Before we start...

- This is technically a ‘members-only’ event (for student org. purposes.)
- Luckily, we count anyone in our Discord a member!

http://puhack.horse/discord

Before we start...

Workshop structure:

- **~10 minutes**: theory.
-
-

Before we start...

Workshop structure:

- **~10 minutes**: theory.
- **~30 minutes**: guided coding.
-

Before we start...

Workshop structure:

- **~10 minutes**: theory.
- **~30 minutes**: guided coding.
- the rest of the time: self-directed hacking.

Before we start...

Workshop structure:

- **~10 minutes**: theory.
- **~30 minutes**: guided coding.
- the rest of the time: self-directed hacking.

guided coding: you'll write a matrix multiplication kernel.

Before we start...

Workshop structure:

- **~10 minutes**: theory.
- **~30 minutes**: guided coding.
- the rest of the time: self-directed hacking.

guided coding: you'll write a matrix multiplication kernel.

hacking: write some other kernels. maybe convolution.

Before we start...

Hi! Rules for this presentation:

- You ***should*** ask questions during the presentation.
-

Before we start...

Hi! Rules for this presentation:

- You ***should*** ask questions during the presentation.
- in fact...

Ask me a question right now.

Before we start...

Hi! Rules for this presentation:

- You ***should*** ask questions during the presentation.
- in fact...

Ask me a question right now.

Another.

Introduction to GPU Programming

Kartavya Vashishtha

February 26, 2026

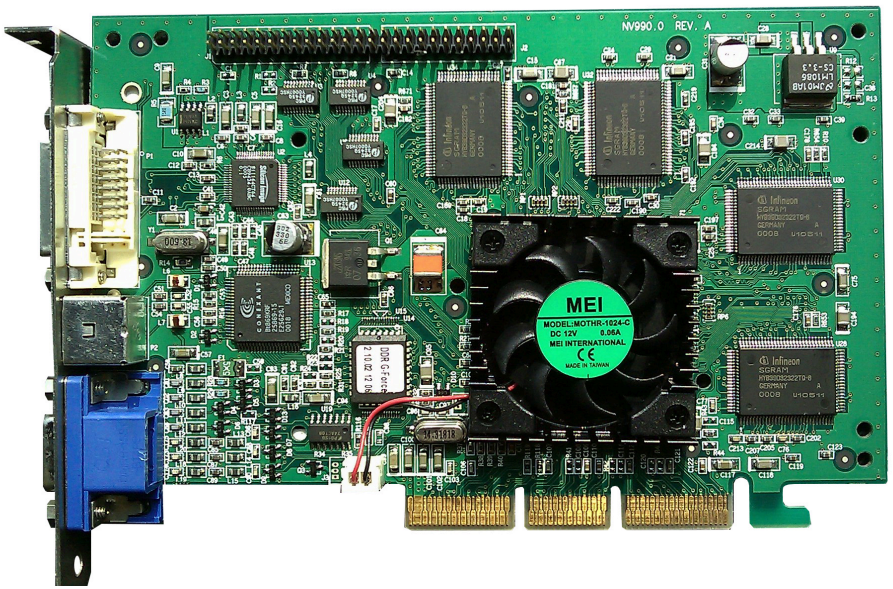
A GPU from a thousand feet away

A GPU from a thousand feet away

where is it?!

A GPU from a thousand feet away

A GPU from a few inches away



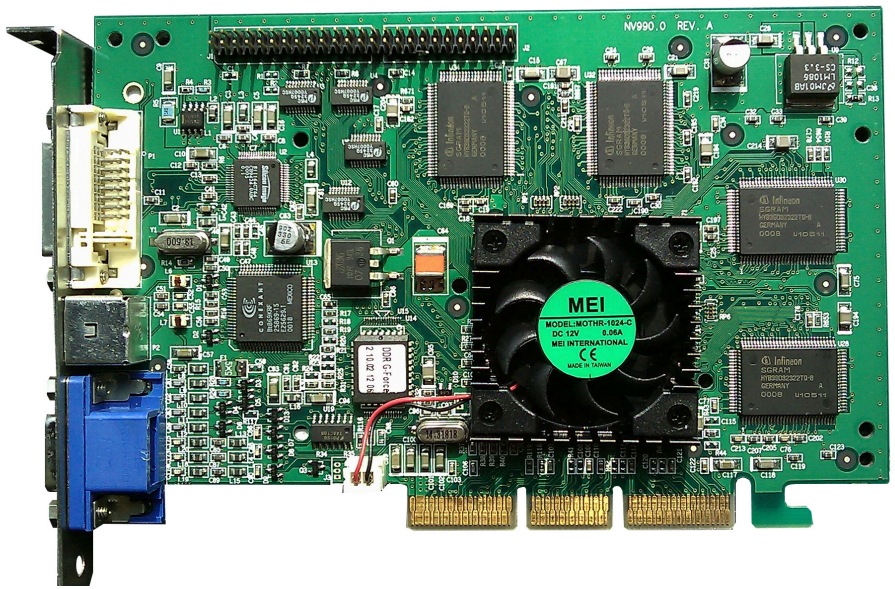
120 MHz Nvidia GeForce 256

- **not** a CPU
-

A GPU from a few inches away

- **not** a CPU
- trades single-thread speed for many-thread throughput

  (more on this later!)



120 MHz Nvidia GeForce 256

A History of GPU Usage

1. **Graphics era:** Fixed-function rendering. Push pixels to a buffer, modify, and dump to screen. Rigid APIs.

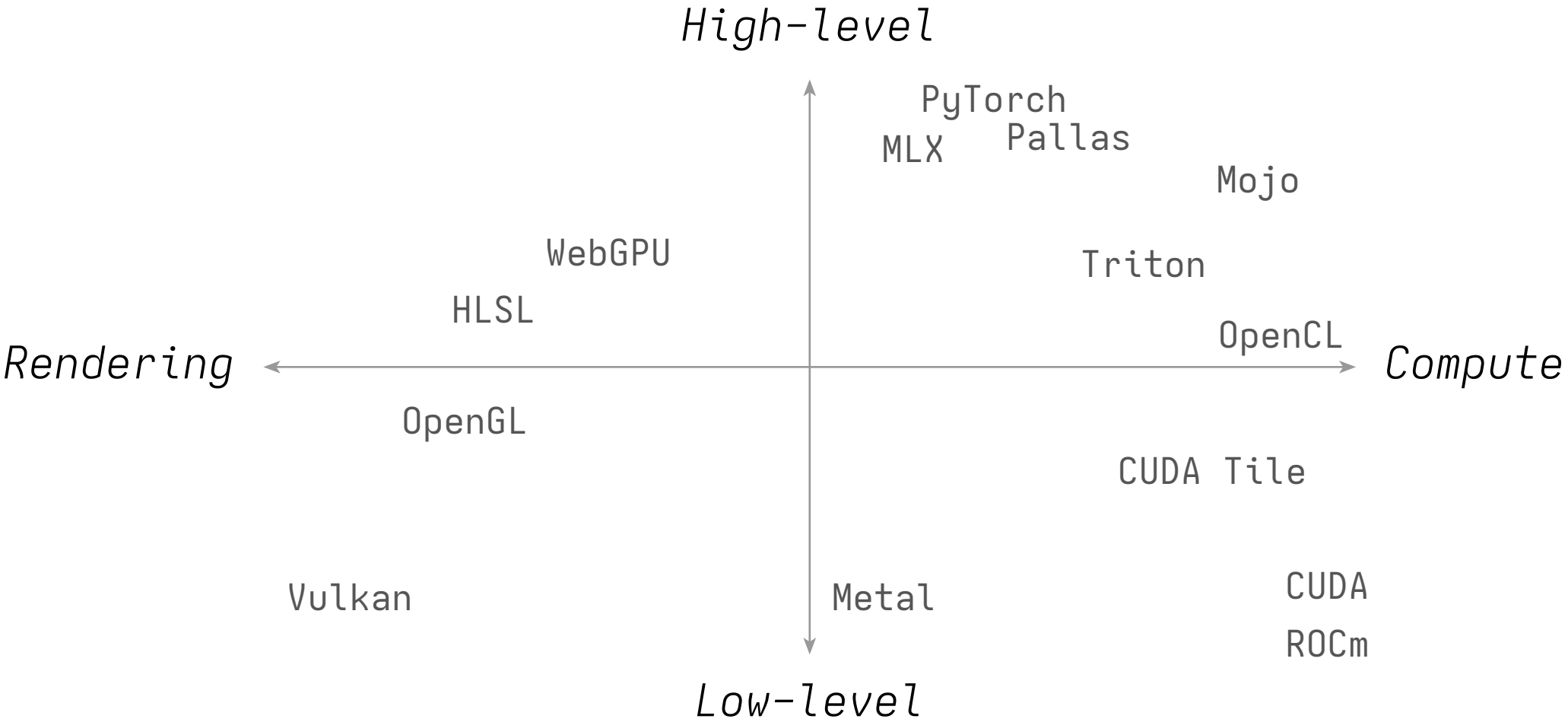
A History of GPU Usage

1. **Graphics era:** Fixed-function rendering. Push pixels to a buffer, modify, and dump to screen. Rigid APIs.
2. **Mid-2000s:** *what if... we use these chips for things that aren't games*. NVIDIA releases CUDA (2007) — general-purpose computing API.

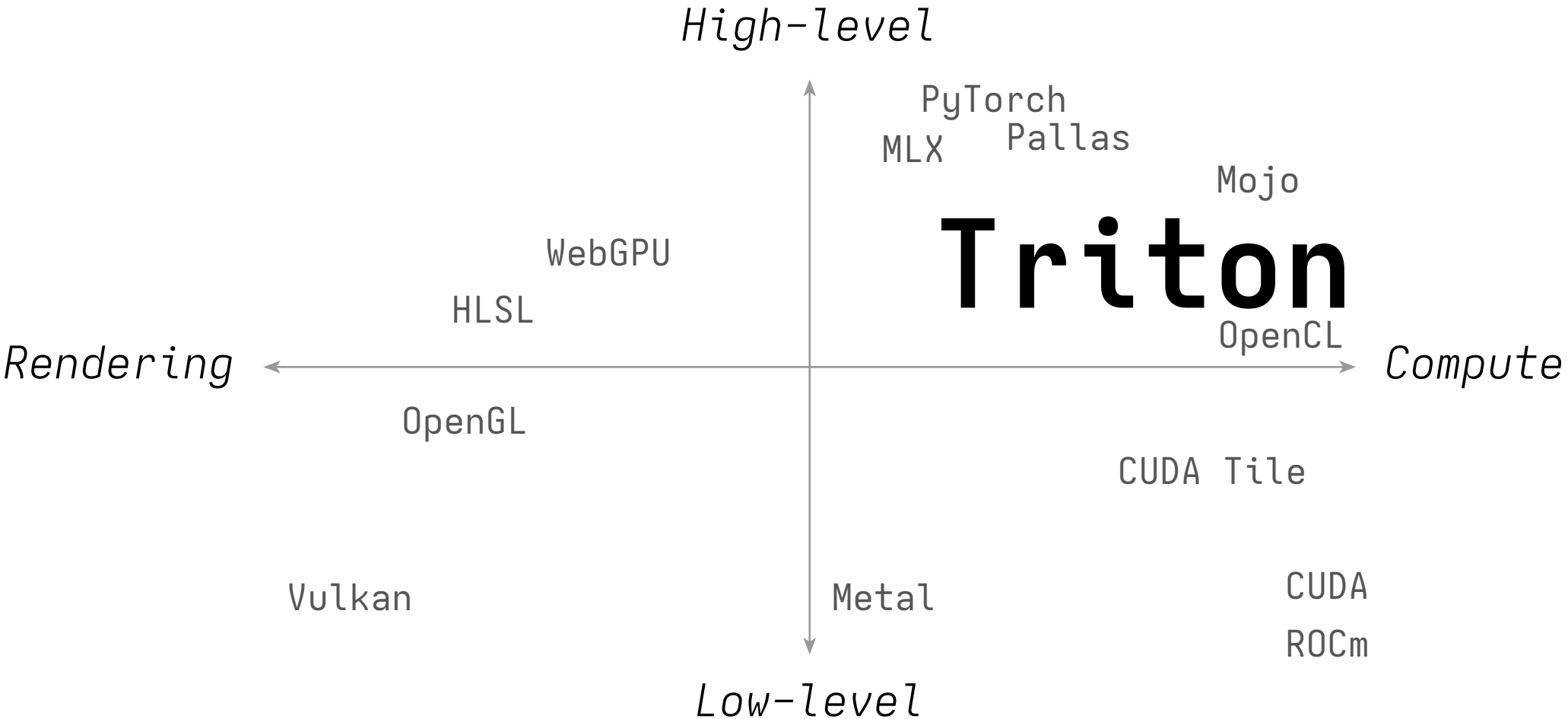
A History of GPU Usage

1. **Graphics era:** Fixed-function rendering. Push pixels to a buffer, modify, and dump to screen. Rigid APIs.
2. **Mid-2000s:** *what if... we use these chips for things that aren't games*. NVIDIA releases CUDA (2007) — general-purpose computing API.
3. **2010s:** people start running machine learning on them.

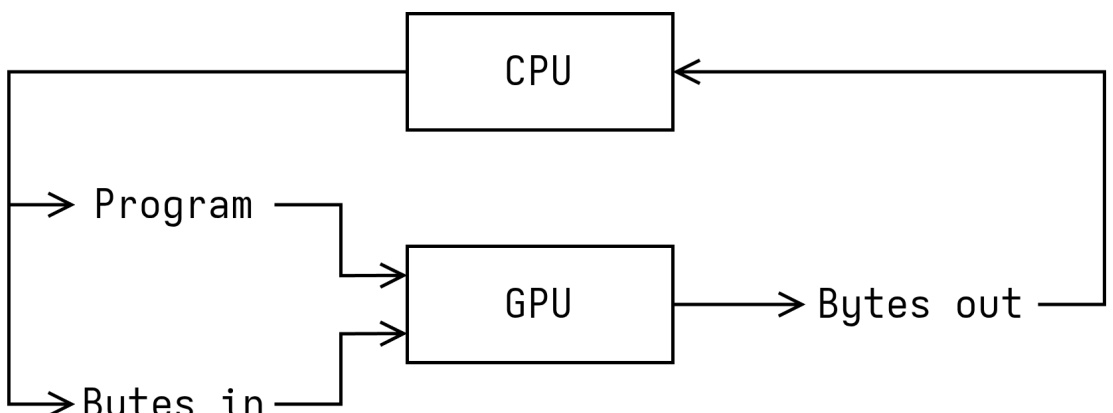
Many ways to program a GPU



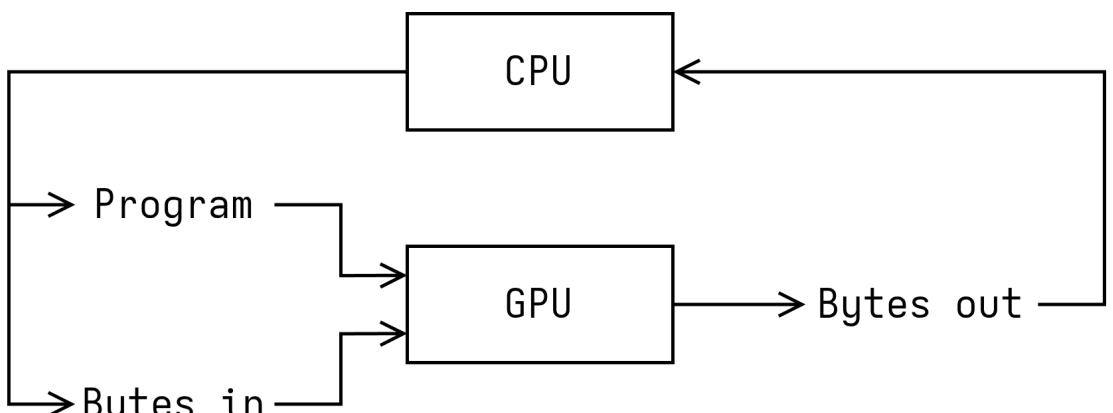
Many ways to program a GPU



A simple model



An (overly) simple model



Switching gears: time to write ReLU!

Recall, or learn very quickly,

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

Pseudocode:

Switching gears: time to write ReLU!

Recall, or learn very quickly,

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

Pseudocode:

```
def relu(x):
  return max(x, 0)
```

Switching gears: time to write ReLU!

Recall, or learn very quickly,

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

Pseudocode:

```
def relu(x):
  return max(x, 0)
```

For an array:

Switching gears: time to write ReLU!

Recall, or learn very quickly,

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

Pseudocode:

```
def relu(x):
  return max(x, 0)
```

For an array:

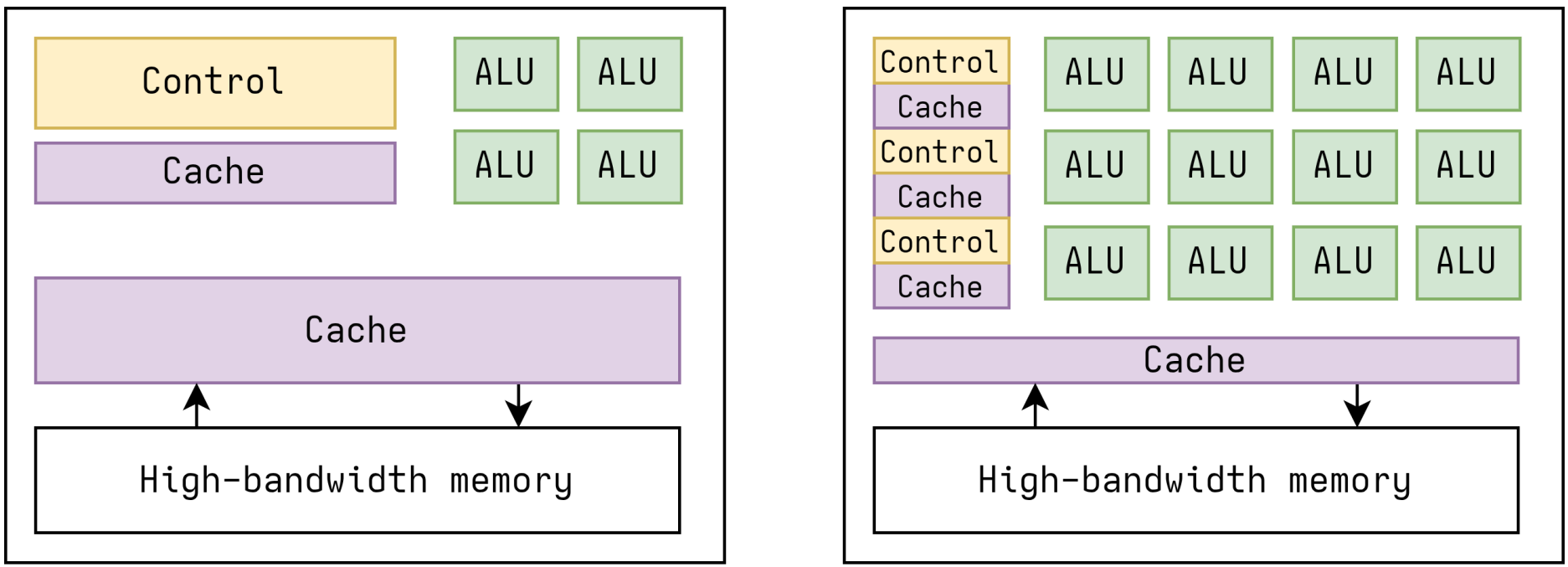
```
def relu_array(arr):
  return [max(x, 0) for x in arr]
```

A GPU from a few millimeters away

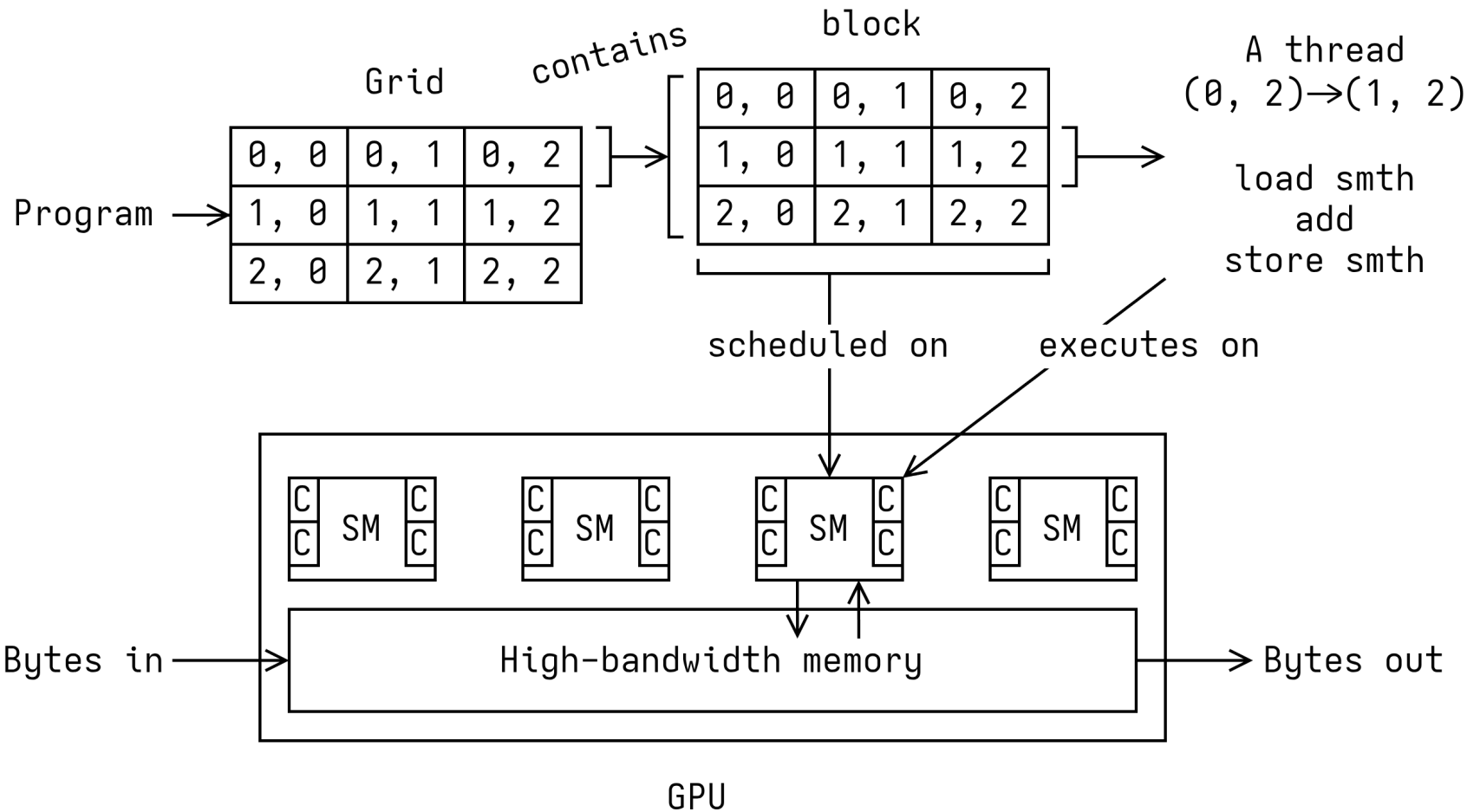
- trades single-thread speed for many-thread throughput
- question: ***how do you divide work between these threads?***

A GPU from a few millimeters away

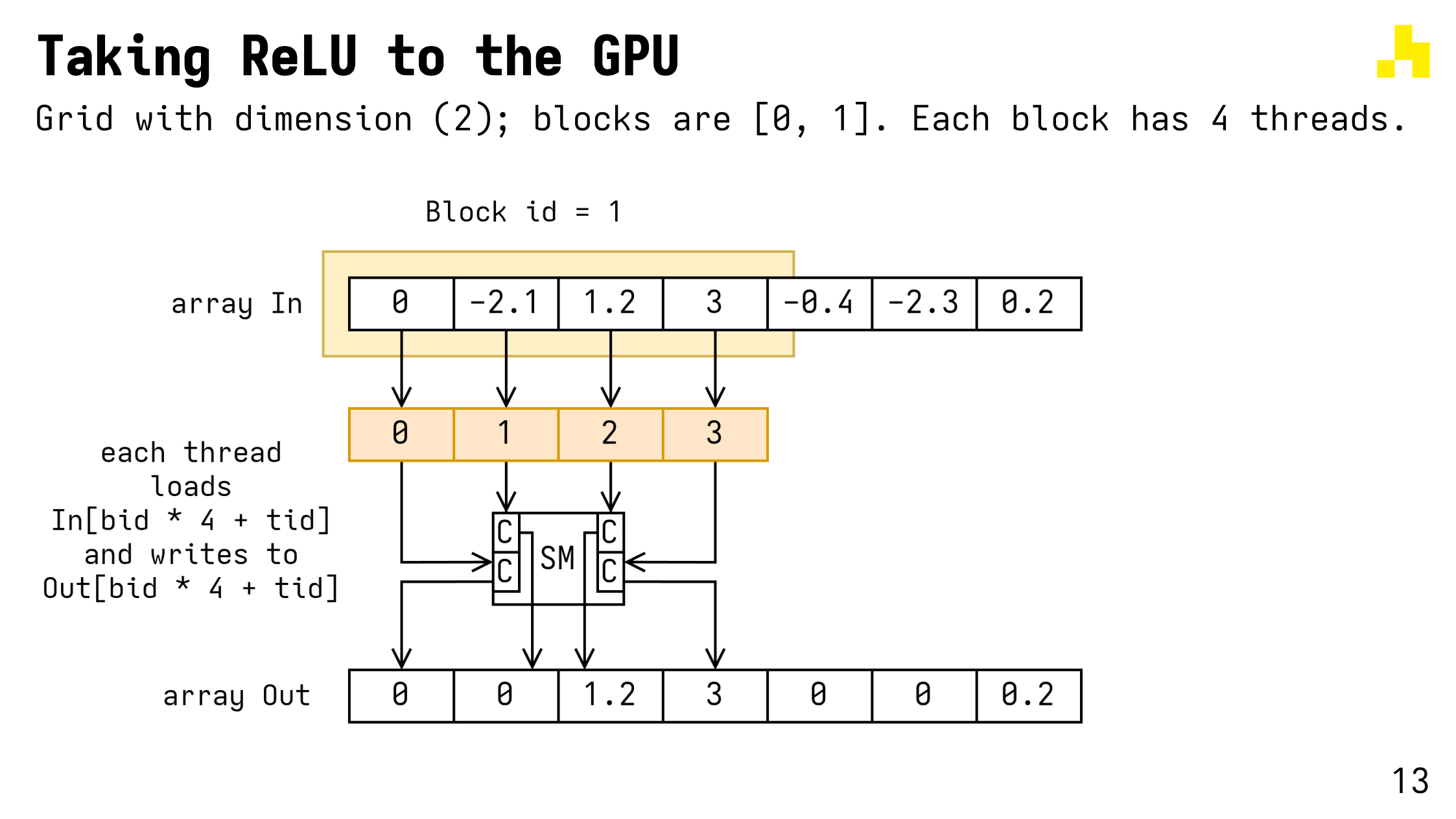
- trades single-thread speed for many-thread throughput
- question: ***how do you divide work between these threads?***



A simple model for a GPU



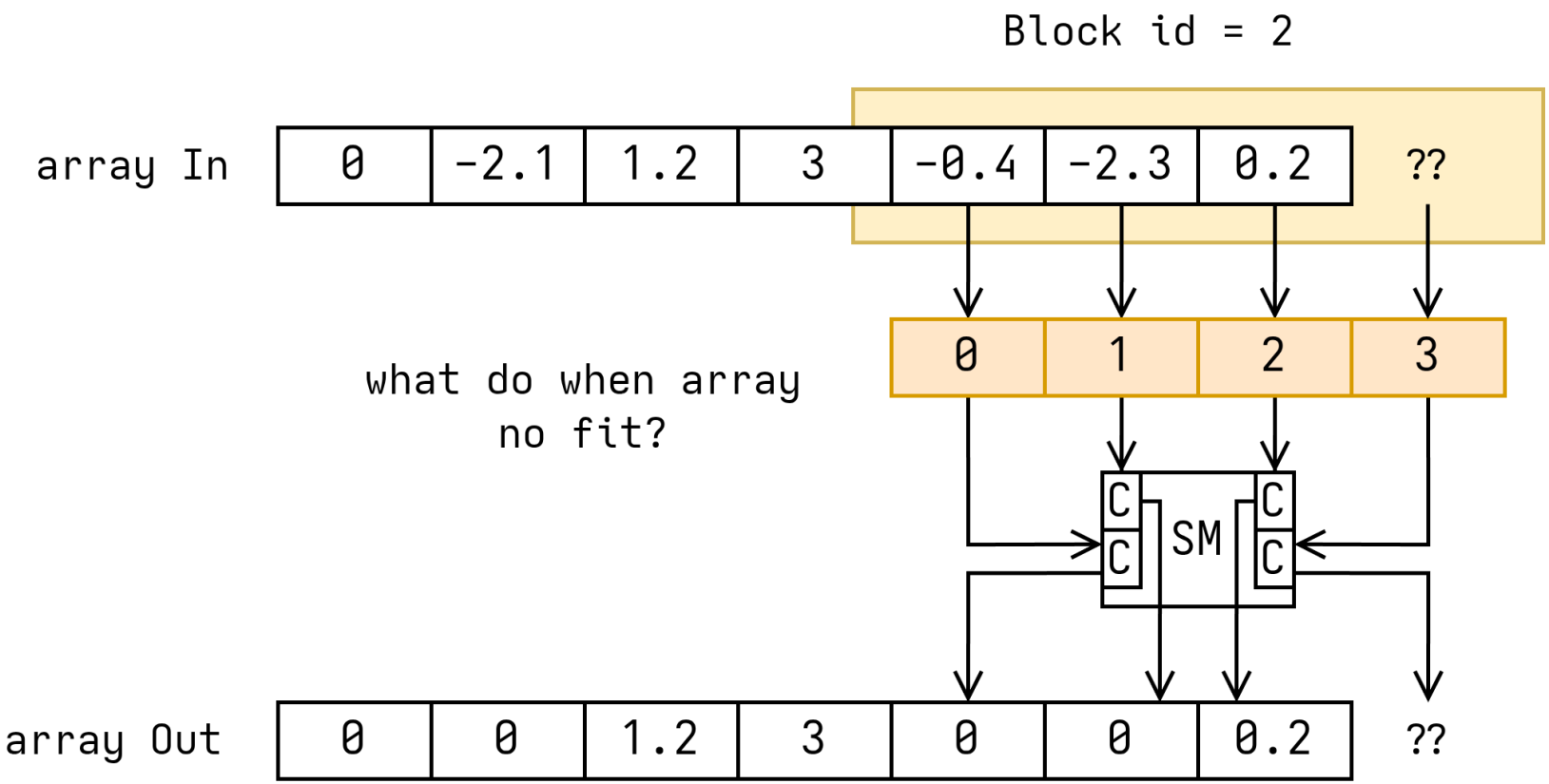
Taking ReLU to the GPU



Grid with dimension (2); blocks are [0, 1]. Each block has 4 threads.

Taking ReLU to the GPU

What happens at block 1?



Time to move over to writing code!!

- go to https://puhack.horse/gpu-colab
- click ‘Copy to Drive’ in top bar.

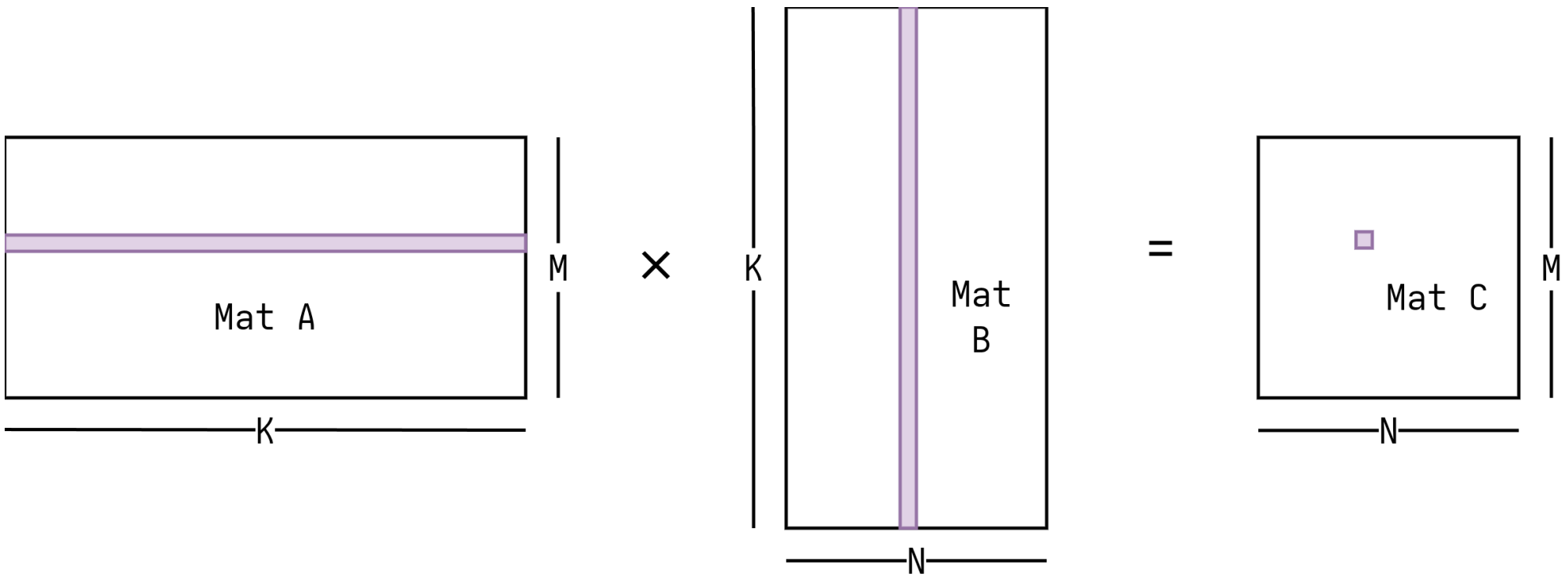
An example problem decomposition

Hm. What sort of problems can be broken into parallel chunks?

An example problem decomposition

Hm. What sort of problems can be broken into parallel chunks?

Enter the humble matmul.



The humble matmul


```
for i in range(M):
    for j in range(N):
        for k in range(K):
            C[i, j] += A[i, k] * B[k, j]
```

- Imagine one thread per (i, j) index:
  - Thread (0, 0) loads **A[0, k]** and **B[k, 0]**
  - Thread (0, 1) loads A[0, k] and **B[k, 1]**
  - Thread (1, 0) loads **A[1, k]** and B[k, 0]
  - etc.
-
-
-
-

The humble matmul


```
for i in range(M):
    for j in range(N):
        for k in range(K):
            C[i, j] += A[i, k] * B[k, j]
```

- Imagine one thread per (i, j) index:
  - Thread (0, 0) loads **A[0, k]** and **B[k, 0]**
  - Thread (0, 1) loads A[0, k] and **B[k, 1]**
  - Thread (1, 0) loads **A[1, k]** and B[k, 0]
  - etc.
- GPUs execute a whole block on one compute unit.
-
-
-

The humble matmul


```
for i in range(M):
    for j in range(N):
        for k in range(K):
            C[i, j] += A[i, k] * B[k, j]
```

- Imagine one thread per (i, j) index:
  - Thread (0, 0) loads **A[0, k]** and **B[k, 0]**
  - Thread (0, 1) loads A[0, k] and **B[k, 1]**
  - Thread (1, 0) loads **A[1, k]** and B[k, 0]
  - etc.
- GPUs execute a whole block on one compute unit.
- This compute unit *has very fast memory*.
-
-

The humble matmul


```
for i in range(M):
    for j in range(N):
        for k in range(K):
            C[i, j] += A[i, k] * B[k, j]
```

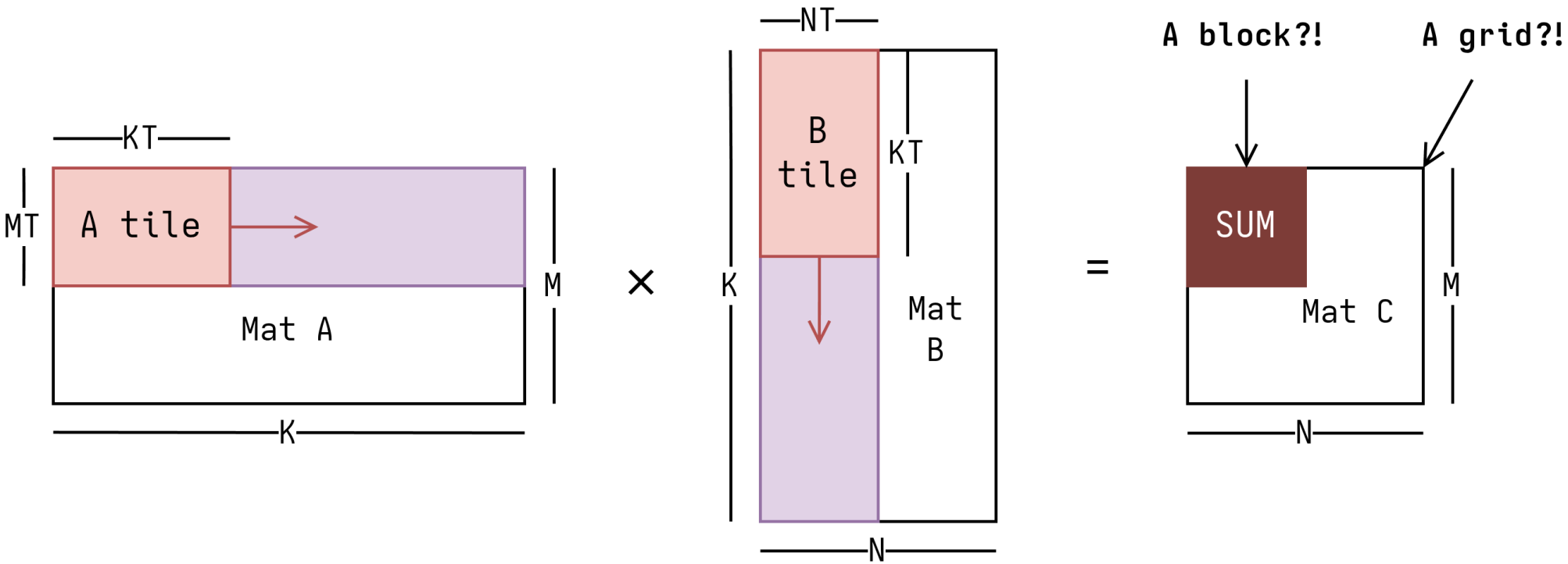
- Imagine one thread per (i, j) index:
  - Thread (0, 0) loads **A[0, k]** and **B[k, 0]**
  - Thread (0, 1) loads A[0, k] and **B[k, 1]**
  - Thread (1, 0) loads **A[1, k]** and B[k, 0]
  - etc.
- GPUs execute a whole block on one compute unit.
- This compute unit *has very fast memory*.
- Sharing these accesses is *good*.
-

The humble matmul


```
for i in range(M):
    for j in range(N):
        for k in range(K):
            C[i, j] += A[i, k] * B[k, j]
```

- Imagine one thread per (i, j) index:
  - Thread (0, 0) loads **A[0, k]** and **B[k, 0]**
  - Thread (0, 1) loads A[0, k] and **B[k, 1]**
  - Thread (1, 0) loads **A[1, k]** and B[k, 0]
  - etc.
- GPUs execute a whole block on one compute unit.
- This compute unit *has very fast memory*.
- Sharing these accesses is *good*.
- So let's write this with tiles!

Presenting: the tiled matmul



Tiled matmul pseudocode


```
def matmul_kernel(A, B, C, M, N, K):
    c_m, c_n = program_id(0), program_id(1)
    accumulator = zeros((MT, NT))

    for k in range(0, K, KT):
        tile_A = A[c_m * MT : (c_m + 1) * MT,        k : k + KT        ]
        tile_B = B[       k : k + KT        , c_n * NT : (c_n + 1) * NT]

        accumulator += dot(tile_A, tile_B)

    C[c_m * MT : (c_m + 1) * MT, c_n * NT : (c_n + 1) * NT] = accumulator
```

Tiled matmul pseudocode


```
def matmul_kernel(A, B, C, M, N, K):
    c_m, c_n = program_id(0), program_id(1)
    accumulator = zeros((MT, NT))

    for k in range(0, K, KT):
        tile_A = A[c_m * MT : (c_m + 1) * MT,        k : k + KT        ]
        tile_B = B[       k : k + KT        , c_n * NT : (c_n + 1) * NT]

        accumulator += dot(tile_A, tile_B)

    C[c_m * MT : (c_m + 1) * MT, c_n * NT : (c_n + 1) * NT] = accumulator
```

CODING TIEM!! ← so excited I cna't spell

Hacking Section!

You can:

- go to https://tensara.org/

or

- go to https://puhack.horse/gpu-colab-conv