The Operating System as Software

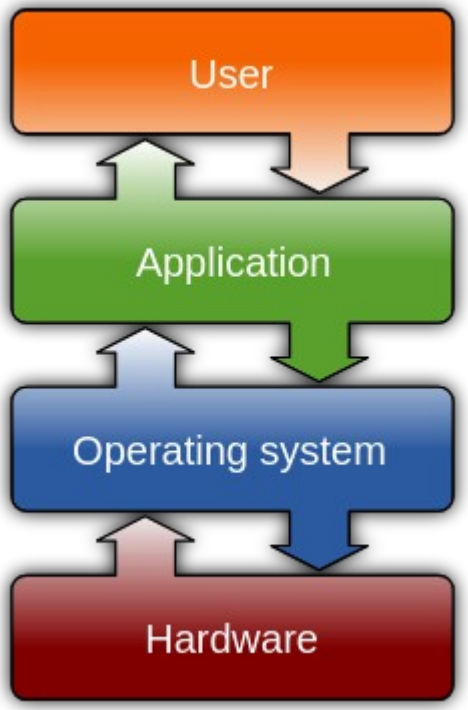
David Marchant

Important Note

- Note that several part of this material is not formally taught in CompSys
- But it’s also not really taught anywhere else
- It won’t be on the exam, and won’t be available as a recording
- Bonus knowledge!

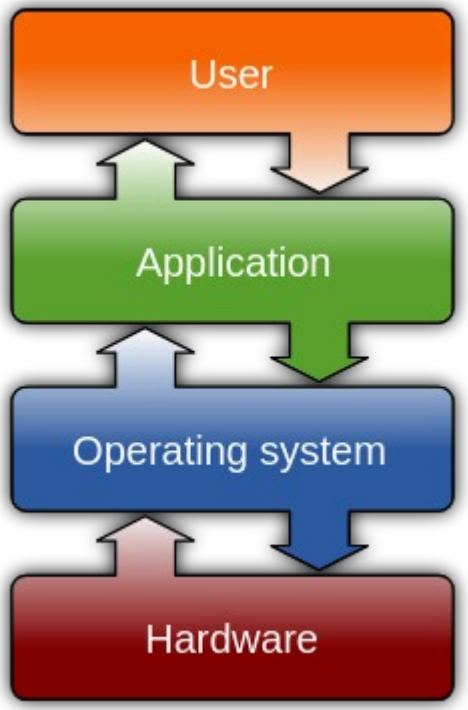
The Operating System

- The OS exists as a sort of middleware between applications and hardware
- Its not strictly necessary to run a computer, and *some* systems exist without an OS at all. And they run *fast*
- But the OS simplifies so much, and without it you really need to do everything from scratch, often in machine code (Boo)



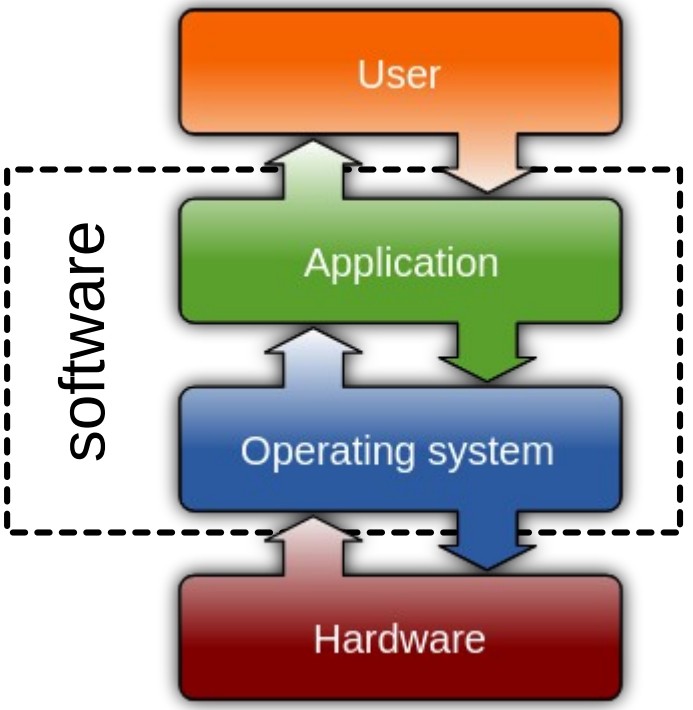
An aside - Layering

- We’ve seen this sort of thing already with language levels
- Layering is an important concept used repeatedly in Computer Science (and other places)
- Its a way of building on previous knowledge



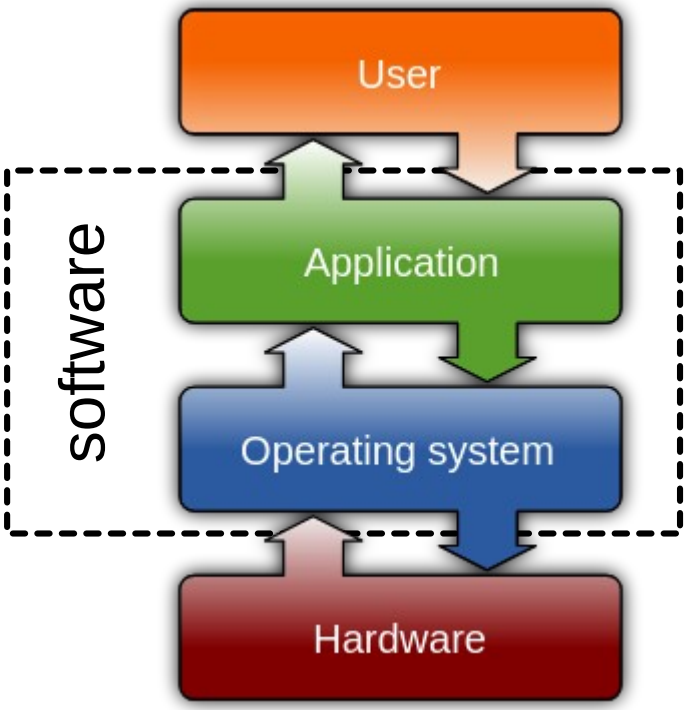
Its All Just Software

- The OS is just software!
  - Note that some components are handled in dedicated hardware (cache, MMU, ALU, etc.)
  - But this is just because they are operations that have become standard enough that we can make dedicated hardware for them. Originally many of these were software
- This makes it powerful, configurable, and slow(er than just machine code)



Its All Just Software

- Consider everything we've covered so far
  - Caching, Processes, Floats, Memory Allocation, etc.
- These are all managed by software, even when they seem to depend on hardware
- And most of them are managed by the OS



Lets consider Processes

- They’re just a collection of (meaningful) data
  - Virtual Memory Space (stack, heap, code, etc.)
  - Execution Context (registers, etc.)
  - Plus some other things added by the OS to track each process (PID, parent, children etc.)
- There’s nothing in hardware defining any of this, and the data that defines a process can just be saved to memory/disk just like any other file
- What makes it a *meaningful* process, is that the OS swaps out these saved collections of data on the fly

The Operating System

- It manages the complete runtime of the computer
- All those processes, memory accesses, blah blah blah
- ….

...

But That’s Not Enough

- The trouble with software is it only does anything when the computer is running
- E.g. powered up and already in a runnable state

Where To Begin?

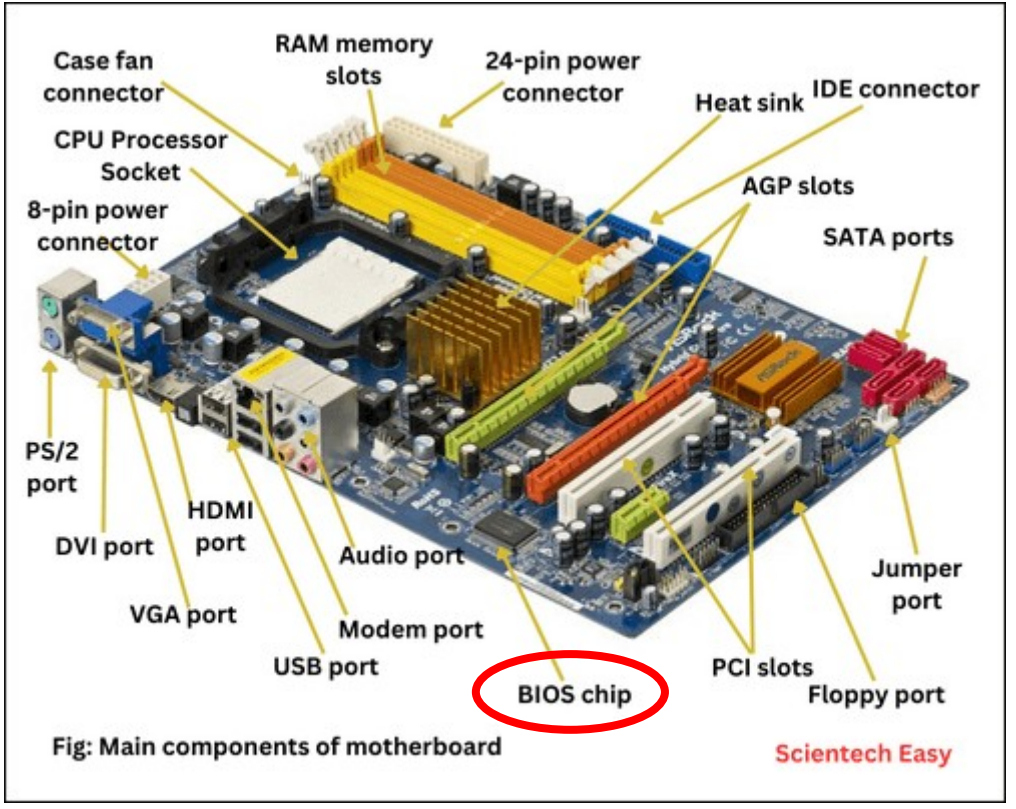
- Modern computers are large, complex beasts
- Operating Systems only complicate things by adding all their own layers
- We need some hardware to start things going
- Or at least start the ball rolling so the OS can setup the rest

BIOS

- Basic Input/Output Stream
- Performs a few functions that are outside our scope (Well, more outside our scope)
- The three important ones though are running basic setup, performing POST test, and handing off to the OS

The BIOS chip

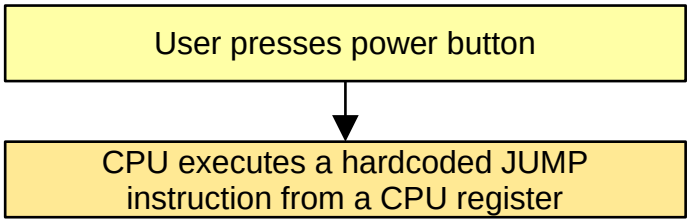
- After *seconds* of internet search, this is the best picture I could find, somehow
- The BIOS chip comes pre-installed
- Usually designed to work with a specific motherboard
- A motherboard being the main component of most PCs, linking everything else together and containing the CPU itself



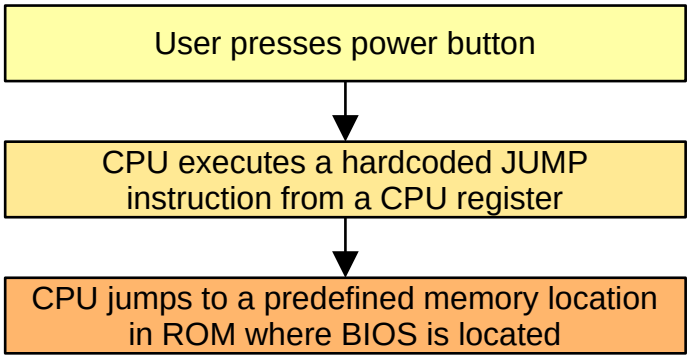
Boot Sequence

User presses power button

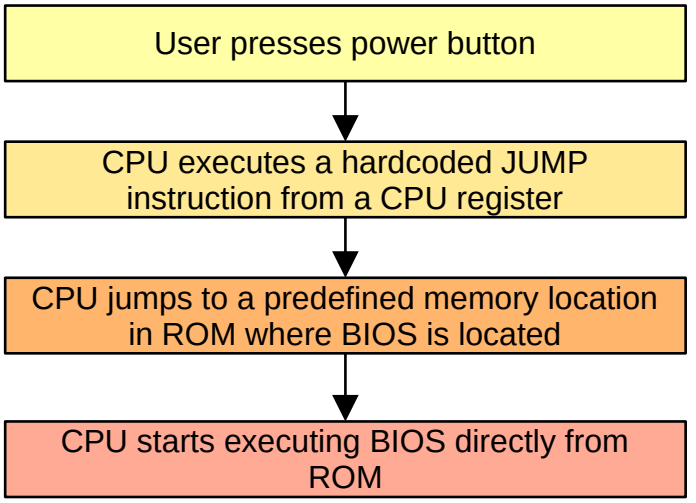
Boot Sequence



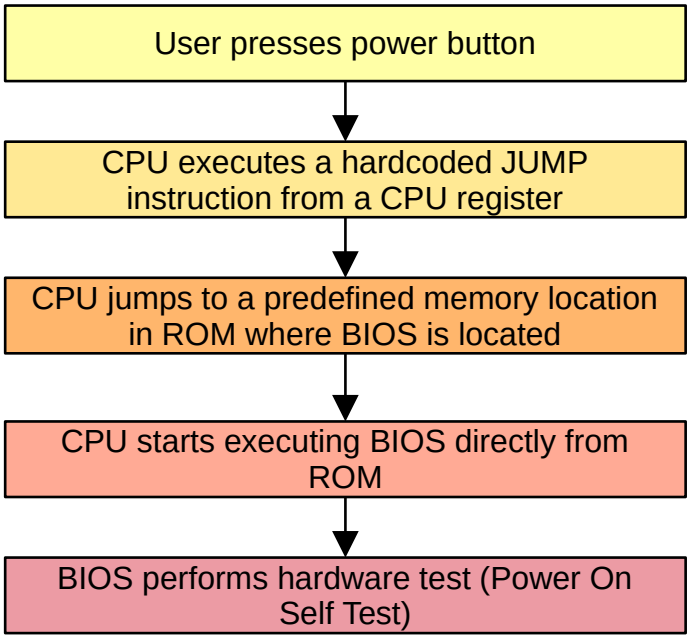
Boot Sequence



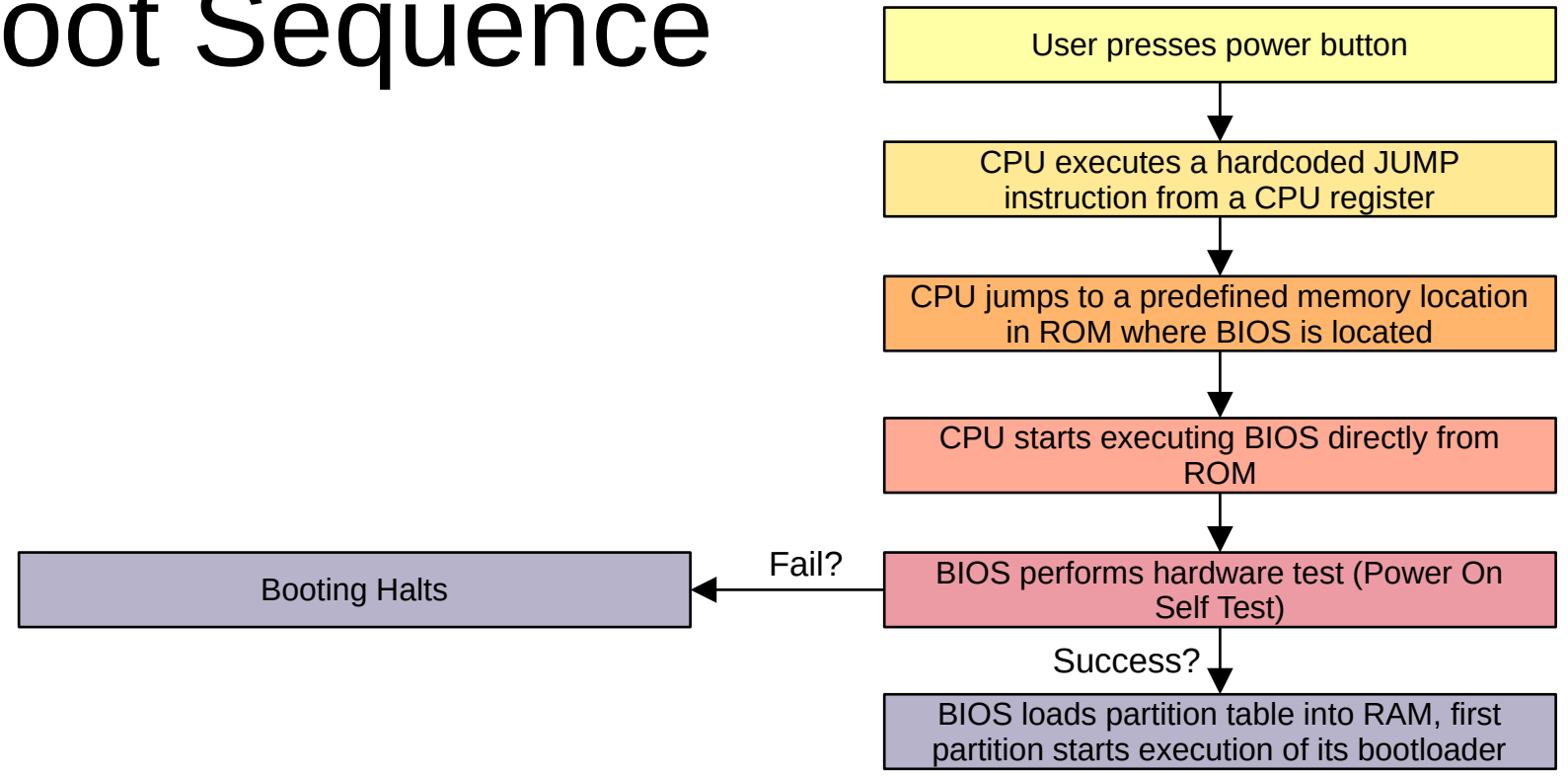
Boot Sequence



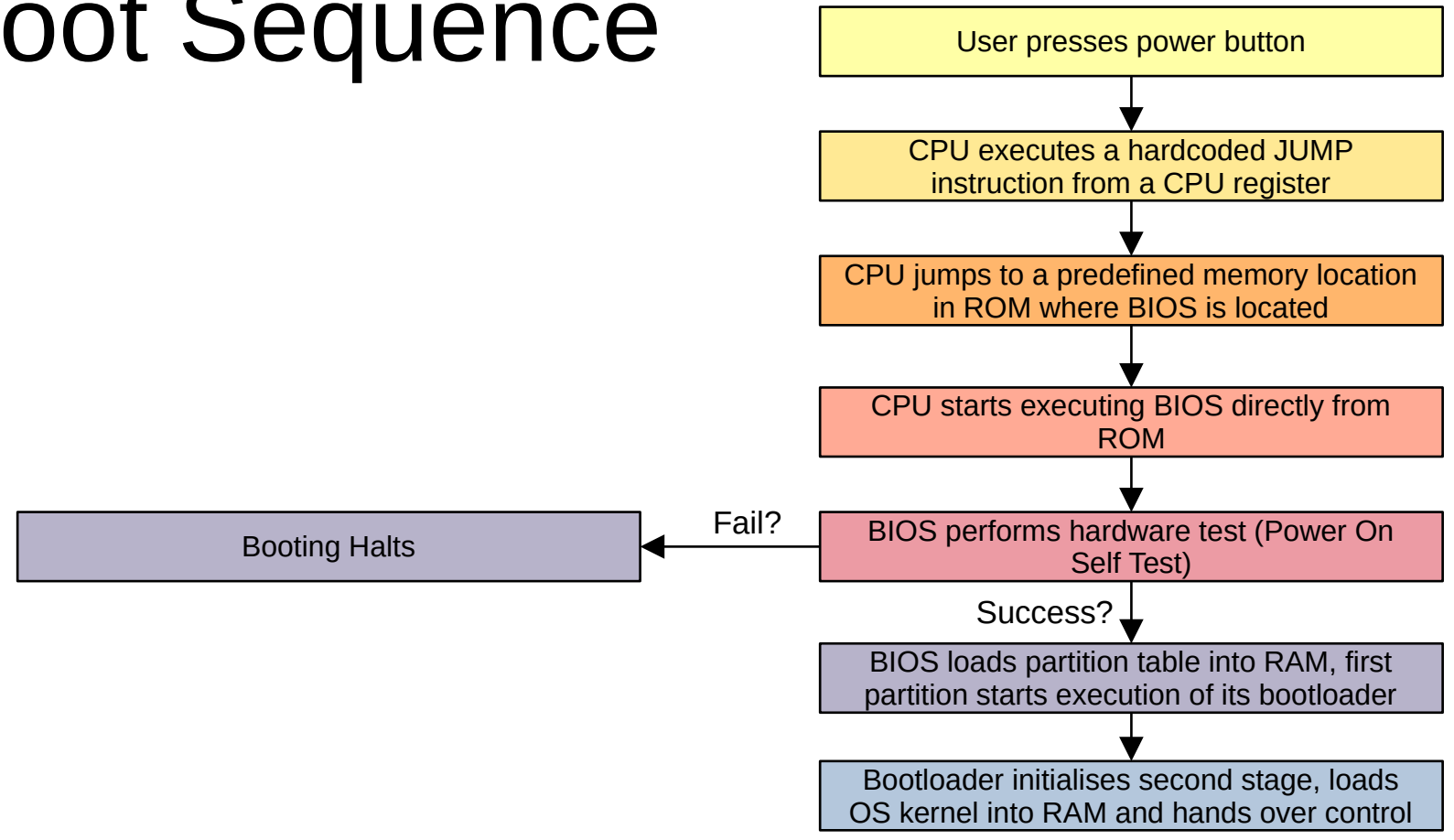
Boot Sequence



Boot Sequence



Boot Sequence



Power On Self Test

- Verifies integrity in BIOS code
- Verify basic hardware components (Registers, timers)
- Verify main memory (RAM)
- Initialise system buses
- Identify connected devices
- Plus other things buts its essentially verifying, identifying and sometime initialising

Bootloader?

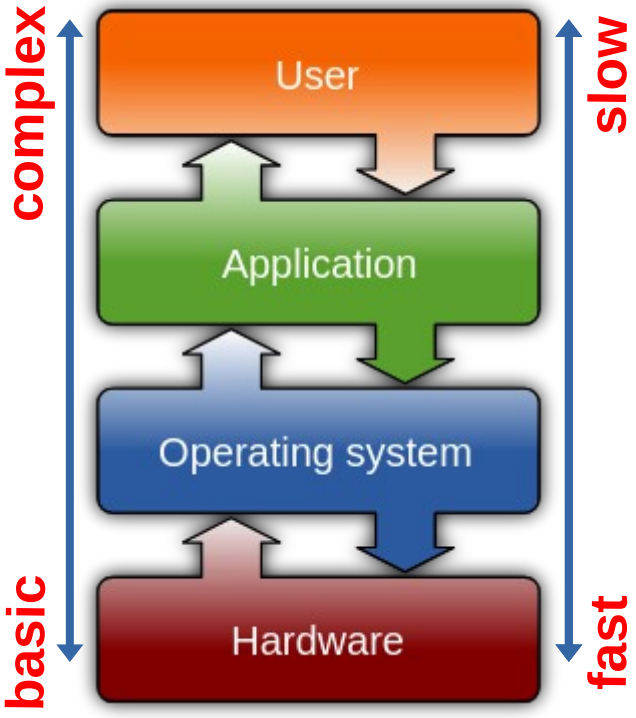
- Stored in non-volatile, read-only memory
- Initialises the different components if necessary, most notably the RAM
- Loads the OS kernel into RAM
- Then hands off control to that OS kernel which will handle everything from then on

Why BIOS in Hardware?

- Most fast memory (SRAM and DRAM) is volatile
- We need to start somewhere
- We could just make all our memory out of non-volatile memory
  - But that’s expensive

Why OS in software?

- Typically, the further up the layers we go, the less time efficient it is
- But also the further up we go, the more concept efficient it is
- Design is always about compromise, computer design is no different
- The OS can provide all sorts of powerful features allowing for concurrency, multi users, security, data management, pretty pictures



But Some Software Becomes Hardware

- MMU used to be just software lookup
- Cache always requires dedicated hardware, but more of it now and its better managed
- ALU/FPU used to be just regular CPU operations
- Implementing in hardware is quick!

Could OS operations be implemented in hardware?

- No
- Hardware requires specifics to cemented to be worth the trouble
- We want to customise a lot of them OS operations (size of virtual memory, users, access to files)
- So we have to accept the overhead for these powerful features