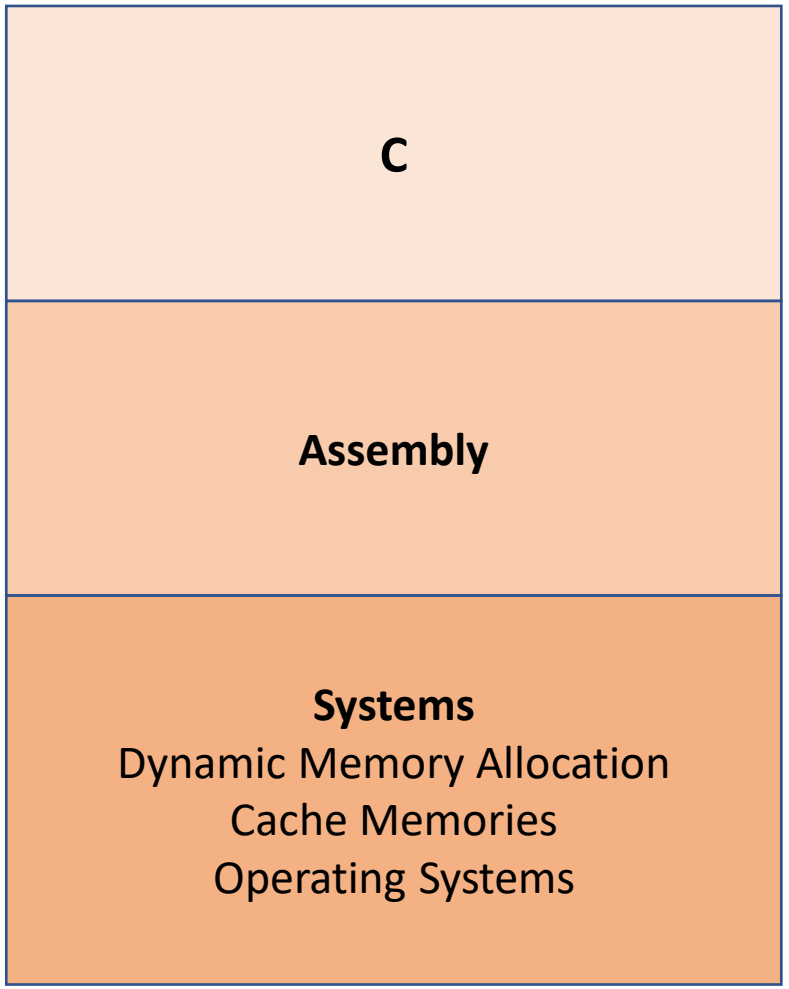
CS 354
Machine Organization and Programming
Lecture 25

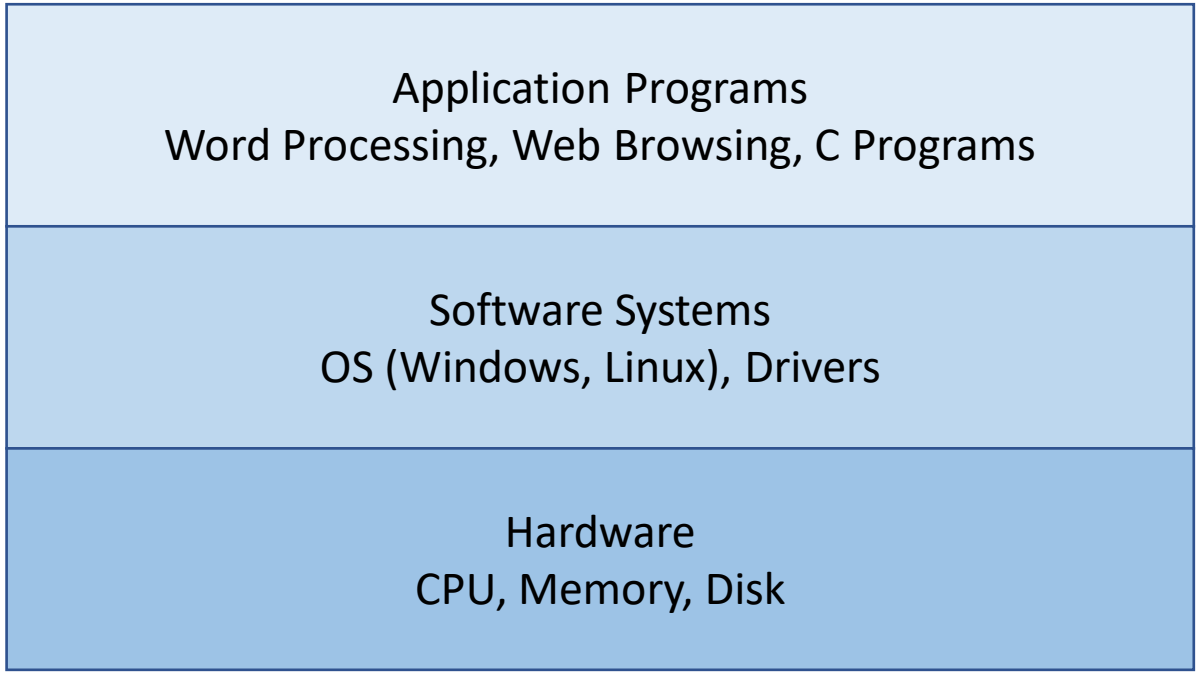
Michael Doescher
Summer 2020

Intro to Operating Systems

Course Overview



How Computers Work



What is an Operating System

- A program that makes your computer much easier to use
- A program that manages computer resources and provides common services needed by applications
- Input / Output, Memory Management, Time Sharing
- Takes care of the details to allow your computer to run two programs at the ‘same time’

How does a computer run two programs at the same time?

- If we only have 1 CPU?
- Memory Address Space is a contiguous block from addr: 0 to addr: MAX

What does the CPU do?

sum.c

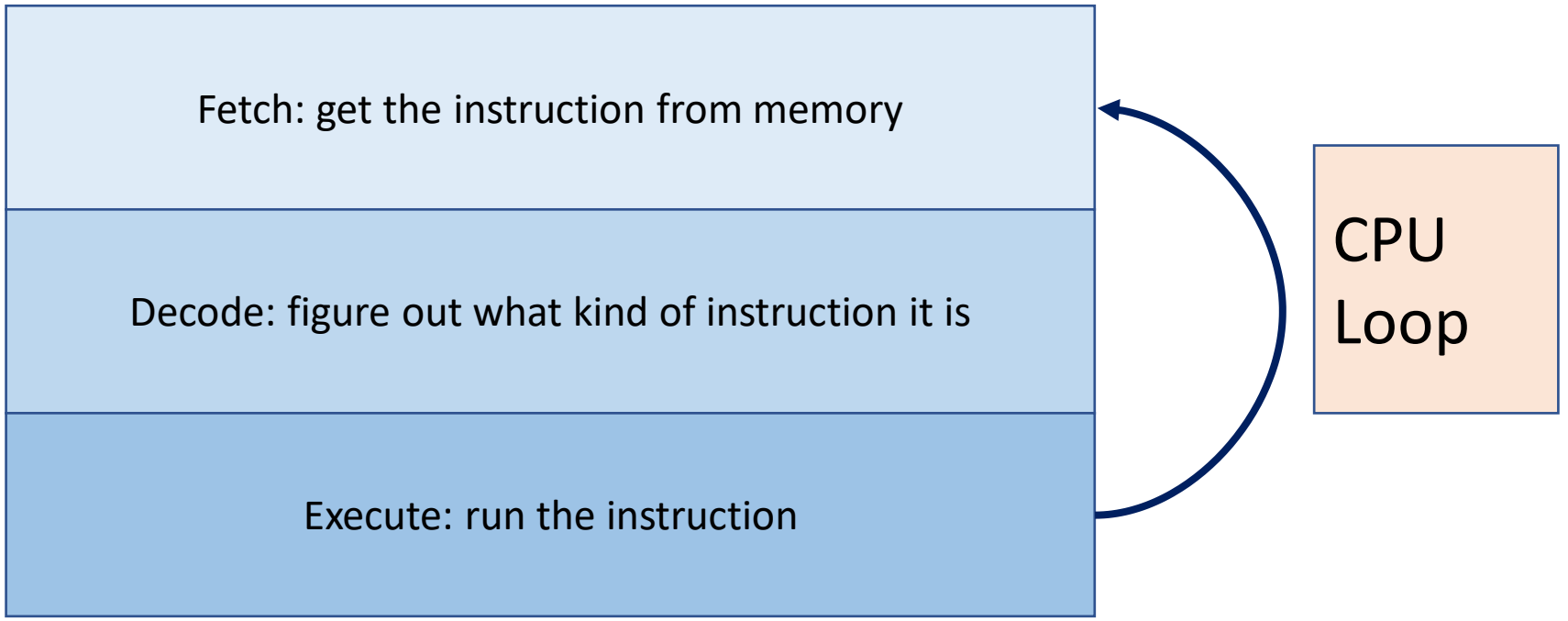
……

a = 1;

b = 2;

c = a + b;

……

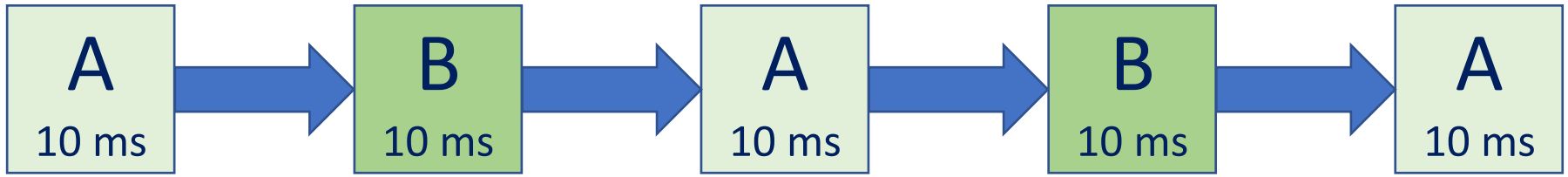


Two Programs - One CPU

Time Sharing

Switch back and forth between processes

OS figures out how to schedule when each program runs on the CPU



Shared Address Space

Memory is a linear array of memory address from 0 to MAX

Programs have access to the entire address space.

What if Program A writes to addr: 0xFA0
Then Program B writes to addr: 0xFA0

Virtual address space

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret
0x201 | instr 44

Heap:
0x410 | 0x18

stack:
0xFA0 | 0x03
0xFA4 | 0x10C
0xFA8 | 0x29
```

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret
0x201 | instr 44

Heap:
0x410 | 0x18

stack:
0xFA0 | 0x03
0xFA4 | 0x10C
0xFA8 | 0x29
```

Shared Address Space

Virtual address space

Addresses used by programs are not the addresses used in physical memory

OS lets us treat the memory that a program sees as the physical memory

OS keeps track of mapping where memory used by each program is really stored in physical memory

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret
0x201 | instr 44

Heap:
0x410 | 0x18

stack:
0xFA0 | 0x03
0xFA4 | 0x10C
0xFA8 | 0x29
```

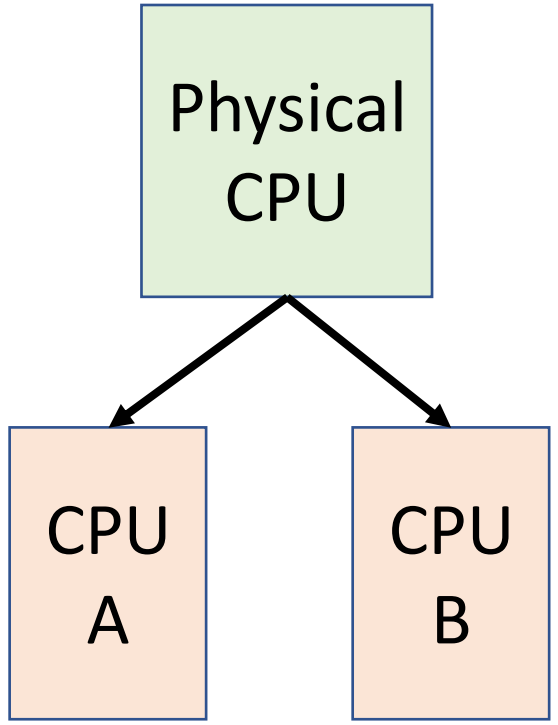
```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret
0x201 | instr 44

Heap:
0x410 | 0x18

stack:
0xFA0 | 0x03
0xFA4 | 0x10C
0xFA8 | 0x29
```

CPU Virtualization

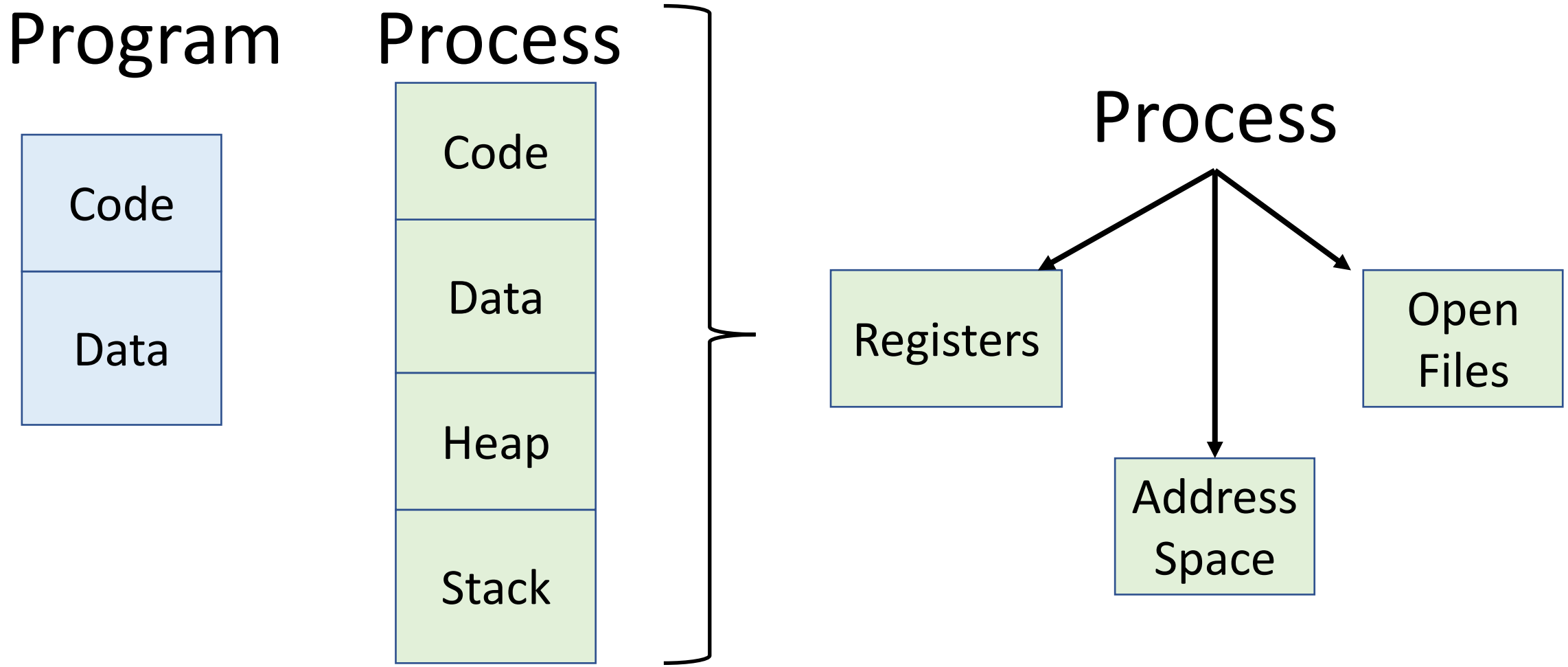
Program A runs on CPU A
Program B runs on CPU B

Each program sees the illusion that they have access to the entire CPU

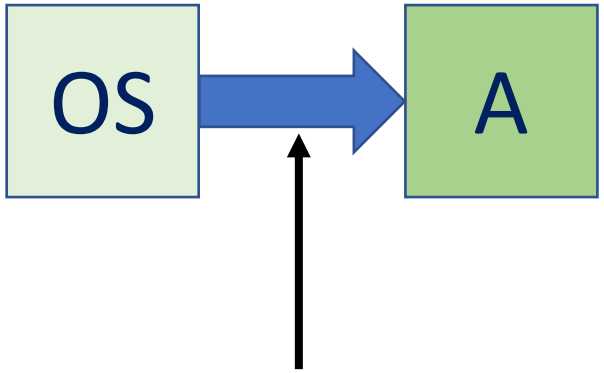
Time sharing creates this illusion.



Processes


Program
Code
Data
Process
Code
Data
Heap
Stack
Process
Registers
Open Files
Address Space

How do you switch processes?

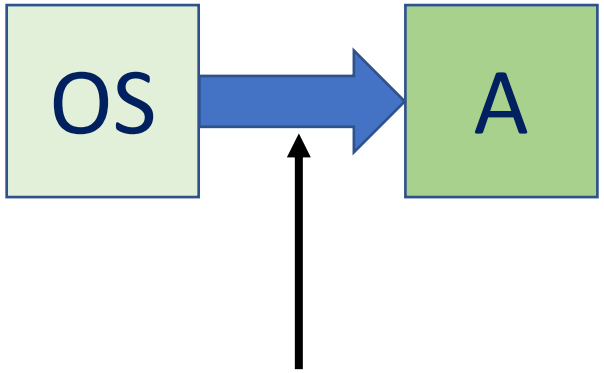


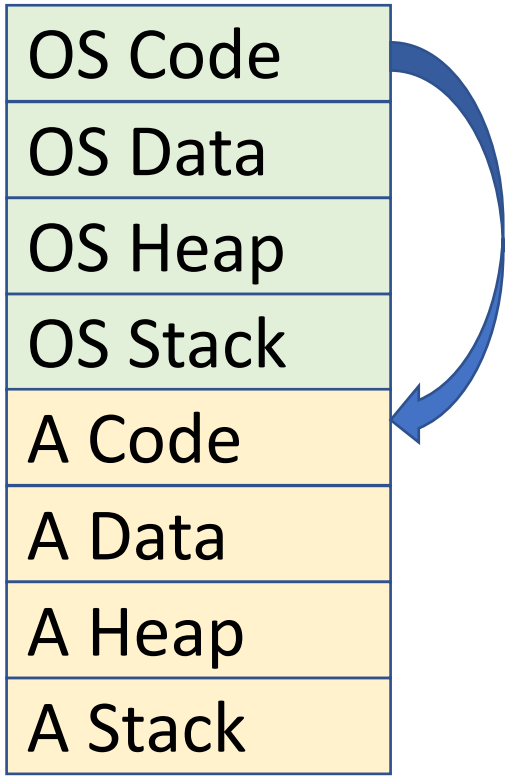
OS chooses
Process A to run

Resources Used - State

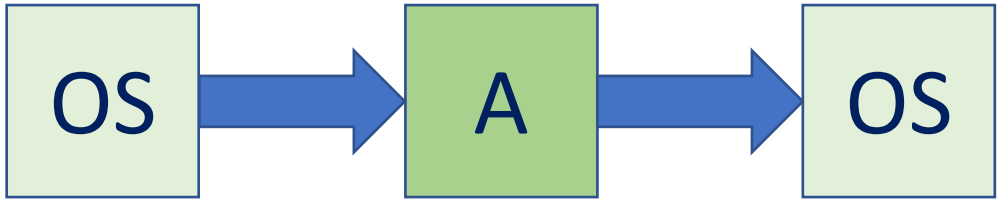
- Registers
- Memory code, data, stack, heap (evict?)
- Address Space
- Open files

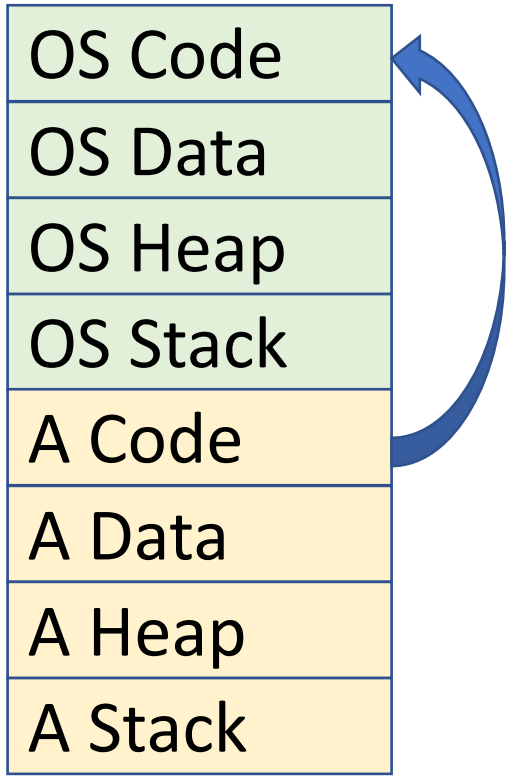
How do you switch processes?





How do you switch back?



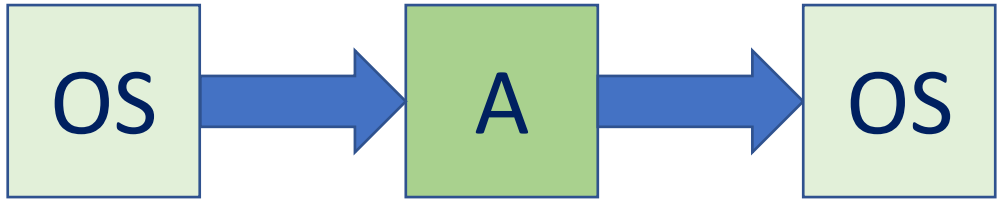


yield();

Requires all process to play nice

```
while(1) {
    ....
}
```

OS needs to regain control



Hardware support

Timer interrupt –
duration set by OS

Hardware Interrupt Handler

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret
0x201 | instr 44
```

Interrupt

%eip
0xFFF0

```
Interrupt Handler:
0xFFF0 ........
0xFFF4 ........
0xFFF8 ........
0xFFFC ........
```

CPU Virtualization Mechanisms

1. Timer Interrupt
2.

At Boot time (power on the computer)
OS tells the hardware the location of the interrupt handlers

Types of interrupts
- timer
- keyboard
- disk

OS sets up the timer interrupt - a physical clock that just produces signals at a regular interval

Problems

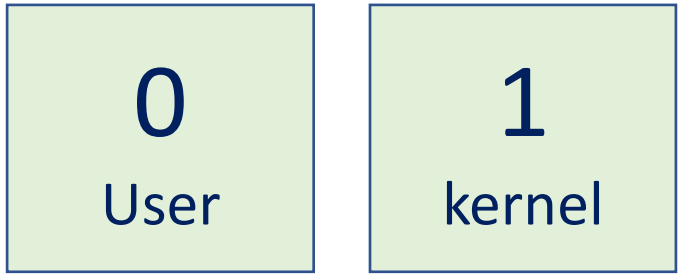
1. Timer Interrupt
2.

- Timer can be switched on and off
- OS can change the address of the timer interrupt code

OS doesn’t want to be interrupted!
What if a user process also switches off the timer?

Solution

1. Timer Interrupt
2. Privilege Mode



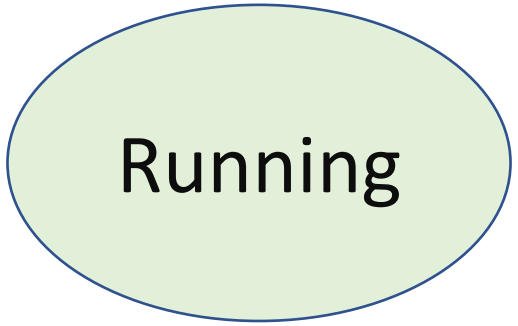
Status Register

set to 0 then user mode
set to 1 then kernel mode (the core of the OS)

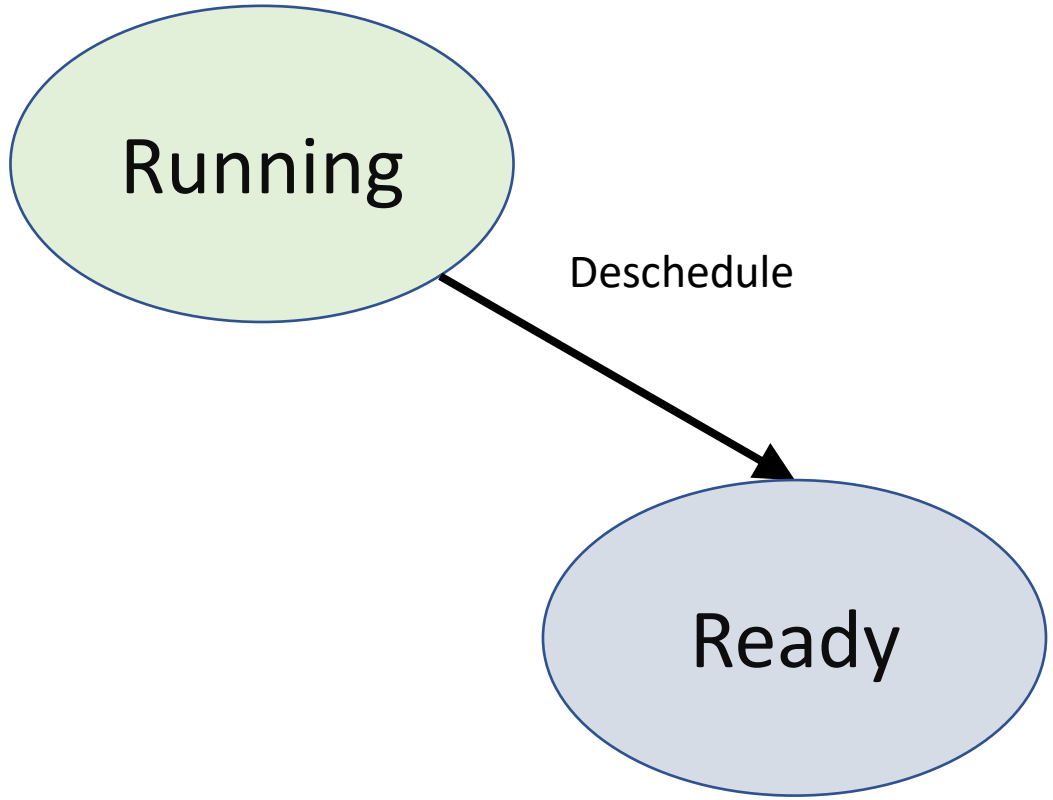
- core = scheduler, memory virtualization, ...
- not core – device drivers, input/output, ...

When kernel mode is set – any instruction can be executed
Example: Turning off the timer interrupt is a privileged instruction

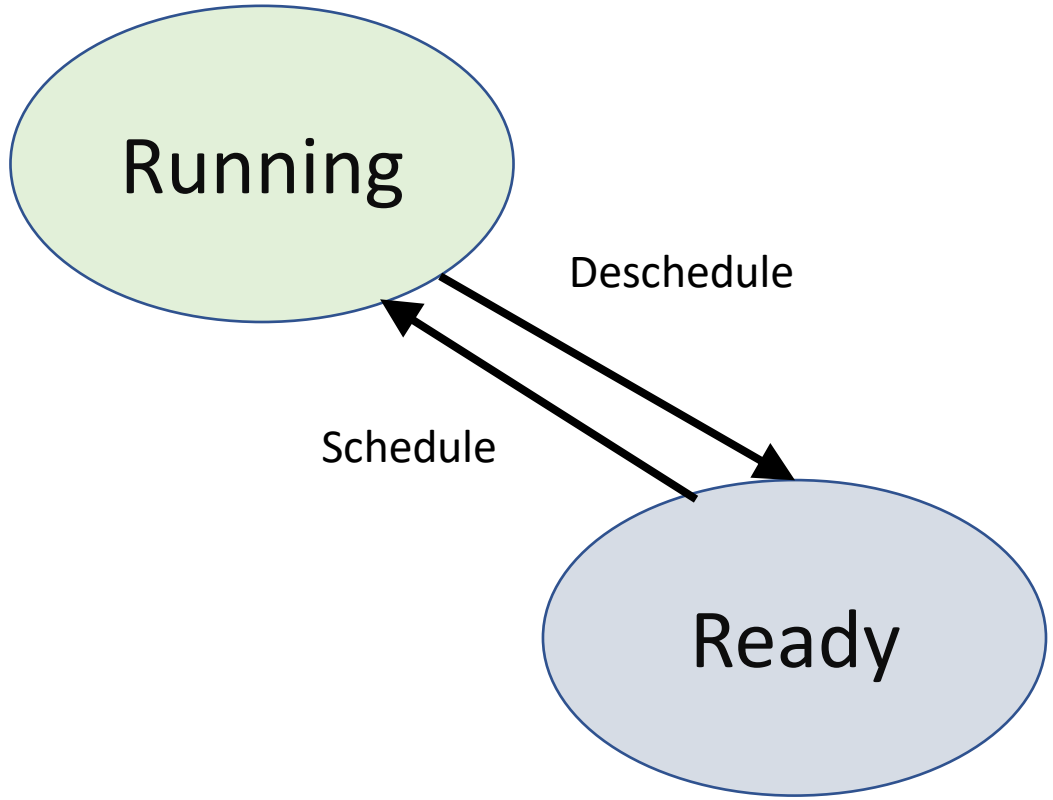
Process State



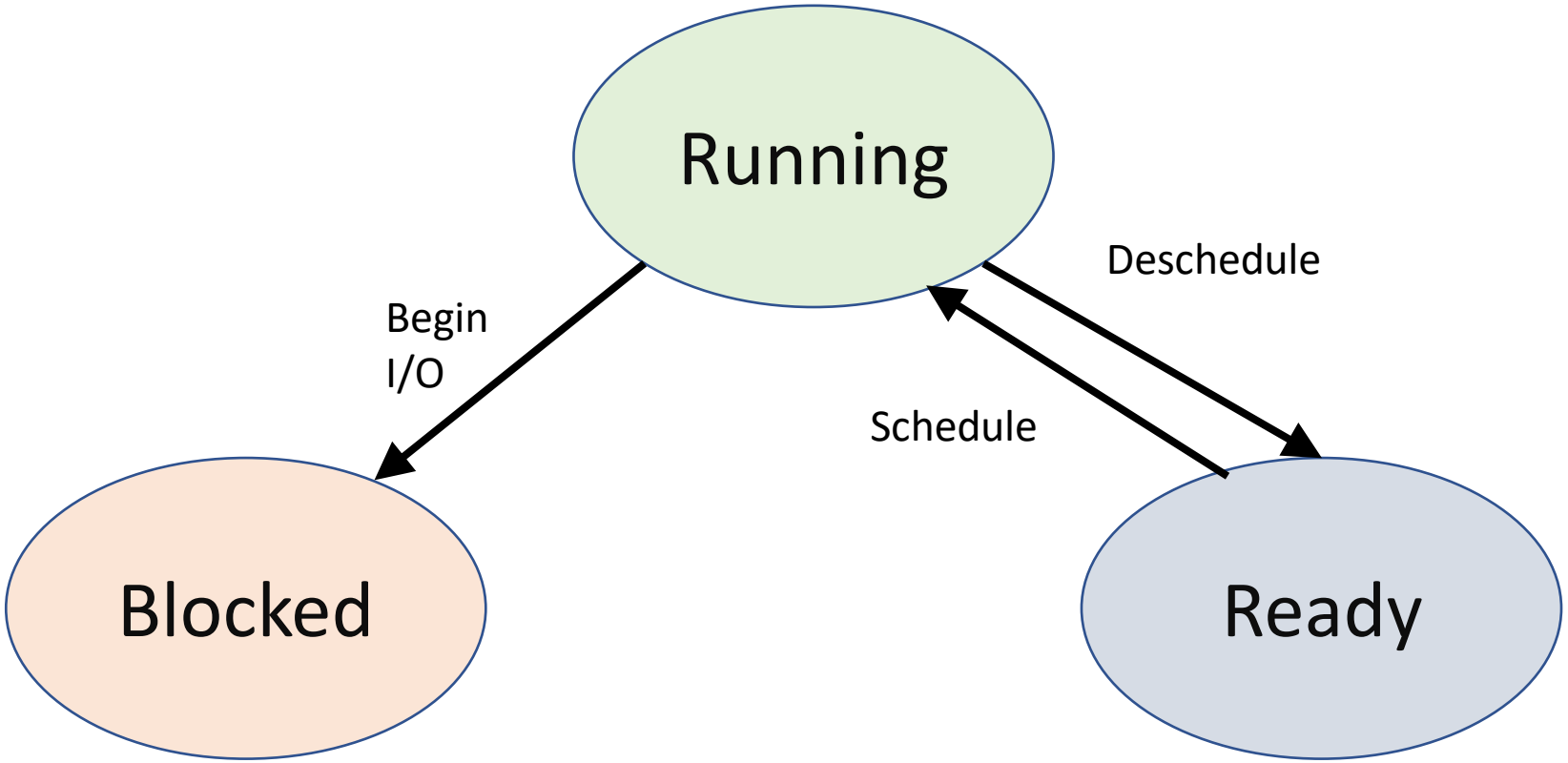
Process State



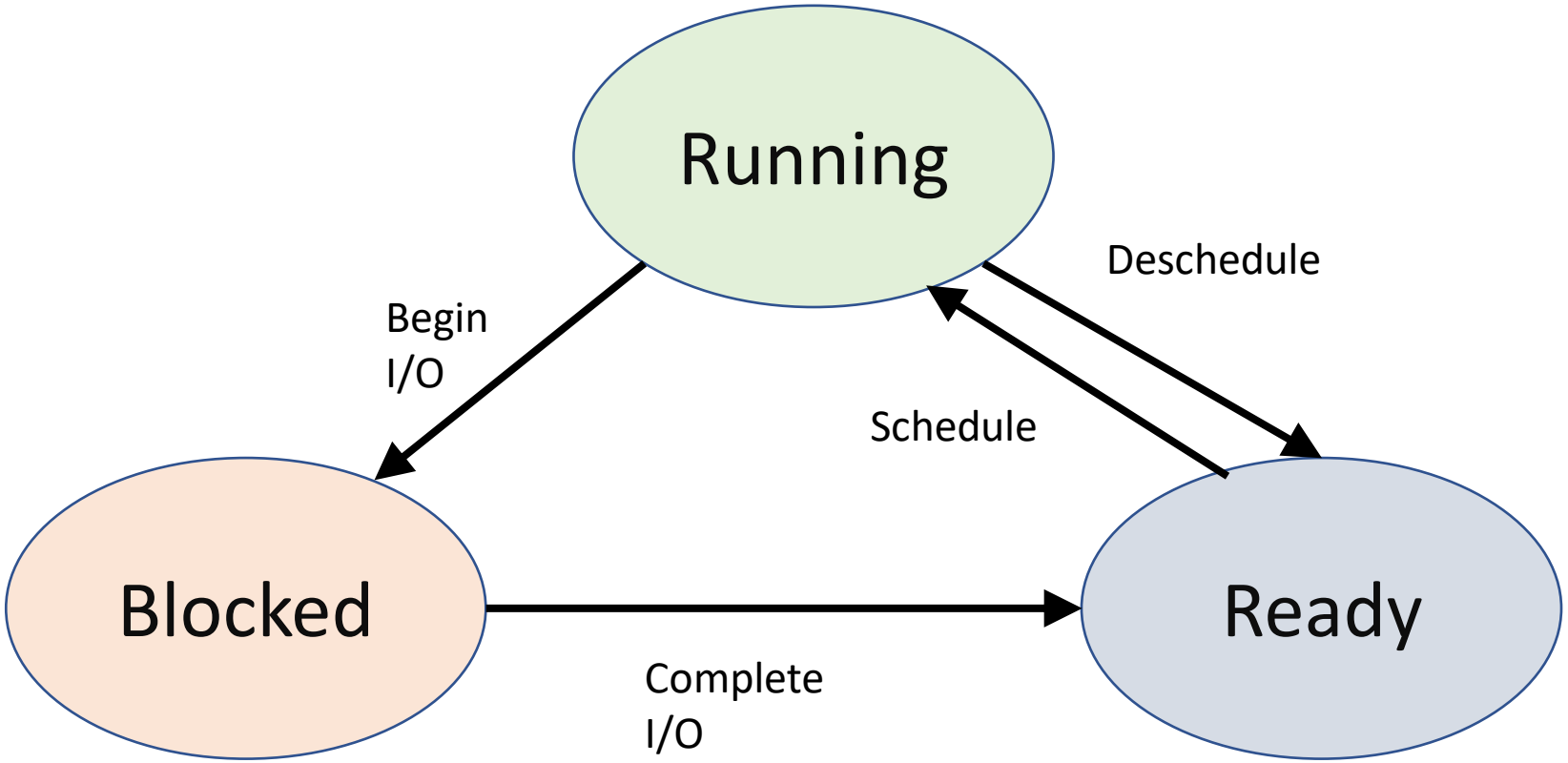
Process State



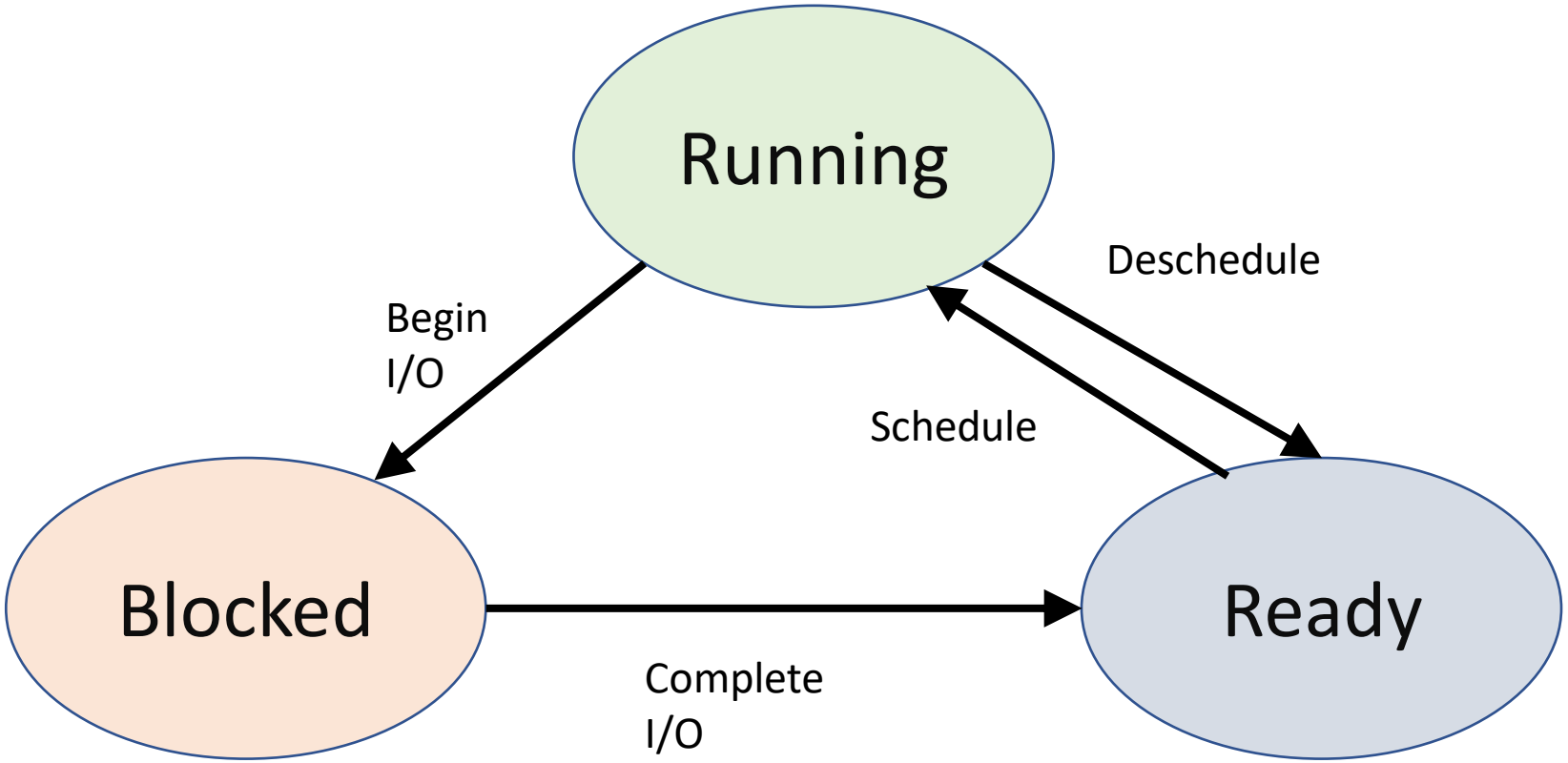
Process State



Process State



Process State



CS 354
Machine Organization and Programming
Lecture 25

Michael Doescher
Summer 2020

Intro to Operating Systems