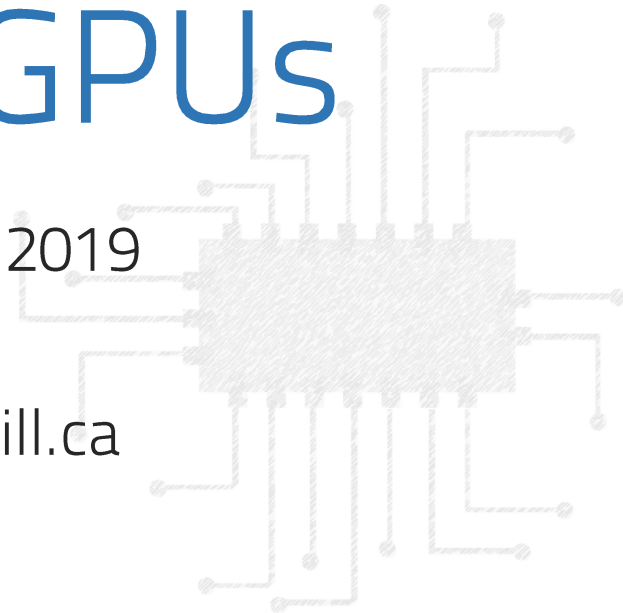


Special Topic: GPUs

COMP 520: Compiler Design 2019

Alexander Krolik

alexander.krolik@mail.mcgill.ca



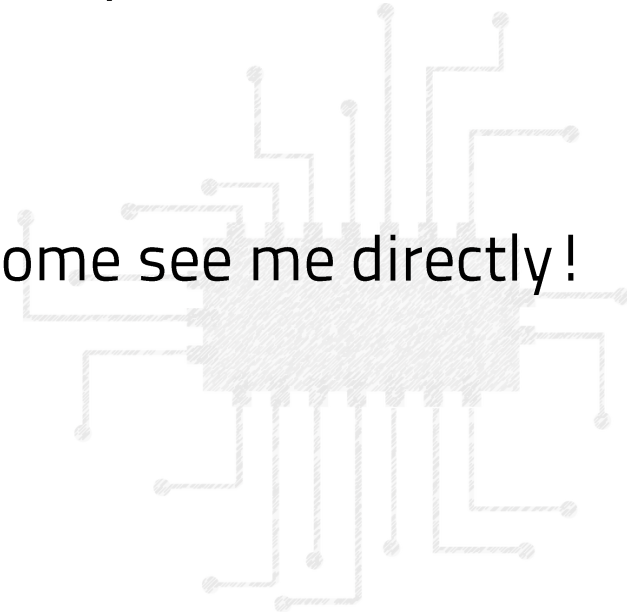


Announcements

- **Milestone 4:** Wednesday, April 10
- **Final report:** Friday, April 12 (or extended)
- **Peephole:** Friday, April 12

- **Group Meeting:** Week of April 8th (or extended)
- **Exam:** April 18th 2PM

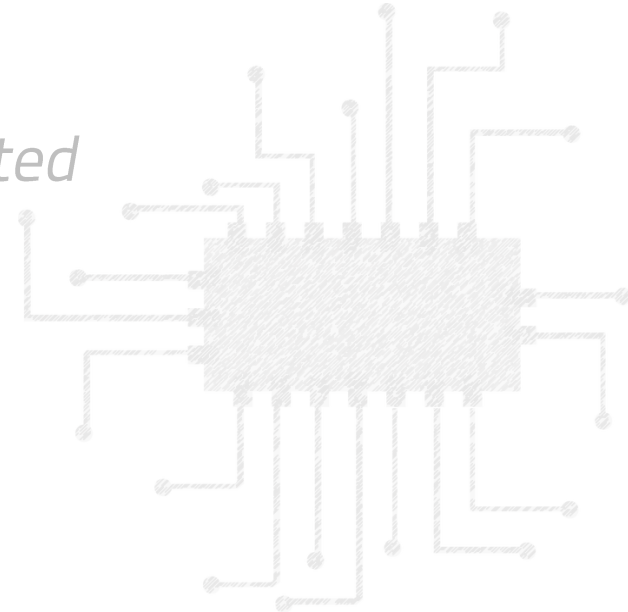
- For immediate feedback on Milestone 3, come see me directly!





Plan

- What is a GPU?
 - Programming model/paradigm
 - Hardware architecture
- Thread and memory management
- GPU algorithms
 - Introductory
 - Reductions
- *PTX (Parallel Thread eXecution) – not tested*





What is a GPU?

A *Graphics Processing Unit (GPU)* is:

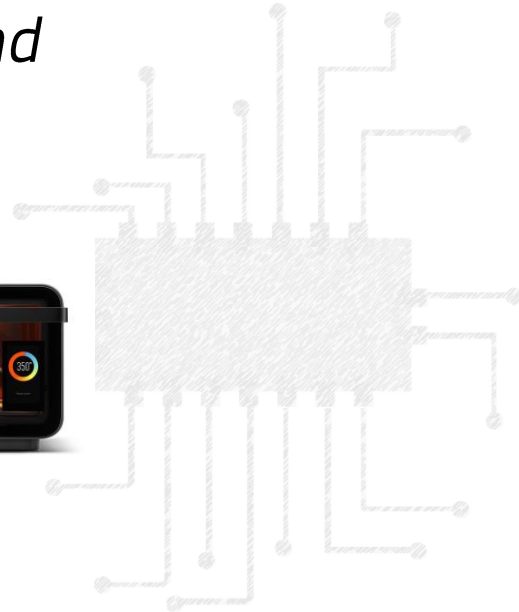
- A specialized processor originally designed for graphics operations;
- Targets throughput computing – i.e. ALL the pixels;
- Works in collaboration with the CPU;
- Also called an *accelerator* or *co-processor*; and
- Can be found in many modern devices.



[ZDNet](#)



[June Oven](#)

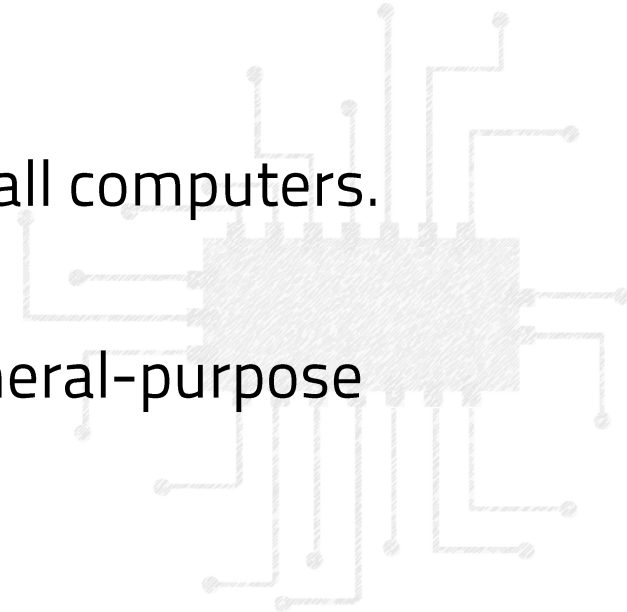




History of GPUs

- Introduced in the 1970's;
- Originally contained hardware for graphics only;
- Transitioned to generic throughput processors in mid 2000's;
 - "General-purpose" GPUs;
- Have been pushed by 2 industries:
 - Gaming;
 - Machine learning; and
- From 2010 onwards, a key component in all computers.

Idea: Transition towards GPU power for general-purpose throughput computation





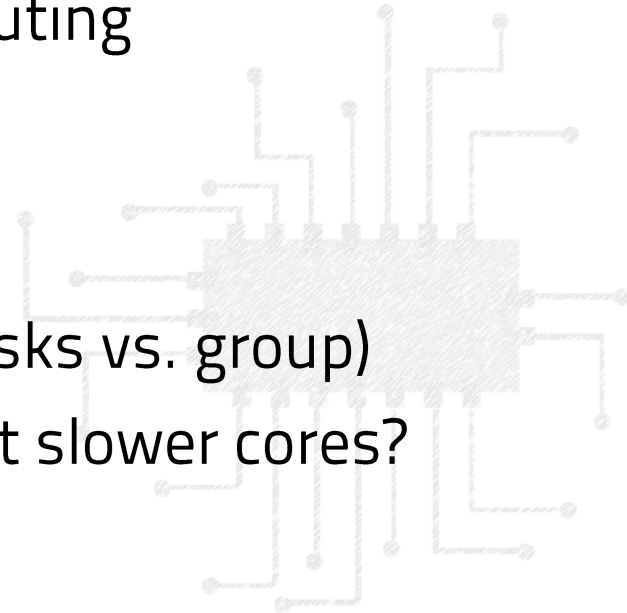
GPU Motivation

Why bother? We have great multi-core CPUs...

- CPUs are optimized for generic-processing of individual tasks
 - Faster cores
 - Smaller number of hardware threads
- GPUs are optimized for throughput computing
 - Slower cores
 - Larger number of hardware threads

Trade-off: Cores vs. speed (i.e. individual tasks vs. group)

Tricky: How can we use parallelism to offset slower cores?



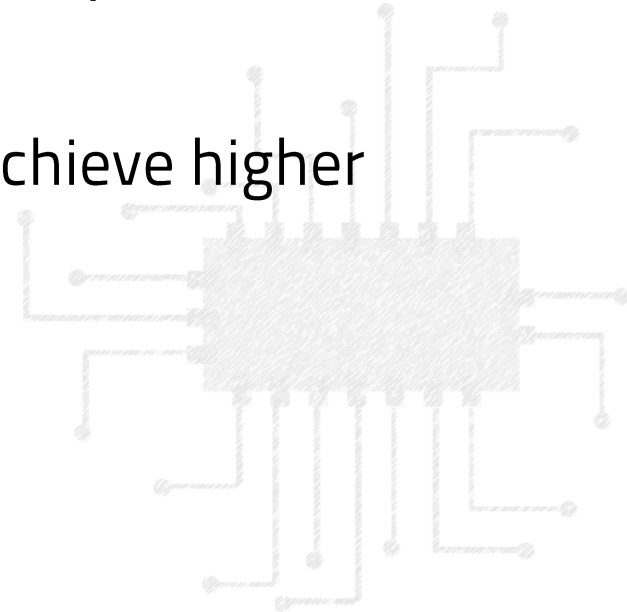


GPU Design Goals

Modern GPUs are designed for:

- High-degrees of parallelism;
- Efficient synchronization and data sharing;
- Generic parallel programming; and
- High-level parallel languages (OpenCL/CUDA).

Idea: By saturating the GPU cores, we can achieve higher throughput than on a CPU.

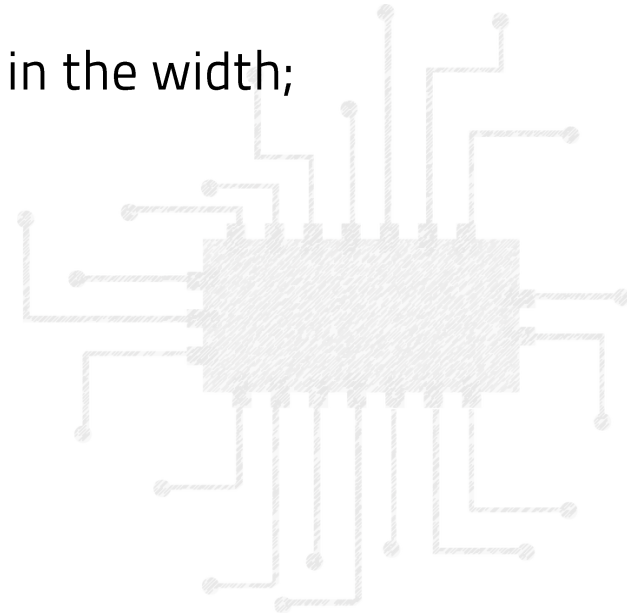




Programming Model

The GPU parallel programming model is based on:

- **SIMT**
 - Single Instruction, Multiple **Thread**
 - Program specifies the behaviour of a single thread;
- **By contrast in SIMD:**
 - Single Instruction, Multiple **Data**
 - Program specifies the behaviour of all threads in the width;
- Hierarchy of threads and memory; and
- Some limited synchronization.



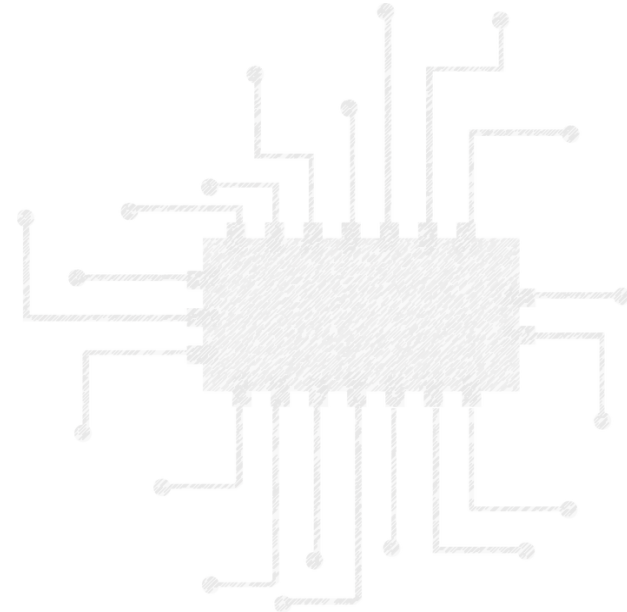


Thread Hierarchy

A **thread** is the basic unit of execution on a GPU.

- Lightweight;
- Low creation time; and
- Computes a single unit of data.

Thread

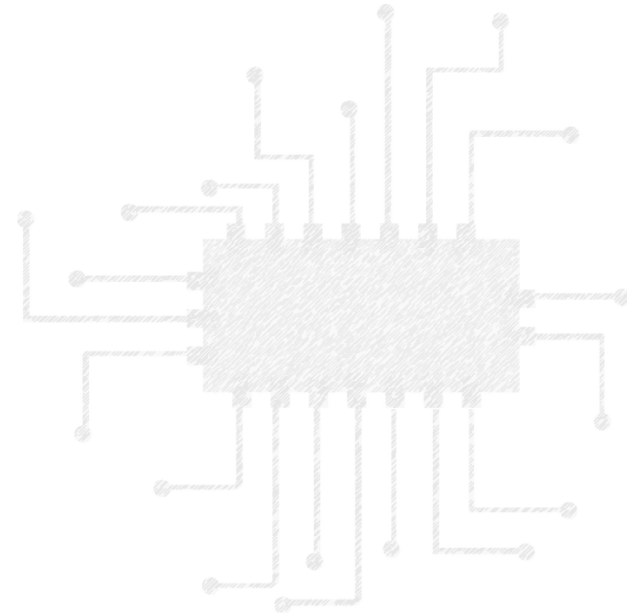
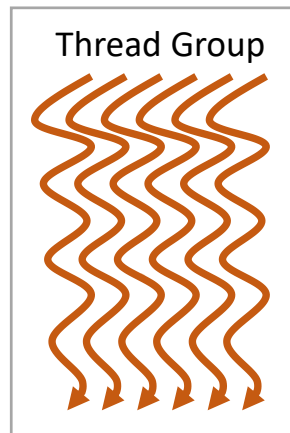




Thread Hierarchy

A **thread group** specifies a collection of threads.

- *May* work together;
- Can be synchronized and share data; and
- Also called a CTA (cooperative thread array) or a workgroup.

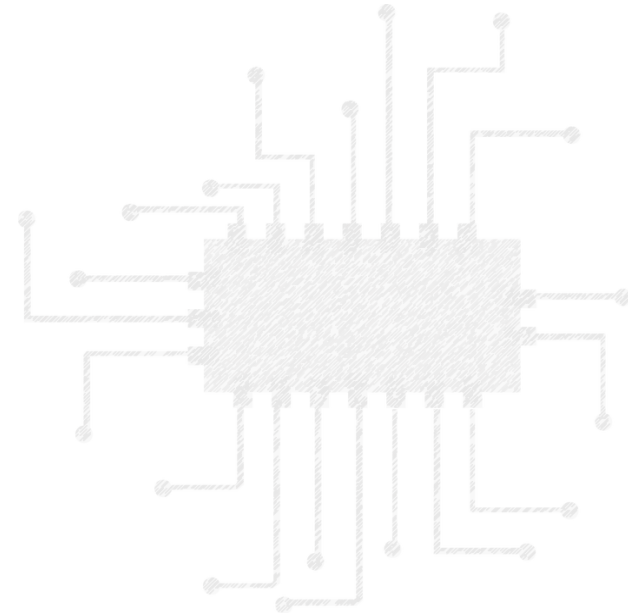
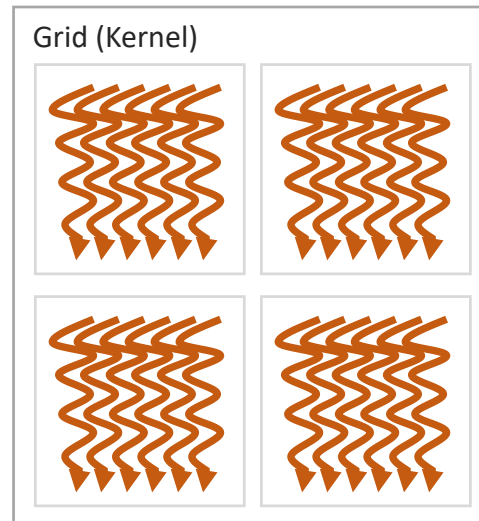




Thread Hierarchy

A **thread grid** is a collection of thread groups.

- Forms a *kernel* - the “program” for the GPU; and
- *Cannot* be synchronized.

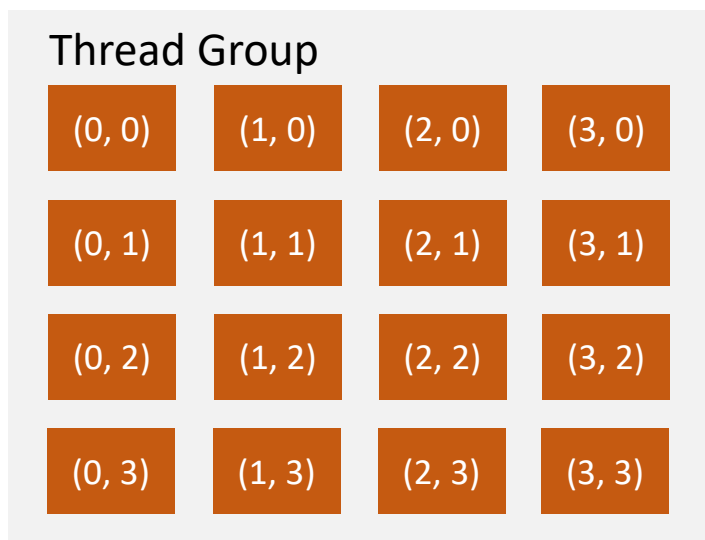




Thread Hierarchy

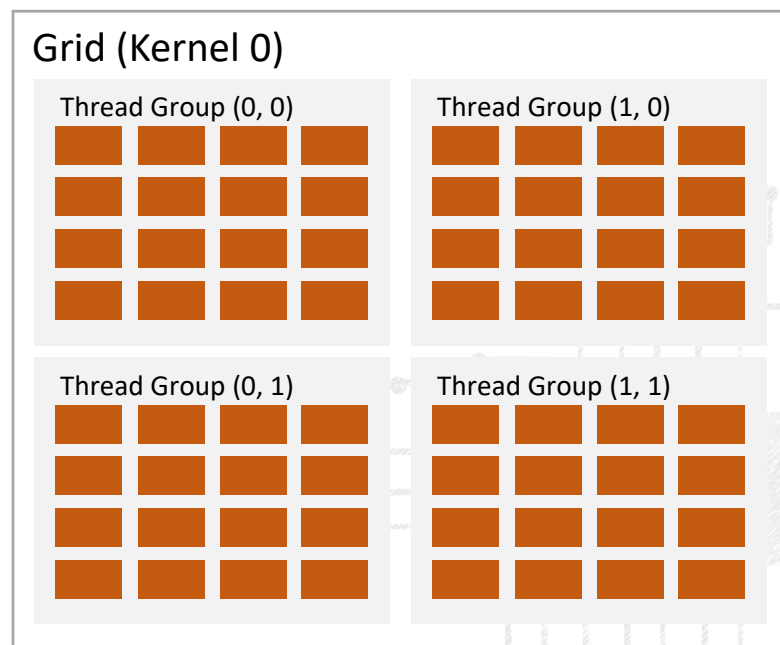
Thread Identifier

Each thread in a group has a unique identifier



Group Identifier

Each thread group in a grid has a unique identifier



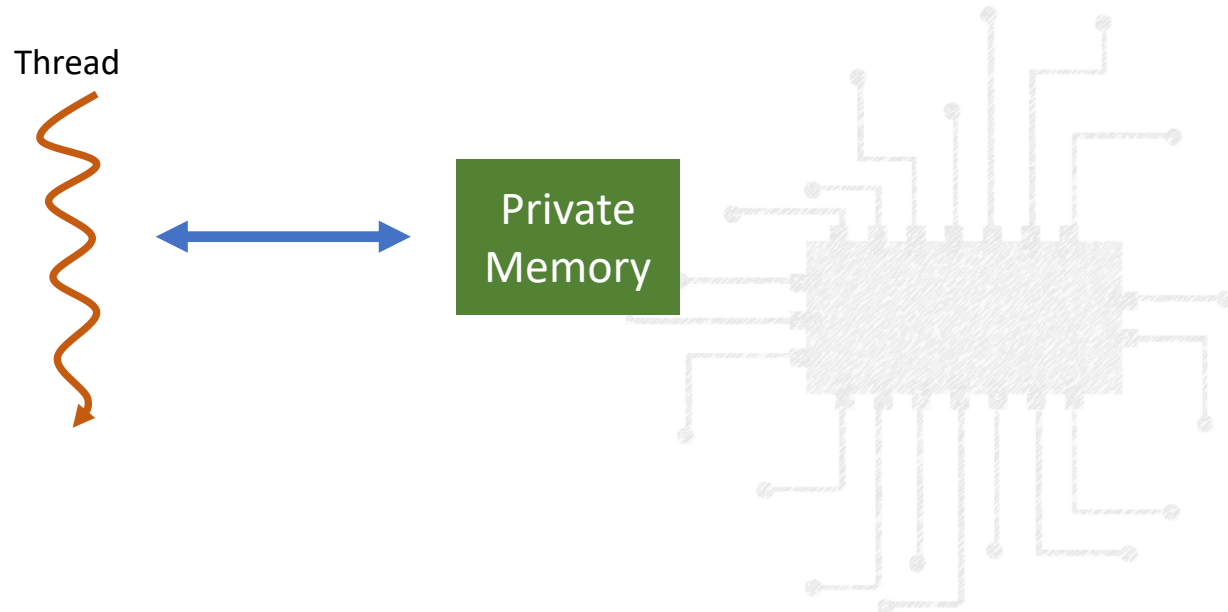
Thread + group identifiers uniquely specify a thread within a kernel



Memory Hierarchy

Each thread has its own **private memory**.

- Analogous to registers; and
- Cannot be accessed by other threads.

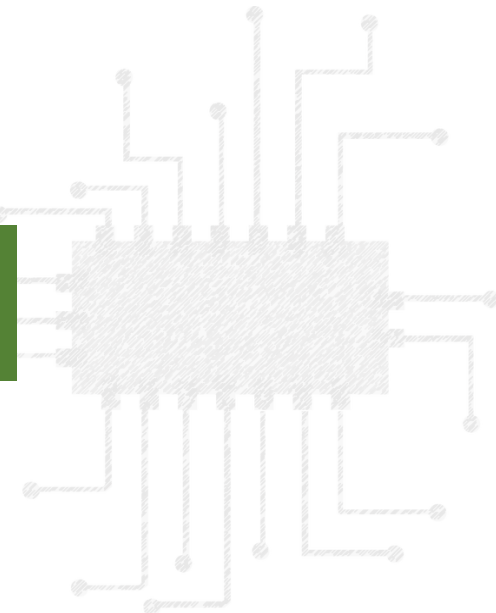
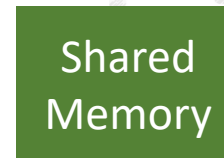
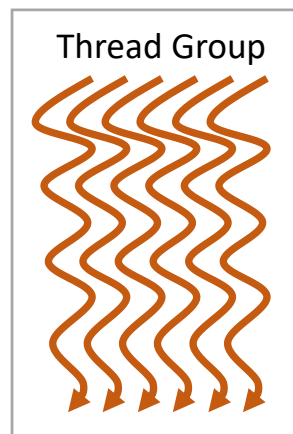




Memory Hierarchy

Each thread group has its own **shared memory**.

- Can be accessed by all threads within the group;
- Requires synchronization; but
- Cannot be accessed by other groups.

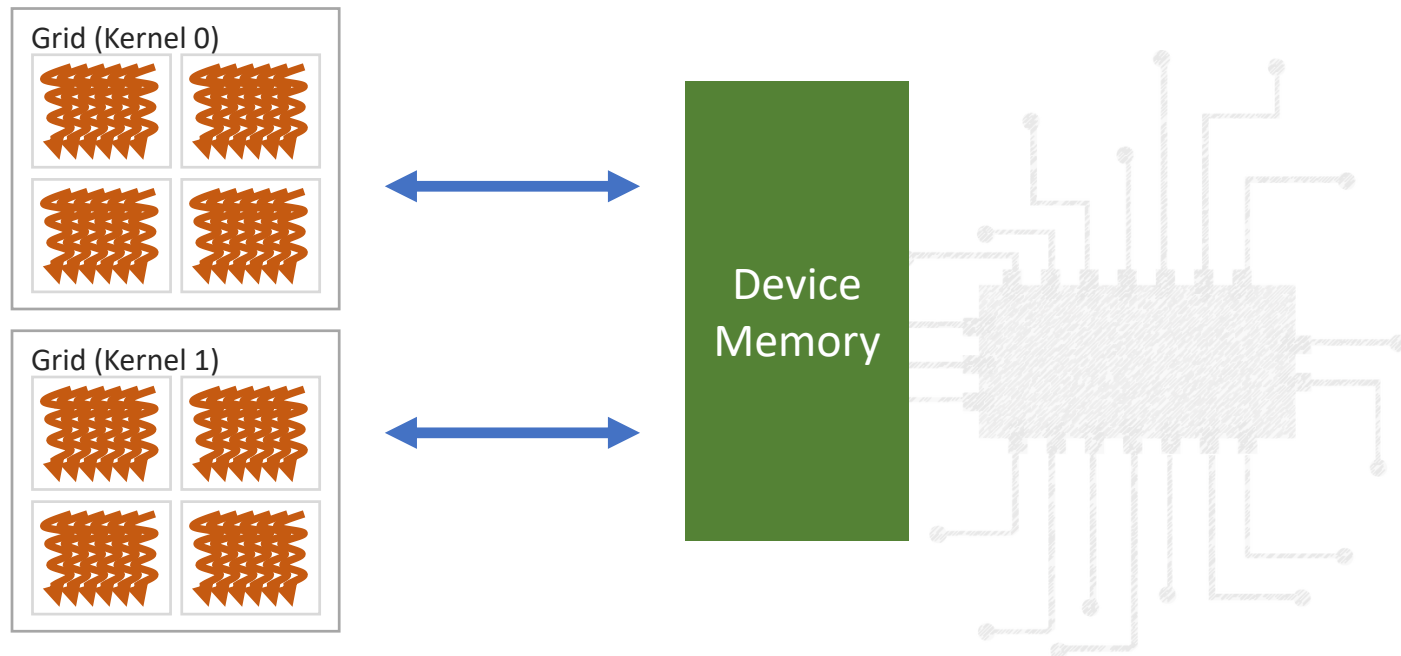




Programming Model

All grids share **device memory** (GPU memory).

- Can be accessed by *all threads in all grids*; but
- *Cannot* be synchronized between groups or grids.





Example: Image Set

Given a 2000x1000 image in RGB, how can we efficiently increase the B value for every pixel using a GPU?

1. What high-level algorithm design should we use?

Each pixel gets its own thread

2. How many threads do we need?

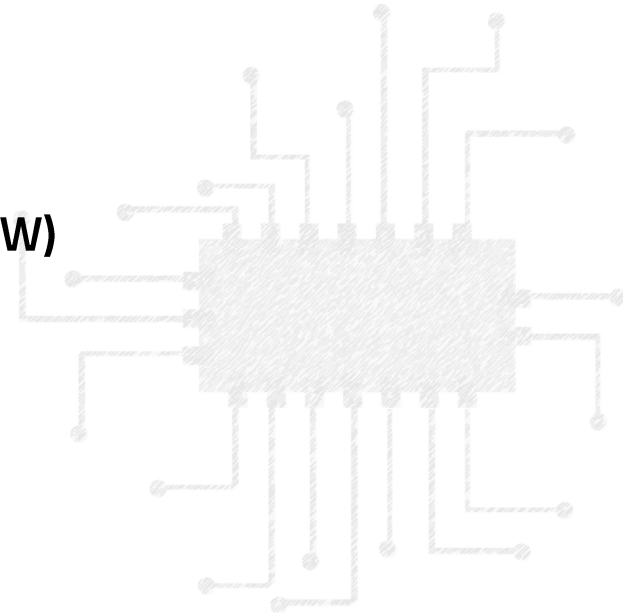
2000x1000

3. What is the thread-hierarchy?

All independent (as many groups as needed for HW)

4. Where is the image data stored?

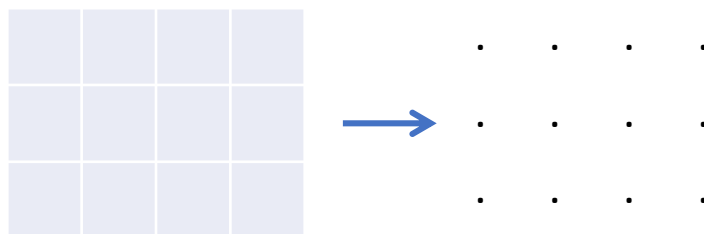
Device memory, no shared memory needed





Example: Image Downsize

Given a 2000x1000 image, compute the downsized image by taking the average RGB of each 10x10 region.



1. What high-level algorithm design should we use?

Each block gets a thread group, each thread averages a single row

2. How many threads do we need?

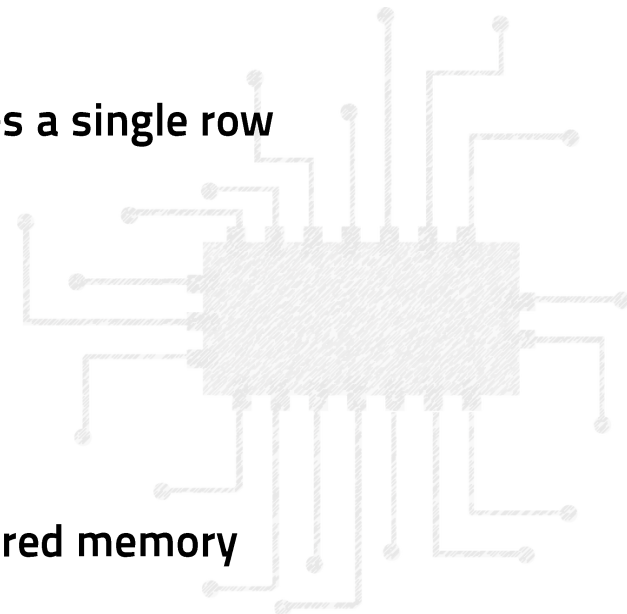
200x100x10 (1 per block per row)

3. What is the thread-hierarchy?

200x100 groups (10 threads each), 1 grid

4. Where is the image data stored?

Image in device memory, intermediate values in shared memory





Machine Model

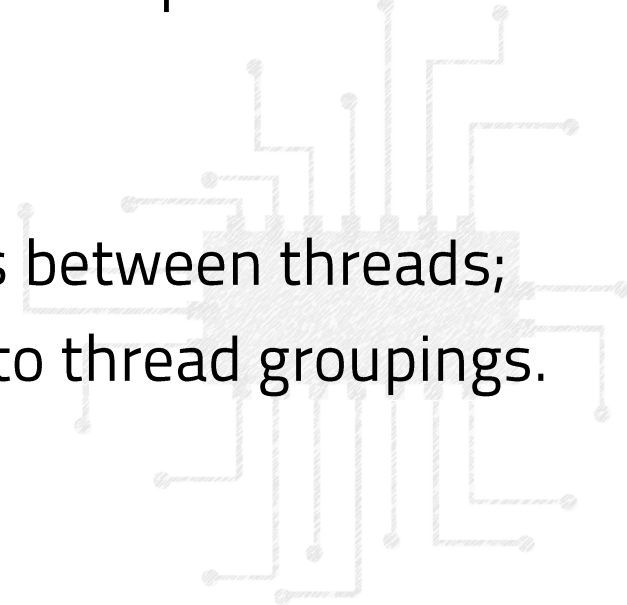
GPU hardware is geared towards high degrees of parallelism.

Computation

- Highly parallel, with thousands (and thousands of threads);
- Hierarchically parallel, threads grouped at multiple levels.

Memory

- High bandwidth, fast concurrent accesses between threads;
- Hierarchical design, levels corresponding to thread groupings.

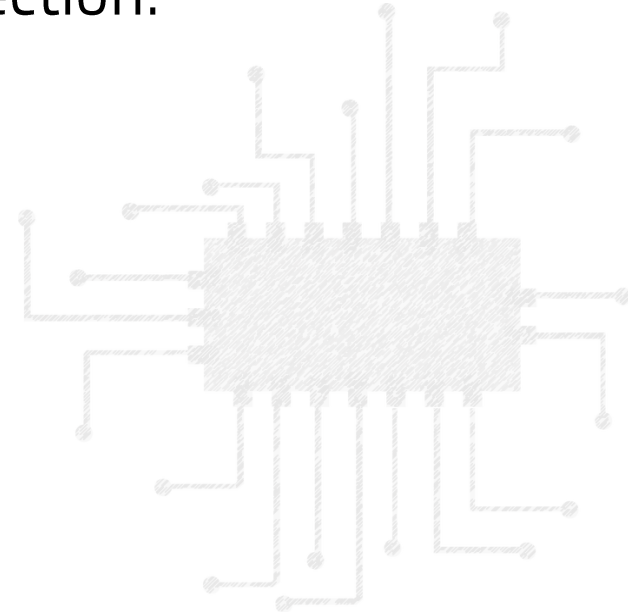




Program Execution

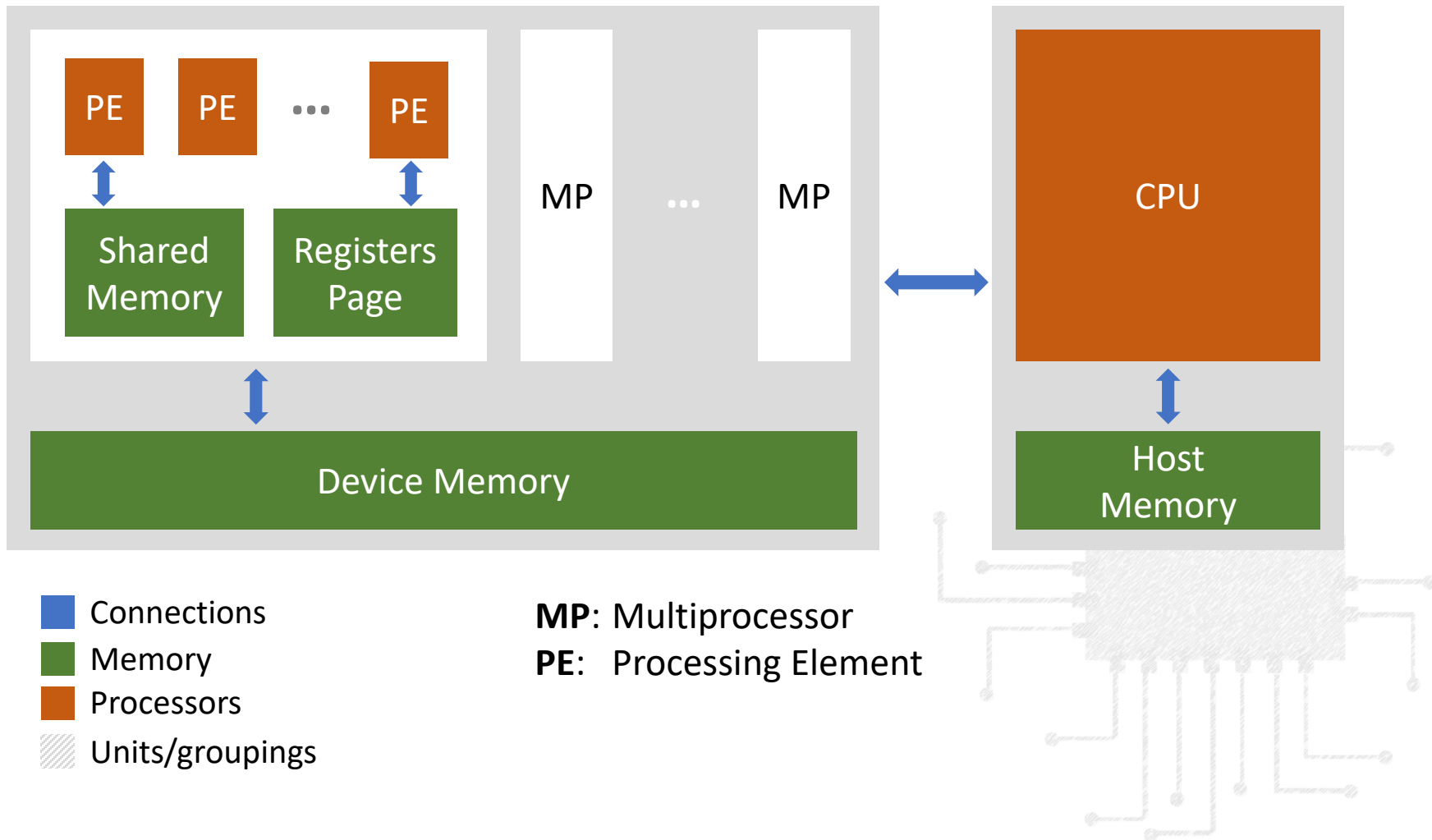
A full GPU program consists of two code sections:

- **Host code** that runs on the CPU;
 - Compiles the program;
 - Transfers the data; and
 - Specifies the thread geometry (number and organization of threads).
- **Kernel (GPU code)** executes the parallel section.



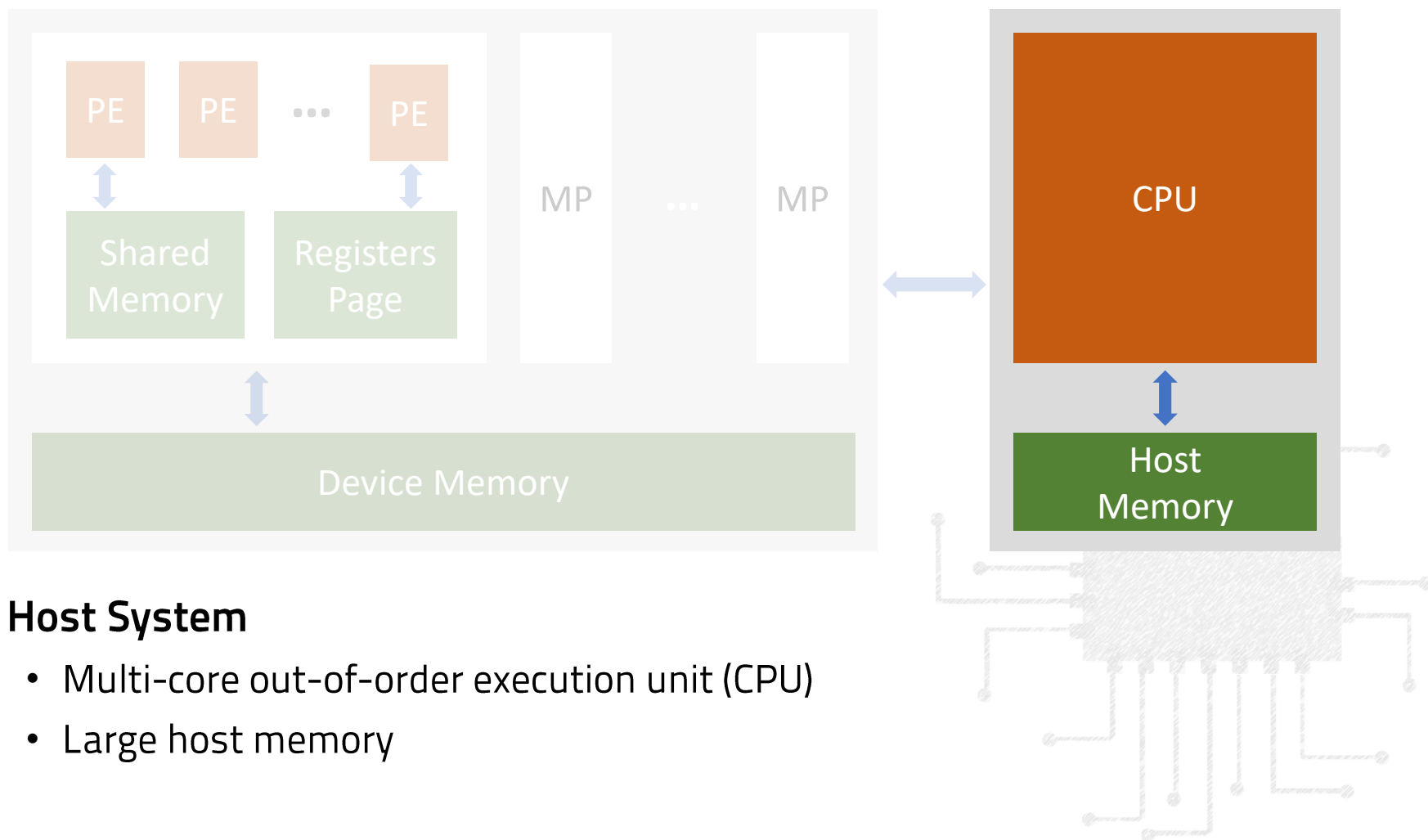


Machine Model



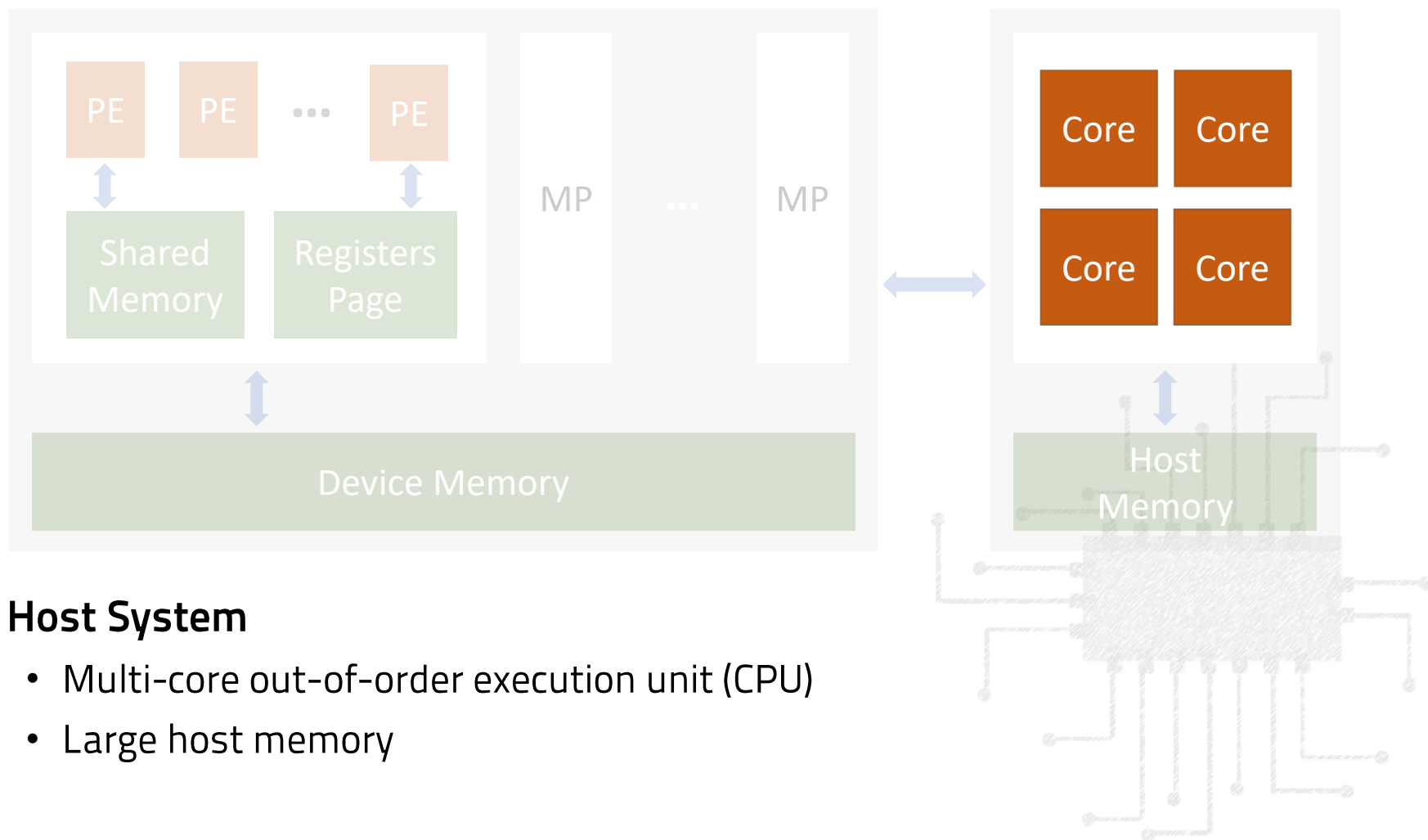


Machine Model





Machine Model

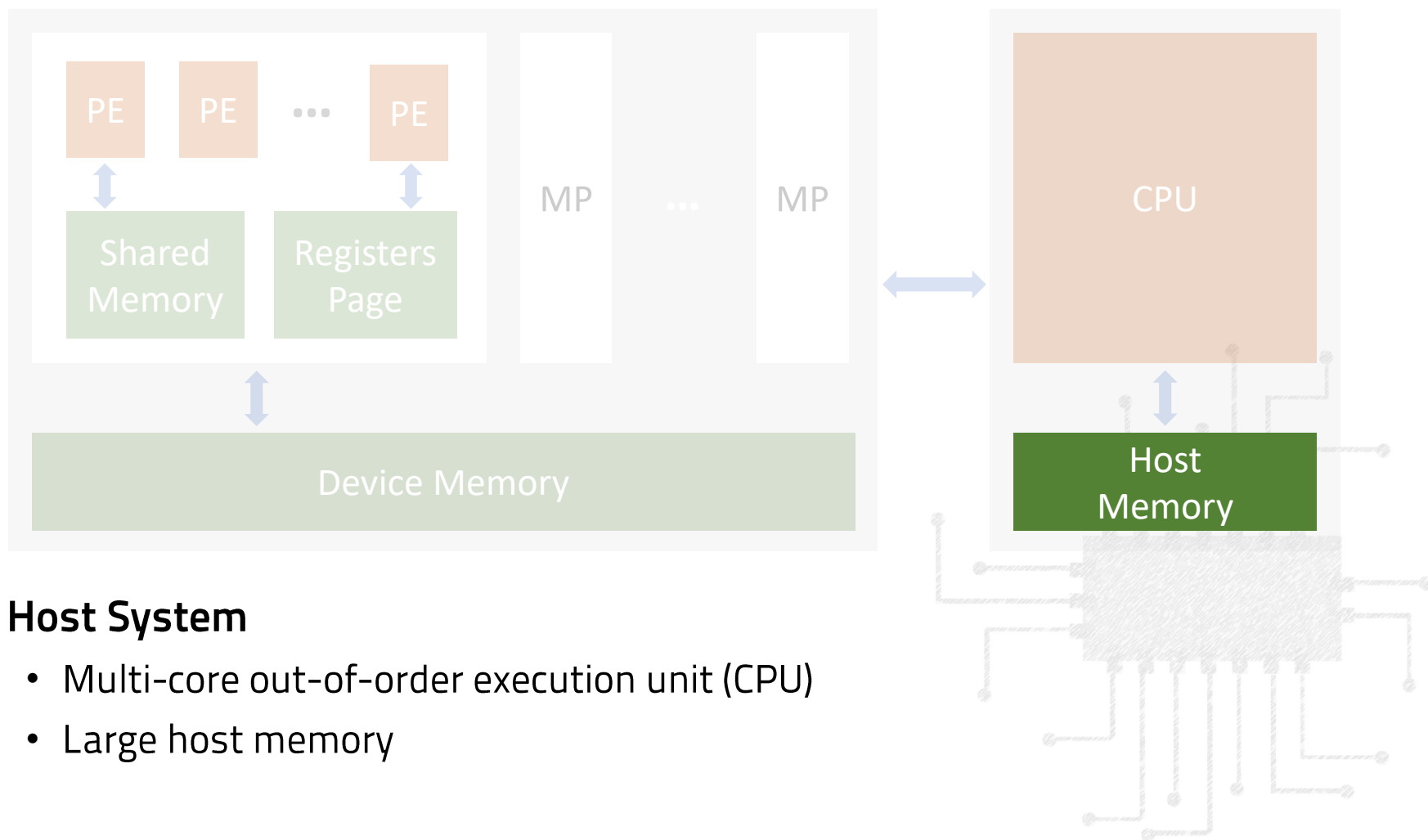


Host System

- Multi-core out-of-order execution unit (CPU)
- Large host memory

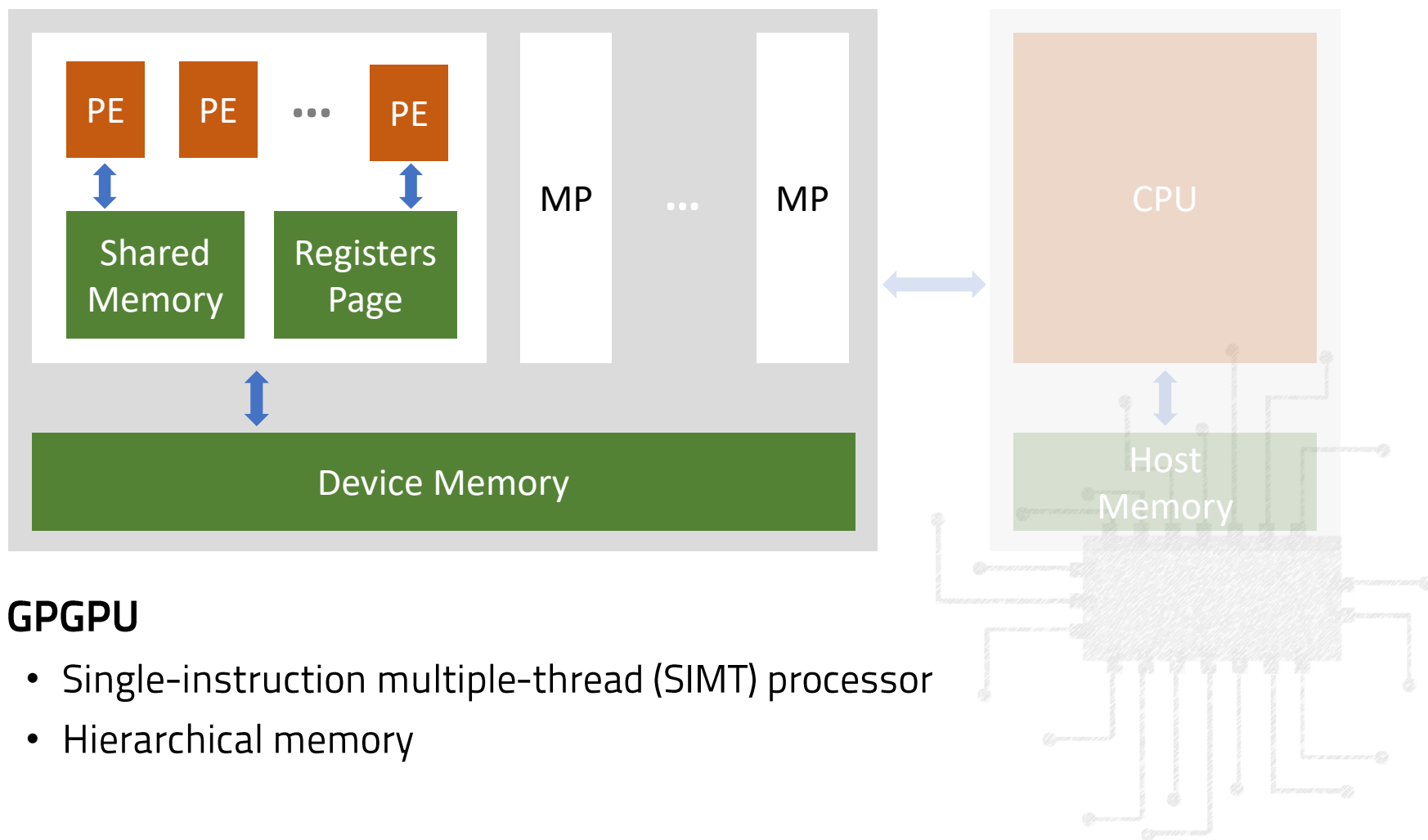


Machine Model





Machine Model

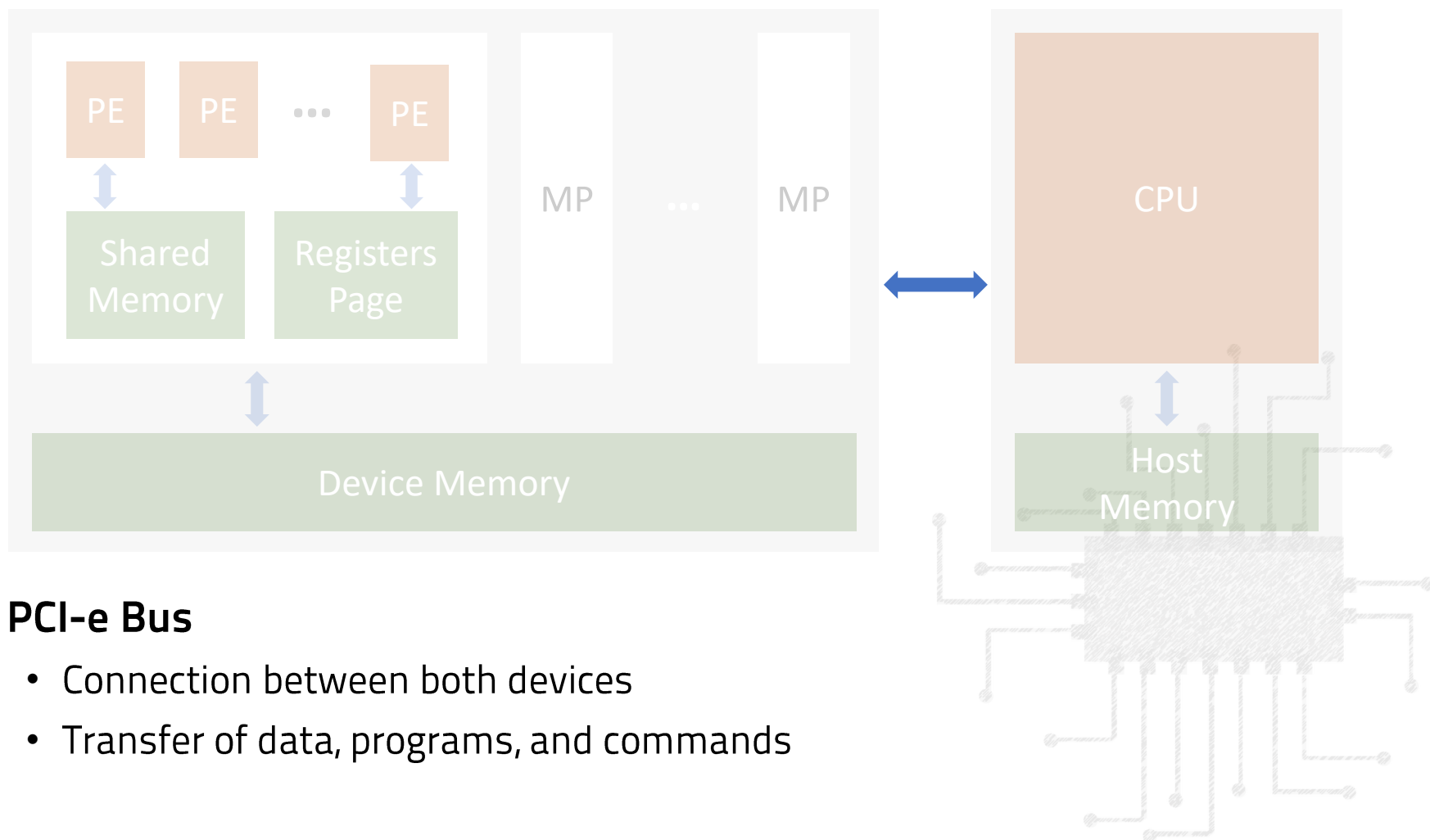


GPGPU

- Single-instruction multiple-thread (SIMT) processor
- Hierarchical memory



Machine Model

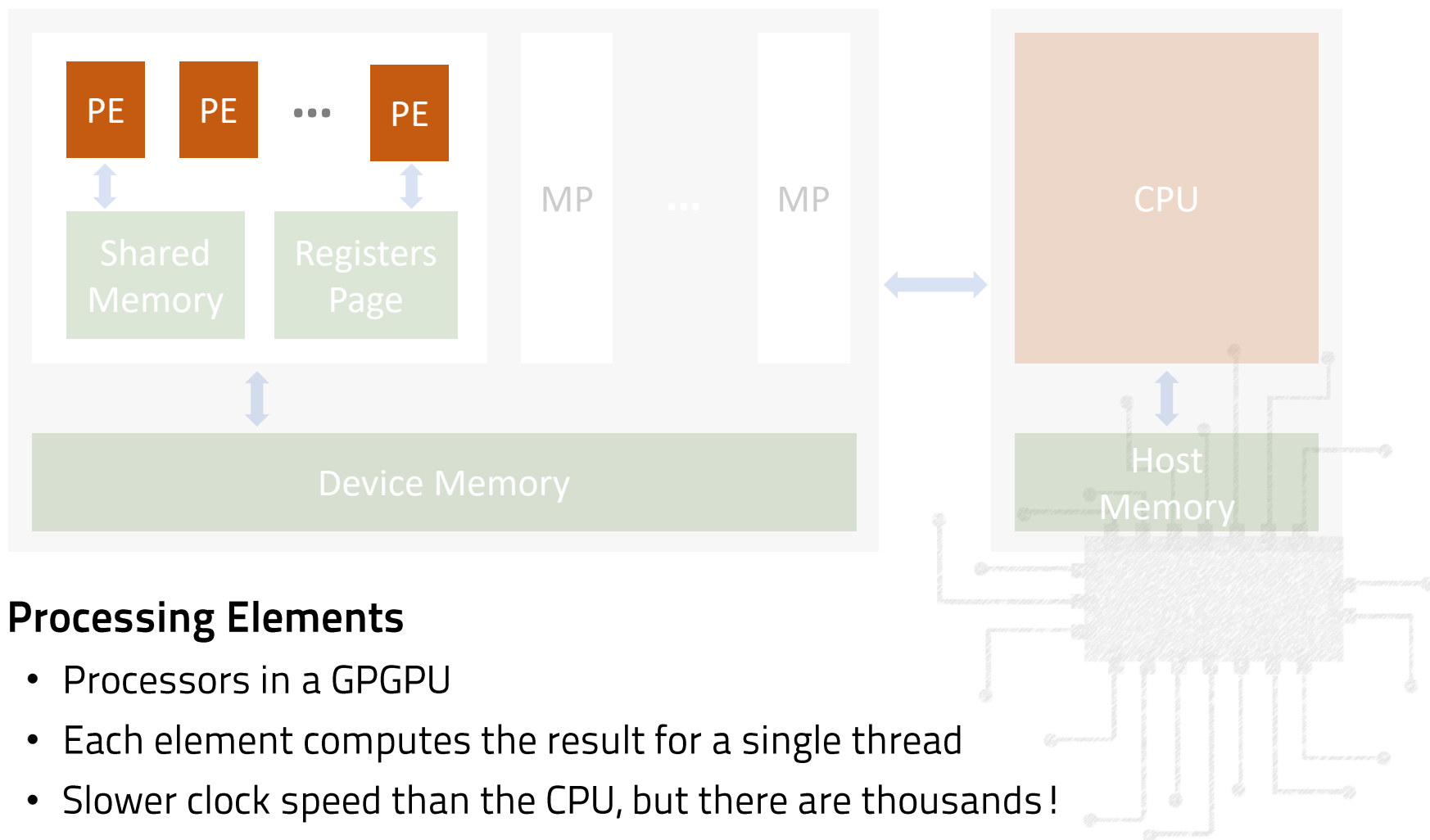


PCI-e Bus

- Connection between both devices
- Transfer of data, programs, and commands



Machine Model

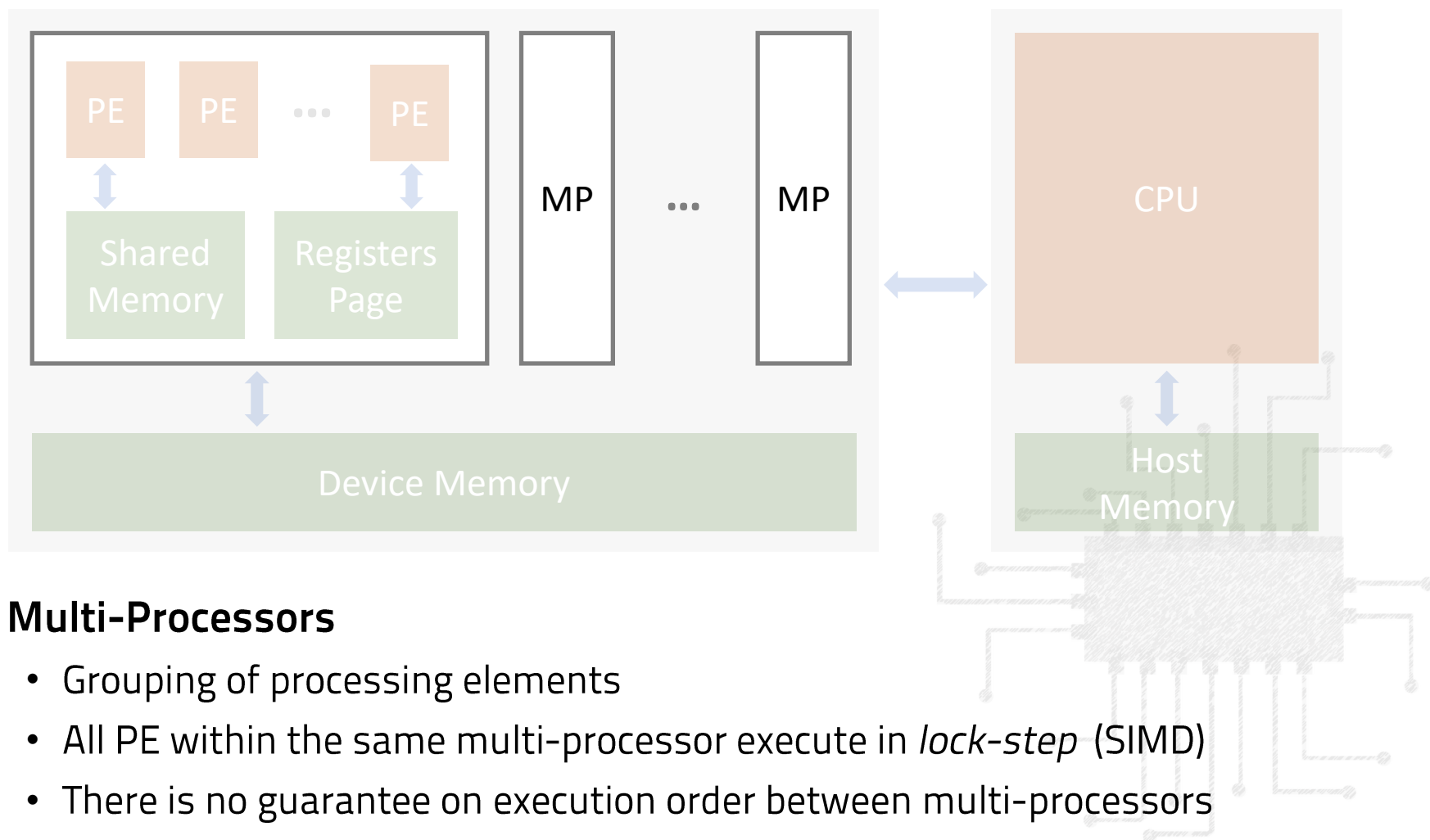


Processing Elements

- Processors in a GPGPU
- Each element computes the result for a single thread
- Slower clock speed than the CPU, but there are thousands!



Machine Model

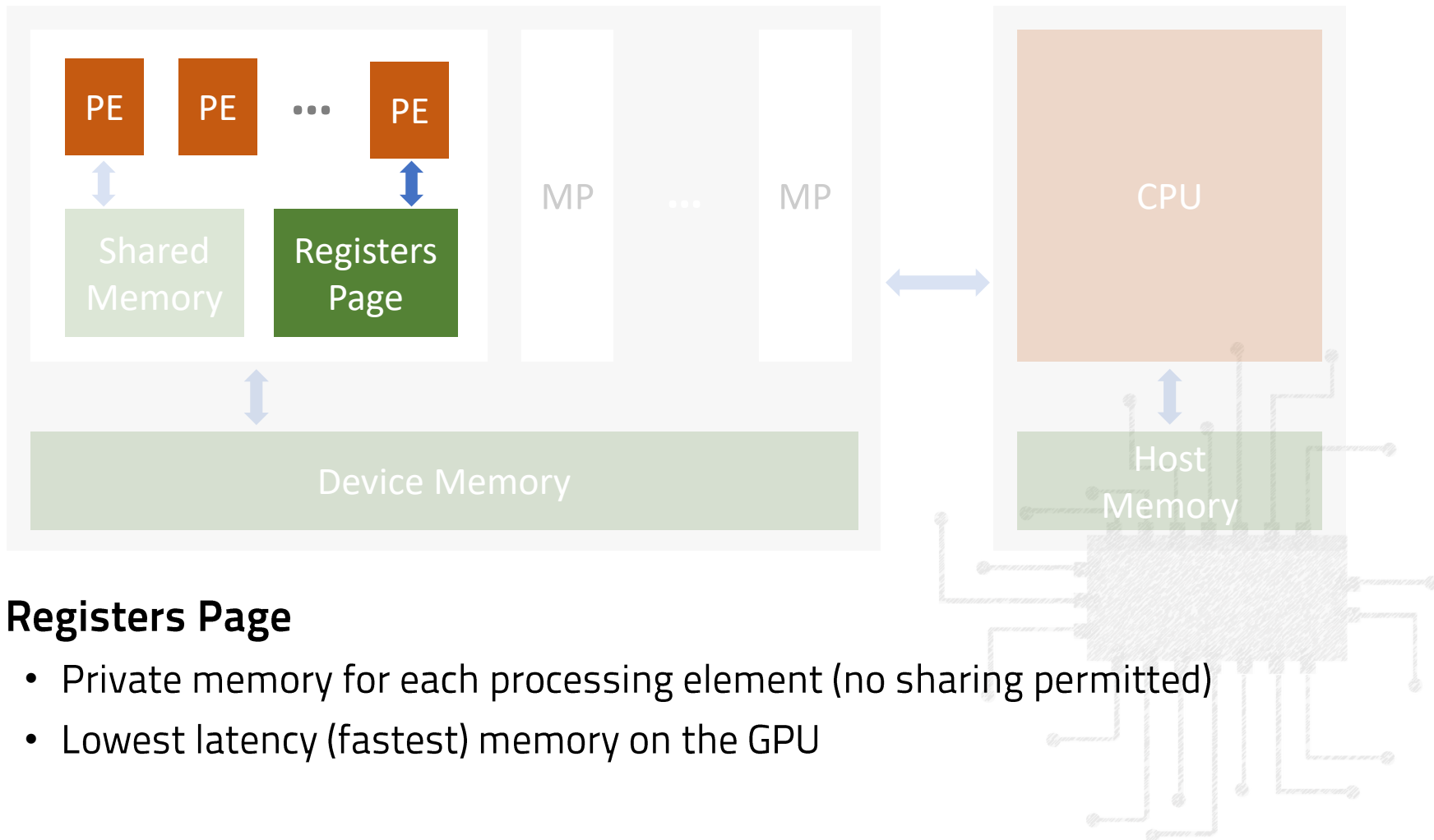


Multi-Processors

- Grouping of processing elements
- All PE within the same multi-processor execute in *lock-step* (SIMD)
- There is no guarantee on execution order between multi-processors



Machine Model

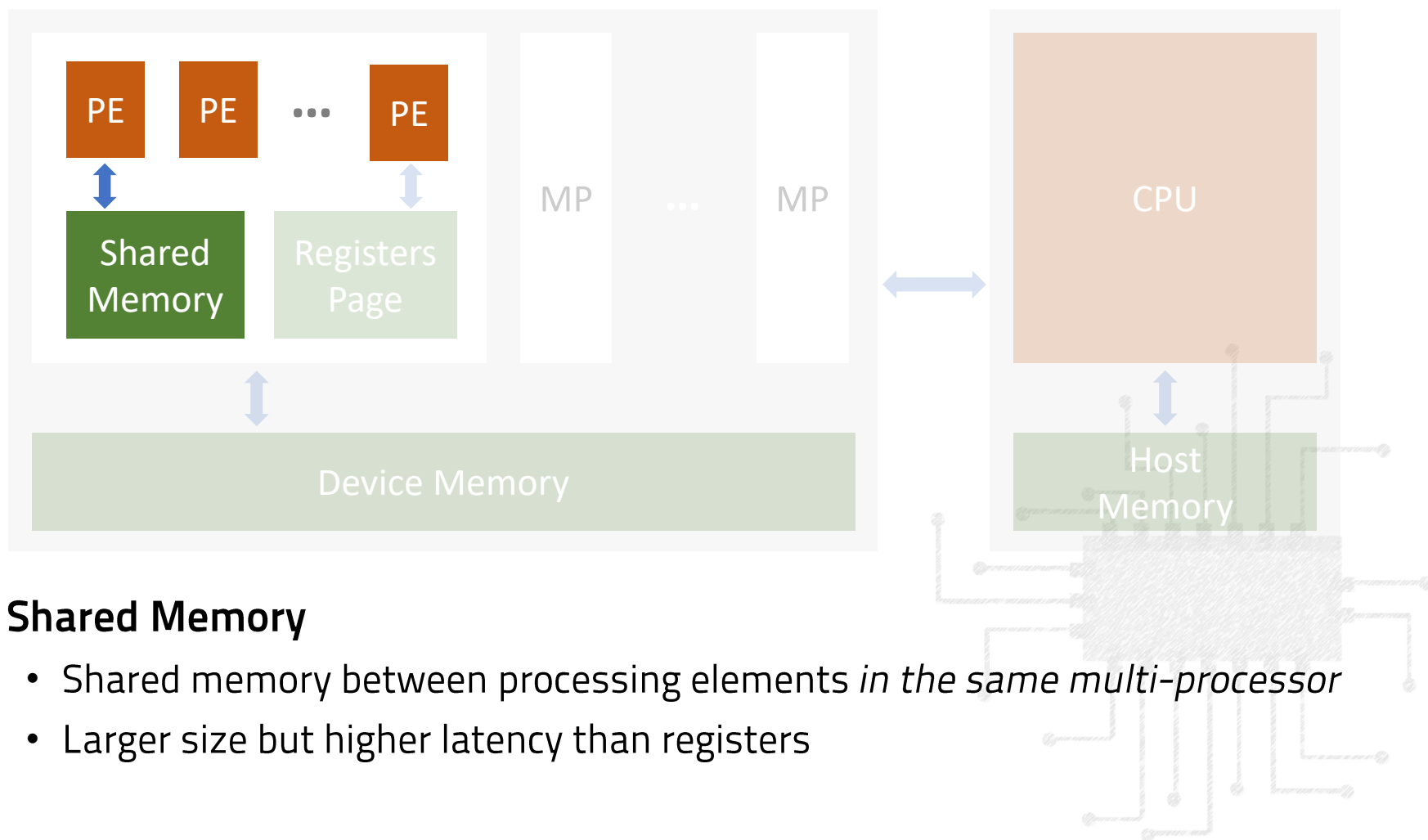


Registers Page

- Private memory for each processing element (no sharing permitted)
- Lowest latency (fastest) memory on the GPU

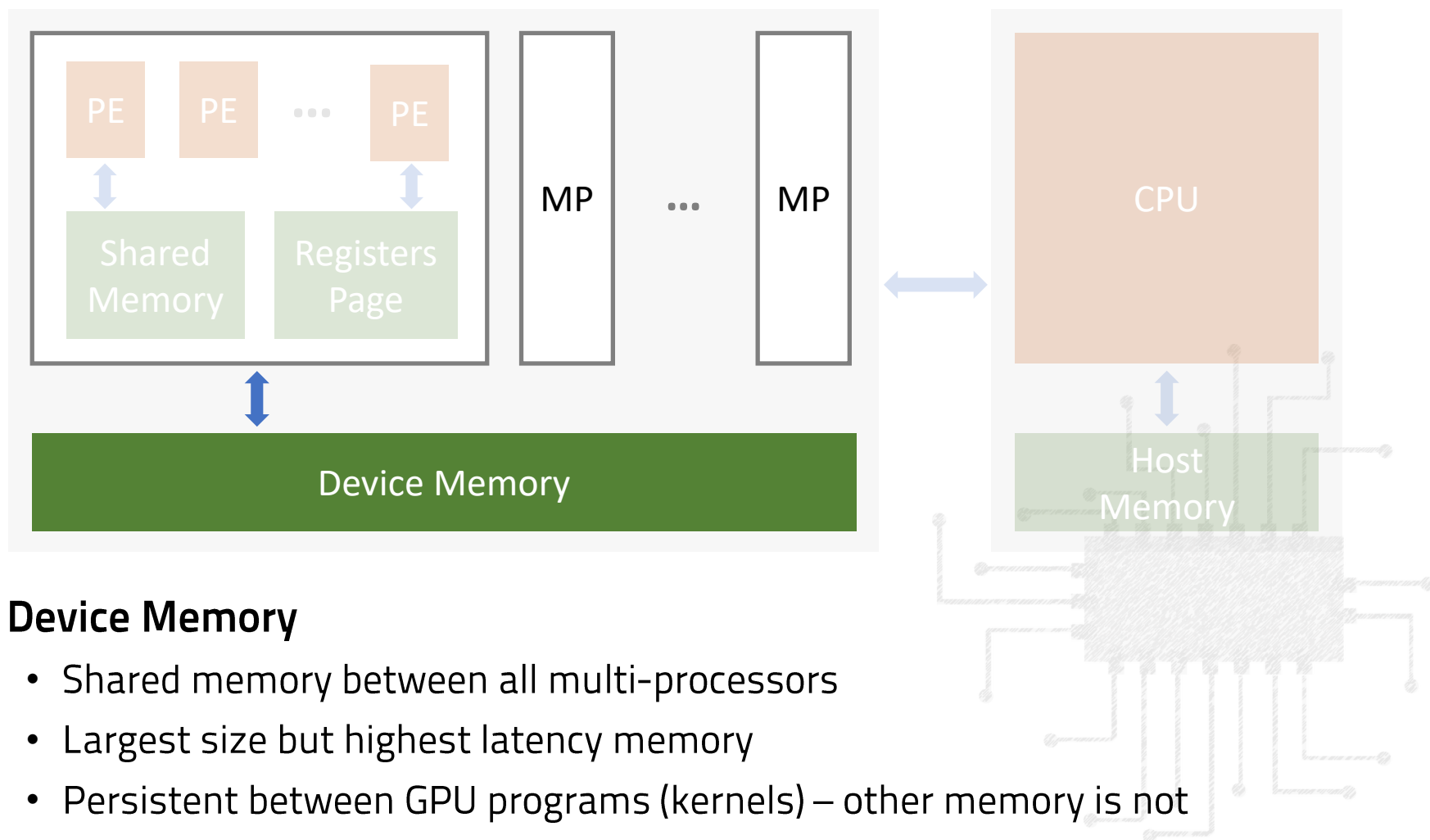


Machine Model





Machine Model



Device Memory

- Shared memory between all multi-processors
- Largest size but highest latency memory
- Persistent between GPU programs (kernels) – other memory is not



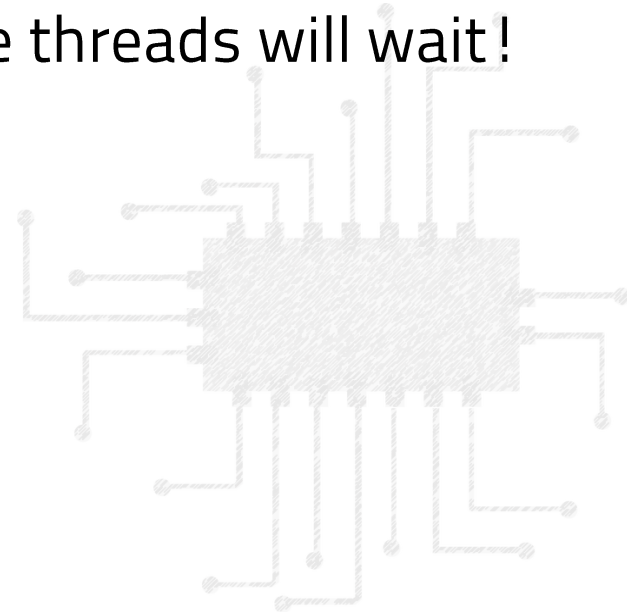
Machine Model (Execution)

Two important questions:

1. How do we efficiently schedule 1000s of threads on a limited number of cores?
2. How do threads in SIMD diverge?

Answers:

1. Round-robin executable threads – some threads will wait!
2. Masking.





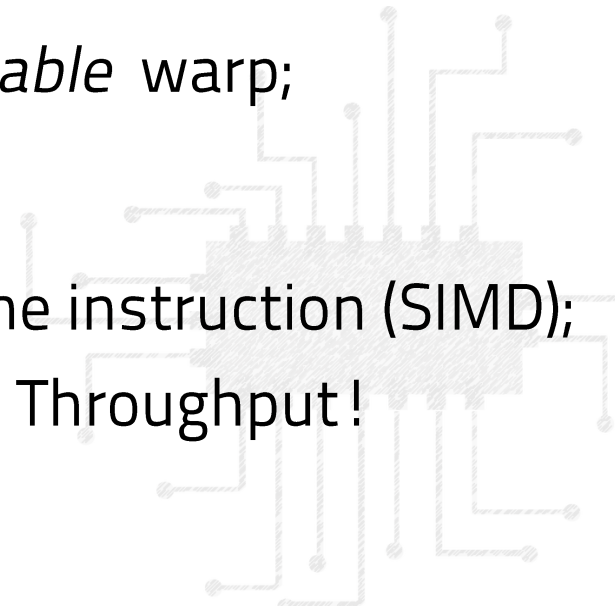
Machine Model (Execution)

A **warp** is the scheduling unit of threads.

- Contains a fixed number (usually 32) consecutive threads;
- Instruction scheduling occurs at the warp level.

For each cycle:

- The instruction unit picks the next *executable* warp;
 - Not waiting for a memory load/store
 - Not blocked (synchronization)
- Each thread in the warp executes the same instruction (SIMD);
- This overlaps computation and wait time. Throughput!





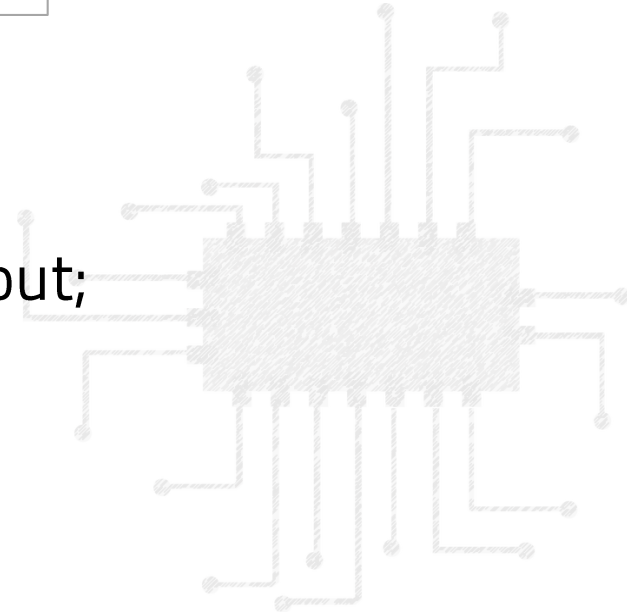
Machine Model (Execution)

Thread divergence occurs when two threads within a warp execute different branches.

```
if (tid.x % 2 == 0) {  
    a += 5;  
} else {  
    a += 4;  
}
```

To handle divergent execution:

- All threads execute the entire structure;
- "Inactive" threads have results masked-out;
- Serializes the execution of an if-else.

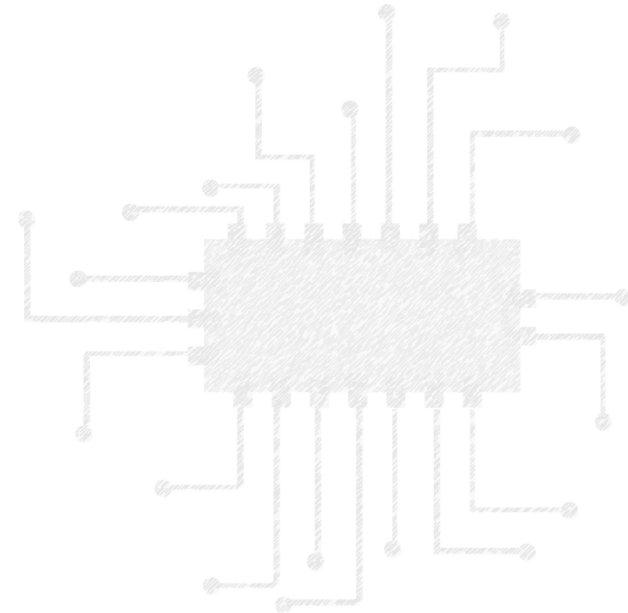
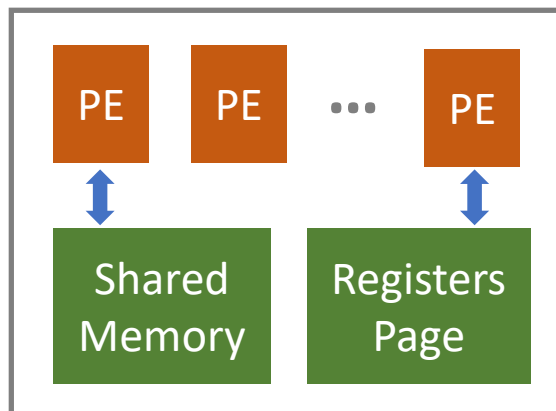




Thread Mapping

Mapping from the programming to the machine model:

- Each thread group is assigned to a multi-processor;
 - Shared memory
- Each thread executes on a processing element.
 - Registers page





Memory Mapping

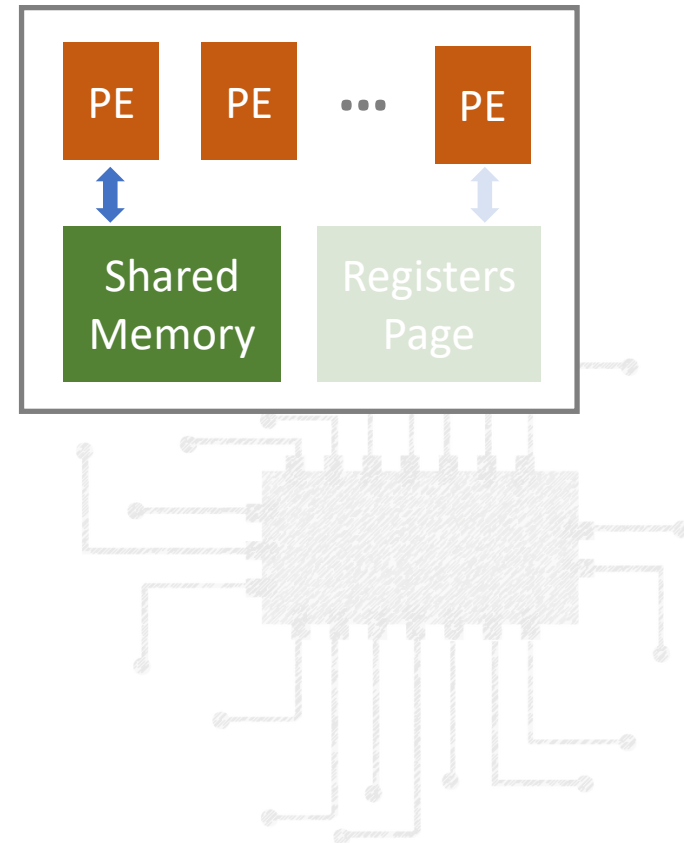
Shared Memory

- Shared by all threads in a group; but
- Not automatically synchronized.

Synchronization

Synchronization ensures that all threads in a group are at the same point in the kernel.

- **Within a group:** Memory barrier
- **Between groups:** Impossible !





Memory Mapping

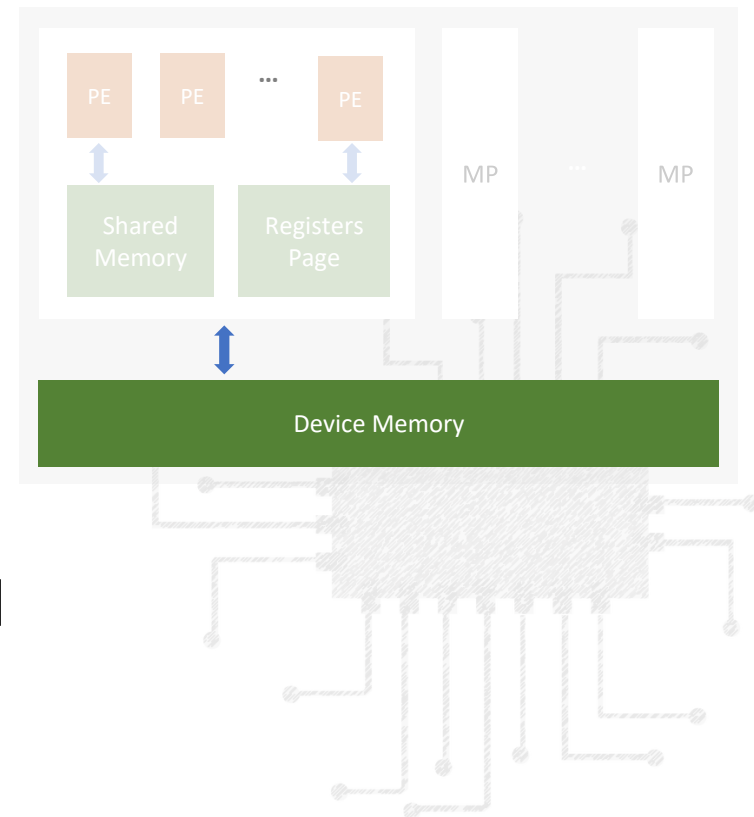
Device Memory

- Shared by all threads on the GPU; but
- Not automatically synchronized!

Synchronization

Synchronization ensures that all threads in a group are at the same point in the kernel.

- **Within a group:** Memory barrier
- **Between groups:** Impossible! (Kernel boundary)





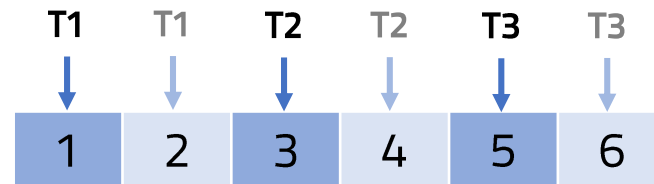
Memory Optimization

Device memory accesses are the **most expensive** instructions – and there are thousands occurring concurrently.

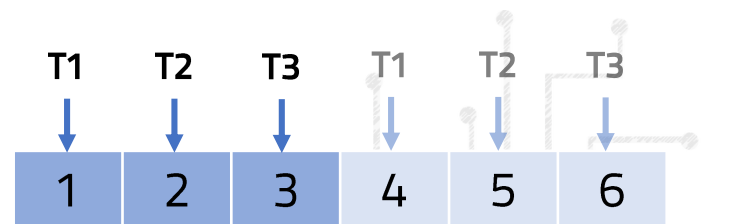
GPUs achieve high-bandwidth by using memory transactions.

Coalescing

- Avoid filling transactions with unused data; and
- Merge concurrent accesses to consecutive memory locations into a single transaction.
- **Pattern:** Access consecutive memory locations from consecutive threads.



Uncoalesced Access Pattern

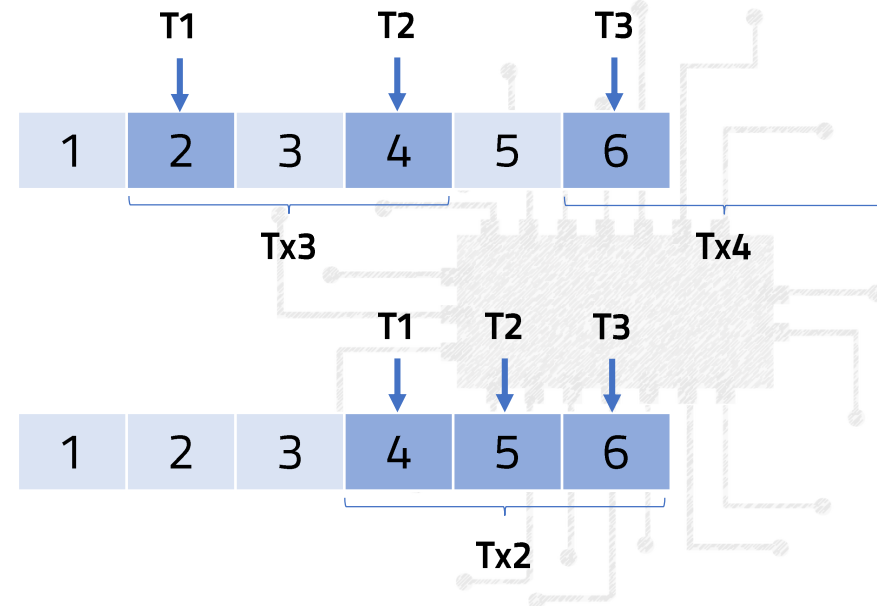
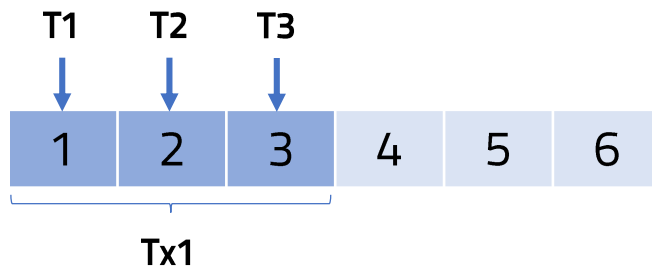
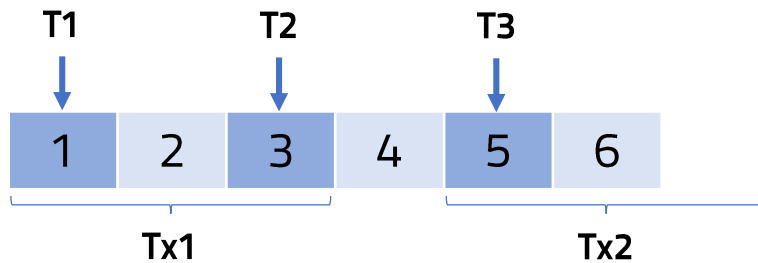


Coalesced Access Pattern



Example: Memory Optimization

Given a kernel with 3 threads each loading 2 values and a transaction size of 3. **How many loads are required if we coalesce/do not coalesce?**





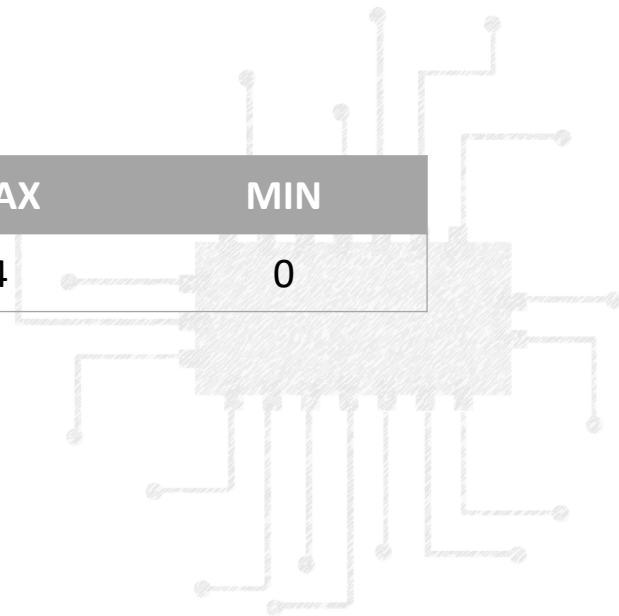
Parallel Reductions

Idea: Group the values of multiple rows into a single value (fold)

0	2	4	2	3	2	3	0
---	---	---	---	---	---	---	---

Common Reductions:

COUNT	SUM	AVG	MAX	MIN
8	16	2	4	0

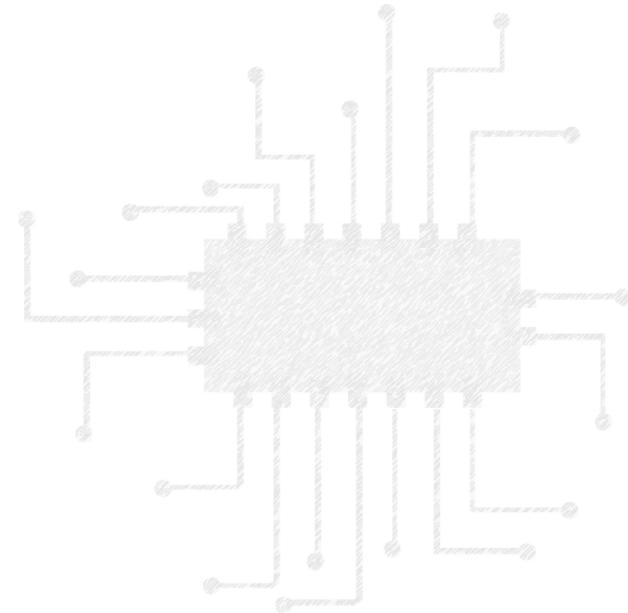




Parallel Reductions (SUM)

device

0	2	4	2	3	2	3	0
---	---	---	---	---	---	---	---



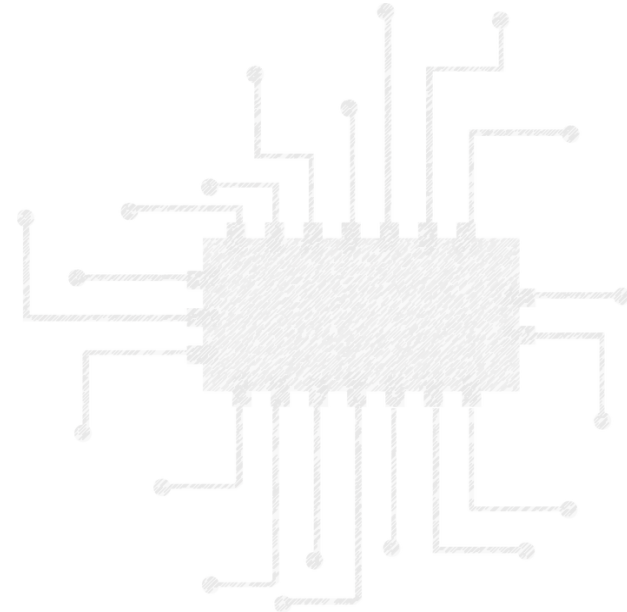


Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device

0	2	4	2	3	2	3	0
---	---	---	---	---	---	---	---

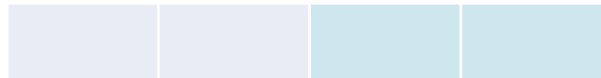
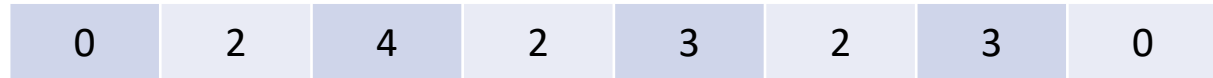




Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

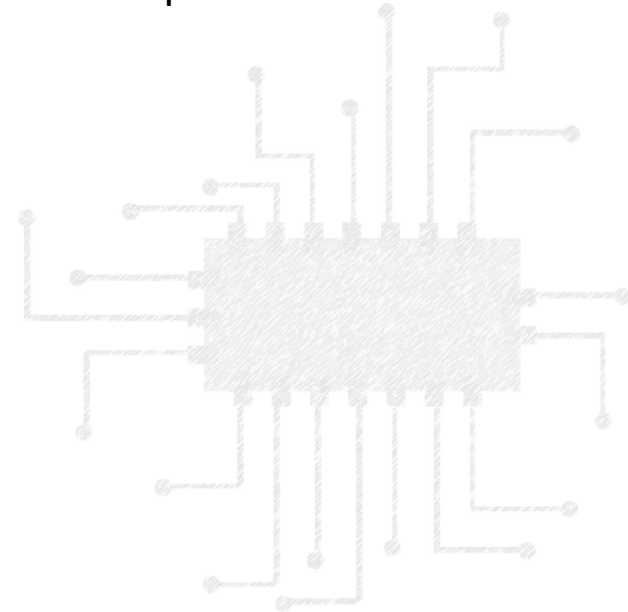
device



group 1

group 2

1 location per thread

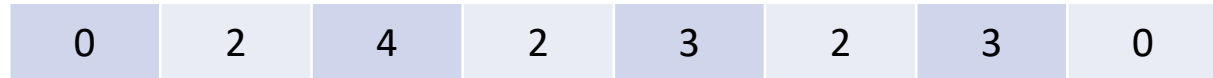




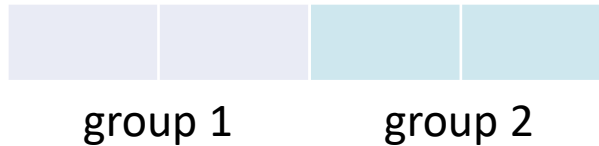
Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

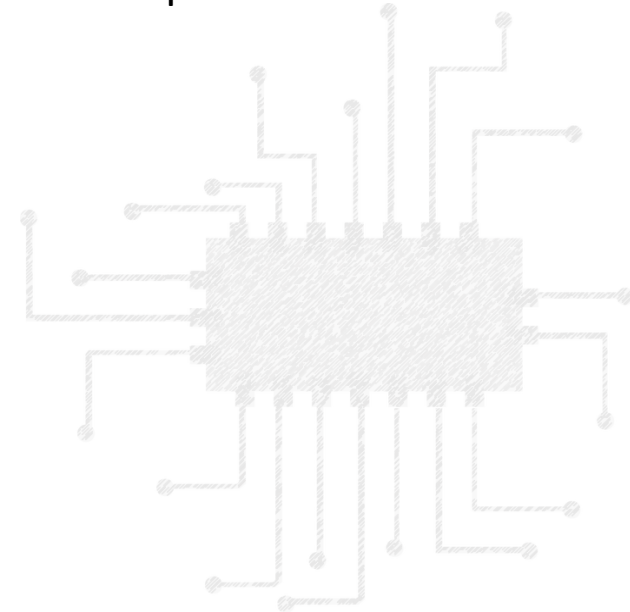
device



shared



1 location per thread

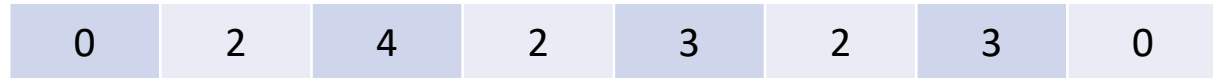




Parallel Reductions (SUM)

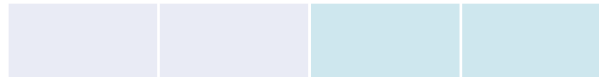
2 thread groups, 2 threads/group = 4 threads

device



group 1, thread 1

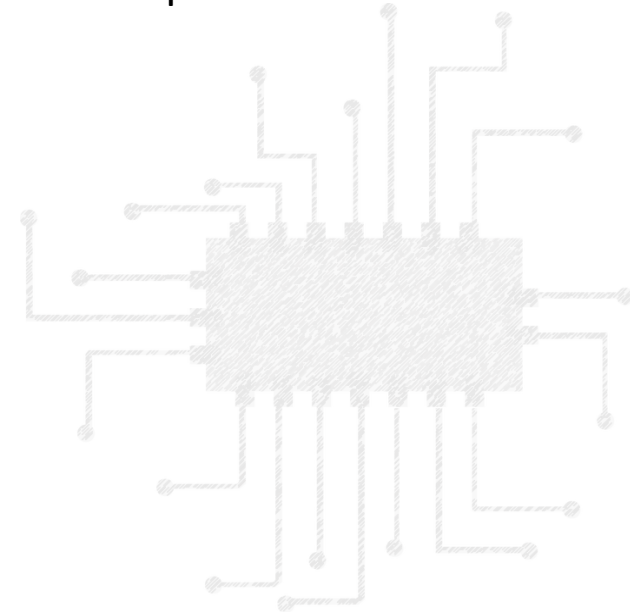
shared



group 1

group 2

1 location per thread





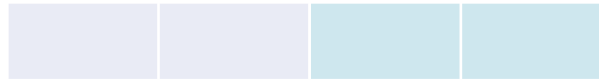
Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device

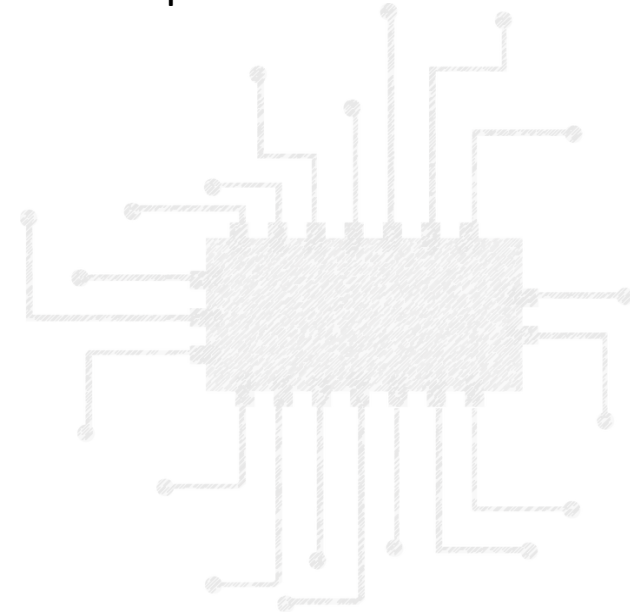


shared



group 1, thread 1

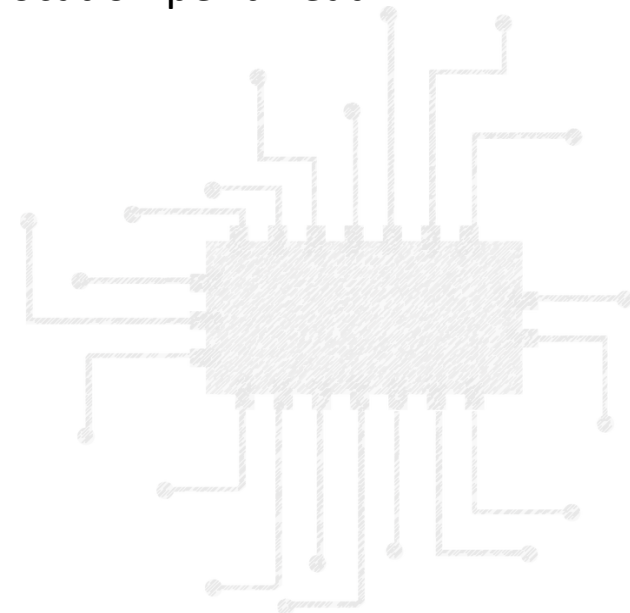
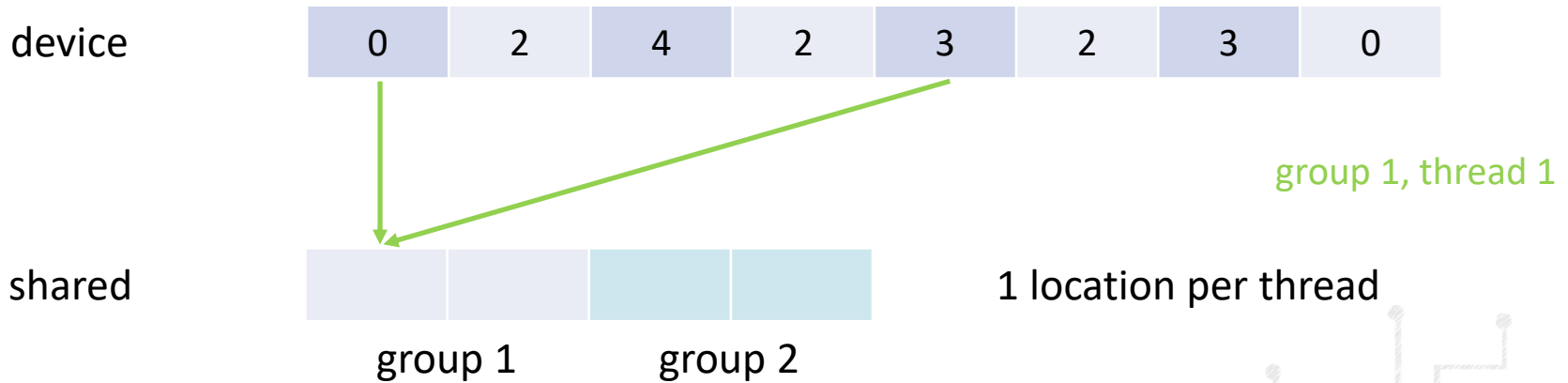
1 location per thread





Parallel Reductions (SUM)

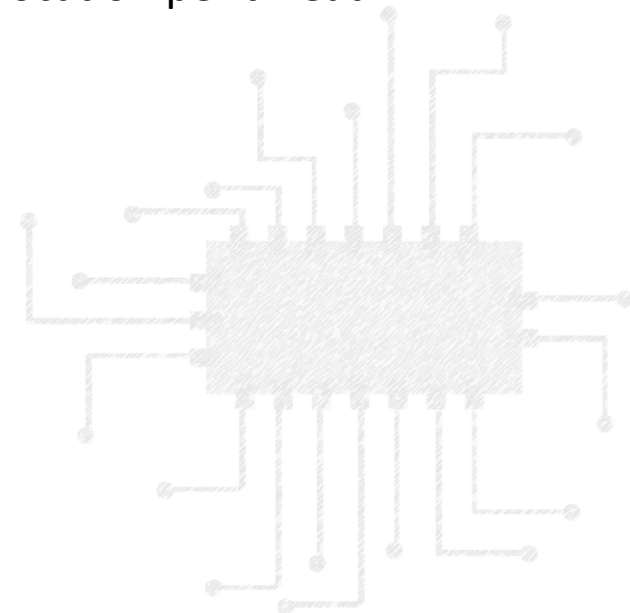
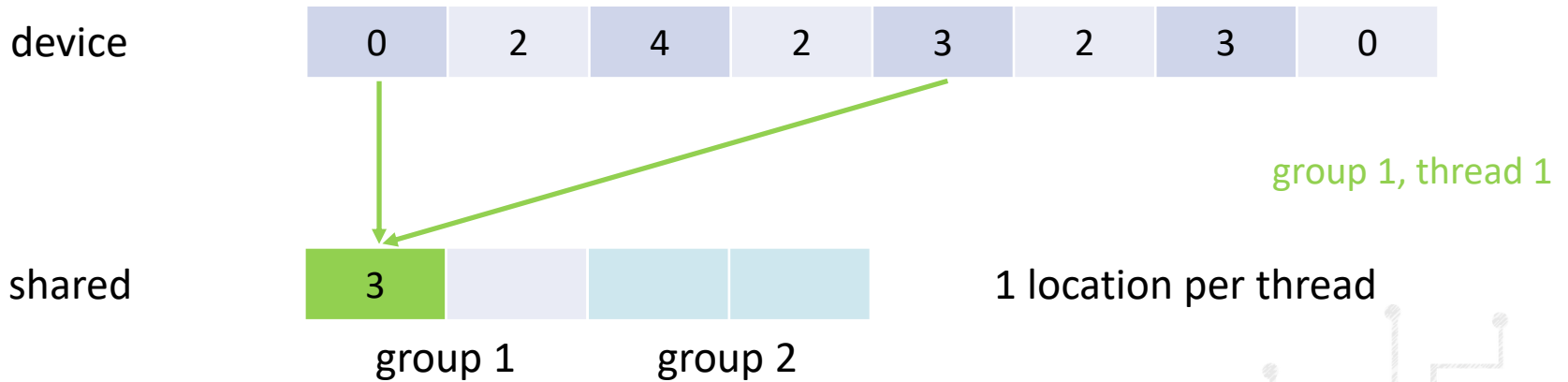
2 thread groups, 2 threads/group = 4 threads





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device



shared

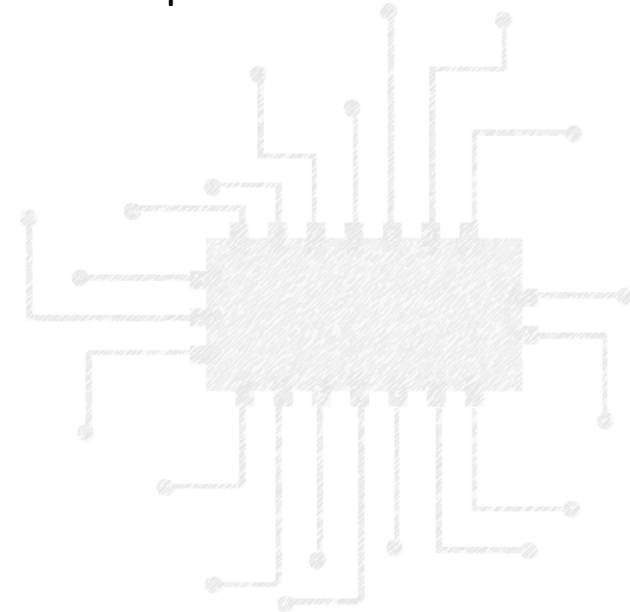


group 1

group 2

group 1, thread 2

1 location per thread

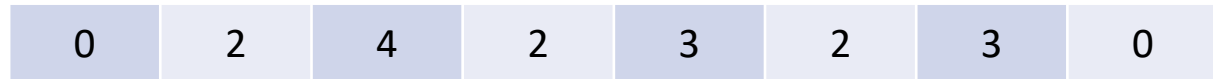




Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device



shared

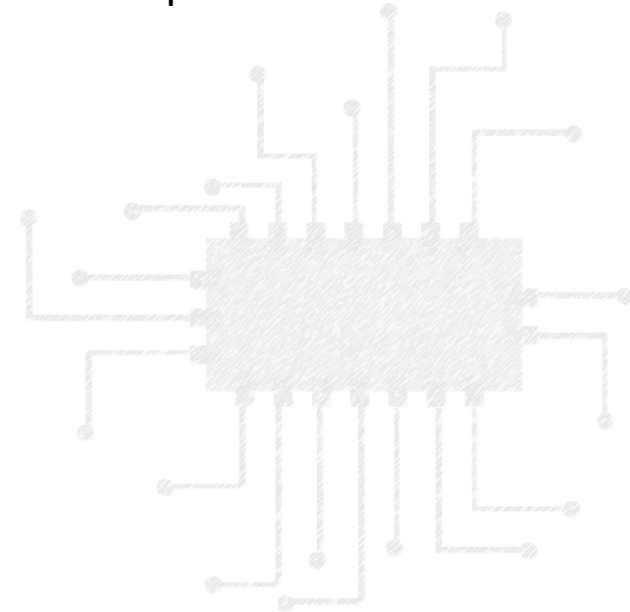


group 1

group 2

group 2, thread 1

1 location per thread





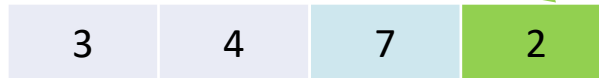
Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device



shared

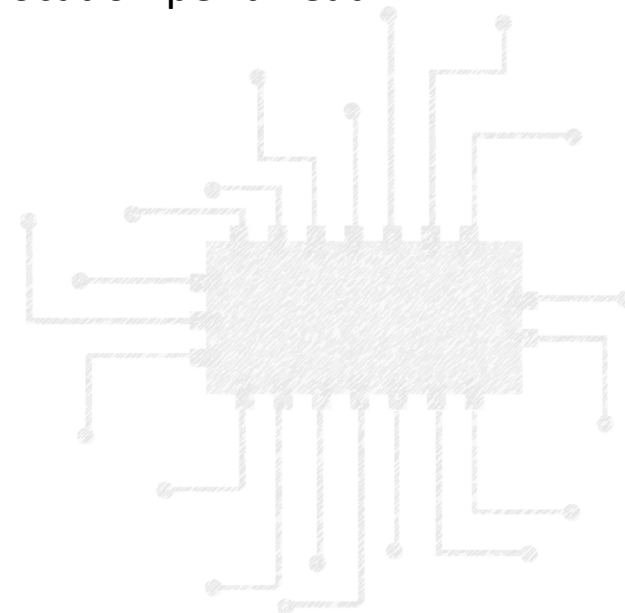


group 1

group 2

1 location per thread

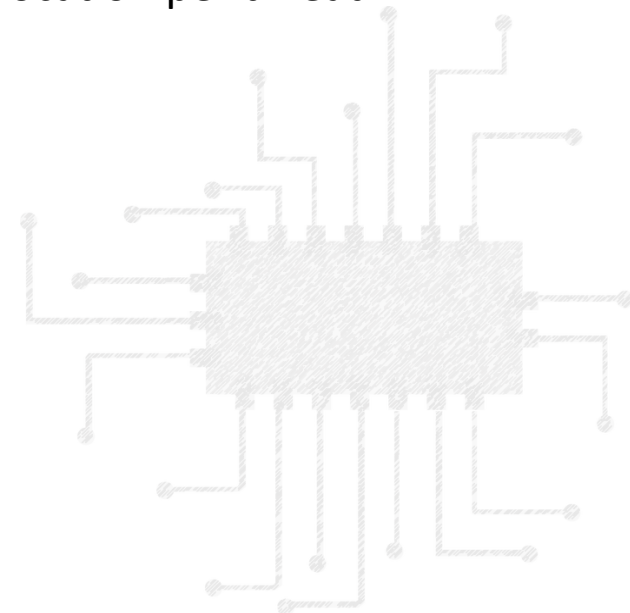
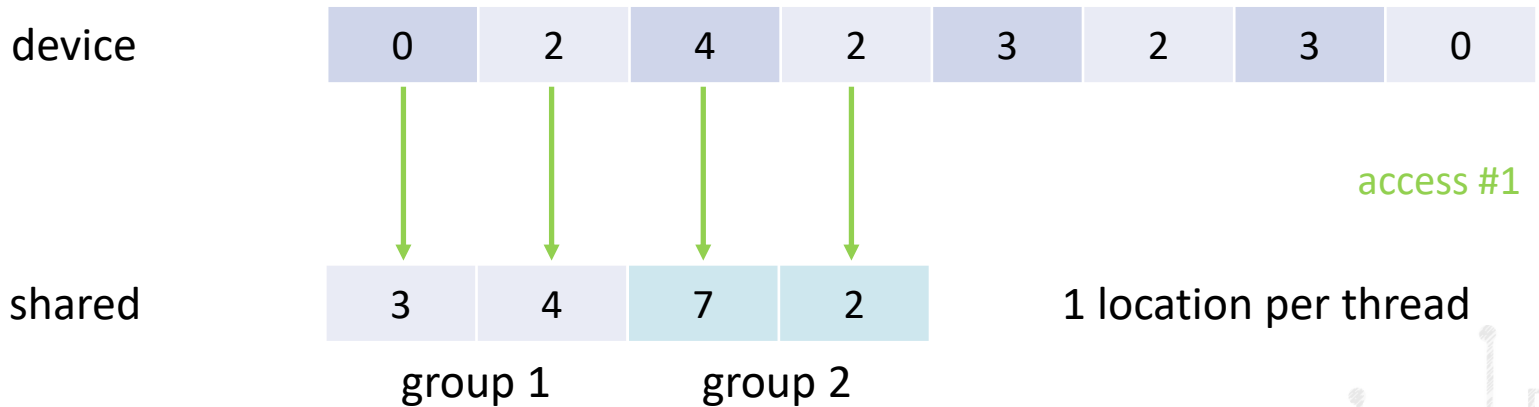
group 2, thread 2





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads





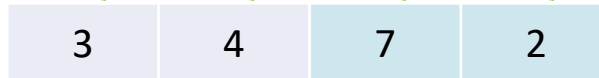
Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device



shared

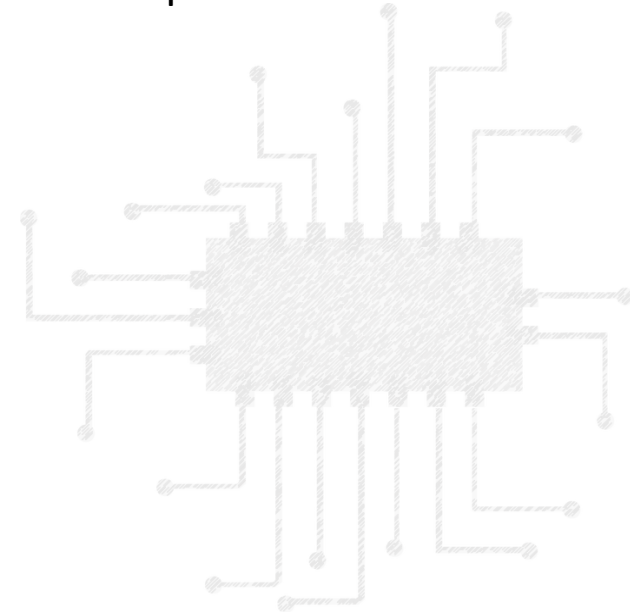


group 1

group 2

access #2

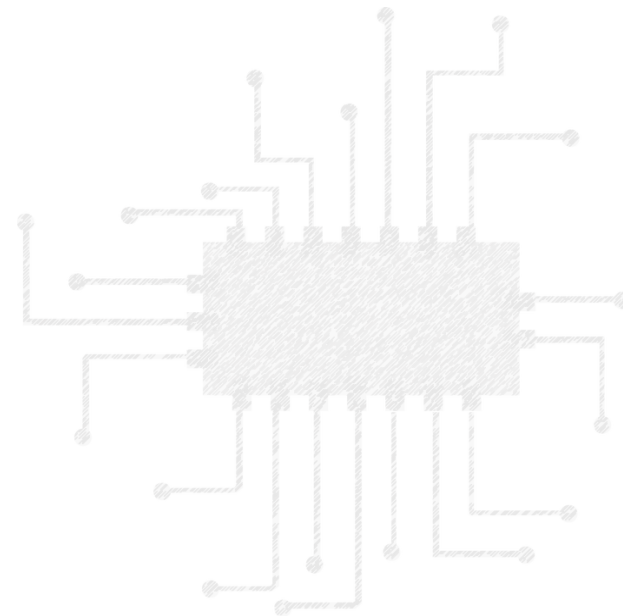
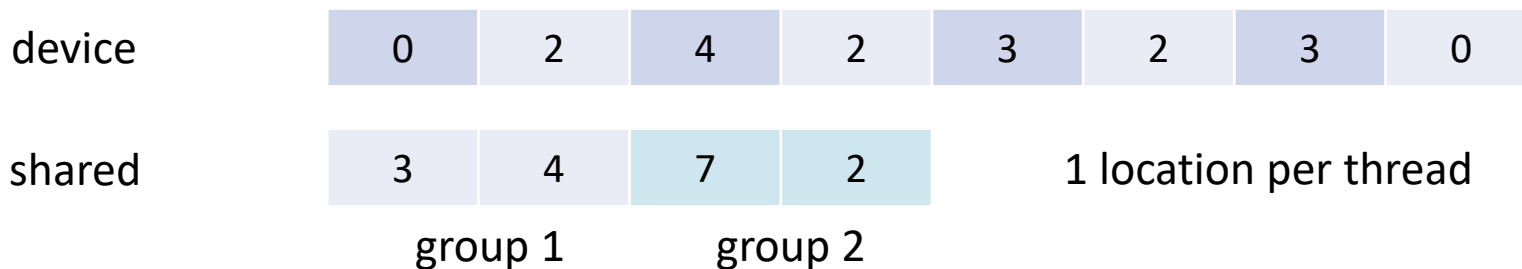
1 location per thread





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device

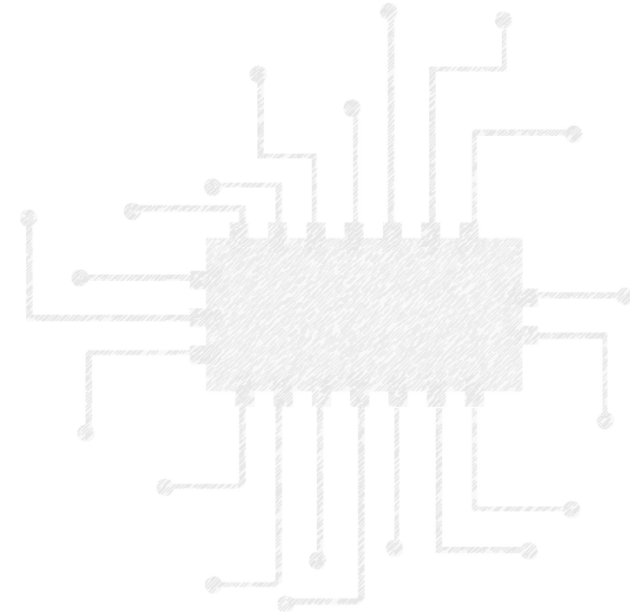
0	2	4	2	3	2	3	0
---	---	---	---	---	---	---	---

shared

3	4	7	2
---	---	---	---

1 location per thread

..... shared synchronization (__syncthreads())

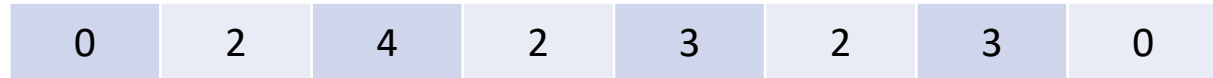




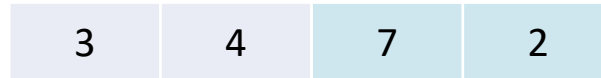
Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device



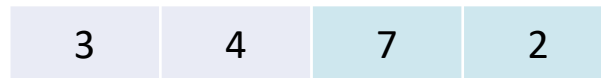
shared



1 location per thread

shared synchronization (`__syncthreads()`)

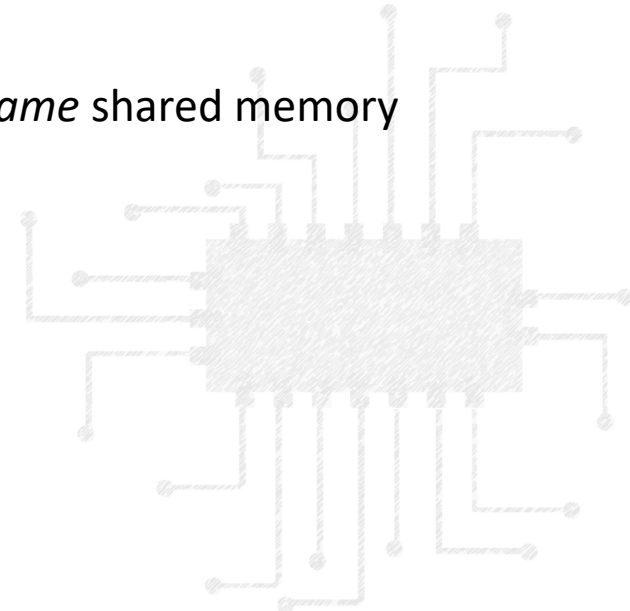
shared



group 1

group 2

Same shared memory

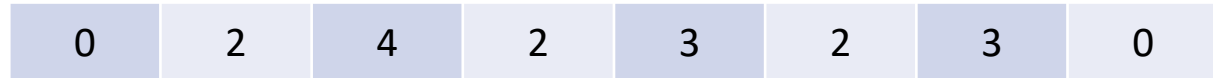




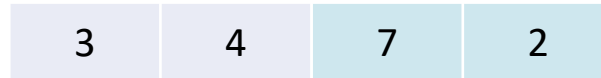
Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device



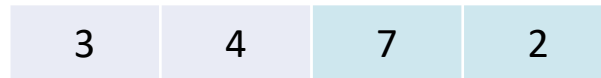
shared



1 location per thread

shared synchronization (`__syncthreads()`)

shared

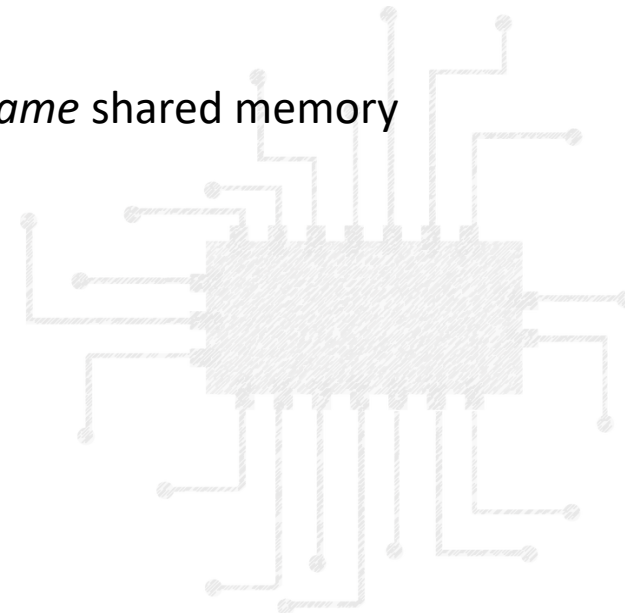


group 1

group 2

group 1, thread 1

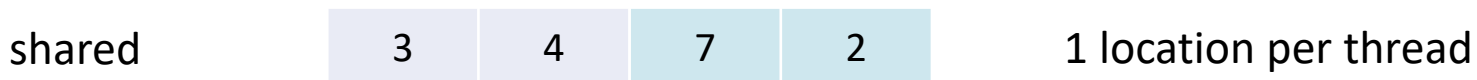
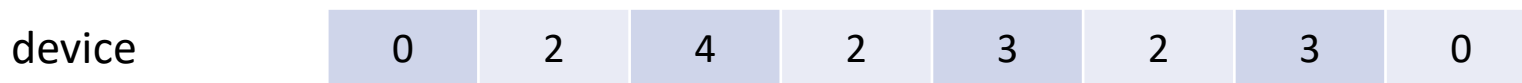
Same shared memory





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

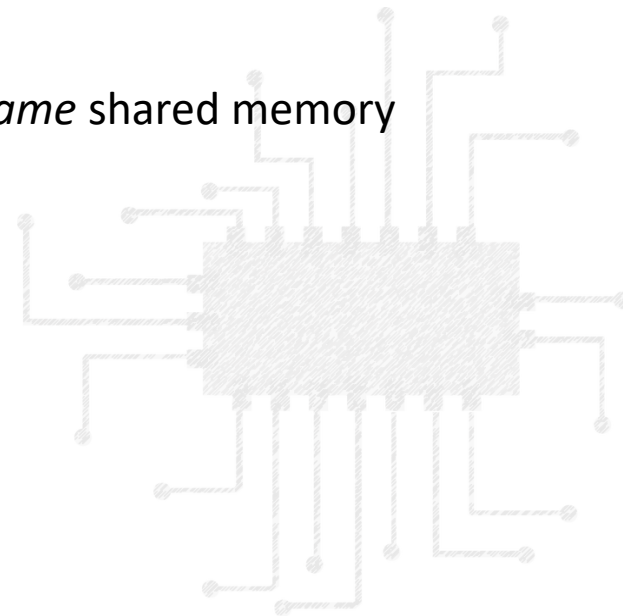


shared synchronization (`__syncthreads()`)



group 1, thread 1

Same shared memory





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

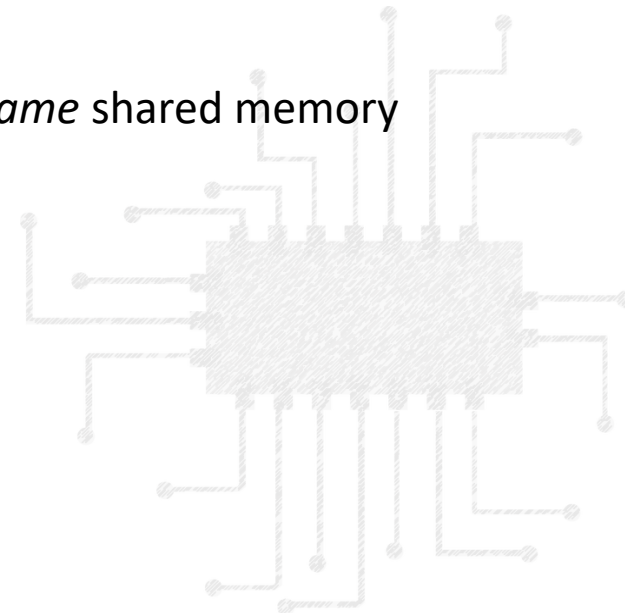


shared synchronization (`__syncthreads()`)



group 1, thread 1

Same shared memory





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2
--------	---	---	---	---

1 location per thread

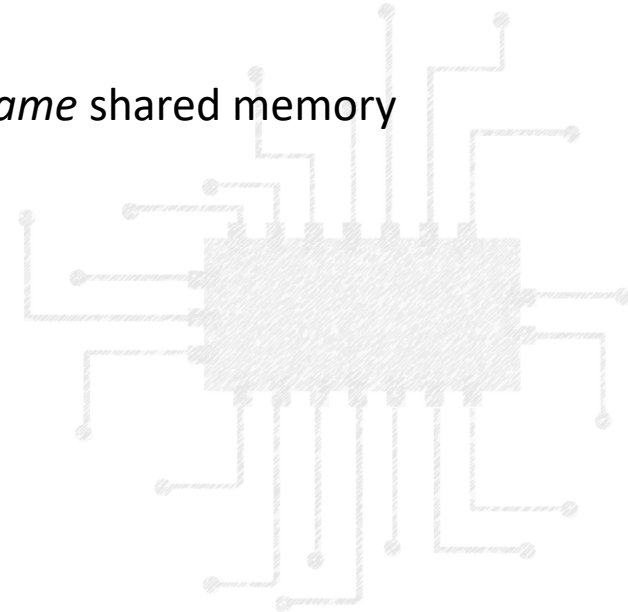
shared synchronization (`__syncthreads()`)

shared	7	4	7	2
--------	---	---	---	---

group 1 group 2

group 1, thread 1

Same shared memory





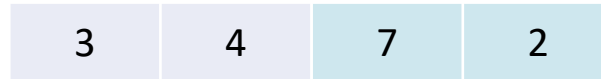
Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device



shared



1 location per thread

shared synchronization (`__syncthreads()`)

shared

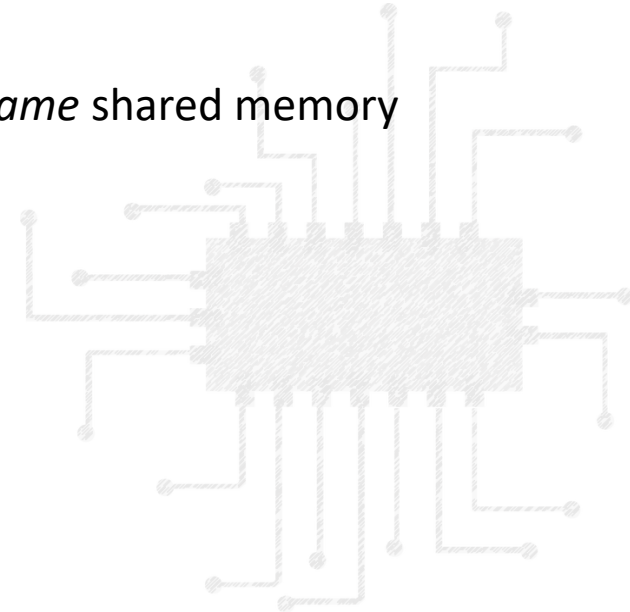


group 1

group 2

group 2, thread 1

Same shared memory





Parallel Reductions (SUM)

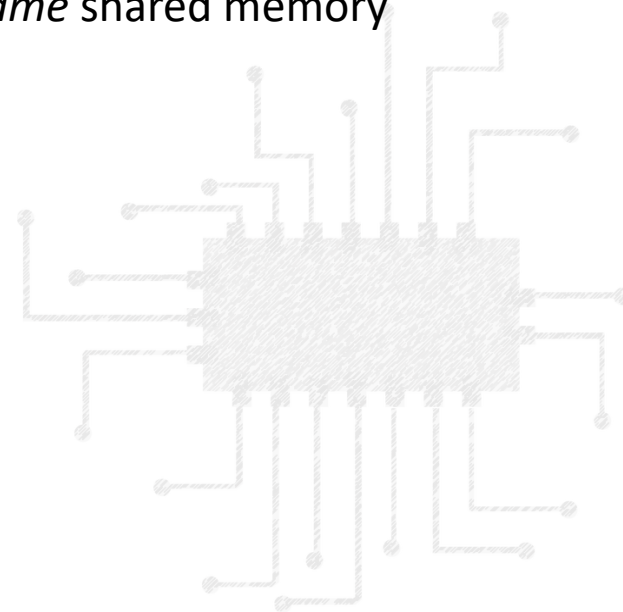
2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
	group 1		group 2					





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

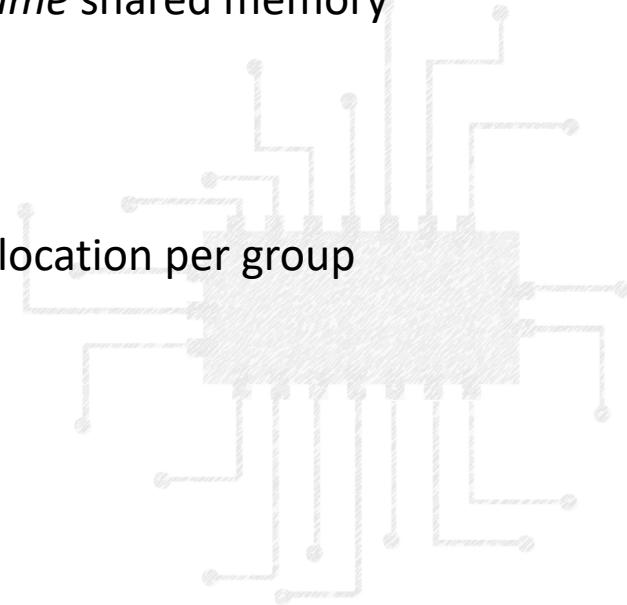
shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
--------	---	---	---	---	---------------------------	--	--	--

device		
--------	--	--

1 location per group





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

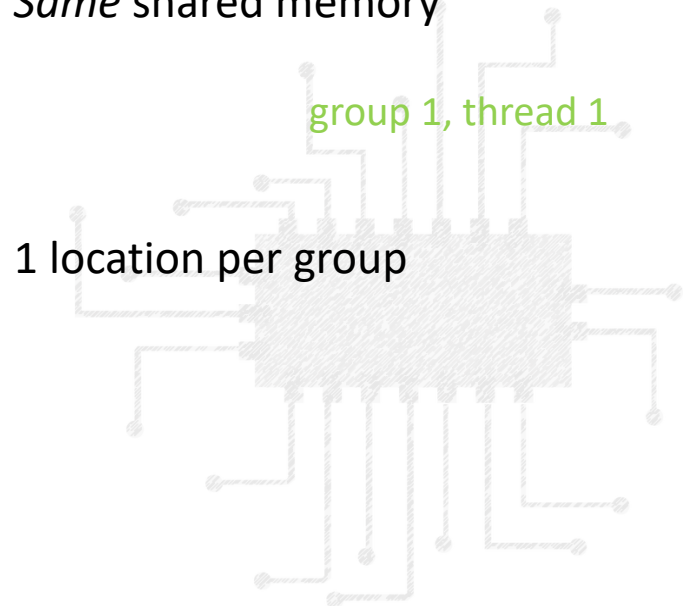
device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
--------	---	---	---	---	---------------------------	--	--	--

device		
--------	--	--





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

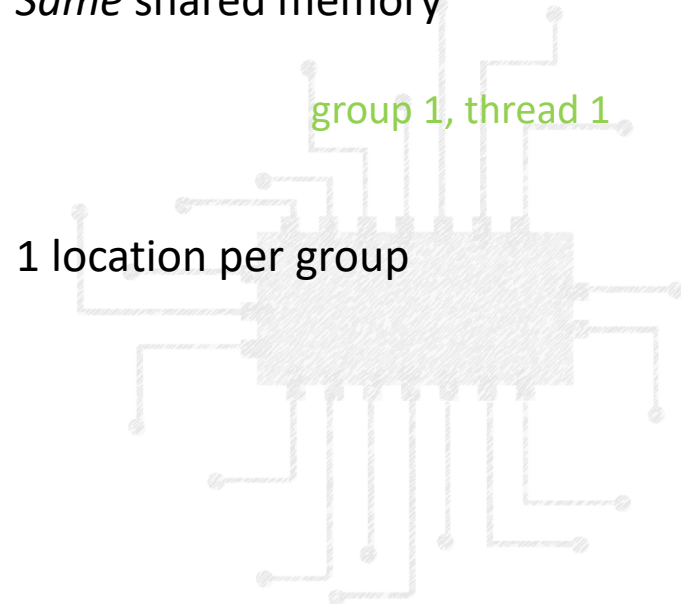
device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
--------	---	---	---	---	---------------------------	--	--	--

device		
--------	--	--





Parallel Reductions (SUM)

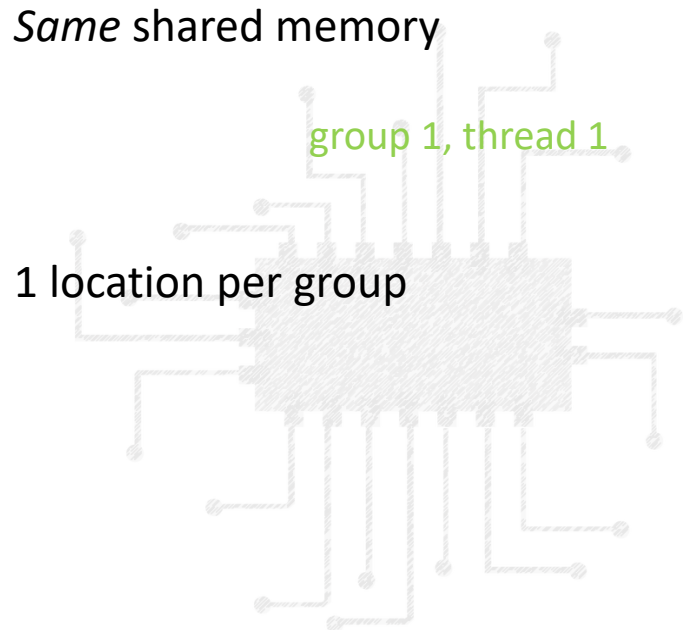
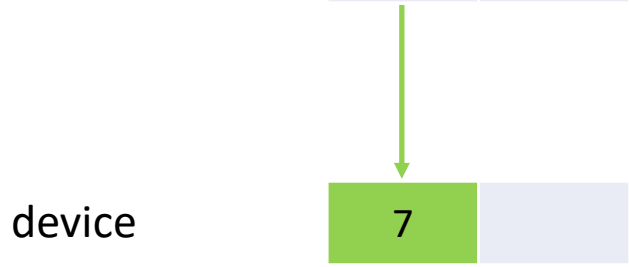
2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
--------	---	---	---	---	---------------------------	--	--	--





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

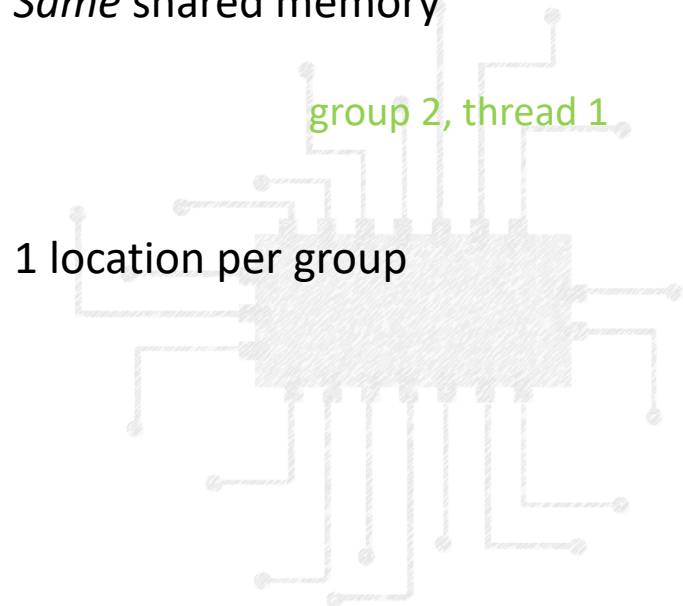
device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
--------	---	---	---	---	---------------------------	--	--	--

device	7							
--------	---	--	--	--	--	--	--	--





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2
--------	---	---	---	---

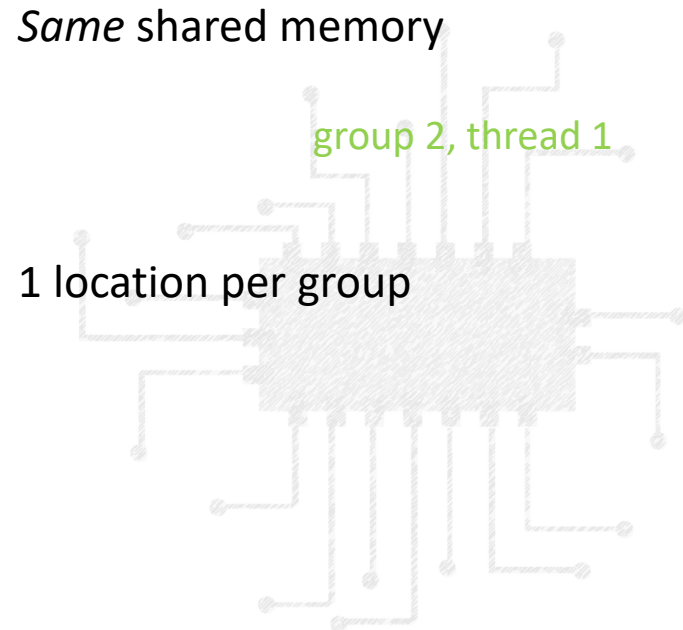
1 location per thread

shared synchronization (`__syncthreads()`)

shared	7	4	9	2
--------	---	---	---	---

Same shared memory

device	7	
--------	---	--





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

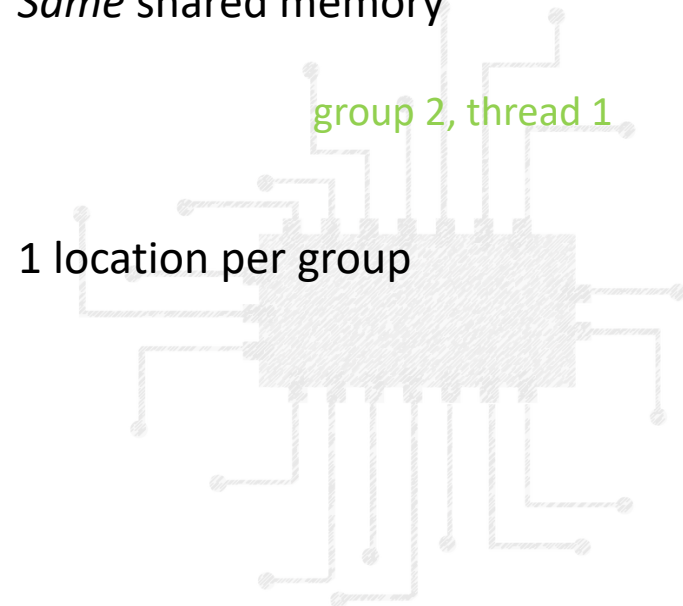
device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
--------	---	---	---	---	---------------------------	--	--	--

device	7	9
--------	---	---





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

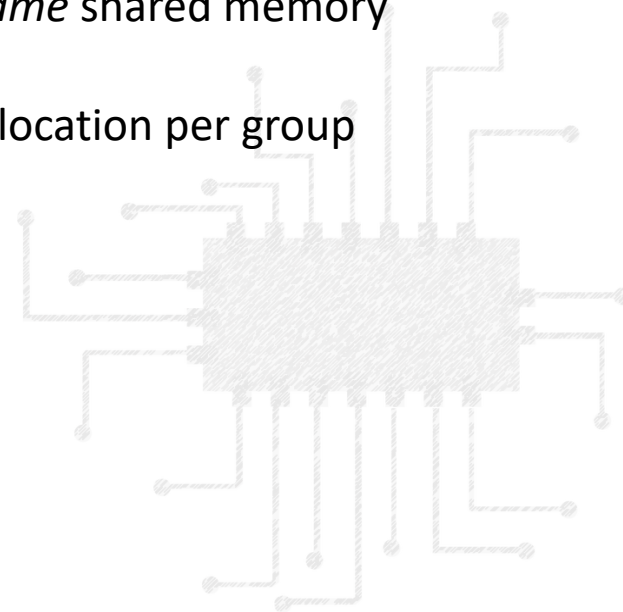
shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
--------	---	---	---	---	---------------------------	--	--	--

device	7	9						
--------	---	---	--	--	--	--	--	--

1 location per group





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

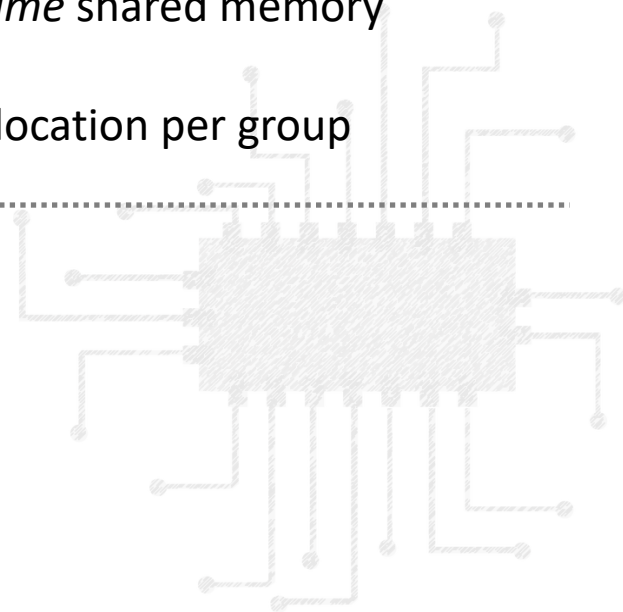
shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
--------	---	---	---	---	---------------------------	--	--	--

device	7	9	1 location per group					
--------	---	---	----------------------	--	--	--	--	--

device synchronization (kernel boundary)





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

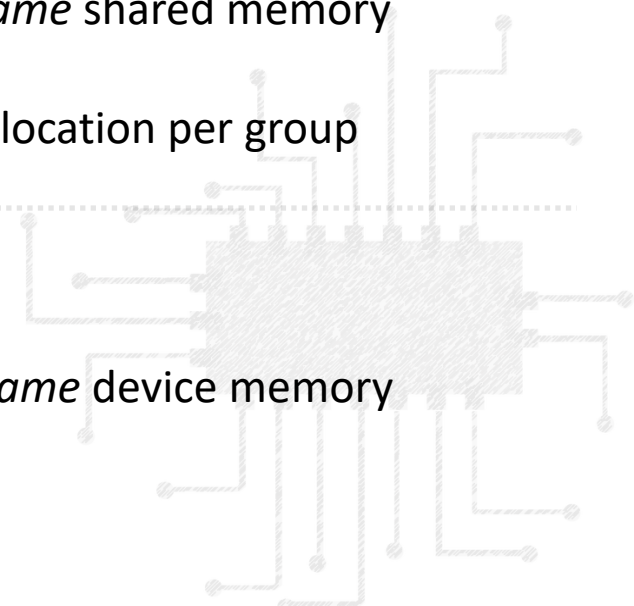
shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
--------	---	---	---	---	---------------------------	--	--	--

device	7	9	1 location per group					
--------	---	---	----------------------	--	--	--	--	--

device synchronization (kernel boundary)

device	7	9	<i>Same</i> device memory					
--------	---	---	---------------------------	--	--	--	--	--





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

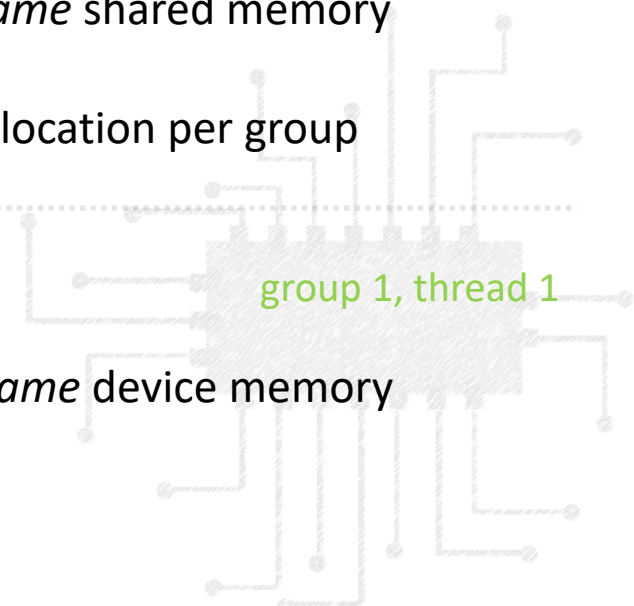
shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
--------	---	---	---	---	---------------------------	--	--	--

device	7	9	1 location per group					
--------	---	---	----------------------	--	--	--	--	--

device synchronization (kernel boundary)

device	7	9	<i>Same</i> device memory					
--------	---	---	---------------------------	--	--	--	--	--





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

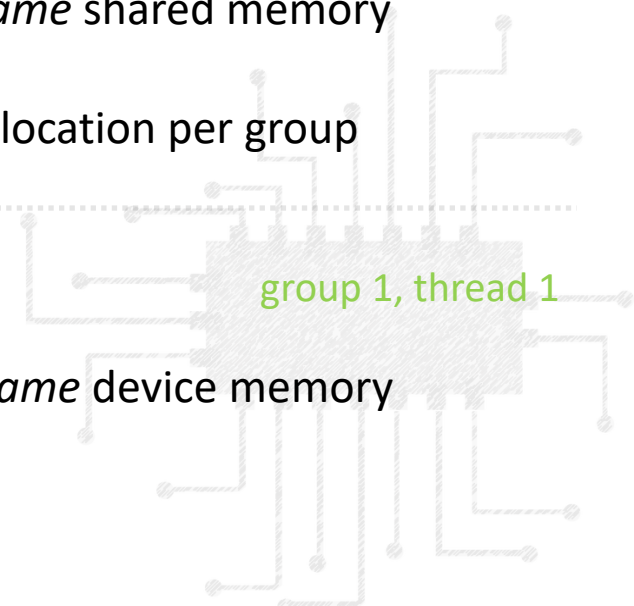
shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same</i> shared memory			
--------	---	---	---	---	---------------------------	--	--	--

device	7	9	1 location per group					
--------	---	---	----------------------	--	--	--	--	--

device synchronization (kernel boundary)

device	7	9	<i>Same</i> device memory					
--------	---	---	---------------------------	--	--	--	--	--





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

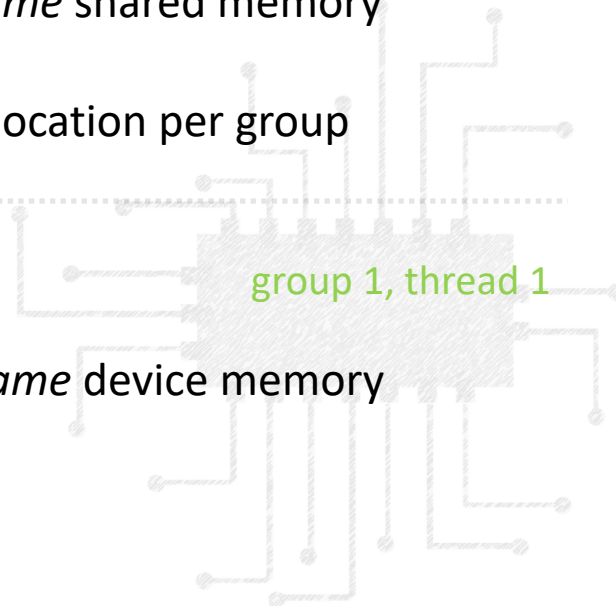
shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same shared memory</i>			
--------	---	---	---	---	---------------------------	--	--	--

device	7	9	1 location per group					
--------	---	---	----------------------	--	--	--	--	--

device synchronization (kernel boundary)

device	7	9	<i>Same device memory</i>					
--------	---	---	---------------------------	--	--	--	--	--





Parallel Reductions (SUM)

2 thread groups, 2 threads/group = 4 threads

device	0	2	4	2	3	2	3	0
--------	---	---	---	---	---	---	---	---

shared	3	4	7	2	1 location per thread			
--------	---	---	---	---	-----------------------	--	--	--

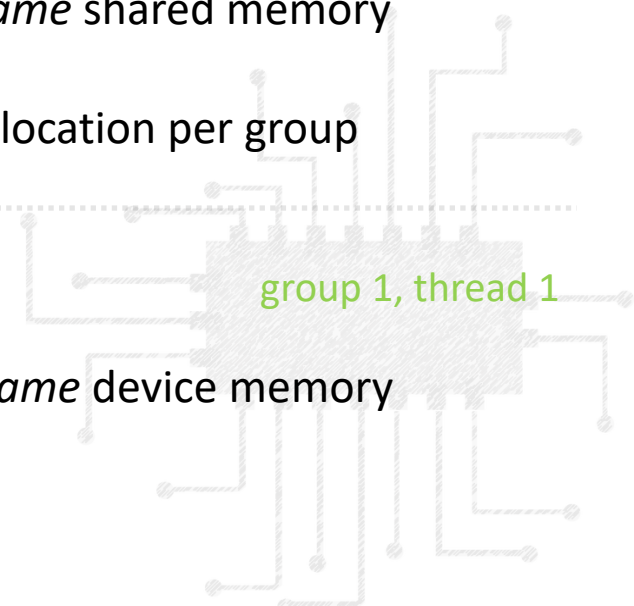
shared synchronization (`__syncthreads()`)

shared	7	4	9	2	<i>Same shared memory</i>			
--------	---	---	---	---	---------------------------	--	--	--

device	7	9	1 location per group					
--------	---	---	----------------------	--	--	--	--	--

device synchronization (kernel boundary)

device	16	9	<i>Same device memory</i>					
--------	----	---	---------------------------	--	--	--	--	--





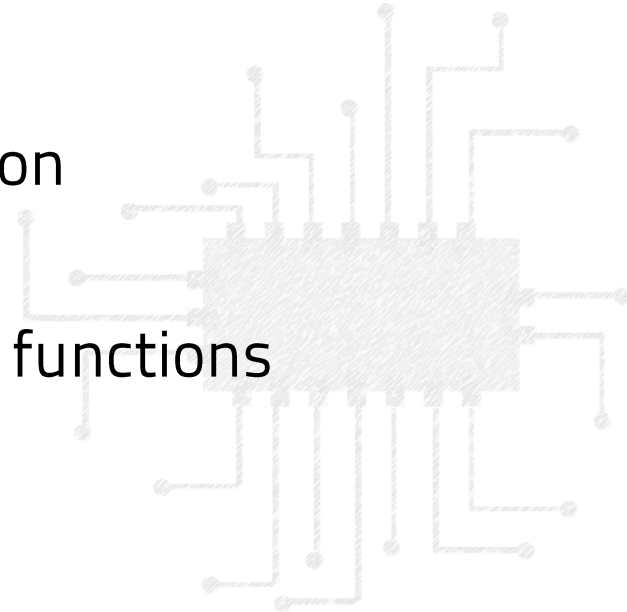
Parallel Thread Execution ISA

PTX

- Low-level virtual machine and instruction set architecture
- First released by NVIDIA in 2007

Goals

- Futureproof across multiple GPU generations
- Machine independent
- Common ISA for optimizers and distribution
- Support scalable targets
- "Easy" hand-coding for high performance functions





Types

Bit Types

- .pred (1-bit)
- .b8
- .b16
- .b32
- .b64

Unsigned Types

- .u8
- .u16
- .u32
- .u64

Signed Types

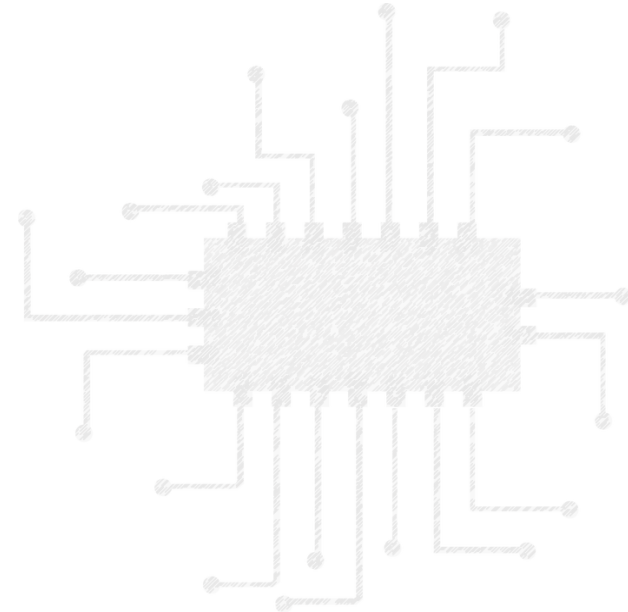
- .s8
- .s16
- .s32
- .s64

Float Types

- .f16
- .f16x2
- .f32
- .f64

Vector Types

- .v2 <type>
- .v4 <type>

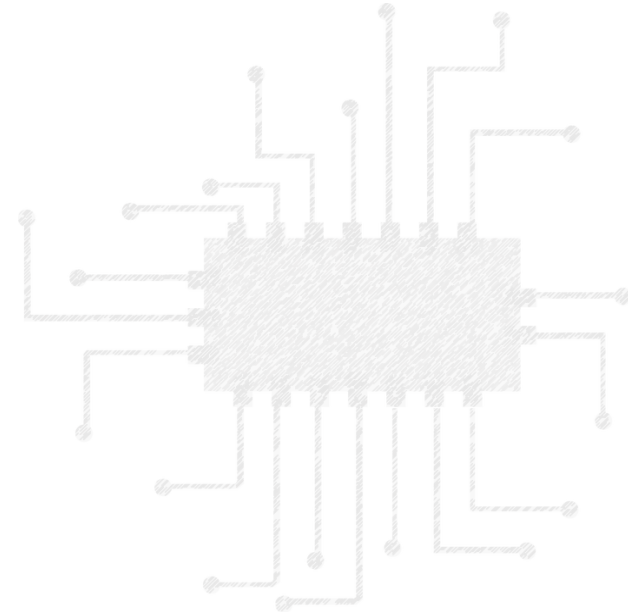




State Spaces

A **state space** is a storage area for variables.

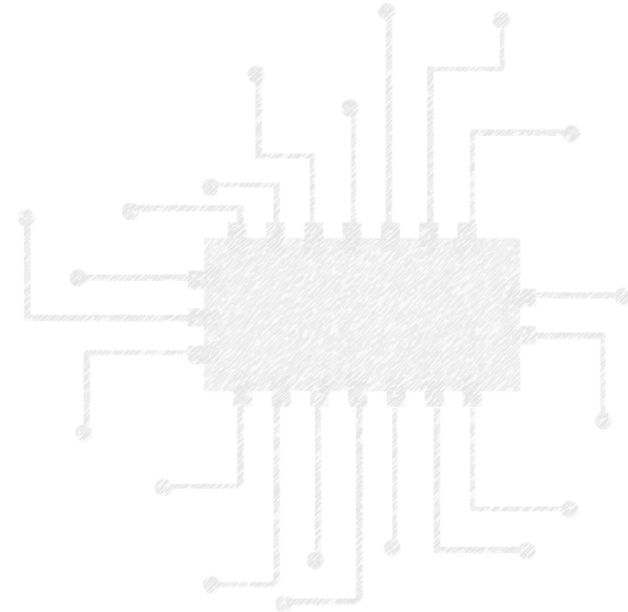
- Characteristics
 - Size
 - Addressability
 - Access speed
 - Access rights (R/W or RO)
 - Sharing (private, CTA, global)





Register Spaces

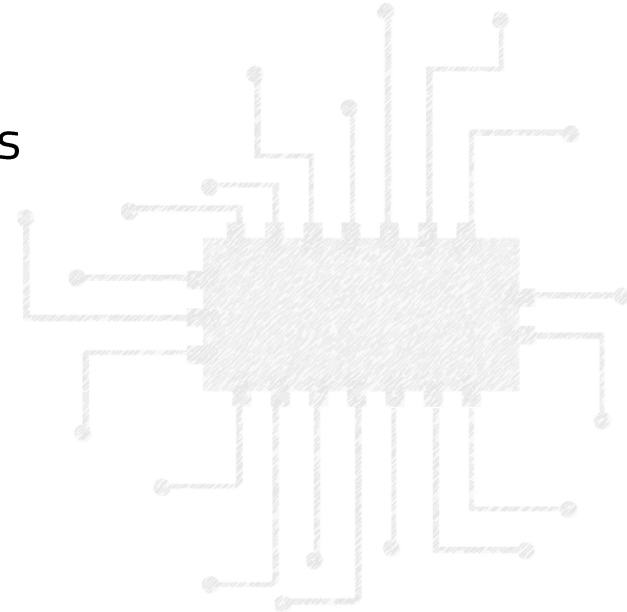
- .reg state space
- Characteristics
 - Fast
 - R/W
 - Private to a thread
- Special register spaces
 - .sreg state spaces
 - Predefined registers (i.e. tid, ctaid, ...)





Addressable Spaces

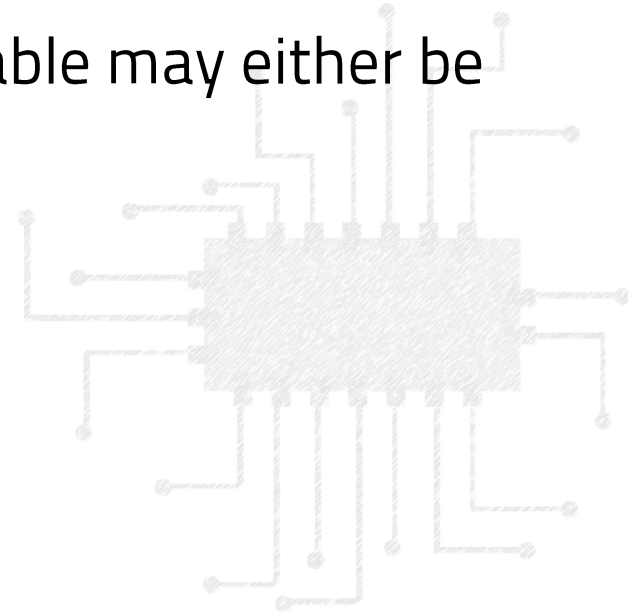
- `.global`, `.local`, `.param` state spaces
- Characteristics
 - Slower
 - R/W (mostly)
 - Shared among CTAs or the CUDA context (`.global`)
- Parameter spaces
 - May contain pointers to other address spaces
 - Used for entry and device (non-entry) functions





Variables

- A variable in PTX consists of
 - Name
 - Type
 - State space
- Registers vs. other variables are distinguished by the state space
- Note: The address of an addressable variable may either be generic, or specific to a state space

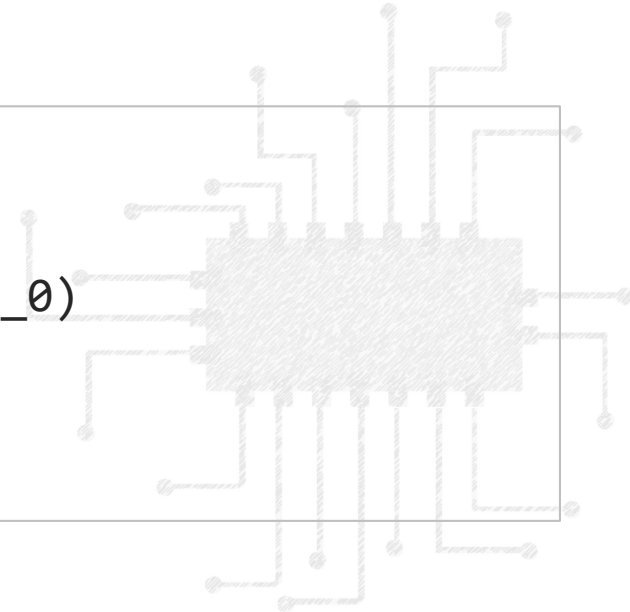




Modules

- At the highest level, a PTX program consists of a module
- Modules consist of
 - PTX version
 - Target device
 - Address bits (32 or 64)
 - List of functions

```
.version 6.1
.target sm_61
.address_size 64
.visible .entry AddTest(.param .u64 AddTest_0)
{
    [...]
}
```





Functions

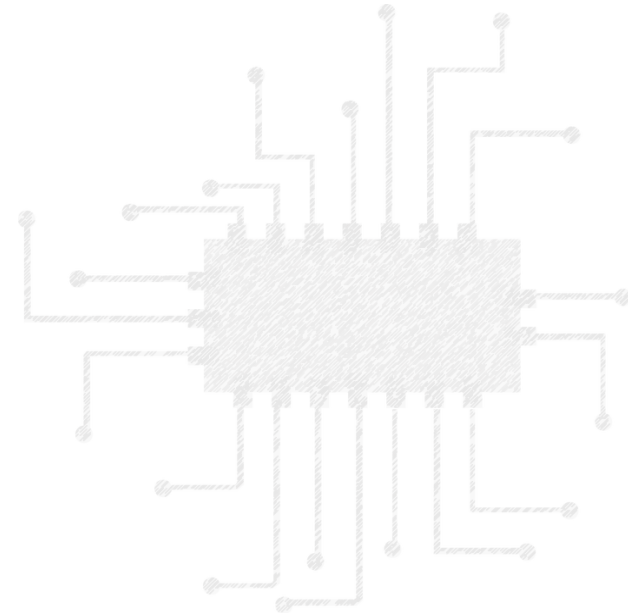
- Entry functions
 - Entry points to a kernel
 - Can be called by the CUDA API
- Device functions
 - Can only be called by other PTX functions
- Parameters/return values are either in the `.param` or `.reg` state spaces

```
.visible .entry AddTest(.param .u64 AddTest_0) { [...] }  
  
.func (.reg .u64 Return) Function(.param .f64 Return_0) { [...] }
```



Instruction Categories

- Arithmetic
- Logical
- Data movement
- Comparison
- Control flow
- Synchronization (memory consistency)





Arithmetic

- The full PTX instruction opcodes typically have
 - Mnemonic
 - Modifiers
 - Operand type

Add instruction

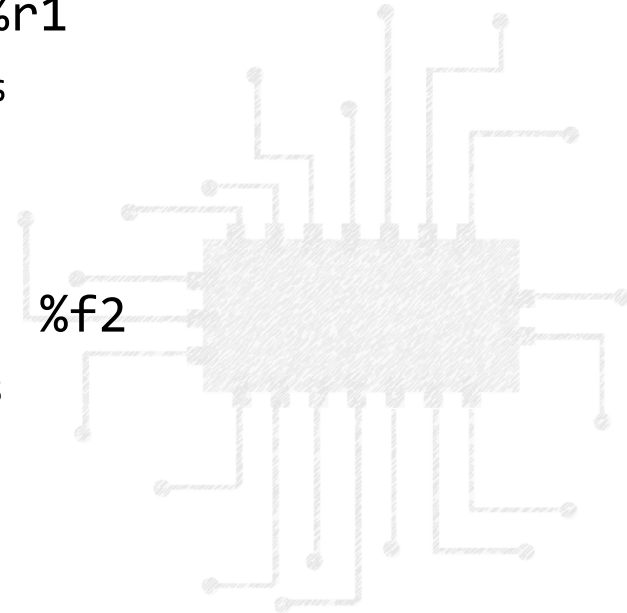
```
add.sat.s32 %r2, %r0, %r1
```

where %r0-%r1 are s32 registers

FMA instruction

```
fma.rn.f16 %f3, %f0, %f1, %f2
```

where %f0-%f3 are f16 registers





Data Movement

- Data movement instructions move data between state spaces

Move instruction

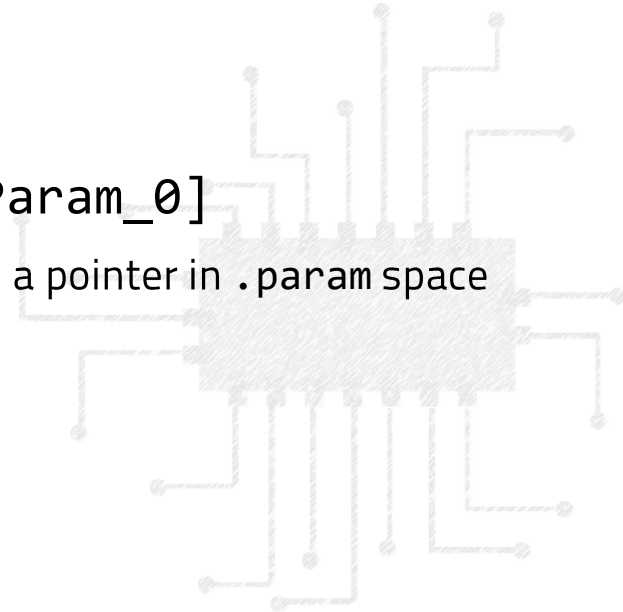
```
mov.u32 %r0, %tid.x
```

where %r0 is a u32 .reg variable, and %tid.x is a u32 .sreg variable

Load instruction

```
ld.param.u64 %rd0, [ExampleParam_0]
```

where %rd0 is a u64 .reg variable, and ExampleParam_0 is a pointer in .param space





Control Flow

- Control flow in PTX uses predicated execution

```
@p add.sat.s32 %r2, %r0, %r1
```

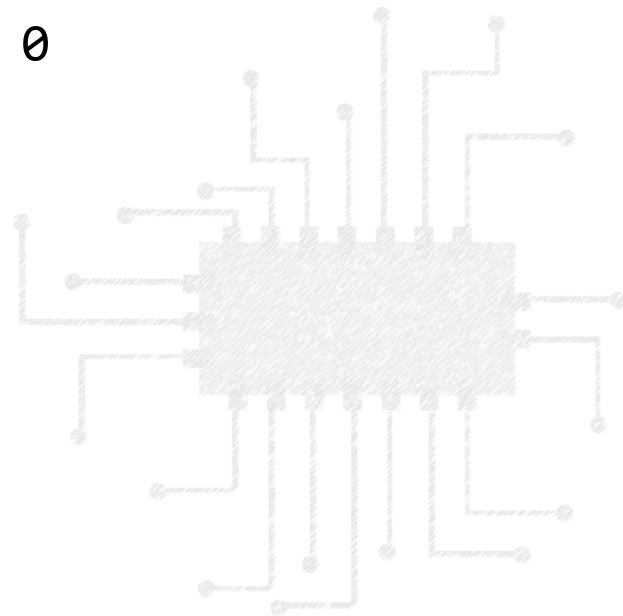
p is a .pred register with 1-bit for each thread (1-execute, 0-skip)

Set predicate instruction

```
setp.eq.u32 p, %tid.x, 0
```

Branch instruction

```
@p bra target_label
```



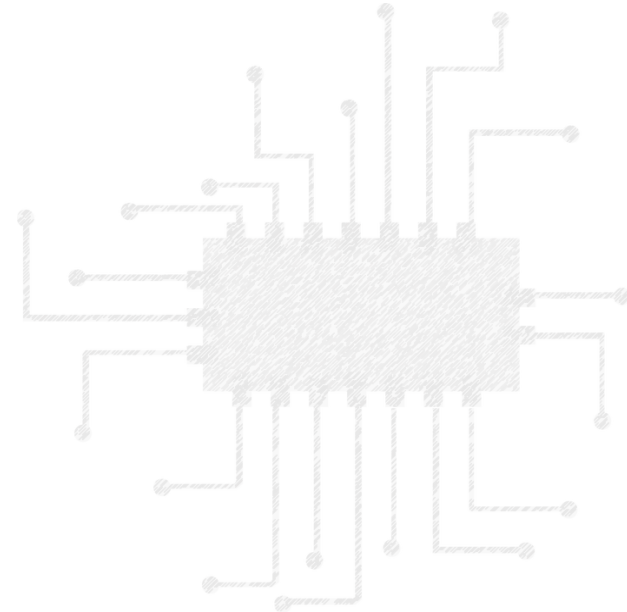


Memory Consistency

- Sync, barrier, fence, ... operations ensure a consistent view of memory from multiple threads

Barrier instruction

```
bar.warp.sync 0xffffffff
```





Example: Add

```
.version 6.1
.target sm_61
.address_size 64
.visible .entry AddTest(.param .u64 .ptr.align 8 AddTest_0)
{
    .reg .u32 %r;           // variable declarations
    .reg .u64 %rd<4>;      // multiple declarations %rd0, %rd1...
    .reg .f64 %f<2>;
    ld.param.u64 %rd0, [AddTest_0]; // load ptr param
    cvta.to.global.u64 %rd1, %rd0; // convert generic to global ptr
    mov.u32 %r, %tid.x;     // load thread id
    mul.wide.u32 %rd2, %r, 8; // calc offset for thread
    add.u64 %rd3, %rd1, %rd2; // calc position for thread
    ld.global.f64 %f0, [%rd3]; // load current value
    add.f64 %f1, %f0, 2.000000; // increment by 2
    st.global.f64 [%rd3], %f1; // write value
    ret;
}
```



Example: If-Else

```
.version 6.1
.target sm_61
.address_size 64
.visible .entry ConditionalTest(.param .u64 ConditionalTest_0)
{
    .reg .pred p;                // predicate register
    [...]
    rem.u32 %r1, %tid.x, 2;       // check if even or odd
    setp.ne.u32 p, %r1, 0;        // set predicate
    @p bra false;                 // conditionally branch
    add.u32 %r2, %r0, 1;
    bra end;
false:
    add.u32 %r2, %r0, 2;
end:                              // converge point
    st.global.u32 [%rd3], %r2;
    ret;
}
```

