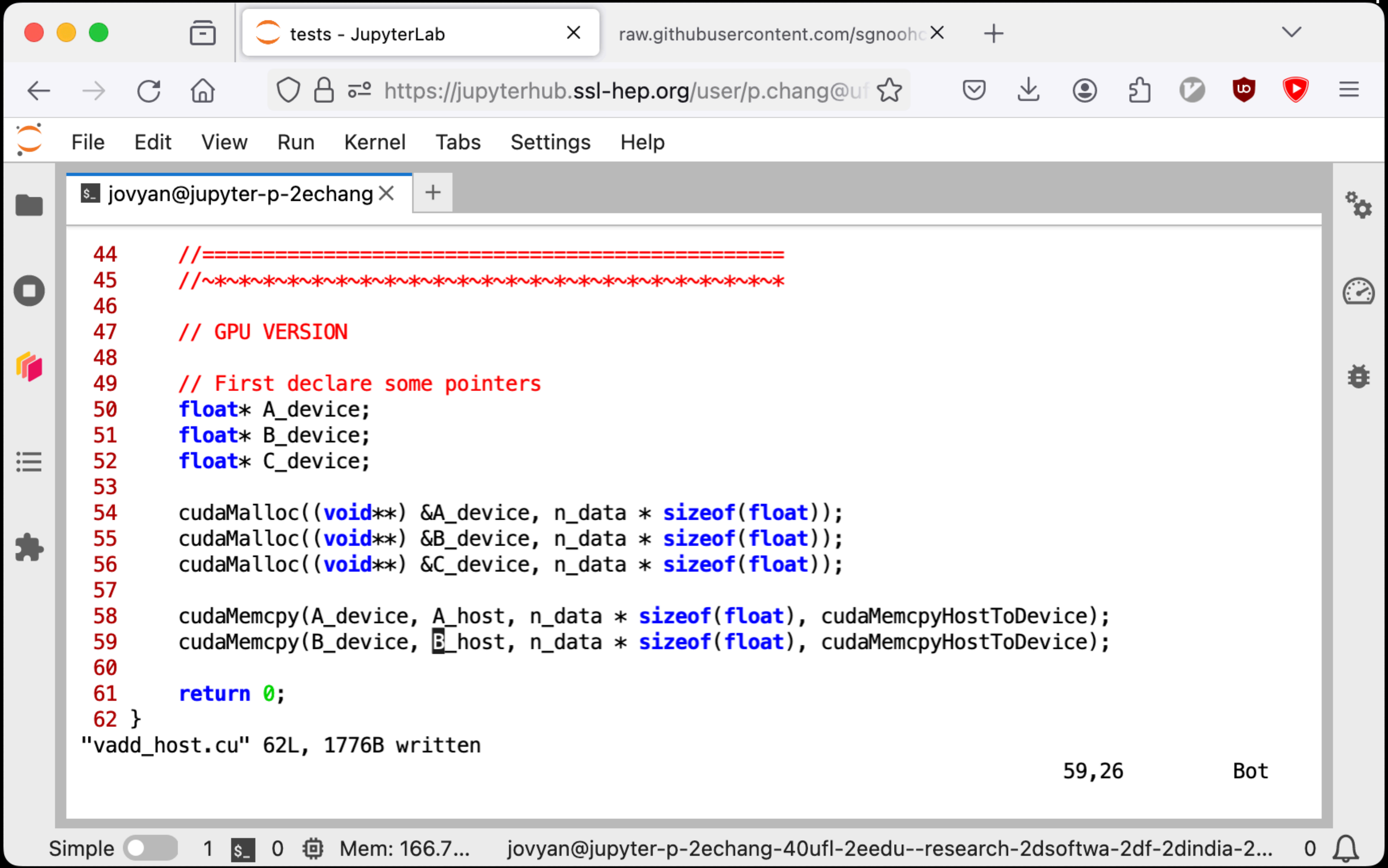
Above looks confusing but it's "kind of like" the following

" A_device = A_host; "

(not quite but something like that)

Copy the inputs to GPU

Title



```
    //================================================
    //~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*

    // GPU VERSION

    // First declare some pointers
    float* A_device;
    float* B_device;
    float* C_device;

    cudaMalloc((void**) &A_device, n_data * sizeof(float));
    cudaMalloc((void**) &B_device, n_data * sizeof(float));
    cudaMalloc((void**) &C_device, n_data * sizeof(float));

    cudaMemcpy(A_device, A_host, n_data * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(B_device, B_host, n_data * sizeof(float), cudaMemcpyHostToDevice);

    return 0;
}
"vadd_host.cu" 62L, 1776B written
                                                              59,26         Bot
```

Title

UF
Chang
Florida

Check whether it compiles

Telling GPU to execute some tasks

UF Chang Florida

Now we can't directly access the memory content on device from host
So how do we execute and perform tasks using them?

We use `__global__` function.

When we write a function with preamble `__global__` the function is now a function that is to be executed on the GPU device. We call these "GPU Kernels"

Following is how it would look like: (if adding only once)

```
__global__ void vec_add(const float* A, const float* B, float* C, int n_data)
{

    int i_data = blockDim.x * blockIdx.x + threadIdx.x;
    if (i_data < n_data)
    {
        C[i_data] = A[i_data] + B[i_data];
    }
    return;
}
```

Telling GPU to execute some tasks

UF Chang Florida

Now we can't directly access the memory content on device from host
So how do we execute and perform tasks using them?

We use `__global__` function.

When we write a function with preamble `__global__` the function is now a function that is to be executed on the GPU device. We call these "GPU Kernels"

Following is how it would look like: (if adding only once)

```
__global__ void vec_add(const float* A, const float* B, float* C, int n_data)
{

    int i_data = blockDim.x * blockIdx.x + threadIdx.x;
    if (i_data < n_data)
    {
        C[i_data] = A[i_data] + B[i_data];
    }
    return;
}
```

What is this…??

Telling GPU to execute some tasks

UF Chang Florida

Now we can't directly access the memory content on device from host
So how do we execute and perform tasks using them?

We use `__global__` function.

When we write a function with preamble `__global__` the function is now a function that is to be executed on the GPU device. We call these "GPU Kernels"

Following is how it would look like: (if adding only once)

```
__global__ void vec_add(const float* A, const float* B, float* C, int n_data)
{

    int i_data = blockDim.x * blockIdx.x + threadIdx.x;
    if (i_data < n_data)
    {
        C[i_data] = A[i_data] + B[i_data];
    }
    return;
}
```

What is this…??

No for loop…?

How multi-threading works in GPU

CUDA launches parallel SIMD jobs in multiple threads

"Threads" are grouped into "Blocks" or "Thread Blocks"

"Thread Blocks" are grouped into a "Grid"

How multi-threading works in GPU

CUDA launches parallel SIMD jobs in multiple threads

"Threads" are grouped into "Blocks" or "Thread Blocks"

"Thread Blocks" are grouped into a "Grid"

Huh?

How multi-threading works in GPU

UF Chang Florida

In our case, a thread would be one addition of elements

thread

A [0]	B [0]	C [0]
A [1]	B [1]	C [1]
+		=
A [2]	B [2]	C [2]
A [3]	B [3]	C [3]

How do we know?

How multi-threading works in GPU

UF Chang Florida

Because each kernel is 1 single thread

```
__global__ void vec_add(const float* A, const float* B, float* C, int n_data)
{

    int i_data = blockDim.x * blockIdx.x + threadIdx.x;
    if (i_data < n_data)
    {
        C[i_data] = A[i_data] + B[i_data];
    }
    return;
}
```

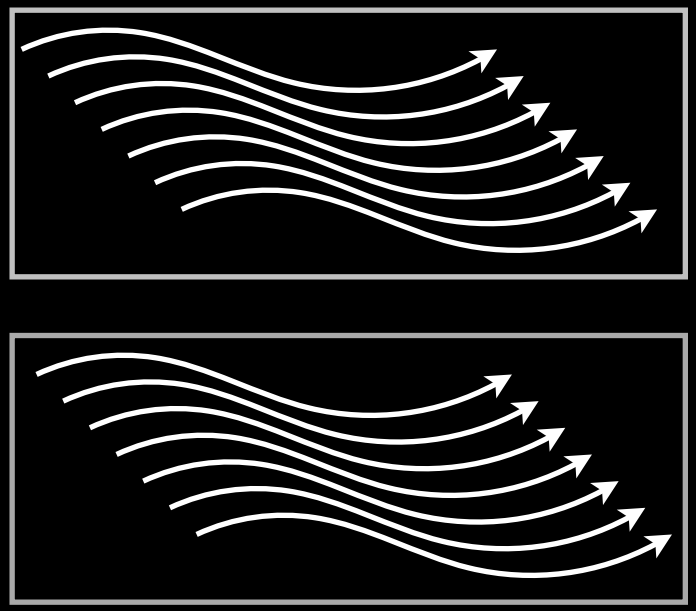
In a single thread, one sum is done between elements

How multi-threading works in GPU

UF Chang Florida

Each thread block contains multiple threads

Thread Block



usually can be up to 1024 threads per block max but depends on the GPU

A [0]	B [0]	C [0]
A [1]	B [1]	C [1]
+		=
…	…	…
A [N]	B [N]	C [N]

Question

So if we have a vector of size 500 being added with another vector of size 500 what would be the total number of threads we need?

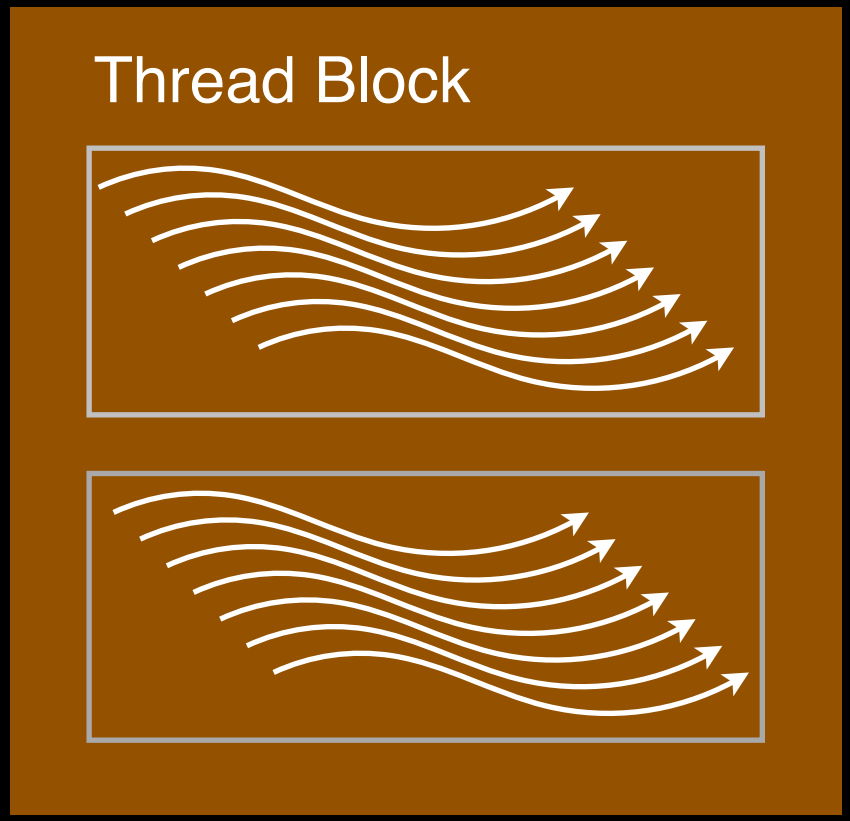
If we group each 200 threads as one thread block how many thread blocks do we need?

How multi-threading works in GPU

UF Chang Florida

Set of thread block is called a “Grid”

Grid



A [0]	B [0]	C [0]
A [1]	B [1]	C [1]
+		=
…	…	…
A [N]	B [N]	C [N]

How multi-threading works in GPU

UF
Chang
Florida

If we have N = 100000 size vector

We need total of 100000 threads

If we group them by 256 threads

We need `int((100000-0.5)/256+1)` blocks

= 391 blocks

Then, we'd say our grid has 391 blocks

A [0]	B [0]	C [0]
A [1]	B [1]	C [1]
+		=
…	…	…
A [N]	B [N]	C [N]

Thread indexing

UF Chang Florida

So if there are so many threads, how does each thread know which one elements to add?

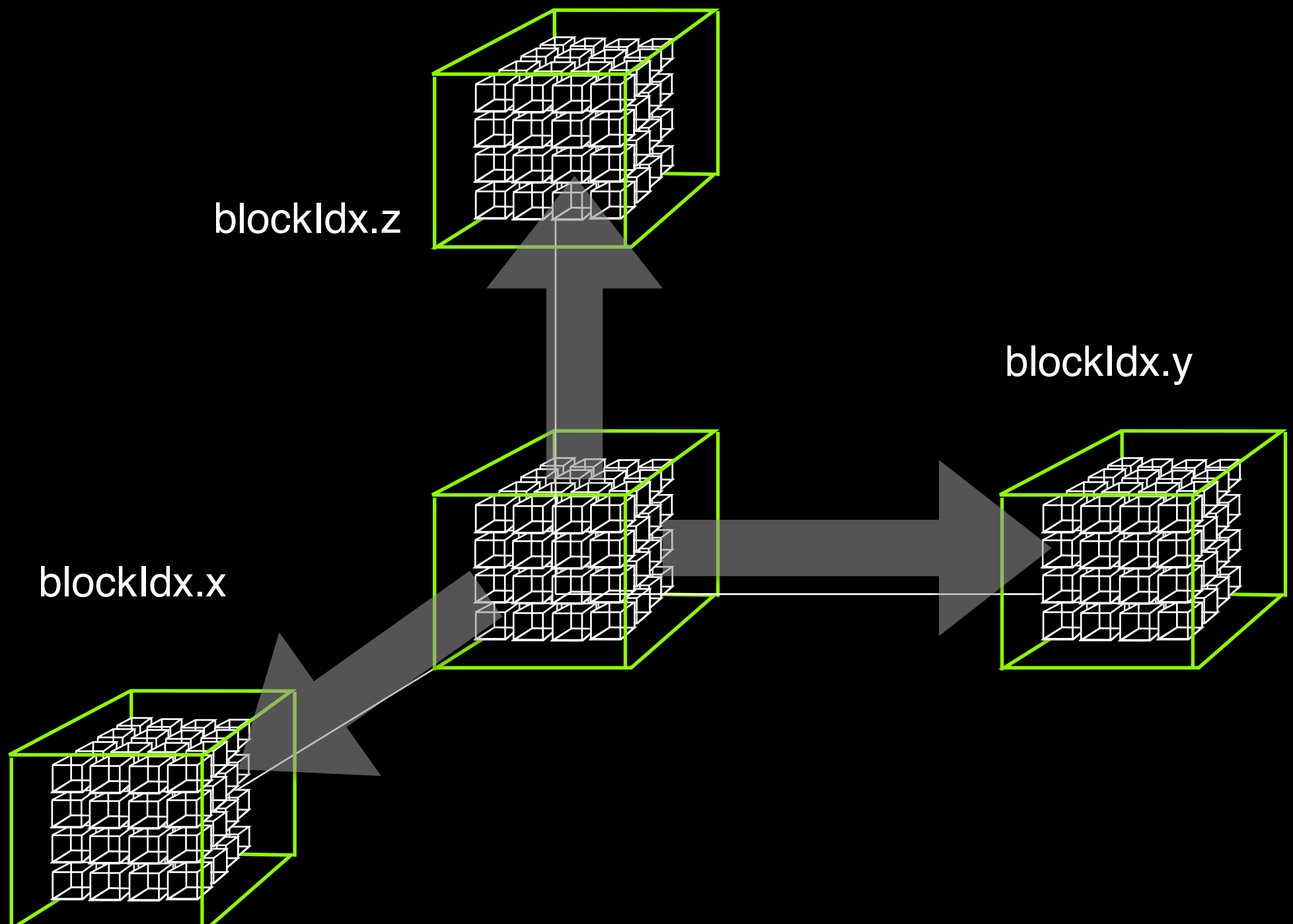
```
__global__ void vec_add(const float* A, const float* B, float* C, int n_data)
{

    int i_data = blockDim.x * blockIdx.x + threadIdx.x;
    if (i_data < n_data)
    {
        C[i_data] = A[i_data] + B[i_data];
    }
    return;
}
```

We use the following to specify which thread we want to work on and define what to do for a given thread

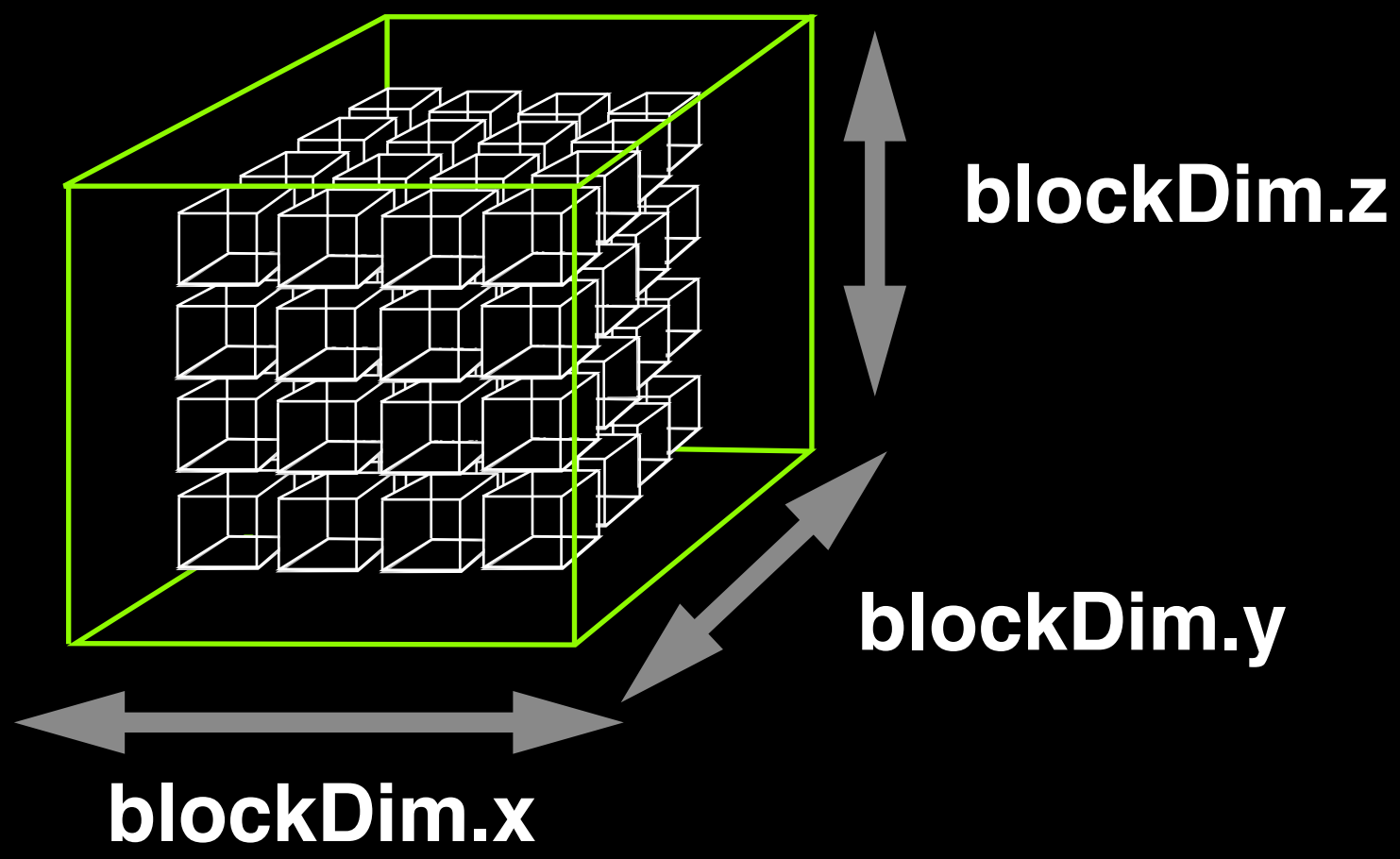
`blockDim blockIdx threadIdx`

blockIdx.x / blockIdx.y / blockIdx.z

UF
Chang
Florida

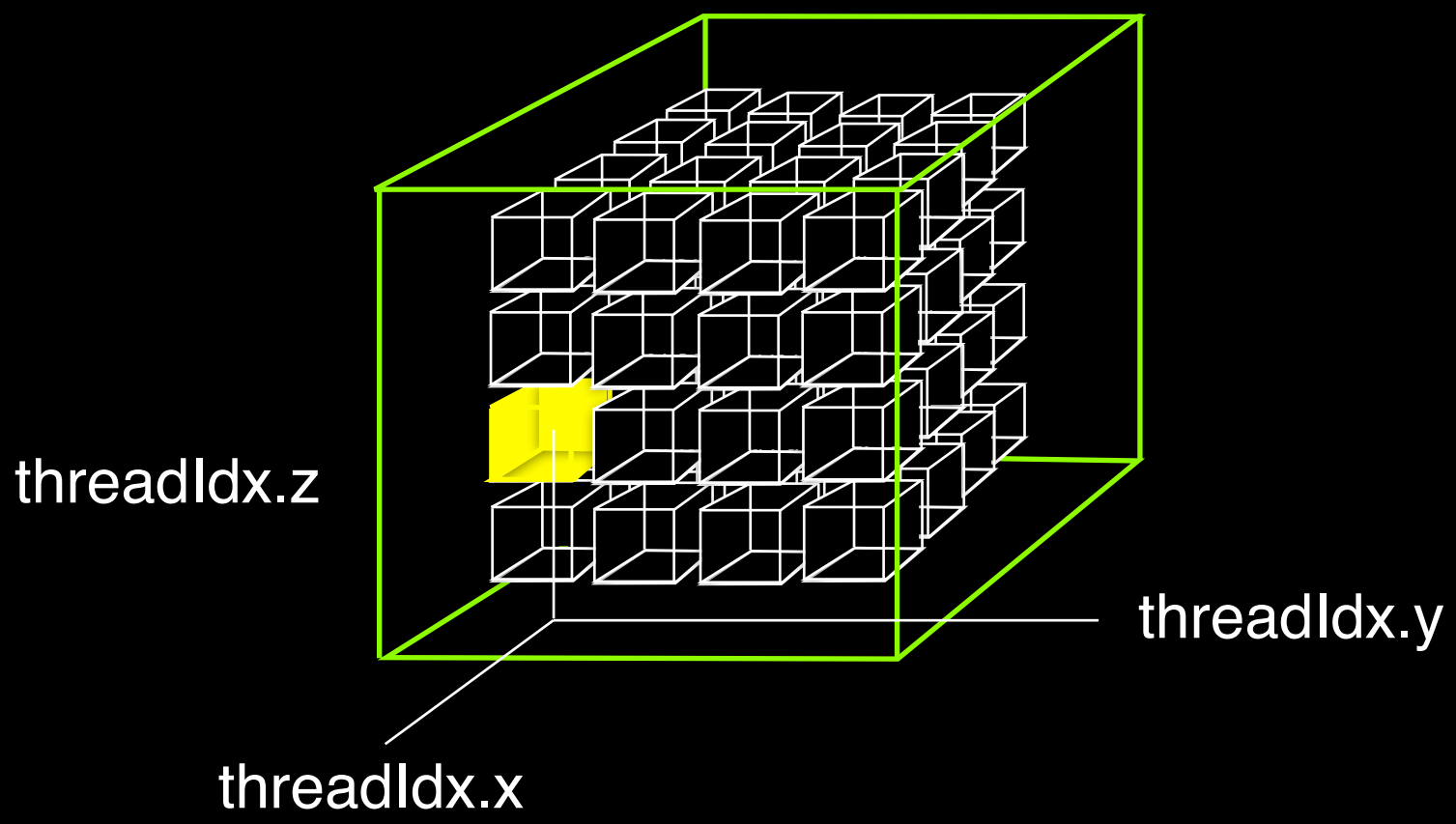


blockDim.x / blockDim.y / blockDim.z

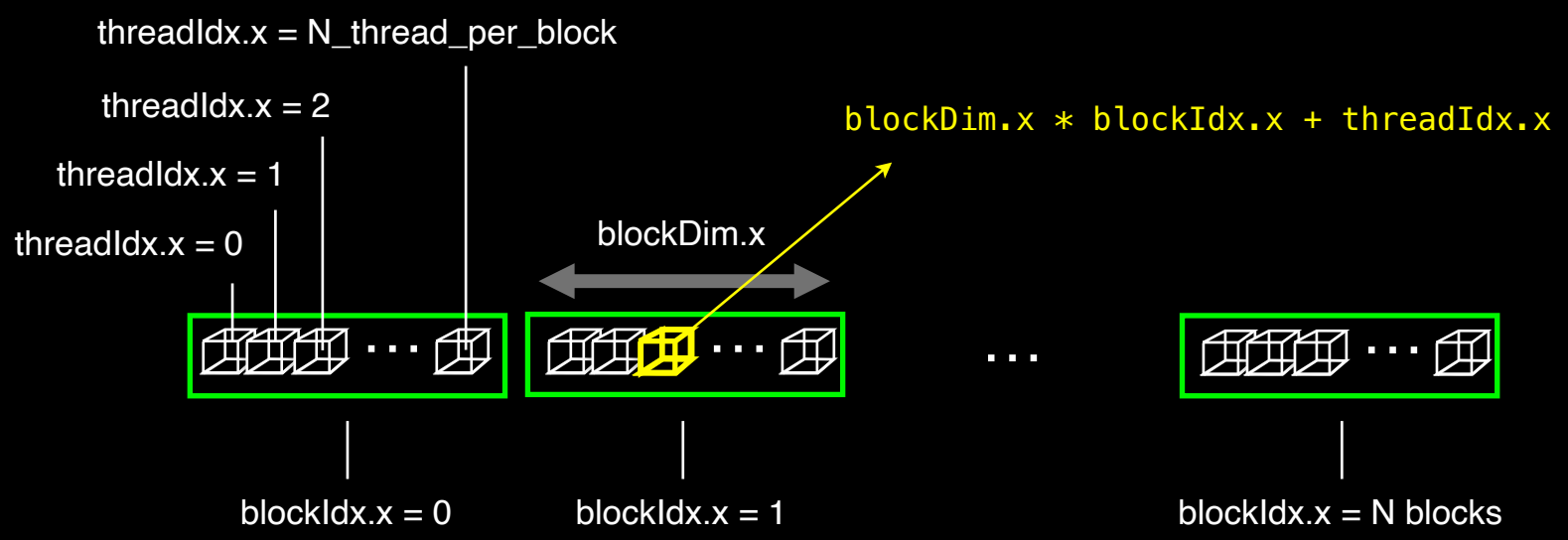


Number of threads in each dimension of each block

blockDim.x / blockDim.y / blockDim.z



In our example



It's a 1 dimensional vector addition

So we will keep it simple and use 1 dimension only
(In a later example we will use more dimension)

```
__global__ void vec_add(const float* A, const float* B, float* C, int n_data)
{

    int i_data = blockDim.x * blockIdx.x + threadIdx.x;
    if (i_data < n_data)
    {
        C[i_data] = A[i_data] + B[i_data];
    }
    return;
}
```

How multi-threading works in GPU

UF Chang Florida

If we have N = 100000 size vector

We need total of 100000 threads

If we group them by 256 threads

We need `int((100000-0.5)/256+1)` blocks

= 391 blocks

Then, we'd say our grid has 391 blocks

A [0]	B [0]	C [0]
A [1]	B [1]	C [1]
+		=
…	…	…
A [N]	B [N]	C [N]

But 391 × 256 = 100096. What's going on with extra 96?

Coming back to our example

UF Chang Florida

```
__global__ void vec_add(const float* A, const float* B, float* C, int n_data)
{

    int i_data = blockDim.x * blockIdx.x + threadIdx.x;
    if (i_data < n_data)
    {
        C[i_data] = A[i_data] + B[i_data];
    }
    return;
}
```

That's why we have a check here

`i_data` may go up to 100096 but `N_data` = 100000
Then these threads do nothing (thread divergence)

How to call the __global__ function

UF Chang Florida

```
vec_add<<<grid_size, block_size>>>(A_device, B_device, C_device, N_data);
```

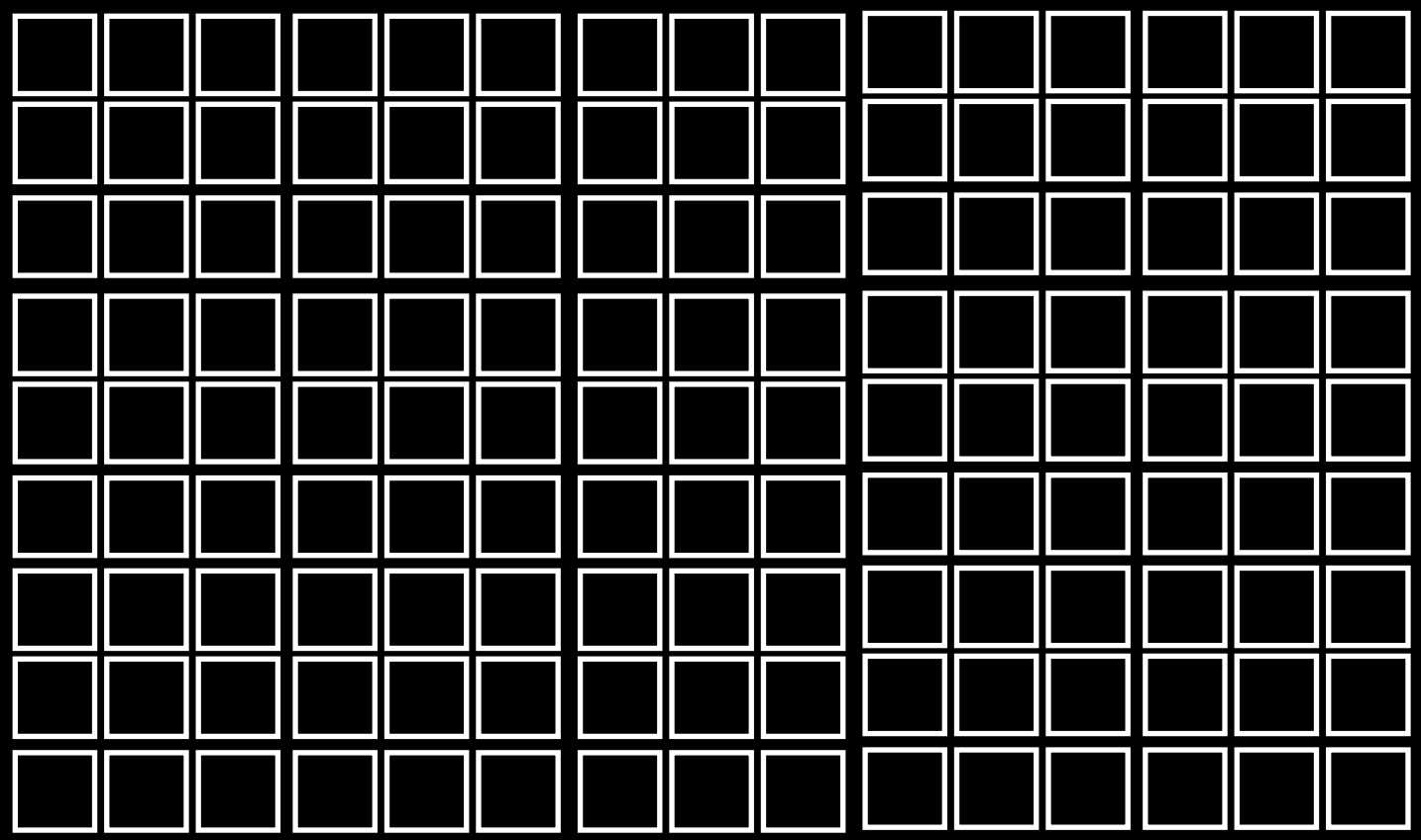
It uses a special <<<, >>> notation

First argument:
number of thread blocks
Second argument:
the size of the thread block or number of thread per block

This then launches a grid of blocks of threads

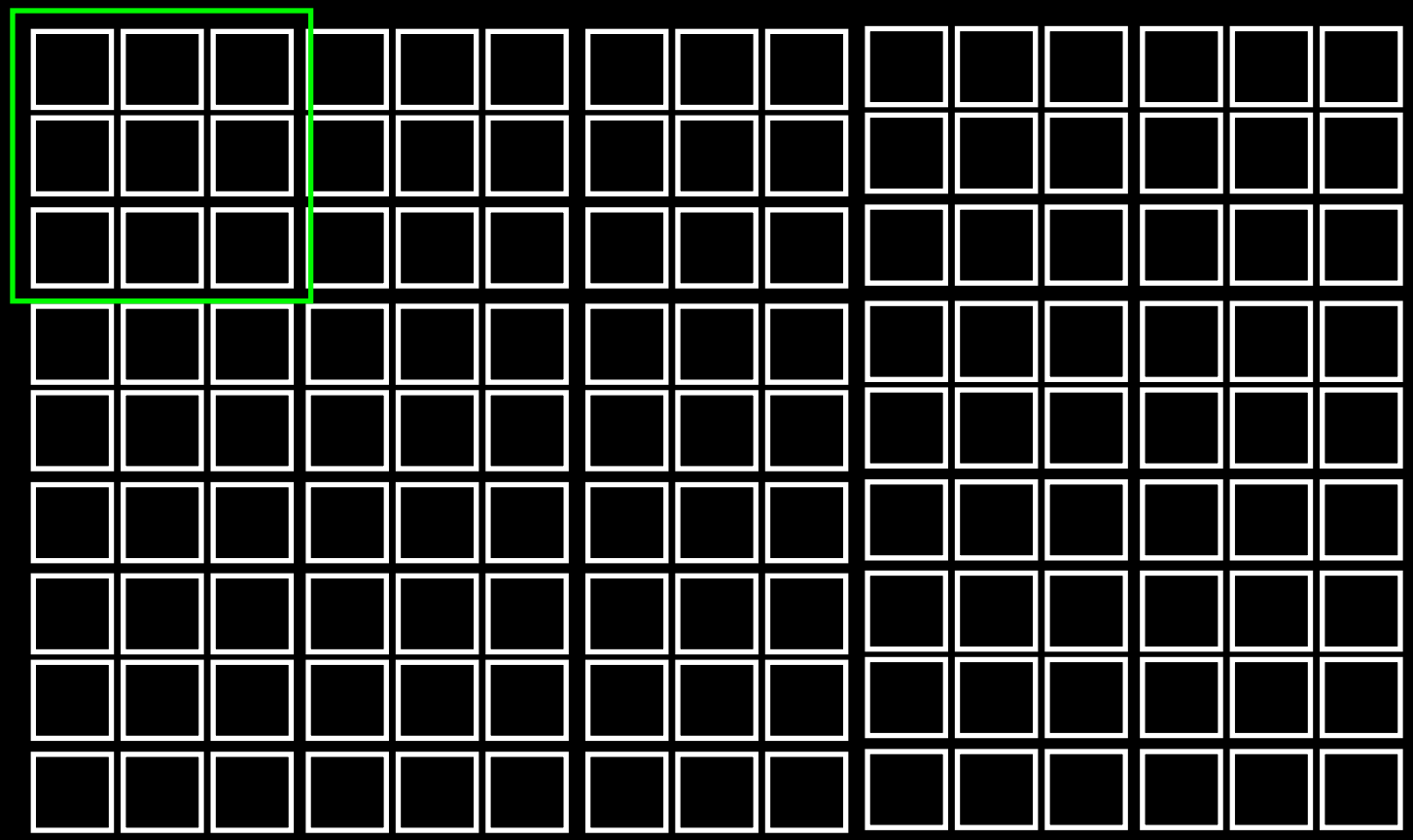
Dimension example

UF
Chang
Florida



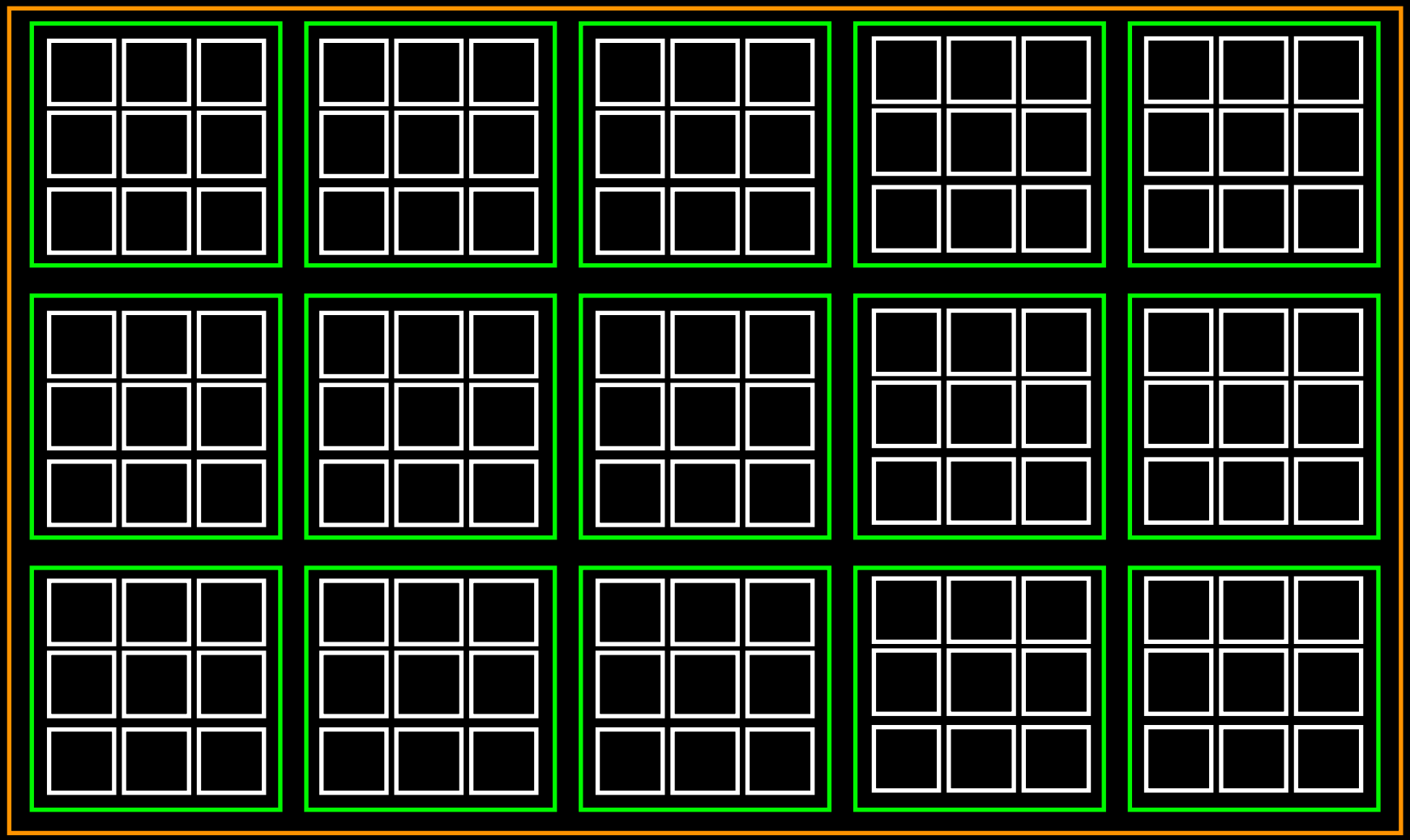
Imagine I have 9 x 15 threads to be done
(perhaps it's a matrix multiplication to produce 9 x 15 matrix)

Dimension example



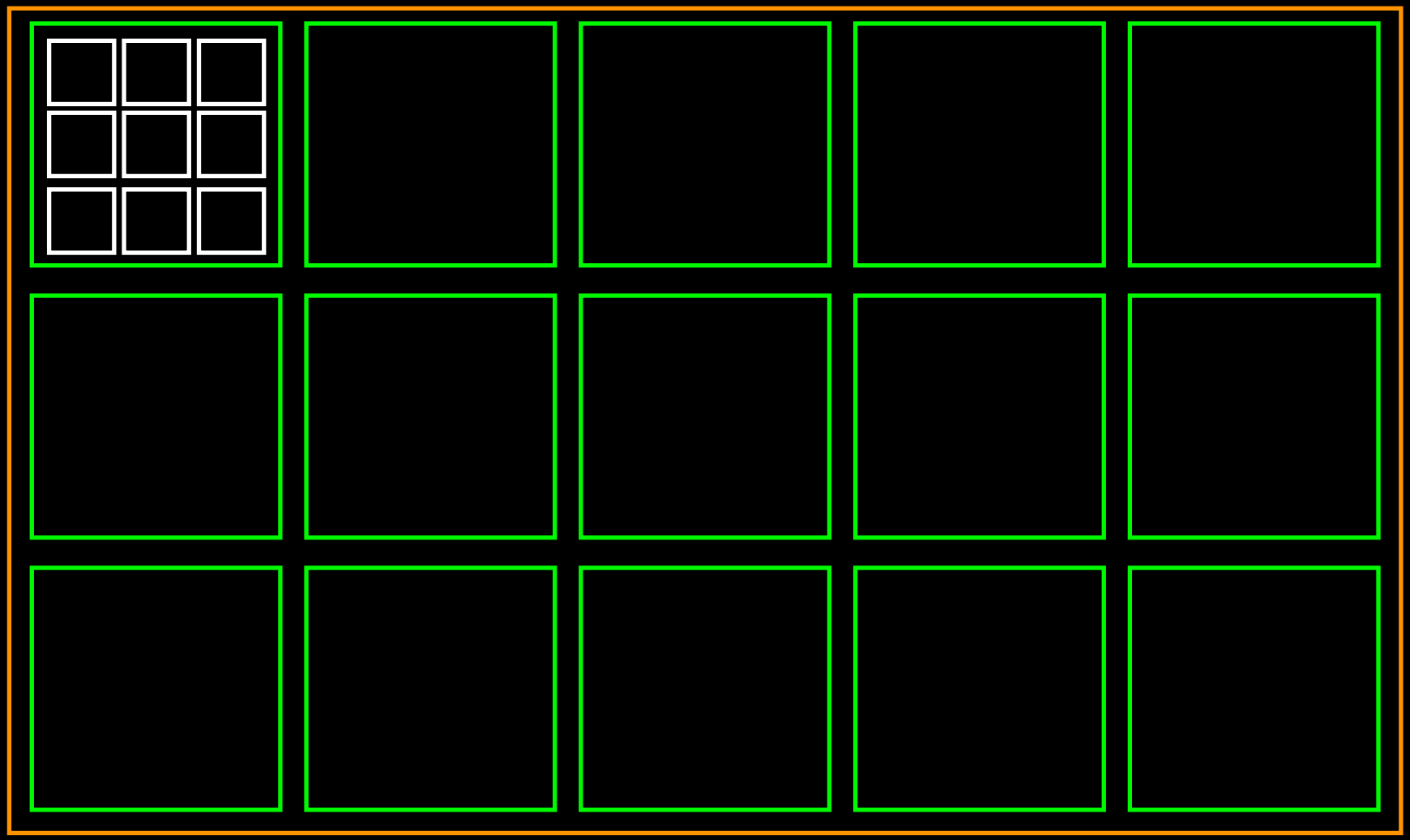
First with some domain knowledge you decide that
"OK I think having 9 thread per thread block is reasonable"

Dimension example



Then, we would have a grid of size = 15 blocks

Dimension example



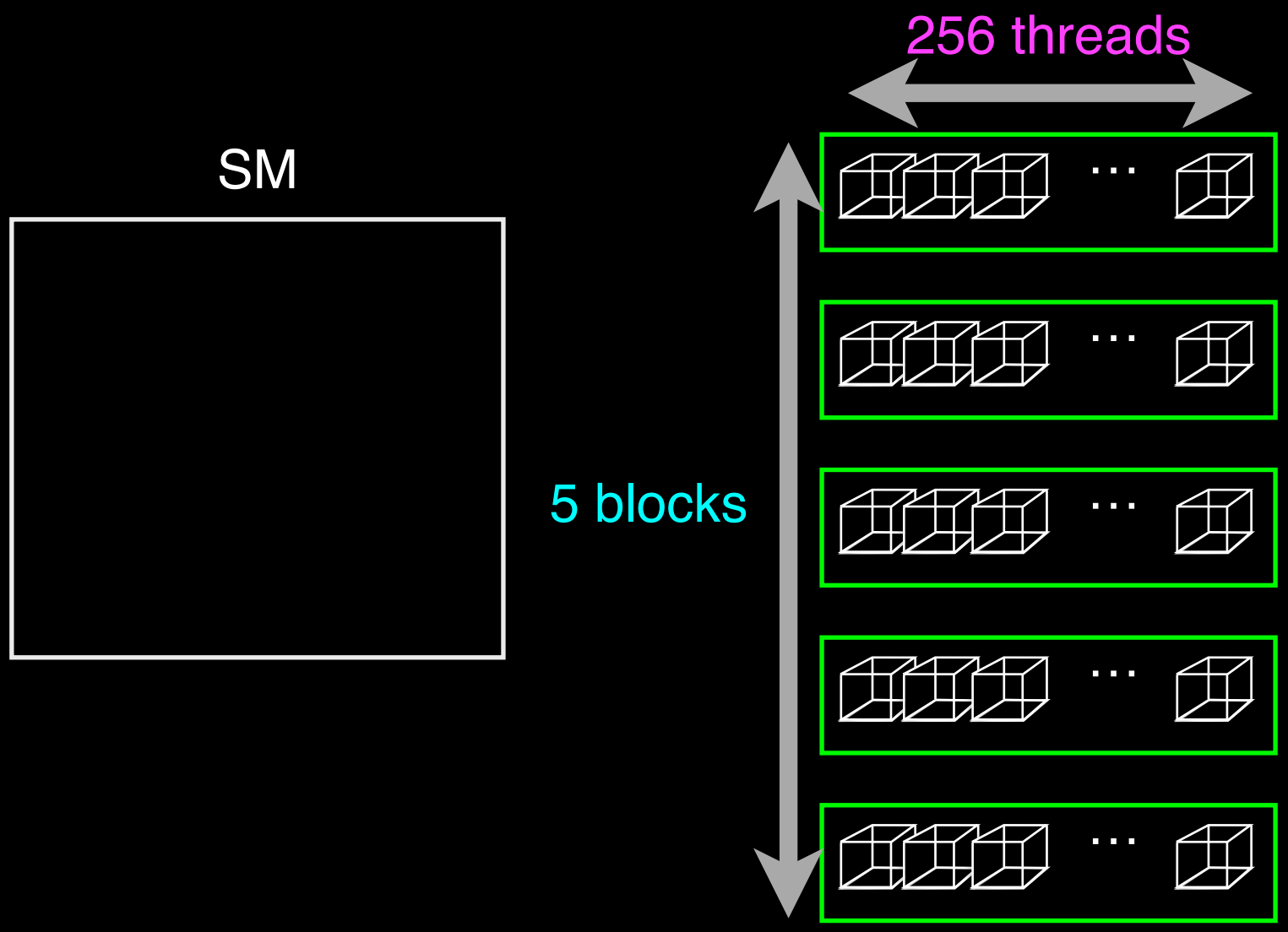
grid size = 15 blocks

block size = 9 threads

Thread blocks and SM

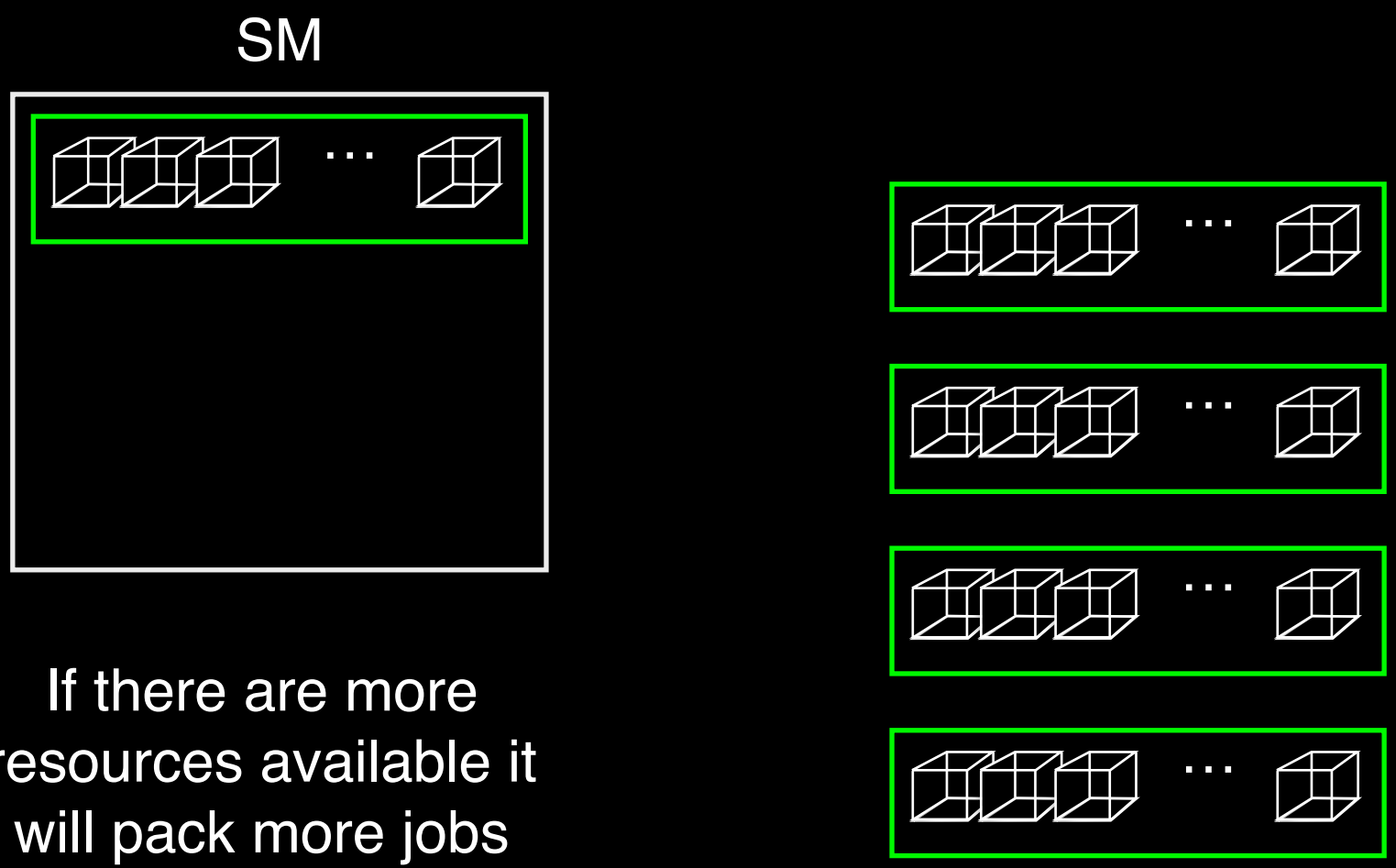
e.g.

```
vec_add<<<5, 256>>>(…);
```



Thread blocks and SM

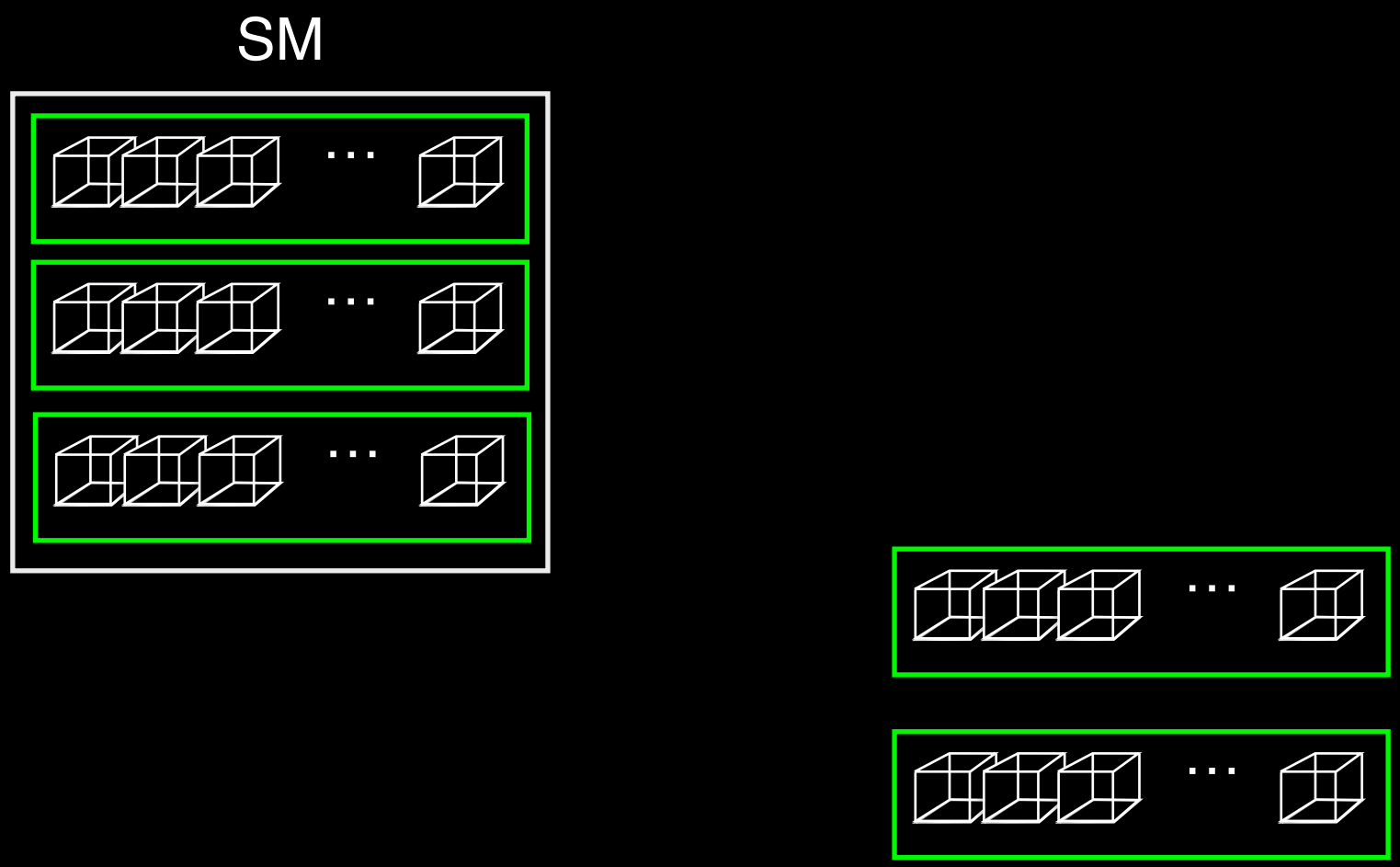
What is physically happening is that each thread block gets matched to a SM



If there are more resources available it will pack more jobs

Thread blocks and SM

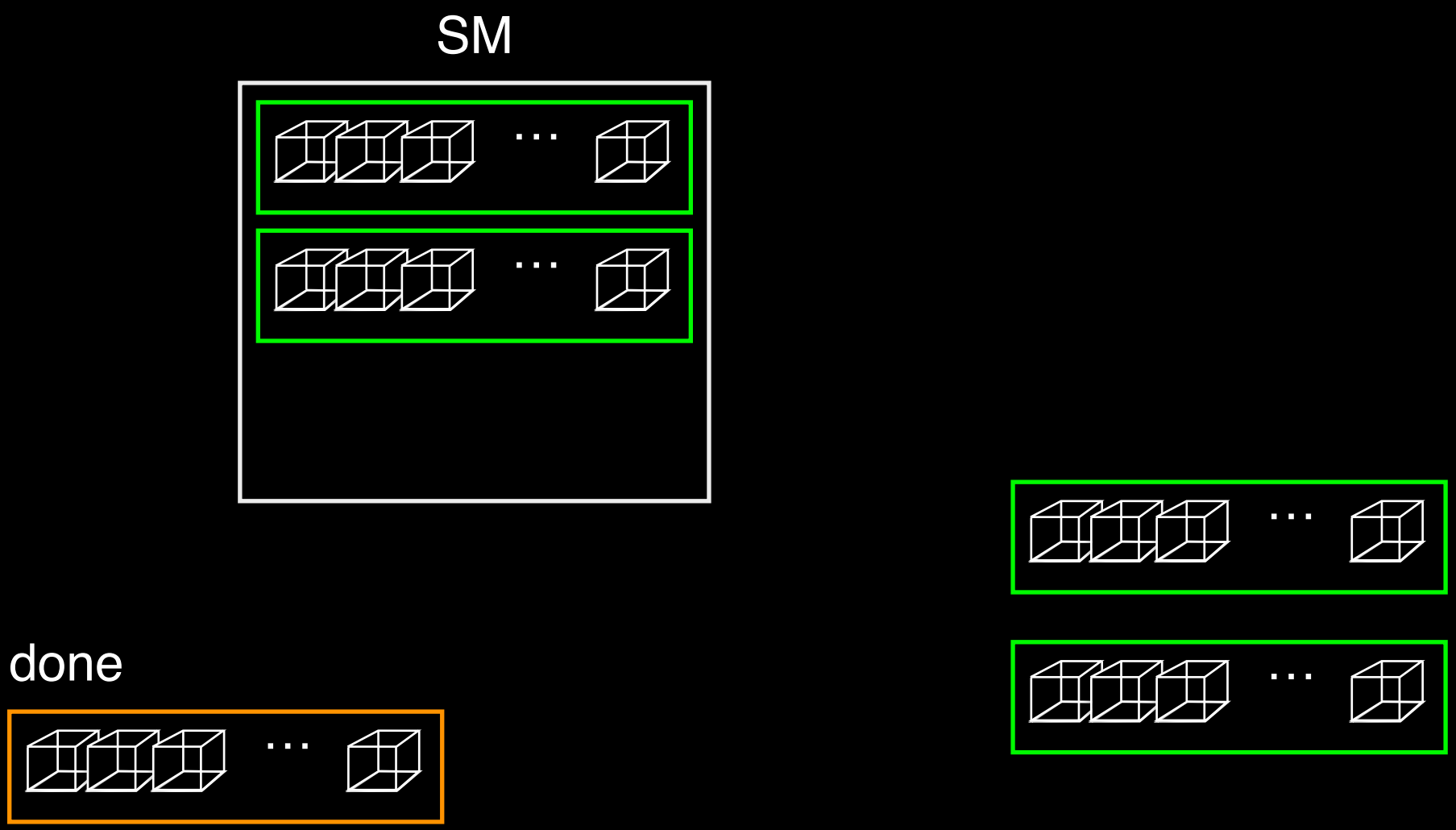
What is physically happening is that each thread block gets matched to a SM



Thread blocks and SM

UF
Chang
Florida

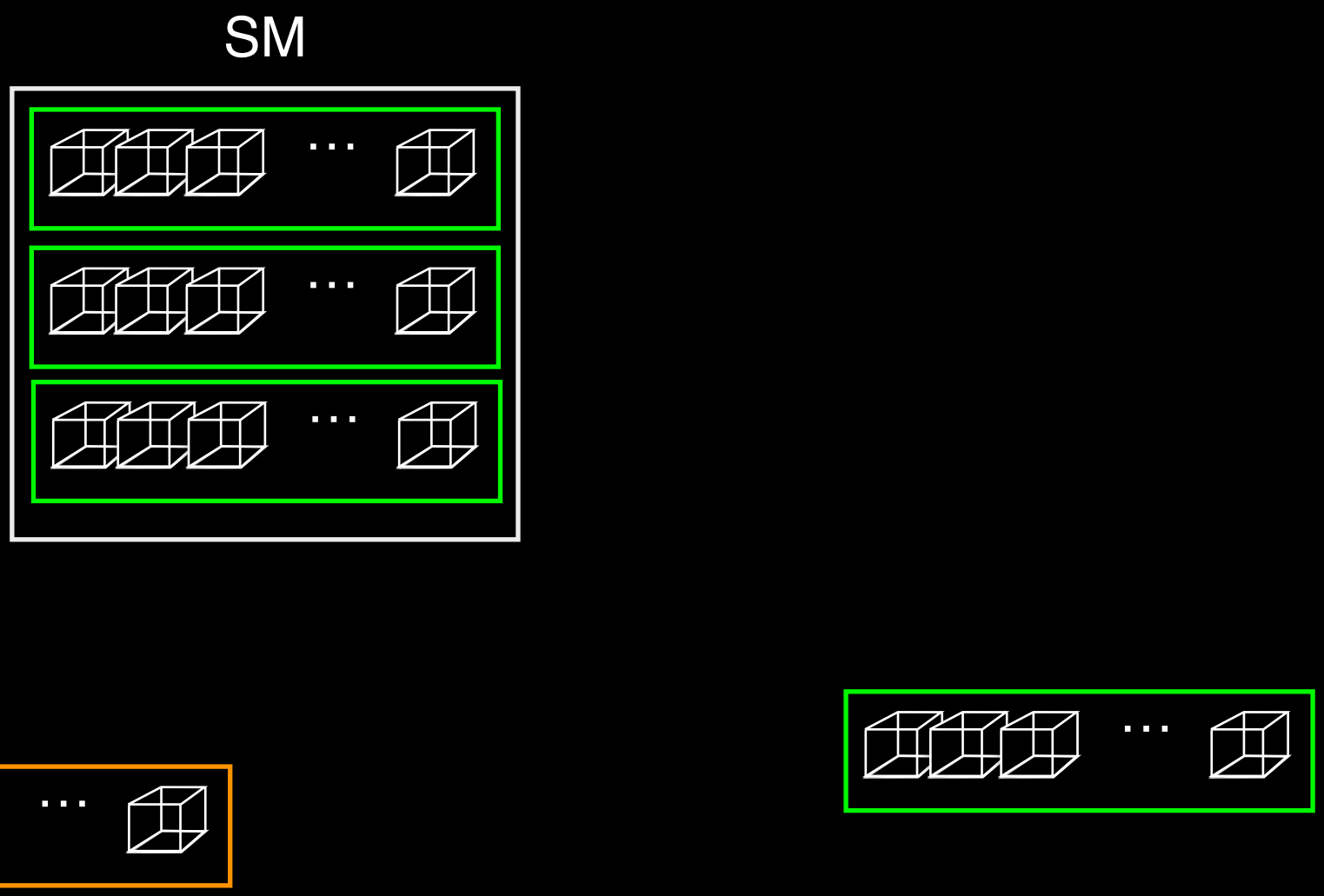
What is physically happening is that each thread block gets matched to a SM



Thread blocks and SM

UF Chang Florida

What is physically happening is that each thread block gets matched to a SM

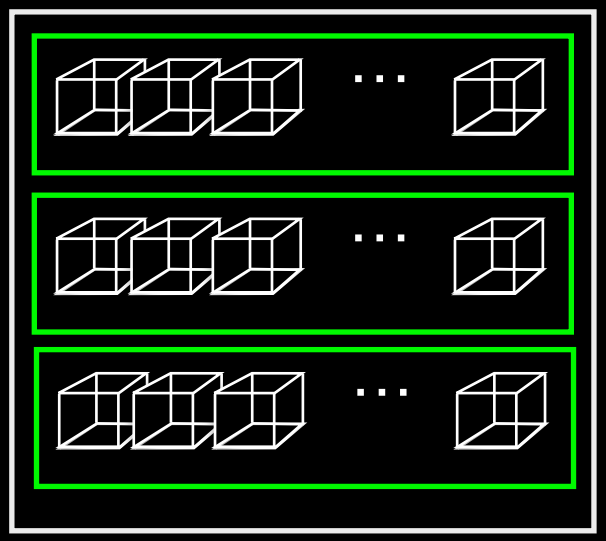


Thread blocks and SM

UF Chang Florida

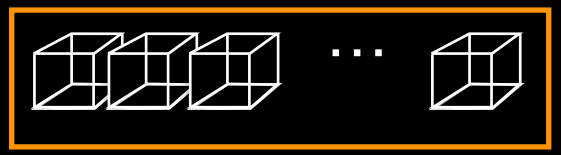
What is physically happening is that each thread block gets matched to a SM

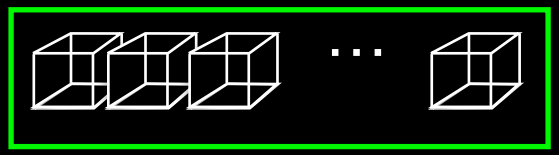
SM



In what order do the threads get executed?

done



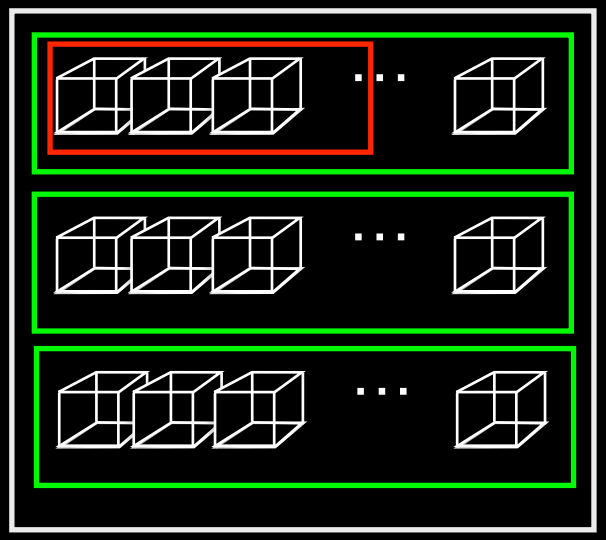


Thread blocks and SM

UF Chang Florida

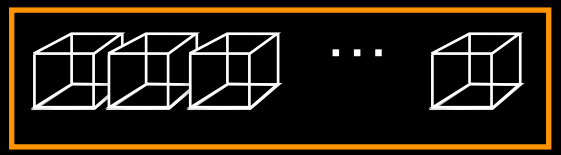
What is physically happening is that each thread block gets matched to a SM

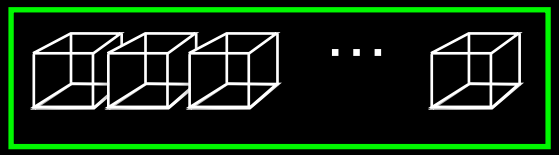
SM



A “Warp” of 32 threads are executed at the same time

done



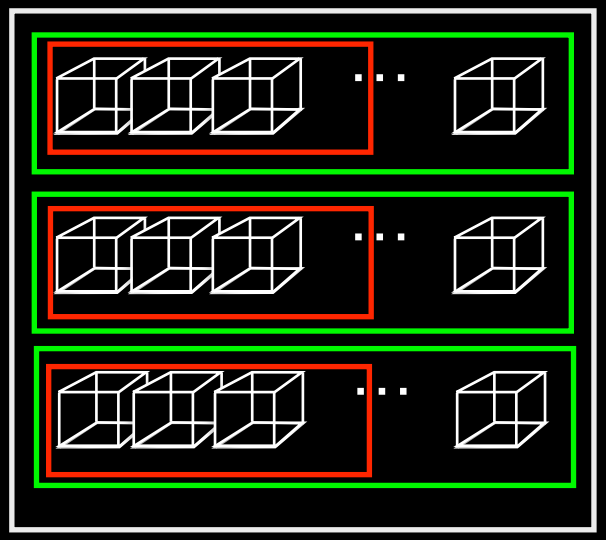


Thread blocks and SM

UF Chang Florida

What is physically happening is that each thread block gets matched to a SM

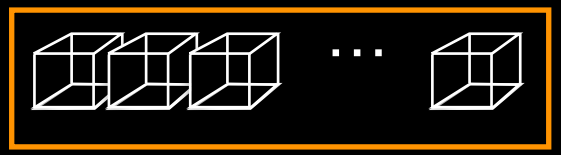
SM

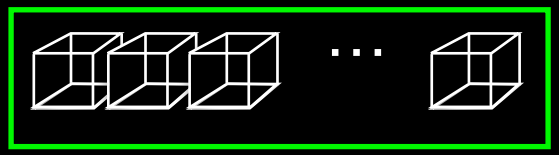


A “Warp” of 32 threads are executed at the same time

Warp schedulers will orchestrate which warps to run

done

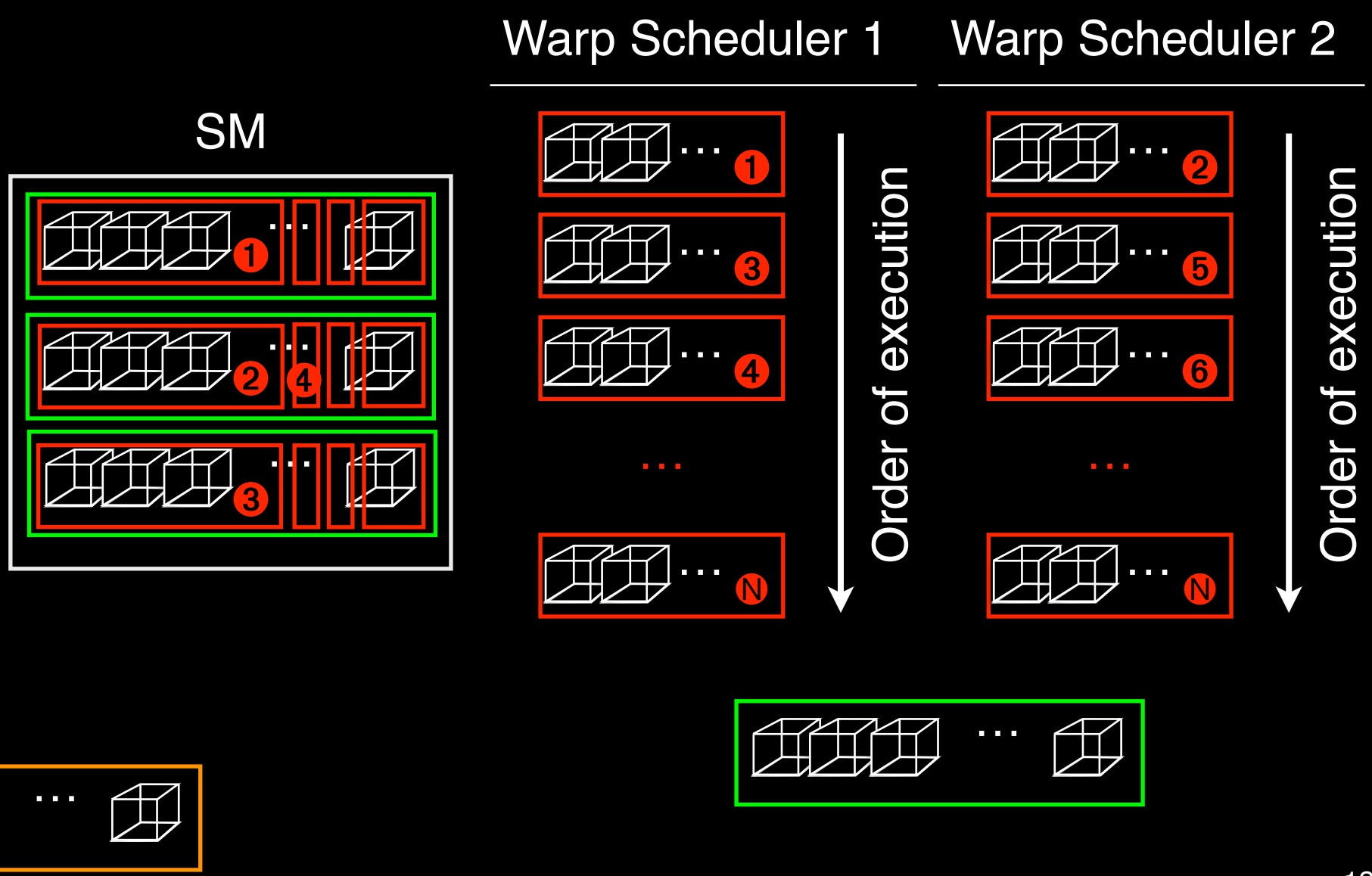




Thread blocks and SM

UF Chang Florida

Warp scheduler decides what runs in what order
(Not something we can really control)



Synchronization

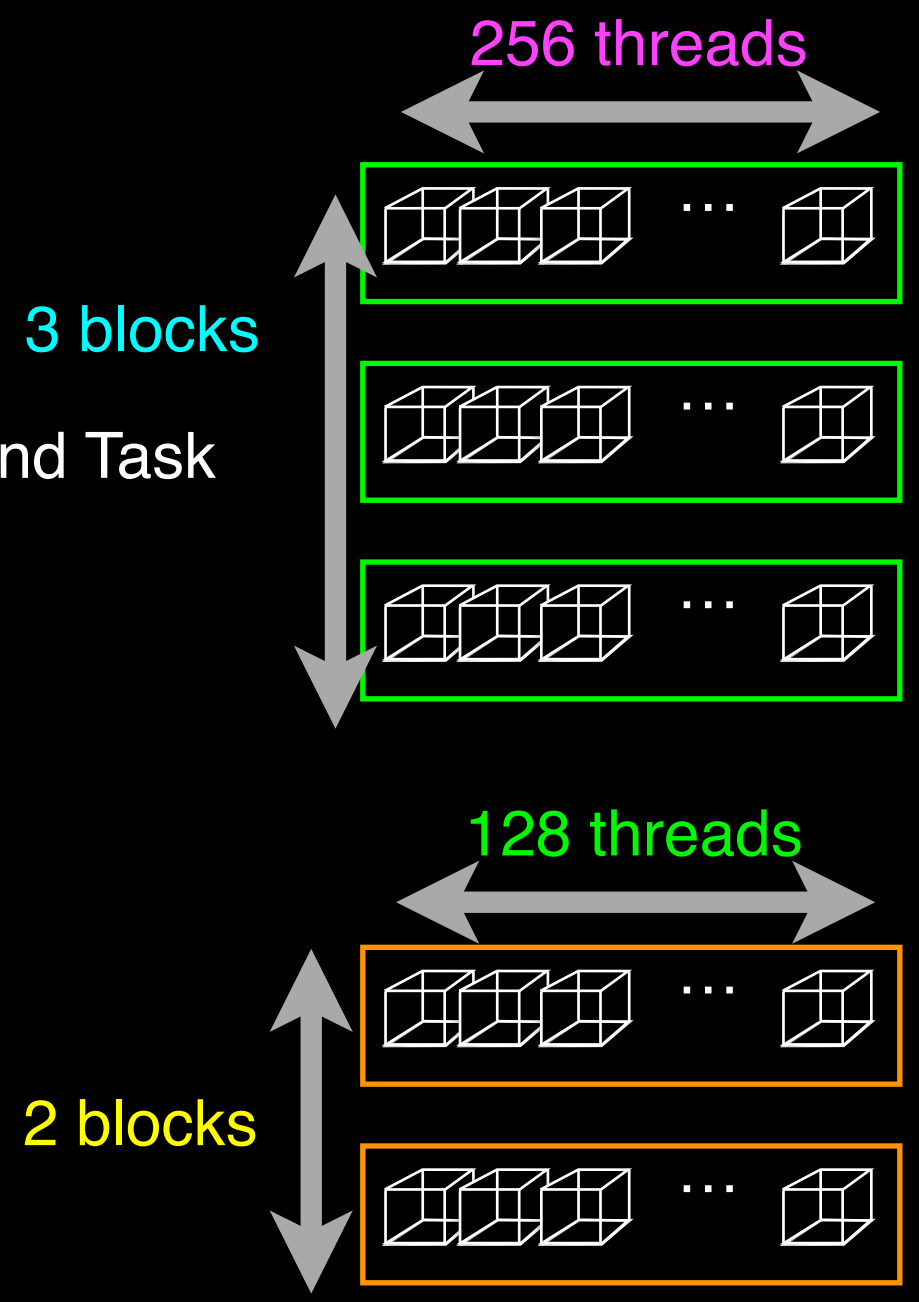
UF Chang Florida

Suppose we have two tasks

```
__global__ MyFirstTask(…)
__global__ MySecondTask2(…)
```

We launch a grid for First Task and Second Task

```
MyFirstTask<<<3, 256>>>(…)
MySecondTask<<<2, 128>>>(…)
```



Synchronization

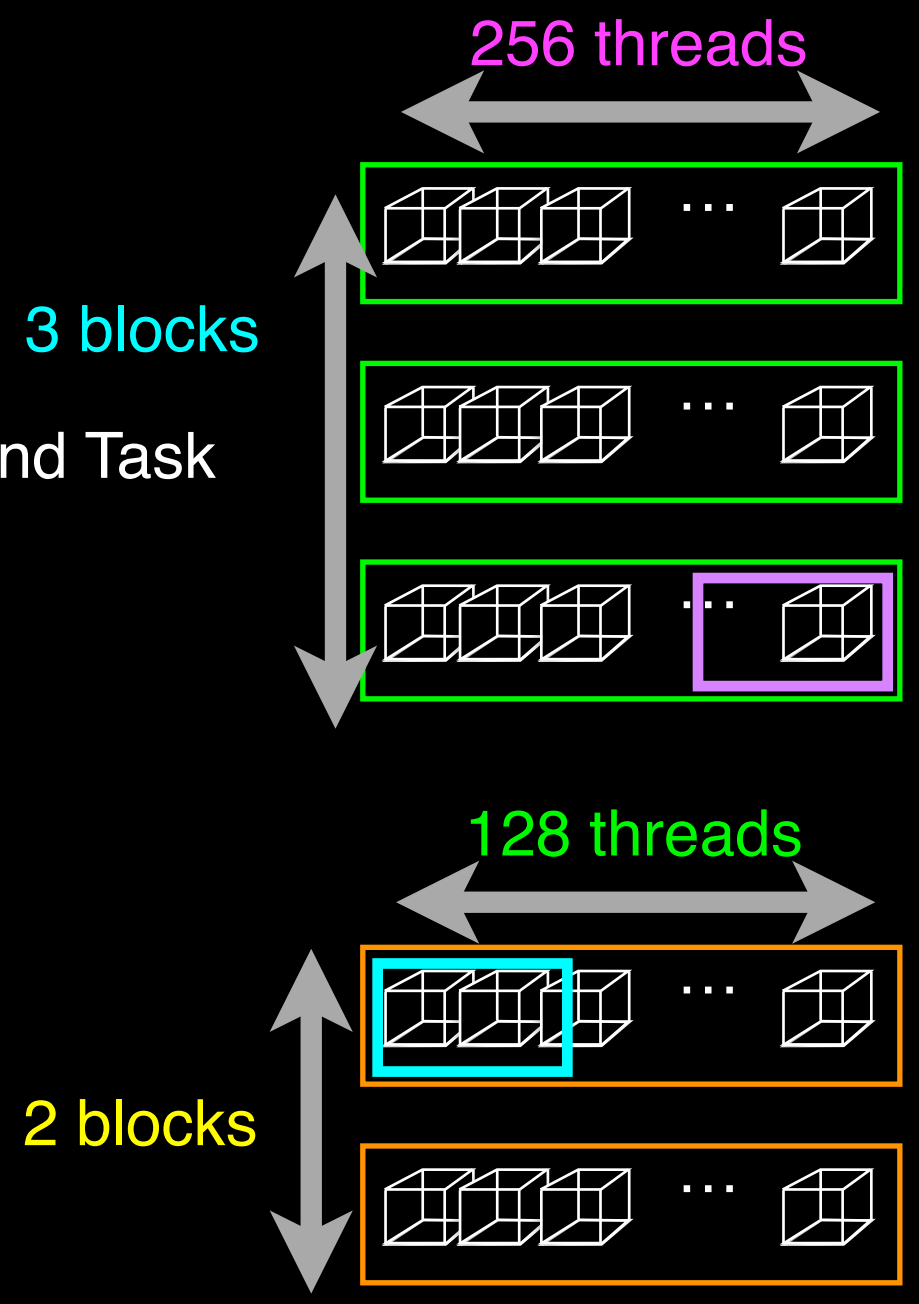
UF Chang Florida

Suppose we have two tasks

```
__global__ MyFirstTask(…)
__global__ MySecondTask2(…)
```

We launch a grid for First Task and Second Task

```
MyFirstTask<<<3, 256>>>(…)
MySecondTask<<<2, 128>>>(…)
```



What if a MySecondTask's Warp starts before the last warp in MyFirstTask finishes…?

Synchronization

UF Chang Florida

Suppose we have two tasks

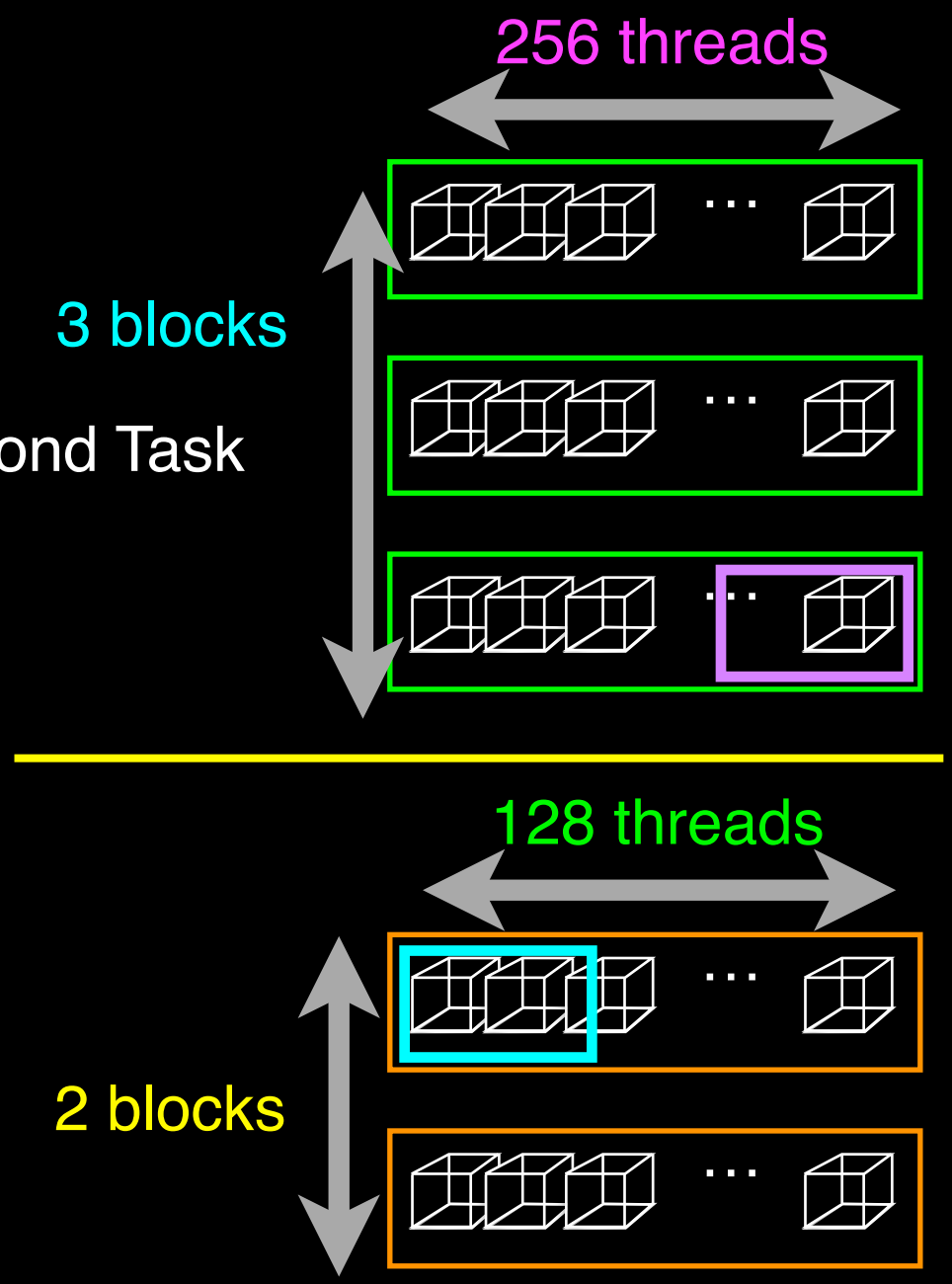
```
__global__ MyFirstTask(…)
__global__ MySecondTask2(…)
```

We launch a grid for First Task and Second Task

```
MyFirstTask<<<3, 256>>>(…)
MySecondTask<<<2, 128>>>(…)
```

We need a way to "divide" the line ("synchronize")

What if a MySecondTask's Warp starts before the last warp in MyFirstTask finishes…?



Synchronization

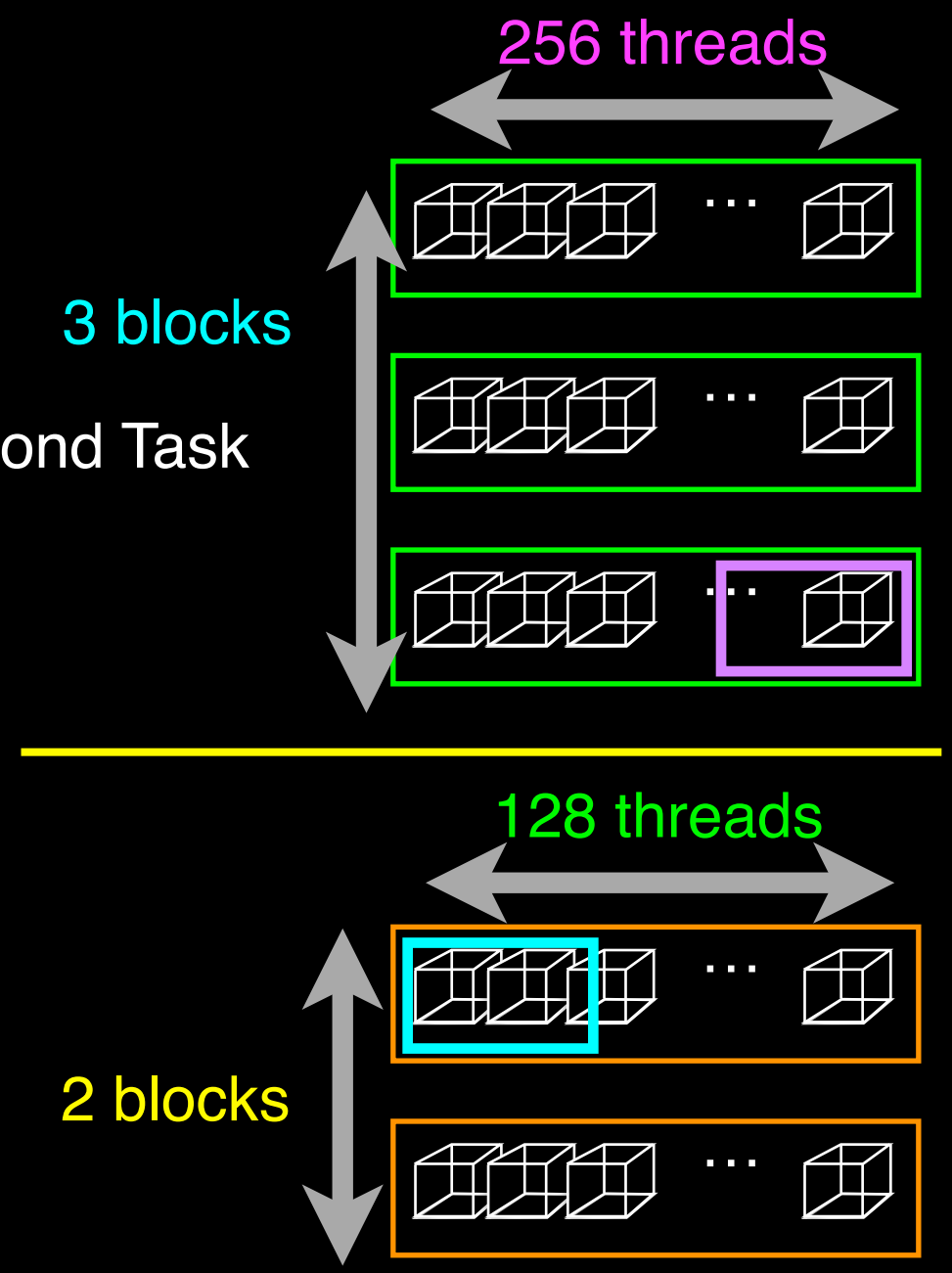
UF Chang Florida

Suppose we have two tasks

```
__global__ MyFirstTask(…)
__global__ MySecondTask2(…)
```

We launch a grid for First Task and Second Task

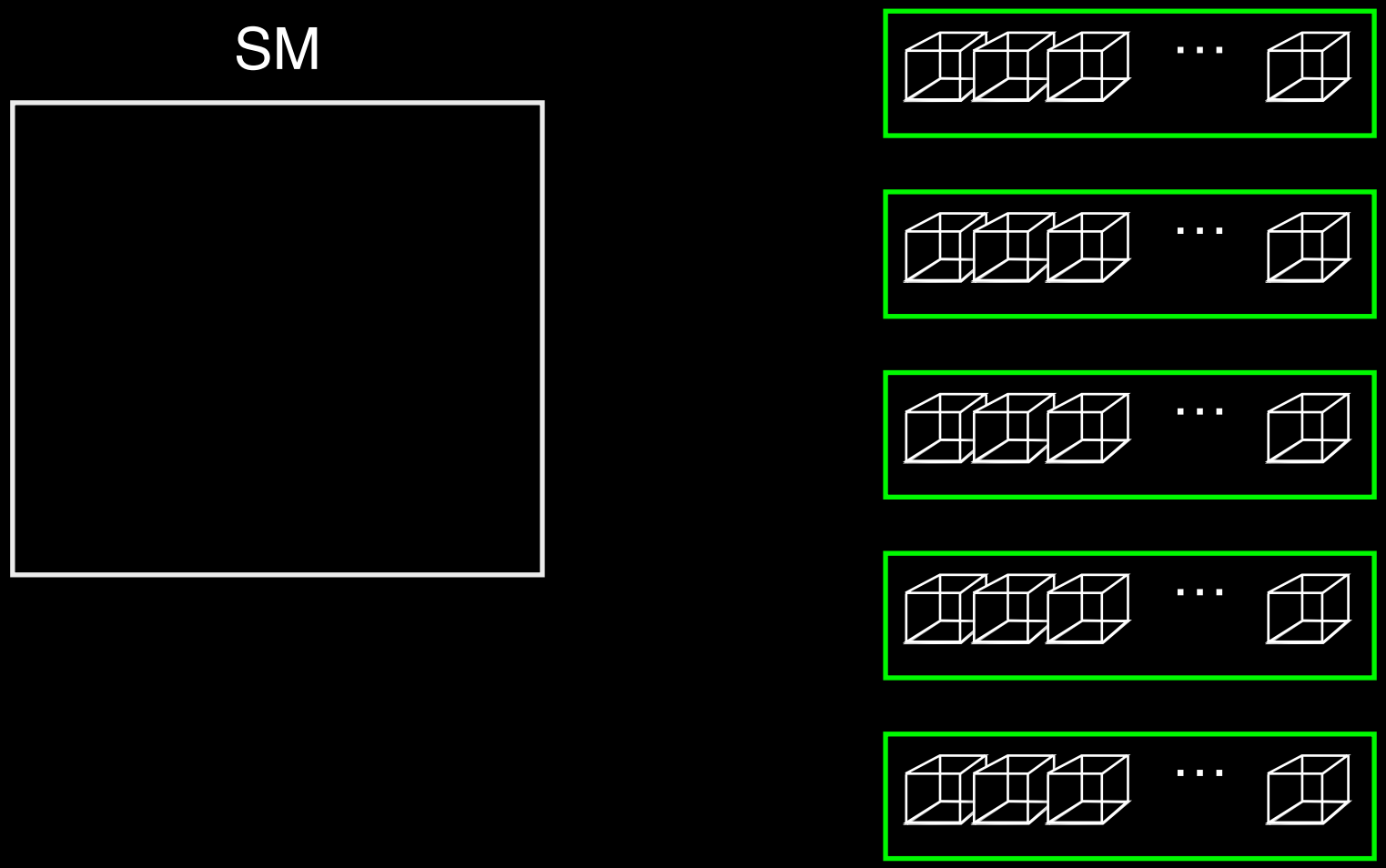
```
MyFirstTask<<<3, 256>>>(…)

cudaDeviceSynchronize();

MySecondTask<<<2, 128>>>(…)
```



Thread blocks and SM

UF
Chang
Florida

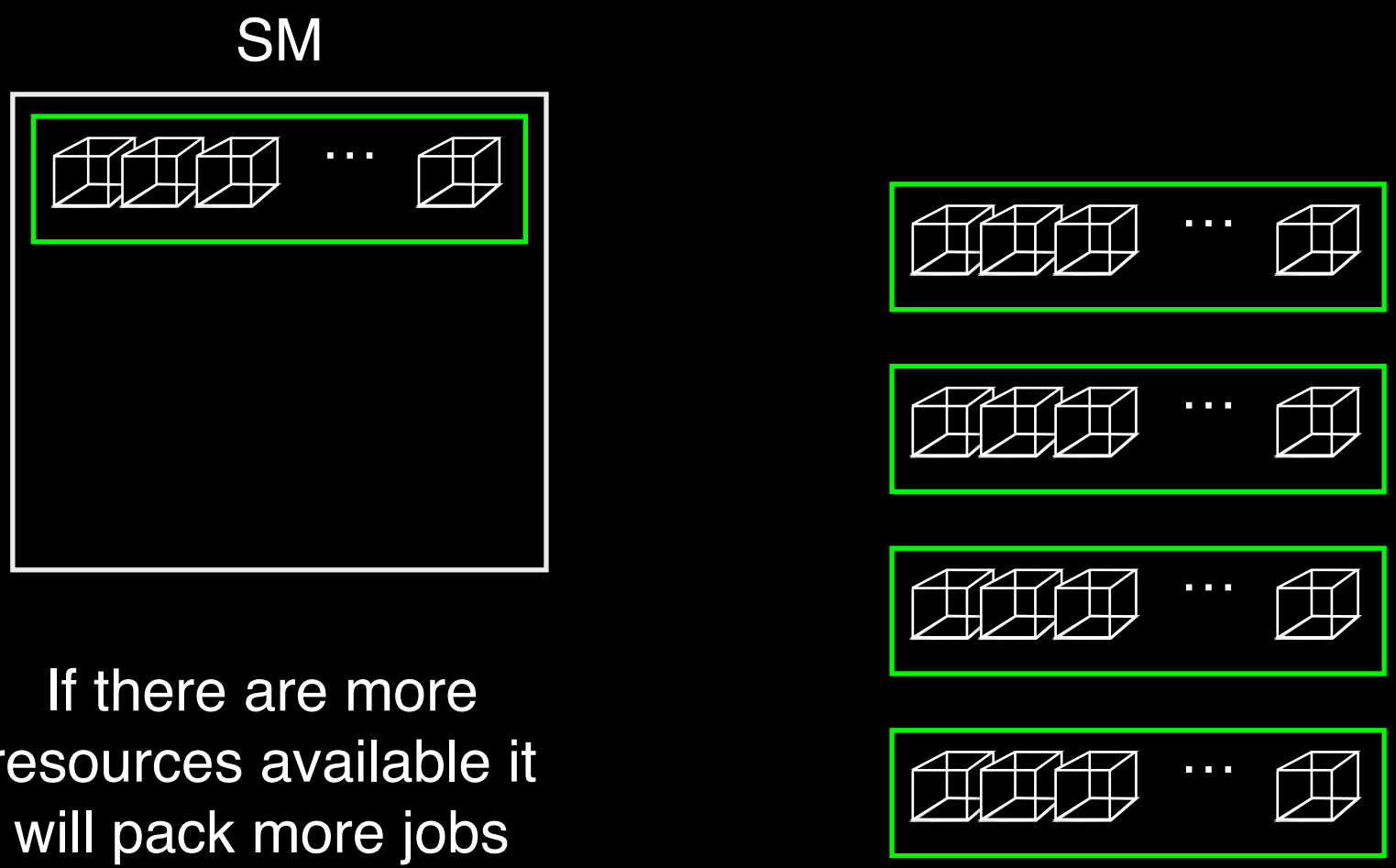
What is physically happening is that each thread block gets matched to a SM



Thread blocks and SM

UF
Chang
Florida

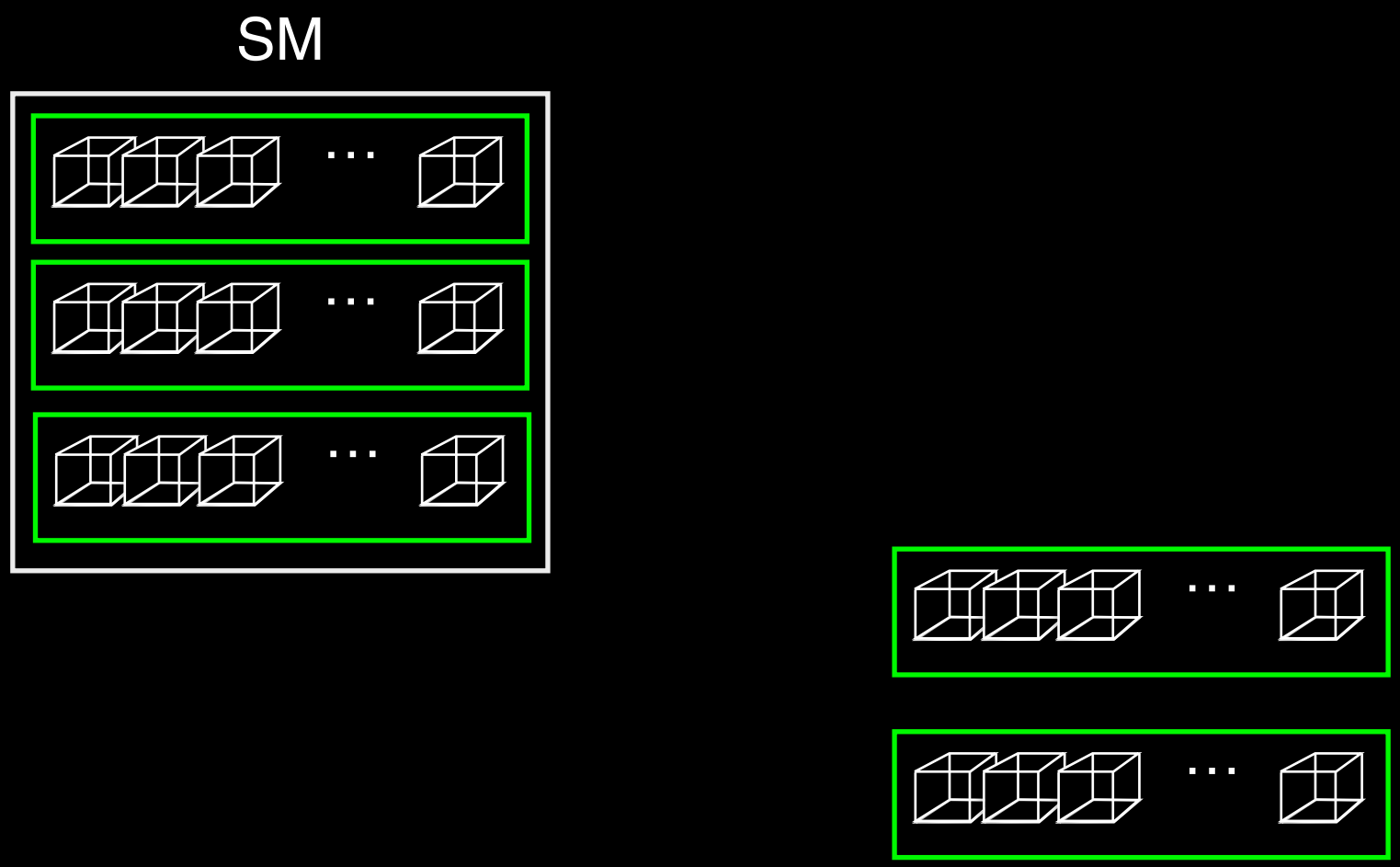
What is physically happening is that each thread block gets matched to a SM



If there are more resources available it will pack more jobs

Thread blocks and SM

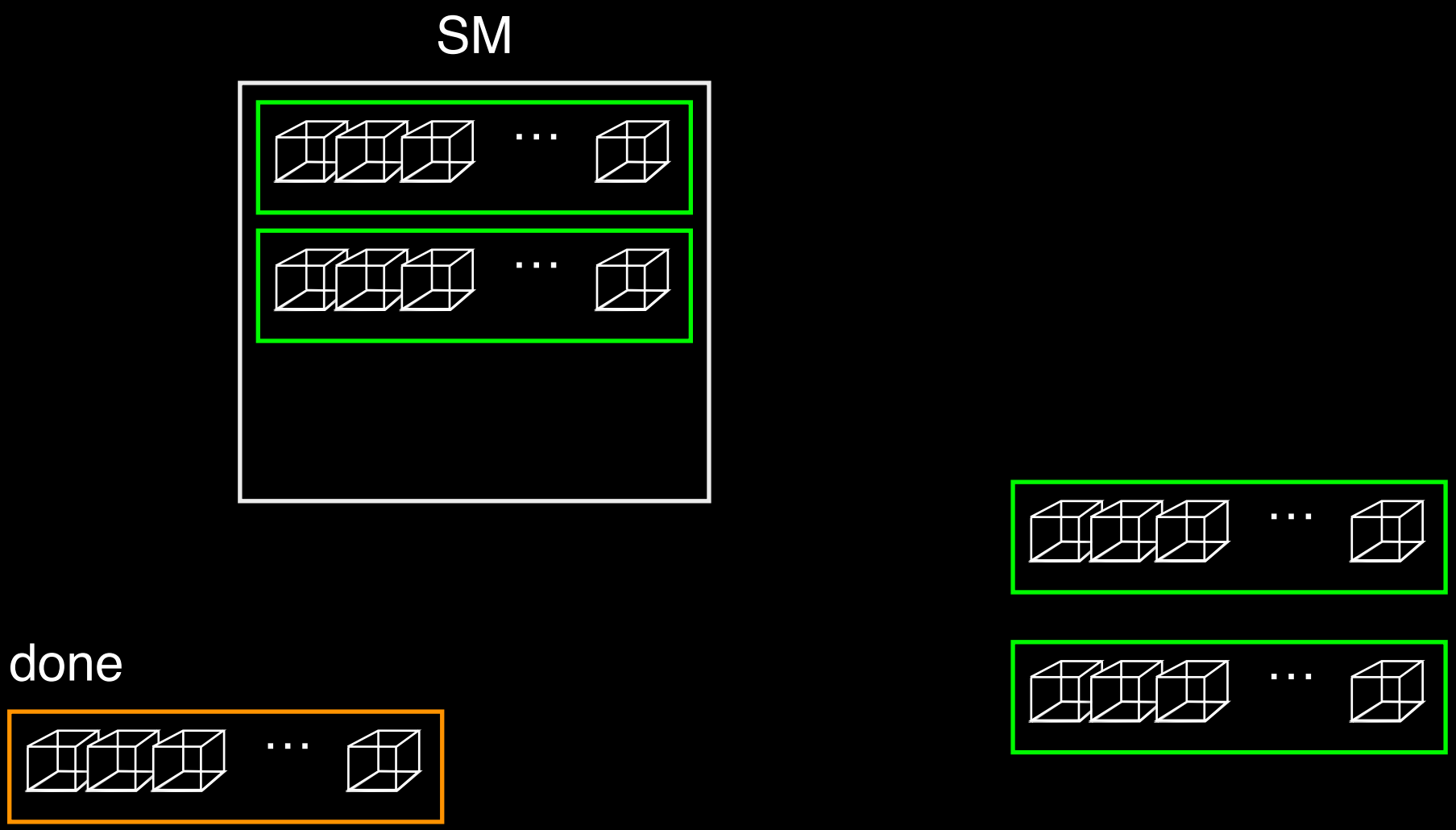
What is physically happening is that each thread block gets matched to a SM



Thread blocks and SM

UF
Chang
Florida

What is physically happening is that each thread block gets matched to a SM

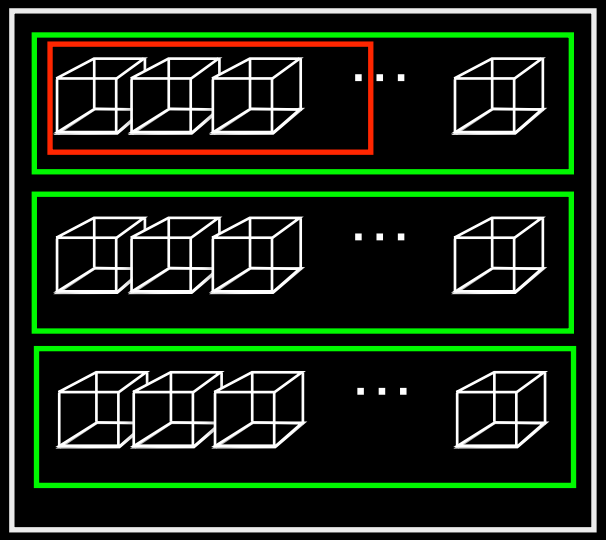


Thread blocks and SM

UF Chang Florida

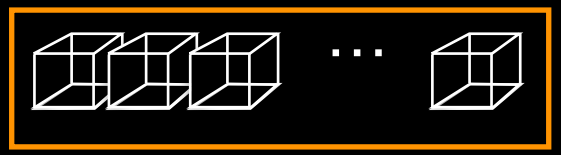
What is physically happening is that each thread block gets matched to a SM

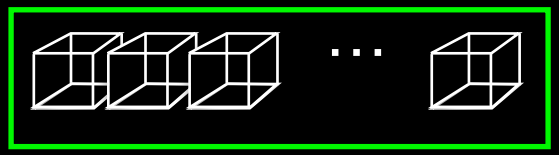
SM



A “Warp” of 32 threads are executed at the same time

done



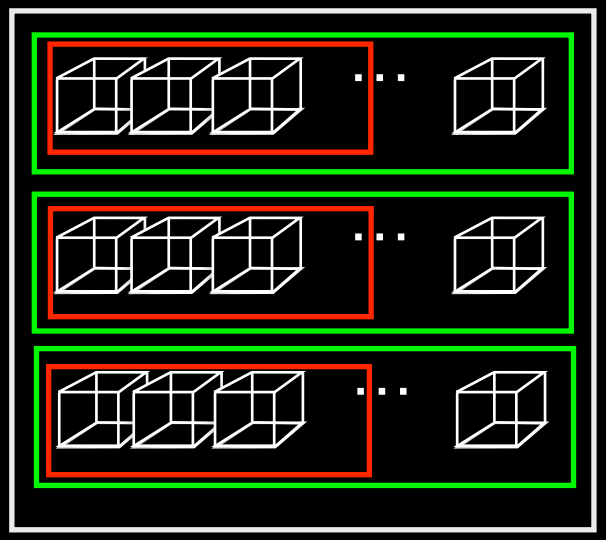


Thread blocks and SM

UF Chang Florida

What is physically happening is that each thread block gets matched to a SM

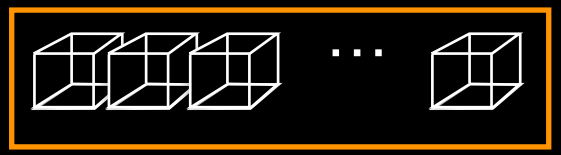
SM

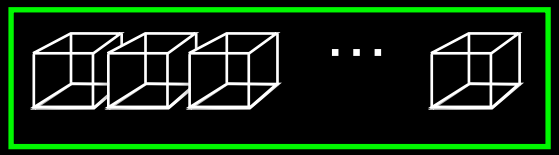


A “Warp” of 32 threads are executed at the same time

Warp schedulers will orchestrate which warps to run

done

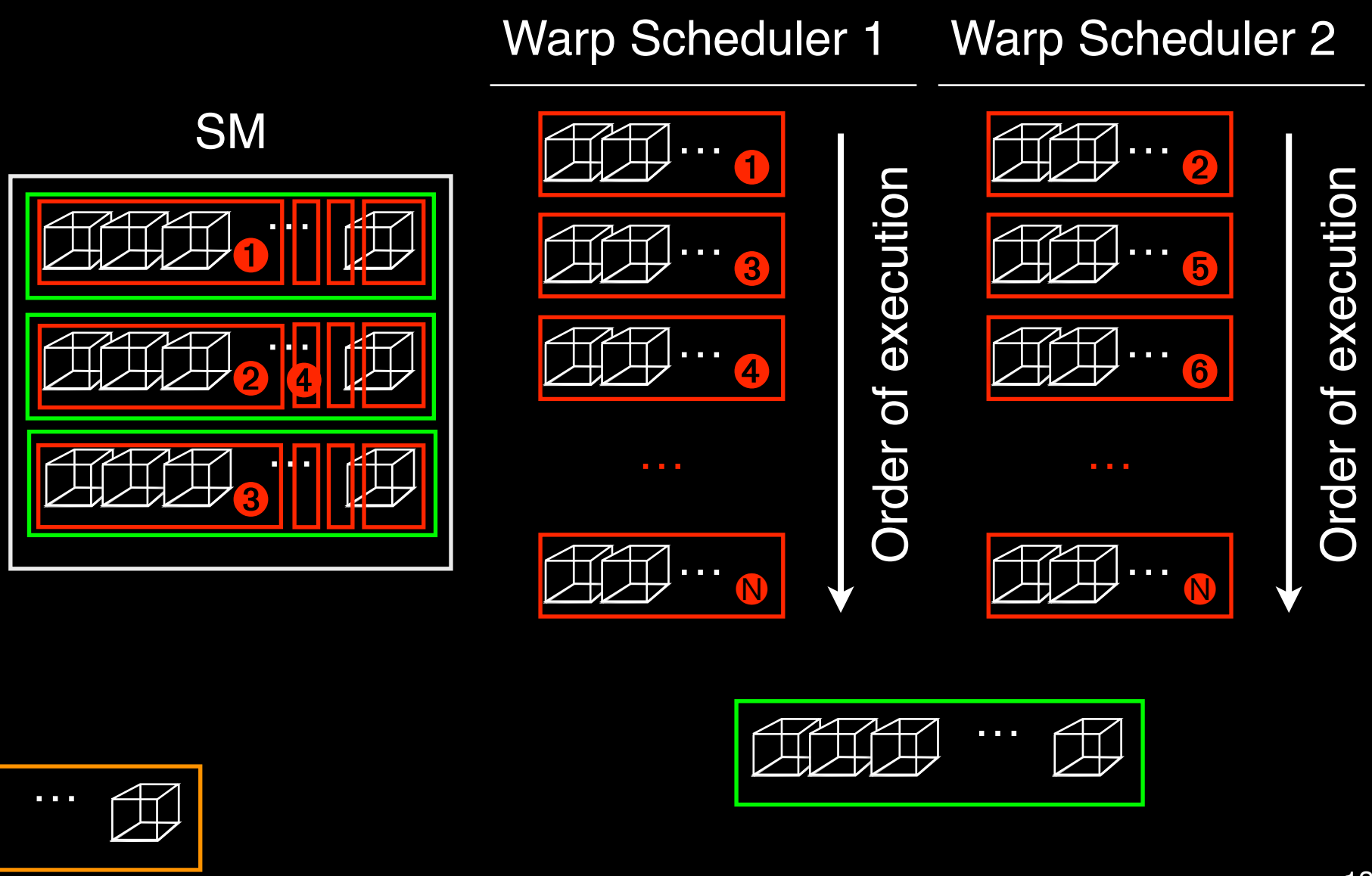




Thread blocks and SM

UF Chang Florida

Warp scheduler decides what runs in what order
(Not something we can really control)



Synchronization

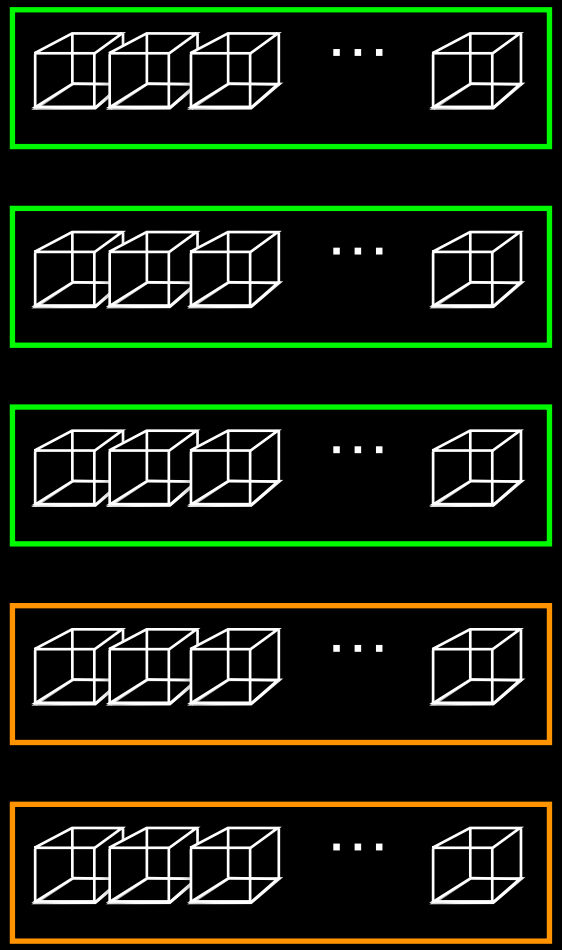
UF Chang Florida

Suppose we have two tasks

```
__global__ MyFirstTask(…)
__global__ MySecondTask2(…)
```

We launch a grid for First Task and Second Task

```
__global__ MyFirstTask(…)
```



Finish coding up

tests - JupyterLab | raw.githubusercontent.com/sgnoohc

https://jupyterhub.ssl-hep.org/user/p.chang@uf

File Edit View Run Kernel Tabs Settings Help

jovyan@jupyter-p-2echang

```
#include <iostream>
#include <chrono>
using namespace std::chrono;

__global__ void vec_add(const float* A, const float* B, float* C, int n_data, int n_ops)
{

    int i_data = blockDim.x * blockIdx.x + threadIdx.x;
    if (i_data < n_data)
    {
        for (int i = 0; i < n_ops; ++i)
        {
            C[i_data] = A[i_data] + B[i_data];
        }
    }
    return;
}

int main()
```

4,0-1 Top

DIFFERENT

Simple 1 0 Mem: 166.7... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2... 0

tests - JupyterLab | raw.githubusercontent.com/sgnooh

https://jupyterhub.ssl-hep.org/user/p.chang@uf

File Edit View Run Kernel Tabs Settings Help

jovyan@jupyter-p-2echang

```
    float* B_device;
    float* C_device;

    cudaMalloc((void**) &A_device, n_data * sizeof(float));
    cudaMalloc((void**) &B_device, n_data * sizeof(float));
    cudaMalloc((void**) &C_device, n_data * sizeof(float));

    cudaMemcpy(A_device, A_host, n_data * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(B_device, B_host, n_data * sizeof(float), cudaMemcpyHostToDevice);

    int block_size = 256;
    int grid_size = int(n_data - 0.5) / block_size + 1;

    vec_add<<<grid_size, block_size>>>(A_device, B_device, C_device, n_data, n_ops);

    cudaDeviceSynchronize();

    return 0;
}
"vadd_host.cu" 82L, 2281B
                                                                82,1          Bot
```

Simple 1 0 Mem: 166.5... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2... 0

More refined version is here:

https://raw.githubusercontent.com/sgnoohc/hsf-india-examples/main/vadd.cu

Result

UF Chang Florida

jovyan@jupyter-p--researc

```
(myenv) jovyan@jupyter-p--research-2dsoftwa-2---dindia-2dmay2024-2df6chacf8:~/hsf-india-examples$ ./vadd
##################################
#                                #
#    Vector Addition Program     #
#            (GPU)               #
#                                #
##################################
 --- Input data ---
 n_data = 10000000
 n_ops  = 1000
 --- GPU Kernel Launch Config ---
 grid_size: 39063
 block_size: 256

 --- Sanity Check ---
 Printing last 10 result
 i: 9999990 C_host[i]: 2e+07
 i: 9999991 C_host[i]: 0
 i: 9999992 C_host[i]: 2e+07
 i: 9999993 C_host[i]: 0
 i: 9999994 C_host[i]: 2e+07
 i: 9999995 C_host[i]: 0
 i: 9999996 C_host[i]: 2e+07
 i: 9999997 C_host[i]: 0
 i: 9999998 C_host[i]: 2e+07
 i: 9999999 C_host[i]: 0

 --- Timing information ---
 time inititalizing       : 141.945 ms
 time allocation          : 129.967 ms
 time sending to GPU      : 6.417 ms
 time executing on GPU    : 104.64 ms
 time retrieving from GPU : 5.72 ms
 -------------------------:
 time total               : 388.691 ms
```

We are adding a vector of size 10M

We are performing addition 1000 times (but we take final result of adding once)

it is an unrealistic situation…

Time it took to create 10M length vectors

Time it took to allocate memory on GPU

Time it took to send data to GPU

Time it took to perform addition 1000 times

Time it took to retrieve data from GPU

Comparison

UF Chang Florida

Initializati	Allocation	Send	Execution	Retrieval	Total
149	-	-	15,461	-	15,610

150x

40x

Initializati	Allocation	Send	Execution	Retrieval	Total
141.9	130.0	6.4	104.6	5.7	388.7

Overhead added

Comparison

UF Chang Florida

If we had run with addition repeated only 10 time

Initializati	Allocation	Send	Execution	Retrieval	Total
149	-	-	173	-	323

115x

0.93x slower!

Initializati	Allocation	Send	Execution	Retrieval	Total
142.0	182.1	14.2	1.5	5.4	345.1

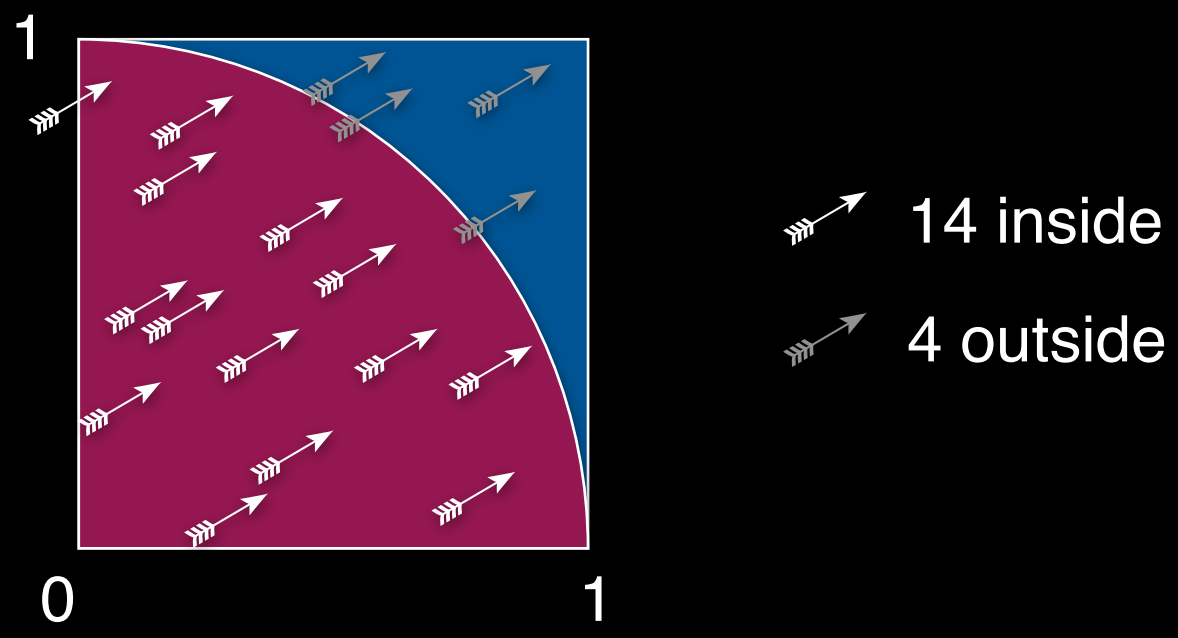
Overhead added

Computing π

UF Chang Florida

This time we will try to compute π

Basic idea of computing π will be via throwing "darts" randomly at a quarter of a unit circle



Since the area of the quarter of a unit circle is π/4 we can estimate π as

$\pi_{est} = 14 / (14+4) \times 4 = 3.11.....$

Title

UF
Chang
Florida

tests - JupyterLab | hsf-india-examples/rand.cu at m

https://jupyterhub.ssl-hep.org/user/p.chang@uf

File Edit View Run Kernel Tabs Settings Help

jovyan@jupyter-p-2echang

```
#include <iostream>
#include <stdio.h>

int main()
{
    std::cout << "################################" << std::endl;
    std::cout << "#                              #" << std::endl;
    std::cout << "#    Computing Pi via Darts    #" << std::endl;
    std::cout << "#                              #" << std::endl;
    std::cout << "################################" << std::endl;

    // we will launch 65536 blocks
    int grid_size = pow(2, 16);

    // we will generate 512 points each block
    int block_size = 512;

    // total threads
    int n_total_threads = grid_size * block_size;
}
~
~
"rand.cu" 20L, 598B written
                                                    20,1          All
```

Simple 1 0 Mem: 167.9... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2... 0

Check it compiles

```
$ nvcc rand.cu -o rand
```

Random number generation

```
#include <curand.h>
#include <curand_kernel.h>
```

UF Chang Florida

We will use the CUDA's API tool to perform RNG

We will throw `n_total_threads worth` of "darts" so we setup states for those

```
// pointer to the array of "curandState" on the device
curandState* state_device;

// malloc array of random state
cudaMalloc((void**) &state_device, n_total_threads * sizeof(curandState));
```

Then we set their states using a GPU kernel defined like:

```
__global__ void setup_curandState(curandState* state)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    curand_init(1234, idx, 0, &state[idx]);
}
```

Then we launch the kernel in a grid

```
setup_curandState<<<grid_size, block_size>>>(state_device);
```

tests - JupyterLab | hsf-india-examples/rand.cu at m

https://jupyterhub.ssl-hep.org/user/p.chang@uf

File Edit View Run Kernel Tabs Settings Help

jovyan@jupyter-p-2echang

```
#include <iostream>
#include <stdio.h>
#include <curand.h>
#include <curand_kernel.h>

int main()
{
    std::cout << "################################" << std::endl;
    std::cout << "#                              #" << std::endl;
    std::cout << "#    Computing Pi via Darts    #" << std::endl;
    std::cout << "#                              #" << std::endl;
    std::cout << "################################" << std::endl;

    // we will launch 65536 blocks
    int grid_size = pow(2, 16);

    // we will generate 512 points each block
    int block_size = 512;

    // total threads
    int n_total_threads = grid_size * block_size;

    // pointer to the array of “curandState” on the device
    curandState* state_device;

    // malloc array of random state
    cudaMalloc((void**) &state_device, n_total_threads * sizeof(curandState));
}
```

23,5 All

Simple 1 0 Mem: 167.9... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2... 0

UF Chang Florida

Title

Check it compiles

Title

UF
Chang
Florida

tests - JupyterLab | hsf-india-examples/rand.cu at m

https://jupyterhub.ssl-hep.org/user/p.chang@uf

File Edit View Run Kernel Tabs Settings Help

jovyan@jupyter-p-2echang

```
#include <stdio.h>
#include <curand.h>
#include <curand_kernel.h>

__global__ void setup_curandState(curandState* state)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    curand_init(1234, idx, 0, &state[idx]);
}

int main()
{
    std::cout << "################################" << std::endl;
    std::cout << "#                              #" << std::endl;
    std::cout << "#    Computing Pi via Darts    #" << std::endl;
    std::cout << "#                              #" << std::endl;
    std::cout << "################################" << std::endl;

"rand.cu" 40L, 1207B written
                                                        10,1          4%
```

Simple 1 0 Mem: 167.8... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2... 0

tests - JupyterLab | hsf-india-examples/rand.cu at m

https://jupyterhub.ssl-hep.org/user/p.chang@uf

File Edit View Run Kernel Tabs Settings Help

jovyan@jupyter-p-2echang

```
 7 {
 8     std::cout << "################################" << std::endl;
 9     std::cout << "#                              #" << std::endl;
10     std::cout << "#    Computing Pi via Darts    #" << std::endl;
11     std::cout << "#                              #" << std::endl;
12     std::cout << "################################" << std::endl;
13
14     // we will launch 65536 blocks
15     int grid_size = pow(2, 16);
16
17     // we will generate 512 points each block
18     int block_size = 512;
19
20     // total threads
21     int n_total_threads = grid_size * block_size;
22
23     // pointer to the array of “curandState” on the device
24     curandState* state_device;
25
26     // malloc array of random state
27     cudaMalloc((void**) &state_device, n_total_threads * sizeof(curandState));
28
29     // actually setup each random state with different index
30     setup_curandState<<<grid_size, block_size>>>(state_device);
31
32     // wait until all threads are done
33     cudaDeviceSynchronize();
34 }
"rand.cu" 34L, 1051B written
                                                                31,0-1        Bot
```

Simple 1 0 Mem: 167.9... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2... 0

Title

UF
Chang
Florida

Check it compiles

Now we setup a counter

UF Chang Florida

A counter in the device will count whether each dart thrown fell inside the quarter circle or not

tests - JupyterLab | hsf-india-examples/rand.cu at m

https://jupyterhub.ssl-hep.org/user/p.chang@uf

File Edit View Run Kernel Tabs Settings Help

jovyan@jupyter-p-2echang

```
30     curandState* state_device;
31
32     // malloc array of random state
33     cudaMalloc((void**) &state_device, n_total_threads * sizeof(curandState));
34
35     // actually setup each random state with different index
36     setup_curandState<<<grid_size, block_size>>>(state_device);
37
38     // wait until all threads are done
39     cudaDeviceSynchronize();
40
41     // setup a counter
42     int* n_inside_device;
43
44     // allocate memory
45     cudaMalloc((void**) &n_inside_device, sizeof(int));
46
47 }
"rand.cu" 47L, 1338B written
                                                    45,52          Bot
```

Simple 1 0 Mem: 167.9... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2... 0

“Inside? or outside?” kernel

UF Chang Florida

We define a “dart throwing” function like the following

```
__global__ void throw_dart(curandState* state, int* n_inside)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    double x = curand_uniform(&state[idx]);
    double y = curand_uniform(&state[idx]);
    double d = sqrt(x * x + y * y);
    if (d <= 1)
    {
        *n_inside += 1;
    }
}
```

parse thread idx of this thread

get two random numbers

get distance

Is this OK??

Now we throw darts like the following

```
throw_dart<<<grid_size, block_size>>>(state_device, n_inside_device);
```

“Inside? or outside?” kernel

UF Chang Florida

We define a “dart throwing” function like the following

```
__global__ void throw_dart(curandState* state, int* n_inside)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    double x = curand_uniform(&state[idx]);
    double y = curand_uniform(&state[idx]);
    double d = sqrt(x * x + y * y);
    if (d <= 1)
    {
        atomicAdd(n_inside, 1);
    }
}
```

parse thread idx of this thread

get two random numbers

get distance

Now we throw darts like the following

```
throw_dart<<<grid_size, block_size>>>(state_device, n_inside_device);
```

atomicAdd

UF Chang Florida

Each thread will try to count up the same memory
This can create a race condition

Race condition is when the result can depend on which thread finishes first (or when)

To avoid this we need to “block” the counting so that no two process can access the same memory

`atomicAdd` provides such feature

```
atomicAdd(n_inside, 1);
```

multiple threads will try to increase n_inside but now it will be properly counted

Other atomic operations

UF
Chang
Florida

- 7.14. Atomic Functions
  - 7.14.1. Arithmetic Functions
    - 7.14.1.1. atomicAdd()
    - 7.14.1.2. atomicSub()
    - 7.14.1.3. atomicExch()
    - 7.14.1.4. atomicMin()
    - 7.14.1.5. atomicMax()
    - 7.14.1.6. atomicInc()
    - 7.14.1.7. atomicDec()
    - 7.14.1.8. atomicCAS()

Title

UF
Chang
Florida

tests - JupyterLab | hsf-india-examples/rand.cu at m

https://jupyterhub.ssl-hep.org/user/p.chang@uf

File Edit View Run Kernel Tabs Settings Help

jovyan@jupyter-p-2echang

```
__global__ void setup_curandState(curandState* state)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    curand_init(1234, idx, 0, &state[idx]);
}

__global__ void throw_dart(curandState* state, int* n_inside)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    double x = curand_uniform(&state[idx]);
    double y = curand_uniform(&state[idx]);
    double d = sqrt(x * x + y * y);
    if (d <= 1)
    {
        atomicAdd(n_inside, 1);
    }
}
```

13,1 6%

Simple 1 0 Mem: 166.7... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2... 0

Title

UF
Chang
Florida

tests - JupyterLab | hsf-india-examples/rand.cu at m

https://jupyterhub.ssl-hep.org/user/p.chang@uf

File Edit View Run Kernel Tabs Settings Help

jovyan@jupyter-p-2echang

```
    setup_curandState<<<grid_size, block_size>>>(state_device);

    // wait until all threads are done
    cudaDeviceSynchronize();

    // setup a counter
    int* n_inside_device;

    // allocate memory
    cudaMalloc((void**) &n_inside_device, sizeof(int));

    throw_dart<<<grid_size, block_size>>>(state_device, n_inside_device);

    // wait until all threads are done
    cudaDeviceSynchronize();


}
"rand.cu" 66L, 1780B written
                                                        62,5          Bot
```

Simple 1 0 Mem: 167.81... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2... 0

Retrieving the result

UF Chang Florida

Make a memory on host and copy back

```
// create a counter on host to copy device number to
int* n_inside_host = new int;

// copy the result to host
cudaMemcpy(n_inside_host, n_inside_device, sizeof(int), cudaMemcpyDeviceToHost);
```

Then use the value to compute pi

```
// estimate pi by counting fraction
double pi_estimate = (double) *n_inside_host / n_total_threads * 4.;

// print pi_estimate
std::cout << " --- Result ---" << std::endl;
std::cout <<  " pi_estimate: " << pi_estimate <<  std::endl;
```

Title

UF
Chang
Florida

tests - JupyterLab | hsf-india-examples/rand.cu at m

https://jupyterhub.ssl-hep.org/user/p.chang@uf

File Edit View Run Kernel Tabs Settings Help

jovyan@jupyter-p-2echang

```
64    cudaDeviceSynchronize();
65
66    // create a counter on host to copy device number to
67    int* n_inside_host = new int;
68
69    // copy the result to host
70    cudaMemcpy(n_inside_host, n_inside_device, sizeof(int), cudaMemcpyDeviceToHost);
71
72    // estimate pi by counting fraction
73    double pi_estimate = (double) *n_inside_host / n_total_threads * 4.;
74
75    // print pi_estimate
76    std::cout << " --- Result ---" << std::endl;
77    std::cout <<  " pi_estimate: " << pi_estimate <<  std::endl;
78
79    return 0;
80
81 }
```

81,1 Bot

Simple 1 0 Mem: 166.7... jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2... 0

Title

UF
Chang
Florida

More refined version is here:
https://raw.githubusercontent.com/sgnoohc/hsf-india-examples/main/rand.cu

Result

UF Chang Florida

```
(myenv) jovyan@jupyter-p--research-2dsoftwa-2---dindia-2dmay2024-2d30ogek5h:~/hsf-india-examples$ ./rand
################################
#                              #
#    Computing Pi via Darts    #
#                              #
################################
 --- Input data ---
 grid_size          = 65536
 block_size         = 512
 total darts thrown = 33554432

 --- Result ---
 pi_estimate: 3.14147
```

Title

Matrix Summation

This time we will try adding a matrix to another matrix

For simplicity we will declare one matrix of 2048 × 2048 size with element of all set to 1

dim3

This time we will use a different object called “dim3” dim3 is basically a three tuple (x, y, z) that can hold three integer

example:

```
dim3 block_size_ex1(16, 16, 16);
dim3 block_size_ex2(16, 16, 1);
```

We can use this to launch 3d grid / 3d blocks

dim3

UF Chang Florida

In our case we want to launch a 1 grid of 16 x 16 block
So we define them like the following

```
// we will perform each element as one thread
int block_len = 16;

// then the block dimensions are defined
dim3 block_size(block_len, block_len, 1);

// compute number of blocks in each dimension
int grid_len = int(m_dim - 0.5) / block_len + 1;

// then for grid size needs to be computed to cover the entire elements
dim3 grid_size(grid_len, grid_len, 1);
```

And we can use it like the following

```
kernel<<<grid_size, block_size>>>(…)
```

cudaMallocHost

UF Chang Florida

Previously we have done something like this

```
float* A_host = new float[n_data];
float* B_host = new float[n_data];
float* C_host = new float[n_data];
```

But one could have instead done this

```
float* A_host;
float* B_host;
float* C_host;
cudaMallocHost((void**) &A_host, n_data * sizeof(float))
cudaMallocHost((void**) &B_host, n_data * sizeof(float))
cudaMallocHost((void**) &C_host, n_data * sizeof(float))
```

Why.....??

Copying data WHILE processing

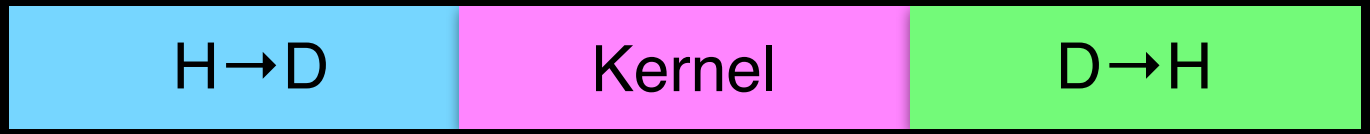
One of the biggest power of GPU is that it can process data while copying stuff in the background!

This can help eliminate or reduce overhead!

For example consider the normal case

```
cudaMemcpy(…, cudaMemcpyHostToDevice);
kernel<<<…,…>>>(…);
cudaMemcpy(…, cudaMemcpyDeviceToHost);
```

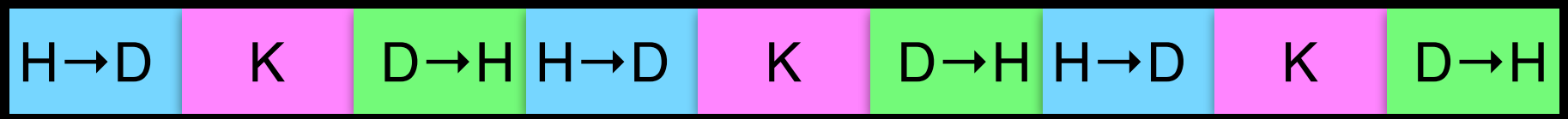
This will process



If you repeatedly process this

UF Chang Florida

Things will all happen in sequence



What if you could stagger?

If you repeatedly process this

UF Chang Florida

Things will all happen in sequence

H→D	K	D→H	H→D	K	D→H	H→D	K	D→H

What if you could stagger?

H→D	K	D→H		
	H→D	K	D→H	
		H→D	K	D→H

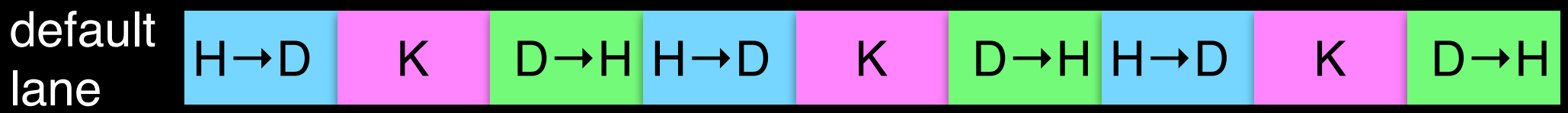
You would win!

cudaStream

UF Chang Florida

in order to stagger and schedule the cuda API or kernel calls, once has to define “lanes” or “streams”

Previously when nothing was specificed they were all running on the so-called “default lane”

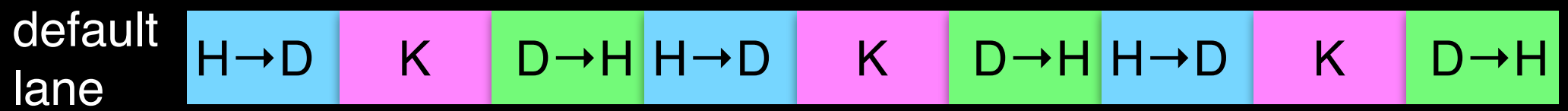


cudaStream

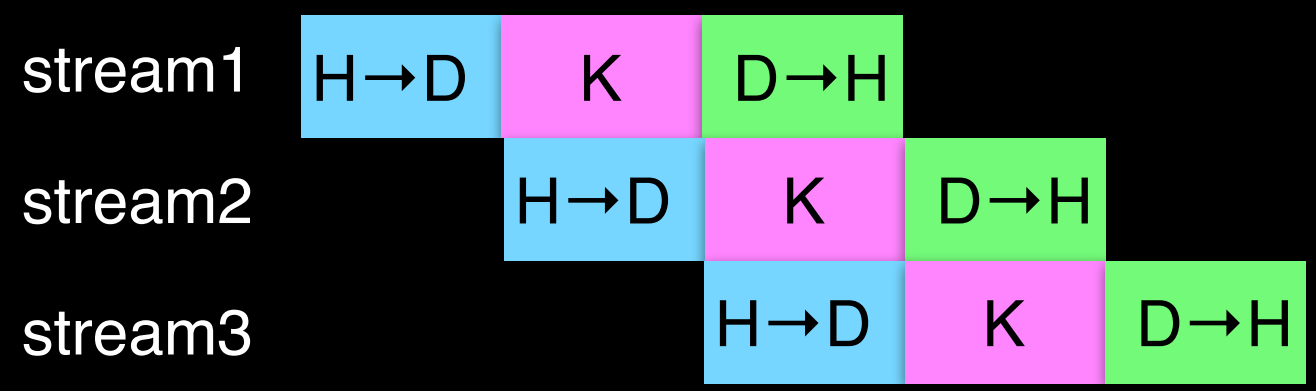
UF Chang Florida

in order to stagger and schedule the cuda API or kernel calls, once has to define “lanes” or “streams”

Previously when nothing was specificed they were all running on the so-called “default lane”



Instead one can define different streams and schedule them



Creating cudaStream


```
// create cuda streams
cudaStream_t stream[n_repeat];
for (int i = 0; i < n_repeat; ++i)
{
    cudaStreamCreate(&stream[i]);
}
```

Simply create cudaStream_t objects

How do I schedule different cudaAPI/kernel to different streams?

For memory copy, we use

```
cudaMemcpyAsync
```

Assuming we have stream[0], stream[1], … created, we would do

```
cudaMemcpyAsync(a_device,
                a_host,
                ntot*sizeof(float),
                cudaMemcpyHostToDevice,
                stream[1])
```

For Kernel calls

For kernel calls we add it to the fourth arguments

```
kernel<<<grid_size, block_size, 0, stream[1]>>>
```

(The third argument is not discussed today, it has to do with shared memory, but I have not particularly found good use of it, so I set it to 0 the default value)

Finish coding up

Refined example here:
https://raw.githubusercontent.com/sgnoohc/hsf-india-examples/main/madd.cu

Title

UF
Chang
Florida

```
jovyan@jupyter-p-2echang-40ufl-2eedu--research-2dsoftwa-2df-2dindia-2d:~$ ./madd
################################
#                              #
#         Matrix Sum           #
#     (Overlap Transfer)       #
#                              #
################################
 --- Sequential Run ---
 Time total (ms): 17.308865

 --- Overlapping Run ---
 Time total (ms): 9.332256
```

Profiler

There are several profilers in Nvidia toolkit

Today I will use Nvidia Visual Profiler (nvvp) to show how the staggering of the data copy call vs. kernel calls look like

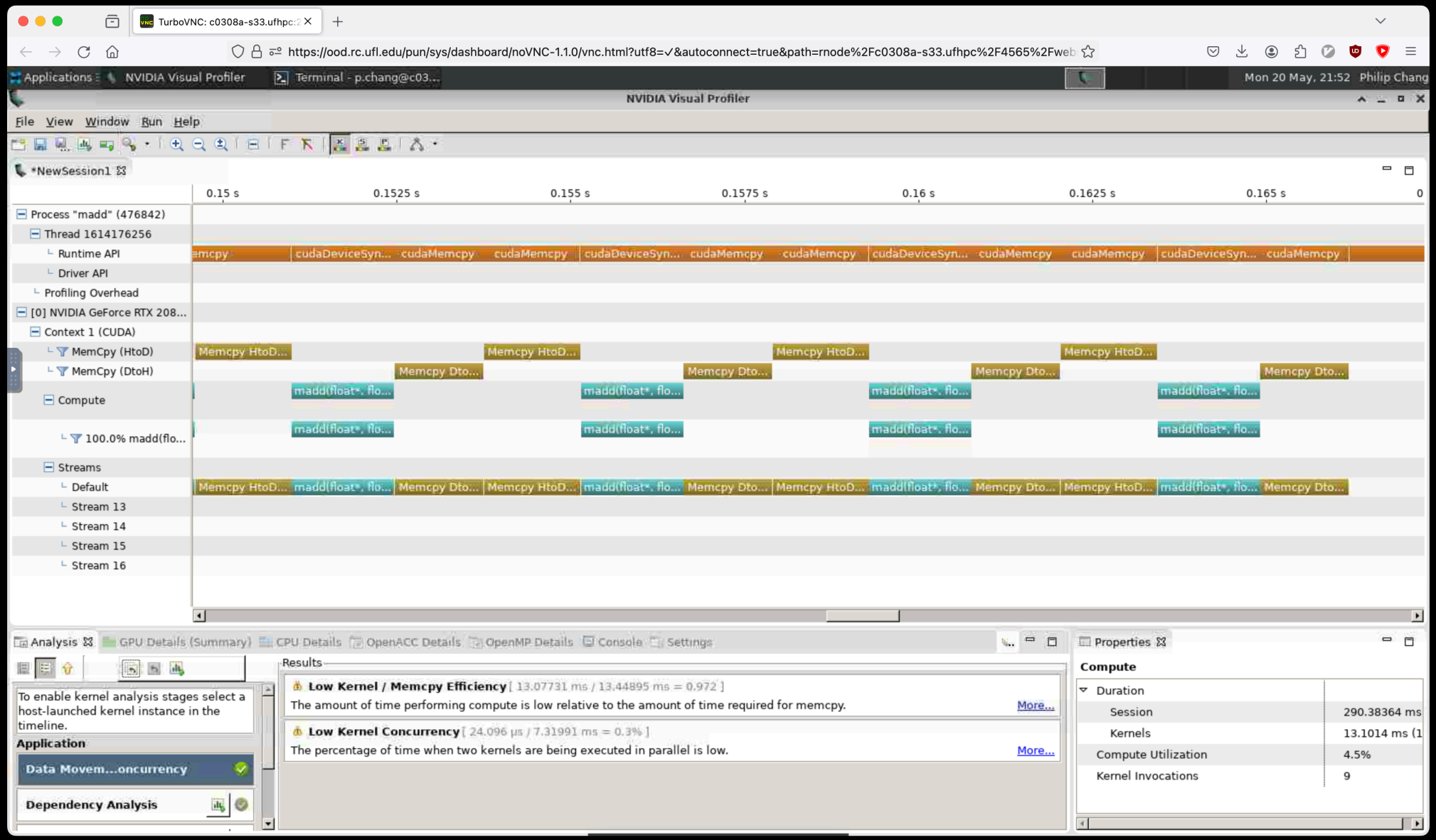
Once the program is compiled

UF
Chang
Florida

```
nvvp ./madd
```

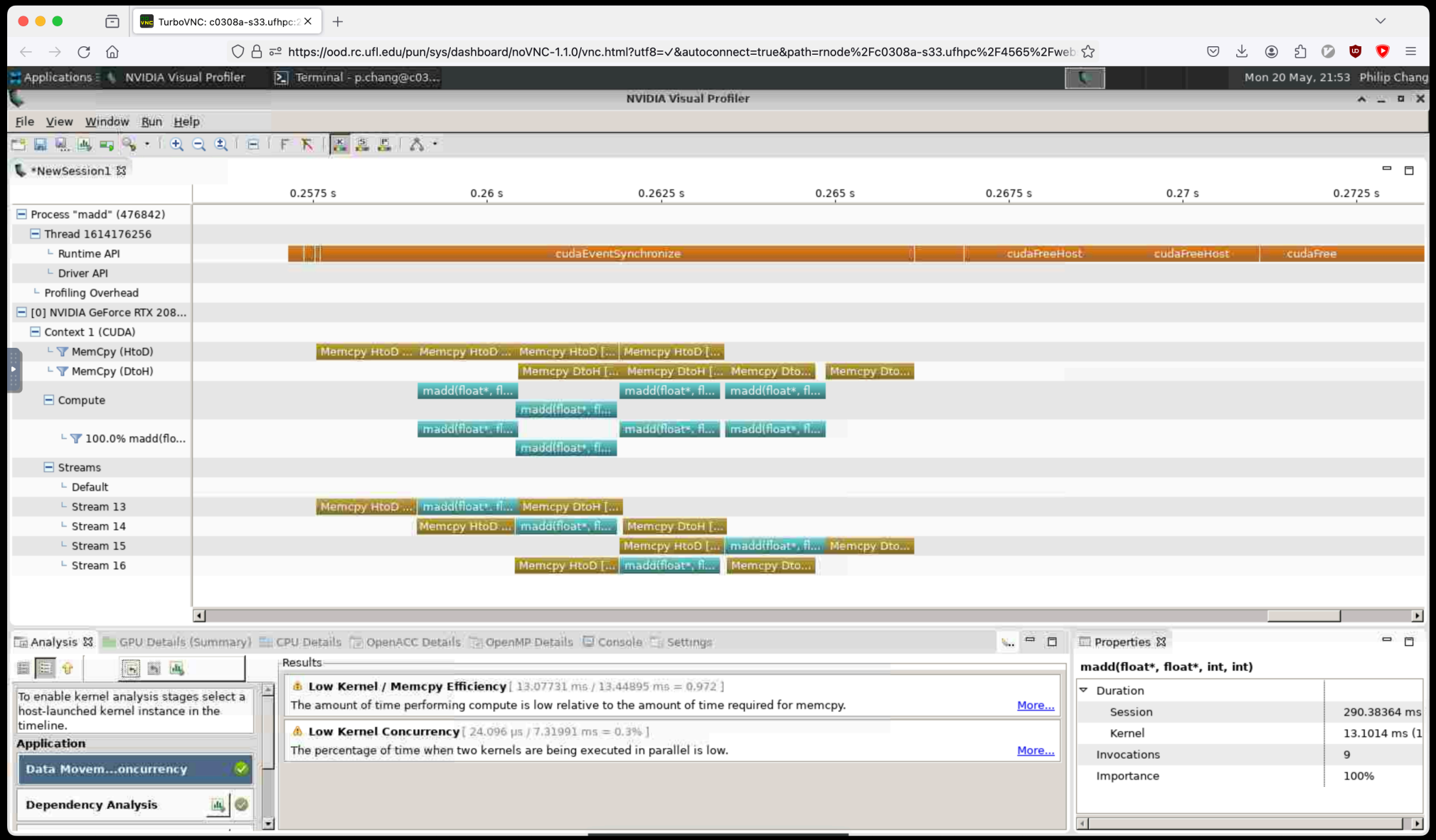
Non-staggered example

UF
Chang
Florida



Staggered example

UF
Chang
Florida



Title

Some tools

UF Chang Florida

Parsing command line for large number

```
#include <cstdlib>


int main(int argc, charg** argv)
{
    unsigned long long int N_data = strtoull(argv[1], nullptr, 10);
}
```

Printing out information and putting requirements on input arguments

```
#include <iostream>

if (argc < 2)
{
    std::cout << "Usage:" << std::endl;
    std::cout << std::endl;
    std::cout << "    " << argv[0] << " N_data" << std::endl;
    std::cout << std::endl;
    std::cout << std::endl;
    return 1;
}
```