Course TÖL401G: Stýrikerfi / Operating Systems 1. Introduction

Chapter Objectives

- Describe the role and responsibilities of an operating system (OS).
- Describe the general organization of a computer system and the role of interrupts.
- Describe the components in a modern multiprocessor computer system.
- Illustrate the transition from user mode to kernel mode.
- *Discuss how operating systems are used in various computing environments*.
- Provide examples of free and open-source operating systems.

Why Study Operating Systems?

- Only few will ever create or modify an operating system:
- Why, then, study operating systems and how they work?

- Simply because all code runs on top of an operating system!
  - Knowledge of how operating systems work is crucial for
    - effective (=works at all),
    - Efficient (=works fast, without wasting resource),
    - secure (=no vulnerabilities)

  programming of any application running on top of an OS.

Contents

Just a brief overview on these topics in this chapter 1, remainder of this course dedicates detailed chapters on each of these topics.

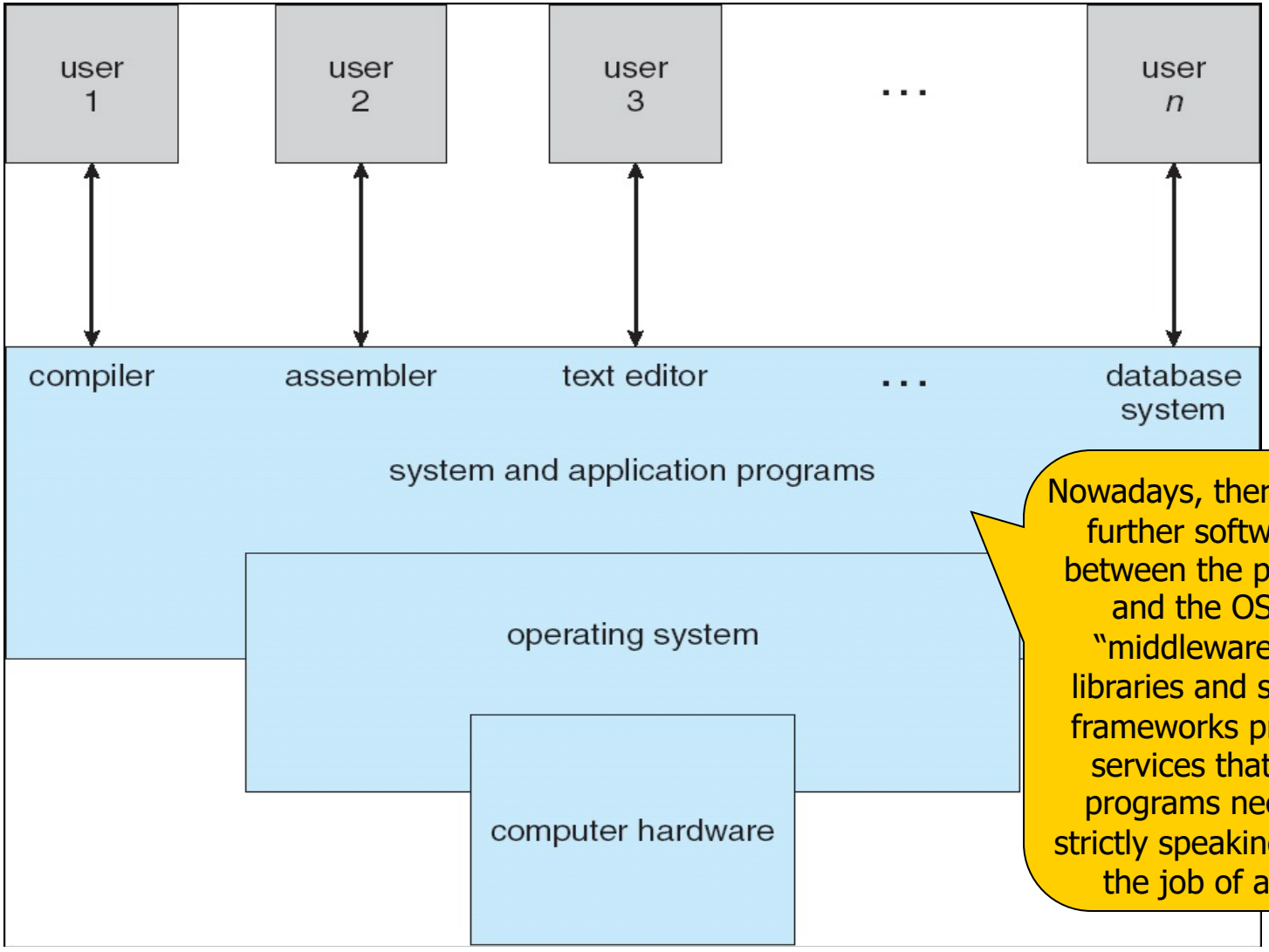
1.1 What is an Operating System?

- An operating system (OS) is a program that acts as an intermediary between a user of a computer (and the used application programs) and the computer hardware.
  - Provides environment for other programs & users to do useful work.
- Operating system goals:
  - Make the computer system convenient to use.
  - Use the computer hardware (& software) resources in an efficient manner.

Computer System Structure (1)

- A computer system can be divided into four components:
  - Hardware – provides basic computing resources:
    - CPU (Central Processing Unit), memory, I/O (Input/Output) devices.
  - Operating system:
    - Controls and coordinates use of hardware among various applications and users.
  - Programs – define the ways in which the system resources are used to solve the computing problems of the users.
    - System programs: copy files, list directory contents, format disc, etc.
    - Application programs: Word processors, compilers, web browsers, database systems, video games, etc.
  - Users:
    - People, machines, other computers.
- Relationship of these components as shown on next slide.

Computer System Structure (2) Relationship of Components



Nowadays, there is often further software in-between the programs and the OS: the "middleware", e.g. libraries and software frameworks providing services that many programs need, but strictly speaking are not the job of an OS.

Operating System Definitions (1)

Different definitions exist depending on view point:

- User view (Top-down view):
  - OS abstracts away hardware detail in order to ease the use of the involved hard- and software. Maximise the work the user performs.
- System view (Bottom-up view):
  - OS is a resource allocator:
    - Manages all resources.
      - Resource: hard- and software components that are relevant for program execution: CPU, Memory, I/O devices, Processes, etc.
    - 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.

Operating System Definitions (2)

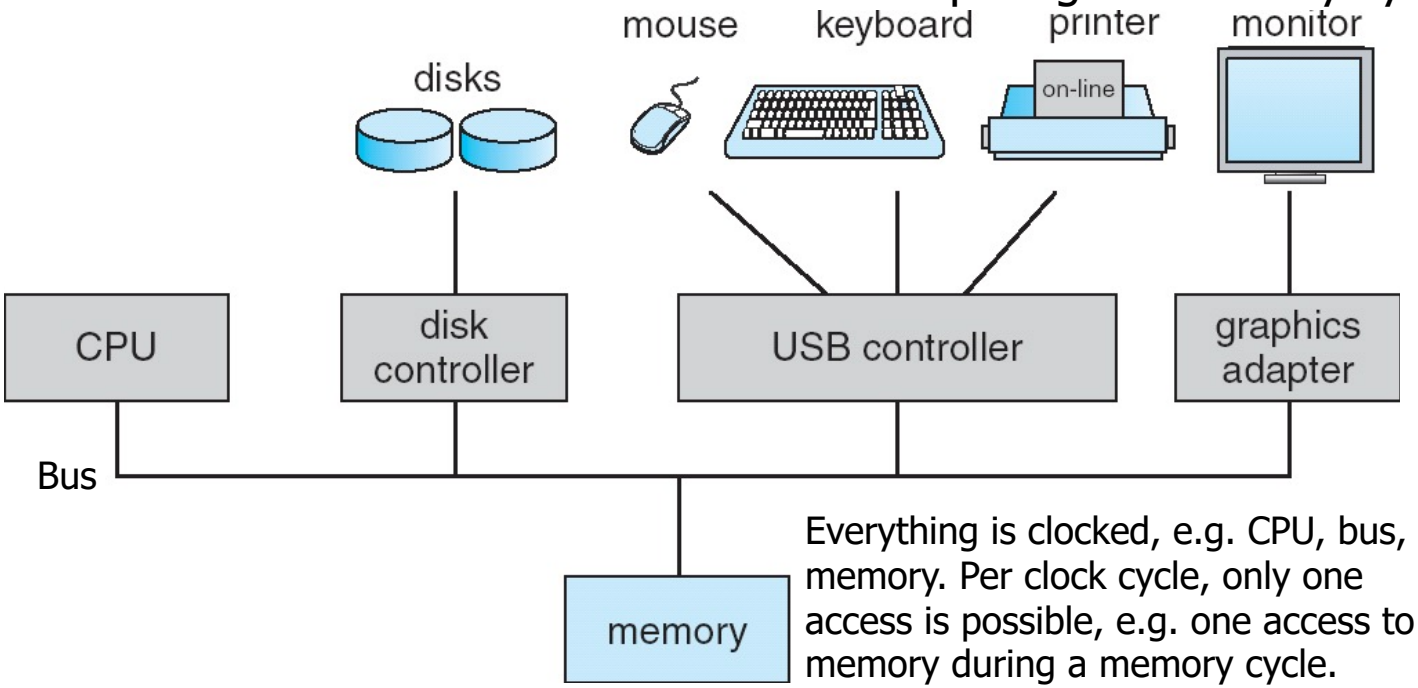
No universally accepted definition exists.

Instead, even various further definitions exist, e.g.:

- OS is "Everything a vendor ships when you order an operating system."
  - Trivia: *Microsoft was found guilty of shipping too much, e.g. bundling Internet Explorer browser, thus preventing competition*.
- "The one program running at all times on the computer is the **kernel.** Everything else is either a system program (ships with the operating system) or an application program (installed separately)."
  - System program: deals with resources managed by the OS.
    - E.g. system program to partition a storage device, create/format a filesystem, copy a file, list directory contents

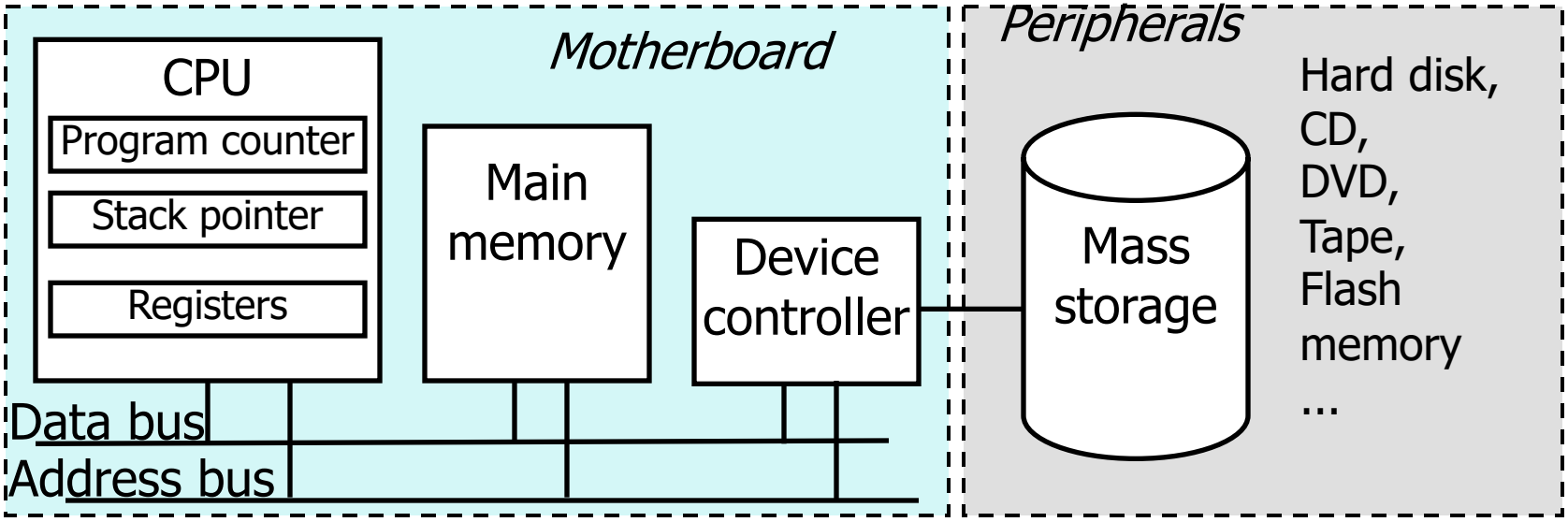
1.2 Computer System Organisation

- Computer-system operation:
  - One or more CPUs, device controllers connected through common bus providing access to shared memory.
  - Concurrent execution of CPUs and devices competing for memory cycles.



  - Further details on next slides

Excursion: Von Neumann Architecture



- CPU can only access locations (& execute instructions) in main memory.
  - Mass storage only indirectly accessible via device controller.
- Program counter: points to location of next instruction to be executed.
- Stack pointer: points to top of stack data structure used by CPU to save return address (program counter value) when calling a sub-routine.
- Registers: limited number of general purpose registers that hold, e.g., values of calculations or addresses of data stored in main memory.

Excursion: Bits and Bytes, k vs. K, etc.

- 1 Bit: smallest unit of information: 0 or 1.
  - (Bit often abbreviated as "b".)
- 1 Byte = 8 Bits.
  - (Byte abbreviated as "B", e.g. 10 B = 10 Bytes.)
  - Using binary system, possible to represent 2^8 (256) different values.
- Word size: number of bits each CPU register and the data bus has:
  - E.g. today's 64 Bit computers have a word size of 64 Bits $\Rightarrow$ in each memory access cycle, 64 Bits can be moved and 2^{64} different values can be stored in a CPU register.
- 1 k (kilo)= 1000
  - E.g. 1 km = 1000 m
- 1 K (kilo as binary prefix, using uppercase letter) = 1024
  - E.g. 1 KB = 1024 B
- Kilo (K) =1024, Mega (M) =1024^2, Giga (G) =1024^3, Tera (T), Peta (P), Exa(E)
- For mental math: 2^{10} (=1024) roughly 10^3 (=1000).
  - (E.g. when you are in the final exam without a pocket calculator…)

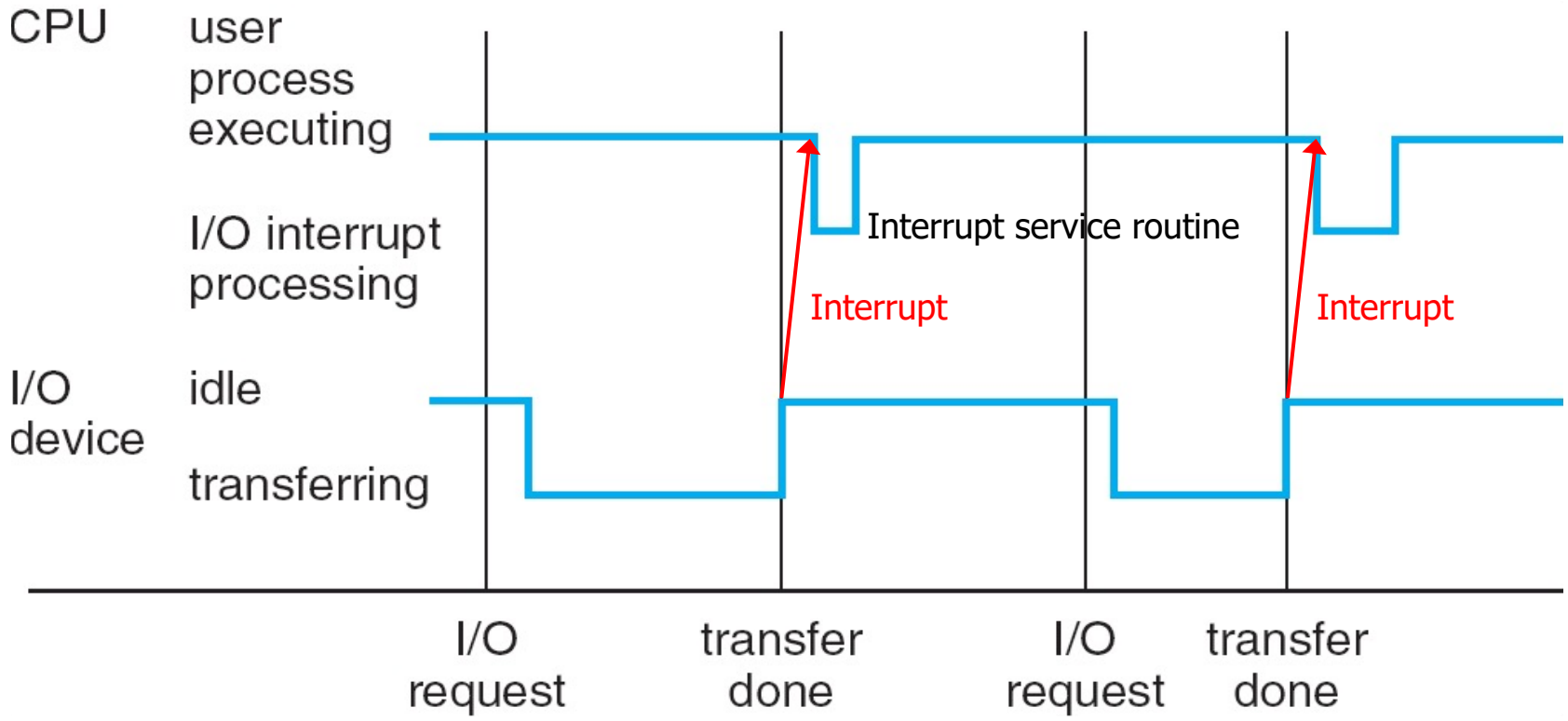
1.2.1 Computer-System Operation

- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer for the data that is currently subject of input or output.
- For inputting and outputting data, CPU copies data via the bus between the local buffers of a device controller and main memory.
- Actual I/O is between local buffer of controller and the actual I/O device.
- Device controller informs CPU that it has finished its operation by causing an *interrupt*.

Common Functions of Interrupts

- **Interrupt** transfers control from any instruction that is currently executed by the CPU to the interrupt service routine.
  - *Interrupt vector* (=array/table in main memory) contains the addresses of all the service routines (=instructions) for each type/priority of interrupt.
  - Interrupt architecture must save the address of the interrupted instruction. Once interrupt service routine completes ex. resumes at the saved address.
  - Usually, interrupts of a lower priority are *disabled*/*masked* while interrupt of a higher priority is being processed: avoid lower priority interrupts prevent the higher priority interrupt service routine from being completed.
- **Hardware** interrupt such as from the s I/O system
- **SW *trap*** /*exception* is a software-generated interrupt caused either by an error (e.g. division by zero) or a user/application program request.
  - System calls (i.e. calling an OS function) typically by means of a software-generated interrupt.
- An operating system is *interrupt driven*.

Interrupt Timeline



- Interrupt processing is time critical:
  - Interrupt service routine must have finished before the next interrupt calls it again: If not, previous I/O data would not get completely processed & thus lost.
  - An I/O controller that raises an interrupt, typically needs the data quickly to be processed otherwise the data gets lost (e.g. if I/O device provides new data before the interrupt service routine grabbed the old data from the device controller).

Interrupt Handling

- Operating system
  - provides interrupt service routines,
  - initializes interrupt vector accordingly.

  } both located in main memory
  - Configures on multicore systems an APIC (Advanced Programmable Interrupt Controller) to which core(s) which interrupts are routed.
- Interrupt service routine preserves the state of the CPU by saving registers and the program counter (typically already saved by the CPU hardware when an interrupt is detected) and restores it afterwards.
  - Different segments of code determine what action should be taken for each different type of interrupt, e.g.:
    - Transferring data between main memory and local buffer of device controller.
    - Executing an OS function (system call).

1.4 Operating System Structure

- **Multiprogramming** needed for efficiency of (old) batch systems:
  - Single user cannot keep CPU and I/O devices busy at all times.
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute.
  - A subset of total jobs in system is kept in memory.
  - One job selected and run via job scheduling.
  - When job has to wait (for I/O for example), OS switches to another job.
  - Rarely used anymore today.
- **Timesharing** (**multitasking**): extension of multiprogramming in which CPU switches jobs so frequently (**not only when job has to wait**) that users can interact with each job while it is running, creating interactive computing.
  - Response time should be < 1 second.
  - Each user has at least one program (process) executing in memory.
  - If several jobs ready to run at the same time: CPU scheduling.
  - If processes don't fit altogether in memory, swapping moves them in and out to run.
  - Virtual memory allows execution of processes not completely in memory.
  - Timesharing used in today's PC operating systems.

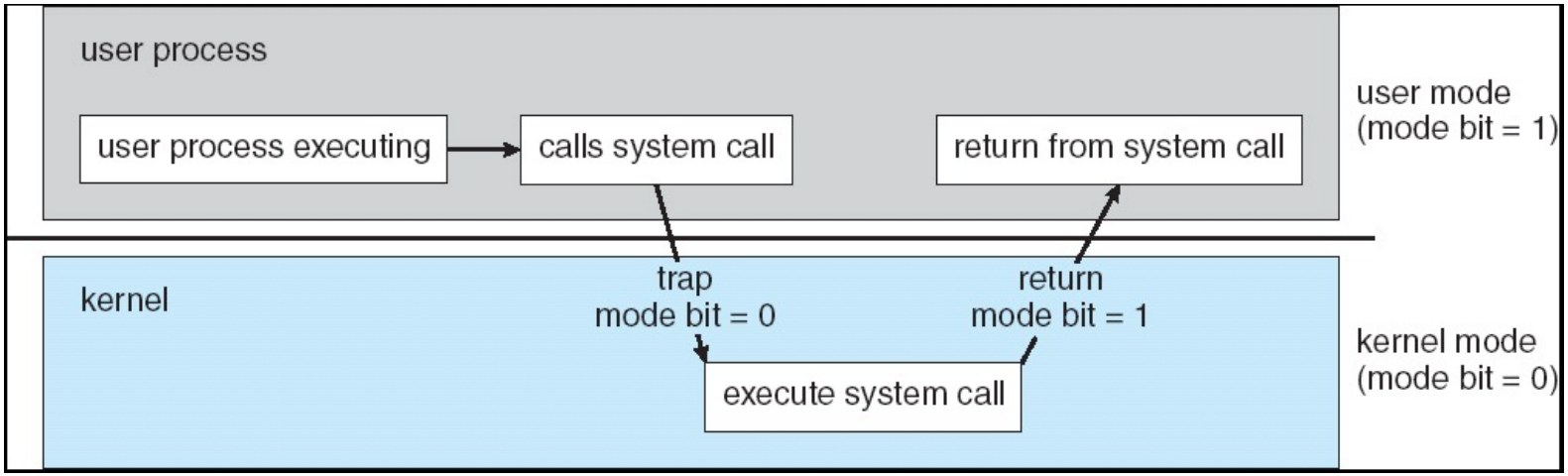
1.5 Operating-System Operations

- An operating system is interrupt driven:
  - Hardware interrupt, e.g.:
    - Service an I/O device controller
    - Periodic timer that initiates frequently switching of multitasking jobs.
  - Software interrupt, e.g.:
    - Error (e.g. Division by zero)
    - Request for operating system service by application ("system call") using an exception or trap.

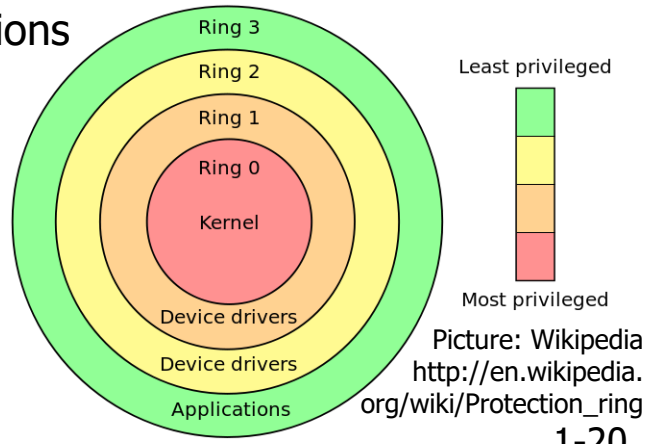
Dual-Mode Operation

- Dual-mode operation allows OS to protect itself and other system components (e.g. processes modifying each other or the operating system):
  - Two modes: User mode and kernel mode (supervisor mode).
  - Mode bit provided by CPU (CPU status bit):
    - Provides ability to distinguish when system is running user code ("User space") or kernel code ("Kernel space").
    - Some CPU instructions are privileged: only executable in kernel mode.
      - Typically, certain memory regions (e.g. containing kernel data structures or access to I/O devices) may only be accessed in kernel mode.
    - Each interrupt changes mode automatically to kernel mode, return from interrupt resets it previous (=typically user) mode.
      - As system calls are called by issuing a software interrupt, the CPU is automatically in kernel mode when a system call shall be executed.
      - **No other way** to switch to kernel mode than by an interrupt!

Transition from User to Kernel Mode



- Memory hardware typically restricts memory access in user mode.
  - ⇒ User process can only access it's own memory ("Memory protection").
    - Process that runs wild, cannot overwrite instructions or data of other processes or even the kernel.
- A CPU may have more than just the two modes, then typically called "rings".
  - There may be even a ring -1 for running a ring 0 kernel in a virtual machine hypervisor (=ring -1). →Ch. 2



Picture: Wikipedia http://en.wikipedia.org/wiki/Protection_ring

Interrupts and Timer

- Not only a system call made by an application switches into kernel mode, but also each interrupt. E.g.:
  - Interrupt raised by I/O device controller:
    - Now, the kernel needs to serve that I/O device controller.
    - Computer hardware is designed to allow I/O device controller only to be accessed in kernel mode.
      - Otherwise an ordinary user process might circumvent the OS and directly access the hardware.
    - Convenient that interrupt switches to kernel mode needed by kernel!
  - Timer used for multitasking(/timesharing):
    - Set timer interrupt after specific period (e.g. every 20ms).
    - Timer interrupt transfers control to operating system:
      - OS gains control even if process is stuck in an infinite loop.
      - CPU scheduling may take place and select next process to be executed.

1.6 Process Management

- A process is a program in execution. It is a unit of work within the system.
  - A program is a passive entity (program file stored on mass storage device),
  - A process is an active entity.
- A process needs resources to accomplish its task:
  - CPU, memory, I/O, files, initial data.
  - Process termination requires operating system to reclaim any reusable resources.
- Single-threaded process has one program counter specifying location of next instruction to execute:
  - Process executes instructions sequentially, one at a time, until completion.
- Multi-threaded process has one program counter per thread.
  - More on threads in chapter 4.
- A typical system has many processes, few users, one operating system running concurrently on one or more CPUs/cores.
  - Concurrency by multiplexing the CPUs among the processes / threads (multitasking/timesharing).

Process Management Activities

- Process management activities of operating system:
  - Creating and deleting both user and system processes,
  - Scheduling processes (and threads) on the CPUs,
  - Suspending and resuming processes,
  - Providing mechanisms for process synchronization,
  - Providing mechanisms for process communication.

1.7 Main Memory Management

- Instructions of a process need to be in main memory in order to execute process.
- Data of a process need to be in main memory during processing.

- Memory management determines what is in main memory when optimizing CPU and resource utilization and computer response to users.
- Memory management activities:
  - Keeping track of which parts of main memory are currently being used and by whom.
  - Deciding which processes (or parts thereof) and data to move into and out of main memory.
  - Allocating and deallocating main memory space as needed.

Low-Level Mass-Storage Management

- Traditionally, disks (now trend to SSD) used to store data that does not fit completely in main memory (e.g. virtual memory) or data that must be kept for a long period of time (e.g. files).
  - Proper management is of central importance.
  - Entire speed of computer operation depends on disk subsystem and the operating system's algorithms used to manage it.

- Low-level mass-storage management activities:
  - Free-space management,
  - Storage allocation,
  - Disk scheduling.

- Some storage needs not be that fast:
  - Tertiary storage includes optical storage, magnetic tape.
    - Seldom-used data, thus speed not crucial. But: still must be managed!

Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software).
- Information in use copied from slower to faster storage temporarily.
- Faster storage (cache) checked first to see if information is there:
  - If it is, information used directly from the cache (fast).
  - If not, data copied to cache and used there.
- Cache smaller than storage being cached (as faster cache memory is more expensive than slower memory).
  - Cache management: important design problem.
  - Cache size and replacement policy.
- Caching at different levels, e.g.:
  - CPU cache as cache for main memory.
  - Main memory as a cache for secondary storage.
  - Web browser cache as cache for network transmission.

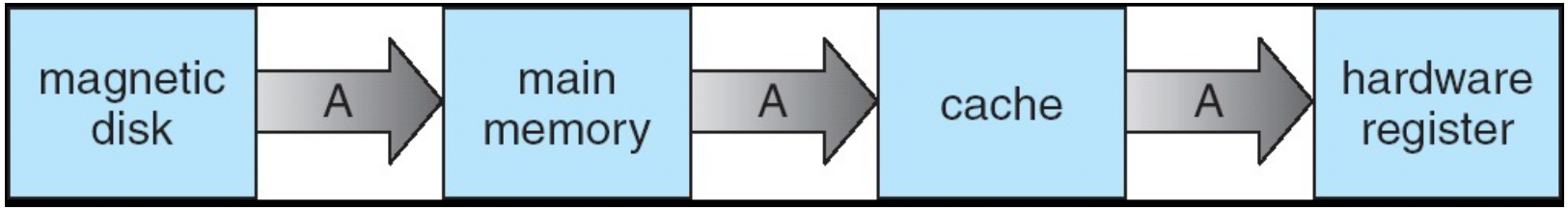
Performance of Various Levels of Storage

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

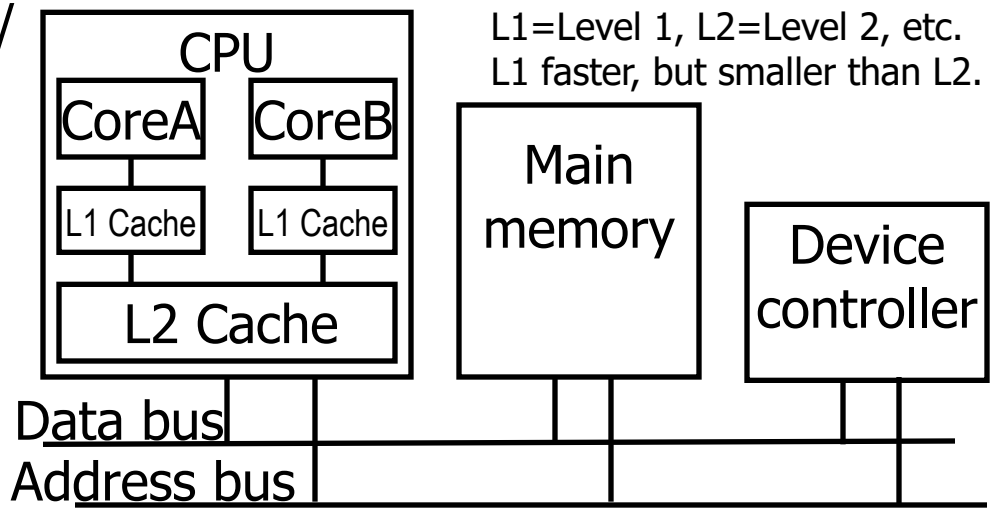
- In the past, cost was the main driver that prevented to use larger main memory.
- Then, the maximum memory addressable by a 32 Bit CPU became a restriction:
  - 32 Bit CPU uses 32 Bit to address memory: maximum memory addressable with 32 Bit: 4GB
- Today's 64 Bit CPUs allows to address larger memory.

Migration of Integer A from Disk to CPU Register

- Care must be taken to use most recent value, no matter where it is stored in the storage hierarchy.



- Multiprocessor environment must provide cache coherency in hardware such that all CPUs/ cores have the most recent value in their cache.
  - E.g.: L1 cache of Core A must be updated if Core B changes a value in main memory that is cached by L1 cache of Core A.



1.9 Protection and Security

- Protection: any mechanism for controlling access of processes or users to resources defined by the OS.
- Security: defense of the system against internal and external attacks.
  - Huge range: denial-of-service, worms, viruses, identity theft, theft of service, …
- To realize protection and security, operating systems generally distinguish among users, to determine who can do what:
  - User identities (user IDs, security IDs) include name and associated number, one per user.
    - User ID then associated with all files, processes of that user to determine access control.
  - Group identifier (group ID) allows set of users to be defined and to manage access control for whole groups of users.
    - Group ID also associated with each process, file.
  - Privilege escalation allows user to change to effective ID with more rights.

1.11 Special Purpose Systems: Embedded systems

- Embedded computers today found everywhere:
  - Cars, DVD players, microwave ovens, etc.

- Have very specific tasks.
  - No general purpose system.
  - Restricted hardware, e.g.
    - No mass storage device,
    - User interface just a few buttons and status light.

  - OS provides limited features, e.g.
    - No mass storage management,
    - Little user interface.

Special Purpose Systems: Real-Time Operating System

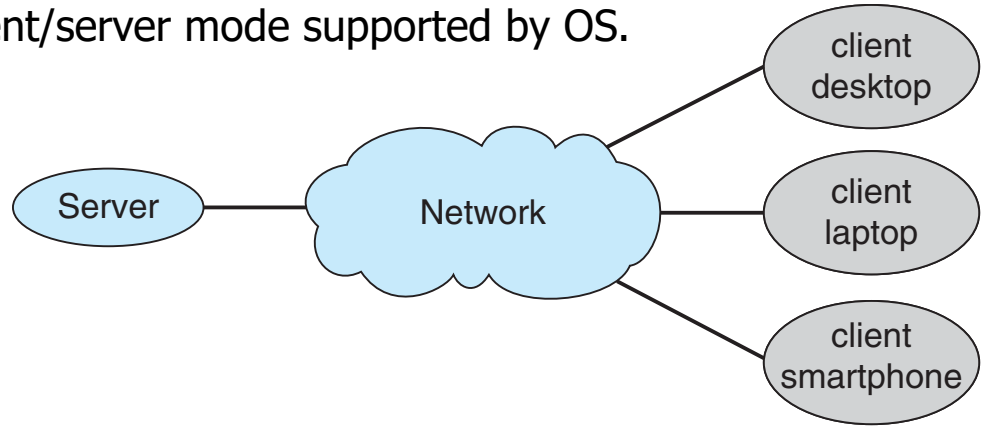
- Embedded systems often run real-time OS.
- Used in controlling devices: processing *must* be done within a defined time constraint, otherwise the system will fail. E.g.:
  - Industrial control system,
  - Traffic control system,
  - Medical systems,
  - Multi-media system (audio/video must be delivered in time to avoid stuttering).
- Real-time OS needs to support processing with well-defined fixed time constrains (ROS-kernel <= 1ms).
  - Must be taken into account when designing e.g. CPU scheduler.

Special Purpose Systems: Mobile Computing

- Smartphones and tablets allow mobile computing.
  - On one hand restricted hardware (e.g. no keyboard, no mouse, no hard disk) in comparison to a PC,
  - but on the other hand even more hardware (e.g. GPS, compass, accelerometer).
  - Tablets and laptops converge and handheld devices have nowadays functionalities like an ordinary computer.
  - Generic OS (e.g. Android based on Linux), however with
    - support for power saving (due to limited battery),
    - special user interface (touch instead of physical keyboard or mouse),
    - wireless (WiFi) and cellular data network (3G, 4G, 5G) support,
    - limited storage capacity support (no hard-disk drivers).

1.12 Computing Environments: Client-Server Computing

- Client-Server Computing / Distributed System:
  - Computer as servers, responding to requests generated by clients.
    - Compute-server provides an interface to client to request services, e.g. database.
    - File-server provides interface for clients to store and retrieve files.
    - Client/server mode supported by OS.



- Web-based Computing (special application of Client-Server computing):
  - Web server; Web browser as client to access powerful cloud services.
  - Web has become ubiquitous: most devices networked to allow web access.
    - E.g. Chromebook: Linux-based OS, mainly providing a web browser.

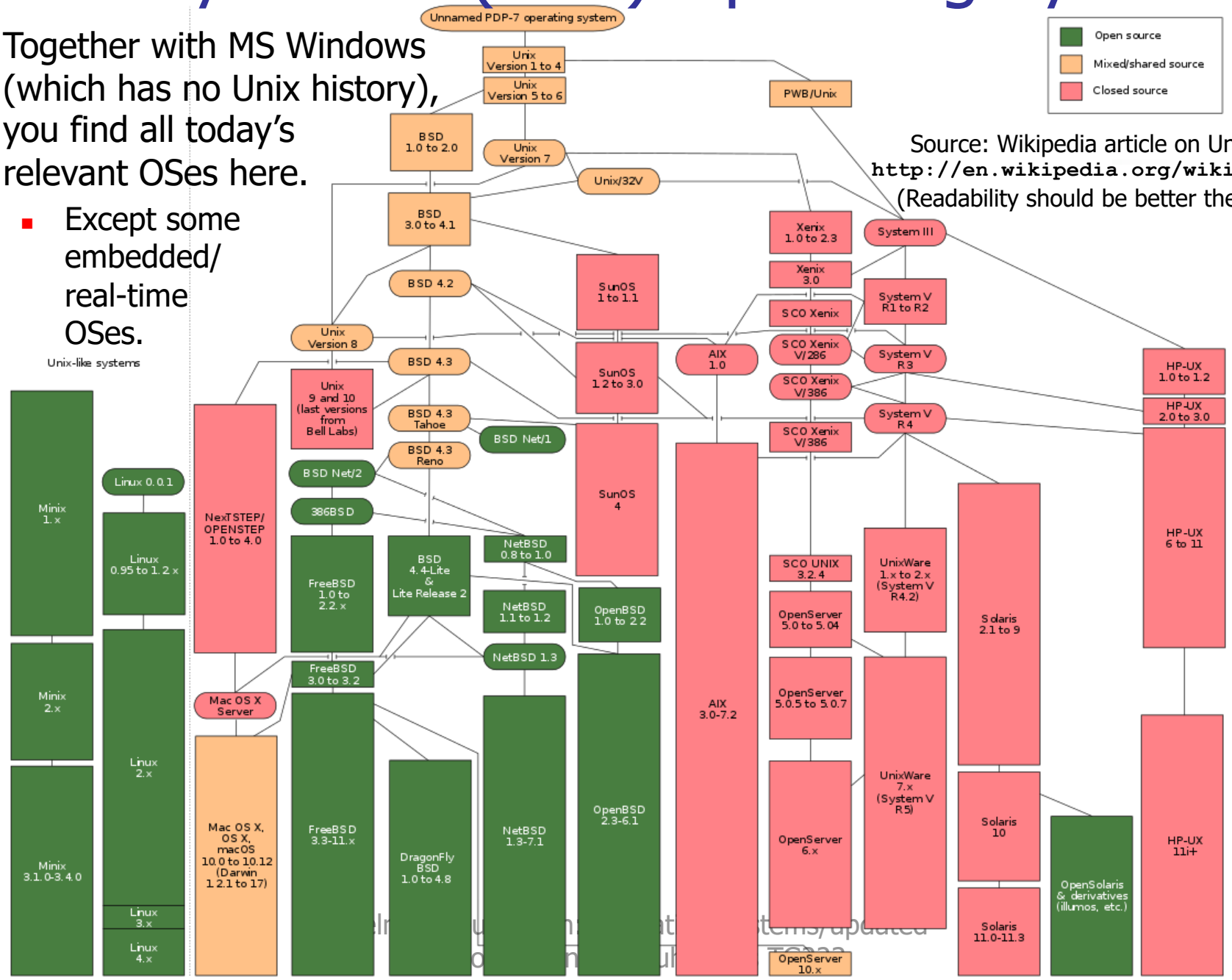
1.13 Free and Open-Source vs. Closed Source Operating Systems

- Free and Open-source software (FOSS): Source code of OS (e.g. Linux) can be accessed (open-source) and modified and redistributed for free (free software).
  - Good for learning about OS implementation.
    - Download the source code of the Linux kernel from **`http://kernel.org`** and have a look at it.
  - Possible to fix bugs and extend on your own.
  - Arguably, more secure: many eyes can view code to find problems.
- Closed-source: Only compiled binary available (e.g. Microsoft Windows).
  - Need to rely on support of vendor to get bugs fixed.
- Hybrid: E.g. Apple Mac OS X: Darwin OS kernel is FOSS, system programs and middleware (e.g. GUI) are not.

History of Unix(-like) Operating Systems

Together with MS Windows (which has no Unix history), you find all today's relevant OSes here.

- Except some embedded/ real-time OSes.



Source: Wikipedia article on Unix
`http://en.wikipedia.org/wiki/Unix`
(Readability should be better there)

1.14 Summary

- Operating system mediates between user/application and underlying hardware.
  - Manages all the resources.
  - Lots of responsibilities.
  - We will have a closer look on each of them in the following chapters.