Introduction

Lecture 1
September 5th, 2017

Jae W. Lee (jaewlee@snu.ac.kr)
Computer Science and Engineering
Seoul National University

***Slide credits**: [CS:APP3e] slides from CMU; [COD5e] slides from Elsevier Inc.*

Course information

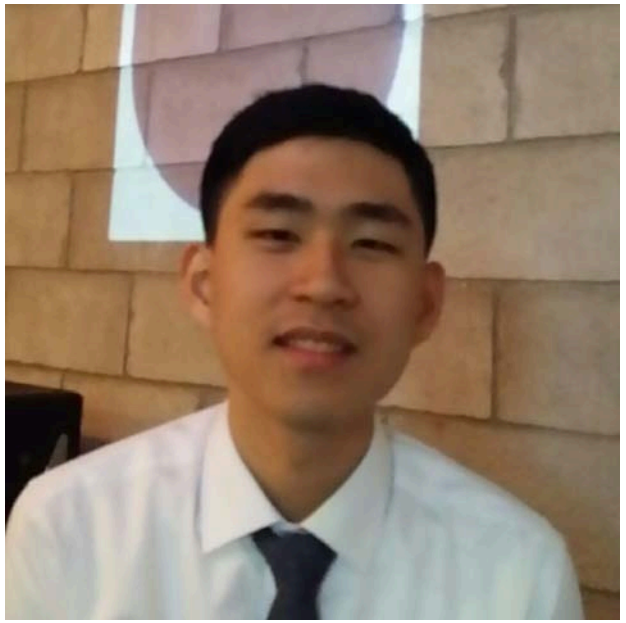
- **Schedule**
  - 11 AM-12:15 PM [Tue/Thur] + recitation lecture (보충강의) [Fri]
  - Lecture Room: #302-208

- **Instructor**
  - Jae Wook Lee (이재욱, jaewlee@snu.ac.kr)
  - Office: Engineering Building #301-506
  - Phone: 02-880-1834
  - Office Hours: Tuesdays 5-6 PM or by appointment

Course information: (Awesome) TAs

- **Daeyeon KIM (김대연)**
  - Email: polaris6921@gmail.com

- **Seonghak KIM (김성학)**
  - Email: ksh931102@gmail.com



- **Office: Engineering Building #301-554-1**
- **Phone: 02-880-1834**
- **Office Hours: TBA**

Course information

- **This is a sophomore-level course with the following prerequisites:**
  - Logic Design (논리설계) – required
  - Basic knowledge of C – recommended
- **This course will be given in English (aka 영어강의).**
  - But you are welcome to ask questions in either Korean or English.
- **Course materials will be distributed through eTL.**
- **All lectures will be videotaped!**
  - To help your study

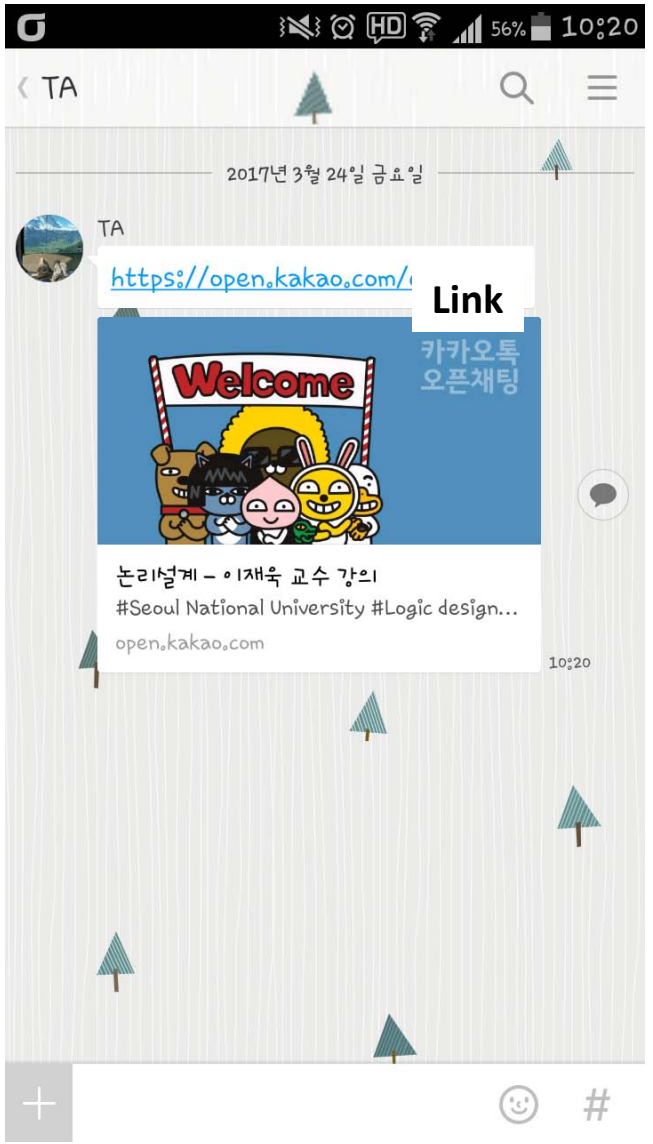
Course information

- **TA recitation on Fridays**
  - TA will review course materials (in Korean) and take questions
  - Your attendance is completely optional!
    - Does not count for your attendance score
    - However, I recommend those who need help to understand course materials attend!
  - First recetation (9/9): "Linux Bootcamp"
  - Quick poll: Which of the following slots do NOT work for you?
    - 11 AM – noon on Friday
    - 1 – 2 PM on Friday
    - 2 – 3 PM on Friday

Course information

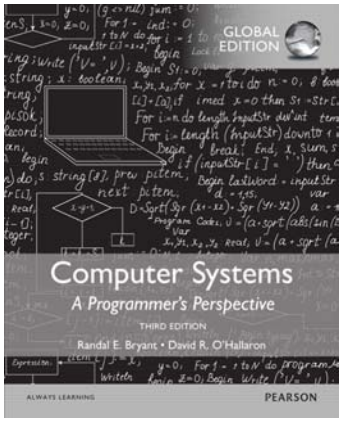
- **Open Kakao chat for the course**
  - We will use open chat for casual, anonymous communication.
    - Some forbidden nicknames include professor, Prof. Lee, 이재욱 (교수), TA, 조교, 김대연, 김성학, etc.
  - Everyone is invited, but not required, to join.
    - Using it is completely optional
  - Primarily used for questions during the class
    - Will answer your questions outside the classroom as time permits

- **More details will be announced soon**

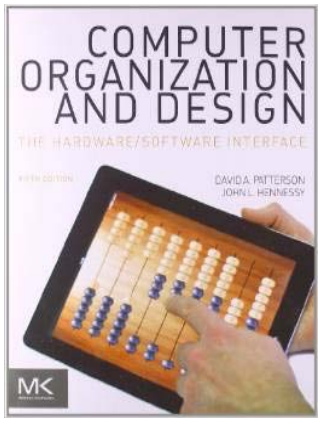


Course information: Textbooks

- **Main: Computer Systems: A Programmer's Perspective [CS:APP3e]**
  - Authors: Randal E. Bryant and David R. O'Hallaron
  - Pearson, 2015 (3rd Edition)
- **Reference: Computer Organization and Design [COD5e]**
  - Authors: David A. Patterson and John L. Hennessy
  - Elsevier Inc., 2013 (5th Edition)
  - You don't have to buy this one, but I may occasionally use it.



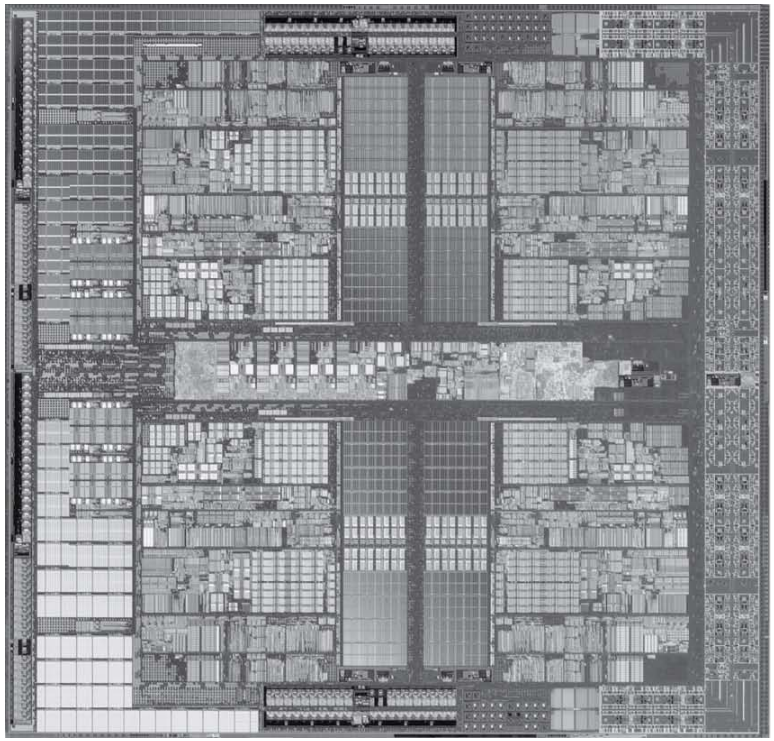
Main

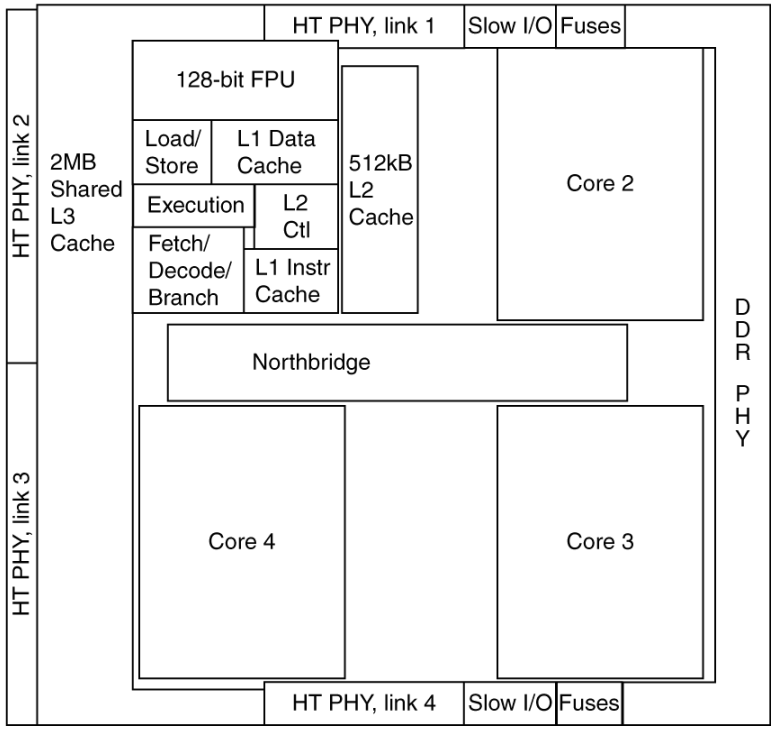


Reference

What we learn in this course: Overview

- You will understand what each block does by the end of this term!
- You will also learn how to program the CPU and write efficient code!





AMD Barcelona: 4 processor cores

What we learn in this course: Specifics

- **How programs are translated into the machine language**
  - And how the hardware executes them
- **The hardware/software interface – Instruction Set Architecture**
- **What determines program performance**
- **How hardware designers/software writers improve performance**
- **What is parallel processing**

What we learn in this course: Course goals

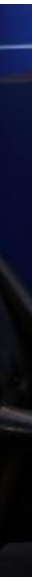
- **To understand**
  - Interfaces
    - Instruction Set Architecture (ISA) – The Hardware/Software Interface
  - Engineering methodology / Correctness criteria / Evaluation methods / Technology trends involved in
    - Processor
    - Cache memory
    - Virtual memory
    - I/O system

Today's processors beyond CPUs (1)

- **Graphics Processing Units (GPUs)**
  - Very good at data-intensive processing
  - Example: Nvidia's Tesla GV100 GPU (Volta)

Today's processors beyond CPUs (2)

- **Application-specific accelerators: Google Tensor Processing Unit**
  - Server racks with TPUs used in AlpahGo matches with Lee Sedol
  - Massive amount of computation with custom hardware design



In-Datacenter Performance Analysis of a Tensor Processing Unit™

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon

Google, Inc., Mountain View, CA USA

Email: {jouppi, cliffy, nishantpatil, davidpatterson}@google.com

To appear at the 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada, June 26, 2017.

Published in June 2017!

(180TFLOPS)

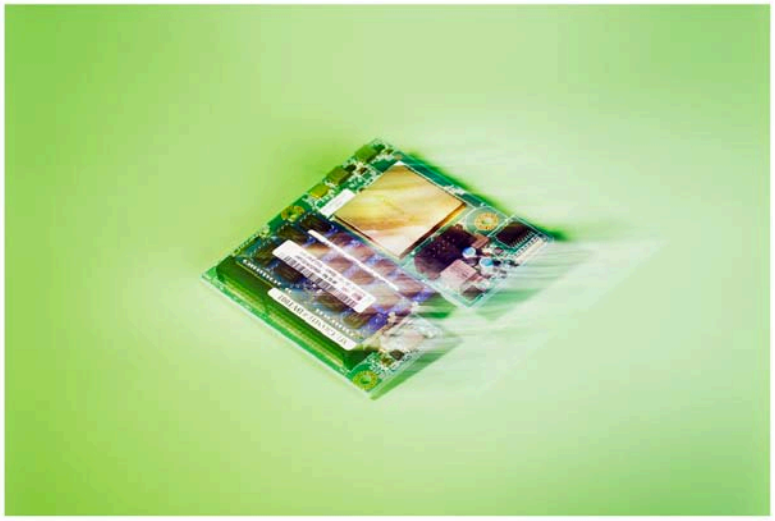
Source: https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html

Today's processors beyond CPUs (3)

- **Application-specific accelerators: Microsoft datacenter**
  - Microsoft online services (like Bing, MSN, Azure) are running on FPGA-based custom processors.

CADE METZ BUSINESS 09.25.16 7:00 PM

MICROSOFT BETS ITS FUTURE ON A REPROGRAMMABLE COMPUTER CHIP



Prototype hardware for Project Catapult, a six-year effort to rebuild Microsoft's online empire for the next wave of AI—and more. CLAYTON COTTERELL FOR WIRED

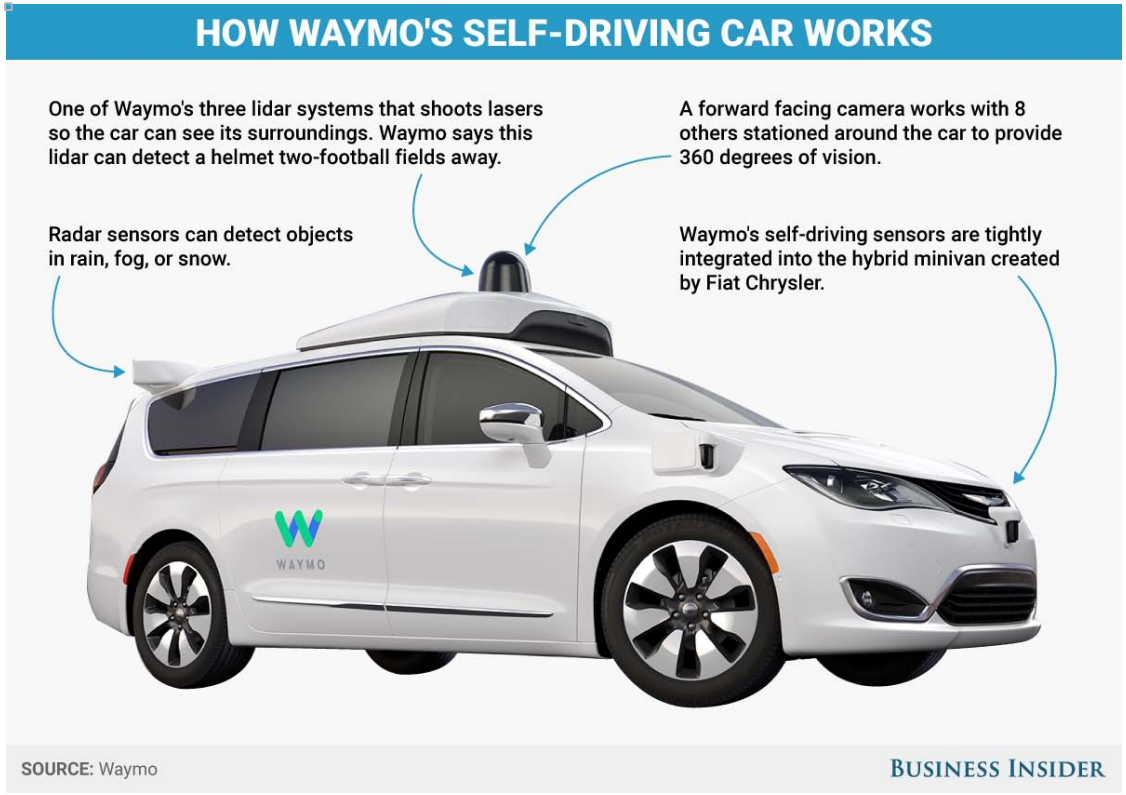
IT WAS DECEMBER 2012, and Doug Burger was standing in front of Steve Ballmer, trying to predict the future.

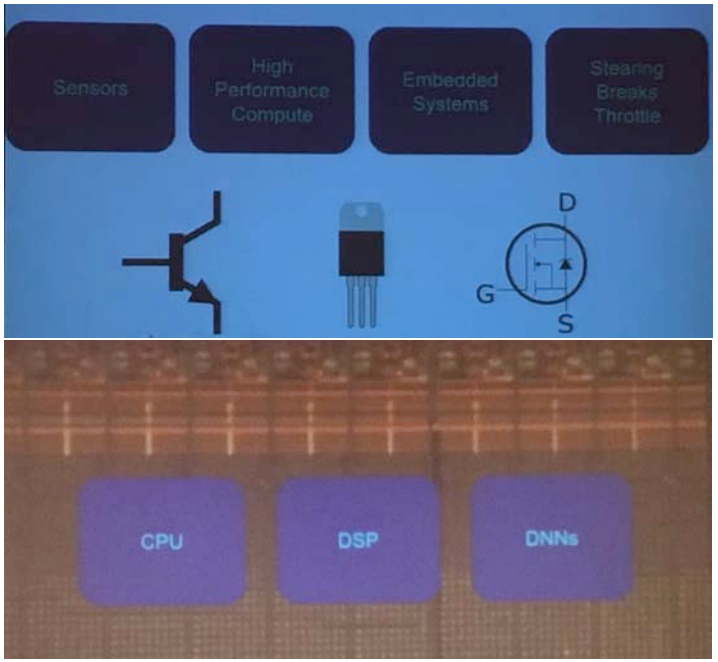
Ballmer, the big, bald, boisterous CEO of Microsoft, sat in the lecture room on the ground floor of Building 99, home base for the company's blue-sky R&D lab just outside Seattle. The

 Source: https://www.wired.com/2016/09/microsoft-bets-future-chip-reprogram-fly/

Today's processors beyond CPUs (4)

- **Application-specific accelerators: Google's self-driving cars**
  - It's a known secret that Google's self-driving cars are running on custom processors with CPU+DSP+Deep Learning accelerators.





Source: Dan Rosenband, "Inside Waymo's Self-Driving Car: My Favorite Transistors", VLSI Symposium, 2017.

Source: http://www.businessinsider.com/how-does-googles-waymo-self-driving-car-work-graphic-2017-1

Outline

Textbook: [CS:APP3e] 1.1-1.6; [COD5e] 1.1, 1.3-1.8

- **Overview**
- **Why you should take this course? Five great realities.**
- **Under the cover**
- **Understanding performance**

- **Course schedule and grading system**

Overview: Computer Revolution

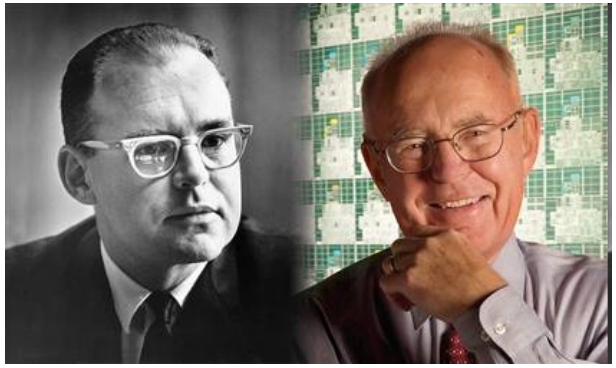
- **Progress in computer technology**
  - Underpinned by Moore's Law
- **Makes novel applications feasible**
  - Computers in automobiles
  - Cell phones
  - Machine learning/AI
  - Human genome project
  - World Wide Web
  - Search engines
- **Computers are pervasive**

Overview: Moore's Law (1965)

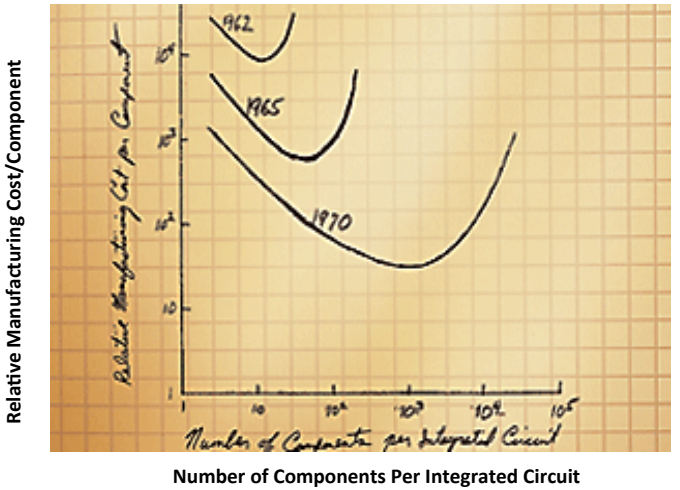
- **The single most important guideline in microprocessor fabrication and architecture**

  "the number of transistors per chip will double every 18 months"

- **Main driving force for the phenomenal growth of IT industry**



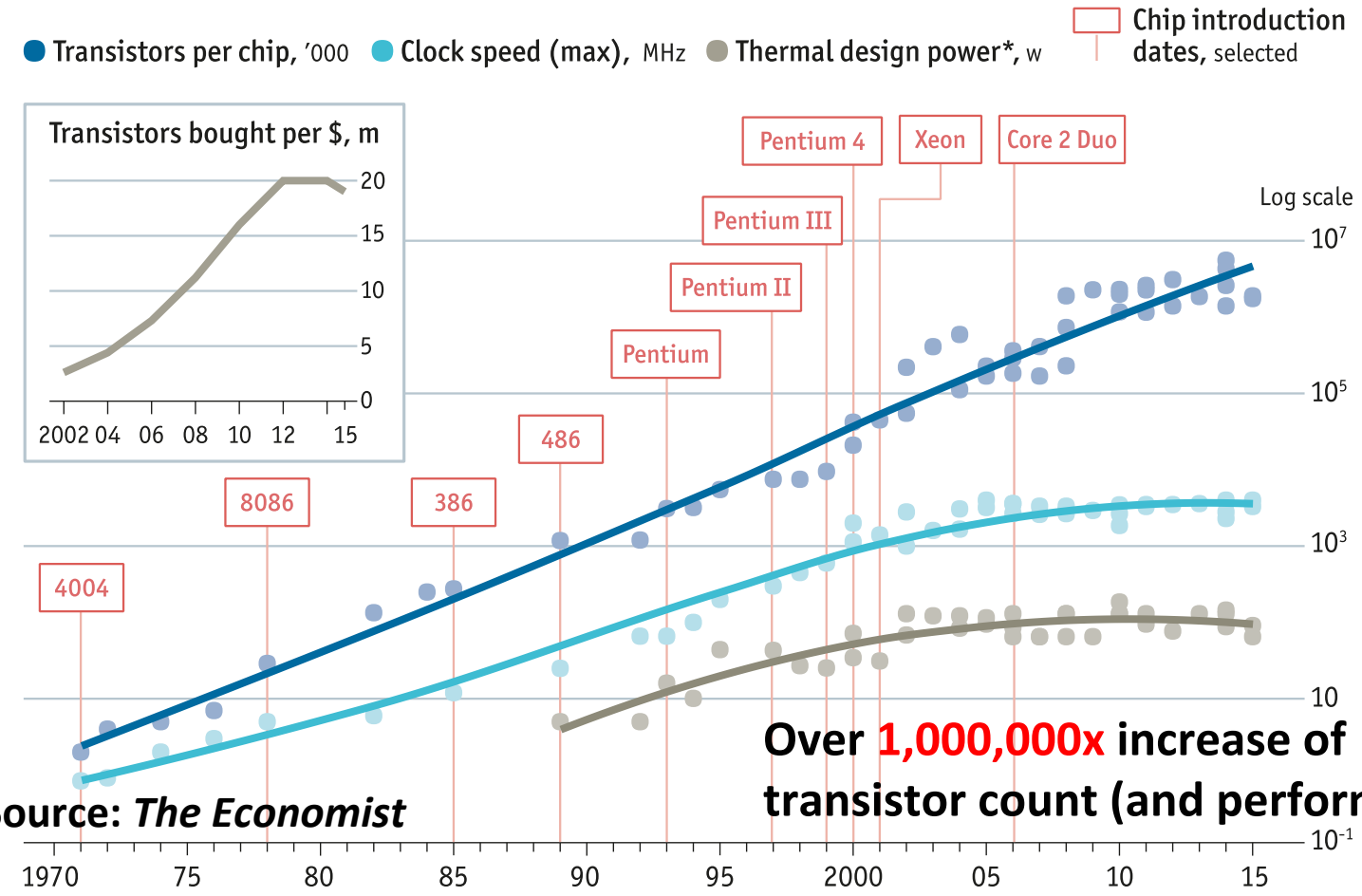
(http://download.intel.com/museum/research/arc_collect/history_docs/pix/hoff1.jpg)



Gordon Moore Original graph from 1965 *(source: www.intel.com)*

Overview: Moore's Law (1965)

- **Glorious 50 years of Moore's Law**
  - Recently being slowed down



Source: *The Economist*

Over 1,000,000x increase of transistor count (and performance)!

Why you should take this course?

- Because….You won't graduate if you don't take this course.
- Because….You want to design the next great instruction set.
  - Instruction set architecture has largely converged, especially in the desktop/server/laptop space.
  - Dictated by powerful market forces (Intel/ARM).
- Because….You want to become a computer architect and design the next great computer systems.
- Because….The design, analysis, implementation concepts that you will learn are vital to all aspects of computer science and engineering – operating systems, computer networks, compiler, programming languages
- Because….The course will equip you with an intellectual toolbox for dealing with a host of systems design challenges
- And much more !!!

Source: Prof. Fernando C. Colon Osorio's lecture notes

Why you should take this course?

- **At this point (some of) your possible reactions:**
  - But… I hate hardware and will NOT design my own processor for sure. Should I still care??
  - I write code in high-level languages like C, Java, Python, MATLAB, etc., and the good compiler takes care of the rest. Why should I bother??

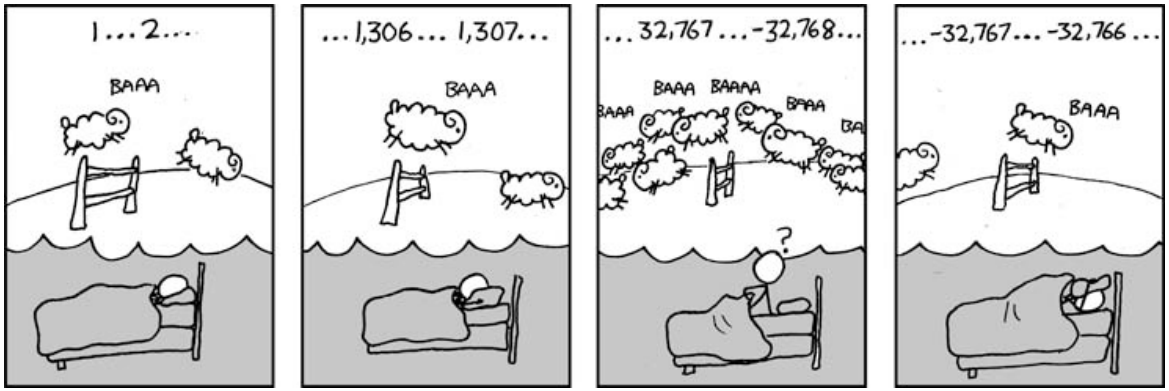
- **Indeed, CS heavily builds on the notion of abstraction**
  - To provide a simplified view of the reality
  - e.g., abstract data types, asymptotic analysis

Why you should take this course?

- **Our Motto: "Abstraction is good but don't forget reality!"**
  - Abstractions have limits (especially in the presence of bugs)
  - Need to understand details of underlying implementations
- **Positive outcomes of learning "realities" in computer system**
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance
  - Prepare for later "systems" classes in CSE
    - Compilers, Operating Systems, Networks, Embedded Systems, Storage Systems, etc.
- **Now I will present five example cases where the reality matters.**

Great Reality #1:
Ints are not Integers, Floats are not Reals

- **Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!
  - Int's:
    - 40000 * 40000 --> 1600000000
    - 50000 * 50000 --> ??



- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Source: xkcd.com/571

Great Reality #1:
Ints are not Integers, Floats are not Reals

- **Computer arithmetic does not generate random values**
  - Arithmetic operations have important mathematical properties
- **Cannot assume all "usual" mathematical properties**
  - Due to finiteness of representations
- **Observation**
  - Need to understand which abstractions apply in which contexts
  - Important issues for compiler writers and serious application programmers

Great Reality #2: You've Got to Know Assembly

- **Chances are, you'll never write programs in assembly**
  - Compilers are much better & more patient than you are
- **But: Understanding assembly is key to machine-level execution model**
  - Behavior of programs in presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage process state
  - Creating / fighting malware
    - x86 assembly is the language of choice!

Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated
- **Memory referencing bugs especially pernicious**
  - Effects are distant in both time and space
- **Memory performance is not uniform**
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

- **Memory referencing bug example: Result is system specific**

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

```
fun(0)   =     3.14
fun(1)   =     3.14
fun(2)   =     3.1399998664856
fun(3)   =     2.0000061035156
fun(4)   =     3.14
fun(6)   =     Segmentation fault
```

Great Reality #3: Memory Matters

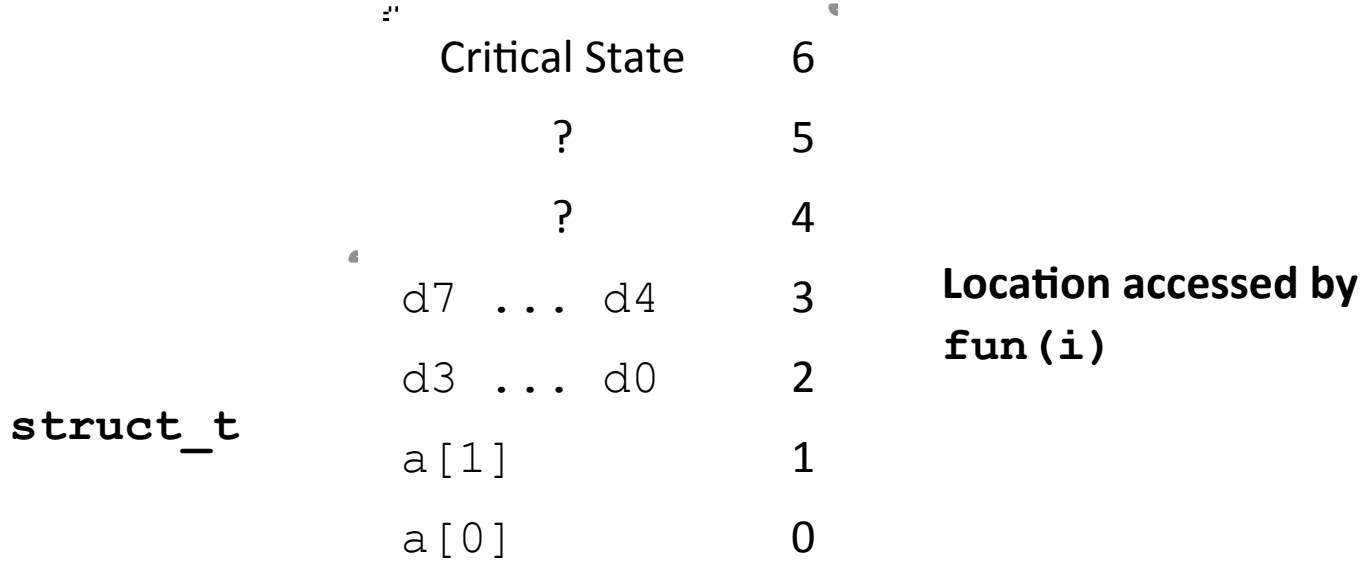
Random Access Memory Is an Unphysical Abstraction

- **Memory referencing bug example: Explanation**

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0)  =  3.14
fun(1)  =  3.14
fun(2)  =  3.1399998664856
fun(3)  =  2.0000061035156
fun(4)  =  3.14
fun(6)  =  Segmentation fault
```

Explanation:



Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

- **C and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- **Can lead to nasty bugs**
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated to one being accessed
    - Effect of bug may be first observed long after it is generated
- **How can I deal with this?**
  - Program in Java, Ruby, Python, ML, …
  - **Understand what possible interactions may occur**
  - Use or develop tools to detect referencing errors (e.g. Valgrind)

Great Reality #4: There's more to performance than asymptotic complexity

- **Constant factors matter too!**
- **And even exact op count does not predict performance**
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **Must understand system to optimize performance**
  - How programs compiled and executed
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality

Great Reality #4: There's more to performance than asymptotic complexity

- Memory system performance 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];
}
```

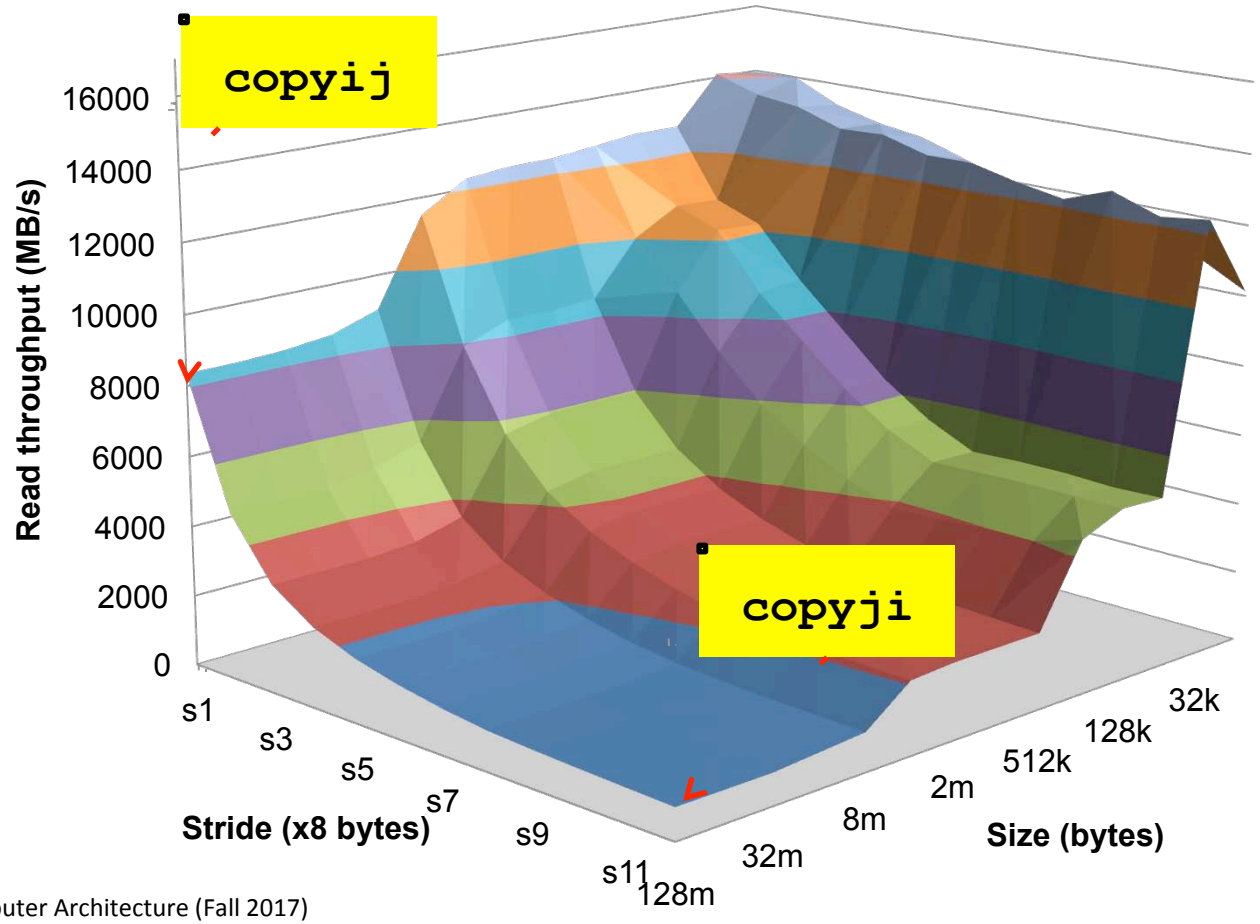
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms 2.0 GHz Intel Core i7 Haswell **81.8ms**

- Hierarchical memory organization
- Performance depends on access patterns
  - Including how step through multi-dimensional array

Great Reality #4: There's more to performance than asymptotic complexity

- Why the performance differs?

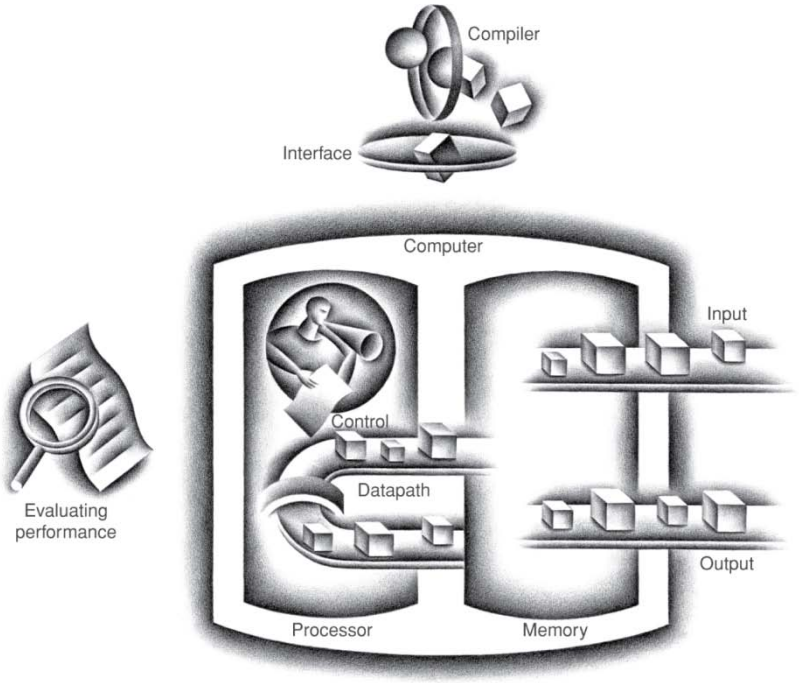


Great Reality #5: Computers do more than execute instructions

- **They need to get data in and out**
  - I/O system critical to program reliability and performance
- **They communicate with each other over networks**
  - Many system-level issues arise in presence of network
    - Concurrent operations by autonomous processes
    - Coping with unreliable media
    - Cross platform compatibility
    - Complex performance issues

Under the Cover: Components of a computer

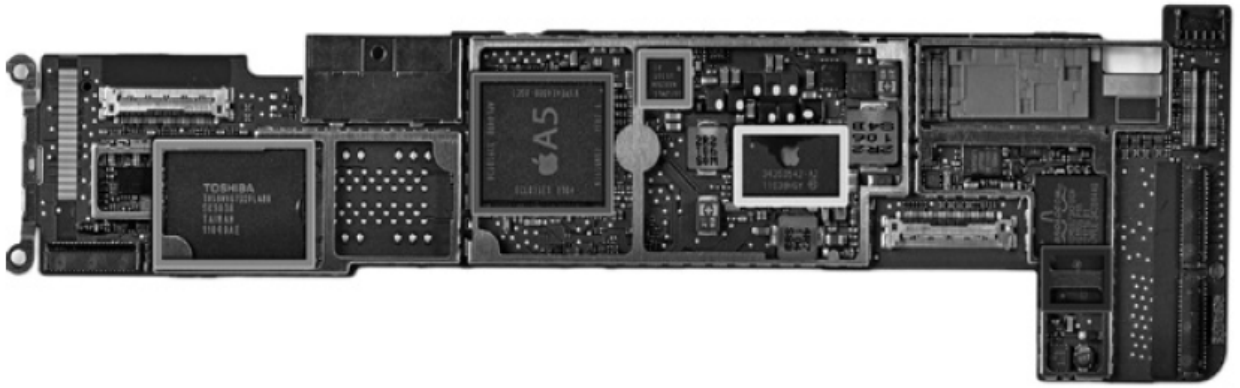
The BIG Picture



- **Common for all kinds of computer**
  - Desktop, server, embedded
- **Processor (Datapath/Control), Memory, and I/O**
- **Input/output (I/O) includes**
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

Under the cover: Opening the box

- Capacitive multitouch LCD screen
- 3.8 V, 25 Watt-hour battery
- Computer board



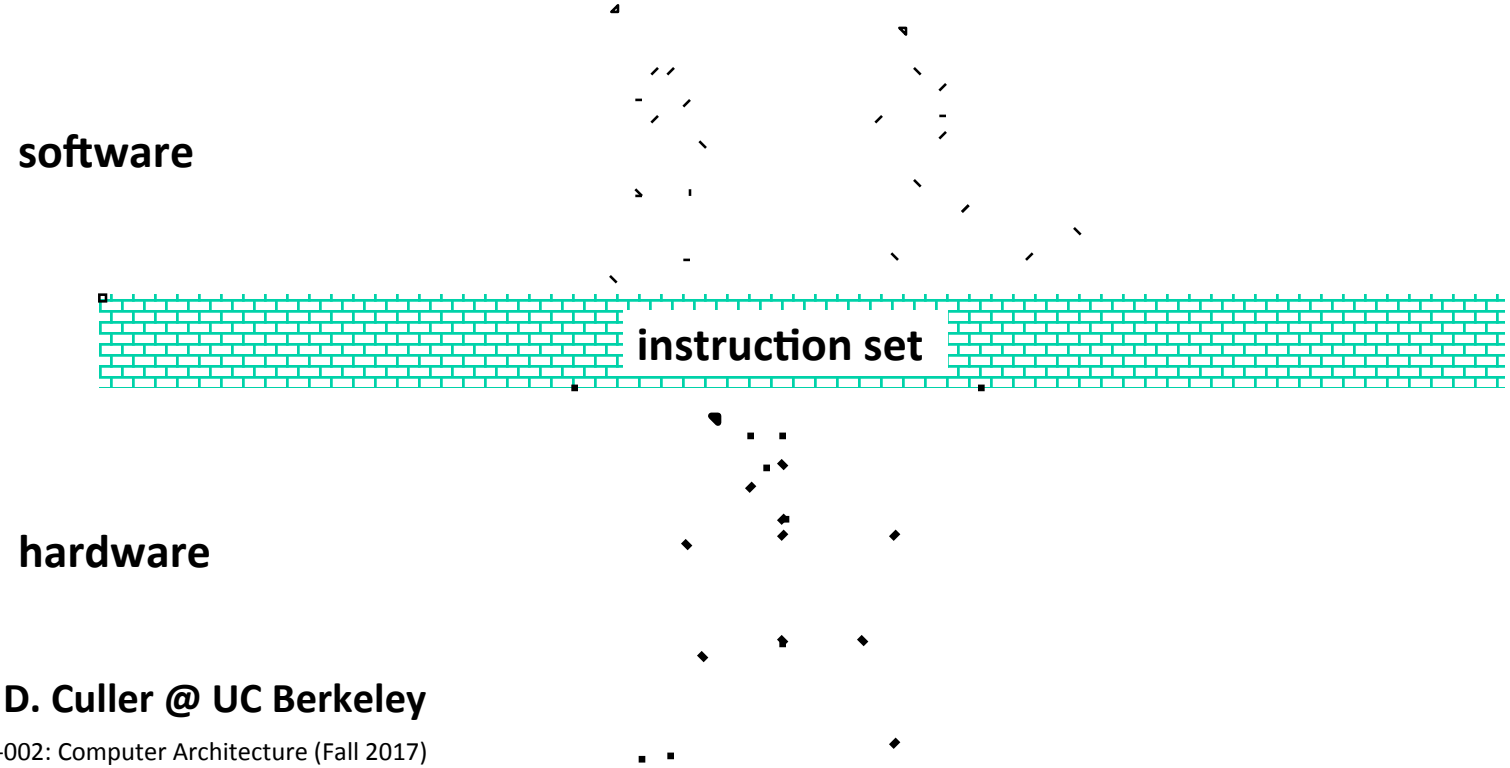
Under the cover: Inside the processor (CPU)

- **Apple A5**

You will understand what each block does by the end of this course!

Under the cover: Abstractions

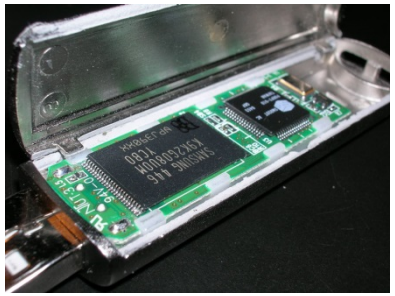
- **Abstraction helps us deal with complexity**
  - Hide lower-level detail
- **Instruction set architecture (ISA)**
  - Hardware abstraction visible to software (compiler or programmer)
  - The hardware/software interface



Source: D. Culler @ UC Berkeley

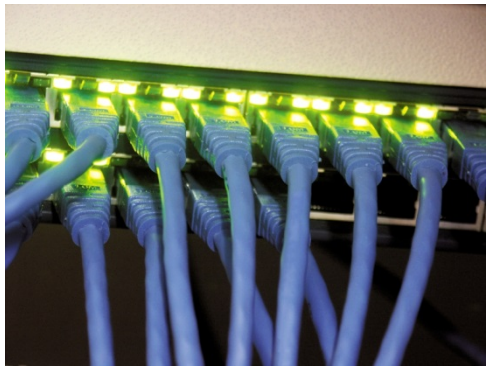
Under the cover: Safe place for data

- **Volatile main memory**
  - Loses instructions and data when power off
  - E.g., DRAM
- **Non-volatile secondary memory**
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)



Under the cover: Networks

- Communication and resource sharing
- Local area network (LAN): Ethernet
  - Within a building
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth



Under the cover: Technology trends

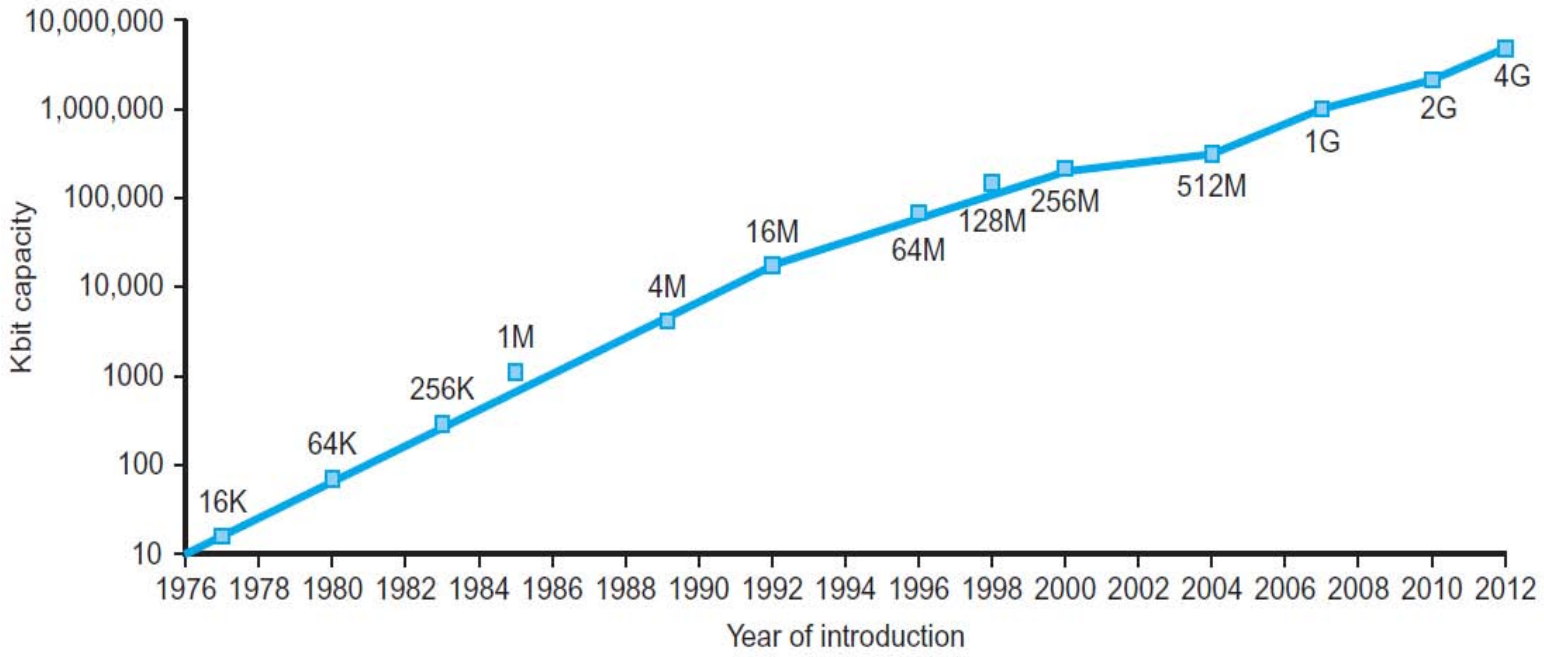
- **Processor**
  - Logic capacity: about 30%/year
  - Clock rate: about 20%/year (and slowing down)

- **Disk**
  - Capacity: about 60%/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

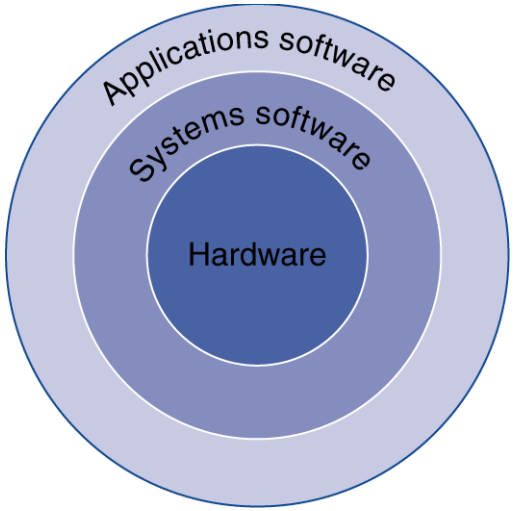
Under the cover: Technology trends

- **DRAM (Memory)**
  - Capacity: about 60%/year (4x every 3 years)
  - Speed: about 10%/year



Understanding performance: 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

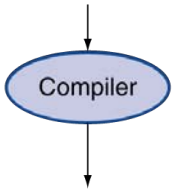


Understanding performance: Levels of program code

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

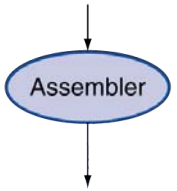
High-level language program (in C)

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



Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000000011000000110000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Understanding performance: Factors

- **Algorithm**
  - Determines number of operations executed
- **Programming language, compiler, instruction set arch.**
  - Determine number of machine instructions executed per operation
- **Processor and memory system**
  - Determine how fast instructions are executed
- **I/O system (including OS)**
  - Determines how fast I/O operations are executed

Course coverage and schedule*

Schedule	Contents	Remarks [CS:APP3e]
Week 1	Course Outline & Introduction	Ch. 1
Week 2	Number Representations	Ch. 2
Weeks 3~5	Instruction Set Architecture (ISA)	Ch. 3
Weeks 6~7	Sequential Architecture	Ch. 4.1-4.3
Weeks 8~10	Pipelined Architecture + **Midterm (10/31)**	Ch. 4.4-4.5
Weeks 11~12	Cache Memory	Ch. 6
Week 13	Virtual Memory	Ch. 9
Week 14	Performance	[COD5e] Ch. 1.6
Week 15	Instruction to Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs)	
Week 16	**Final**	

**** Schedule is tentative and subject to change.***

Grading System*

- Programming Assignments: 35% (~5 times)
- Midterm: 25%
- Final: 40%
- Attendance/Class Participation: up to ±5%

** Subject to change.*