Operating Systems

David Hay
Dror Feitelson

OS Course Objectives

- To provide a grand tour of the major operating systems components
- To provide coverage of basic computer system organization

Course material

- Prof. Feitelson's notes.
- **Modern Operating Systems**, Tanenbaum
- **Operating Systems**, Stallings
- **Operating Systems Concepts**, Silberschatz, Galvin, Gagne
- **The Design of the UNIX Operating System**, Bach

- **The lectures will cover more than what appears on the slides.**

  Some slides are borrowed from

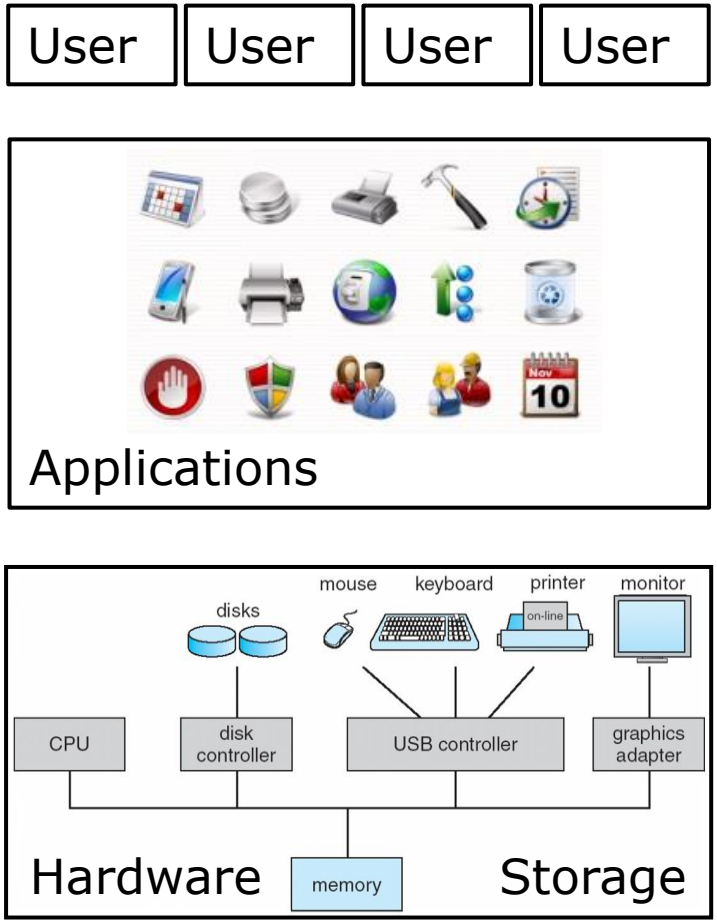
  Anat Bremler-Barr, Danny Hendler, Idit Keidar, Tanenbaum and Silberschatz et. al.

Admins

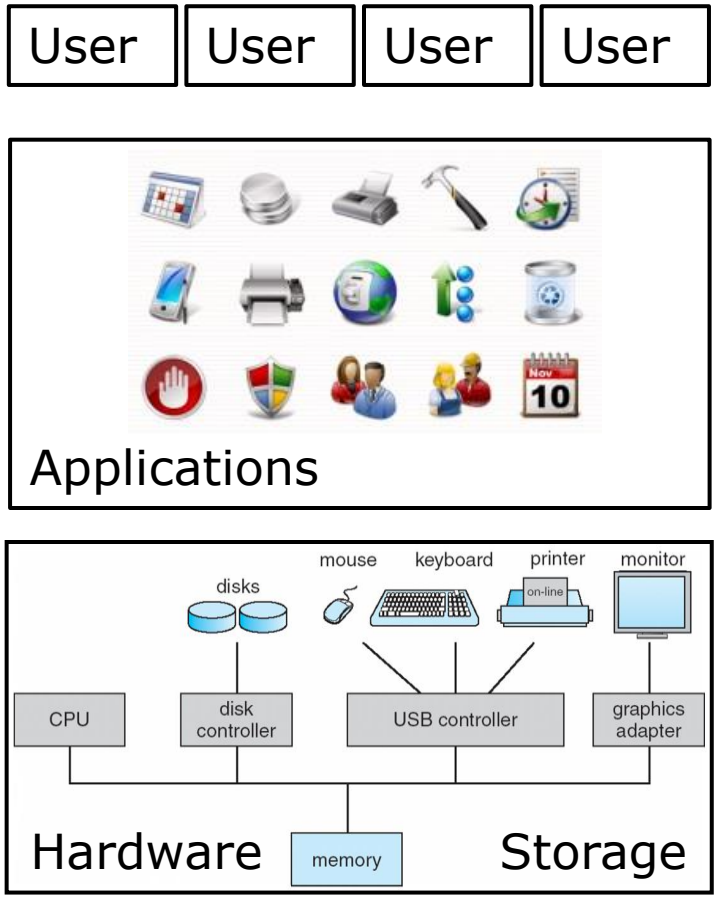
- Final Exam (~60%)
- 5 Exercises (~40%)
  - Should be written in C++
  - Some exercises will have also theoretical questions.
  - Zero tolerance to cheating!!!
- Possible bonus: Weekly Moodle quizzes
- Grade will be calculated by the formulas in the course's website
- <u>Important:</u> You must get at least 60 in the exam itself and at least 60 on the average exercises in order to pass.

What is an Operating System?

- But first, what is a **Computer System**?
  - Users
  - Applications (software programs)
  - Hardware
  - Data

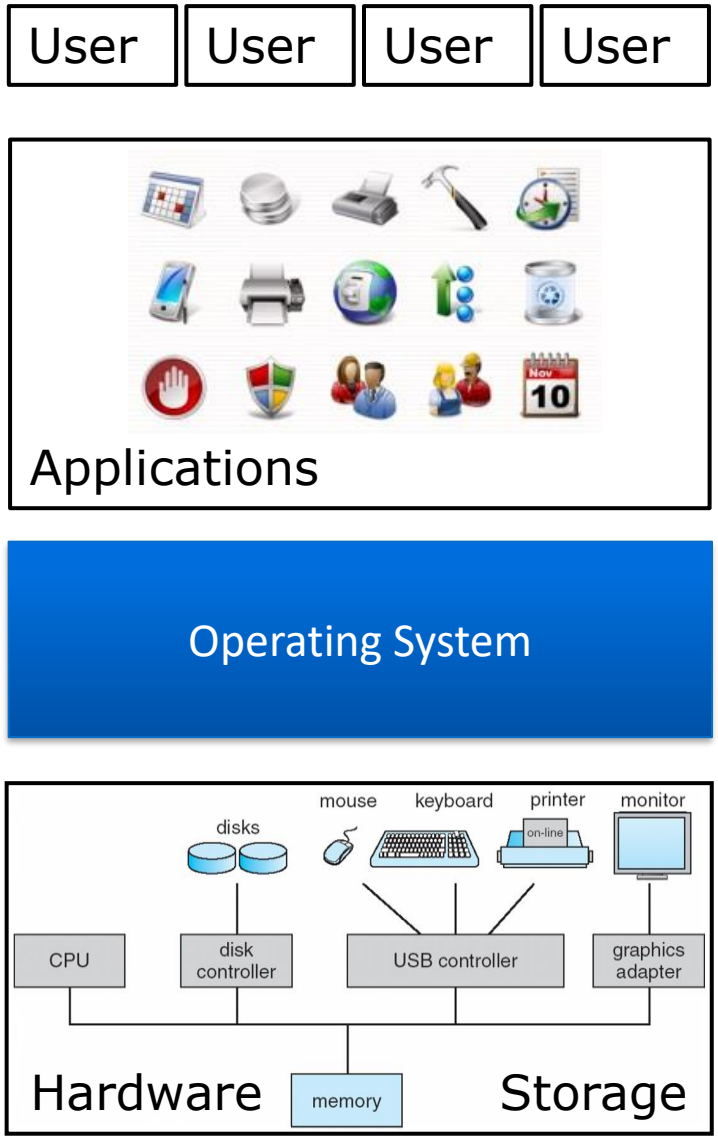


What is an Operating System?



What is an Operating System?

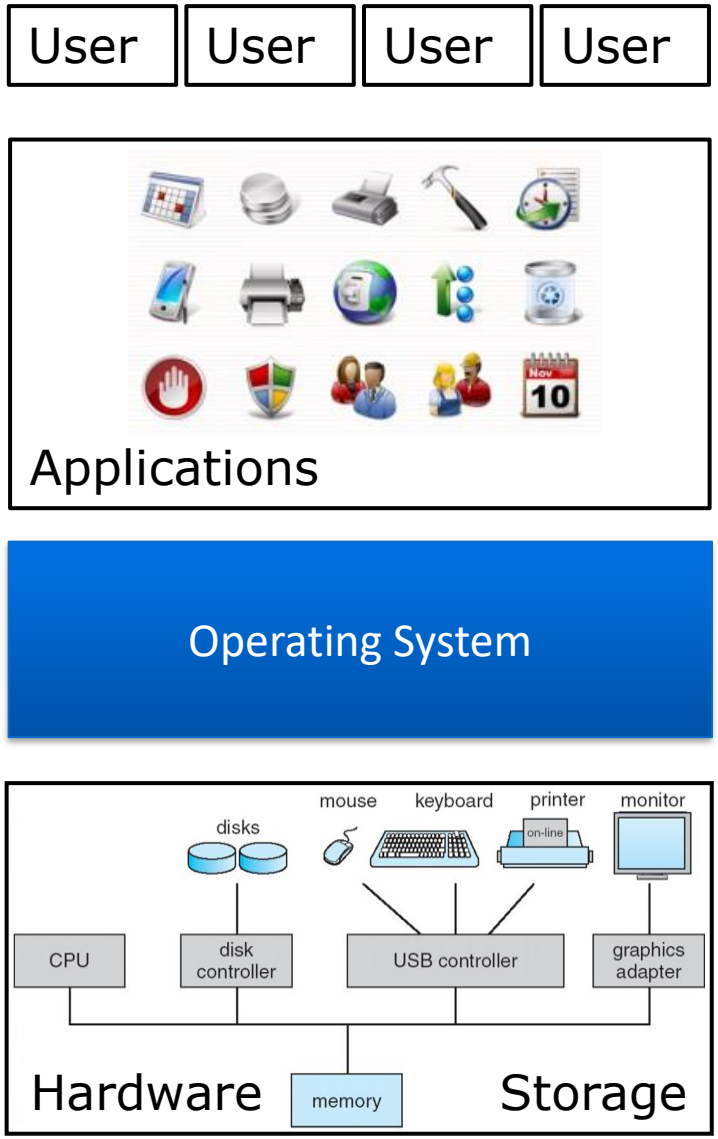
- A **program** that provides an **environment** within which other programs can do useful work
  - Intermediary between the hardware and users
  - No useful work on its on!
  - Reacts to events (reactive program)
  - Always there to help (resident program)



What is an Operating System?

- A **program** that provides an **environment** within which other programs can do useful work
  - Intermediary between the hardware and users
  - No useful work on its on!
  - Reacts to events (reactive program)
  - Always there to help (resident program)

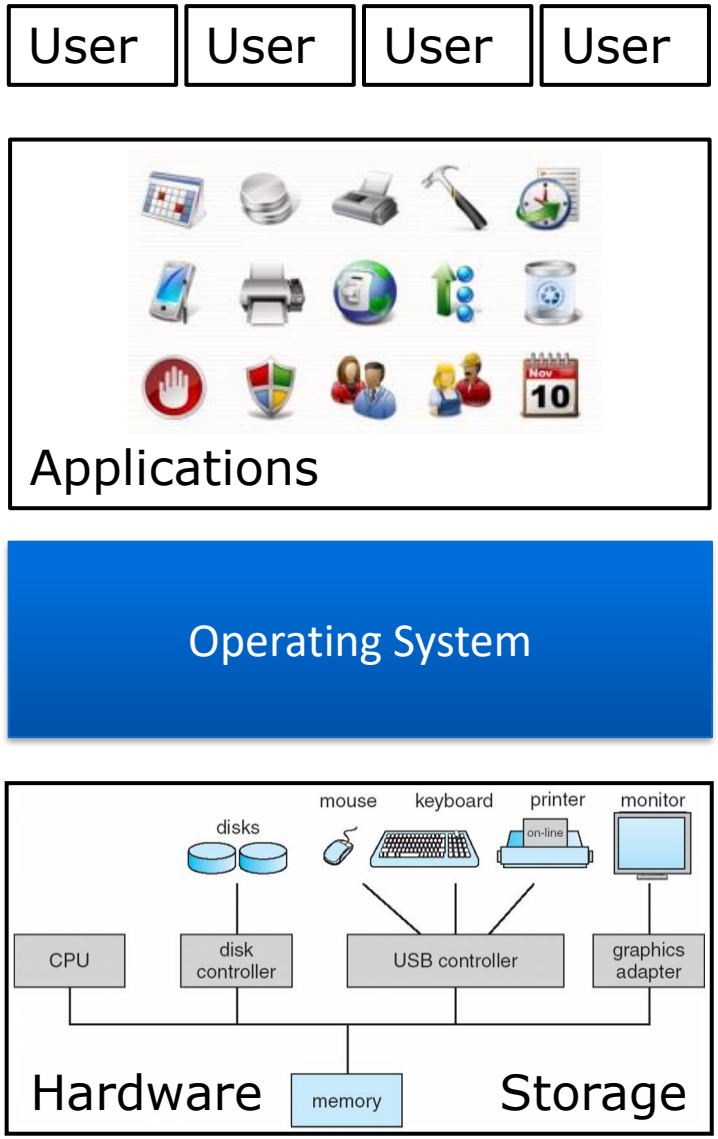
- Make the computer system convenient and safe to use
  - Abstraction, Virtualization



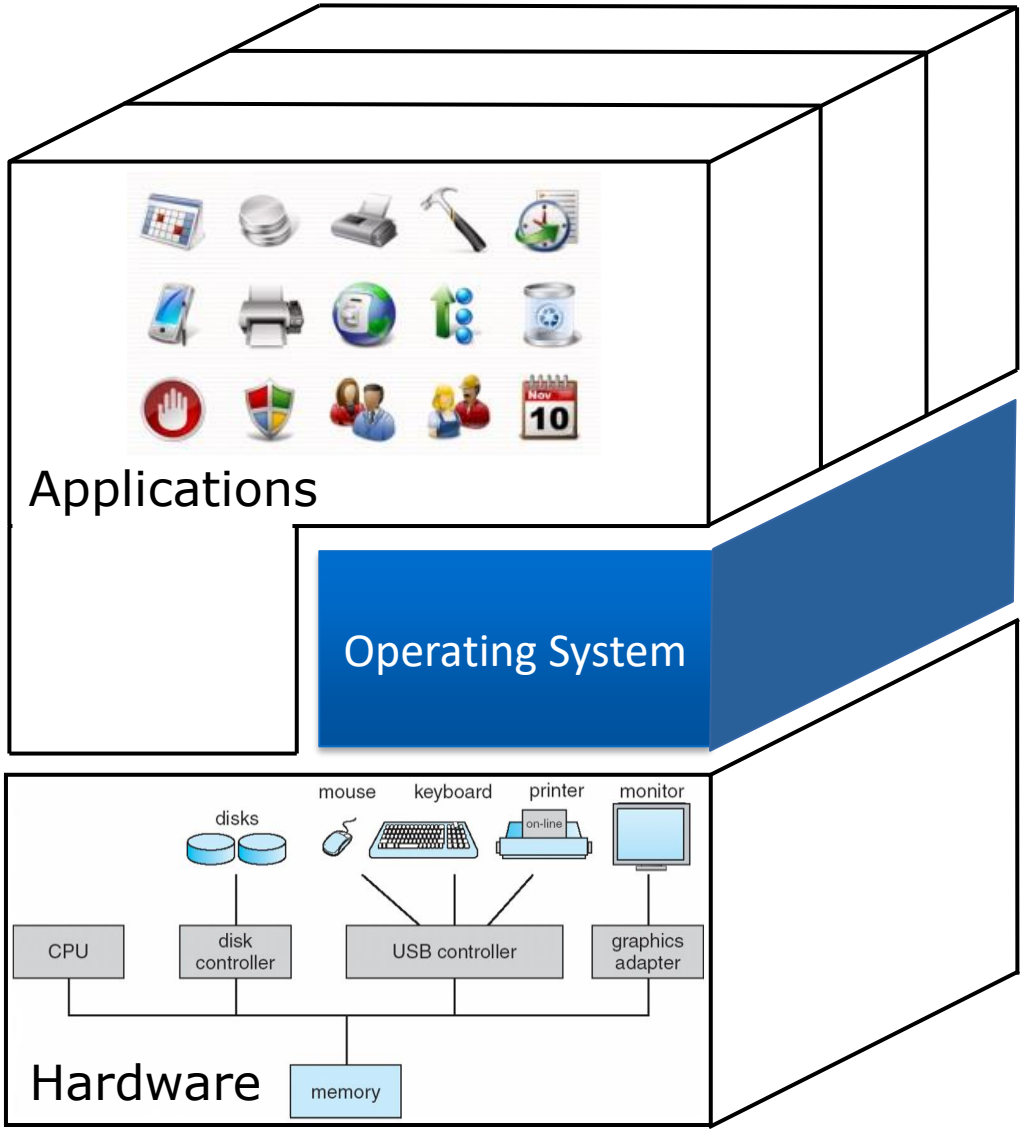
What is an Operating System?

- A **program** that provides an **environment** within which other programs can do useful work
  - Intermediary between the hardware and users
  - No useful work on its on!
  - Reacts to events (reactive program)
  - Always there to help (resident program)

- Make the computer system convenient and safe to use
  - Abstraction, Virtualization
- Use the computer hardware in an efficient manner
  - Resource management

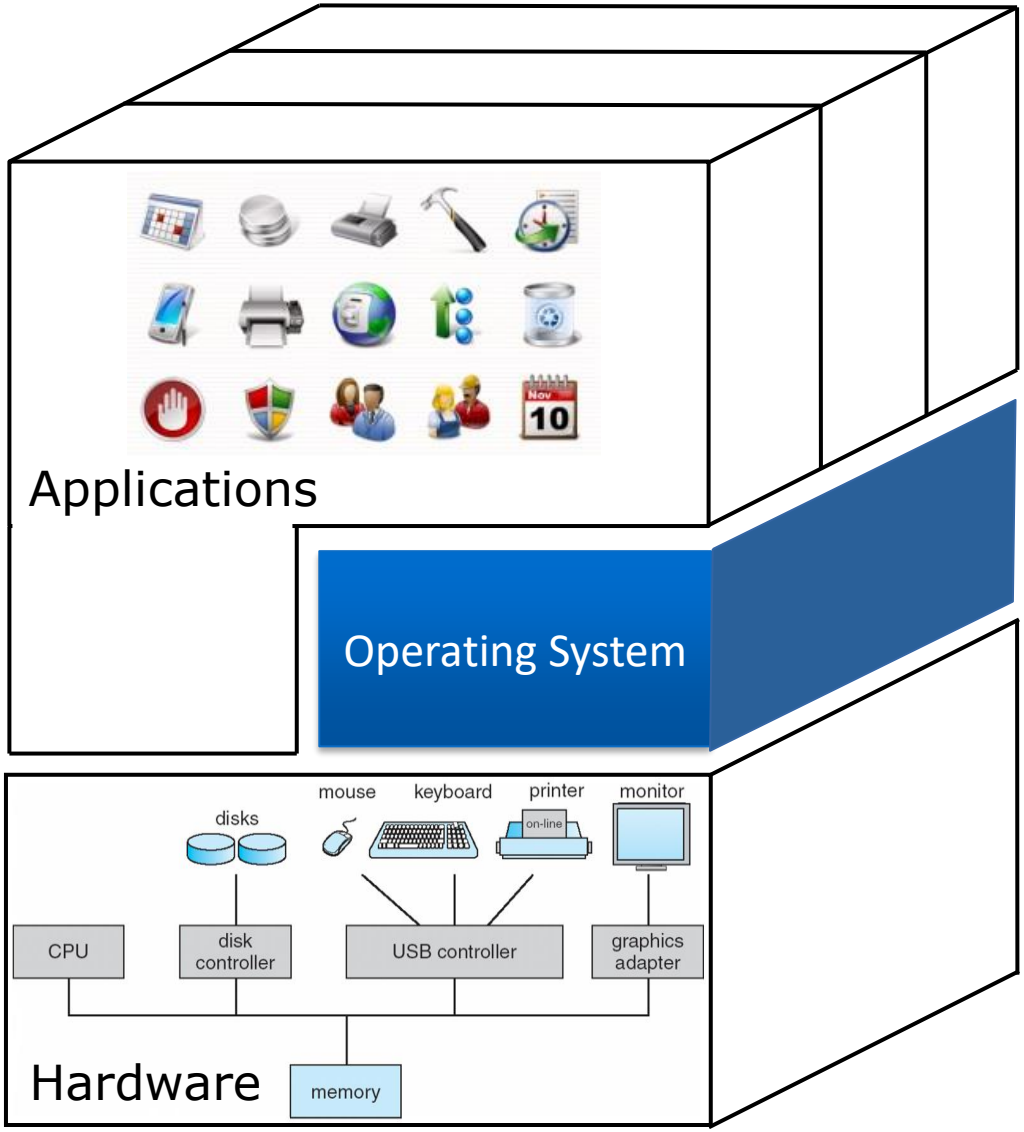


Better Picture



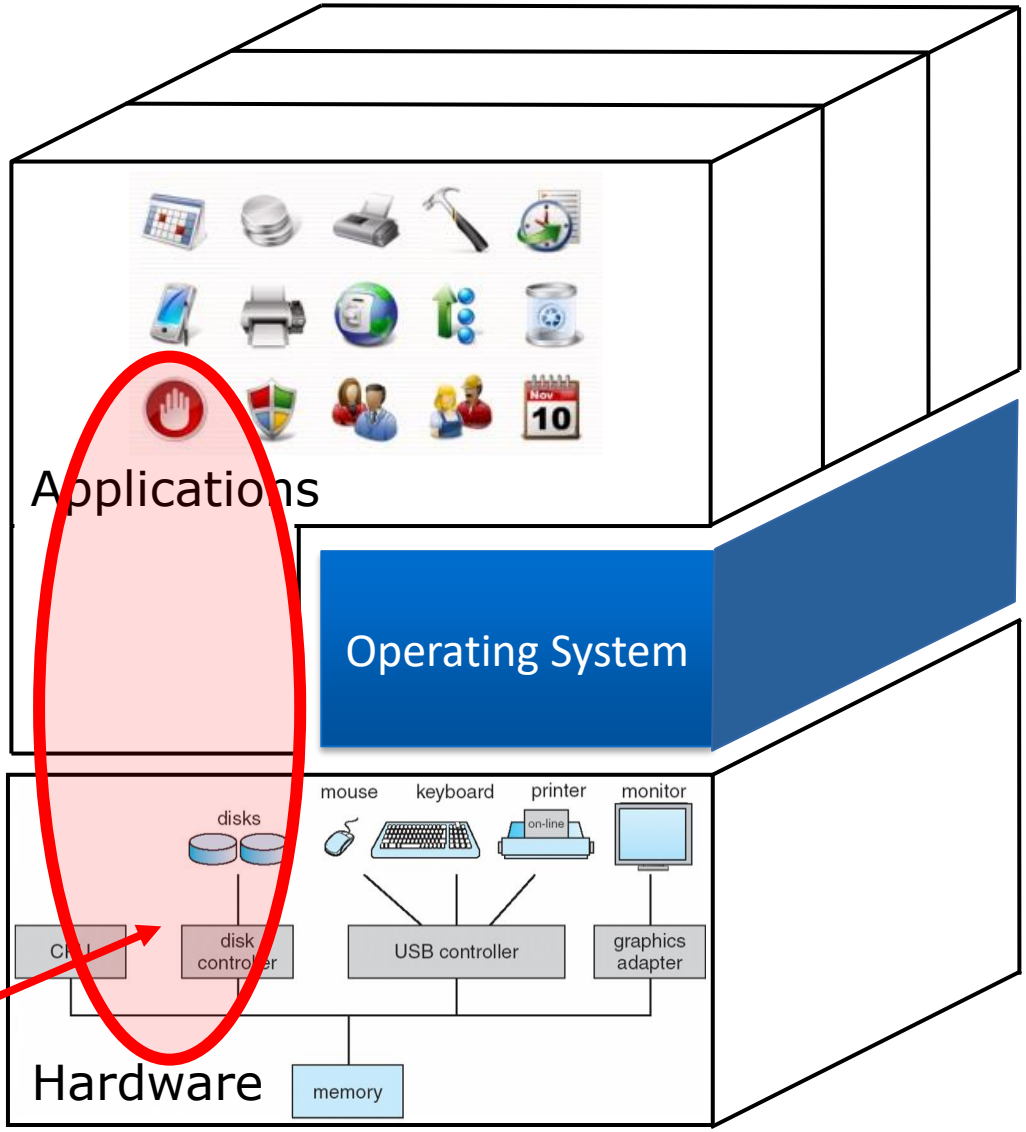
Better Picture

- The OS does not mediate between the applications and the hardware for every operation.
  - Most of the time it doesn't run!
- Applications run directly on the hardware (CPU) and read/write from the memory
  - But cannot directly access, for example, the keyboard



Better Picture

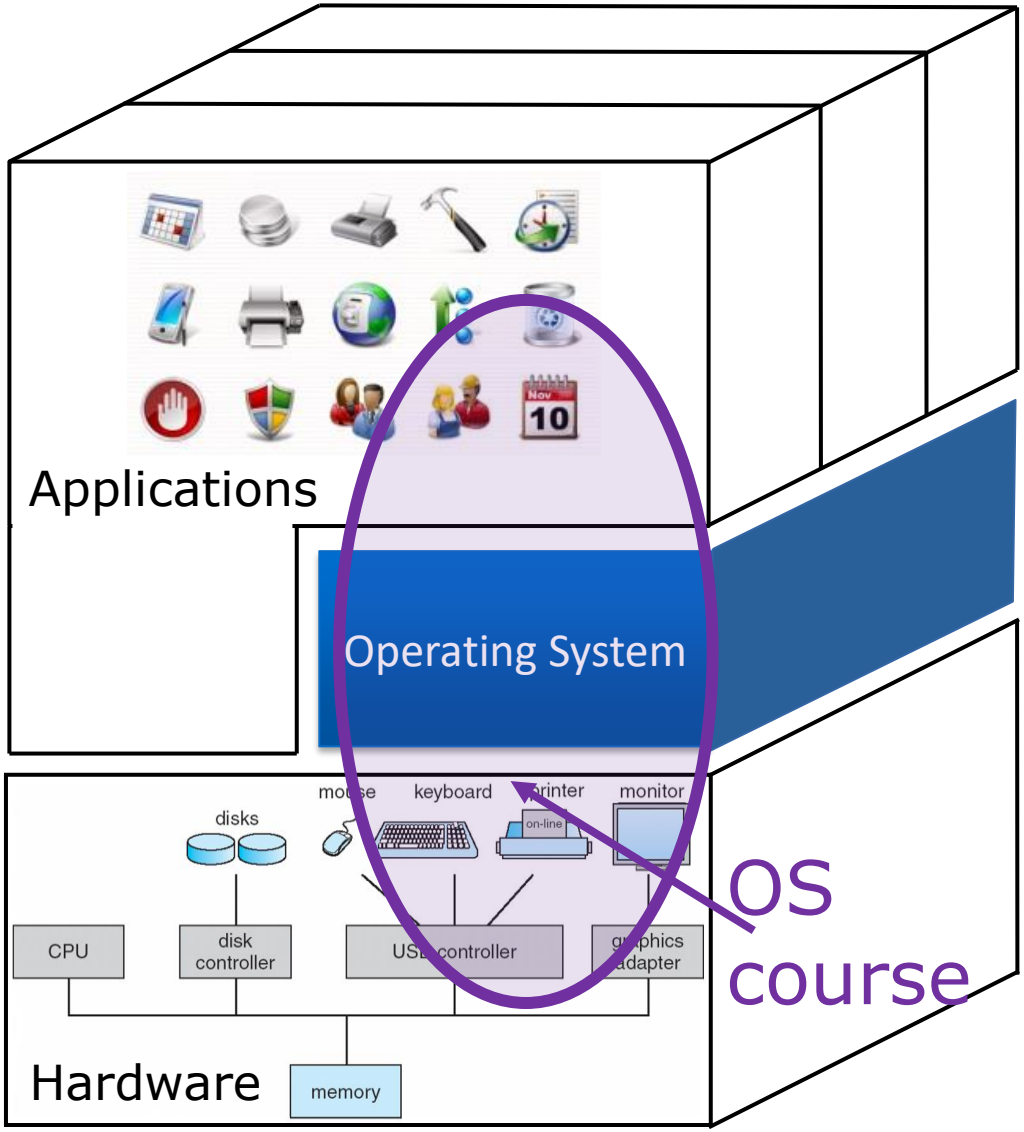
- The OS does not mediate between the applications and the hardware for every operation.
  - Most of the time it doesn't run!
- Applications run directly on the hardware (CPU) and read/write from the memory
  - But cannot directly access, for example, the keyboard



NAND course

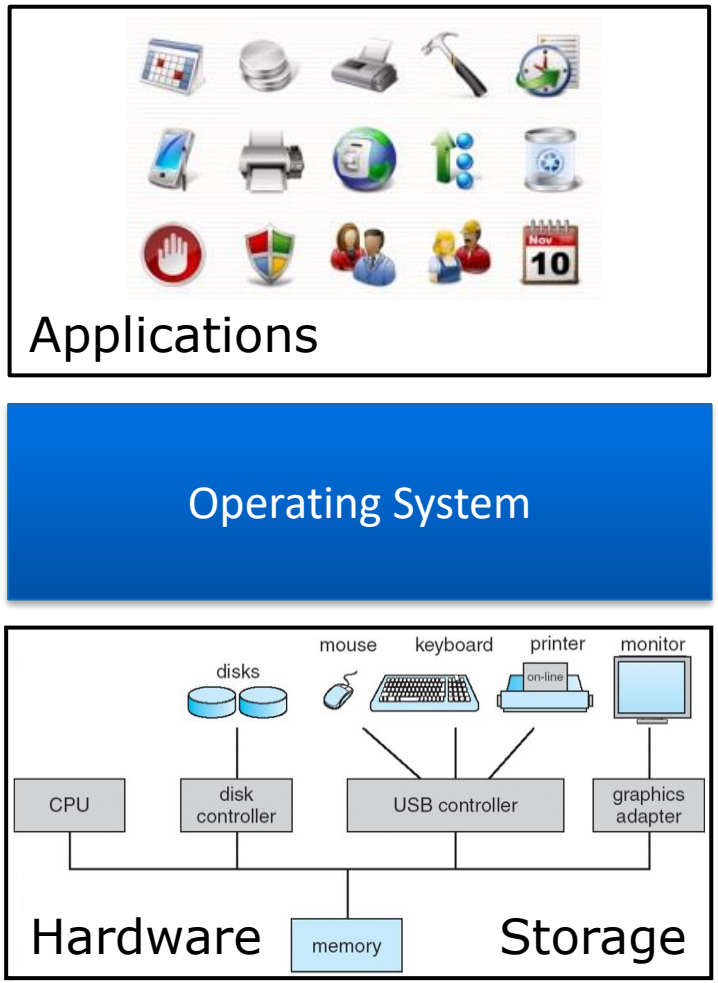
Better Picture

- The OS does not mediate between the applications and the hardware for every operation.
  - Most of the time it doesn't run!
- Applications run directly on the hardware (CPU) and read/write from the memory
  - But cannot directly access, for example, the keyboard



Better Picture

- The OS does not mediate between the applications and the hardware for every operation.
  - Most of the time it doesn't run!
- Applications run directly on the hardware (CPU) and read/write from the memory
  - But cannot directly access, for example, the keyboard



Actually, it is way more complicated

- Sometimes more than one OS is running on the same hardware
  - Virtualization
- Sometimes the OS is running on more than one CPU
  - Multi-core architectures

Life without an OS

- Put your program in the first disk block
- Need help with the mouse, monitors, file systems
- No abstractions!
- Your program is not portable
- Your program controls the hardware without any limitations
- Only one program is running…

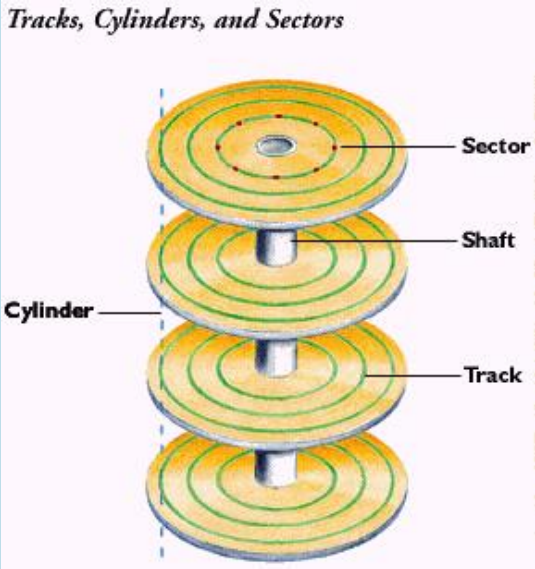
Life without an OS

- Put your program in the first disk block
- Need help with the mouse, monitors, file systems
- No abstractions!
- Your program is not portable
- Your program controls the hardware without any limitations
- Only one program is running…

Instead of:

return-code = read(fd, buf, nbytes)

One should take care of:



Life without an OS

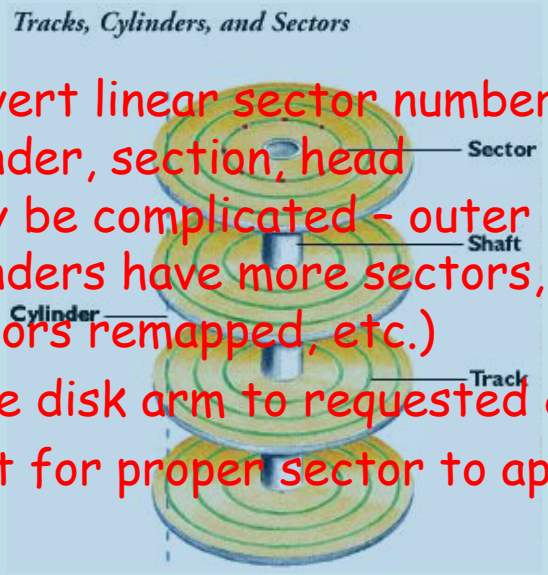
- Put your program in the first disk block
- Need help with the mouse, monitors, file systems
- No abstractions!
- Your program is not portable
- Your program controls the hardware without any limitations
- Only one program is running…

Instead of:

return-code = read(fd, buf, nbytes)

One should take care of:

- …
- Read linear sector 17,403 from disk 2
- Convert linear sector number to: cylinder, section, head (may be complicated – outer cylinders have more sectors, bad sectors remapped, etc.)
- Move disk arm to requested cylinder
- Wait for proper sector to appear
- …



Life without an OS

- Put your program in the first disk block
- Need h
  mous
  systen
- No abstractions.
- Your program is not portable
- Your program controls the hardware without any limitations
- Only one program is running…

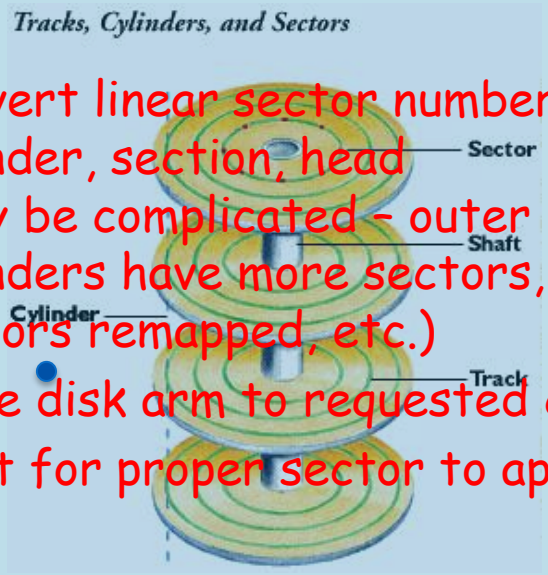
OS provides **abstraction**

Instead of:

return-code = read(fd, buf, nbytes)

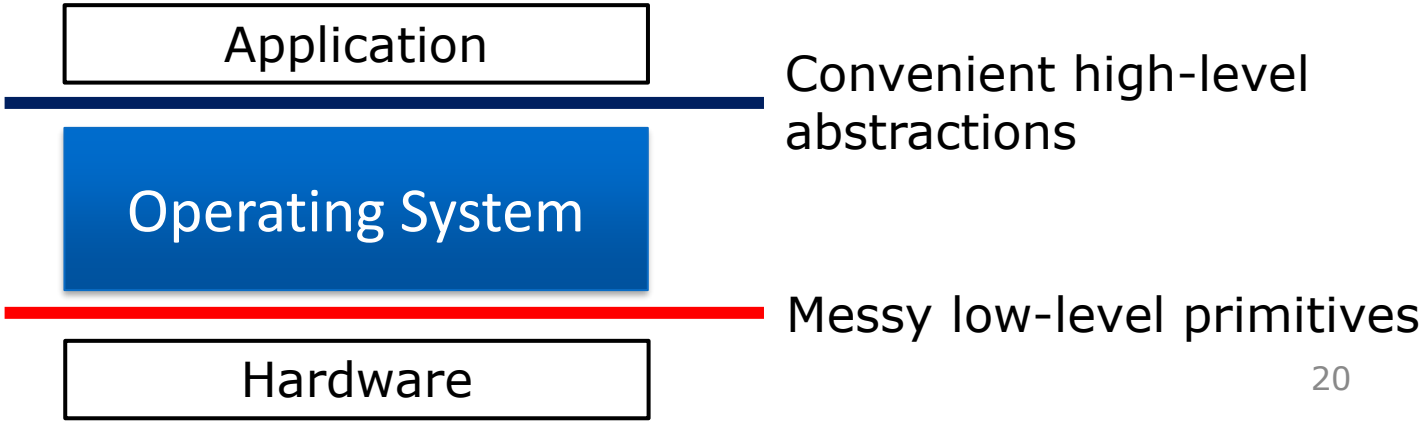
One should take care of:

- …
- Read linear sector 17,403 from disk 2
- Convert linear sector number to: cylinder, section, head (may be complicated – outer cylinders have more sectors, bad sectors remapped, etc.)
- Move disk arm to requested cylinder
- Wait for proper sector to appear
- …



Abstractions

- The OS presents applications with an abstract machine
  - More powerful: includes new abstractions that do not exist in the hardware
  - Simpler: does not include all the complexities the OS has to deal with



Abstraction Example: Files

Hardware:

- Store blocks of data
- Fixed size
- Addressing by location on disk (surface, sector, and track)

Operating system:

- Store data in named files
- Can have arbitrary size
- Addressing by byte offset and length
- Hierarchical directory structure

Abstraction Example: Files

Topics we will touch in the course (for example):
1. What are the abstractions that the OS use?
2. How the abstractionsn are implemented?

ing system:

data in
files

have arbitrary
ize

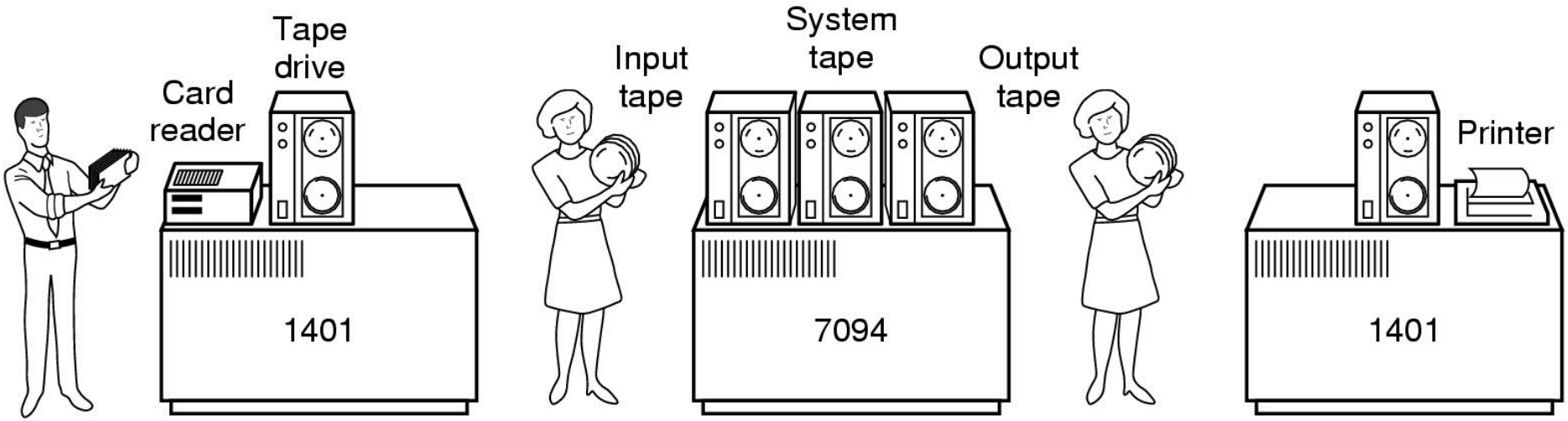
- A
location on disk
(surface, sector,
and track)

- Addre ng by byte
offset and length
- Hierarchical
directory structure

A stroll down memory lane…

- **First generation (1945-1955)**
  - vacuum tubes, plug boards – no operating system
- **Second generation (1955-1965)**
  - transistors, batch systems – multiple programs on disk
  - GMOS for mainframes
- **Third generation (1965-1980)**
  - Minicomputers (usage of Integrated Circuits)
  - multiprogramming/timesharing -*user interaction (time-sharing)*
- **Fourth generation (1980-present)**
  - personal computers – *graphic user-interface*
  - Networks – *file & computing services*
  - Web-computing, *Handheld devices , Cellular phones*
- **Fifth Generation (Present)**
  - Multiprocessor, Multicore
  - Virtualization, Cloud

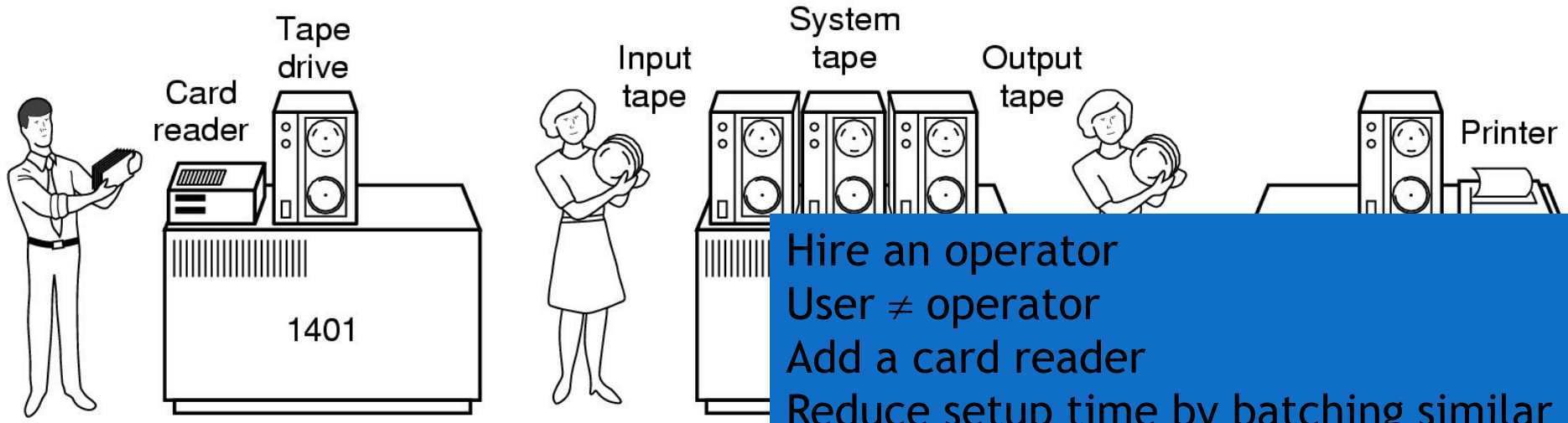
Second generation 1955-1965



Early batch system

1. bring cards to 1401
2. read cards to tape
3. put tape on 7094 which does computing
4. put tape on 1401 which prints output

Second generation 1955-1965



Hire an operator
User ≠ operator
Add a card reader
Reduce setup time by batching similar jobs
Automatic job sequencing – automatically transfers control from one job to another.
Resident monitor
- initial control in monitor
- control transfers to job
- when job completes control transfers back to monitor

Early batch system

1. bring cards to 1401
2. read cards to tape
3. put tape on 7094 which does computing
4. put tape on 1401 which prints output

Spooling: Simultaneous Peripheral Operation On-Line

- Overlap I/O of one job with computation of another job
- While executing one job, the OS:
  - Reads next job from card reader into a storage area on the disk (job queue)
  - Outputs printout of previous job from disk to printer
- Job pool – data structure that allows the OS to select which job to run next in order to increase CPU utilization
- Concept still is use today: print spooling

Performance Terminology

Performance Terminology

- **Latency/Response time זמן השהייה/תגובה:** The *time* it takes to perform a job or process a data.
  - Units: seconds

Performance Terminology

- **Latency/Response time זמן השהייה/תגובה:** The *time* it takes to perform a job or process a data.
  - Units: seconds
- **Throughput תפוקה:** The *rate* of performing work
  - Example: The amount of data processed in a time unit. Units: byte/second. In general: “something per second”

Performance Terminology

- **Latency/Response time זמן השהייה/תגובה:** The *time* it takes to perform a job or process a data.
  - Units: seconds
- **Throughput תפוקה:** The *rate* of performing work
  - Example: The amount of data processed in a time unit. Units: byte/second. In general: "something per second"
- **Utilization ניצולת:** The *fraction* of time the CPU is performing useful tasks
  - Namely, the time the CPU is not idle

Performance Terminology

- **Latency/Response time זמן השהייה/תגובה:** The *time* it takes to perform a job or process a data.
  - Units: seconds
- **Throughput תפוקה:** The *rate* of performing work
  - Example: The amount of data processed in a time unit. Units: byte/second. In general: "something per second"
- **Utilization ניצולת:** The *fraction* of time the CPU is performing useful tasks
  - Namely, the time the CPU is not idle
- **Overhead תקורה:** *Excess* or *indirect* resources that are <u>required</u> to attain a particular goal (usually as a fraction)
  - computation time, memory, bandwidth, or other

Performance Terminology

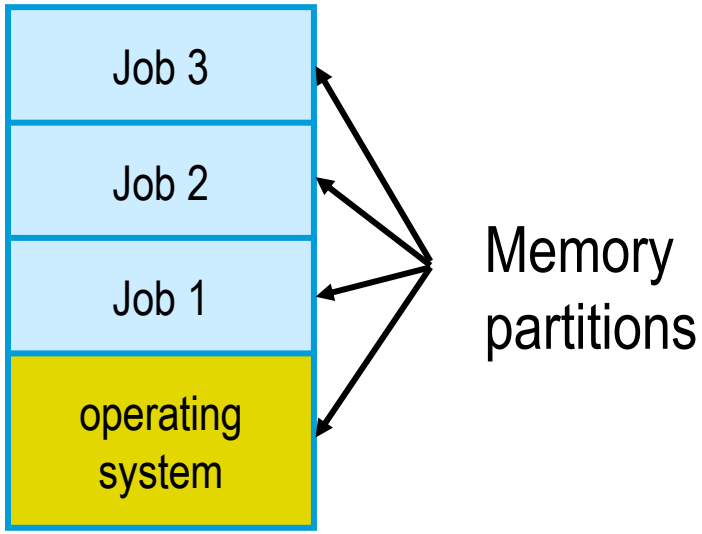
- **Latency/Response time זמן השהייה/תגובה:** The *time* it takes to perform a job or process a data.
  - Units: seconds
- **Throughput תפוקה:** The *rate* of performing work
  - Example: The amount of data processed in a time unit. Units: byte/second. In general: "something per second"
- **Utilization ניצולת:** The *fraction* of time the CPU is performing useful tasks
  - Namely, the time the CPU is not idle
- **Overhead תקורה:** *Excess* or *indirect* resources that are <u>required</u> to attain a particular goal (usually as a fraction)
  - computation time, memory, bandwidth, or other
- **CPU Usage:** The *fraction* of time the CPU is executing instructions
  - Utilization + Time overhead = CPU Usage

Performance Terminology

- **Latency/Response time זמן השהייה/תגובה:** The *time* it takes to perform a job or process a data.
  - Units: seconds
- **Throughput תפוקה:** The *rate* of performing work
  - Example: The amount of data processed in a time unit. Units: byte/second. In general: "something per second"
- **Utilization ניצולת:** The *fraction* of time the CPU is performing useful tasks
  - Namely, the time the CPU is not idle
- **Overhead תקורה:** *Excess* or *indirect* resources that are <u>required</u> to attain a particular goal (usually as a fraction)
  - computation time, memory, bandwidth, or other
- **CPU Usage:** The *fraction* of time the CPU is executing instructions
  - Utilization + Time overhead = CPU Usage

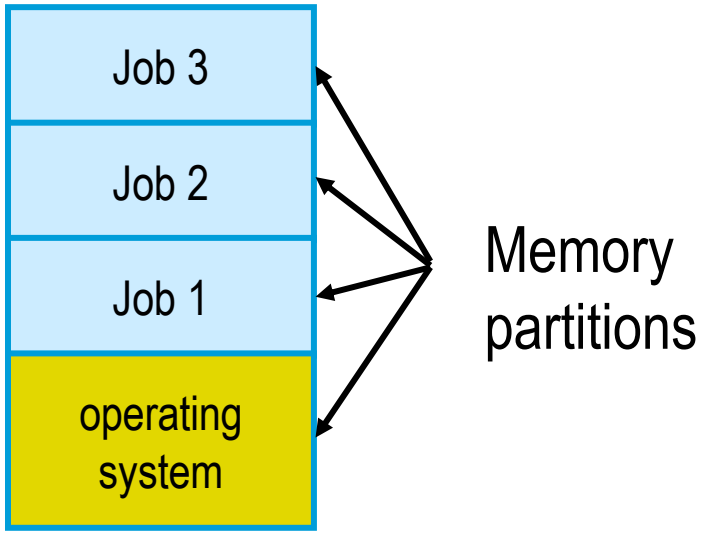
Third generation 1965–1980: Multiprogramming/Time sharing

- Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.



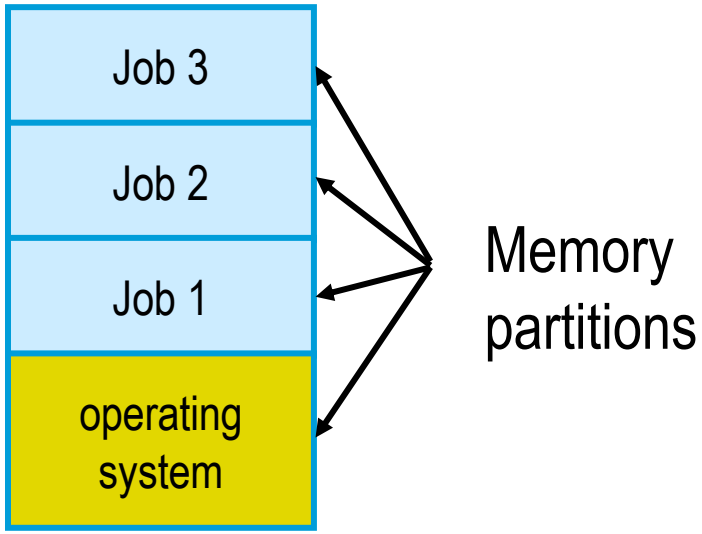
Third generation 1965–1980: Multiprogramming/Time sharing

- Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.
- **Multi-programming:** when the first program reached an instruction waiting for a peripheral, the second program in memory is given a chance to run



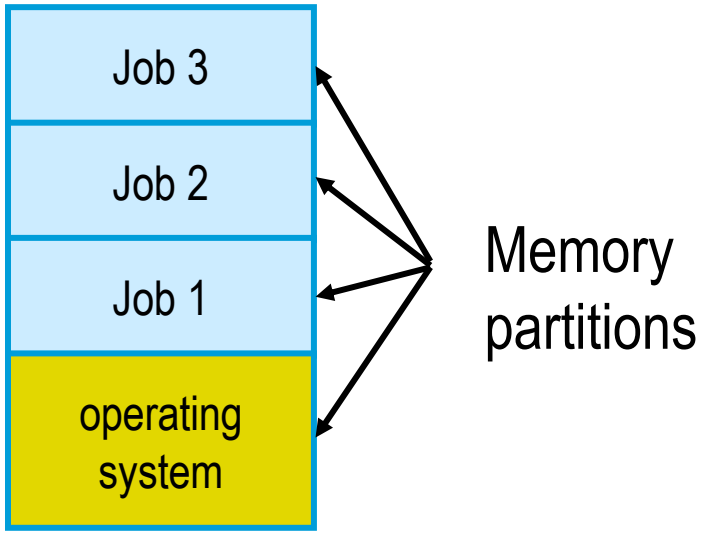
Third generation 1965–1980: Multiprogramming/Time sharing

- Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.
- **Multi-programming:** when the first program reached an instruction waiting for a peripheral, the second program in memory is given a chance to run
- **Time Sharing:** switch program even if not waiting for peripheral



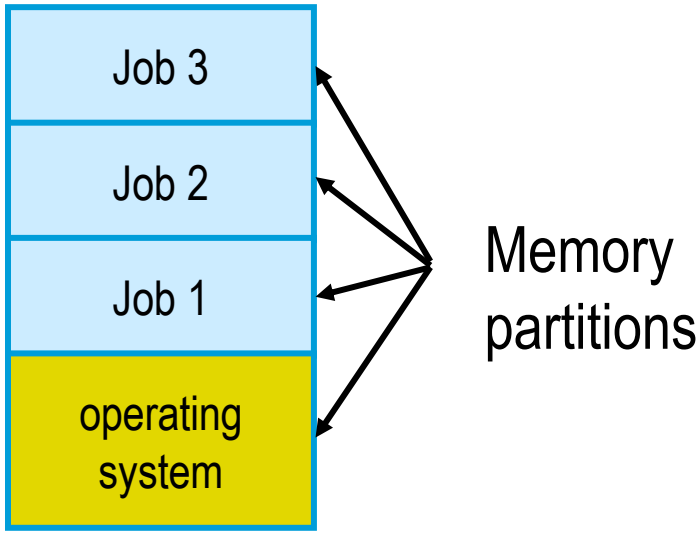
Third generation 1965–1980: Multiprogramming/Time sharing

- Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.
- **Multi-programming:** when the first program reached an instruction waiting for a peripheral, the second program in memory is given a chance to run
- **Time Sharing:** switch program even if not waiting for peripheral
- **Goals:** maximize resource utilization, good response time



Third generation 1965–1980: Multiprogramming/Time sharing

- Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.
- **Multi-programming:** when the first program reached an instruction waiting for a peripheral, the second program in memory is given a chance to run
- **Time Sharing:** switch program even if not waiting for peripheral
- **Goals:** maximize resource utilization, good response time



Time sharing is sometimes called **Multitasking** or **time-slicing**

Multi-programming/Time-sharing

Assumptions

- CPU is always there, and therefore, not using it is a waste
- CPU is much faster than I/O
- Memory is large enough to host many programs
- The peripheral device can access the memory directly (DMA)
- More than one task to do at a time

Multi-programming/Time-sharing

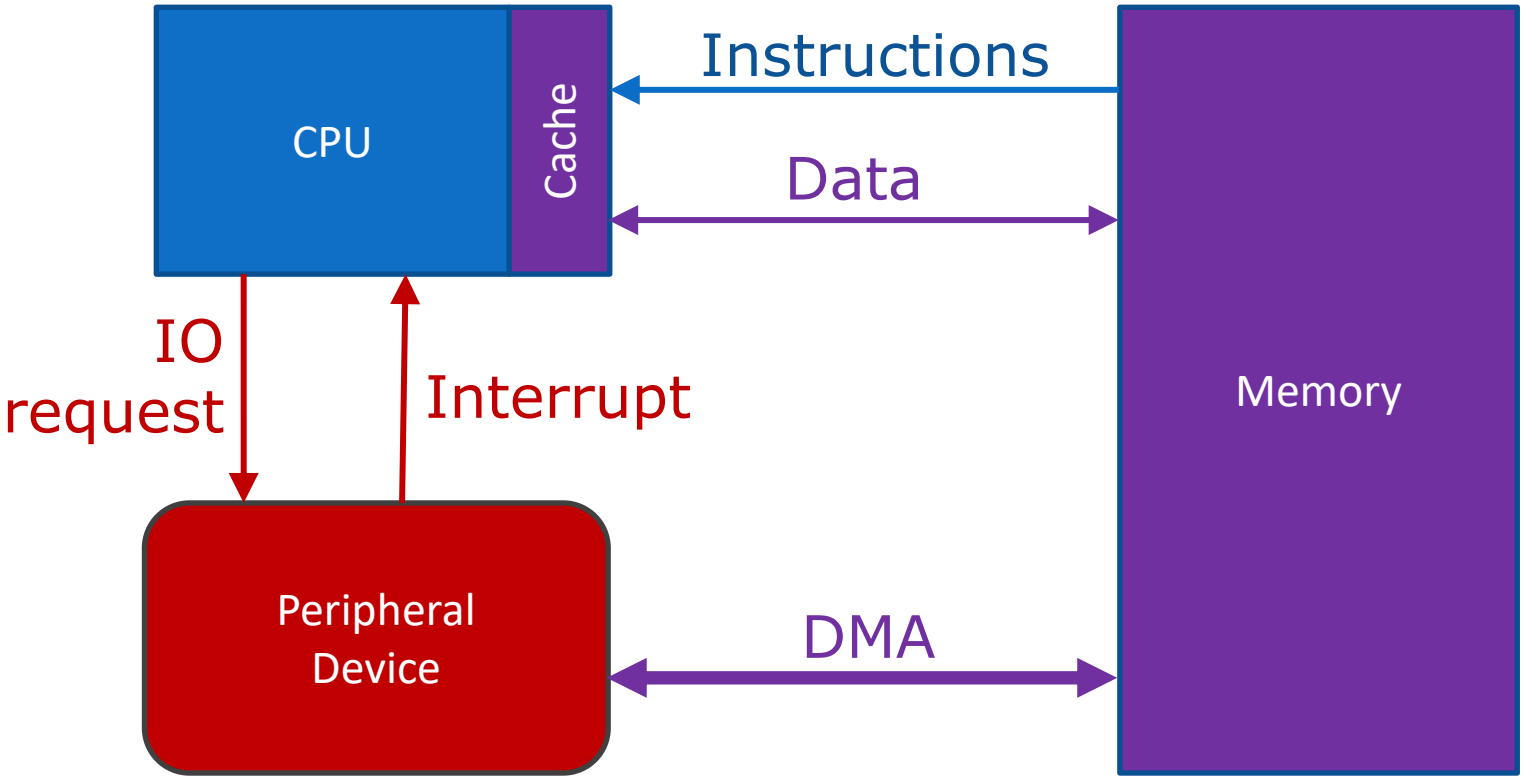
Assumptions

- CPU is always there, and therefore, not using it is a waste
- CPU is much faster than I/O
- Memory is large enough to host many programs
- The peripheral device can access the memory directly (DMA)
- More than one task to do at a time

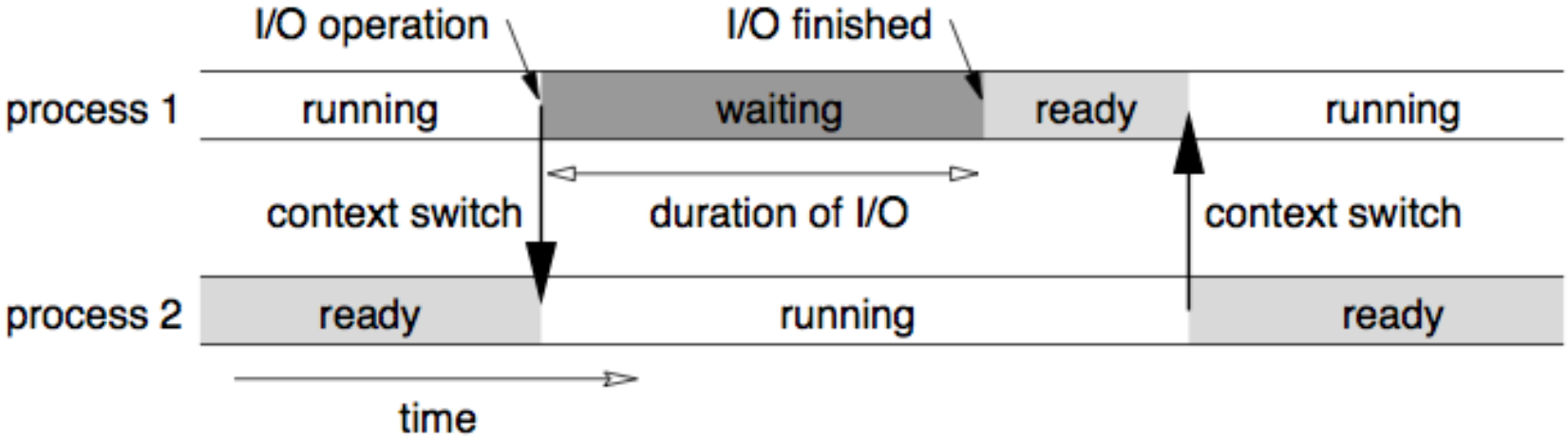
Required OS Features

- I/O primitives
- Memory management
  - Memory protection
- Scheduler
  - Manages flow of jobs in and out of the CPU
- Spooler
  - Manages slower I/O devices (e.g. printer Spooler)

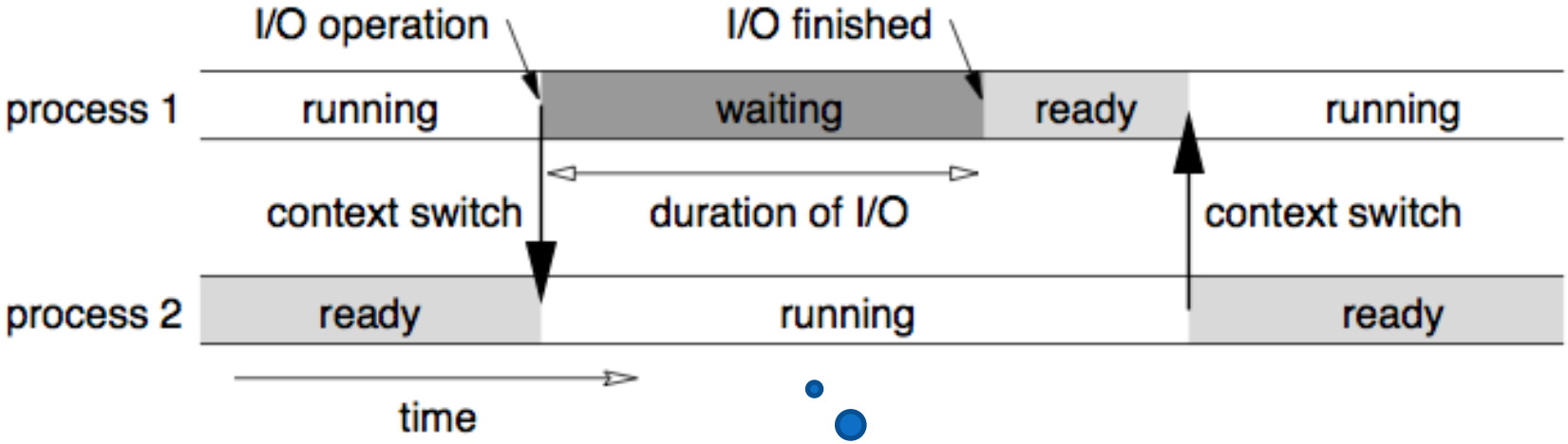
Single Process Architecture (von-Neumann Architecture)



Multiprogramming/Time Sharing: Pictorial Flow

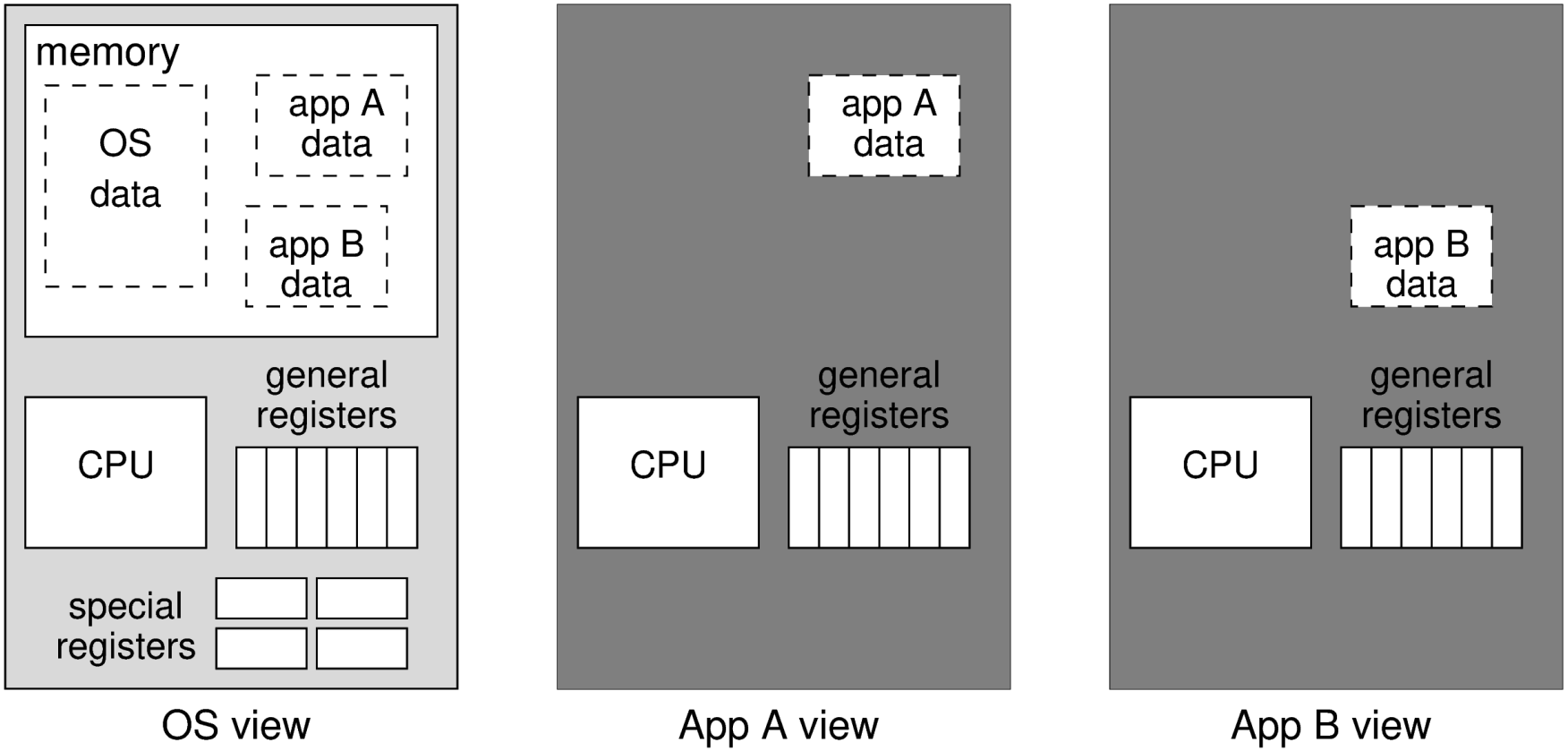


Multiprogramming/Time Sharing: Pictorial Flow



BTW, where is the OS?

OS Provides Isolation



OS view

App A view

App B view

OS Provides Resource Management

- The OS allocates resources
  - Decides who gets to run on the CPU
  - Decides who gets frames of memory
- Can use different considerations:
  - Efficiency
  - Performance
  - Fairness

Main Components of today's OS

- Program Execution
- Process management
- Memory management
- I/O and device drivers
- File System
- Command interpreter/ Shell / (G)UI
- Networking
- (Protection and Security; Error Detection; Accounting; Resource Allocation)

Today

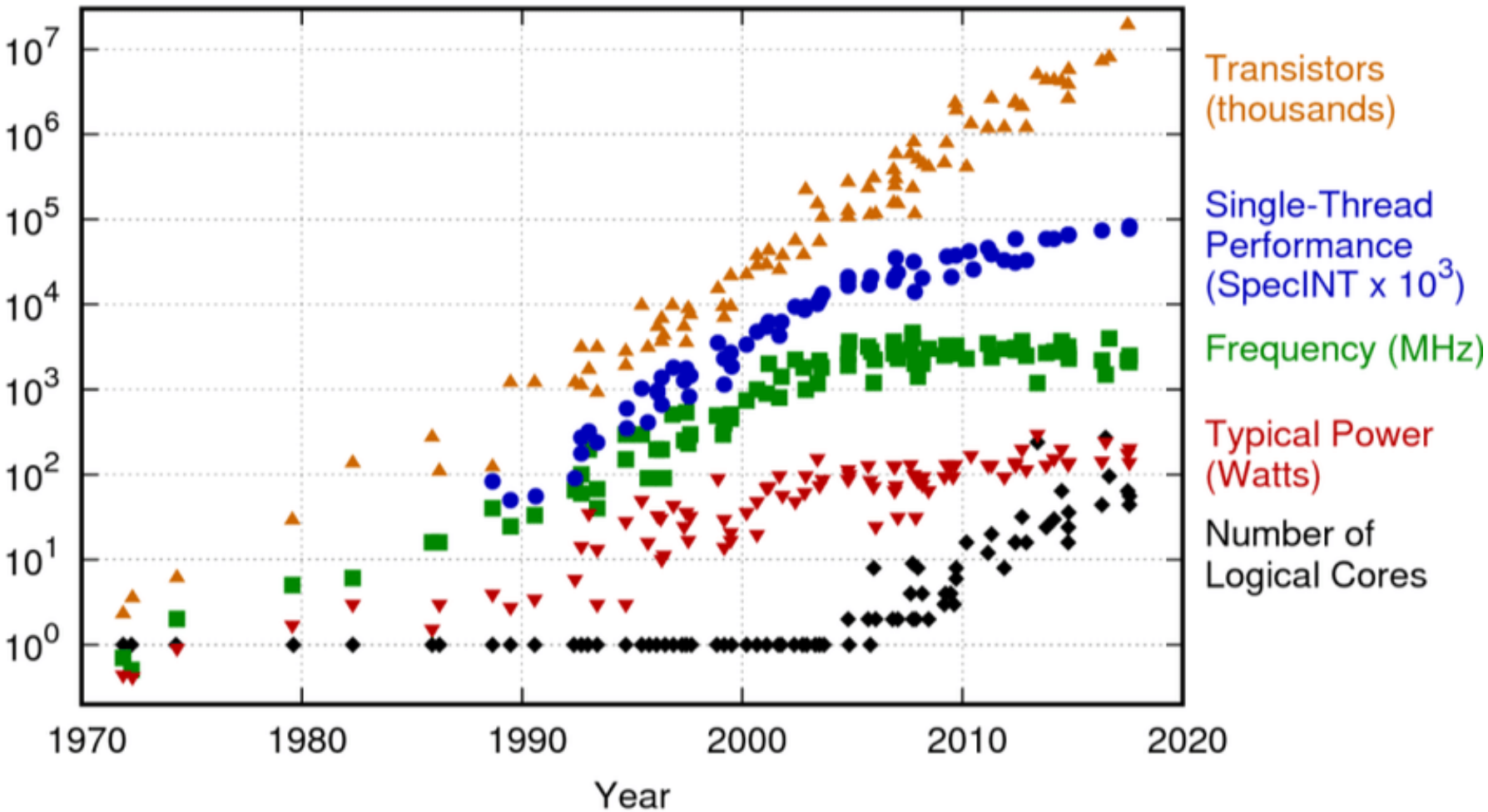
Architectural Changes:

- Multi-core
- Virtualization

New Applications:

- The cloud
- Mobile computing

Technology Changes



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Multi-core

- Processors cannot keep up with demands

Multi-core

- Processors cannot keep up with demands
- Solution: Parallelism. Computer system with multiple processors/cores
  - Even mobile phones have dual core nowadays

Multi-core

- Processors cannot keep up with demands
- Solution: Parallelism. Computer system with multiple processors/cores
  - Even mobile phones have dual core nowadays
- Many challenges for OS
  - Some resources are shared, some not
  - Complex management
  - Many architectural options to achieve parallelism: many processors, many cores, hardware/hyper-threads

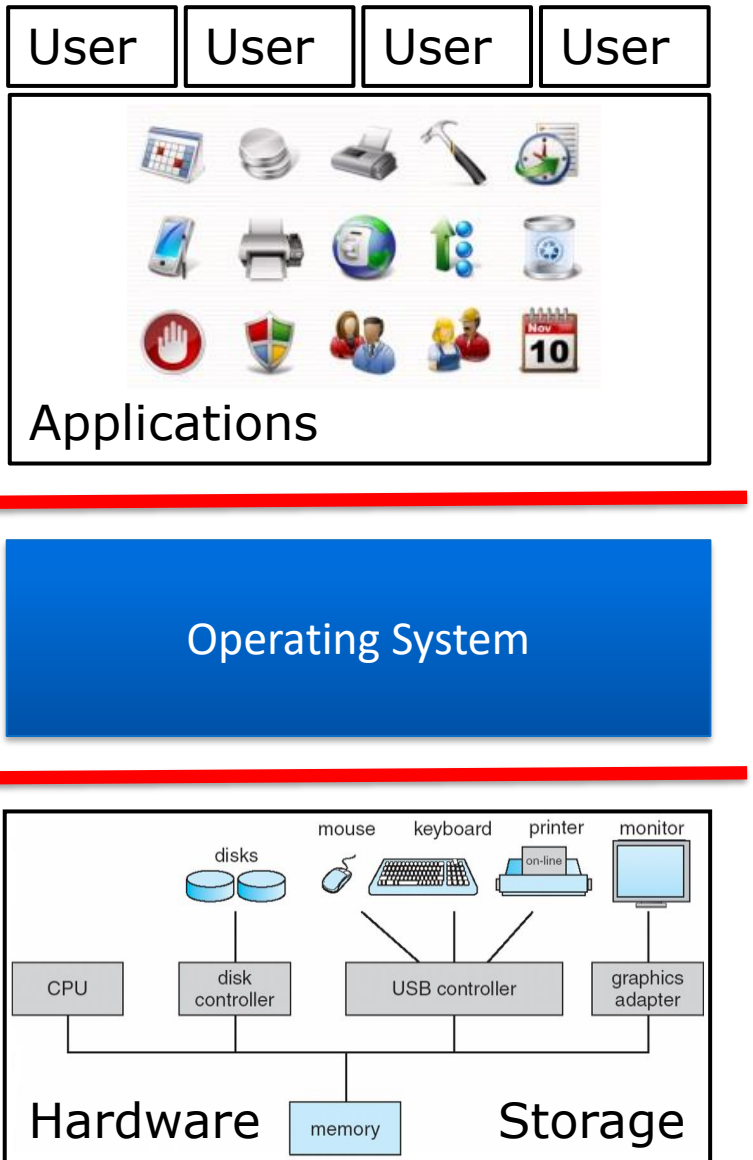
Virtualization

- **Virtualize** *vb (computer science) (tr)* to transform (something) into an artificial computer-generated version of itself which functions as if it were real.
- **Virtualization** is the creation of a virtual (מדומה), rather than actual (ממשי), version of something, such as an operating system, a server, a storage device, or network resources.
- **Virtualization** deals with “extending or replacing an existing interface so as to mimic the behavior of another system”

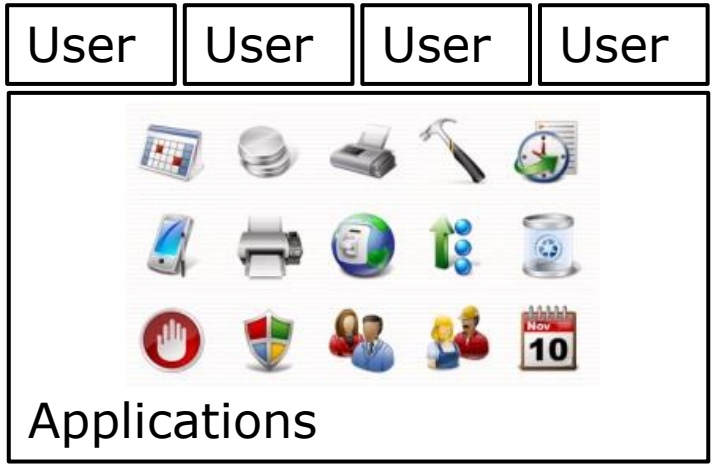
Simple (and not so simple) Examples of Virtualization

- Example 1: Dividing hard drive into ***partitions***.
  - A partition is a **logical** division of the hard drive to create, in effect, two separate hard drives. In reality, however, there is only one hard drive hardware
- Example 2: Virtual memory
- Example 3: Virtual Private Network (VPN)

Virtual Machines (One Example)

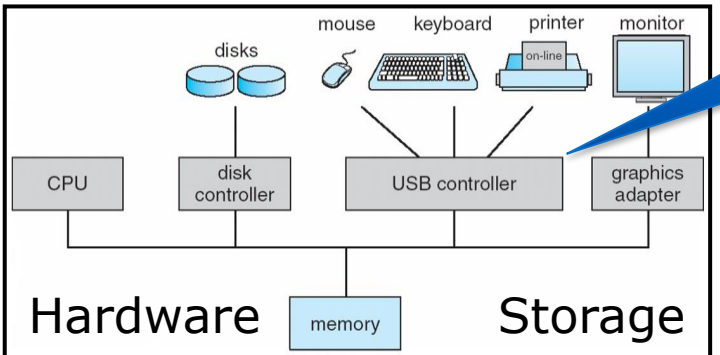


Virtual Machines (One Example)



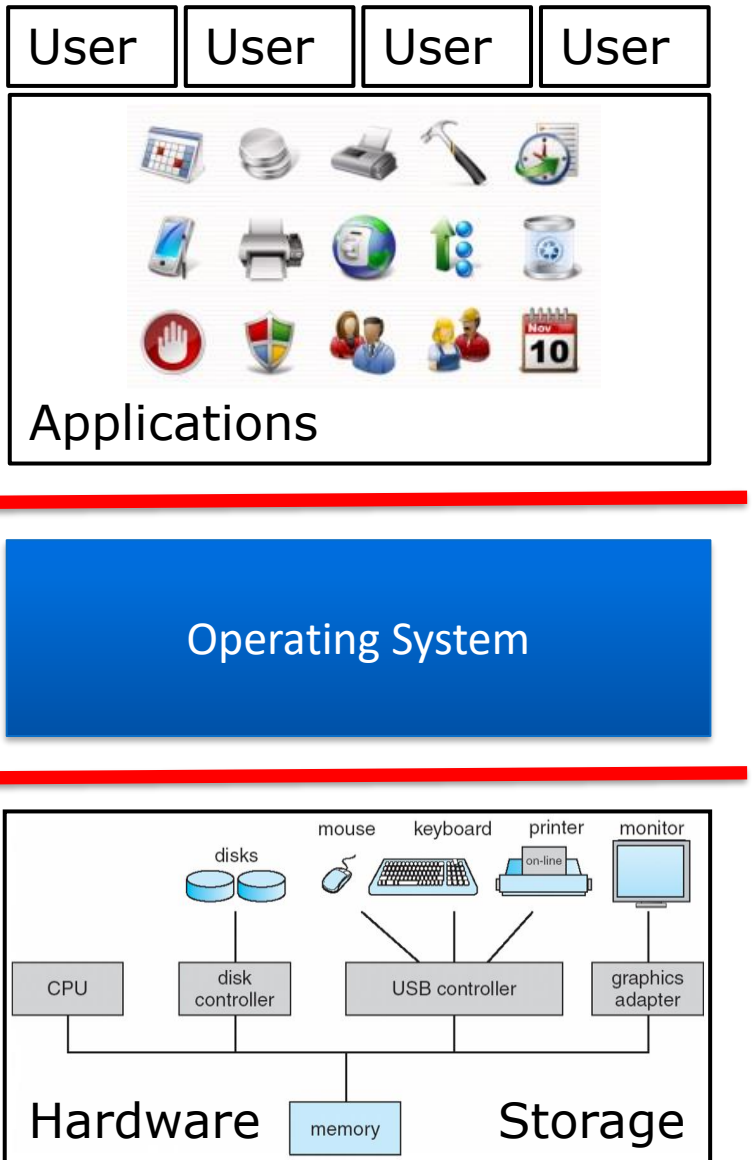
Operating System

OS controls the hardware and is tightly coupled with it

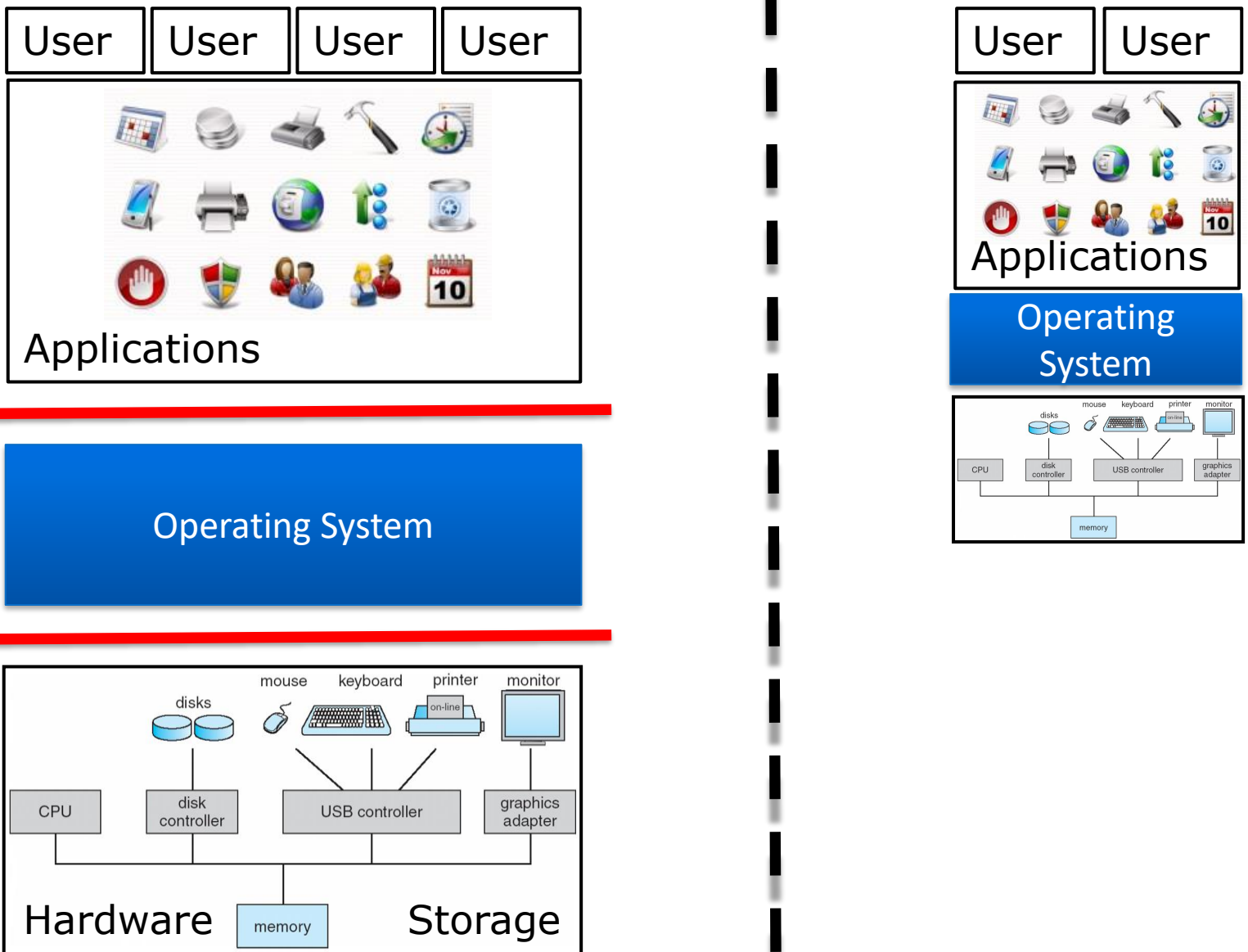


Physical Hardware

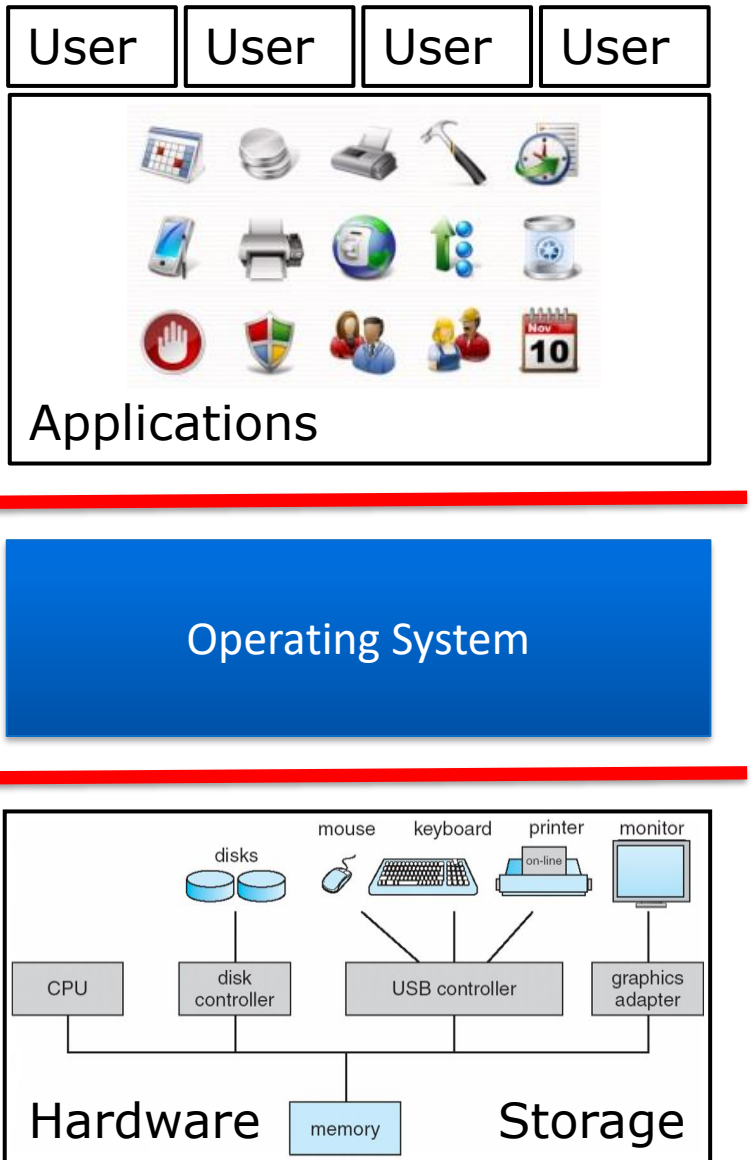
Virtual Machines (One Example)

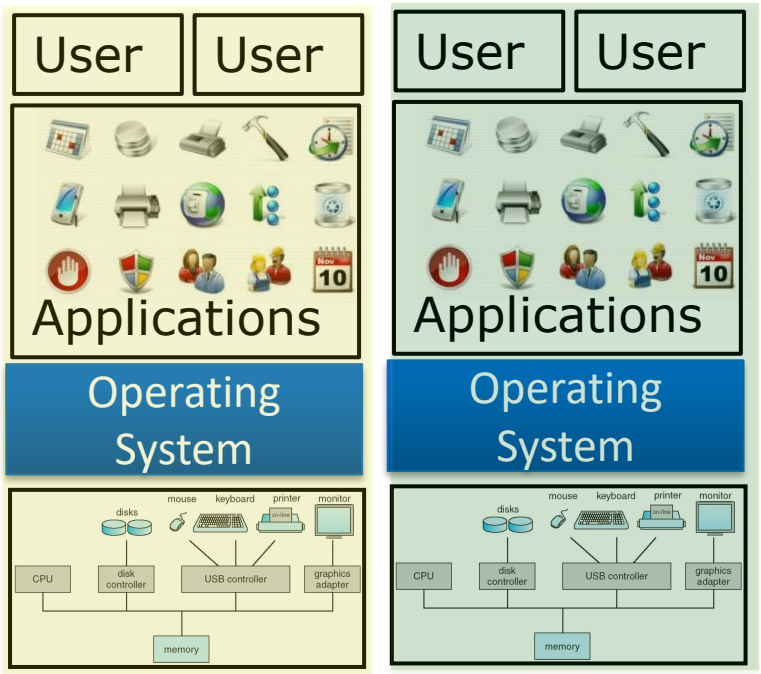


Virtual Machines (One Example)

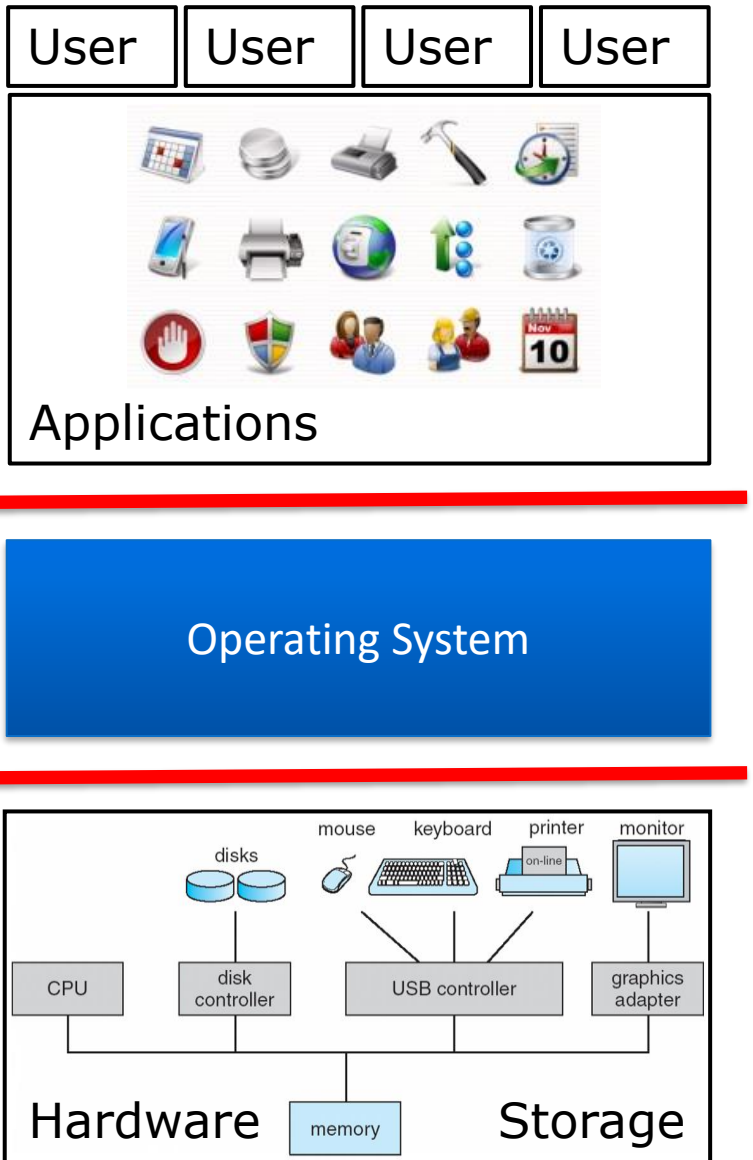


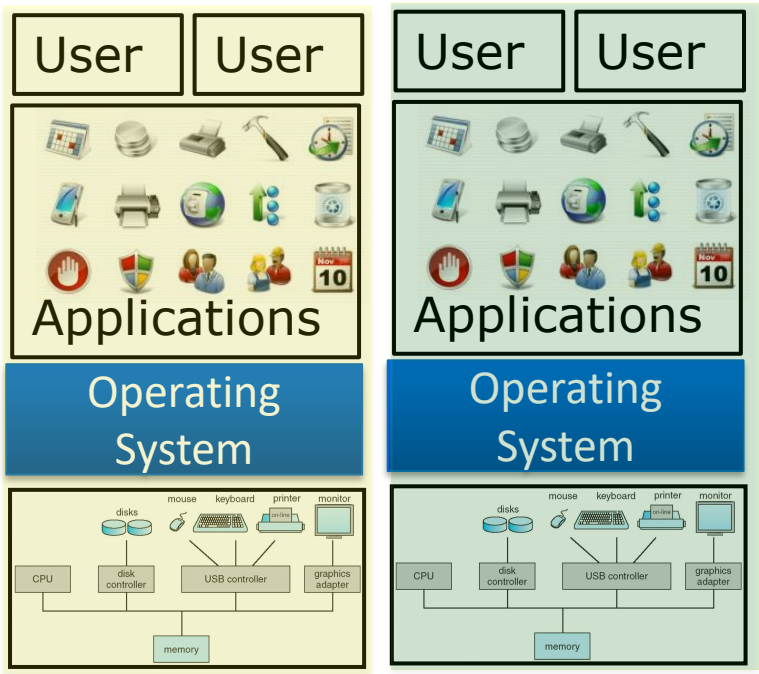
Virtual Machines (One Example)

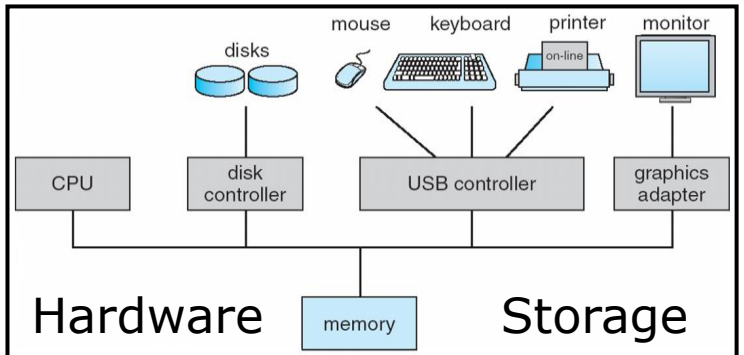




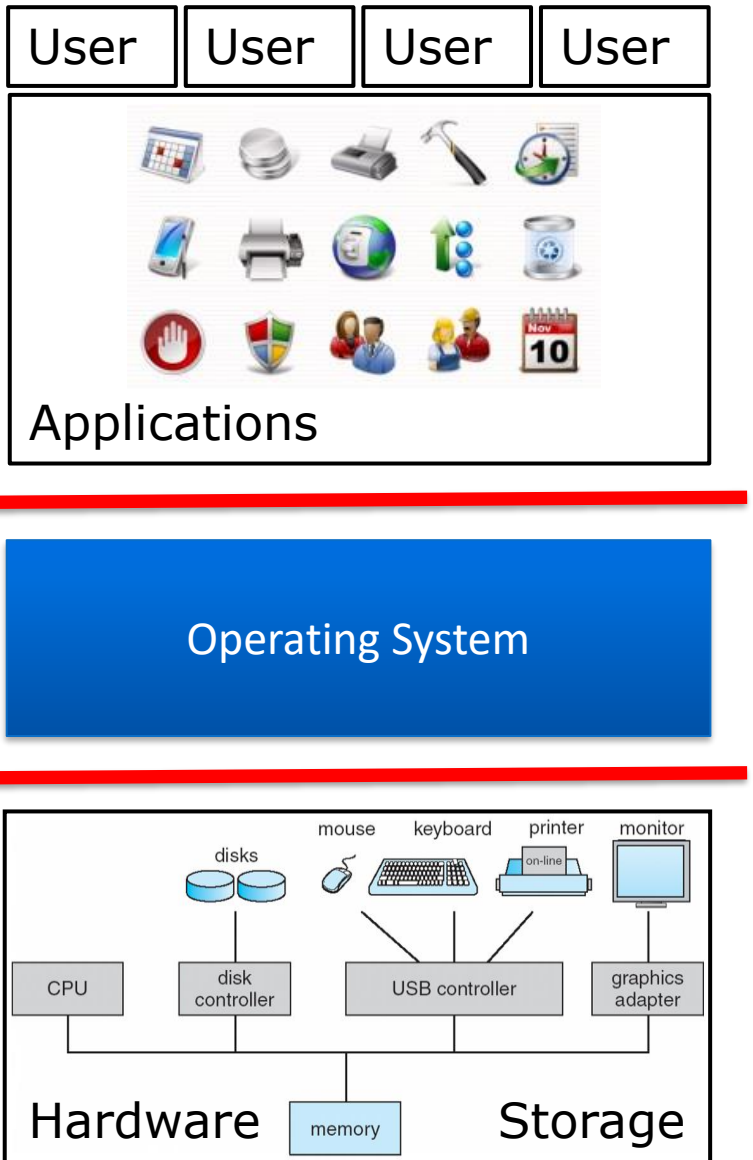
Virtual Machines (One Example)

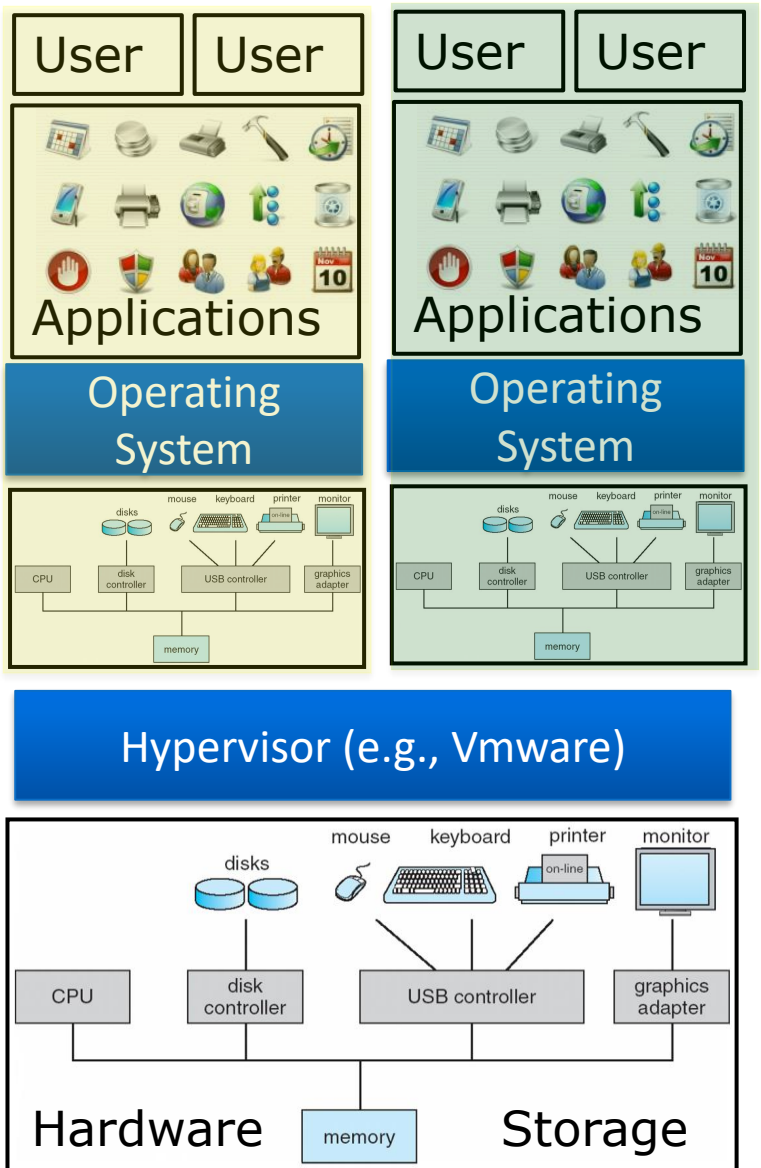




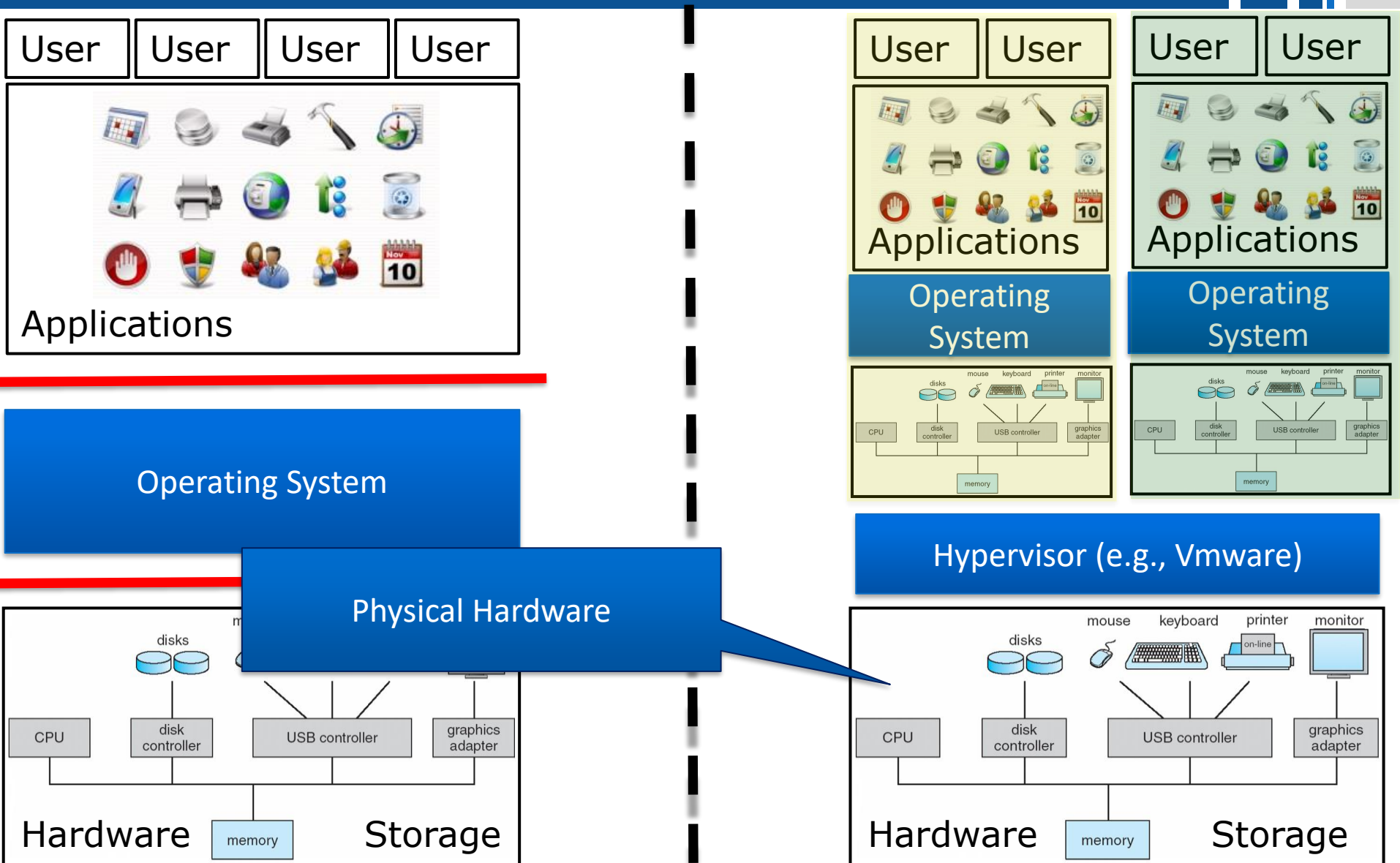


Virtual Machines (One Example)

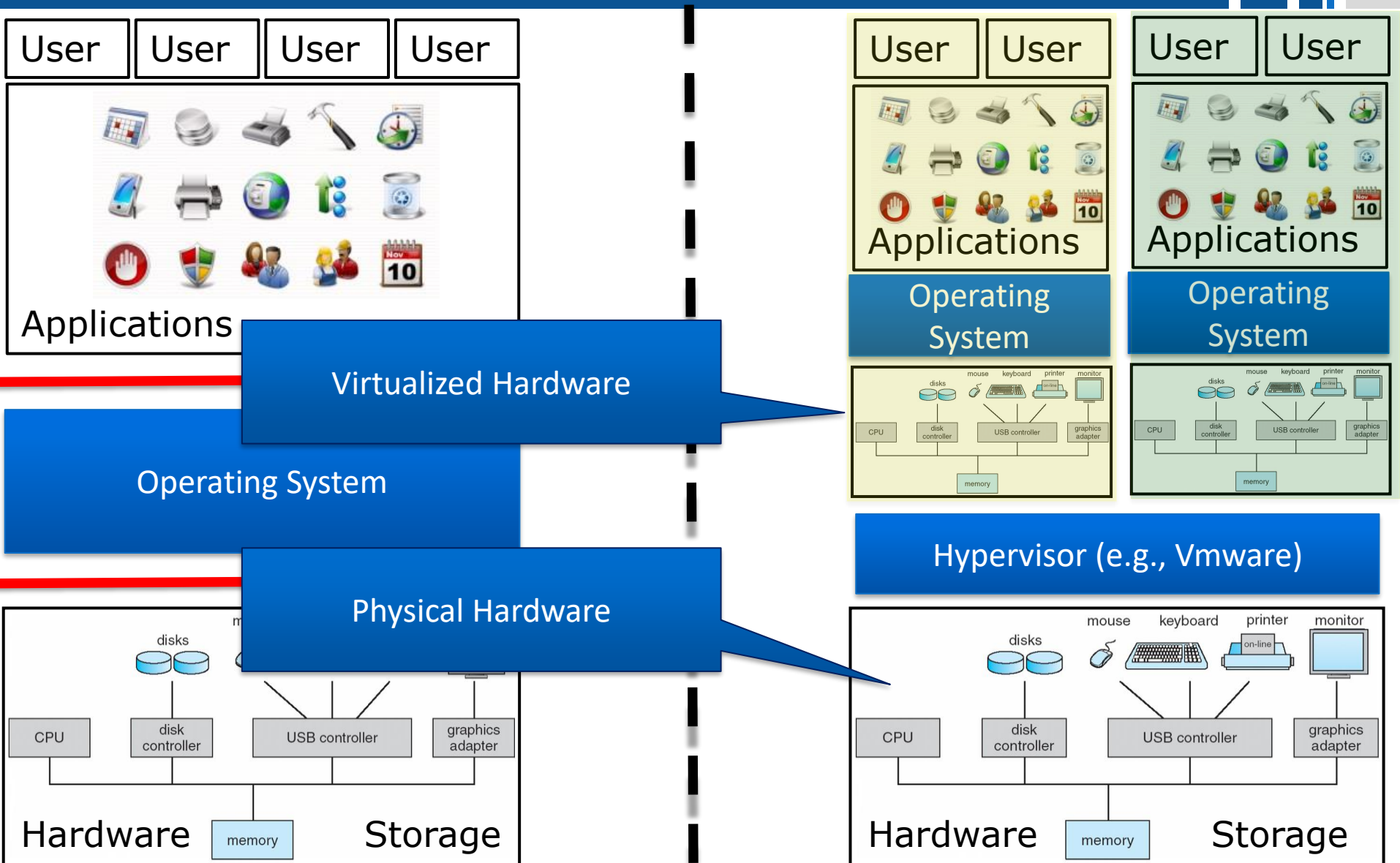




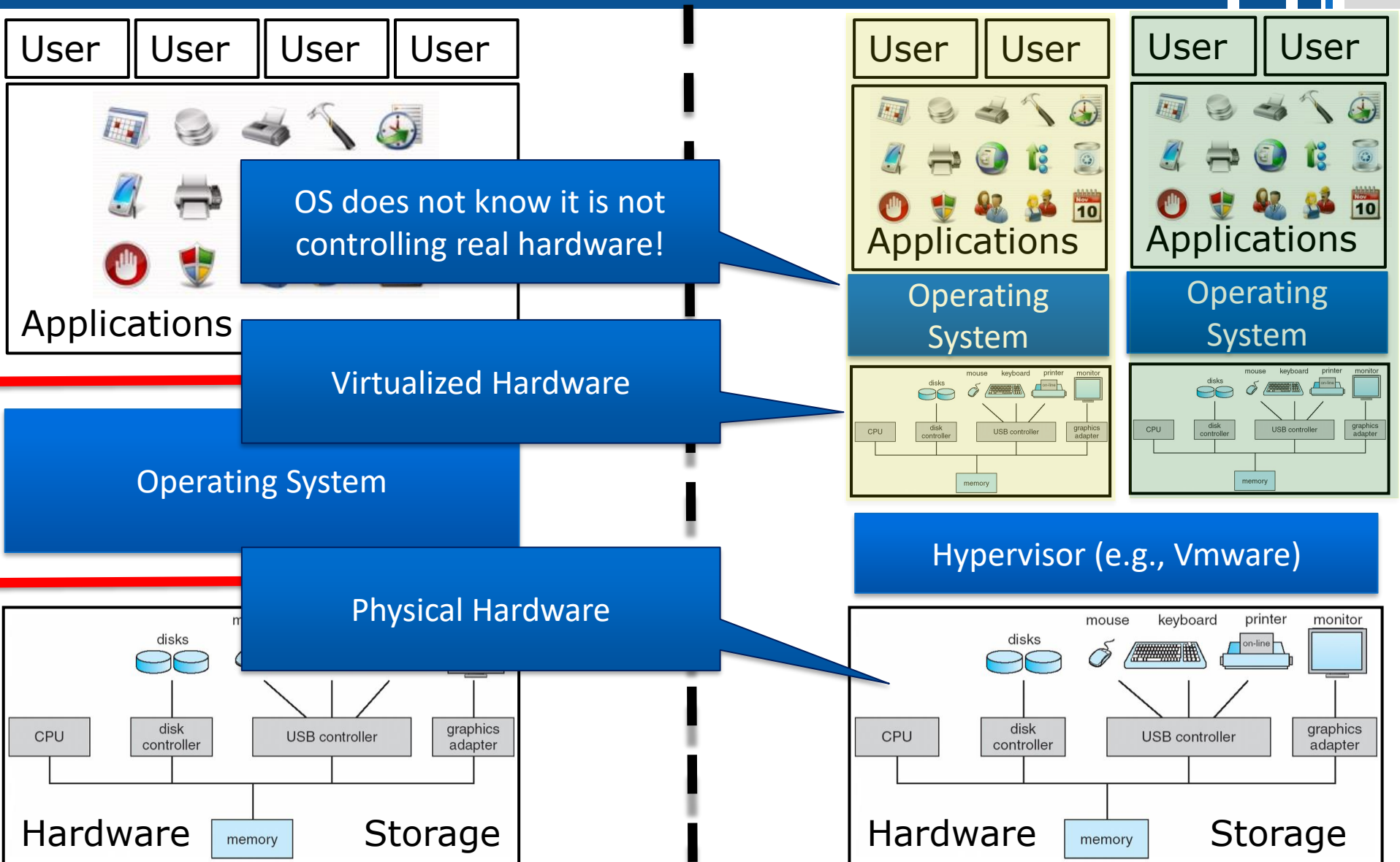
Virtual Machines (One Example)



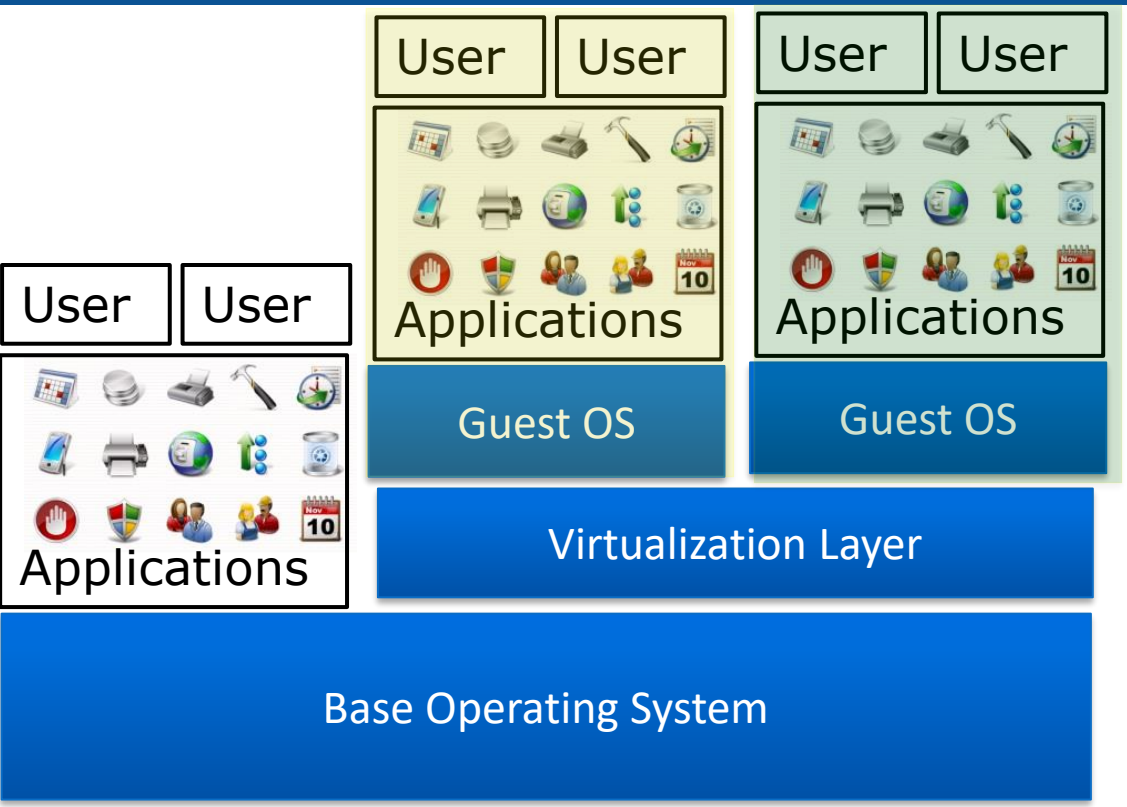
Virtual Machines (One Example)

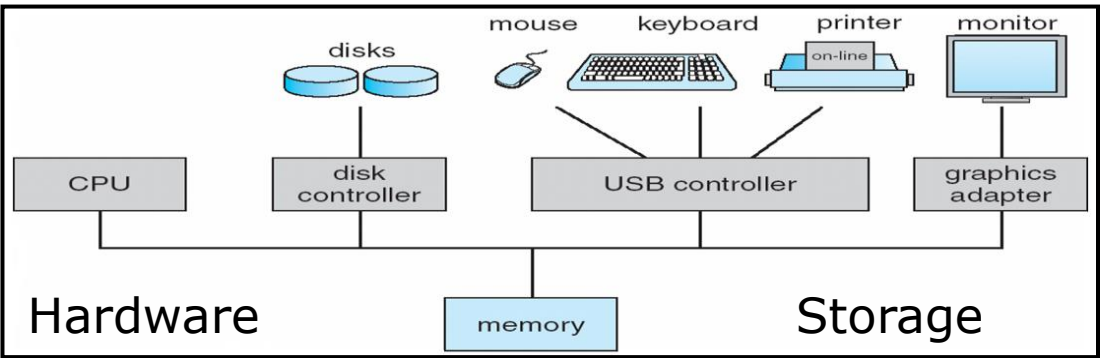


Virtual Machines (One Example)

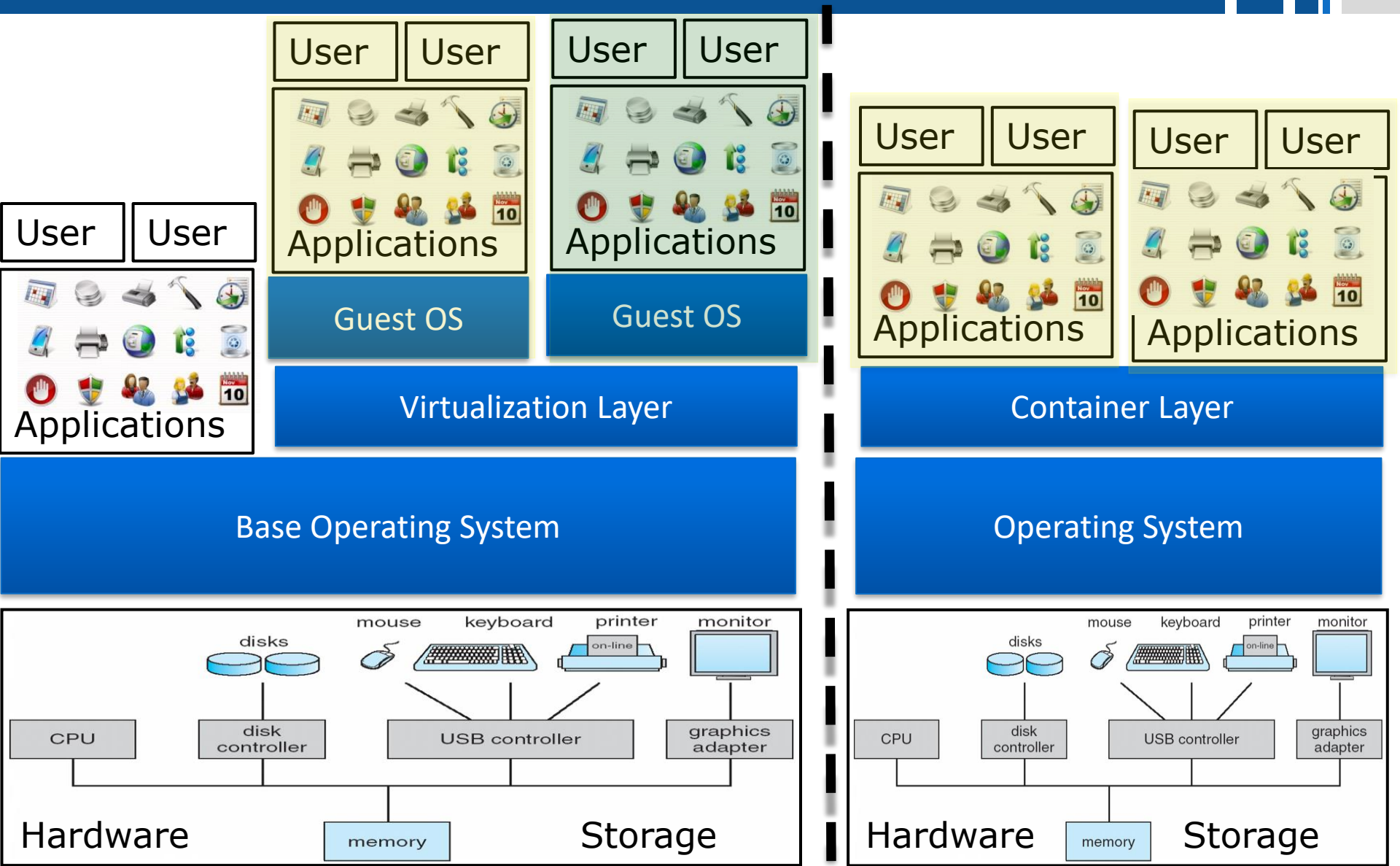


Other Virtualization Flavors





Other Virtualization Flavors



Virtualization: Some Features

- **VM Snapshot**: freeze a copy of virtual machine with the specific state of the virtual machine
- **VM Migration**: move a VM to another host
- **Scaling**: Add/remove more VM to cope with demand
  - Very important for the cloud
- **Isolation:** Even when running on the same hardware, the operation of one VM are (almost) completely isolated from the others → Added security

However, there might be an overhead in performance.

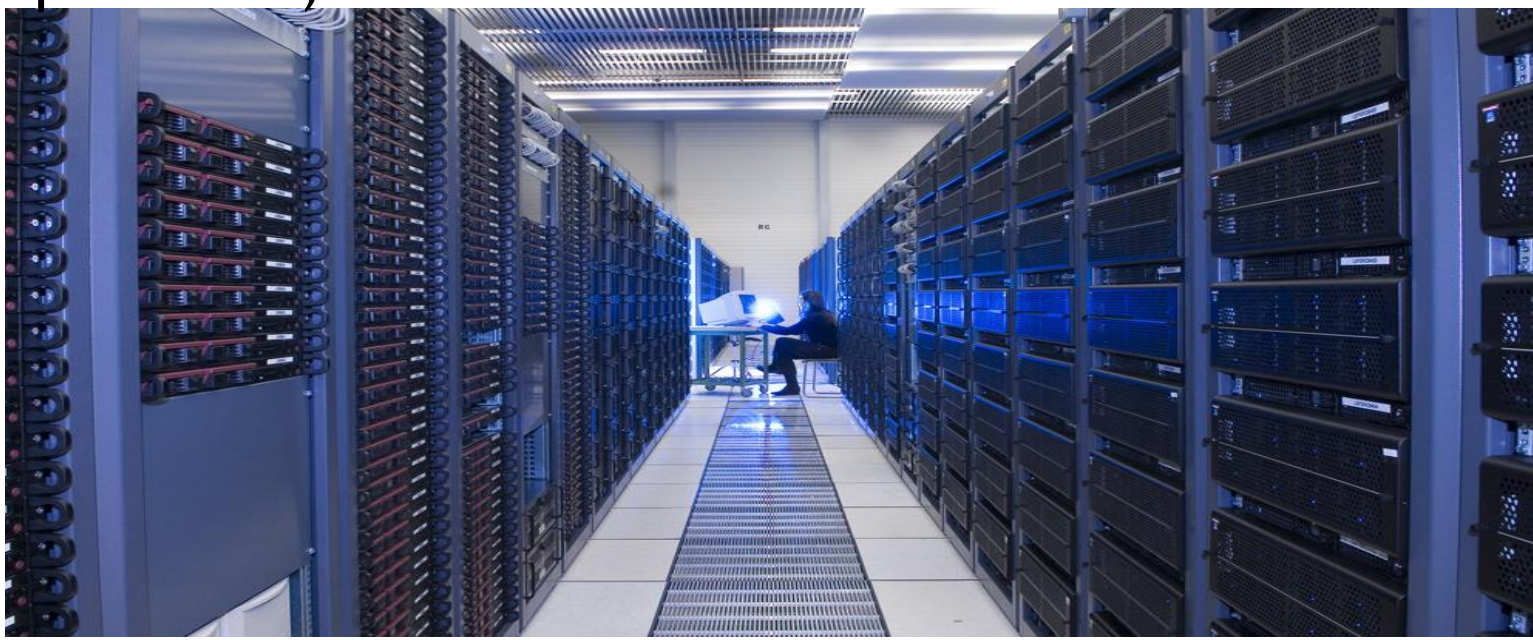
The Cloud

- Provides computing resources over the Internet
  - No need of HW computer per application → just browser
  - Many flavors: Software as a Service, Infrastructure as a Service, and many more
- Delivers a hosting environment that is immediate, flexible, scalable, secure, and available – while saving corporations money, time and resources
  - Uses virtualization for resource sharing (many VMs on the same server),

The Cloud

Assumptions:

1. HW and maintenance are cheaper for big companies (e.g., Amazon, Microsoft, Google).
2. Big companies have more experience and infrastructure (home-made protocols).



https://www.youtube.com/watch?v=avP5d16wEp0

Mobile Computing

Mobile Computing

Statue of an angel holding a mobile phone at St. John's Cathedral in Den Bosch, Netherlands

What's the difference?

	Desktop	Mobile Computing
Power	Electricity	Battery
Network	LAN	WiFi / Cellular
Main Memory	8-64GB RAM	3GB
Disk	TB HardDisk 512GB SSD	8/16/64 GB Flash Memory
CPU	Multiple Processors Cores	Slower and Smaller CPU & Cores
Input	Mouse / Keyboard	Touch Screen
Weight	No main concern	Concern

Kilo=10^3, **M**ega=10^6
Giga=10^9, **T**era=10^{12}

What's the difference?

	Desktop	Mobile Computing
Power		
Network		
Main Memory		
Disk		
CPU		
Input		
Weight		

Multi-programming/Time-sharing

Assumptions

- CPU is always there, and therefore, not using it is a waste
- CPU is much faster than I/O
- Memory is large enough to host many programs
- Direct memory access (DMA)
- More than one task to do at a time

Required OS Features

- I/O primitives
- Memory management
  - Memory protection
- Scheduler
  - Manages flow of jobs in and out of the CPU
- Spooler
  - Manages slower I/O devices (e.g. printer Spooler)

What's the difference?

	Desktop	Mobile Computing
Power		
Network		
Main Memory		
Disk		
CPU		
Input		
Weight		

Multi-programming/Time-sharing

Assumptions

- ✗ CPU is always there, and therefore, not using it is a waste
- ✓ CPU is much faster than I/O
- ✗ Memory is large enough to host many programs
- ✓ Direct memory access (DMA)
- ✗ More than one task to do at a time

Required OS Features

- I/O primitives
- Memory management
  - Memory protection
- Scheduler
  - Manages flow of jobs in and out of the CPU
- Spooler
  - Manages slower I/O devices (e.g. printer Spooler)

Some Requirements for successful Mobile OS

- Easy and consistent for app developers
  - Inconsistent hardware (different screen size, different camera, different CPUs)
  - Two main players: iOS, Android,
- Responsive, fast
  - Users with limited attention time span
- Efficient – Battery efficient
  - Every single CPU cycle consumes battery power.
  - The more time the CPU can spend sleeping the longer the battery life will be→ Less Multitasking
- Secure
  - Lots of personal data on phones. Mobile is easy to lose or steal.
- Run with limited RAM
  - "Being a Responsible Background App"

More types of OS (the OS zoo)

- Mainframe operating systems
  - VM (IBM), VMS(Compaq)
- Server operating systems
- Multiprocessor operating systems
- Personal computer operating systems
- Real-time operating systems
  - VxWorks
- Embedded operating systems
  - VxWorks
- Smart card operating systems
- Network OS
- OS for smart devices
-