Linux Lecture-1

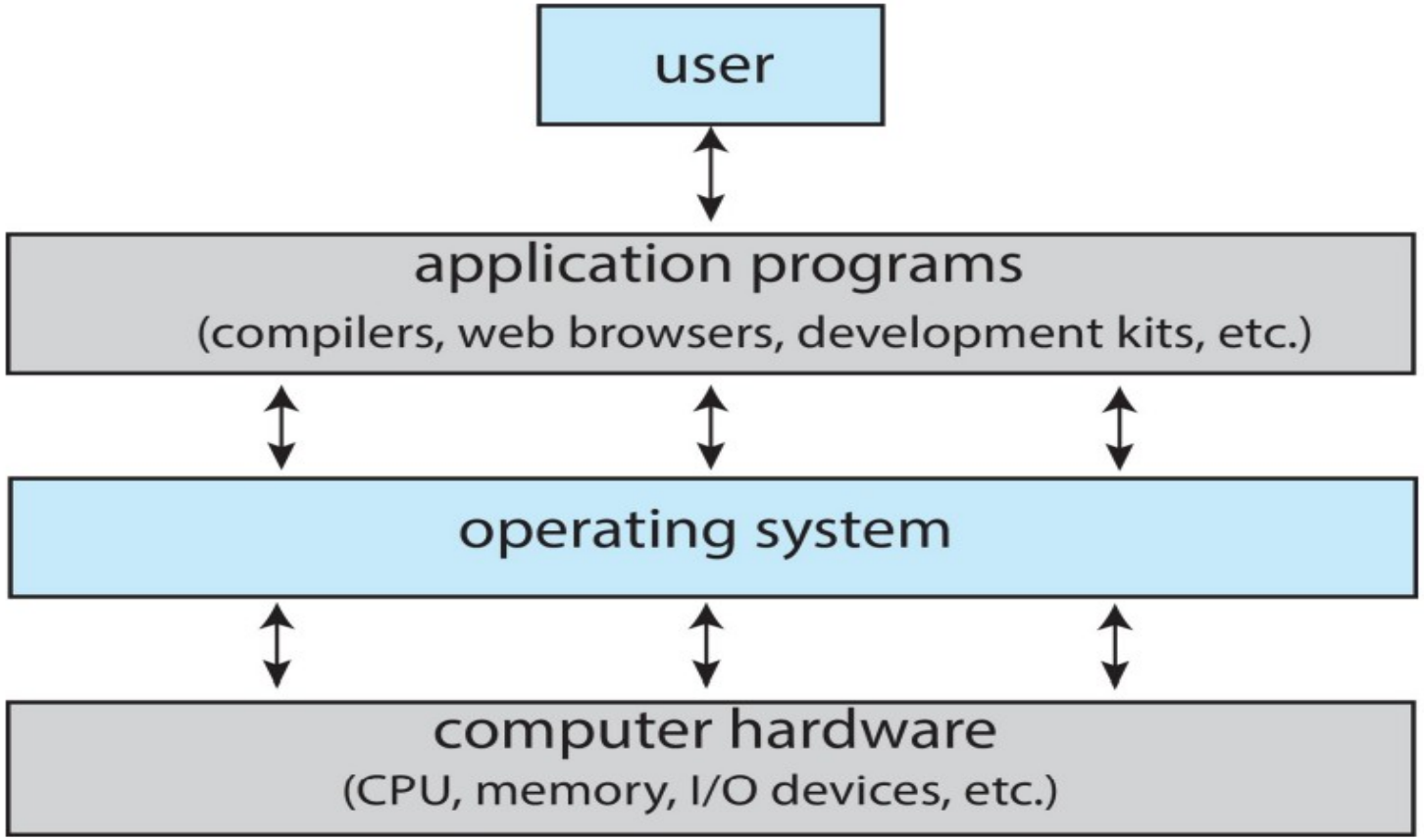
Topics

- What is an OS?
- Computer System Organization
- Multiprogramming vs Multitasking
- User Mode & Kernel Mode
- System Call, Interrupt, Trap & Signal
- Processes
- Threads
- Scheduling Algorithms
- Linux History
- Pros & Cons
- Distributions
- Virtualization

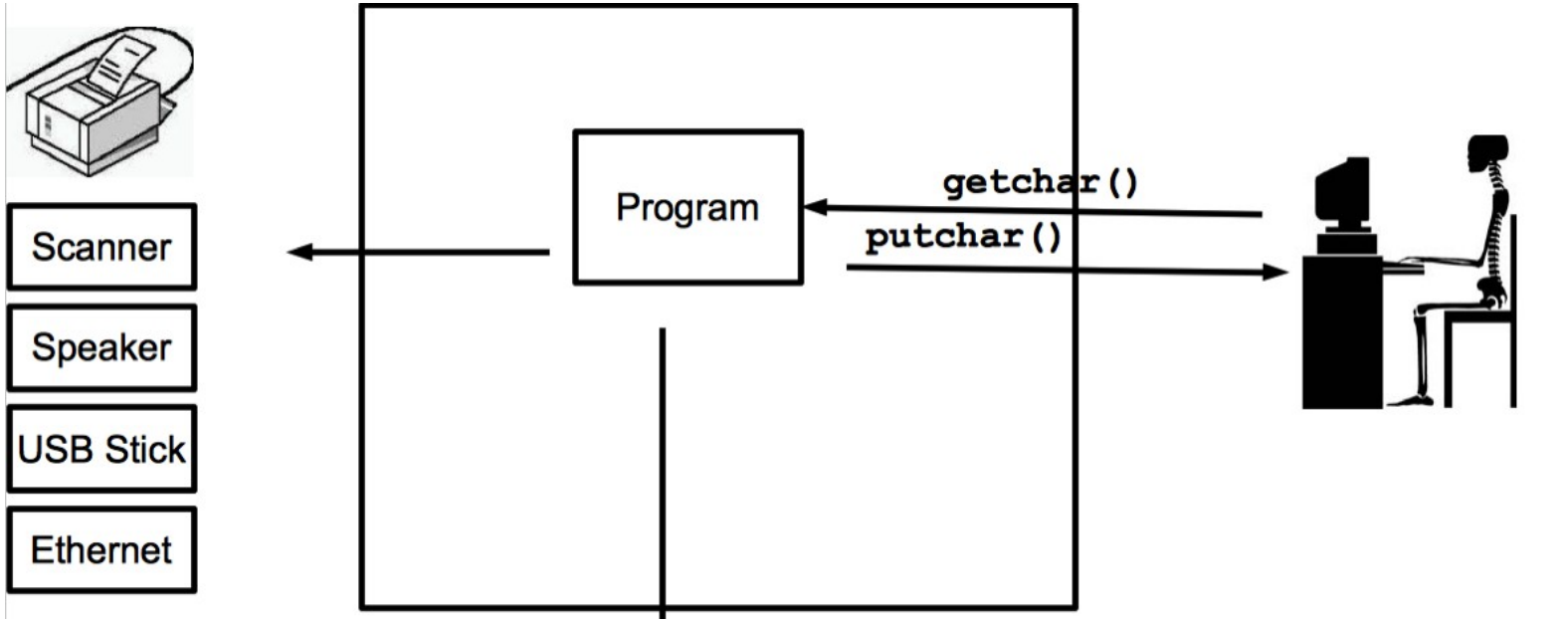
What is an OS?

- An OS is a program running at all times on the computer (usually called the kernel), that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware
- Provides an orderly and controlled allocation of the processor(s), memory(ies) and I/O devices among the various programs competing for them.
- Sits between programs & hardware
- Sits between different programs
- Sits between different users
- Primary goal of OS is convenience of user and secondary goal is efficient operation of the computer system.

Operating System - An Abstraction



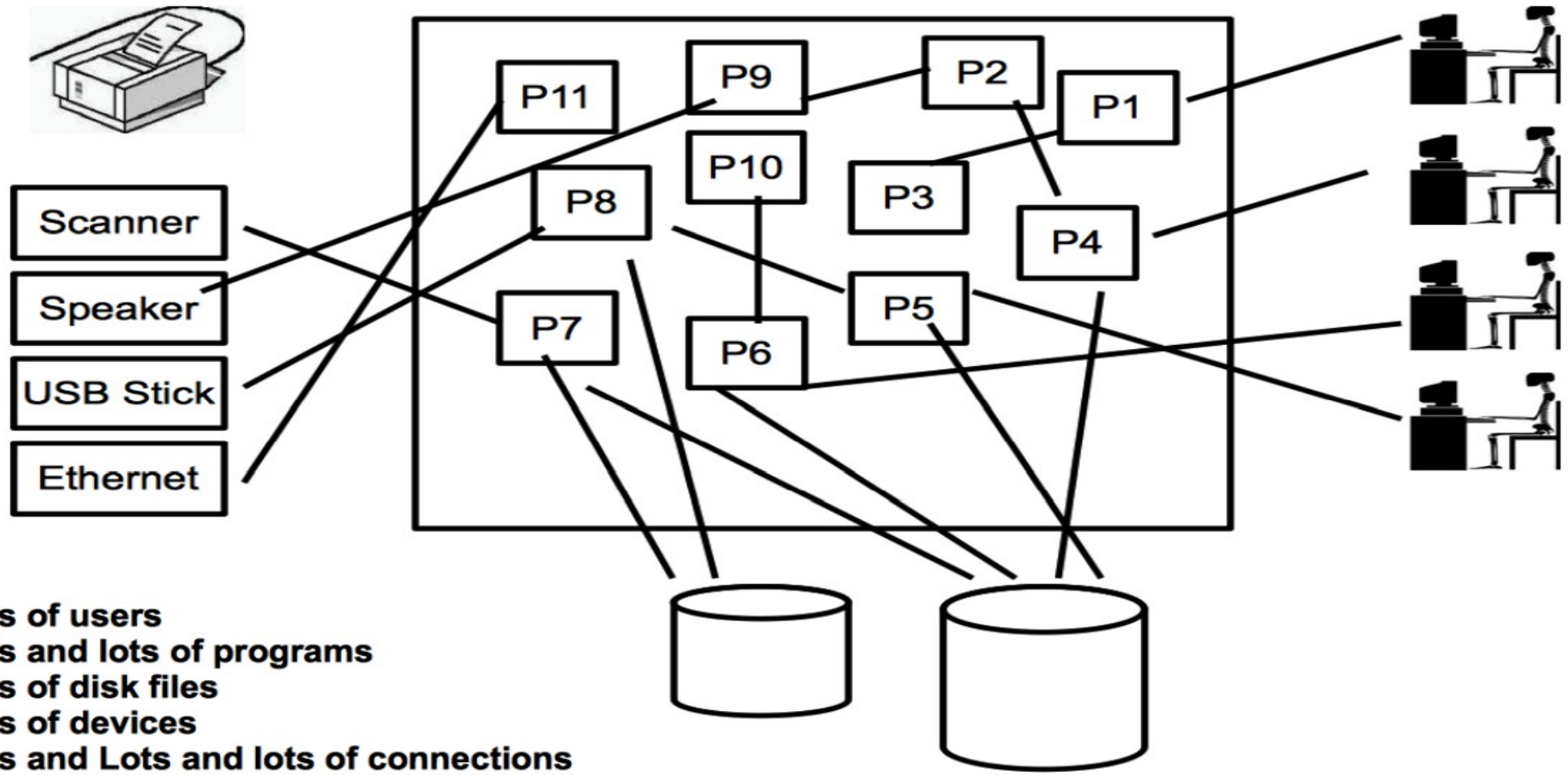
Operating System - An Abstraction



An application programmer normally write programs that acquire data from an external source like a file or a keyboard, process the data, and deliver the output to external sources such as files or the console

```
int main()
{
    int c;
    while ((c = getchar()) != EOF)
        putchar(c);
    return 0;
}
```

Operating System - An Abstraction



Lots of users
Lots and lots of programs
Lots of disk files
Lots of devices
Lots and Lots and lots of connections

Who is managing all these resources and connecting various devices to correct programs

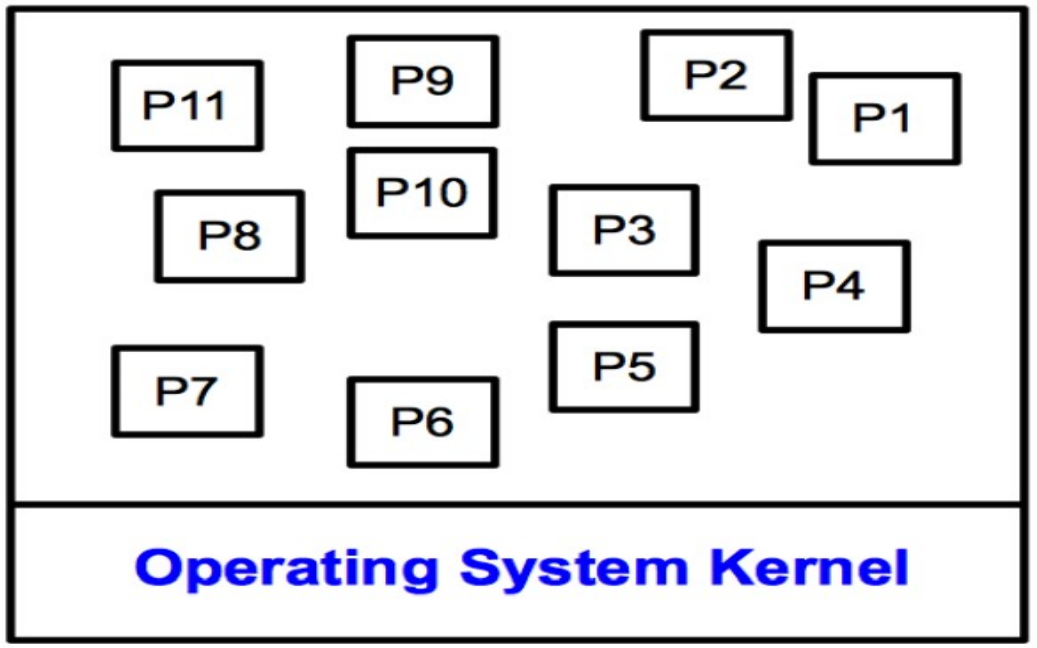
Operating System - An Abstraction

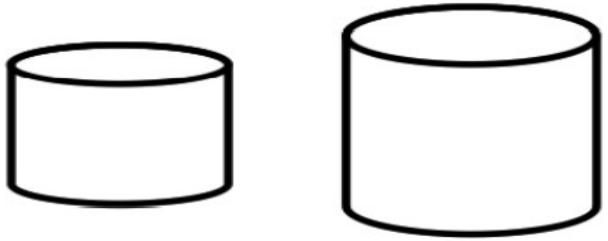
Scanner

Speaker

USB Stick

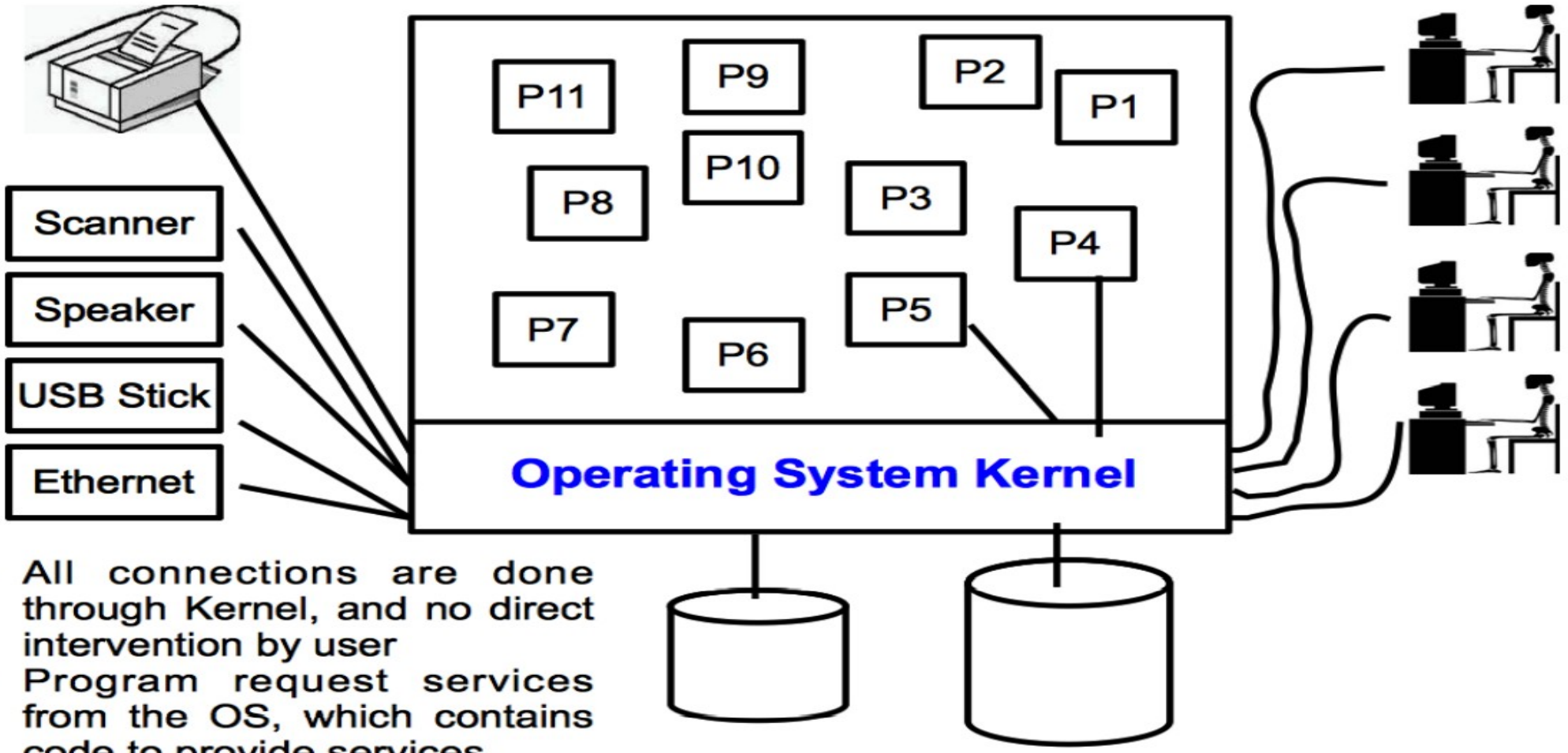
Ethernet





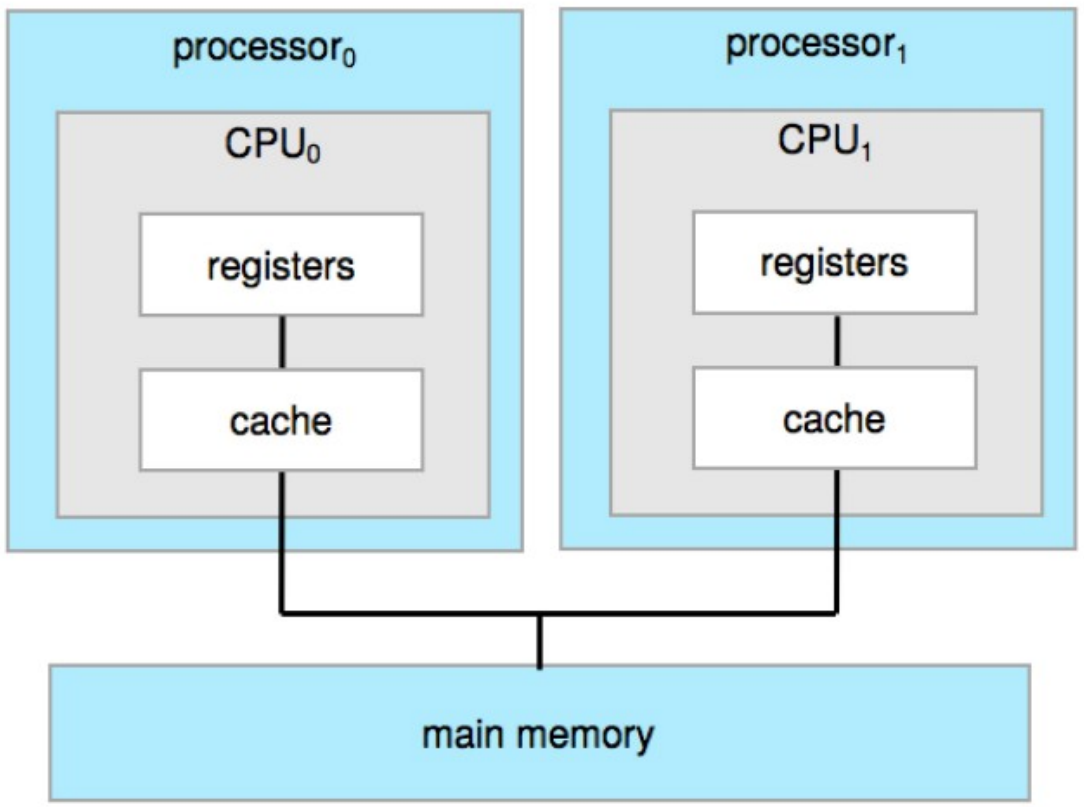
Role of Operating System is to manage all these resources and to connect various devices to the correct programs

Operating System - An Abstraction

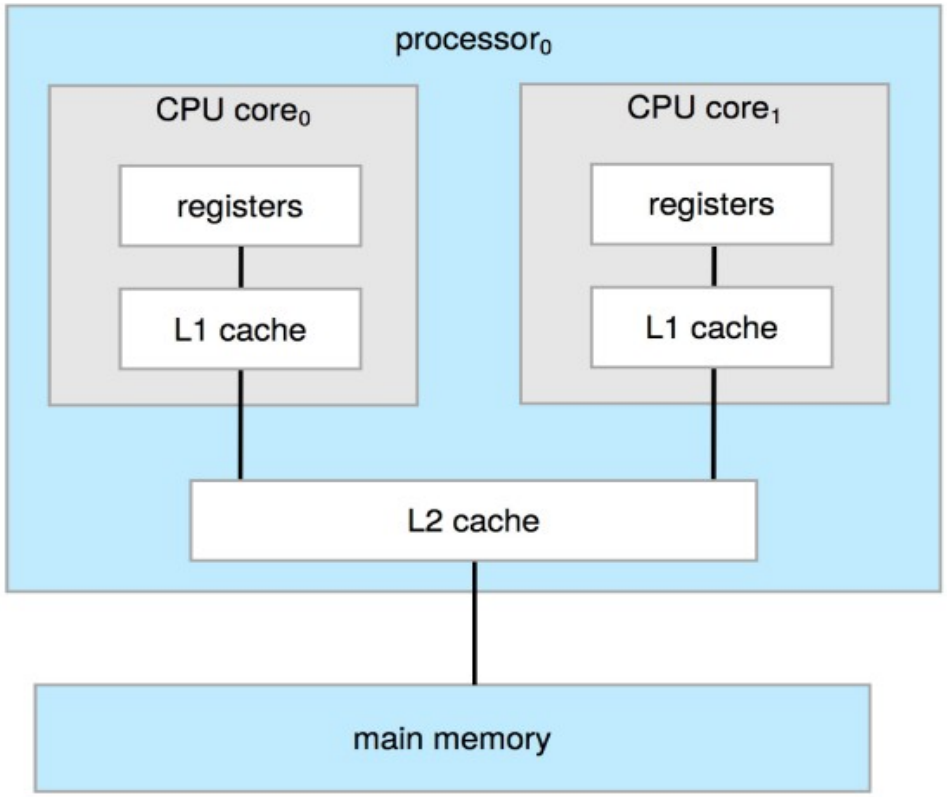


- All connections are done through Kernel, and no direct intervention by user
- Program request services from the OS, which contains code to provide services
- An operating system provides these services via its API

Computer System Organization



Multi-processor Architecture



Multi-core Architecture

Multiprogramming vs Multitasking

Multiprogramming

- Single user cannot keep CPU and I/O devices busy at all time
- Multiprogramming organizes jobs so CPU always has one to execute
- One job selected and run via job scheduling
- When it has to wait (for I/O for example), OS switches to another job

Multitasking

- CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing
- Each user has at least one process executing in memory
- If several processes are ready to run at the same time, then OS performs CPU scheduling
- If processes don’t fit in memory, swapping moves them in and out to run

Protection

In Multitasking there are multiple processes using various resources of the computer. Sounds great, but like no free lunch, it has disadvantage as well, i.e. the issue of protection

- Keep user programs from crashing the OS
- Keep user programs from crashing each other
- Keep parts of OS from crashing other parts

Protection is implemented by keeping two modes

Dual Mode Operation

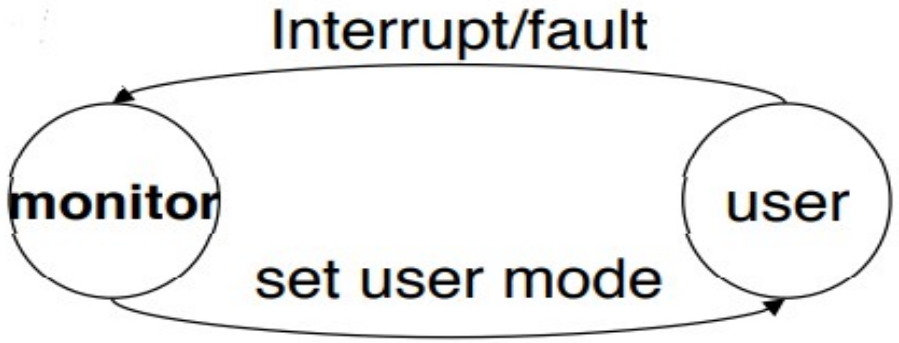
To protect the OS and all other programs and their data from any malfunctioning program, protection is needed for any shared resource. It is done by keeping two modes:

User Mode. Execution done on behalf of a user program

Monitor/Kernel/System/Supervisor Mode. Execution done on behalf of OS

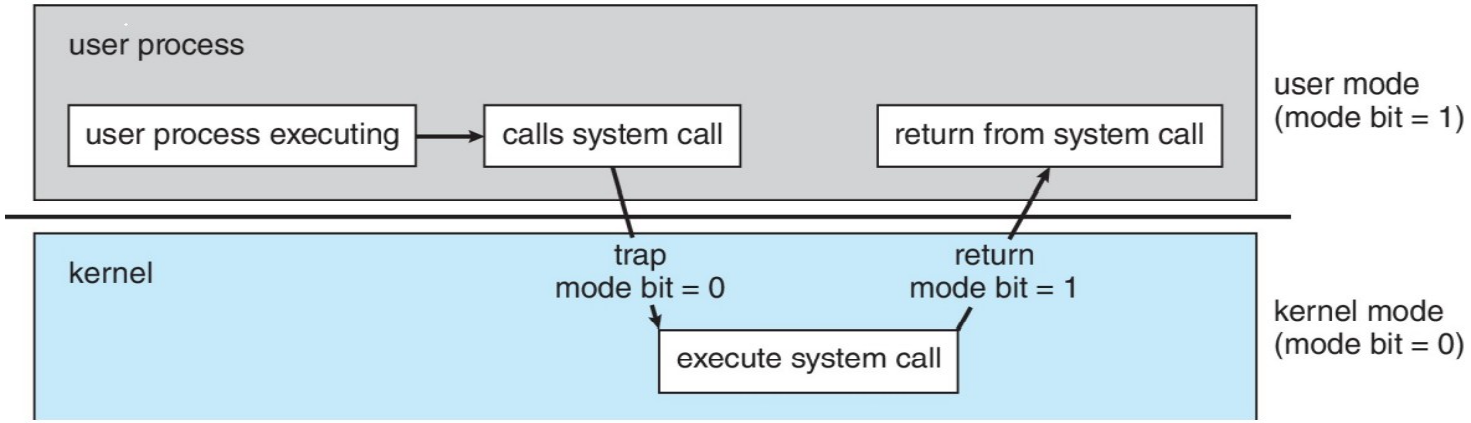
Mode bit added to computer h/w to indicate the current mode: Kernel(0), User (1)

When an interrupt or fault occurs hardware switches to monitor mode. Privileged instructions can only be executed in monitor mode.



Transition from User to Kernel Mode

- A user process must not be given an open access to kernel code
- Any user/application request that involves access to any system resource must be handled by the kernel code
- The mechanism used by an application program to request a service from Operating System is via a system call
- A **System call** is usually a request to the OS kernel to do a h/w, system specific or privileged operation.



Types of Entry Points to Kernel

- When a program makes a System Call.
- Interrupt: An event generated by an I/O device to get the attention of CPU and control goes to the OS. When an I/O device has generated an Interrupt; e.g. a disk controller has generated an interrupt to CPU that my reading is complete the data is now sitting in my buffer, You can go and get it.
- Trap: An event generated by CPU and control goes to the OS. When a trap occurs; e.g. If a program has made a division by zero, a trap will be generated which will execute a different piece of code in kernel (.
- Signal: A notification given to a process by the OS because the process did something, the user did something, one process wants to tell another process something. When a signal comes to a process some piece of kernel code will be executed

System Call

A system call is the controlled entry point into the kernel code, allowing a process to request the kernel to perform a privileged operation. Before going into the details of how a system call works, following points need to be understood:

- A system call changes the processor state from user mode to kernel mode, so that the CPU can access protected kernel memory
- The set of system calls is fixed. Each system call is identified by a unique number
- Each system call may have a set of arguments that specify information to be transferred from user space to kernel space and vice versa
- All OS’s offer their own System Calls

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Users, Programs and Processes

- Users have accounts on the system. Users write programs, then execute programs. Different users may execute same program. One user may execute many instances of the same program
- A Program by itself is not a Process: Program is a passive entity like the contents of a file stored on a disk. Process is an active entity, in which the Program Counter specifies the next instruction to be executed
- A Process is

  A program in execution

  An instance of a program running on a computer

  An entity that can be assigned to and executed on a processor
- The UNIX system creates a process every time you run an external command, and the process is removed from the system when the command finishes its execution

CPU Bound and I/O Bound Processes

- **I/O-bound** process – spends more time doing I/O than computations; Many short CPU bursts
- Examples: Word processing, text editors. Billing system of WAPDA which involves lot of printing.

- **CPU-bound** process – spends more time doing computations; Few very long CPU bursts
- Examples: Simulation of NW traffic involving lot of mathematical calculation, scientific applications involving matrix multiplication, DSP applications

5-State Process Model

Broadly speaking the life of a process consist of CPU burst and I/O burst but in reality

- A process may be waiting for an event to occur; e.g. a process has created a child process and is waiting for it to return the result
- A process may be waiting for a resource which is not available at this time
- Process has gone to sleep for some time

So generally speaking a Process may be in one of the following five states:

new: The process is being created (Disk to Memory)

ready: The process is in main memory waiting to be assigned to a processor

running: Instructions are being executed

waiting: The process is waiting for some event to occur (I/O completion or reception of a signal)

terminated: The process has finished execution

5-State Process Model

As a process executes, it changes state:

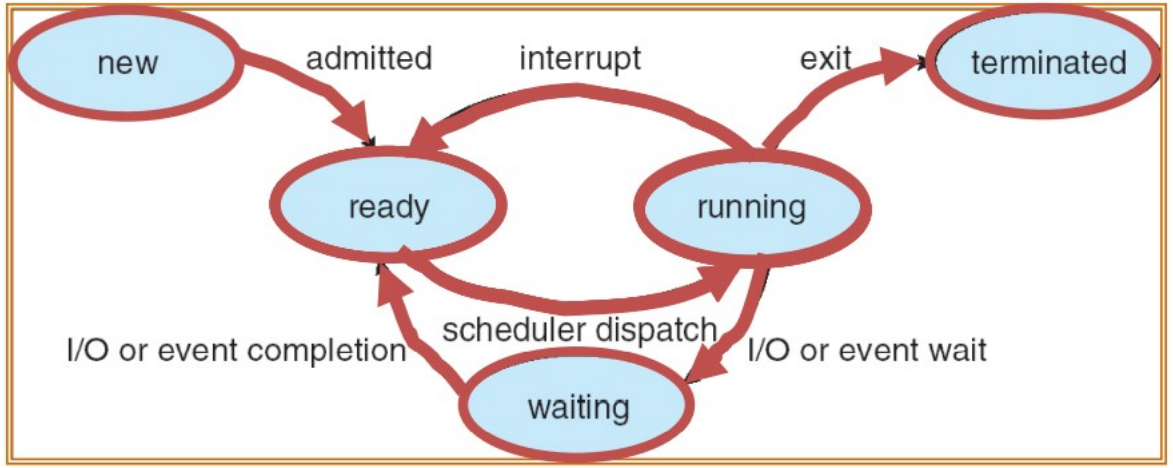
– new: The process is being created

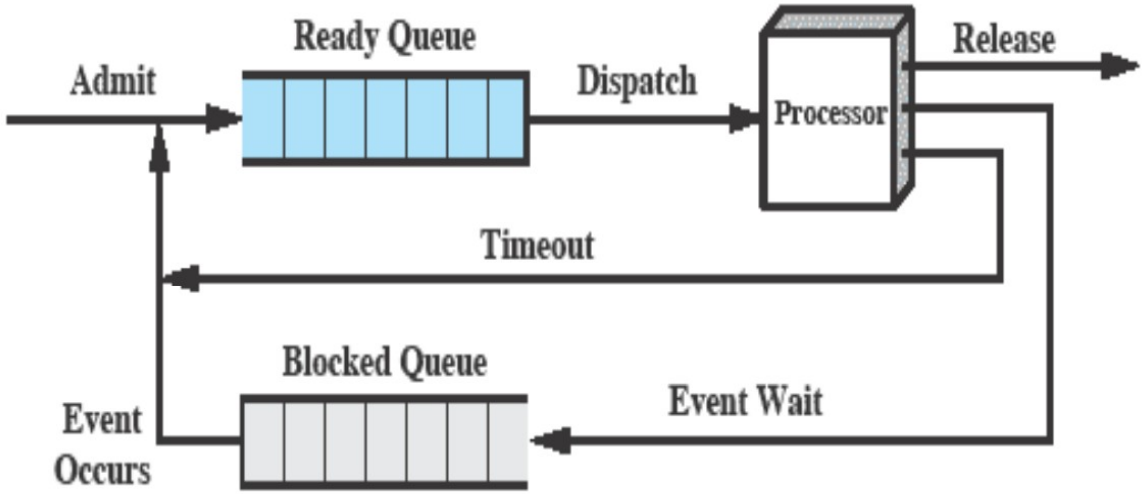
– ready: The process is waiting to run

– running: Instructions are being executed

– waiting: Process waiting for some event to occur

– terminated: The process has finished execution

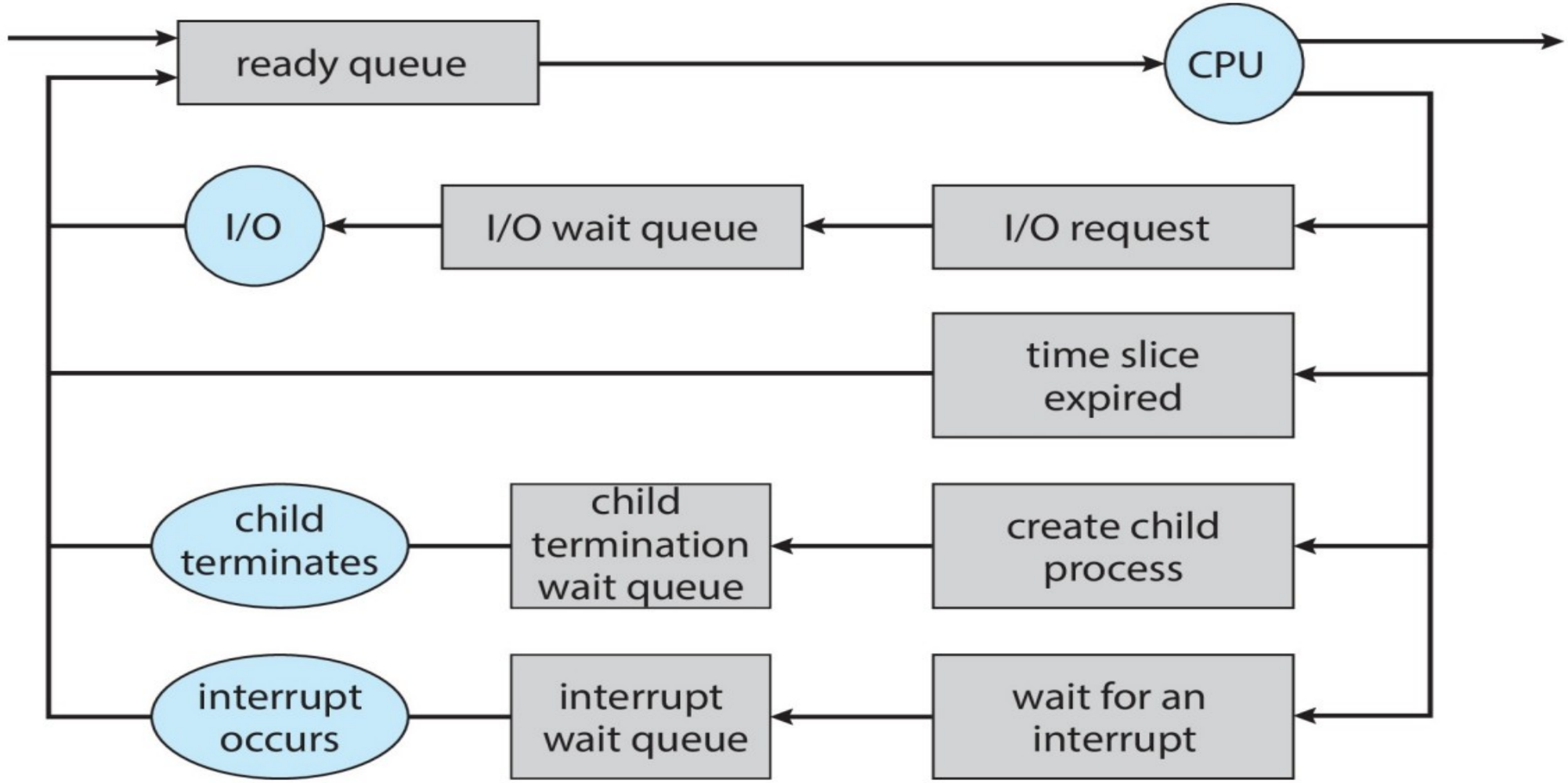




Process Scheduling Queues

- **Job Queue** – When a process enters the system it is put into a job Queue. This queue consists of all processes in the system
- **Ready Queue** – This queue consists of processes that are residing in main memory and are ready and waiting to execute. It is generally stored as a link list
- **Device Queues** – When the process is allocated the CPU, it executes for a while and eventually quits as it may need an I/O. The list of processes waiting for a particular I/O device is called a device queue. Each device has its own device queue
- A process in its life time will be migrating from one Q to another Q

Queuing Diagram



Orphan vs Zombie Process

Orphan Process

- If a parent has terminated before reaping its child, and the child process is still running, then that child is called orphan
- In UNIX all orphan processes are adopted by init or systemd which do the reaping

Zombie Process

- Zombie Process is a process that has terminated but its parent has not collected its exit status and has not reaped it. So a parent must reap its children
- When a process terminates but still is holding system resources like PCB and various tables maintained by OS. It is half-alive & half-dead because it is holding resources like memory but it is never scheduled on the CPU
- The only way to remove them from the system is to kill their parent, at which time they become orphan and adopted by inept or systemd

Threads

Every process has two characteristics:

- **Resource ownership** process includes a virtual address space to hold the process image
- **Scheduling** follows an execution path that may be interleaved with other processes

These two characteristics are treated independently by the operating system. The unit of resource ownership is referred to as a process, while the unit of dispatching is referred to as a thread

A thread is an execution context that is independently scheduled, but shares a single addresses space with other threads of the same process

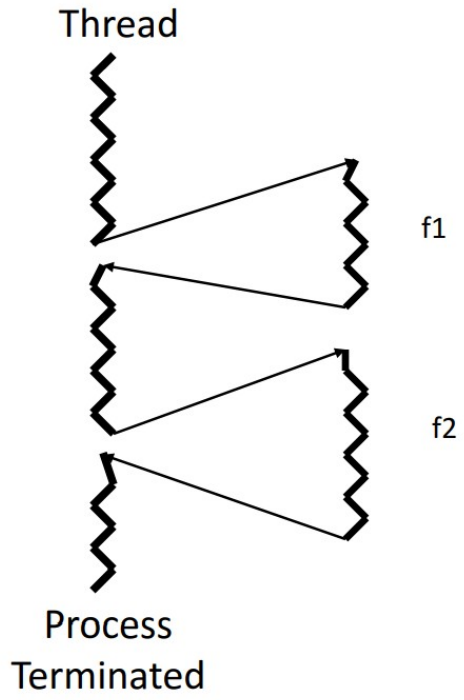
Threads

Single Threaded Process

```
main()
{
      …
      f1(…);
      …
      f2(…);
      …
}

f1(…)
{  …  }

f2(…)
{  …  }
```

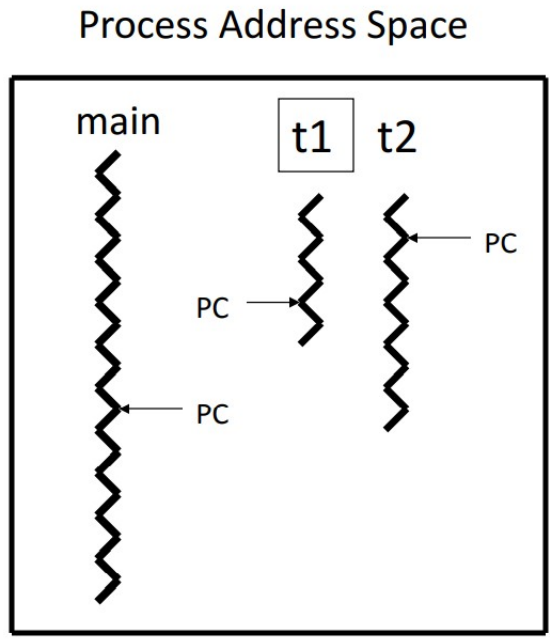


Multithreaded Process

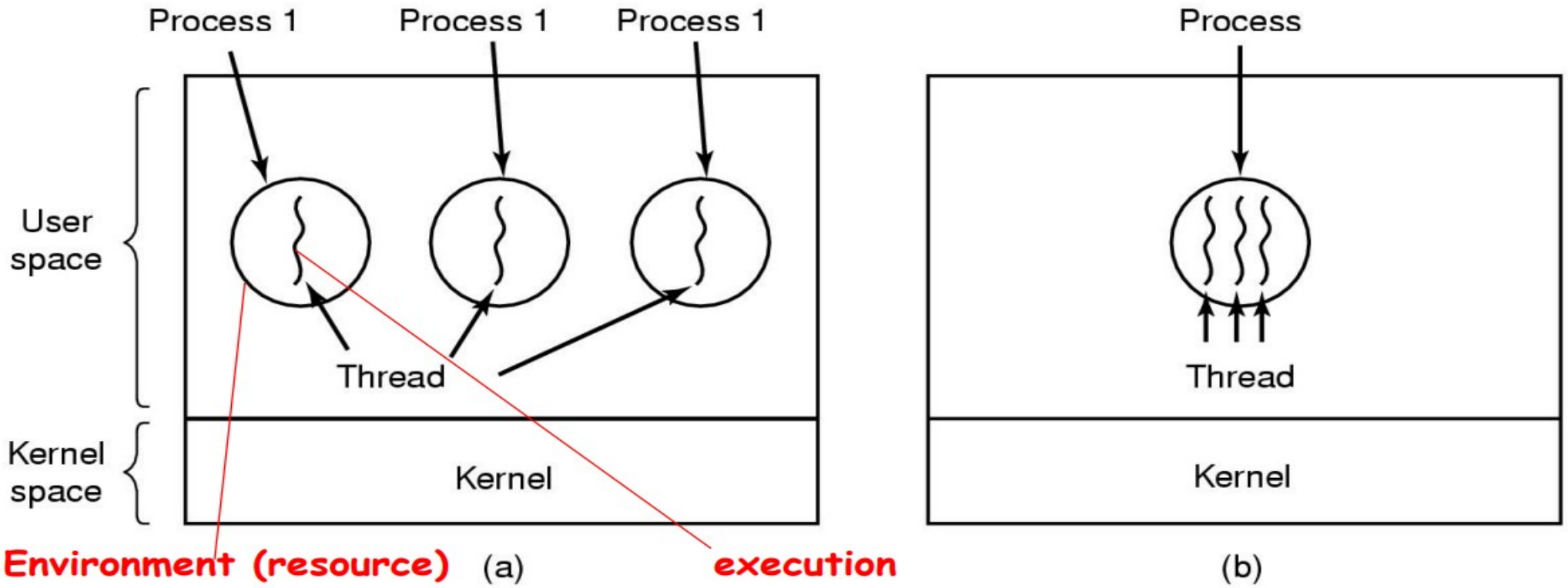
```
main()
{
      …
      thread(t1,f1);
      …
      thread(t2,f2);
      …
}

f1(…)
{  …  }

f2(…)
{  …  }
```



Threads



(a) Three processes, each with one thread
(b) One process with three threads

Processes are used to group resources together; Threads are the entities scheduled for execution on the CPU.

Threads vs Process

Similarities

- A thread can also be in one of many states like new, ready, running, blocked, terminated
- Like processes only one thread is in running state (single CPU)
- Like processes a thread can create a child thread

Differences

- No “automatic” protection mechanism is in place for threads—they are meant to help each other
- Every process has its own address space, while all threads within a process operate within the same address space

Scheduling Algorithms

- FCFS
- SJF
- Priority
- Round Robin

Name some OSs

- UNIX
- Linux
- Windows
- Sun Solaris
- Mac
- Android
- IOS
- Symbian
- BlackBerry

Linux History

UNIX

- In 1969 at Bell Laboratories UNIX was deployed.
- Simple and elegant.
- Written in the C programming language instead of in assembly code.
- Able to recycle code.
- The code recycling features were very important. Until then, all commercially available computer systems were written in a code specifically developed for one system. UNIX on the other hand needed only a small piece of that special code, which is now commonly named the kernel. This kernel is the only piece of code that needs to be adapted for every specific system and forms the base of the UNIX system. The operating system and all other functions were built around this kernel and written in a higher programming language, C

Linux History

1969
→ Birth of **'C'** and **Unix OS**

1

1970's
→ Growth of Unix because of open-source collaboration
→ Commercial sale of Unix

2

1980s
→ Companies developing their own Unix:- **IBM**(AIX), **Solaris**(Sun OS), **HP**(HP-UX)....

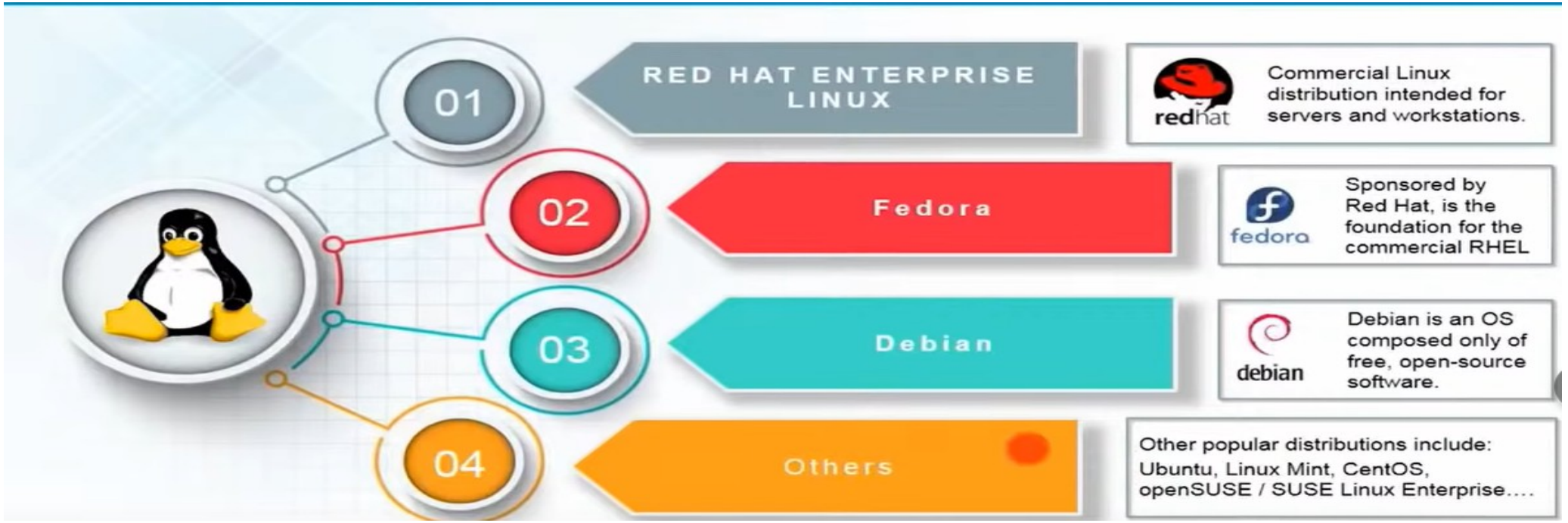
Mid-to-late 1980s
→ Birth of free software movement → **GNU Project**

3

1990s
→ Linus Torvalds put the **Linux kernel** source code online.
→ Resulted in usage of **'Linux + GNU'**

Distributions

- A Linux distribution includes a kernel and a collection of applications. – Linux kernel – Applications (GNU, etc) – Desktop (Gnome, KDE, etc.)
- Dozens of distributions available to suit a large variety of needs. – RedHat, SUSE, Ubuntu, CentOS, Debian



Pros & Cons

Pros

- Open Source
- Kernel is free
- Linux was made to keep on running
- Linux is secure and versatile
- Linux is scalable

Cons

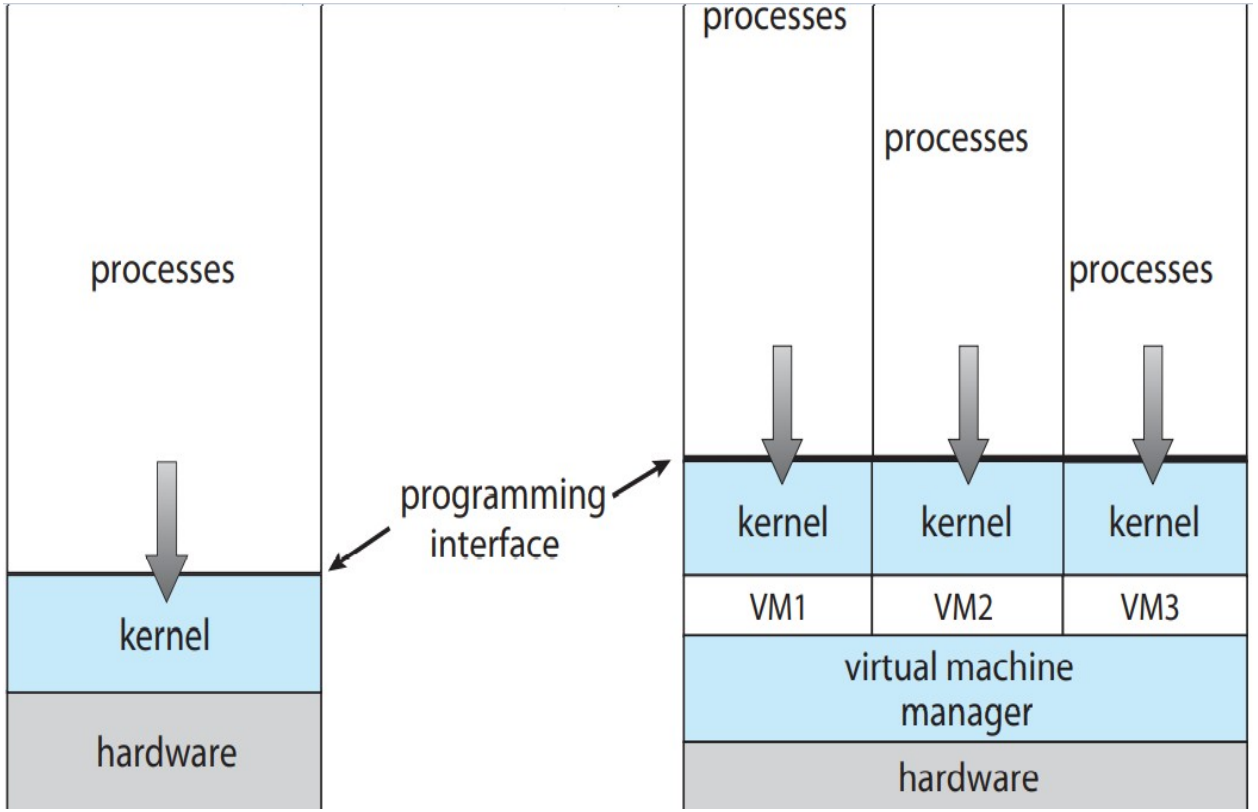
- There are far too many different distributions
- Linux is not very user friendly and confusing for beginners

Configuring Ubuntu for Lab

- Install Ubuntu on your local machine as the host OS. https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview
- Install Ubuntu on WSL on Windows 10 https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-10#1-overview
- Install a Hypervisor (Virtual Box) on your windows machine and install Ubuntu image on the hypervisor. https://www.toptechskills.com/linux-tutorials-courses/how-to-install-ubuntu-1804-bionic-virtualbox/

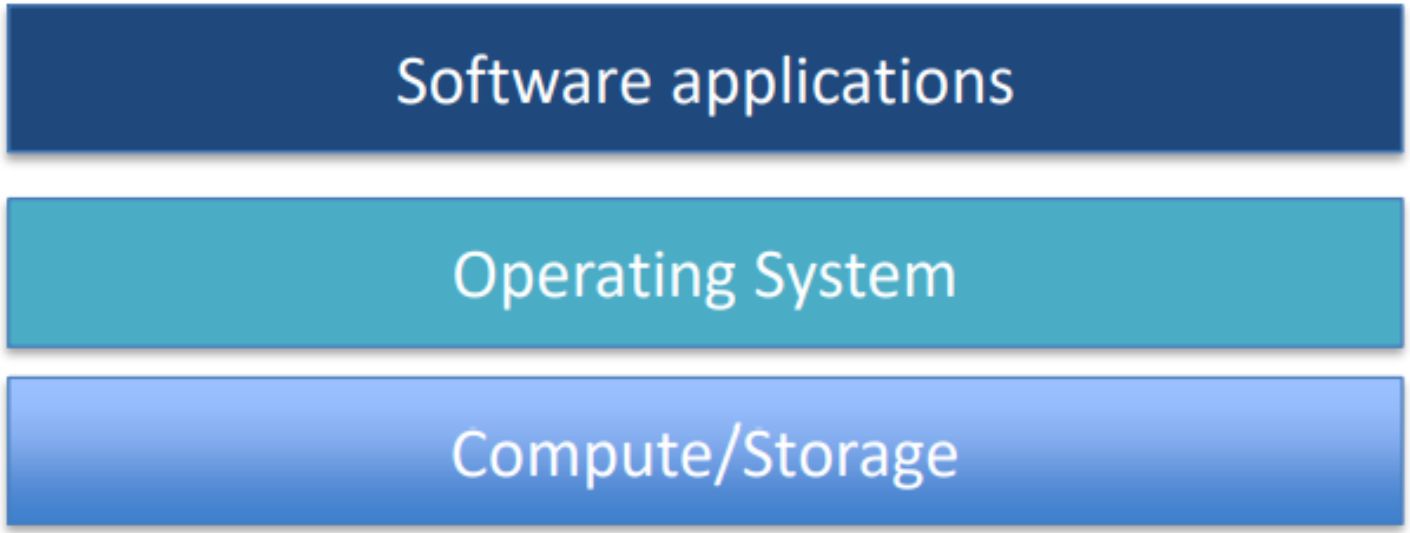
Virtualization

- Virtualization is a framework or methodology of dividing the resources of a computer system into multiple execution environments
- Virtualization is a Technology that transforms hardware into software
- Virtualization allows to run multiple operating systems as virtual machines. Each copy of an operating system is installed in to a virtual machine.



Virtualization

- A stack: combination of infrastructure, OS, and applications required to support a service.



Virtualization Solves Problem

- Servers become as easy to maintain as software.
  – It’s easier to move software around than physical servers.
- No need to purchase a separate server for each user or team because virtualization provides partitioning.

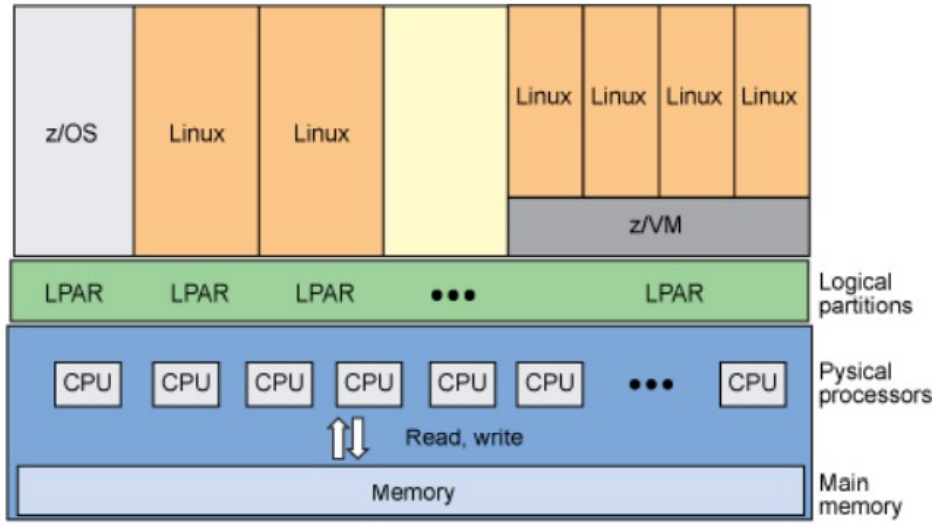
Virtualization

- **Host machine**: physical server that supports multiple virtual machines.
- **Virtual machine**: virtualized server instance running on a host machine (also known as an instance or a workload).
- **Host OS**: operating system running on a bare-metal server.
- **Guest OS**: operating system running on a virtual machine.
- **Hypervisor**: Software that emulates and manages the communication between virtual machines and the physical server.

Virtualization Types

Hardware Virtualization

- Part of physical server dedicated to a specific stack.
- Big expensive mainframe-type systems



OS Partitioning

- Host (or guest) operating system can partition out processes, memory, and scheduling to emulate a separate OS environment.
- Separate OS environments all use same kernel. If host is Linux then all “guests” are Linux.
- Example: Docker containers

Hypervisor Types

Two types available:

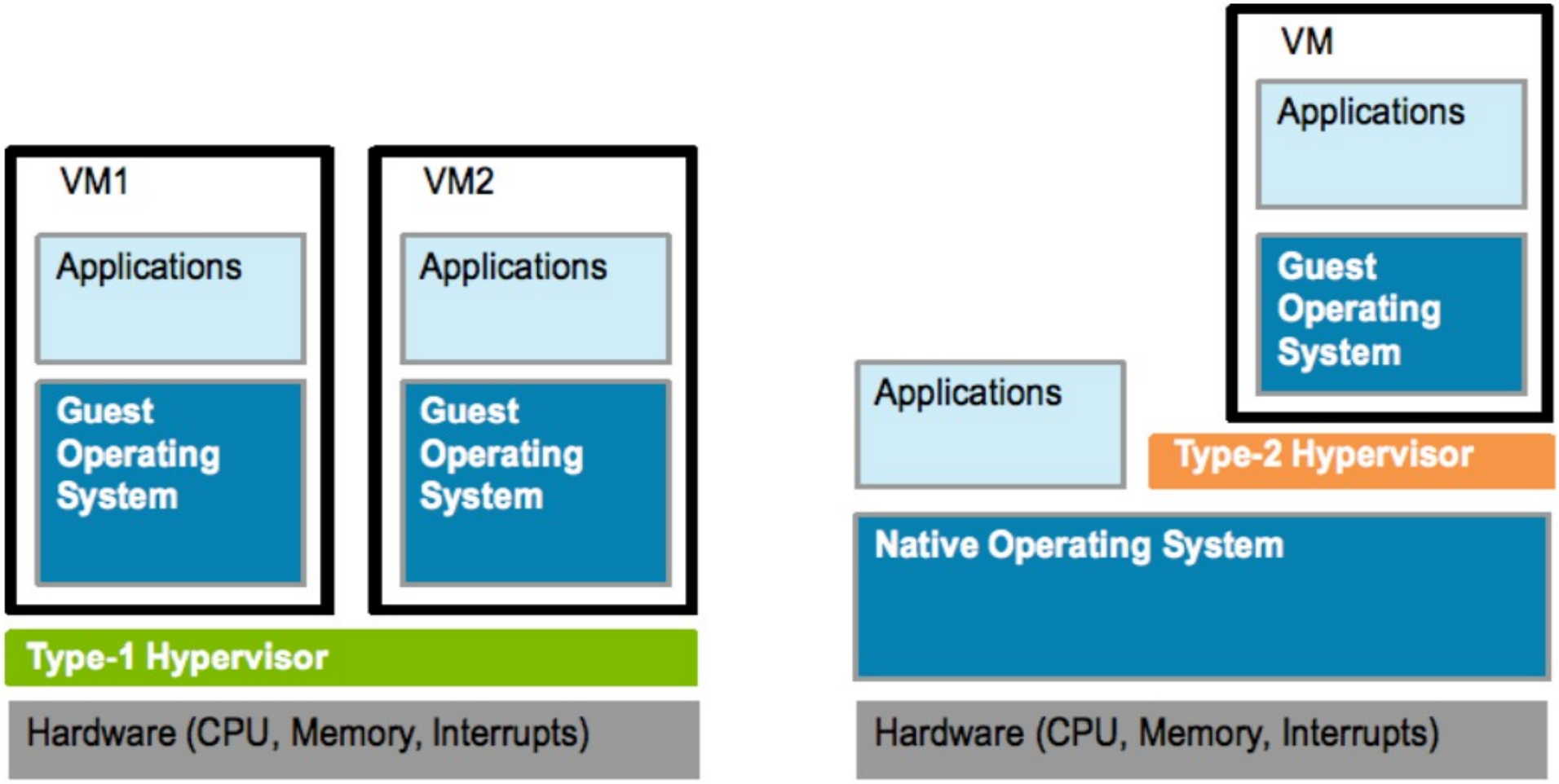
Type 1 Hypervisor

- Runs on top of the bare-metal server and sits between hardware and operating systems in the stacks.
- Examples: VMWare ESX, Microsoft Virtual Server, Xen

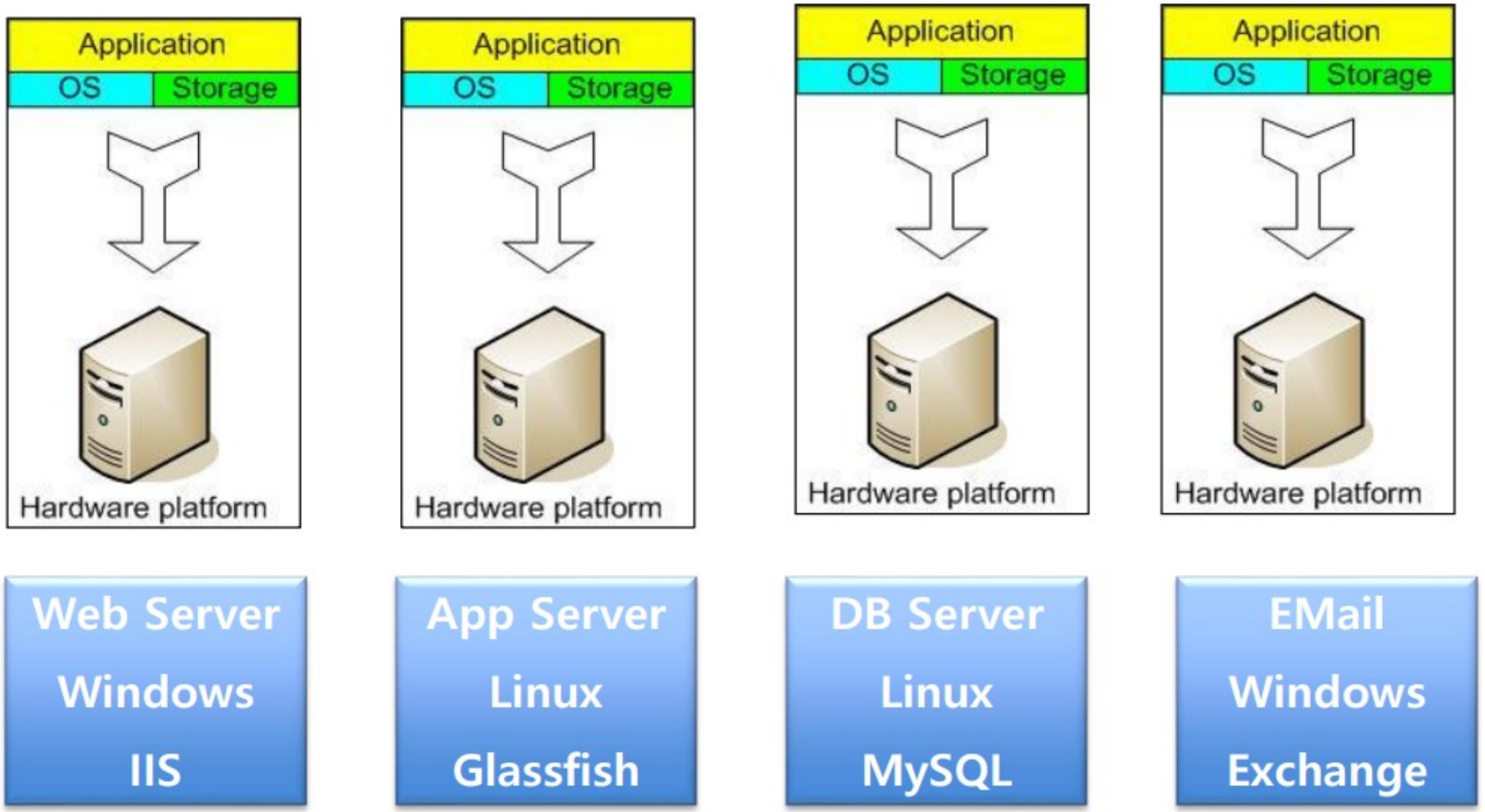
Type 2 Hypervisor

- Runs on top of an operating system.
- Easy installation (no special hypervisor or storage required).
- Example: Oracle VirtualBox, VMWare Fusion

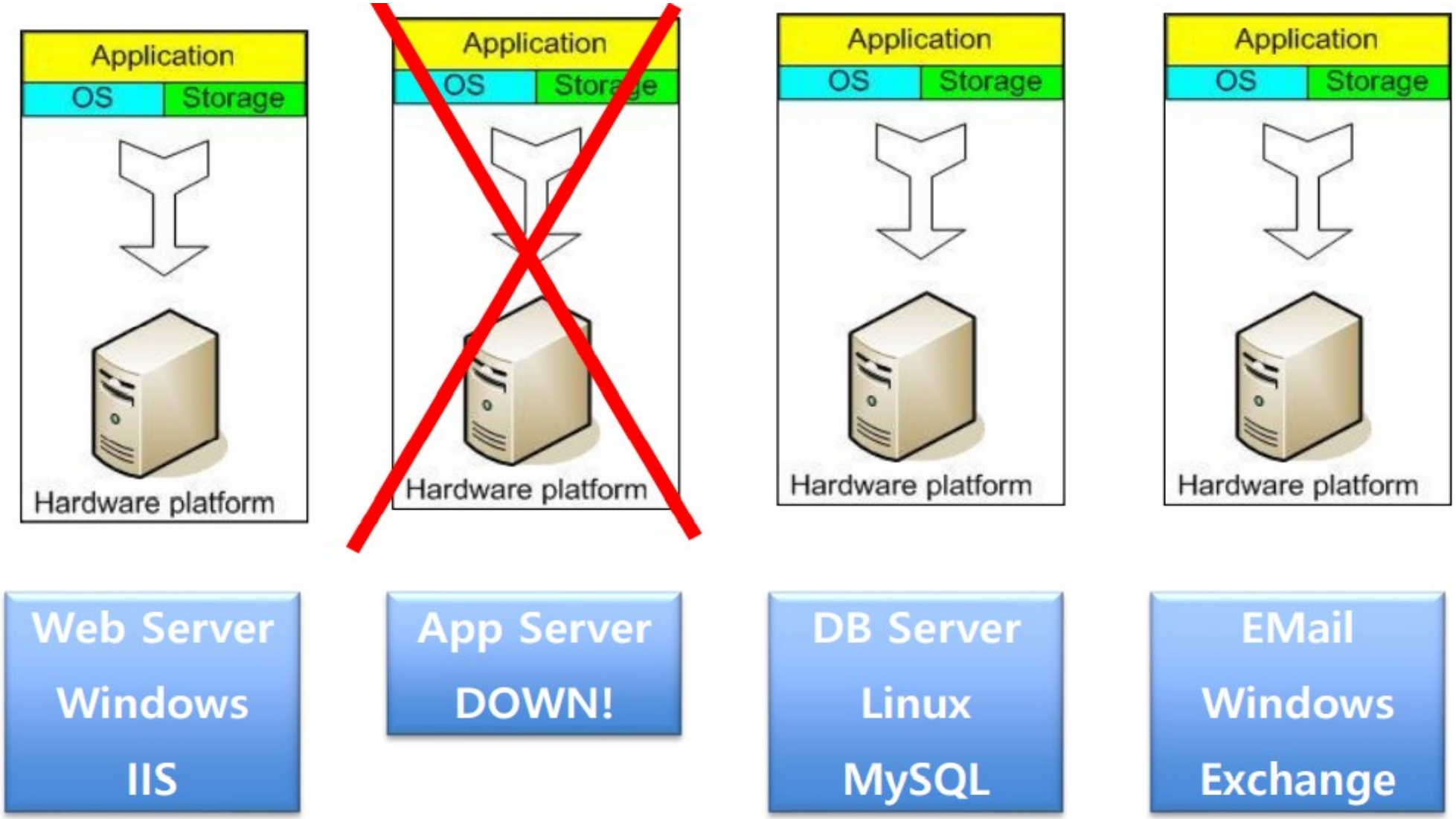
Type 1 vs Type 2 Hypervisor



The Traditional Server Concept



And if something goes wrong



The Traditional Server Concept

Pros

- Easy to conceptualize
- Fairly easy to deploy
- Easy to backup
- Virtually any application/service can be run from this type of setup

Cons

- Expensive to acquire and maintain hardware
- Not very scalable
- Difficult to replicate
- Redundancy is difficult to implement
- Vulnerable to hardware outages
- In many cases, processor is under-utilized

The Virtual Server Concept