CS140E: embedded OS

Stanford Winter 2026

The Staff

- Dawson Engler (if stuff is broken it's my fault)
- Head TA: Joseph Shetaye (rockets, booms, winning bets)
- TAs:
  - Maximilien Cura (Rust + insane hacks)
  - Aditya Sriram (ox64 riscv + bass)
- Section leads: (All: unpaid volunteers(!))
  - Rohan Ram Chanani + Asanshay Gupta (the GPU guys)
  - James Yu-tang Chen + Sai Ketan Konkimalla (the DOOM! guys)
  - Ron Dubinsky (Paxos+networking)
  - Stuart Sul (ELF, real world, and top-tier AI)
  - (Almost 10% of 140e'25 came back to help!)

The Students

Some staff-centric context:

- Lab classes usually capped around 30 students: We graduated 73 last year.
- Class start: 530pm. Common: staff helping til after midnight.
- Our former head TA Joe Tan sighted many times at 3am.

This year:

- 180 applications.
- Very hard folders: already a very self-selecting group.
- Max, Joseph, Asanshay, Ron each spent roughly 20-30hrs over 6 weeks to pick ~80 folders.
- (Context: More hours than a typical ms/phd admission commmitte member.)

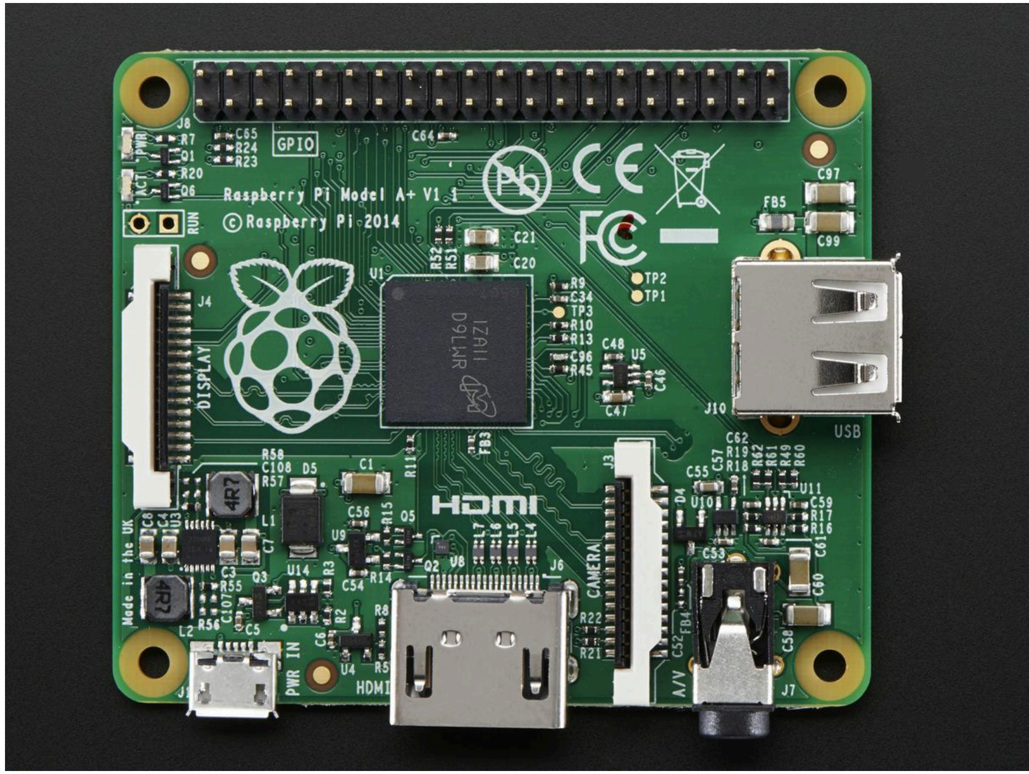
A note about work

Stanford doesn't pay to staff lab classes:

- Most of the staff are unpaid volunteers.
- This class is way too much work even if paid.
- So if you need help outside class please only bug non-volunteers (Max, Aditya, me).

Class: Write small, clean OS on a r/pi zero w.

https://en.wikipedia.org/wiki/Raspberry_Pi



Class philosophy

Write a complete, narrow OS all the way down to the bare metal.

Typical OS class:

- a bunch of complicated stuff covered superficially, but with alot of bugs and starter code.

Our approach:

- You'll write complete, simple versions that work. On real hardware.
- In fact: b/c of novel tools you'll build: it will be *surprising* if code broken.
- And: Because you write it all, you will understand much more thoroughly
- The hope: easy to do delta off of your knowledge to more fancy things

Why write bare-metal OS code?

Power:

- Real control of real hardware = real superpower.
- Can build many things easily that are essentially impossible on linux/macos.
- Guaranteed nanosecond performance, actually secure, many devices.

Powerful

- If you can write kernel code: can write almost anything (non-math-y)
- Bugs here are some of the hardest yet invented by entropy:
  - device hardware errata, page table mistakes, interupt lock-ups, trashing a register during context switching = good luck with that.
- You'll think and code more clearly b/c otherwise it sucks.
- Bonus: the bugs you hit will *really* teach you about the

Why I do it.

Small bare-metal system you wrote yourself:

- Fluency is 100.
- Flow state is maxxed.
- Very very easy to drop in and immediately make changes.
- And since it's bare metal, you can make it do crazy things, quickly.
- At my age, not as easy to have fun writing code. This does it :)

Class org: Labs. More labs. Then: Final project.

Organization:

- Very little lecture (today is longest)
- 18 labs, 2x per week. Start at 530pm. Go til you want to leave.
- Prelab material (plus gradescope prelab) before Lab.
- Lab: walk in, by the end of the lab, you have a complete working simple version
  of a key trick.
- E.g., 300-400 line working interrupt handling or veritual memory system.

This class has no textbook.

Readings = raw primary sources:

- datasheets and architecture manuals.
- (Though we do give you annotations and commentary.)
- Some are intense: virtual memory is about 100 pages of tough material.

This class has no textbook.

Why:

- Classes present a fake reality. The real real world is not:
  - A clean textbook of systematized knowledge.
  - With pretty, worked out problems
  - And bolded text saying you should do multigrid relaxation on problem 4.
- It is:
  - Difficult to understand documents with no high bit. That are: wrong, incomplete, not written to be used ("passive definitional voice"), describing nouns that themselves have bugs.

This class has no textbook.

Promises:

1. If you understand these, you can do things others cannot.
2. After 10 datasheets, the 11th is not a big deal.

Who is this class for?

Class is heavily tuned for those that are interested in low level code/hw.

- It is built to move fast with a bunch of people that are very interested
- Only take the class if:
    i. You find Stanford easy and/or
    ii. You are very interested and have a very open quarter.
- Sub-category: If you're on the spectrum (hi).

Our goals:

- We don't waste your time;
- You cover adult stuff in a way and level you haven't seen;
- You'll find many hardcore people you want to work with in the future

Warning: Don't take this class thinking it is an "easier version of 112" (it is not).

Why R/PI A+?

- Most OSes write code on a fake simulator
  - A lot of work, not that cool at the end
- R/pi = real computer for about $16 and an ounce of weight
- Many examples / blog posts of how to do various things
- Unlike most machines, makes interacting with the real world easy
- Can build many interesting systems b/c can use weird hardware easily
  - motion sensors, IR sensors, accelerometer, gyroscope, light sensor...
- Actually fast compared to modern SBCs (1GHz), and has an MMU.

Goal: you will develop two super-powers

Power 1: Differential debugging

- Efficiently answering "why doesn't this work" for complex things
- Swap working pieces + Binary search

Power 2: Epsilon development

- Epsilon sprint paradox: When building systems, the smaller the step you take, the faster you can run.

Differential debugging

You write code, it doesn't work, the error could be:

- The code you wrote
- Hardware fault (bad manufacturing, smoked something)
- Wiring mistake
- Subtle cache issue
- Compiler problem (more on this)
- ...

You will get good at breaking down problems by swapping pieces between a **working system** (yesterday's code, your partner's lab) and a **non-working system** (today's code, your lab)

Example from next lab

You get the following set of stuff:

To run you:

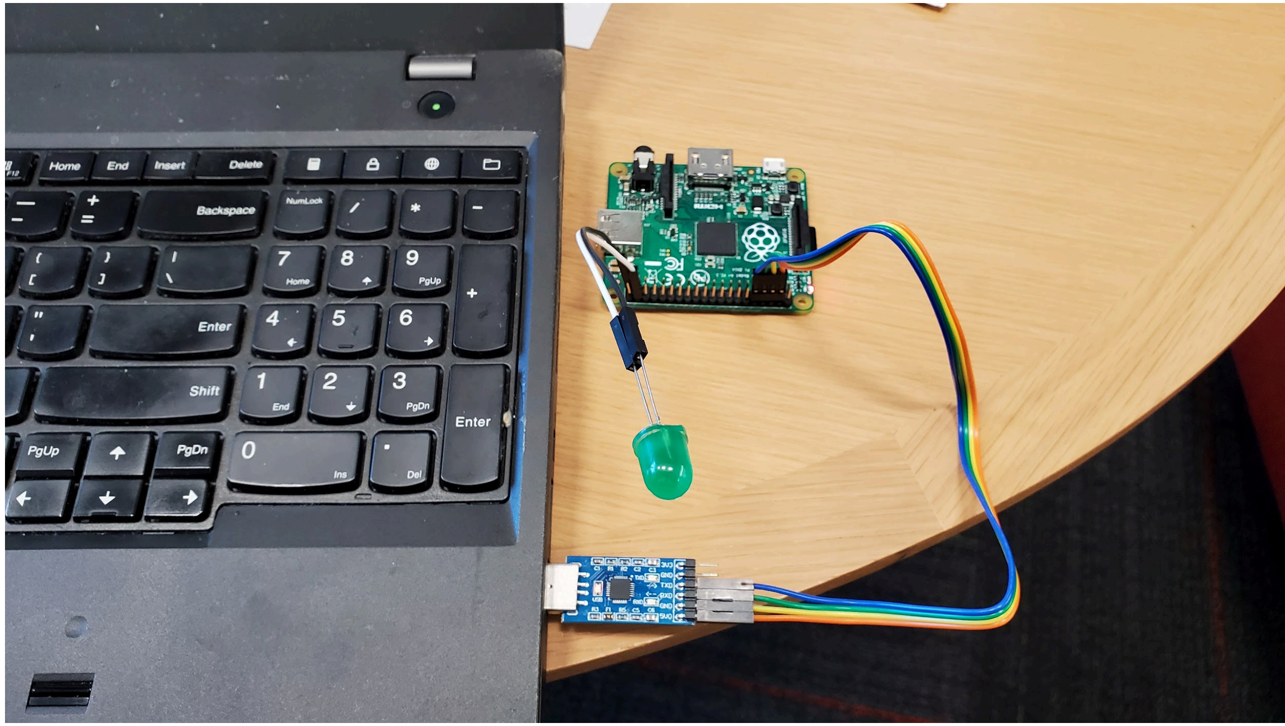
1. Copy blink.bin to sd
2. Wire up led
3. Wire up serial device
4. Plug into your laptop

It doesn't work. But your partner's does.

What is messed up?

Partner's	Yours
Working	Not working

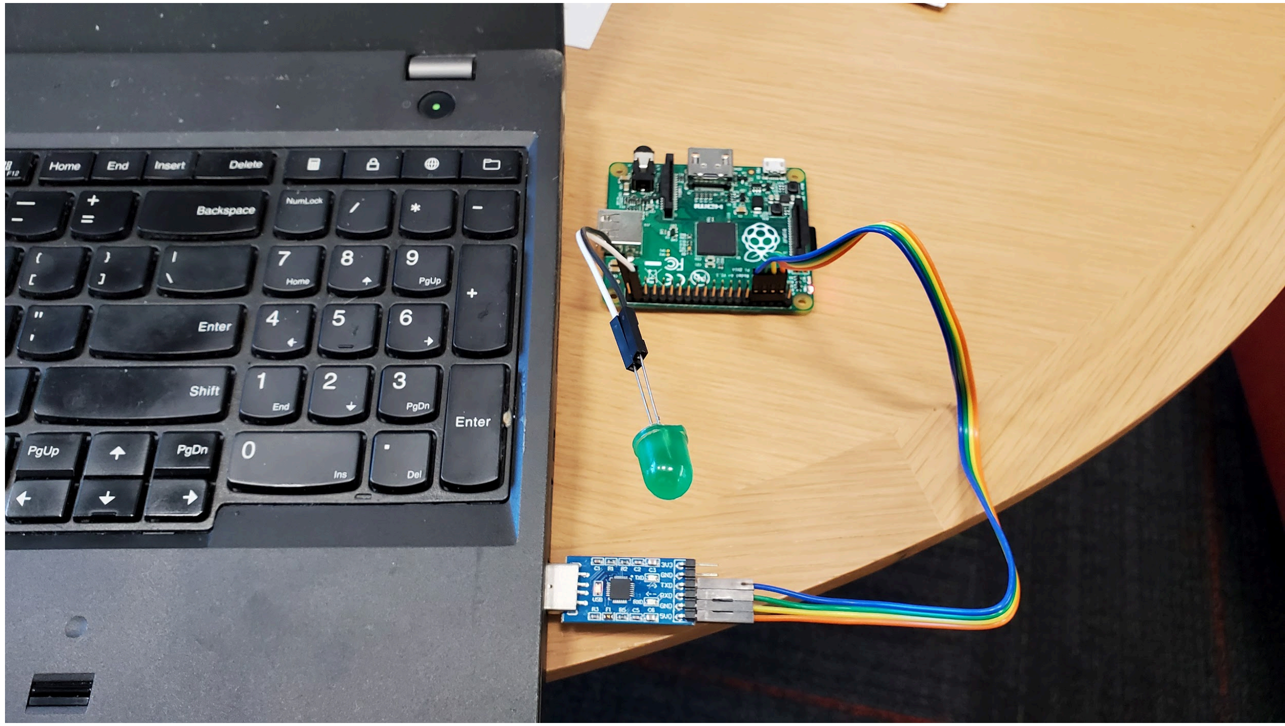
What to do first?



What is messed up?

Partner's	Yours
Working	Not working

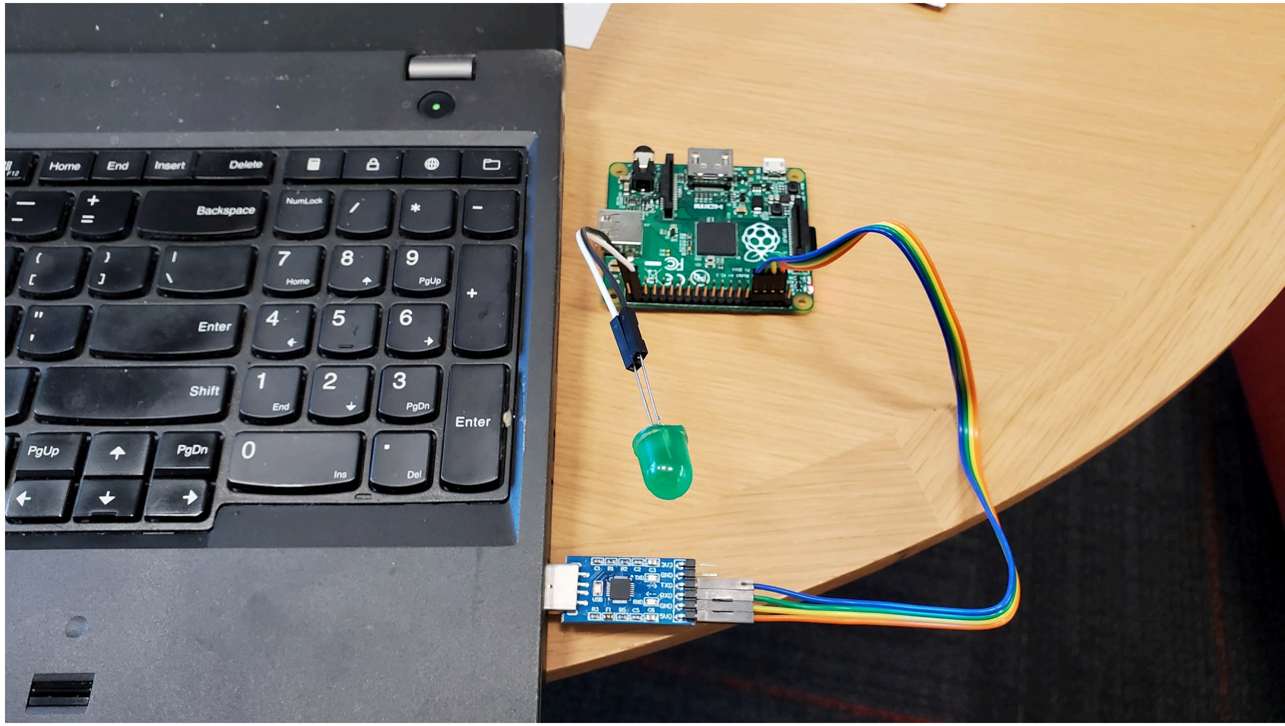
- What does swapping tell you if it doesn't work?
- What does swapping tell you if it works?



What is messed up?

Partner's	Yours
Working	Not working

Swapping works: how to narrow down with least work?

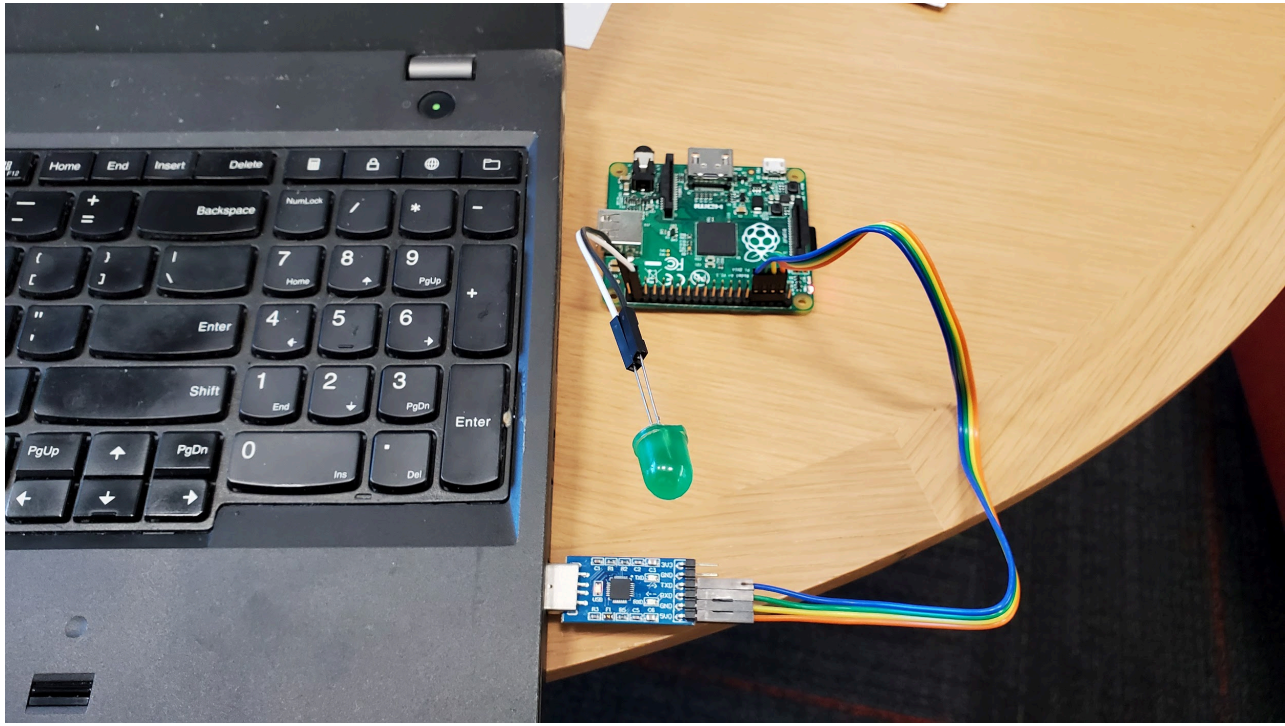


What is messed up?

Partner's	Yours
Working	Not working

Entire class: whenever we control device, has some software component S (can be wrong) and some hardware component H (can be broken).

Doesn't work = linear equation solving with two variables. **How to isolate?**



Epsilon sprinting: Slow is fast

What is wrong?

- If I did X, it's X.
- If I did X1+X2+...+Xn it could be any, or some combination.

Inverting crash/bug to root cause is **much harder** in the latter case.

My epsilon-sprint theorem:

> Given a working system W_k and a change C, then as $|C| \to \varepsilon$, the time + computation (IQ) it takes to figure out why { W_k + C } doesn't work goes to 0.

Related claim: the time it takes to debug why a change broke the system increases non-linearly with the size of the change.

Before TA: make sure you did an epsilon delta

Assume: 3 10-minute bugs per 80 students and 6 TAs:

- Arrival rate: (N × bugs_per_student) / hours: (80 × 3) / 6 = 40 bugs/hour
- Service rate: 6 bugs per TA / hr.
- Service capacity: 6 TAs * 6 bugs / hr = 36/hr
- Utilization: Arrival / (C x U) = 40/36 = 110%.
    - After 6 hours: 24 deep. (Worse b/c TAs quit, bursty)

140e code development:

- PLEASE DO NOT DO multiple changes and then ask "it doesn't work"
- Always take working system, add the smallest epsilon change possible to make break. Only ask then.
- Useful IRL. Also, often: you fix bug.

Administrivia

Rules:

- Feel free to leave early, but don't be late (makes us run O(n))
- Don't use llms or other people's code. You have to write.
- Don't miss more than 2 labs (even that is very tough).

Grade:

- "A" requires 3+ hard extensions
- Absolutely must turn in lab w/in 7 days. No exceptions!
- Usually prelab due before lab. Won't accept after.
- Will have a final project (roughly 3+ labs worth of work)

Plus side: We pay for food. We pay for equipment. We stay til midnight or so.

Administrivia

Two labs each week.

- Each lab will have pre-lab work you should turn in before lab
- Ideally finish during the lab period (I will stay til everyone is done)
- Must finish within a week of the lab, or start losing a letter grade each day
- Must pre-arrange missed labs. It's a problem to miss more than a couple.

There (tentatively) will be three "capstone" homework assignments that consolidate a chunk of labs together.

If you've done the lab, this shouldn't be a big deal.

New: grade partly depends on understanding

Problem:

- If we don't look for cheating, it can happen.
- Because we've been able to pick great people, it is rare. But not 0.

DO not want:

- To ignore: unfair.
- But also don't want: in-class exams, cold-calling, MOSS detection.

Hack

- Grade partly based on how well you understand your code *when done*.
- It's fine to be confused before. But after should have a grasp of it.
- Pro: no tests, more time for helping people.

Administrivia

You can work with other people!

However, you *must* type and turn in everything yourself.

(We will pay for food you order during lab.)

What to do now

1. Clone the class git repository:

```
git clone git@github.com:dddrrreee/cs140e-26win.git
```

   https://github.com/dddrrreee/cs140e-26win

2. For lab next Tuesday (one week), make sure you:
   - Have a way to write either a micro-SD or SD card
   - Have a way to plug in a standard USB device
   - Do PRELAB for lab 2-trust