Operating Systems
Chapter 1

Marie Roch

contains slides from:

Tanenbaum 2001, 2008, 2015

Silbershatz, Galvin, and Gagne 2003

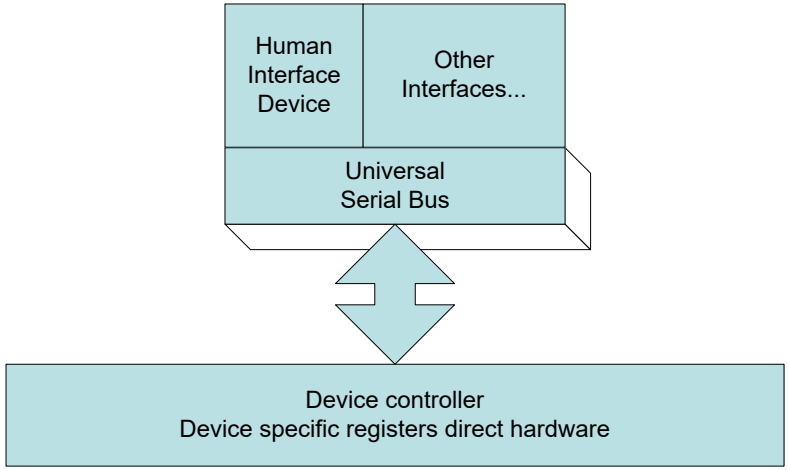
What is an Operating System?

- It is an extended machine
  - Hides the messy details which must be performed
  - Presents user with a virtual machine, easier to use

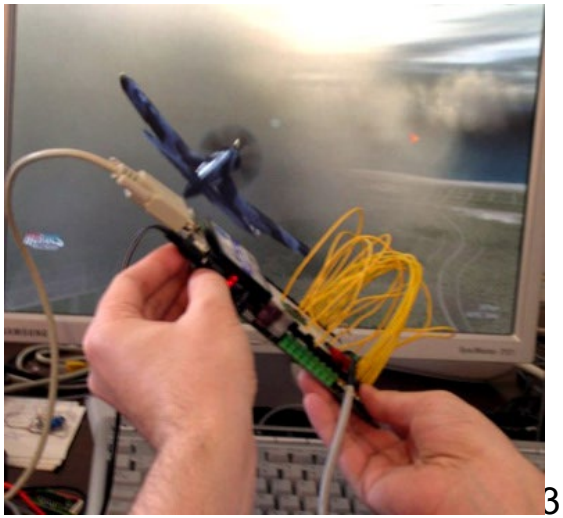
- It is a resource manager
  - Each program gets time with the resource
  - Each program gets space on the resource

What is an operating system?

- Extended machine
  - abstraction
  - standardization



http://www.kraftbrands.com/activegame/pad.aspx



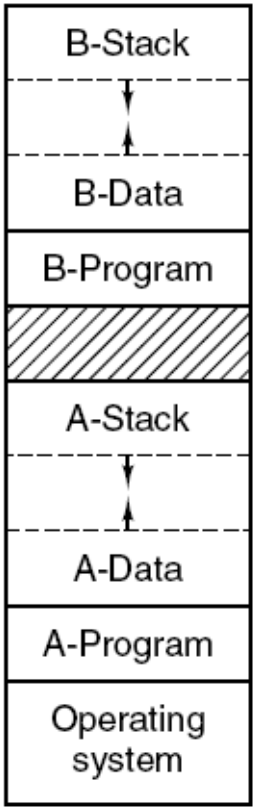
http://techepics.com/files/_PC_controller.jpg

What is an operating system?

- Resource manager
  - sharing
    - time
    - space
  - protection

http://texasholdemblogger.files.wordpress.com/2007/08/13campsite-outhouse.gif

Time sharing:
One after another please…



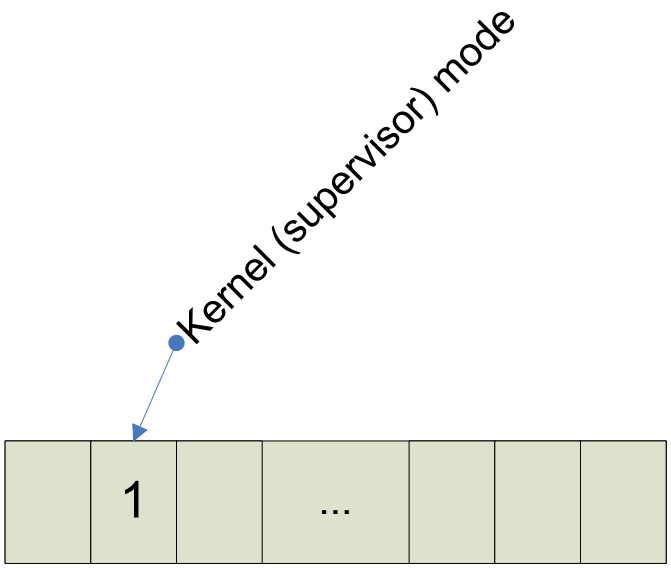
Space sharing

Privileged instructions

- To maintain integrity of the system, not every process should be able to execute every type of instruction.

- User mode vs. Supervisory mode
  - Prohibit privileged instructions in user mode
  - Controlled by a bit in the program status word (PSW)
  - Also known as kernel-mode

Privileged instructions



program status word (PSW)

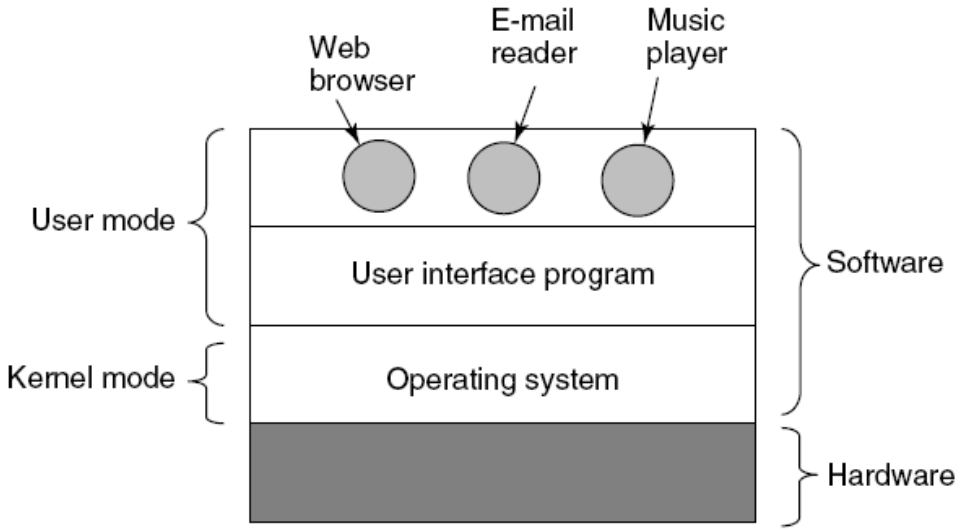


Figure 1-1. Tanenbaum 2008

What type of instructions should be privileged?

20,000 Foot view of the last 60+ years

Vacuum tubes

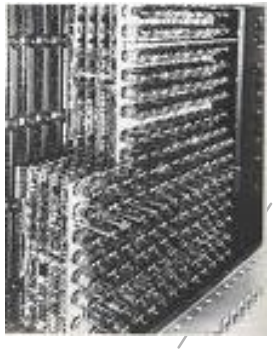
Integrated circuits & multiprogramming

1945

1955

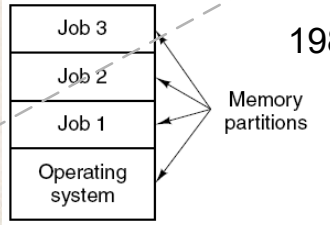
1965

1980

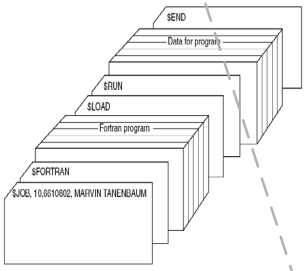


Thomson invented 1897
(1950s memory shown)

Kilby & Noyce's first IC (1959)



Bardeen & Bratt's
transistor (1947)

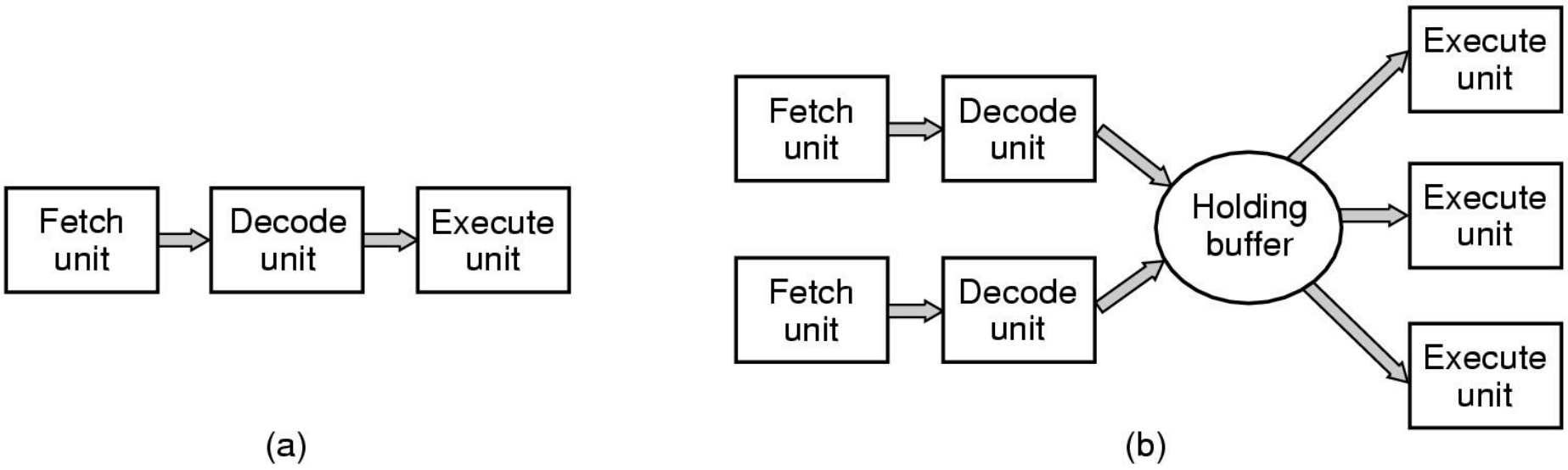


Transistors and batch systems

Personal computing

1955 1965 1980 2008

Processor Architecture



(a) A three-stage pipeline
(b) A superscalar CPU

Buses

common buses

DMI – Direct media interface
ISA – industry standard architecture
PCI – peripheral component interconnect
PCI Express
SATA – serial advanced technology attachment
SCSI – small computer system interface
USB – universal serial bus

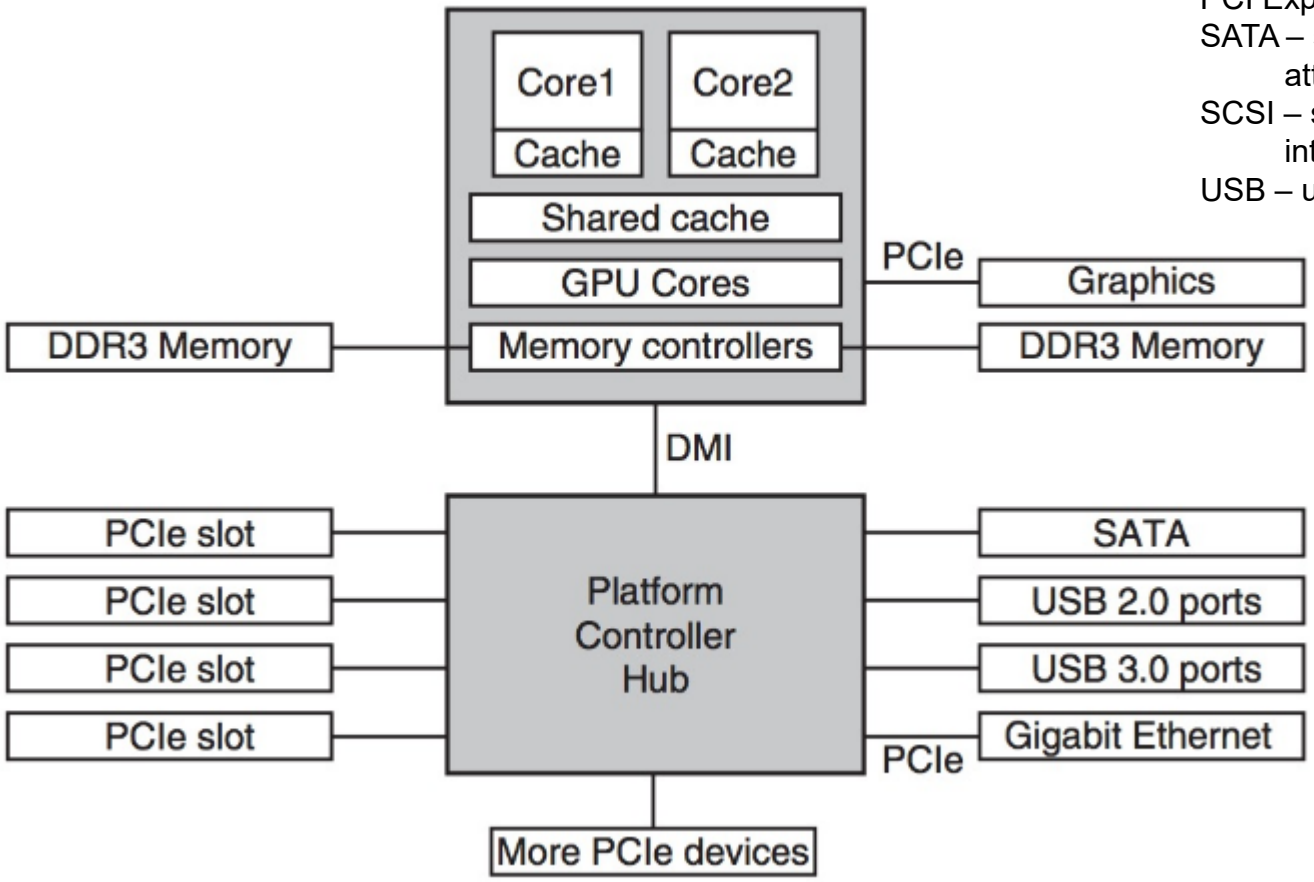


Figure 1-12. The structure of a large x86 system

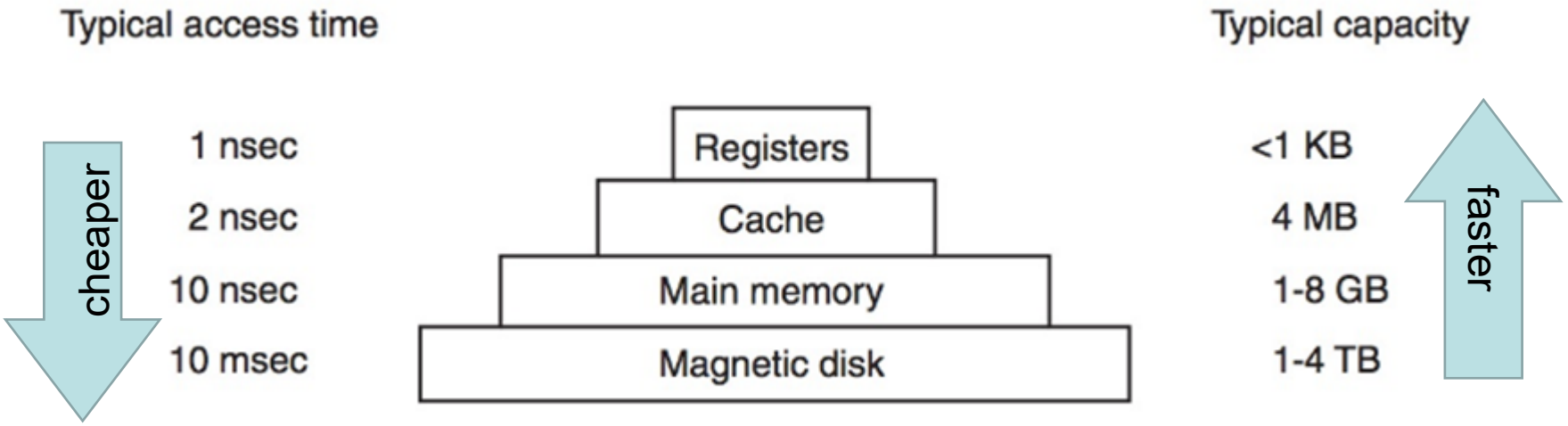
Memory

Typical access time		Typical capacity
1 nsec	Registers	<1 KB
2 nsec	Cache	4 MB
10 nsec	Main memory	1-8 GB
10 msec	Magnetic disk	1-4 TB

Note: Mid 2010s numbers & technologies, different numbers today, but the same idea…

Fig 1.9 Tanenbaum and Bos

Caching

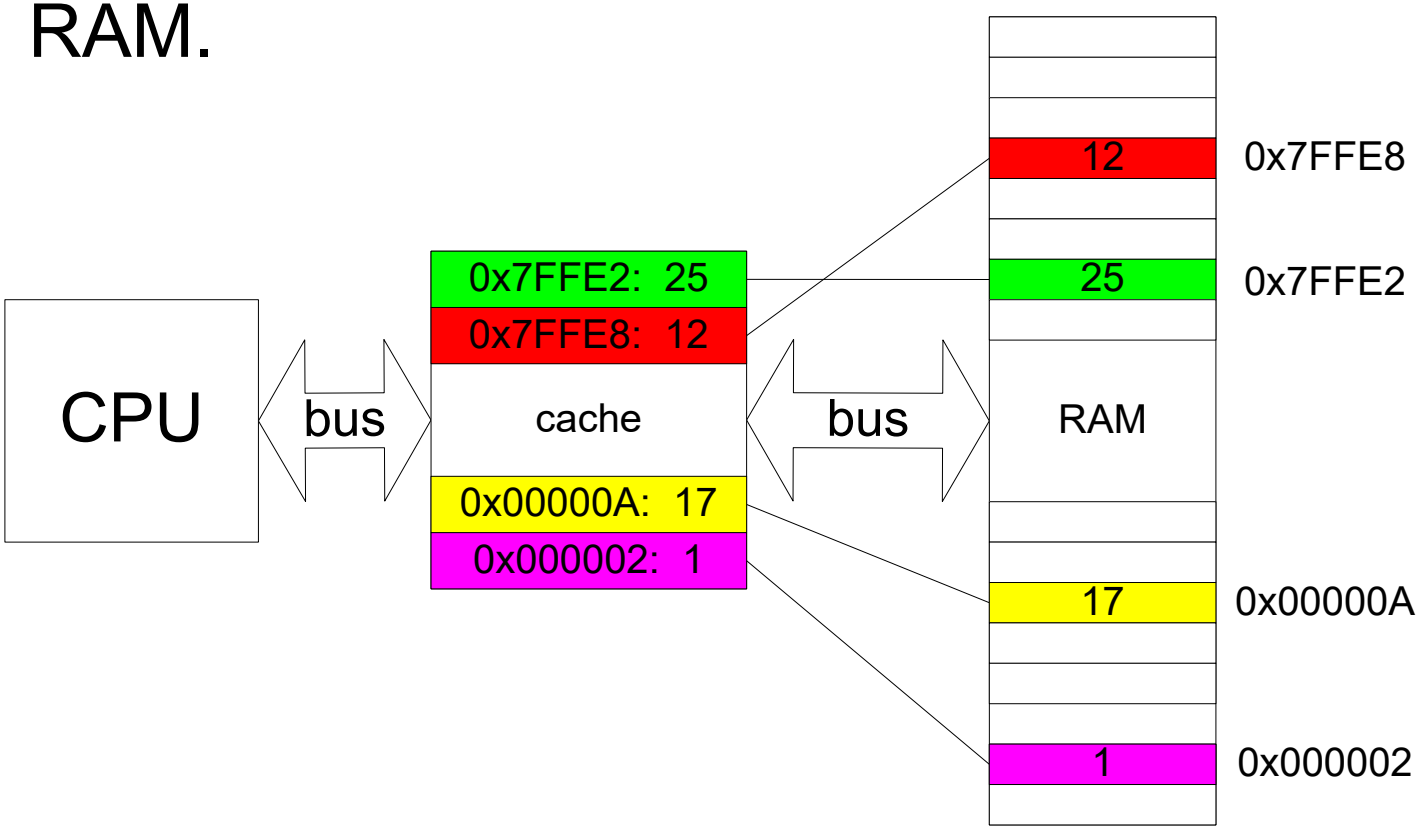


Basic concept of a cache:

- Store a frequently used subset of a slow device on a faster one.
- On access, check the cache first, then the slower device.
- Process should be *transparent* to user.

HW Review: Memory cache

- Small memory is placed between the CPU and RAM.



Memory Cache

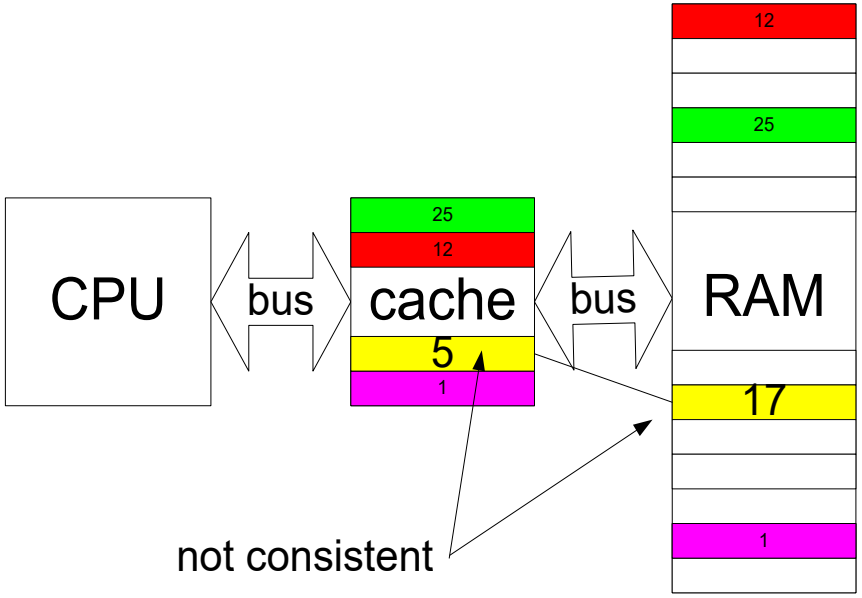
- **How the cache works**
  - CPU Access/Fetch
    - Address sent out on bus
    - Cache searches table for address in O(1) time
    - Search result:
      - HIT: Address found, return contents of address
      - MISS: Address not in cache
        - » Request sent to RAM
        - » Result is saved into cache (might overwrite another entry)

HW Review: Memory Cache

- **How the cache works**
  - CPU Write
    - Address and data sent out on bus
    - Cache updates/inserts into cache
    - *Might* also write data to RAM (called write-through)
  - At times, a current cache entry will have to be overwritten when a new one is needed.

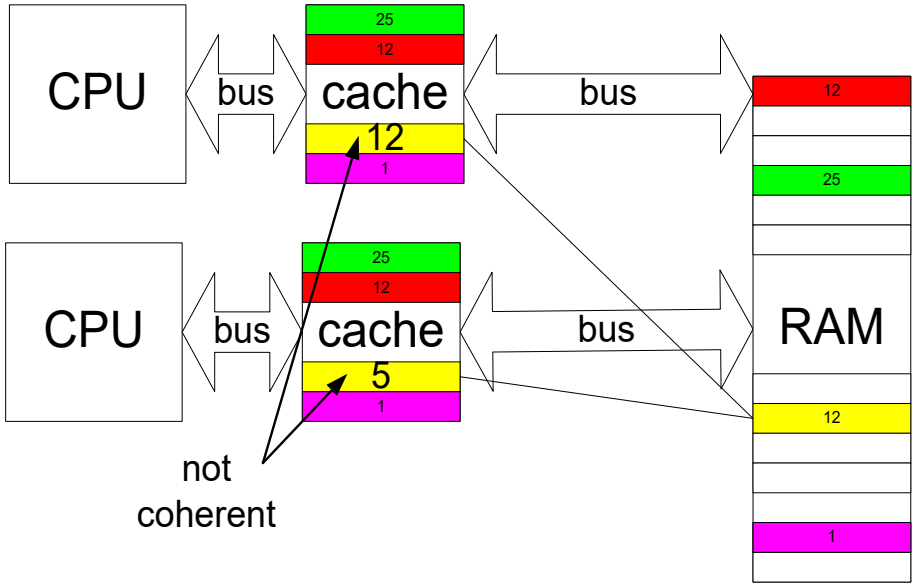
HW Review: Memory Cache

- **Cache issues**
  - Consistency: Addresses in the cache have the same value as those in RAM.



HW Review: Memory Cache

- Coherency:
  - Issue when multiple CPUs share memory
  - Any address that is in more than one CPU's cache must have the same value in each cache it is present.



Multithreaded and Multicore Chips

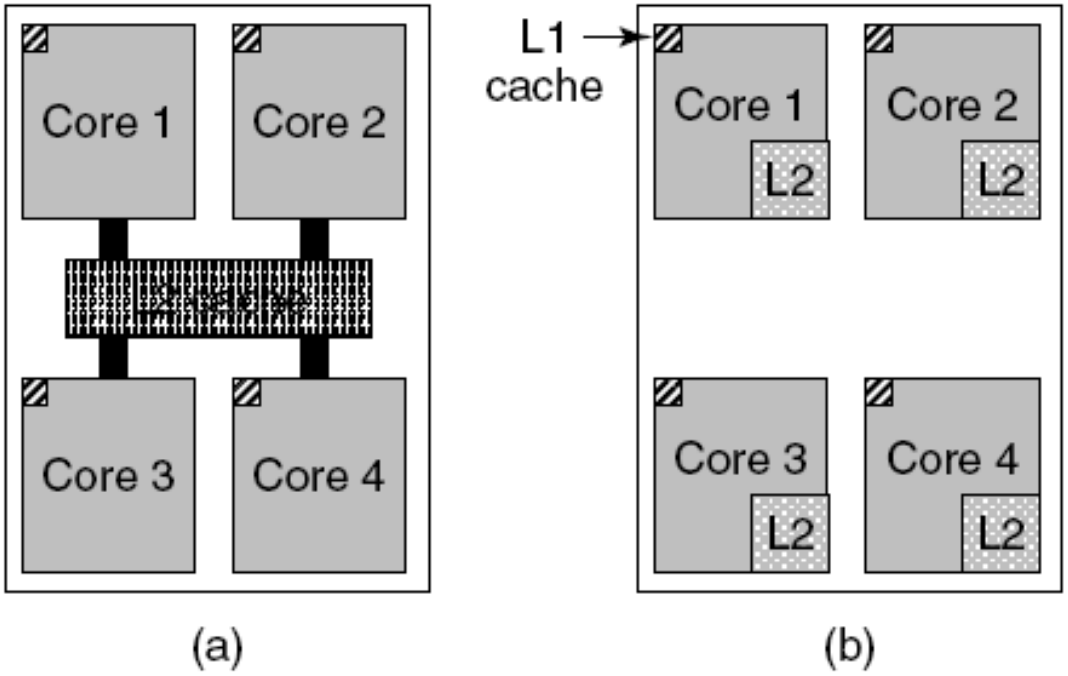
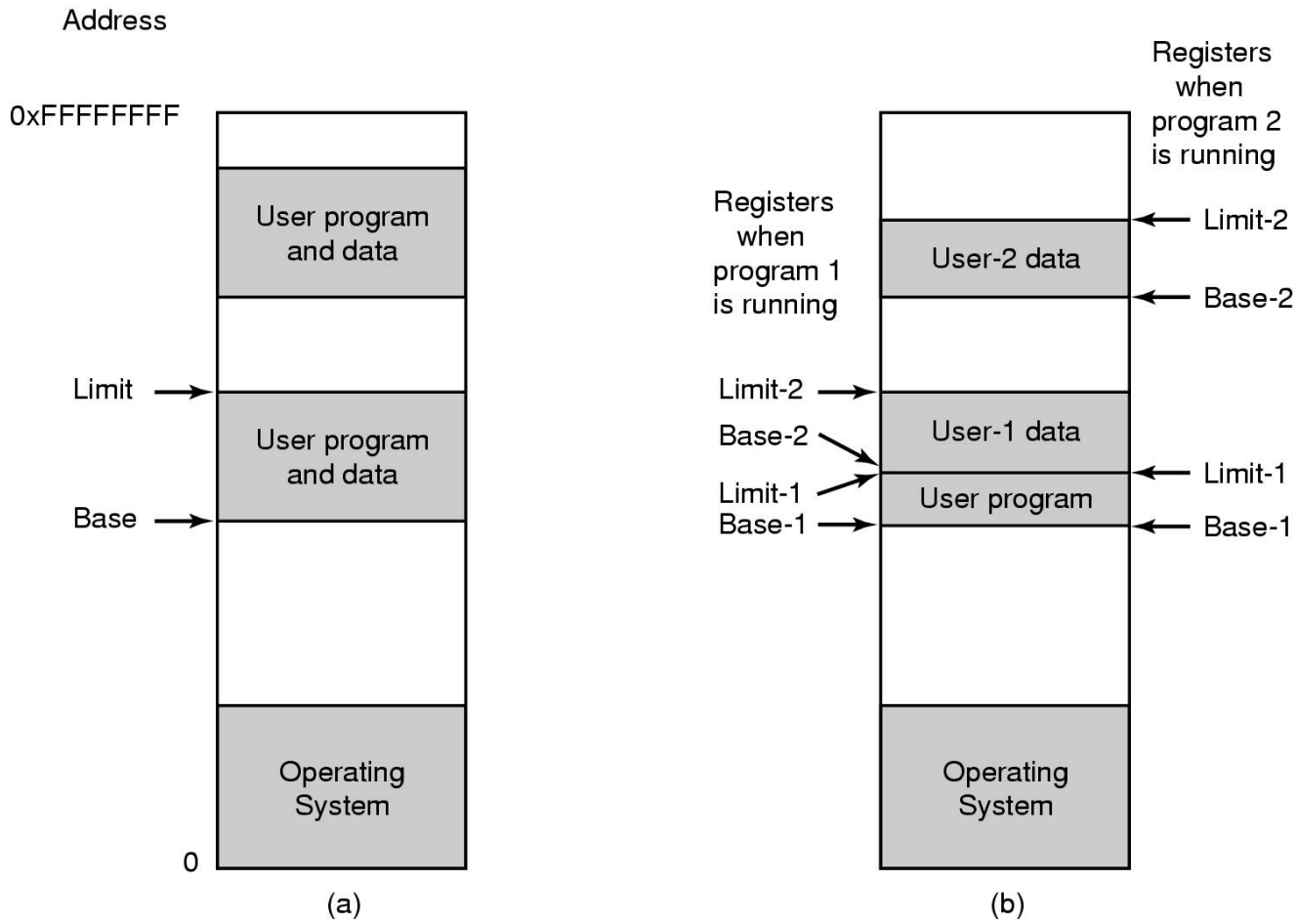


Figure 1-8.

(a) A quad-core chip with a shared L2 cache.

(b) A quad-core chip with separate L2 caches.

HW Review: Memory protection

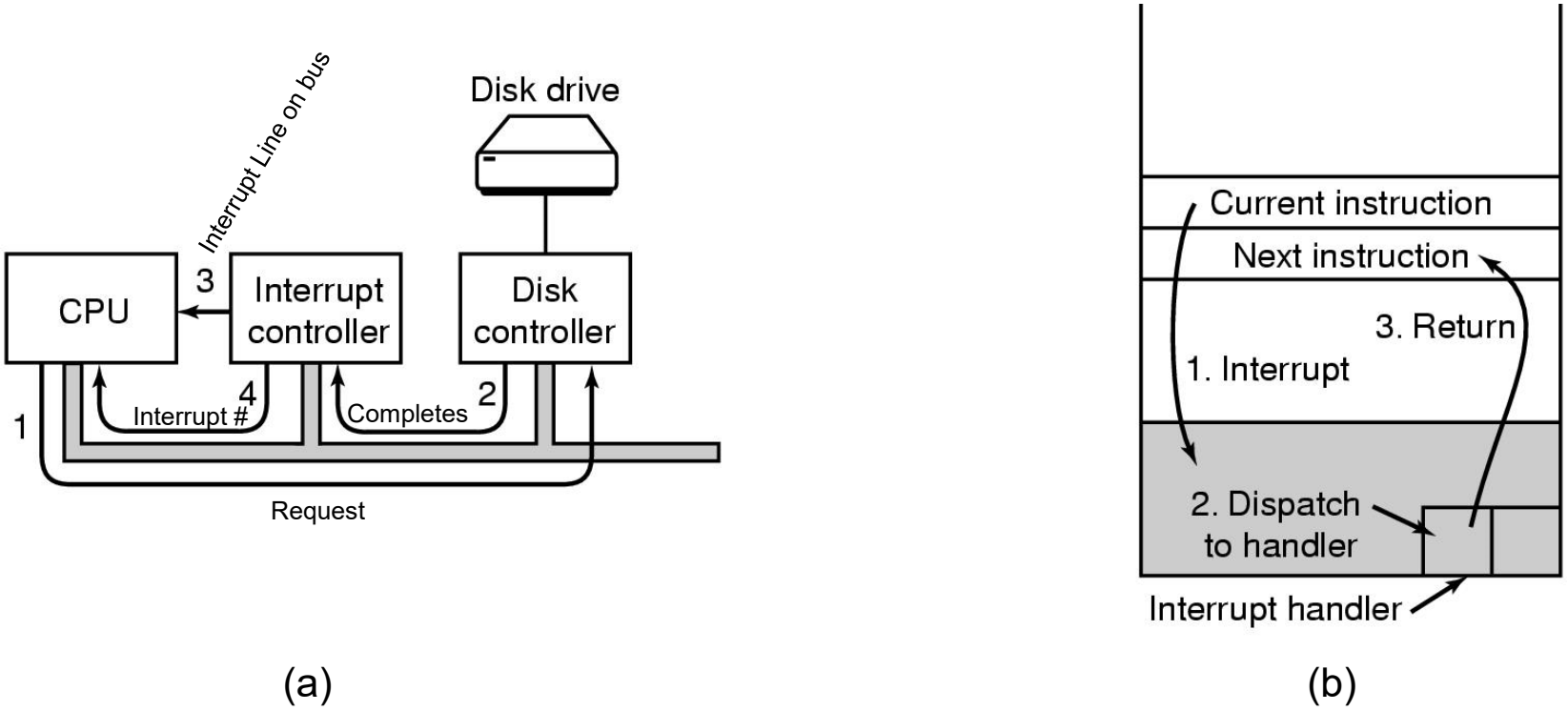


One base-limit pair and two base-limit pairs

HW Review: I/O Devices

- All Input/Output (I/O) mediated by a controller
- Two types of I/O
  - Programmed: CPU polls device
  - Interrupt driven: Device signals upon completion
- I/O can be
  - Synchronous
  - Asynchronous

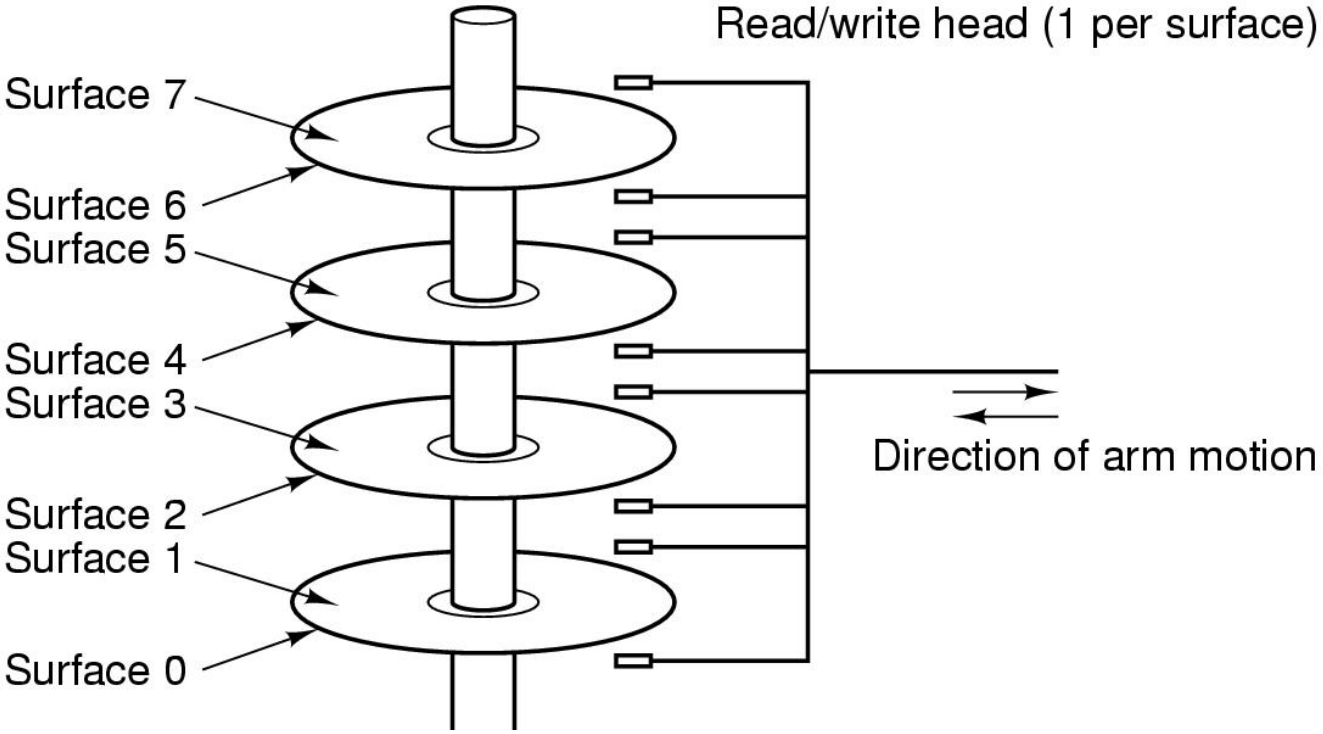
I/O Devices



(a) Steps in starting an I/O device and getting interrupt

(b) How the CPU is interrupted

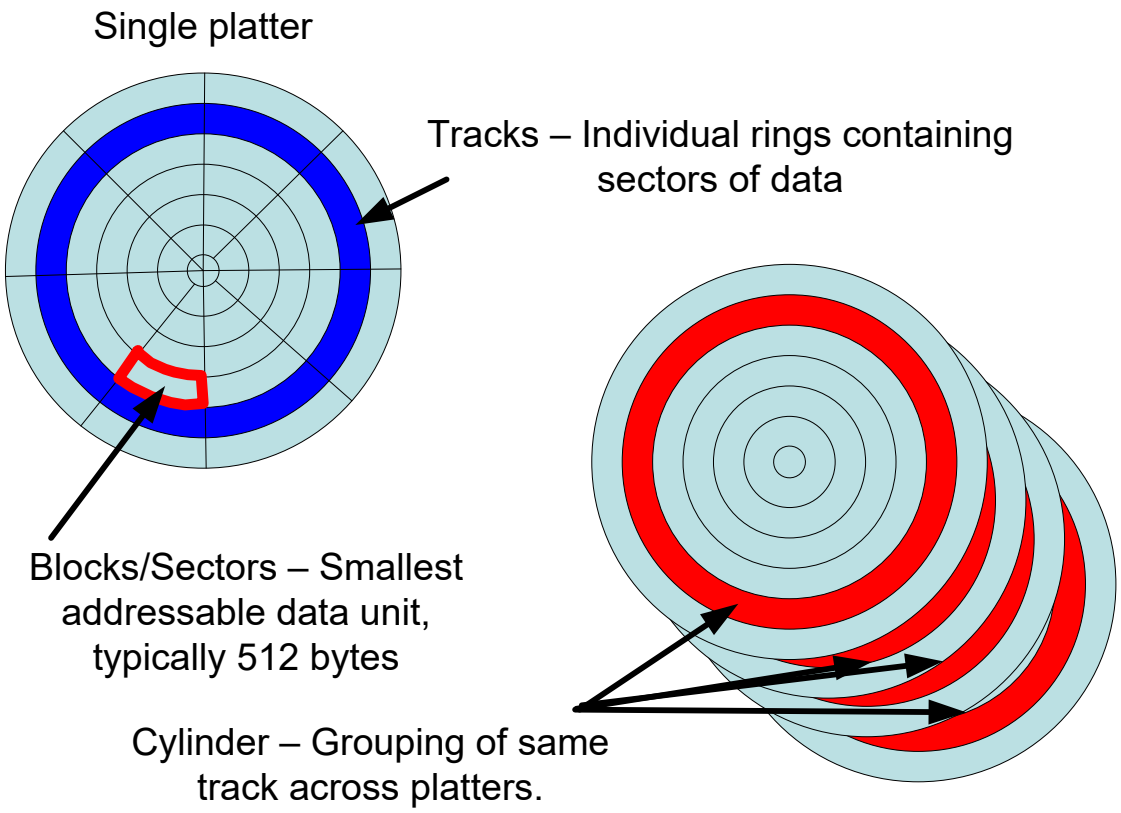
HW Review: Magnetic disks



Structure of a disk drive

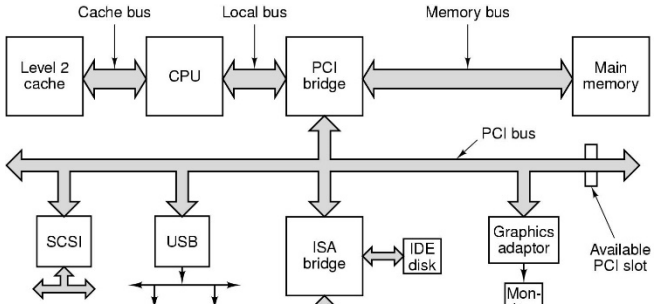
HW Review: Magnetic disks

- Anatomy of a disk drive continued



HW Review: Direct Memory Access (DMA)

- Problem: CPU-managed data transfer between peripheral and RAM consumes CPU resources.
- DMA:
  - CPU indicates source/destination RAM area and instructs controller on what should be done.
  - CPU can continue executing during peripheral/RAM transfer as long as it does not need to access RAM.
  - Controller executes operation and tells CPU when it is done via interrupt.



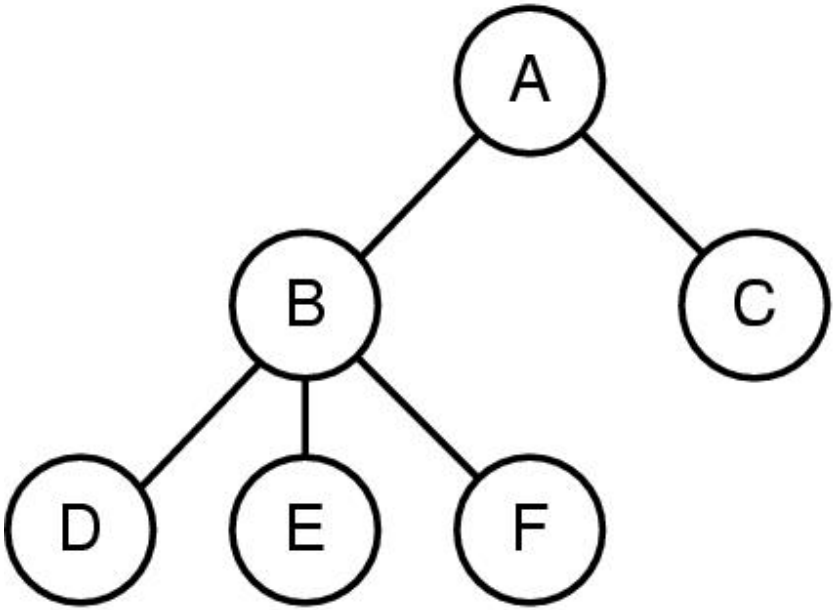
Simplified View of Interrupt Handling

1. Device signals interrupt controller by placing signals on bus lines.
2. Controller signals the CPU and places an interrupt number associated with the device on the bus.
3. If the CPU has interrupts enabled:
    1. Save PC, SP, & PSW.
    2. Shift to supervisor mode.
    3. Execute an interrupt service routine associated with the interrupt selected from the interrupt service table.
    4. Restore PC, SP, & PSW and continue executing process. (Note that not all operating systems will return to the point right away.)

Operating System Concepts: Bootstrap

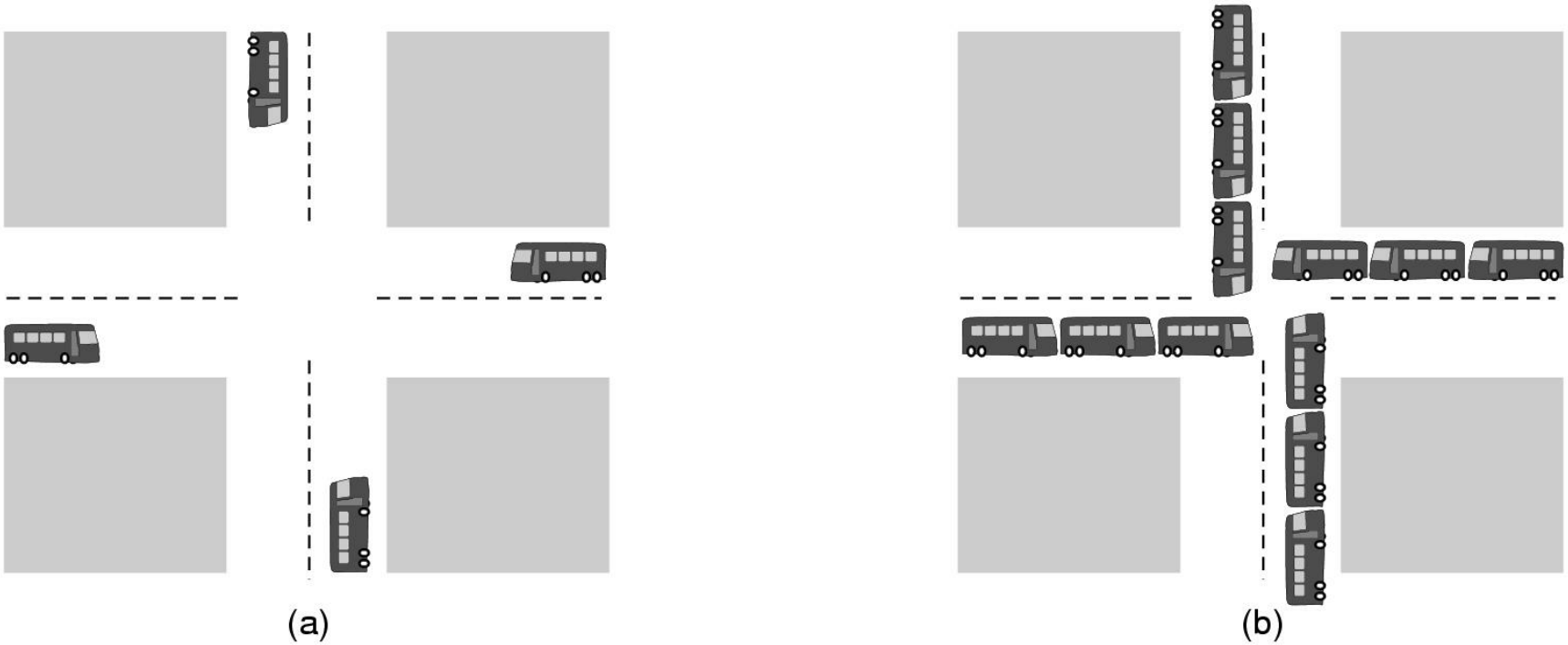
- How do we load the operating system?
- Simplified overview for Intel architectures:
  - Basic input/output system (BIOS) initializes and executes the power on self test (POST)
  - BIOS queries buses and performs low level configuration of the peripherals (plug and play)
  - BIOS loads the boot sector of the designated I/O device.
  - The boot sector contains a small program which typically loads and executes a more sophisticated loader program.
  - The loader program loads the operating system and the initial process is started.

Concepts: Processes



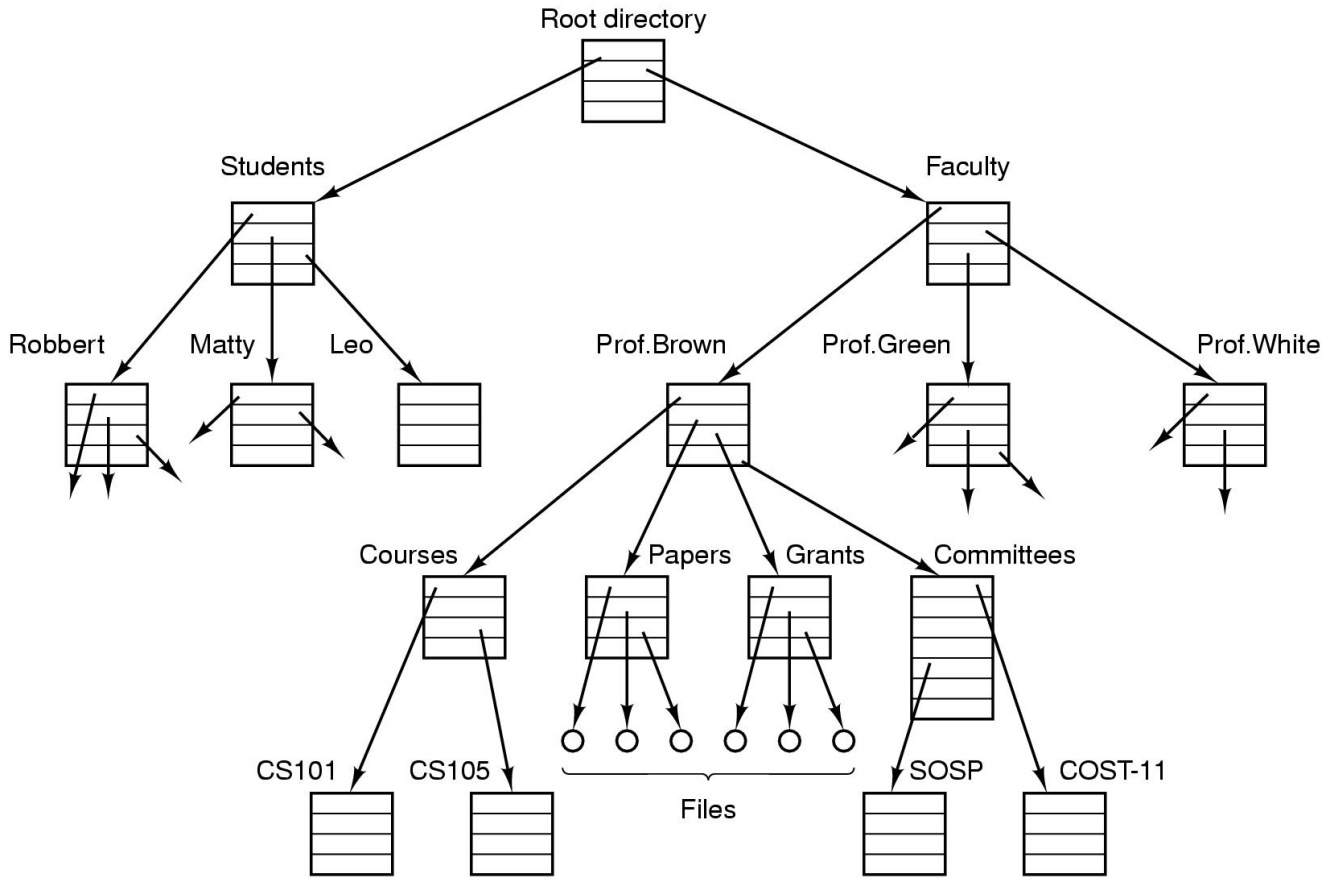
- A process tree
  - A created two child processes, B and C
  - B created three child processes, D, E, and F

Concepts: Deadlocks



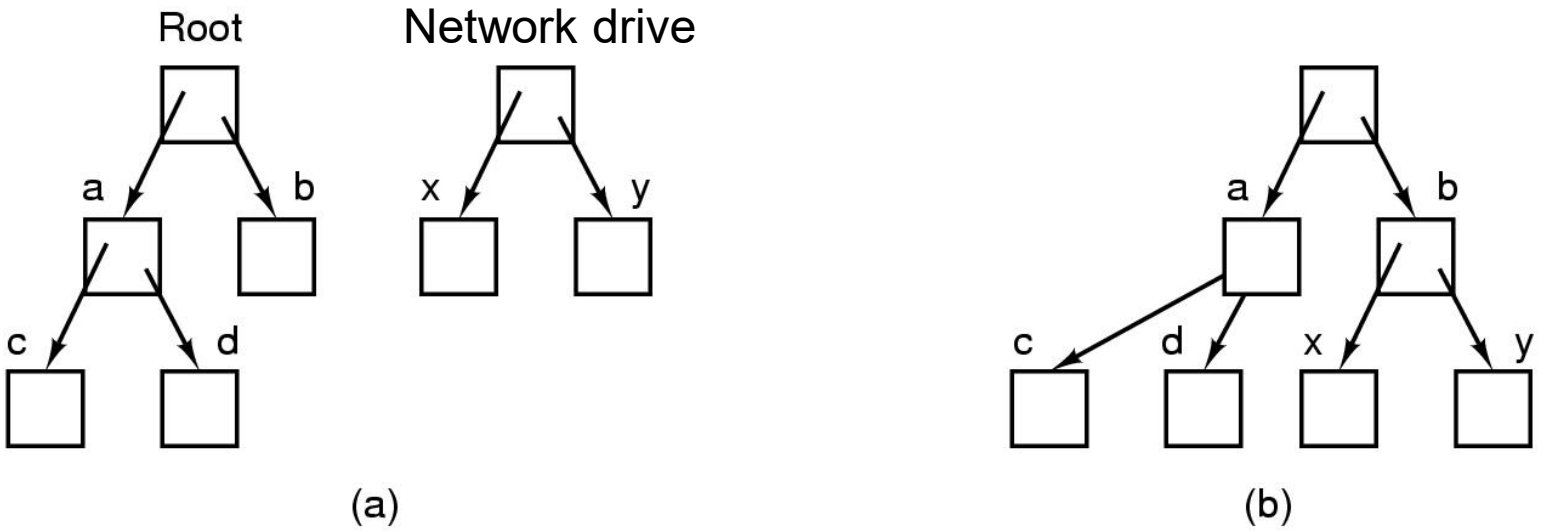
(a) A potential deadlock. (b) an actual deadlock.

Concepts: File systems



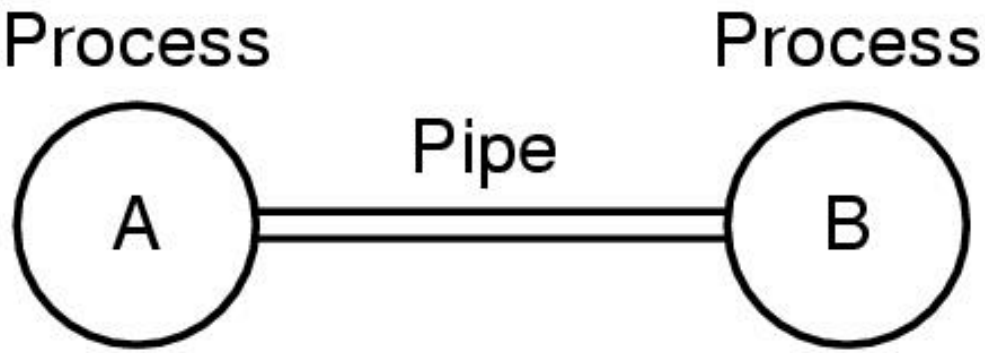
File system for a university department

Concepts: File Systems



- Before mounting,
  - files on network drive, USB drive, etc. are inaccessible
- After mounting network drive on b,
  - files on network drive are part of file hierarchy

Concepts: Interprocess communication

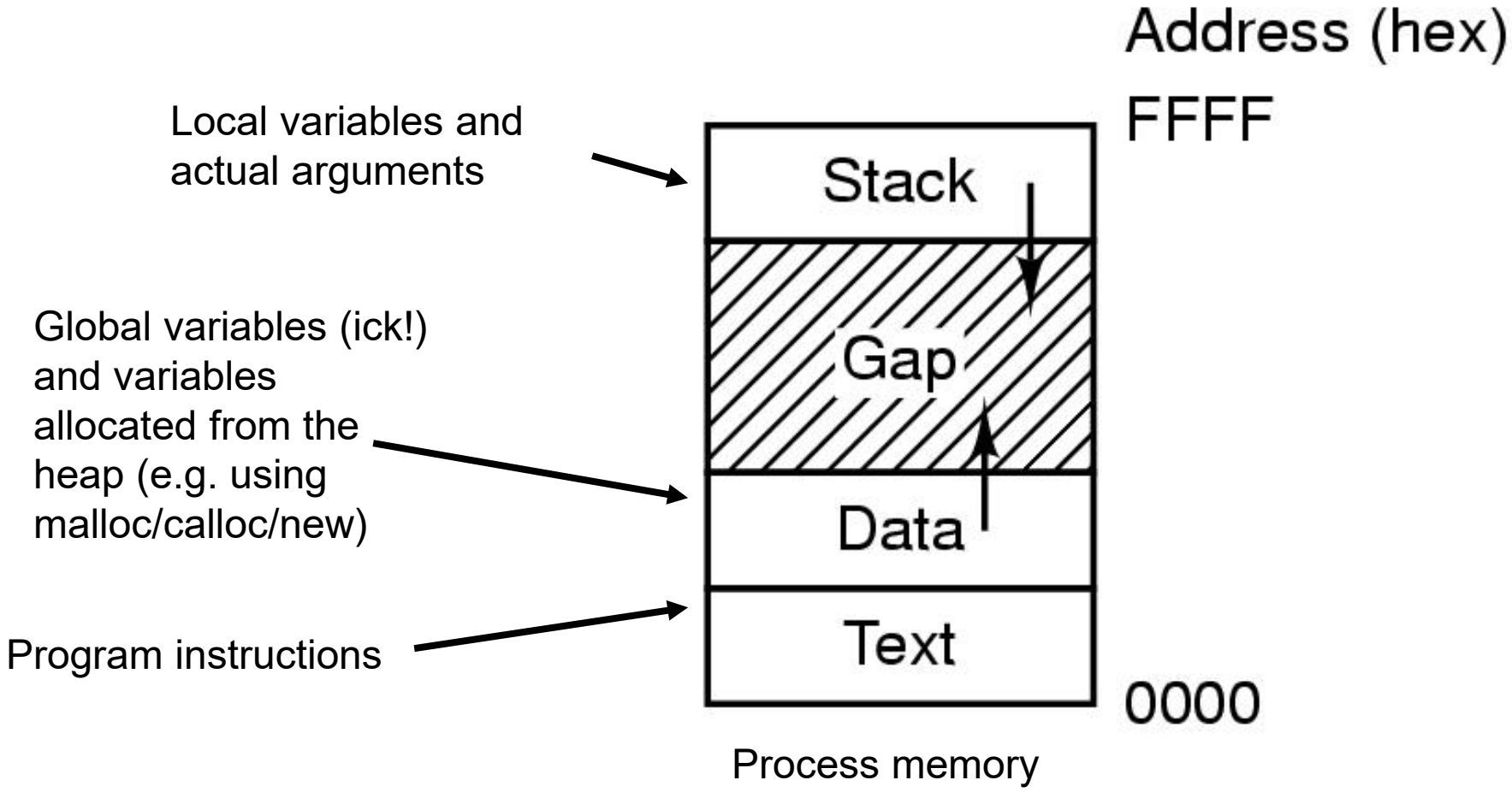


Two processes connected by a pipe

Concepts: Function calls

- Run-time models allow compiled programs to:
  - pass parameters
  - resolve variable scope
  - track subroutine invocation
- We need to understand a small part of this to understand how user programs communicate with the operating system.

Concepts: Function calls

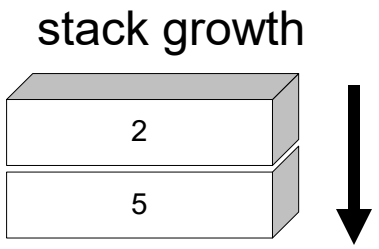


Concepts: Function calls

foobar(alpha, beta)

- let alpha = 5, beta = 2
- Push arguments

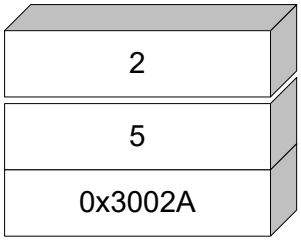
  (typically in reverse order)



- Call function read

  push return address on stack

  execute jump subroutine instruction

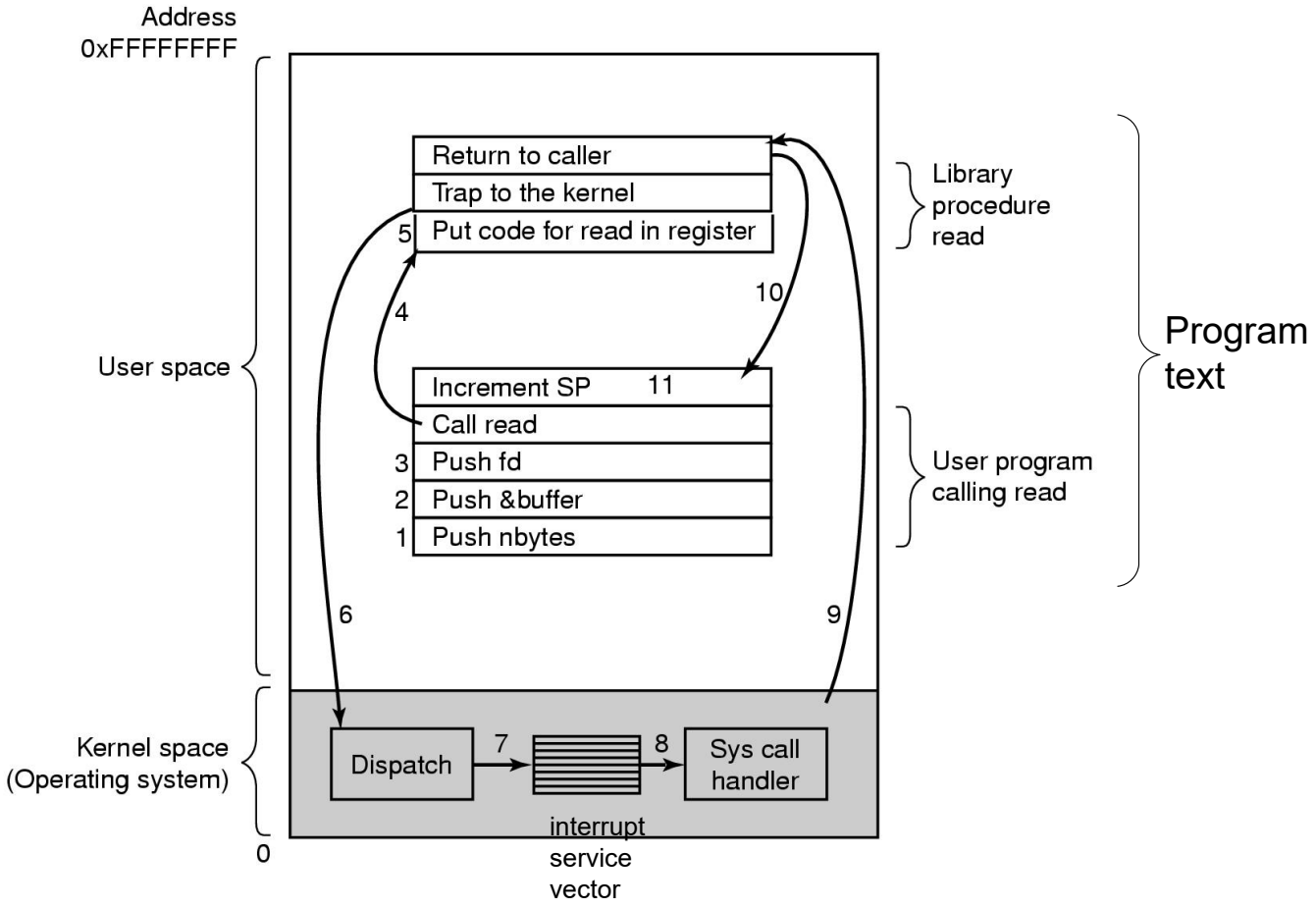


- Pop stack upon return

Concepts: System calls

- Typically starts by calling a library function
- Library function sets a register indicating what system functionality should be performed
- Executes a special instruction called a TRAP which triggers a software interrupt.
- Kernel executes and returns with RTE

Concepts: System Calls



Execution of system call:

read (fd, buffer, nbytes)

Concepts: System Calls

- Permits processes to make requests of the operating system
- Sample process management calls from UNIX

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

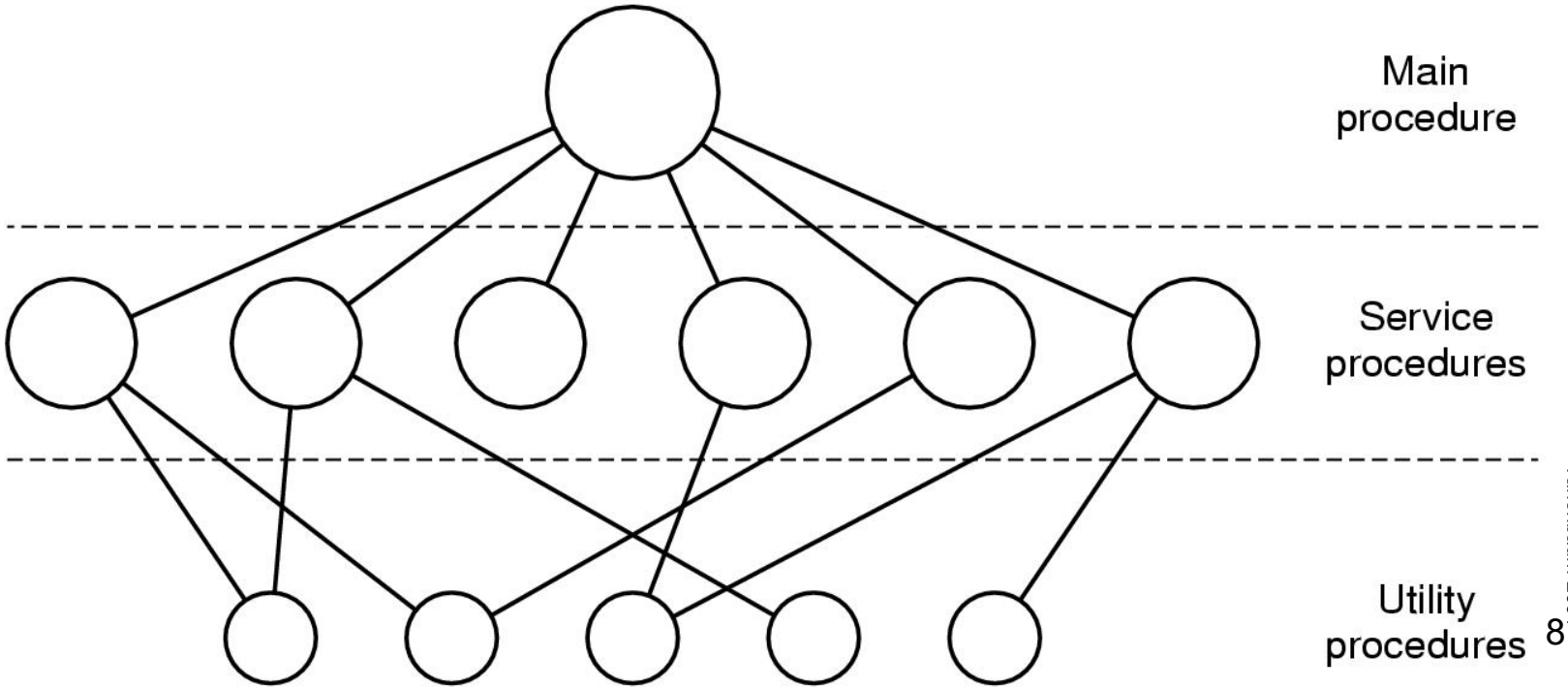
Figure 1-18 Tanenbaum 2001

Concepts

- Memory management
- Security
- Shells permit the user to communicate with the operating system
  - text-oriented command interpreter
  - graphical user interface (GUI)

Operating System Architecture

- Monolithic systems
  - Traditional program design
  - Collection of procedures



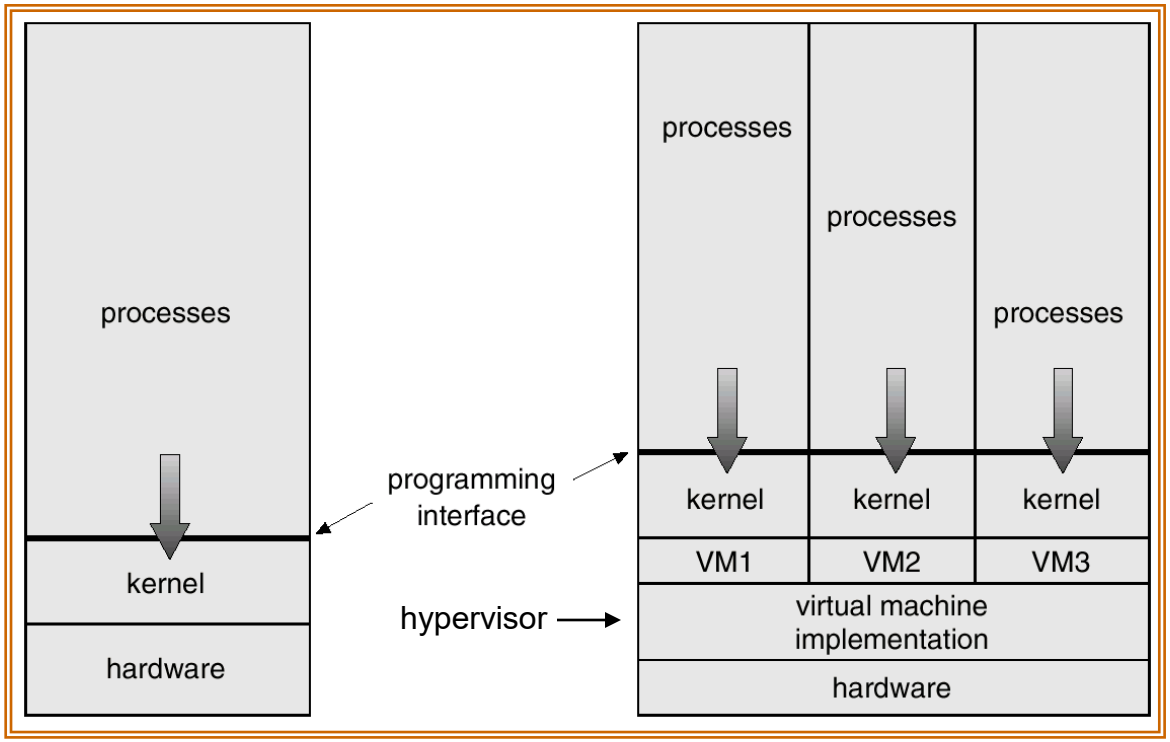
Architecture: Layered Systems

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Structure of the THE operating system

Architecture: Virtual Machines

- Provides *emulation* of multiple *physical machines* called virtual machines.

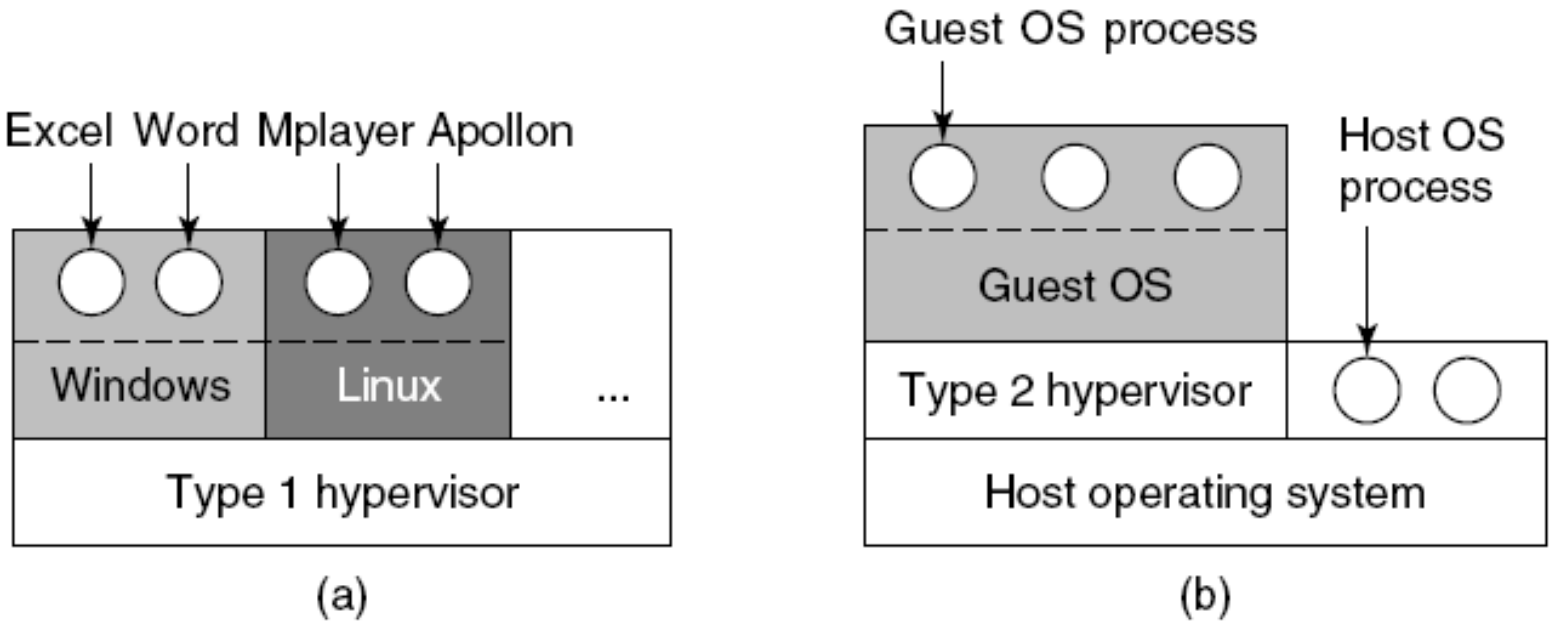


Non-virtual Machine

Virtual Machine

Architecture: Virtual Machines

Figure 1-29. (a) Type 1 hypervisor.
(b) Type 2 hypervisor.



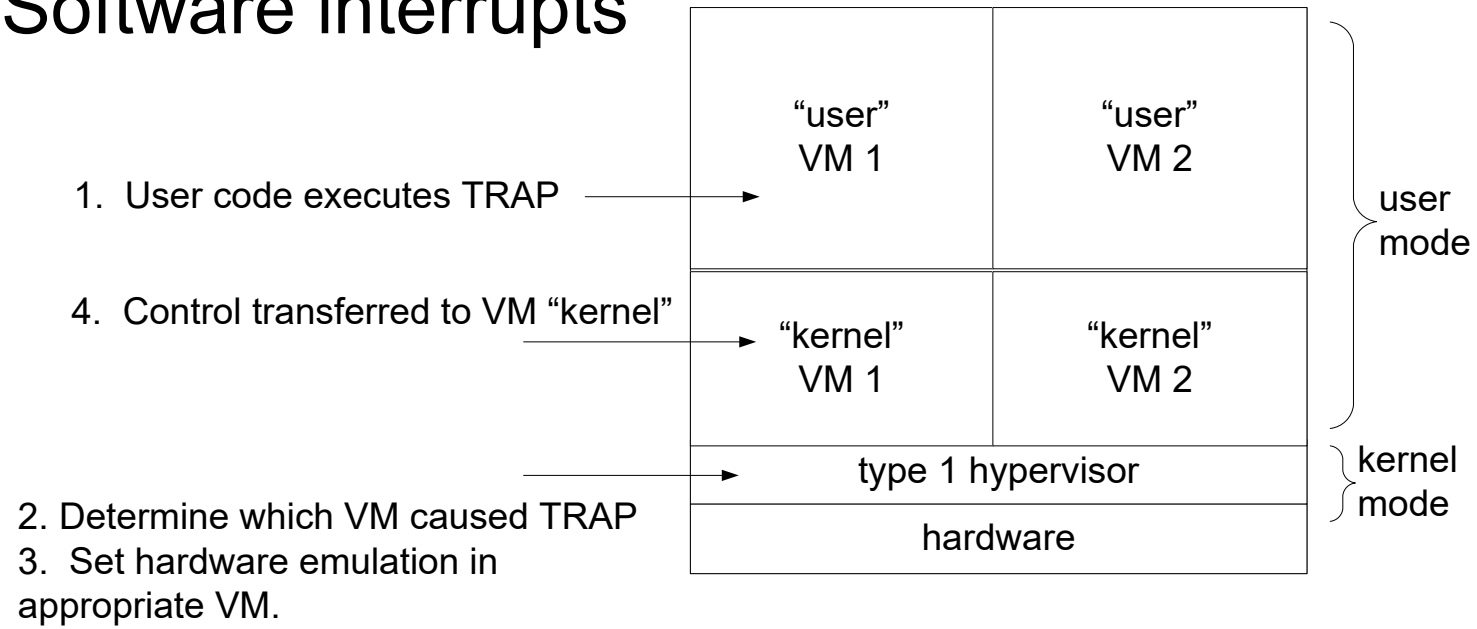
Type 1 hypervisors require hardware support.

Architecture: VM Components

- Hypervisor (monitor)
  - Interfaces with hardware.
  - Manages sharing of physical resources between virtual machines.
  - Updates data structures representing virtual machines.
  - Provides emulation of hardware for virtual machines.

Architecture: Virtual machines

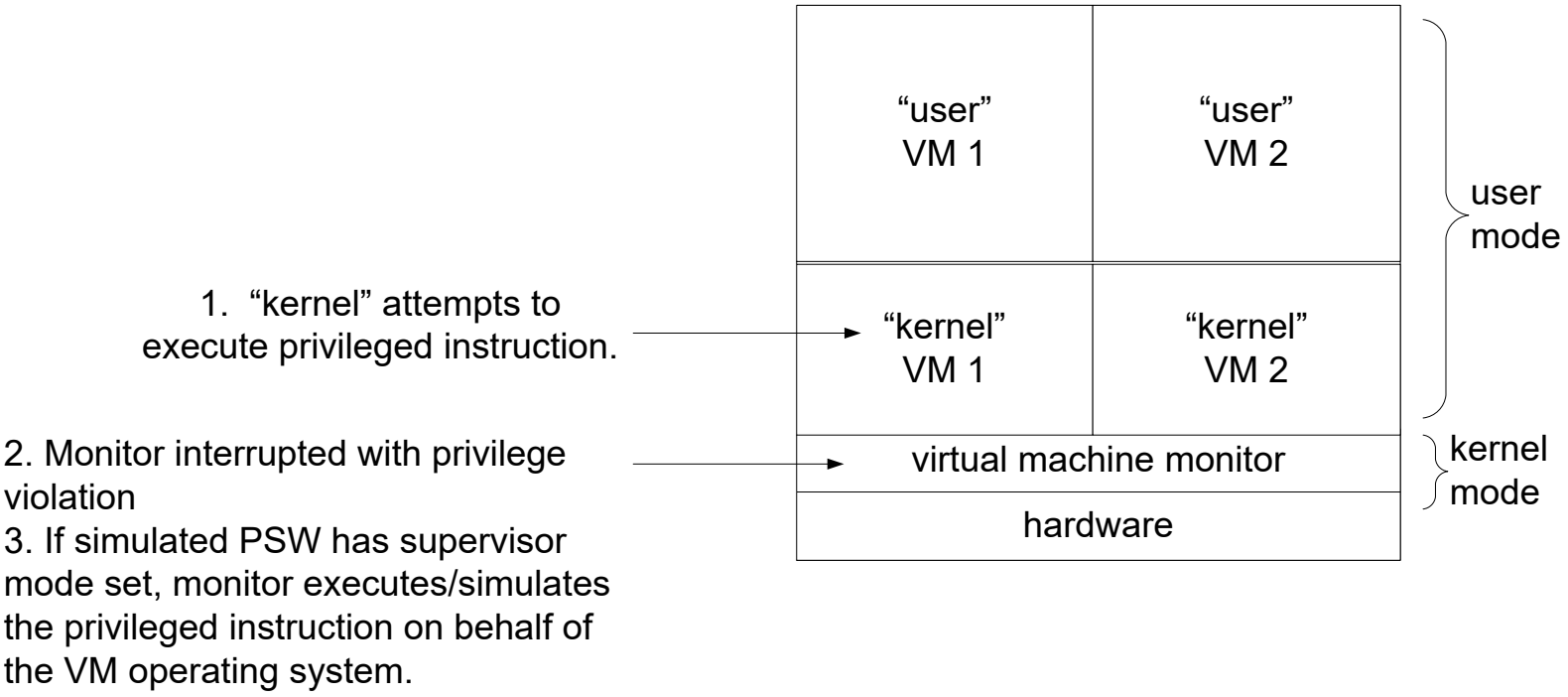
- Software interrupts



- Hardware interrupts similar, except hypervisor handles physical device and sets up virtual device.

Architecture: Virtual machines

- Execution of privileged instructions



Microkernels

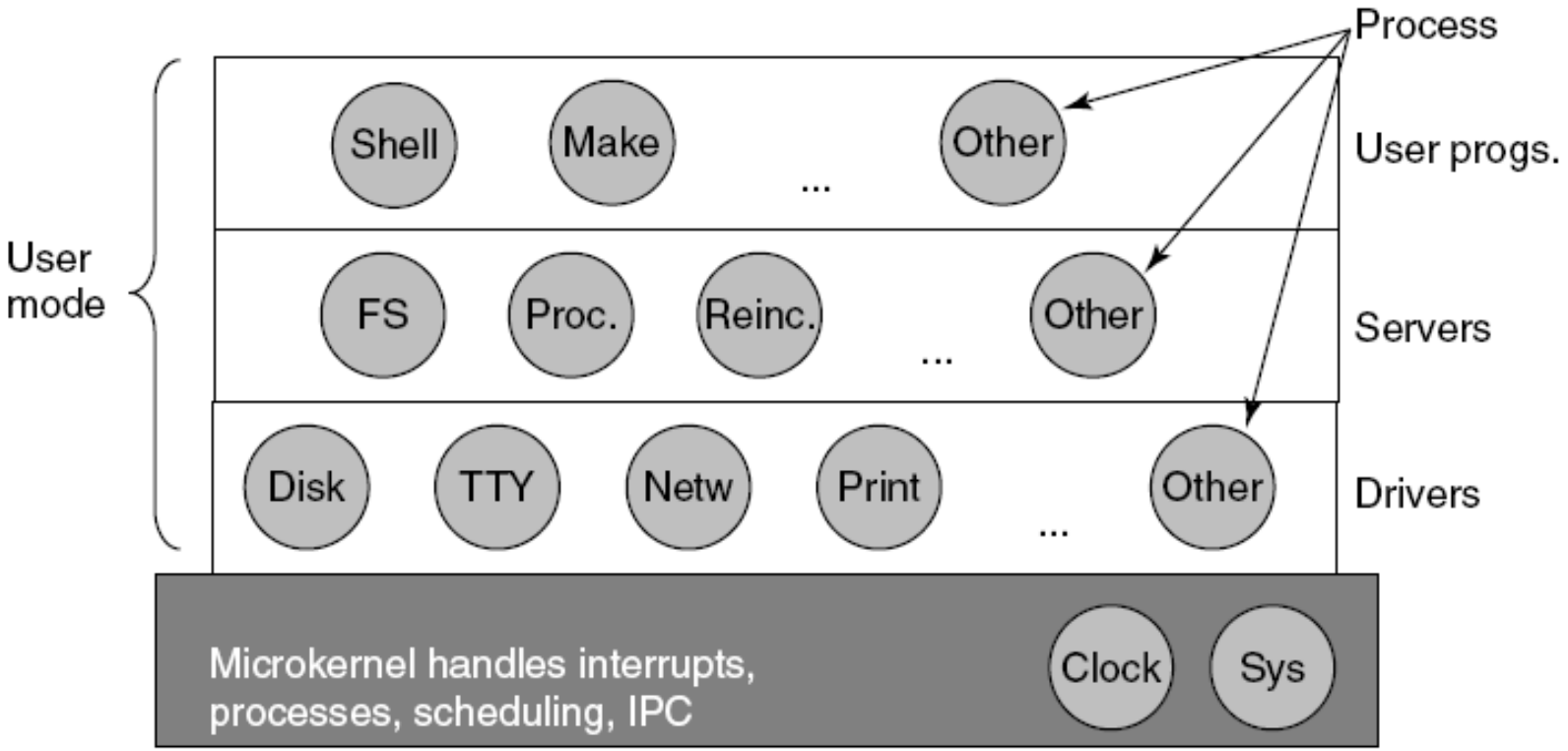


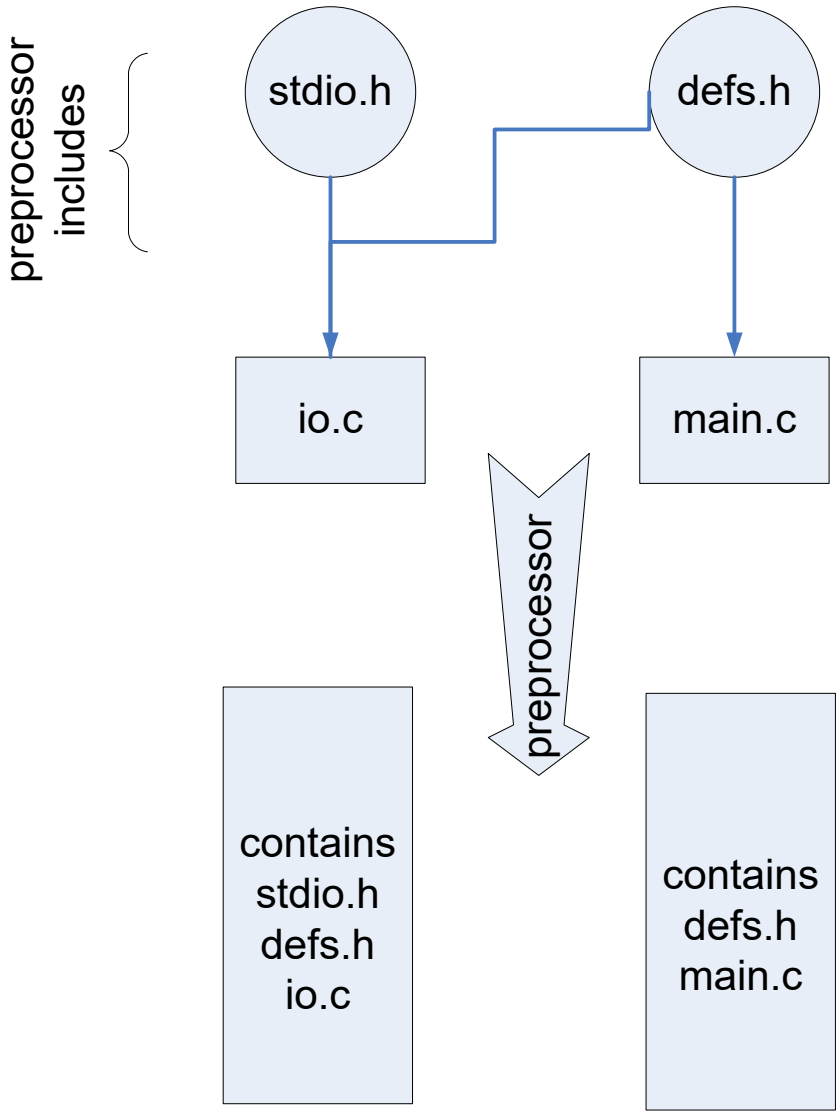
Figure 1-26. Structure of the MINIX 3 system.

Essential services run in user mode and make system calls to a very small kernel. User processes send messages to the services.

Other Architectures

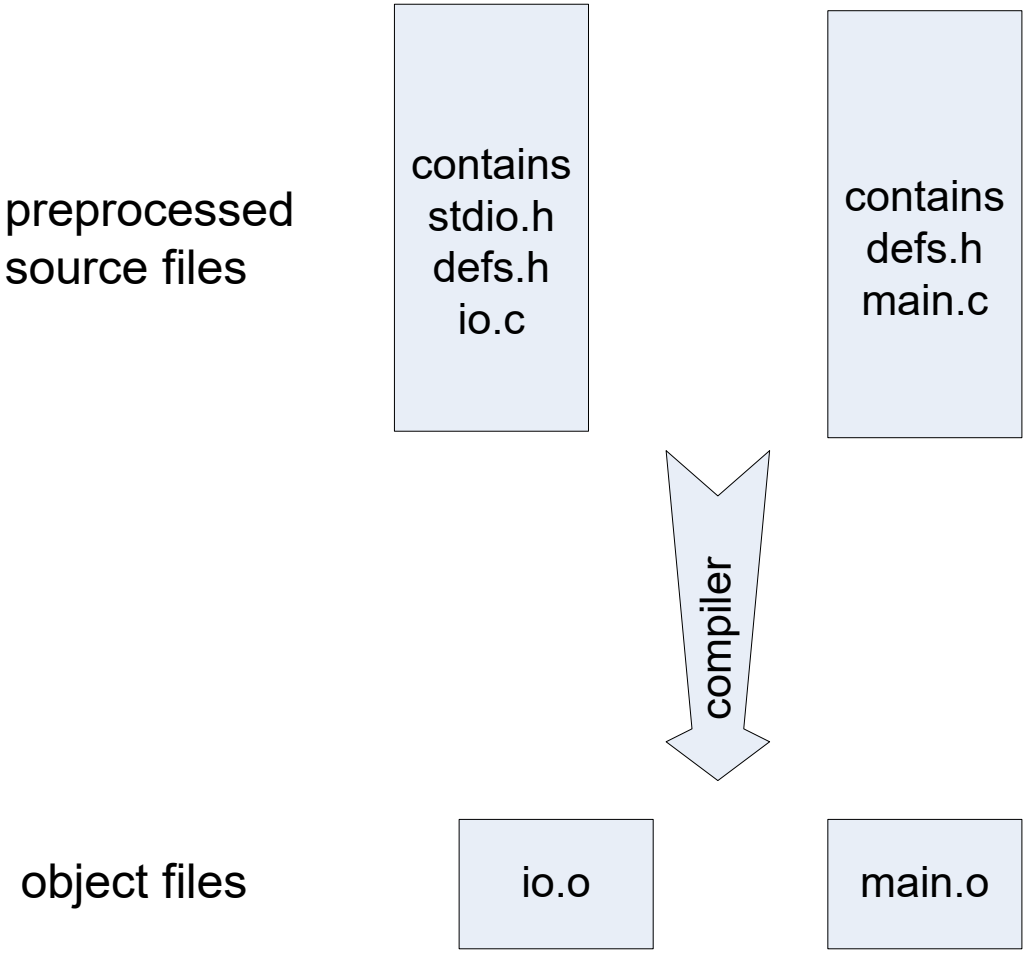
- Exokernels: Similar to virtual machines
  - Each machine allocated a subset of the resources.
  - Saves a mapping layer as exokernels partition resources.
    e.g. this part of the disk belongs to exokernel 3

Review: C/C++ compilation

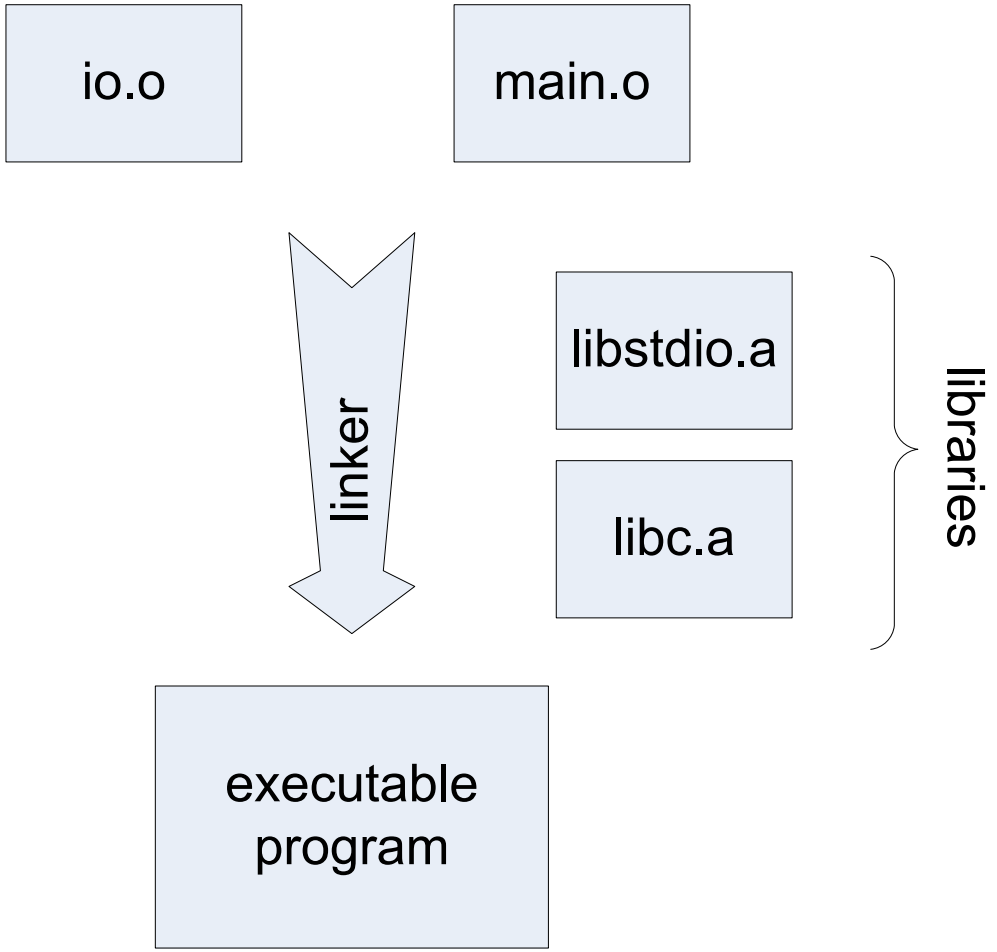


What happens if variables or functions are declared in defs.h?

C/C++ compilation



C/C++ linking



The linker is usually invoked by the compiler transparently