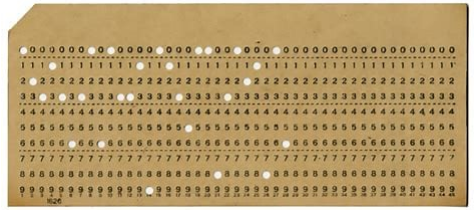
Operating Systems

A little history

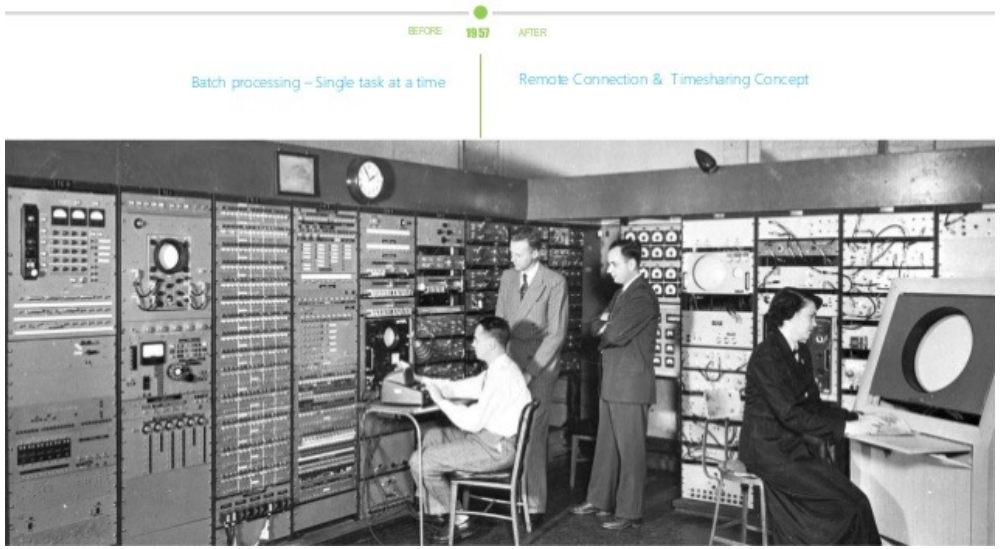
- Early computers were initially single user systems and evolving from hard wired programs to punched cards, magnetic tape and other methods of feeding a new program into the computer system.
- Using a computer meant signing up for a time slot Actually walking up to the computer center at the appointed time, sitting in front of a single monitor and working on your program.
  - Each user was allocated a minimum 15-minute slot
  - They usually spent 10 minutes in setting up the equipment to do their computation ... By the time you got your calculation going, you may have had only 5 minutes or less of actual computation completed—wasting two thirds of the time slot.

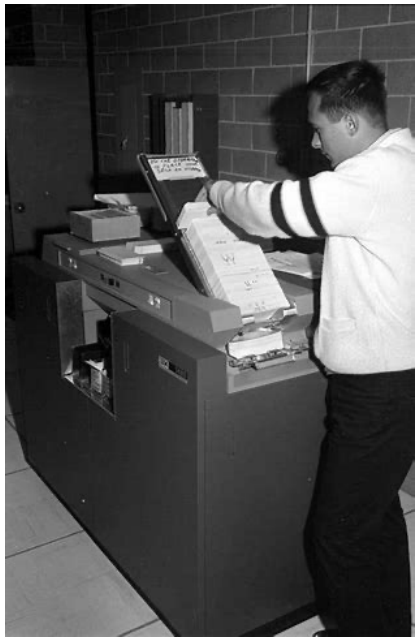
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.1524&rep=rep1&type=pdf



Costs …

- The economic cost of all the setup time was staggering. Companies were charging $300 per hour for computer time on an IBM 701 in 1954 dollars, that's $2,866 in 2020 dollars!
  - The first operating systems were designed to fix this problem.
  - Automatically load a new program when the currently executing program completed or crashed.
  - This evolved into a revolutionary OS mechanism called ‘Batch Processing’





State of the art … circa 1970s and back again

- Timesharing systems
  - Remote terminals wired to mainframes
  - Your good friend the command line/ console!
- Early real-time systems
  - Tiny (often custom) OS's to run machines .. .like the Apollo guidance system!
- Centralized computing, and all-control to the OS was the approach
- In the 80s, Personal Computers drove a push for simpler (User) Interfaces (vs. interfaces for experts), and Computing became decentralized
- Today? OS and computing trends are shifting again
  - Centralization: Cloud computing
  - Costs: Hourly prices range from $0.011/hour to $0.27/hour ($94/year to $2367/year).

https://aws.amazon.com/emr/pricing/

Types of Operating Systems

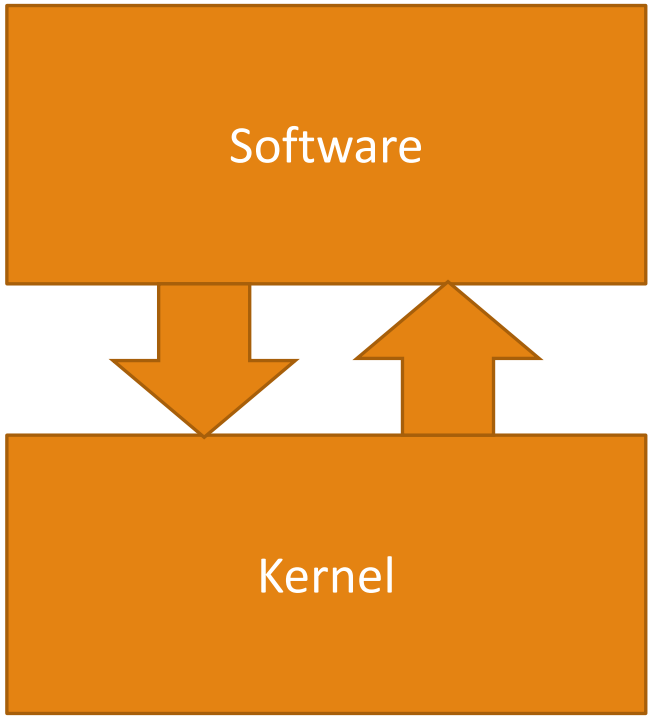
Whatever the user model, or commercial model, the Operating System drives the capabilities of the computer system.

The OS capabilities often set boundaries for your application architecture

Types of OS architectures

- Monolithic systems
- Layered systems
- Microkernels

Monolithic

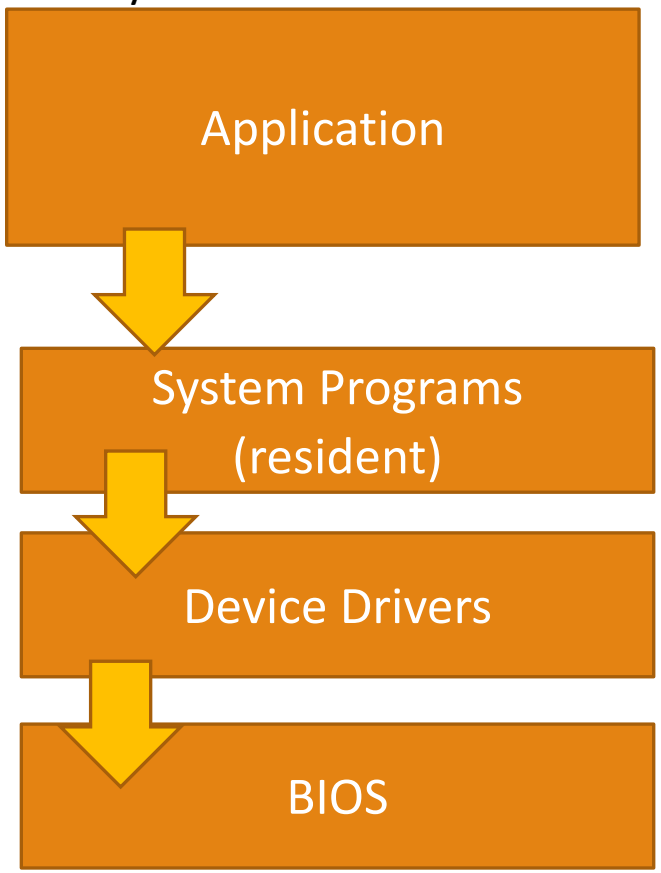


- Early computer systems
- Basic embedded controllers circa 1980s

[Semi]-Layered

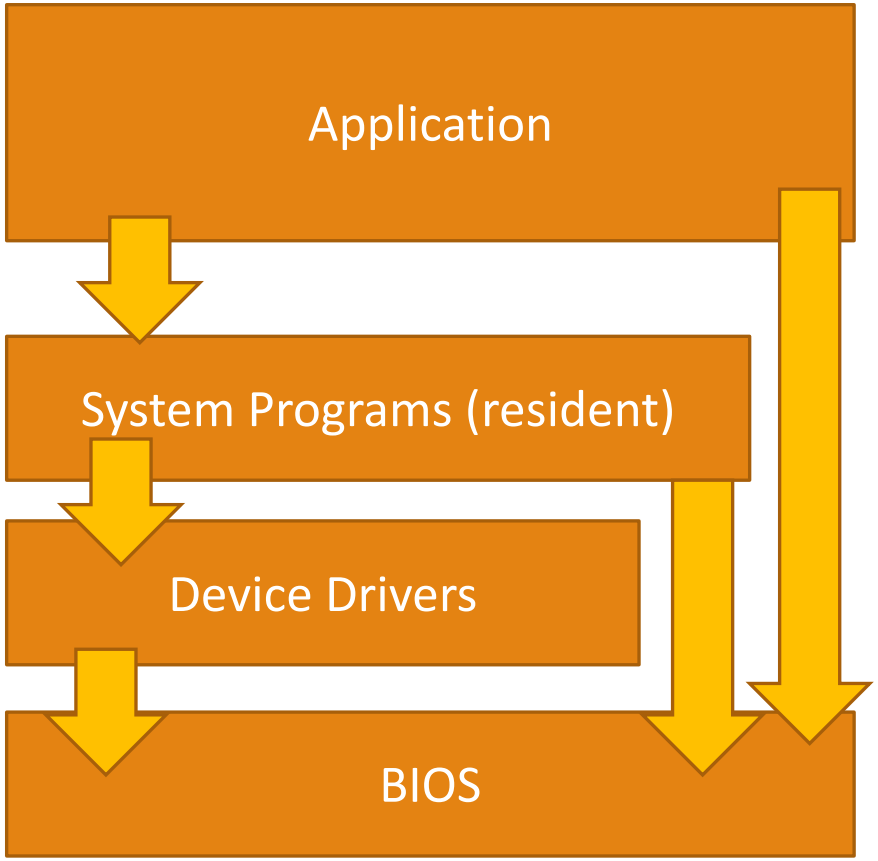
Each layer talks to the next layer

Theory



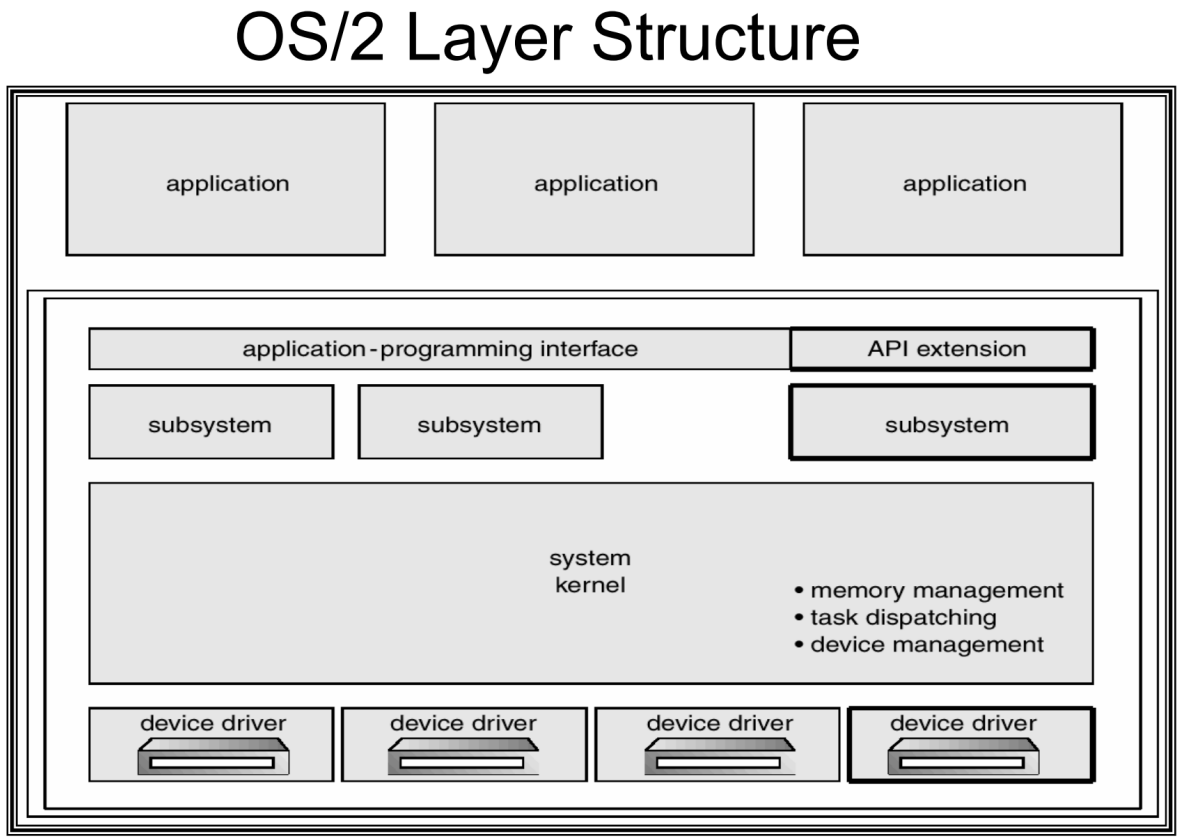
Basic Input/ Output Services

DOS



Basic Input/ Output Services

Improved OS architectures



Enforced layers

Advantages?

- Isolation
- Stability

Kernel Space and User Space …

Memory:

- User space [Applications run here, cannot access kernel]
- Kernel Space (protected) [Core low level OS functions with full access to HW]

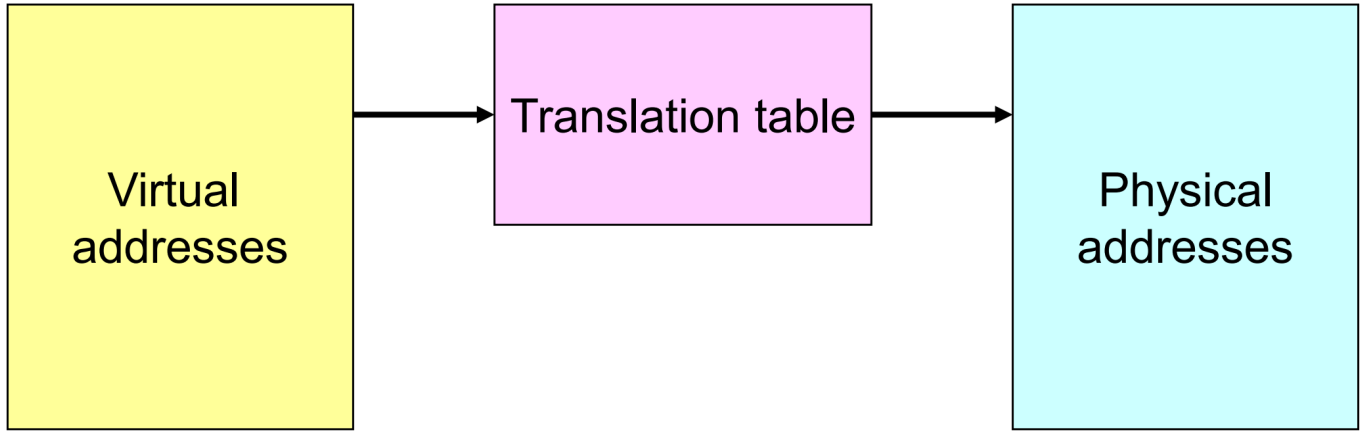
Modern OS's eliminate dangers of programmer error by preventing Applications from getting to Kernel space – this is why you can no longer take down the ENTIRE computer if you are app crashes. That, plus memory management enable isolation per process

This enables multi-process programming

Multi-programmed OS With Memory Protection

- Memory protection keeps user programs from crashing one another and the OS
- Two hardware-supported mechanisms
  - Address translation
  - Dual-mode operation
- Each process is associated with an address space, or all the physical addresses a process can touch
- However, each process believes that it owns the entire memory, starting with the virtual address 0
- The missing piece is a translation table to translate every memory reference from virtual to physical addresses

Address Translation



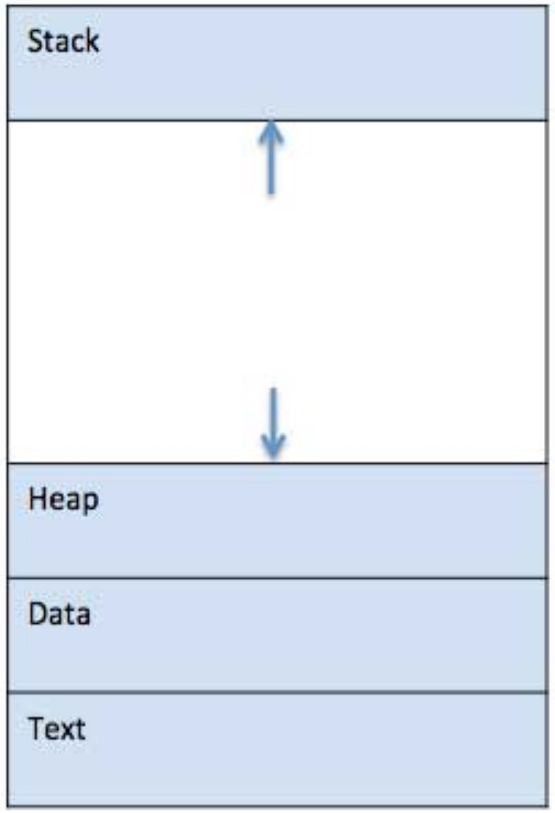
- Translation provides protection
  - Processes cannot talk about other processes' addresses, nor about the OS addresses
- OS uses physical addresses directly
  - No translations

Why do you get kernel panic and blue-screens?

Virtual Memory

What is swap space?

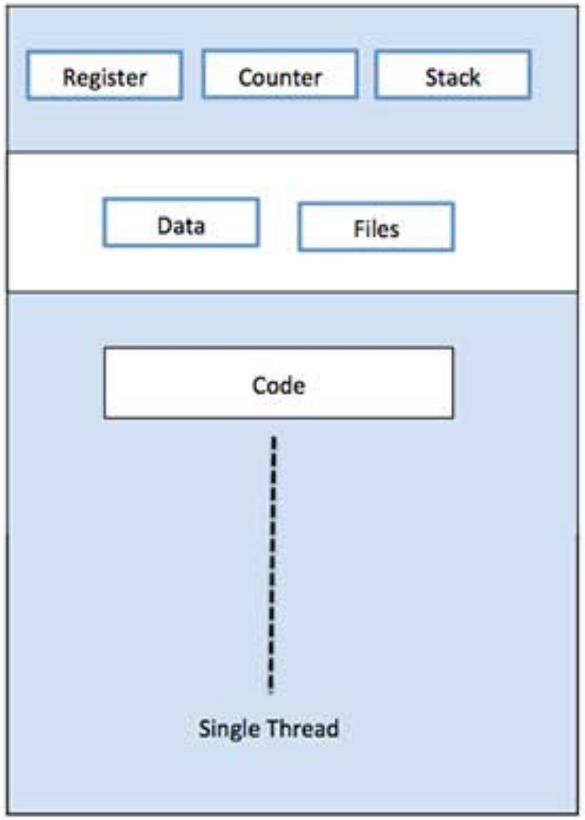
How does an application ‘run’?



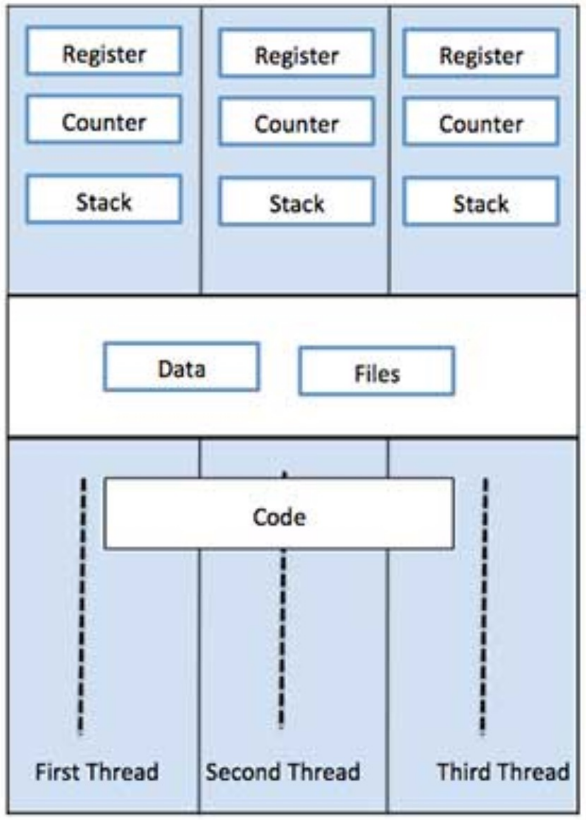
	Component & Description
1	Stack The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	Heap This is dynamically allocated memory to a process during its run time.
3	Text This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	Data This section contains the global and static variables.

Process vs. Thread: The above is a PROCESS!

Process vs. Thread



Single Process P with single thread



Single Process P with three threads

Process	Thread
Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Multi-processing

How does an OS run multiple applications?

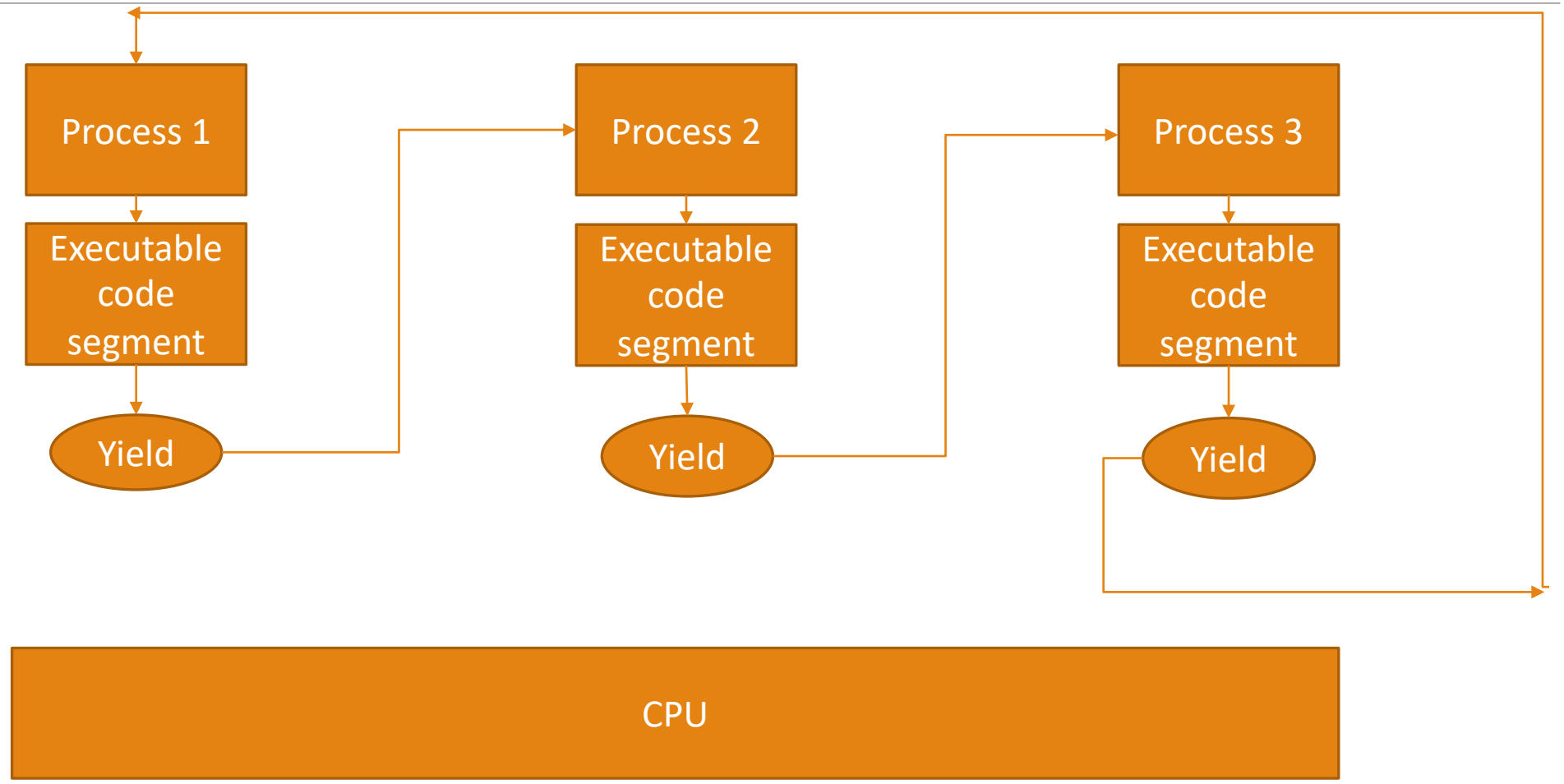
- Co-Operative Multiprocessing
- Timesharing scheduling
- Priority Pre-emptive multi-tasking

Address space isolation (see prior slides)

- Technically, multiprocessing can occur w/o this … but watch out

Co-Operative Multitasking

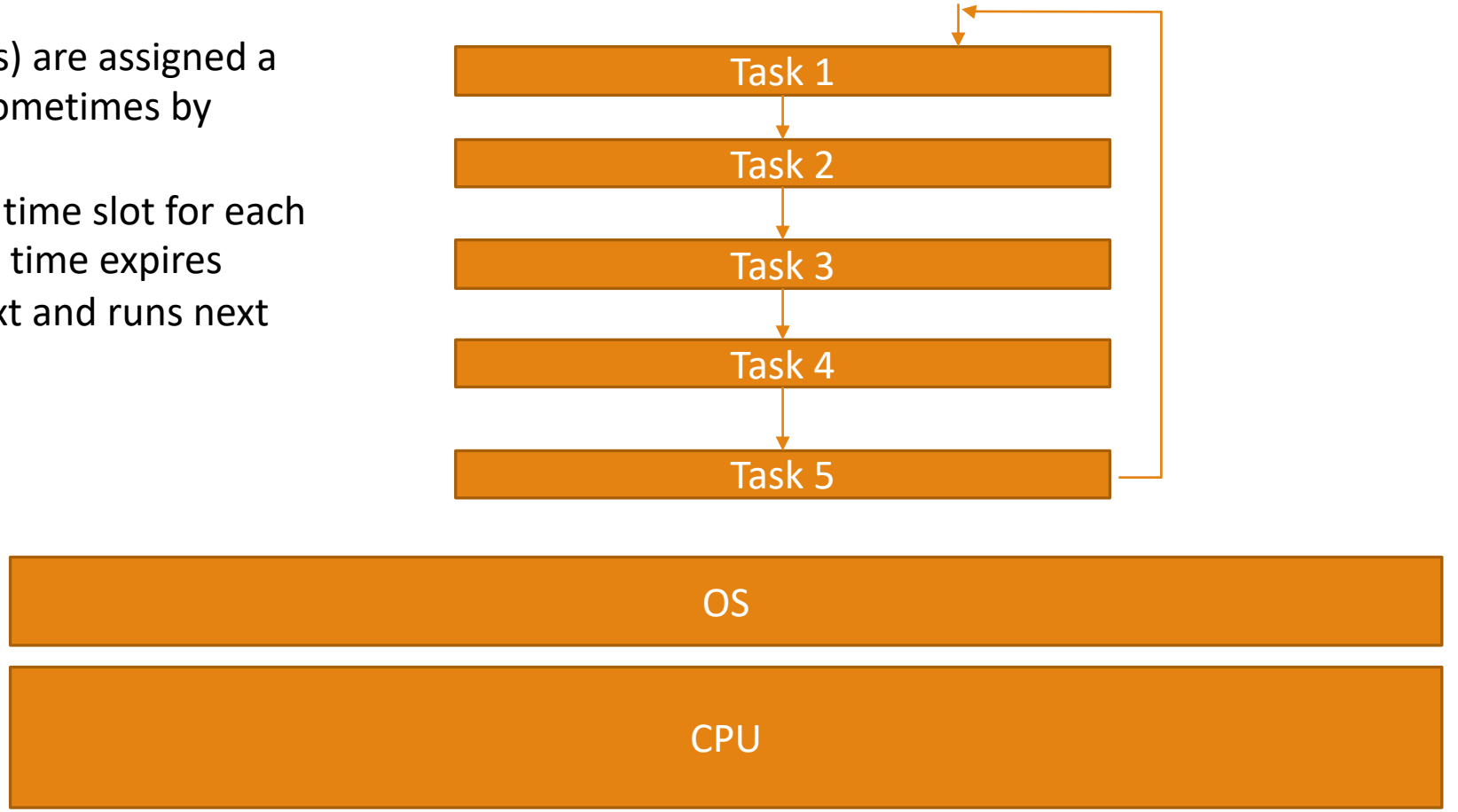
- All processes are memory resident
- Running process has full control of CPU
- It must explicitly 'yield' control to allow another process to run



e.g. Windows pre-Windows 95

Timesharing Multitasking

- Processes (tasks) are assigned a ‘time to run’ (sometimes by priority)
- OS Schedules a time slot for each
- Runs tasks until time expires
- Switches context and runs next priority process



Priority Pre-emptive Multitasking

- Processes (tasks) are assigned a priority
- OS Schedules a time slot for each
- Runs tasks until time (per priority) expires
- Interrupt triggers OS
- Switches context (under control of OS) and runs next priority process (pre-empts process, even in the middle of execution)

e.g. modern OSes: Windows, Linux, MacOS

Task ID	Priority
1	1
2	1
3	2
4	3
5	2