High-Performance Scientific Computing
Lecture 4: OpenCL

MATH-GA 2011 / CSCI-GA 2945 · September 26, 2012

Today

Tool of the day: Make

Chips for Throughput

OpenCL: Overview

OpenCL: Between host and device

OpenCL: Device Language

OpenCL: Synchronization

Bits and pieces

- HW1 graded before weekend
- HW2 due
- HW3 out
- Sign up for HPC account
- Any more OMP questions?
- OMP anecdote

Final project

Examples from two years ago:

- GPU-parallel finite difference solver in flexible geometries
- GPU-parallel password cracking
- MPI-parallel CFD via the vortex method
- GPU-parallel ruling extraction (geometry)

Remarks:

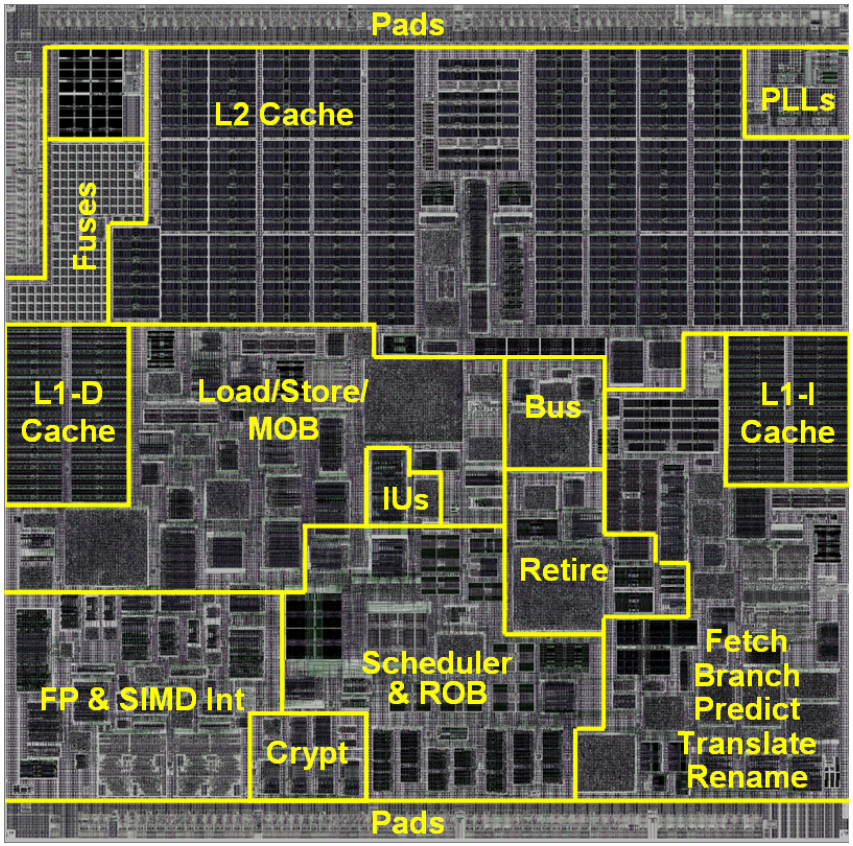
- Group projects encouraged!
- Use the mailing list to find buddies
- Non-numerical algorithms ok

Outline

Demo time

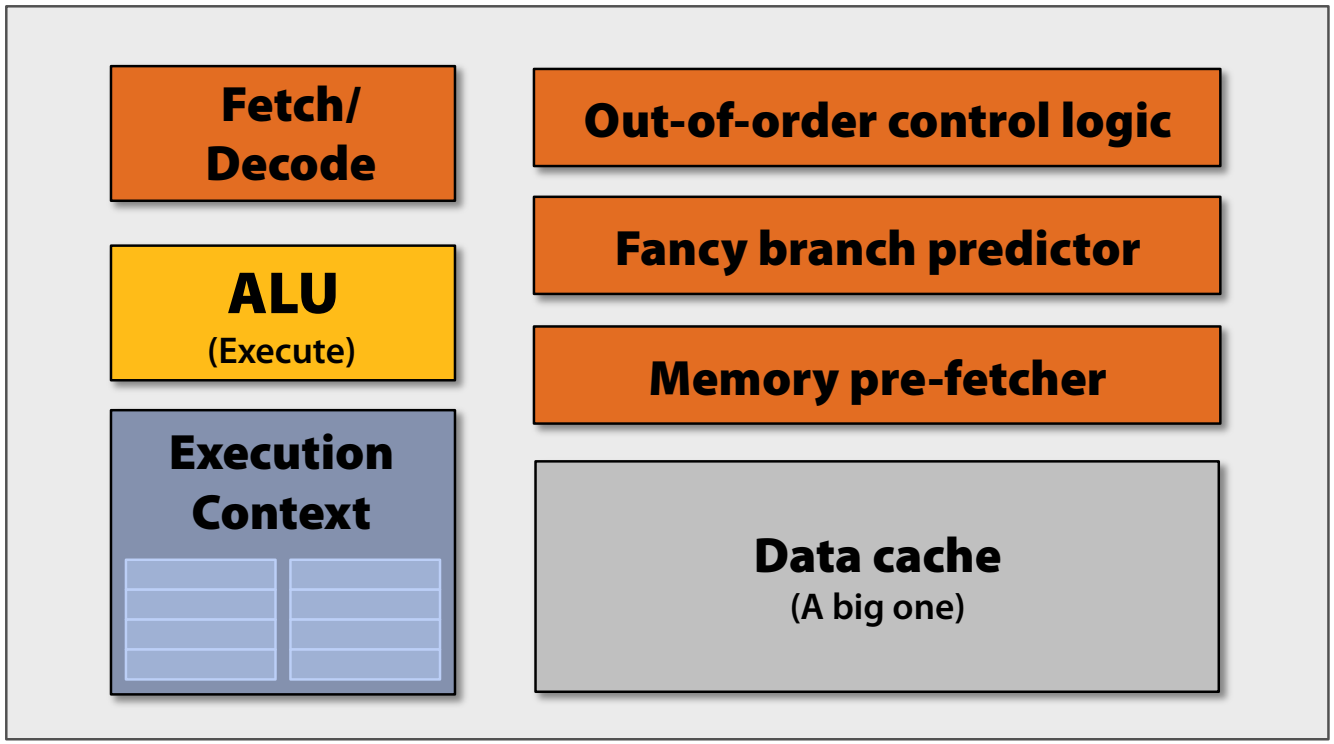
Outline

CPU Chip Real Estate



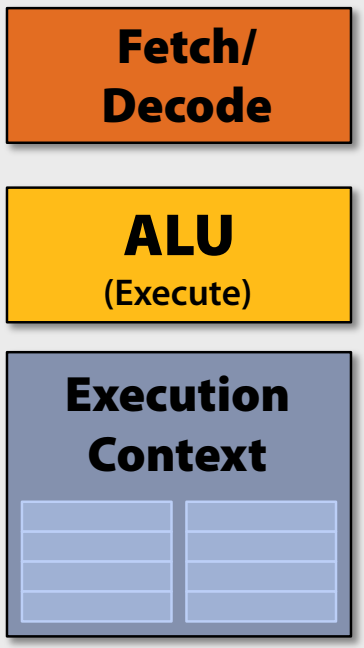
Die floorplan: VIA Isaiah (2008).
65 nm, 4 SP ops at a time, 1 MiB L2.

"CPU-style" Cores



Credit: Kayvon Fatahalian (Stanford)

Slimming down

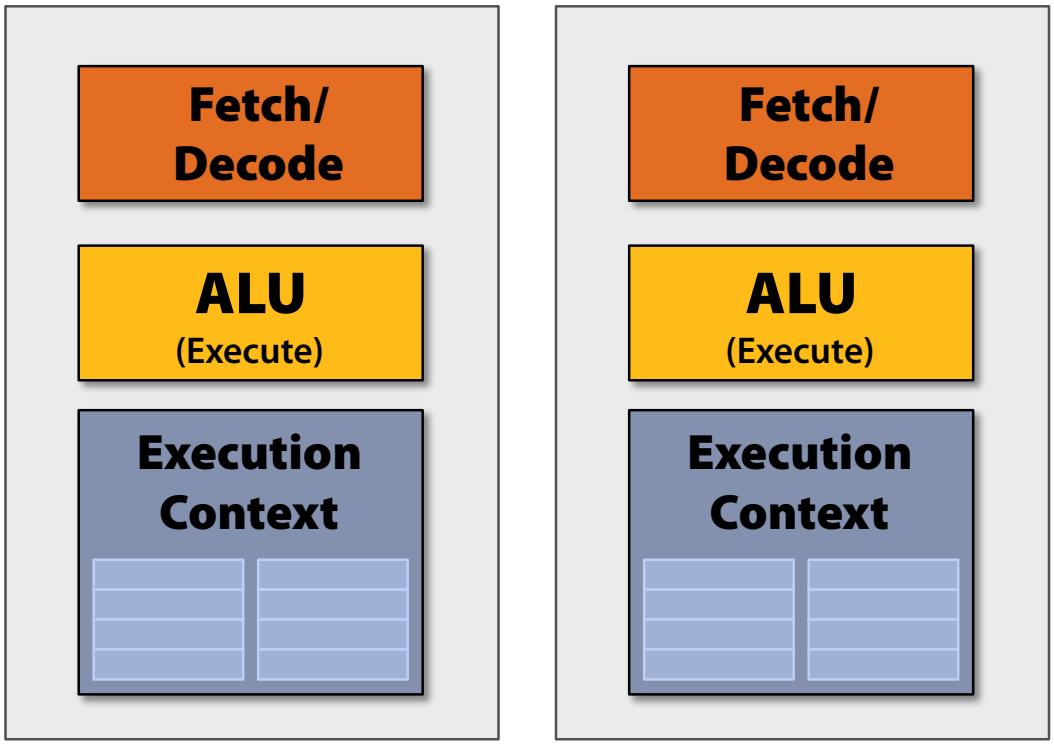


Idea #1:

Remove components that help a single instruction stream run fast

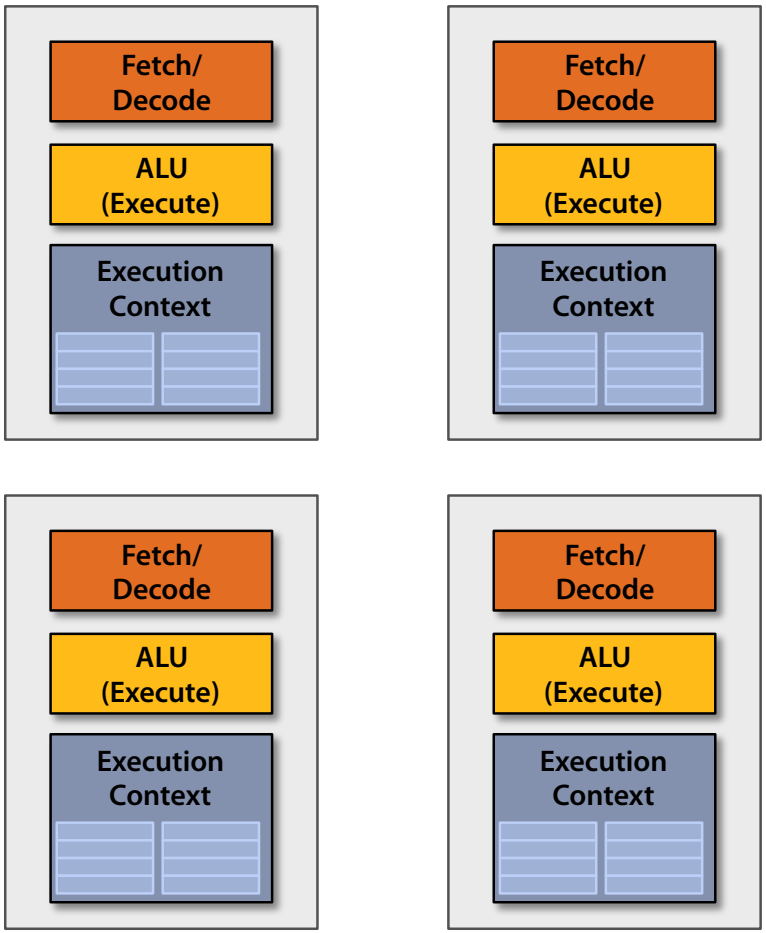
Credit: Kayvon Fatahalian (Stanford)

More Space: Double the Number of Cores



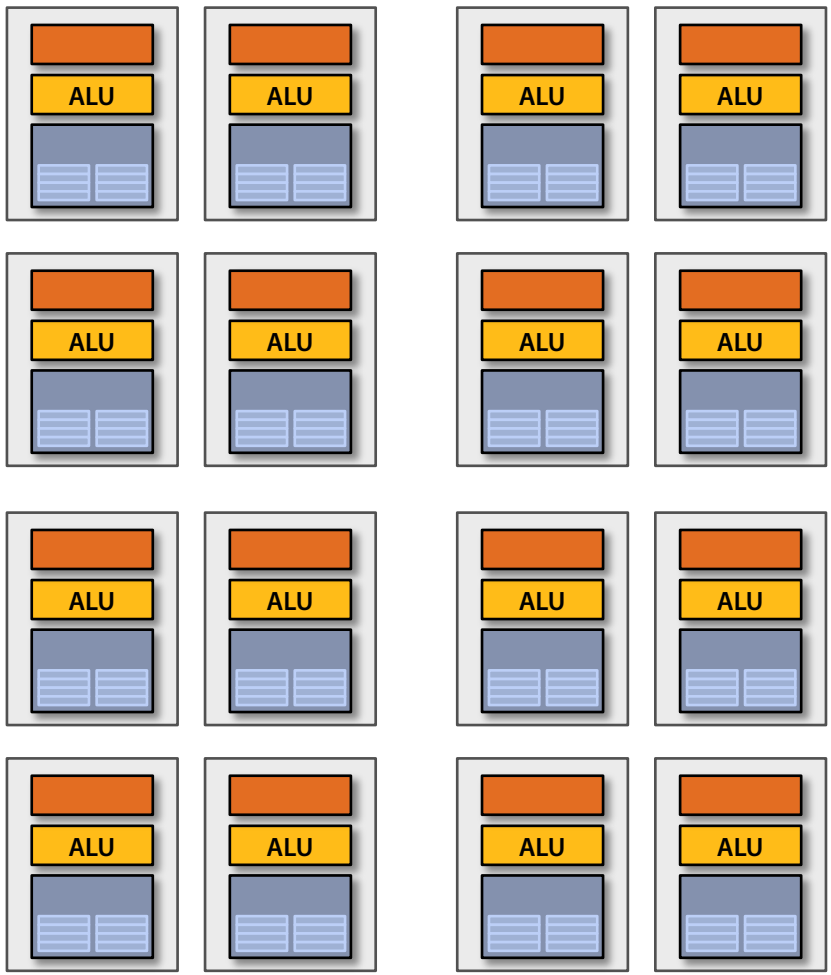
Credit: Kayvon Fatahalian (Stanford)

. . . again



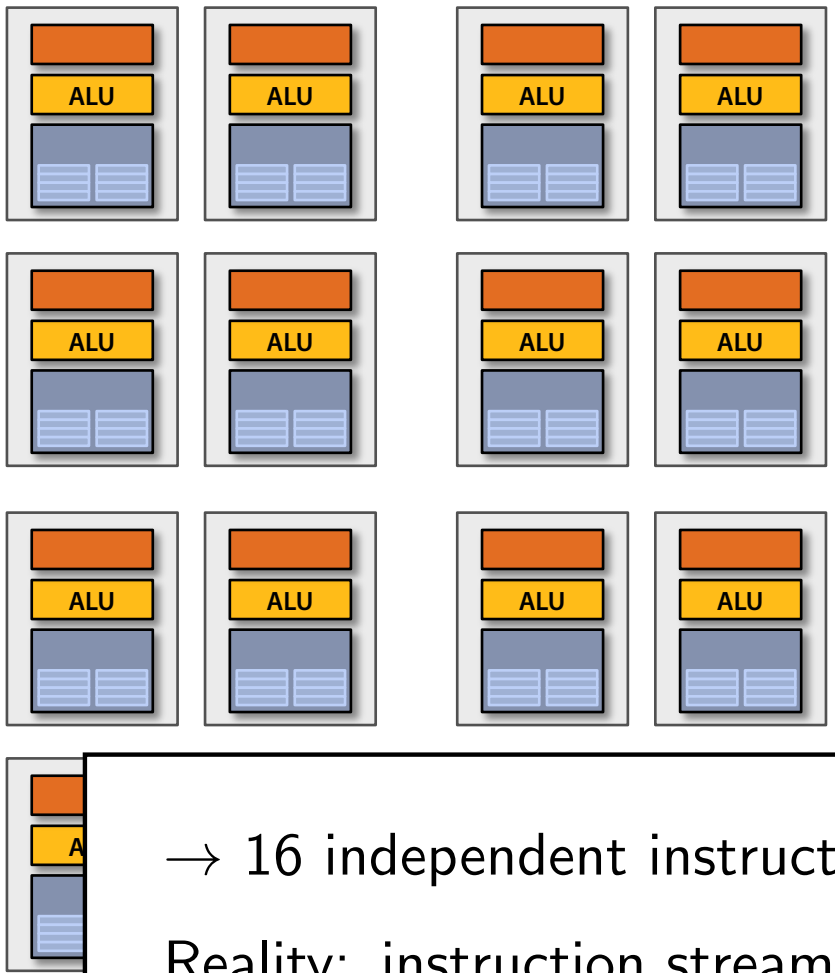
Credit: Kayvon Fatahalian (Stanford)

. . . and again



Credit: Kayvon Fatahalian (Stanford)

. . . and again

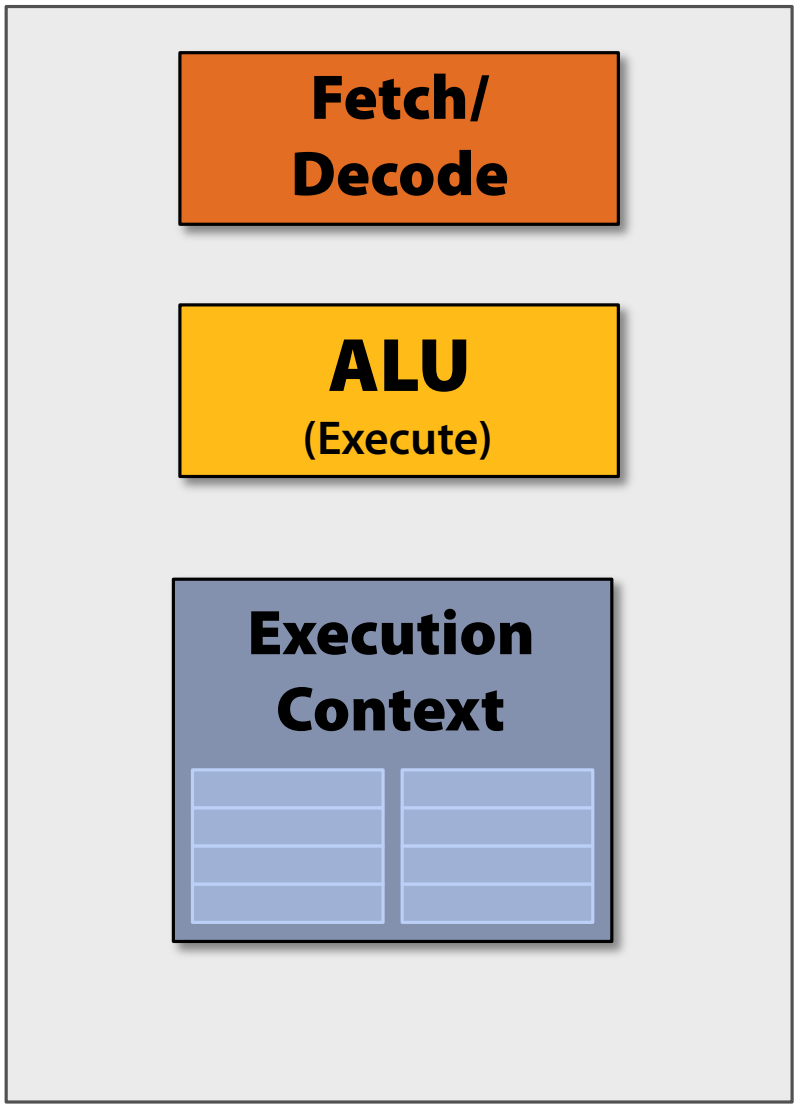


→ 16 independent instruction streams

Reality: instruction streams not actually very different/independent

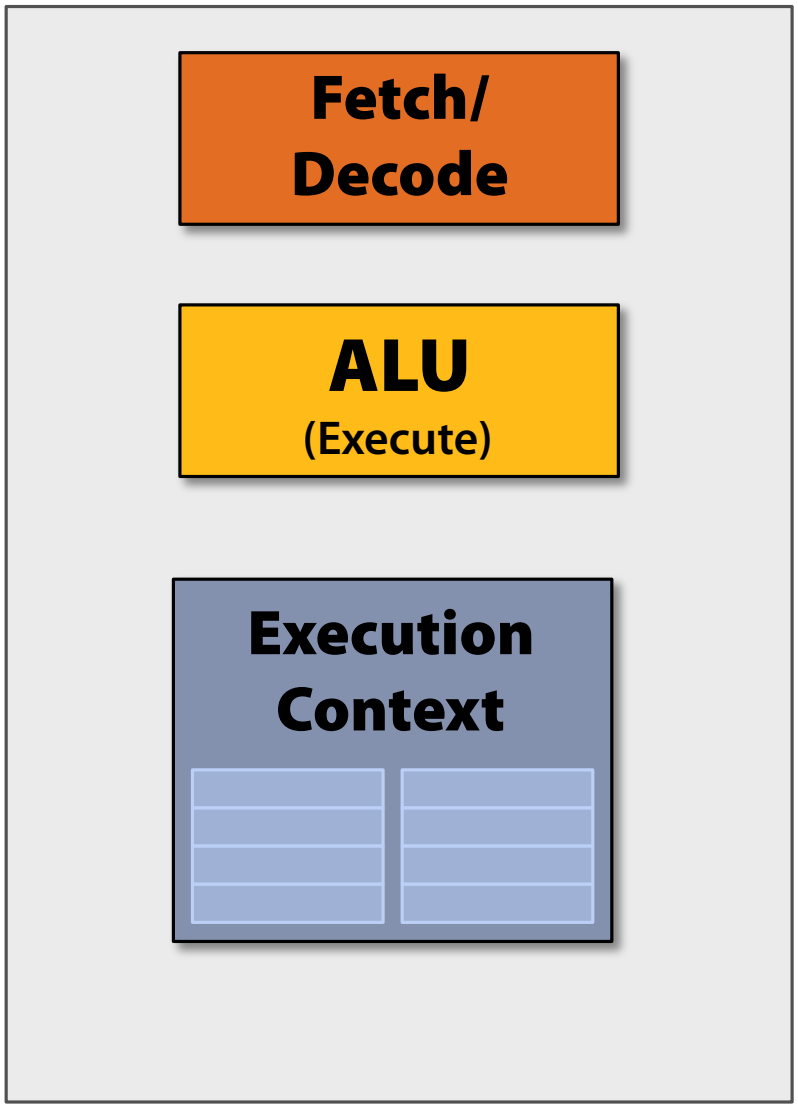
Credit: Kayvon Fatahalian (

Saving Yet More Space



Credit: Kayvon Fatahalian (Stanford)

Saving Yet More Space



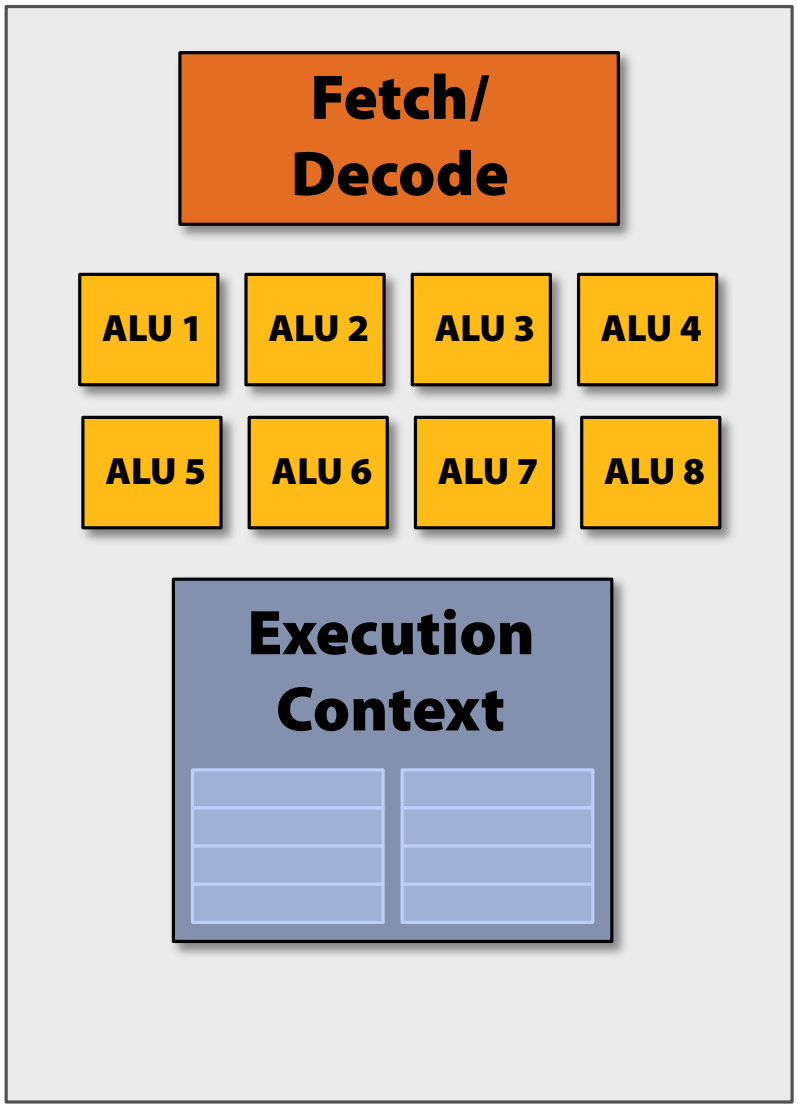
Credit: Kayvon Fatahalian (Stanford)

Idea #2

Amortize cost/complexity of managing an instruction stream across many ALUs

→ **SIMD**

Saving Yet More Space



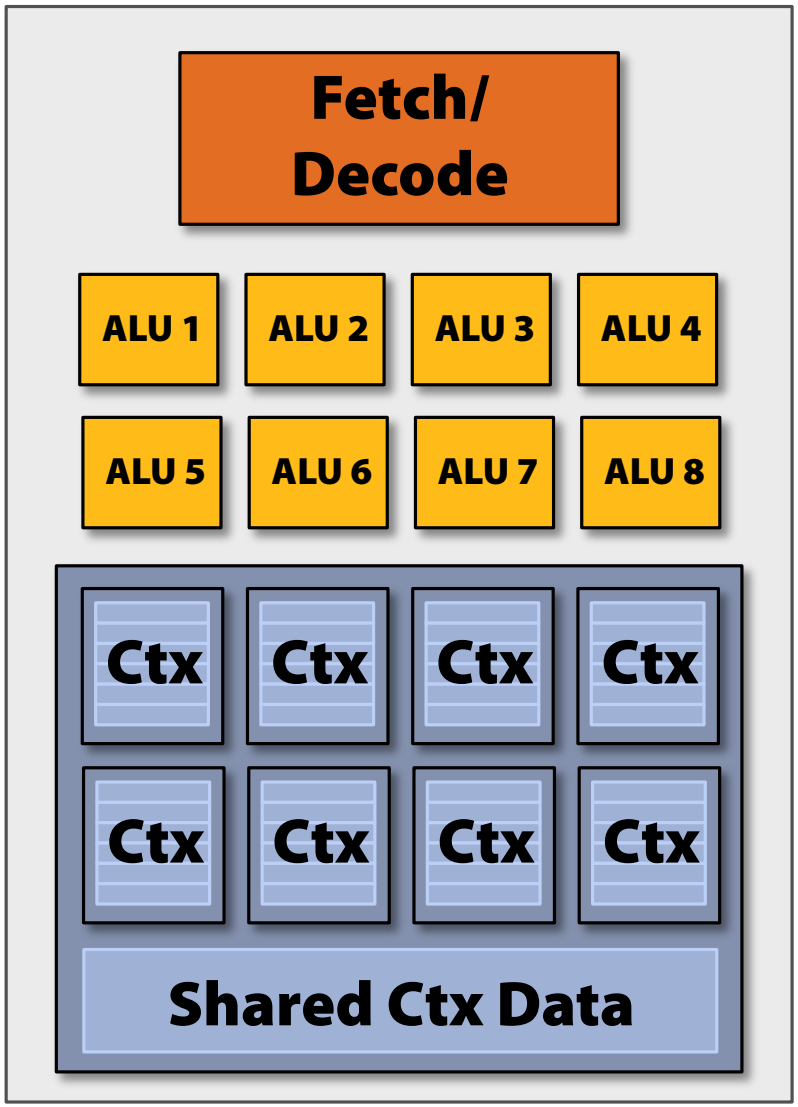
Credit: Kayvon Fatahalian (Stanford)

Idea #2

Amortize cost/complexity of managing an instruction stream across many ALUs

→ **SIMD**

Saving Yet More Space



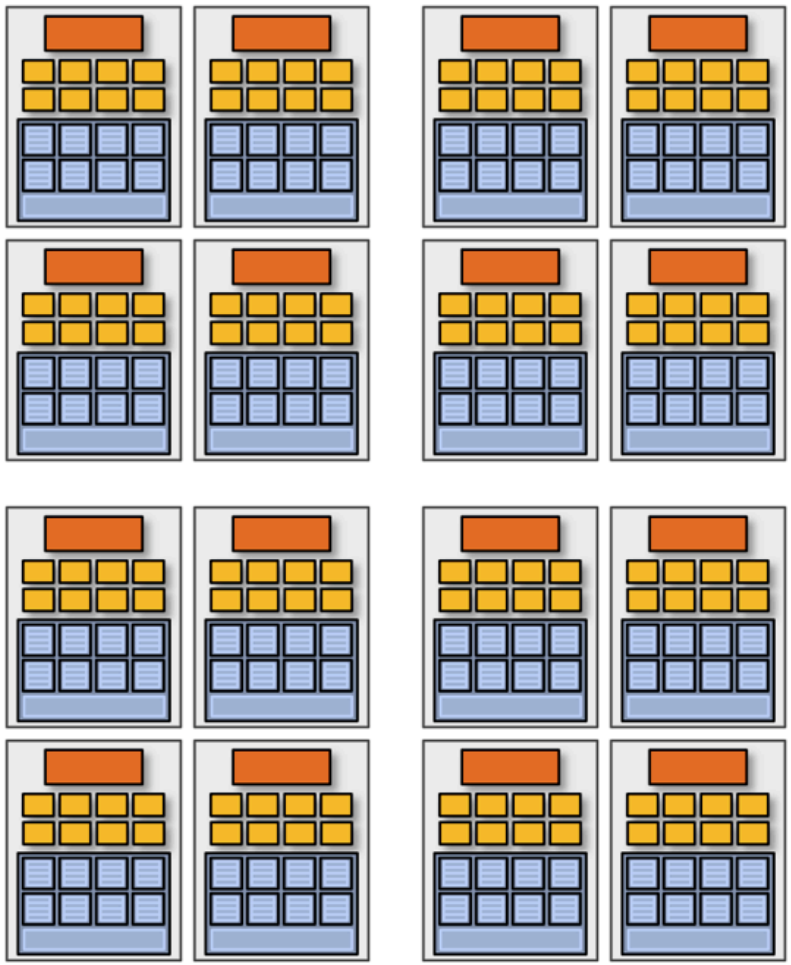
Credit: Kayvon Fatahalian (Stanford)

Idea #2

Amortize cost/complexity of managing an instruction stream across many ALUs

→ **SIMD**

Gratuitous Amounts of Parallelism!



Credit: Kayvon Fatahalian (Stanford)

Gratuitous Amounts of Parallelism!

Example:

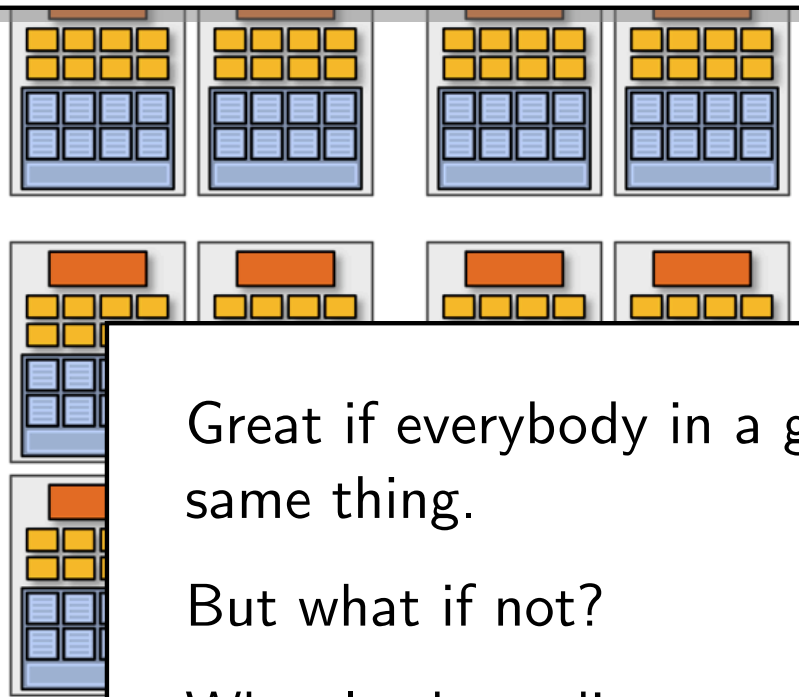
128 instruction streams in parallel
16 independent groups of 8 synchronized streams

Credit: Kayvon Fatahalian (Stanford)

Gratuitous Amounts of Parallelism!

Example:

128 instruction streams in parallel
16 independent groups of 8 synchronized streams



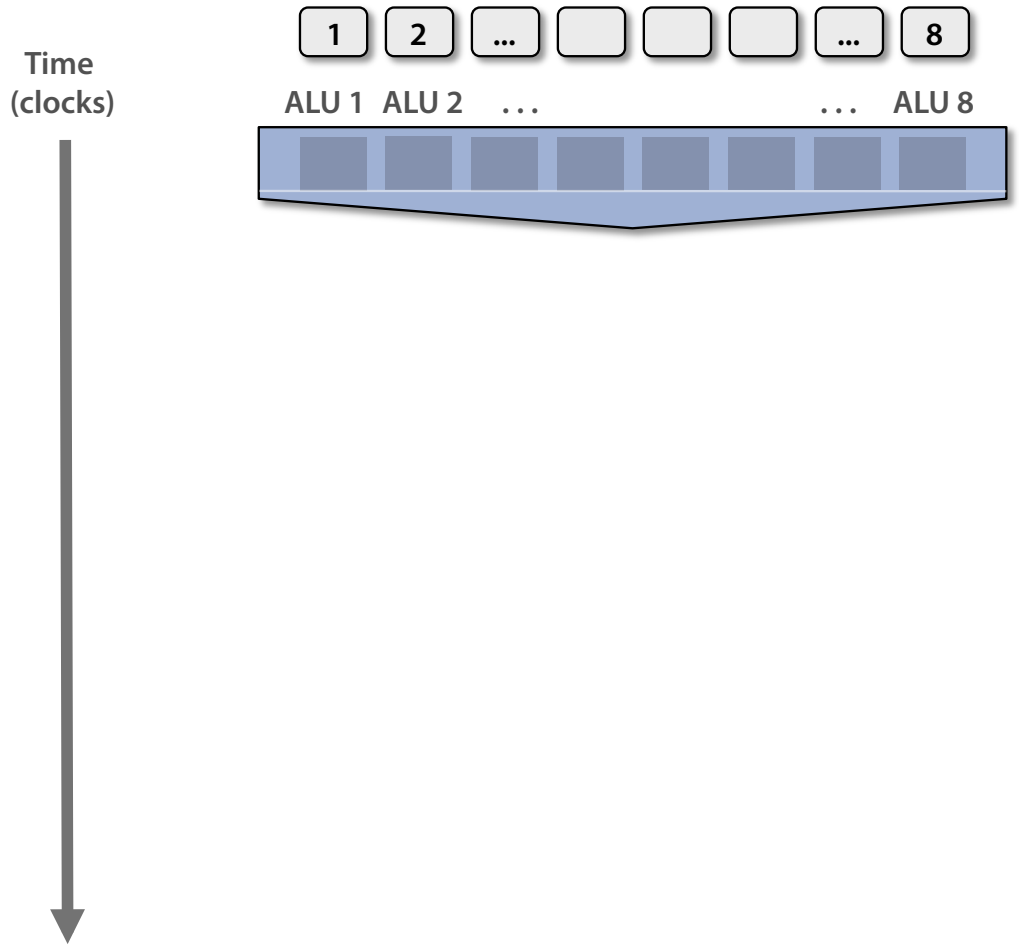
Great if everybody in a group does the same thing.

But what if not?

What leads to divergent instruction streams?

Credit: Kayvon Fatahalian (

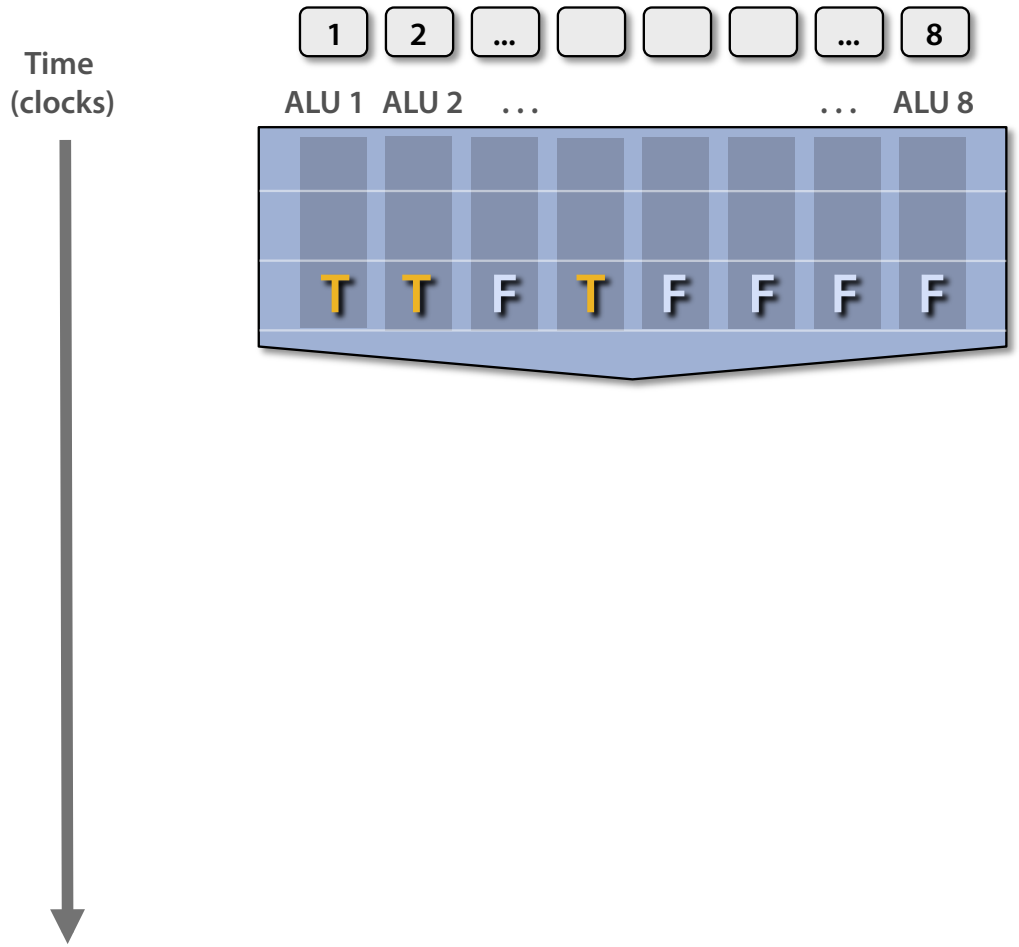
Branches




```
<unconditional
shader code>

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}

<resume unconditional
shader code>
```

Credit: Kayvon Fatahalian (Stanford)

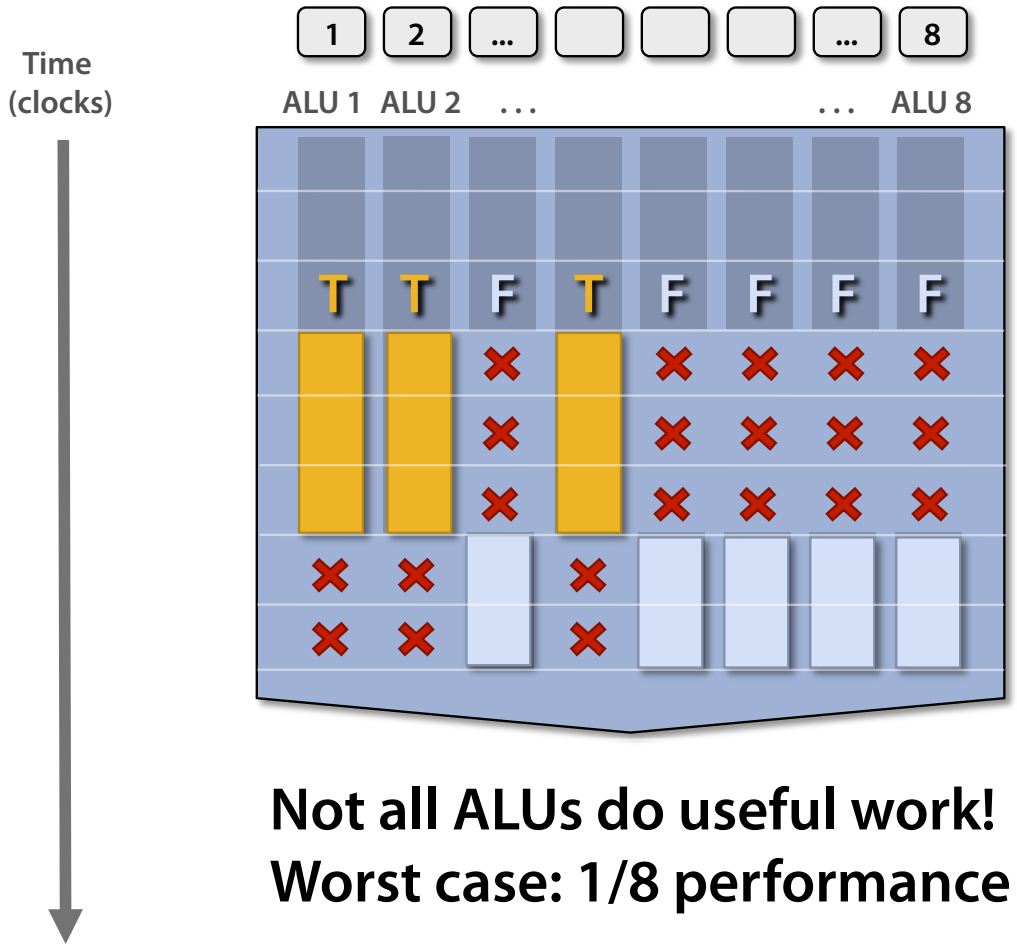
Branches




```
<unconditional
shader code>

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}

<resume unconditional
shader code>
```

Credit: Kayvon Fatahalian (Stanford)

Branches

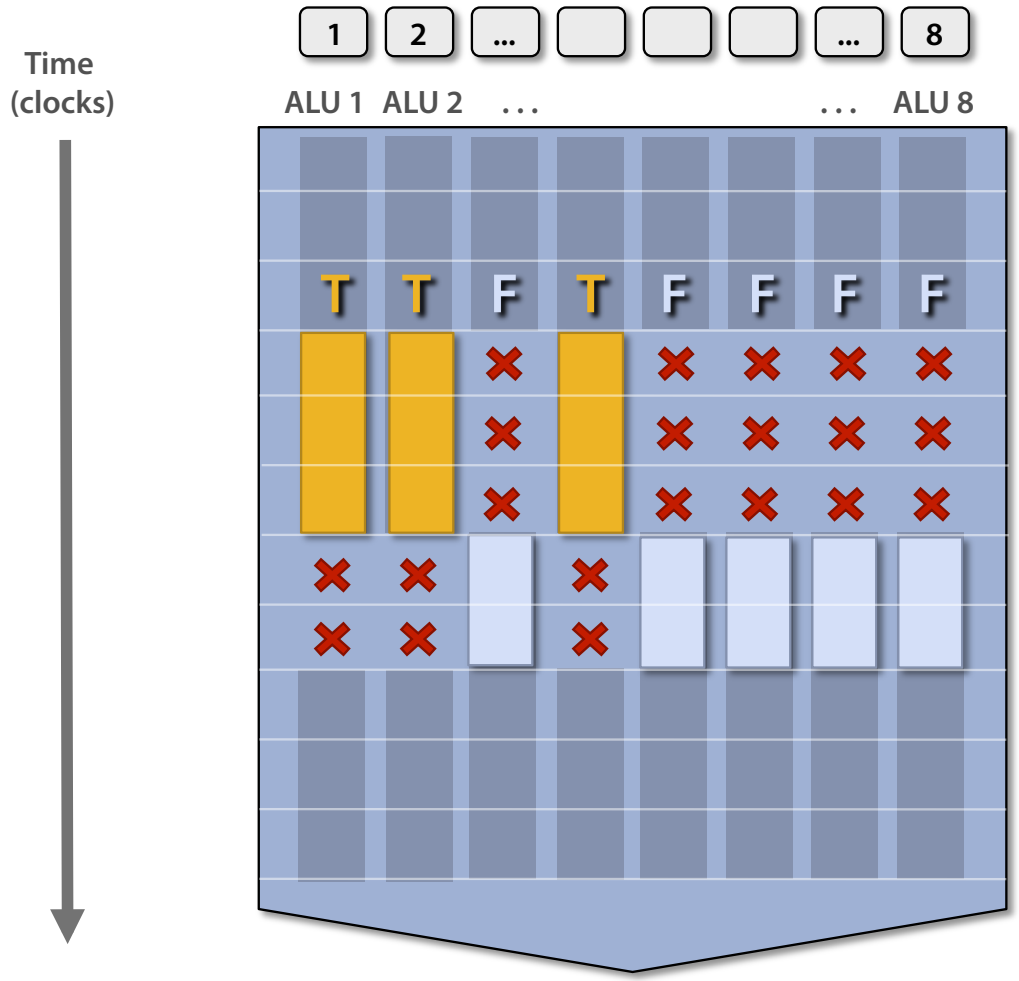


Not all ALUs do useful work!
Worst case: 1/8 performance

```
<unconditional
shader code>

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}

<resume unconditional
shader code>
```

Credit: Kayvon Fatahalian (Stanford)

Branches



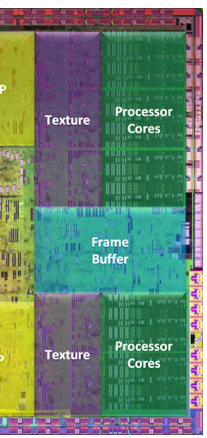

```
<unconditional
shader code>

if (x > 0) {
    y = pow(x, exp);
    y *= Ks;
    refl = y + Ka;
} else {
    x = 0;
    refl = Ka;
}

<resume unconditional
shader code>
```

Credit: Kayvon Fatahalian (Stanford)

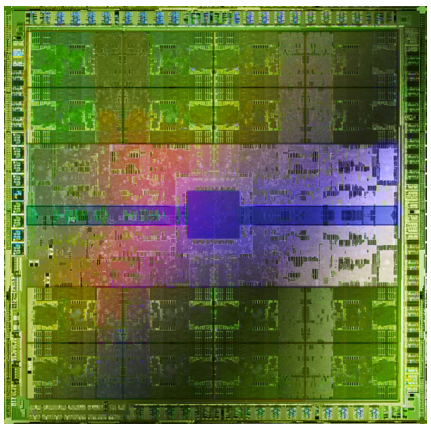
Recent Processor Architecture

- Commodity chips
- "Infinitely" many cores
- "Infinite" vector width
- Must hide memory latency ($\rightarrow$ ILP, SMT)

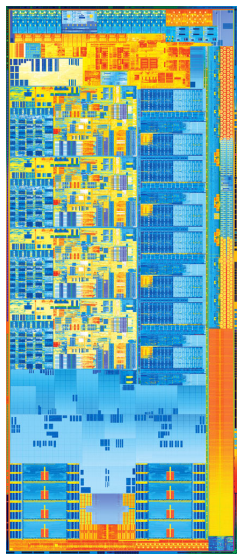
- Compute bandwidth $\gg$ Memory bandwidth
- Bandwidth only achievable by *homogeneity*



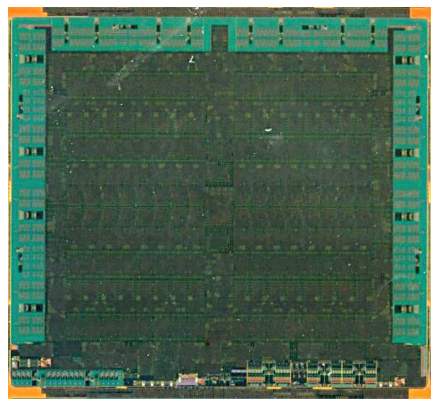
T200
08)



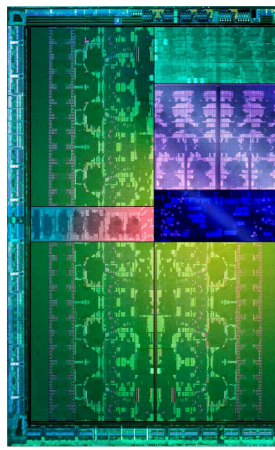
Nv Fermi
(2010)



Intel IVB
(2012)



AMD Tahiti
(2012)



Nv GK1
(2012

Outline

What is OpenCL?

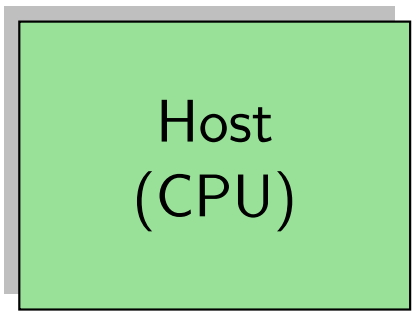
OpenCL (Open Computing Language) is an open, royalty-free standard for general purpose parallel programming across CPUs, GPUs and other processors. [OpenCL 1.1 spec]

- Device-neutral (Nv GPU, AMD GPU, Intel/AMD CPU)
- Vendor-neutral
- Comes with 'JIT' compilation

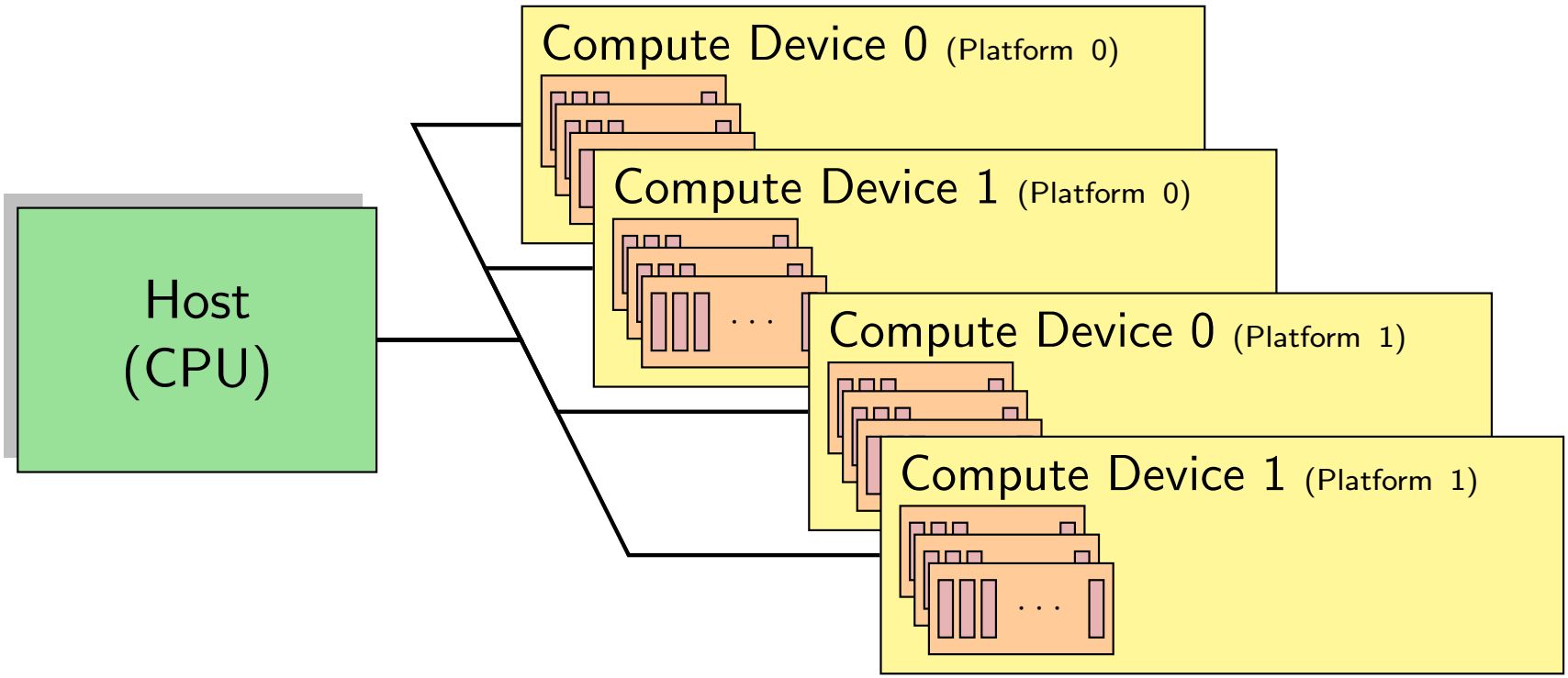
Defines:

- Host-side programming interface (library)
- Device-side programming language (!)

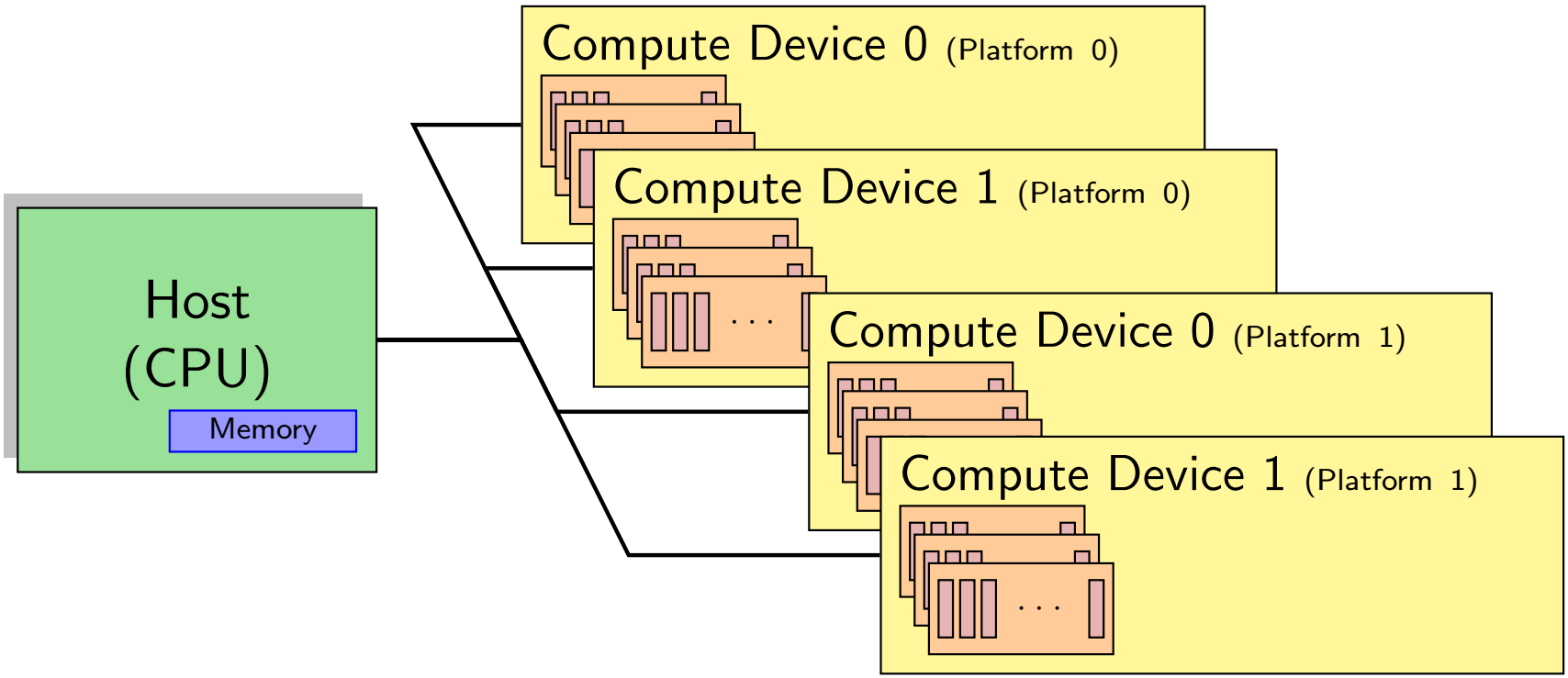
OpenCL: Vocabulary



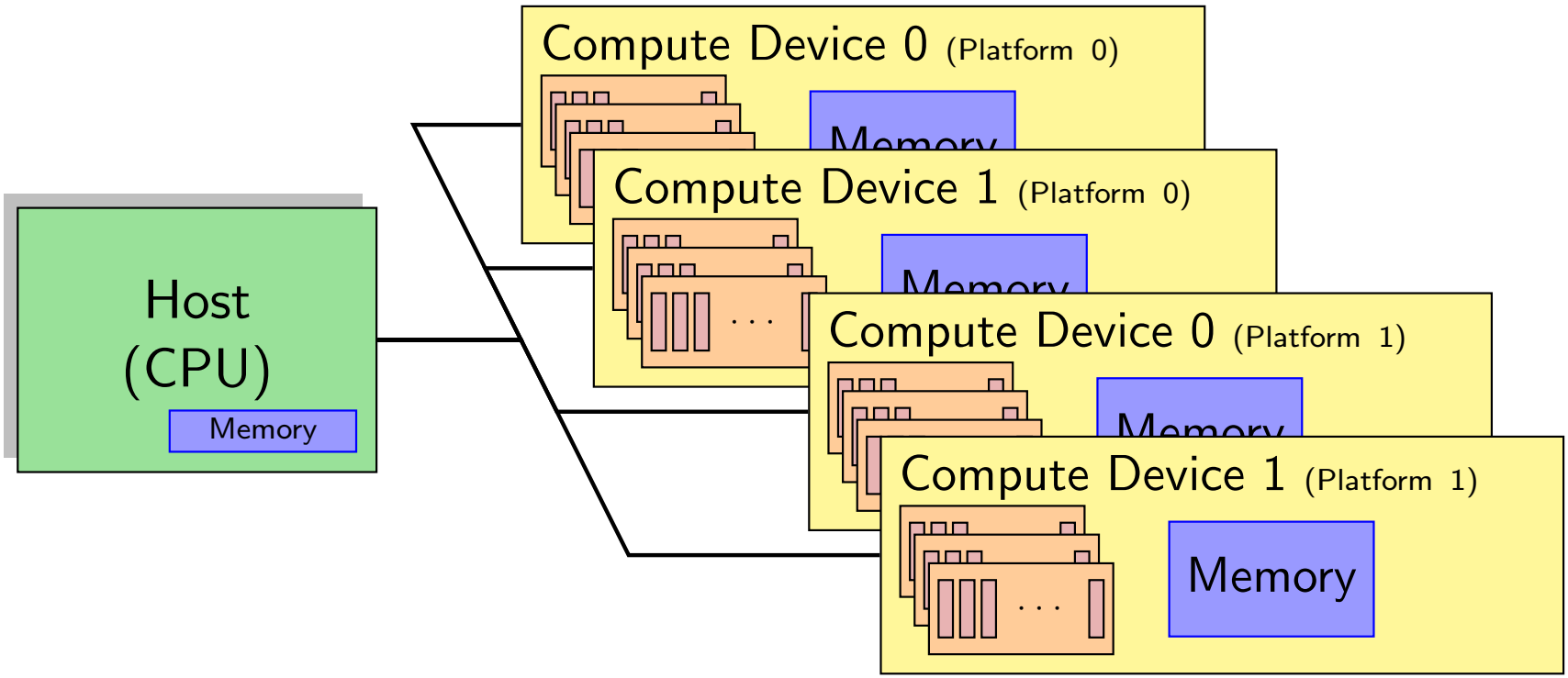
OpenCL: Vocabulary



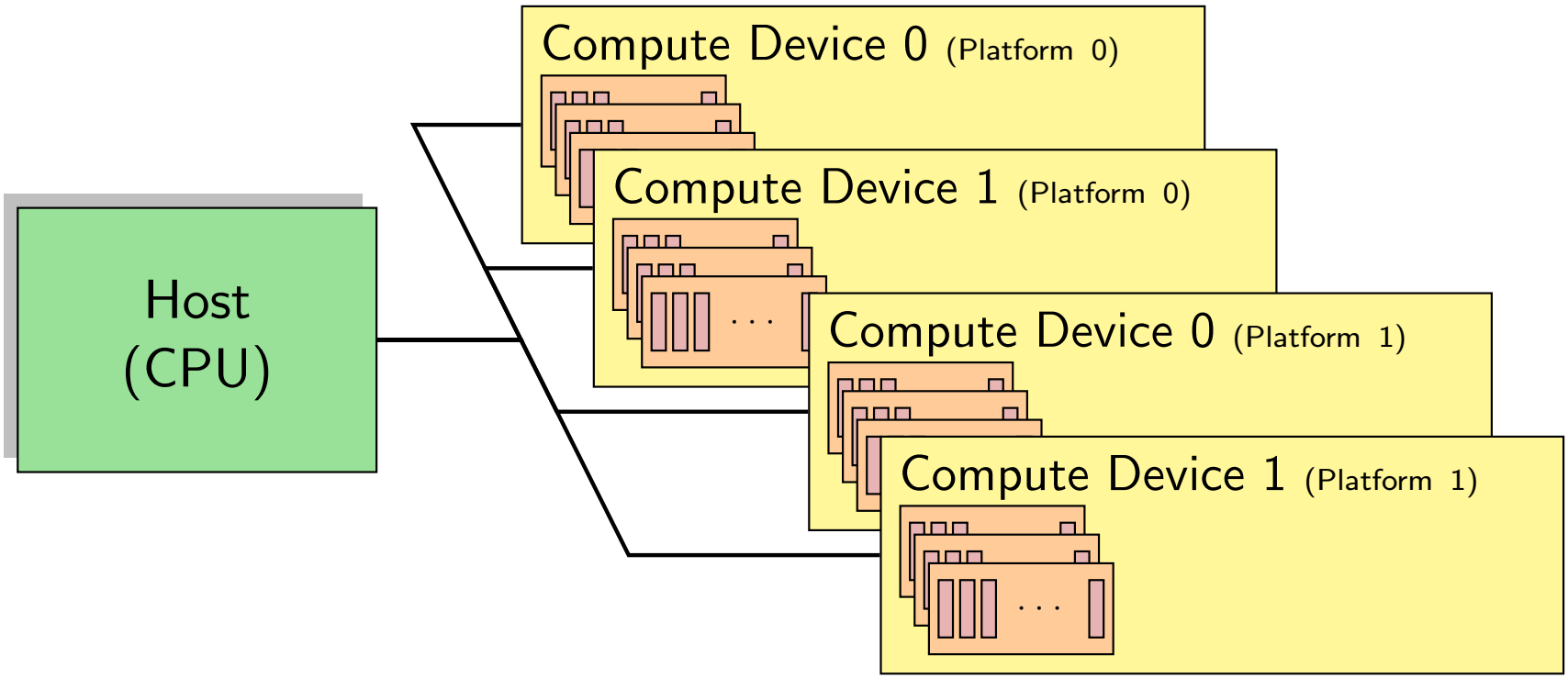
OpenCL: Vocabulary



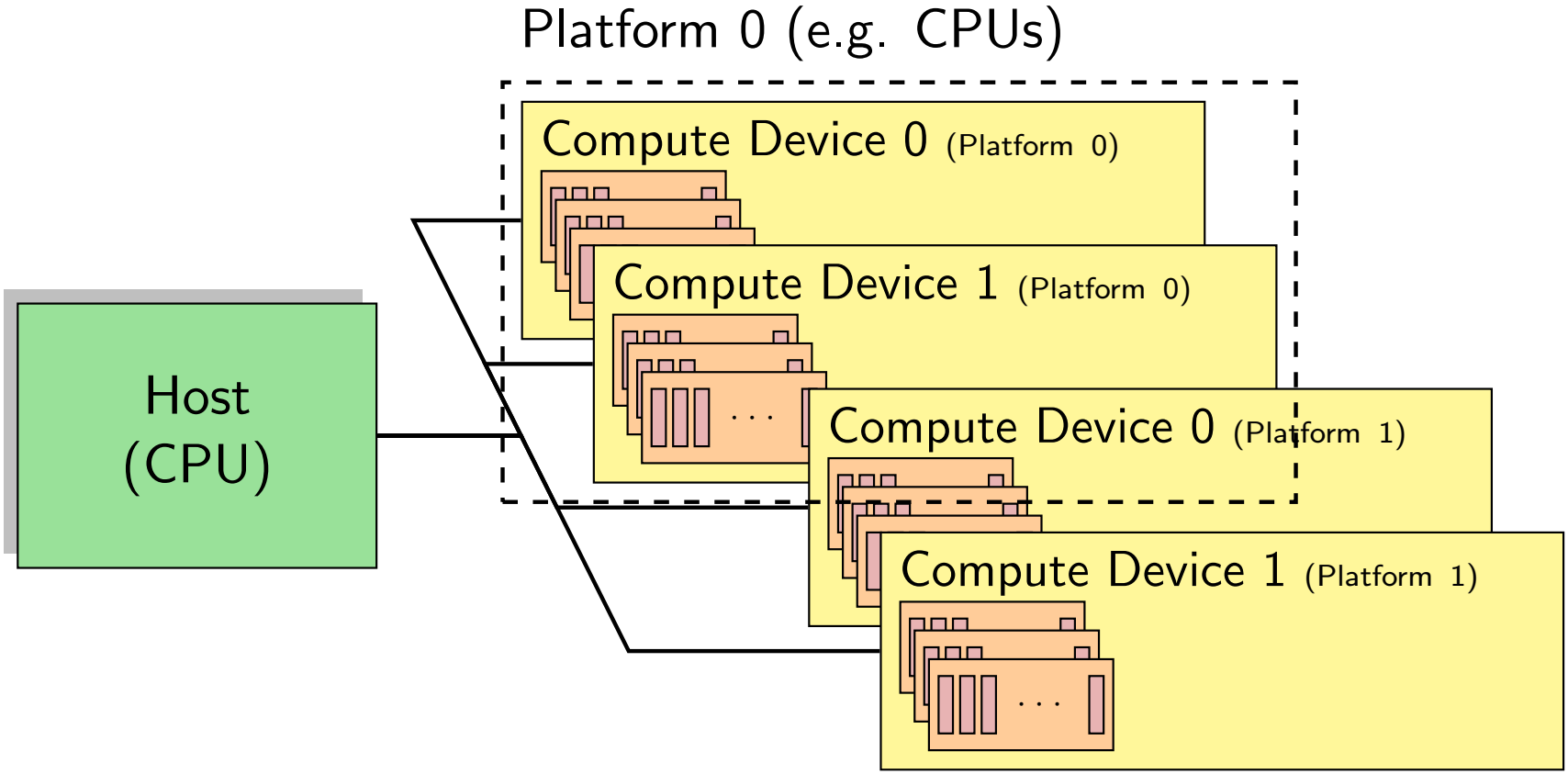
OpenCL: Vocabulary



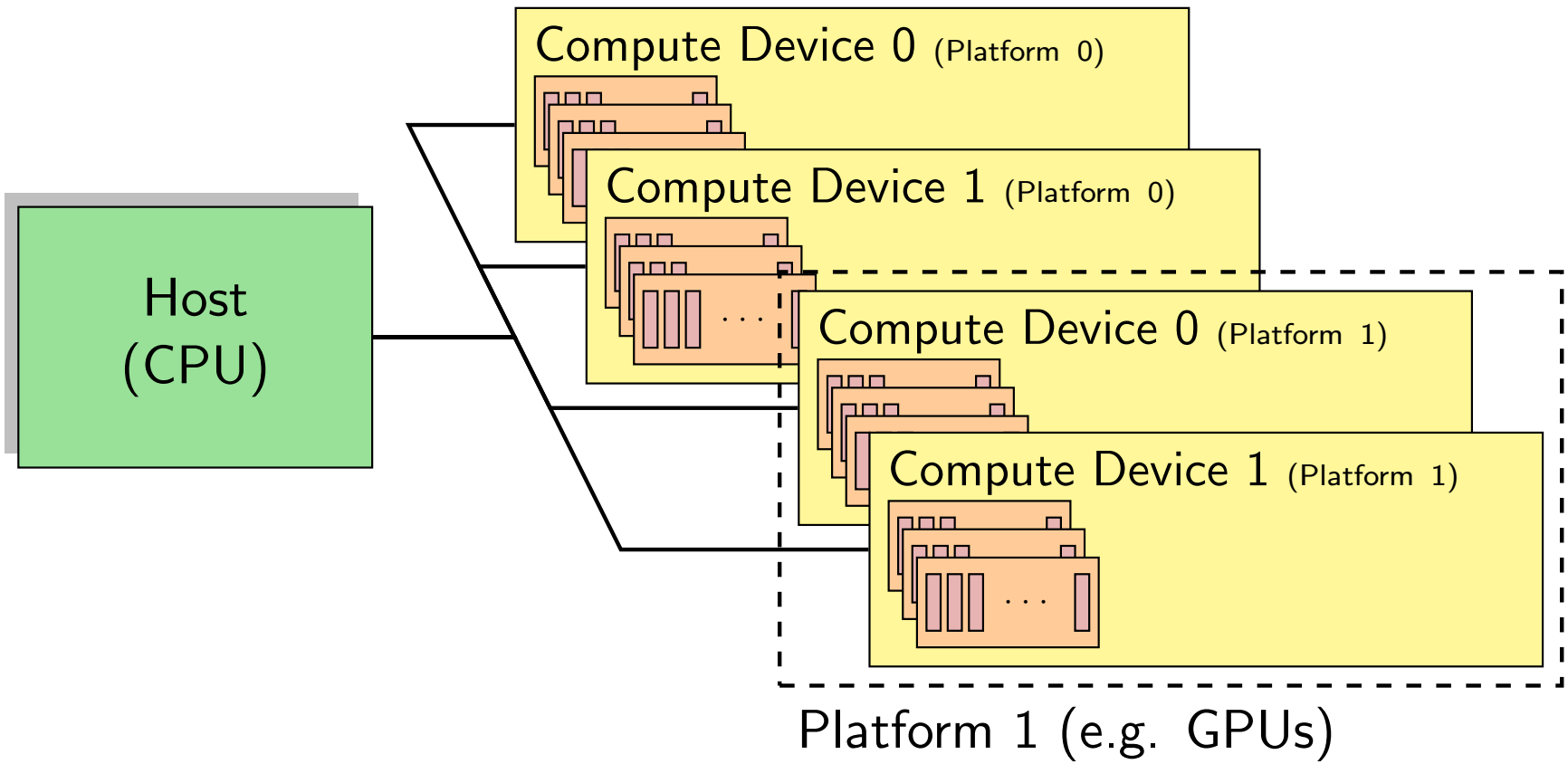
OpenCL: Vocabulary



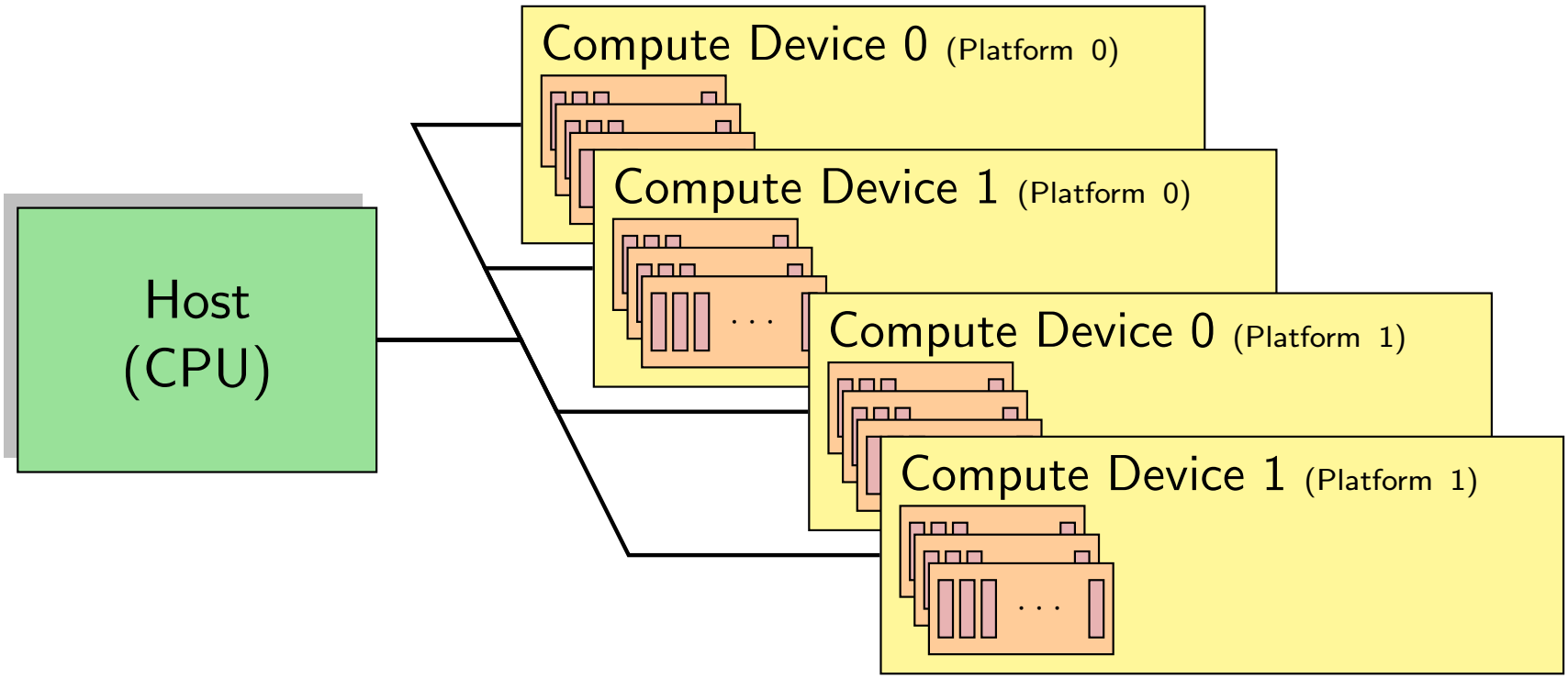
OpenCL: Vocabulary



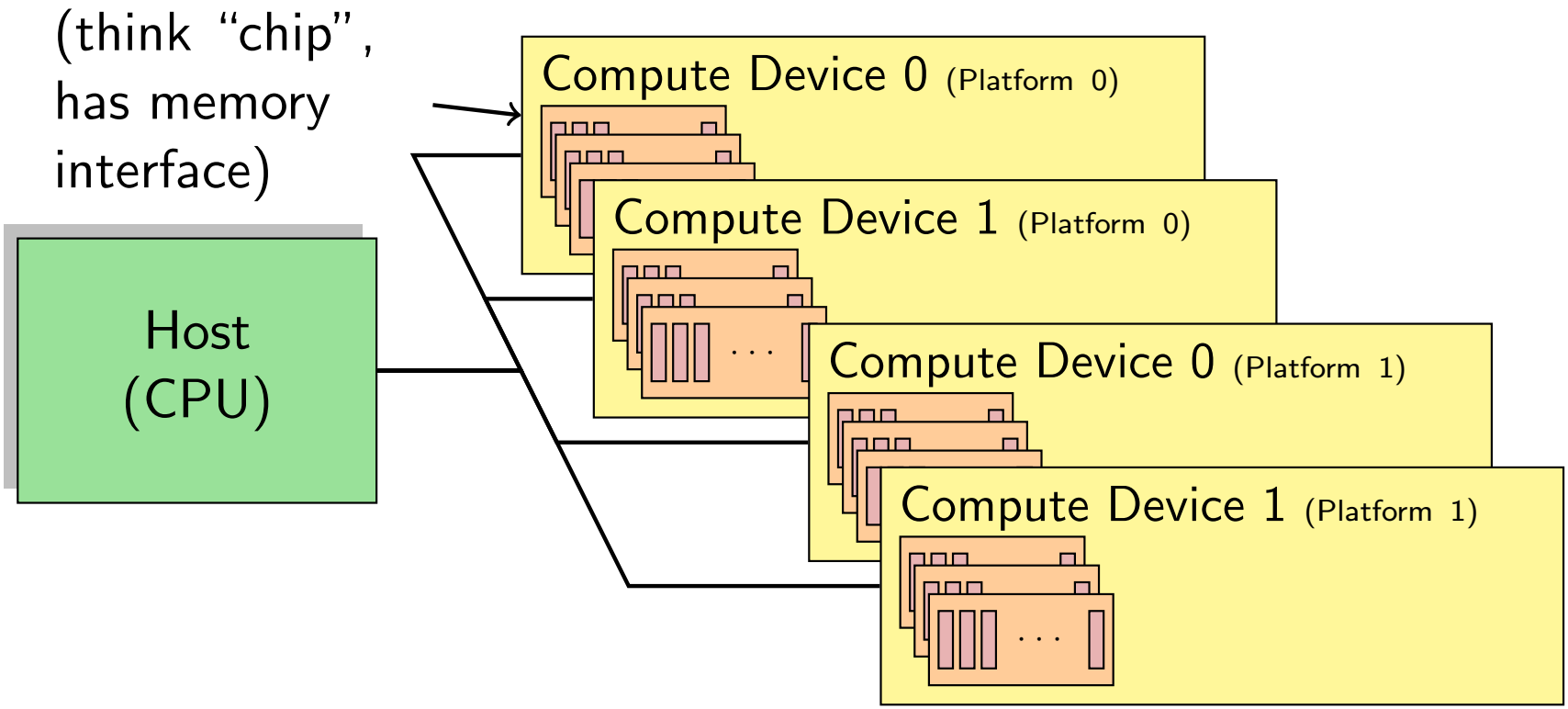
OpenCL: Vocabulary



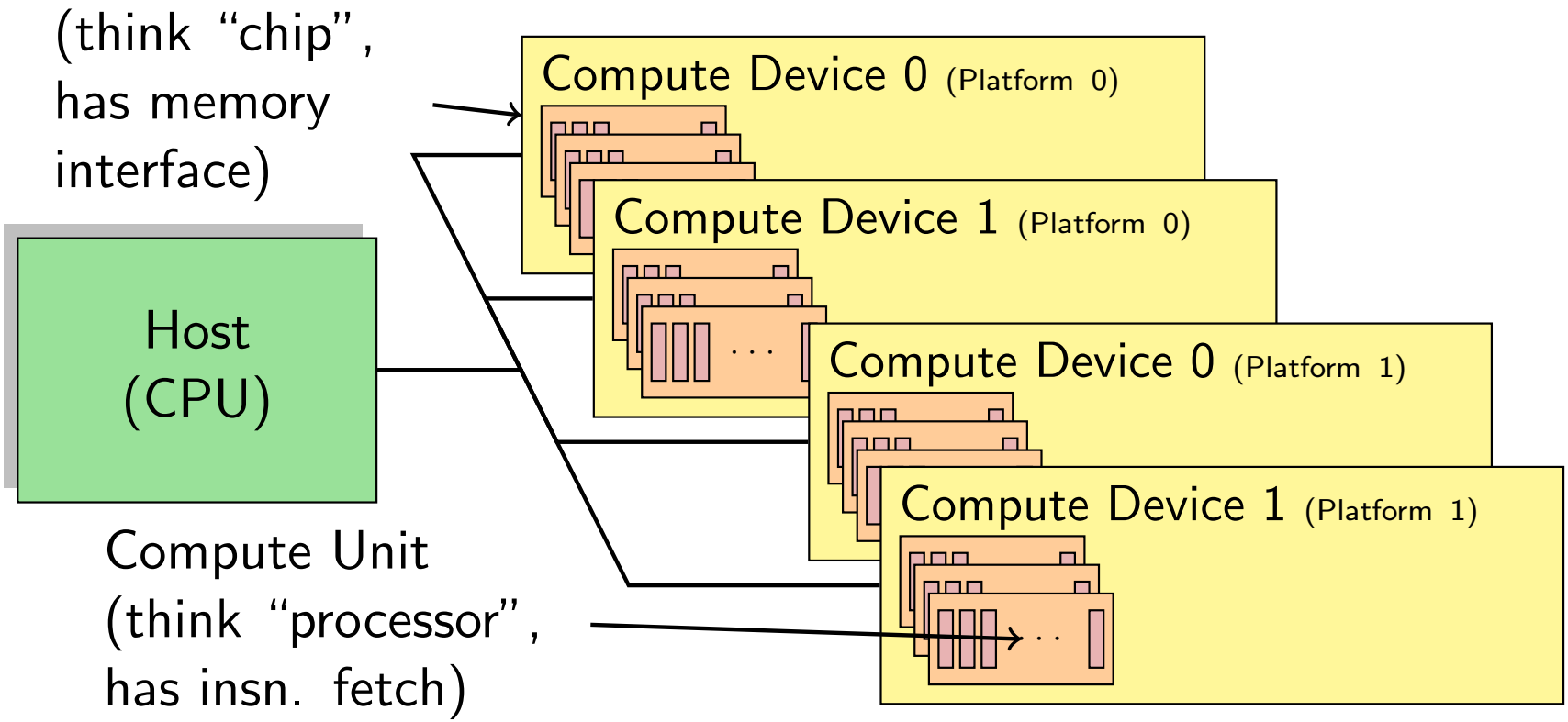
OpenCL: Vocabulary



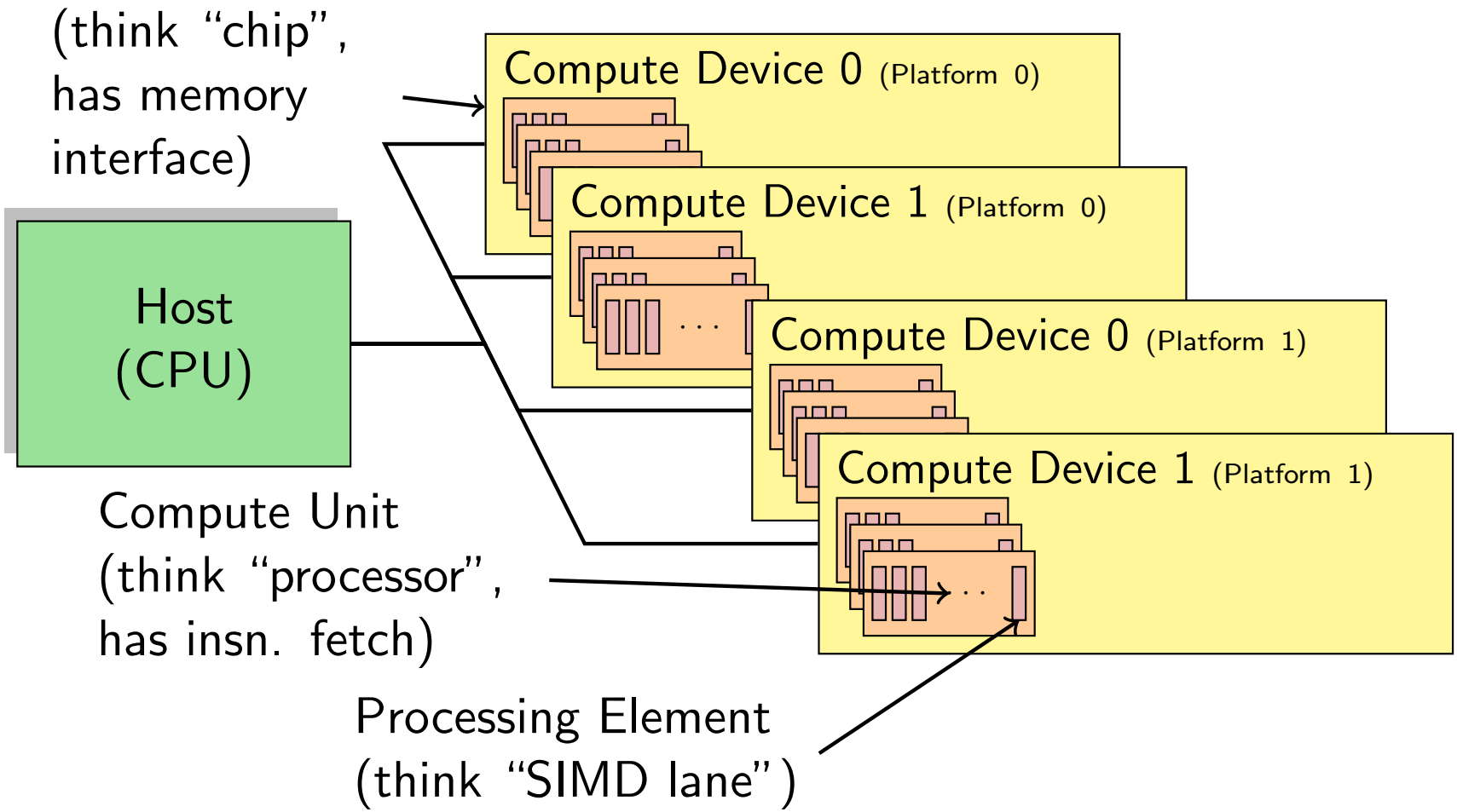
OpenCL: Vocabulary



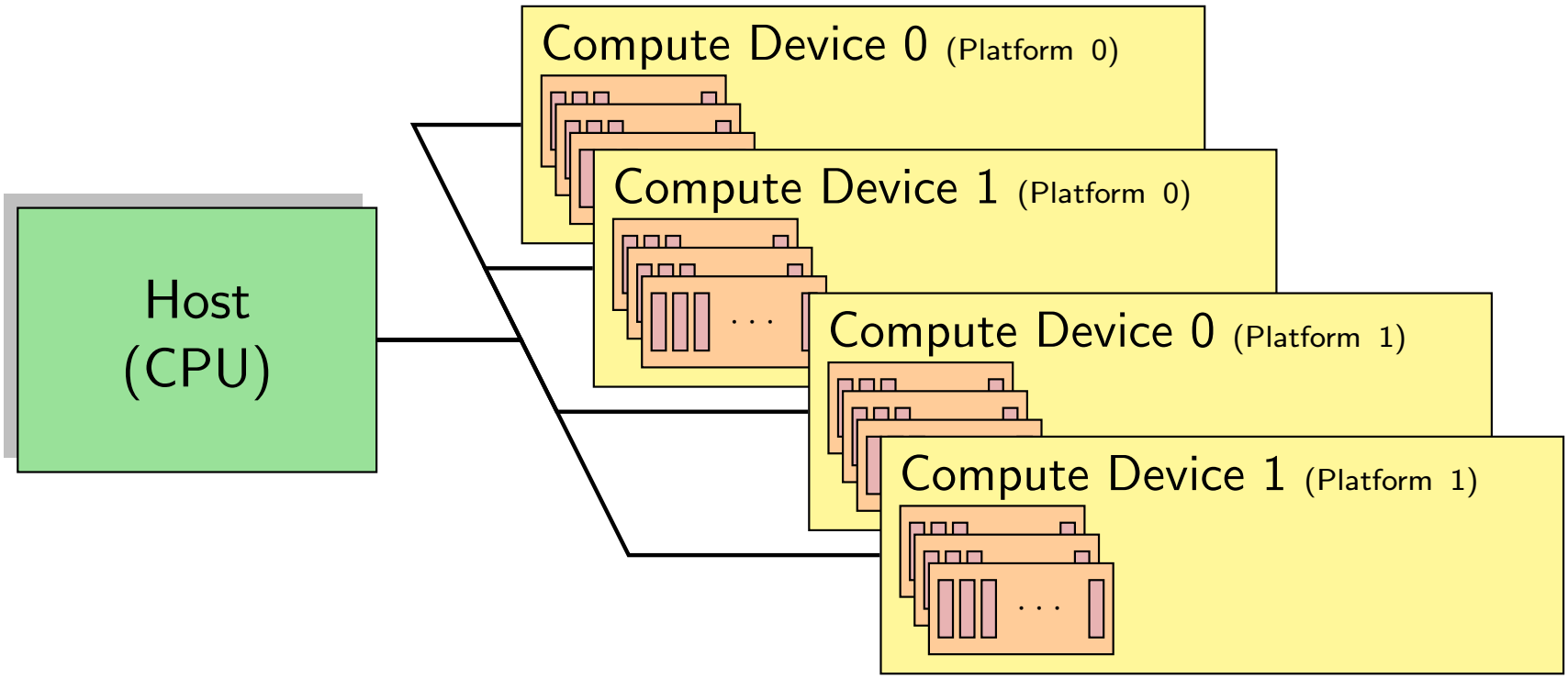
OpenCL: Vocabulary



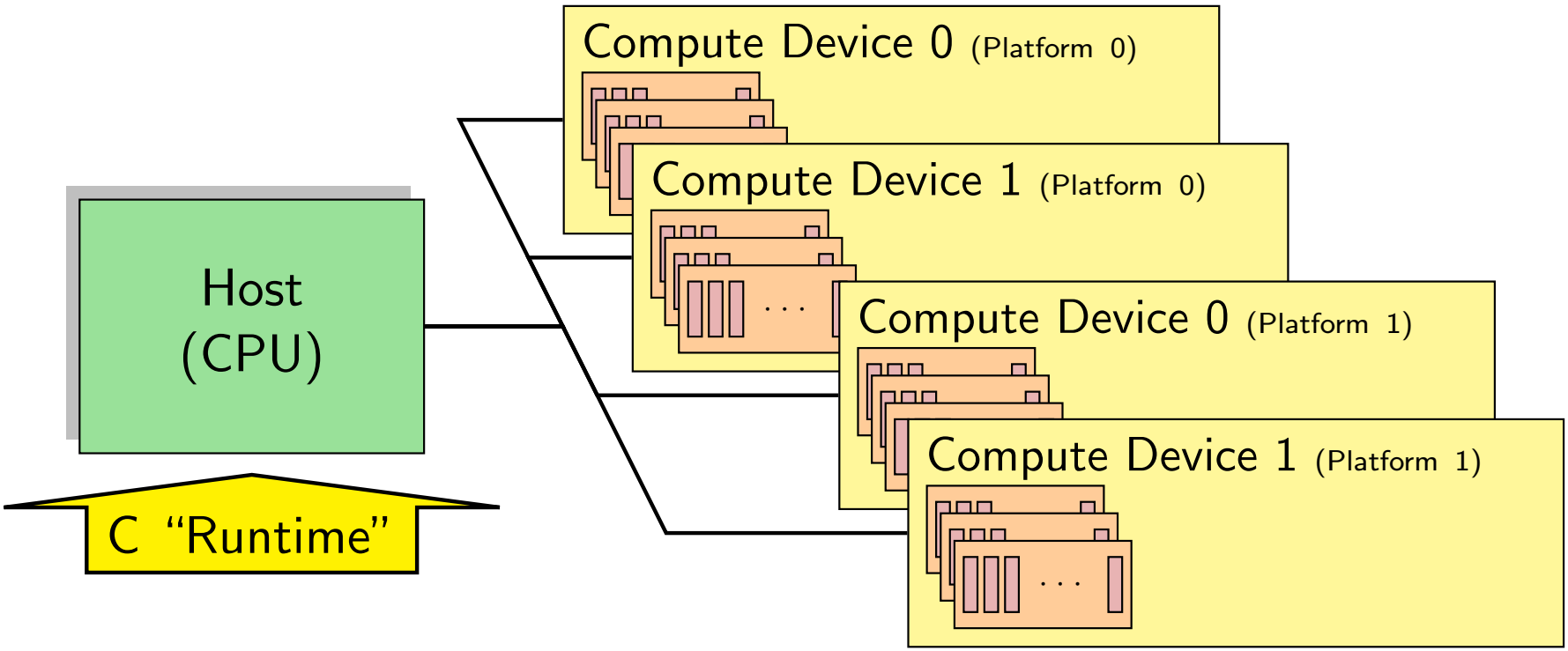
OpenCL: Vocabulary



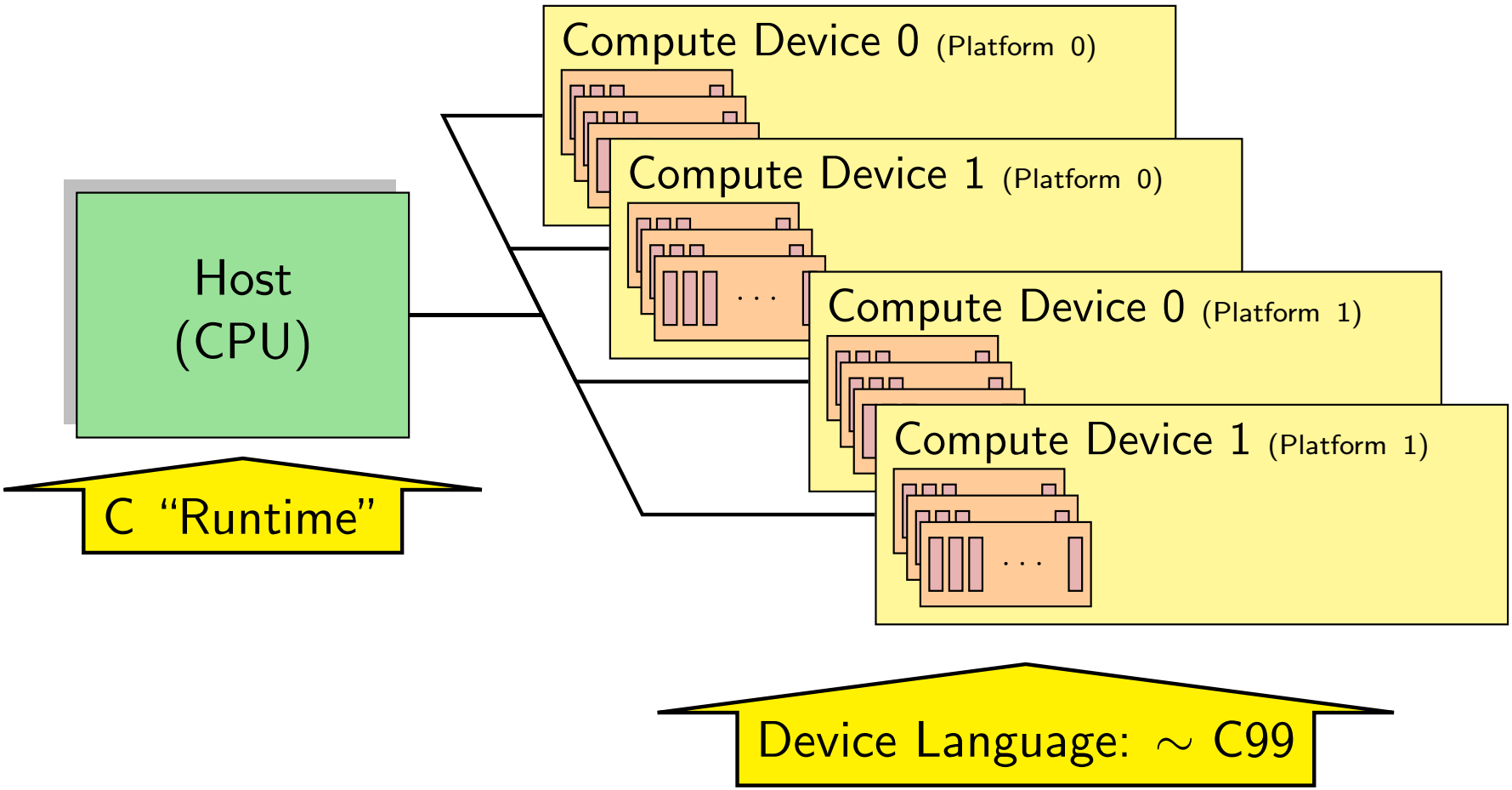
OpenCL: Vocabulary



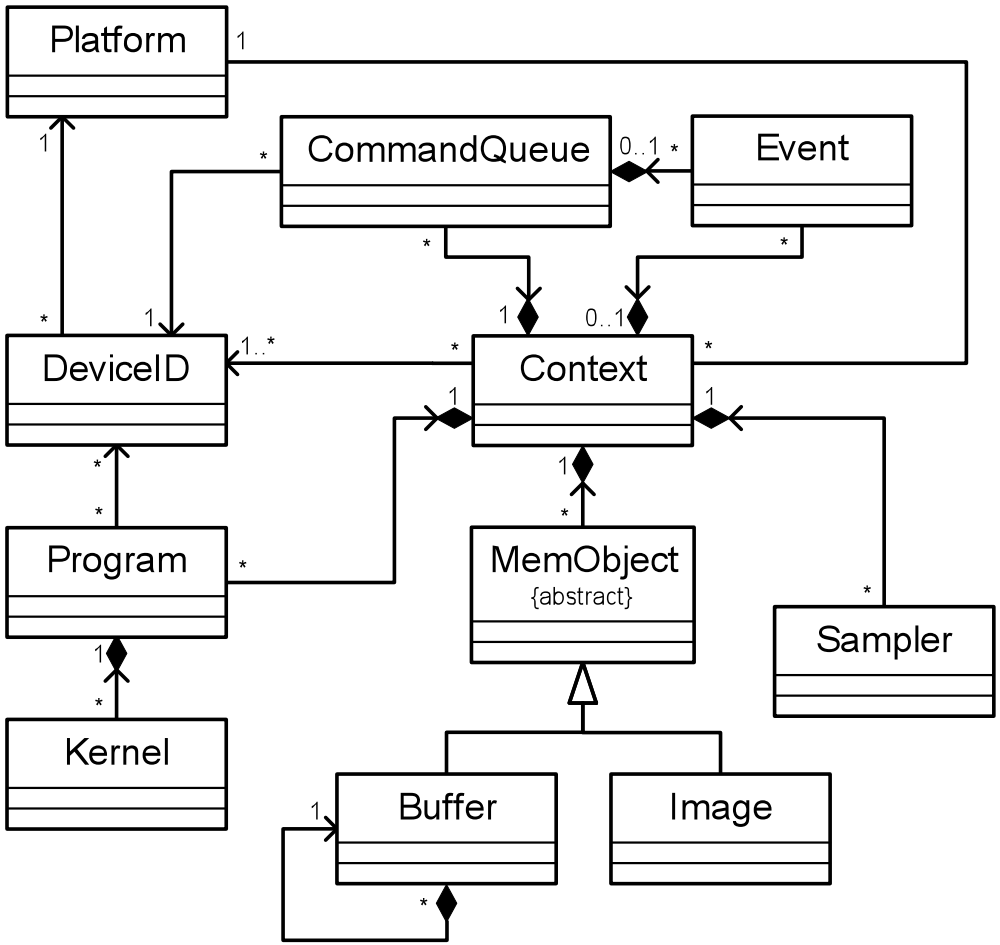
OpenCL: Vocabulary



OpenCL: Vocabulary

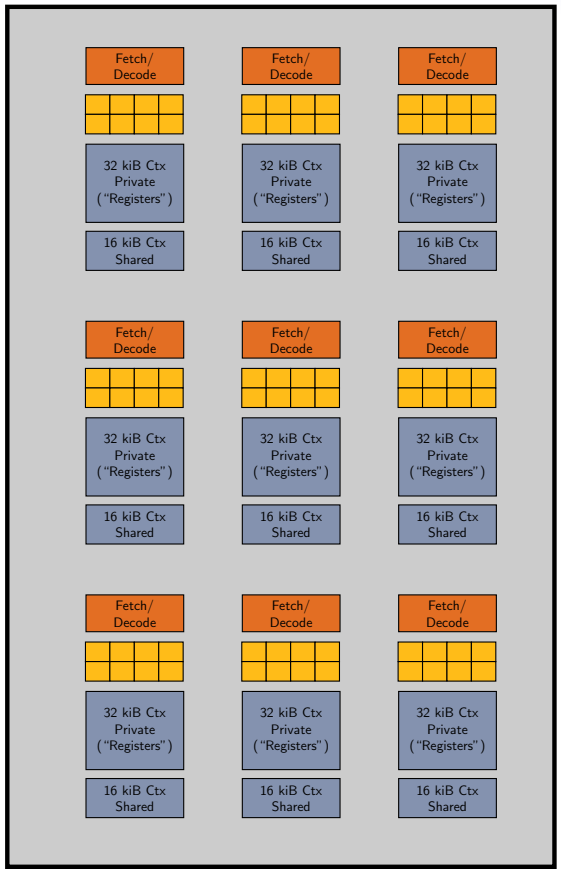


OpenCL Object Diagram

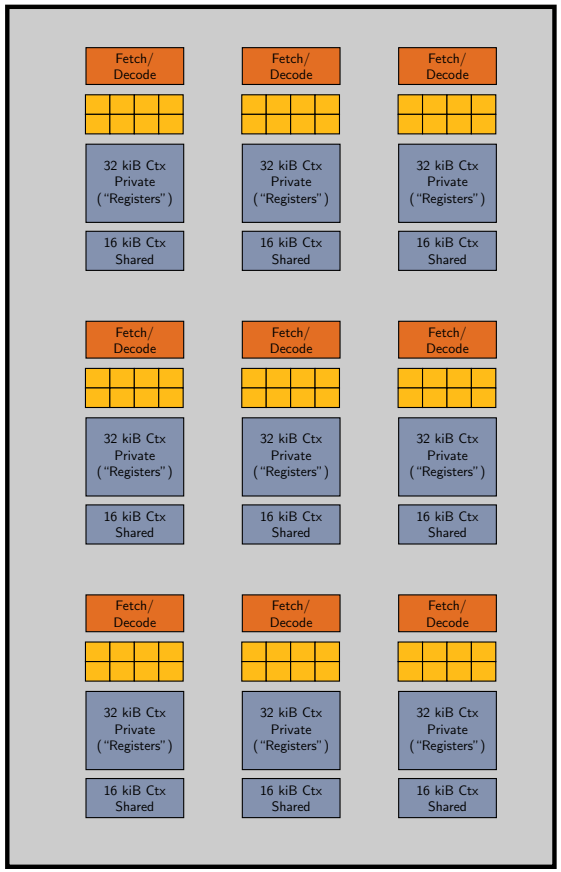


Credit: Khronos Group

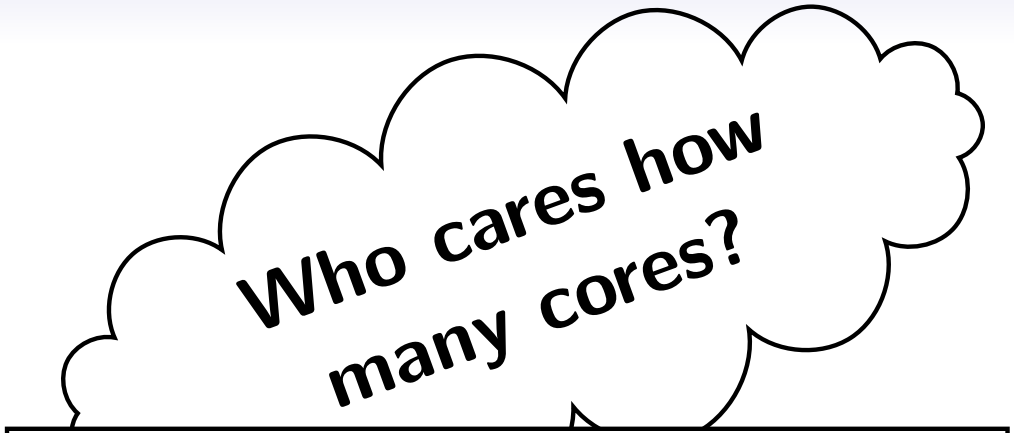
Connection: Hardware $\leftrightarrow$ Programming Model



Connection: Hardware $\leftrightarrow$ Programming Model

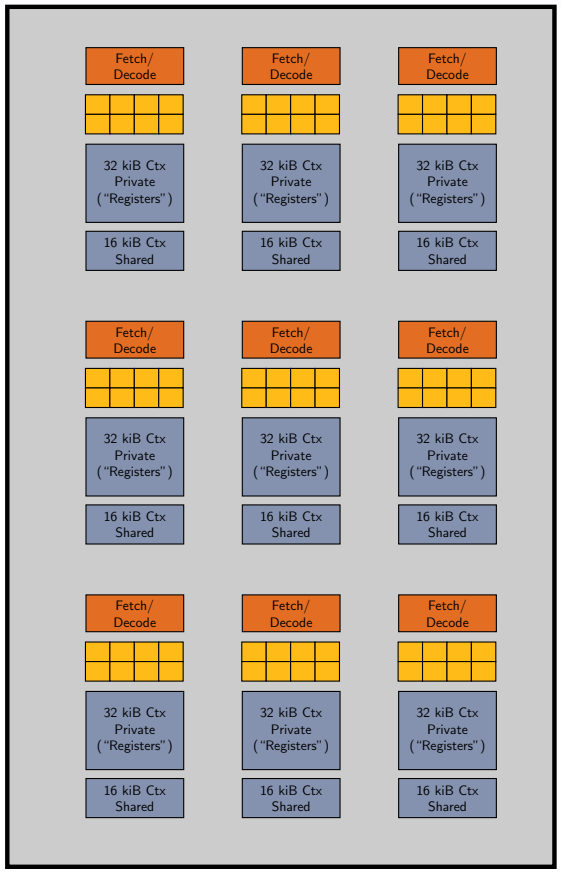


Connection: Hardware $\leftrightarrow$ Programming Model

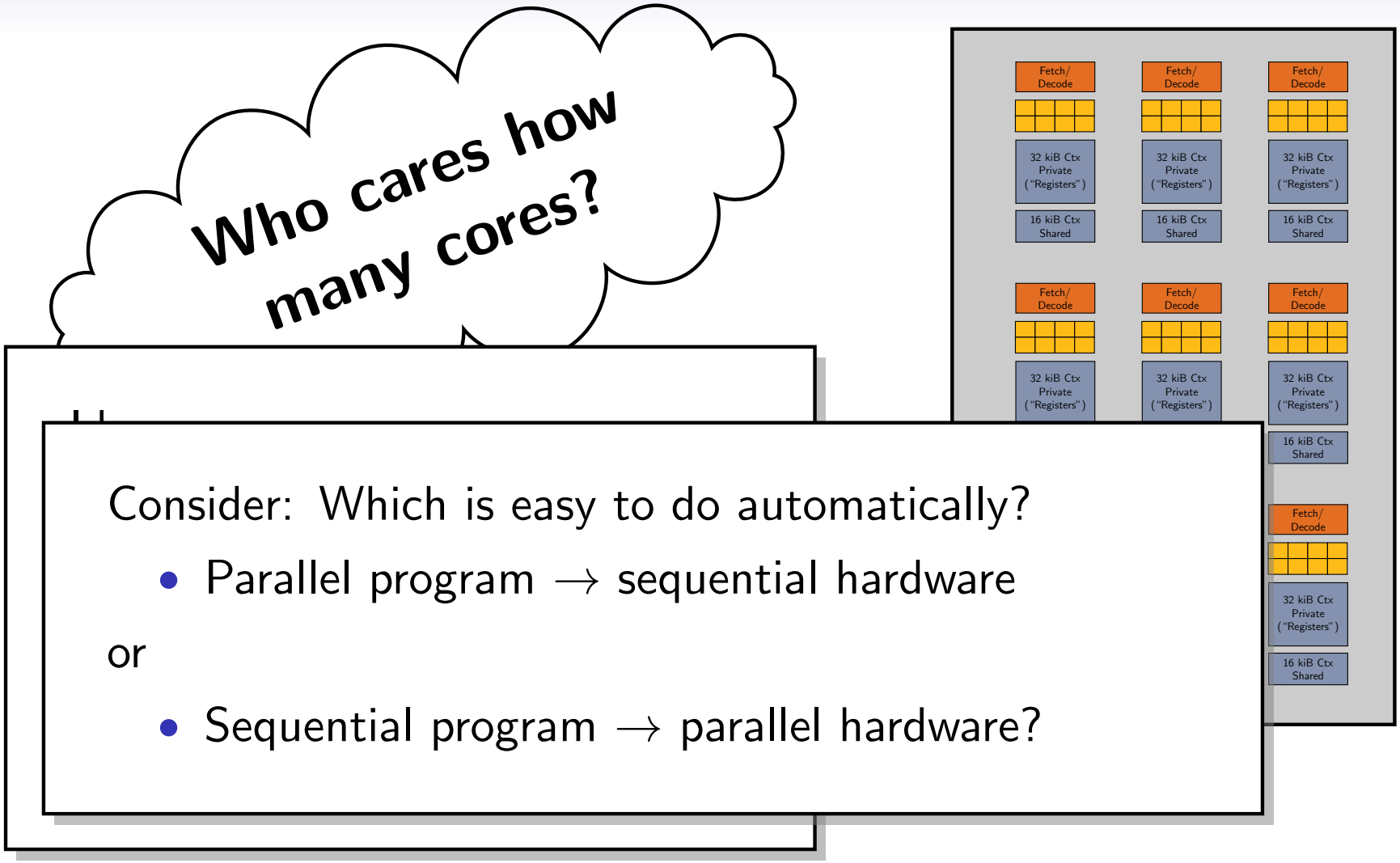


Idea:

- Program as if there were "infinitely" many cores
- Program as if there were "infinitely" many ALUs per core



Connection: Hardware $\leftrightarrow$ Programming Model



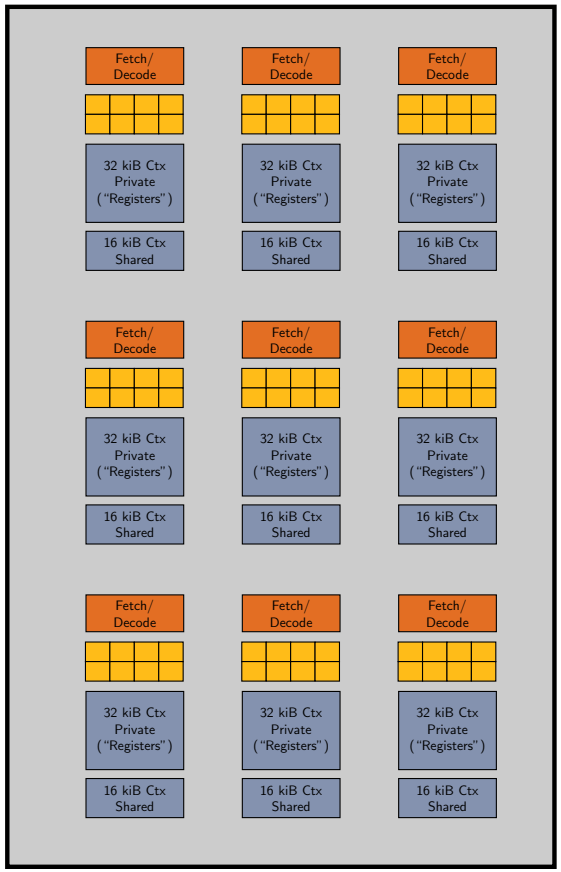
Consider: Which is easy to do automatically?

- Parallel program $\rightarrow$ sequential hardware

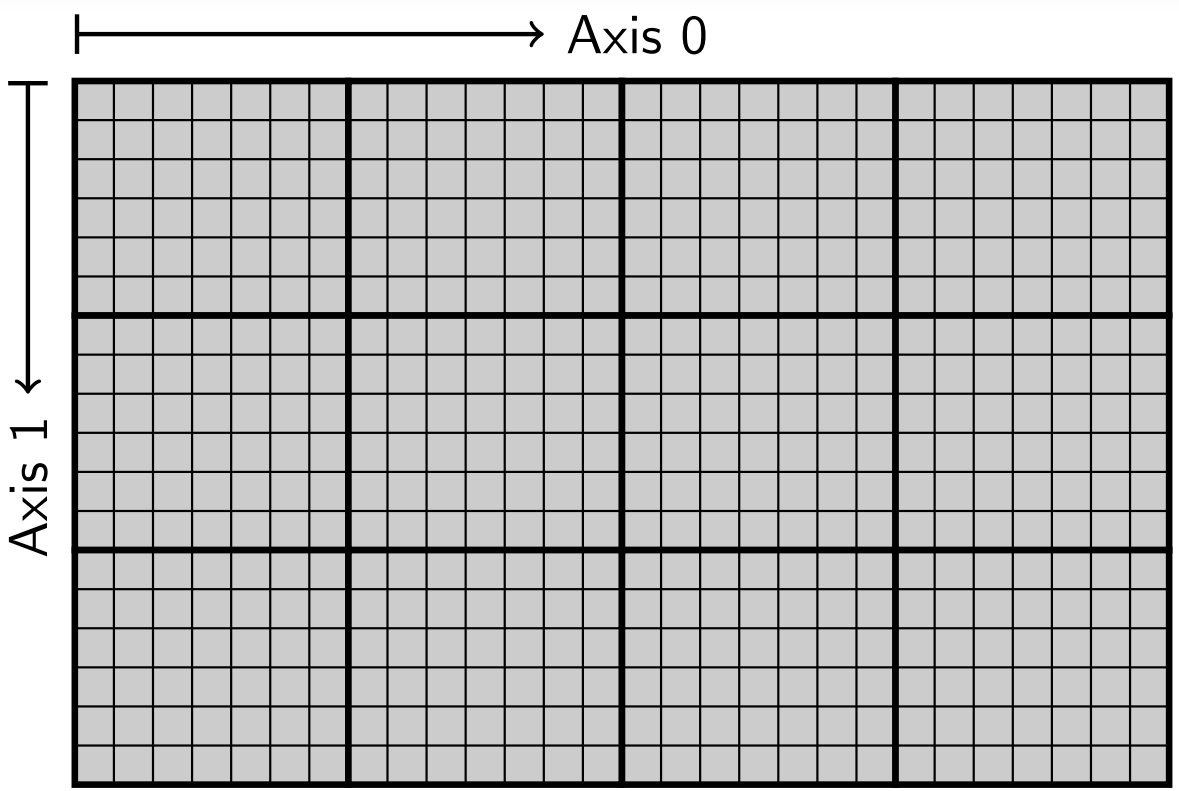
or

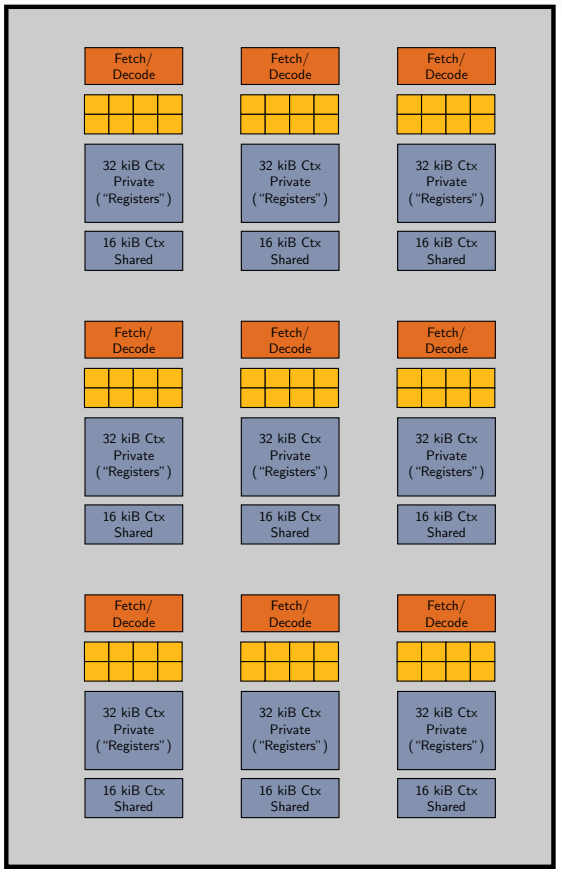
- Sequential program $\rightarrow$ parallel hardware?

Connection: Hardware $\leftrightarrow$ Programming Model

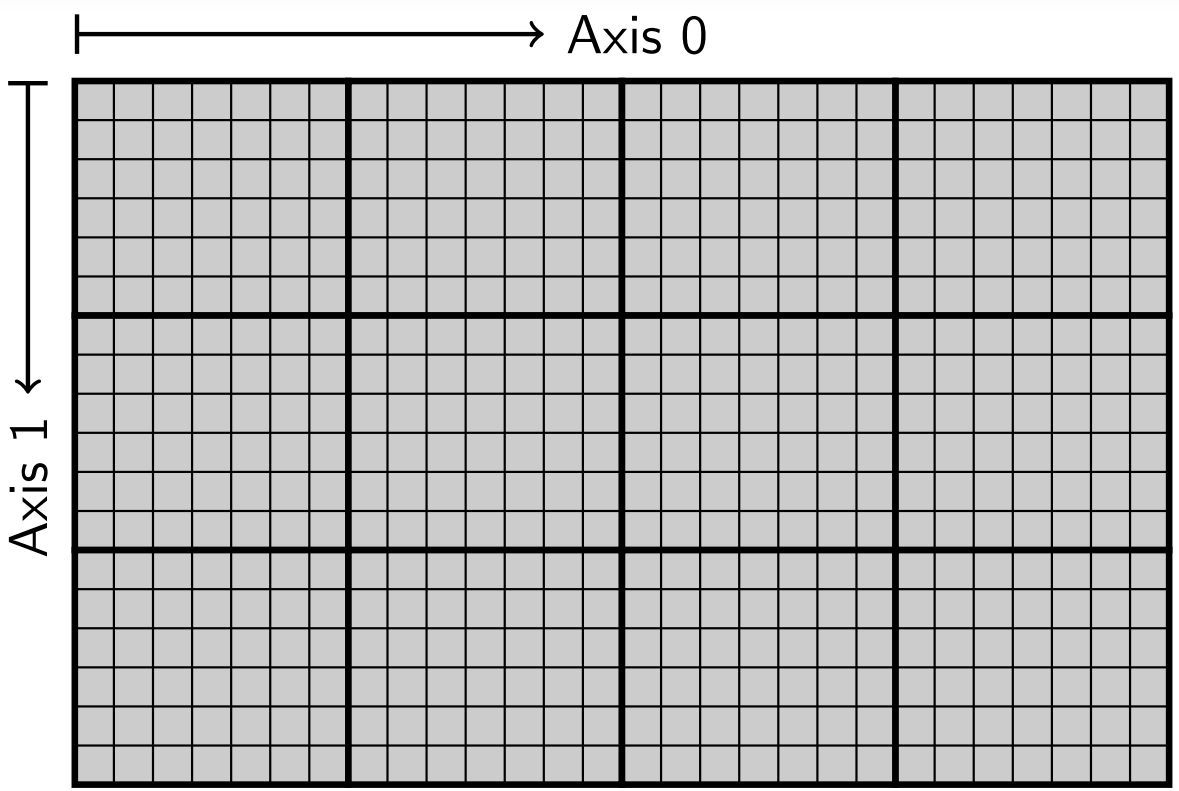


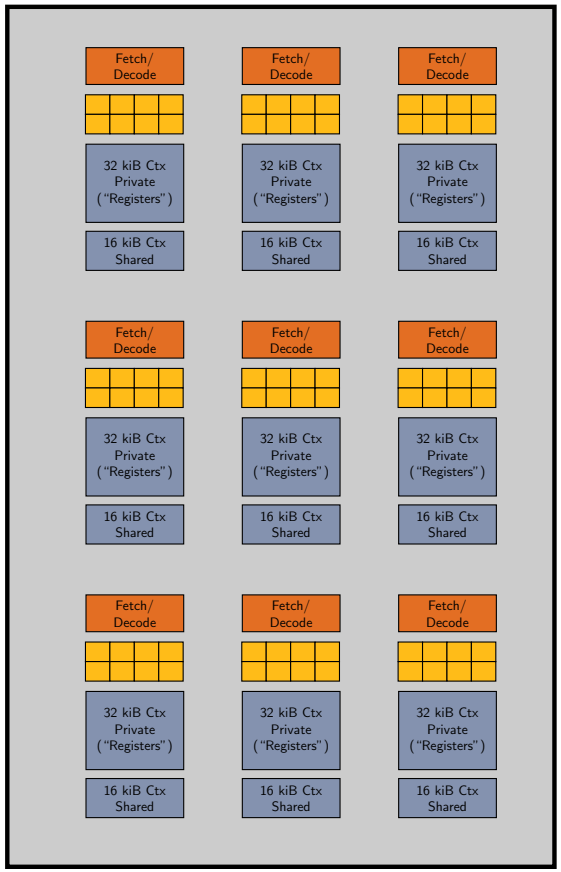
Connection: Hardware $\leftrightarrow$ Programming Model





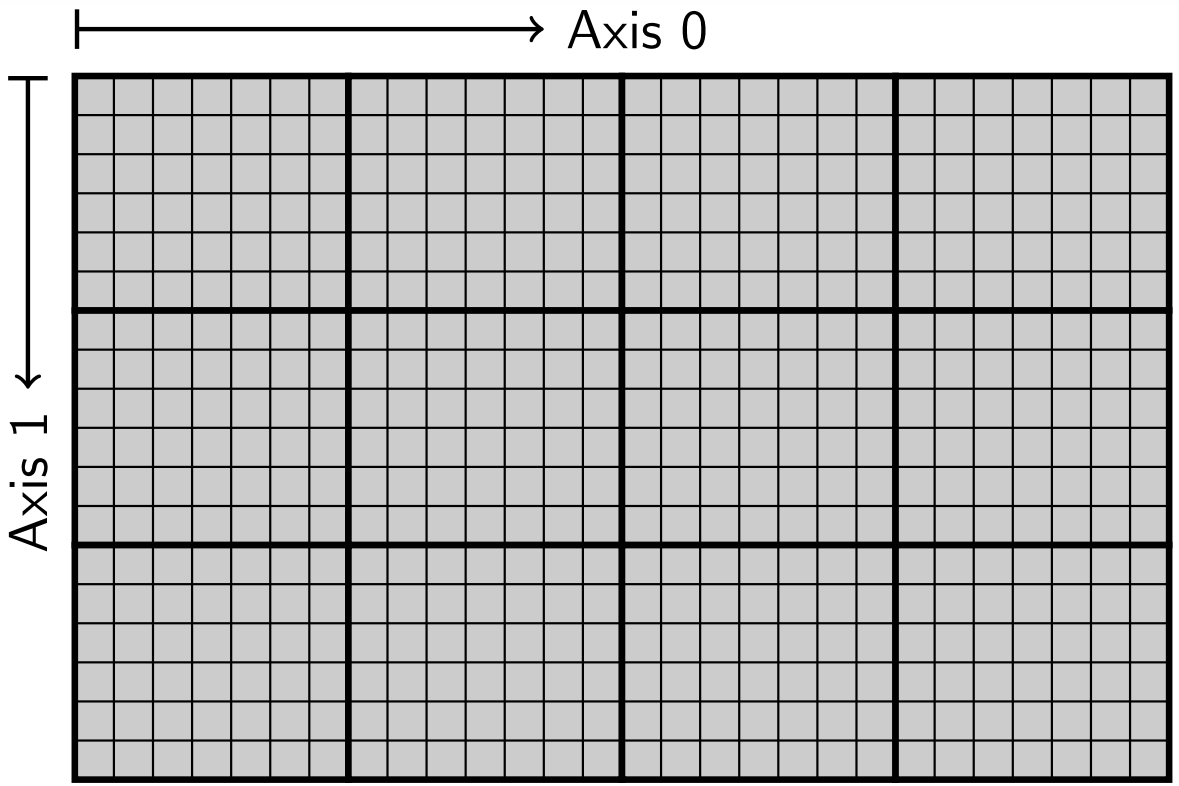
Connection: Hardware ↔ Programming Model



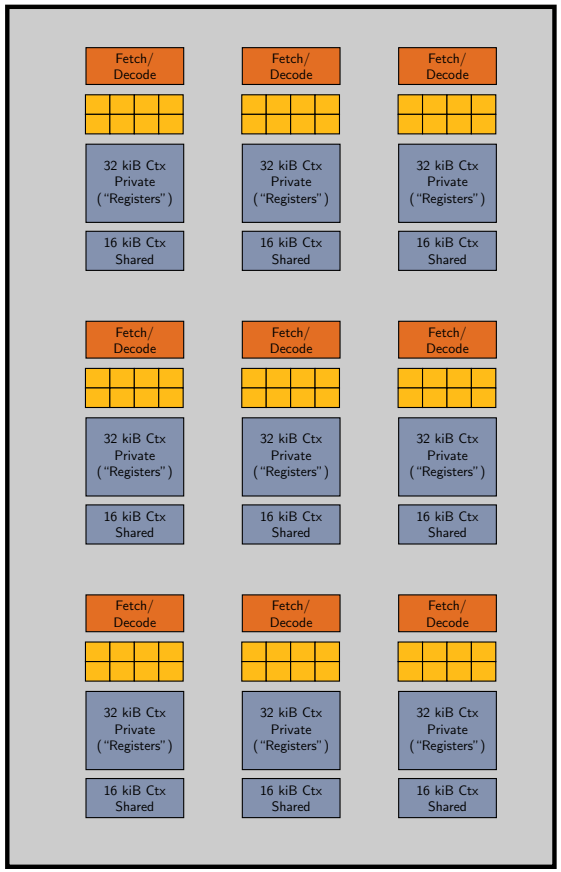


Hardware

Connection: Hardware ↔ Programming Model

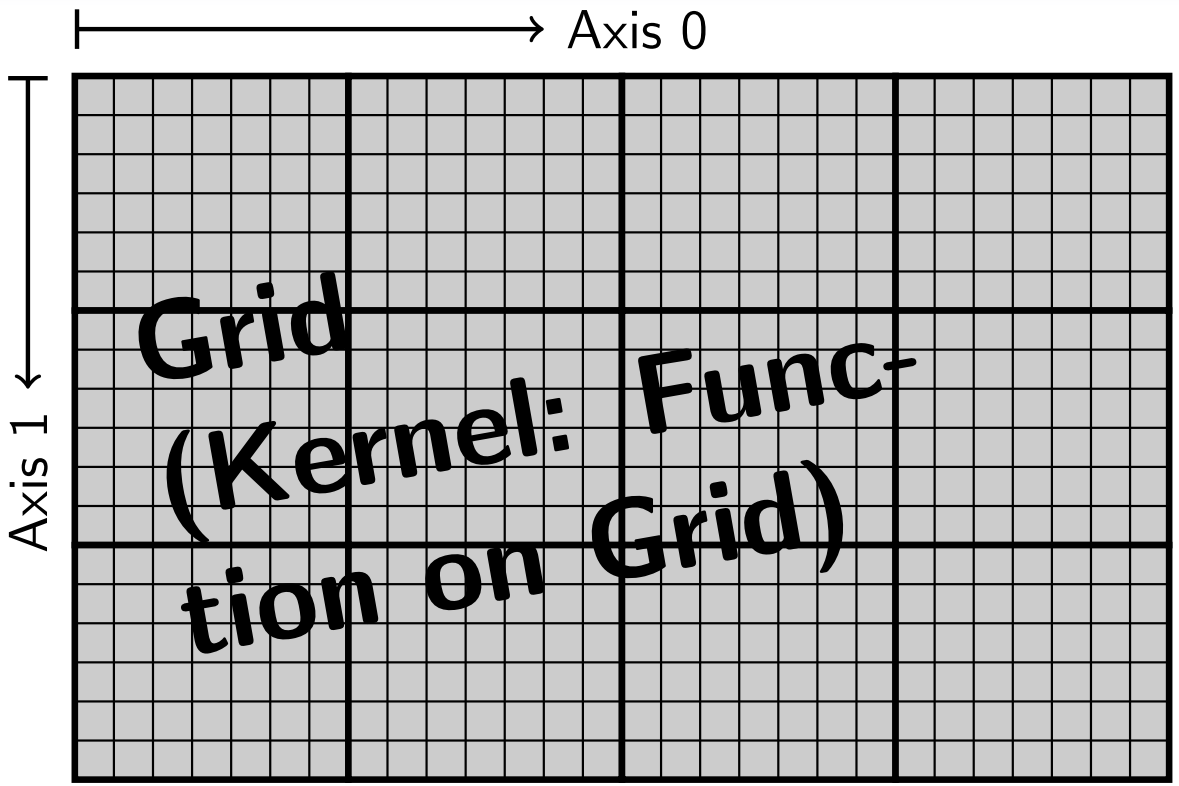


Software representation

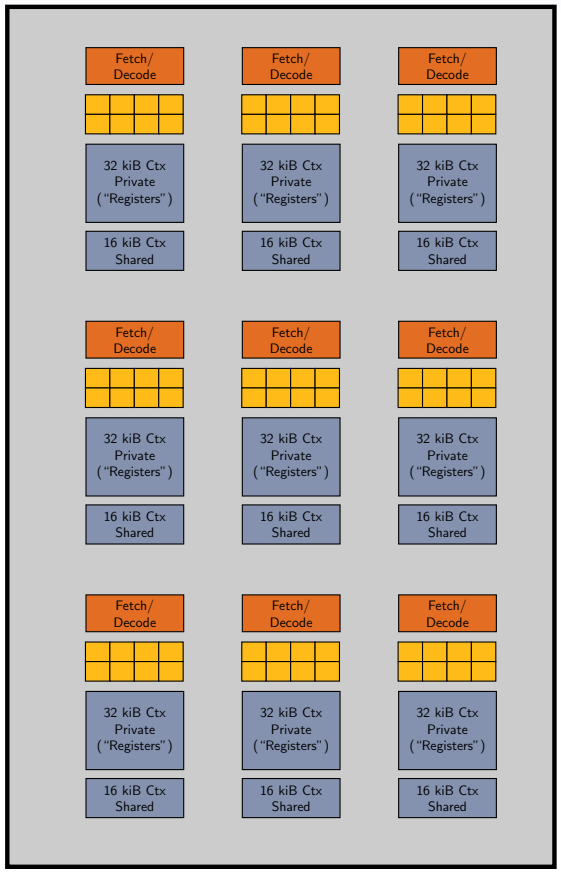


Hardware

Connection: Hardware ↔ Programming Model

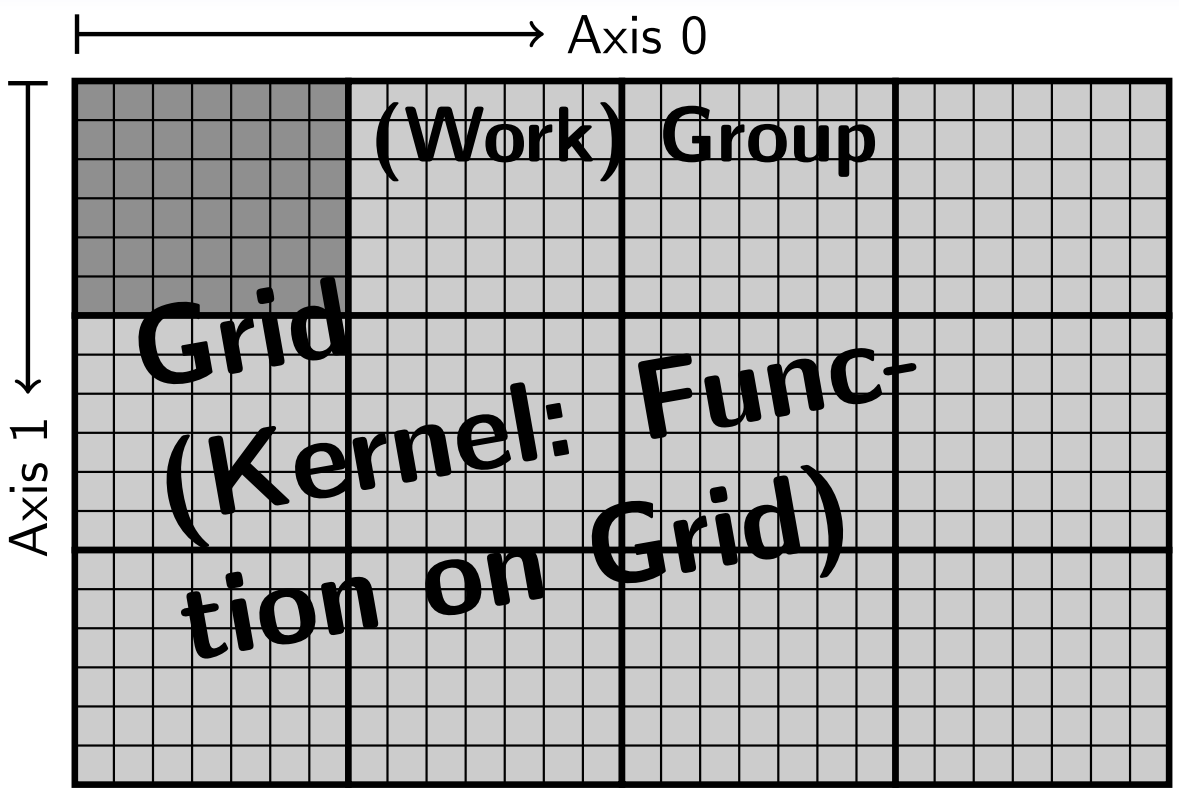


Software representation

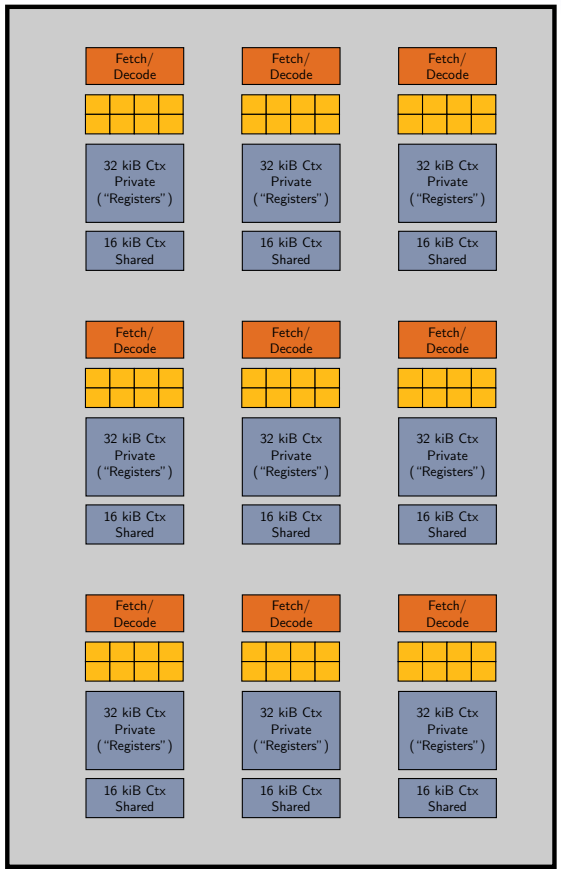


Hardware

Connection: Hardware $\leftrightarrow$ Programming Model

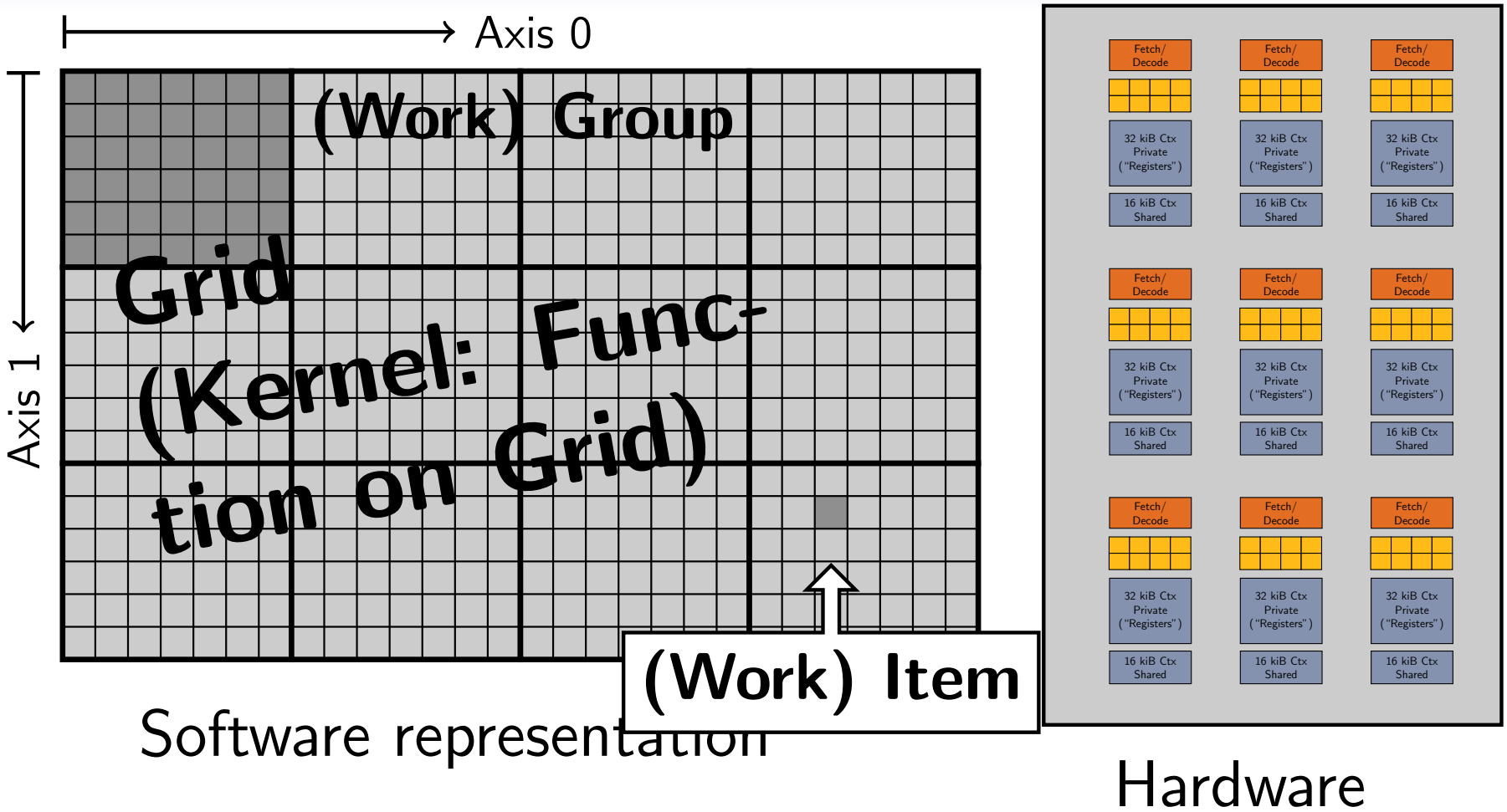


Software representation



Hardware

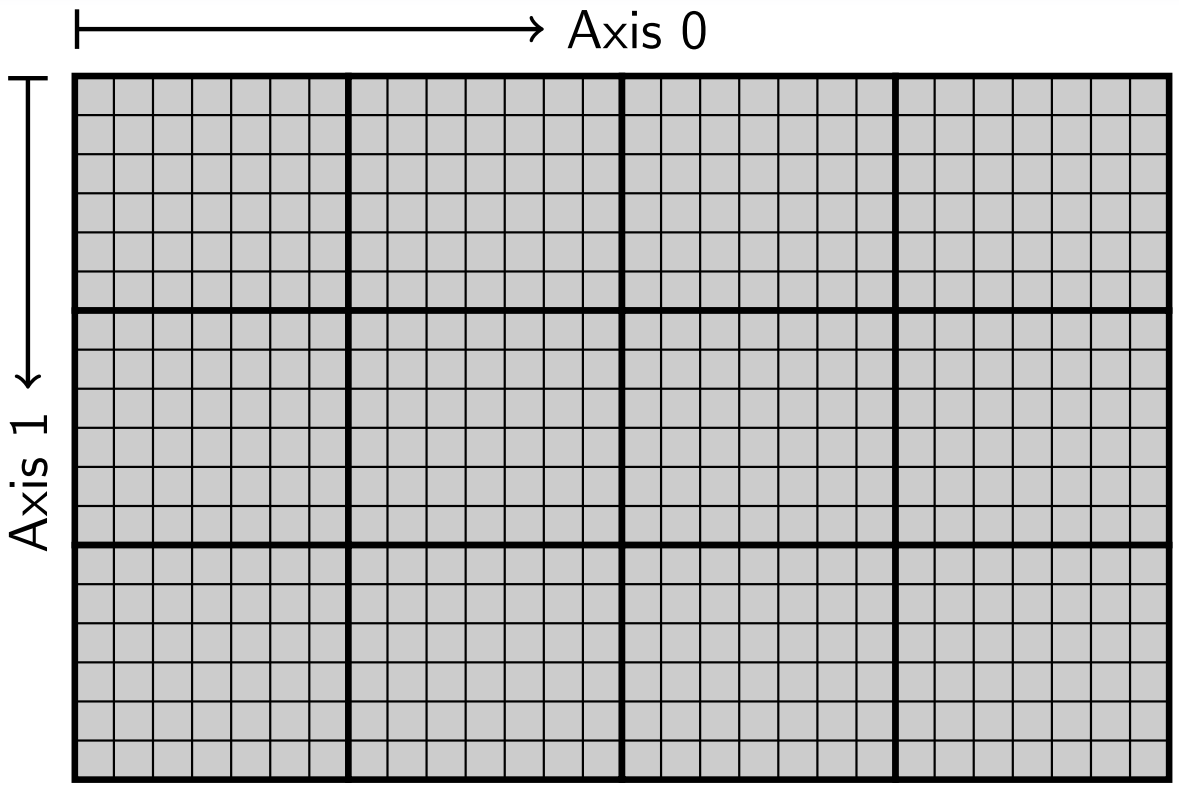
Connection: Hardware $\leftrightarrow$ Programming Model



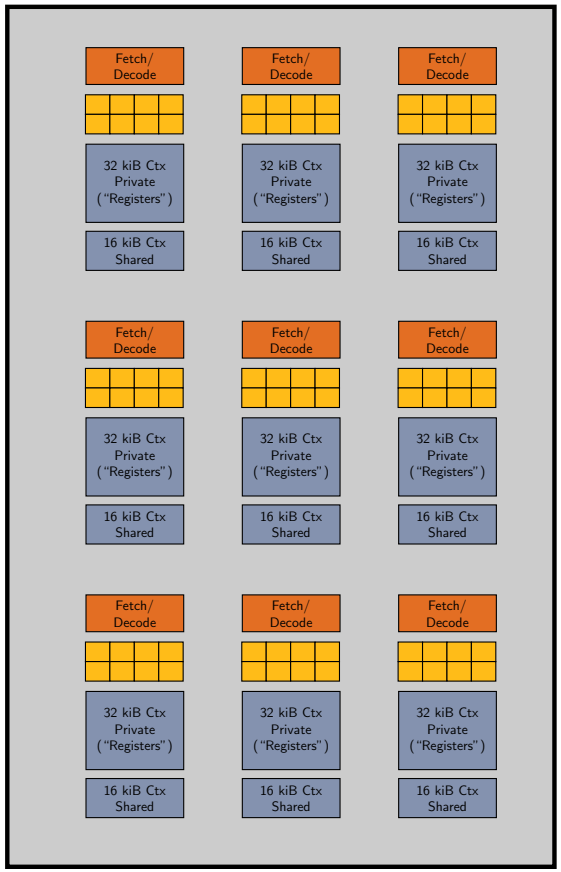
Software representation

Hardware

Connection: Hardware ↔ Programming Model

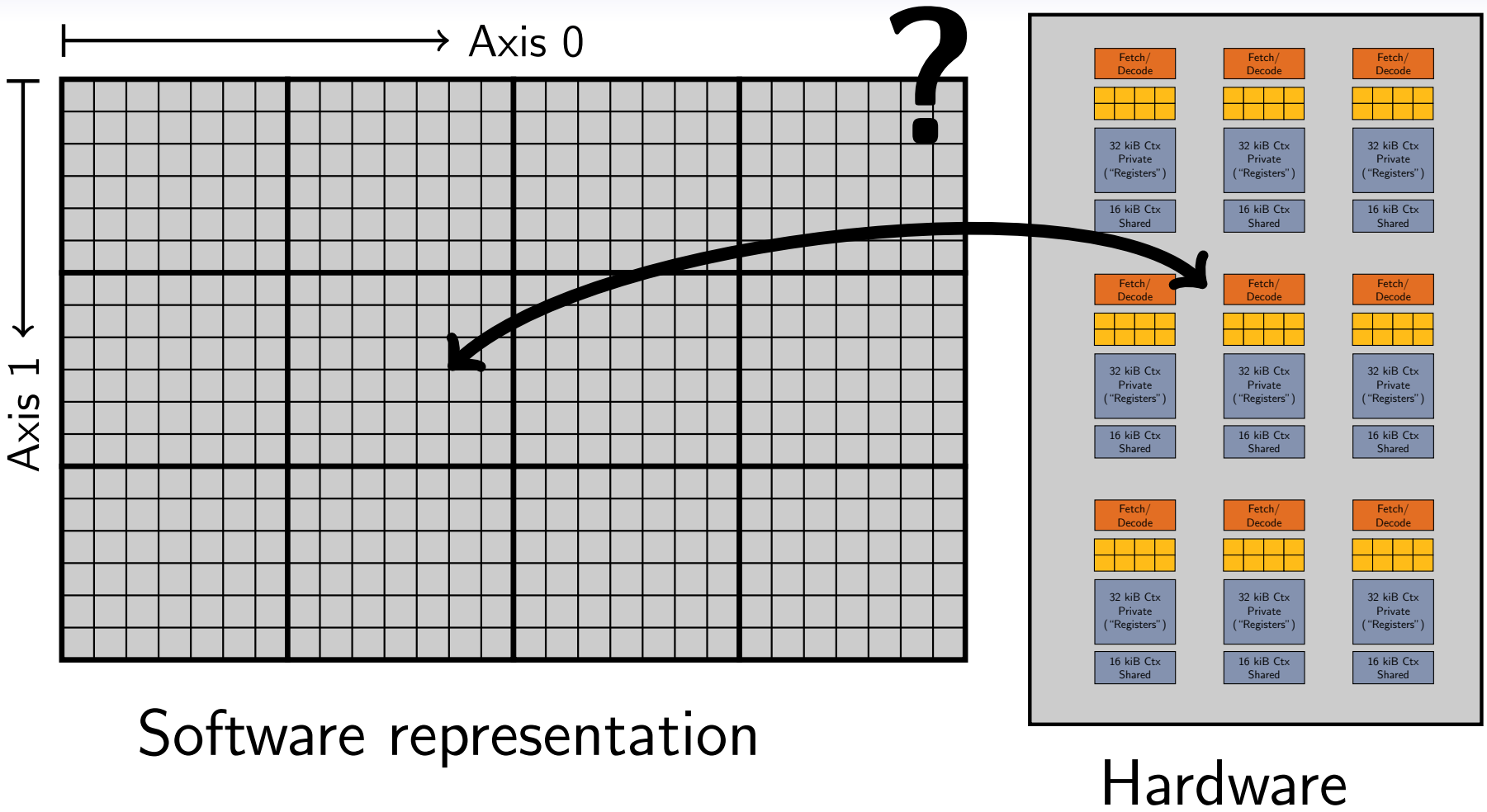


Software representation

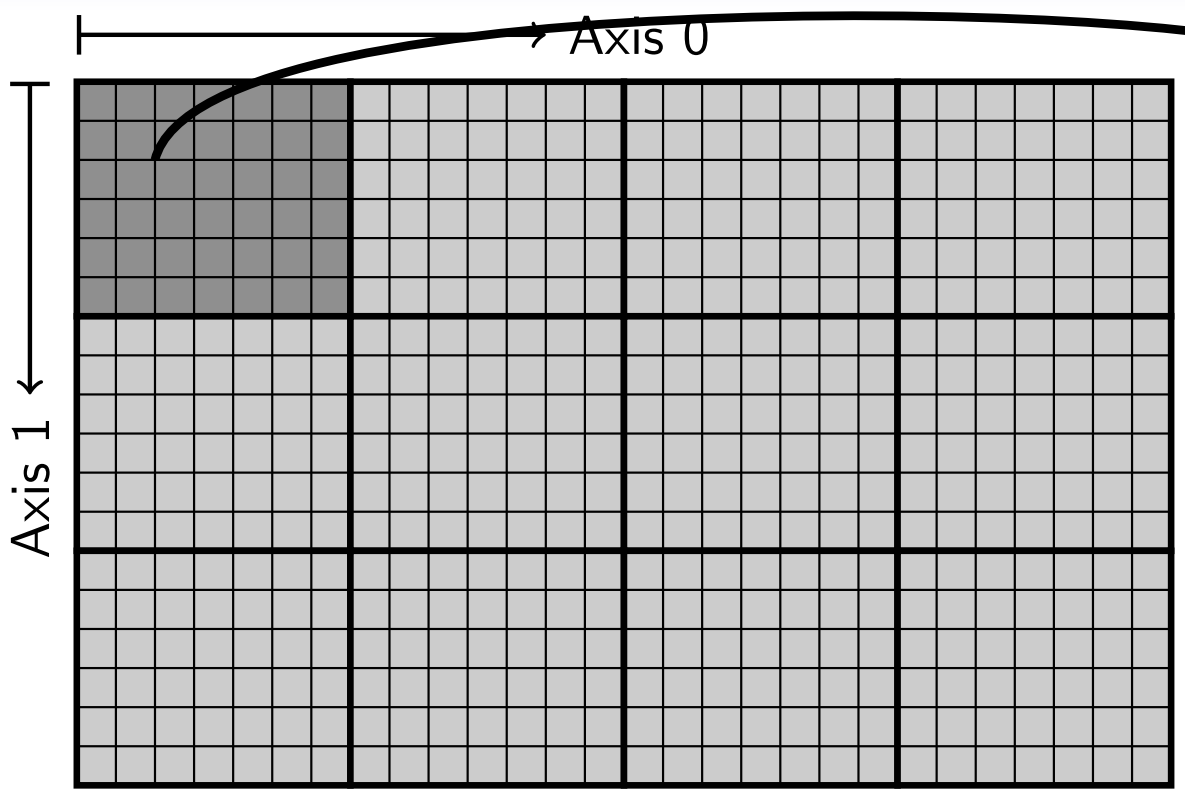


Hardware

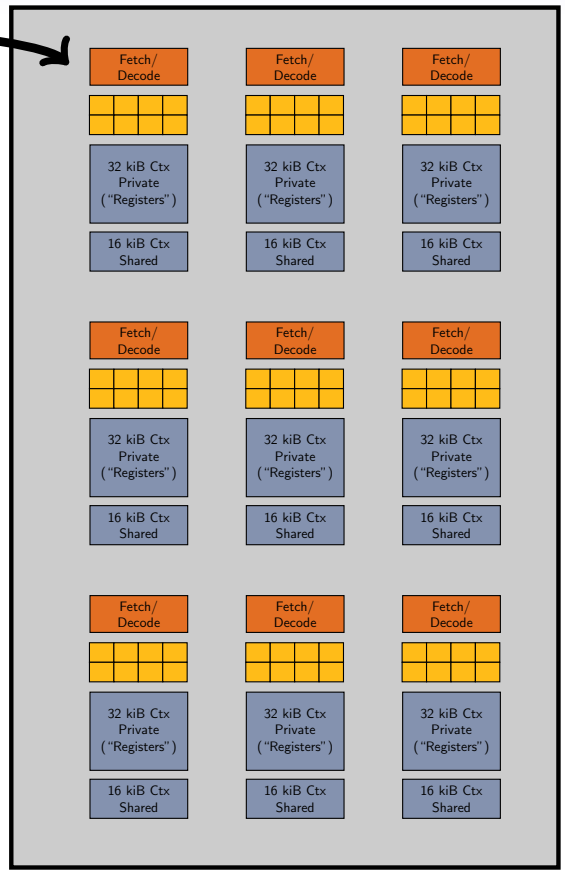
Connection: Hardware ↔ Programming Model



Connection: Hardware $\leftrightarrow$ Programming Model

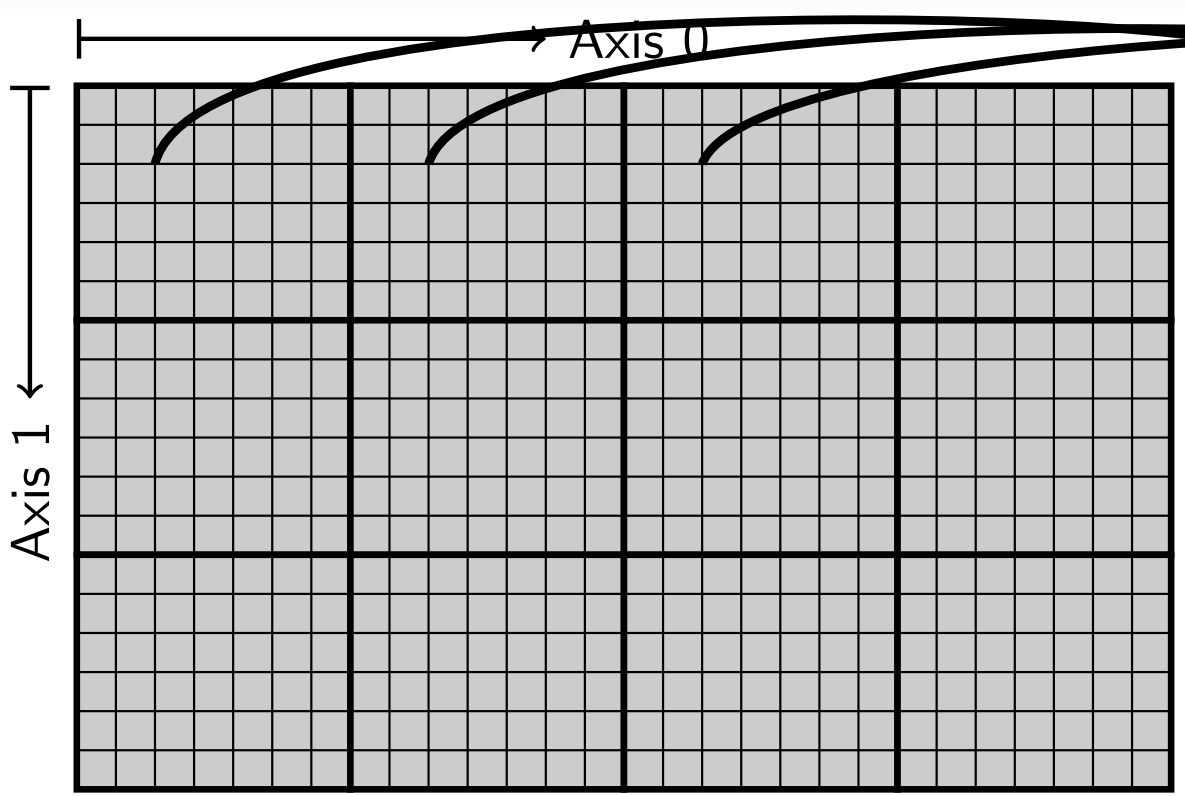


Software representation

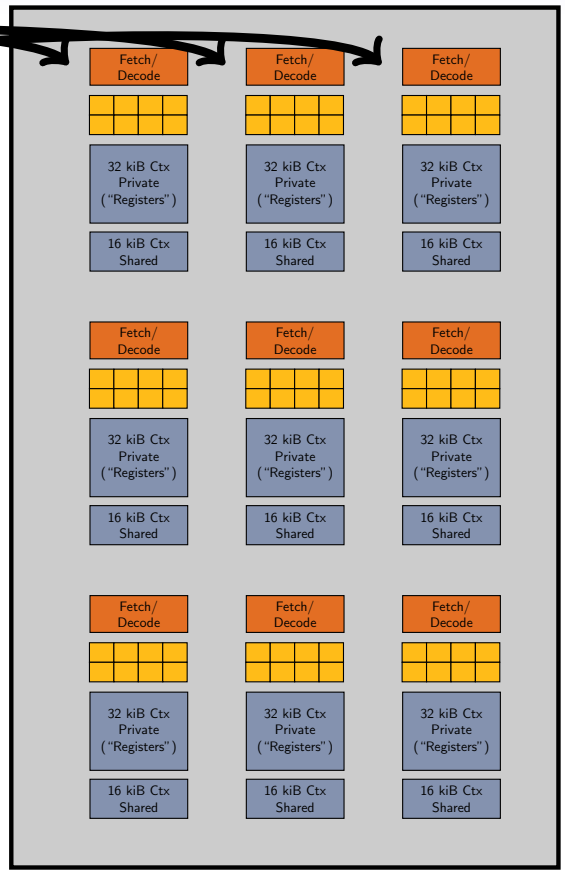


Hardware

Connection: Hardware $\leftrightarrow$ Programming Model

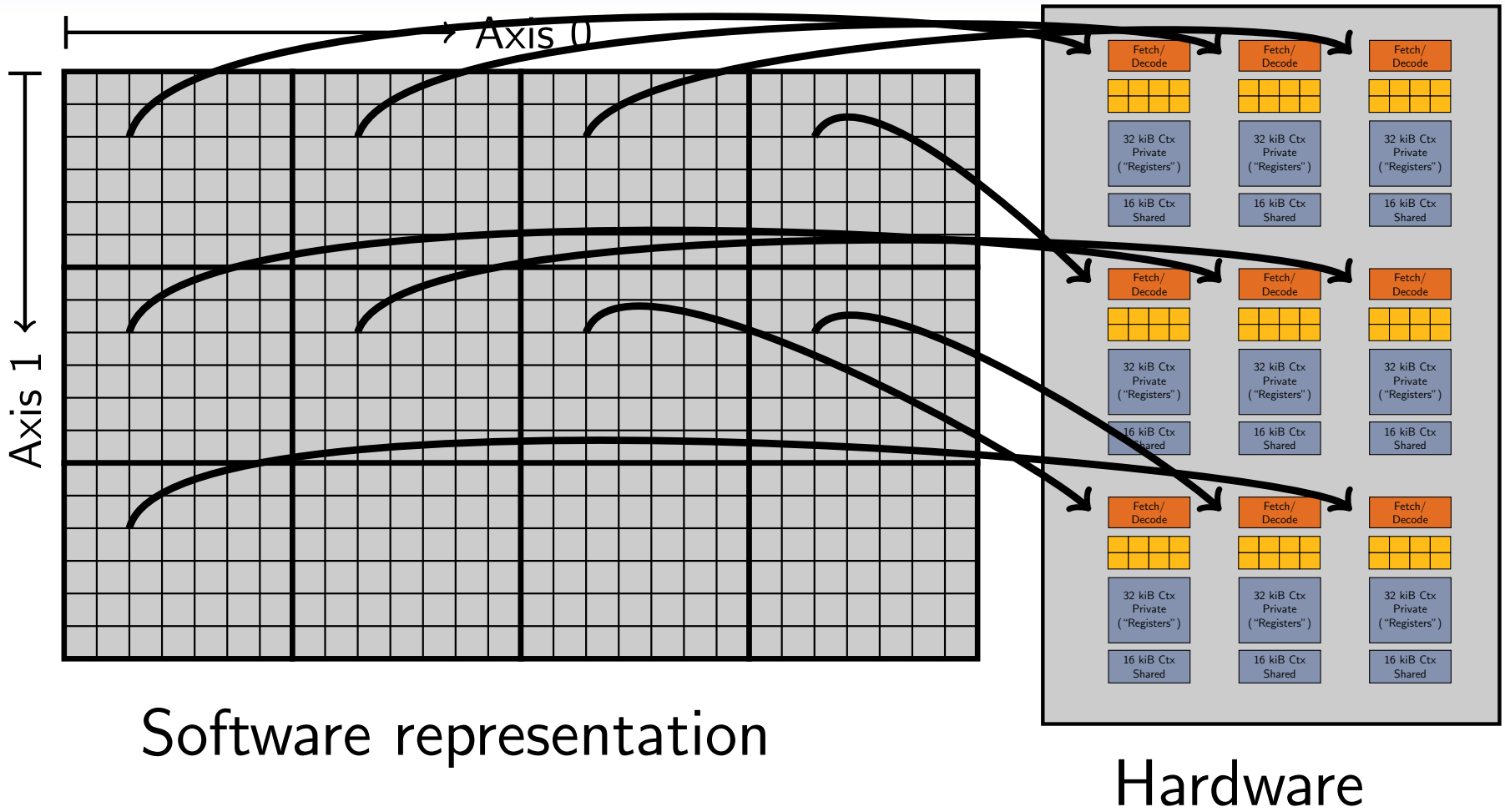


Software representation



Hardware

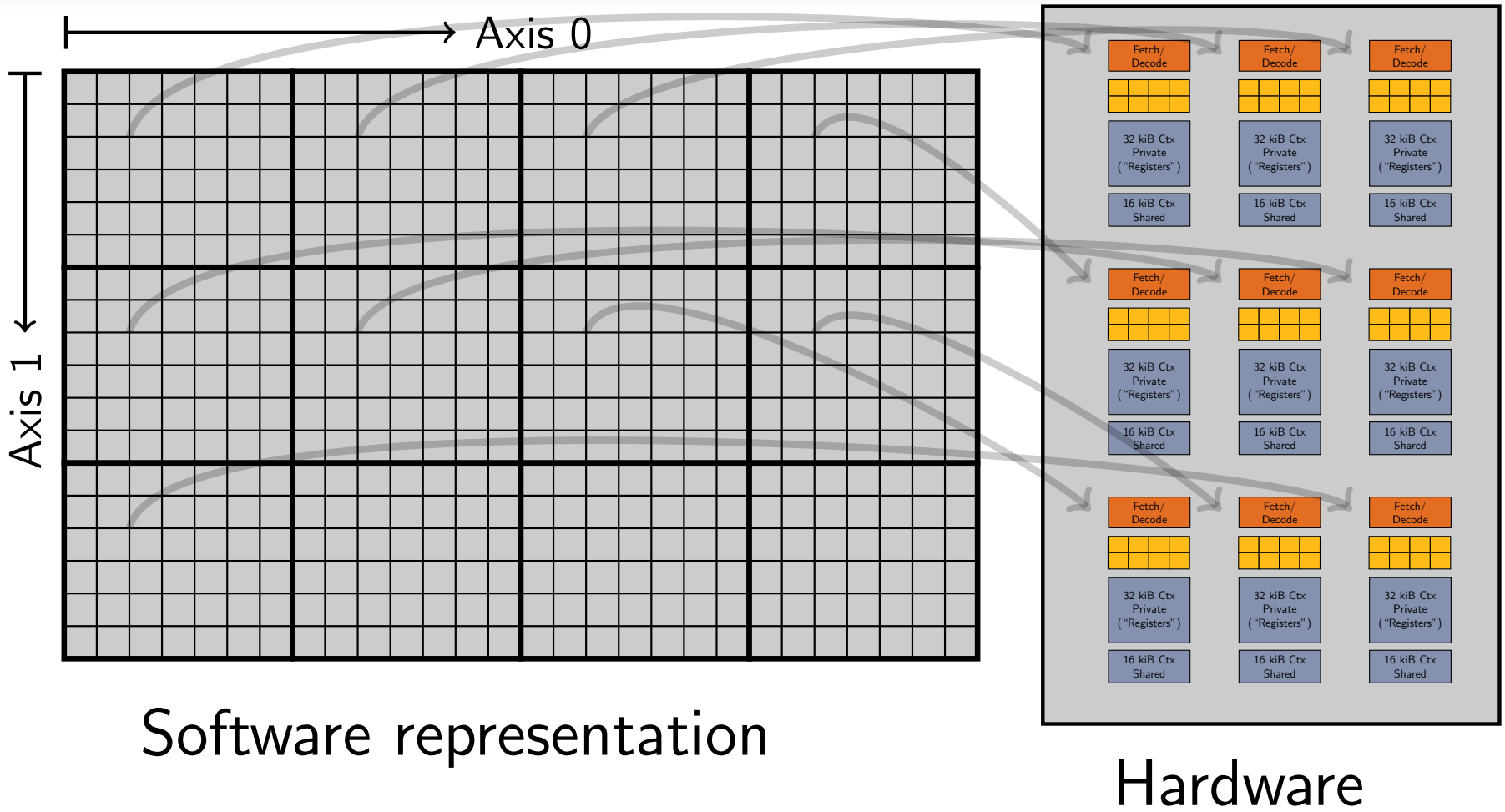
Connection: Hardware $\leftrightarrow$ Programming Model



Software representation

Hardware

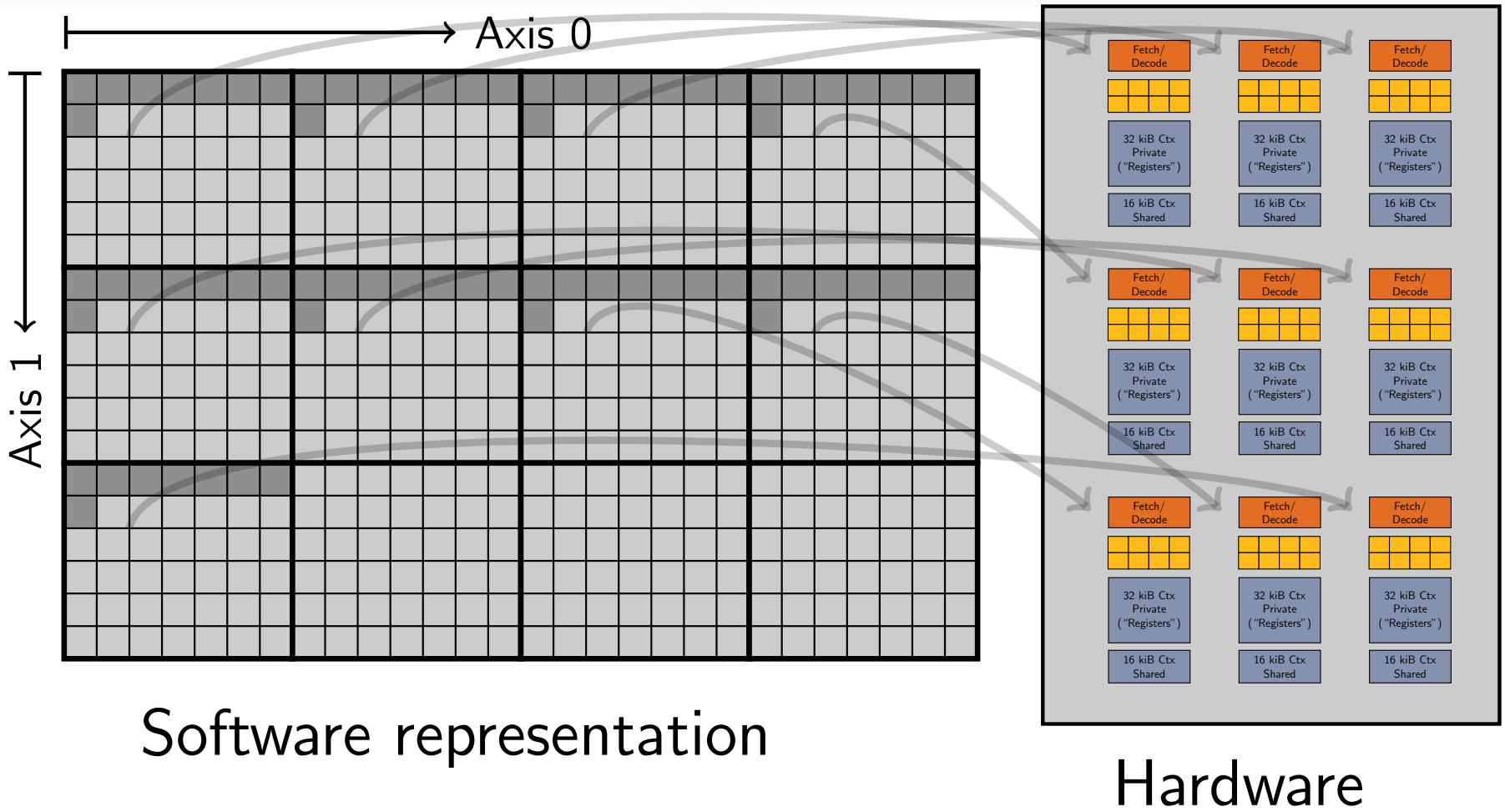
Connection: Hardware ↔ Programming Model



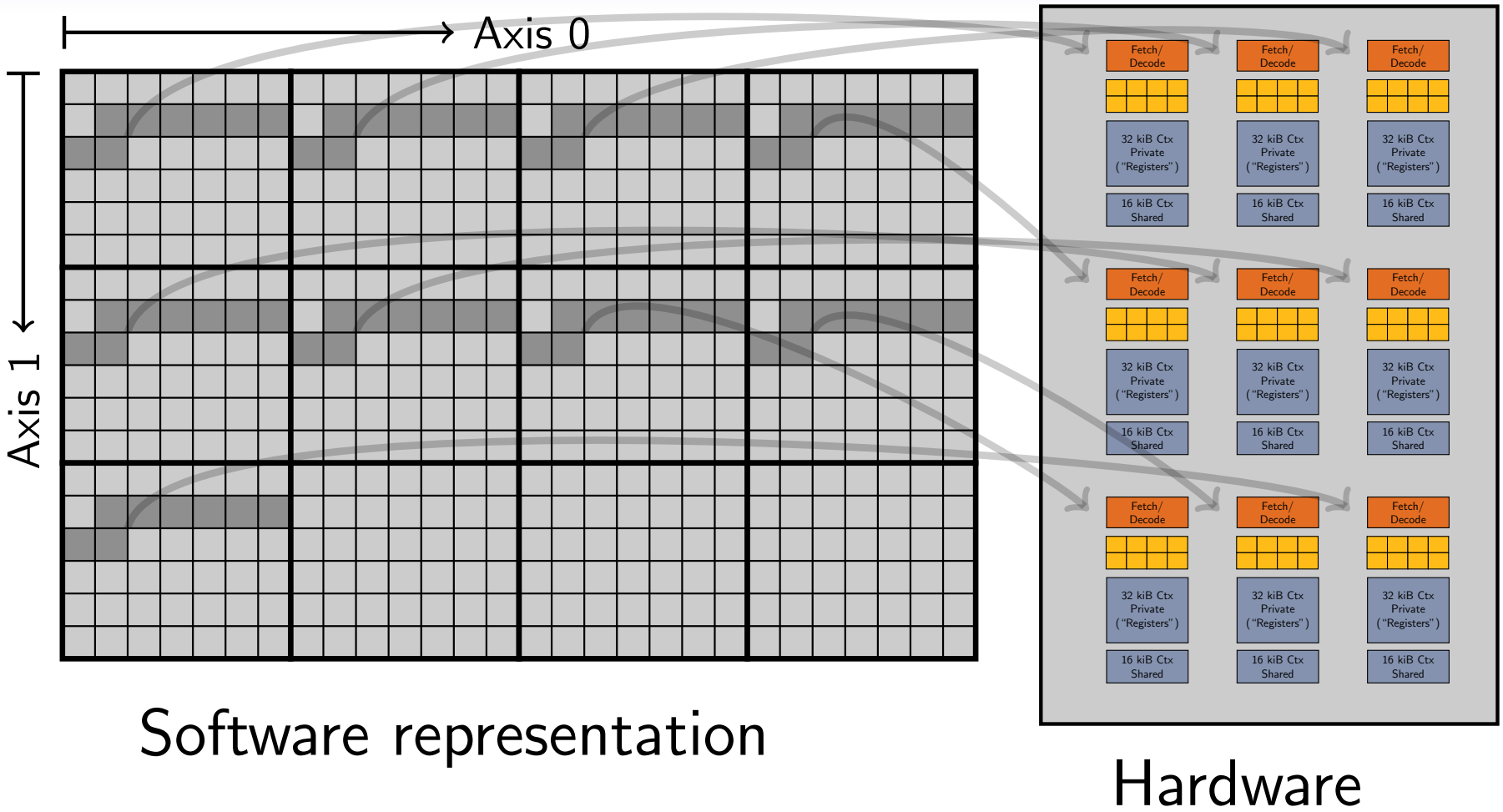
Software representation

Hardware

Connection: Hardware ↔ Programming Model



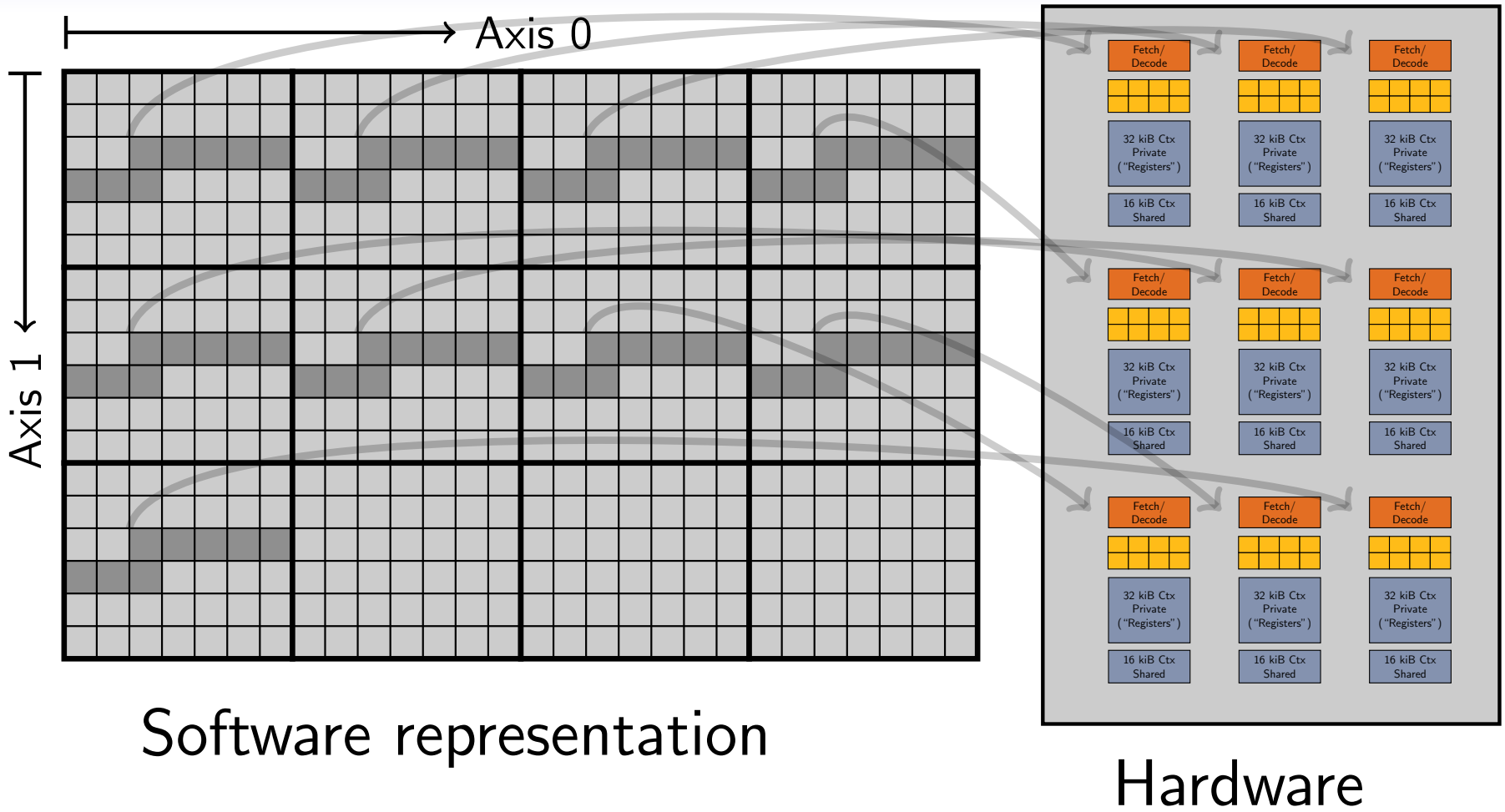
Connection: Hardware $\leftrightarrow$ Programming Model



Software representation

Hardware

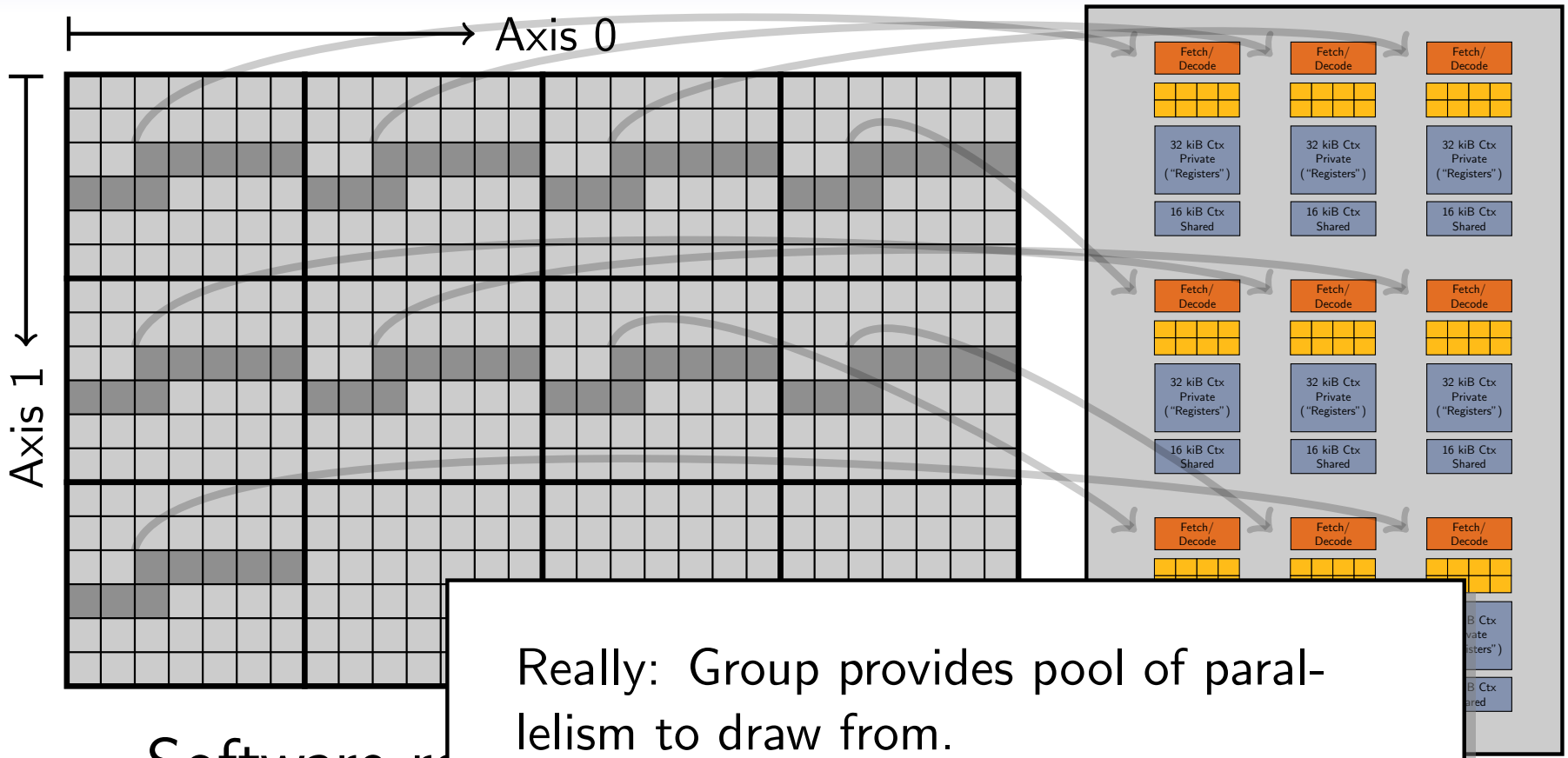
Connection: Hardware ↔ Programming Model



Software representation

Hardware

Connection: Hardware ↔ Programming Model

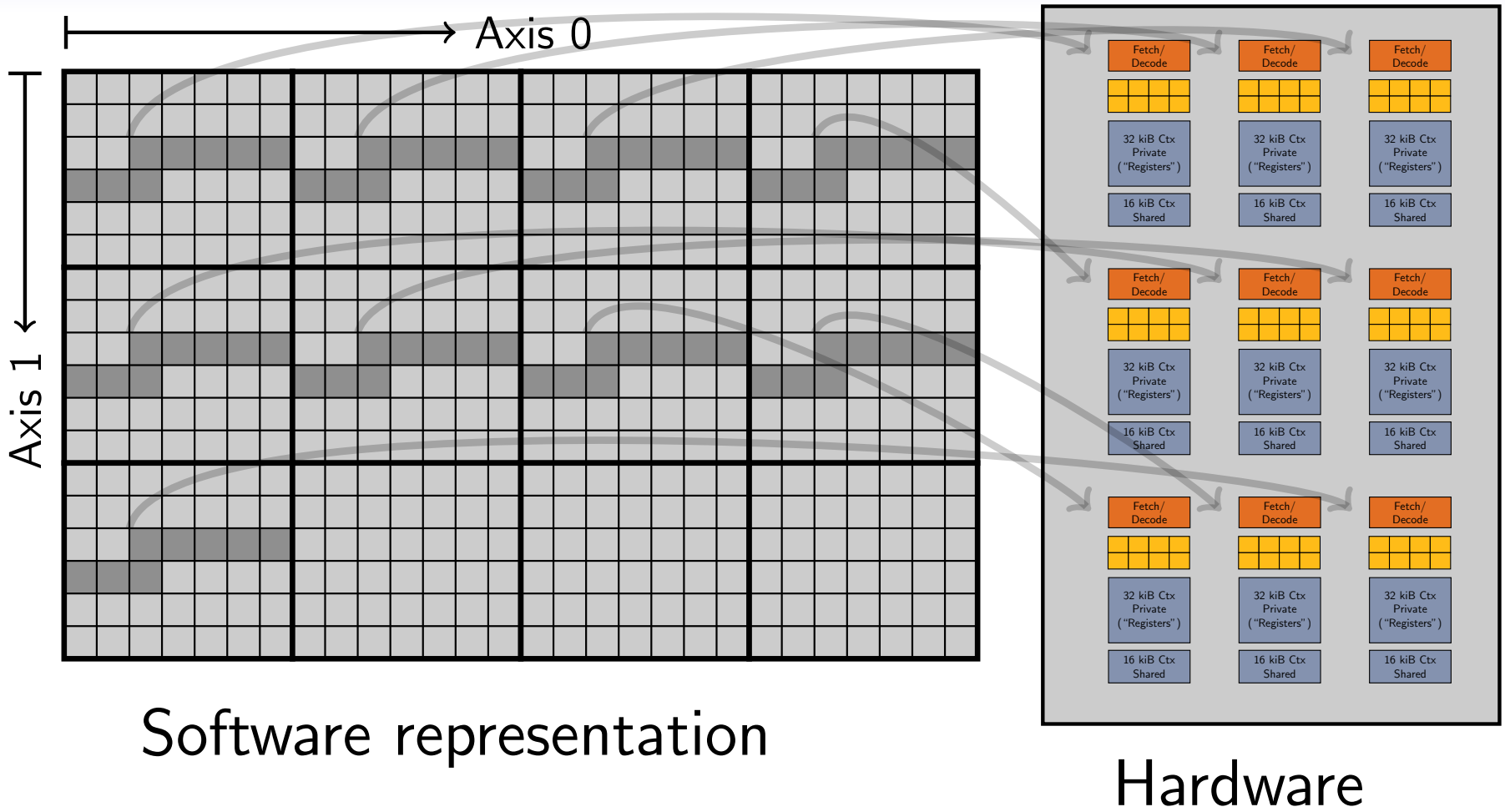


Software re

Really: Group provides pool of parallelism to draw from.

X,Y,Z order *within* group matters. (Not *among* groups, though.)

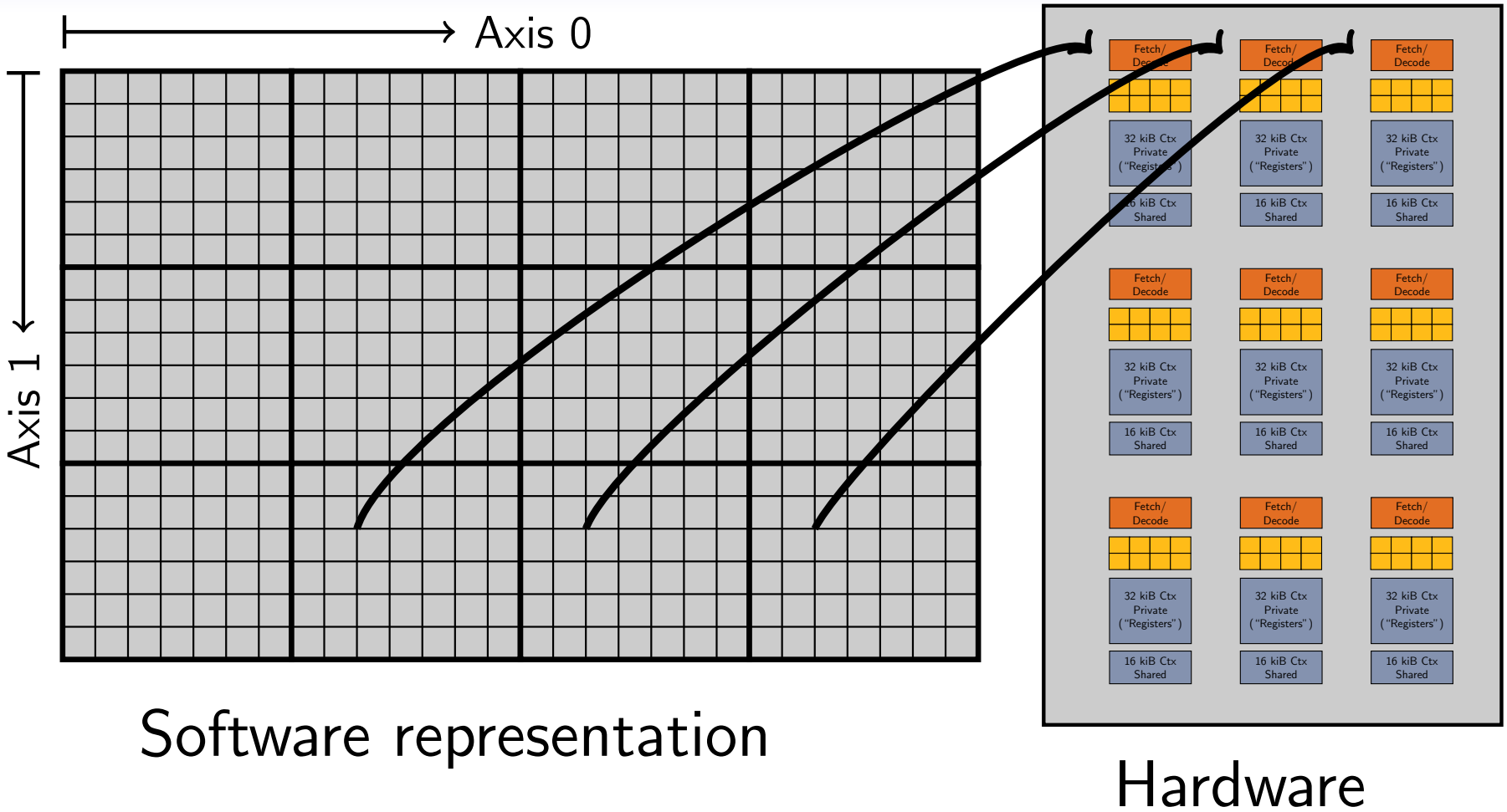
Connection: Hardware ↔ Programming Model



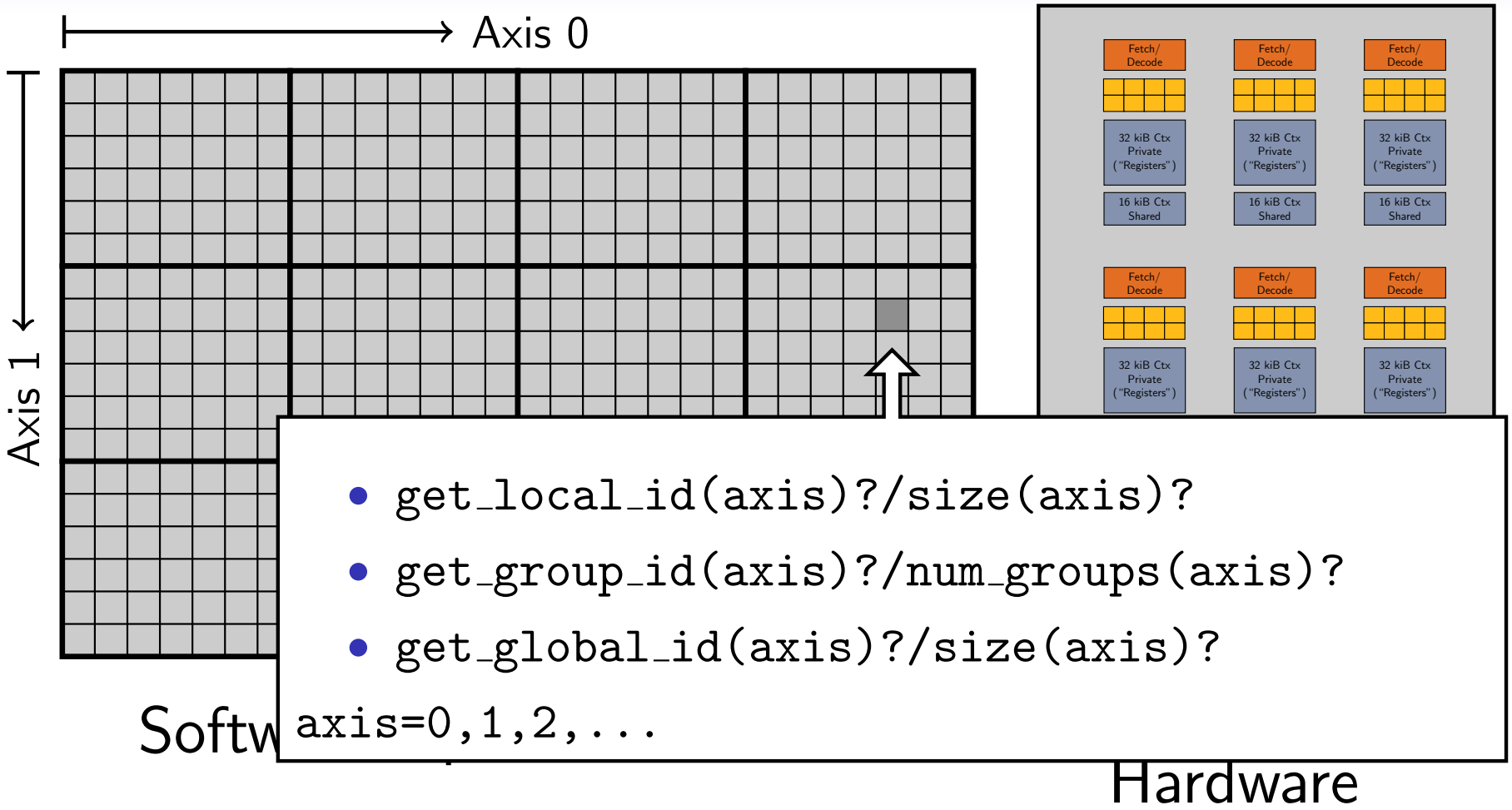
Software representation

Hardware

Connection: Hardware ↔ Programming Model



Connection: Hardware ↔ Programming Model

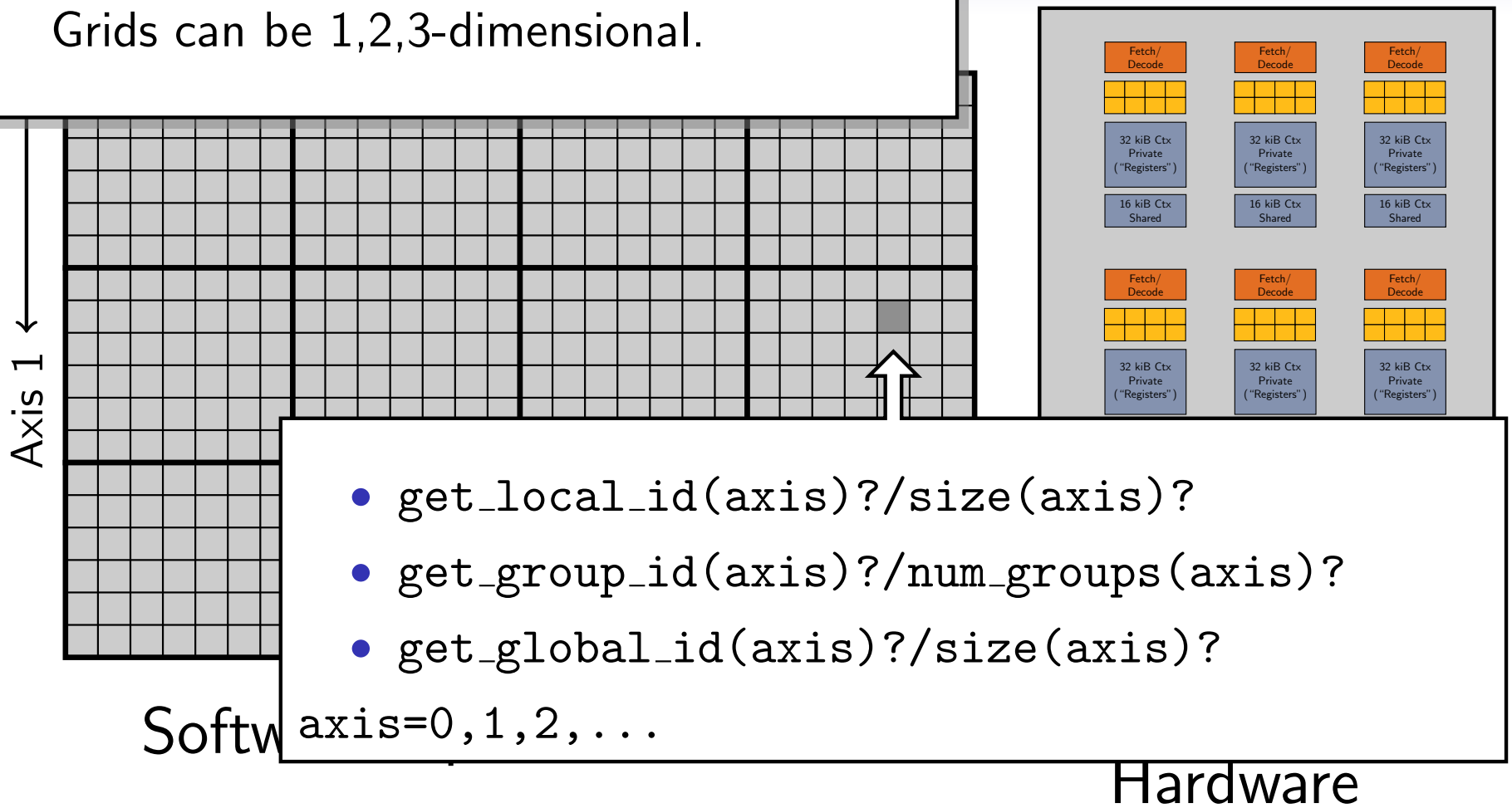


- `get_local_id(axis)?/size(axis)?`
- `get_group_id(axis)?/num_groups(axis)?`
- `get_global_id(axis)?/size(axis)?`

`axis=0,1,2,...`

Connection: Hardware ↔ Programming Model

Grids can be 1,2,3-dimensional.



- `get_local_id(axis)?/size(axis)?`
- `get_group_id(axis)?/num_groups(axis)?`
- `get_global_id(axis)?/size(axis)?`

`axis=0,1,2,...`

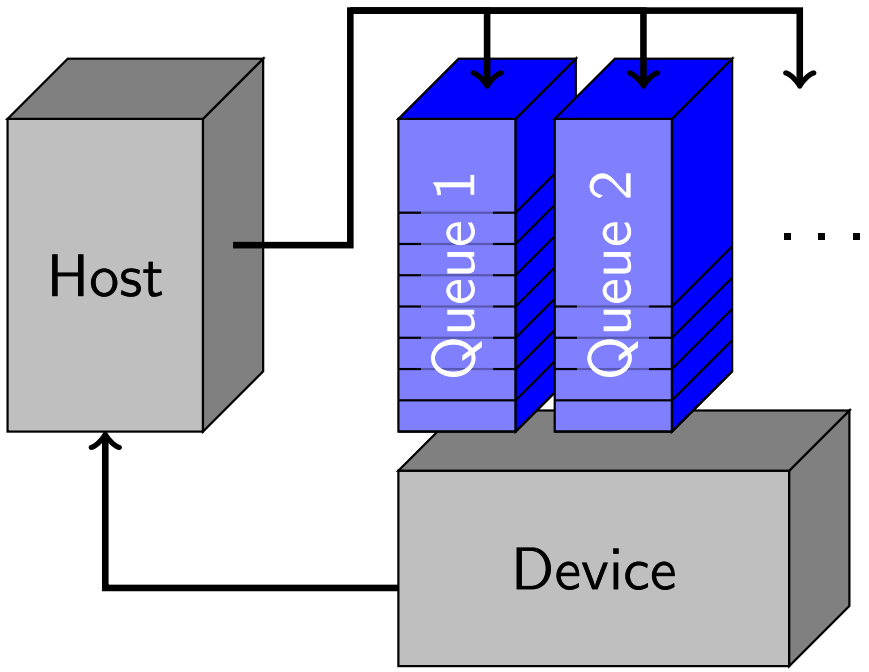
Dive into OpenCL: Preparation

Demo time

Outline

OpenCL: Command Queues

- Host and Device run asynchronously
- Host submits to queue:
  - Computations
  - Memory Transfers
  - Sync primitives
  - ...
- Host can wait for drained queue
- Profiling



Outline

OpenCL Device Language

OpenCL device language is C99, with these differences:

- ⊕ Index getters
- ⊕ Memory space qualifiers
- ⊕ Vector data types
- ⊕ Many generic ('overloaded') math functions
- ⊕ Synchronization
- ⊖ Recursion
- ⊖ Fine-grained `malloc()`
- ⊖ Function pointers

Address Space Qualifiers

Type	**Per**	**“Speed”**
private*)	work item	super-fast
local	group	fast
global	grid	kinda slow

*) default, so optional

Address Space Qualifiers

Type	**Per**	**“Speed”**
private*)	work item	super-fast
local	group	fast
global	grid	kinda slow

*) default, so optional

Should really discuss “speed” in terms of latency/bandwidth.

Both decrease with distance from the point of execution.

Outline

Concurrency and Synchronization

GPUs have layers of concurrency.

Each layer has its synchronization primitives.

Concurrency and Synchronization

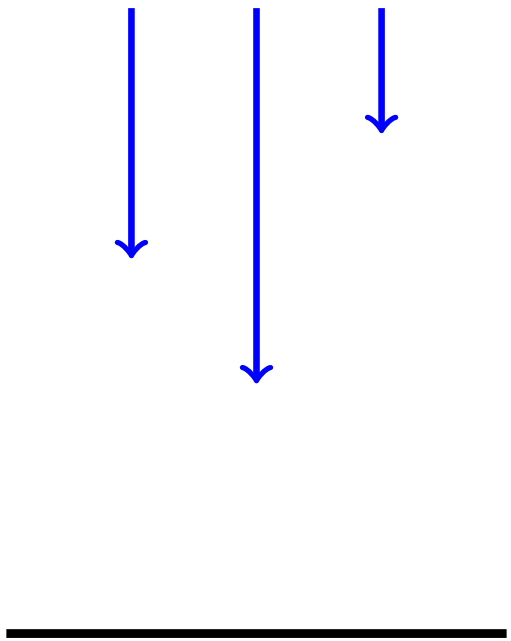
GPUs have layers of concurrency.

Each layer has its synchronization primitives.

- Intra-group:
  `barrier(...)`,
  `mem_fence(...)`
  `...` =
  `CLK_{LOCAL,GLOBAL}_MEM_FENCE`
- Inter-group:
  Kernel launch
- CPU-GPU:
  Command queues, Events

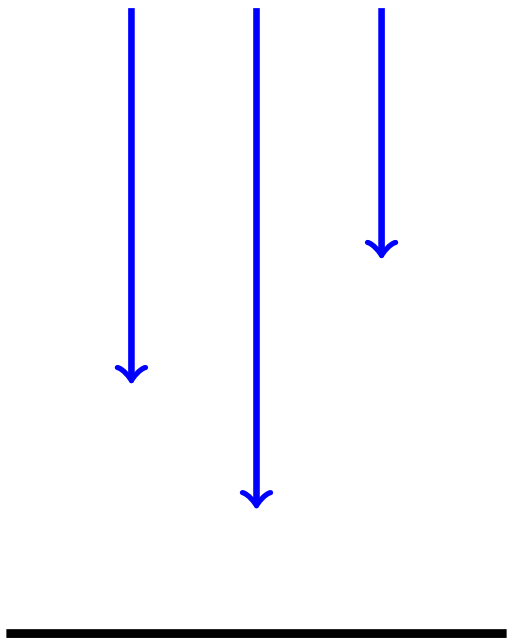
Synchronization

What is a Barrier?



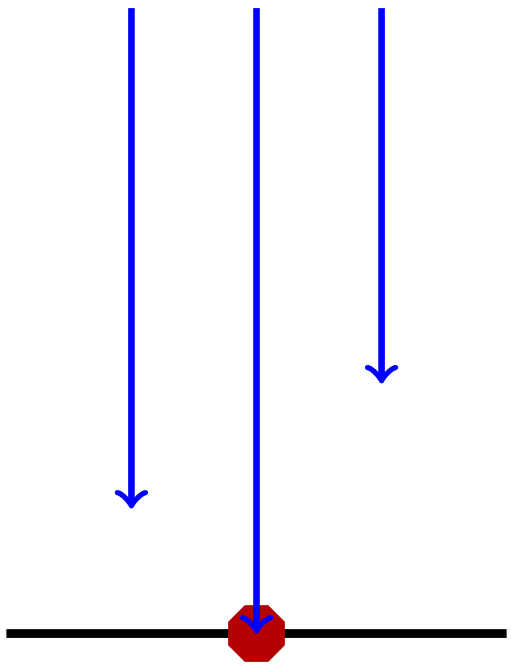
Synchronization

What is a Barrier?



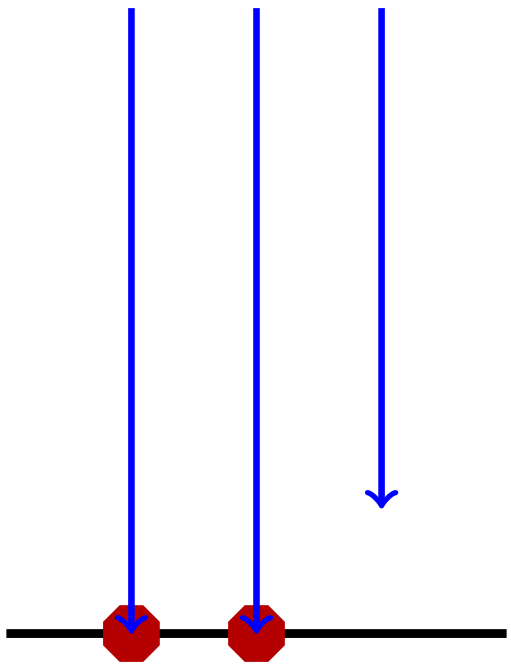
Synchronization

What is a Barrier?



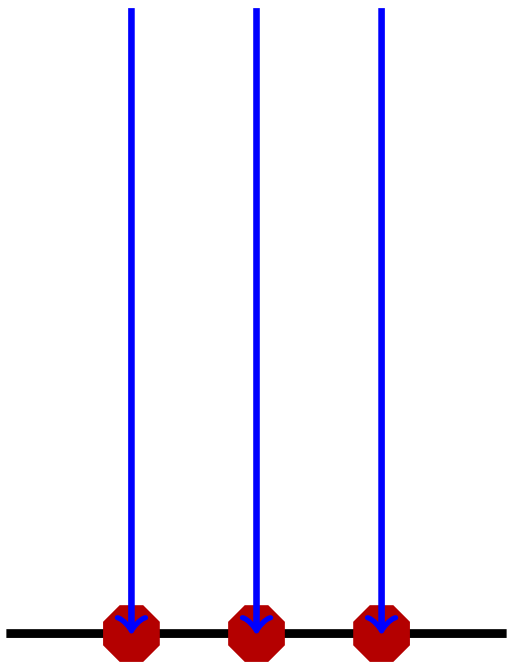
Synchronization

What is a Barrier?



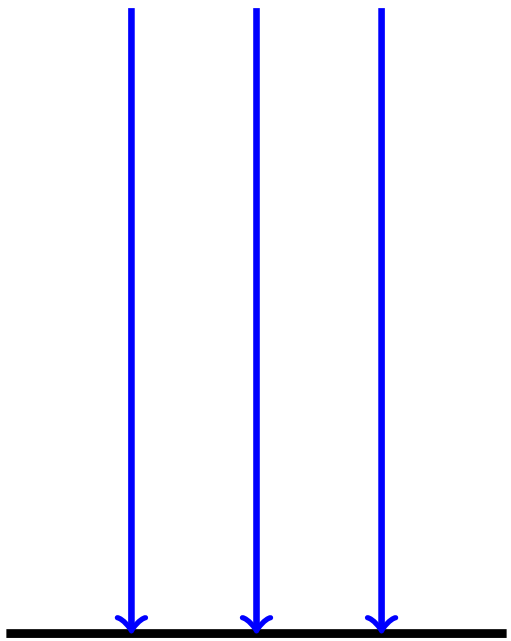
Synchronization

What is a Barrier?



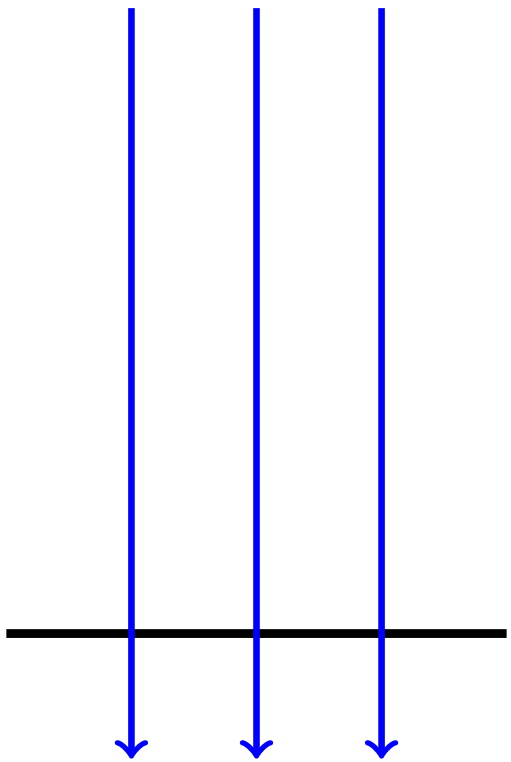
Synchronization

What is a Barrier?



Synchronization

What is a Barrier?



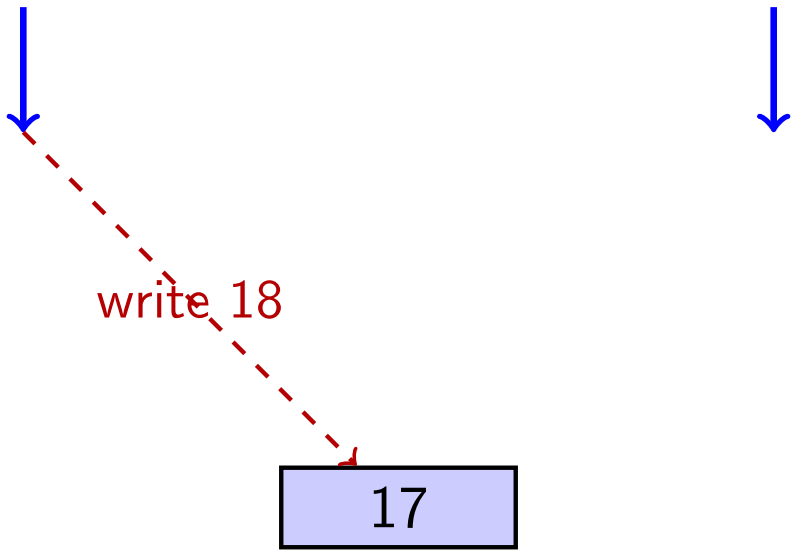
Synchronization

What is a Memory Fence?

17

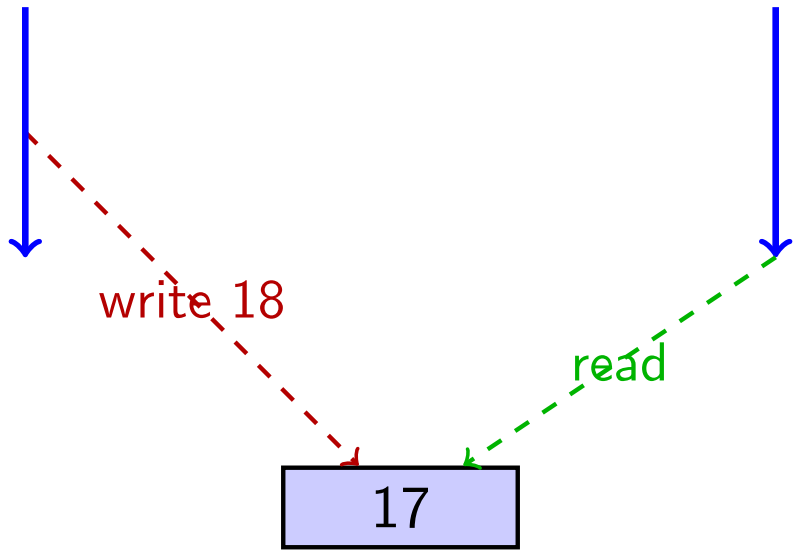
Synchronization

What is a Memory Fence?



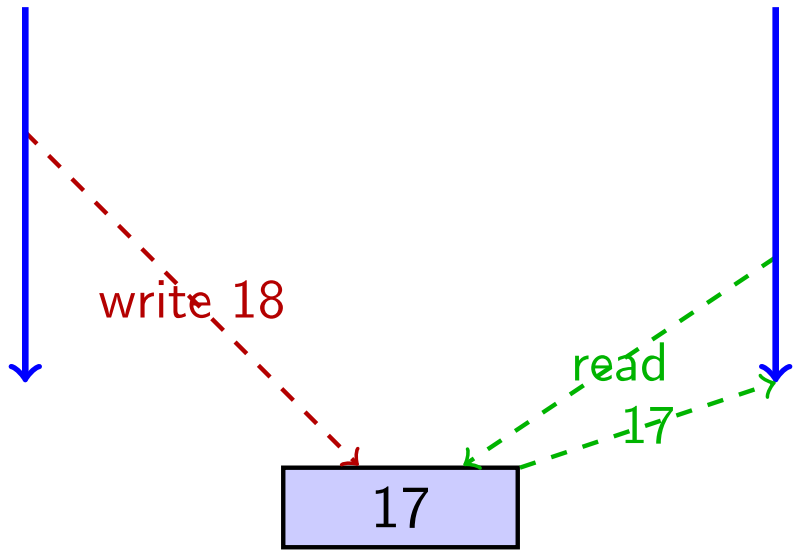
Synchronization

What is a Memory Fence?



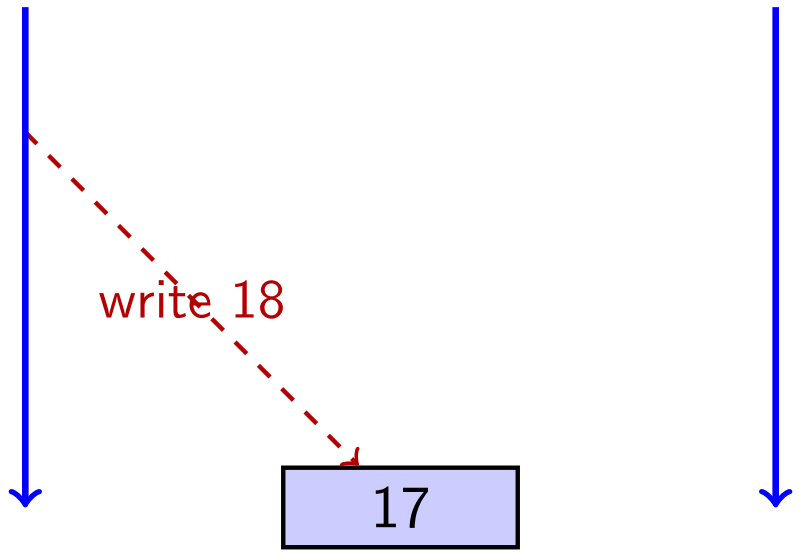
Synchronization

What is a Memory Fence?



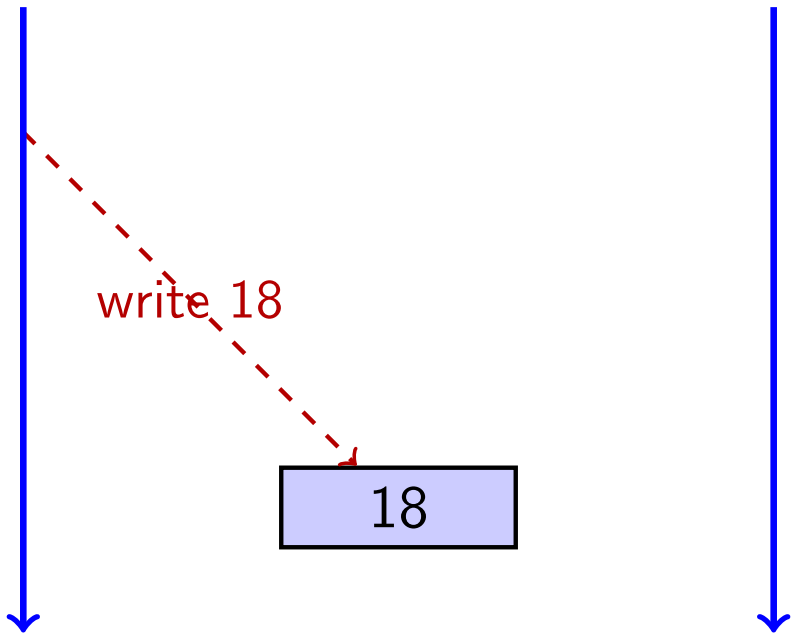
Synchronization

What is a Memory Fence?



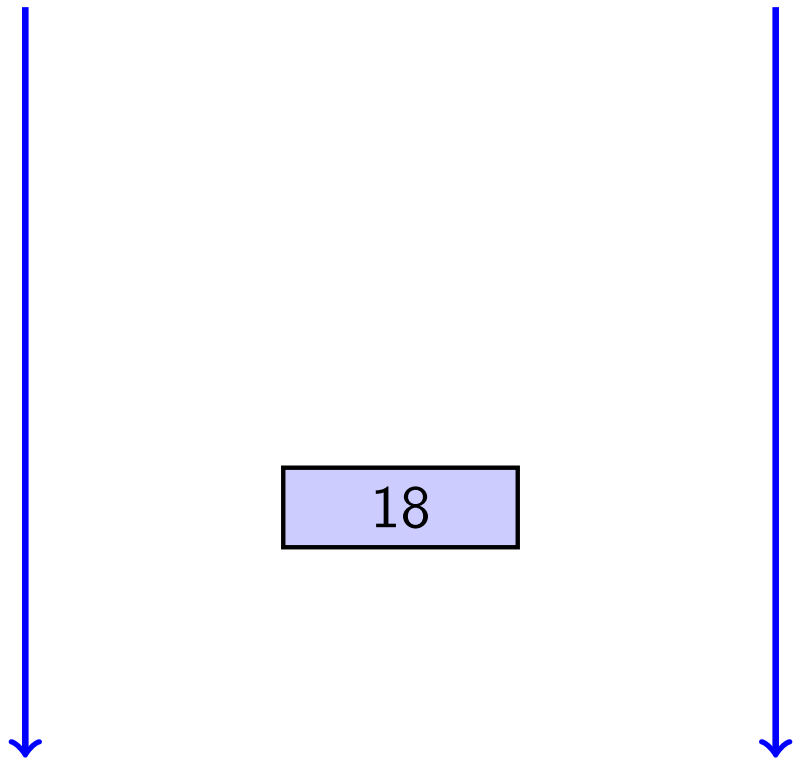
Synchronization

What is a Memory Fence?



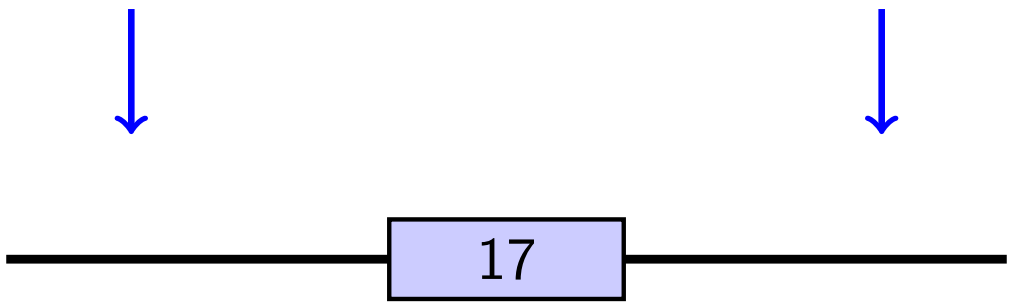
Synchronization

What is a Memory Fence?



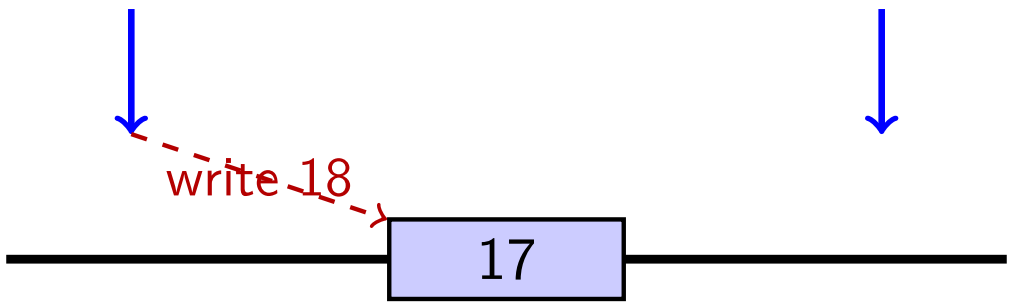
Synchronization

What is a Memory Fence? An ordering restriction for memory access.



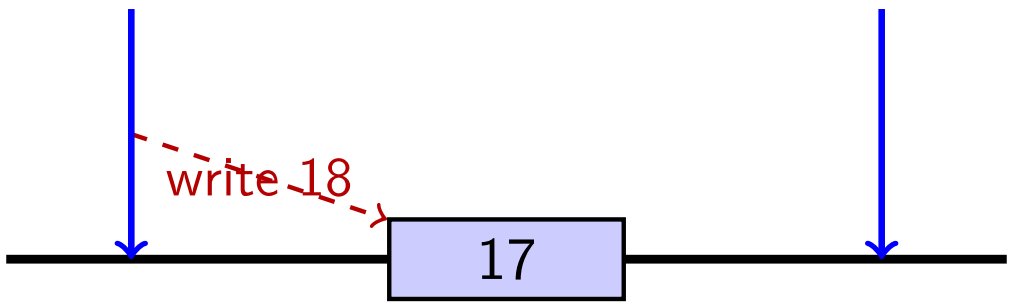
Synchronization

What is a Memory Fence? An ordering restriction for memory access.



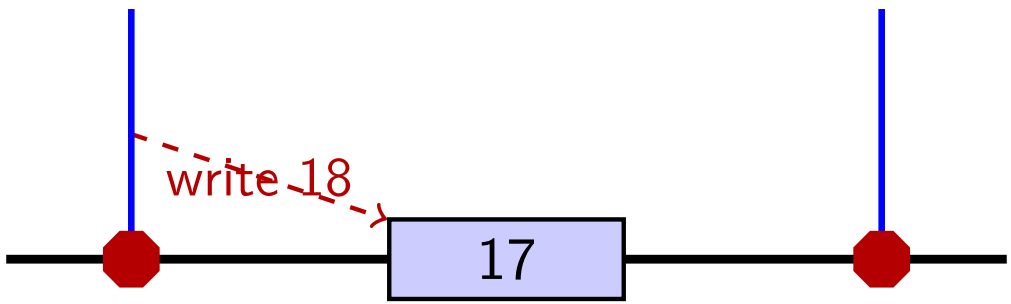
Synchronization

What is a Memory Fence? An ordering restriction for memory access.



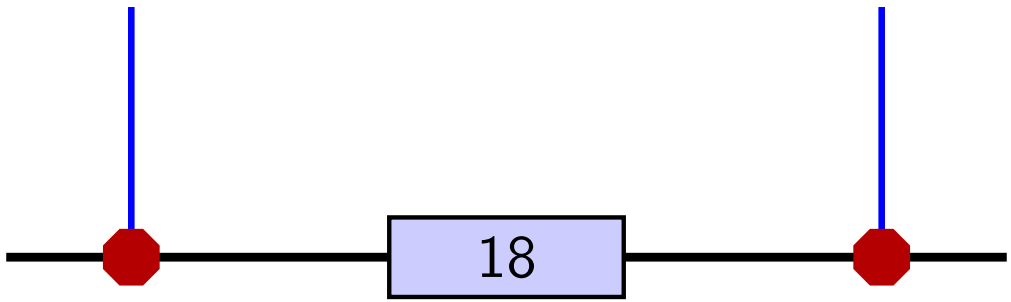
Synchronization

What is a Memory Fence? An ordering restriction for memory access.



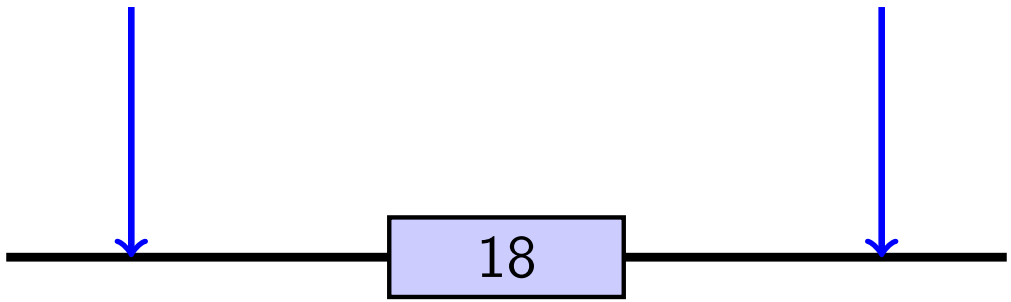
Synchronization

What is a Memory Fence? An ordering restriction for memory access.



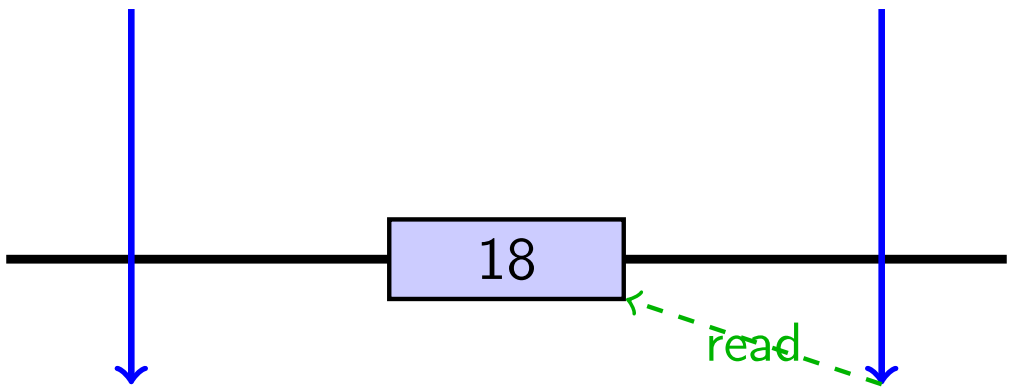
Synchronization

What is a Memory Fence? An ordering restriction for memory access.



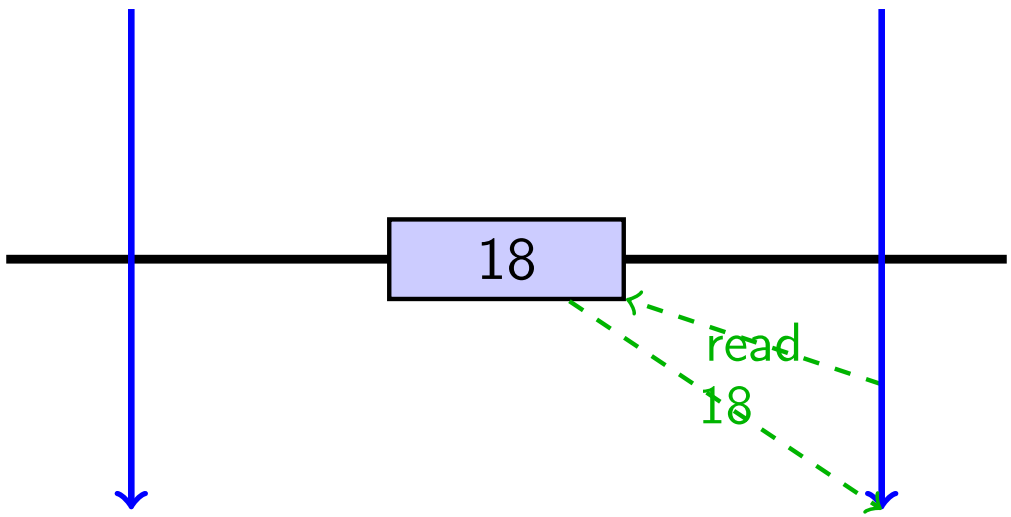
Synchronization

What is a Memory Fence? An ordering restriction for memory access.



Synchronization

What is a Memory Fence? An ordering restriction for memory access.



Synchronization between Groups

Golden Rule:

Results of the algorithm must be independent of the order in which work groups are executed.

Synchronization between Groups

Golden Rule:

Results of the algorithm must be independent of the order in which work groups are executed.

Consequences:

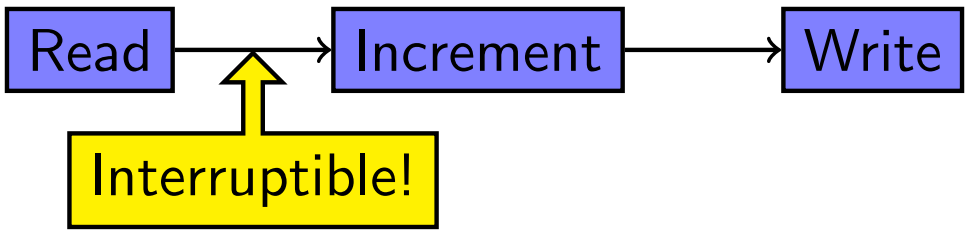
- Work groups may read the same information from global memory.
- But: Two work groups may not validly write different things to the same global memory.
- Kernel launch serves as
  - Global barrier
  - Global memory fence

Atomic Operations

Collaborative (inter-block) Global Memory Update:

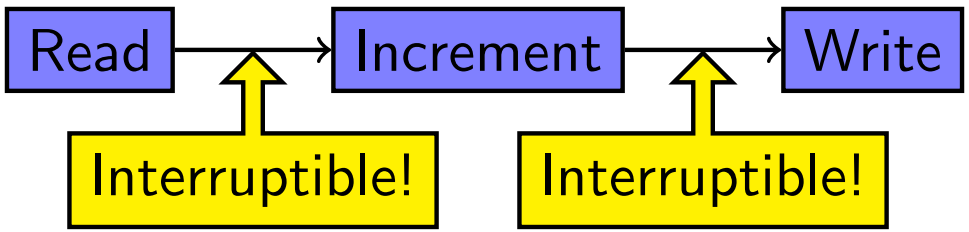
Atomic Operations

Collaborative (inter-block) Global Memory Update:



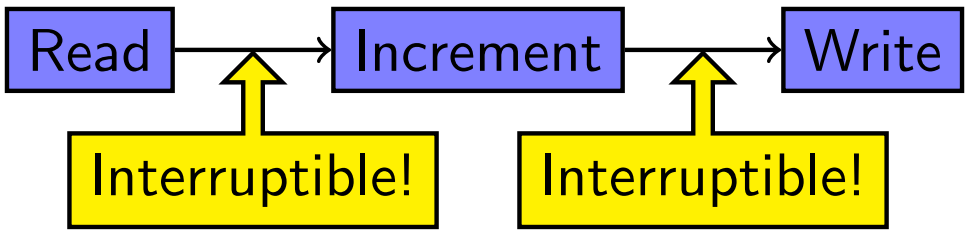
Atomic Operations

Collaborative (inter-block) Global Memory Update:

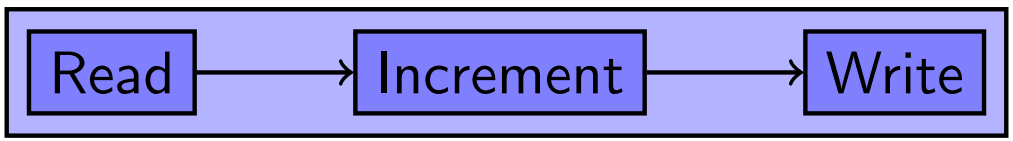


Atomic Operations

Collaborative (inter-block) Global Memory Update:

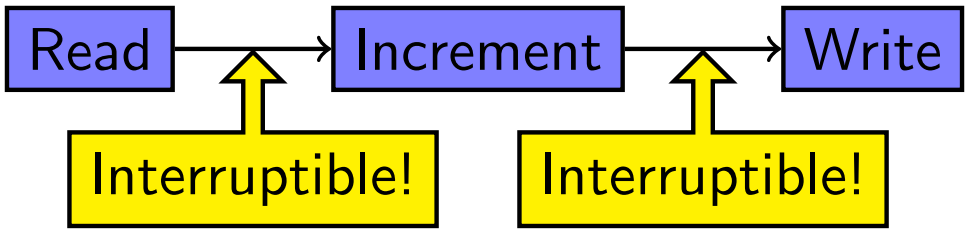


Atomic Global Memory Update:

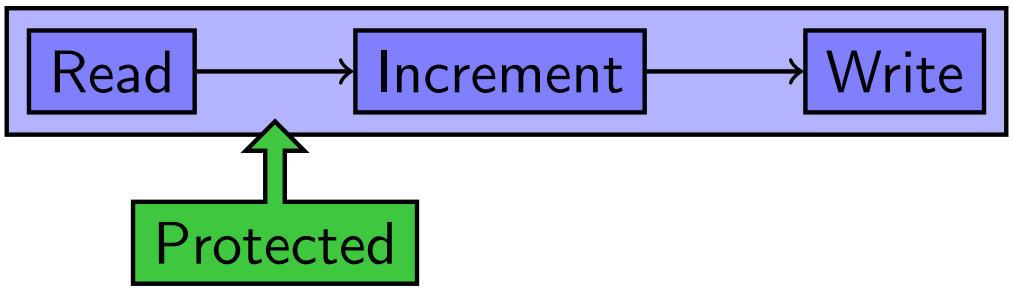


Atomic Operations

Collaborative (inter-block) Global Memory Update:

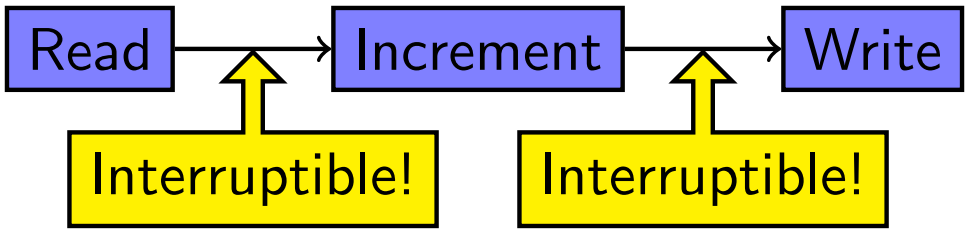


Atomic Global Memory Update:

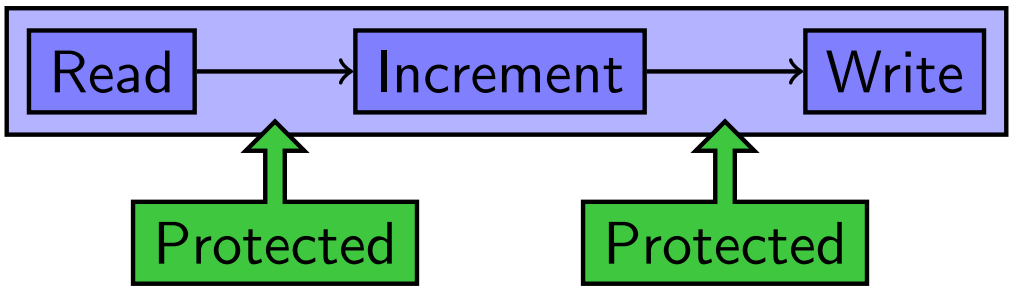


Atomic Operations

Collaborative (inter-block) Global Memory Update:

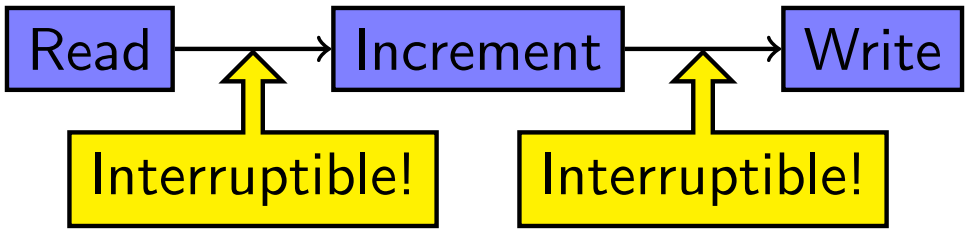


Atomic Global Memory Update:

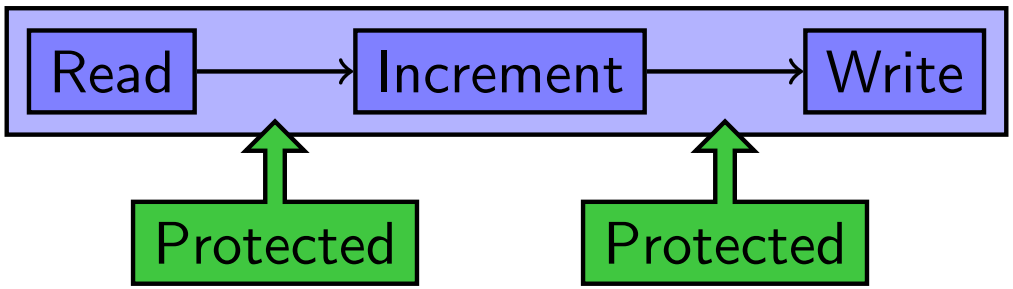


Atomic Operations

Collaborative (inter-block) Global Memory Update:



Atomic Global Memory Update:



How?

atomic_{add,inc,cmpxchg,...}(int *global, int value);

Atomic: Compare-and-swap

```
int atomic_cmpxchg (__global int *p, int cmp, int val)
int atomic_cmpxchg (__local int *p, int cmp, int val)
```

Does:

- Read the 32-bit value (referred to as old) stored at location pointed by p.
- Compute `(old == cmp) ? val : old`.
- Store result at location pointed by p.
- Returns old.

Atomic: Compare-and-swap

```
int atomic_cmpxchg (__global int *p, int cmp, int val)
int atomic_cmpxchg (__local int *p, int cmp, int val)
```

Does:

- Read the 32-bit value (referred to as old) stored at location pointed by p.
- Compute `(old == cmp) ? val : old`.
- Store result at location pointed by p.
- Returns old.

Implement atomic `float` add?

Questions?

?

Image Credits

- Isaiah die shot: VIA Technologies
- Onions: flickr.com/darwinbell CC