CUDA Programming – Thread Organization

by

Dr. Nileshchandra Pikle

Assistant Professor

&

“A certified CUDA instructor by NVIDIA”

NVIDIA

DEEP
LEARNING
INSTITUTE

CERTIFIED
INSTRUCTOR

Contents

- CUDA Programming Model
- Thread Organzation using **HelloGPU.cu**
- **Case study:**
  1. vector addition
  2. Matrix Transpose
  3. 1D convolution
  4. 2D convolution

A typical C program

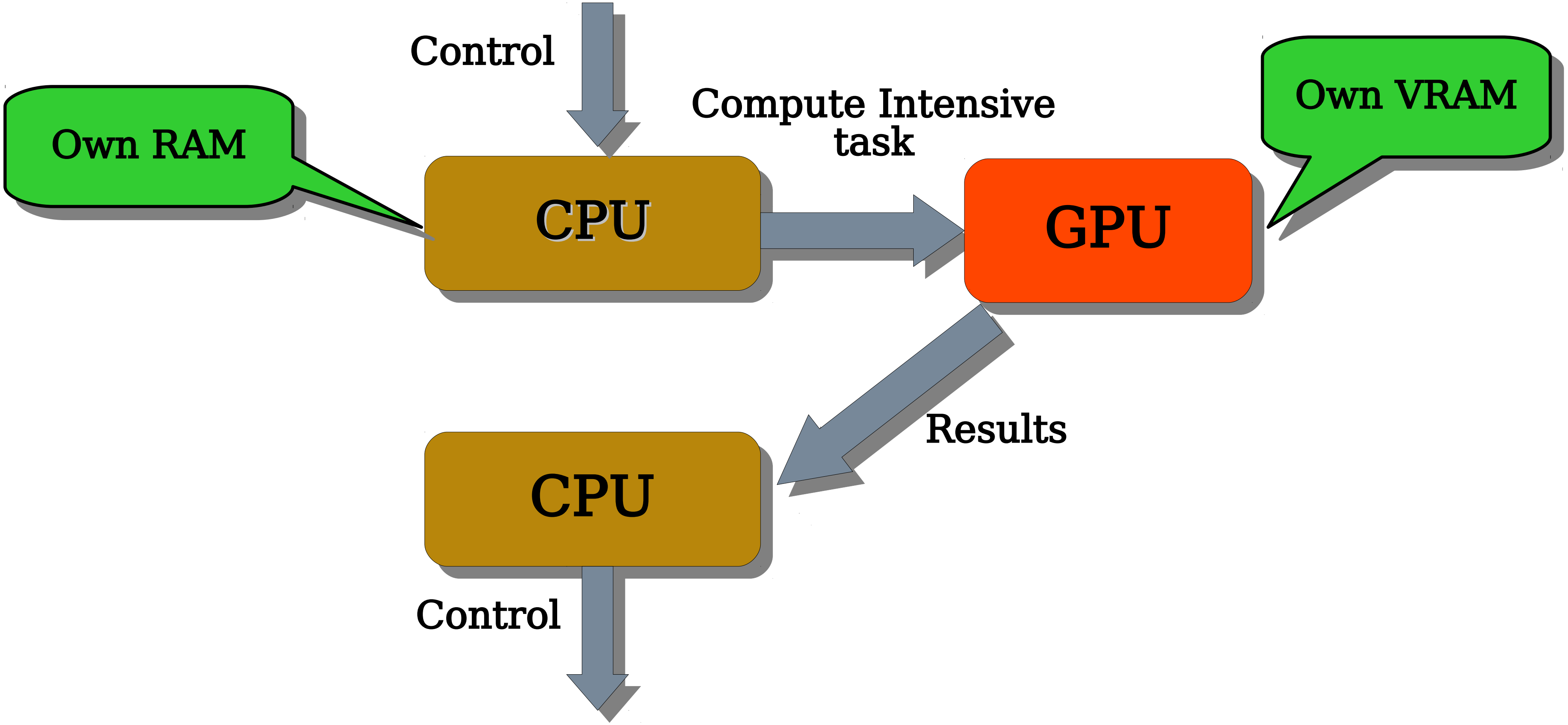
1. Include libraries
2. Declare variables
3. Allocate memory to the variables
4. Initialize variables
5. Perform computations
6. Store results
7. Free varables

GPU is a coprocessor/ accelerator

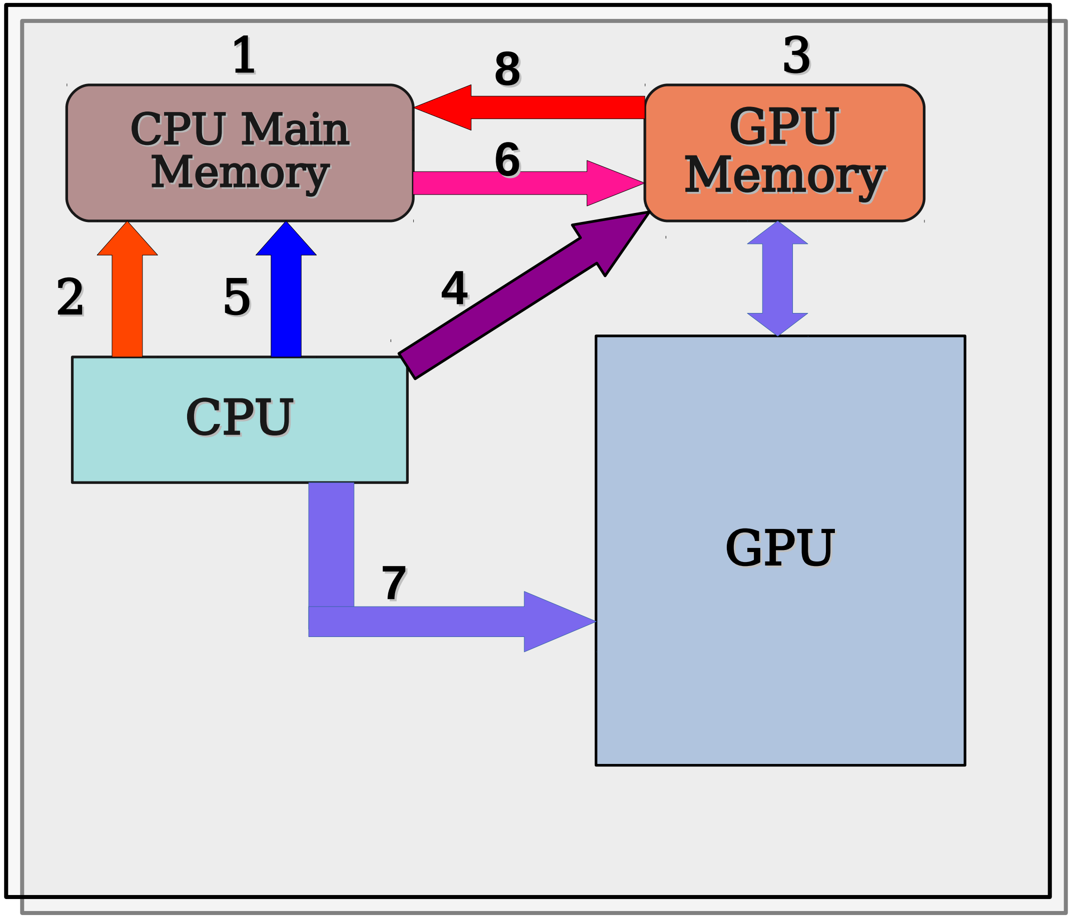
Only computationally intensive jobs are diverted towards GPU

GPU is a coprocessor/ accelerator

Only computationally intensive jobs are diverted towards GPU



GPU Program Execution Model



1. Declare CPU variables
2. Allocate memory to CPU variables
3. Declare GPU variables
4. Allocate memory to GPU variables
5. Initialize data in CPU memory
6. Copy data from CPU memory to GPU memory
7. CPU instruct to GPU for parallel Execution
8. Copy results back from GPU Memory to CPU memory

9. Free both CPU & GPU memories

CUDA

- **Compute Unified Device Architecture**

  - Used to code on NVIDIA GPUs
- File name

  - file_name**.cu**
- Compile

  - **nvcc** flags file_name.cu
- Run

  ./a.out

CUDA – Thread Organization

- Thread is basic execution unit on GPU
- GPU favors SIMT instructions

  Single Instruction Multiple Threads
- **Eg**. vector addition

```
for(i = 0; i < N; i++)
{
vecC[i] = vecA[i] + vecB[i];
}
```

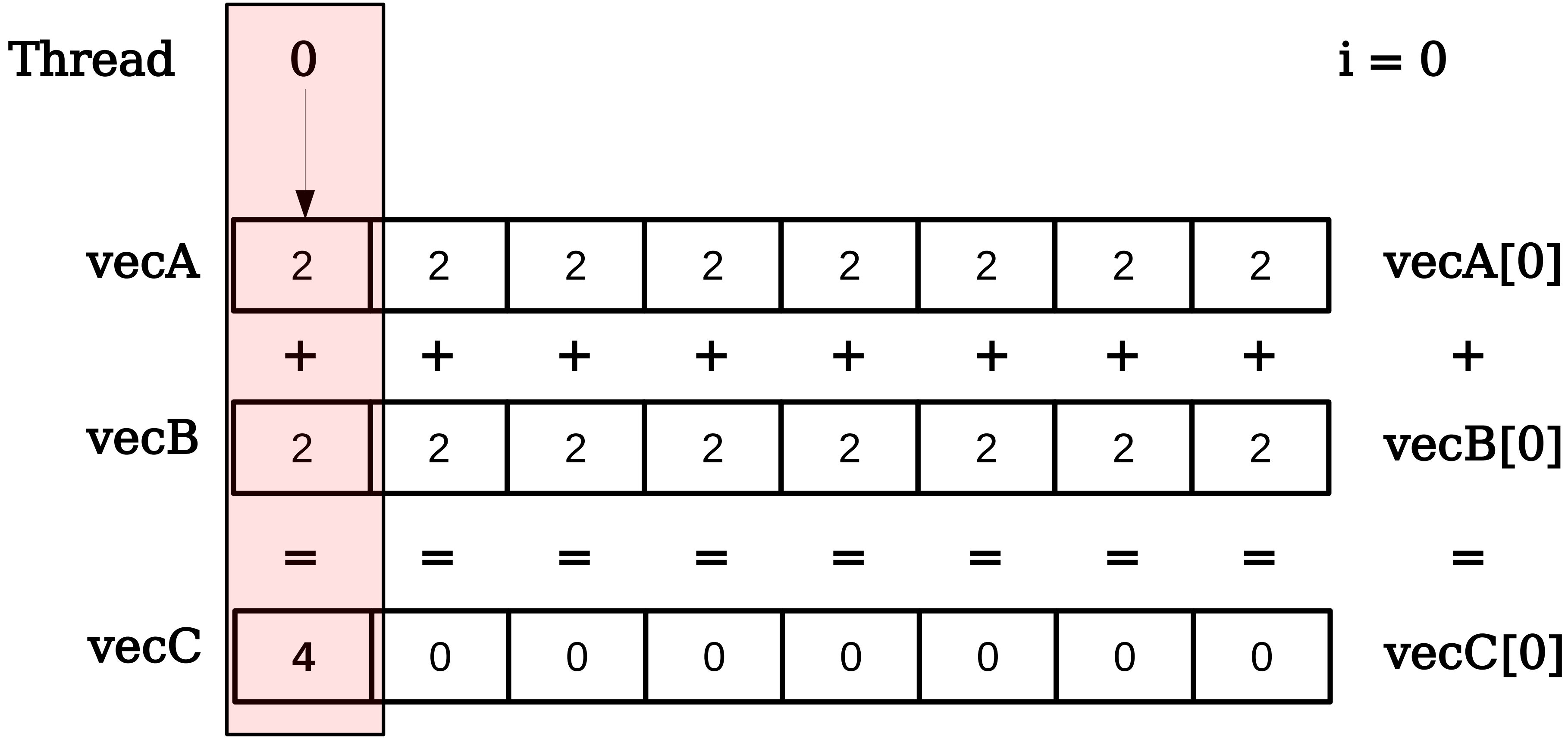
CUDA – Thread Organization

- **Sequential Execution using one thread**

vecA	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
vecC	**0**	0	0	0	0	0	0	0

CUDA – Thread Organization

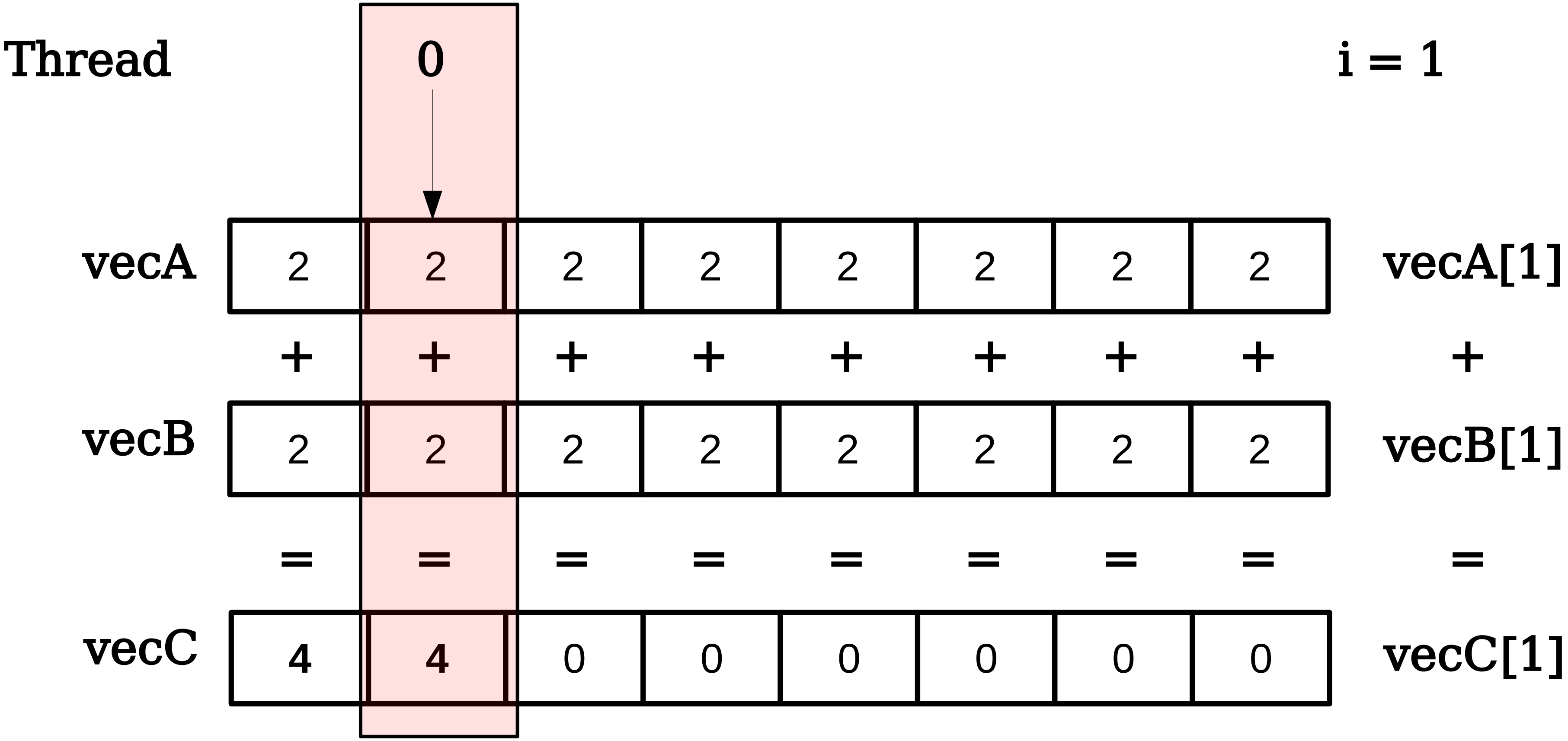
- **Sequential Execution using one thread**



Thread	0								i = 0
vecA	2	2	2	2	2	2	2	2	vecA[0]
	+	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2	vecB[0]
	=	=	=	=	=	=	=	=	=
vecC	4	0	0	0	0	0	0	0	vecC[0]

CUDA – Thread Organization

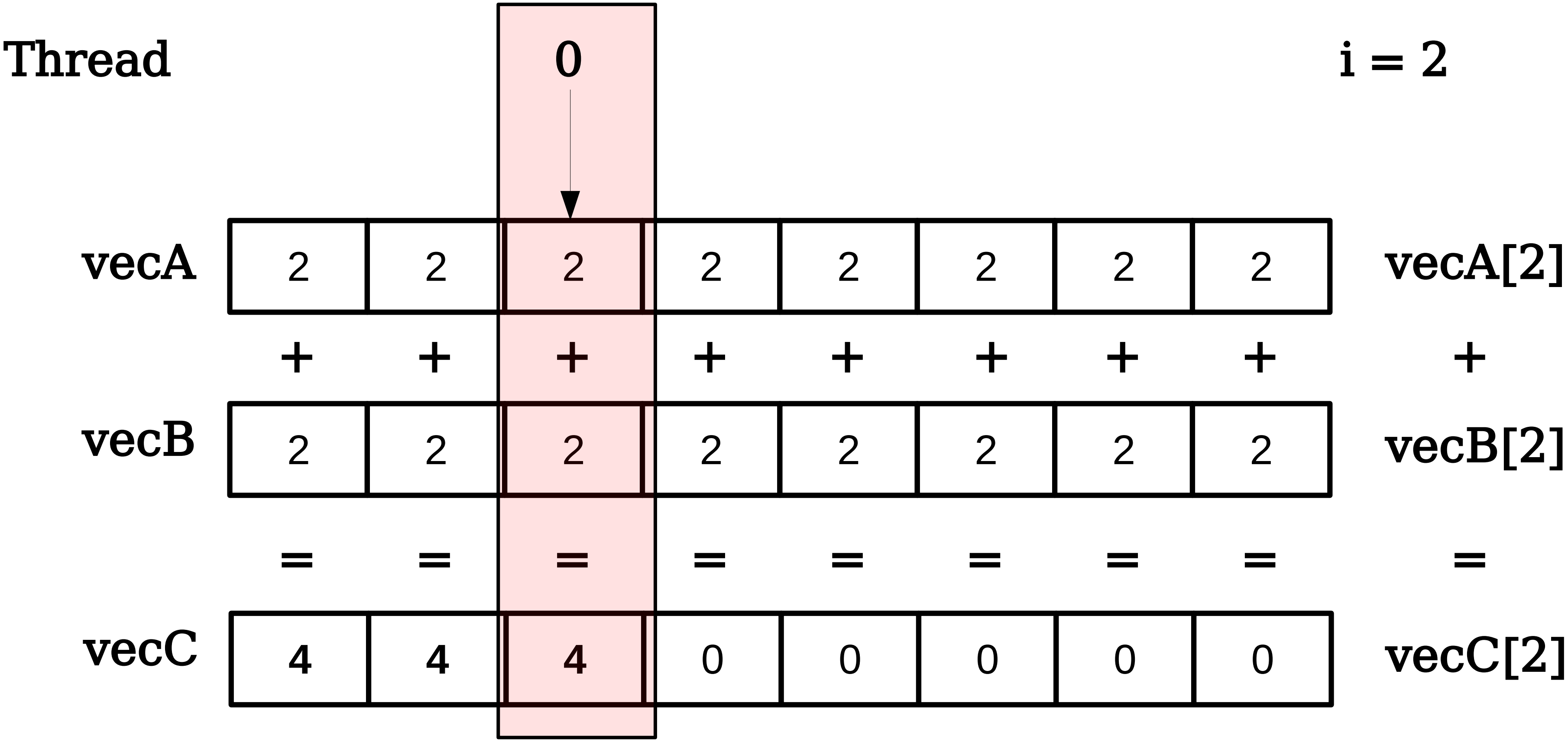
- Sequential Execution using one thread



Thread		0							i = 1
vecA	2	2	2	2	2	2	2	2	vecA[1]
	+	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2	vecB[1]
	=	=	=	=	=	=	=	=	=
vecC	4	4	0	0	0	0	0	0	vecC[1]

CUDA – Thread Organization

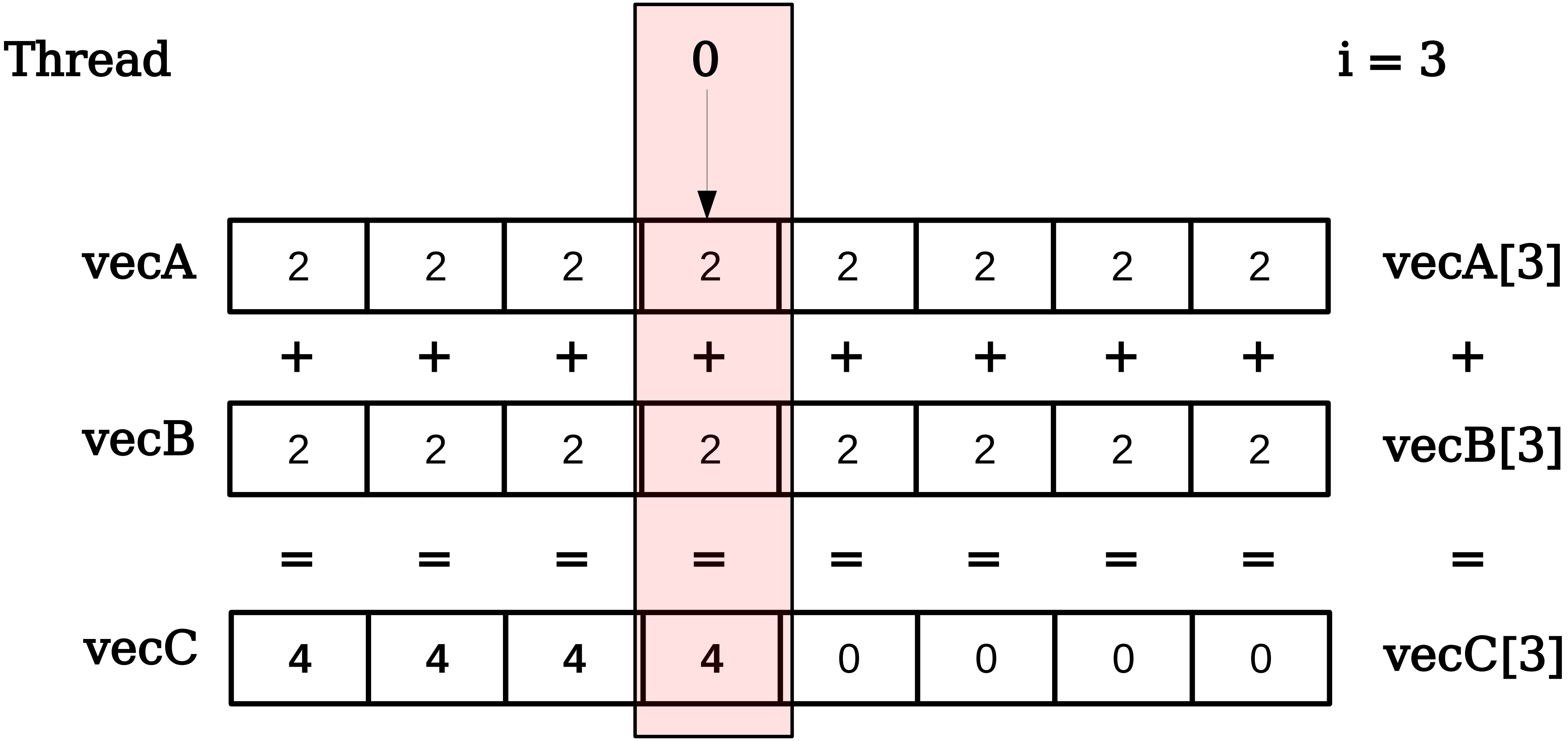
- **Sequential Execution using one thread**



vecA	2	2	2	2	2	2	2	2	vecA[2]
	+	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2	vecB[2]
	=	=	=	=	=	=	=	=	=
vecC	4	4	4	0	0	0	0	0	vecC[2]

CUDA – Thread Organization

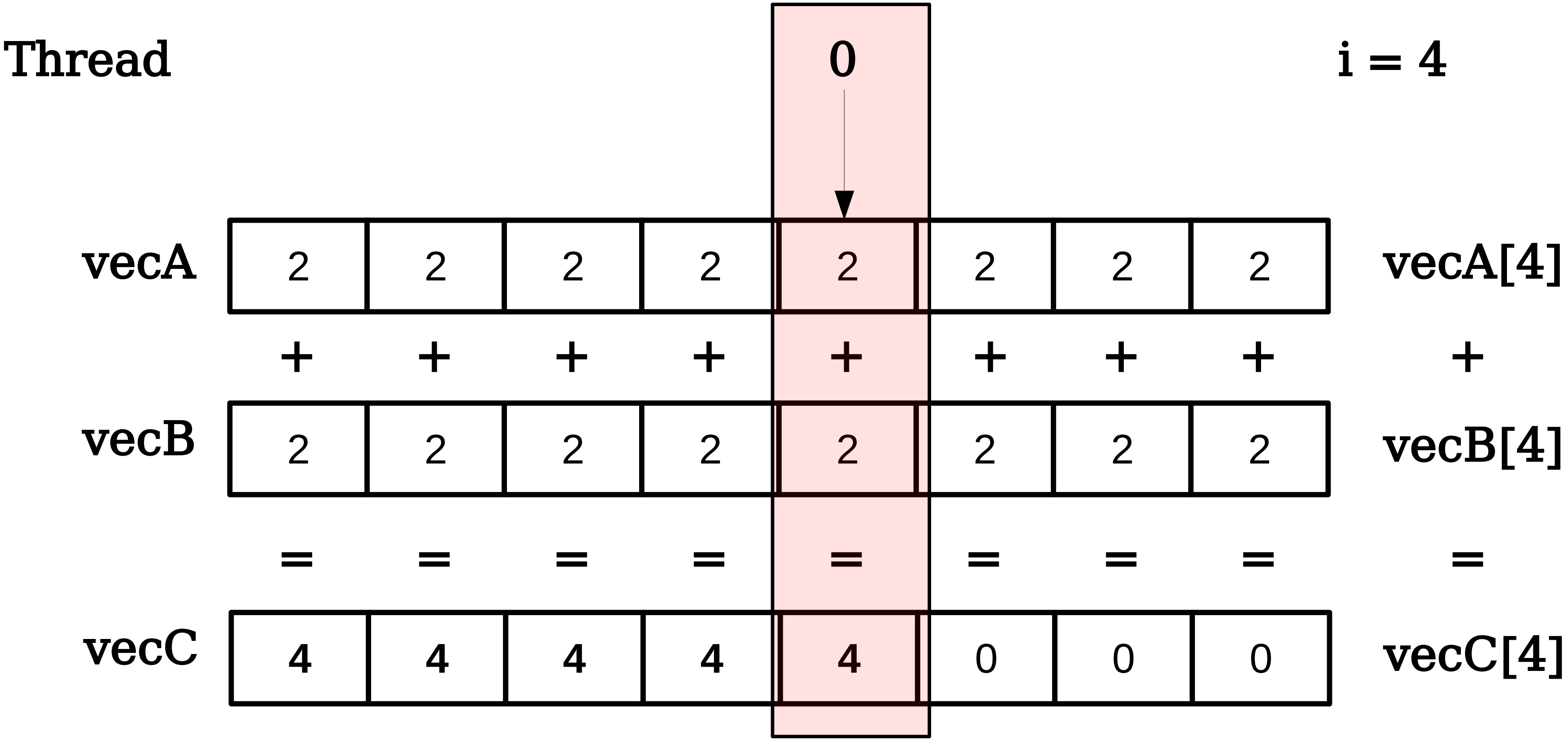
- Sequential Execution using one thread



Thread				0					i = 3
vecA	2	2	2	2	2	2	2	2	vecA[3]
	+	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2	vecB[3]
	=	=	=	=	=	=	=	=	=
vecC	4	4	4	4	0	0	0	0	vecC[3]

CUDA – Thread Organization

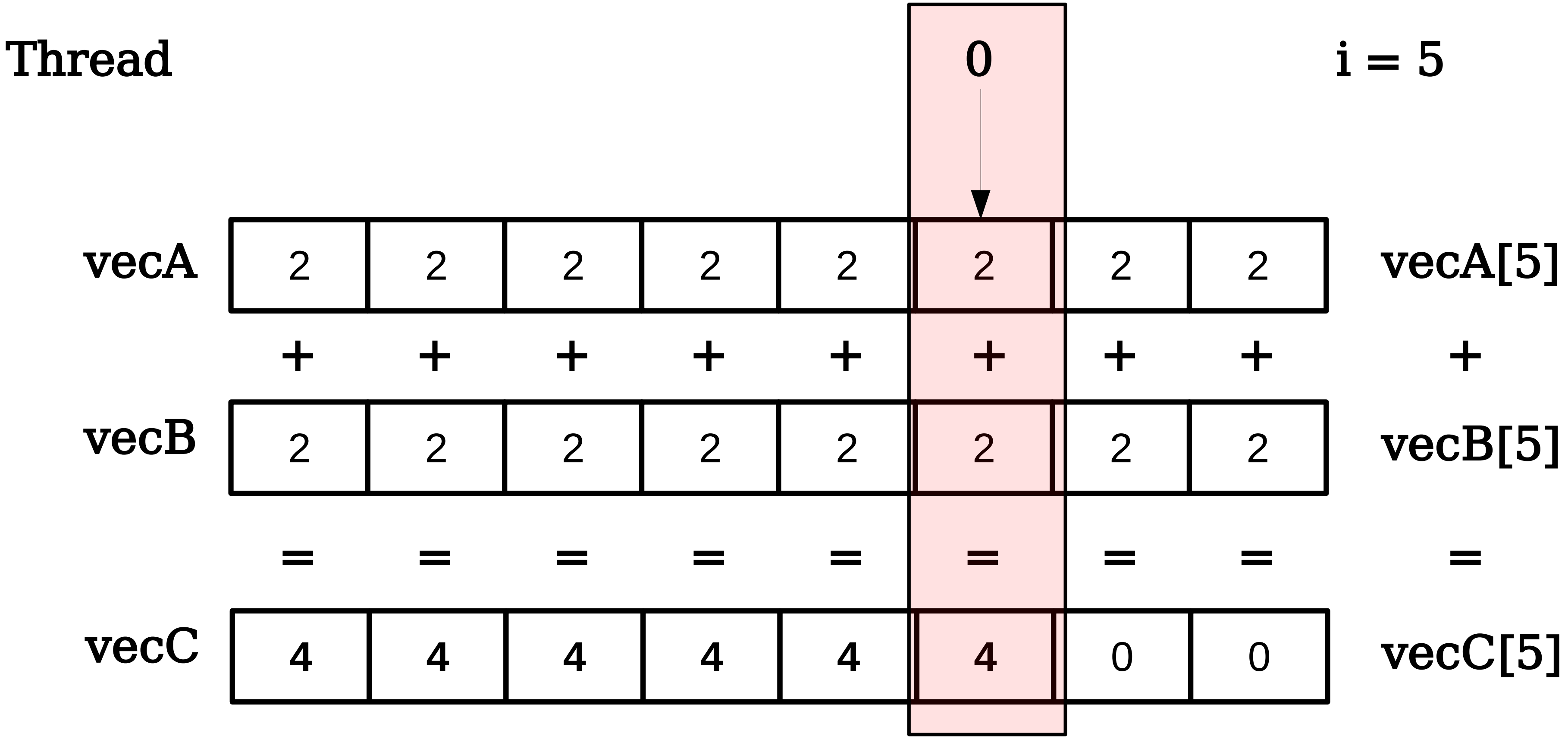
- **Sequential Execution using one thread**



Thread					0				i = 4
vecA	2	2	2	2	2	2	2	2	vecA[4]
	+	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2	vecB[4]
	=	=	=	=	=	=	=	=	=
vecC	4	4	4	4	4	0	0	0	vecC[4]

CUDA – Thread Organization

- Sequential Execution using one thread



Thread

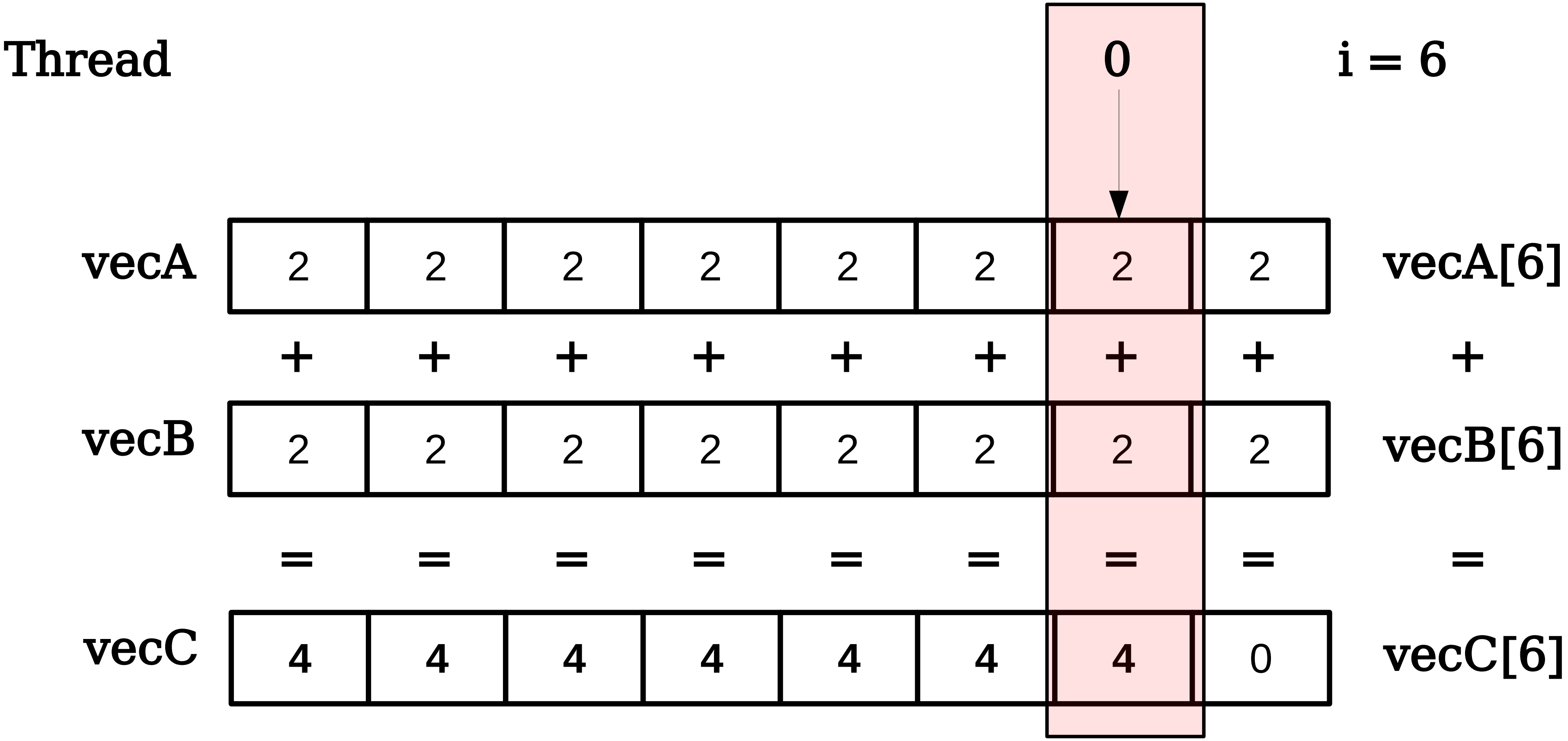
0

i = 5

vecA	2	2	2	2	2	2	2	2	vecA[5]
	+	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2	vecB[5]
	=	=	=	=	=	=	=	=	=
vecC	4	4	4	4	4	4	0	0	vecC[5]

CUDA – Thread Organization

- **Sequential Execution using one thread**

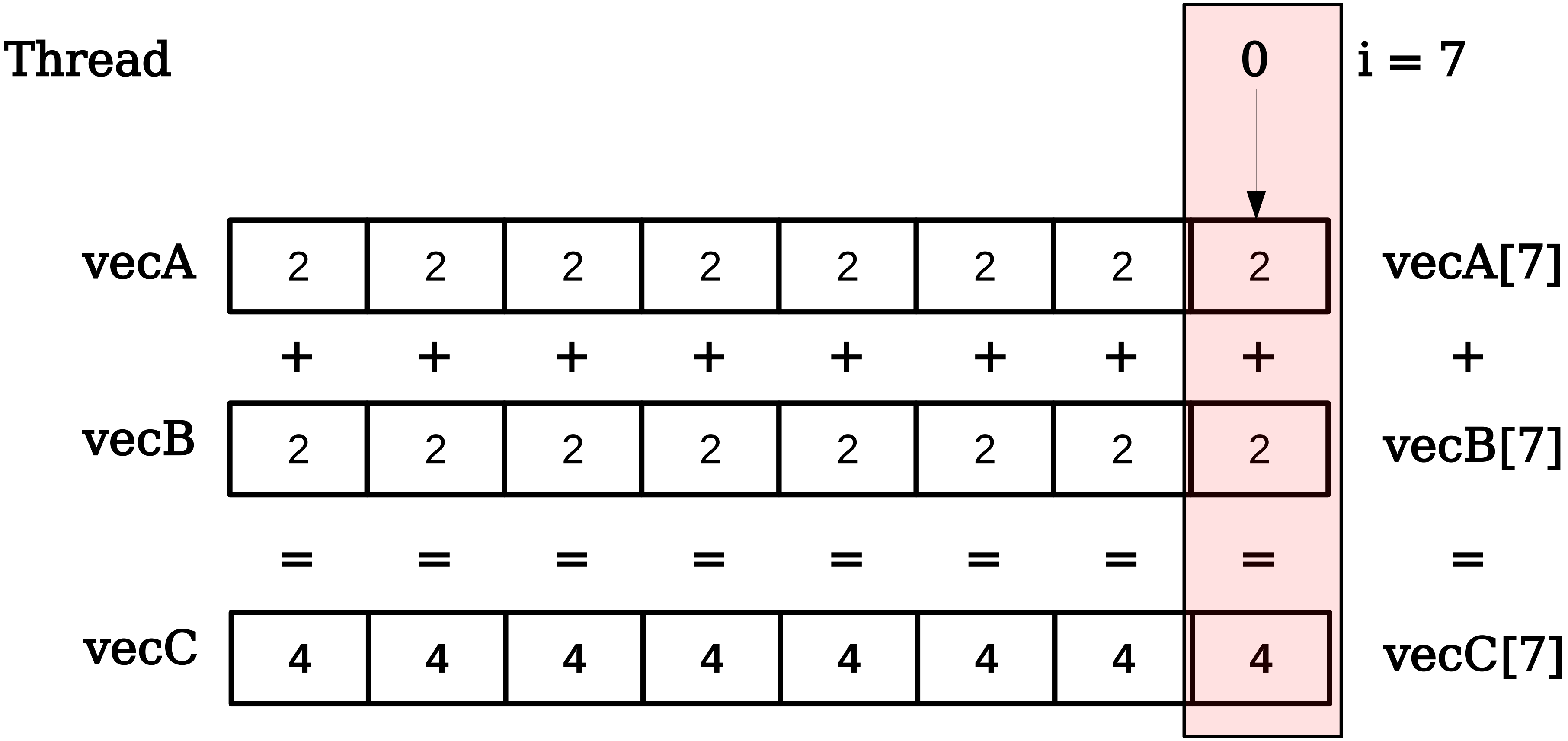


Thread 0 i = 6

vecA	2	2	2	2	2	2	2	2	vecA[6]
	+	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2	vecB[6]
	=	=	=	=	=	=	=	=	=
vecC	4	4	4	4	4	4	4	0	vecC[6]

CUDA – Thread Organization

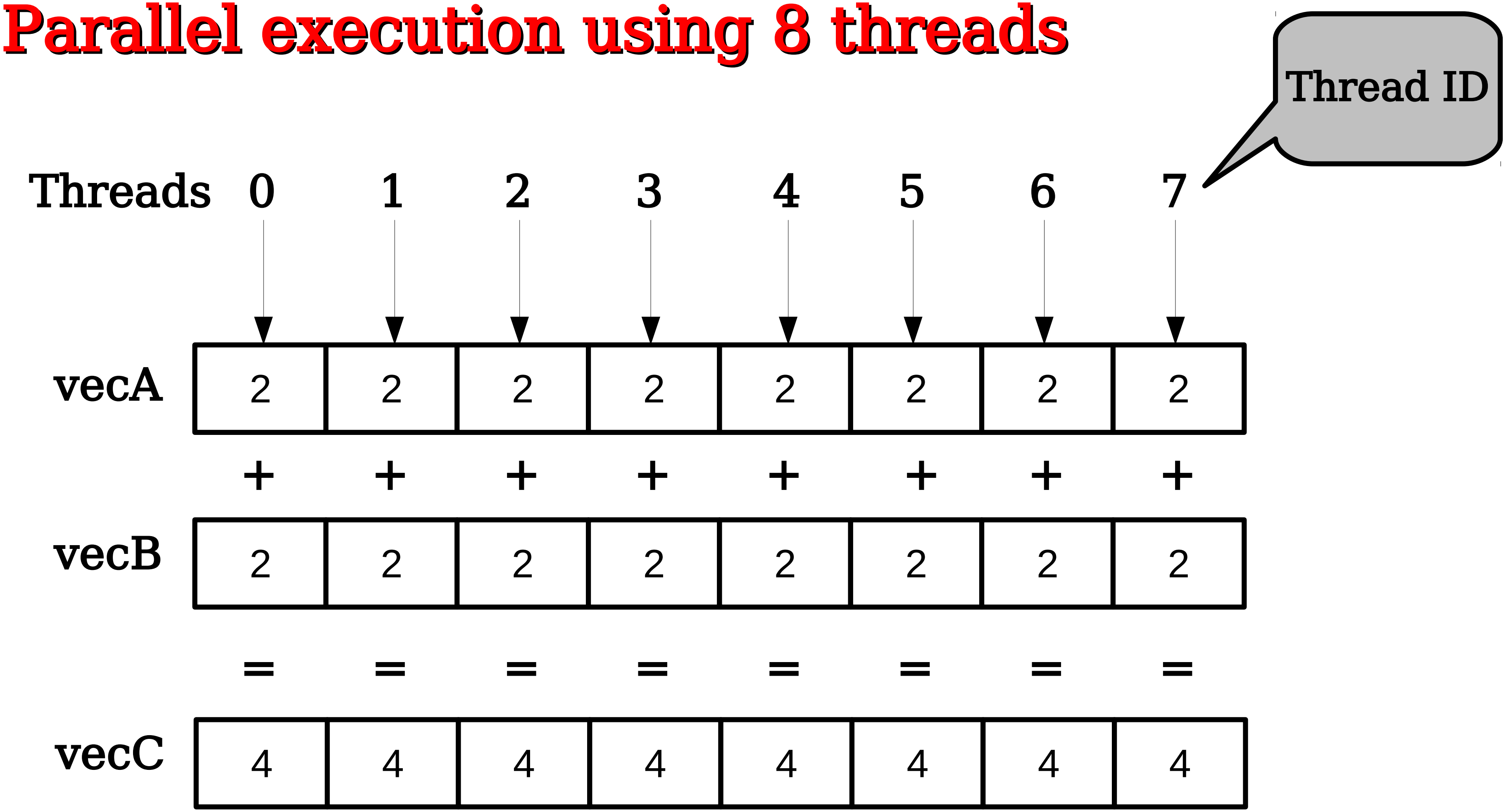
- **Sequential Execution using one thread**



Thread								0	i = 7
vecA	2	2	2	2	2	2	2	2	vecA[7]
	+	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2	vecB[7]
	=	=	=	=	=	=	=	=	=
vecC	4	4	4	4	4	4	4	4	vecC[7]

CUDA – Thread Organization

- **Parallel execution using 8 threads**

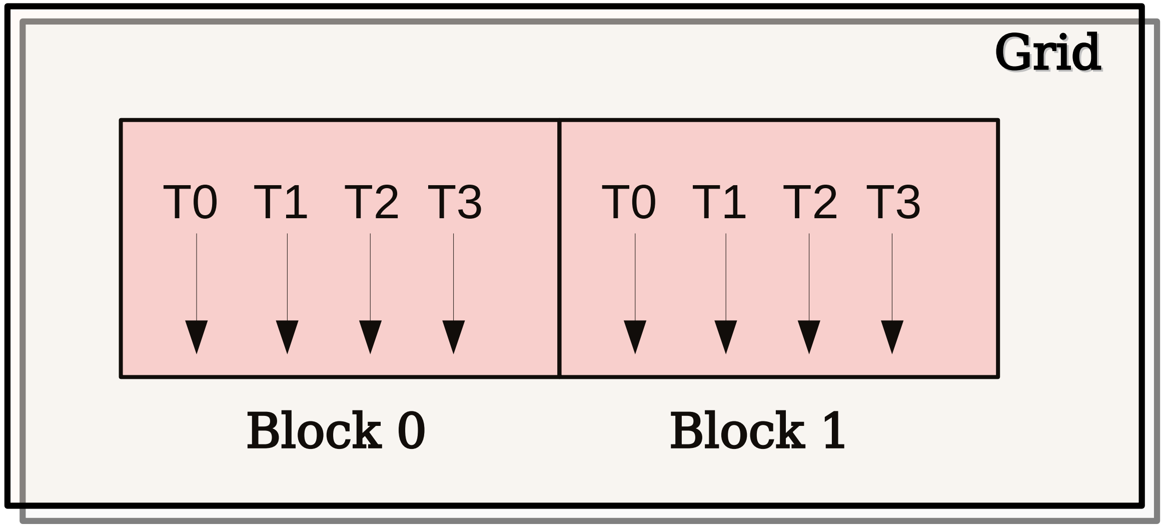


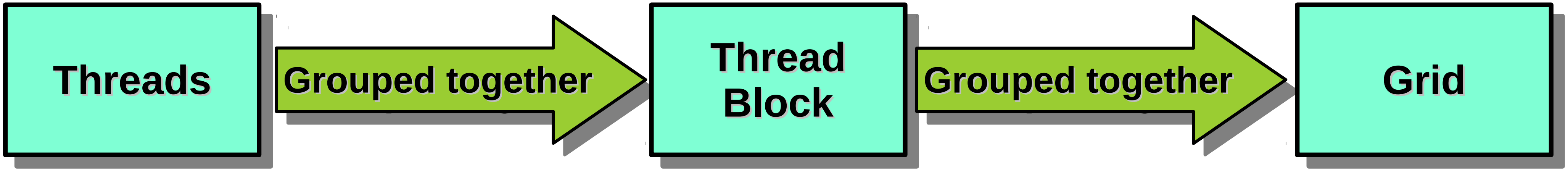
Thread ID

Threads	0	1	2	3	4	5	6	7
vecA	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
vecC	4	4	4	4	4	4	4	4

CUDA – Thread Organization

- Unlike **OpenMP**, threads are organized in **CUDA** in different way





Threads

Grouped together

Thread Block

Grouped together

Grid

CUDA – Thread Organization

- **Every thread has**

  1. Thread Id – retrieved by threadIdx.**x**

  -Unique Thread ID (number) but local to block

  2. Block Id - retrieved by blockIdx.**x**

  -Unique Block ID (number)

  3. Block Dimensions - retrieved by blockDim.**x**

  - Number of threads per block

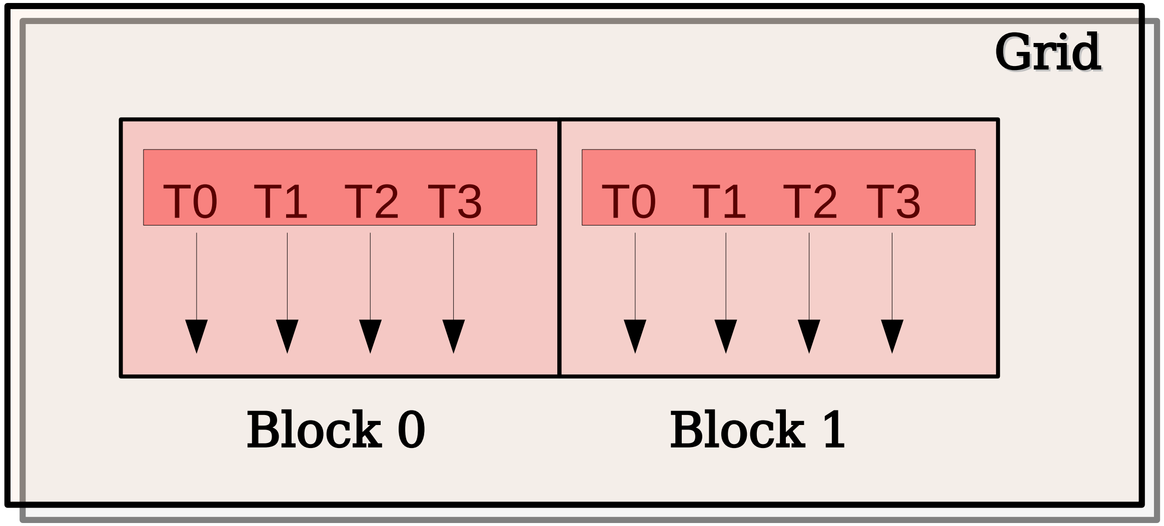
  4. Grid Dimensions - retrieved by GridDim.**x**

  - Number of blocks in grid

CUDA – Thread Organization

- **Every thread has**

1. Thread Id- Gives thread number local to thread block

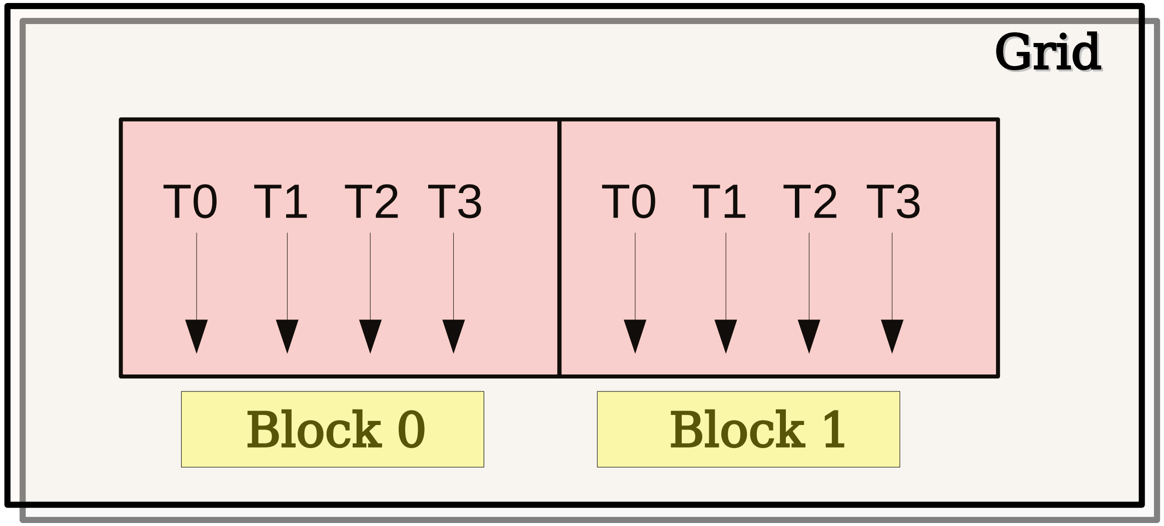


Observe: thread Ids T0, T1, T2 & T3 are numbered in both thread Blocks

CUDA – Thread Organization

- **Every thread has**

  2. Block Id- Gives block number

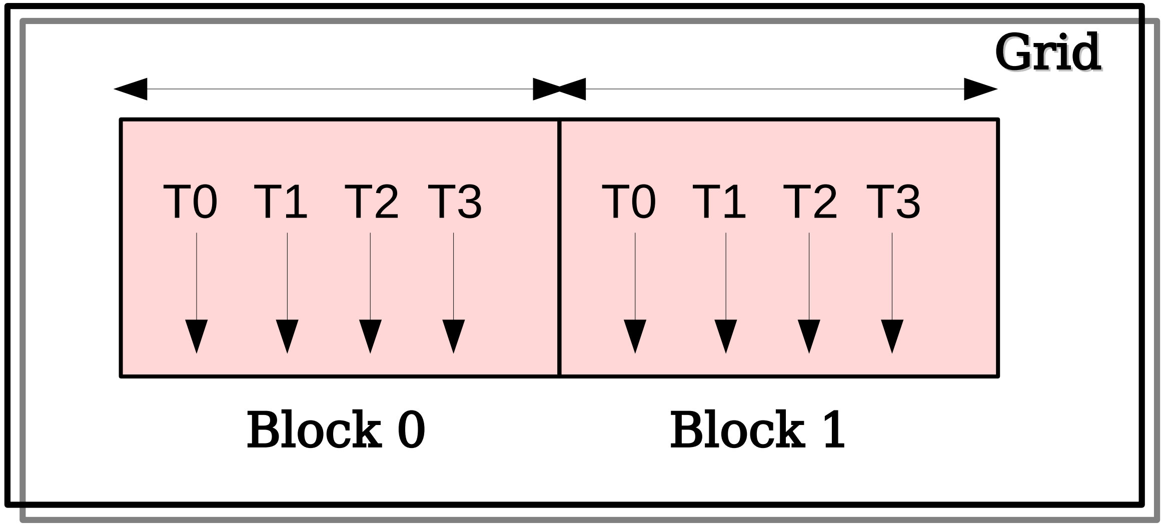


Observe: Block indices also starts with 0

CUDA – Thread Organization

- **Every thread has**

3. Block Dimension- Gives number of threads in a block



Observe: **Each block has 4 threads.**
Each thread block has same number of threads

CUDA – Thread Organization

- Write helloGPU.cu program
- Function executed on GPU is called as kernel
- While calling fuction, thread configuration is specified

Program references

1. 1_helloGPU.cu
2. 1_helloGPU_ThreadOrganization.cu

Case Study – 1
Vector addition in CUDA

Vector addition program

- **Declare variables on CPU and GPU**

```
int *h_a, *h_b, *h_c; // CPU varaibles
int *d_a, *d_b, *d_c; // GPU varaibles
```

- **Allocate memory for CPU variables**

```
h_a = (int *)malloc(N*sizeof(int));
h_b = (int *)malloc(N*sizeof(int));
h_c = (int *)malloc(N*sizeof(int));
```

Vector addition program

- **Initialize data on CPU**

```
for(i = 0; i < N; i++)
{
  h_a[i] = 2;
  h_b[i] = 2;
  h_c[i] = 0;
}
```

Vector addition program

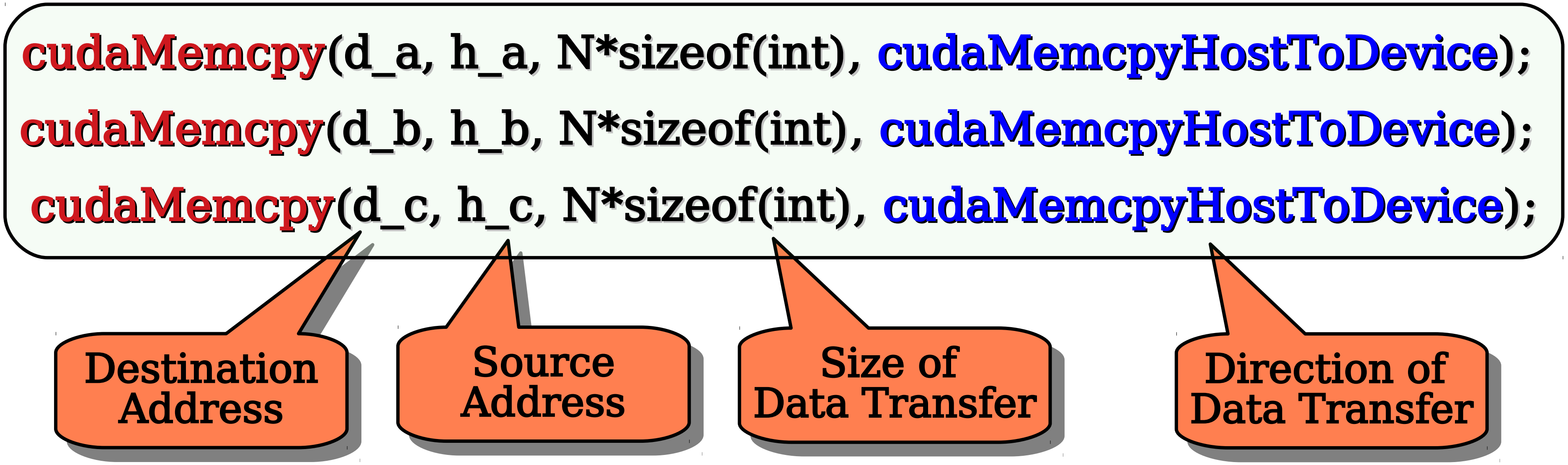
- **Allocate memory for GPU varaibles**

```
cudaMalloc((void **)&device_variable, size);
```

```
cudaMalloc((void **)&d_a, N*sizeof(int));
cudaMalloc((void **)&d_b, N*sizeof(int));
cudaMalloc((void **)&d_c, N*sizeof(int));
```

Vector addition program

- **Data transfer Host to Device**



```
cudaMemcpy(d_a, h_a, N*sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(d_b, h_b, N*sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(d_c, h_c, N*sizeof(int), cudaMemcpyHostToDevice);
```

Destination Address

Source Address

Size of Data Transfer

Direction of Data Transfer

Vector addition program

- **Kernel Launch on device**

- **vecAdd<<<numB, numT >>>(d_a, d_b, d_c, N);**

  numT = # threads per block

  numB = total # thread blocks

- **Therefore Total # threads launched = numT*numB**

Eg. If N = 128 and numT = 128

Vector addition program

Case-1: N = # threads per block
A single thread block is sufficient

```
__global__ void vecAdd_kernel(int *d_a, int *d_b, int *d_c, int N){
   int tid = threadIdx.x;
    d_c[tid] = d_a[tid] + d_b[tid];
}
```

__global__ keyword indicates **device function** i.e. executed on **GPU**

threadIdx**.x** returns thread Id of each thread and it is stored in **tid**

Vector addition program

Eg: N = 8 Threads per block = 8
Number of blocks = 1

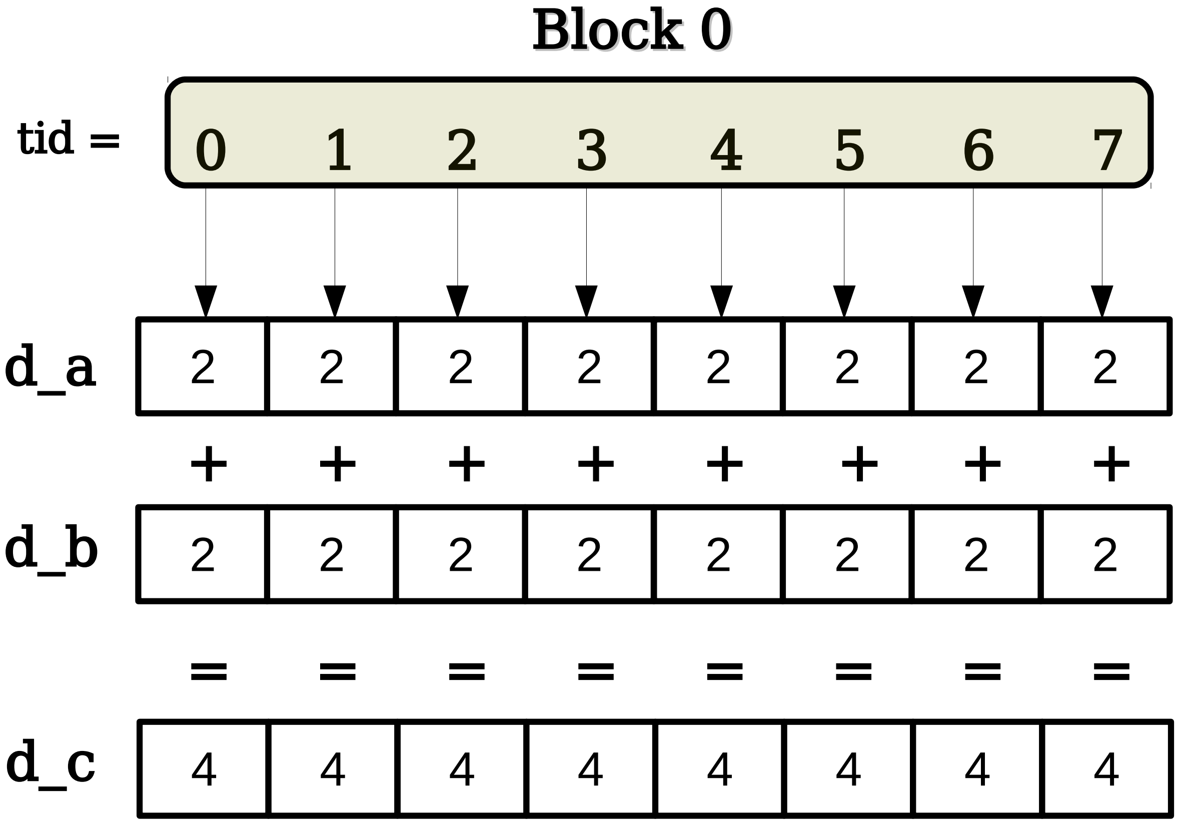
```
VecAdd<<<1,8>>>(d_a, d_b, d_c, N);
```

```
__global__ void
vecAdd_kernel(int *d_a, int
*d_b, int *d_c, int N){

  int tid = threadIdx.x;

  d_c[tid] = d_a[tid] + d_b[tid];

}
```



Block 0

tid =	0	1	2	3	4	5	6	7
d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	4	4	4	4	4	4	4	4

Vector addition program

- N = 32 then numT = 32 numB = 1

  N = 64 then numT = 64 numB = 1

  N = 1024 then numT = 1024 numB = 1

- **Problem: There is a limit on Threads Per Block! 1024**

  What if N > 1024?

- **Solution:**

  1. Use stride

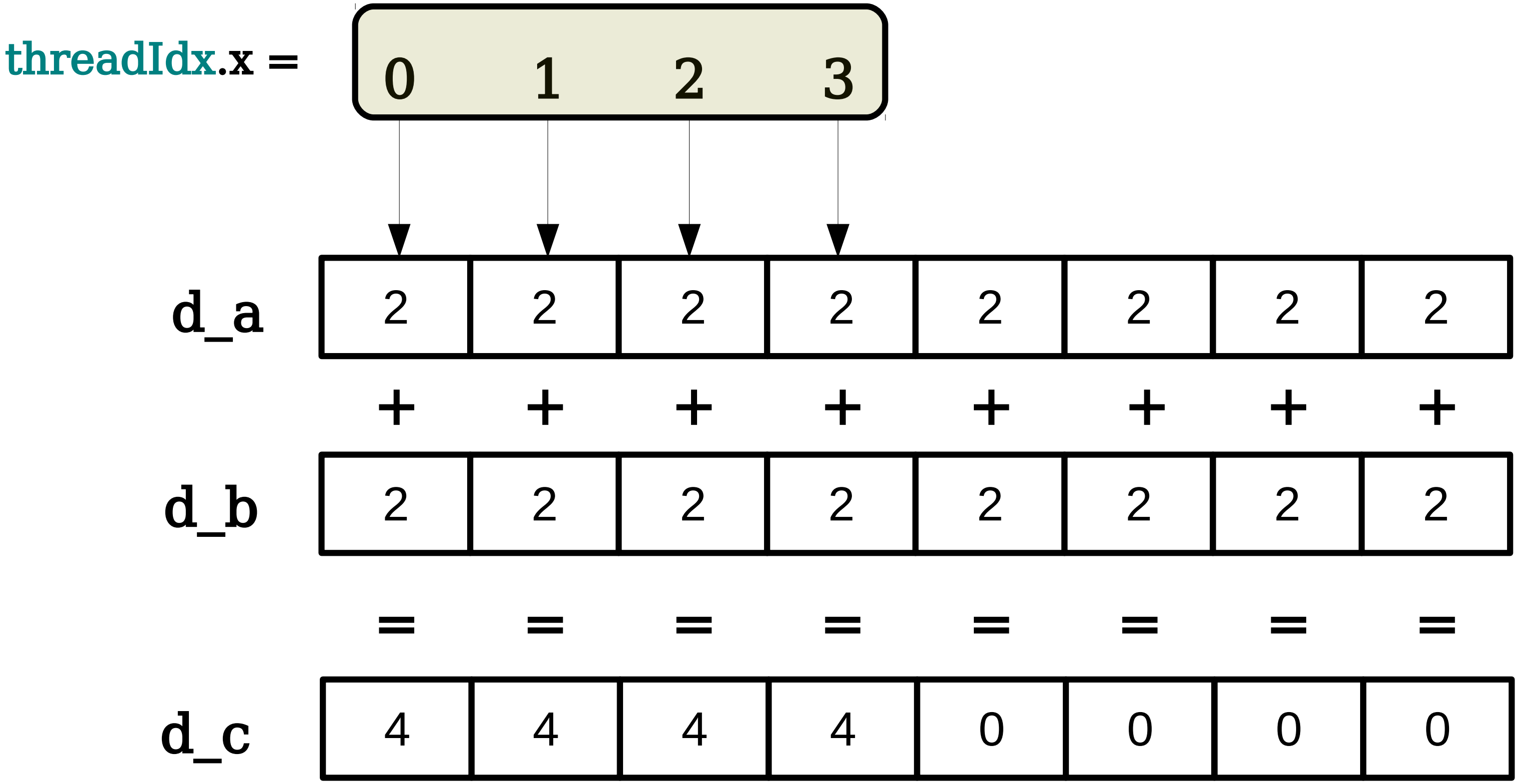
  2. Launch multiple blocks

Vector addition program

- Single thread block
- **Solution 1:** Using stride

Iteration 1

stride = 4



threadIdx.x = 0 1 2 3

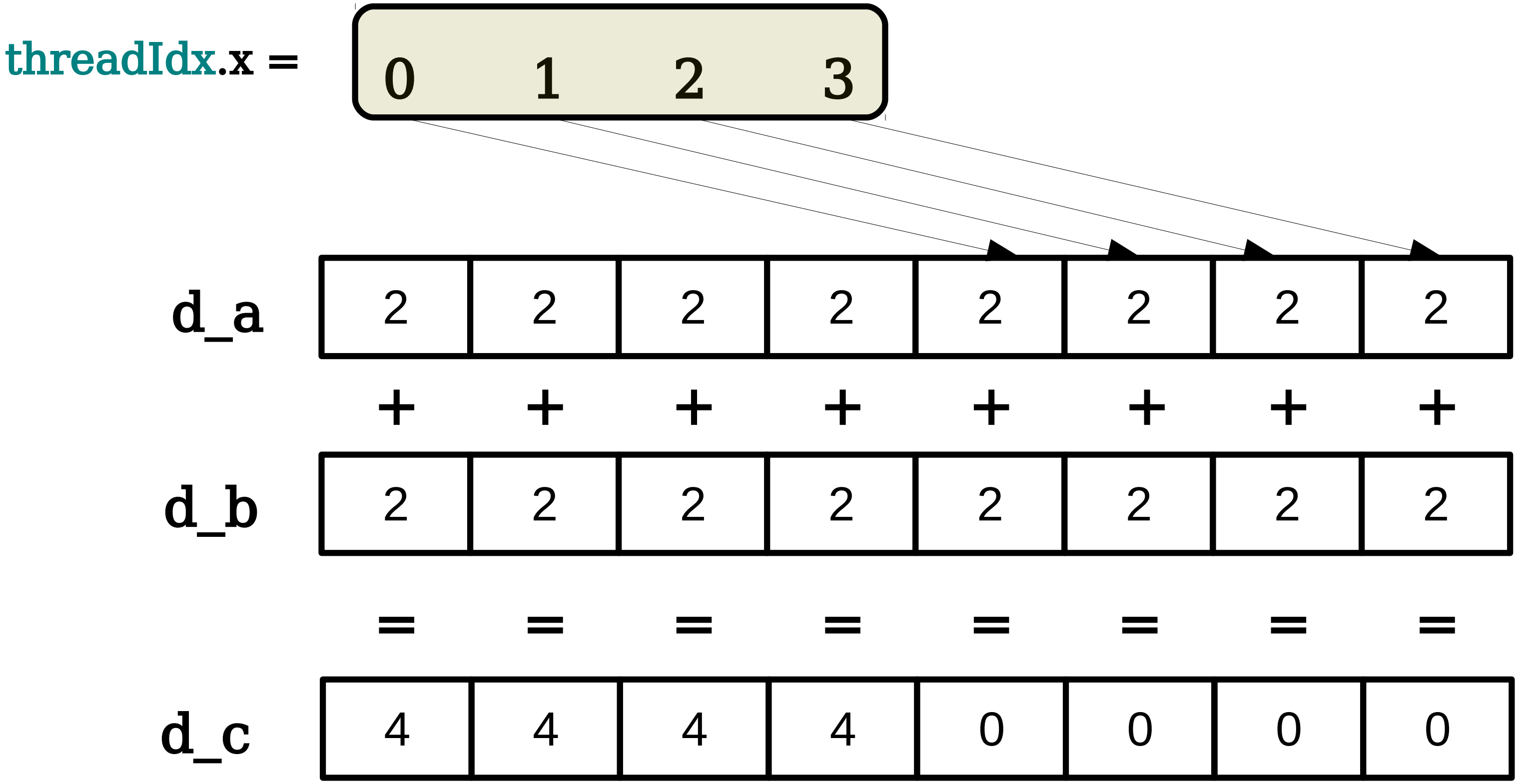
d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	4	4	4	4	0	0	0	0

Vector addition program

- Single thread block
- **Solution 1:** Using stride

Iteration 2

stride = 4



threadIdx.x = 0 1 2 3

d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	4	4	4	4	0	0	0	0

Vector addition program

- **Single thread block with stride**

```
__global__ void vecAdd_kernel(int *d_a, int *d_b, int *d_c, int N)
 {
     int tid = threadIdx.x;
     int i;
     for(i = 0; i < N; i+= blockDim.x)
     {
         d_c[i] = d_a[i] + d_b[i];
     }
}
```

Thread 0 performs operations on i = 0, 4, 8, ...

Thread 1 performs operations on i = 1, 5, 9, ...

Thread 2 performs operations on i = 2, 6, 10, ...

Thread 3 performs operations on i = 3, 7, 11, ...

Refer code

2vecAddUsingOneBlock(N>numT).cu

Vector addition program

- **Solution 2: Use multiple thread blocks**
- **Eg.** if N = 8 and numT = 4

  Then **numB = 2** for size N = 8

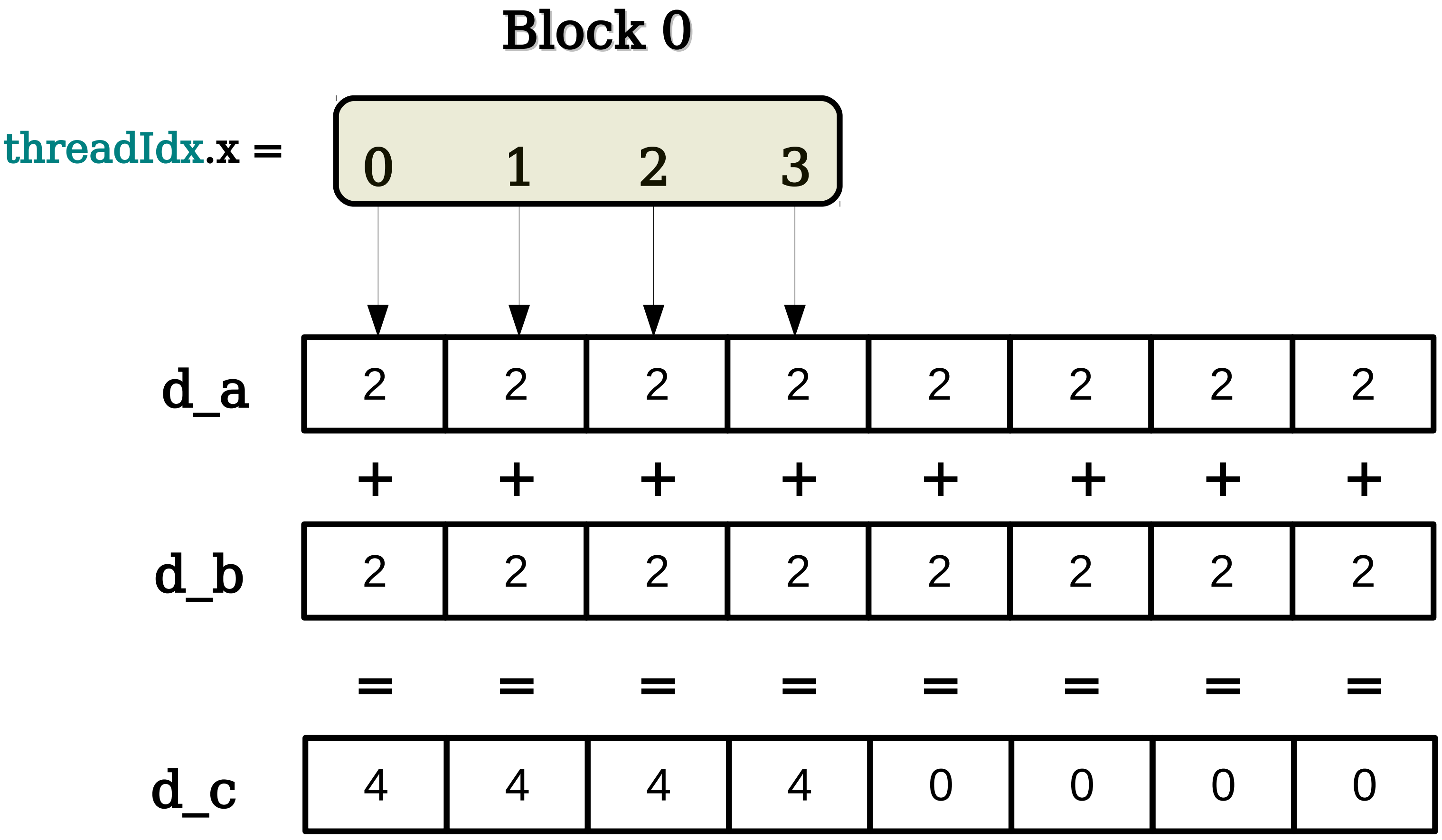
  Total number of threads = numT * numB

  = 4 * 2

  = 8

Vector addition program

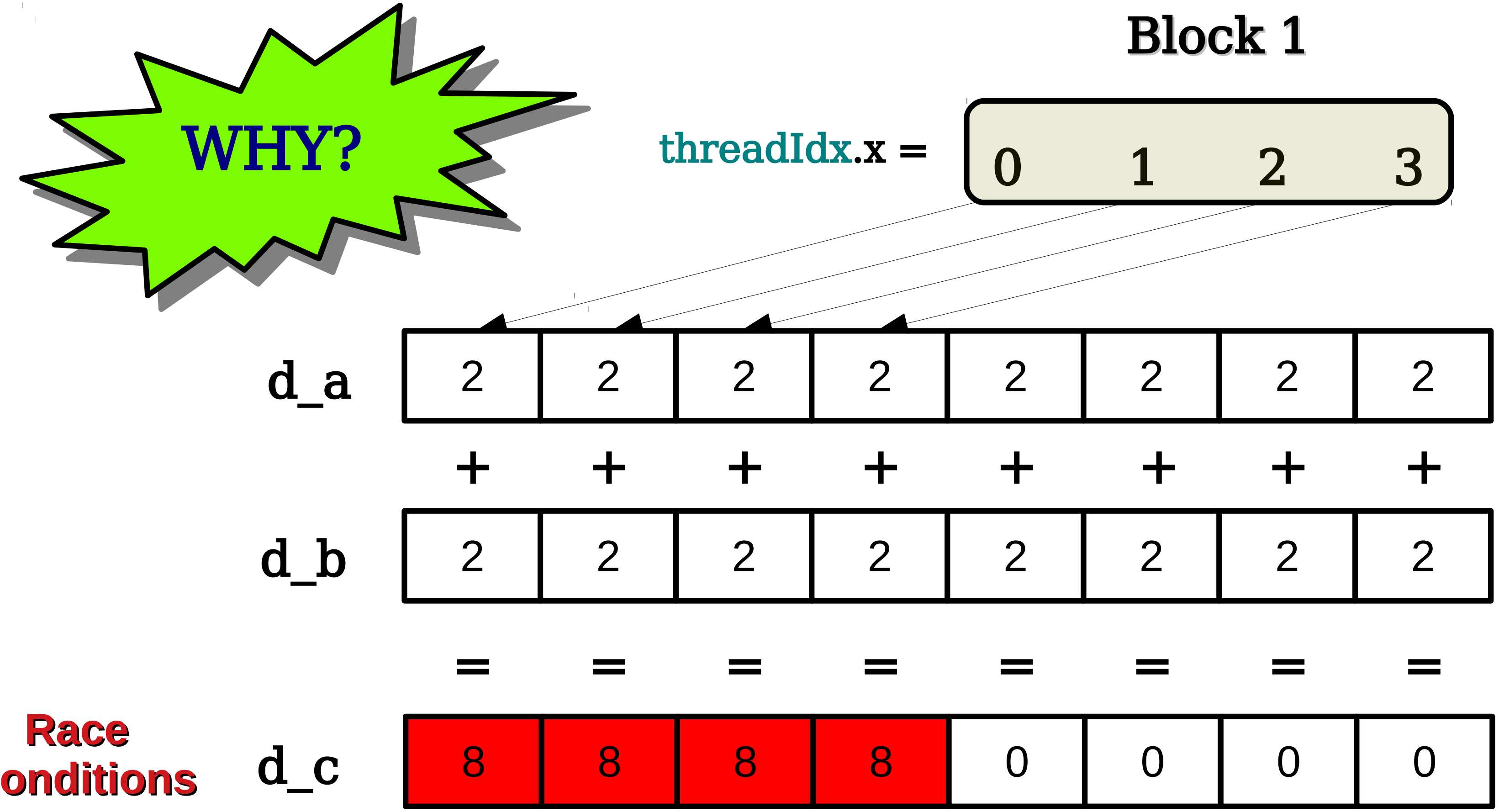
Block 0 mapping to vectors indexed from 0 to 3



d_a	2	2	2	2	2	2	2	2
d_b	2	2	2	2	2	2	2	2
d_c	4	4	4	4	0	0	0	0

Vector addition program

Block 1 also mapping to vectors indexed from 0 to 3 **WRONG!!!**



WHY?

Block 1

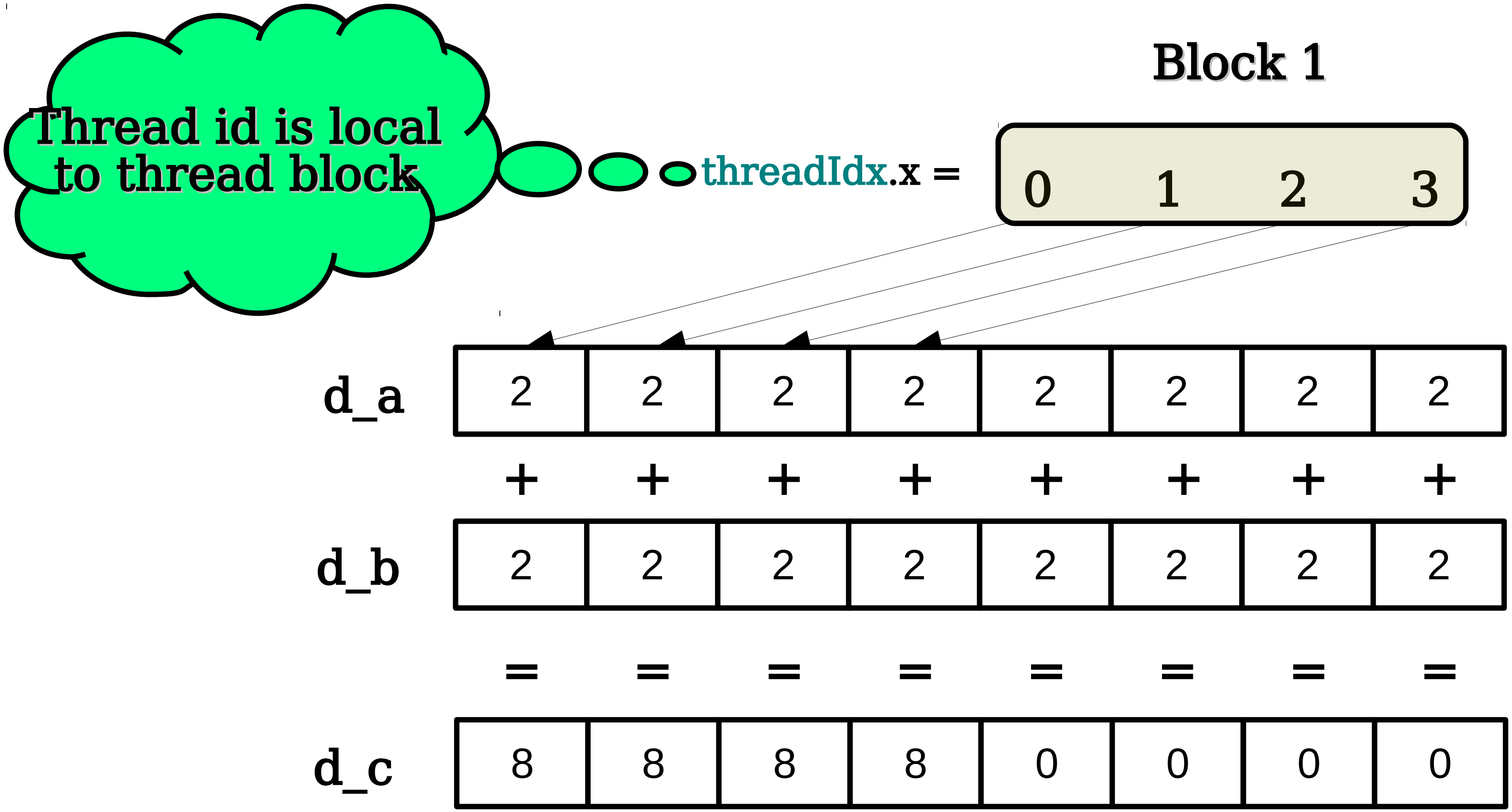
threadIdx.x = 0 1 2 3

d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	8	8	8	8	0	0	0	0

Race Conditions

Vector addition program

Block 1 also mapping to vectors indexed from 0 to 3 **WRONG!!!**



Thread id is local to thread block

threadIdx.x =

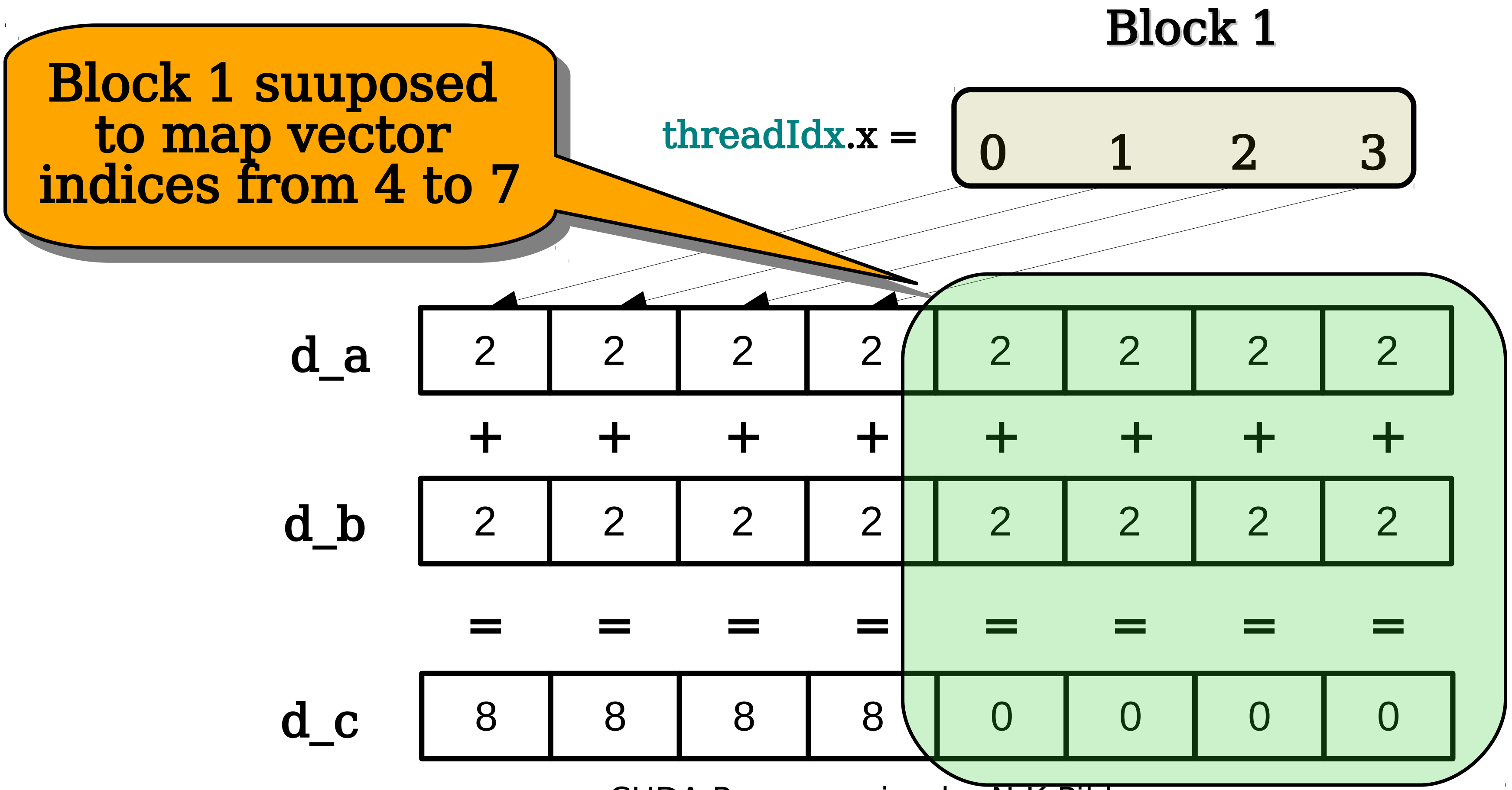
Block 1

threadIdx.x	0	1	2	3

d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	8	8	8	8	0	0	0	0

Vector addition program

Block 1 also mapping to vectors indexed from 0 to 3 **WRONG!!!**



Block 1 suuposed to map vector indices from 4 to 7

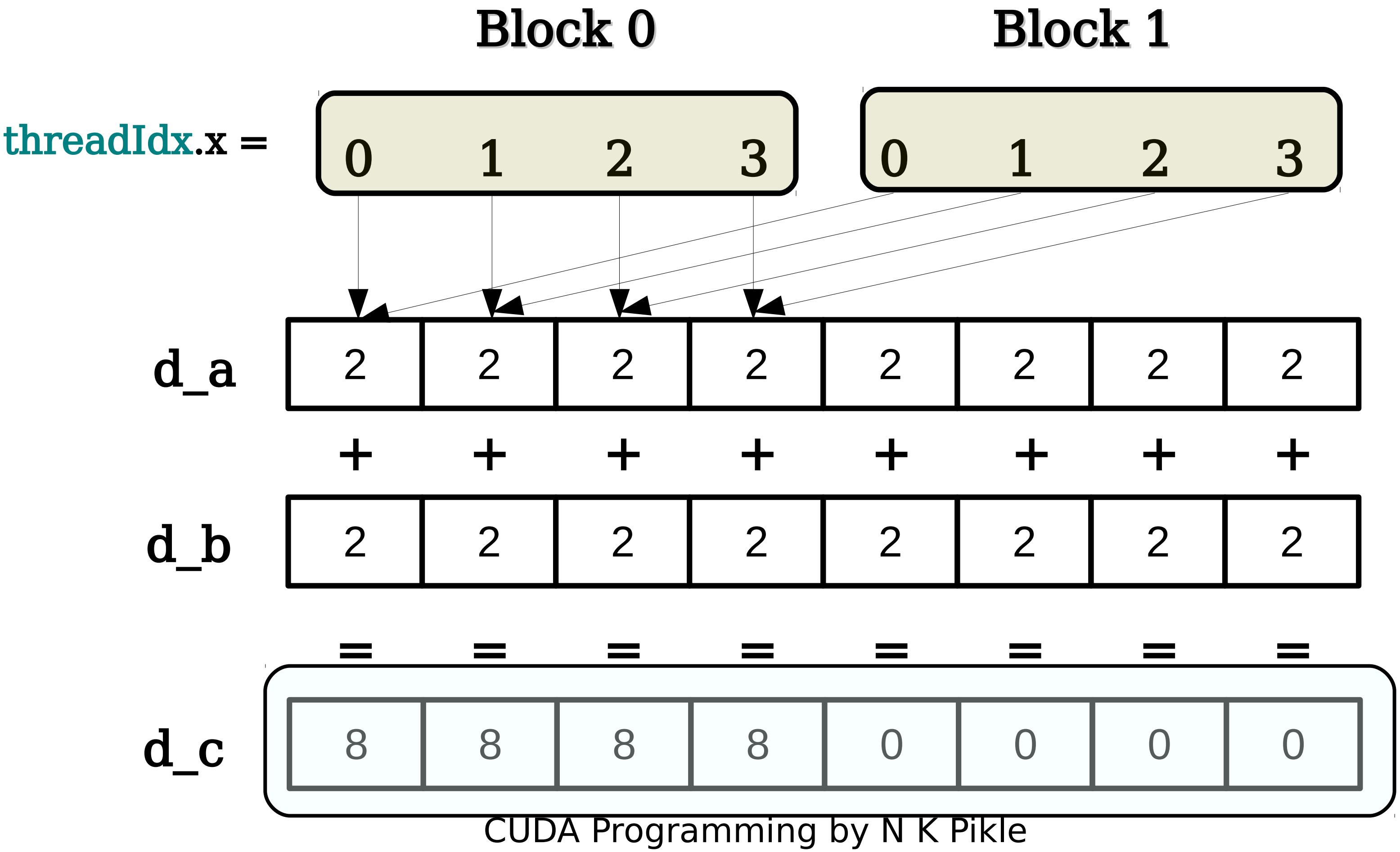
Block 1

threadIdx.x = 0 1 2 3

d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	8	8	8	8	0	0	0	0

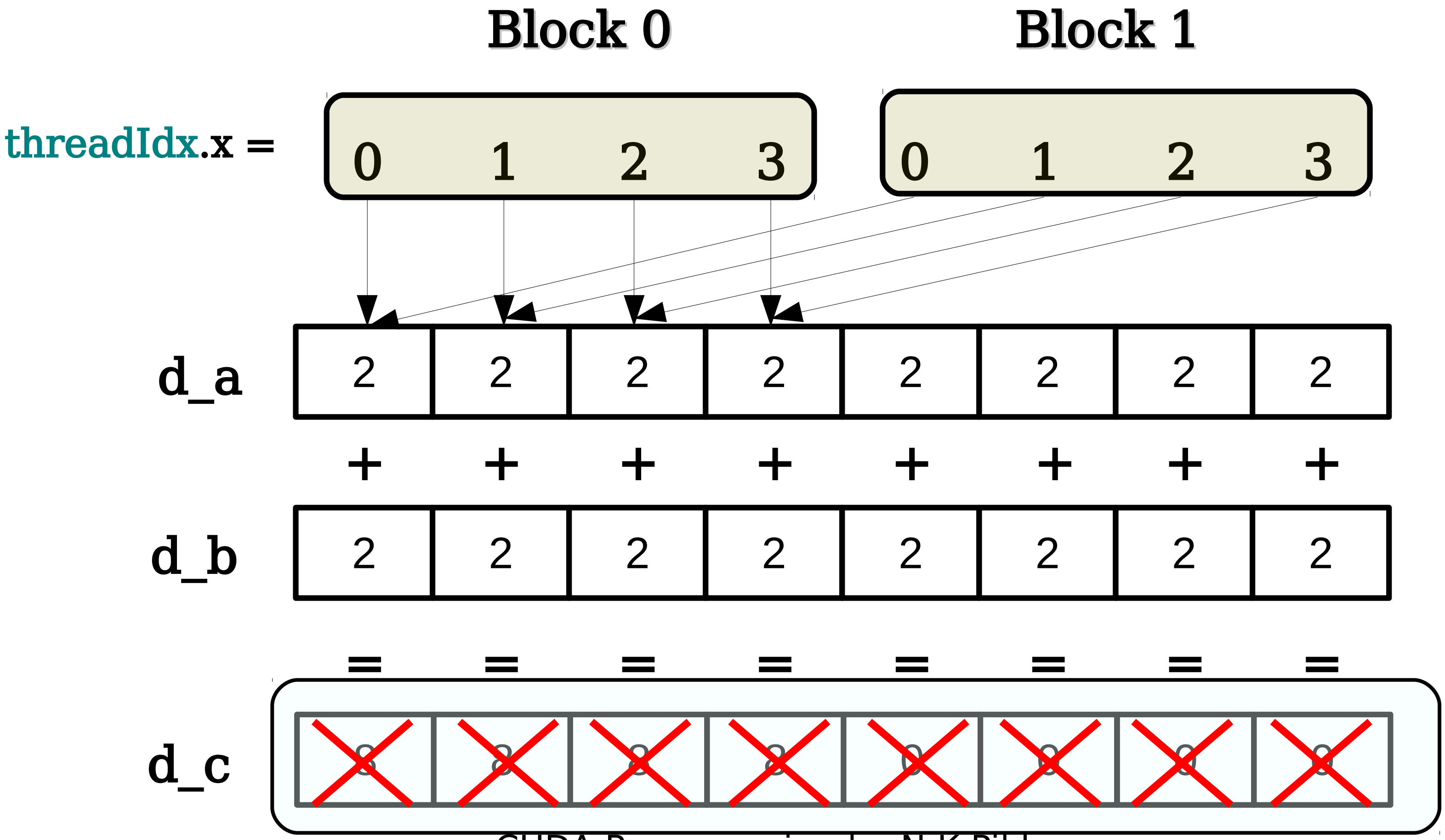
Vector addition program

Parallel Execution: Wrong answer as addition is performed Only on first 4 indices



Vector addition program

Parallel Execution: Wrong answer as addition is performed Only on first 4 indices



Solution: Compute global Thread Id

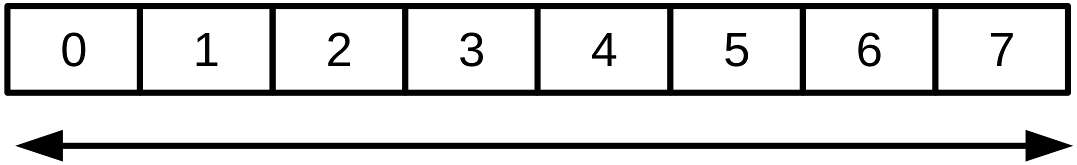
Vector addition program

- **Launching number of threads and blocks**

  Eg. if N = 8

1) A single thread block with 8 threads

   vecAdd<<<1,8>>>(d_a,d_b,d_c,N);



0	1	2	3	4	5	6	7

blockDim.x = 8

gridDim.x = 1 => Total number of thread blocks

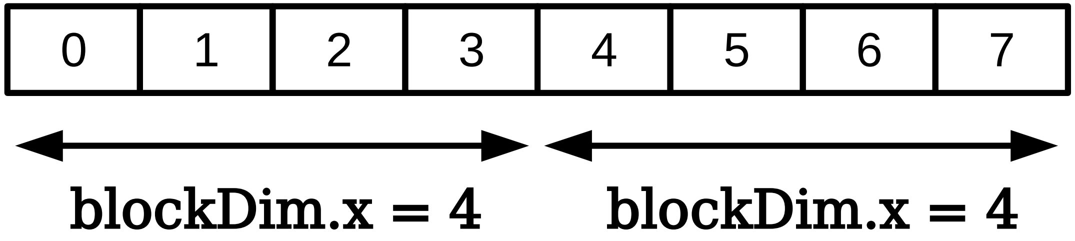
Vector addition program

- **Launching number of threads and blocks**

  Eg. if N = 8

  2) Two thread blocks with 4 threads

  vecAdd<<<2,4>>>(d_a,d_b,d_c,N);



0	1	2	3	4	5	6	7

blockDim.x = 4 **blockDim.x = 4**

gridDim.x = 2 => Total number of thread blocks

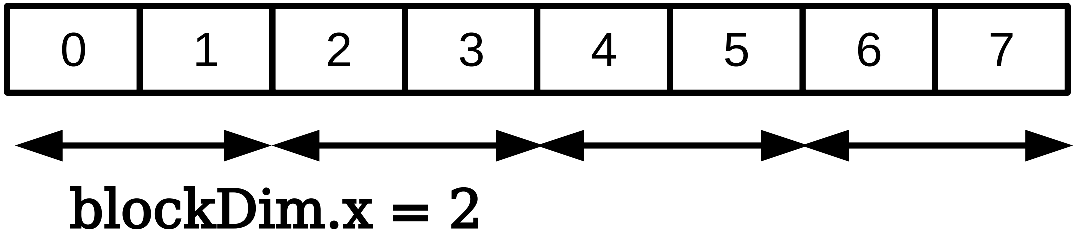
Vector addition program

- **Launching number of threads and blocks**

  Eg. if N = 8

  3) Four thread blocks with 2 threads

  vecAdd<<<4,2>>>(d_a,d_b,d_c,N);



0	1	2	3	4	5	6	7

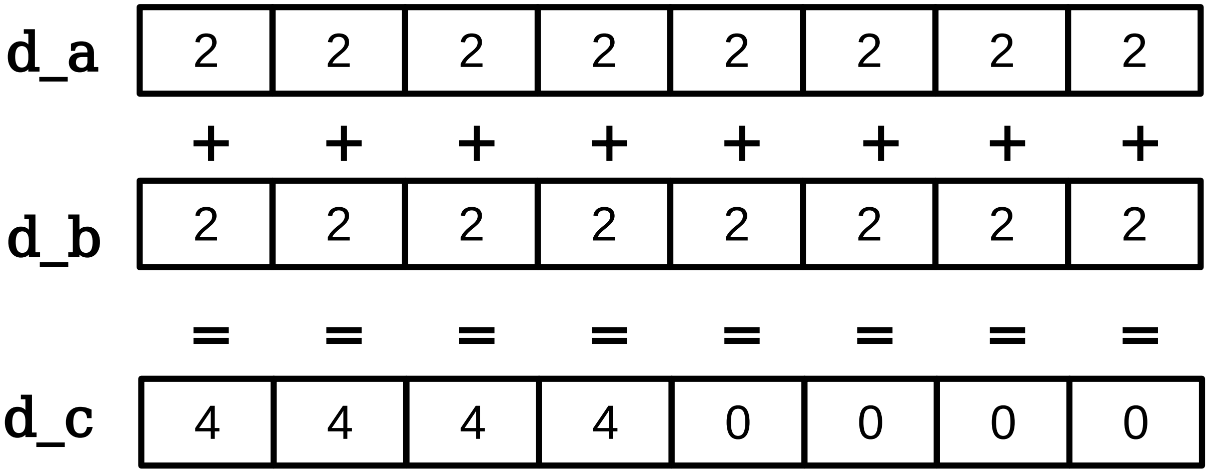
blockDim.x = 2

gridDim.x = 4 => Total number of thread blocks

Vector addition program

- **Mapping threads from multiple blocks to data**

```
int gid = threadIdx.x +  blockIdx.x * blockDim.x
```



d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	4	4	4	4	0	0	0	0

Vector addition program

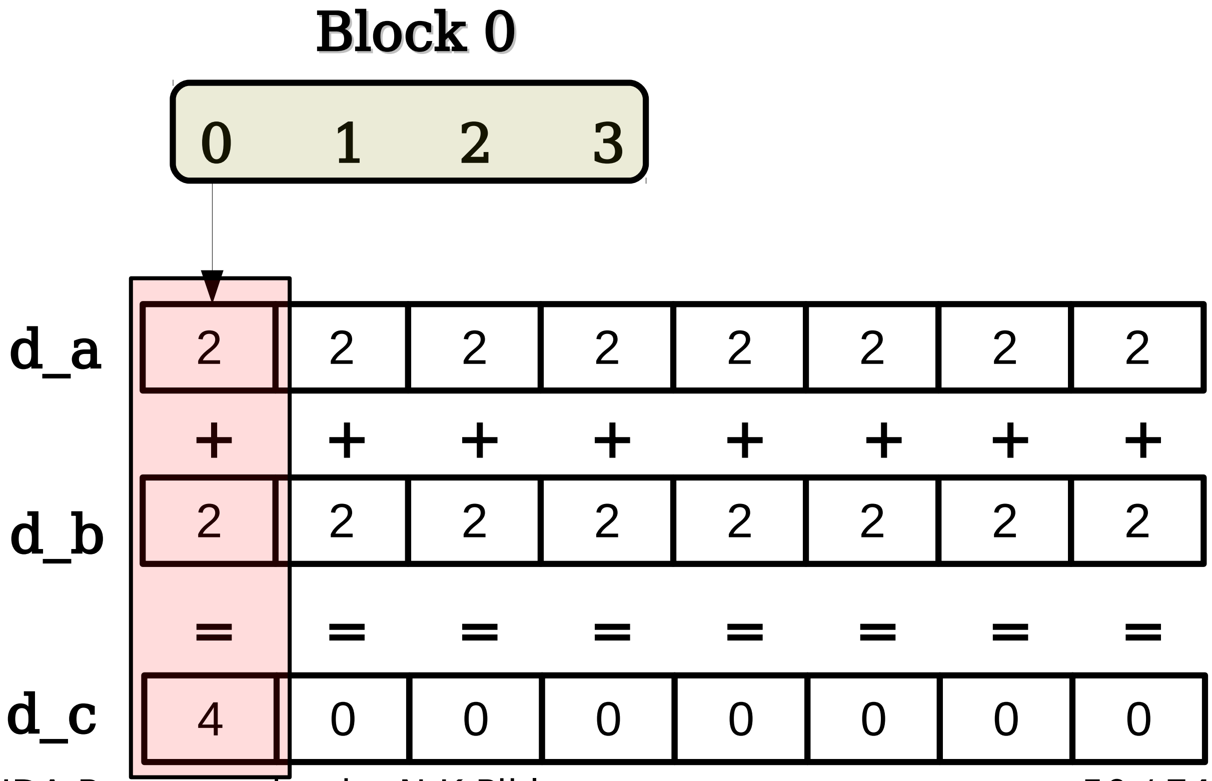
- **Mapping threads from multiple blocks to data**

```
int gid = threadIdx.x +  blockIdx.x * blockDim.x
    gid =       0      +     0      *     4
    gid =   0
```

threadIdx.x = 0

blockIdx.x = 0

blockDim.x = 4



Block 0

0 1 2 3

d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	4	0	0	0	0	0	0	0

Vector addition program

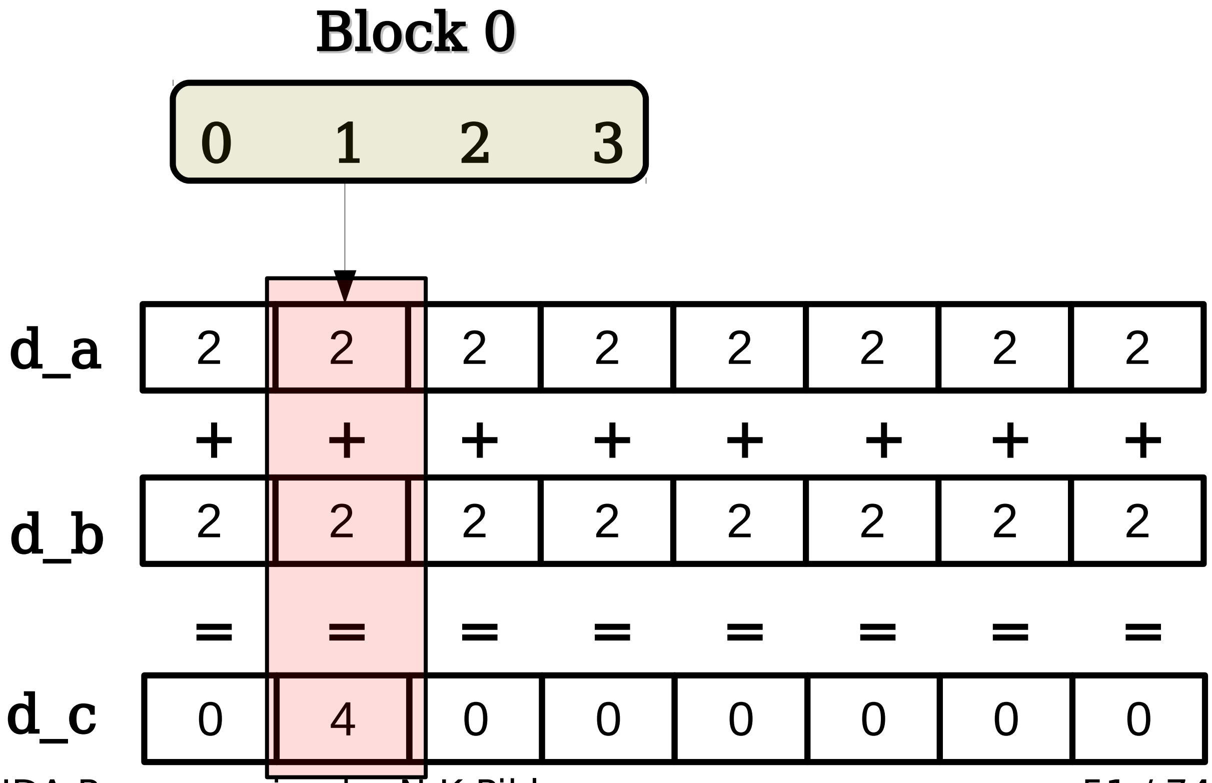
- Mapping threads from multiple blocks to data

```
int gid = threadIdx.x +  blockIdx.x * blockDim.x
    gid =        1          +      0        *       4
    gid =   1
```

threadIdx.x = 1

blockIdx.x   = 0

blockDim.x = 4



Block 0

0 1 2 3

d_a 2 2 2 2 2 2 2 2

+ + + + + + + +

d_b 2 2 2 2 2 2 2 2

= = = = = = = =

d_c 0 4 0 0 0 0 0 0

Vector addition program

- Mapping threads from multiple blocks to data

int gid = threadIdx.x + blockIdx.x * blockDim.x

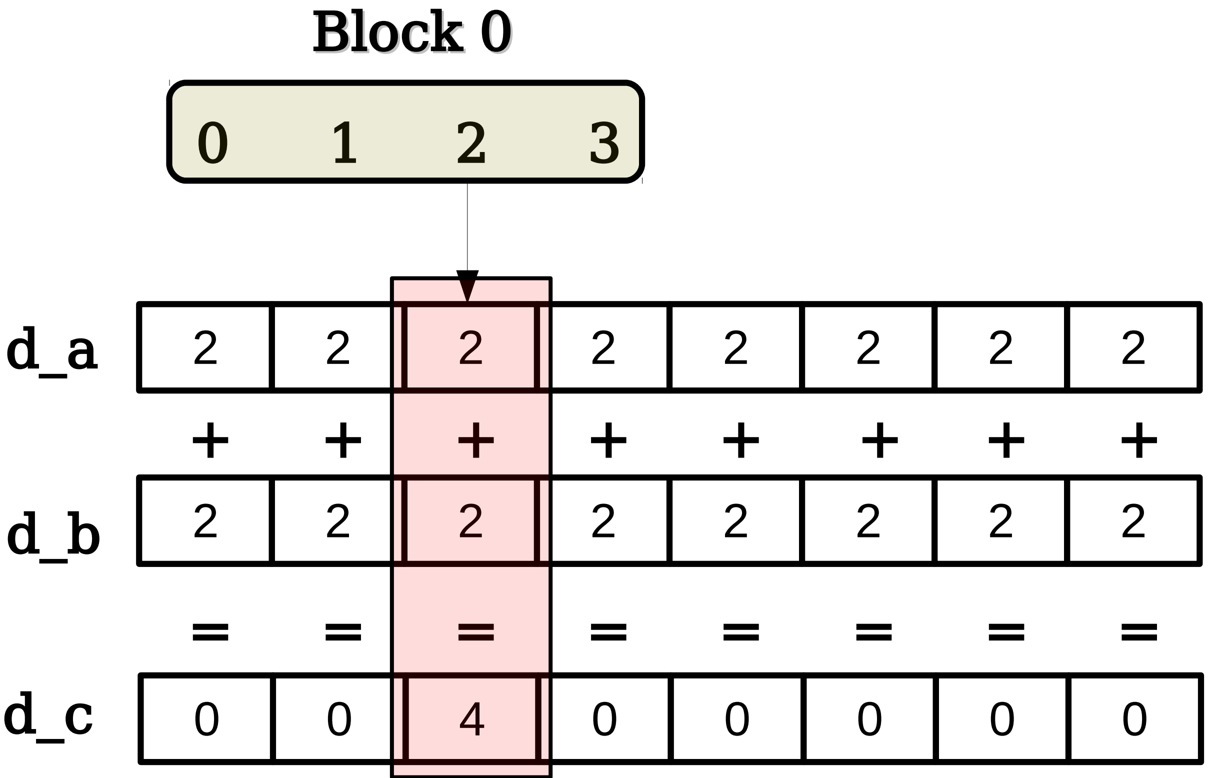
gid = 2 + 0 * 4

gid = 2

threadIdx.x = 2

blockIdx.x = 0

blockDim.x = 4



Block 0

0	1	2	3

d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	0	0	4	0	0	0	0	0

Vector addition program

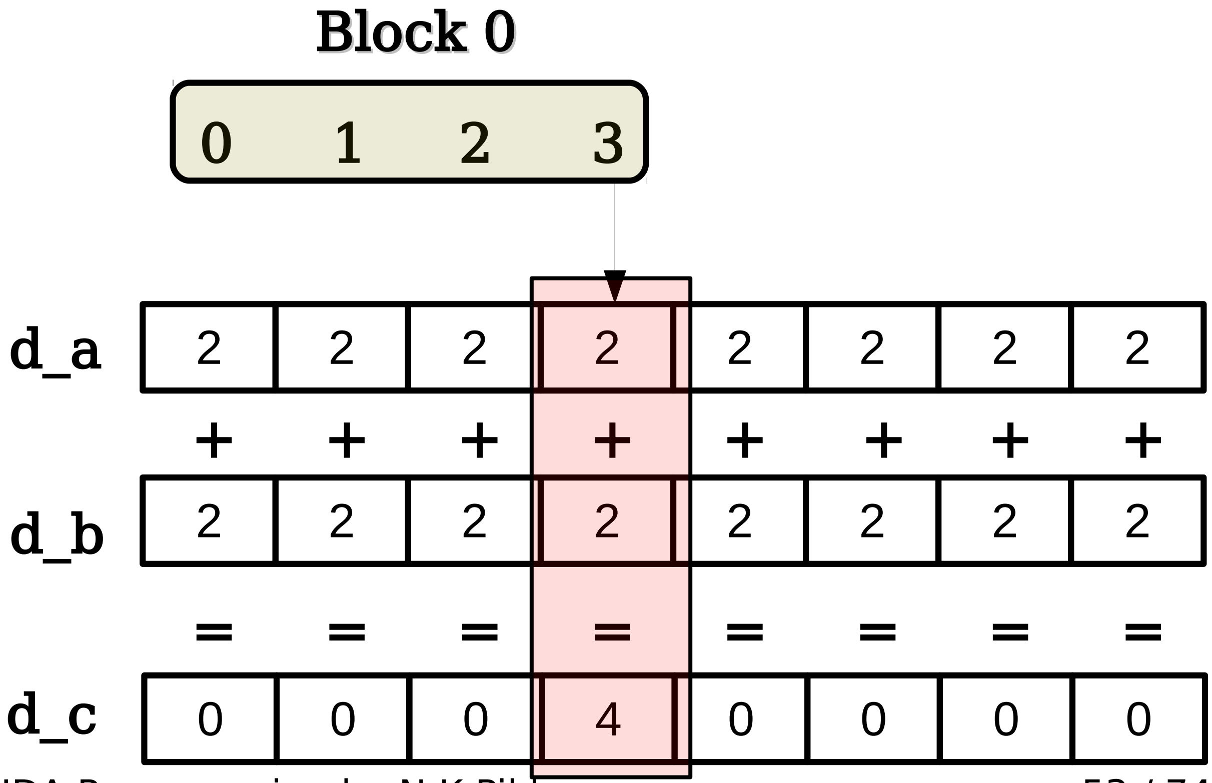
- **Mapping threads from multiple blocks to data**

```
int gid = threadIdx.x +  blockIdx.x * blockDim.x
    gid =       3       +      0       *       4
    gid =   3
```

threadIdx.x = 3

blockIdx.x = 0

blockDim.x = 4



Vector addition program

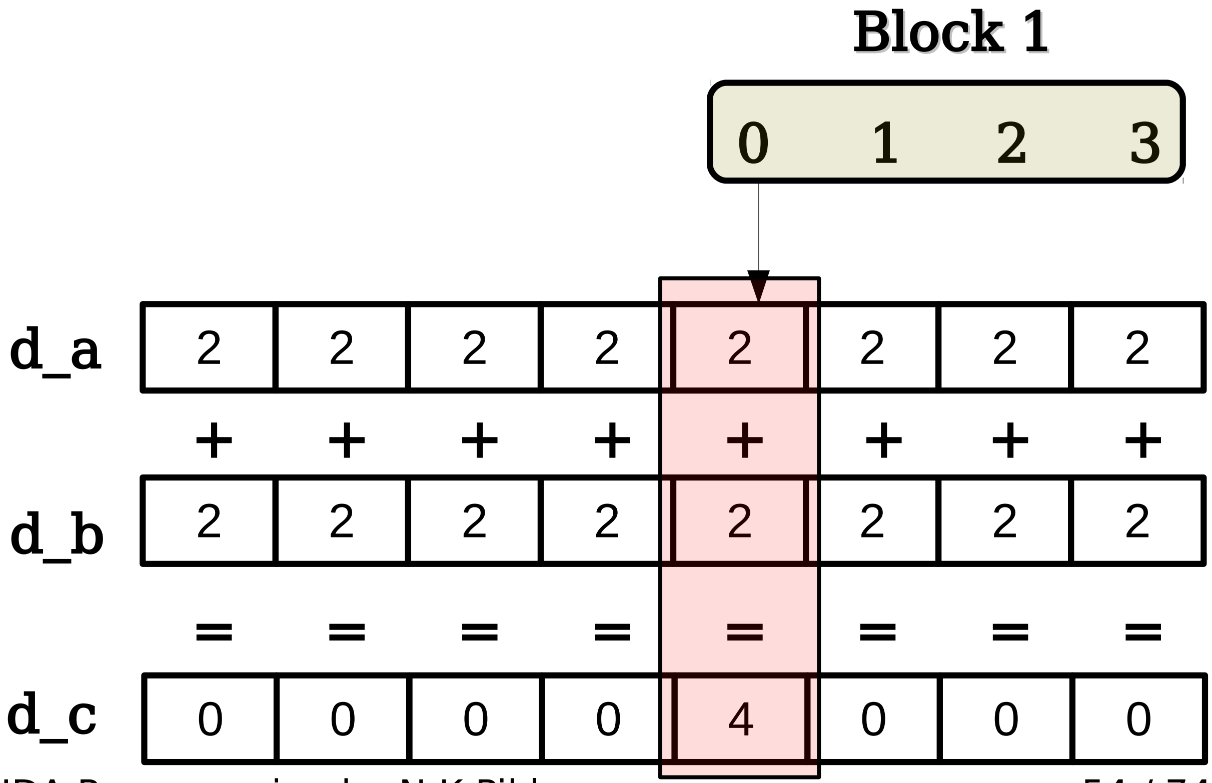
- Mapping threads from multiple blocks to data

```
int gid = threadIdx.x +  blockIdx.x * blockDim.x
    gid =        0       +      1      *      4
    gid =   4
```

threadIdx.x = 0

blockIdx.x   = 1

blockDim.x = 4



Block 1

0	1	2	3

d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	0	0	0	0	4	0	0	0

Vector addition program

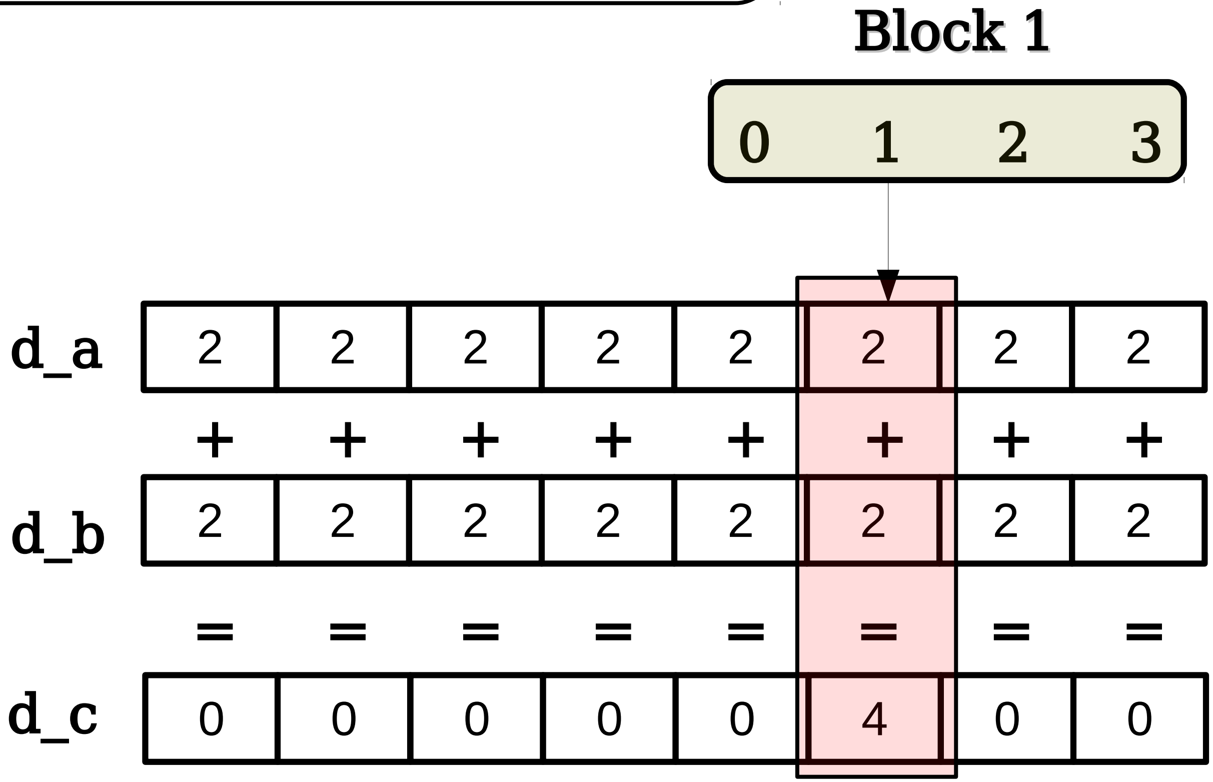
- **Mapping threads from multiple blocks to data**

```
int gid = threadIdx.x +  blockIdx.x * blockDim.x
    gid =       1      +      1     *      4
    gid =  5
```

threadIdx.x = 1

blockIdx.x = 1

blockDim.x = 4



d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	0	0	0	0	0	4	0	0

Vector addition program

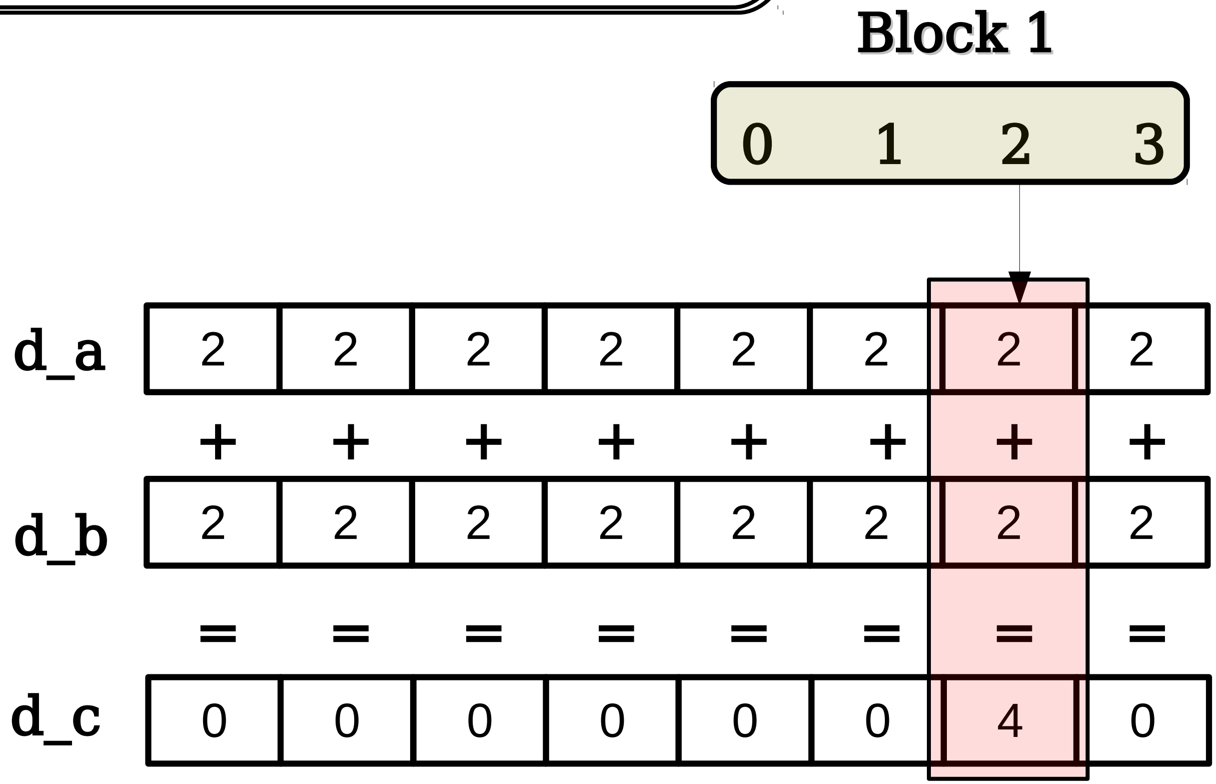
- **Mapping threads from multiple blocks to data**

```
int gid = threadIdx.x +  blockIdx.x * blockDim.x
    gid =       2        +      1       *       4
    gid =   6
```

threadIdx.x = 2

blockIdx.x   = 1

blockDim.x = 4



Vector addition program

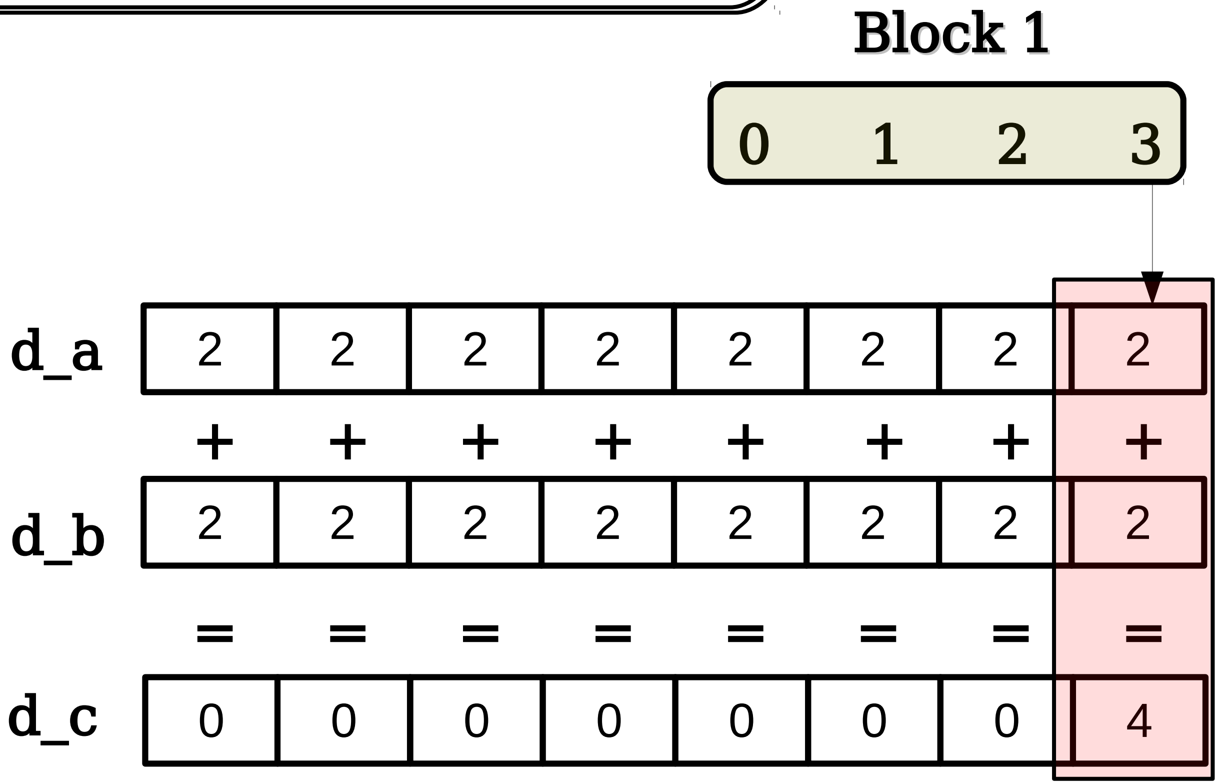
- Mapping threads from multiple blocks to data

```
int gid = threadIdx.x +  blockIdx.x * blockDim.x
    gid =       3      +      1     *      4
    gid =   7
```

threadIdx.x = 3

blockIdx.x = 1

blockDim.x = 4



Block 1

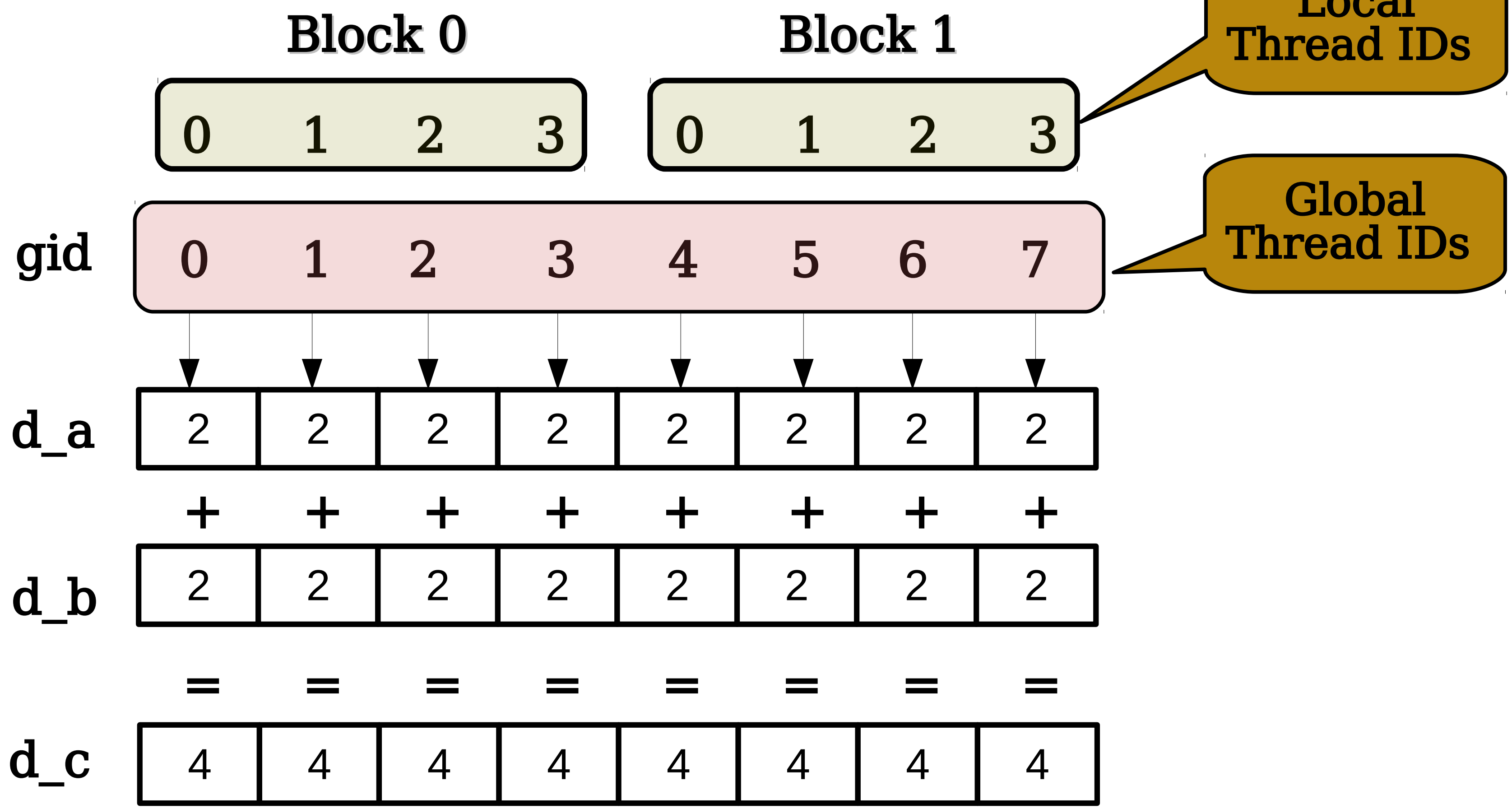
0 1 2 3

d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	0	0	0	0	0	0	0	4

Vector addition program

- Mapping threads from multiple blocks to data

```
int gid = threadIdx.x +  blockIdx.x * blockDim.x
```



	Block 0				Block 1			
	0	1	2	3	0	1	2	3
gid	0	1	2	3	4	5	6	7
d_a	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
d_b	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
d_c	4	4	4	4	4	4	4	4

Local Thread IDs

Global Thread IDs

Vector addition program

- **Problems**

1. What if number of threads greater than N?
2. What if number of threads less than N?

1. What if number of threads greater than N?

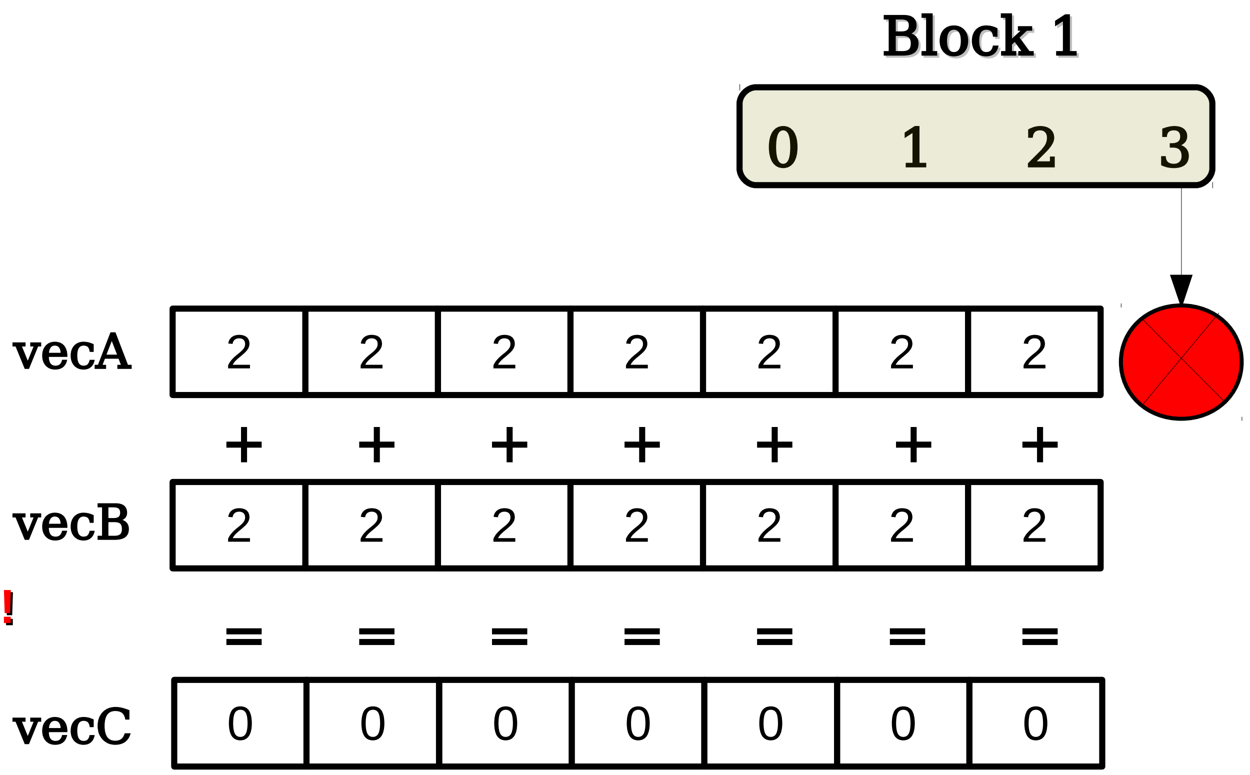
```
int gid = threadIdx.x +  blockIdx.x * blockDim.x
    gid =        3         +       1        *        4
    gid =   7
```

N = 7

threadIdx.x = 3

blockIdx.x = 1

blockDim.x = 4



Block 1

0 1 2 3

vecA	2	2	2	2	2	2	2
	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2
	=	=	=	=	=	=	=
vecC	0	0	0	0	0	0	0

Segmentation Fault!!!

What if number of threads greater than N?

```
__global__ void vecAdd_kernel(int *a, int *b, int *c, int N){
  int gid = threadIdx.x + blockIdx.x * blockDim.x;
  if(gid < N){
  c[gid] = a[gid] + b[gid];
}
```

Only threads having gid < N are allowed to execute the addition statement

Program references

1vecAddUsingManyBlocks.cu

2. What if number of threads less than N?

- Vector addition using a #threads < N

```
__global__ void vecAdd_kernel(int *a, int *b, int *c, int N){
  int gid = threadIdx.x + blockIdx.x * blockDim.x;
  int stride = blockDim.x * gridDim.x;
  int i;
  for(i = gid; i < N; i += stride){
  c[i] = a[i] + b[i];
}
```

#blocks in grid

What if number of threads less than N?

```
int gid = threadIdx.x + blockIdx.x * blockDim.x;
int stride = blockDim.x * gridDim.x;
for(i = gid; i < N; i += stride)
```

threadIdx.x = 0

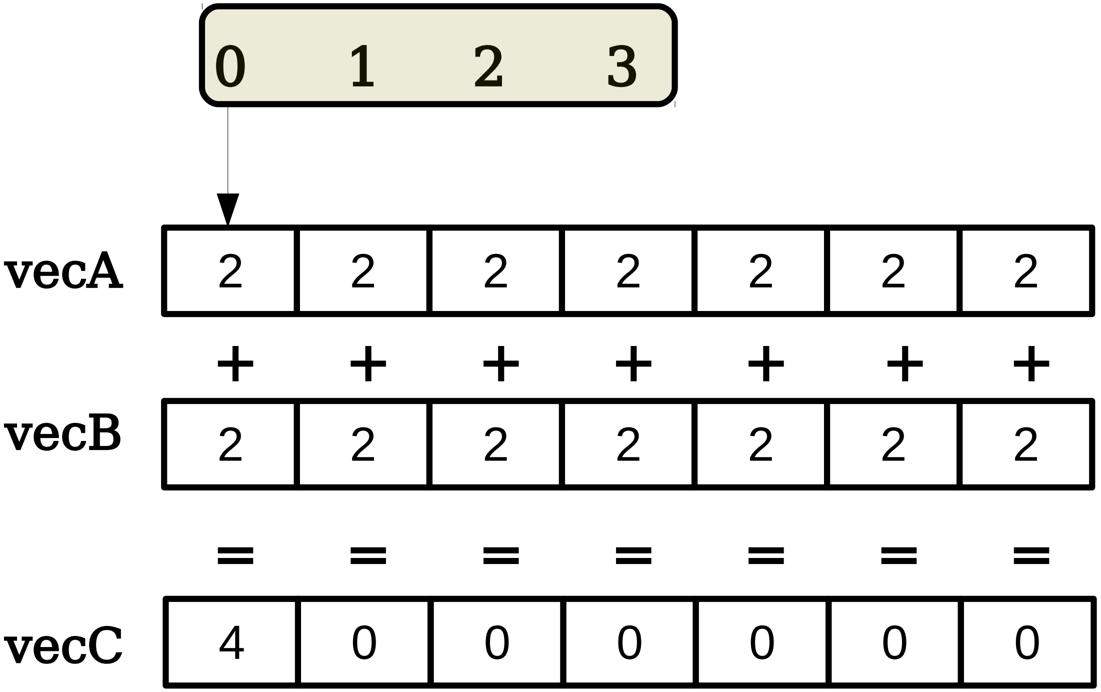
blockIdx.x = 0

blockDim.x = 4

gid = 0

Stride = 4

i = 0



vecA	2	2	2	2	2	2	2
	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2
	=	=	=	=	=	=	=
vecC	4	0	0	0	0	0	0

What if number of threads less than N?

```
int gid = threadIdx.x + blockIdx.x * blockDim.x;
int stride = blockDim.x * gridDim.x;
for(i = gid; i < N; i += stride)
```

threadIdx.x = 1

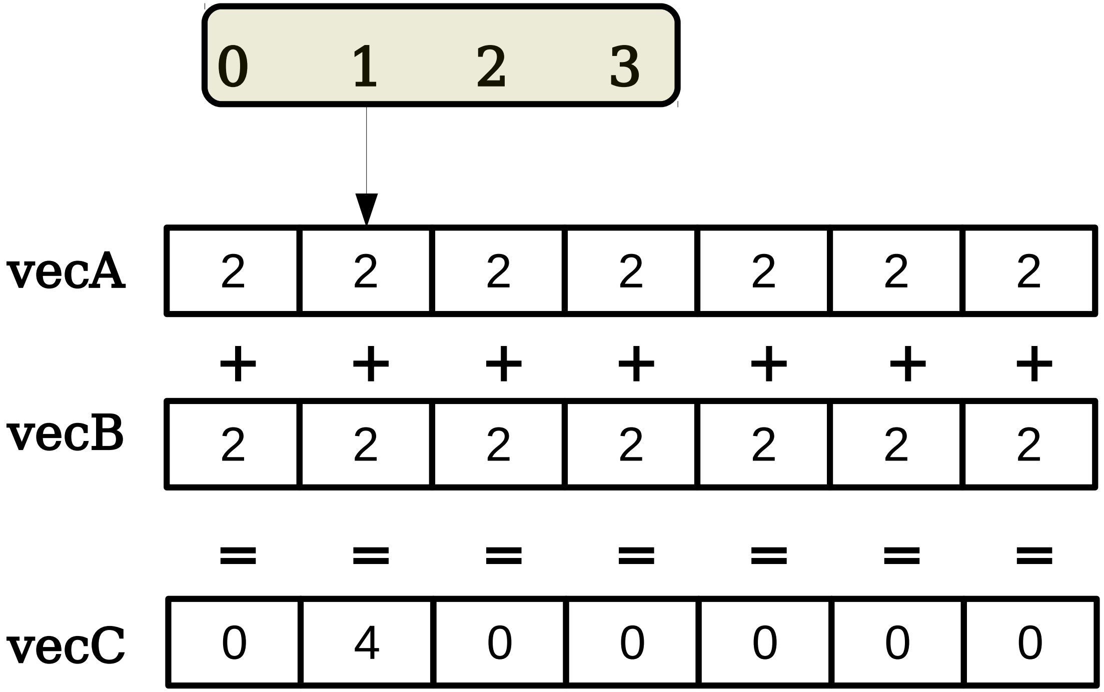
blockIdx.x = 0

blockDim.x = 4

gid = 1

Stride = 4

i = 1



0 1 2 3

vecA	2	2	2	2	2	2	2
	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2
	=	=	=	=	=	=	=
vecC	0	4	0	0	0	0	0

What if number of threads less than N?

```
int gid = threadIdx.x + blockIdx.x * blockDim.x;
int stride = blockDim.x * gridDim.x;
for(i = gid; i < N; i += stride)
```

threadIdx.x = 2

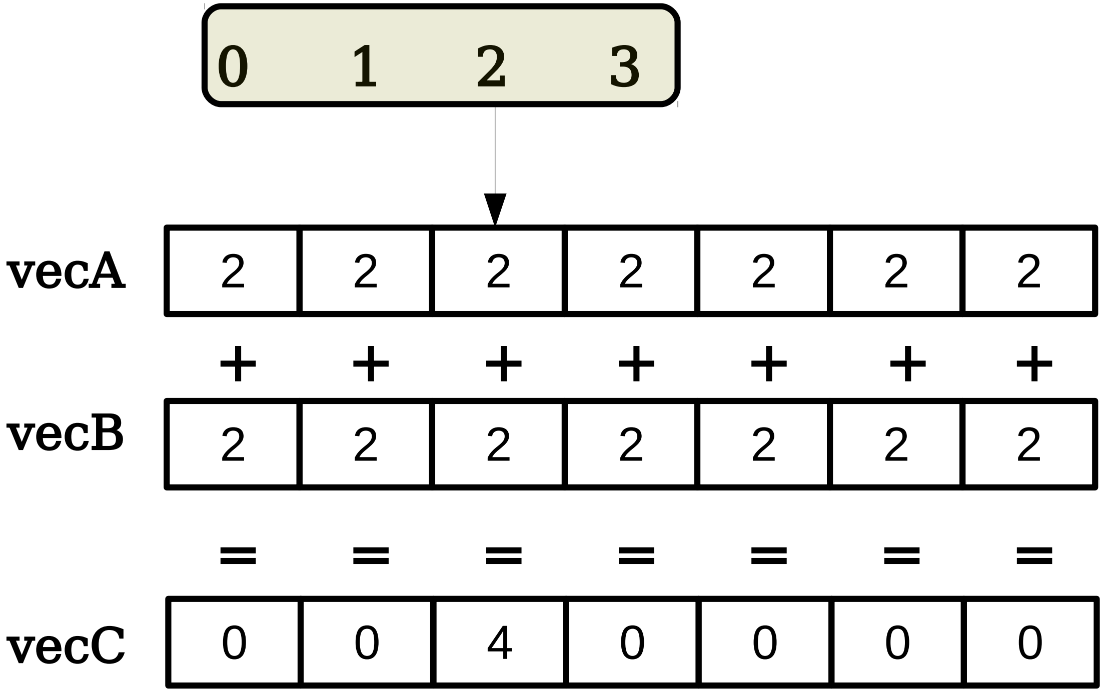
blockIdx.x = 0

blockDim.x = 4

gid = 2

Stride = 4

i = 2



0 1 2 3

vecA 2 2 2 2 2 2 2

+ + + + + + +

vecB 2 2 2 2 2 2 2

= = = = = = =

vecC 0 0 4 0 0 0 0

What if number of threads less than N?

```
int gid = threadIdx.x + blockIdx.x * blockDim.x;
int stride = blockDim.x * gridDim.x;
for(i = gid; i < N; i += stride)
```

threadIdx.x = 3

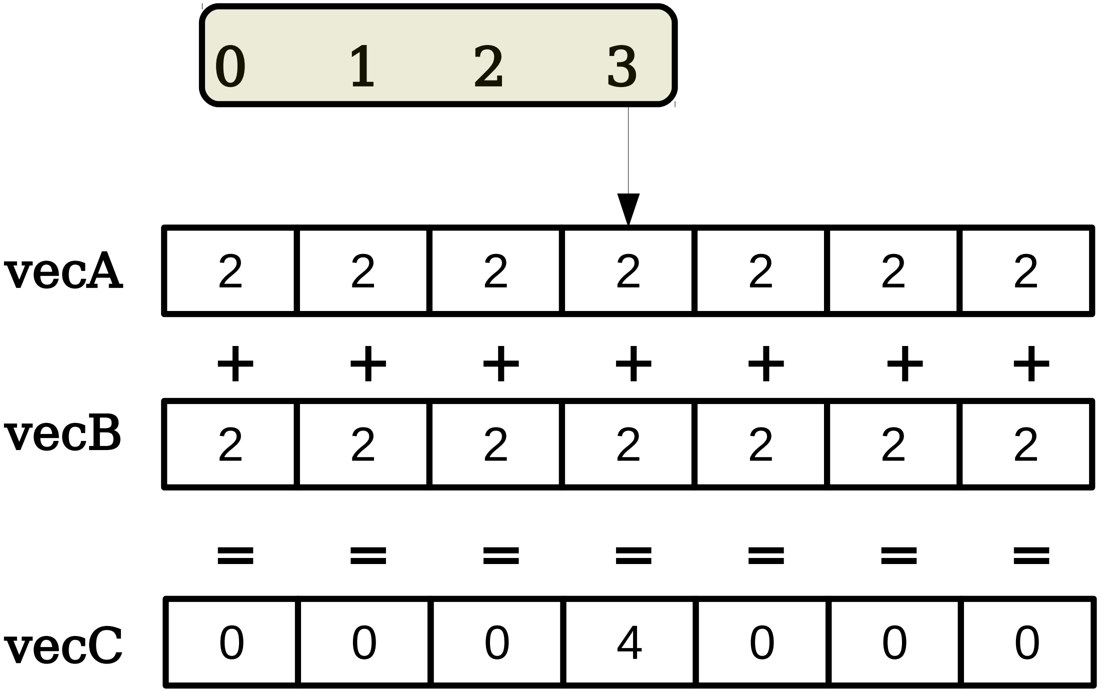
blockIdx.x = 0

blockDim.x = 4

gid = 3

Stride = 4

i = 3

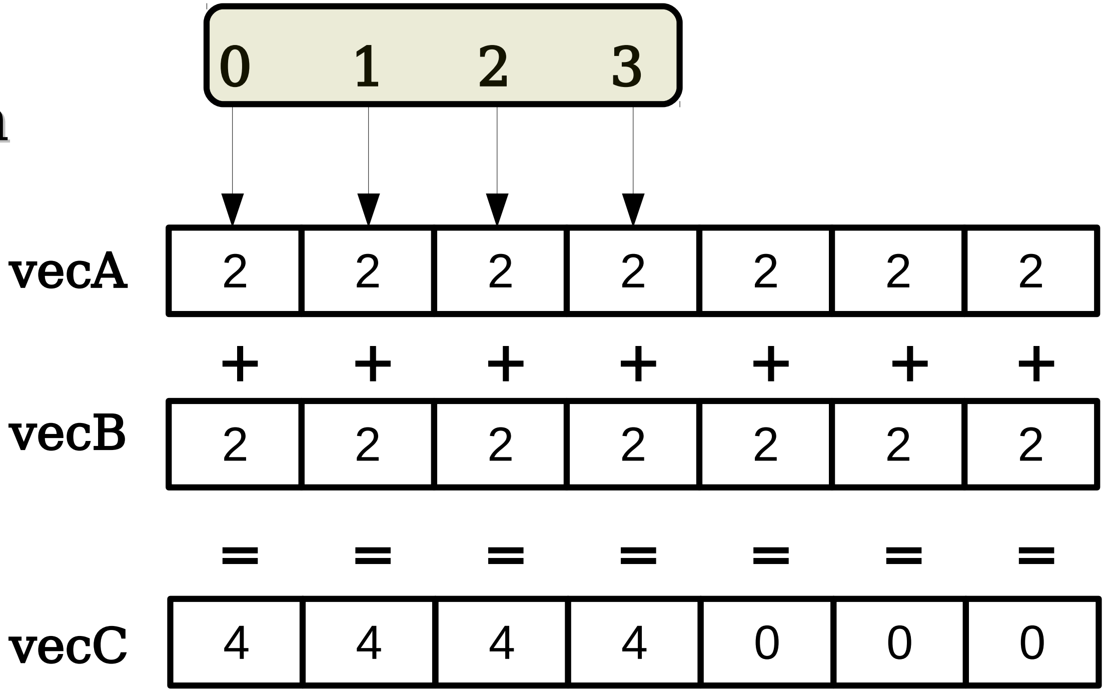


vecA	2	2	2	2	2	2	2
	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2
	=	=	=	=	=	=	=
vecC	0	0	0	4	0	0	0

What if number of threads less than N?

```
int gid = threadIdx.x + blockIdx.x * blockDim.x;
int stride = blockDim.x * gridDim.x;
for(i = gid; i < N; i += stride)
```

Actually all threads from one block are executed simultaneously



0 1 2 3

vecA	2	2	2	2	2	2	2
	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2
	=	=	=	=	=	=	=
vecC	4	4	4	4	0	0	0

What if number of threads less than N?

```
int gid = threadIdx.x + blockIdx.x * blockDim.x;
int stride = blockDim.x * gridDim.x;
for(i = gid; i < N; i += stride)
```

threadIdx.x = 0

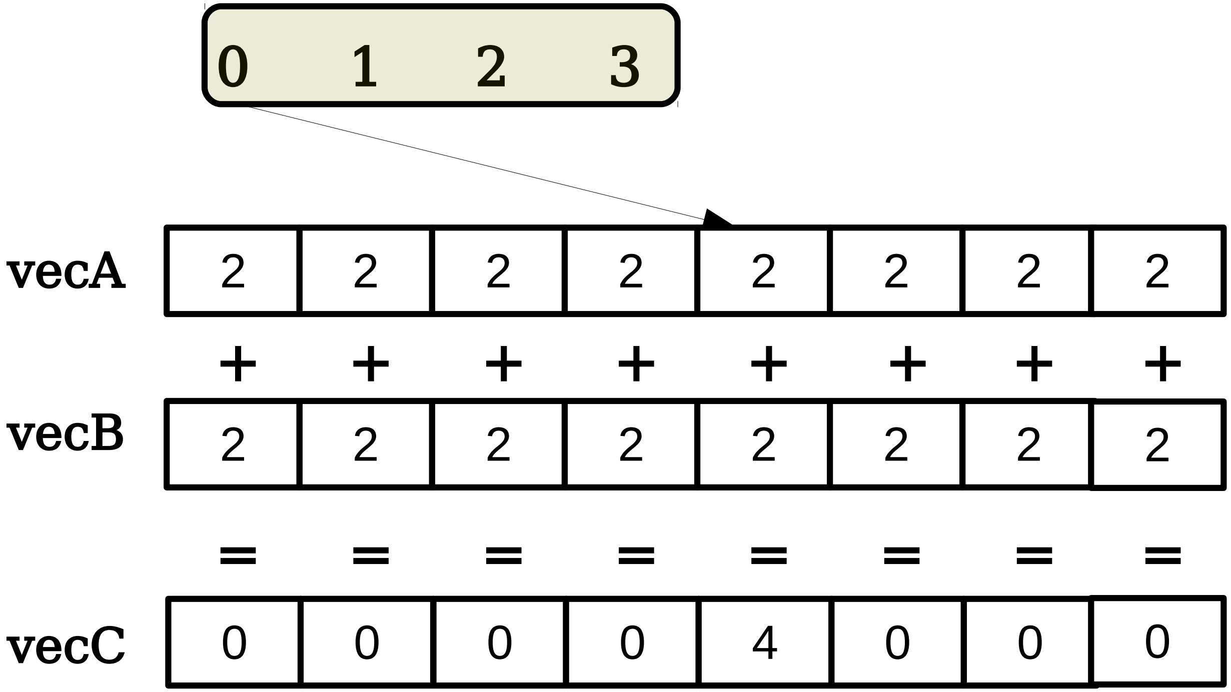
blockIdx.x = 0

blockDim.x = 4

gid = 0

Stride = 4

i = 4



vecA	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
vecC	0	0	0	0	4	0	0	0

What if number of threads less than N?

```
int gid = threadIdx.x + blockIdx.x * blockDim.x;
int stride = blockDim.x * gridDim.x;
for(i = gid; i < N; i += stride)
```

threadIdx.x = 1

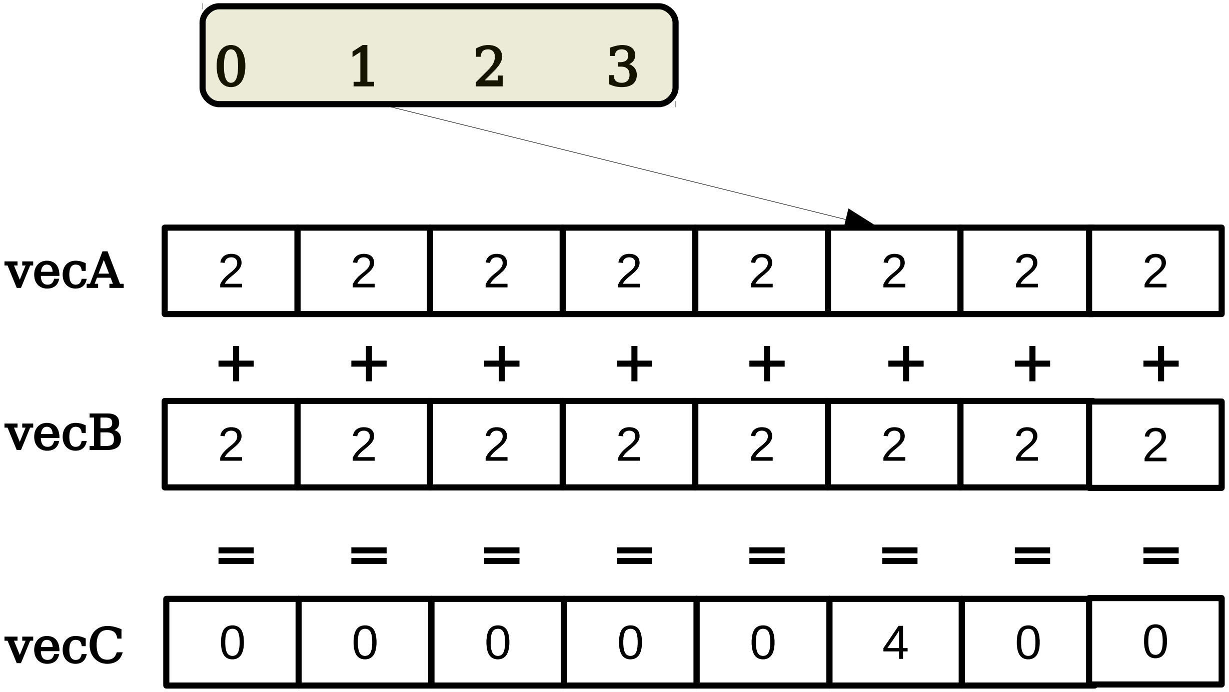
blockIdx.x = 0

blockDim.x = 4

gid = 1

Stride = 4

i = 5



0 1 2 3

vecA	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
vecC	0	0	0	0	0	4	0	0

What if number of threads less than N?

```
int gid = threadIdx.x + blockIdx.x * blockDim.x;
int stride = blockDim.x * gridDim.x;
for(i = gid; i < N; i += stride)
```

threadIdx.x = 2

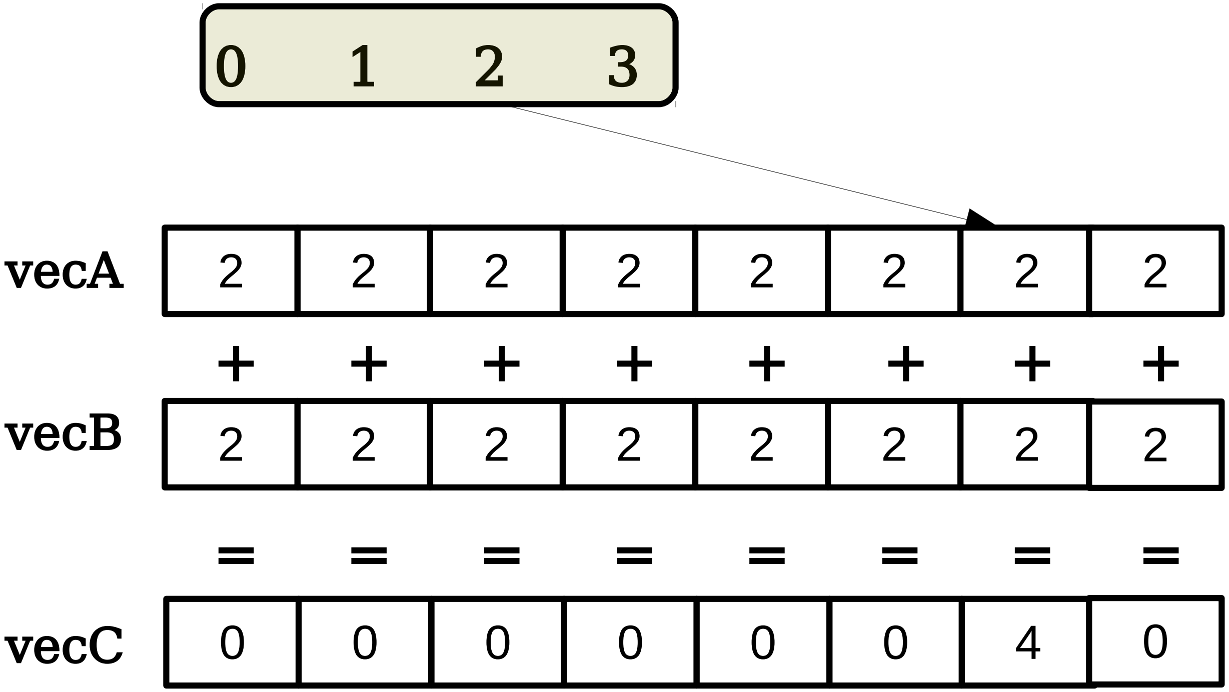
blockIdx.x = 0

blockDim.x = 4

gid = 2

Stride = 4

i = 6



0	1	2	3

vecA	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
vecC	0	0	0	0	0	0	4	0

What if number of threads less than N?

```
int gid = threadIdx.x + blockIdx.x * blockDim.x;
int stride = blockDim.x * gridDim.x;
for(i = gid; i < N; i += stride)
```

threadIdx.x = 3

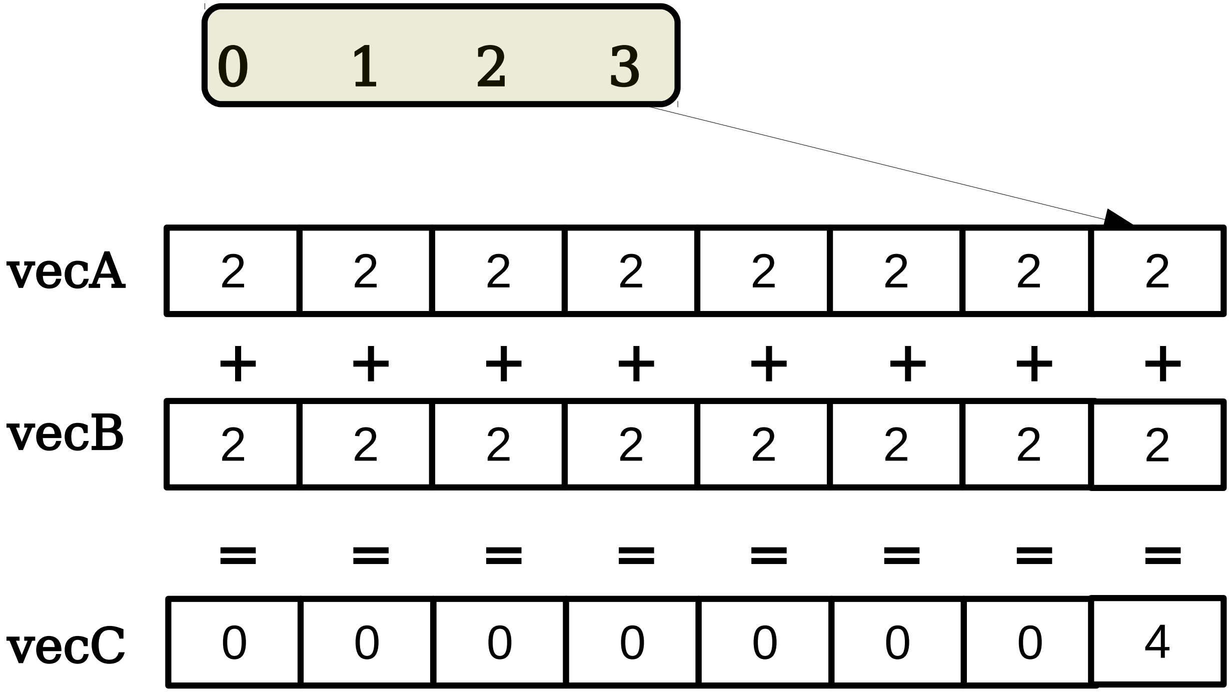
blockIdx.x = 0

blockDim.x = 4

gid = 3

Stride = 4

i = 7



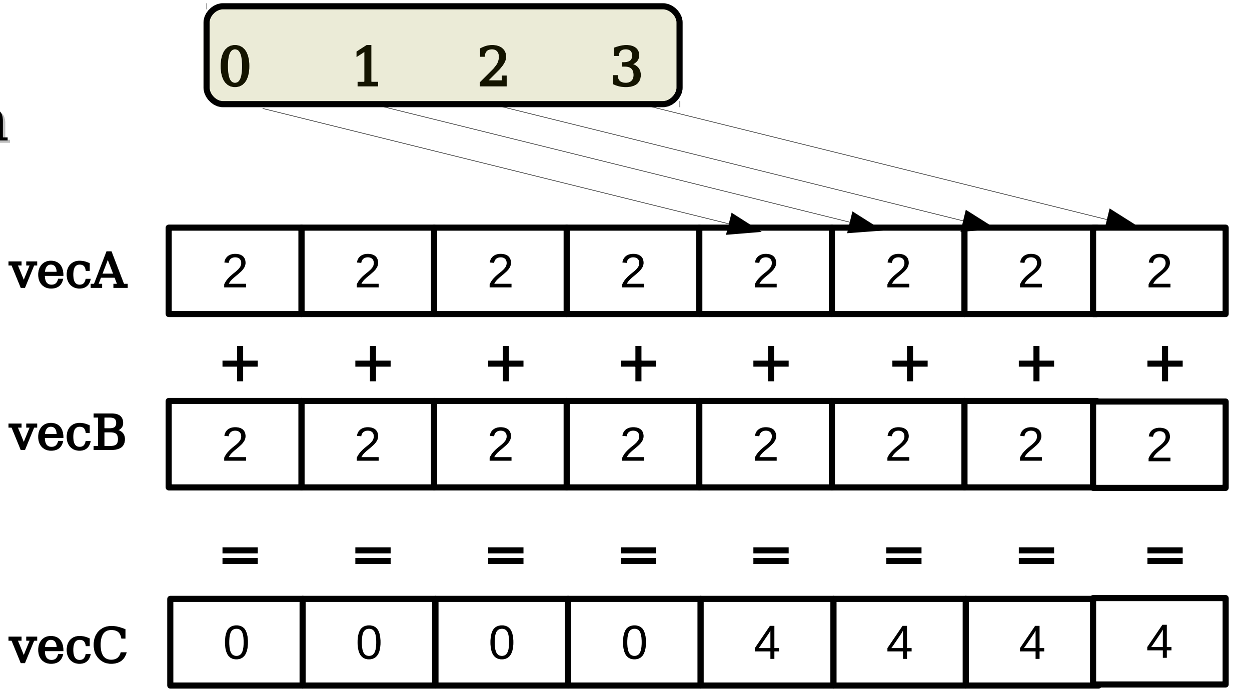
0 1 2 3

vecA	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
vecC	0	0	0	0	0	0	0	4

What if number of threads less than N?

```
int gid = threadIdx.x + blockIdx.x * blockDim.x;
int stride = blockDim.x * gridDim.x;
for(i = gid; i < N; i += stride)
```

Actually all threads from one block are executed simultaneously



0 1 2 3

vecA	2	2	2	2	2	2	2	2
	+	+	+	+	+	+	+	+
vecB	2	2	2	2	2	2	2	2
	=	=	=	=	=	=	=	=
vecC	0	0	0	0	4	4	4	4

What if number of threads less than N?

Program references

2vecAddUsingManyBlocks(N>threads).cu

Summary

- **Thread organization 1 D**
- **Case study vector addition**

  scenario 1 using single thread block

  1. N less than or equal to block size
  2. N greater than thread block size
- **Scenario 2** using multiple thread blocks

  1. N less than or equal to total threads
  2. N greater than thread total threads