CSE 3421:
Introduction to Computer Architecture

Xiaodong Zhang

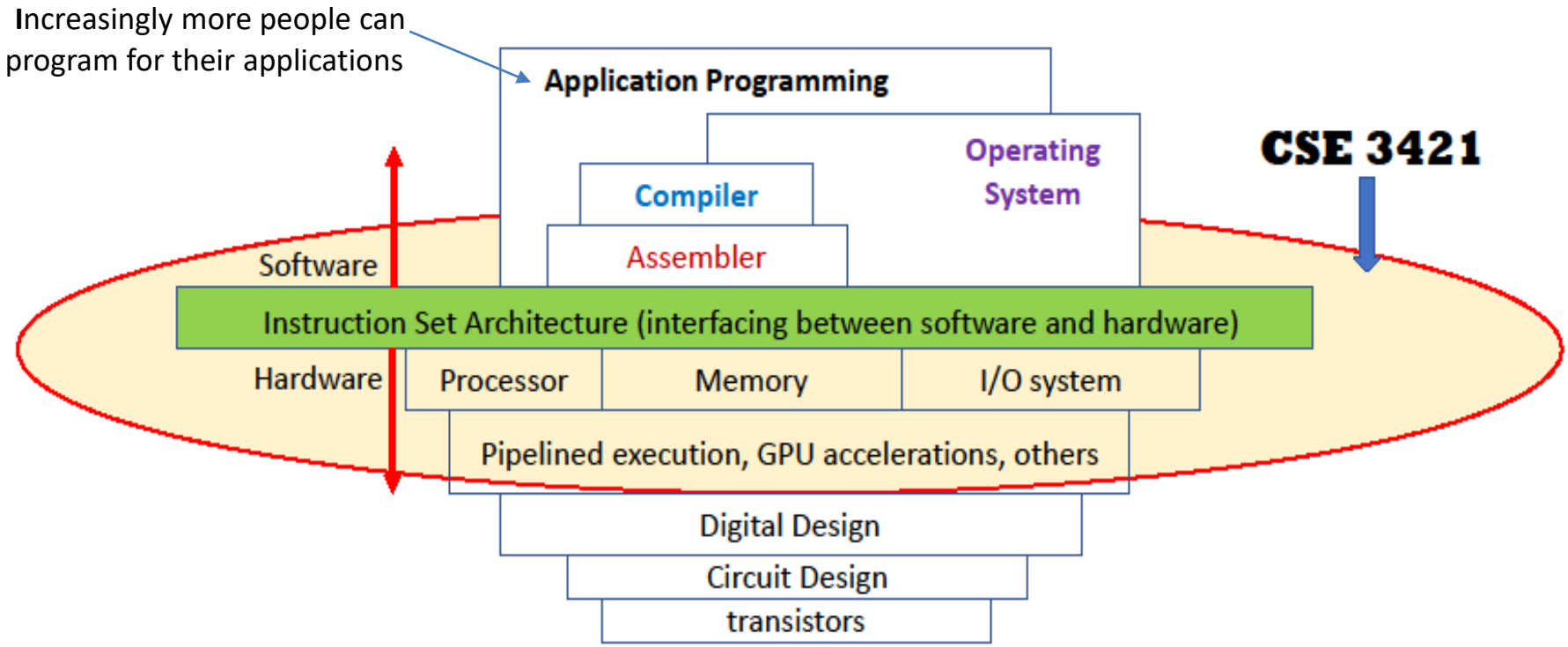
Administrivia

- All course communication through Canvas
  - Homework will be posted in Canvas but due in hard copy, in class – late homework is not accepted
  - Lectures will be delivered in Hitchcock Hall 035
  - A CSE3421 Piazza site will be created for questions and discussions
- Major References:
  - *"Computer Organization & Design: The Hardware/Software Interface, Fourth Edition", D.A. Patterson & J.L. Hennessy, 2009 by Elsevier Inc. (2019 version is also useful)*
  - *Lecture Slides (will be available online)*
  - *Additional Notes for the class*
- TA by a Ph.D. student
  - Saumya Sahai

Why learn Computer Architecture?

- CA is one of the most fundamental classes of CS
  - Every medical school student must study Anatomy
  - CA is the “Anatomy” of CS
- Computer system architects in both hardware and software are based on the “Anatomy”
- Writing efficient software must be CA-aware
- After programming for some years, it is the right time to gain more insights into the “box”.
- This is the class for you
  - to be a qualified CS major
  - to become a proficient software engineer
  - to become a system architect

Where is CSE 3421 in Computing Ecosystem?



- We cover three spaces:
  - Virtual space: Assembly code
  - Physical memory space: CPU, pipeline, cache, and memory system
  - Storage space: virtual memory, page table, file systems, hard disks and SSD

To be insightful into key questions after the class

- After turning on a computer
  - What are the basic operations to be performed inside the “box”
  - What happened to CPU? Cache? DRAM? and Disk?
- When you use “word software” to write your homework
  - what happened to CPU? Cache? DRAM and Disk?
- When you compile your program
  - what happened to CPU? Cache? DRAM and Disk?
- When you execute your program
  - what happened to CPU? Cache? DRAM and Disk?
- How is a program executed in hardware?
- How are data blocks moving around inside the “box”?
- …..

Opportunity Cost

- What cost to pay?
  - When one has an alternative over another one in opportunities, to choose one, she must pay the cost for the other one
  - To not attend the class for the opportunity of sleeping, doing something else, you may have to pay the cost of missing classes
- We do active learning in the class
  - It is also called “Flipped” classroom, where learning experiences come from interactions and discussions
  - We are not quite a “flipped” classroom, because I mainly give lectures, but I initiate many discussions
- For the best interests of yours, participation is 5% grade
  - Some of the discussions will be included in exams
  - Raising and answering questions, and participating discussions are best practice

About me

- At Ohio State since 2006
- I was the Department Chair from 2006-2018.
- My Research is on Data Management in Computer Systems
  - Architecture, OS, parallel processing, and databases
- Architecture work focuses on memory/storage systems
- Three results of mine are in general-purpose processors
  - Multi-column cache (1997): a fast and low miss rate cache
  - Permutation interleaving (2000) is adopted in all memory controllers
  - Cache management in multicores (2008) in many existing systems

The evolution of computing R&D in three stages

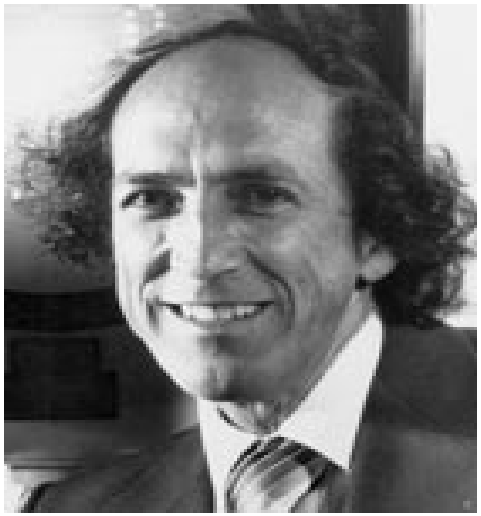
- R&D of computers is for computing (1930s -1990s)
  - 1936-1937 ABC in Iowa State, Zues in Berlin (electronic tube)
  - 1945, ENIAC in U. Penn (electronic tube, judge declared ENIAC patent invalid, 10/19/1973)
  - 1950, Transistors from Bell Labs
  - 1958-1962, CMOS integrated circuits from Fairchild and RCA
  - 1965, Moore's Law, 1971 first CPU chip (Intel 4004)
  - 1960s-1990s, OS, compilers, databases, HPC, ….
  - Turning physical world into digital world by simulation for deep analysis
  - Many great breakthroughs to advance the society: climate, material, automobile, airline, …
- R&D of computers is for connectivity (1990s – 2010s)
  - 1981-2017: **Bandwidth:** from 50K bps to 100P bps (2 trillion times (10^{12}))
  - 1981-2017: # of **devices/user:** from 0.1 to 10 (100 times)
  - Internet phone, shopping, searching, facebook, twiter, YouTube, Zoom …
- R&D of computers is for data (starting from this century)
  - The data explosion today is not an extension from the physical world
  - The new digital world accurately trace and record our own behavior
  - 90% of total data in the world is generated in last two years by us

Ending the Era of Moore's and Dennard Scaling Laws

"The number of transistors incorporated in a chip will approximately double every 24 month".

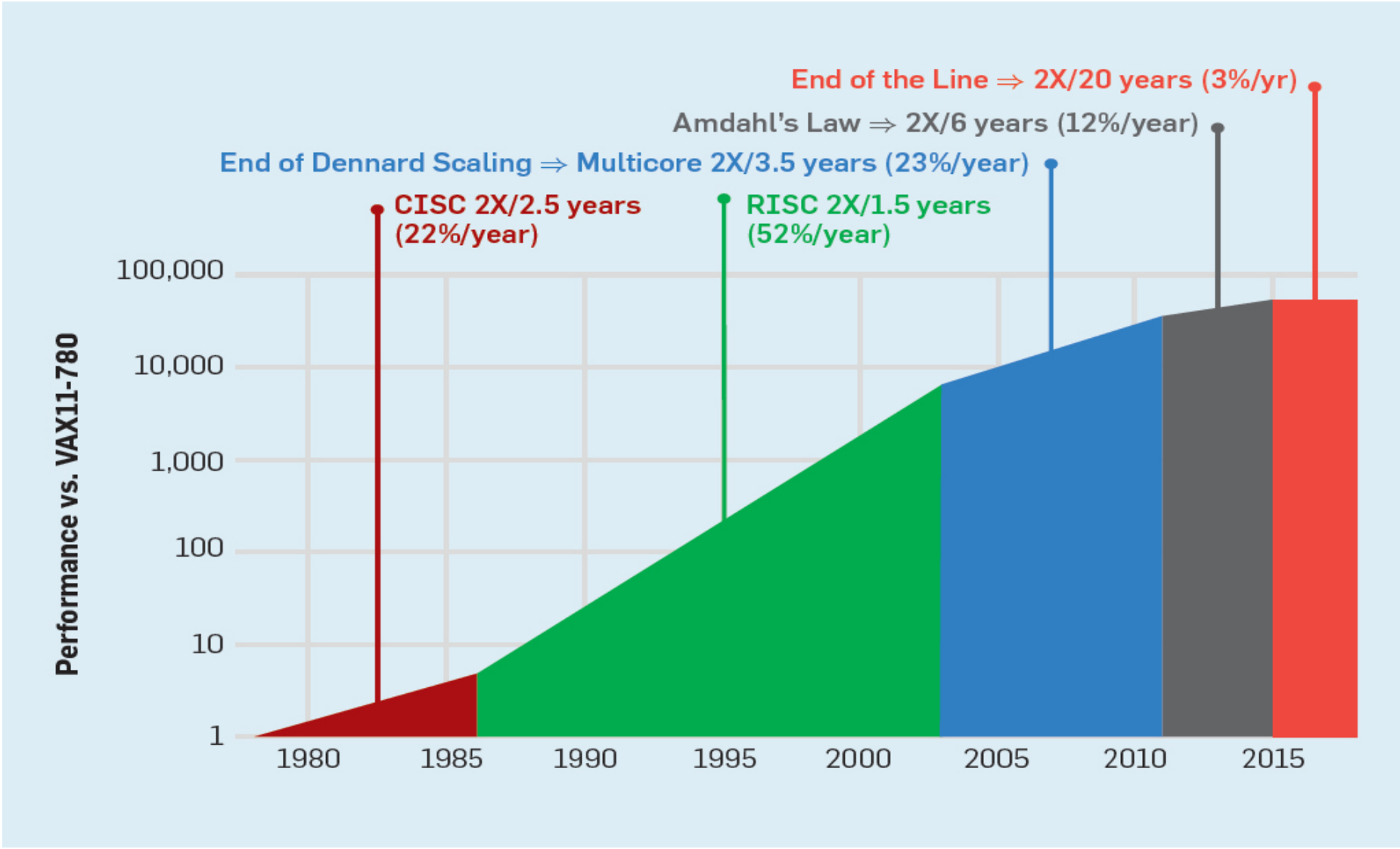
Gordon Moore, 1978 (based on his observation in 1965)

"We can keep power consumption constant as the number of transistors increases in a chip".



Robert Dennard, 1974 (this scaling ended in 2005)

Chip Speed Growth Driven by Moore's and Dennard Scaling



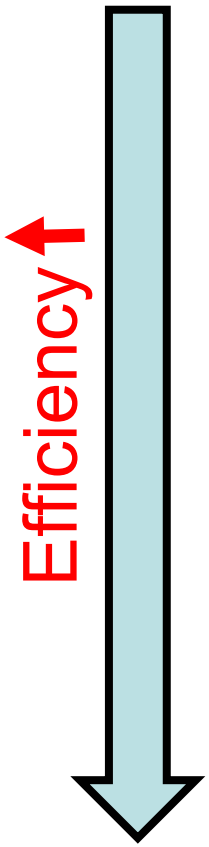
We have built a powerful and general-purpose ecosystem

- **A simple and one-size-fit-all computing abstraction**
  - Any computing task is expressed by programs and executed by ISA
  - A task is divided by a sequence of standard execution time unit
  - Under multiprogramming model, OS assigns time units to each task
  - Data blocks are moved up and down in the memory hierarchy
  - Programming is simple-English-based and architecture/OS independent
  - Billions of users rely on this abstraction (CS is the center of all)

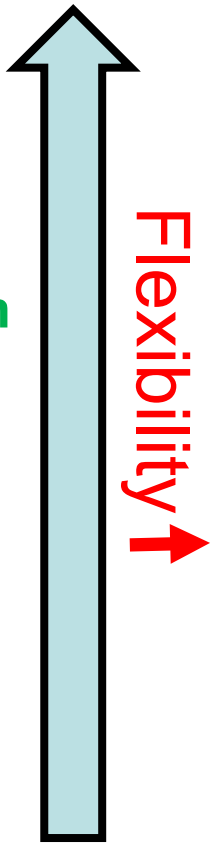
- **Moore's law has hidden dark side of the ecosystem**
  - Efficiency continues to become low in both power and execution
  - Non-inclusive to others, e.g. SIMD, data flow, and domain specific

- **As Moore's law is ending**
  - The one-size-fit-all ecosystem must be restructured for continued high performance computing and high throughput data processing

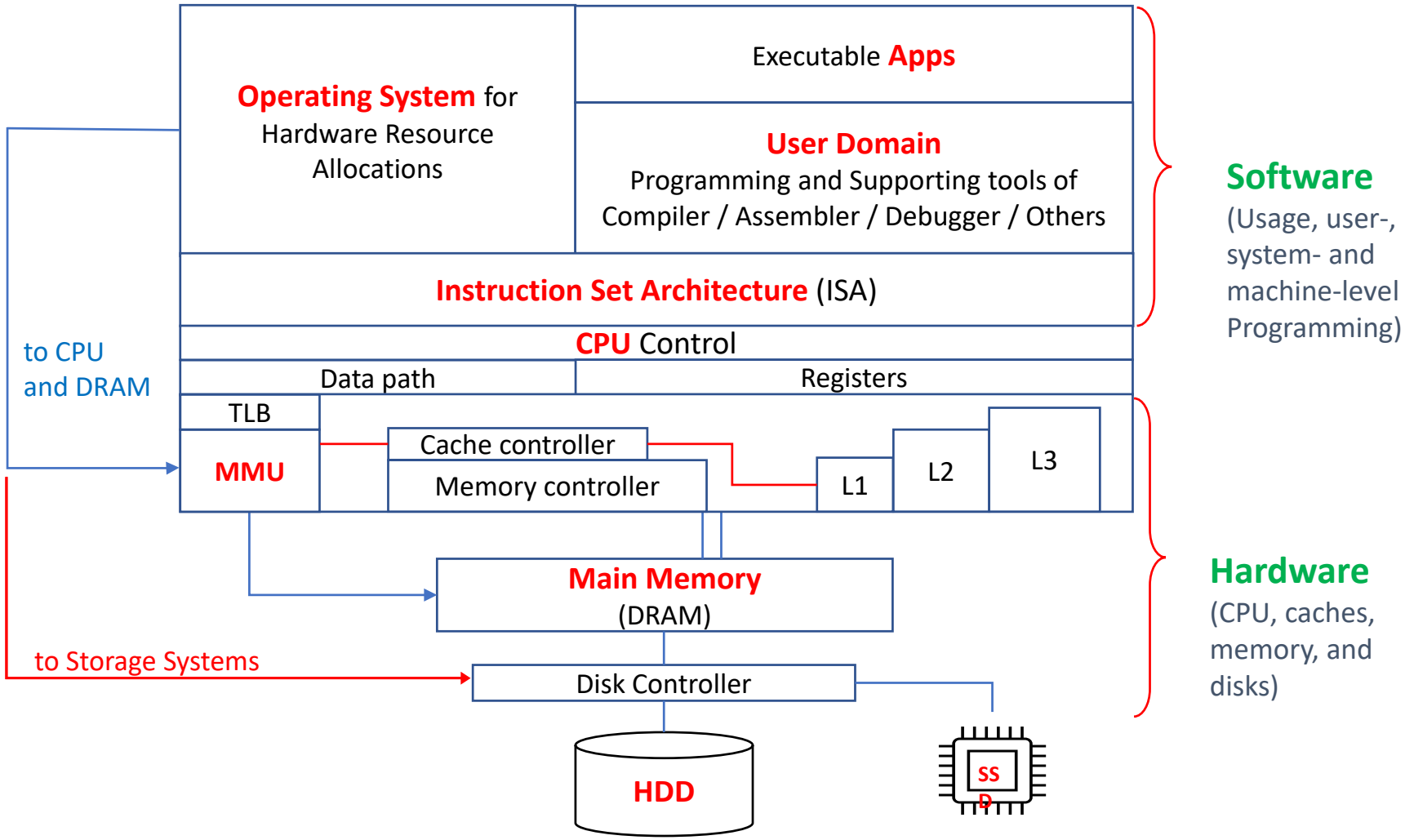
General-purpose Computing is Flexible but not Efficient
(a conflict between fairness and efficiency)



- CPU and Multicores
  - Mainstream environment: **very flexible** to all apps
  - **Power/performance efficiency are low**
- GPUs (graphic processing unit)
  - **Efficient for applications with high SIMT parallelism**
  - Special programming in **low flexibility**
- FPGAs (field programmable gate arrays)
  - **Efficient for applications defined by the hardware**
  - Very special programming in **very low flexibility**
- ASICs (application specific integrated circuits)
  - **Very efficient for applications by customized chips**
  - Expert programing with **no flexibility**

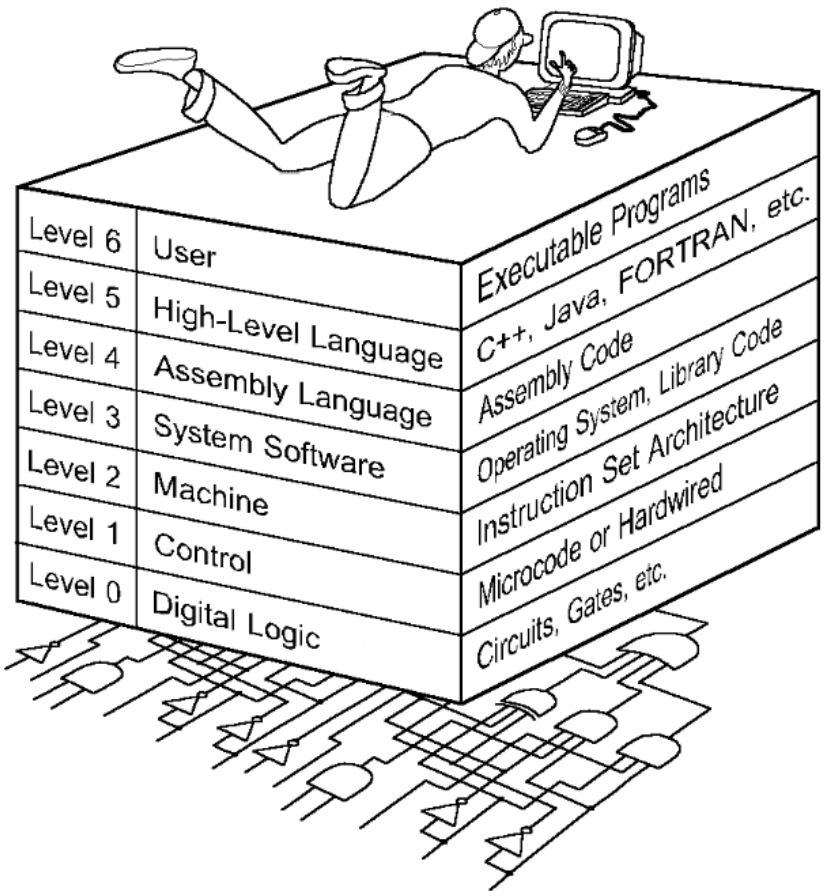


The Entire Hardware-Software Stack (where is your expertise?)



A deep hierarchy from users to the circuit

- Each virtual machine layer is an abstraction of the level below it.
- The machines at each level execute their own particular instructions, calling upon machines at lower levels to perform tasks as required.
- Computer circuits ultimately carry out the work.
- This class covers levels 1, 2, and 3, plus some 4.



The user and language levels

- Level 6: The User Level
  - Program execution and user interface level.
  - E.g. Apps in your iPhone and laptops.
- Level 5: High-Level Language Level
  - The level with which we interact when we write programs in languages such as C, Pascal, Lisp, and Java.
  - A starting point for CSE majors, and are soon required for all majors, like calculus and English

Assembly code and system primitives

- Level 4: Assembly Language Level
  - Acts upon assembly language produced from Level 5, as well as instructions programmed directly at this level. (generated from compiler or interpreter, or programed in the format of ISA)
- Level 3: System Software Level
  - Controls executing processes on the system.
  - Protects system resources.
  - Accesses to physical memory space.

Architecture Level

- Level 2: Machine Level
  - Also known as the Instruction Set Architecture (ISA) Level, informing the machine to perform basic operations. An assembly program is based on ISA (An English article is based on basic words and grammars)
  - Consists of instructions that are particular to the architecture of the machine.
  - Programs written in machine language need no compilers, interpreters, or assemblers.

Hardware Control Level

- Level 1: Control Level
  - A *control unit* decodes and executes instructions and moves data through the system.
  - Control units can be *microprogrammed* or *hardwired*.
  - A microprogram is a program written in a low-level language that is implemented by the hardware.
  - Hardwired control units consist of hardware that directly executes machine instructions.

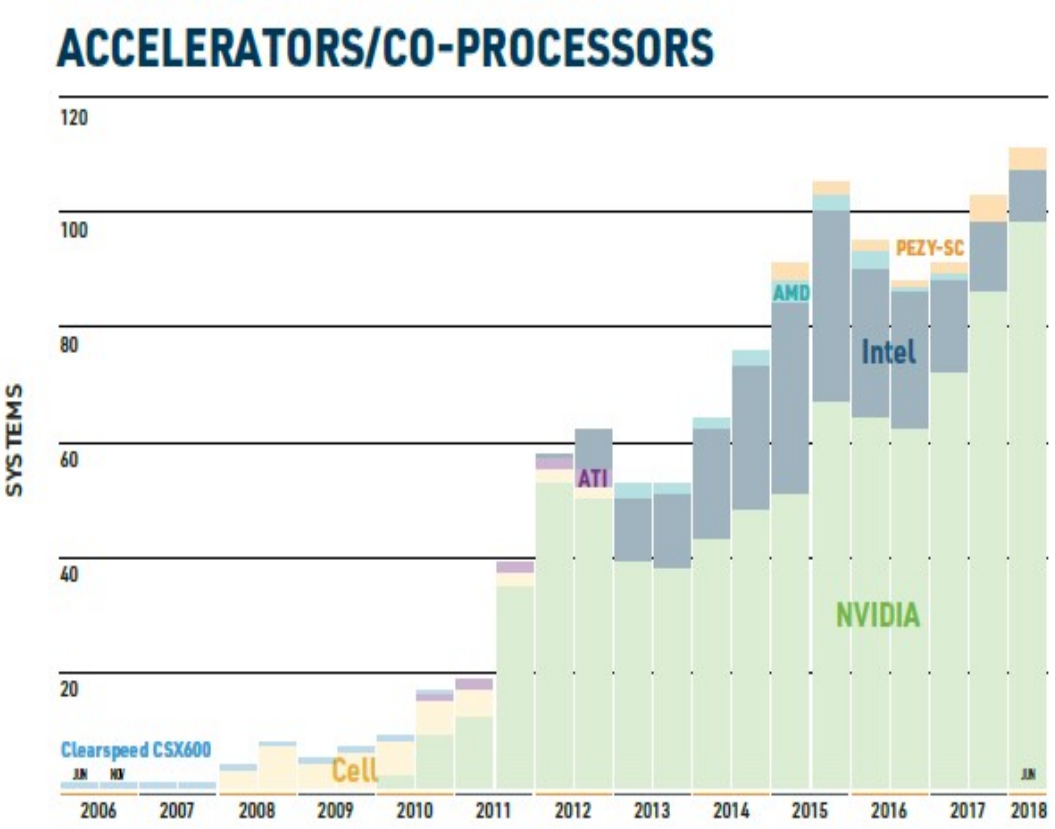
Circuit Level

- Level 0: Digital Logic Level
  - This level is where we find digital circuits (the chips).
  - Digital circuits consist of gates and wires.
  - These components implement the mathematical logic of all other levels.
- This class covers levels 1, 2, and 3, plus some 4.

Donald Knuth: 1974 Turing Award Winner

"One of the main characteristics of a computer science mentality is the ability to jump very quickly between levels of abstraction, between a low level and a high level, almost unconsciously".

Hardware-independent algorithms may be inefficient



Number of hardware accelerators in Top 500 Supercomputers in 2018

Courtesy of The Next Platform

- The complexity of an algorithm is in ***O*** notation only for # computing ops, and moving data is free
- CPU cycles are cheap, but data movement is the bottleneck of computing
- Many algorithms have to be architecture-aware
  - for memory hierarchy
  - for parallelisms
  - for hardware acceleration

Algorithm Design beyond Computing Complexity

- Three pillars in algorithms design
  - Low computing complexity (a low count of comp operations)
  - High data locality (low data movement)
  - High parallelisms (high % of independent operations)

- Best algorithms have perfect three pillars
  - Only a small group is implementation independent
  - Hardware/system/algorithms co-design for the three pillars

- What happens if we only get one or two pillars
  - We may have challenges and uncertainty in performance

Three Pillars of Algorithms Design and Implementation

Notes	Parallelism	Data Locality and Movement	Complexity
Highly **Inefficient** Algorithms	0	0	0
One positive pillar is not sufficient: algorithms are still **inefficient**	0	0	1
	0	1	0
	1	0	0
Efficient algorithm execution is subject to **low data movement**	1	0	1
Highly efficient sequential algorithm execution	0	1	1
Efficient algorithm execution is subject to **sufficiently high parallelism** and **low data movement**	1	1	0
Ideal and perfect sequential and parallel algorithm execution	1	1	1

Classes of computers for different applications

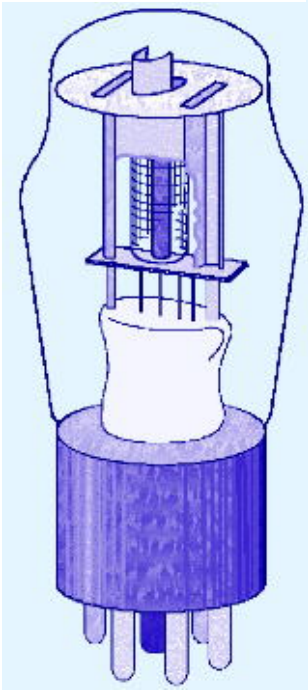
- Personal Mobile Device (PMD)
  - e.g. smart phones, tablet computers, laptops
  - Emphasis on energy efficiency and real-time
- Desktop Computing
  - Emphasis on price-performance
- Servers
  - Emphasis on availability, scalability, throughput
- Clusters / Warehouse Scale Computers
  - Used for “Software as a Service (SaaS)”
  - Emphasis on availability and price-performance
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast interconnection networks
- Embedded Computers (e.g. in cameras, GPS, e-readers, cars, …)
  - Emphasis: light and price
- Wearable computers (e.g. watch, glasses, …)

Computer Ecosystems

- Ecosystems
  - Human: a livable and imperfect environment consisting of human interacting with their dependent natural resources: air, water, soil, plants, weather, other living things
  - Computer: a hardware/software environment supporting for large volumes of users
- The birth and death in ecosystems
  - Survival of the fittest (Darwin, 1809-1882)
  - Rarely have revolution, but evolution
- Survive in hardware ecosystems (performance, functions and price based)
  - (1) supercomputers, increase price/performance, not judged by market
  - (2) general-purpose computers: increase performance, constant prices in market
  - (3) smart devices: increasing functions, decreasing prices and sizes, i-X, …
- Survive in software ecosystems (user group size based)
  - (1) Infrastructure software: windows, Linux, BSD, …
  - (2) basic application software: MySQL, Oracle, DB2, Hadoop, Hive ….
  - (3) Internet service software: Google, Facebook, Youtube, Zoom, ….
- The computer ecosystem is a global market
  - Prices, application scopes, and critical mass of users determine the survivals

First Computer in the World

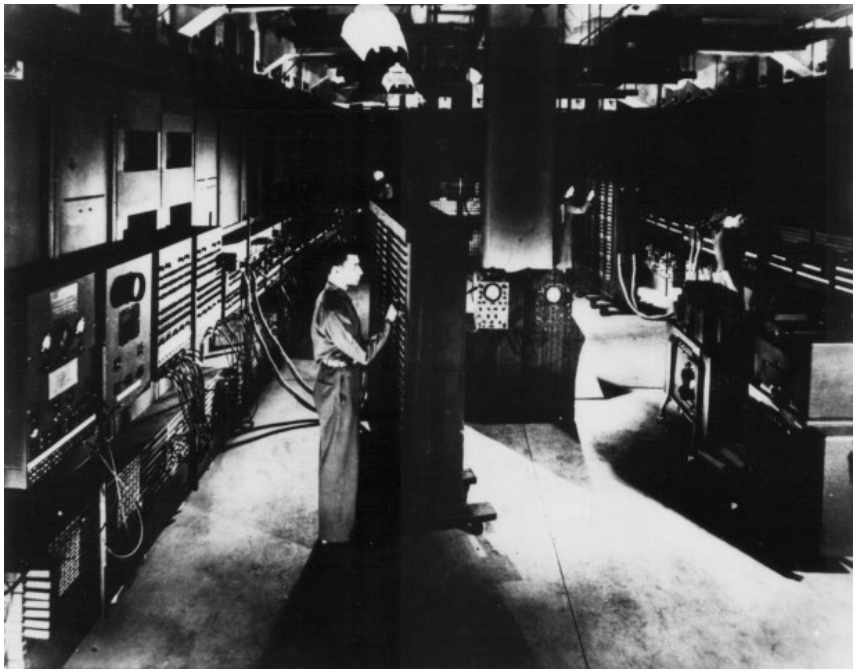
- The First Generation: Vacuum Tube Computers (1936 - 1945)



- Atanasoff Berry Computer (1937 - 1938) solved systems of linear equations. (not in real usage)
- John Atanasoff and Clifford Berry of Iowa State University.
- Konrad Zuse (1936-1945) built computers Z1-Z4 in Germany (not in real usage)
- ENIAC (1945, played an important role in in the Manhattan Project)

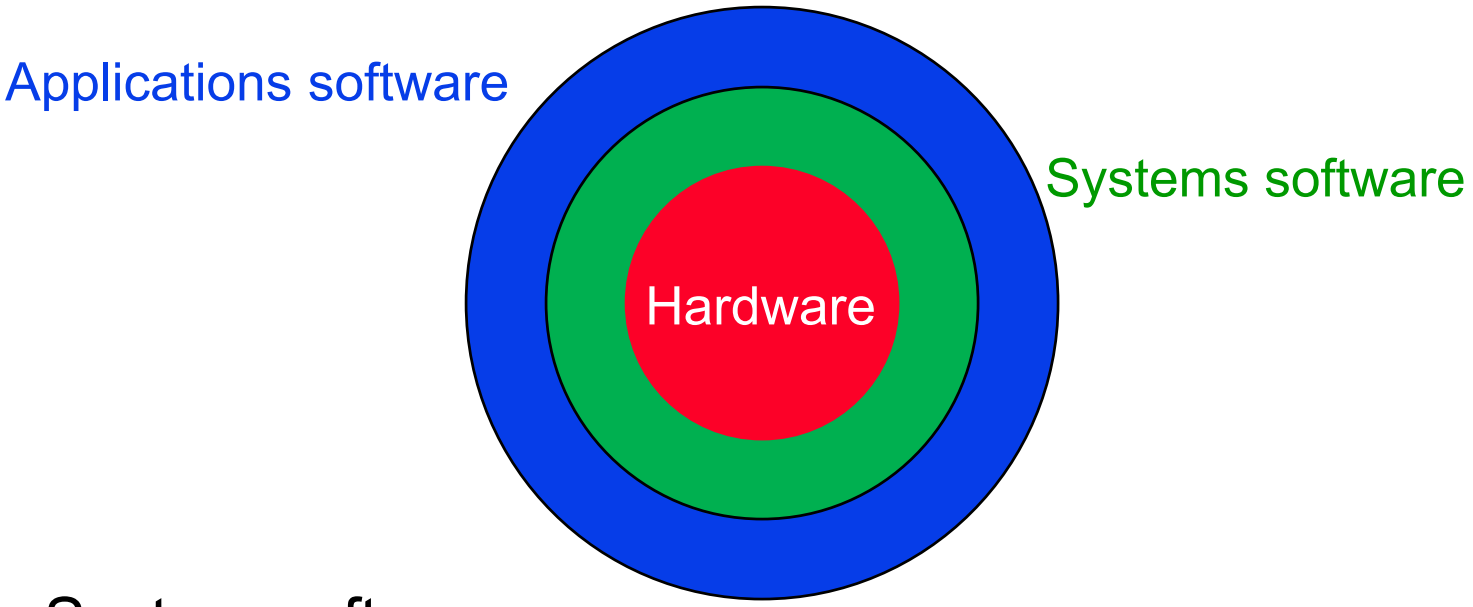
1945: ENIAC based on the prototype of ABC

- ENIAC built during World War II was the first general purpose computer
  - Used for computing artillery firing tables
  - 80 feet long by 8.5 feet high and several feet wide
  - Each of the twenty 10 digit registers was 2 feet long
  - Used 18,000 vacuum tubes
  - Performed 1900 additions per second



- Fugaku (2020)
  - 415.5 petaFLOPS (10^{15})

Below the Program

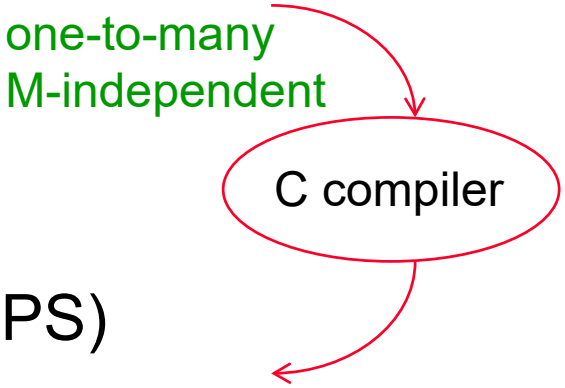


- System software
  - Operating system – a supervising program that interfaces the user's program with the hardware (e.g., Linux, MacOS, Windows)
    - Handles basic input and output operations
    - Allocates storage and memory
    - Provides for protected sharing among multiple applications
  - Compiler – translate programs written in a high-level language (e.g., C, Java) into instructions that the hardware can execute

Below the Program, Con't

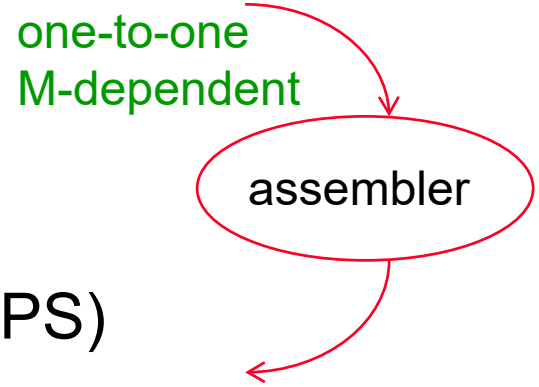
- High-level language program (in C)

```
swap (int v[], int k)
(int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
)
```



- Assembly language program (for MIPS)

```
swap:   sll     $2, $5, 2
        add     $2, $4, $2
        lw      $15, 0($2)
        lw      $16, 4($2)
        sw      $16, 0($2)
        sw      $15, 4($2)
        jr      $31
```



- Machine (object, binary) code (for MIPS)

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
 . . .
```

Course topics:

1. MIPS processor/memory instruction set architecture
2. Processor/memory performance metrics
3. Main memory (SRAM and DRAM) design
4. Design of integer arithmetic logic unit (ALU)
5. Floating point representation and arithmetic
6. Design of datapath and control logic
7. Pipelining
8. Cache and memory architecture
9. GPU
10. Storage systems

Additional notes based on advanced technologies in products

Your Learning Experience is the most important

Education Goal of the class:

- Critical Thinking and Innovative Thinking on top of the Foundational Knowledge

The purpose of the study:

- To prepare us for innovation