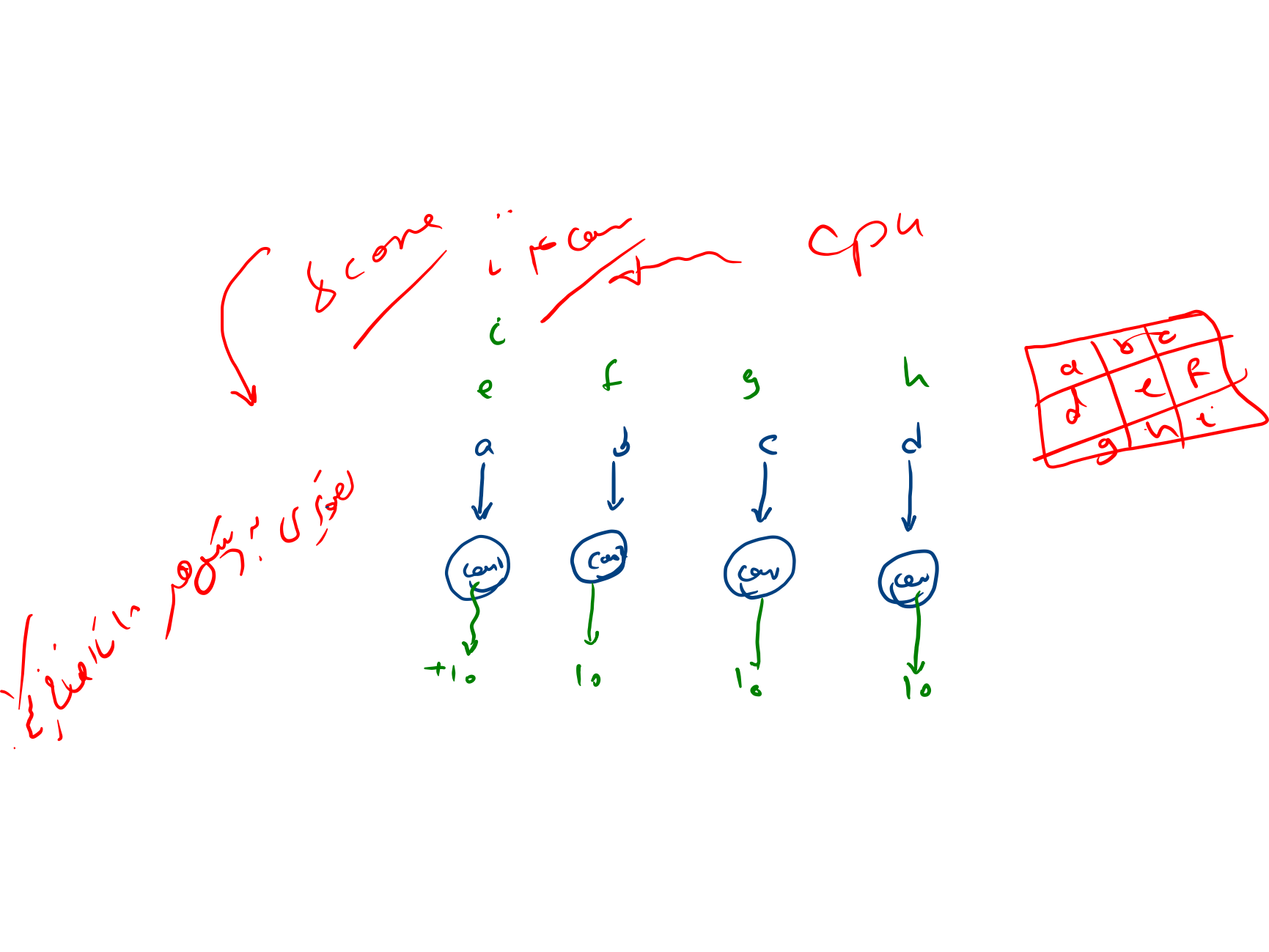
GPU Programming.

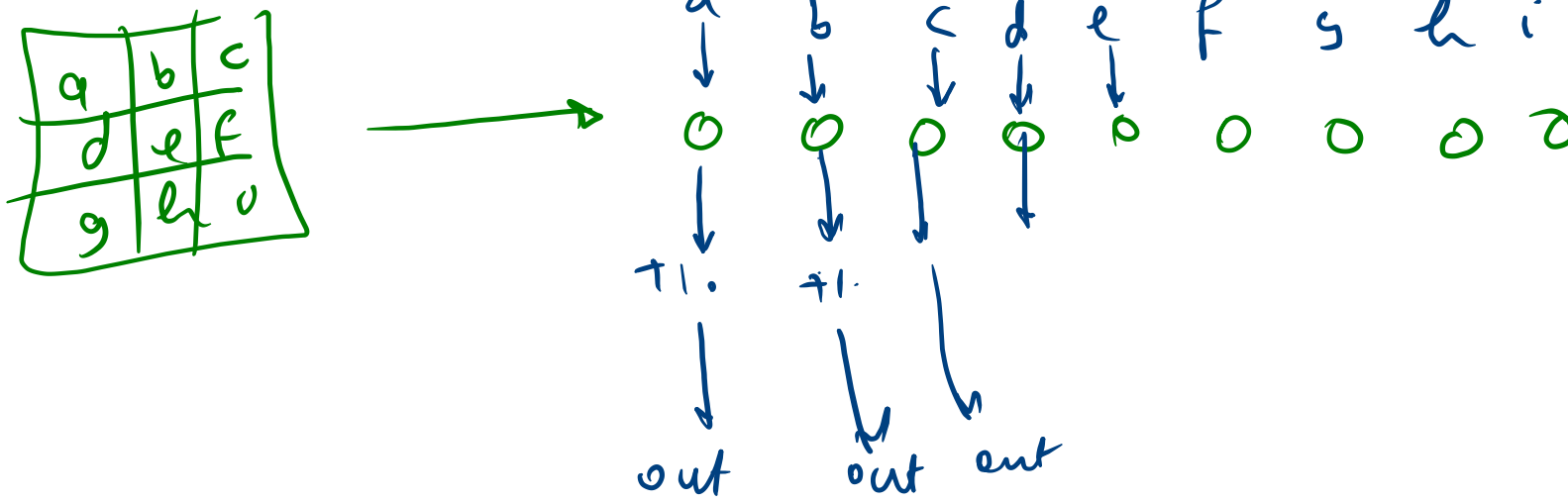

Gpu


core
cpu
c
e
f
g
h
a
b
c
d
+10
10
10
10
a b c
d e f
g h i

1 Sequential

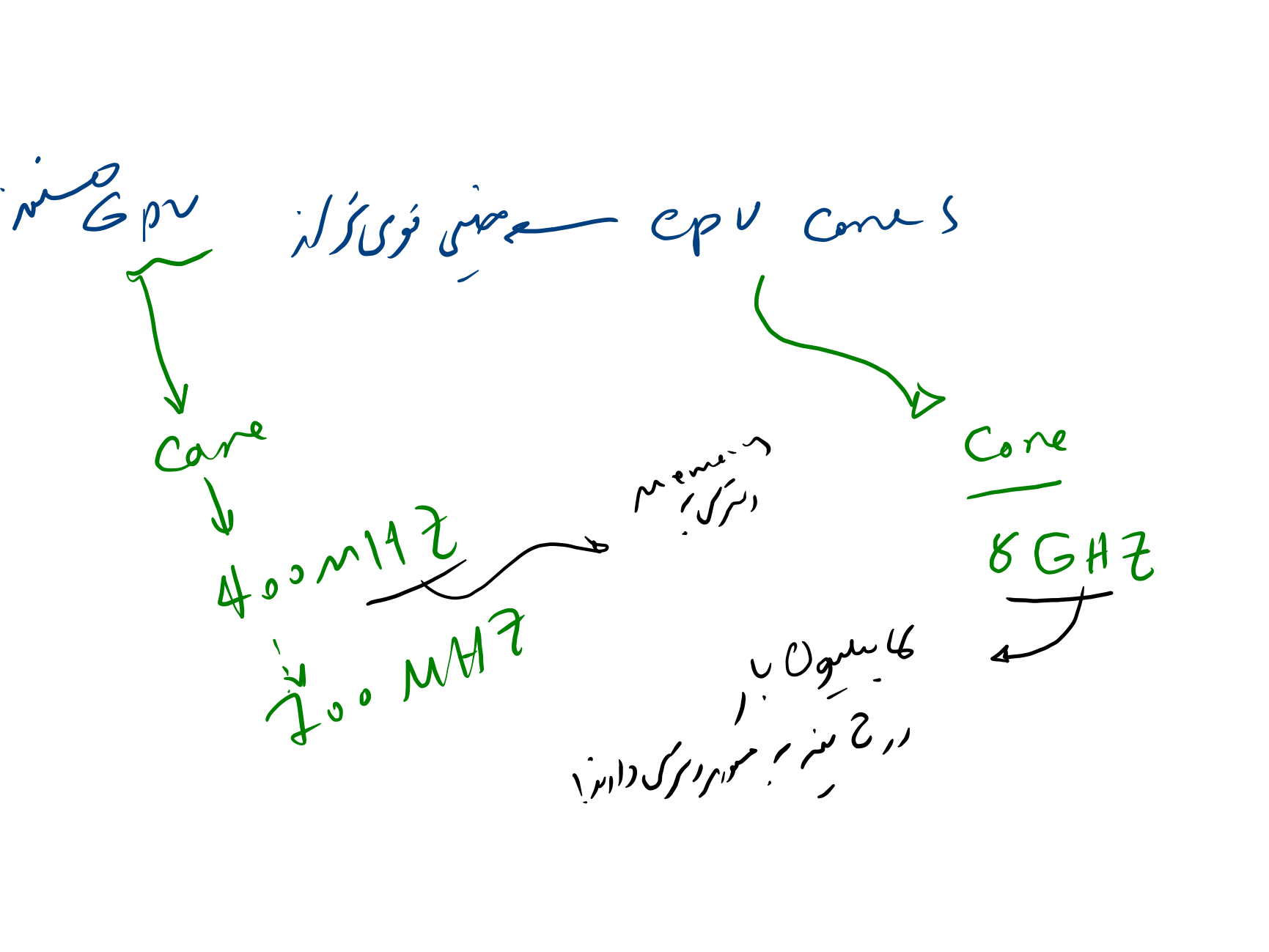
CPU سیستم اولویت کارش ← هیچ ترتیب مشخص

- word
- I/O
- Hand shake


a b c
d e f
g h i
a b c d e f g h i
+1.
+1.
out
out
out

حالا ما در GPU هم تعداد زیادی core داریم

پس چرا نمی‌آییم به جای CPU هم از GPU استفاده کنیم!؟


Gpu
cpu cores
Core
400MHz
700 MHz
Core
8GHz

≈ 5 clock cycle ← CPU ← عمل ضرب

≈ 100 clock cycle ← GPU

این عمل در GPU سریعتر صورت میگیرد!

compute

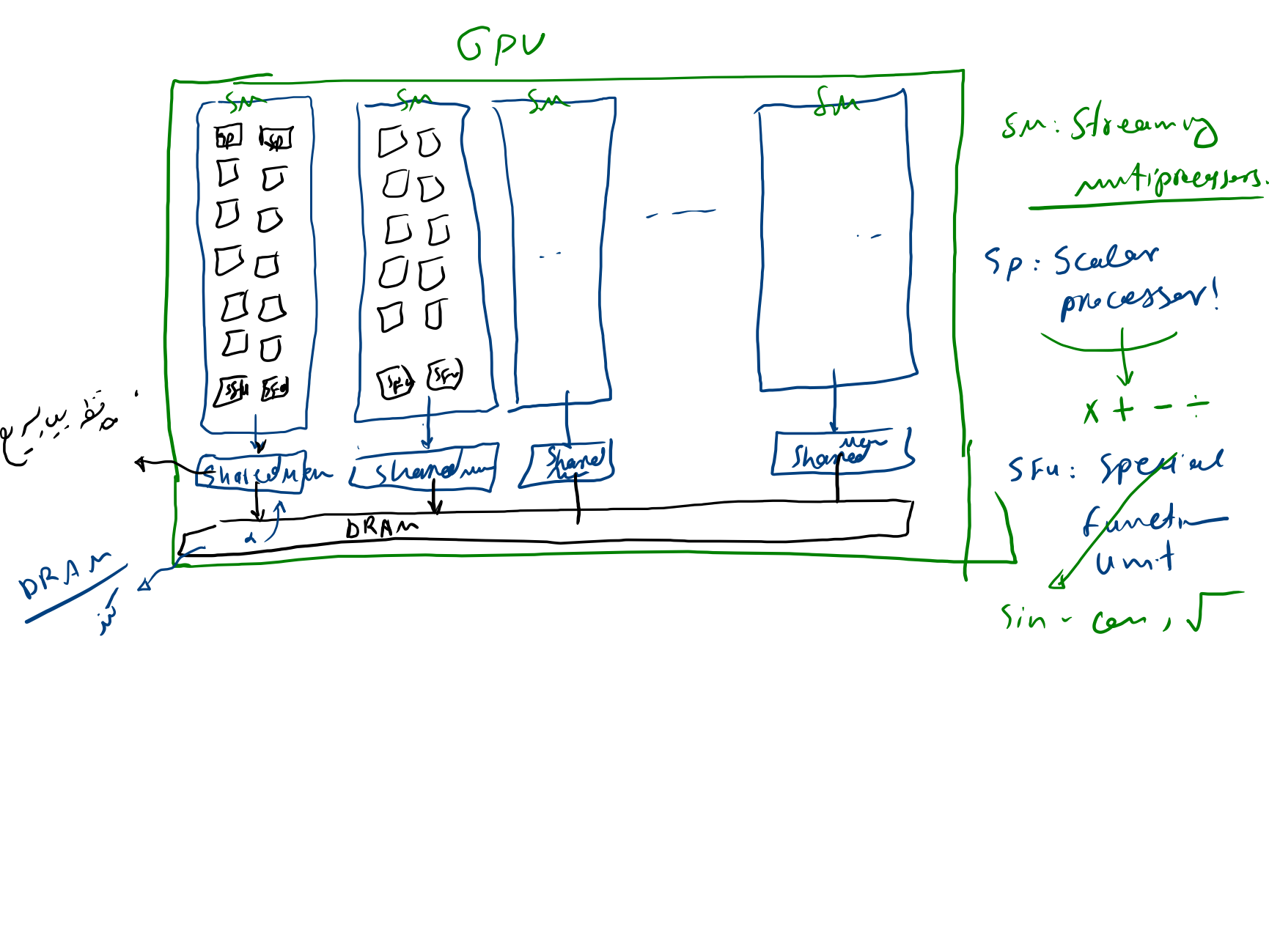
Heterogeneous.

CPU + GPU

high performance

ساختار GPU

بدون داشتن سخت‌افزار GPU ها می‌توانید کد برای [illegible] بنویسید!


GPU
SM
SP
SFU
Shared mem
DRAM
SM: Streaming multiprocessors.
SP: Scalar processor!
x + - ÷
SFU: Special function unit
Sin - Cos, √
DRAM

✓ shared mem vs cpu mem

هزینه از سمت cpu ← کارآمدتر

هزینه از سمت Shared mem ← کارآمدتر

---

cpu ⟶ 4 core

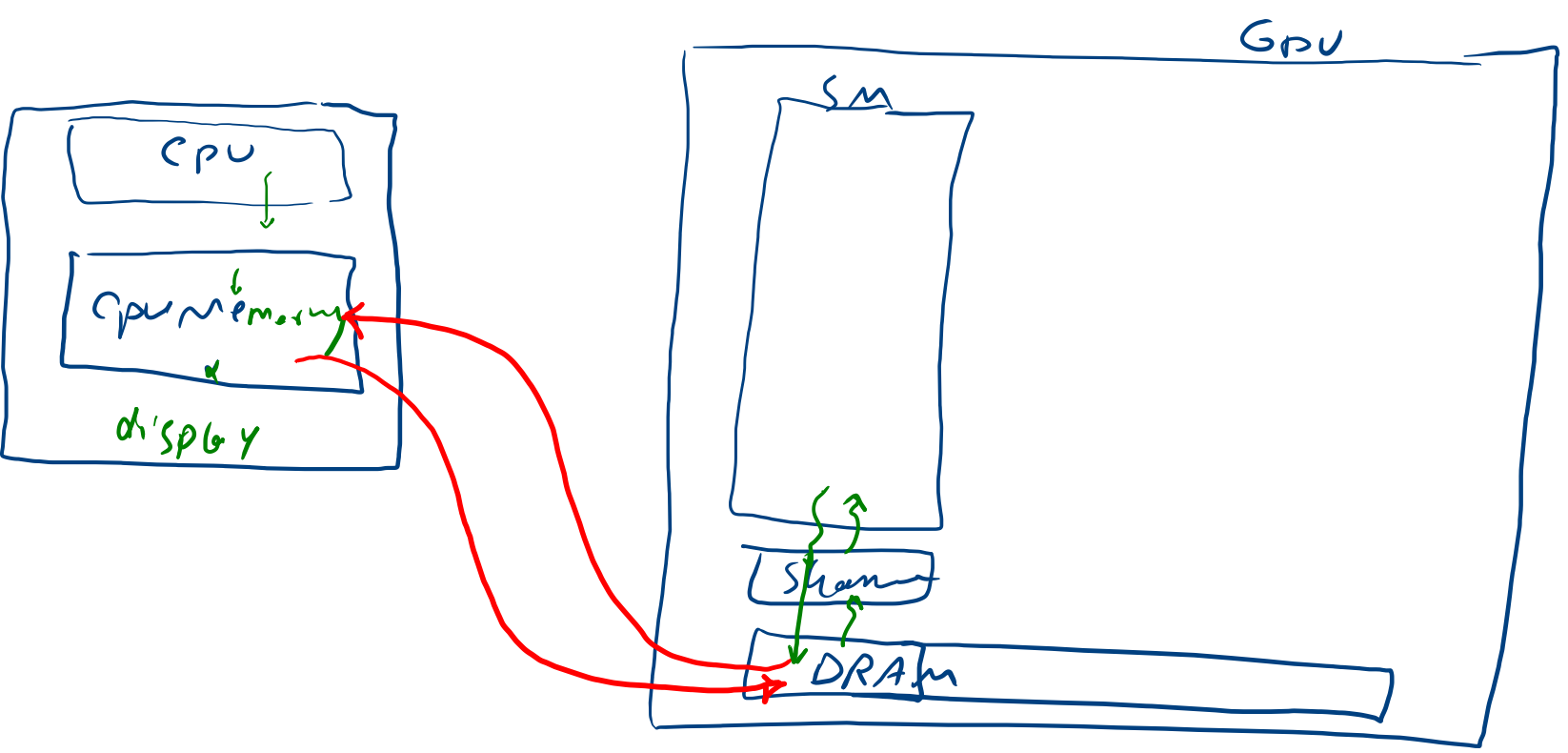
gpu ⟶ 4000 core!

Nvidia

Gforce RTX 2080 ti

4352 sp

64 sp per SM

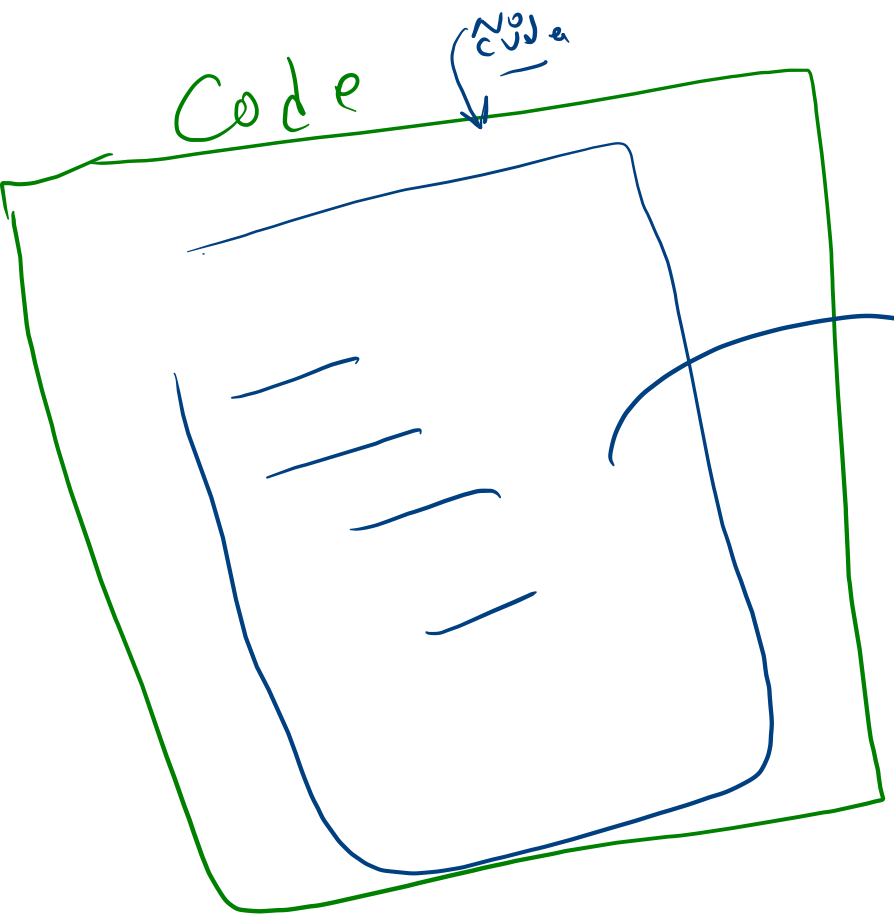

Gpu
SM
cpu
display
DRAM

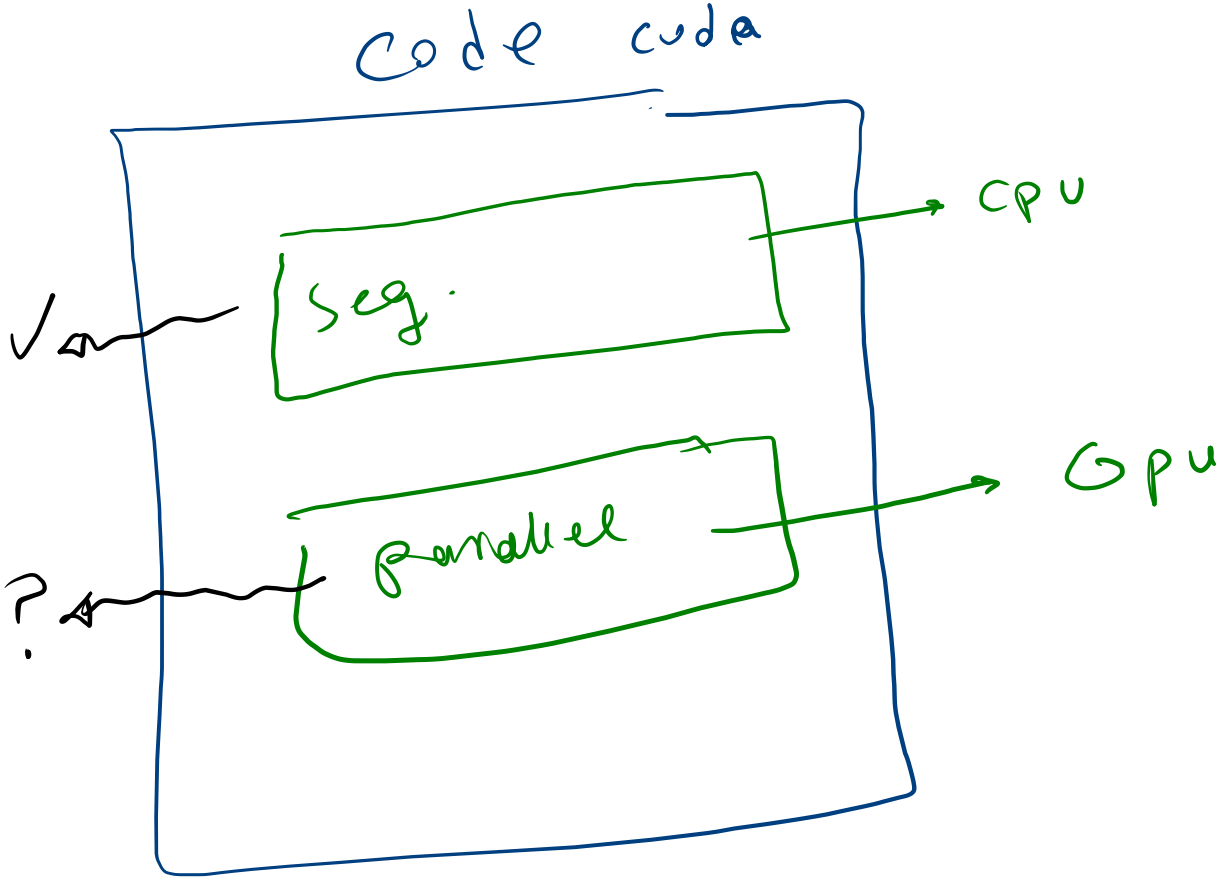
Cuda

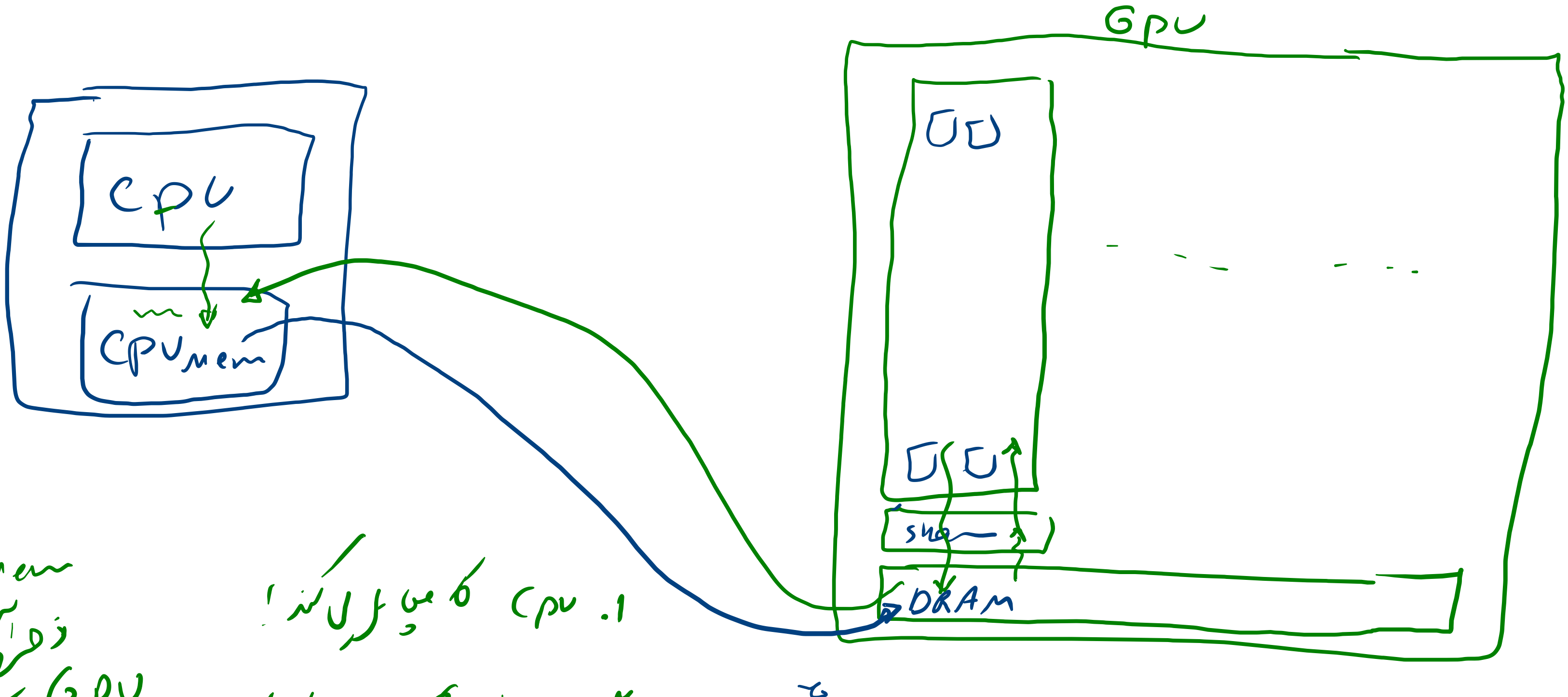
Compute unified device Arch.

CUDA ← یک افزونه یا متنی بر C/C++

در قالب کدهایی با C++ ← بعضی از الگوها را از CPU بردارند و اجرا شوند بر روی → GPU


Code
(default) accelerator
CPU


Code cuda
cpu
seq.
Gpu
parallel



cpumem از طریق درایور

۱. cpu کامپایل کند!

۲. درخواست حافظه را روی GPU

۳. داده ها از cpumem کپی شوند در DRAM

۴. GPU روی داده‌ها کار را انجام دهد

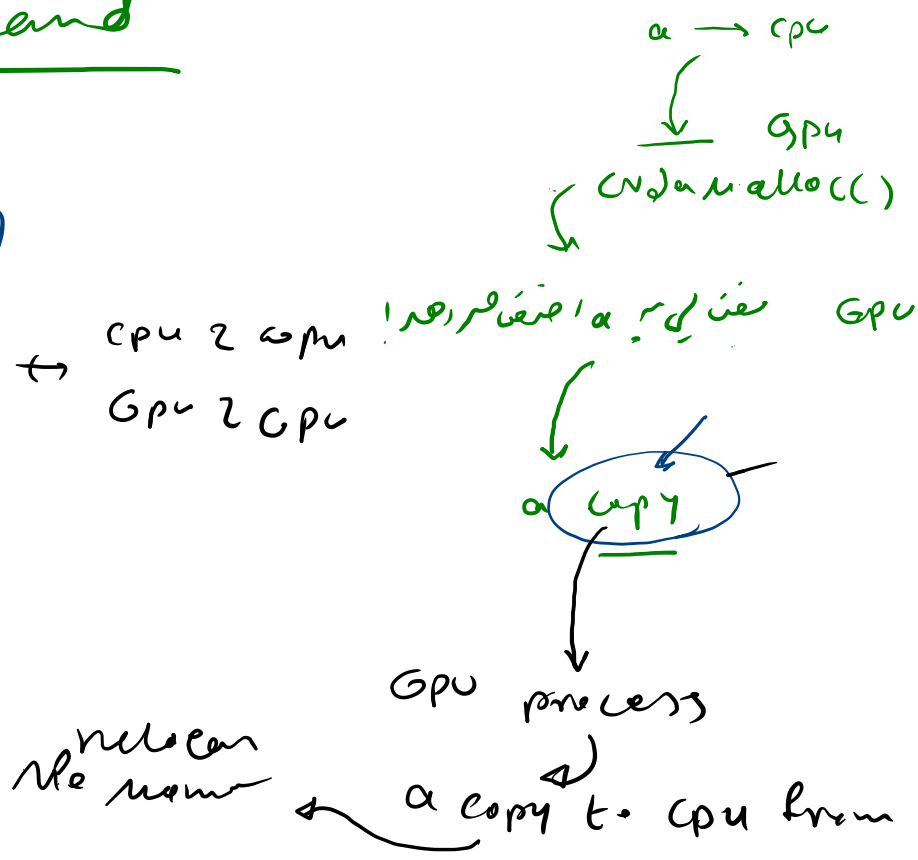
۵. کپی دوباره تغییرات از GPU در cpumem

۶. استفاده از result در cpu

cuda basic command

C++ / C

- cudaMalloc()
- cudaMemcpy() ↔ CPU 2 GPU / GPU 2 CPU
- cudaFree()



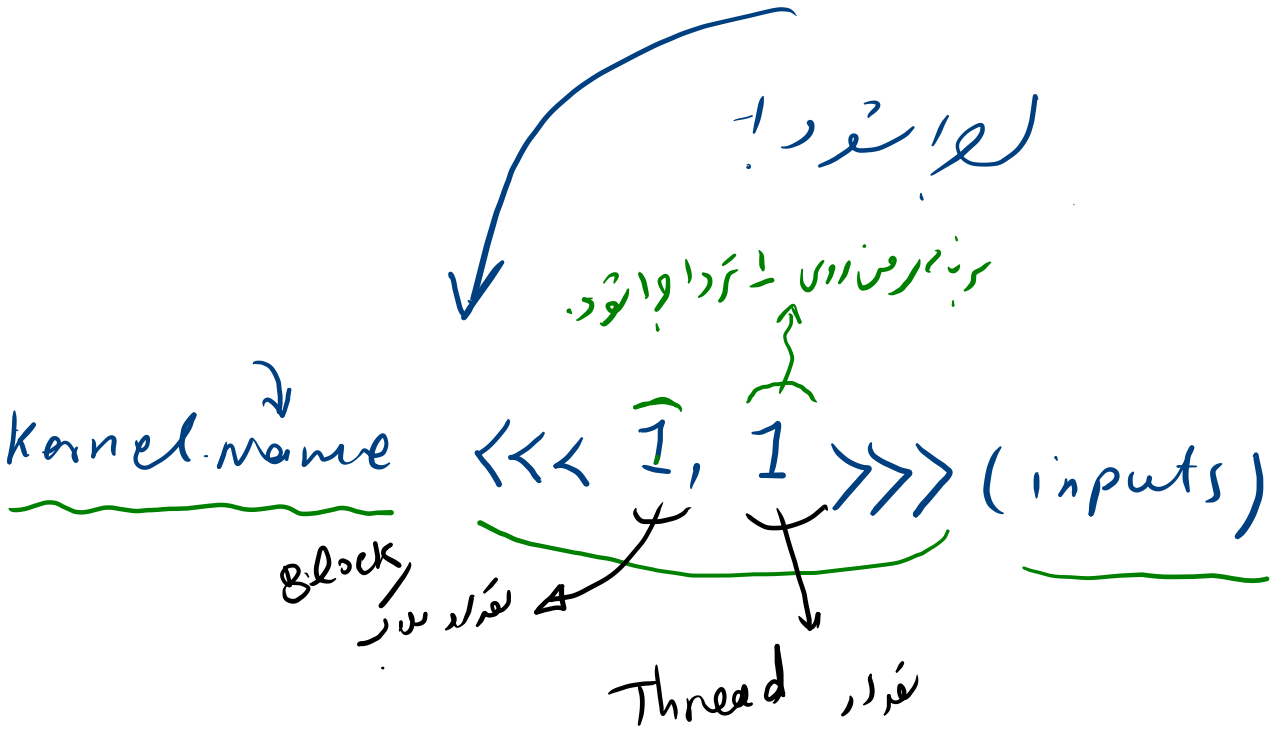
چگونه یک برنامه برای GPU نوشته می‌شود؟

```
__global__
```

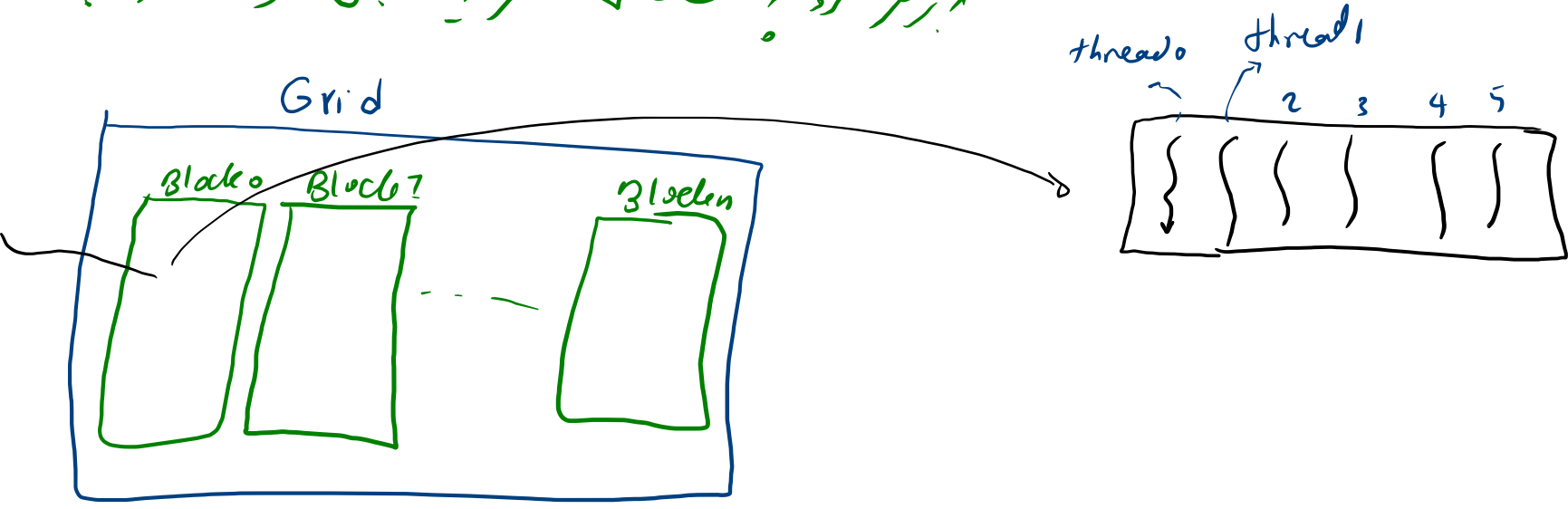
```
__global__ void kernel_name ( inputs )
{
}
```

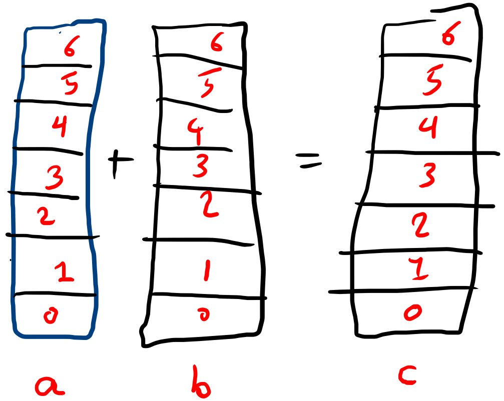
دستورهایی که هر رشته باید اجرا کند

* تفاوت کرنل را از cpu صدا زدن تابع روی cpu



هر بلوک کدام ترد می‌باید تعیین کنیم!





$a[0] + b[0] = c[0] \rightsquigarrow T0$

$a[1] + b[1] = c[1] \rightsquigarrow T1$

$a[2] + b[2] = c[2] \rightsquigarrow T2$

$a[3] + b[3] = c[3] \rightsquigarrow T3$

$\vdots$

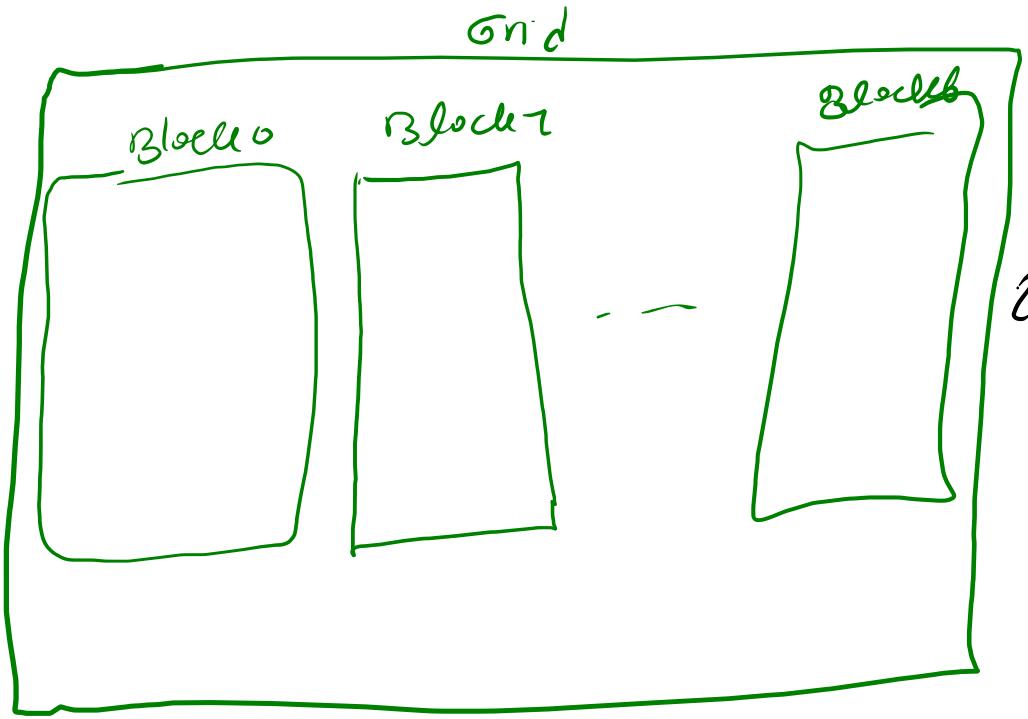
$a[6] + b[6] = c[6] \rightsquigarrow T6$

تعداد پردازه‌های [illegible] نیاز
را در هر دوره همروندی اش محاسبه

7 ترد

7

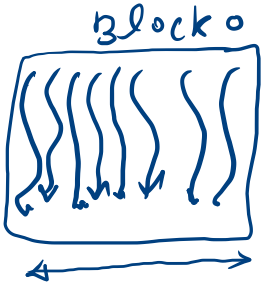
چون حوصله 7 تا عمل جمع دارم!



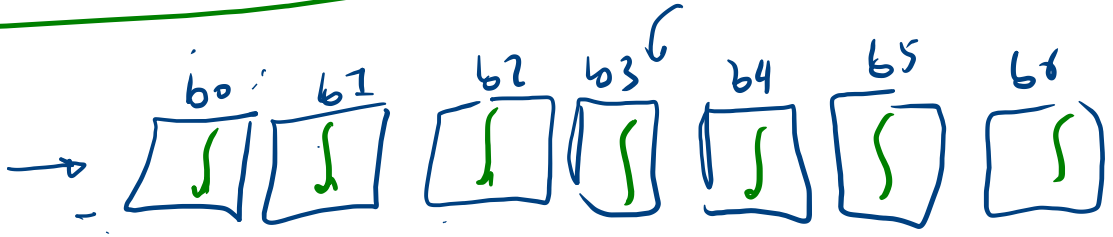
Kernel Nam <<< 1, 7 >>>( )

یک بلوک ایجاد کن!

7 تا ترد را از آن بلوک استفاده کن



kernel Nam <<< 7, 1 >>>()



kN <<< 1, 7 >>> vs kN <<< 7, 1 >>>

↓ speed ↑

Shared Memory!

Block

تعداد Thread در هر بلاک را با توجه به حافظه کنترل کنید

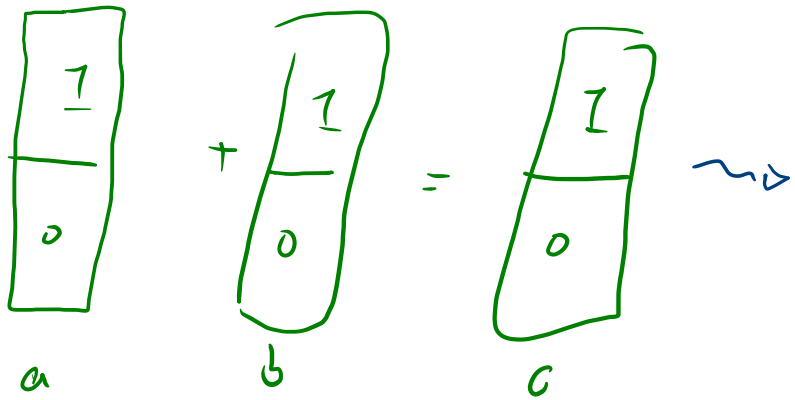
1024

512

6

9

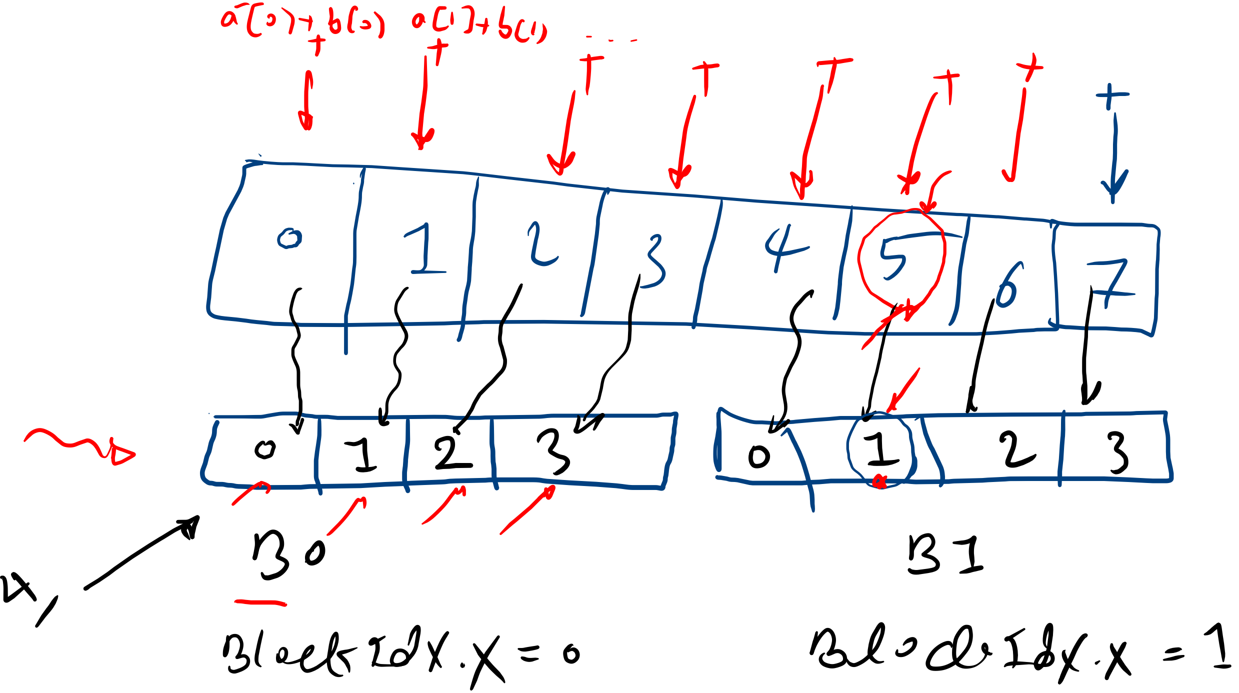
(M)



a[·] + b[·] = c[·] → To

- Thread Idx.x
- Block Idx.x

→ Grid Idx.x

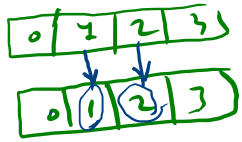


سوال: مر

T : 1
b : 1
← کدام ایندکس را انجام میدهد؟ :

$$\text{index} = \text{ThreadIdx.x} + \text{blockIdx.x} * M = 1 + (1)4 = 5$$

مثال کرنل



مزیت آرایه دوبعدی (در آرایه یک بعدی)

```
__global__ void multiply (float* result, float *a, float *b)
{
    int i = threadIdx.x
    result[i] = a[i] * b[i]
}
```

```
int i = blockIdx.x * blockDim.x + threadIdx.x;
```

M

pycuda

size-byte

cuda.mem_alloc (↑) ← برای تخصیص حافظه GPU

cuda.memcpy_htod (gpu_mem, cpu_mem) ← برای انتقال داده‌ها

kernel launch → source model (kernelname) ↓

mod.get_function (↑) ← kernelname

kernelname (inputs, block=( ))