

# So What's Next?

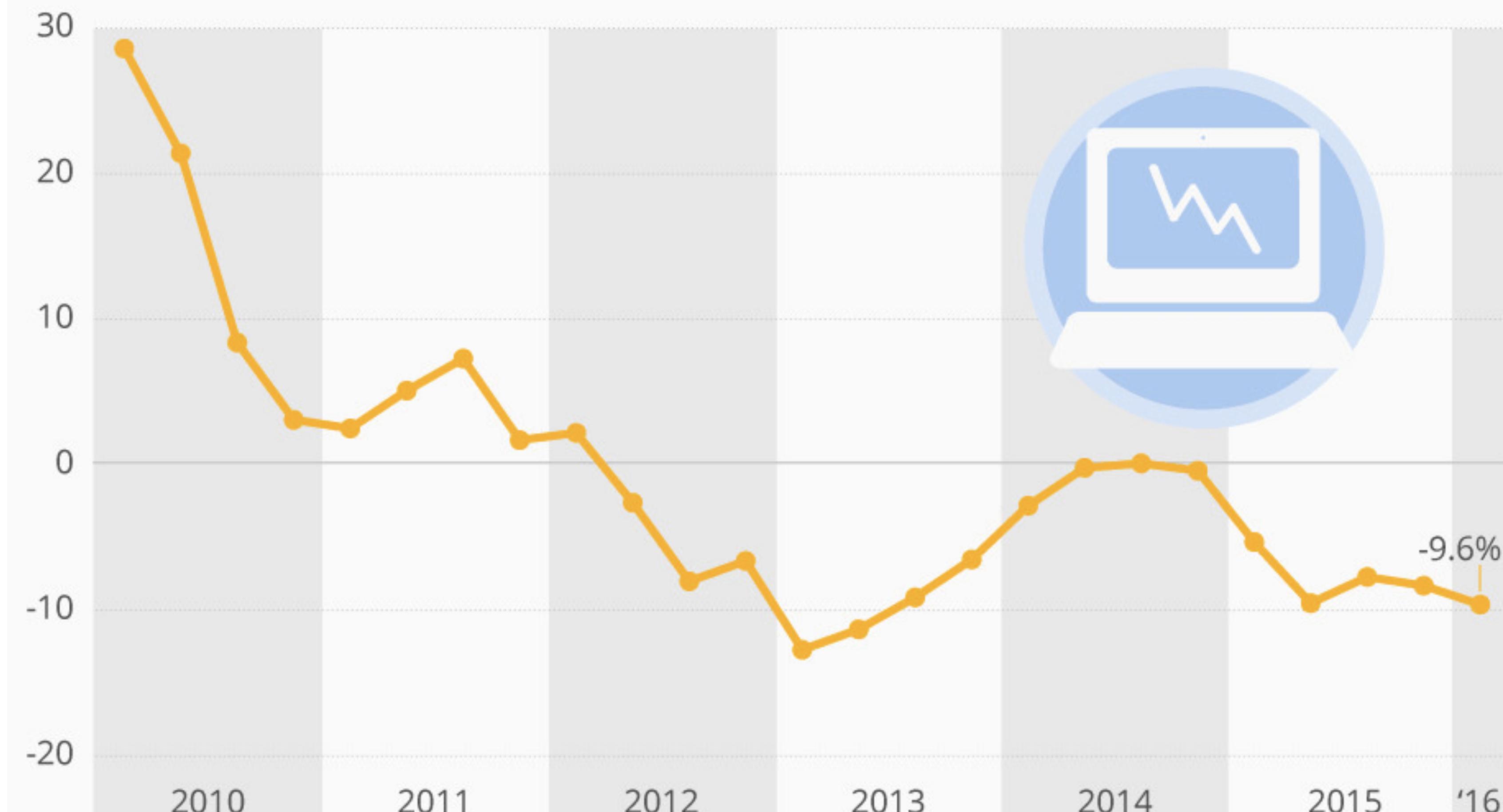
# Where Do We Go From Here?

- A Review of the Class
- A Map of the Future
- Future Classes at Berkeley



## Worldwide PC Market Shrinking Further

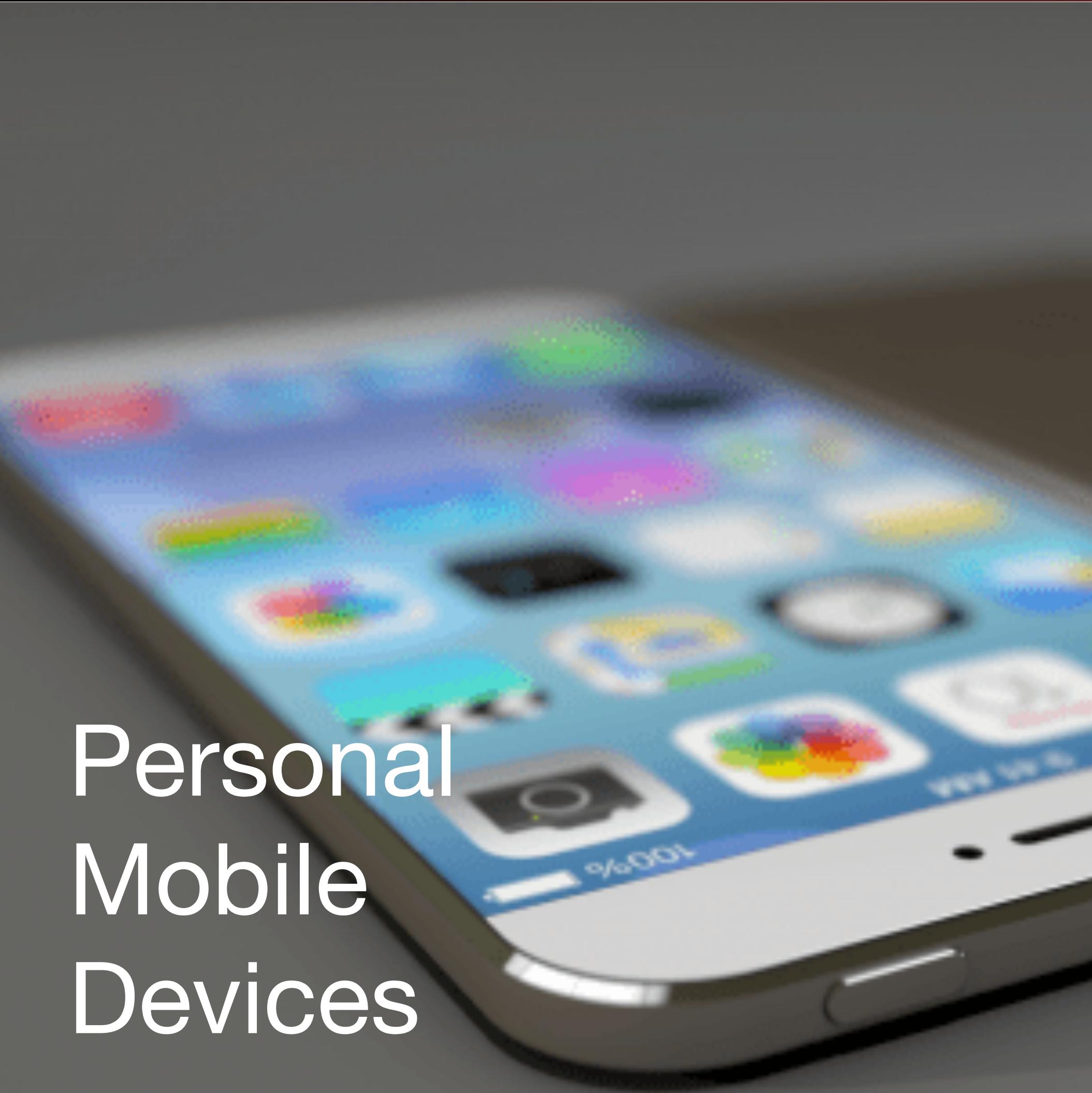
Global PC shipments since 2010\*



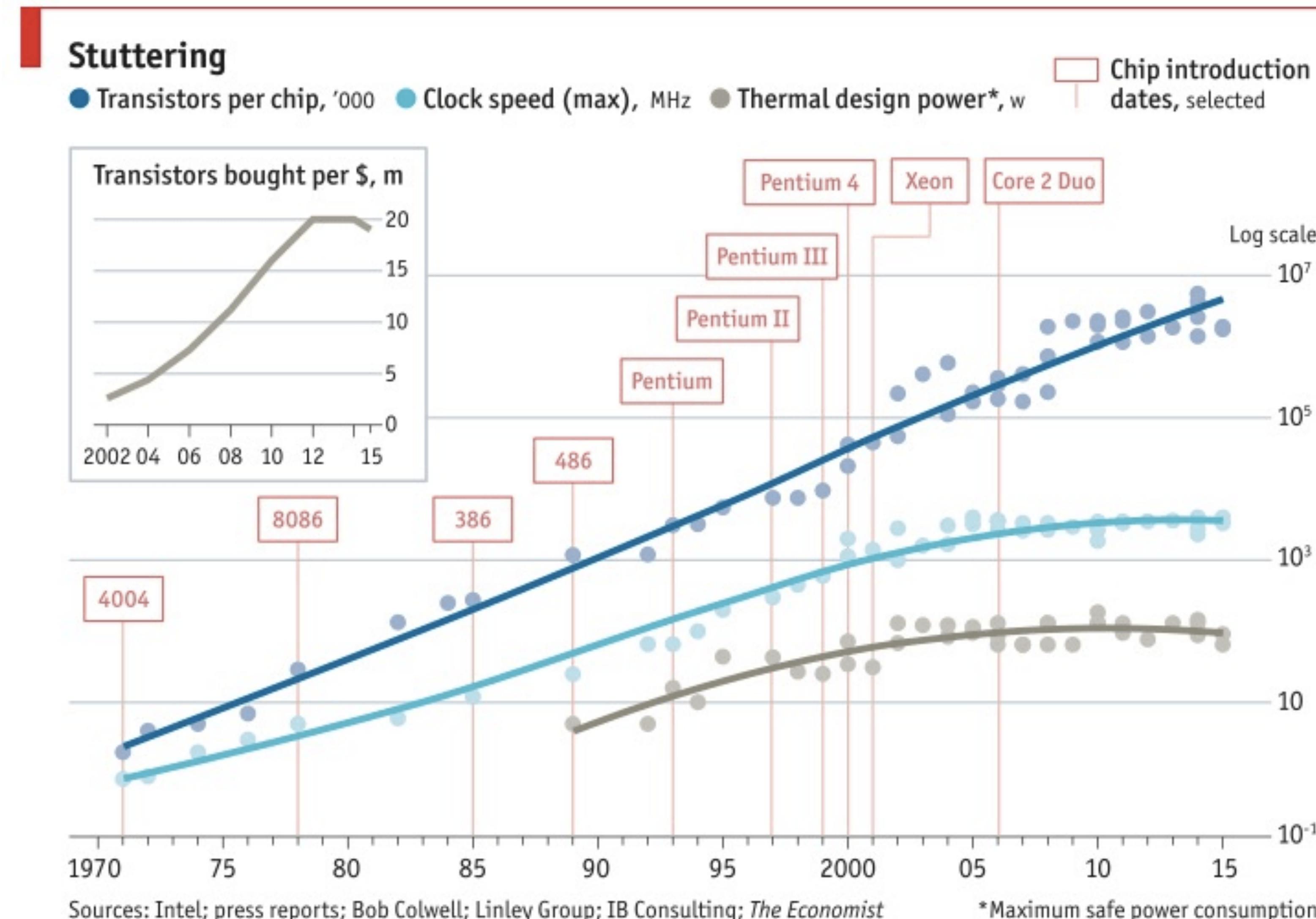
# Current 61C: The Same Concepts Over a Mass Scale

Computer Science 61C

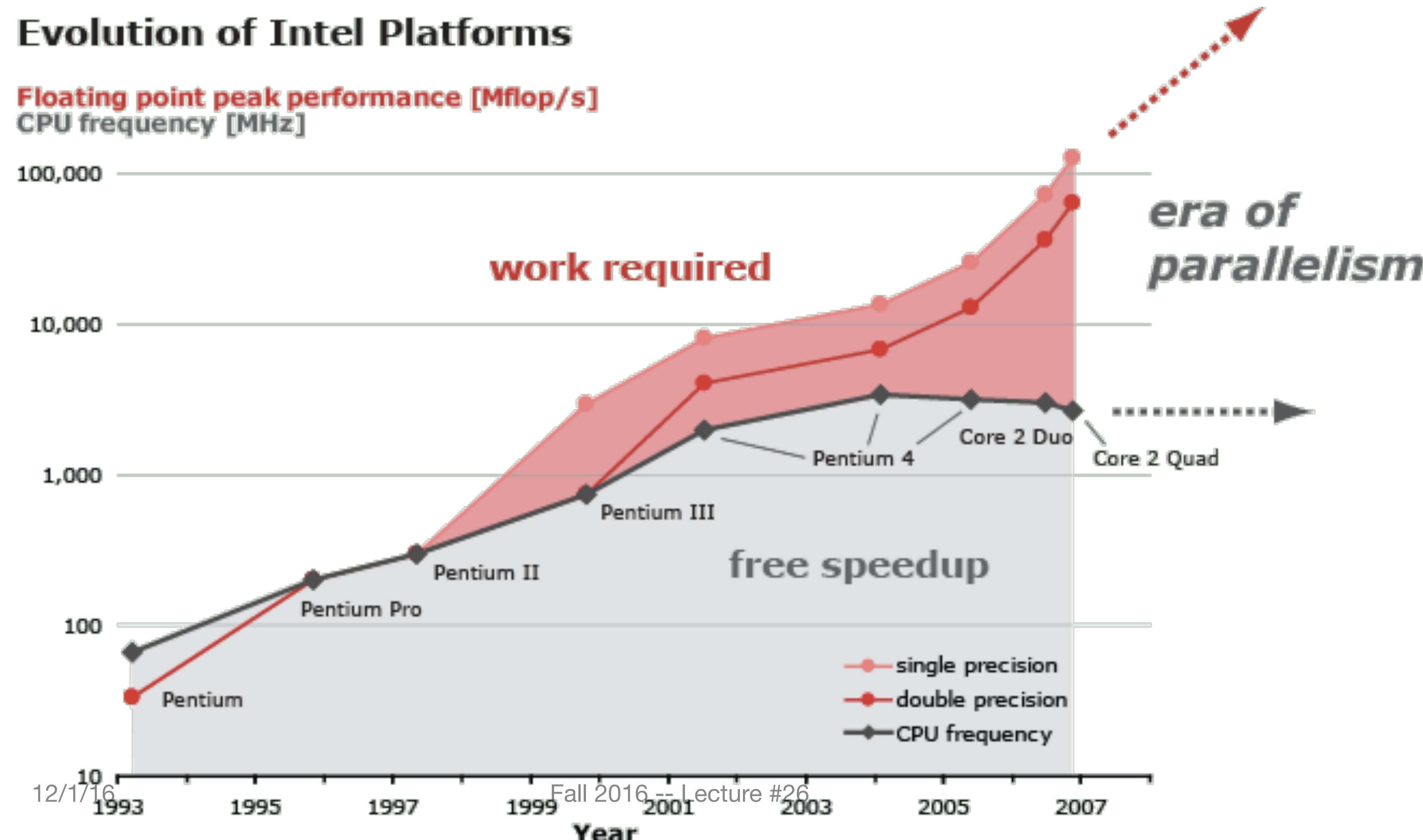
Wawrynek & Weaver



# All Have Hit the Single-Thread Brick Wall



# Leaving Parallelism the *only* way to improve throughput



# Well, with an \* of course...

- "If you put enough rockets on it, a pig will fly...  
And Intel has strapped that pig to a Saturn V"
  - Truly heroic efforts have gone into performance for an ISA that, even in 64b mode, is crippled by legacy as there are only a few general registers and a lot of complex operations
- The big deals:
  - Translate x86 to a RISC-like internal representation (micro-OPs)
    - Adds a couple of pipeline stages
  - Heroically out-of-order, superscalar, and register renaming
    - ~150 actual registers although x86 only has a couple general purpose integer registers
  - Heroic efforts at branch prediction
    - But really hit diminishing returns:  
A mispredicted branch **kills** performance and you can only predict so well

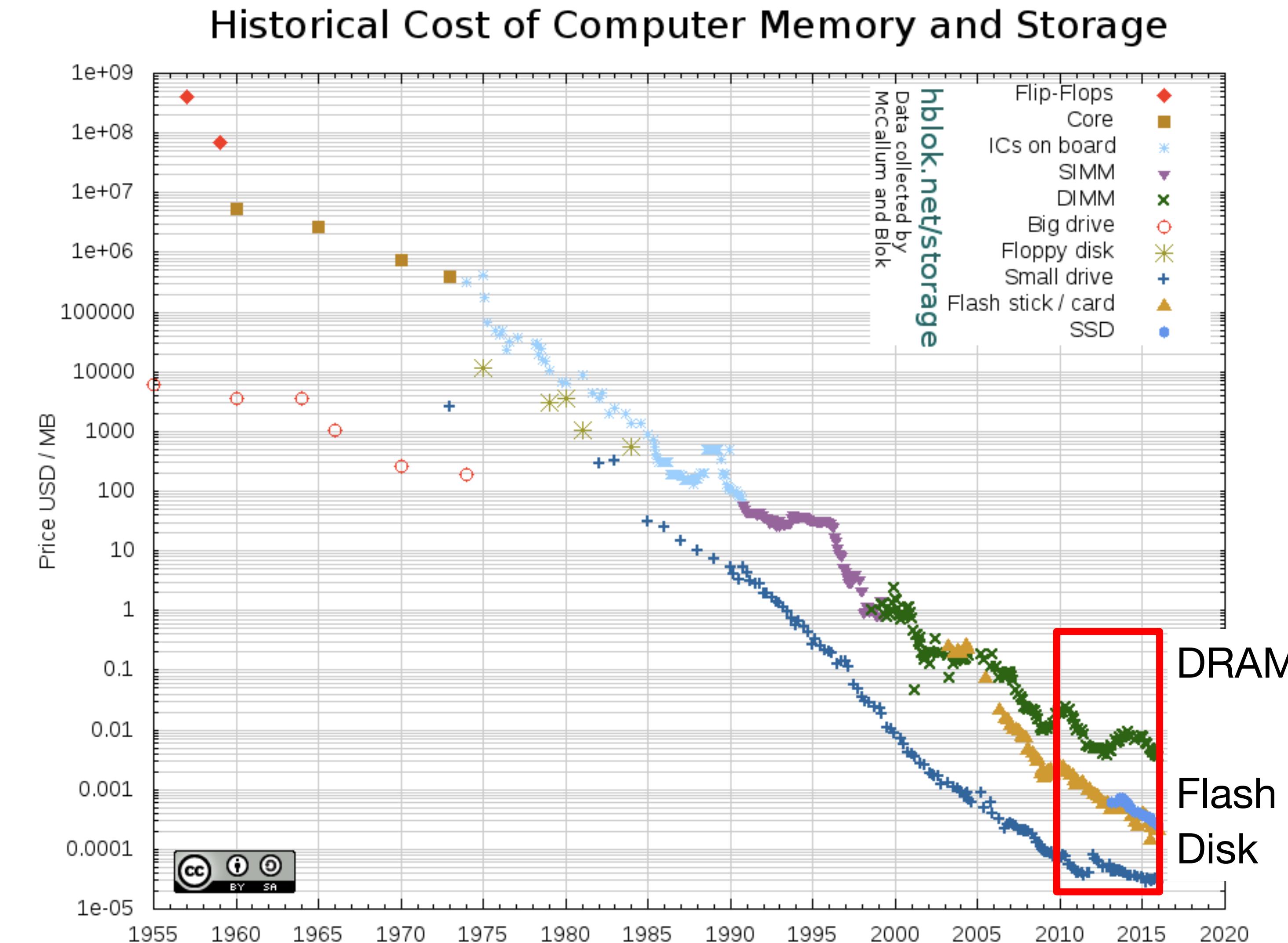
# Apple is far less wedded to ISA... And Has More Money Than God...

- So the Apple M1: A ***high performance*** Arm v8.5
  - 64b Arm mostly competed in the power space:  
Having 2-3 less pipeline stages needed for translation saves a lot...  
and phones were never wedded to x86
    - Arm isn't quite a RISC: its only semi-RISC as there are a lot of instructions even just restricted to 64b...
      - Example of the not-really-RISC nature:  
**FJCVTZS**: Floating-point Javascript Convert to Signed fixed-point, rounding toward Zero
      - But it **does** have a RISC-sized register file: 32 registers.
  - So the M1 says...
    - Hey, lets strap Arm to the ***same Saturn-V rocket!***
      - Yes, its horribly expensive development costs, but we have billions of \$ in cash doing nothing, so we might as well
      - And we were already getting close on our latest phones!  
Our limit was power consumption...

# Results are stunningly good

- It isn't just process
  - Intel is still mostly at a 14nm process, although latest at "10nm+"
  - AMD uses TSMC's 7nm process
  - M1 is TSMC's 5nm process
- It isn't just implementation architecture
  - 4 high performance cores use all the tricks x86 does, 4 low power cores
- It probably comes down to an observation:  
All the renaming in the world can't make up for not having enough registers for the compiler
- Of course, this is a ***one time*** boost in sequential performance
  - It is ***not repeatable*** so hey...

# And Other Things Are Still Getting Cheaper & Better (until the Crypto-Bros strike...)



# New-School Machine Structures

## Software

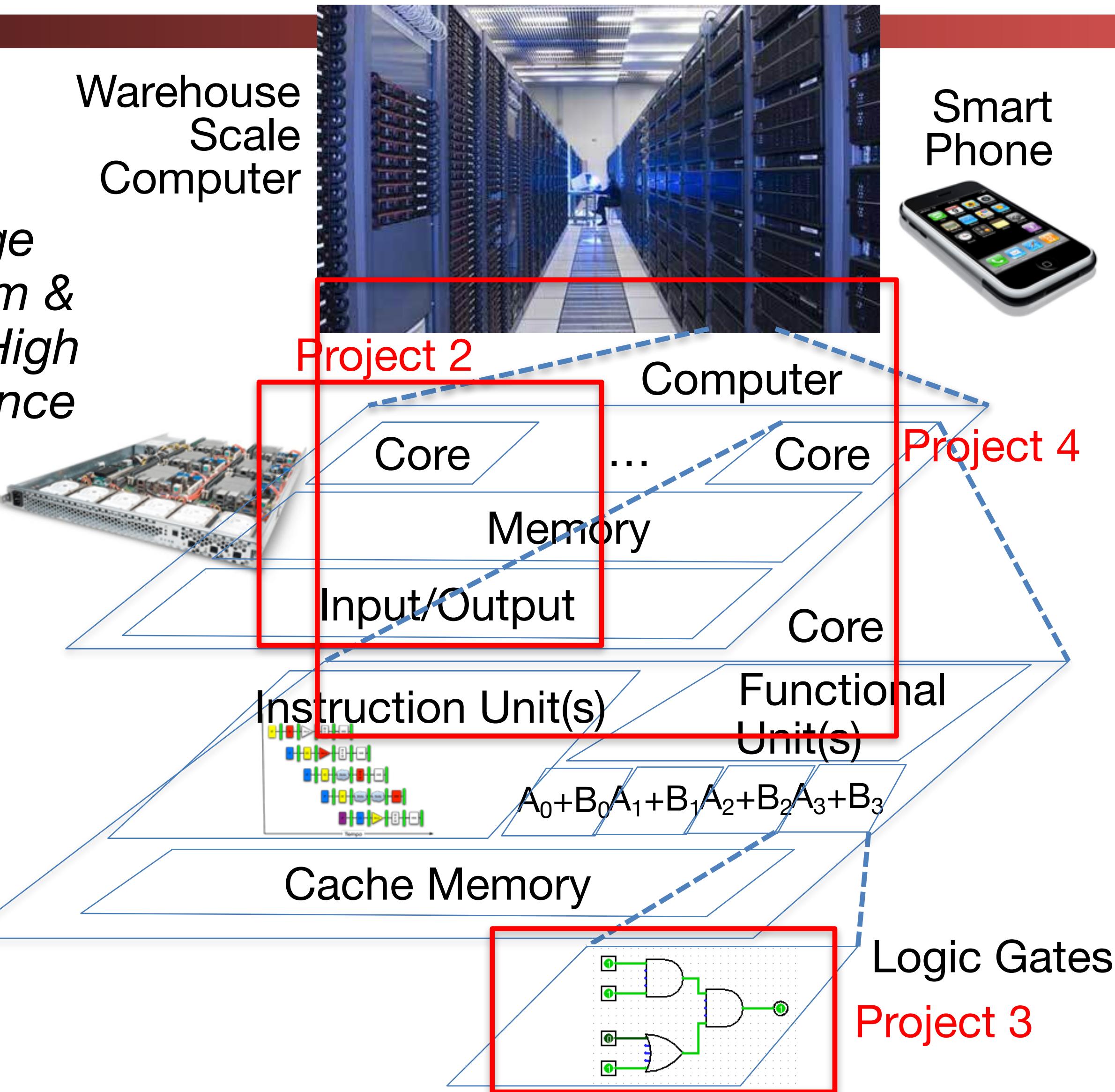
## Hardware

Computer Science 61C

Wawrzynek & Weaver

- Parallel Requests  
Assigned to computer  
e.g., Search “@ncweaver”
- Parallel Threads  
Assigned to core  
e.g., Lookup, Ads
- Parallel Instructions  
>1 instruction @ one time  
e.g., 5 pipelined instructions
- Parallel Data  
>1 data item @ one time  
e.g., Add of 4 pairs of words
- Hardware descriptions  
All gates functioning in parallel at same time
- Programming Languages

*Leverage  
Parallelism &  
Achieve High  
Performance*



# Six Great Ideas in Computer Architecture

- Design for Moore's Law:
  - Multicore & Thread-Level Parallelism (Multicore, Parallelism, OpenMP, Project #4)
- Abstraction to Simplify Design
  - And when in doubt, add another layer of abstraction
- Make the Common Case Fast
  - The design philosophy behind RISC
- Dependability via Redundancy
  - ECC, RAID, and clusters of systems
- Memory Hierarchy
  - Caches, Caches, and More Caches...
- Performance via Parallelism/Pipelining/Prediction

# The Five Kinds of Parallelism

- Request Level Parallelism
  - Google & warehouse scale computers
- Instruction Level Parallelism
  - Pipelining & 152/252 topics: Superscalar, out-of-order execution, branch prediction
- (Fine Grain) Data Level Parallelism:
  - SIMD instructions, graphics cards
- (Course Grain) Data/Task Level Parallelism:
  - Map/Reduce: Hadoop and Spark
  - Or Nick pegging the CPU on The Beast
- Thread Level Parallelism:
  - Multicore systems, OpenMP, Go

# JohnW's First Computer: 1973, IBM System/360

- Most famous of classic “mainframe” computers
- First computer to separate “architecture” from “implementation”:
  - Birth of the ISA (instruction set architecture)
  - Allows backward compatibility of code
- Jobs “submitted” on punched cards and output on printer
- 16.6 million instructions per second!
- Typically, 1024KB main memory
- Programmed with Fortran

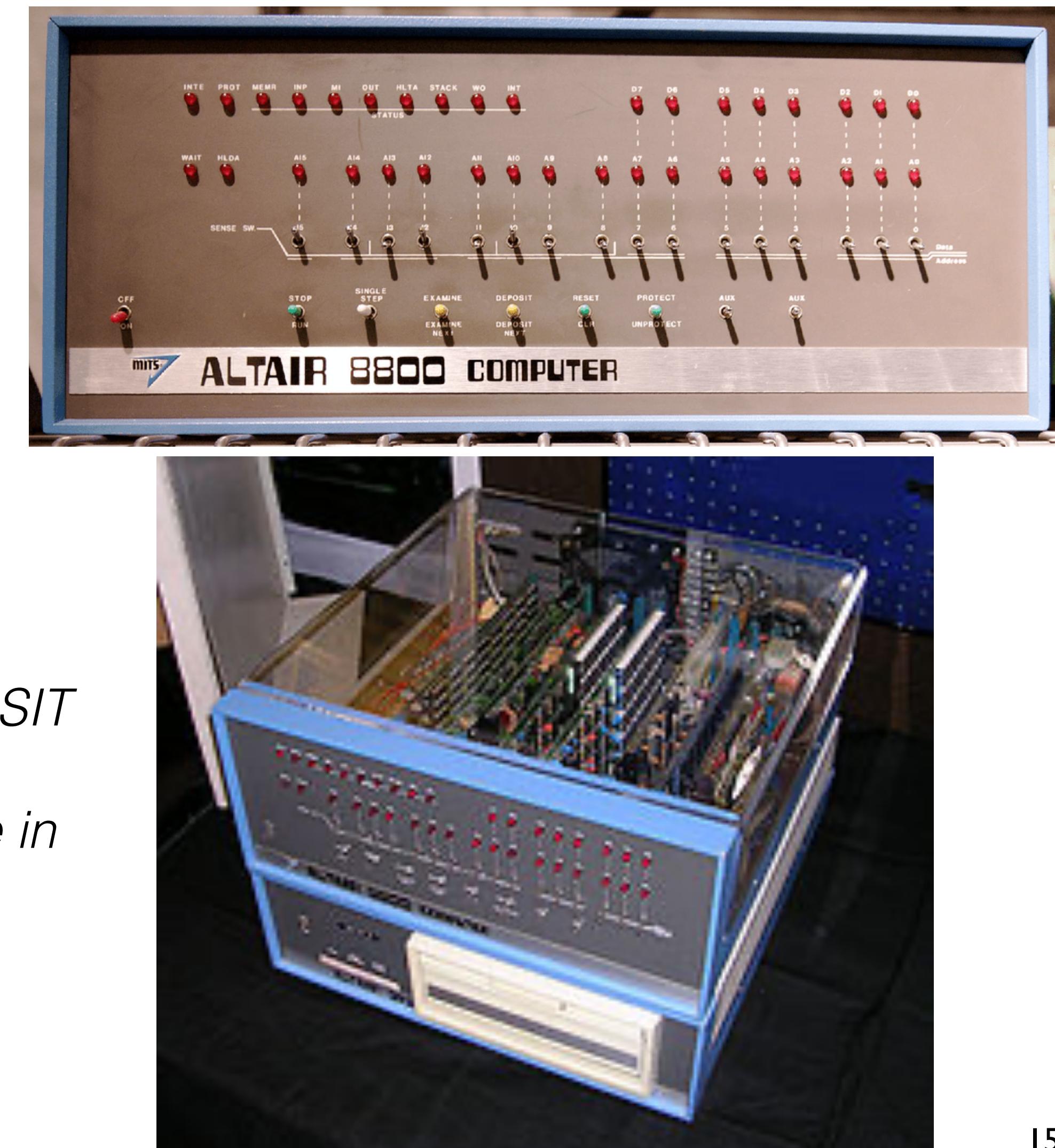


# JohnW's 2nd Computer: 1978, MITS Altair 8800

- Beginning of personal computers
- Designed around Intel 8080 CPU chip:
  - 2 MHz, with common instructions using 4, 5, 7, 10, or 11 cycles
- PC Cards plugged into S-100 bus - became industry standard bus
- Kit price: \$439, assembled \$621, memory and I/O cards sold separately:
  - 4096 word memory card \$264/\$338

*Initially, programming was via the front panel - toggling switches corresponding to the desired opcode in binary, then used the 'DEPOSIT NEXT' switch to load that instruction into the next address of the machine's memory. Repeat until all the opcodes of the program were in place. The only output from the programs was the patterns of LEDs.*

- Eventually tape readers, floppy disk, and the BASIC programming language introduced



# Nick's First Computer: 1980, Apple II+

- MOS 6502 processor:
  - 8b processor with a 16b address bus
- 16kB of RAM
  - Extended it to 32kB with a memory card
- Floppy drive: 140kB disks
- 280x192, 6 color graphics in "Hi Resolution" Mode
- ~\$4000 in today's money!
- Languages supported included BASIC and Logo
  - Logo is remarkably subtle and cool, its remarkably similar to scheme under the hood



# Nick's Freshman Year

## Computer: 1991

- 25MHz 68040, 32b CISC processor, 6 stage pipeline ***with floating point!***
  - Whopping 4kB I\$ and 4kB D\$
- 20 MB of memory
  - I expanded it from the original 8 MB, it cost me a ***fortune!***
- 1120x832 2-bit grayscale display
  - But I'd rather have a sharp grayscale display than an ugly color display at the time
- ~100 MB hard drive, 2.88MB floppy drive
  - About \$9k in today's dollars...  
Other kids got cars for HS graduation, I got a computer



# But That Was Sufficient For 60B...

- The predecessor to current 61C
  - Added more learning of C
  - Didn't include parallel programming, data-center stuff, RAID, etc...
- But otherwise, the contents looked rather familiar
  - Basically include caches, I/O, virtual memory, assembly (MIPS not RISC-V but well, same diff), C
  - "All RISCs are the same except for one or two 'seemed like a good idea at the time' decisions"
    - RISC-V is just so new we don't know what those are (apart from not building in PAC from the start)
    - MIPS has a software-managed TLB (more and more expensive page faults) and a "branch delay slot"
  - But with a bit more handholding on learning C and assembly because it was the second semester class

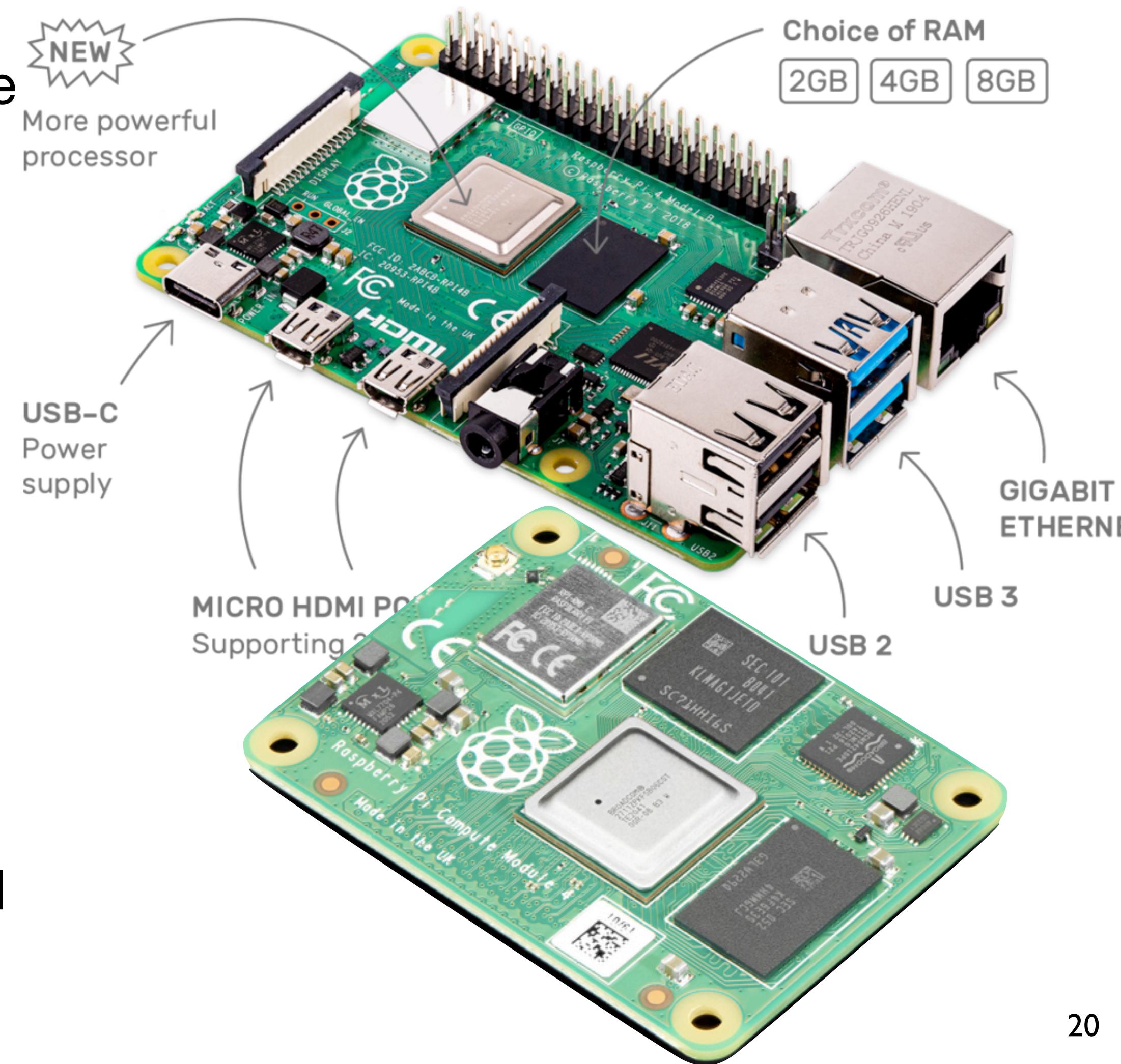
# One of Nick's Research Computers...

- Yeup, an RPi3
  - ~50x single-thread performance
  - ~200x multi-threaded performance
  - 50x the RAM
- Only difference from what you might have:
  - I stuck in a 128GB SDCARD: 1000x the storage!



# And now I'm loving the RPi 4...

- Take the RPi3 and turn it up to 11+
  - ~2x the single threaded & multithreaded performance
  - More RAM options: up to 8 GB RAM
  - Faster I/O with 2x USB3 ports, Gigabit Ethernet
- And the compute module...
  - Remove the physical connectors: Instead provide 2x 100 pin connectors on the other side
  - Get the raw PCIe x1 rather than USB3
  - And put it on a card to stick in larger systems
  - \$45 for a version with 2GB RAM, 16 GB storage, and WiFi/Bluetooth



# And Nick's Beast!!!

- 12 core Ryzen-9 processor
  - 3.8 GHz, 2 threads/core
  - 32KiB I\$, 32KiB D\$, 512KiB L2\$/core
  - 64 MB shared L3 cache
  - 32 GB RAM, 1 TB SSD
  - 10 TFLOP single precision GPU
  - Driving a 3840 x 2160 pixel, 43" 'monitor'
- A close to spare-no-expense desktop...
  - And ***still*** costs significantly less than my Apple ][+ I had as a kid!



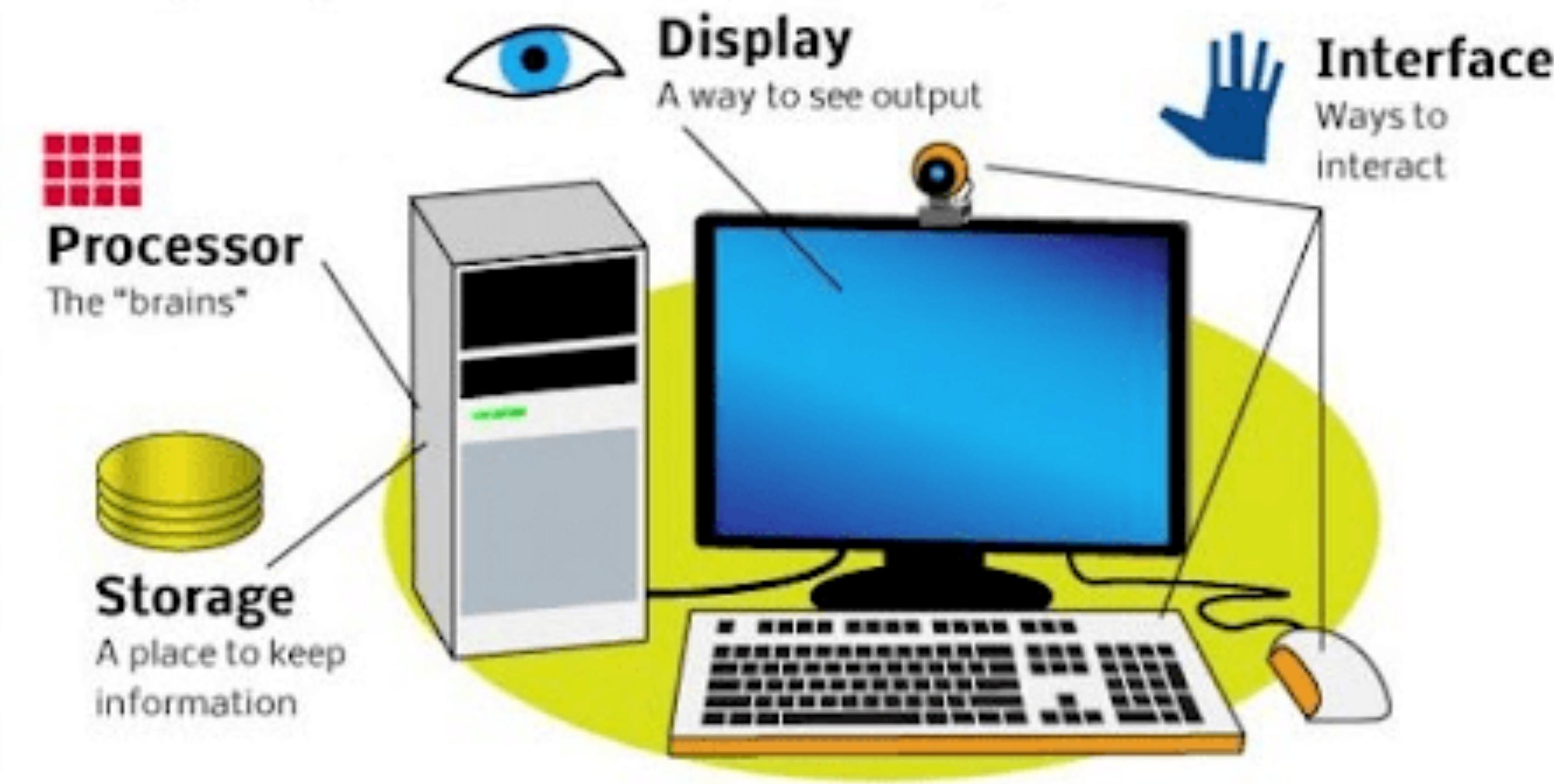
# Your Computer is Going Away

Computer Science 61C

Wawrynek &amp; Weaver

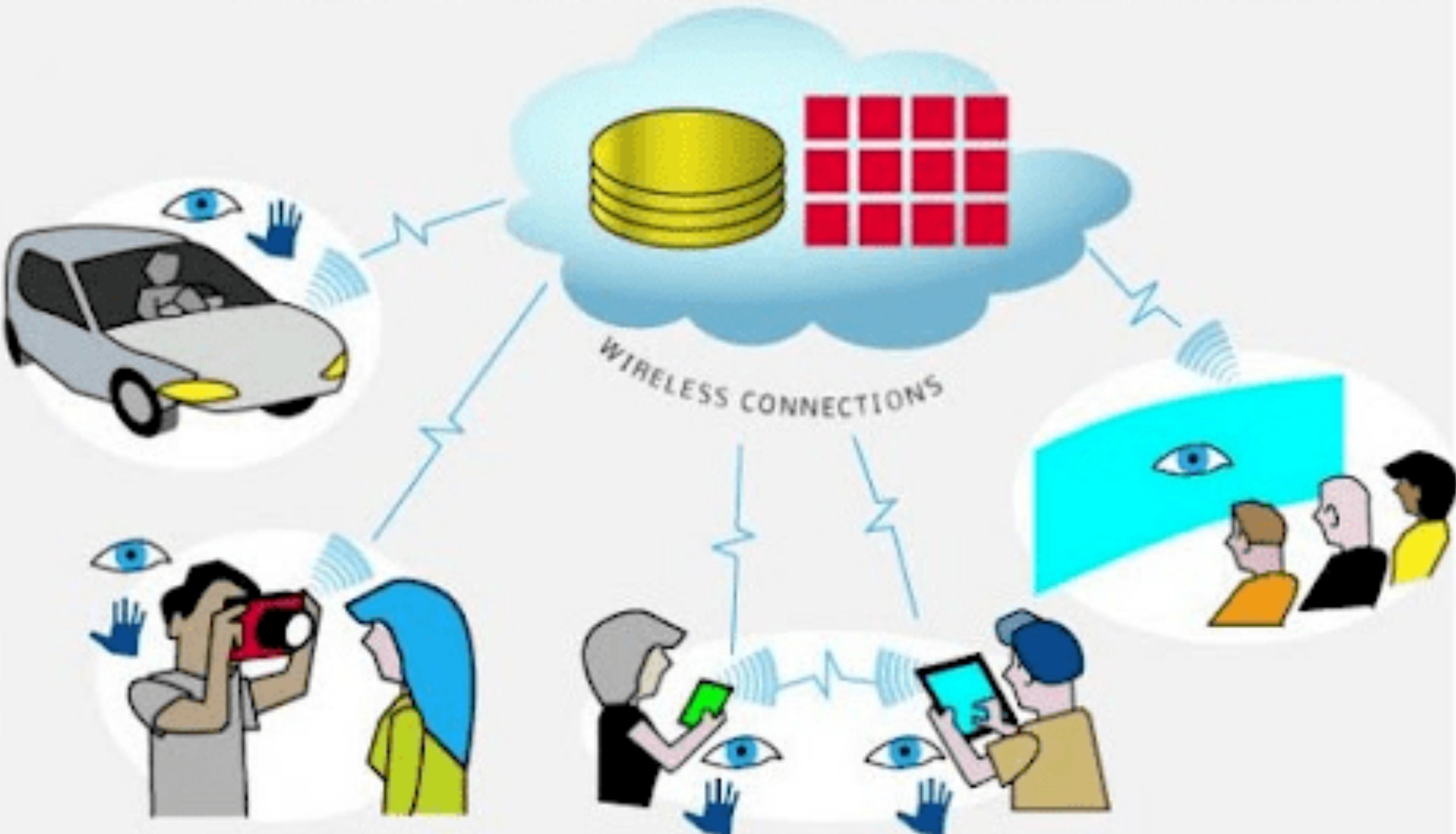
Soon, your smartphone, TiVo, laptop, television -- all of your current gadgets -- will be obsolete. The future is "ubiquitous computing." Think Google Docs, but on every screen you use, running every program you use -- every device drawing from the same pool of data and processing power. Here's how we got to this point.

Currently, all digital devices include these four components:



# 2010

In the emerging **ubiquitous computing era**, every device accesses all its data and processing power from the Internet “**cloud**.” This means the devices themselves need not have any on-board processing or data storage, reducing their price and increasing their deployment. Additionally, the interface will move beyond the mouse and keyboard into task specific form-factors. Computers will be everywhere, but you won’t even notice them.



# But A Dissent From The Cloudy Future...

- The “Cloud” is really just a name for someone else’s computer...
- And you are therefore trusting them to do right by your data...
- It could be because you pay them
  - Amazon EC2
- It could be because you bought “ohh shiny”
  - Apple
- It could be because they are ~~selling your soul~~ using your data for their own profit
  - Google
- And its not like the "cloud" is cheaper! The computer in your hand is obscene by the standards of a decade ago
  - And wireless communication is ***not getting better!***  
4G is already at the "Shannon's Limit" for all practical purposes

# Nick's Happy Prediction: The Fabrication Revolution...

- We've seen incredibly powerful and cheap compute modules with built-in networking
  - RPi 4: \$35-75
  - RPi CM4: \$25-90 depending on capacity
    - 1, 2, 4, 8 GB RAM
    - 0, 8, 16, 32 GB Solid state storage
    - WiFi/Bluetooth optional
  - Raspberry Pi Zero 2 W: \$15, 1 GHz quad core 64b CPU, 512 MB RAM, Mini HDMI, WiFi/Bluetooth, I/O, SD-card slot & camera interface
- Amdahl's Law applies to cost optimization...
  - If you have a \$15 RPi Zero 2 W + SD Card to drive your product...
    - Oh, which runs on 5V and is even kind and gives you a 3.3V/1A power output
    - The rest of the cost has to be pretty damn low before its worth replacing with something cheaper
- So the compute & communication to make a device is effectively **free**:
  - When in doubt, you can ***throw a computer at the problem***

# But It's Not Just The Compute & Control...

- 3D printers, laser cutters, C&C Machines all make prototyping stuff cheap
  - And direct paths to go from 1 to 10 to 1000 to 100,000 thingies
- And logistics
  - Time from manufacturer to me doesn't actually care where I am in the US:  
I could run a design business from a shack in the woods
  - And the pandemic has turned a lot of business trips into "couldn't this just be a Zoom call?"
  - And direct to consumer marketing



**FedEx®**  
**KICKSTARTER**

# You Can Even Do Custom *Compute Platforms*...

Computer Science 61C

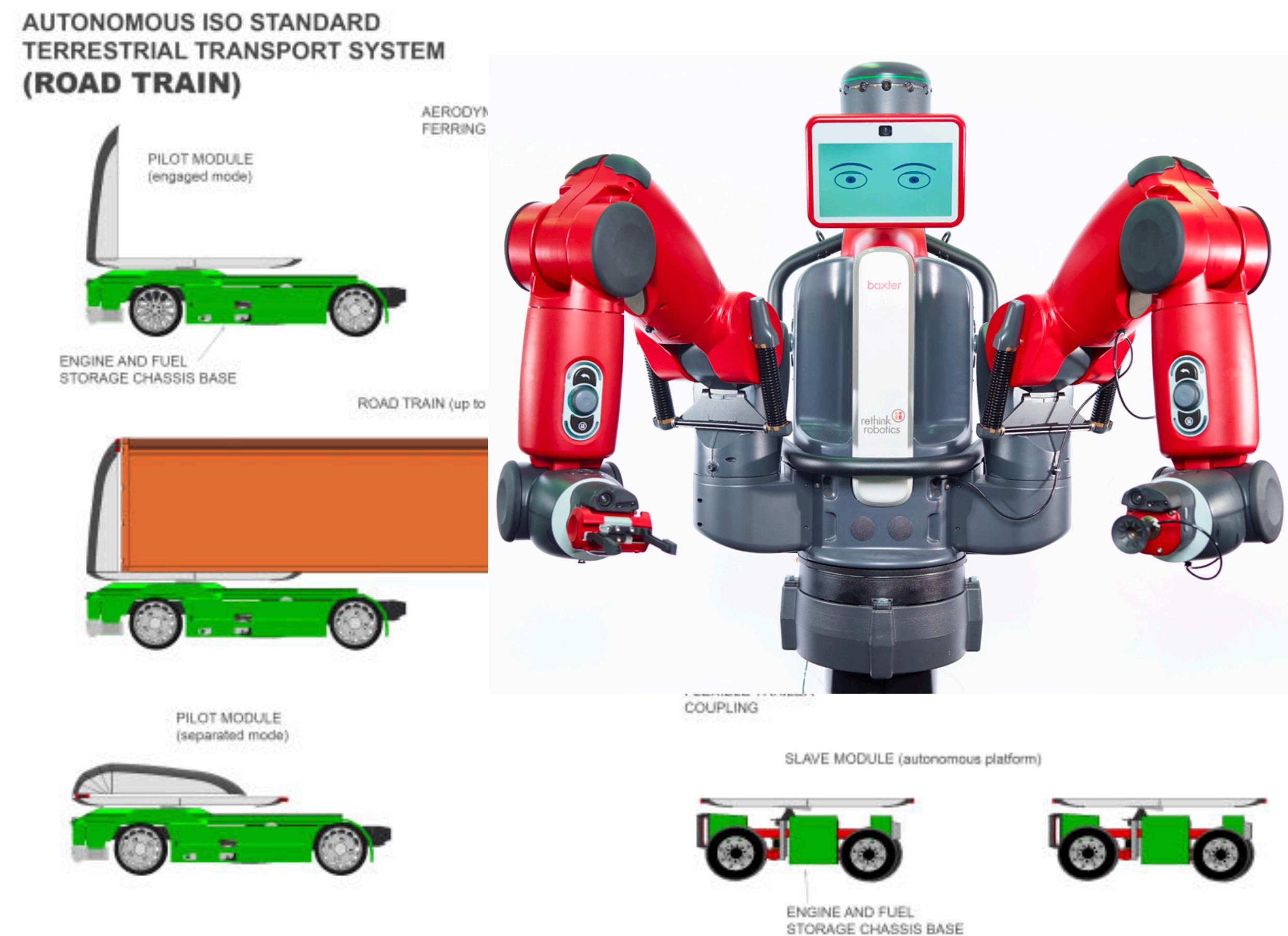
Wawrzynek & Weaver

- Nick's drone control board: Cost: ~\$4000 for the first 5
  - Should be <\$100/each for 500 fully assembled, tested, & delivered
- Specifications:
  - ST H7 embedded CPU for flight control duties and to drive 8 PWM/servo outputs
    - Keeping a quad-copter drone in the air is a hard realtime problem: Standard OSs are great unless you need a <10ms update loop
  - GPS, accelerometer, compass, barometer
  - Backside slot for CM4
    - 2 1080p camera inputs
  - Coral AI accelerator
  - Frontside slot for cellphone/modem
  - Power distribution accepting unregulated 7-36V input: provides both 5V/4A and 3.3V/4A
- This is incredibly powerful
  - For slightly more than "hobby" money!  
Certainly pocket lint for a trivially funded startup



# Nick's Gloomy Prediction: Automation and Its Discontents...

- We are getting damn close to the autonomous long-haul truck
  - If it costs \$100K to automate a semi-truck it will pay for itself in <2 years!
- And a lot of jobs with robots
  - EG, the \$20k Baxter human-safe robot:  
One robot only needs to replace .2 humans to pay for itself in 2 years
    - This particular one failed because it was too early...  
But look at Boston Dynamics or Amazon's warehouse-worker bots
- Plus all the AI-related dislocation
  - Automate out the "paper pushing" jobs
- Scary Prediction:  
20 years from now we will have  
>20% unemployment
  - Yes, baseline unemployment 20 years out == COVID crisis today!



Design from **Logan**, © 2017 21st Century Fox

# And Now: Your Future Classes...

- CS61C is a prerequisite to most/all "system" classes here at Berkeley
  - And some thoughts about them...

# CS 161: Computer Security

- CS161 is the only other ***full stack*** course after 61C
  - Security touches basically everything in computer science:  
So welcome to another speedrun class
- We covered some of the critical ***mechanisms*** needed for security
  - Paging/Virtual memory enforces ***isolation***:  
Prevents processes from interfering with each other
  - Attacks exploit the ***call frame***:  
Buffer overflow attacks not just crashing programs but overwriting the return address or other such information
- Security and hardware also have interesting interactions
  - One example: ***Rowhammer***

# Should You Take CS161 First Or Last?

- If we switch to a 5-required class format, something like CS161 will have to be in it
  - Security and robustness is a pervasive problem
  - We've also scaled the class to handle hundreds of students a semester
- But this class is the one other speed-run class
  - EG, we lightly cover a good half of 168 in just two weeks!
- So if you liked the 61C speedrun, take 161 first
  - Especially if you like my teaching style, I'm teaching it next semester...
- But if you hated the speed run aspects of 61C, take 161 LAST!

# CS 162:

# Operating Systems

- Operating Systems is all about several big ideas:
  - Managing concurrency/multiprocessors
    - This enables parallelism
  - Isolation through Virtual Memory
  - I/O & Interrupts
- Builds very strongly on what you've already learned
  - Just far more advanced than what you've already done:  
Focuses on concurrency, virtual memory & isolation, filesystems, and I/O
  - Class successfully scaled so far...

# A 162 Project: Caches in the Filesystem

- In 162 you improve the Pintos filesystem
- One of the big aspect is adding caching
  - The default system doesn't cache reads or writes, so this *hurts*
  - This touches on I/O (you're writing to disk) caching strategies (how you allocate blocks, write-back implementation, and other areas), etc
- 162 projects are a *beast*
  - Large, group of 4 C programming projects
    - Use Project 4 as a test. Project 4, especially the non-speedup parts, is really 162 prep: If you liked project 4 you'll do fine on 162's projects. If you didn't, avoid 162
    - And only take the class if you already have a good group!
    - But it is a *good pain*: real feeling of satisfaction when you are done!

# CS 164: Compilers...

- In 61C we introduced the CALL flow:  
Compiler  
Assembler  
Linker  
Loader
- We saw how to do the assembler/linker/loader
  - They are fairly simple
  - We defined a calling convention
    - So how we can make sure functions can call each other on the assembly level
    - 164 completes that flow...

# 164 Project: Building a compiler

- The compiler itself is broken up into pieces
  - Lexer: Converts text into ***tokens***
  - Parser: Determines the ***structure*** of the program
  - Semantic Analyzer: What does the program ***mean?***
  - Optimizer: Make the program ***better***
  - Code Generator: Output the assembly code (or C, depending, because C is portable assembly language anyway)
- The last part is very much a followup to 61C:  
Rather than writing assembly, you are writing the program  
that writes the assembly version of the program

# CS 168: Networking

- How do we turn the network I/O into something usable
  - We have a unreliable, "best effort" system
  - Lets make something useful
  - And build on top of that...
- Also the foundation for the warehouse scale computer
  - The ability to tie together multiple systems into a cohesive whole
  - Unfortunately rarely taught... But it may be offered next fall!

# CS 169: Software Engineering

- How to build bigger systems
  - Once you get  $>4$  people working on a project, things get complicated
  - This class is about addressing those problems
- Very project centric
  - Large team project for a real world system
- Projects can often be rather web centric:
  - Real web applications are distributed client/server beasts of complexity.  
So how do you tame that complexity?

# CS 184: Graphics

- How do we go from some notion of geometry to bits on the screen...
  - And other related issues
  - Lots of programming in C++
  - Modern graphics is enabled by GPUs
    - Ability to do massive amounts of parallelism
    - Ability to efficiently use massive memory bandwidth by thread-switching

# CS 186:

# Databases

- How to actually manage the data on these systems?
- We've got amazing computers
  - Quad CPU, gazillion core beasts
  - A ton of memory
  - Huge amounts of disk
- How can we get the most out of them?
  - Databases are an incredibly powerful primitive
  - And built well, they need to understand the hardware they are running on

# More on 186...

- Databases are *insanely* powerful tools...
  - I've only done one paper where I needed to use Hadoop for massive map/reduce parallelism
    - A billion+ records that needed significant computational analysis
  - I've done **dozens** of papers where "throw everything in the big A)(#@\*)(# Postgres database" was step 1...
    - Our research group even custom ordered a beefy server we called **mammoth**, just to process millions of records for a paper
- I've heard that it can be somewhat boring...
  - But **damn** it is useful. I wish I took it as an undergrad
  - "If my data is structured and anywhere between 1 MB and a few TB, its going into postgres"

# CS188 and CS189: AI & Machine Learning

- I personally like dunking on AI/Machine Learning at times...
  - Mostly because I don't understand how it works
    - But then again, nobody does!
    - But it really has become an incredibly powerful tool
- The new driver is not the algorithms, but the computers!
  - Many ML algorithms **vectorize** extremely well (for every element do X style parallelism):  
Acts as a classic SIMD parallel computation
  - The graphics cards now have an **obscene** amount of SIMD computation:  
**trillions** of operations per second
    - In the end, it is dense matrix multiply
      - As opposed to the classic supercomputer problems which are sparse matrix multiply

# EECS 151:

# Introduction to Digital Design & Integrated Circuits

- Just call it "Project 3 on Steroids!"
  - Building systems from the gates up
  - Two versions of the lab
    - ASIC: Build a RISC-V processor core for fabrication as a custom chip... and only test it in simulation
    - FPGA: Build a RISC-V processor core, and have it ***run on a real FPGA!***
    - Choose to take one lab or both!
  - Also instantly employable skills:
    - Apple is throwing a ton of \$ around at good digital designers:  
If you want to work for Apple, take 151 with FPGAs and then the 151 lab-only for ASICs

# CS 152: Computer Architecture

- And how modern CPUs actually work and are designed ...
  - Want to understand how you can actually make 100-deep out of order reorder buffers on 14 stage pipelines with vector coprocessors?
  - Or how graphics cards are able to compute 100x more than a CPU?
  - This class is for you!