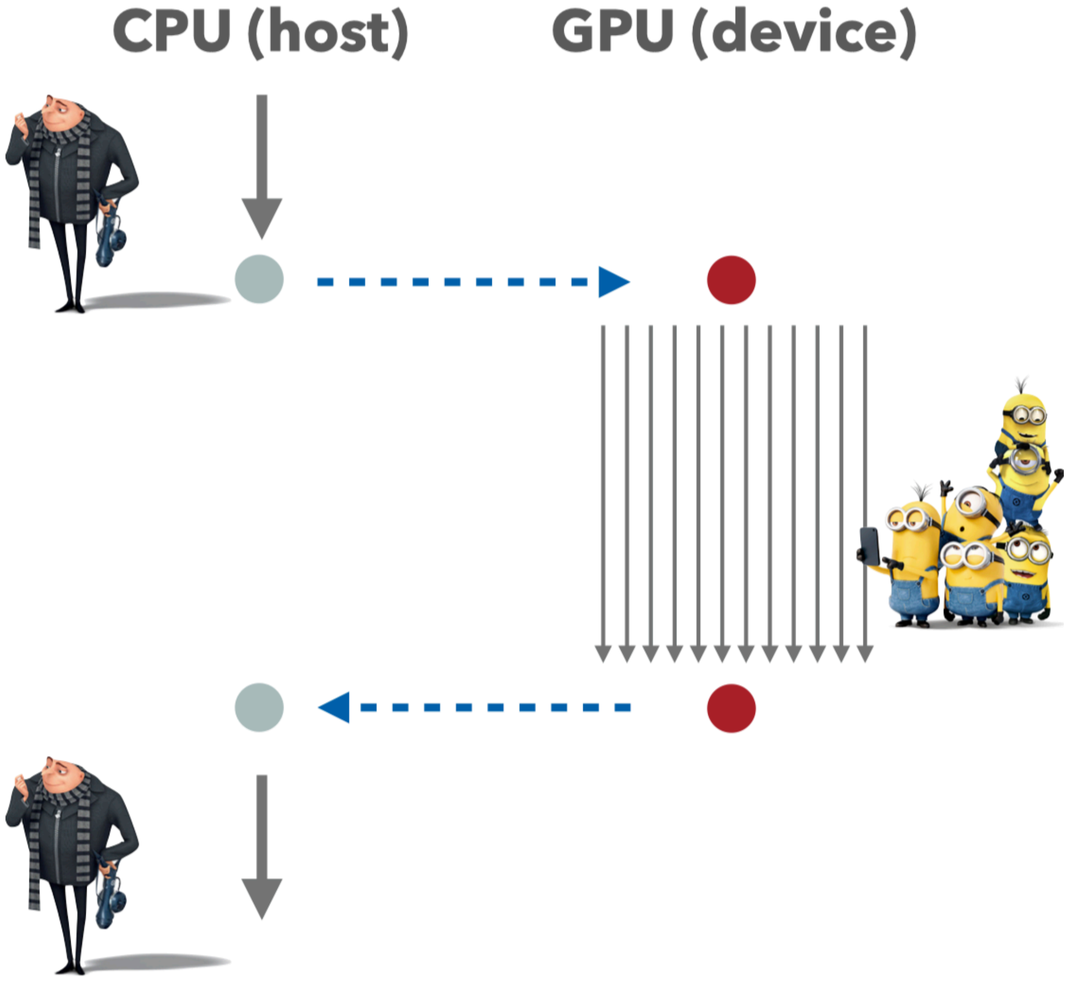
ME 471/571

First CUDA programs

SIMPLE WORKFLOW IN CUDA C

- host executes serial part of the program
- copy data from host to device
- multiple threads execute kernel on parts of data
- copy result from device to host
- host continues the execution of serial program



EXAMPLE 1 - ADDITION

Compute c = a+b

1) Allocate space on GPU for a, b, c
2) Copy values of a, b from host to device
3) Launch kernel to add numbers
4) Copy value of c from device to host
5) Free GPU memory

EXAMPLE 1 - ADDITION

Compute c = a+b

1) Allocate space on GPU for a, b, c
2) Copy values of a, b from host to device
3) Launch kernel to add numbers
4) Copy value of c from device to host
5) Free GPU memory

cudaMalloc

```
cudaMalloc((float**) &d_a, size)
```

pointer to device copy of a

size in bytes

EXAMPLE 1 - ADDITION

Compute c = a+b

1) Allocate space on GPU for a, b, c — `cudaMalloc`
2) **Copy values of a, b from host to device** — **`cudaMemcpy`**
3) Launch kernel to add numbers
4) Copy value of c from device to host
5) Free GPU memory

```
cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice)
```

device copy of a — *host copy of a* — *size in bytes* — *direction of copy*

EXAMPLE 1 - ADDITION

Compute c = a+b

1) Allocate space on GPU for a, b, c — `cudaMalloc`
2) Copy values of a, b from host to device — `cudaMemcpy`
3) **Launch kernel to add numbers** — **`add_on_GPU`**
4) Copy value of c from device to host
5) Free GPU memory

custom kernel

```
__global__ add_on_GPU(float *a, float *b, float *c){
    *c = *a + *b;
    }
```

EXAMPLE 1 - ADDITION

Compute c = a+b

1) Allocate space on GPU for a, b, c — `cudaMalloc`
2) Copy values of a, b from host to device — `cudaMemcpy`
3) Launch kernel to add numbers — `add_on_GPU`
4) **Copy value of c from device to host** — **`cudaMemcpy`**
5) Free GPU memory

```
cudaMemcpy(&c, d_c, size, cudaMemcpyDeviceToHost)
```

host address of c — *device copy of c* — *size in bytes* — *direction of copy*

EXAMPLE 1 - ADDITION

Compute c = a+b

1) Allocate space on GPU for a, b, c — `cudaMalloc`
2) Copy values of a, b from host to device — `cudaMemcpy`
3) Launch kernel to add numbers — `add_on_GPU`
4) **Copy value of c from device to host** — `cudaMemcpy`
5) Free GPU memory — **`cudaFree`**

```
cudaFree(d_a);
```

EXAMPLE 2 – VECTOR ADDITION

Vector addition will have the same flow:

1) Allocate device memory for vectors a, b, c — **`cudaMalloc`**
2) Copy vectors a and b to the device — **`cudaMemcpy`**
3) Launch vector addition kernel — **`sumArraysOnGPU`**
4) Copy result vector c to host — **`cudaMemcpy`**
5) Deallocate device memory — **`cudaFree`**

EXAMPLE 2 - VECTOR ADDITION

```
void sumArraysOnHost(float *A, float *B, float *C, int N)
{
    for(int i=0; i<N i++)
        C[i] = A[i] + B[i];
}
```

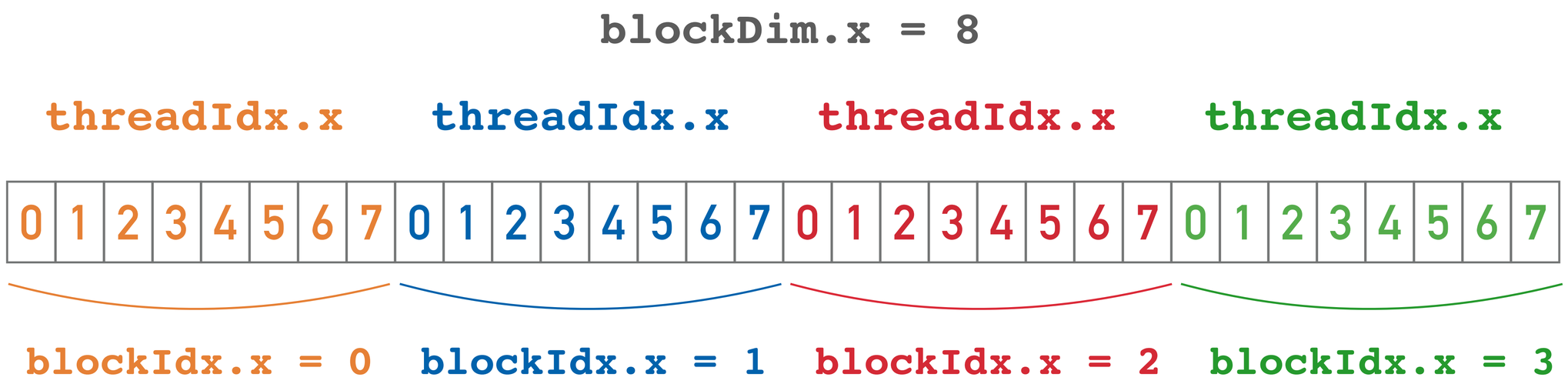
EXAMPLE 2 - VECTOR ADDITION

```
void sumArraysOnHost(float *A, float *B, float *C, int N)
{
    for(int i=0; i<size i++)
        C[i] = A[i] + B[i];
}


__global__ void sumArraysOnGPU(float *A, float *B, float *C, int N)
{
    int i=threadIdx.x;
    if(i<N)
        C[i] = A[i] + B[i];
}
```

BLOCKS AND THREADS

`blockIdx.x`	*index of a block*
`blockDim.x`	*number of threads in a block*
`threadIdx.x`	*index of a thread in a block*



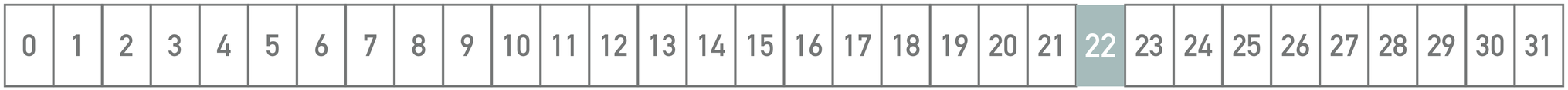
BLOCKS AND THREADS

blockDim.x = 8

threadIdx.x **threadIdx.x** **threadIdx.x** **threadIdx.x**

blockIdx.x = 0 **blockIdx.x = 1** **blockIdx.x = 2** **blockIdx.x = 3**

index = blockIdx.x * blockDim.x + threadIdx.x



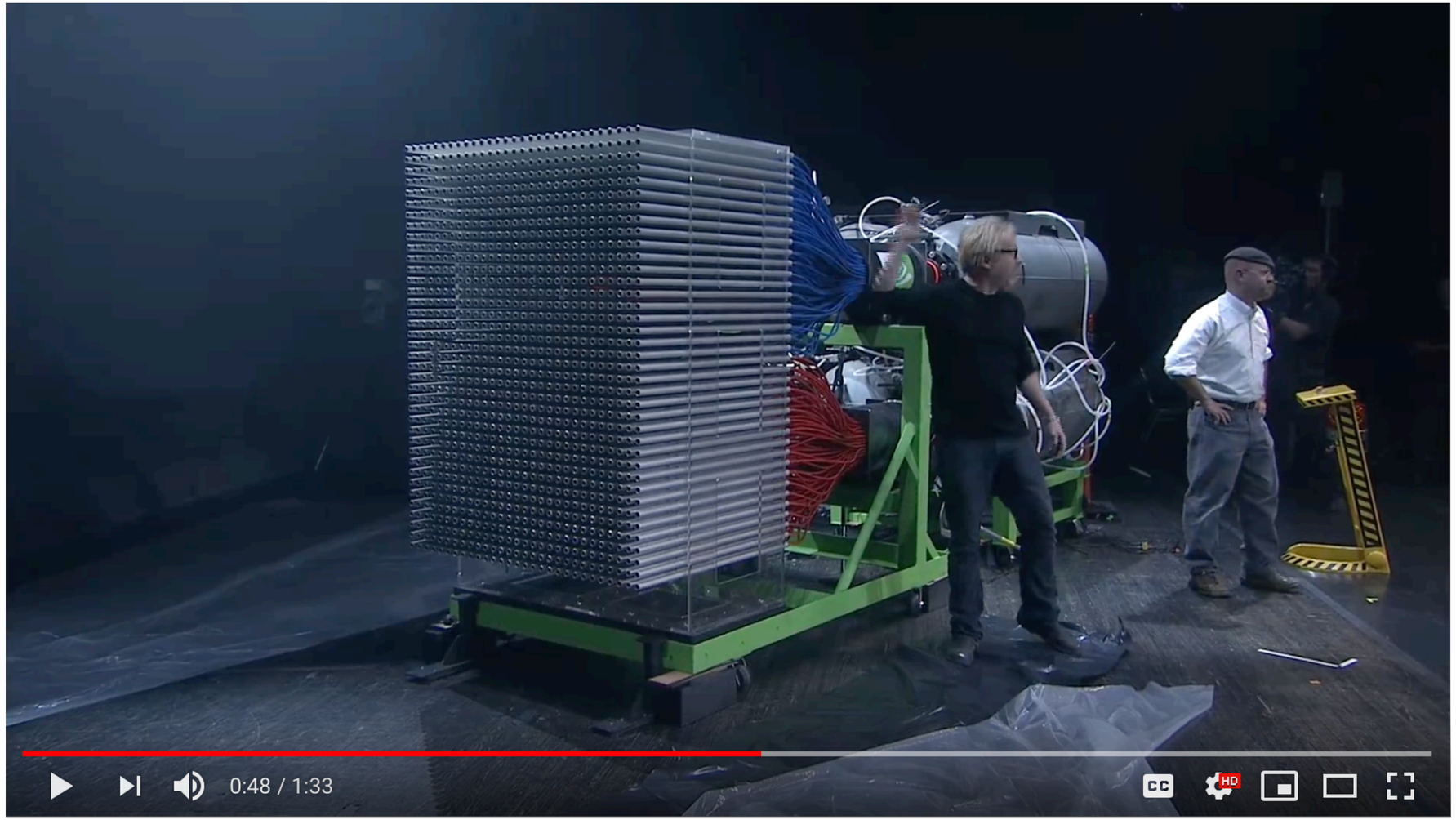

```
index = blockIdx.x * blockDim.x + threadIdx.x
      =      2      *      8      +      6
      =      22
```

EXAMPLE 2 - VECTOR ADDITION

```
void sumArraysOnHost(float *A, float *B, float *C, int N)
{
    for(int i=0; i<size i++)
        C[i] = A[i] + B[i];
}


__global__ void sumArraysOnGPU(float *A, float *B, float *C, int N)
{
    int i= blockIdx.x*blockDim.x + threadIdx.x;
    if(i<N)
        C[i] = A[i] + B[i];
}
```

CPU LOOPS VS GPU THREADS



https://youtu.be/-P28LKWTzrI