FIT1047 – Week 6

Operating Systems

Recap

An operating system

- makes computers easier to use
  - for the end user
  - for the programmer
- provides illusion of multiple processes running simultaneously
  - by virtualising the resources (CPU, memory, disk etc)
  - by protecting system from malicious or buggy programs

Today's Goals

Process scheduling:

when to switch between processes

Virtual memory:

how to virtualise the RAM

Process scheduling (when to switch)

Scheduling policies

We've seen the mechanisms for process switching:

- user and kernel mode
- context switching on timer interrupts

Now we need to look at policies:

- how long is each process allowed to run?

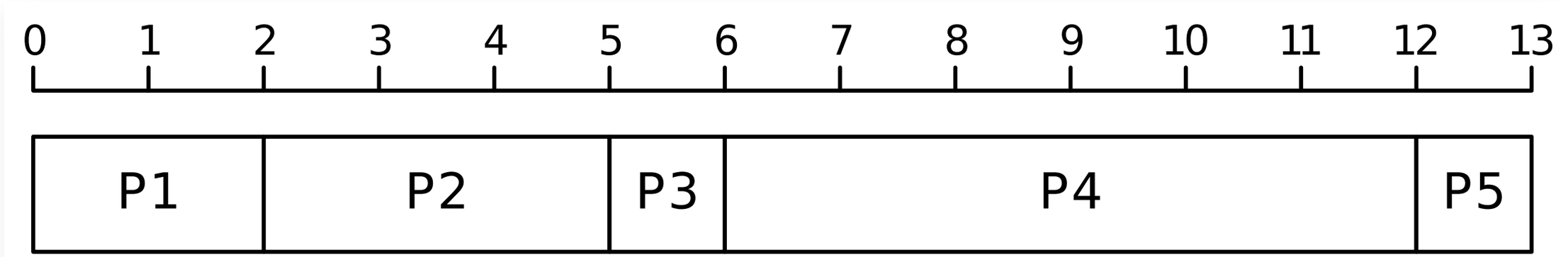
Scheduling

Can aim for different goals:

- turnaround time:
  how long does a process take from arrival to finish?
- fairness:
  all processes get a fair share of processing time

Poor turnaround

Schedule processes in some arbitrary order. E.g. first-come first-served:

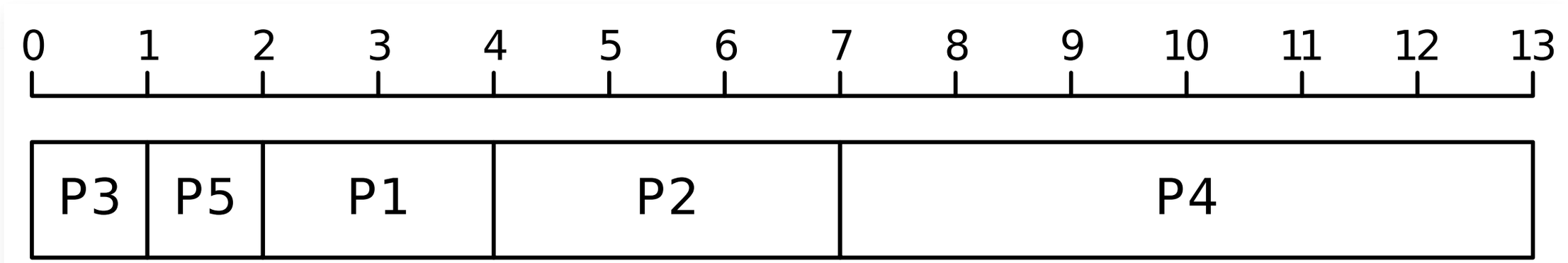


$$\frac{2+5+6+12+13}{5} = 7.6$$

On average processes wait 7.6 time units to complete.

Good turnaround

Order processes by their length, shortest ones first.



$$\frac{1+2+4+7+13}{5} = 5.4$$

Average goes down to 5.4 time units!

(this is in fact optimal)

Turnaround scheduling

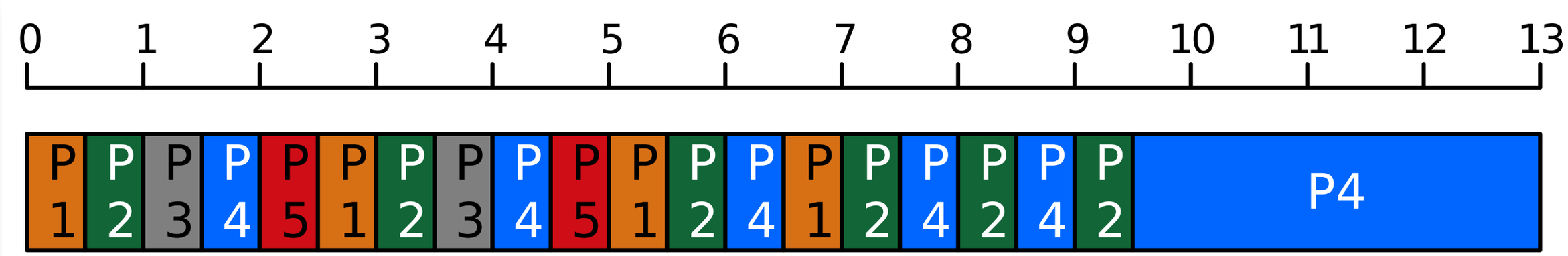
Unrealistic:

- We don't know how long processes will take
- We want to use several processes simultaneously, even if every one of them takes longer

Fairness

Allocate time slices for each process, schedule them in a Round-Robin fashion:

Fairness



The shorter the slice, the fairer the schedule!

But: context switches take time. OS needs to find the right compromise.

Summary: scheduling

OS needs to decide when to run which process.

Modern OSs use a form of round-robin scheduling.

More details in the textbook (if you're interested, won't be assessed).

Virtual memory

Virtual memory (VM)

Reminder: user programs run directly on the CPU.

Can only access I/O through system calls.

How to prevent them from accessing memory of another process?

Virtualise the memory!

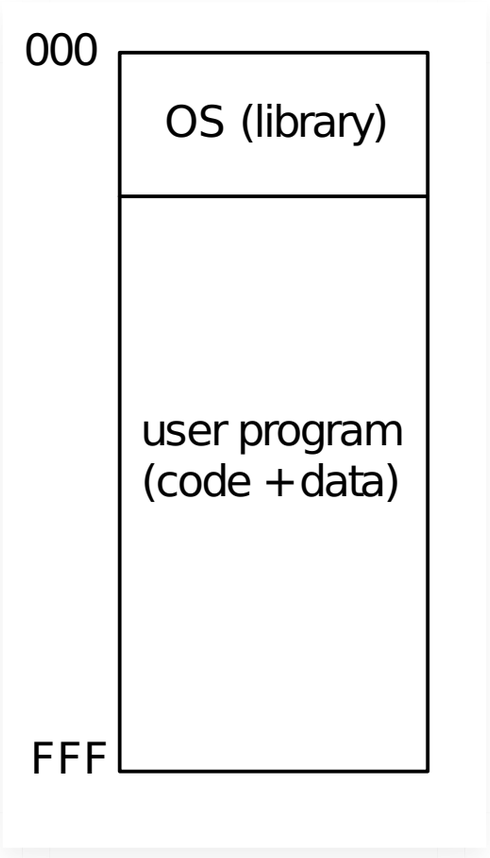
Virtual Memory

Goal: No process can access any memory except its own.

Mechanism:

Each process has its own address space.

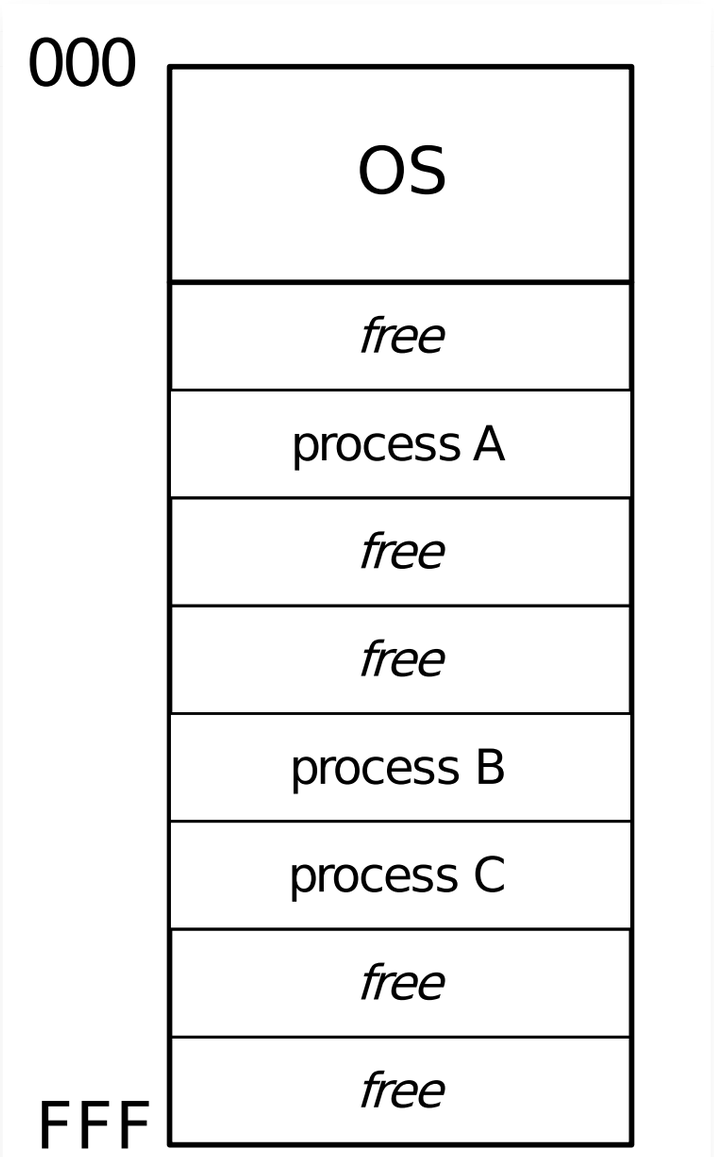
Single process



Early computers:

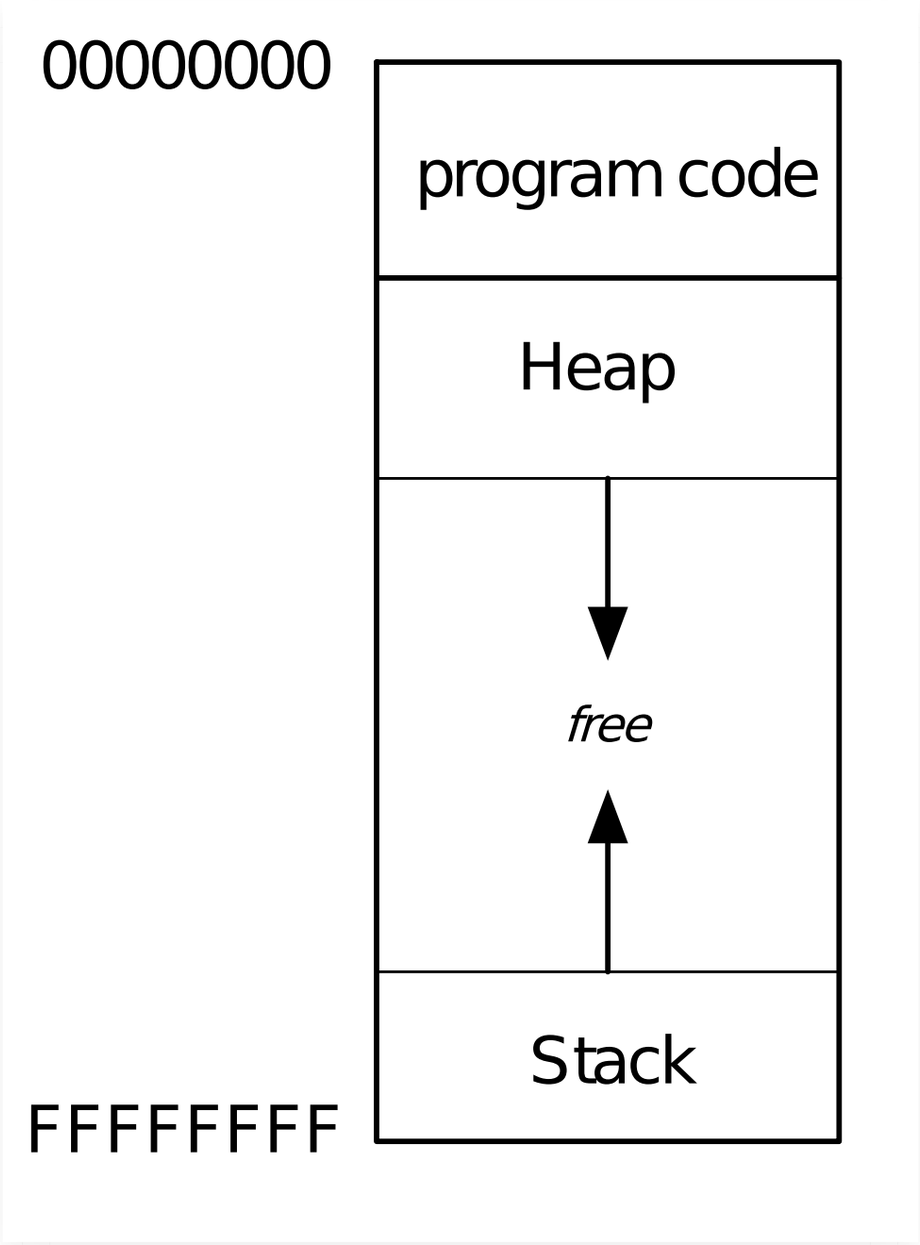
- OS is just a library of subroutines
- Process has entire memory to itself

Multiprogramming



- Every process gets a fixed block of memory
- But for the process it looks like that block starts at address 0
- OS/hardware need to ensure protection

Address spaces



- Each process has a private address space
- Always starts at 0
- Heap: long-lived objects
- Stack: intermediate results, subroutine arguments
- How can the OS/hardware make this work?

Virtual addresses

Each process thinks its addresses start at 0.

But they map to different physical memory locations.

We call the addresses virtual addresses, and the abstraction virtual memory.

The OS and hardware translate between virtual and physical addresses.

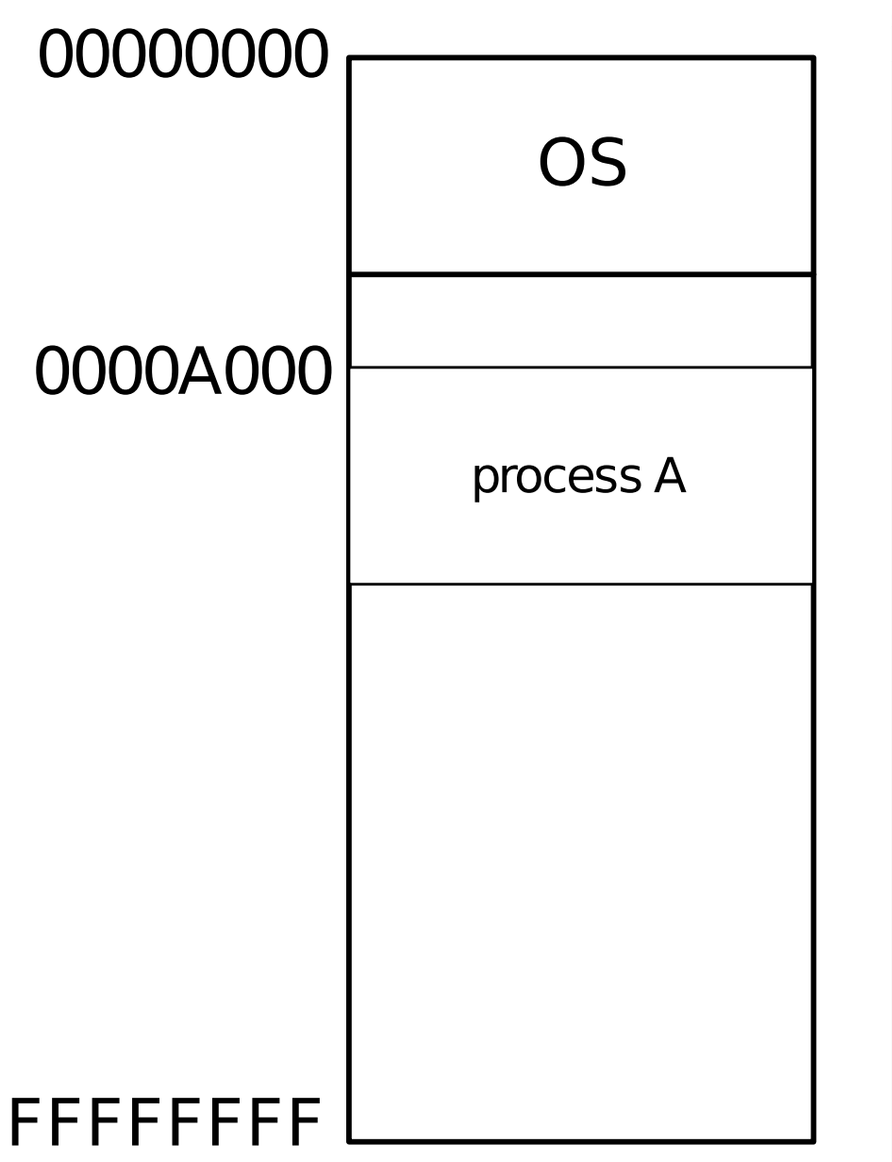
A simple approach

Each process has its own base address X (start address in physical memory).

When a process accesses virtual address Y, the CPU must access physical address X+Y.

- Add a new register B (base register)
- When OS switches between processes: set B to process base address
- All instructions use B!
- E.g. Load X really means Load B+X

A simple approach




```
Load  200
Add   300
Store 200
Halt
```

When switching to process A:

- Set base register to 0000A000
- Load from 0000A000+200 = 0000A200
- Add from 0000A000+300 = 0000A300
- Store into 0000A000+200 = 0000A200

Memory protection

Make sure processes only access their own memory!

Similarly simple fix:

- Add bounds register
  contains largest address the current process may access
- All instructions check that memory access is between base and bounds
- Otherwise, raise error (like an interrupt) to give control back to OS

Virtual memory

Problems with simple approach:

- Each process only gets a fixed block
- No way to dynamically shrink or enlarge

Modern approach:

- Hardware and OS allocate smaller chunks of memory ("pages")
- Pages are mapped dynamically into the process address space

Huge virtual memory

RAM is limited

- may not be enough for all processes at the same time

Virtual memory is the solution:

- save currently unused pages to external storage (hard disk)
- access of unavailable page raises an error
- OS can swap that page back in from external storage

Works very well if swapping is not too frequent.

Summary

Scheduling:

- Round-robin-style scheduling with time slices to achieve fairness

Virtual memory:

- Each process thinks it is alone (its addresses start at 0)
- Map virtual addresses to physical addresses