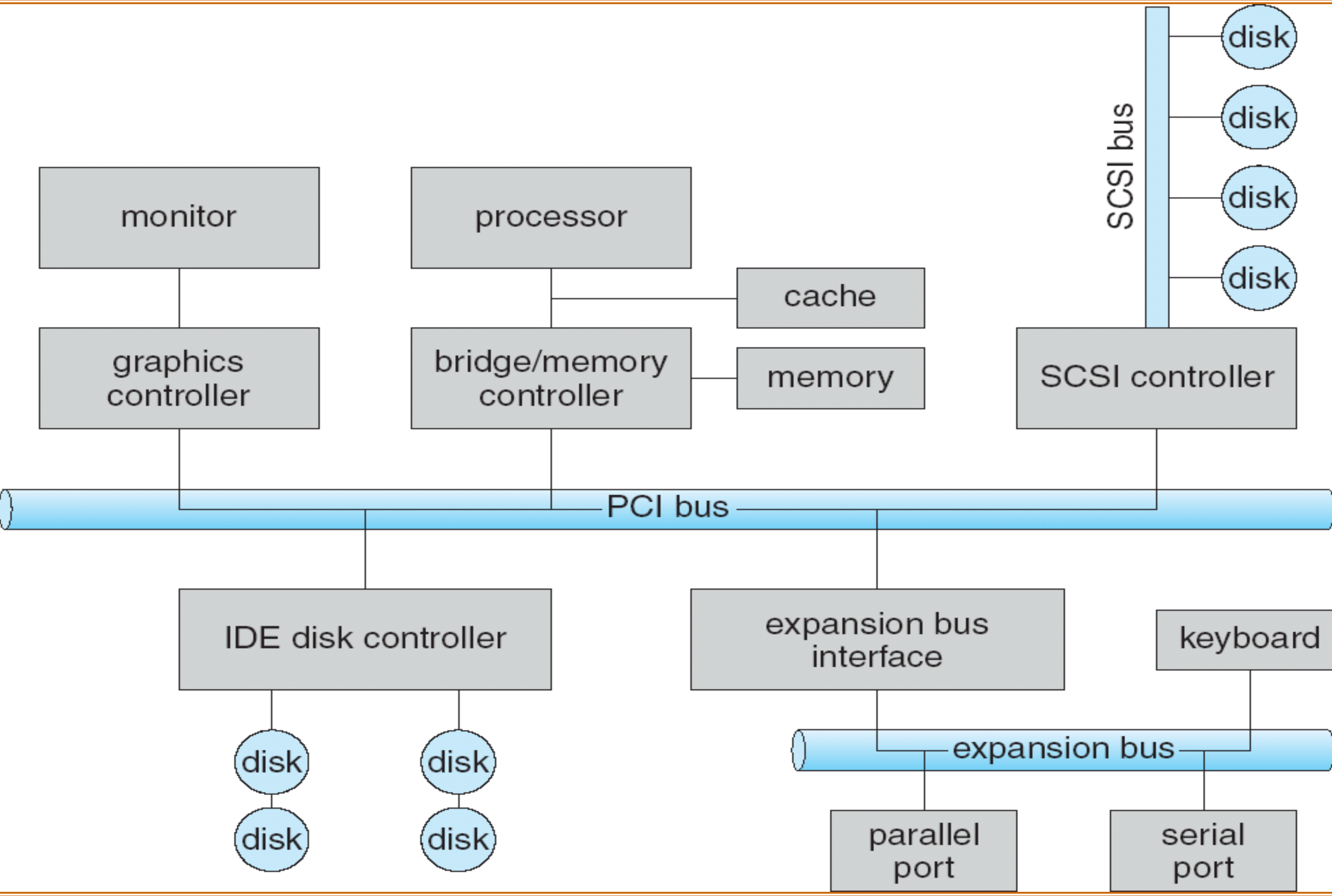
Operating Systems Principles

Recap of Some Basics

Slides are adapted from Prof. Bhuvan Urgaonkar


monitor
processor
cache
graphics controller
bridge/memory controller
memory
SCSI bus
disk
disk
disk
disk
SCSI controller
PCI bus
IDE disk controller
expansion bus interface
keyboard
disk
disk
disk
disk
expansion bus
parallel port
serial port

Summary

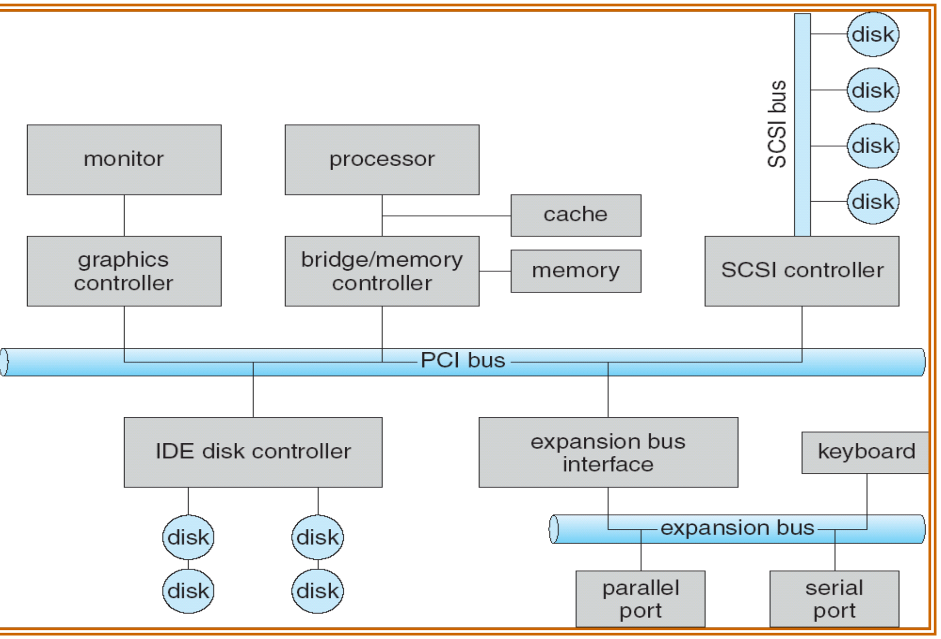
- Sharing of resources can cause problems related to
  - Correctness
    - E.g., a program's register contents being corrupted by another
  - Fairness
    - E.g., a program not getting enough CPU cycles
- For certain resources OK to have hardware-driven sharing
- For others, would like to have software-driven sharing

Summary

- Need for a software **other than the programs** that facilitates correct and fair sharing of hardware resources
  - **The operating system**
  - May require special support from H/W (e.g., memory protection for register sharing that we saw in last class)

Programs

OS



Hardware

- Sharing of resources could be done by h/w, OS, or even programs
  - E.g., Sharing of busses done by h/w
  - E.g., sharing of CPU cycles done by OS
  - E.g., sharing of registers done by programs themselves
- Discuss other resources and their sharing

A More Complete Definition

Software that offers following functionality

- OS is a **resource allocator**
  - Divides up resources among programs
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer
- OS **virtualizes** resources
  - Hides hardware details and offers programs easier interfaces to work with

What We Will Study

- How the OS allocates, controls, and virtualizes resources to ensure safety (isolation), performance, and fairness

- Before diving into OS design, we need to understand some basics
- Certain aspects of CPU operation
  - Instruction Set Architecture (ISA)
  - Instruction execution
  - CPU modes
  - Interrupts and Traps
- Certain basics about programs
  - Compilation, Linking, and Loading
- Certain aspects of systems programming
  - Using system calls

Recap of Basics

- Key to OS acting as a control program and a resource controller are following CPU/memory facilities
  - #1: Multiple CPU modes and privileged instructions
  - #2: The notion of traps (part of CPU design)
    - Related concepts: Interrupts and signals
  - #3: Memory address translation and protection facilities (hardware called memory controller)
    - Related: Translation Lookaside Buffer (TLB)

ISA

- The set of instructions the CPU offers
    - Interface provided by CPU to software
        - That is, to both programs and OS
- Examples
    - Load (from memory to register)
    - Store (from register to memory)
    - Jump
    - ALU instructions (add, sub)
    - Halt (the computer)
    - IO instructions, and many more

Instructions Involving Registers and Memory

- Generic examples
    - Load Reg, @MemAdd
    - Store Reg, @MemAdd

    - Add Reg1, Reg2, Reg3

The Need for “Privileged” Instructions

- Can a program execute all instructions in the ISA?
  - Should it?
- Answer: No
- Why?
- E.g., what could go wrong if a program were allowed to execute the “halt” instruction?
- E.g., what could go wrong if a program was allowed to execut IO instructions? (will become more clear later in the course)

Privileged Instructions

- **Key Idea #1:** Let some (appropriate) subset of ISA be privileged
  - Only the OS gets to execute privileged instructions
- The CPU is designed in a way to disallow programs to execute privileged instructions
- But how would the CPU know the difference between a program and the OS?
  - An instruction is an instruction!

Dual CPU Mode

- **Dual-mode** operation allows OS to protect itself from programs and programs from each other
  - **User mode** and **OS/kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
- If user code executes priv. instruction, CPU enters a special error-like state and control jumps to OS

Dual CPU Mode

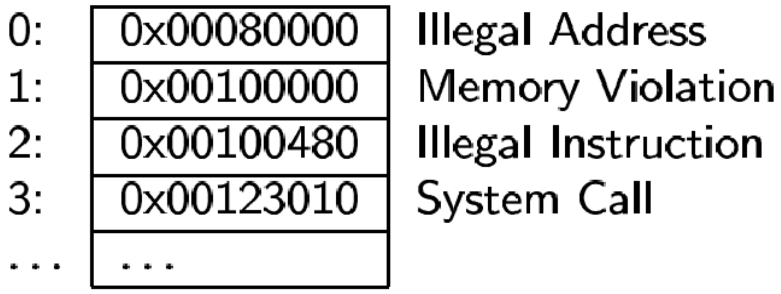
- OS runs with CPU in kernel mode
- Is responsible to ensure programs run with CPU in user mode
- What is required to realize the above?
  - OS is the first software to run!
    - The booting up of the OS
  - OS has the ability to change CPU mode from kernel to user
  - Programs have the ability to change CPU mode from user to kernel

Traps

- **Key Idea #2:** Let the CPU be designed so it switches to kernel mode whenever certain conditions that require OS attention occur
  - These special conditions are called traps
  - Example 1: Programs are offered a special instruction via which they can raise a trap
    - Used to implement system calls
  - Example 2: Segmentation fault, division by zero, more to come

Traps

- On detecting trap, CPU must:
  - Save process state
  - Transfer control to trap handler (in OS)
    - CPU indexes *trap vector* by trap number
    - Jumps to address
  - Restore process state and resume

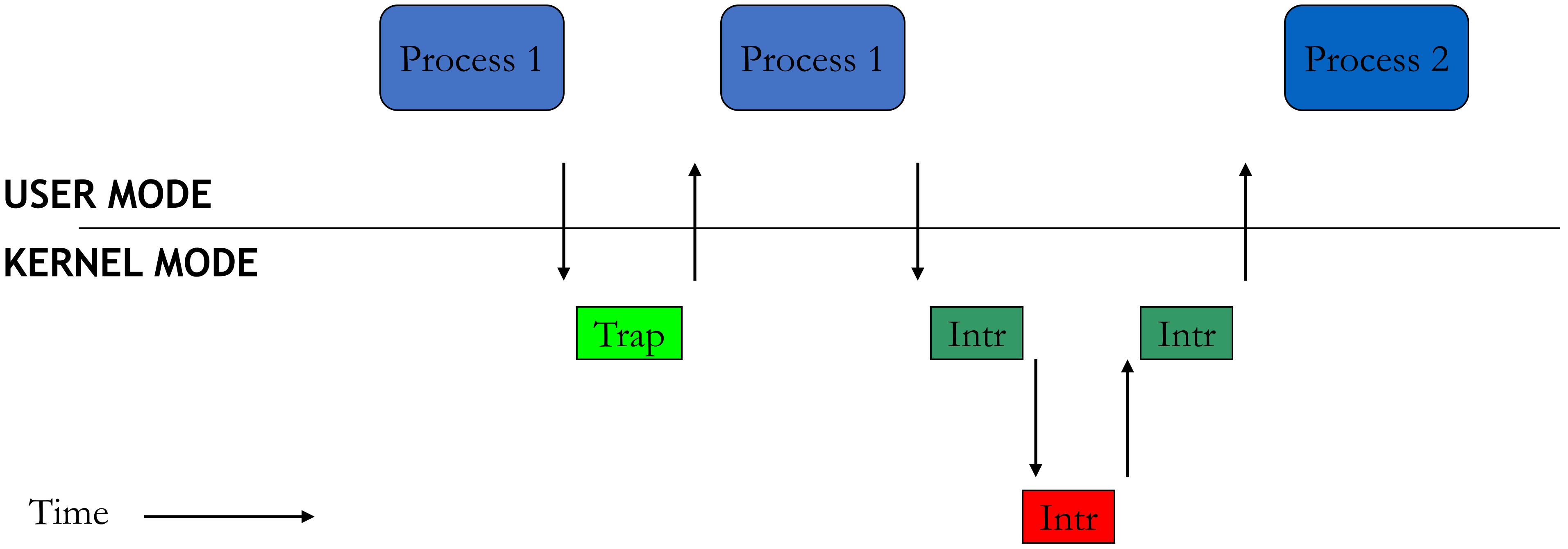


0:	0x00080000	Illegal Address
1:	0x00100000	Memory Violation
2:	0x00100480	Illegal Instruction
3:	0x00123010	System Call
...	...	

Interrupts

- Special conditions **external to the CPU** that require OS attention
  - Note difference from traps
- CPU designed to switch to kernel mode upon detecting an interrupt
- Example: A keystroke raises an interrupt

Interrupts and Traps



- Only two ways to enter kernel mode from user mode