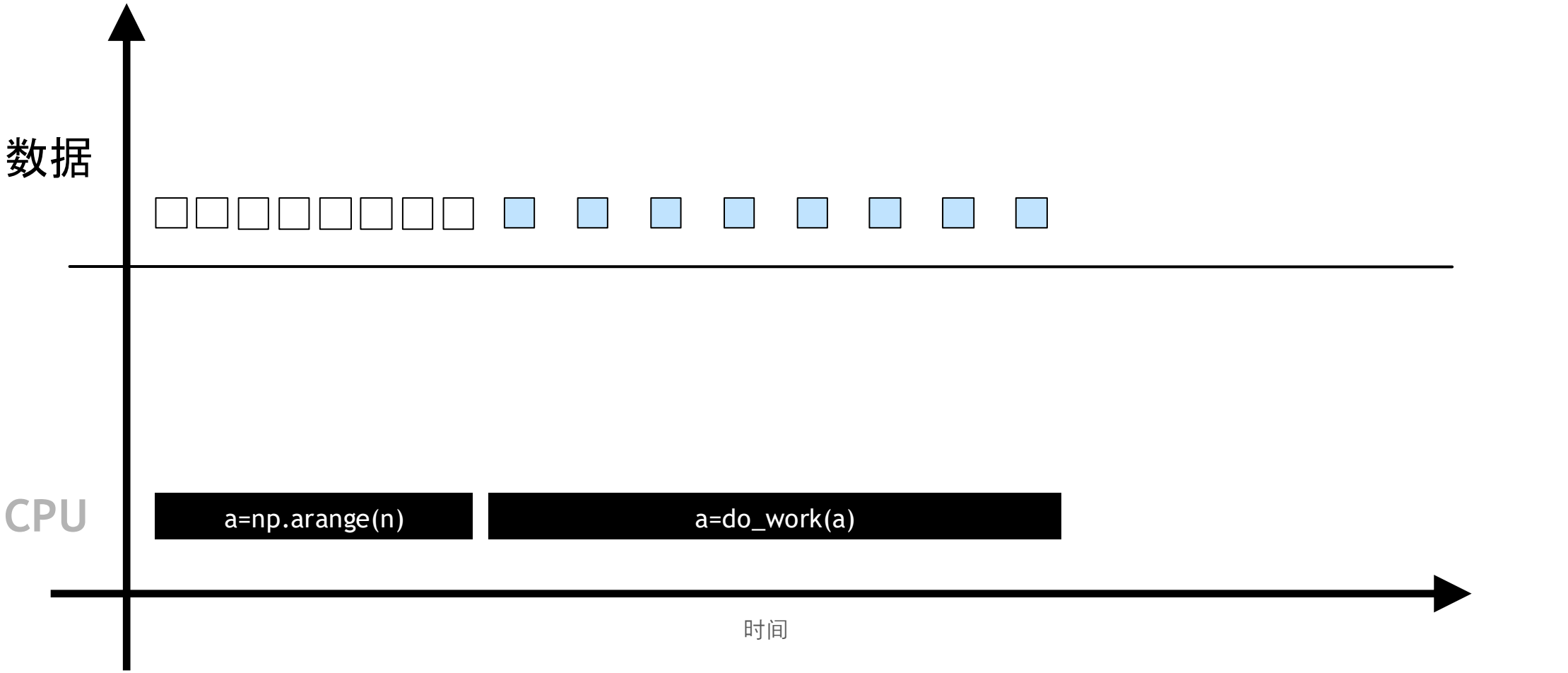
NVIDIA | DEEP LEARNING INSTITUTE

加速计算基础 —— CUDA Python

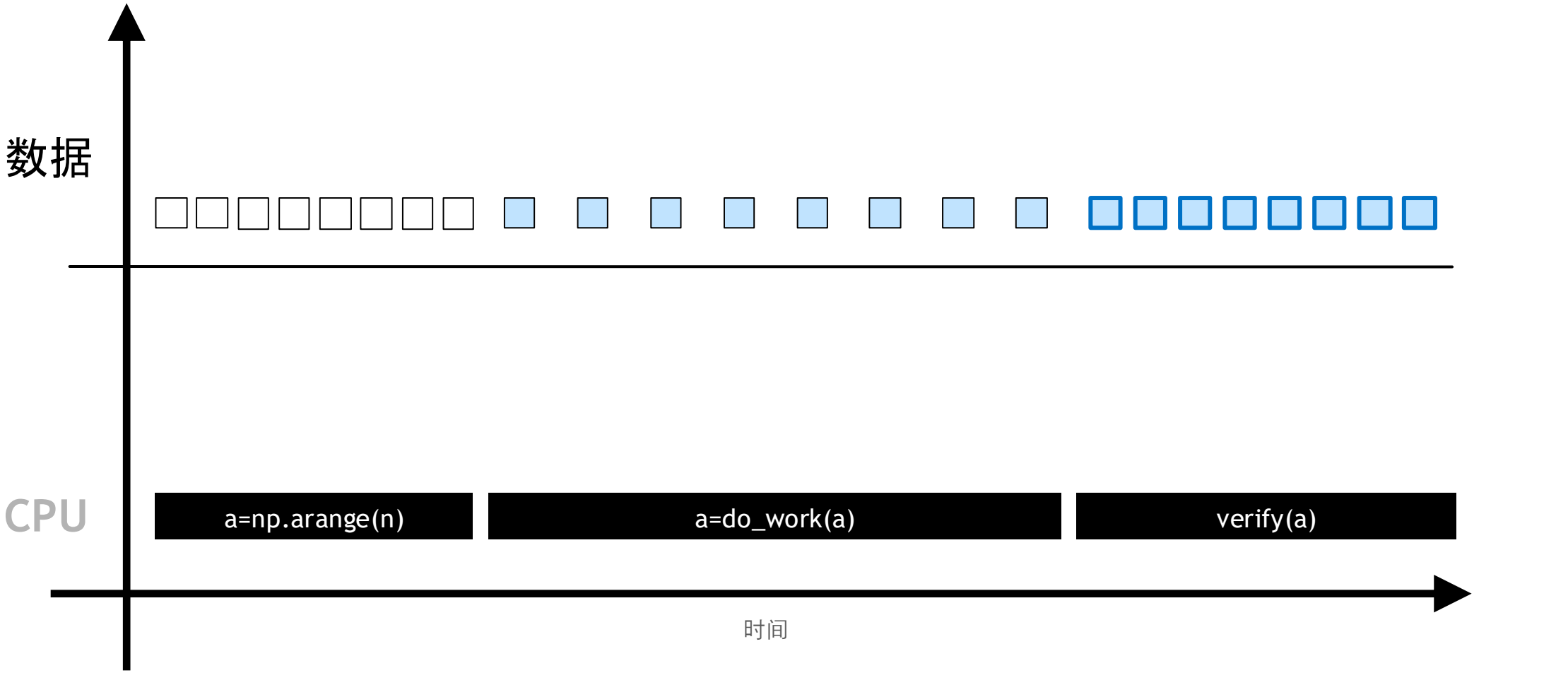
GPU 加速应用程序与 CPU 应用程序对比

在 CPU 应用程序中，
数据在 CPU 上进行分配

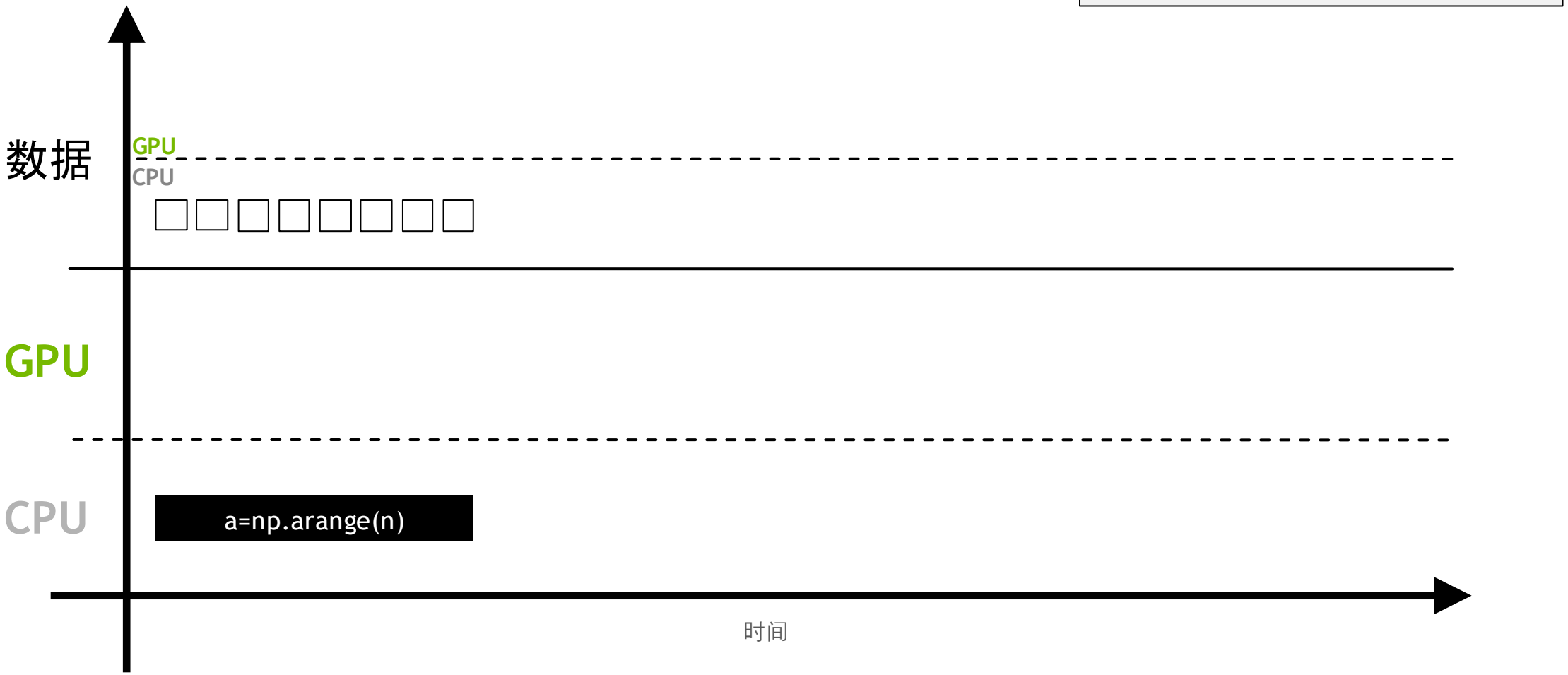
...并且所有工作均在 CPU 上串行执行



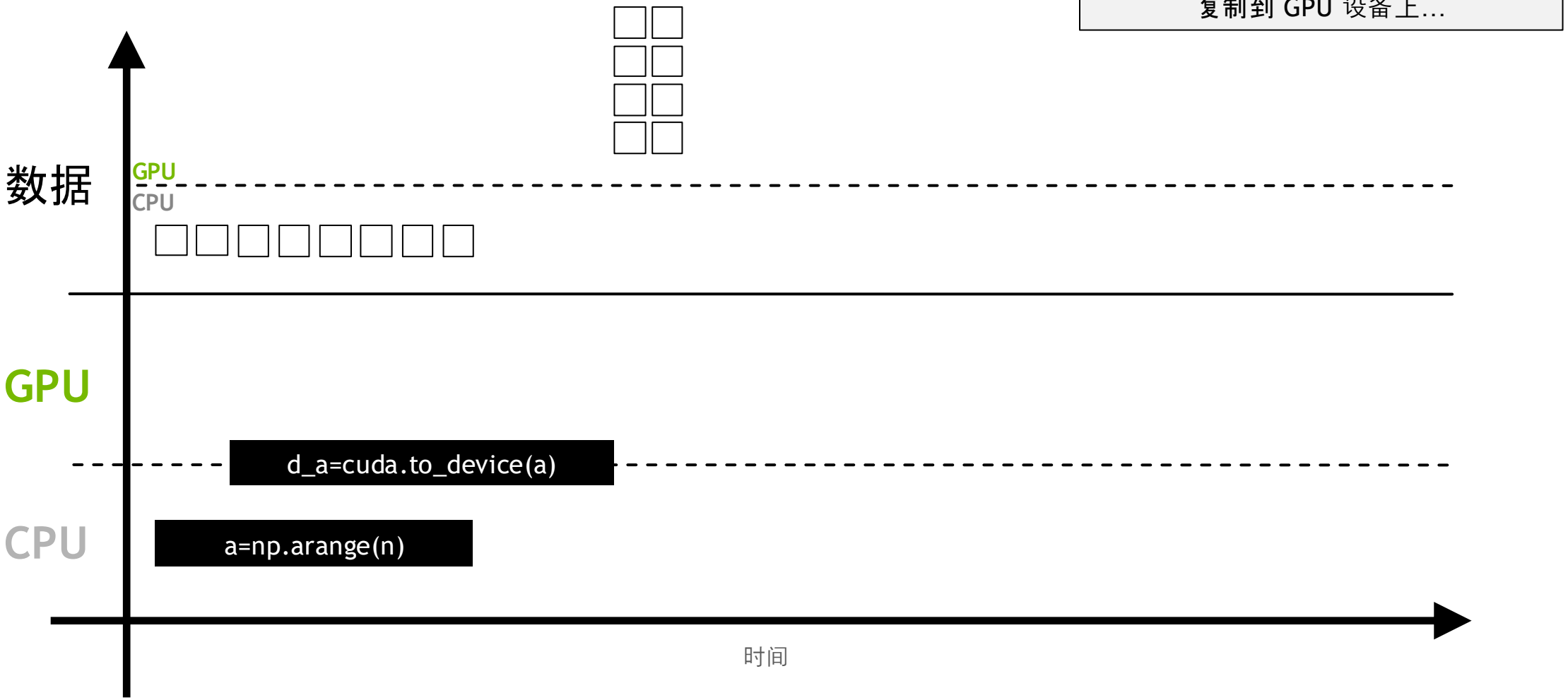
...并且所有工作均在 CPU 上串行执行



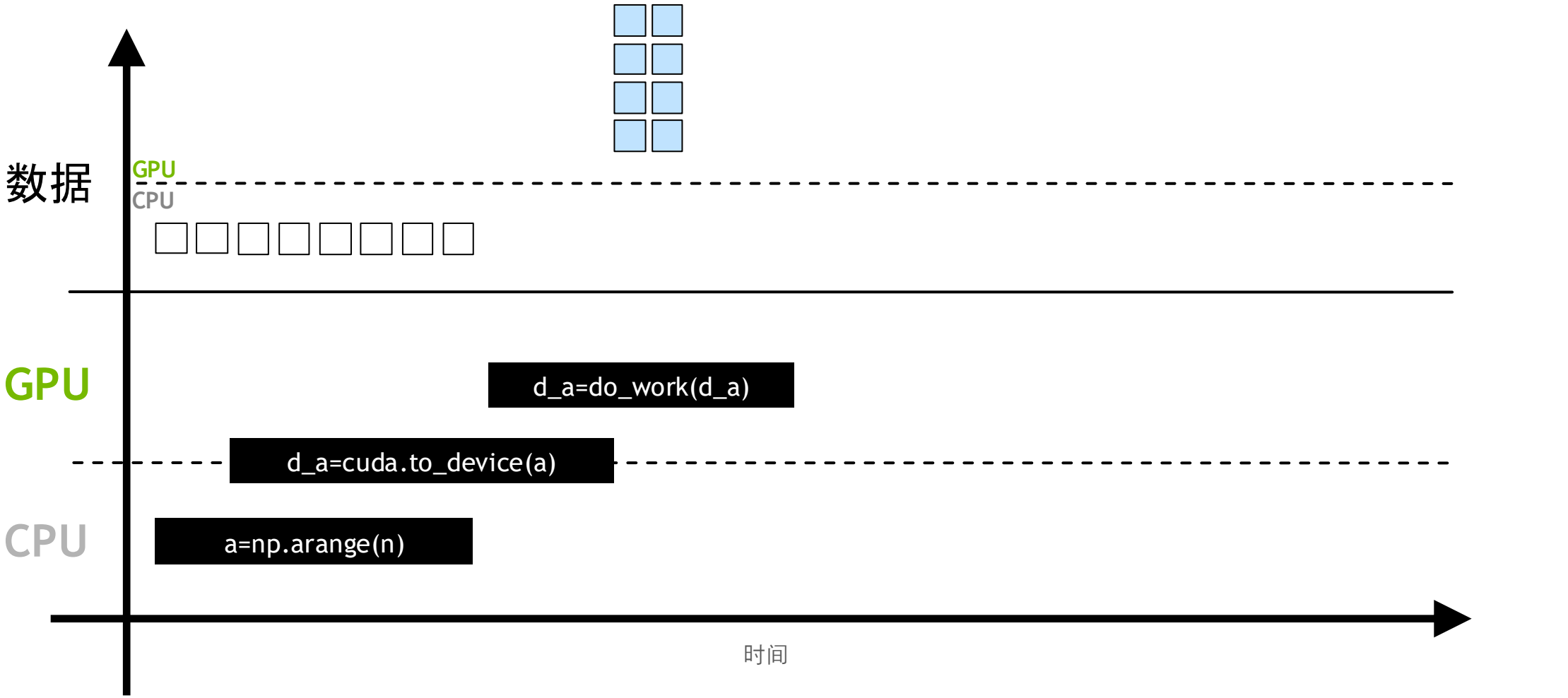
在加速应用程序中，
既有主机内存，也有设备内存。



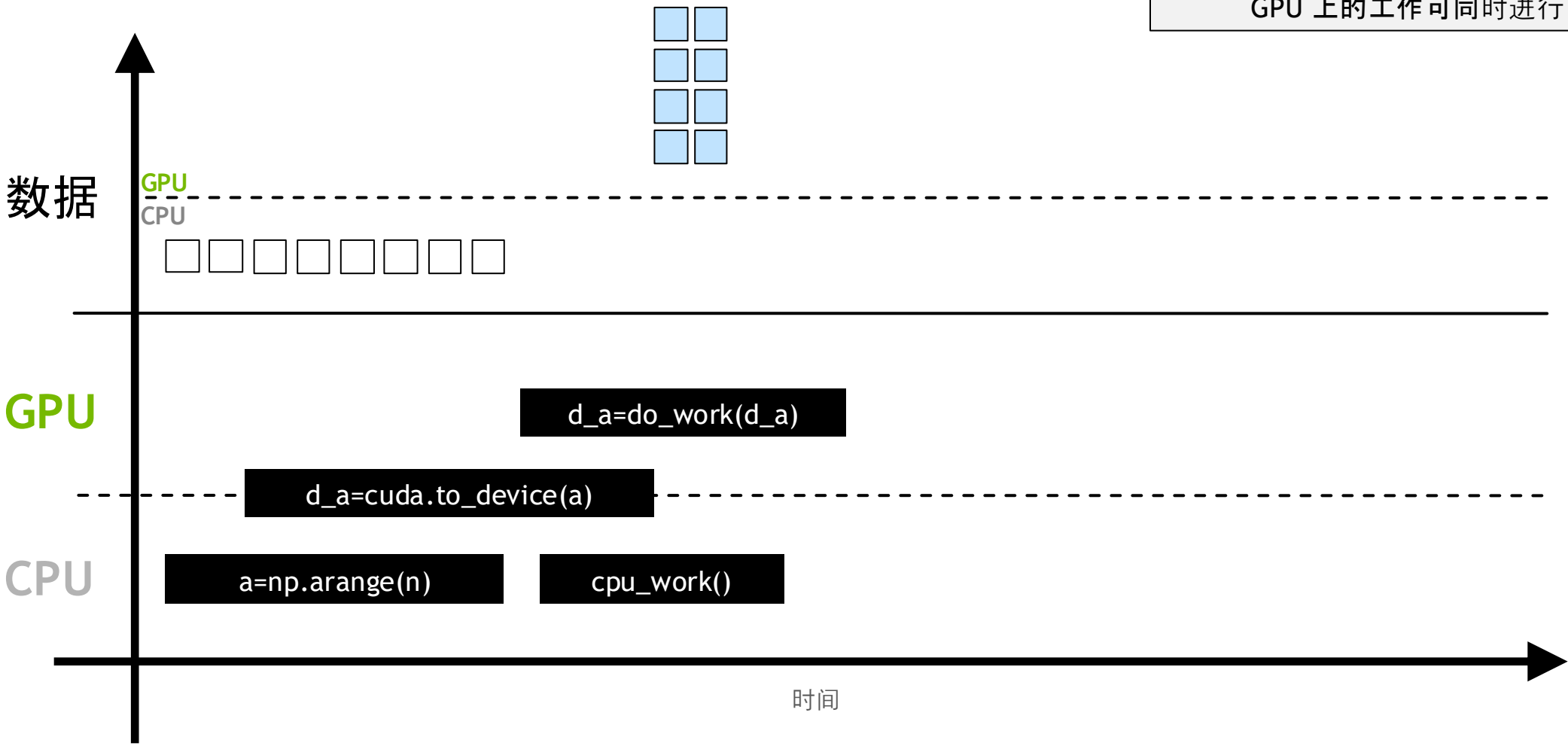
CPU 上初始化的数据可以
复制到 GPU 设备上...



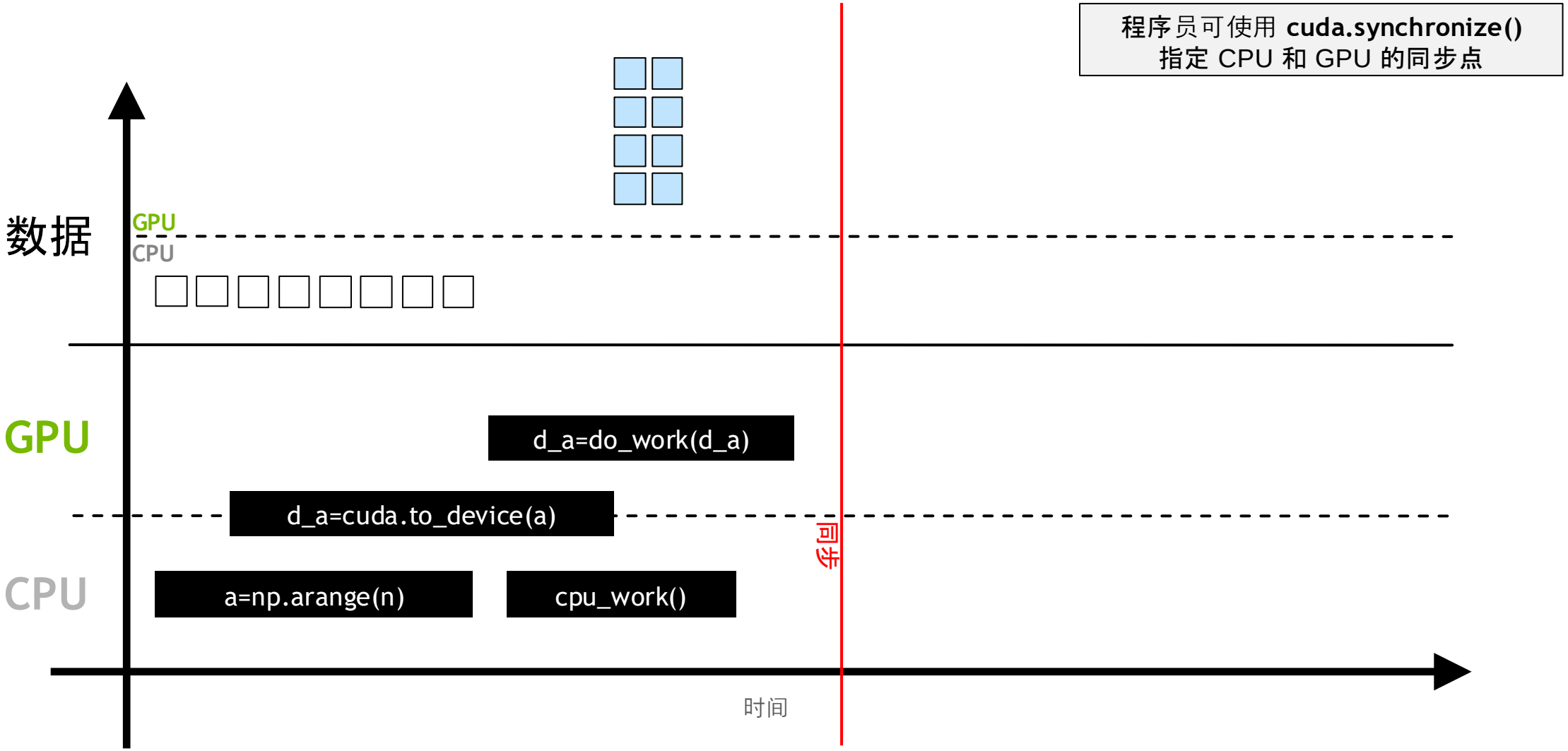
...这里数据就能被并行处理



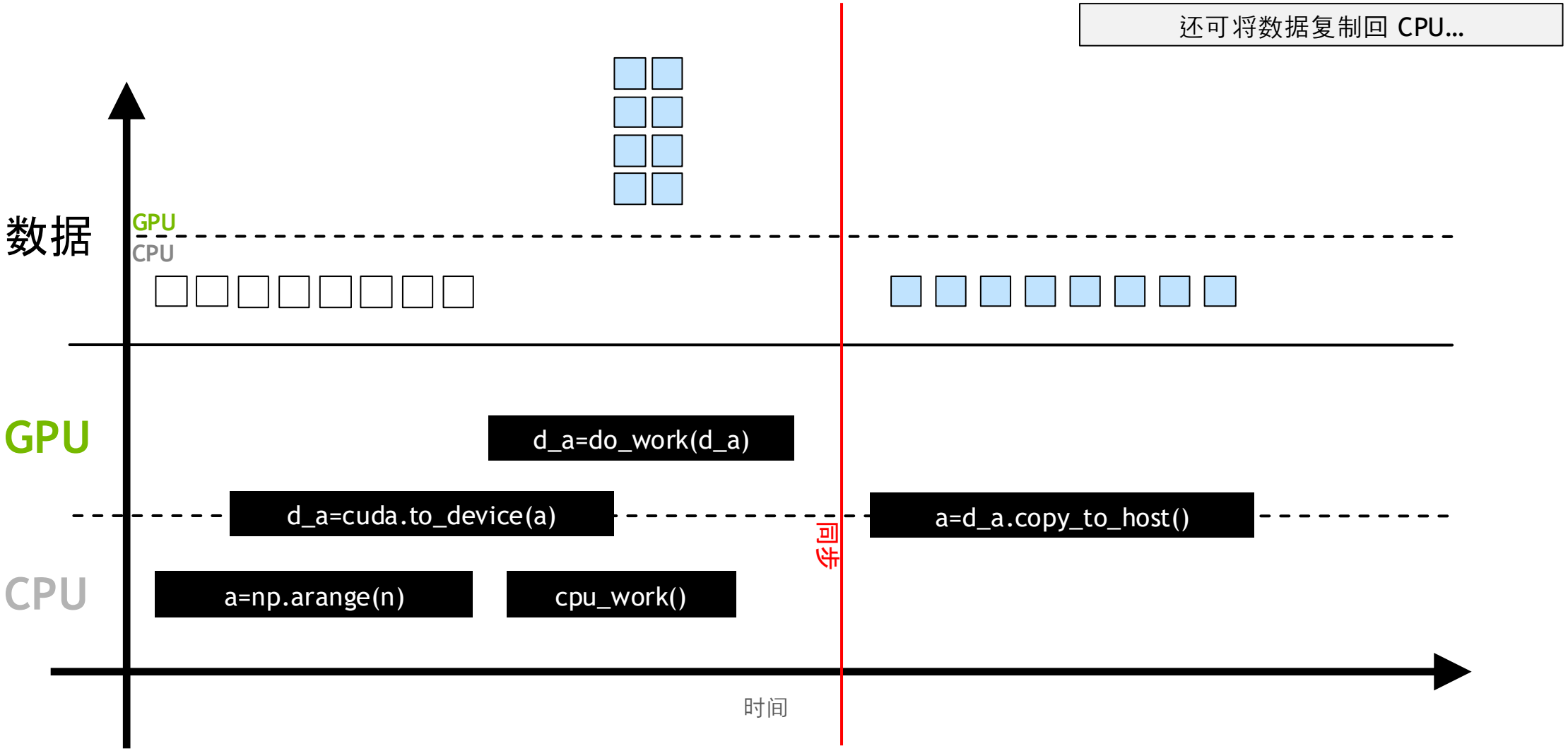
GPU 工作与主机异步，所以 CPU 和 GPU 上的工作可同时进行



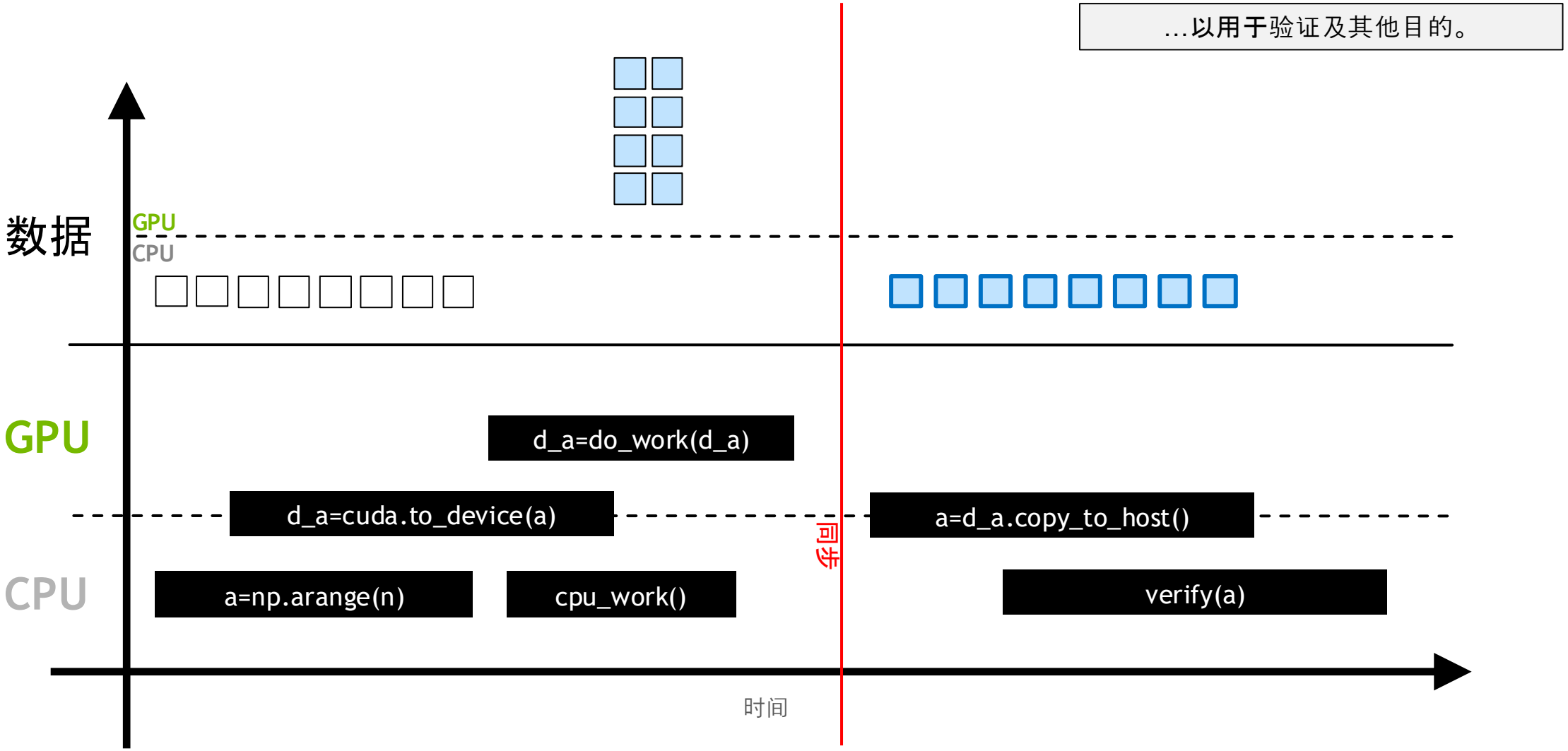
程序员可使用 **cuda.synchronize()**
指定 CPU 和 GPU 的同步点



还可将数据复制回 CPU...

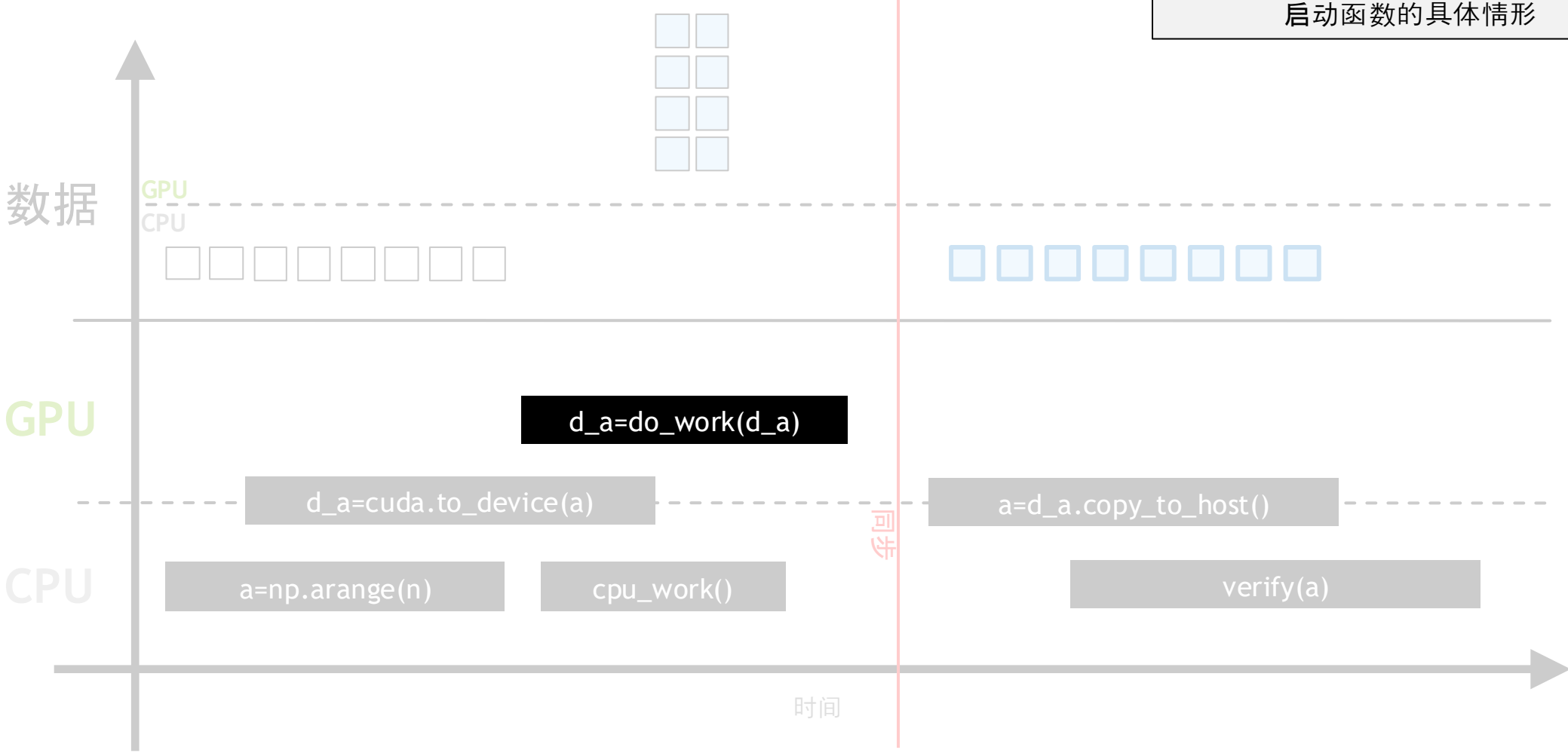


...以用于验证及其他目的。



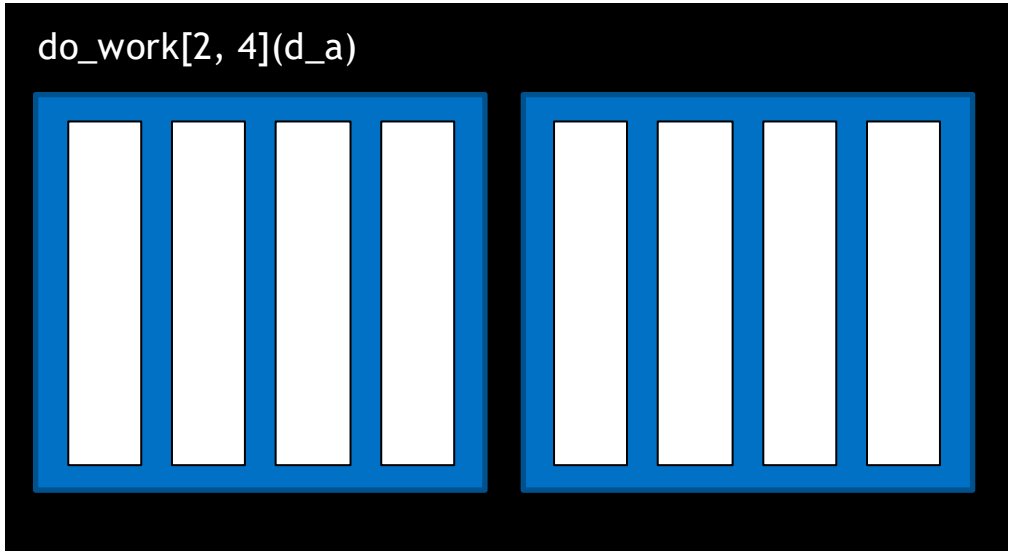
CUDA 线程层次结构

我们来深入探讨一下在 GPU 上
启动函数的具体情形

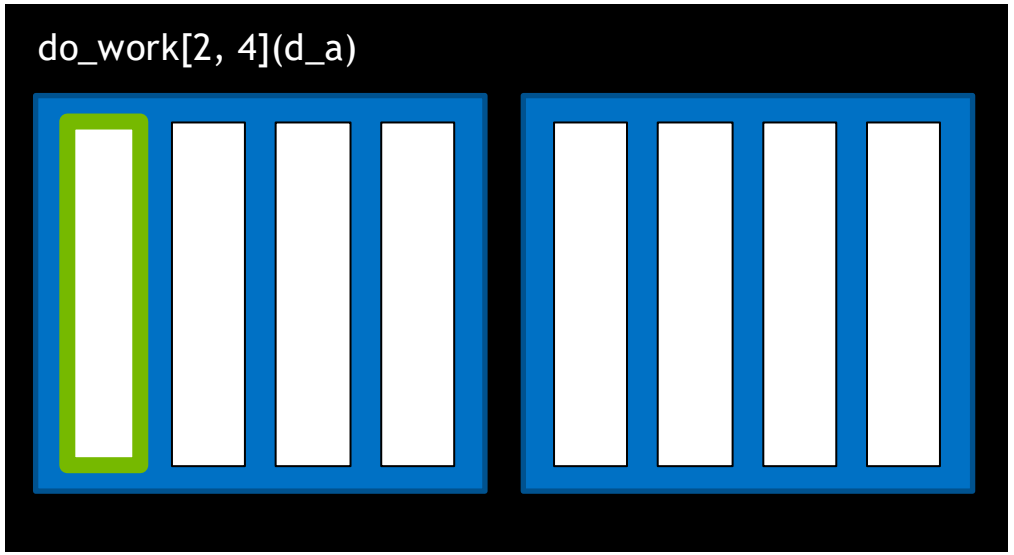


GPU 可并行执行工作

GPU

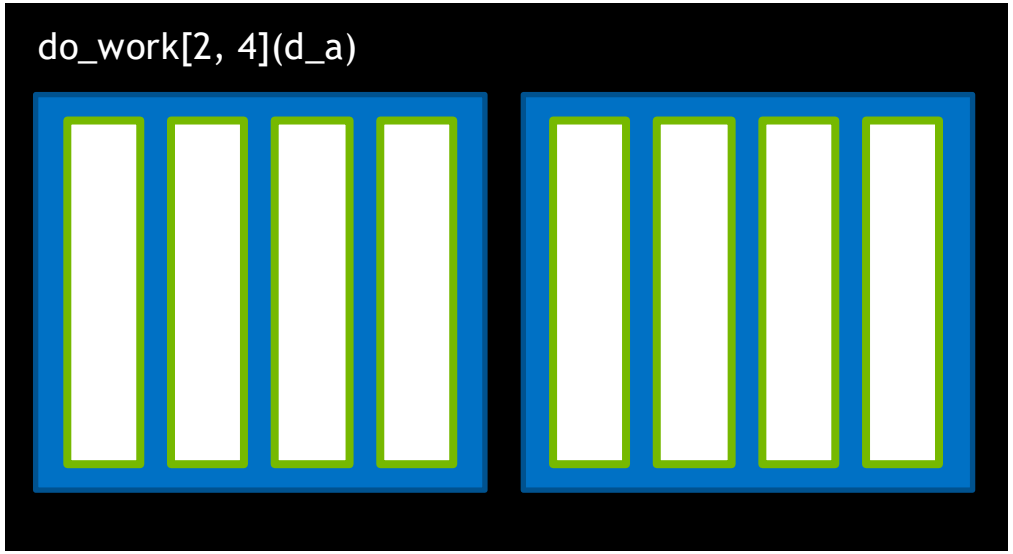


GPU



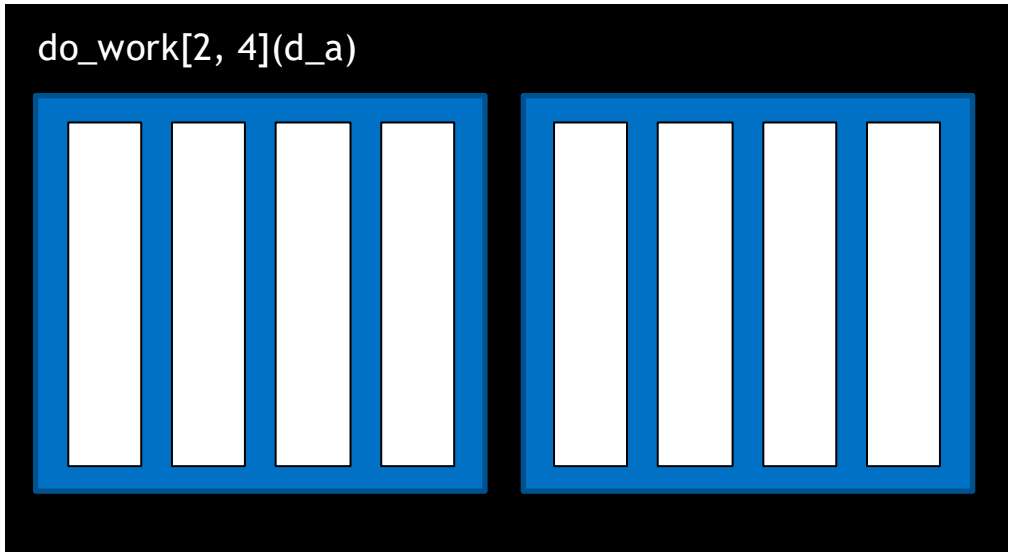
多个线程并行运行

GPU



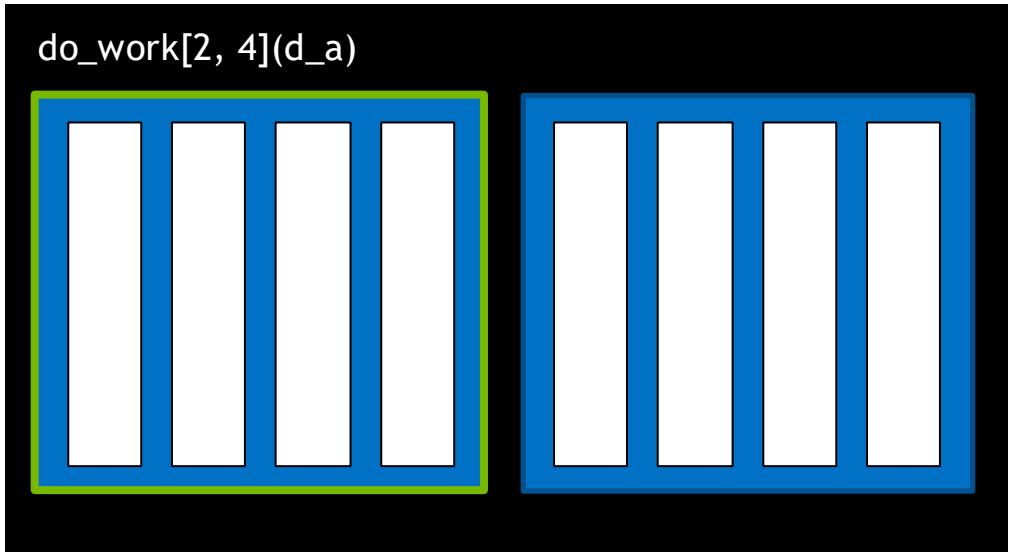
CUDA 可并行处理数千个线程。为简单起见，我们已大幅缩减图片尺寸。

GPU



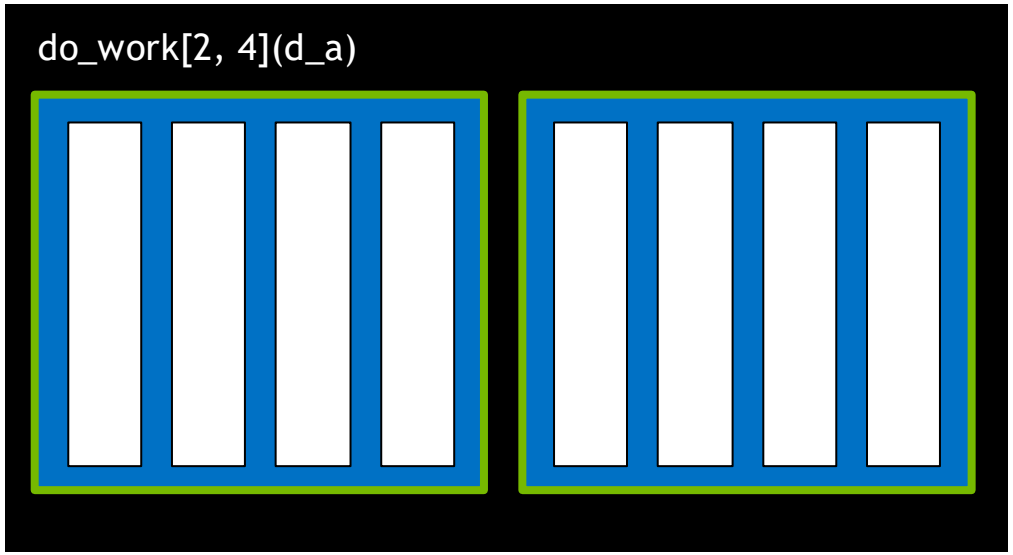
线程的集合称为块

GPU



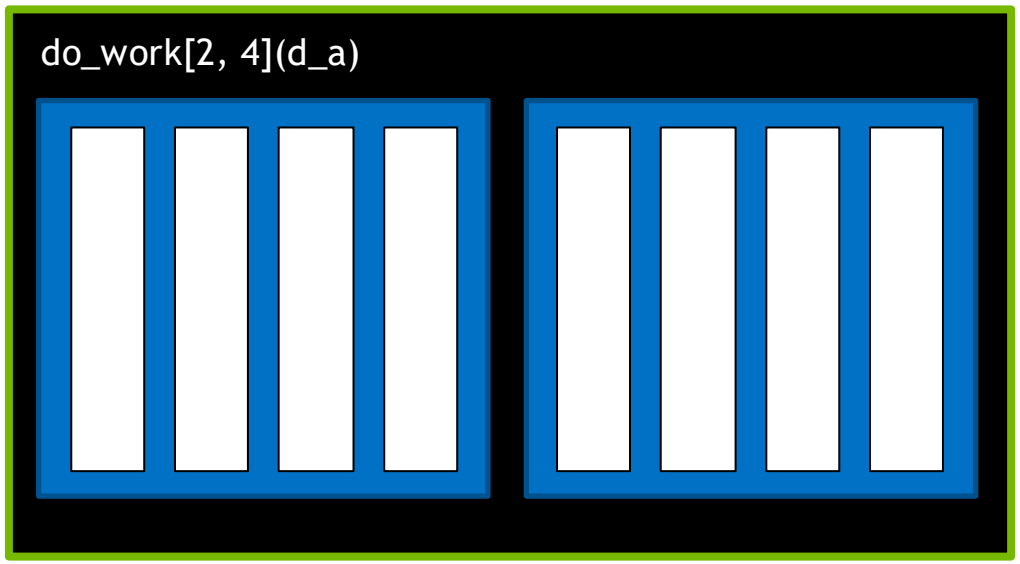
可以有很多个线程块

GPU



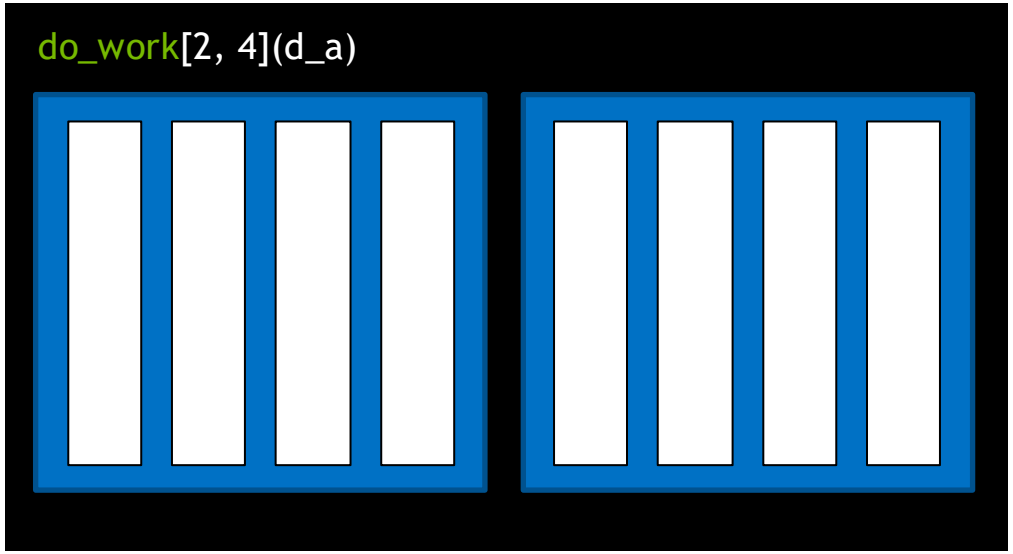
与给定核函数启动相关联的块的
集合称为网格

GPU



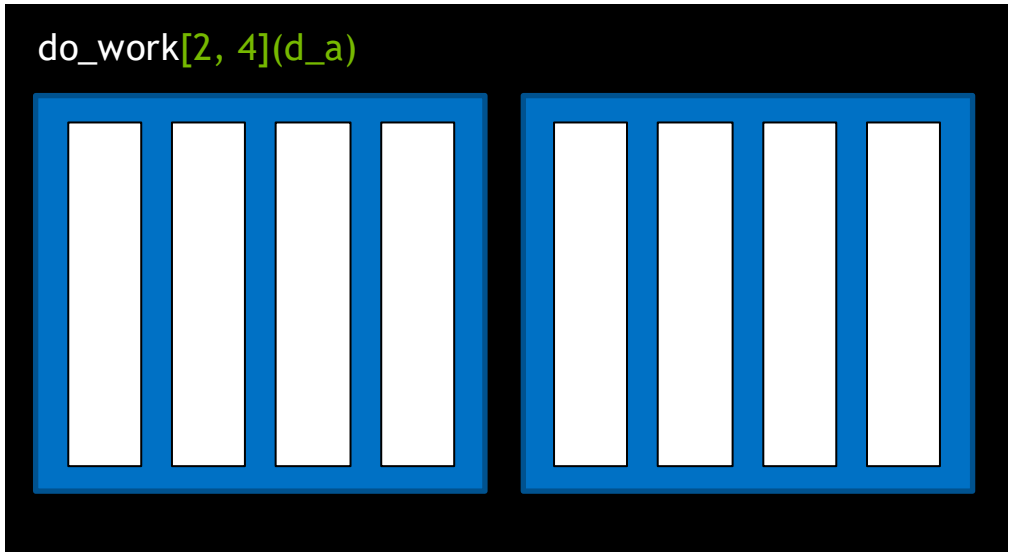
GPU 函数称为核函数

GPU



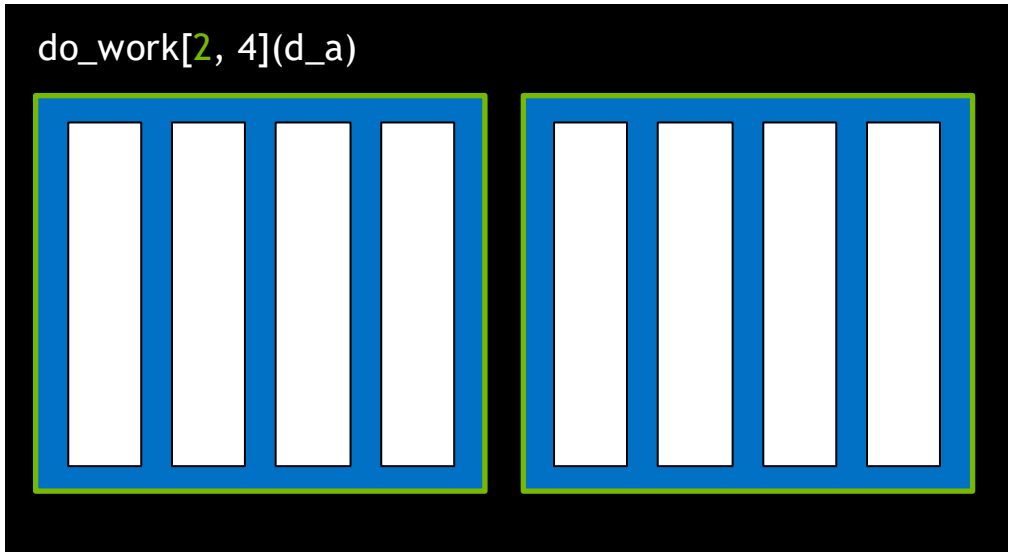
核函数通过执行配置启动

GPU



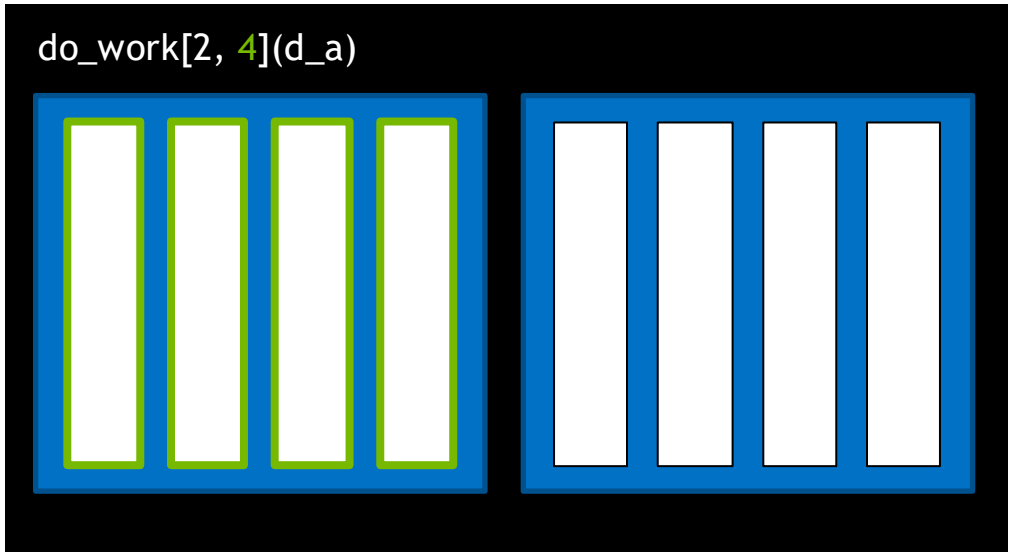
执行配置定义了网格中的块的数量

GPU



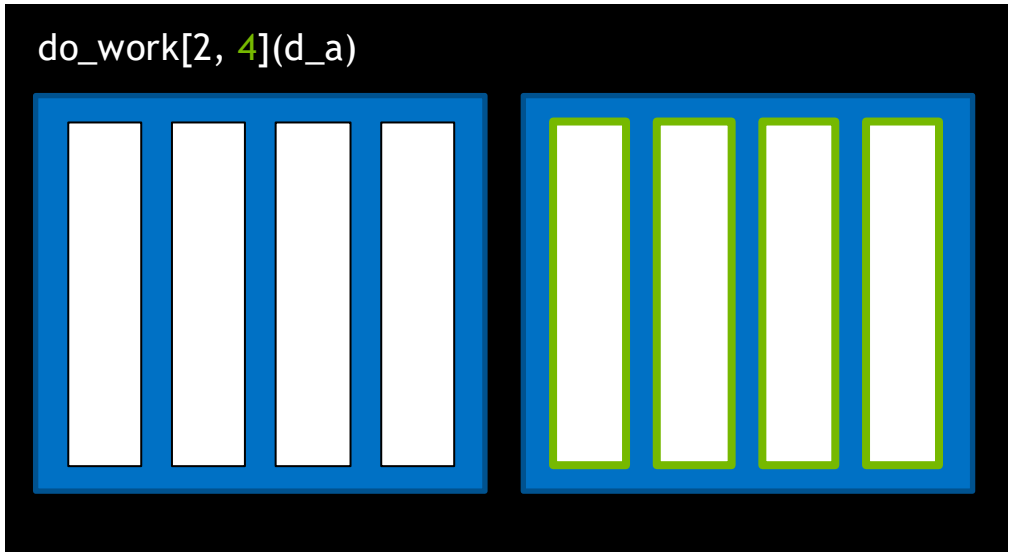
...以及每个块中的线程的数量

GPU



网格中的每个块均包含相同数量的线程

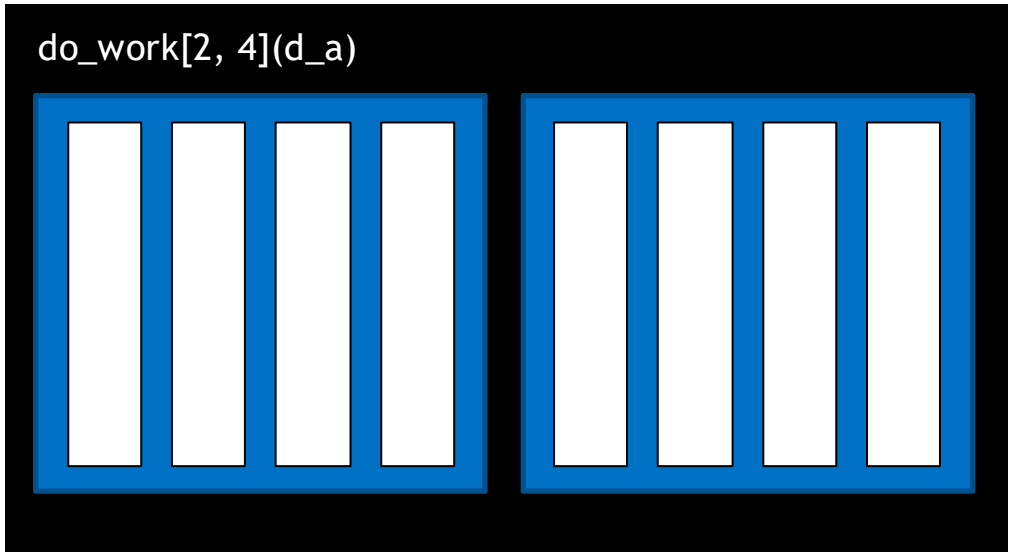
GPU



CUDA 提供的线程层次结构变量

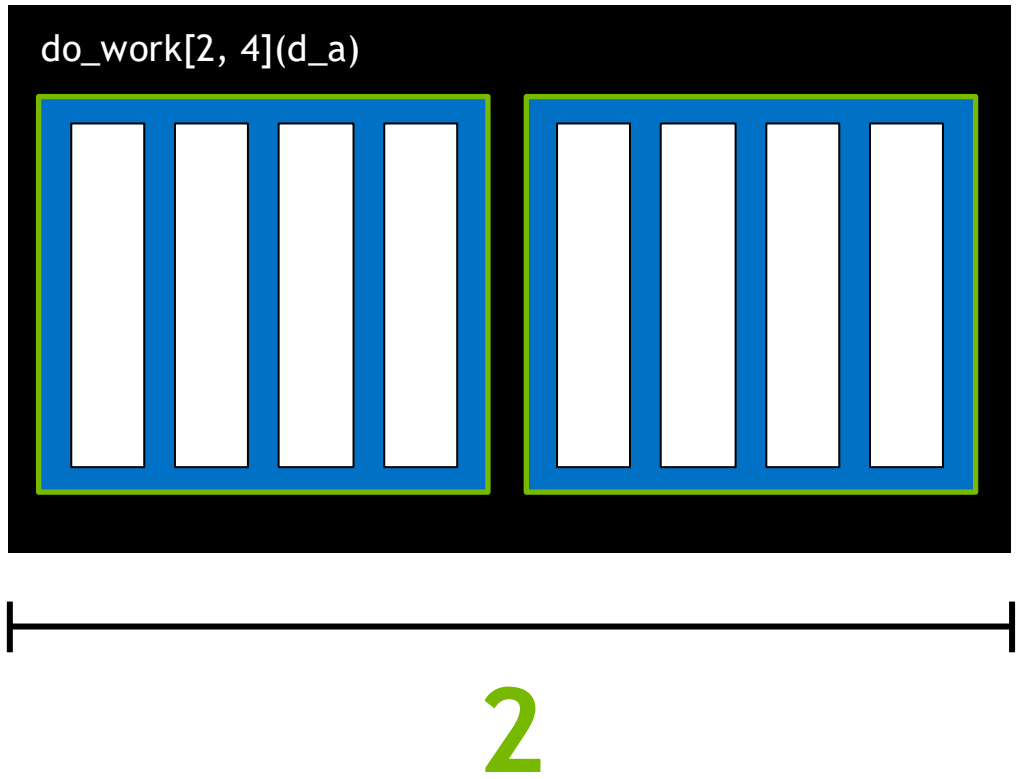
在核函数定义中，CUDA 提供了描述它所执行的线程、块和网格的变量

GPU



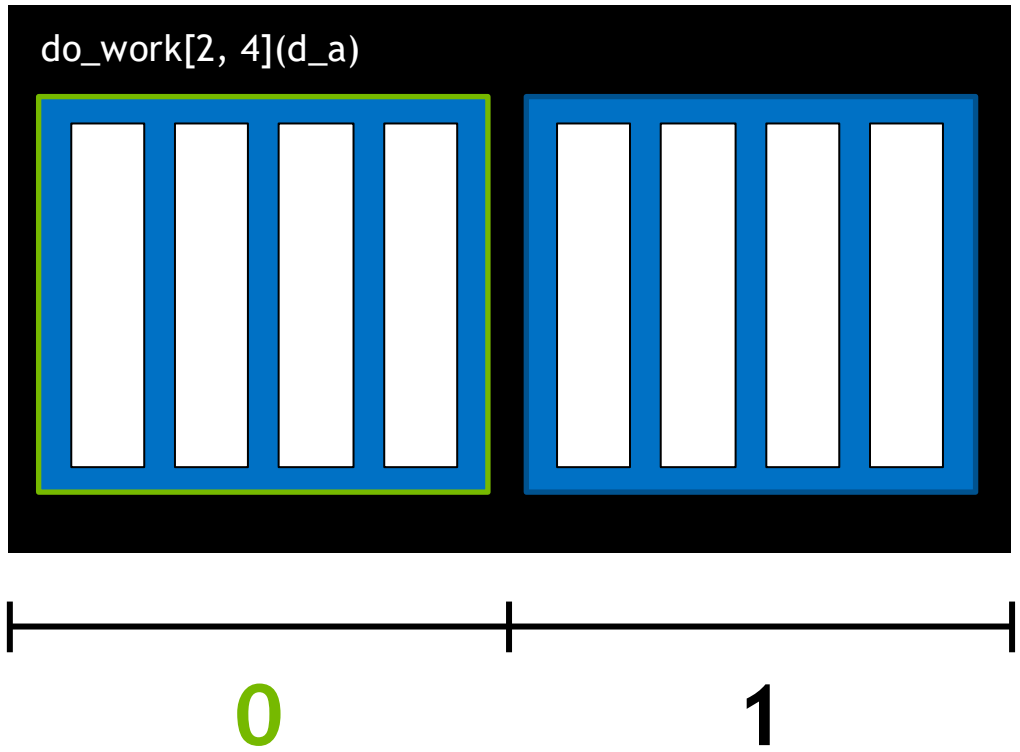
gridDim.x 是网格中的块数，
在本例中为 **2**

GPU



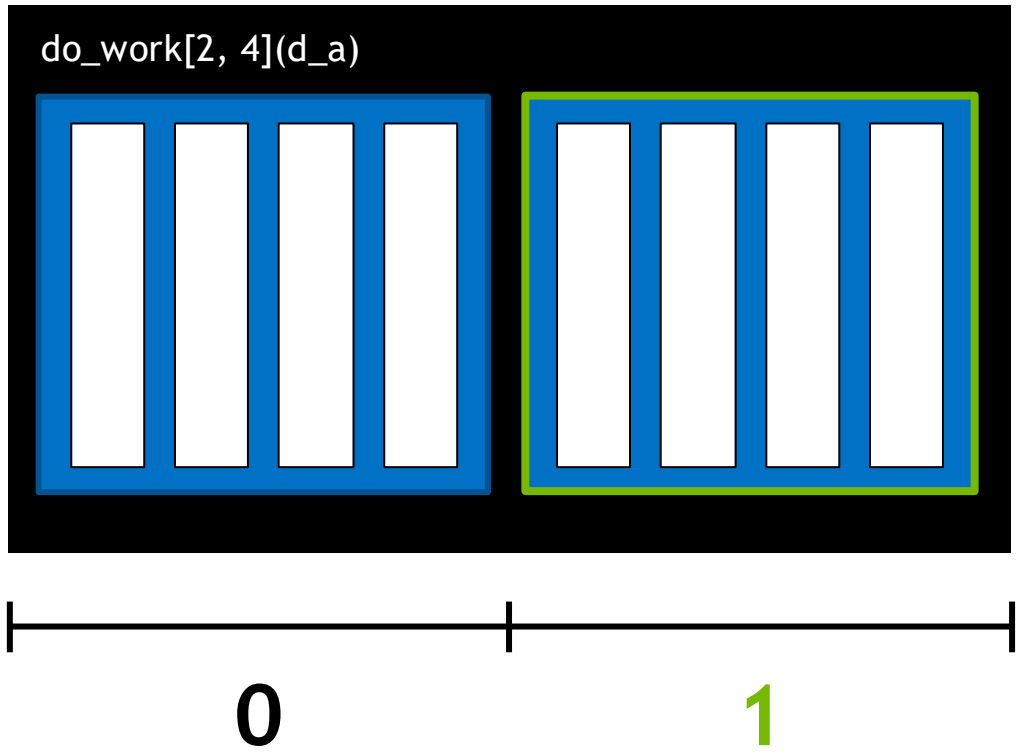
blockIdx.x 是网格中当前块的索引，
在本例中为 **0**

GPU



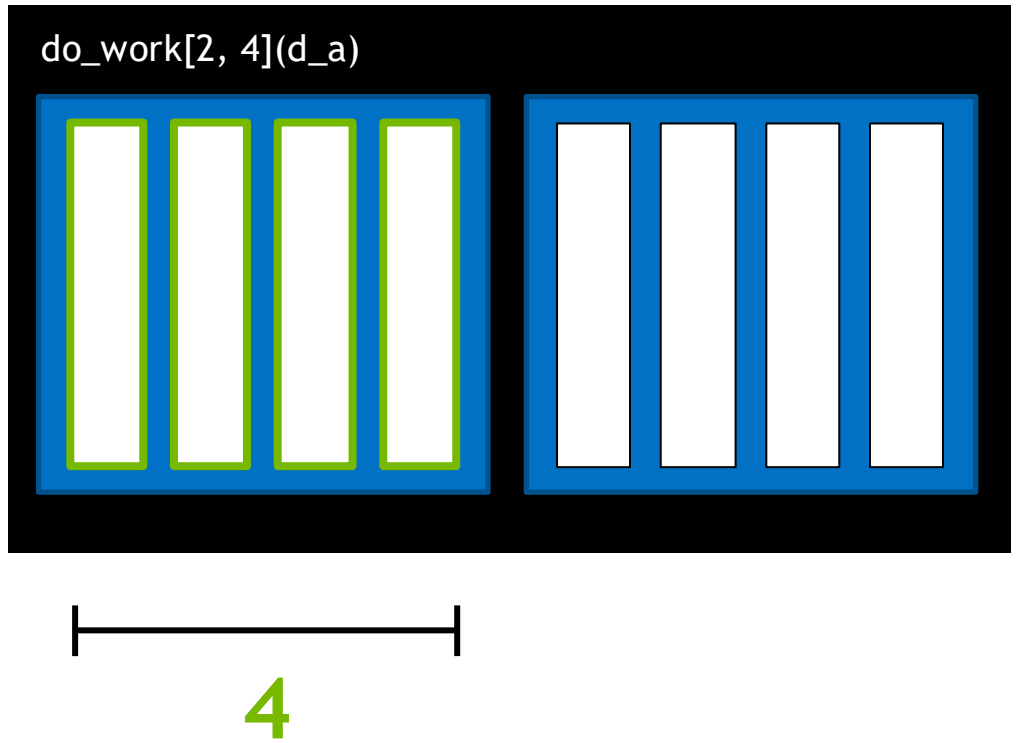
blockIdx.x 是网格中当前块的索引，在本例中为 **1**

GPU



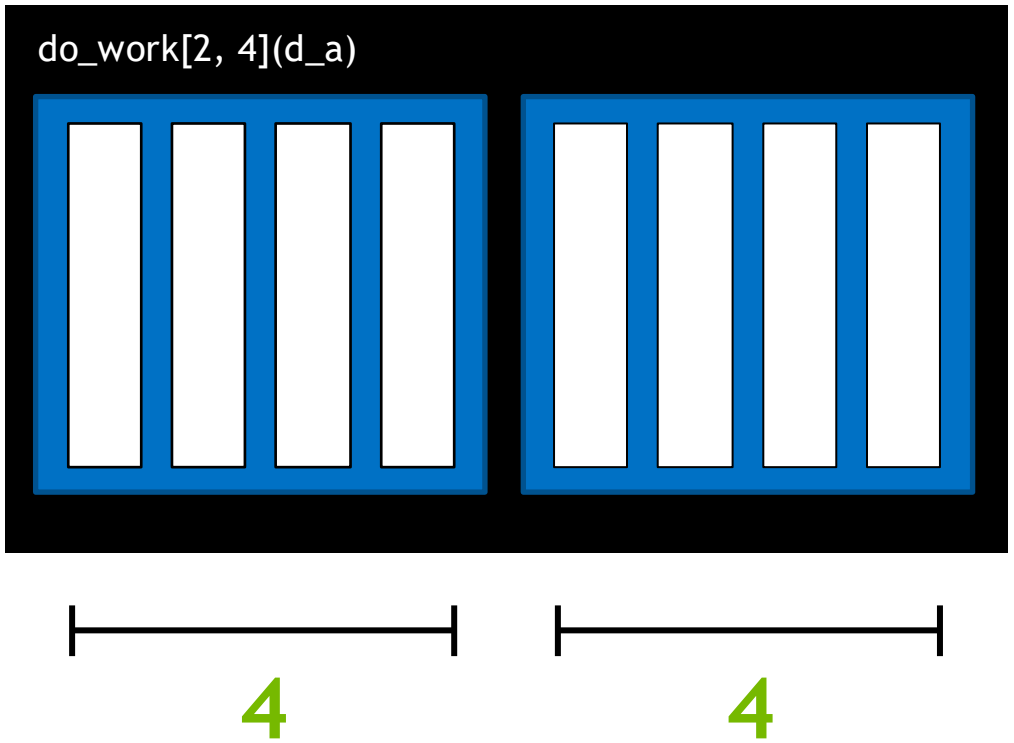
在核函数中，**blockDim.x** 描述了块中的线程数。在本例中为 **4**

GPU



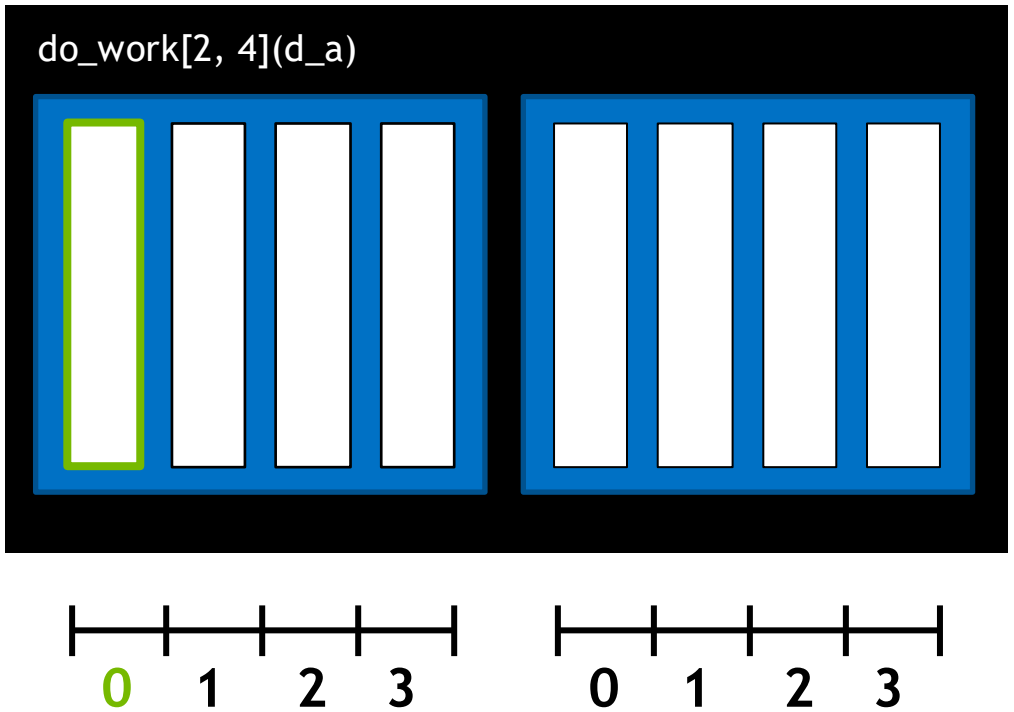
网格中的所有块均包含相同数量的线程

GPU



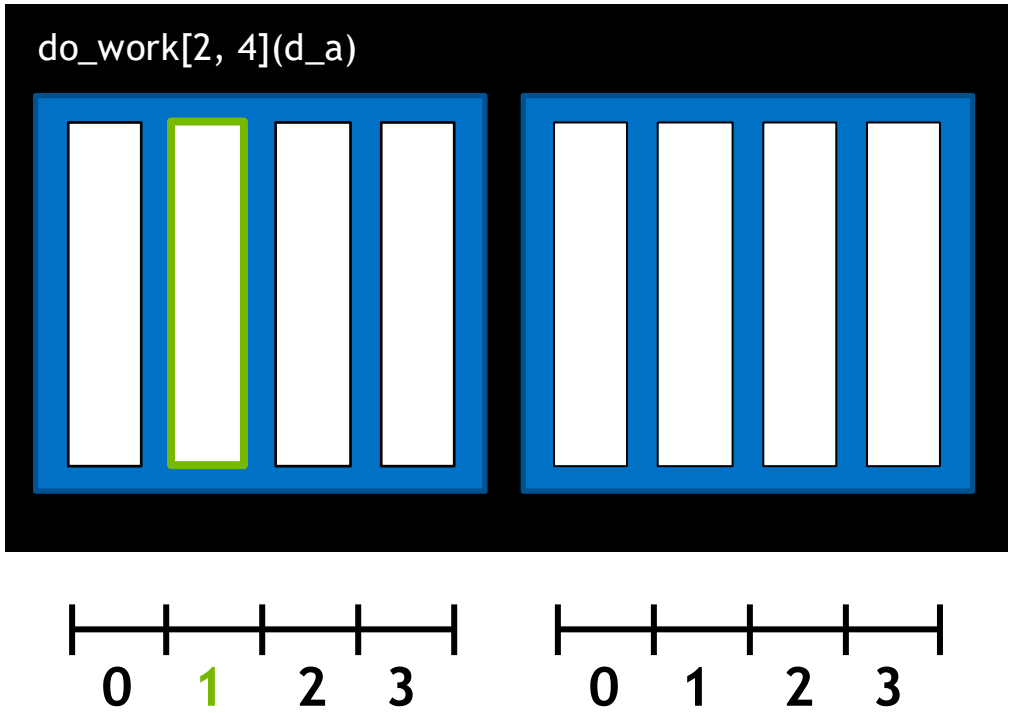
在核函数中，**threadIdx.x** 描述了块中所包含线程的索引。在本例中为 **0**

GPU



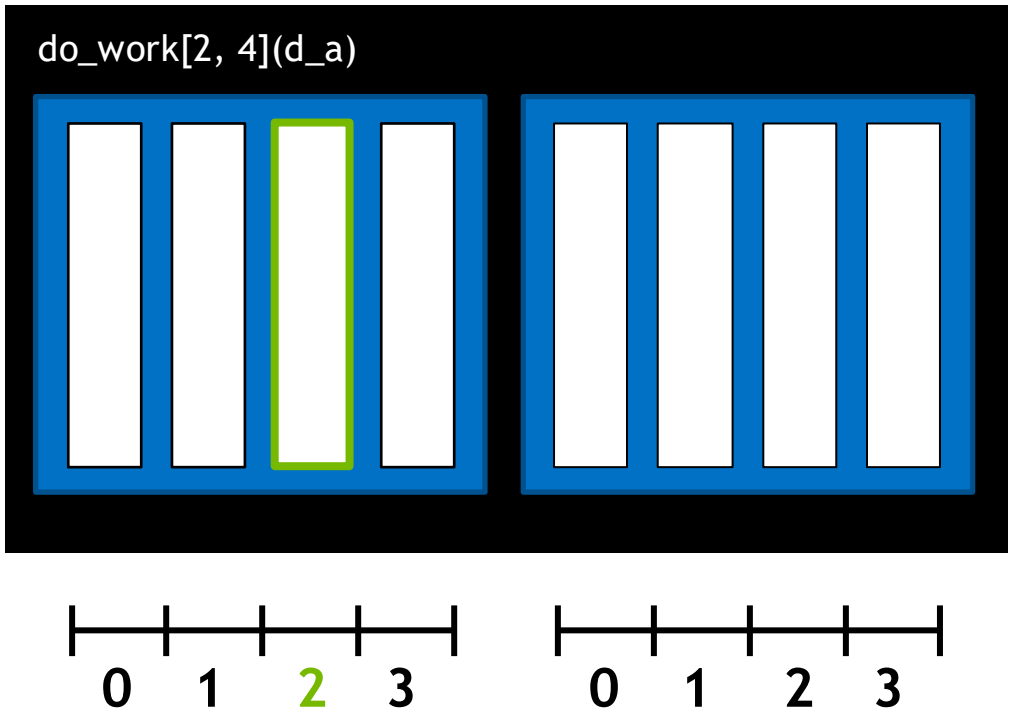
在核函数中，**threadIdx.x** 描述了块中所包含线程的索引。在本例中为 **1**

GPU



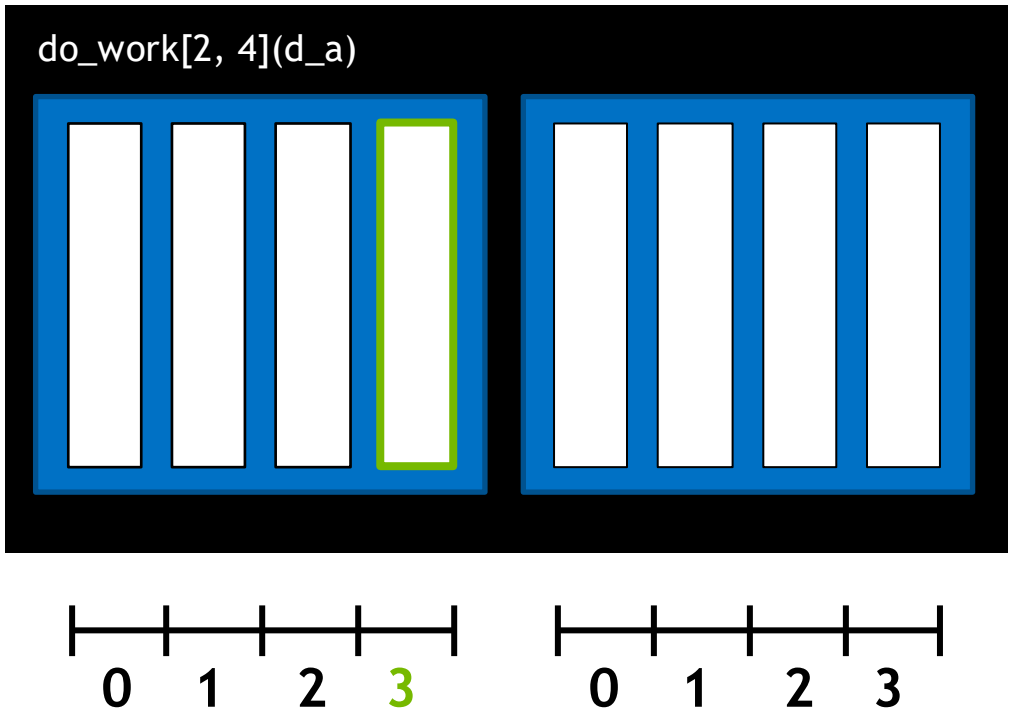
在核函数中，**threadIdx.x** 描述了块中所包含线程的索引。在本例中为 **2**

GPU



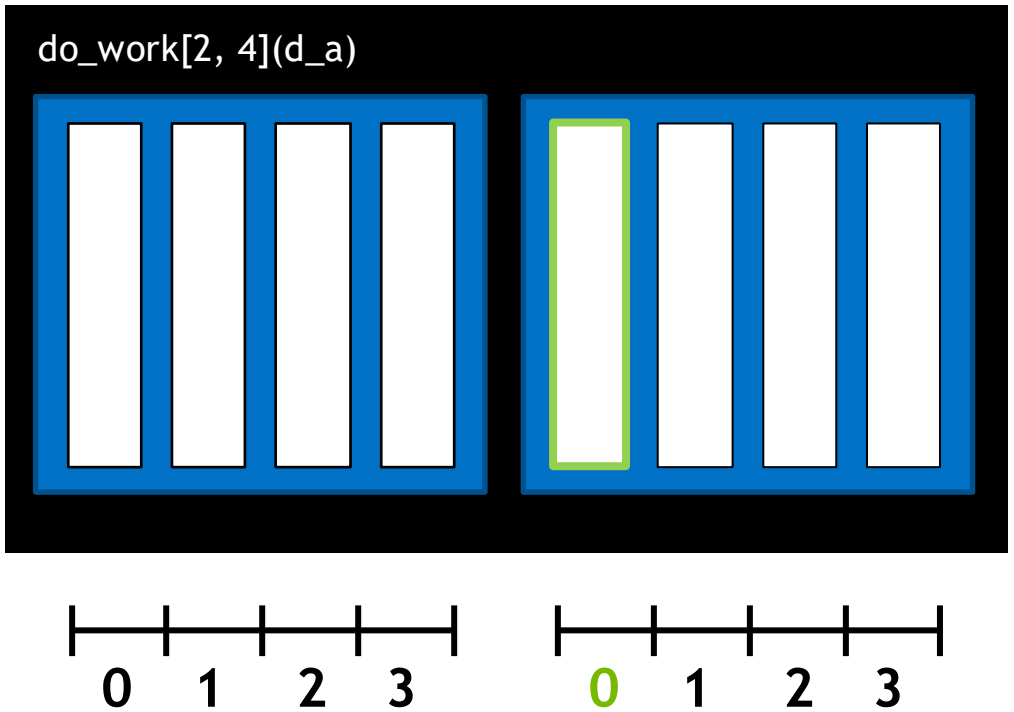
在核函数中，**threadIdx.x** 描述了块中所包含线程的索引。在本例中为 **3**

GPU



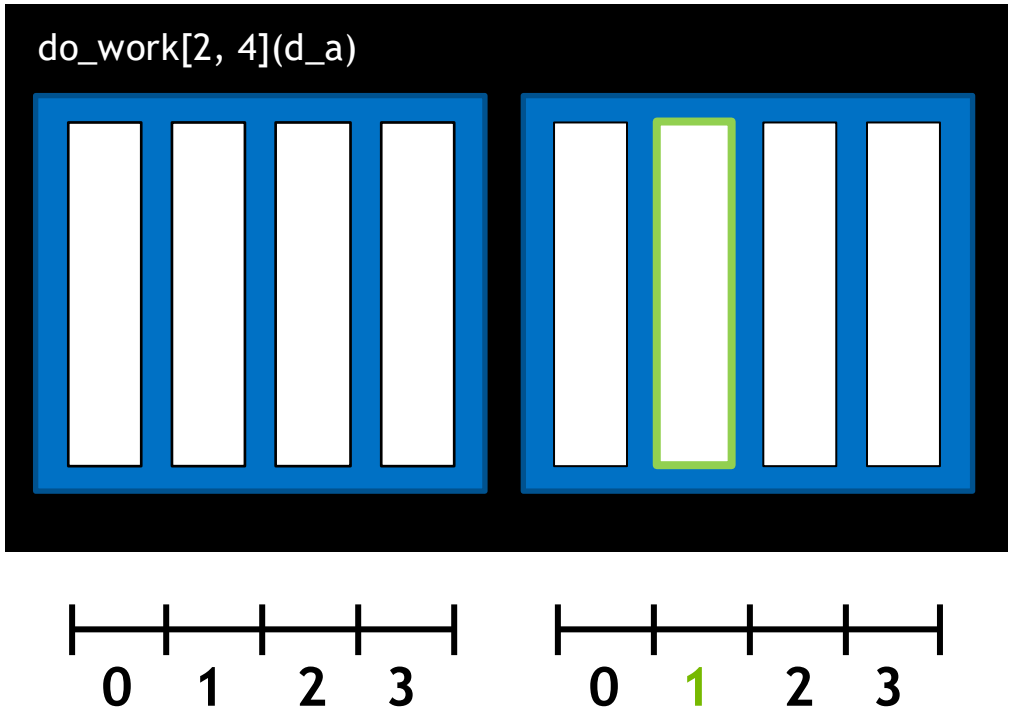
在核函数中，**threadIdx.x** 描述了块中所包含线程的索引。在本例中为 **0**

GPU



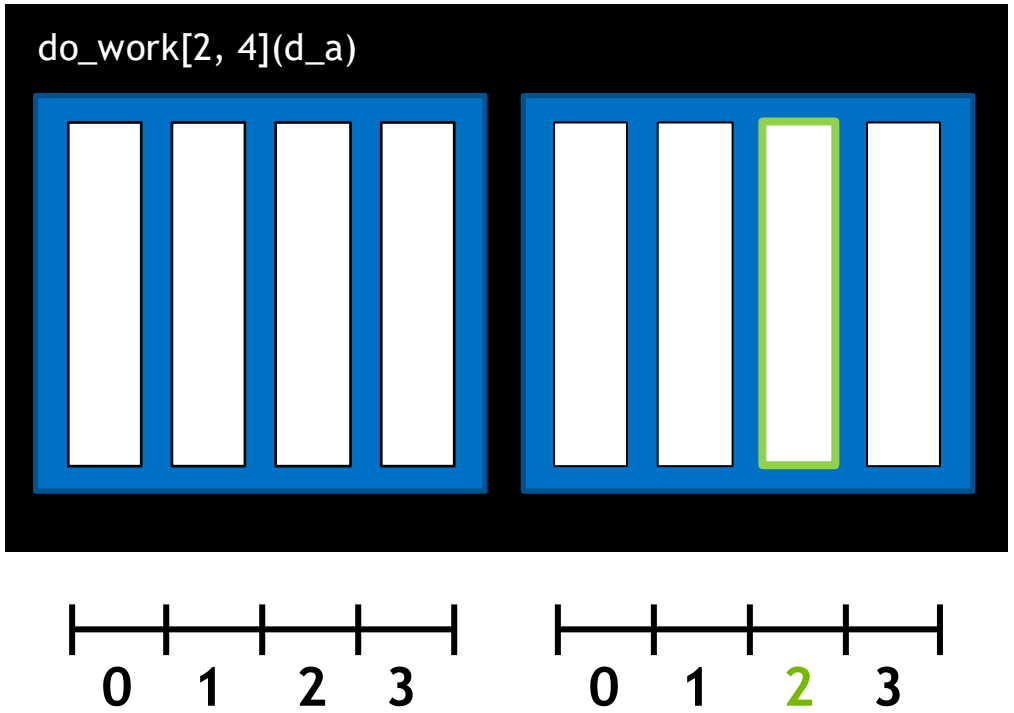
在核函数中，**threadIdx.x** 描述了块中所包含线程的索引。在本例中为 **1**

GPU



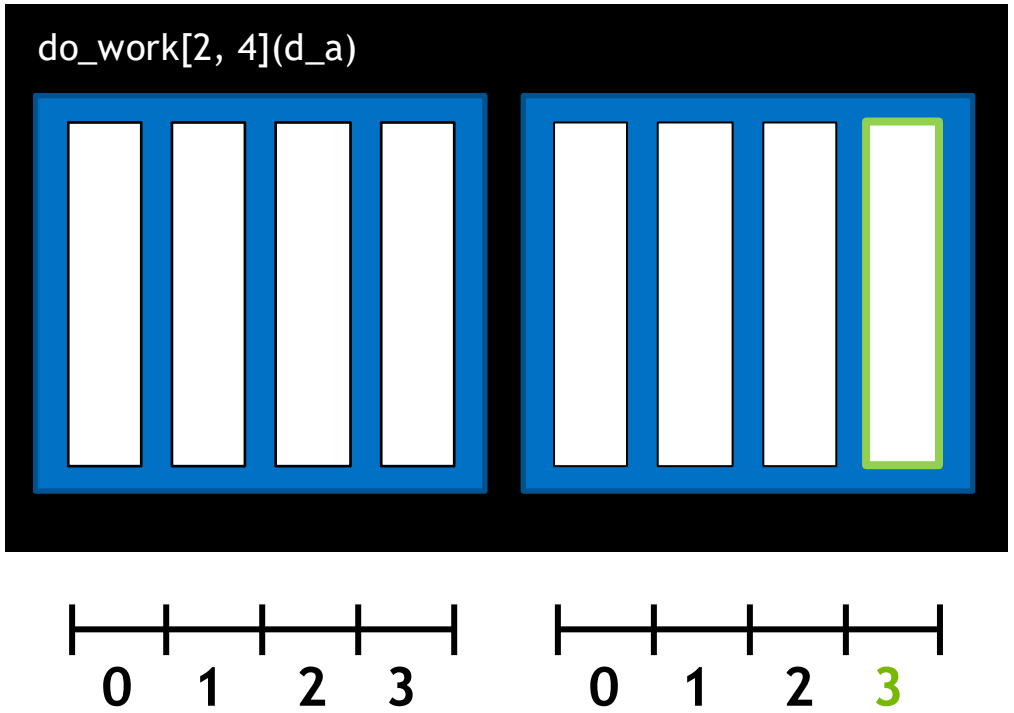
在核函数中，**threadIdx.x** 描述了块中所包含线程的索引。在本例中为 **2**

GPU



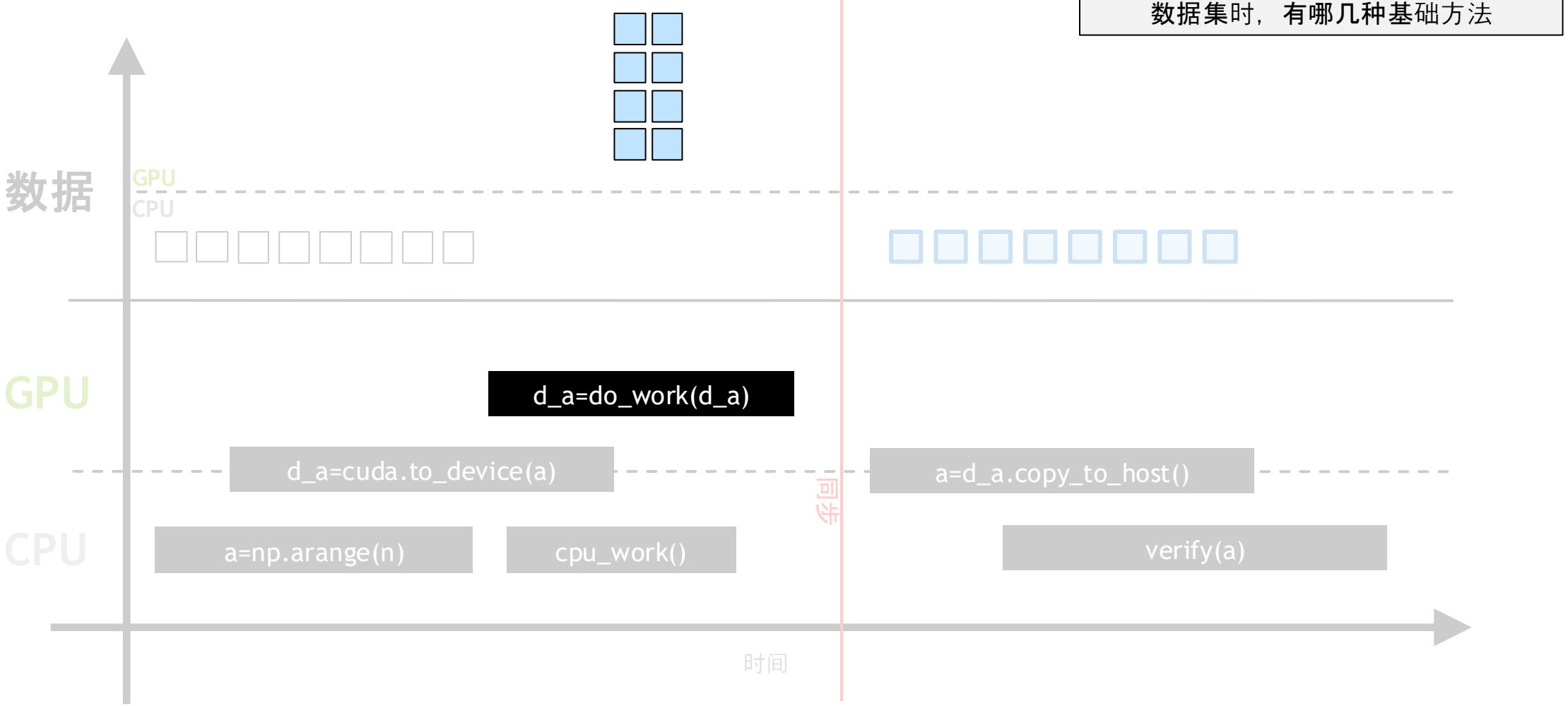
在核函数中，**threadIdx.x** 描述了块中所包含线程的索引。在本例中为 **3**

GPU



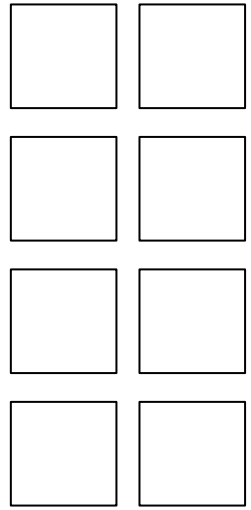
协调并行线程

让我们了解一下在协调并行线程处理**数据集**时，**有哪几种基**础方法



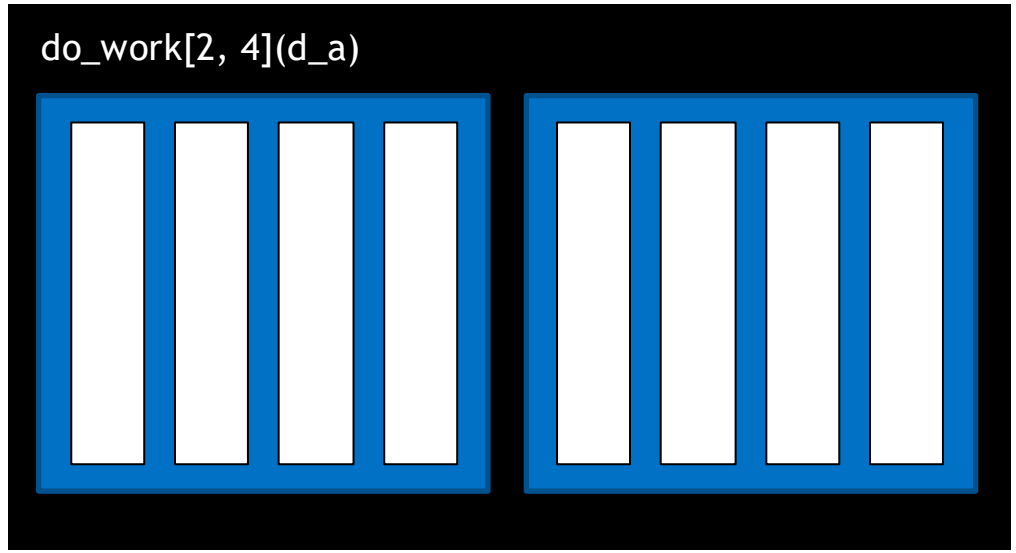
GPU

数据



假设数据位于起始索引为 0 的向量中

GPU



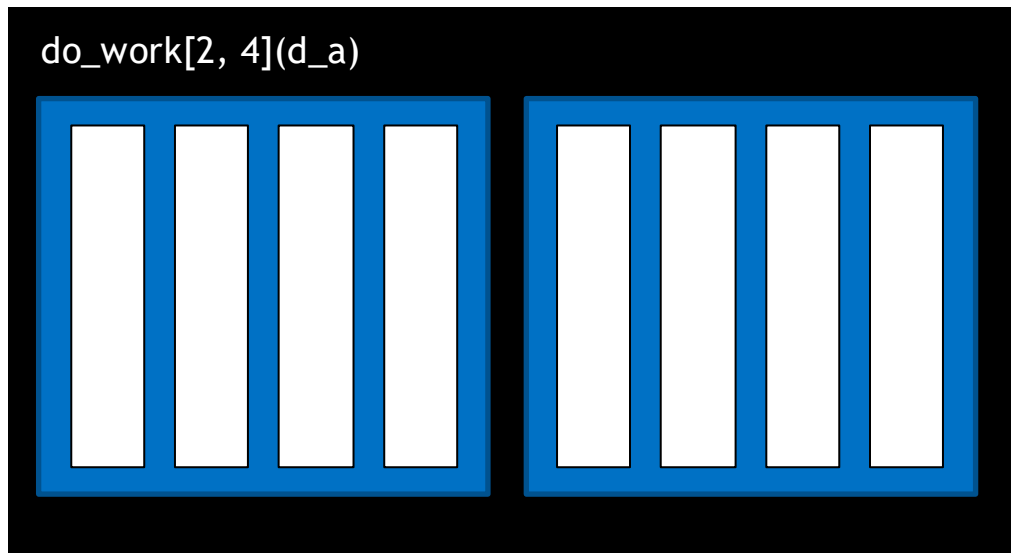
GPU

数据

0	4
1	5
2	6
3	7

假设数据位于起始索引为 0 的向量中

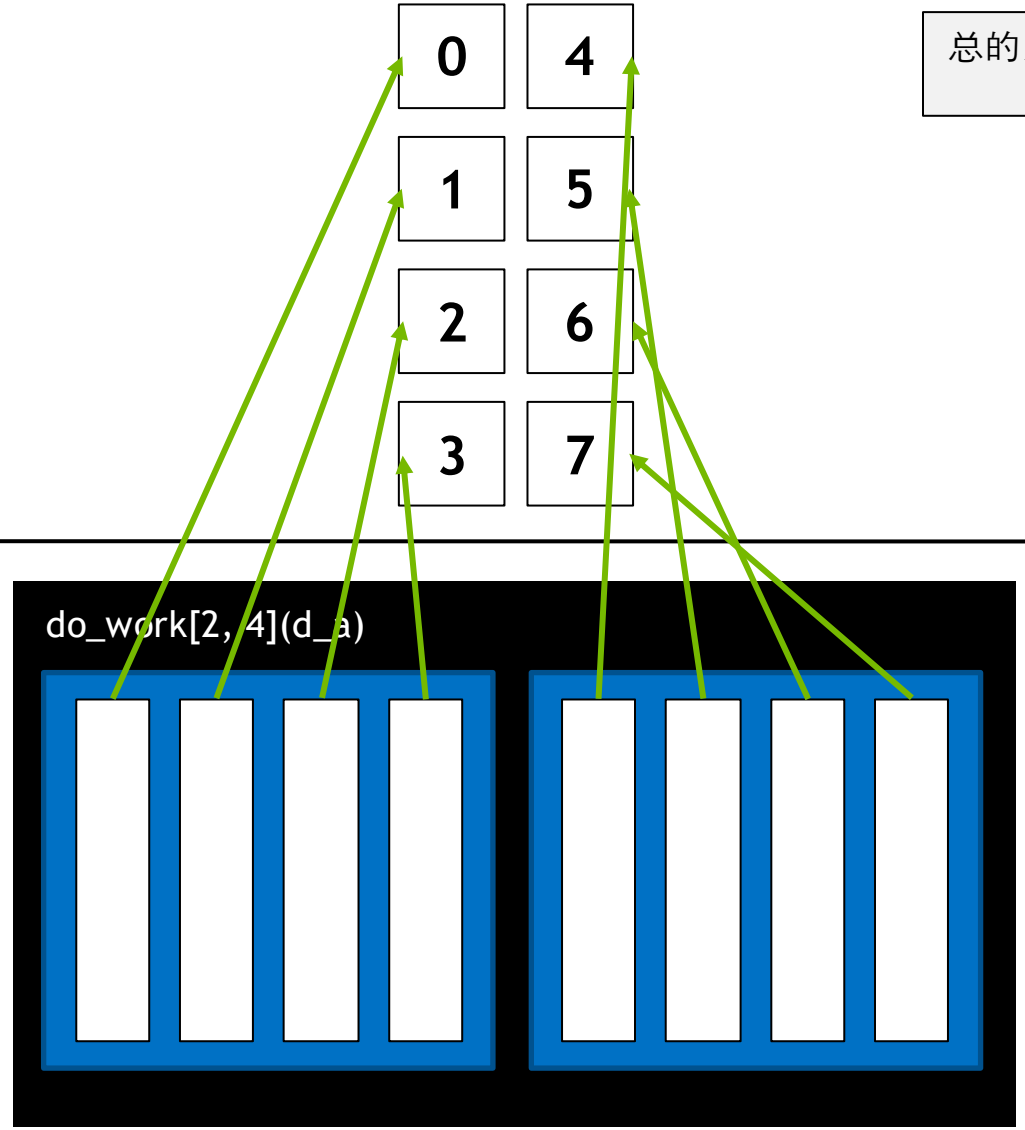
GPU



总的来说，**必**须映射每个线程以处理数据中的元素

GPU

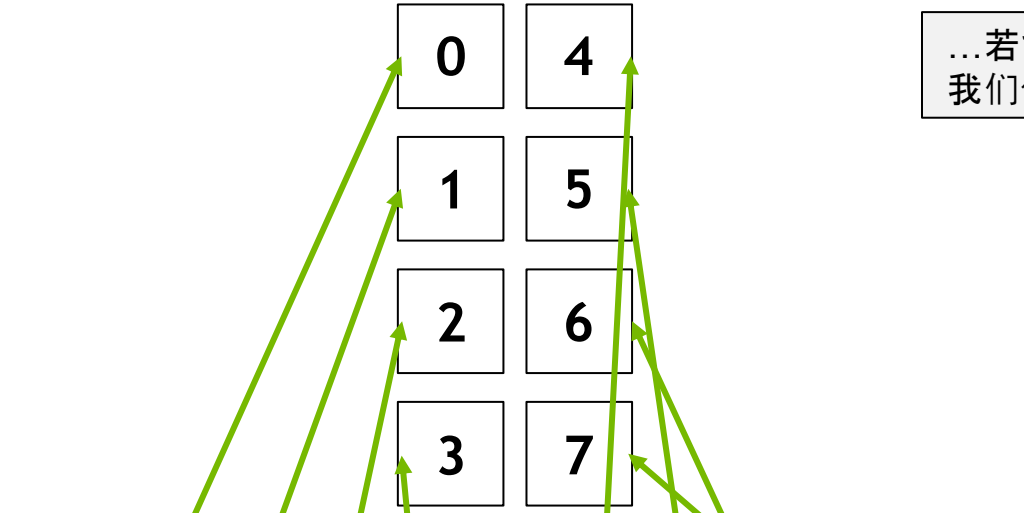
数据



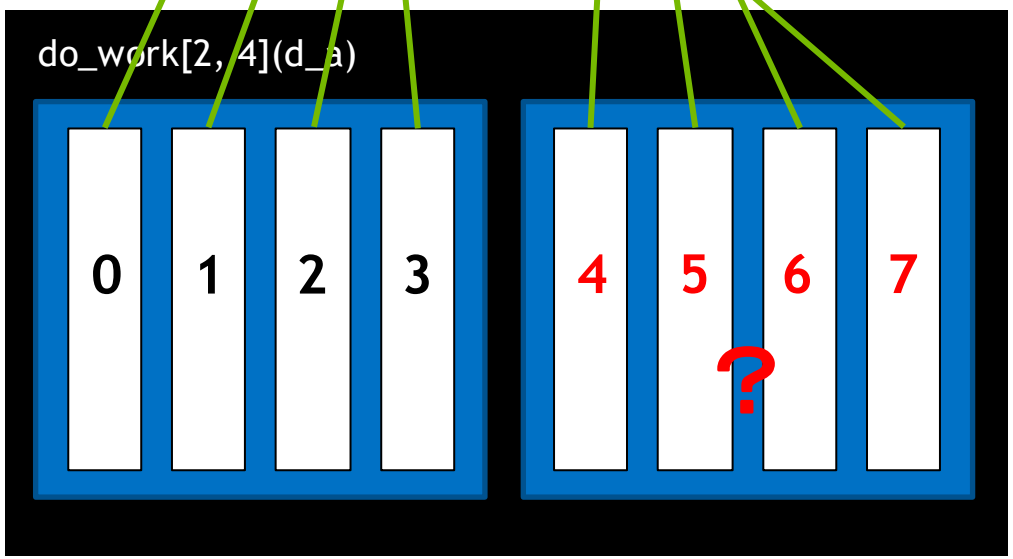
GPU

…若能计算出整个网格中的线程索引，
我们便可将该索引映射到数据中的索引

GPU
数据



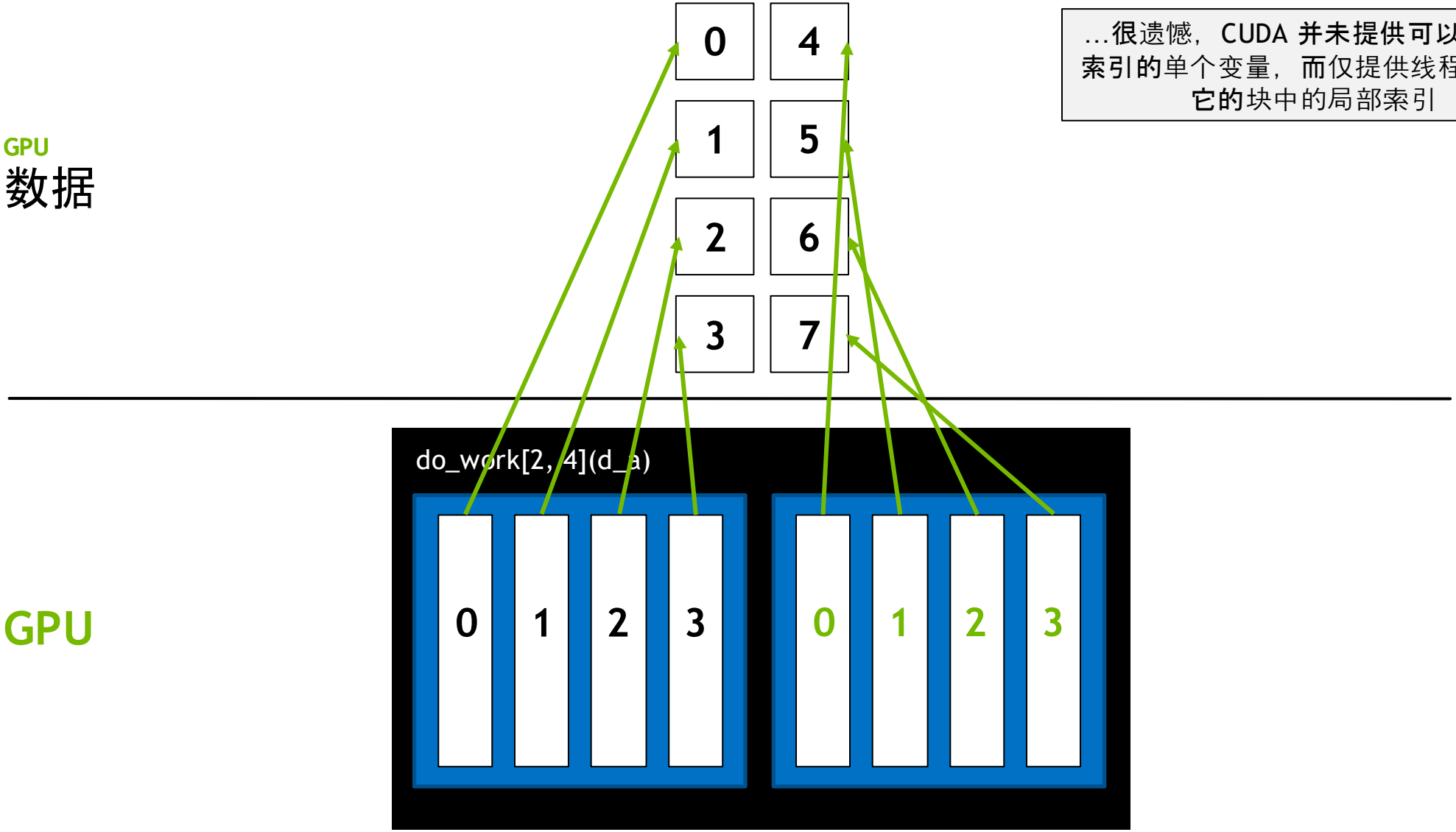
GPU



...很遗憾，CUDA 并未提供可以获得此索引的单个变量，而仅提供线程在包含它的块中的局部索引

GPU

数据



GPU

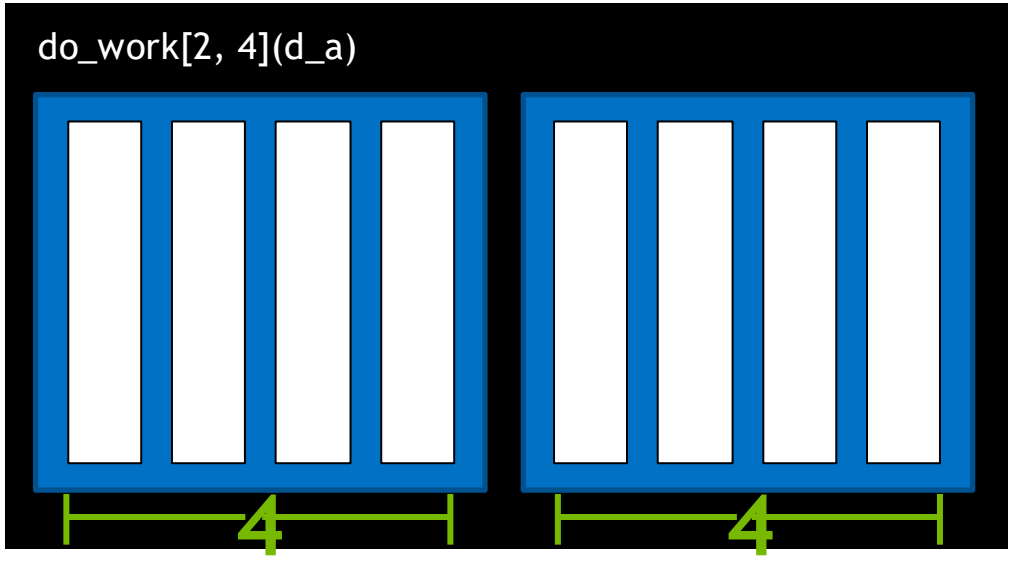
GPU

数据

0	4
1	5
2	6
3	7

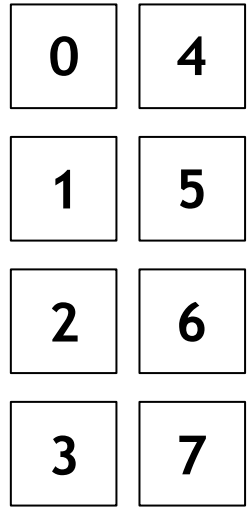
不过，我们还可通过一个惯用方法来计算该值。回想一下，每个线程都可以通过 **blockDim.x** 访问所在块的大小

GPU



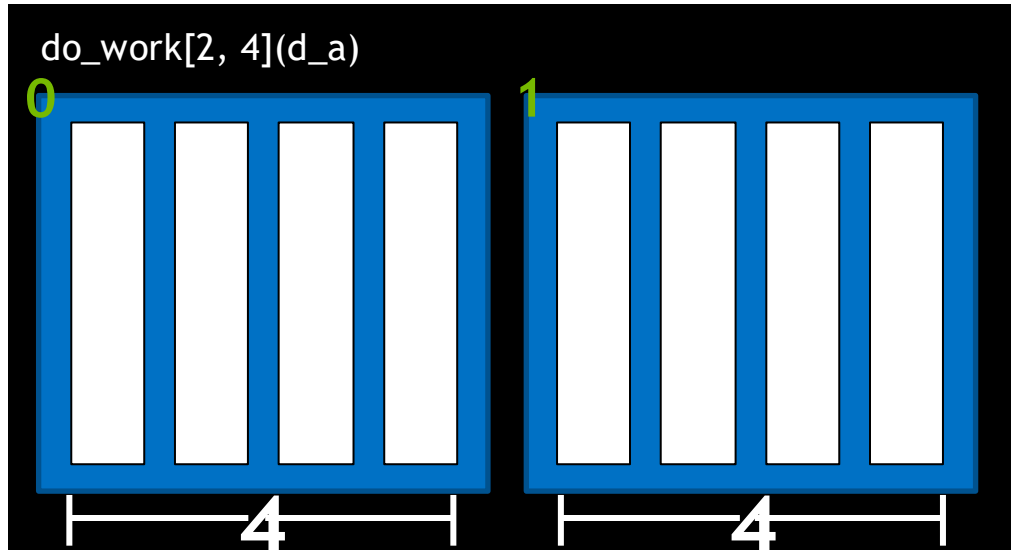
GPU

数据



…并通过 **blockIdx.x** 访问网格内其所在块的索引

GPU



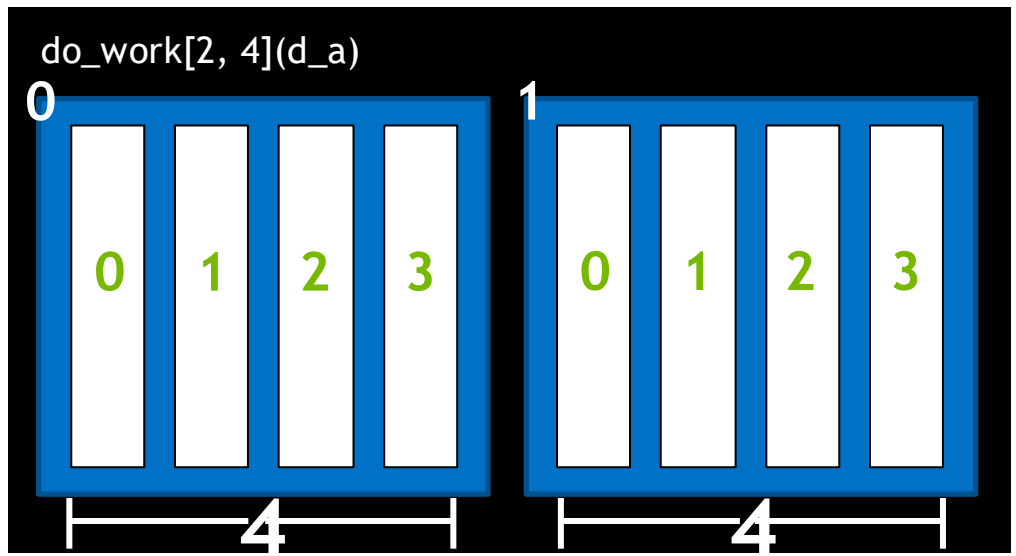
GPU

数据

0	4
1	5
2	6
3	7

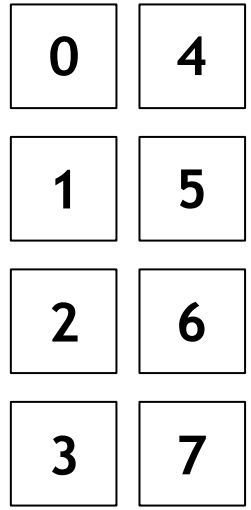
...并通过 **threadIdx.x** 访问所在块内自身的线程索引

GPU



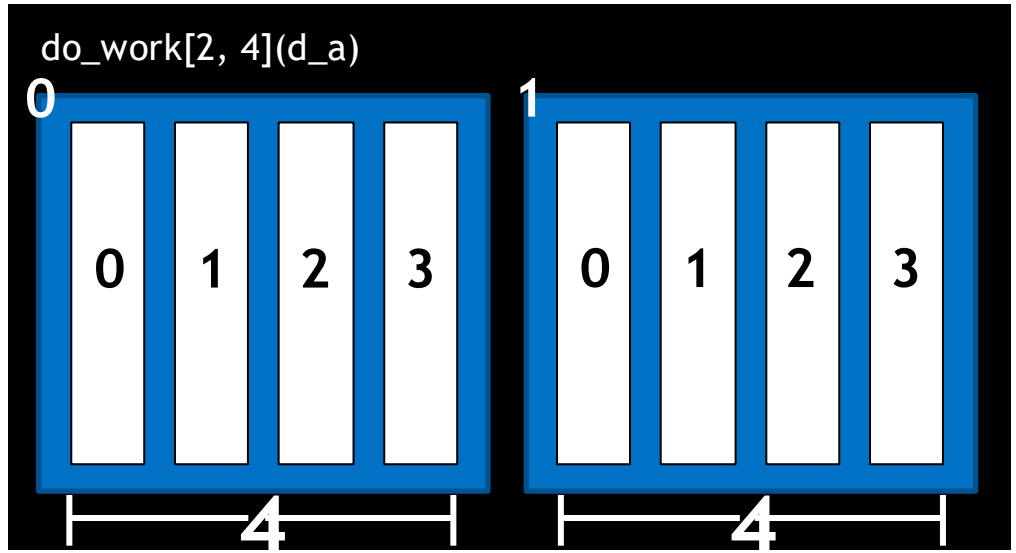
GPU

数据



利用这些变量，**threadIdx.x + blockIdx.x * blockDim.x** 公式将返回当前线程在整个网格中的唯一索引，之后我们便可将其映射至数据元素。

GPU



GPU

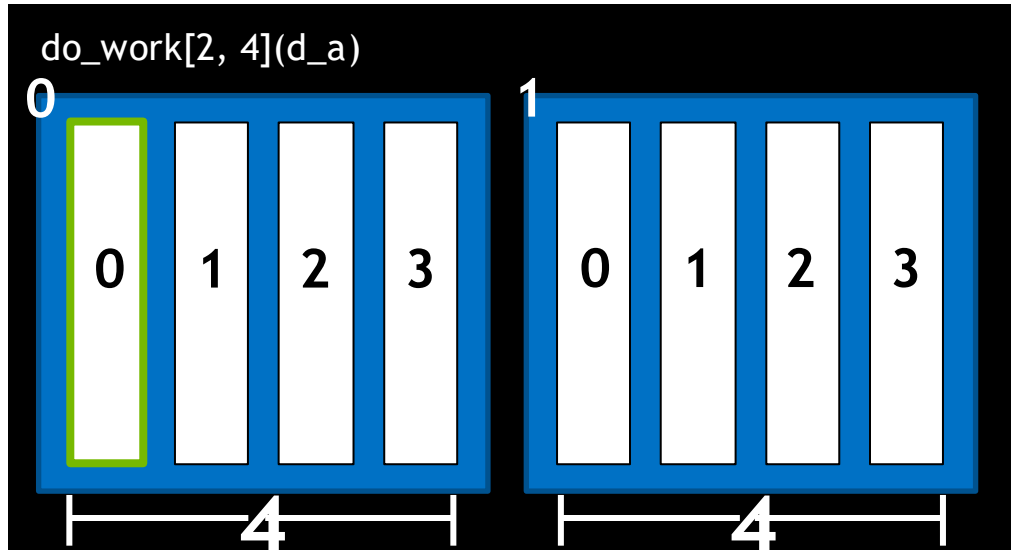
数据

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
0		0		4

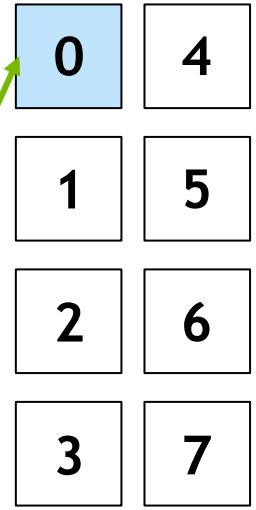
data_index
?

GPU



GPU

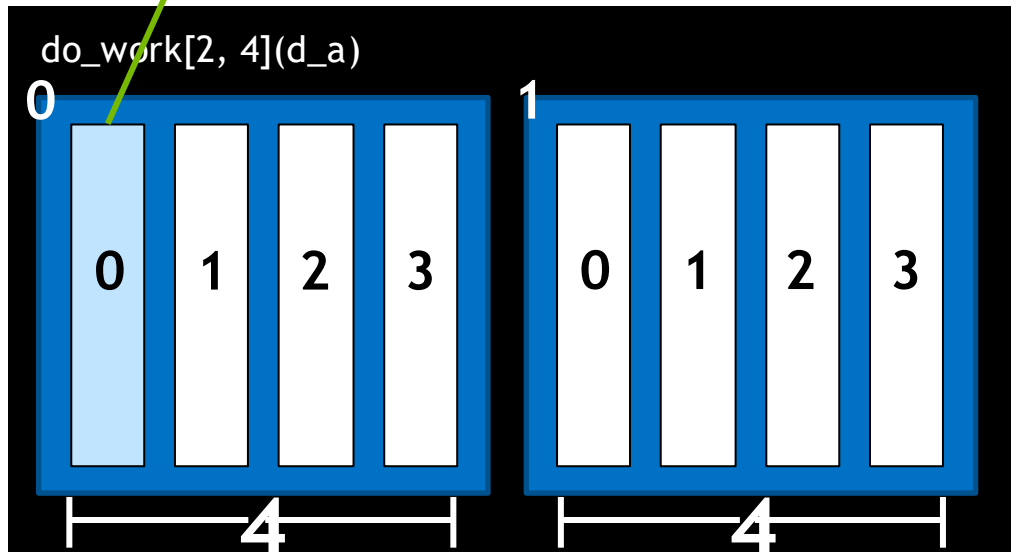
数据



threadIdx.x	+	blockIdx.x	*	blockDim.x
0		0		4

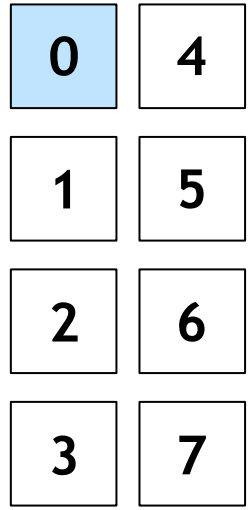
data_index
0

GPU



GPU

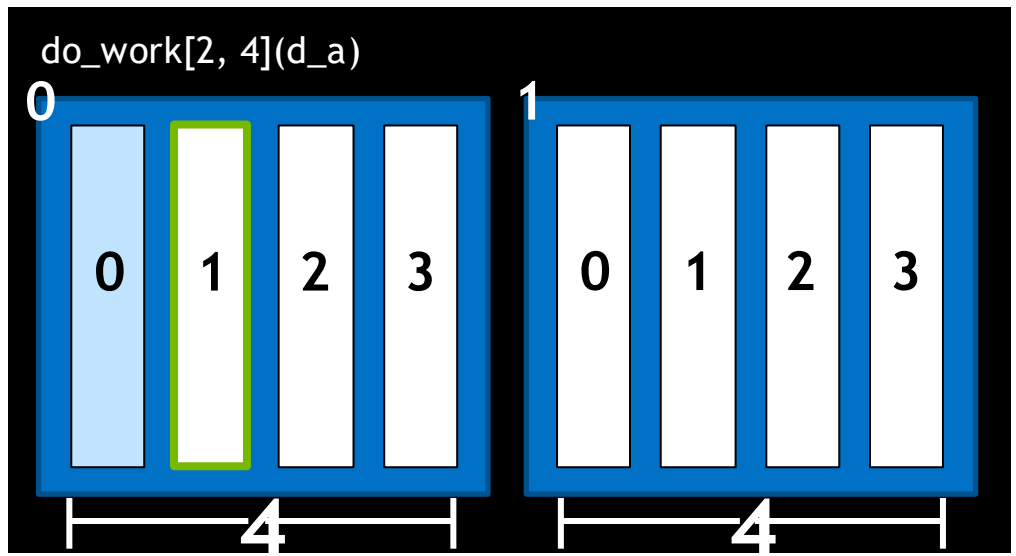
数据



threadIdx.x	+	blockIdx.x	*	blockDim.x
1		0		4

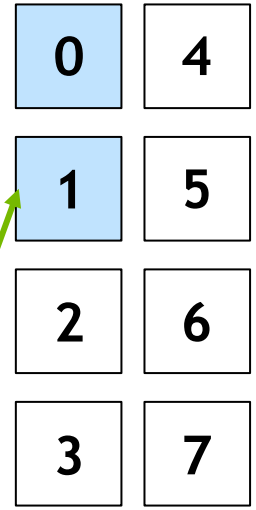
data_index
?

GPU



GPU

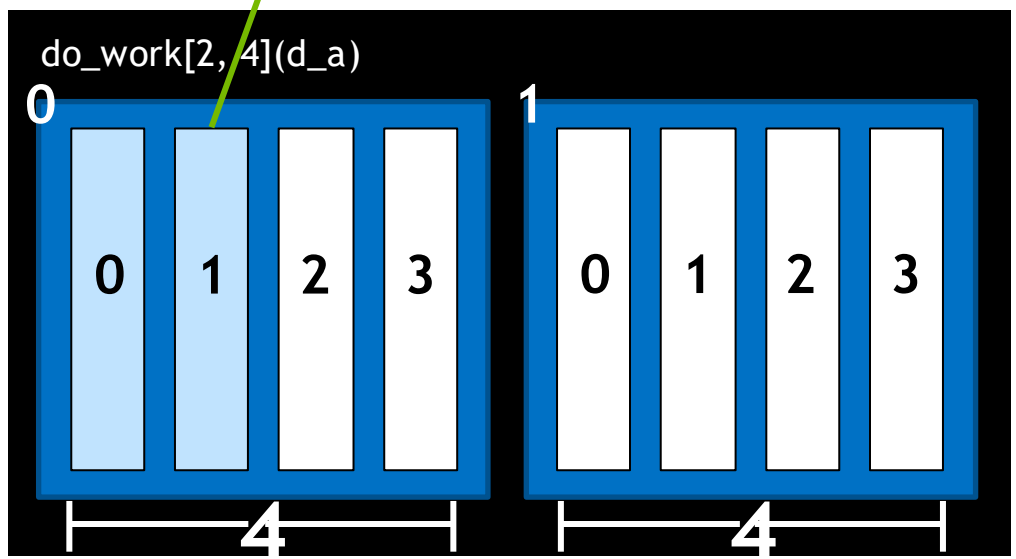
数据



threadIdx.x	+	blockIdx.x	*	blockDim.x
1		0		4

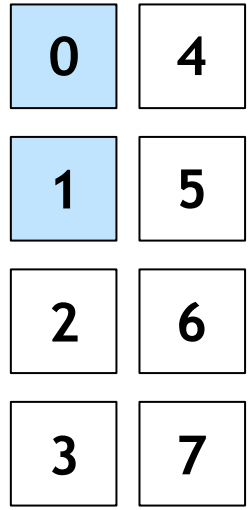
data_index
1

GPU



GPU

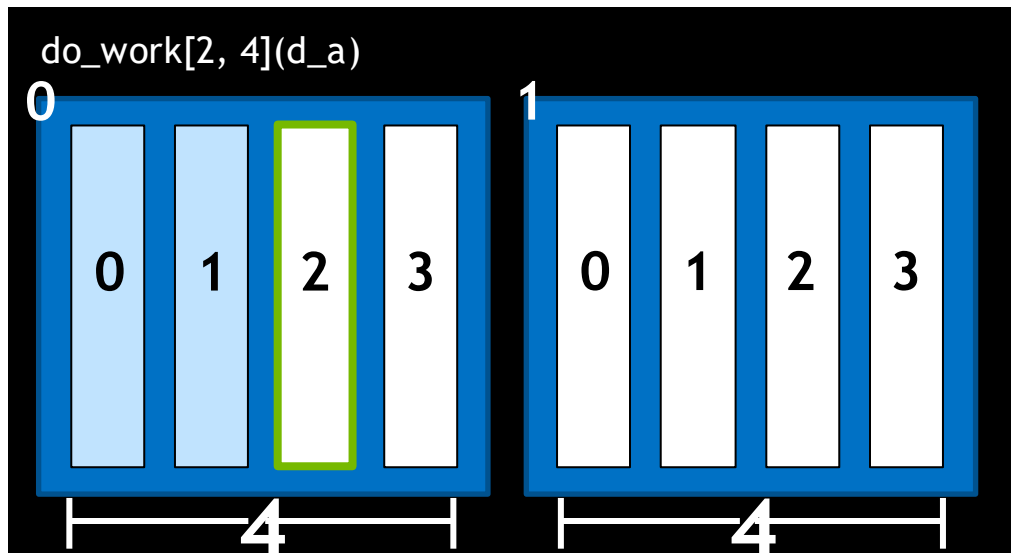
数据



threadIdx.x	+	blockIdx.x	*	blockDim.x
2		0		4

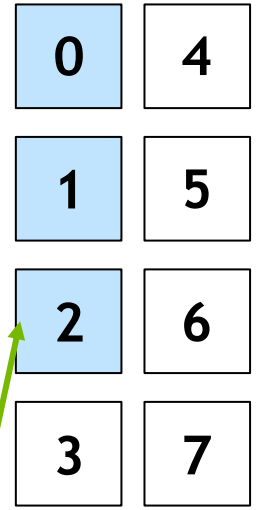
data_index
?

GPU



GPU

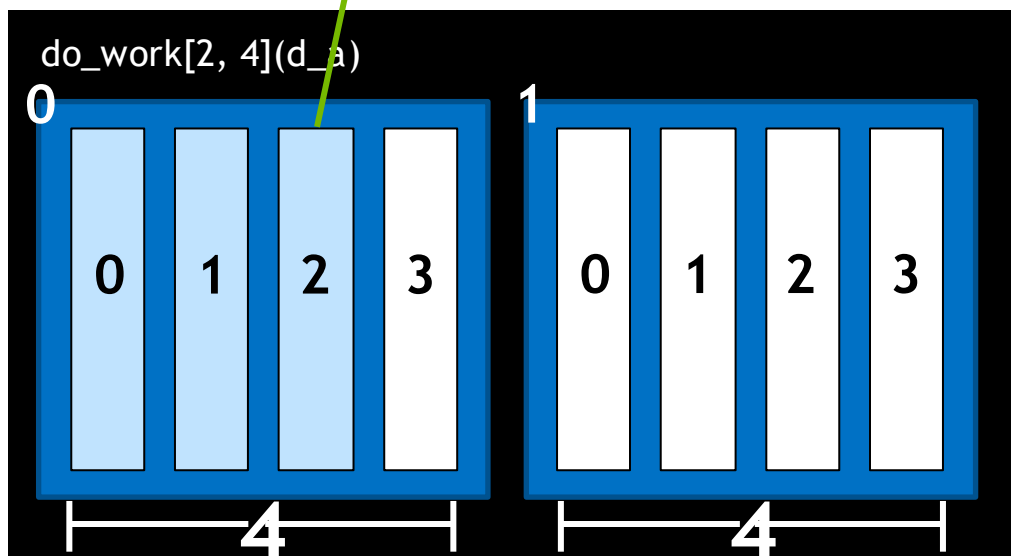
数据



threadIdx.x	+	blockIdx.x	*	blockDim.x
2		0		4

data_index
2

GPU



GPU

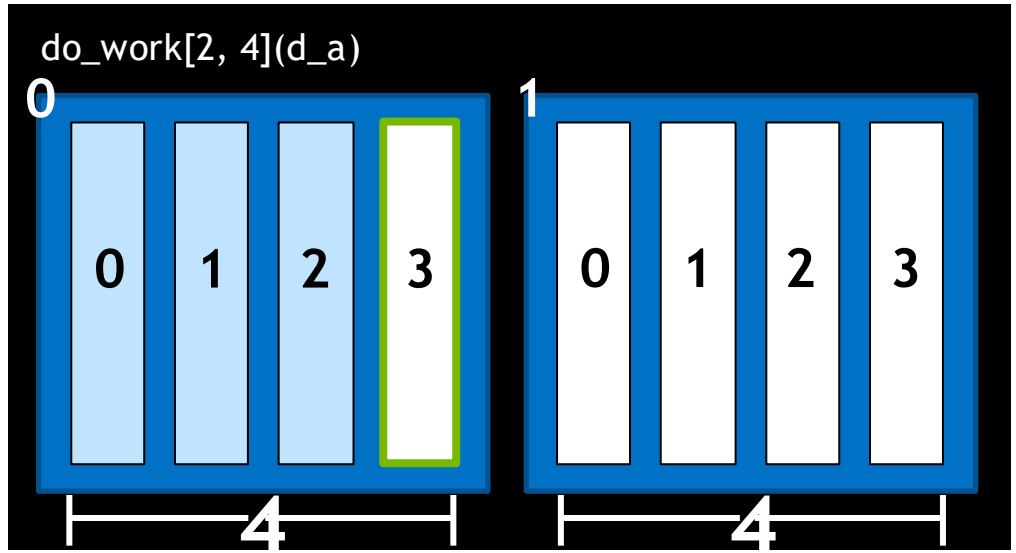
数据

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
3		0		4

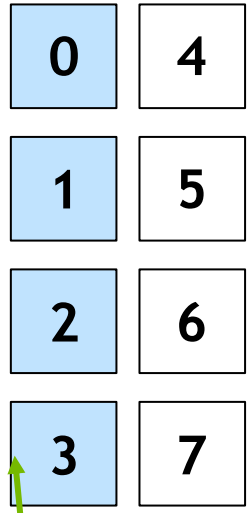
data_index
?

GPU



GPU

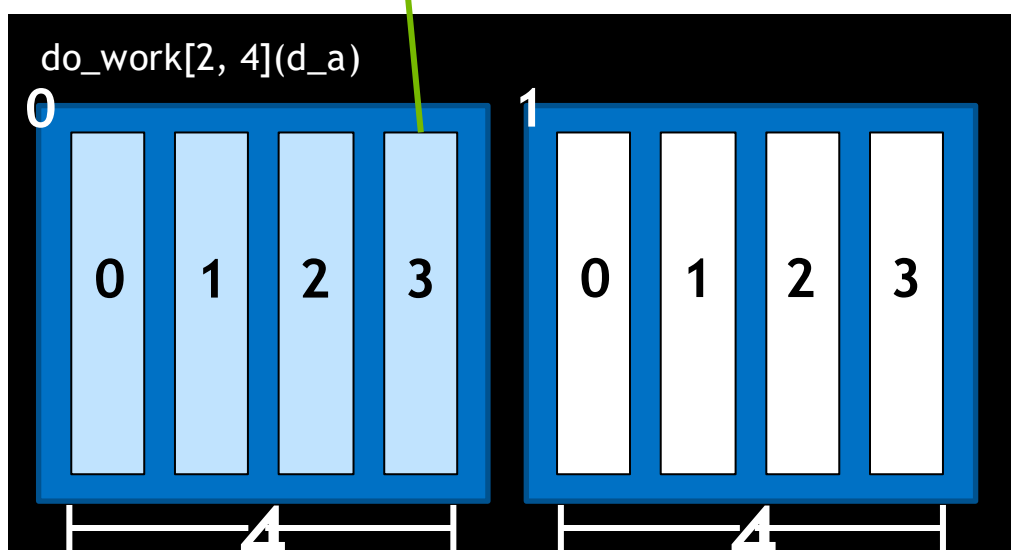
数据



threadIdx.x	+	blockIdx.x	*	blockDim.x
3		0		4

data_index
3

GPU



GPU

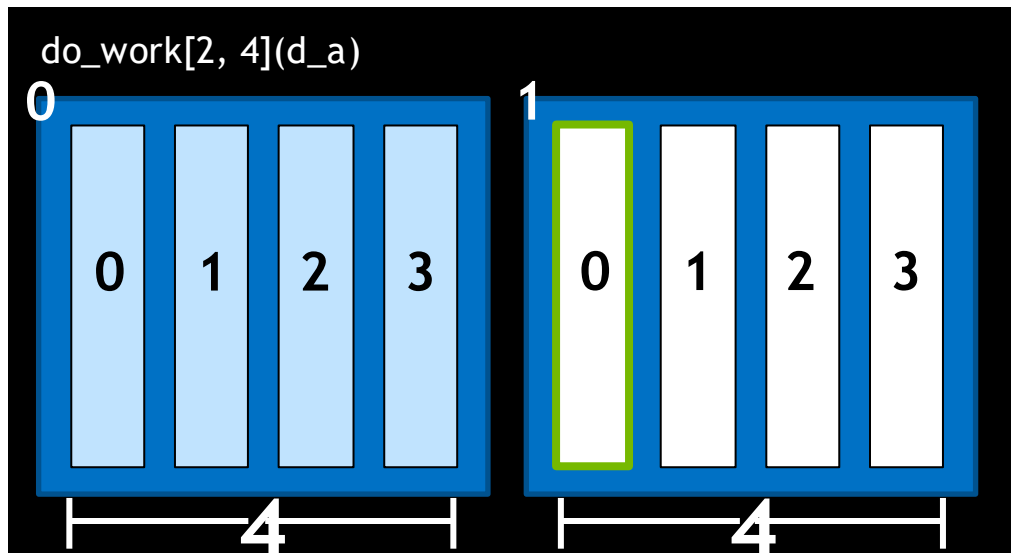
数据

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
0		1		4

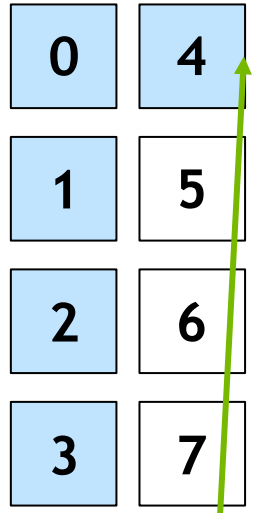
data_index
?

GPU



GPU

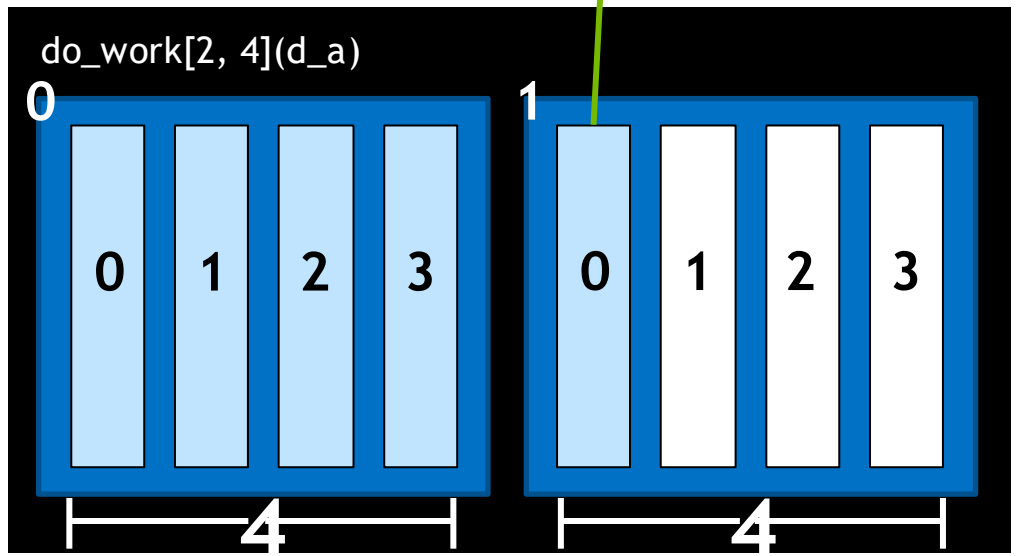
数据



threadIdx.x	+	blockIdx.x	*	blockDim.x
0		1		4

data_index
4

GPU



GPU

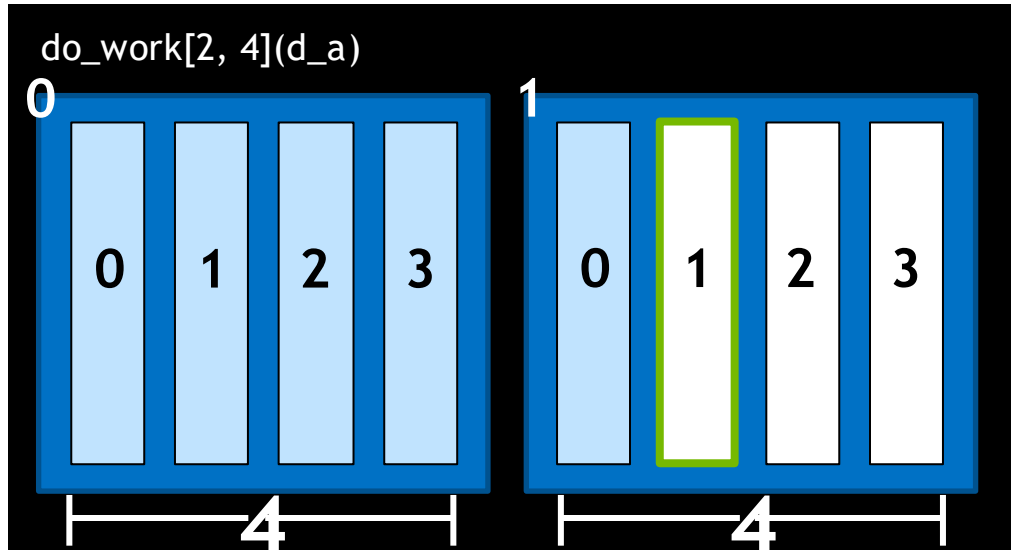
数据

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
1		1		4

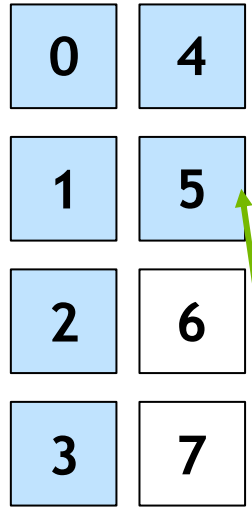
data_index
?

GPU



GPU

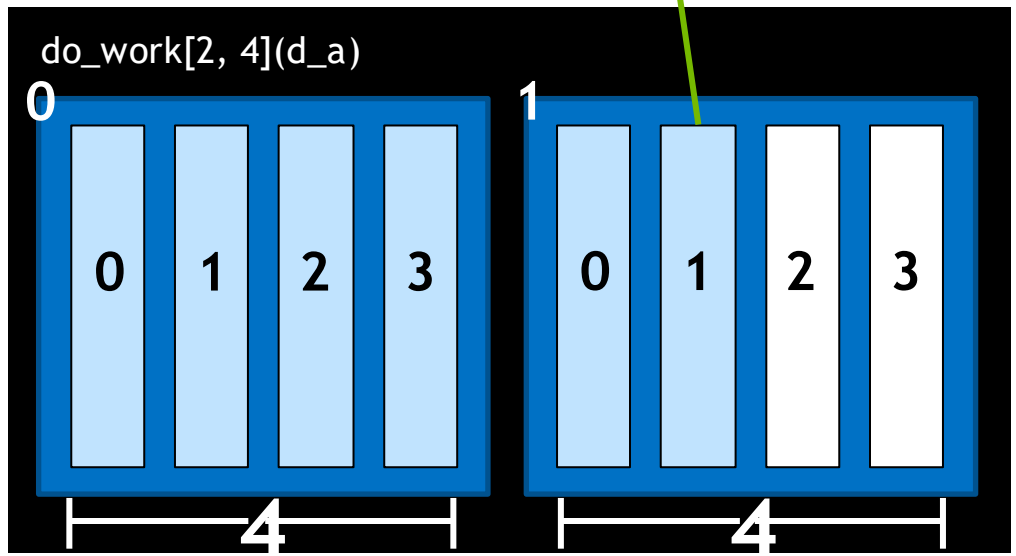
数据



threadIdx.x	+	blockIdx.x	*	blockDim.x
1		1		4

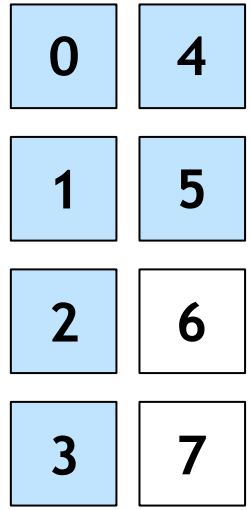
data_index
5

GPU



GPU

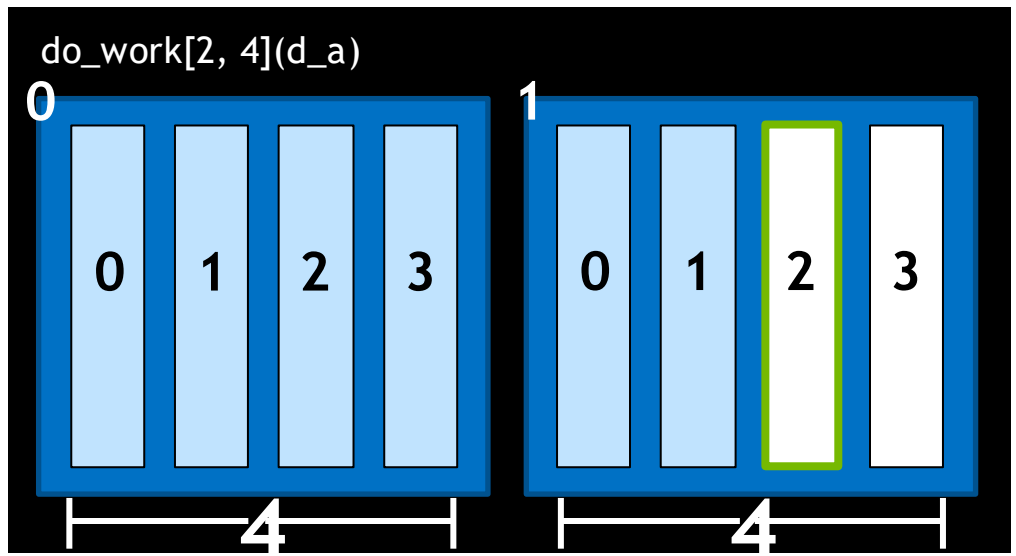
数据



threadIdx.x	+	blockIdx.x	*	blockDim.x
2		1		4

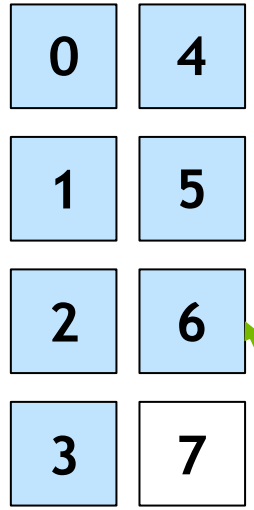
data_index
?

GPU



GPU

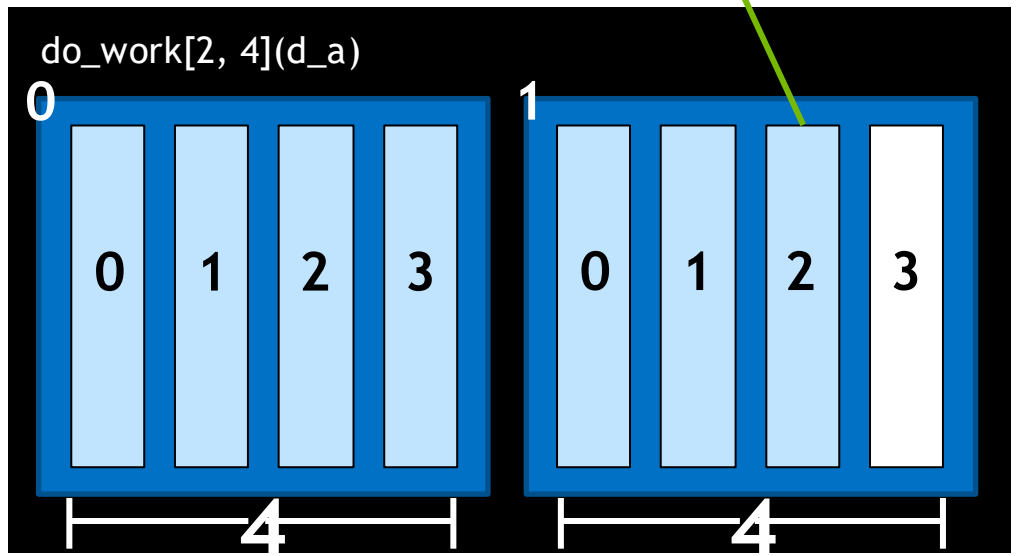
数据



threadIdx.x	+	blockIdx.x	*	blockDim.x
2		1		4

data_index
6

GPU



GPU

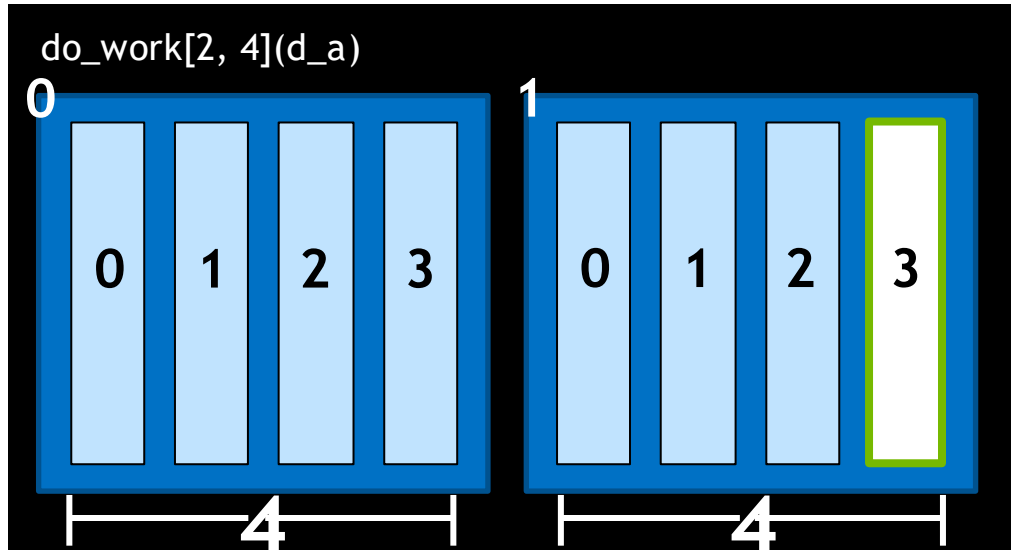
数据

0	4
1	5
2	6
3	7

threadIdx.x	+	blockIdx.x	*	blockDim.x
3		1		4

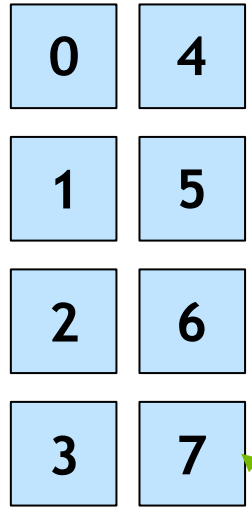
data_index
?

GPU



GPU

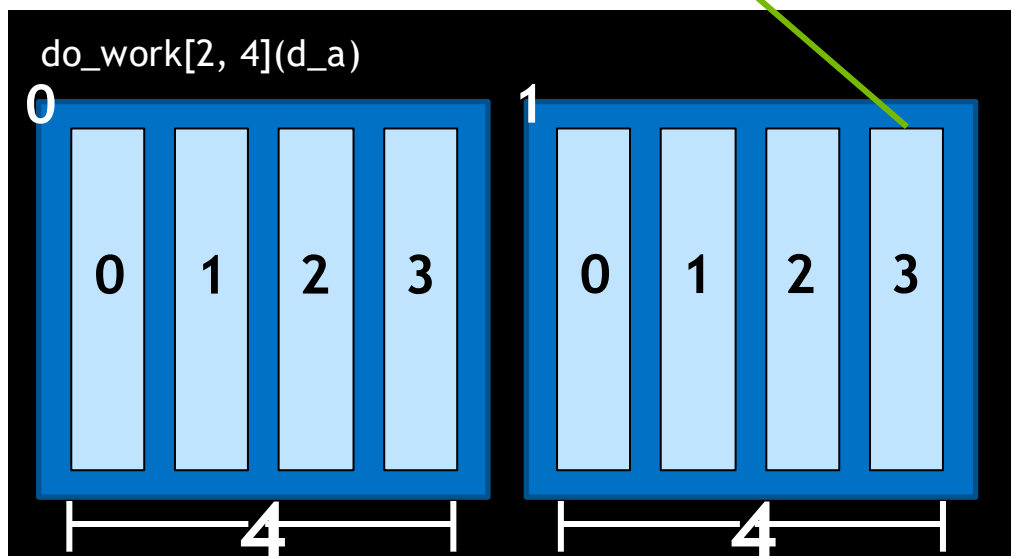
数据



threadIdx.x	+	blockIdx.x	*	blockDim.x
3		1		4

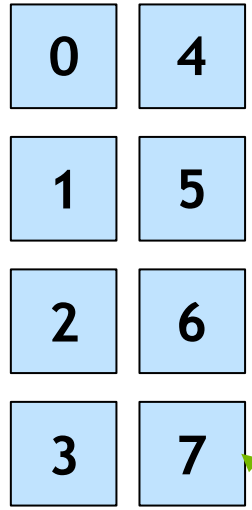
data_index
7

GPU



GPU

数据



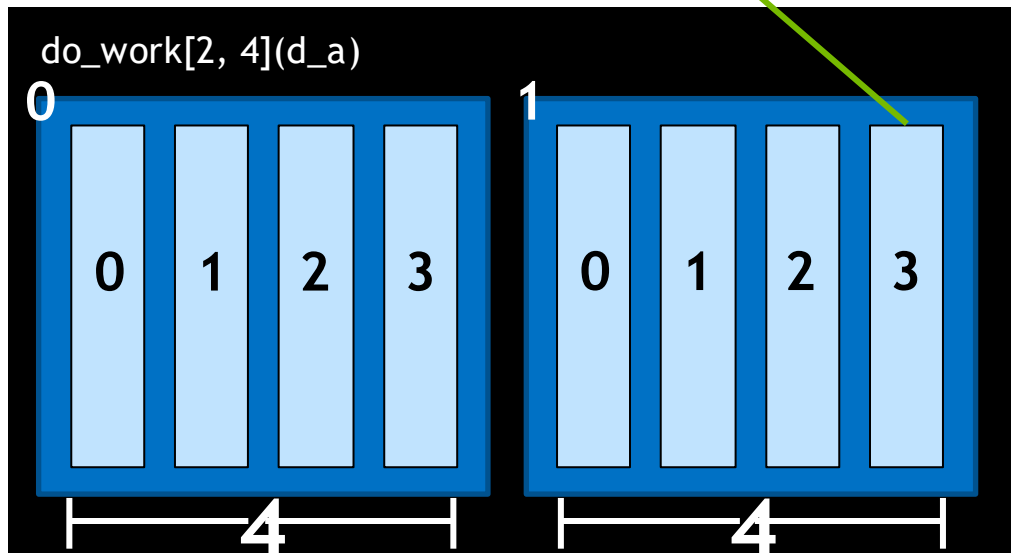
threadIdx.x	+	blockIdx.x	*	blockDim.x
3		1		4

data_index
7

grid(1)
7

为方便起见，Numba 提供了 `cuda.grid()` 函数，该函数可返回线程在网格中的唯一索引。

GPU



NVIDIA | DEEP LEARNING INSTITUTE

学习更多课程，请访问 www.nvidia.cn/DLI