ECS518U - Operating Systems
Week 2

Processes

Tassos Tombros

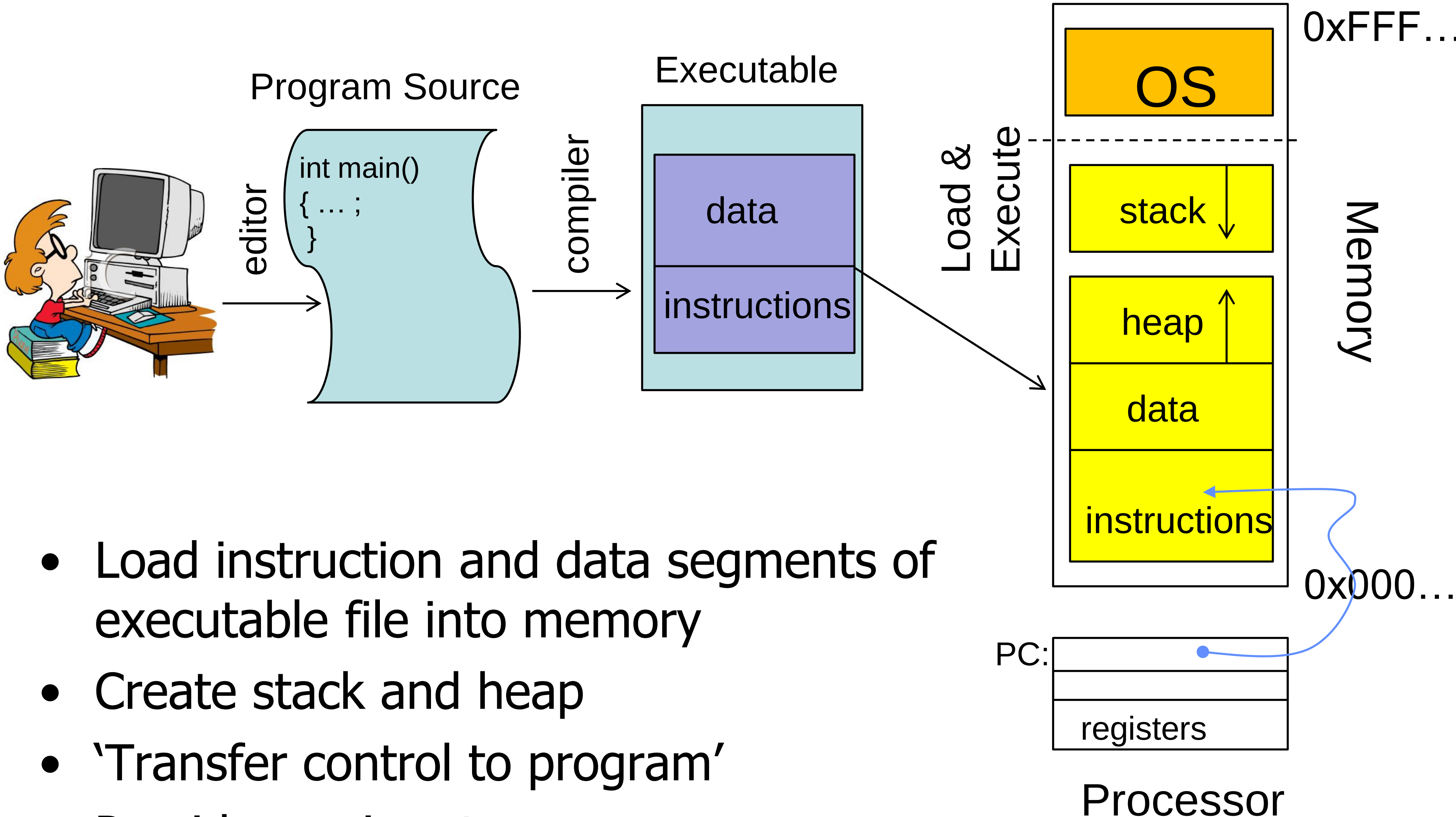
Outline

- Why processes? What is a Process?
- Executing processes
- States of a process
- Data about processes
- Processes and the OS
- Context switching
- **Reading:**
  - **Stallings:** Chapter 3, parts of Chapter 9
  - **Tanebaum:** Chapter 2, sections 2.1, 2.4
- Next week: Scheduling, Process control with PHP

Things you will learn today

- **What** are processes, **why** are they used and what data is kept about processes
- The **illusion** of ‘many things at the same time’: How we switch between processes
- Different **states** in which processes can be (and state transitions)
- **How does the OS protect itself vs. processes** and processes from each other?
- Next week:
  - Process scheduling (choosing which process gets running next)
  - Practical: processes in PHP (fork, wait, exec, signal)
- Later lectures
  - Issues of concurrency: e.g. deadlock
  - Programming with threads

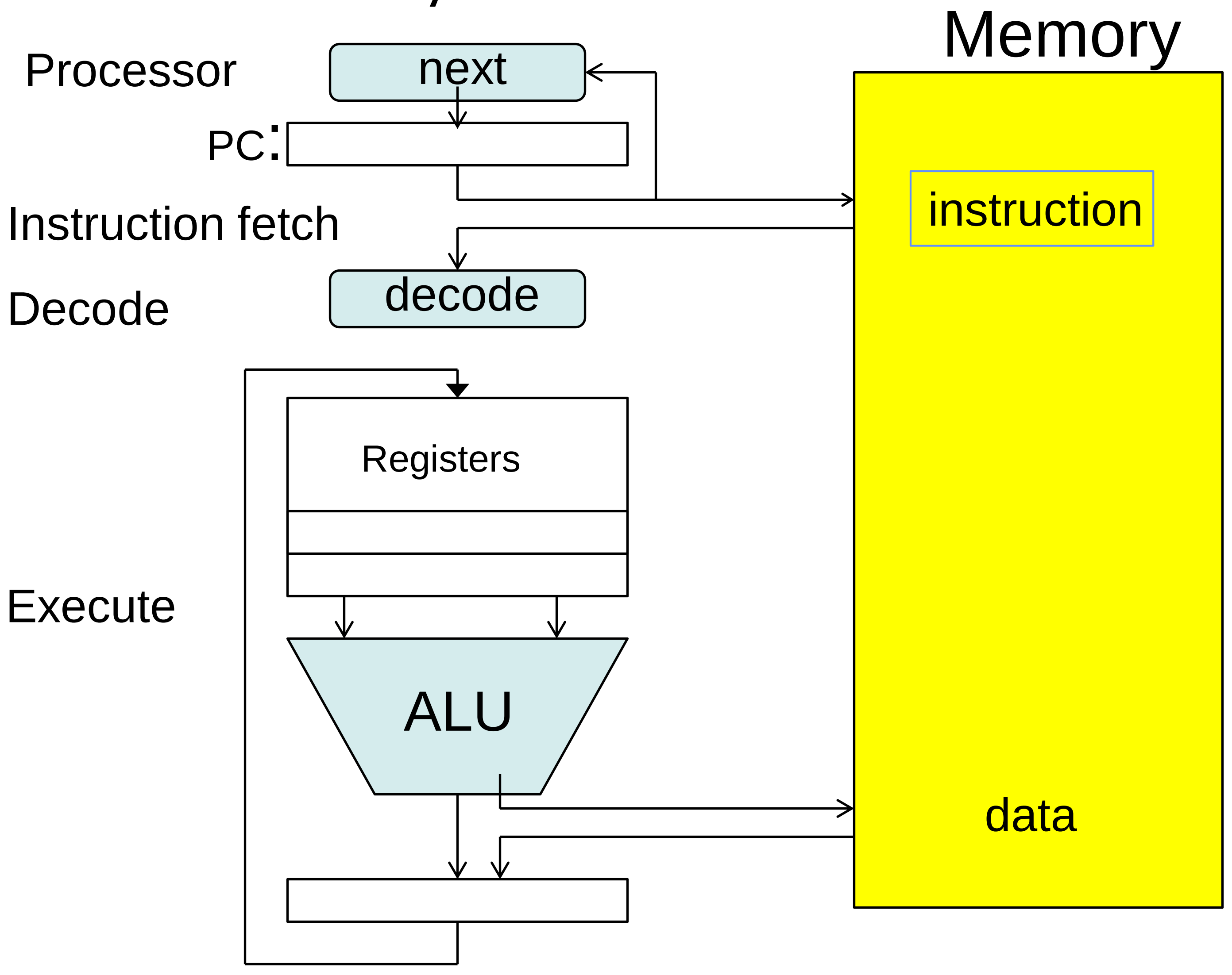
OS Bottom Line: Run Programs



- Load instruction and data segments of executable file into memory
- Create stack and heap
- ‘Transfer control to program’
- Provide services to program
- While protecting OS and program

Fetch/Decode/Execute cycle

The instruction cycle



Why Processes?

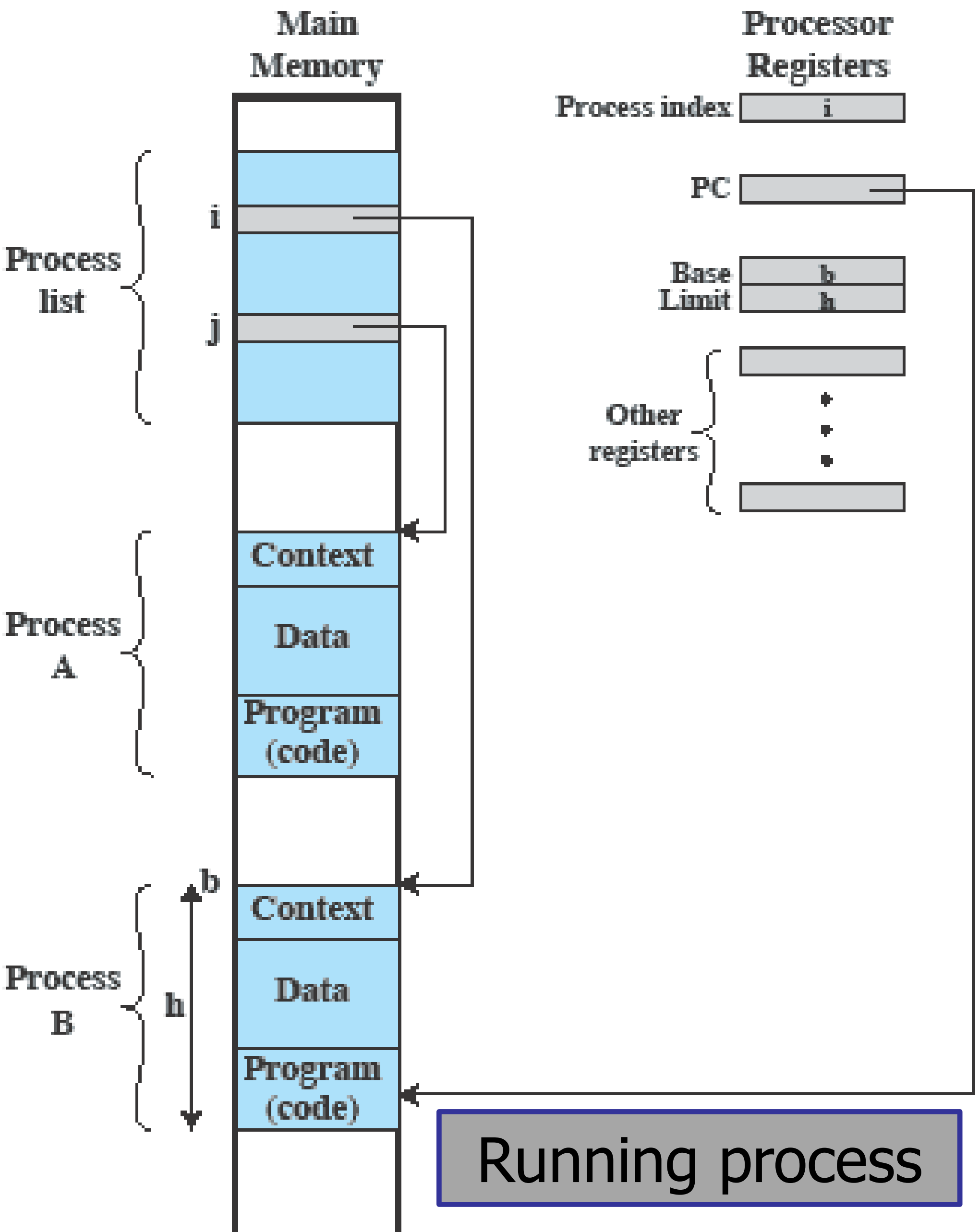
- An OS without processes runs only one program at a time
  - MSDOS – IBM PC
- **Inconvenient**
  - I like to listen to music while coding
- **Inefficient**
  - The CPU is idle when the program waits for I/O
  - Which is the slowest form of I/O?
    - User input

Throughput: getting as much calculation done as possible

What is a Process ?

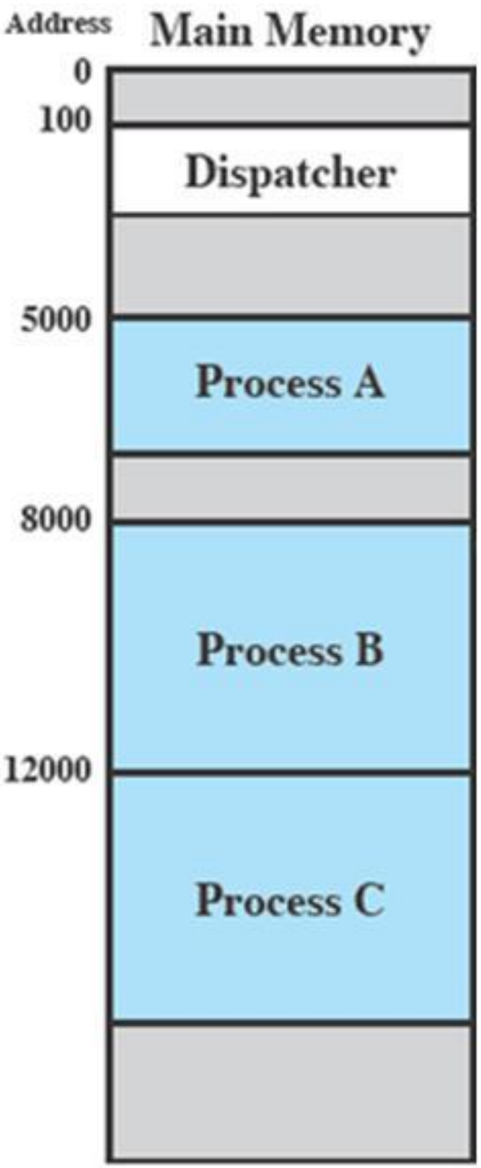
- **A *process* is an instance of a running program**, with
  - **current state** (memory contents, program counter, stack/heap pointers, other registers, data, etc.)
  - system **resources** (amount of memory, files it is using, etc.)
- Therefore the OS must keep data about processes
  - How many processes are there?
  - Is the process waiting – what for?
  - Has the process opened a file?
- Why does the OS need this data?

Processes in Memory



- **Program code**, read in from file when the program is executed
- **Data** (global & static variables, heap, stack, etc.)
- **Process context** is data the OS needs to manage the process (e.g. **Process Control Block**, discussed later)

Executing Processes

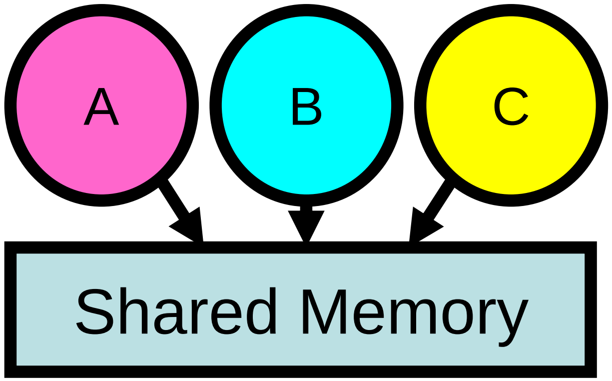


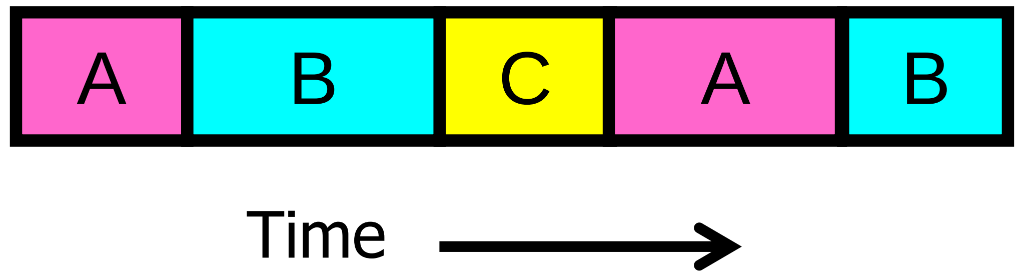
- **From the point of view of the process**, the process runs to completion in sequence
- **From the point of view of the user**, the computer appears to run multiple programs at once
- **From the point of view of the CPU and the OS**, the picture is different

Processes A, B, C in memory
  - 5000: starting address of program of process A
  - 8000: starting address of program of process B
  - 12000: starting address of program of process C

- **Dispatcher** (OS)
  - Switches between processes

Executing Processes





- Assume a single processor. How do we provide **the illusion of ‘running multiple programs at once’**?
  - Multiplex in time!
- Each process needs a structure to hold:
  - Program Counter (PC), Stack Pointer (SP)
  - Registers (Integer, Floating point, others…?)
- How to switch from one process to the next?
  - Save PC, SP, and registers in current process control block
  - Load PC, SP, and registers from new process control block
- **What triggers process switch?**
  - Timer, voluntary yield (e.g. sleep()), I/O, other things

The Basic Problem of Concurrency

- How to give the **illusion of executing many processes at the same time**?
  - Let processes think that they have exclusive access to shared resources (CPU, DRAM, I/O, etc.)
- New problems
  - Which processes could be run? **Process states**
  - OS must protect itself from user programs and must protect user programs from each other (**Protection**)
  - How to choose which process to run next? **Scheduling** (next week)
- Unprotected models were common until not so long ago:
  - Windows 3.1/Early Macintosh (switch only with yield)
  - Windows 95—ME (switch with both yield and timer)
  - Unprotected model: Each process can access the data of every other process (good for sharing, bad for protection)

LAB REVIEW / FEEDBACK

- **ITL culture**
- Why do we ask you to use Linux via the shell?
- **Use the `man` pages** for Linux commands
- Some points to highlight:
  - Pipes, e.g.: `ls -l | wc` (how does Linux actually implement pipes?)
    - By passing the output of one program as input to the other one

  - File permissions and chmod

`-rw-r--r--. 1 tassos staff   1325 Jan 11  2016 confidential.txt`

What will `chmod 666 confidential.txt` do?

It will make the file not so confidential – everyone will be able to read and write

Make sure you are comfortable with basic commands (`ls`, `cd`, `cp`, `mv`, `etc.`) and be careful with `rm`, `rmdir`

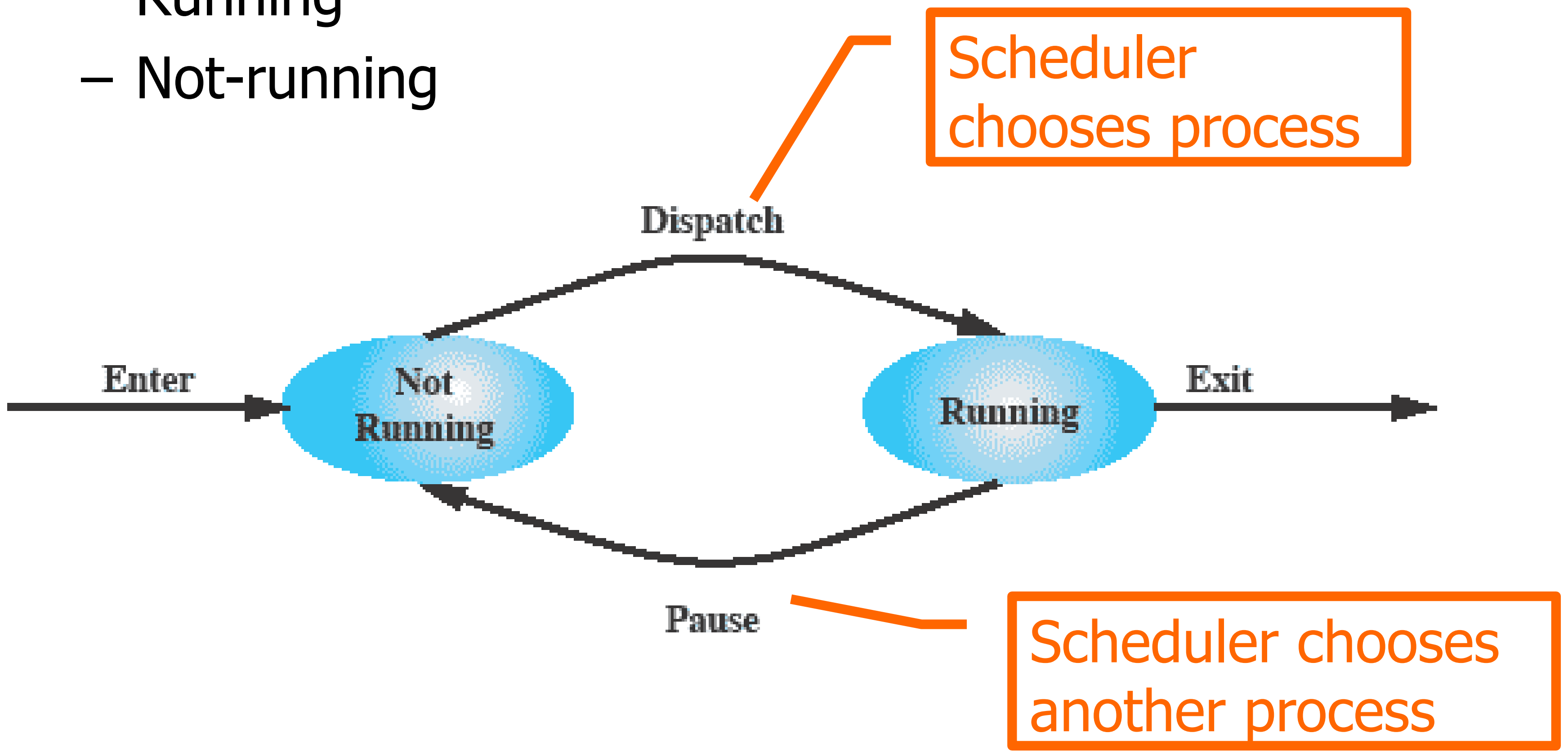
States of a process

- Processes are in different states (e.g. at least **running**, **not running**, but more…)
- Some interesting questions to examine:
  - What causes a change of state?
  - Do all processes need to be in main memory?
  - How does the OS track process states?

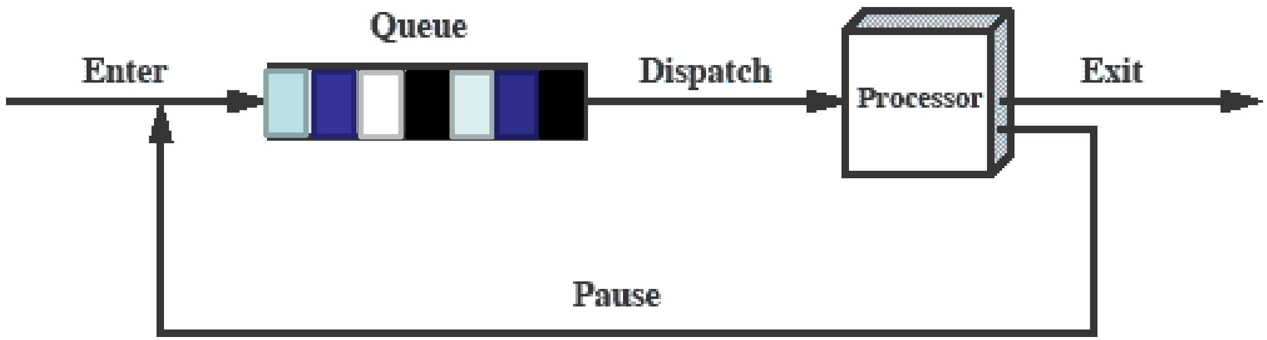
This topic is linked to scheduling – we will examine scheduling in detail next week

Two-State Process Model

- The simplest model
- Process may be in one of two states
  - Running
  - Not-running



OS Has a Queue of Processes

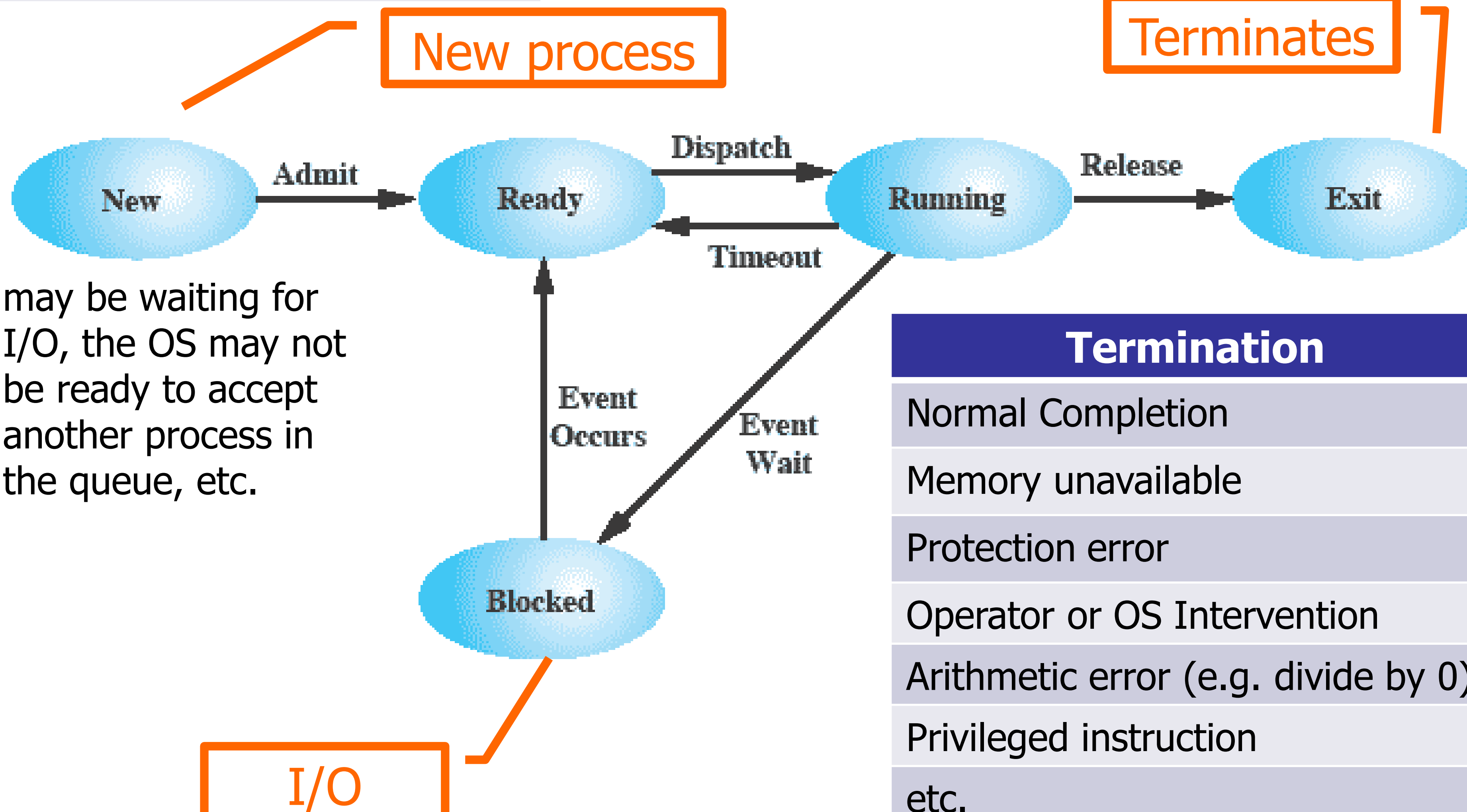


(b) Queuing diagram

- Processes moved by the dispatcher of the OS to the CPU then back to the queue until the task is completed
- Choosing a process for the CPU from the queue is a **scheduling** problem

Five-State Process Model

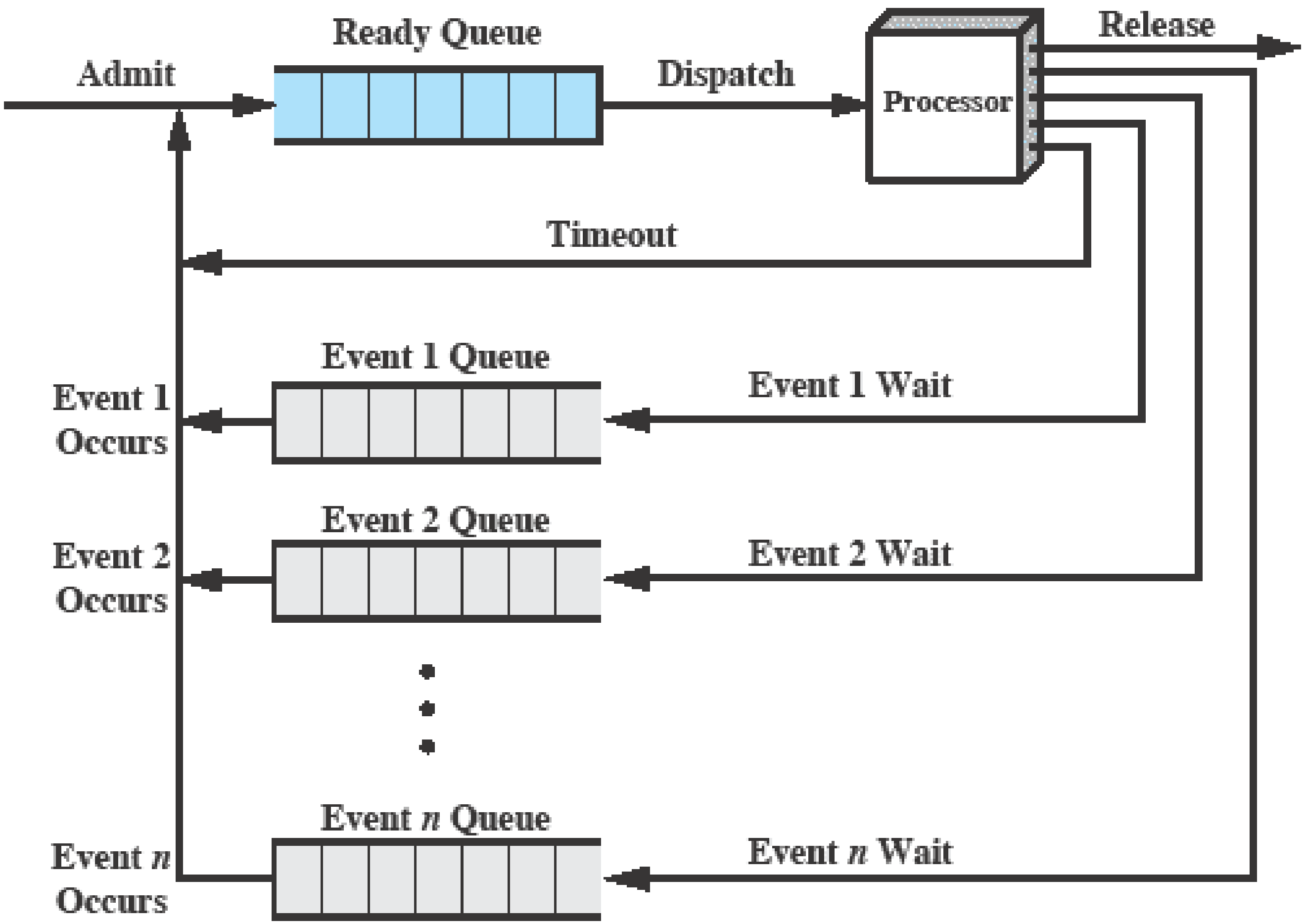
New Process Creation
Login
New batch job
Created by OS to provide service
Spawned by existing process



may be waiting for I/O, the OS may not be ready to accept another process in the queue, etc.

Termination
Normal Completion
Memory unavailable
Protection error
Operator or OS Intervention
Arithmetic error (e.g. divide by 0)
Privileged instruction
etc.

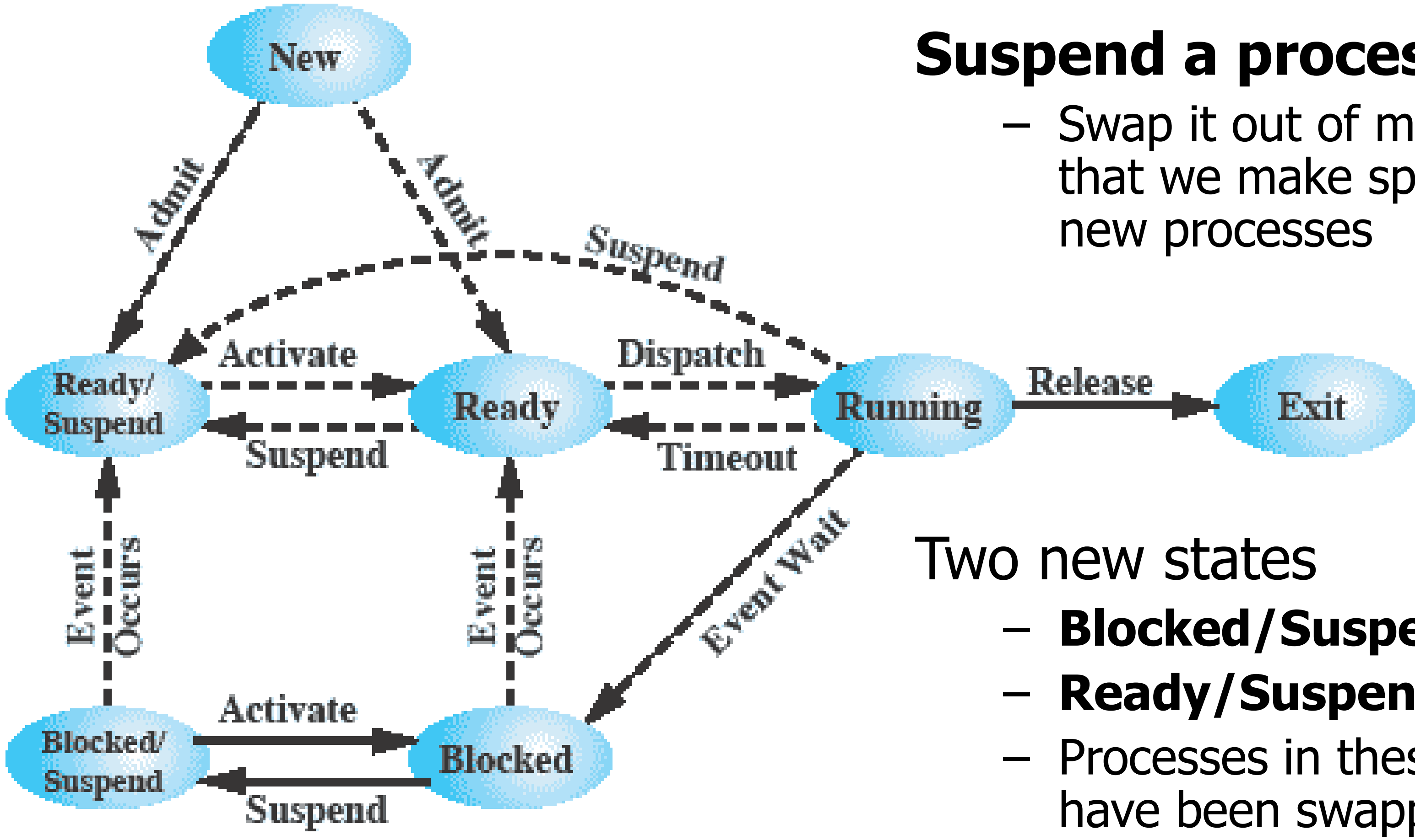
Multiple Queues



Can have one queue of processes for each different type of event:

- queue waiting for disk I/O
- queue waiting for user input
- queue waiting for network response
- etc.

Suspending a Process



Suspend a process

- Swap it out of memory so that we make space for new processes

Two new states

- **Blocked/Suspend**
- **Ready/Suspend**
- Processes in these states have been swapped out of main memory to virtual memory (disk)

Process **priority** is taken into account in many of these transitions (part of scheduling algorithms)

Trivia

- Approx. number of bugs per 1K lines of code? Per 70M? (OSs are very complex things)

  Even a moderate estimate of 1 bug per 1000 lines of code gives a good idea of the complexity. The question is not "Is there a bug?" but "How much damage can it do?"

- What is "core dump" and why is it called like this?

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

  https://en.wikipedia.org/wiki/Magnetic-core_memory

- Tabs in Chrome

  Every tab in Chrome is actually a separate process.

- Who is Neil Harbisson and why is that of any relevance to us?
  - https://www.ted.com/talks/neil_harbisson_i_listen_to_color
  - http://www.dazeddigital.com/artsandculture/article/31102/1/why-this-artist-got-an-antenna-implanted-in-his-skull

  Why is he mentioned here? Hard to imagine personal computing getting more personal!! Relevant to the slides about changing trends in computing from Week 1

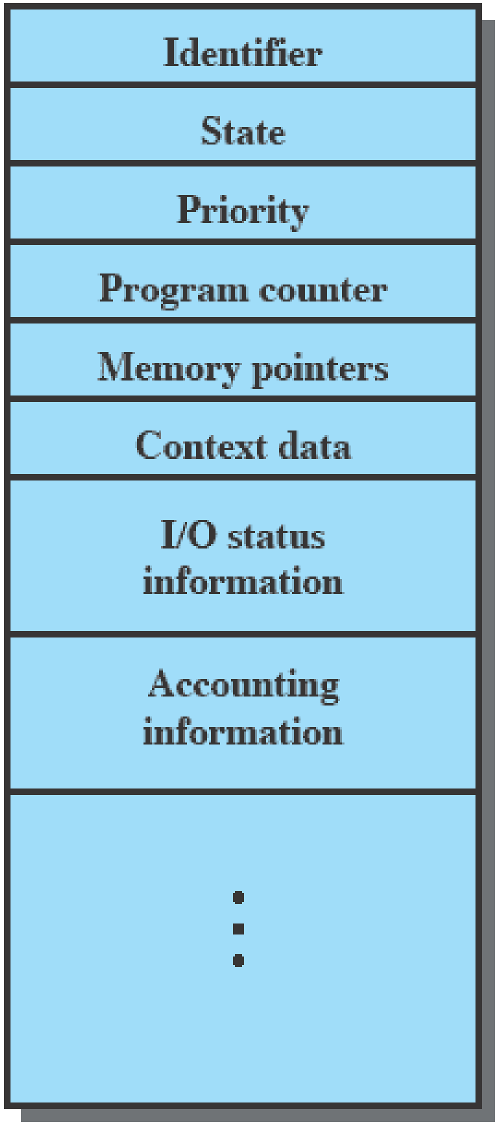
Process Control Block

- The OS needs to keep information about each process
- **Think why?**
- So that it can save the information for process A when it switches from A to B, load up the information for process B, etc.
- **The type of information kept is OS specific**
  - For example in Linux the kernel stores the list of processes in a doubly linked list
  - Each element of the list is of the type **struct task_struct**
  - You can find the definition in <linux/sched.h> (e.g. https://github.com/torvalds/linux/blob/master/include/linux/sched.h)

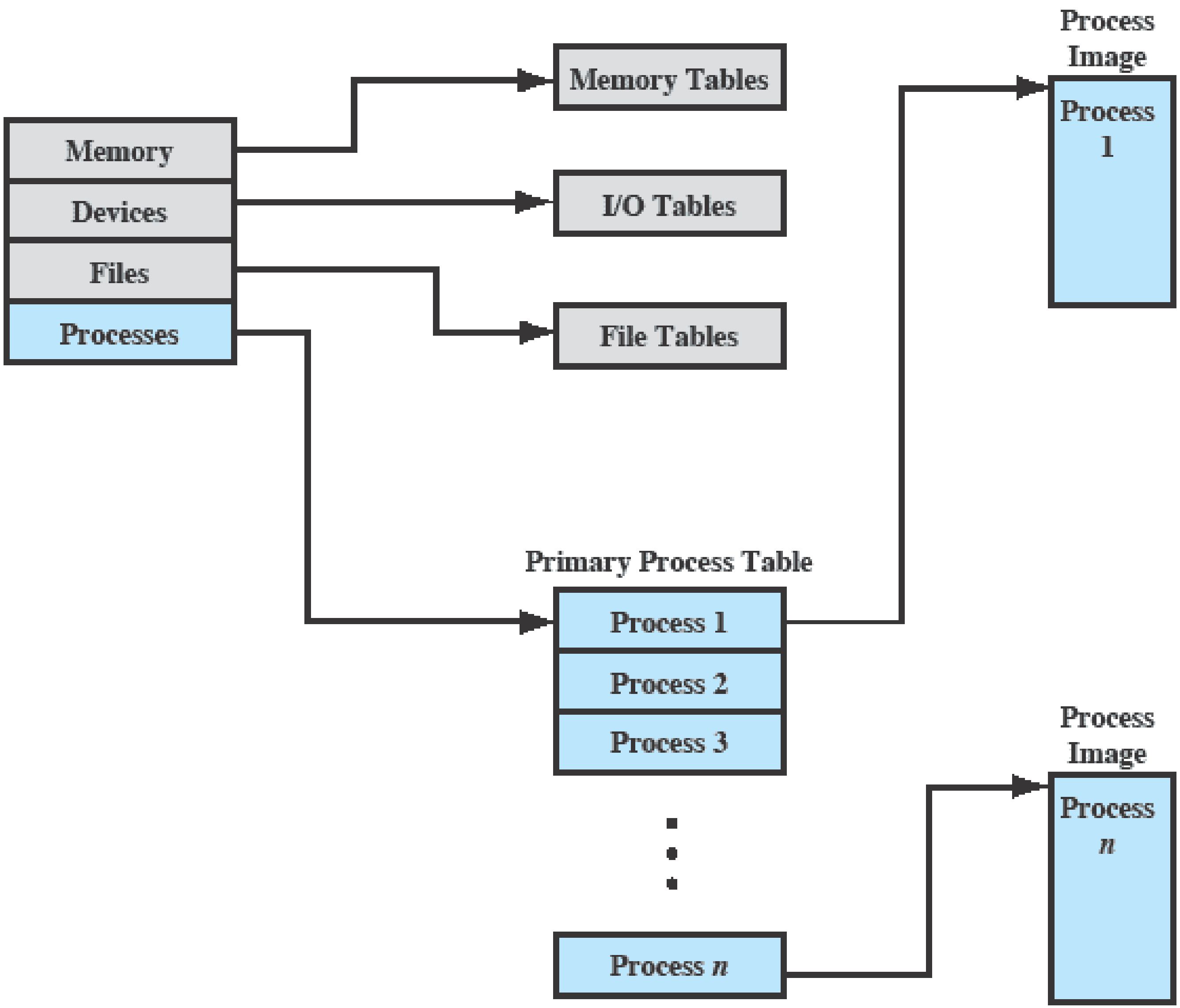
Process Control Block

- **Kernel represents each process as a process control block (PCB)**
  - Identifier of the process (in Unix systems also of the parent process)
  - State (running, ready, blocked, …)
  - Priority – for scheduling
  - PC, stack pointers, values of registers, …
  - Execution time, …
- **Kernel Scheduler** maintains a data structure containing the PCBs
- Scheduling algorithm selects the next one to run

Enough information to resume an interrupted process



OS Control Tables



Finding Process Information

(This is given as task in Lab 2 – check QMPlus)

- ps and top
- Demo
  - ps
  - ps -lf
  - ps -elf
  - top
  - pstree

```
I   Marks a process that is idle (sleeping for > 20 seconds).
R   Marks a runnable process.
S   Marks a process that is sleeping for < 20 seconds.
T   Marks a stopped process.
U   Marks a process in uninterruptible wait.
Z   Marks a dead process (a ``zombie'').
```

Protection

- The **OS must protect itself from user processes**
  - Reliability: compromising the operating system usually causes a crash
  - Security: limit the scope of what processes can do
  - Privacy: limit each process to the data it is permitted to access
  - Fairness: each should be limited to its appropriate share of resources (CPU time, memory, I/O, etc.)
- It must **protect processes from one another**
- Primary mechanism: limit the translation from program address space to physical memory space (to discuss in later weeks)
  - A process can only touch what is mapped into its own address space
- **Additional Mechanisms:**
  - **Dual mode operation** (user mode vs. kernel mode)
  - **System call** processing

User vs. Kernel Mode

- Hardware provides at least two modes:
  - '**Kernel**' mode (or 'supervisor' or 'protected' or 'system')
  - '**User**' mode: Normal programs executed
- What is needed in the hardware to support 'dual mode' operation?
  - a **bit** of state (user/system mode bit)
  - certain operations / actions only permitted in system/kernel mode
    - In user mode they fail
  - User→Kernel transition sets system mode AND saves the user PC, etc.
    - Operating system code carefully puts aside user state then performs the necessary operations
  - Kernel→User transition clears system mode AND restores appropriate user PC and other state data
- **THINK:** Which transition is crucial from a security point of view? (user to kernel, because if the kernel is 'hijacked' by a malicious piece of code, control of the OS (and therefore the device) will be lost)

3 types of mode transfers

- **System call**
  - Process requests a system service, e.g., read()
  - Like a function call, but ‘outside’ the process
  - Does not have the address of the system function to call
- **Interrupt**
  - External event triggers **context switch** (switching the running process), normally independent of the user process
  - eg. Timer, I/O device
  - Why must users never be able to enable/disable interrupts?
- **Trap or Exception**
  - Internal event in process triggers context switch
  - e.g., Protection violation (segmentation fault), Divide by zero, …

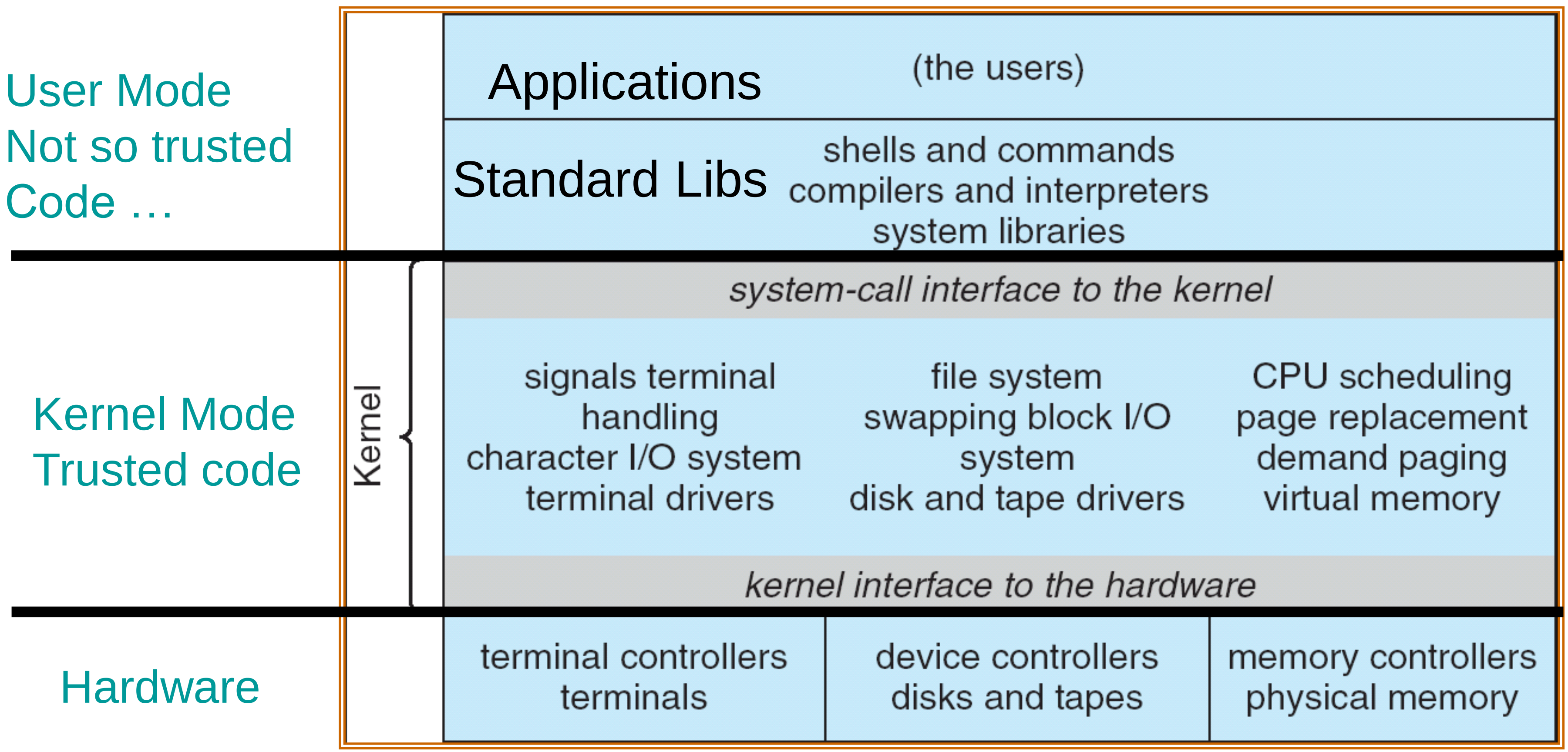
Exam style Question

Explain why a computer can increase its throughput (i.e. get more work done) by working on multiple processes at once, rather than just one. **[4 marks]**

Your answer:

OS and Application Programs

- How does an application process use the OS?
- It goes through the ‘call gates’ (bouncer analogy)



System Calls – call gates

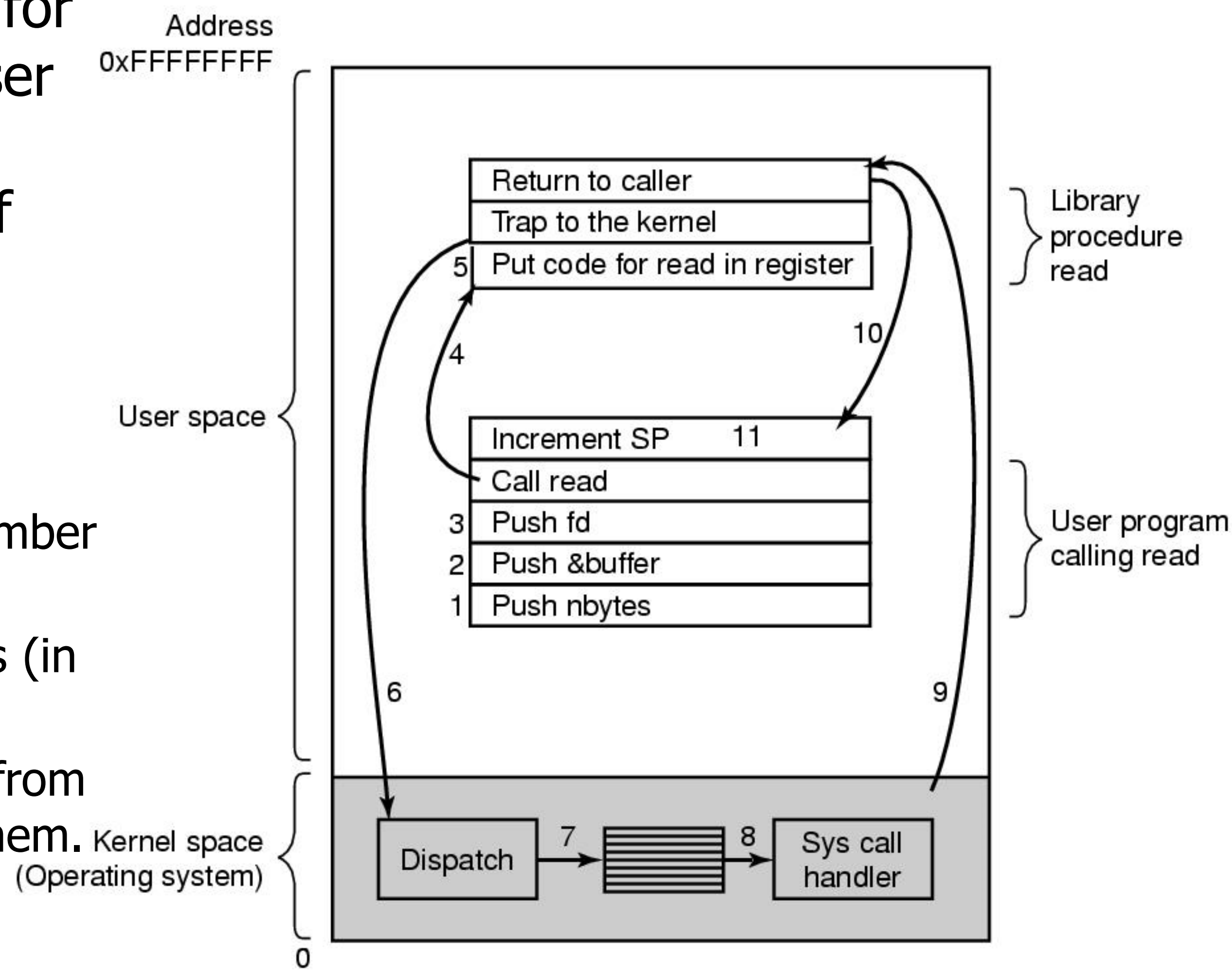
- Programming interface to OS
  - ‘Call’ the OS
- You do not explicitly use a system call in your code
  - You use e.g. a PHP function or a C library function that in its own code ‘communicates’ with the kernel
- Example of an ‘interface’ to a system call
  - ‘read’ bytes from a file
  - fd – file descriptor
  - buffer – place for results

```
C code: read(fd, buffer, nbytes)
Java code: c = inputStream.read()
```

http://linux.die.net/man/2/read

How System Calls Work

- Should be impossible for buggy or malicious user program to cause the kernel to corrupt itself
- Uses a trap
  - Like an interrupt
- **Sys call handler**
  - Table mapping call number to handler
  - Locate user arguments (in user stack)
  - Copy user arguments from user mem. to kernel mem. **(WHY?)**
  - Validate arguments
  - Copy results back into user memory



Context Switch: Switching Processes

- Process switch
  - The OS gains control to co-ordinate the switch
  - Remember, context switch is one of the ways into kernel mode (via Interrupt), so we also have a mode switch

- Process switching issues
  - What events trigger a process switch?
  - Mode switching and process switching
  - OS updates process data structures - overheads

When to switch processes

Process switching events

Mechanism	Cause	Use
Interrupt	External to the execution of the current instruction	Reaction to an asynchronous external event
Trap	Associated with the execution of the current instruction	Handling of an error or an exception condition
Kernel call	Explicit request	Call to an operating system function

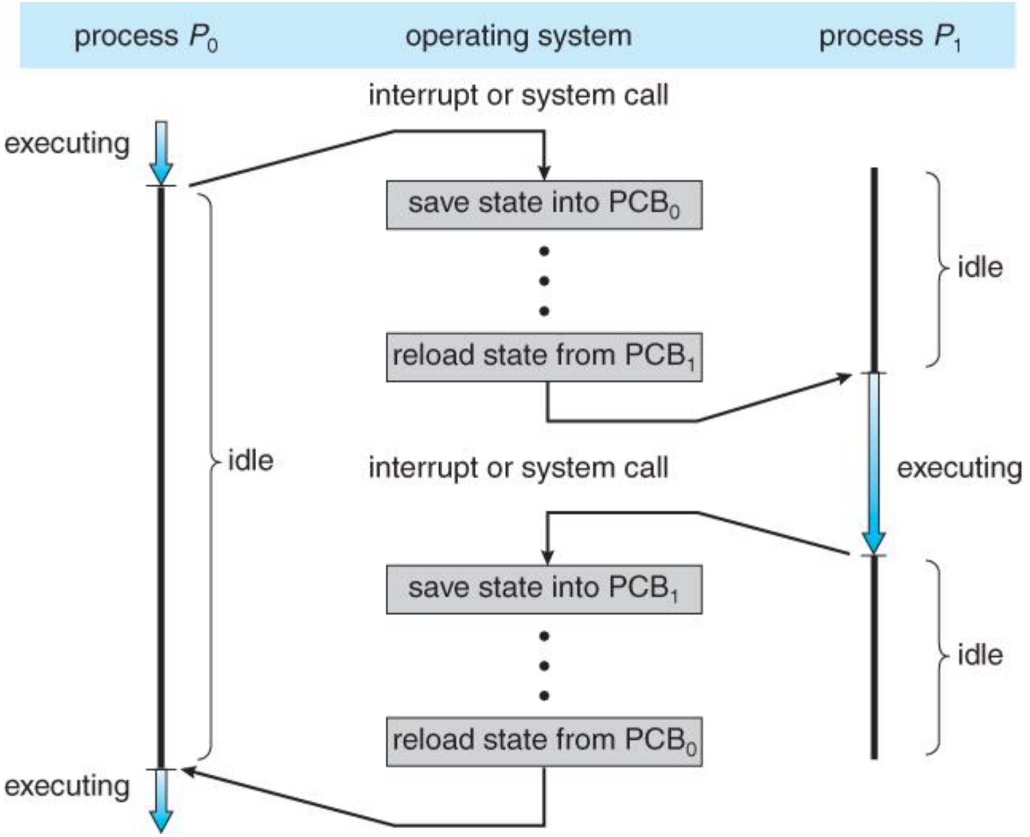
Change of Process State

1. Save program counter, stack pointer and other registers (context)
2. Update the process control block (PCB)
3. Move process to queue
   - ready; blocked; ready/suspend
4. Select new process for execution
5. … update its PCB
6. Update memory-management data structures
7. Restore context of the selected process

Cost of Process Switch

- Lots of data to store/load
- Scheduling
- Cache reloading
- Fast compared to I/O
- A mode switch (w/o process switch) is quicker

→ 1000s of cycles



Look ahead: Lab 2

- First PHP scripting exercise
  - Sample code provided
  - Extend sample code to implement simple tasks (checking which of your files in a directory are larger than xyz Kbytes via different methods)
- Preparation
  - Check the online PHP manual (QMPlus ->Resources ->PHP and Java)
  - Type the sample code and run it, understand how it works
- **Remember:** you need `vminstance` to run php in the ITL (instructions on QMPlus)

Summary

- **Process**: instance of a running program
- **Process context**: data OS has about a process
- **Process state**: OS keeps track of the state of each process: running, ready, blocked, …
- **Protection**: OS has user and kernel modes to prevent processes interfering
  - **Mode switch:** switching from user mode to kernel mode and vice versa
- **Context switch**: switching the running process