

# GPU Teaching Kit

## Accelerated Computing

### Lecture 2.6 - Introduction to CUDA C Unified Memory

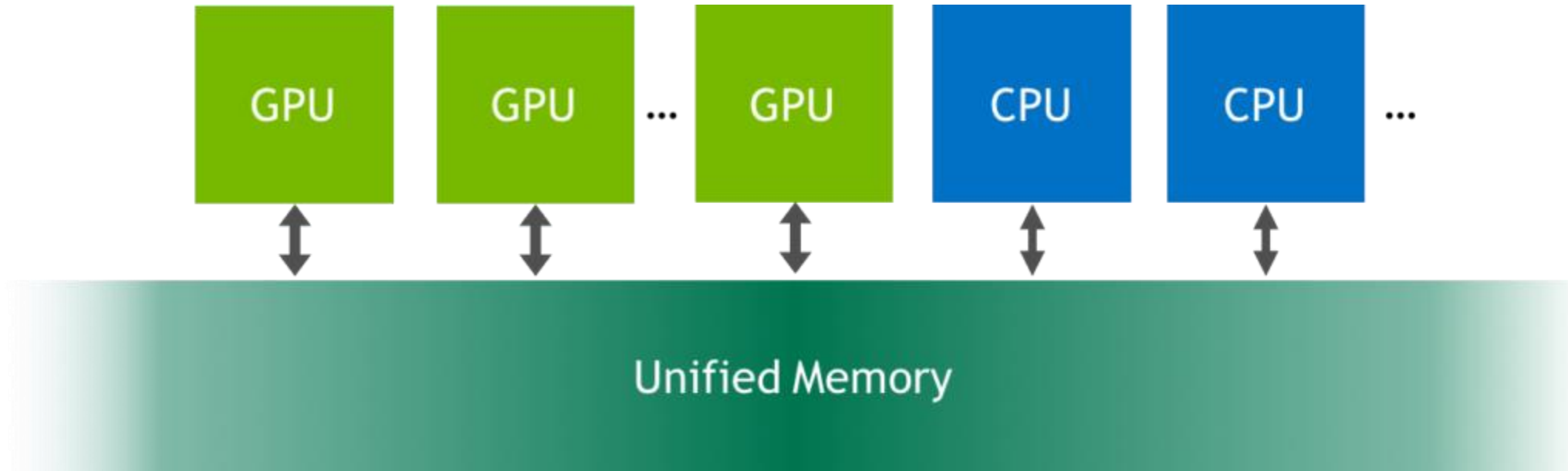


# Objective

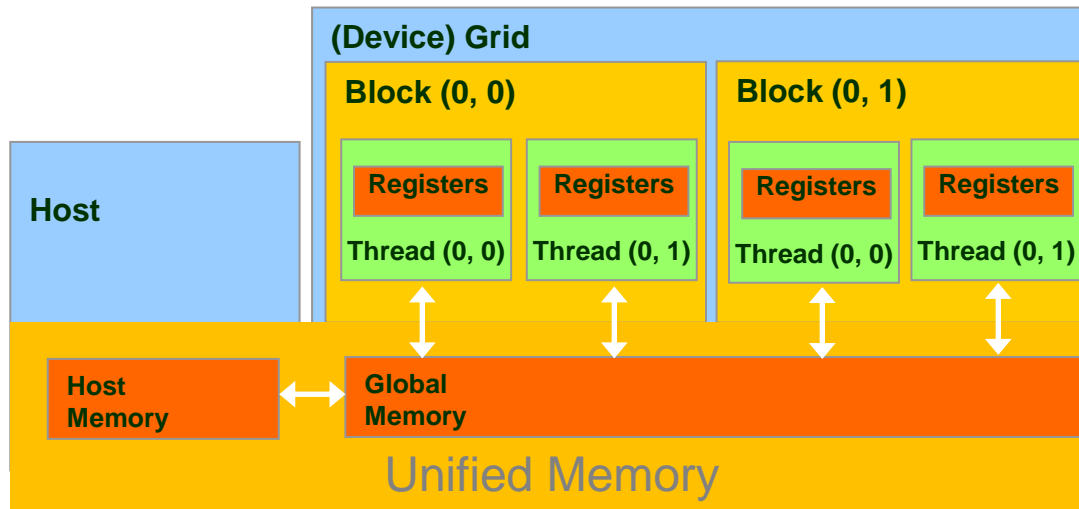
- To learn the basic API functions in CUDA host code for CUDA Unified Memory
  - Unified Memory Allocation
  - Data Transfer in Unified Memory

# CUDA Unified Memory (UM)

- Is a single memory address space accessible both from the host and from the device.
- The hardware/software handles automatically the data migration between the host and the device maintaining consistency between them.

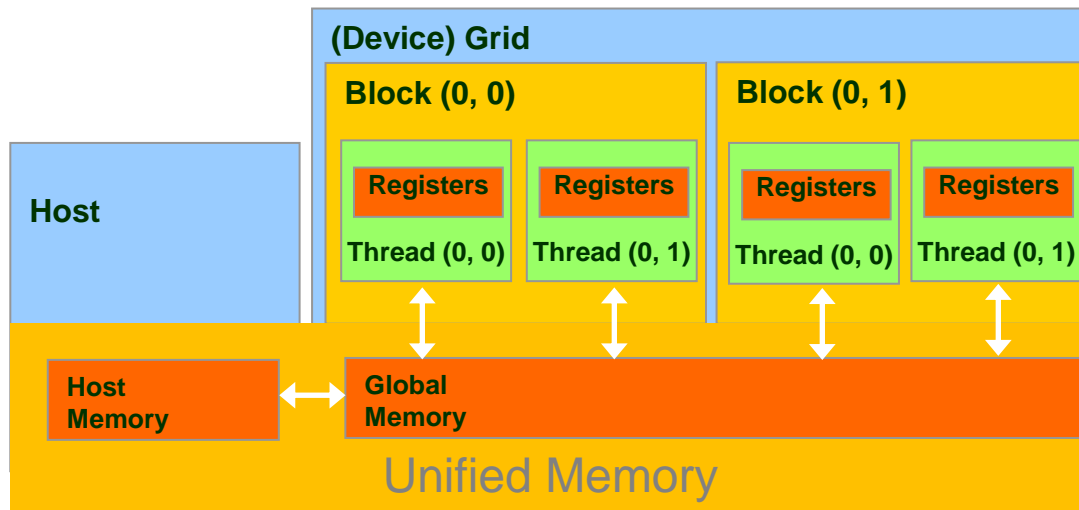


# Partial Overview of CUDA Memories



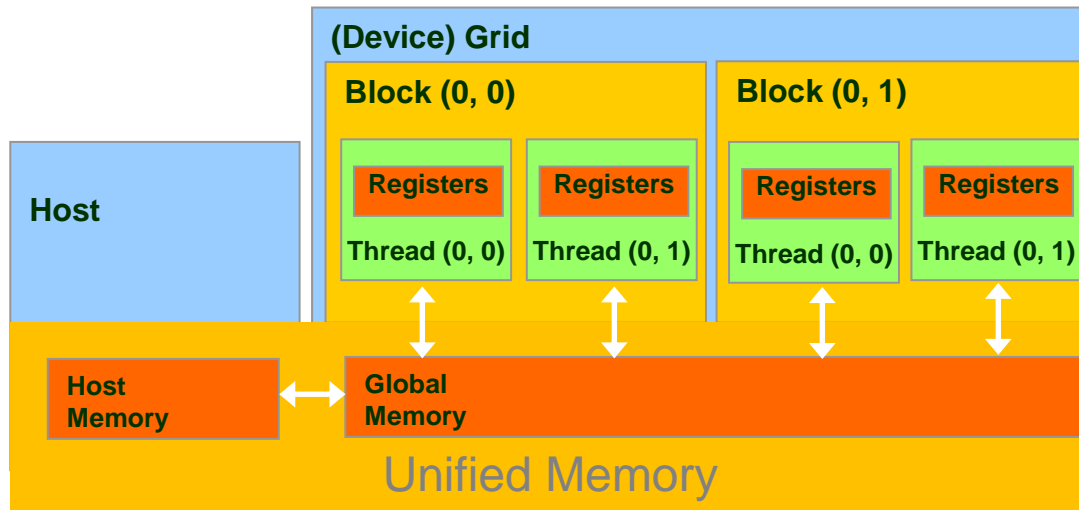
- Device code can:
  - R/W per-thread registers
  - R/W all-shared global memory
  - R/W managed memory (Unified Memory)
- Host code can
  - Transfer data to/from per grid global memory
  - R/W managed memory

# Partial Overview of CUDA Memories



- `cudaMallocManaged()`
  - Allocates an object in the Unified Memory address space.
  - Two parameters, with an optional third parameter.
  - Address of a pointer to the allocated object
  - Size of the allocated object in terms of bytes
  - [Optional] Flag indicating if memory can be accessed from any device or stream
- `cudaFree()`
  - Frees object from unified memory.
  - One parameter
    - Pointer to freed object

# Partial Overview of CUDA Memories



- `cudaMemcpy()`
  - Memory data transfer
  - Requires four parameters
    - Pointer to destination
    - Pointer to source
    - Number of bytes copied
    - Type/Direction of transfer
  - Depending on the transfer type, the driver may decide to use the memory on the host or the device.
  - In Unified Memory this function is utilized to copy data between different arrays, regardless of position.

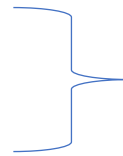
# Putting it all together, vecAdd CUDA host code using Unified Memory

```
int main() {
```

```
    float *m_A, float *m_B, float *m_C, int n;
```

```
    int size = n * sizeof(float);
```

```
    cudaMallocManaged((void**) &m_A, size);  
    cudaMallocManaged((void**) &m_B, size);  
    cudaMallocManaged((void**) &m_C, size);
```



Allocation of Managed Memory

```
    // Memory initialization on the Host
```



m\_A, m\_B gets initialized on the host

```
    // Kernel invocation code - to be shown later
```



The device performs the actual vector addition

```
    cudaFree(m_A); cudaFree(m_B); cudaFree(m_C);
```

```
}
```

# CUDA Unified Memory for different architectures

## Prior to compute capability 6.x

- There is no specialized hardware units to improve UM efficiency.
- For data migration the full memory block needs to be copied synchronically by the driver.
- No memory oversubscription.

## Compute capability 6.x onwards

- There are specialized hardware units managing page faulting.
- Data is migrated on demand, meaning that data gets copied only on page fault.
- Possibility to oversubscribe memory, enabling larger arrays than the device memory size.

# GPU Teaching Kit

Accelerated Computing

The GPU Teaching Kit is licensed by NVIDIA under the [Creative Commons Attribution-NonCommercial 4.0 International License](#).

