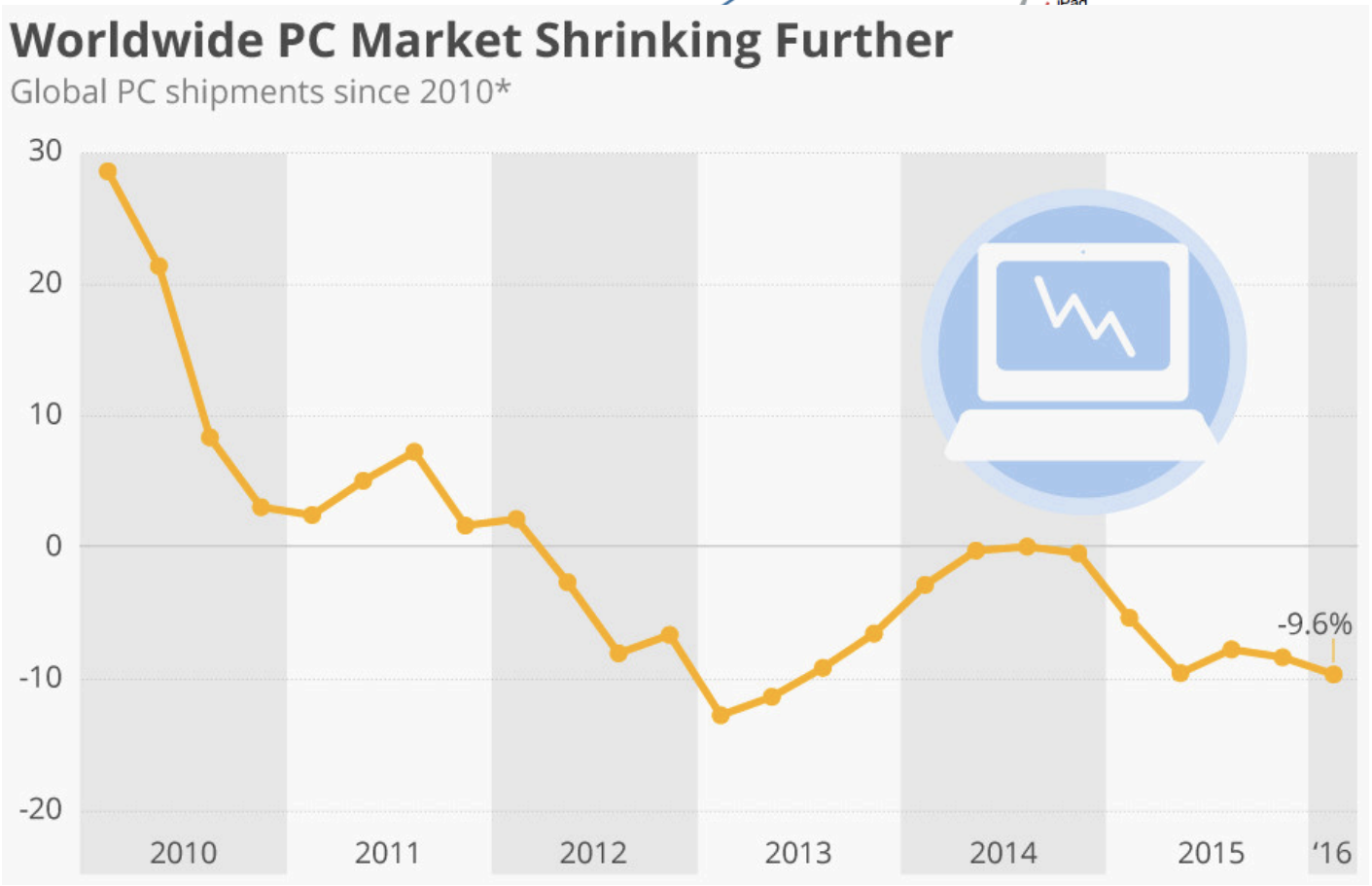
So What's Next?

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

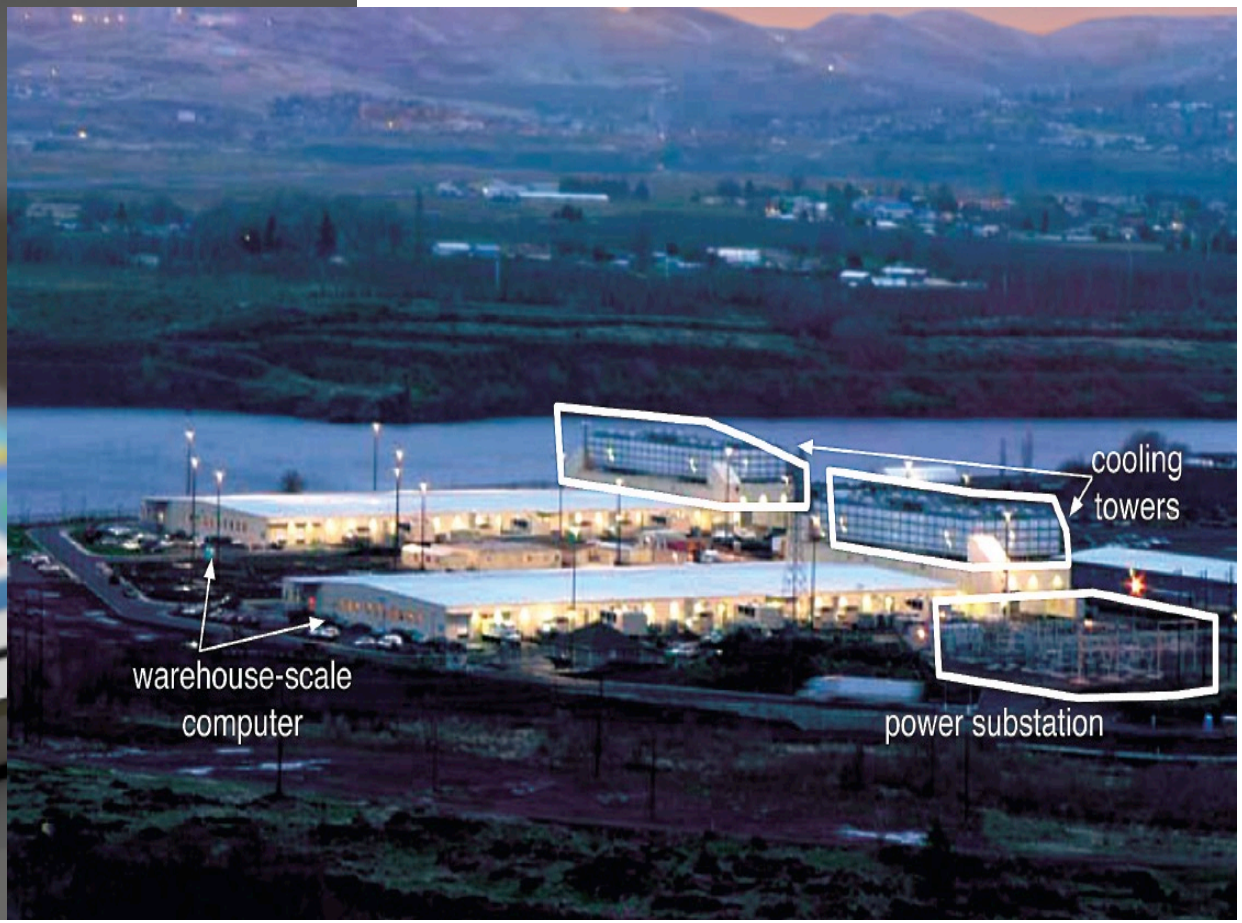
Where Do We Go From Here?

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

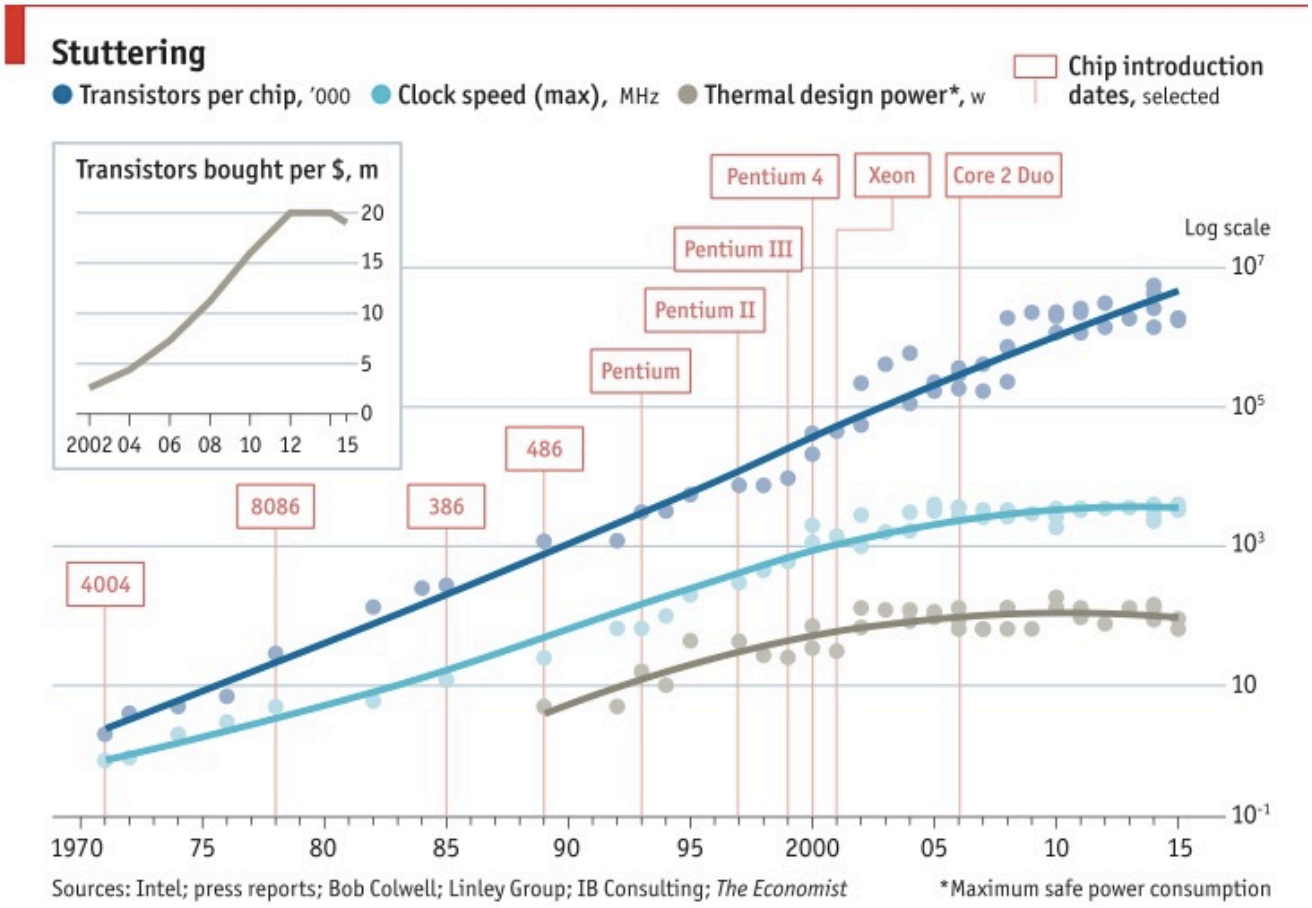

Units Shipped Per Year
1000000
© asymco.com
100000
PC
Android
iPhone
iPad


Worldwide PC Market Shrinking Further
Global PC shipments since 2010*
30
20
10
0
-10
-20
-9.6%
2010
2011
2012
2013
2014
2015
'16

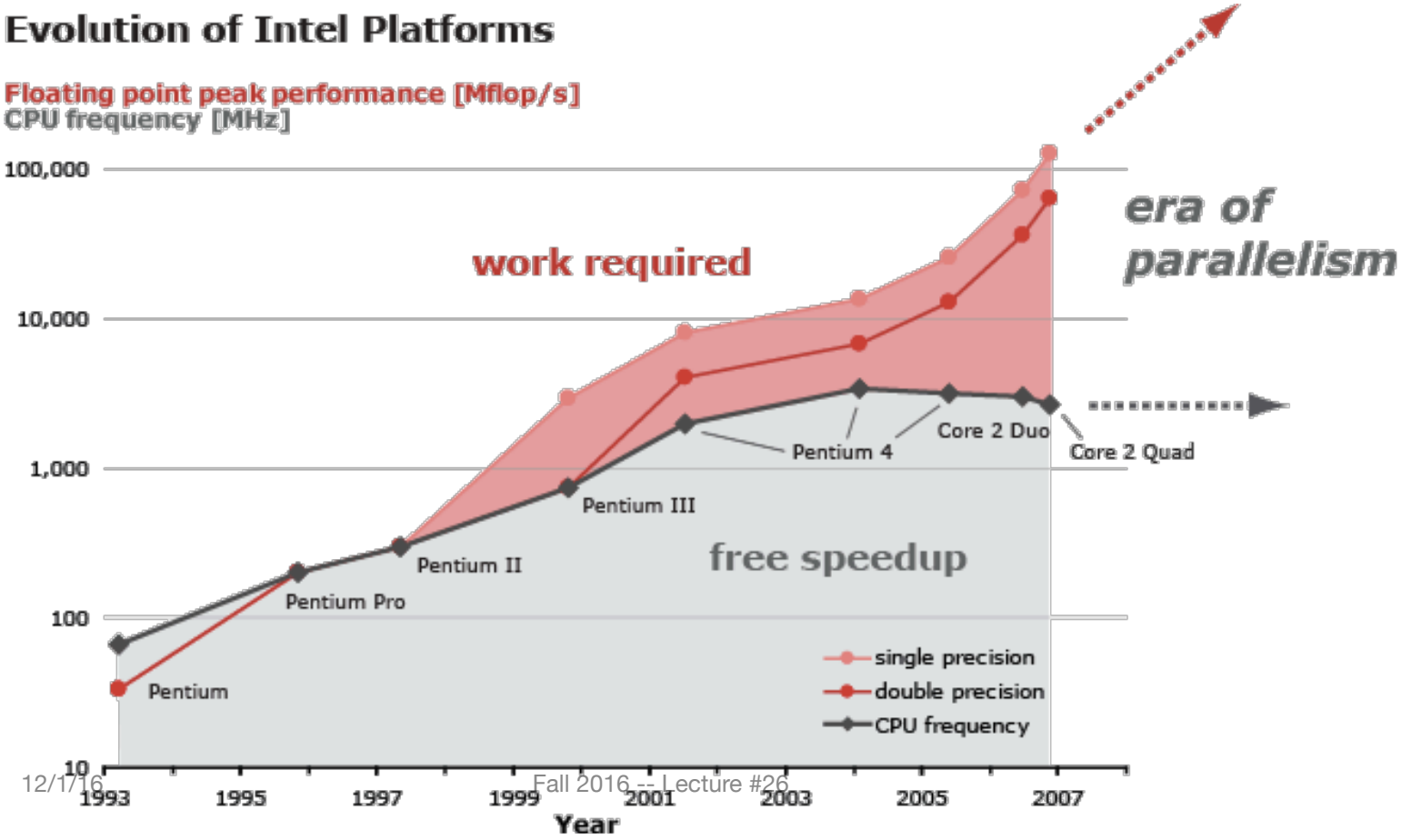
Current 61C:
The Same Concepts Over a Mass Scale



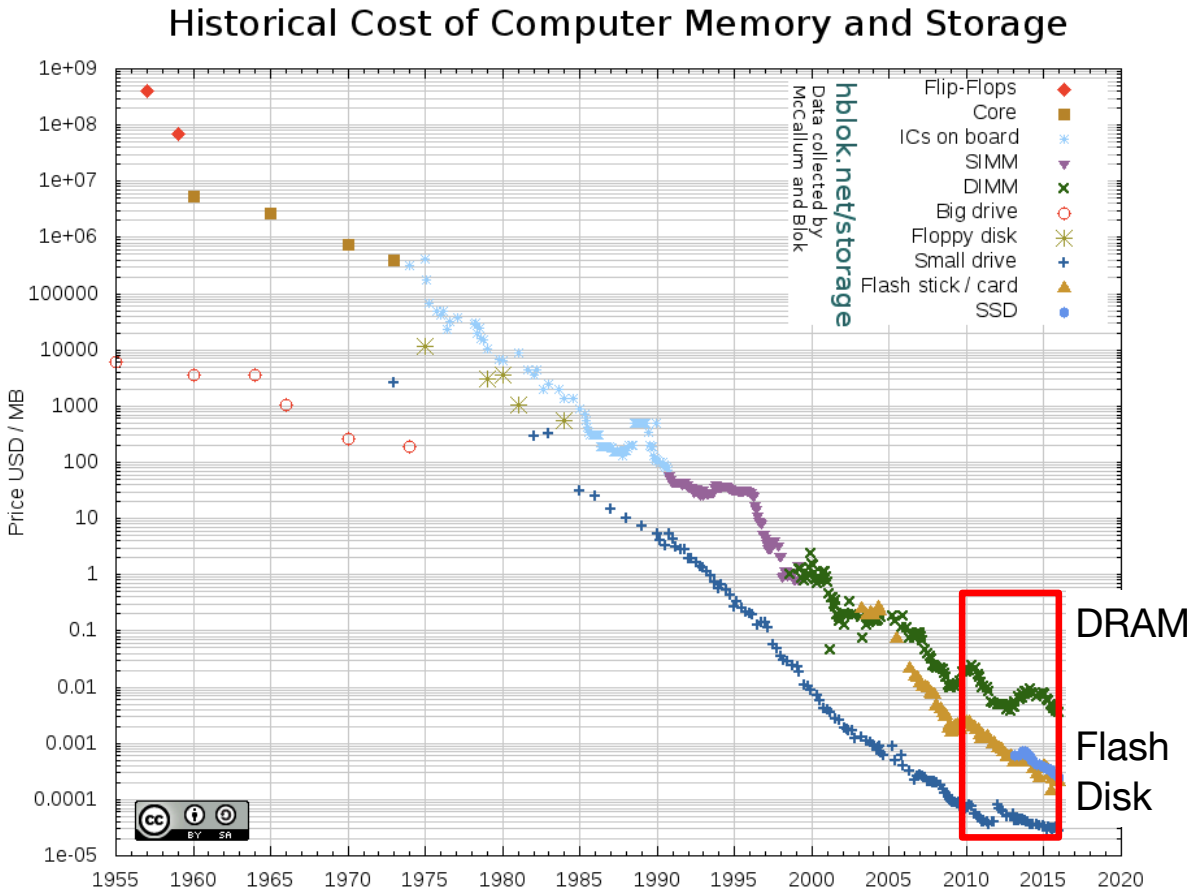
All Have Hit the Single-Thread Brick Wall



Leaving Parallelism the ***only*** way to improve throughput



But Things Are Still Getting Cheaper & Better



New-School Machine Structures

Software *Hardware*

- 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

Warehouse Scale Computer

Smart Phone

Project 2

Computer

Core … Core

Project 4 & Project 5

Memory

Input/Output

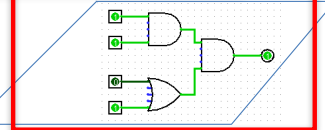
Core

Instruction Unit(s)

Functional Unit(s)

$A_0+B_0A_1+B_1A_2+B_2A_3+B_3$

Cache Memory



Logic Gates

Project 3

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
- Thread Level Parallelism:
  - Multicore systems, OpenMP, Go

Nick's First Computer: 1980, Apple][+

- 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
- ~$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

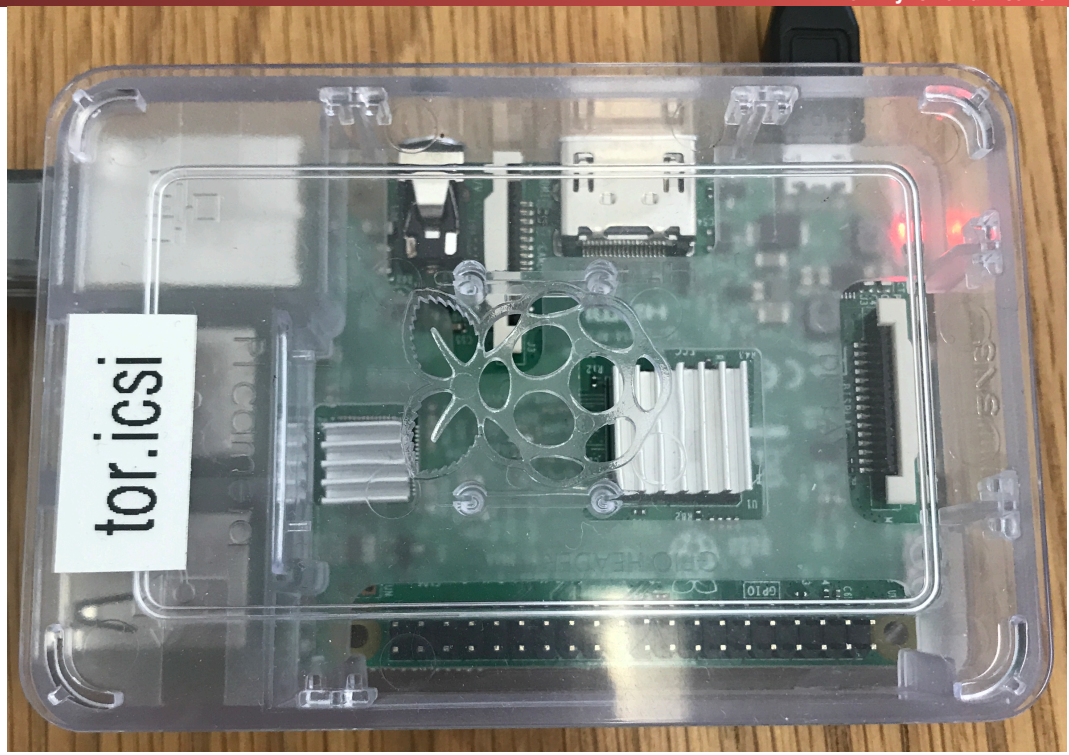
- 25MHz 68040, 32b processor
- 20 MB of memory
  - I expanded it from the original 8 MB, it cost me a ***fortune!***
- 1120x832 2-bit grayscale display
  - But I'd rather have a sharp grayscale display than an ugly color display at the time
- ~100 MB hard drive, 2.88MB floppy drive
  - About $9k in today's dollars
  - And it was evacuated from the Oakland Hills fire

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, C
  - But with a bit more handholding on learning C and assembly because it was the second semester class

One of Nick's Research Computers…

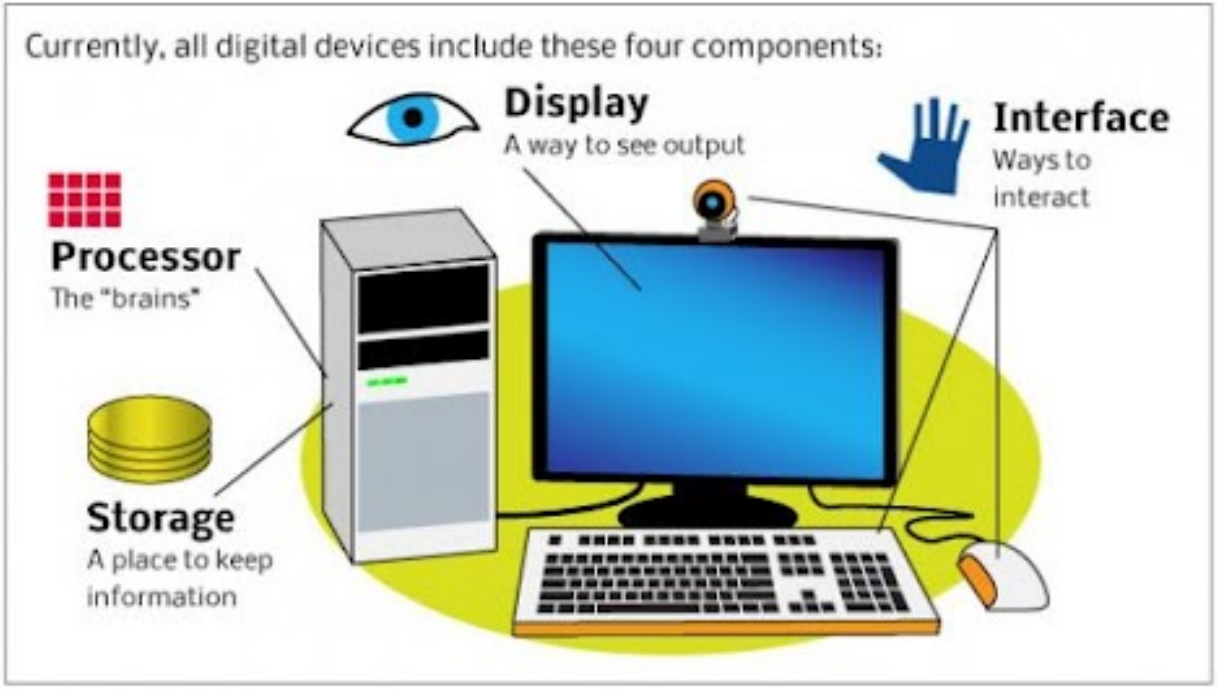
- Yeup, an RPi3
  - ~50x single-thread performance
  - ~200x multi-threaded performance
  - 50x the RAM
- Only difference from what you might have:
  - I stuck in a 128GB SDCARD



TechNewsDaily www.technewsdaily.com

Your Computer is Going Away

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.



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

Nick's Happy Prediction: The Fabrication Revolution…

- We've seen incredibly powerful and cheap compute modules with built-in networking
  - RPi 3: $35
  - RPi-0: $10
- Amdahl's Law applies to cost optimization…
  - If you have a $15 RPi 0 + SD Card to drive your product…
  - 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…

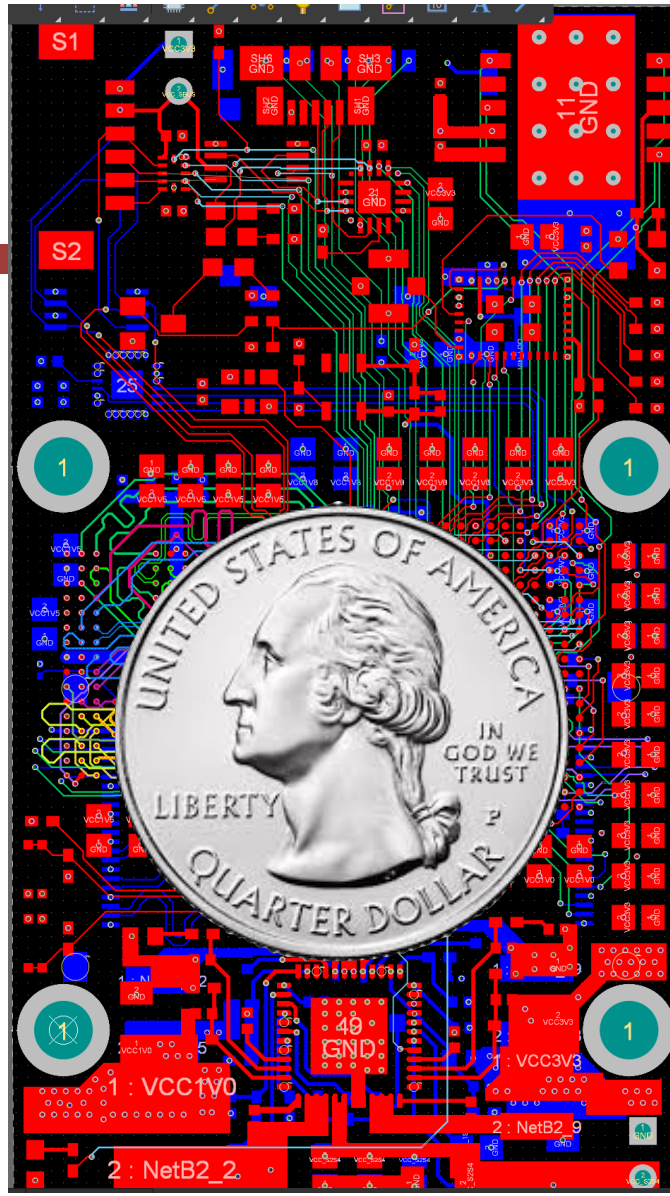
- 3D printers, laser cutters, C&C Machines all make prototyping stuff cheap
  - And direct paths to go from 1 to 10 to 1000 to 100,000 thingies
- And logistics
  - Time from manufacturer to me doesn't actually care where I am in the US:
    I could run a design business from a shack in the woods
- And direct to consumer marketing

FedEx®

KICKSTARTER

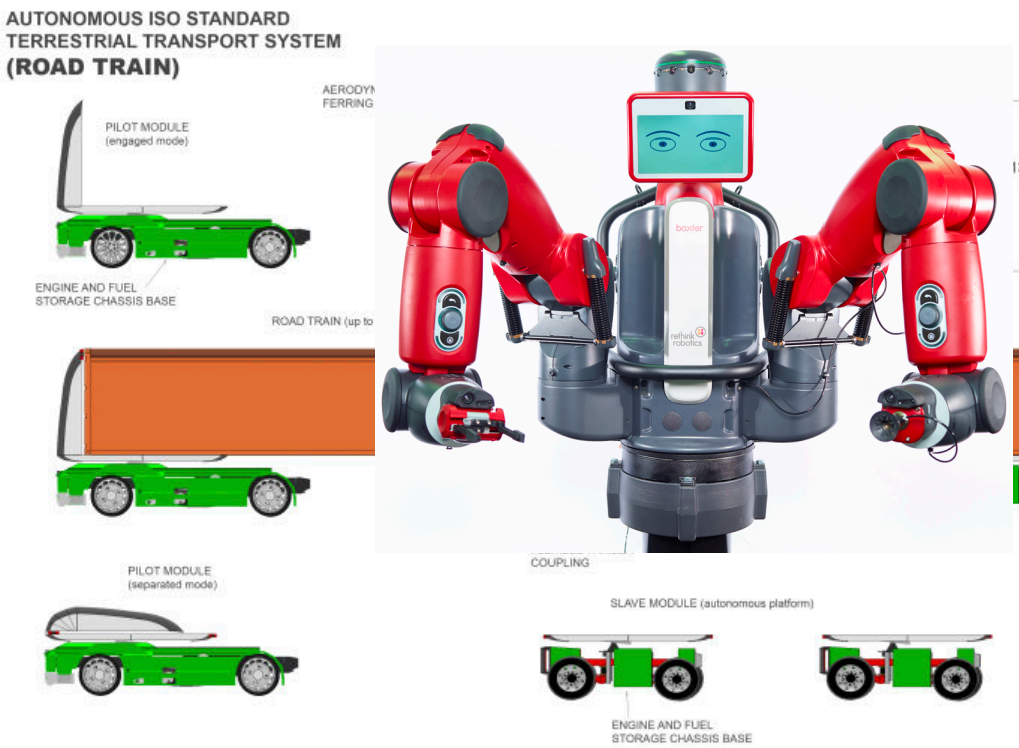
You Can Even Do Custom ***Computers***...

- Nick's drone control board:
  In the process of sending it out for fabrication
- Cost: ~$8000 for the first 5
  - Should be <$300/each for 500
  - Includes GPS, 2x accelerometers, 1/2 GB DRAM, WiFi, BlueTooth, 2x 1080p/30fps camera interfaces, SD card, dual core 500 MHz ARM & decent sized FPGA
- 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
- Plus all the AI-related dislocation
  - Automate out the "paper pushing" jobs
- Scary Prediction: 20 years from now we will have >20% unemployment



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

Announcements!

- Homework 5 due tomorrow
- Project 5 due monday
- HKN course evaluations...
  - If >90% respond, ***everyone*** will get extra EPA points:
    (and EPA points are not used in calculating the curve)
- EPA self-evaluation survey out soon

And Now:
Your Future Classes...

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

CS161:
Computer Security

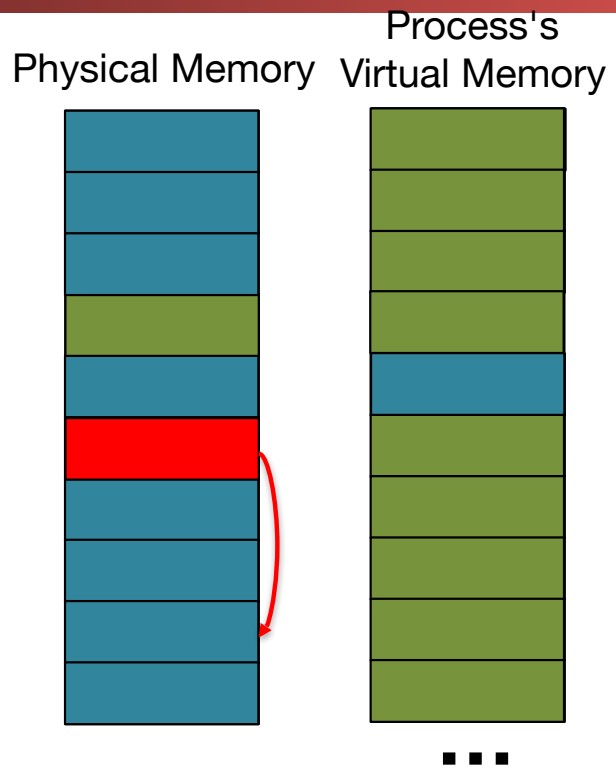
- CS161 is the only other ***full stack*** course after 61C
  - Security touches basically everything in computer science
- We covered some of the critical ***mechanisms*** needed for security
  - Paging/Virtual memory enforces ***isolation***:
    Prevents processes from interfering with each other
  - 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***

The Ultimate Page-Table Trick: Rowhammer

- An ***unspeakably cool*** security vulnerability…
- DRAM (unless you pay for error correcting (ECC) memory) is actually unreliable
  - Can repeatedly read/write the same location ("hammer the row" and eventually cause an error in ***some physically distinct memory location***
- Can tell the OS "I want to map this same block of memory at multiple addresses in my process…"
  - Which creates additional page table entries, lots of them. Lots and lots of them. Lots and lots and lots and lots and lots of them...
- Enter ***Rowhammer***
  - It seems all vunerabilities get named now, but this one is cool enough to deserve a name!
- Touches on virtual memory, hardware failures, and breaks security

How RowHammer Works

- Step 1: Allocate a single page of memory
- Step 2: Make the OS make a gazillion page-table entries pointing to the same page
- Step 3: Hammer the DRAM until one of those entries gets corrupted
  - Now causes that memory page to point to a set of page table entries instead
- Step 4: ***Profit***
  - Well, the ability to read and write to any physical address in the system, same difference



CS162:
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 that what you've already done:
    Focuses on concurrency, virtual memory & isolation, filesystems, and I/O

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

CS164: 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, 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

CS168: 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

CS186: Databases

- How to actually manage the data on theses 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

CS188: AI

- 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

EECS 151

- Just call it "Project 3 on Steroids!"
  - Building systems from the gates up

CS 152

- And how modern CPUs actually work...
  - 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!