CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Introduction to GPUs

CSCS Summer School 2025

Andreas Jocksch, Prashanth Kanduri, Radim Janalik & Ben Cumming

Course Overview

Over these two days we will cover a range of topics:

- Learn about the GPU memory model;
- Implement parallel CUDA kernels for simple linear algebra;
- Learn how to scale our parallel kernels to utilize all resources on the GPU;
- Learn about thread cooperation and synchronization;
- Learn how to profile GPU applications;
- Port the miniapp to the GPU.

Course Overview

We focus on HPC and GPU architectures, specifically:

- HPC development on heterogeneous systems (CPU + GPU)
- Understanding architecture concepts to enable software development
- Largely transferable mental models to GPUs from other vendors
- Focus on fundamentals, and key ideas to get started with GPU programming with CUDA as the framework of instruction

Course Overview

There aren't many prerequisites for the course:

- No GPU or graphics experience required.
- We assume C++11 knowledge.
- The generic GPU programming concepts from CUDA are useful for when:
  - Developing with OpenACC, OpenCL and GPU-ready libraries.
  - Using ML frameworks that use GPU for compute.

Today's topics

- Introduction to GPUs and their architecture.
- Comments on porting applications to GPUs.
- The Nvidia software platform.
- Writing kernels for GPUs.

Why GPUs?

First, what do we really want?

- Scientific results, as fast as possible.

First, what do we really want?

- Scientific results, as fast as possible.

What's stopping us?

- Processor compute speed
- Data movement
- Algorithmic efficiency

Increasing CPU Performance

There are 3 ways to increase performance on the CPU:

1. Increase the CPU clock speed (do every operation faster).

Increasing CPU Performance

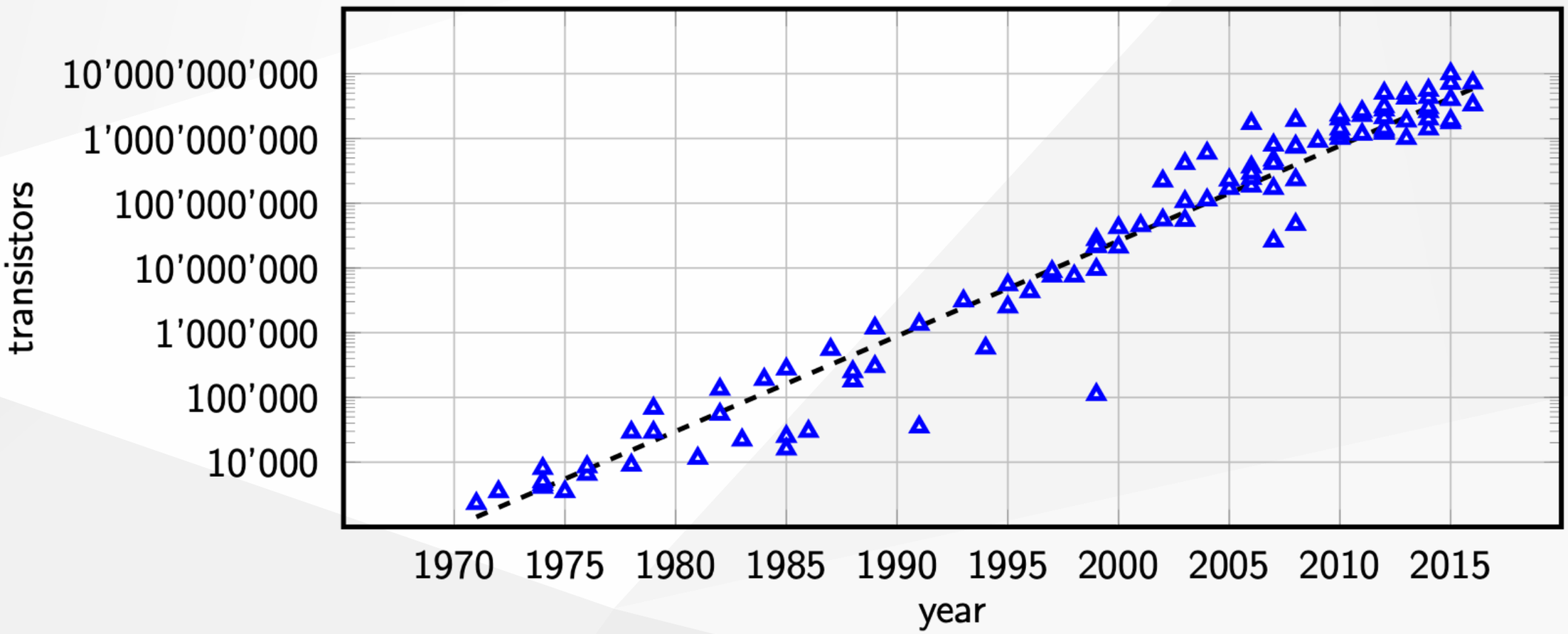
There are 3 ways to increase performance on the CPU:

1. Increase the CPU clock speed (do every operation faster).
2. Increase the number of operations per clock cycle:
    - vectorization;
    - instruction level parallelism;
    - more cores / multiple machines.

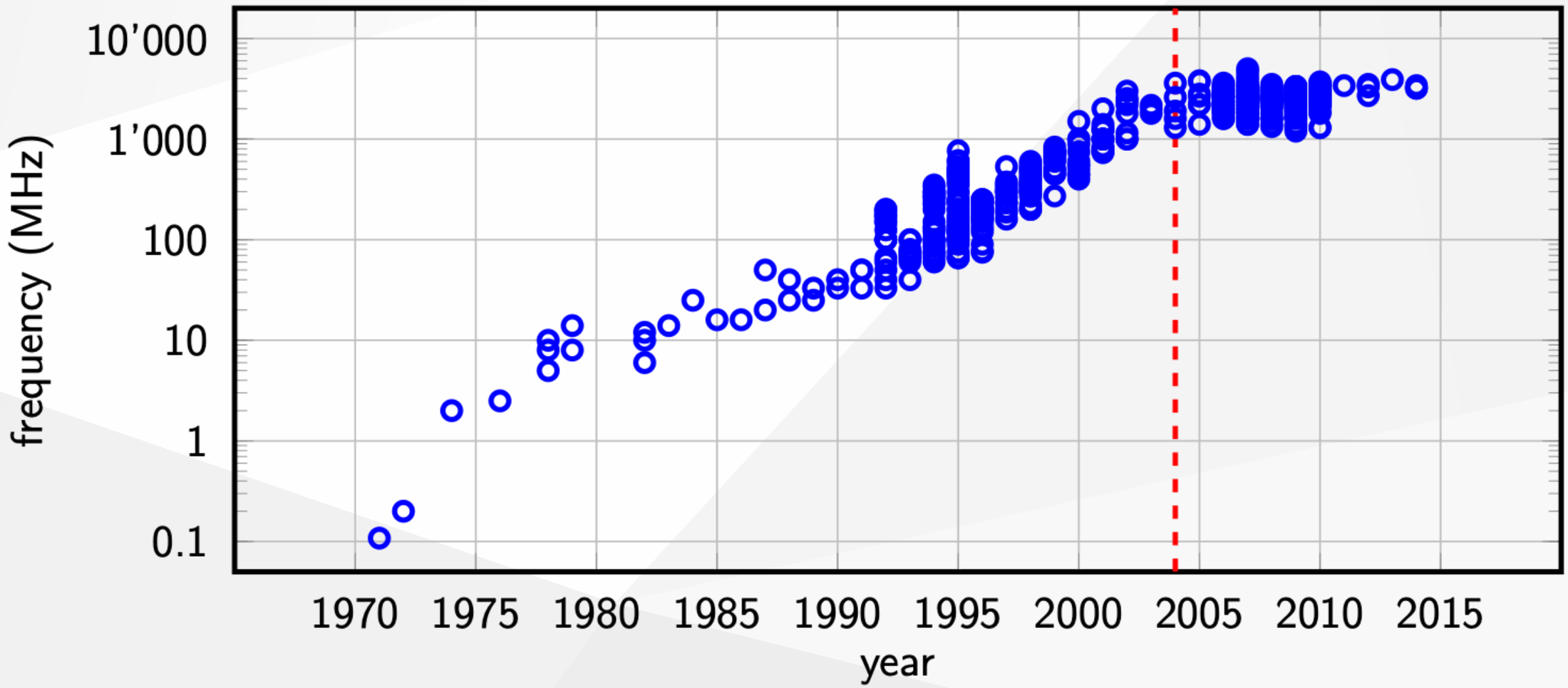
Increasing CPU Performance

There are 3 ways to increase performance on the CPU:

1. Increase the CPU clock speed (do every operation faster).
2. Increase the number of operations per clock cycle:
   - vectorization;
   - instruction level parallelism;
   - more cores / multiple machines.
3. Access/process data more efficiently:
   - Avoid stalling the processor while it waits for data to come from memory
     $\rightarrow$ Better use of cache
   - Avoid stalling the processor pipeline
     $\rightarrow$ Better use of branch prediction features.



The number of transistors in processors has increased exponentially for 45 years.



And clock speeds were, for a time, also increasing.

MPI and the Free Lunch

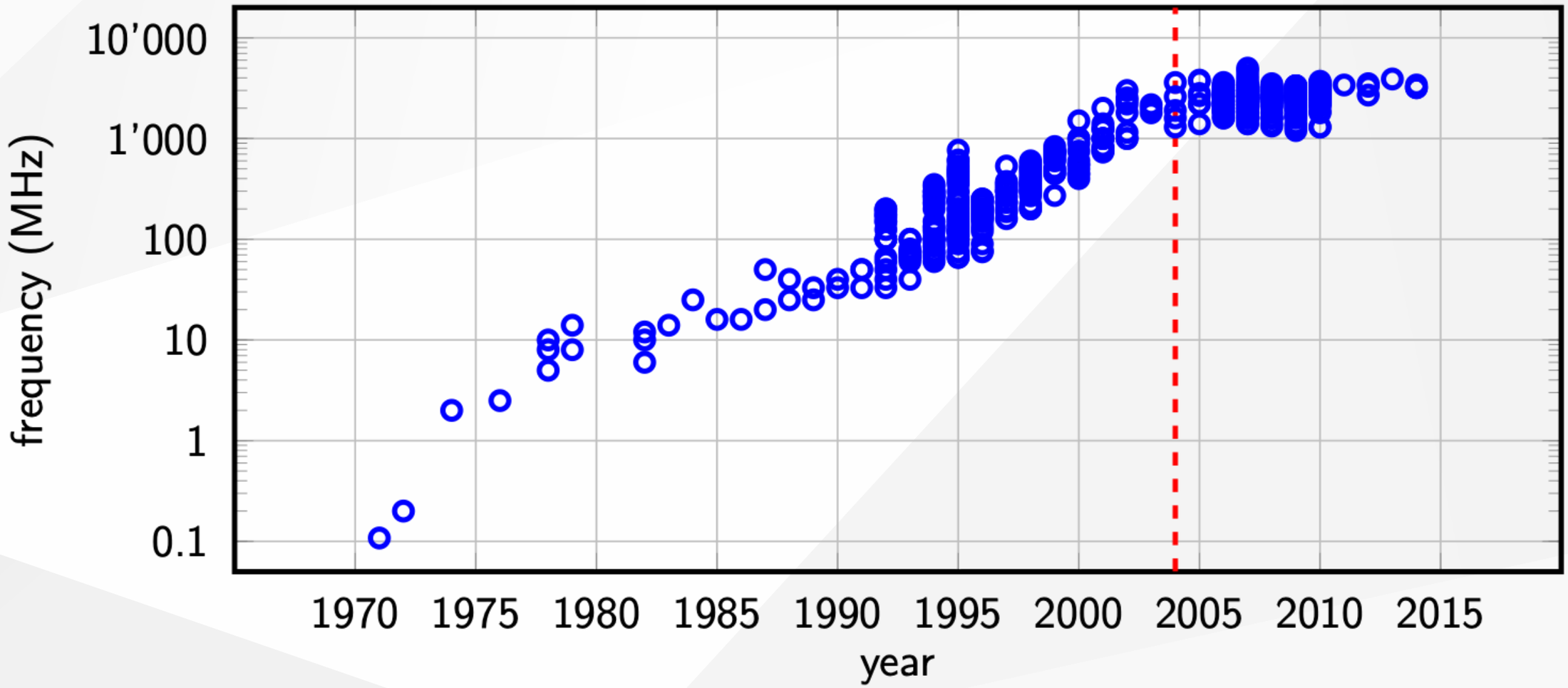
HPC applications were ported to use the message passing library MPI in the late 90s and early 2000s at great cost and effort

- Individual nodes with one or two CPUs
- Break problem into chunks/sub-domains
- Explicit message passing between sub-domains

The "free lunch" was the regular speedup in codes as CPU clock frequencies increased and as the number of nodes in systems increased

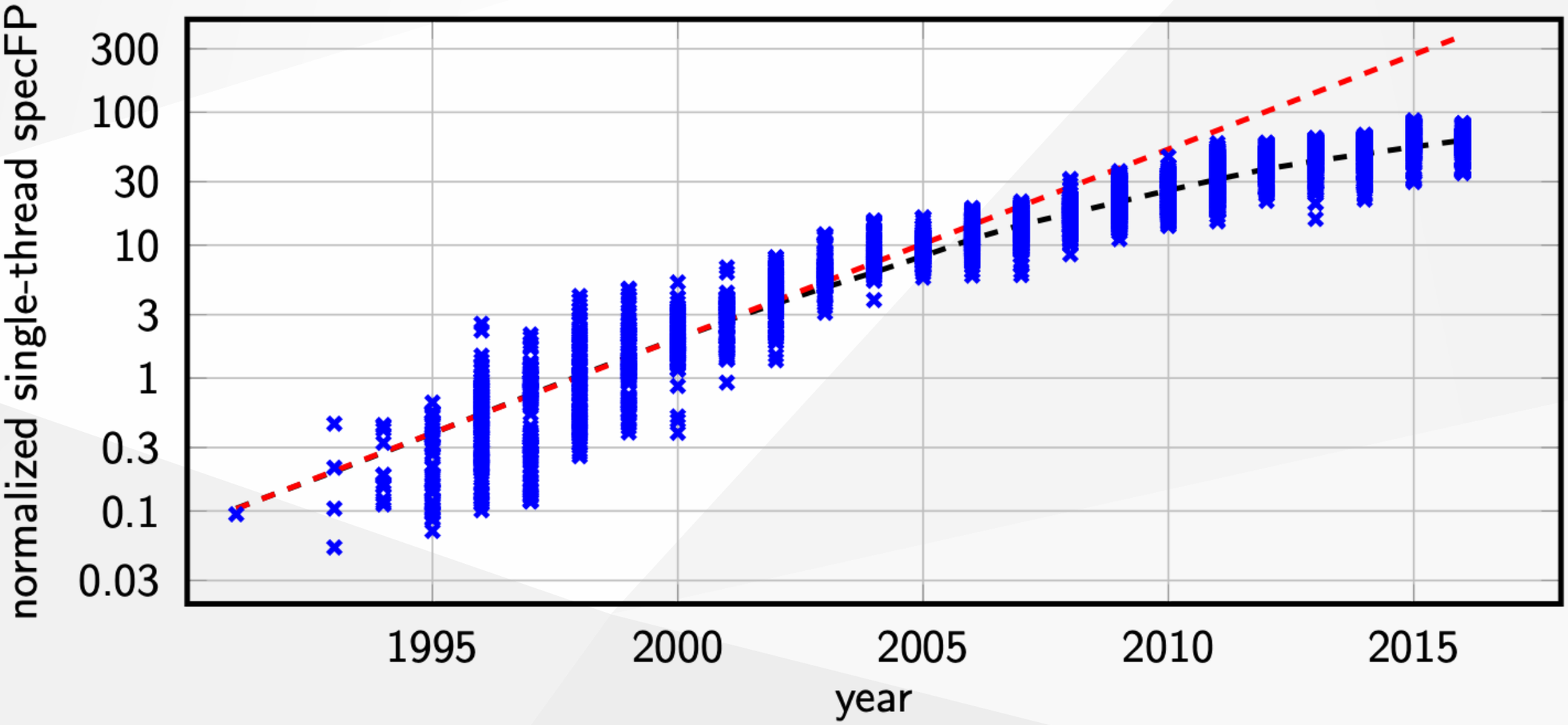
- With little/no effort, each new generation of processor bought significant speedups.

... but there is no such thing as a free lunch.



Clock speeds peaked around 2005.

- We can not build increasingly faster simple processors.
- $Power \sim Frequency^3$
- Higher frequency $\rightarrow$ much more power, more heat, more money, more difficult to engineer and build.



As a result, floating point performance per core is not keeping up...

One solution: More parallelism

Instead of scaling some base frequency like this:

$$P_{\text{tot}} \sim (k \cdot Frequency)^3,$$

we keep the base frequency fixed and increase the number of processors:

$$P_{\text{tot}} \sim N_{\text{cpu}} \times Frequency^3$$

In the best case, with N_{cpu} processors working in parallel, the job is done in $1/N_{\text{cpu}}$ time.

It takes the same power for a CPU to go 10x faster as it does for 1000 CPUs to run in parallel.

Clock Frequency WILL NOT Increase

In fact, clock frequencies have been going down as the number of cores increases:

- A 4-core Haswell processor at 3.5 GHz ($4 \times 3.5 = 14$ Gops/second) has the same power consumption as a 12-core Haswell at 2.6 GHz ($12 \times 2.6 = 31$ Gops/second);
- A P100 GPU with 3584 CUDA cores runs at 1.1 GHz.
- A H100 GPU with 18432 CUDA cores runs at 1.1 GHz.

Caveat

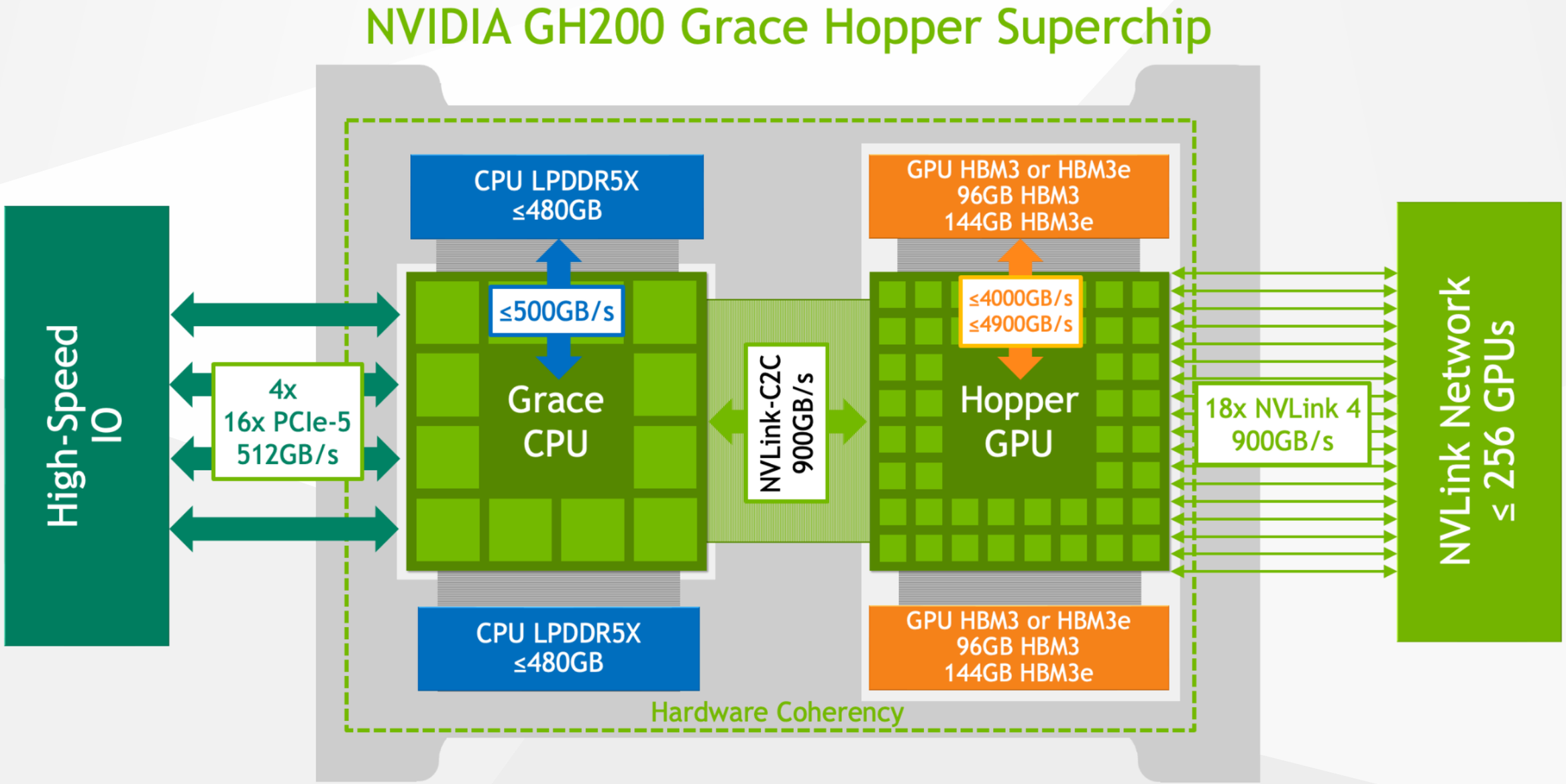
It is not reasonable to compare a CUDA core and an X86 core.

Parallelism WILL Increase

- The number of cores in both CPUs and accelerators will continue to increase
- The width of vector lanes in CPUs will increase
  - 8×SIMD double for AVX512 and SVE (Intel and ARM).
- The number of threads per core will increase
  - Intel SkyLake: 2 threads/core
  - Intel KNL: 4 threads/core
  - IBM Power-8: 8 threads/core

There is a trend towards more parallelism "on node"
Multi-Core CPUs get more cores and wider vector lanes:
• 24-core×SMT 4×SIMD 128: IBM Power9 (2017); • 28-core×SMT 2×SIMD 512: Intel Xeon (2020) • 48-core×SMT 1×SIMD 512: Fujitsu ARM A64FX (2020) • 72-core×SMT 1×SIMD 256: Nvidia Grace ARM64 (2023)
Many-Core Accelerators with many highly-specialized cores and high-bandwidth memory:
• NVIDIA P100 GPUs with 3,584 cores (2016); • NVIDIA V100 GPUs with 5,120 cores (2017); • NVIDIA A100 GPUs with 8,192 cores (2020); • NVIDIA H100 GPUs with 18,432 cores (2023)

An Alps Module



There are 4 of these on a node... that's a lot of parallelism!

Memory is Slow

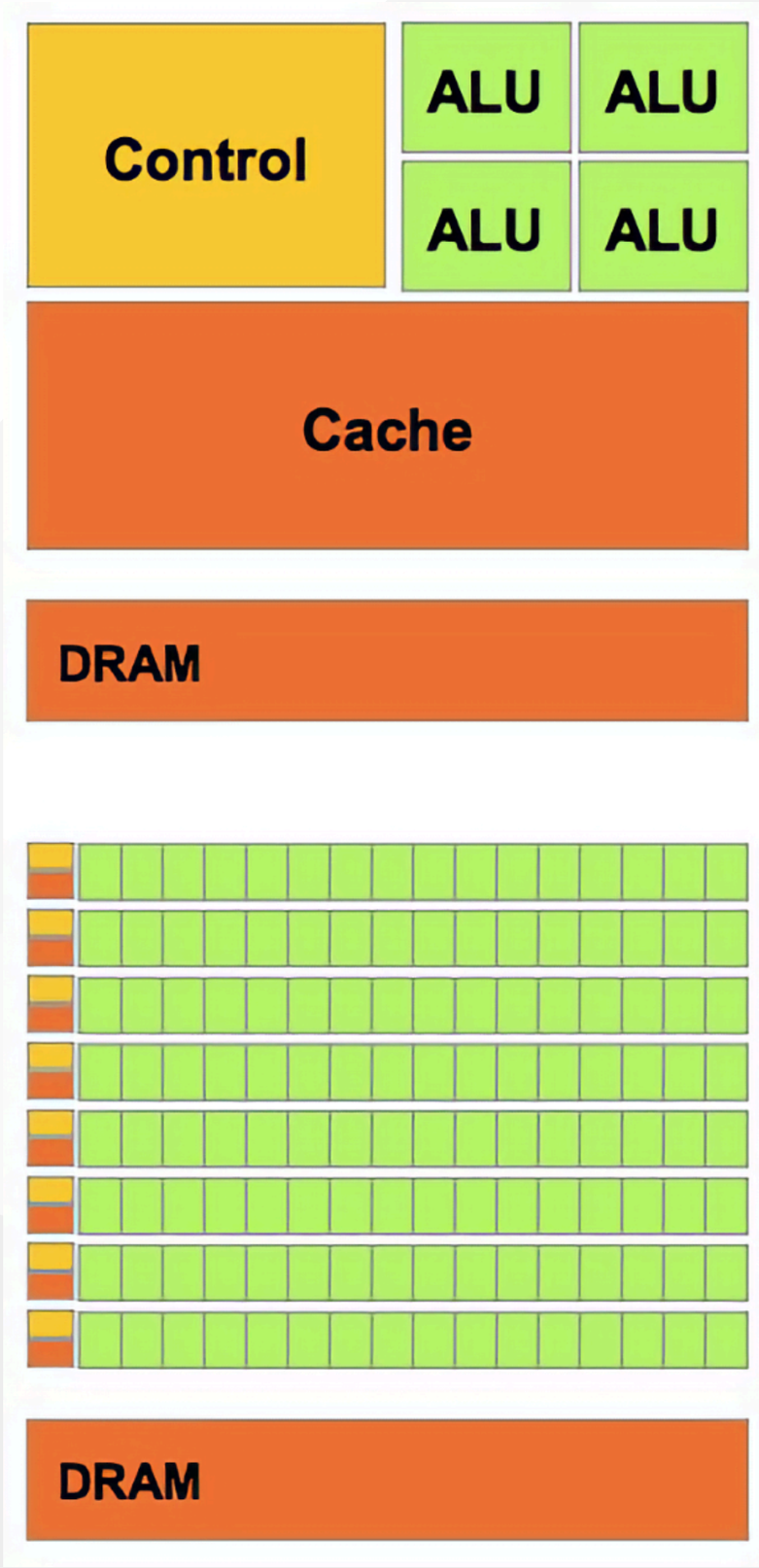
However, memory is now *much* slower than processors

- For both CPU and GPU the latency of fetching a cache-line from memory is 100s of cycles...
- ... 100s of cycles that the processor is stalled
- Latency has to be hidden or reduced to minimise stalling

Low-Latency or High-Throughput?

CPU

- Optimized for low-latency access to cached data sets.
- Control logic for out-of-order and speculative execution.

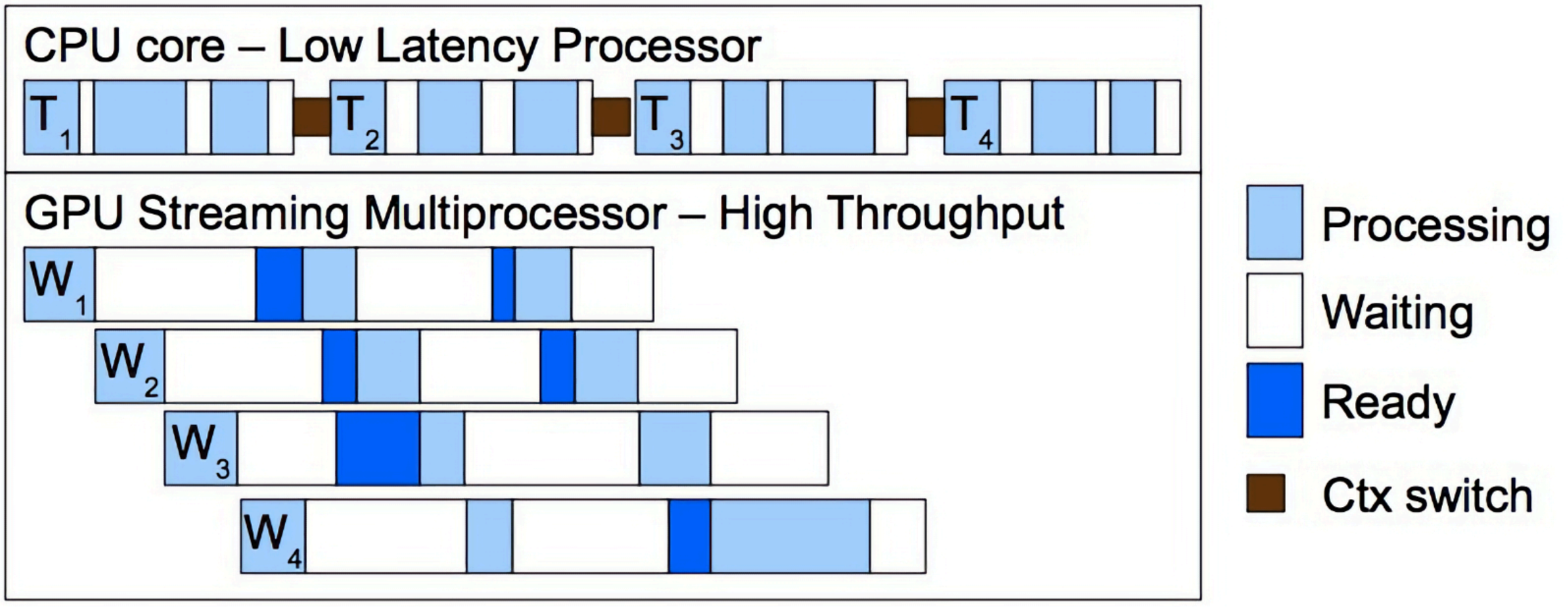


GPU

- Optimized for data-parallel, throughput computation.
- Architecture tolerant of memory latency.
- More transistors dedicated to computation.

GPUs are Throughput Devices

- CPU cores are optimized to minimize latency between operations
- GPUs aim to minimize latency between operations by scheduling multiple warps (thread bundles)



Many Applications aren't Designed for Many-Core

- Exposing sufficient fine-grained parallelism for multi and many core processors is hard.
- New programming models are required.
- New algorithms are required.
- Existing code has to be rewritten or refactored

On-node parallelism will continue to increase:

- Piz Daint @ CSCS (2015): 1 GPU + 1 CPU
- Marconi100 @ CINECA (2020): 4 GPU + 2 CPU
- EUROHPC pre-exascale (2021): 4 GPU + 1 CPU
- US ECP exascale (2021-2023): 4 GPU + 1 CPU
- **ALPS @ CSCS (2024): 4 GPU + 4 CPU**

TLDR: Energy Consumption Drove Change

Writing good concurrent code for many-core is diffcult

- But the days of easy speed up each generation of CPU are over
  - Performance gains must not increase power consumption
- This course will be about one type of many-core architecture NVIDIA GPUs
  - CUDA is GPU vendor-specific.
  - Conceptually very close to HIP, OpenCL, SYCL and Metal (AMD/NVIDIA/Intel/Apple)