Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Fall 2019

4190.308: Computer Architecture

Course Information

- Schedule
  - 9:30 – 10:45 (Tuesday & Thursday) – Also several supp. classes on Friday nights
  - Lecture room: Engineering Bldg. #301-203
  - 3 credits
  - Official language: English

- TA: Jae-Hoon Shim (mattjs@snu)
- SNU eTL system for exam/project scores
- http://csl.snu.ac.kr for announcements and lecture slides
- http://sys.snu.ac.kr for project submissions and automatic grading

About Me

- Jin-Soo Kim (김진수)
  - Professor @ CSE Dept.
  - Systems Software & Architecture Laboratory
  - Operating systems, storage systems, parallel and distributed computing, embedded systems, …

- E-mail: jinsoo.kim@snu.ac.kr
- Tel: 02-880-7302
- Office: Engineering Bldg. #301-520 (office hours: Tuesday & Thursday)
  But, space remodeling is in progress until the end of September!
- The best way to contact me is by email

Myths About This Course

- It's an introductory course
  - Introduction to Computers?
  - About 20% of students have dropped every semester
- It's all about hardware
  - It's about how to separate work between software and hardware, and about how to design the interface between them
- It's not relevant for software engineers
  - Writing good software requires understanding details of underlying implementation
- Who needs to know assembly language these days?
  - Well, you'll see

Prerequisites

- Prerequisites
  - Programming Practice (4190.103A) – C programming
  - Logic Design (M1522.000700) – Must!
  - Data Structure (M1522.000900) – Recommended
- You should be familiar with the followings:
  - Shells and basic Linux commands
  - C (and Python!) programming skills
  - Basic knowledge on digital circuits and systems
- Accessible Linux (Ubuntu 18.04.3 LTS or similar) or MacOS machine

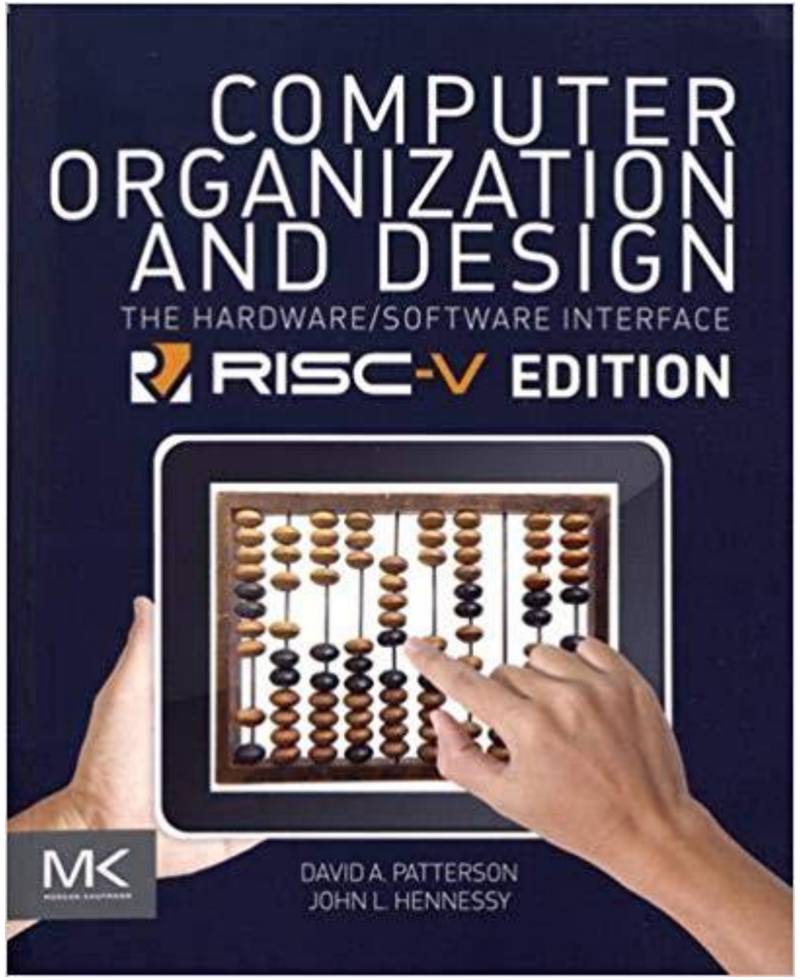
Policies for Non-major Students

- Your course registration form ("초안지") will be accepted only if …
  - You have an experience on C programming and debugging on Linux (gcc/gdb) and
  - You are familiar with Python and
  - You have already taken the "Logic Design" course

- Other introductory CSE courses for non-major students:
  - M1522.000600 Computer Programming
  - M1522.000700 Logic Design
  - M1522.000900 Data structures
  - 4190.101 Discrete Mathematics
  - 4190.103 Programming Practice

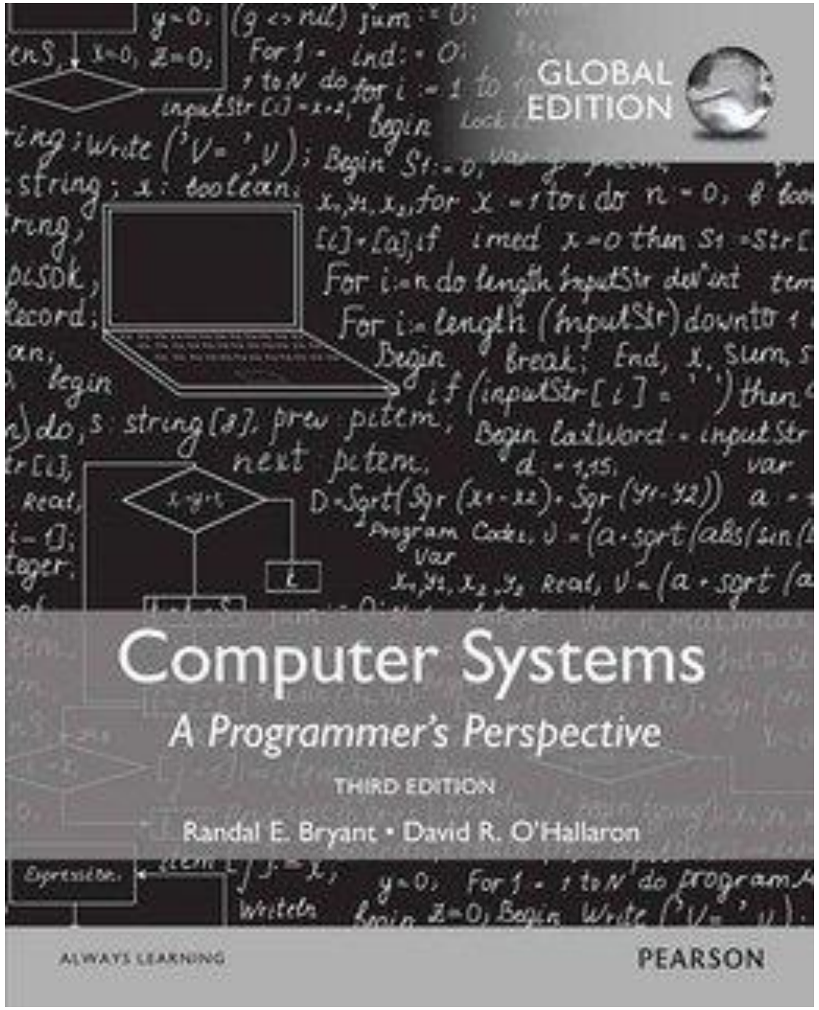
New Textbook

- Computer Organization and Design: The Hardware/Software Interface (RISC-V Edition)
  - David A. Patterson and John L. Hennessy (Turing Award Recipients in 2017)
  - First Edition
  - Morgan Kaufmann, 2017
  - http://booksite.elsevier.com/9780128122754/



Previous Textbook

- Computer Systems:
  A Programmer's Perspective
  - Randal E. Bryant and David R. O'Hallaron
  - Third Edition
  - Pearson Education Limited, 2016
  - Based on x86-64
  - http://csapp.cs.cmu.edu

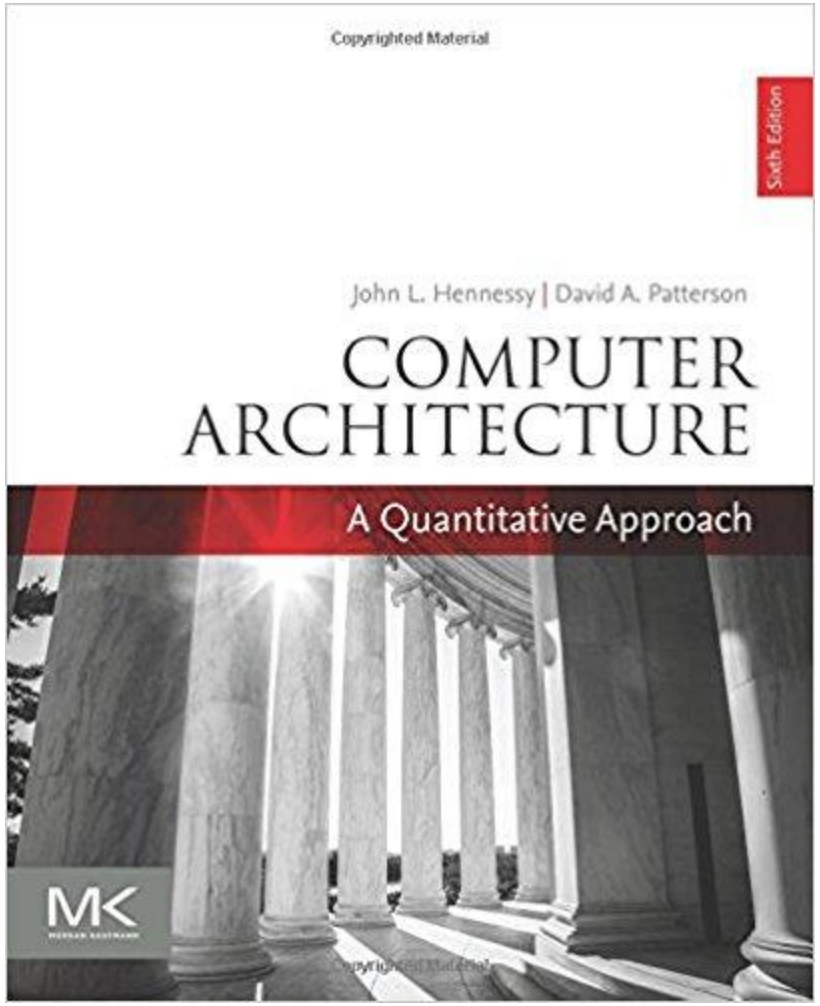


Reference

- Computer Architecture:
  A Quantitative Approach

  - John L. Hennessy and David A. Patterson
  - Sixth Edition
  - Morgan Kaufmann, 2017

  - http://booksite.elsevier.com/9780128119051



Topics

- Introduction to Computer Architecture
- Integers and Floating Points
- RISC-V Instruction Set Architecture
- Sequential Architecture
- Pipelined Architecture
- Cache
- Virtual memory
- I/O
- Parallel Computer Architecture

Projects (subject to change)

- C programming
- RISC-V assembly programming
- Designing pipelined processor
- Optimizing RISC-V assembly programs for pipelined processor
- Cache simulation

Grading Policy (subject to change)

- Exams: 60%
  - Midterm: 25%
  - Final: 35%
- Projects: 40%

- University policy requires students to attend at least 2/3 of the scheduled classes. Otherwise, you'll fail this course.
- We are using the electronic attendance system.
- Also, if you miss one of the exams, you'll fail this course.

Cheating Policy

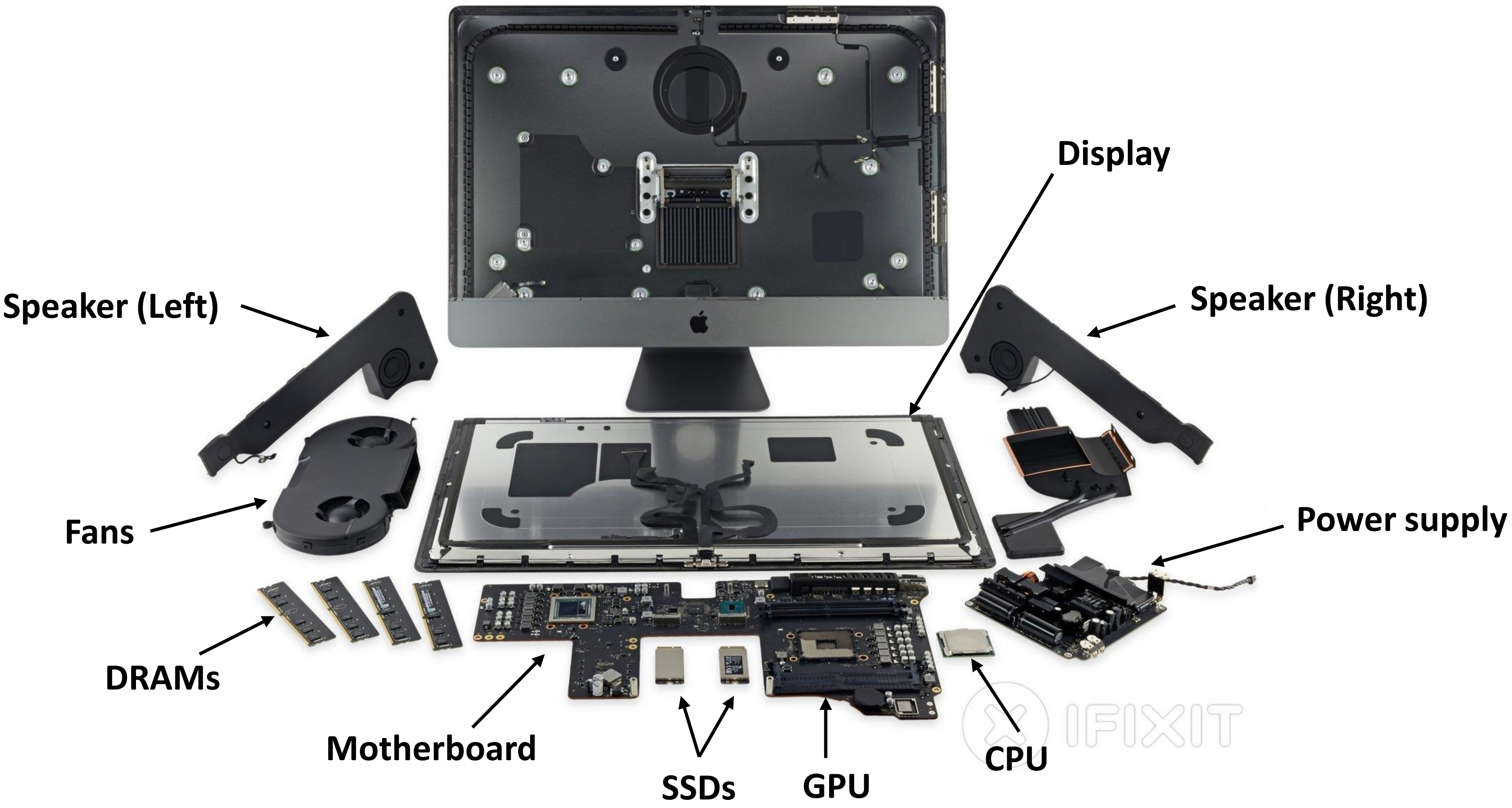
- What is cheating?
  - Copying another student's solution (or one from the Internet) and submitting it as your own
  - Allowing another student to copy your solution
- What is NOT cheating?
  - Helping others use systems or tools
  - Helping others with high-level design issues
  - Helping others debug their code
- Penalty for cheating
  - Severe penalty on the grade (F) and report to dept. chair
  - Ask helps to your TA or instructor if you experience any difficulty!

Introduction to Computer Architecture

Classes of Computers

- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Supercomputers
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to large data centers
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

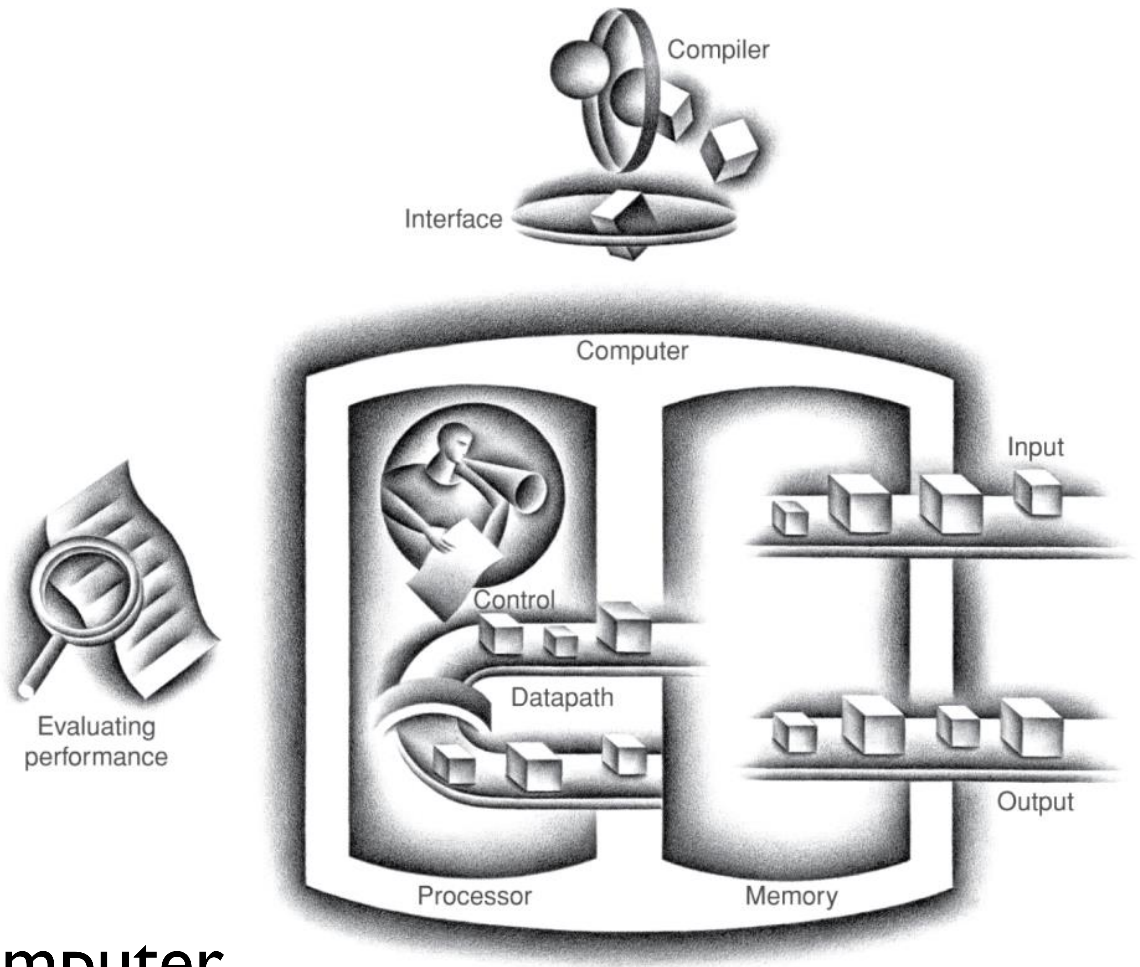
Opening the Box (iMac Pro)



Source: https://www.ifixit.com/Teardown/iMac+Pro+Teardown/101807

Components of a Computer

- CPU: Control + Datapath
- Memory
- I/Os
  - User-interface devices:
    Display, keyboard, mouse, sound, …
  - Storage devices:
    HDD, SSD, CD/DVD, …
  - Network adapters:
    Ethernet, 3G/4G/5G, WiFi, Bluetooth, …



- Same components for all kinds of computer

What Happened:

104 cabinets
(76 computes,
8 switches,
20 disks)

9298 cores

150m^2

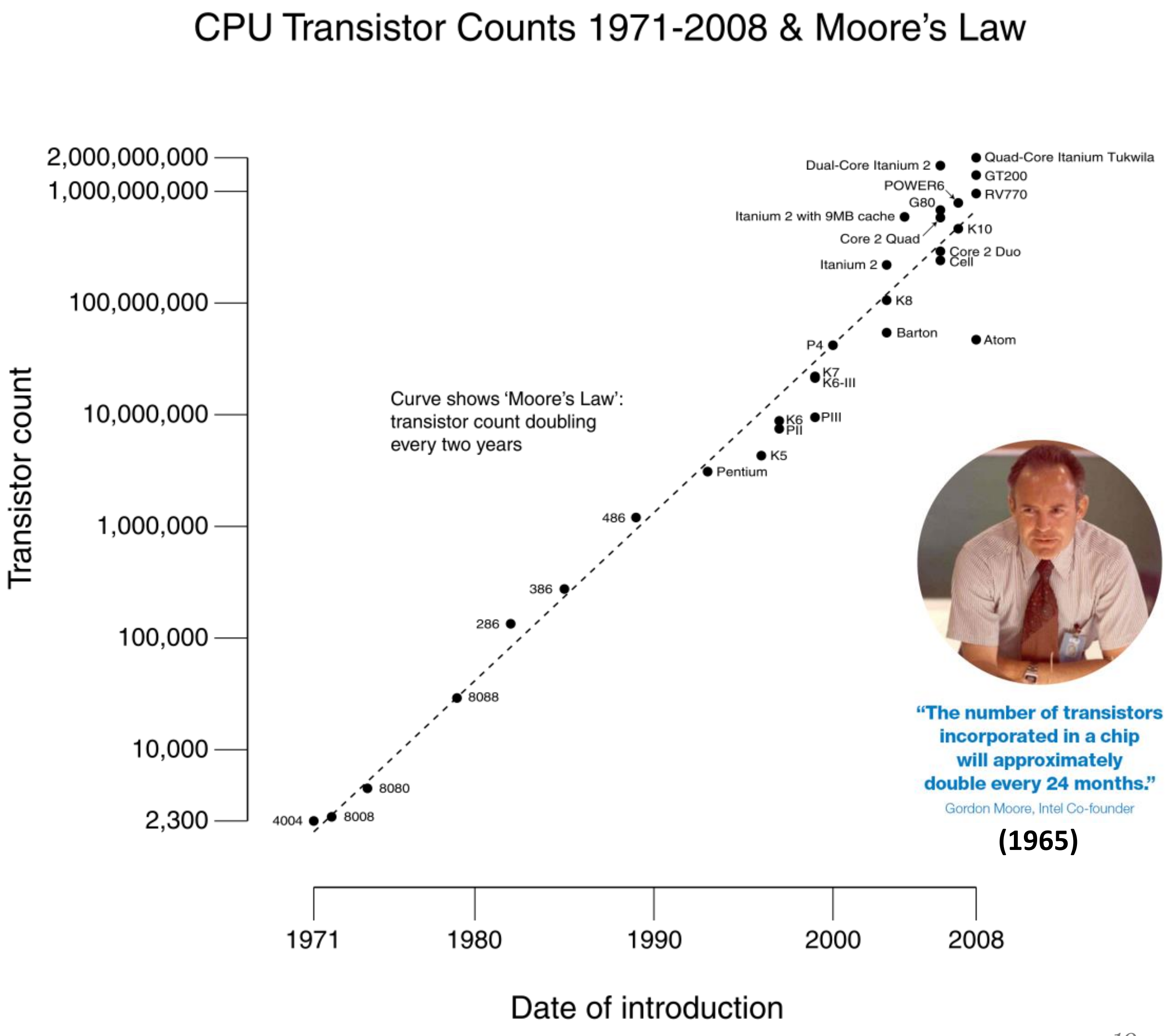
1997	2017
	
ASCI Red at Sandia	Cavium ThunderX2
1.3 TF/s, 850 KW	~ 1.1 TF/s, ~ 0.2 KW

3.5 orders of magnitude

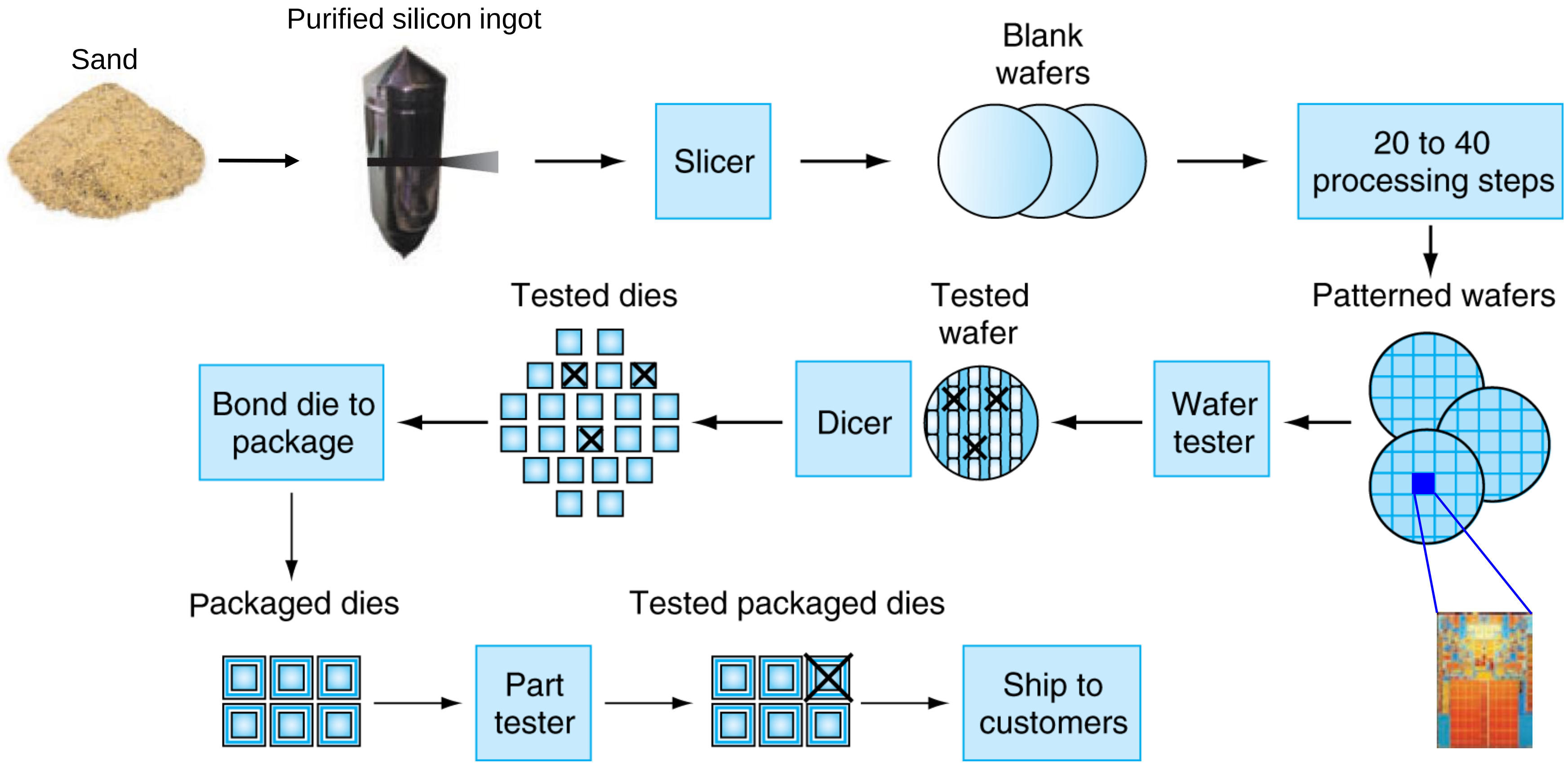
Source: David Keyes, "Algorithmic Adaptations to Extreme Scale Computing," ATPESC, 2018.

The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - World Wide Web (WWW)
  - Smartphones
  - Search engines
  - Human genome project
  - Self-driving cars
  - Artificial intelligence
  - VR/AR
- Computers are pervasive

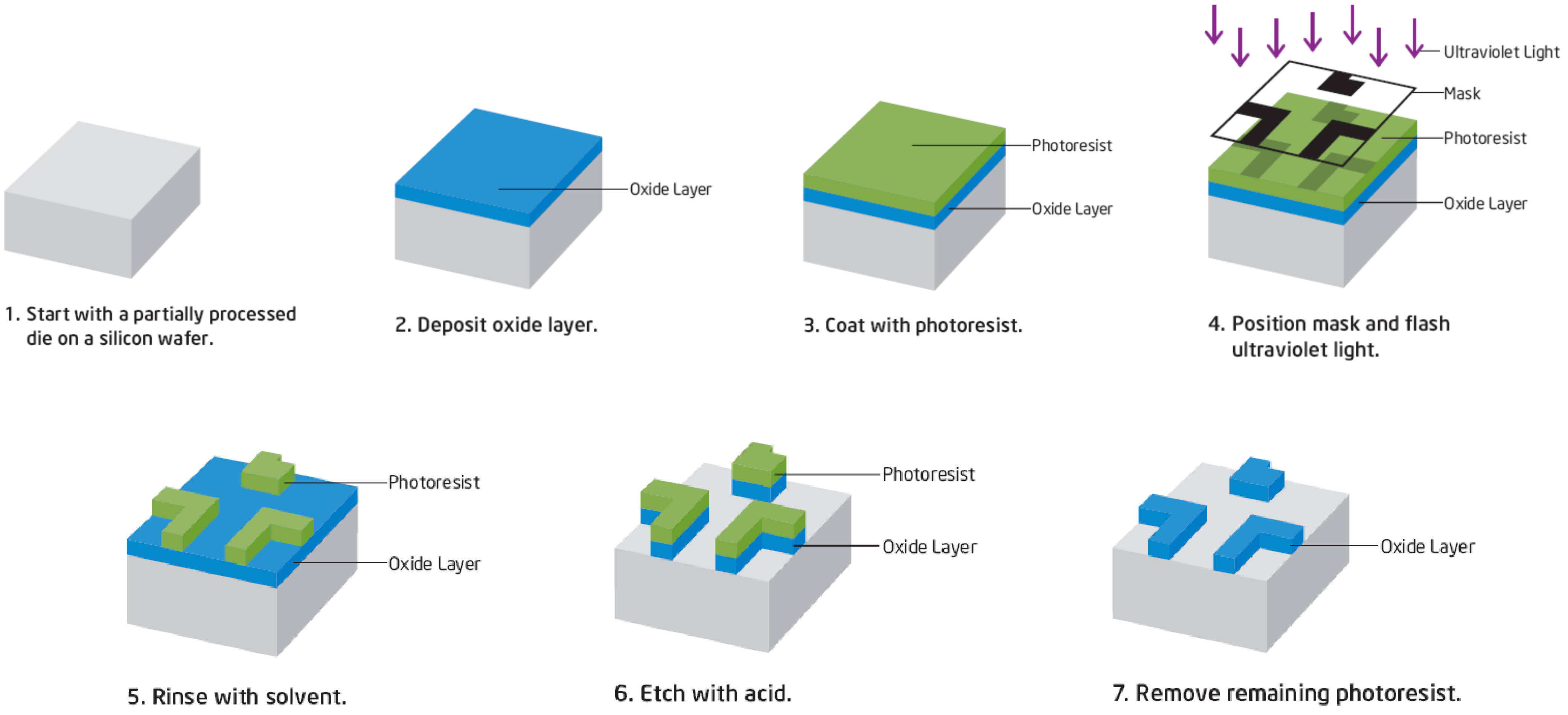


From Sand to Circuits



Yield: proportion of working dies per wafer

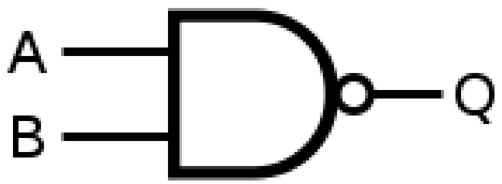
Processing ICs

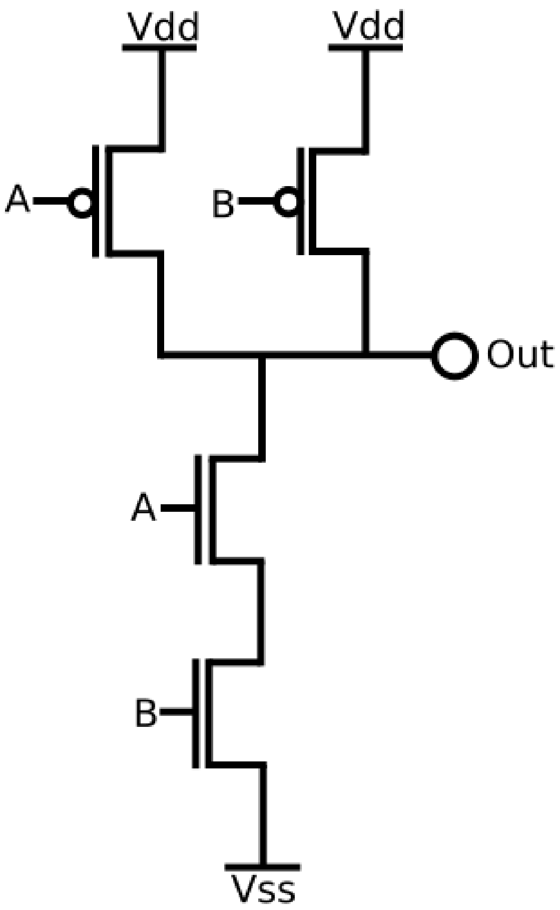


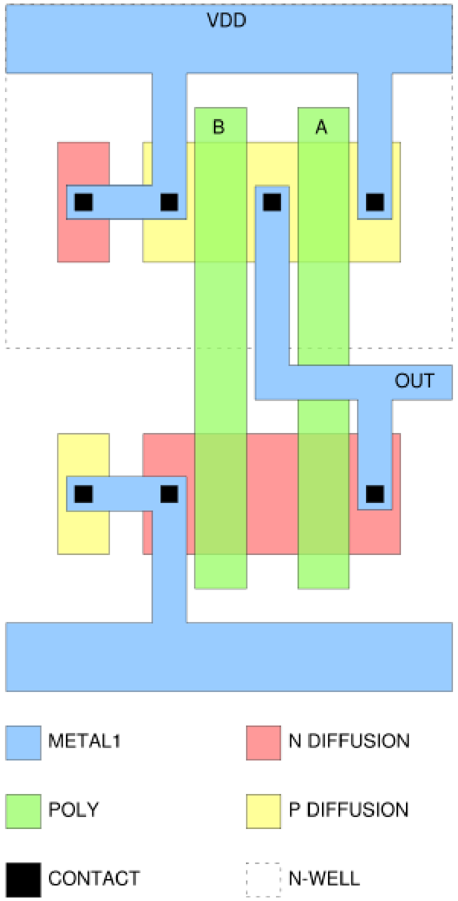
Source: Intel

Realizing a Logic Gate

- NAND logic built with CMOS technology







Source: https://en.wikipedia.org/wiki/NAND_gate

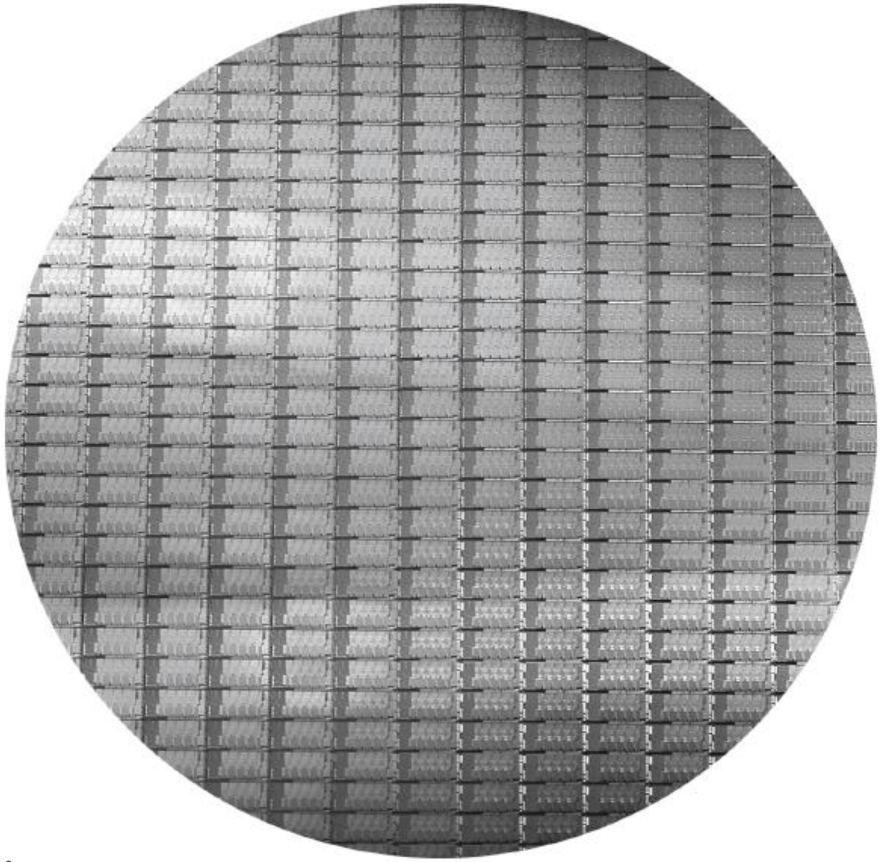
Intel Core i7 Wafer

- 12-inch (300mm) wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm

$$Cost\ per\ die = \frac{Cost\ per\ wafer}{Dies\ per\ wafer \times Yield}$$

$$Dies\ per\ wafer \approx Wafer\ area / Die\ area$$

$$Yield = \frac{1}{\left(1 + \left(Defects\ per\ area \times \frac{Die\ area}{2}\right)\right)^2}$$



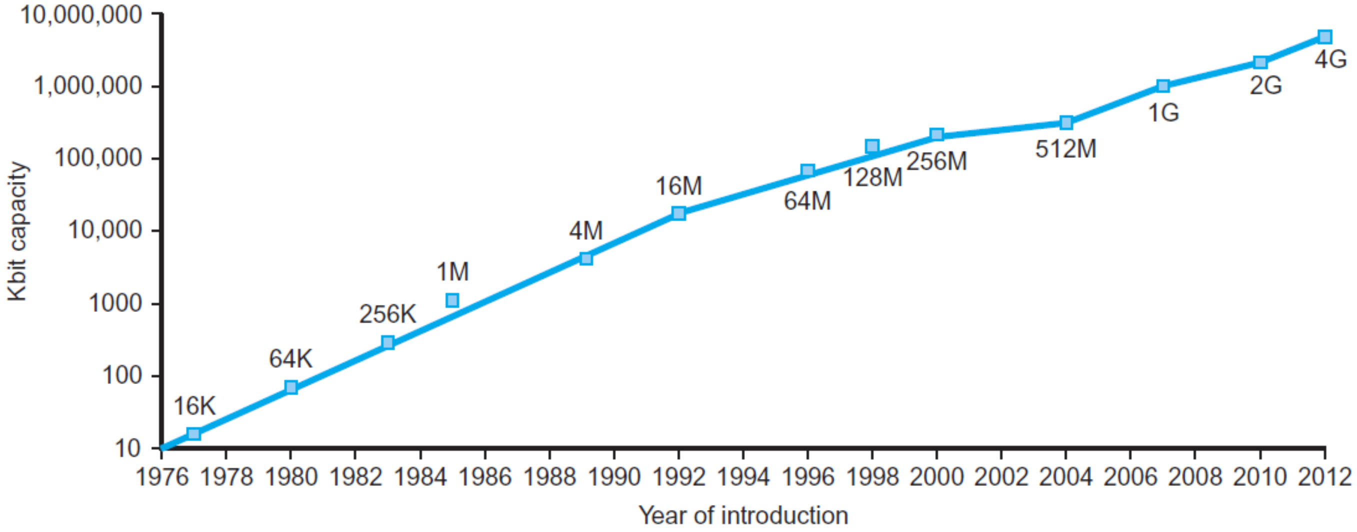
- Wafer cost and area are fixed
- Defect rate determined by manufacturing process
- Die area determined by architecture and circuit design

Technology Trends

- CPU
  - Logic capacity: ~ 30% / year
  - Clock rate: ~ 20% / year

Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

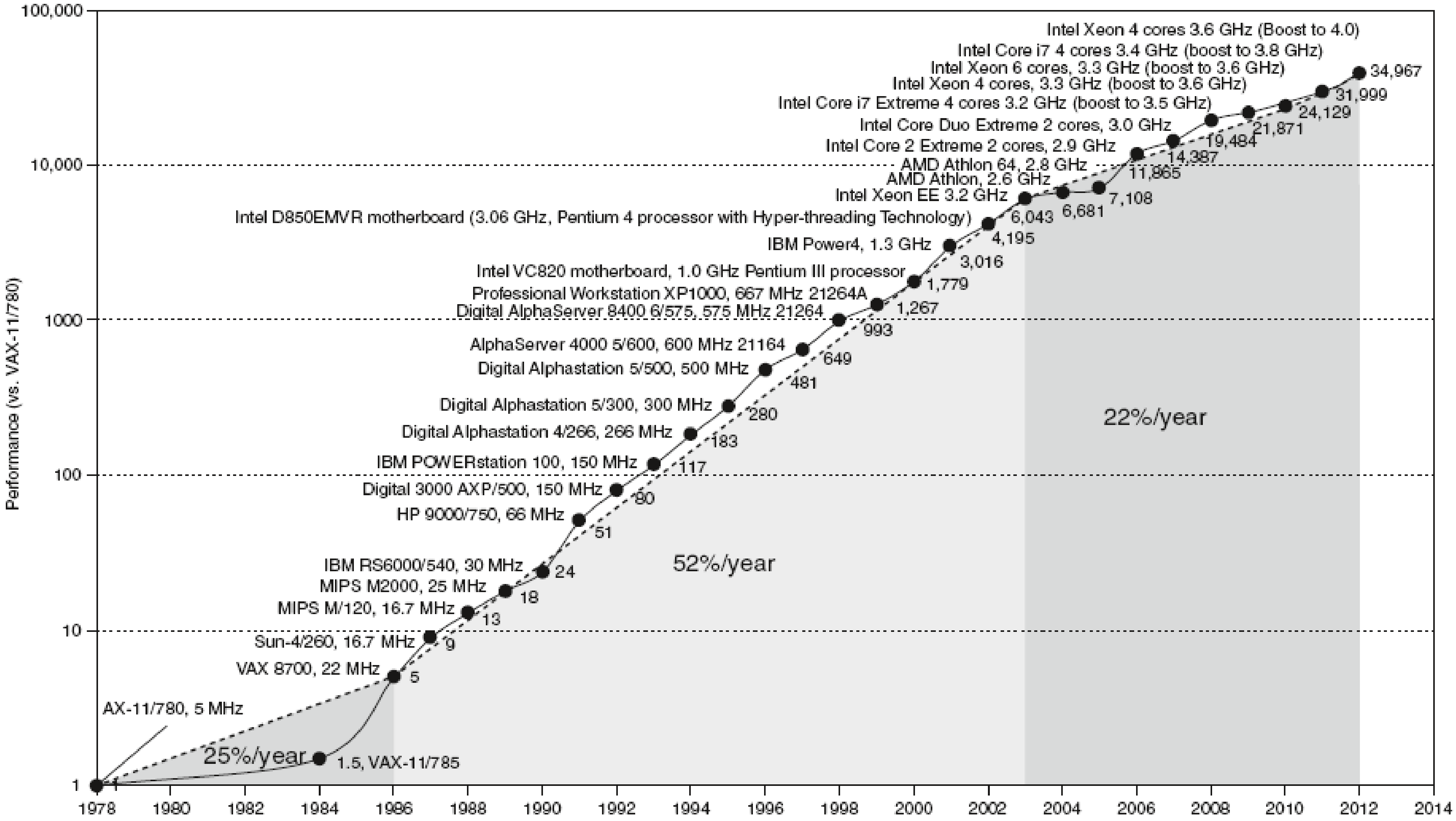
- Memory
  - DRAM capacity: ~ 60% / year (4x every 3 years)
  - DRAM speed: ~ 10% / year



Architecture: Exploiting Parallelism

- Instruction level parallelism (ILP)
  - Pipelining
  - Superscalar
  - Out-of-order execution
  - Branch prediction
  - VLIW (Very Long Instruction Word)
- Data level parallelism (DLP)
  - SIMD / Vector instructions
- Task level parallelism (TLP)
  - Simultaneous multithreading (Hyperthreading)
  - Multicore

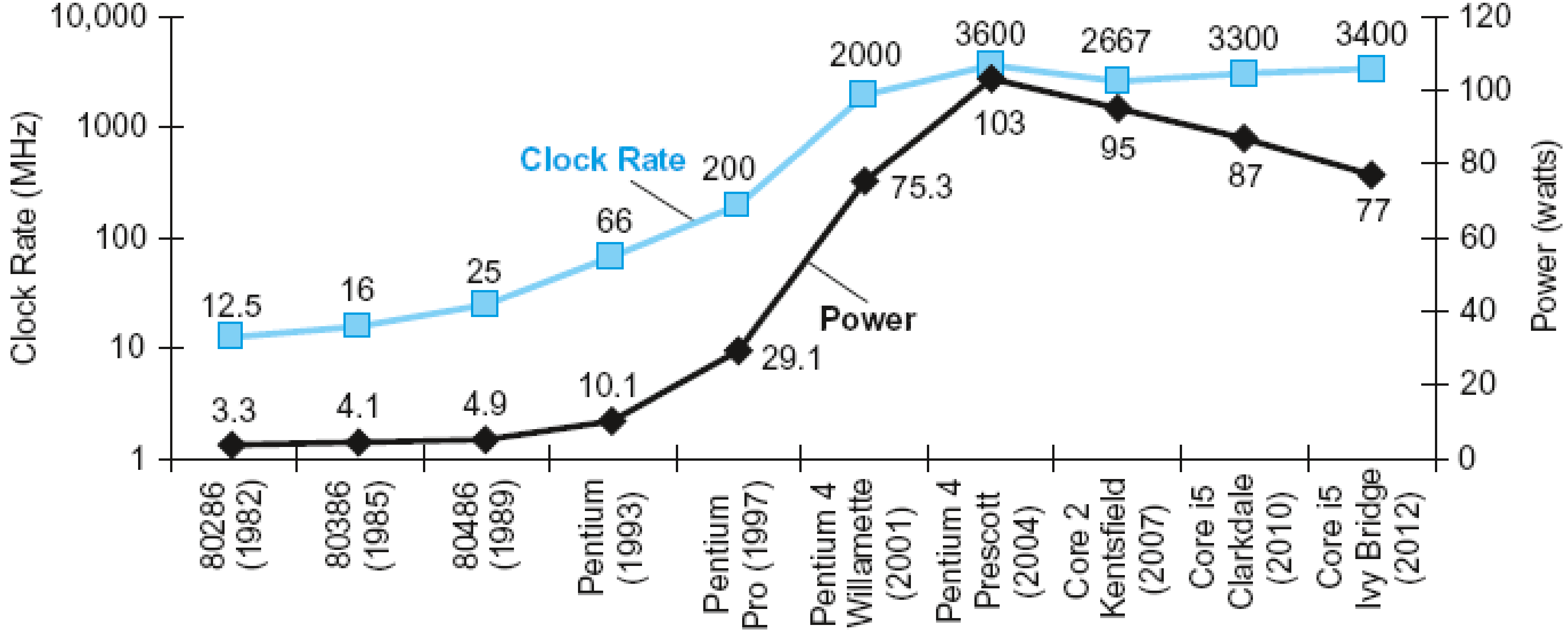
Uniprocessor Performance



Power Trends

- In CMOS IC technology,

$$Power = Capactive\ Load \times Voltage^2 \times Frequency$$



Reducing Power

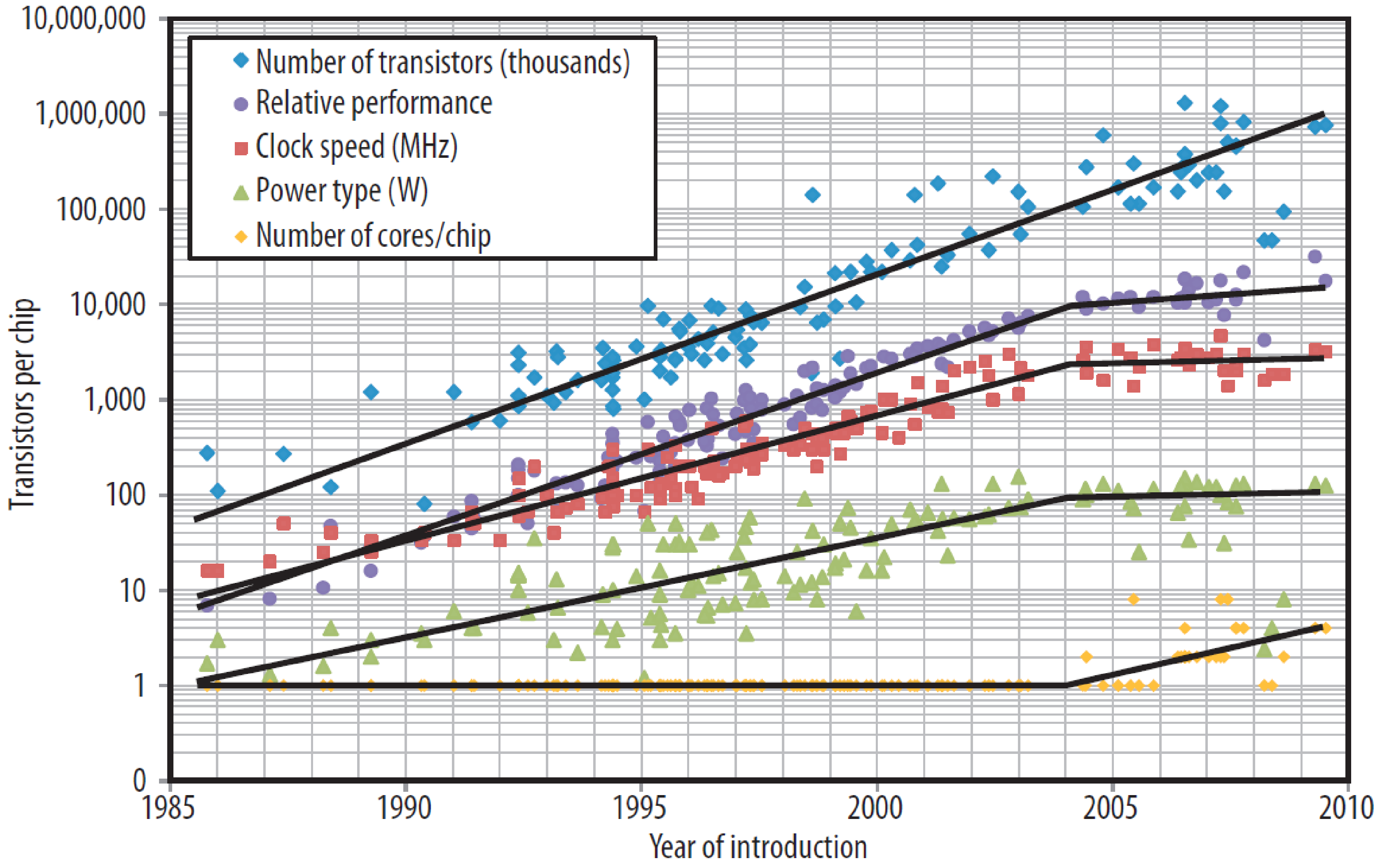
- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times {V_{old}}^2 \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

The Shift to Multicores

- ILP wall
  - Control dependency
  - Data dependency
- Memory wall
  - Memory latency improved by 10% / year
  - Cache shows diminishing returns
- Power wall

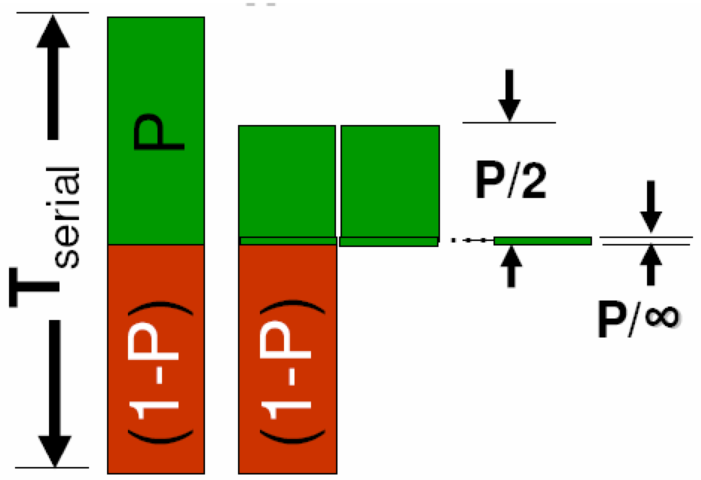


Source: IEEE Computer, 2011

Amdahl's Law

- The theoretical upper limit of speed up is limited by the serial portion of the code

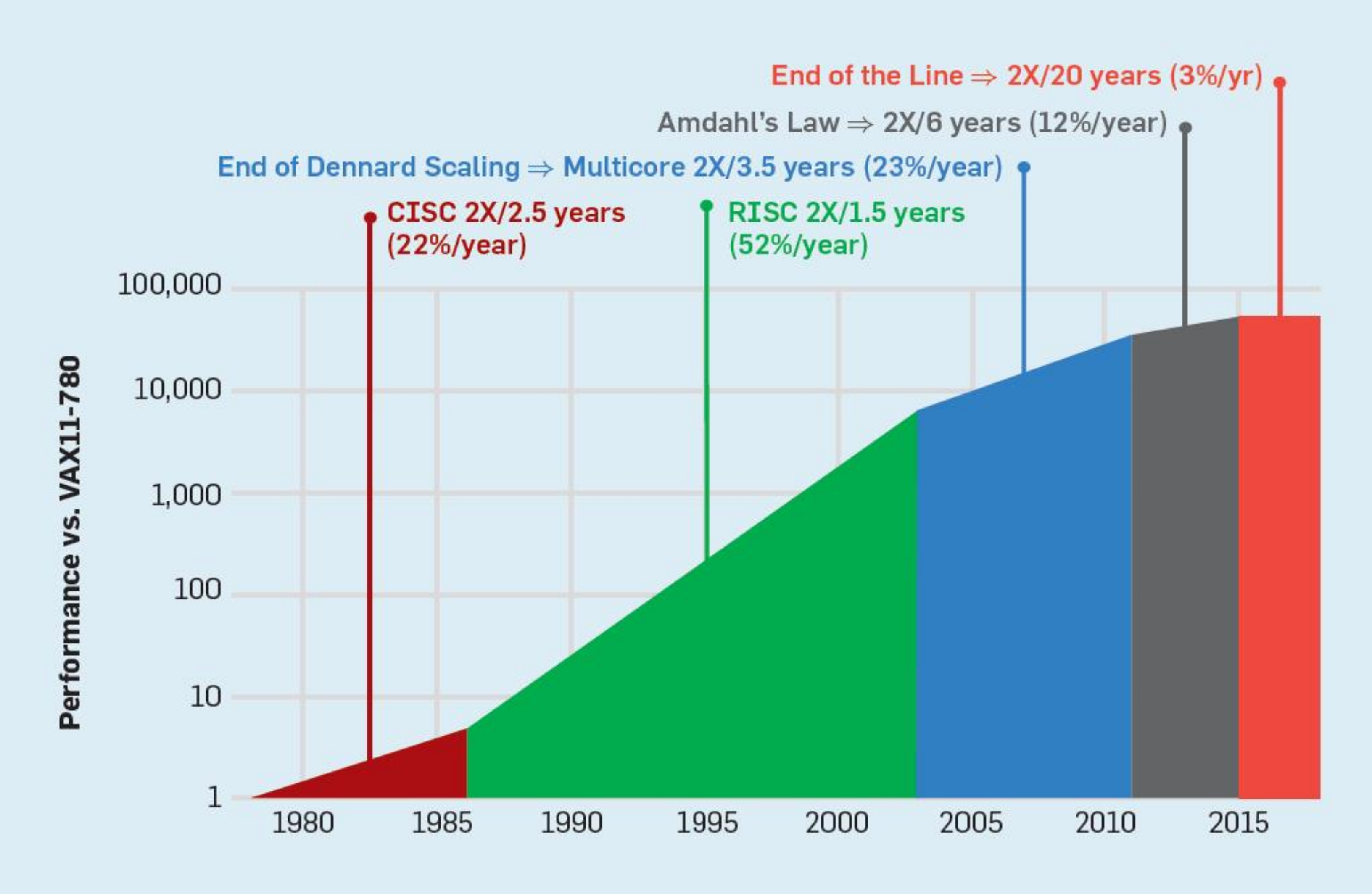
$$Speedup = \frac{1}{(1-P)+\frac{P}{n}}$$



  - S = (1 – P): the time spent executing the serial portion
  - n: the number of processor cores

- Corollary: make the common case fast

A New Golden Age?

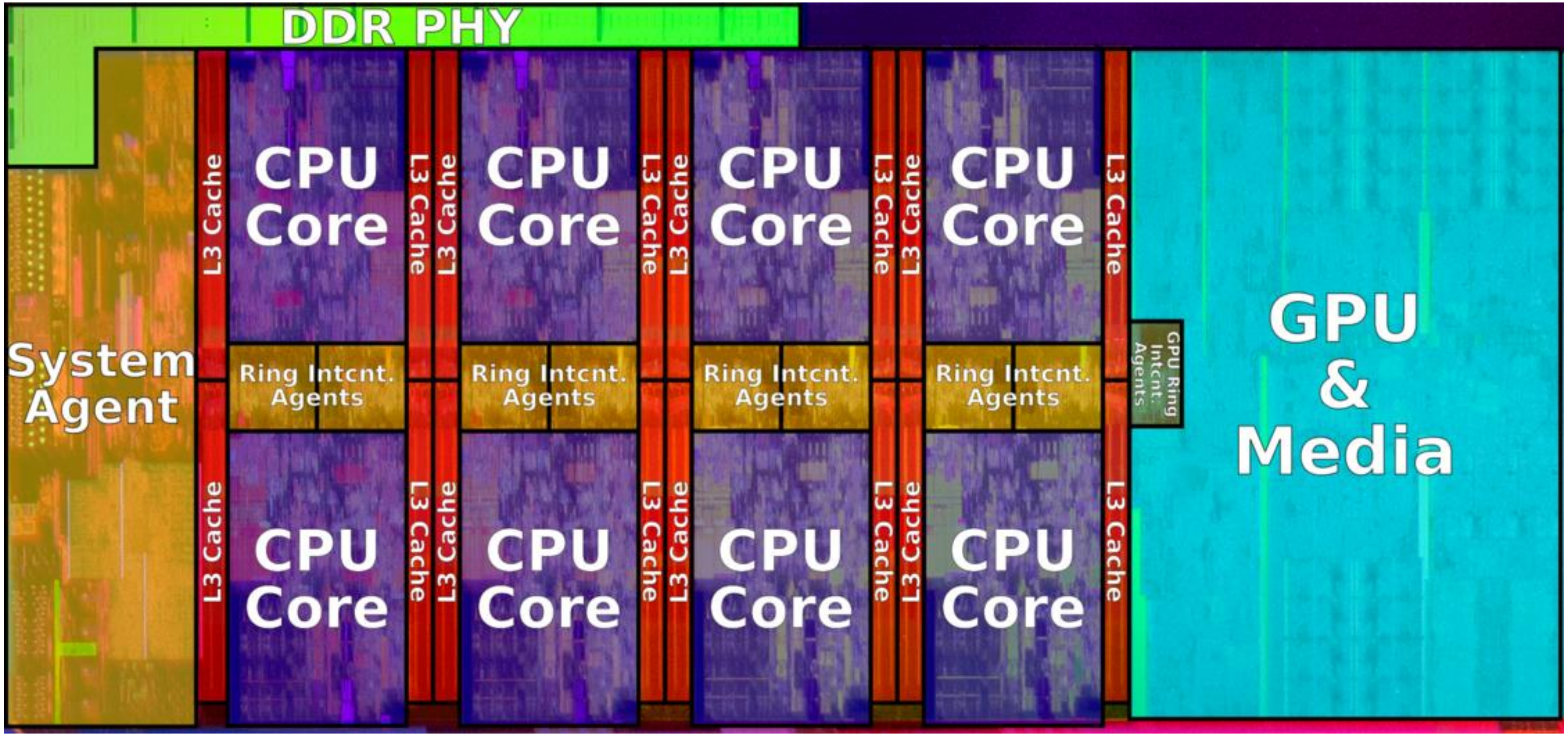


End of Moore's law

A New Golden Age for Computer Architecture (CACM, Feb. 2019)

Source: J. L. Hennessy and D. A. Patterson, "A New Golden Age for Computer Architecture", CACM, 2019.

Intel Core i9-9900K (Coffee Lake, 2018)



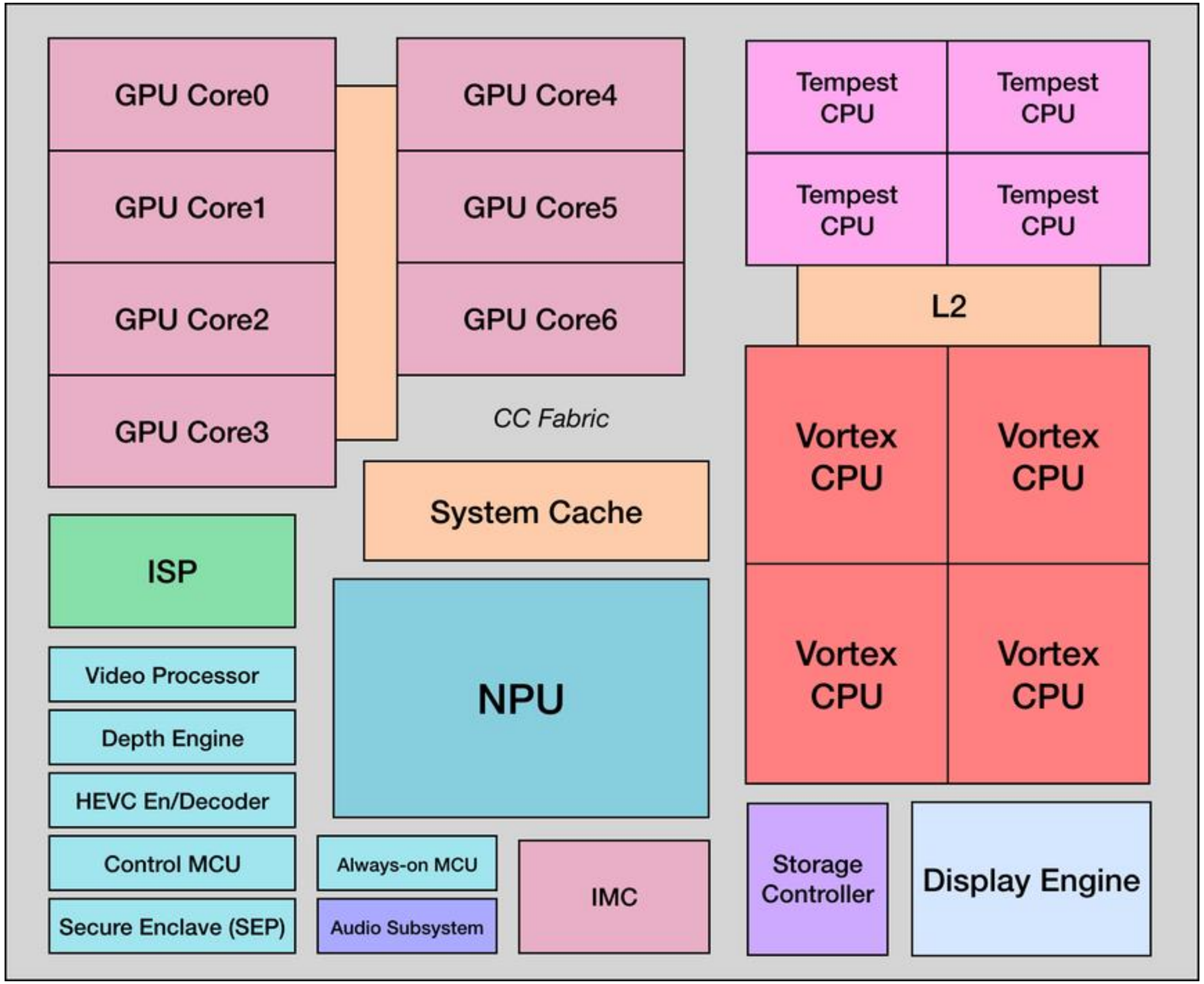
Process:
14nm

Transistors:
~ 3B

Die size:
~ 177 mm^2

Source: https://en.wikichip.org/wiki/intel/core_i9/i9-9900k

Apple A12X Bionic (2018)



Process:
7nm

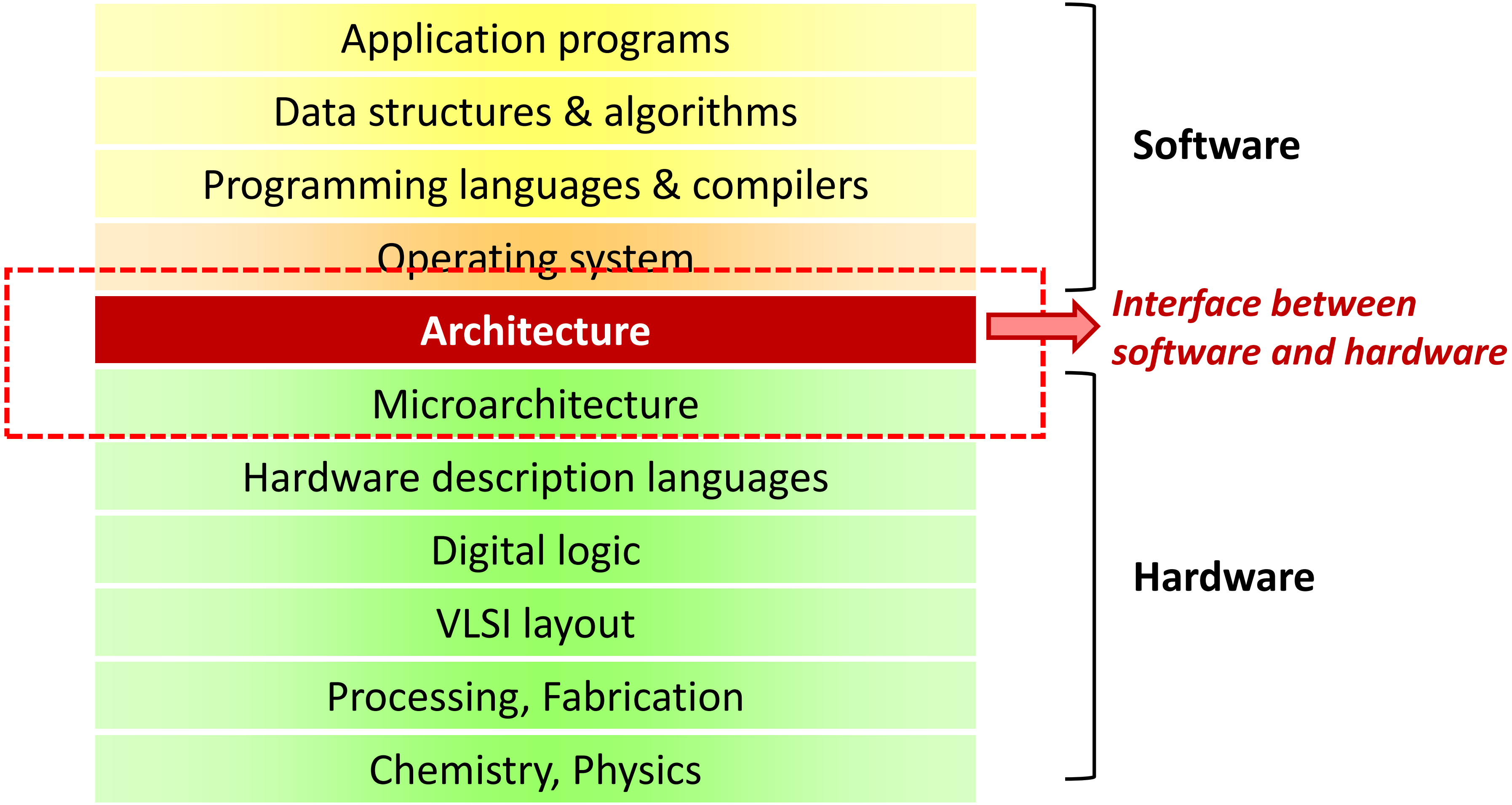
Transistors:
~ 10B

Die size:
~ 122 mm²

Source: https://en.wikichip.org/wiki/apple/ax/a12x

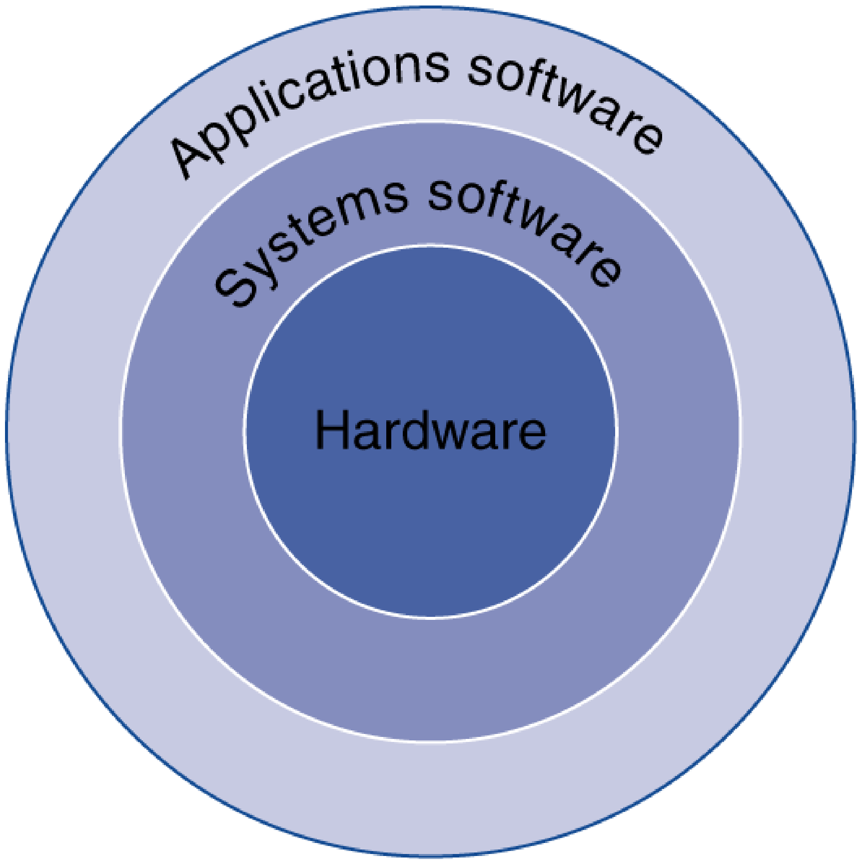
Taming Complexity

- Levels of abstractions



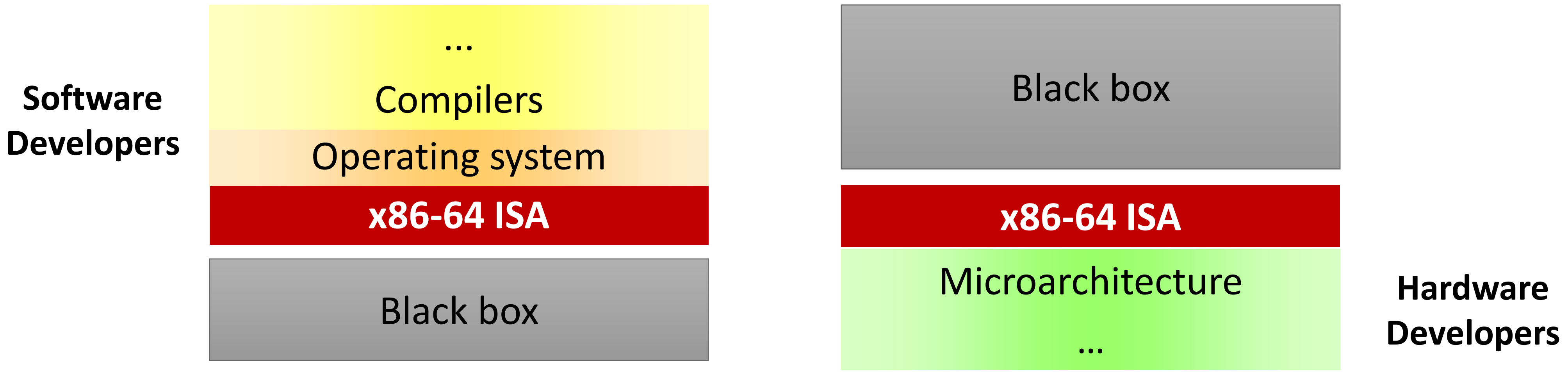
Below Your Program

- **Application software**
  - Written in high-level language
- **System software**
  - Compiler: translates HLL code to machine code
  - Operating system: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- **Hardware**
  - Processor, memory, I/O controllers



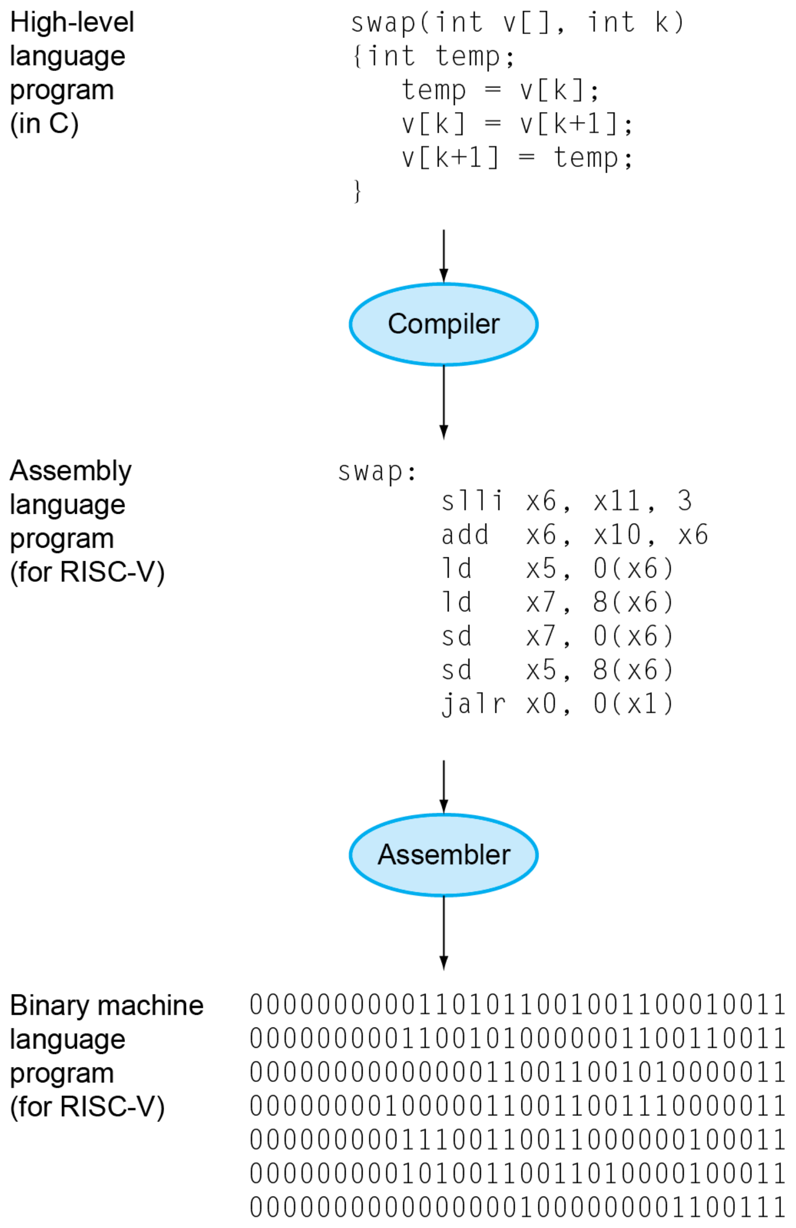
Instruction Set Architecture (ISA)

- The hardware/software interface
  - Hardware abstraction visible to software (OS, compilers, …)
  - Instructions and their encodings, registers, data types, addressing modes, etc.
  - Written documents about how the CPU behaves
  - e.g. All 64-bit Intel CPUs follow the same x86-64 (or Intel 64) ISA



Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
  - For humans
- Machine language
  - Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



Abstraction is Good, But …

- Abstraction helps us deal with complexity
  - Hide lower-level details

- These abstractions have limits
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations

- This is why you should take this course seriously even if you don't want to be a computer architect!

Example #1: Int's ≠ Integers, Float's ≠ Reals

- Is $x^2 \geq 0$?
  - Float's: ??
  - Int's: ??

```
int x = 50000;
printf (“%s\n”, (x*x >= 0)? “Yes” : “No”);
```

- Is (x + y) + z == x + (y + z)?
  - Unsigned & Signed Int's: ??
  - Float's: ??

```
float x = 1e20, y = -1e20, z = 3.14;
printf (“%s\n”, (x+y)+z==x+(y+z)? “Yes” : “No”);
```

Example #2: More Than Just GHz

CPU	Clock Speed	SPECint2000	SPECfp2000
Athlon 64 FX-55	2.6GHz	1854	1782
Pentium 4 Extreme Edition	3.46GHz	1772	1724
Pentium 4 Prescott	3.8GHz	1671	1842
Opteron 150	2.4GHz	1655	1644
Itanium 2 9MB	1.6GHz	1590	2712
Pentium M 755	2.0GHz	1541	1088
POWER5	1.9GHz	1452	2702
SPARC64 V	1.89GHz	1345	1803
Athlon 64 3200+	2.2GHz	1080	1250
Alpha 21264C	1.25GHz	928	1019

Example #3: Constant Factors Matter

- There's more to performance than asymptotic complexity
- Array copy example

```
void copyij (int src[2048][2048],
             int dst[2048][2048])
{
  int i, j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

4.3 ms

```
void copyji (int src[2048][2048],
             int dst[2048][2048])
{
  int i, j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

81.8 ms

`copyji()` is 20x slower on 2.0GHz Intel Core i7 Haswell. Why?

Example #4: Memory Matters

- Memory referencing bug example

```c
/* Echo Line */
void echo()
{
    // Way too small!
    char buf[4];
    gets(buf);
    puts(buf);
}

int main()
{
    printf(“Type: “);
    echo();
    return 0;
}
```

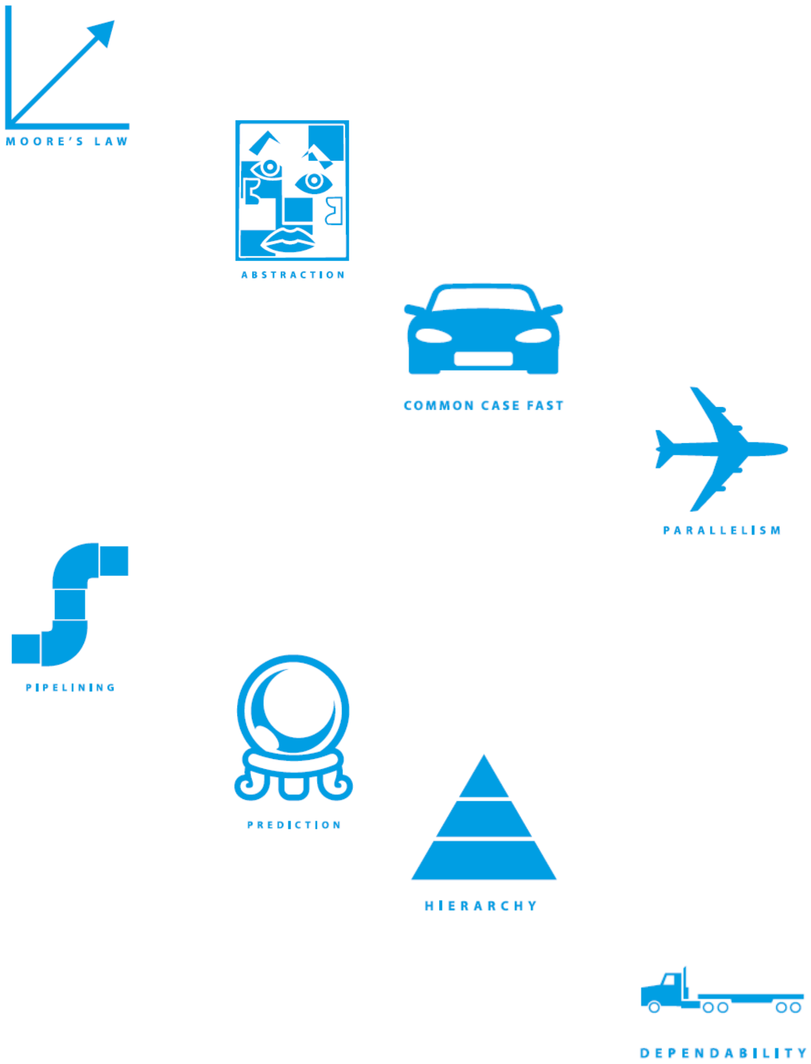
```
$ ./bufdemo
Type:012
012

$ ./bufdemo
Type: 01234567890123456789012
01234567890123456789012

$ ./bufdemo
Type: 012345678901234567890123
Segmentation fault (core dumped)
```

What You Will Learn

- How data are represented?
- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface – Instruction Set Architecture (ISA)
- What determines program performance
- How hardware designers / software developers improve performance
- What is parallel processing

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



Why Take This Course?

- To graduate!
- To design the next great instruction set? Well…
  - ISA has largely converged, especially in desktop / server / laptop / mobile space
  - Dictated by powerful market forces (Intel/ARM)
- To get a job in Intel, NVIDIA, ARM, Apple, Qualcomm, Google, …
  - Tremendous organizational innovations relative to established ISA abstractions
- Design, analysis, and implementation concepts that you'll learn are vital to all aspects of computer science and engineering
- This course will equip you with an intellectual toolbox for dealing with a host of systems design challenges
- And finally, just for fun!

Partially borrowed from David Culler's CS252 lecture slides

Summary

- Modern Computer Architecture is about managing and optimizing across several levels of abstraction w.r.t. dramatically changing technology and application load
- This course focuses on
  - RISC-V Instruction Set Architecture (ISA) – A new open interface
  - An implementation based on Pipelining (Microarchitecture) – how to make it faster?
  - Memory hierarchy – How to make trade-offs between performance and cost?
- Understanding Computer Architecture is vital to other "systems" courses:
  - System programming, Operating systems, Compilers, Embedded systems, Computer networks, Multicore computing, Distributed systems, Mobile computing, Security, Machine learning, Quantum computing, etc.