OPERATING SYSTEMS

#2

After A.S.Tanenbaum, Modern Operating Systems, 3rd edition

Uses content with permission from Assoc. Prof. Florin Fortis, PhD

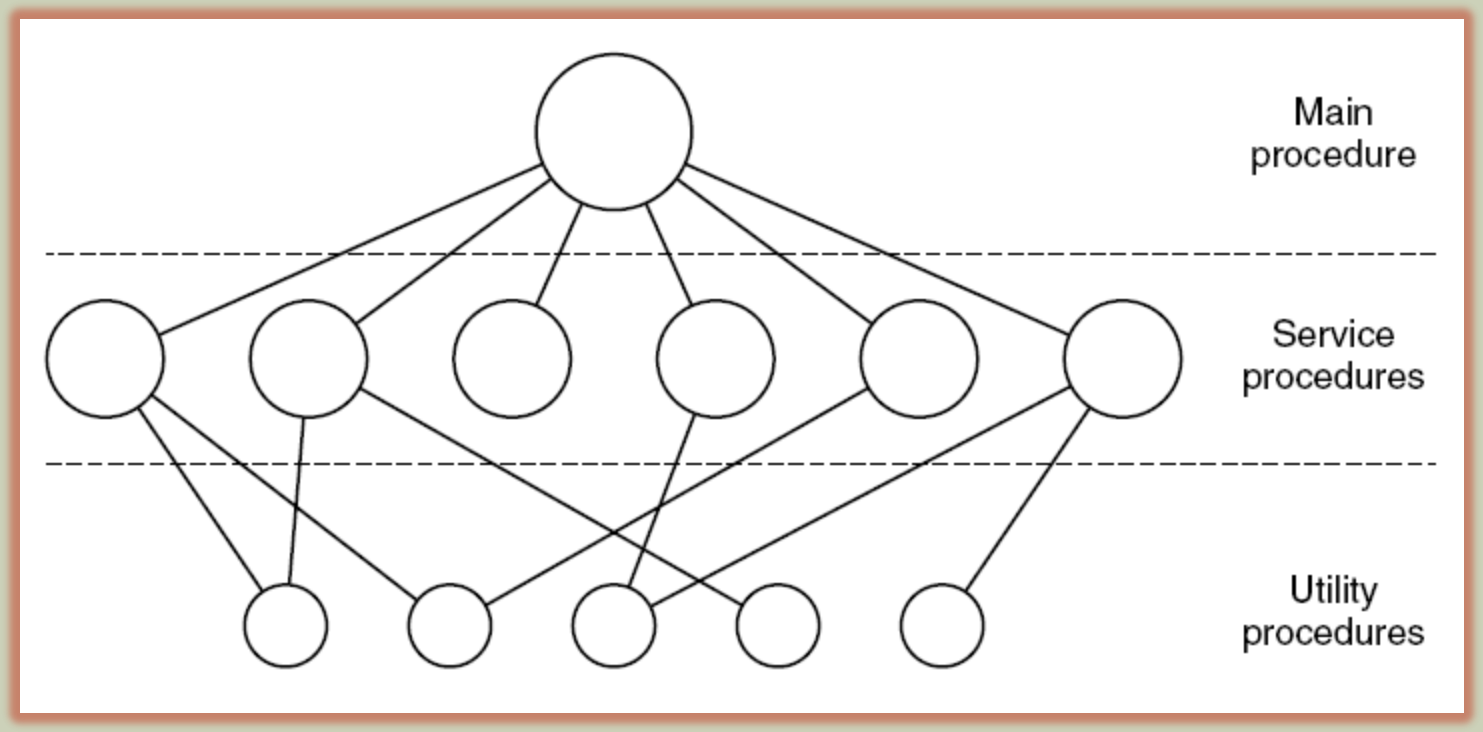
INTRODUCTION

Operating systems structure

OPERATING SYSTEM STRUCTURE MONOLITHIC SYSTEMS

- Monolithic systems – basic structure:
  - A main program that invokes the requested service procedure.
  - A set of service procedures that carry out the system calls.
  - A set of utility procedures that help the service procedures.

OPERATING SYSTEM STRUCTURE
MONOLITHIC SYSTEMS



A simple structuring model for a monolithic system.

OPERATING SYSTEM STRUCTURE MONOLITHIC SYSTEMS. MS-DOS

- Two largely used operating systems have been characterized by a kind of monolithic structure:

1. The MS-DOS operating system;
   - is an OS with a weak structure, without a clear delimitation of interfaces or the levels of functionality.
   - MS-DOS offer a simple approach for the I/O devices: every application can directly access every I/O device.
   - Also, MS-DOS always ignored a hardware issue at processor level: the dual mode of operation; keeping the OS at the level of 8088/80286 processors.

OPERATING SYSTEM STRUCTURE MONOLITHIC SYSTEMS. UNIX

- Early UNIX OS
  - The first UNIX OS are characterized by a (limited) monolithic structure.
  - One can identify two distinct components: the OS kernel and system applications.
  - However, the services offered by these two components are very large, essentially the same, from different points of view.

OPERATING SYSTEM STRUCTURE LAYERED SYSTEMS.

- Layered systems offer a clearer construction and an easier administration of the operating system, with several drawbacks:
  - The definition of the different layers must be clearly made prior to the design of the entire OS;
  - Making system calls from higher layers requires a lot of overhead in order to identify the target layer and the original layer for the system call.

OPERATING SYSTEM STRUCTURE LAYERED SYSTEMS. THE

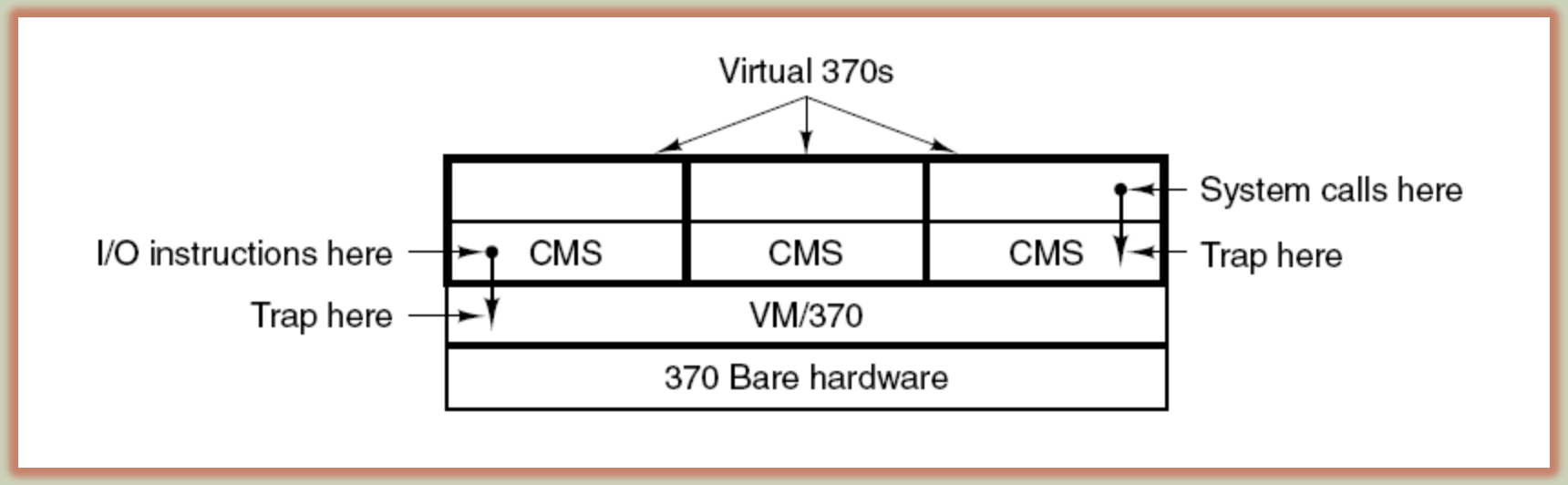
- Structure of the THE operating system

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

OPERATING SYSTEM STRUCTURE VIRTUAL MACHINES

- Virtual machines have been developed starting from early time-sharing machines:
  - A virtual machine should offer an extension of the machine, by a convenient interface;
  - A virtual machine should offer the basic services of multiprogramming.
  - In the same time, the virtual machine should offer a clear separation between these two functions.

OPERATING SYSTEM STRUCTURE VIRTUAL MACHINES. VM/370

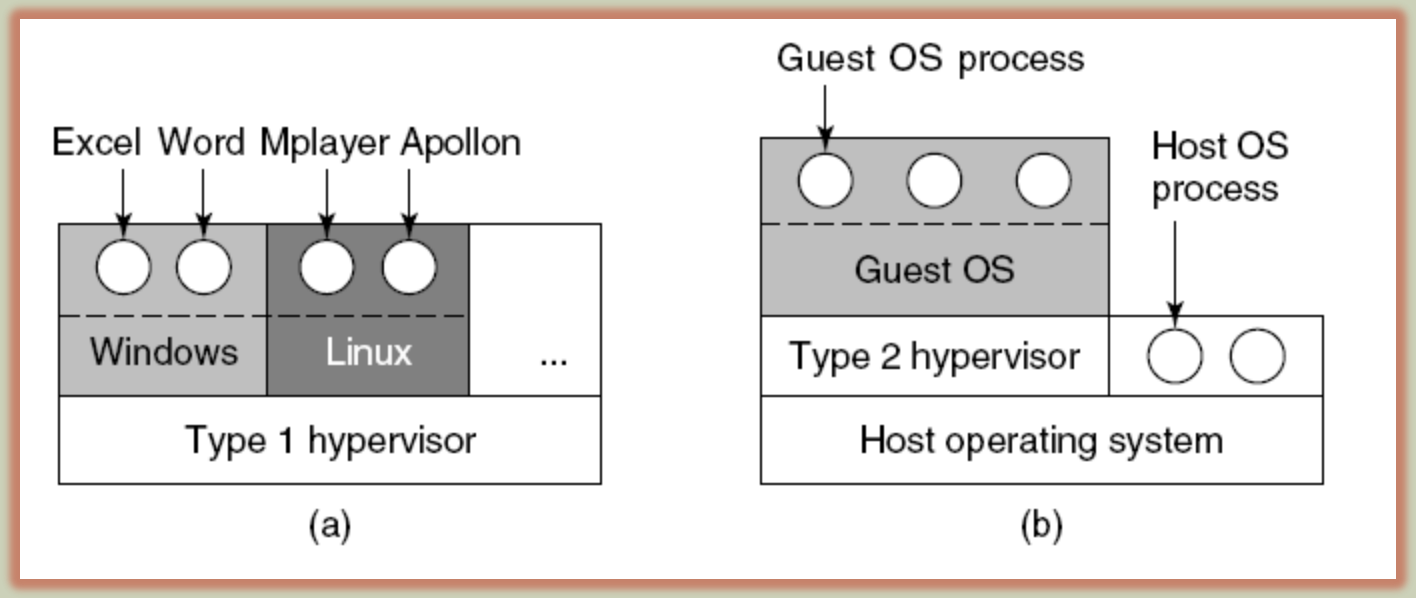


The structure of VM/370 with CMS.

OPERATING SYSTEM STRUCTURE VIRTUAL MACHINES

- By a VM/370 virtual machine, the very basic requirements are met. Such a machine should, however, respond to some distinctive allocation issues:
  - In order to avoid conflicts due to the usage of I/O devices, it should be necessary to develop *virtual devices*, similar with the original devices, managed at a basic level of the OS.
  - Theoretically, virtual machines should run only in the user space. Thus, the virtual machine should run its own virtual user mode and virtual kernel mode. When solving a system call, the virtual machine should switch first to its virtual kernel mode, and then trigger the same switch (this time to the real kernel mode) in the monitor of virtual machines.

OPERATING SYSTEM STRUCTURE VIRTUAL MACHINES. MODERN VIRTUAL MACHINES

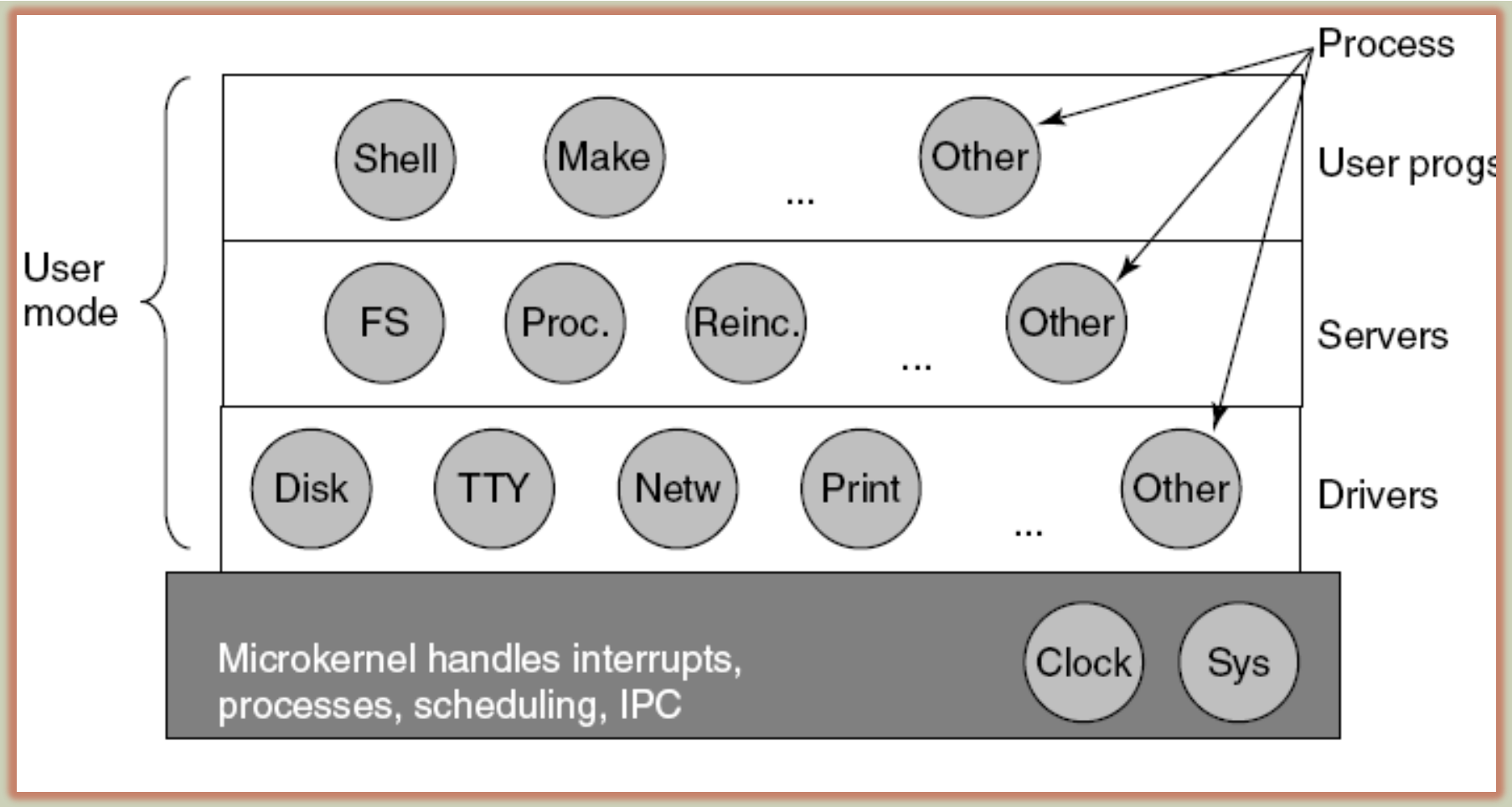


(a) A type 1 hypervisor.
(b) A type 2 hypervisor.

OPERATING SYSTEM STRUCTURE MICROKERNELS

- While virtual machines are offering exact copies for the physical machine, in this situation we have copies of the original systems, but only with a part of the original resources.
- Such a structure is based on a central application (named *exo-kernel/micro-kernel*), running in kernel mode, whose main task is to allocate resources to the virtual machines and to manage the allocated resources.

OPERATING SYSTEM STRUCTURE MICROKERNELS. MINIX 3

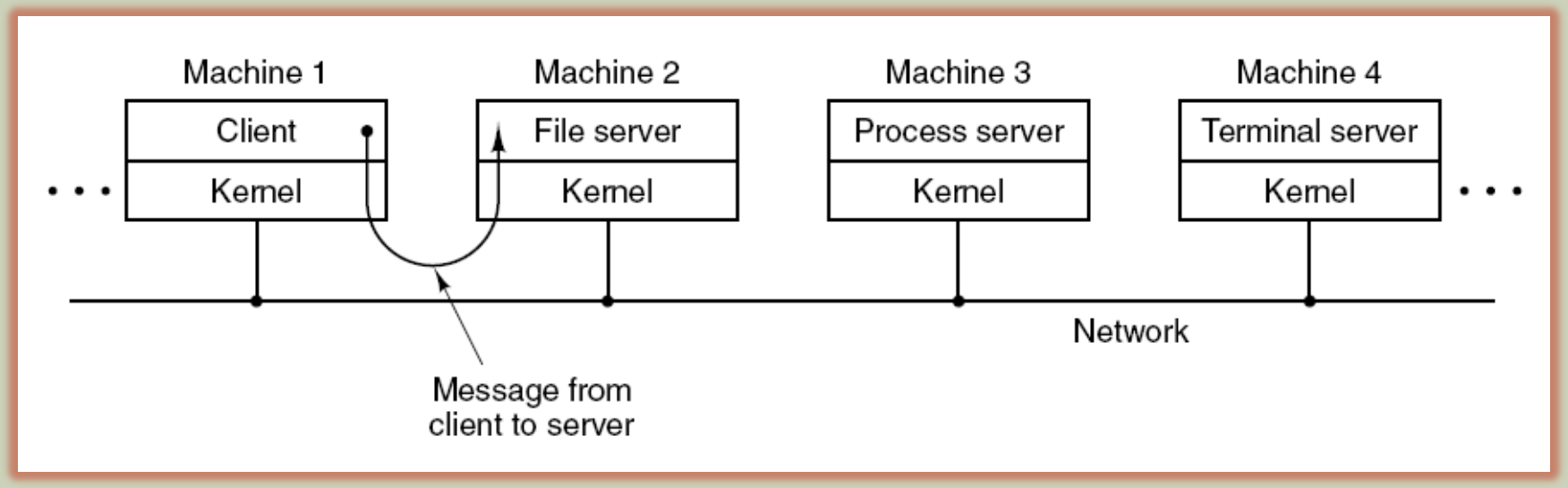


Structure of the MINIX 3 system.

OPERATING SYSTEM STRUCTURE CLIENT-SERVER

- In this model the services of the OS are offered through specialized processes, which are able to handle only certain type of requests.
  - Parts of the operating system are being implemented directly in user processes (client processes).
  - The OS kernel should be more likely oriented through communication control between the client and server processes.
- Theoretically, all the processes run in user space, with the exception of the kernel. Practically, some of the OS functions can be offered only in kernel mode.
- The client-server model can be easily adapted for usage in a distributed environment.

OPERATING SYSTEM STRUCTURE CLIENT-SERVER MODEL



The client-server model over a network.

INTRODUCTION

Protection mechanisms

INTRODUCTION
PROTECTION MECHANISMS

- The very first computing systems were single-user systems, completely controlled by the unique user of the system.
- Once the resident monitor appeared, some simple memory protection mechanisms were required.
- Resource sharing and spooling issued new protection problems, due to the common storage space that can be used.

INTRODUCTION
PROTECTION MECHANISMS

- With multiprogramming and time-sharing, there is the possibility that multiple processes and/or users exist simultaneously in the system, with more protection issues to be solved.

- Typical execution errors are identified at hardware level and are managed under the control of the OS.
  - The affected hardware component issues some specialized instructions (traps) to the OS.
  - The control is then transferred to the OS by using the same mechanism as for interrupts, and by using again the interrupt vector.

INTRODUCTION
PROTECTION MECHANISMS

Dual mode of operation

- Protection mechanisms are required in order to protect programs and data against malfunction of other programs. They are also required for the OS itself.
- A possibility for this type of protection is offered by the dual mode of operation.
  - The *dual mode of operation* was initially implemented at software level, in order to limit the access to certain services accessible only to the OS.
- Later, the dual mode of operation was offered at hardware level, via a specialized bit (the mode bit), controlled only by the OS.

INTRODUCTION
THE DUAL MODE OF OPERATION

- During the boot operation, the computing system is running into monitor mode. The OS itself is doing its job in the monitor mode. All user processes are using only the user mode.
- Each interrupt or trap is accompanied by an immediate switch to the monitor mode, after the OS obtained the control of the system.
  - The OS needs this mode in order to solve the tasks related with the interrupt/trap instruction. Observe that the OS always runs in monitor mode.
- The OS should switch back to user mode once the control is returned to user process.

INTRODUCTION
THE DUAL MODE OF OPERATION

- The dual mode of operation is used as a protection mechanism by specifying a set of processor instructions that can be executed only in this mode of operation (privileged instructions). These instructions are meant only to the OS.
- The access to privileged instructions is made at hardware level, each attempt to use such an instruction being considered as an execution of an illegal instruction. Also, a trap instruction is issued to the OS in order to signal the situation.

INTRODUCTION
THE DUAL MODE OF OPERATION

- If an OS cannot offer a dual mode of operation, user processes could be able to access or even modify areas that are meant only to the OS.

- This kind of behavior can affect the functionality of the OS, or it could be possible that processes take the control of the OS.

INTRODUCTION
PROTECTION MECHANISMS

I/O protection

- The computing system define I/O operations as being privileged instructions, accessible only in monitor mode.
- Because interrupts are the preferred way to solve I/O operations, the interrupt mechanism could offer the possibility to gain the control over the system by a user process.
- An interrupt is solved by using the interrupt vector. If a process is permitted to alter an entry from this vector, then it could make the OS to pass the control over the system without switching back to the user mode.
- This problem is solved by making all the operations that affect the interrupt vector as being privileged instructions.

INTRODUCTION
PROTECTION MECHANISMS

Memory protection

- Interrupt routines also run in monitor mode. These offer another easy-to-break point in the system.
  - The same solution can be used in order to avoid disasters: make them protected!
- This kind of protection for both interrupt vector and interrupt routine is effective only if there is a good protection mechanism for memory, too.
  - This is because all interrupt information are kept in the memory by the OS.
- In order to have an effective memory protection mechanism, we need a hardware mechanism eventually based on the base and limit registers.

INTRODUCTION
MEMORY PROTECTION

- The protection is realized at processor level, comparing all the generated addresses with the values from registers.
- Each attempt to access kernel (monitor) memory or other user's memory should be immediately sanctioned by issuing a (fatal) trap instruction, treated as a fatal error.
- The base and limit registers must be protected by offering instructions to alter the values stored here only in monitor mode.
- The OS should be the only application able to alter these registers and access information from the entire memory space. This requirement is needed in order to enforce some memory management tasks.

INTRODUCTION
PROTECTION MECHANISMS

Processor protection

- Processor protection requires a different approach. This is because the OS, during its execution, offer periodically the processor to (user) processes.
- The solution, inspired by time-sharing, is to use a timer. This is set-up for a specified amount of time, fixed or variable.
- When using a variable amount of time, there is a need for another timer with fixed amount of time and a counter, that is decreased at every tick. An interrupt is issued only when the counter reaches the 0 value.
- The functioning of the counter and of the timer should be protected by private instructions.

INTRODUCTION
PROCESSOR PROTECTION

- The OS must assure that the timer is being set-up every time the control is offered to another process.
  - The treatment for the timer interrupt depends on the policy of the OS: it could be a fatal error, a simple context switch or a request for supplemental time, if possible.
- Timers can be used as a simple mechanism for specifying the total time of execution, or the duration of resource usage.
- Another frequent usage of timers is in the case of time-sharing systems, where they are used to signal the end of a slice of time.
- All timer operations should be implemented as private operations.

INTRODUCTION

Types of operating systems

TYPES OF OPERATING SYSTEMS

- Mainframe operating systems
- Server operating systems
- Multiprocessor operating systems
- Personal computer operating systems
- Handheld operating systems
- Embedded operating systems
- Sensor node operating systems
- Real-time operating systems
- Smart card operating systems

INTRODUCTION

Operating systems concepts

OPERATING SYSTEMS CONCEPTS

1. Processes
2. Address spaces
3. Files
4. Input/Output
5. Protection
6. The shell
7. Ontogeny recapitulates phylogeny
   - Large memories
   - Protection hardware
   - Disks
   - Virtual memory

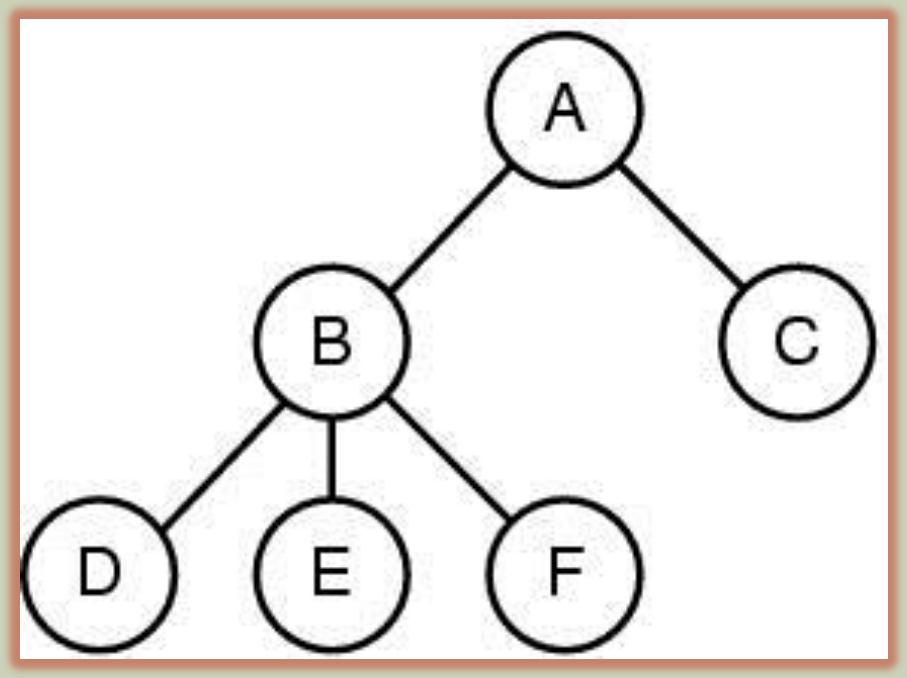
OPERATING SYSTEMS CONCEPTS SYSTEM CALLS

- Offer the interface between processes and the operating system.
- Are usually offered as instruction in an assembly language, or as instructions in a higher level language.
  - Even in the first case, system calls could be realized from higher level languages.
- The entire function of the operating system is based on system calls: usually processes ask for services from the Operating System through the system calls interface.

OPERATING SYSTEMS CONCEPTS PROCESSES

- A ***process*** is a running program. For every process there is an associated address space.
- Process execution is done in a sequential manner, instruction after instruction: the processor (single core) is able to execute only one instruction at a time.
- Modern operating systems are using a ***table of processes*** in order to store all the necessary information about processes.

OPERATING SYSTEMS CONCEPTS PROCESSES



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

OPERATING SYSTEMS CONCEPTS PROCESSES

- Process management is realized through specific system calls, the main program using these calls being the command line interpretor (the shell).
- For a proper management and security of processes, the operating system is using different information (e.g. various process identifiers) that exist during the entire lifetime of a process.

OPERATING SYSTEMS CONCEPTS PROCESSES

- **Tasks of the operating system**
  - Creation and “deletion” of processes;
  - Suspension and continuation of processes;
  - Process synchronization through specific mechanisms;
  - Inter process communication through specific mechanisms;
  - Solving deadlocking situations through specific mechanisms.

OPERATING SYSTEMS CONCEPTS PROCESSES. SYSTEM CALLS

- **Typical system calls for process management**
  - UNIX processes are based on the fork()...exec() mechanism.
  - For Win32 one can use the CreateProcess() call.
  - Process termination is made by calling exit() – UNIX or ExitProcess() – WIN32.
  - Waiting for process termination is accomplished by calling wait() – UNIX, WaitForSingleObject() – WIN32.
  - The kill() – UNIX system call offer both a simple communication mechanism and a way to forcibly terminate a process.

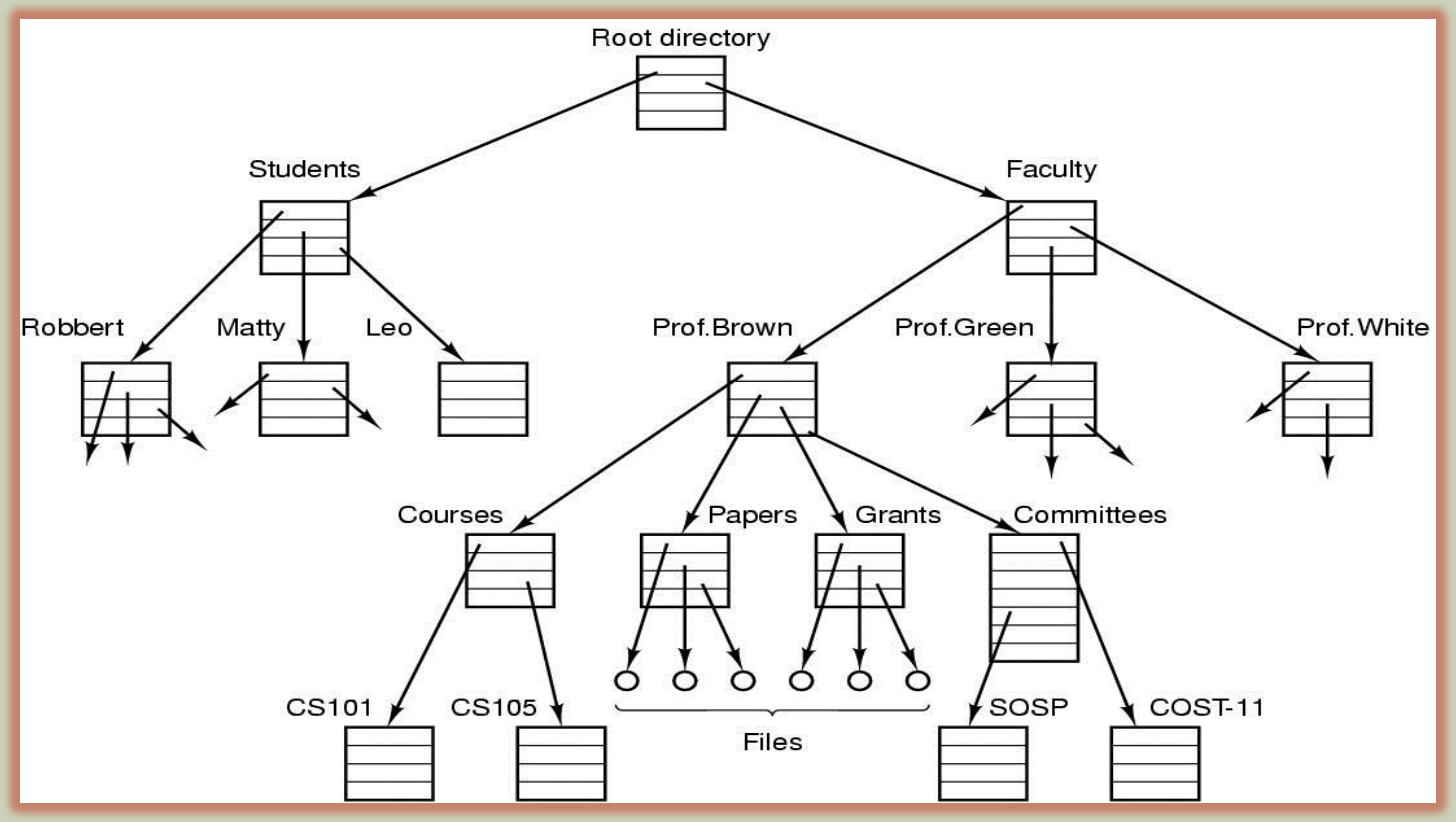
OPERATING SYSTEMS CONCEPTS
THE FILE SYSTEM

- The file system is supported by most of modern OS. By using a file system, an OS hides the peculiarities of disks or other secondary storage devices behind a uniform interface.
- The file is the logical unit for storing information in a computing system.
- A **file** is, in its simplest definition, just a sequence of bytes.
- Directories (folders) have been developed in order to provide a simple way for file grouping and organization.

OPERATING SYSTEMS CONCEPTS
THE FILE SYSTEM

- Regularly, there is a tree-like organization of the files in a file system.
- File accessing rights and file protection are well implemented in most of the file systems.
- When using a tree-like organization, the files are identified by using a path name. Notions like current directory (.), parent directory (..), root directory (/), absolute or relative path name are common when talking about a tree-like organization.

OPERATING SYSTEMS CONCEPTS
THE FILE SYSTEM

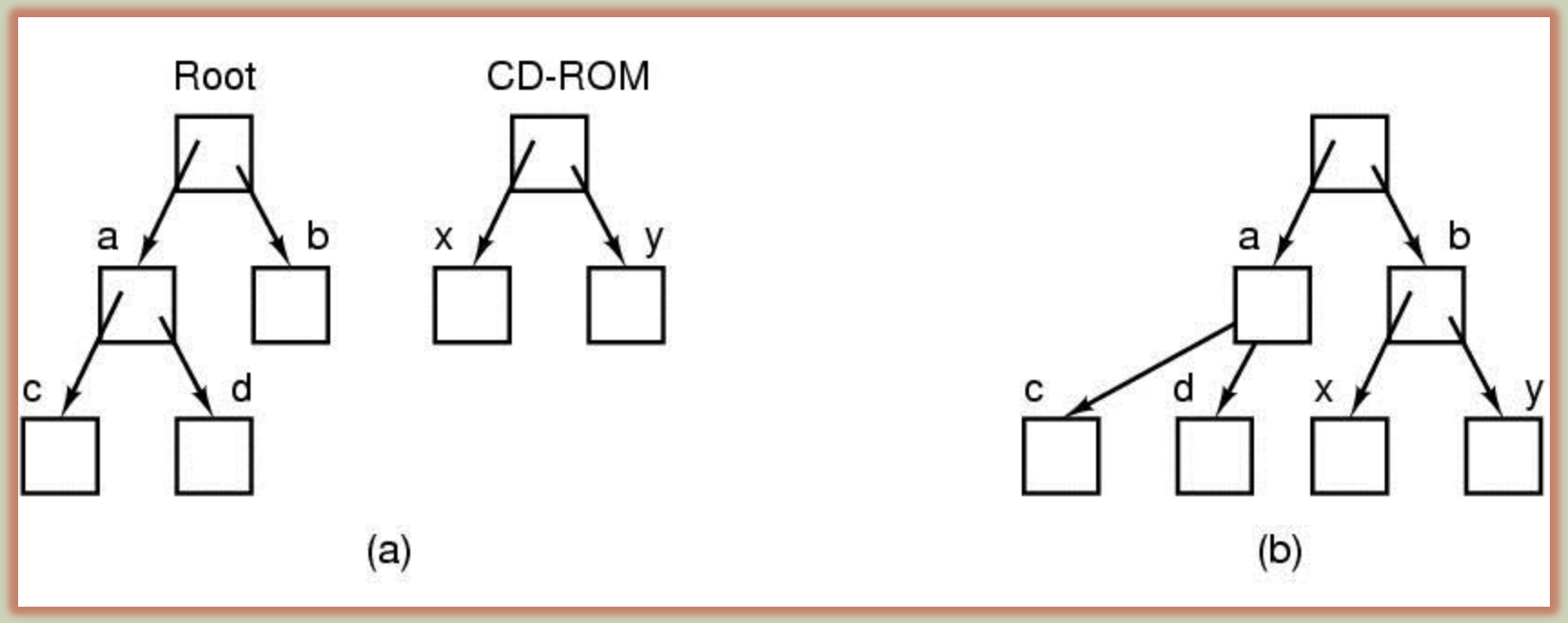


A file system for a university department.

OPERATING SYSTEMS CONCEPTS
THE FILE SYSTEM

- **Tasks of the Operating System**
  - File creation and removal;
  - Folder creation and removal;
  - Primitives needed in order to handle files/folders;
  - Mapping files over secondary storage devices;
  - Mounting file systems.

OPERATING SYSTEMS CONCEPTS
THE FILE SYSTEM



(a) Before mounting, the files on the CD-ROM are not accessible.

(b) After mounting, they are part of the file hierarchy.

OPERATING SYSTEMS CONCEPTS
THE FILE SYSTEM. SYSTEM CALLS (1)

- **File management (system calls)**
  - The open() – UNIX, CreateFile() – Win32 system calls, used in order to open a file.
  - File creation is offered by open() or creat() – UNIX, CreateFile() – Win32 system calls.
  - One can close a file by using close() or CloseHandle() system calls.
  - Read/write file data, by read()/write() or ReadFile()/WriteFile().
  - File pointer position can be controlled by lseek() or SetFilePointer() calls.
  - Information about files can be obtained by using the stat() family or GetFileAttributesEx() calls.

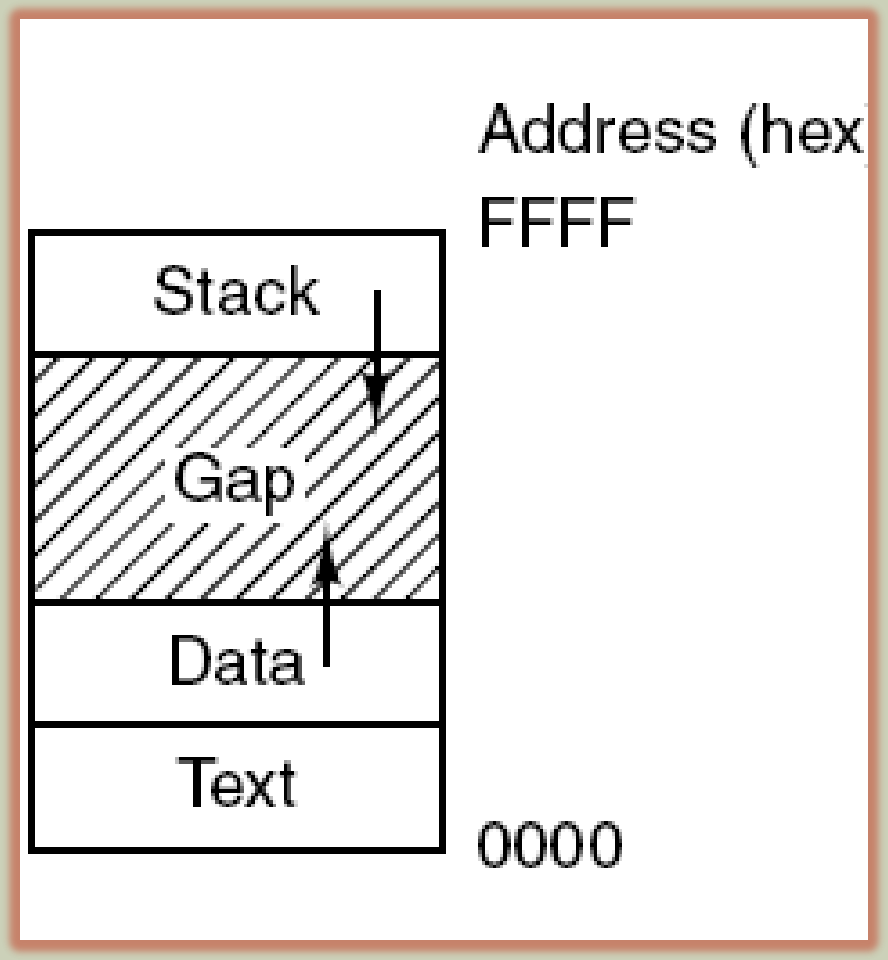
OPERATING SYSTEMS CONCEPTS
THE FILE SYSTEM. SYSTEM CALLS (2)

- **Folder management (system calls)**
  - Even if most of the operating systems offer the same basic treatment for files and folders, there are specialized system calls/library functions for folder management:
  - Folder creation and removal, by using mkdir()/rmdir(), or CreateDirectory()/RemoveDirectory() calls.
  - Change current (working) directory (folder), by using chdir() or SetCurrentDirectory() calls.
  - Removing a folder entry, by using unlink() or DeleteFile() calls.

OPERATING SYSTEMS CONCEPTS
THE MEMORY

- The main memory is the main repository for programs under execution.
- The memory offers its own protection mechanisms, used in order to avoid accidental interaction of stored programs
- Memory management depends on the type of devices that are offering the necessary storage support.

OPERATING SYSTEMS CONCEPTS
THE MEMORY



- Processes have three segments: text, data, and stack.

OPERATING SYSTEMS CONCEPTS
THE MEMORY. MAIN MEMORY

- Programs in execution are always stored in the **main memory** of the system.
- **Tasks of the operating system (main memory)**
  - To follow memory usage;
  - To make decisions about processes to be store in memory;
  - Memory allocation and de-allocation.

OPERATING SYSTEMS CONCEPTS
THE MEMORY. SECONDARY STORAGE

- Other application, or data for programs under execution are stored (permanently) on secondary storage devices.
- Due to their frequent usage, an efficient management is an important task for the OS.
- **Tasks of the operating system**
  - Management of free space;
  - Space allocation, on request;
  - Secondary storage devices scheduling.

OPERATING SYSTEMS CONCEPTS
THE SHELL

- The command line interpreter (the shell) is one of the most important applications in an OS. It offers a basic interface to the operating system.
- Several operating systems integrate the command line interpreter in the kernel.
- The tasks and functioning of the shell are “inspired” from the tasks of the ancient resident monitory.

OPERATING SYSTEMS CONCEPTS
THE SHELL. SKELETON OF A SHELL

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
        type_prompt( );                         /* display prompt on the screen */
        read_command(command, parameters);      /* read input from terminal */

        if (fork( ) != 0) {                     /* fork off child process */
             /* Parent code. */
             waitpid(–1, &status, 0);           /* wait for child to exit */
        } else {
             /* Child code. */
             execve(command, parameters, 0);    /* execute command */
        }
}
```

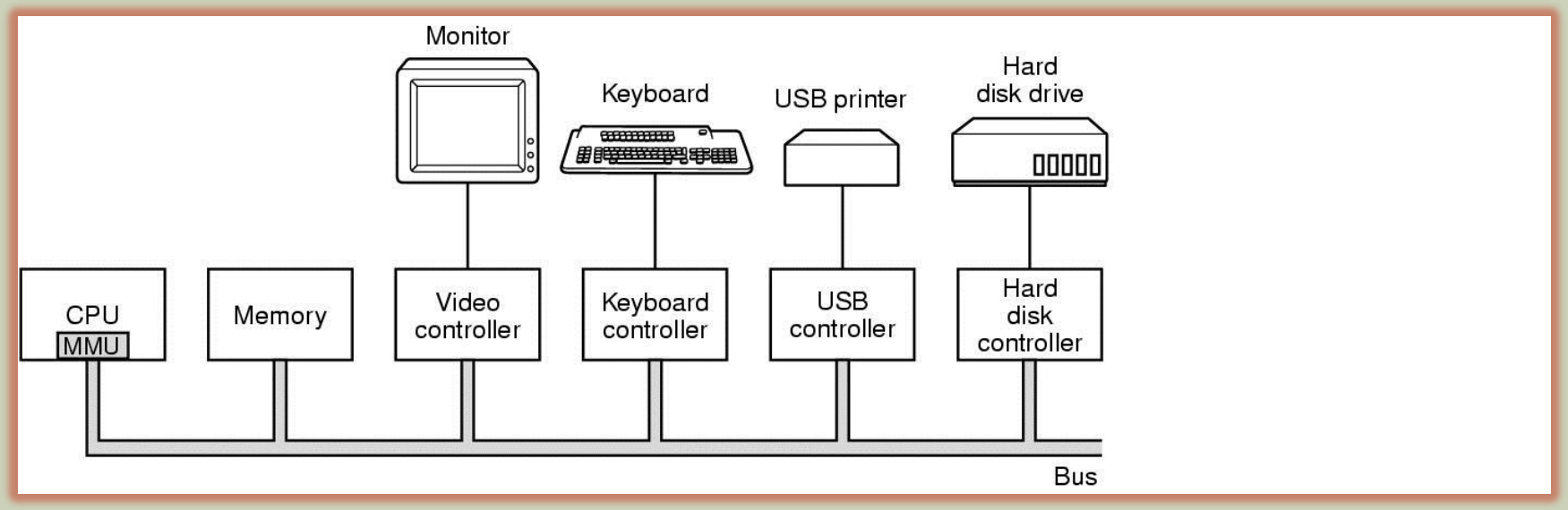
NEWS IN OS!

- History http://en.wikipedia.org/wiki/History_of_operating_systems
- Haiku Open Source http://video.google.com/videoplay?docid=2363314480765878 79
- Gernot Heiser on microkernels http://www.builderau.com.au/video/soa/Convergence-of-kernel-philosophies/0,2000064338,22447293p,00.htm
- Linus Torvalds – Green http://www.builderau.com.au/video/soa/Linux-is-ready-to-go-green-Linus-Torvalds/0,2000064338,22440588p,00.htm
- Google Operating System: gOS Cloud OS http://blog.wired.com/gadgets/2008/12/video-gos-cloud.html
- Contiki Small http://www.sics.se/contiki/news/uipv6-snapshot-release.html
- gSpeak Data Intensive http://oblong.com/

INTRODUCTION

Computer hardware review

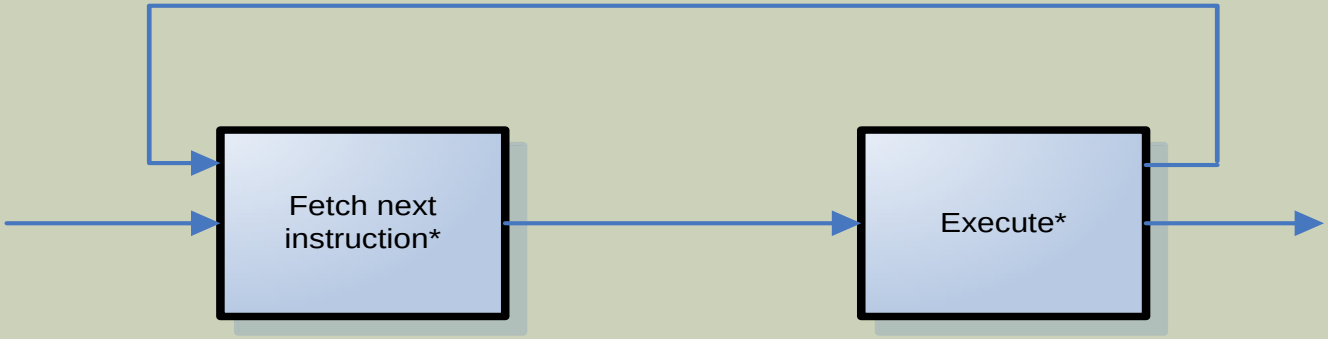
COMPUTER HARDWARE REVIEW



- Some of the components of a personal computer

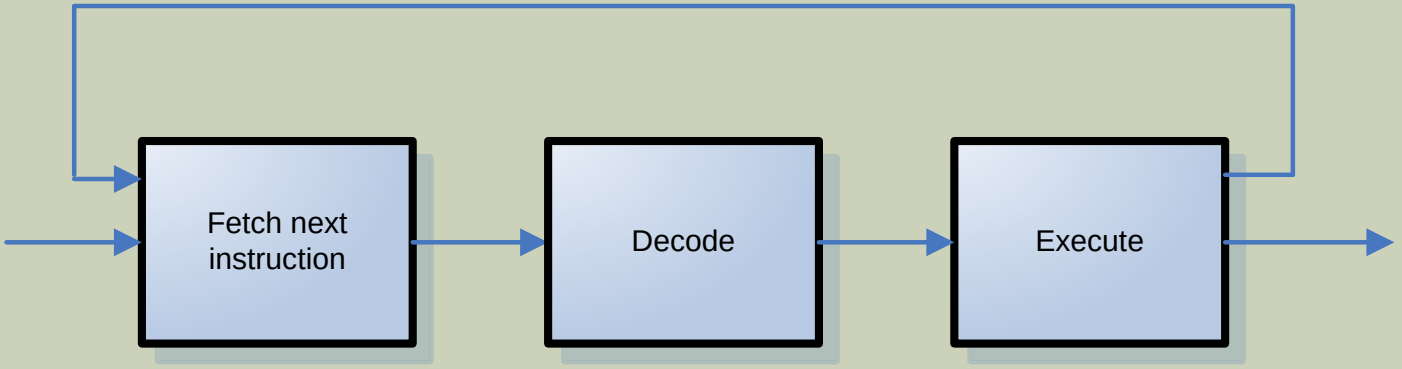
SIMPLE MODELS OF EXECUTION

A) Classical model of execution



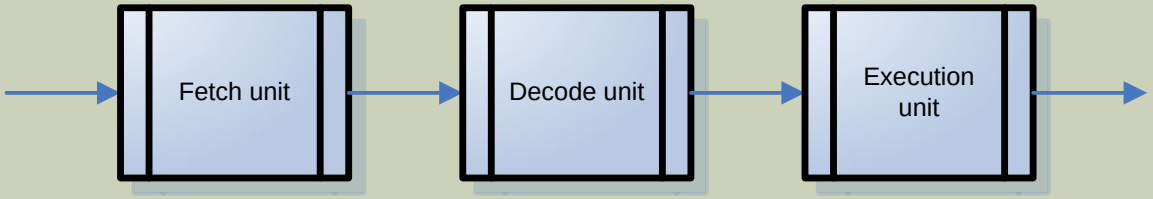
*Observe that decoding is hidden inside Fetch and Execute steps

B) Simple model of execution (three stages)

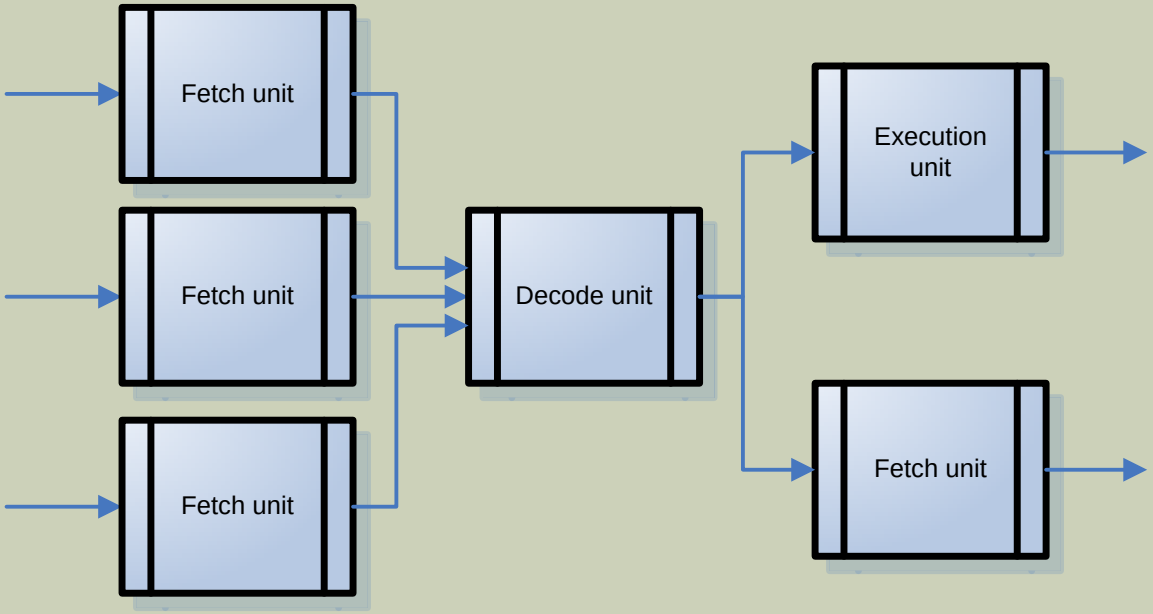


THREE STAGES OF EXECUTION (PIPELINE)

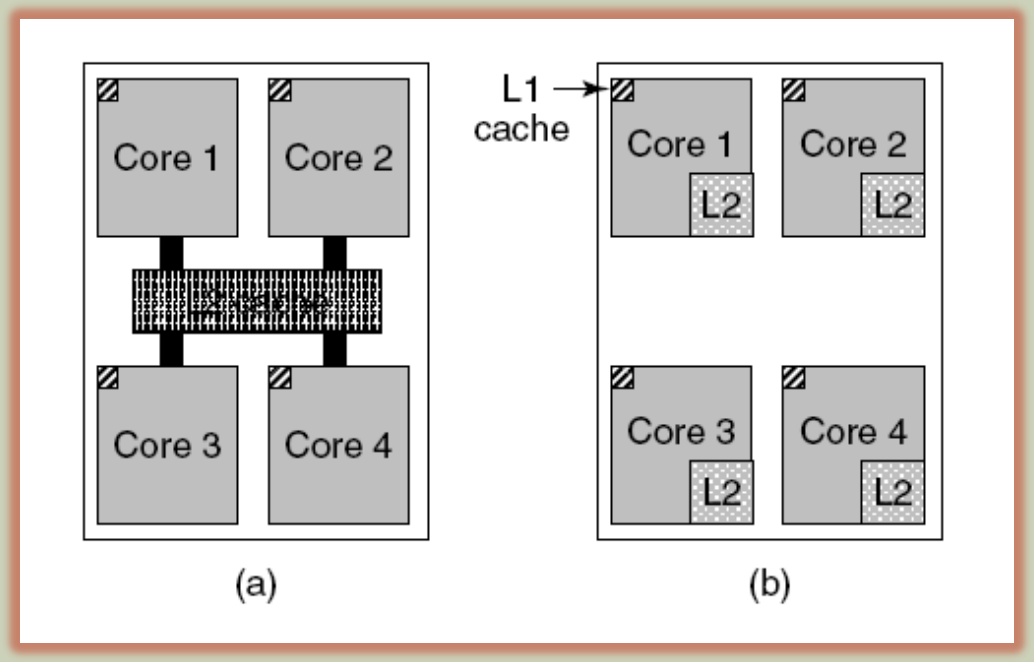
C) Execution based on units (three stages, pipeline)



D) Execution based on several units (pipeline)

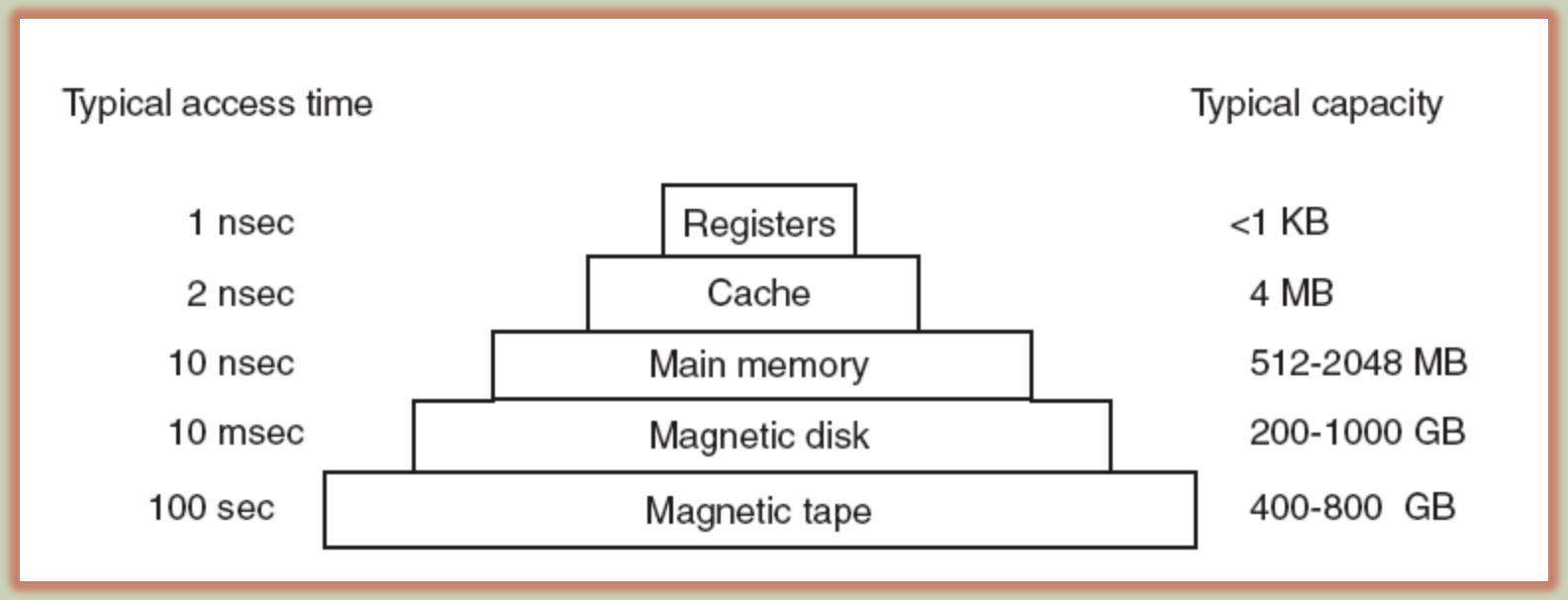


COMPUTER HARDWARE REVIEW MULTITHREADED AND MULTICORE



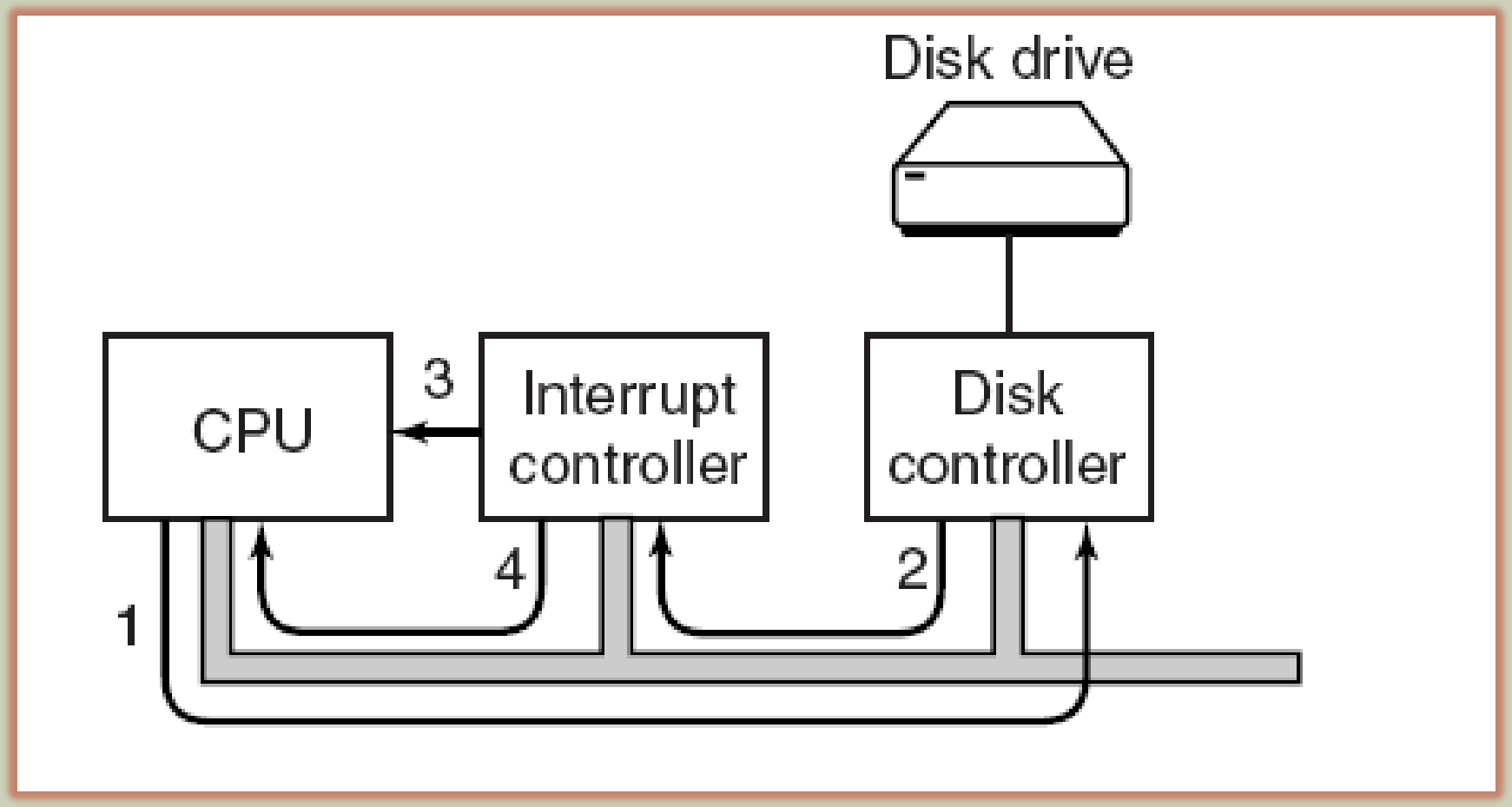
(a) A quad-core chip with a shared L2 cache.
(b) A quad-core chip with separate L2 caches.

COMPUTER HARDWARE REVIEW MEMORY



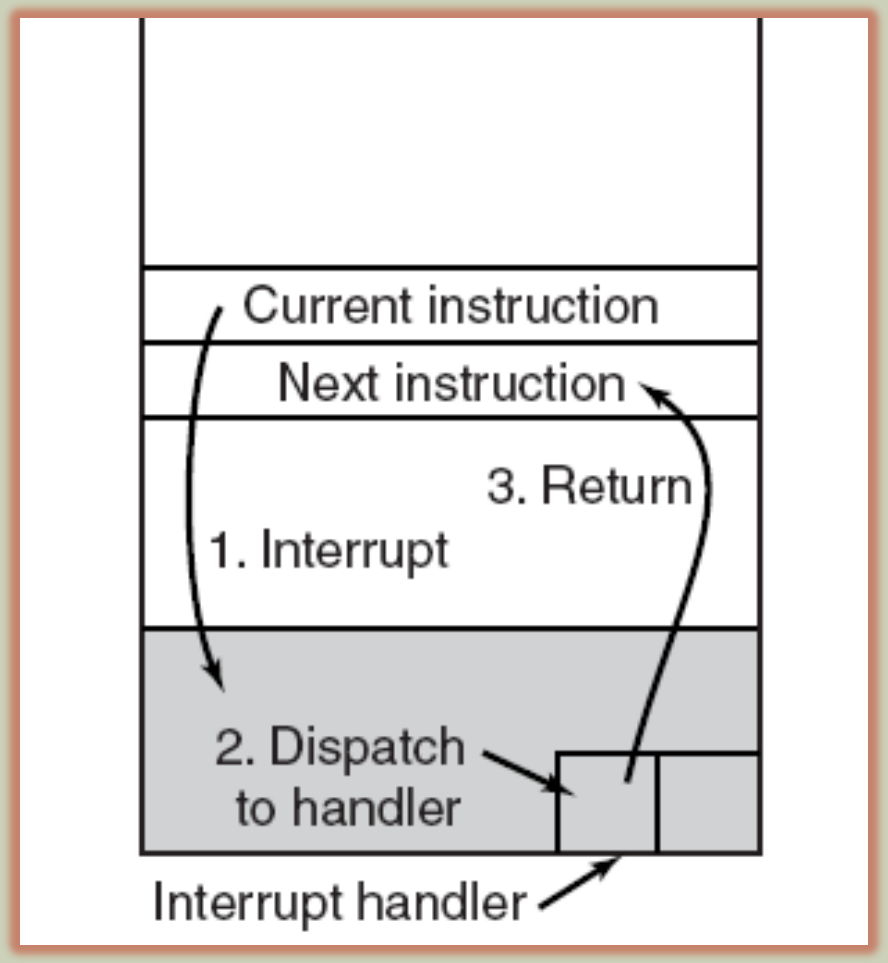
A typical memory hierarchy.
The numbers are very rough approximations.

COMPUTER HARDWARE REVIEW
I/O DEVICES



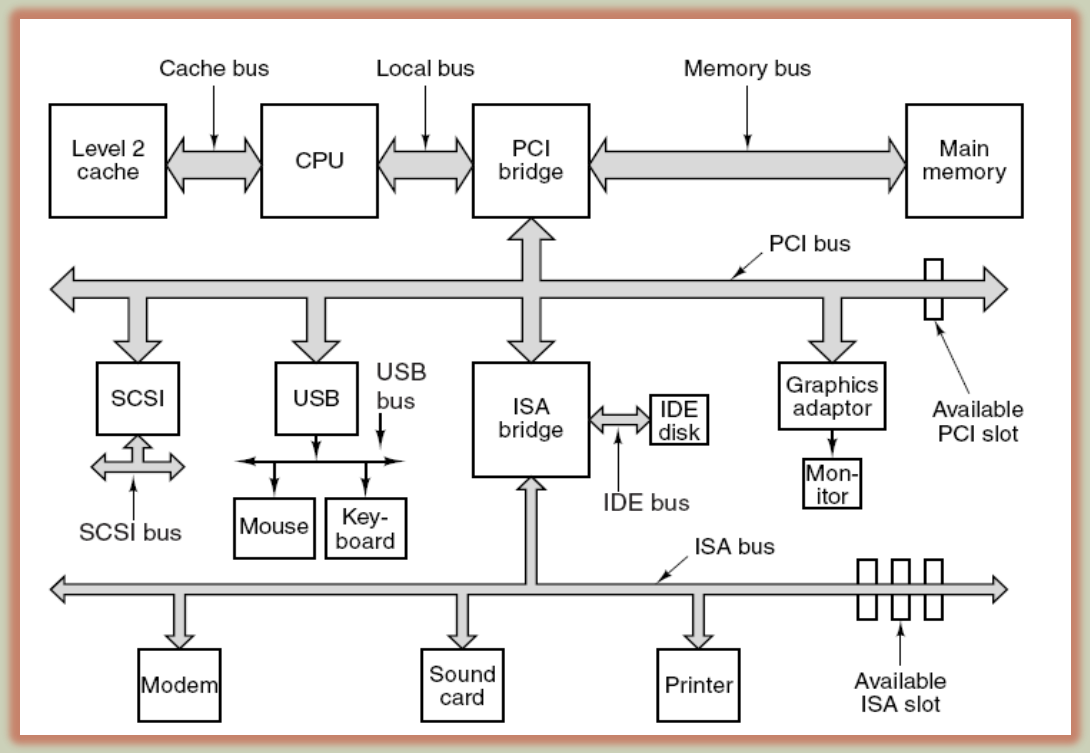
The steps in starting an I/O device and getting an interrupt

COMPUTER HARDWARE REVIEW
I/O DEVICES



- Interrupt processing involves taking the interrupt, running the interrupt handler, returning to the user program

COMPUTER HARDWARE REVIEW BUSES



The structure of a large Pentium system