INTRO TO CS2

LECTURE 1-1A

JIM FIX, REED COLLEGE CS2-F20

WELCOME TO CS2!

- **Today's plan**
  - Go over the syllabus at **https://jimfix.github.io/csci221**
    - ➡Topics, themes, goals, context, assignments
  - Introduction to C programming
- **Tomorrow's plan**
  - Lab 1: write some simple C programs
- **TO DOs: "Homework 0"**
  - Get a GitHub account.
  - Install C++ and other tools.

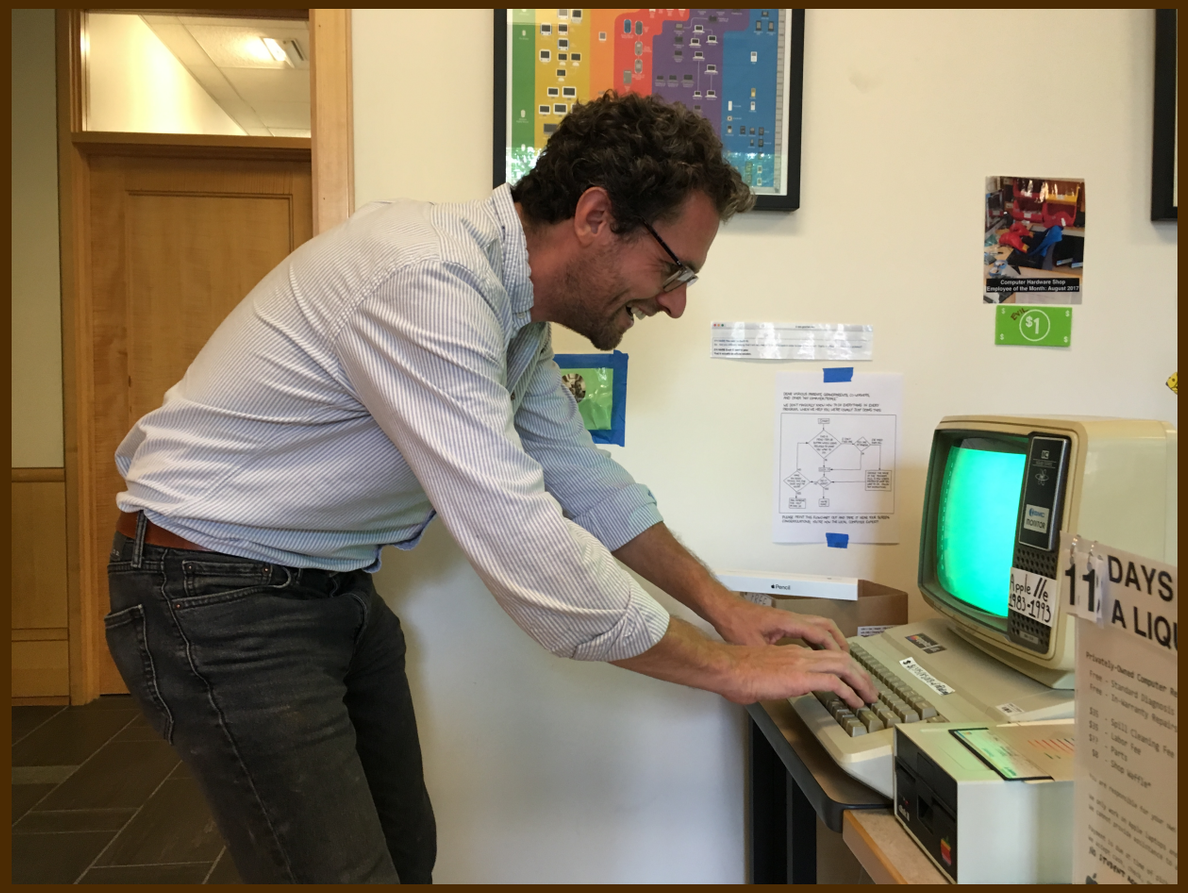
COURSE TOPIC #1

Computing systems: from the ground, up

- work at the bit level to represent data, numbers in binary
- use transistors to build AND, OR, NOT gates
- use logic, boolean algebra to devise circuits that process
- add registers, memory, and a clock to have changeable state
- devise instructions to control processor and memory state
- structure instructions into “subroutines”: procedures and functions
- structure memory as a “call stack” to manage subroutine execution
- structure data with pointers, make linked structures

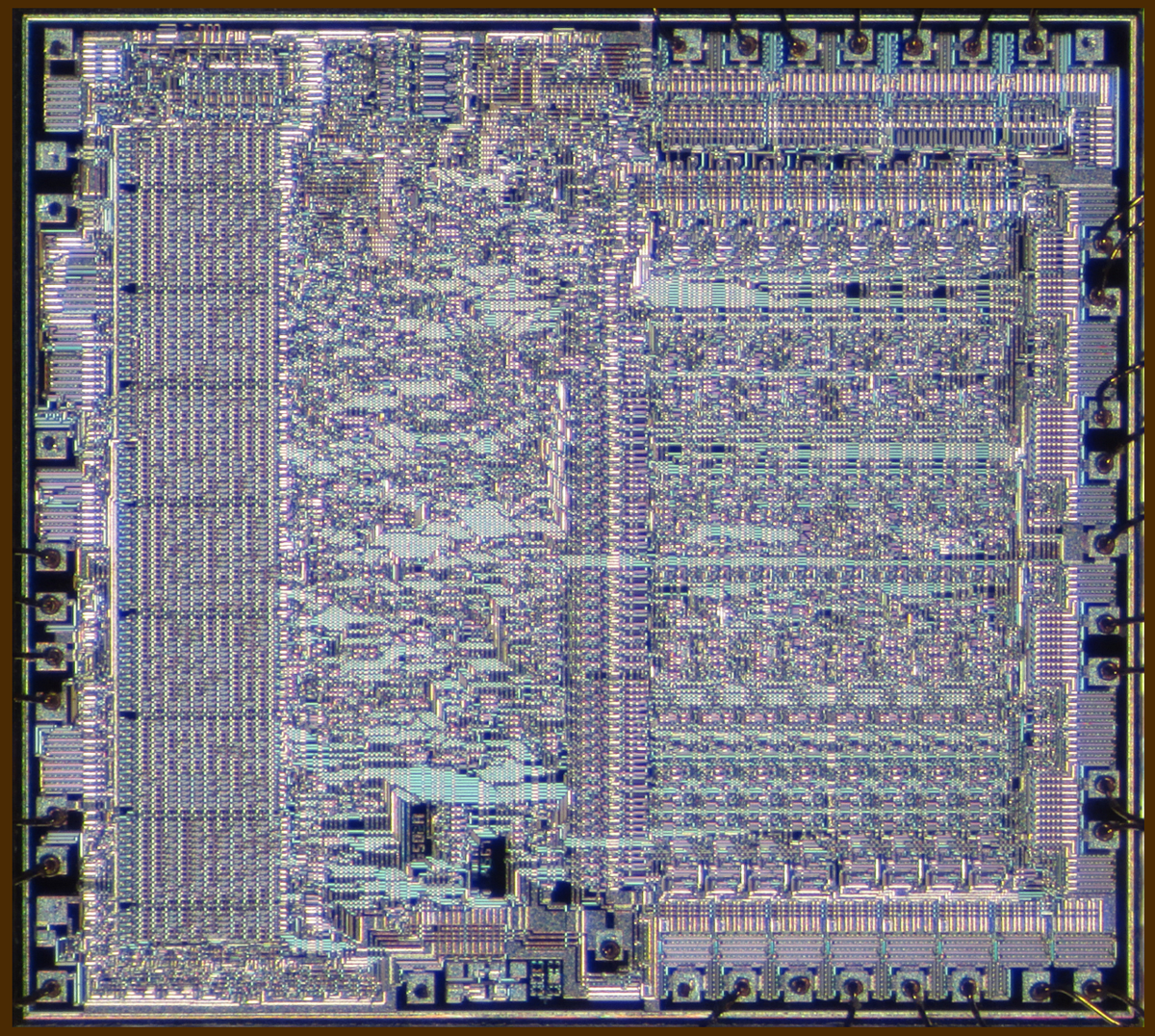
A MEANDER THROUGH MY COMPUTING HISTORY

- I started programming (in BASIC) around 1982 on my cousin's Apple II and then my own Commodore 64:



MY HISTORY

- Both built on the 6502, a mid-70s processor
- 65536 bytes of memory (64KB)
- 8-bit architecture, 16-bit addresses
- 1MHz clock
- ~5000 nm features
- 16 mm^2 die



REED'S COMPUTING HISTORY

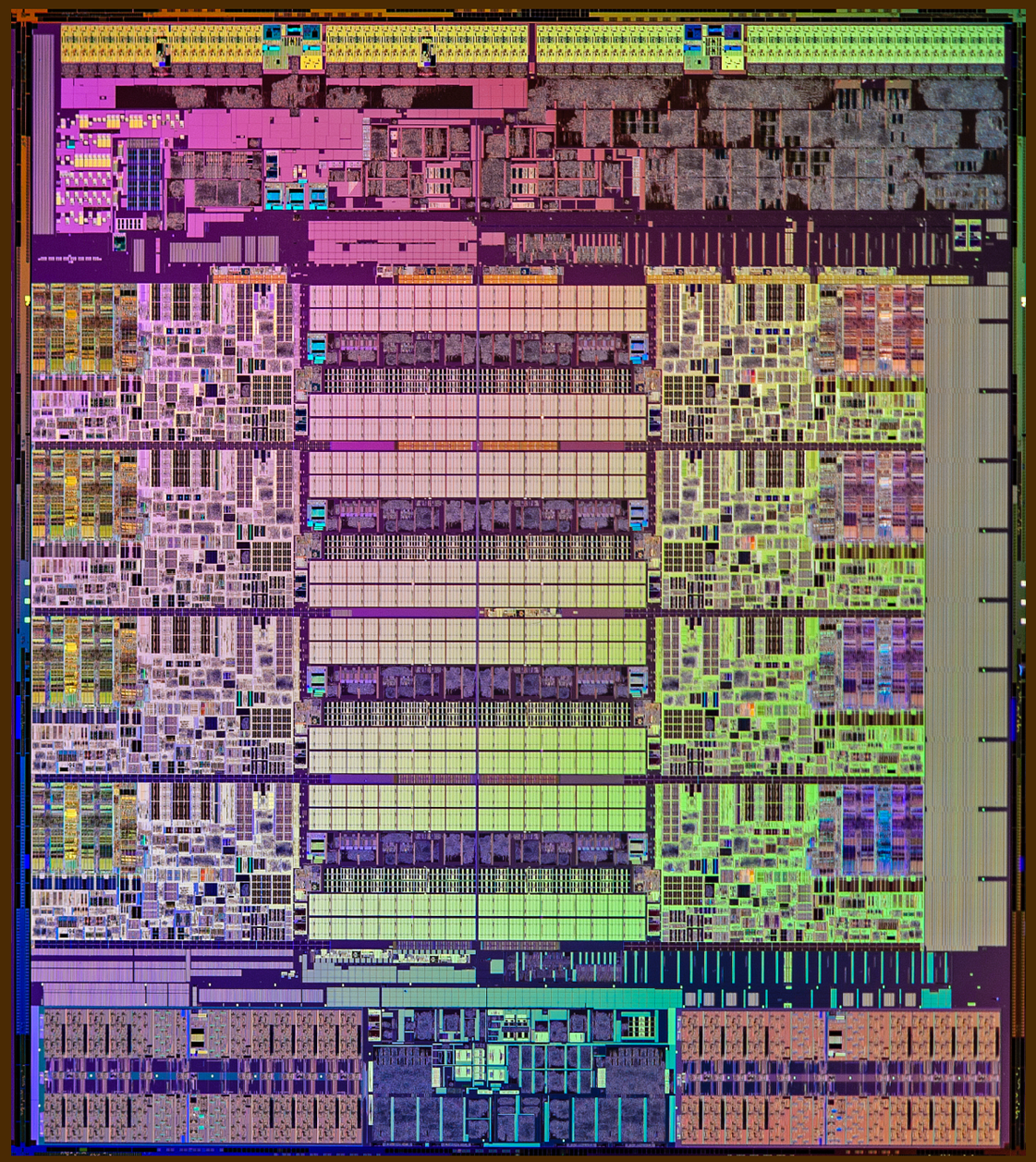
- Though there were computers that preceded it, Reed's computer science explorations started with its purchase of a DEC PDP 11/70
- in 1977, ran Berkeley's Unix, UCB's version of Bell Labs Unix
- Students sat at a bunch of terminal consoles in the basement of Eliot Hall, in "the terminal ward."
- (Prof. Richard Crandall and students built a laser network transmitter to tie it with computer terminals in the Physics building.)

THE PDP 11/70

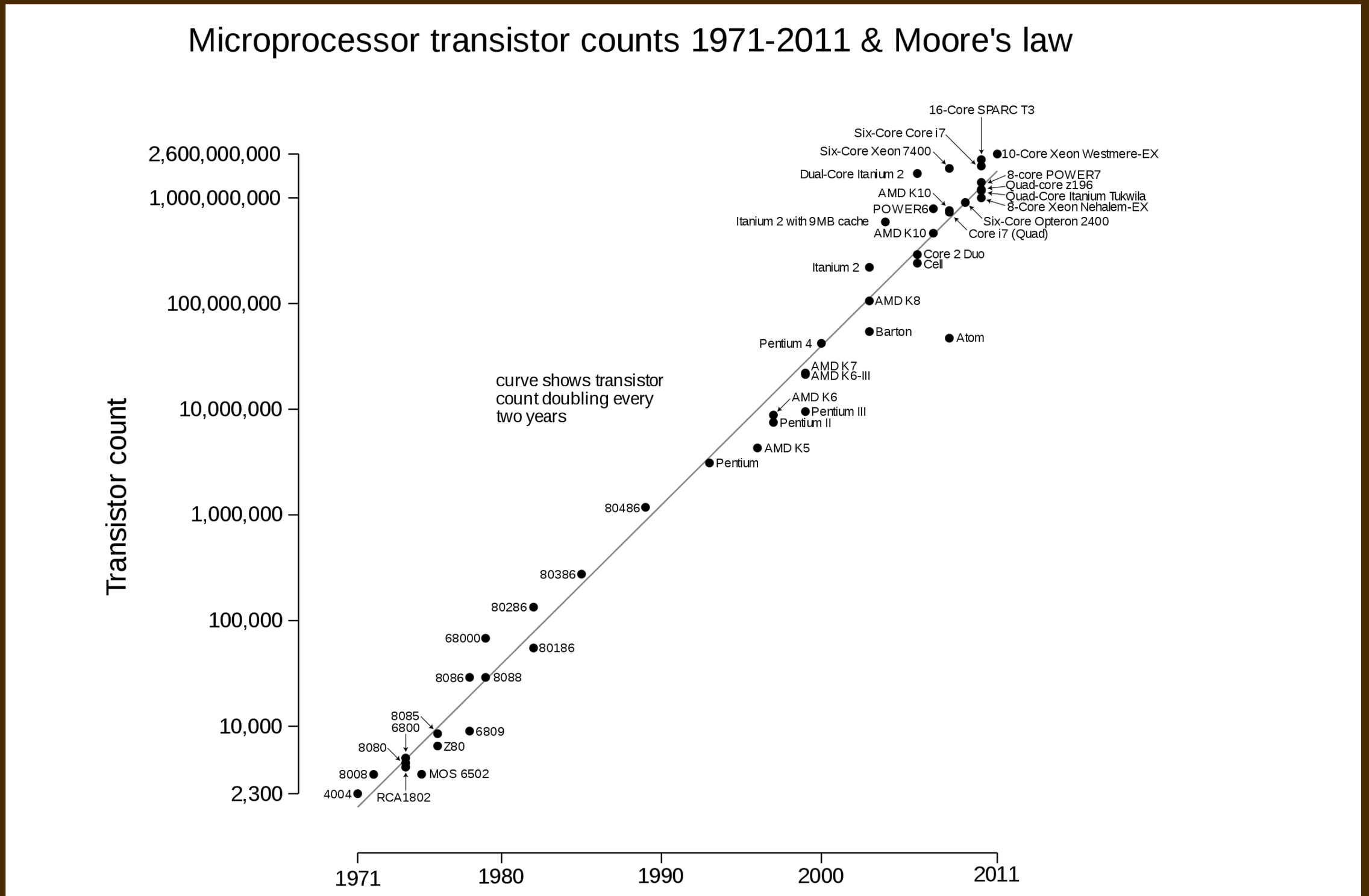
- The PDP 11 was the development platform for C and Unix at Bell Labs
- 16-bit architecture, 18-24 bit addresses

MY (2013) LAPTOP

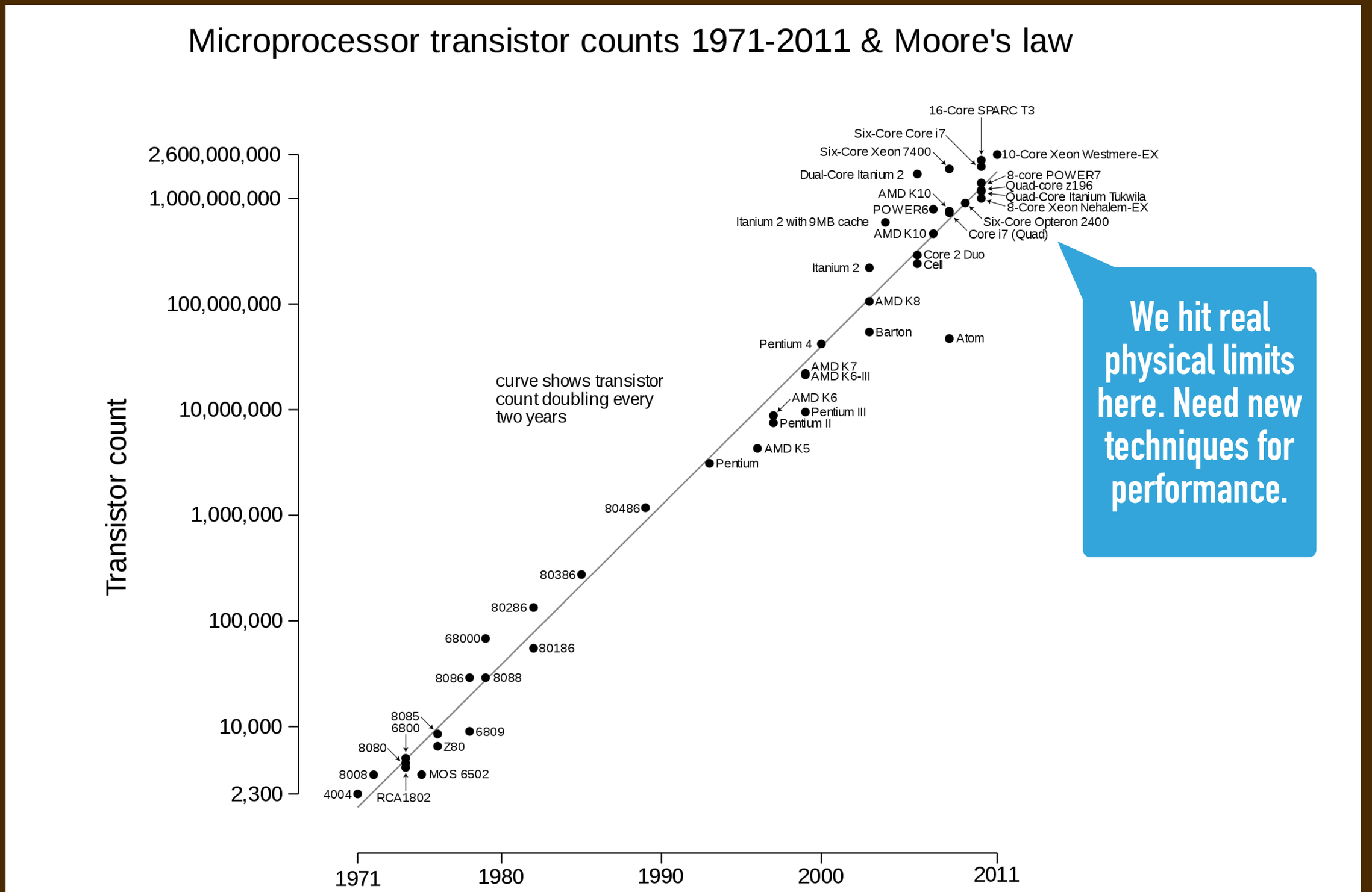
- From 2013, based on an Intel Core i7
- runs OSX 10.11.6, based on Mach OS
- 16 GB of memory, 2.8 GHz clock
- 64-bit architecture, 64-bit addresses
- 1.3 billion transistors
- 181 mm^2
- 22 nm feature size
- 2 cores
- Picture: similar family, 8 core



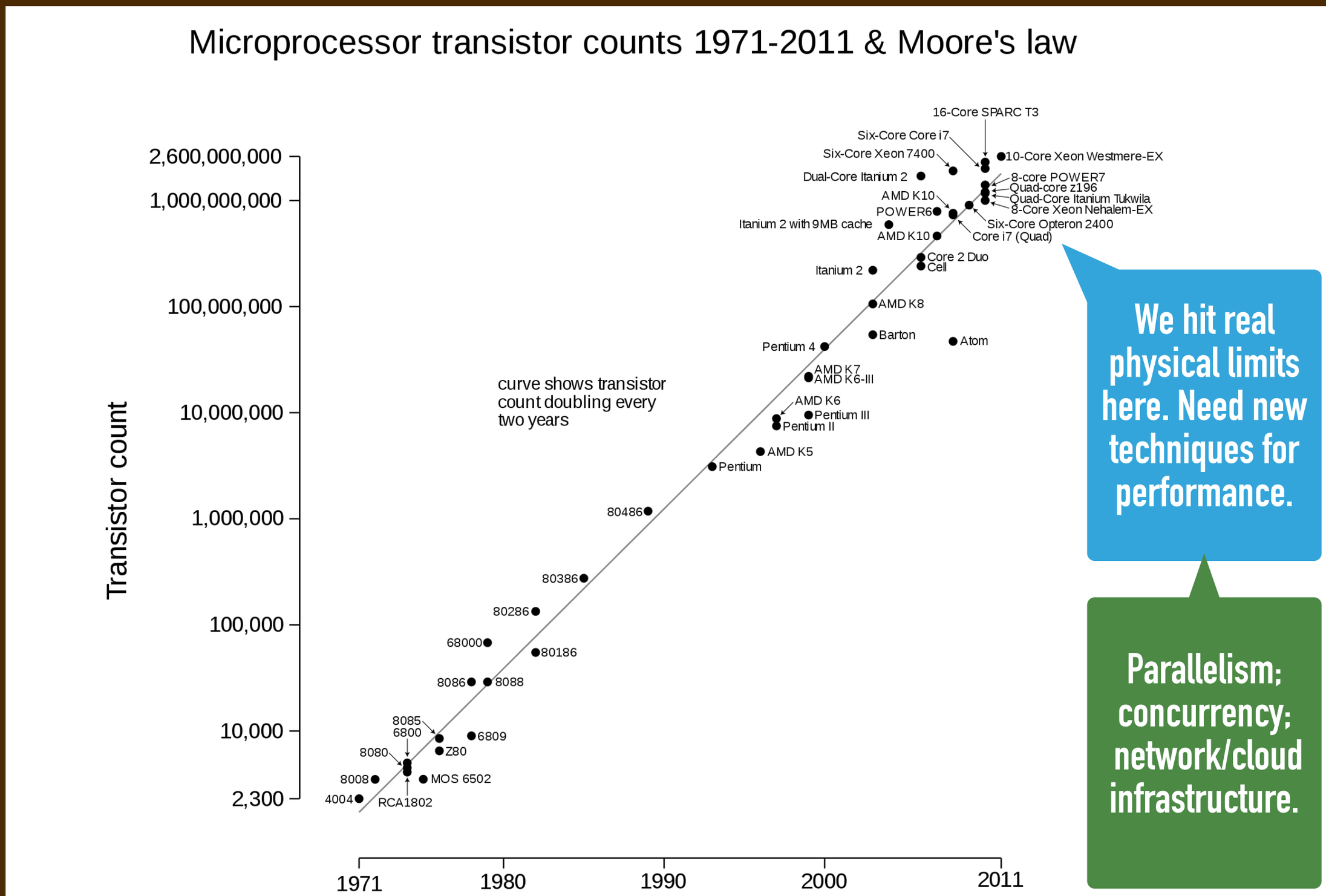
PROCESSOR PERFORMANCE SKYROCKETED OVER 40 YEARS



PROCESSOR PERFORMANCE SKYROCKETED OVER 40 YEARS



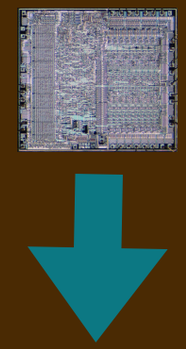
PROCESSOR PERFORMANCE SKYROCKETED OVER 40 YEARS

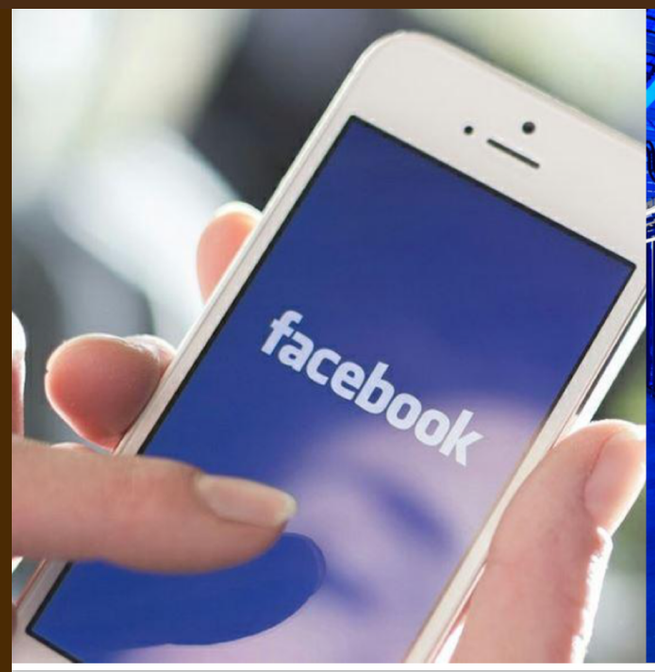


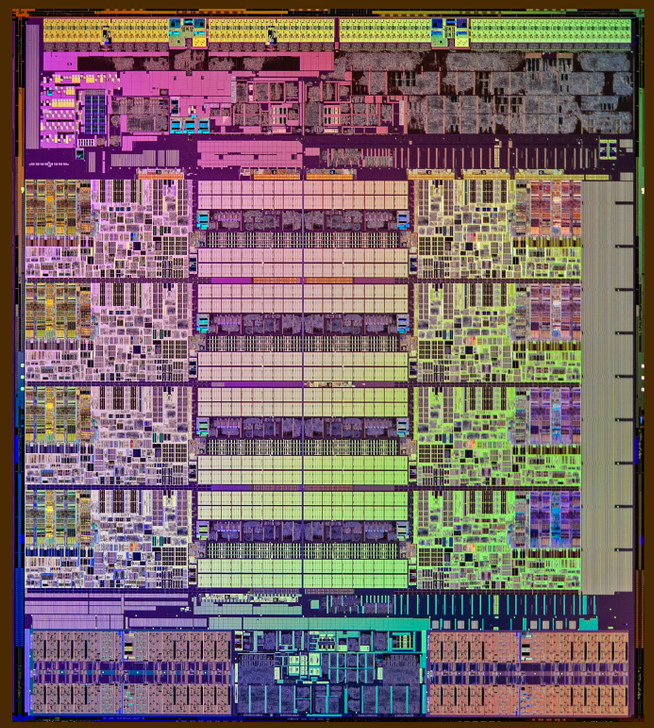
PARALLEL COMPUTATION: YESTERDAY AND TODAY

- **More of my history:** programmed parallel computers in the late 80s
  - ➡BBN Butterfly 64-node computer (Livermore)
  - ➡MasPar MP-2 with 16384 4-bit processors
    - These were kitchen appliance-sized machines and cost $1M+.
- **Today's computers** have several processors on a chip
  - ➡normal to buy computer with a 4-core chip;
    there are 16-64 core chips available for only 4-16x the price
  - ➡graphics processors (GPUs) have 500-2000 "streaming" processors\
    - So there are 80s supercomputers on a single chip, and under $15K!

SYSTEMS FROM YESTERDAY -> TODAY







COURSE TOPIC #1

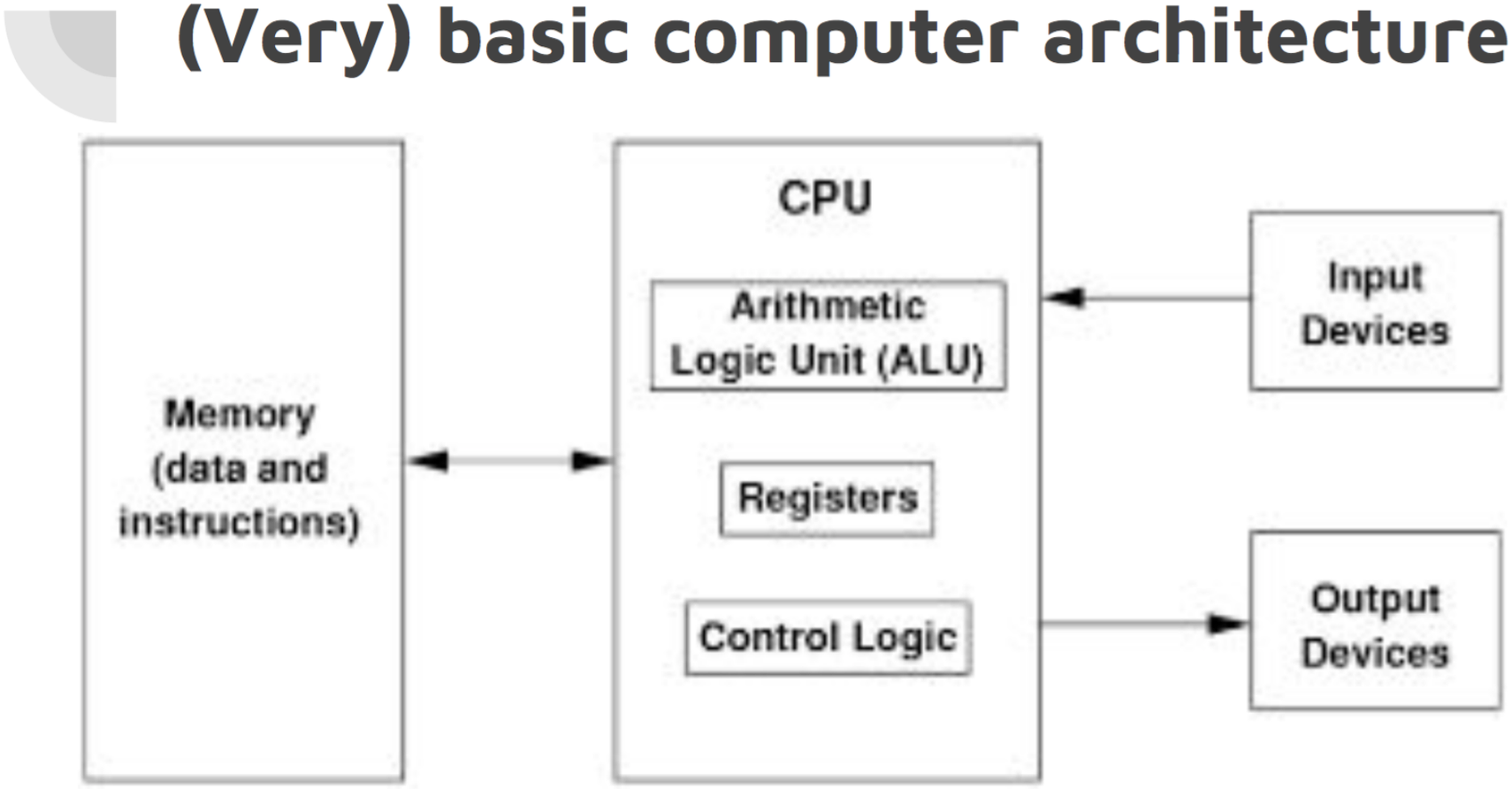
Computing systems: from the ground, up

- Through bits, transistors, gates, circuit components, instructions, subroutines, structured data and code…

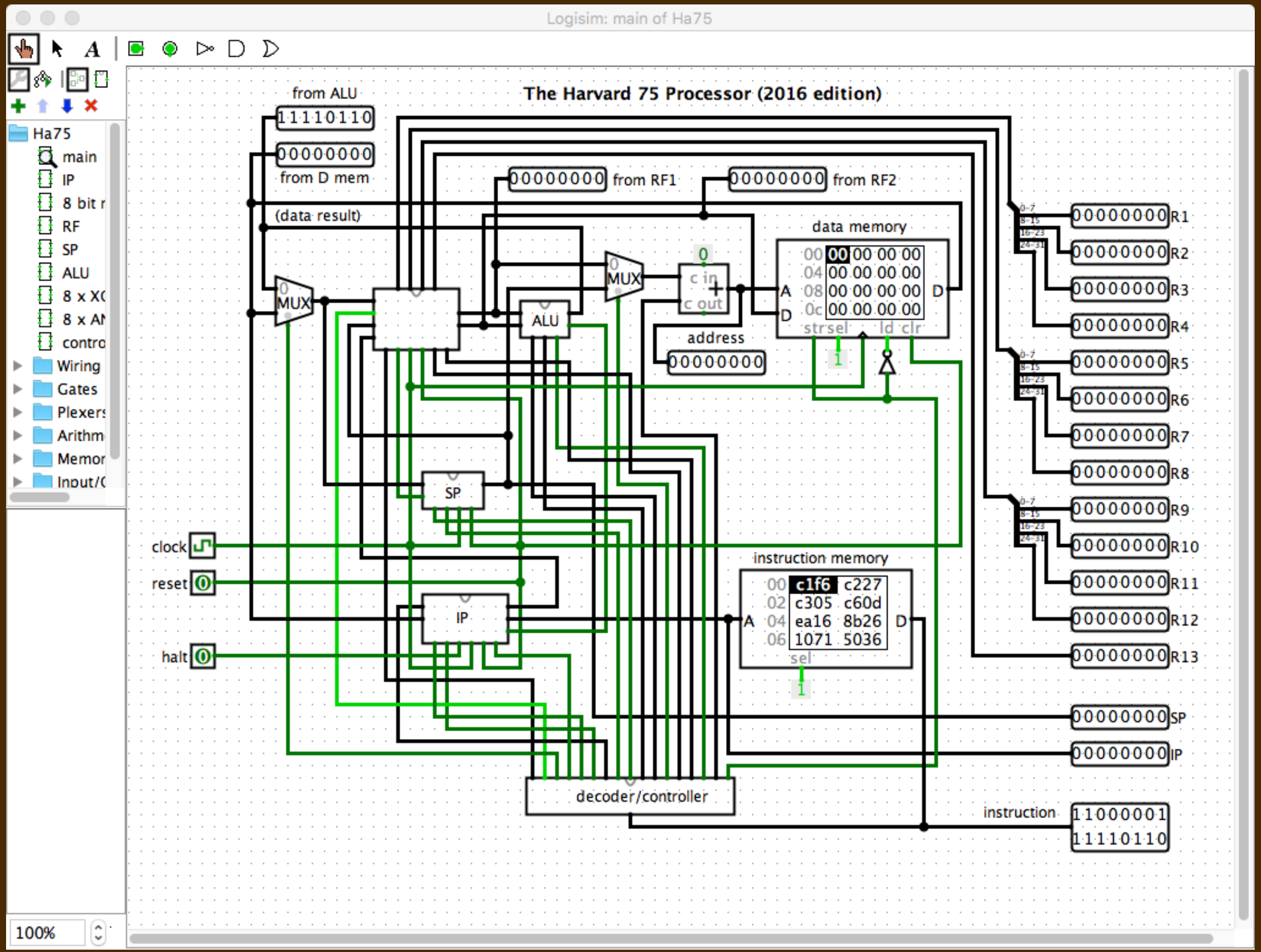
Regarding 70s/80s versus 2020 technology:

- Yes, significant advances in transistor tech and fab, smaller transistor components with more on a chip, lots of complex execution tricks, much faster execution, …
- *BUT* despite all these advances, the details haven't really changed, *at least not in principle*, in 40+ years.

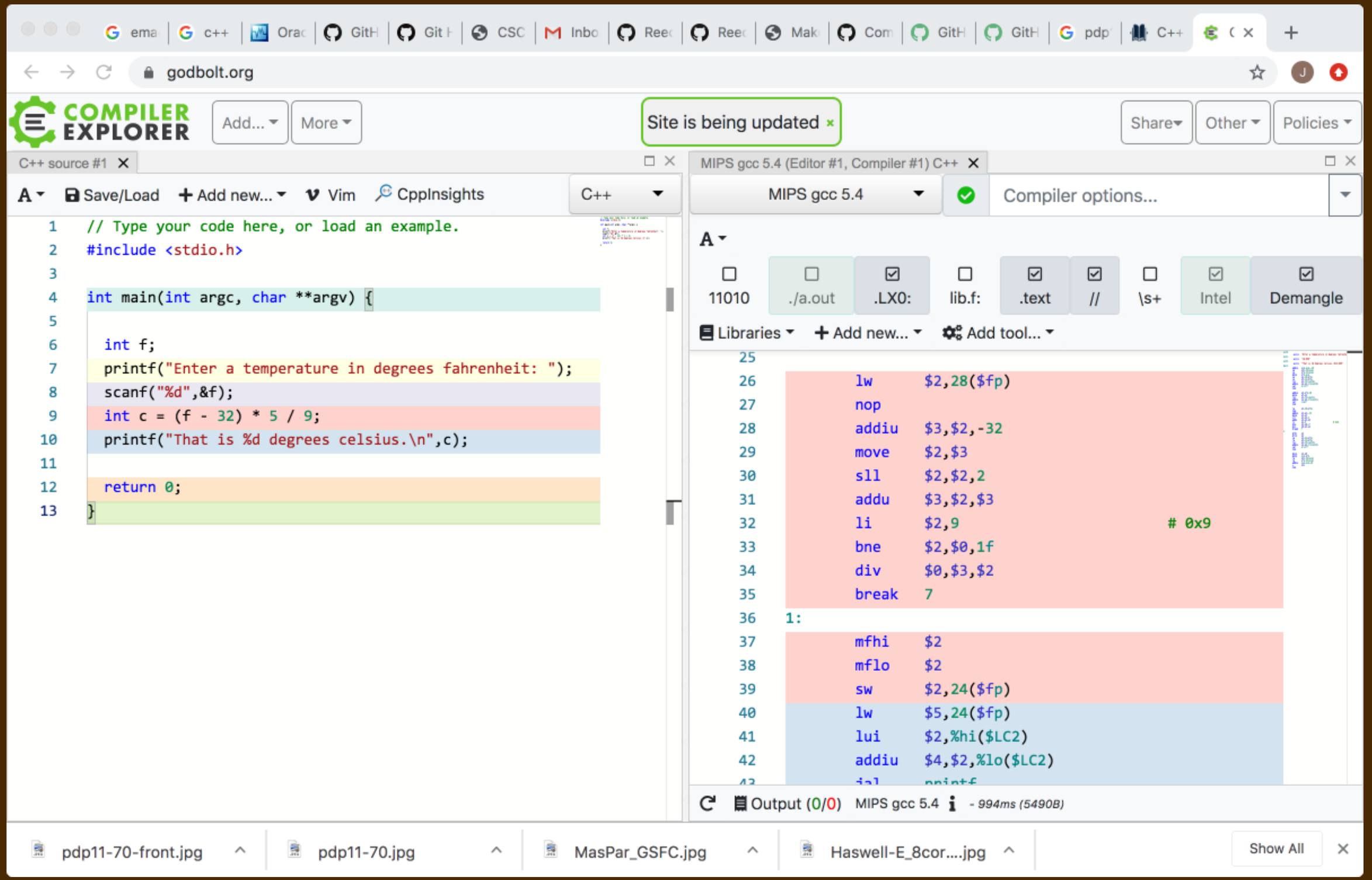
IT'S STILL ALL UNDERSTANDABLE



IT'S STILL ALL UNDERSTANDABLE



IT'S STILL ALL UNDERSTANDABLE



COURSE TOPIC #2

An introduction to systems-level programming:

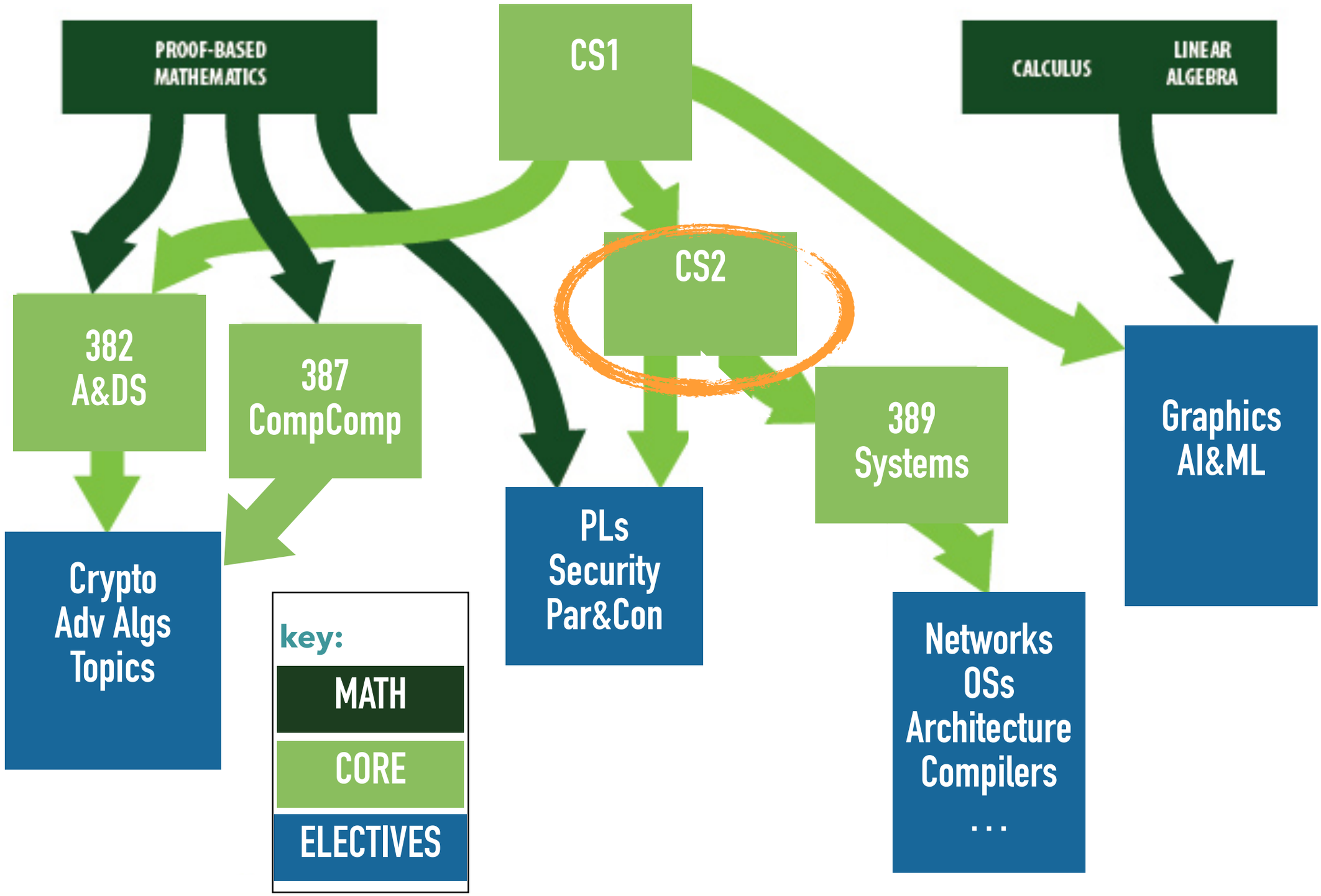
- machine-level programming of a MIPS 32-bit processor
- low-level programming in C (w/ explicit pointers)
- advanced programming in C++ (w/ its STL)
- introduction to concurrency and network programming
- a look at careful program resource management

WHY "SYSTEMS-LEVEL" PROGRAMMING

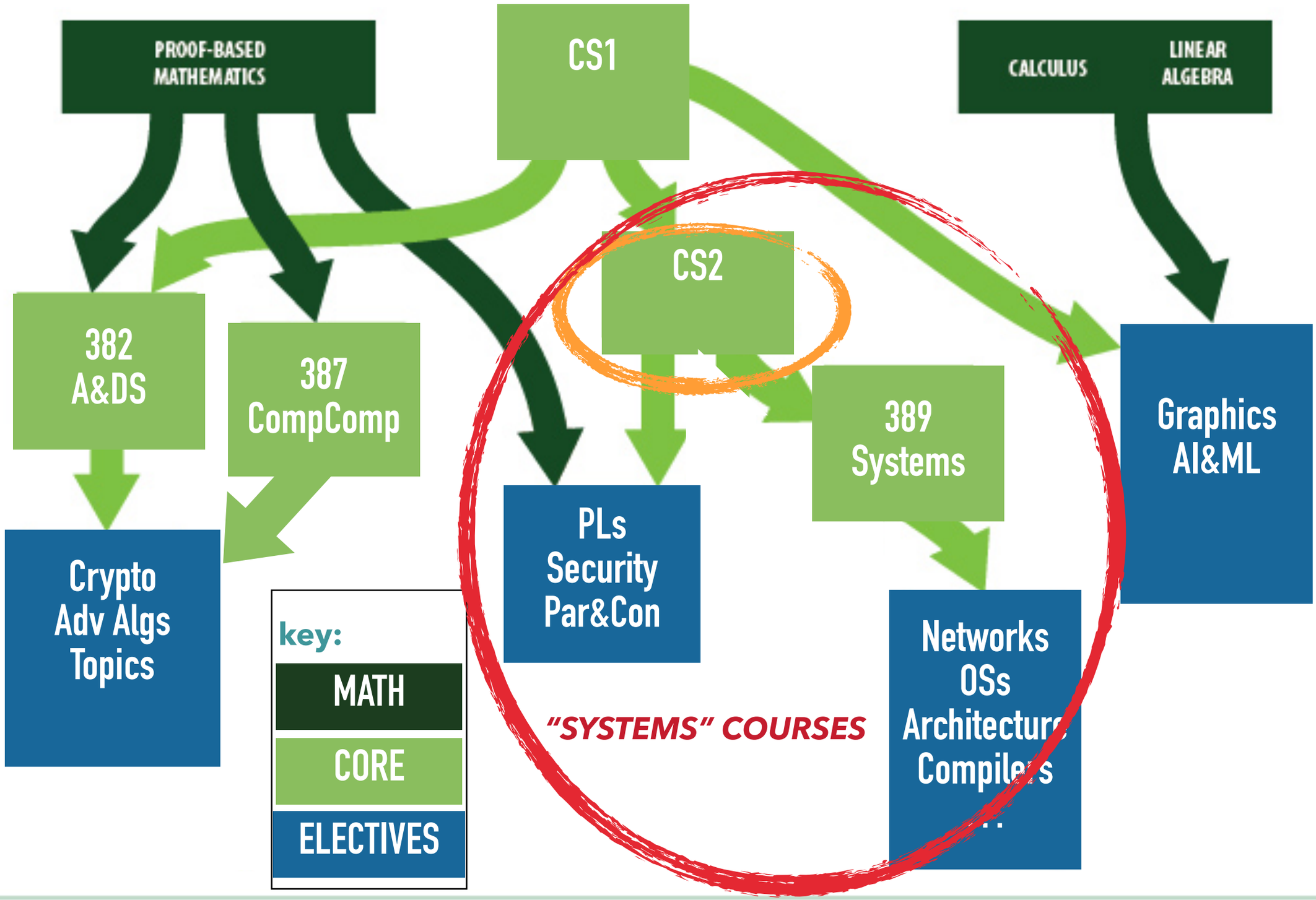
- Gain intuition for how applications and programs actually run.
  - ➡ There are many beautiful engineering ideas.
- Provide a framework for talking about performance, efficiency, costs, energy, etc. and managing memory carefully.
  - ➡ This lays the foundation for CSCI 389 and thus the systems electives.
- Begin your transition from programmer to "meta-programmer."
  - ➡ You can someday advance our tools and infrastructure, fix vulnerabilities. Many vulnerabilities are from C, C++.

It can be a challenge, with tricky puzzles. It's also rich, great fun!

CS2'S PLACE WITHIN THE CS MAJOR



CS2'S PLACE WITHIN THE CS MAJOR



COURSE TOPIC #3

Object-oriented programming in *modern* C++

- widely used industrial language
- requires maturity and sophistication
- many advanced features: access control, generics with templates, lambdas, “smart" pointers, …
- the STL, a rich library of classes
- learn to develop larger projects with many components
- an opportunity for apprenticeship

SEMESTER SCHEDULE

- **Weeks 1-4:** Intro to C programming; array- & pointer-based data structures
- **Weeks 5-6:** Digital logic and processor circuit design
- **Weeks 7-8:** MIPS 32-bit processor assembly programming
- **Weeks 9-12:** Object-oriented programming in C++ and its STL
- **Weeks 13:** multithreading and networking

- See the syllabus at https://jimfix.github.io/csci221

RESPONSIBILITIES

Programming assignments:

- A weekly Tuesday lab exercise; short programming problems
  - Attempt to complete before Wednesday's lecture; can collaborate.
  - Graded credit/no credit.
- A weekly homework; a series of programming problems
  - Complete before Tuesday's lab on your own.
  - Graded with feedback, plan to hold "code conferences."
- 3 or 4 longer-term programming projects.
  - Examples: parser and compiler; circuit simulator; text analysis.

Exams: two mid-term exams and a comprehensive final.

LOGISTICS: A TYPICAL WEEK IN CS2

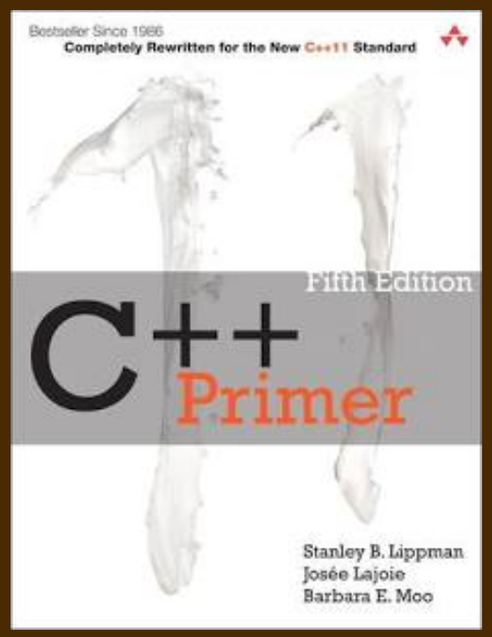
- Monday: introduce a topic
- Tuesday: exercise topic in a lab assignment; make an earnest attempt
- Wednesday: continue that topic, driven by questions from lab exercises
- Thursday: assign homework on topic; due the following Tuesday
- Next Tuesday: homework due
- Next Friday: get feedback on that homework

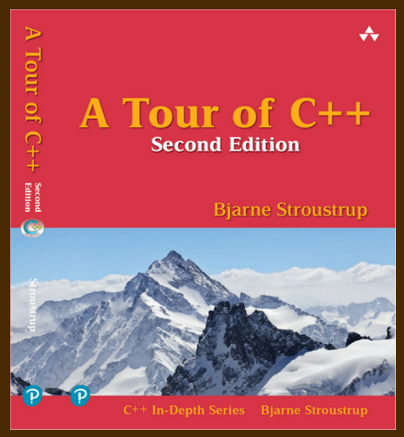
- Every 3-4 weeks:
  - Assign programming new project, due in ~2 weeks.

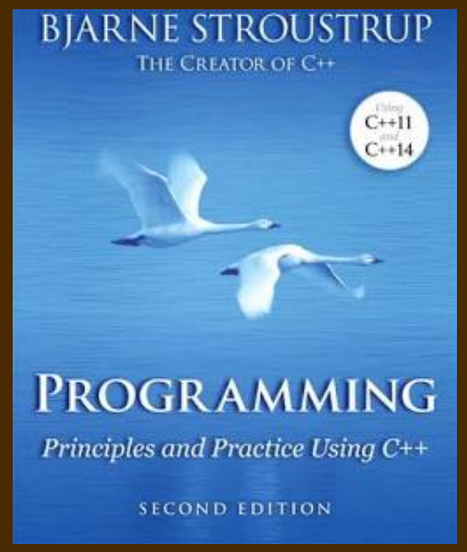
TEXTS

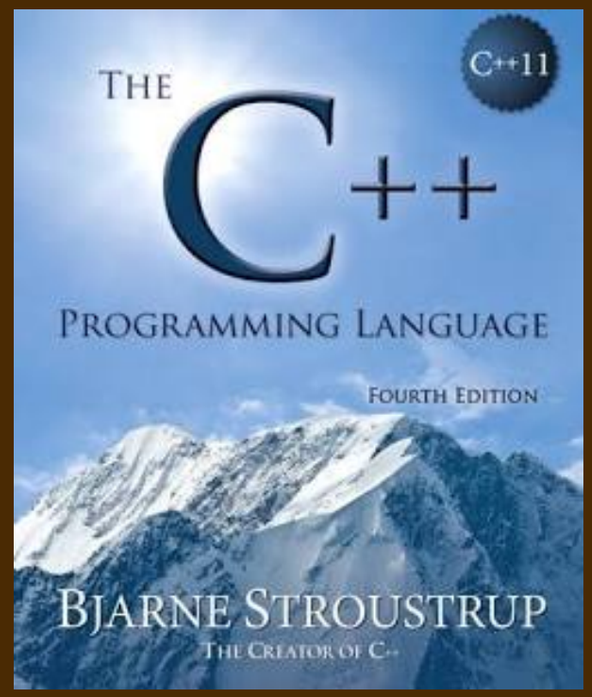
- Note: They are all optional. Can find similar references on the web.

- Bjarne Stroustrup's C++ texts









- Lippman's *C++ Primer*
- A few other supplements: some systems texts (see the syllabus)

YOUR TO-DO LIST

Please do the following:

- Carefully read the syllabus at the course website.
- Complete by Tuesday, 5pm:
  - Get a GitHub account.
  - Fill out a course form that I'll share by email tomorrow morning.
  - I will add you to our CS2 *GitHub classroom* Tuesday night.
    - ➡ Look for an email confirmation from me by Wednesday.
- Attempt to install C++ and Unix tools on your computer.
    - ➡ Look for the **Install C++** link under **Week 1** of the syllabus.

INITIAL WEEK'S SUMMARY

Monday/Today: overview of the course and syllabus!

Tuesday: lab meeting/Zoom is ***cancelled***; set up Git and C++ on your own.

Wednesday lecture: introduction to C programming

Thursday morning: C programming warm-up as Homework 1

Next Monday: Labor Day; ***no lecture***

Next Tuesday lab: practice using Git; finish C warm-up

Your **TO-DOs:** Git account; return e-form; try C++ install