Operating System Concepts
Ch. 1: Introduction

Silberschatz, Galvin & Gagne

Universiteit Leiden
The Netherlands

An Operating System

- What?
- Why?
- Where?
- Definition?
- How?

What?

- What is an Operating System? The textbooks answers:

  "A program that acts as an intermediary between a user of a computer and the computer hardware"

- A single program or software package? Hard to define.

Why?

What are the goals of an Operating System? Why are these developed?

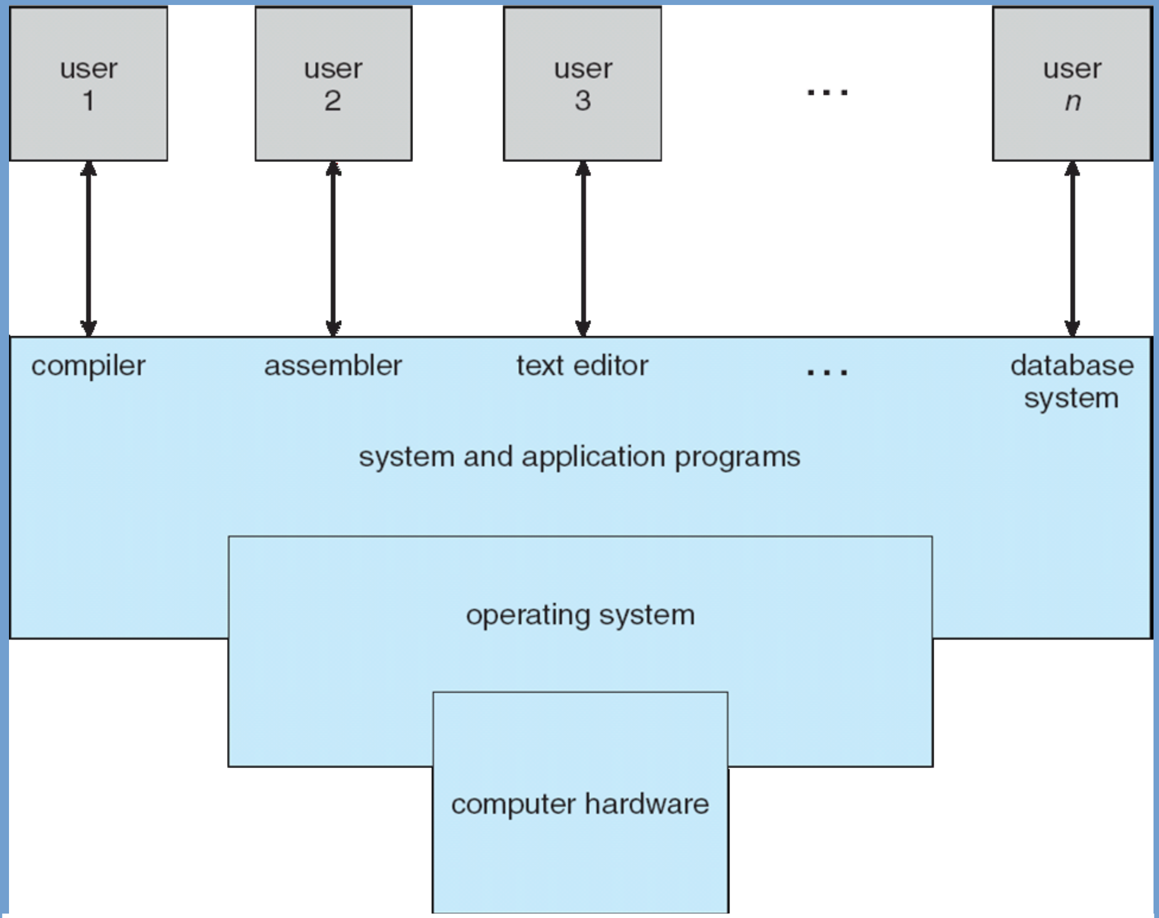
- Make a computer system ***convenient*** to use.
  - Imagine everybody has to write their own drivers and write bare-metal software …
- Use computer hardware in an ***efficient*** manner.
  - Share available resources
- So, combined: allow a system to be used by multiple users and provide an interface to write programs against.

Structure of computer systems

- The textbook divides a computer system into 4 main components:
  - ***Hardware***, providing the resources used for computing: CPU, main memory, disk drives, network interfaces.
  - ***Operating System***, which controls and coordinates the use of the hardware and provides an abstract interface to the hardware.
    - Note: situated between hardware and application programs.
  - ***Application Programs***, programs that run on top of the operating systems and solve the user's problems.
  - ***Users of the system***, people but also other computers and machines.

Structure of computer systems (2)

- Schematically:



Source: Silberschatz et al., Operating System Concepts, 9th Edition

Where?

- Operating Systems are implemented and optimized for different purposes.
  - *Desktop computers*: easy of use, good performance. Energy consumption or inefficient use of the hardware not immediately a concern.
  - *Smartphones*: modern UI, low response times, good battery life.
  - *(Classical) shared computers*: good performance, responsiveness, efficient use of available resources, fair scheduling.
    - This is in fact where it all started: *mainframe* computers, later *minicomputers* and *microcomputers*.
  - *Various embedded systems*, which do not have a clear user interface. These typically take action in response to observed sensor inputs.

Mainframe

Source: https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_2423PH3165.html

Minicomputer

Source: http://nl.wikipedia.org/wiki/Minicomputer#mediaviewer/File:Pdp-11-40.jpg

Microcomputer

Source: http://www.tpsoft.com/museum_images/IBM%20PC.JPG

Towards a definition

- There is in fact no universally accepted definition of "Operating System".
- Often much more than a single "program". What is counted as part of the OS? What isn't?
- An OS typically consists of:
  - A ***kernel***: a program (executable) that is always loaded in memory and in control in the background.
  - ***System programs*** that support the kernel.
- Some systems come with various application programs (Notepad, Patience, various Linux packages) that one would not count as part of the OS.
  - And what about web browsers? Part of major lawsuit in the past!

Main responsibilities

An Operating System has two main responsibilities:

- Resource allocation
  - The system manages and operates available resources (CPU cycles, main memory, space on disk drives, etc.)
  - Ensures efficient and fair use: has policies in place to decide what to do in case of conflicting requests.

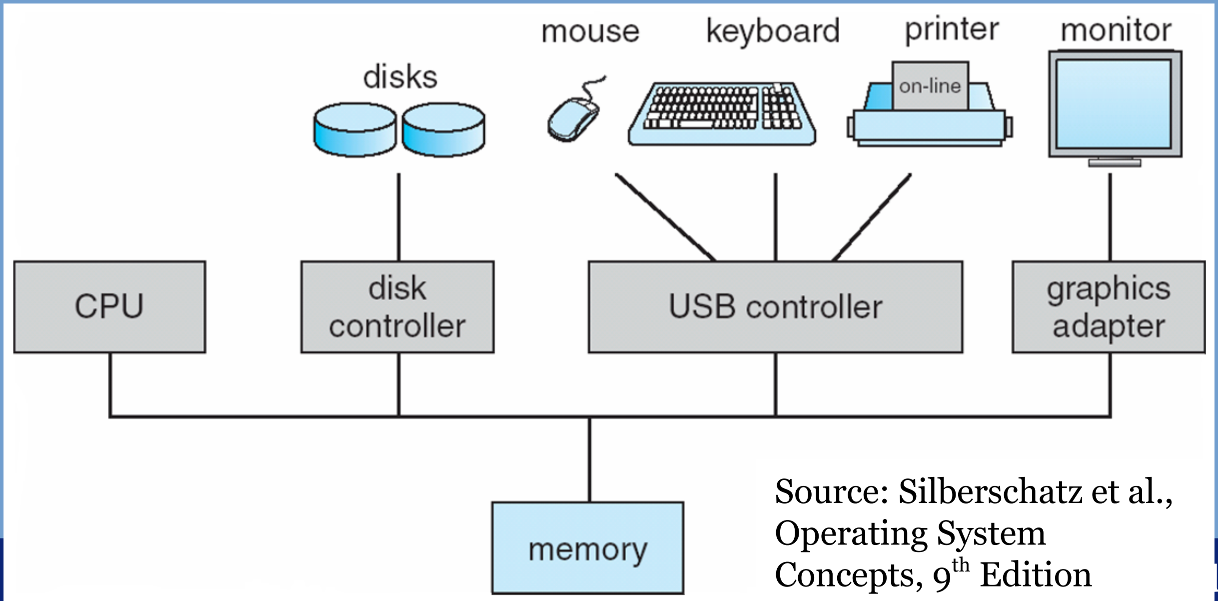
- Control & isolation
  - It controls the execution of programs to prevent errors, harm to the computer and other users.

How?

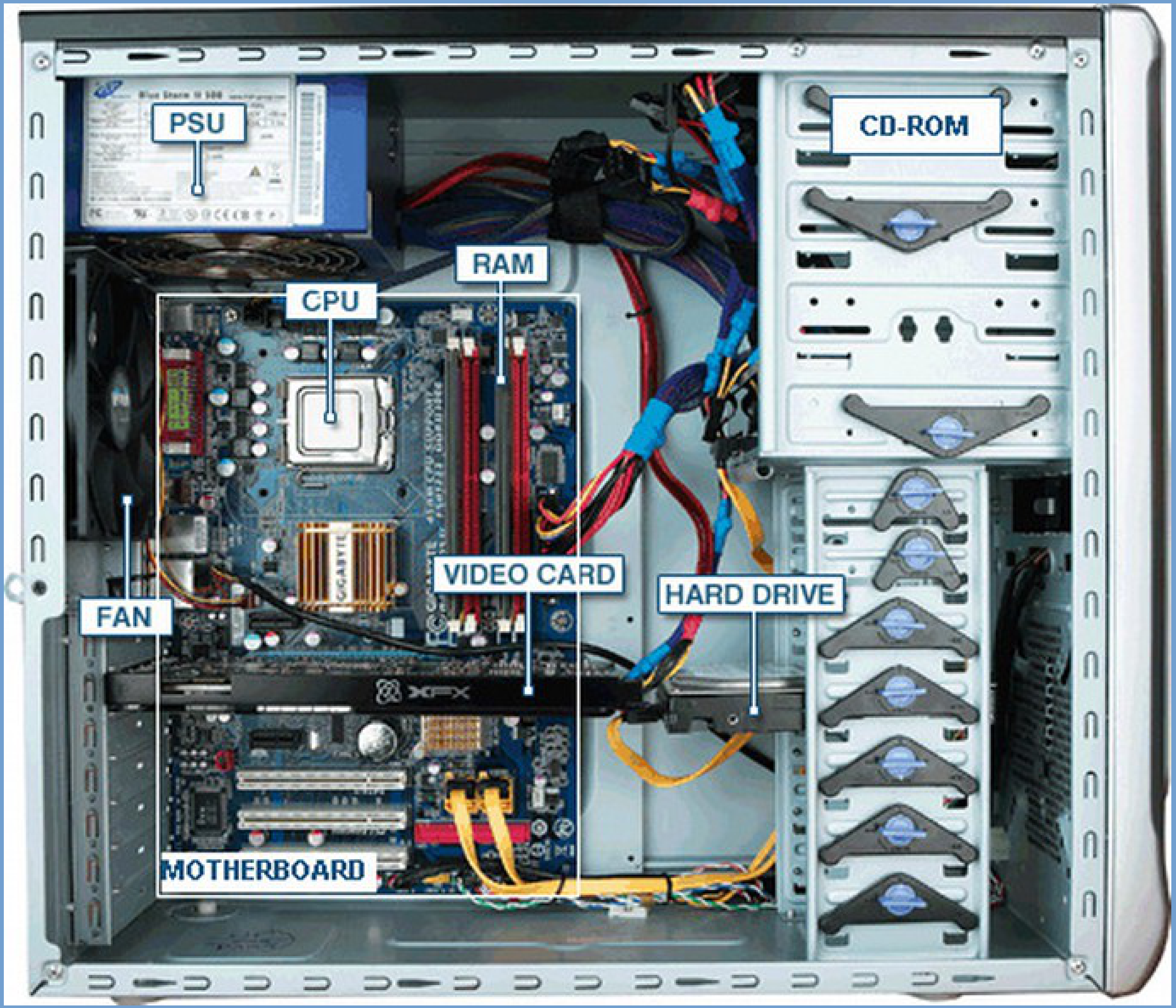
- To be able to discuss ***how*** operating systems are implemented, we must understand the underlying hardware organization.
- Why? Because an operating system controls and operates the hardware comprising a computer system.
- The hardware available influences the design of the operating system!
  - For instance, what is the backing store from which programs are loaded? Tape? Hard drive? SSD? Non-volatile storage?
  - And on the other hand, the operating system must supply device drivers that can control/operate these devices.

Organization of computer systems

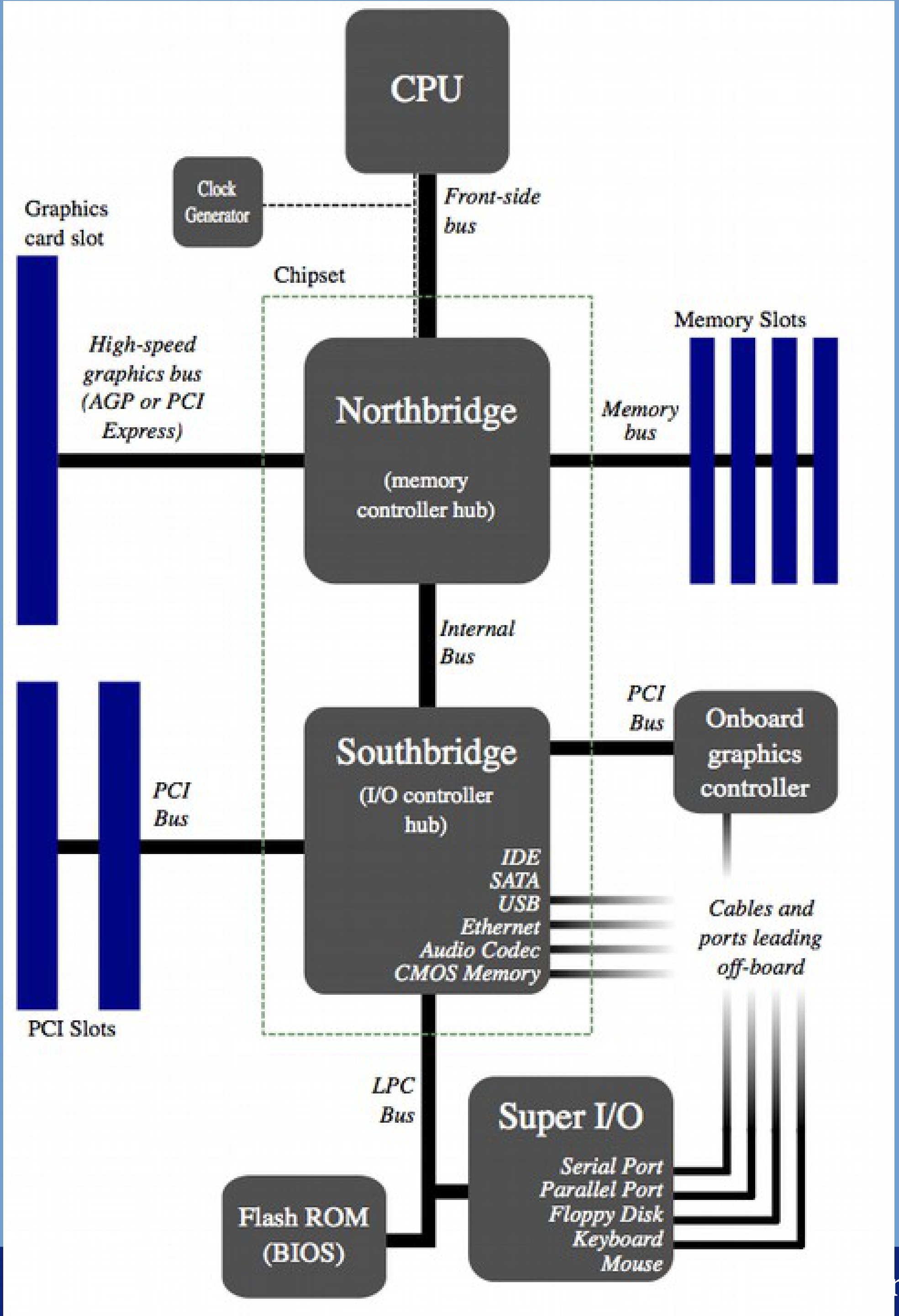
- As we know from computer architecture: it all starts with CPU(s) and main memory. These are connected by a memory bus.
- Next to this, there are many peripherals.
  - These either share the same memory bus, or are connected with an additional bus.
  - CPUs and devices operate concurrently
  - CPUs and devices compete for access to main memory.



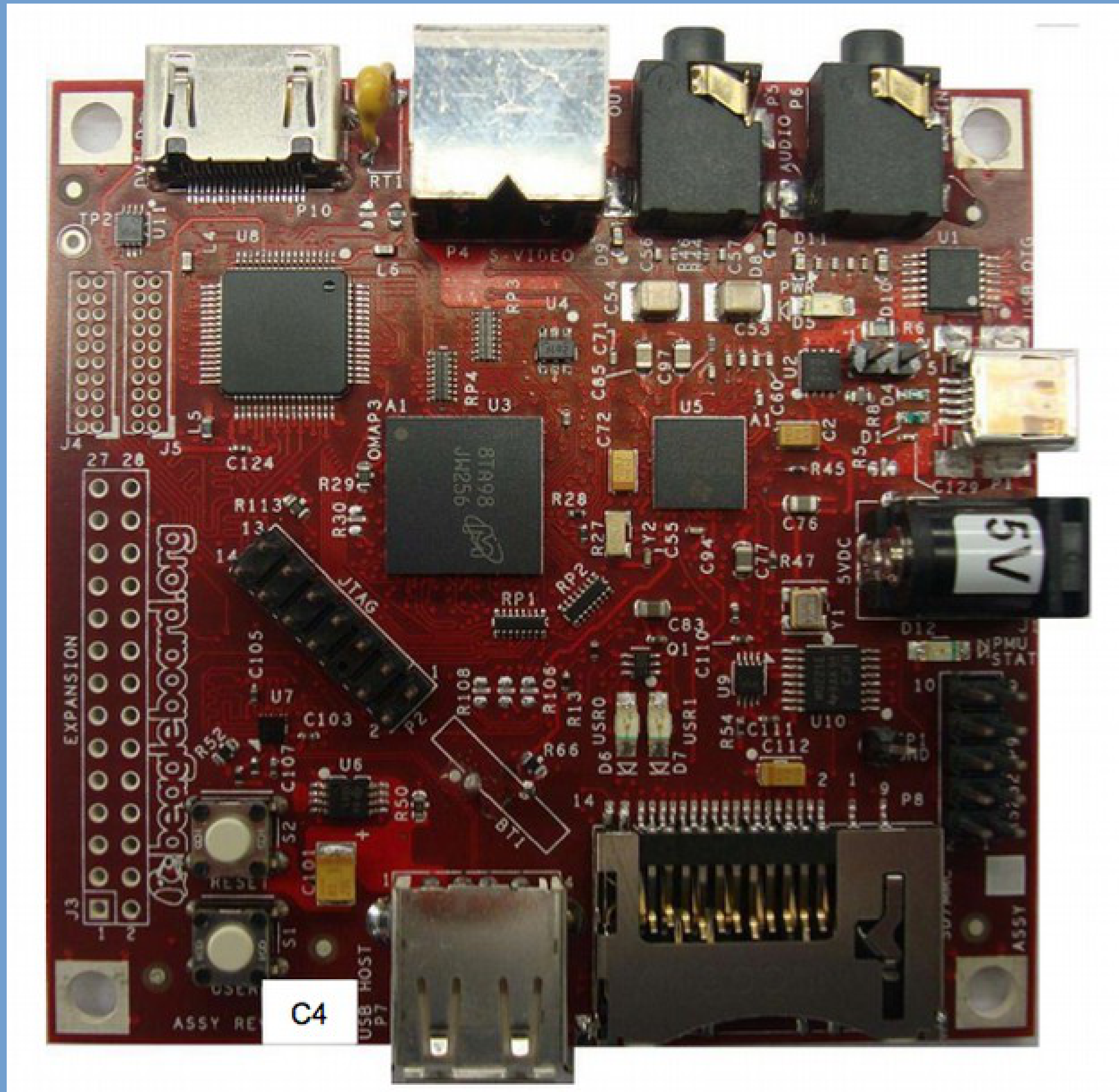
Source: Silberschatz et al., Operating System Concepts, 9th Edition



Source: http://tazalink.blogspot.nl/2011/02/some-useful-parts-of-your-pc.html



Source: http://en.wikipedia.org/wiki/Northbridge_%28computing%29


C4
5V

Operation of computer systems

- When the system is powered on, a bootstrap program is loaded.
  - Often stored in a ROM or EEPROM chip on the mainboard.
  - BIOS, EFI, OpenFirmwire, ...
  - Its purpose is to perform low-level initialization of the system and to load the kernel into memory and jump to it.
- The kernel further initializes all devices and internal data structures.
- After that, it sits idle awaiting commands.
  - From the user, system programs, or other computers.

Operation of computer systems (2)

- How does the kernel receive commands?
  - Interrupts; raised by devices.
  - Exceptions or traps; raised by software due to an error or to put a request.
- Example interrupts
  - *Keyboard controller*: When key strokes are present in the internal buffer, the keyboard controller generates an interrupt.
  - *Disk drive*: OS requests transfer of disk blocks. Once completed, disk I/O controller generates an interrupt.
  - *Networking*: When a network packet is received, an interrupt is generated.

Device Interrupts

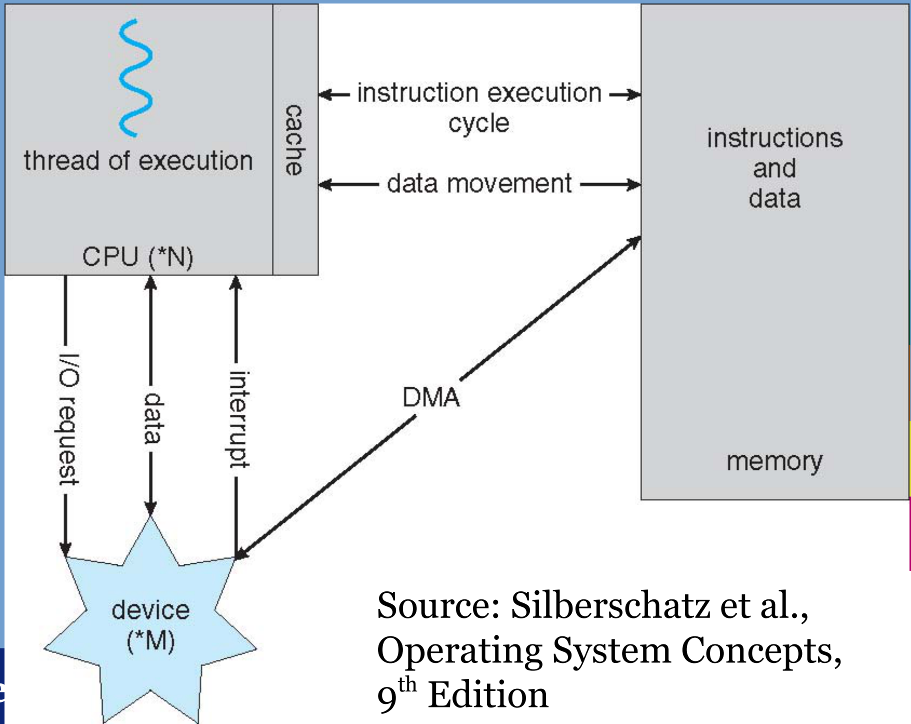
- Each device has a controller (typically). These controllers and the CPU execute concurrently.
- The controller has a local data buffer in which (some) data can be stored that is currently processed.
- At some point, this data needs to be transferred to main memory (or vice versa).
- Using an interrupt, service is requested from the CPU to transfer this data.
  - Nowadays Direct-Memory Access (DMA) is more common.

Device Interrupts (2)

- An interrupts raises a line on the CPU, causing it to suspend its current task (and save state: registers & program counter) and jump to an *interrupt service routine (ISR)*.
- Which routine to jump to?
  - Either poll an interrupt controller to find out,
  - or we have a vectored interrupt system.
- From an interrupt vector follows a pointer of an ISR to jump to, which will handle this interrupt.
- ISRs are installed by the operating system.

Direct Memory Access (DMA)

- DMA allows device controllers to access main memory directly, without involvement of the CPU.
- So, the CPU can do something useful while the device controller performs the data transfer to/from memory.
- The device controller signals the CPU when the entire transfer is completed, instead of when its local data buffer is full and needs transfer.



Source: Silberschatz et al., Operating System Concepts, 9th Edition

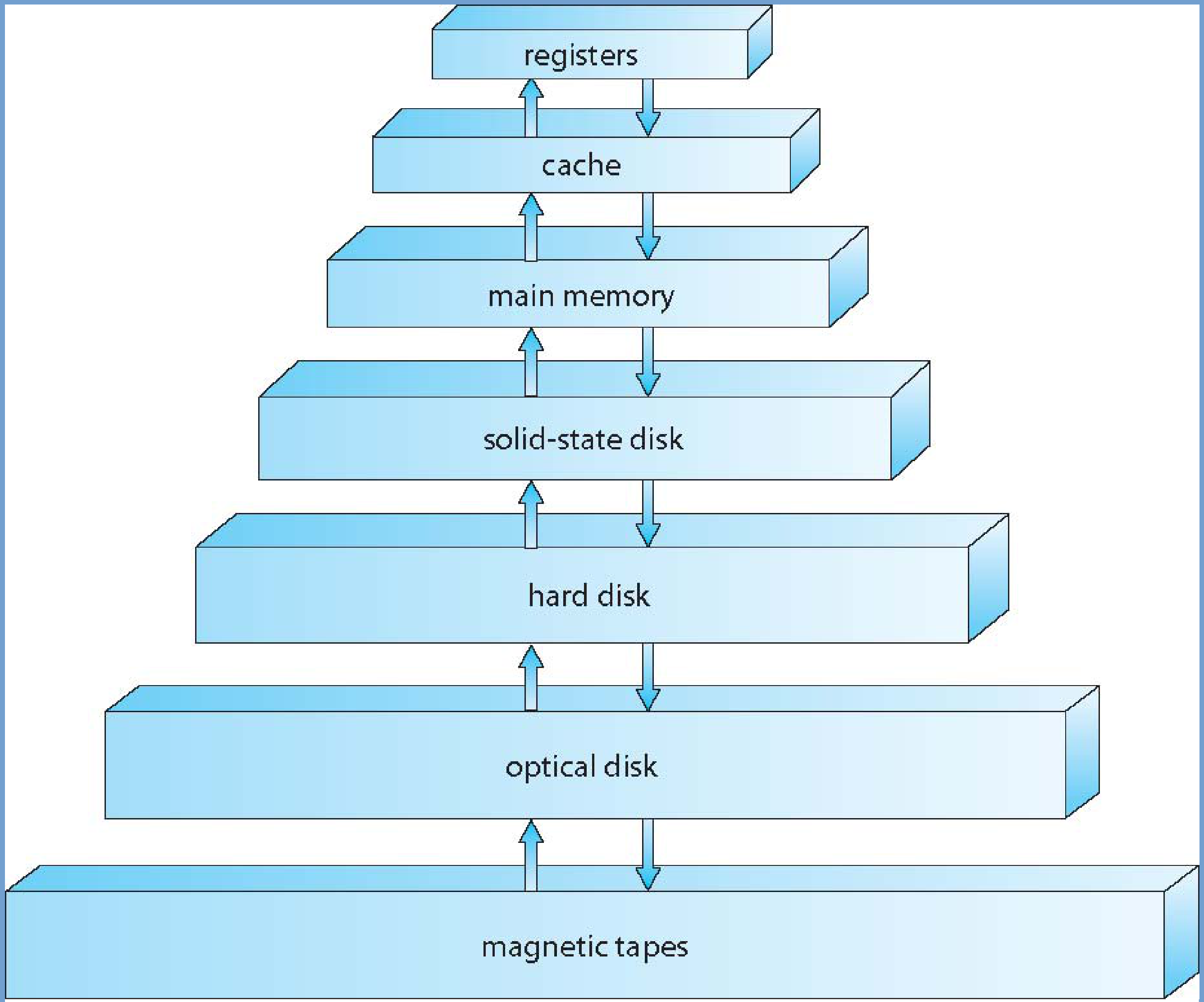
Storage structure

- The CPU can directly access main memory.
  - Load/store instructions. Random access.
  - Important: ***volatile***, switch off power and all contents are lost.
- A *secondary storage* level is present that is non-volatile and has greater capacity.
  - Classically: tape storage.
  - Hard drives: glass platters with magnetic recording material. The platters spin at high RPM and disk heads move.
  - Flash memory & Solid State Drives (SSDs).
    - Faster than hard drives, no moving parts.
    - Various formats / packages.

Storage structure (2)

- Storage systems are organized in a hierarchy:
  - Trade-offs: speed, cost, capacity, volatility.
- Different storage systems have different controllers and require different device drivers.

Storage Device Hierarchy



Source: Silberschatz et al., Operating System Concepts, 9th Edition

Caching

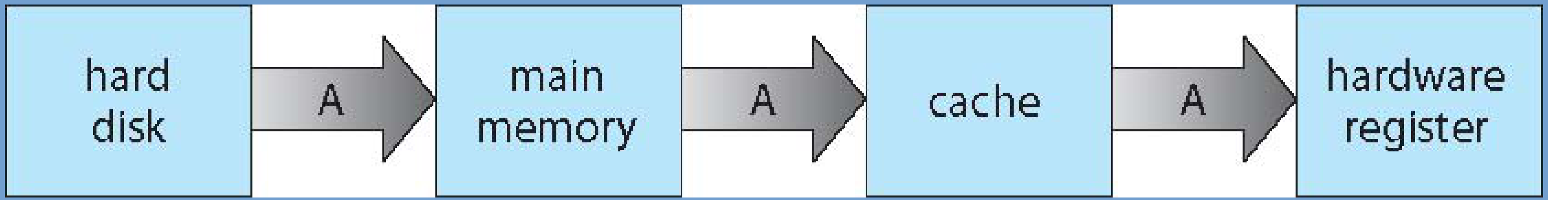
- We already discussed the concept of caching in the Computer Architecture course:
  - The focus was on caching contents of the "slow" RAM.
  - Multiple level cache: L1, L2, L3. Associativity. Inclusive vs. Exclusive.
- Caching is in fact a generic concept:
  - Temporarily store data from a slower storage level in a faster storage level.
    - Often a copy, but as as saw in CA this is not necessarily required.
  - Faster storage level has less capacity, so only part of the slower level can be cached.
  - Therefore, we need policies for management & sizing of the cache and replacement of cache contents.
  - This happens at many, many places in modern computer systems! Not only in hardware, but also in software.

Caching (2)

- Example caches:
  - Disk block cache in main memory (RAM).
  - Web browser cache (caches data retrieved from web server over network).
  - Font cache (cache of all fonts installed on a system)
  - Flash-based cache of hard disk-based RAID array.
- Cache operating like the "RAM cache":
  - First check if requested data is available in the cache.
  - If not, fetch it from the slower storage level.
  - Note: this can in fact trigger a chain of caching! (See later on).

Caching (3)

- An example of data migration from lower to higher storage levels in the cache of disk access:



Source: Silberschatz et al., Operating System Concepts, 9th Edition

- OS requests a disk read: data is transferred from hard disk to buffer cache in main memory (RAM).
- Program performs load/store instruction from/to this data buffer:
  - Data is loaded into a CPU register.
  - Along the way, the data will be cached in CPU cache.

Caching (4)

- Caching can become very complicated:
  - Caching in a multi-core processor: cache coherency (see CA).
  - Software caches accesses by multiple processes.
  - Caches of data stored on another computer over the network (distributed systems).
- Sizing (can be dynamic), when to replace/remove entries, when to write back, etc.

There are only two hard things in Computer Science: cache invalidation and naming things.

-- Phil Karlton

Example Quantification of Different Storage Levels

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 SRAM	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

Source: Silberschatz et al., Operating System Concepts, 9th Edition

Table 2.2 Example Time Scale of System Latencies

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 μs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

Source: *Systems Performance: Enterprise and the Cloud*, Brendan Gregg.

Computer System Architecture

Computer systems can be organized in different ways.

- Single-processor system
  - Only general-purpose CPUs are counted.
  - Becoming harder to come by! All smartphones, laptops, desktops are now multi-core.
- Multi-processor system
  - Choice of multiple "cores" on one chip, or multiple CPUs within a single system (or both!).
- Clustered system
  - Combine multiple computers (nodes) into a single system, interconnect with high-speed network.
  - Require specially written software (parallelized software).

Computer System Architecture (2)

- Multi-processor systems
  - First appeared in server systems dual or quad CPUs on a single motherboard.
  - These days also common in desktops, laptops and smartphones.
  - Typical core counts:
    - *Smartphones*: 2 – 6 cores. Combination of "small" and "large" cores becoming widespread as well (e.g. ARM big.LITTLE).
    - *Laptops*: up to 4 cores.
    - *Desktops*: 4 – 6 cores.
    - *Servers*: up to 20 – 24 cores per CPU, 2 CPUs per server is very common.

Computer System Architecture (3)

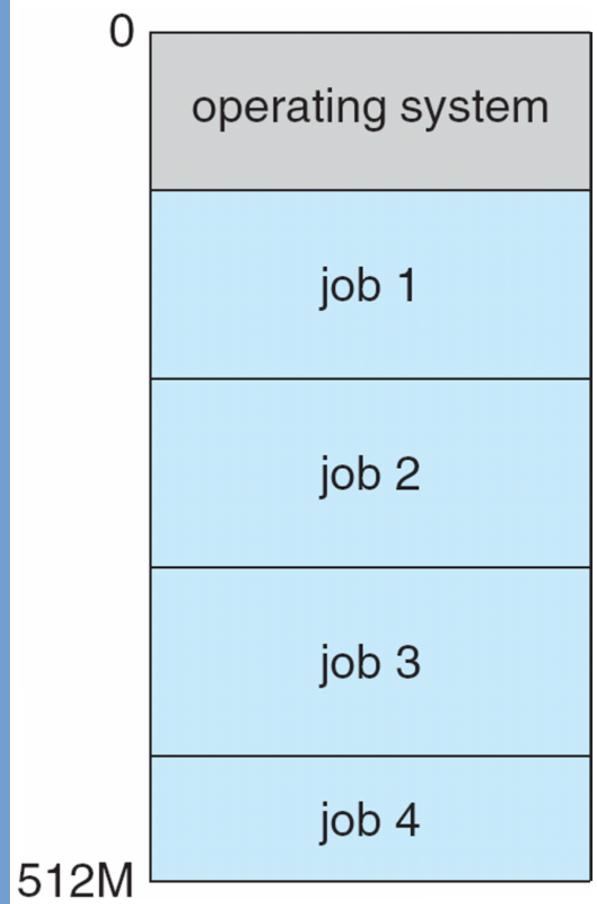
- Why multi-processor systems?
  - **Increased throughput**
    - More cores: do more work in less time.
    - Recall Amdahl's Law! *N* processors does often not result in *N* times speedup.
  - **Economy of scale**
    - One 40-core computer cheaper to acquire and operate than 40 single-core computers.
      - In particular, also think of cooling in data centers!
    - Disks, power supplies, etc. can be shared.
  - **Increased reliability**
    - Some systems can continue operation when one CPU fails. In this case multi-processor systems lead to more reliable systems.
    - Unfortunately not the case for common Intel-based servers ....
- Most common structure: *Symmetric MultiProcessing (SMP)*
  - All CPUs are “equal”: then can all perform the same kind of work.

Operating System Structure

- Very simple Operating Systems only support one user, one program at a time.
  - DOS comes to mind.
  - Only the very basics of an environments in which programs can be executed are provided.
- A single user & single program cannot make efficient use of all available hardware resources.
  - Also think about the time mainframes were used.
  - A single program cannot keep the CPU and all I/O devices busy all the time.

Operating System Structure (2)

- To allow for more efficient use of the hardware, ***multiprogramming*** systems were designed.
  - These were batch systems: there was a queue of jobs that were processed one after the other.
  - Multiprogramming systems keep multiple jobs in memory.
  - A ***job scheduler*** determines the next job to load into memory.
  - When a job blocks on I/O (tape drive) or otherwise, the system can switch to another job. The CPU is kept busy at all times.



Source: Silberschatz et al., Operating System Concepts, 9th Edition

Operating System Structure (3)

- When computers became more powerful, supervisors could interrupt batch processing to run an interactive job (e.g. for debugging).
- This was later further extended to ***timesharing*** (***multitasking***) systems.
  - The CPU rapidly switches jobs, allowing each of them to make progress for a short period of time.
  - Users have the impression the computer is running multiple jobs (tasks) at the same time.
  - Short response times (< 1 second), resulting in multiple ***interactive*** programs running “at the same time”.
  - For instance, mainframes could now execute many different interactive programs for different users. (Think of old-school reservation systems).
  - Swapping & Virtual memory are techniques to be able to deal with process mixes that do not entirely fit in main memory.

Operating System Structure (4)

- Imagine a system with multiple programs active that all have equal & unrestricted access to the hardware resources.
  - Uncoordinated writes to for instance hard drives.
  - Interleaved writes to printers.
  - Active programs can write in each others main memory.
- Clearly, there must be an entity with more privileges that is in control.
  - This is the Operating System kernel.
  - Control and isolation responsibility.

Operating System Organization

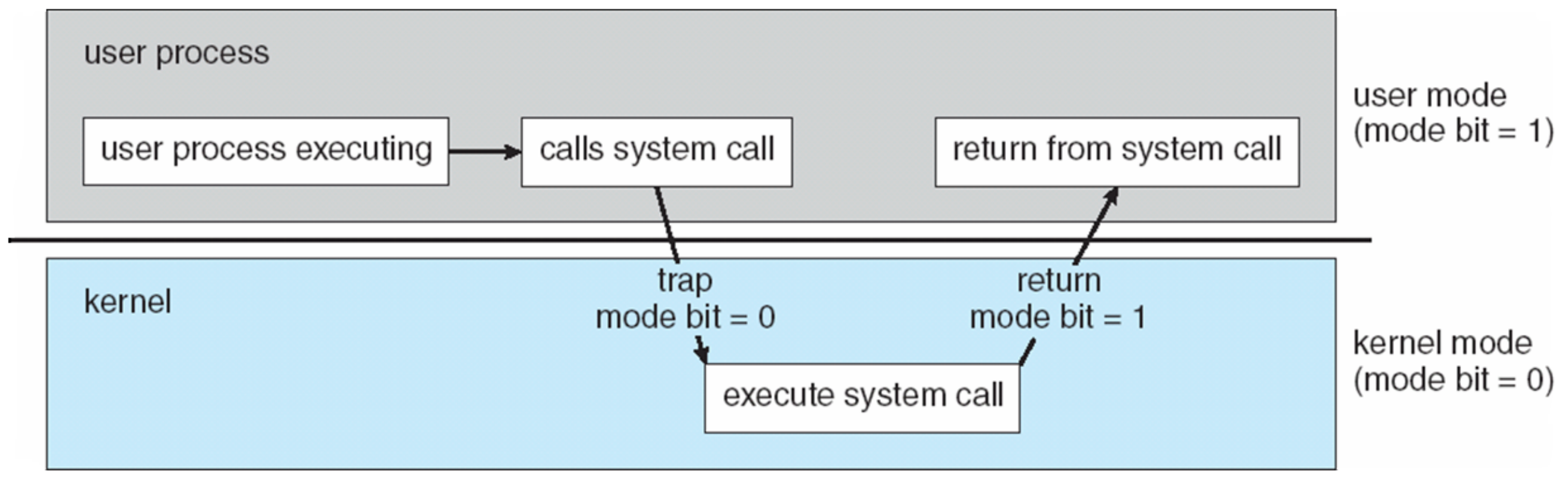
- An Operating System relies on ***dual-mode operation***.
  - *User-mode* and *kernel-mode*.
  - Hardware maintains a ***mode bit*** (often in a system register), which indicates whether the current instruction is executed in user-mode or kernel-mode.
  - ***Privileged instructions*** can only be executed in kernel-mode.
  - So, the kernel is protected from user-mode processes and only the kernel can directly control hardware and control user processes through the privileged instructions.

Operating System Organization (2)

- When the kernel boots, it is running in kernel-mode.
- Once the first instruction of an ordinary program is run, the system switches to user-mode.
- How do we get back into kernel-mode?
  - Recall: interrupt-driven system.
  - *Hardware interrupt*: interrupt service routine is a kernel routine that executes in kernel-mode.
  - *Software interrupt*: because of a fault (division by zero), or ***system call request*** (trap mechanism).
  - When the HW/SW interrupt handler returns, the system switches back to user-mode.

Operating System Organization (3)

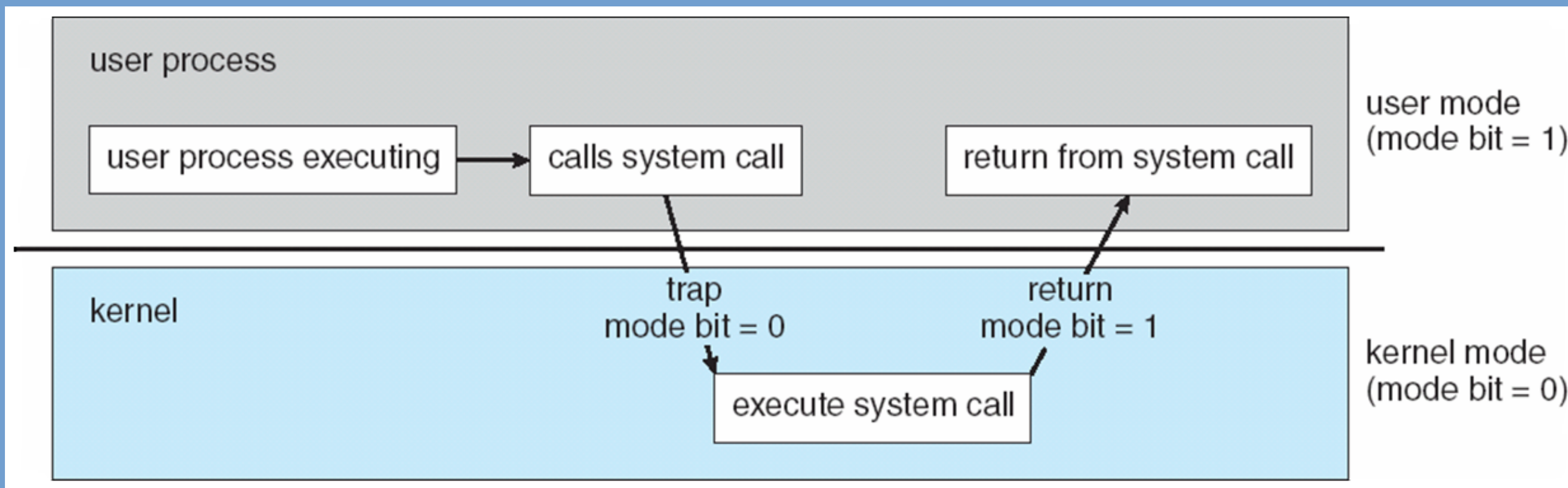
- Schematic overview of system call invocation:



Source: Silberschatz et al., Operating System Concepts, 9th Edition

Operating System Organization (3)

- Schematic overview of system call invocation:



- Assembly example (recall Security course):

- Example from libc:

```
00000000000cb040 <execve>:
   cb040:   mov     $0x3b,%eax
   cb045:   syscall
   cb047:   cmp     $0xfffffffffffff001,%rax
   cb04d:   jae     cb050 <execve+0x10>
   cb04f:   retq
```

Operating System Organization (4)

- Hardware interrupts are important besides software interrupts (traps).
  - What if no program ever performs a system call? But simply executes an infinite loop?
- Modern hardware contains a programmable timer controller.
  - This generates interrupts at set times.
  - The OS kernel is guaranteed to periodically obtain control of the system this way.
  - Clearly, we only want the OS kernel to be able to program this timer controller, hence another reason to have privileged instructions.

Organization System Organization (5)

- Now, a very quick tour of the main components of Operating System kernels:
  - Process Management
  - Memory Management
  - File Management
  - Mass-Storage Management
  - I/O Subsystem
  - Protection & Security

Process Management

- A program is a **passive** entity on a storage device. It consists of simply instructions (code) and initialization data.
- When a program is loaded into memory:
  - a stack and heap are also allocated
  - it will maintain state in CPU registers (most notably the program counter!)
  - may open files and other resources
  - is now an **active** entity that we refer to as a ***process***.

Process Management (2)

- Typical tasks:
  - Creation of processes
  - Termination of processes
  - Suspend / resume of processes (process control)
  - Setting scheduler properties
  - Providing primitives for communication between processes (InterProcess Communication: IPC)

Memory Management

- Programs and associated data must be loaded into main memory for execution.
  - The CPU can only access main memory directly.
- But where?
  - The OS kernel must manage the main memory and keep track of what parts are in use and what parts are free.
  - Processes may request additional memory allocations, these must be handled appropriately.
  - When more memory space is requested than is available, what to do?
    - Perhaps temporarily move processes (or part thereof) out of the main memory onto secondary storage.

File Management

- Data is organized in terms of files and directories.
  - The OS kernel must provide this uniform and logical view.
  - And regardless of actual storage device (hard disk, floppy, SD card, etc.)
- Tasks include:
  - Organization of files into directories / Virtual File System
    - Manipulation thereof: creating/remove files/directories.
  - Access control
  - Mapping VFS onto secondary storage.
    - How are directories, directory entries and metadata encoded on the actual disk?

Mass-Storage Management

- The devices on which files are stored must be managed and operated.
- Non-volatile devices that are used for long term and large-scale data storage.
- As these devices are typically significantly slower than main memory, good datastructures and algorithms are paramount for good performance of the computer system as a whole.
- Tasks:
  - Disk scheduling (disk arm movement)
  - Free-space management & block allocation

I/O subsystem

- Provide a generic interface for accessing I/O devices.
  - For instance, in UNIX “everything is a file”. For every device a file is present in /dev through which it can be accessed.
- Provide a generic device driver interface, such that device drivers can be written and provided by third-parties.
- Provide generic implementations of buffering, caching and spooling.
  - Spooling is the temporal storage of output data for a certain device, before it is sent to that device.
  - Typically done for devices that do not allow random access, such as printers and tape drives.

Protection & Security

- Protection: Controlling access to resources provided by OS.
  - Notion of user IDs and group IDs.
  - These IDs are associated with active processes, files stored on file systems, etc.
  - ID verification is performed when attempting to access a certain resource (access control).
- Security: defending a system from internal and external attacks.
- We do not go into detail in this course: we have a separate course on Security in the bachelor.

Computing Environments

- Operating Systems are used in many different computing environment – a brief overview.
- Traditional:
  - Originally: mainframes with terminals attached
  - '90s: file/print servers with stand-alone PCs connected as clients through an office network.
  - Now: networking & internet ubiquitous (wireless & 4G LTE). Access to interactive web services from desktops, laptops, tablets, smartphones, watches.

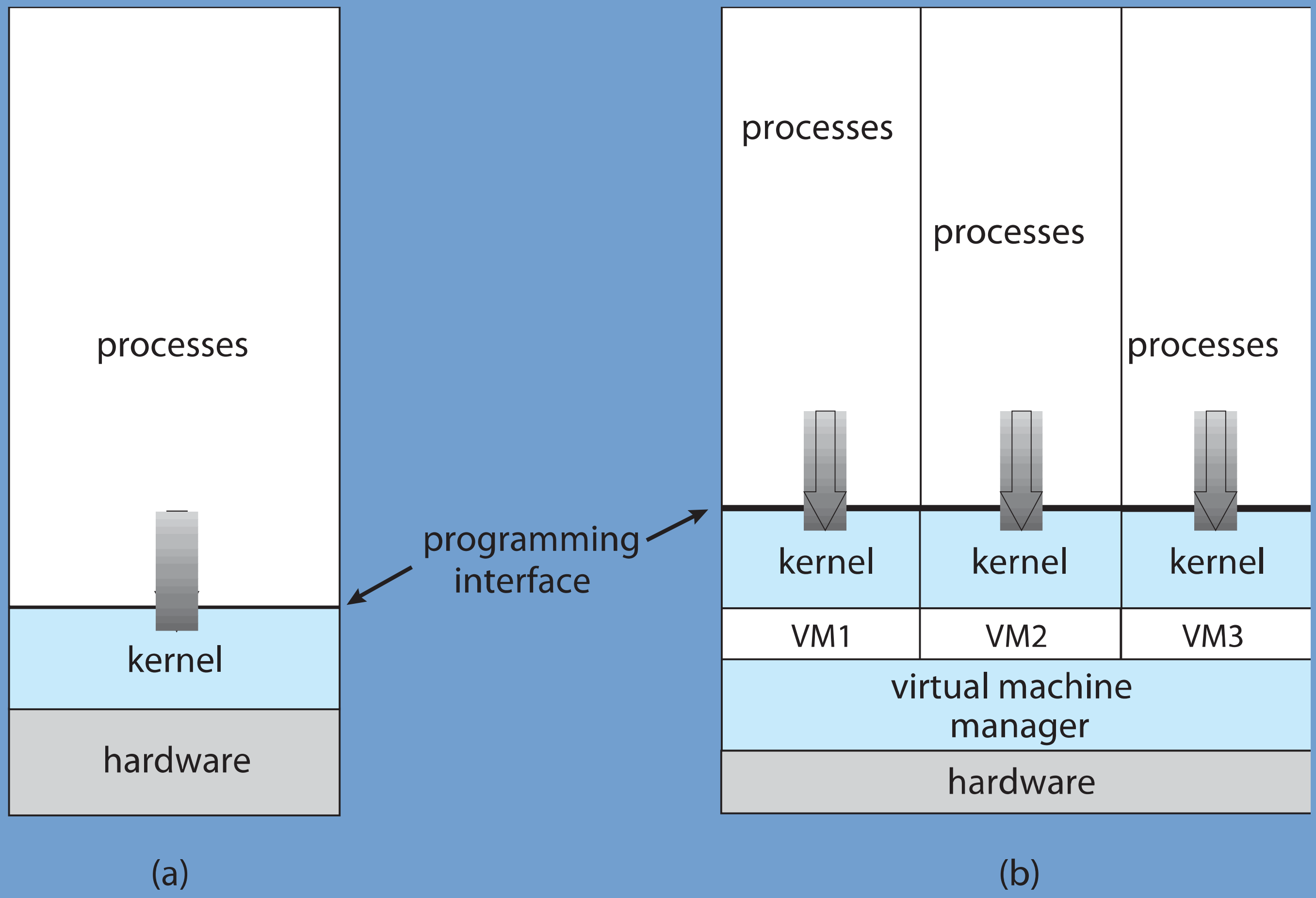
Computing Environments (2)

- Client-Server computing
  - A client connects with a server to request a service.
    - Could be a file or compute service.
  - Servers respond to requests generated by clients.
  - Although this dates back to original office networks, this model is still in use today: websites & in particular web services.
- Distributed computing
  - A collection of (possibly different) machines connected by a network
  - Together, these machines provide services to users.
  - Modern websites invoke many different components running on different machines to complete requests. Microservices.
  - In the past, Network Operating Systems have been developed that combine a collection of computers into a single entity.
  - Peer-to-peer computing: dynamic systems, all nodes are equal.

Computing Environments (3)

- Virtualization: allow an Operating System to be executed as a (guest) process within a host Operating System.
  - Very widely used these days: data centers (consolidation), but also in development: testing & QA.
  - Virtualization services are provided by a Virtual Machine Manager (VMM).
  - Guest Operating System must have been compiled for same architecture. Code is run directly on the CPU.
  - Some systems extend the mode bit mechanism such that guest OS obtains lower privilege level than host OS.
- When the guest OS is compiled for a different architecture than the host, we consider this to be emulation. The instructions of the guest OS must be interpreted to be able to execute it (= slow).

Computing Environments (4)



Source: Silberschatz et al., Operating System Concepts, 9th Edition

Computing Environments (5)

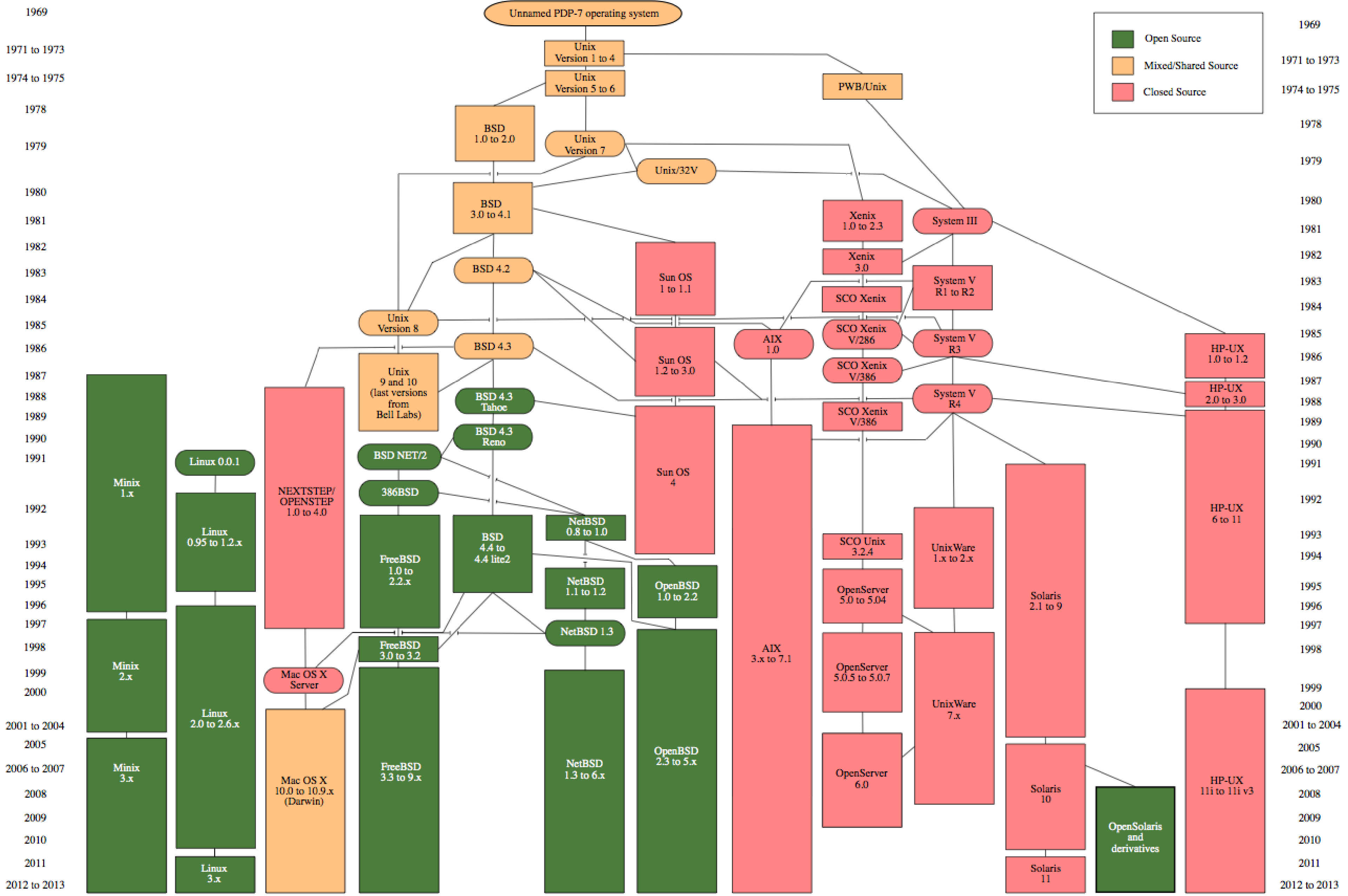
- Compute, storage resources available “through the cloud”, “as a service”.
  - Can (temporarily) rent compute and storage services from vendors. Pay for what you use.
  - Not important where these resources are physically located.
    - (Although, the closer, the lower the latency).
  - Based on virtualization technology.
  - SaaS: Software as a Service: “rent” use of software through a subscription.
  - Vendors: Amazon EC2, Google Cloud Engine, Rackspace, and many many more.

Computing Environment (6)

- Real-time Operating Systems
  - Special purpose and limited OS. Specially designed and tuned for the final application.
  - Often used in embedded devices.
  - Not always a clear UI. Trigger actions in response to sensor data.
    - And these actions must be triggered within a set deadline. This MUST be guaranteed, otherwise the system is not of much use.
  - Very widespread!
  - Think cars, airbag controllers, airplanes, digital TV receivers, ...

Open Source Operating Systems

- Many Operating Systems are "Open Source".
  - Examples: Linux, FreeBSD, OpenBSD, Minix, Haiku, ReactOS, Contiki.
  - The source code of such systems is freely available.
  - Contrary to e.g. Microsoft Windows and parts of macOS.
    - (macOS is in fact based on an Open Source kernel).



Source: http://en.wikipedia.org/wiki/Unix-like

UNIX, Linux, FreeBSD, POSIX, ...

What are the differences between UNIX and Linux?

- First UNIX was developed end of '60, beginning of '70.
- Late '70s/beginning 80's, many derivatives of the original UNIX system appeared: BSD, Solaris, HP-UX, AIX, etc.
- Are these different systems compatible?
- People started to work on standardization:
  - Single UNIX Specification (SUS)
  - POSIX
  - Common definition of SUS and POSIX: Open Group Base specification

UNIX, Linux, FreeBSD, POSIX, … (2)

So when is a system a UNIX?

- Officially, only these systems that are SUS certified (and thus compliant) may be called UNIX systems.
- All others that try to adhere to these standards are "UNIX-like".

Linux, FreeBSD, etc are *UNIX-like* operating systems

- E.g. Linux: implemented from scratch but tries to be fully compliant.
- Linux is just the kernel!
- System utilities typically the GNU utilities.
- (This is why Debian is called GNU/Linux).

UNIX, Linux, FreeBSD, POSIX, … (3)

Practical differences between UNIX and UNIX-like:

- Differences in file system organization. Some things are put at different locations.
- Subtle differences in command line tools. GNU tools often have more options and are easier to use (less picky).
  - For example, BSD “cp” does not allow options to be specified **after** the file names, GNU “cp” does.
- GNU sometimes has extensions to the C library. Such extensions are not available on other systems.

End of Chapter 1.