

Building Blocks: Hardware

Processors, Cores, Memory and Accelerators

Partners



Funding



Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

Outline

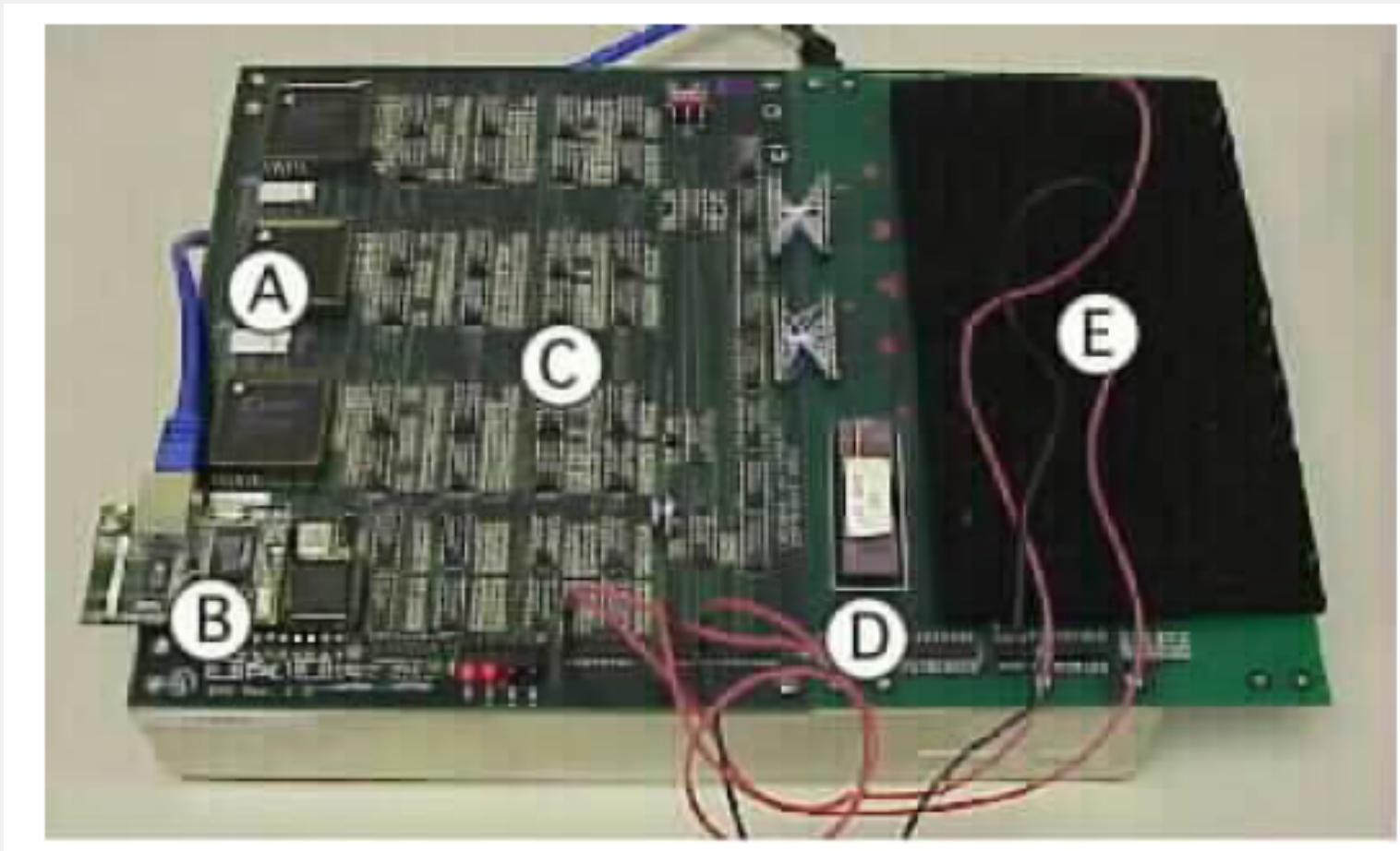
Building Blocks of HPC systems

- Processors
- Memory
- Interconnect
- Storage
- What does performance depend on?
 - Limits to performance
- Evolution of the Processor
 - Moore's Law
 - Parallelism in Hardware
 - Vector instructions
 - Hardware threads
 - Multicore
- Accelerators (GPU)
 - What are they good for?

What is a computer?



What is a computer?

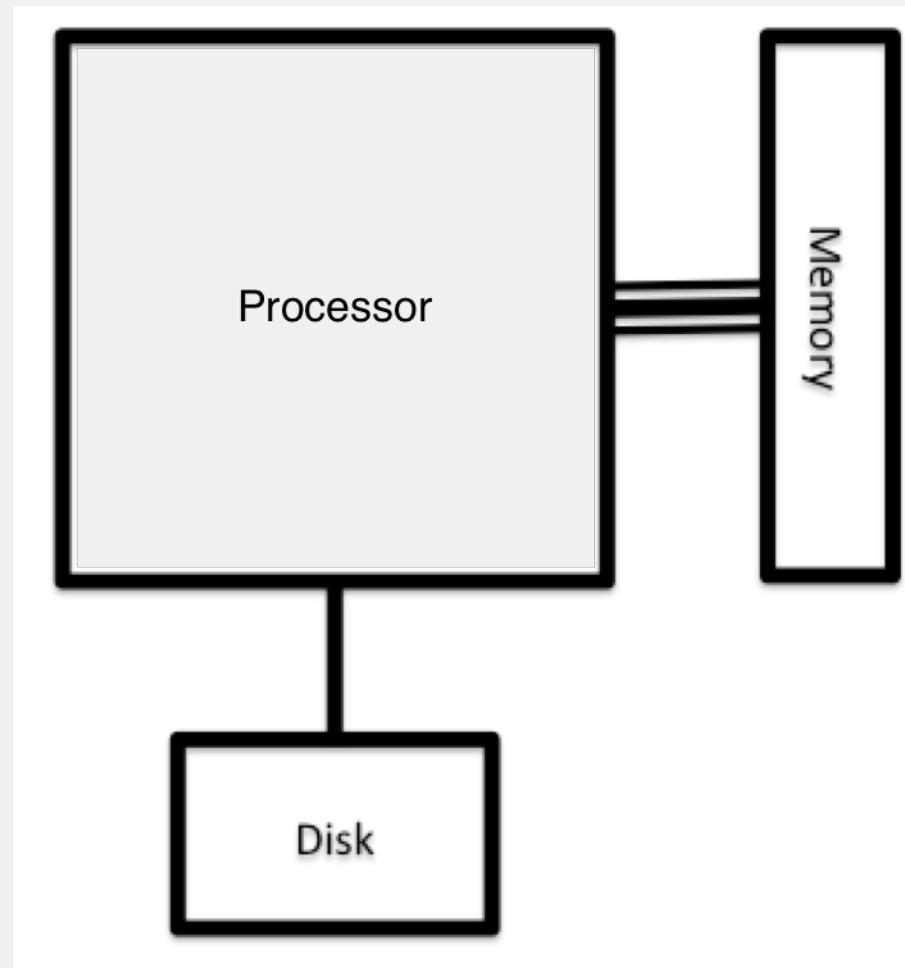


Computational Building Blocks

Four principal hardware technologies make up HPC systems:

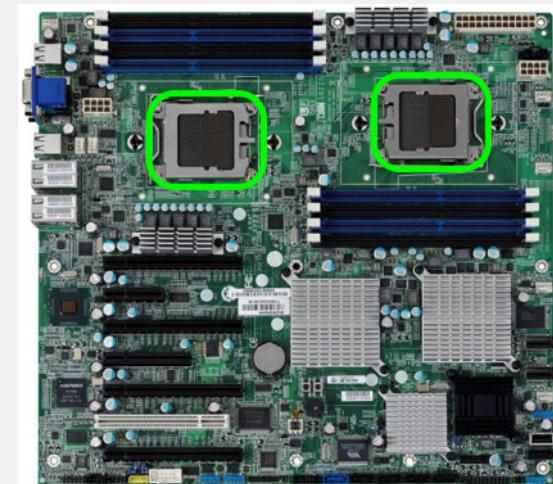
- Processors (& accelerators)
 - to calculate
- Memory
 - for temporary storage of data
- Interconnect
 - enabling processors to talk to each other (and the outside world)
- Storage
 - disks for storing input/output data, other for long term archiving of data
- We will focus on the first two of these

Anatomy of a Computer



What do you mean, “processor”?

- Terminology gradually varied over time and in different contexts (hardware, software)
- Usually taken to mean “the thing you plug in to a socket on the motherboard” (e.g. two processor sockets below)
- ARCHER2: each node has two processors (each in its own socket)
- Your laptop: one multicore processor

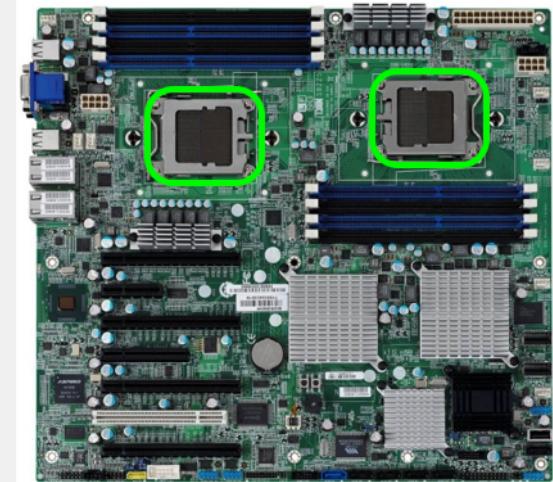


What do you mean, “CPU”?

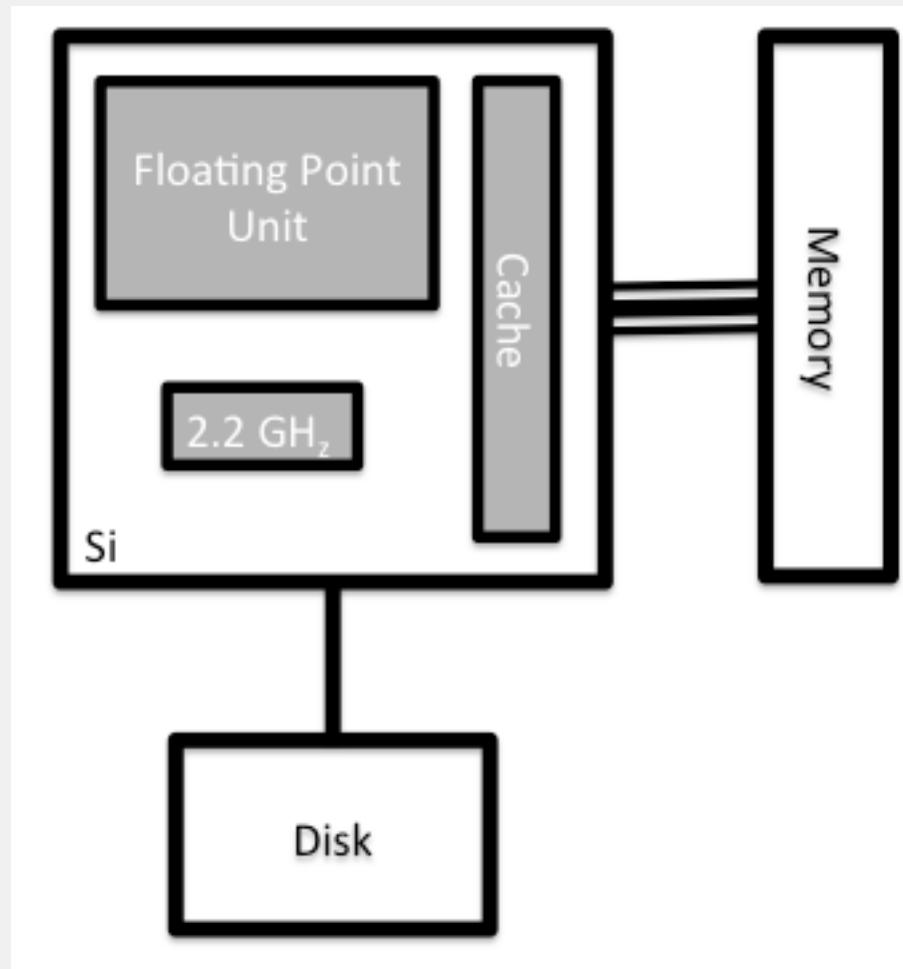
Ambiguous!

Nowadays may refer to:

- Entire multicore processor
 - Usage from single-core processor era
- A (physical) processor core
- A logical core
 - Anything that looks to the Operating System like an independently addressable and usable core to assign threads or processes to execute on
 - E.g. physical cores, also logical subunit(s) of physical cores



Anatomy of a Computer (single-core processor)



Processors

- Basic functionality
 - execute instructions to perform arithmetic operations (integer and floating point)
 - load data from memory and store data to memory
 - decide which instructions to execute next
- Mathematical operations performed on values in registers (local storage in the processor)
 - Moving data between memory and registers = load and store instructions
 - Separate integer and floating point registers
 - Typical size ~100 values
- Basic characteristics:
 - Clock speed
 - Peak floating point capability

Functional Units

Functional units are the basic building blocks of processors

- Number and type of units varies with processor design.

Core units for any processor include:

- **Instruction unit**

- Responsible for fetching, decoding and dispatching of instructions.
- Fetches instruction from instruction caches
- Decodes instruction.
- Sends the instructions to the appropriate unit
- May also be responsible for scheduling instructions (see later).

Functional Units

Core units for any processor include:

- **Integer unit**
 - Handles integer arithmetic
 - Integer addition, multiplication and division
 - Logical ops (and, or, shift etc.)
 - Also known as arithmetic and logic unit (ALU)
- **Floating point unit**
 - Handles floating point arithmetic
 - Addition, multiplication, division.
 - Usually the critical resource for HPC
 - Machines sold by peak floating point operations per second (flop/s)

Functional Units

Core units for any processor include:

- **Control unit**
 - Responsible for branches and jumps
- **Load/store unit**
 - Responsible for loading data from memory and storing it back.
- **Register file**
 - Local storage in the CPU
 - Accessed by name (not address)
 - Separate files for integers/addresses and floating point

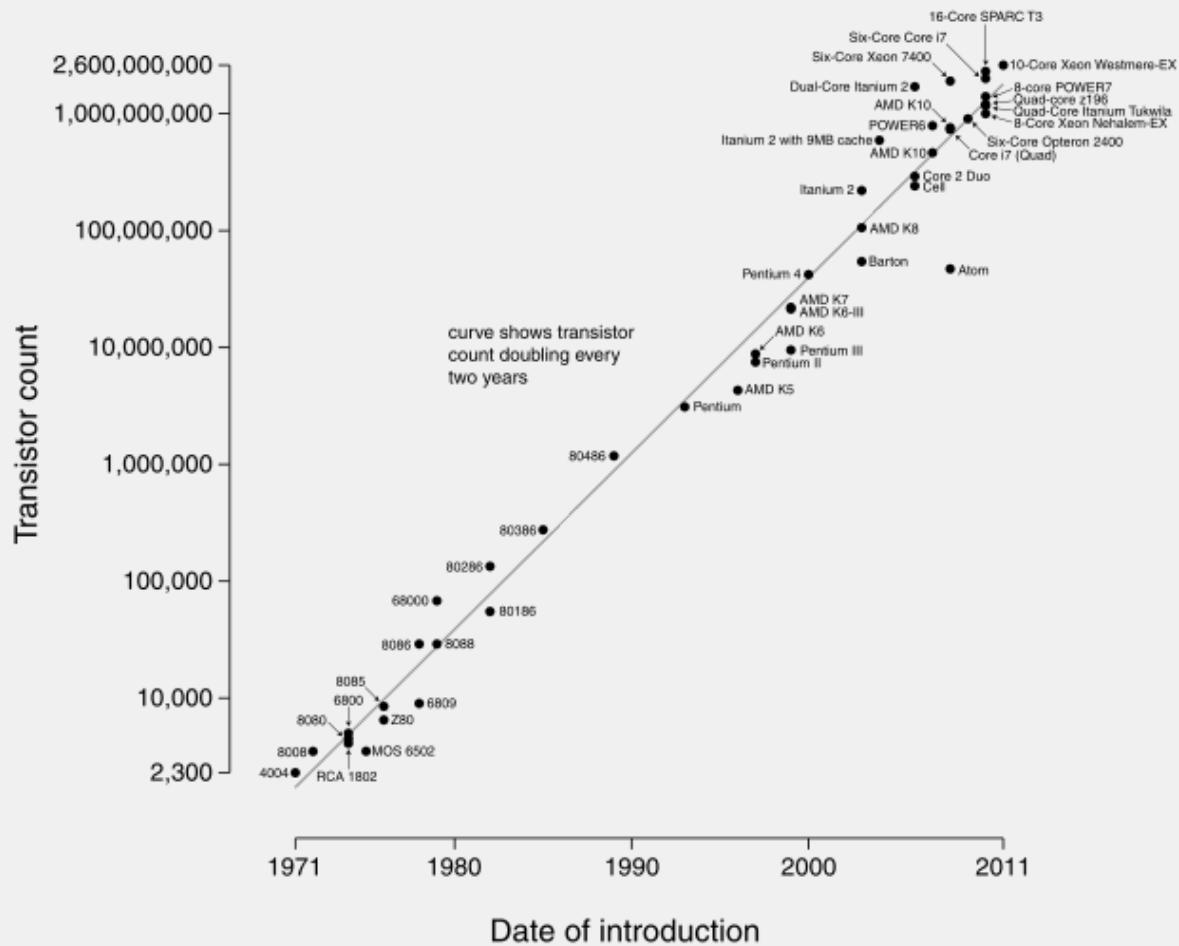
Also memory management, cache controller, bus interface, graphics/multimedia,.....

Moore's Law, Processor Evolution, and Parallelism in Hardware

Moore's Law

Micropocessor Transistor Counts 1971-2011 & Moore's Law

- Number of transistors doubles every 18-24 months
 - enabled by advances in semiconductor technology and manufacturing processes



What to do with all those transistors?

- For over 3 decades (the “good old days”) until early 2000’s
 - processors became more complicated / sophisticated
 - caches became bigger
 - clock speeds increased year on year (100 MHz, 200 MHz, 400MHz, ...)
 - for your program to run faster just wait a year and buy a newer processor
- Clock rate increases as inter-transistor distances decrease
 - so performance doubled every 18-24 months
- Came to a grinding halt about a decade ago
 - reached power and heat limitations
 - who wants a laptop that runs for an hour and scorches your trousers!

Processor performance

- Clock speed determines rate at which instructions are executed
 - modern chips are around 2-3 GHz
 - integer and floating point calculations can be done in parallel
 - can also have multiple issue, e.g. simultaneous add and multiply
 - peak flop rate is just clock rate x no. of floating point operations per clock cycle
- Decades of research and \$\$\$ into hardware innovations, complex design and optimisations
 - Out-of-order execution
 - Assembly code specifies an order of instructions....
 - Hardware chooses to reorder instructions to minimise pipeline stalls
 - Requires some complex bookkeeping to ensure correctness
 - Branch prediction
 - Hardware tries to guess which way the next branch will go
 - Uses a hardware table that tracks the outcomes of recent branches in the code.
 - Keeps the pipeline going and only stalls if prediction is wrong

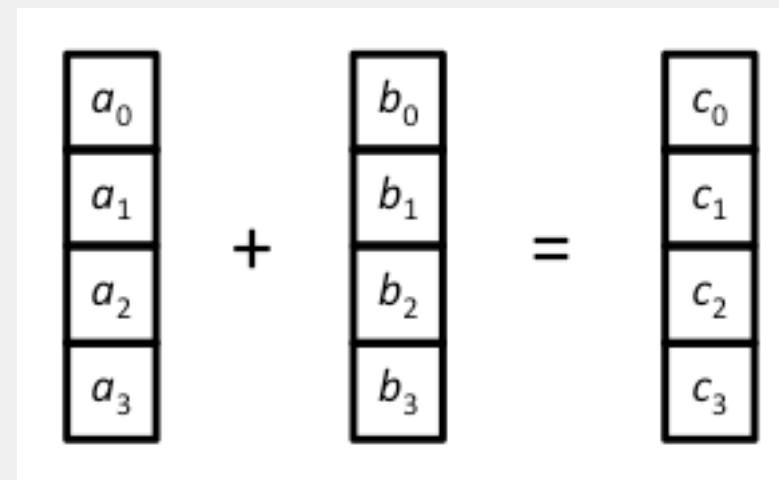
Processor performance

Introduce parallelism into the processor itself

- **Vector instructions (vectorisation) (“SIMD” = “Single Instruction, Multiple Data)**
- **Multiple cores close together on same chip (multicore processor)**
- **Simultaneous Multi-Threading (“SMT”)**

Single Instruction Multiple Data (SIMD)

- For example, vector addition:

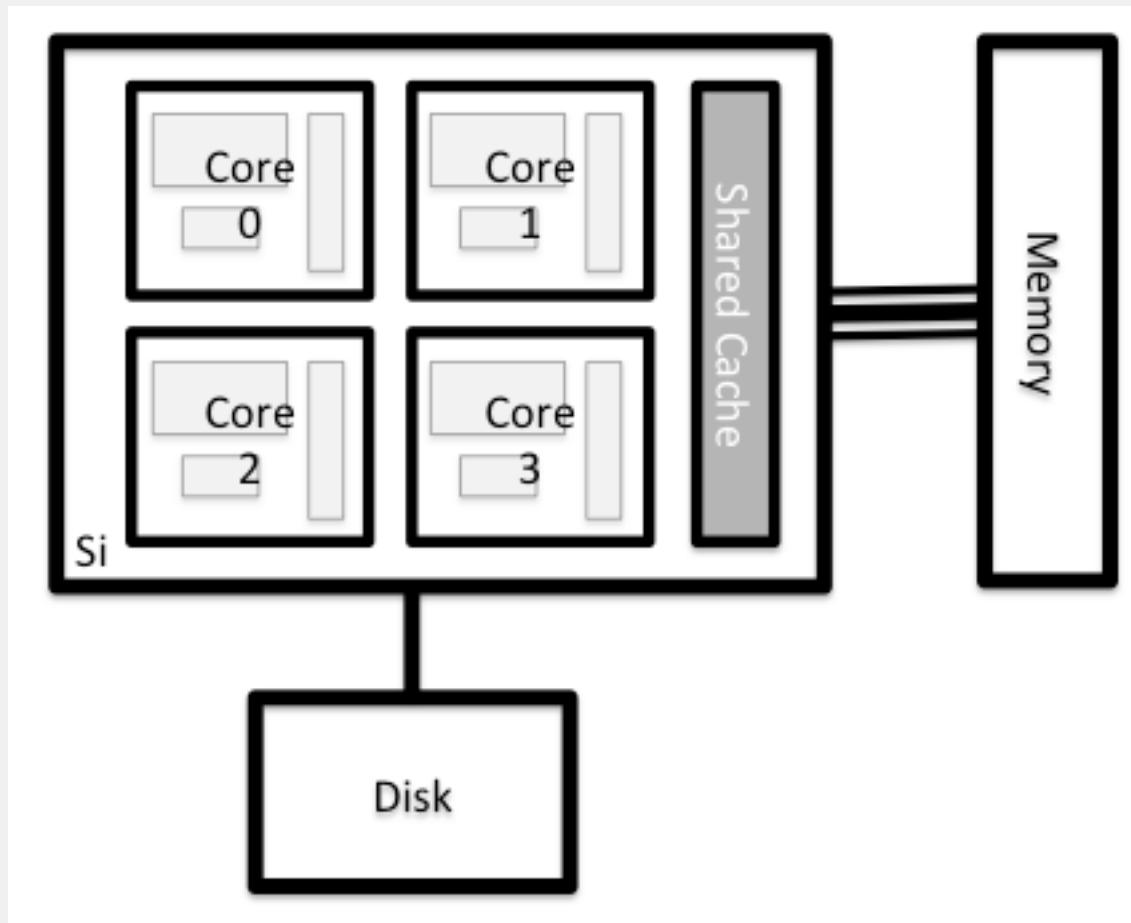


- single instruction adds 4 numbers
- potential for 4 times the performance
- SIMD width (vector width) is the number of operations encoded in a SIMD instruction
 - Often 2, 4 or 8

Multicore

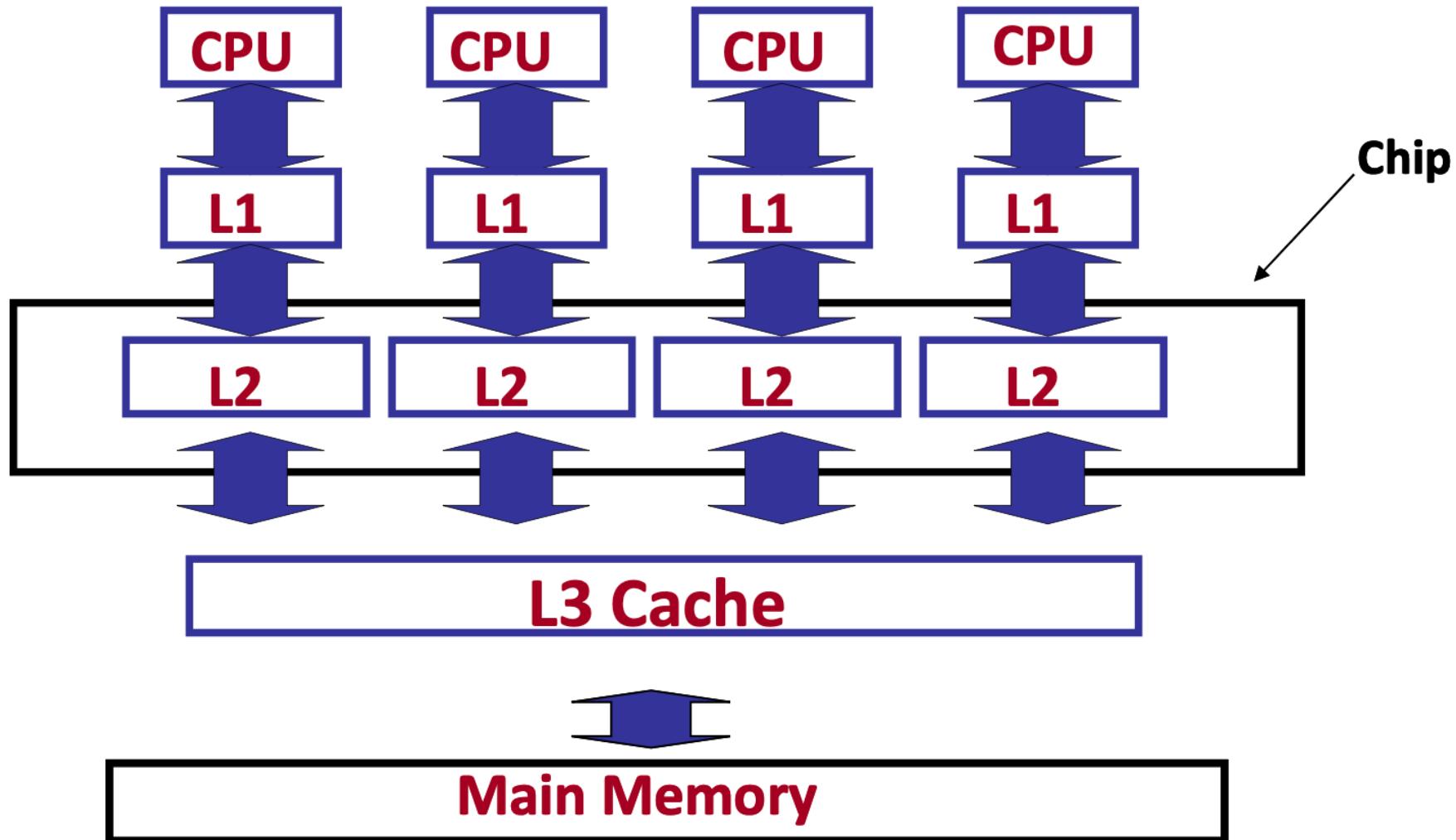
- Twice the number of transistors gives 2 choices
 - a new more complicated processor with twice the clock speed
 - two versions of the old processor with the same clock speed
- Second option is more power efficient
 - and now the only option as we have reached heat/power limits
- Effectively two independent single-core processors
 - ... except they can share cache
 - Commonly called “cores”
 - Cores may also share some functional units

Anatomy of a computer (single processor, multicore)



- Cores share path to memory
 - More cores makes this an *increasing* bottleneck!

Multicore Cache Hierarchy



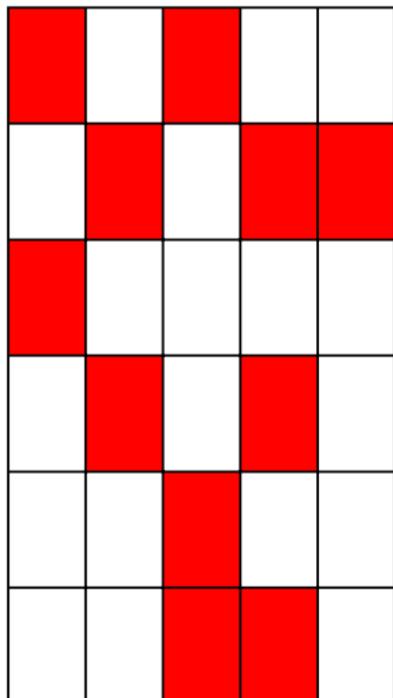
Multicore Cache Hierarchy

- Cores on the same chip can communicate with low latency and high bandwidth
 - reads and writes through shared cache
- Cores share space in the shared cache
 - Possible contention: one thread may suffer capacity and/or conflict issues caused by threads/processes on another core
 - If only single core is running, then it may have access to the whole shared cache
- Cores also share off-chip bandwidth
 - access to main memory

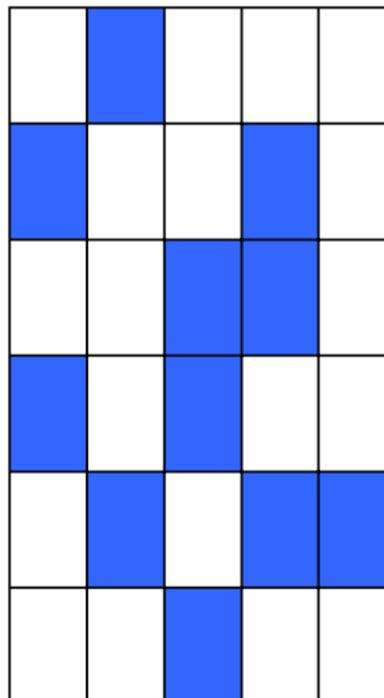
Simultaneous Multi-Threading (SMT)

- Some processors supports running multiple instruction streams simultaneously on the same processor, e.g.
 - stream 1: loading data from memory
 - stream 2: multiplying two floating-point numbers together
- Requires some replication of hardware, but everything else is shared between threads
 - functional units, register file, memory system (including caches)
- “Threading” can be a misnomer - can refer to processes as well as threads
 - These are “hardware threads”, not software threads
 - = ability to execute instructions threads or processes
 - Appear to the Operating System as distinct logical cores
 - For most architectures, two or four threads is all that makes sense
 - Intel Xeon & AMD EPYC Zen2 supports 2-way SMT

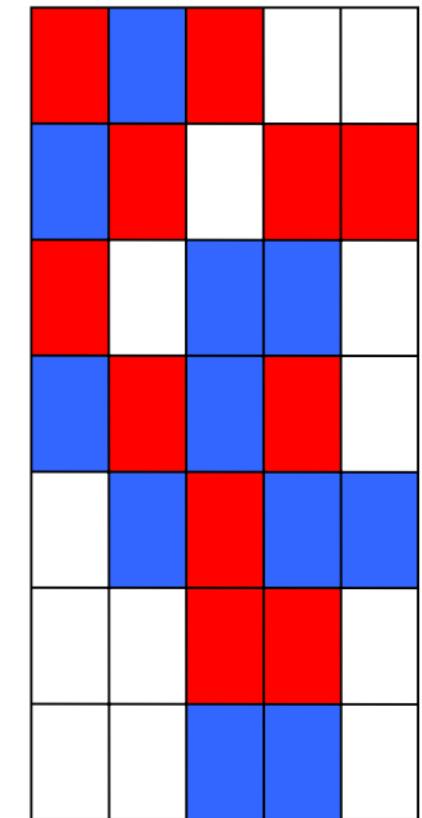
Simultaneous Multi-Threading (SMT)



Two threads on two physical cores



Time



Two threads on one physical core
with SMT enabled

Simultaneous Multi-Threading (SMT)

- How successful is SMT?
 - depends on the application, and how the threads contend for the shared resources.
- In practice, gains seem to be limited to around 1.2 to 1.3 times speedup over a single thread
 - benefits will be limited if both threads are using the same functional units (e.g. FPUs) intensively.
- For some codes, SMT can cause slow down
 - increased contention for memory bandwidth and/or cache space
 - increasing the number of threads/processes can increase overheads such as communication or load imbalance

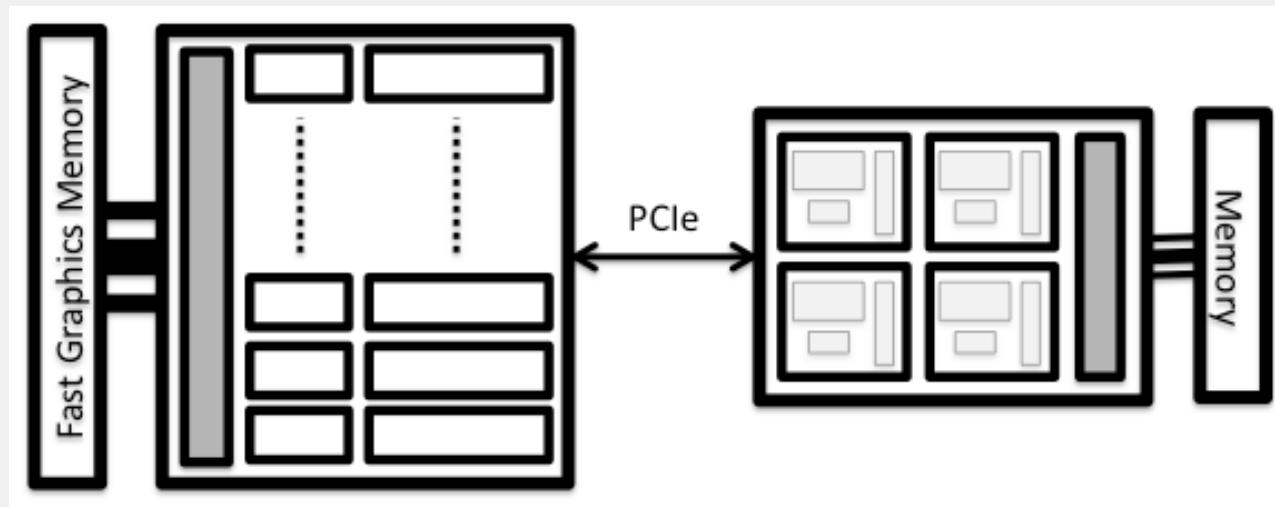
AMD EPYC Zen2 (Rome) Processor

- Proper name: AMD EPYC 7742
- No. of cores = 64
- Clock rate = 2.25 GHz (base)
- 2 FMA floating point vector units
- 256-bit wide AVX vector instructions
 - 4 double precision floating point ops
- Peak 16 flops per clock
= 36 Gflop/s per core
= 2304 Gflop/s per socket
= 4608 Gflop/s per node
- SMT: 2 hardware threads per core

Accelerators

Anatomy

- An *Accelerator* is an additional resource that can be used to off-load heavy floating-point calculation
 - additional processing engine attached to the standard processor
 - has its own floating point units and memory

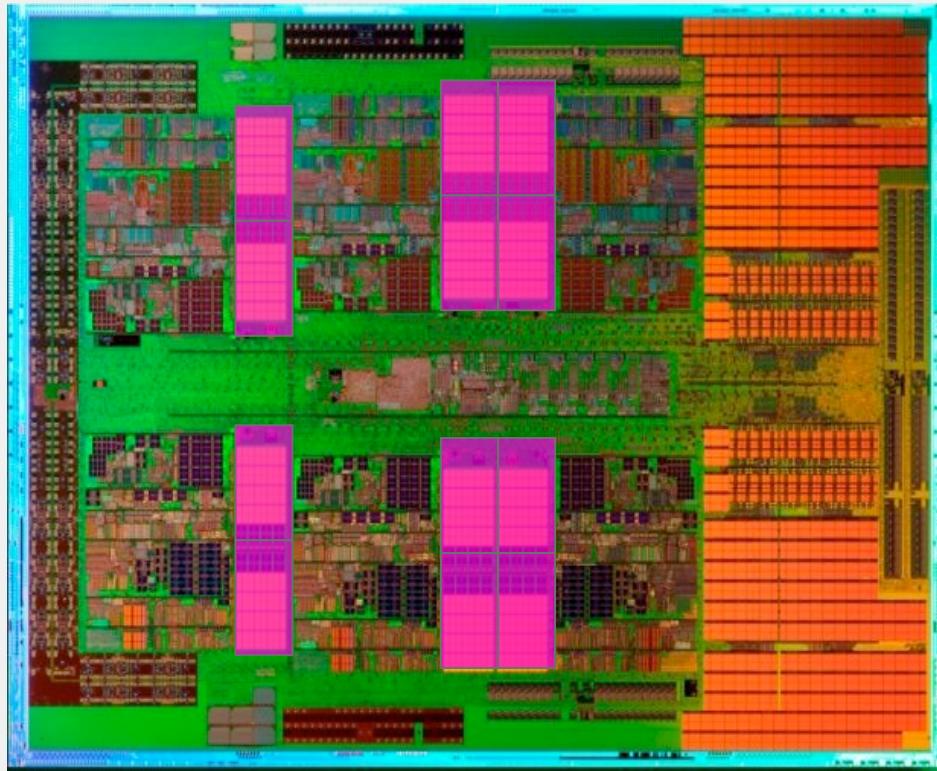


Accelerators

- Very widespread to include on compute nodes alongside the main CPU
 - large numbers of relatively simple cores
 - high memory bandwidth
 - separate memory from CPU
- Much of current interest is focussed on GPGPUs (general purpose graphics processing units)
 - low cost due to high mass market volumes
 - tricky to program
 - significant overheads in moving data to/from GPU memory

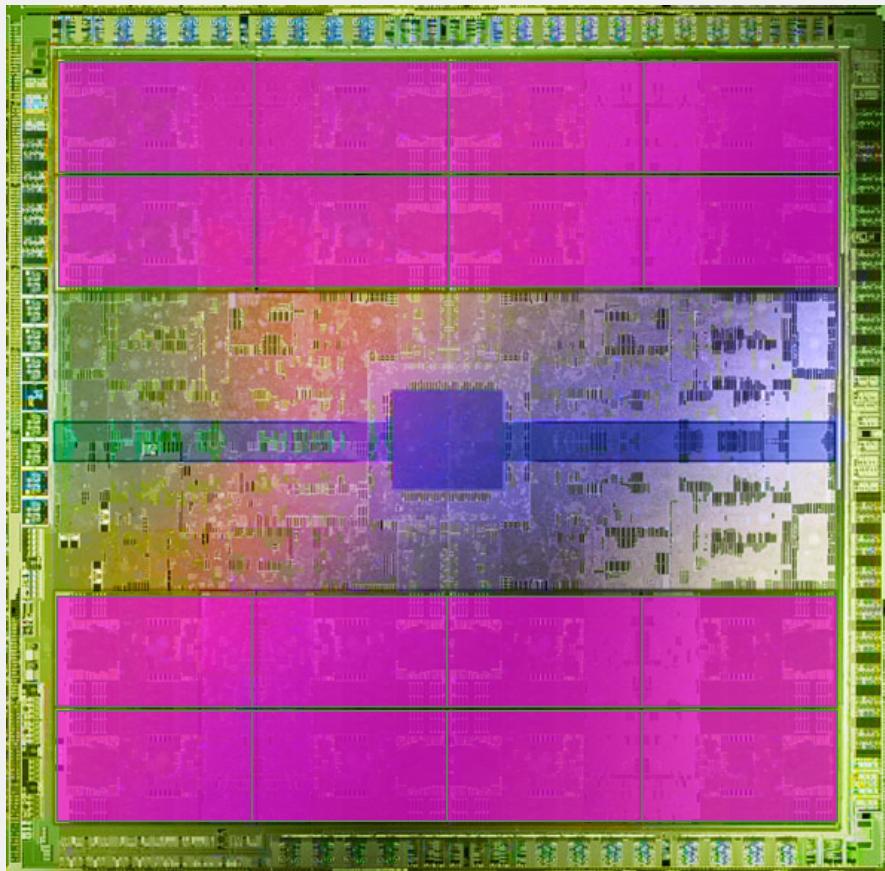
AMD 12-core CPU

- Not much space on CPU is dedicated to computation

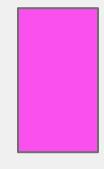


 = compute unit
 (= core)

NVIDIA Fermi GPU



- GPU dedicates much more space to computation
 - At expense of caches, controllers, sophistication etc

 = compute unit
(= SM
= 32 CUDA cores)

Memory



Data Access Bottlenecks

Performance depends on getting data to the processor quickly

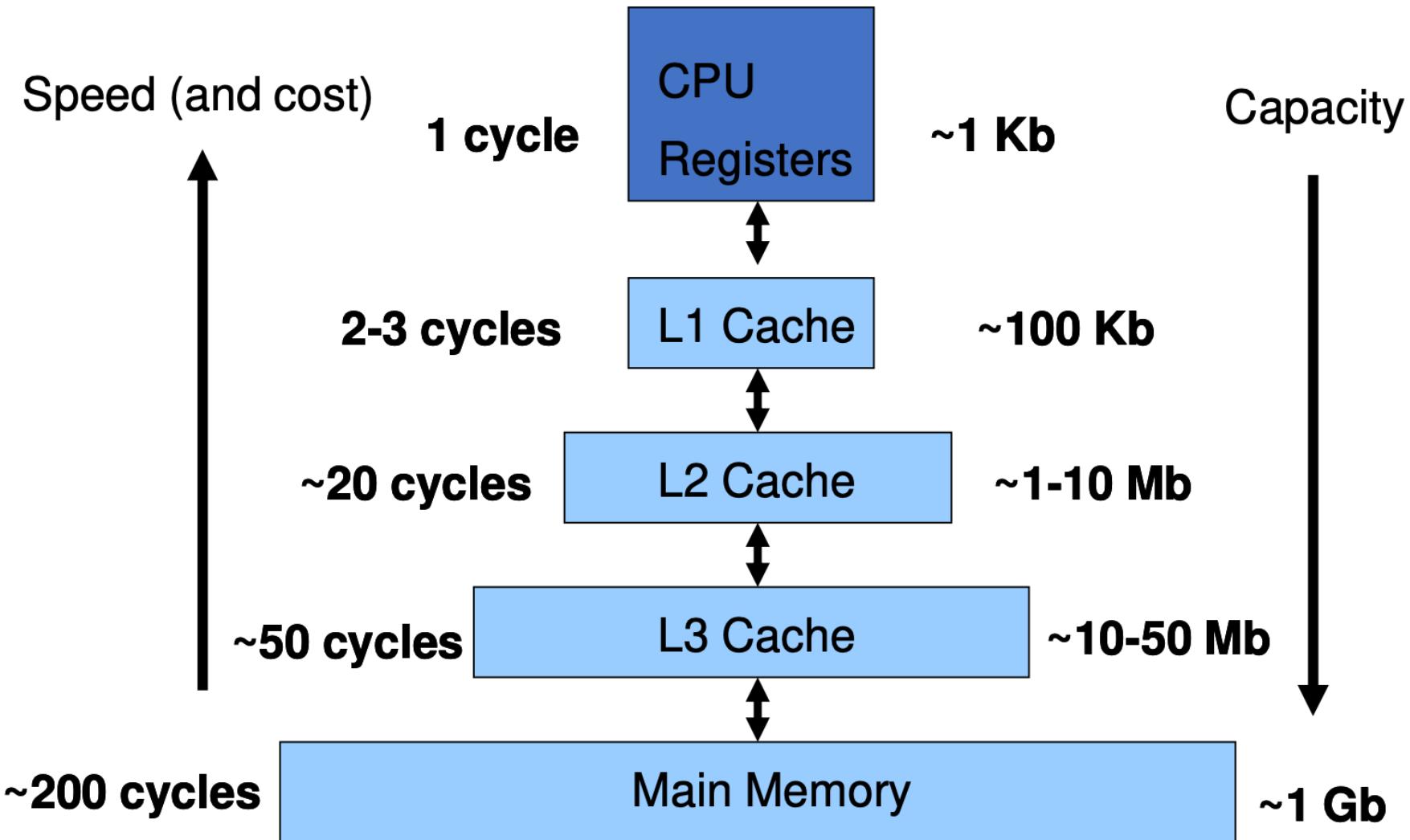
- Latency: time delay until data starts arriving
- Bandwidth: amount of data arriving per second

- Disk access is slow
 - few hundred MB/s
 - significantly higher latency than memory

- Memory access is faster than disk
 - Large enough memory may contain all application data
 - Can still be too slow - few tens of GB/s

- Accessing cache (fast memory inside processor) is much faster
 - hundreds of GB/s
 - limited in size: a few MB at most

Memory Hierarchy



Performance (overview)

Application performance often described as:

- Compute bound (limited by processor performance)
- Memory bound (limited by memory access)
- IO bound (limited by disk access)
- Communication bound (limited by interconnect)

For majority of current HPC codes:

- many calculations are limited by memory bandwidth
- processor can calculate much faster than it can access data