Introduction to Computer Architecture and Performance

Tim Mattson

… with lots of help from past presentations created by Felice Pantaleo and Severre Jarp

Introduction to me and my 40-year Career

PhD chemistry
1979-1985

PostDoc 1986
Caltech Concurrent Computation Project

Jobs with startups … numerical analysis, signal processing, scientific computing, and parallel computing
1987-1990

1990-1993
Linda and pre-MPI message passing

1993
Supercomputing Systems Division

1994
Message Passing Interface

1994
FDIV Bug

1996
ASCI Red:
World's first TFLOP

1997
Portable multithreading

2002-2005
Director of life sciences

2007
World's First TFLOP chip

2008
Portable GPU programming

2013
GraphBLAS

2015
Sparse Array storage engine

2015
BigDAWG Polystore system

2018-Aug'2023

oneAPI
Future of heterogenous computing

Machine Programming

Unified Theory of data and computing

2023-2038
Retirement

Tim Mattson
RIP

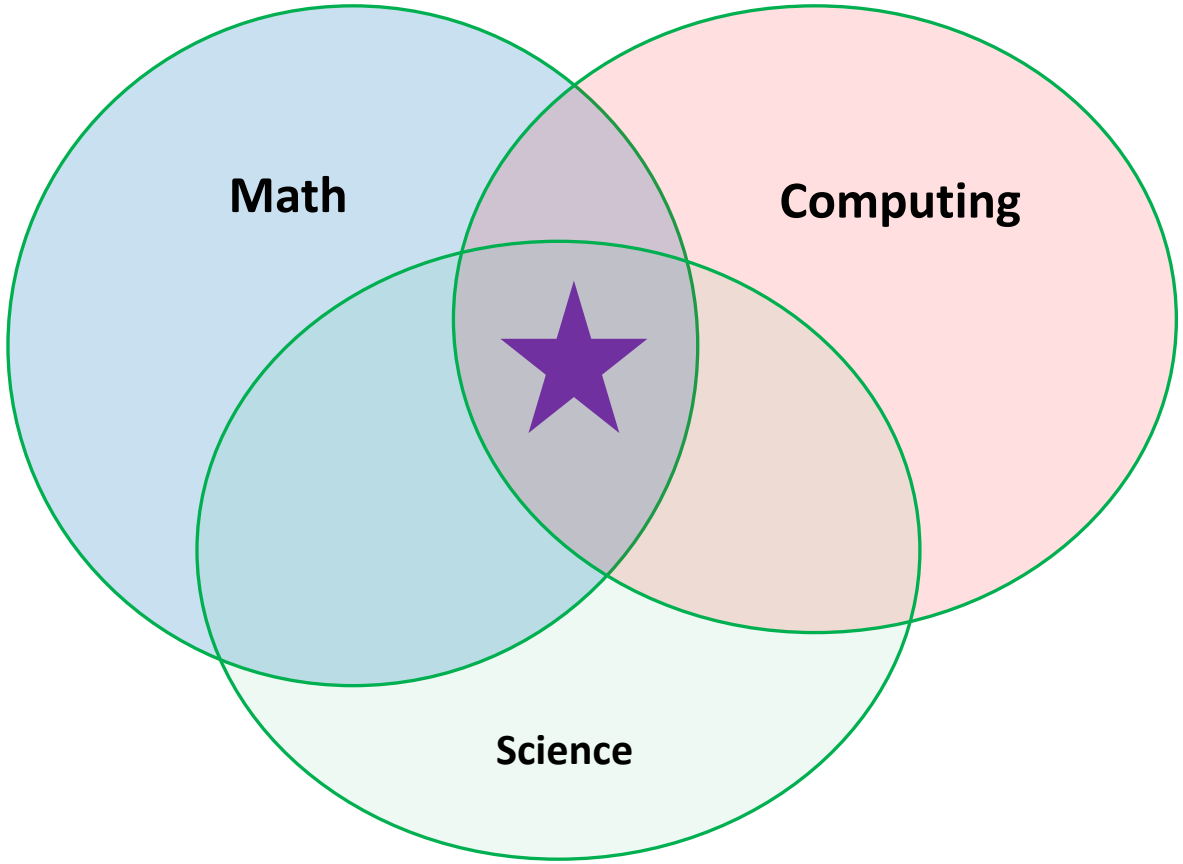
1958-2038

Here lies the Christmas bat

Write, teach, kayak, and think

Scientific Computing

Scientific Research, by design, pushes the limits of human knowledge … so applications in computational science often push the limits of what is possible computationally.



High Performance Computing … Systems (Software and Hardware) optimized for performance.

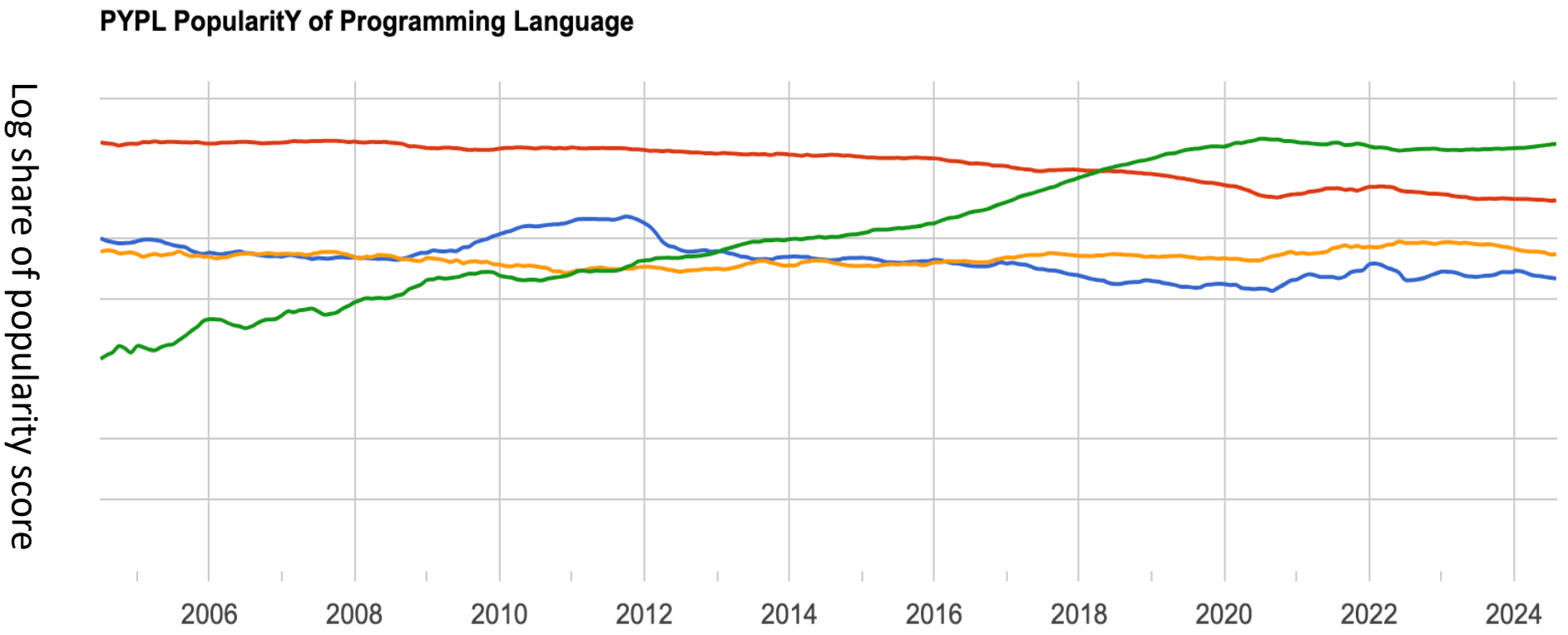
Requires reasoning about how software maps onto the features of the hardware

Much of what we do in Scientific Computing requires High Performance Computing (HPC).

★ Scientific Computing: Turn science into math. Turn math into software. Run software to get answers.

The most popular programming languages

Consider the changes in most popular programming languages…



	Share of total scores 2016	2024
C/C++	8%	6%
Java	25%	16%
JavaScript	7%	8%
Python	12%	30%

The top 3 languages, used by the overwhelming majority of programmers, hide Hardware details.

Even if workflow management and data processing are done with Python … to "do" HPC today, you need to be proficient with an HPC Language (C++, C, or Fortran)

Our focus at ESC'25

http://pypl.github.io/PYPL.html

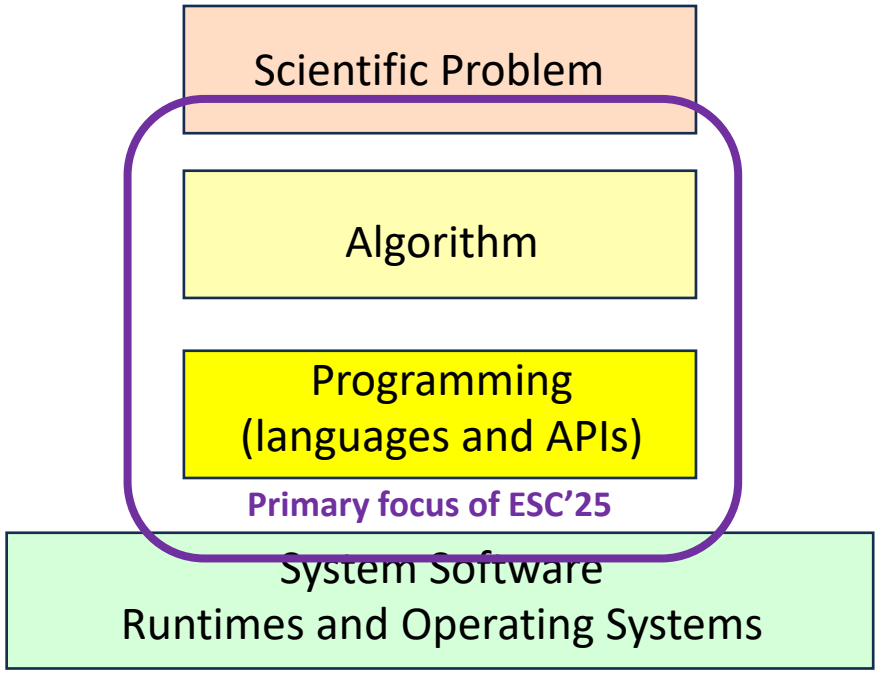
The Components of Scientific Computing

Scientific Problem	A well posed scientific problem (This is what you bring to ESC)
Algorithm	A step-by-step procedure that (1) is mathematically correct, (2) Computationally stable, and (3) Efficiently uses system resources
Programming (languages and APIs)	Languages and Application Programming Interfaces for writing HPC software
System Software Runtimes and Operating Systems	Software to manage the system and support program execution

Computer Architecture

Instruction Set Architecture (ISA)	The low-level interface to the computer presented to a programmer
Microarchitecture	The microarchitecture: A design for how the ISA is implemented
Physical (Logic, Devices, semiconductor physics)	The hardware … processors, memories, interconnects and more.

The Components of Scientific Computing

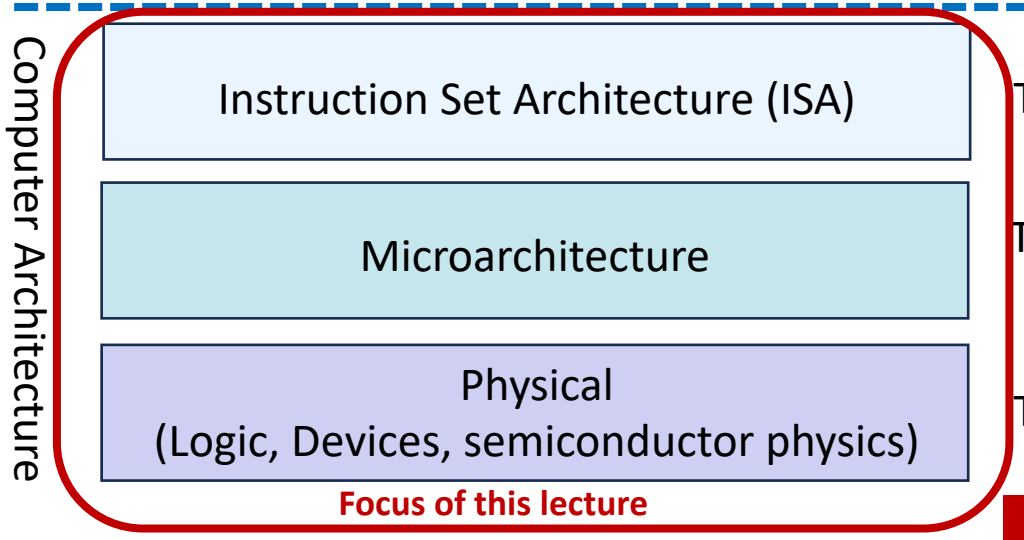


A well posed scientific problem (This is what you bring to ESC)

A step-by-step procedure that (1) mathematically correct*, (2) Computationally stable, and (3) Efficiently uses system resources

Languages and Application Programming Interfaces for writing HPC software

Software to manage the system and support program execution



The low-level interface to the computer presented to a programmer

The microarchitecture: A design for how the ISA is implemented

The hardware … processors, memories, interconnects and more.

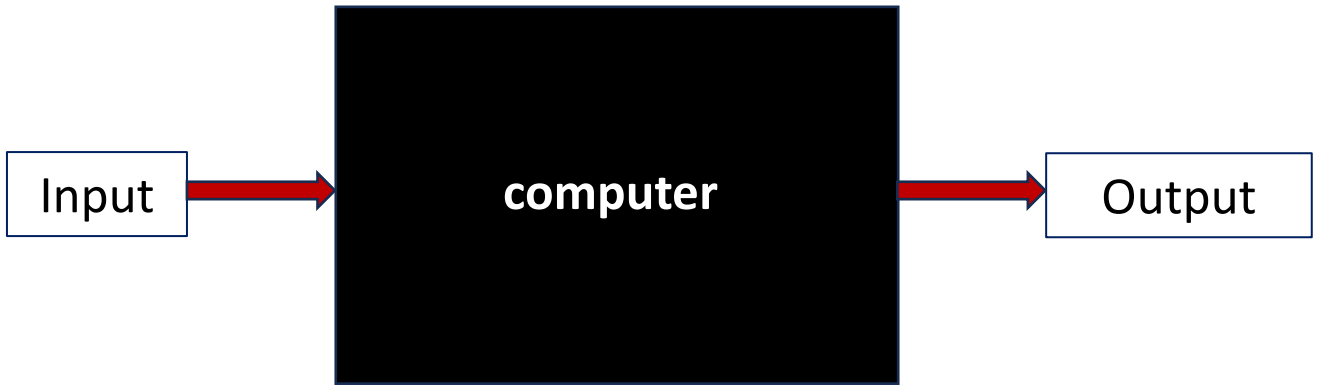
* Which includes accounting for the properties of floating point arithmetic … a topic we'll cover this afternoon

Let’s go back to basics ….

What is a computer?

What is a computer:

- **Computer:**
  - A machine that transforms *input values* into *output values*.

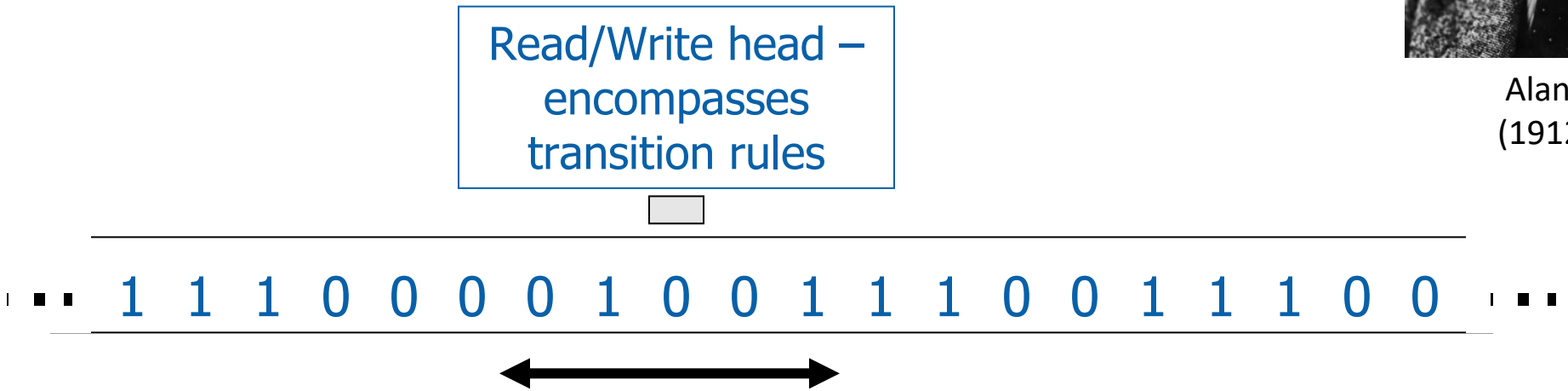


- The computer as a black-box is not very helpful. We need a bit more detail.

Computer models: Turing Machine

Alan Turing
(1912-1954)

- Alan Turing proposed a general model of a computer and showed that it was universal:



- Read an “infinite” tape of 1’s and 0’s. Based on the pattern of values, shift the tape, read values and write values. These are controlled by transition rules (i.e. a program)
- This was useful for proving mathematical theorems about computing, but not for actually working with computers.

Von Neumann or a “stored Program” Model

- John von Neumann proposed a more useful model where a computer consists of: (1) control unit, (2) Arithmetic-Logic unit (ALU), (3) registers that hold values close to the ALU, and (4) memory that holds both the data and the sequence of instructions(the program).

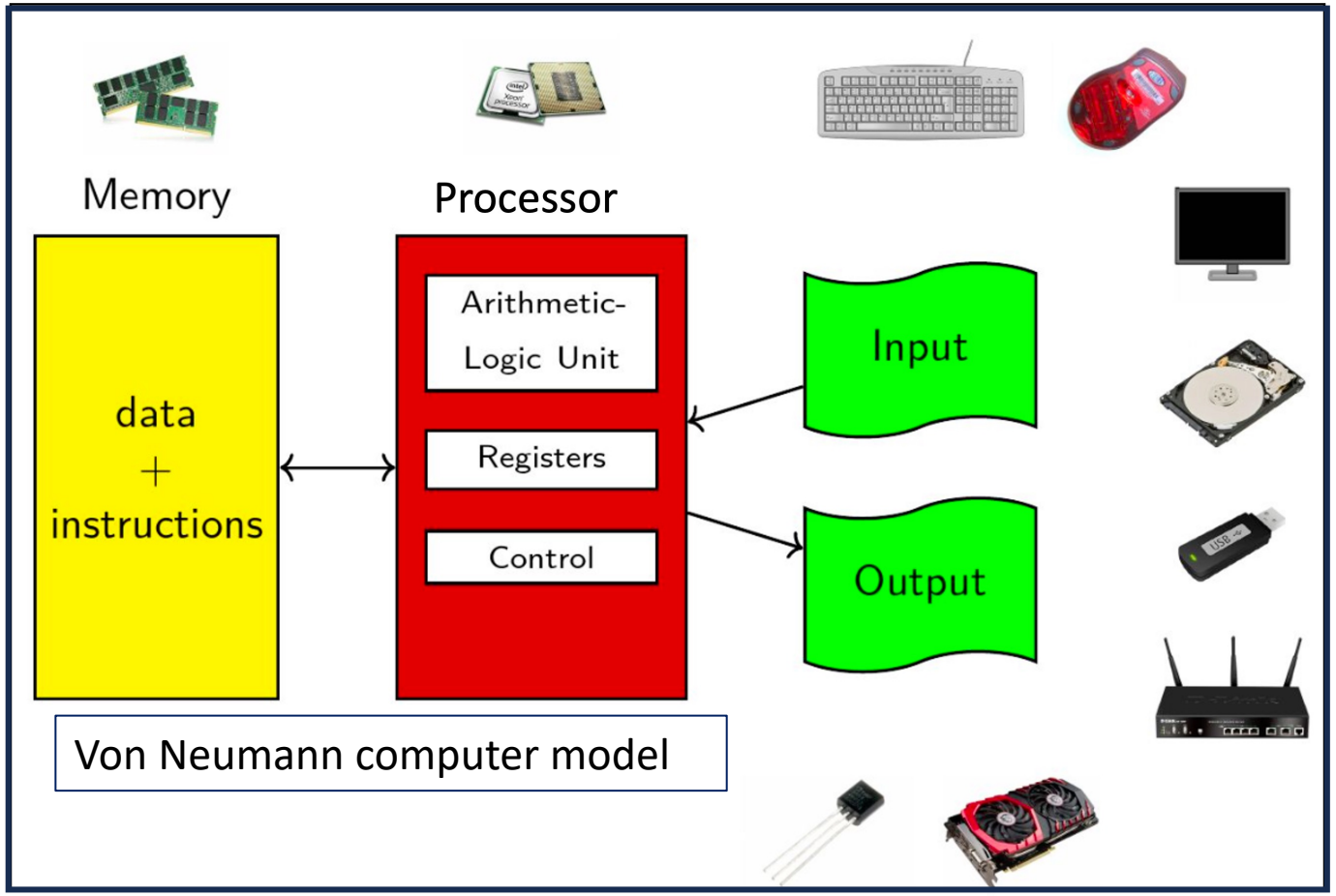
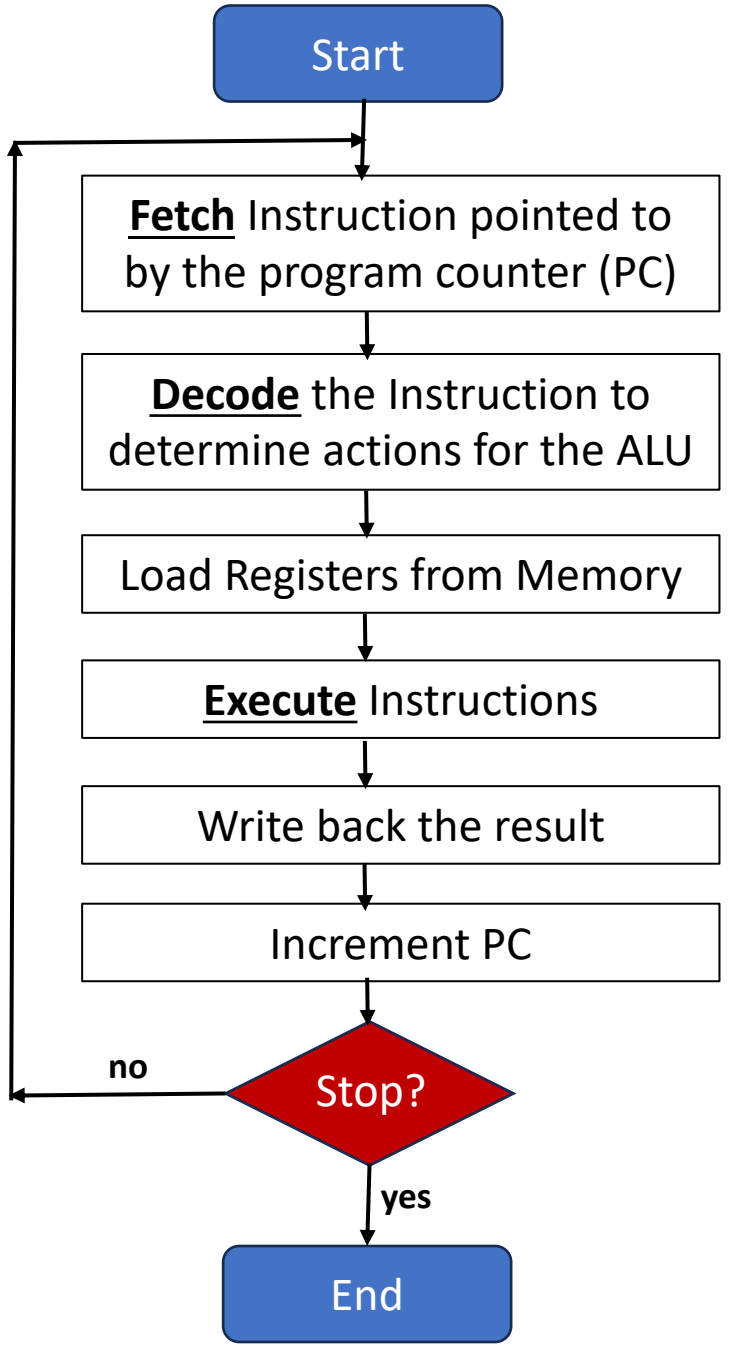


Image Source: Felice Pantaleo, CERN, ESC’23

Program Execution



Von Neumann or a "stored Program" Model

- John von Neumann proposed a more useful model where a computer consists of: (1) control unit, (2) Arithmetic-Logic unit (ALU), (3) registers that hold values close to the ALU, and (4) memory that holds both the data and the program (the sequence of instructions).

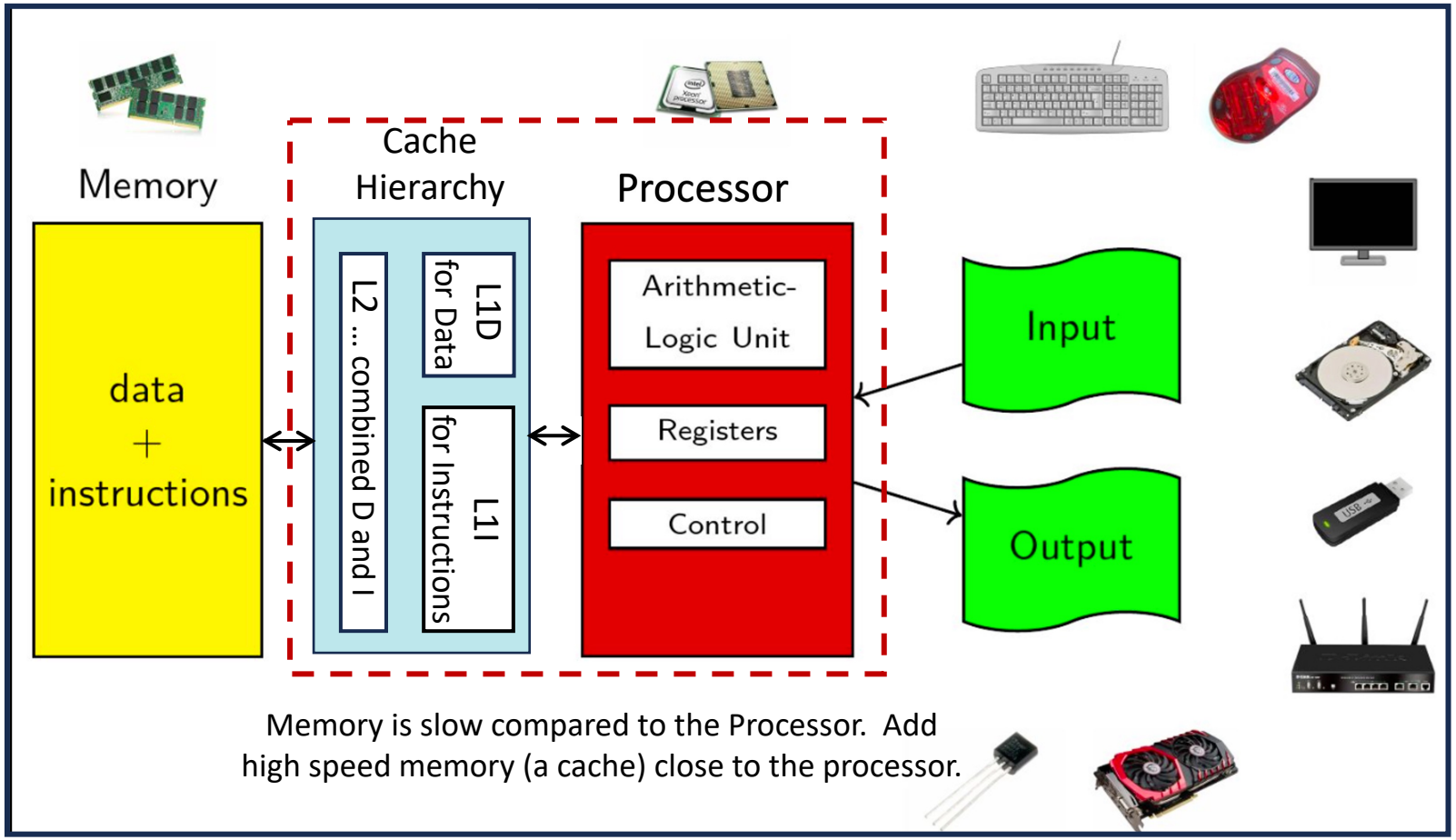
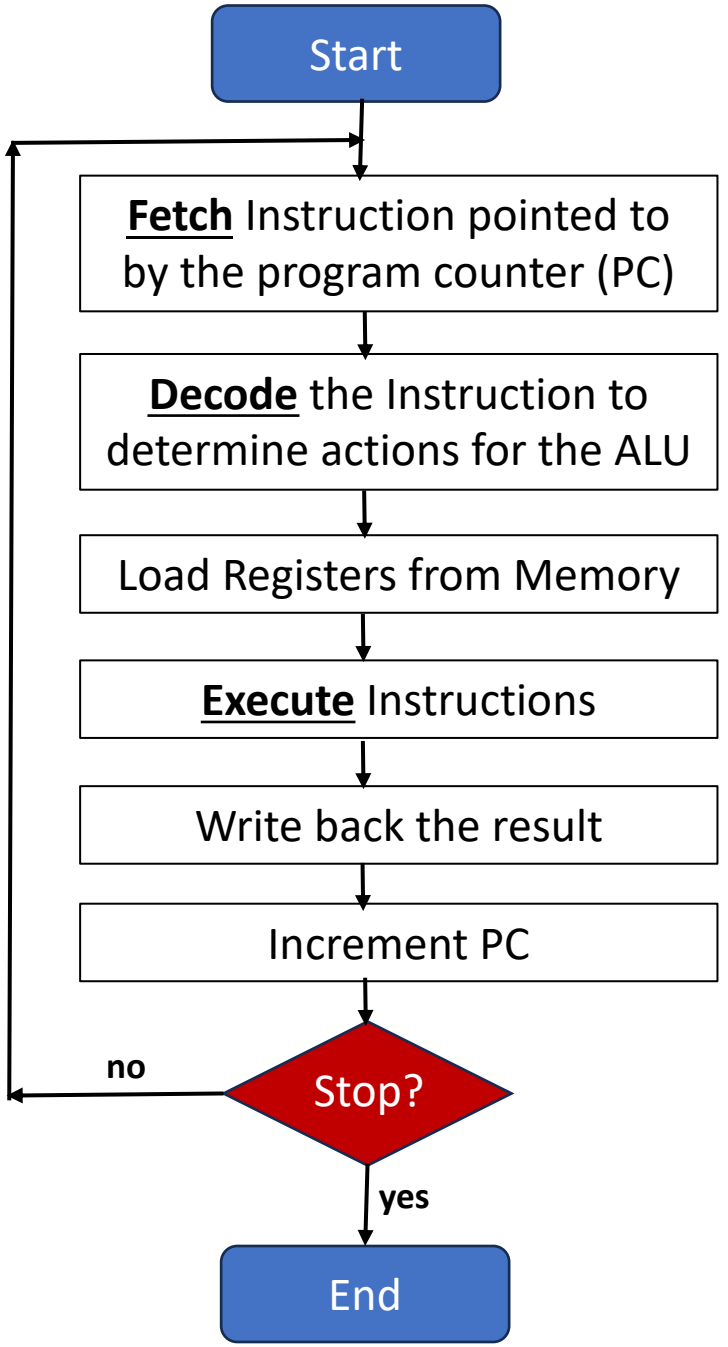


Image Source: Felice Pantaleo, CERN, ESC'23

Program Execution

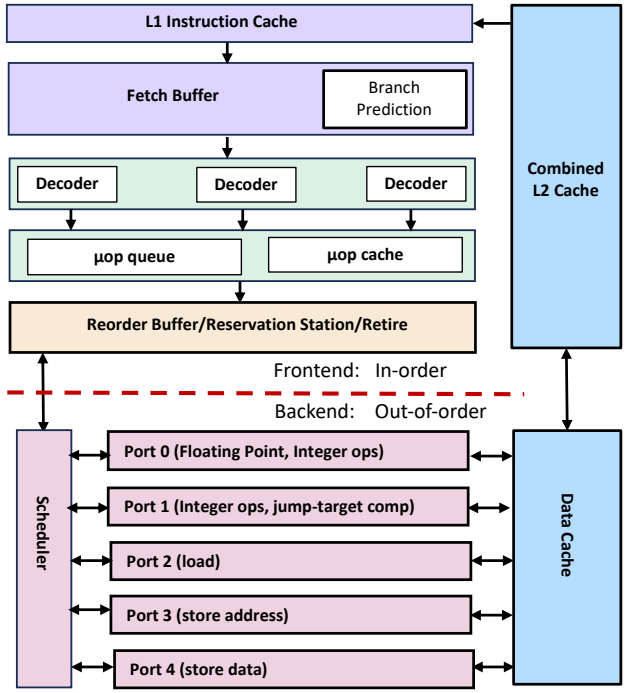


Modern computers follow the von-Neuman model

To understand computers more deeply, we need to explore the topic of computer architecture

Computer Architecture:

Conceptual design of a computer



Intel® Pentium Pro™ microarchitecture

Instruction Set Architecture (ISA)

Computer interface presented to a programmer

Microarchitecture

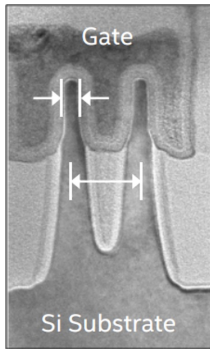
Abstract design for how the ISA is implemented

Physical

Logic, Devices, semiconductor physics

```
12  .L3:
13          fcvt.d.w        fa5,a5
14          fcvt.d.s        fa4,fa4
15          addi    a5,a5,1
16          fadd.d  fa5,fa5,ft0
17          fmul.d  fa5,fa5,fa3
18          fcvt.s.d        fa5,fa5
19          fmul.s  fa5,fa5,fa5
20          fcvt.d.s        fa5,fa5
21          fadd.d  fa5,fa5,fa1
22          fdiv.d  fa5,fa2,fa5
23          fadd.d  fa5,fa5,fa4
24          fcvt.s.d        fa4,fa5
25          bne     a0,a5,.L3
26          fmul.s  fa0,fa0,fa4
27          ret
```

RISC-V assembly code for the "pi program loop" generated by gcc -O3



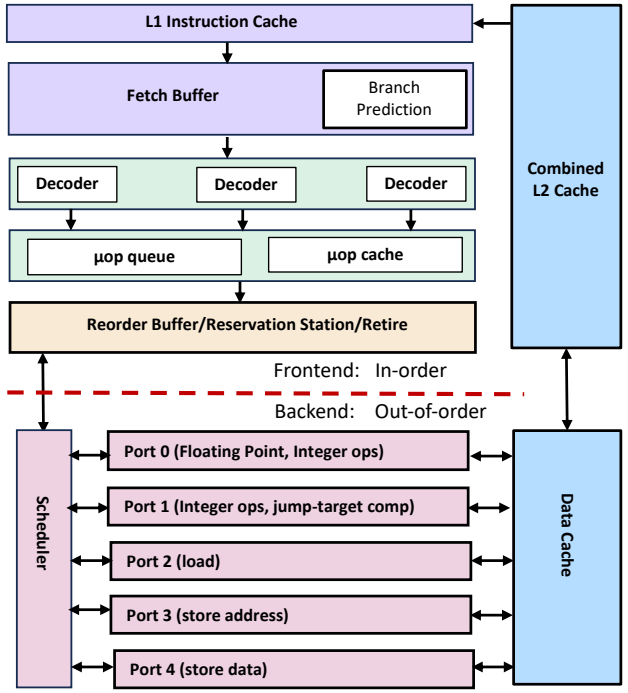
Electron microscopic image of an Intel transistor for the 14 nm process technology

Assembly code from https://godbolt.org/

Transistor: https://www-ssl.intel.com/content/dam/www/public/us/en/documents/technology-briefs/bohr-14nm-idf-2014-brief.pdf

Computer Architecture:

Conceptual design of a computer



Intel® Pentium Pro™ microarchitecture

Instruction Set Architecture (ISA)

Computer interface presented to a programmer

Microarchitecture

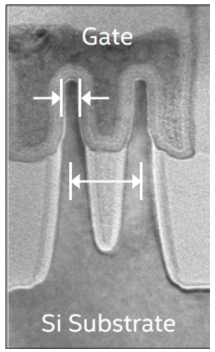
Abstract design for how the ISA is implemented

Physical

Logic, Devices, semiconductor physics

```
12  .L3:
13          fcvt.d.w        fa5,a5
14          fcvt.d.s        fa4,fa4
15          addi    a5,a5,1
16          fadd.d  fa5,fa5,ft0
17          fmul.d  fa5,fa5,fa3
18          fcvt.s.d        fa5,fa5
19          fmul.s  fa5,fa5,fa5
20          fcvt.d.s        fa5,fa5
21          fadd.d  fa5,fa5,fa1
22          fdiv.d  fa5,fa2,fa5
23          fadd.d  fa5,fa5,fa4
24          fcvt.s.d        fa4,fa5
25          bne     a0,a5,.L3
26          fmul.s  fa0,fa0,fa4
27          ret
```

RISC-V assembly code for the "pi program loop" generated by gcc -O3



Electron microscopic image of an Intel transistor for the 14 nm process technology

Assembly code from https://godbolt.org/

Transistor: https://www-ssl.intel.com/content/dam/www/public/us/en/documents/technology-briefs/bohr-14nm-idf-2014-brief.pdf

Instruction Set Architecture (ISA)

- The instruction set architecture (ISA), is the interface to the hardware presented to the programmer
- Two major classes of ISA
  - CISC: **Complex instruction set Computer**. Large set of instructions to cover numerous special cases. Example: Intel® x86 ISA
  - RISC: **Reduced instruction set Computer**. Smaller set of instructions, easier to work with and implement. Example: ARMv8

ISA features	**Intel x86* (CISC)**	**ARMv8 (RISC)**
Class of ISA	Register/memory ISA … operations can reference registers or memory.	Register ISA … Instructions work on registers only .. Exposed to memory through load-store operations.
Memory address	Bytes addressing	Byte addressing, but objects must be aligned An object of size **s** bytes is aligned if (Addr mod s = 0).
Registers exposed by architecture definition	16 general purpose and 16 floating point	31 general purpose 32 floating point registers
Encoding an ISA … Instruction widths	Variable length, ranging from 1 to 18 bytes. Can result smaller executables.	Fixed length, 4 byte Thumb instructions: 2-byte
Number of instructions	Exact count is difficult … over 3500	Base = 354, SIMD/FP = 404, SVE = 508 … total ~1266

ISA details are challenging to nail down. The Intel ISA manual is over 5000 pages.
Hence numbers on this slide convey a general sense of size and miss many details and special cases.

* these numbers are for the Intel® 64 x86 ISA. * SIMD: single instruction multiple data or vector instructions. FP: floating point SVE: Scalable vector instructions

Instruction sets: Complex (CISC) vs Reduced (RISC)

```
1  void add_abc(double *a, double *b, double *c)
2  {
3      *c = *a + *b;
4  }
```

Compare assembly code for a simple function for CISC (x86-64) and RISC (ARM) processors

COMPILER EXPLORER

https://godbolt.org/

CISC

Ops work on registers and addresses in memory.

Complex but extra options for aggressive optimization

x86-64 gcc 14.2 | -O2 | Output... | Filter... | Libraries | Overrides

```
1  add_abc(double*, double*, double*):
2          movsd   xmm0, QWORD PTR [rdi]
3          addsd   xmm0, QWORD PTR [rsi]
4          movsd   QWORD PTR [rdx], xmm0
5          ret
```

- Load double at address [rdi] into register xmm0
- Add double at address [rsi] to xmm0, put result in xmm0
- Store double in xmm0 to address [rdx]
- Branch to return address on the stack

RISC

All ops on registers

Consistency means smaller and simpler instruction set

ARM GCC 14.2.0 | -O2 | Output... | Filter... | Libraries | Overrides

```
1  add_abc(double*, double*, double*):
2          vldr.64 d16, [r0]
3          vldr.64 d17, [r1]
4          vadd.f64        d16, d16, d17
5          vstr.64 d16, [r2]
6          bx      lr
```

- Load double at address [r0] into register d16
- Load double at address [r1] into register d17
- Add double in d17 to double in d16, put result in d16
- Store double in d16 to address [r2]
- Branch to return address lr

Computer Architecture: CISC vs RISC

- Intel pioneered mass-market-computing through the IBM PC.

- Intel's x86 ISA is a CISC instruction set and that played a key role in the history of computing.
  - Starting with the Intel 8086 CPU in 1978 and continuing to today as a frequently used architecture for servers and laptops.

- ARM: the dominant commercial RISC vendor starting with the ARM1.
  - ARM1, 1985, 25 thousand transistors compared to Intel's 1985 CPU (i386) with 275 thousand transistors.

- **Every new CPU ISA since 1980 has been based on a RISC ISA**.
  - As we'll see later, internal to a modern CISC CPU from Intel is a RISC execution engine.

The "golden handcuffs" of legacy applications will keep CICS/x86 around for many years. But in terms of innovative designs and the future, **RISC has "won"**.

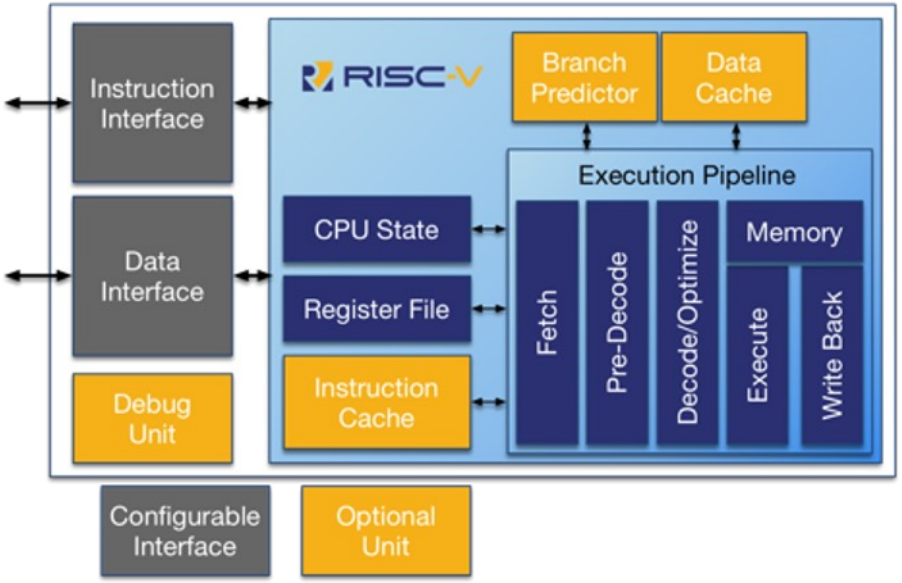
RISC across the computer industry

- ARM licenses CPU designs for others to implement.
  - Used extensively in cell-phones, tablets, Apple laptops, embedded processors, and other devices. The number one CPU by volume.
  - ARM is moving into Servers and HPC … For example, Nvidia is shipping chips for HPC using ARM (Nvidia® Hopper™)
  - ARM charges a royalty for each unit sold and vigorously protects its monopoly over the ISA.

- Just as Open-Source Software changed the nature of the software industry, an Open-Source ISA will change the hardware industry.

- RISC-V (pronounced RISK-Five) is an open-source ISA.
  - 2010: research project in the Computer Science Department at the University of California, Berkeley.
  - 2011: The first RISC-V specifications were released.
  - 2015: RISC-V International was established to promote adoption and standardization of the RISC-V ISA. Now has over 200 members.

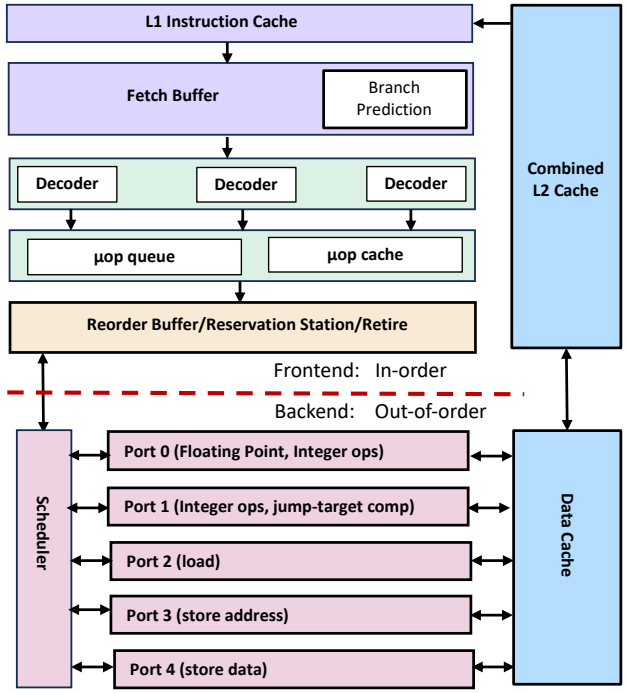
RISC-V block diagram



- Load-store ISA
- 32 bit instruction format
- RISC-V base ISA has only 50 instructions compared to 354 for ARM8 base ISA

Big companies like Apple and Google will tire of paying royalties per unit to ARM. The future is RISC-V

Computer Architecture:
Conceptual design of a computer



Intel® Pentium Pro™ microarchitecture

Instruction Set Architecture (ISA)

Computer interface presented to a programmer

Microarchitecture

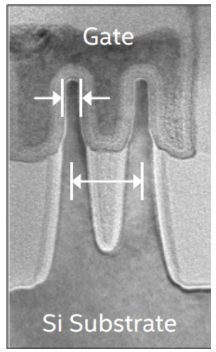
Abstract design for how the ISA is implemented

Physical

Logic, Devices, semiconductor physics

```
12  .L3:
13          fcvt.d.w        fa5,a5
14          fcvt.d.s        fa4,fa4
15          addi    a5,a5,1
16          fadd.d  fa5,fa5,ft0
17          fmul.d  fa5,fa5,fa3
18          fcvt.s.d        fa5,fa5
19          fmul.s  fa5,fa5,fa5
20          fcvt.d.s        fa5,fa5
21          fadd.d  fa5,fa5,fa1
22          fdiv.d  fa5,fa2,fa5
23          fadd.d  fa5,fa5,fa4
24          fcvt.s.d        fa4,fa5
25          bne     a0,a5,.L3
26          fmul.s  fa0,fa0,fa4
27          ret
```

RISC-V assembly code for the "pi program loop" generated by gcc -O3



Electron microscopic image of an Intel transistor for the 14 nm process technology

Assembly code from https://godbolt.org/

Transistor: https://www-ssl.intel.com/content/dam/www/public/us/en/documents/technology-briefs/bohr-14nm-idf-2014-brief.pdf

A modern CPU

Intel® X86 architecture

Sandy Bridge Microarchitecture

By the time we are done, **most** of this will make sense to you.

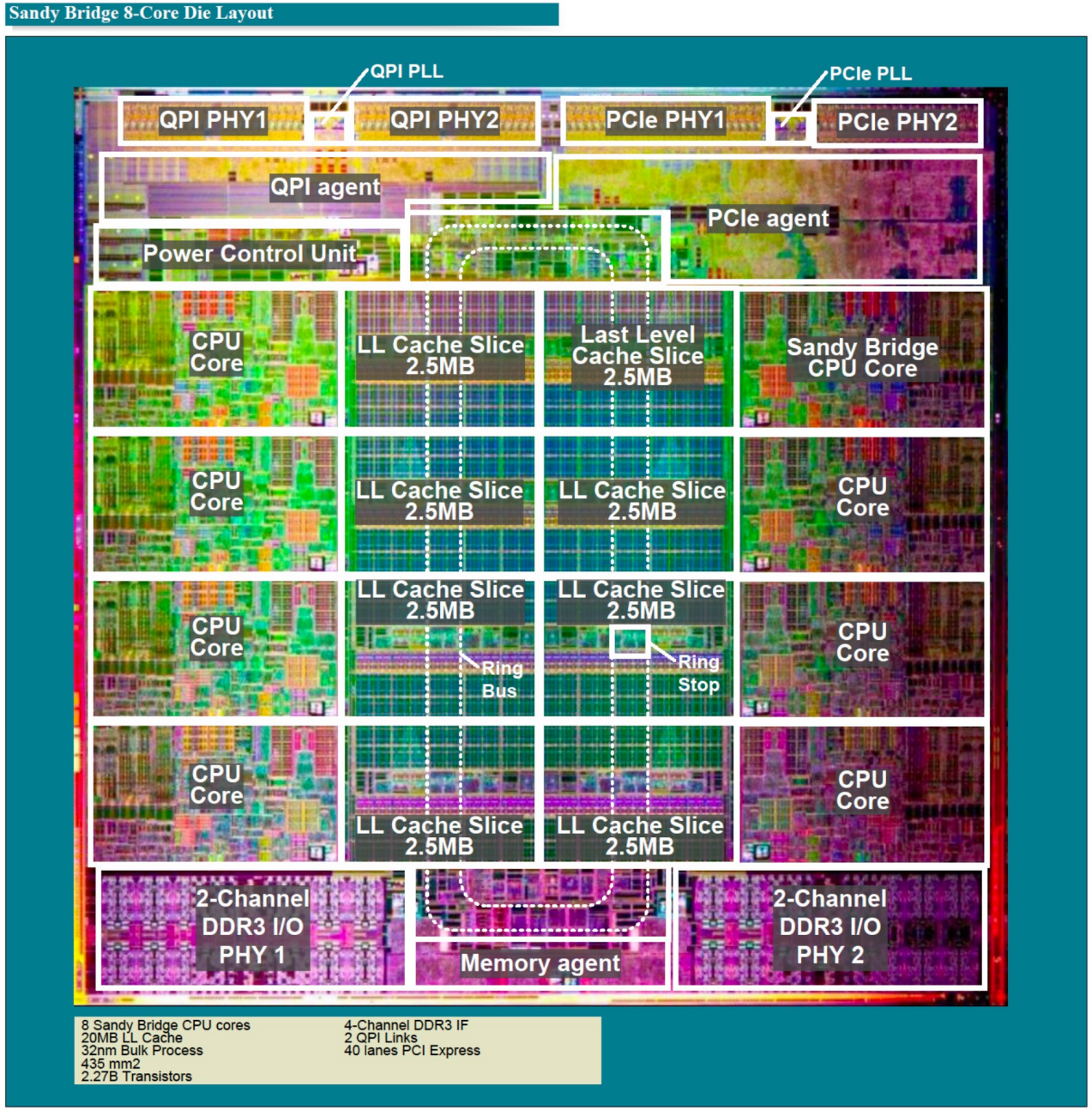
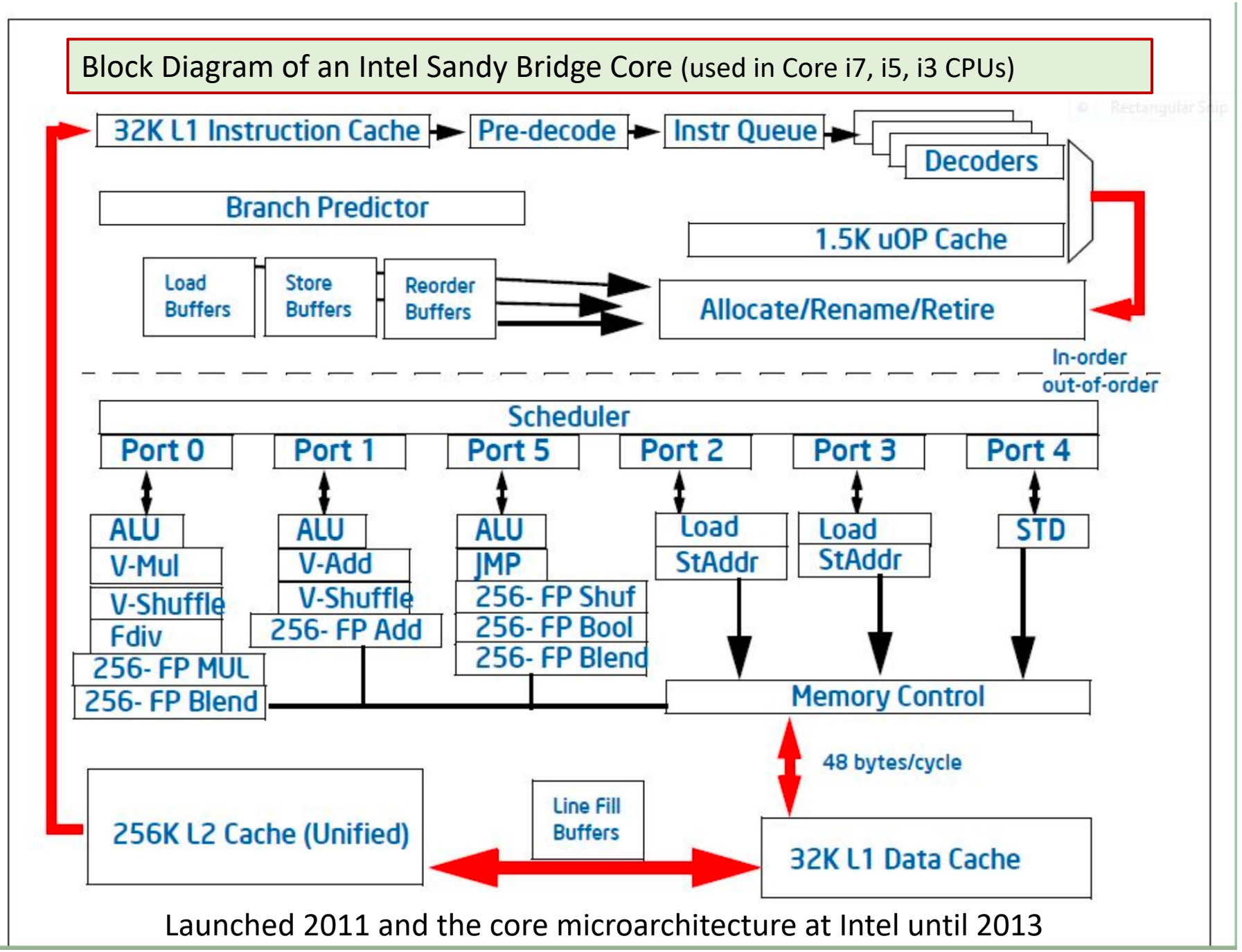


Image Source: Victor Eijkhout, processor architecture lecture, 2023

Structure of a Sandy Bridge CPU core

- The point of a microarchitecture is to support the architecture with aggressive optimization to achieve high performance.
- A modern microarchitecture can be extremely complex … both for CISC and RISC chips

By the time we are done, **much** of this will make sense to you.



Launched 2011 and the core microarchitecture at Intel until 2013

The key to performance inside a CPU:
Instruction level parallelism (ILP)

- The fundamental equation of quantitative architecture analysis:

$$\boldsymbol{Time_{CPU} = N_{instructions} * \frac{cycles}{Instruction} * \frac{seconds}{cycle}}$$

$N_{instructions}$≡ Number of instructions in an executable

$$\frac{\boldsymbol{cycles}}{\boldsymbol{Instruction}} = \frac{Total\ number\ cycles\ to\ execute\ a\ program}{N_{instructions}} = \textbf{CPI}$$

- An architecture that lets multiple instructions make forward progress each cycle reduces the Cycles per Instruction (CPI) … if all goes well, we can design architectures where CPI < 1.

- We do this with **Instruction Level Parallelism (ILP)**

Let's go back to the late 80's and 90's to look at Instruction level parallelism through the lens of x86 CPUs

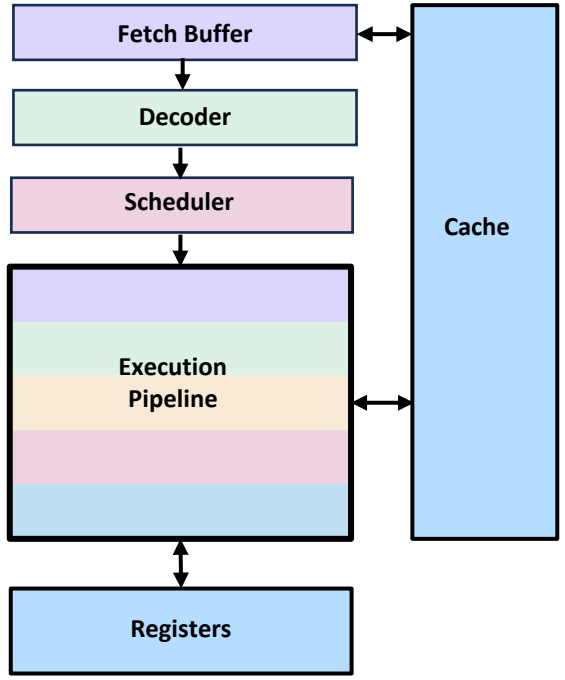
X86 … "an architecture that is difficult to explain and impossible to love"

Hennessy and Patterson, 2nd ed, page D-2

While we focus on x86 chips, all the techniques we'll discuss are found in modern in RISC chips as well

Pipelining

i486™ 1.2 Million transistors, 50 Mhz



… plus an integrated floating point unit

- Intel added pipelined instructions to i486 in 1989
- More than doubled the performance compared to a i386 at the same clock rate.
- The five stage i4586 pipeline, one cycle per stage

F	Fetch an instruction from the instruction cache.
D1	Decode the instruction.
D2	Translate memory addresses and displacements for the instruction
EX	Execute the instruction.
WB	Retire the instruction, write results back to registers and/or memory.

Five instructions going through the pipeline

Cycles (each stage takes one cycle)

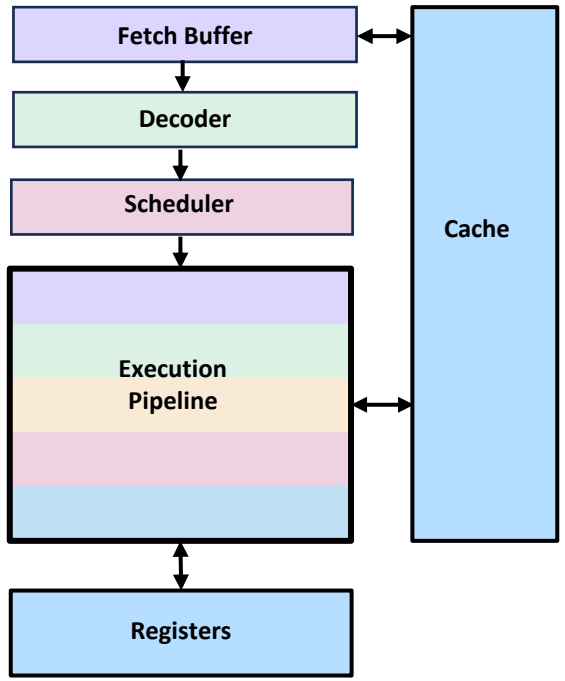
```
    1    2    3    4    5    6    7    8    9
1:  [F ][D1][D2][EX][WB]
2:       [F ][D1][D2][EX][WB]
3:            [F ][D1][D2][EX][WB]
4:                 [F ][D1][D2][EX][WB]
5:                      [F ][D1][D2][EX][WB]
```

Note: after five cycles, the pipeline is full and we get a result per cycle.

Pipelining Performance

1989

i486™ 1.2 Million transistors, 50 Mhz



… plus an integrated floating point unit

- Given the following definitions:

$$n = number\ of\ operations$$
$$\ell = number\ of\ stages$$
$$\tau = time\ per\ cycle$$

- Runtime without pipelining: $t(n) = n\ell\tau$

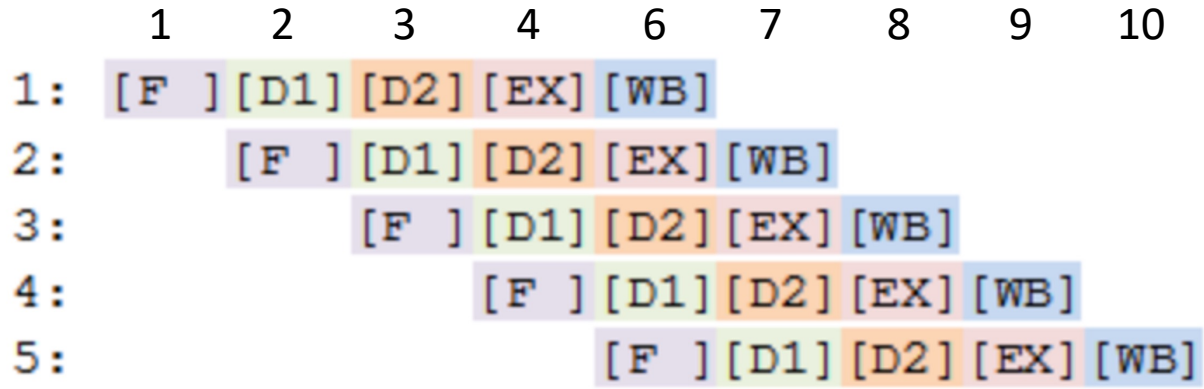
- With pipelining you need to setup the pipeline (costs s cycles) and fill the pipeline (costs ℓ cycles) at which point you have completed one instruction. Then for the next (n - 1) cycles you get one result per cycle. The runtime with pipelining is:

$$t(n) = (s + \ell + n\text{-}1)\,\tau$$

- For n much greater than $(s + \ell - 1)$, $t(n) \approx n\,\tau$, so the code runs ℓ times faster.

Five instructions going through the pipeline

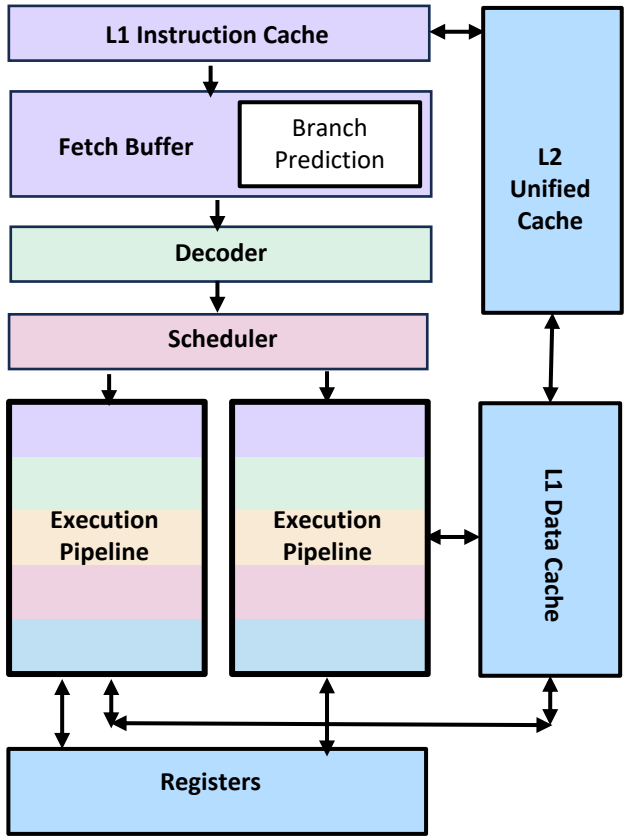
Cycles (assuming each stage takes one cycle)



Note: after five cycles, the pipeline is full and we get a result per cycle.

Superscalar + branch prediction

Pentium™ 3.1 Million transistors, 66 Mhz, 5 stage pipeline



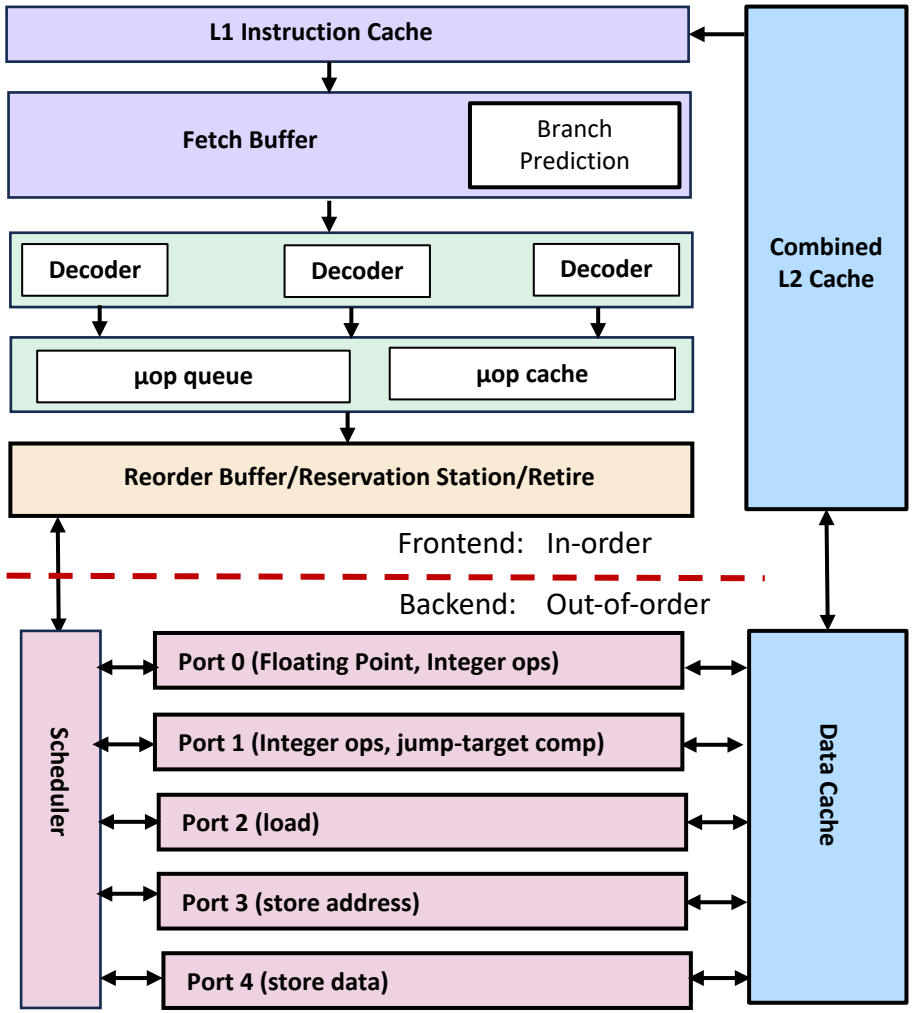
… plus an integrated floating point unit shared between pipelines

- The Pentium CPU was introduced in 1993 with major innovations
  - **Superscalar execution**: Added a second execution pipeline so it could keep two pipelined instruction streams in flight at the same time.
  - **Branch prediction**: Speculate on branches a program might take to load next instructions and get a head start should the branch be taken.
  - **Separate L1 caches for data and instructions**: A unified second level cache that holds data and instructions but separate L1 caches for data and instructions to reduce conflicts.

- Branch prediction is important. With two execution pipelines, its challenging to keep enough work in the execution units so they are fully occupied.

Out of Order (OOO) + speculation

Pentium Pro CPU, 5.5 Million transistors, 200 MHz,
14 stage pipeline

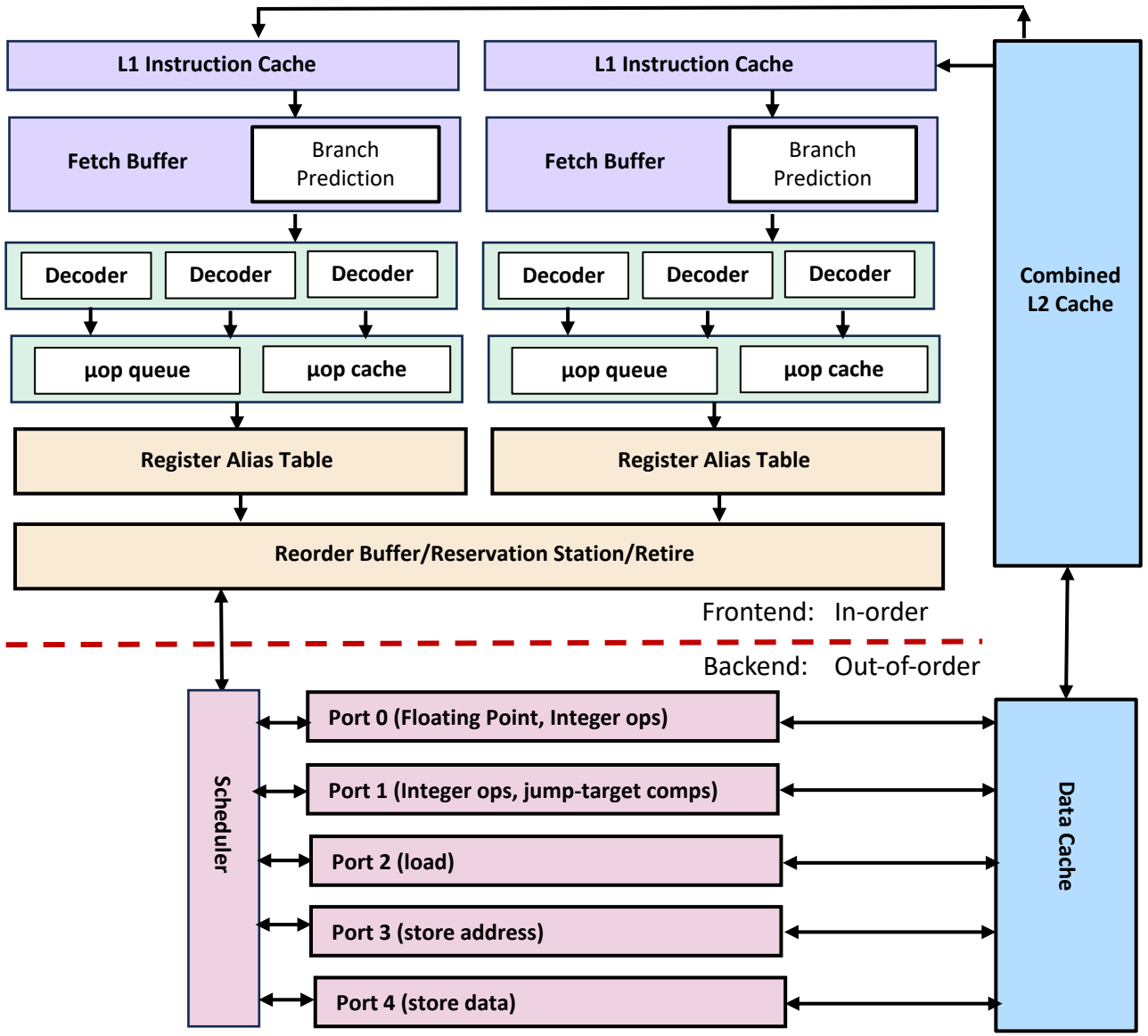


- The Pentium Pro CPU was a major performance upgrade and made Intel CPUs suitable for demanding technical computing workloads (and was used inside the computer ranked as the fastest computer in the world from 1996 to 2000).
- It added the following microarchitectural innovations:
  - Input CISC instructions were decoded into fixed-length, load/store micro-ops (µop). This made the internal execution engine inside the Intel CPU a RISC chip.
  - Micro-ops were reordered and executed based on availability of data … hence compared to input CISC instructions, they execute Out of Order (OOO)
  - Micro-ops complete in-order … i.e., they are retired in an order consistent with the input program
  - Register usage efficiency greatly enhanced by renaming them to avoid spurious conflicts due to register naming in code.
  - Dynamic speculative execution to generate enough work to fully occupy the execution units … a powerful capability but branch misprediction is very expensive as all involved pipelines must be flushed.

Simultaneous Multithreading … or the Intel Marketing term, hyperthreading

2002

Pentium 4 CPU, 125 Million transistors, 3.06 GHz, 20 stage pipeline



- Out of order execution of pipelined execution units creates so much opportunity for instruction level parallelism that it can be challenging to keep the resources fully occupied.
- Solution … replicate the in-order front end of the processor so two front-ends feed a single out-of-order backend.
- These two in-order front ends are managed as distinct threads by the OS typically with single cycle context switching overhead between them. We call these hardware threads.
- They are usually exposed to the operating system as an additional core … so the case hyperthreading case on this slide results in the OS treating the system having two cores.

Be careful with hyperthreading. If your work load can saturate the functional units on a CPU with a single thread, hyperthreading adds overhead and can slow down your code.

HPC programmers working highly optimized, compute bound codes often turn it off by default.

ILP is great, but we can get carried away

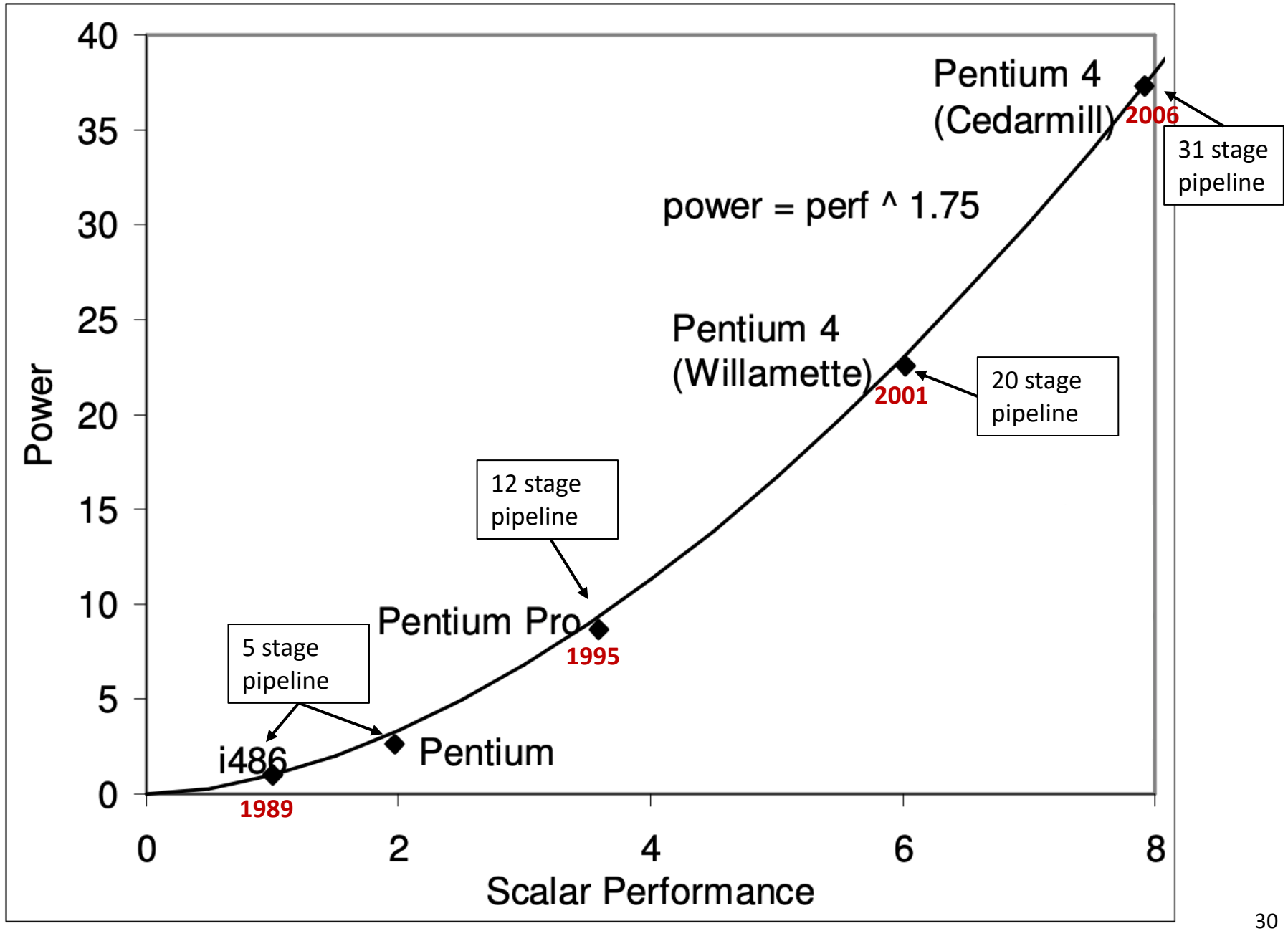
Normalized Power vs. scalar performance for Intel CPUs

Assume multiple generations of Intel CPUs using the same process technology as for i486.

Any changes are due to microarchitectural enhancements

This shows the unsustainable power demands of every deepening pipelines.

Power and Performance scaled to the i486 … e.g., Petium 4 is 6 times faster than an i486 but uses 22 times more power.



Energy per instruction Trends in Intel® Microprocessors, Ed Grochowski and Murali Annavaram, 2006

Normalized Power vs. scalar performance for Intel CPUs

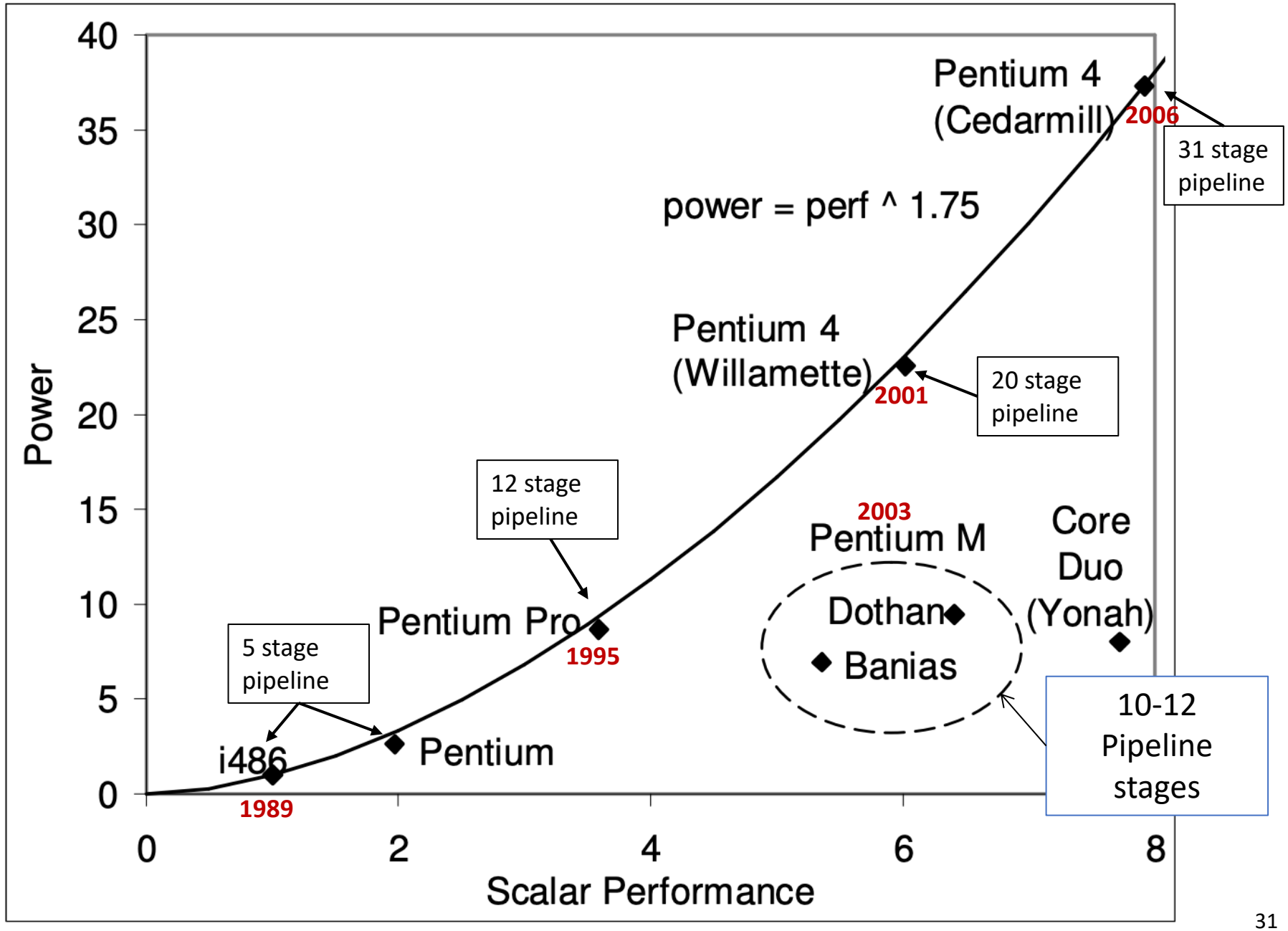
Assume multiple generations of Intel CPUs using the same process technology as for i486.

Any changes are due to microarchitectural enhancements

This shows the unsustainable power demands of every deepening pipelines.

This plot ended around 2006. More modern CPUs have pipeline depths of 14 to 20

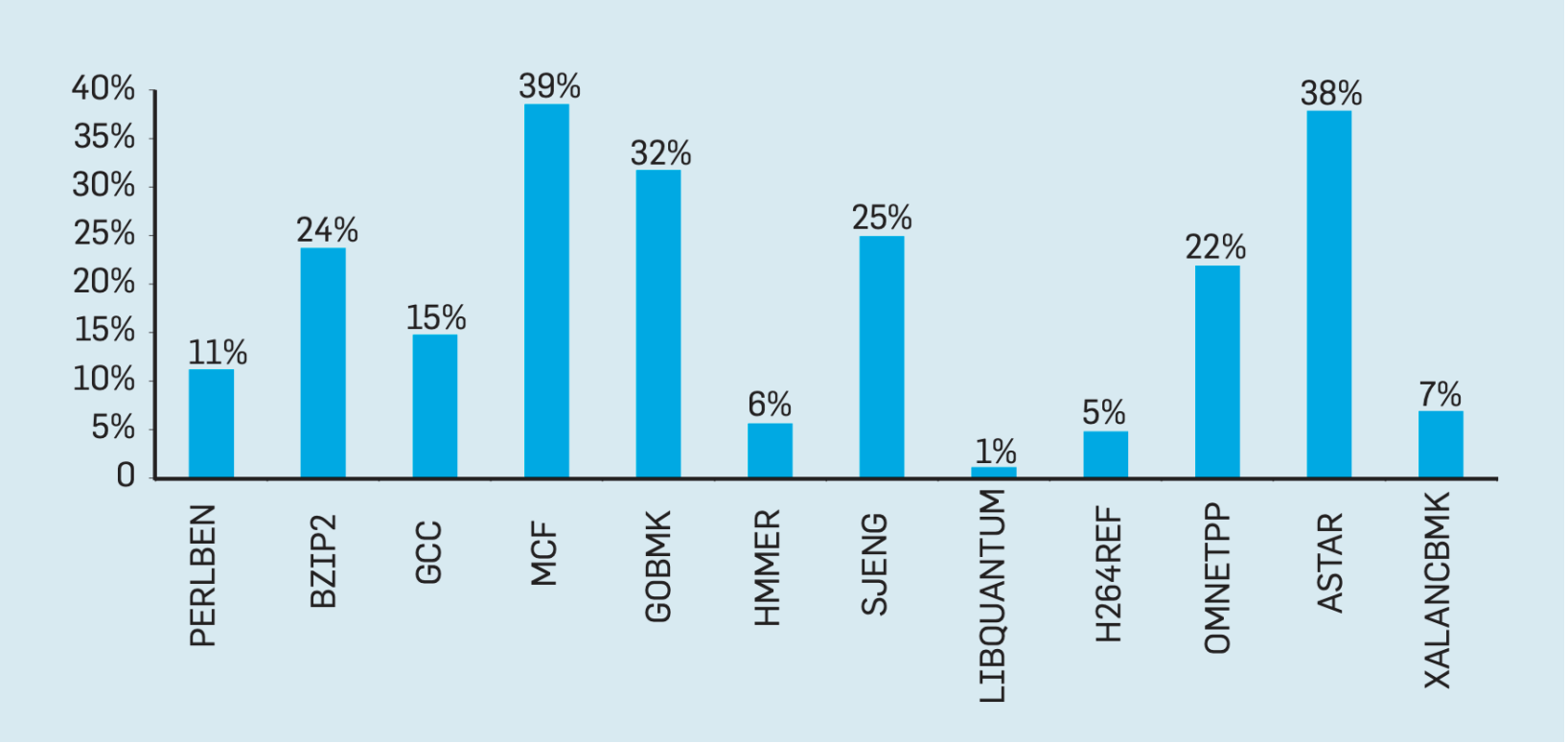
For example the Raptor Cove Intel® Xeon™ microarchitecture (2023) has 12 pipeline. stages



Energy per instruction Trends in Intel® Microprocessors, Ed Grochowski Murali Annavaram, 2006

Branch prediction

- Percentage of instructions wasted for SPEC integer benchmarks on an Intel core i7 due to incorrect branch prediction.



If you cancel a mispredicted branch, you have to flush the associated pipelines

That's really bad if you have deep pipelines

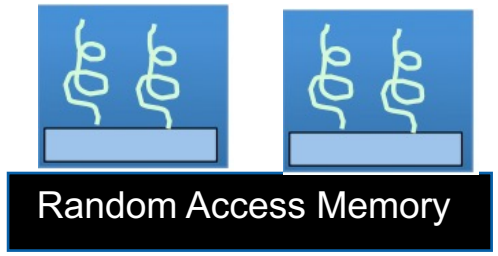
- On average, 19% of the instructions are wasted for these benchmarks on an Intel Core i7. The amount of wasted energy is greater, however, since the processor must use additional energy to restore the state when it speculates incorrectly.

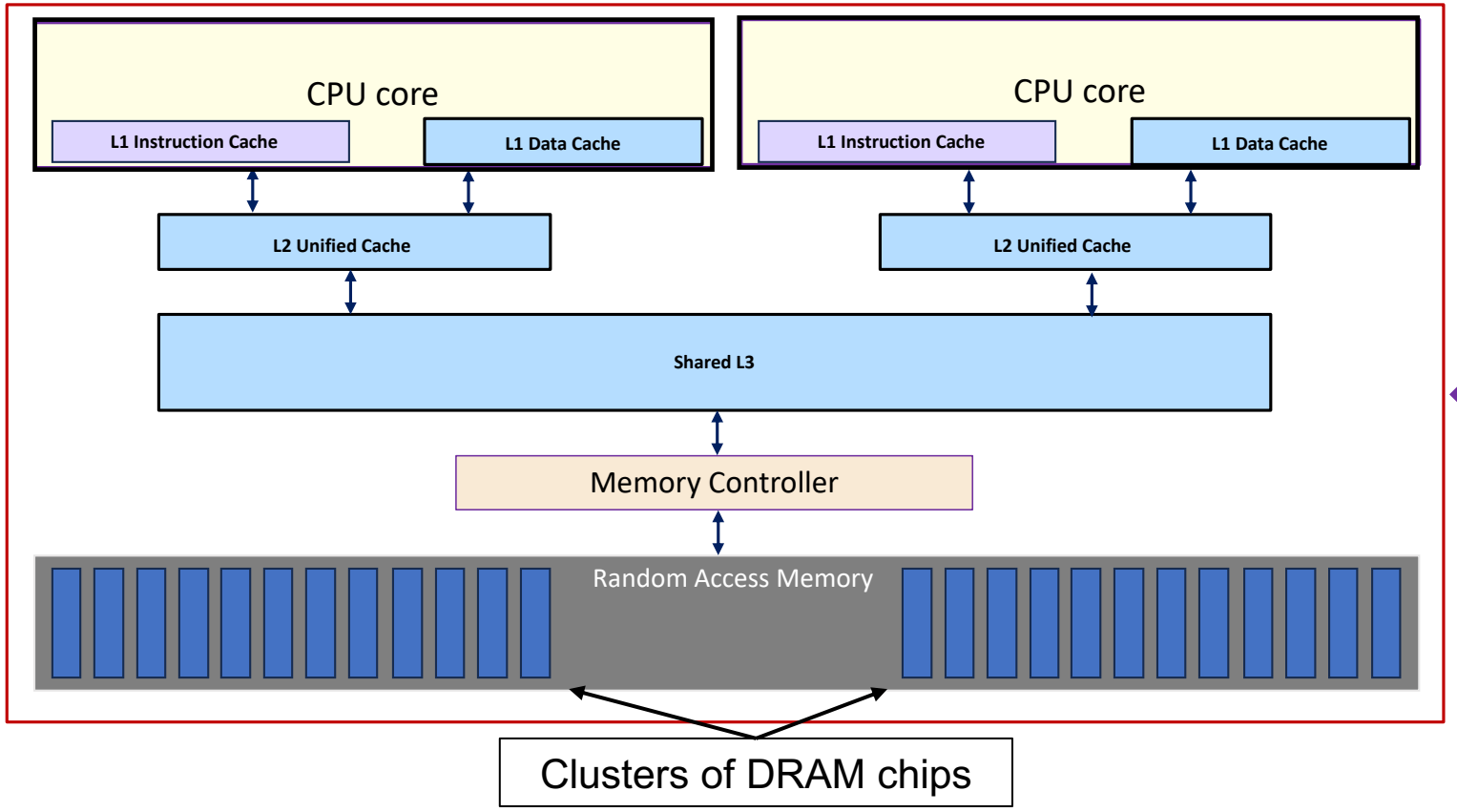
Image from "A New Golden Age for Computer Architecture", John Hennessy and David Patterson, CACM, 2019

Enough about the processors, what about the memory hierarchy

the Memory Hierarchy

We like to draw pictures like this



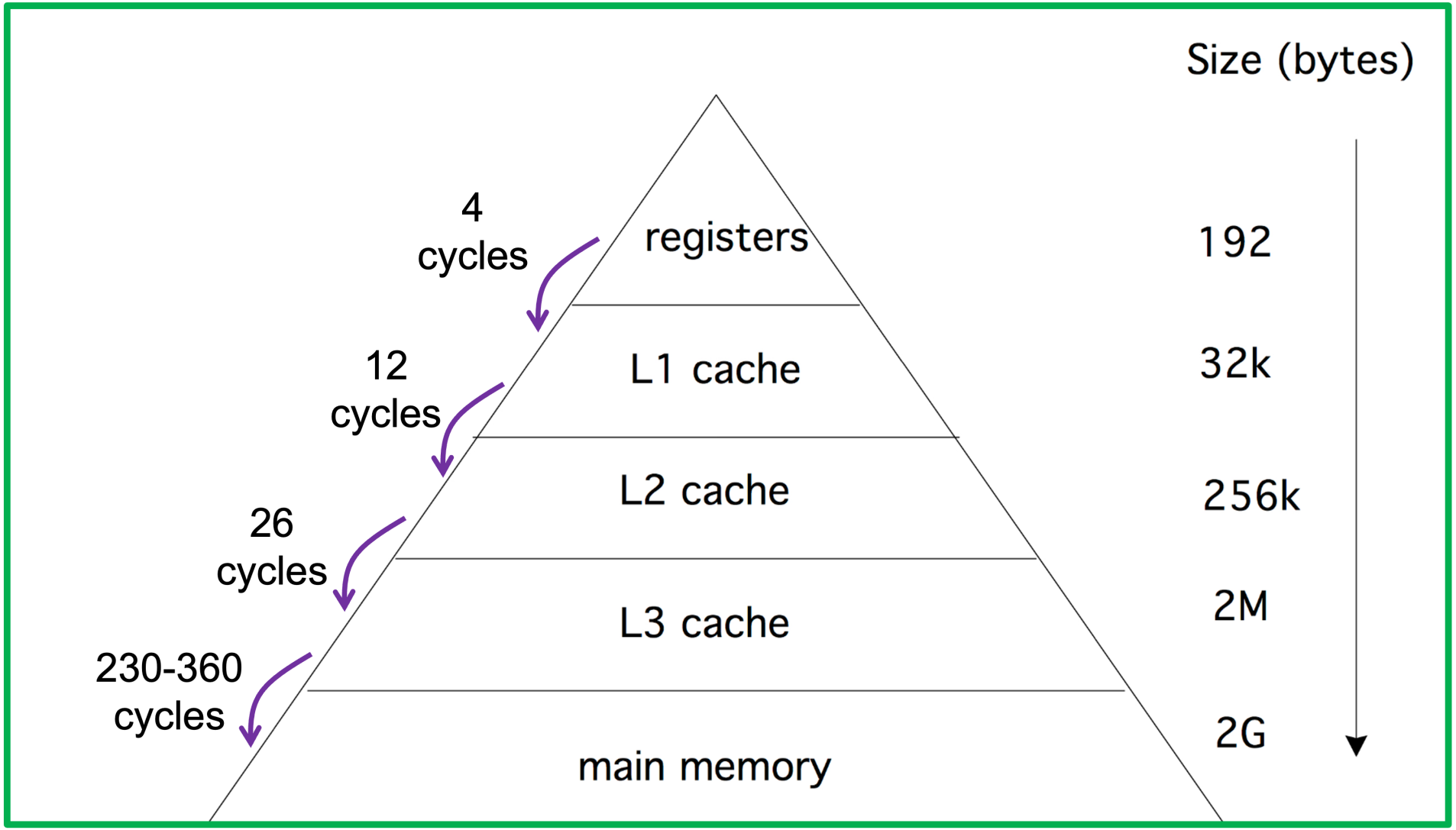


But reality is much more complex

DRAM: Dynamic Random Access Memory

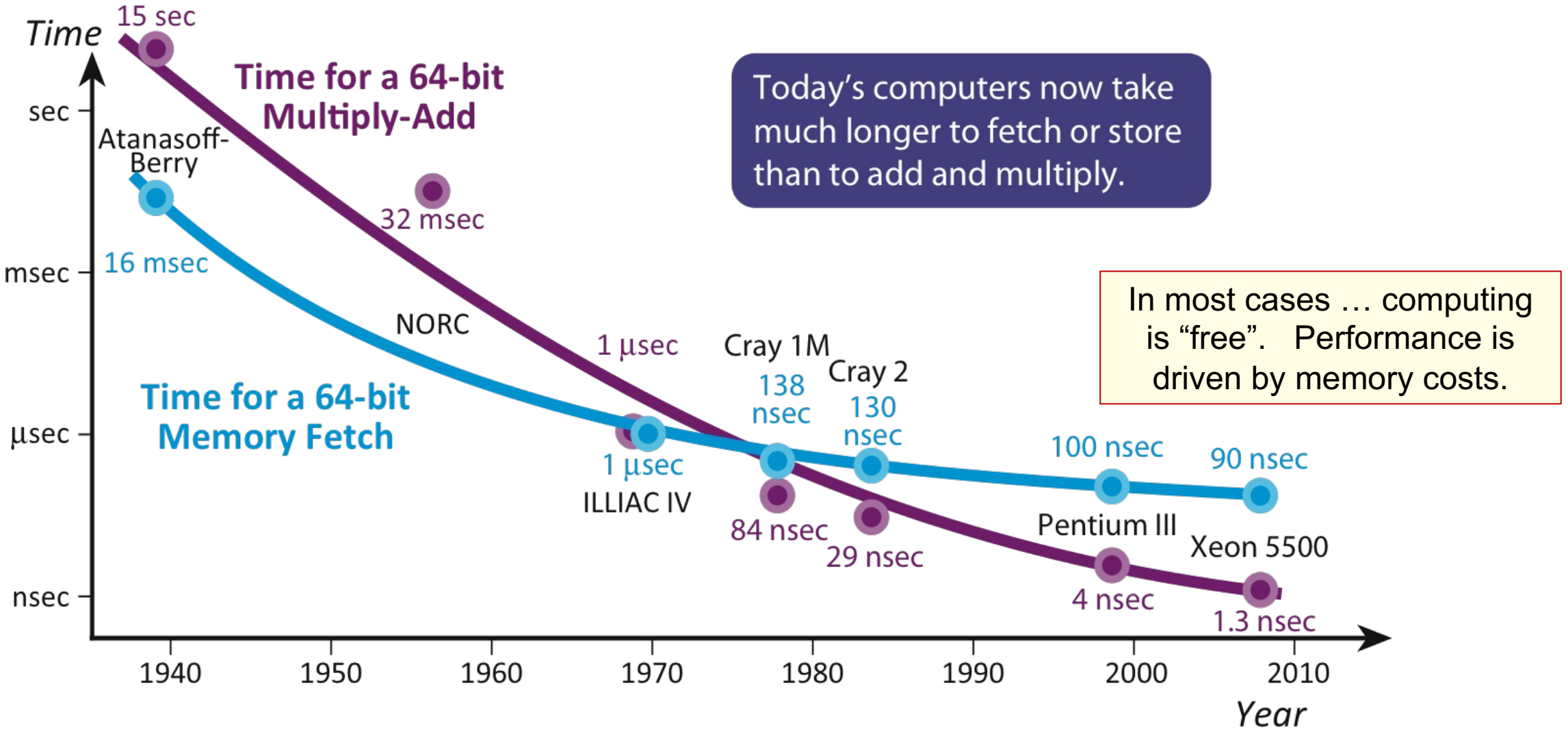
L1, L2, L3 pronounced Level 1, Level 2, Level 3. Also written sometimes as $L1, $L2, $L3

Latencies across the Memory Hierarchy



Data accessed by cache lines. Your algorithms need to organize data access patterns around the caches.

Latencies across the Memory Hierarchy



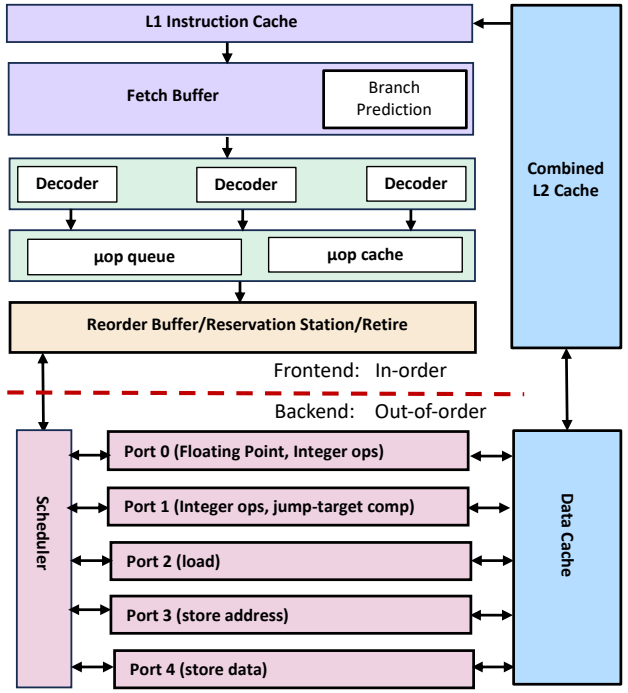
Today's computers now take much longer to fetch or store than to add and multiply.

In most cases … computing is "free". Performance is driven by memory costs.

We will have MUCH more to say about memory later this week with lectures on "Efficient Memory Management"

Computer Architecture:

Conceptual design of a computer



Intel® Pentium Pro™ microarchitecture

Instruction Set Architecture (ISA)

Computer interface presented to a programmer

Microarchitecture

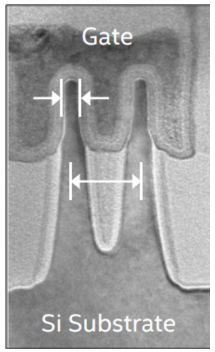
Abstract design for how the ISA is implemented

Physical

Logic, Devices, semiconductor physics

```
12  .L3:
13          fcvt.d.w        fa5,a5
14          fcvt.d.s        fa4,fa4
15          addi    a5,a5,1
16          fadd.d  fa5,fa5,ft0
17          fmul.d  fa5,fa5,fa3
18          fcvt.s.d        fa5,fa5
19          fmul.s  fa5,fa5,fa5
20          fcvt.d.s        fa5,fa5
21          fadd.d  fa5,fa5,fa1
22          fdiv.d  fa5,fa2,fa5
23          fadd.d  fa5,fa5,fa4
24          fcvt.s.d        fa4,fa5
25          bne     a0,a5,.L3
26          fmul.s  fa0,fa0,fa4
27          ret
```

RISC-V assembly code for the "pi program loop" generated by gcc -O3

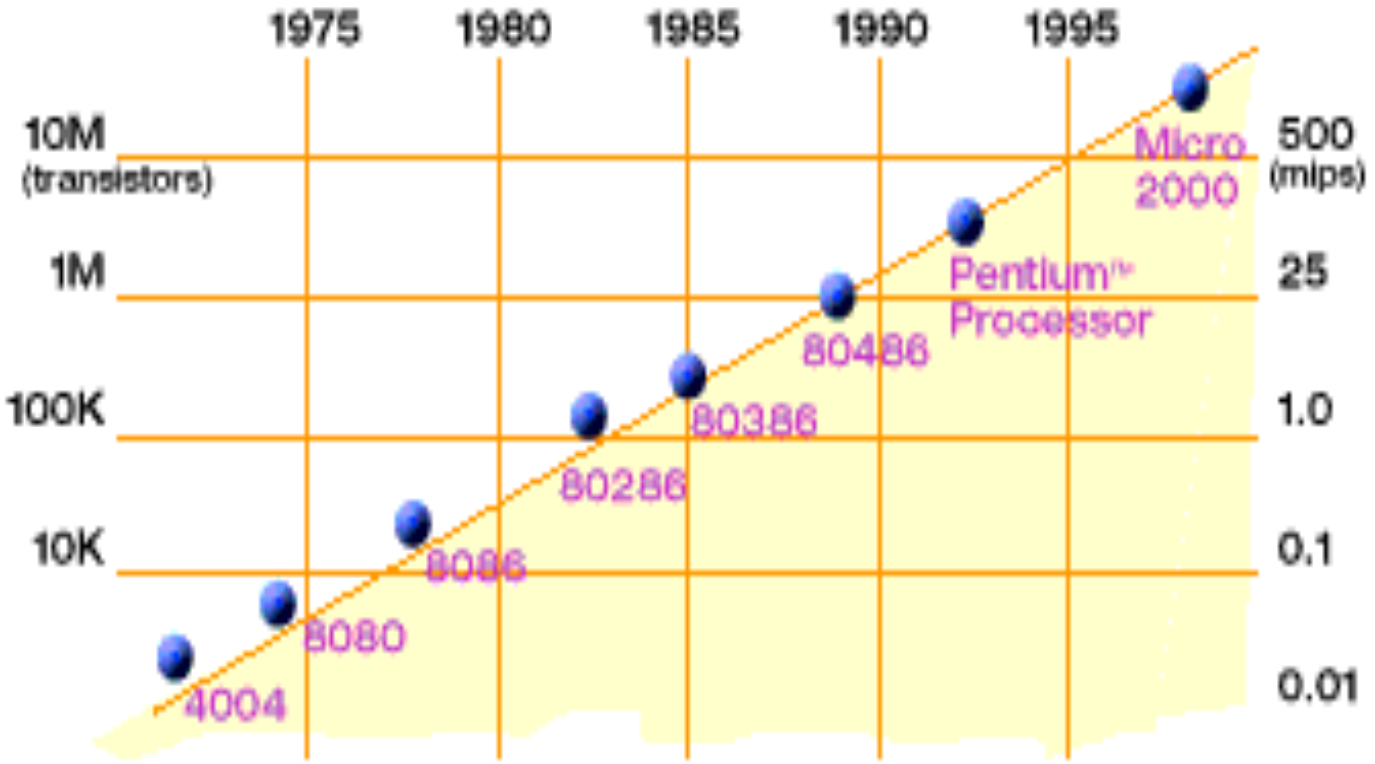


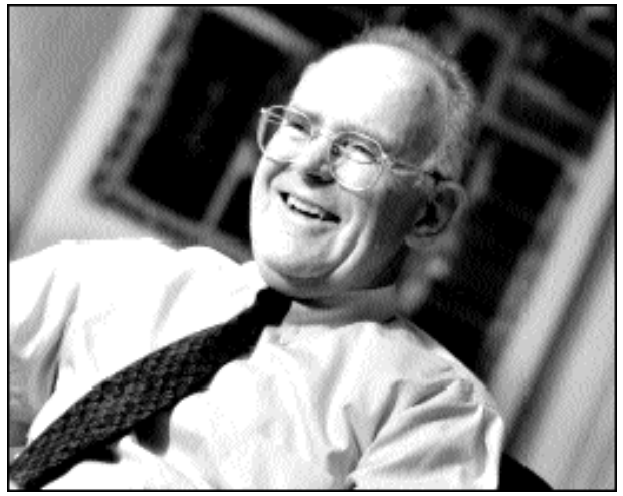
Electron microscopic image of an Intel transistor for the 14 nm process technology

Assembly code from https://godbolt.org/

Transistor: https://www-ssl.intel.com/content/dam/www/public/us/en/documents/technology-briefs/bohr-14nm-idf-2014-brief.pdf

Moore's Law





- In 1965, Intel co-founder Gordon Moore predicted (from just 3 data points!) that semiconductor density would double every 18 months.
  - ***He was right!*** Over the last 50 years, transistor densities have increased as he predicted.

"Cramming more components onto integrated circuits", G.E. Moore, Electronics, 38(8), April 1965

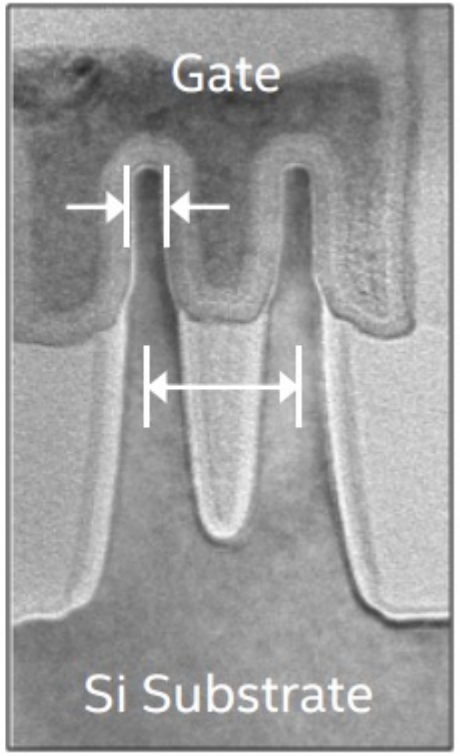
Slide source: UCB CS 194 Fall'2010

We've come a long way since Gordon Moore proposed his famous law

An electron microscope image of a single Intel transistor using the 14 nm process

8 nm Fin Width

42 nm Fin Pitch



We put billions of these transistors on a single chip

How small is a nm (nanometer)? One nm is 10^{-9} meters. Light travels about one foot in 10^{-9} seconds.

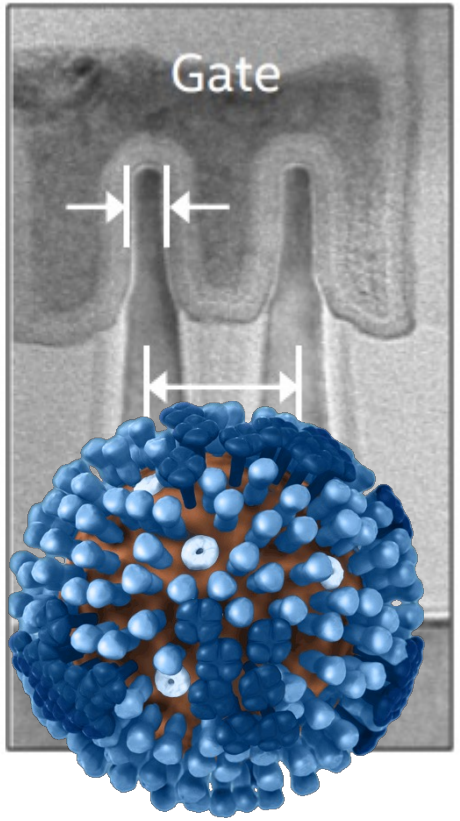
https://www-ssl.intel.com/content/dam/www/public/us/en/documents/technology-briefs/bohr-14nm-idf-2014-brief.pdf

We've come a long way since Gordon Moore proposed his famous law

An electron microscope image of a single Intel transistor using the 14 nm process

8 nm Fin Width

42 nm Fin Pitch



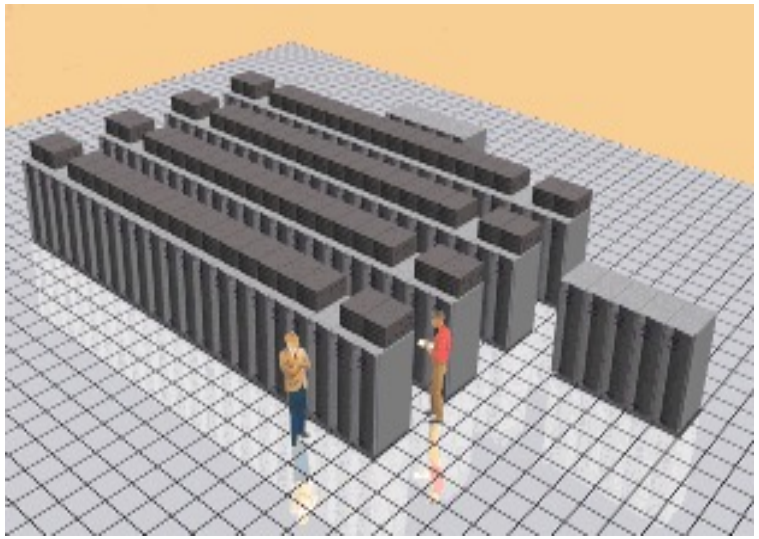
We put billions of these transistors on a single chip

An influenza virus is around 100 nm across!
http://www.cdc.gov/flu/images/h1n1/3D_Influenza/3D_Influenza_transparent_no_key_full_med2.gif

https://www-ssl.intel.com/content/dam/www/public/us/en/documents/technology-briefs/bohr-14nm-idf-2014-brief.pdf

Moore's Law: A personal perspective

First TeraScale* computer: 1997



Intel's ASCI Option Red

Intel's ASCI Red Supercomputer

9000 CPUs

one megawatt of electricity.

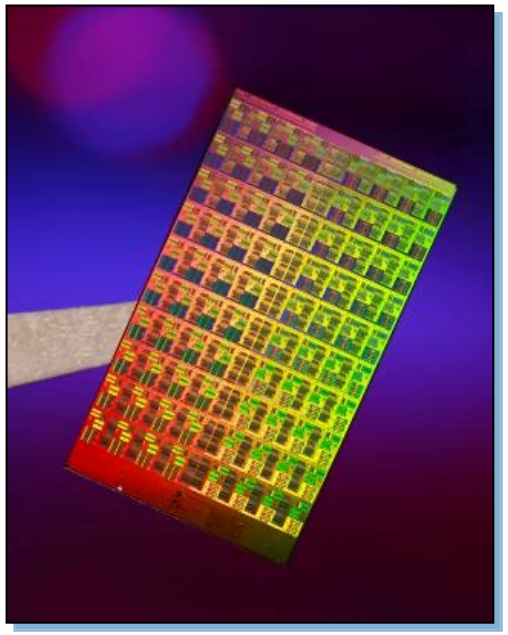
1600 square feet of floor space.

*Double Precision TFLOPS running MP-Linpack

A TeraFLOP in 1996: The ASCI TeraFLOP Supercomputer, Proceedings of the International Parallel Processing Symposium (1996), T.G. Mattson, D. Scott and S. Wheat.

10 years later

First TeraScale% chip: 2007



Intel's 80 core teraScale Chip

1 CPU

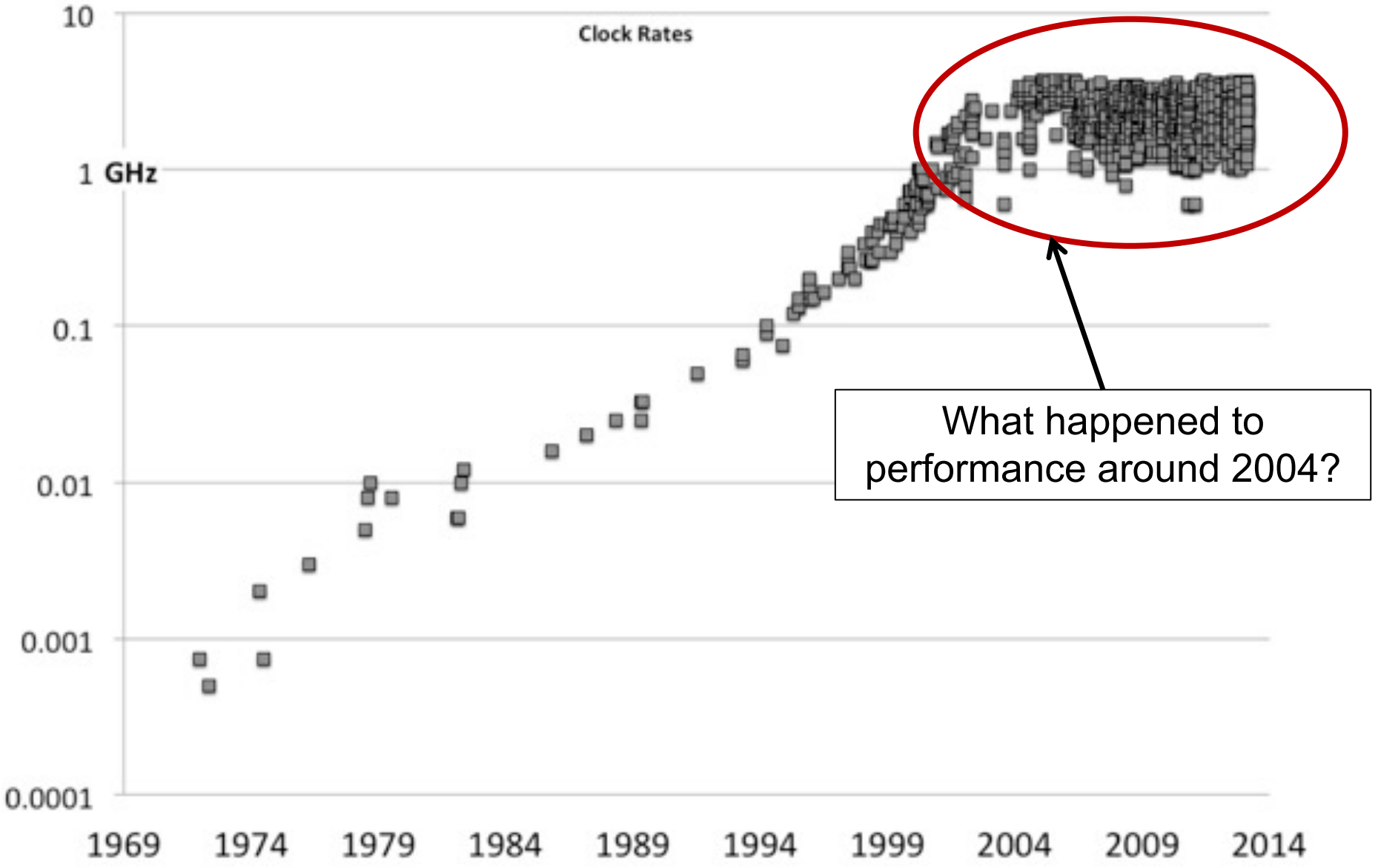
97 watt

275 mm2

%Single Precision TFLOPS running stencil

Programming Intel's 80 core terascale processor SC08, Austin Texas, Nov. 2008, Tim Mattson, Rob van der Wijngaart, Michael Frumkin

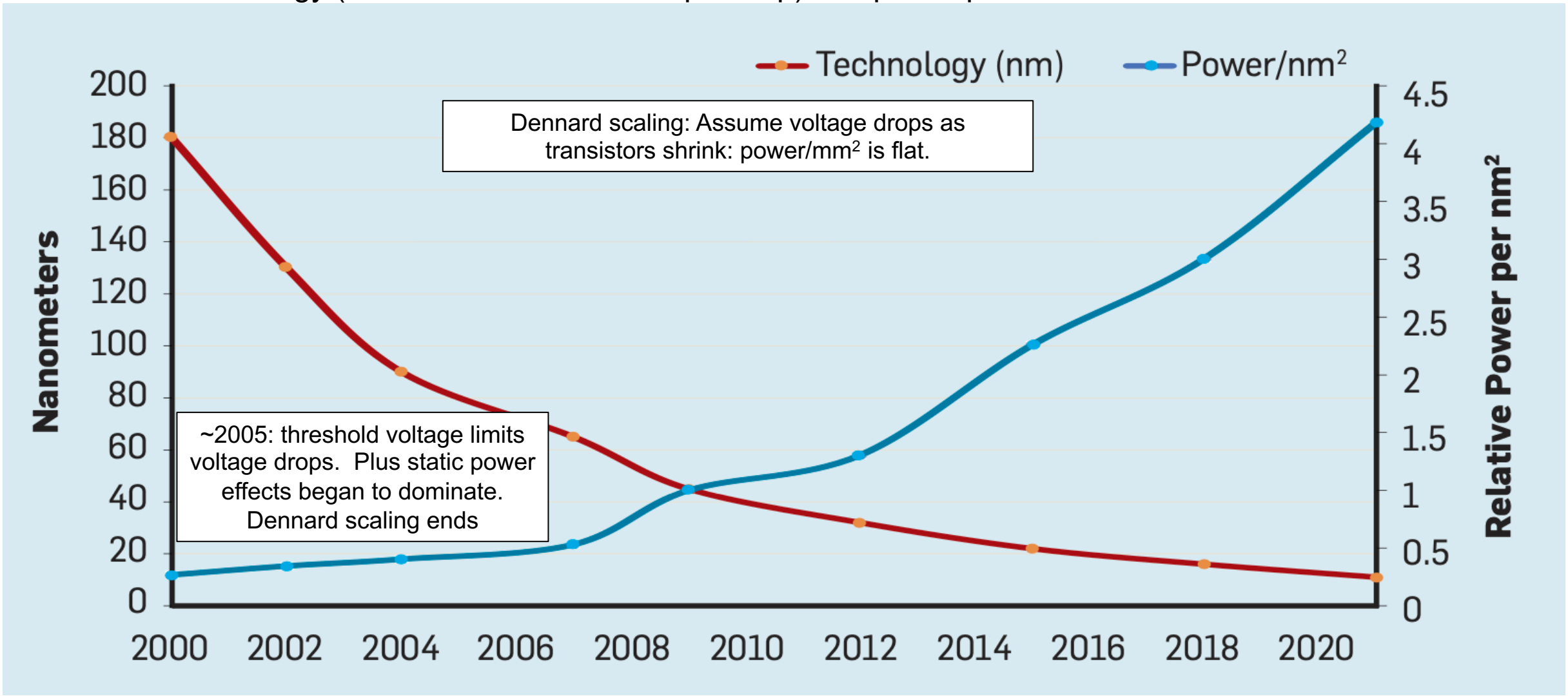
CPU Frequency (GHz) over time (years)



Source: James Reinders (from the book “structured parallel programming”)

Dennard Scaling

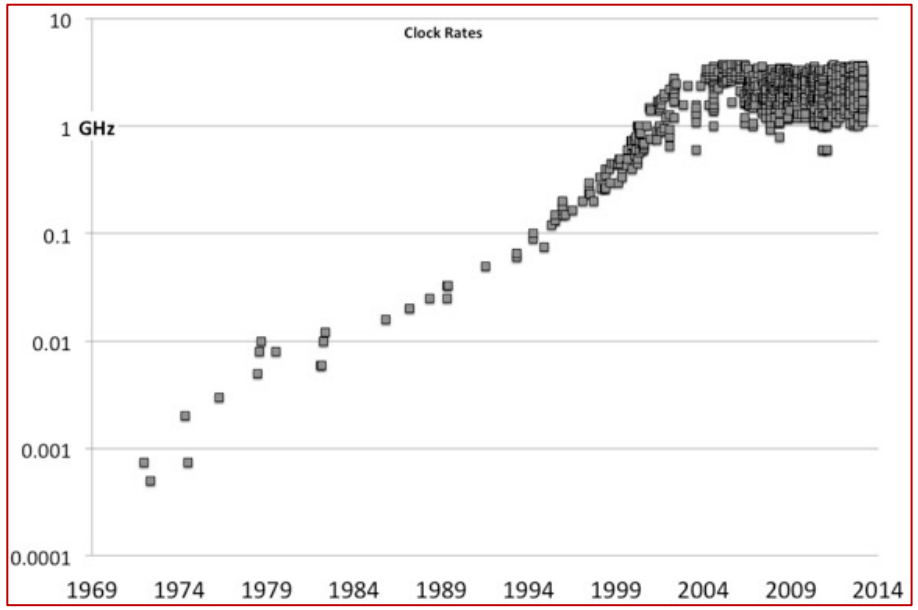
- Process technology (translates to Transistors per chip) and power per mm^2

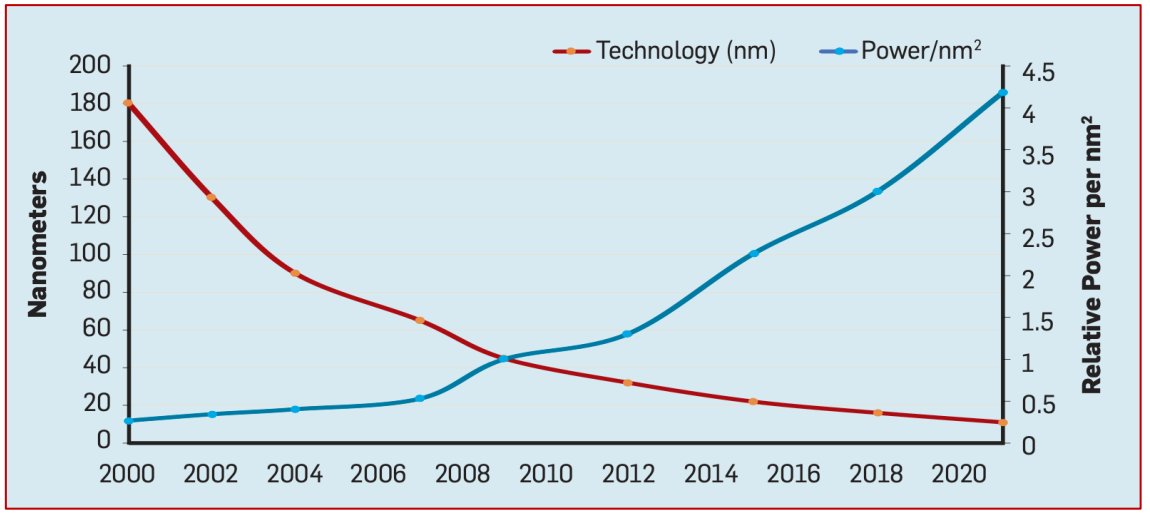


Process technology nodes defined by the smallest feature on a chip (i.e. gate length in nm). After 22 nm, it's become a marketing term that doesn't map to a specific feature's length.

Image from "A New Golden Age for Computer Architecture", John Hennessy and David Patterson, CACM, 2019

Growing Performance in a post-Dennard-Scaling world?





As we'll see in our next lecture, the path forward is parallel computing …
spreading your work across many processing elements.

Parallel Programming is the essence of HPC. At ESC, we'll explore parallel programming in detail:

- OpenMP: a particularly simple API for CPU programming
- Threaded Building Blocks (TBB): a C++ based approach for task parallel programing on CPUs
- CUDA: GPU programming using NVIDIA's CUDA programing model.
- MPI: parallel programming across distributed memories … the standard model for clusters

Summary … Getting the most from our processors

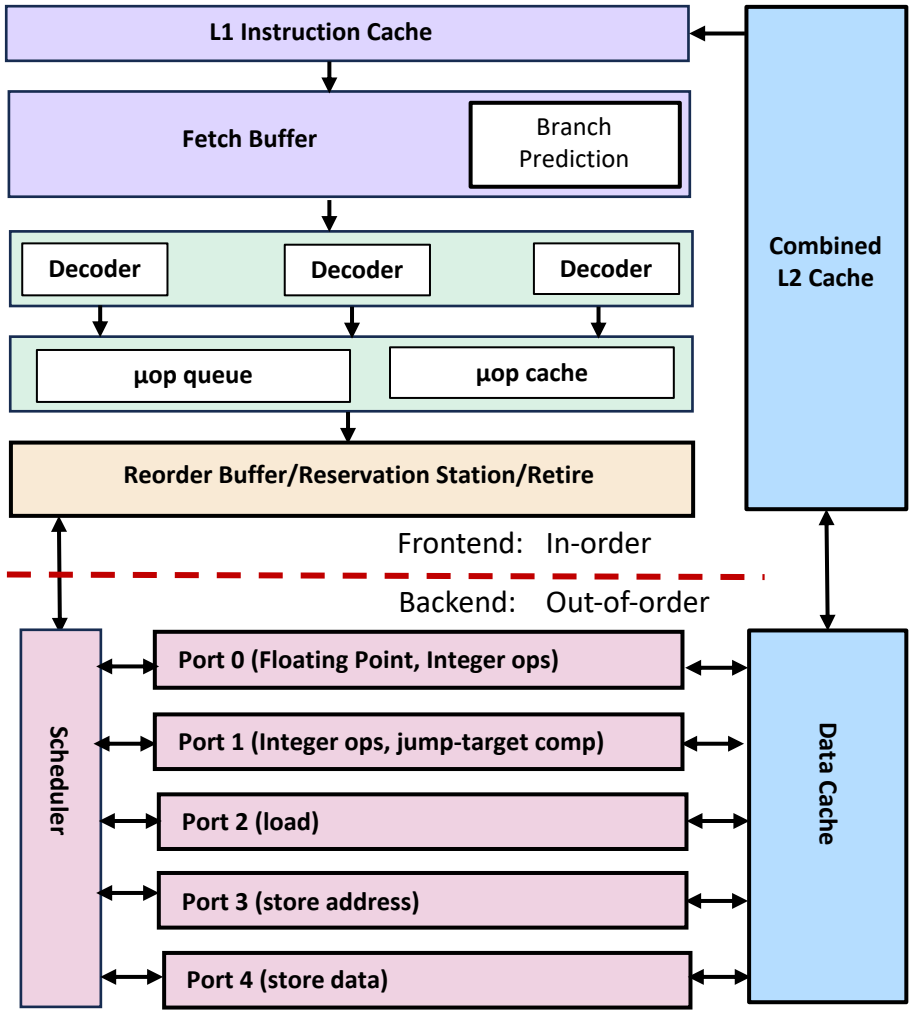
Getting the most from a CPU

We can think of a CPU in two parts: A **front-end** and a **back-end**

The CPU Front-end is where instructions are **fetched** and **decoded**

Helping the Front-end

- Avoid complex branching that jumps through the programs set of instructions.
- Keep code local. If possible, inline functions when possible
- Keep loops short so they fit in the µop cache



Intel® Pentium Pro™ microarchitecture

The CPU Back-end is where instructions **execute**

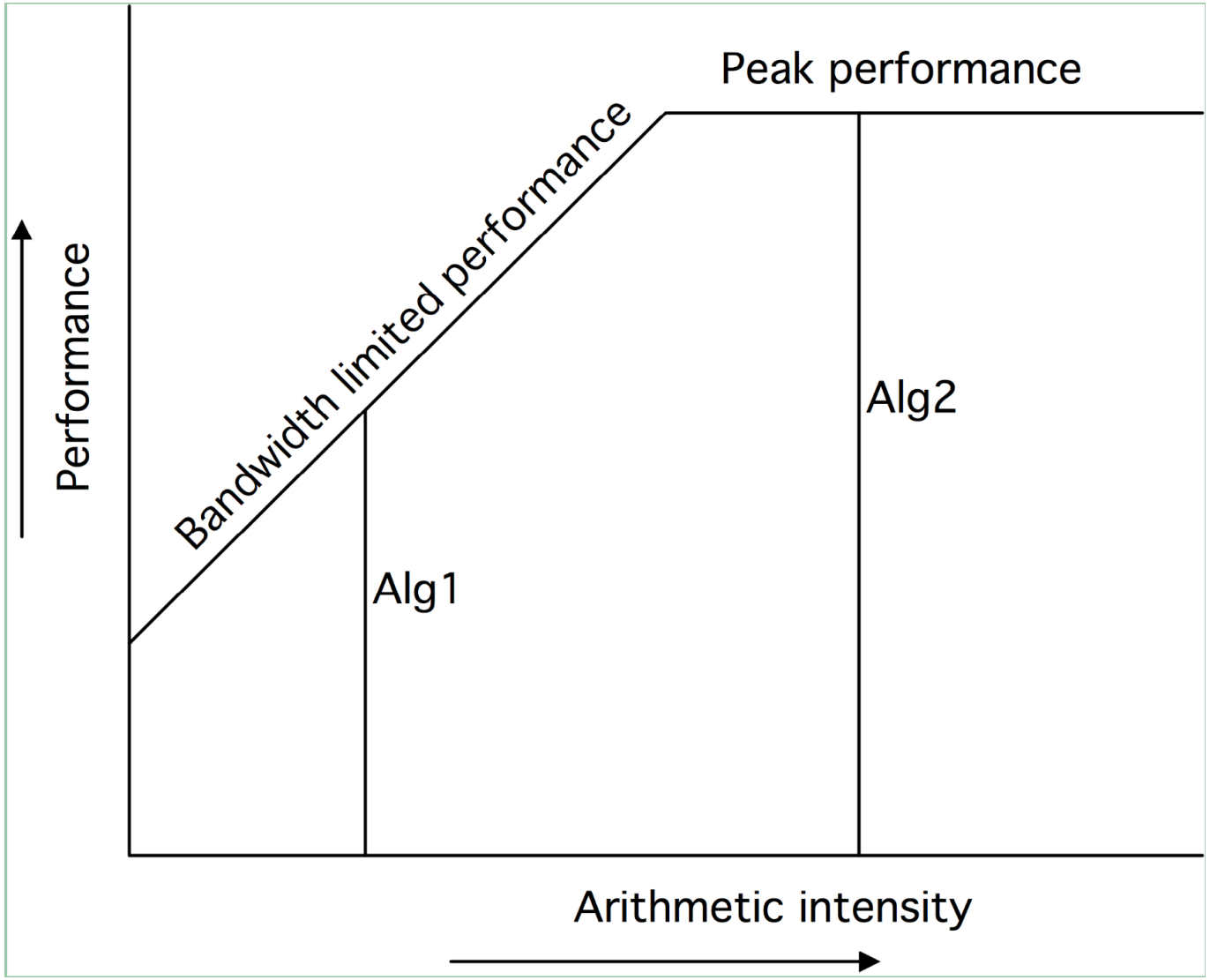
Helping the Back-end

- Organize data so you are computing from data in-cache
- Use all available functional units including vector-units (**topic of next lecture**)
- Remember … not all operations are created equally. Divisions and Square root ops cost 10 to 40 times the cost of a multiply.

Roofline plots: Understanding algorithm performance

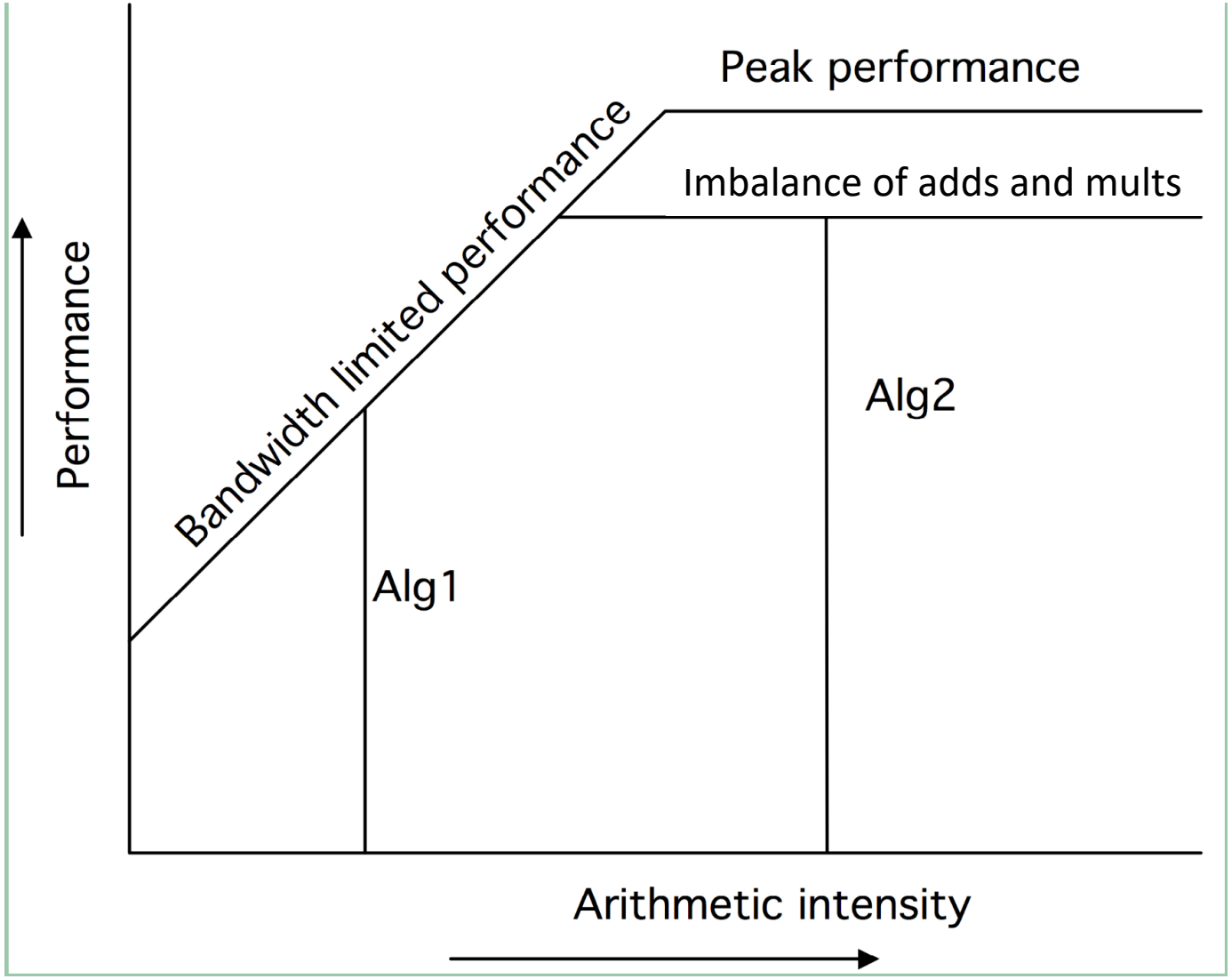
Different algorithms hit different performance limits:

- Alg1: compute bound
- Alg2: bandwidth bound.



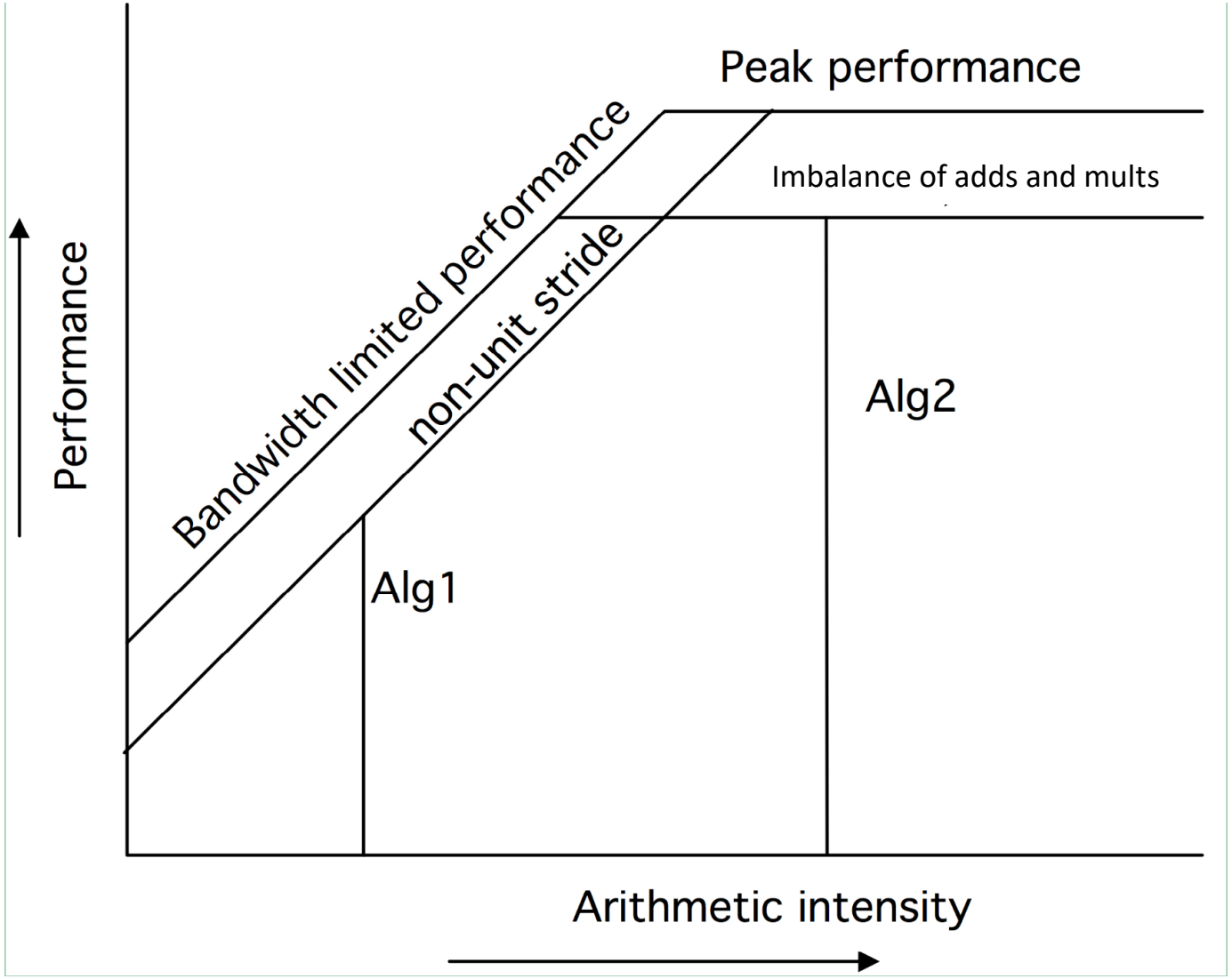
Arithmetic Intensity: operations per data transfer

Roofline plots: Understanding algorithm performance



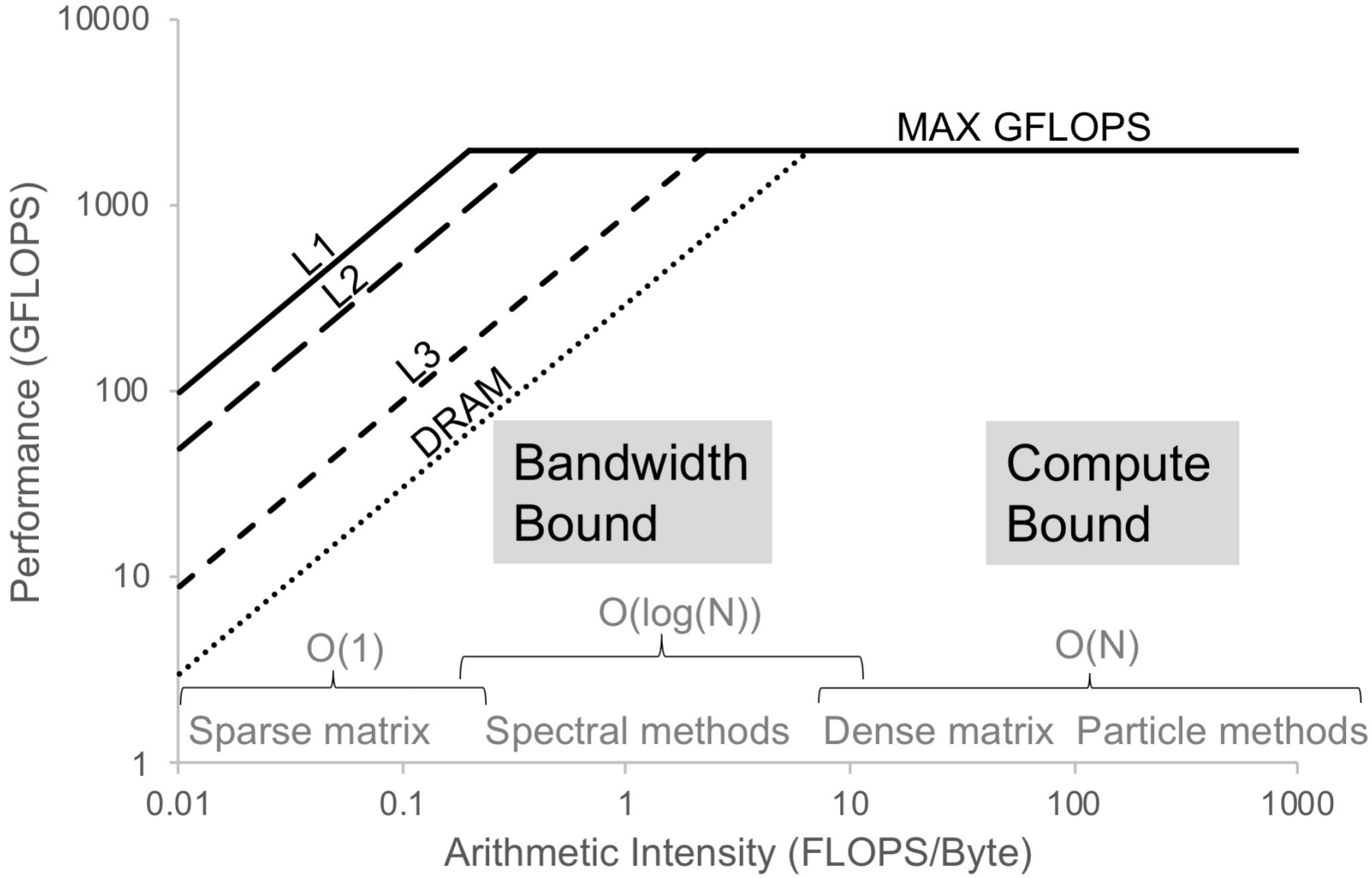
Algorithm 2 is not effectively using all the functional units.

Roofline plots: Understanding algorithm performance



Algorithm 1 has non-optimal data transfer behavior.

Roofline plots: Understanding algorithm performance



Conclusion … and a summary of the ESC technical program

- Scientific computing and HPC are tightly coupled … as you move from math to algorithm to software, performance needs to on your mind.

- Dennard Scaling has ended … so even as Moore's law marches on, we need parallelism to get higher performance (**a major topic of the next lecture**)

- At ESC we will cover the core topics every scientific computing professional needs to master:
  - Using all the resources of a processor to maximize performance
  - Writing, debugging, and optimizing C++ software.
  - Supporting Python by mapping python onto C++ functions
  - Mathematics on computers … from computer arithmetic to pseudo-random numbers
  - Effective memory management
  - Parallel programming for CPUs (OpenMP and TBB), GPUs (CUDA and a quick survey of other GPU programming models), and clusters (MPI)