Limited Direct Execution

How does the OS keep control?

COM S 352
Iowa State University
Matthew Tan Creti

Limited Direct Execution

Time-sharing is the idea that multiple processes can run together on a single machine as though they are in sole control of the machine

How do processes share time?

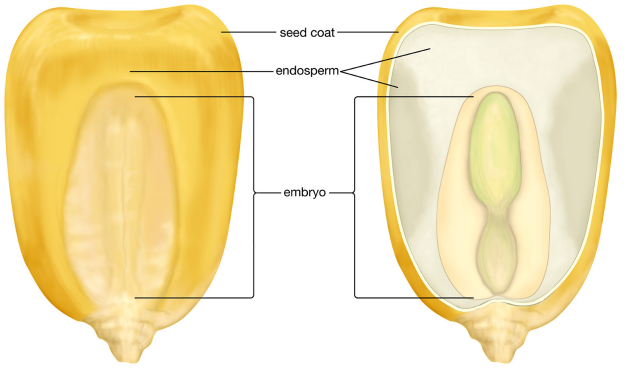
multiprogramming – when a process waits for I/O, the OS can have another take over the CPU

multitasking – each process gets a time-slice, a time limit before the next process gets to execute on the CPU

How does the OS keep control?

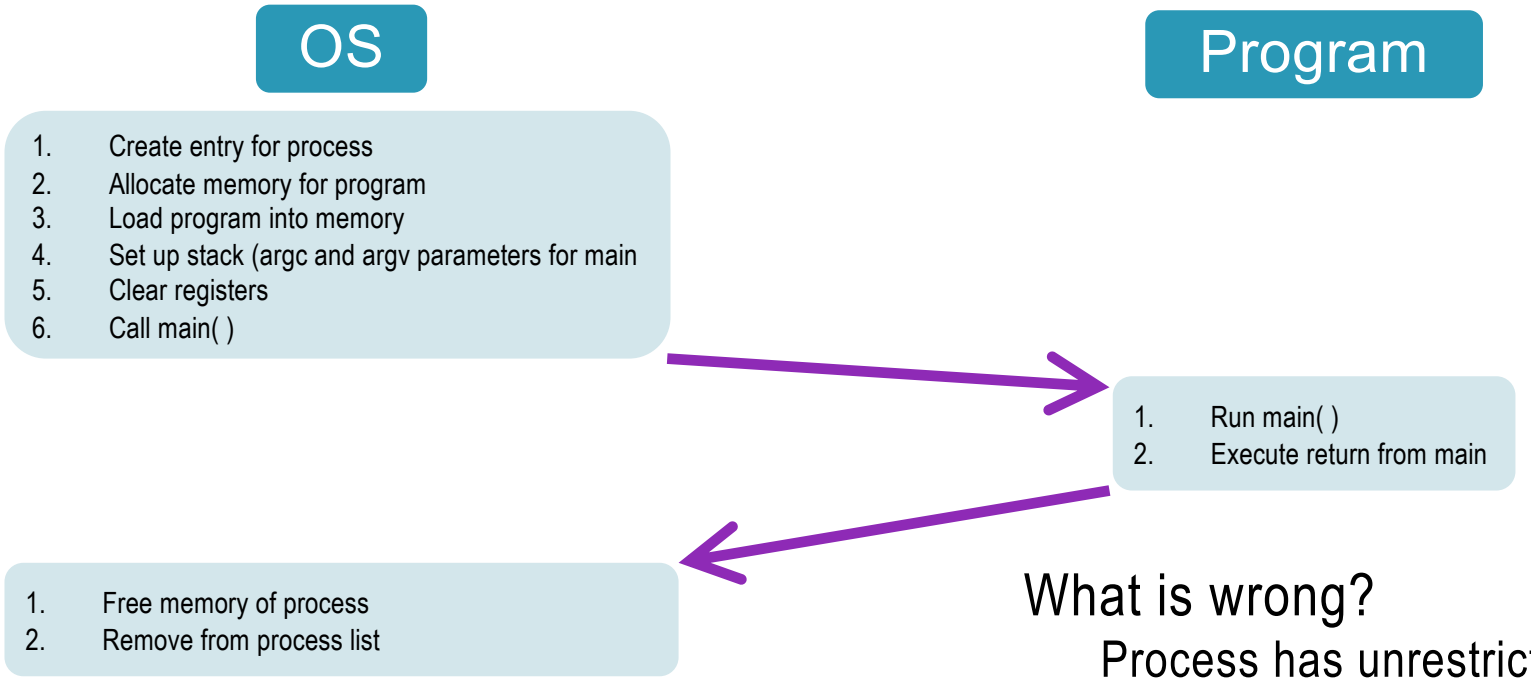
Hardware provides interrupts, kernel mode and user mode

We will see how the OS uses these three mechanisms to implement CPU virtualization



Corn kernel [source]

Direct Execution



What is wrong?

Process has unrestricted access to memory and resources.

OS has no way to switch another process, must wait for program to finish.

Restricting Access with Processor Mode

CPU has bit that indicates if in user mode or kernel mode

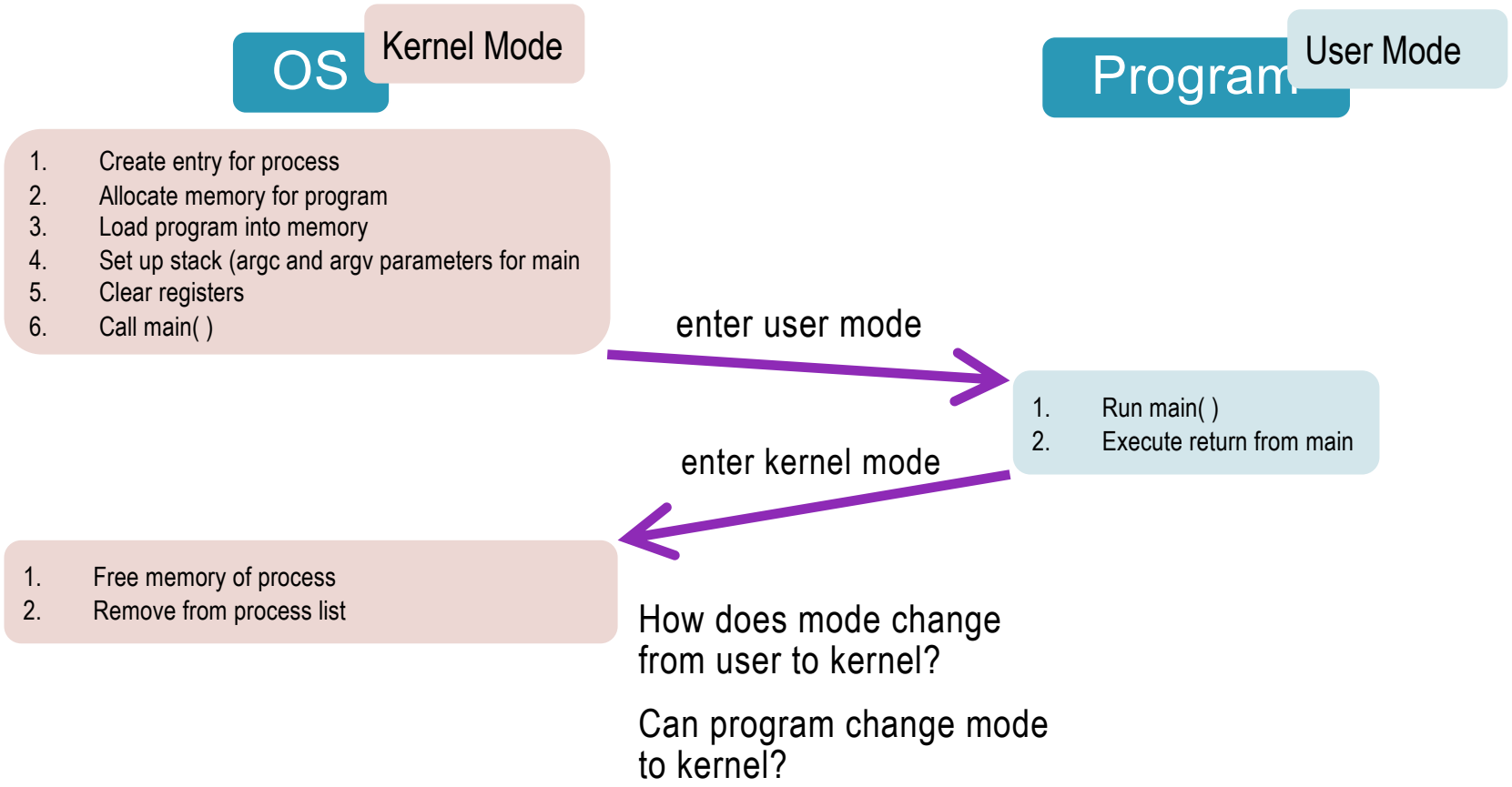
When in user mode, “privileged” instructions not allowed and memory boundaries are enforced

- Cannot read/write outside of address space bounds
- Cannot read/write I/O devices

When in kernel mode, all instructions are allowed

Normal processes only execute in user mode, the OS executes in kernel mode

Processor Mode



System Call

Problem: How can the program perform privileged operations, for example read from a file?

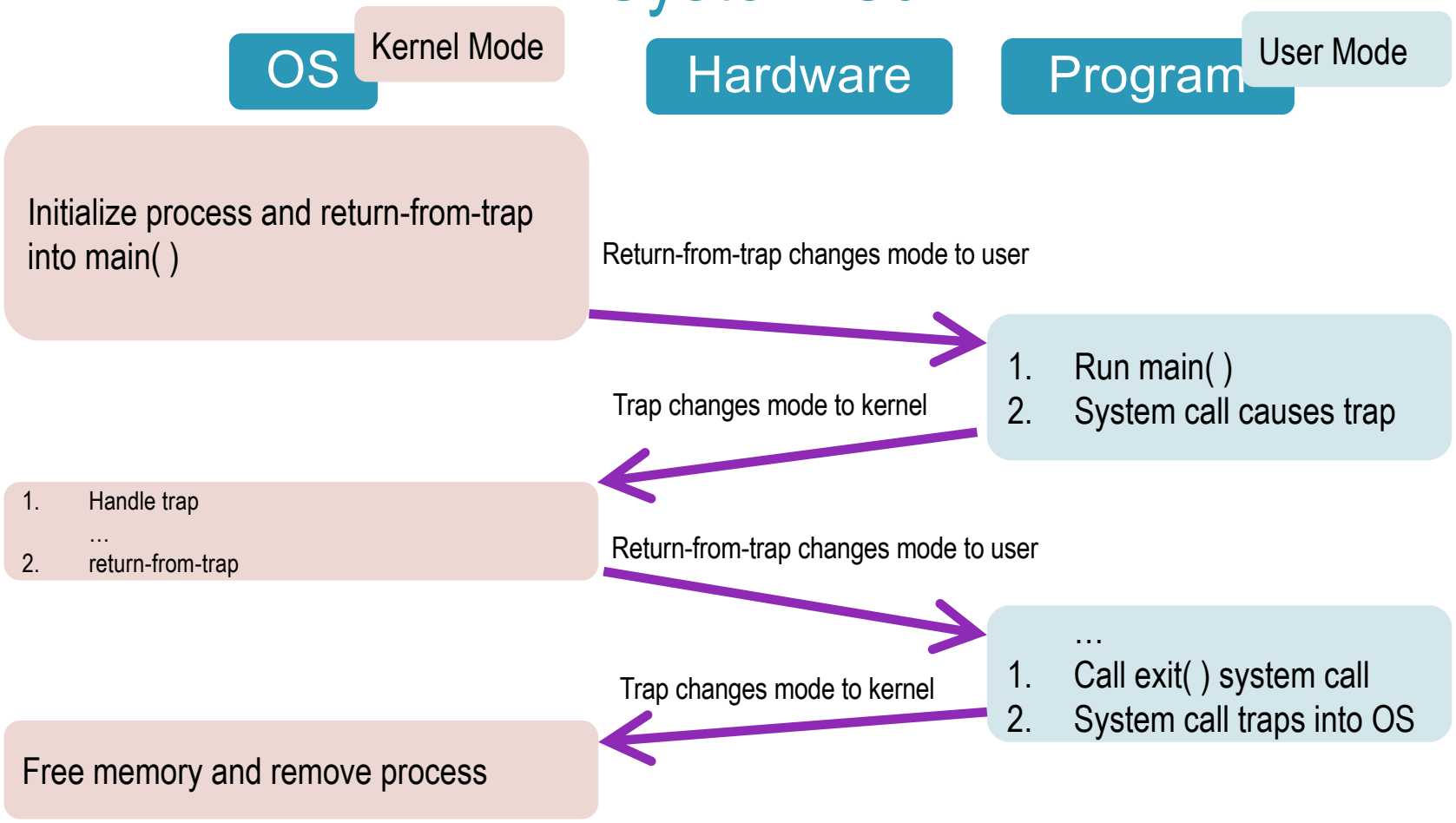
If no modes, user could just make a functional call to kernel libraries, easy!

A system call is different from a normal function call in that it transfers control to the OS.

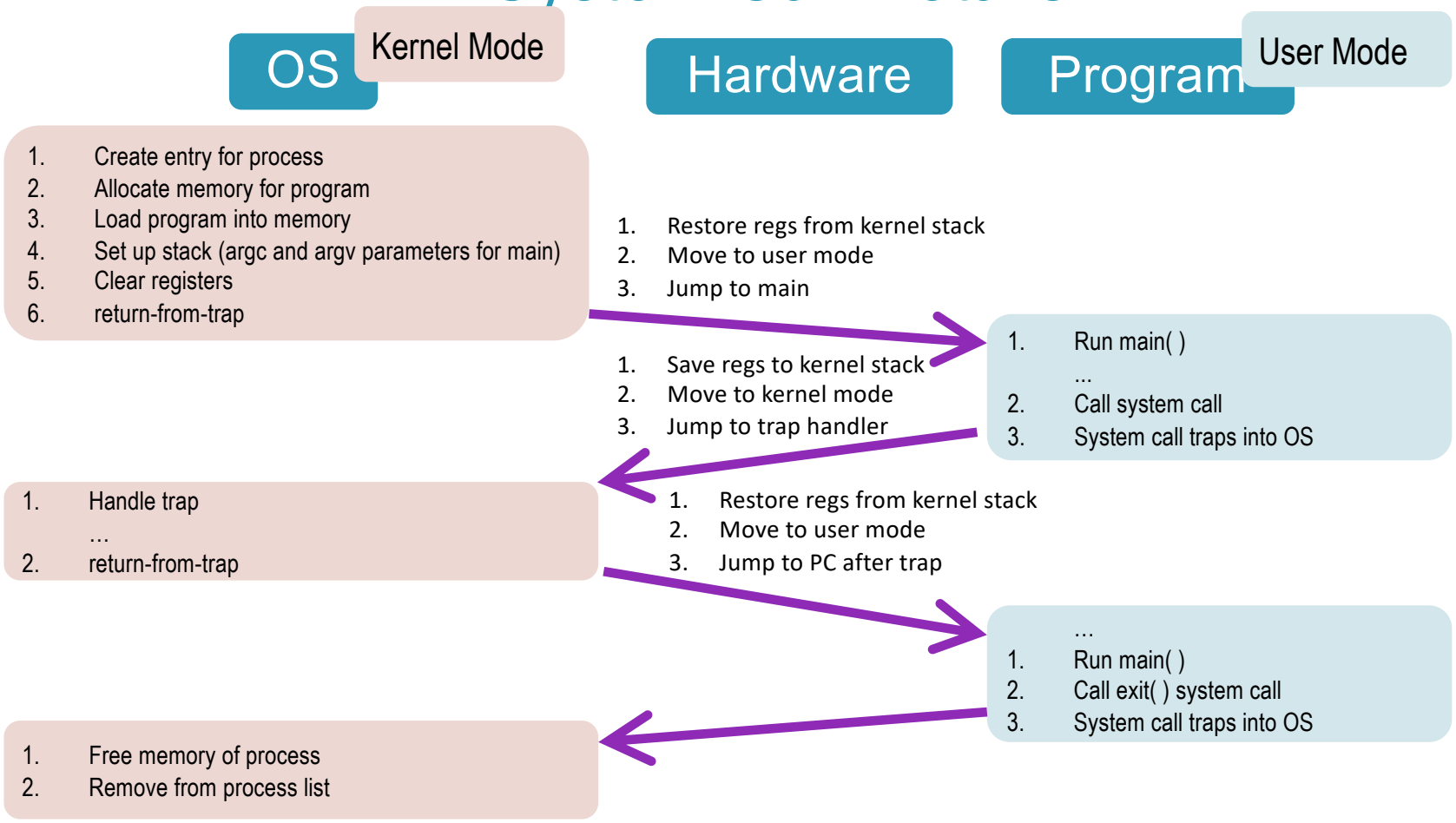
A program initiates a system call by causing a trap (a software generated interrupt)

For example, the RISC V instruction that causes a trap is ecall

System Call

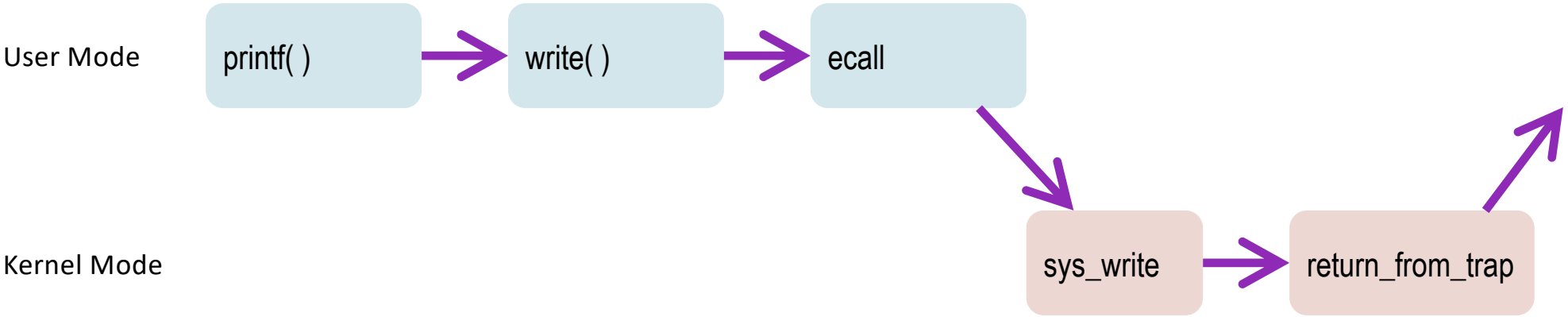

OS
Kernel Mode
Hardware
Program
User Mode
Initialize process and return-from-trap into main( )
Return-from-trap changes mode to user
1. Run main( )
2. System call causes trap
Trap changes mode to kernel
1. Handle trap
…
2. return-from-trap
Return-from-trap changes mode to user
…
1. Call exit( ) system call
2. System call traps into OS
Trap changes mode to kernel
Free memory and remove process

System Call Details



System Call Example

Library functions like printf typically have some code that executes in user mode, it then makes one or more system calls.



swtch.S

```
1  # Context switch
2  #
3  #   void swtch(struct context *old, struct context *new);
4  #
5  # Save current registers in old. Load from new.
6
7
8  .globl swtch
9  swtch:
10     sd ra, 0(a0)
11     sd sp, 8(a0)
12     sd s0, 16(a0)
13     sd s1, 24(a0)
14     sd s2, 32(a0)
15     sd s3, 40(a0)
16     sd s4, 48(a0)
17     sd s5, 56(a0)
18     sd s6, 64(a0)
19     sd s7, 72(a0)
20     sd s8, 80(a0)
21     sd s9, 88(a0)
22     sd s10, 96(a0)
23     sd s11, 104(a0)
24
25     ld ra, 0(a1)
26     ld sp, 8(a1)
27     ld s0, 16(a1)
28     ld s1, 24(a1)
29     ld s2, 32(a1)
30     ld s3, 40(a1)
31     ld s4, 48(a1)
32     ld s5, 56(a1)
33     ld s6, 64(a1)
34     ld s7, 72(a1)
35     ld s8, 80(a1)
36     ld s9, 88(a1)
37     ld s10, 96(a1)
38     ld s11, 104(a1)
39
40     ret
```

store registers to old context
sd = store doubleword command

load new context to registers
ld = load doubleword command