[Lecture 1, part 3]

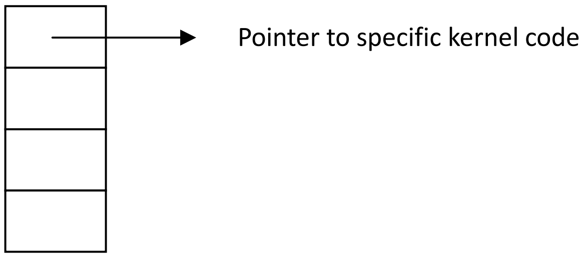
Kernel interaction with the hardware:

Interrupt:

Like input / output, user input, between kernel and device driver.
Devices call back by interrupts.
They are serviced immediately.
And they are user transparent.

Implementation:

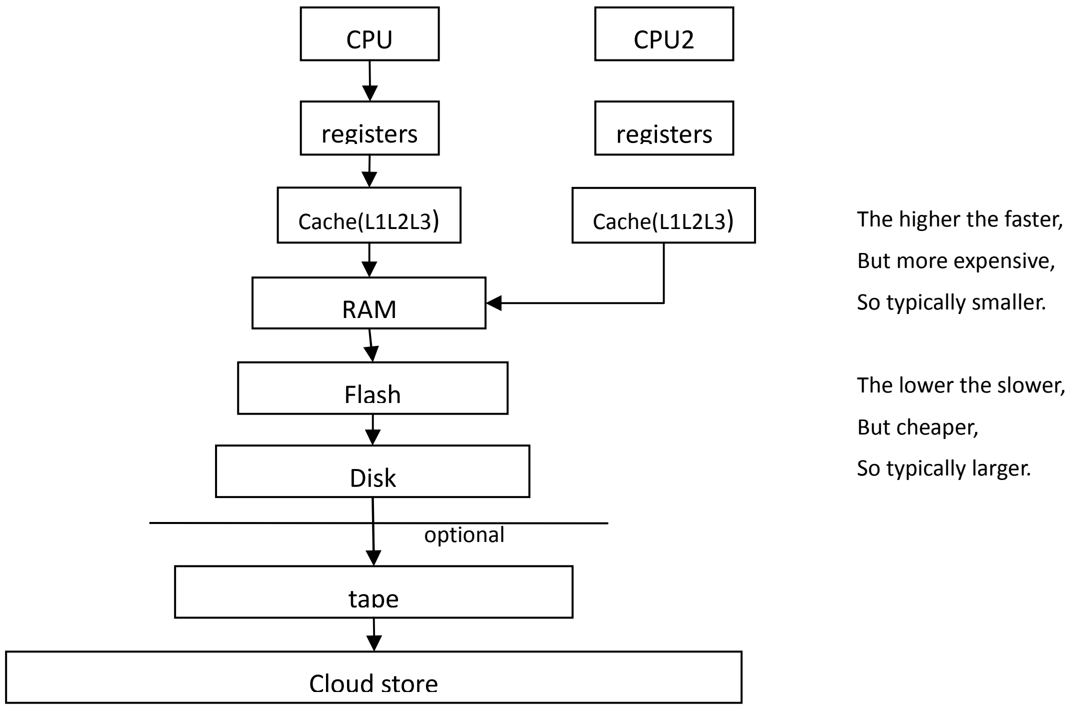
Each device only has limited number of interrupt events.
So we can define an interrupt table
For each element, there is a pointer points to specific kernel code to handle this interrupt.



Typical hardware:

like keyboard, mouse, and some other processors like GPU.

Memory Hierarchy:



Difficulties:

Cache consistency

Cache coherency (for multiple CPU)

Process:

Program in RAM, Instance of a running program,

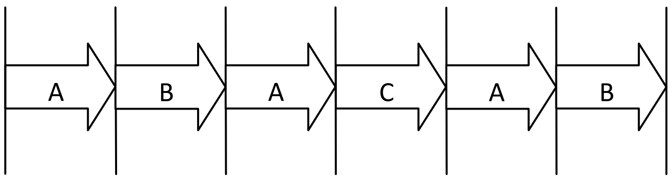
To start a running process, just load the stuff into memory, start the first instruction.

CPU Scheduling:

Divide CPU time to time slices to different processes.

The longer the time-slices are, the less interactive.

But too short will make context switching overwhelming, lost performance.



Implementation:

Set up a timer, call interrupt handler, switch out the process running, load other process.

OS Services:

Design Decision:

Real time operating system: like telesurgery.

Commodity operating system: easy to use, but less interactive, real-time is not as important.

Mechanism: is about How to do things.

Policy: is about What you want to do.

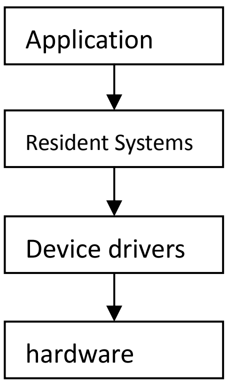
Separation of the two:

Mechanism can be reused, kept when policy was canceled.

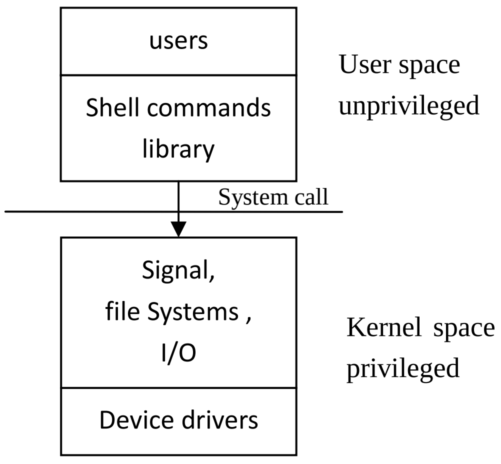
The separation of Mechanism and Policy gives you flexibility.

Structure of real word OS:

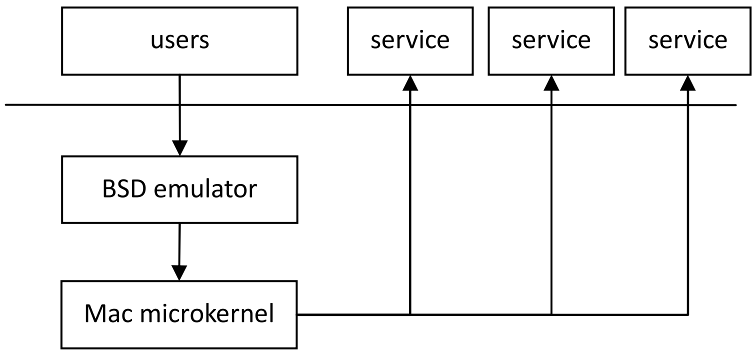
MS-DOS



UNIX



Mac OS X



Microkernel:

Push as much services as possible to user space.

Features of microkernel

Easy to swap out modules

Security, failed modules will not affect kernel.

Less code run in privilege mode, which makes it slower though.