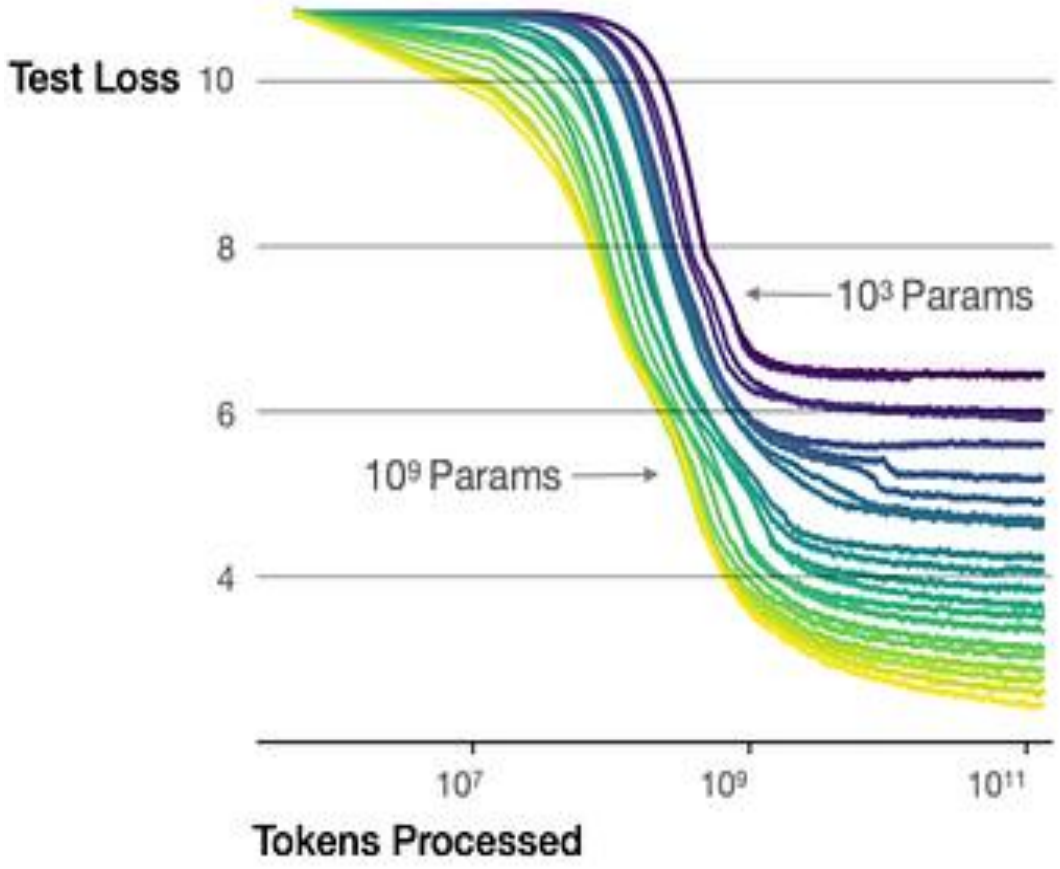
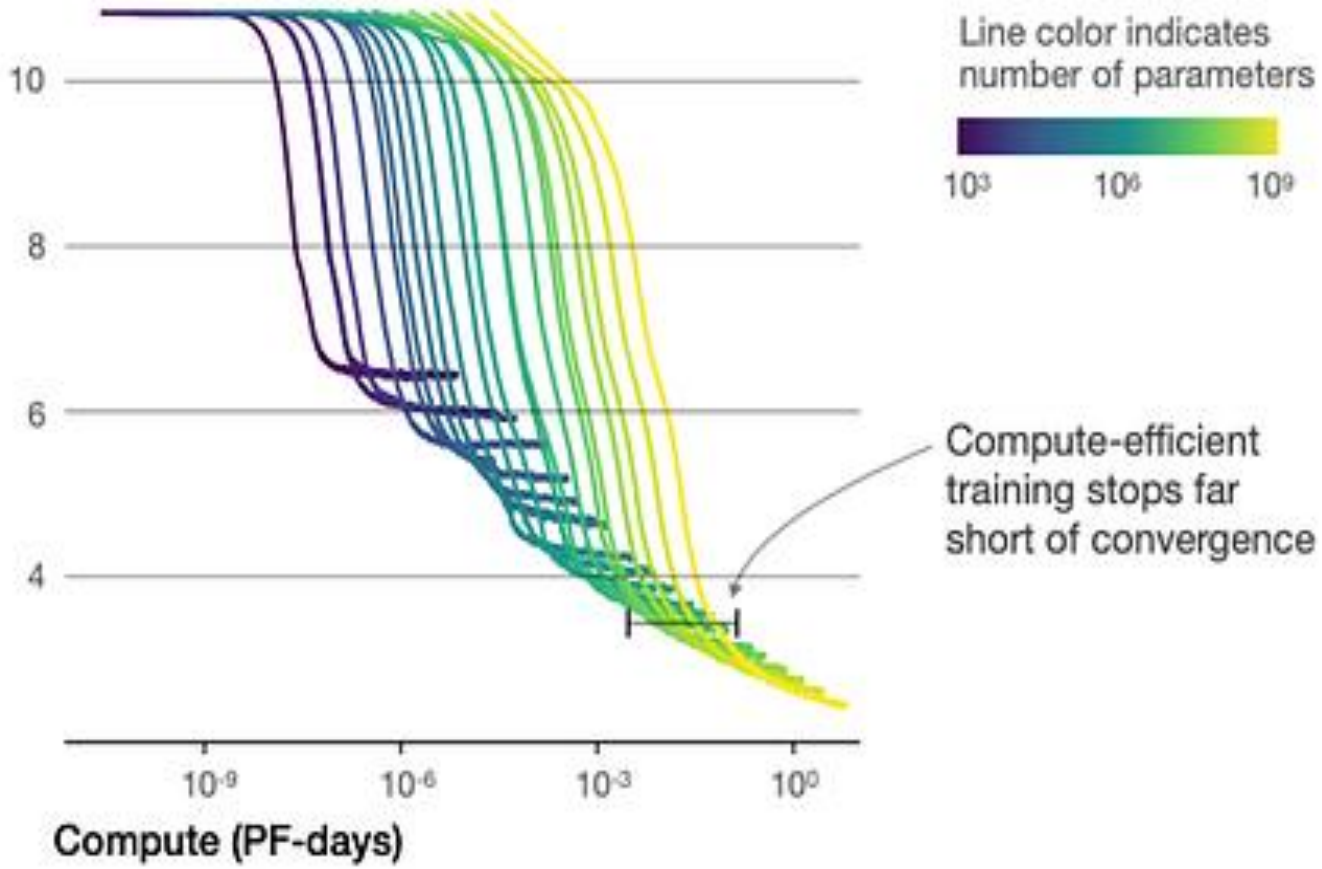
Large Language Models: A Hands-on Approach

GPU-Architecture

Why GPUs For LLM

- **Neural scaling law**





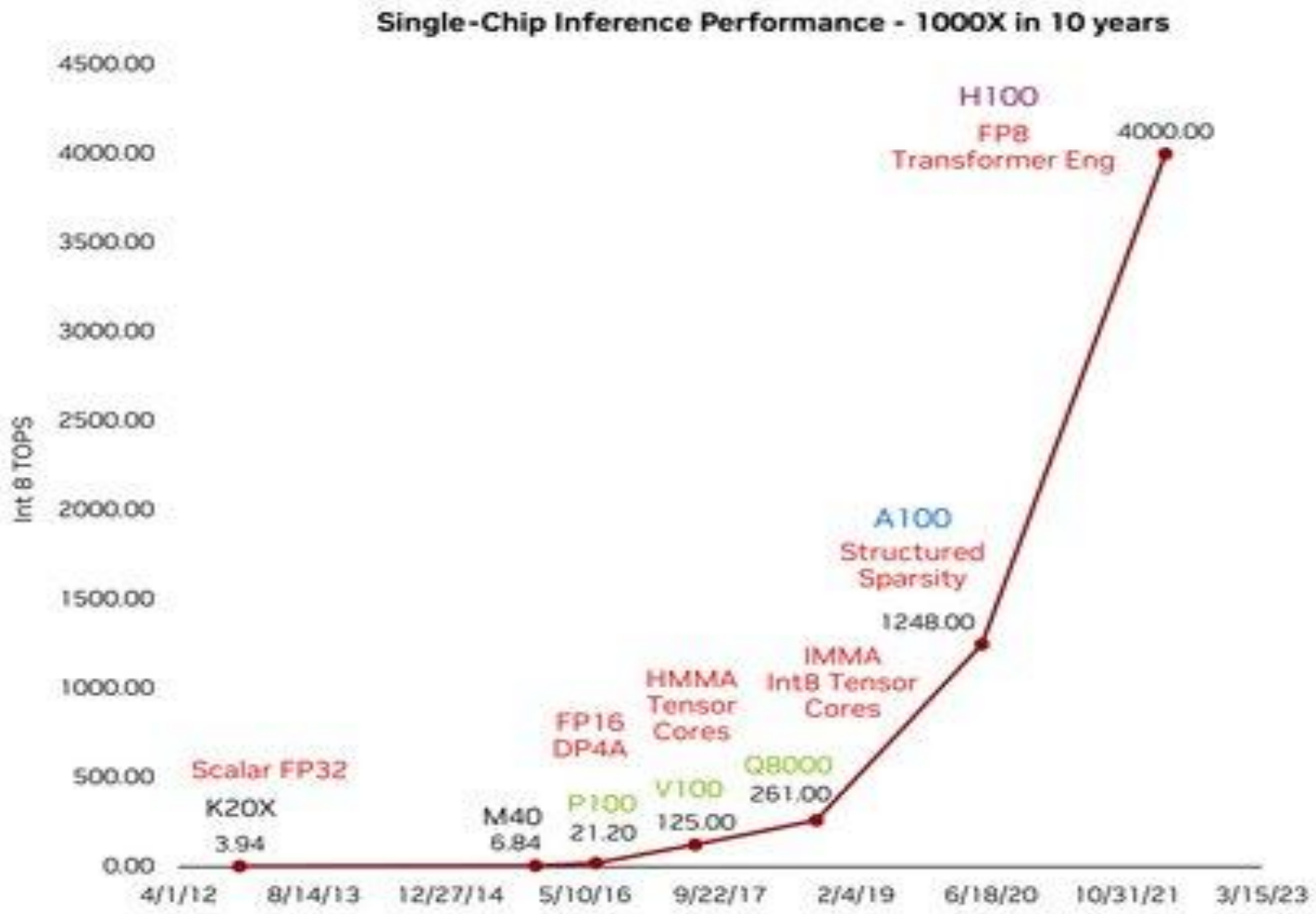
https://en.wikipedia.org/wiki/Neural_scaling_law

Kaplan et. Al. 2020 scaling laws for neural language models

Why GPUs For LLM

Gains from

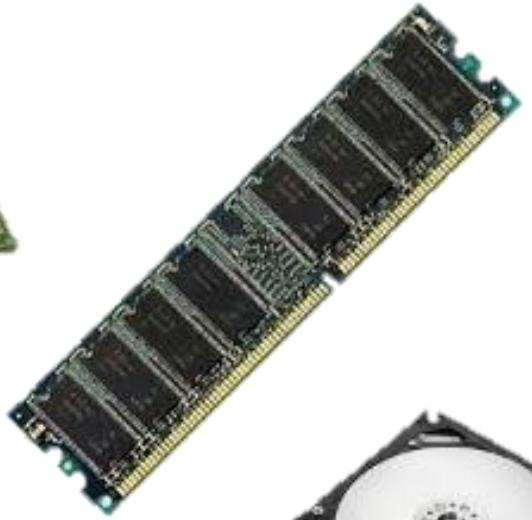
- Number Representation
  - FP32, FP16, Int8
  - (TF32, BF16)
  - ~16x
- Complex Instructions
  - DP4, HMMA, IMMA
  - ~12.5x
- Process
  - 28nm, 16nm, 7nm, 5nm
  - ~2.5x
- Sparsity
  - ~2x
- Model efficiency has also improved – overall gain > 1000x



https://spectrum.ieee.org/nvidia-gpu

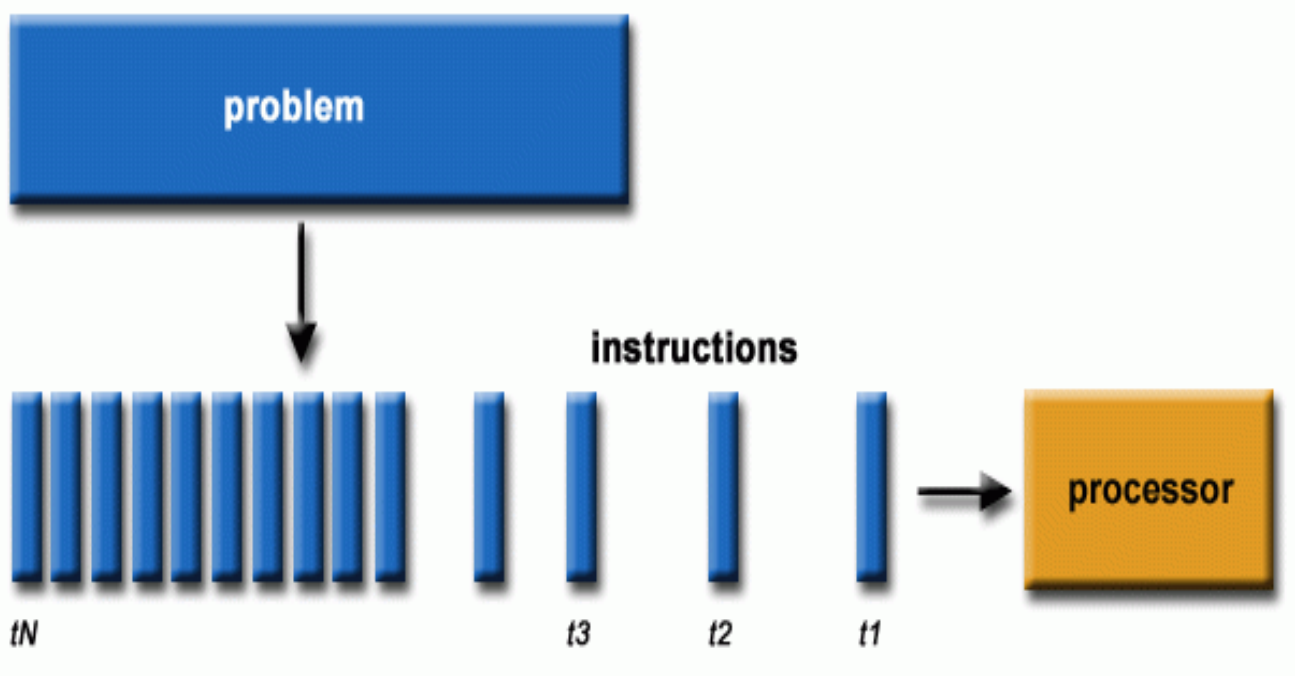
Building Blocks of Computers

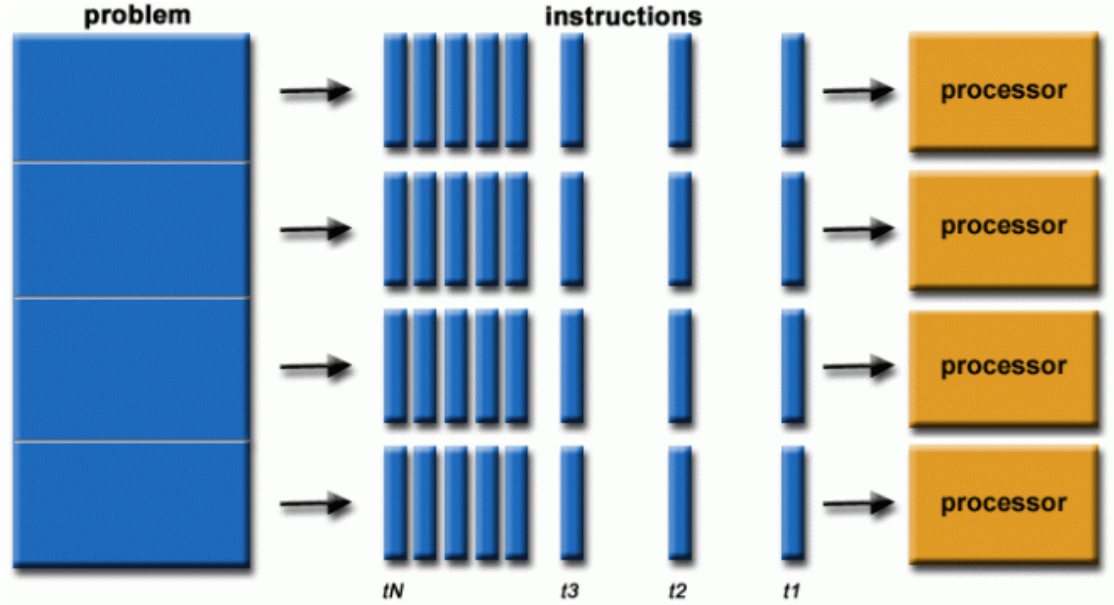
- Central Processing Units (CPUs)
  - processing information
- Memory
  - storing information
- Storage
  - keeping information
- Networks
  - moving information



- All need to improve to get faster systems

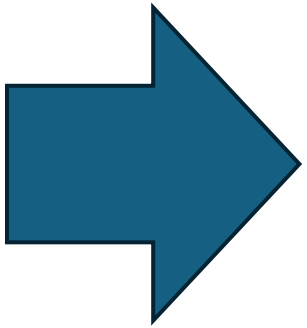
Serial/ Parallel Computing





Elements of Parallel Computing

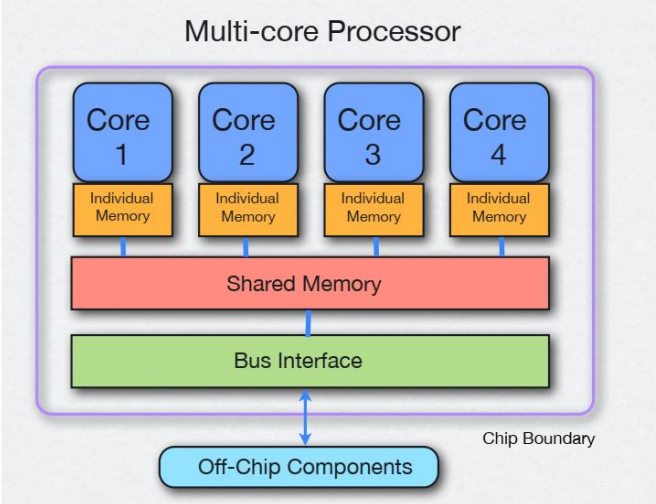
- **Hardware**
  - Multiple Levels of Parallelism (ILP, DLP, TLP)
  - Multiple Memories
  - Interconnection Network
- **System Software**
  - Parallel Operating System
  - Programming Constructs to Orchestrate Concurrency
- **Application Software**
  - Parallel Algorithms

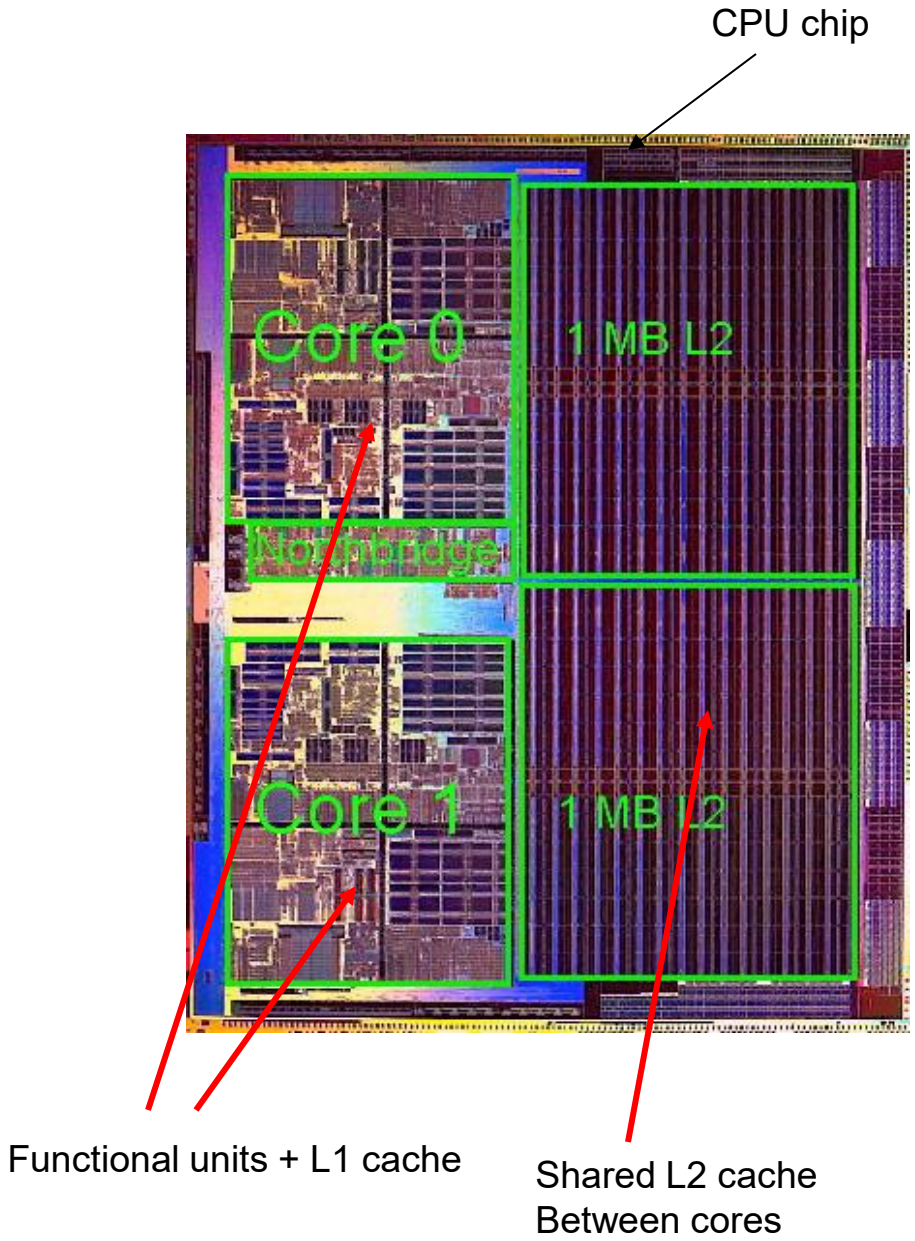


- ***GOALs***
  - **Achieve Speedup = Ts/Tp**
  - **Solve large problems**

Internal Parallelism

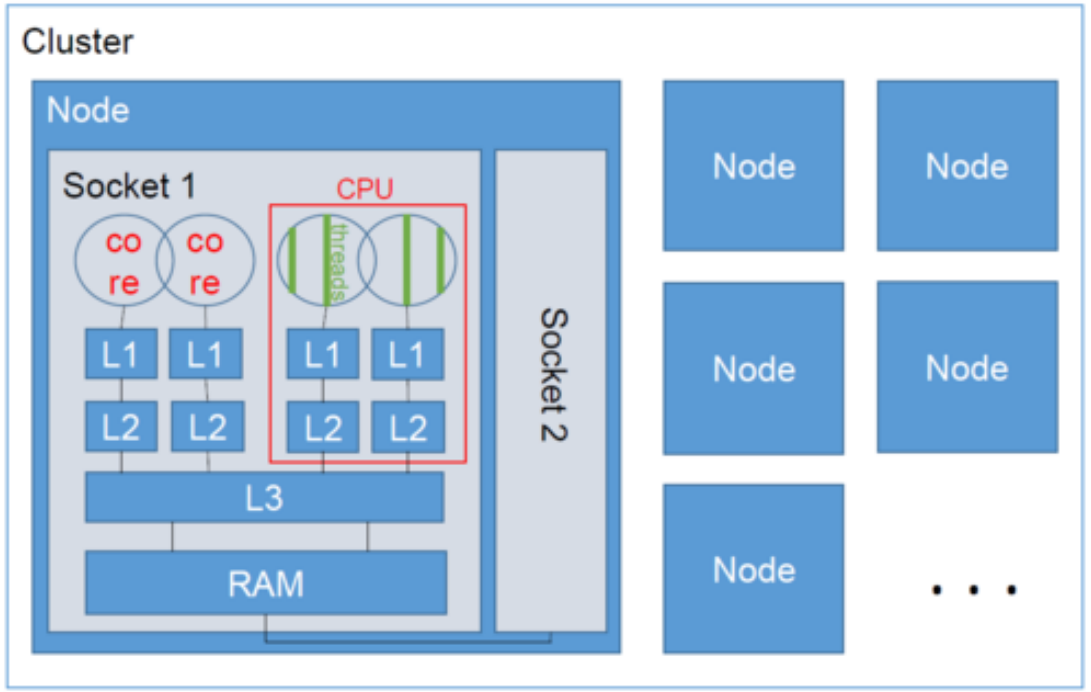
- Multi-core processors: Intel dual-core, quad-core
  - Multiple execution cores (functional units, registers, L1 cache)
  - Multiple cores share L2 cache, memory
  - Lower energy consumption
- Need *FAST* memory access to provide data to multiple cores
  - Effective memory bandwidth per core is reduced





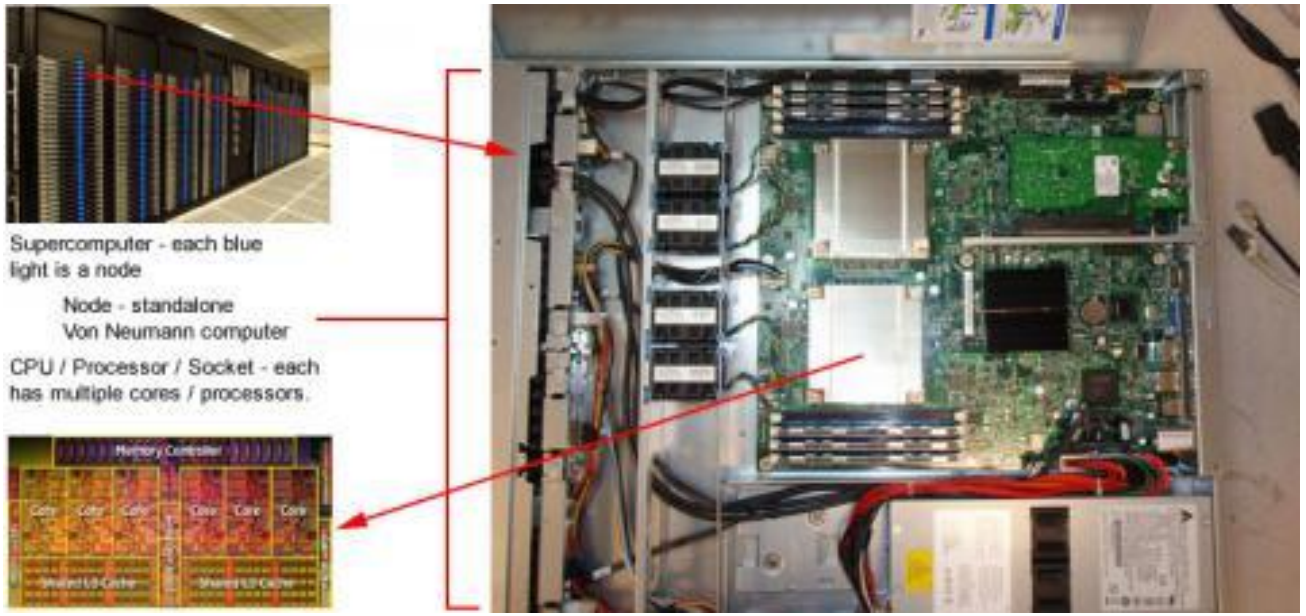
CPU / Core / Socket

- In the past, a CPU (Central Processing Unit) was a singular execution component for a computer.
- Then, individual CPUs were subdivided into multiple "cores", each being a unique execution unit.
- CPUs with multiple cores are sometimes called "sockets" - vendor dependent. The result is a node with multiple CPUs, each containing multiple cores.



Node

- A standalone "computer in a box." Usually comprised of multiple CPUs/processors/cores, memory, network interfaces, etc.
- Nodes are networked together to comprise a supercomputer.



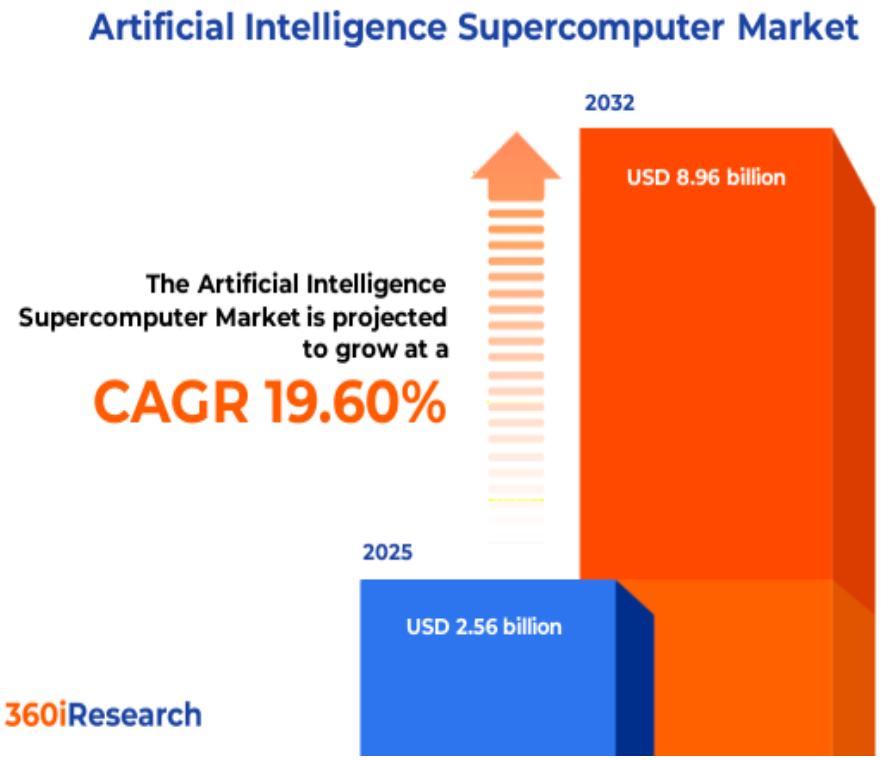
Supercomputer/Cluster/HPC

- A set of computers connected over a Local Area Network (LAN) that functions as a single large multiprocessor.
- Performance comes from more processors per chip rather than higher clock

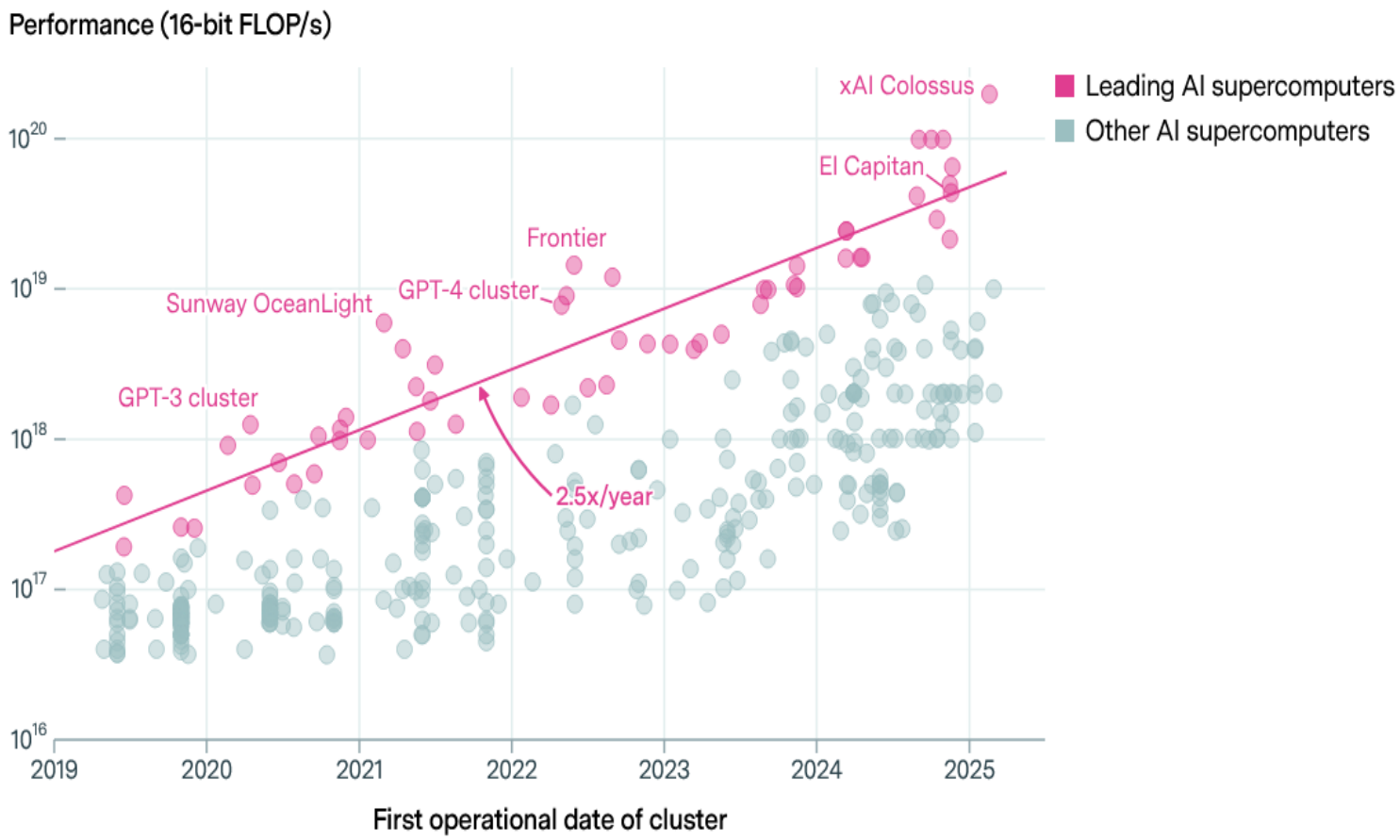
Cray XE6 – Monte Rosa (CSCS)

CERN Server

Supercomputer in AI



The performance of AI supercomputers has doubled every 9 months



*https://epoch.ai/blog/trends-in-ai-supercomputers

World’s Fastest Supercomputer

Location	Lawrence Livermore National Laboratory and U.S. Department of Energy
Power	35 MW
Speed	1.742 exaFLOPS (Rmax) / 2.746 exaFLOPS (Rpeak)

1.74 quintillion operations per second,
Using AMD CPUs and GPUs

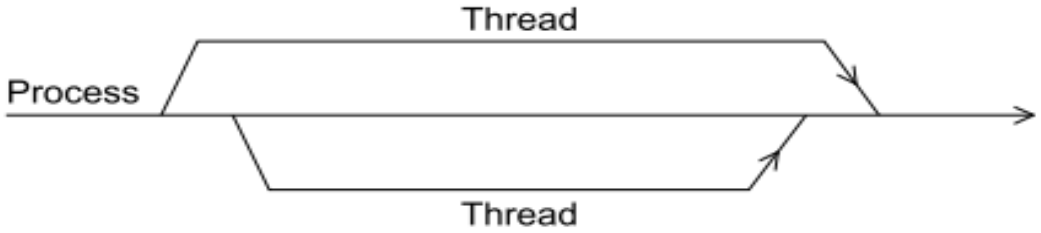
11,039,616 CPU and GPU cores

- 43,808 AMD 4th Gen EPYC 24C 24 core 1.8 GHz CPUs (1,051,392 cores) and
- 43,808 AMD Instinct MI300A GPUs (9,988,224 cores).

El Capitan

Process/Processes

- A process is an instance of a running program or a program under execution
- *Isolated* from other processes on the same machine.
- It has its own private section of the machine's memory.
- Processes normally share no memory between them.
  - A process can't access another process's memory or objects at all.
  - Sharing memory between processes is *possible* on most operating systems, but it needs special effort.



Thread

- A thread is an independent flow of control that operates within the same address space as other independent flows of control within a process.
- Thread and process characteristics are grouped into a single entity called a process
- A thread is a locus of control inside a running program.

Communications

- Parallel tasks typically need to exchange data.
- Accomplished, through a shared memory bus or over a network.
- The actual event of data exchange is commonly referred to as communications, regardless of the method employed.

Task

- A logically discrete section of computational work.
- A task is typically a program or program-like set of instructions that is executed by a processor.
- A parallel program consists of multiple tasks running on multiple processors.

Synchronization

- The coordination of parallel tasks in real time, very often associated with communications.
- Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.

Granularity

- Qualitative measure of the ratio of computation to communication.
- **Coarse**: relatively large amounts of computational work are done between communication events
- **Fine**: relatively small amounts of computational work are done between communication events

Parallel Overhead

- Required execution time that is unique to parallel tasks, as opposed to that for doing useful work.

- Parallel overhead can include factors such as:
  - Task start-up time
  - Synchronizations
  - Data communications
  - Software overhead imposed by parallel languages, libraries, operating system, etc.
  - Task termination time

Massively Parallel

- Refers to the hardware that comprises a given parallel system - having many processing elements.
- The meaning of "many" keeps increasing, but currently, the largest parallel computers are comprised of processing elements numbering in the hundreds of thousands to millions.

Embarrassingly (IDEALY) Parallel

- Solving many similar, but independent tasks simultaneously;
- Little to no need for coordination/communication between the tasks.

Scalability

Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more resources.

Factors that contribute to scalability include:

- Hardware - particularly memory-CPU bandwidths and network communication properties
- Application algorithm
- Parallel overhead related
- Characteristics of your specific application

Latency

- It is the delay between the processor issuing a request for a memory item, and the item actually arriving.
- various latencies, such as the transfer from memory to cache, cache to register, or summarize them all into the latency between memory and processor.
- Latency is measured in (nano) seconds, or clock periods.

.

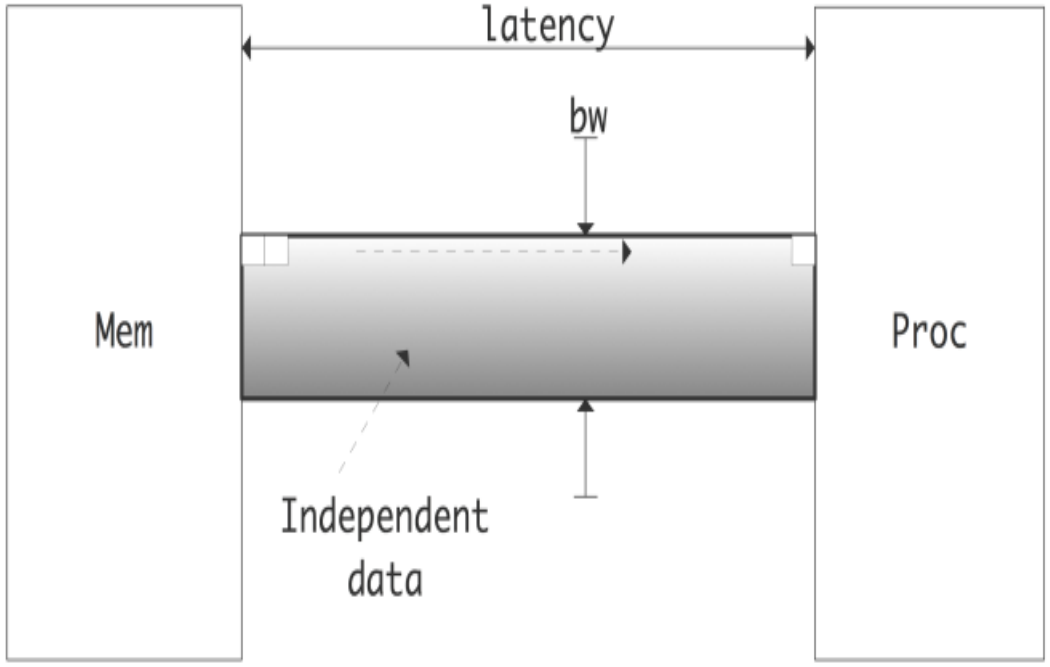
Bandwidth

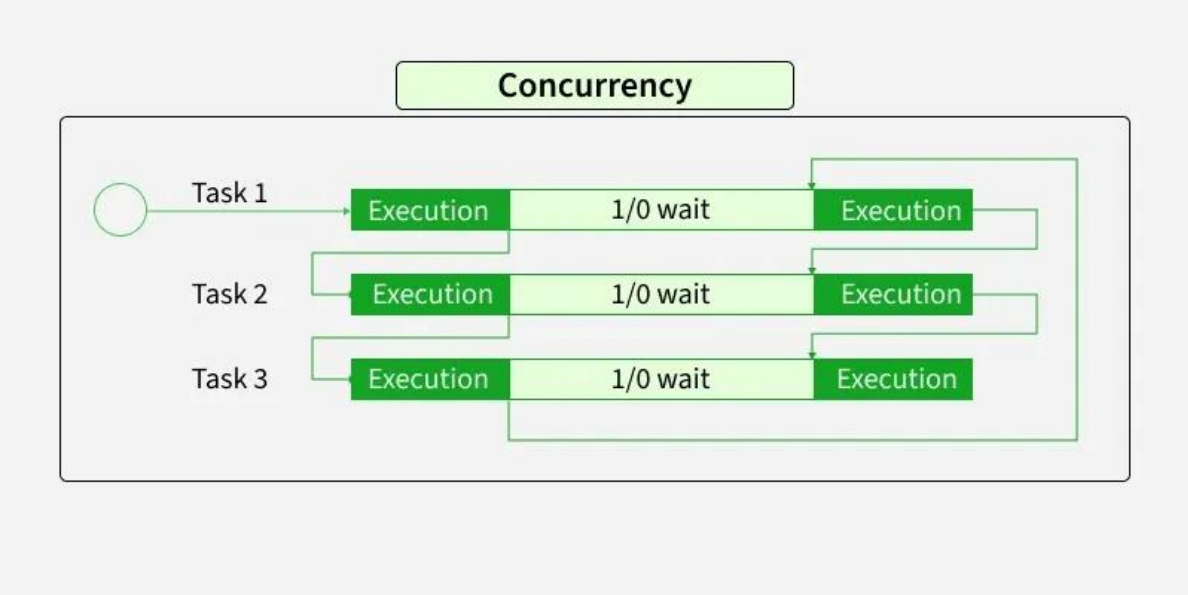
- The rate at which data arrives at its destination, after the initial latency is overcome.
- Bandwidth is measured in bytes (Kilobytes, Megabytes, Gigabytes) per second or per clock cycle.
- The bandwidth between two memory levels is usually the product of the cycle speed of the channel (the *bus speed*) and the *bus width*: the number of bits that can be sent simultaneously in every cycle of the bus clock.

Concurrency

- Execution of the multiple instruction/computation by a processor at the same time without affecting the outcome
- Concurrency means executing multiple tasks at the same time but not necessarily simultaneously.

Concurrency = Bandwidth × Latency*:*





Concurrency

Note: Concurrency is not parallelism:

Concurrency is about dealing with lots of things at once

but parallelism is about doing lots of things at once

- Examples of parallel execution of the concurrent units
  - Multiple computers in a network
  - Multiple applications running on one computer
  - Multiple processors in a computer (multiprocessor cores on a single chip)
  - Web sites must handle multiple simultaneous users.
  - Mobile apps need to do some of their processing on servers (“in the cloud”).

Types of Parallelism

Instruction Level Parallelism

- Simultaneous execution of a sequence of instructions in a computer program
- Perform the same operation on different items of data at the same time
- The parallelism grows with the size of the data.
- Different instructions within a stream can be executed in parallel
- Pipelining, out-of-order execution, speculative execution
- Depends on dataflow

1. e = a + b
2. f = c + d
3. g = e * f

Operations 1 and 2 do not depend on any other operation, so they can be computed simultaneously

Types of Parallelism

Data Parallelism

- The same operation is performed concurrently (that is, in parallel) on elements in a source collection or array.
- Partition the data used in solving the problem among the cores.
- Each core carries out similar operations on it's part of the data.
- Different pieces of data can be operated on in parallel
- SIMD: Vector processing, array processing
- Systolic arrays (*tightly coupled data processing units (DPUs)*), streaming processors

Types of Parallelism

Task Parallelism

- Partition various tasks carried out to solve the problem among the cores.
- Perform distinct computations or tasks at the same time; with the number of tasks fixed, the parallelism is not scalable.
- Different “tasks/threads” can be executed in parallel
- Multithreading
- Multiprocessing (multi-core)

Task Parallelism: Creating Tasks

- **Threads**: Partition a single problem into multiple related tasks
  - Implicitly: Thread-level Parallelism (OpenMP, CUDA)
    - Partition a single thread
  - Explicitly: Parallel programming (MPI)
    - Easy when tasks are natural in the problem
      - Web/database queries
    - Difficult when natural task boundaries are unclear

- **Processes**: Run many independent tasks together
  - Easy when there are many processes
    - Batch simulations, different users, cloud computing workloads
  - Does not improve the performance of a single task

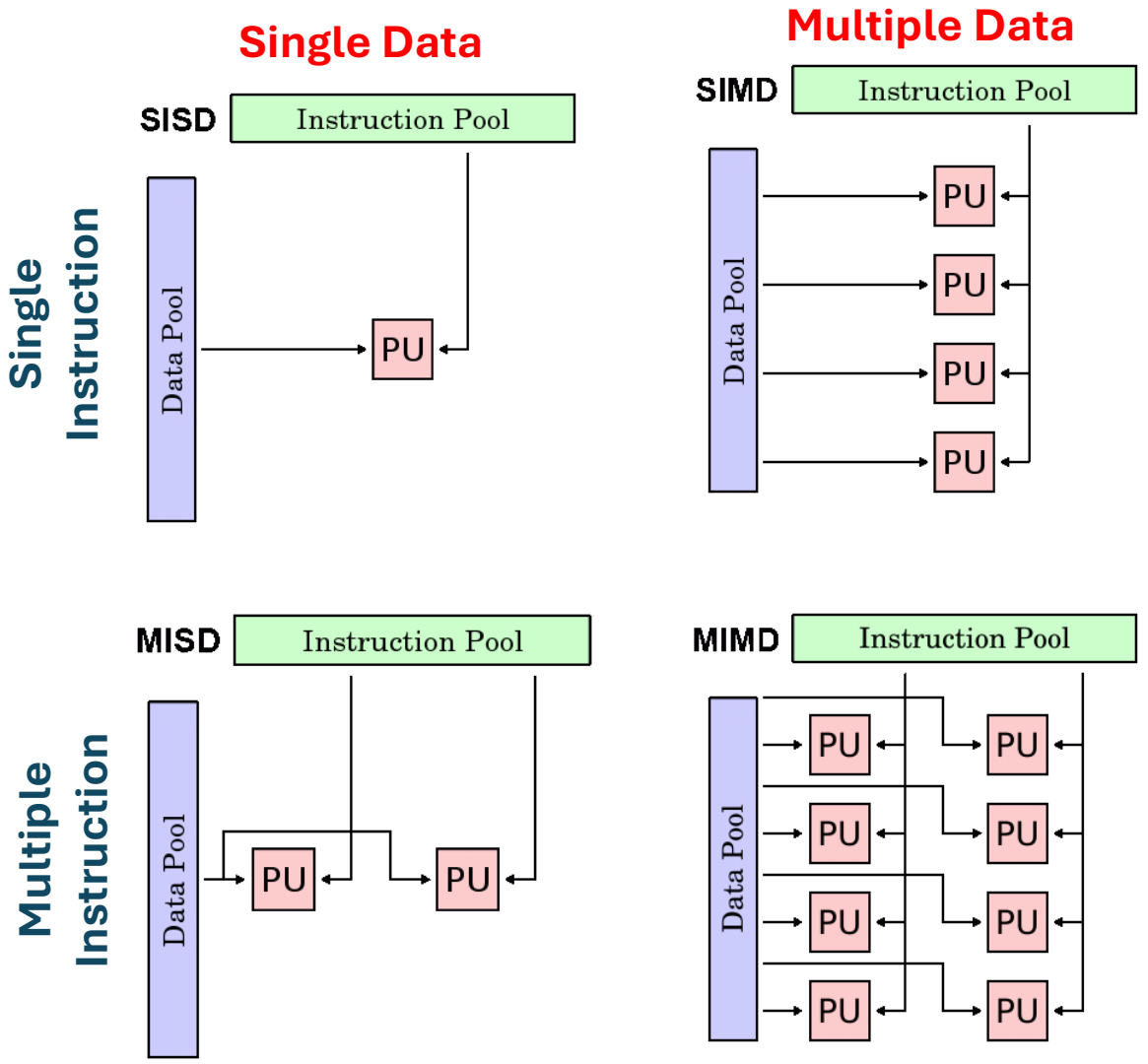
Parallel Computing Platforms: Architectures

- **Flynn's Taxonomy (1966)**

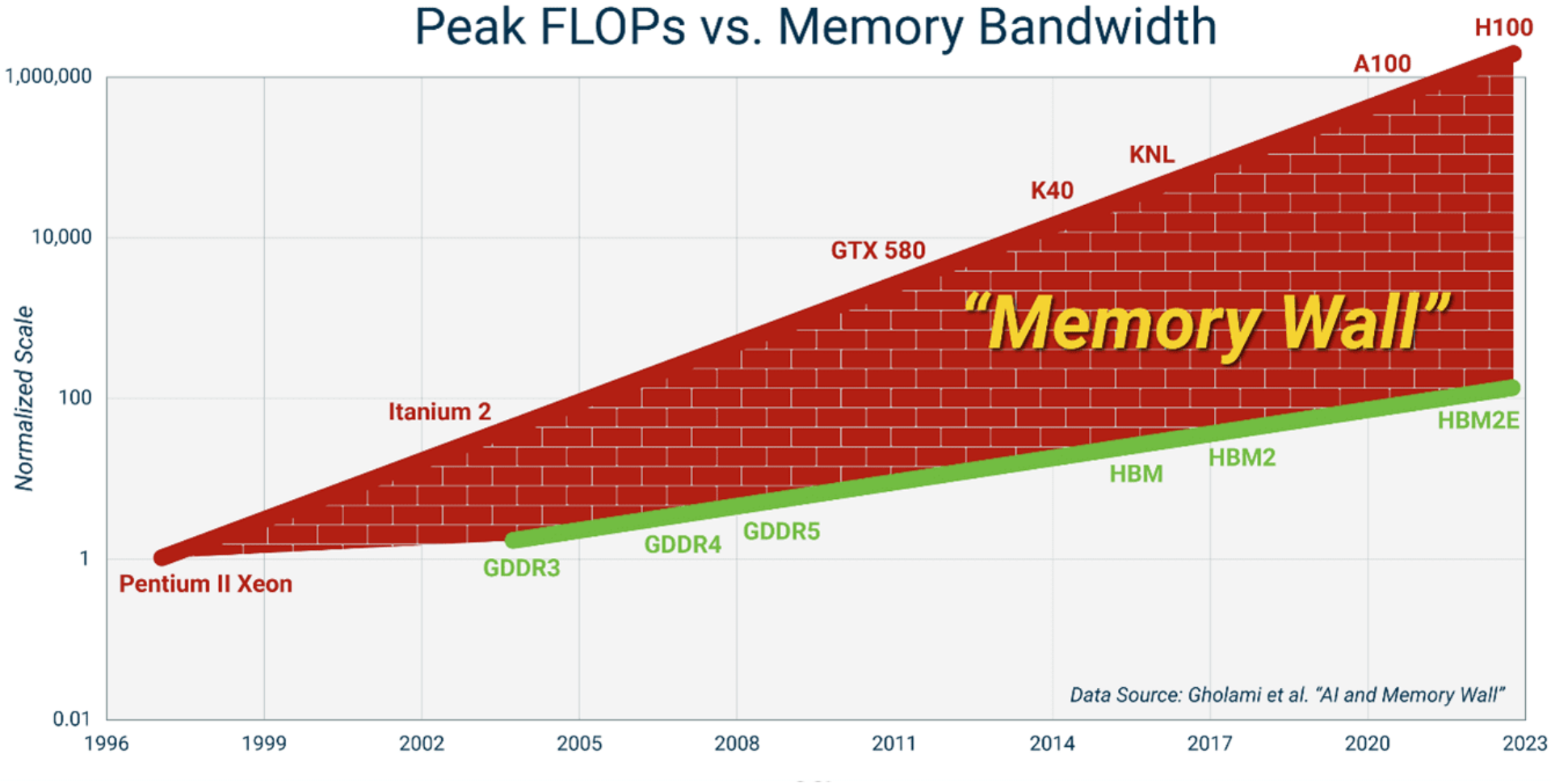
  Distinguishes multi-processor computer architectures according to how they can be classified

  Instruction Stream and ***Data Stream***.

  Each with 2 states: ***Single*** or ***Multiple***.



Memory


Peak FLOPs vs. Memory Bandwidth
Normalized Scale
1,000,000
10,000
100
1
0.01
1996
1999
2002
2005
2008
2011
2014
2017
2020
2023
H100
A100
KNL
K40
GTX 580
Itanium 2
Pentium II Xeon
"Memory Wall"
HBM2E
HBM2
HBM
GDDR5
GDDR4
GDDR3
Data Source: Gholami et al. "AI and Memory Wall"

Memory Hierarchy

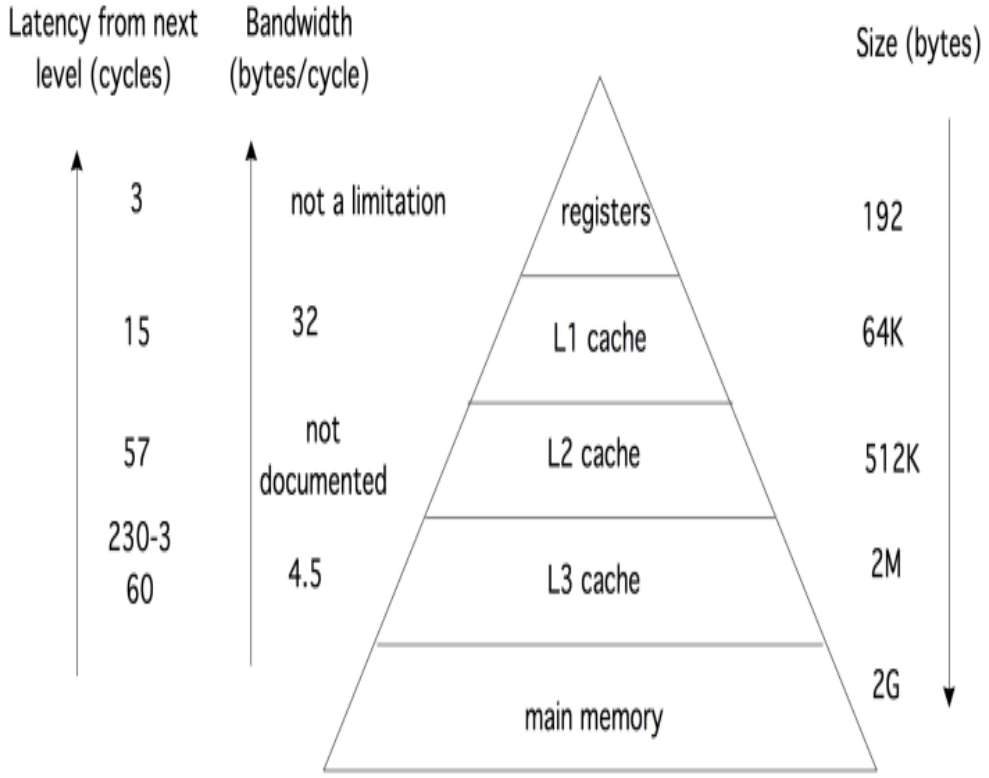
- Overcomes the **Memory Wall problem**
- Various memory levels between the Floating-Point Unit (FPU) and the main memory
- The registers and the caches, together called the **memory hierarchy.**

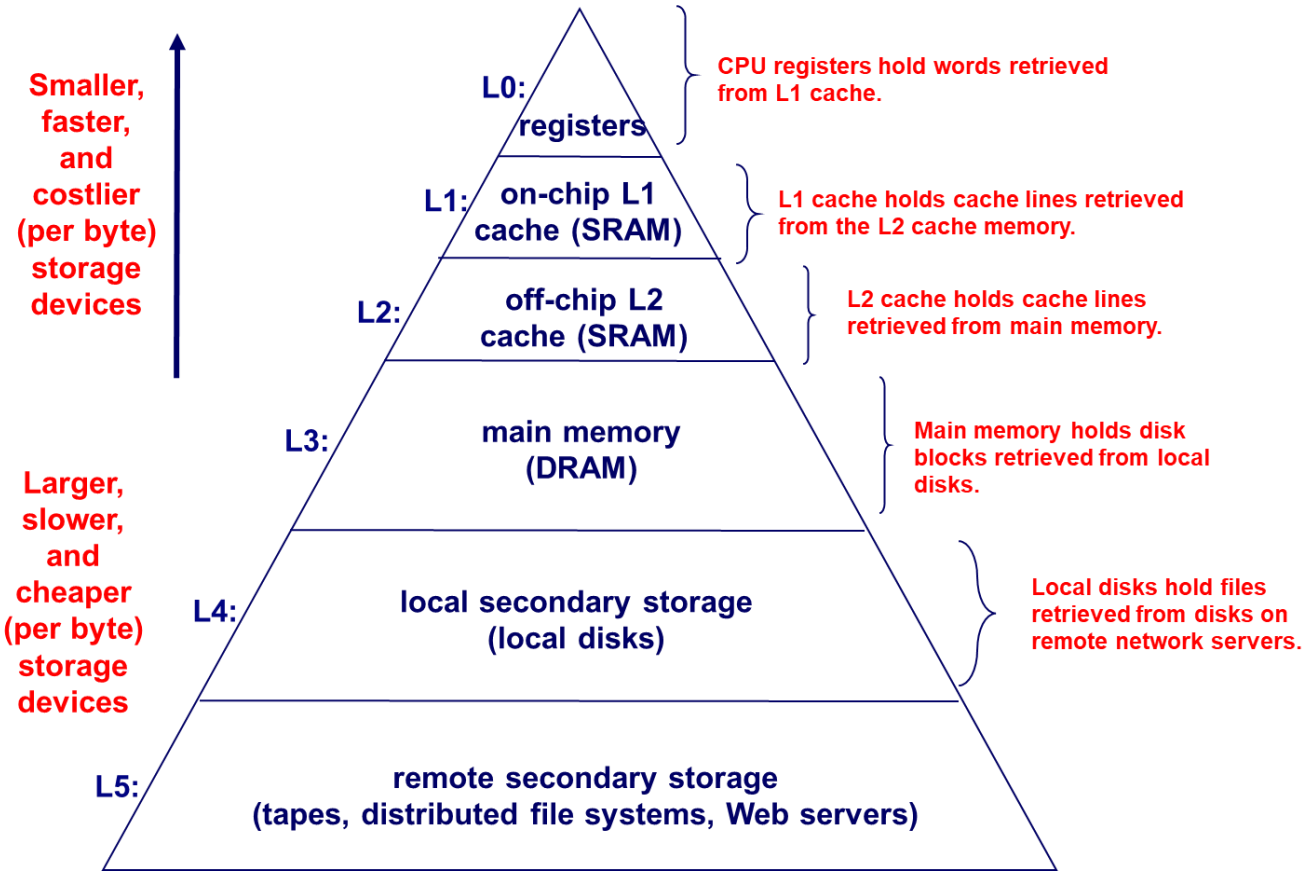
Various components of the Memory Hierarchy are

- **Buses**: The wires that move data around in a computer, from memory to CPU or to a disc controller or screen
- **Registers**: A Small amount of memory that is internal to the processor
- **Caches**: Are the main memory where lots of data can reside for a long time

Memory Hierarchy

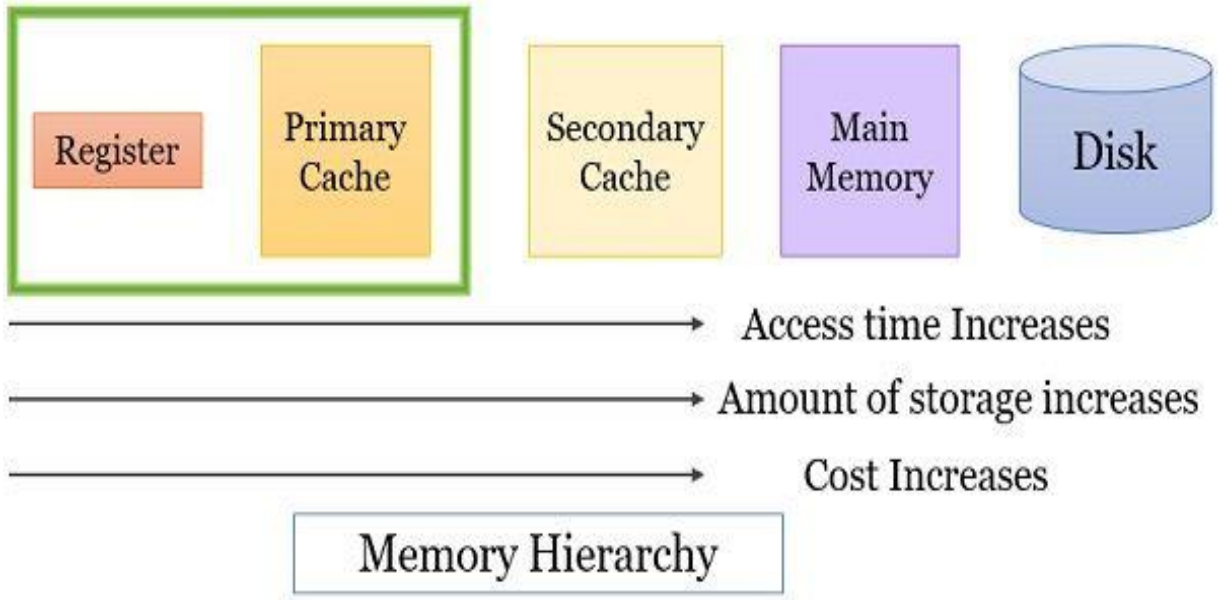
Cache Levels:





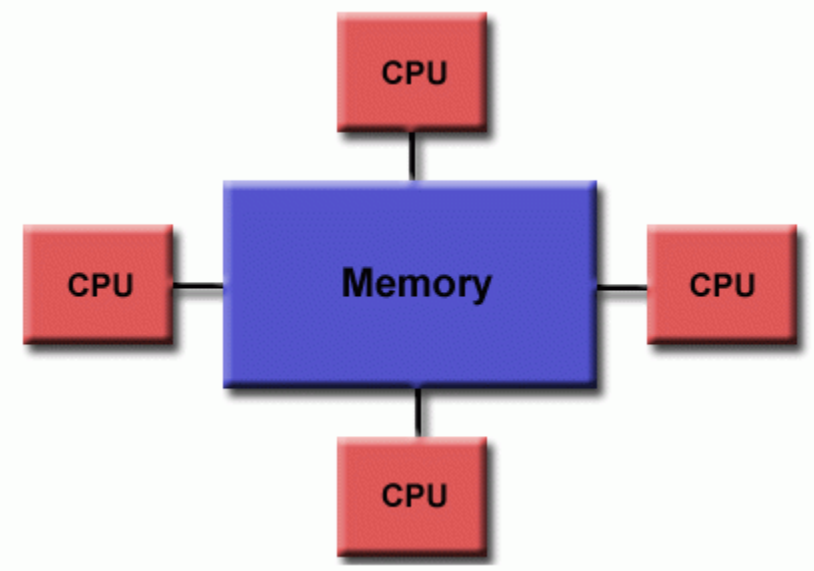
Memory Hierarchy

- Balancing performance with cost
  - Small memories are fast but expensive
  - Large memories are slow but cheap
- Exploit locality to get the best of both worlds
  - locality = re-use/nearness of accesses
  - allows most accesses to use small, fast memory



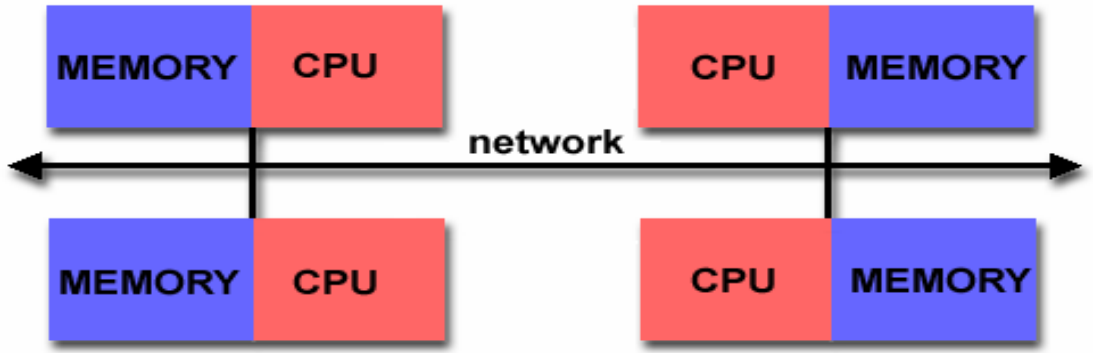
Parallel Computing Platforms :Memory

- Shared memory
  - Historically, shared memory machines have been classified as ***UMA*** and ***NUMA***, based upon memory access times.



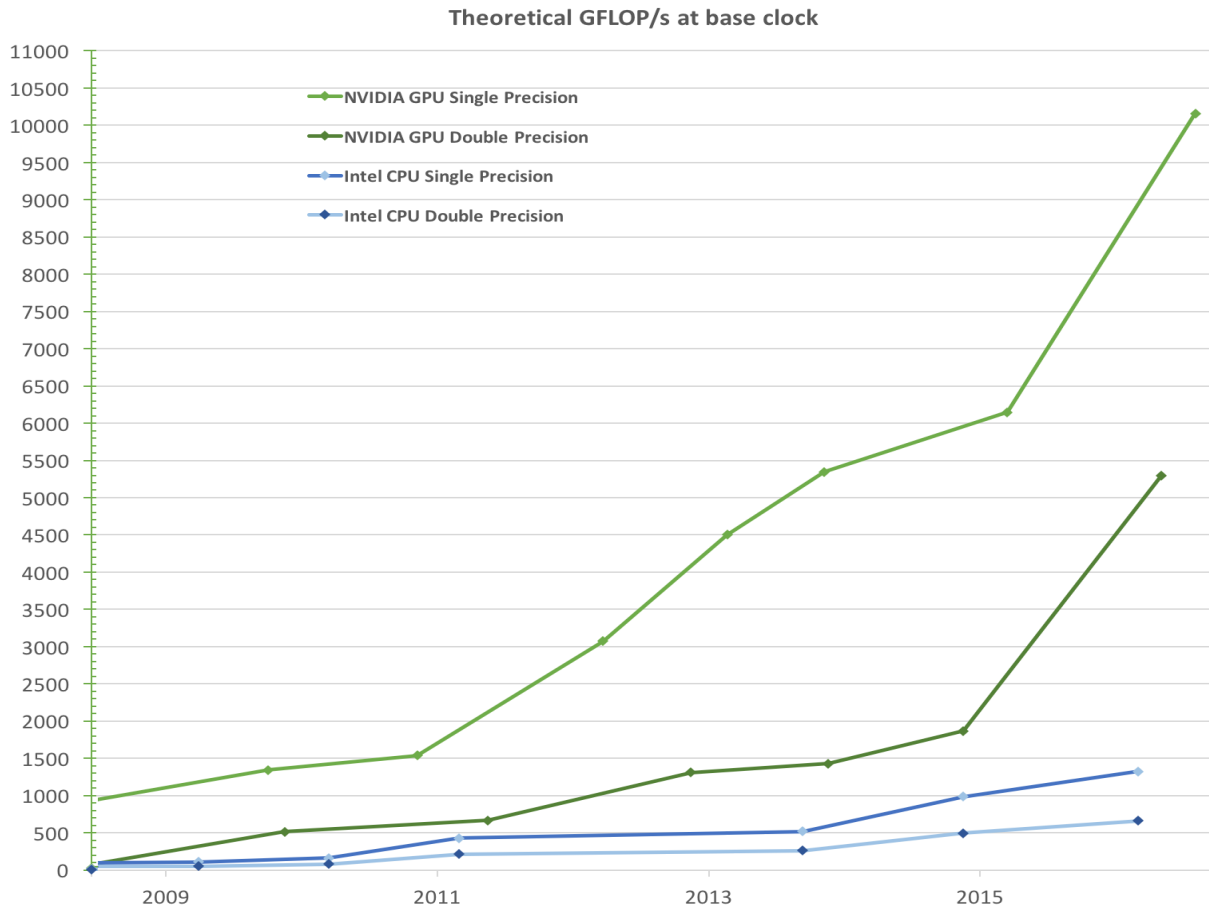
- Distributed Memory

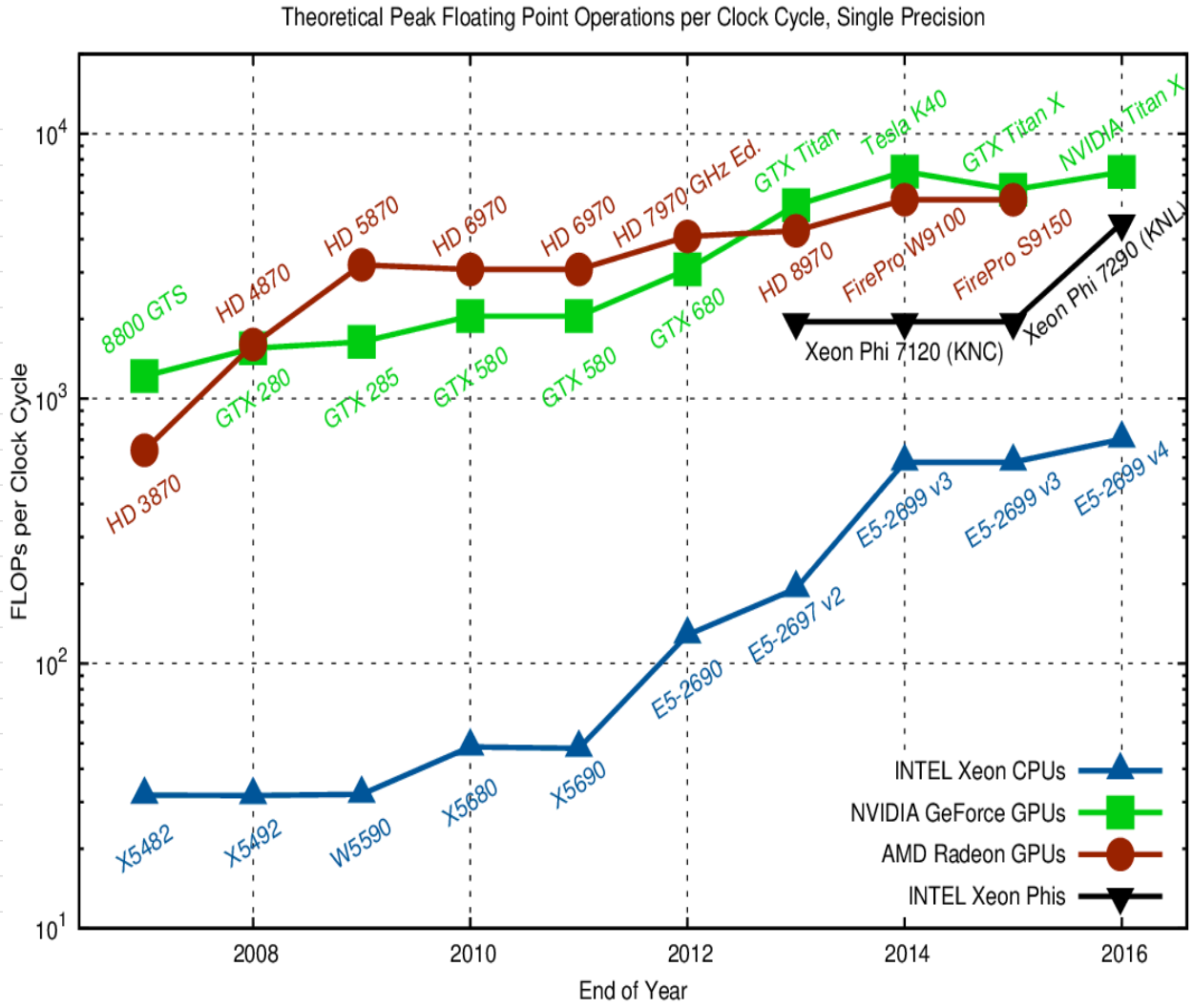
- Hybrid Distributed-Shared Memory



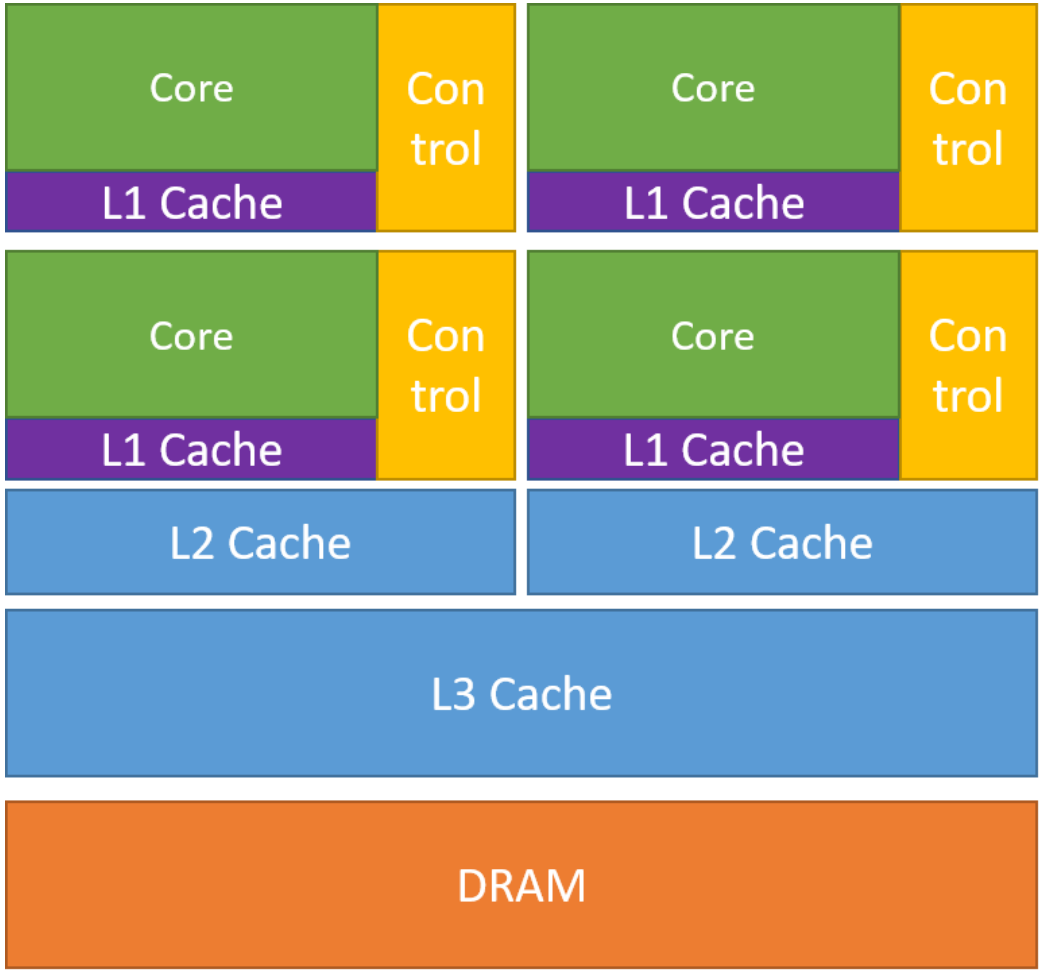
Why GPU?

Peak performance in Gflop/s of GPUs and CPUs in single and double precision, 2009-2016.

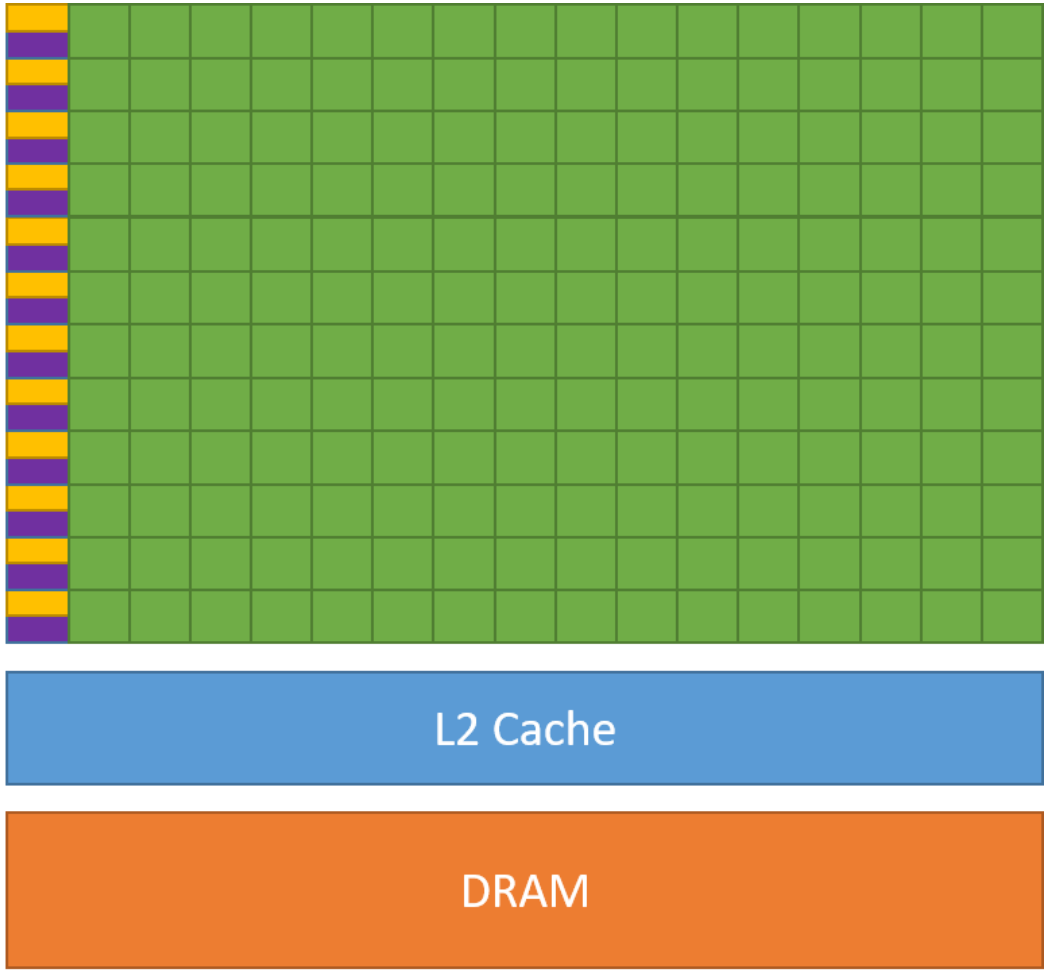




Why GPU?

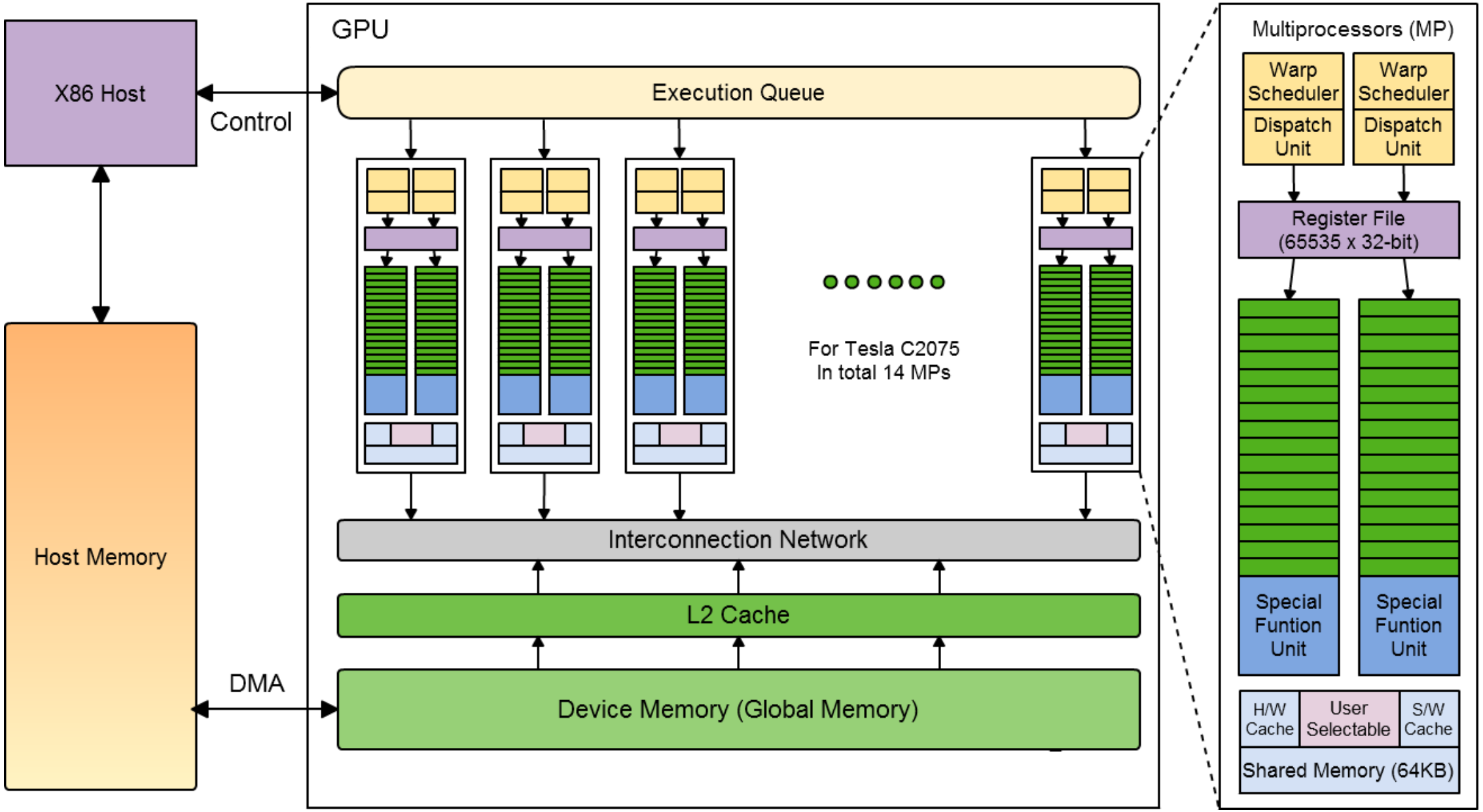


CPU



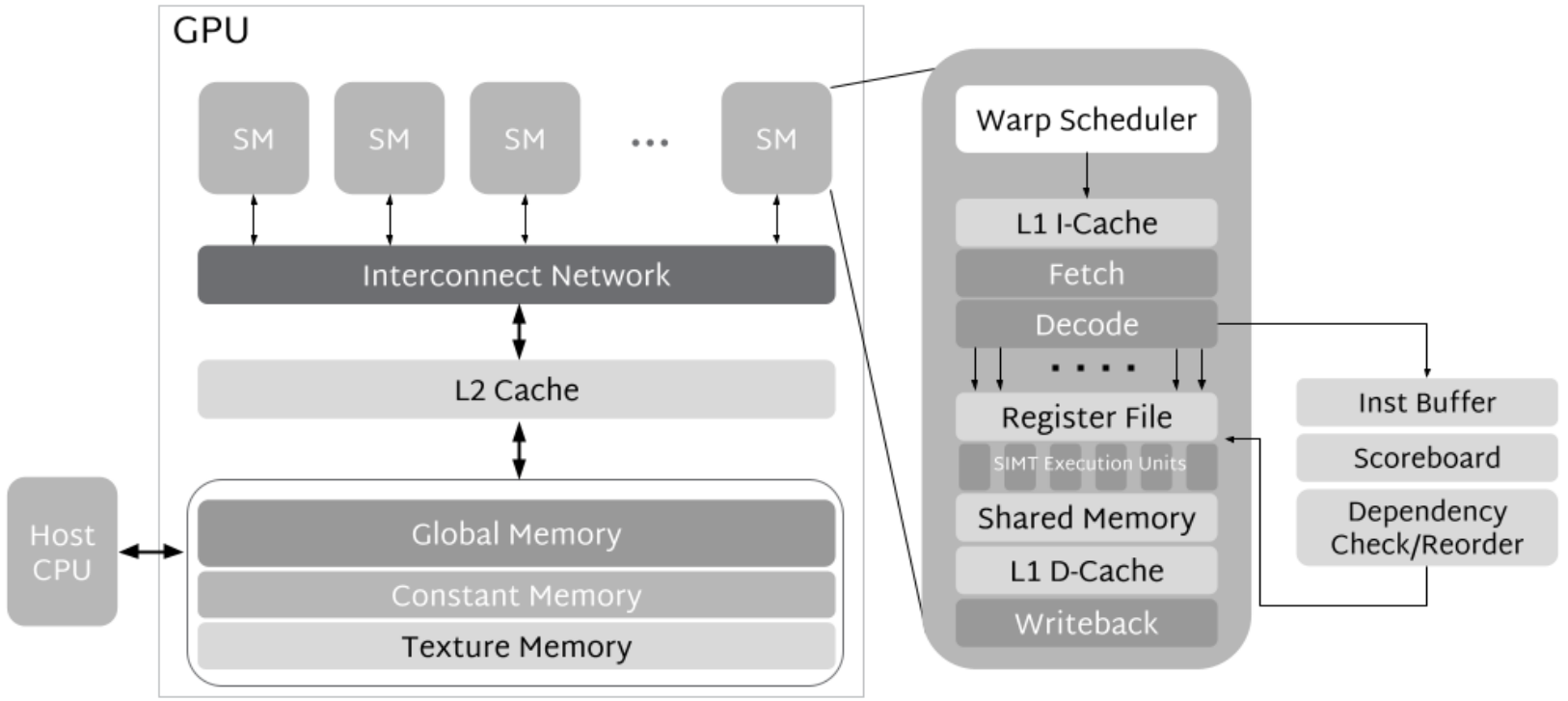
GPU

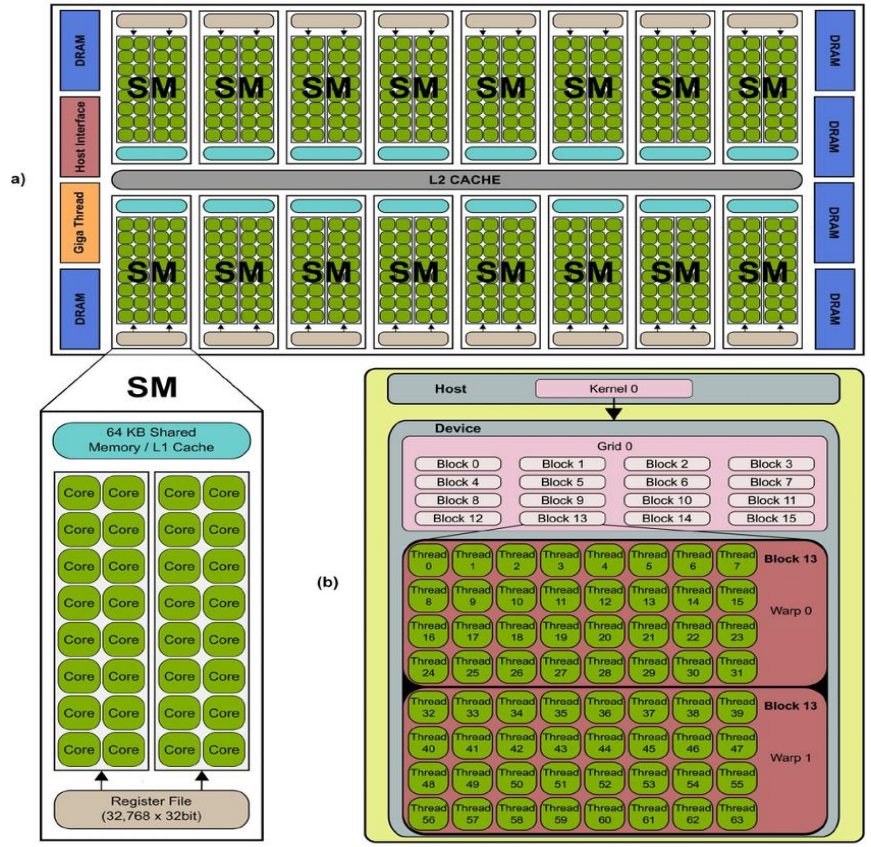
GPU Architecture



Architecture of a modern GPU

Van Newman Architecture



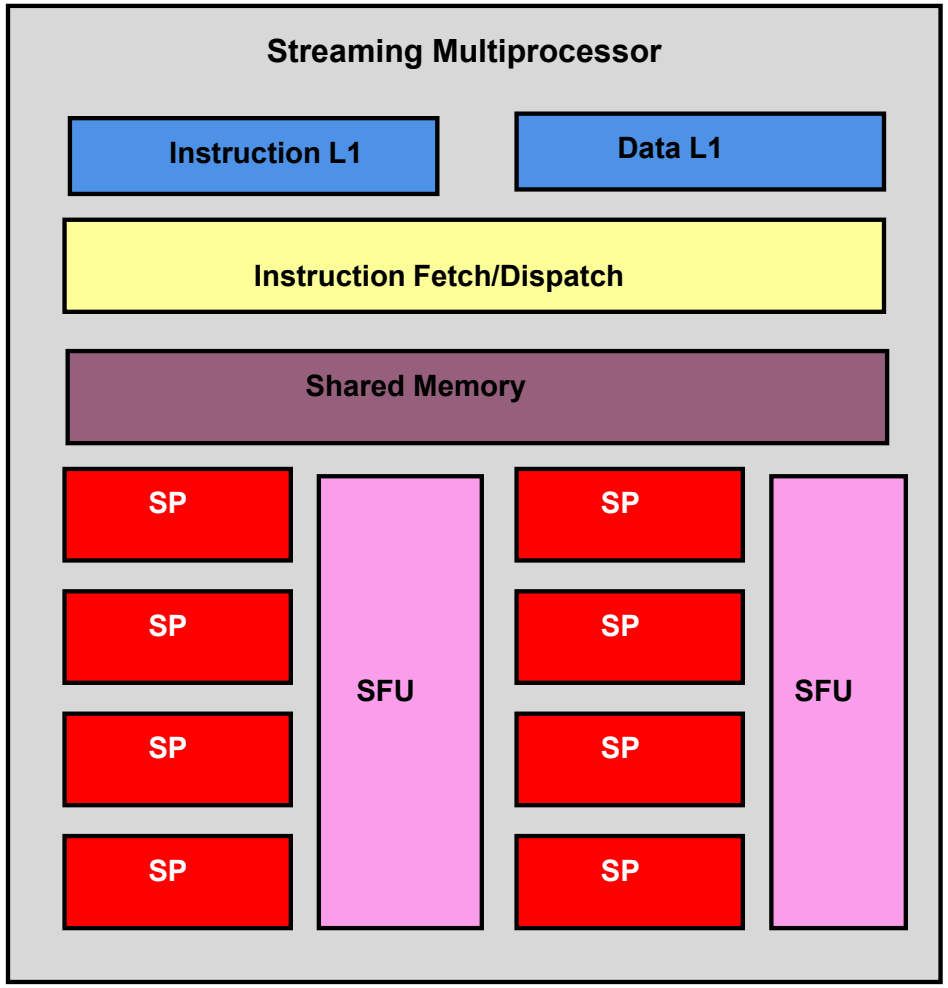


Streaming Multiprocessor (SM)

- Streaming Multiprocessor (SM)
  - 8 Streaming Processors (SP)/Core
  - 2 Special Function Units (SFU)

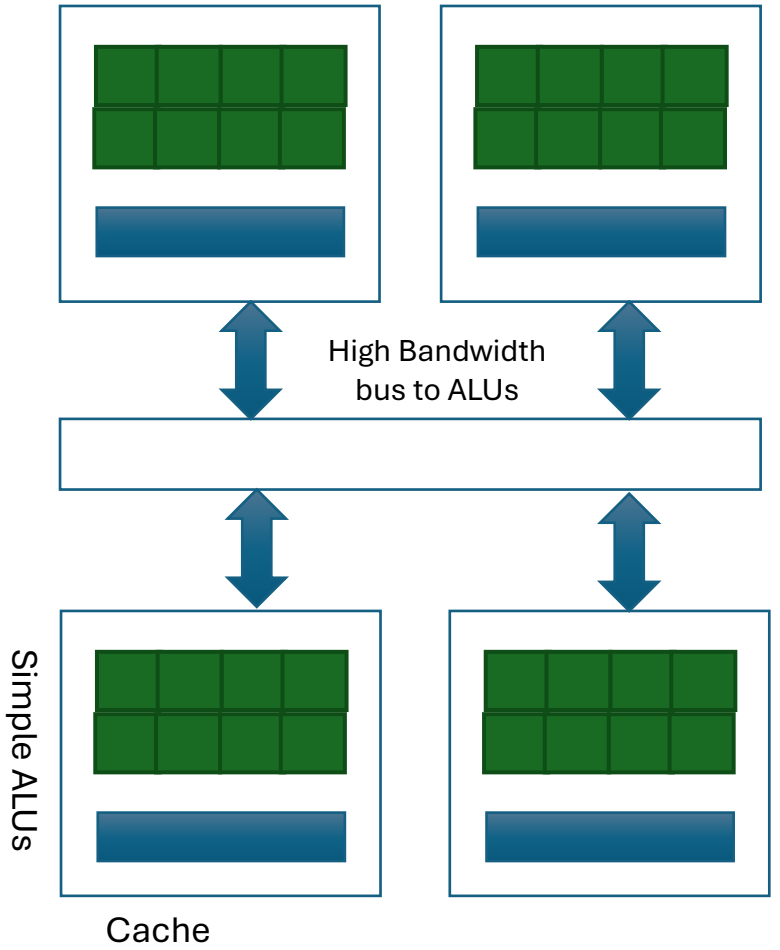
- Multi-threaded instruction dispatch
  - 1 to 512 threads active
  - Shared instruction fetch per 32 threads
  - Cover latency of texture/memory loads

- 20+ GFLOPS
- 16 KB shared memory
- DRAM texture and memory access

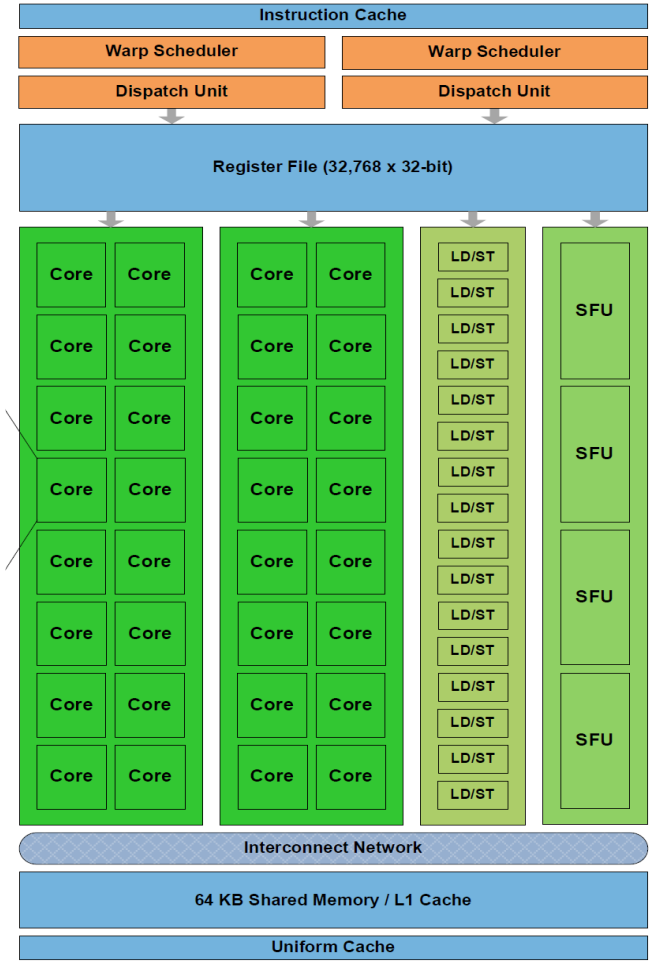


Modern GPU Architecture

- Generic many core GPU
  - Less space devoted to control logic and caches
  - Large register files to support multiple thread contexts
- Low-latency hardware-managed thread switching
- Large number of ALUs per “core” with a small user-managed cache per core
- Memory bus optimized for bandwidth
  - ~150 GBPS bandwidth allows us to service a large number of ALUs simultaneously



Streaming Multiprocessor (SM)



Fermi Streaming Multiprocessor (SM)

Early generation

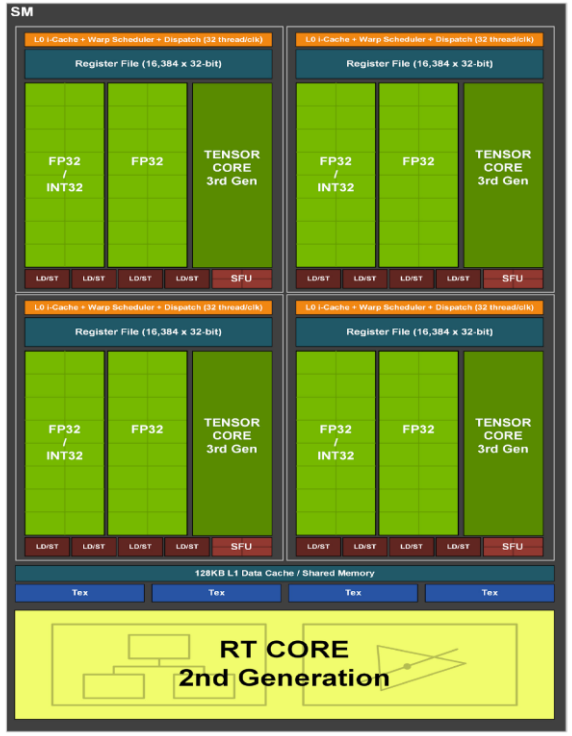
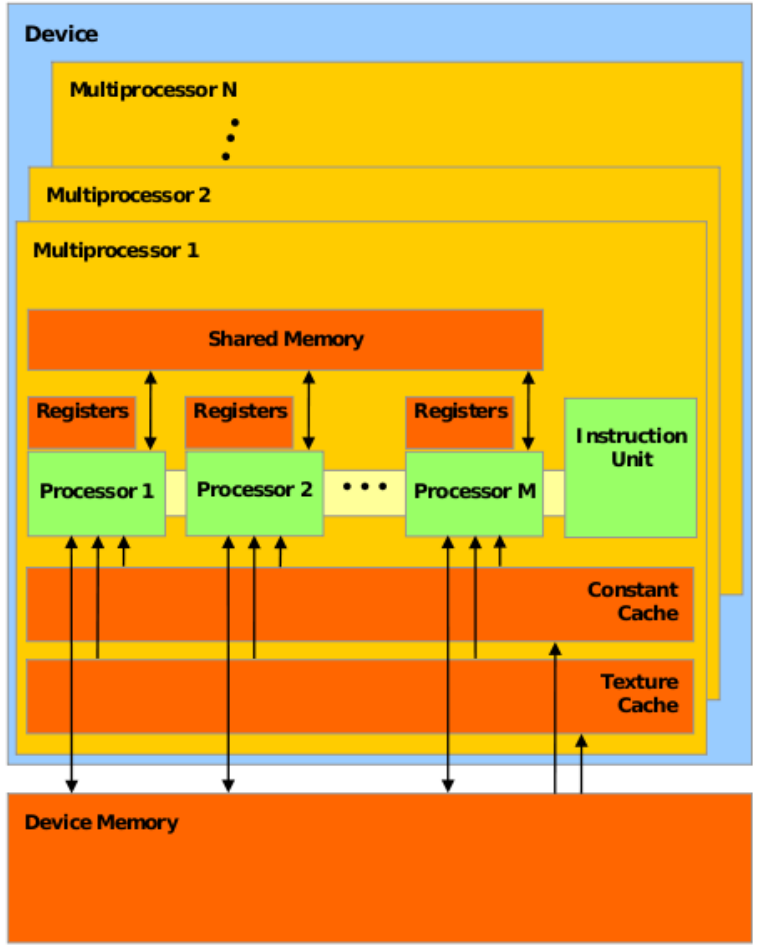


Figure 3. GA10x Streaming Multiprocessor (SM)

Latest generation



Block diagram

Modern GPU Architecture (Volta 2017)

21B transistors

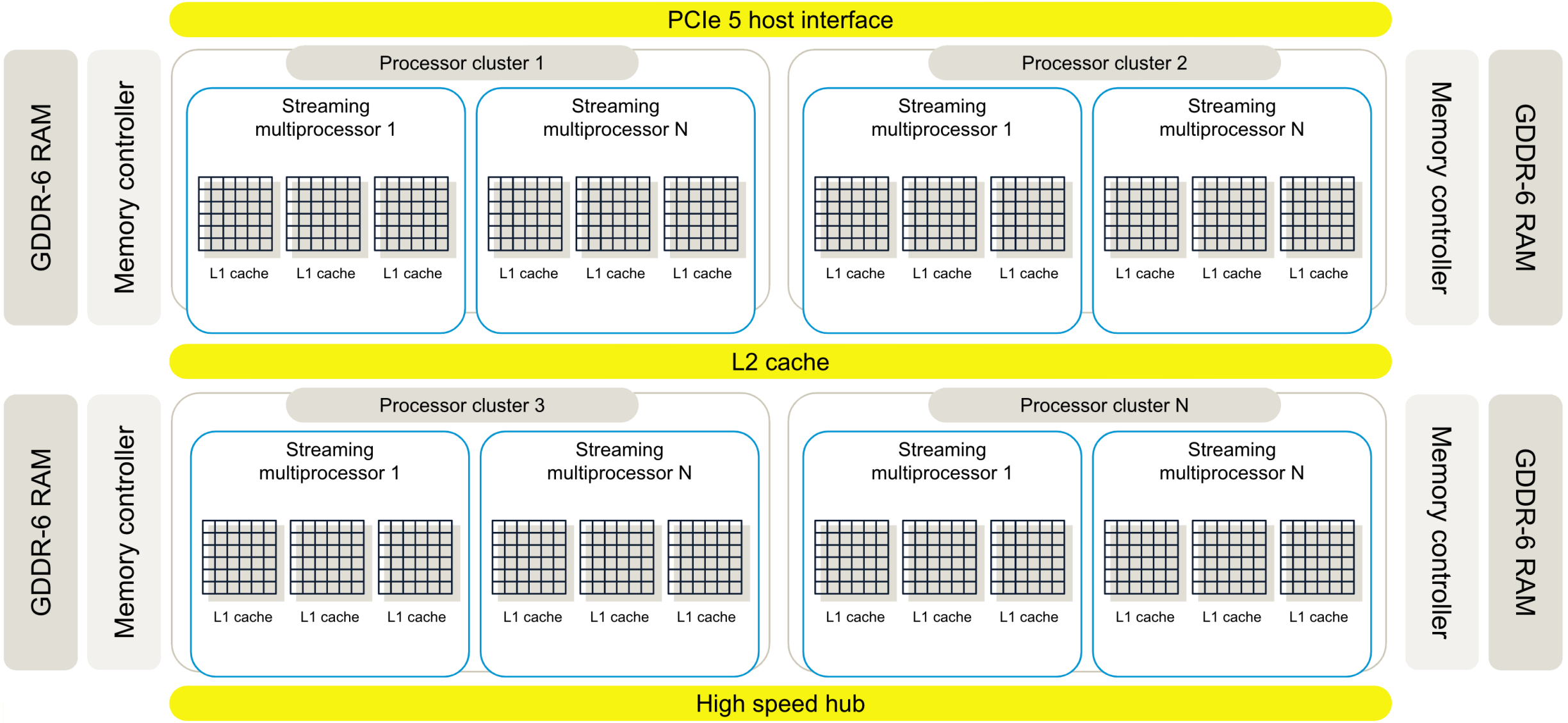
815 mm²

80 SM
5120 CUDA Cores
640 Tensor Cores

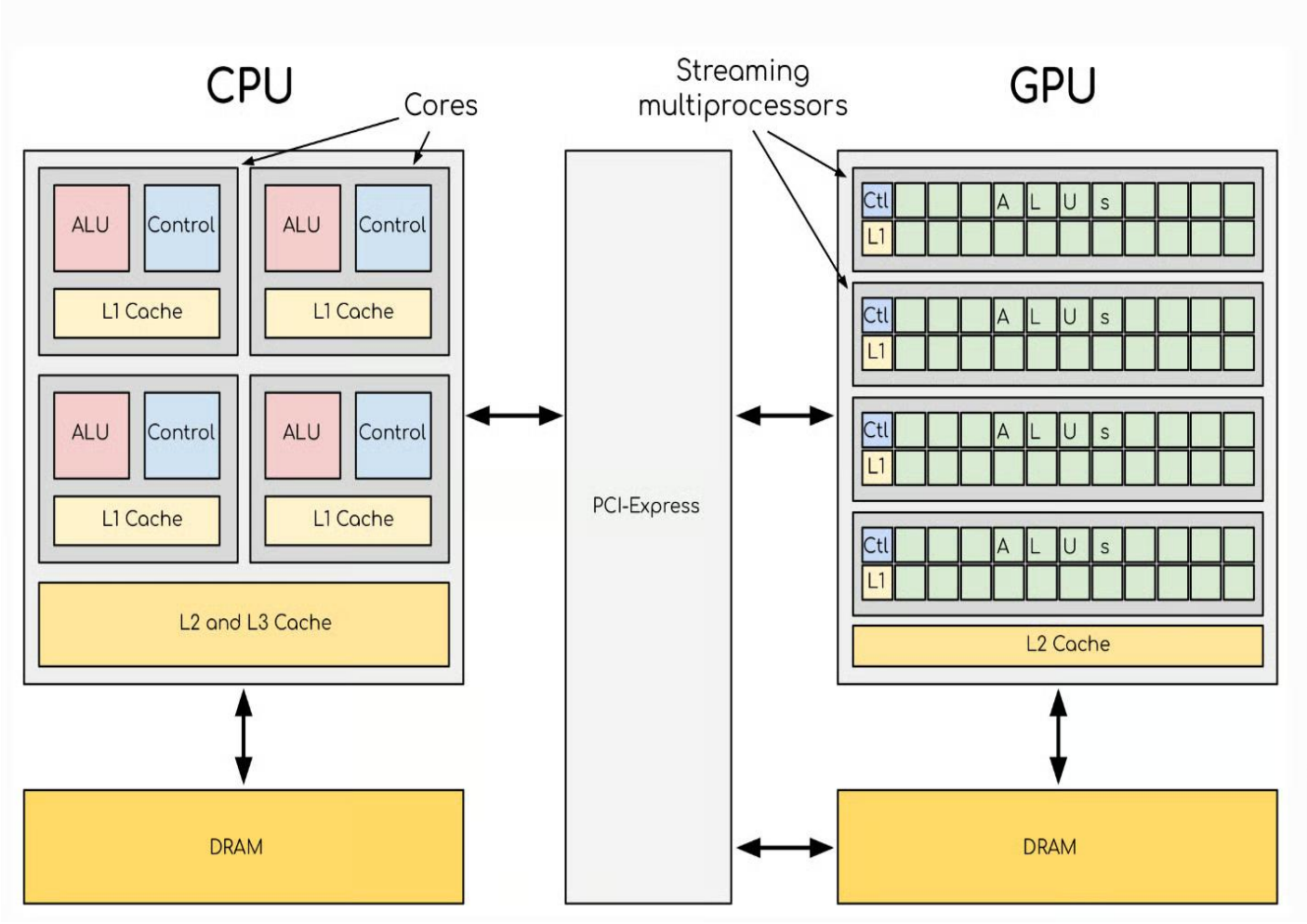
16/32 GB HBM2
900 GB/s HBM2
300 GB/s NVLink

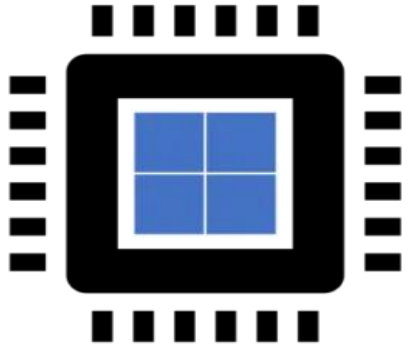
*full GV100 chip contains 84 SMs

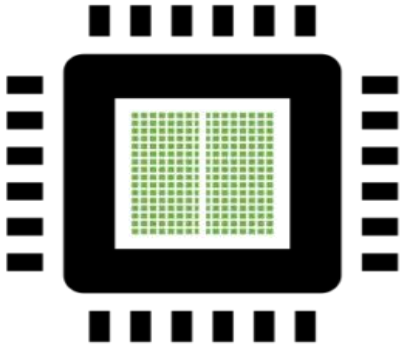
Modern GPU Architecture



CPU vs GPU

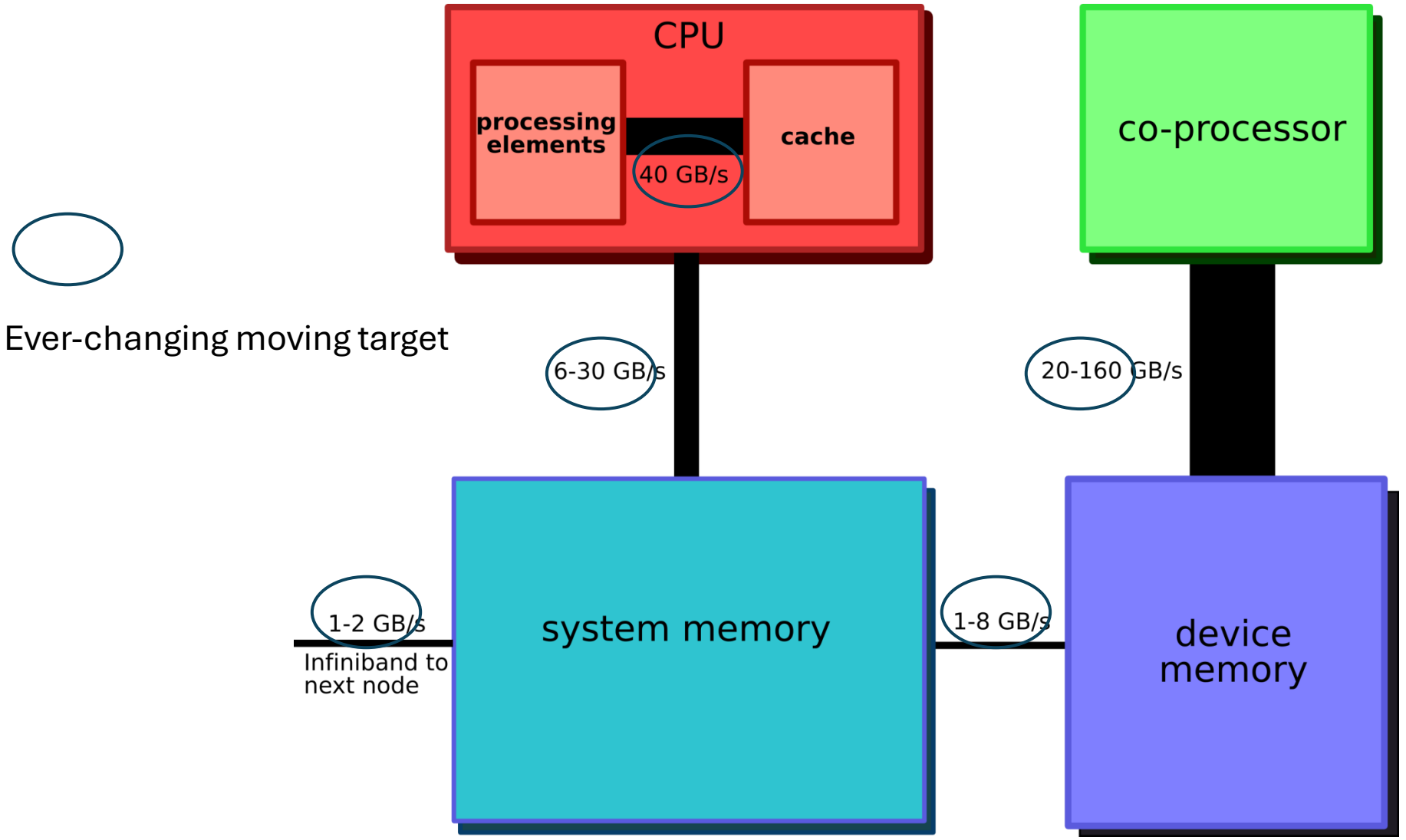






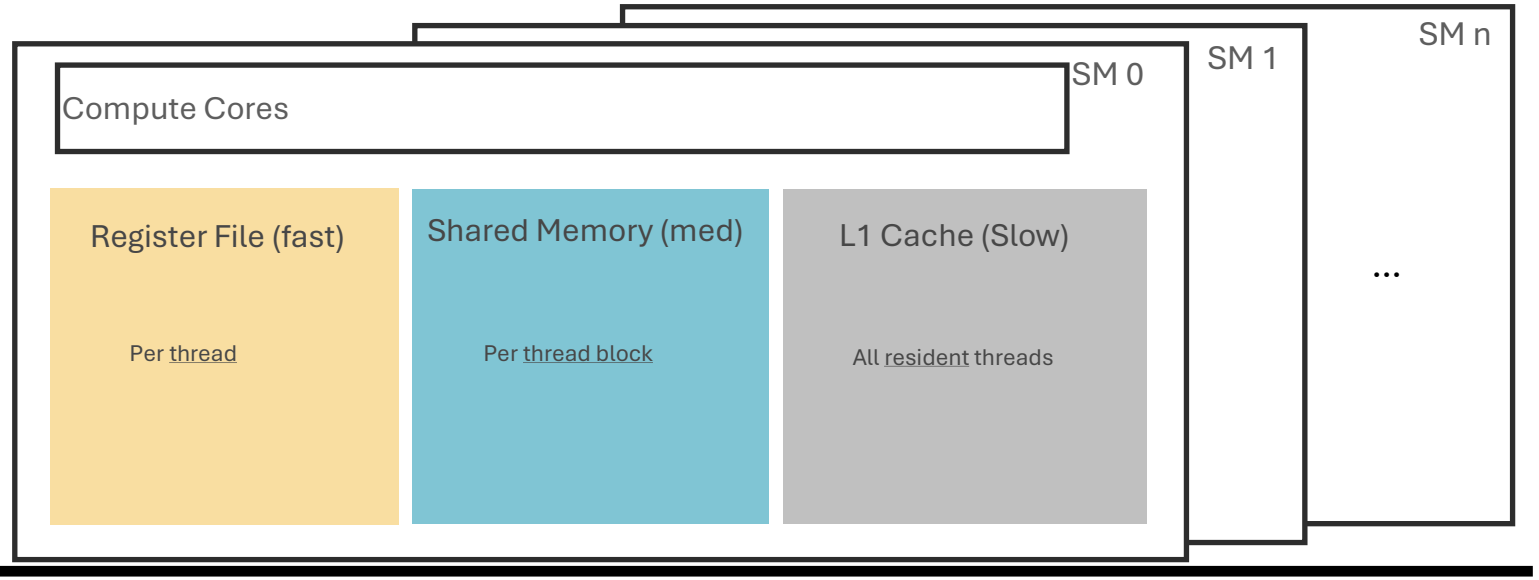
CPU	GPU
Central Processing Unit	Graphics Processing Unit
4-8 Cores	100s or 1000s of Cores
Low Latency	High Throughput
Good for Serial Processing	Good for Parallel Processing
Quickly Process Tasks That Require Interactivity	Breaks Jobs Into Separate Tasks To Process Simultaneously
Traditional Programming Are Written For CPU Sequential Execution	Requires Additional Software To Convert CPU Functions to GPU Functions for Parallel Execution

Organization of a CPU-GPU system

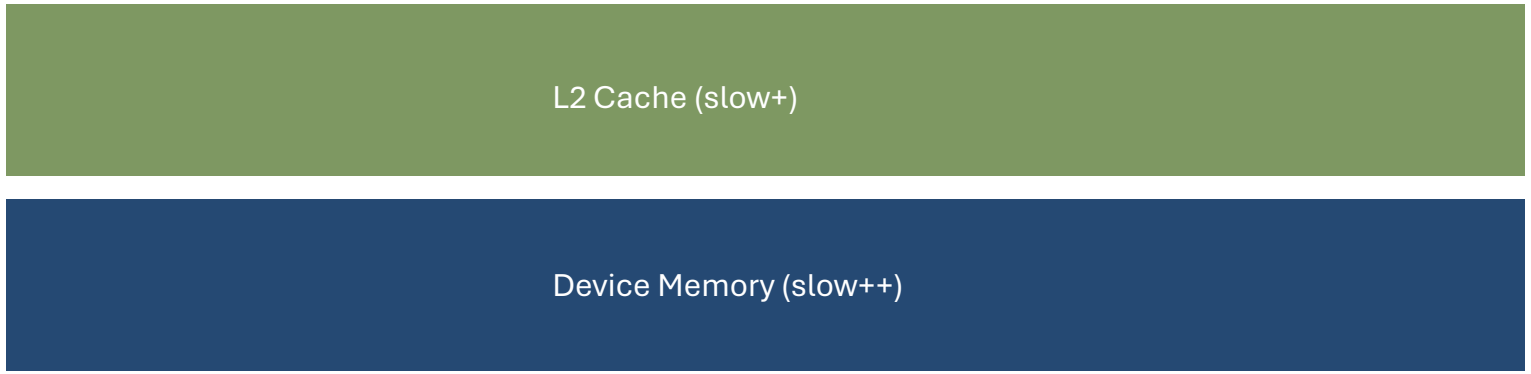


Memory Spaces in GPU

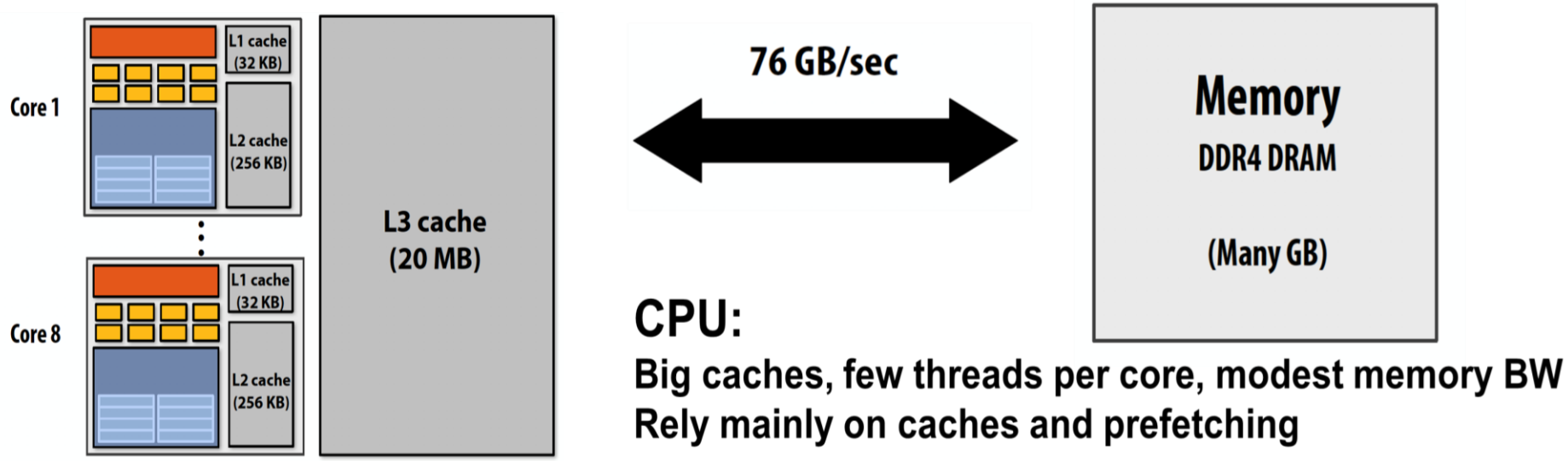
On-chip:
- Register file
  - Usage determined by compiler
  - Spills go to local memory
- Shared memory, i.e. scratchpad
  - Programmer managed
  - Bank conflicts
- L1 cache



Off-chip:
- L2 cache
  - Bandwidth filter for DRAM rather than reducing latency as in CPUs
- Device memory (DRAM)
  - Several spaces: global memory, texture memory, local memory
  - Different spaces have different caching policies



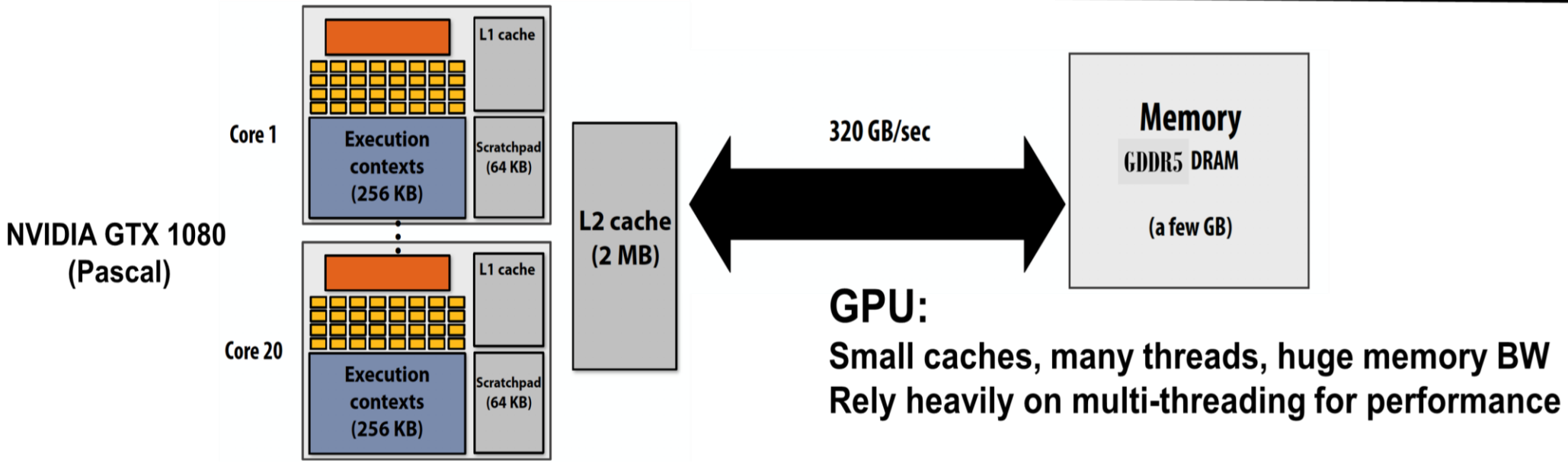
CPU vs GPU : Memory Hierarchy



CPU:

Big caches, few threads per core, modest memory BW

Rely mainly on caches and prefetching



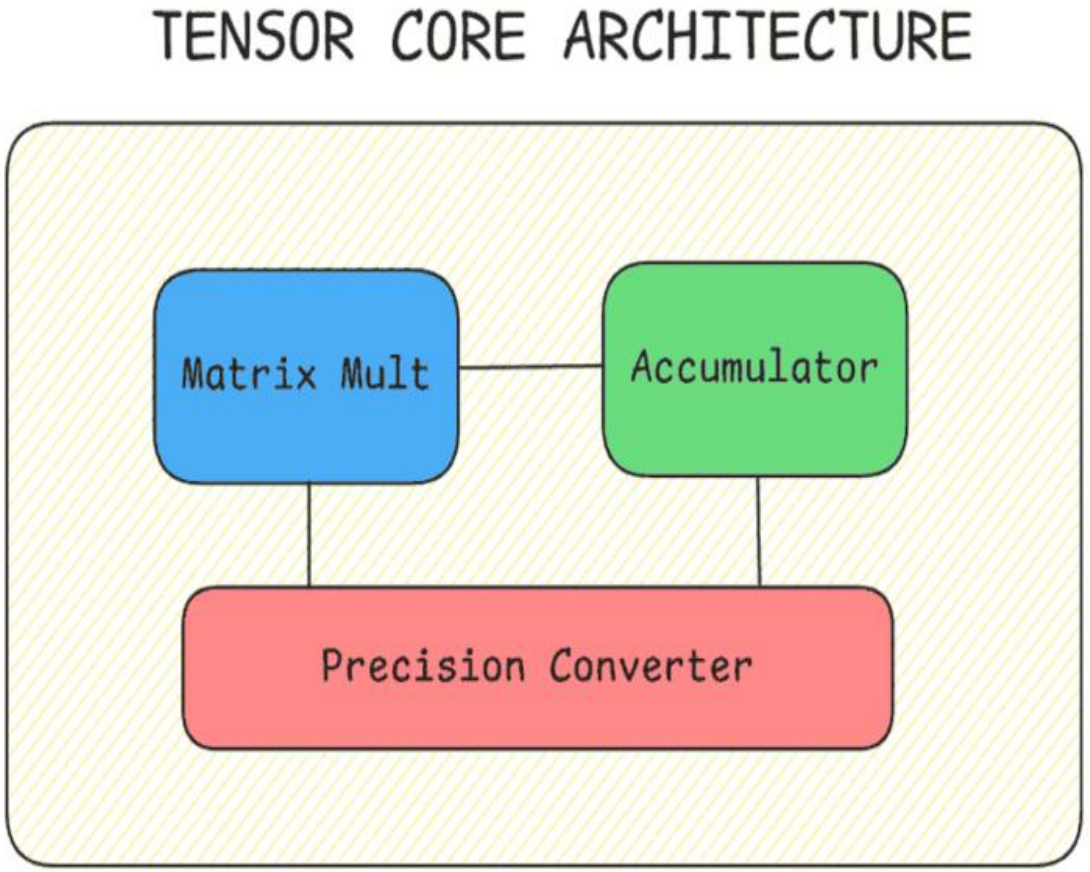
GPU:

Small caches, many threads, huge memory BW

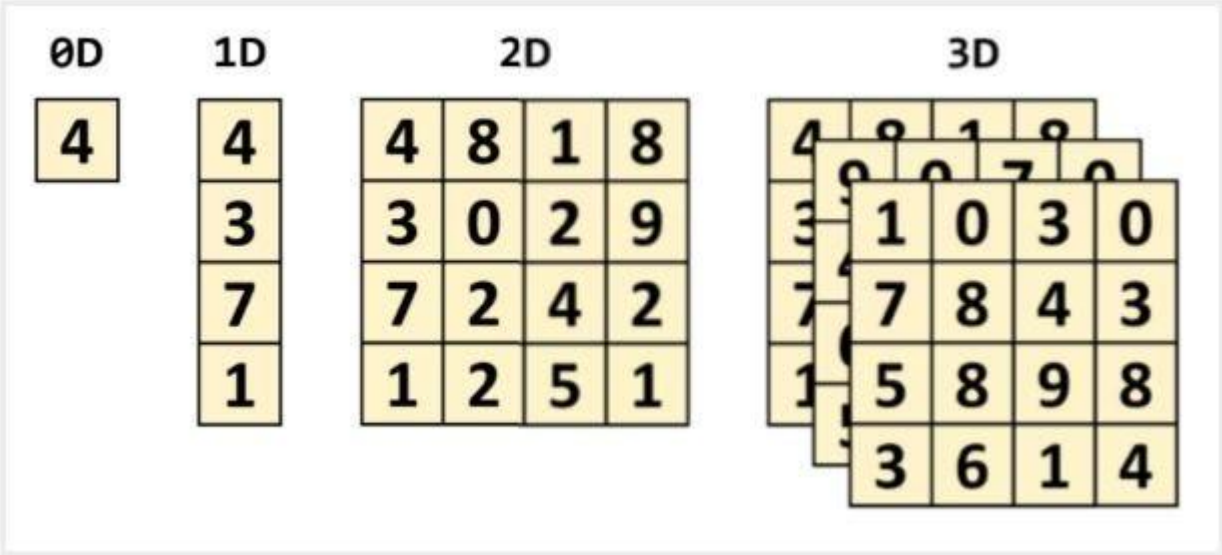
Rely heavily on multi-threading for performance

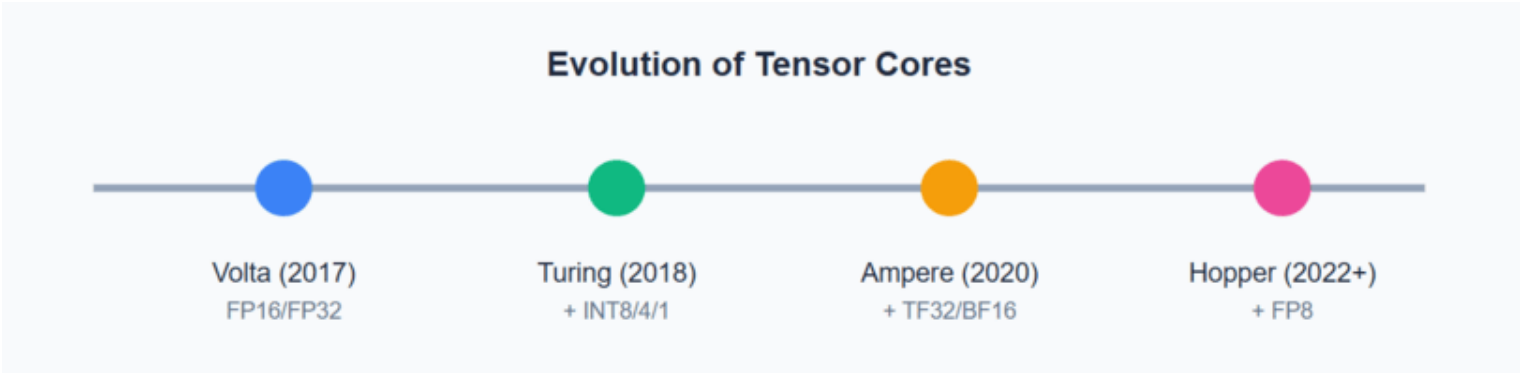
Tensor Core in GPU

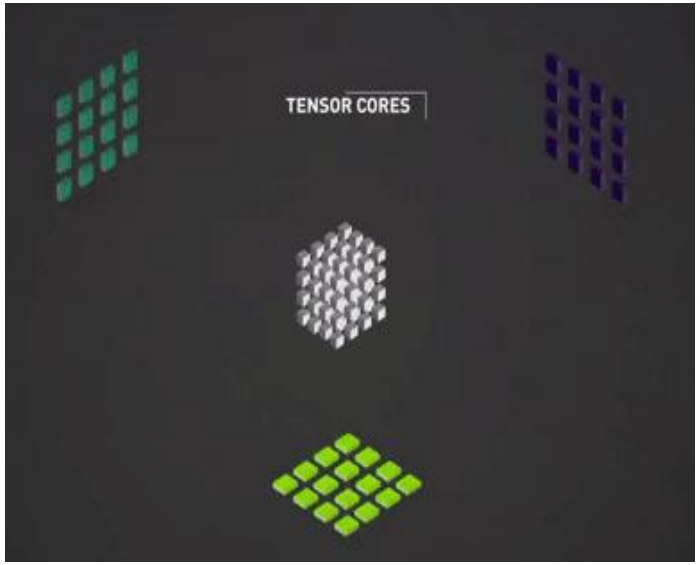
- Specialized processing units within NVIDIA GPUs to accelerate tensor operations
- Engineered to perform matrix and convolution operations
- Accelerate matrix operations
- Optimize large-scale matrix operations
- Improve precision and accuracy through mixed-precision operations
- Future-proof investment for AI infrastructure

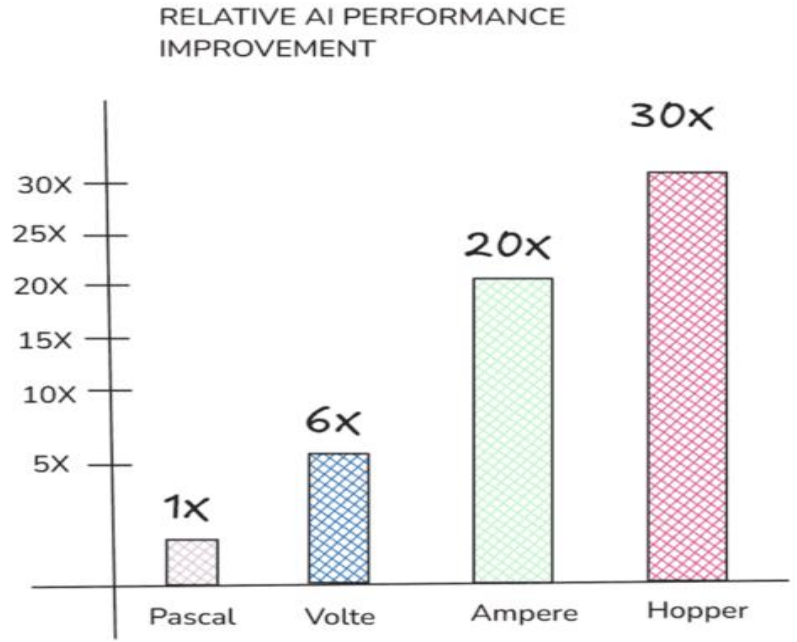


Tensor Core in GPU









Tensor Core in GPU

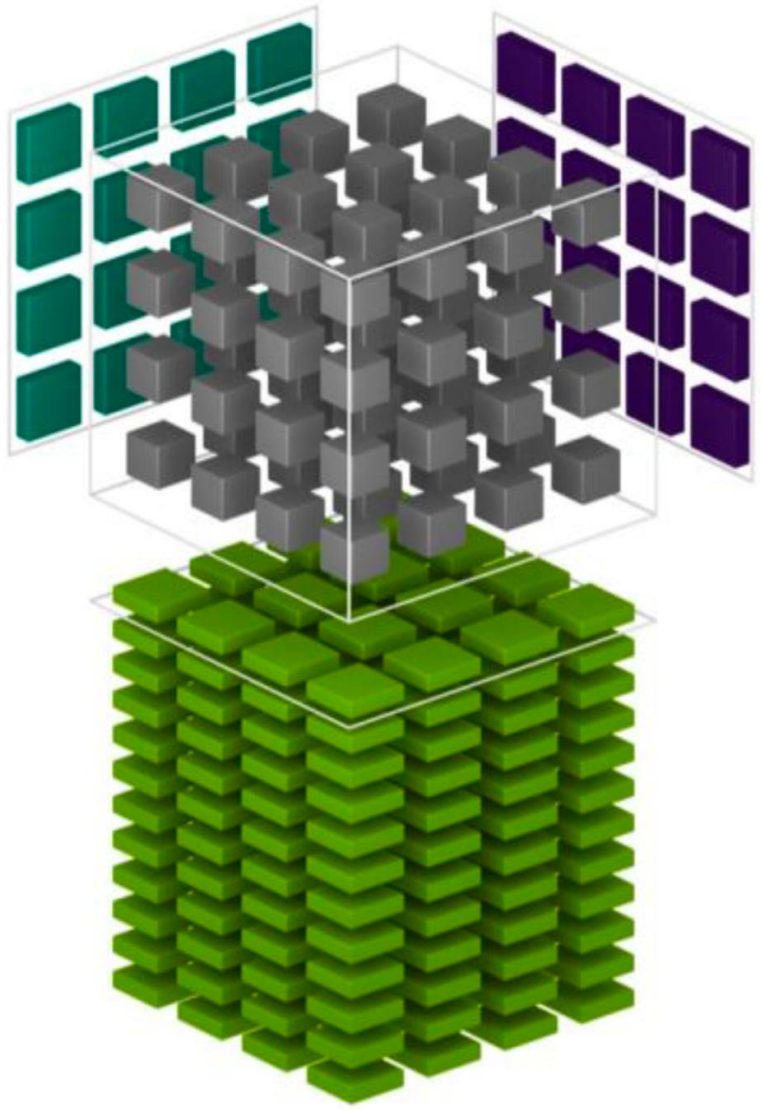
	Tensor cores	**CUDA cores**
Functionality	Specialized hardware designed for deep learning tasks	General-purpose parallel processors
Performance	Optimized for matrix operations, providing faster performance in deep learning workloads	Provide high performance for a wide range of computational tasks
Flexibility	Specifically designed for AI and deep learning applications	Can be utilized for various computational tasks beyond deep learning
Programming	Requires utilizing specialized libraries and frameworks like TensorFlow and PyTorch	Can be programmed using CUDA programming language

NVIDIA GPU's

Platform	CUDA Cores	Tensor Cores
HGX A100	55,296	3,456
HGX H100	135,168	4,224
HGX B200	160,000+	5,000+

Architecture	HGX A100	HGX H100	HGX H200	HGX B100	HGX B200
	8 x A100 SXM	8 x H100 SXM	8 x H200 SXM	8 x B100 SXM	8 x B200 SXM
	Ampere	Hopper		Blackwell	
VRAM Size	640GB	1.1TB		1.44TB/1.5TB	
VRAM Bandwidth	8 x 2TB/s	8 x 3.35TB/s	8 x 4.8TB/s	8 x 8TB/s	8 x 8TB/s
FP16 (FLOPS)	2.4P	8P	8P	14P	18P
INT8 (OPS)	4.8P	16P	16P	28P	36P
FP8 (FLOPS)	X	16P	16P	28P	36P
FP6 (FLOPS)	X	X	X	28P	36P
FP4 (FLOPS)	X	X	X	56P	72P
GPU-to-GPU Bandwidth	600GB/s	900GB/s		1.8TB/s	
NVLink Bandwidth	4.8TB/s	7.2TB/s		14.4T/s	
Ethernet	200Gb/s	400Gb/s + 200Gb/s		2 x 400Gb/s	
InfiniBand	8 x 200Gb/s	8 x 400Gb/s		8 x 400Gb/s	
GPU Power Consumption	3.2kw	5.6kw		5.6kw	8kw
Total Power Consumption	6.5kw	10.2kw		10.2kw	14.3kw
Notes	ConnectX-6 NIC	ConnectX-7 NIC		BlueField-3 DPU ConnectX-7 NIC	

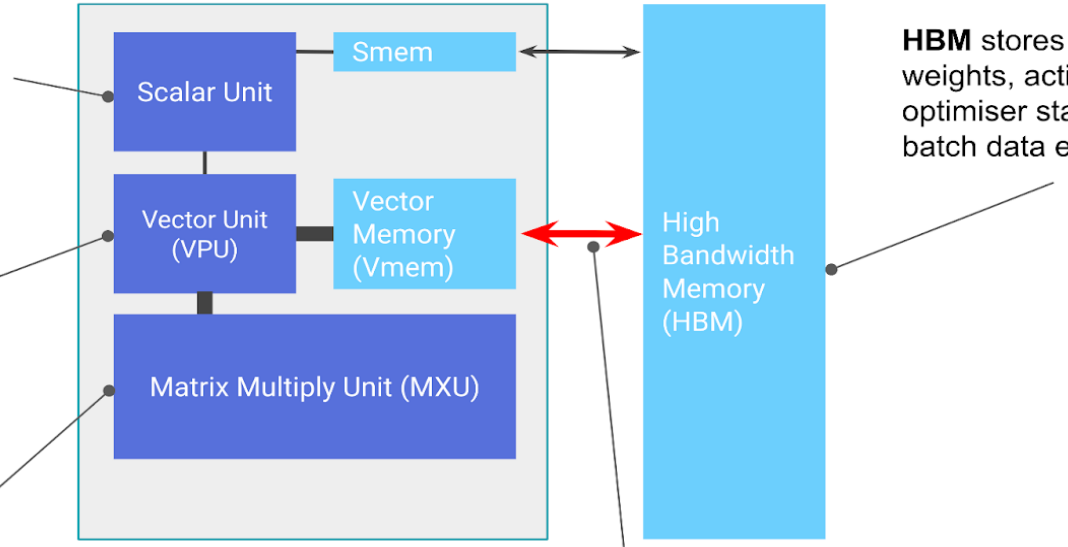
Tensor Processing Unit (TPU)



The **Scalar Unit** sort of acts like a CPU 'dispatching' instructions to the VPU and MXU

The **VPU** performs elementwise operations (e.g. activations), loads data into the MXU

The MXU performs matrix multiplications - and is therefore our driver of chip FLOP/s.

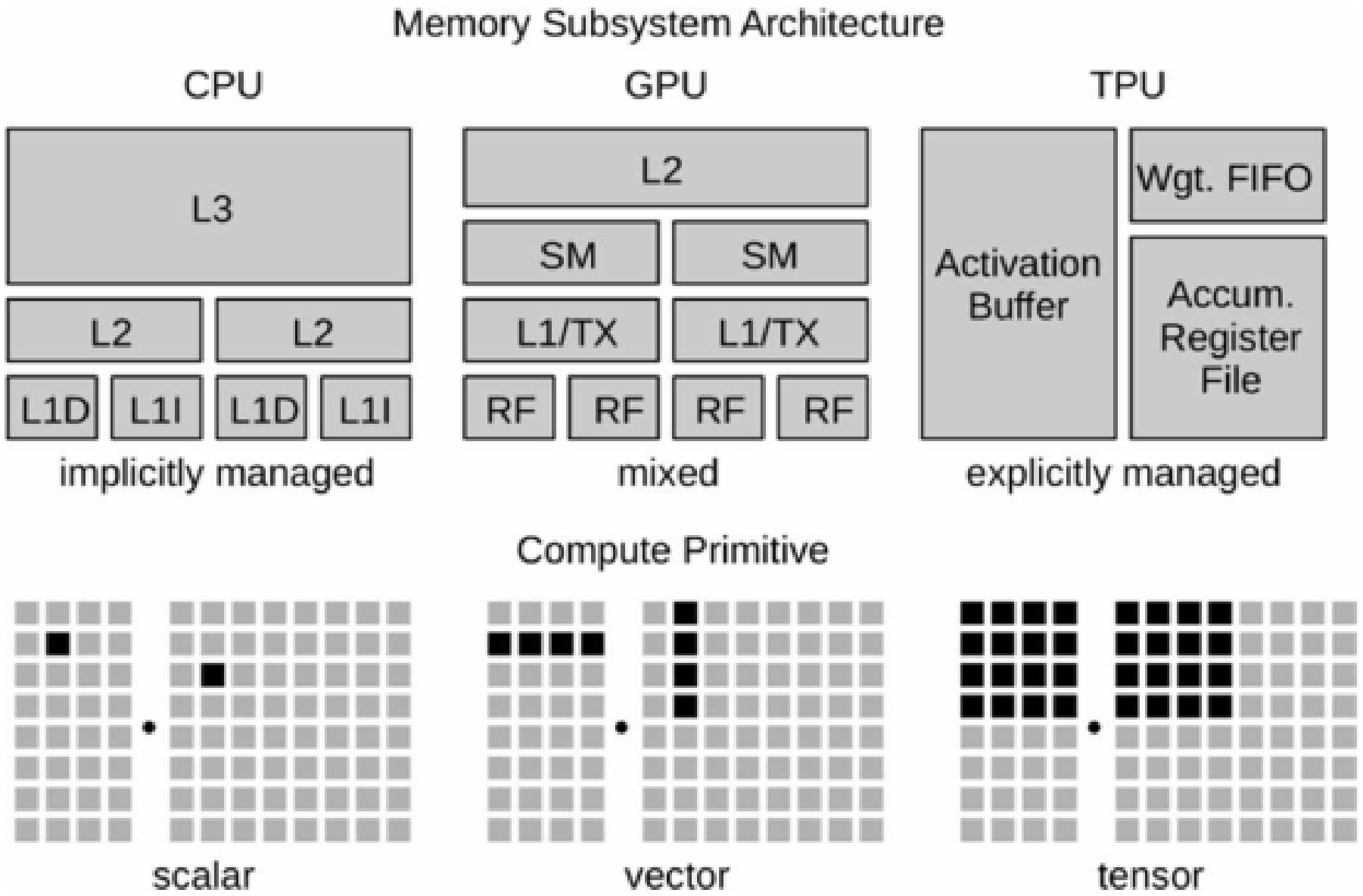


Abstract layout of a TPU TensorCore.

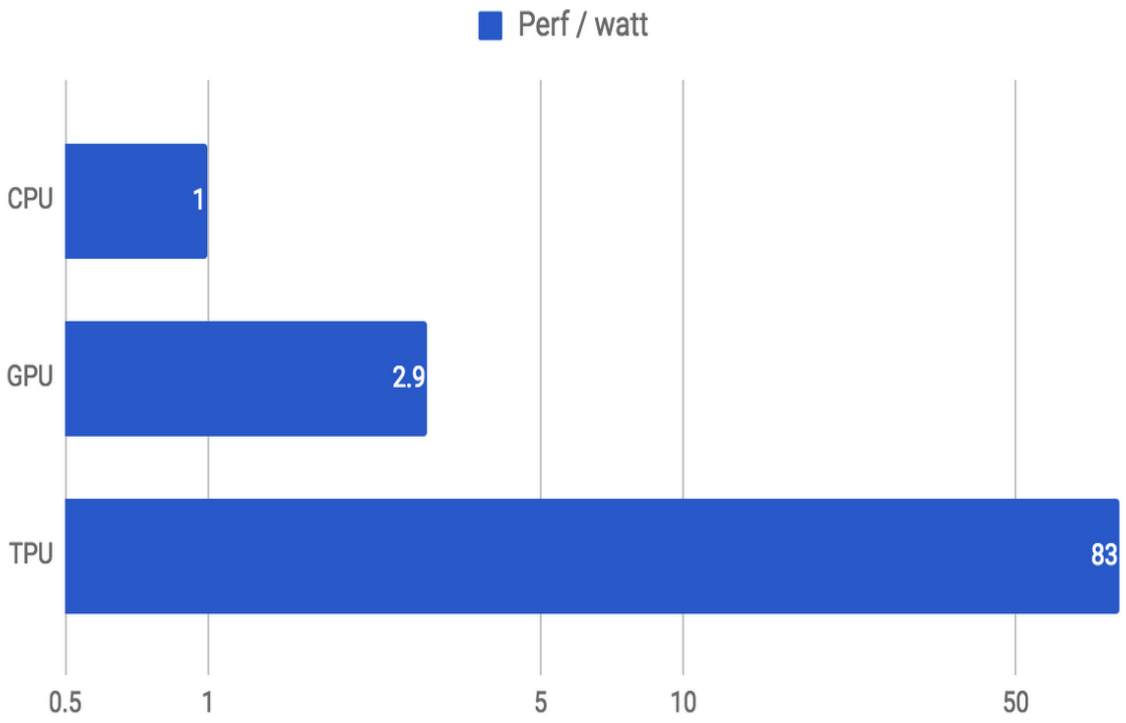
HBM stores the weights, activations, optimiser states, new batch data etc

HBM bandwidth: determines how fast data goes to and from the computational elements

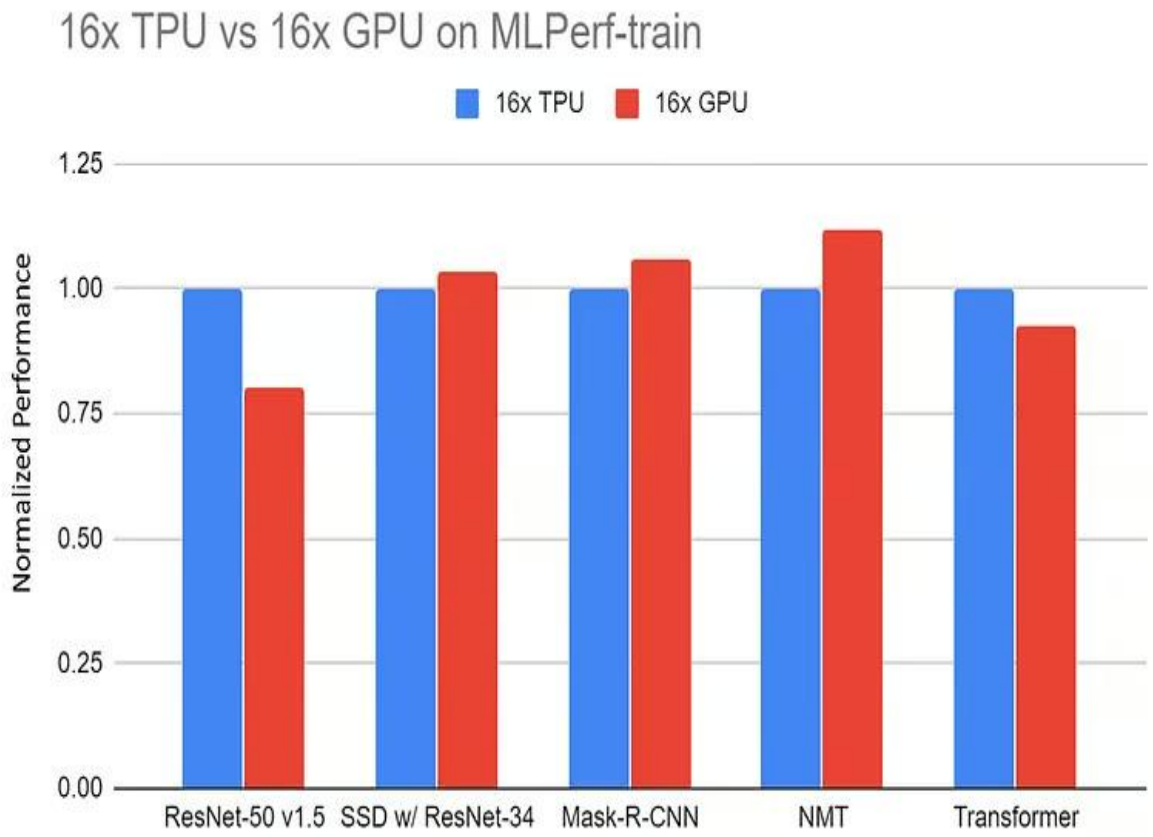
Tensor Processing Unit (TPU)



Tensor Processing Unit (TPU)



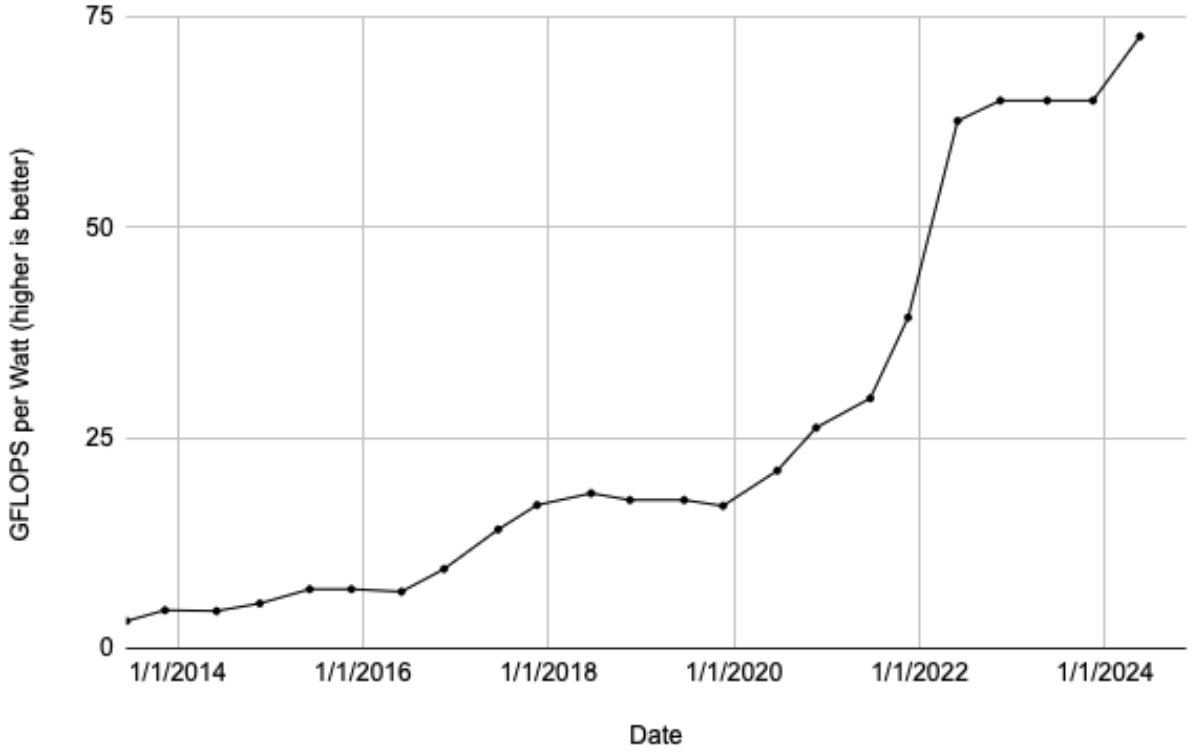
Matrix multiply performance/watt (in log scale)(Incremental, weighted mean)



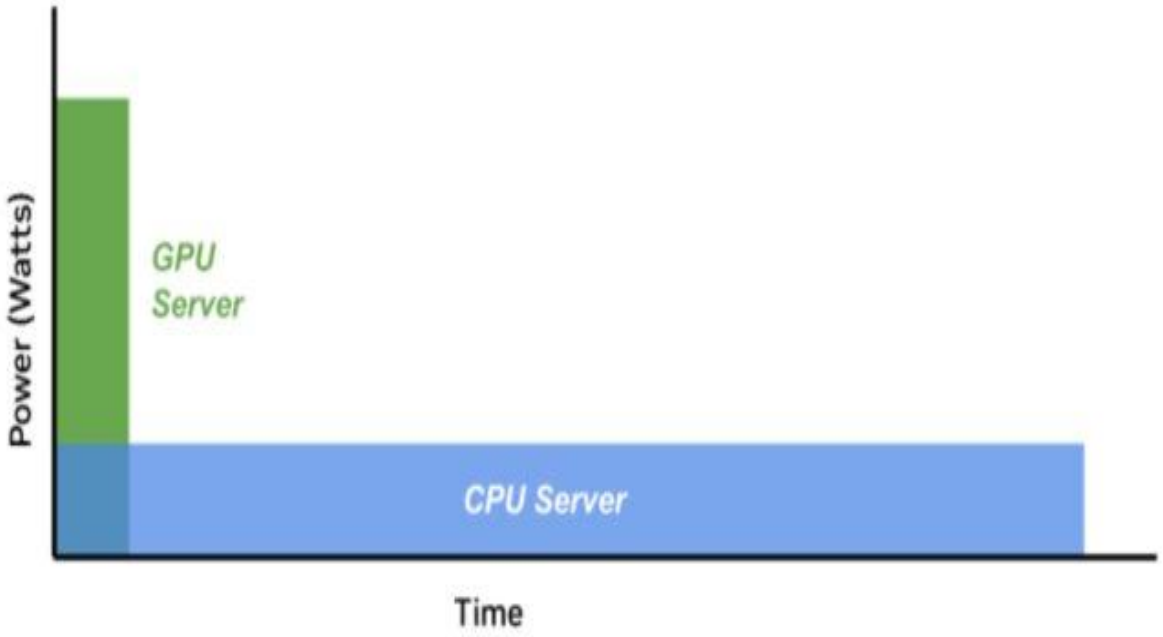
16x GPU server (DGX-2H) vs 16x TPU v3 server normalized performance on MLPerf-train benchmarks.

Feature	GPUs	TPUs
Computational Architecture	Thousands of small, efficient cores for parallel processing	Prioritize tensor operations, specialized architecture
Performance	Versatile, excel in various AI tasks, including deep learning and inference	Optimized for tensor operations, often outperform GPUs in specific deep learning tasks
Speed and Efficiency	E.g., 128 sequences with BERT model: 3.8 ms on V100 GPU	E.g., 128 sequences with BERT model: 1.7 ms on TPU v3
Cost	NVIDIA A100: $10,000 - $15,000/unit, $2.93/hour	Google Cloud TPU v3: $4.50/hour; TPU v4: $8.00/hour
Availability	Widely available from multiple vendors (NVIDIA, AMD, Intel), for consumers and businesses	Mainly accessible through Google Cloud Platform (GCP)
Ecosystem and Development Tools	Supported by many frameworks (TensorFlow, PyTorch, Keras, MXNet, Caffe), extensive libraries (CUDA, cuDNN, RAPIDS)	Integrated with TensorFlow, supports JAX, optimized by TensorFlow XLA compiler
Scalability in Enterprise Applications	Scalable for large AI projects, on-premises or cloud (AWS, Azure), high memory bandwidth, parallel processing	Tightly integrated into cloud infrastructure (GCP), on-demand scalability, managed services for deploying AI models

Energy Efficiency



Energy-efficiency gains over time for the most efficient supercomputer on the TOP500 list. Source: TOP500.org



Accelerated systems use parallel processing on GPUs to do more work in less time, consuming less energy than CPUs.

Thank You