CUDA Essentials

CUDA Workflow and Data Movement

Stefano Markidis and Sergio Rivas-Gomez

Four Key-points

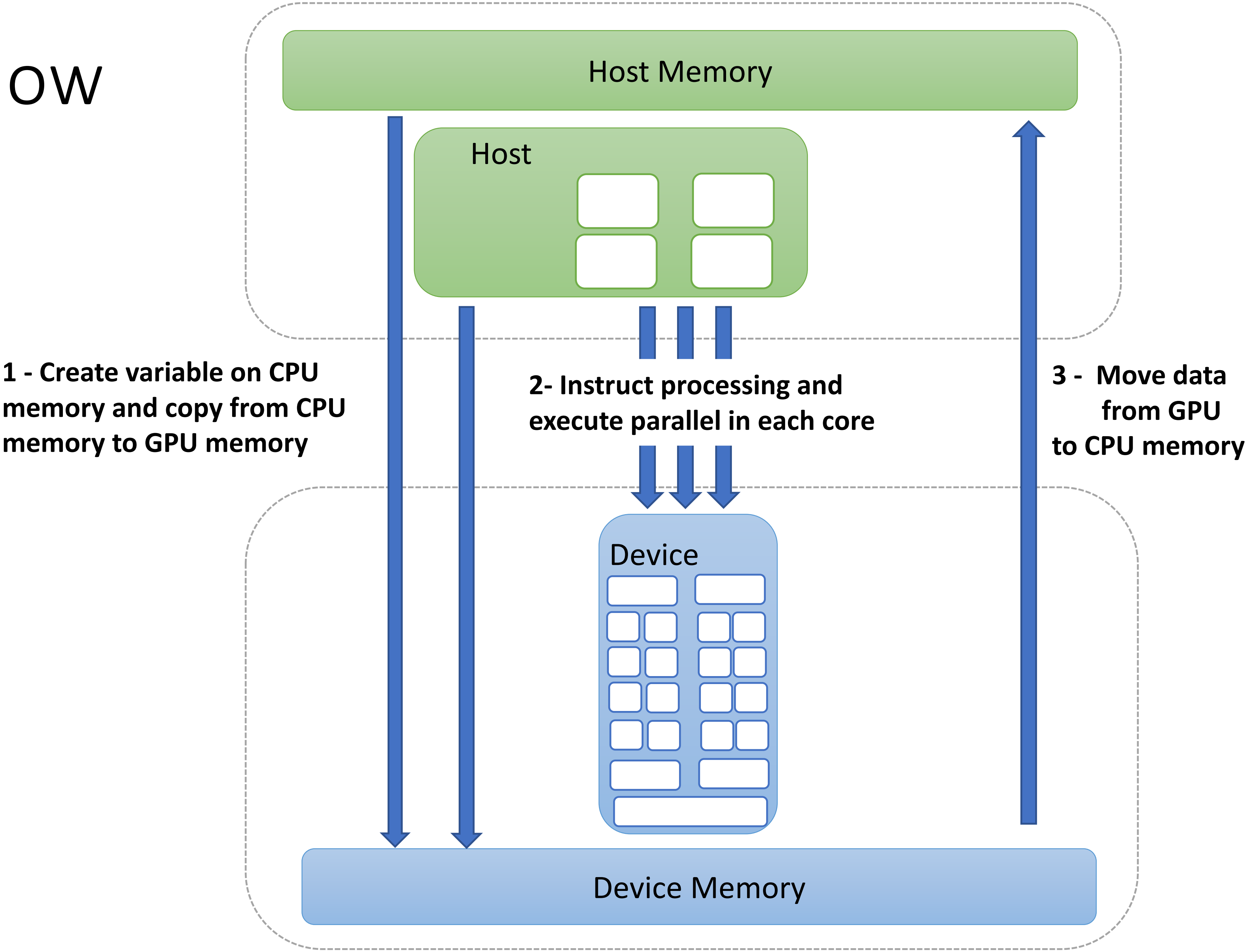
- CUDA has its own particular “terminology” to express traditional concepts
- When developing an application for GPUs, we follow the typical CUDA workflow: 1. move input data from CPU to GPU, 2. make computations on the GPU, 3. move the output results data back to CPU.
- To allocate memory on the GPU, CUDA provides `cudaMalloc()`, similar to C `malloc()`
- We move data from/to GPU by copying data with `cudaMemcpy`() from/to GPU memory location to/from a CPU memory location.

CUDA Jargon

- **Host** = CPU
- **Device** = GPU
- **Kernel** = function executed on the GPU by each thread
- **Launch** = CPU instructs GPU to execute a kernel
- **Execution configuration** = definition of how many threads to run on the GPU and how to group them

I FORGOT TO MAKE AN AGENDA FOR THIS MEETING, SO I'LL JUST FREESTYLE IT WITH JARGON.
Dilbert.com
@ScottAdamsSays

CUDA Workflow



Host Memory

Host

1 - Create variable on CPU memory and copy from CPU memory to GPU memory

2- Instruct processing and execute parallel in each core

3 - Move data from GPU to CPU memory

Device

Device Memory

Allocate Memory on GPU

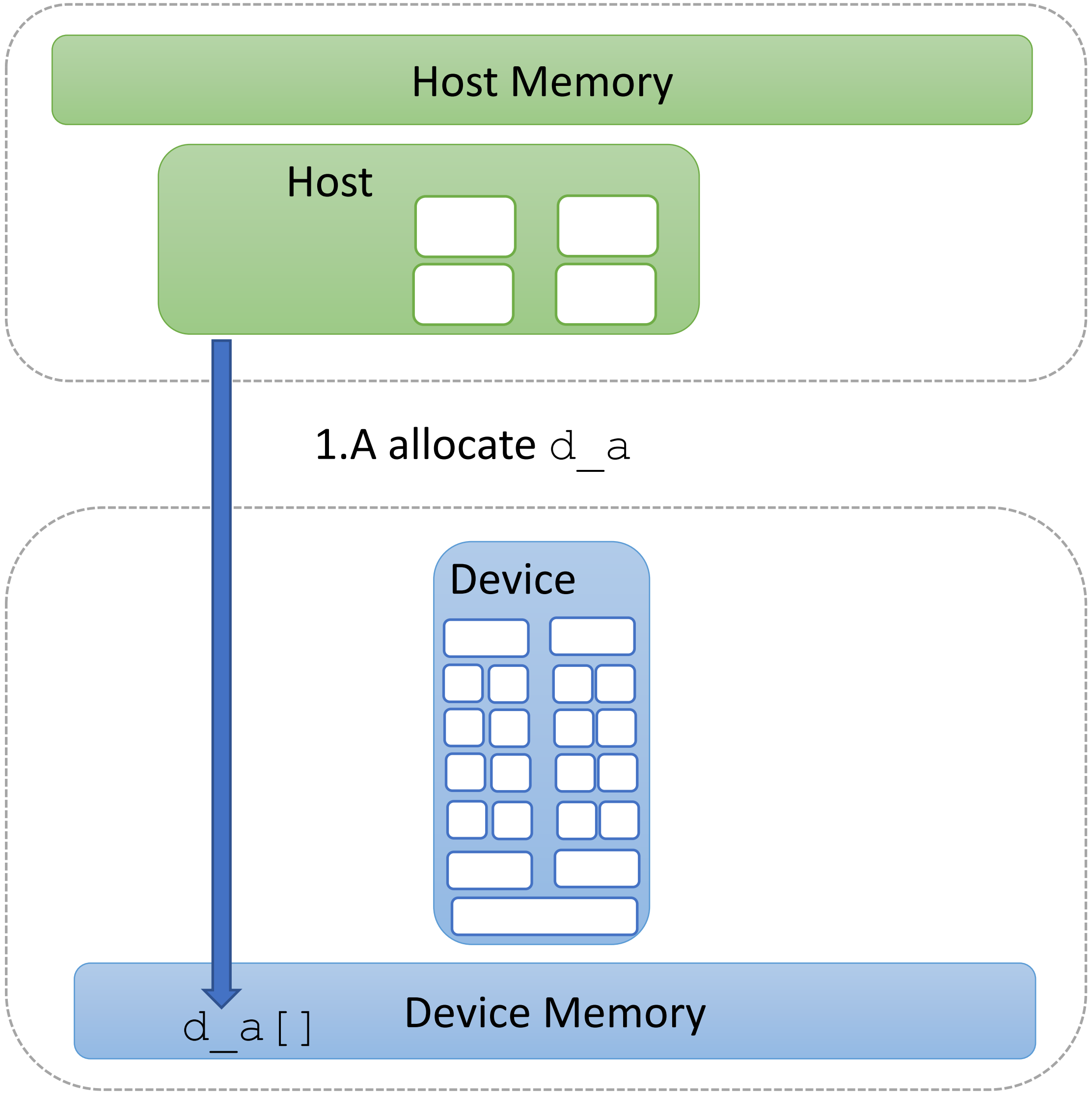
The first step is to allocate variables on the GPU memory:

```
cudaMalloc (void** devPtr, size_t size)
```

allocates `size` **bytes** of linear memory on the device and returns in `*devPtr` a pointer to the allocated memory.

Example: if we want to allocate an array called `a` of type `double` with 20 elements, simply we use

```
double *d_a = null; //declare pointer
cudaMalloc(&d_a, 20*sizeof(double));
```



Deallocate Memory on GPU

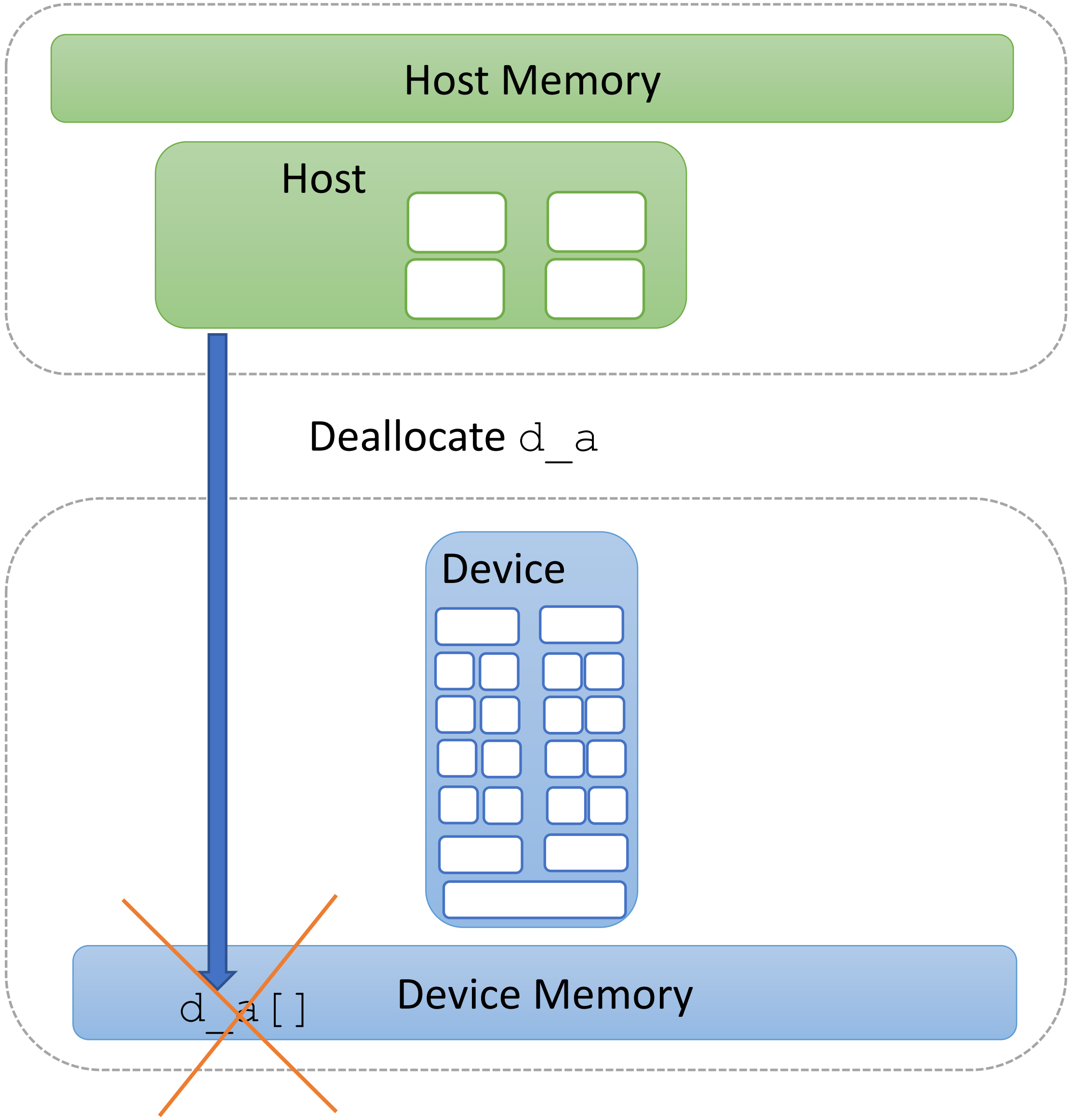
CUDA doesn’t provide automatic memory management (garbage collector), so remember to deallocate memory once you don’t need it anymore.

```
cudaFree(void* devPtr)
```

frees the device memory space pointed by `devPtr`, and no longer in use.

Example: if we want to deallocate `d_a`

```
cudaFree(d_a); // free the memory
```



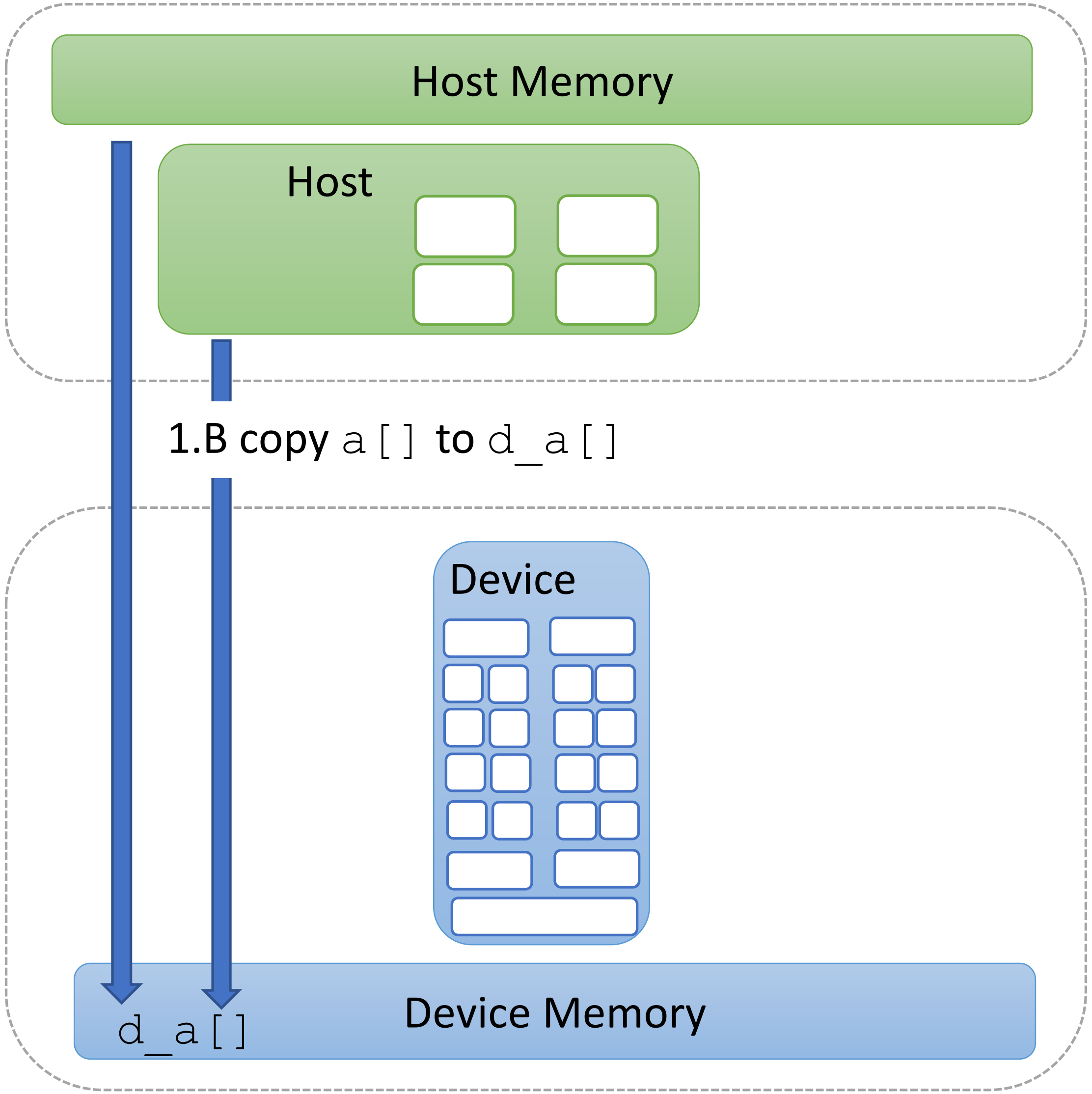
Move Data from CPU to GPU Memory

We can move data from CPU memory to GPU memory by copying data with

```
cudaMemcpy(void* dest, void* src, size_t
size,cudaMemcpyHostToDevice)
```

Example: copy `a`, on CPU memory, to `d_a`, on GPU memory

```
double *a = (double*) malloc(20*sizeof(double));
for(int u=0; i < 20; i++) a[i] = i*2.0;
cudaMemcpy(d_a, a,
20*sizeof(double),cudaMemcpyHostToDevice);
```



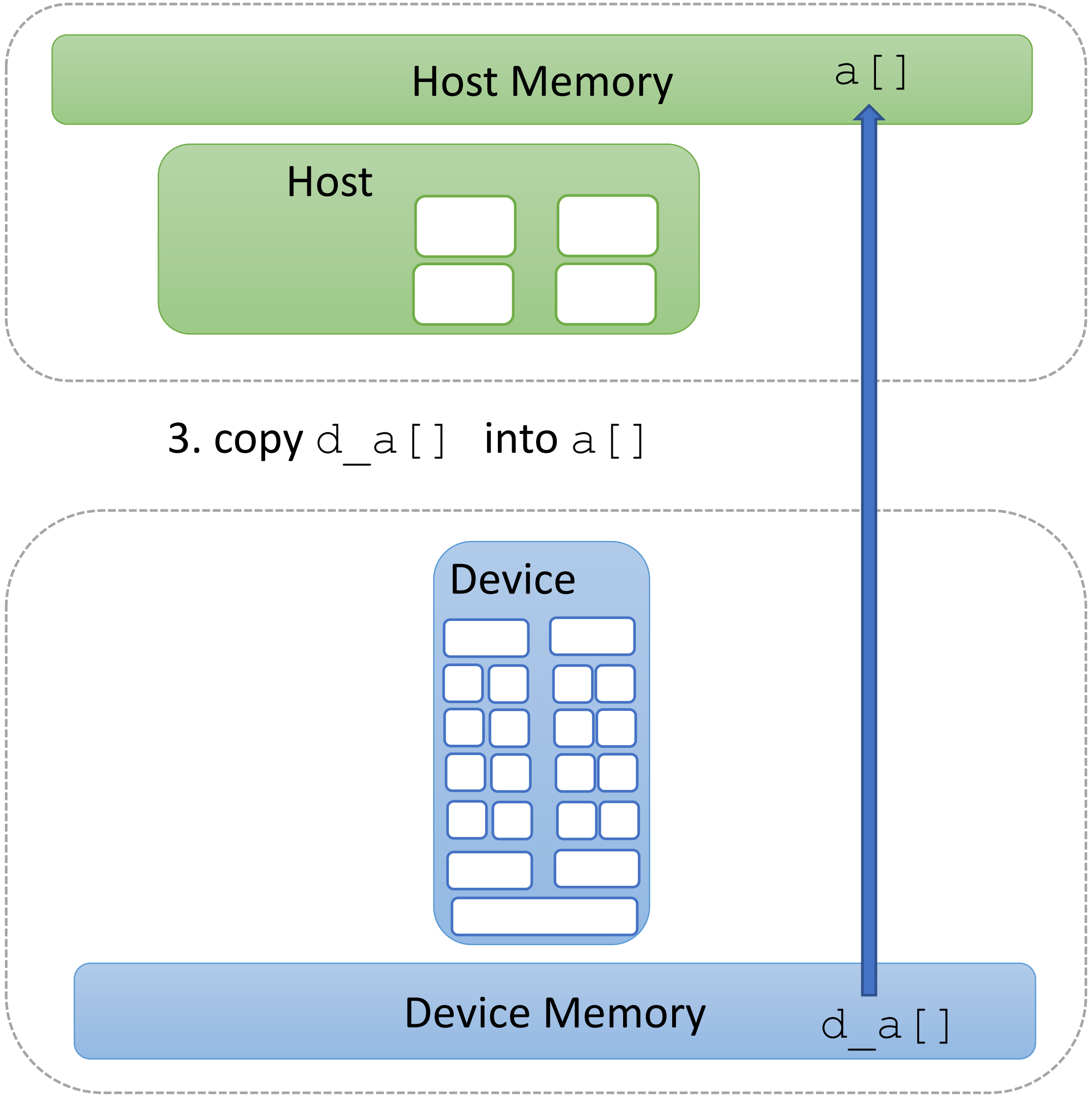
How do we move back data from GPU to CPU mem.?

We just change the value of the last argument of `cudaMemcpy`:

```
cudaMemcpy(void* dest, void* src, size_t
size, cudaMemcpyDeviceToHost)
```

Example: we want to move the values of `d_a[]` into `a[]`:

```
cudaMemcpy(a, d_a, 20*sizeof(double),
cudaMemcpyDeviceToHost)
```



To Summarize

- CUDA has its own jargon.
- When developing an application for GPUs, we follow the typical CUDA workflow: 1) move input data from CPU to GPU, 2) make computations on the GPU and 3) move the output results data back to CPU.
- To allocate memory on the GPU, CUDA provides `cudaMalloc()`, similar to C `malloc()`
- We move data from/to GPU by copying data with `cudaMemcpy()` from/to GPU memory location to/from a CPU memory location.