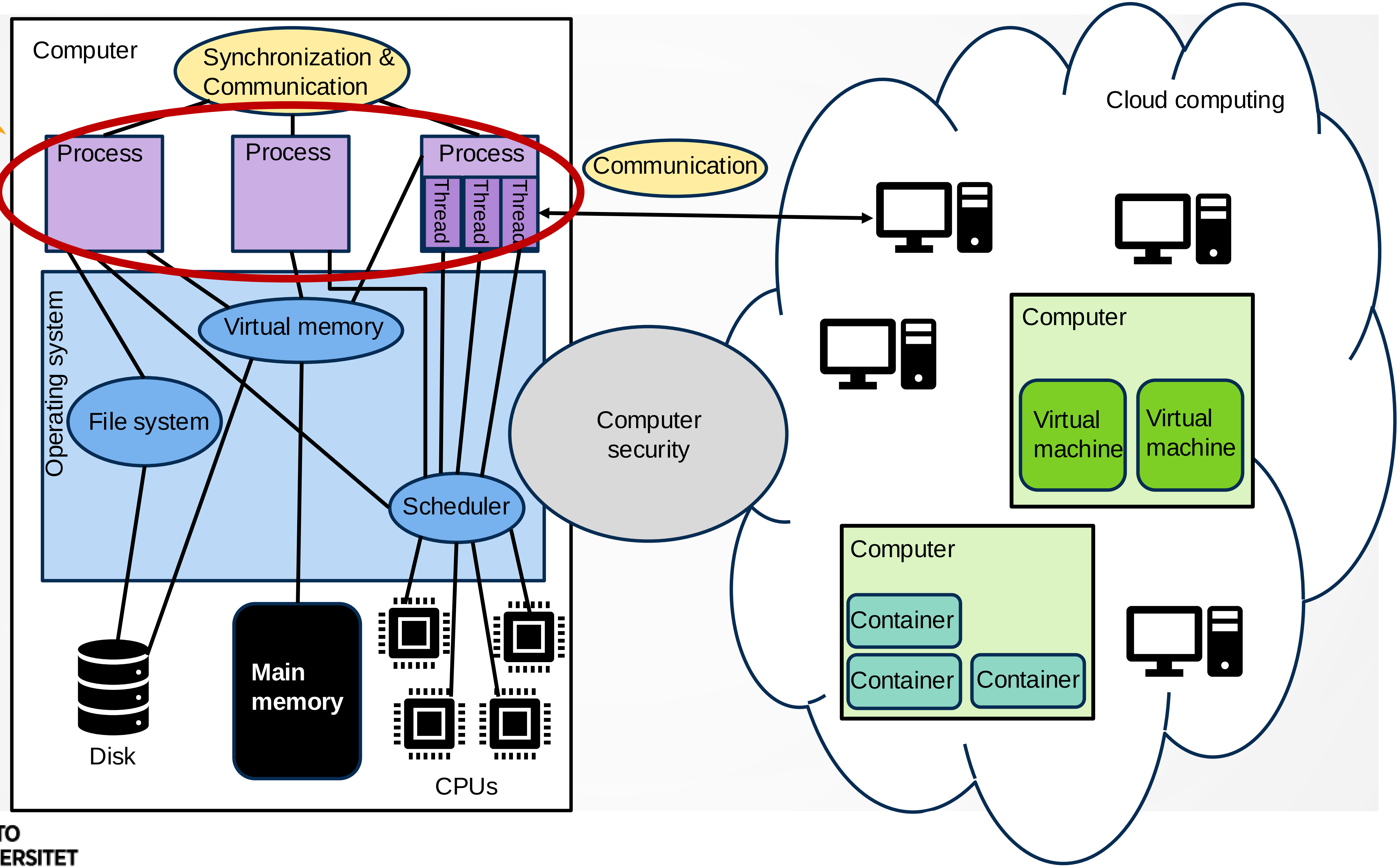
COMPUTING PLATFORMS

Processes

Today's topic



LEARNING OUTCOMES

- After today's lecture, you
  - Understand process as an abstraction of a running program
  - Know how OS keeps track of processes
  - Can define the terms process, process control block and process trace
  - Can explain process state transitions
  - Can explain different modes of execution
  - Can explain how OS switches between processes

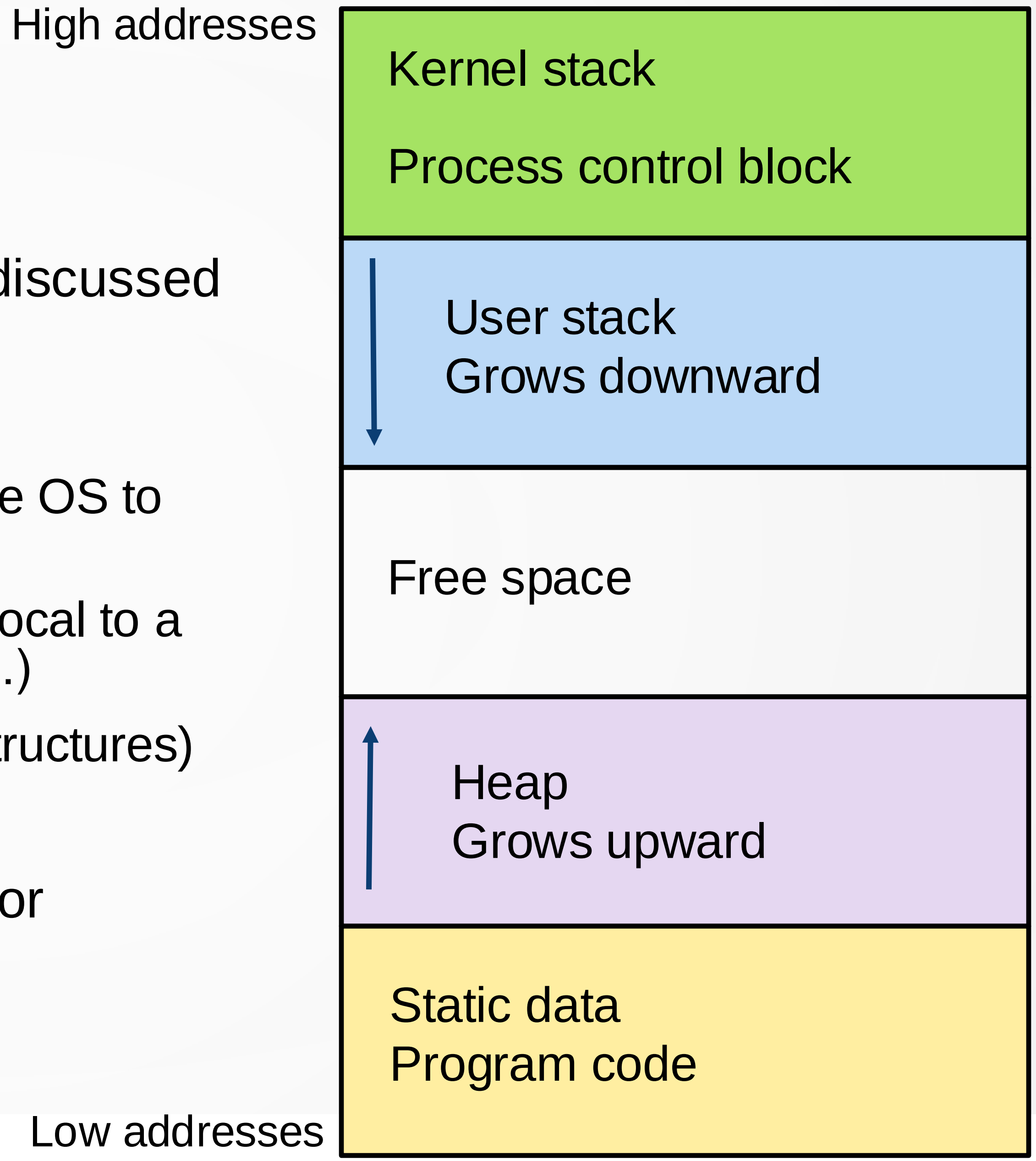
WHAT IS A PROCESS?

- An instance of a program running on a computer
- Also an entity that can be assigned to be run on a processor
- An **abstraction** of a **running program**

- Consists of two parts:
  - **Program code**
  - A **set of data** associated with that code

PROCESS IMAGE

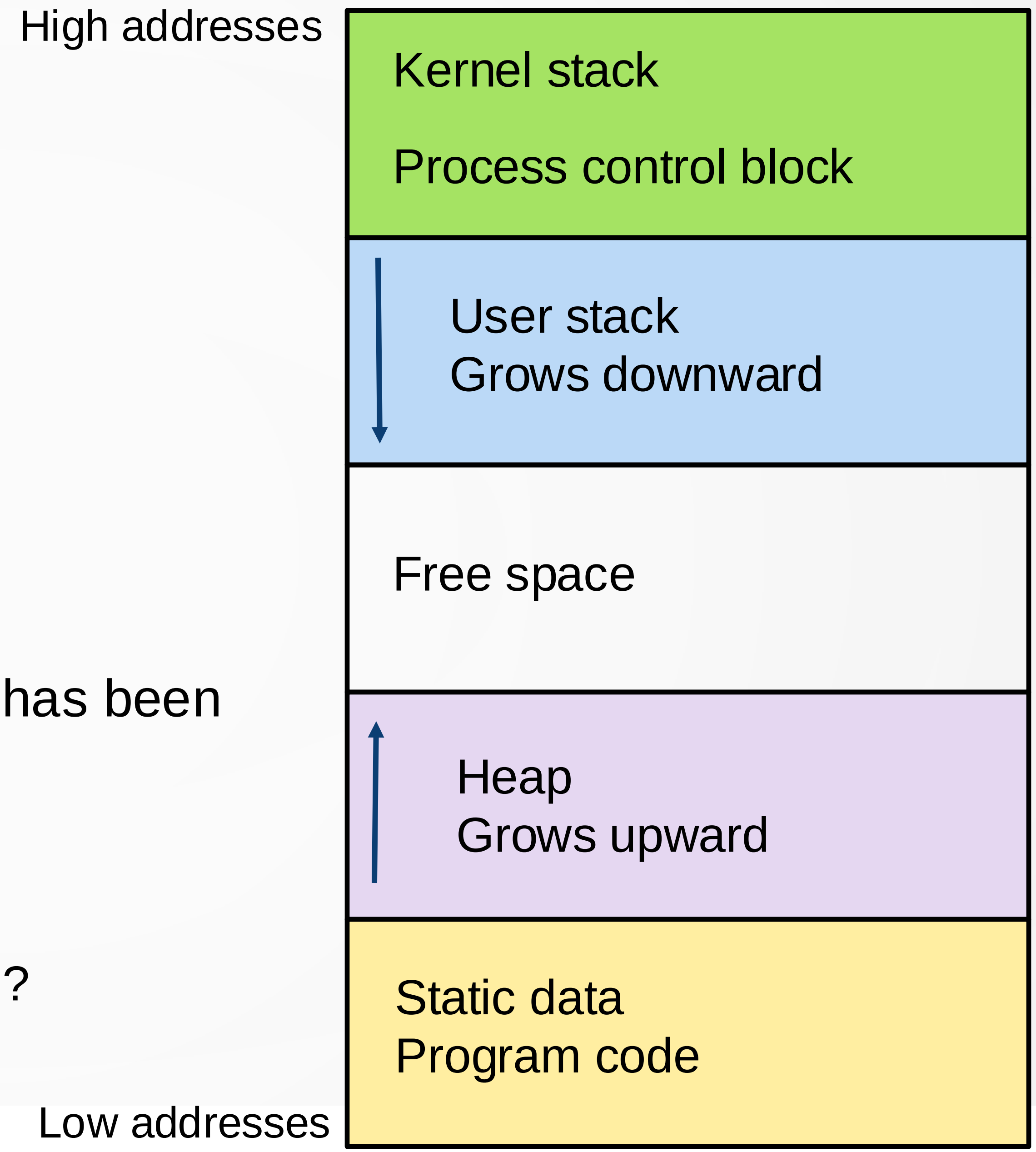
- Processes are images in *virtual memory* (discussed later in the course)
- Typical elements of a process image:
  - **Process control block** (data needed by the OS to keep track of the process)
  - **User stack** (storing parameters, variables local to a function, calling addresses of procedures,…)
  - **Heap** (storing dynamically allocated data structures)
  - Program code and static variables
- User program cannot access kernel stack or process control block (attempt results in segmentation fault)



PROCESS IMAGE

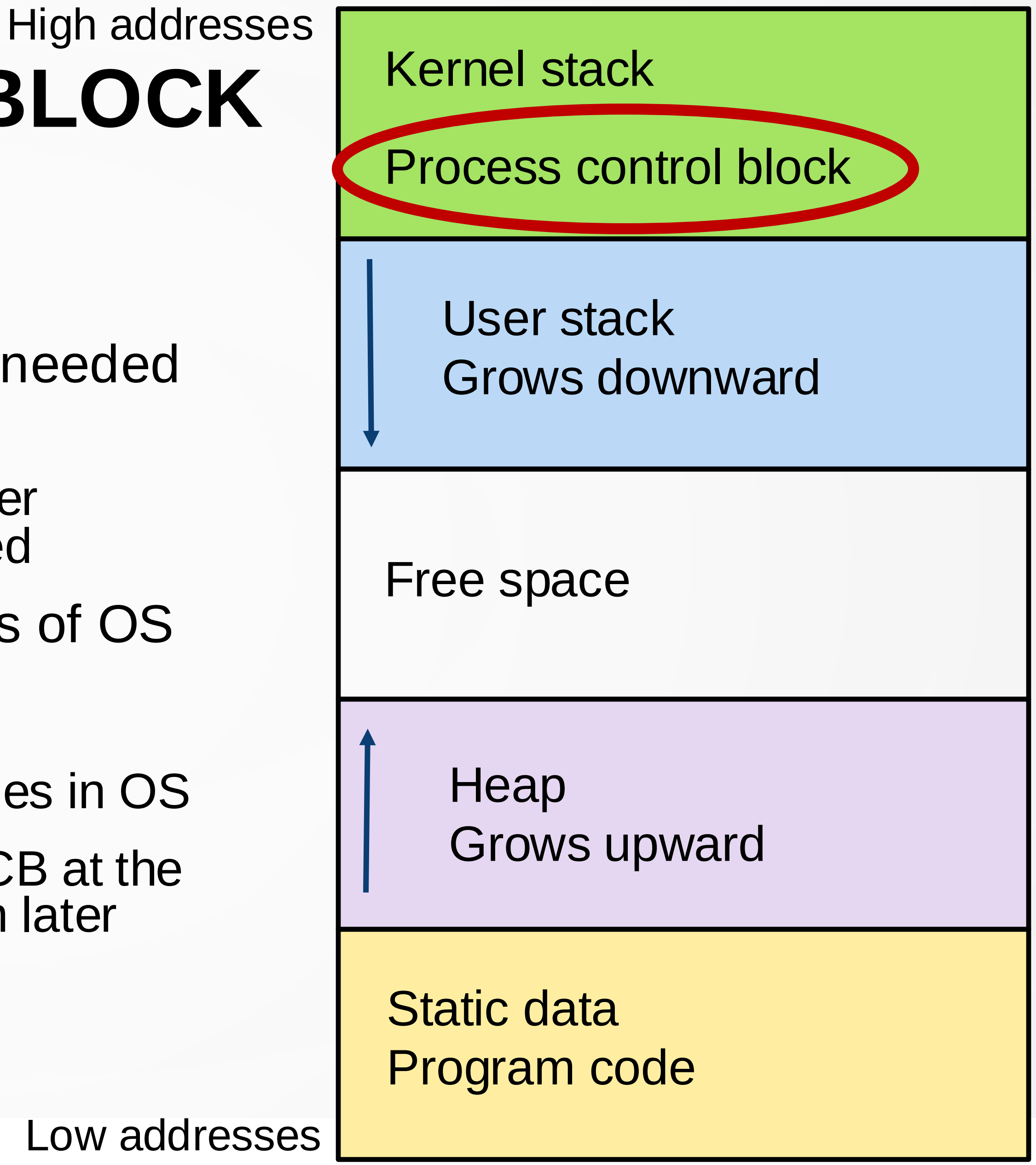
```
def example(n):
     a = set()
     a.add(n)
     return a

print(example(1))
```

- Consider the program above (assuming it has been compiled to machine code):
  - Where is the parameter *n* stored?
  - Where is the variable *a* stored?
  - Where is the set stored in variable *a* stored?
  - Where is the machine code stored?



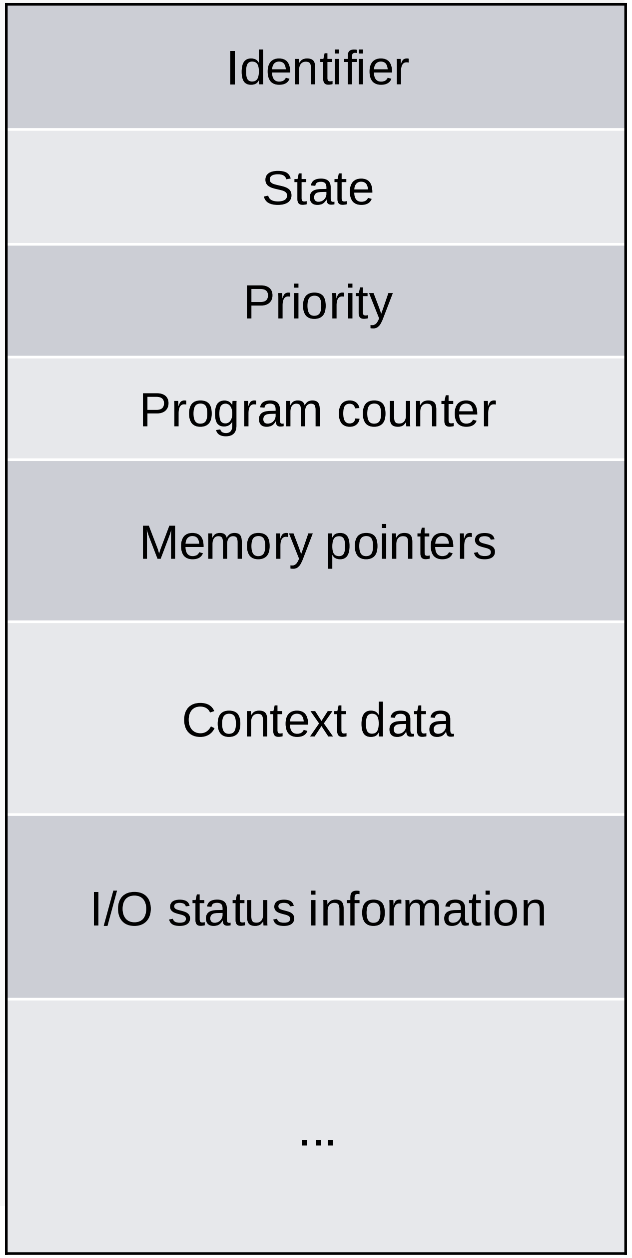
PROCESS CONTROL BLOCK (PCB)

- Most important data structure in an OS
- Contains **all information** about a process needed by OS
  - Running process can be interrupted and later resumed as if the interruption never occurred
- PCBs are read and modified by all modules of OS
- **Protection** of PCBs is very important!
  - Design change of PCB may affect all modules in OS
  - What if different modules want to update PCB at the same time? (Concurrency will be covered in later lectures)



PROCESS CONTROL BLOCK (PCB)

- **Identifier** (unique ID of a process)
- **State** (running, ready to run, blocked,…)
- **Priority** (priority level relative to other processes)
- **Program counter** (address of next instruction to execute)
- **Memory pointers** (pointers to different memory areas in process image)
- **Context data** (data in registers)
- **I/O status information**
- ...



PROCESS EXECUTION: TRACE

- **Process trace**: sequence of instructions that are executed (for a given process)

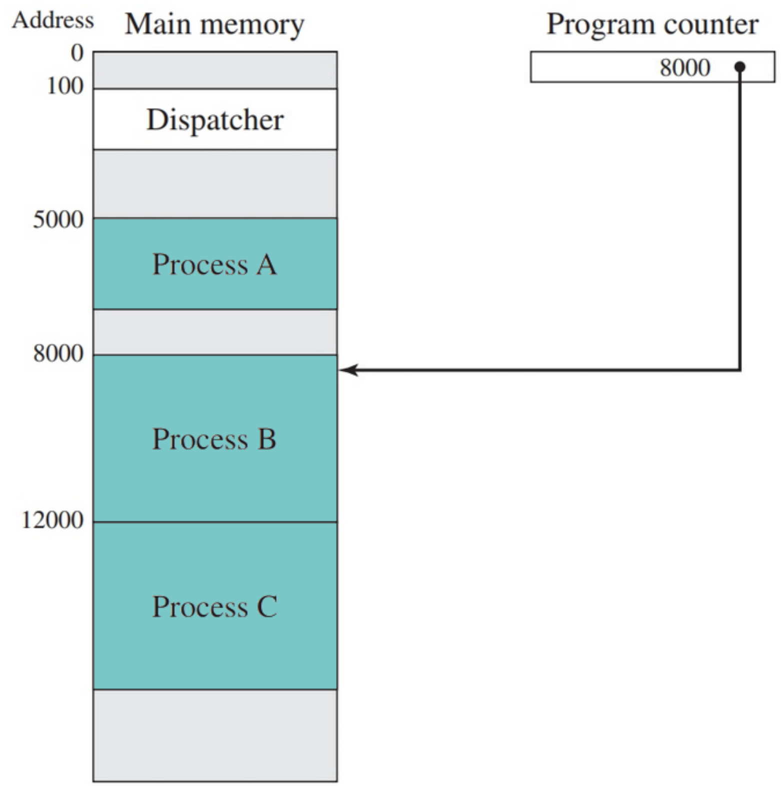


Figure 3.2 **Snapshot of Example Execution (Figure 3.4) at Instruction Cycle 13**

(a) Trace of process A	(b) Trace of process B	(c) Trace of process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = Starting address of program of process A
8000 = Starting address of program of process B
12000 = Starting address of program of process C

Figure 3.3 **Traces of Processes of Figure 3.2**

Figures from [Stallings, Operating systems: Internals and design principles, 9th ed]

PROCESS EXECUTION: CPU'S VIEW

- Behavior of a processor is characterized by interleaving the process traces

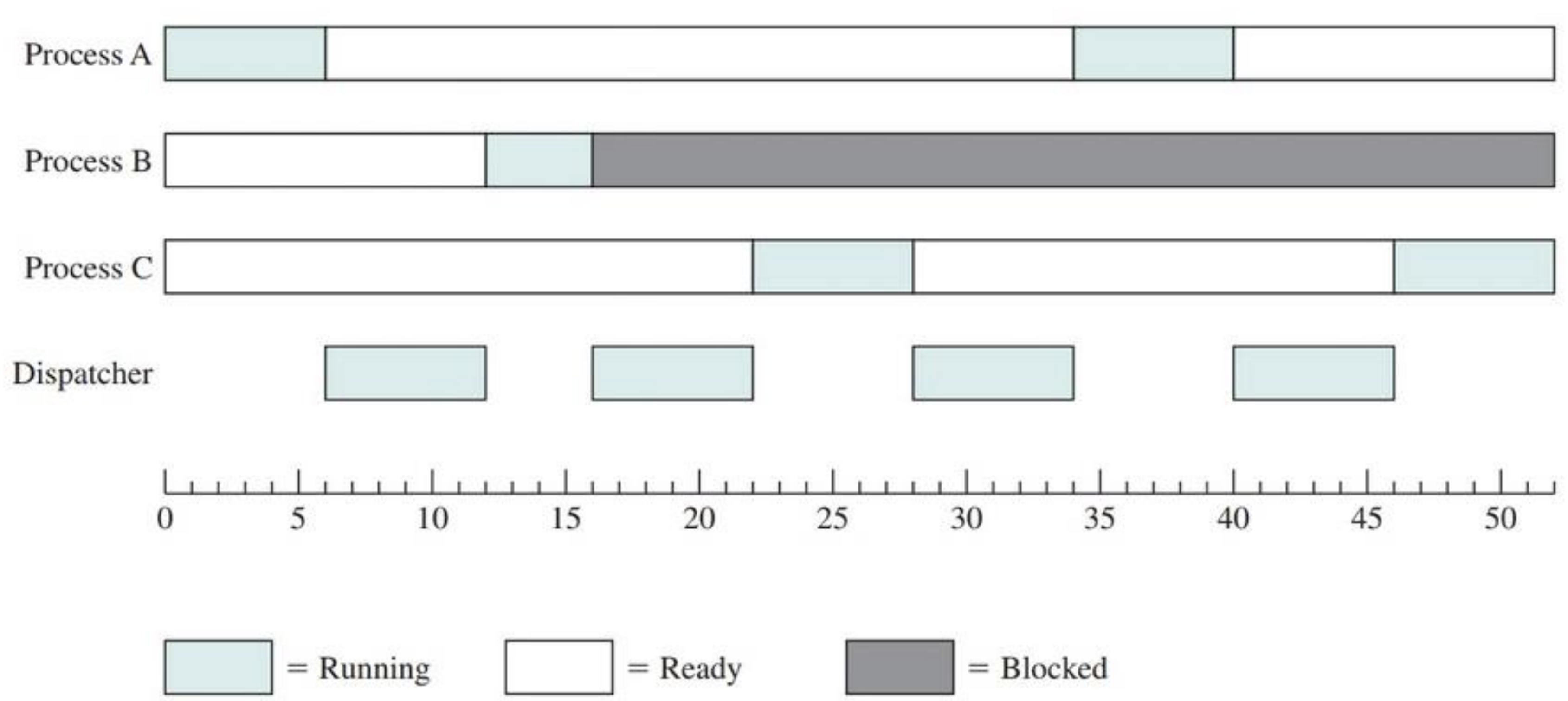


Figure 3.7 Process States for the Trace of Figure 3.4

Figure from [Stallings, Operating systems: Internals and design principles, 9th ed]

PROCESS STATES

- Simplest model: process is **running** or **not running**
- **New process**: OS creates PCB and process enters the system in **not running** state
- From time to time, the currently running process is **interrupted**, and the **dispatcher** of the OS **selects** another process to run

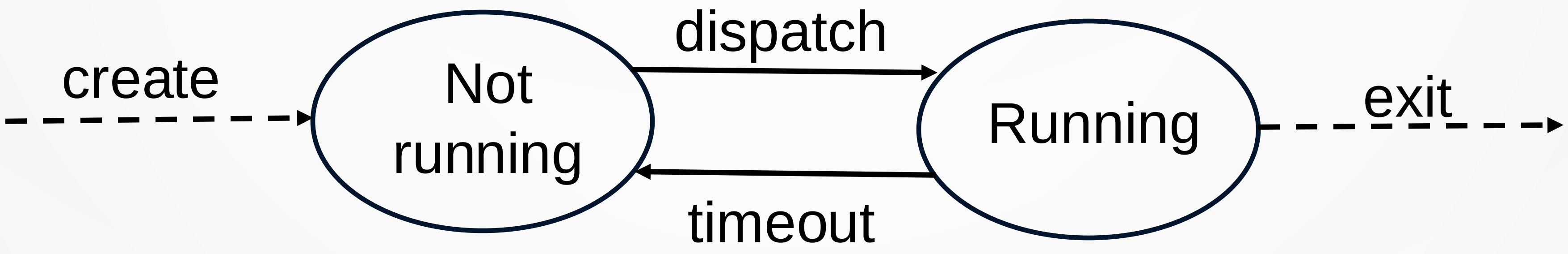


Figure adapted from Fig 3.5a, [Stallings, Operating Systems: Internals and design principles, 9th ed]

PROCESS STATES: FIVE STATE MODEL

- If all processes are always **ready to execute**, then two state model works
  - Processes are in a queue and processor handles them in round-robin manner
- Some processes are **blocked** because e.g.
  - Process needs a **resource** (e.g. file) which is **not available**
  - Process initiated an **action** (e.g. I/O operation) that **must complete** before the process can continue
  - Process is **waiting for another process** to provide data or a message from another process
- Solution: split **not running** state to **ready** and **blocked** states
  (We also add **new** and **exit** states.)

PROCESS STATES: FIVE STATE MODEL

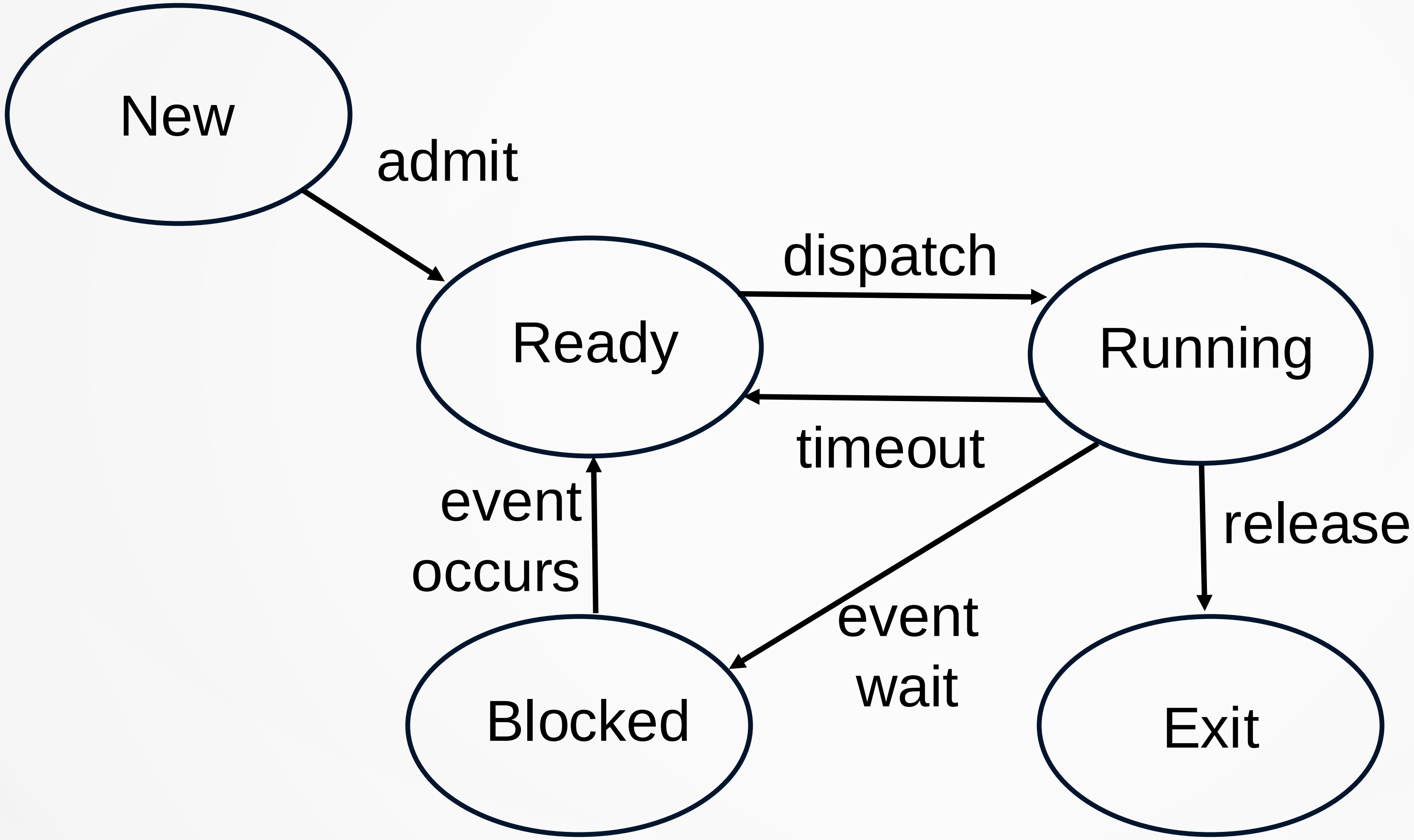
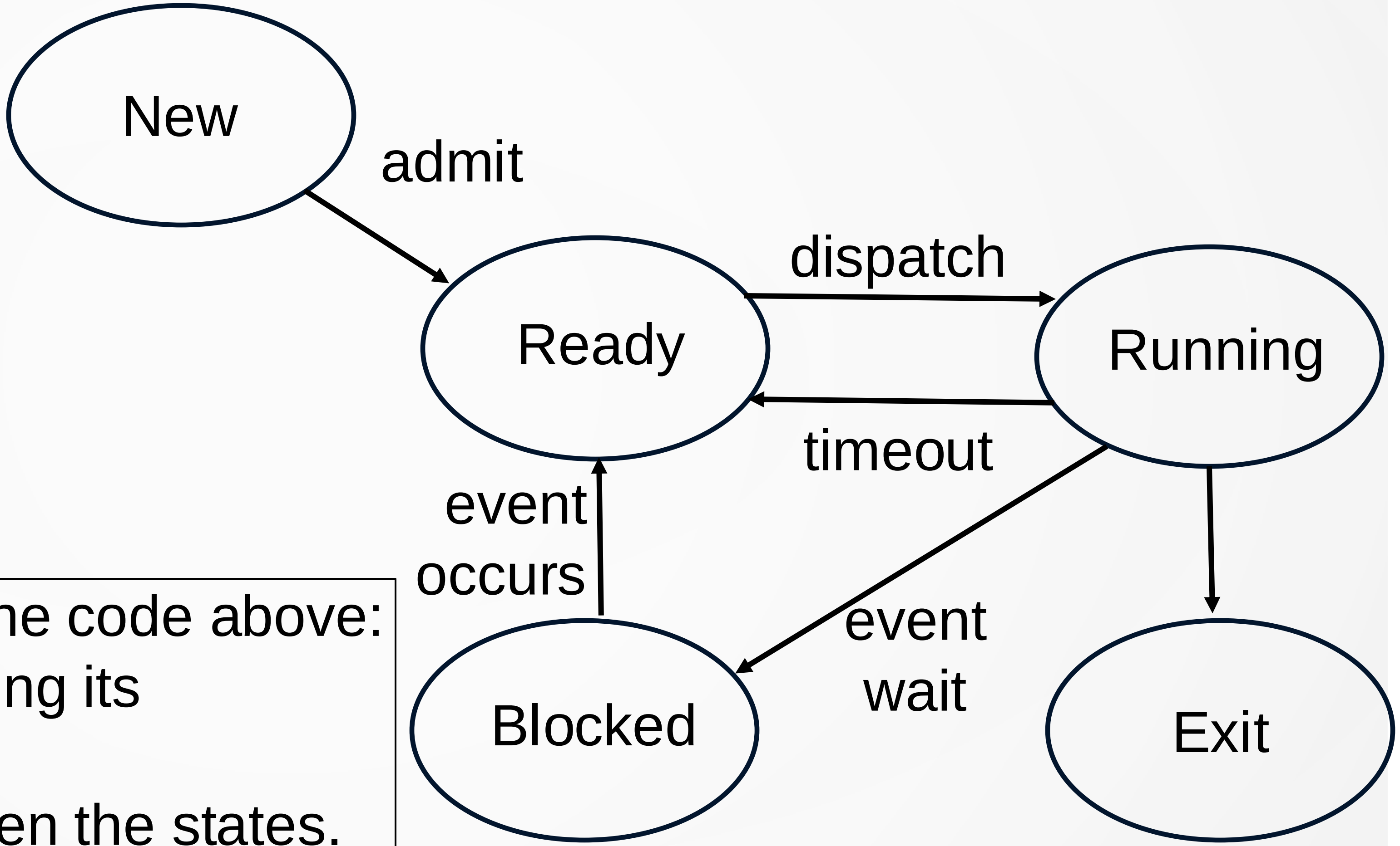


Figure adapted from Fig 3.6, [Stallings, Operating Systems: Internals and design principles, 9th ed]

PROCESS STATES: EXAMPLE

```
def example(n):
     a = set()
     a.add(n)
     return a

print(example(1))
```



Consider a process executing the code above:

- Which states does it visit during its execution?
- Explain the transitions between the states.

Figure adapted from Fig 3.6, [Stallings, Operating Systems: Internals and design principles, 9th ed]

PROCESS STATES: SUSPENDING

- What if memory is full and all processes in memory are waiting for I/O?
- A process (or part of it) can be **swapped** (i.e. moved) from main memory to disk
  - When memory is full and no process in main memory is in **ready** state, OS **swaps one blocked process out** to disk into **suspend** queue
  - Suspend queue consists of existing processes that have been **temporarily** kicked out of main memory
  - OS can then bring in a process from **suspend** queue or **new** queue
  - Execution continues with the newly arrived process
- Also other reasons for suspending than swapping: user request, timing, parent process request...

PROCESS STATES: SUSPENDING

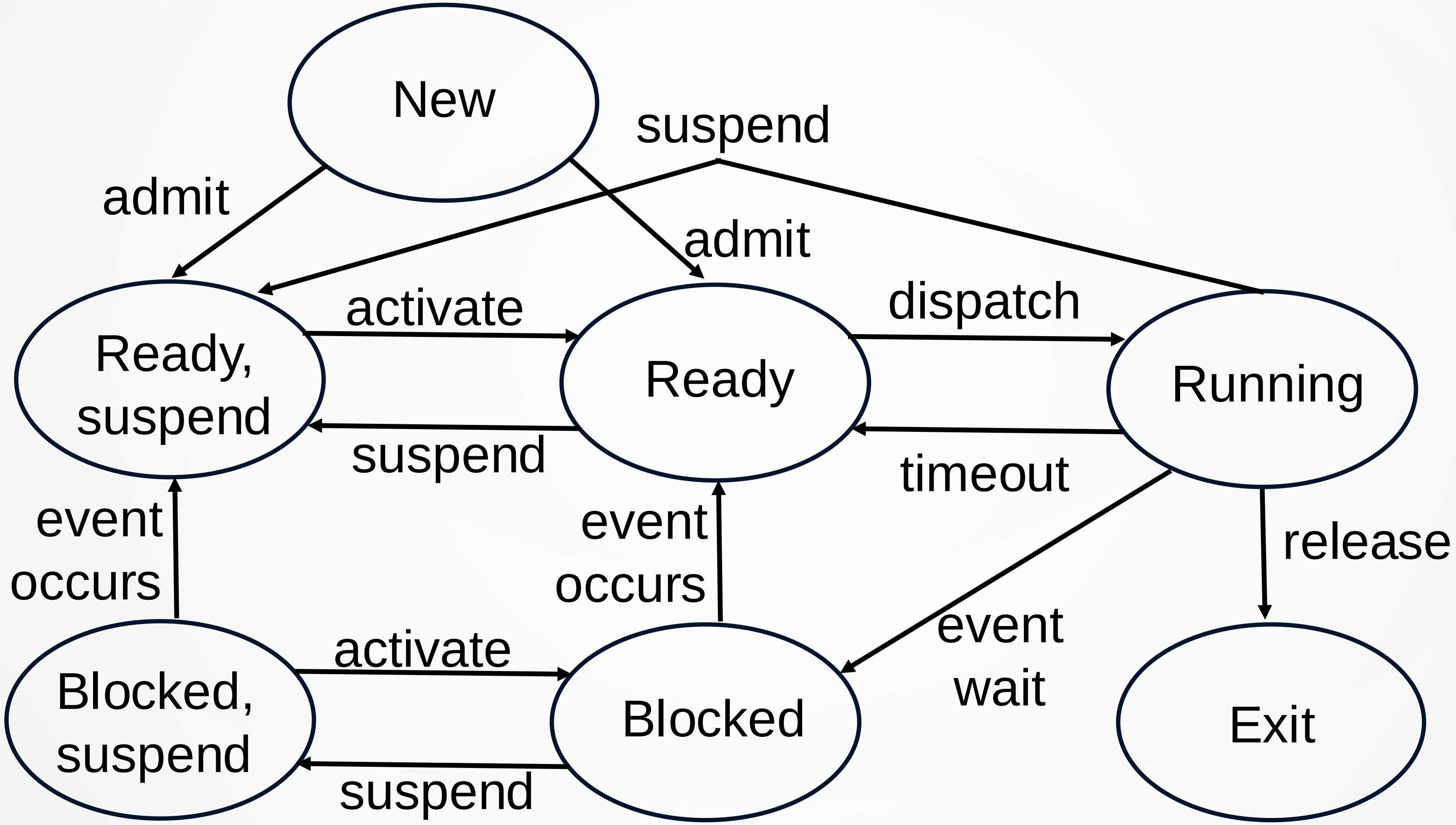


Figure adapted from Fig 3.9b, [Stallings, Operating Systems: Internals and design principles, 9th ed]

HOW DOES OS KEEP TRACK OF PROCESSES?

- OS manages system resources for processes
- **Memory tables** keep track of main memory and secondary memory (covered in detail when we discuss virtual memory)
- **I/O tables** keep track of the status of I/O devices and channels
- **File tables** keep track of files (existence, open files, … covered later when we discuss file management)
- **Process tables** keep track of processes by linking to PCBs

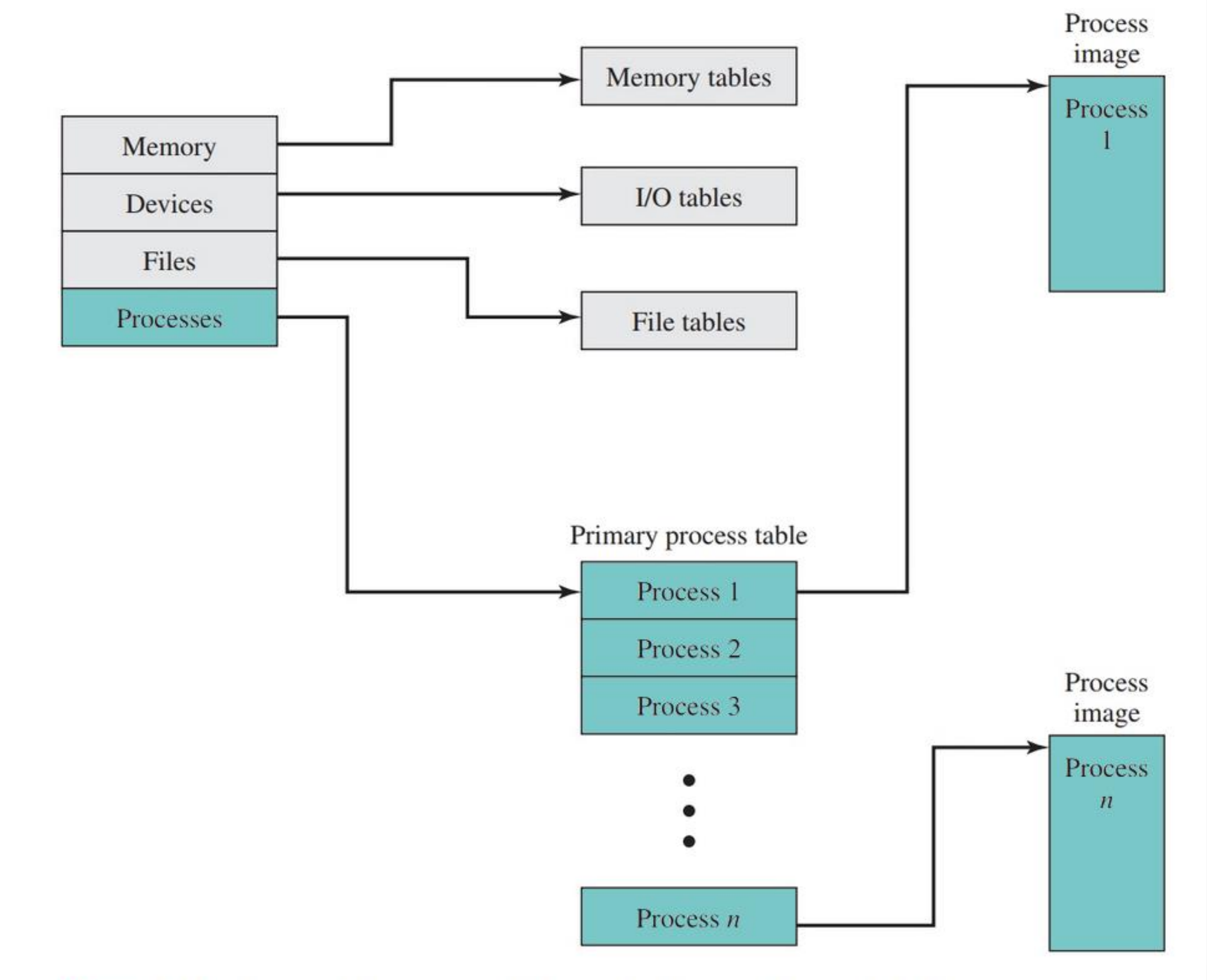


Figure 3.11 General Structure of Operating System Control Tables

Figure from [Stallings, Operating systems: Internals and design principles, 9th ed]

HOW DOES OS KEEP TRACK OF PROCESSES?

- (Single-core) processor can **execute one process at a time**
- Processor is shared among multiple processes so **all appear to be progressing**
- OS needs to schedule **process switches**
- OS maintains queues for processes in different states to allow for efficient scheduling (covered in a later lecture)

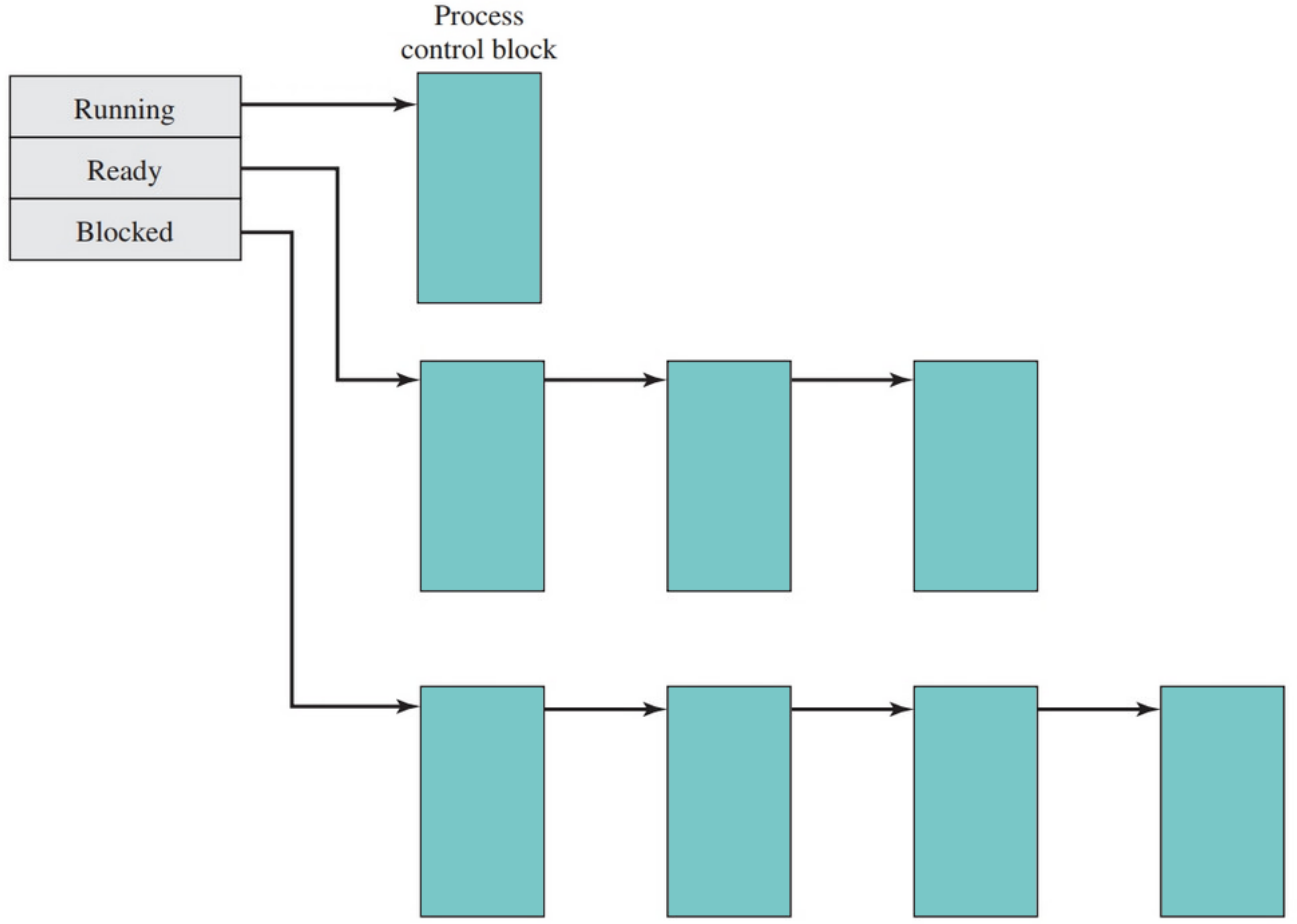


Figure 3.14 Process List Structures

Figure from [Stallings, Operating systems: Internals and design principles, 9th ed]

MODES OF EXECUTION

- OS is executed on a processor just like other processes
- OS **must be able to read and update OS data structures**
- User process **must not be able to mess up the system** (e.g. by accessing or updating OS data structures)
- **Limited mode of execution** needed
- (At least) two modes of execution
  - **User mode**: less privileged, user processes typically execute in this mode
  - **System mode** (control mode / kernel mode): more privileged, kernel of OS executes in this mode)

HOW IS THE MODE CHANGED?

- OS (executing in system mode) **can switch to a less privileged mode** (user mode) and continue execution at an arbitrary point (e.g. user process code)
- A user process (executing in user mode) **cannot be allowed to switch to more privileged mode** (system mode) arbitrarily
- User mode -> system mode switch uses **interrupts**, **traps** and **supervisor calls**
  - Processor switches to system mode and continues execution at an interrupt handler (setup by OS, i.e. interrupt handler is OS code)

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

STEPS OF MODE SWITCH

- If an interrupt is pending, **processor** performs a **mode switch**:
  1. **Processor** sets the program counter to the starting address of an **interrupt handler**
  2. **Processor switches from user mode to system mode**
  3. **OS saves the context** of the process that has been interrupted into PCB of the interrupted process
- What is saved? - any information that interrupt handler might change and is needed to resume the process that was interrupted
  - Processor state information (including the program counter, other processor registers and stack information)
- After returning from interrupt handler, switch to user mode; if different process will run, need to perform **process switch**

PROCESS SWITCH: HOW DOES OS GAIN CONTROL?

- **Co-operative approach**: wait for supervisor calls
  - Supervisor calls privileged, hence OS will gain control
  - If process makes no supervisor calls or errors, OS never gains control
- **Non-cooperative approach**: OS takes control
  - **Hardware support**: use a timer
  - Timer interrupt gives control to OS at certain intervals

Note on terminology: Stallings uses the term process switch, also the term **context switch** is used;
OSTEP uses the term **trap handlers,** while Stallings more often uses the term **interrupt handler;**
Also the term **system call** is used for **supervisor call**

STEPS OF PROCESS SWITCH

1. Perform **mode switch** to system mode (**saves context** of the processor to PCB of the currently running process)
2. **Update PCB** of the process currently in **running** state
   - Change the state to **ready**, **blocked**, **ready/suspend**, or **exit**
3. Move PCB of this process to appropriate queue (**ready**, **blocked** on event *i*, **ready/suspend**)
4. Select another process for execution (discussed later in the course)
5. Update PCB of the selected process
   - Change the state to **running**
6. Update memory management data structures (discussed later in the course)
7. **Restore the context** of the processor to that which existed when the selected process was last switched out (this context is saved in the PCB)
8. Change mode to **user mode**

IS OS A PROCESS TOO?

- OS is software, i.e. set of programs executed by the processor
- OS frequently releases control and relies on processor to restore control to OS
- Alternative ways to implement an OS
  a) Separate kernel: OS has own region of memory, one kernel stack, supervisor calls handled via process switch
  b) Many kernel stacks: inside process images, supervisor calls handled by mode switch
  c) Daemon processes: OS implemented as a set of processes, convenient when multiple processors available

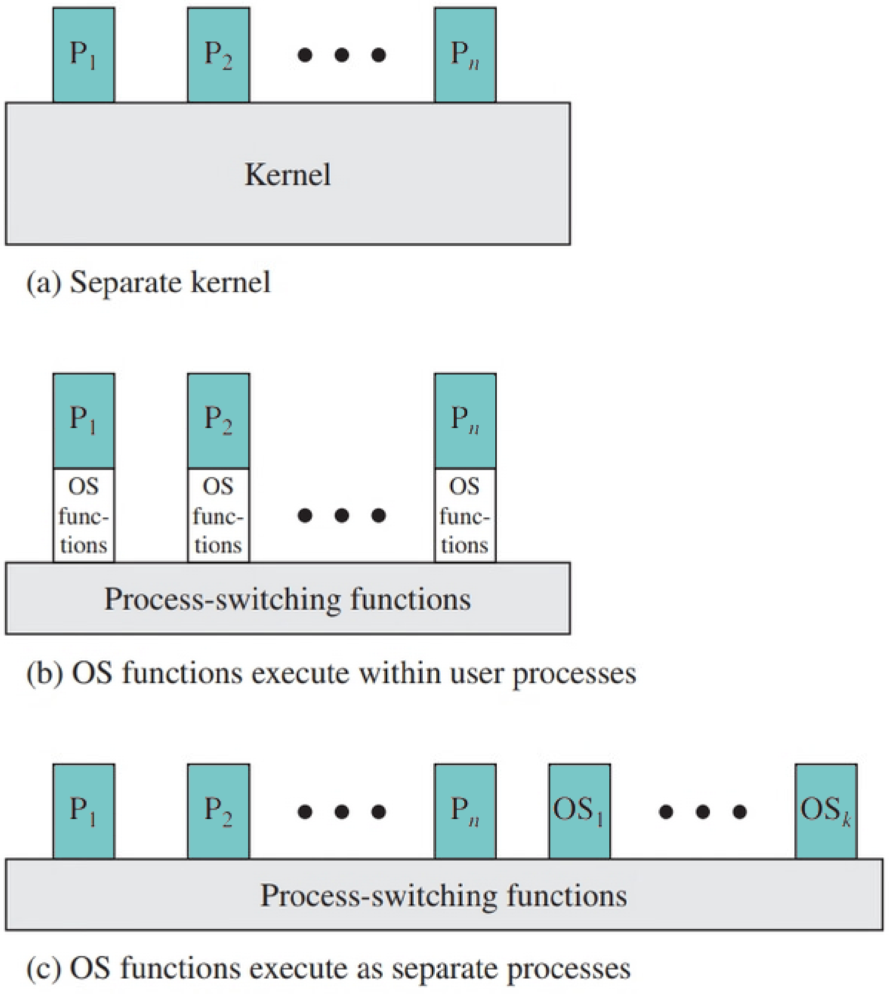


Figure 3.15 **Relationship between Operating System and User Processes**

Figure from [Stallings, Operating systems: Internals and design principles, 9th ed]

SUMMARY

- **Process** is an **abstraction of a running program**
- Key concepts:
  - Process image, process control block
  - Process trace
  - Process states and state transitions
  - How does OS maintain information of processes
  - User mode, system/kernel mode
  - Mode switch, process switch (context switch)