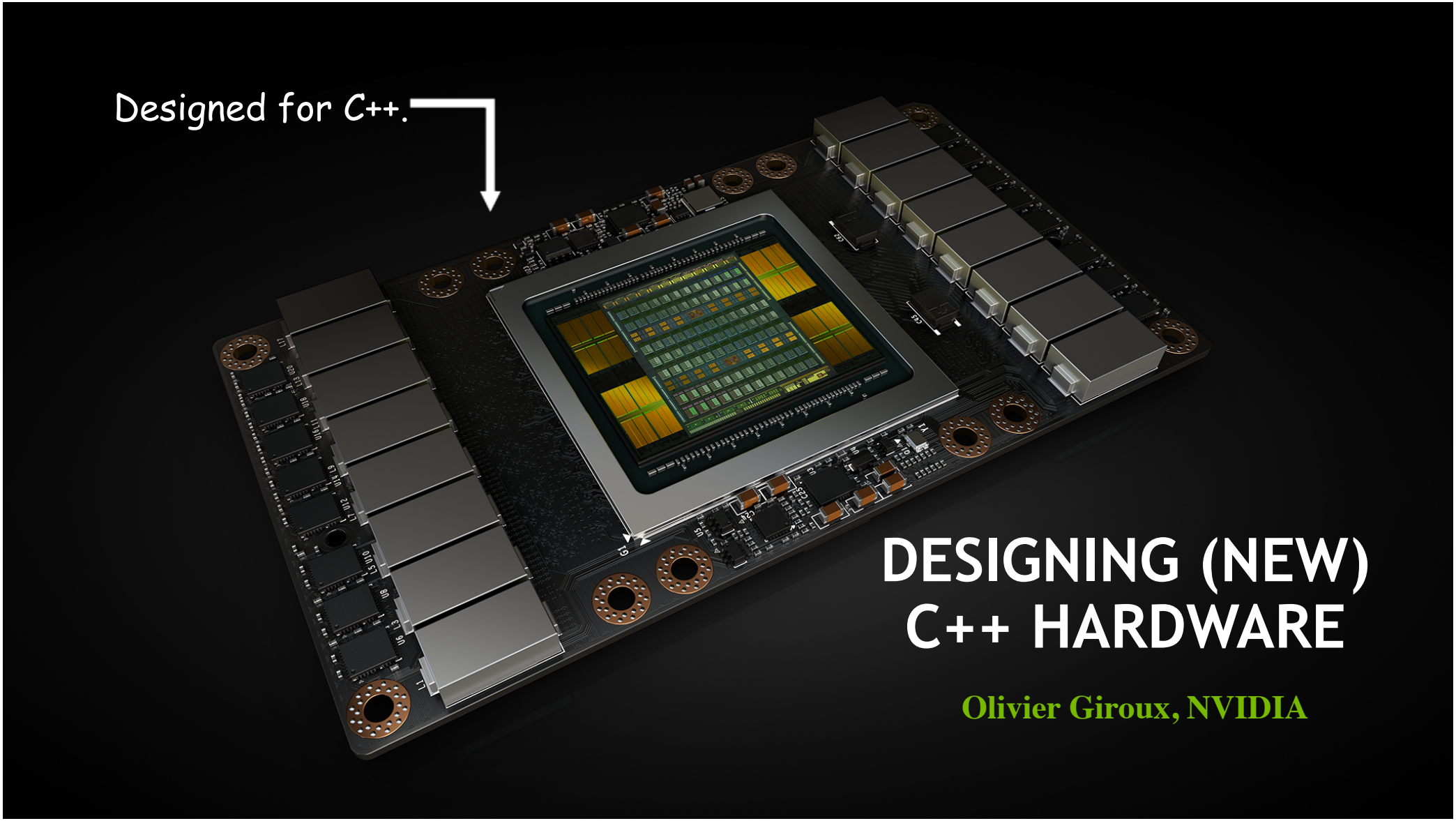
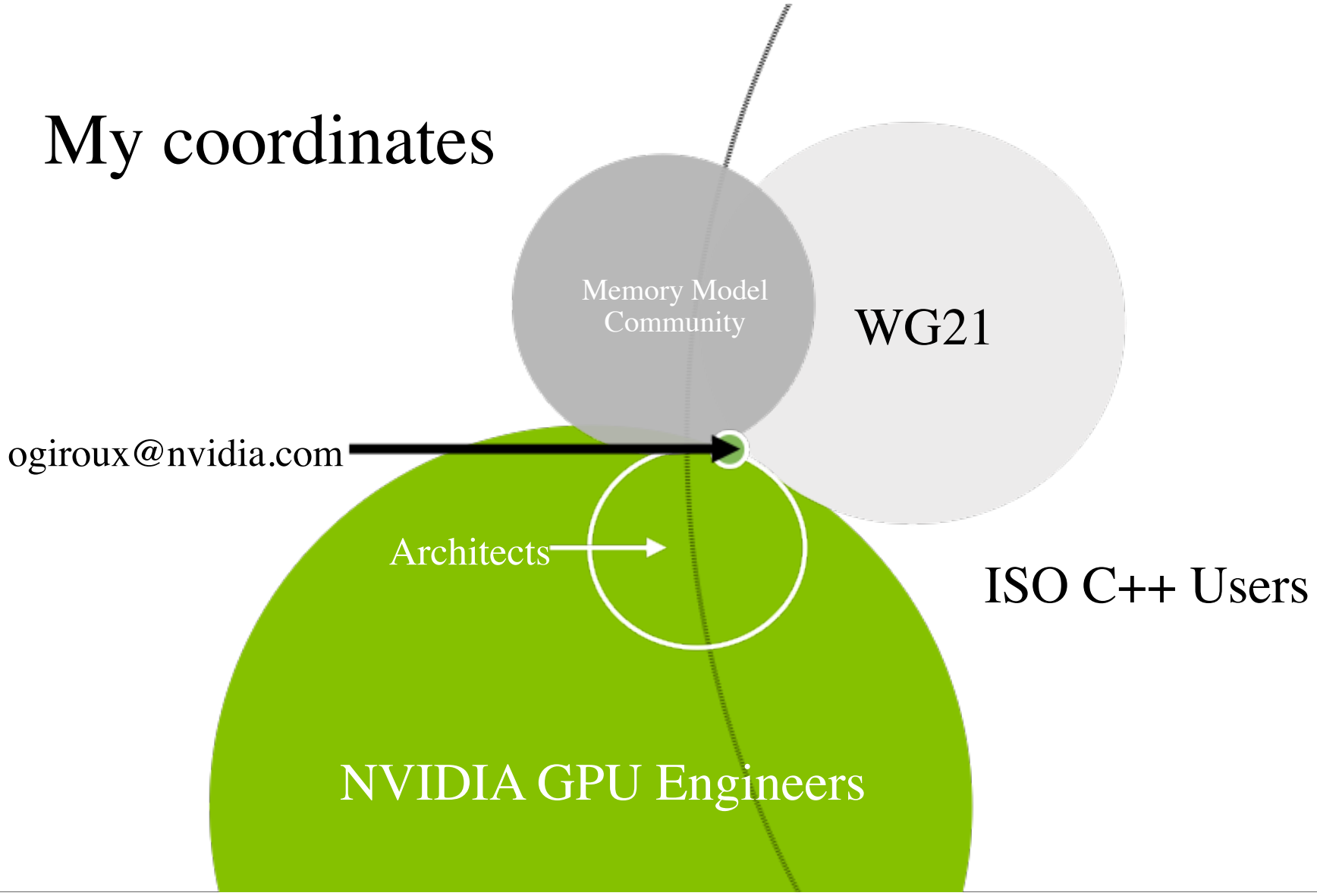

Designed for C++.
DESIGNING (NEW)
C++ HARDWARE
Olivier Giroux, NVIDIA

My coordinates



AGENDA

Not code

Not code

Not code

Bad code

Not code

What makes C++ portable?

WHAT MAKES C++ PORTABLE?

SUPPORT IN C++
Non-8-bit `char`
Noncommittal `sizeof`
Non-2's comp. `int`
Non-IEEE `float`
Non-endian addressing
Aligned addressing
Segmented memory

WHAT MAKES C++ PORTABLE?

SUPPORT IN C++	STORAGE
Non-8-bit `char`	✅
Noncommittal `sizeof`	✅
Non-2's comp. `int`	✅
Non-IEEE `float`	✅
Non-endian addressing	✅
Aligned addressing	✅
Segmented memory	✅

WHAT MAKES C++ PORTABLE?

SUPPORT IN C++	STORAGE	ARITHMETIC (UB)
Non-8-bit `char`	✅	✅
Noncommittal `sizeof`	✅	✅
Non-2's comp. `int`	✅	✅
Non-IEEE `float`	✅	✅
Non-endian addressing	✅	✅
Aligned addressing	✅	✅
Segmented memory	✅	✅

WHAT MAKES C++ PORTABLE?

SUPPORT IN C++	STORAGE	ARITHMETIC (UB)	20TH CENTURY TRIVIAL PURSUIT
Non-8-bit `char`	✅	✅	✅
Noncommittal `sizeof`	✅	✅	✅
Non-2's comp. `int`	✅	✅	✅
Non-IEEE `float`	✅	✅	✅
Non-endian addressing	✅	✅	✅
Aligned addressing	✅	✅	Useful.
Segmented memory	✅	✅	Useful.

WHY IS MOST OF THIS NOT HELPFUL?

FALSE CHOICES

Most options are **dictated**.

New CPU? Match AARCH64.

New GPU? Match the host.

GPUs match *all the hosts*.

BAD CHOICES

Most alternatives are **bad**.

Negligible area savings.

Negligible power savings.

Programmer surprise.

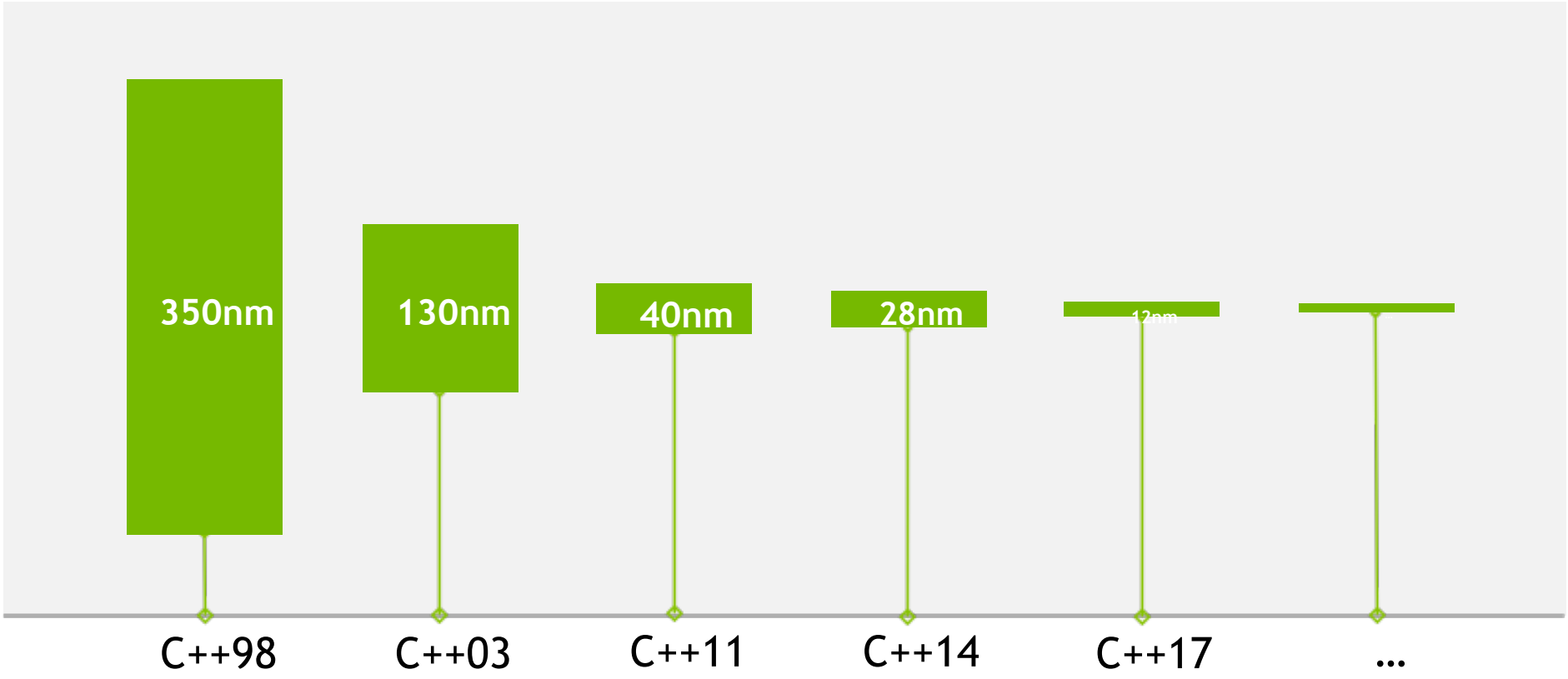
IT'S TOO RISKY IN 2017

NVIDIA TAPE-OUTS *versus* C++ REVISIONS

C++ implementations go all the way down to atoms.

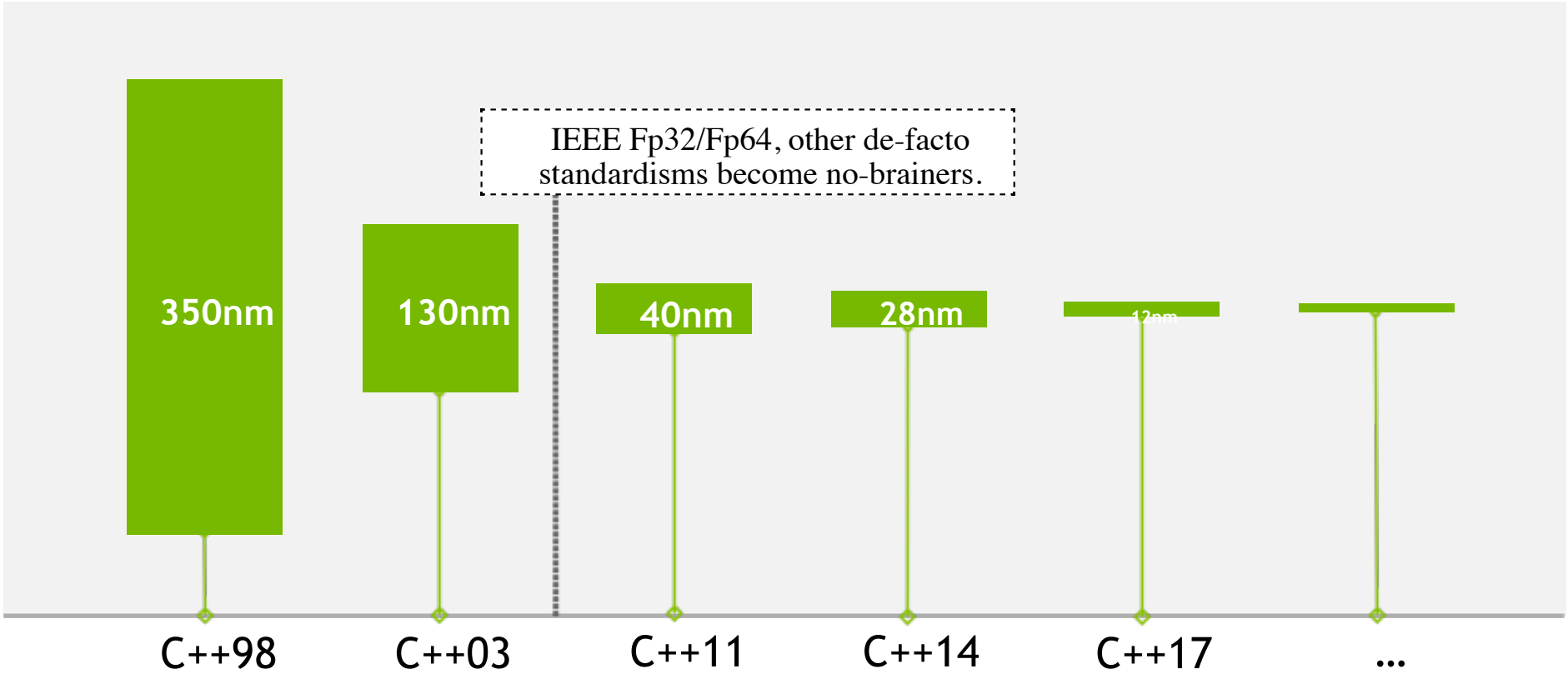
C++98 C++03 C++11 C++14 C++17 …

NVIDIA TAPE-OUTS *versus* C++ REVISIONS



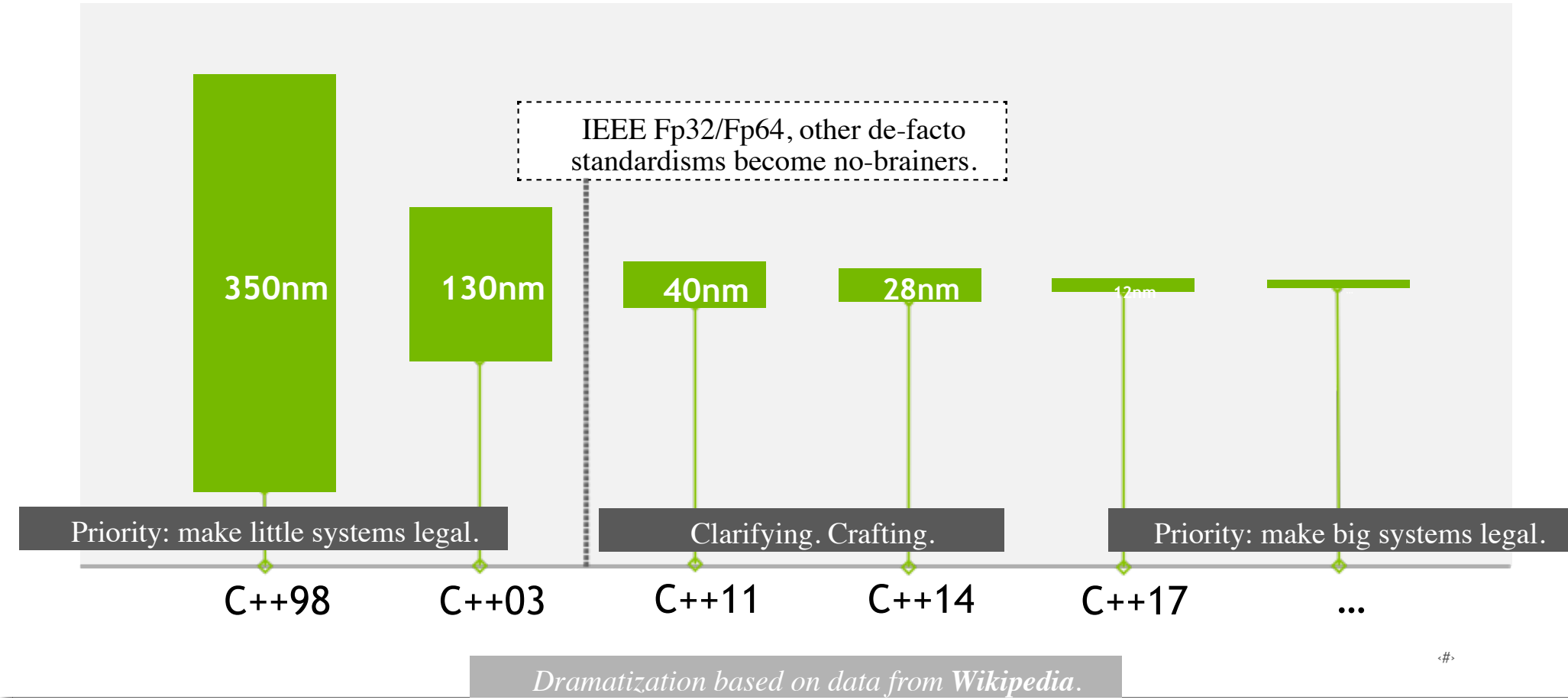
Dramatization based on data from ***Wikipedia****.*

NVIDIA TAPE-OUTS *versus* C++ REVISIONS



*Dramatization based on data from **Wikipedia**.*

NVIDIA TAPE-OUTS *versus* C++ REVISIONS



*Dramatization based on data from **Wikipedia**.*

TOWARDS NON-‘CLASSIC’ PORTABILITY

Then:

TRANSISTOR SCARCITY
Preexisting systems

SMALL C++ SYSTEMS
Not going take away

Now:

TRANSISTOR ABUNDANCE
Preexisting language

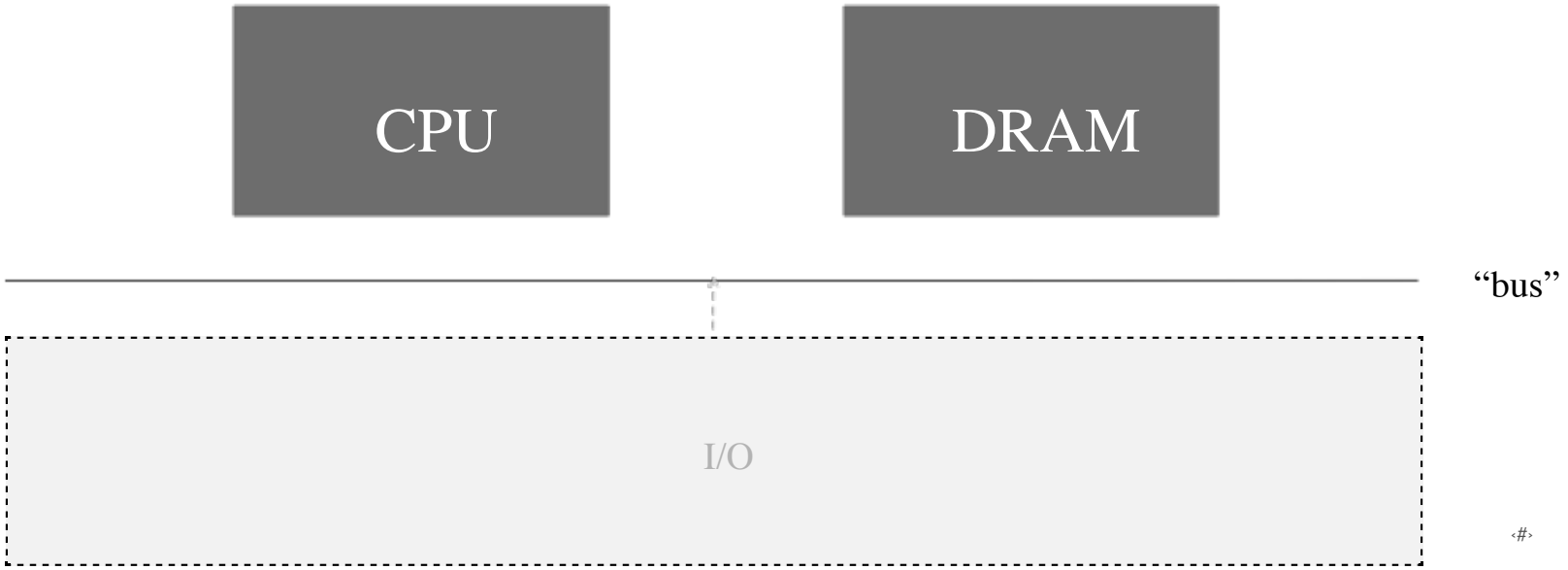
BIG C++ SYSTEMS
Extended in C++17

What will make C++-next portable?

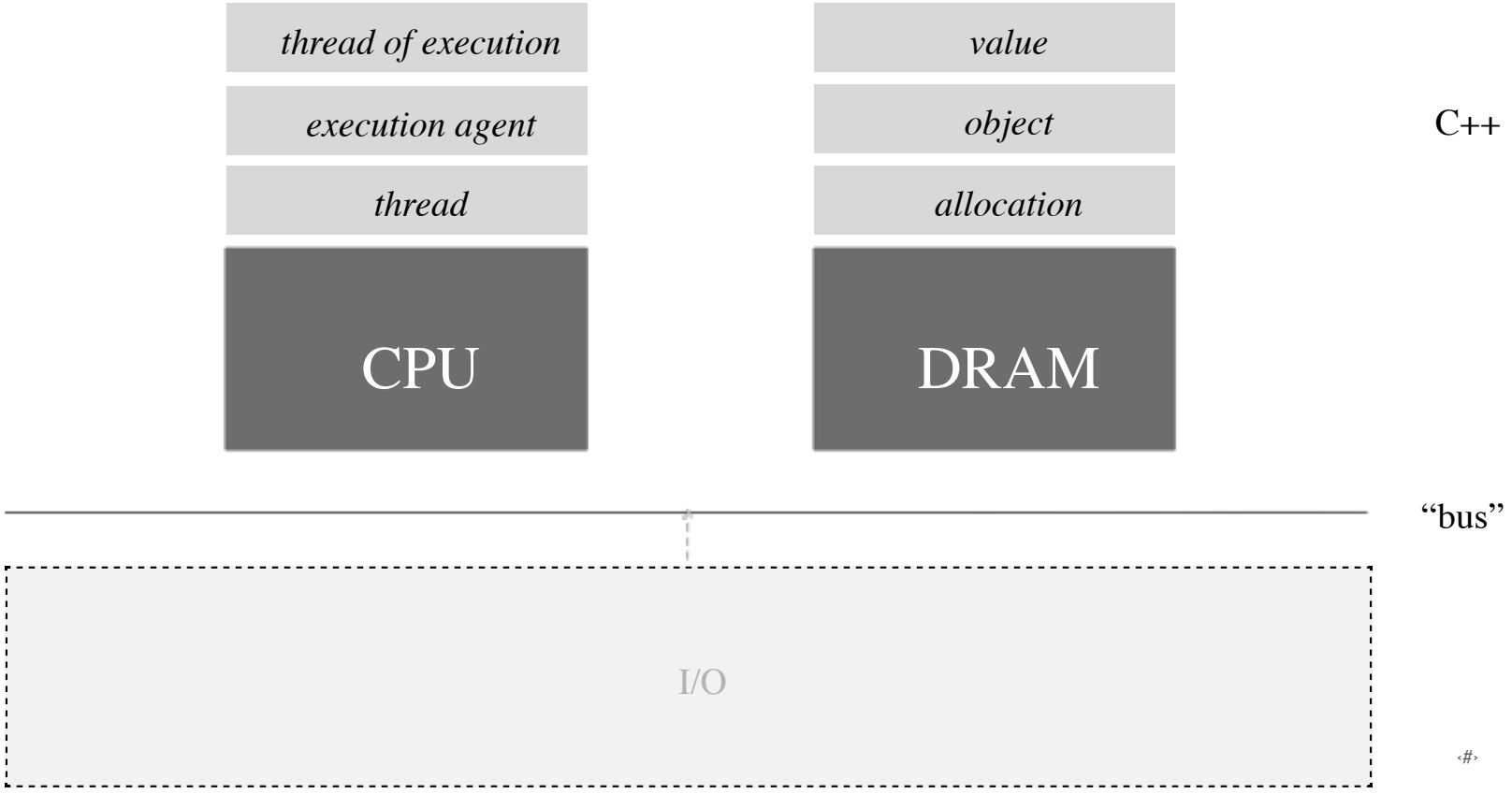
TOPOLOGY

1980’S TOPOLOGY

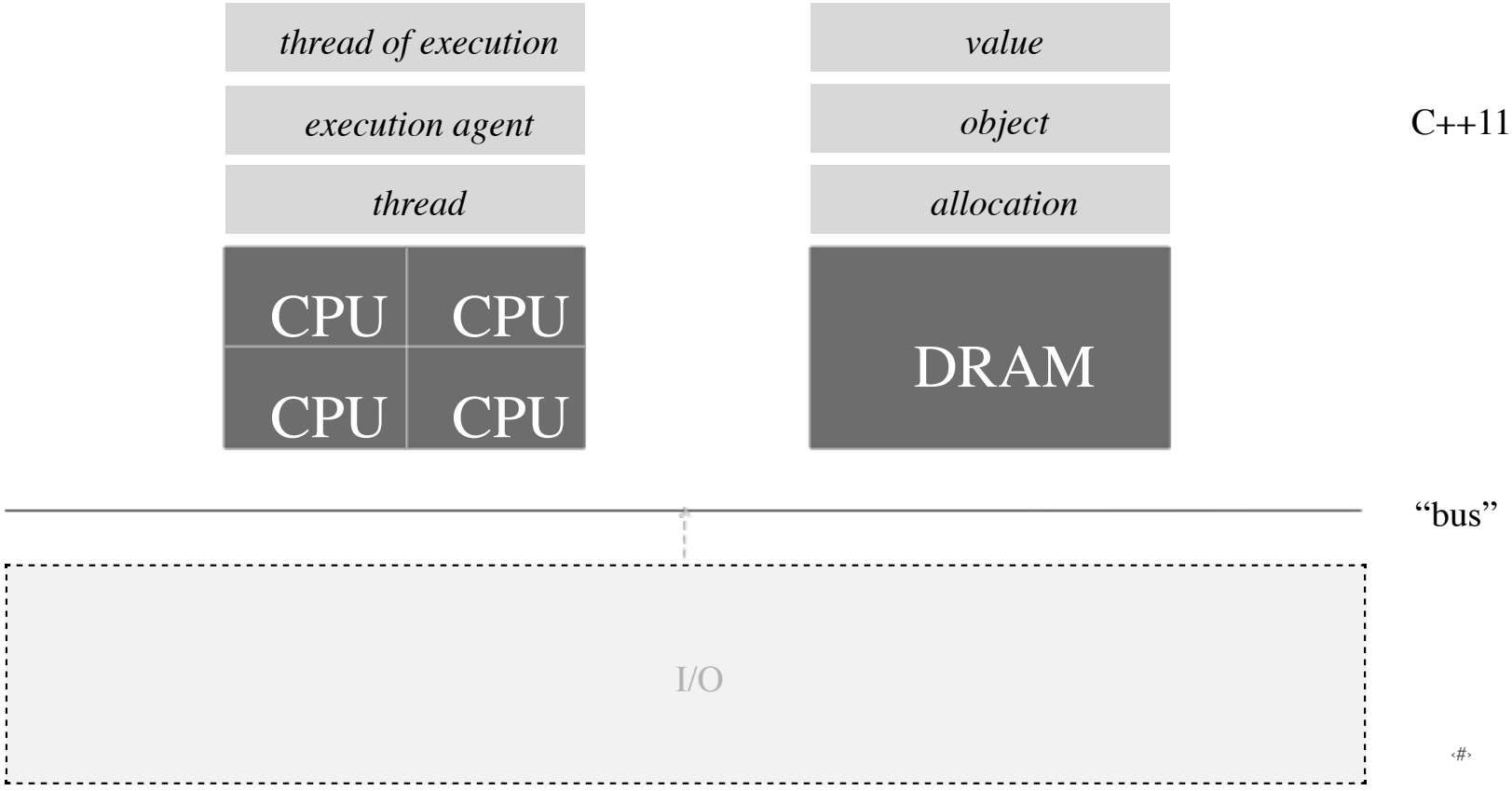
1980’S TOPOLOGY



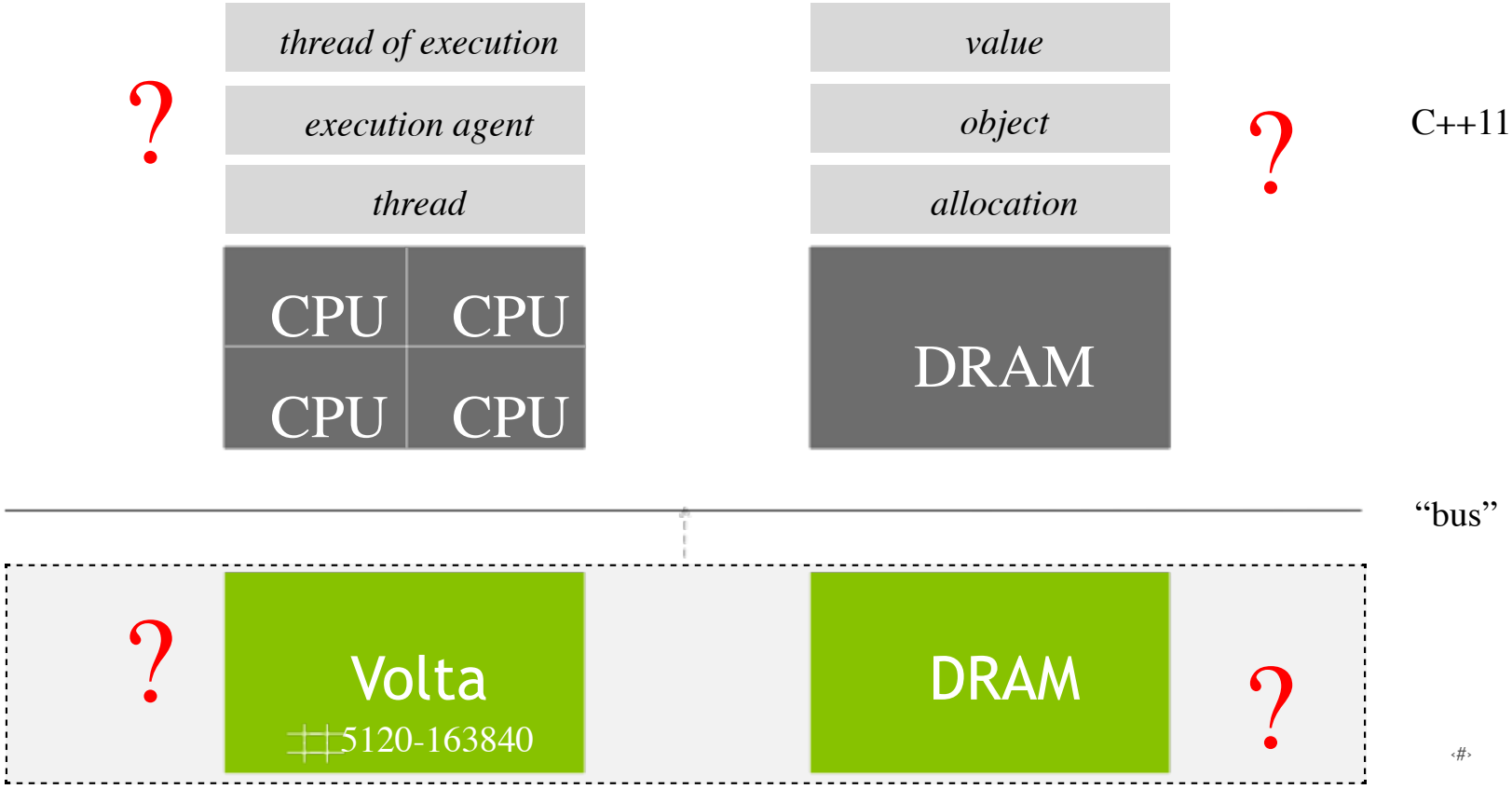
1980’S TOPOLOGY



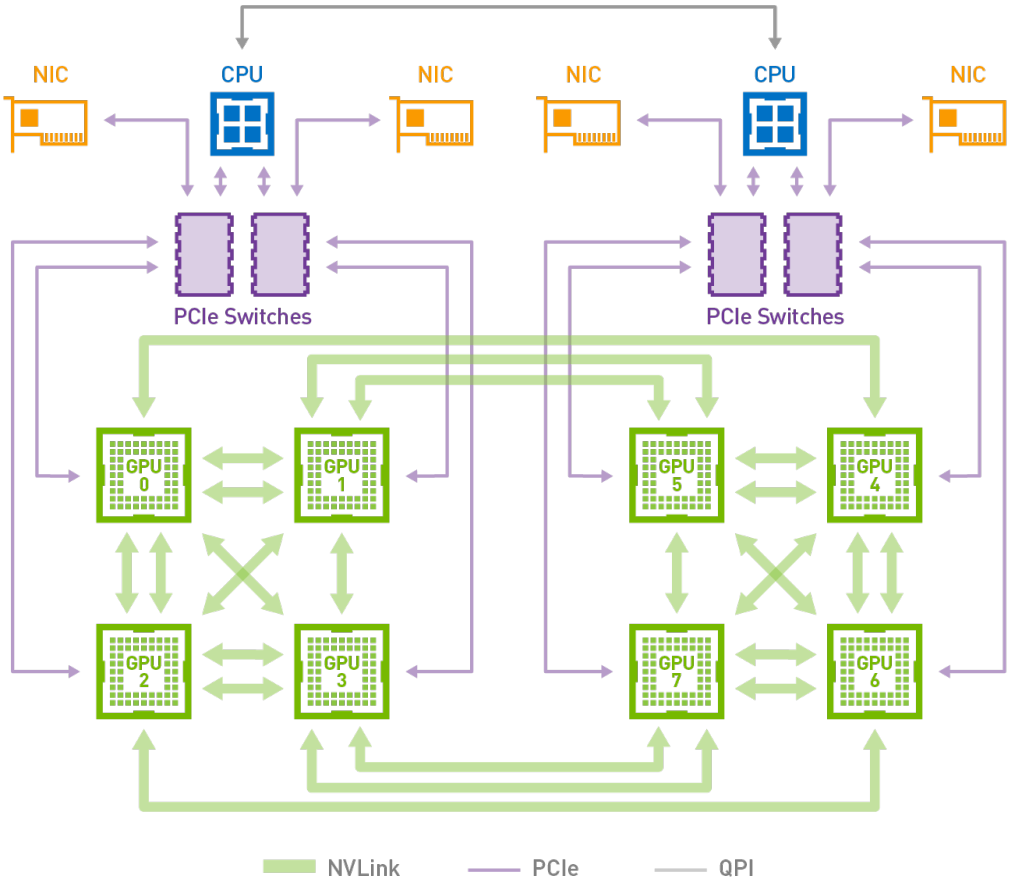
1990-2000'S TOPOLOGY



2010’S TOPOLOGY



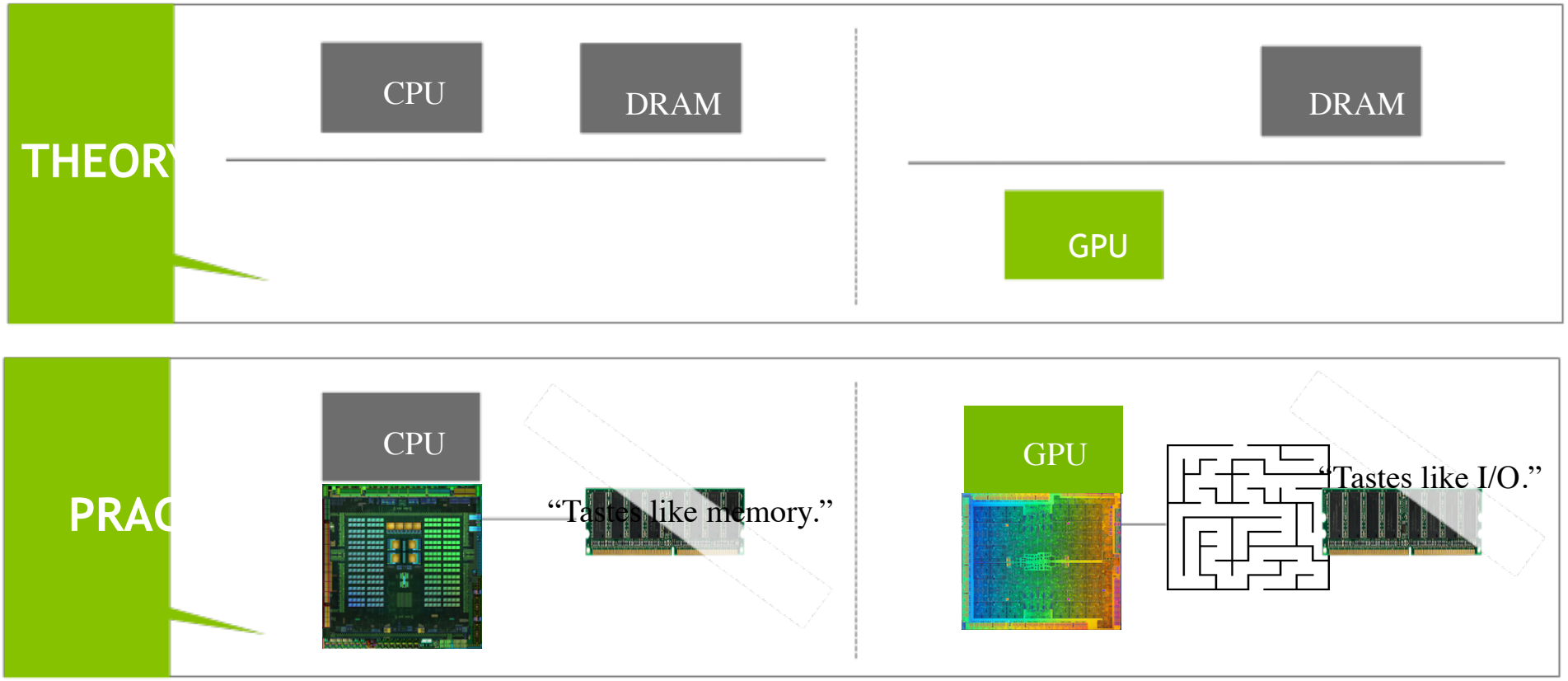
SHOW ❤ FOR TOPOLOGY



Designing Volta for C++ memory

COHERENCY

COHERENCY

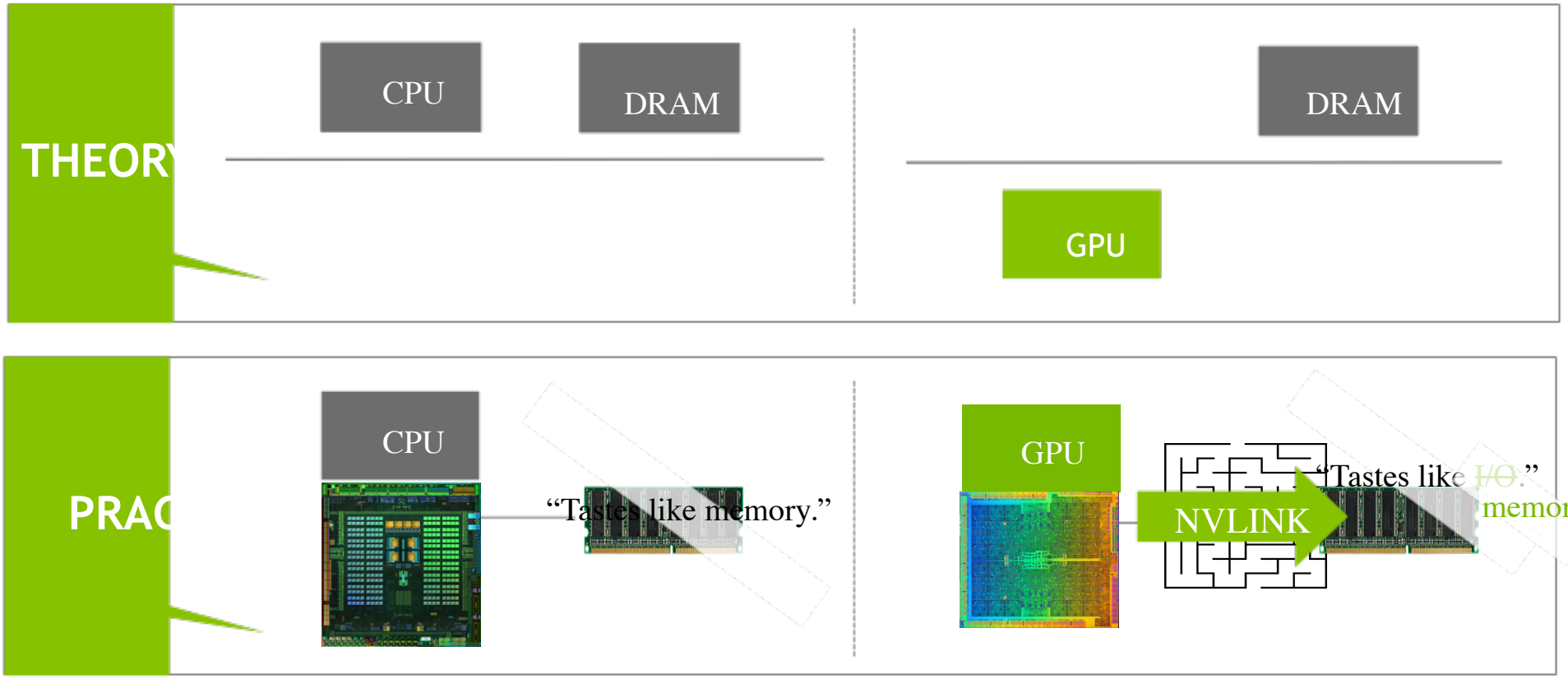

THEORY
CPU
DRAM
DRAM
GPU
PRAC
CPU
“Tastes like memory.”
GPU
“Tastes like I/O.”

COHERENCY

ARCHITECTURE	CUDA	X86	ARM & POWER
Tesla & Fermi	1+	`cudaMalloc` & `cudaMemcpy`	
Kepler & Maxwell	6+	`cudaMallocManaged` + symmetric heap	
Pascal & Volta	8+	`cudaMallocManaged` + paging	`cudaMallocManaged` (NVLINK)
	Linux HMM	`malloc` + paging	`malloc` (NVLINK)

"Tastes like memory."

COHERENCY


THEORY
CPU
DRAM
DRAM
GPU
PRAC
CPU
“Tastes like memory.”
GPU
NVLINK
“Tastes like I/O.”
memory

COHERENCY

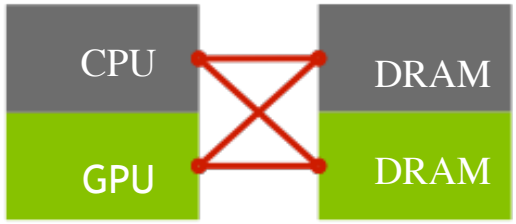
ARCHITECTURE	CUDA	X86	ARM & POWER
Tesla & Fermi	1+	`cudaMalloc` & `cudaMemcpy`	
Kepler & Maxwell	6+	`cudaMallocManaged` + symmetric heap	
Pascal & Volta "Tastes like memory."	8+	`cudaMallocManaged` + paging	`cudaMallocManaged` + NVLINK
	Linux HMM	`malloc` + paging	`malloc` + NVLINK

CONSISTENCY FOR CPUS

:

C++11	load	store	exchange	fence
(not atomic)	✅	✅	-	-
relaxed	✅	✅	✅	-
consume	🔻	-	🔻	-
acquire	✅	-	✅	✅
release	-	✅	✅	✅
acq_rel	-	-	✅	✅
seq_cst	✅	✅	✅	✅

CONSISTENCY FOR VOLTA = SAME



C++11	load	store	exchange	fence
(not atomic)	✅	✅	-	-
relaxed	✅	✅	✅	-
consume	🔻	-	🔻	-
acquire	✅	-	✅	✅
release	-	✅	✅	✅
acq_rel	-	-	✅	✅
seq_cst	✅	✅	✅	✅

COMPLETELY NEW memory model for Volta. Outline similar to POWER.

See the PTX 6.0 ISA programming guide, chapter 8.

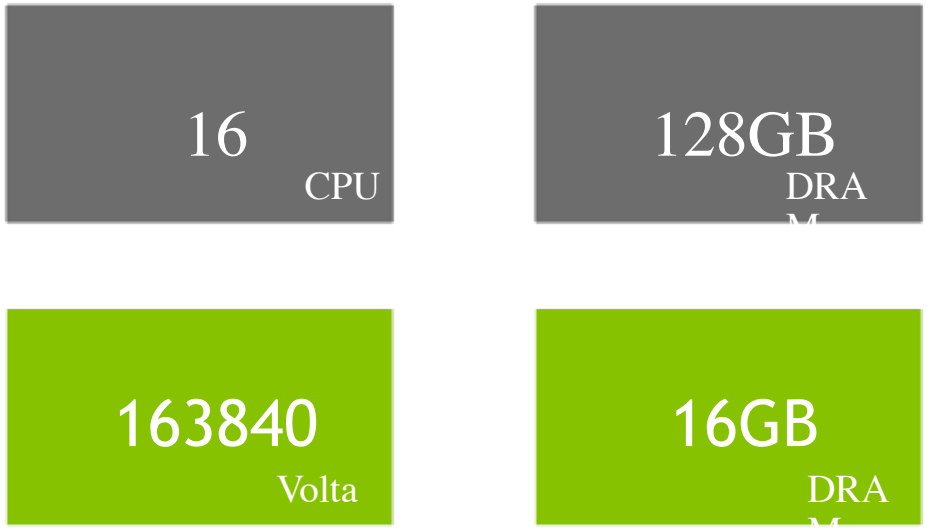
WE ❤️ THE MEMORY MODEL IN C++11

Good example of “what makes C++ portable”.

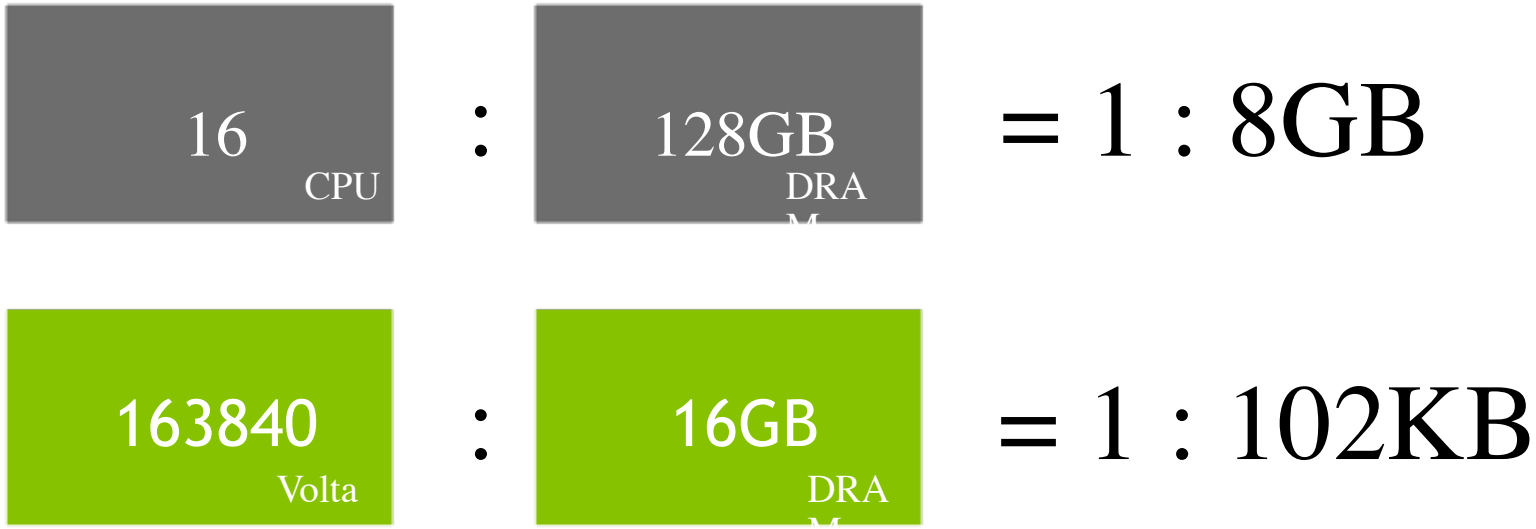
Not: `volatile`, was tyranny for us architects.

Open issues for C++ memory

FOOTPRINT



FOOTPRINT



FOOTPRINT

Shall we spend all this on `thread_local`? (y/N)

Whoever invented TLS didn't think this far ahead.

FOOTPRINT

That's a lot of stacks.

Forced to choose one:
performance, or
note 11 (4.7p1).

MEMORY PROGRESS REPORT

PROBLEM	GPU DISPOSITION	C++ DISPOSITION
Coherency	Pascal-Volta.	P0668.
Consistency	Volta, for assembly. (CUDA C++ TBD.)	
Footprint	-	P0072.

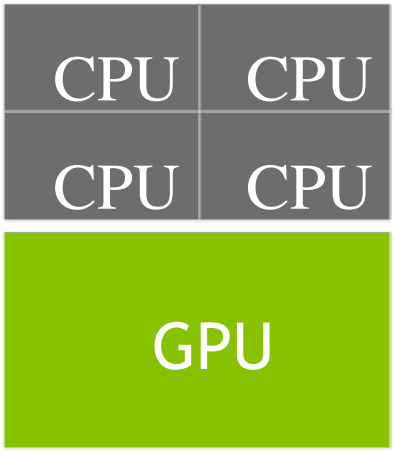
Designing Volta for C++ execution

PROGRESS

thread of execution

execution agent

thread



= A chain of evaluations in your code.

= A thing that runs your code.

= A particularly onerous example of that thing.

PROGRESS

thread of execution

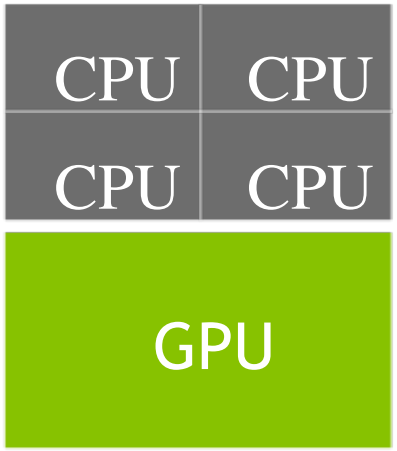
= A chain of evaluations in your code.

execution agent

= A thing that runs your code.

thread

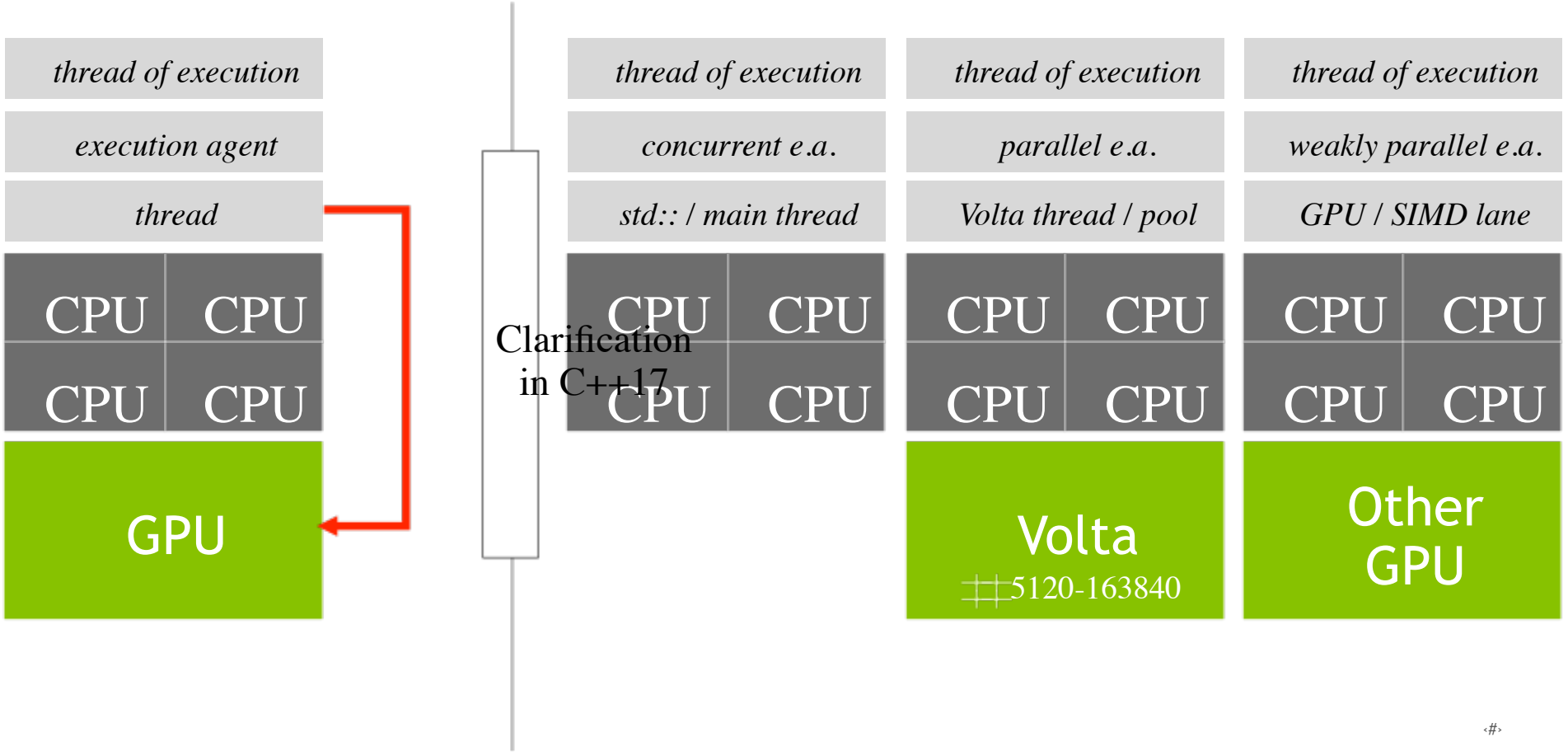
= A particularly onerous example of that thing.



= Runs anything.

= Runs things that aren't onerous.

PROGRESS

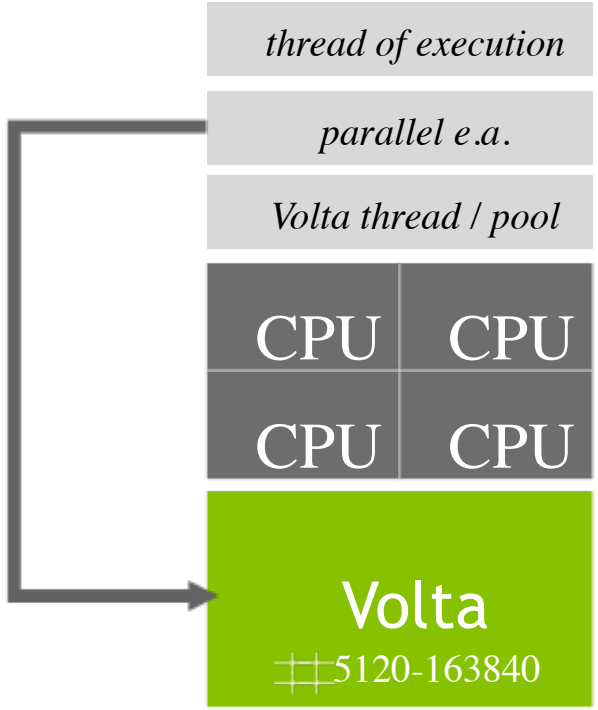

thread of execution
execution agent
thread
CPU
CPU
CPU
CPU
GPU
Clarification
in C++17
thread of execution
concurrent e.a.
std:: / main thread
CPU
CPU
CPU
CPU
thread of execution
parallel e.a.
Volta thread / pool
CPU
CPU
CPU
CPU
Volta
5120-163840
thread of execution
weakly parallel e.a.
GPU / SIMD lane
CPU
CPU
CPU
CPU
Other
GPU

PROGRESS

Not “business as usual”.

A concerted effort by dedicated engineers.

Volta is alone of its kind.



CLARIFYING SIMT C++

Sorry, CUDA `atomic<T>` isn't ready yet.

CLARIFYING SIMT C++

Sorry, CUDA `atomic<T>` isn't ready yet.

```
enum { unlocked = 0, locked = 1 };

//atomic<int> mutex = ATOMIC_VAR_INIT(unlocked);
volatile int mutex = unlocked;

void demo() {

  //int old;
  //while(!mutex.compare_exchange_strong(old=unlocked, locked))
  while(atomicCAS(&mutex,unlocked,locked)==locked)
    /* some kind of backoff here */
    ;

  /* Among throughput processors, only Volta is
     guaranteed to run this critical section. */
```

Open issues for C++ execution

STATE OF HETEROGENEITY

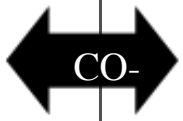
`__host__ __device__`

Short version: nothing to report today.

This (and **lack of compiler integration**) is behind most conformance issues.

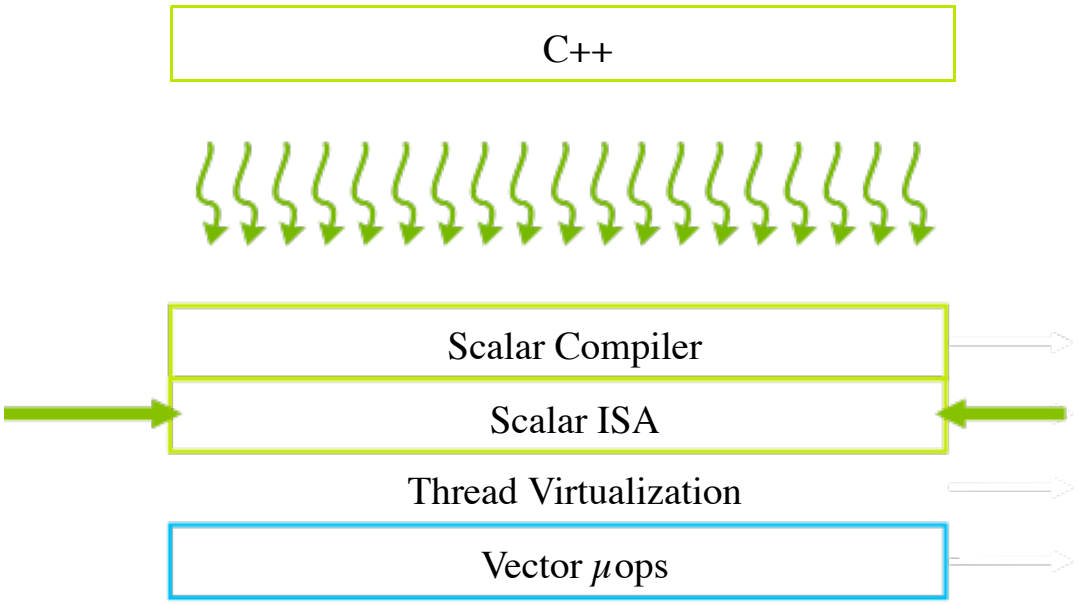
e.g.: no `std::vector::operator[]` or anything from std:: because of this.

EXECUTION PROGRESS REPORT

PROBLEM	GPU DISPOSITION	C++ DISPOSITION
Progress	Clarified in Volta. 	Clarified in C++17.
SIMT C++	Clarified in Volta.	-
Heterogeneity	TBD	

VOLTA C++ ARCHITECTURE WRAP-UP

VOLTA: MANY SCALAR C++ THREADS



Most of C++14 is in CUDA 9.0

Array of scalar threads

NVIDIA GPUs have always used


QUESTIONS.