
AMD
Fusion 11
DEVELOPER SUMMIT

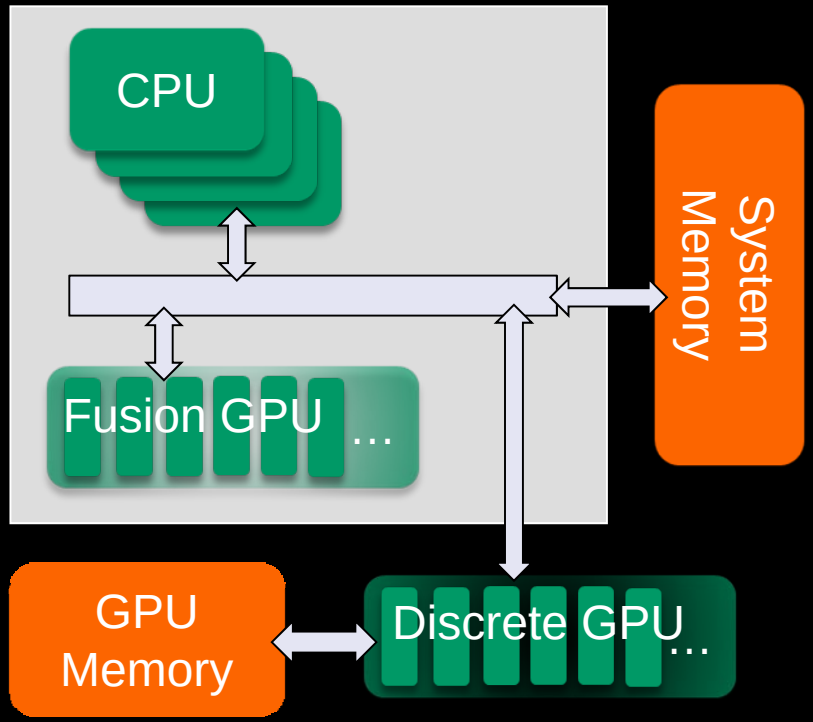
INTRODUCTION TO OPENCL™

A Beginner's Tutorial

Udeepta Bordoloi
AMD

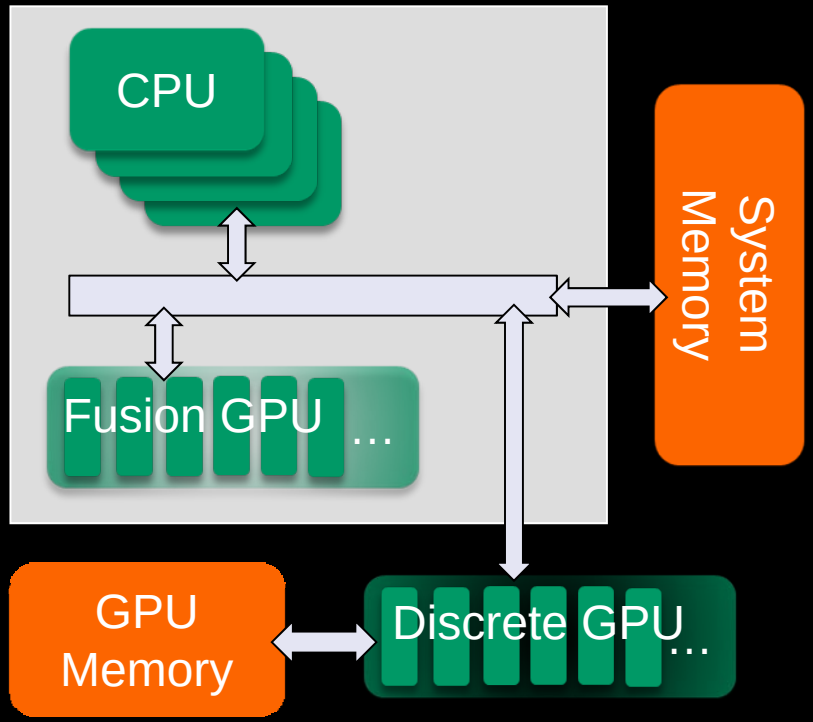
IT'S A HETEROGENEOUS WORLD

- Heterogeneous computing
  - The new normal
- Many CPU's – 2, 4, 8, …
- Very many GPU processing elements – 100's
- Different vendors, configurations, architectures
- The multi-million dollar question
  - How do you avoid developing and maintaining different source code versions?



HETEROGENEOUS SYSTEM CONSIDERATIONS

- CPU is great for serial tasks
  - Lower throughput, lower latency
- Discrete GPU excels at data parallel problems
  - High ALU, high memory bandwidth, higher latency
  - Bandwidth in the order or hundred of GB/s
  - Transfer over PCIe®
- Fusion GPU
  - DX11 class, shares system memory with CPU
  - Bandwidth in the order or tens of GB/s
  - Zero Copy
- Which parts of your code should run on which device?
- Where do you keep your data?
- When to communicate and synchronize between CPU and GPU?



WHAT IS OPENCL™

- Framework for programming on heterogeneous systems
  - Multi-core CPUs
  - Massively parallel GPUs
  - Cell, FPGAs etc
- Industry standard
- Open specification
- Cross-platform
  - Windows®, Linux®, Mac OS
- Multi-vendor
  - AMD, Apple, Creative, IBM, Imagination, Intel, NVIDIA, Samsung

OPENCL: OVERVIEW

How to execute a program on the device (GPU)?

- Kernel
  - Performs GPU calculations
  - Reads from, and writes to memory
- Based on C
  - Restrictions
    - No recursion, etc.
  - Additions
    - Vector data types (int 4)
    - Synchronization
    - Built in functions (sin, log)

How to control the device (GPU)

- Host Program
  - C API
- Steps
  1. Initialize the GPU
  2. Allocate memory buffers on GPU
  3. Send data to GPU
  4. Run Kernel on GPU
  5. Read data from GPU
- Commands are queued

KERNEL

EXPOSING PARALLELISM

C function

```
for (int i = 0; i < 24; i++)
{
        Y[i] = a*X[i] + Y[i];
}
```

- Serial execution, one iteration after the other

EXPOSING PARALLELISM

C function

```
for (int i = 0; i < 24; i++)
{
        Y[i] = a*X[i] + Y[i];
}
```

- Serial execution, one iteration after the other

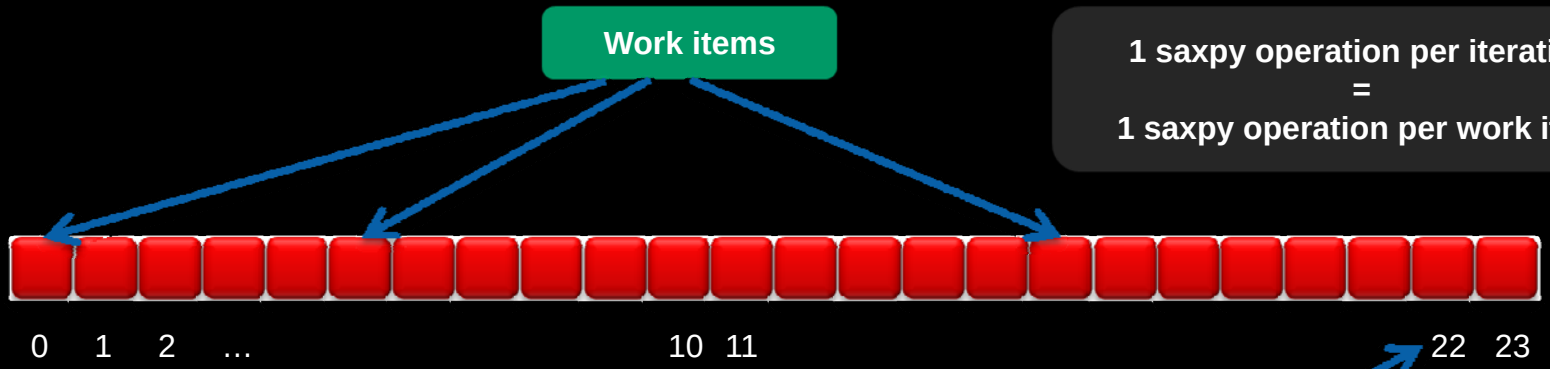
OpenCL kernel

```
__kernel void
saxpy(const __global float * X,
            __global float * Y,
                  const float a)
{
        uint i = get_global_id(0);
        Y[i] = a* X[i] + Y[i];
}
```

- Parallel execution, multiple iterations at the same time

WORK ITEM

- Think of work item as a parallel “thread” of execution

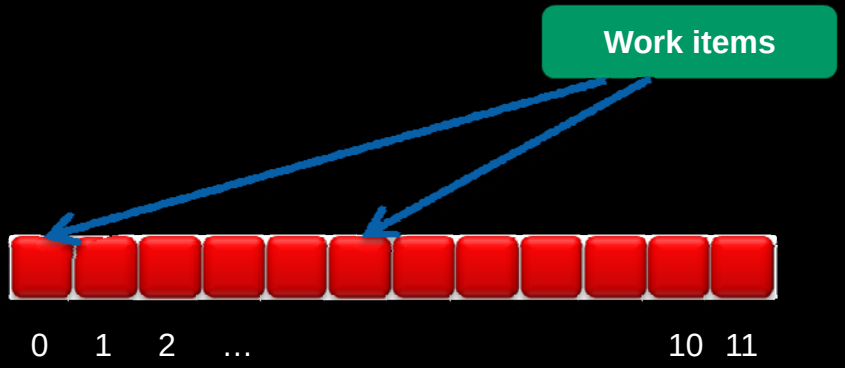



```
for (int i = 0; i < 24; i++)
{
        Y[i] = a*X[i] + Y[i];
}
```

```
{
uint i = get_global_id(0);
Y[i] = a* X[i] + Y[i];
}
```

ITERATIONS → WORK ITEMS

- Iterations can become work items (if parallelizable)



2 saxpy operations per iteration
=
2 saxpy operations per work item

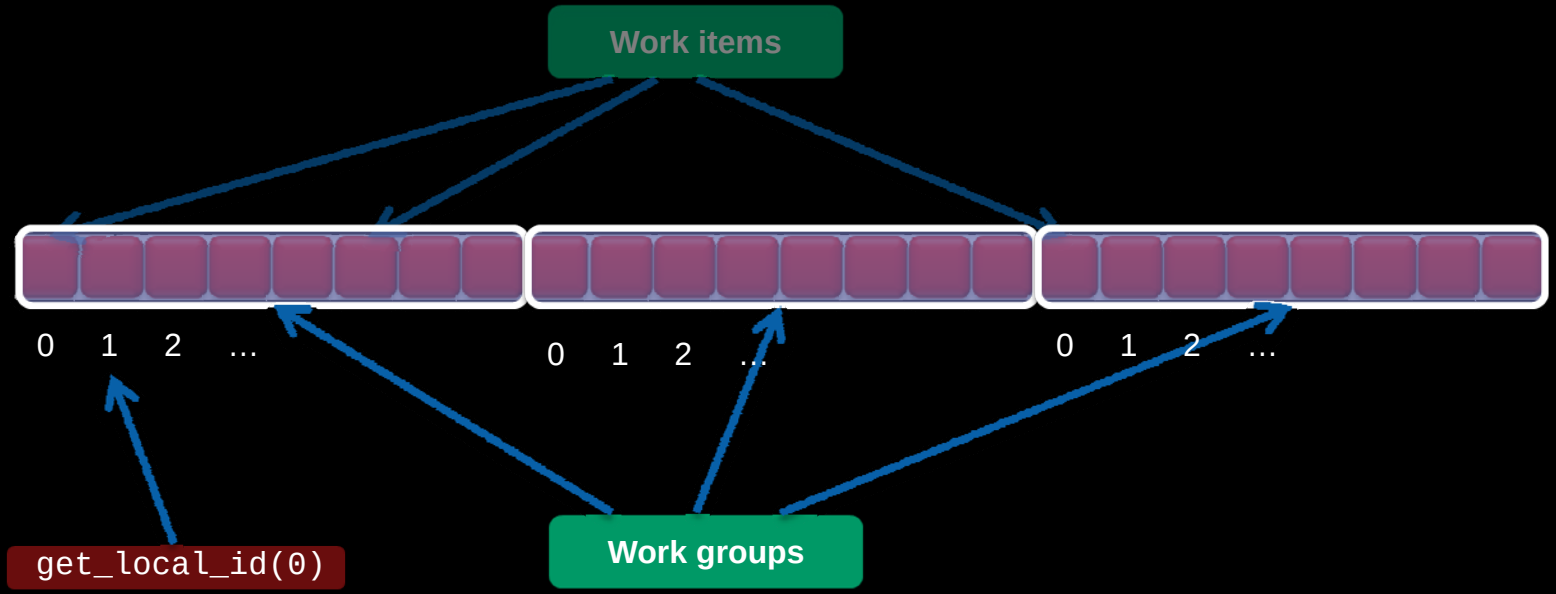
```
for (int i = 0; i < 12; i++)
{
        j = 2 * i;
        Y[j]   = a*X[j] + Y[j];
        Y[j+1] = a*X[j+1] + Y[j+1];
}
```

```
{
        uint j = 2 * get_global_id(0);
        Y[j]   = a*X[j] + Y[j];
        Y[j+1] = a*X[j+1] + Y[j+1];
}
```

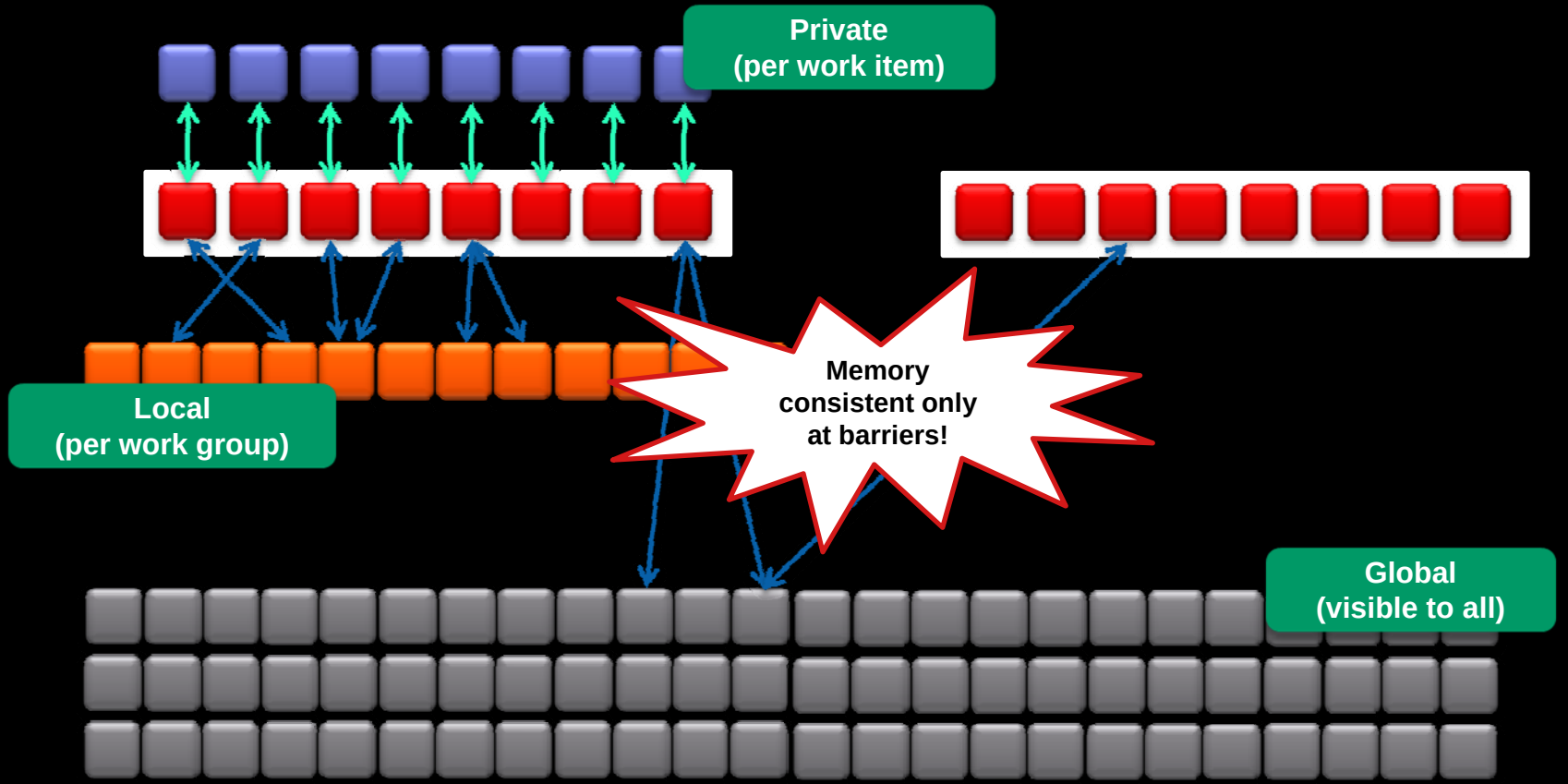
WORK GROUP

Divide the execution domain into groups

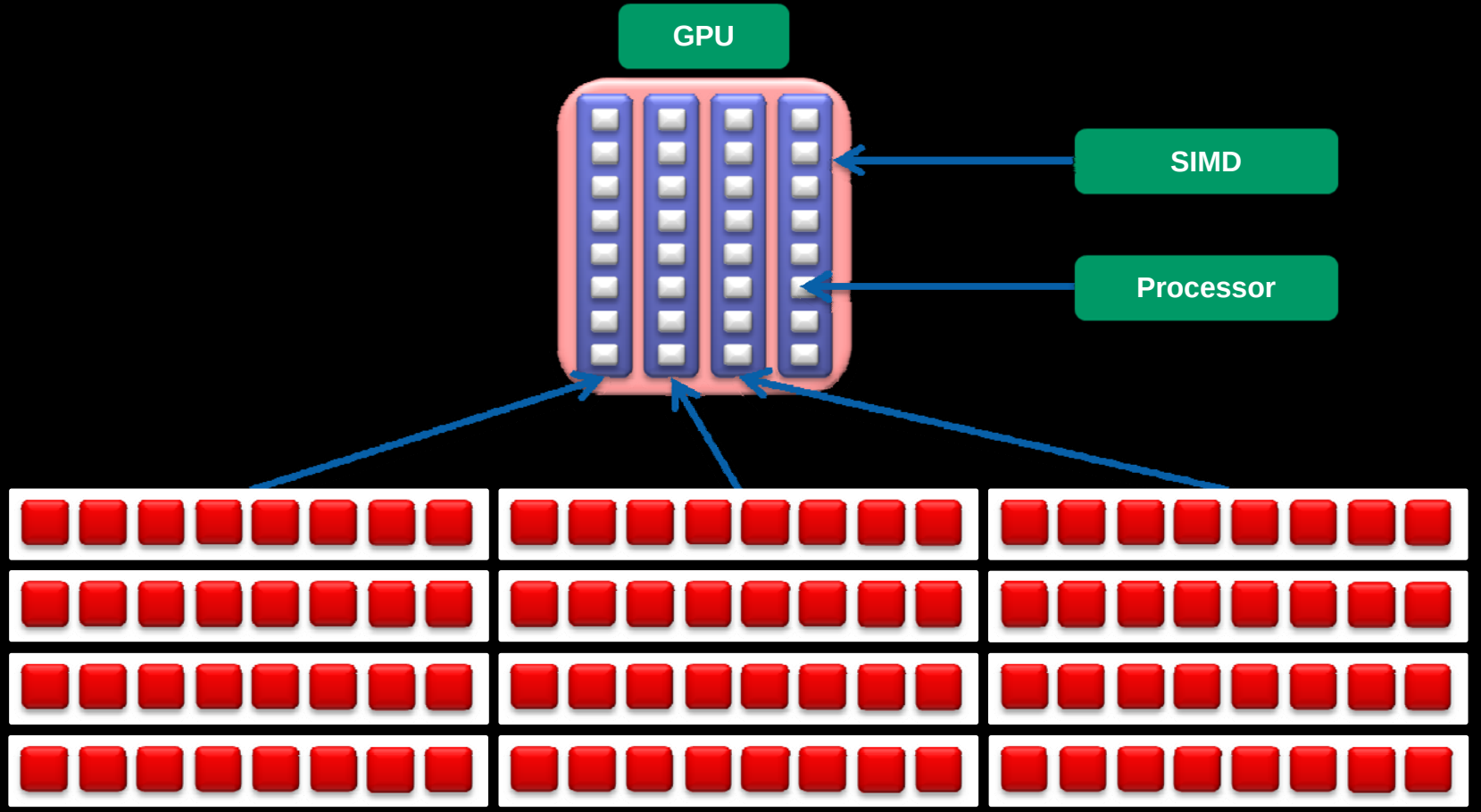
Can exchange data and synchronize inside a group



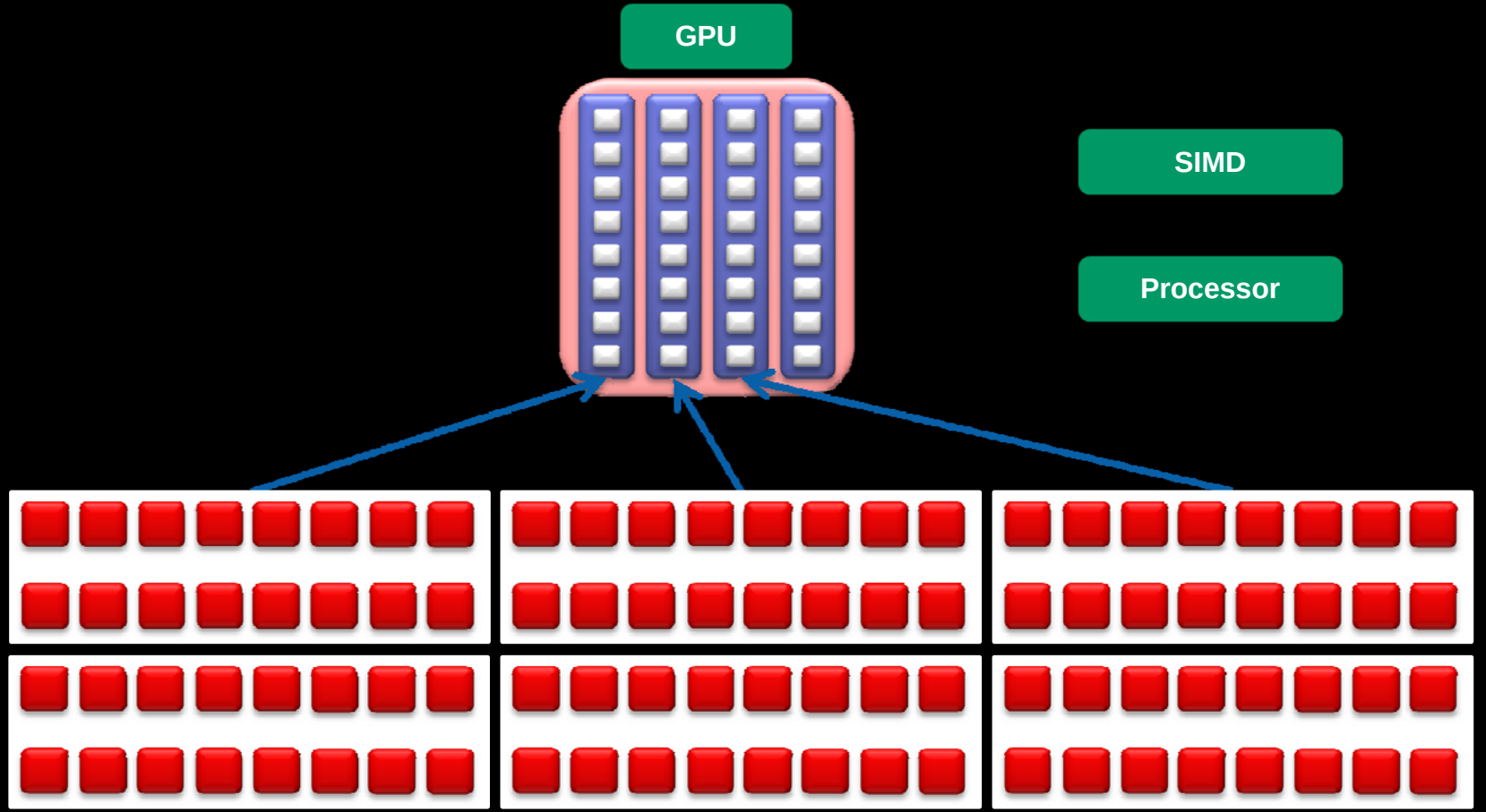
MEMORY SPACES



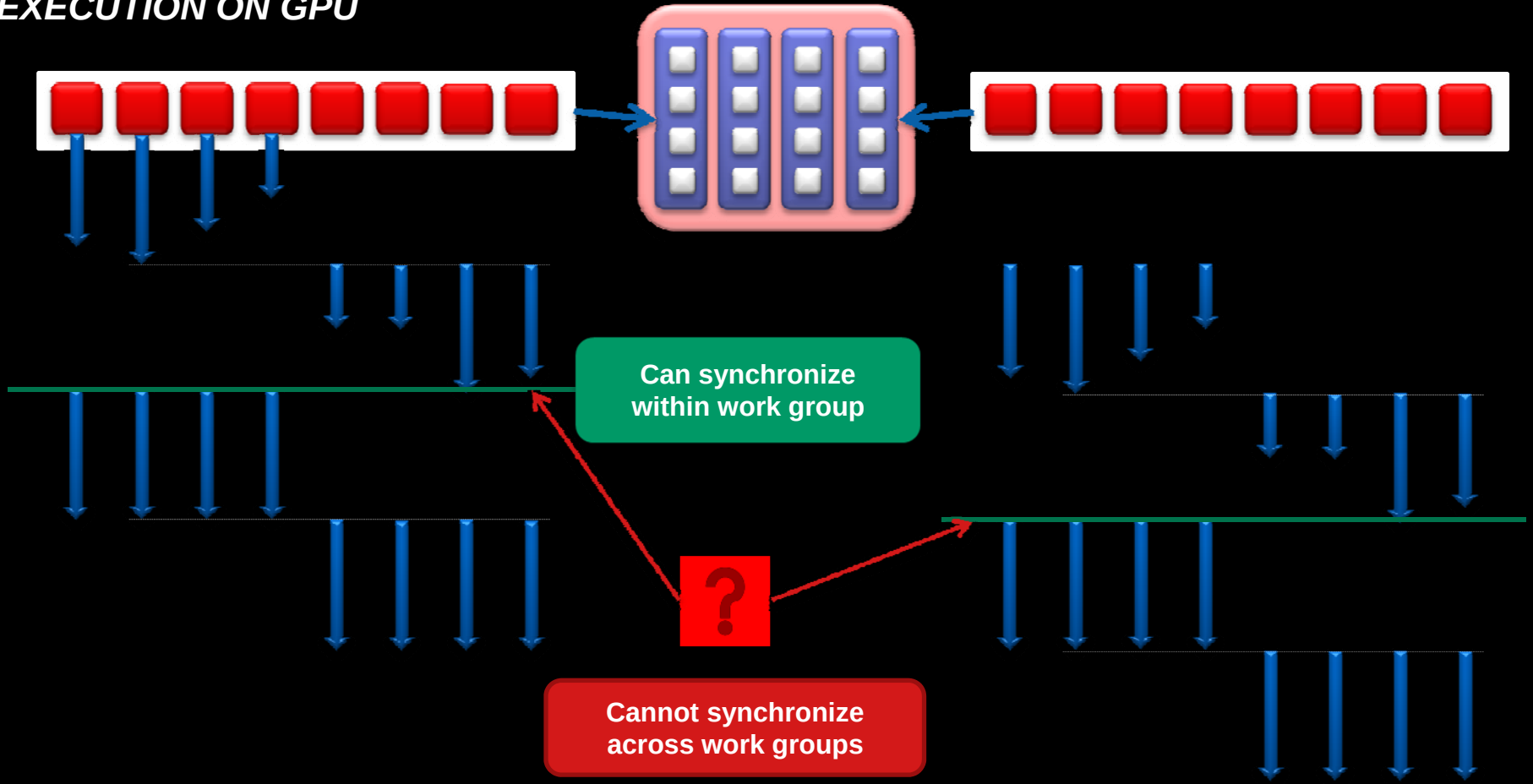
WORK GROUPS ON GPU



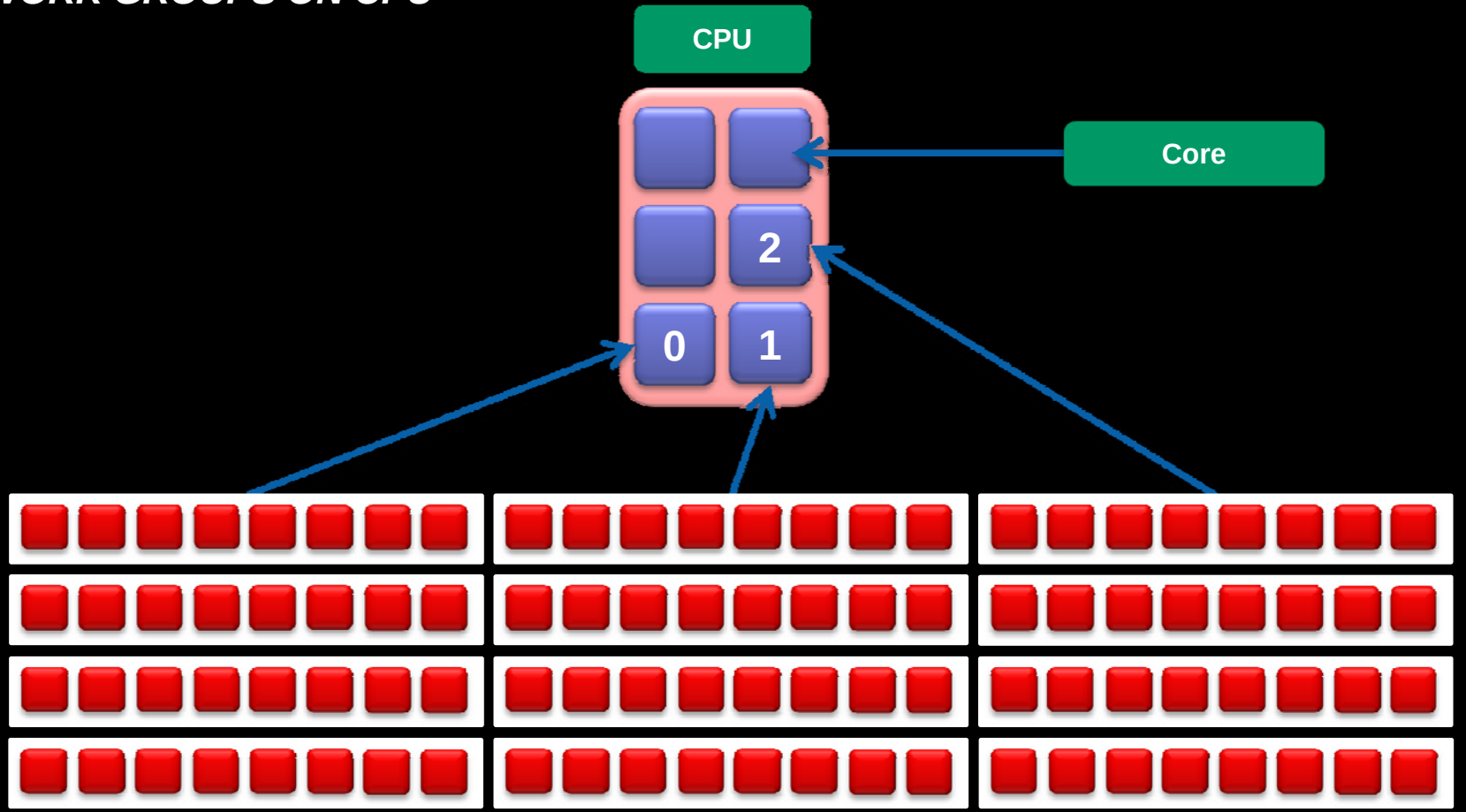
WORK GROUPS ON GPU



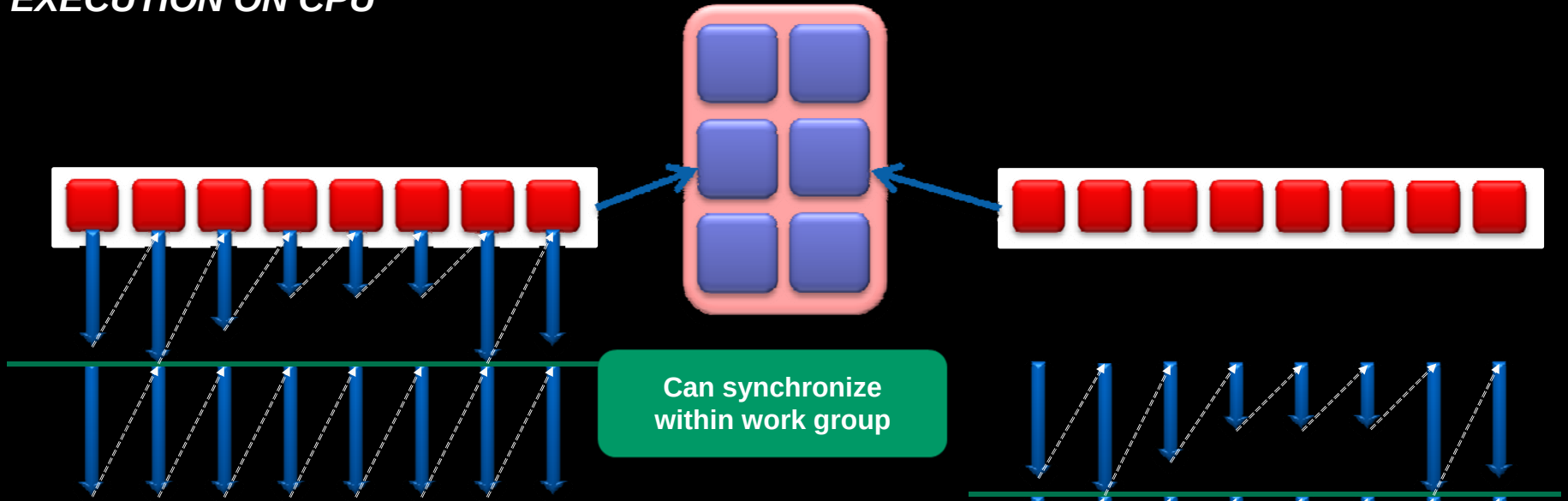
EXECUTION ON GPU



WORK GROUPS ON CPU

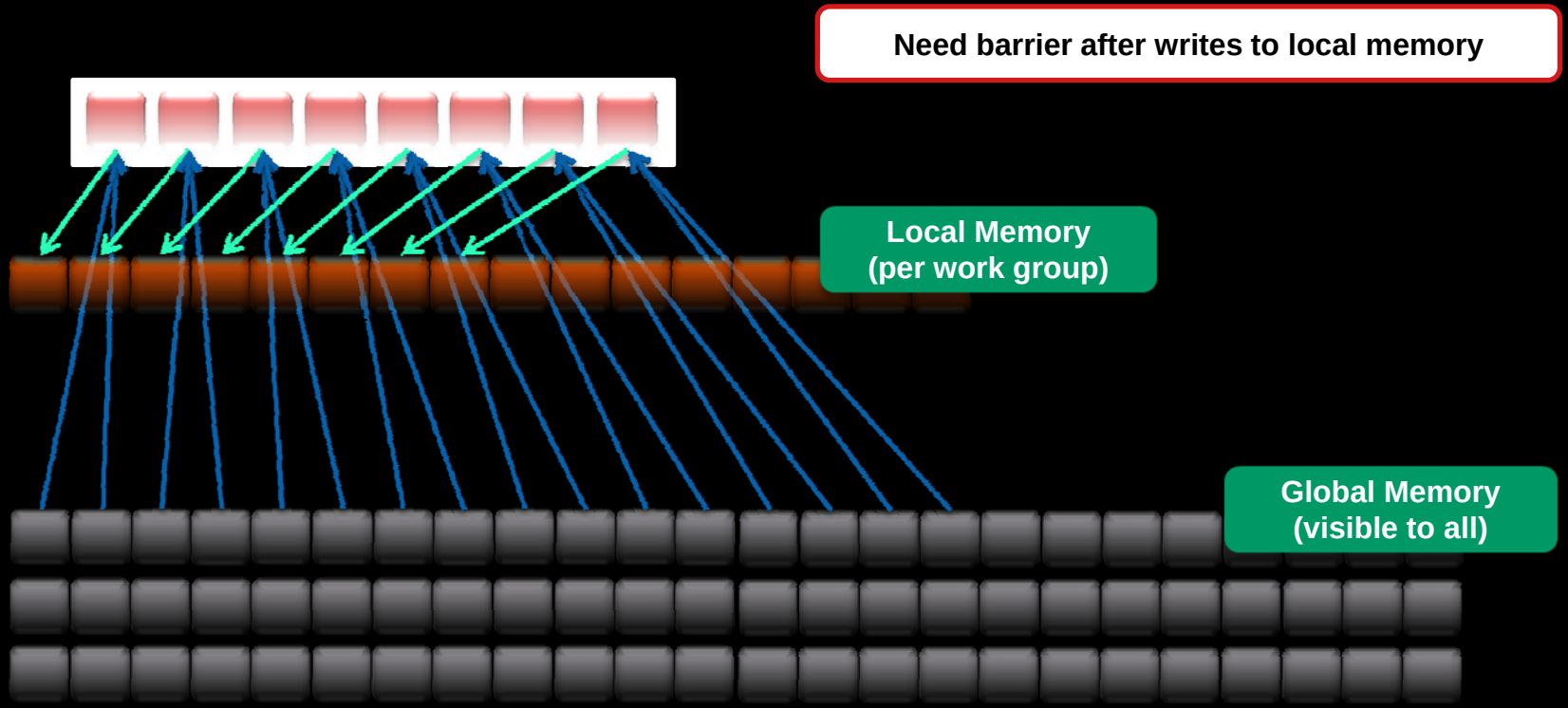


EXECUTION ON CPU



REDUCTION EXAMPLE

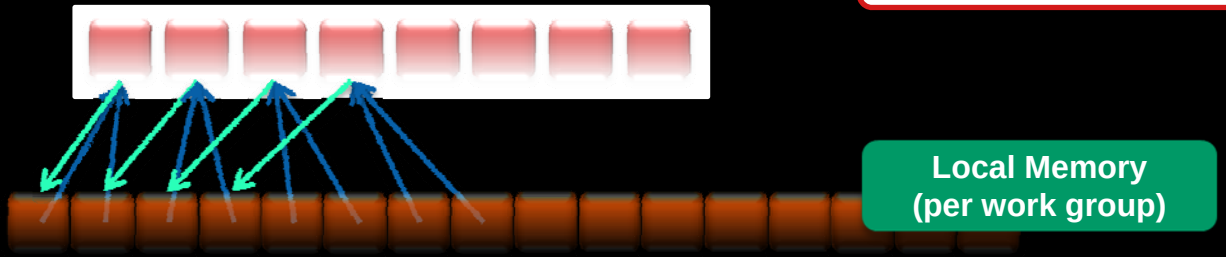
WITHIN EACH WORK GROUP



WITHIN EACH WORK GROUP

Need barrier after reads from local memory

Need barrier after writes to local memory

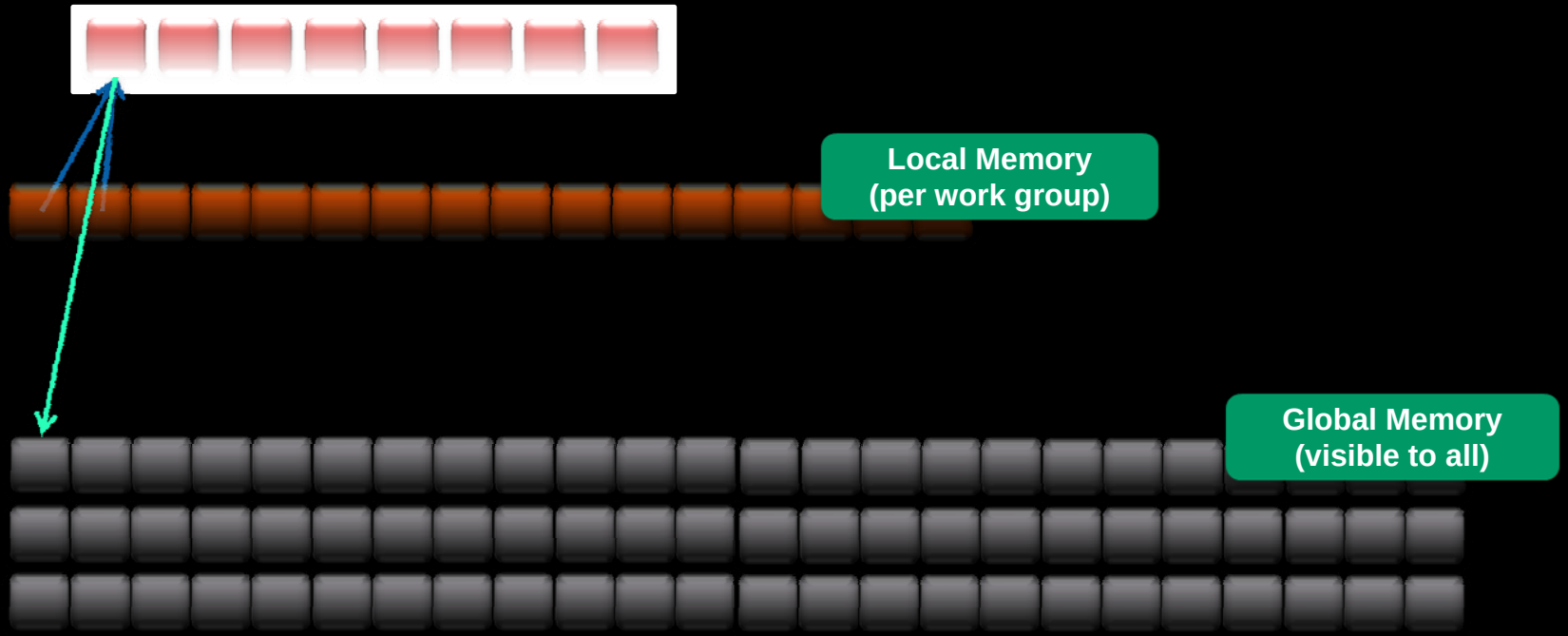


WITHIN EACH WORK GROUP

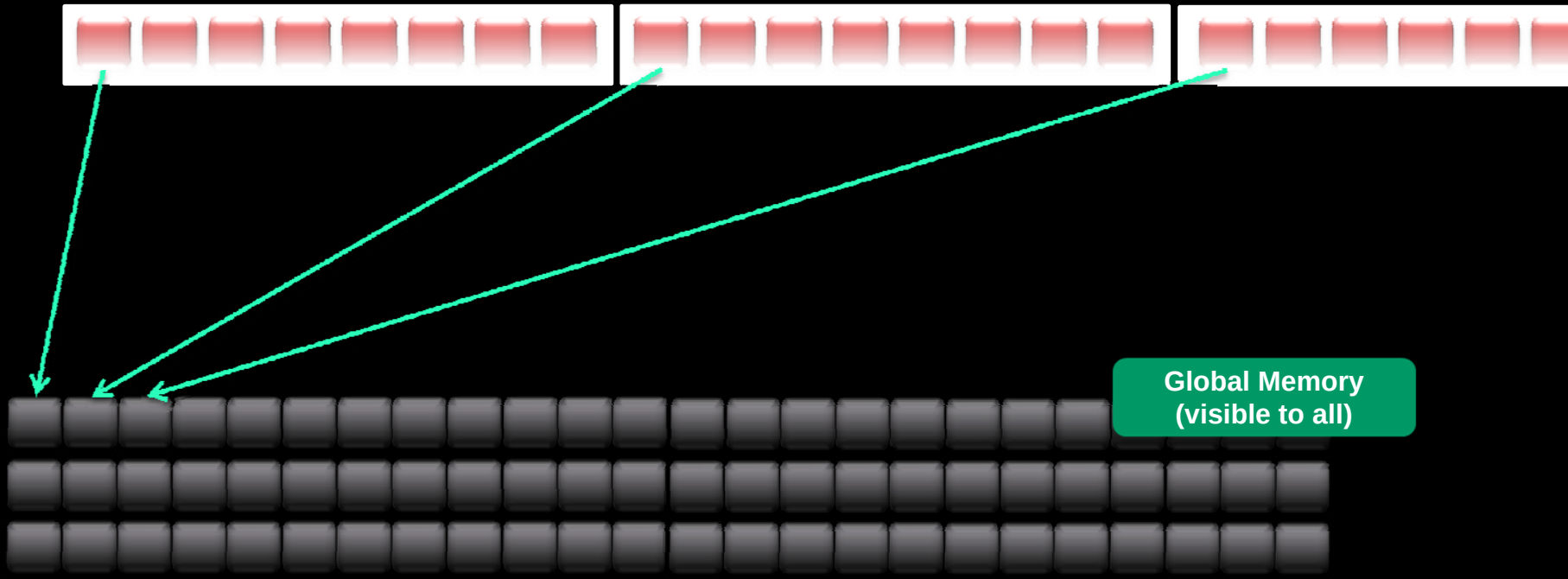
Need barrier after reads from local memory

Need barrier after writes to local memory

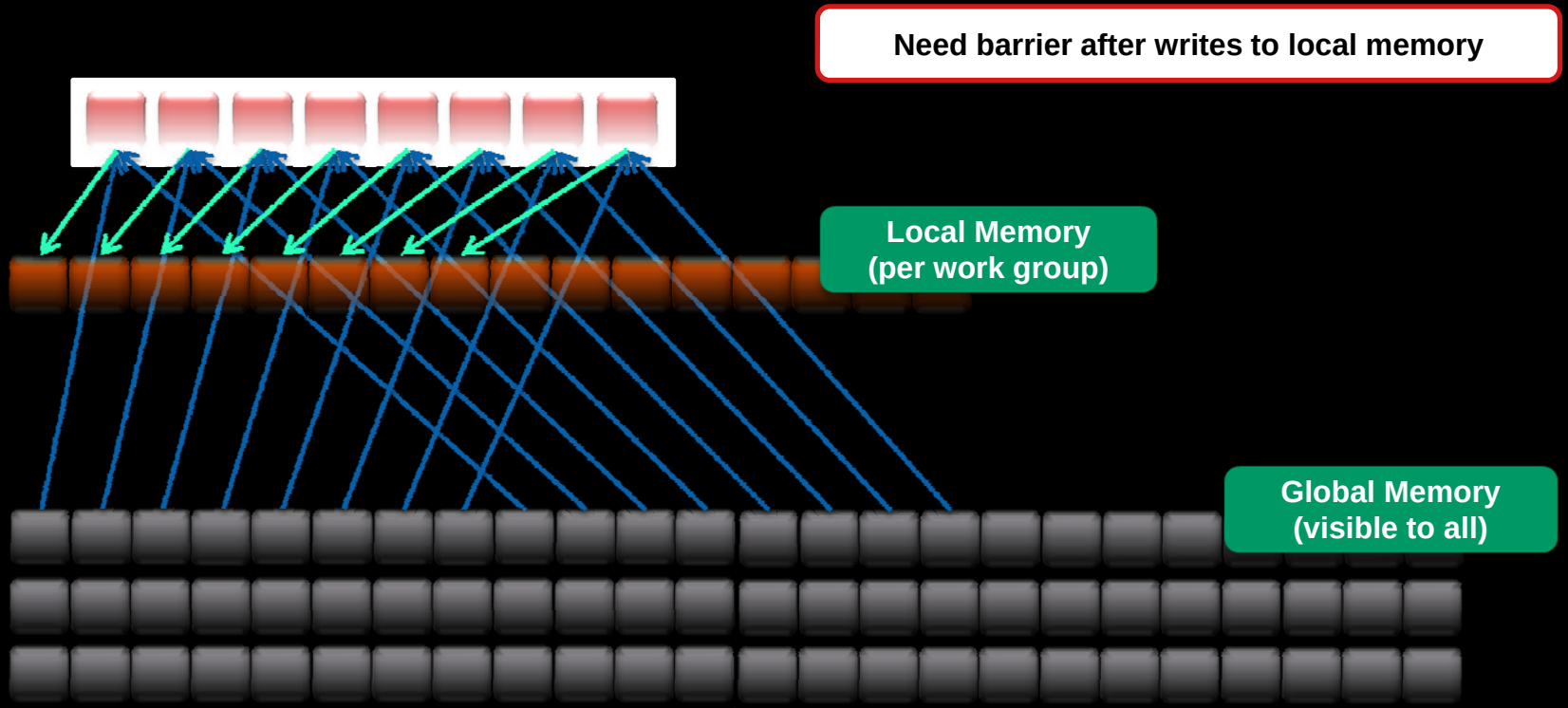
WITHIN EACH WORK GROUP



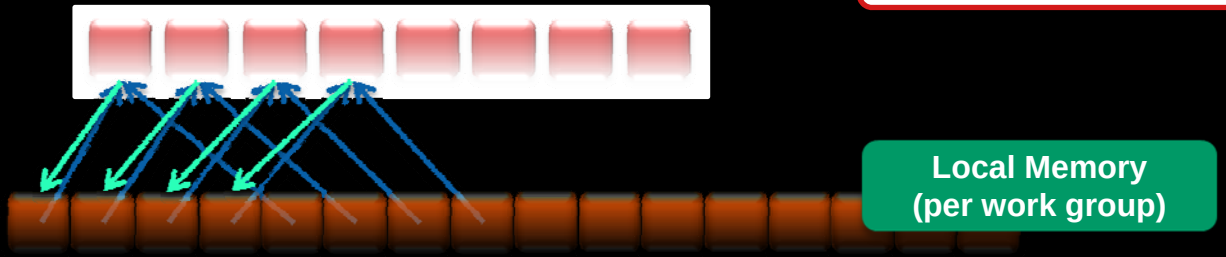
ACROSS WORK GROUPS



WITHIN EACH WORK GROUP



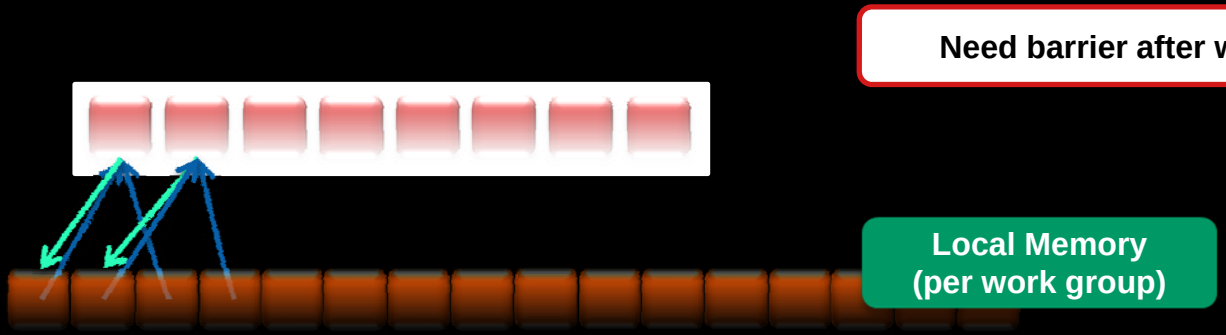
WITHIN EACH WORK GROUP



Need barrier after writes to local memory

Local Memory
(per work group)

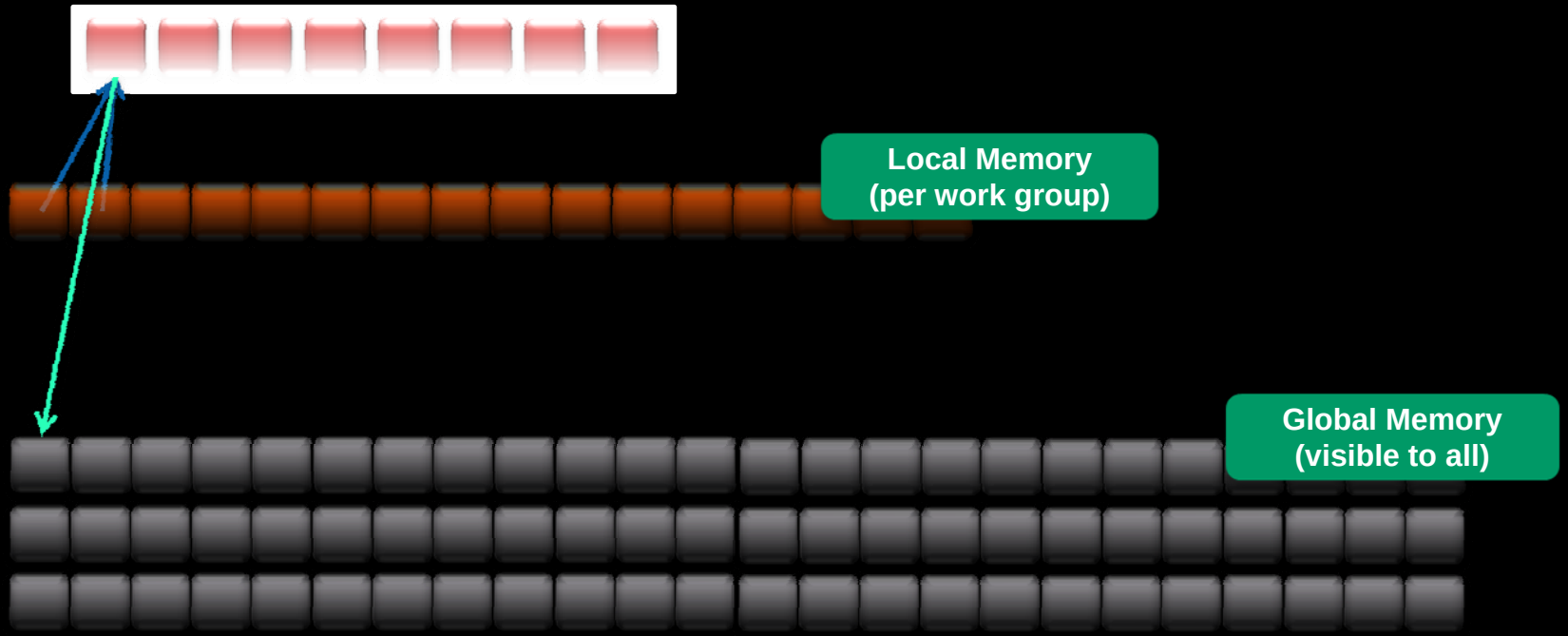
WITHIN EACH WORK GROUP



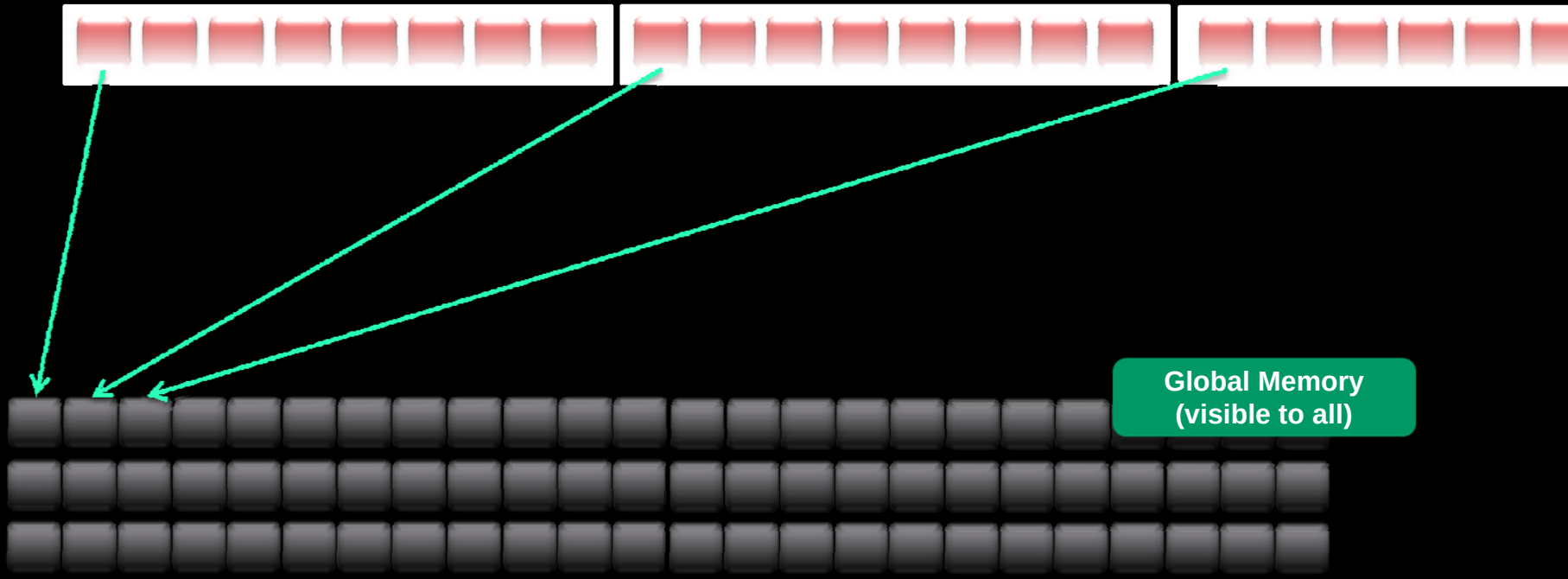
Need barrier after writes to local memory

Local Memory
(per work group)

WITHIN EACH WORK GROUP



ACROSS WORK GROUPS



HOST PROGRAM

COMMAND QUEUE

- Enables asynchronous (non-blocking) exection of OpenCL commands
  - Look for OpenCL commands clEnqueue…()
- Accepts:
  - Kernel execution commands
  - Memory commands
  - Synchronization commands
- In-order queue
  - Commands complete before next command starts
- Out-of-order queue
  - Programmer responsibility to synchronize command execution

HOST PROGRAM: BASIC SEQUENCE FOR A GPU DEVICE

- Initialization
  - Find the GPU
  - Initialize the GPU
  - Compile the program for GPU (kernel)
- Memory
  - Create input, output buffers on the GPU
  - Copy data from CPU memory to GPU memory
- Execution
  - Run **kernel** on the GPU
  - Run multiple kernels if needed
  - Wait till GPU is finished
- Memory
  - Copy data from GPU memory to CPU memory

QUESTIONS

Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

OpenCL is a trademark of Apple Inc. used with permission by Khronos.

BACKUP