CMPEN 431
Computer Architecture
Fall 2022

Abstractions, Technology, and Performance

Kiwan Maeng

[Adapted from *Computer Organization and Design, 5th Edition*,
Patterson & Hennessy, © 2014, Morgan Kaufmann]

Before We Start: A Little Bit About Myself…

- Name: Kiwan Maeng
  - Call me **Professor Maeng** (no Mr. Maeng or Kiwan)
- Pronouns: he/him
- PhD at Carnegie Mellon University (2016—2021)
  - Compilers/systems for low-power, energy-harvesting IoT devices
- Research Scientist at Meta AI (2021—2022)
  - Privacy-preserving ML training/inference (e.g., federated learning)
  - On-device AI training/inference

Course Administration

- Instructor: Kiwan Maeng (pronouns: he/him)

  W204B, Westgate Bldg
  Office Hours: Tu/Th, 1:30PM-2:30PM
- TAs: Shakya Chakrabarti (pronouns: he/him)

  Qian Ma (pronouns: he/him)

  Office Hours: posted on Canvas
- URL: Canvas
- Recommended

  Text: *Computer Org and Design*, 5th Ed., Patterson and Hennessy, 2014
- Slides: PDF on Canvas before the lecture
- ACK: Profs. Mary Jane Irwin, John Sampson, Mahmut Kandemir

Grading Information

- Grade determinants
  - 3 midterm exams (9/27, 10/27, 12/6) – 70%
  - No final exam
  - 3 Programming Projects – 20%
    - To be submitted on Canvas by 23:59 on the due date.
    - Students have 72h grace period, automatically consumed in 1h increment (1h and 1s late is 2h late) for each late submission.
    - After all the grace period is consumed, 50% score for less-than-1-day submissions, 0% for otherwise.
  - On-line (Canvas) quizzes – 5%
    - Score does not matter. Graded based on participated/not participated
  - In-class participation – 5%
    - Through Kahoot!
- Exams will be in-class (let us know **ASAP** if you cannot make it)
- Grades will be posted on Canvas
- Always send emails via **Canvas**
- Use Canvas' **discussion board** for discussions/questions.

Participation

- Participation will be evaluated through Kahoot!
- There will be at least one Kahoot! Quiz every class.
- **Participate with your PSU ID** (e.g., kvm6242) to get the participation score.
- If you participate in **at least one** Kahoot! Quiz, you will get full participation score for that class. **If not, you will get zero score for that class**.
  - If you were at class but could not participate due to technical difficulty, **let me know after the class before I leave**!
- All the participation scores will be added and normalized and rounded to the scale of 0--5 at the end of the semester.

Participation – Example

- Today, it will not be graded.

Keep the Class Friendly and Inclusive

- No harassment or bullying.
- Do not mock other students or be judgmental.
- Be respectful to other students.

- I maintain **zero-tolerance policy** – any form of harassment or bullying will be taken to the appropriate school-level attention.

Academic Integrity & Other Policies

- **Academic integrity materials and quizzes are in Canvas.**
  - Must be finished by **this Friday**!
- Quizzes must be answered without discussing with others
- Exams must be taken without discussing with others
- Projects must be done without sharing the code
  - Discussing general ideas are okay.
  - Helping other students for a tool/environment setup is okay.
- Whenever not sure, ask a question at Canvas!

- I maintain **zero-tolerance policy** – any academic integrity violation will be immediately reported to the department.

Academic Integrity

- Academic integrity is a core value at Penn State. Policies guiding what behaviors are considered to be acting in accordance with community standards for academic integrity exist at multiple levels within the institution, and all must be heeded.
- The University defines academic integrity as the pursuit of scholarly activity in an open, honest and responsible manner. All students should act with personal integrity, respect other students' dignity, rights and property, and help create and maintain an environment in which all can succeed through the fruits of their efforts (refer to Senate Policy 49-20). Dishonesty of any kind will not be tolerated in this course. Dishonesty includes, but is not limited to, cheating, plagiarizing, fabricating information or citations, facilitating acts of academic dishonesty by others, having unauthorized possession of examinations, submitting work of another person or work previously used without informing the instructor, or tampering with the academic work of other students. Students who are found to be dishonest will receive academic sanctions and will be reported to the University's Office of Student Conduct for possible further disciplinary sanctions (refer to Senate Policy G-9).
- The CSE department has a departmental academic integrity statement that can be found here: http://www.eecs.psu.edu/students/resources/EECS-CSE-Academic-Integrity.aspx

Masking Requirement and Covid

- While COVID-19 cases have decreased substantially since fall of 2021, COVID-19 remains a pandemic. More transmissible variants are a major concern. Penn State urges everyone to continue to take steps to protect not only themselves, but their colleagues, friends, and the campus by practicing good hand hygiene, staying home if you are sick, being up to date on vaccinations and boosters, and wearing a mask indoors. There is evidence that masks are effective in reducing the transmission of COVID-19 (e.g., Li et al., 2020, Lima et al., 2020, Talic et al., 2021) and everyone is strongly encouraged to wear masks while indoors.

TAs

- Shakya Chakrabarti and Qian Ma
  - Contact them via Canvas
- TAs will
  - Prepare quizzes
  - Prepare programming projects/exams and grade them
  - Answer questions at Canvas
  - Hold office hours
- Any questions/concerns about homeworks or programming projects?
  - Contact TAs first and then me
- Any concerns your exam score?
  - Contact TAs first and then me

Course Content

- CPU design, pipelining, cache/memory hierarchy design, multiprocessor/multicore architectures, storage.
  - "This course will introduce students to the architecture-level design issues of a computer system. They will apply their knowledge of digital logic design to explore the high-level interaction of the individual computer system hardware components. Concepts of sequential and parallel architecture including the interaction of different memory components, their layout and placement, communication among multiple processors, effects of pipelining, and performance issues, will be covered. Students will apply these concepts by studying and evaluating the merits and demerits of selected computer system architectures."
  - The Goal: To learn what determines the capabilities and performance of computer systems and to understand the interactions between the computer's architecture and its software so that future software designers (compiler writers, operating system designers, database programmers, application programmers, …) can achieve the best cost-performance trade-offs and so that future computer architects understand the effects of their design choices on software.

What You Should Know – 271, 331, 311

- Basic logic design and machine organization
  - logical minimization, FSMs, component design
  - processor, memory, I/O
- Create, assemble, run, debug programs in an assembly language
  - MIPS preferred
- Create, compile, and run C (C++, Java, Python) programs
- Create, organize, and edit files and run programs on Unix/Linux
- **We have Quiz 0 for testing your background knowledge on these topics (deadline: this Friday).**

Academic Integrity Quiz + Quiz 0 is Out! (Due 8/26)

- Academic Integrity Quiz must be finished before anything else.
  - You must get a **100% score** (multiple attempts allowed).
- HW0 tests your background knowledge.
  - Score does not matter; do not feel pressured and let me know what you do and do not know.
- Again: **No grace period for Quizzes**.
  - You do it on time, you get the full score.
  - You miss the deadline, you get zero score.

Why Study Computer Architecture (Now)?

- Computer architecture has been studied for a long time…
- Also, you might not want to become a computer architect anyways.
- Why should I (or you) still study it?
  - New applications are introduced (e.g., ML, graph, metaverse, …).
  - New design goals emerge (e.g., security/privacy, energy-efficiency, sustainability, …).
  - Understanding of the lower layer is essential in realizing a better design at a higher layer (e.g., compiler, OS, application, …)
  - It is fun to know how things work!

Classes of Computers

- Desktop/laptop (PC)/tablet computers
  - Single user
  - General purpose, variety of software/applications
  - Subject to cost/performance/power tradeoff
- Servers/Clouds/Data Centers/Supercomputers
  - Multiple, simultaneous users
  - Network based, terabytes of memory, petabytes of storage
  - High capacity, performance, reliability/availability, security
  - Range from small servers to building sized
- Embedded computers (processors)
  - Hidden as components of systems, used for one predetermined application (or small set of applications)
  - Stringent power/performance/cost constraints
  - Lots of IPs
  - Handhelds, wearables (fitness bands, smart watches, smart glasses), IoTs

Embedded Processor Characteristics

The largest class of computers spanning the widest range of applications and performance

- Often have minimum performance requirements
- Often have stringent limitations on cost and form factor
- Often have stringent limitations on power consumption
- Often have low tolerance for failure

The Post PC Era

- Personal mobile devices (PMDs)
  - Battery operated, touch screen (no mouse, no keyboard)
  - Connects to the Internet, download “apps”
  - A few hundreds of dollars (or less) in cost
  - Smart phones, tablets, electronic glasses, cameras, …
- Cloud computing
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS) and Function as a Service (FaaS) deployed via the Cloud
  - Parts of software run on a PMD and parts in the Cloud
  - Multicloud, hybrid cloud
  - Amazon, Google, Microsoft, …

Design Goals

- What are some of the important design goals of a computer hardware?

Design Goals

- Function
  - Needs to be correct (witness the Pentium FDIV divider bug) http://en.wikipedia.org/wiki/Pentium_FDIV_bug
    - Unlike software, its difficult to update once deployed
  - Varies as to what functions should be supported as “base” hardware primitives
- High performance
  - Can’t measure performance by just looking at the clock rate
  - What matters is the performance for the intended applications (real-time, server, etc.)
    - An impossible goal is to have the fastest possible design for all applications
- Power (energy) consumption
  - Energy in – battery life, cost of electricity
  - Energy out – cooling costs, machine room costs

Design Goals, Continued

- Low cost
  - Per unit manufacturing costs (wafer cost)
  - Mask costs and design costs
- Reliability
  - Once verified as functioning correctly, does it continue to? For how long between failures?
  - Hard (permanent) faults versus transient faults
  - Reliability demands may be application specific
- Form factor
  - Critical for embedded systems

In electronics, a **wafer** is a thin slice of semiconductor, such as a crystalline silicon, used for the fabrication of integrated circuits

Challenge is to balance the relative importance of the design goals

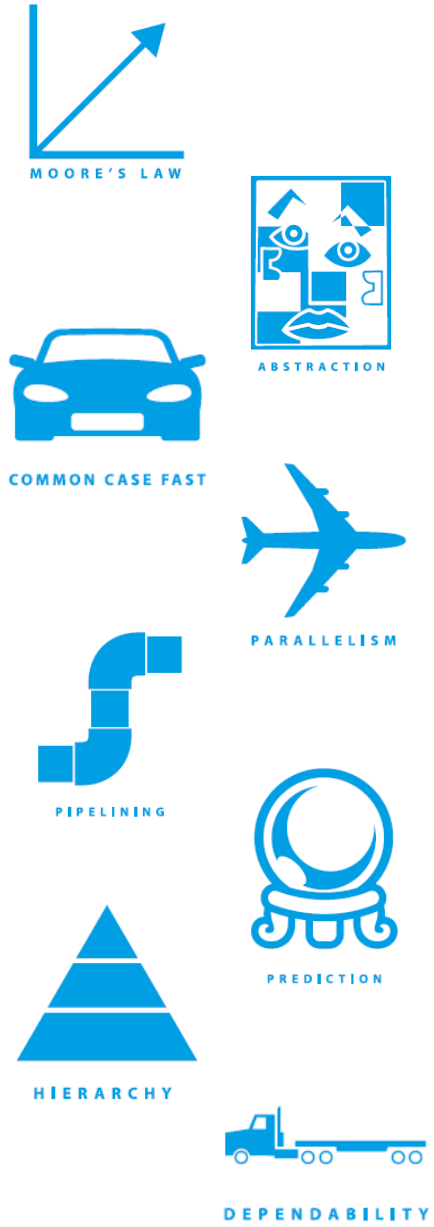
Design Goals, Newer!

- Security/Privacy
  - Must not have security vulnerabilities
    - e.g., Meltdown & Spectre (2018)
  - Run only the desired application & Do not leak data outside.
    - e.g., Intel SGX, AMD SEV, Arm TrustZone, Nvidia Hopper Architecture
- Sustainability (greener hardware)
  - Some hardware structure emits more carbon during manufacturing.
  - Manufacturing a typical rack server generates 6-9t CO2 (https://www.delltechnologies.com/asset/en-us/products/servers/technical-support/Full_LCA_Dell_R740.pdf).
- Programmability
  - GPU vs. TPU?

The target keeps moving!

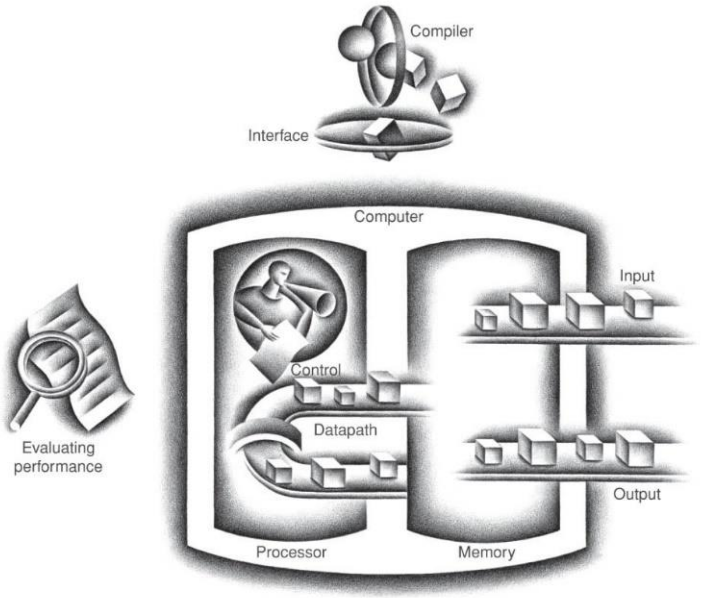
Eight Great Ideas in Computer Architecture

- Design for ***Moore's Law***
- Use ***abstraction*** to simplify design
- Make the ***common case fast***
- Performance *via* ***parallelism***
- Performance *via* ***pipelining***
- Performance *via* ***prediction***
- ***Hierarchy*** of memories
- ***Dependability*** *via* redundancy



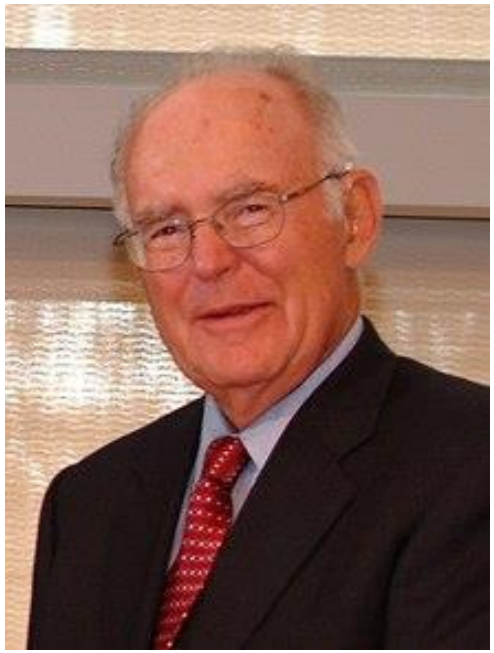
The Five Classic Components of a Computer

- Input: Data into memory
  - Keyboard, mouse, microphone, …
  - CD/DVD, Flash drive, …
  - Network adapters
- Output: Data out of memory
  - Display, speaker, …
  - CD/DVD, Flash drive, …
  - Network adapters
- Memory
- Datapath: a set of functional units that carry out data processing operations
- Control: determines the operation of others



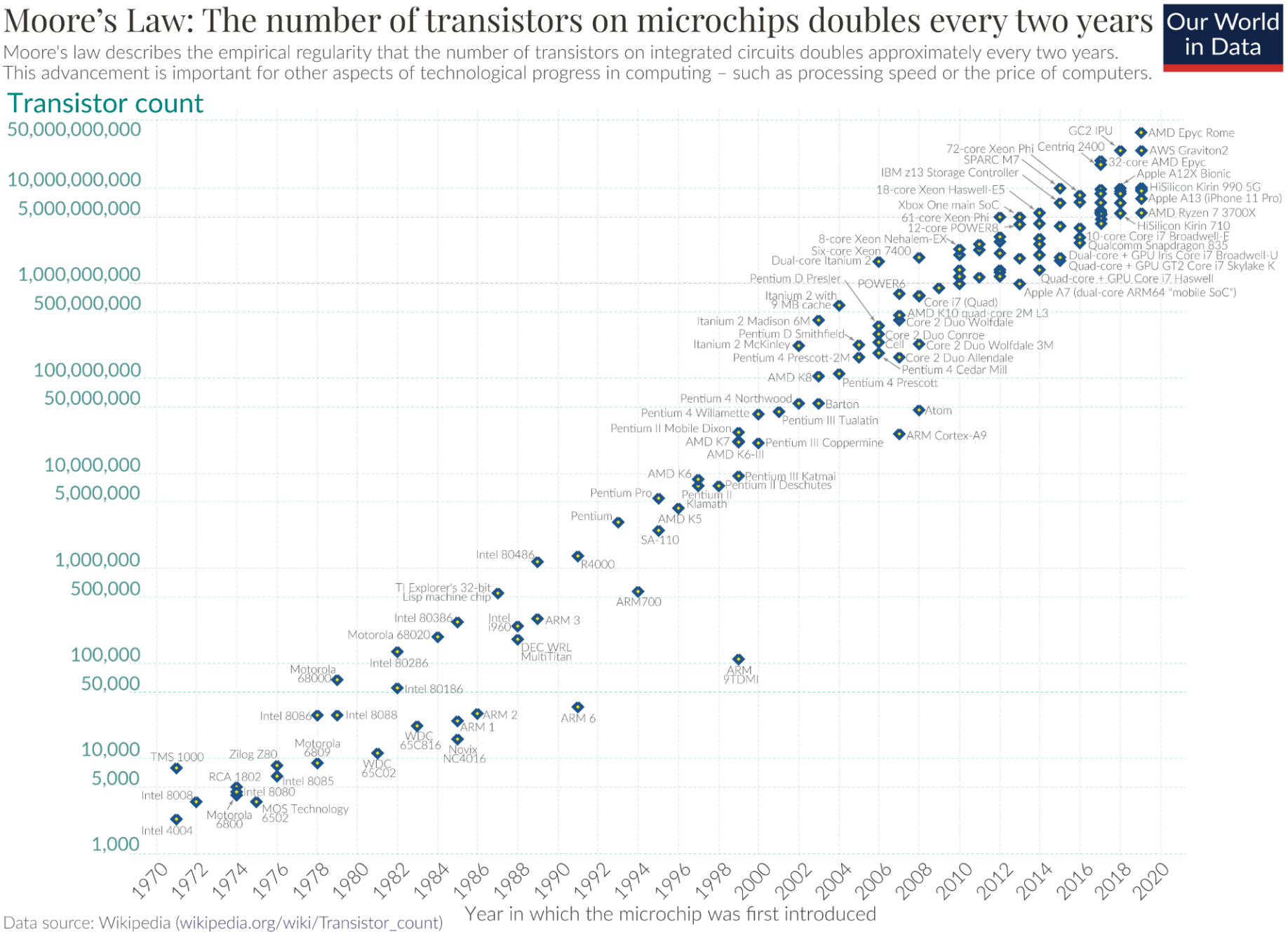
Moore's Law: 2X transistors / "2 years"

Transistor counts in an IC will double every (two) years!



Gordon Earle Moore
(Intel co-founder)

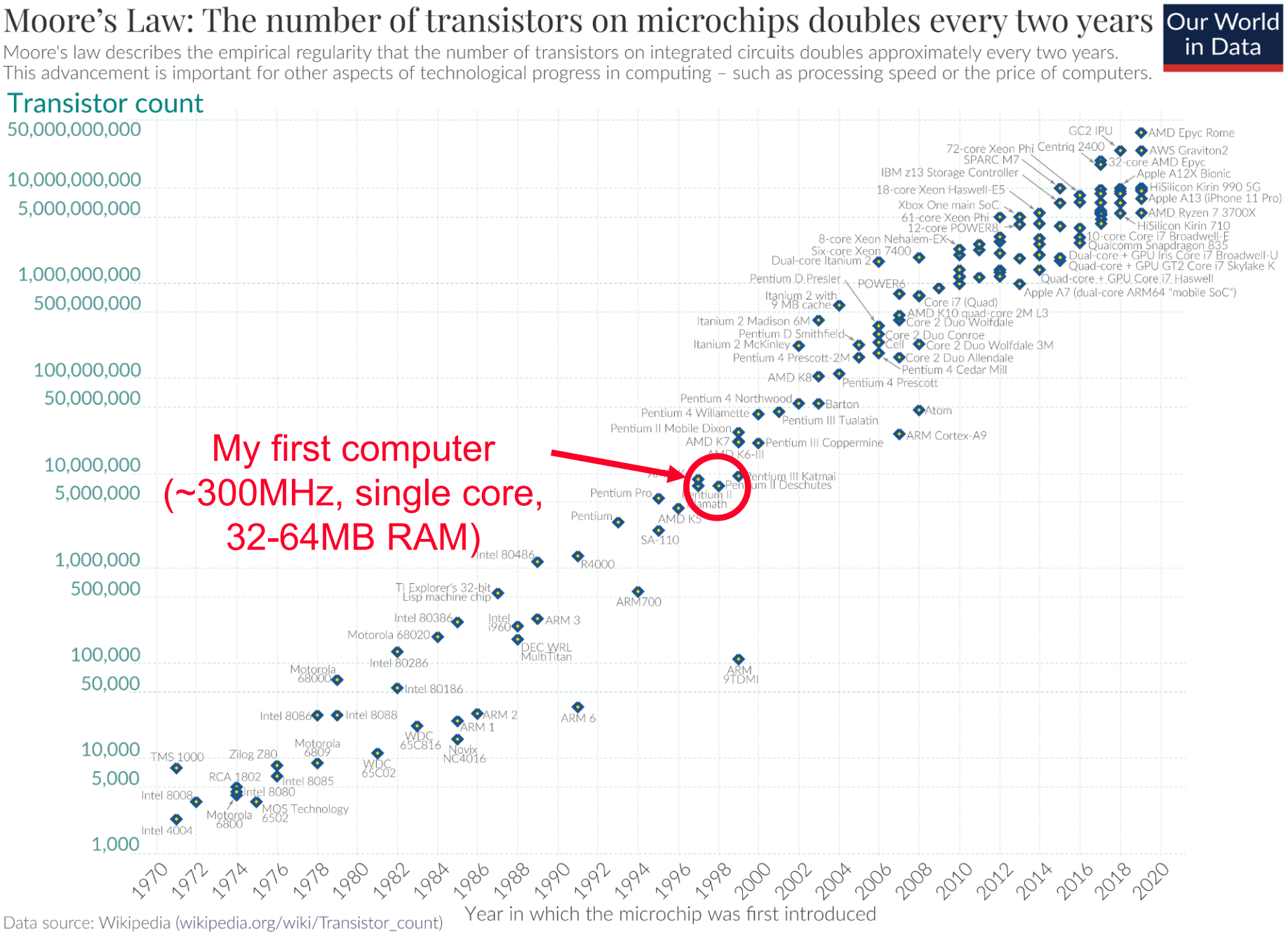
Moore’s Law: 2X transistors / “2 years”



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

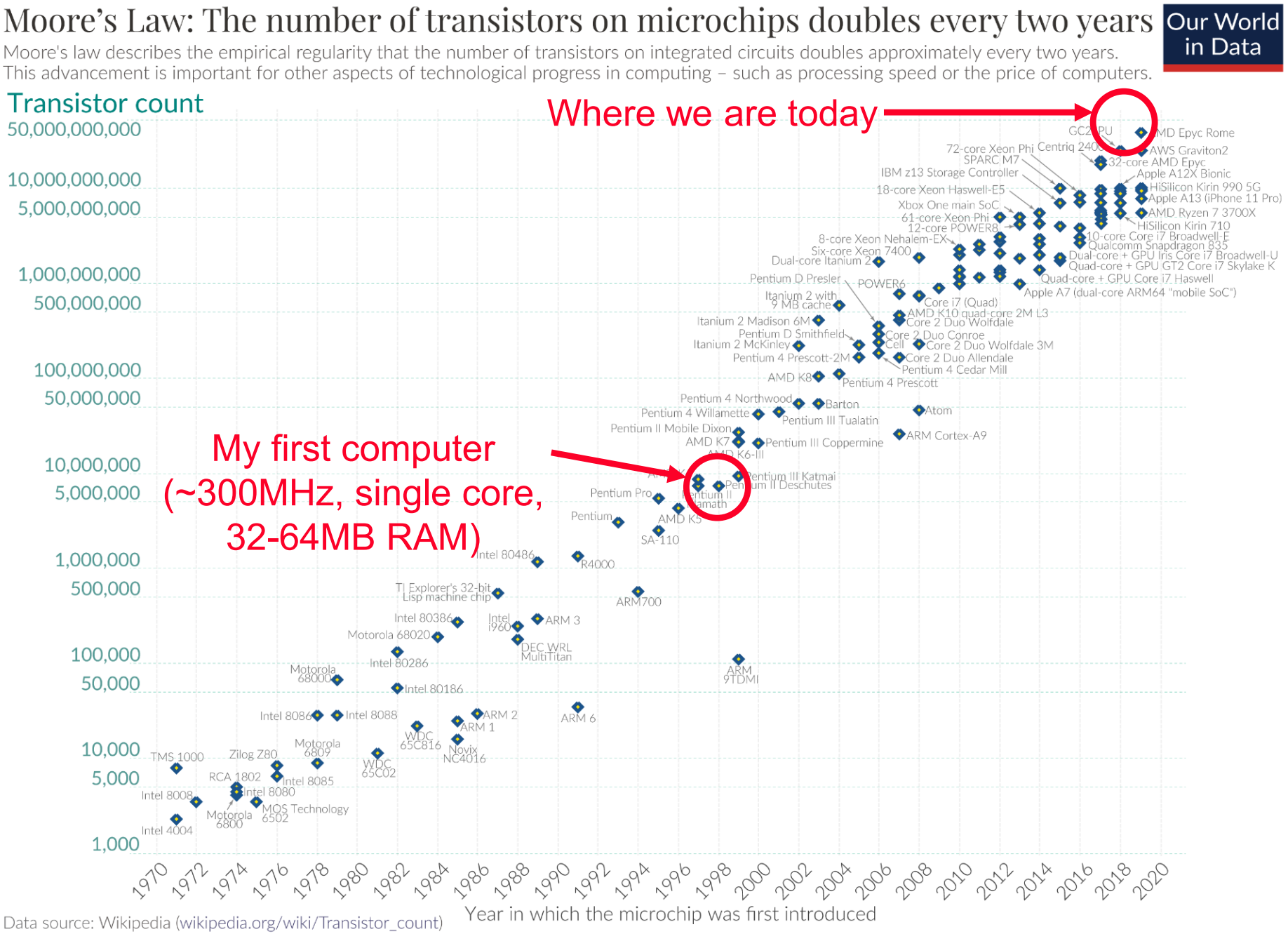
OurWorldinData.org – Research and data to make progress against the world’s largest problems.

Moore's Law: 2X transistors / "2 years"



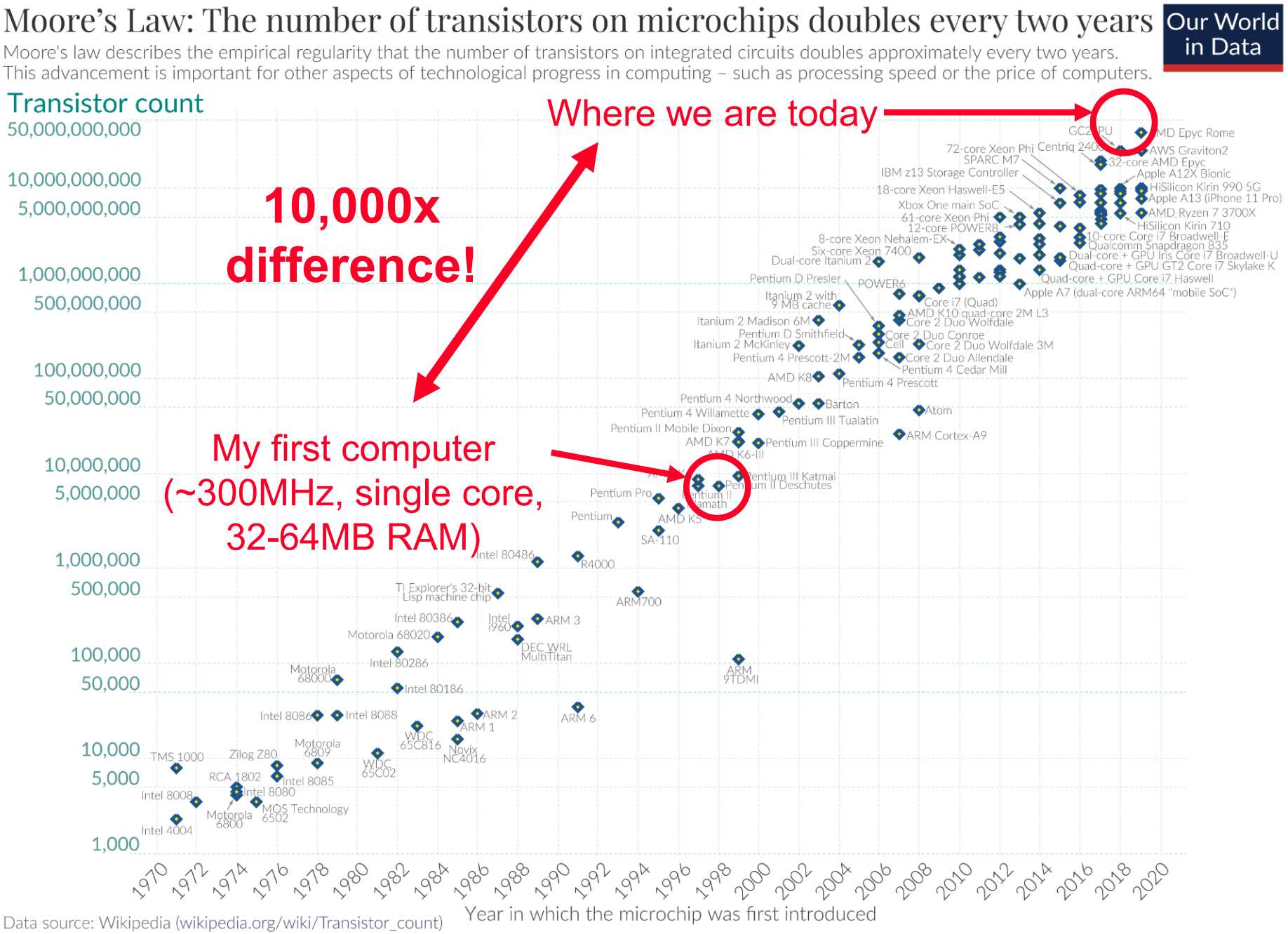
Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

Moore's Law: 2X transistors / "2 years"



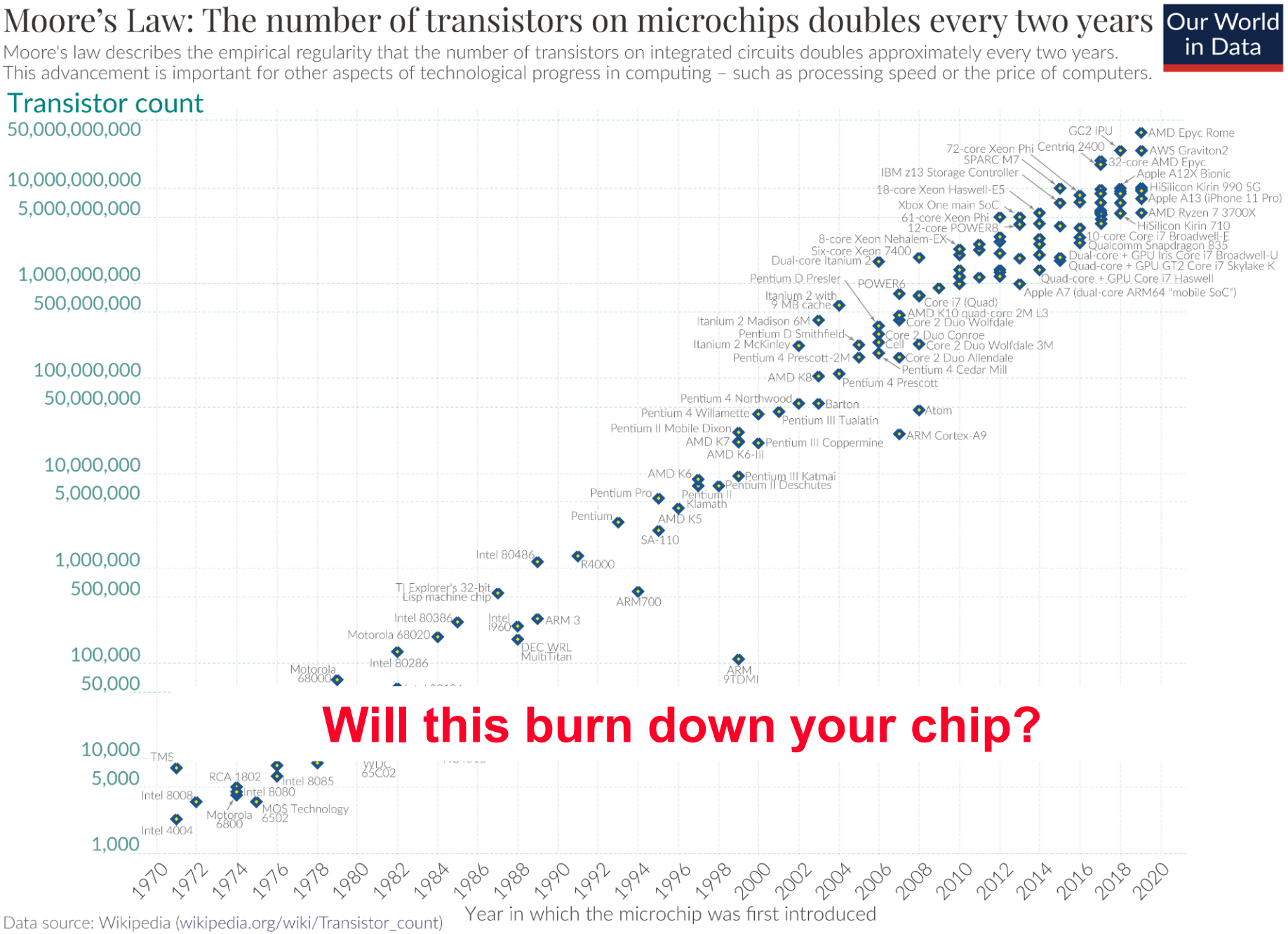
Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

Moore's Law: 2X transistors / "2 years"



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

Moore’s Law: 2X transistors / “2 years”

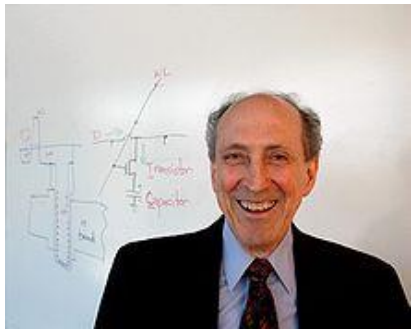


Will this burn down your chip?

Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world’s largest problems.

Dennard Scaling: Power Consumption Stays the Same

- P (power) = C (capacitance) * V^2 (voltage) * f (frequency)
- 2x transistor → 0.5x transistor area
- 0.5x area → 0.7x length (sqrt(0.5))
- 0.7x length → **0.7x voltage** (V = -E * d)
- 0.7x length → 0.7x circuit delay (v = d / t)
- 0.7x circuit delay → **1.4x frequency**
- 0.7x length, 0.5x area → **0.7x capacitance** (C = eps * A / d)
- 0.7x capacitance, 1.4x frequency, 0.7x voltage → 0.5x power per transistor
- 2x transistor, 0.5x power → **constant chip power**
- **2x transistor + 1.4x frequency every two years!**



Robert H. Dennard

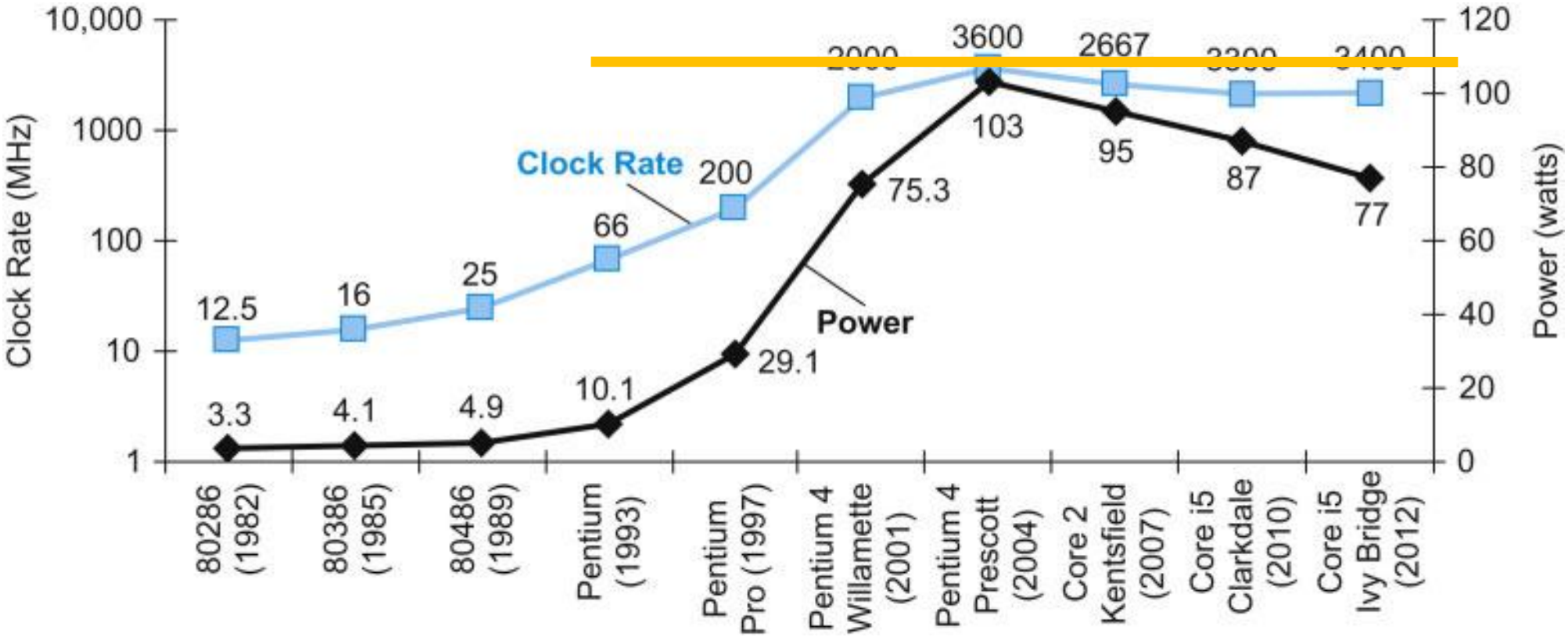
Dennard Scaling: Power Consumption Stays the Same

- P (power) = C (capacitance) * V^2 (voltage) * f (frequency)
- 2x transistor → 0.5x transistor area
- 0.5x area → 0.7x length (sqrt(0.5))
- 0.7x length → **0.7x voltage** (V = -E * d)
- 0.7x length → 0.7x circuit delay (v = d / t)
- 0.7x circuit delay → **1.4x frequency**
- 0.7x length, 0.5x area → **0.7x capacitance** (C = eps * A / d)
- 0.7x capacitance, 1.4x frequency, 0.7x voltage → 0.5x power per tran
- 2x trans r
- **2x transistor + 1.4x frequency every two years!**

Robert H. Dennard

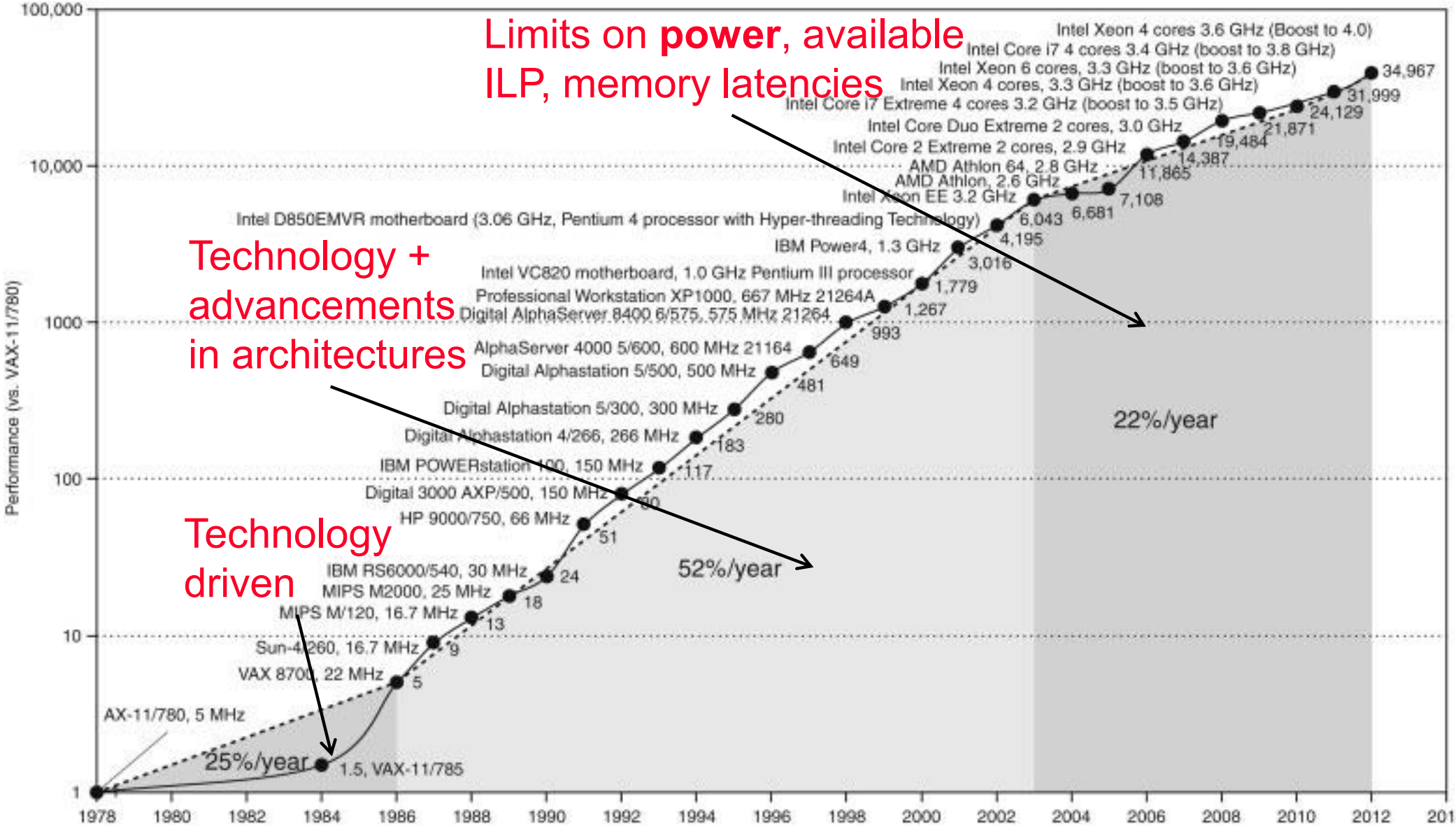
So, can we scale up the transistor count infinitely?

What Really Happened Instead: The Power Wall



- With reducing dimension and voltage, leakage current became non-negligible → power goes up instead of staying constant

Growth in Processor Performance (SPECint)

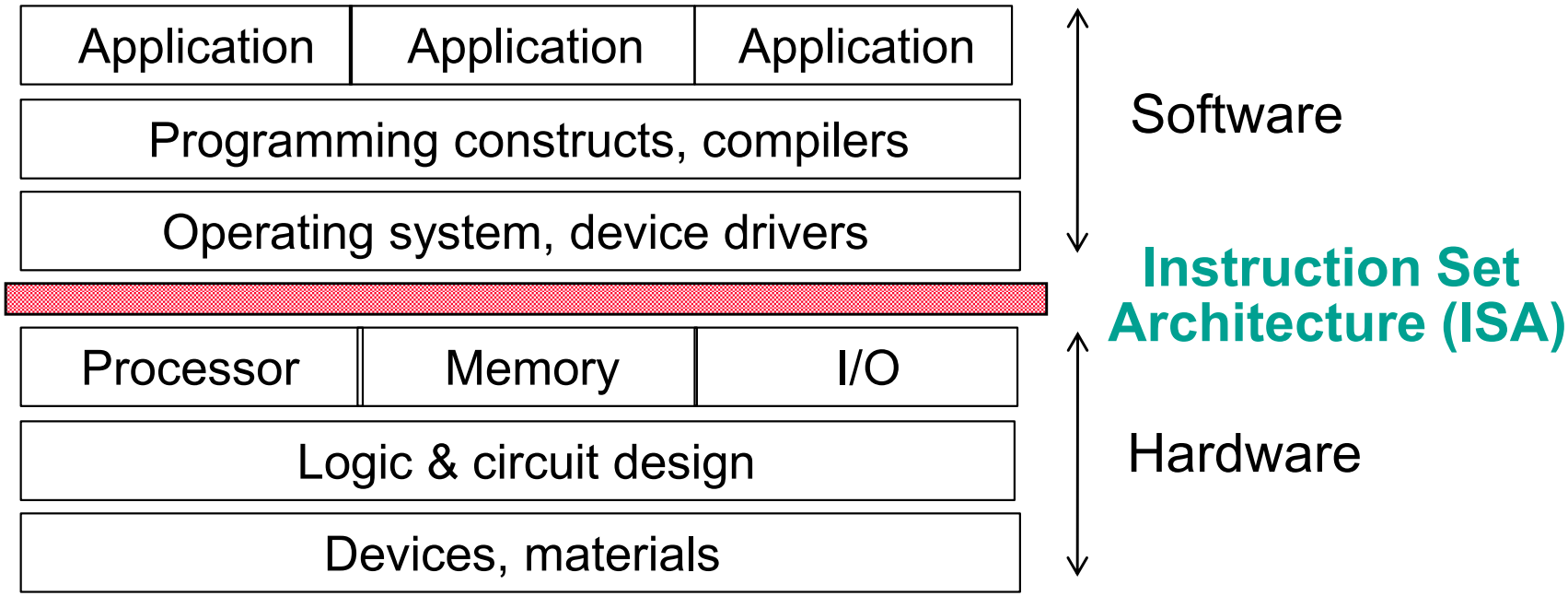


Where to go from here?

Abstraction and Layering

- Abstraction is the only way to deal with complex systems
  - Divide the processor into components, each with an
    - Interface: inputs, outputs, behaviors
    - Implementation: “black box” with timing information
- Layering the abstractions makes life even simpler
  - Divide the components into layers
    - Implement layer X using the interfaces of layer X-1
    - Don’t need to know the interfaces of layer X-2 (but sometimes it helps)
- Two downsides to layering
  - Inertia: layer interfaces become entrenched over time (“standards”) which are very difficult to change even if the benefit is clear
  - Opaque: can be hard to reason about performance

Abstraction and Layering in Architecture



- The ISA serves at the boundary between the software and hardware
  - Facilitates the parallel development of the software layers and the hardware layers
  - Lasts through many generations (portable)

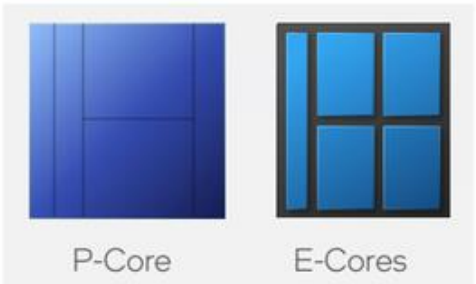
ISA (Instruction Set Architecture)

- An instruction set architecture (ISA), also called computer architecture, is an **abstract model of a computer**. A device that executes instructions described by that ISA, such as a central processing unit (CPU), is called an **implementation.** In general, an ISA defines the supported instructions, data types, registers, the hardware support for managing main memory, fundamental features (such as the memory consistency, addressing modes, virtual memory), and the input/output model of a family of implementations of the ISA. – Wikipedia
  - Enable different **implementation** and **compatibility** across hardware generations.

The Move to Multicore Processors

- The power challenge has forced a change in the design of microprocessors
- Modern microprocessors put multiple cores per chip (processor)

Intel i7 Alder Lake CPU



Multicore Performance Issues

- Private L1 caches, private or shared L2, … LL caches?
  - Best performance depends upon the applications and how much information they “share” in the cache, or how much they conflict in the cache
- Contention for memory controller(s) and port(s) to DRAM
- On-chip network hierarchy (NoC)
- Requires explicitly parallel programming (multiple (parallel) threads for one application)
  - Compare with instruction level parallelism (ILP) where the hardware executes multiple instructions at once (so hidden from the programmer; a.k.a. implicit parallelism)
  - Parallel programming for performance is hard to do
    - Load balancing across cores
    - Cache sharing/contention, contention for DRAM controller(s)
    - Have to optimize for thread communication and synchronization
  - Can compilers help?

Defining Performance

- Response time (execution time) – how long does it take to do a task
  - Important to individual users
- Throughput (bandwidth) – number of tasks completed per unit time
  - Important to datacenter managers and system administrators
- How are response time and throughput affected by
  1. Replacing the core with a faster version?
  2. Adding more cores?
- Our focus, for now, will be *response time*

Relative Performance

- To maximize performance, need to minimize execution time

$$\text{performance}_X = 1 / \text{execution\_time}_X$$

If computer X is n times faster than computer Y, then

$$\frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution\_time}_Y}{\text{execution\_time}_X} = n$$

- Decreasing response time almost always improves throughput

Relative Performance Example

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

We know that A is n times faster than B if

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = n$$

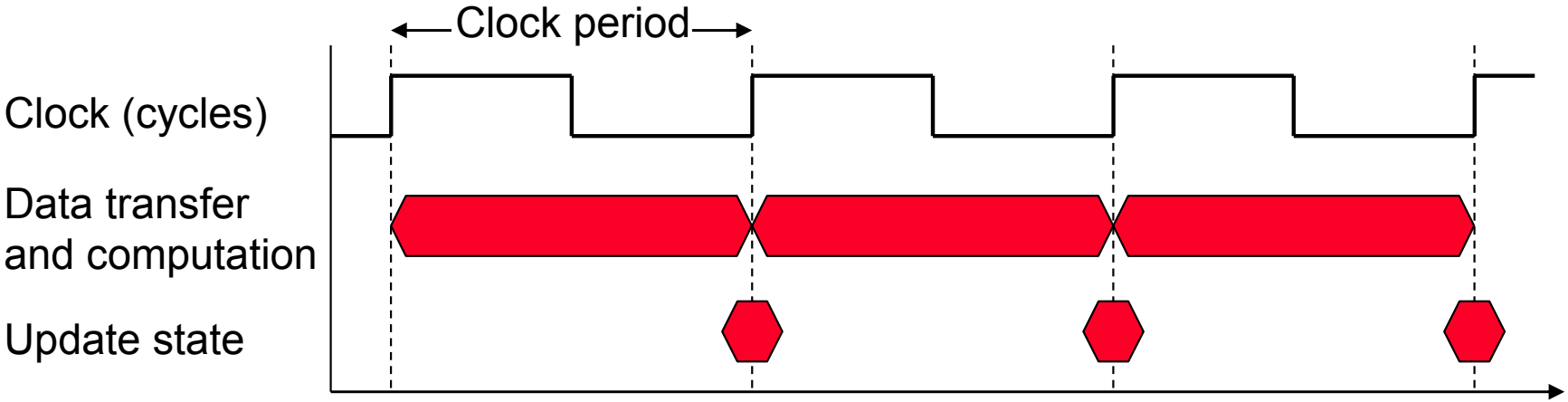
The performance ratio is

$$\frac{15}{10} = 1.5$$

So, A is said to be 1.5 times faster than B (or A is 50% faster than B!)

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- Clock period (cycle): duration of a clock cycle
  - E.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - E.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

Performance Factors

- CPU execution time (CPU time) – time the CPU spends working on a task (not including time waiting for I/O or running other programs)

$$\text{CPU execution time for a program} = \text{\# CPU clock cycles for a program} \times \text{clock cycle time}$$

or

$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{\text{clock rate}}$$

- Can improve performance by reducing either the length of the clock cycle (increasing clock rate) or reducing the number of clock cycles required for a program
- The architect must often trade off clock rate against the number of clock cycles for a program

Improving Performance Example

- A program runs on computer A with a 2 GHz clock in 10 seconds. What clock rate must computer B run at to run this program in 6 seconds? Unfortunately, to accomplish this, computer B will require 1.2 times as many clock cycles as computer A to run the program.

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{clock rate}_A}$$

$$\text{CPU clock cycles}_A = 10 \text{ sec} \times 2\text{x}10^9 \text{ cycles/sec} = 20 \times 10^9 \text{ clock cycles}$$

$$\text{CPU time}_B = \frac{1.2 \times 20 \times 10^9 \text{ clock cycles}}{\text{clock rate}_B}$$

$$\text{clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = 4 \text{ GHz}$$

Instruction Performance

- Not all instructions take the same amount of time to execute
  - One way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction

$$\text{\# CPU clock cycles for a program} = \text{\# Instructions for a program} \times \text{Average clock cycles per instruction}$$

- Clock cycles per instruction (CPI) – the <u>average</u> number of clock cycles each instruction takes to execute
  - A way to compare two different implementations of the same ISA

	Computer_A	Computer_B
Avg CPI	2	1.2

THE Performance Equation

- Our basic performance equation is then

$$\text{CPU time} = \text{Instruction\_count} \times \text{CPI} \times \text{clock\_cycle}$$

or

$$\text{CPU time} = \frac{\text{Instruction\_count} \times \text{CPI}}{\text{clock\_rate}}$$

- This equation separates the three key factors that affect performance
  - Can measure the CPU execution time by running the program
  - The clock rate is usually given
  - Can measure overall instruction count by using profilers/ simulators without knowing all of the implementation details
  - CPI varies by instruction type and ISA implementation for which we must know the implementation details
  - Note that instruction count is dynamic (i.e., *not* the number of program lines)

Average (Effective) CPI

- Computing the overall average CPI is done by looking at the different types of instructions and their individual cycle counts and averaging

$$\text{Overall effective CPI} = \sum_{i=1}^{n} (CPI_i \times IC_i)$$

  - Where IC_i is the count (percentage) of the number of instructions of class i executed
  - CPI_i is the number of clock cycles per instruction for that instruction class
  - n is the total number of instruction classes

- The overall effective CPI varies by instruction mix – a measure of the dynamic frequency of instructions for one or many programs

A Simple Performance Tradeoff Example

Op	Freq	CPI_i	Freq x CPI_i			
ALU	50%	1	.5	.5	.5	.25
Load	20%	5	1.0	.4	1.0	1.0
Store	10%	3	.3	.3	.3	.3
Branch	20%	2	.4	.4	.2	.4
Average (Effective) CPI			$\sum$ = 2.2	1.6	2.0	1.95

- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

  CPU time new = 1.6 x IC x CC so 2.2/1.6 means 37.5% faster

- How does this compare with using branch prediction to shave a cycle off the branch time?

  CPU time new = 2.0 x IC x CC so 2.2/2.0 means 10% faster

- What if two ALU instructions could be executed at once?

  CPU time new = 1.95 x IC x CC so 2.2/1.95 means 12.8% faster

Determinates of CPU Performance

CPU time = Instruction_count x CPI x clock_cycle

	Instruction_ count	CPI	clock_cycle
Algorithm	X		
Programming language	X	X	
Compiler	X	X	
ISA	X	X	X
Core organization		X	X
Technology			X

Workloads and Benchmarks

- Benchmarks – a set of programs that form a "workload" specifically chosen to measure performance

- SPEC (System Performance Evaluation Cooperative) creates standard sets of benchmarks starting with SPEC89. SPEC CPU2006 consists of 12 integer benchmarks (CINT2006) and 17 floating-point benchmarks (CFP2006).

  www.spec.org for (significantly) newer benchmarks

- There are also benchmark collections for power workloads (SPECpower_ssj2008), for mail workloads (SPECmail2008), for multimedia workloads (mediabench), GPUs (Rodinia) …

SPEC CINT2006 on Intel i7 (CR = 2.66GHz)

Name	ICx10^9	CPI	ExTime (sec)	RefTime (sec)	SPEC ratio
perl	2,252	0.60	508	9,770	19.2
bzip2	2,390	0.70	629	9,650	15.4
gcc	794	1.20	358	8,050	22.5
mcf	**221**	**2.66**	221	9,120	41.2
go	1,274	1.10	527	10,490	19.9
hmmer	**2,616**	0.60	590	9,330	15.8
sjeng	1,948	0.80	586	12,100	20.7
libquantum	659	**0.44**	109	20,720	190.0
h264avc	**3,793**	**0.50**	713	22,130	31.0
omnetpp	**367**	**2.10**	290	6,250	21.5
astar	1,250	1.00	470	7,020	14.9
xalancbmk	1,045	0.70	275	6,900	25.1
Geometric Mean					**25.7**

Comparing and Summarizing Performance

- How do we summarize the performance for an entire benchmark set with a single number?
  - First the execution times are normalized giving the "SPEC ratio" (bigger is faster, i.e., SPEC ratio is the inverse of execution time)
  - The SPEC ratios are then "averaged" using the geometric mean (GM)

$$GM = \sqrt[n]{\prod_{i=1}^{n} \text{SPEC ratio}_i}$$

- Guiding principle in reporting performance measurements is reproducibility – list everything another experimenter would need to duplicate the experiment (version of the operating system, compiler settings/flags, input set used, specific computer configuration (clock rate, cache sizes and speed, memory size and speed, etc.))

Speedup Measurements

- The speedup of the SS core is
  - Assumes the cores have the same IC & CC

$$\text{speedup} = s_n = \frac{\text{\# scalar cycles}}{\text{\# superscalar cycles}}$$

- To compute average speedup performance can use
  - Geometric mean

$$GM = \sqrt[n]{\prod_{i=1}^{n} s_i}$$

  - Harmonic mean

$$HM = n / \left(\sum_{i=1}^{n} 1/s_i \right)$$

    - assigns a larger weighting to the programs with the smallest speedup
  - EX: two programs with same scalar cycles, with a SS speedup of 2 for program1 and 25 for program2
    - GM = $\sqrt{(2 * 25)} = 7.1$
    - HM = $2 / (.5 + .04) = 2 / .54 = 3.7$

Why Not Arithmetic Mean?

App	Machine 1	Machine 2
A	1	2
B	2	4
C	3	6
D	4	1

- Normalizing to Machine 1

App	Machine 1	Machine 2
A	1 (1.0)	2 (0.5)
B	2 (1.0)	4 (0.5)
C	3 (1.0)	6 (0.5)
D	4 (1.0)	1 (4.0)
AM	1.0	**1.375**
GM	1.0	**0.841**

- Normalizing to Machine 2

App	Machine 1	Machine 2
A	1 (2.0)	2 (1.0)
B	2 (2.0)	4 (1.0)
C	3 (2.0)	6 (1.0)
D	4 (0.25)	1 (1.0)
AM	**1.5625**	1.0
GM	**1.1892**	1.0

Amdahl's Law

- Used to determine the maximum expected improvement to overall system performance when only part of the system is improved

$$T_{improved} = \frac{T_{affected}}{improvemen\ t\ factor} + T_{unaffected}$$

- How much faster must the multiplier be to get a 2x performance improvement overall if multiples account for 20 seconds of the 100 second run time?

- Corollary: Make the common case fast

Summary: Evaluating ISAs

- Design-time metrics
  - Can it be implemented, in how long, at what cost (size, power)?
  - Can it be programmed? Ease of compilation?
- Static Metrics
  - How many bytes does the program occupy in memory?
- Dynamic Metrics
  - How many instructions are executed? How many bytes does the corefetch to execute the program?
  - How many clocks are required per instruction?
  - How "lean" (fast) a clock is practical?

Best Metric: Time to execute the program!

depends on the instructions set, the processor organization, and compilation techniques.