



BITS Pilani
Pilani Campus

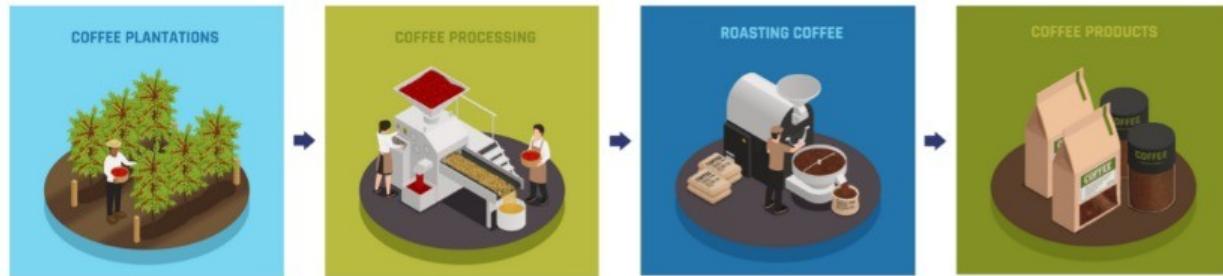
COMPUTING AND DESIGN SESSION 1

Faculty – Thangakumar J
WILP & Department of CS & IS

Today's Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Introduction• Notion of Separation of Concerns• Systems and complexity• Why Modularity, Abstraction, Layering and Hierarchy aren't enough	R3 (Page 24-30) T2 (1.2, 1.3 and 1.5),

Introduction



Coffee time !!!!

Coupling?

- Ask any Engineer and they will explain in detail the importance of *Cohesion* and *Coupling*.
- But *Product Management* and *Design* have often sidelined these concepts as being too engineering-specific. However, in reality, many Product Principles are in fact centered around them.
- Thus unknowingly, we build products and make product decisions based on these very concepts.
- Therefore, let us dive into these core concepts and absorb them so that we can consciously make better product decisions and build better products.

Separation of concerns



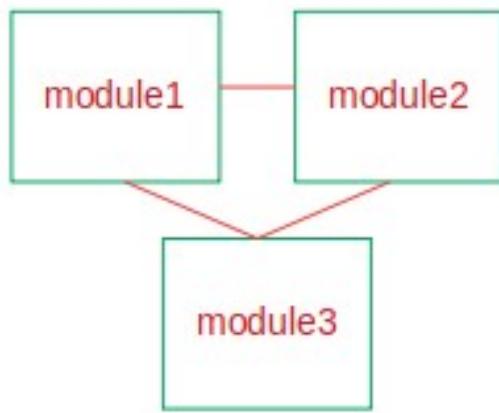
Software system must be decomposed into parts with minimal overlap in functionality. Methodology to follow when decomposing:

Coupling

- The degree of dependency between two modules
- We always want **low** coupling

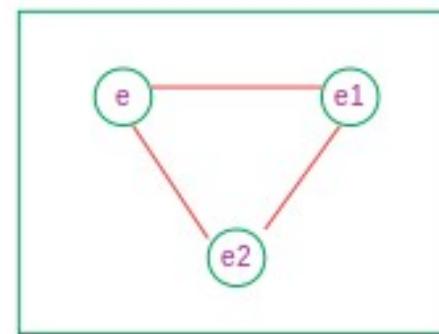
Cohesion

- The measure of how strongly-related is the set of functions performed by a module
- We always want **high** cohesion



COUPLING

Vs

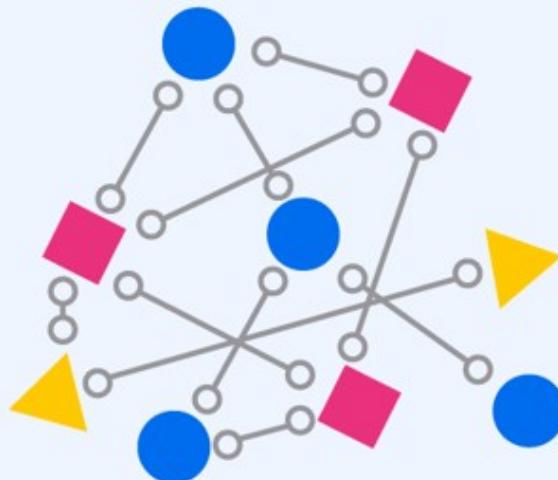


Module

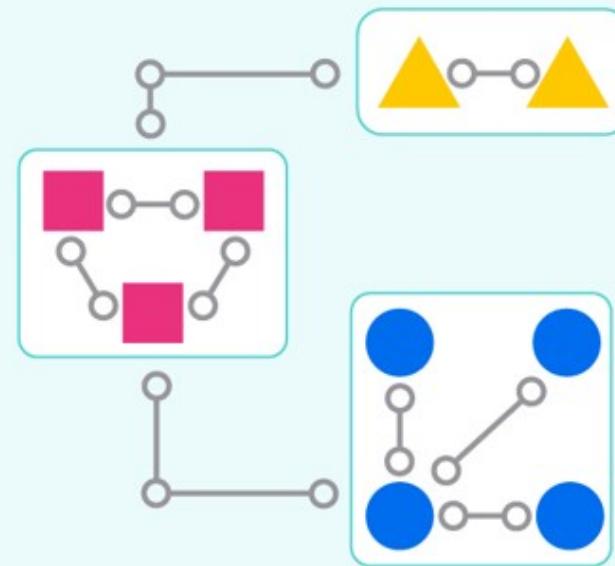
Cohesion

Cohesion + Coupling

Cohesion + Coupling



Without



With

Any product can be broken down into two constituents:

Components: *Individual elements that perform a single task.* In our *Smartphones*, individual components are RAM, ROM, Light Sensor, Lens, Image Processor, CPU, GPU, etc.

Modules: *Groups of functionally similar components.* The above *Smartphone* components are grouped together as Memory(RAM, ROM), Camera(Light Sensor, Lens, Image Processor), System-on-Chip(CPU, GPU), etc.

High Cohesion is when the components of a module are directed towards performing a single task. For example, components of the *Camera* module viz. Light Sensor, Lens, Image Processor are all directed towards performing a single task of capturing images. Similarly, the *Memory* and *Display* modules have *High Cohesion*.

Low cohesion is the exact opposite of High Cohesion. Here, each module ends up performing more than one task since its components are not directed towards a single task. Needless to say, we should always avoid *Low Cohesion*. But, more on that later.

Cohesion:

Module 1



Camera

Capturing Images

Light Sensor, Lens,
Image Processor

Module 2



Processor and Computation

CPU and GPU

Module 3



Memory

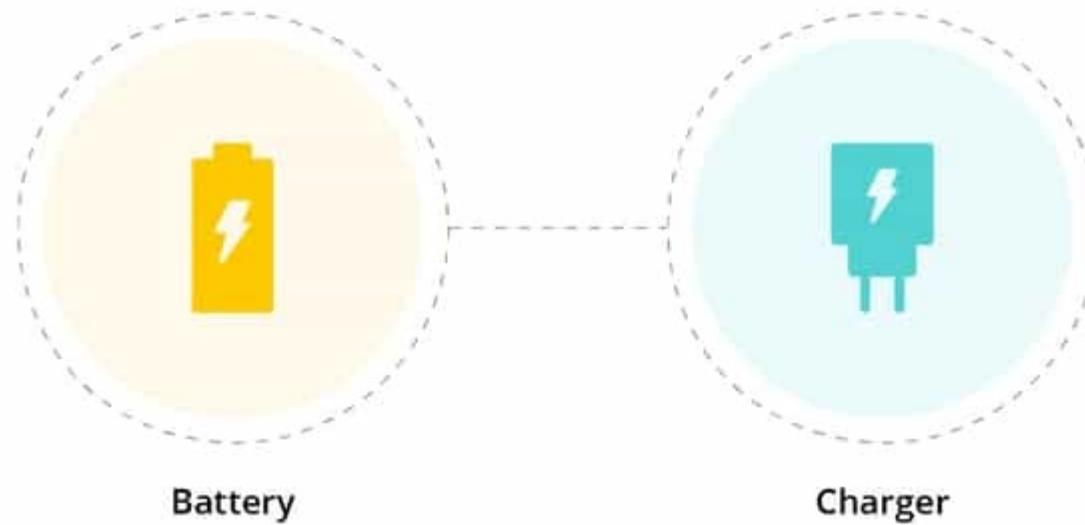
Storage

RAM and ROM

Coupling:

Low/Loose Coupling is when two modules are very lightly dependent on each other to perform their duties. For example, the *Battery* and *Battery-charger* are loosely coupled. If the *Battery-charger* gets corrupted, another *Battery-charger* can be used to charge the *Battery*, and if the *Battery* gets corrupted, then the same *Battery-charger* can still be used to charge other phones batteries.

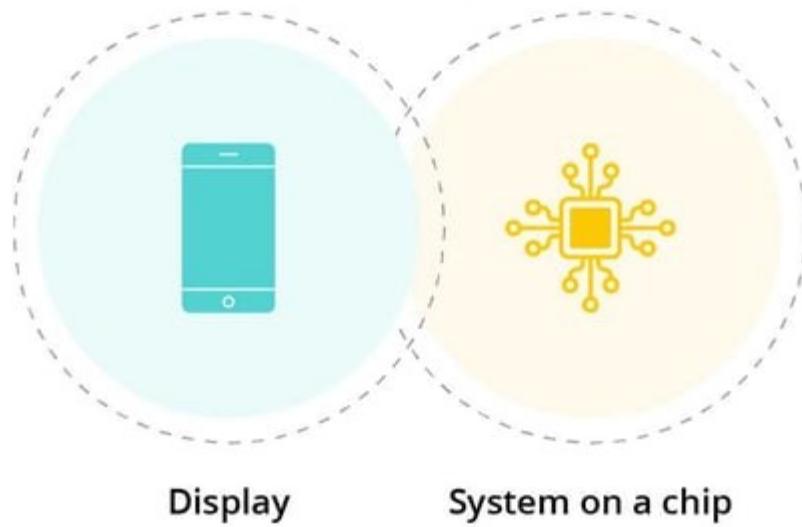
Low Coupling

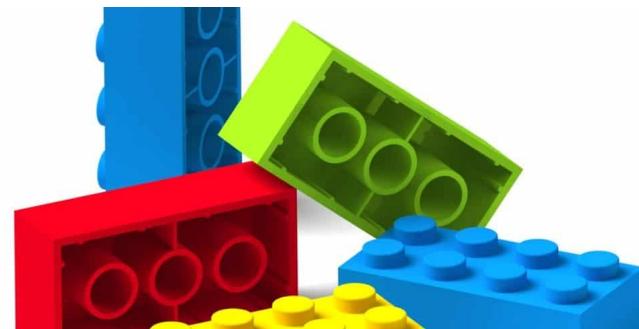
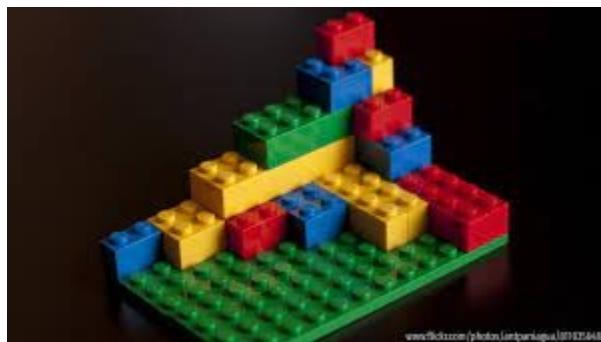


High/Tight Coupling is when two modules are heavily dependent on each other to perform their duties. For example, in contrast to the Battery and Battery charger, the *Display* module and *System-on-Chip* module are more tightly coupled.

If the *System-on-Chip* module gets corrupted, the *Display* module will not be able to perform its duties and if the *Display* gets corrupted, the *System-on-chip (GPU)* will not be able to perform its duties properly.

High Coupling





Separation of concerns - Example of Classes using OOP

Student

- ID
- Name
- Age
- Gender
-

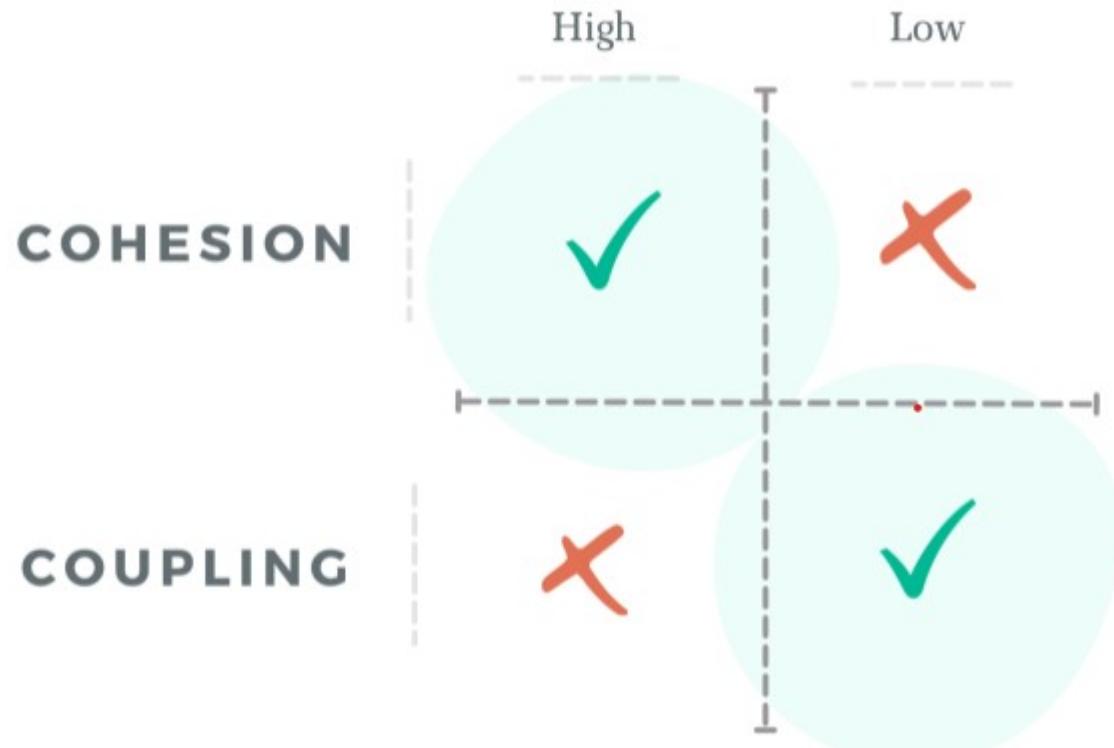
Course

- ID
- Name
- Description
- Units
-

Cohesion :
Coupling :

Everything about Student or Course in respective classes
A change to student class should not impact course class

Better Maintenance ; Reusable; Extensible



In fact, it is a seesaw. If your modules have High Cohesion, you naturally get Low Coupling between them. But, if you have Low Cohesion, your modules will have High Coupling.

SYSTEMS AND COMPLEXITY

- A system is a set of **interconnected components** that has an **expected behavior** observed **at the interface** with its environment.
- Webster's definition of "system" used the word "complex".
- Looking up that term, we find that complex means "difficult to understand".
- A set of signs of complexity that can help confirm a diagnosis here are five signs of complexity:
 - **Large number of components**
 - **Large number of interconnections**
 - **Many irregularities**
 - **A long description**
 - **A team of designers, implementers, or maintainers**

SYSTEMS AND COMPLEXITY (Contd.)

- The university **library** exhibits all five signs of **complexity**, but unanimity is not essential.
 - On the other hand, the presence of only one or two of the signs may not make a compelling case for complexity.
 - Systems considered in **thermodynamics** contain an unthinkable large number of components (elementary particles) and **interactions**, yet from the right point of view they do not qualify as complex because there is a simple, methodical description of their behavior.
 - It is exactly when **we** lack such a simple, methodical description that **we** have **complexity**.
-

SYSTEMS AND COMPLEXITY (Contd.)

- One objection to conceiving complexity as being based on the five signs is that all systems are indefinitely, perhaps infinitely, complex because the deeper one digs the more signs of complexity turn up.
- Thus, even the simplest digital computer is made of gates, which are made with transistors, which are made of silicon, which is composed of protons, neutrons, and electrons, which are composed of quarks, which some physicists suggest are describable as vibrating strings, and so on.

And our coffee machine is not complex!

SOURCES OF COMPLEXITY - Cascading and Interacting Requirements



- There are many sources of complexity, but two merit special mention.
 - The first is in the **number of requirements** that the designer expects a system to meet.
 - The second is one particular requirement: **maintaining high utilization**.
- The underlying cause of this scenario is that the personal computer has been designed to meet many requirements: a well-organized file system, expandability of storage, ability to attach a variety of I/O devices, connection to a network, protection from malevolent persons elsewhere in the network, usability, reliability, low cost—the list goes on and on.
- Each of these **requirements adds complexity** of its own, and the interactions among **them add still more complexity**.

SOURCES OF COMPLEXITY - Cascading and Interacting Requirements

- The principle is subjective because complexity itself is subjective—its magnitude is in the mind of the beholder.
- Figure 1.1 provides a graphical interpretation of the principle.

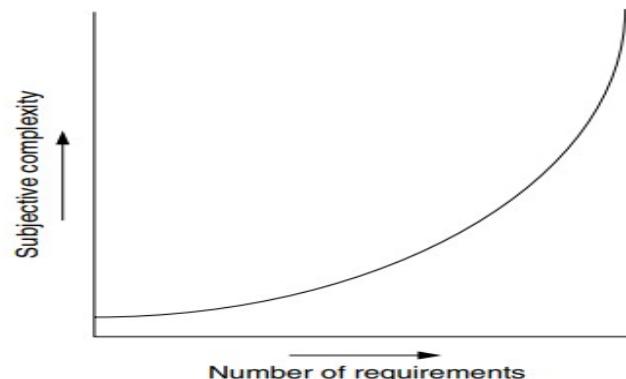


FIGURE 1.1

The principle of escalating complexity.

Maintaining High Utilization

- One requirement by itself is frequently a **specific source of complexity**.
- It starts with a desire for high performance or high efficiency.
- Whenever a **scarce resource is involved, an effort arises to keep its utilization high**.
- As a general rule, the more one tries to increase utilization of a limited resource, the greater the complexity (see Figure 1.2).
- The perceptive reader will notice that Figures 1.1 and 1.2 are identical.

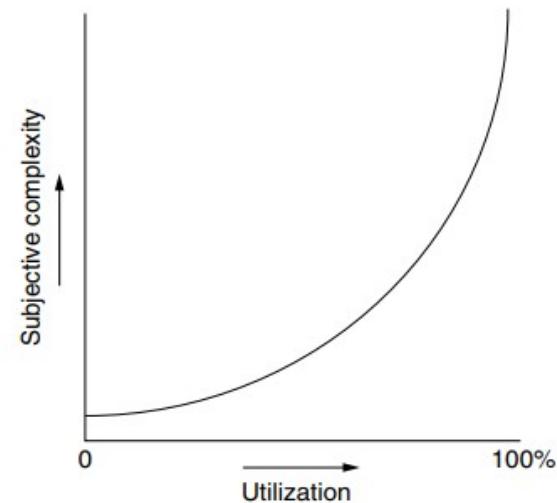


FIGURE 1.2

An example of diminishing returns: complexity grows with increasing utilization.

COPING WITH COMPLEXITY 1 - Modularity

- The simplest, most important tool for reducing complexity is the **divide-and-conquer technique**: analyze or design the system as a collection of interacting subsystems, called modules.
- The power of this technique lies primarily in being able to **consider interactions among the components** within a module without simultaneously thinking about the components that are inside other modules.
- To see the impact of reducing interactions,
 - consider the debugging of a large program with, say, N statements.
 - Assume that the number of bugs in the program is proportional to its size and the bugs are randomly distributed throughout the code.
 - The programmer compiles the program, runs it, notices a bug, finds and fixes the bug, and recompiles before looking for the next bug.
 - Assume also that the time it takes to find a bug in a program is roughly proportional to the size of the program.
- We can then model the time spent debugging:

$$\text{BugCount} \sim N$$

$$\text{DebugTime} \sim N * \text{BugCount}$$

$$\sim N^2$$

COPING WITH COMPLEXITY 1 - Modularity

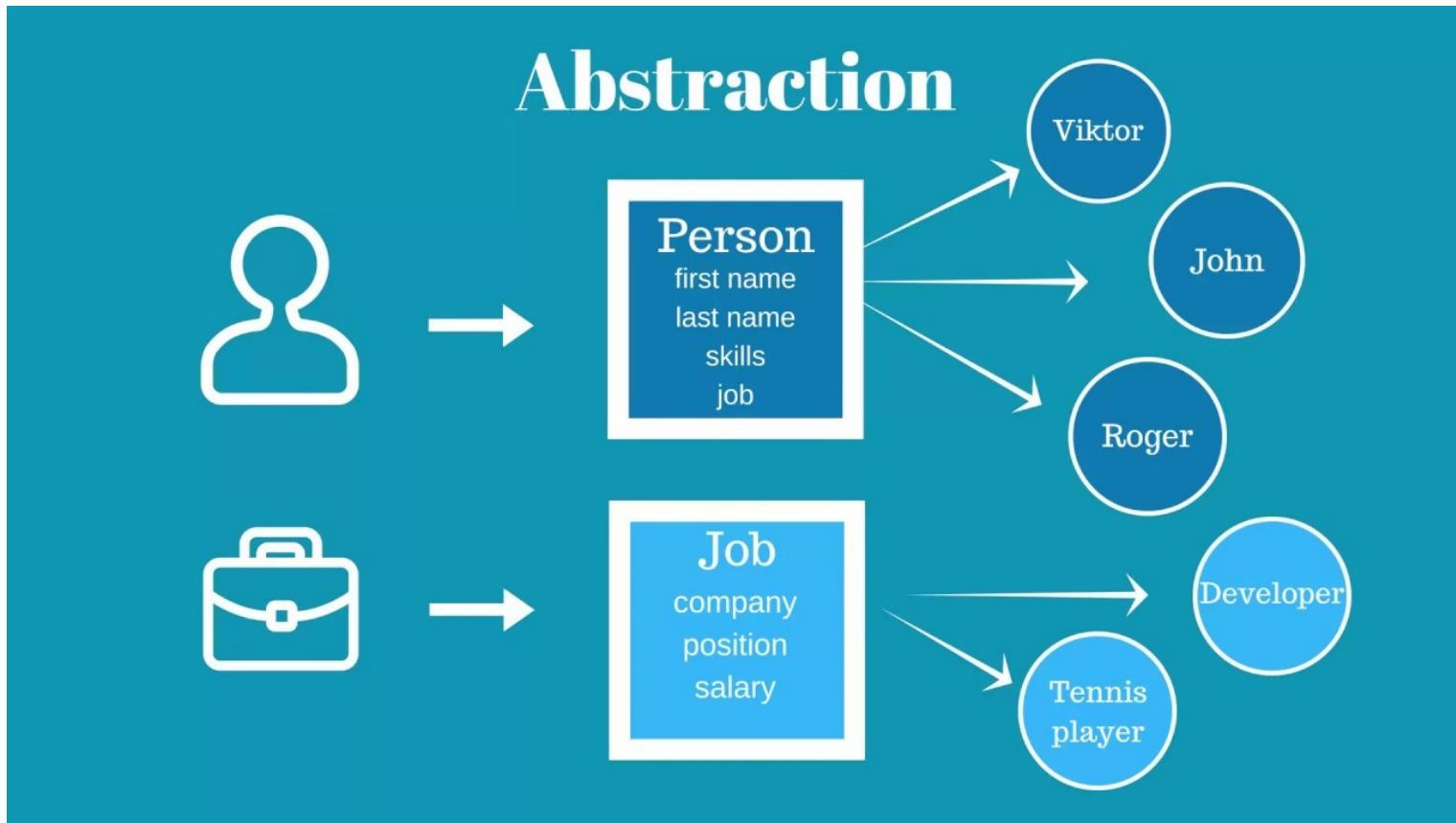
- Unfortunately, the **debugging time grows proportional** to the square of the program size.
- Now suppose that the programmer divides the program into K modules, each of roughly equal size, so that each module contains N/K statements.
- To the extent that the modules implement independent features, one hopes that discovery of a bug usually **will require examining only one module**.
- The time required to debug any one module is thus reduced in two ways: the smaller module can be debugged faster, and since there are fewer bugs in smaller programs, any one module will not need to be debugged as many times.
- These two effects are partially offset by the need to debug all K modules.
- Thus our model of the time required to debug the system **of K modules** becomes

$$\begin{aligned}\text{DebugTime} &\sim (N/K)^2 * K \\ &\sim N^2/ K\end{aligned}$$

COPING WITH COMPLEXITY 1 - Modularity

- Modularization into K components thus reduces debugging time by a factor of K.
- Although the detailed mechanism by which modularity reduces effort differs from system to system, this property of modularity is universal.
- For this reason, one finds modularity in every large system.
- The feature of modularity that we are taking advantage of here is that it is easy to replace an inferior module with an improved one, thus allowing incremental improvement of a system without completely rebuilding it.
- Modularity thus helps control the complexity caused by change.
- This feature applies not only to debugging but to all aspects of system improvement and evolution.
- The reason is that once an interface has been used by another module, changing the interface requires replacing at least two modules.
- If an interface is used by many modules, changing it requires replacing all of those modules simultaneously. For this reason, it is particularly important to get the modularity right.

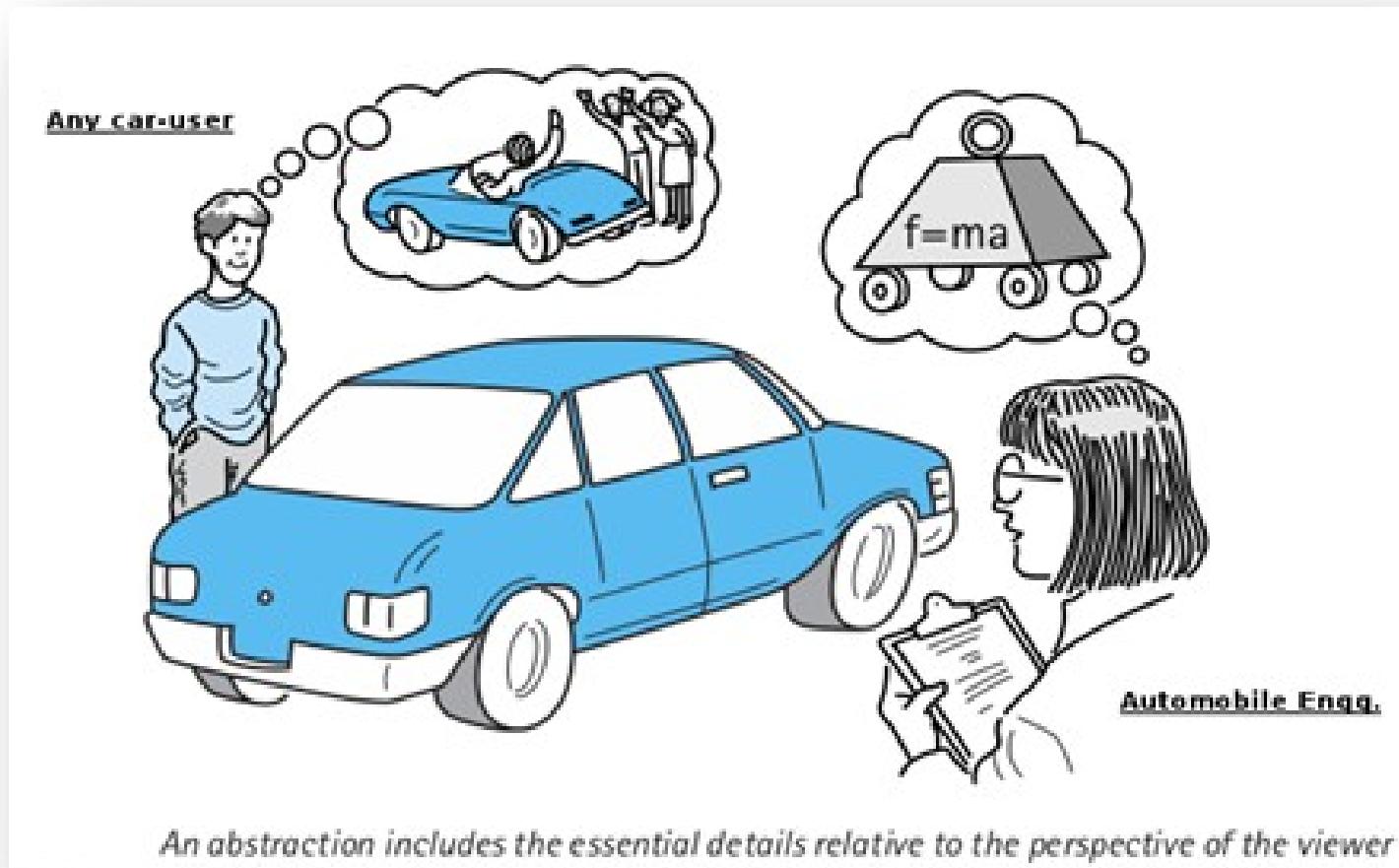
Abstraction



What does the above abstract picture notify?

Abstraction can be found in practically all real-world machines.

- Your automobile is an excellent illustration of abstraction. Turning the key or pressing the start button is how you start a car. You don't need to know how your car's engine starts or what components it contains. The user is completely unaware of the car's internal implementation and complicated logic.
 - We can heat our food in Microwave. We press some buttons to set the timer and type of food. Finally, we get a hot and delicious meal. The microwave internal details are hidden from us. We have been given access to the functionality in a very simple manner.
-



Types of Abstraction

- There are two types of abstraction.
- Data Abstraction
- Process Abstraction
- Data Abstraction

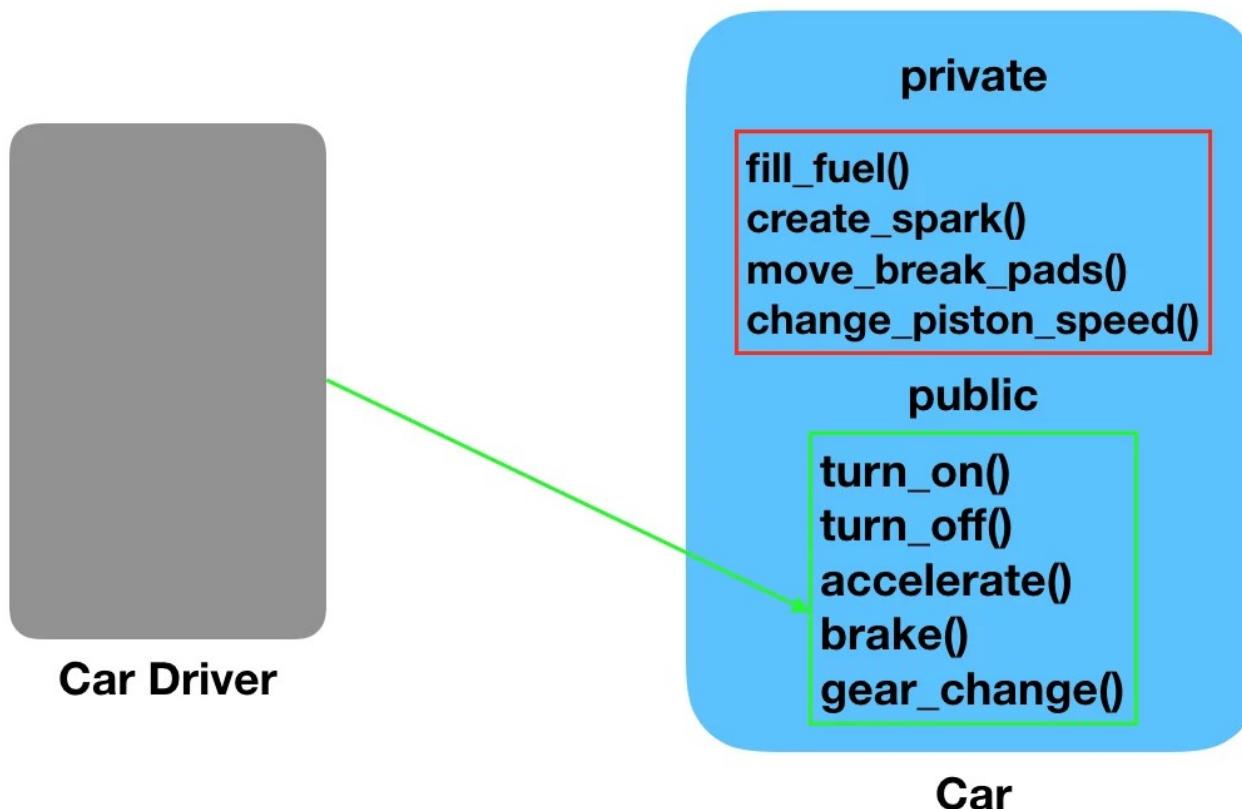
- Full Name
- Address
- Contact Number
- Tax Information
- Favorite Food
- Favorite Movie
- Favorite Actor
- Favorite Band

Okay, we might not need all these customer information for a banking application

When the object data is not visible to the outer world, it creates data abstraction. If needed, access to the Objects' data is provided through some method

Process abstraction

Process Abstraction



Process Abstraction

- A piece of software is essentially a set of statements written in any programming language. The majority of the time, statements are similar and are repeated in several locations.
- The process of finding all such assertions and exposing them as a unit of work is known as process abstraction. When we build a function to execute any task, we usually use this feature.

Abstraction

- An important assumption in the numerical example of the effect of modularity on debugging time may not hold up in practice: that discovery of a bug should usually lead to examining just one module.
 - Abstraction is separation of interface from internals, of specification from implementation.
 - Because abstraction nearly always accompanies modularity, some authors do not make any distinction between the two ideas.
 - One sometimes sees the term functional modularity used to mean modularity with abstraction.
 - The general ability of sequential circuits to remember state is abstracted into particular, easy-to-describe modules called registers.
 - Programs are designed to hide details of their representation of complex data structures and details of which other programs they call.
-

COPING WITH COMPLEXITY 1 - Modularity

- The goal of minimizing interconnections among modules may be defeated if unintentional or accidental interconnections occur as a result of implementation errors
- Software is particularly subject to this problem because the modular boundaries provided by separately compiled subprograms are somewhat soft and easily penetrated by errors in using pointers, filling buffers, or calculating array indices.
- For this reason, system designers prefer techniques that enforce modularity by interposing impenetrable walls between modules.
- The robustness principle is one of the key ideas underlying modern mass production.
- The breakthrough came with the realization that if one specified tolerances for components and designed each component to mate with any other component that was within its specified tolerance, then it would be possible to modularize and speed up manufacturing by having interchangeable parts.

COPING WITH COMPLEXITY 1 - Modularity

- Some systems implement the safety margin principle by providing two modes of operation, which might be called “shake-out” and “production”.
 - In shake-out mode, modules check every input carefully and refuse to accept anything that is even slightly out of specification, thus allowing immediate discovery of problems and of programming errors near their source.
 - In production mode, modules accept any input that they can reasonably interpret, in accordance with the robustness principle.
- Carefully designed systems blend the two ideas: accept any reasonable input but report any input that is beginning to drift out of tolerance so that it may be repaired before it becomes completely unusable.

Layering

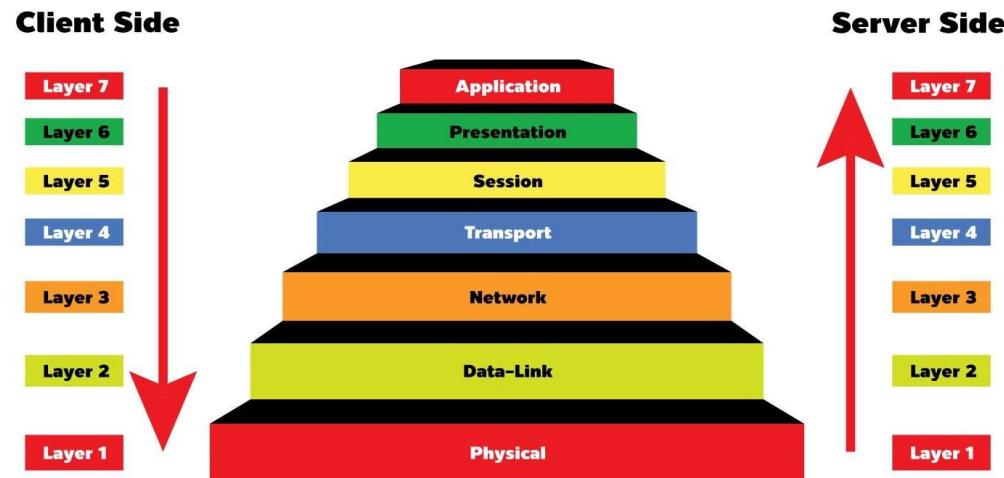
- One powerful way to reduce module interconnections is to employ a particular method of module organization known as layering.
- In designing with layers, one builds on a set of mechanisms that is already complete (a lower layer) and uses them to create a different complete set of mechanisms (an upper layer).
- A layer may itself be implemented as several modules, but as a general rule, a module of a **given layer** interacts only with its peers in the same layer and with the modules of the next higher and next lower layers.
- That restriction can significantly **reduce the number of potential inter-module interactions in a big system.**
- Some of the best examples of this approach are found in computer systems: **an interpreter for a high-level language is implemented using a lower-level, more machine-oriented, language.**

COPING WITH COMPLEXITY 1 - Modularity

- Thus, nearly every computer system comprises several layers.
- The lowest layer consists of gates and memory cells, upon which is built a layer consisting of a processor and memory.
- On top of this layer is built an operating system layer, which acts as an augmentation of the processor and memory layer.
- Finally, an application program executes on this augmented processor and memory layer.
- In each layer, the functions provided by the layer below are rearranged, repackaged, re-abstracted, and reinterpreted as appropriate for the convenience of the layer above.

Level 6	User	Executable Programs
Level 5	High Level Language	C++ , Java
Level 4	Assembly Language	Assembly Code
Level 3	System Software	Operating System
Level 2	Machine	Instruction Set Architecture
Level 1	Control	Microcode
Level 0	Digital Logic	Circuits , Gates

OSI MODEL



- Computer System Level Hierarchy is the combination of different levels that connects the computer with the user and that makes the use of the computer. It also describes how the computational activities are performed on the computer and it shows all the elements used in different levels of system.

- **Level-0:**

It is related to digital logic. Digital logic is the basis for digital computing and provides a fundamental understanding of how circuits and hardware communicate within a computer. It consists of various circuits and gates etc.

- **Level-1:**

This level is related to control. Control is the level where microcode is used in the system. Control units are included in this level of the computer system.

- **Level-2:**

This level consists of machines. Different types of hardware are used in the computer system to perform different types of activities. It contains instruction set architecture.

- **Level-3:**

System software is a part of this level. System software is of various types. System software mainly helps in operating the process and it establishes the connection between hardware and user interface. It may consist operating system, library code, etc.

- **Level-4:**

Assembly language is the next level of the computer system. The machine understands only the assembly language and hence in order, all the high-level languages are changed in the assembly language. Assembly code is written for it.

- **Level-5:**

This level of the system contains high-level language. High-level language consists of C++, Java, FORTRAN, and many other languages. This is the language in which the user gives the command.

- **Level-6:**

This is the last level of the computer system hierarchy. This consists of users and executable programs.

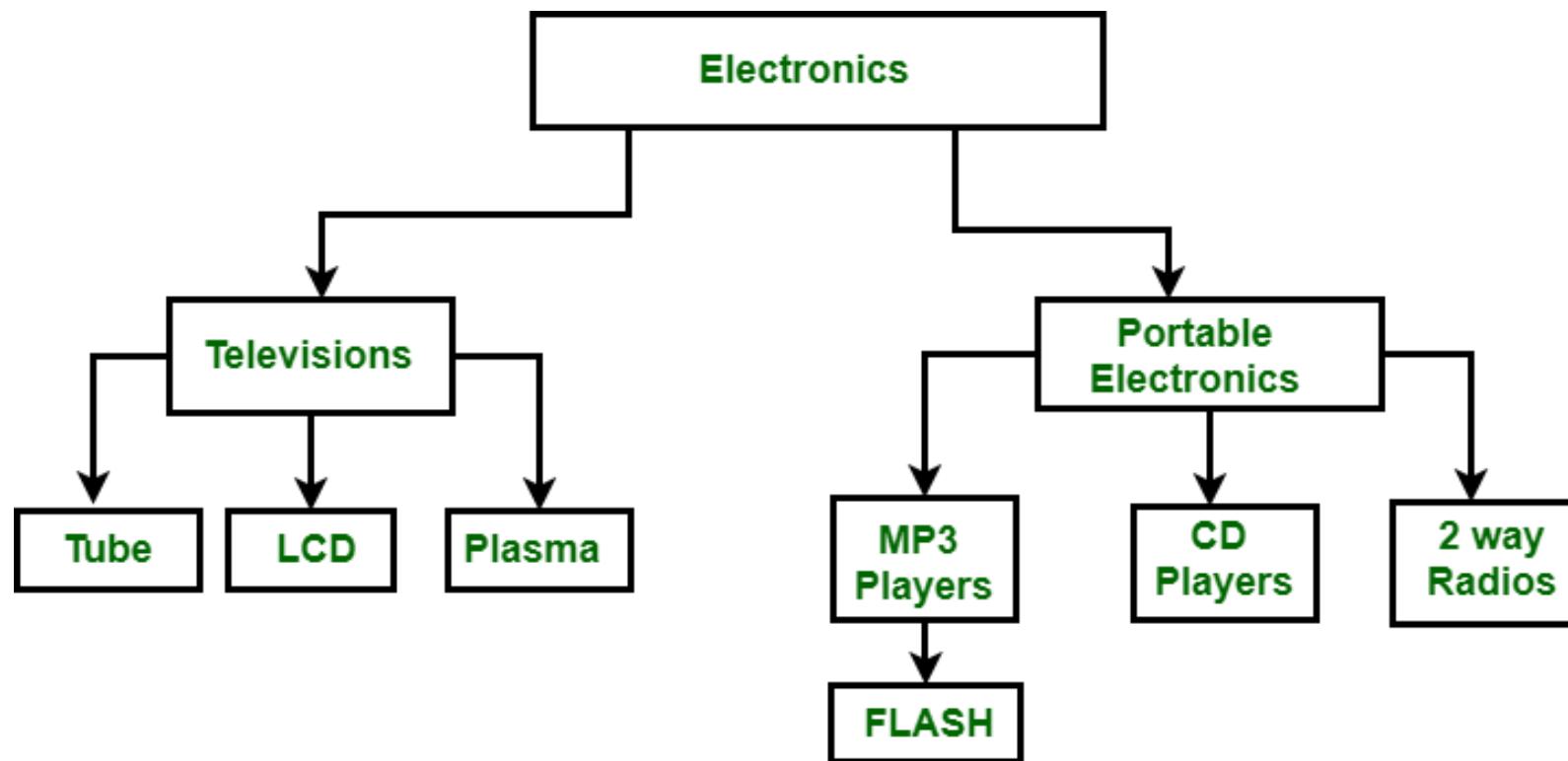
Hierarchy

- The final major technique for coping with complexity also reduces interconnections among modules but in a different, specialized way.
- Start with a small group of modules, and assemble them into a stable, self-contained subsystem that has a well-defined interface.
- Next, **assemble a small group of subsystems to produce a larger subsystem.**
- This process continues until the final system has been constructed from a small number of relatively large subsystems.
- The **result is a tree-like structure known as a hierarchy.**
- Hierarchy constrains a system of **N components, which in the worst case might exhibit $N * (N - 1)$ interactions**, so that each component can interact only with members of its own subsystem, except for an interface component that also interacts with other members of the subsystem at the next higher level of hierarchy.

COPING WITH COMPLEXITY 1 - Hierarchy

- Analogous to the way that modularity reduces the effort of debugging, hierarchy reduces the number of potential interactions among modules from square-law to linear.
- This effect is most strongly noticed by the designer of an individual module.
- If there are no constraints, each module should in principle be prepared to interact with every other module of the system.
- The advantage of a hierarchy is that the module designer can focus just on interactions with the interfaces of other members of its immediate subsystem.

Hierarchy



COPING WITH COMPLEXITY II

- Modest physical limits in hardware and very distant physical limits in software together give us the opportunity to create systems of unimaginable—and unmanageable— complexity, and the rapid pace of technology change tempts designers to deliver systems using new and untested ground rules.
 - These two effects amplify the complexity of computer systems when compared with systems from other engineering areas.
 - Thus, computer system designers need some additional tools to cope with complexity.
-

Why modularity, abstraction, layering, and hierarchy aren't enough

- Modularity, abstraction, layering, and hierarchy are a major help, but by themselves they aren't enough to keep the resulting complexity under control.
 - The right modularity from a sea of plausible alternative modularities
 - The right abstraction from a sea of plausible alternative abstractions
 - The right layering from a sea of plausible alternative layerings
 - The right hierarchy from a sea of plausible alternative hierarchies
- Although some design principles are available, they are far too few, and the only real guidance comes from experience with previous systems.

Iteration

- The essence of iteration is to start by building a simple, working system that meets only a modest subset of the requirements and then evolve that system in small steps to gradually encompass more and more of the full set of requirements.
- The idea is that small steps can help reduce the risk that complexity will overwhelm a system design.
- Having a working system available at all times helps provide assurance that something can be built and provides on-going experience with the current technology ground rules as well as an opportunity to discover and fix bugs.
- Finally, adjustments for technology changes that arrive during the system development are easier to incorporate as part of one or more of the iterations.

Iteration

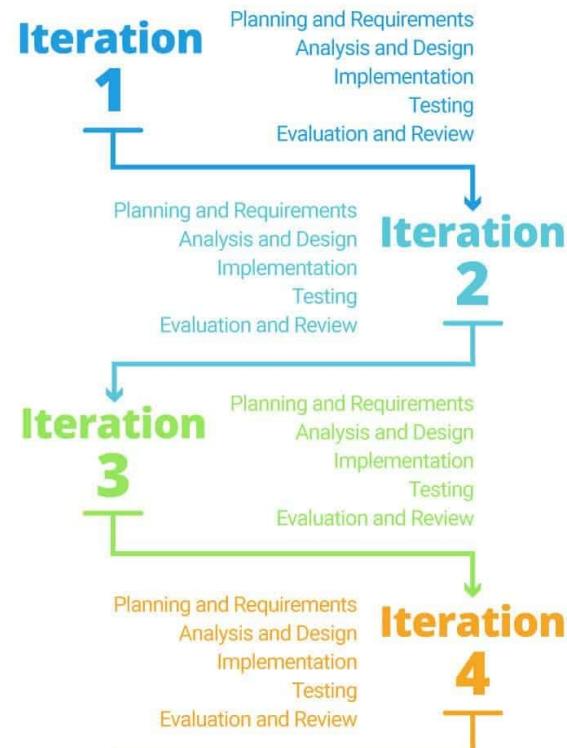
- Document the assumptions behind the design so that when the time comes to change the design you can more easily figure out what else has to change. Expect not only to modify and replace modules, but also to re-modularize as the system and its requirements become better understood.
 - Take small steps
 - Don't rush
 - Plan for feedback
 - Study failures
- Iteration sounds like a straightforward technique, but several obstacles tend to interfere with it. **The main obstacle is that as a design evolves through a series of iterations, a risk of losing conceptual integrity arises.**
- In most organizations, good news flows rapidly throughout the organization, but bad news often gets confined to the part of the organization that discovers it, at least until it can fix the problem and report good news.

Iteration

- This phenomenon, the bad-news diode, can prevent realization that changing a different part of the system is more appropriate.
- A longer-term risk of iteration sometimes shows up when the initial design is both simple and successful. Success can lead designers to be overconfident and to be too ambitious on a later iteration.
- Technology has improved in the time since deployment of the initial version of the system and feedback has suggested lots of new features.
- Each suggested feature looks straightforward by itself, and it is difficult to judge how they might interact.
- The result is often a disastrous overreaching and consequent failure that is so common that it has a name: the second-system effect.
- Iteration can be thought of as applying modularity to the management of the system design and implementation process.

Iterative Process

These five steps can be repeated as many times as needed.



- **Step One: Planning and Requirements:** In this stage, map out the initial requirements, gather the related documents, and create a plan and timeline for the first iterative cycle.
- **Step Two: Analysis and Design:** Finalize the business needs, database models, and technical requirements based on the plan. Create a working architecture, schematic, or algorithm that satisfies your requirements.
- **Step Three: Implementation:** Develop the functionality and design required to meet the specifications.
- **Step Four: Testing:** Identify and locate what's not working or performing to expectations. Stakeholders, users, and product testers weigh in with their experience.
- **Step Five: Evaluation and Review:** Compare this iteration with the requirements and expectations.

- After you complete these steps, it's time to tackle the next cycle. In the iterative process, the product goes back to step one to build on what's working. Identify what you learned from the previous iteration.
- This iterative development, sometimes called *circular* or *evolutionary development*, is based on refining the first version through subsequent cycles, especially as you gather and include requirements. It allows you to remain flexible as you identify new needs or unexpected business issues.

Keep it simple

- Remarkably, one of the most effective techniques in coping with complexity is also one that is most difficult to apply: simplicity.
 - Previous systems give a taste of how great things could be if more features were added.
 - the technology has improved so much that cost and performance are not constraints.
 - each of the suggested new features has been successfully demonstrated somewhere.
 - none of the exceptions or other complications seems by itself to be especially hard to deal with.
 - there is fear that a competitor will market a system that has even more features.
 - among system designers, arrogance, pride, and overconfidence are more common than clear awareness of the dangers of complexity.



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 2

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Thangakumar J
WILP & Department of CS & IS



Last Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Introduction• Notion of Separation of Concerns• Systems and complexity• Why Modularity, Abstraction, Layering and Hierarchy aren't enough	R3 (Page 24-30) T2 (1.2, 1.3 and 1.5),

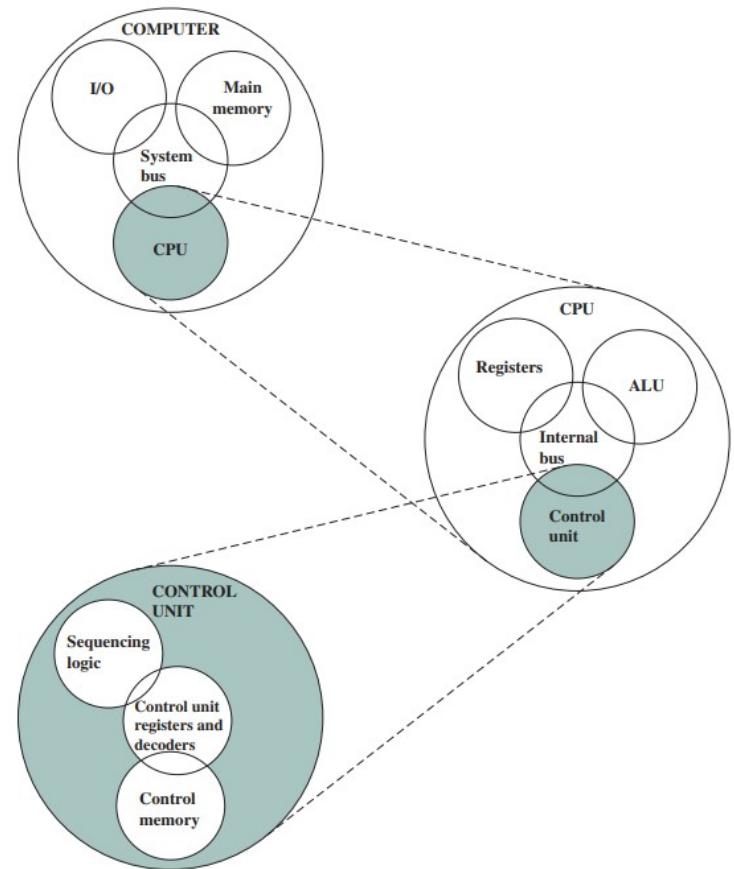
ABSTRACTION & THE NOTION OF INTERFACE

Today's topic

- Major Modules of a Typical Computer System: Processor(s), Memory, I/O and Storage Devices
- Hardware abstraction vs. Software
- The three fundamental Abstractions
 - Memory
 - Interpreters
 - Communication links
- Organizing Computer Systems
 - A Hardware Layer : The Bus
 - A Software Layer : The File Abstraction

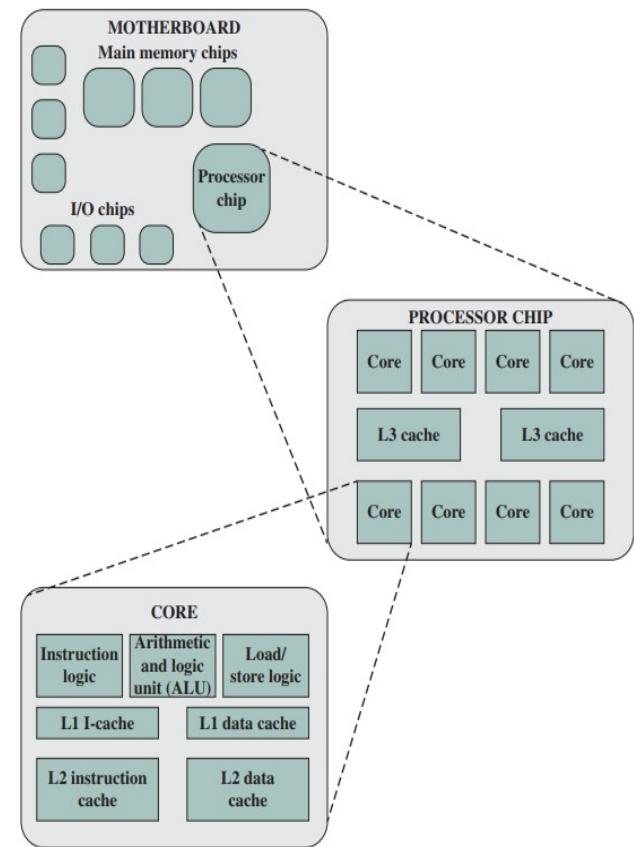
Computer: Top-level Structure

- Figure provides a **hierarchical** view of the internal structure of a traditional single-processor computer.
- The main structural components:
 - Central processing unit (CPU)**
 - Main memory**
 - System interconnection**



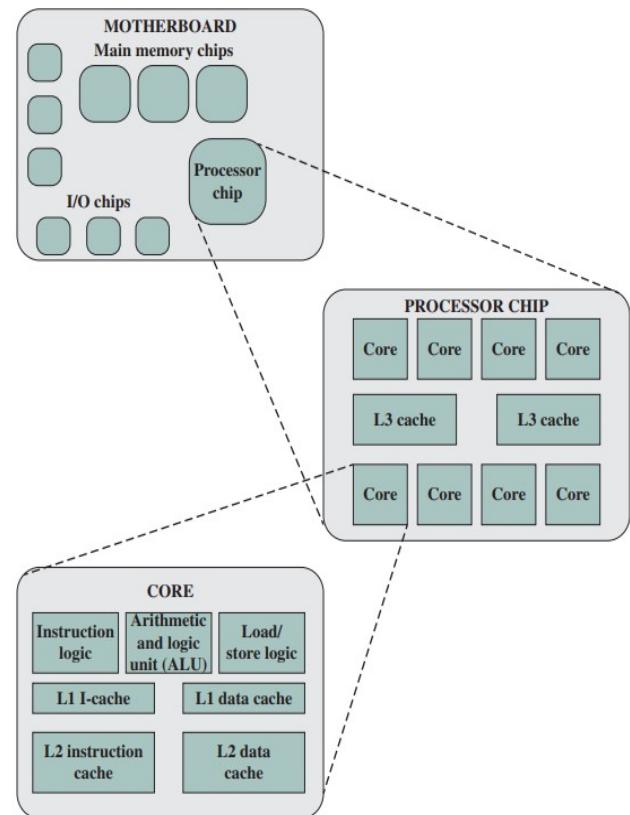
Simplified View of Major Elements of a Multicore Computer

- Figure is a simplified view of the principal components of a typical multicore computer.
- A printed circuit board (PCB) is a rigid, flat board that holds and interconnects chips and other electronic components.
- The board is made of layers, typically two to ten that interconnect components via copper pathways that are etched into the board.



Simplified View of Major Elements of a Multicore Computer

- The main printed circuit board in a computer is called a system board or motherboard, while smaller ones that plug into the slots in the main board are called expansion boards.
- The most prominent elements on the motherboard are the chips.

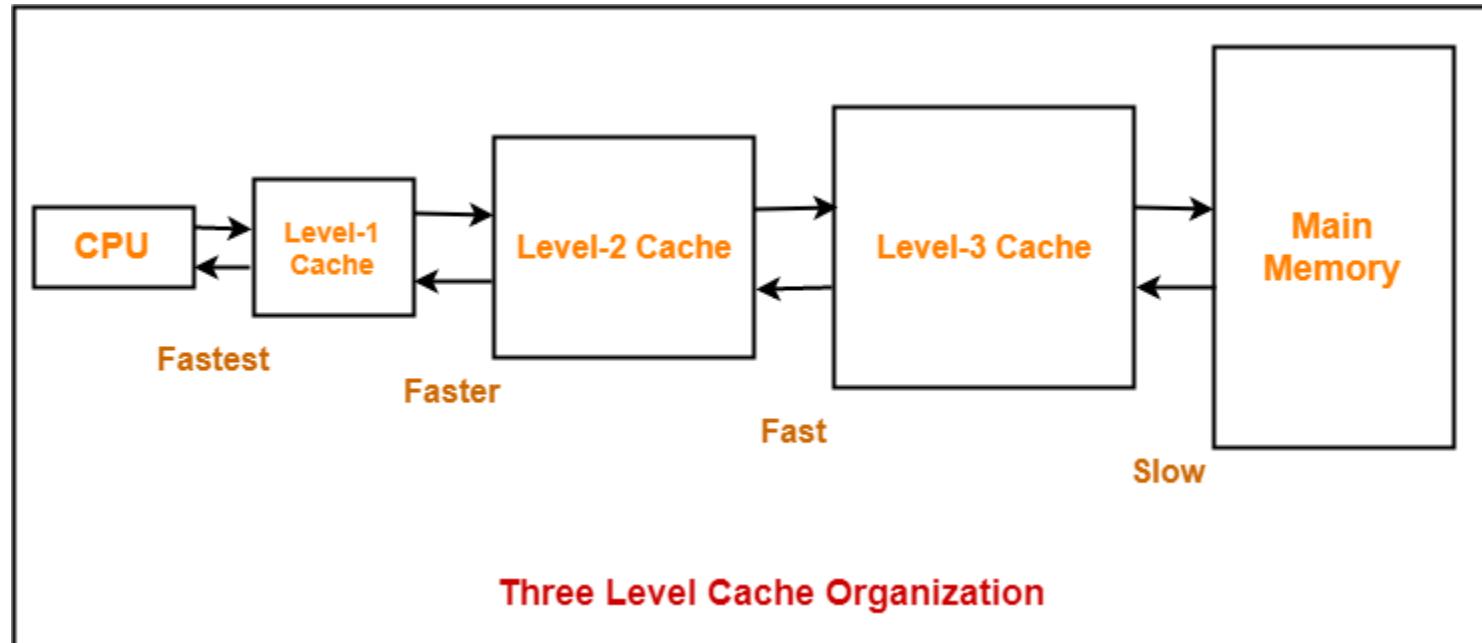


Simplified View of Major Elements of a Multicore Computer

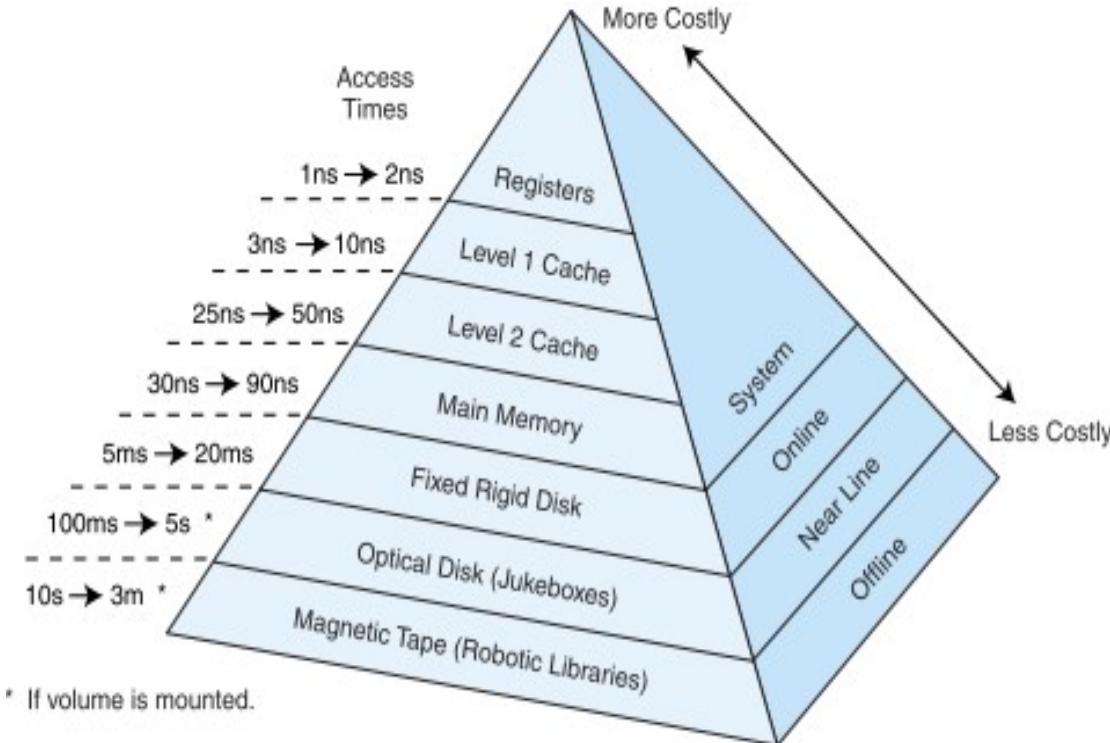
Lets look at

- Chip
- Mother board
- Memory, I/O controller, and other components
- Expansion slots
- Chips -> Transistors
- Multi level cache L1-L2-L3 Cache

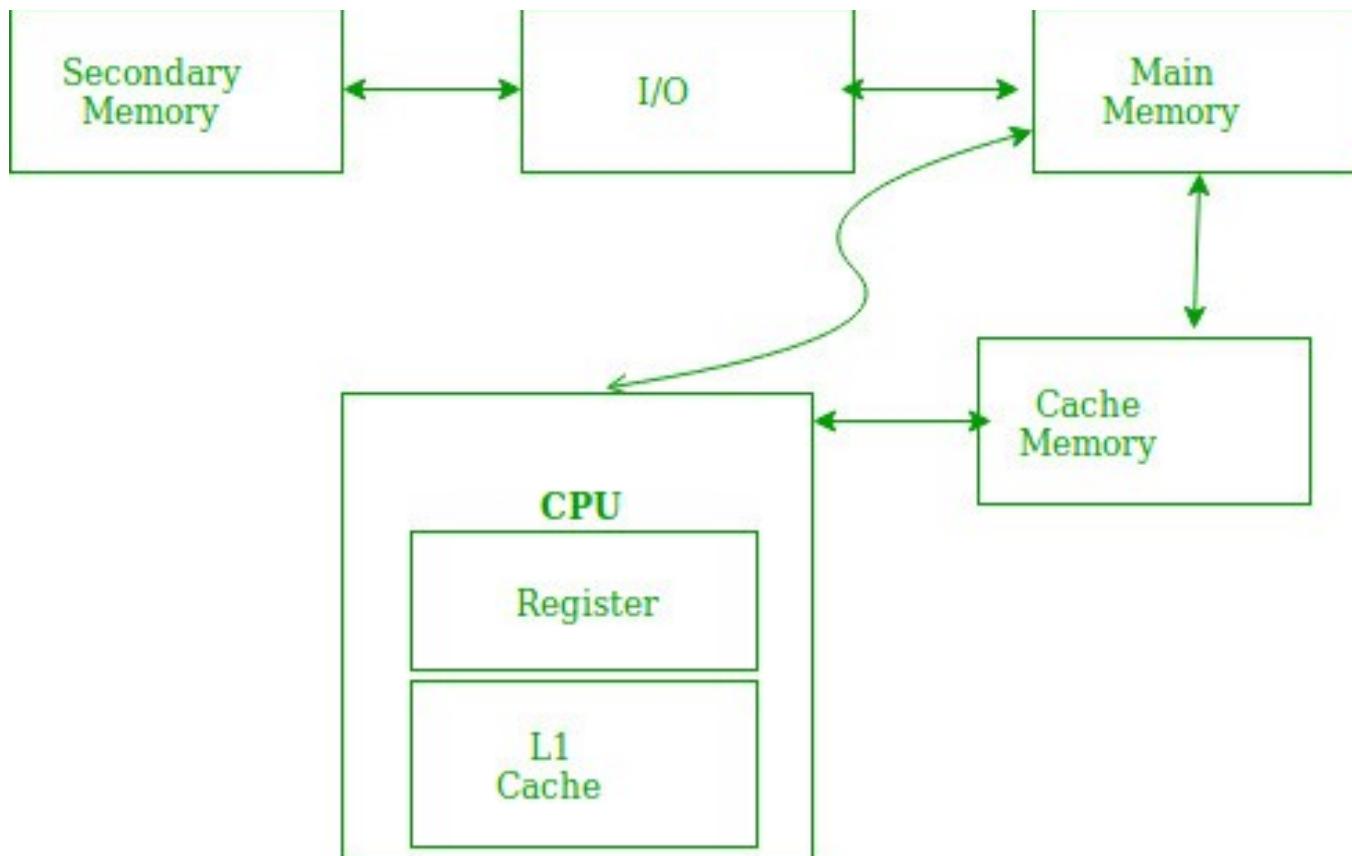
Simplified View of Major Elements of a Multicore Computer

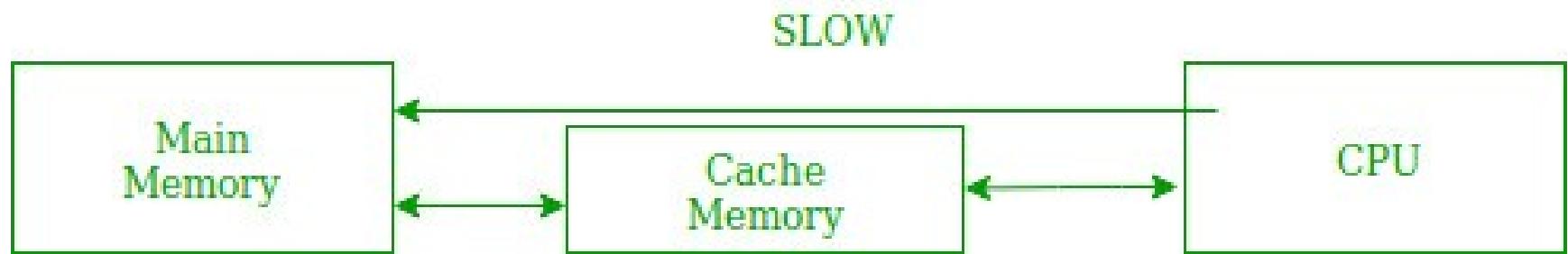


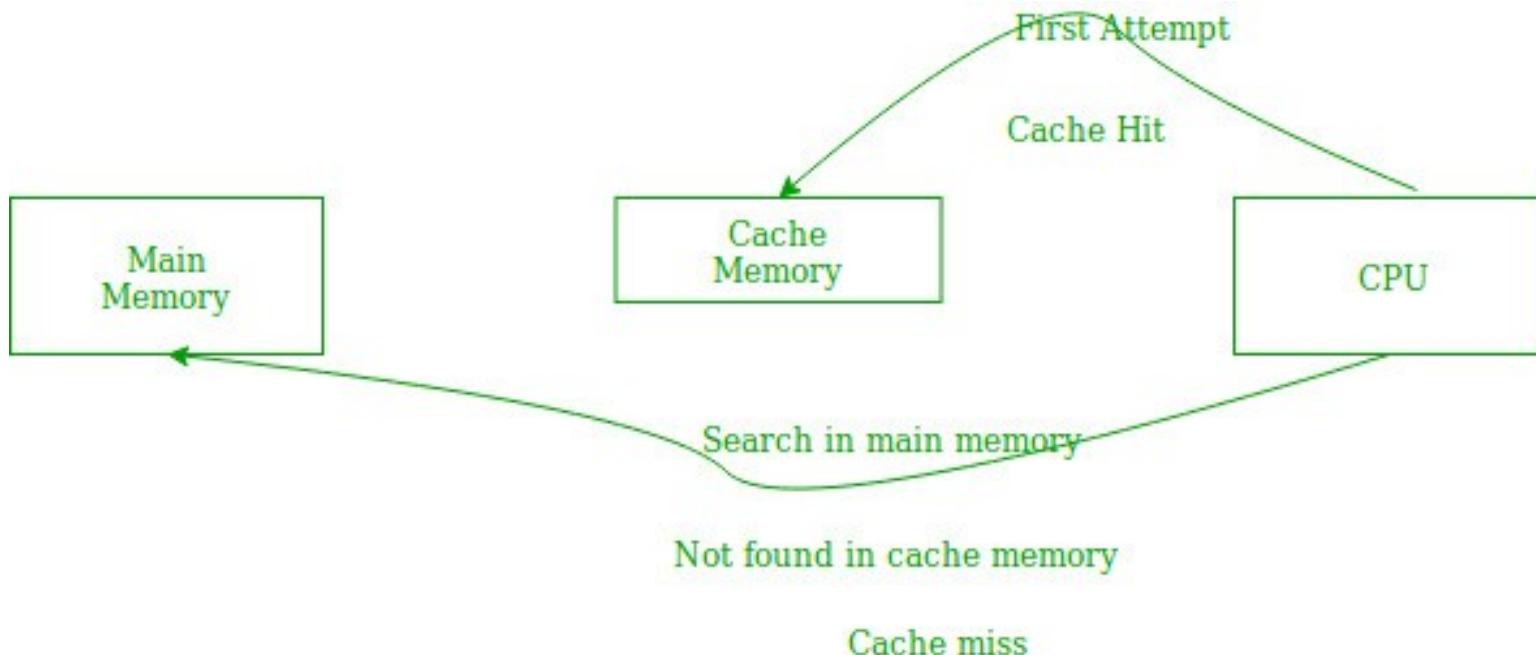
Computer Memory Hierarchy



Source: Null, Linda and Lobur, Julia (2003). *Computer Organization and Architecture* (p. 236). Sudbury, MA: Jones and Bartlett Publishers







Simplified View of Major Elements of a Multicore Computer

- L1 cache, split between an instruction cache (I-cache) that is used for the transfer of instructions to and from main memory, and an L1 data cache, for the transfer of operands and results.
- L2 cache as part of the core. In many cases, this cache is also split between instruction and data caches, although a combined, single L2 cache is also used.

Simplified View of Major Elements of a Multicore Computer

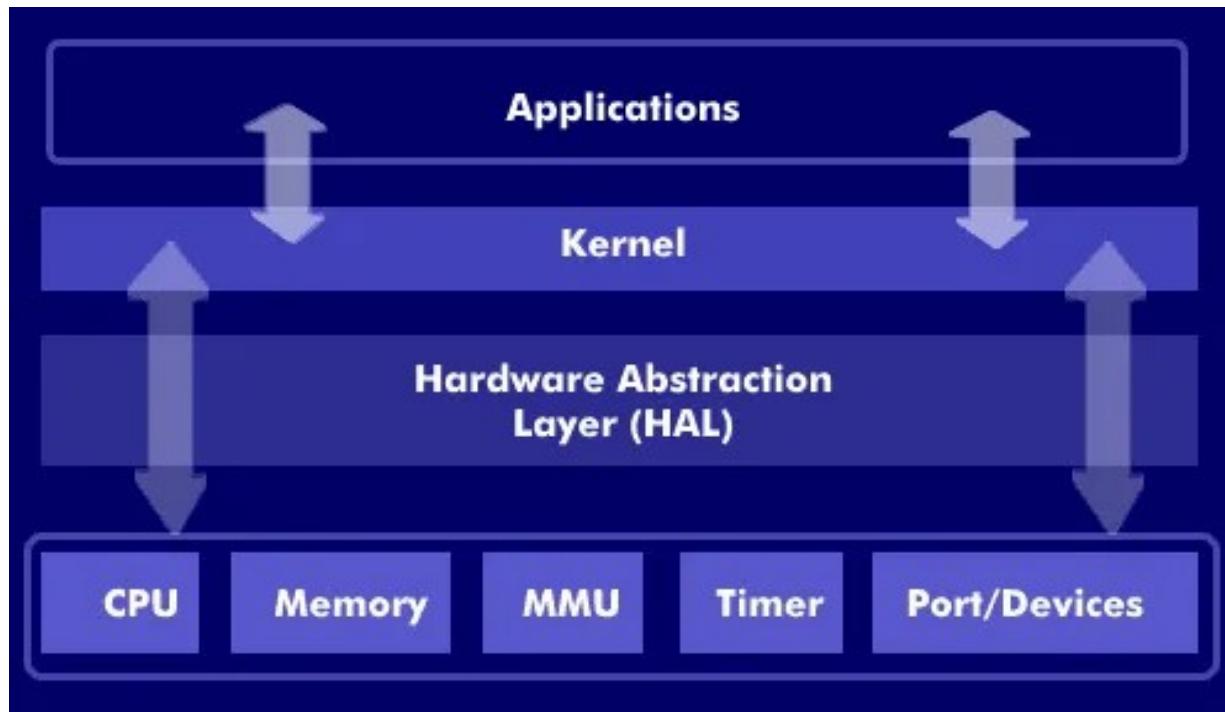
In general terms, the functional elements of a core are:

- Instruction logic: This includes the tasks involved in fetching instructions, and decoding each instruction to determine the instruction operation and the memory locations of any operands.
- Arithmetic and logic unit (ALU): Performs the operation specified by an instruction.
- Load/store logic: Manages the transfer of data to and from main memory via cache.

HARDWARE ABSTRACTION VS SOFTWARE ABSTRACTION

- **Hardware abstractions** are sets of routines in software that provide programs with access to hardware resources through programming interfaces.
- Hardware abstractions often allow programmers to write **device-independent, high performance applications** by providing standard operating system (OS) calls to hardware.
- **Hardware abstraction layer** is a thin layer of software at the base of the operating system that offers a uniform interface between the underlying hardware and the higher layers of the operating system, hiding hardware differences from those higher layers.

Hardware Abstraction Layer Example

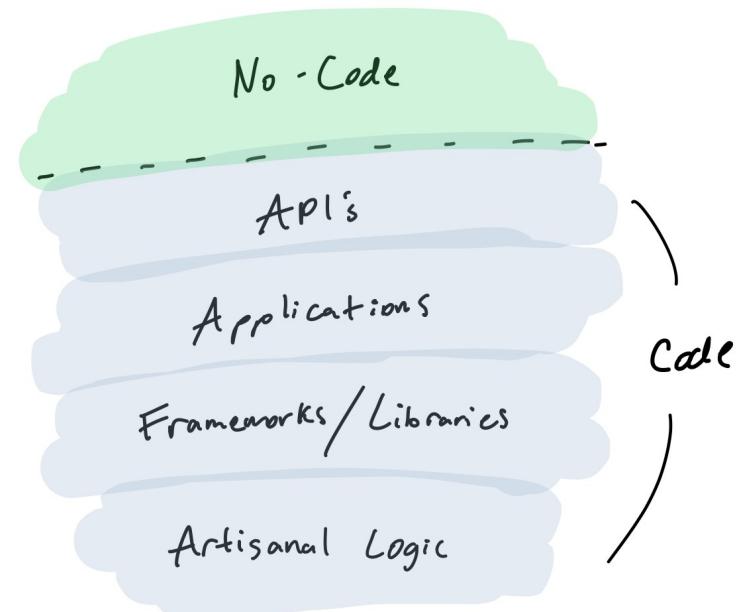


Software abstraction

The software abstraction is the creation of abstract concept-objects by mirroring common features or attributes of various non-abstract objects or systems of study.

Examples of software models that use layers of abstraction include the OSI model for network protocols, OpenGL and other graphics libraries.

Modern day example

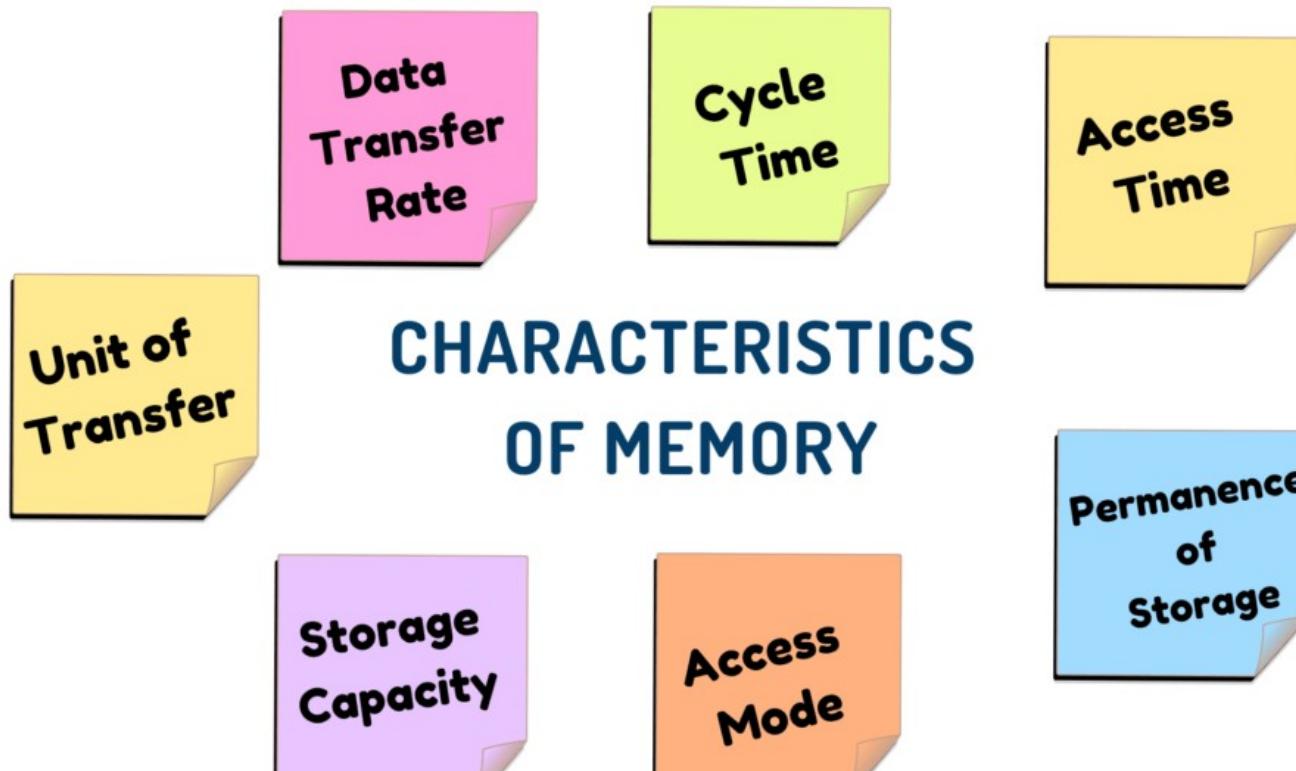


THE THREE FUNDAMENTAL ABSTRACTIONS

The three fundamental Abstractions

1. Memory
2. Interpreters
3. Communication links

THE THREE FUNDAMENTAL ABSTRACTIONS - Memory



THE THREE FUNDAMENTAL ABSTRACTIONS - Memory

- Memory, sometimes called storage, is the system component that remembers data values for use in computation.
- All memory devices fit a simple abstract model that has two operations, named write and read:
 - *WRITE (name, value)*
 - *Value \leftarrow READ (name)*

THE THREE FUNDAMENTAL ABSTRACTIONS - Memory

- Memories can be either volatile or non-volatile.
 - Volatile - Consumes energy to retain information
 - Non Volatile – retains its content, and when power is again available, READ operations return the same values as before.
 - By connecting a volatile memory to a battery or an uninterruptible power supply, it can be made durable, which means that it is designed to remember things for at least some specified period, known as its durability.
 - Even non-volatile memory devices are subject to eventual deterioration, known as decay, so they usually also have a specified durability, perhaps measured in years

Hardware memory devices:

RAM chip
Flash memory
Magnetic tape
Magnetic disk
CD-R and DVD-R

Higher level memory systems:

RAID
File system
Database management system



Read/Write Coherence & Atomicity

- Two useful properties for a memory are read/write coherence and before-or-after atomicity.
 - Result of the READ of a named cell is always the **same** as the most recent write to that cell.
- Before-or-after atomicity: **Irrespective** of before or after operations
- There are a surprising number of threats to read/write coherence and before-or-after atomicity:
 - **Concurrency**
 - **Remote storage**
 - **Performance enhancements**
 - **Cell size incommensurate with value size**
 - **Replicated storage**

Memory Latency

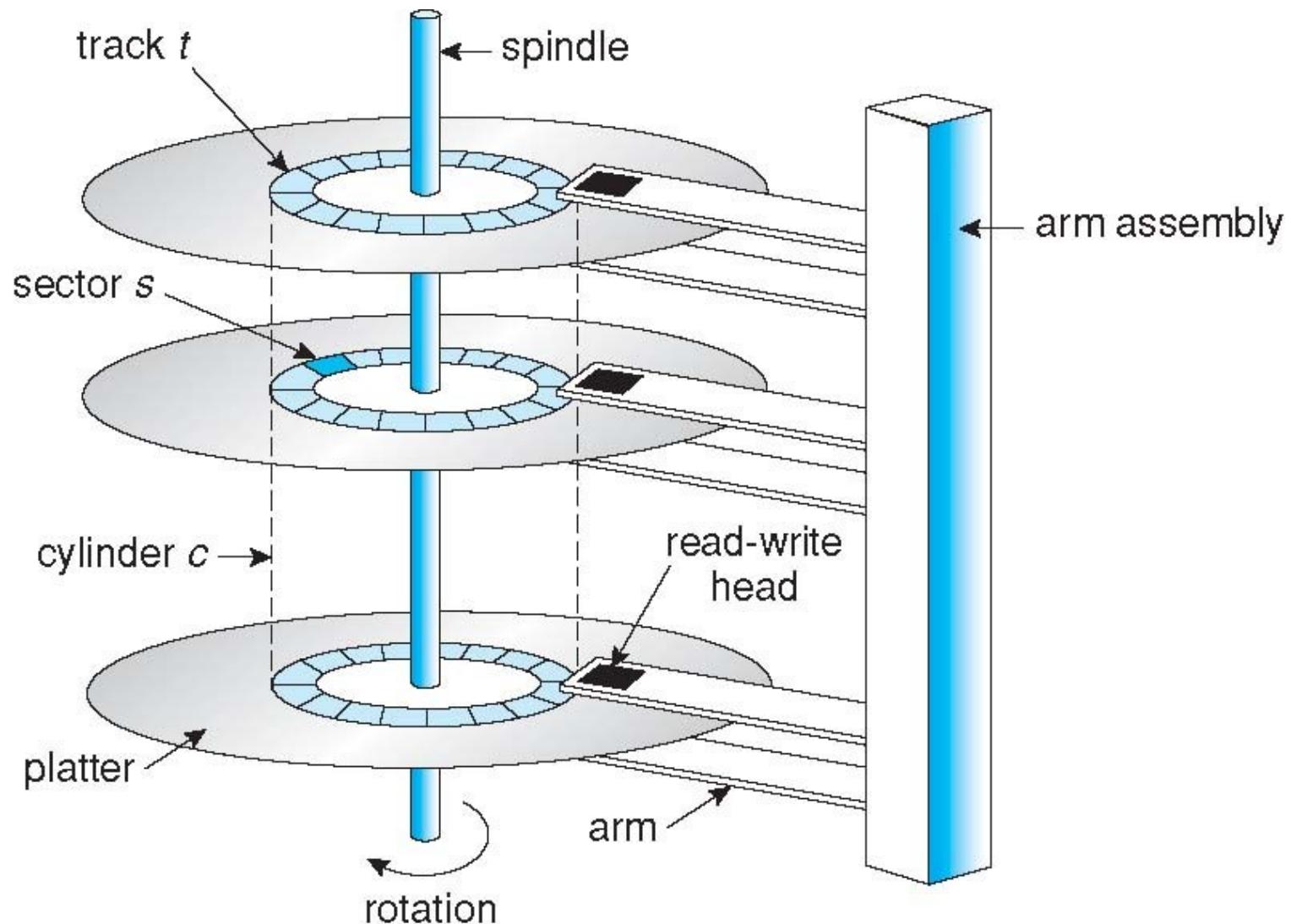
- An important property of a memory is the time it takes for a read or a write to complete, which is known as its latency (often called access time).
- In the magnetic disk memory the latency of a particular sector depends on the mechanical state of the device at the instant the user requests access.
- A random access memory (RAM) is one for which the latency for memory cells chosen at random is approximately the same as the latency for cells chosen in the pattern best suited for that memory device.
- An electronic memory chip is usually configured for random access.

Overview of Mass Storage Structure



- **Magnetic disks** provide bulk of secondary storage of modern computers
 - Drives rotate at 60 to 200 times per second
 - **Transfer rate** is rate at which data flow between drive and computer
 - **Positioning time (random-access time)** is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
 - **Head crash** results from disk head making contact with the disk surface
 - That's bad
- Disks can be removable
- Drive attached to computer via **I/O bus**
 - Busses vary, including **EIDE, ATA, SATA, USB, Fibre Channel, SCSI**
 - **Host controller** in computer uses bus to talk to **disk**

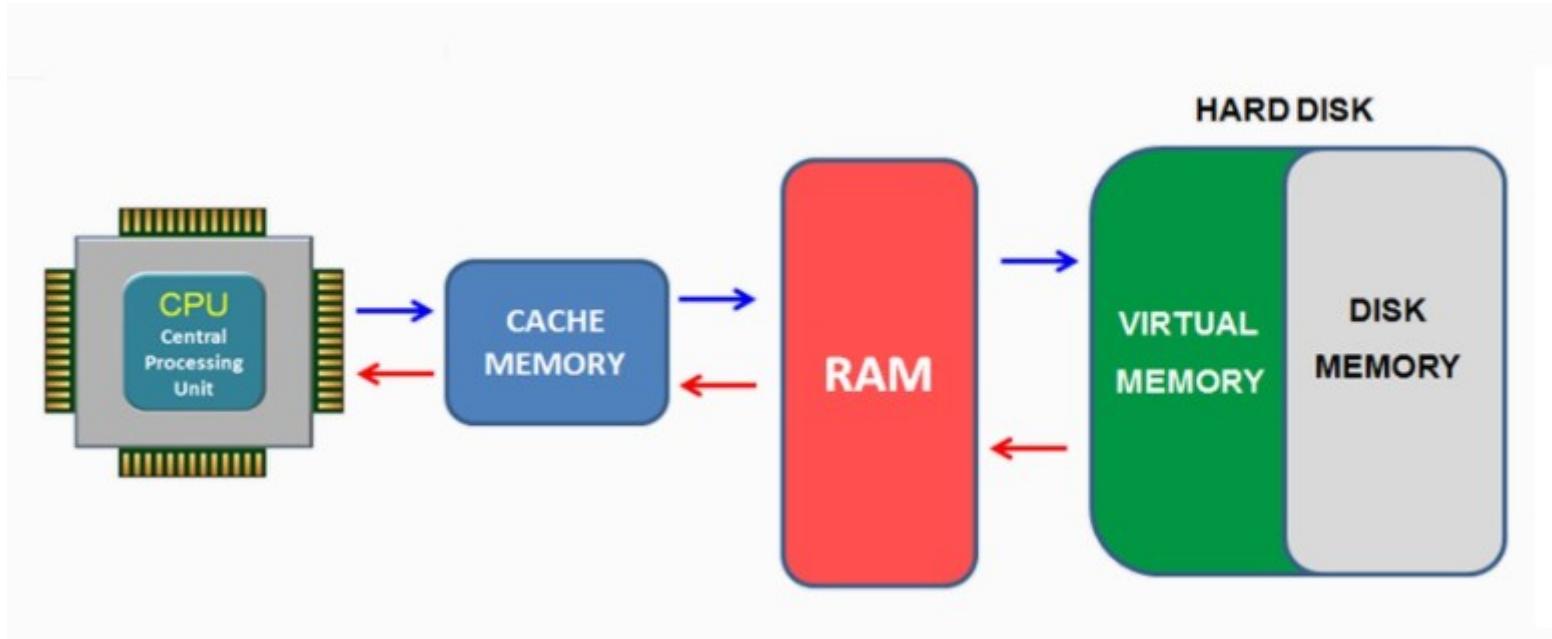
Moving-head Disk Mechanism



Disk Scheduling

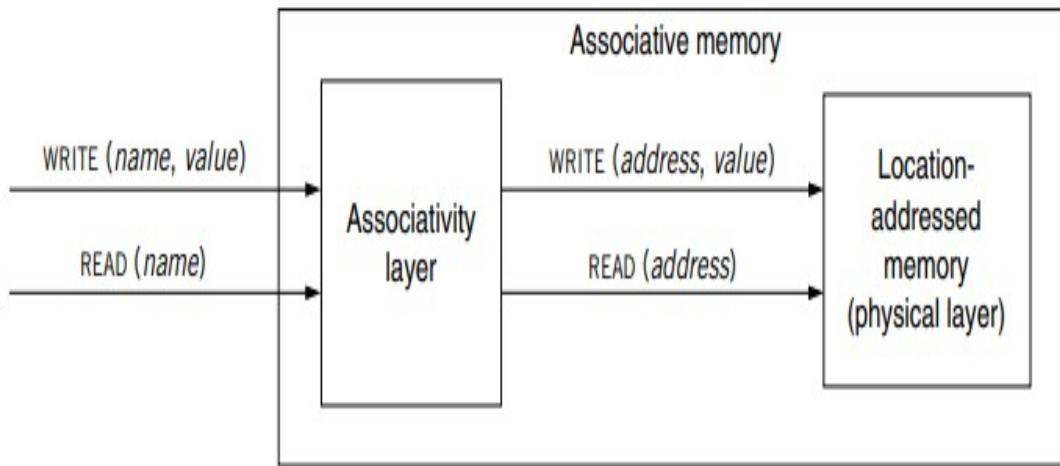
- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Access time has two major components
 - **Seek time** is the time for the disk are to move the heads to the cylinder containing the desired sector
 - **Rotational latency** is the additional time waiting for the disk to rotate the desired sector to the disk head
- Minimize seek time
- Seek time \approx seek distance
- Disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first byte transferred and the last byte transferred

Memory names and addresses



Memory Names and Addresses

- Physical implementations of memory devices nearly always name a memory cell by the geometric coordinates of its **physical storage location**.
- Thus, for example, an electronic memory chip is organized as a two-dimensional array of flip-flops, each holding one named bit.
- A memory system that accepts unconstrained names is called an associative memory.
- Since **physical memories are generally location-addressed**, a designer creates an associative memory by interposing an **associativity layer**, which may be implemented either with hardware or software that maps **unconstrained higher-level names** to the constrained integer names of an underlying location-addressed memory.

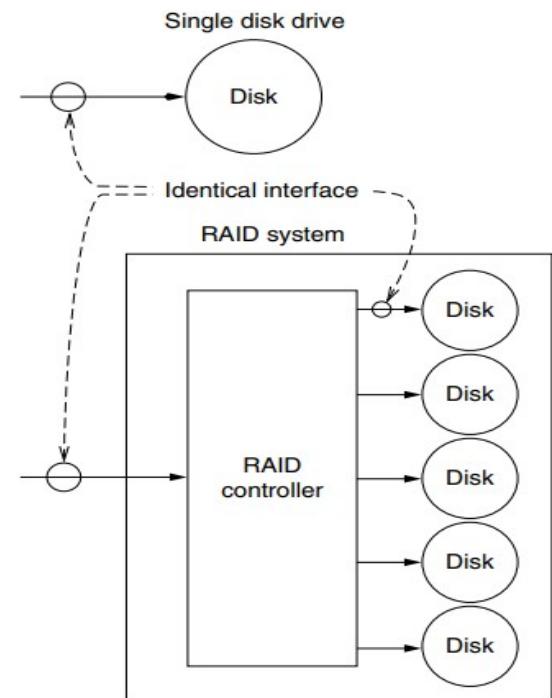


Exploiting the Memory Abstraction: RAID



<https://www.youtube.com/watch?v=U-OCdTeZLac>

- RAID is an acronym for Redundant Array of Independent (or Inexpensive) Disks.
- A RAID system consists of a set of disk drives and a controller configured with an electrical and programming interface that is identical to the interface of a single disk drive, as shown in Figure.
- The RAID controller intercepts read and write requests coming across its interface, and it directs them to one or more of the disks. RAID has two distinct goals:
 - *Improved performance, by reading or writing disks concurrently*
 - *Improved durability, by writing information on more than one disk*



Abstraction II - Interpreters

- Interpreters are the active elements of a computer system; they perform the actions that constitute computations.
- Figure lists some examples of interpreters that may be familiar. As with memory, interpreters also come in a wide range of physical manifestations.

Abstraction II - Interpreters

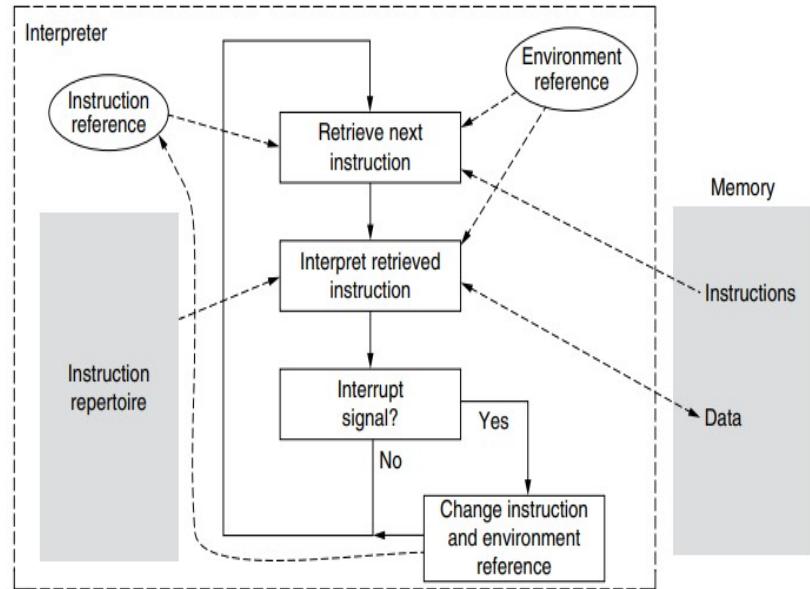
- However, they too can be described with a simple abstraction, consisting of just three components:
 1. *An instruction reference, which tells the interpreter where to find its next instruction*
 2. *A repertoire, which defines the set of actions the interpreter is prepared to perform when it retrieves an instruction from the location named by the instruction reference*
 3. *An environment reference, which tells the interpreter where to find its environment, the current state on which the interpreter should perform the action of the current instruction.*

Hardware:

Pentium 4, PowerPC 970, UltraSPARC T1
 disk controller
 display controller

Software:

Alice, AppleScript, Perl, Tcl, Scheme
 LISP, Python, Forth, Java bytecode
 JavaScript, Smalltalk
 TeX, LaTeX
 Safari, Internet Explorer, Firefox



```

1  procedure INTERPRET()
2    do forever
3      instruction ← READ (instruction_reference)
4      perform instruction in the context of environment_reference
5      if interrupt_signal = TRUE then
6          instruction_reference ← entry point of INTERRUPT_HANDLER
7          environment_reference ← environment ref of INTERRUPT_HANDLER
  
```

- 1) Where is the instruction – Sequential or non-sequential
- 2) Where are the operands
- 3) Interrupts

Processors

- The processor's instruction reference is a **program counter**, stored in a fast memory register inside the processor.
- The program counter contains the address of the memory location that stores the **next instruction** of the current program.
- The environment reference of the processor consists in part of a small amount of built-in location-addressed memory in the form of named (by number) **registers** for fast access to temporary results of computations.
- The processor also **implements interrupts**. An interrupt can occur because the processor has detected **some problem** with the running program.
- An interrupt can also occur because a signal **arrives from outside the processor**, indicating that some external device needs attention.

Communication Links

- A communication link provides a way for information to move between physically separated components.
- Programs that invoke send and receive must take these different semantics explicitly into account.
- On the other hand, some communication link implementations do provide a layer that does its best to hide a send/receive interface behind a read/write interface.
- Just as with memory and interpreters, designers organize and implement communication links in layers.

ORGANIZING COMPUTER SYSTEMS WITH NAMES AND LAYERS

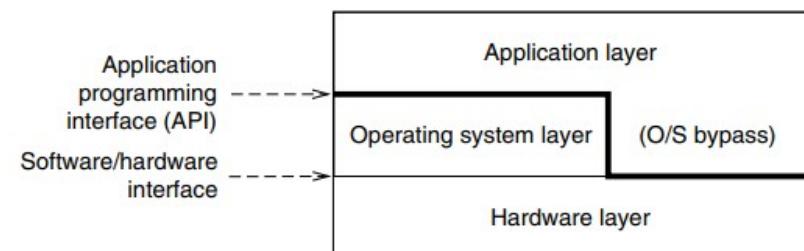
The bottom layer consists of hardware components, such as processors, memories, and communication links.

The middle layer consists of a collection of software modules, called the operating system,

The top layer consists of software that implements application specific functions,

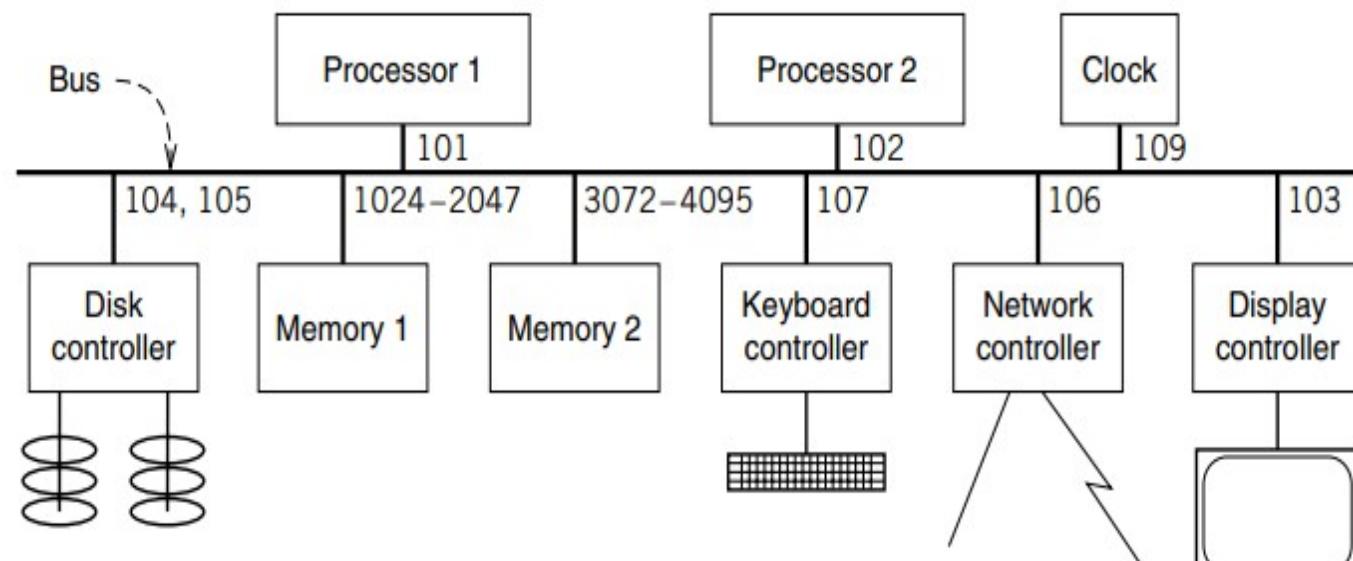
In principle, every software module can be implemented in hardware.

Similarly, most hardware modules can also be implemented in software, except for a few foundational components such as transistors and wires.



A Hardware Layer: The Bus

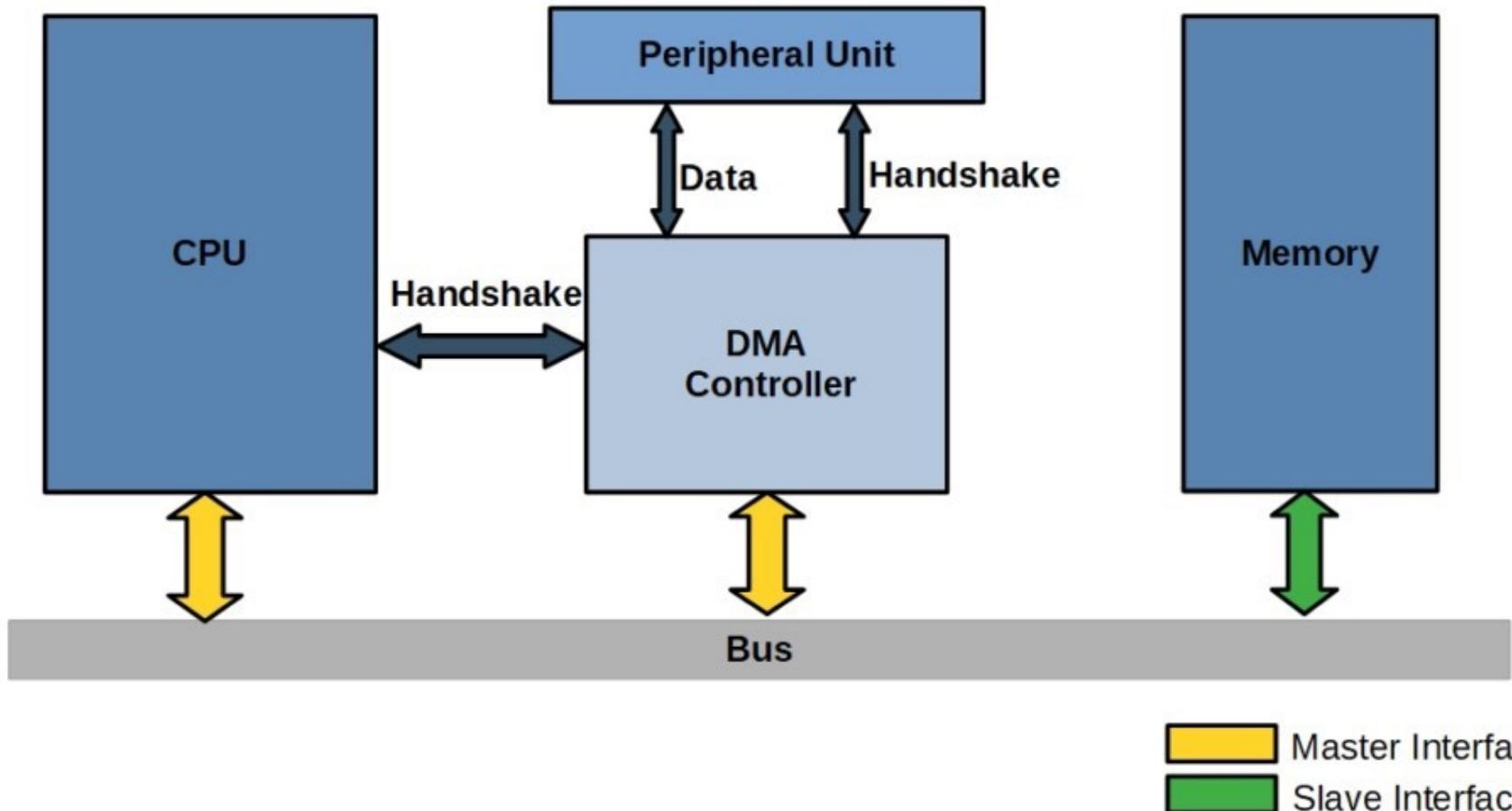
- The processor modules interpret programs,
 - the random access memory modules store both programs and data,
 - and the input/output (I/O) modules implement communication links to the world outside the computer.
- The I/O modules turn out to be specialized interpreters that implement I/O programs.



A Hardware Layer: The Bus

- Any bus module that wishes to send a message over the bus must know a **bus address** that the intended recipient is configured to accept.
-
- For example, suppose that processor #2, while interpreting a running application program, encounters the instruction
LOAD 1742, R1
- which means “**load the contents of memory address 1742 into processor register R1**”.
- In the simplest scheme, the **processor just translates addresses** it finds in instructions directly to bus addresses without change.

A Hardware Layer: The Bus



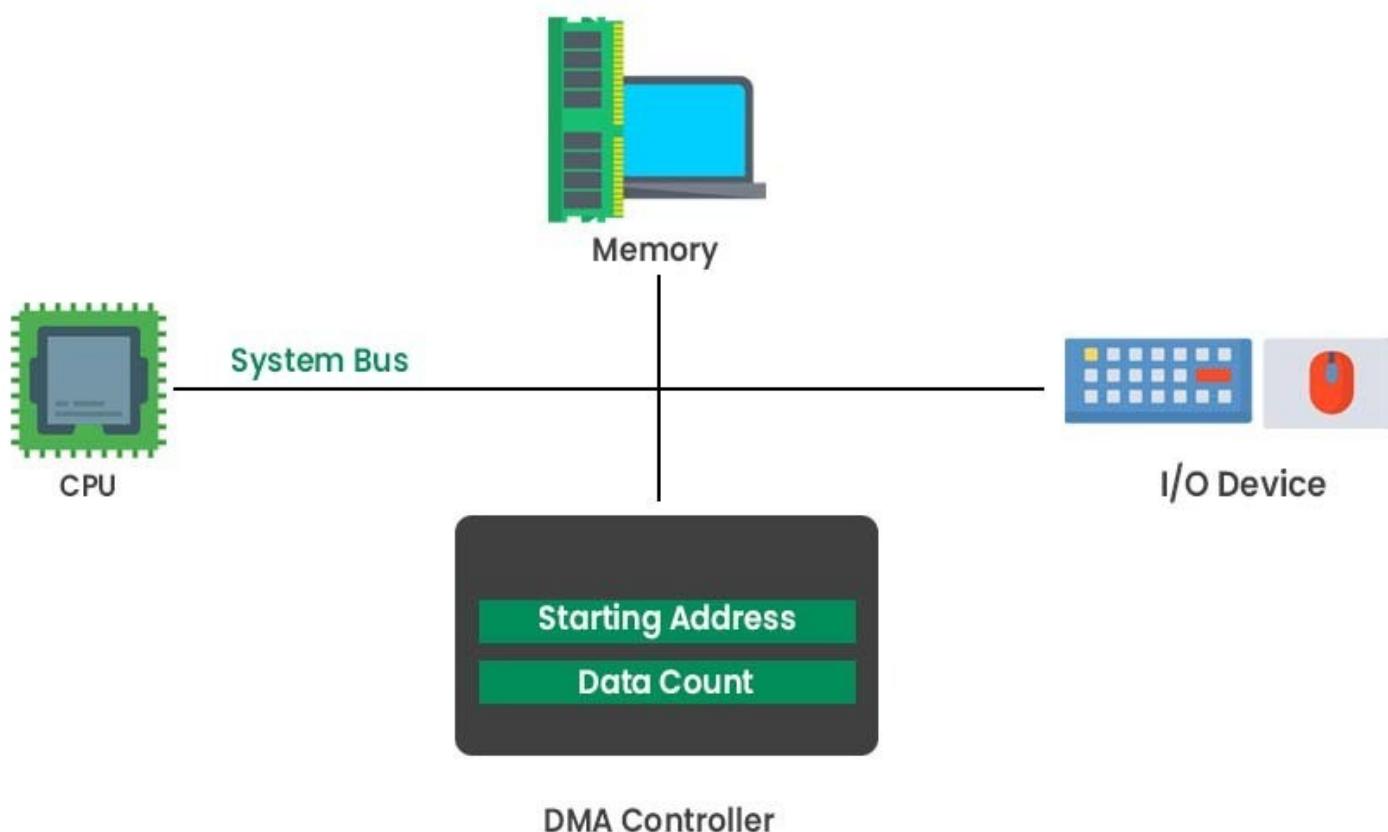
A Hardware Layer: The Bus

- DMA- Upgrading the disk controller to use a technique called direct memory access, or DMA.
 - With this technique, when a processor sends a request to a disk controller to read a block of data from the disk, it includes the address of a buffer in memory as a field of the request message.
 - Then, as data streams in from the disk, the disk controller sends it directly to the memory module, incrementing the memory address appropriately between sends.
 - In addition to relieving the load on the processor, DMA also reduces the load on the shared bus because it transfers each piece of data across the bus just once (from the disk controller to the memory) rather than twice (first from the disk controller to the processor and then from the processor to the memory).

- DMA basically stands for Direct Memory Access.
- It is a process which enables data transfer between the Memory and the IO (Input/ Output) device without the need of or you can say without the involvement of CPU during data transfer.

Working of DMA

- For DMA, you basically need a hardware called DMAC (Direct Memory Access Controller) which will help in the throughout process of data transfer between the Memory and IO device directly.
- First what happens is IO device sends the DMA request to DMA Controller, then further DMAC device sends HOLD signal to CPU by which it asks CPU for several information which are needed while transferring data.
- CPU then shares two basic information with DMAC before the Data transfer which are: Starting address (memory address starting from where data transfer should be performed) and Data Count (no of bytes or words to be transferred).
- CPU then sends HLDACK (Hold Acknowledgement) back to DMAC illustrating that now DMAC can successfully pass on the information.
- Then further DMAC shares the DMA ACK (DMA Acknowledgement) to the IO device which would eventually let IO device to access or transfer the data from memory in a direct and efficient manner.



A Software Layer

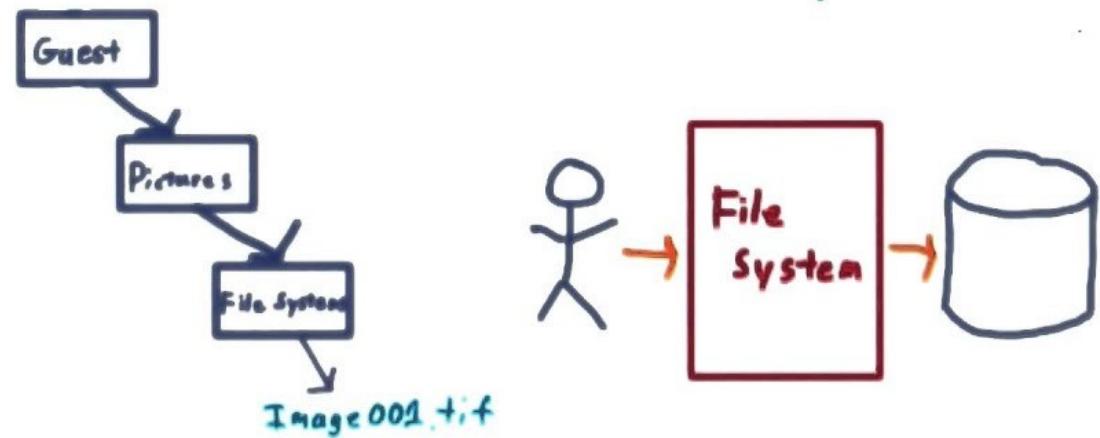
- The File Abstraction
 - The middle and higher layers of a computer system are usually implemented as software modules.
 - To make this layered organization concrete, consider the file, a high-level version of the memory abstraction.
- A file holds an array of bits or bytes, the number of which the application chooses. A file has two key properties:
 - It is durable
 - It has a name

A Software Layer

File System Concept

Key Abstractions

1. File
2. Filename
3. Directory Tree



A Software Layer

- Taken together, these two features mean that if, for example, Alice creates a new file named “strategic plan”, writes some information in it, shuts down the computer, and the next day turns it on again, she will then be able to read the file named “strategic plan” and get back its content.
- Furthermore, she can tell Bob to look at the file named “strategic plan”.
- When Bob asks the system to read a file with that name, he will read the file that she created.
- Most file systems also provide other additional properties for files, such as timestamps to determine when they were created, last modified, or last used, assurances about their durability, and the ability to control who may share them.

Software layer - Files

```
character buf
file ← OPEN ("strategic plan", READWRITE)
input ← OPEN ("keyboard", READONLY)
display ← OPEN ("display", WRITEONLY)
// buffer for input character
// open file for reading and writing
// open keyboard device for reading
// open display device for writing

while not END_OF_FILE (input) do
    READ (input, buf, 1)
    WRITE (file, buf, 1)
    WRITE (display, buf, 1)
// read 1 character from keyboard
// store input into file
// display input

CLOSE (file)
CLOSE (input)
CLOSE (display)
```

Software layer - Files

- The system layer implements files using modules from the hardware layer.
 - A typical API for the file abstraction contains calls to open a file, to read and write parts of the file, and to close the file.
 - The open call translates the file name into a temporary name in a local name space to be used by the read and write operations.
 - Also, open usually checks whether this user is permitted access to the file. As its last step, open sets a cursor, sometimes called a file pointer, to zero.

Software layer - files

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num;
    FILE *fptr;
    fptr = fopen("C:\\program.txt","w");

    if(fptr == NULL)
    {
        printf("Error!");
        exit(1);
    }

    printf("Enter num: ");
    scanf("%d",&num);

    fprintf(fptr,"%d",num);
    fclose(fptr);

    return 0;
}
```

Software layer - files

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num;
    FILE *fptr;

    if ((fptr = fopen("C:\\program.txt","r")) == NULL){
        printf("Error! opening file");

        // Program exits if the file pointer returns NULL.
        exit(1);
    }

    fscanf(fptr,"%d", &num);

    printf("Value of n=%d", num);
    fclose(fptr);

    return 0;
}
```

Software layer - files

- The cursor records an offset from the beginning of the file to be used as the starting point for reads and writes.
- A call to read delivers to the caller a specified number of bytes from the file, starting from the read cursor.
- It also adds to the read cursor the number of bytes read so that the next read proceeds where the previous read left off.
- If the program asks to read bytes beyond the end of the file, read returns some kind of end-of-file status indicator.

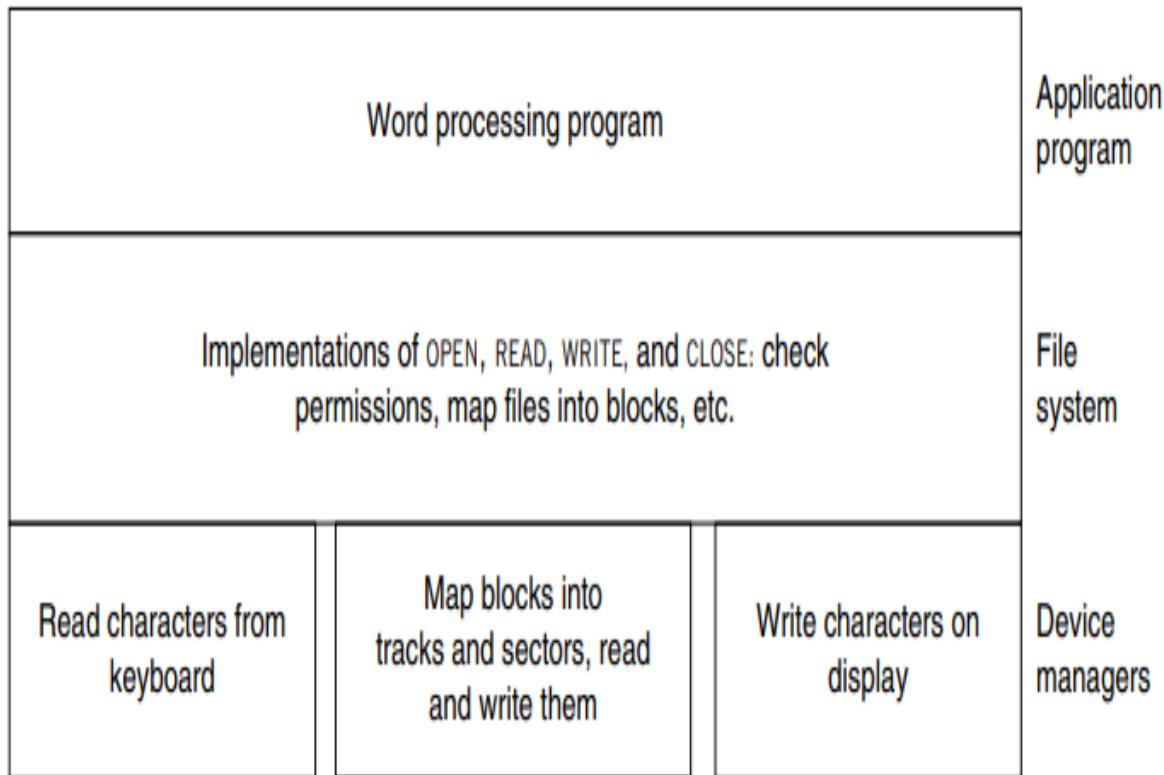
Software layer - files

- If there is not enough space on the device to write that many bytes, the write procedure fails by returning some kind of device-full error status or exception.
- Finally, when the program is finished reading and writing, it calls the close procedure. Close frees up any internal state that the file system maintains for the file.
- The file system module implements the file API by mapping bytes of the file to disk sectors.
- CLOSE was introduced to simplify resource management: when an application invokes close, the file system knows that the application doesn't need the resources (e.g., the cursor) that the file system maintains internally.

Software layer - files

- The file is such a convenient memory abstraction that in some systems every input/output device in a computer system provides a file interface
- In such systems, files not only are an abstraction for non-volatile memories (e.g., magnetic disks), but they are also a convenient interface to the keyboard device, the display, communication links, and so on.
- In such systems, each I/O device has a name in the file naming scheme.
- A program opens the keyboard device, reads bytes from the keyboard device, and then closes the keyboard device, without having to know any details about the keyboard management procedure, what type of keyboard it is, and the like.
- Similarly, to interact with the display, a program can open the display device, write to it, and close it. The program need not know any details about the display.

Software layer - files





BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 3

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Thangakumar J
WILP & Department of CS & IS



Last Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Major Modules of a Typical Computer System: Processor(s), Memory, I/O and Storage Devices• Hardware abstraction vs. Software• The three fundamental Abstractions• Organizing Computer Systems	<p>R6 (1.2)</p> <p>T2 (2.1 and 2.3)</p>

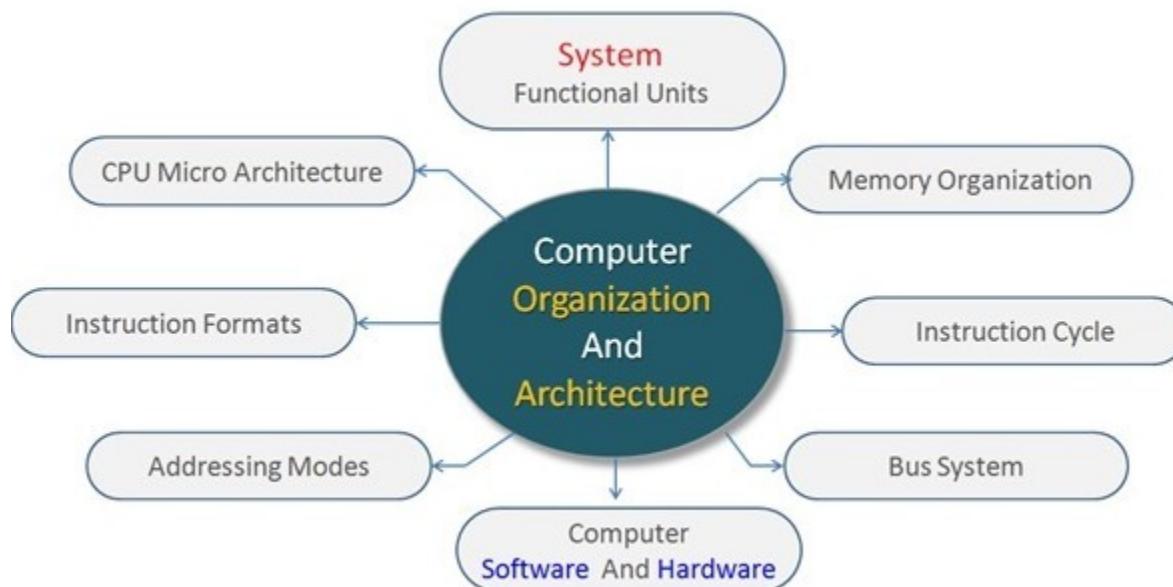
ISA as the Interface

Today's topic

- Computer Architecture Vs. Organization (Abstraction vs. Realization)
- Von-Neumann Architecture vs Harvard Architecture
- Features and Components of an ISA.
 - CISC Vs. RISC
- Design Goals

Computer Architecture VS Organization (Abstraction vs Realization)

- Computer architecture refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program.



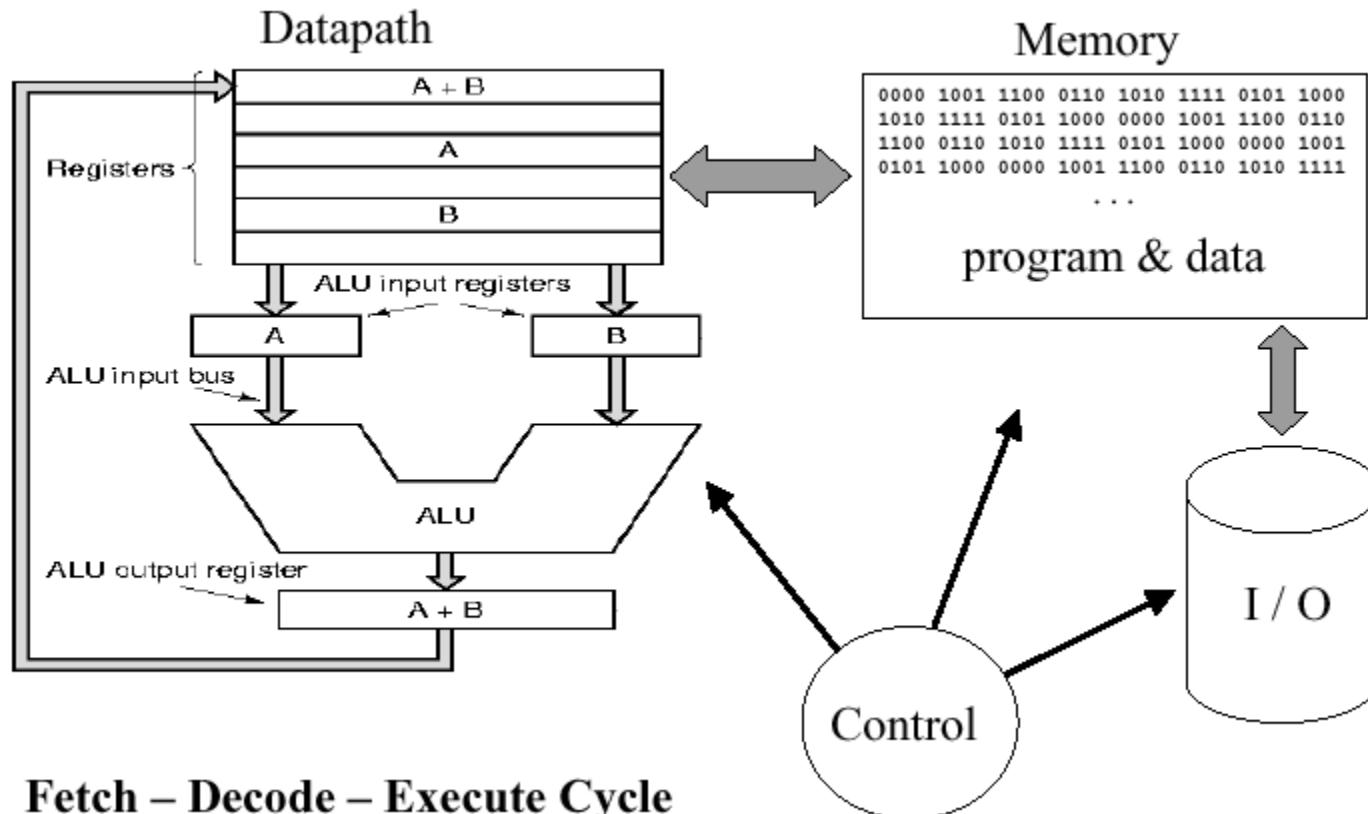
• A term that is often used interchangeably with computer architecture is Instruction Set Architecture (ISA).

- The ISA defines instruction formats, instruction opcodes, registers, instruction and data memory; the effect of executed instructions on the registers and memory; and an algorithm for controlling instruction execution.

- Computer organization refers to the operational units and their interconnections that realize the architectural specifications.

- Examples of architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and techniques for addressing memory.

Basic ISA cycle



- Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer and peripherals; and the memory technology used.
- For example,
 - it is an architectural design issue whether a computer will have a multiply instruction.
 - It is an organizational issue whether that instruction will be implemented by a special multiply unit or by a mechanism that makes repeated use of the add unit of the system.

- The organizational decision may be based on the anticipated frequency of use of the multiply instruction, the relative speed of the two approaches, and the cost and physical size of a special multiply unit.
- Historically, and still today, the distinction between architecture and organization has been an important one.
-

- Many computer manufacturers offer a family of computer models, all with the same architecture but with differences in organization.

- Consequently, the different models in the family have different price and performance characteristics.

- Furthermore, a particular architecture may span many years and encompass a number of different computer models, its organization changing with changing technology.

•A prominent example of both these phenomena is the IBM System/370 architecture.

- This architecture was first introduced in 1970 and included a number of models.
- The customer with modest requirements could buy a cheaper, slower model and, if demand increased, later upgrade to a more expensive, faster model without having to abandon software that had already been developed.

- Over the years, IBM has introduced many new models with improved technology to replace older models, offering the customer greater speed, lower cost, or both.
- These newer models retained the same architecture so that the customer's software investment was protected.

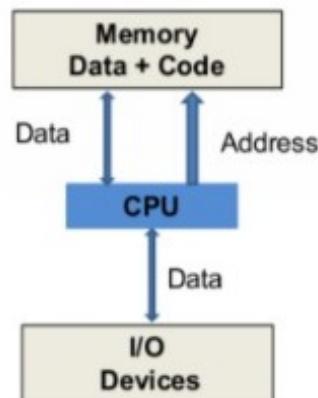
- Remarkably, the System/370 architecture, with a few enhancements, has survived to this day as the architecture of IBM's mainframe product line.
- In a class of computers called microcomputers, the relationship between architecture and organization is very close.
- Changes in technology not only influence organization but also result in the introduction of more powerful and more complex architectures.
- Generally, there is less of a requirement for generation-to-generation compatibility for these smaller machines.
- Thus, there is more interplay between organizational and architectural design decisions.

Summary – Computer Organization vs Computer Architecture

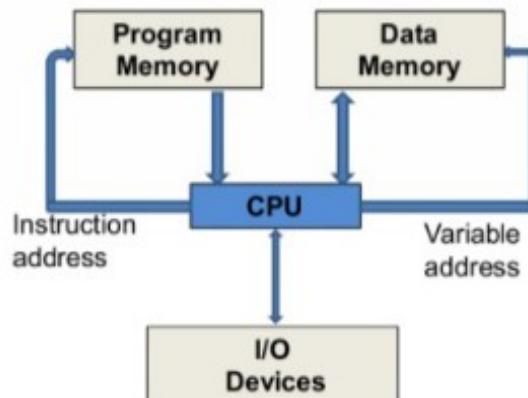


Computer Organization	Computer Architecture
Often called microarchitecture (low level)	Computer architecture (a bit higher level)
Transparent from programmer (ex. A programmer does not worry much how addition is implemented in hardware)	Programmer view (i.e. Programmer has to be aware of which instruction set used)
Physical components (Circuit design, Adders, Signals, Peripherals)	Logic (Instruction set, Addressing modes, Data types, Cache optimization)
How to do ? (implementation of the architecture)	What to do ? (Instruction set)

Von-Neumann Architecture vs Harvard Architecture

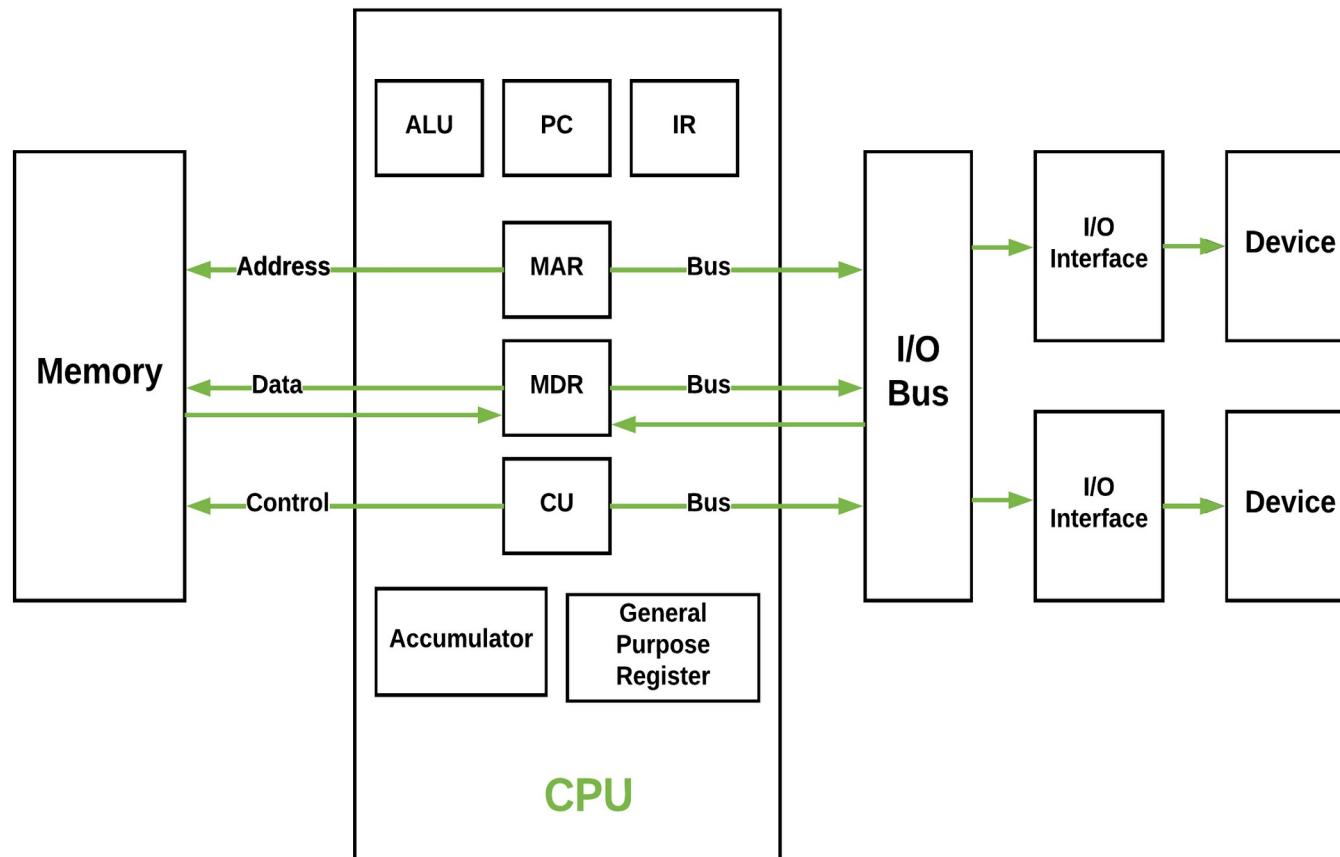


Von Neumann Machine



Harvard Machine

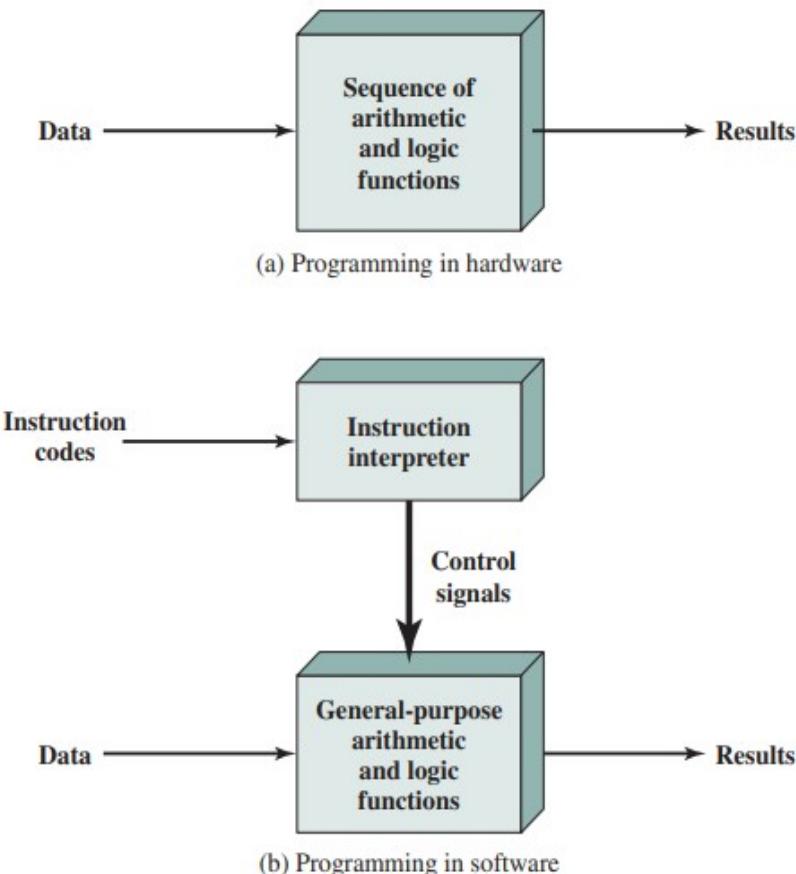
Von-Neumann Architecture vs Harvard Architecture



- Virtually all contemporary computer designs are based on concepts developed by John von Neumann at the Institute for Advanced Studies, Princeton. Such a design is referred to as the von Neumann architecture and is based on three key concepts:
 1. Data and instructions are stored in a single read–write memory.
 2. The contents of this memory are addressable by location, without regard to the type of data contained there.
 3. Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next.
- There is a small set of basic logic components that can be combined in various ways to **store binary data and perform arithmetic and logical operations on that data.**

- If there is a particular computation to be performed, a configuration of logic components designed specifically for that computation could be constructed.
- We can think of the process of connecting the various components in the desired configuration as a form of programming.
- The resulting “program” is in the form of hardware and is termed a hardwired program.
- Now consider this alternative. Suppose we construct a general-purpose configuration of arithmetic and logic functions.
 - This set of hardware will perform various functions on data depending on control signals applied to the hardware.
 - In the original case of customized hardware, the system accepts data and produces results (Figure. a).

- With general-purpose hardware, the system accepts data and control signals and produces results.
- Thus, instead of rewiring the hardware for each new program, the programmer merely needs to supply a new set of control signals, and let us add to the general-purpose hardware a segment that can accept a code and generate control signals (Figure. (b)).

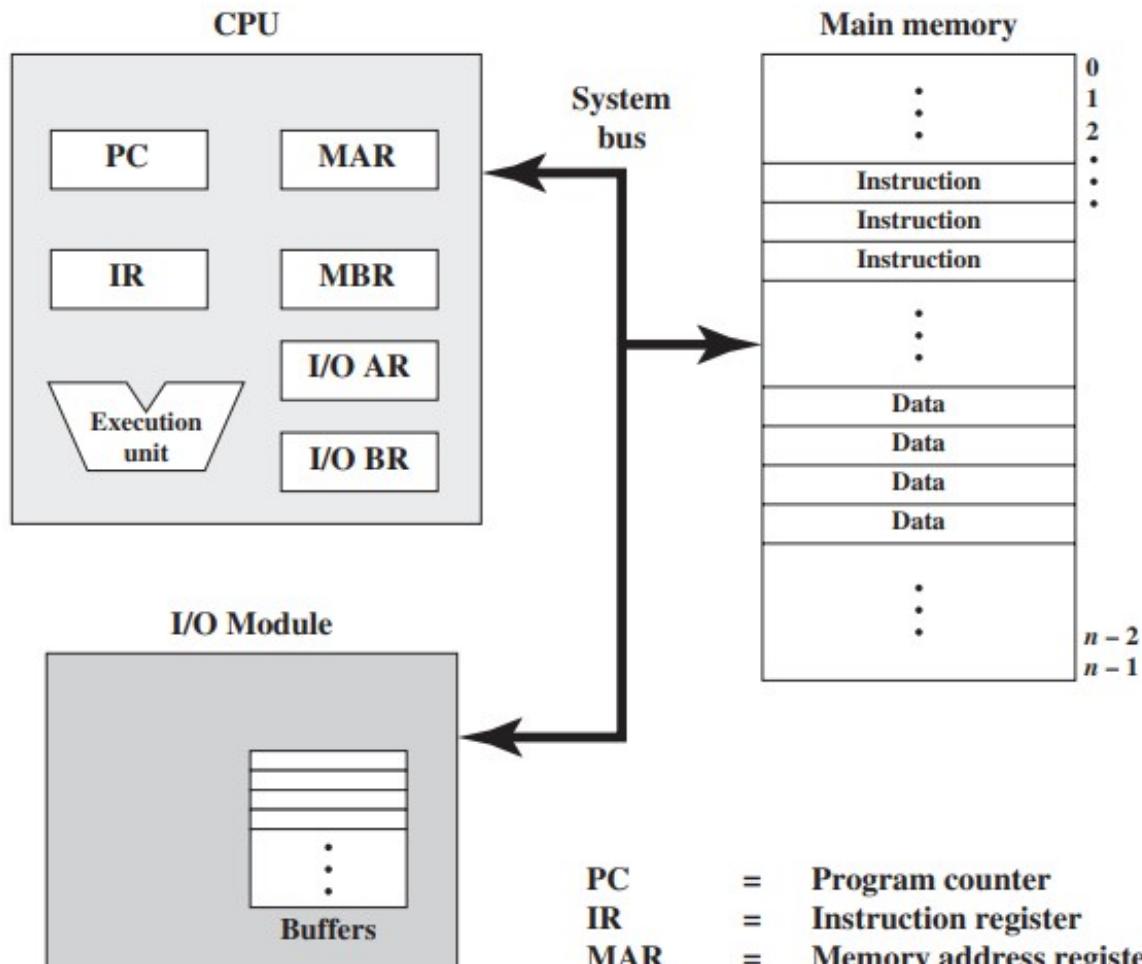


- Programming is now much easier. Instead of rewiring the hardware for each new program, all we need to do is provide a new sequence of codes.
- Each code is, in effect, an instruction, and part of the hardware interprets each instruction and generates control signals.
- To distinguish this new method of programming, a sequence of codes or instructions is called software.
- Figure (b) indicates two major components of the system: an instruction interpreter and a module of general-purpose arithmetic and logic functions.
- These two constitute the CPU.
- Several other components are needed to yield a functioning computer.
- Data and instructions must be put into the system.

- For this we need some sort of input module.
- This module contains **basic components for accepting data and instructions** in some form and converting them into an internal form of signals usable by the system.
- A means of reporting results is needed, and this is in the form of an **output module**.
- Taken together, these are referred to as I/O components.
- One more component is needed.
- An input device will bring **instructions and data in sequentially**.
- But a program is not invariably executed sequentially; it may jump around (e.g., the IAS jump instruction).

- Similarly, operations on data may require access to more than just one element at a time in a predetermined sequence. Thus, there must be a place to temporarily store both instructions and data.
- That module is called memory, or main memory, to distinguish it from external storage or peripheral devices. Von Neumann pointed out that the same memory could be used to store both instructions and data.

- Figure illustrates these top-level components and suggests the interactions among them.
 - The CPU exchanges data with memory.
 - For this purpose, it typically makes use of two internal (to the CPU) registers:
- a memory address register (MAR), which specifies the address in memory for the next read or write, and a memory buffer register (MBR), which contains the data to be written into memory or receives the data read from memory.



PC	=	Program counter
IR	=	Instruction register
MAR	=	Memory address register
MBR	=	Memory buffer register
I/O AR	=	Input/output address register
I/O BR	=	Input/output buffer register

- For this purpose, it typically makes use of two internal (to the CPU) registers:
- a memory address register (**MAR**), which specifies the address in memory for the next read or write, and a memory buffer register (**MBR**), which contains the data to be written into memory or receives the data read from memory. Similarly, an I/O address register (**I/OAR**) specifies a particular I/O device.
- An I/O buffer register (**I/OBR**) is used for the exchange of data between an I/O module and the CPU.
- A memory module consists of a set of locations, defined by sequentially numbered addresses. Each location contains a binary number that can be interpreted as either an instruction or data.
- An I/O module transfers data from external devices to CPU and memory, and vice versa.
- It contains internal buffers for temporarily holding these data until they can be sent on.

Features and Components of an ISA

- One of the most visible forms of evolution associated with computers is that of programming languages. As the cost of hardware has dropped, the relative cost of software has risen.
- Along with that, a chronic shortage of programmers has driven up software costs in absolute terms. Thus, the major cost in the life cycle of a system is software, not hardware.

Features and Components of an ISA

- Adding to the cost, and to the inconvenience, is the element of unreliability: it is common for programs, both system and application, to continue to exhibit new bugs after years of operation.
- The response from researchers and industry has been to develop ever more powerful and complex high- level programming languages.

- These high- level languages (HLLs):

- (1) allow the programmer to express algorithms more concisely;
- (2) allow the compiler to take care of details that are not important in the programmer's expression of algorithms; and
- (3) often support naturally the use of structured programming and/or object-oriented design.

- Alas, this solution gave rise to a perceived problem, known as the **semantic gap**, the difference between the operations provided in HLLs and those provided in computer architecture.
- Symptoms of this gap are alleged to include **execution inefficiency, excessive machine program size, and compiler complexity**.

- Designers responded with architectures intended to close this gap. Key features include large instruction sets, dozens of addressing modes, and various HLL statements implemented in hardware.
- An example of the latter is the CASE machine instruction on the VAX. Such complex instruction sets are intended to:
 - Ease the task of the compiler writer.
 - Improve execution efficiency, because complex sequences of operations can be implemented in microcode.

- Provide support for even more complex and sophisticated HLLs.
- Meanwhile, a number of studies have been done over the years to determine the characteristics and patterns of execution of machine instructions generated from HLL programs.
- The results of these studies inspired some researchers to look for a different approach:
 - namely, to make the architecture that supports the HLL simpler, rather than more complex.

To understand the line of reasoning of the RISC advocates, we begin with a brief review of instruction execution characteristics.

- The aspects of computation of interest are as follows:
 - **Operations performed:** These determine the functions to be performed by the processor and its interaction with memory.
 - **Operands used:** The types of operands and the frequency of their use determine the memory organization for storing them and the addressing modes for accessing them.
 - **Execution sequencing:** This determines the control and pipeline organization.

Instruction Execution Characteristics

- High Level Language (HLL) programs mostly comprises-
 - Assignment statements (e.g. $A \leftarrow B$)
 - Conditional statements (e.g. IF, LOOP)
 - Procedure call/return (e.g. Functions in C)
- Key for ISA designer-
 - These statements should be supported in optimal fashion

RISC Architecture

- Earlier computers had small and simple instruction set.
Why???
 - In the early 1980's, designers recommended that computers use **fewer instructions** with **simple constructs** so they can be executed much faster without accessing memory as often.

RISC Characteristics

- Relatively few instructions
 - 128 or less
- Relatively few addressing modes
 - Memory access is limited to LOAD and STORE instructions
- All operations done within the registers of the CPU
 - Use of overlapped register windows for optimization
- Fixed Length (4 Bytes), easily decoded instruction format
 - Instruction execution time consistent
- Hardwired control

RISC Processors

- MIPS R4000
 - First commercially available RISC processor
 - Supports thirty-two 64-bit registers
 - 128KB of high speed cache
- SPARC (Sun)
 - Based on Berkeley RISC model
- PowerPC (IBM)
- ARM processor family
- Apple iPods

CISC Architecture

- Later complexity and number of instructions both are increased. Why???
 - Simplify the compilation
- The trend into computer hardware complexity was influenced by various factors, such as
 - To provide support for more customer applications
 - Adding instructions that facilitate the translation from high level language into machine language programs
 - Single machine instruction for each high level language statement
 - Ex: VAX computer, IBM/370 computers, Intel x86 based processors, Motorola 68000 Series, Virtual address extension

CISC Characteristics

- A large number of instructions
- Some instructions for special tasks used infrequently
- A large variety of addressing modes (5 to 20)
- Variable length instruction formats
- Instructions that manipulate operands in memory
- However, it soon became apparent that a complex instruction set has a number of disadvantages
- These include a **complex instruction decoding** scheme, an **increased size of the control unit**,

RISC and CISC Comparison



- Computer performance equation
 - $\text{CPU Time} = (\text{instructions/program}) \times (\text{avg. cycles/instruction}) \times (\text{seconds/cycle})$
- Example CISC Program
 - `mov ax, 20`
 - `mov bx, 5`
 - `mul ax, bx`

Example RISC
Program

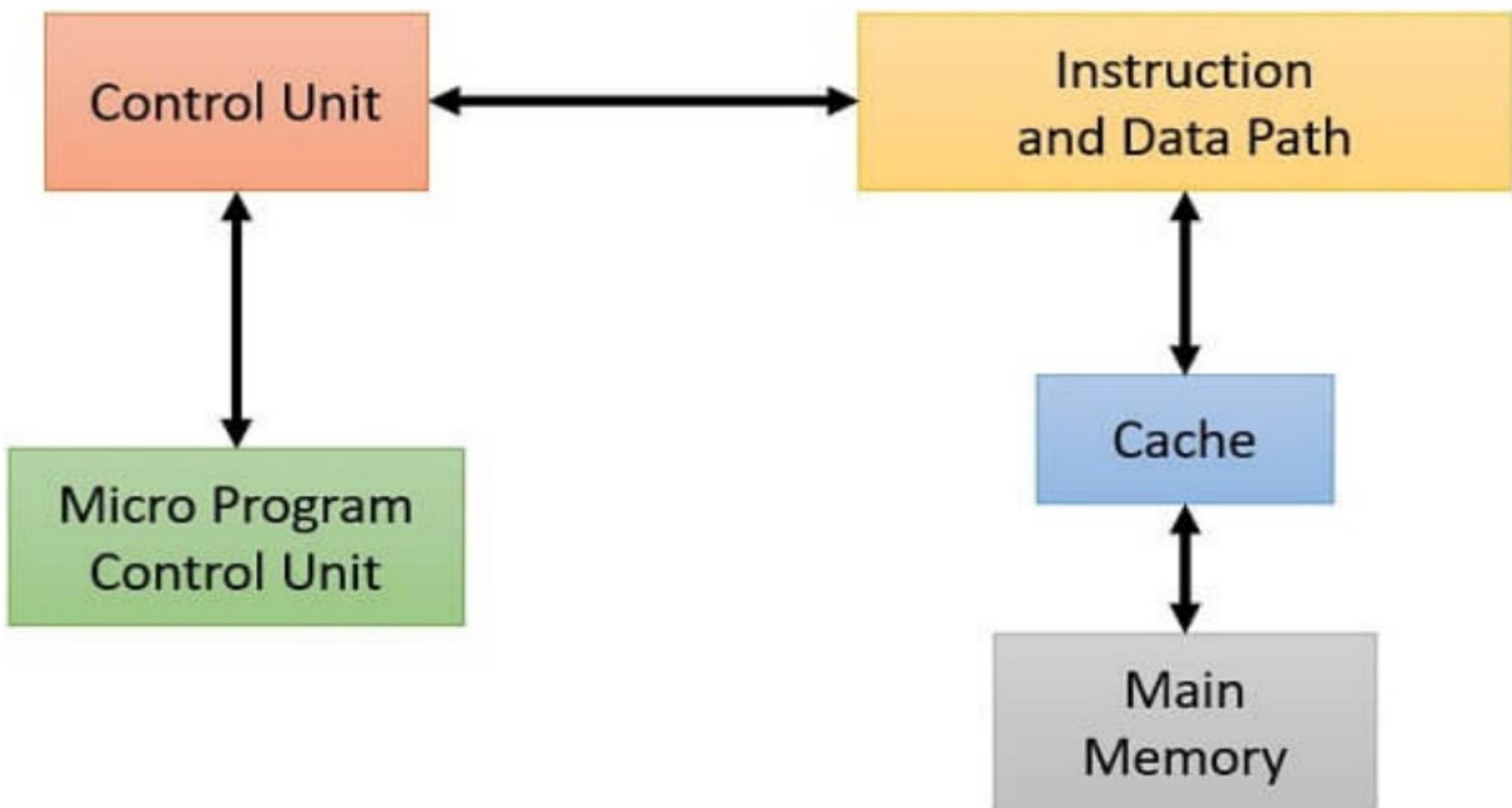
```
mov ax,0
mov bx, 20
mov cx,5
again add ax, bx
loop again
```

RISC vs CISC Performance Summary

- The CISC approach attempts to minimize the number of instructions per program by sacrificing the number of cycles per instruction.
- RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program.

RISC Vs. CISC Controversy

- Now the general trend in computer architecture and organization has been toward-
 - Increasing processor complexity
 - More instructions
 - More addressing modes
 - More registers
- No definitive test set of programs exists to compare both architectures
 - Performance varies with the programs
- Most commercially available machines are mixture of RISC and CISC characteristics



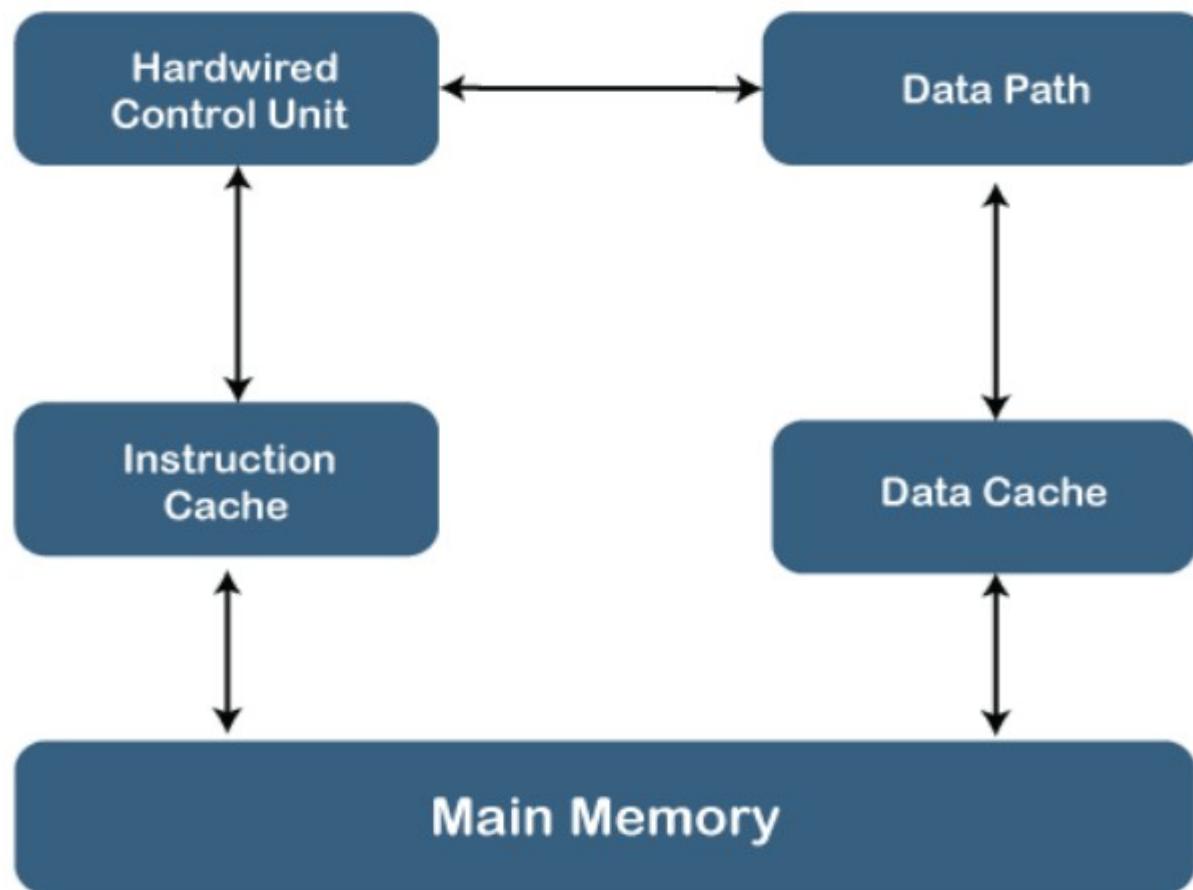
CISC Architecture

Advantages of CISC Processors

1. The code size is comparatively shorter which minimizes the memory requirement.
2. Execution of a single instruction accomplishes several low-level tasks.
3. Complex addressing mode makes the memory access flexible.
4. CISC instruction can directly access memory locations.

Disadvantages of CISC Processors

1. Though the code size is minimized but it requires several clock cycles to execute a single instruction. Thereby reduce the overall performance of the computer.
2. Implementing **pipelining** for CISC instruction is a bit complicated.
3. The **hardware structure** needs to be more complex to simplify software implementation.
4. Designed to minimize the memory requirement when memory was smaller and costlier. But situation is different today



RISC Architecture

Advantages of RISC Processors

1. The RISC processor's performance is better due to the simple and **limited number of the instruction set.**
2. It requires several transistors that make it cheaper to design.
3. RISC allows the **instruction to use free space on a microprocessor** because of its simplicity.
4. RISC processor is simpler than a CISC processor because of its simple and quick design, and it can complete its work in one clock cycle.

Disadvantages of RISC Processors

1. The RISC processor's performance may vary according to the code executed because subsequent instructions may depend on the previous instruction for their execution in a cycle.
2. Programmers and compilers often use complex instructions.
3. RISC processors require very fast memory to save various instructions that require a large collection of cache memory to respond to the instruction in a short time.

CISC vs RISC

- After the initial enthusiasm for RISC machines, there has been a growing realization that
 - (1) RISC designs may benefit from the inclusion of some CISC features and that
 - (2) CISC designs may benefit from the inclusion of some RISC features.
- The result is that the more recent RISC designs, notably the PowerPC, are no longer “pure” RISC and the more recent CISC designs, notably the Pentium II and later Pentium models, do incorporate some RISC characteristics.

CISC vs RISC

- For many years, the general trend in computer architecture and organization **has been toward increasing processor complexity:**
 - more instructions, more addressing modes, more specialized registers, and so on.
 - The RISC movement represents a fundamental break with the philosophy behind that trend.

- Naturally, the appearance of RISC systems, and the publication of papers by its proponents extolling RISC virtues, led to a reaction from those involved in the design of CISC architectures.
- The work that has been done on assessing merits of the RISC approach can be grouped into two categories:
 - **Quantitative:** Attempts to compare program size and execution speed of programs on RISC and CISC machines that use comparable technology.
 - **Qualitative:** Examines issues such as high- level language support and optimum use of VLSI real estate.

- Most of the work on quantitative assessment has been done by those working on RISC systems, and it has been, by and large, favorable to the RISC approach.
- Others have examined the issue and come away unconvinced.

There are several problems with attempting such comparisons:

- There is no pair of RISC and CISC machines that are comparable in life-cycle cost, level of technology, gate complexity, sophistication of compiler, operating system support, and so on.
- No definitive test set of programs exists. Performance varies with the program.

- Most of the comparative analysis on RISC has been done on “toy” machines rather than commercial products.
 - Furthermore, most commercially available machines advertised as RISC possess a mixture of RISC and CISC characteristics.
 - Thus, a fair comparison with a commercial, “pure-play” CISC machine (e.g., VAX, Pentium) is difficult.

- The qualitative assessment is, almost by definition, subjective.
 - Several researchers have turned their attention to such an assessment, but the results are, at best, ambiguous, and certainly subject to rebuttal and, of course, counter-rebuttal [COLW85b].
 - In more recent years, the RISC versus CISC controversy has died down to a great extent. This is because there has been a gradual convergence of the technologies.

- As chip densities and raw hardware speeds increase, RISC systems have become more complex.
- At the same time, in an effort to squeeze out maximum performance, CISC designs have focused on issues traditionally associated with RISC, such as an increased number of general-purpose registers and increased emphasis on instruction pipeline design.



RISC vs. CISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

The CISCs: x86, VAX (Virtual Address eXtension to PDP-11)

- Variable length instructions: 1-321 bytes!!!
- 14 GPRs + PC + stack-pointer + condition codes
- Data sizes: 8, 16, 32, 64, 128 bit, decimal, string
- Memory-memory instructions for all data sizes

The RISCs: MIPS, PA-RISC, SPARC, PowerPC, Alpha, ARM

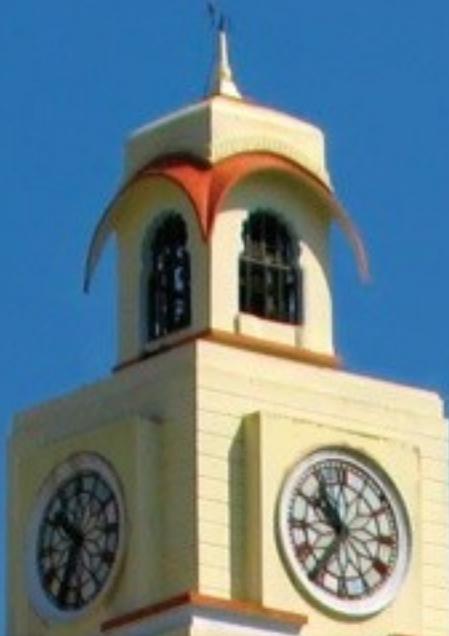
- 32-bit instructions
- 32 integer registers, 32 floating point registers, load-store
- 64-bit virtual address space
- Few addressing modes (Alpha has one, SPARC/PowerPC have more)

Design Goals

- The ultimate goals of the ISA designer are:
 - To create an ISA that allows for fast hardware implementations
 - To simplify choices for the compiler
 - To ensure the longevity of the ISA by anticipating future technology trends
- Often trade-offs (particularly between 1 & 2)
- Examples ISAs: X86, PowerPC, SPARC, ARM, MIPS, IA-64
- May have multiple hardware implementations of the same ISA
- Example: i386, i486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium IV



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 4

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Thangakumar J
WILP & Department of CS & IS



Last Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Computer Architecture Vs. Organization (Abstraction vs. Realization)• Von-Neumann Architecture vs Harvard Architecture• Features and Components of an ISA.• Design Goals	<p>R6 (1.1)</p> <p>R6 (3.1)</p> <p>R6 (15.1)</p>

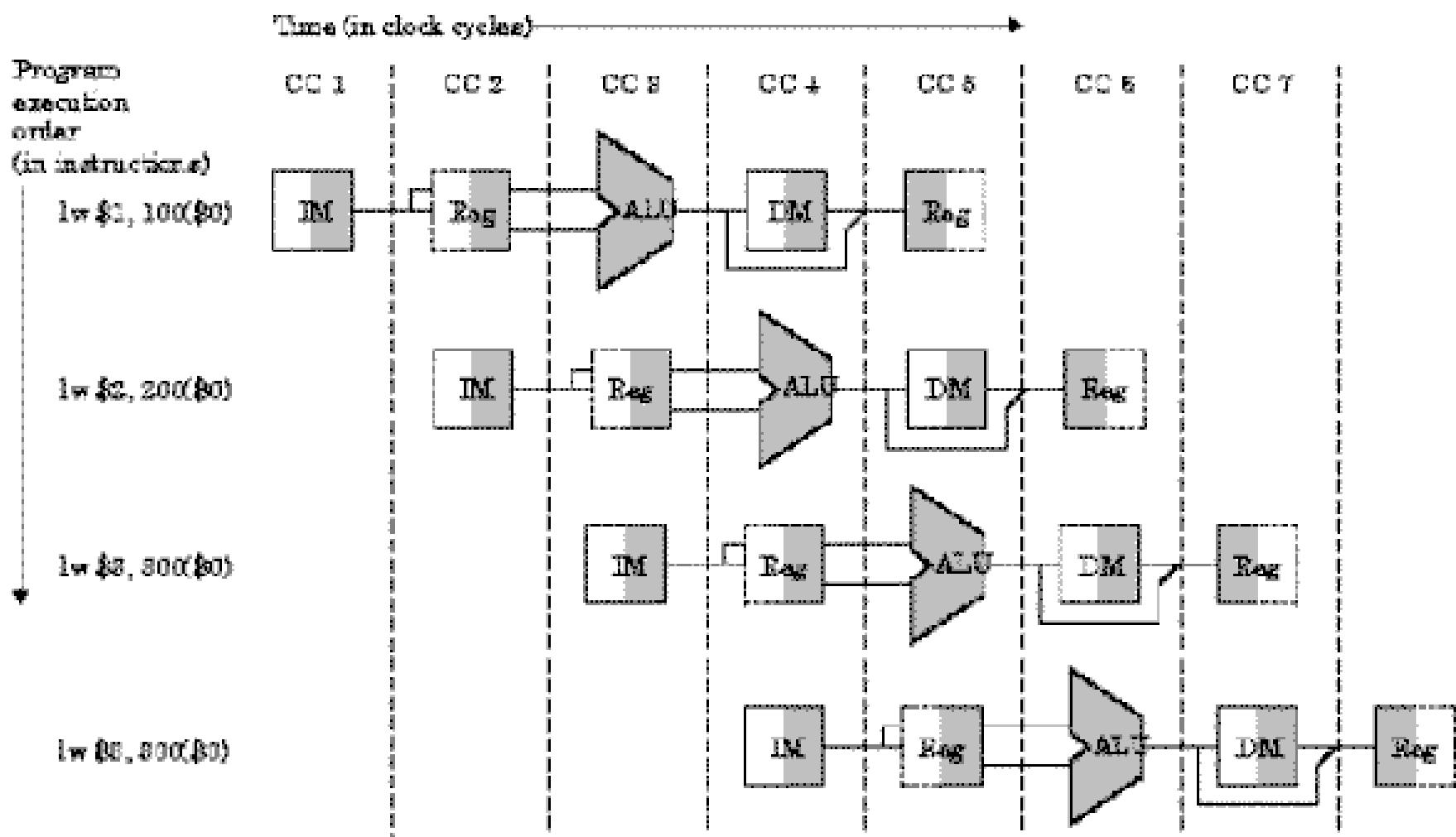
ISA as the Interface

Today's topic

- Understanding MIPS/RISC architecture

MIPS Architecture

- MIPS follows RISC principles and based on Harvard Architecture
- MIPS architecture is a register architecture
- MIPS ISA is Load/Store architecture
- R2000 is a 32-bit processor
- Uses fixed length instruction format.



- The R2000 is a 32-bit microprocessor chip set developed by MIPS Computer Systems that implemented the MIPS I instruction set architecture (ISA).
- Introduced in January 1986, it was the first commercial implementation of the MIPS architecture and the first commercial RISC processor available to all companies.
- The R2000 competed with Digital Equipment Corporation (DEC) VAX minicomputers and with Motorola 68000 and Intel Corporation 80386 microprocessors.
- R2000 users included Ardent Computer, DEC, Silicon Graphics, Northern Telecom and MIPS's own Unix

MIPS Register Set

Types of Registers

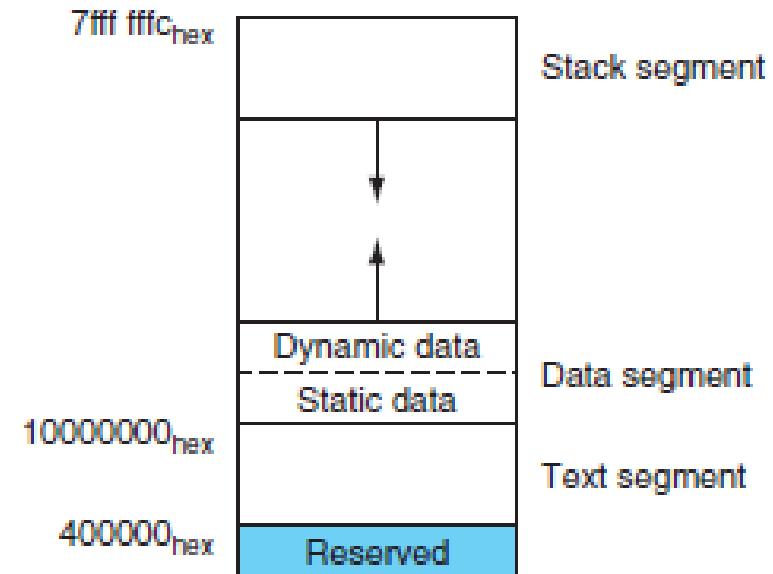
- 32 general-purpose registers (\$0 – \$31)
- Program Counter (PC)
- Two special Purpose register (HI and LO)
 - Used to hold the results of integer *multiply* and *divide* instruction.
 - integer *multiply* operation, HI and LO register hold the 64-bit result.
 - integer *divide* operation, the 32-bit quotient is stored in the LO and the remainder in the HI register.

Usage Convention

Register name	Number	Intended usage
Zero	0	Constant 0
at	1	Reserved for assembler
v0,v1	2,3	Values for function results and Exp evaluation
a0,a1,a2,a3	4-7	Arguments 1-4
t0-t7	8-15	Temporary (not preserved across call)
s0-s7	16-23	Saved temporary(preserved across call)
t8,t9	24,25	Temporary (not preserved across call)
k0,k1	26,27	Reserved for OS kernel
gp	28	Pointer to Global area
sp	29	Stack Pointer
fp	30	Frame pointer(if needed);Otherwise, a saved register \$s8
ra	31	Return address(used by a procedure call)

Memory Usage

- A program's address space consists of three parts:
 - Code/Text segment
 - Data segment
 - Stack segment

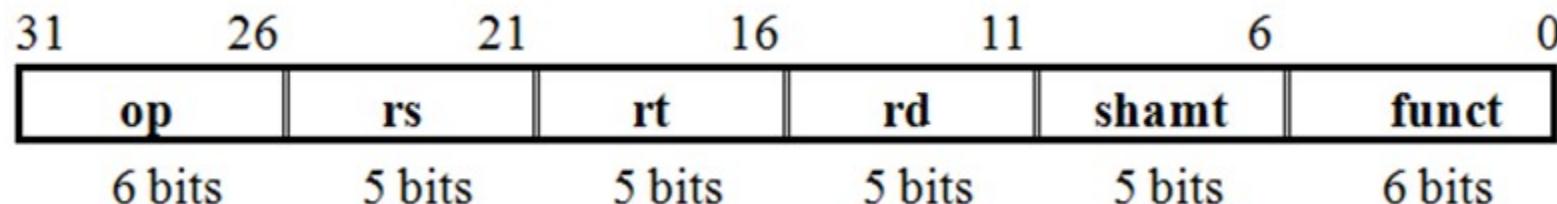


Instruction Format

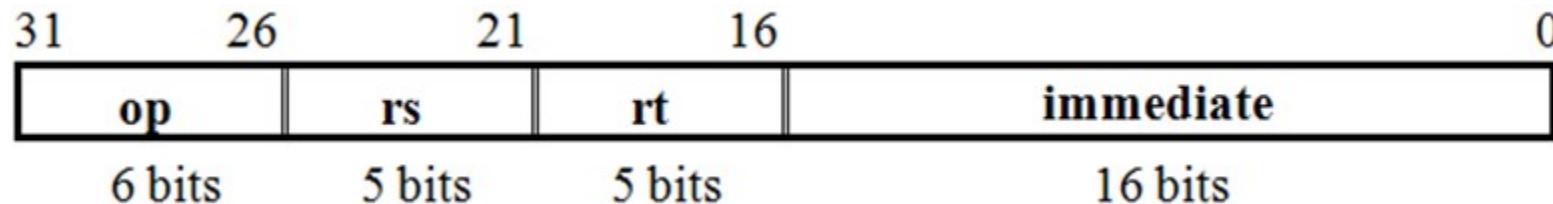
- Fixed-length instruction format
- 32-bits long
- Three different instruction formats:
 1. Immediate (I-type)
 2. Jump (J-type)
 3. Register (R-type)
- Meaning of various fields in the instruction format
 - op: Opcode
 - rs: The first register source operand
 - rt: The second register source operand
 - rd: The register destination operand
 - shamt / sa: shift amount
 - funct: function code

Instruction Format

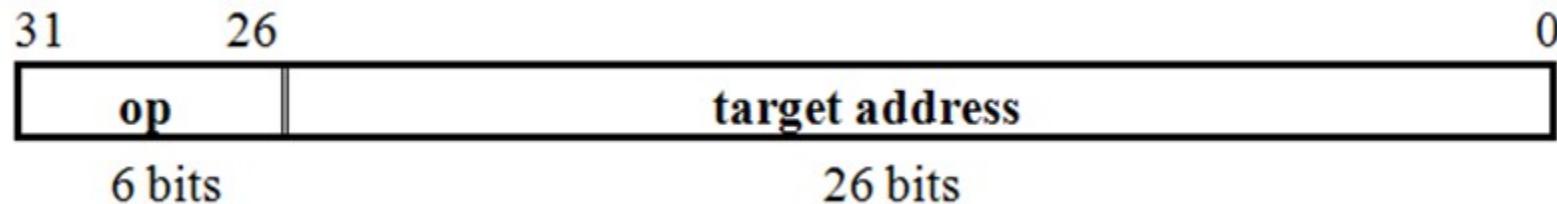
R-Type:



I-Type:



J-Type:



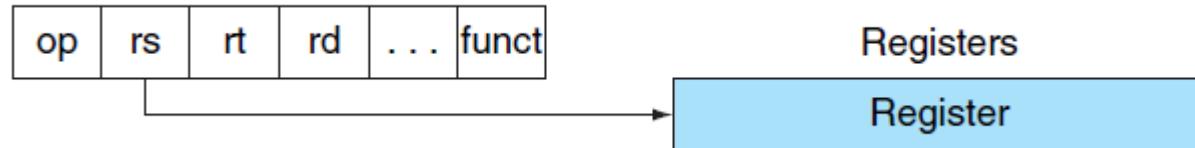
Addressing modes

- Immediate Addressing Mode

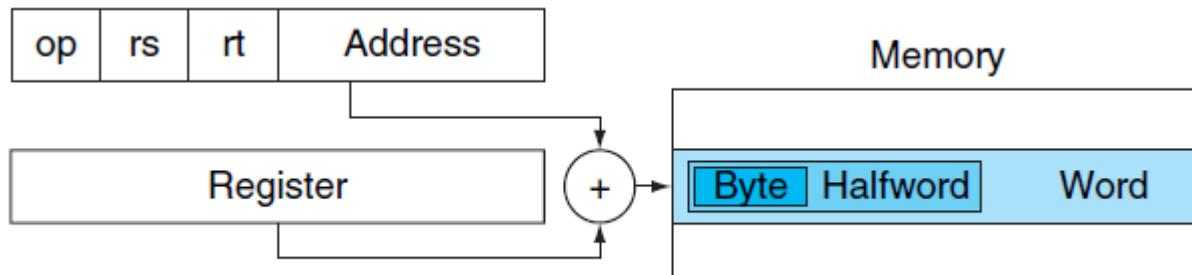
1. Immediate addressing



- Register Addressing Mode

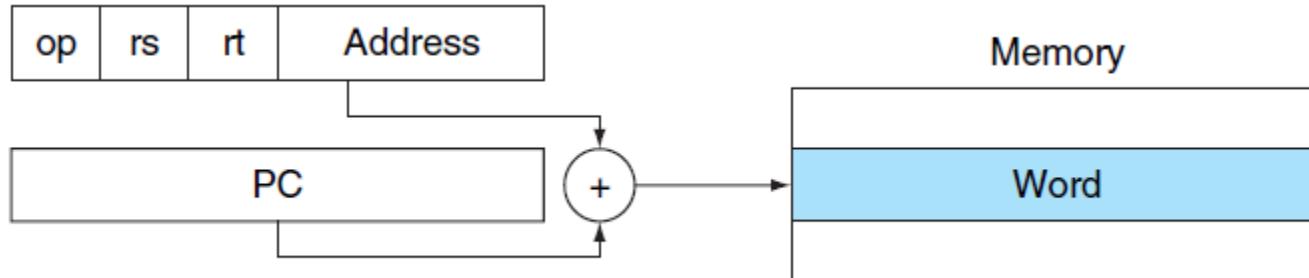


- Base or Displacement Addressing Mode

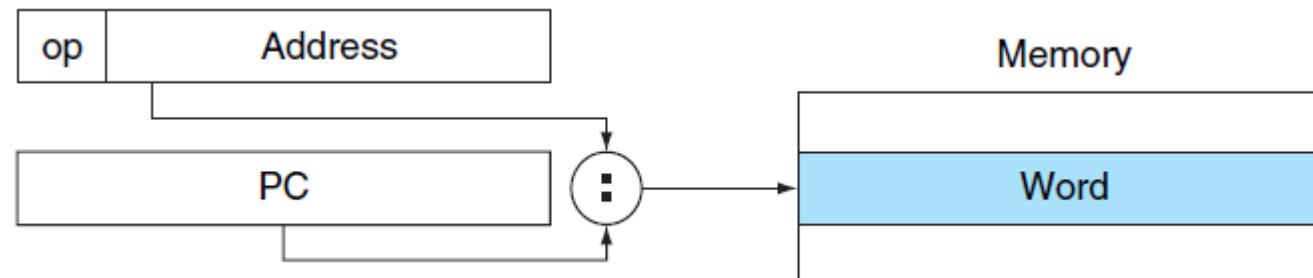


Addressing modes

- PC- Relative Addressing Mode



- Pseudo Direct Addressing Mode





Immediate Addressing Mode



Eg: 1. addi \$s0, \$zero,7 $\$s0 = 0 + 7$

Eg: 4. ori \$t0,\$t1,8 $\$t0 = \$t1 | 8$





Register Addressing Mode



Eg:1. add \$s1,\$s2,\$s3

$\$s1 = \$s2 + \$s3$

Eg:2. or \$t0,\$t1,\$t2

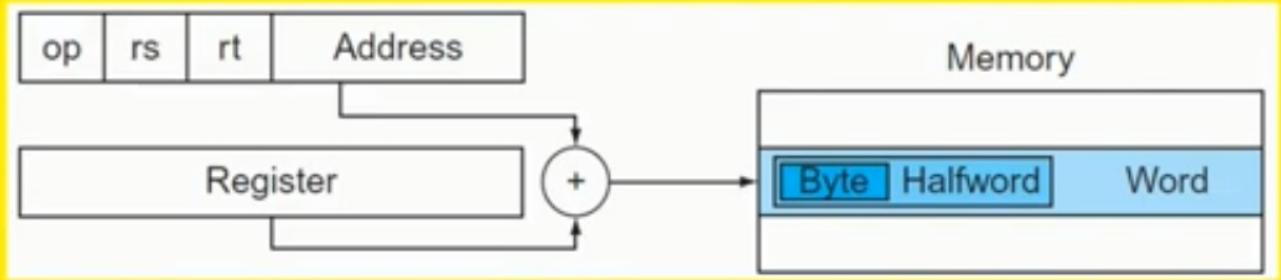
$\$t0 = \$t1 + \$t2$

Eg:3. srl \$s7,\$s0,4

$\$s7 = \$s0 \gg 4$



Base / Displacement Addressing Mode



Eg:1. **lw \$s1, 4(\$s2)**

\$s1 = Mem [\$s2+4]

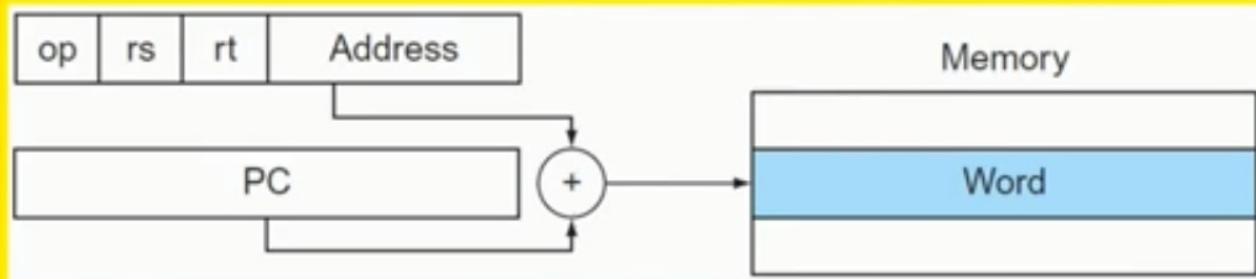
Eg:2. **sw \$s0, 8(\$s4)**

Mem [\$s4+8] = \$s0





PC - Relative Addressing Mode

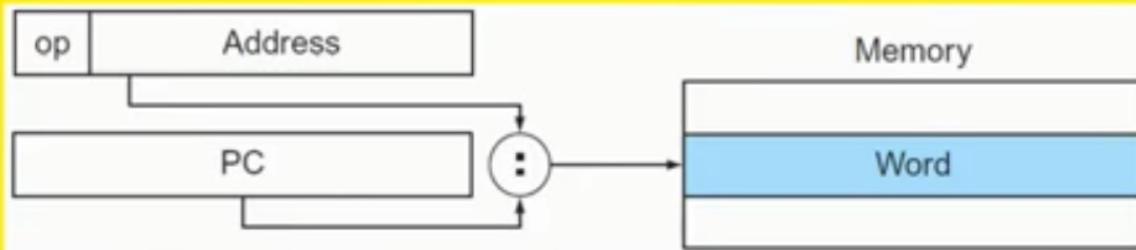


Eg:1. **beq \$s1, \$s2, 25** if(\$s1==\$s2) goto PC+4+100

Eg:2. **bne \$s0, \$s1, 20** if(\$s0!=\$s1) goto PC+4+80



Pseudo Direct Addressing Mode



Eg:1. j 2500 go to 10000

Eg:2. jr \$ra goto \$ra

Eg:3. jal 2500 \$ra=PC+4 goto 10000



EXAMPLE

Starting address 8000

```
Loop: add $t1,$s3,$s3
      add $t1,$t1,$t1
      add $t1,$t1,$s6
      lw $t0,0($t1)
      bne $t0,$s5,Exit
      add $s3,$s3,$s4
      j loop
```

Exit:

Name	Register Number
\$zero	0
\$v0 - \$v1	2-3
\$a0 - \$a3	4-7
\$t0 - \$t7	8-15
\$s0 - \$s7	16-23
\$t8 - \$t9	24-25
\$gp	28
\$sp	29
\$fp	30
\$ra	31

Register name	Number
Zero	0
at	1
v0,v1	2,3
a0,a1,a2,a3	4-7
t0-t7	8-15
s0-s7	16-23
t8,t9	24,25
k0,k1	26,27
gp	28
sp	29
fp	30
ra	31

EXAMPLE

Starting address 8000

Loop: add \$t1,\$s3,\$s3
 add \$t1,\$t1,\$t1
 add \$t1,\$t1,\$s6
 lw \$t0,0(\$t1)
 bne \$t0,\$s5,Exit
 add \$s3,\$s3,\$s4
 j loop

Exit:

Instruction	Opcode
j label	000010

Instruction	Opcode
addi rt, rs, immediate	001000
addiu rt, rs, immediate	001001
andi rt, rs, immediate	001100
beq rs, rt, label	000100
bgez rs, label	000001
bgtz rs, label	000111
blez rs, label	000110
bltz rs, label	000001
bne rs, rt, label	000101
lb rt, immediate(rs)	100000
lbu rt, immediate(rs)	100100
lh rt, immediate(rs)	100001
lhu rt, immediate(rs)	100101
lui rt, immediate	001111
lw rt, immediate(rs)	100011
lwcl rt, immediate(rs)	110001
ori rt, rs, immediate	001101

Instruction	Function
add rd, rs, rt	100000
addu rd, rs, rt	100001
sub rd, rs, rt	100010
subu rd, rs, rt	100011
and rd, rs, rt	100100
or rd, rs, rt	100101
xor rd, rs, rt	100110
nor rd, rs, rt	100111
slt rd, rs, rt	101010
sltu rd, rs, rt	101011
sll rd, rt, sa	000000
srl rd, rt, sa	000010
sllv rd, rt, rs	000100
srlv rd, rt, rs	000110
sra rd, rt, sa	000011
srav rd, rt, rs	000111

EXAMPLE

Starting address 8000
 Loop: add \$t1,\$s3,\$s3
 add \$t1,\$t1,\$t1
 add \$t1,\$t1,\$s6
 lw \$t0,0(\$t1)
 bne \$t0,\$s5,Exit
 add \$s3,\$s3,\$s4
 j loop

Exit:

8000
 8004
 8008
 8012
 8016
 8020
 8024
 8028

0	19	19	9	0	32
0	9	9	9	0	32
0	9	22	9	0	32
35	9	8		0	
5	8	21		2	
0	19	20	19	0	32
2					

MIPS Instruction Set

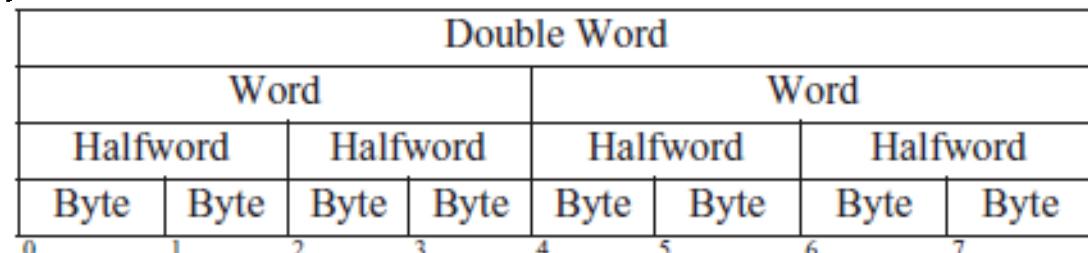
- Arithmetic Instructions (ADD, SUB etc.)
- Logical Instructions (OR, AND, SHIFT, etc.)
- Data Transfer Instructions (Load and Store)
- Decision Making Instructions (J, BNE, BEQ, etc.)
- Stack Related Instructions (PUSH and POP)

A Basic MIPS Implementation

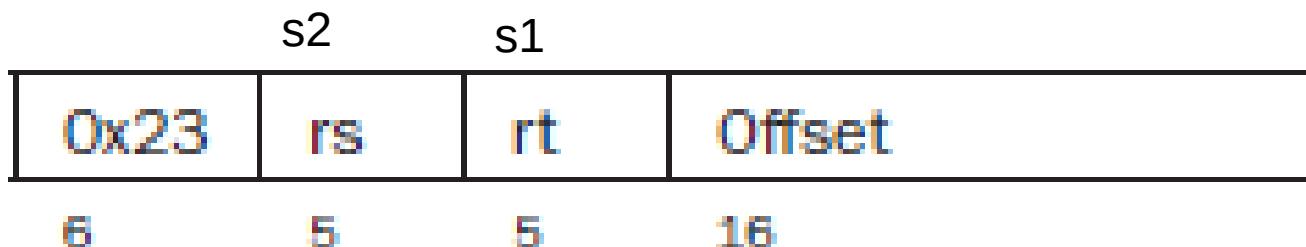
- Basic implementation includes a subset of core MIPS instruction set
 - Memory reference instruction
 - lw and sw
 - Arithmetic and logical instructions
 - add, sub, and, or and slt (set if less than)
 - Branch instructions
 - beq and j

Memory Reference Instruction - lw

- copies data from memory to register
- Format : lw reg, address <http://alumni.cs.ucr.edu/~vladimir/cs161/mips.html>
- Example : lw \$s1, 100(\$s2)
- $\$s1 \leftarrow \text{memory}[100 + \$s2]$
- Alignment restriction
- MIPS is **Big endian**
- **Spilling registers**



<https://student.cs.uwaterloo.ca/~isg/res/mips/opcodes>



Memory Reference Instruction - sw

- Store Instruction
- copies data from register to memory
- Format : sw reg, address
- Example : sw \$s1, 100(\$s2)

memory[100 +\$s2] \leftarrow \$s1

s2	s1		
0x2b	rs	rt	Offset
6	5	5	16

Arithmetic Instructions :add

add des, src1, src2

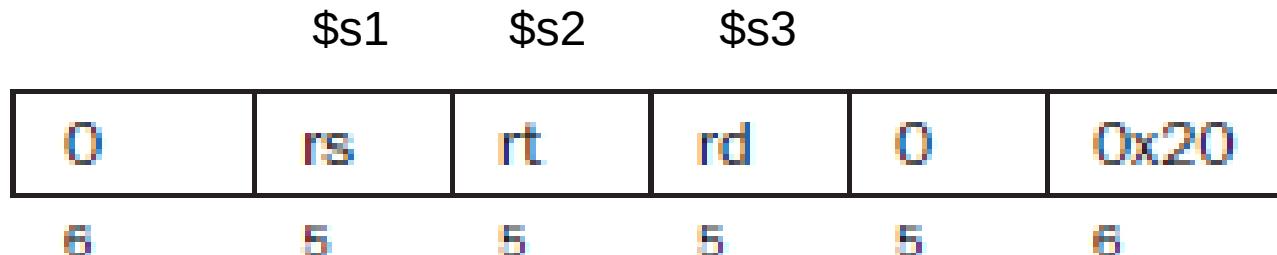
Addressing mode: register

Example: add \$s3, \$s1, \$s2

meaning : $\$s3 = \$s1 + \$s2$

opcode : 0 , function: 0x20

sa :0



The opcode is the machine code representation of the instruction mnemonic. The opcode field is 6 bits long (bit 26 to bit 31) but always set to 0 in the R format. The actual instruction to perform is placed in the funct field.

Arithmetic Instructions :sub

sub des, src1, src2

Addressing mode: register

Example: sub \$s3, \$s1, \$s2

meaning : $\$s3 = \$s1 - \$s2$

opcode : 0 , function: 0x22

sa :0

	\$s1	\$s2	\$s3		
0	rs	rt	rd	0	0x22
6	5	5	5	5	6

Logical instructions :and

and des, src1, src2

Addressing mode: register

Example: and \$s3, \$s1, \$s2

meaning : \$s3 = \$s1 & \$s2 (bit by bit)

opcode : 0 , function: 0x24

sa :0

	\$s1	\$s2	\$s3		
0	rs	rt	rd	0	0x24
6	5	5	5	5	6

Logical instructions :or

or des, src1, src2

Addressing mode: register

Example: or \$s3, \$s1, \$s2

meaning : $\$s3 = \$s1 \mid \$s2$ (bit by bit)

opcode : 0 , function: 0x25

sa :0

	\$s1	\$s2	\$s3		
0	rs	rt	rd	0	0x25
6	5	5	5	5	6

Logical instructions :slt

set on less than : slt

Format: slt des, src1, src2

set des = 1, if src1 < src2

Example:

slt \$s3, \$s1, \$s2

	\$s1	\$s2	\$s3		
	0	rs	rt	rd	0
6	5	5	5	5	6

Set register rd to 1 if rs is less than rt, and to 0 otherwise

Branch Instructions: beq and j

beq : branch if equal

- format : beq \$s1, \$s2, label
- If $\$s1 = \$s2$ then branch to address specified as

label			
4	rs	rt	Offset
6	5	5	16

j : jump unconditional

- format : j label

2	target
6	26

- and \$s1, \$s2, \$s3

0	rs	rt	rd	0	0x24
6	5	5	5	5	6

Summary

- lw \$s1, 100(\$s2)

0x23	rs	rt	Offset
6	5	5	16

- sw \$s1, 100(\$s2)

0x2b	rs	rt	Offset
6	5	5	16

- add \$s1, \$s2, \$s3

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

- sub \$s1, \$s2, \$s3

0	rs	rt	rd	0	0x22
6	5	5	5	5	6

- or \$s1, \$s2, \$s3

0	rs	rt	rd	0	0x25
6	5	5	5	5	6

- slt \$t0, \$s1, \$s2

0	rs	rt	rd	0	0x2a
6	5	5	5	5	6

- beq \$s1, \$s2, label

4	rs	rt	Offset
6	5	5	16

- j label

2	target
6	26



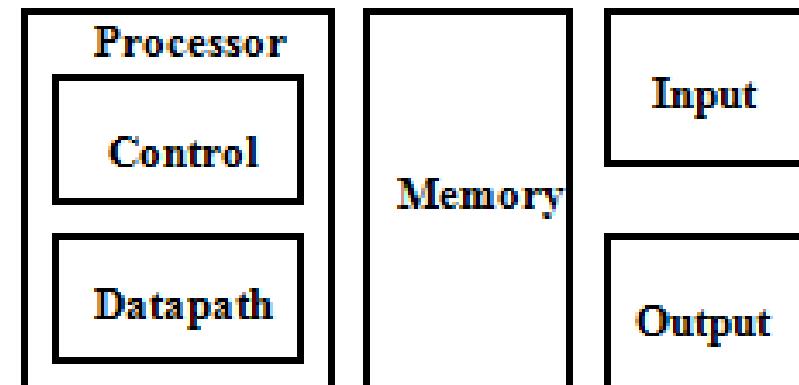
BITS Pilani
Pilani Campus

Data Path Design



Introduction

- The Five Classic Components of a Computer
 - Datapath: The component of the processor that performs data processing operations
 - Control: The component of the processor that commands the datapath, memory and I/O devices according to the instructions of the memory
 - Memory
 - Input device
 - Output device



Building a datapath

- Datapath elements :memory – instruction/data, register file, ALU, Adders, Multiplexers....
- Demonstrate Datapath implementation for the following instruction:
 - lw and sw
 - add, sub, and, or and slt
 - beq and j
- Instruction Cycle = Fetch + Execute

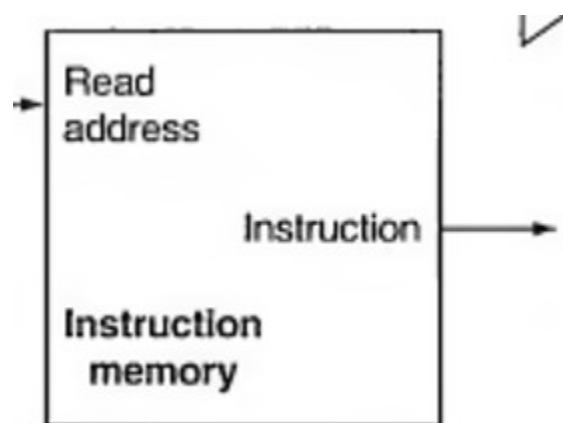
Lets Do this...

write a datapath used for fetching instructions and incrementing the program counter

Step number 1
Instruction memory

It has instructions

Given instruction address it gives instruction

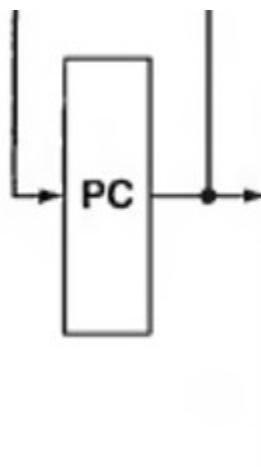


Lets Do this...

write a datapath used for fetching instructions and incrementing the program counter

Step number 2

PC – Program counter is a 32 bit register



It has memory address of where the program is

Updated after every clock cycle

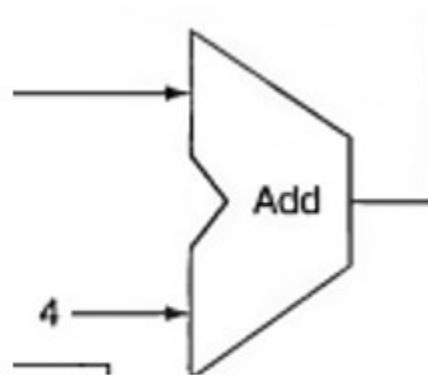
Readable and writeable

Lets Do this...

write a datapath used for fetching instructions and incrementing the program counter

Step number 3

Adder



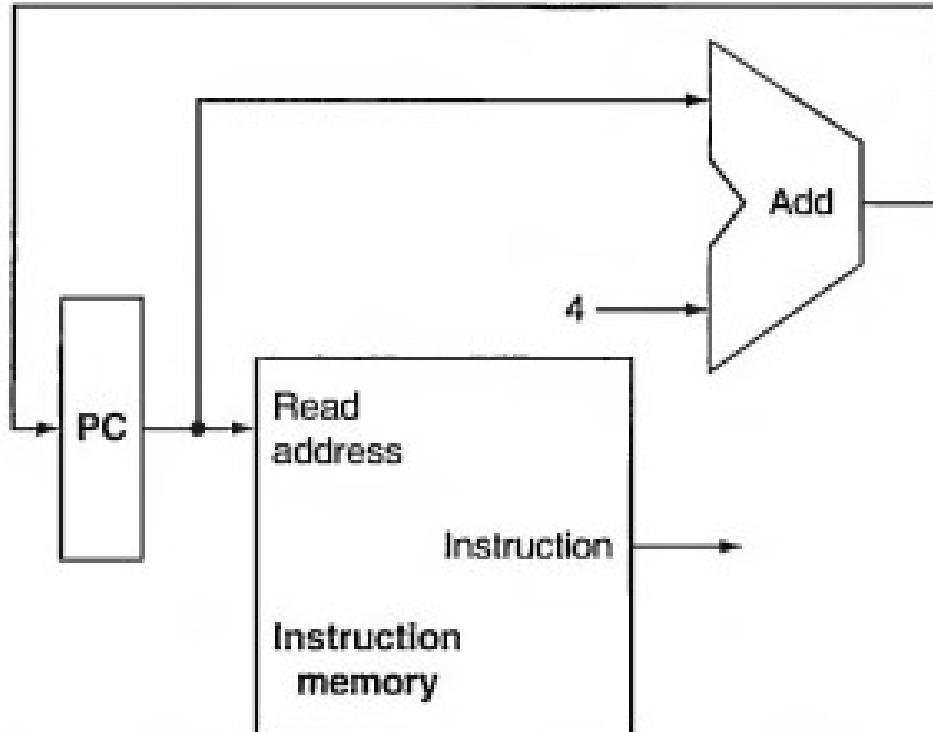
It takes two input values and gives an output

Takes PC and 4 (as address is 32 bits, 4 bytes) and gives output

Output is the next instruction address

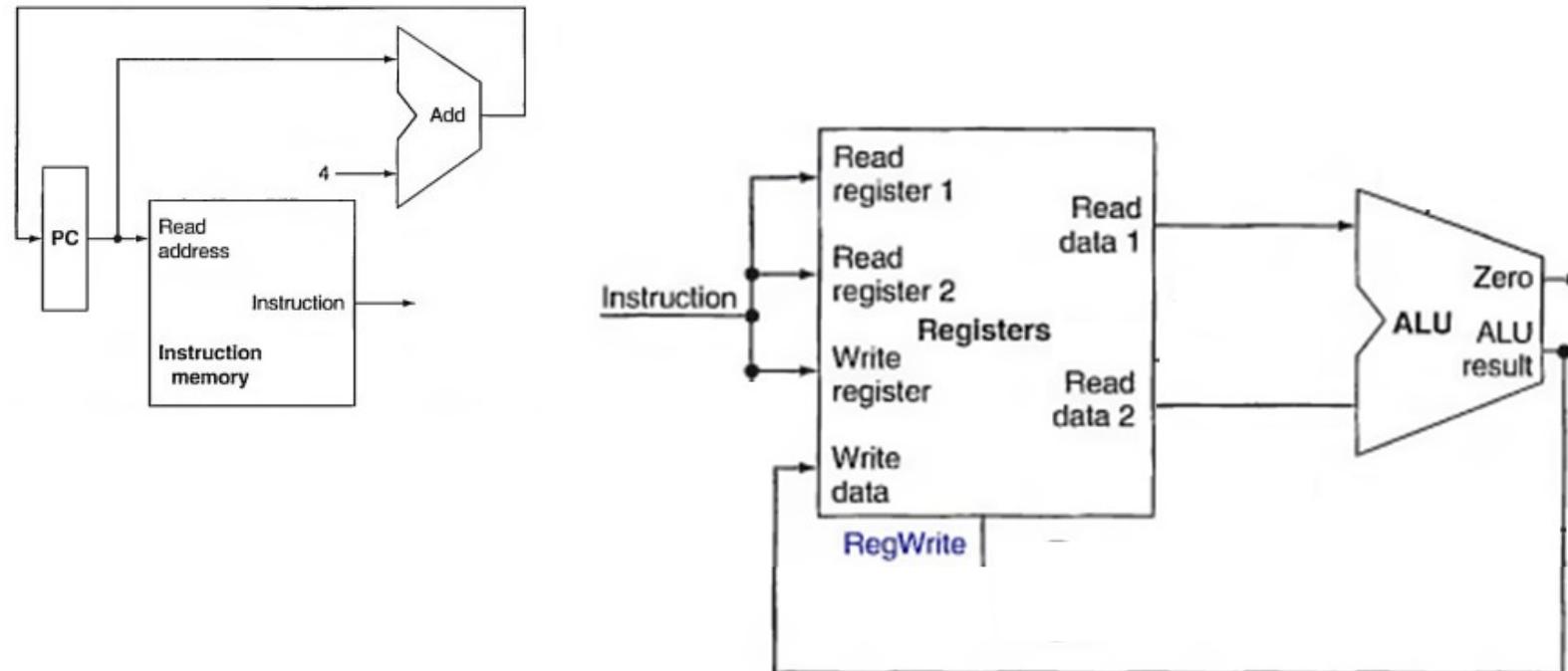
Lets Do this...

write a datapath used for fetching instructions and incrementing the program counter

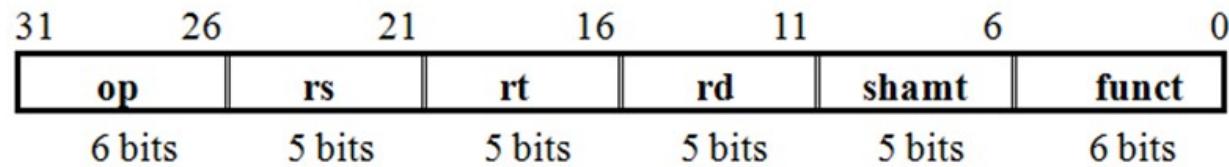


Steps - Data path for R-Type

1. Get instruction from Program Counter

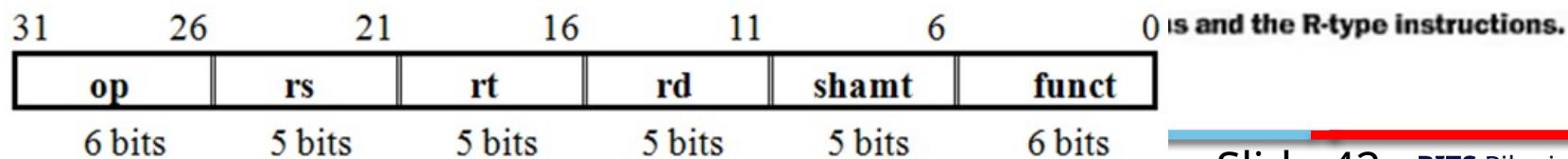
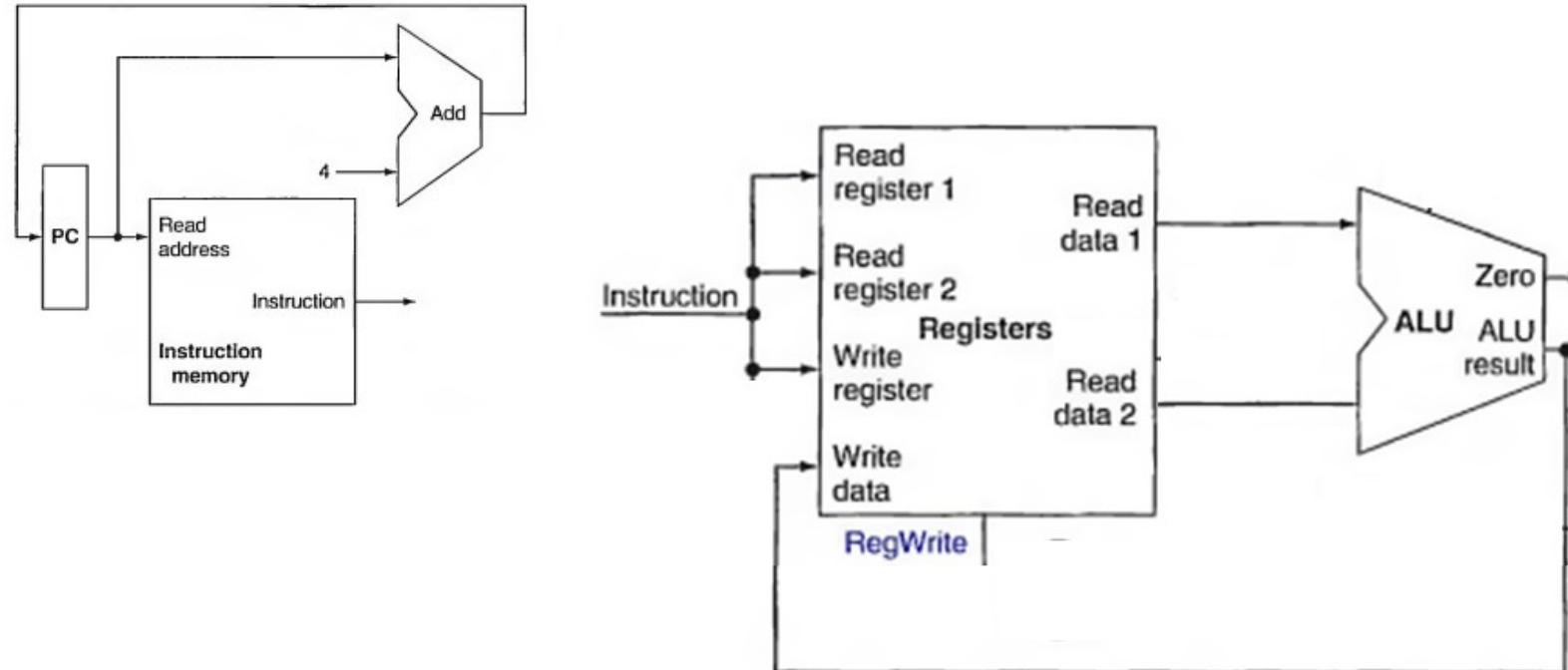


The datapath for the memory instructions and the R-type instructions.



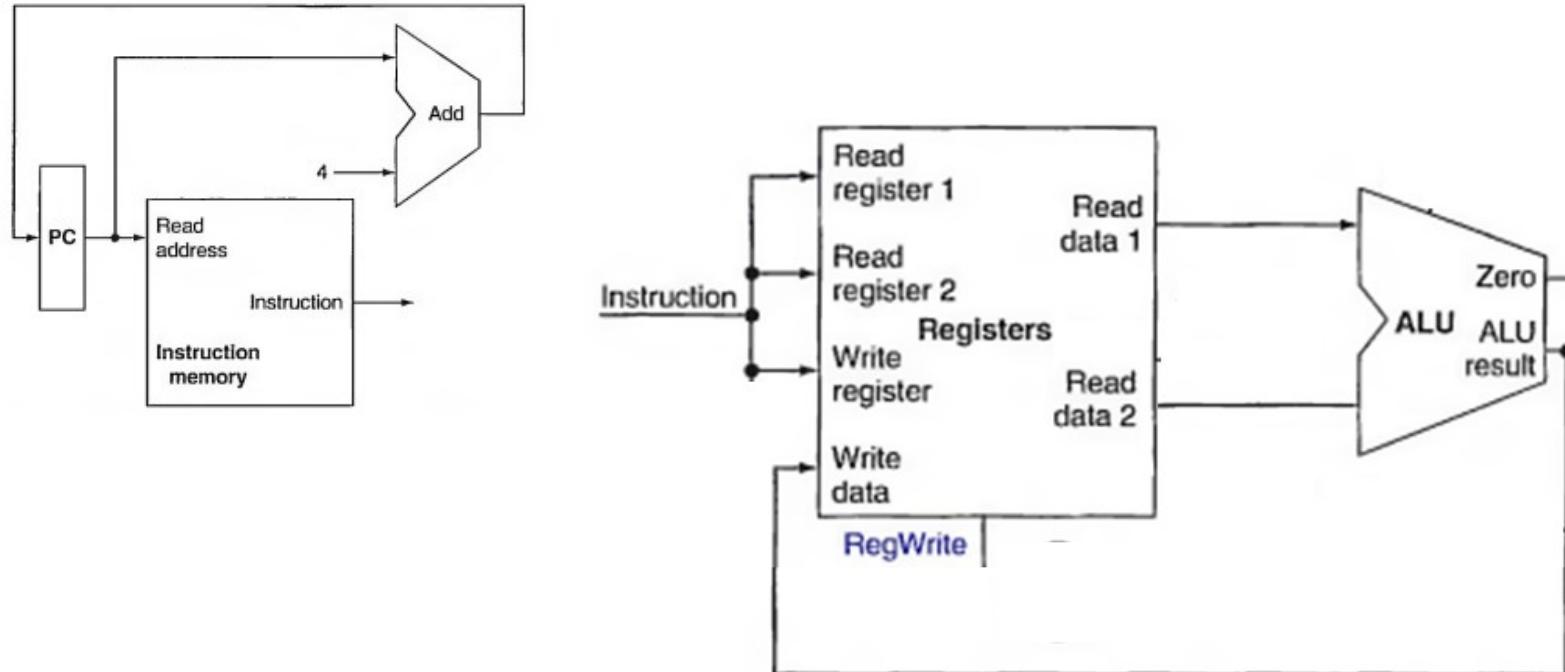
Steps - Data path for R-Type

2. Fetch instruction from Instruction memory
3. Decode Instruction



Steps - Data path for R-Type

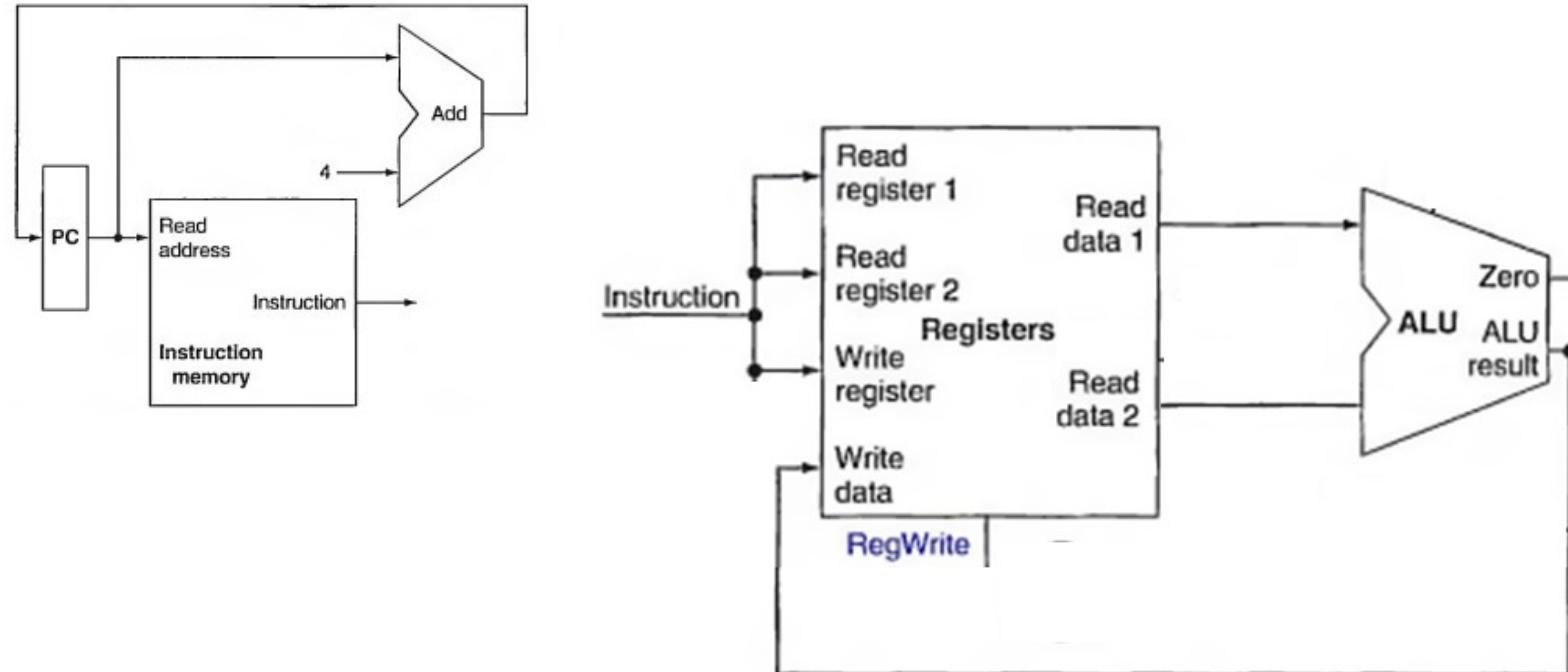
4. Pass rs, rt and rd into read register and write register arguments



The datapath for the memory instructions and the R-type instructions.

Steps - Data path for R-Type

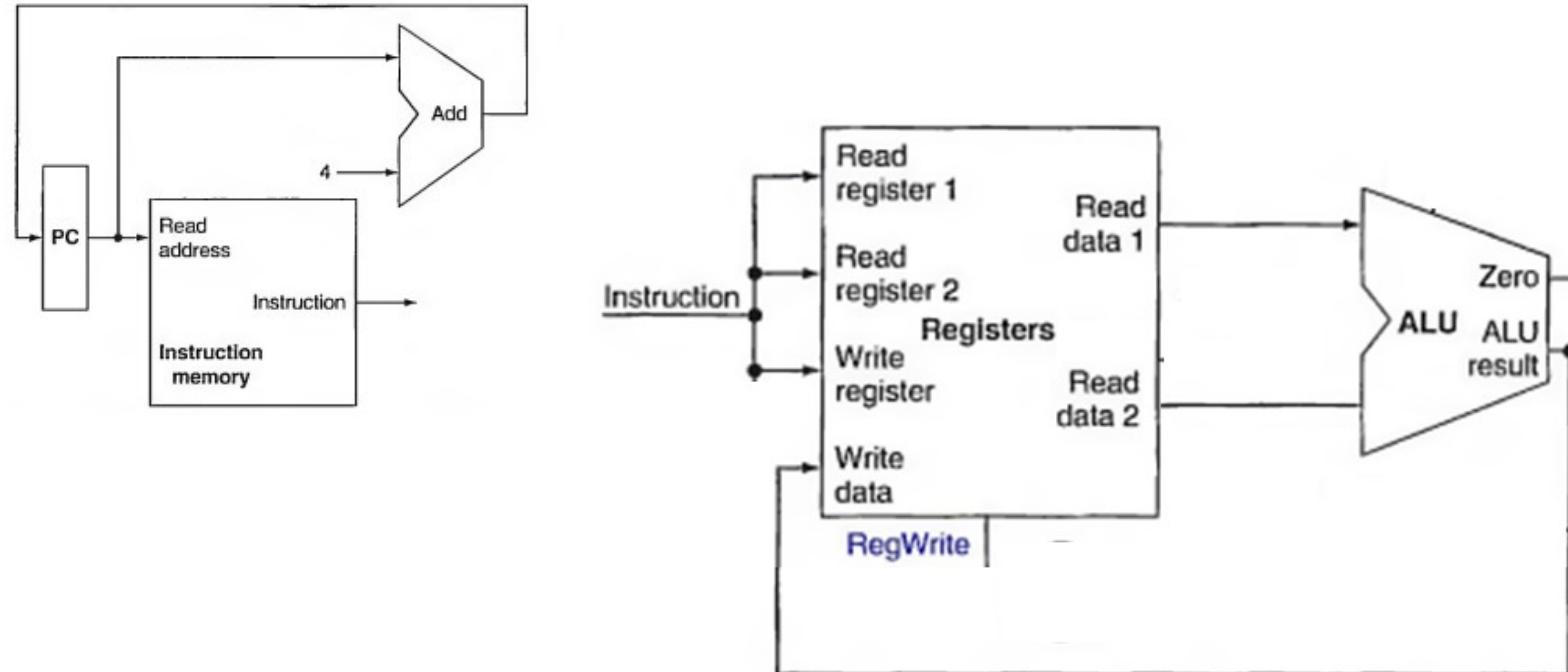
5. Retrieve Data from read register 1, read register 2(rs,rt)



The datapath for the memory instructions and the R-type instructions.

Steps - Data path for R-Type

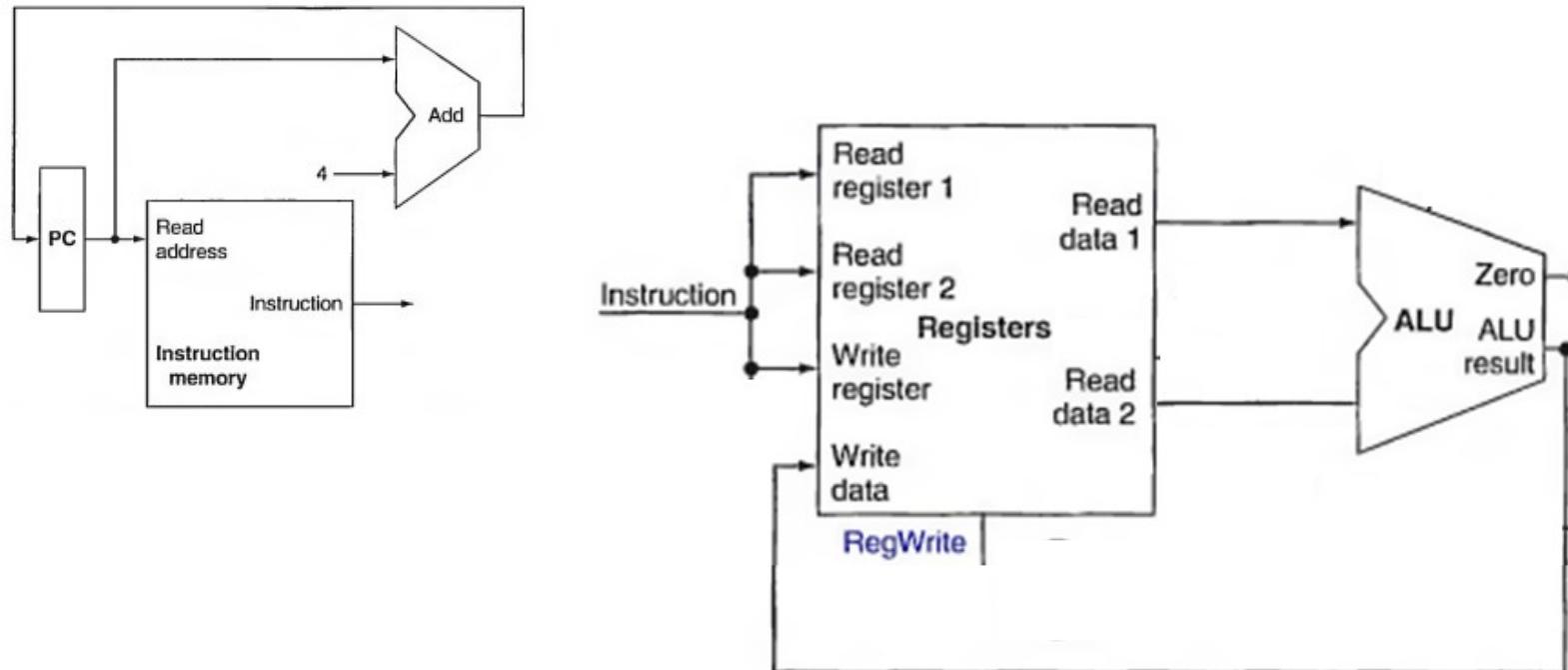
6. Pass contents of rs and rt into the ALU as operands to be performed.



The datapath for the memory instructions and the R-type instructions.

Steps – Data path for R-Type

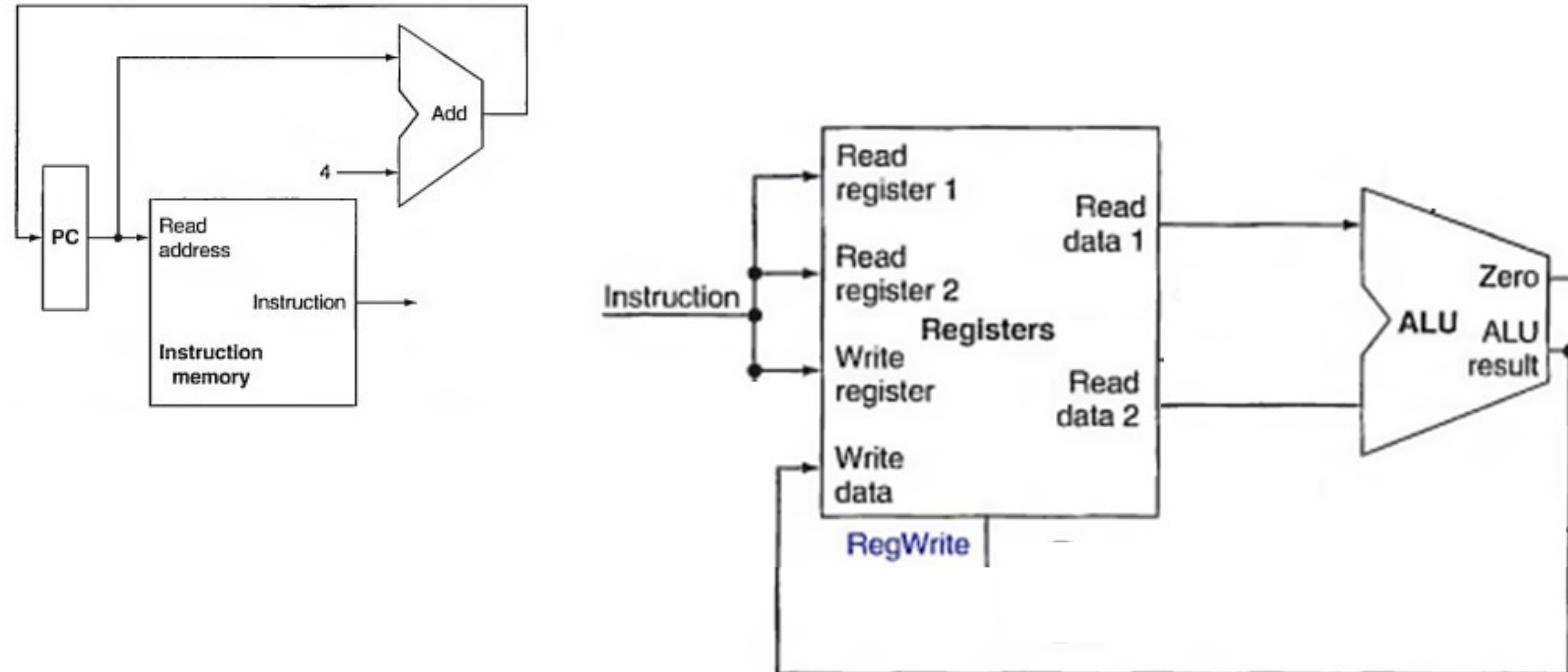
7. Retrieve result of operation performed by ALU and pass back as the write data argument of the register field (RegWrite enabled)



The datapath for the memory instructions and the R-type instructions.

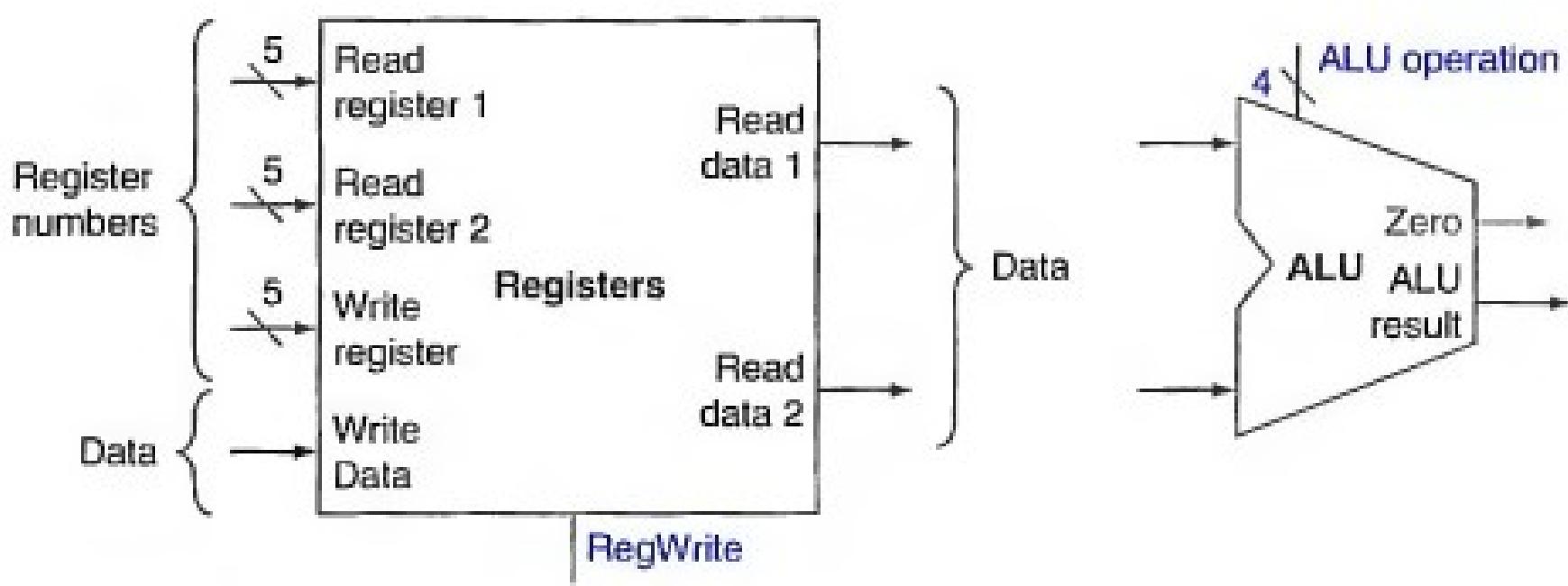
Steps - Data path for R-Type

8. Add 4 bytes to the PC value to get the next instruction address



The datapath for the memory instructions and the R-type instructions.

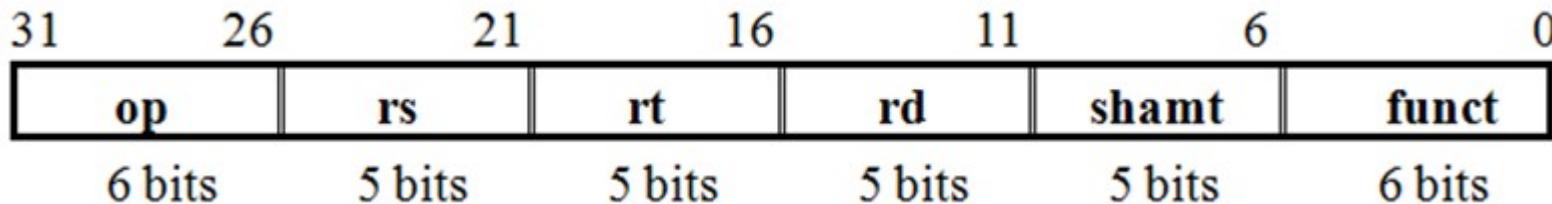
Data Path for R-type instruction



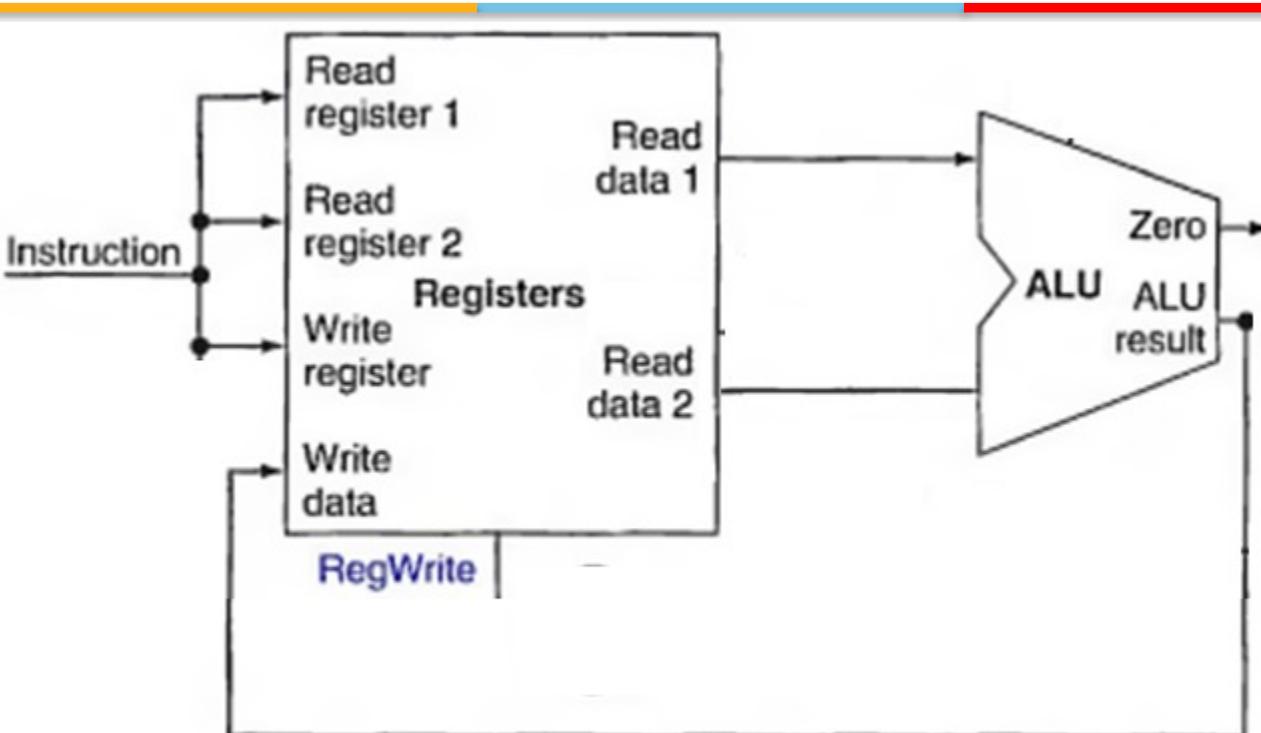
a. Registers

examples.

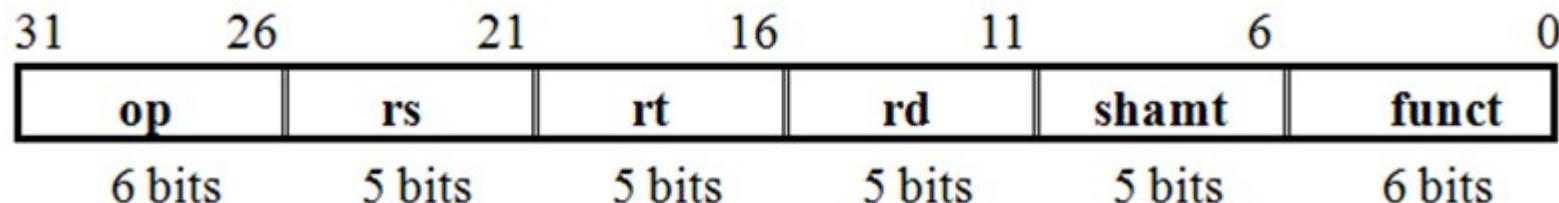
add, sub, and, or, slt \rightarrow rs, rt, rd



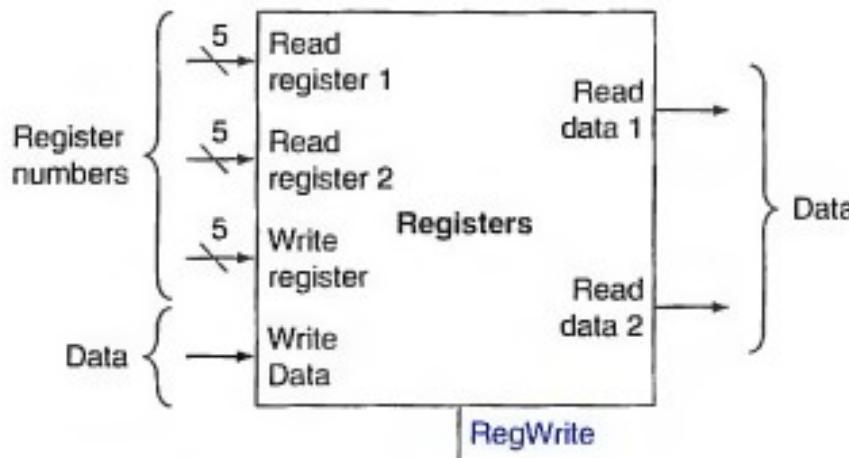
Data Path for R-type instruction...



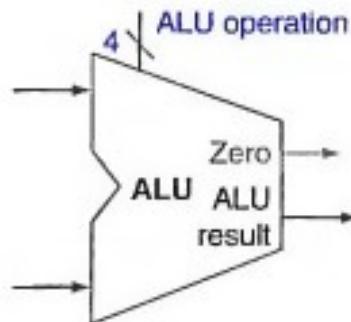
The datapath for the memory instructions and the R-type instructions.



Data Path for lw and sw instructions



a. Registers

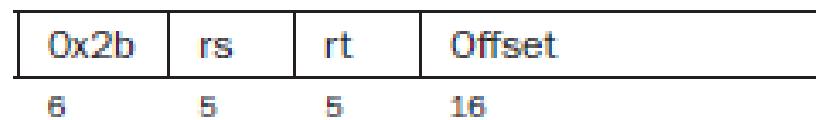


b. ALU

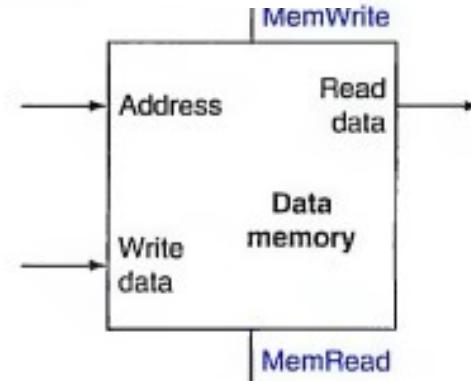
lw \$s1, 100(\$s2)



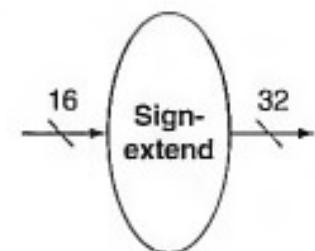
sw \$s1, 100(\$s2)



Instruction Summary



a. Data memory unit



b. Sign extension unit

Data Path for lw instructions

lw – Load word from memory

\$s2 – Source register

\$s1 – Destination register

100 – offset, sign extended

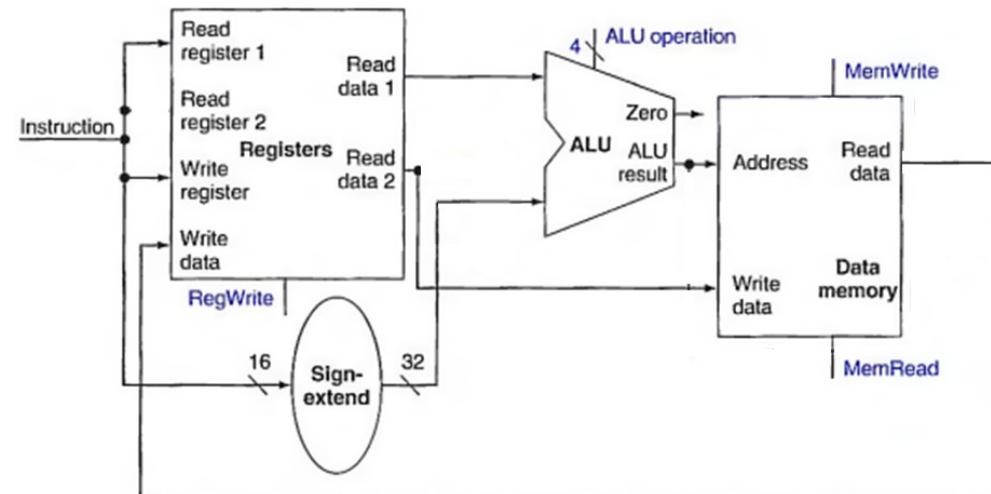
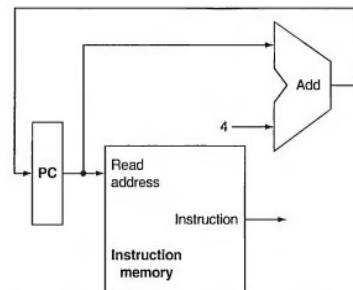
Sign extension done to fill higher order bits

$\$s1 \leftarrow \text{Memory}[100 + \$s2]$

lw \$s1, 100(\$s2)

s2 s1

0x23	rs	rt	Offset
6	5	5	16



Two data path required

- Access memory
- Sign extension

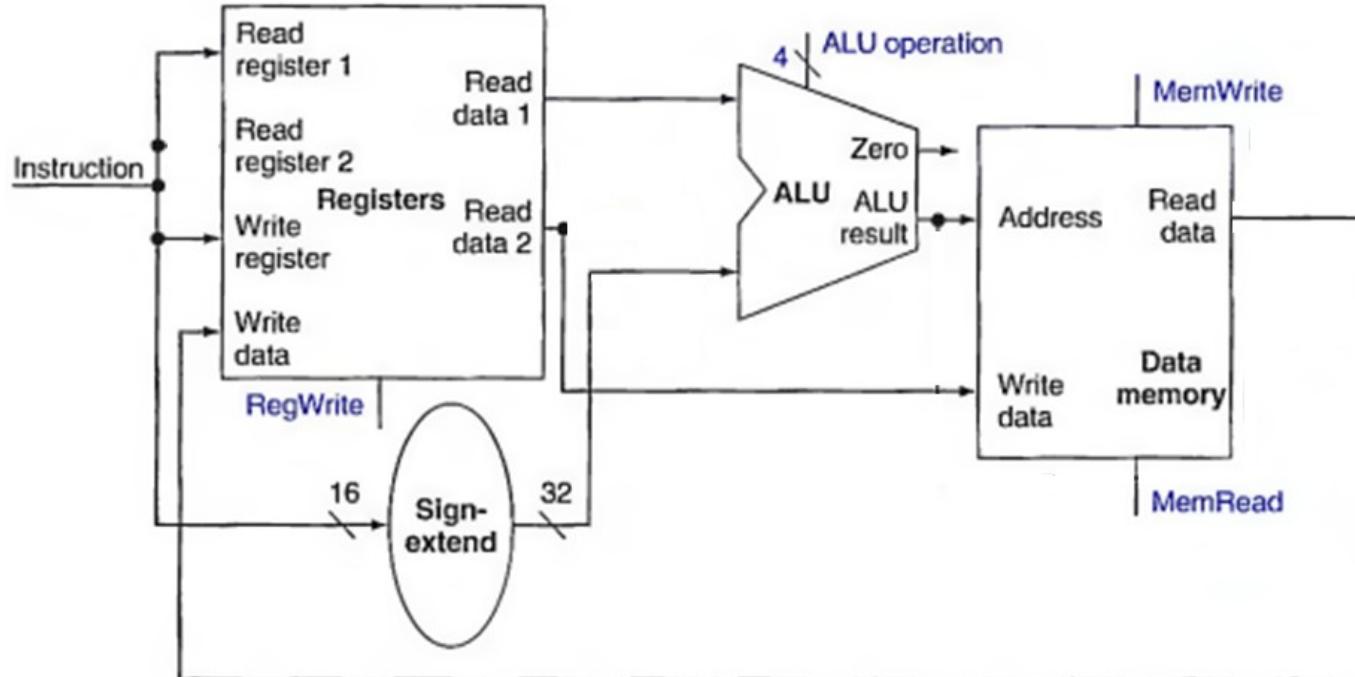
Data Path for lw instructions

There are two inputs

1. Address of memory location to access
2. Data to be written to memory

Control

1. MemWrite
2. MemRead



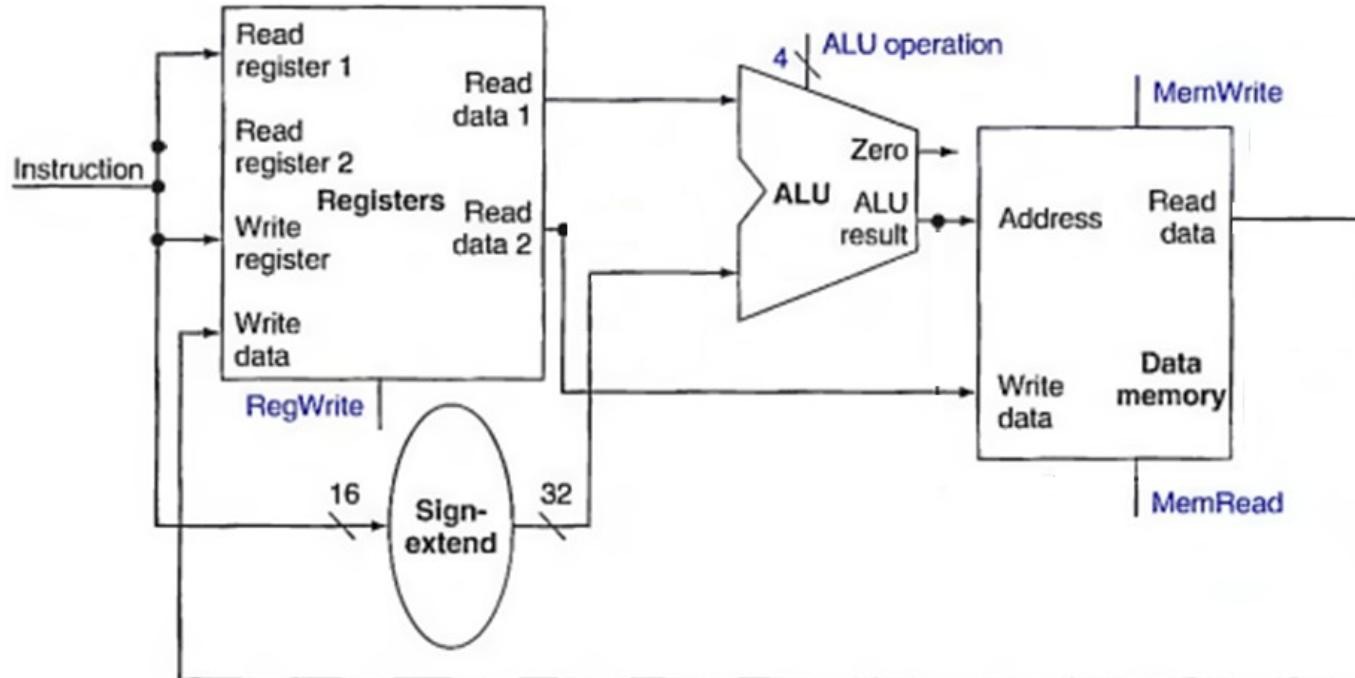
`lw $s1, 100($s2)`

0x23	rs	rt	Offset
------	----	----	--------

Data Path for lw instructions

Sign extension

We take most significant bit field of original field until we have reached the desired field width



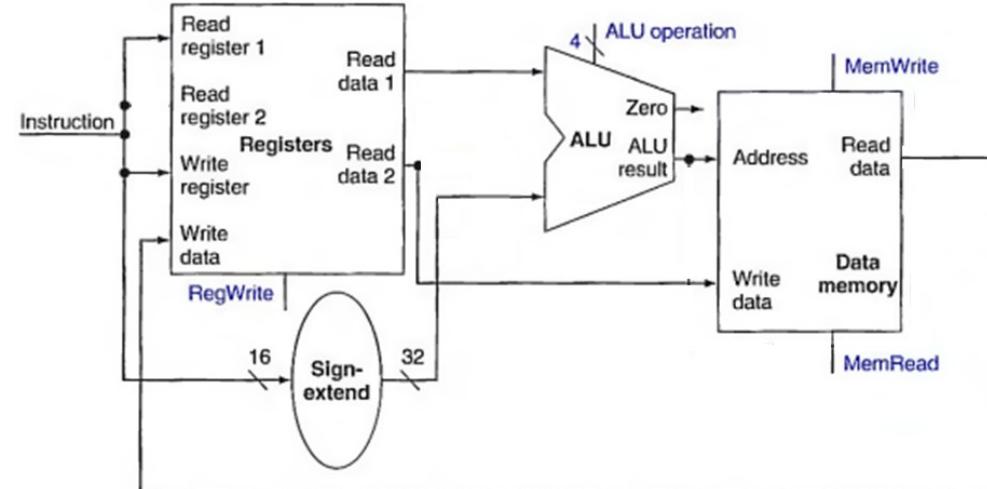
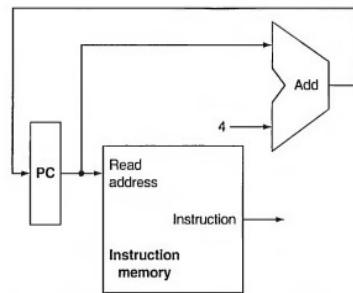
Data Path for lw instructions

$\$s1 \leftarrow Memory[100 + \$s2]$

lw \$s1, 100(\$s2)

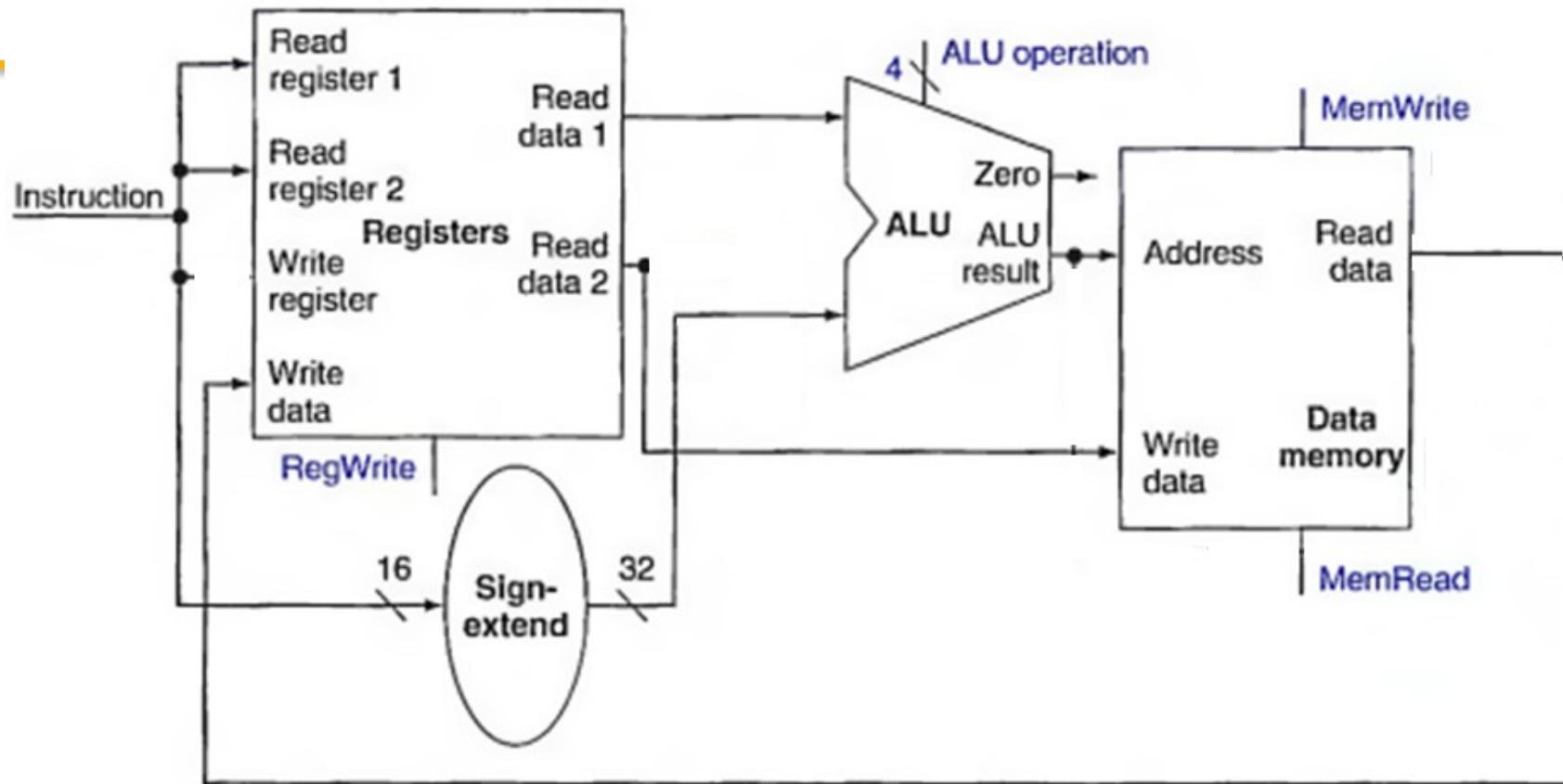
s2 s1

0x23	rs	rt	Offset
6	5	5	16



1. Extend sign
2. Obtain address after computing using ALU
3. Provide address
4. Read Data from memory
5. Load that into register
6. PC operations like before

Data Path for sw instructions



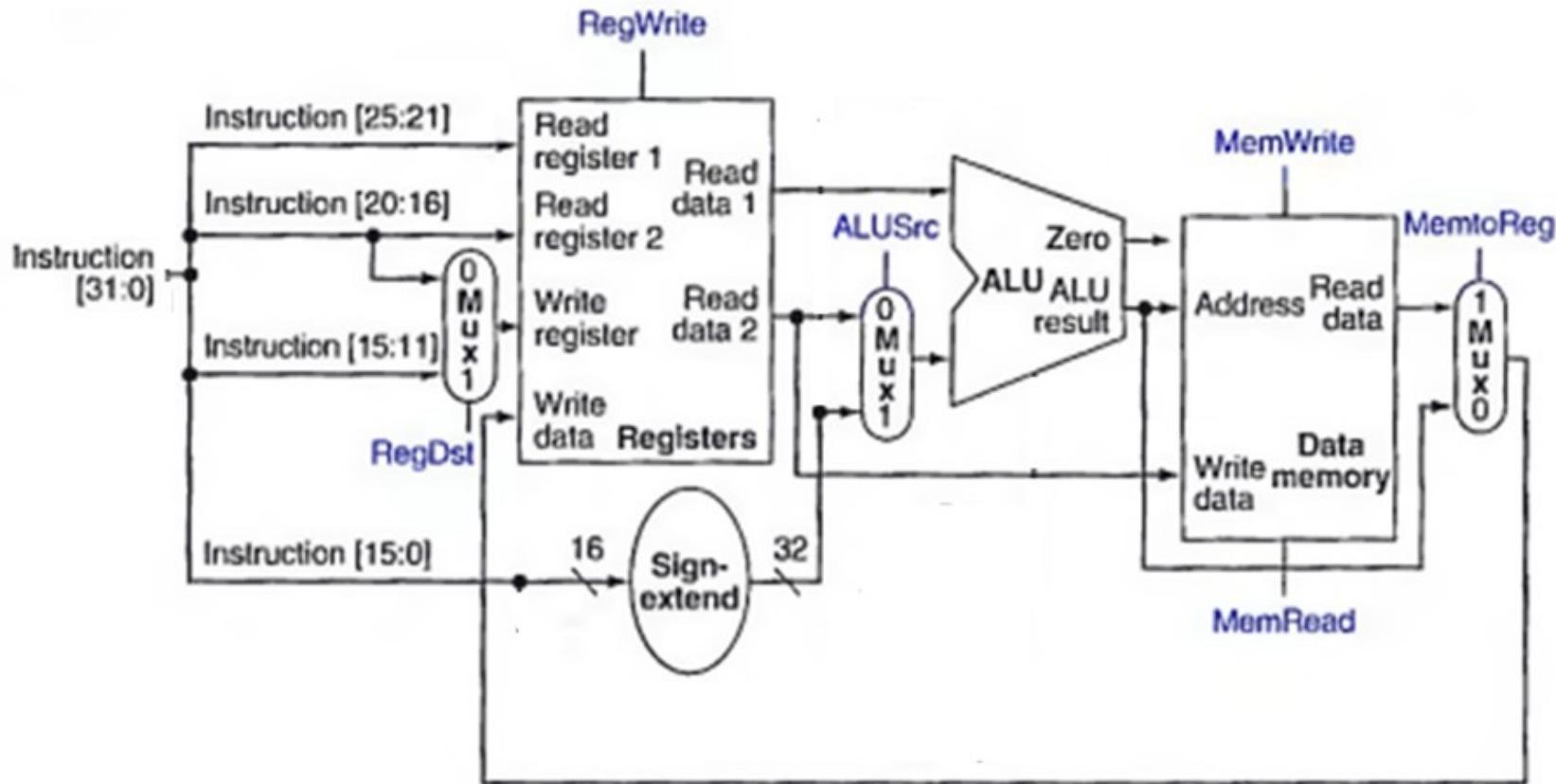
sw \$s1, 100(\$s2)

Memory[100 +\$s2] <- \$s1

1. Extend sign
 2. Obtain address after adding in ALU
 3. Provide address
 4. Store data from register to memory in address from step 3
 5. Program counter operations as before

0x2b	rs	rt	Offset
------	----	----	--------

Combined Datapath for R-Type and I-Type instructions



The datapath with all necessary multiplexors and all control lines identified.

R-Type

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

I-Type

0x2b	rs	rt	Offset
6	5	5	16

BEQ instruction

beq : branch if equal

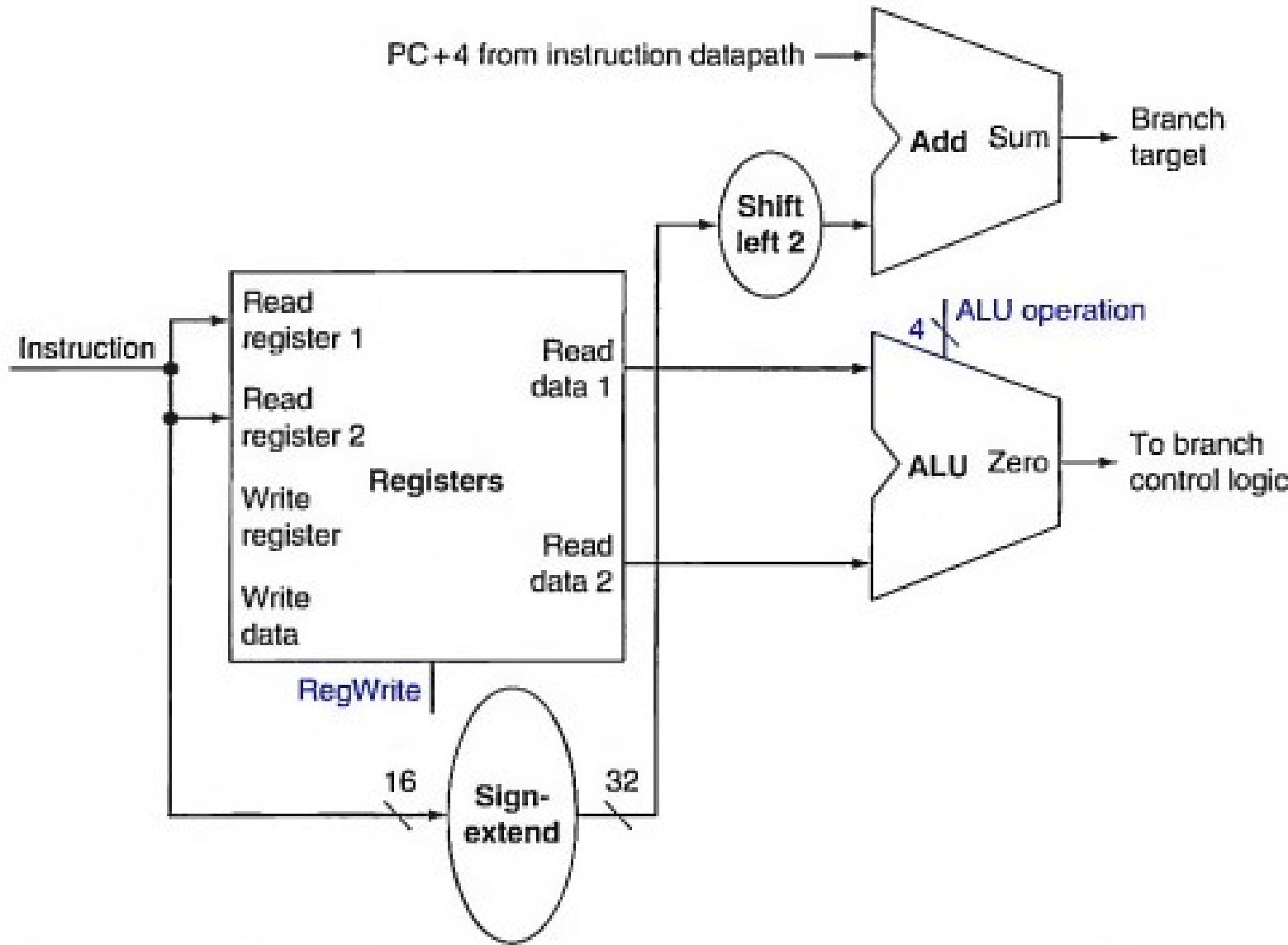
- format : beq \$s1, \$s2, label
- If $\$s1 = \$s2$ then branch to address specified as Label/offset

4	rs	rt	Offset
6	5	5	16

beq instruction

- beq \$s1, \$s2, offset
- Two important points to be noted
 - The instruction set architecture defines that the **base for the branch address** calculation is the address of the instruction following the branch
 - the architecture also states that the offset field is **shifted left 2 bits**; offset specified in immediate is **words** and not **bytes**; shifting left by 2bits gets that result
- branch taken or branch not taken

Datapath for beq instruction



Jump instruction

j <26 bit jump offset>

the jump instruction operates by replacing the lower 28 bits of the PC with the lower 26 bits of the instruction shifted left by 2 bits

Creating a single cycle datapath

Simplest datapath executes all instructions in one clock cycle

- No element in the datapath can be used more than once
- Any element needed more than once must be duplicated

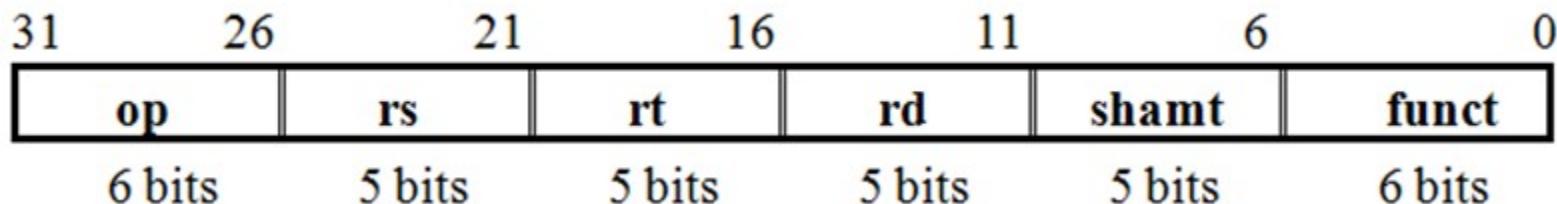
Control unit design

[Previous](#)

The ALU control

- used by load and store instructions
- used by r-type instruction : AND, OR, add, sub and slt
 - opcode : zero
 - operation performed based on “function field”

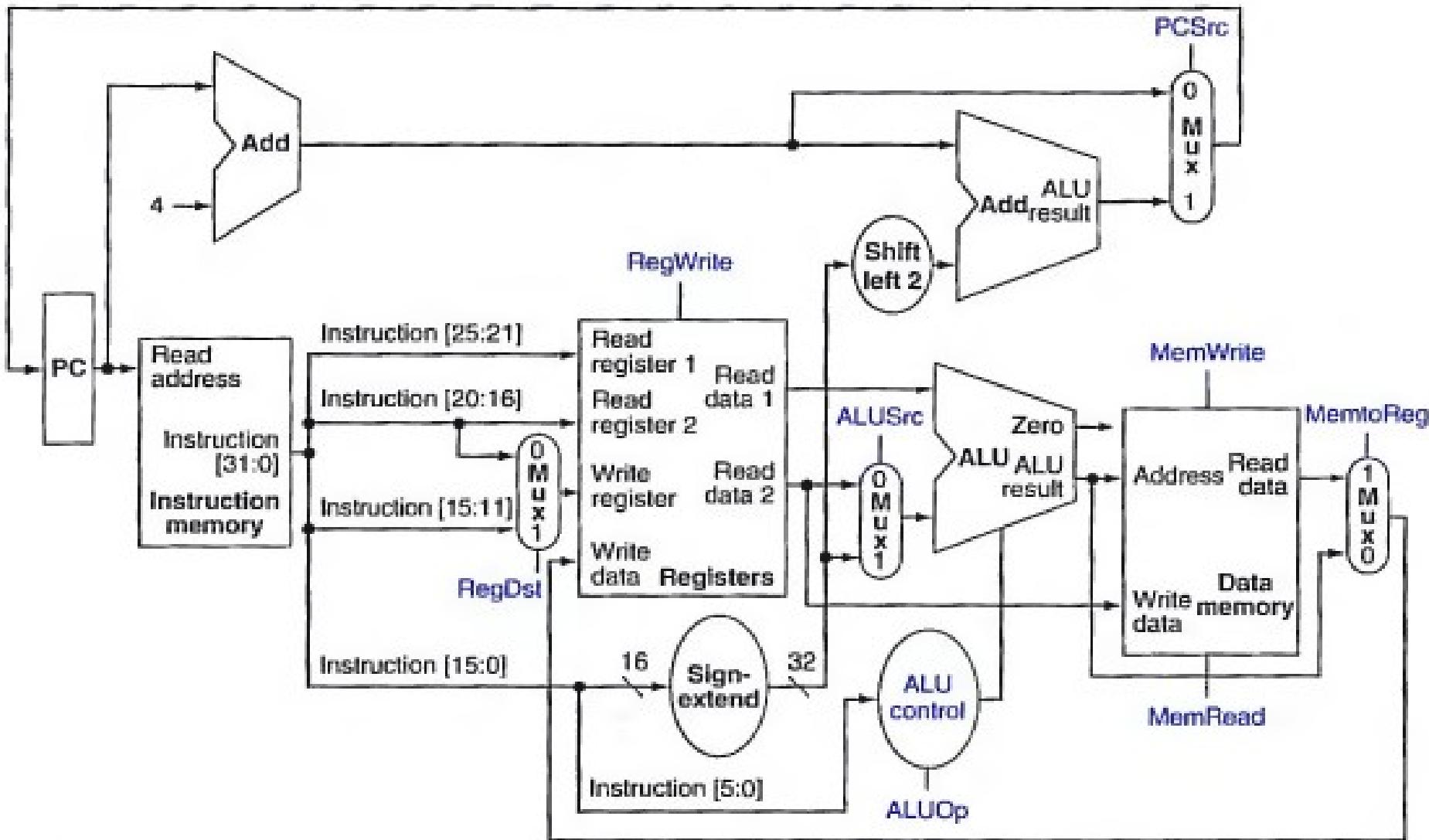
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR



Contd...

- ALU control unit generates 4 bit ALUOperation control signal based on
 - function field
 - 2 bit control field → ALUOp
 - 00 → lw and sw
 - 01 → beq (sub)
 - 10 → add, subtract, AND, OR and slt
- Main Control unit generates 7 control signals : RegDst, RegWrite, ALUSrc, PCSrc, MemRead, MemWrite and MemtoReg

Datapath



The datapath with all necessary multiplexors and all control lines identified.

Effects of 7 control signals

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

Major observations

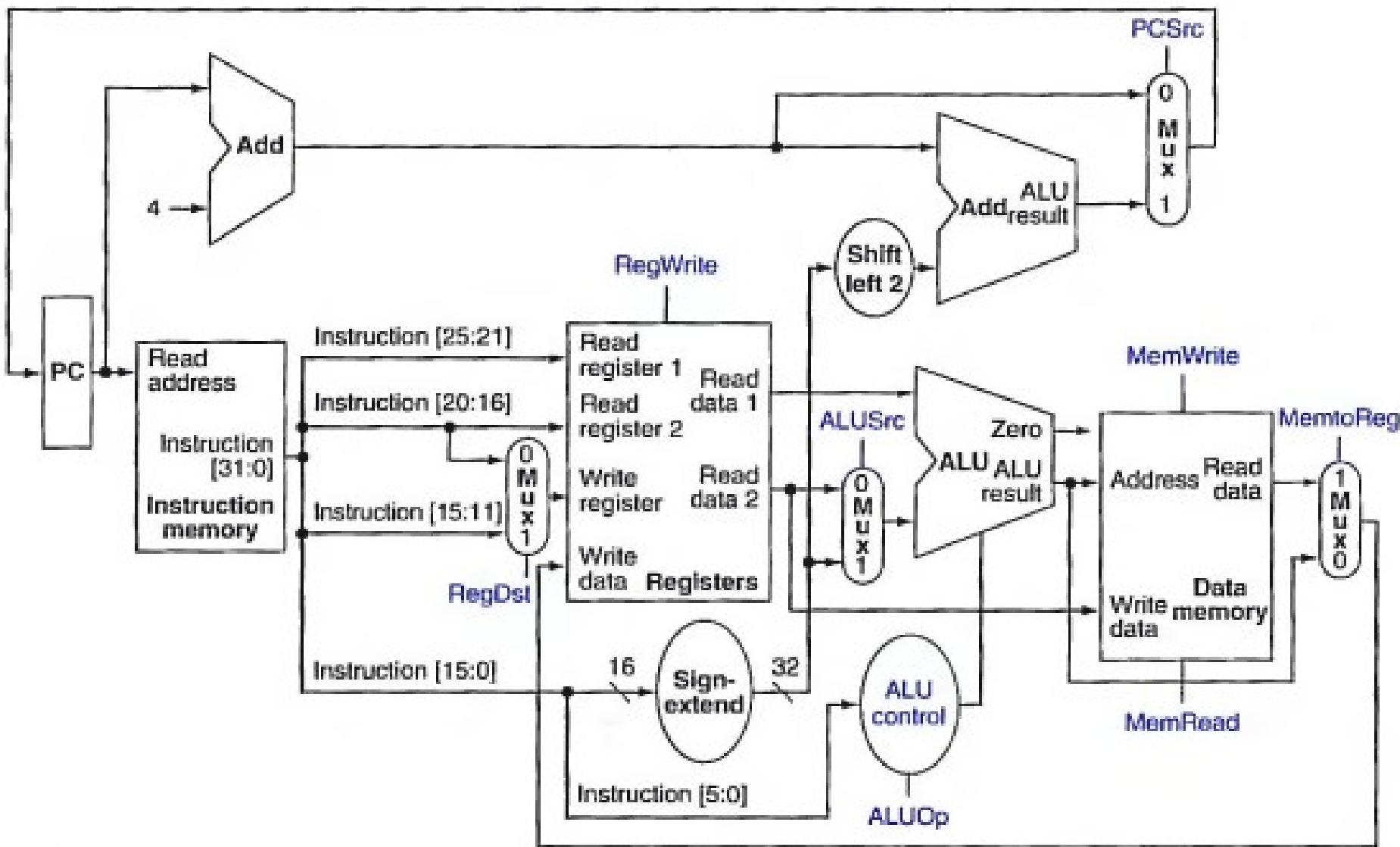
- opcode is always contained in bits 31:26
- The two registers to be read are always specified by the rs and rt fields at positions 25:21 and 20:16
 - r-type
 - branch equal
 - store
- The base register for load and store instruction is always in bit positions 25:21 i.e. rs
- The 16 bit offset for branch equal, load and store is always in positions 15:0
- The destination register is in one of two places.
 - load instruction \rightarrow rt 20:16
 - r type instruction \rightarrow rd 15:11

Execution steps: add \$s1, \$s2, \$s3

1. The instruction is fetched from _____, and the PC is incremented to _____
2. Two registers _____ and _____ are read from the register file
3. The ALU operates on the data read from the register file, based on the _____ code
4. The result from the ALU is written into the _____ register in register file

1. IR, 4
2. \$s2, \$s3
3. Opcode
4. \$s1

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add \$s1,\$s2,\$s3

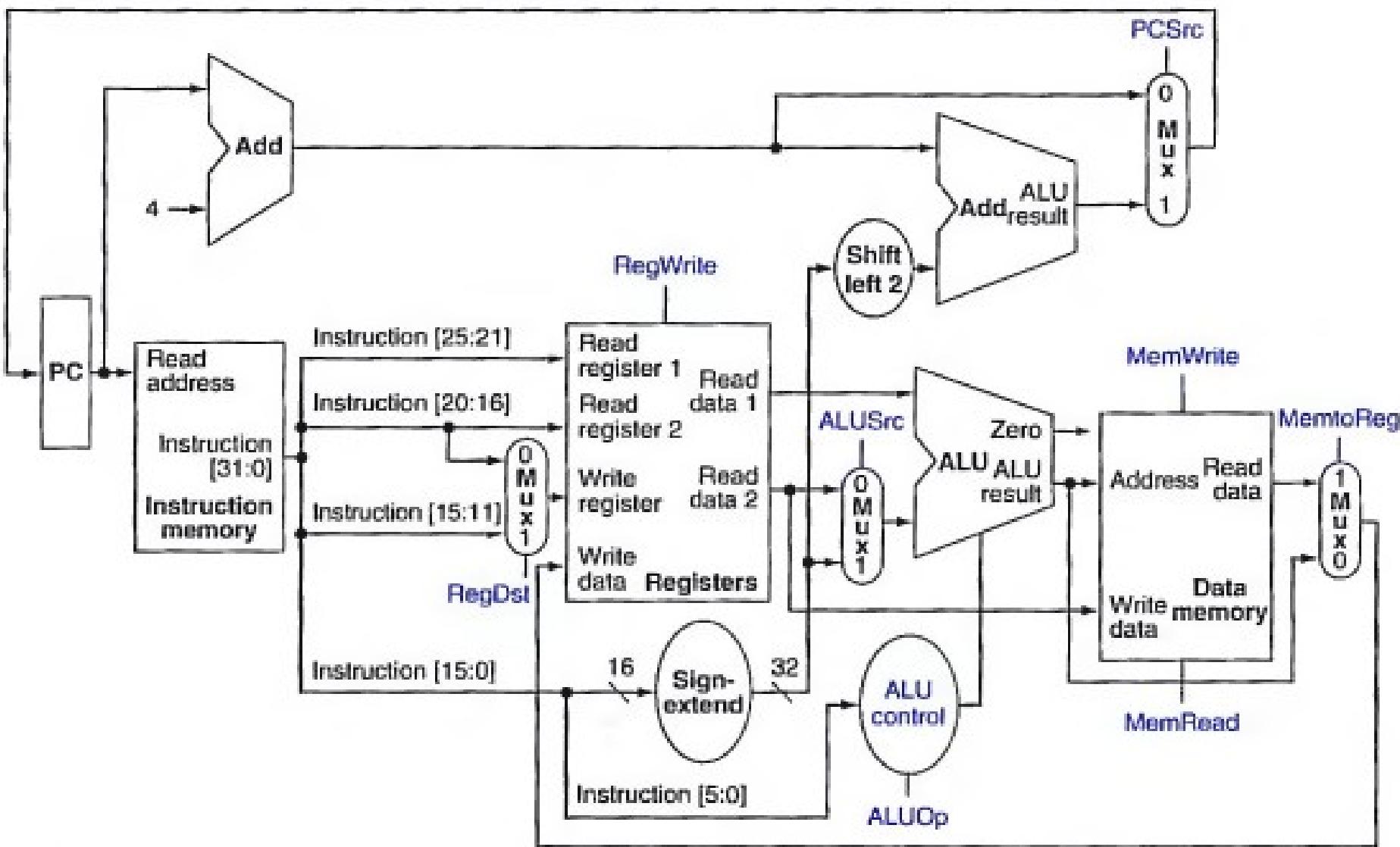


The datapath with all necessary multiplexors and all control lines identified.

Execution steps: lw \$s1, 100(\$s2)

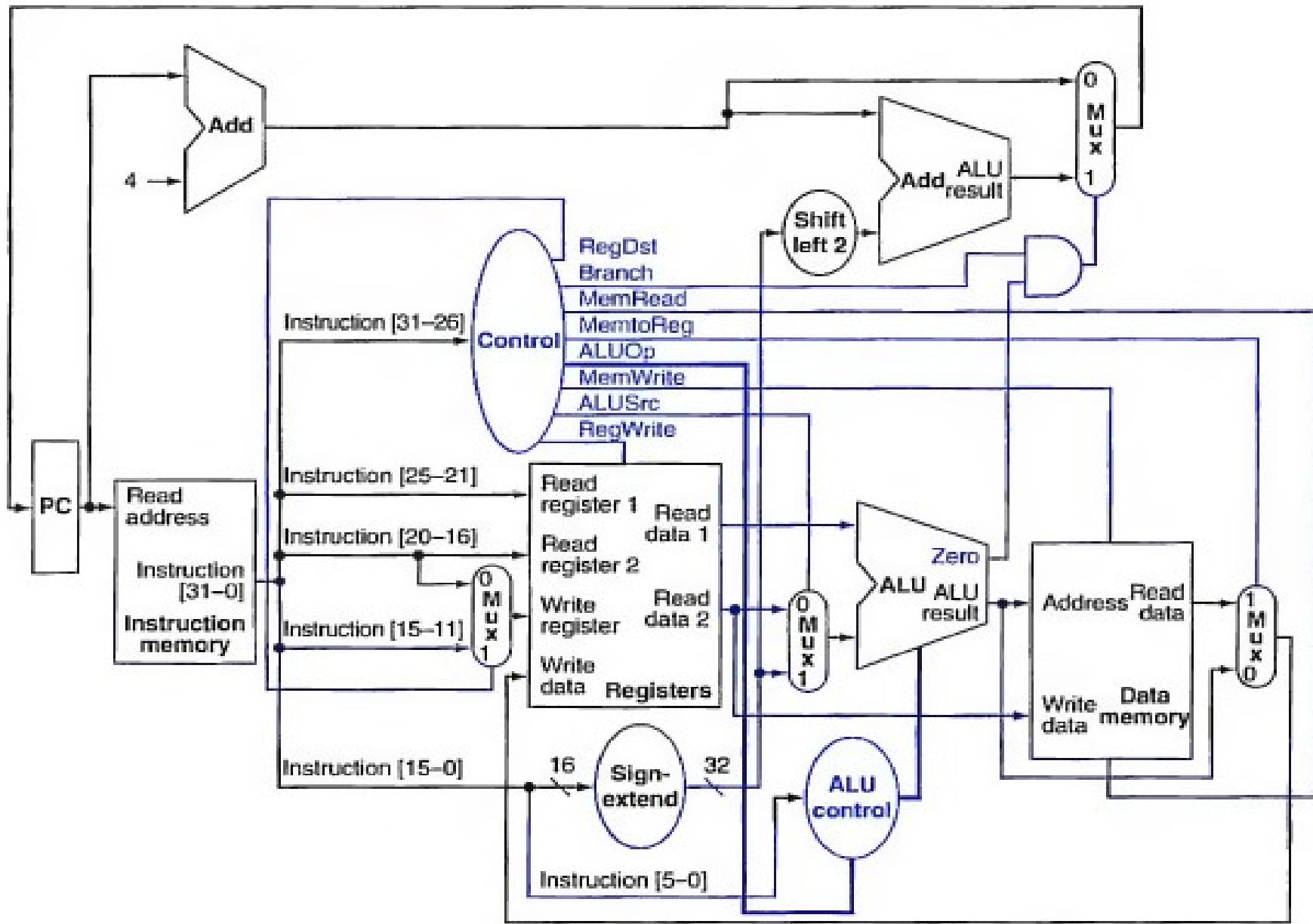
1. The instruction is fetched, and the PC is incremented
2. A register _____ value is read from the register file
3. The ALU computes the sum of the value read from the register file and the sign extended lower 16 bits of the instruction
4. the sum from the ALU is used as the address for the data memory
5. The data from the memory unit is written into _____ in register file
 1. Offset
 2. \$s1

Name	Format	Example				Comments
lw	I	35	18	17	100	lw \$s1, 100(\$s2)



The datapath with all necessary multiplexors and all control lines identified.

Complete Data Path





BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus



COMPUTING AND DESIGN SESSION 6

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Thangakumar J
WILP & Department of CS & IS



BITS Pilani
Pilani Campus

Operating systems – case study



Android ...



- **Android** is a *Linux-based platform* for *mobile devices* ...
 - *Operating System*
 - *Middleware*
 - *Applications*
 - *Software Development Kit (SDK)*
- Which kind of **mobile devices** ... (examples)



SMARTPHONES



TABLETS



EREADERS

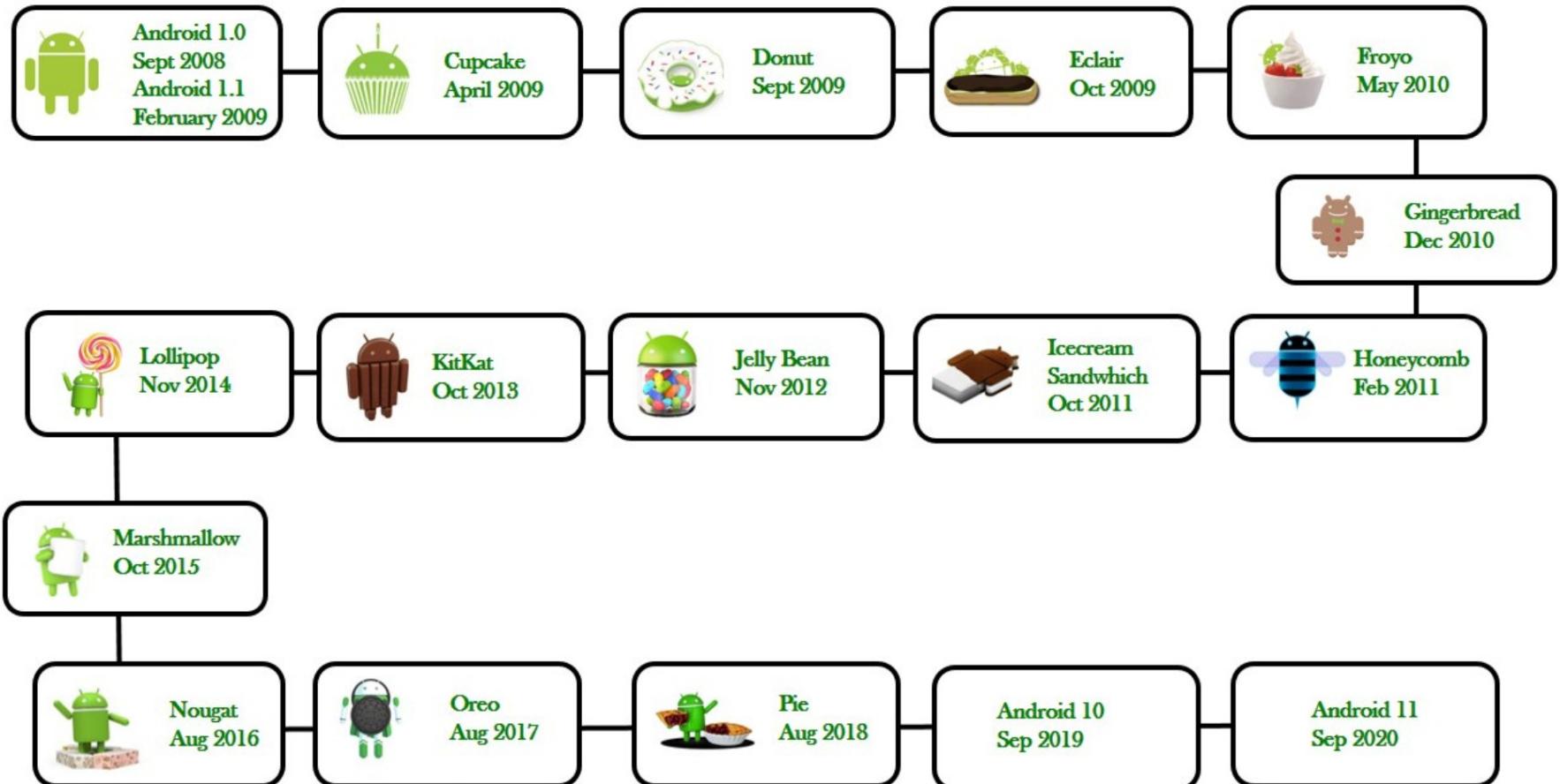


ANDROID TV



GOOGLE GLASSES



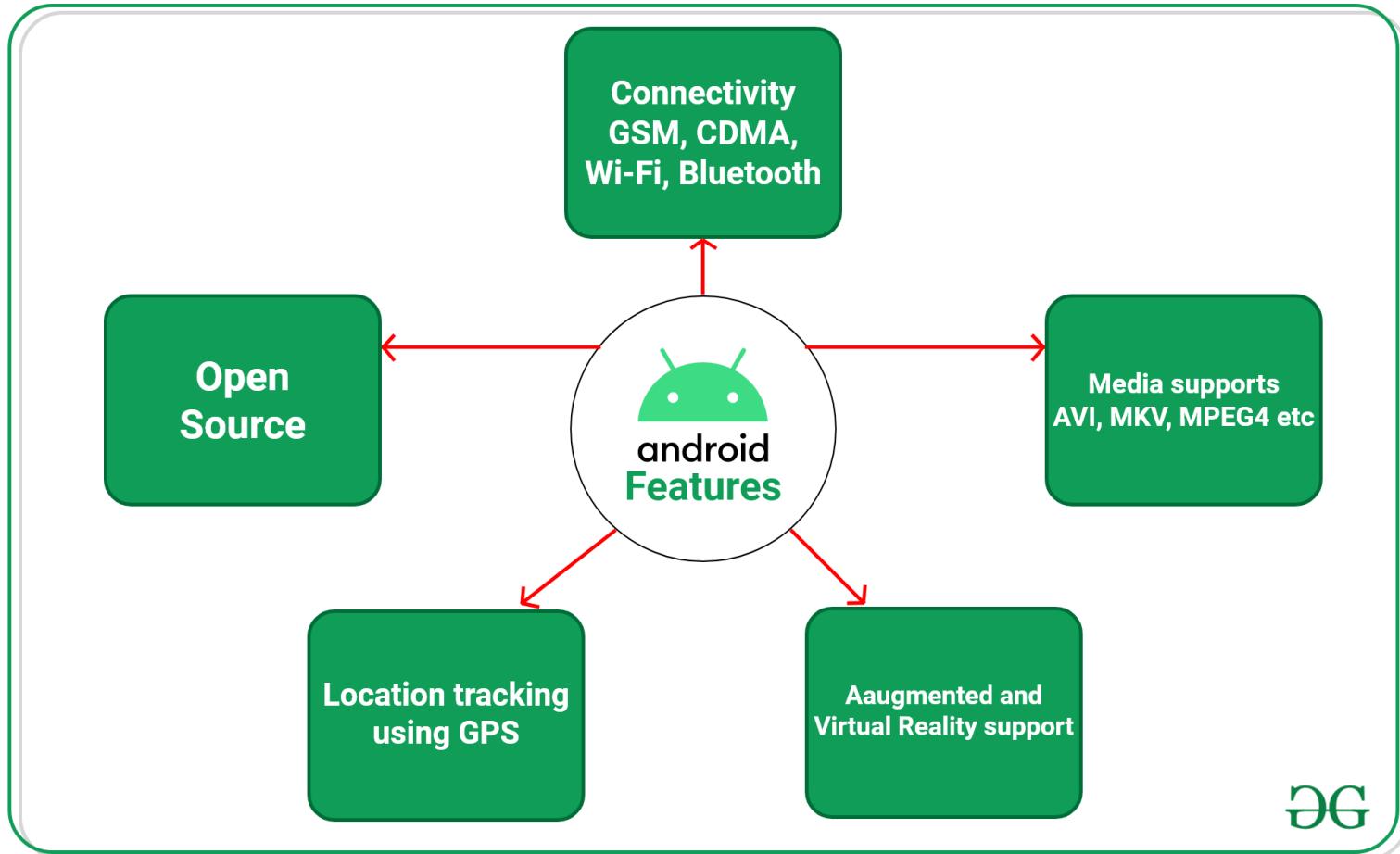


VERSIONS OF ANDROID



Major Devices that runs on Android Operating System



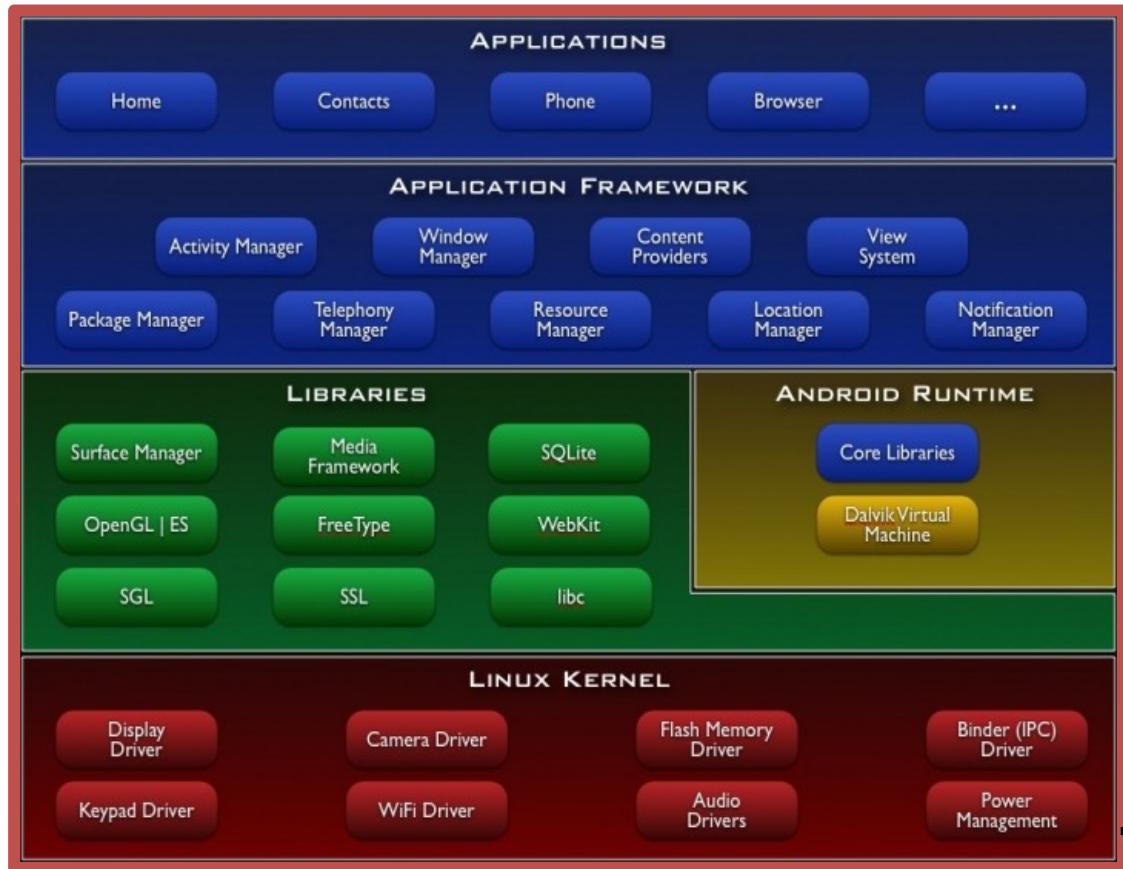


- Android Open Source Project so we can customize the OS based on our requirements.
- Android supports different types of connectivity for GSM, CDMA, Wi-Fi, Bluetooth, etc. for telephonic conversation or data transfer.
- Using wifi technology we can pair with other devices while playing games or using other applications.
- It contains multiple APIs to support location-tracking services such as GPS.
- We can manage all data storage-related activities by using the file manager.

- It contains a wide range of media supports like AVI, MKV, FLV, MPEG4, etc. to play or record a variety of audio/video.
- It also supports different image formats like JPEG, PNG, GIF, BMP, MP3, etc.
- It supports multimedia hardware control to perform playback or recording using a camera and microphone.

- Android has an integrated open-source WebKit layout-based web browser to support User Interfaces like HTML5, and CSS3.
- Android supports multi-tasking means we can run multiple applications at a time and can switch between them.
- It provides support for virtual reality or 2D/3D Graphics.

The Android Architecture



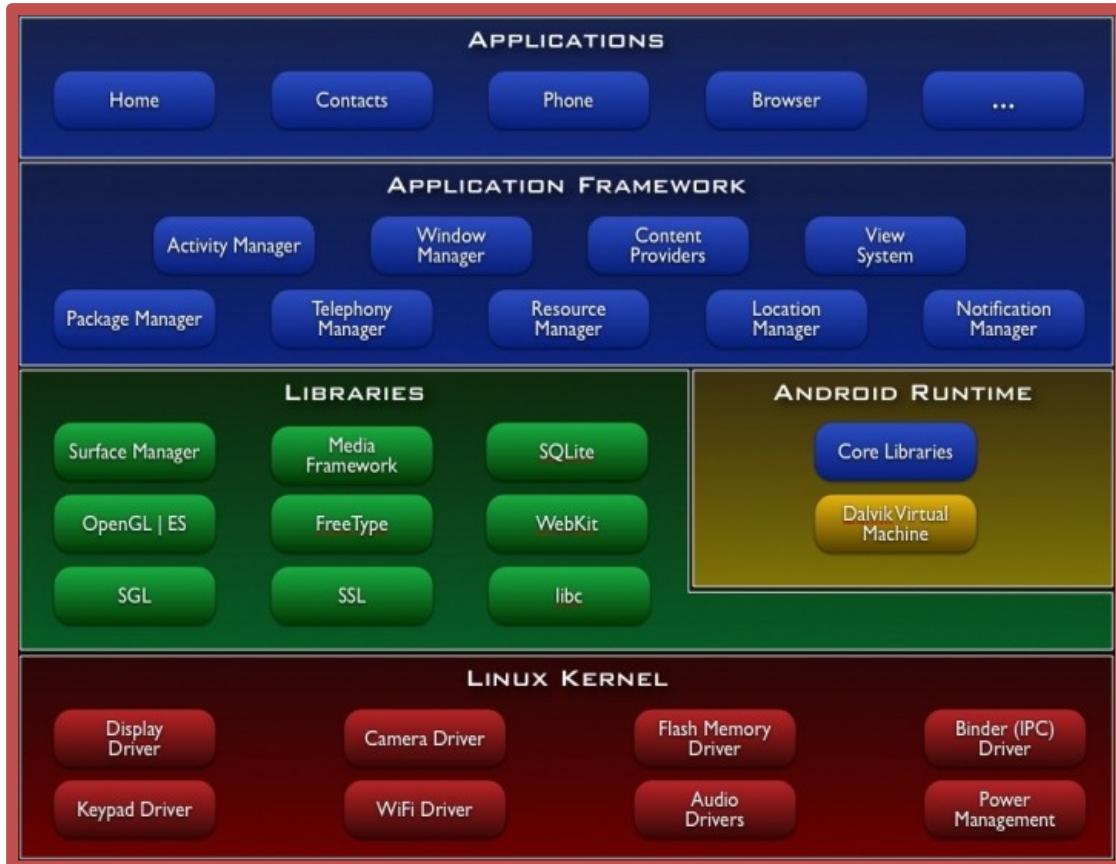
Built on top of
Linux kernel

Advantages:

- **Portability** (i.e. easy to compile on different hardware architectures)
- **Security** (e.g. secure multi-process environment)
- **Power Management**

- Memory and Process management □ File & Network I/O □ Device Drivers
- Preemptive Multitasking
- Lean, efficient, and secure
- Open Source

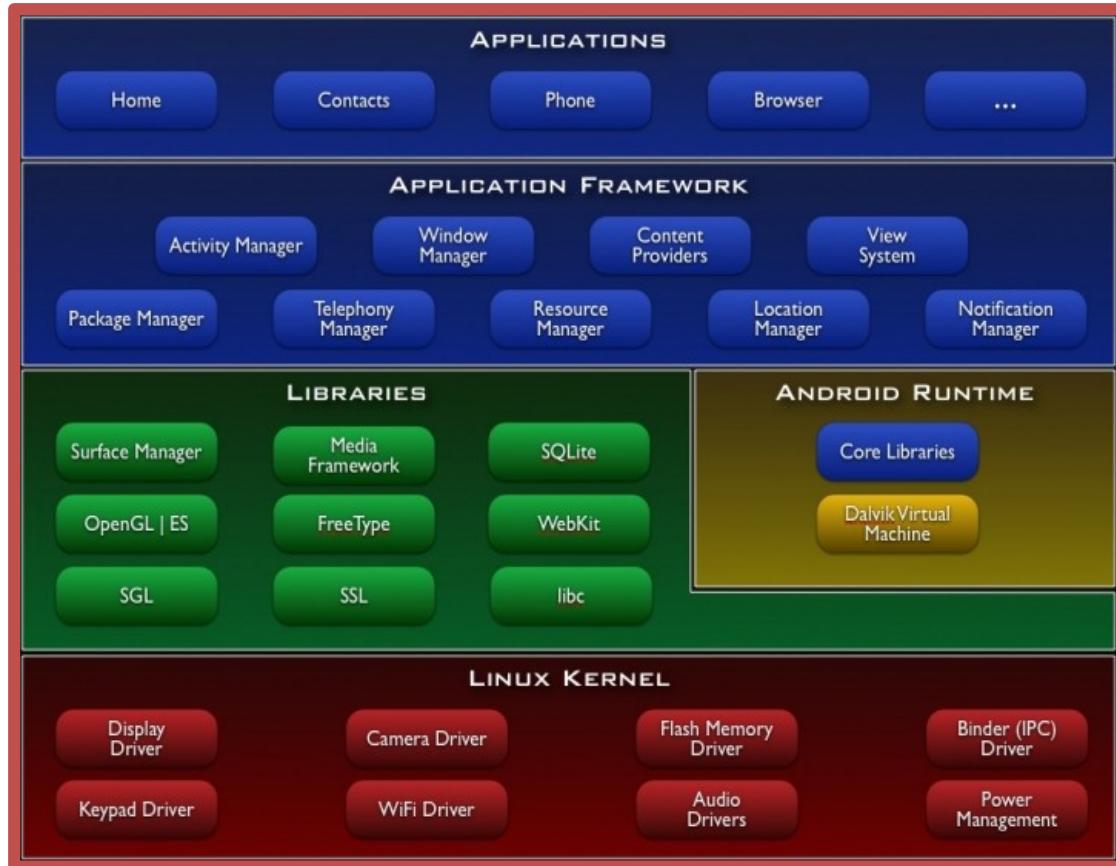
The Android Architecture



Native Libraries (C/C++ code)

- **Graphics** (Surface Manager)
- **Multimedia** (Media Framework)
- **Database DBMS** (SQLite)
- **Font Management** (FreeType)
- **WebKit**
- **C libraries** (Bionic)
-

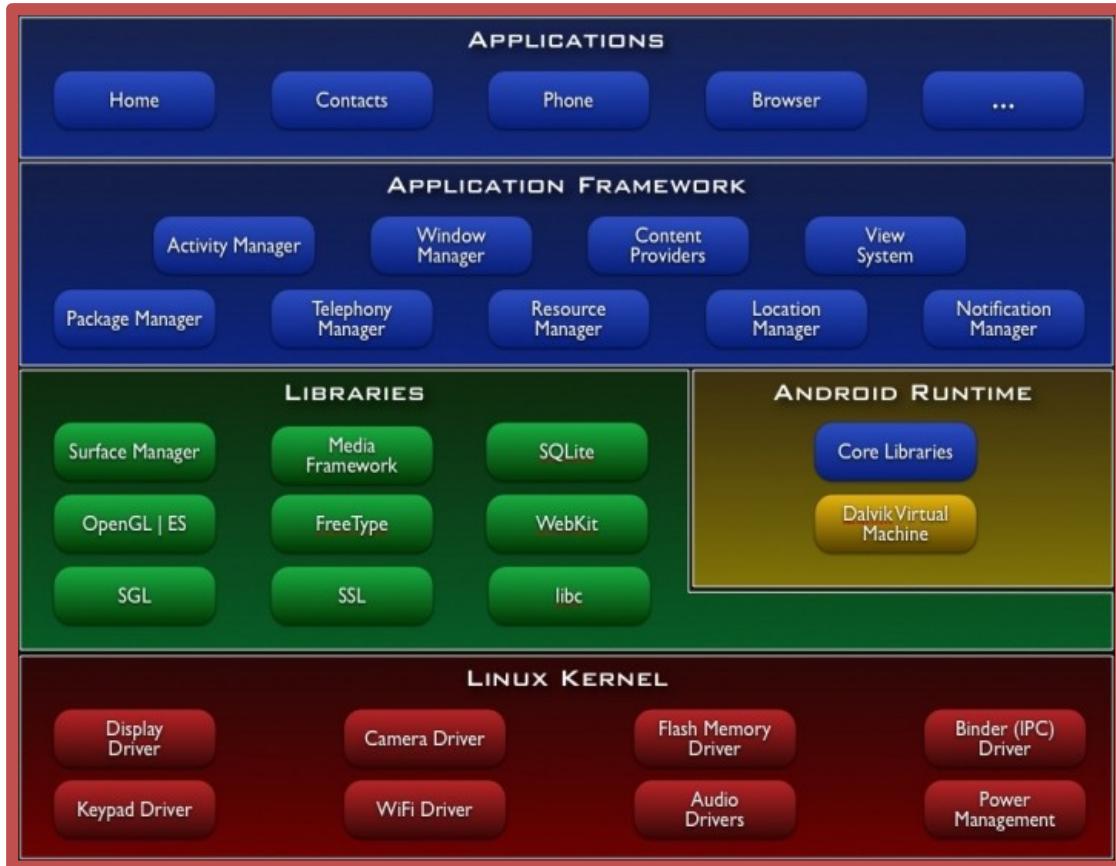
The Android Architecture



Application Libraries (Core Components of Android)

- Activity Manager
- Packet Manager
- Telephony Manager
- Location Manager
- Contents Provider
- Notification Manager
-

The Android Architecture

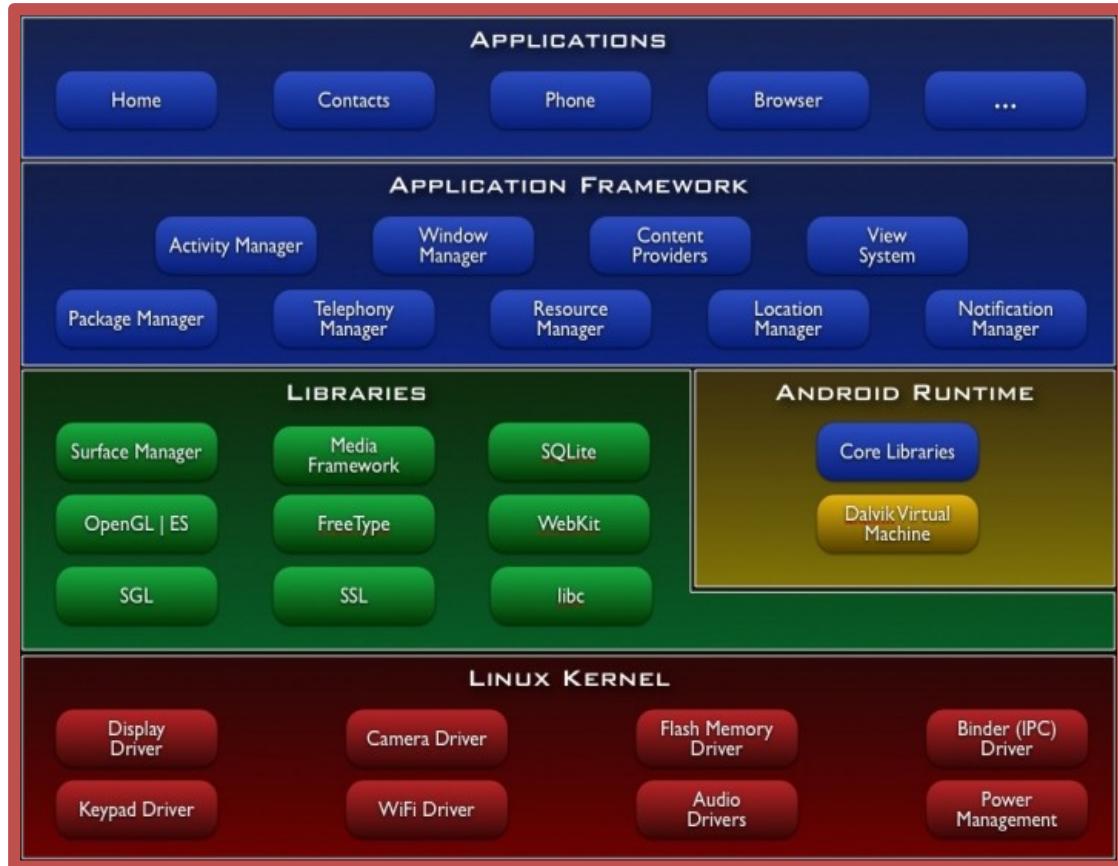


Applications

(Written in **Java** code)

- **Android Play Store**
- **Entertainment**
- **Productivity**
- **Personalization**
- **Education**
- **Geo-communication**
-

The Android Architecture



Dalvik Virtual Machine (VM)

- **Novel Java Virtual Machine implementation (not using the Oracle JVM)**
- **Open License** (Oracle JVM is not open!)
- **Optimized for memory-constrained devices**
- **Faster than Oracle JVM**
-

Android Applications Design

APPLICATION DESIGN:

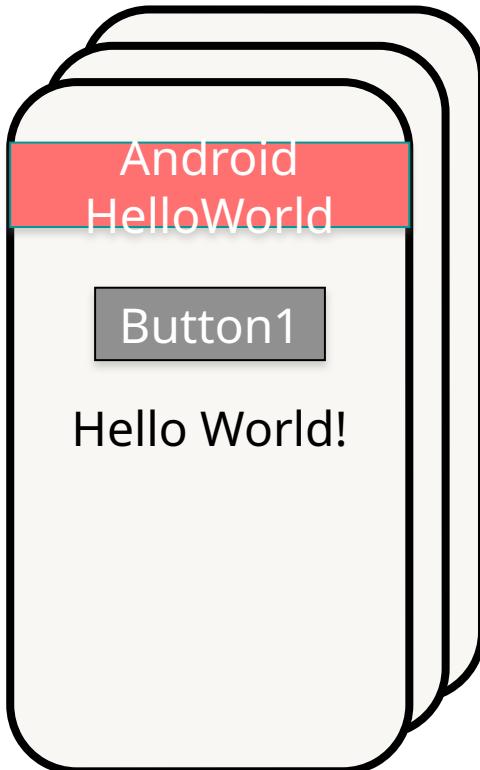
- **GUI** Definition
- **Events** Management
- Application **Data** Management
- **Background** Operations
- **User** Notifications

Android Applications Design

APPLICATION COMPONENTS

- **Activities**
- **Intents**
- **Services**
- **Content Providers**
- **Broadcast Receivers**

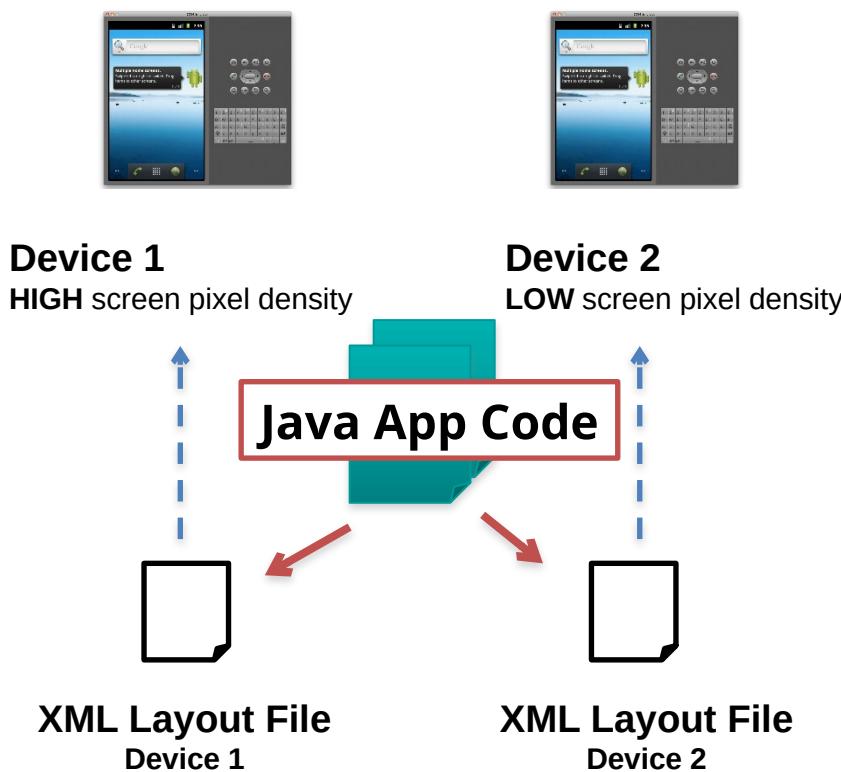
Android Components: Activities



- An **Activity** corresponds to a **single screen** of the **Application**.
- An Application can be composed of *multiple screens* (Activities).
- The **Home Activity** is shown when the user launches an application.
- Different activities can exchange information one with each other.

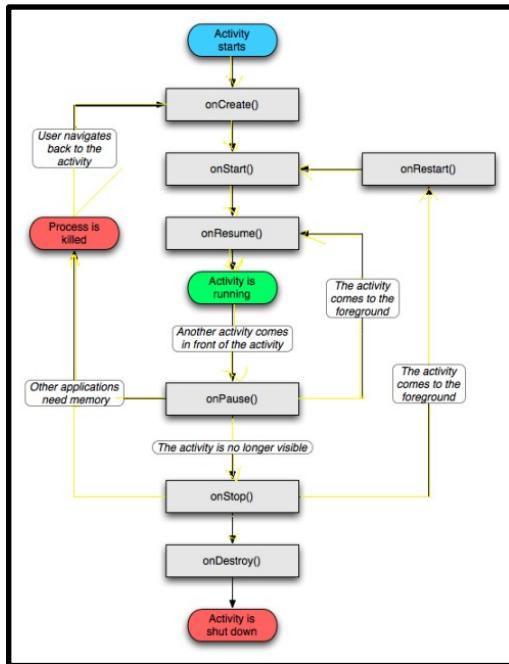
Android Components: Activities

EXAMPLE



- Build the **application layout** through XML files (like HTML)
- Define **two** different XML **layouts** for two different devices
- At **runtime**, Android detects the current device configuration and loads the appropriate resources for the application
- **No need to recompile!**
- Just add a new XML file if you need to support a new device

Android Components: Activities



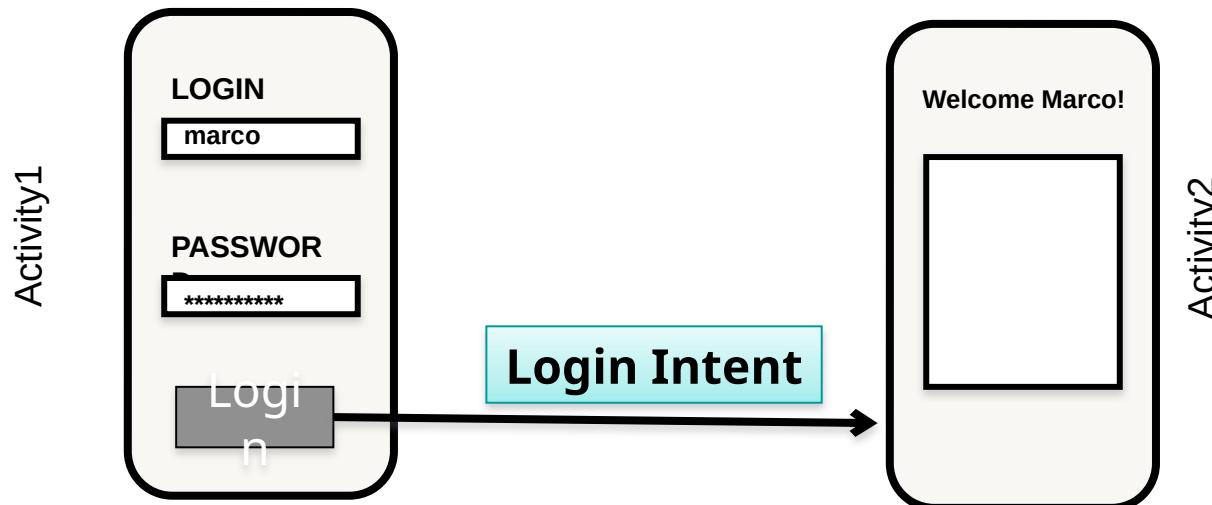
- The **Activity Manager** is responsible for creating, destroying, managing activities.
- Activities can be on different **states**: *starting, running, stopped, destroyed, paused*.
- Only one activity can be on the **running** state at a time.
- Activities are organized on a **stack**, and have an event-driven life cycle (details later ...)

Android Components: Activities

- Main difference between Android-programming and Java (Oracle)-programming:
 - **Mobile devices have constrained resource capabilities!**
- Activity lifetime depends on **users' choice** (i.e. change of visibility) as well as on **system constraints** (i.e. memory shortage).
- Developer must implement **lifecycle methods** to account for state changes of each Activity ...

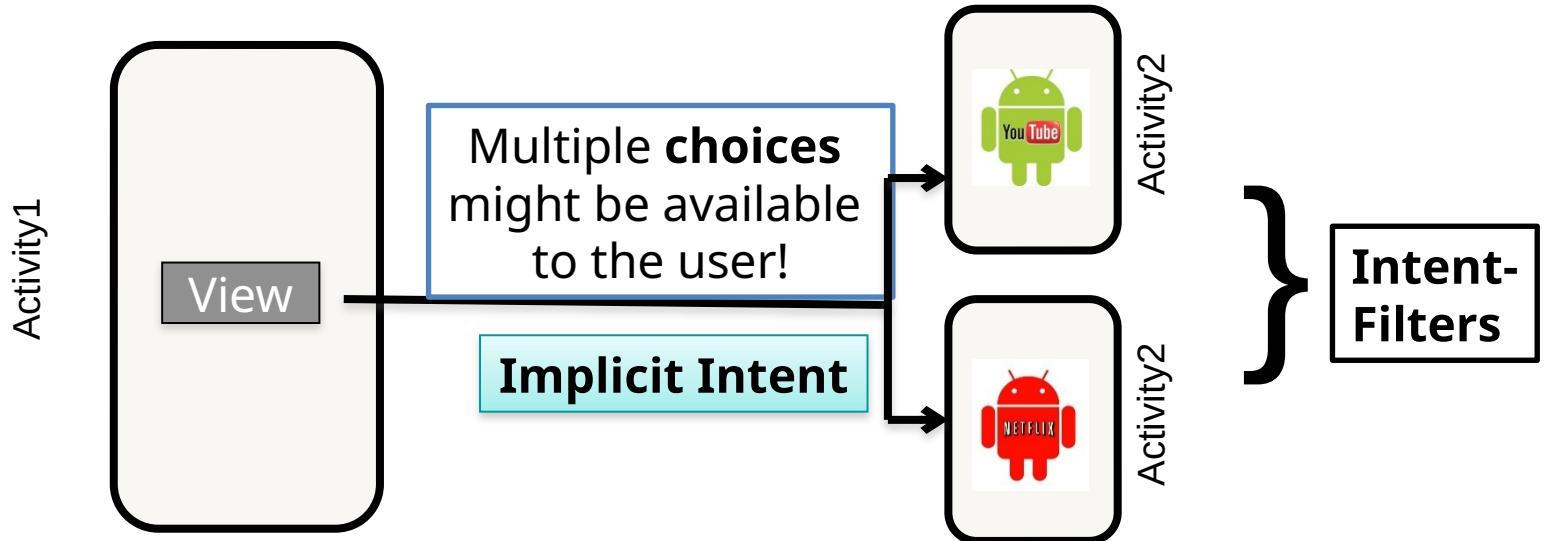
Android Components: Intents

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Explicit Intent** → The component (e.g. *Activity1*) specifies the destination of the intent (e.g. *Activity 2*).



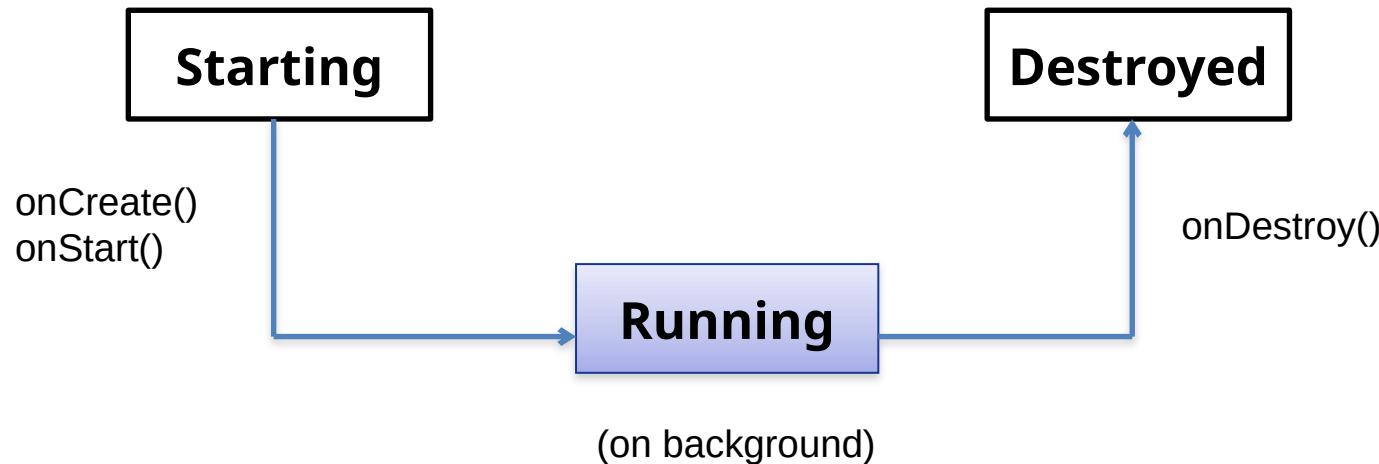
Android Components: Intents

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Implicit Intent** → The component (e.g. *Activity1*) specifies the type of the intent (e.g. “View a video”).



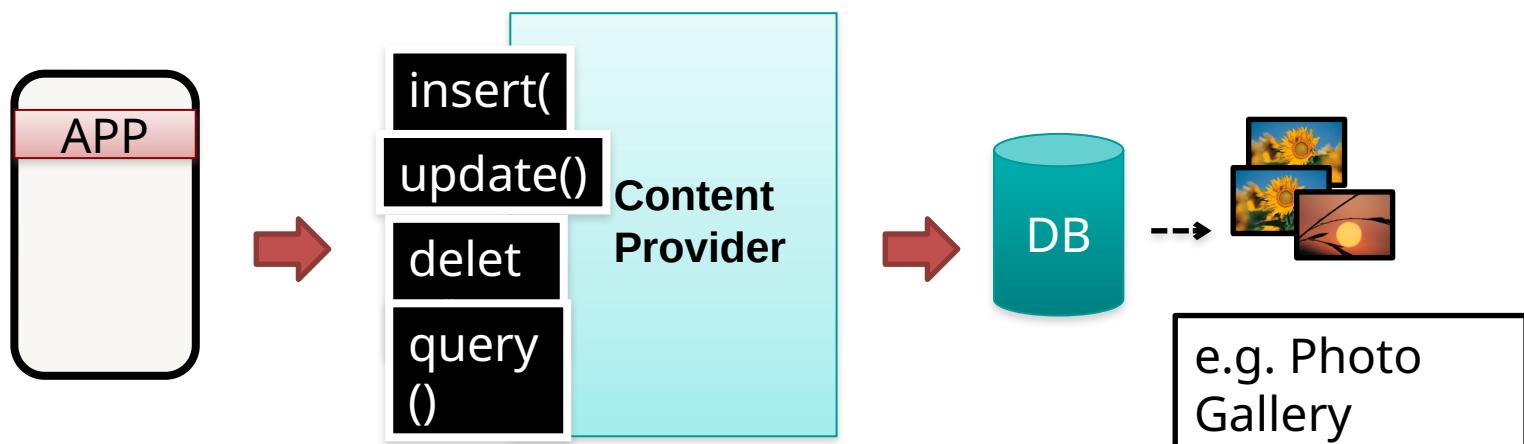
Android Components: Services

- **Services:** like Activities, but run in **background** and do not provide an user interface.
- Used for **non-interactive** tasks (e.g. networking).
- Service life-time composed of 3 states:



Android Components: Content Providers

- Each Android **application** has its own **private** set of data (managed through *files* or through *SQLite* database).
- **Content Providers**: Standard **interface** to *access and share data among different applications*.



Android Components: *System API*

- Using the **components** described so far, Android applications can then leverage the system API ...

SOME EXAMPLEs ...

- *Telephony Manager* data access (call, SMS, etc)
- *Sensor management* (GPS, accelerometer, etc)
- *Network connectivity* (Wifi, bluetooth, NFC, etc)
- *Web surfing* (HTTP client, WebView, etc)
- *Storage management* (files, SQLite db, etc)
-

Android Components: Google API

- ... or easily interface with other **Google services**:



Android Application Security

- Android applications run with a distinct system identity (Linux user ID and group ID), in an **isolated** way.
- Applications must explicitly share resources and data. They do this by declaring the ***permissions*** they need for additional capabilities.
 - Applications statically **declare** the permissions they require.
 - User must **give his/her consensus** during the installation.

Case Study - Linux

- 1. Evolution of UNIX**
- 2. LINUX Operating System & Design goals**
- 3. Interfaces to Linux**
- 4. List of Linux Utility Programs**
- 5. Linux kernel**
- 6. Process in Linux and PID, UID, GID in Linux**
- 7. Process Management System Calls in Linux**
- 8. User space thread and kernel space thread**
- 9. Booting process in Linux**
- 10. Memory management in Linux**
- 11. Paging in Linux**
- 12. Network File System calls in Linux**

LINUS TORVALDS POSTS FAMOUS MESSAGE - "HELLO EVERYBODY OUT THERE..." - AND RELEASES FIRST LINUX CODE



SLACKWARE BECOMES FIRST WIDELY ADOPTED DISTRIBUTION



TECH GIANTS BEGIN ANNOUNCING PLATFORM SUPPORT FOR LINUX



IBM RUNS FAMOUS LINUX AD DURING THE SUPERBOWL



THE LINUX FOUNDATION IS FORMED TO PROMOTE, PROTECT AND STANDARDIZE LINUX. LINUS IS A FELLOW



LINUX TURNS 20 AND POWERS THE WORLD'S SUPERCOMPUTERS, STOCK EXCHANGES, PHONES, ATMS, HEALTHCARE RECORDS, SMART GRIDS, THE LIST GOES ON



1991

1992

1993

1996

1998

1999

2003

2005

2007

2010

2011



LINUS LICENSES LINUX UNDER THE GPL, AN IMPORTANT DECISION THAT WILL CONTRIBUTE TO ITS SUCCESS IN THE COMING YEARS



LINUS VISITS AQUARIUM, GETS BIT BY A PENGUIN AND CHOOSES IT AS LINUX MASCOT



RED HAT GOES PUBLIC



LINUS APPEARS ON THE COVER OF BUSINESSWEEK WITH A STORY THAT HAILS LINUX AS A BUSINESS SUCCESS

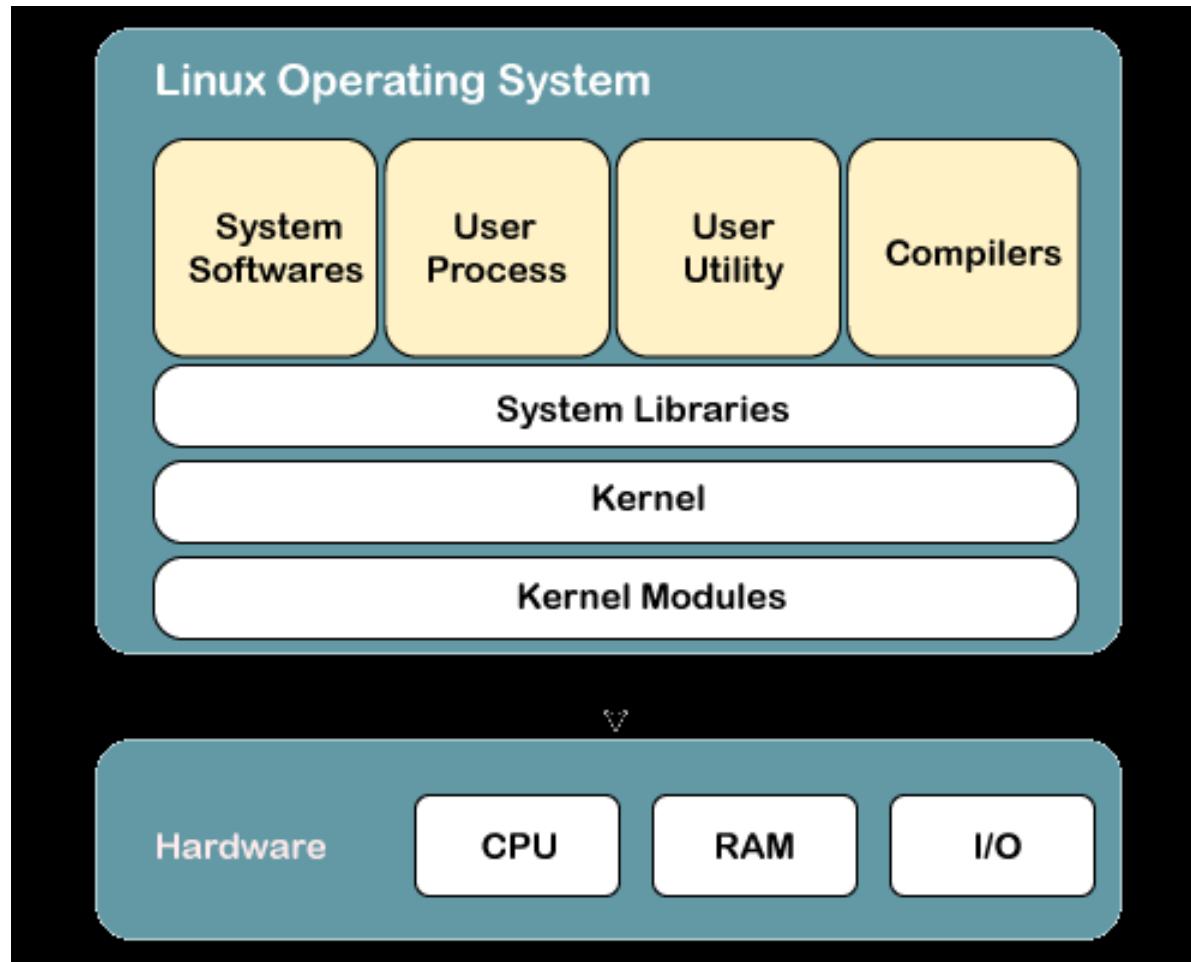


THE LINUX-BASED ANDROID OS OUTSHIPS ALL OTHER SMARTPHONE OSES IN THE U.S. AND CLIMBS TO DOMINANCE



THE LINUX FOUNDATION
<http://www.linuxfoundation.org/>

Linux architecture



Design goals of Linux

- Designed to handle **multiple processes and multiple users** at the same time
- Multi-user, multitasking system with a full set of UNIX-compatible tools.
- Speed, efficiency, and standardization.
- Designed to be compliant with the relevant POSIX documents.
- The Linux programming interface adheres to the **SVR4 UNIX semantics**.

List of Linux Utility Programs

Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
ps	List running processes
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

➤ Commands :

- **cat** - to create, view, concatenate files.
- **mv** - mv is used to move one or more files or directories
- **rm** - removes the entries for a specified file, group of files.
- **cmp, comm, diff** - to display the differences by comparing the files
- **chmod** - used to change the access permissions
- **top** - Displays list of running processes
- **head** - Pick lines from the beginning
- **tail** - Pick lines from the end
- **sort** - Sort, merge and remove
- **wc** - to find number of lines, word count, byte and characters.

cat – DISPLAYING AND CREATING FILES

5.1 cat: DISPLAYING AND CREATING FILES

cat is one of the most well-known commands of the UNIX system. It is mainly used to display the contents of a small file on the terminal:

```
$ cat dept.1st
01|accounts|6213
02|progs|5423
03|marketing|6521
04|personnel|2365
05|production|9876
06|sales|1006
```

cat, like several other UNIX commands, also accepts more than one filename as arguments:

```
cat chap01 chap02
```

The contents of the second file are shown immediately after the first file without any header information. In other words, **cat** concatenates the two files—hence its name.

cat – DISPLAYING AND CREATING FILES

cat options (-v and –n)

Displaying Nonprinting Characters (-v) **cat** is normally used for displaying text files only. Executables, when seen with **cat**, simply display junk. If you have nonprinting ASCII characters in your input, you can use **cat** with the **-v** option to display these characters.

Numbering Lines (-n) The **-n** option numbers lines. C compilers indicate the line number where errors are detected, and this numbering facility often helps a programmer in debugging programs. But then your **vi** editor can show line numbers too, and if your version of **cat** doesn't support **-n**, you can use the **pr** (12.2) command to do the same job.

cat – DISPLAYING AND CREATING FILES

cat is also useful for creating files

```
$ cat > foo
```

A > symbol following the command means that the output goes to the filename following it. cat used in this way represents a rudimentary editor.

[*Ctrl-d*]

```
$ _
```

Prompt returns

```
$ cat foo
```

A > symbol following the command means that the output goes to the filename following it. cat used in this way represents a rudimentary editor.

rm - DELETING FILES

```
rm chap01 chap02 chap03
```

rm chap could be dangerous to use!*

A file once deleted can't be recovered. **rm** won't normally remove a directory, but it can remove files from one. You can remove two chapters from the progs directory without having to "cd" to it:

```
rm progs/chap01 progs/chap02
```

Or rm progs/chap0[12]

You may sometimes need to delete all files in a directory as part of a cleanup operation. The *, when used by itself, represents all files, and you can then use **rm** like this:

```
$ rm *
```

```
$ -
```

All files gone!

DOS users, beware! When you delete files in this fashion, the system won't prompt you with the message All files in directory will be deleted! before removing the files! The \$ prompt will return silently; the work has been done. The * used here is equivalent to *.* used in DOS.

rm – DELETING FILES

5.3.1 rm Options

Interactive Deletion (-i) Like in **cp**, the **-i** (interactive) option makes the command ask the user for confirmation before removing each file:

```
$ rm -i chap01 chap02 chap03
rm: remove chap01 (yes/no)? ?y
rm: remove chap02 (yes/no)? ?n
rm: remove chap03 (yes/no)? [Enter]
```

No response—file not deleted

A **y** removes the file, any other response leaves the file undeleted.

rm – DELETING FILES

Recursive Deletion (-r or -R) With the **-r** (or **-R**) option, **rm** performs a tree walk—a thorough recursive search for all subdirectories and files within these subdirectories. At each stage, it deletes everything it finds. **rm** *won't normally remove directories, but when used with this option, it will.* Therefore, when you issue the command

```
rm -r *
```

Behaves partially like rmdir

you'll delete all files in the current directory and all its subdirectories. If you don't have a backup, then these files will be lost forever.

Forcing Removal (-f) **rm** prompts for removal if a file is write-protected. The **-f** option overrides this minor protection and forces removal. When you combine it with the **-r** option, it could be the most risky thing to do:

```
rm -rf *
```

Deletes everything in the current directory and below

rm – DELETING FILES

Caution: Make sure you are doing the right thing before you use `rm *`. Be doubly sure before you use `rm -rf *`. The first command removes only ordinary files in the current directory. The second one removes everything—files and directories alike. If the root user (the super user) invokes `rm -rf *` in the `/` directory, the entire UNIX system will be wiped out from the hard disk!

mv – RENAMING FILES

The **mv** command renames (moves) files. It has two distinct functions:

- It renames a file (or directory).
- It moves a group of files to a different directory.

mv doesn't create a copy of the file; it merely renames it. No additional space is consumed on disk during renaming. To rename the file `chap01` to `man01`, you should use

```
mv chap01 man01
```

If the destination file doesn't exist, it will be created. For the above example, **mv** simply replaces the filename in the existing directory entry with the new name. By default, **mv** doesn't prompt for overwriting the destination file if it exists. So be careful again.

cmp – COMPARING TWO FILES

You may often need to know whether two files are identical so one of them can be deleted. There are three commands in the UNIX system that can tell you that. In this section, we'll have a look at the **cmp** (compare) command. Obviously, it needs two filenames as arguments:

```
$ cmp chap01 chap02  
chap01 chap02 differ: char 9, line 1
```

The two files are compared byte by byte, and the location of the first mismatch (in the ninth character of the first line) is echoed to the screen. By default, **cmp** doesn't bother about possible subsequent mismatches but displays a detailed list when used with the **-l** (list) option.

comm – WHAT IS COMMON?

Suppose you have two lists of people and you are asked to find out the names available in one and not in the other, or even those common to both. **comm** is the command you need for this work. It requires two *sorted* files, and lists the differing entries in different columns. Let's try it on these two files:

```
$ cat file1
c.k. shukla
chanchal singhvi
s.n. dasgupta
sumit chakrobarty
```

```
$ cat file2
anil agarwal
barun sengupta
c.k. shukla
lalit chowdury
s.n. dasgupta
```

Both files are sorted and have some differences. When you run **comm**, it displays a three-columnar output:

comm – WHAT IS COMMON?

```
$ comm file[12]
```

Comparing file1 and file2

anil agarwal

barun sengupta

c.k. shukla

chanchal singhvi

lalit chowdury

s.n. dasgupta

sumit chakrobarty

The first column contains two lines unique to the first file, and the second column shows three lines unique to the second file. The third column displays two lines common (hence its name) to both files.

comm – WHAT IS COMMON?

This output provides a good summary to look at, but is not of much use to other commands that take **comm**'s output as their input. These commands require single-column output from **comm**, and **comm** can produce it using the options **-1**, **-2** or **-3**. To drop a particular column, simply use its column number as an option prefix. You can also combine options and display only those lines that are common:

```
comm -3 foo1 foo2
```

```
comm -13 foo1 foo2
```

Selects lines not common to both files
Selects lines present only in second file

The last example and one more with the other matching option (**-23**) has more practical value than you may think, but we'll not discuss their application in this text.

diff – CONVERTING ONE FILE TO OTHER

diff is the third command that can be used to display file differences. Unlike its fellow members, **cmp** and **comm**, it also tells you which lines in one file have to be *changed* to make the two files identical. When used with the same files, it produces a detailed output:

```
$ diff file1 file2
0a1,2
> anil aggarwal
> barun sengupta
2c4
< chanchal singhvi
--
> lalit chowdury
4d5
< sumit chakrobarty
```

Or diff file[12]

*Append after line 0 of first file
this line*

and this line

Change line 2 of first file

Replacing this line

with

this line

Delete line 4 of first file

containing this line

umask: DEFAULT FILE AND DIRECTORY PERMISSIONS

- Usually we have,
 - rw-rw-rw- (octal 666) for regular files
 - rwxrwxrwx (octal 777) for directories
- The default is transformed by subtracting the user mask from it to remove one or more permissions
- We can evaluate the current value of the mask as,

```
$ umask
```

```
022
```

The umask (UNIX shorthand for "user file-creation mode mask") is a **four-digit octal number that UNIX uses to determine the file permission for newly created files**. Every process has its own umask, inherited from its parent process.

FILE OWNERSHIP

- When you create a file, you become its owner (third column)
- Group owner of the file (fourth column)
- Several users may belong to a single group, but the privileges of the group are set by the owner of the file and not by the group members

FILE OWNERSHIP

- UNIX follows a three-tiered file protection system that determines a file's access rights
- Filetype owner (rwx) groupowner (rwx) others (rwx)

- Example:

```
-rwxr-xr-- 1 kumar metal 20500 may 10 19:21 chap02
```

r w x

r - x

r - -

owner/user

group owner

others

CHANGING FILE PERMISSIONS

- A file or a directory is created with a default set of permissions, which can be determined by umask
- Let us assume that the file permission for the created file is **-rw-r--r--**
- Using **chmod** command, we can change the file permissions and allow the owner to execute his file

RELATIVE AND ABSOLUTE PERMISSIONS

- In a relative manner, specify the changes to the current permissions
- In an absolute manner, specify the final permissions

RELATIVE PERMISSIONS

- chmod only changes the permissions specified in the command line and **leaves the other permissions unchanged**
- Syntax
chmod category operation permission filename(s)

RELATIVE PERMISSIONS

- chmod takes an expression as its argument which contains:
 1. user category (user, group, others)
 2. operation to be performed (assign or remove a permission)
 3. type of permission (read, write, execute)

RELATIVE PERMISSIONS

Category	operation	permission
u - user	+ assign	r - read
g - group	- remove	w - write
o - others	= absolute	x - execute
a - all (ugo)		

RELATIVE PERMISSIONS

- Examples

```
-rw-r--r-- 1      kumar  metal  1906  sep    23:38
                  xstart
```

`chmod u+x xstart`

```
-rwxr--r-- 1      kumar  metal  1906  sep    23:38
                  xstart
```

The command assigns (+) execute (x) permission to the user (u), other permissions remain unchanged

RELATIVE PERMISSIONS

- chmod ugo+x xstart
- chmod a+x xstart
- chmod +x xstart

```
-rwxr-xr-x 1 kumar metal 1906 sep 23:38 xstart
```

chmod accepts multiple file names in command line

- chmod u+x note note1 note3

RELATIVE PERMISSIONS

Let initially

```
-rwxr-xr-x 1 kumar metal 1906 sep 23:38 xstart
```

chmod go-r xstart

Then, it becomes

```
-rwx--x--x 1 kumar metal 1906 sep 23:38 xstart
```

ABSOLUTE PERMISSIONS

- Need not to know the current file permissions
- Set all nine permissions explicitly
- A string of three octal digits is used as an expression
- The permission can be represented by one octal digit for each category
- For each category, we add octal digits

ABSOLUTE PERMISSIONS

Octal	Permissions	Significance
0		- - -
	no permissions	
1		- - X
	execute only	
2		- W -
	write only	
3		- W X
	write and execute	
4		r - -
	read only	
5		r - X

ABSOLUTE PERMISSIONS

- Using relative permission

`chmod a+rwx xstart`

- Using absolute permission

`chmod 666 xstart`

`chmod 644 xstart`

`chmod 761 xstart`

will assign all permissions to the owner, read and write permissions for the group and only execute permission to the others

ABSOLUTE PERMISSIONS

- 777 signifies all permissions for all category, but still we can prevent a file from being deleted
- 000 signifies absence of all permissions for all category, but still we can delete a file
- It is the **directory permissions** that determines whether a file can be deleted or not

PERMISSIONS

- Only owner can change the file permissions
- User can not change other user's file's permissions
- But the system administrator can do anything

THE SECURITY IMPLICATIONS

- Let the default permission for the file xstart is
-rw-r--r--
 - `chmod u-rw, go-r xstart` or
 - `chmod 000 xstart`
-
- This is simply useless but still the user can delete this file

chmod RECURSIVELY

```
chmod -R a+x shell_scripts
```

This makes all the files and subdirectories found in the `shell_scripts` directory, executable by all users

DIRECTORY PERMISSIONS

- It is possible that a file cannot be accessed even though it has read permission, and can be removed even when it is write protected
- The default permissions of a directory are
 rwxr-xr-x (755)
- A directory must never be writable by group and others

DIRECTORY PERMISSIONS

This becomes 644 (666-022) for ordinary files and
755 (777-022) for directories

umask 000

Indicates, we are not subtracting anything and the
default permissions will remain unchanged

Note that, changing system wide default
permission settings is possible using chmod but
not by umask

CHANGING FILE OWNERSHIP

- On BSD and AT&T systems
- Kumar – owner (user)
- Metal – group owner
- If sharma copies a file of kumar, then sharma will become its owner and he can manipulate the attributes
- chown and chgrp
- On BSD, only system administrator can use chown
- On other systems, only the owner can change both

chown Command

- Changing ownership requires superuser permission, so use **su** command

ls -l note

```
-rwxr----x 1 kumar metal 347 may 10 20:30 note
```

- chown sharma note; ls -l note

```
-rwxr----x 1 sharma metal 347 may 10 20:30 note
```

chgrp Command

- This command changes the file's group owner
- No superuser permission is required

```
ls -l dept.lst
```

```
-rw-r--r-- 1 kumar metal 139 jun 8 16:43 dept.lst
```

```
chgrp dba dept.lst;
```

```
ls -l dept.lst
```

```
-rw-r--r-- 1 kumar dba 139 jun 8 16:43 dept.lst
```

sort : ordering a file

Sorting is the ordering of data in ascending or descending sequence. The sort command orders a file and by default, the entire line is sorted

This default sorting sequence can be altered by using certain options. We can also sort one or more keys (fields) or use a different ordering rule.

sort options

The important sort options are:

- | | |
|-----------------|--|
| -k n | sorts on nth field |
| -k m,n | starts sort on mth field and ends sort on nth field |
| -k m.n | starts sort on nth column of mth field |
| -u | removes repeated lines |
| -n | sorts numerically |
| -r | reverses sort order |
| -m list | merges sorted files in list |
| -c | checks if file is sorted |
| -o fname | places output in file fname |

sort : ordering a file

sort -t"|" -k 2 shortlist

sorts the second field (name)

sort -t"|" -r -k 2 shortlist or

sort -t"|" -k 2r shortlist

sort order can be reversed with this -r option.

sort -t"|" -k 3,3 -k 2,2 shortlist

sorting on secondary key is also possible as shown above.

sort -t"|" -k 5.7,5.8 shortlist

we can also specify a character position within a field to be the beginning of sort as shown above (sorting on columns).

head - display the start of a file

head displays the head, or start, of the file.

Syntax

head [options] file

Common Options

-n number number of lines to display, counting from the top of the file

-number same as above

Examples

By default **head** displays the first 10 lines. You can display more with the "**-n number**", or "**-number**" options, e.g., to display the first 40 lines:

head -40 filename or **head -n 40 filename**

tail - display the end of a file

tail displays the tail, or end, of the file.

Syntax

tail [options] file

Common Options

-number number of lines to display, counting from the bottom of the file

Examples

The default is to display the last 10 lines, but you can specify different line or byte numbers, or a different starting point within the file. To display the last 30 lines of a file use the **-number** style:

tail -30 filename

WC – COUNTING LINES, WORDS AND CHARACTERS

UNIX features a universal word-counting program that also counts lines and characters. It takes one or more filenames as arguments and displays a four-columnar output. Before you use **wc** on the file **infile**, just use **cat** to view its contents:

```
$ cat infile
I am the wc command
I count characters, words and lines
With options I can also make a selective count
```

You can now use **wc** without options to make a “word count” of the data in the file:

```
$ wc infile
 3    20    103 infile
```

wc counts 3 lines, 20 words and 103 characters. The filename has also been shown in the fourth column. The meanings of these terms should be clear to you as they are used throughout the book:

WC – COUNTING LINES, WORDS AND CHARACTERS

- A **line** is any group of characters not containing a newline.
- A **word** is a group of characters not containing a space, tab or newline.
- A **character** is the smallest unit of information, and includes a space, tab and newline.

wc offers three options to make a specific count. The **-l** option counts only the number of lines, while the **-w** and **-c** options count words and characters, respectively:

```
$ wc -l infile
```

3 infile

Number of lines

```
$ wc -w infile
```

20 infile

Number of words

```
$ wc -c infile
```

103 infile

Number of characters

WC – COUNTING LINES, WORDS AND CHARACTERS

When used with multiple filenames, **wc** produces a line for each file, as well as a total count:

```
$ wc chap01 chap02 chap03
 305  4058 23179 chap01
 550  4732 28132 chap02
 377  4500 25221 chap03
1232 13290 76532 total
```

A total as a bonus

Tee Command

Tee command is used to store and view (both at the same time) the output of any other command.

Tee command writes to the STDOUT, and to a file at a time as shown in the examples below

The following command writes the output only to the file and not to the screen.

```
$ ls > file
```

The following command (with the help of tee command) writes the output both to the screen (stdout) and to the file.

```
$ ls | tee file
```

top - display top CPU processes

SYNOPSIS

top [-] [d *delay*] [p *pid*] [q] [c] [C] [S] [s] [i] [n *iter*] [b] DESCRIPTION

top provides an ongoing look at processor activity in real time.

It displays a listing of the most CPU-intensive tasks on the system, and can provide an interactive interface for manipulating processes.

It can sort the tasks by CPU usage, memory usage and runtime. can be better configured than the standard top from the procps suite.

Most features can either be selected by an interactive command or by specifying the feature in the personal or system-wide configuration file. See below for more information.

grep Command

grep to search a file(s) for a pattern and display.

grep options to display, count, line numbers or
filenames

Regular expressions

Basic regular expressions (BRE)

Extended regular expressions (ERE)

sed to edit / manipulate an input stream

substitution features

repeated and remembered patterns

grep Command

grep : searching for a pattern

grep options

Basic regular expressions (BRE)

An introduction

The character class

The *

The dot

Specifying pattern locations Metacharacters

grep Command

- It scans the file / input for a pattern and displays lines containing the pattern, the line numbers or filenames where the pattern occurs
- It's a command from a special family in UNIX for handling search requirements

grep options pattern filename(s)

emp.lst (Data used for grep)

- 2233 | a.k.shukla | g.m | sales | 12/12/52 | 6000
- 9876 | jai sharma | director | production | 12/03/50 | 7000
- 5678 | sumit chakrobarty | d.g.m. | marketing | 19/04/43 | 6000
- 2365 | barun sengupta | director | personnel | 11/05/47 | 7800
- 5423 | n.k.gupta | chairman | admin | 30/08/56 | 5400
- 1006 | chanchal singhvi | director | sales | 03/09/38 | 6700
- 6213 | karuna ganguly | g.m. | accounts | 05/06/62 | 6300
- 1265 | s.n. dasgupta | manager | sales | 12/09/63 | 5600
- 4290 | jayant choudhury | executive | production | 07/09/50 | 6000
- 2476 | anil aggarwal | manager | sales | 01/05/59 | 5000
- 6521 | lalit chowdury | directir | marketing | 26/09/45 | 8200
- 3212 | shyam saksena | d.g.m. | accounts | 12/12/55 | 6000
- 3564 | sudhir agarwal | executive | personnel | 06/07/47 | 7500
- 2345 | j. b. sexena | g.m. | marketing | 12/03/45 | 8000
- 0110 | v.k.agrawal | g.m. | marketing | 31/12/40 | 9000

grep Command

grep “sales” emp.lst

- Patterns with and without quotes is possible
- It's generally safe to quote the pattern
- Quote is mandatory when pattern involves more than one word
- It returns the prompt in case the pattern can't be located

grep president emp.lst

grep Command

- When grep is used with multiple filenames, it displays the filenames along with the output

```
grep "director" emp1.lst emp2.lst
```

Where it shows filename followed by the contents

grep frequently used options

- i ignores case for matching
- v doesn't display lines matching expression
- n displays line numbers along with lines
- c displays count of number of occurrences
- l displays list of filenames only

grep Command

- e exp specifies expression with this option
- x matches pattern with entire line
- f file takes patterns from file, one per line
- E treats pattern as an extended RE
- F matches multiple fixed strings

grep Command

1. grep -i 'agarwal' emp.lst
2. grep -v 'director' emp.lst > otherlist
wc -l otherlist will display 11 otherlist
1. grep -n 'marketing' emp.lst
2. grep -c 'director' emp.lst
3. grep -c 'director' emp*.lst
will print filenames prefixed to the line count

grep Command

1. `grep -i 'manager' *.lst`
will display filenames *only*
2. `grep -e 'Agarwal' -e 'aggarwal' -e 'agrawal'`
`emp.lst`
will print matching multiple patterns

BASIC REGULAR EXPRESSIONS

- It is tedious to specify each pattern separately with the -e option
- grep uses an expression of a different type to match a group of similar patterns
- if an expression uses meta characters, it is termed a regular expression
- Some of the characters used by regular expression are also meaningful to the shell

BRE character subset

- * Zero or more occurrences
- g*** nothing or g, gg, ggg, etc.
- .
- .
- *
- [pqr]** a single character p, q or r
- [c1-c2]** a single character within the ASCII range represented by c1 and c2

The character class

- grep supports basic regular expressions (BRE) by default and extended regular expressions (ERE) with the –E option
- A regular expression allows a group of characters enclosed within a pair of [], in which the match is performed for a single character in the group

grep Command

```
grep “[aA]g[ar][ar]wal” emp.lst
```

- A single pattern has matched two similar strings
- The pattern [a-zA-Z0-9] matches a single alphanumeric character. When we use range, make sure that the character on the left of the hyphen has a lower ASCII value than the one on the right

Negating a class (^) (caret)

THE *

* Zero or more occurrences of the previous character

g* nothing or g, gg, ggg, etc.

grep “[aA]gg*[ar][ar]wal” emp

Notice that we don't require to use –e option three times to get the same output!!!!

THE DOT

A dot matches a single character

.* signifies any number of characters or none

grep “j.*saxena” emp.lst

^ and \$

Most of the regular expression characters are used for matching patterns, but there are two that can match a pattern at the beginning or end of a line

- ^ for matching at the beginning of a line
- \$ for matching at the end of a line

grep Command

grep “^2” emp.lst

Selects lines where emp_id starting with 2

grep “7...\$” emp.lst

Selects lines where emp_salary ranges between
7000 to 7999

grep “^[^2]” emp.lst

Selects lines where emp_id doesn't start with 2

When meta characters lose their meaning

- It is possible that some of these special characters actually exist as part of the text
- Sometimes, we need to escape these characters

Eg: when looking for a pattern g^* , we have to use \

To look for [, we use \[

To look for .*, we use \.*

EXTENDED RE (ERE)

- If current version of grep doesn't support ERE, then use egrep but without the –E option
 - -E option treats pattern as an ERE
- + matches one or more occurrences of the previous character
- ? Matches zero or one occurrence of the previous character

grep ERE

b+ matches b, bb, bbb, etc.

b? matches either a single instance of b or nothing

These characters restrict the scope of match as compared to the *

grep -E “[aA]gg?arwal” emp.lst

The ERE set

ch+	matches one or more occurrences of character ch
ch?	Matches zero or one occurrence of character ch
exp1 exp2	matches exp1 or exp2
(x1 x2)x3	matches x1x3 or x2x3

Matching multiple patterns

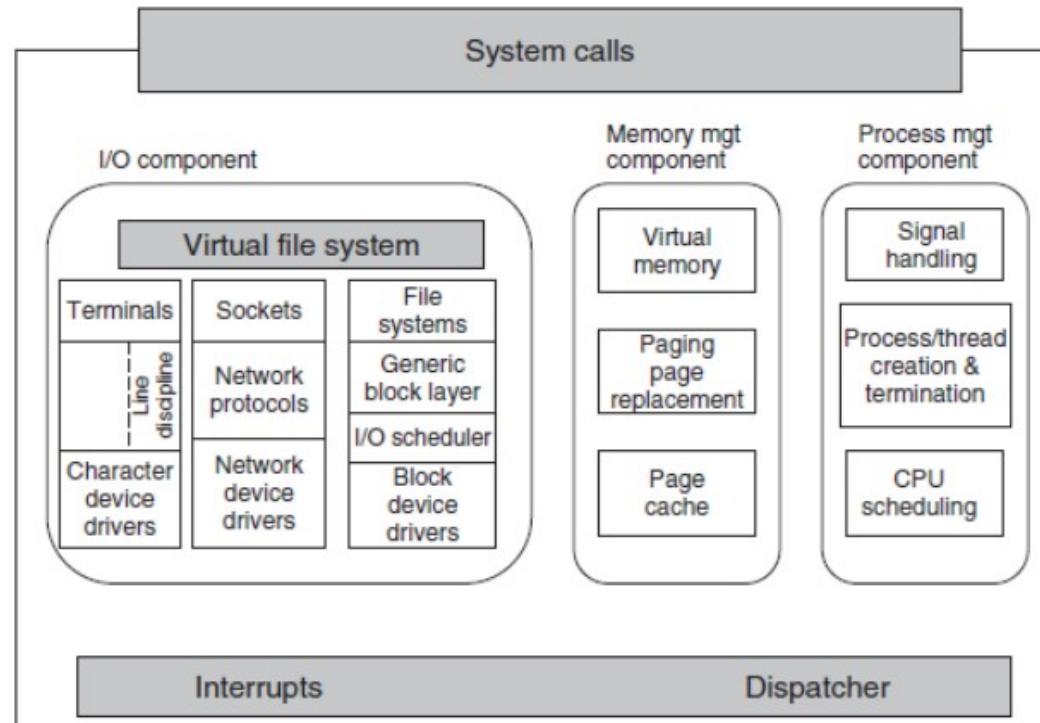
```
grep -E 'sengupta|dasgupta' emp.lst
```

We can locate both without using –e option twice,
or

```
grep -E '(sen|das)gupta' emp.lst
```

Linux Kernel

- It is a Unix-like computer operating system kernel.
- The **Linux kernel API**, the Application Programming Interface (API) through which user programs interact with the kernel, is meant to be very stable and to not break user space programs.

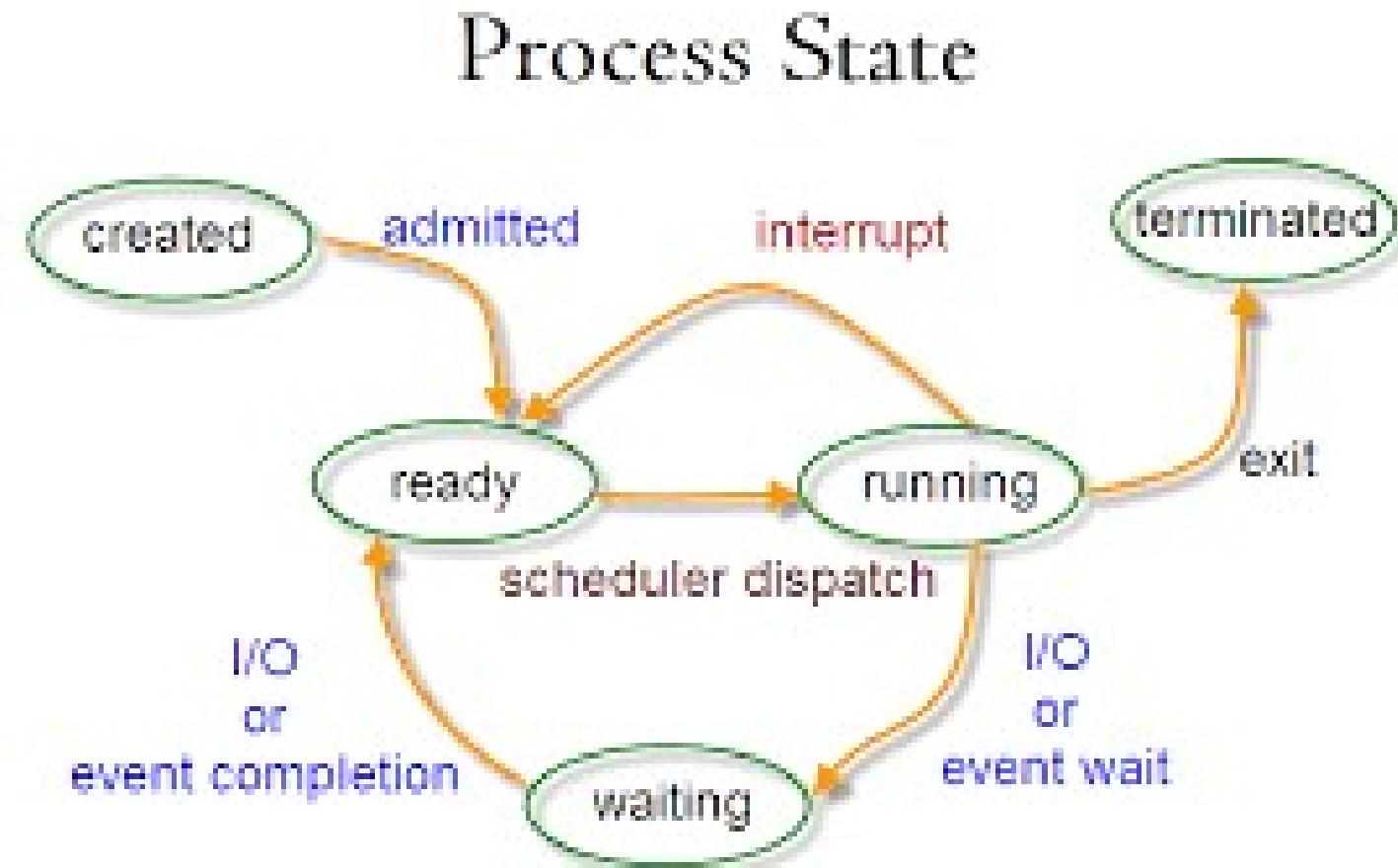


Process in Linux and PID, UID, GID in Linux

- The fork system call creates an exact copy of the original process.
- The forking process is called the **parent process**. The new process is called the **child process**.
- Process Identifier is when each process has a unique identifier associated with it known as **process id**.
- **User and Group Identifiers (UID and GID)** are the identifiers associated with a processes of the user and group.

Signal	Cause
SIGABRT	Sent to abort a process and force a core dump
SIGALRM	The alarm clock has gone off
SIGFPE	A floating-point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written to a pipe which has no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

Linux Processes



Linux Processes

- State
- Running
- Waiting
- Stopped
- Zombie – dead processes (not removed)
- Scheduling Information
- Identifiers
- Inter-Process Communication
- Links
- Times and Timers
- Virtual memory
- Processor Specific Context
- File system

Process Management System Calls in Linux

Syscall	Description
<u>CLONE</u>	Create a child process
<u>FORK</u>	Create a child process
<u>VFORK</u>	Create a child process and block parent
<u>EXECVE</u>	Execute program
<u>EXECVEAT</u>	Execute program relative to a directory file descriptor
<u>EXIT</u>	Terminate the calling process
<u>EXIT_GROUP</u>	Terminate all threads in a process
<u>WAIT4</u>	Wait for process to change state
<u>WAITID</u>	Wait for process to change state

Syscall	Description
<u>GETPID</u>	Get process ID
<u>GETPPID</u>	Get parent process ID
<u>GETTID</u>	Get thread ID

Syscall	Description
<u>SETSID</u>	Set session ID
<u>GETSID</u>	Get session ID

Process Management System Calls in Linux

Syscall	Description
<u>SETPGID</u>	Set process group ID
<u>GETPGID</u>	Get process group ID
<u>GETPGRP</u>	Get the process group ID of the calling process

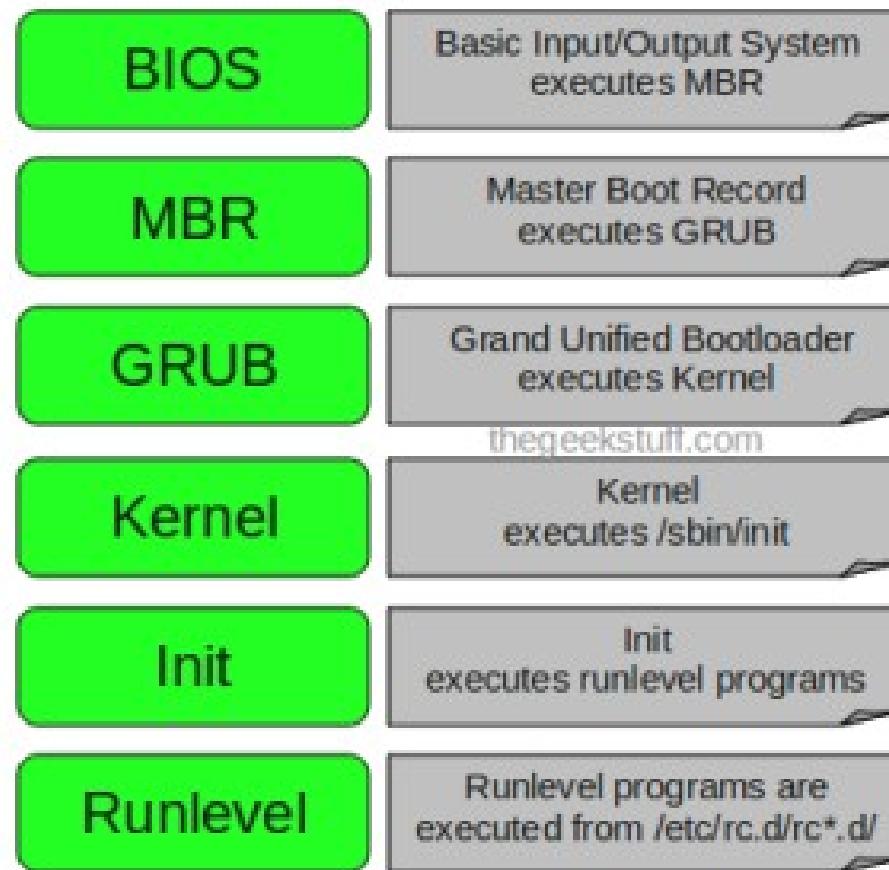
Syscall	Description
<u>SETUID</u>	Set real user ID
<u>GETUID</u>	Get real user ID
<u>SETGID</u>	Set real group ID
<u>GETGID</u>	Get real group ID
<u>SETRESUID</u>	Set real, effective and saved user IDs
<u>GETRESUID</u>	Get real, effective and saved user IDs
<u>SETRESGID</u>	Set real, effective and saved group IDs
<u>GETRESGID</u>	Get real, effective and saved group IDs
<u>SETREUID</u>	Set real and/or effective user ID
<u>SETREGID</u>	Set real and/or effective group ID
<u>SETFSUID</u>	Set user ID used for file system checks
<u>SETFSGID</u>	Set group ID used for file system checks
<u>GETEUID</u>	Get effective user ID

<u>GETEGID</u>	Get effective group ID
<u>SETGROUPS</u>	Set list of supplementary group IDs
<u>GETGROUPS</u>	Get list of supplementary group IDs

User and Kernel Space Thread

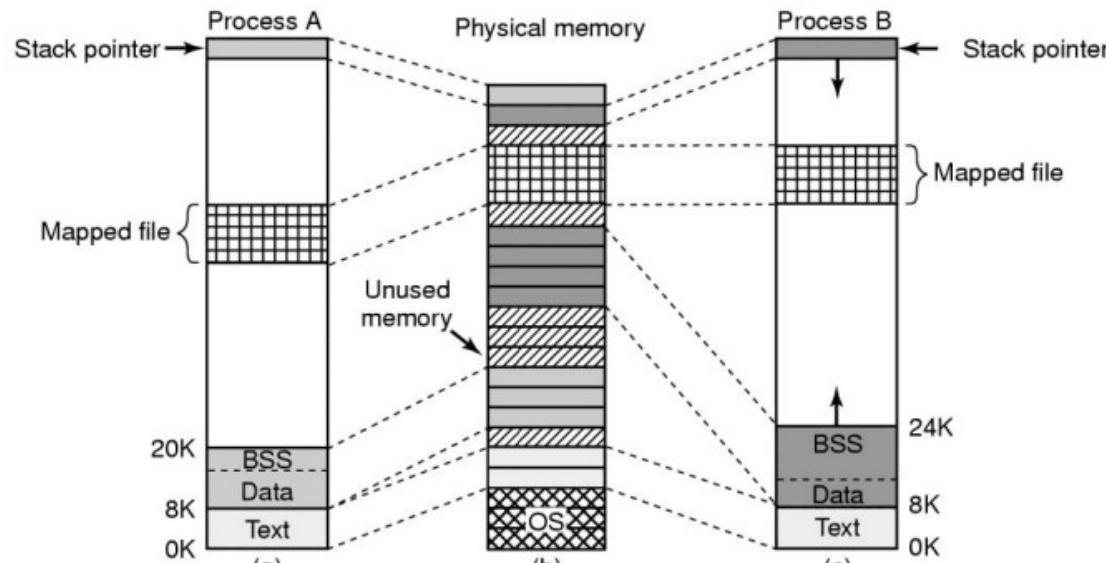
- **User Level Threads** – User managed threads.
- **Kernel Level Threads** – Operating System managed threads acting on kernel, an operating system core.
- **User-Space Threads:** cooperative multitasking, user threads typically can switch faster than kernel threads.
- **Kernel-space threads** often are implemented in the kernel using several tables.

Booting process in Linux



Memory Management in Linux

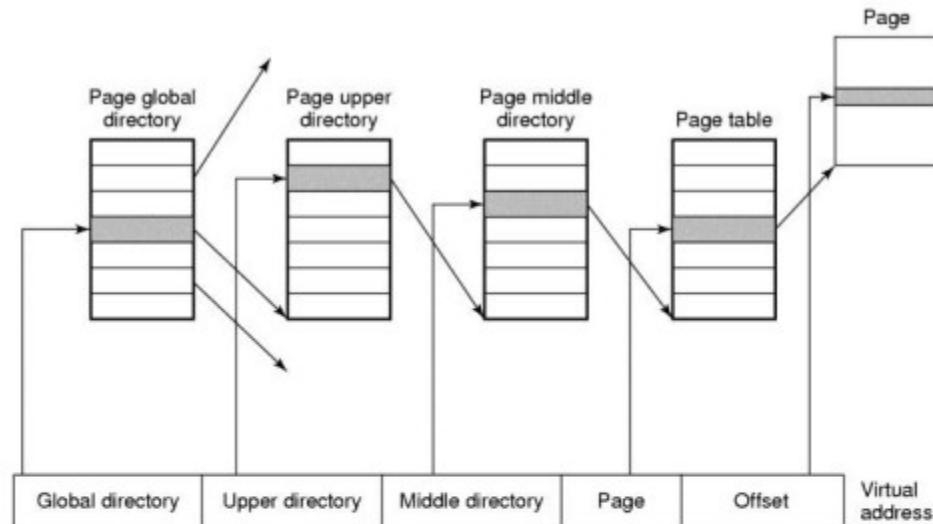
- The **memory management subsystem** is one of the most important parts of the operating system.
- Virtual memory** makes the system appear to have more memory than it actually has by sharing it between competing processes as they need it.



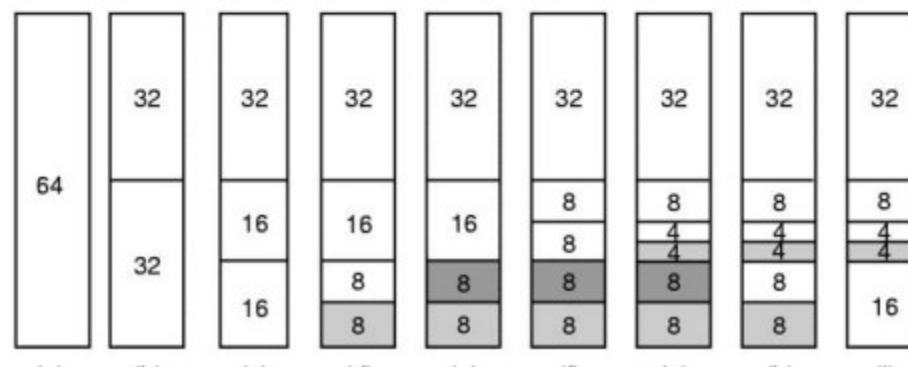
Paging in Linux

- The technique of only loading virtual pages into memory as they are accessed is known as demand paging.
- There are three kernel variables which control the paging operation: `minfree`, `desfree`, `lotsfree`.
- A `demand paging system` is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance.

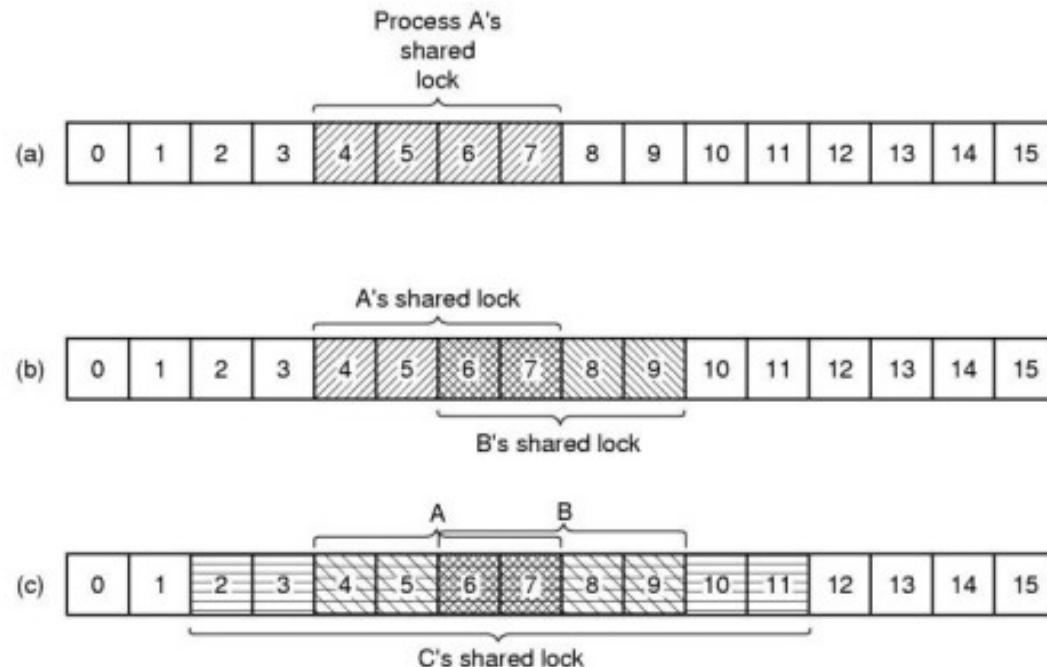
Physical Memory Management



Memory Allocation Mechanisms



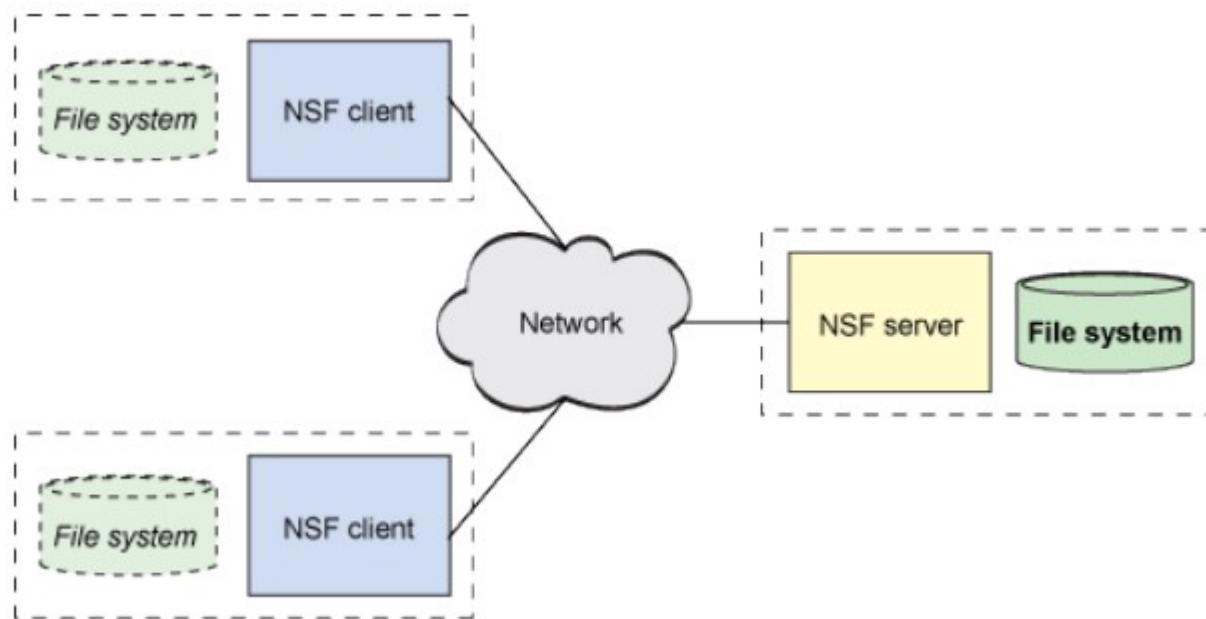
The Linux File System



(a) A file with one lock. (b) Addition of a second lock. (c) A third lock.

Network File System (NFS) calls in Linux

- The Network File System (NFS) is a way of mounting **Linux discs/directories** over a network.
- An NFS server can export one or more directories that can then be mounted on a remote Linux machine.



Android vs Linux

The “Not-So-Obvious” Differences

	Android	Linux
Type of Software	Operating System	Kernel
Processor architecture support	ARM A architecture	x86 and x64 architectures
Communication Channel	Optimized for Mobile networks	Optimized for Ethernet and Wi-Fi
Need for creation	To make Mobile Devices smart and as a competitor for iOS	For the spirit of having an open source software and for research and development purposes
How the OS makes money	By pushing google products and through commissions from apps, movies, music, books, etc. that are sold on the Google Play Store	Most distros are community maintained and are non-profitable organizations, main source of income include tech support in the enterprise.

The Similarities

	Android	Linux
Kernel	Linux Based Kernel	Linux Kernel
License Type	Open Source, Apache License	Open Source, GPL V2.0 License
Price	Free	Free



The Obvious Differences

	Android	Linux
Main Device Type	Smart phones	Personal Computers and Servers
Main use cases	Content Consumption and Communication	Content creation and more serious works
Other Application areas	Limited to use in Mobiles, Tablets, TVs, etc.	Can be ported to virtually any device like laptops, desktops, servers, routers, phones, TVs, washing machines, refrigerators, microwave ovens, industrial applications, etc.
Ease of use	Very Easy	Has a little bit of a learning curve based on the Distro you choose!
Maintained by	Google	Lots of Companies and communities maintain the various Linux distros

The Technical Differences

	Android	Linux
User Space	Android Run time (custom java runtime) and Core Libraries	Mainly composed of GNU and X11 server
Kernel Optimization	Optimized for power consumption	A balanced trade-off between performance and power consumption
Shell interface	Usually locked up, need to root the device to get access to the shell interface	Available out of the box as shell is a particularly important part of Linux Distros!
Base for the Graphical UI	Google designed Surface Flinger on top of OpenGL libraries	Most distros use X11 X server based Graphical User Interface.

The Technical Differences

BIOS/EFI	Not present on ARM devices	Standard for all x86/x64 motherboards
C library	Lightweight Bionic libc	GNU C library

The Technical Differences

Main Architectural elements	Just the Kernel, HAL and a non-GNU user space and its own desktop environment	Kernel, hardware abstraction layer (HAL) (drivers), GNU User space (apps) and UI layers (Desktop environment)
Hardware Drivers	Almost all of the drivers are proprietary	Majority are open source
Kernel's connection with peripherals	Usually I2c and SPI, so the kernel needs to be aware that a particular peripheral exists beforehand, and drivers have to be preloaded	Uses PCI, PCIe, USB etc. These protocols are self-identifying and allow the kernel to probe these devices and load necessary drivers
Bootloader	Simple bootloader, that does the minimum necessary and hands the control over to the kernel	Standardized bootloaders like Grub2 that can be customized



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 6

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Thangakumar J
WILP & Department of CS & IS





BITS Pilani
Pilani Campus

Operating systems – case study



Android ...



- **Android** is a *Linux-based platform* for *mobile devices* ...
 - *Operating System*
 - *Middleware*
 - *Applications*
 - *Software Development Kit (SDK)*
- Which kind of **mobile devices** ... (examples)



SMARTPHONES



TABLETS



EREADERS

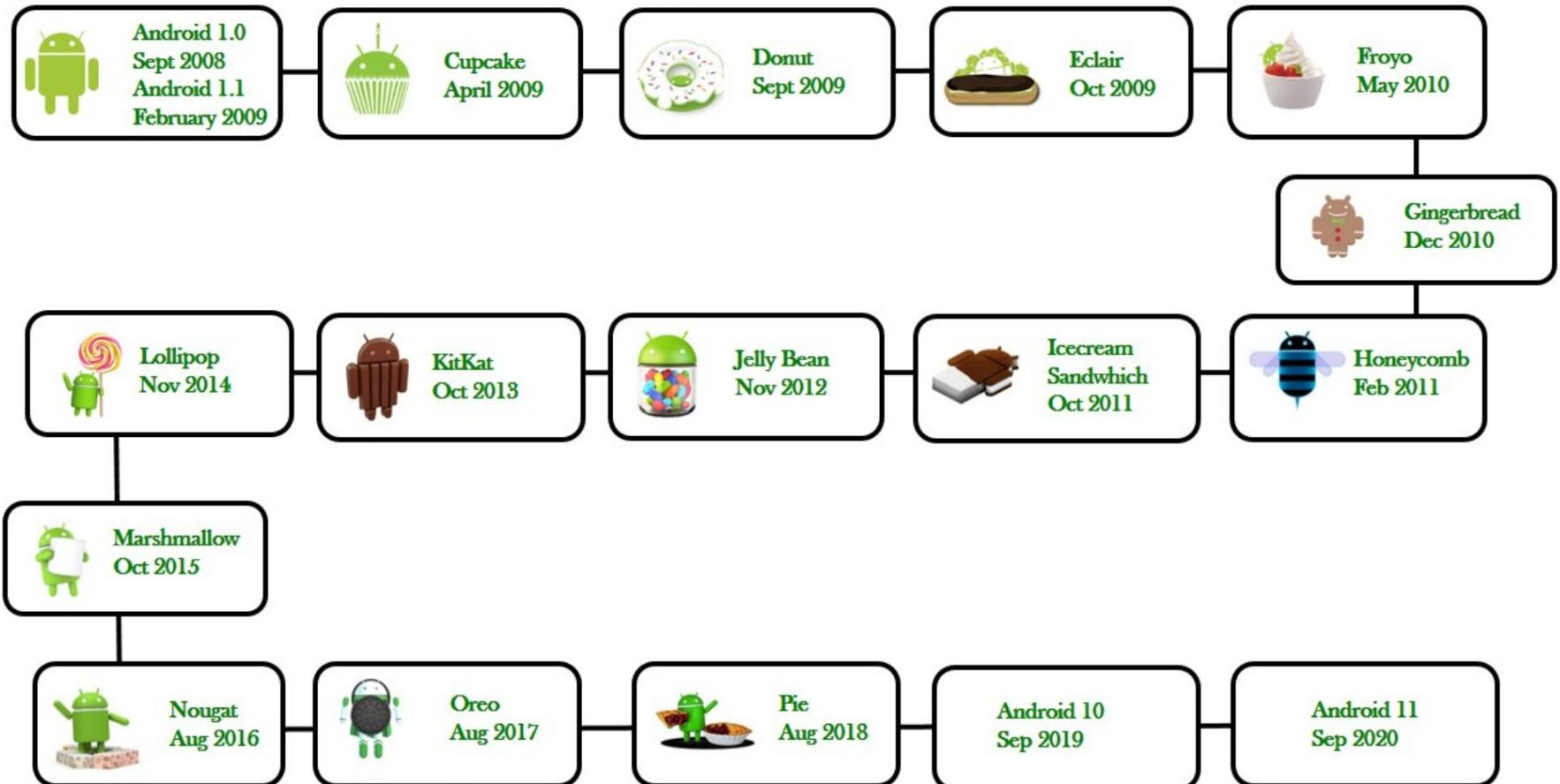


ANDROID TV



GOOGLE GLASSES



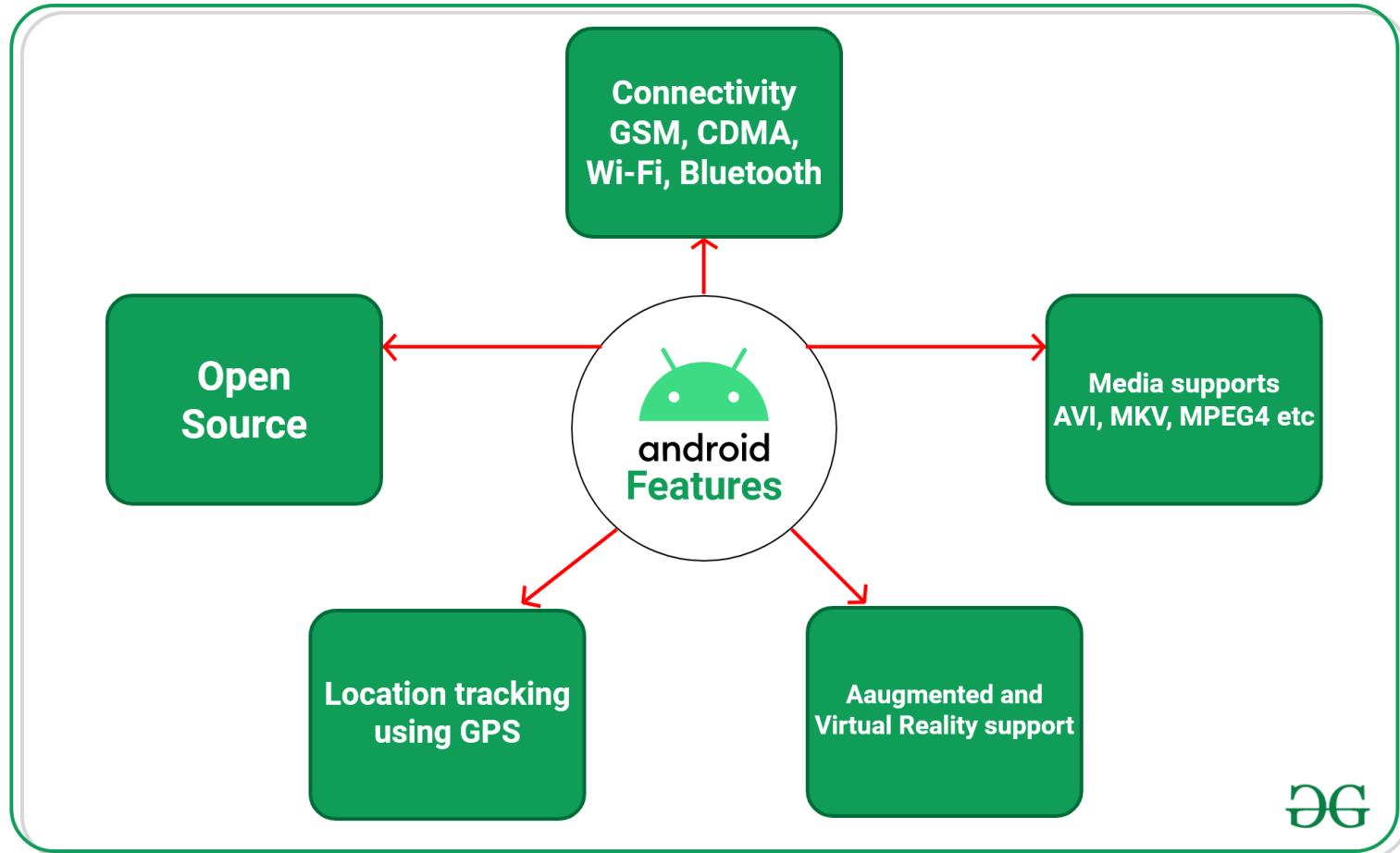


VERSIONS OF ANDROID



Major Devices that runs on Android Operating System



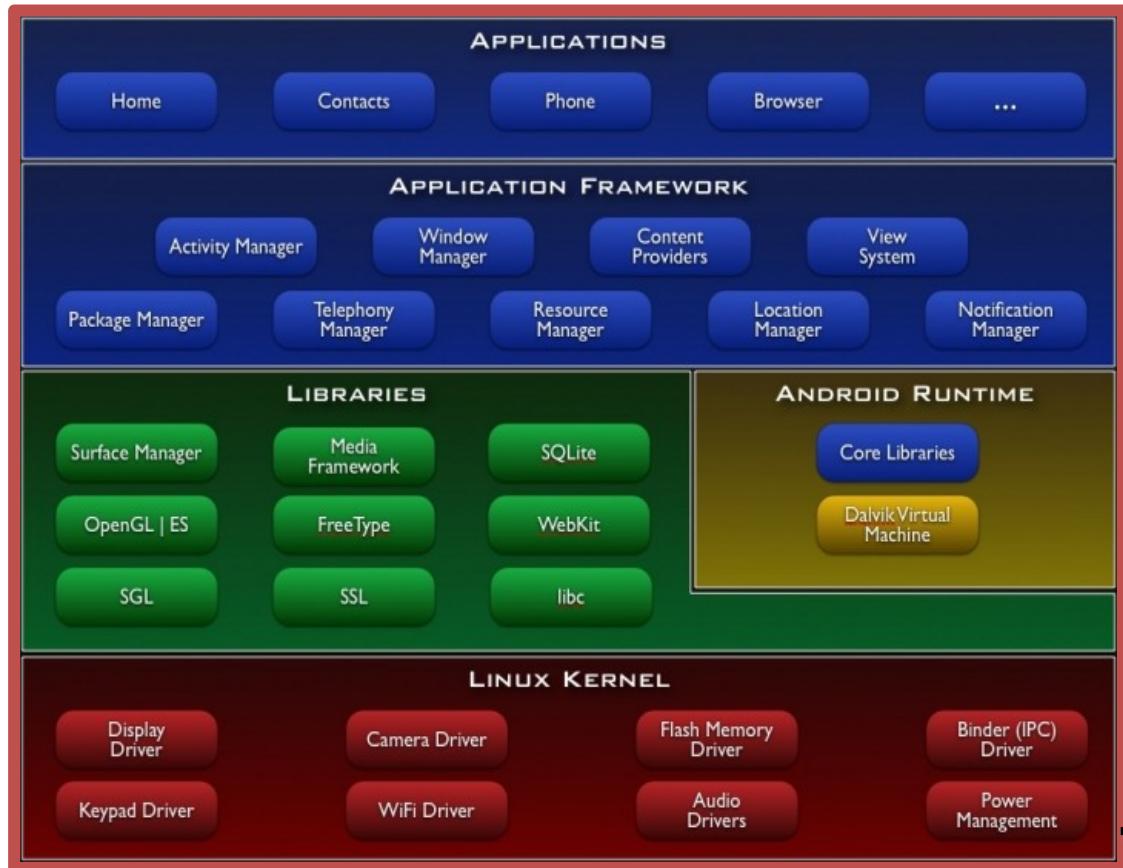


- Android Open Source Project so we can customize the OS based on our requirements.
- Android supports different types of connectivity for GSM, CDMA, Wi-Fi, Bluetooth, etc. for telephonic conversation or data transfer.
- Using wifi technology we can pair with other devices while playing games or using other applications.
- It contains multiple APIs to support location-tracking services such as GPS.
- We can manage all data storage-related activities by using the file manager.

- It contains a wide range of media supports like AVI, MKV, FLV, MPEG4, etc. to play or record a variety of audio/video.
- It also supports different image formats like JPEG, PNG, GIF, BMP, MP3, etc.
- It supports multimedia hardware control to perform playback or recording using a camera and microphone.

- Android has an integrated open-source WebKit layout-based web browser to support User Interfaces like HTML5, and CSS3.
- Android supports multi-tasking means we can run multiple applications at a time and can switch between them.
- It provides support for virtual reality or 2D/3D Graphics.

The Android Architecture



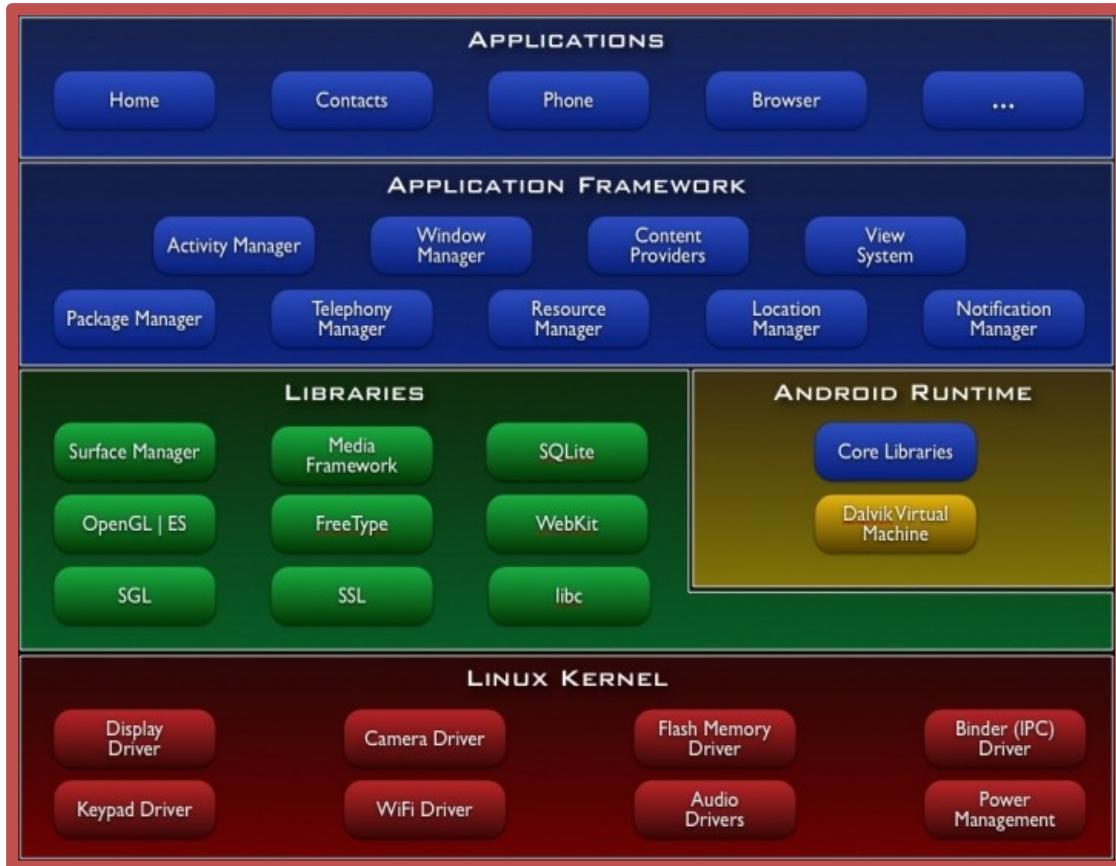
Built on top of
Linux kernel

Advantages:

- **Portability** (i.e. easy to compile on different hardware architectures)
- **Security** (e.g. secure multi-process environment)
- **Power Management**

- Memory and Process management □ File & Network I/O □ Device Drivers
- Preemptive Multitasking
- Lean, efficient, and secure
- Open Source

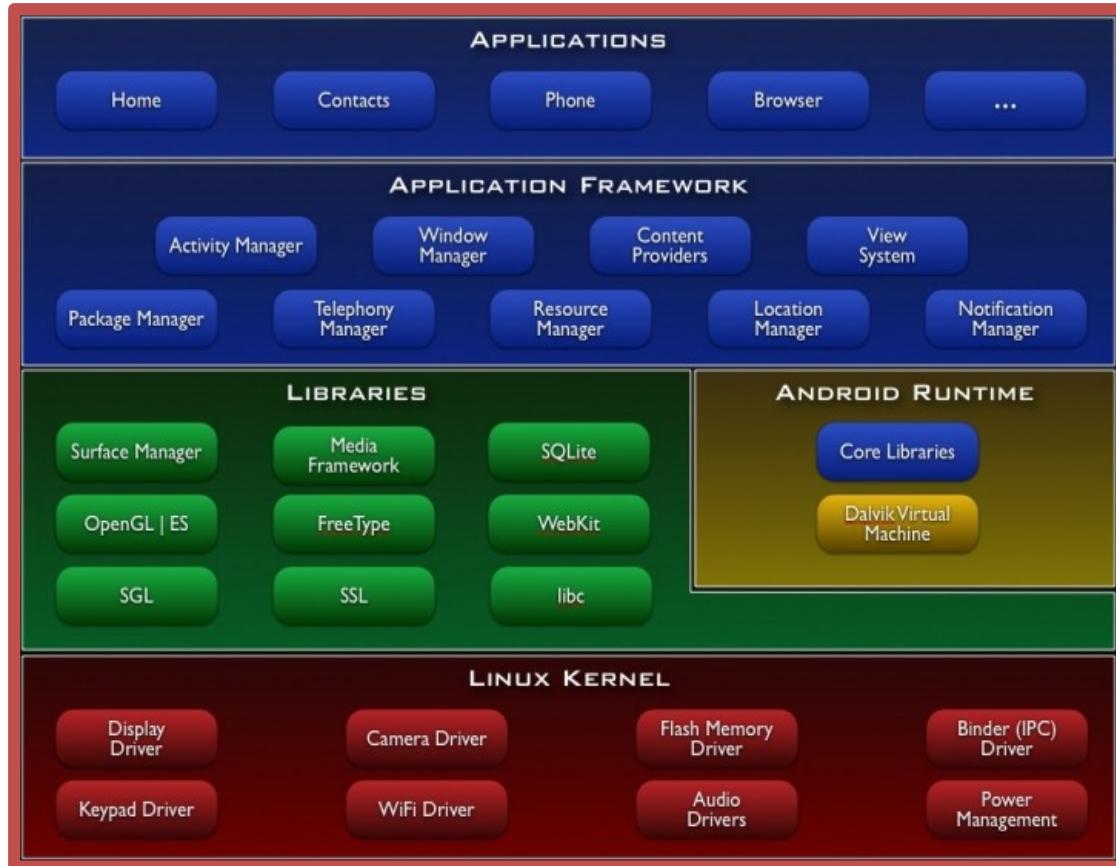
The Android Architecture



Native Libraries (C/C++ code)

- **Graphics** (Surface Manager)
- **Multimedia** (Media Framework)
- **Database DBMS** (SQLite)
- **Font Management** (FreeType)
- **WebKit**
- **C libraries** (Bionic)
-

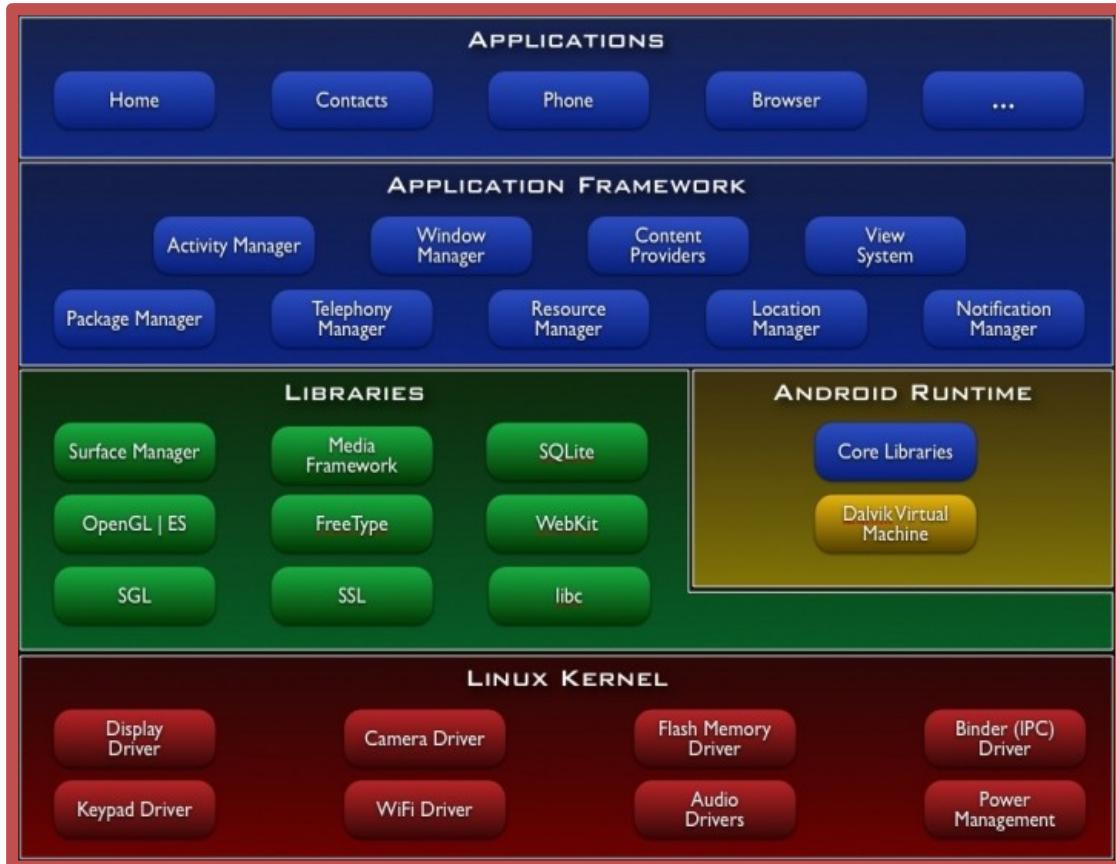
The Android Architecture



Application Libraries (Core Components of Android)

- Activity Manager
- Packet Manager
- Telephony Manager
- Location Manager
- Contents Provider
- Notification Manager
-

The Android Architecture

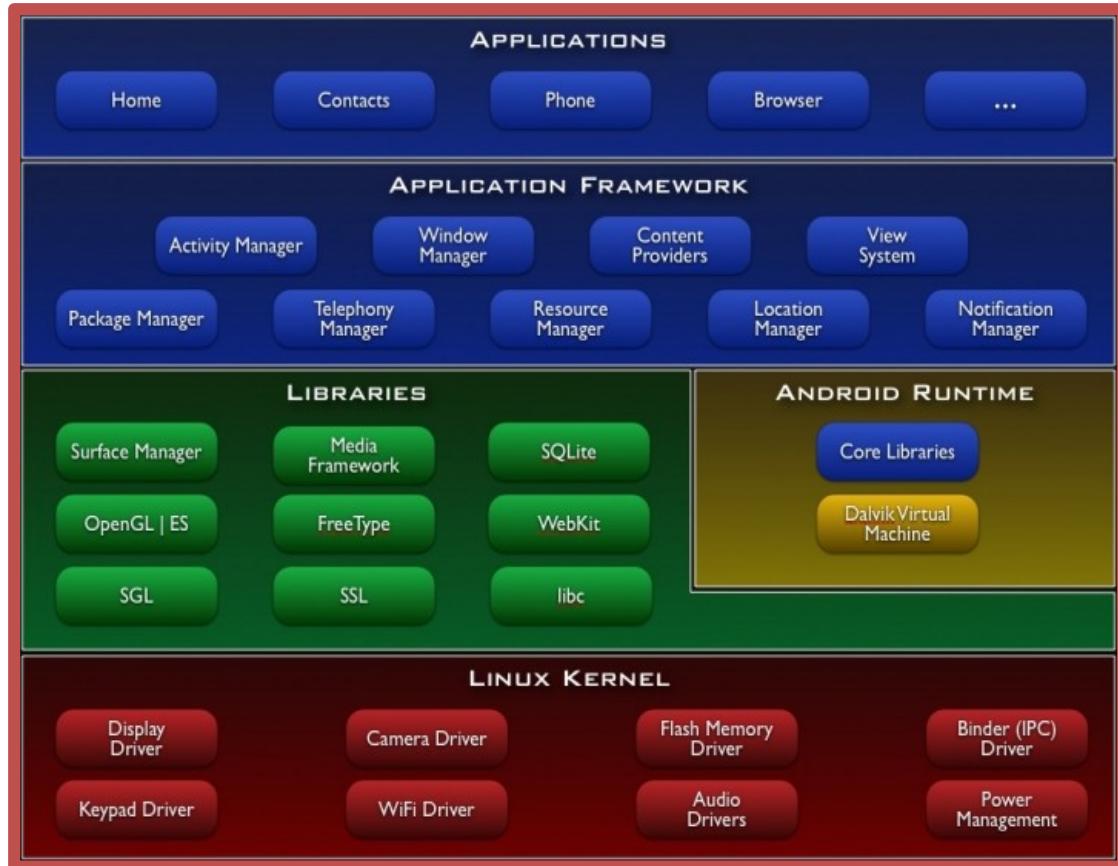


Applications

(Written in **Java** code)

- **Android Play Store**
- **Entertainment**
- **Productivity**
- **Personalization**
- **Education**
- **Geo-communication**
-

The Android Architecture



Dalvik Virtual Machine (VM)

- **Novel Java Virtual Machine implementation (not using the Oracle JVM)**
- **Open License** (Oracle JVM is not open!)
- **Optimized for memory-constrained devices**
- **Faster than Oracle JVM**
-

Android Applications Design

APPLICATION DESIGN:

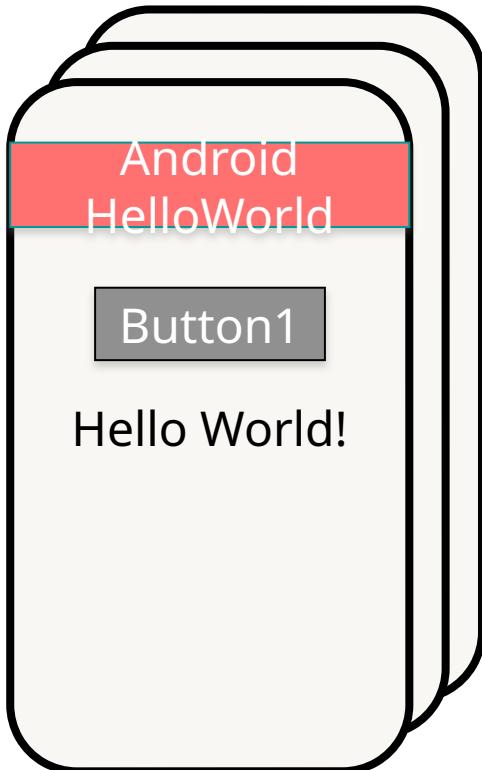
- **GUI** Definition
- **Events** Management
- Application **Data** Management
- **Background** Operations
- **User** Notifications

Android Applications Design

APPLICATION COMPONENTS

- **Activities**
- **Intents**
- **Services**
- **Content Providers**
- **Broadcast Receivers**

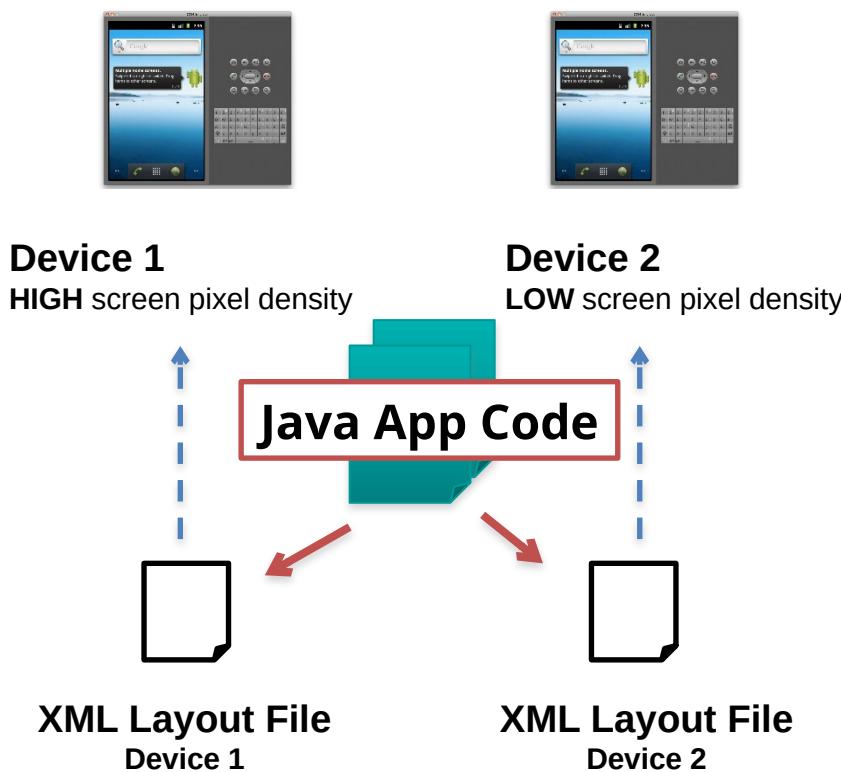
Android Components: Activities



- An **Activity** corresponds to a **single screen** of the **Application**.
- An Application can be composed of *multiple screens* (Activities).
- The **Home Activity** is shown when the user launches an application.
- Different activities can exchange information one with each other.

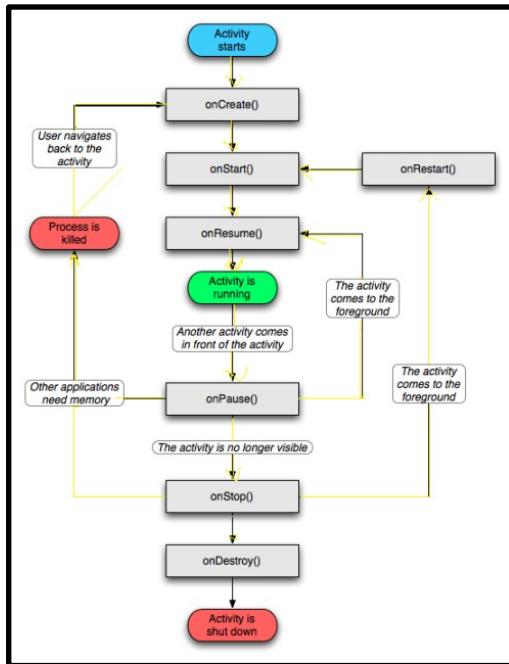
Android Components: Activities

EXAMPLE



- Build the **application layout** through XML files (like HTML)
- Define **two** different XML **layouts** for two different devices
- At **runtime**, Android detects the current device configuration and loads the appropriate resources for the application
- **No need to recompile!**
- Just add a new XML file if you need to support a new device

Android Components: Activities



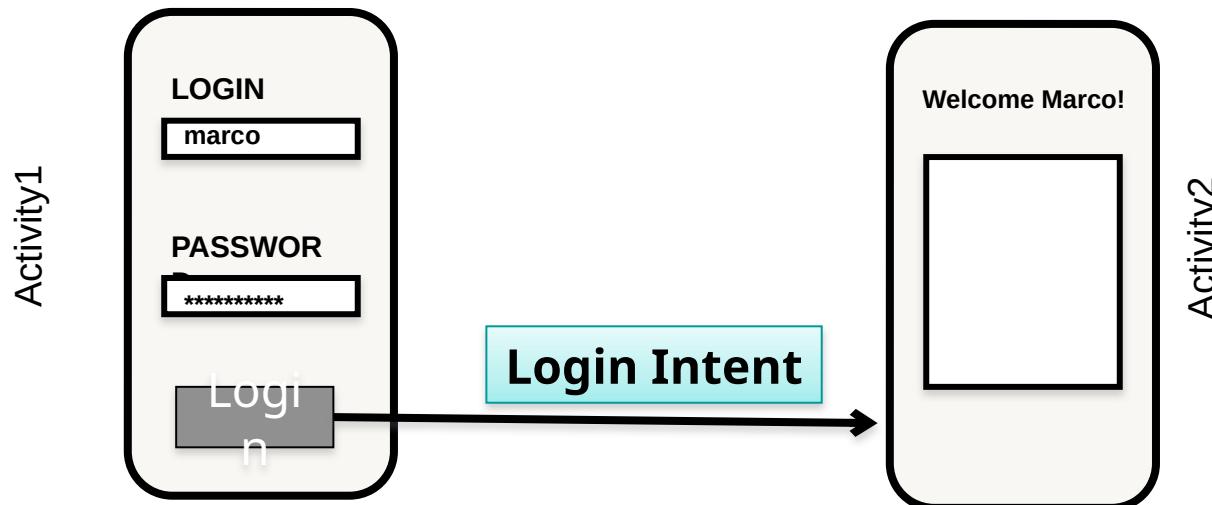
- The **Activity Manager** is responsible for creating, destroying, managing activities.
- Activities can be on different **states**: *starting, running, stopped, destroyed, paused*.
- Only one activity can be on the **running** state at a time.
- Activities are organized on a **stack**, and have an event-driven life cycle (details later ...)

Android Components: Activities

- Main difference between Android-programming and Java (Oracle)-programming:
 - **Mobile devices have constrained resource capabilities!**
- Activity lifetime depends on **users' choice** (i.e. change of visibility) as well as on **system constraints** (i.e. memory shortage).
- Developer must implement **lifecycle methods** to account for state changes of each Activity ...

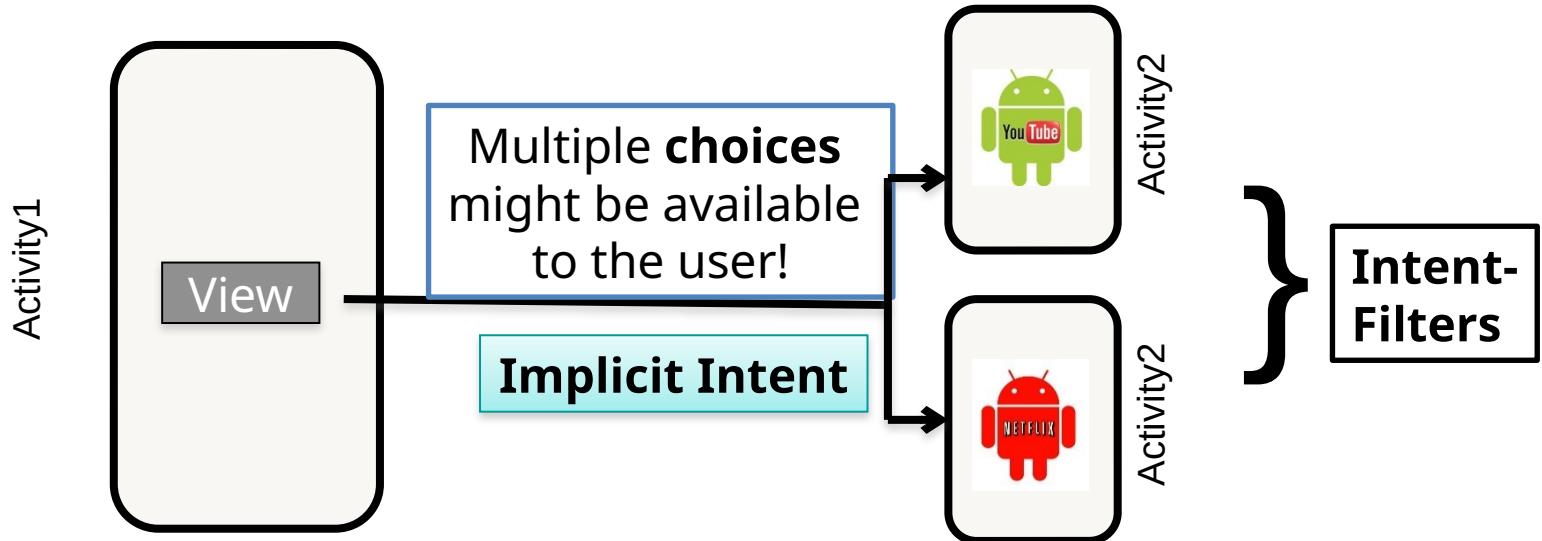
Android Components: Intents

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Explicit Intent** → The component (e.g. *Activity1*) specifies the destination of the intent (e.g. *Activity 2*).



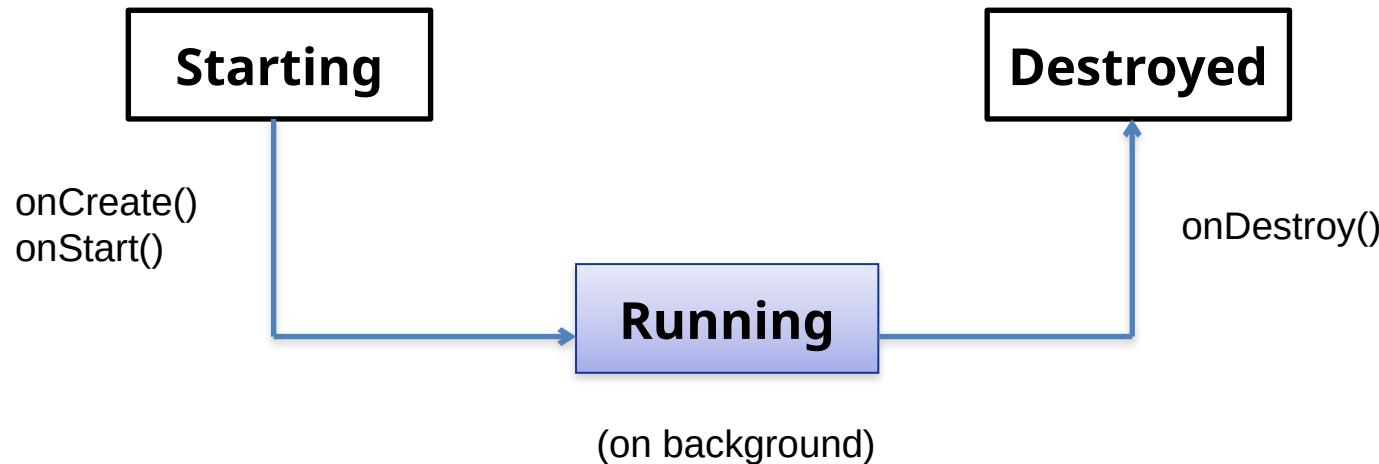
Android Components: Intents

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Implicit Intent** → The component (e.g. *Activity1*) specifies the type of the intent (e.g. “View a video”).



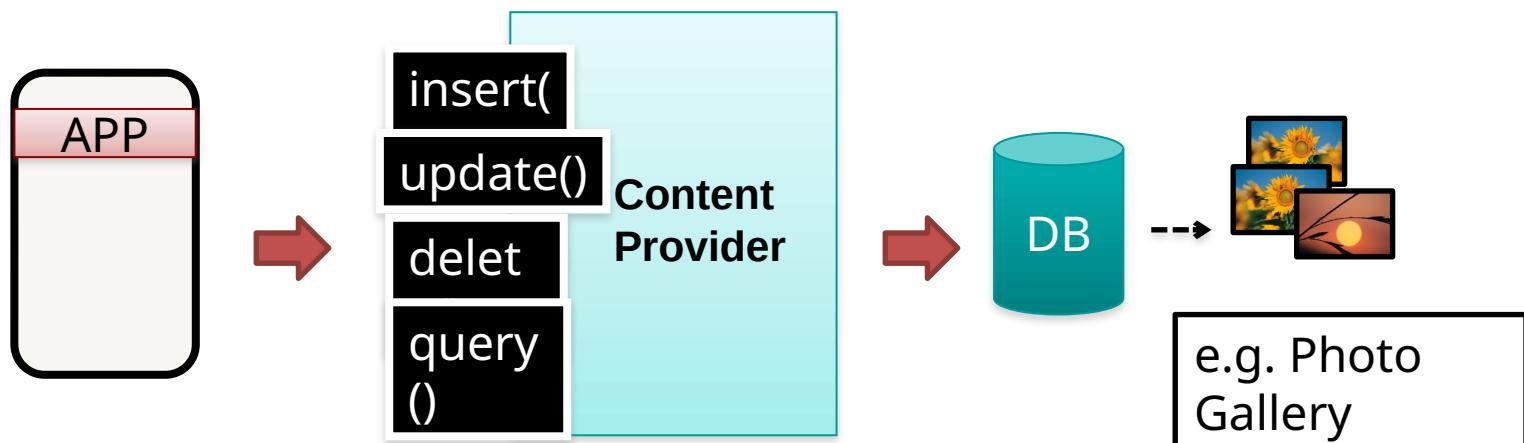
Android Components: Services

- **Services:** like Activities, but run in **background** and do not provide an user interface.
- Used for **non-interactive** tasks (e.g. networking).
- Service life-time composed of 3 states:



Android Components: Content Providers

- Each Android **application** has its own **private** set of data (managed through *files* or through *SQLite* database).
- **Content Providers**: Standard **interface** to *access and share data among different applications*.



Android Components: *System API*

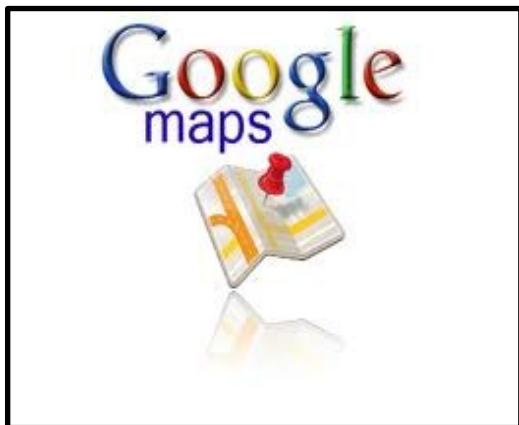
- Using the **components** described so far, Android applications can then leverage the system API ...

SOME EXAMPLEs ...

- *Telephony Manager* data access (call, SMS, etc)
- *Sensor management* (GPS, accelerometer, etc)
- *Network connectivity* (Wifi, bluetooth, NFC, etc)
- *Web surfing* (HTTP client, WebView, etc)
- *Storage management* (files, SQLite db, etc)
-

Android Components: Google API

- ... or easily interface with other **Google services**:



Android Application Security

- Android applications run with a distinct system identity (Linux user ID and group ID), in an **isolated** way.
- Applications must explicitly share resources and data. They do this by declaring the ***permissions*** they need for additional capabilities.
 - Applications statically **declare** the permissions they require.
 - User must **give his/her consensus** during the installation.

Case Study - Linux

- 1. Evolution of UNIX**
- 2. LINUX Operating System & Design goals**
- 3. Interfaces to Linux**
- 4. List of Linux Utility Programs**
- 5. Linux kernel**
- 6. Process in Linux and PID, UID, GID in Linux**
- 7. Process Management System Calls in Linux**
- 8. User space thread and kernel space thread**
- 9. Booting process in Linux**
- 10. Memory management in Linux**
- 11. Paging in Linux**
- 12. Network File System calls in Linux**

LINUS TORVALDS POSTS FAMOUS MESSAGE - "HELLO EVERYBODY OUT THERE..." - AND RELEASES FIRST LINUX CODE



SLACKWARE BECOMES FIRST WIDELY ADOPTED DISTRIBUTION



TECH GIANTS BEGIN ANNOUNCING PLATFORM SUPPORT FOR LINUX



IBM RUNS FAMOUS LINUX AD DURING THE SUPERBOWL



THE LINUX FOUNDATION IS FORMED TO PROMOTE, PROTECT AND STANDARDIZE LINUX. LINUS IS A FELLOW



LINUX TURNS 20 AND POWERS THE WORLD'S SUPERCOMPUTERS, STOCK EXCHANGES, PHONES, ATMS, HEALTHCARE RECORDS, SMART GRIDS, THE LIST GOES ON



1991

1992

1993

1996

1998

1999

2003

2005

2007

2010

2011



LINUS LICENSES LINUX UNDER THE GPL, AN IMPORTANT DECISION THAT WILL CONTRIBUTE TO ITS SUCCESS IN THE COMING YEARS



LINUS VISITS AQUARIUM, GETS BIT BY A PENGUIN AND CHOOSES IT AS LINUX MASCOT



RED HAT GOES PUBLIC



LINUS APPEARS ON THE COVER OF BUSINESSWEEK WITH A STORY THAT HAILS LINUX AS A BUSINESS SUCCESS

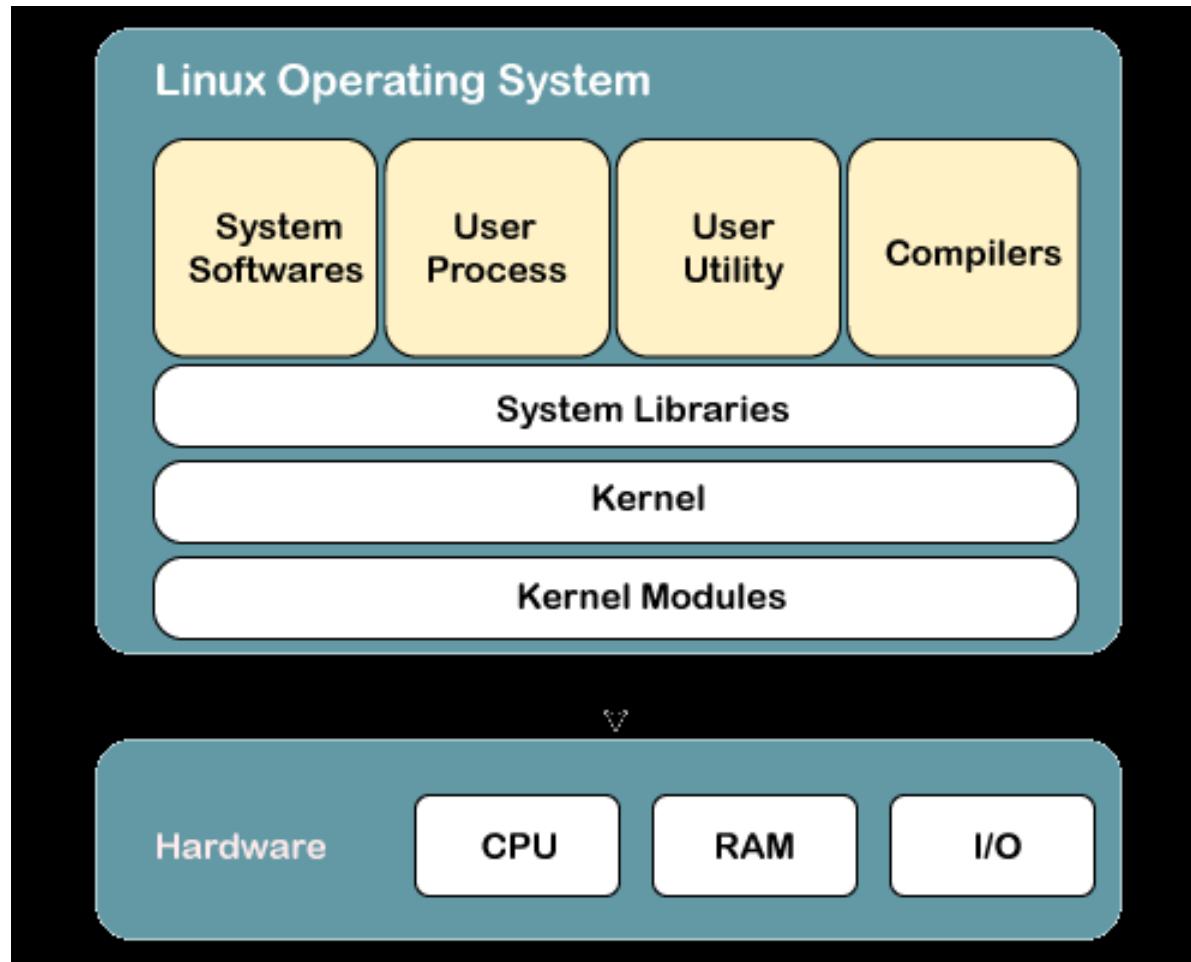


THE LINUX-BASED ANDROID OS OUTSHIPS ALL OTHER SMARTPHONE OSES IN THE U.S. AND CLIMBS TO DOMINANCE



THE LINUX FOUNDATION
<http://www.linuxfoundation.org/>

Linux architecture



Design goals of Linux

- Designed to handle **multiple processes and multiple users** at the same time
- Multi-user, multitasking system with a full set of UNIX-compatible tools.
- Speed, efficiency, and standardization.
- Designed to be compliant with the relevant POSIX documents.
- The Linux programming interface adheres to the **SVR4 UNIX semantics**.

List of Linux Utility Programs

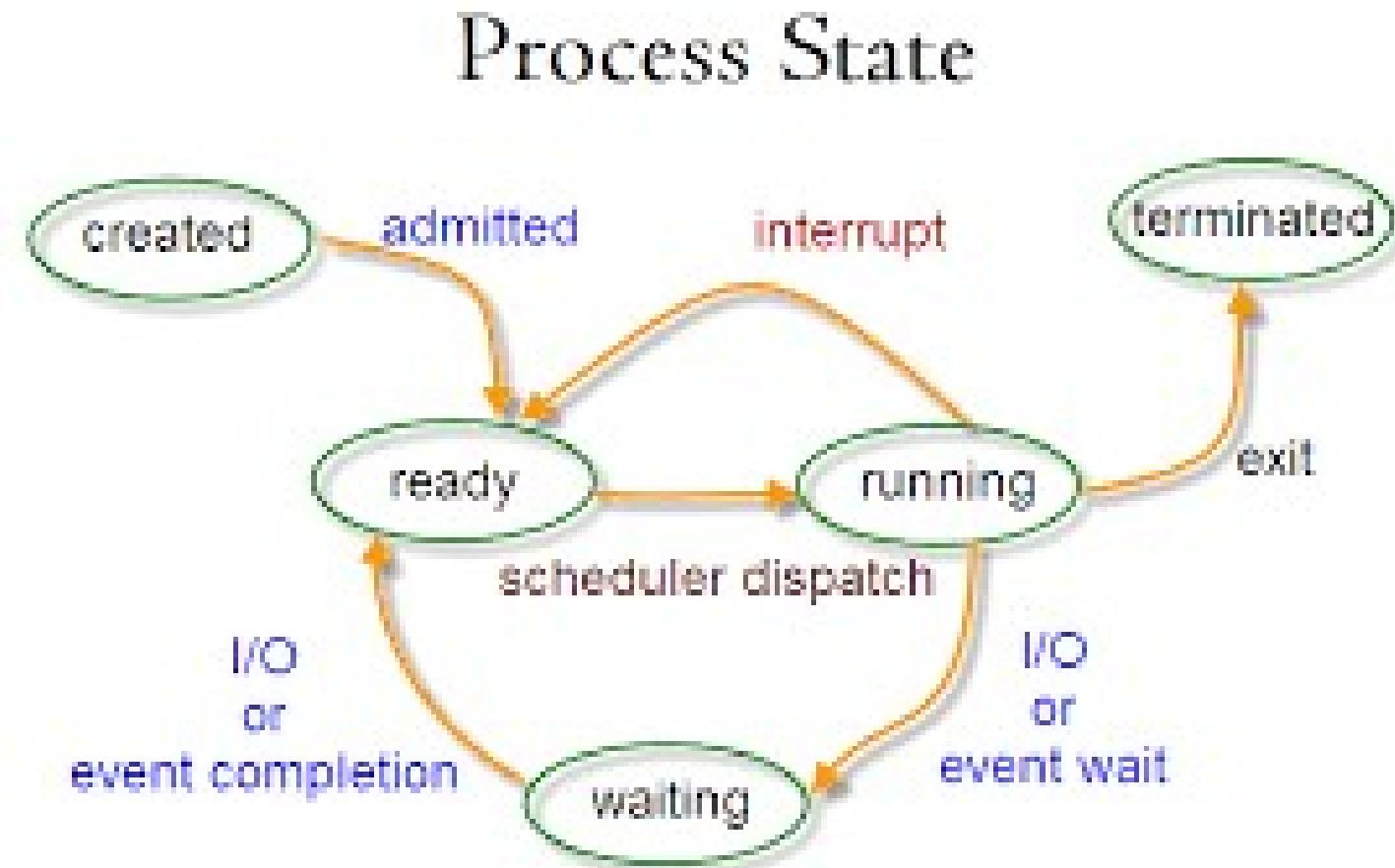
Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
ps	List running processes
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

Process in Linux and PID, UID, GID in Linux

- The fork system call creates an exact copy of the original process.
- The forking process is called the **parent process**. The new process is called the **child process**.
- Process Identifier is when each process has a unique identifier associated with it known as **process id**.
- **User and Group Identifiers (UID and GID)** are the identifiers associated with a processes of the user and group.

Signal	Cause
SIGABRT	Sent to abort a process and force a core dump
SIGALRM	The alarm clock has gone off
SIGFPE	A floating-point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written to a pipe which has no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

Linux Processes



Linux Processes

- State
- Running
- Waiting
- Stopped
- Zombie – dead processes (not removed)
- Scheduling Information
- Identifiers
- Inter-Process Communication
- Links
- Times and Timers
- Virtual memory
- Processor Specific Context
- File system

Process Management System Calls in Linux

Syscall	Description
<u>CLONE</u>	Create a child process
<u>FORK</u>	Create a child process
<u>VFORK</u>	Create a child process and block parent
<u>EXECVE</u>	Execute program
<u>EXECVEAT</u>	Execute program relative to a directory file descriptor
<u>EXIT</u>	Terminate the calling process
<u>EXIT_GROUP</u>	Terminate all threads in a process
<u>WAIT4</u>	Wait for process to change state
<u>WAITID</u>	Wait for process to change state

Syscall	Description
<u>GETPID</u>	Get process ID
<u>GETPPID</u>	Get parent process ID
<u>GETTID</u>	Get thread ID

Syscall	Description
<u>SETSID</u>	Set session ID
<u>GETSID</u>	Get session ID

Process Management System Calls in Linux

Syscall	Description
<u>SETPGID</u>	Set process group ID
<u>GETPGID</u>	Get process group ID
<u>GETPGRP</u>	Get the process group ID of the calling process

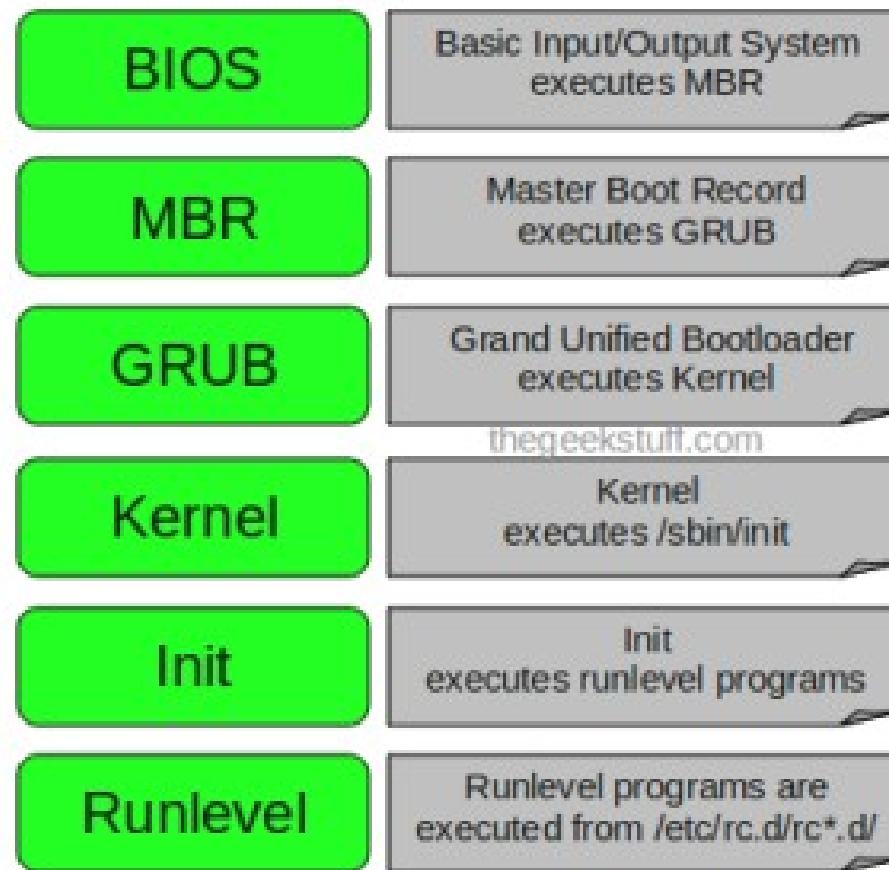
Syscall	Description
<u>SETUID</u>	Set real user ID
<u>GETUID</u>	Get real user ID
<u>SETGID</u>	Set real group ID
<u>GETGID</u>	Get real group ID
<u>SETRESUID</u>	Set real, effective and saved user IDs
<u>GETRESUID</u>	Get real, effective and saved user IDs
<u>SETRESGID</u>	Set real, effective and saved group IDs
<u>GETRESGID</u>	Get real, effective and saved group IDs
<u>SETREUID</u>	Set real and/or effective user ID
<u>SETREGID</u>	Set real and/or effective group ID
<u>SETFSUID</u>	Set user ID used for file system checks
<u>SETFSGID</u>	Set group ID used for file system checks
<u>GETEUID</u>	Get effective user ID

<u>GETEGID</u>	Get effective group ID
<u>SETGROUPS</u>	Set list of supplementary group IDs
<u>GETGROUPS</u>	Get list of supplementary group IDs

User and Kernel Space Thread

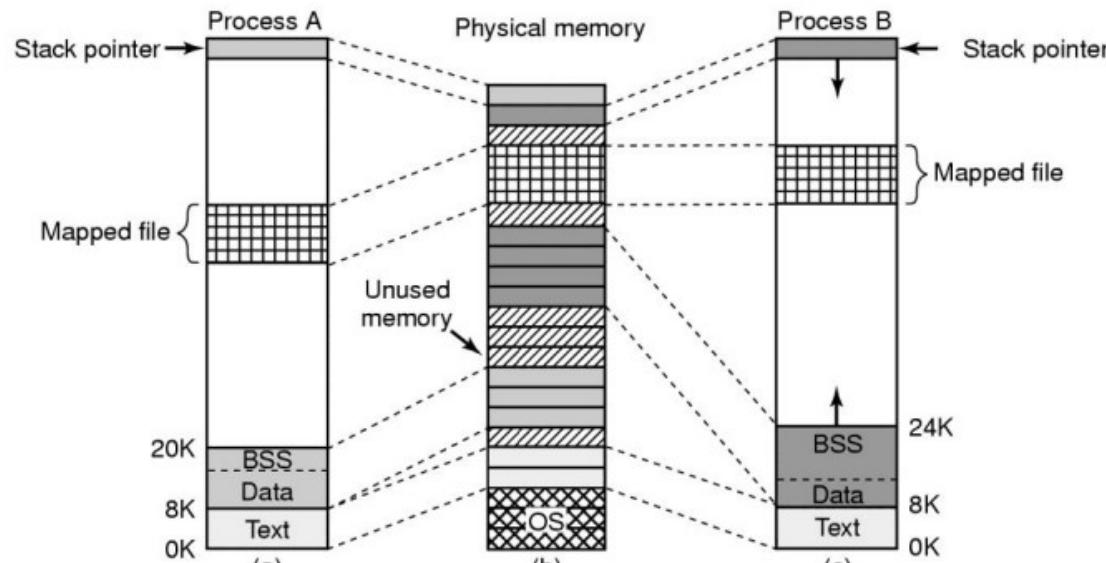
- **User Level Threads** – User managed threads.
- **Kernel Level Threads** – Operating System managed threads acting on kernel, an operating system core.
- **User-Space Threads:** cooperative multitasking, user threads typically can switch faster than kernel threads.
- **Kernel-space threads** often are implemented in the kernel using several tables.

Booting process in Linux



Memory Management in Linux

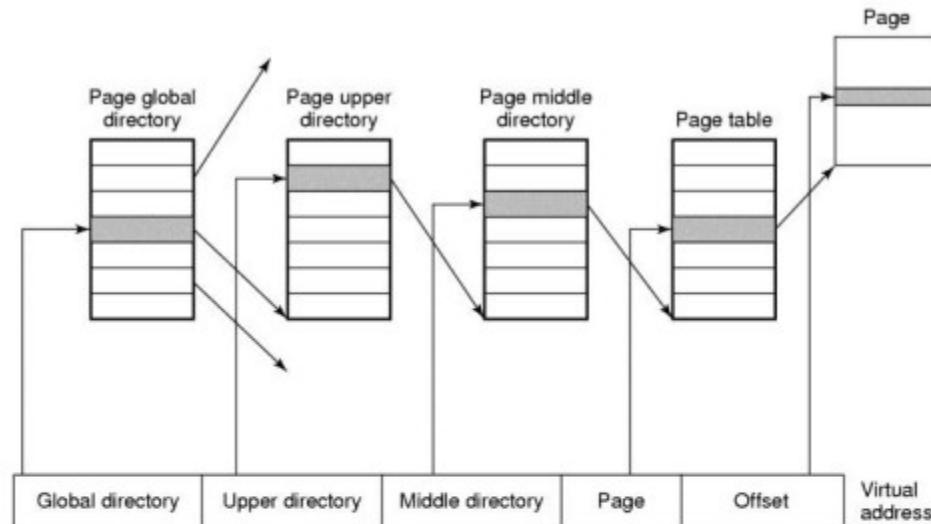
- The **memory management subsystem** is one of the most important parts of the operating system.
- Virtual memory** makes the system appear to have more memory than it actually has by sharing it between competing processes as they need it.



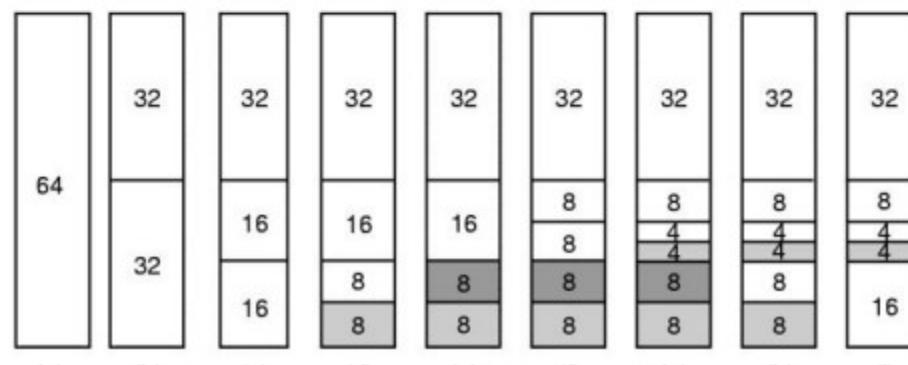
Paging in Linux

- The technique of only loading virtual pages into memory as they are accessed is known as demand paging.
- There are three kernel variables which control the paging operation: `minfree`, `desfree`, `lotsfree`.
- A `demand paging system` is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance.

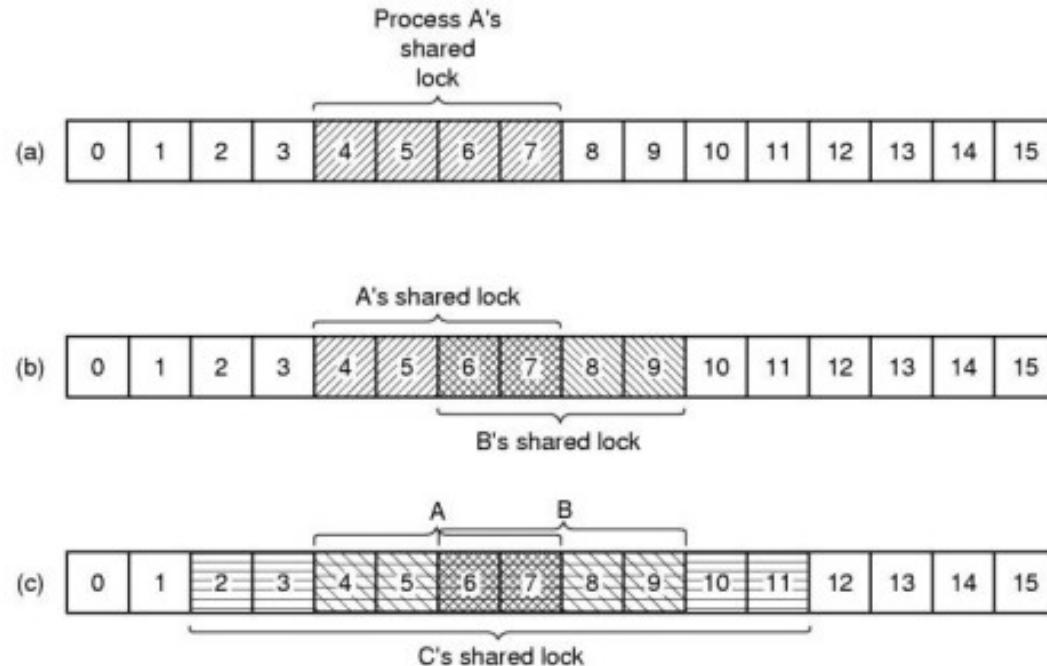
Physical Memory Management



Memory Allocation Mechanisms



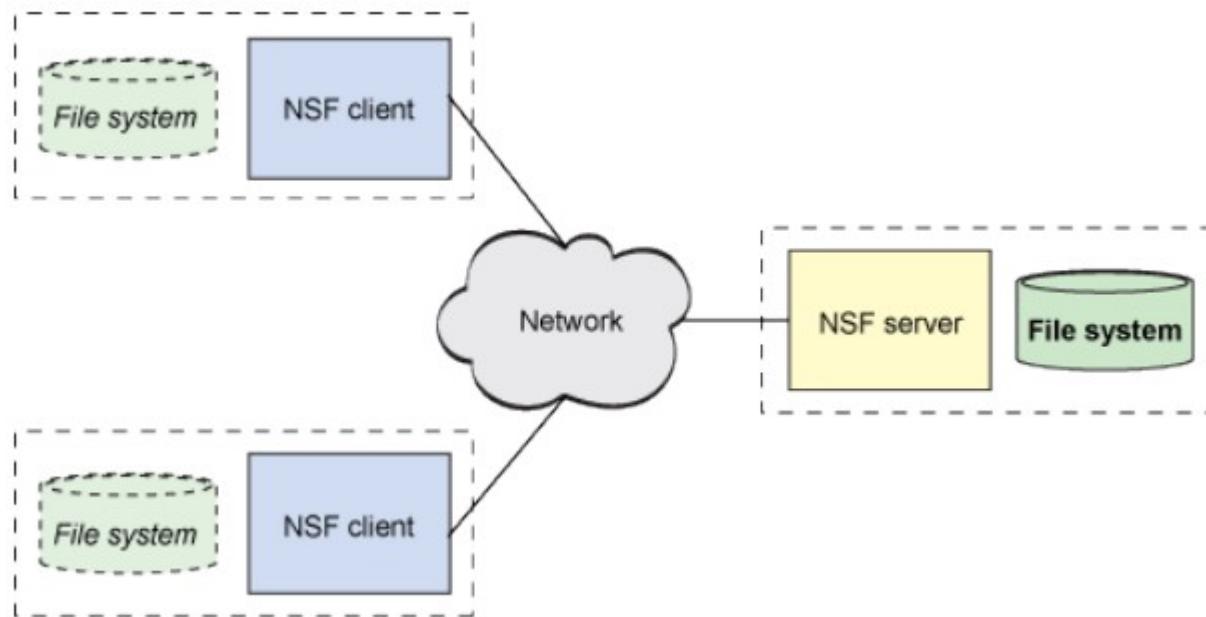
The Linux File System



(a) A file with one lock. (b) Addition of a second lock. (c) A third lock.

Network File System (NFS) calls in Linux

- The Network File System (NFS) is a way of mounting **Linux discs/directories** over a network.
- An NFS server can export one or more directories that can then be mounted on a remote Linux machine.



Android vs Linux

The “Not-So-Obvious” Differences

	Android	Linux
Type of Software	Operating System	Kernel
Processor architecture support	ARM A architecture	x86 and x64 architectures
Communication Channel	Optimized for Mobile networks	Optimized for Ethernet and Wi-Fi
Need for creation	To make Mobile Devices smart and as a competitor for iOS	For the spirit of having an open source software and for research and development purposes
How the OS makes money	By pushing google products and through commissions from apps, movies, music, books, etc. that are sold on the Google Play Store	Most distros are community maintained and are non-profitable organizations, main source of income include tech support in the enterprise.

The Similarities

	Android	Linux
Kernel	Linux Based Kernel	Linux Kernel
License Type	Open Source, Apache License	Open Source, GPL V2.0 License
Price	Free	Free



The Obvious Differences

	Android	Linux
Main Device Type	Smart phones	Personal Computers and Servers
Main use cases	Content Consumption and Communication	Content creation and more serious works
Other Application areas	Limited to use in Mobiles, Tablets, TVs, etc.	Can be ported to virtually any device like laptops, desktops, servers, routers, phones, TVs, washing machines, refrigerators, microwave ovens, industrial applications, etc.
Ease of use	Very Easy	Has a little bit of a learning curve based on the Distro you choose!
Maintained by	Google	Lots of Companies and communities maintain the various Linux distros

The Technical Differences

	Android	Linux
User Space	Android Run time (custom java runtime) and Core Libraries	Mainly composed of GNU and X11 server
Kernel Optimization	Optimized for power consumption	A balanced trade-off between performance and power consumption
Shell interface	Usually locked up, need to root the device to get access to the shell interface	Available out of the box as shell is a particularly important part of Linux Distros!
Base for the Graphical UI	Google designed Surface Flinger on top of OpenGL libraries	Most distros use X11 X server based Graphical User Interface.

The Technical Differences

BIOS/EFI	Not present on ARM devices	Standard for all x86/x64 motherboards
C library	Lightweight Bionic libc	GNU C library

The Technical Differences

Main Architectural elements	Just the Kernel, HAL and a non-GNU user space and its own desktop environment	Kernel, hardware abstraction layer (HAL) (drivers), GNU User space (apps) and UI layers (Desktop environment)
Hardware Drivers	Almost all of the drivers are proprietary	Majority are open source
Kernel's connection with peripherals	Usually I2c and SPI, so the kernel needs to be aware that a particular peripheral exists beforehand, and drivers have to be preloaded	Uses PCI, PCIe, USB etc. These protocols are self-identifying and allow the kernel to probe these devices and load necessary drivers
Bootloader	Simple bootloader, that does the minimum necessary and hands the control over to the kernel	Standardized bootloaders like Grub2 that can be customized



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN SESSION 7

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Balamurali Shankar
WILP & Department of CS & IS



BITS Pilani
Pilani Campus

Operating systems – Memory Interface



Last Session

List of Topic Title

Hardware Abstractions (Contd...)

Case Study 1: Linux on Intel processor

Case Study 2: Android on a mobile chipset

Memory Interface

Today's topic

- Levels of memory management
- The memory management design problem
- Solution to the memory management design problem
- Dynamic memory allocation

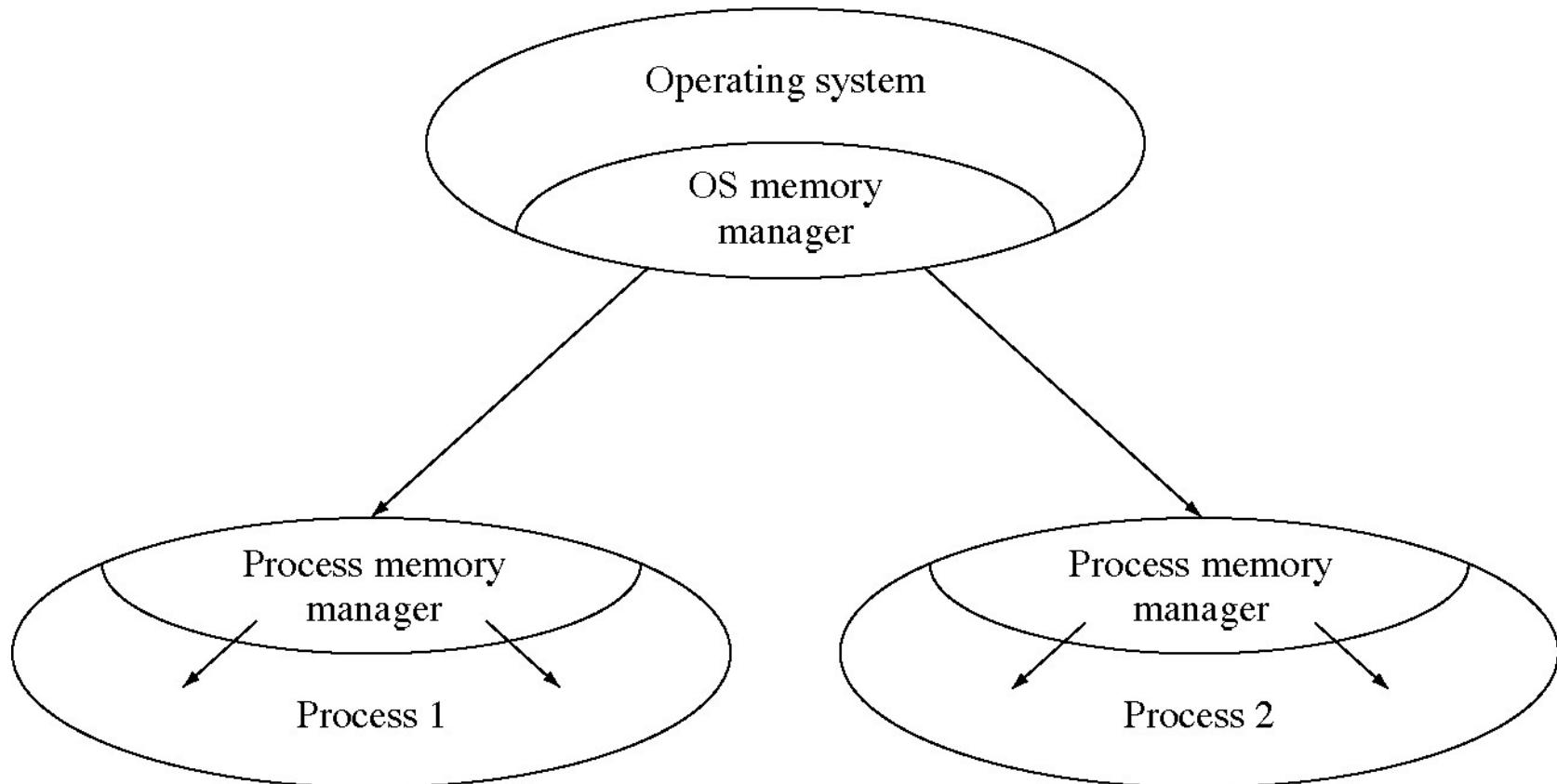


BITS Pilani
Pilani Campus

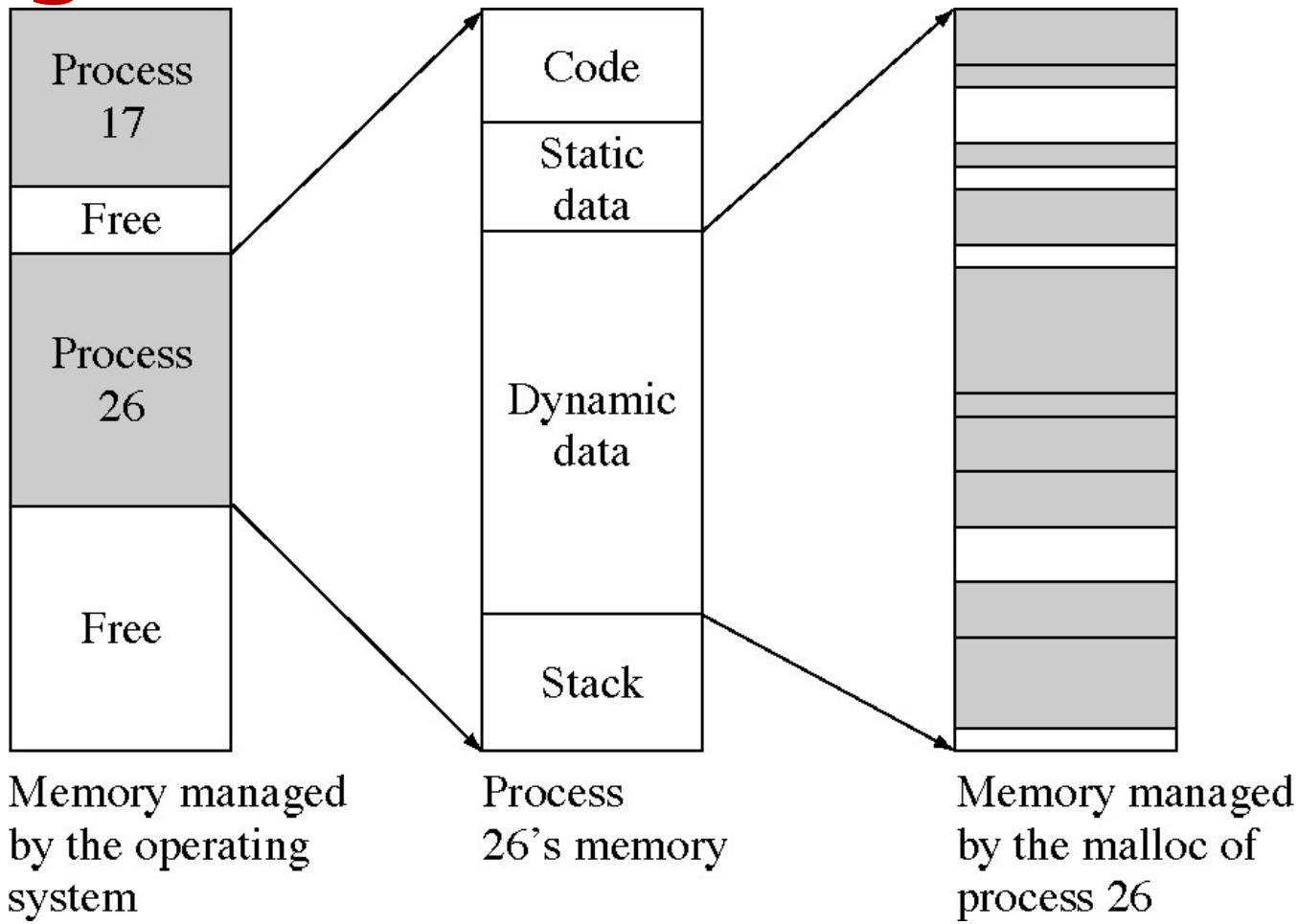
Levels of Memory management



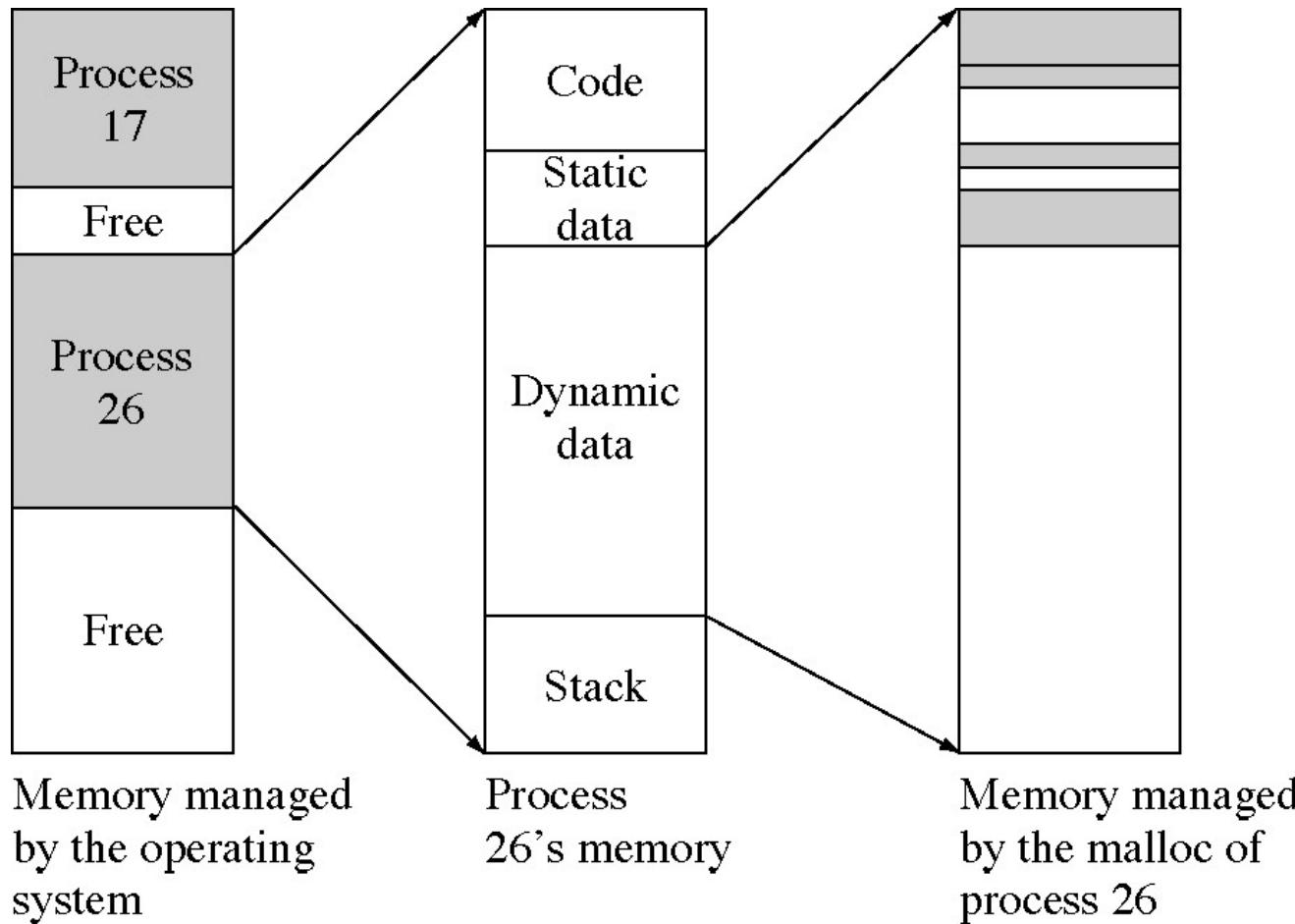
Two levels of memory management



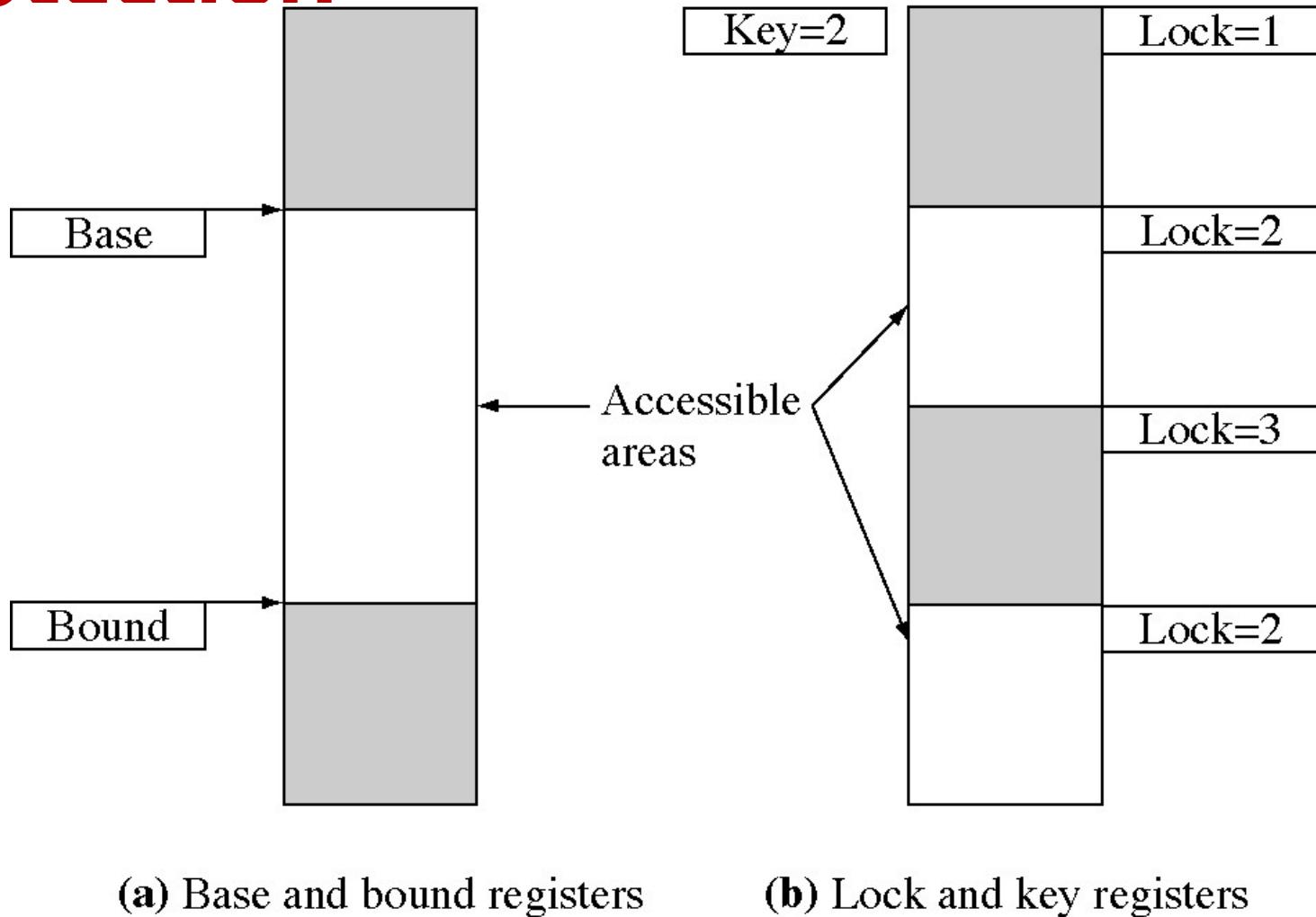
Two levels of memory management



Free memory at the malloc level, but not at the OS level



Two forms of memory protection





BITS Pilani
Pilani Campus

Memory management design problem

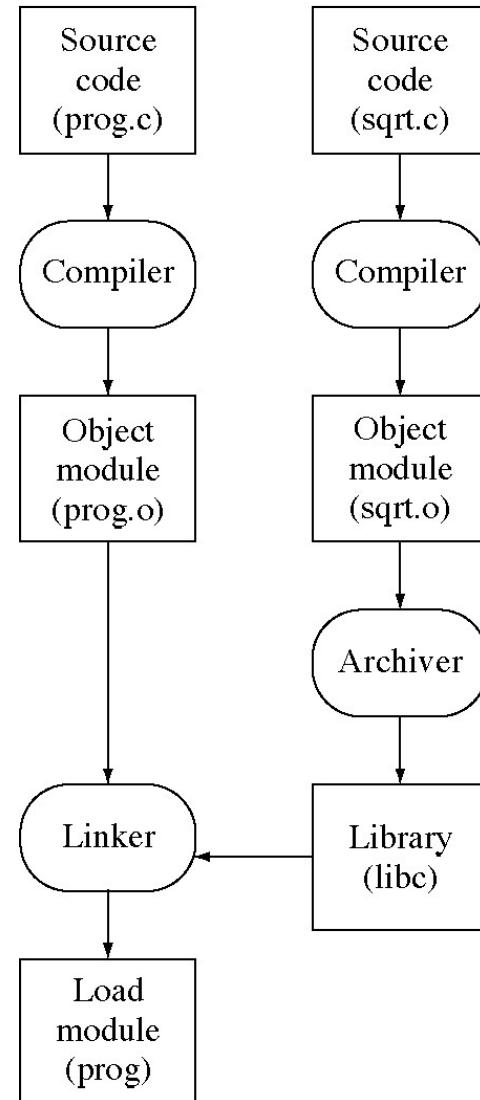


Memory Management Requirements

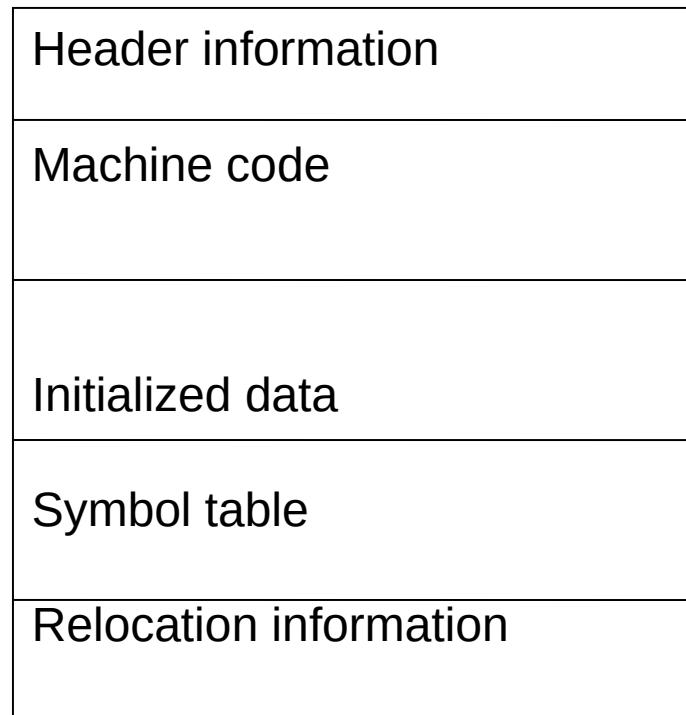


- Five requirements
 - Relocation
 - Protection
 - Sharing
 - Logical organization
 - Physical organization

Creating a load module



Object module format



Sample C++ program



```
• #include <iostream.h>
  #include <math.h>
  float arr[100];
  int size = 100;
  void main(int argc, char **argv) {
    int i; float sum = 0;
    for(I=0; I<size; ++i) {
      cin >> arr[i];//or ">>"(cin, arr[i])
      arr[i] = sqrt(arr[i]);
      sum += arr[i];
    }
    cout << sum;// or "<<"(cout, sum)
  }
```

Linker function

- Combine the object modules into a load module
- Relocate the object modules as they are being loaded
- Link the object modules together as they are being loaded
- Search libraries for external references not defined in the object modules

Linker algorithm (1 of 3)



- 1. Initialize by creating an empty load module and an empty symbol table
- 2. Read the next object module or library name from the command line

Linker algorithm (2 of 3)



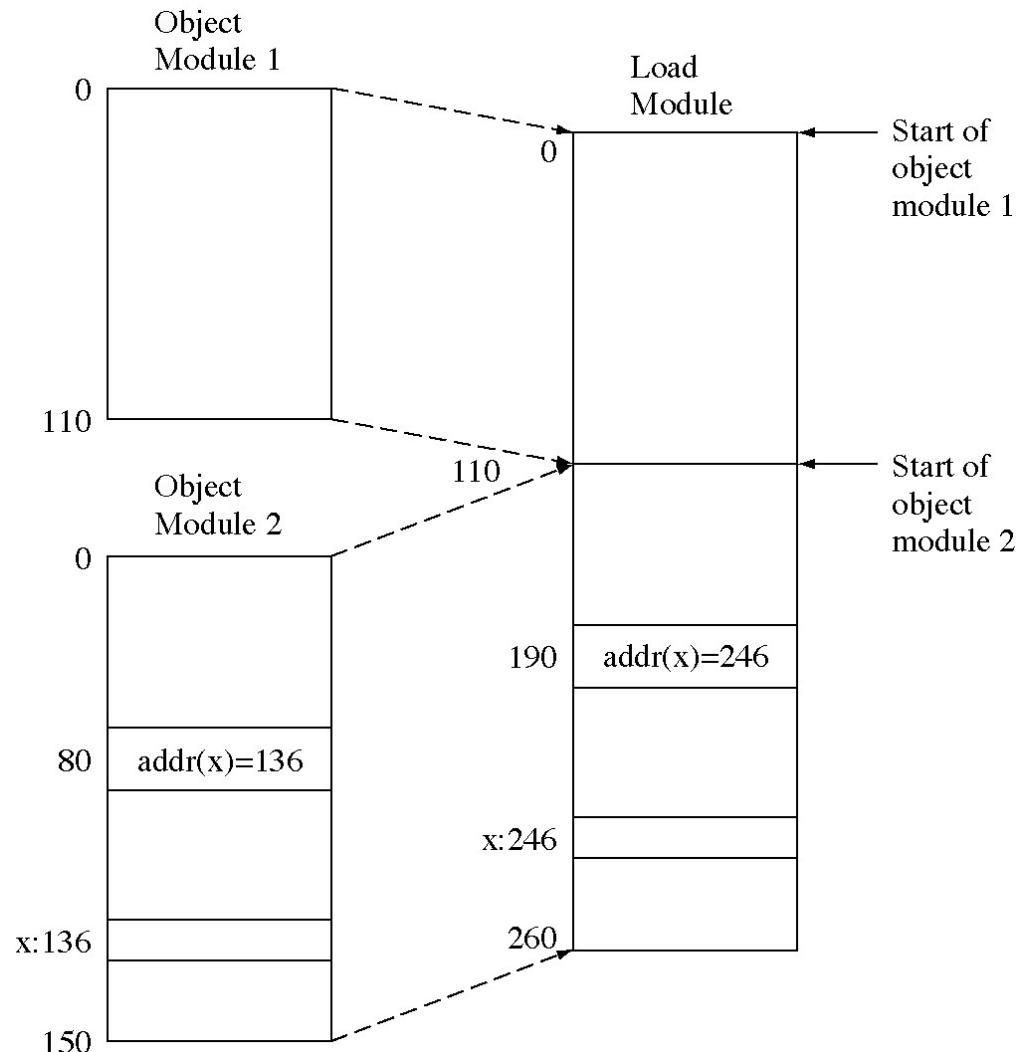
- 3. If it is an object module then:
 - a. Insert it into the load module
 - b. Relocate it and its symbols
 - c. merge its symbol table into the global symbol table
 - d. For each undefined external reference in the object module's symbol table:
 - (1) If the symbol is already in the global symbol table then copy the value to the object module.
 - (2) If not then insert it (as undefined) into the global symbol table and make a note to fix up the symbol late
 - e. For each defined symbol in the object module, fix up all previous references to the symbol (in object modules loaded earlier)

Linker algorithm (3 of 3)

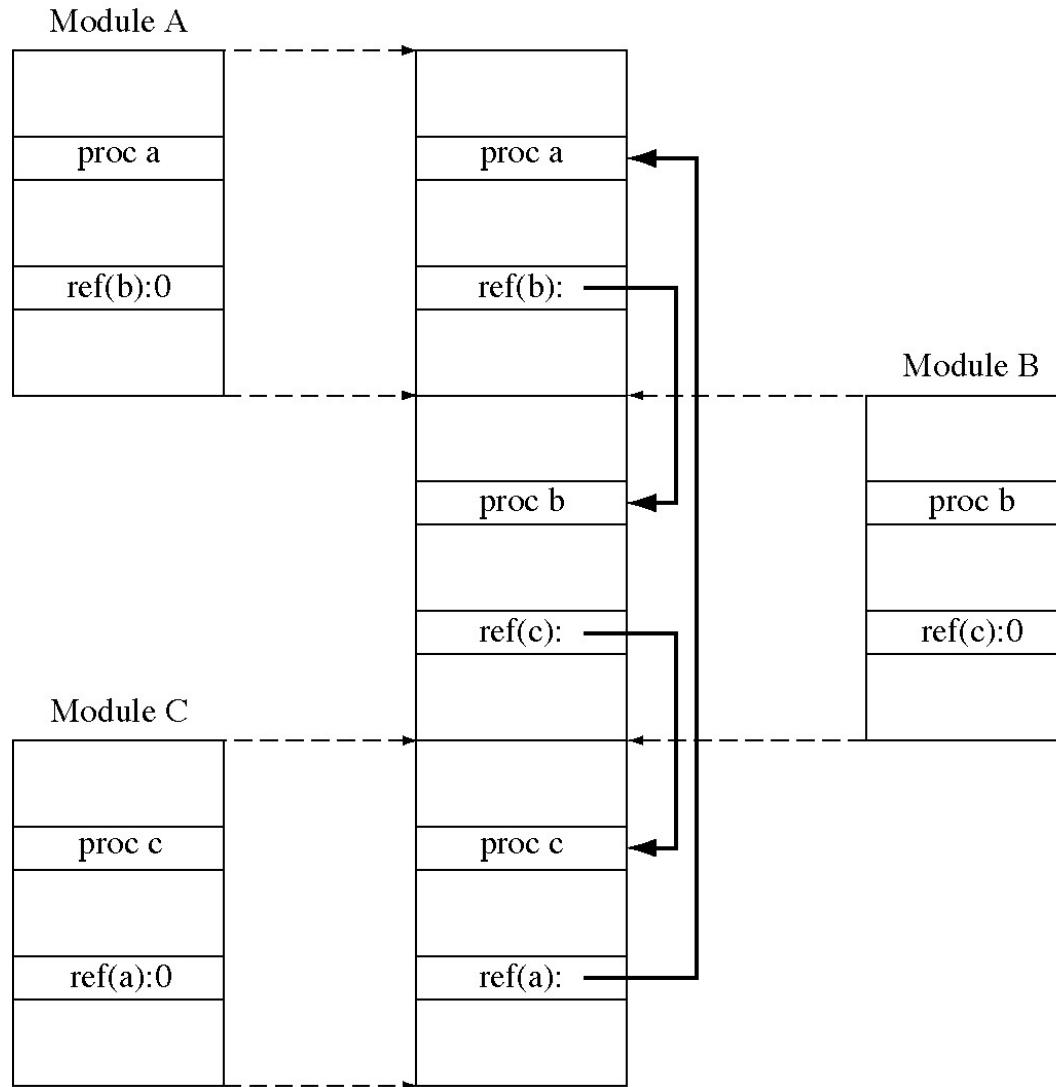


- 4. If it is a library then:
 - a. Find each undefined symbol in the global symbol table
 - b. See if the symbol is defined in a module in this library
 - c. If so, then load the object module as described in step 3.
- 5. Go back to step 2.

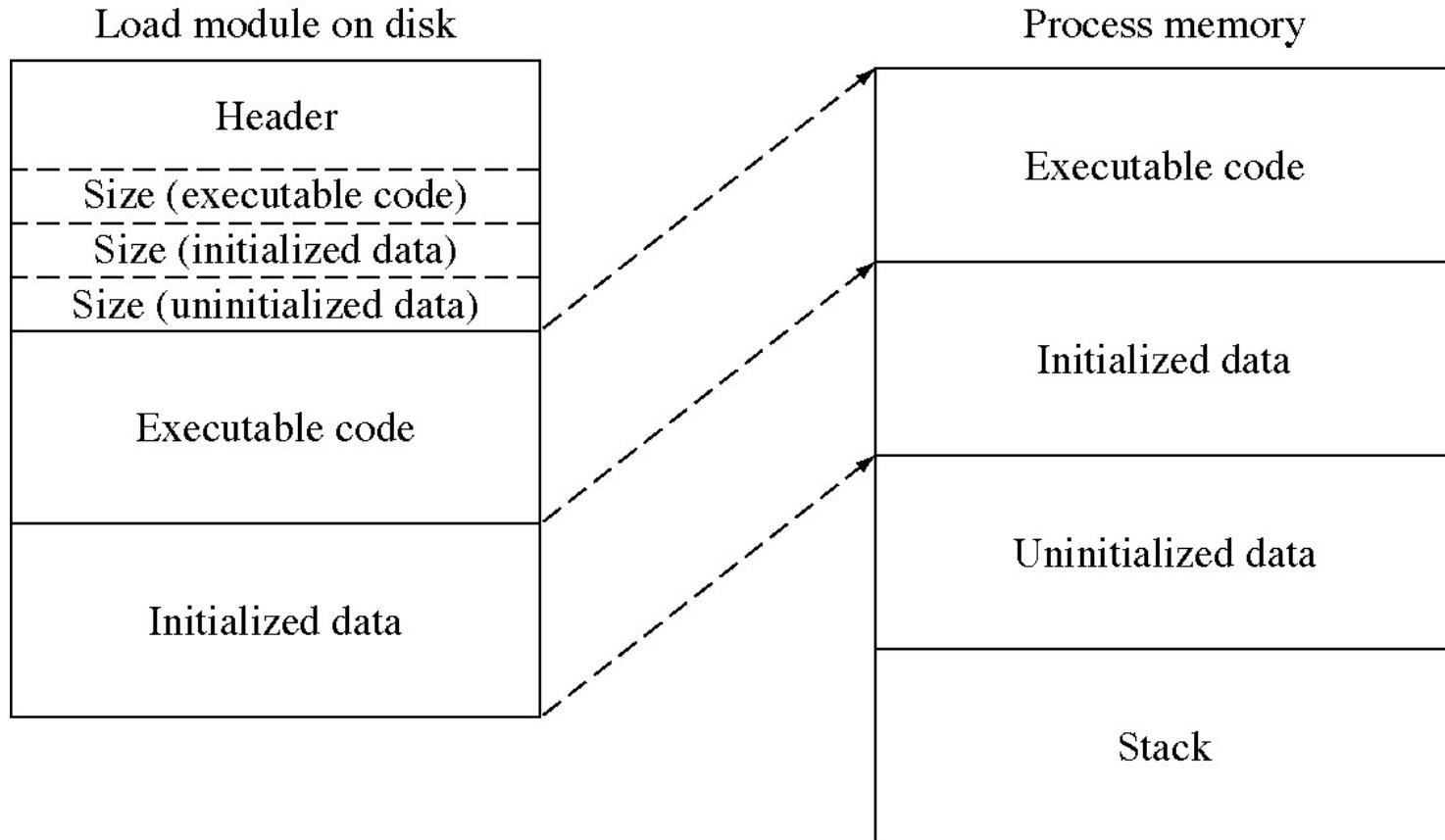
Relocation



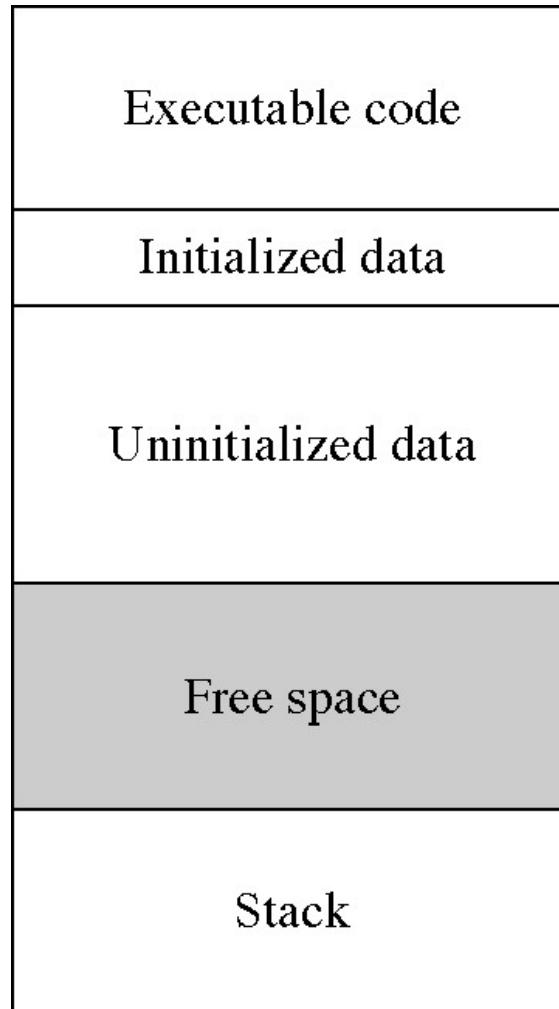
Linking



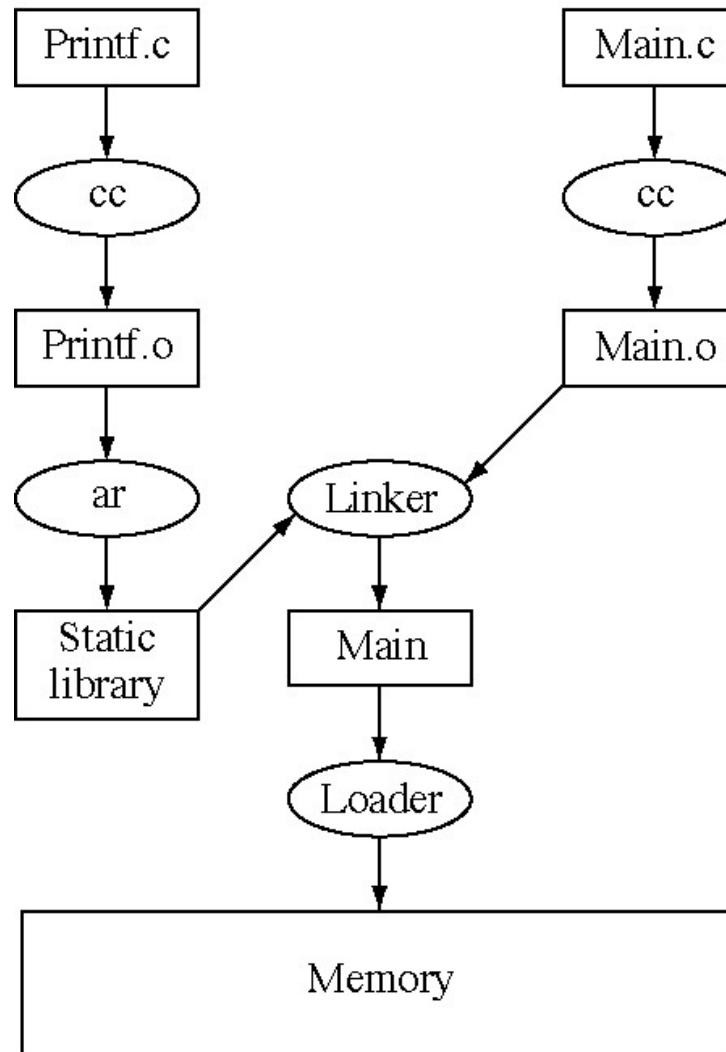
Loading a program into memory



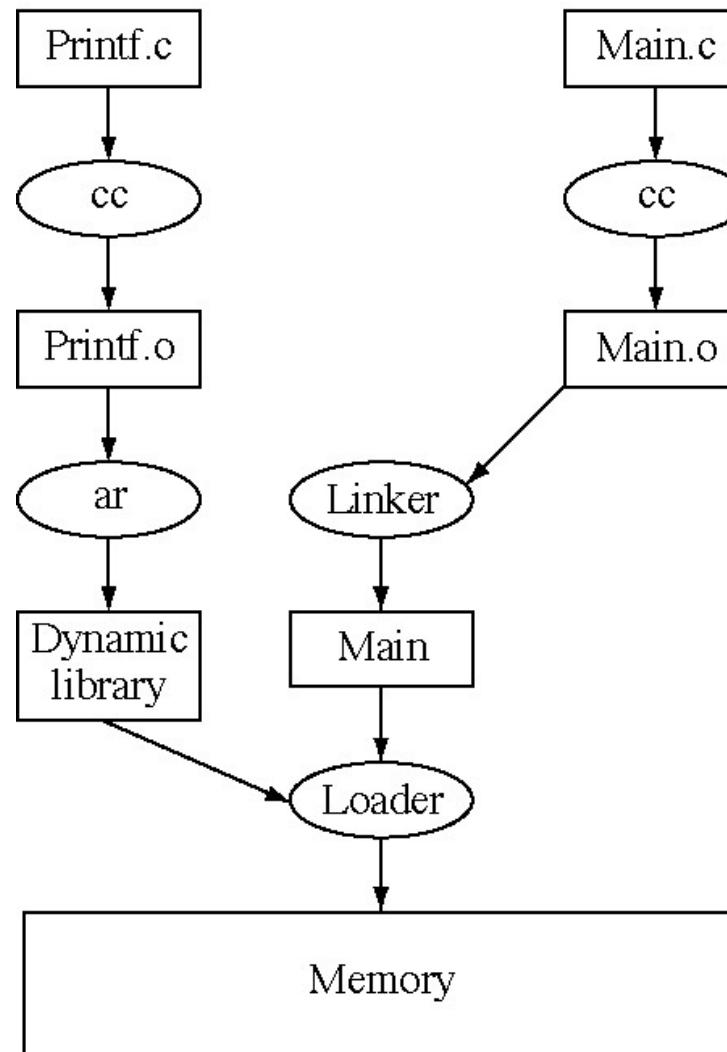
Memory areas in a running process



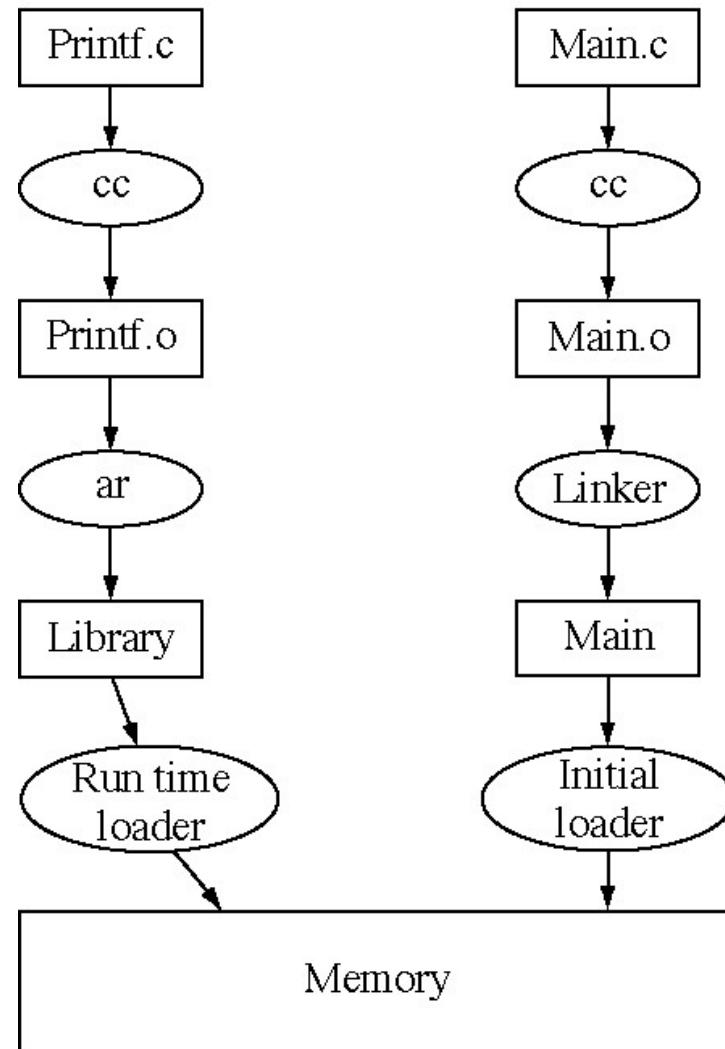
Normal linking and loading



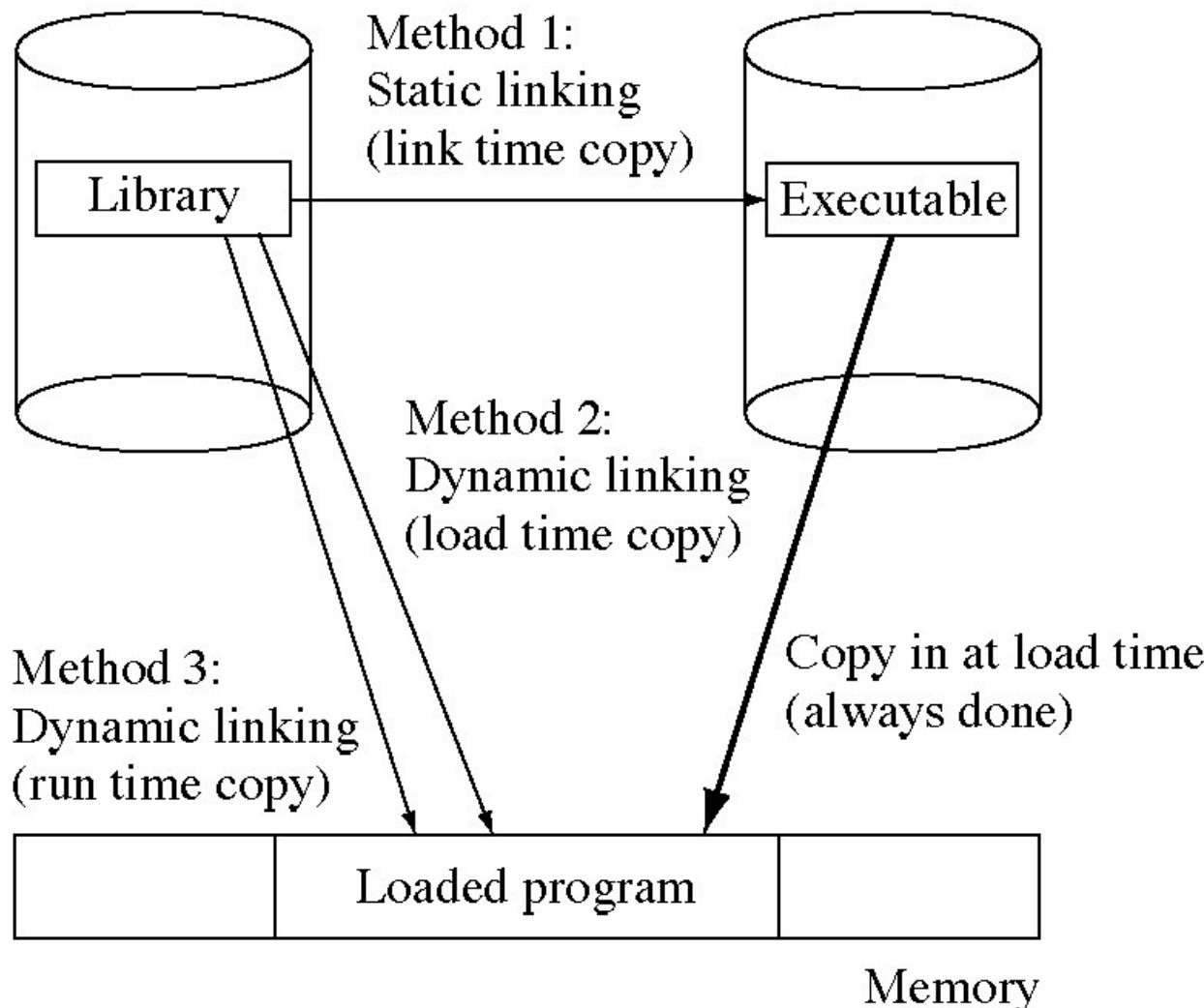
Load-time dynamic linking



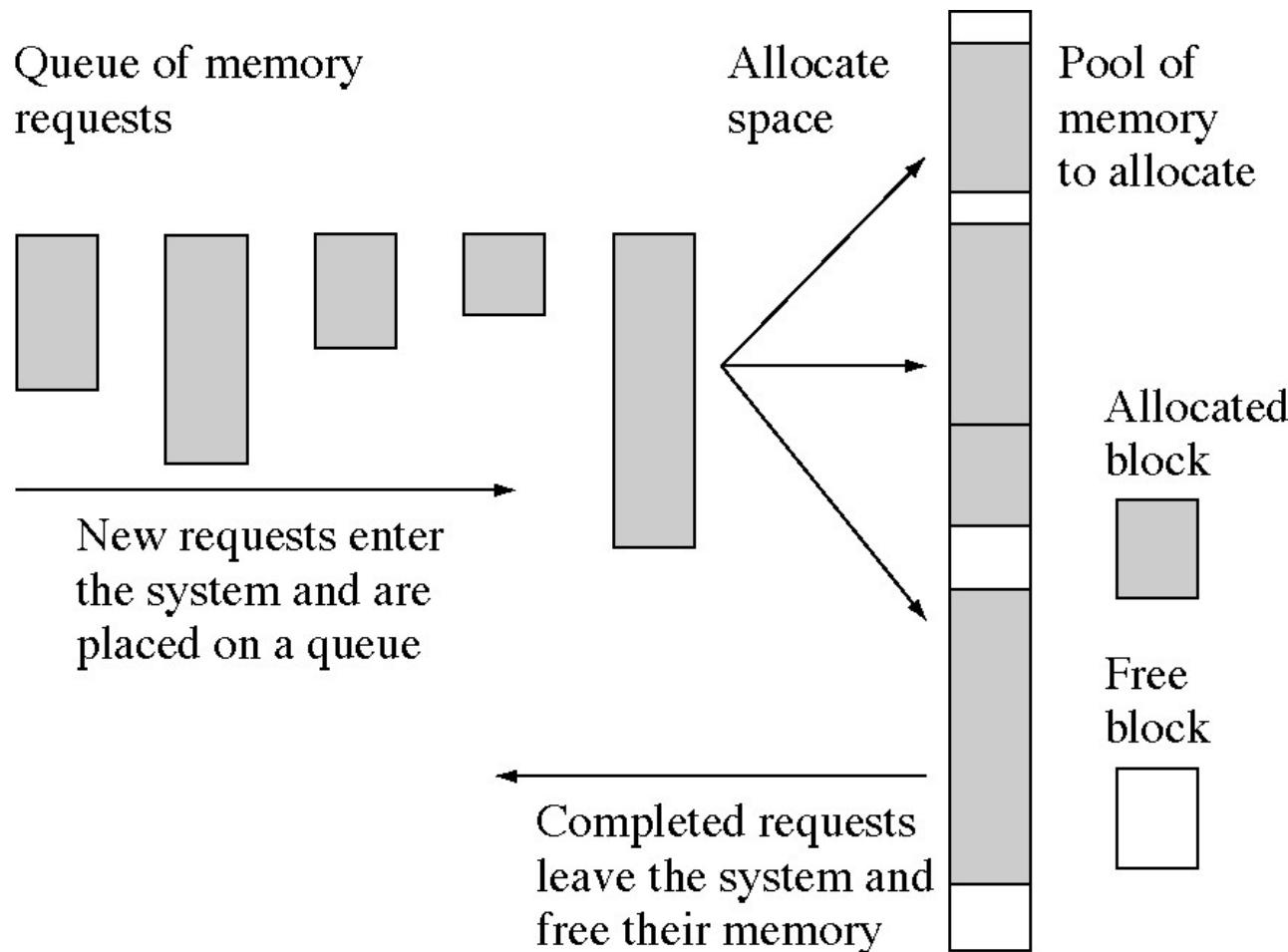
Run-time dynamic linking



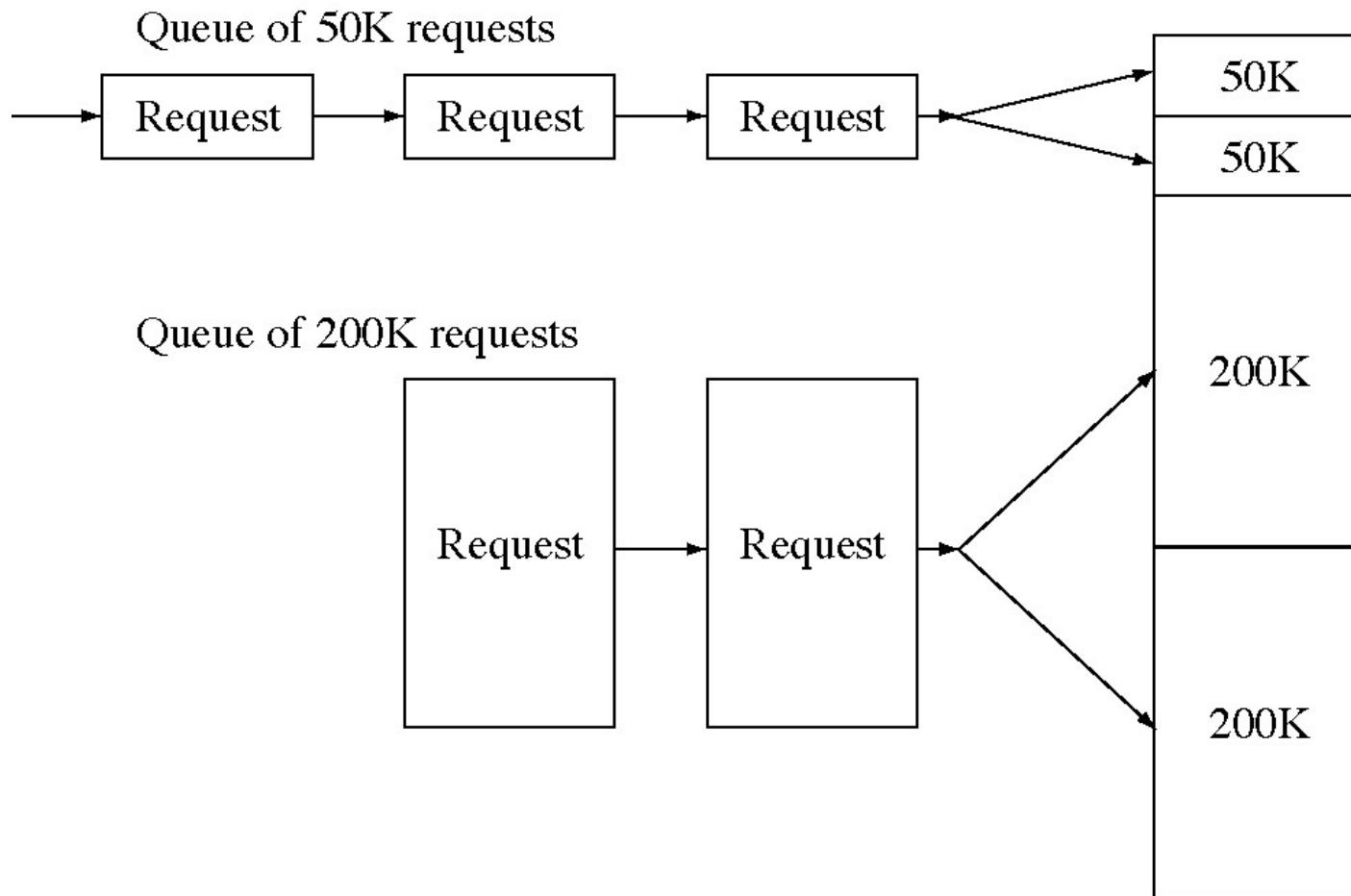
Static and dynamic linking



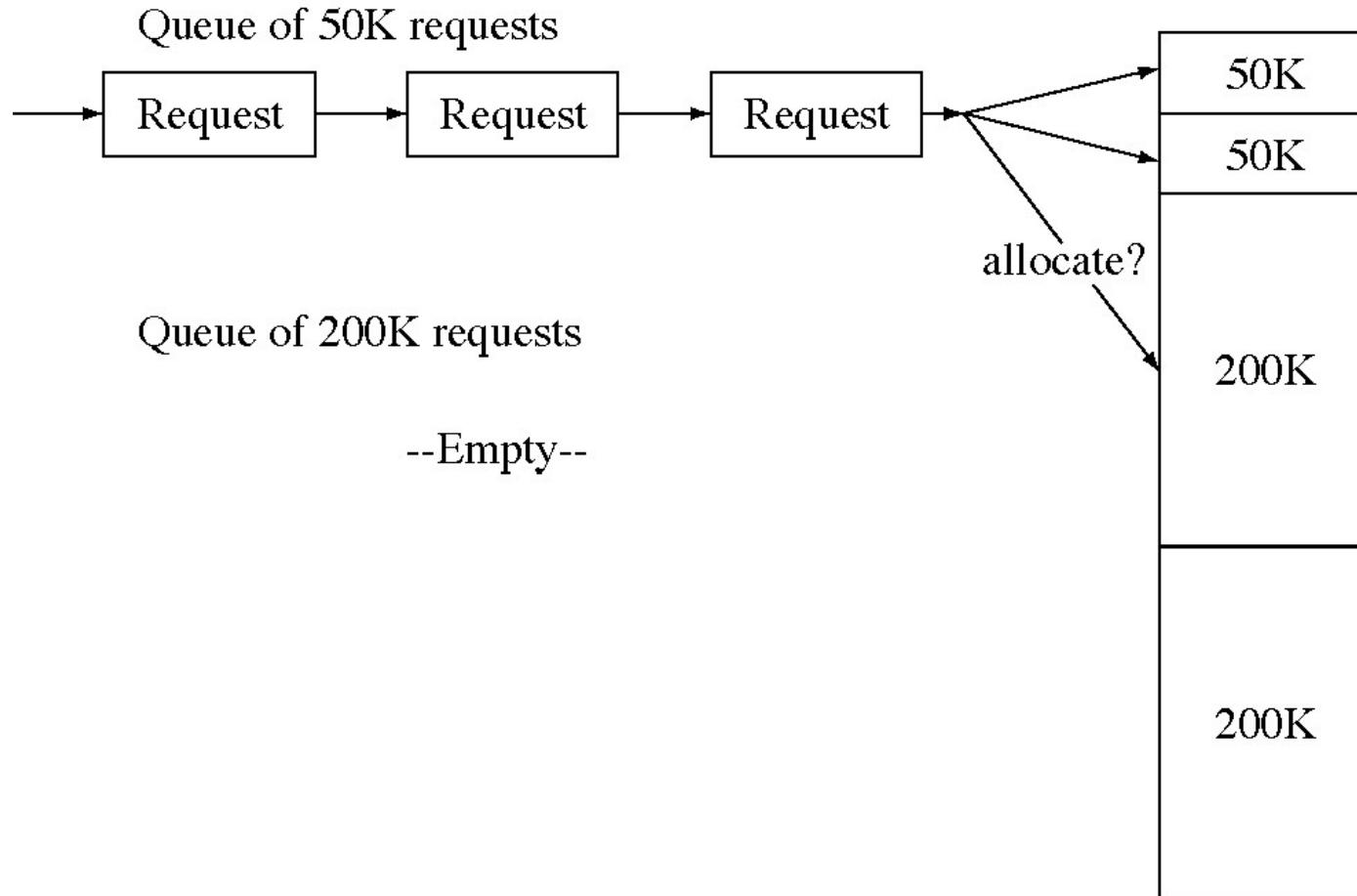
Memory allocation problem



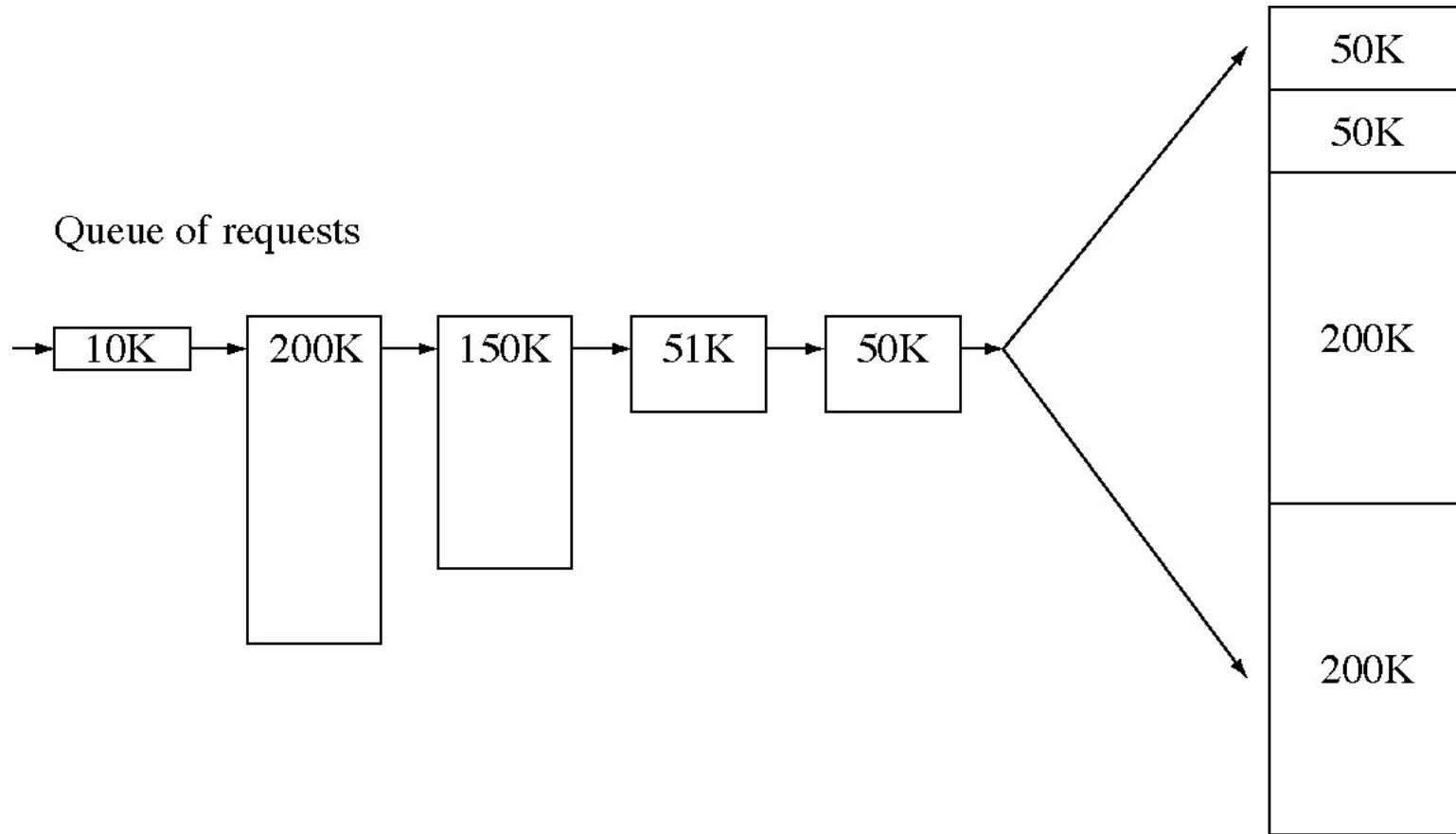
Queue for each block size



Allocate a large block to a small request?

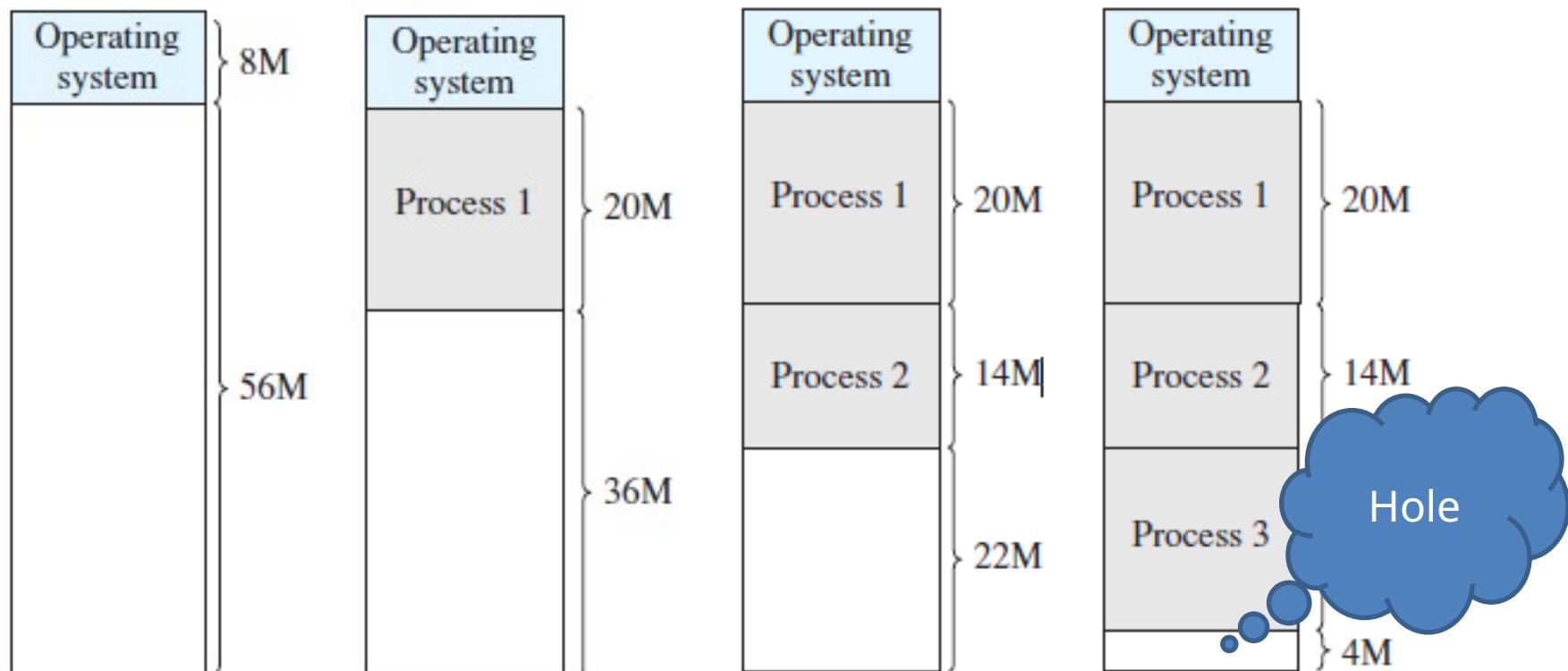


Variably-sized memory requests

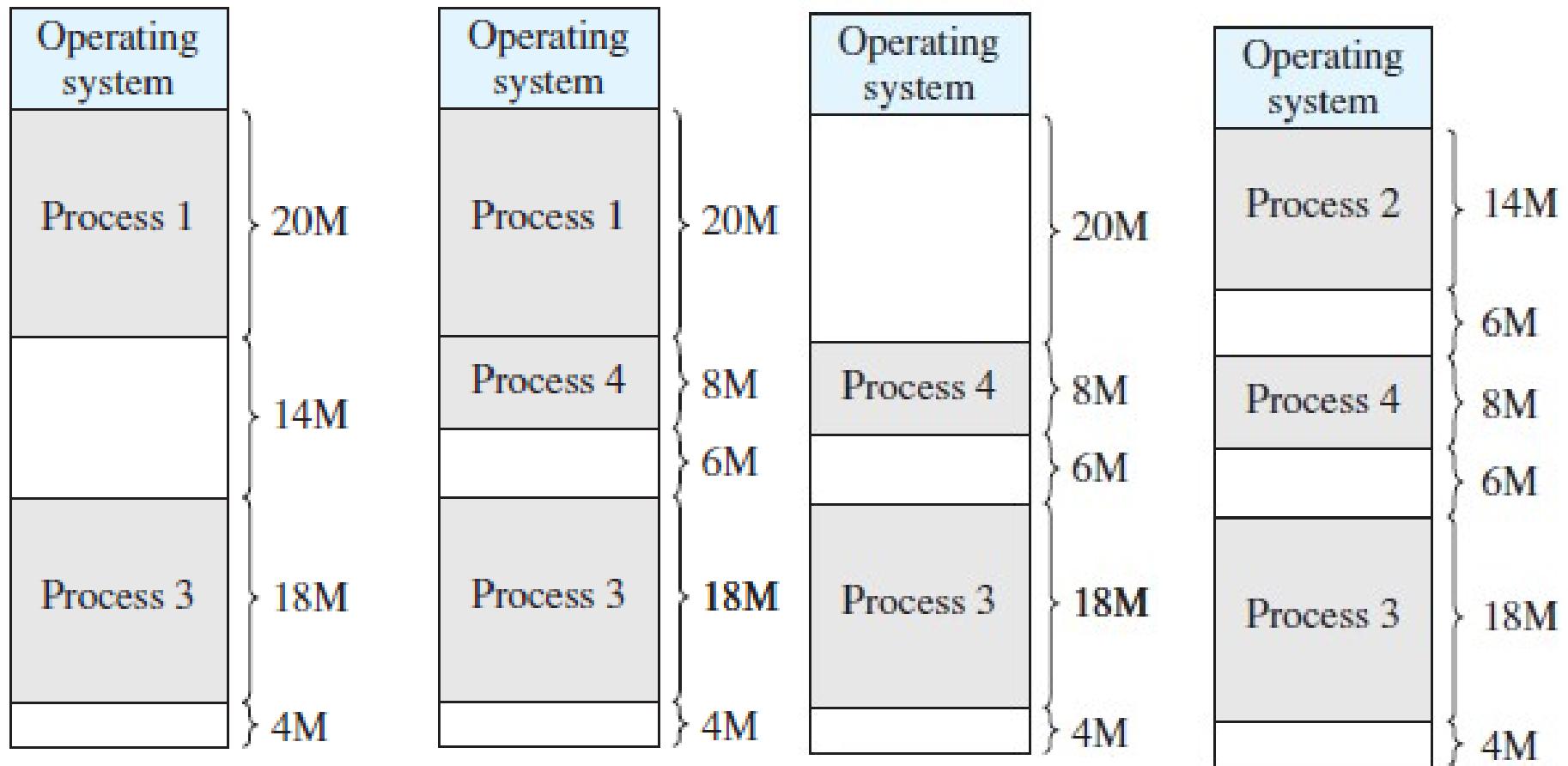


Dynamic Partitioning or Variable partition allocation

- The partitions are of variable length and number.
- Exact memory allocation



Contd...





BITS Pilani
Pilani Campus

Solution to Memory management design problem & Dynamic memory allocation

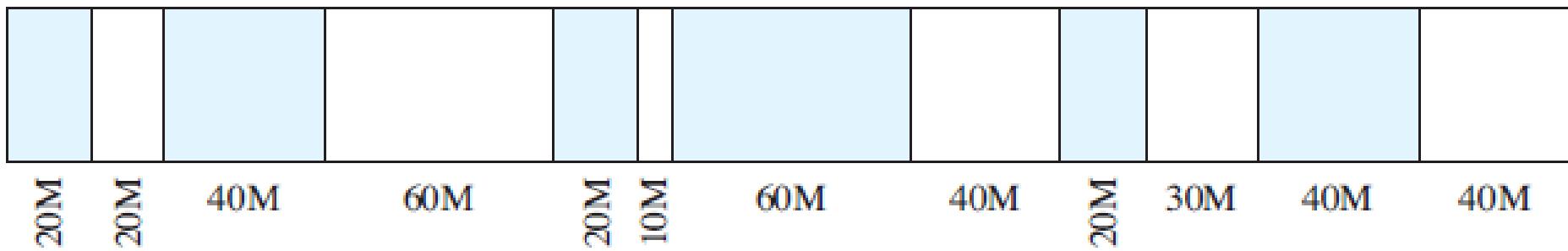


Dynamic Storage-Allocation

- Four schemes:
 - **First-fit**: Allocate the *first* hole that is big enough.
 - **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
 - **Next-fit**: begins to scan memory from the location of the last placement, and chooses the next available block that is large enough.
 - **Worst-fit**: Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole.
 - Which one is better?
-

Problem ...

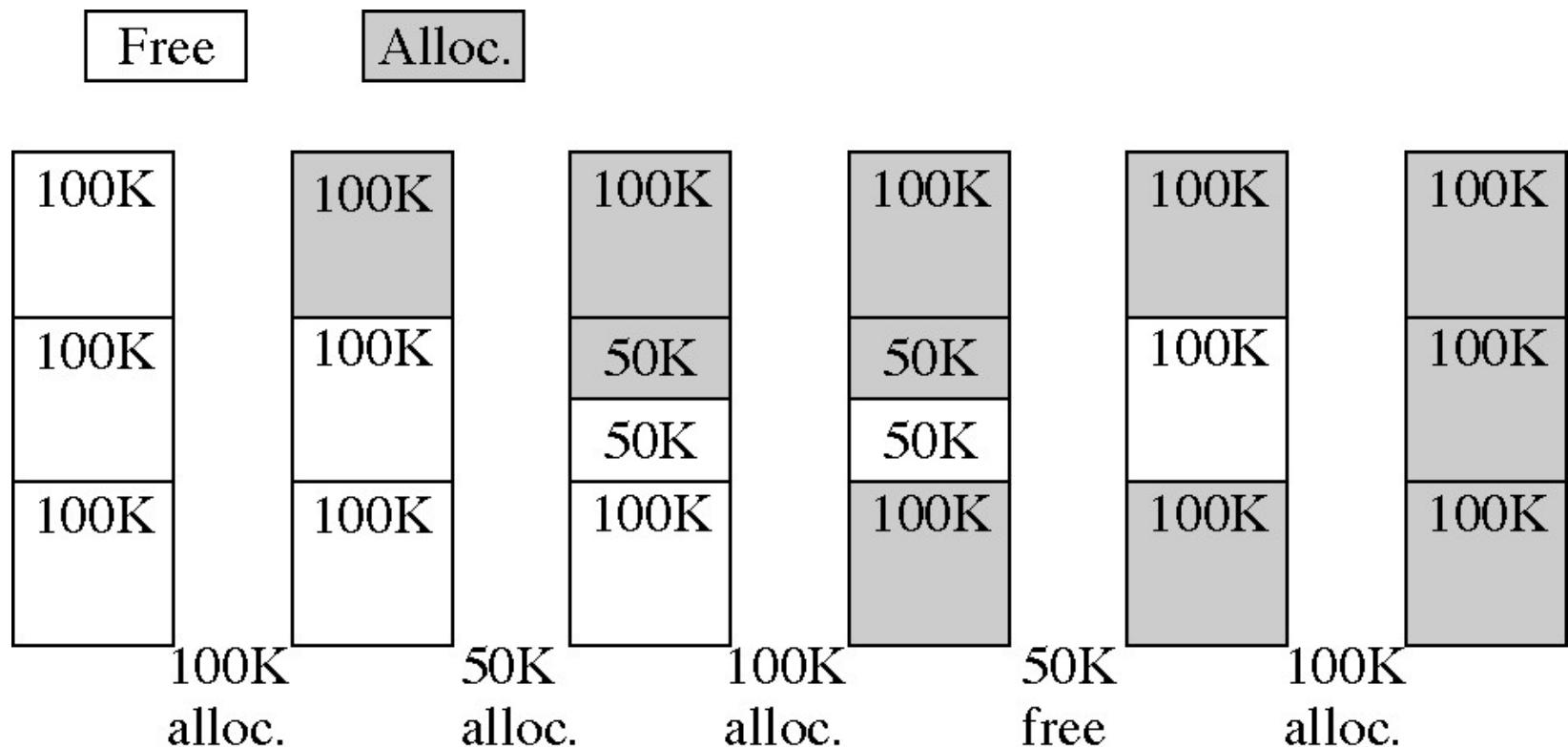
A dynamic partitioning scheme is being used, and the following is the memory configuration at a given point in time:



The shaded areas are allocated blocks; the white areas are free blocks. The next three memory requests are for 40M, 20M, and 10M. Indicate the starting address for each of the three blocks using the following placement algorithms:

- First-fit
- Best-fit
- Next-fit. Assume the most recently added block is at the beginning of memory.
- Worst-fit

The buddy system



Problem

Consider a memory block of 16K. Perform the following:

Allocate (A: 3.5K)

Allocate (B: 1.2K)

Allocate (C: 1.3K)

Allocate (D: 1.9K)

Allocate (E: 3.2K)

Free (C)

Free (B)

Allocate (F: 1.6K)

Allocate (G: 1.8K)

Consider a memory block of 16K. Perform the following:

Allocate (A: 3.5K)	Free (C)
Allocate (B: 1.2K)	Free (B)
Allocate (C: 1.3K)	Allocate (F: 1.6K)
Allocate (D: 1.9K)	Allocate (G: 1.8K)
Allocate (E: 3.2K)	

- 16K Memory Block

16K

- Allocate (A: 3.5K)



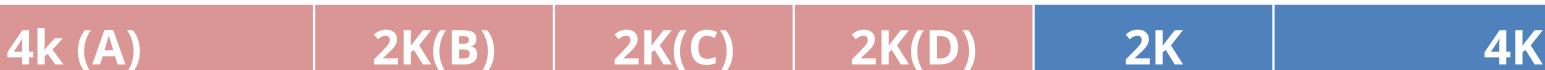
- Allocate (B: 1.2K)



- Allocate (C: 1.3K)



- Allocate (D: 1.9K)



Consider a memory block of 16K. Perform the following:

Allocate (A: 3.5K)	Free (C)
Allocate (B: 1.2K)	Free (B)
Allocate (C: 1.3K)	Allocate (F: 1.6K)
Allocate (D: 1.9K)	Allocate (G: 1.8K)
Allocate (E: 3.2K)	

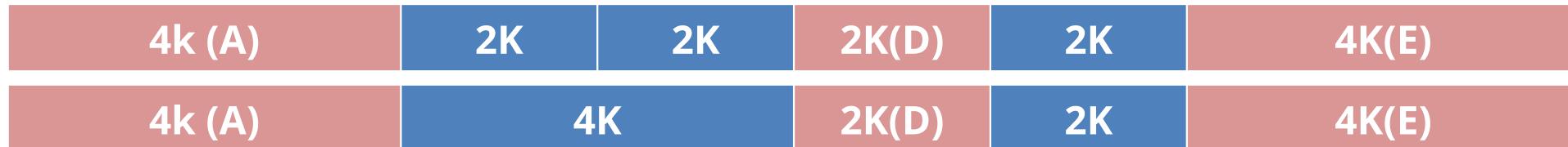
- Allocate (E: 3.2K)



- Free (C)



- Free (B)



- Allocate (F: 1.6K)



- Allocate (G: 1.8K)



A system has 1024 KB of memory, and it uses the Buddy System for memory allocation. The following memory allocation and deallocation requests occur:

Process A requests 200 KB of memory.

Process B requests 350 KB of memory.

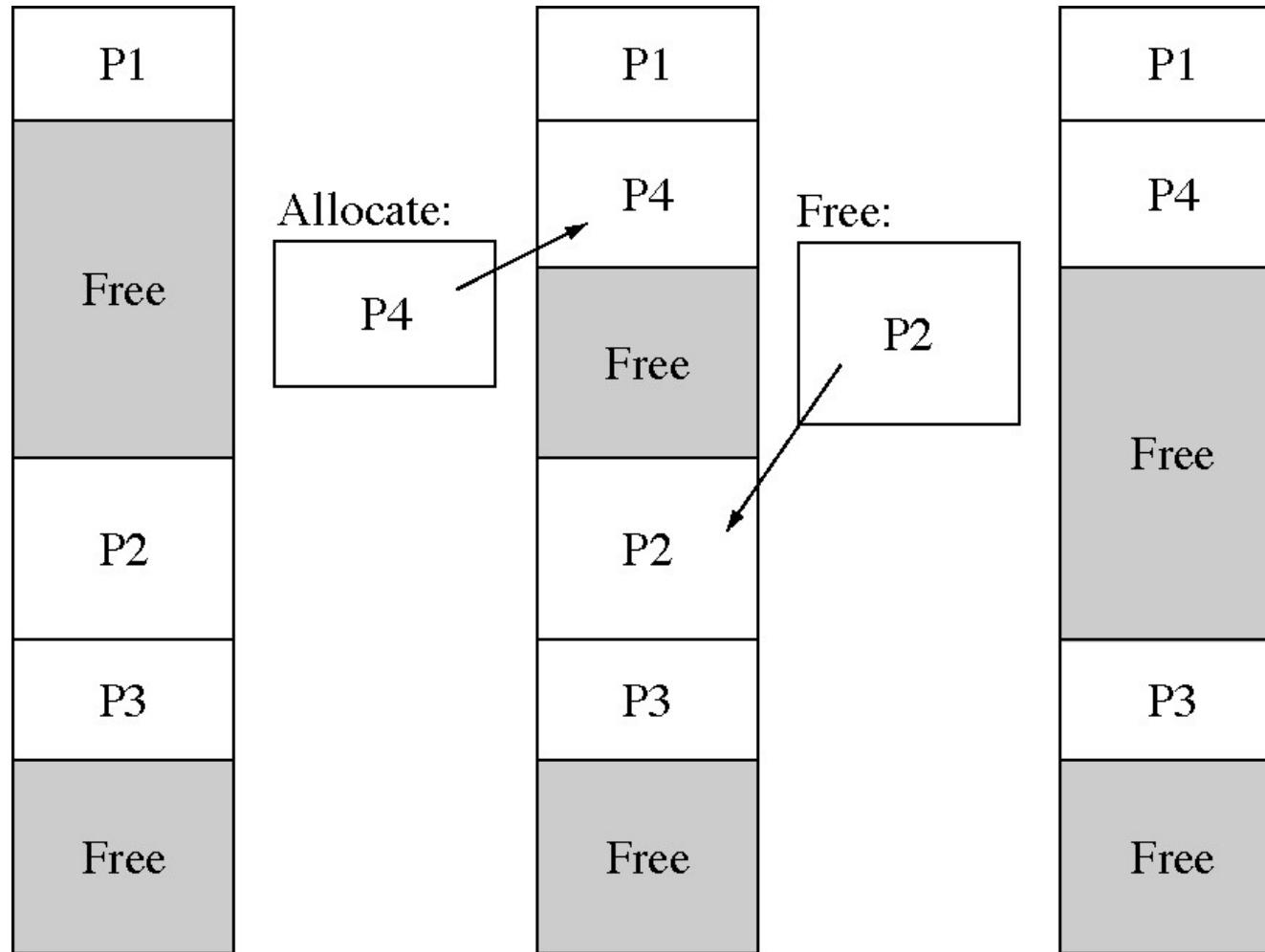
Process C requests 100 KB of memory.

Process A releases its memory.

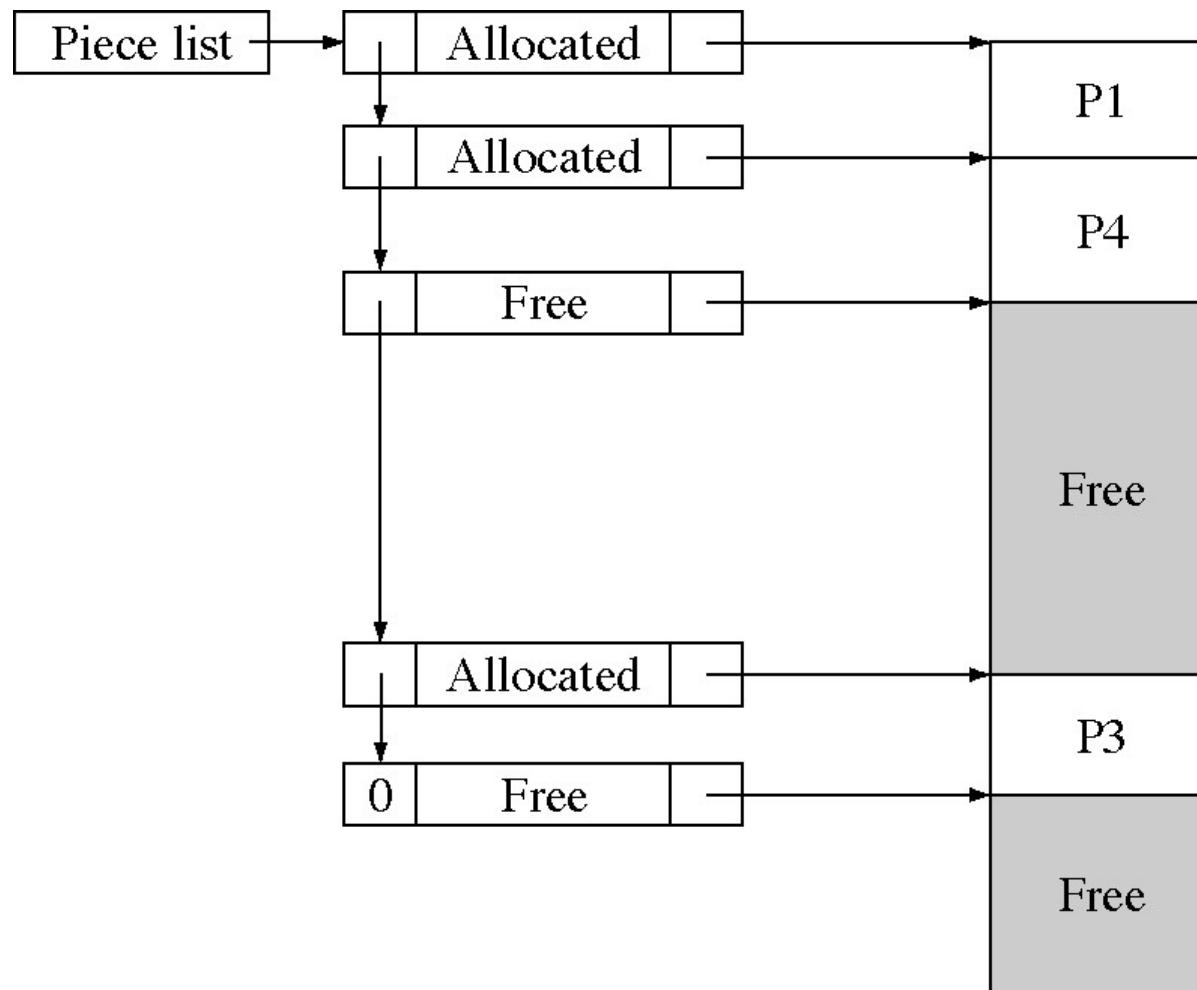
Process D requests 250 KB of memory.

Process C releases its memory.

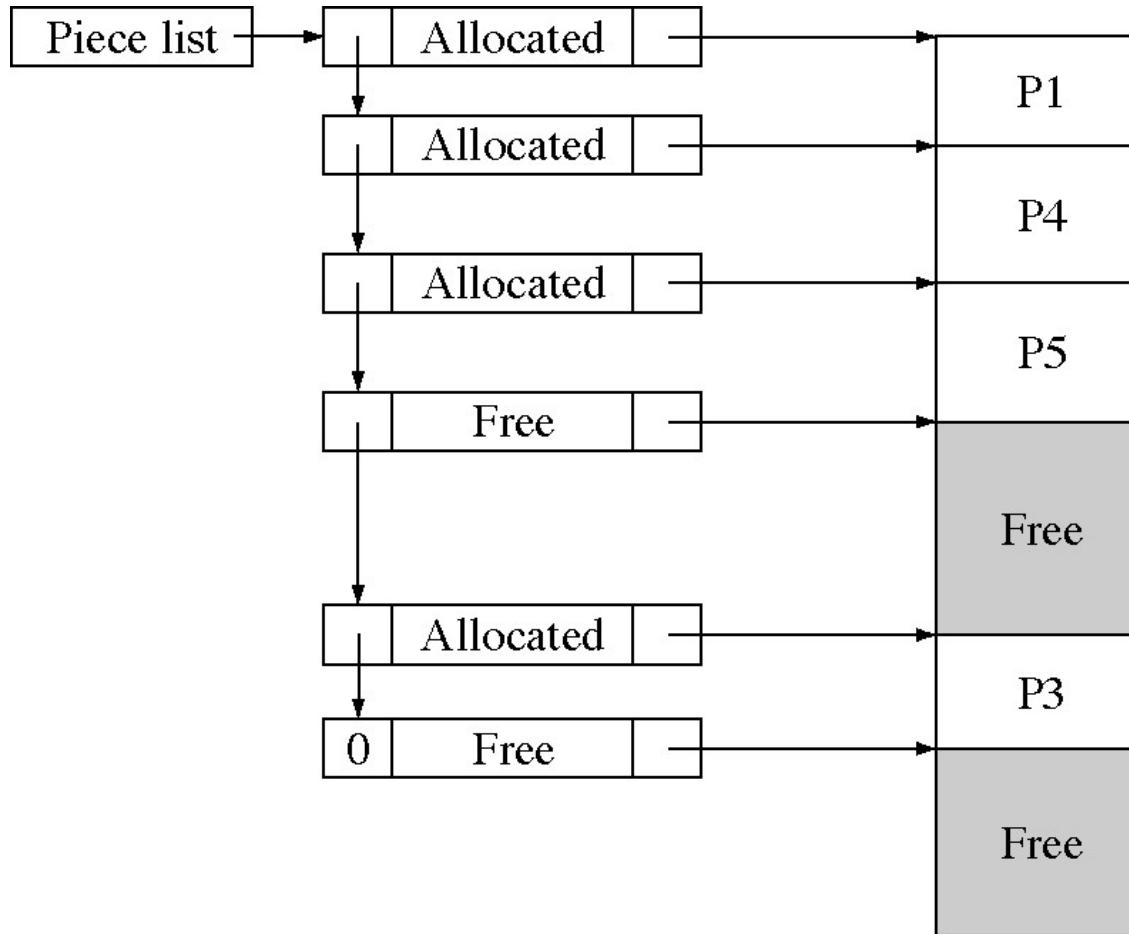
Allocating and freeing blocks



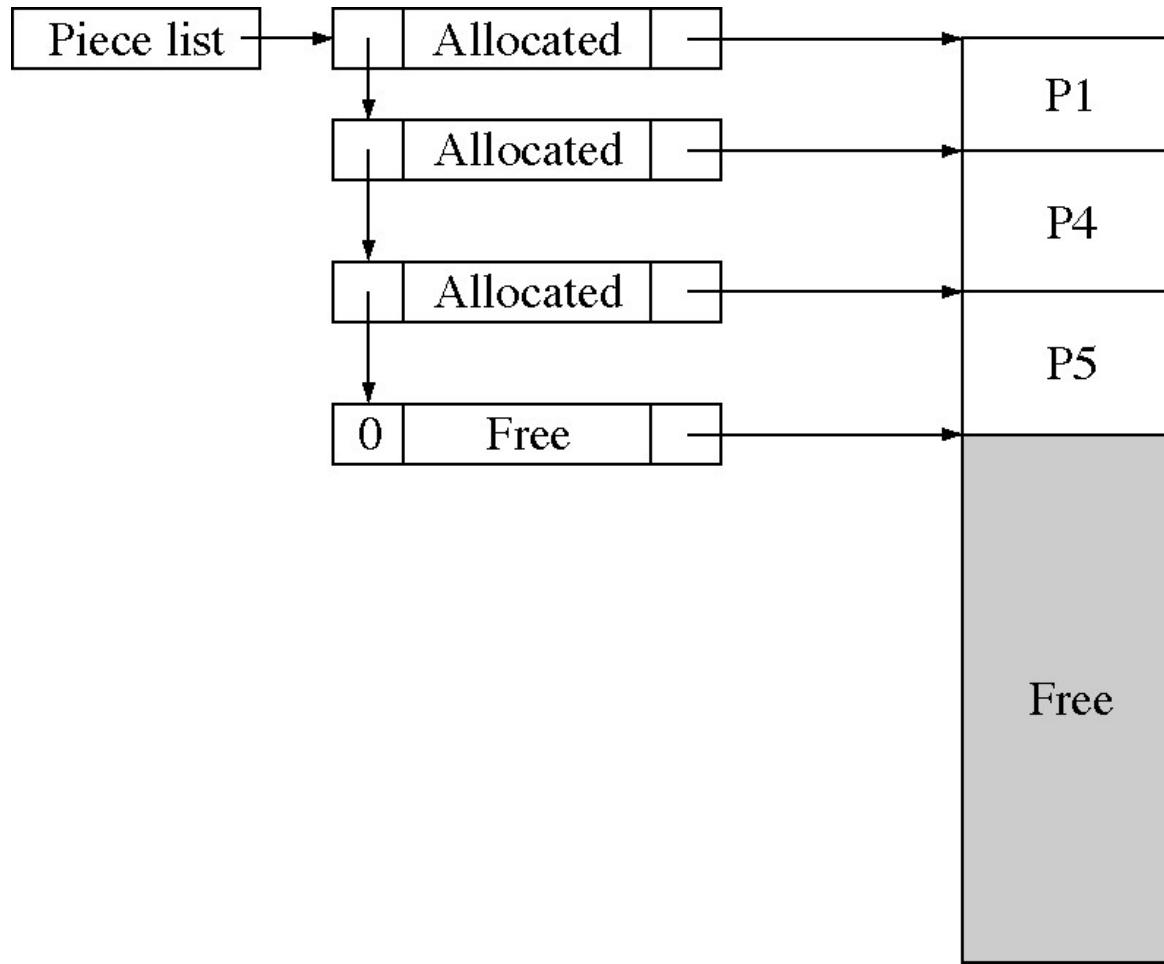
The block list method



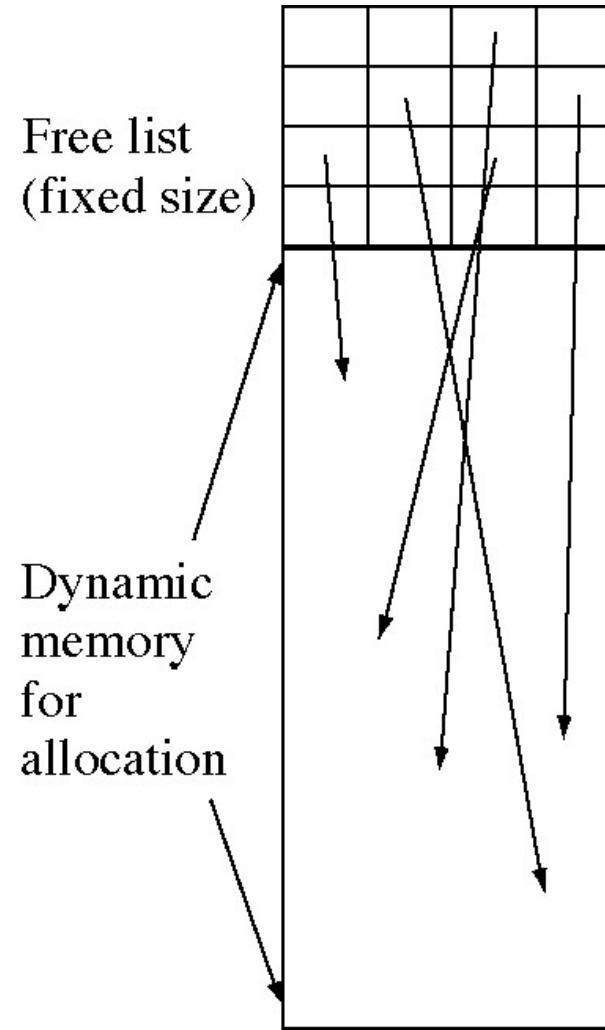
After allocating P5



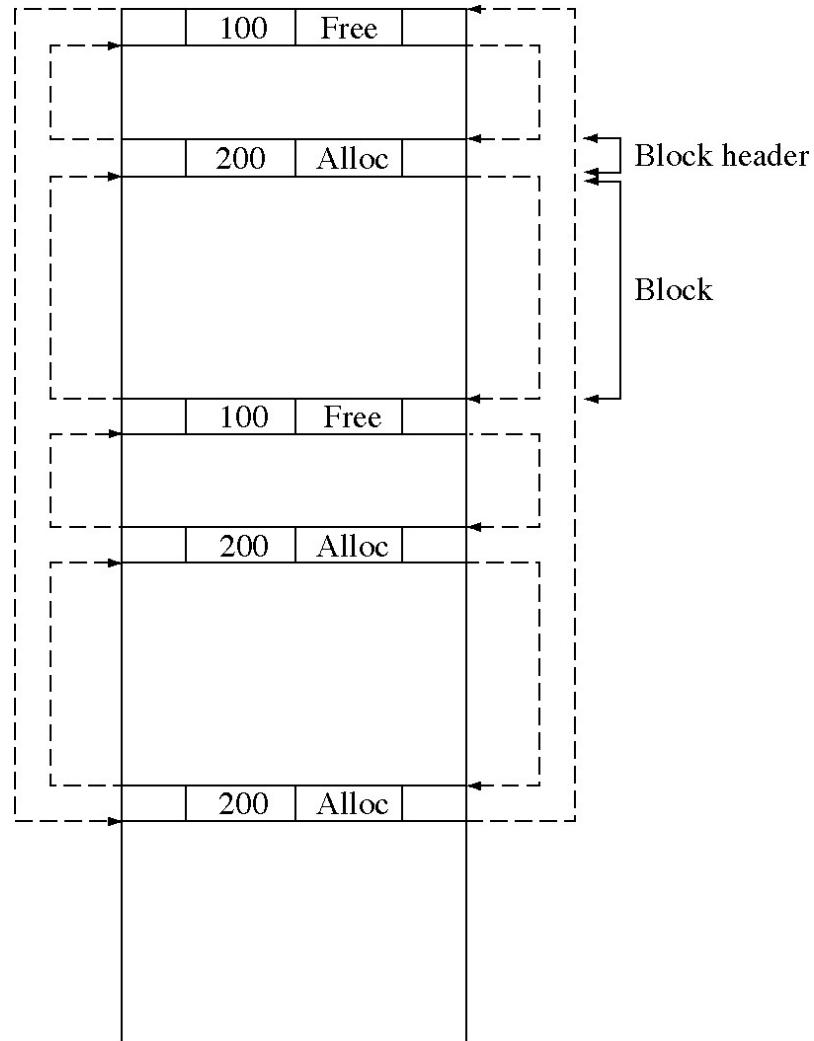
After freeing P3



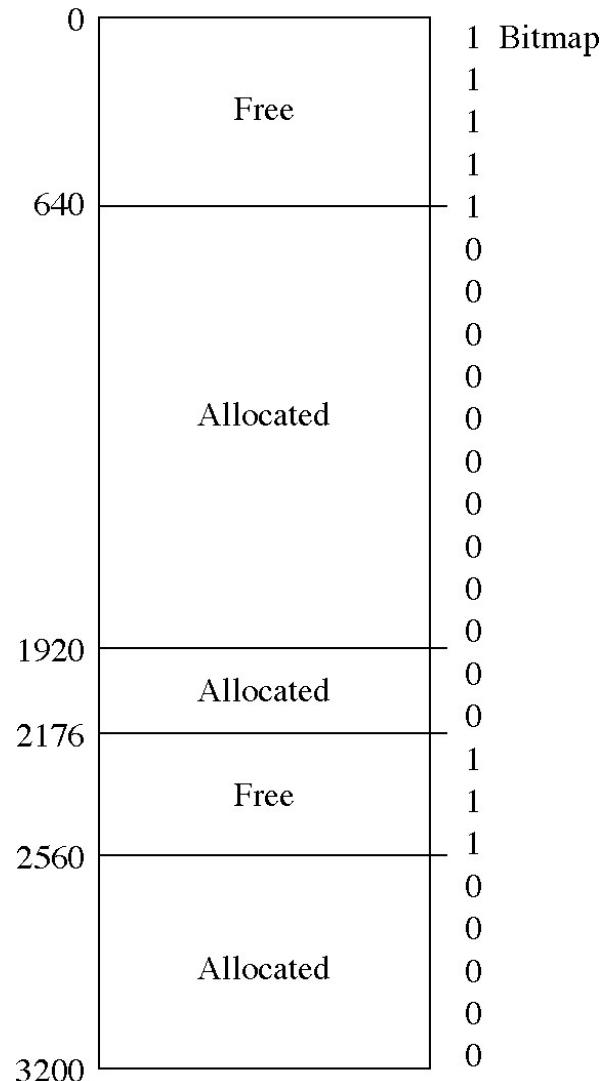
Reserving space for the block list



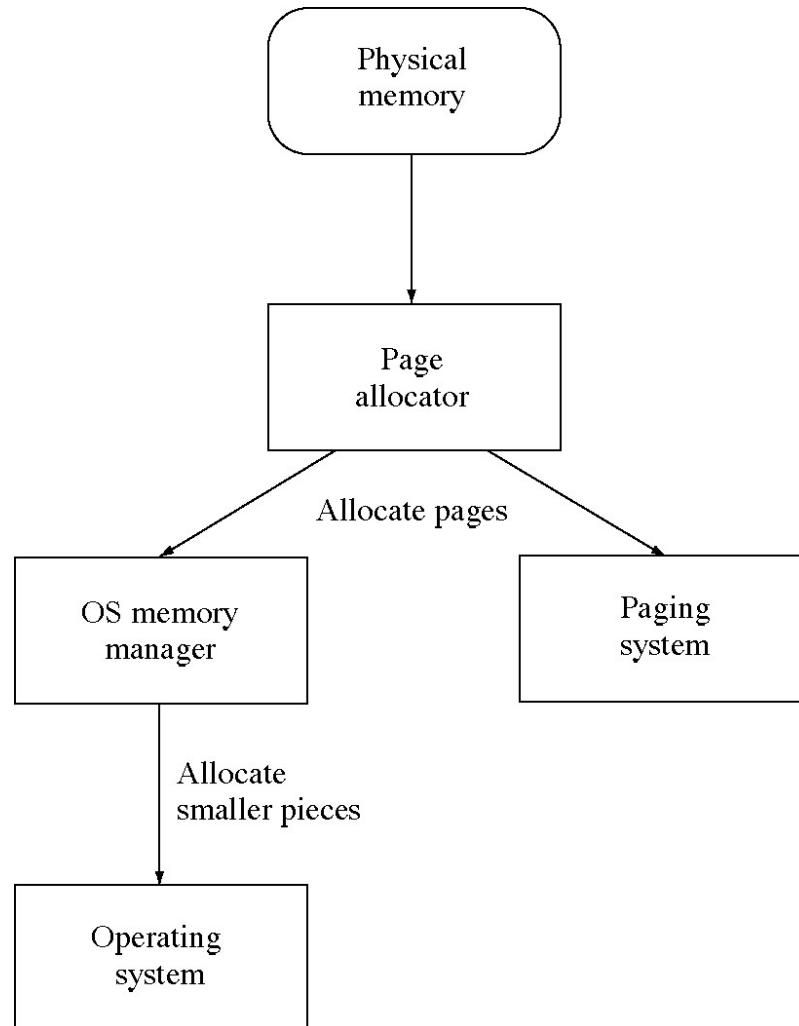
Block list with list headers



Bitmap method



Allocating memory in a paging system



Paging

- Paging permits the physical address space of a process to be non-contiguous.
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2).
- Divide logical memory into blocks of same size called **pages**.
- Keep track of all free frames.
- To run a program of size n pages, need to find n free frames and load program.
- Keep track of all free frames.
- Set up a page table to translate logical to physical

Example

Process A : 4 pages

Process B : 3 pages

Process C : 4 Pages

Process D : 5 Pages

Main Memory : 15 frames

Frame number	Main memory					
0		A.0	A.0	A.0	A.0	A.0
1		A.1	A.1	A.1	A.1	A.1
2		A.2	A.2	A.2	A.2	A.2
3		A.3	A.3	A.3	A.3	A.3
4			B.0	B.0	B.0	B.0
5			B.1	B.1	B.1	B.1
6			B.2	B.2	B.2	B.2
7				C.0	C.0	C.0
8				C.1	C.1	C.1
9				C.2	C.2	C.2
10				C.3	C.3	C.3
11						D.0
12						D.1
13						D.2
14						D.3

(a) Fifteen available frames

(b) Load process A

(c) Load process B

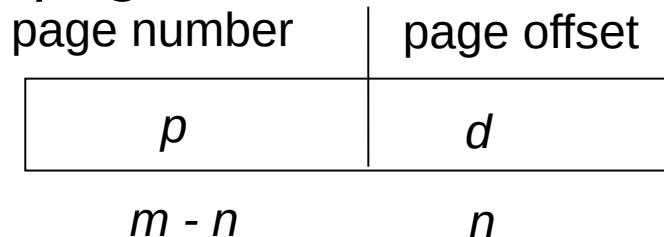
(d) Load process C

(e) Swap out B

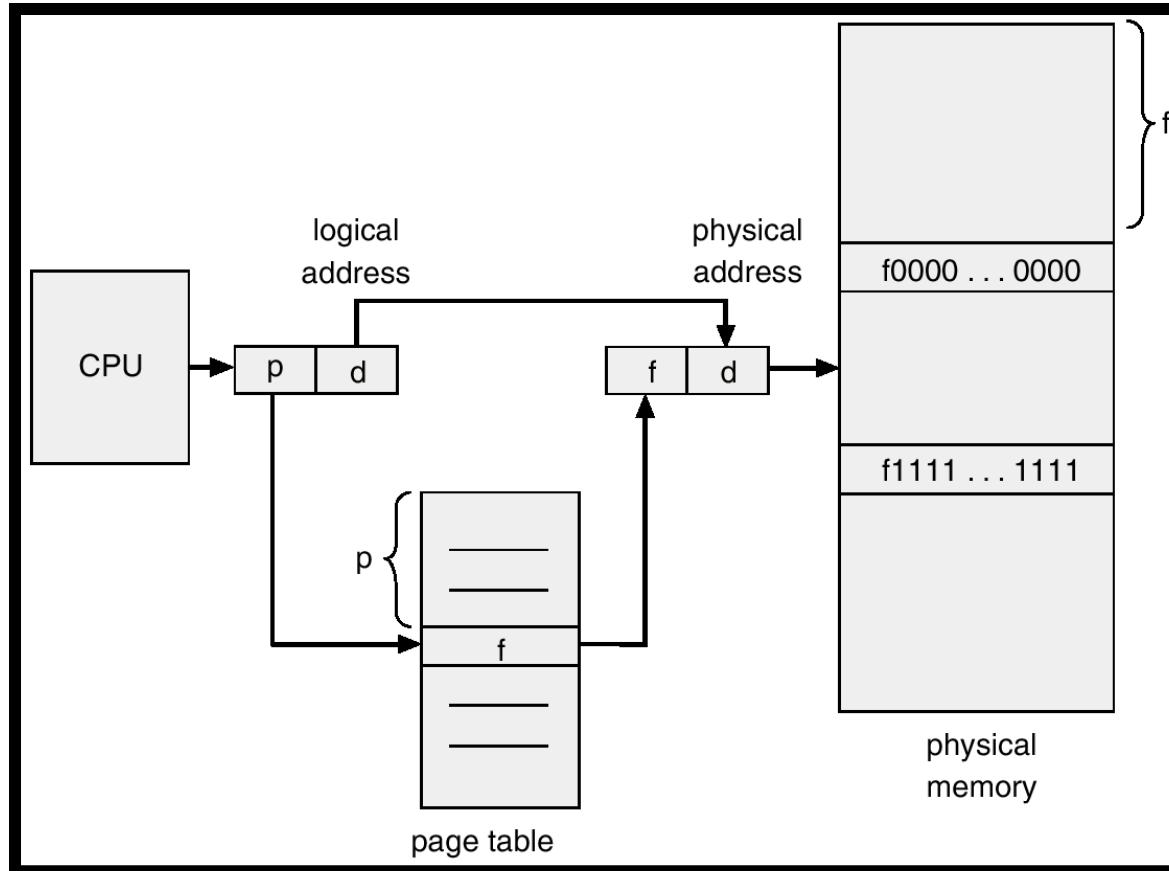
(f) Load process D

Address Translation Scheme

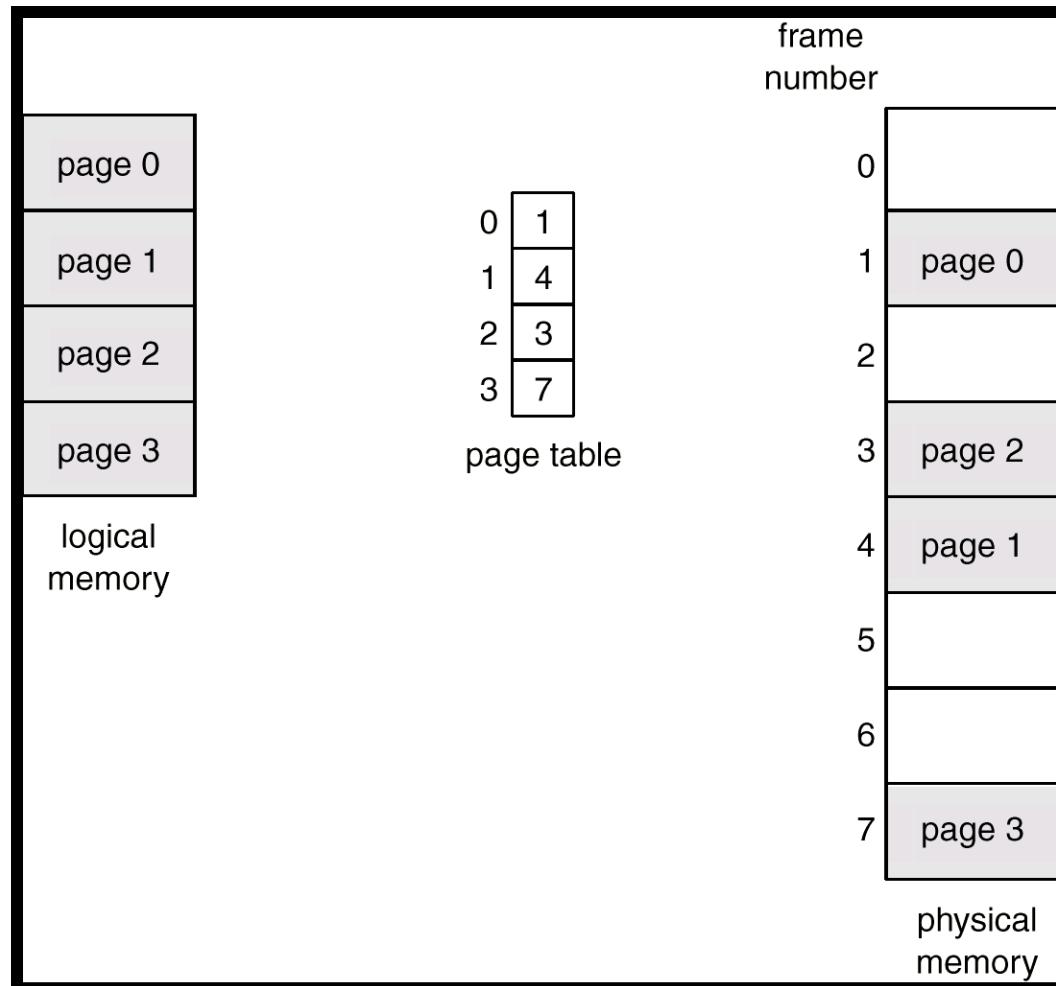
- Address generated by CPU is divided into:
 - *Page number (p)* – used as an index into a *page table*. Page table contains base address of each page in physical memory.
 - *Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit.
 - For given logical address space 2^m and *page size* 2^n



Address Translation Architecture



Paging Example



Static Memory Allocation	Dynamic Memory Allocation
Memory is allocated <u>before the execution</u> of the program begins. (During Compilation)	Memory is allocated <u>during the execution</u> of the program.
No memory allocation or deallocation actions are performed during Execution.	Memory Bindings are established and destroyed during the Execution.
Variables remain permanently allocated.	Allocated only when program unit is active.
Implemented using stacks and heaps	Implemented using data segments.
Pointer is needed to accessing variables.	No need of Dynamically allocated pointers.
Faster execution than Dynamic	Slower execution than static
More memory Space required.	Less Memory space required.



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN SESSION 8

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Balamurali Shankar
WILP & Department of CS & IS



BITS Pilani
Pilani Campus



Operating systems – Virtual Memory Design

Last Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Levels of memory management• The memory management design problem• Solution to the memory management design problem• Dynamic memory allocation	<p>T1 (10.1)</p> <p>T1 (10.5)</p> <p>T1(10.6)</p> <p>T1(10.7)</p>





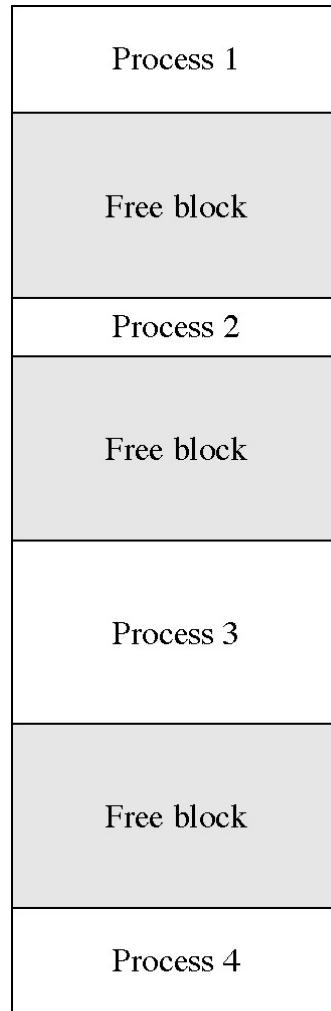
Virtual Memory Design

Today's topic

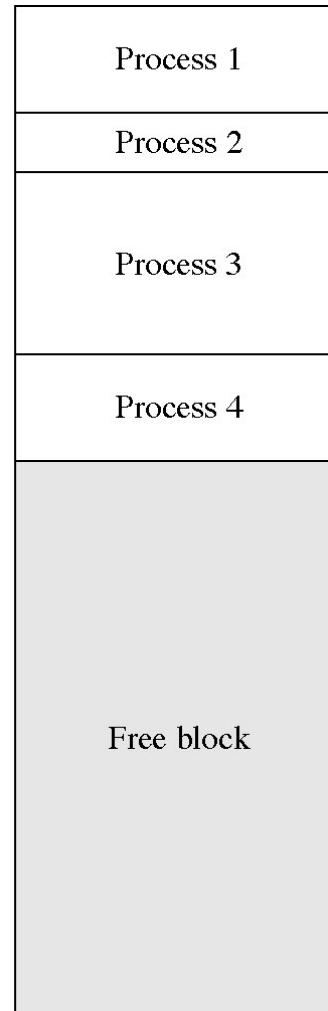
- Fragmentation and compaction
- Page table location : Hardware register Vs. Main memory Vs Cache
- Implementing Virtual memory



Fragmentation / Compaction



Before compaction

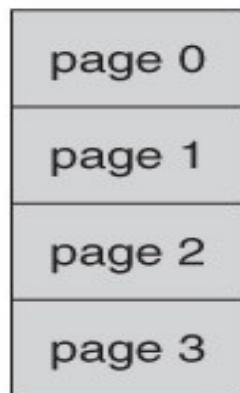


After compaction

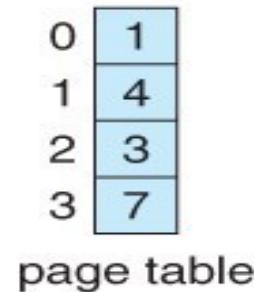
Fragmentation / Compaction

- Without memory mapping, programs require physically continuous memory
- Large blocks mean large fragments
 - and wasted memory
- We need hardware memory mapping to address this problem
 - segments
 - pages
- We will look at a series of potential solutions

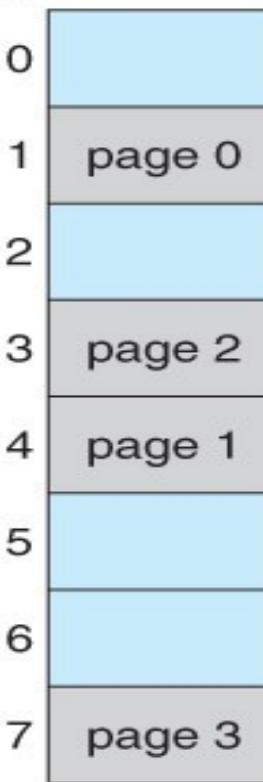
Paging



logical
memory

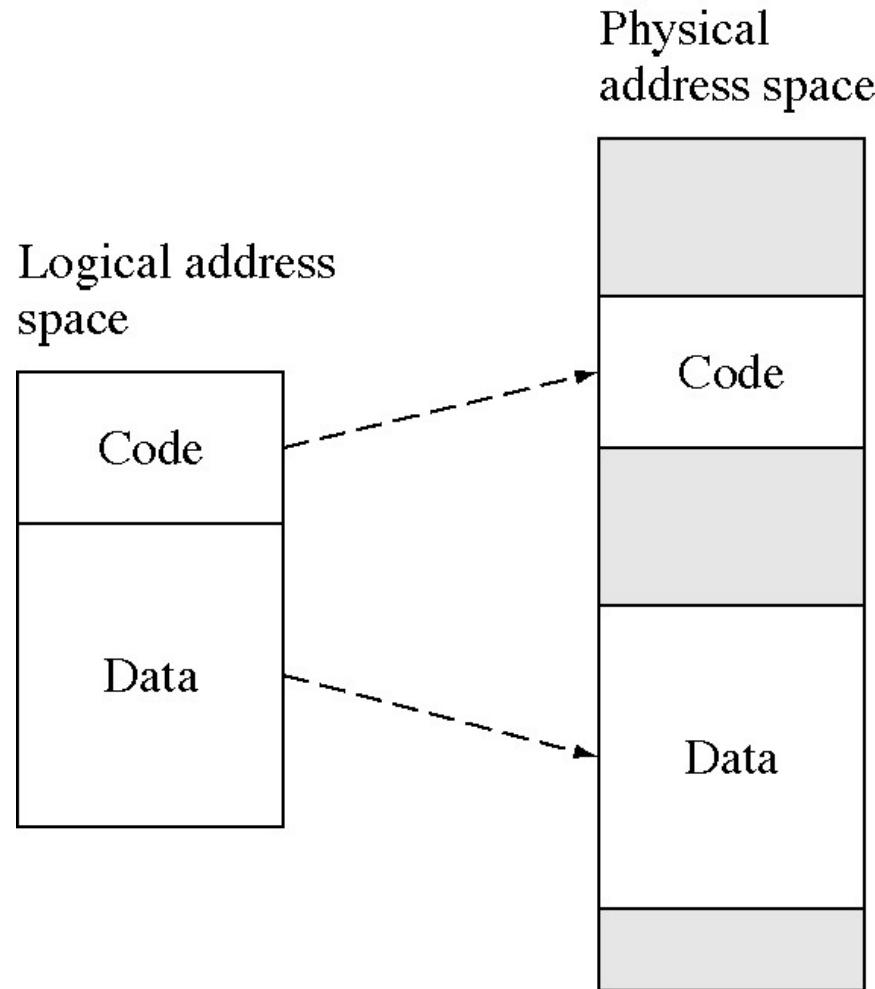


frame
number

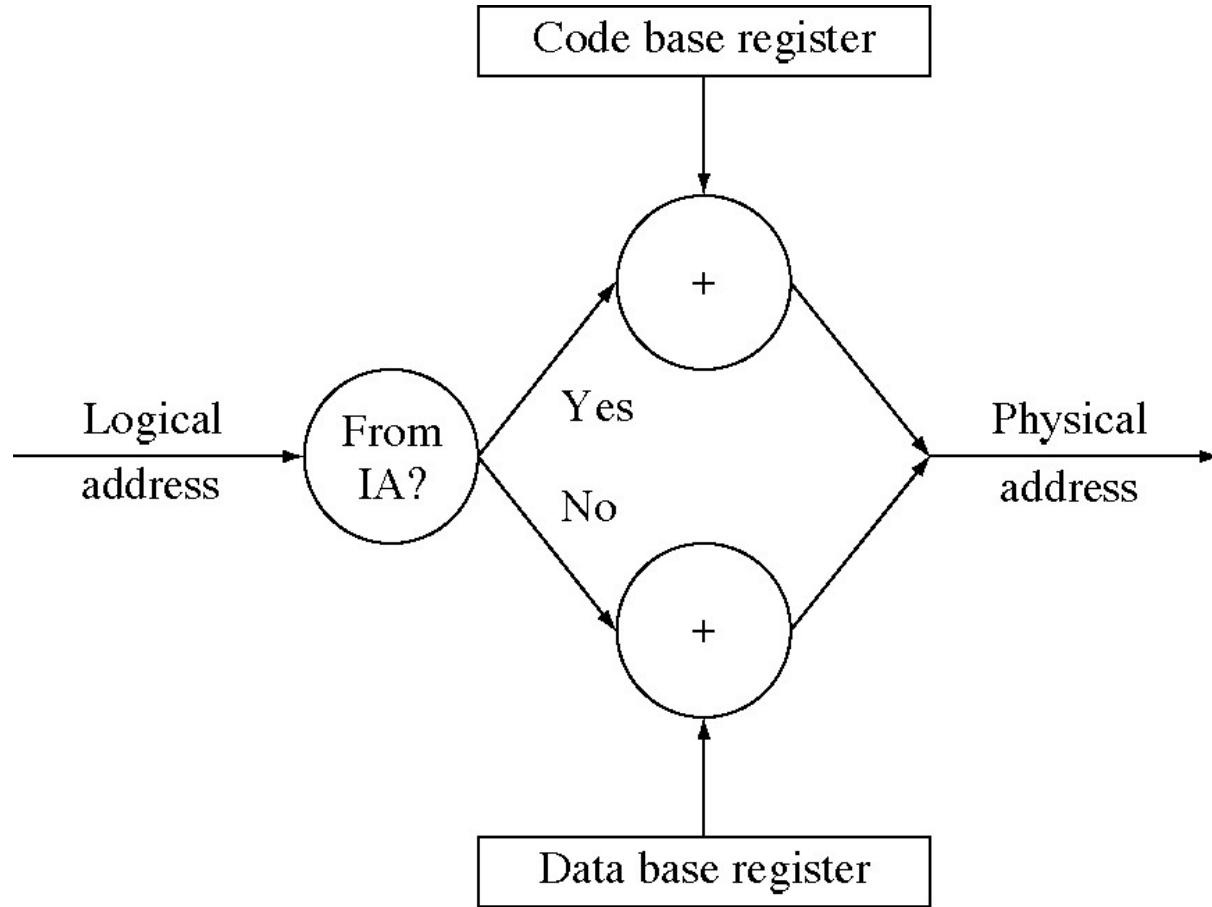


physical
memory

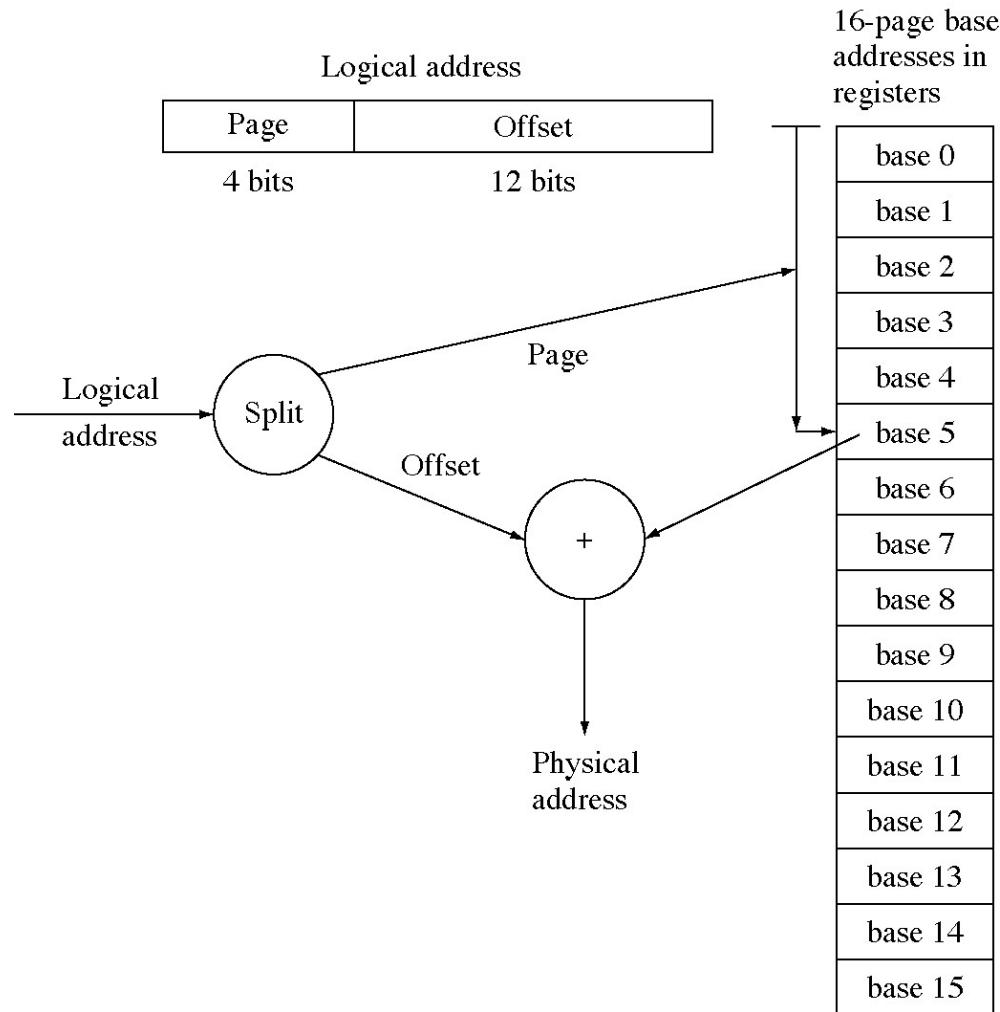
Separate code and data spaces



Code/data memory relocation



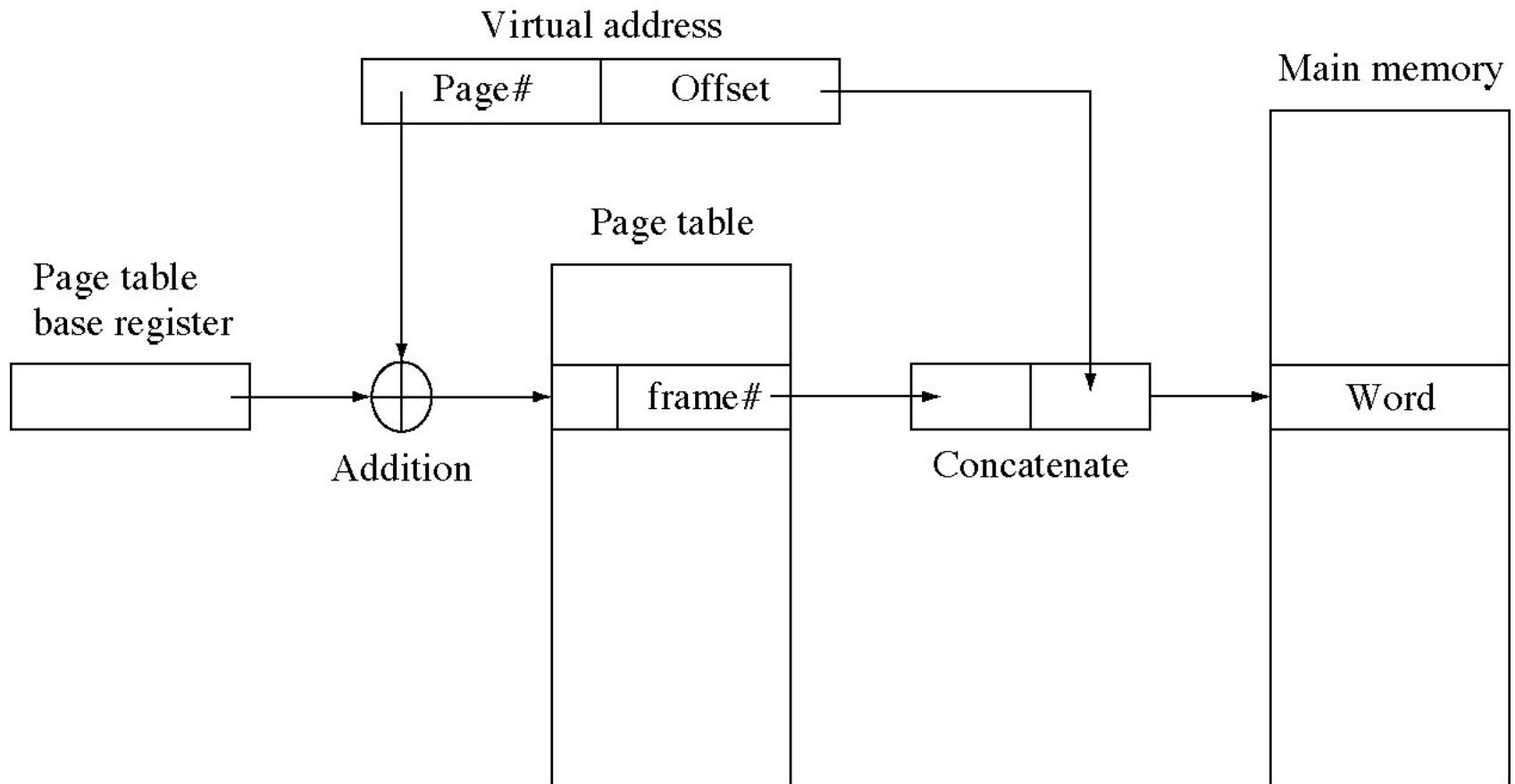
Hardware register page table



Problems with page tables in registers

- Practical limit on the number of pages
- Time to save and load page registers on context switches
- Cost of hardware registers
- *Solution:* put the page table in memory and have a single register that points to it

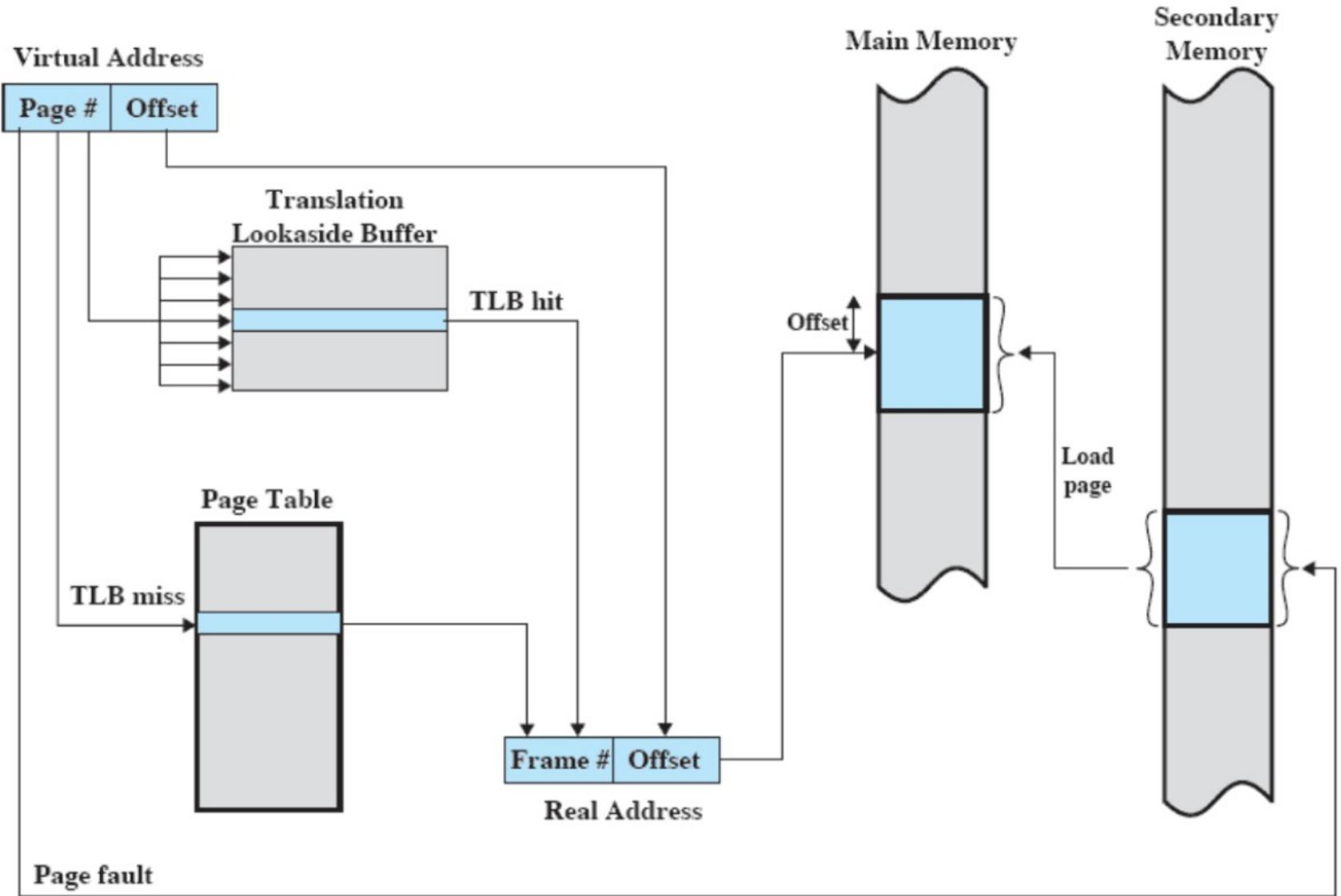
Page table mapping



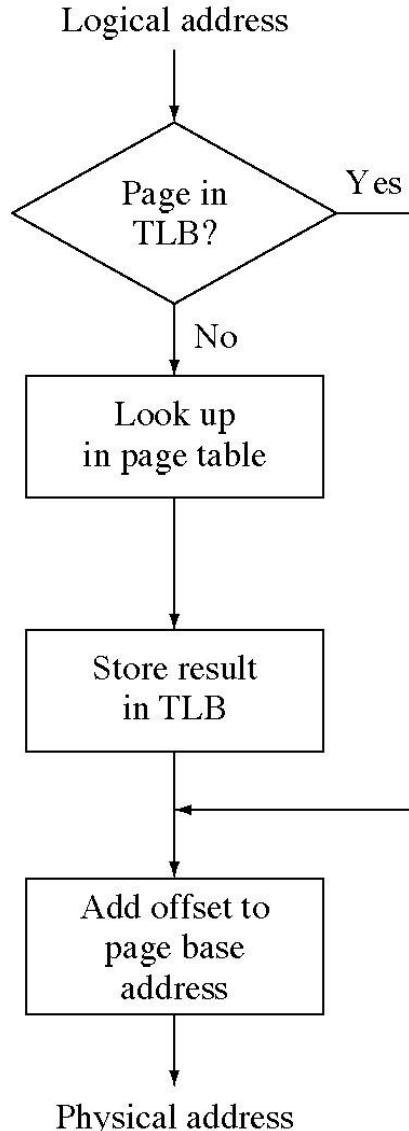
Problems with page tables in memory

- Every data memory access requires a corresponding page table memory access
 - the memory usage has doubled
 - and program speed is cut in half
- *Solution:* caching page table entries
 - called a *translation lookaside buffer*
 - or TLB

Translation look aside buffer



TLB flow chart

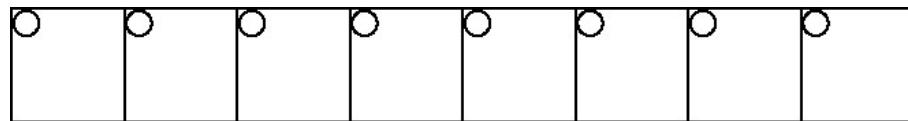


Why TLBs work



- Memory access is not random, that is, not all locations in the address space are equally likely to be referenced
- References are localized because
 - sequential code execution
 - loops in code
 - groups of data accessed together
 - data is accessed many times
- This property is called *locality*
- TLB hit rates are 90+% .

Good and bad cases for paging

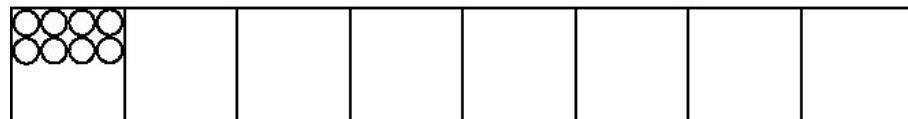


```
for(i=0;i<8;++i)
  sum+=a[i][0];
```

(worst case)

○ = Memory reference

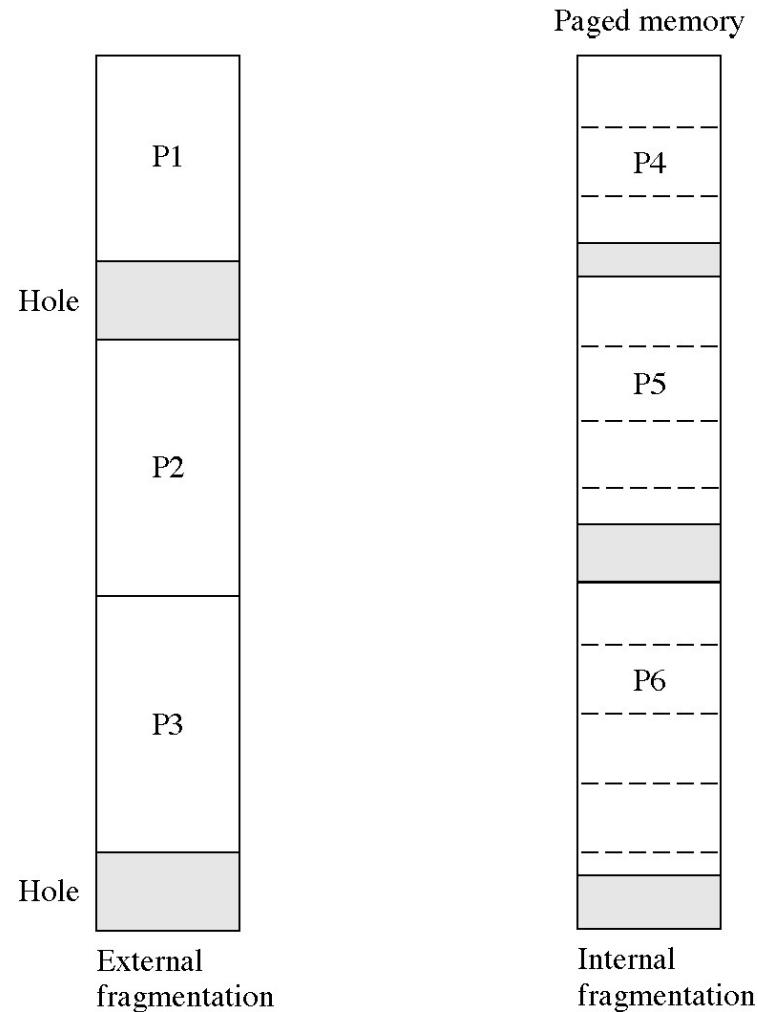
□ = Page



```
for(j=0;j<8;++j)
  sum+=a[0][j];
```

(best case)

Internal and external fragmentation



Page and page frame

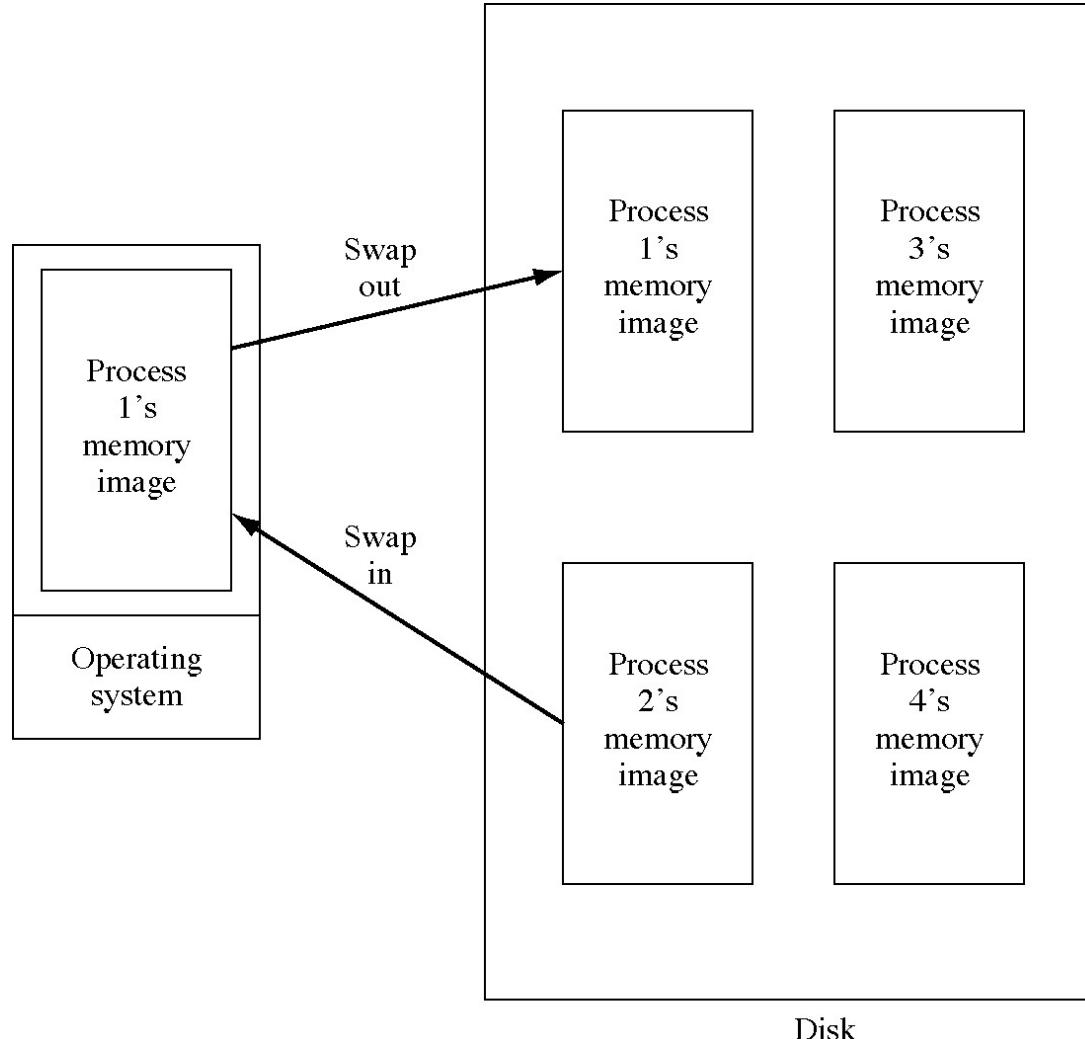
- Page
 - the information in the page frame
 - can be stored in memory (in a page frame)
 - can be stored on disk
 - multiple copies are possible
- Page frame
 - the physical memory that holds a page
 - a resource to be allocated

Page table protection

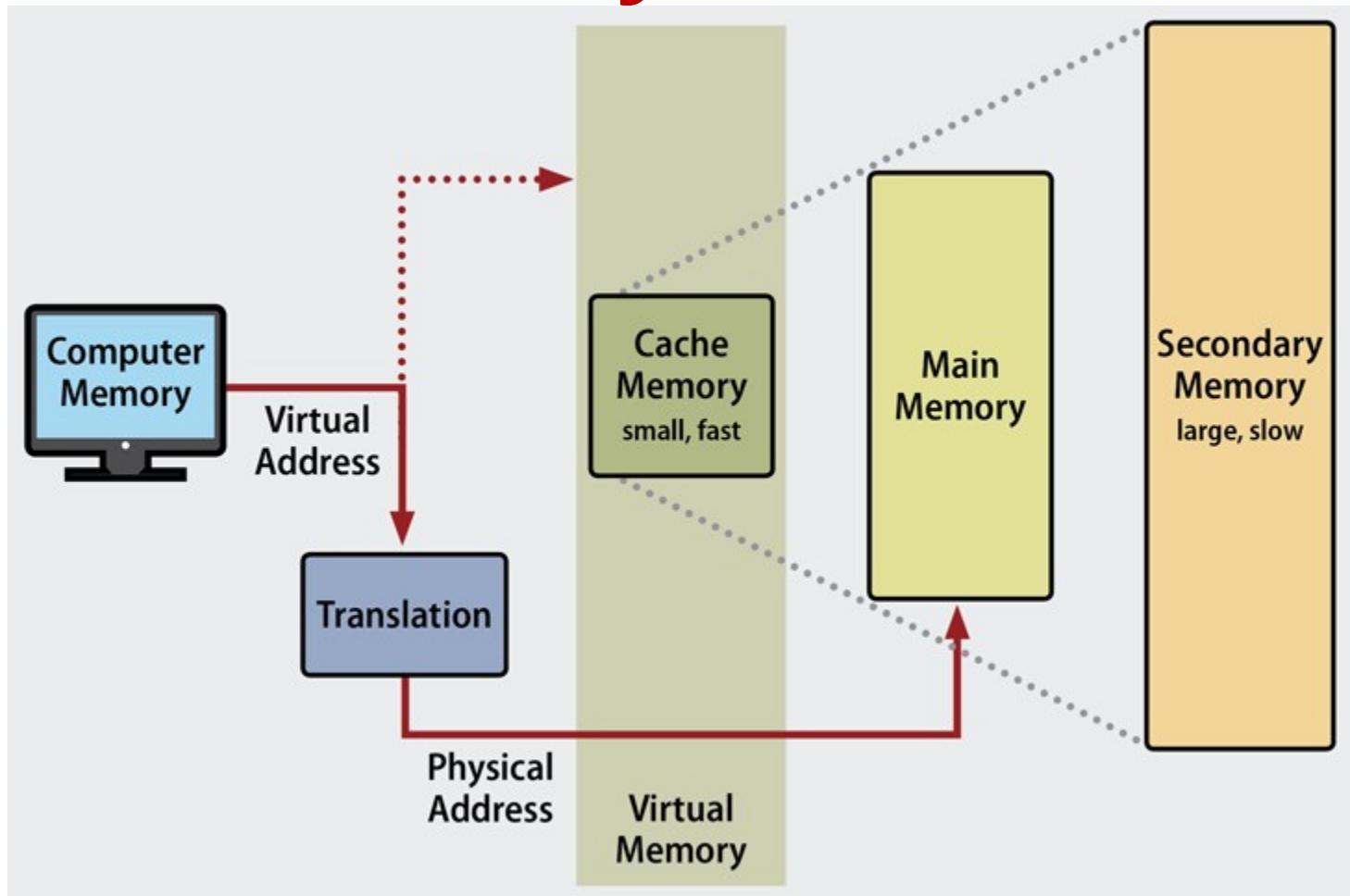


- Three bits control: read, write, execute
- Possible protection modes:
 - 000: page cannot be accessed at all
 - 001: page is read only
 - 010: page is write only
 - 100: page is execute only
 - 011: page can be read or written
 - 101: page can be read as data or executed
 - 110: write or execute, unlikely to be used
 - 111: any access is allowed

Swapping



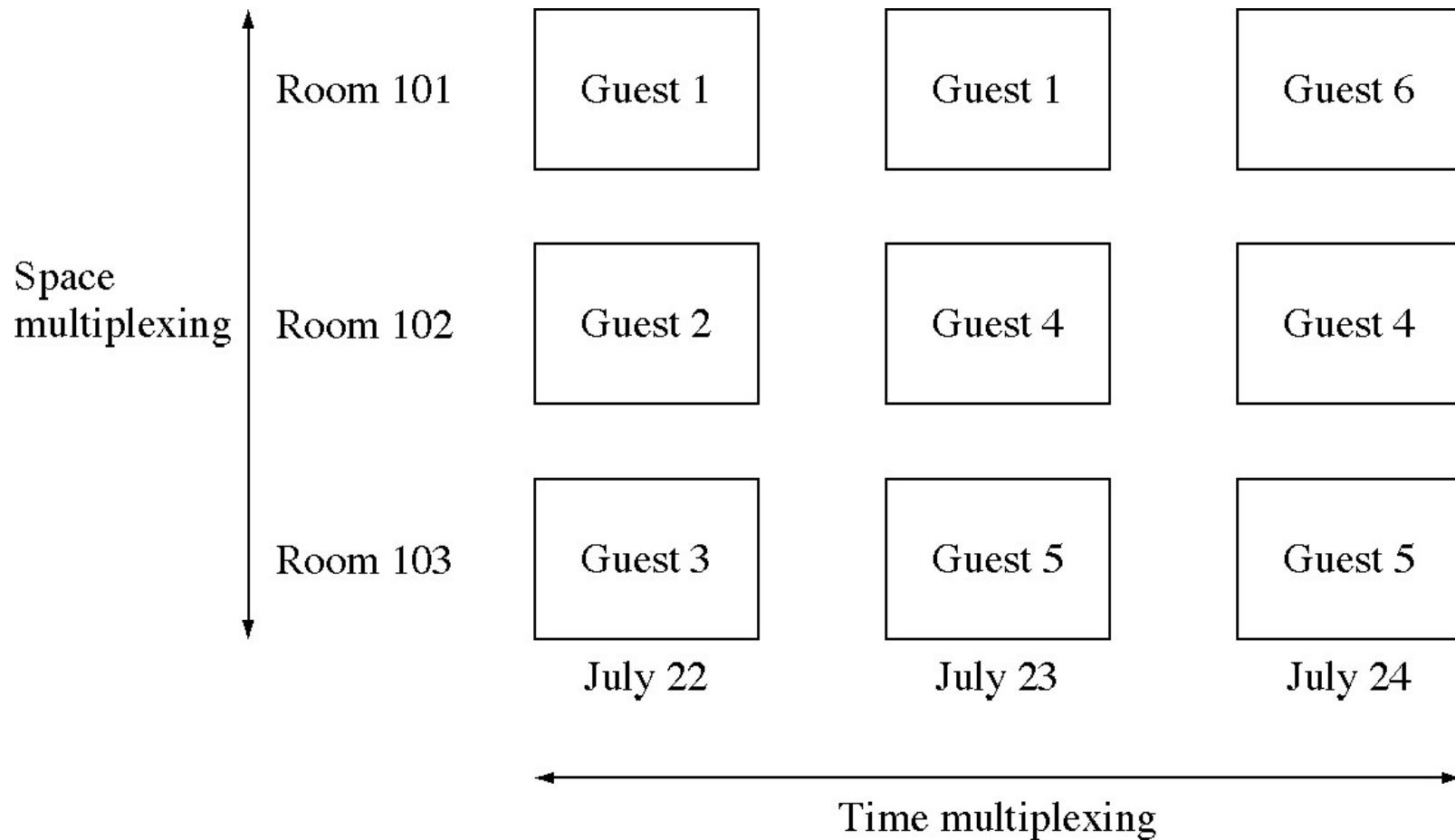
Virtual memory



Time and space multiplexing

- The processor is a time resource
 - It can only be time multiplexed
- Memory is a space resource
 - We have looked at space multiplexing of memory
 - Now we will look at time multiplexing of memory

Time and space multiplexing of hotel rooms



Implementation of virtual memory

- Virtual memory allows
 - time multiplexing of memory
 - users to see a larger (virtual) address space than the physical address space
 - the operating system to split up a process in physical memory
- Implementation requires extensive hardware assistance and a lot of OS code and time
 - but it is worth it

Virtual memory software

- The virtual memory (a.k.a. paging) system in the OS must respond to four events
 - process creation
 - process exit
 - process dispatch
 - page fault

Process creation actions



- 1. Compute program size (say N pages)
- 2. Allocate N page frames of swap space
- 3. Allocate a page table (in the OSs memory) for N page table entries.
- 4. Initialize the swap area
- 5. Initialize the page table: all pages are marked as not present.
- 6. Record the location in the swap area and of the page table in the process descriptor

Process exit actions

- 1. Free the memory used by the page table
- 2. Free the disk space in the swap area
- 3. Free the page frames in process was using

Process dispatch actions

- 1. Invalidate the TLB (since we are changing address spaces)
- 2. Load the hardware page table base register with the address of the page table for this process

Page fault actions

- 1. Find the faulting page (say page K)
- 2. Find an empty page frame. This will involve replacing a page.
- 3. Read in page K to this page frame
- 4. Fix up the page table entry for page K. Mark it present and set the base address.
- 5. Restart the process with the instruction that caused the page fault

Locality



- Programs do not access their address space uniformly
 - they access the same location over and over
- *Spatial locality*: processes tend to access location near to location they just accessed
 - because of sequential program execution
 - because data for a function is grouped together
- *Temporal locality*: processes tend to access data over and over again
 - because of program loops
 - because data is processed over and over again

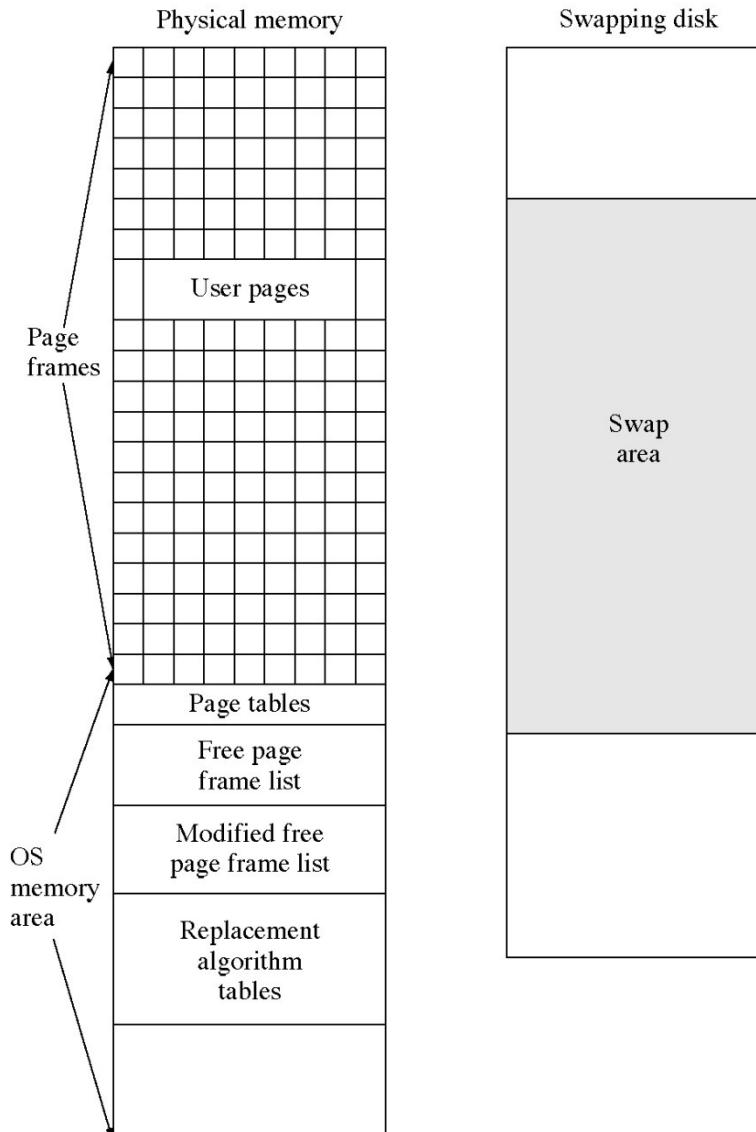
Design technique: Locality

- Locality is almost always present
 - and often we can optimize a design by taking advantage of it.
- Caching is a common name for systems that take advantage of locality to optimize operations

Practicality of paging

- Paging only works because of locality
 - at any one point in time programs don't need most of their pages
- Page fault rates must be very, very low for paging to be practical
 - like one page fault per 100,000 or more memory references

VM data structures



Reactive and proactive user interfaces



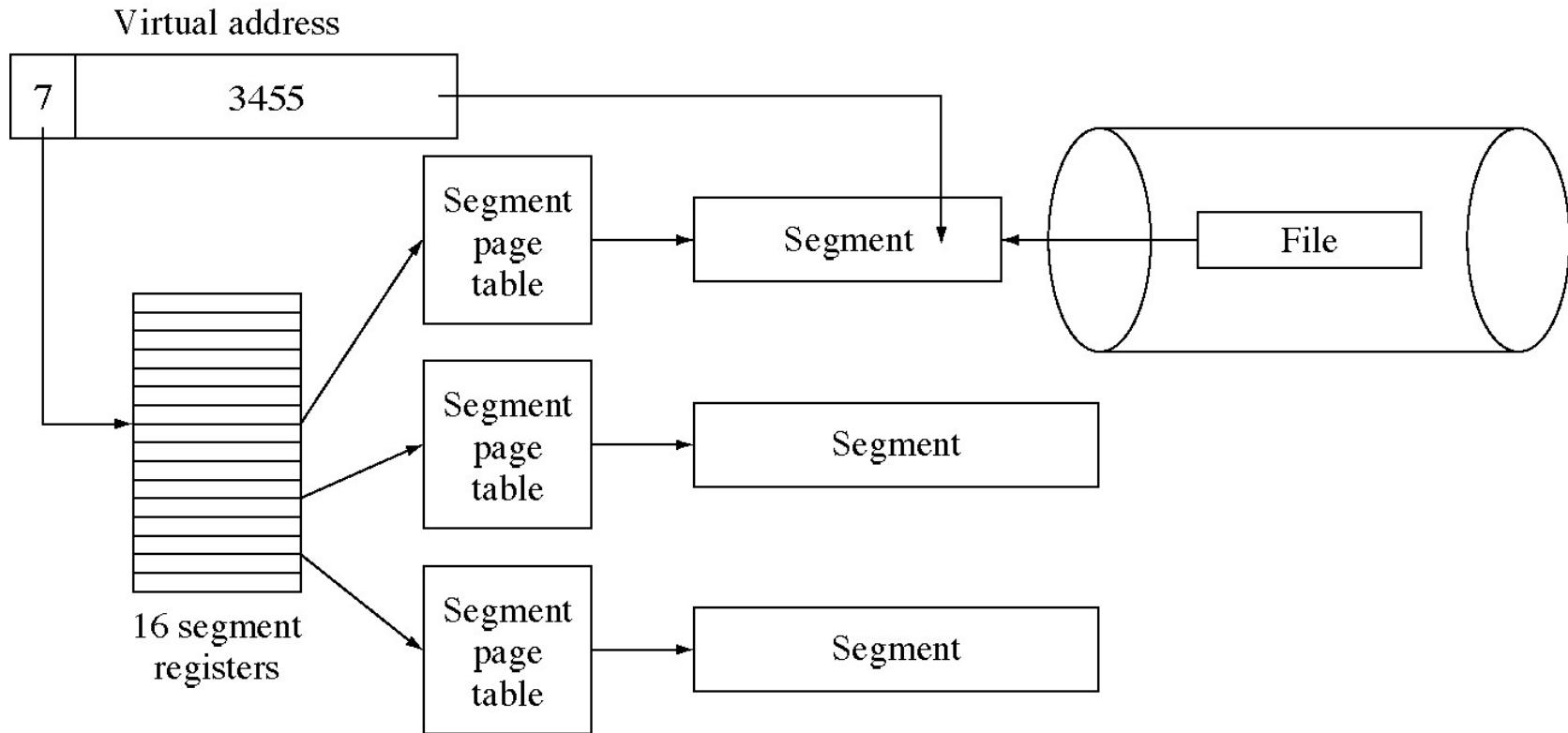
- Graphical user interfaces are generally reactive, they wait for user actions
 - this is a good model and puts the user in control which is good psychologically
- Agents are a new user interface concept
 - agents are proactive
 - they go out and look for useful things to do

Design technique: Polling, software interrupts and hooks



- How can a process know when an event occurs?
There are two approaches
- *polling*: it can check periodically
- *interrupts*: it can ask another process to interrupt it when the event occurs
 - the other process is usually the one that causes or handles the event
 - so it is not much trouble for it to inform the waiting process that the event has occurred.

Virtual memory in the IBM 801



File mapping



- *File mapping* is the mapping of a file on disk into the virtual address space of a process.
- File I/O then consists of reading and writing words in the virtual address space
 - no system calls are required for read and write
- This is also called a *memory-mapped file*.
- The I/O system and the paging system both move data between disk and memory
 - so it makes sense to combine them



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN SESSION 9

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Dr Thangakumar J
WILP & Department of CS & IS



BITS Pilani
Pilani Campus

Operating systems - I/O Abstraction



Last Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Fragmentation and compaction• Page table location : Hardware register Vs. Main memory Vs Cache• Implementing Virtual memory	<p>T1 (11.1)</p> <p>T1(11.2.4, 11.2.5, 11.2.6)</p> <p>T1(11.7)</p>

I/O Abstraction

Today's
topic

- **I/O System Hardware**
 - Device Controllers
 - Types of devices : Terminal Devices, Communication Devices and Disk Devices
- **Data transfer schemes**
 - Programmed I/O, Interrupt Driven I/O, DMA

INPUT OUTPUT - STORY LINE

1

POLLING

- CHECK
- INITIATE
- COMPLETE

3

DIRECT MEMORY ACCESS

- INPUT OUTPUT IN BLOCKS
- WHICH DEVICE, HOW MANY BLOCKS, MEMORY LOCATION.
- CPU “interrupted” to check for errors; assigns next program

2

INTERRUPTIONS

- SAVE CONTEXT IN MEMORY
- INTERRUPT SERVICE ROUTINE
- LOOKUP TO EXECUTE INTERRUPT SERVICE ROUTINE

4

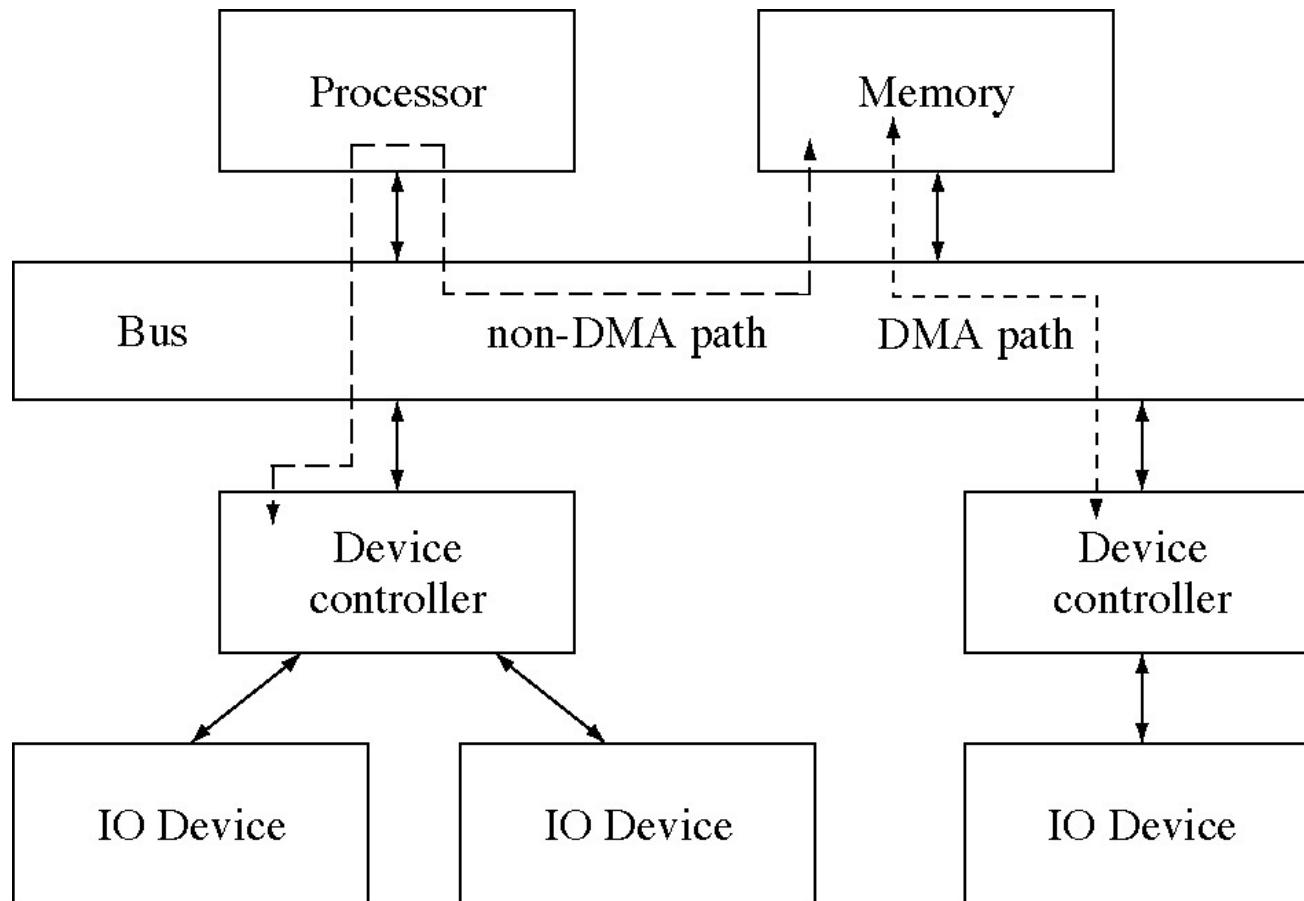
INPUT OUTPUT PROCESSORS RESUME OPERATIONS

- IOP CAN MANAGE MANY DEVICES
- IOP CAN MANAGE HIGH LEVEL IOP
- MORE SOPHISTICATED THAN DMA

DEVICES AND CONTROLLERS

- ***Input device:*** Generates data for a computer to read.
- ***Output device:*** Accepts data from computer.
- ***Device controller:*** an intermediate electronic device that interfaces between the computer system bus and one or more devices
- ***I/O processor or channel:*** a programmable device controller

Device Controller and I/O Devices



- An **interface Unit**
- Communicates with both I/O devices and computer system (over bus)
- Figure shows the connection between the I/O devices, the device controller and the system bus.
- **I/O Device** – transfer of data functionality; **I/O Controller** – interface to the computer functionality.
- **I/O Processor (Channels)** – Device controller know how to execute programs of more than one instruction.
- A controller with Direct Memory Access (DMA) can directly read and write the memory in the system. This can **transfer data faster than non-DMA controller**.

TERMINAL DEVICES

- **Terminal** – a device that allows a user to communicate with a computer.

Screen, Keyboard.

- **Serial Port** – A connection which allows communication of a stream of bits.

- **ASCII code** - display-a-character; E.g., display command 65 represents

A.

- Fixed number of lines and columns – 24*80 (1960 bytes)

- The screen is refreshed from 30 to 60 times a second.

- **Screen Refreshing** - A beam of electrons at a special chemical (Phosphor) on the inside of the screen that glows for a short time after it has been hit by the electron beam

- **Display** – Glass tube (Cathode Ray Tube or **CRT**)

- **Scan Line** – a line from the upper left corner and moves to the right horizontally. Fixed number of dot positions are available along the horizontal line.

- **Pixel** – dot position

- **Horizontal Retrace** – beam turned off (Pixel goes to right end of horizontal line) and moved back to the left end of the screen (previous line starts). Continues till it reaches a lowest right corner.

- **Vertical Retrace** – moves back up to the upper left corner (beam is off)

- **Character Generator Memory** – Converts an ASCII Characters from screen memory into an array of 8 by 14 dots that can be written on the screen.
- Shape of the character is changed by changing the content of Character Generator Memory (ROM chip or writable screen generator memory).
- ASCII codes (7 bit long), screen memory uses (8 bit) bytes. 128 ASCII characters are not used.
- The keyboard generates character codes for the keys that are typed, and these character codes are sent on to the device controller.

Mouse Devices

- Connected either with Keyboard or directly to the computer by its own serial port. So mouse provides two types of inputs.
 - keyboard events
 - key down
 - key up
 - mouse event
 - mouse button down
 - mouse button up
 - mouse movement
- It is a relative device reports the change in position in the two direction.
- Its button reports when it is pressed or released.
- Coded as multibyte records.

Color and Color Maps

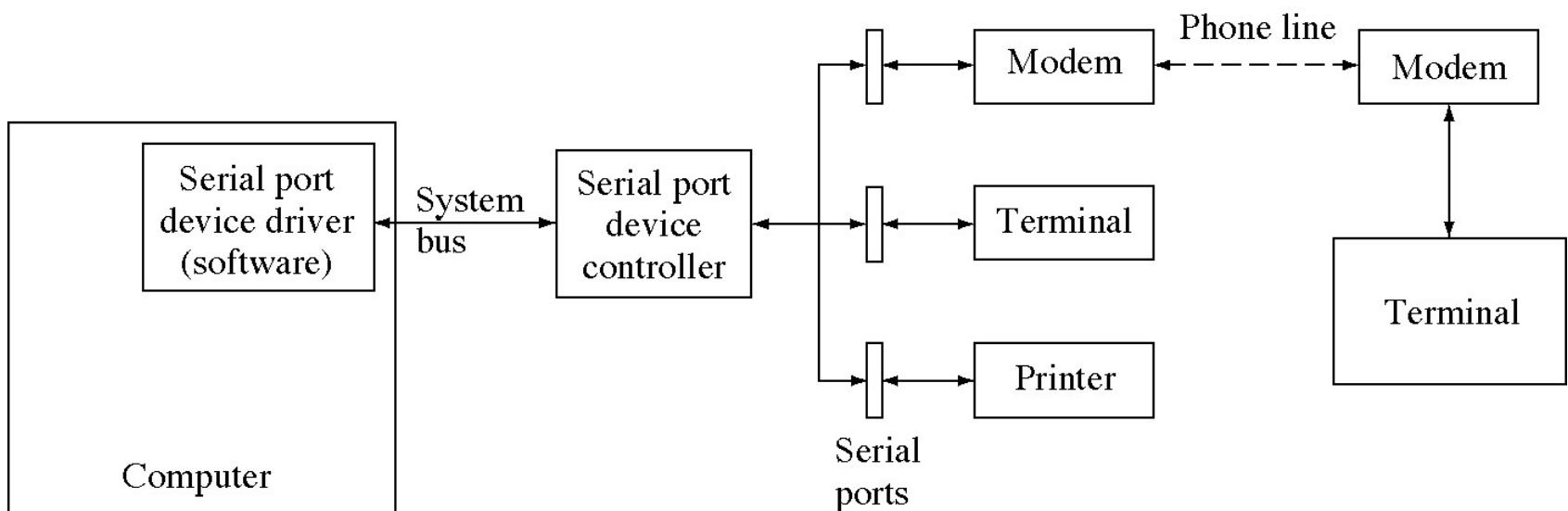
- Usually specified with 24 bits.
- **Color Map** – a set of 256 24-bit registers which map 8-bit color identifiers into 24-bit color specifications.
- Managed by window manager.

Modems

- **Modulation** – the process of converting bits to sound
- **Demodulation** – reverse process
- **Modem (Modulator – demodulator)** – the hardware device that knows how to do this.
- Modem devices are always paired because signals are modulated on one end and demodulated on the other end.

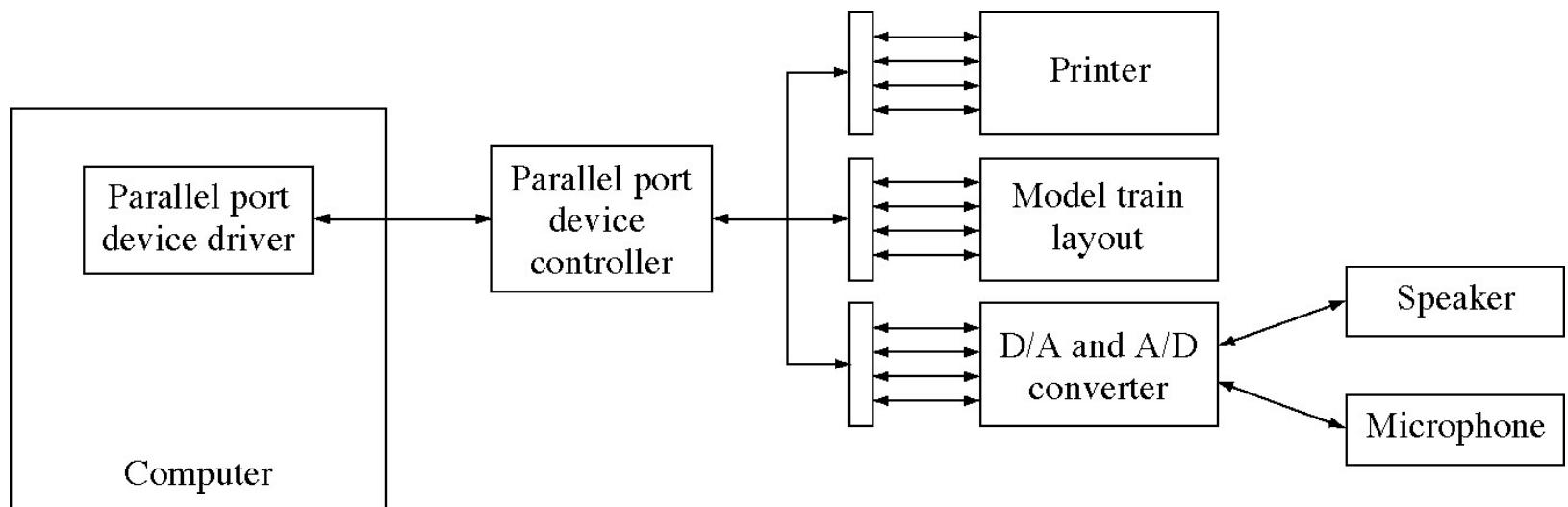
COMMUNICATION DEVICES – Serial Port

- **A Serial Port Controller** – Connects to a system bus and provides one or more serial ports for devices.
- **Example:** a printer, a mouse



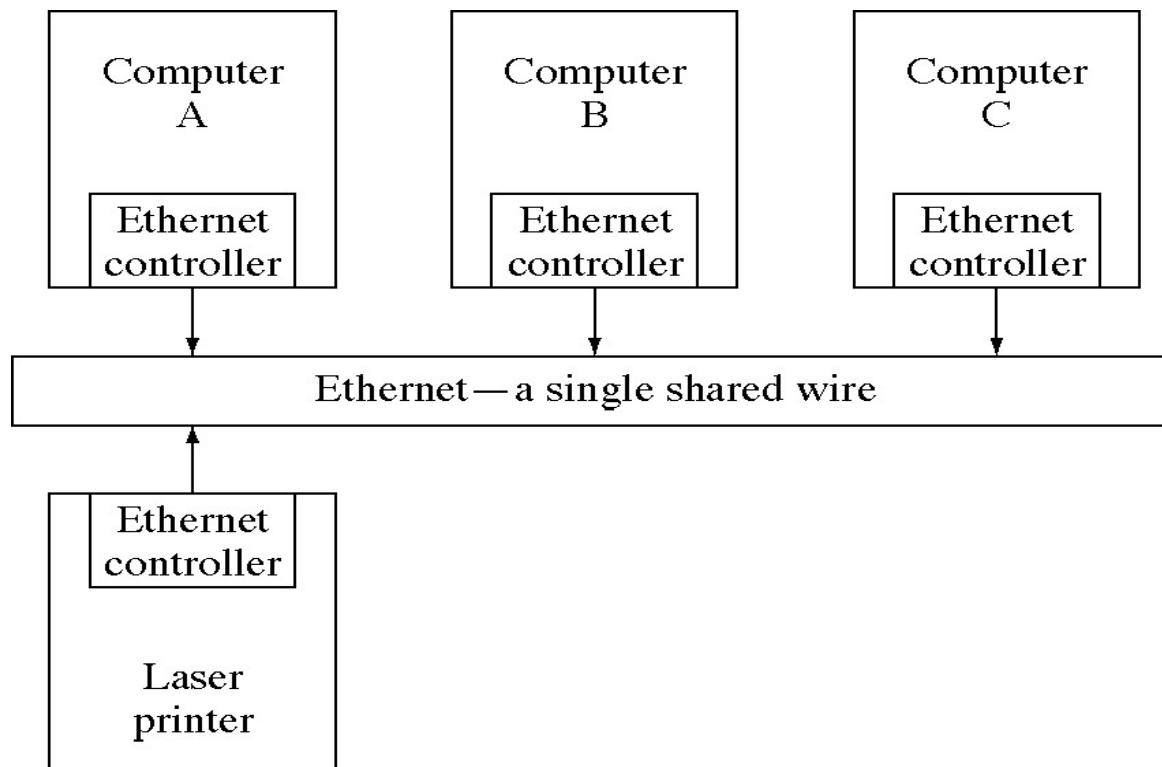
Parallel Ports

- It has 8 or 16 lines for communicating information and so can transfer 8 or 16 bits at a time.
- A **parallel port controller** – Connected to the system bus and provides one or more parallel ports.



Ethernet Devices

- **A Ethernet** – technology that allows to connect a number of computers together.
- **Packets** – packages of bits that are transferred in a unit.



Magnetic Disk Drive

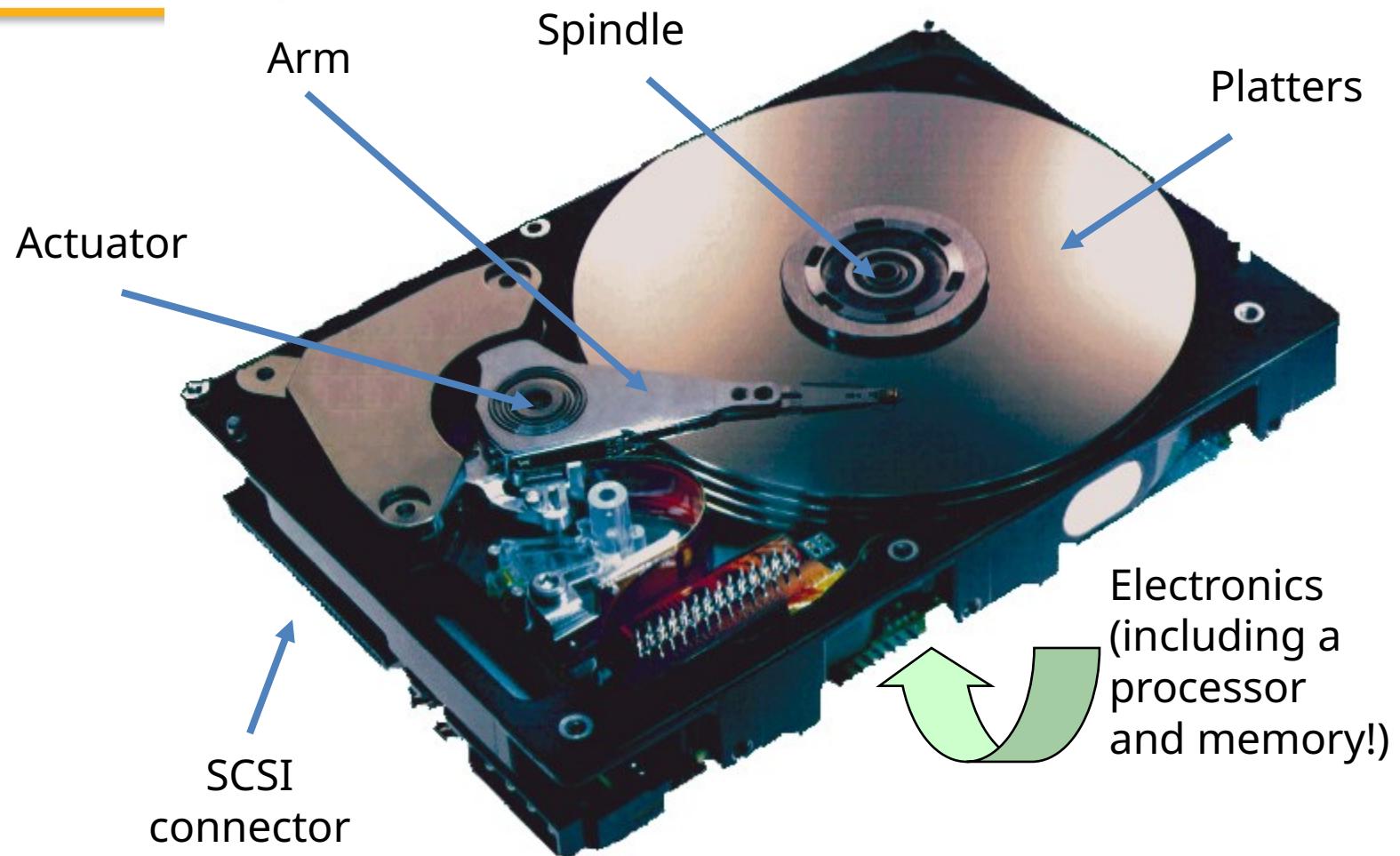
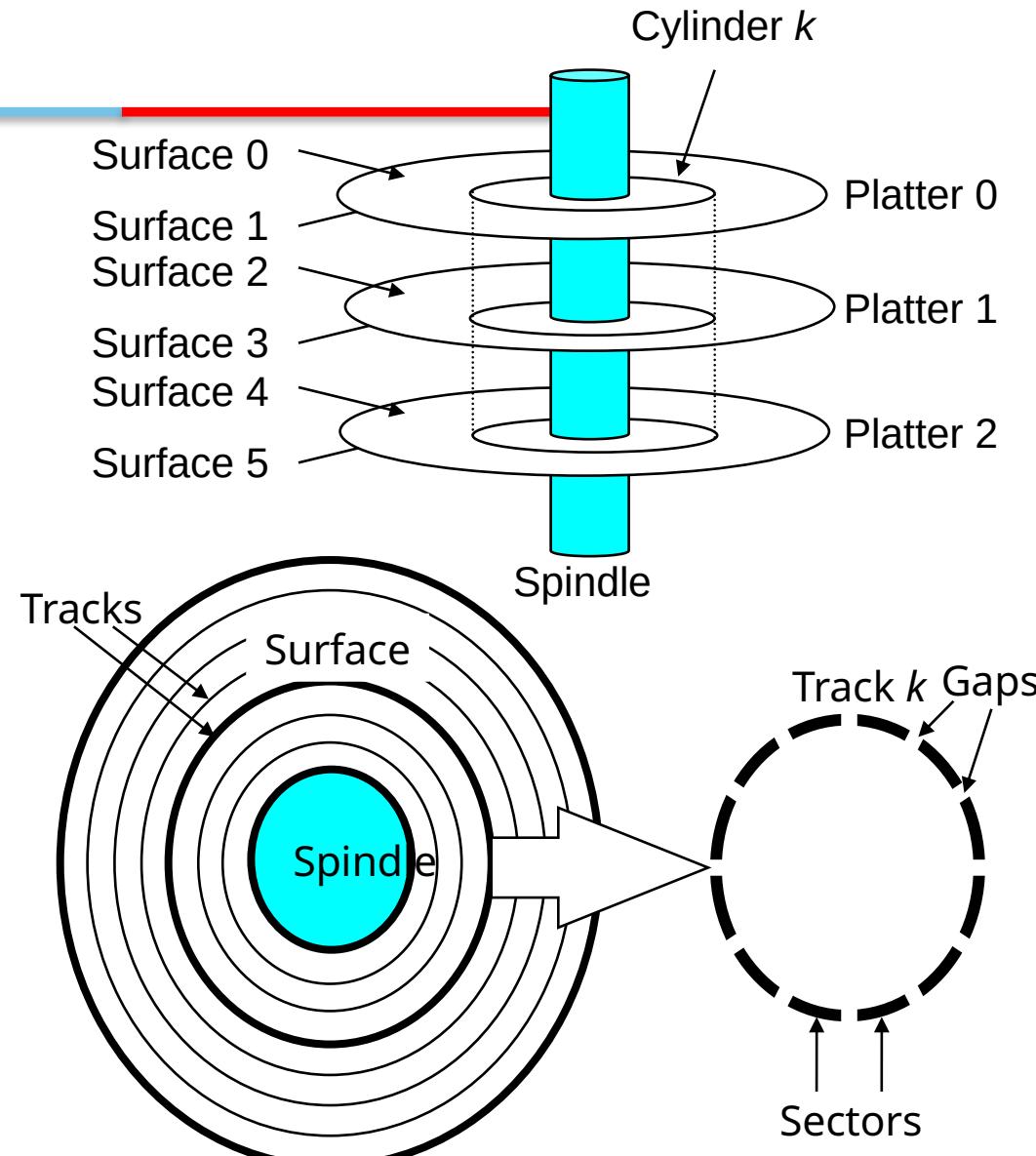


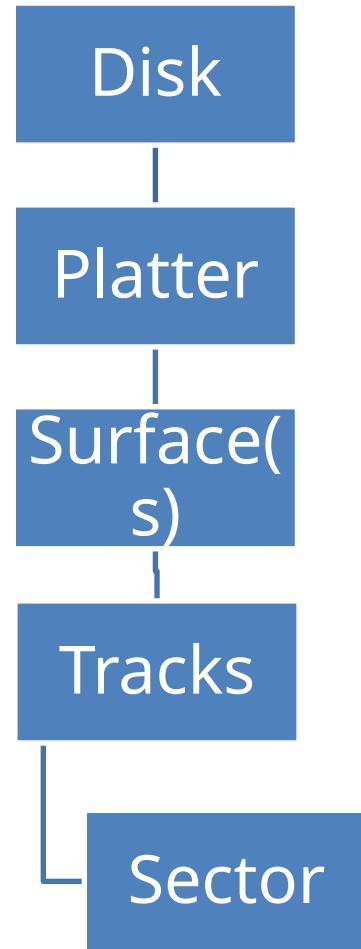
Image courtesy of Seagate Technology

Disk Geometry

- Disks consist of **platters**, each with two **surfaces**.
- Each surface consists of concentric rings called **tracks**
- Aligned tracks form a cylinder
- Each track consists of **sectors** separated by gaps.



Disk Geometry (contd)



Disk Capacity

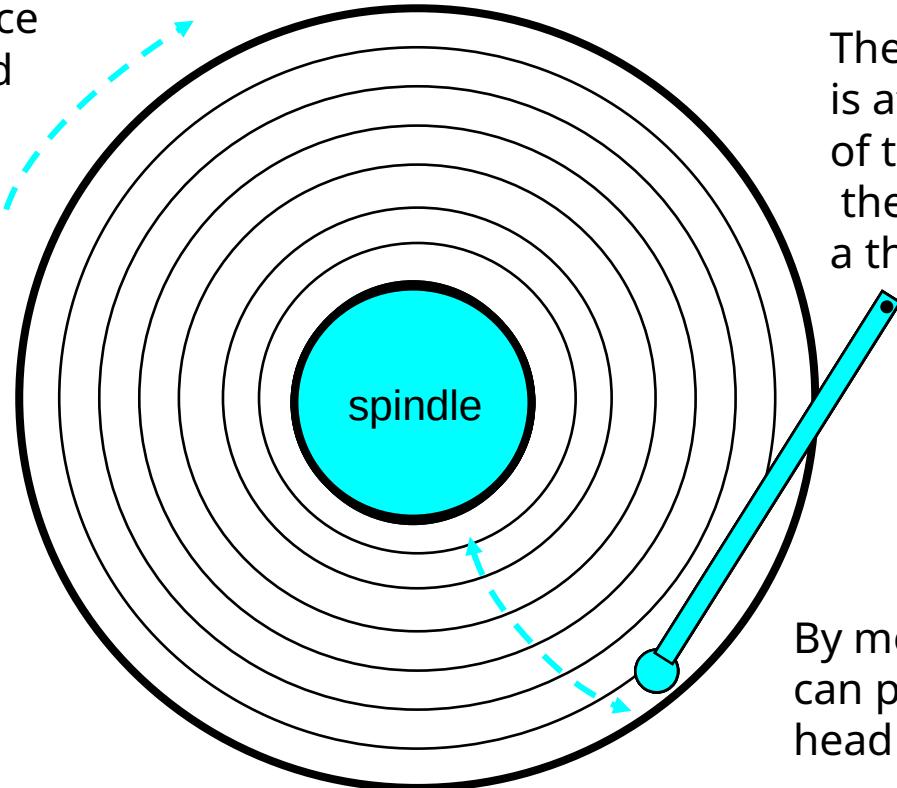
- **Capacity**: maximum number of bits that can be stored.
 - Vendors express capacity in units of gigabytes (GB /TB), where $1\text{ GB} = 2^{30}\text{ Bytes}$, $1\text{ TB} = 2^{40}\text{ Bytes}$,
- Capacity is determined by these technology factors:
 - **Recording density** (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.
 - **Track density** (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment.
 - **Areal density** (bits/in²): product of recording and track density.

Computing Disk Capacity

- Capacity = (# bytes/sector) x (avg. # sectors/track) x (# tracks/surface) x (# surfaces/platter) x (# platters/disk)
- Example:
 - 512 bytes/sector
 - 300 sectors/track (on average)
 - 20,000 tracks/surface
 - 2 surfaces/platter
 - 5 platters/disk
- Capacity = $512 \times 300 \times 20000 \times 2 \times 5$
= 30,720,000,000
= 28.61 GB

Disk Operation (Single-Platter View)

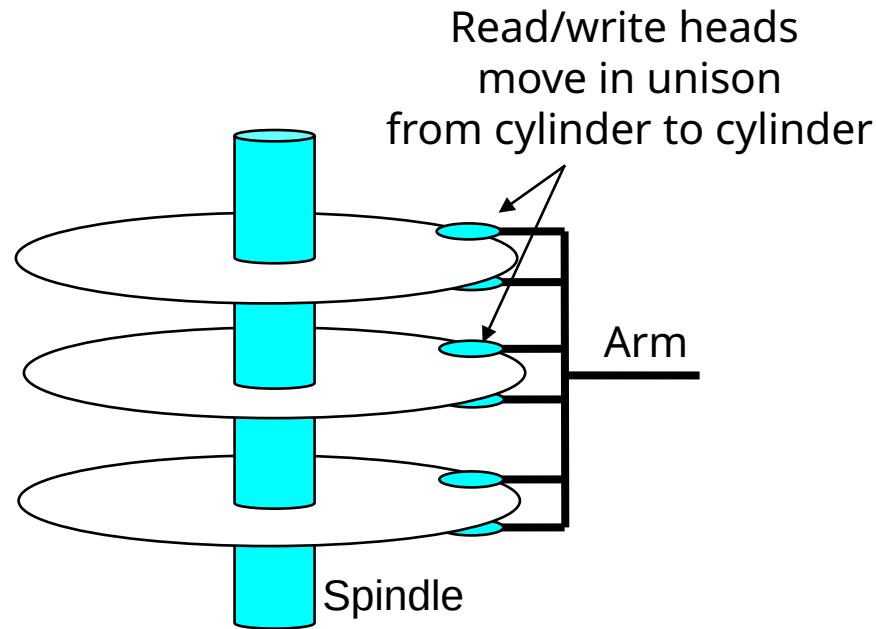
The disk surface spins at a fixed rotational rate



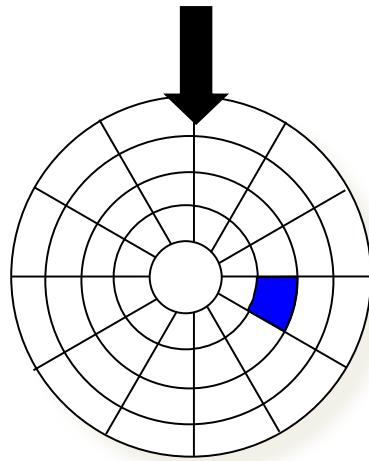
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

Disk Operation (Multi-Platter View)

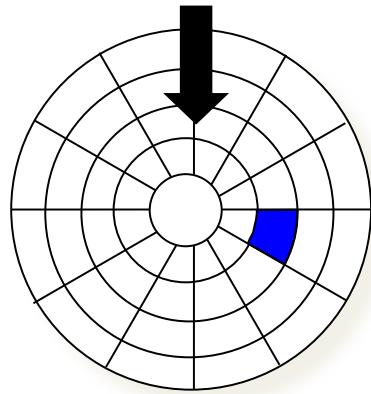


Disk Access



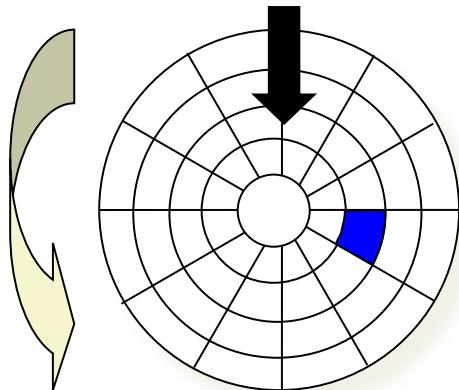
Need to access a sector
colored in blue

Disk Access



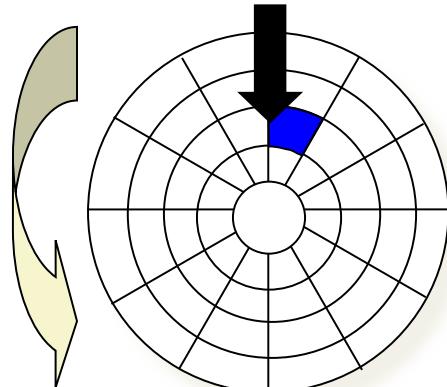
Head in position above a track

Disk Access



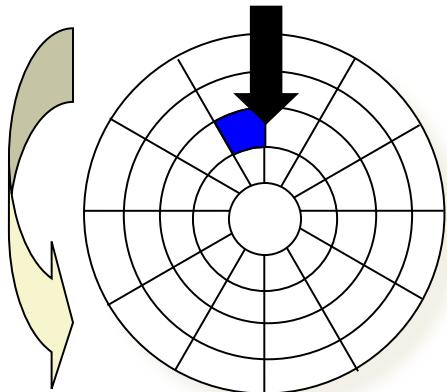
Rotate the platter in counter-clockwise direction

Disk Access – Read



About to read blue sector

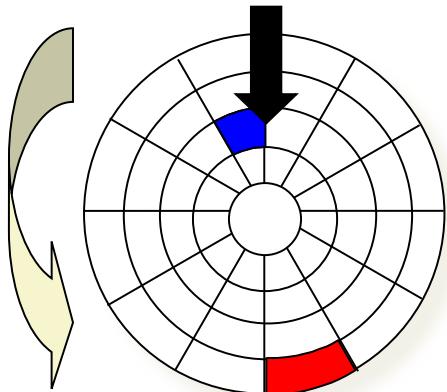
Disk Access – Read



After BLUE read

After reading blue sector

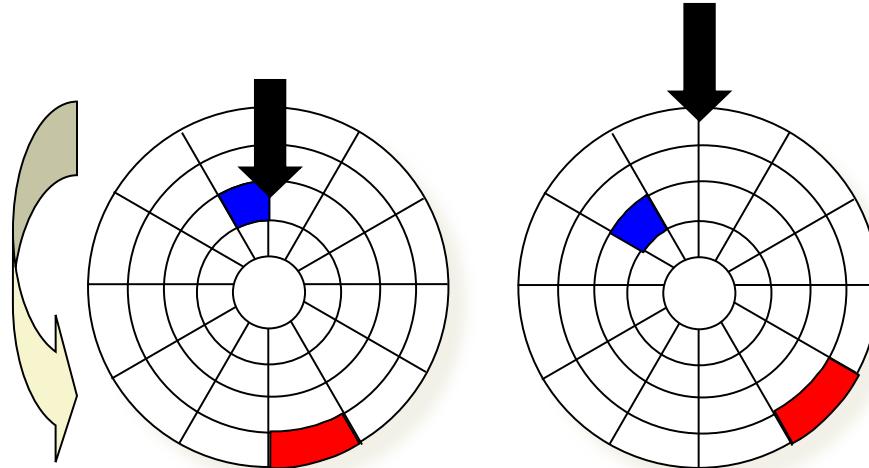
Disk Access – Read



After **BLUE** read

Red request scheduled next

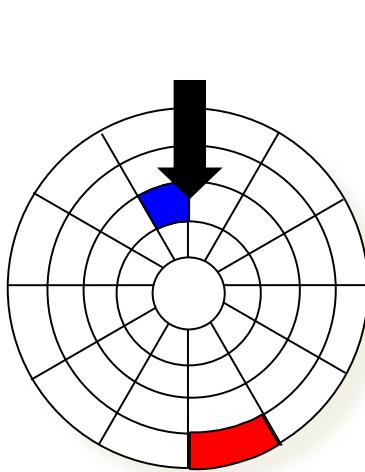
Disk Access – Seek



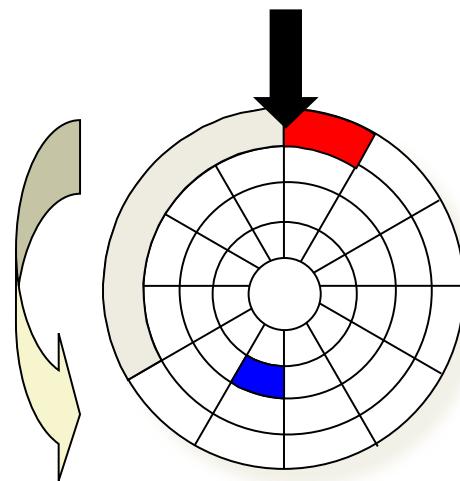
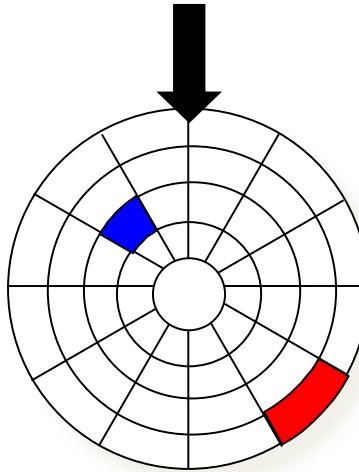
After **BLUE** read Seek for **RED**

Seek to red's track

Disk Access – Rotational Latency



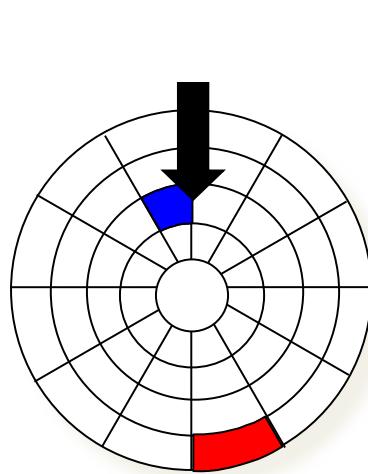
After BLUE read
Seek for RED



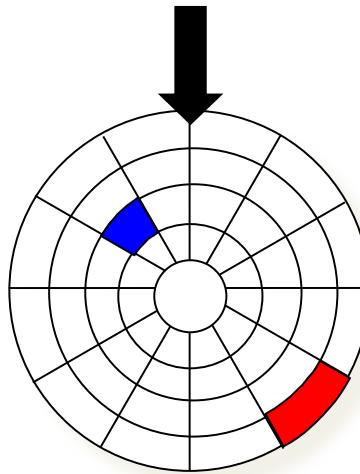
Rotational latency

Wait for red sector to rotate around

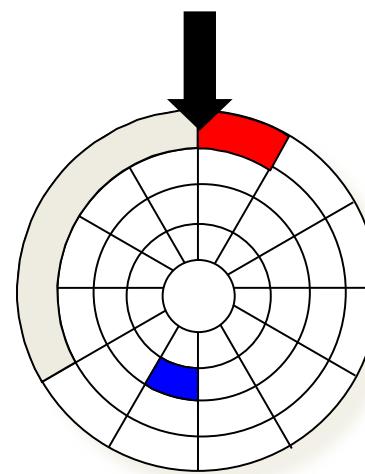
Disk Access – Read



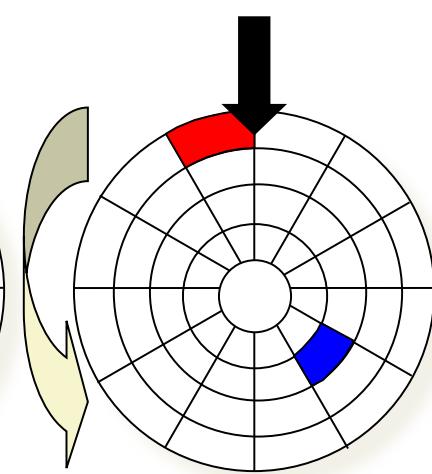
After **BLUE** read



Seek for **RED**



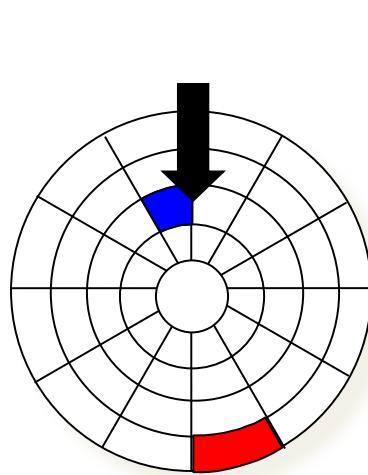
Rotational latency



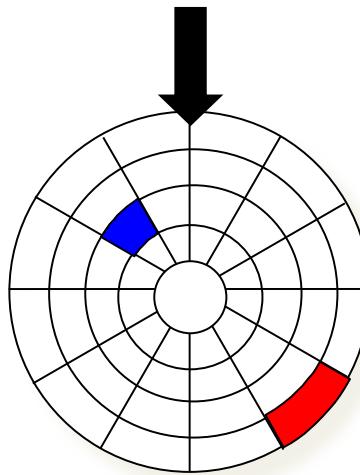
After **RED** read

Complete read of red

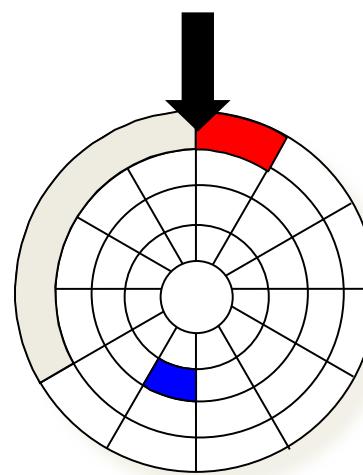
Disk Access - Access Time Components



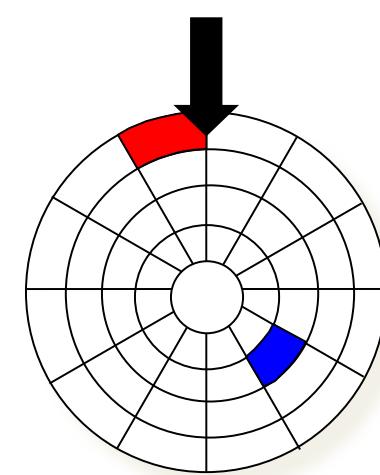
After **BLUE** read



Seek for **RED**



Rotational latency After **RED** read



After **RED** read



Data transfer



Seek



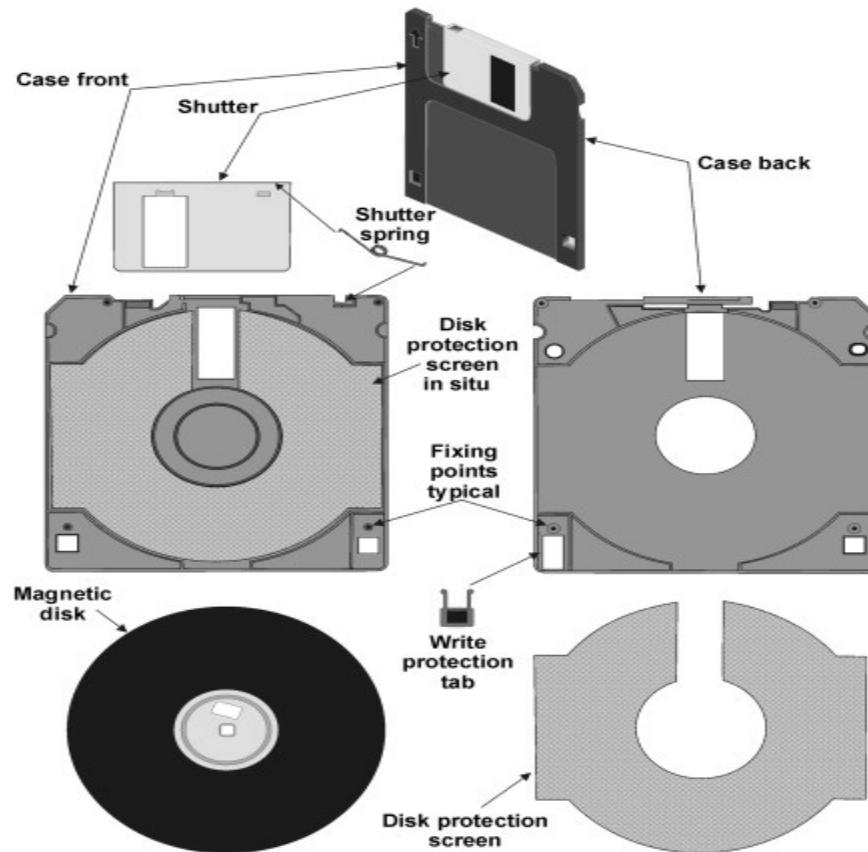
Rotational
latency



Data transfer

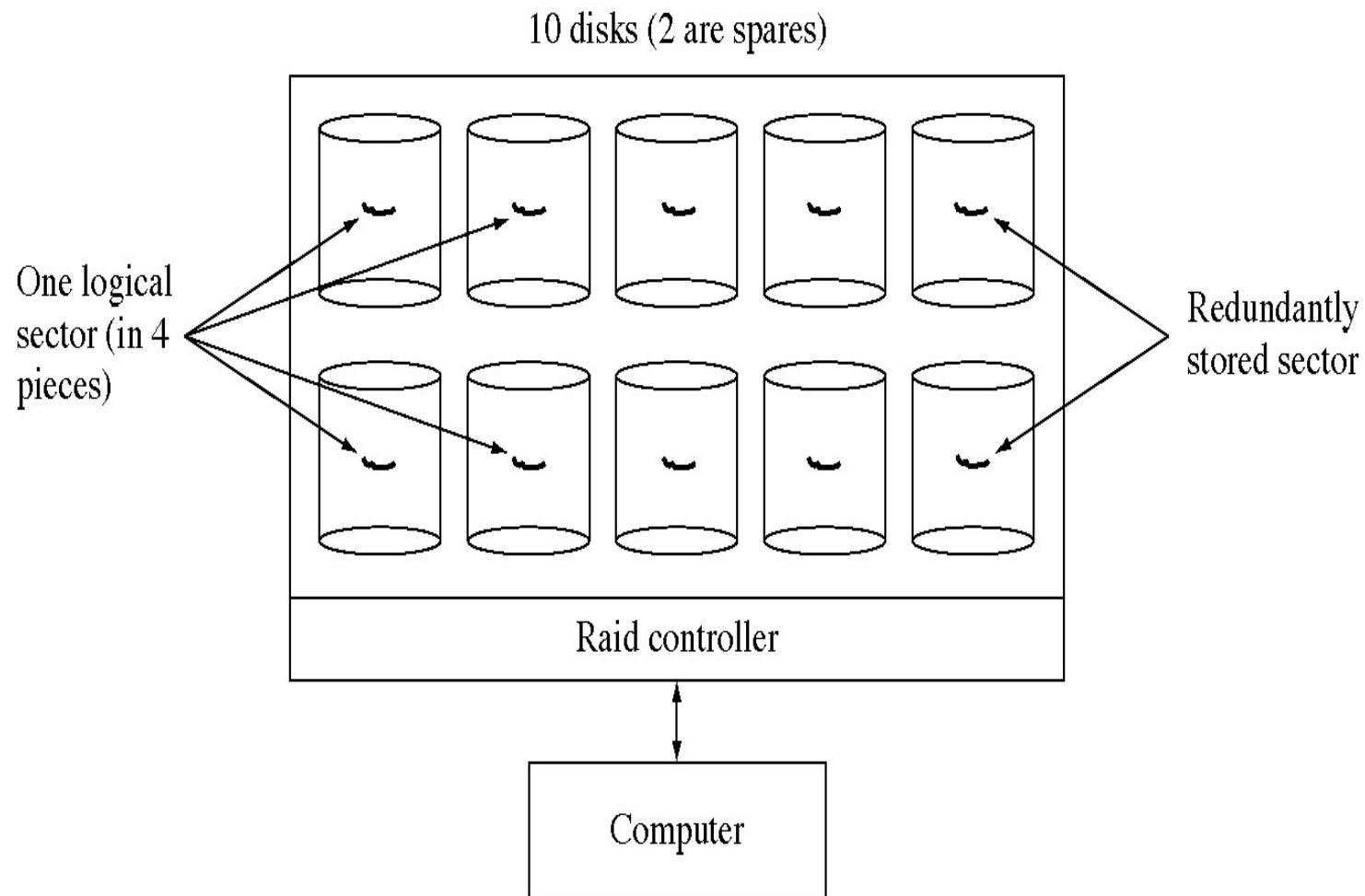
Floppy Disks

- Rigid disk, maximum speed and capacity but expensive and media are not removable.



RAID Devices

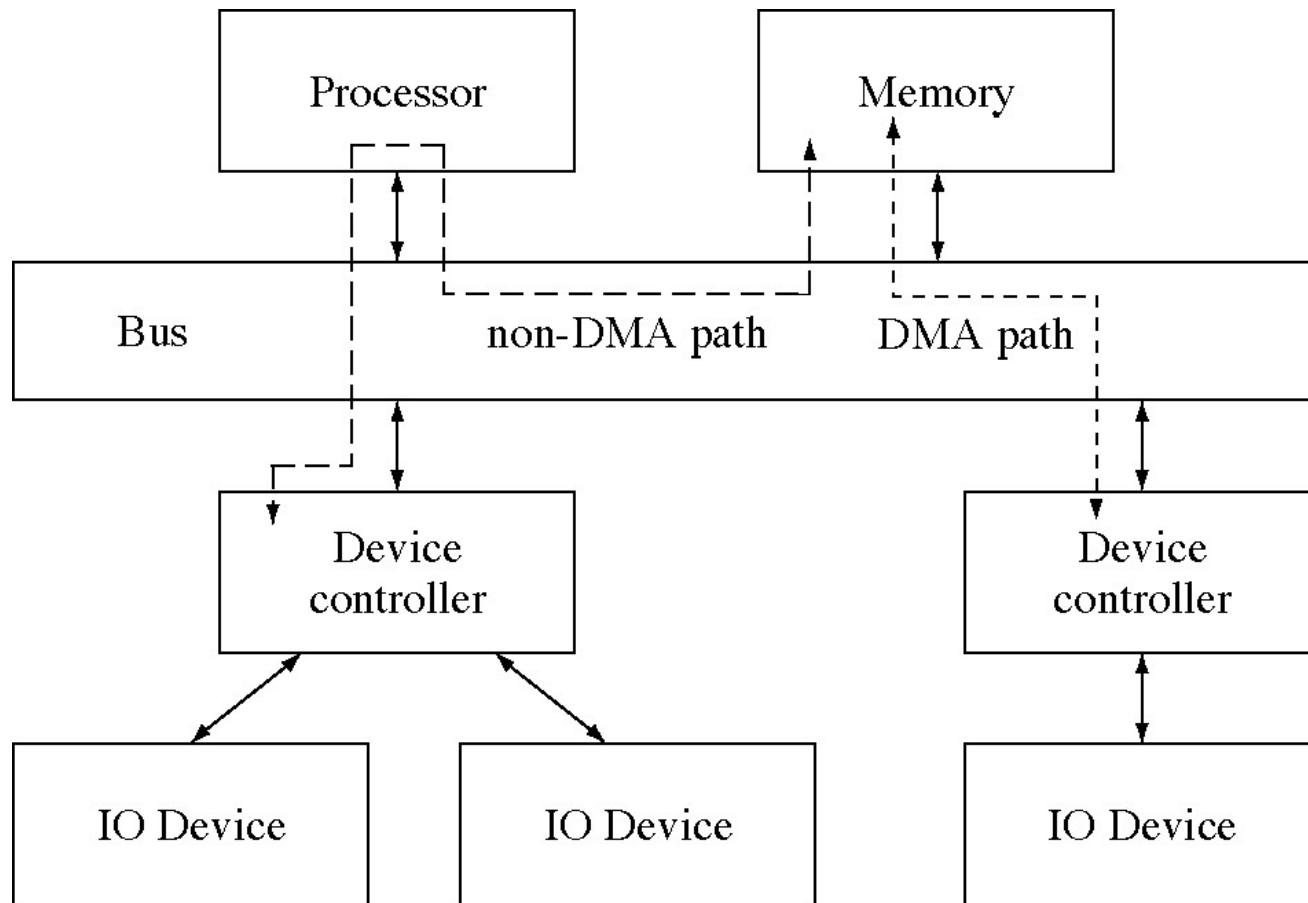
- Disk can only spin fast
 - to increase speed, parallelism is used
- ***RAID: redundant array of inexpensive disks***
 - **redundant**: RAID can be used to increase reliability through redundancy
 - **array**: RAID uses disks in parallel
 - **inexpensive**: RAID uses disks that are manufactured in the highest volume and therefore have the best performance/cost ratio



I/O Methods

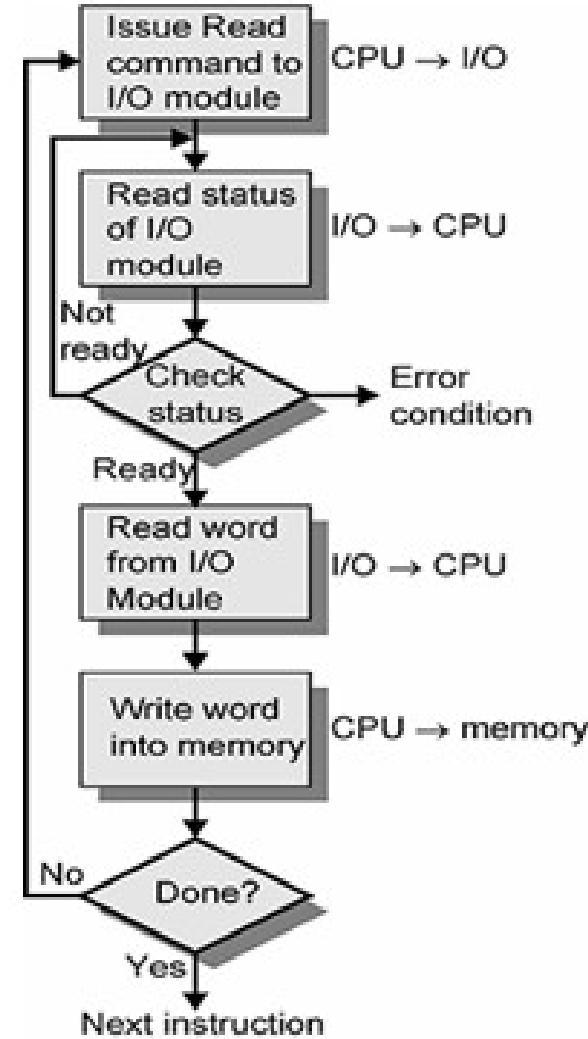
- Programmed
- Interrupt driven
- Direct Memory Access (DMA)

Device Controller and I/O Devices



Programmed I/O

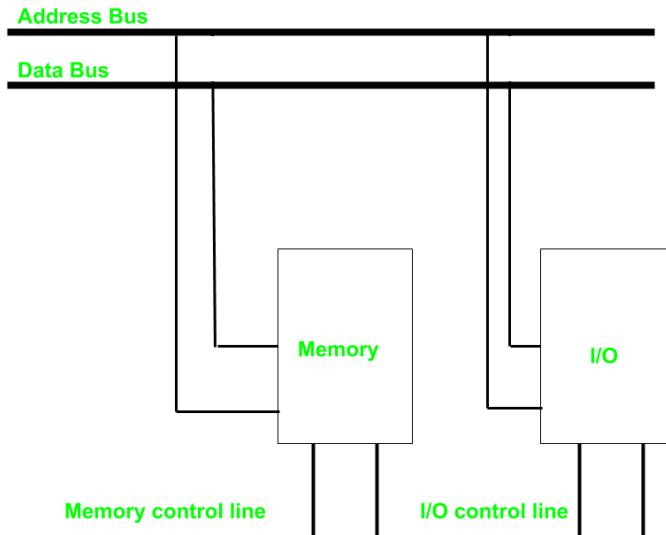
- CPU has direct control over I/O
 - Sensing status
 - Read/write commands
 - Transferring data
- CPU waits for I/O module to complete operation
- Commands
 - Control - telling module what to do
 - Test - check status
 - Read/Write
- **Wastes CPU time!!! Why?**



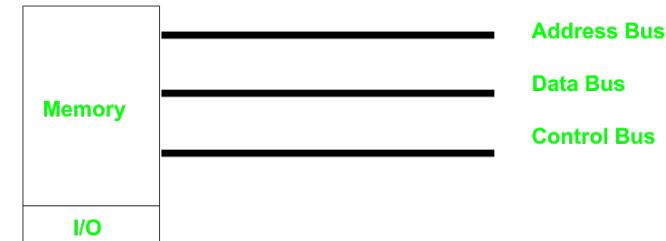
Addressing I/O Devices

- Memory mapped I/O
 - Devices and memory share an address space
 - I/O looks just like memory read/write
 - No special commands for I/O
 - Large selection of memory access commands available
- Isolated I/O
 - Separate address spaces
 - Need I/O or memory select lines
 - Special commands for I/O

Isolated I/O

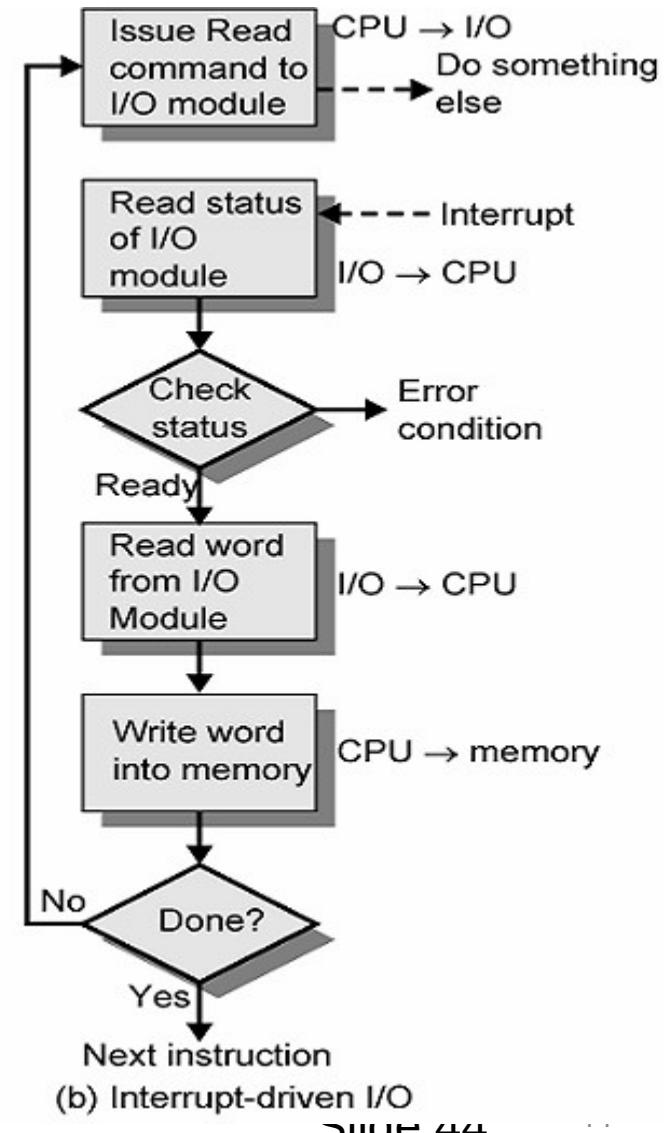


Memory mapped I/O



Interrupt Driven I/O

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU once data ready
- If interrupted:-
 - Save context (registers values)
 - Process interrupt
 - Fetch data & store



Device Identification-1

- How does processor determine which device issued the interrupt?
 - Multiple Interrupt Lines
 - Impractical to provide one line to each I/O device as number of devices increase
 - Software Poll
 - Processor branches to ISR (Interrupt service routine)
 - ISR poll each I/O module to identify which module caused the interrupt
 - Time consuming process

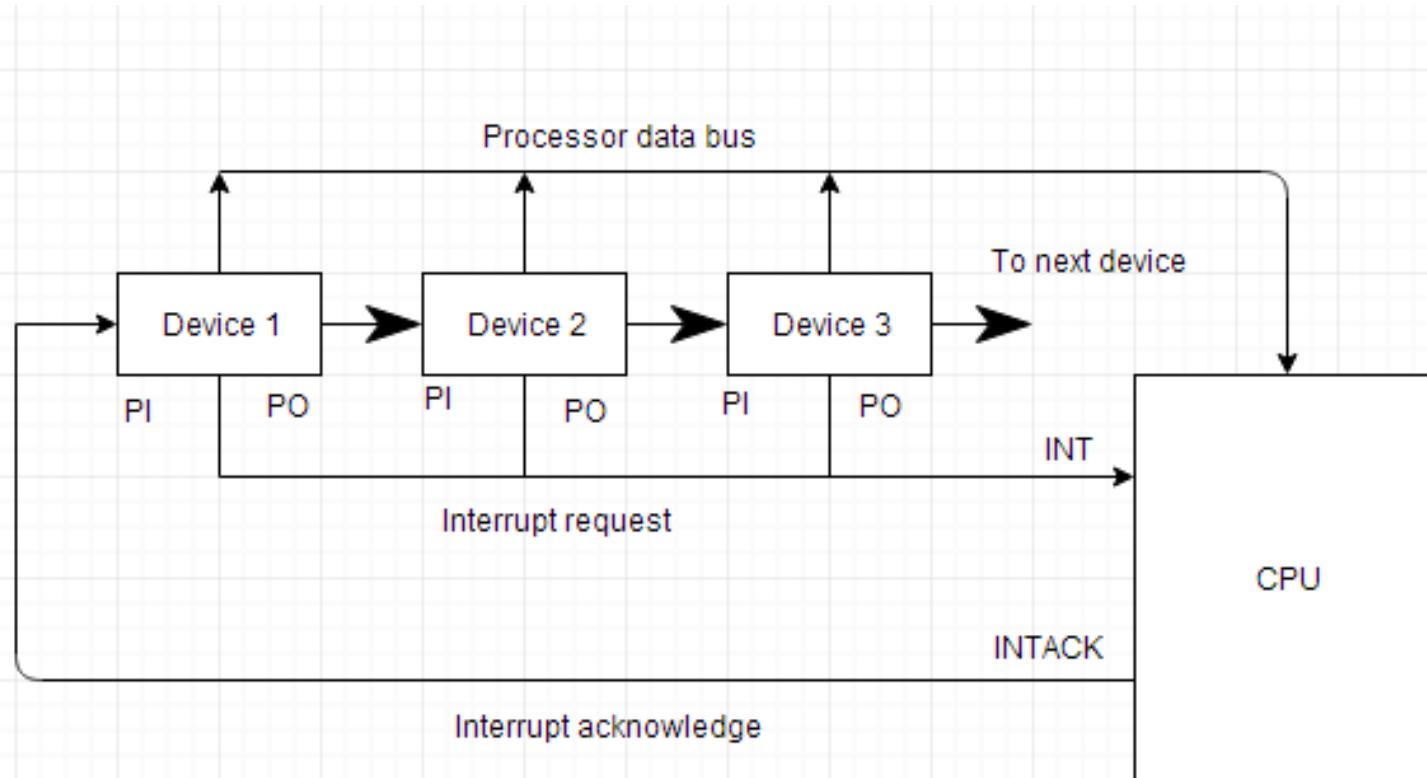
Device Identification-2

- Hardware Poll
 - All I/O modules share a common interrupt request line
 - Interrupt ACK line is Daisy changed through the modules
 - Requesting module places address of the I/O module on data bus which is called as a vector
 - Technique is called as vectored Interrupt

Daisy Chaining

- This way of deciding the interrupt priority consists of serial connection of all the devices which generates an interrupt signal.
- The device with the highest priority is placed at the first position followed by lower priority devices and the device which has lowest priority among all is placed at the last in the chain.

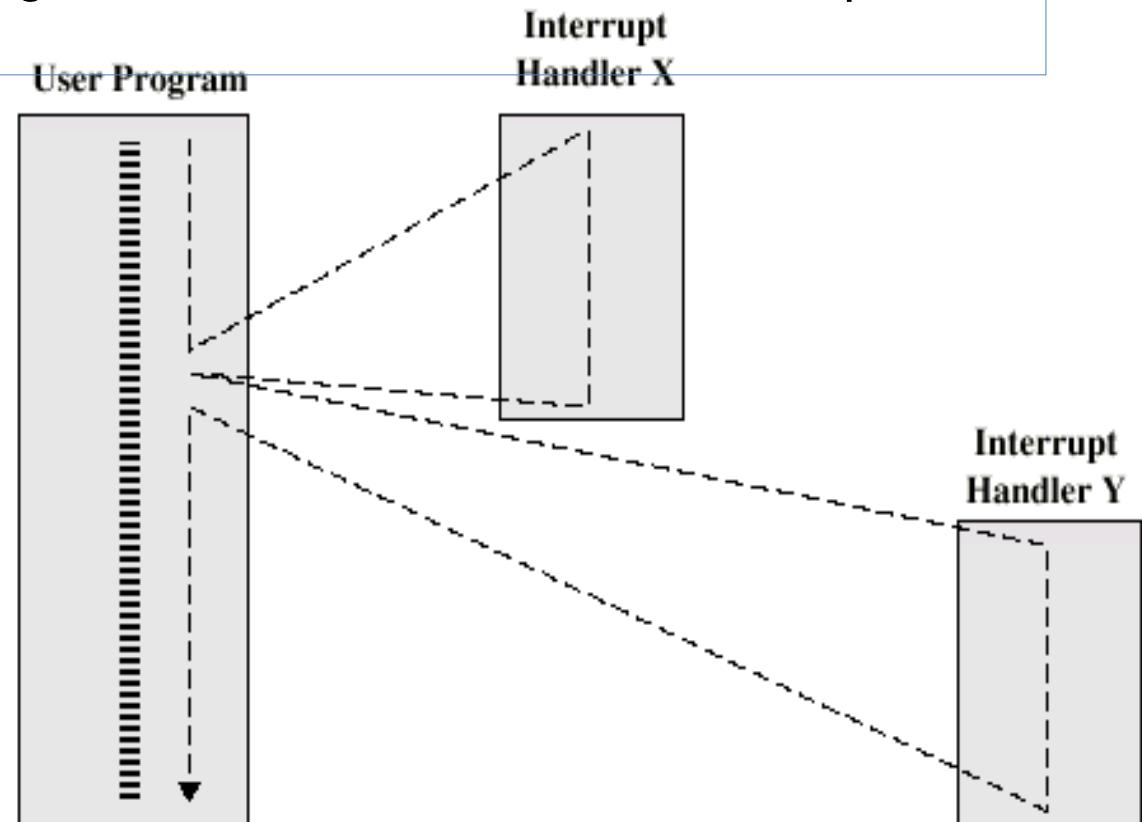
Daisy Chaining



Multiple Interrupts - Sequential

Processor will ignore further interrupts whilst processing one interrupt

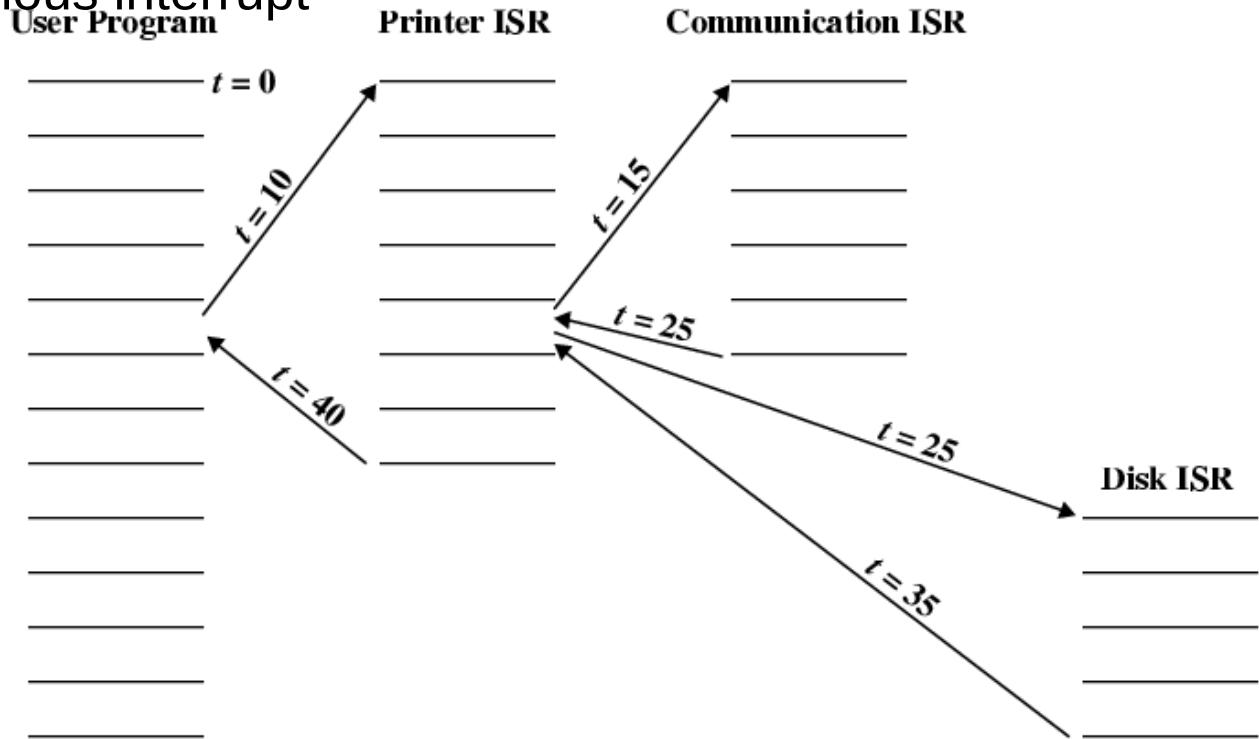
Interrupts remain pending and are checked after first interrupt has been processed



Multiple Interrupts: Priority

Low priority interrupts can be interrupted by higher priority interrupts

When higher priority interrupt has been processed, processor returns to previous interrupt



Review Questions

- When a device interrupt occurs, how does the processor determine which device issued the interrupt?
- What is context switching?
- How programmed I/O is different from Interrupt Driven I/O?
- How software polling is different from Daisy chain method to handle multiple interrupts?

PROGRAMMED I/O

- Data are exchanged between the processor and the I/O module.
- Direct control of the I/O operation. Sensing device status, sending a read or write command, and transferring the data.
- With interrupt-driven I/O, the processor issues an I/O command, executes other instructions, and is interrupted by the I/O module when the latter has completed its work.
- **DMA** - With both programmed and interrupt I/O, the processor extracts data from main memory for output and stores data in main memory for input.

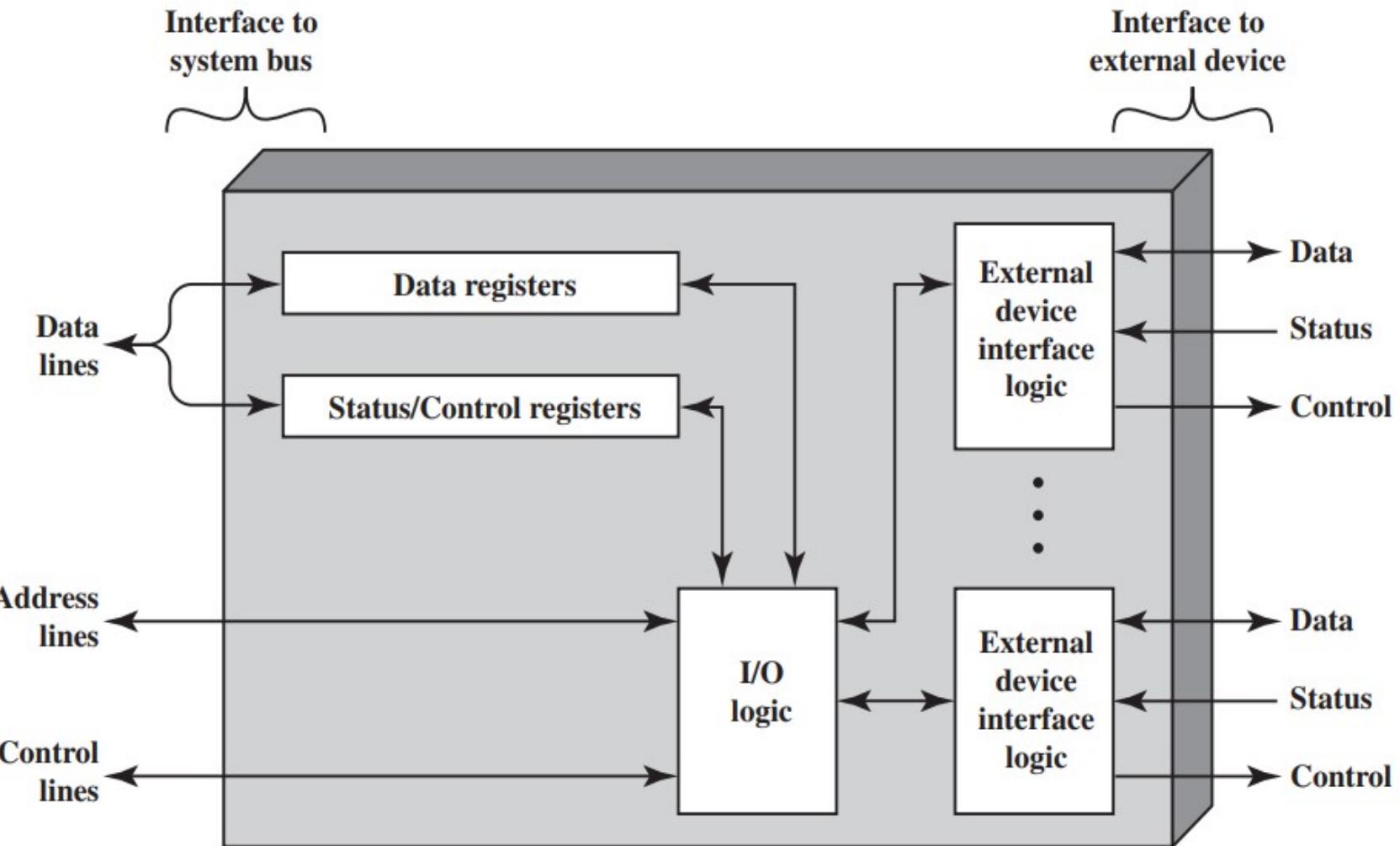
I/O Techniques

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Overview of Programmed I/O

- The I/O module will perform the requested action and then set the appropriate bits in the I/O status register.
- I/O module does not interrupt the processor.
- It periodically checks the status of the I/O module until it finds that the operation is complete.

Block Diagram of an I/O Module



I/O Commands

Four types of I/O commands:

- **Control**
- **Test**
- **Read**
- **Write**

- The instructions are easily mapped into I/O commands
- A simple **one-to-one relationship**
- The form of the instruction depends on the way in which external devices are addressed
- **Memory- mapped I/O:** single address space for memory locations and I/O devices, same machine instructions to access both memory and I/O devices, a single read line and a single write line are needed on the bus.
- **Isolated I/O:** full range of addresses may be available for both, , the bus may be equipped with memory read and write plus input and output command lines, the bus may be equipped with memory read and write plus input and output command lines.

MEMORY MAPPED IO

A method to perform input/output (I/O) operations between the central processing unit (CPU) and peripheral devices in a computer that uses one address space for memory and IO devices

Uses the same address space for both memory and IO devices

As the memory mapped IO uses one address space for both IO and memory, the available addresses for memory are minimum

Uses the same instructions for both IO and memory operations

Less efficient

IO MAPPED IO

A method to perform input/output (I/O) operations between the central processing unit (CPU) and peripheral devices in a computer that uses two separate address spaces for memory and IO device

Uses two separate address spaces for memory and IO device

All the addresses can be used by the memory

Uses separate instructions for read and write operations in IO and memory

More efficient

INTERRUPT-DRIVEN I/O

- The processor issues a **READ command** and then it goes off. At the end of each instruction cycle, the **processor checks for interrupts**. When interrupts occurs, the processor saves the context of the current program and processes the interrupt. The processor reads the word of data from the I/O module and stores it in memory. It then restores the context of the program it was working on and resumes execution.
- Interrupt I/O is **more efficient** because it **eliminates needless waiting**.
- **Disadvantage of Interrupt I/O:** **consumes a lot of processor time** (every word of data that goes from memory to I/O module or from I/O module to memory must pass through the processor)

Design Issues

- Two design issues arise in implementing interrupt I/O.
 - How does the processor determine which device issued the interrupt?
 - If multiple interrupts have occurred, how does the processor decide which one to process?
- Four general categories of techniques are in common use:
 - Multiple interrupt lines
 - Software poll
 - Daisy chain (hardware poll, vectored)
 - Bus arbitration (vectored)

INTERRUPT VERSUS POLLING



INTERRUPT

An event that is triggered by external components other than the CPU that alerts the CPU to perform a certain action

When an interrupt occurred, the interrupt handler executes

Can occur at any time

Interrupt-request line indicates that device needs a service

Does not waste much CPU cycles

It is inefficient when the device interrupts the CPU frequently

POLLING

An activity of sampling the status of an external device by a client program as a synchronous activity

In polling, the CPU provide the service

Occurs at regular intervals

Command ready bit indicates the device needs a service

Wastes lot of CPU cycles

It is inefficient in polling, when the CPU rarely finds requests from the devices

Drawbacks of Interrupt Driven and Programmed I/O

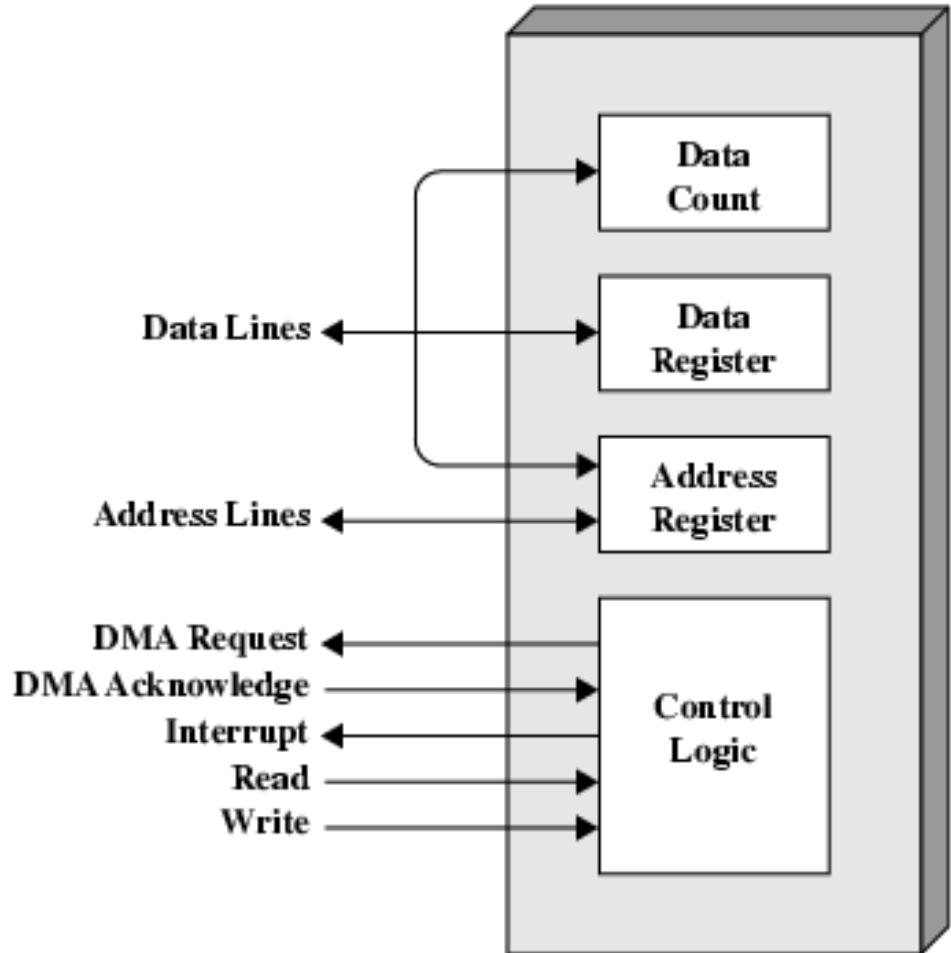
- Requires active CPU Intervention
 - Transfer rate is limited by the speed with which the processor can test and service the device
 - CPU is tied up in managing an I/O transfer
- Using programmed I/O processor can move data at higher rate at the cost of doing nothing else
- Interrupt I/O frees up the processor at the expense of the I/O transfer rate

Direct Memory Access (DMA)

- A special control unit can be provided to allow transfer of a block of data directly between an external device and the main memory.
- No continuous intervention by the processor is required
- This control circuit can be part of I/O device interface and called as DMA controller
- Good for moving large volume of data

DMA Module and Operation

- CPU tells DMA controller:-
 - Read/Write
 - Device address
 - Starting address of memory block for data
 - Amount of data to be transferred
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished

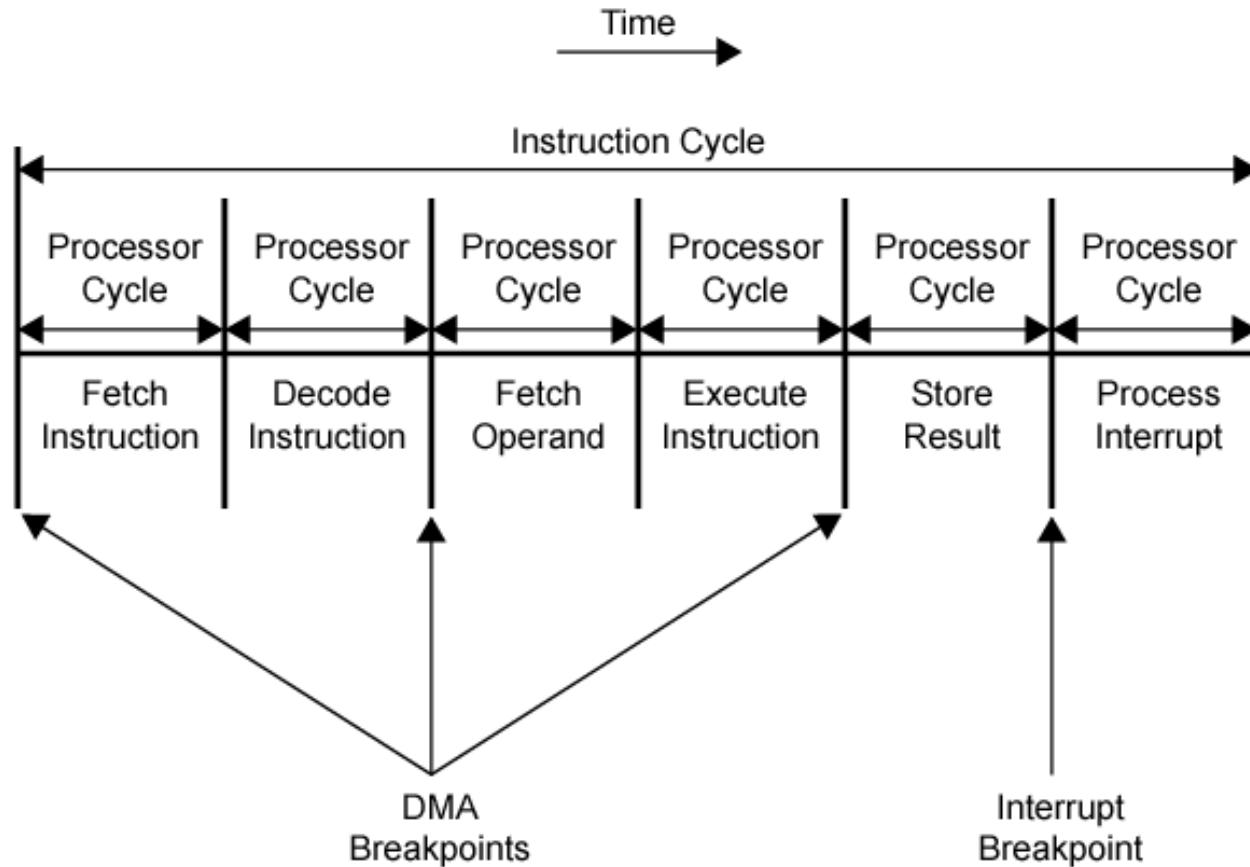


DMA Transfer Method: Cycle Stealing



- Memory accesses by the processor and by the DMA controllers are interwoven
- DMA controller takes over bus for a cycle
- Transfer of one word of data
- Not an interrupt
 - CPU does not switch context
- CPU suspended just before it accesses bus
 - i.e. before an operand or data fetch or a data write
- Slows down CPU but not as much as CPU itself doing transfer

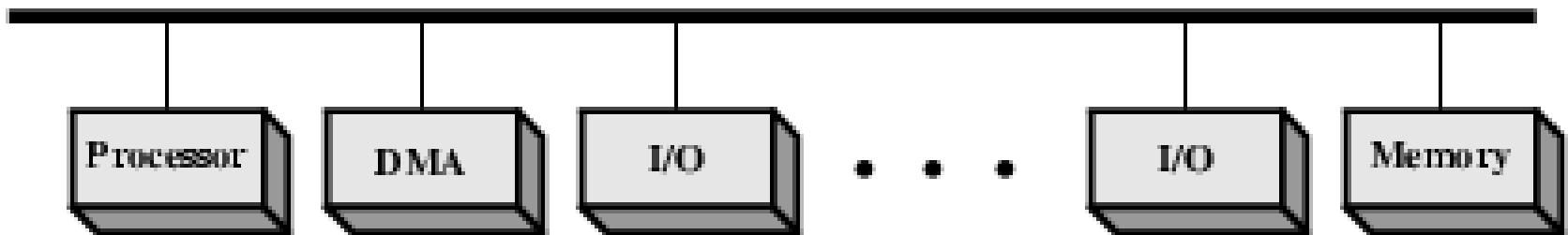
DMA and Interrupt Breakpoints During an Instruction Cycle



DMA Transfer Method: Burst Mode

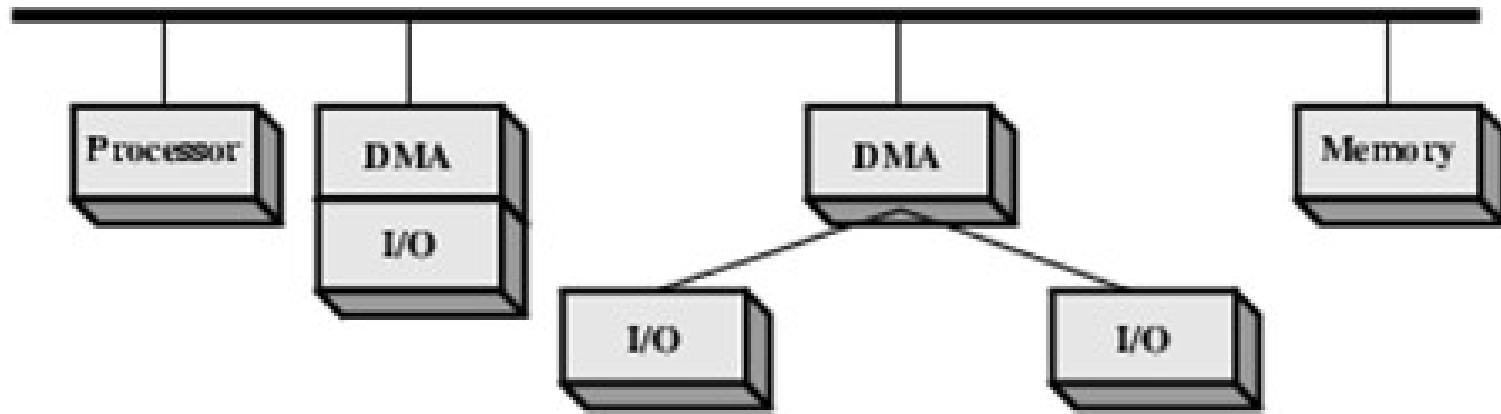
- DMA controller is given exclusive access to the main memory to transfer a block of data without interruption
- Most DMA controllers incorporate a data storage buffer
- It reads a block of data using **burst mode** from the main memory and stores it into its input buffer
- Then it is transferred to the desired I/O device

DMA Configurations (1)



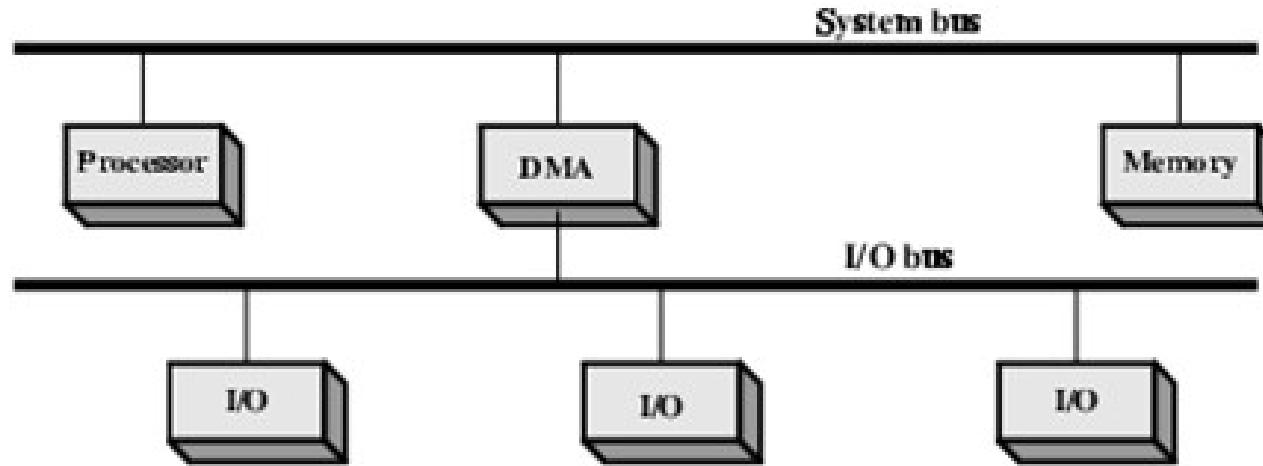
- Single Bus, Detached DMA controller
- Each transfer uses bus twice
 - I/O to DMA then DMA to memory
- CPU is suspended twice!

DMA Configurations (2)



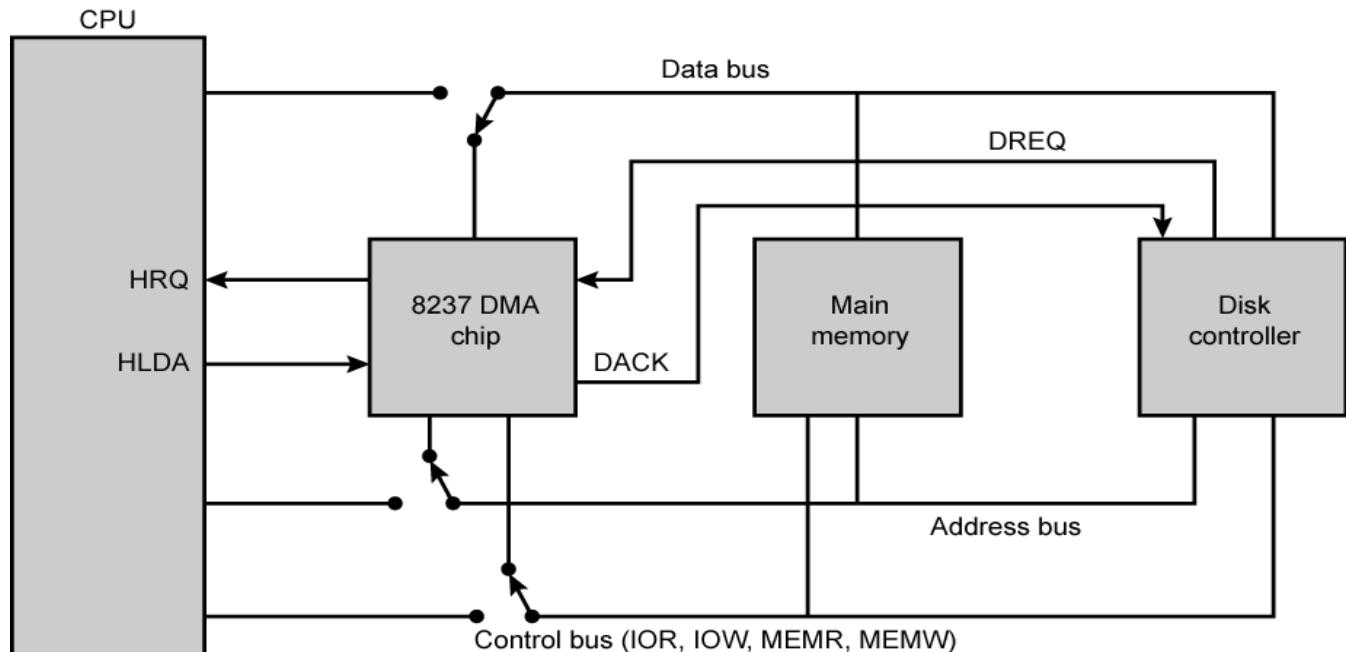
- Single Bus, Integrated DMA controller
- Controller may support >1 device
- Each transfer uses bus once
- CPU is suspended once!

DMA Configurations (3)



- Separate I/O Bus
- Bus supports all DMA enabled devices
- Each transfer uses system bus once
- CPU is suspended once

Intel 8237A DMA Controller

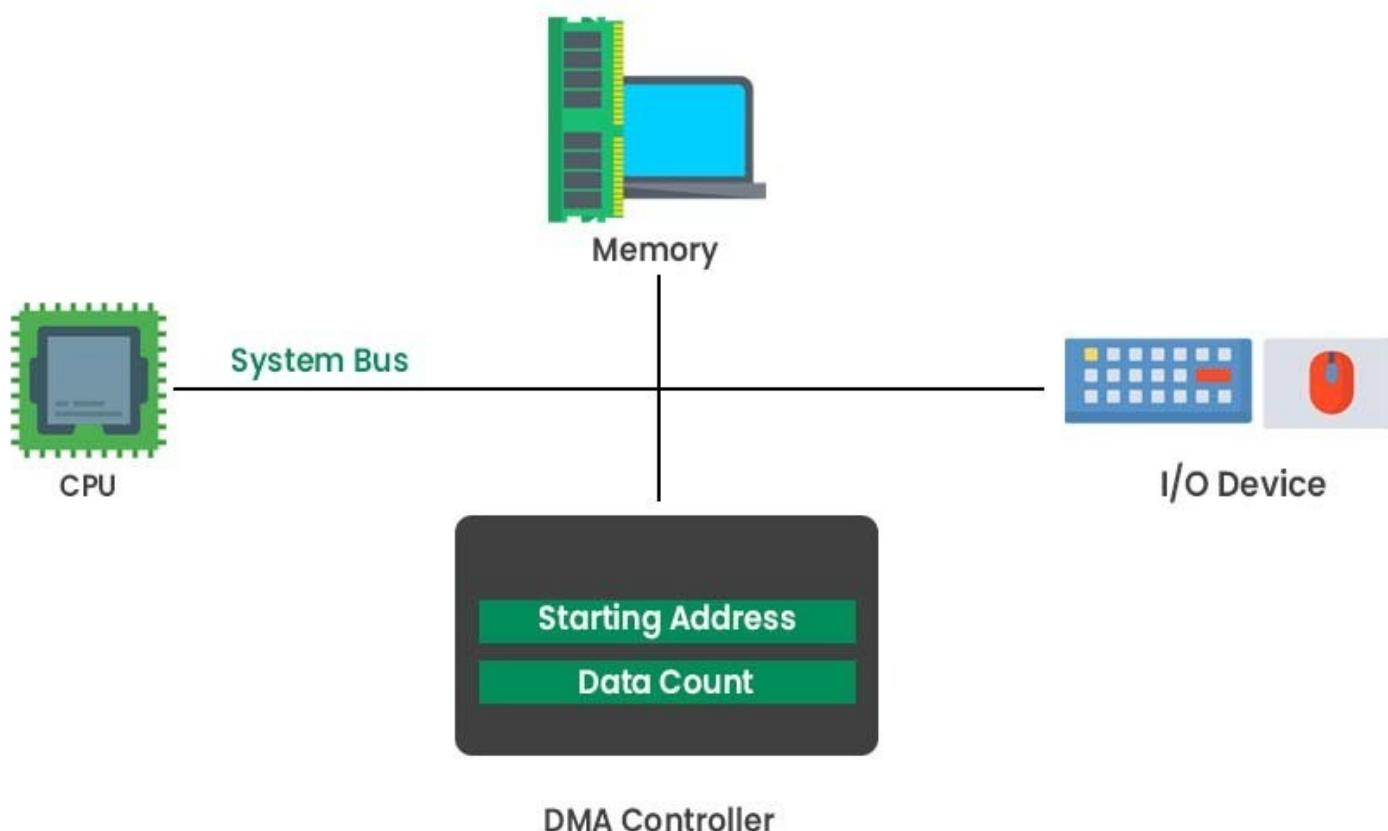


DACK = DMA acknowledge
 DREQ = DMA request
 HLDA = HOLD acknowledge
 HRQ = HOLD request

- DMA basically stands for Direct Memory Access.
- It is a process which enables data transfer between the Memory and the IO (Input/ Output) device without the need of or you can say without the involvement of CPU during data transfer.

Working of DMA

- For DMA, you basically need a hardware called DMAC (Direct Memory Access Controller) which will help in the throughout process of data transfer between the Memory and IO device directly.
- First what happens is IO device sends the DMA request to DMA Controller, then further DMAC device sends HOLD signal to CPU by which it asks CPU for several information which are needed while transferring data.
- CPU then shares two basic information with DMAC before the Data transfer which are: Starting address (memory address starting from where data transfer should be performed) and Data Count (no of bytes or words to be transferred).
- CPU then sends HLDACK (Hold Acknowledgement) back to DMAC illustrating that now DMAC can successfully pass on the information.
- Then further DMAC shares the DMA ACK (DMA Acknowledgement) to the IO device which would eventually let IO device to access or transfer the data from memory in a direct and efficient manner.



Modes of DMA Transfer :

- Now after getting some brief idea about DMA and its working it's the time to analyze Modes of DMA Transfer.
- During the DMA Transfer CPU can perform only those operation in which it doesn't require the access of System Bus which means mostly CPU will be in blocked state.
- For how much time CPU remains in the blocked state or we can say for how much time CPU will give the control of DMAC of system buses will actually depend upon the following modes of DMA Transfer and after that CPU will take back control of system buses from DMAC.
- Burst mode, cycle stealing mode and interleaving mode

Burst Mode

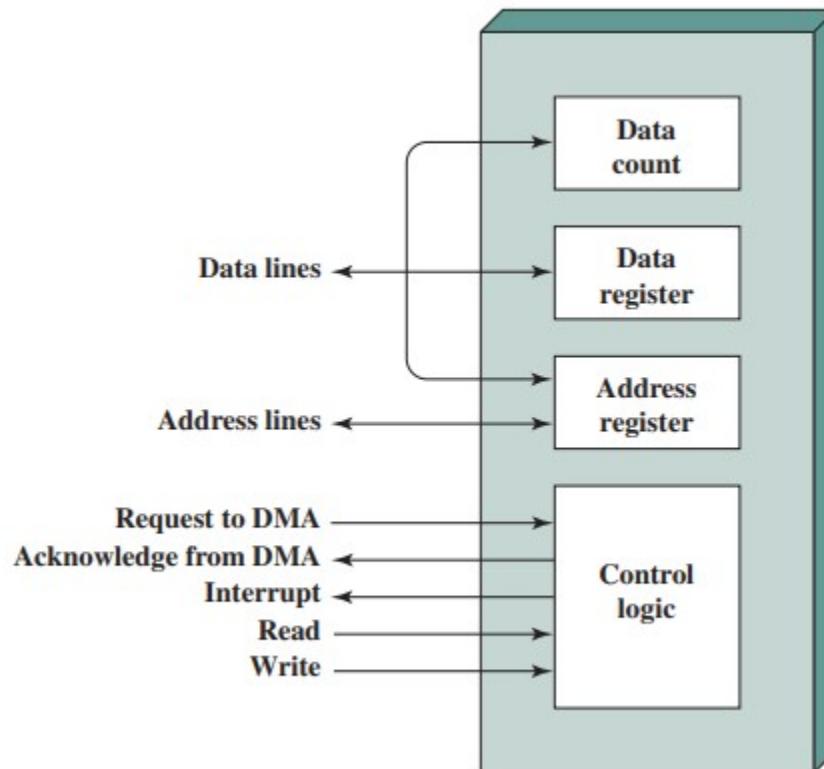
- In this mode Burst of data (entire data or burst of block containing data) is transferred before CPU takes control of the buses back from DMA.
- This is the quickest mode of DMA Transfer since at once a huge amount of data is being transferred.
- Since at once only the huge amount of data is being transferred so time will be saved in huge amount.

DIRECT MEMORY ACCESS (DMA)

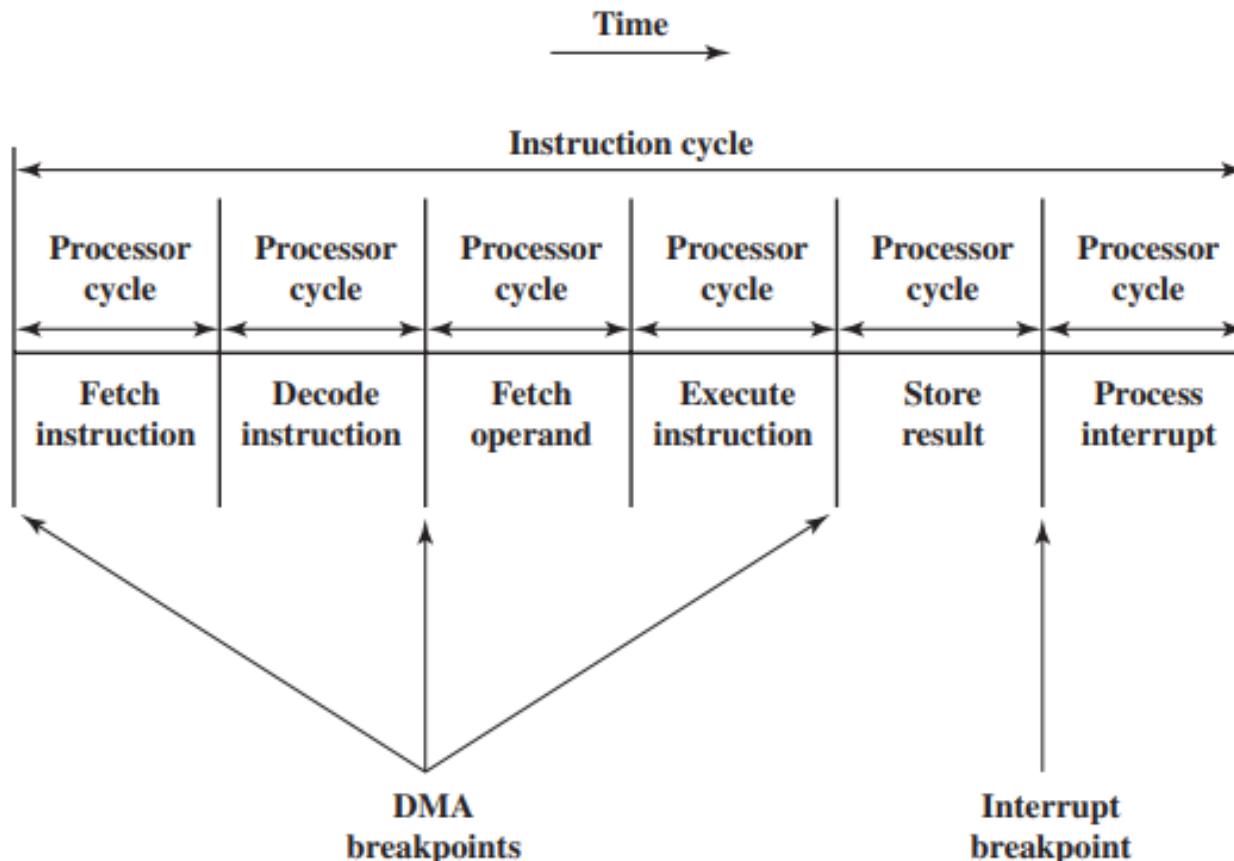
- Drawbacks of Programmed and Interrupt-Driven I/O:
 - The I/O transfer rate is limited by the speed with which the processor can test and service a device.
 - The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.
- When large volumes of data are to be moved, a more efficient technique is required: direct memory access (DMA).

DMA Function

- An additional module on the system bus
- Capable of mimicking the processor
- **Cycle stealing** - the DMA module in effect steals a bus cycle



DMA and Interrupt Breakpoints during an Instruction Cycle



- The processor is suspended just before it needs to use the bus.
- The DMA module transfers one word and returns control to the processor.
- The processor pauses for one bus cycle. The overall effect is to cause the processor to execute more slowly.
- For a multiple-word I/O transfer, DMA is more efficient than interrupt-driven or programmed I/O.



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 10

Course design by - Dr. Lucy J. Gudino &

Dr Chandrasekhar

Faculty - Dr Thangakumar J

WILP & Department of CS & IS



BITS Pilani
Pilani Campus

Operating systems - I/O Abstraction (cont..)



Last Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• I/O System Hardware<ul style="list-style-type: none">◦ Device Controllers◦ Types of devices : Terminal Devices, Communication Devices and Disk Devices• Data transfer schemes• Programmed I/O, Interrupt Driven I/O, DMA	<p>T1 (14.1 – 14.4)</p> <p>R6 (7.3-7.5)</p>

INPUT OUTPUT - STORY LINE

1

POLLING

- CHECK
- INITIATE
- COMPLETE

3

DIRECT MEMORY ACCESS

- INPUT OUTPUT IN BLOCKS
- WHICH DEVICE, HOW MANY BLOCKS, MEMORY LOCATION.
- CPU “interrupted” to check for errors; assigns next program

2

INTERRUPTIONS

- SAVE CONTEXT IN MEMORY
- INTERRUPT SERVICE ROUTINE
- LOOKUP TO EXECUTE INTERRUPT SERVICE ROUTINE

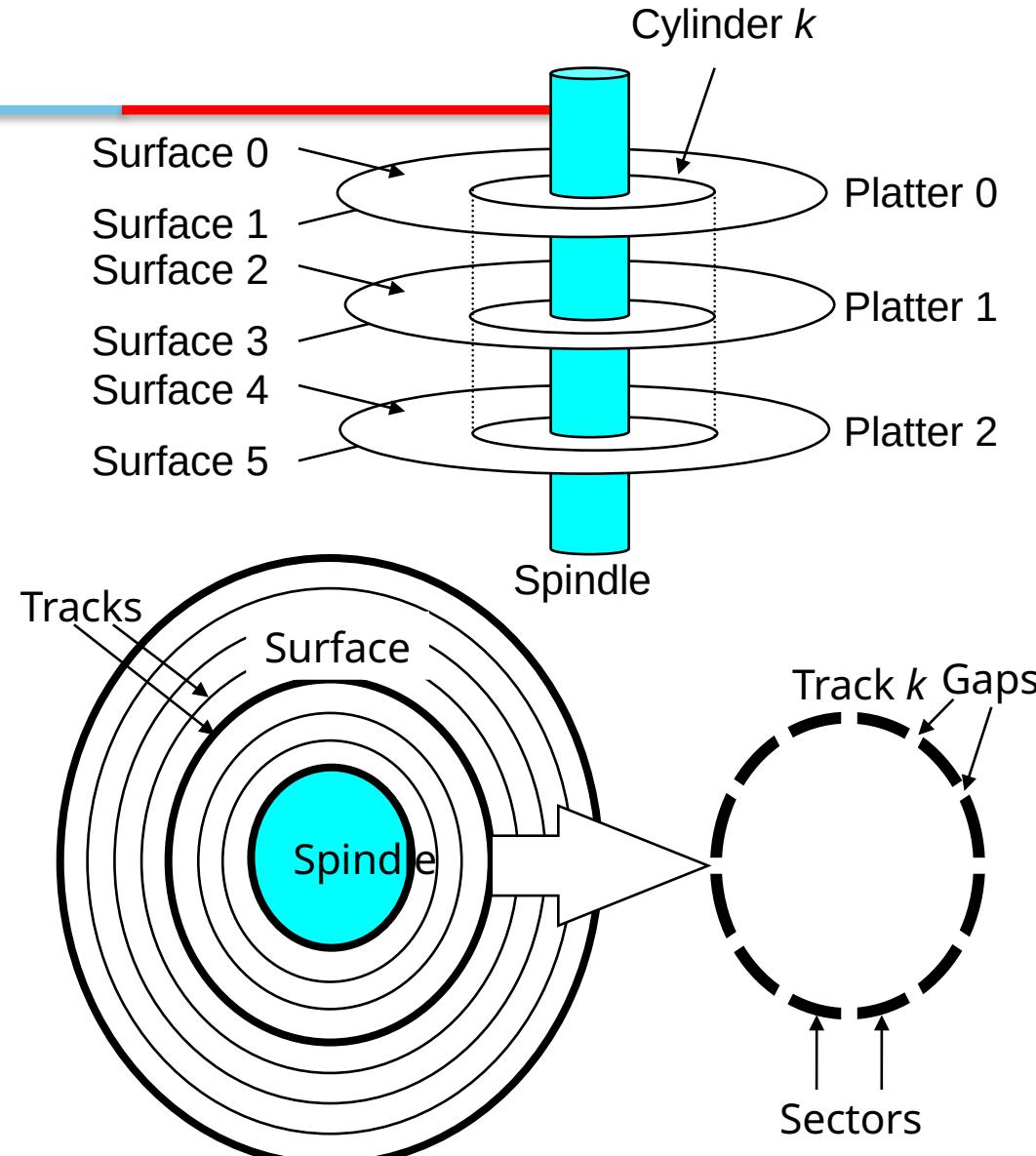
4

INPUT OUTPUT PROCESSORS RESUME OPERATIONS

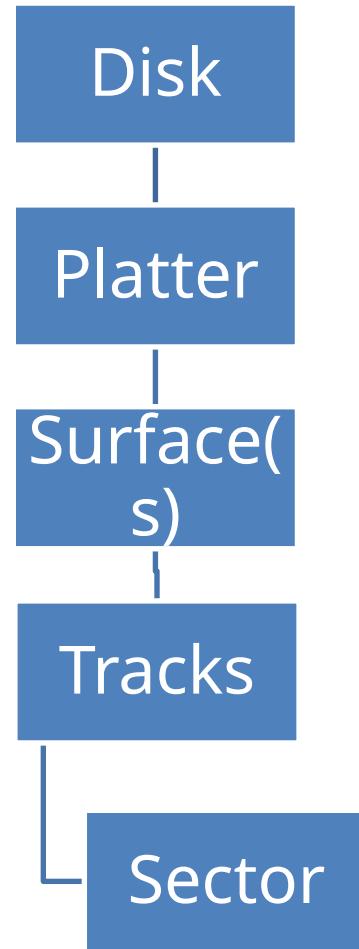
- IOP CAN MANAGE MANY DEVICES
- IOP CAN MANAGE HIGH LEVEL IOP
- MORE SOPHISTICATED THAN DMA

Disk Geometry

- Disks consist of **platters**, each with two **surfaces**.
- Each surface consists of concentric rings called **tracks**
- Aligned tracks form a cylinder
- Each track consists of **sectors** separated by gaps.

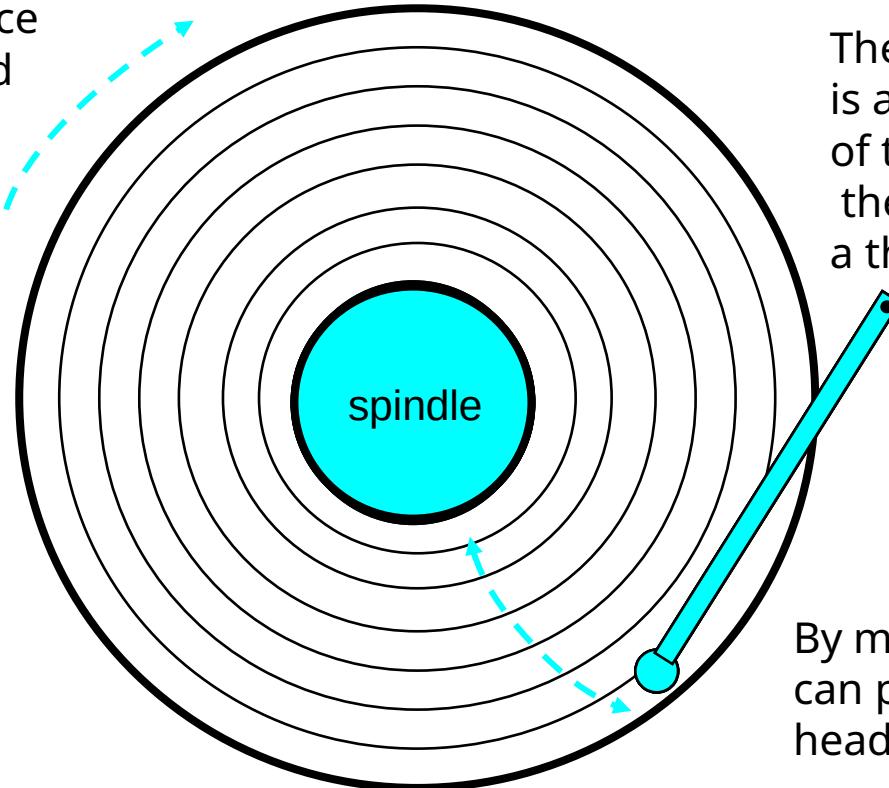


Disk Geometry (contd)



Disk Operation (Single-Platter View)

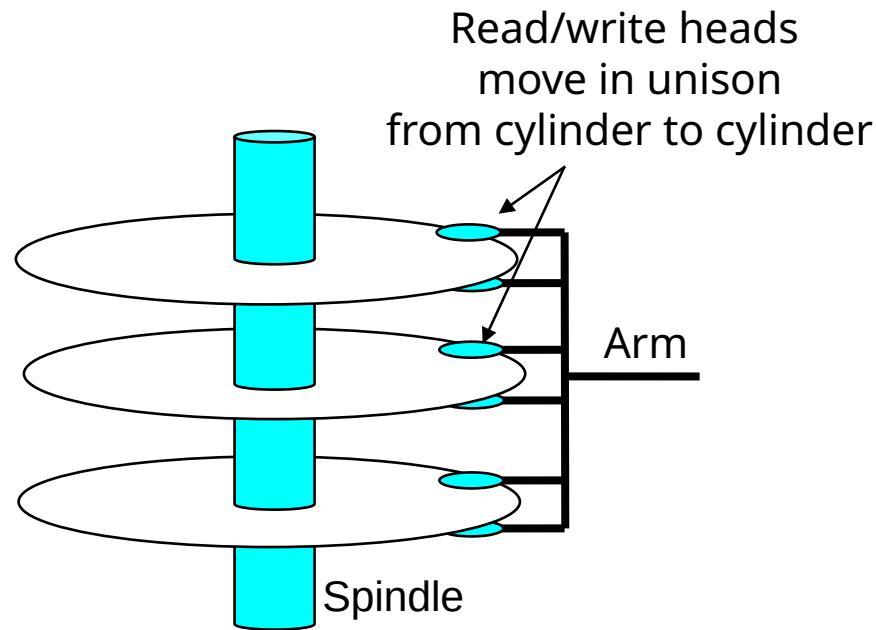
The disk surface spins at a fixed rotational rate



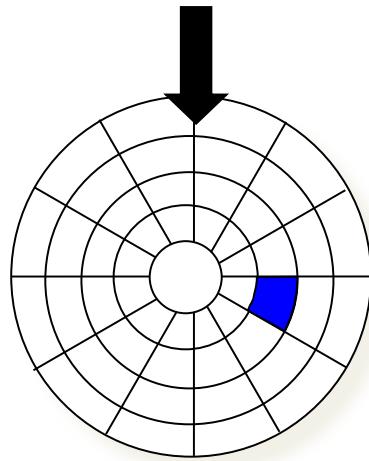
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

Disk Operation (Multi-Platter View)

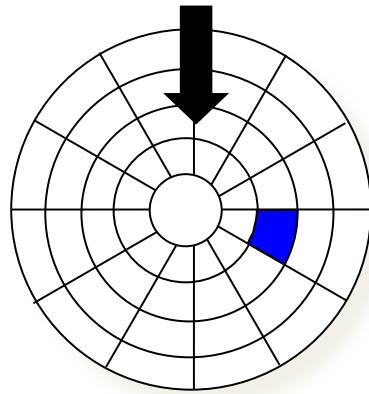


Disk Access



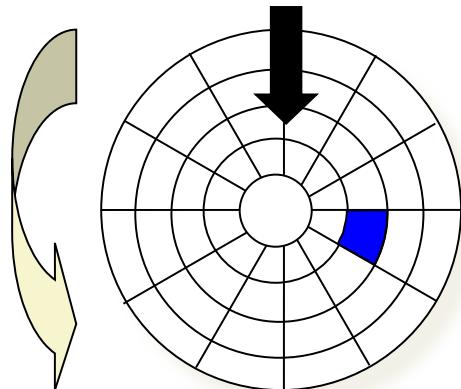
Need to access a sector
colored in blue

Disk Access



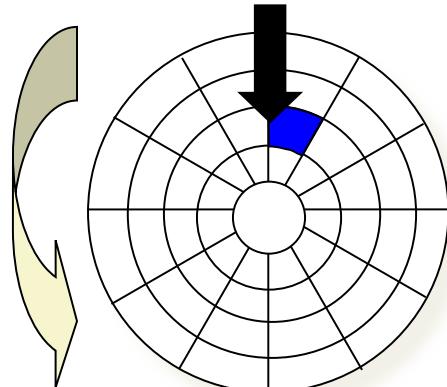
Head in position above a track

Disk Access



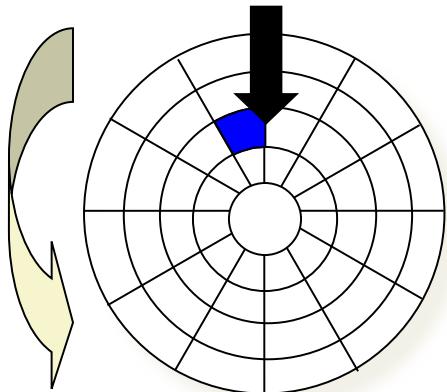
Rotate the platter in counter-clockwise direction

Disk Access – Read



About to read blue sector

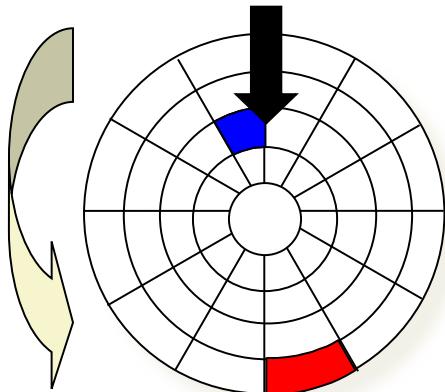
Disk Access – Read



After **BLUE** read

After reading blue sector

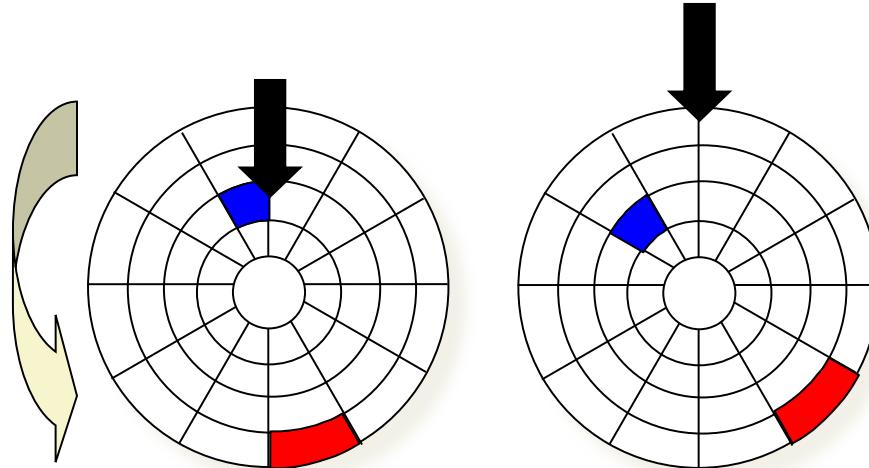
Disk Access – Read



After **BLUE** read

Red request scheduled next

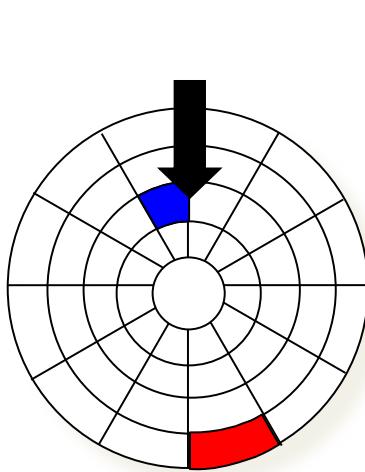
Disk Access – Seek



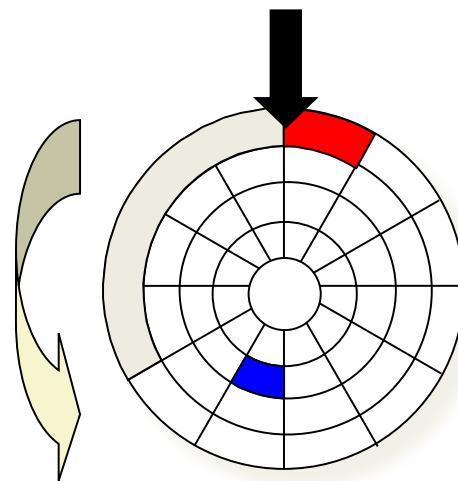
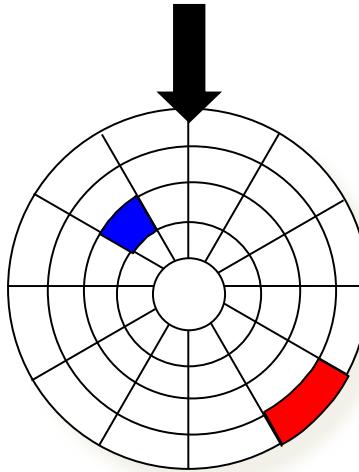
After **BLUE** read Seek for **RED**

Seek to red's track

Disk Access – Rotational Latency



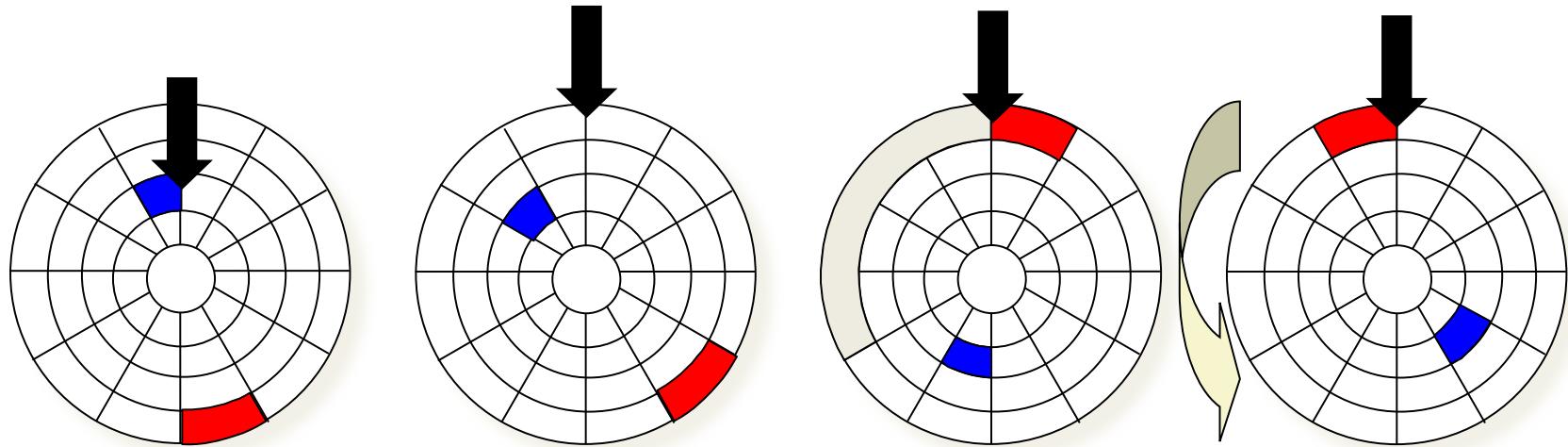
After BLUE read
Seek for RED



Rotational latency

Wait for red sector to rotate around

Disk Access – Read



After **BLUE** read

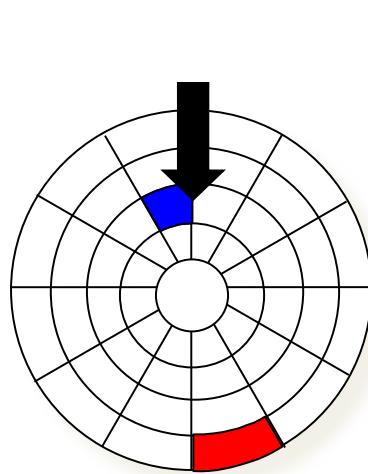
Seek for **RED**

Rotational latency

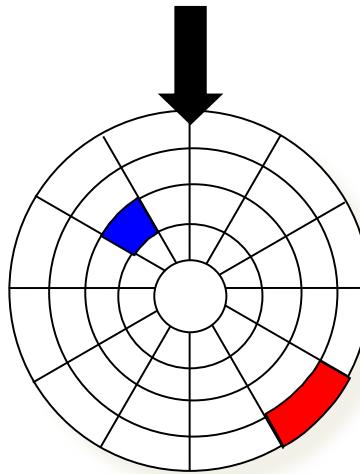
After **RED** read

Complete read of red

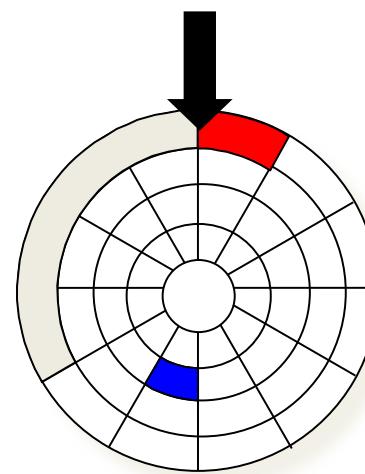
Disk Access - Access Time Components



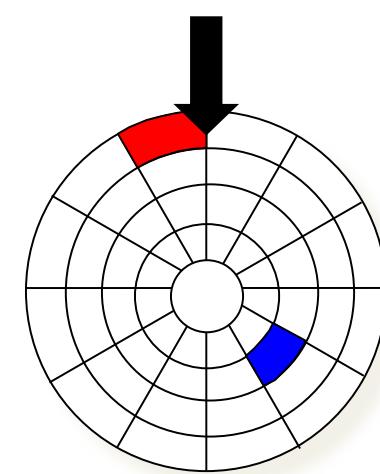
After **BLUE** read



Seek for **RED**



Rotational latency After **RED** read



After **RED** read



Data transfer



Seek



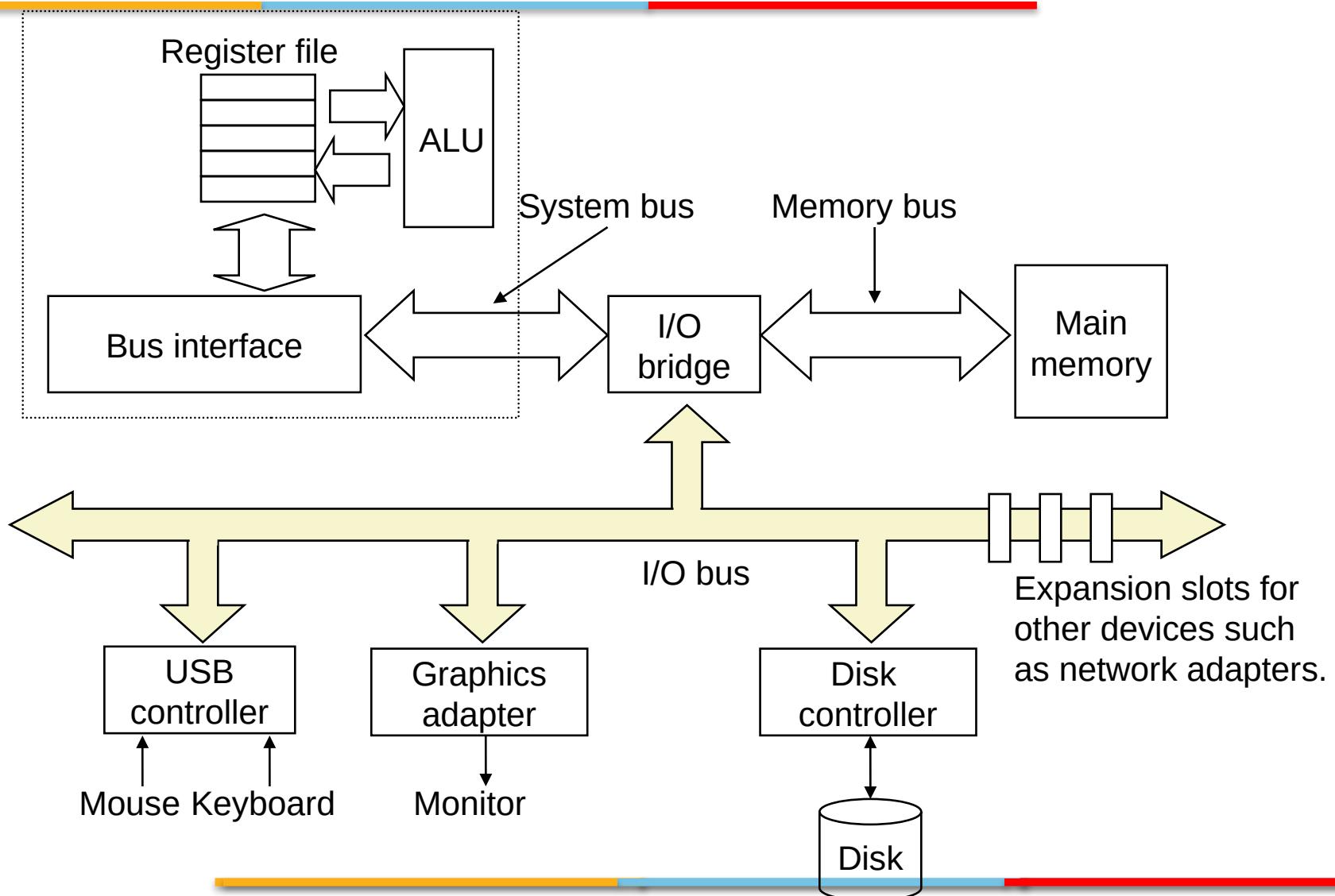
Rotational
latency



Data transfer

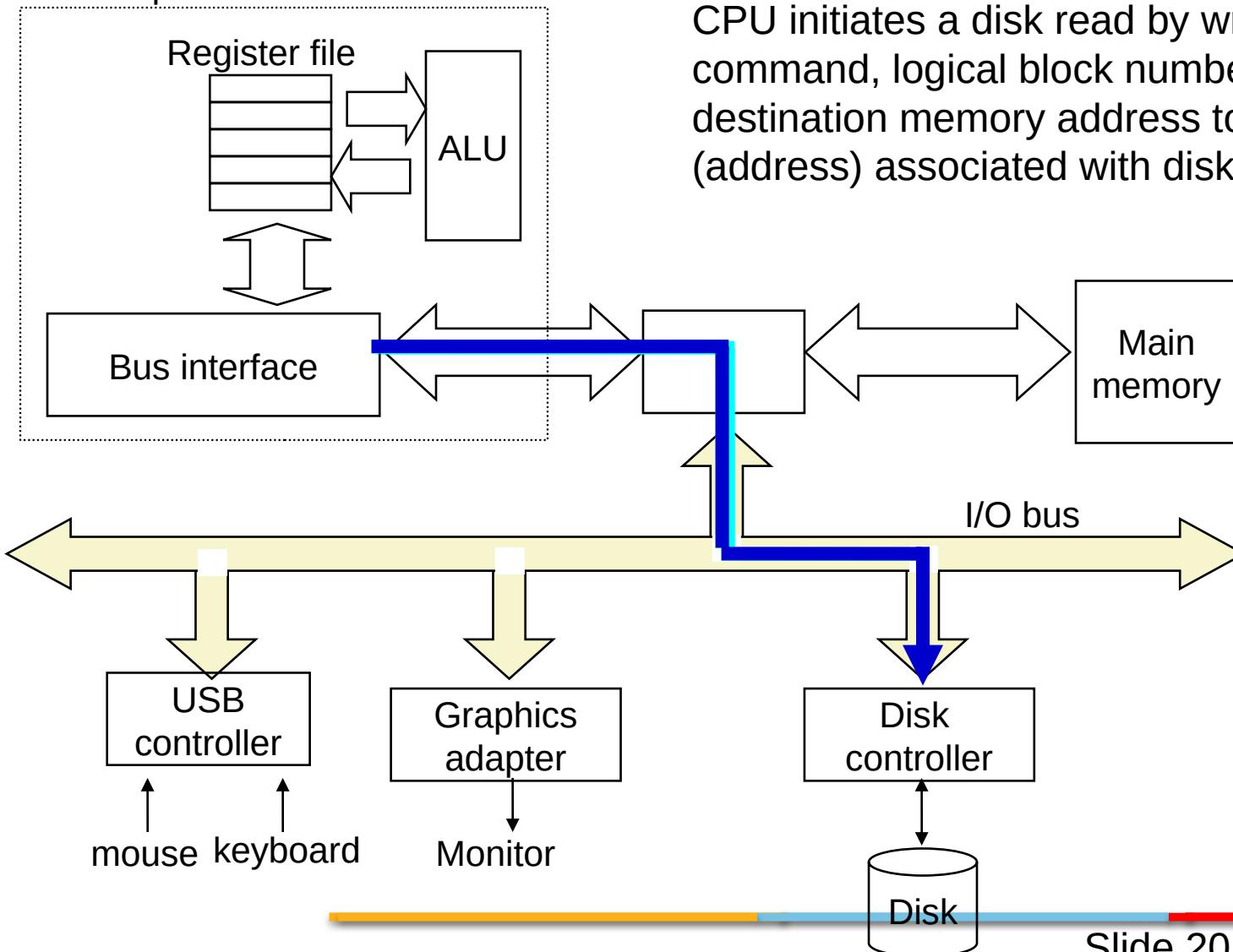
I/O Bus

CPU chip



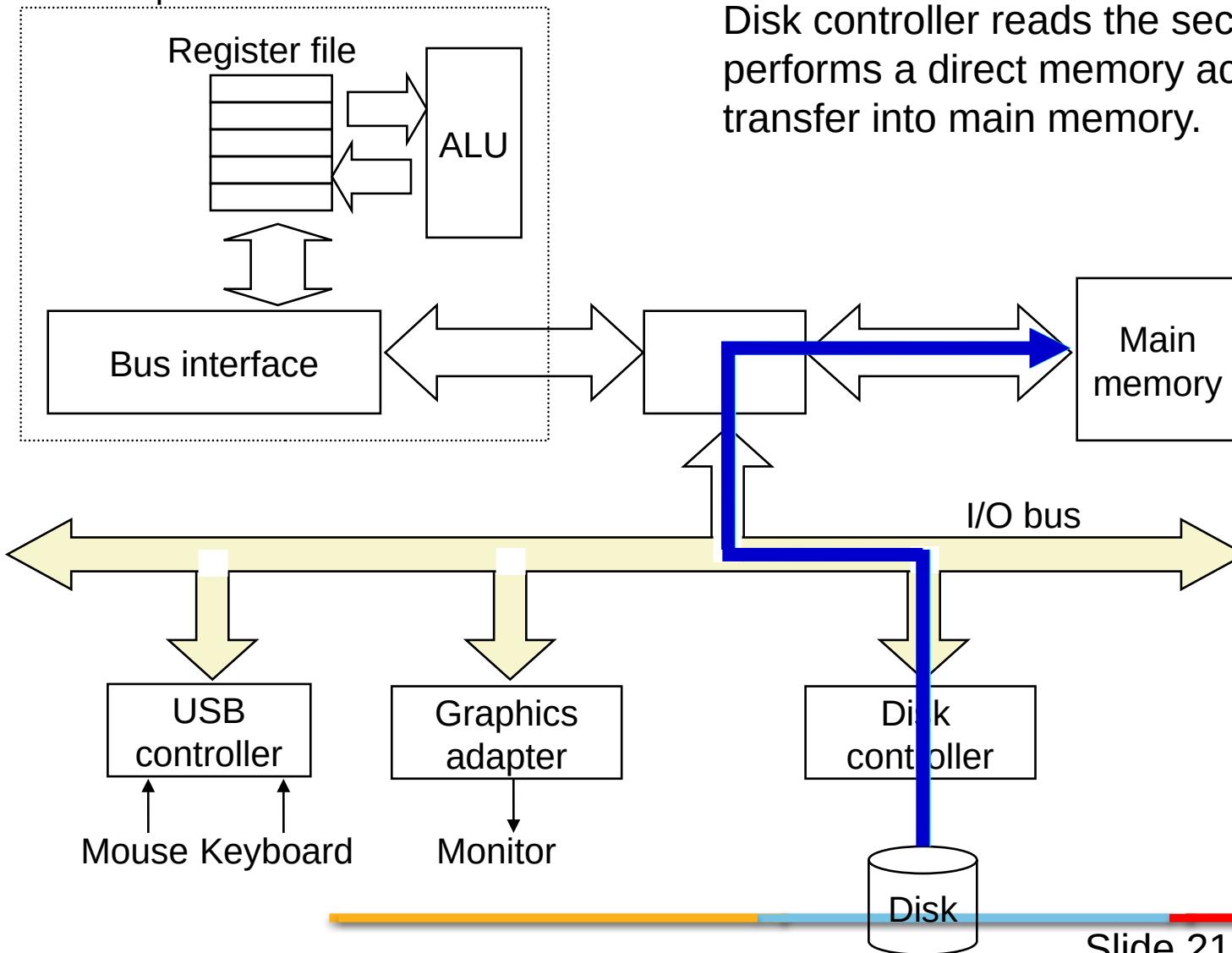
Reading a Disk Sector (1)

CPU chip



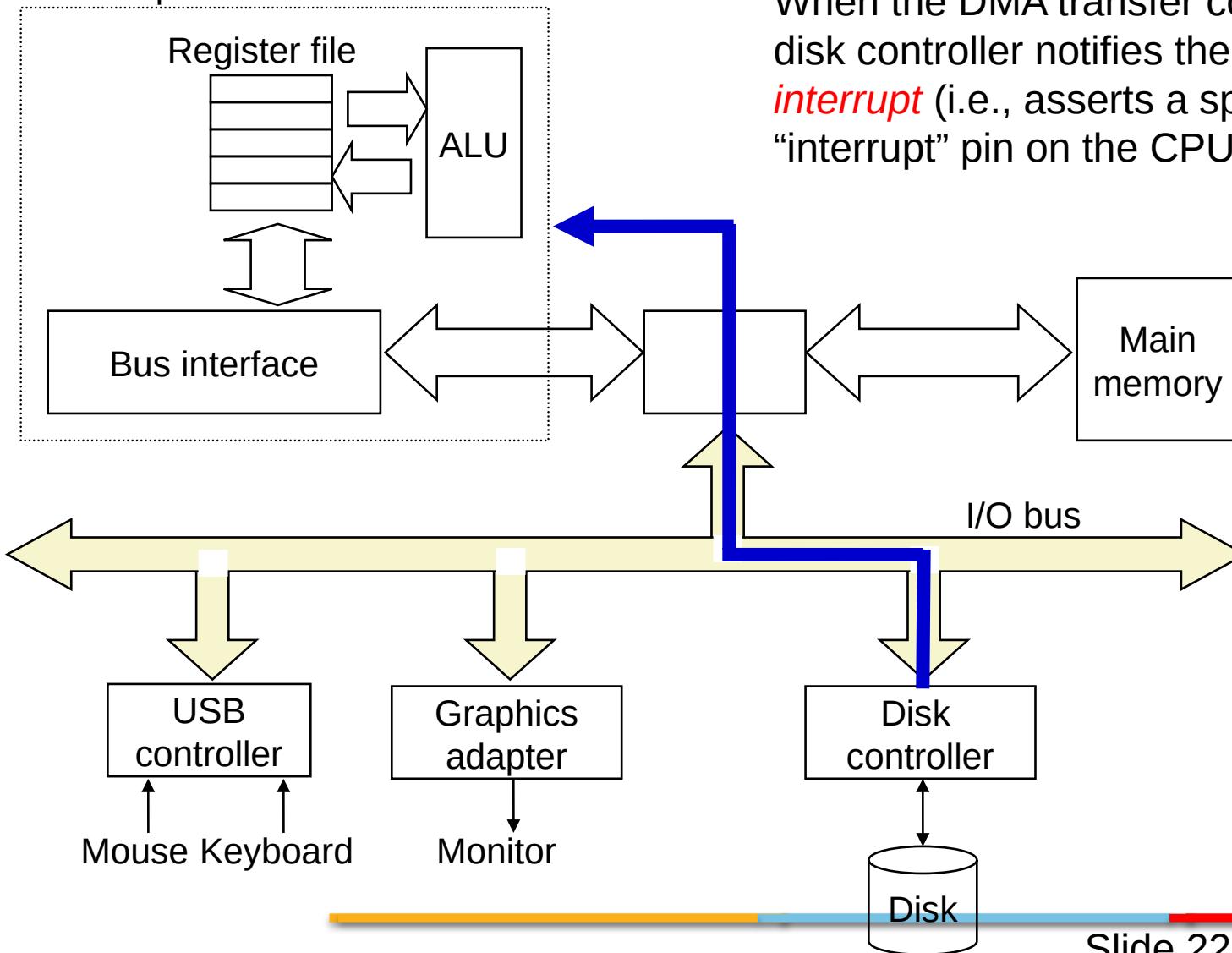
Reading a Disk Sector (2)

CPU chip

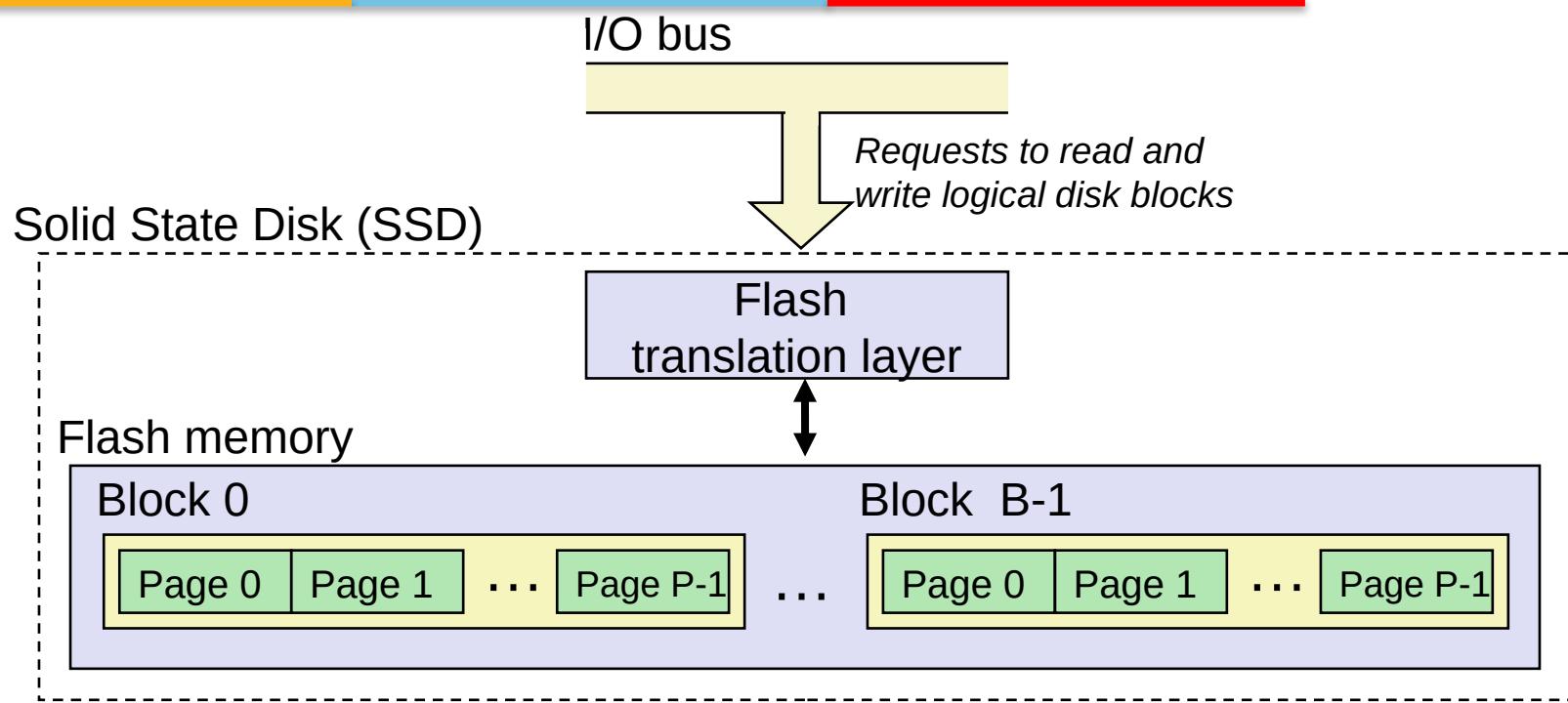


Reading a Disk Sector (3)

CPU chip



Solid State Disks (SSDs)



- Pages: 512KB to 4KB, Blocks: 32 to 128 pages
- Data read/written in units of pages.
- Page can be written only after its block has been erased
- A block wears out after about 100,000 repeated writes.

SSD Performance Characteristics

Sequential read tput*	550 MB/s	Sequential write tput	470 MB/s
Random read tput	365 MB/s	Random write tput	303 MB/s
Avg seq read time	50 us	Avg seq write time	60 us

Tput → Throughput

- Sequential access faster than random access
 - Common theme in the memory hierarchy
- Random writes are somewhat slower
 - Erasing a block takes a long time (~1 ms)
 - Modifying a block page requires all other pages to be copied to new block
 - In earlier SSDs, the read/write gap was much larger.

Source: Intel SSD 730 product specification.

SSD Tradeoffs vs Rotating Disks

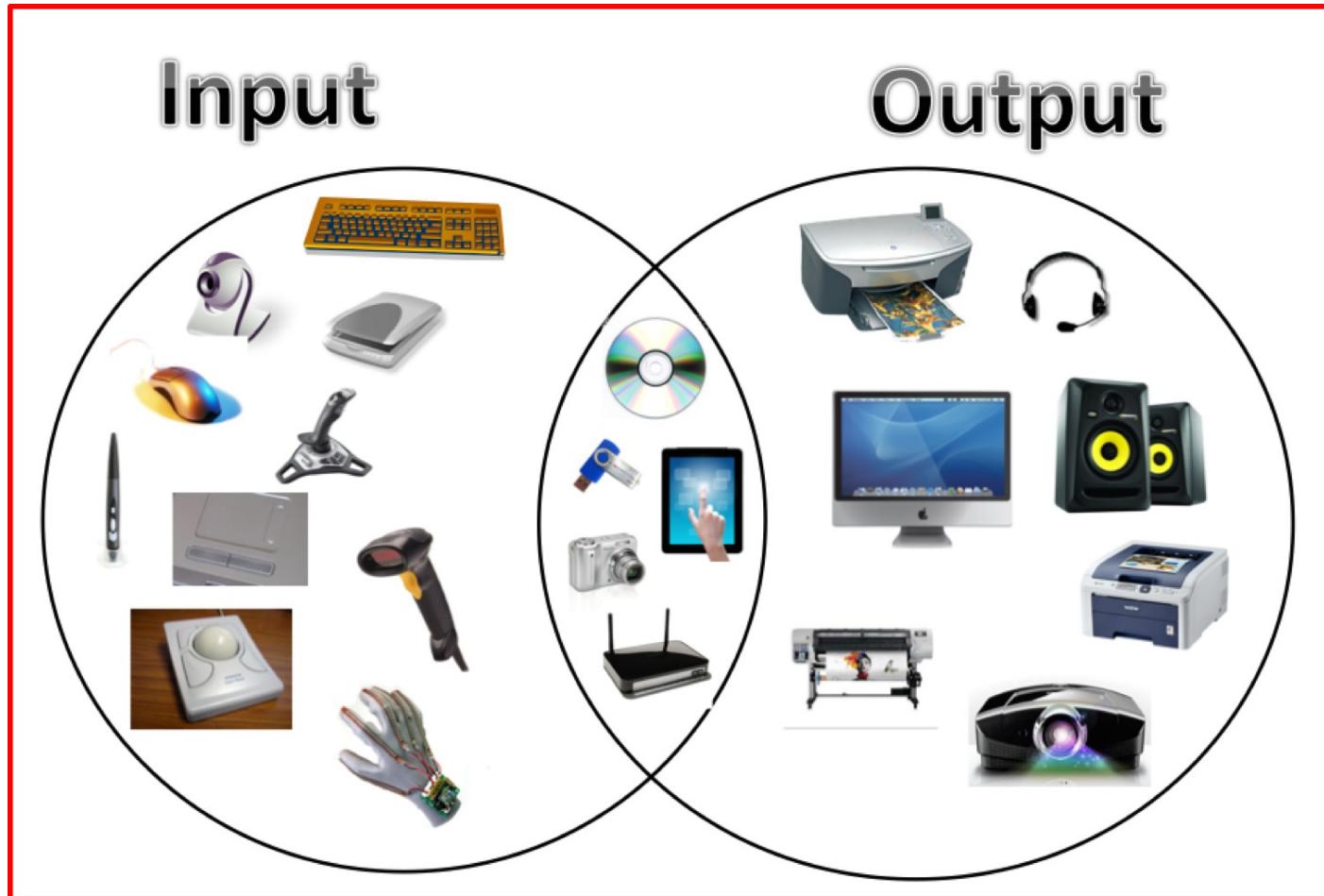
- Advantages
 - No moving parts → faster, less power, more rugged
- Disadvantages
 - Have the potential to wear out
 - Mitigated by “wear leveling logic” in flash translation layer
 - E.g. Intel SSD 730 guarantees 128 petabyte (128×10^{15} bytes) of writes before they wear out
 - In 2015, about 30 times more expensive per byte
- Applications
 - MP3 players, smart phones, laptops
 - Beginning to appear in desktops and servers

I/O Abstraction (Contd...)

Today's
topic

- **I/O System Software**
 - Device Drivers
 - Device Driver Interfaces
 - The Block Device Interfaces
 - The Character Device Interfaces
 - Device Numbers
- **Unification of Files and I/O Devices**
- **Mechanisms for improving I/O performance**
 - Prefetching
 - Caching
 - Buffering

Input and output devices



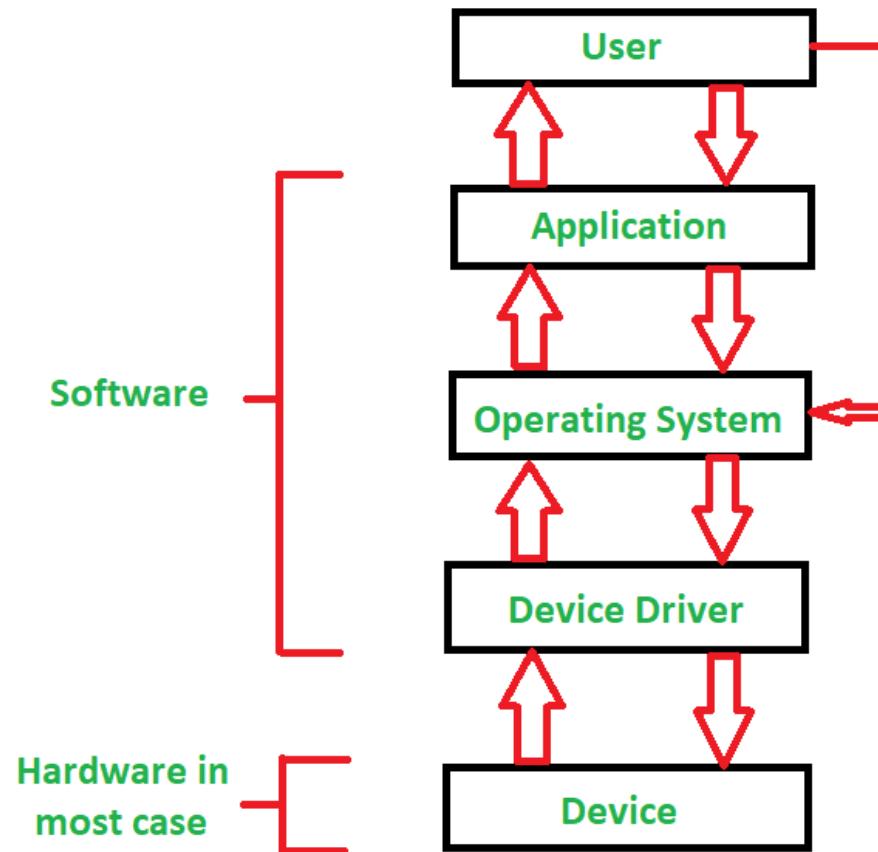
DEVICE DRIVERS

Software that allows your computer to communicate to hardware devices



DEVICE DRIVERS

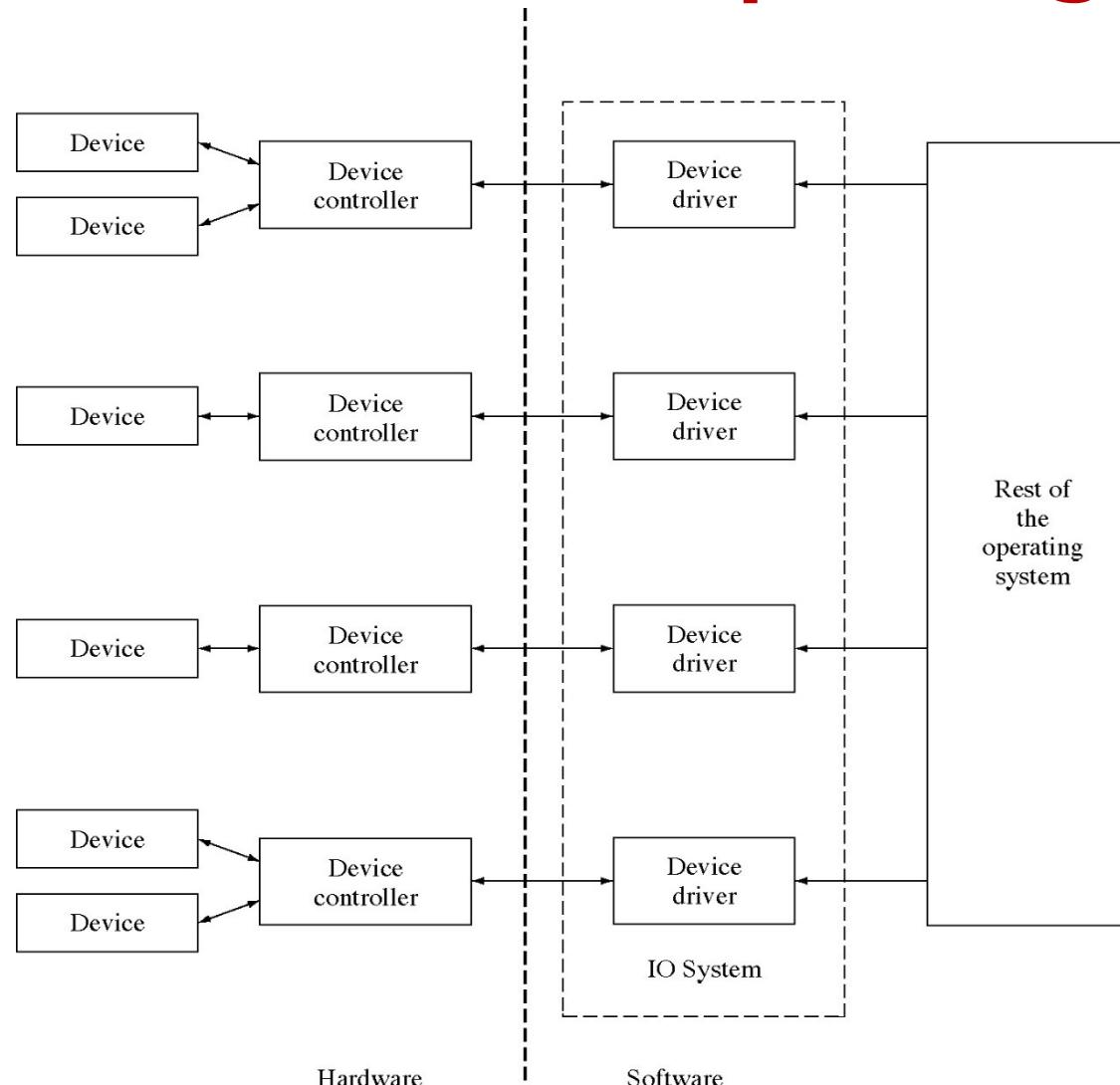
Abstract layer or translation layer



DEVICE DRIVERS

- **Devices** – Complex and hard to use
- It is controlled by **Device Controllers** through Device Controller register (SCSI Devices)
- **Device Driver** – the modules that contains everything about device controller and the device attached to it.
 - An interface module that communicates with both device language and OS language.

Device Drivers in an Operating System

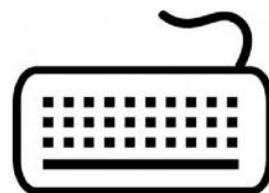


The Two Categories of Device Drivers

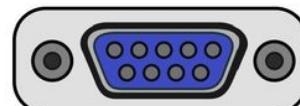


Character vs. Block Devices

Character Devices



keyboard



serial port

Data streams from a device *sequentially*

Block Devices



hard disk drive



SD card
SD CARD
16GB Micro SD HC I



USB drive

Data can be *randomly accessed* in an arbitrary order by software

The Two Categories of Device Drivers

- **Block Device Interface** – the device drivers will all have the same interface; **Block Devices** – A device with **fixed size, addressable blocks**
- **Character Device (Character Device interface)** – a non-block devices send or receive data in streams of bytes with **no addresses** associated with the bytes and with **no fixed block size**.
- Both these interfaces have an **open and close**.

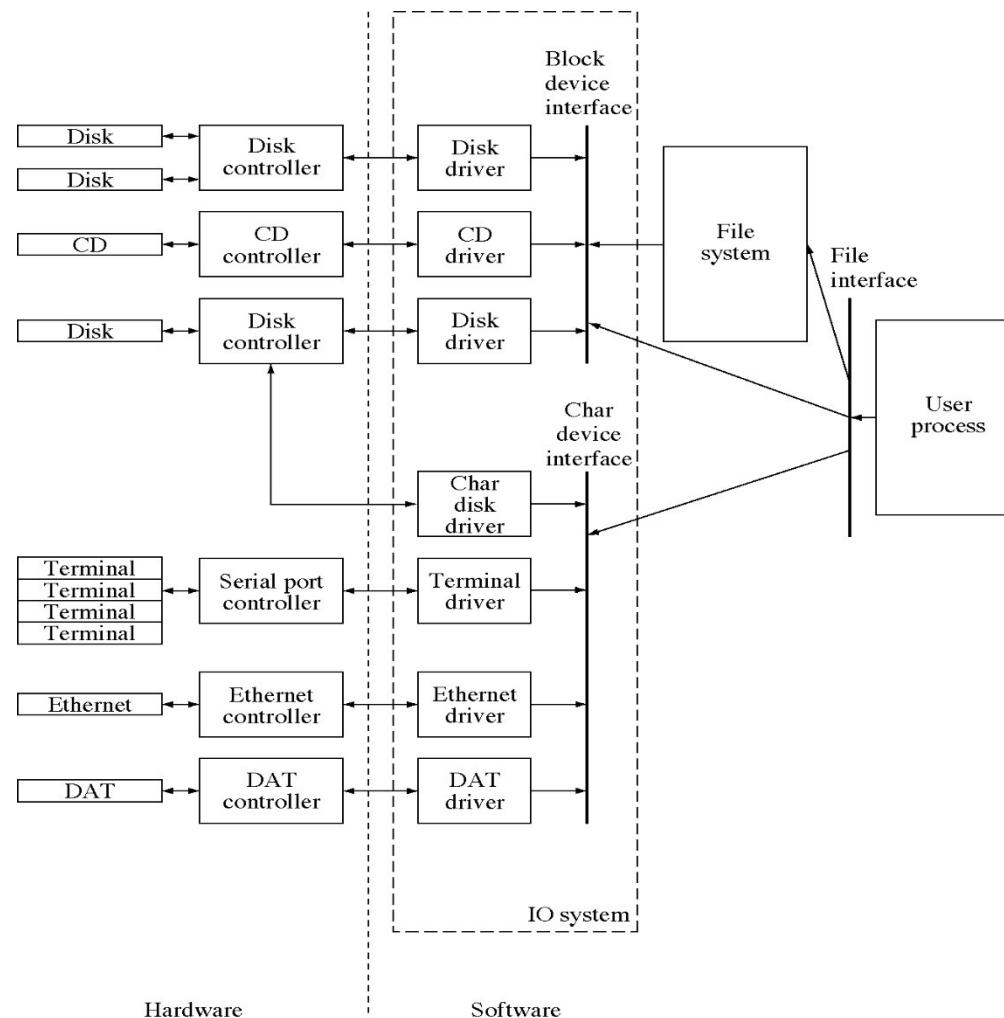
DEVICE NUMBER

- The main use of device numbers is to **name which device to use** (among those controlled by a single controller)
- But they are often used to **convey other information**, for example:
 - device 0 rewinds when done, device 8 does not
 - device 0 uses normal-sized paper, device 8 uses legal-sized paper
 - device 0 writes at high density, device 1 at medium density and device 2 at low density

UNIFICATION OF FILES AND I/O DEVICES

- The device driver interface is nearly the same as the file interface
- The file interface is a set of operations – **open, read, write, seek and close**
- The I/O system has code to translate from the file interface to the device driver interface, so file commands can be used on devices

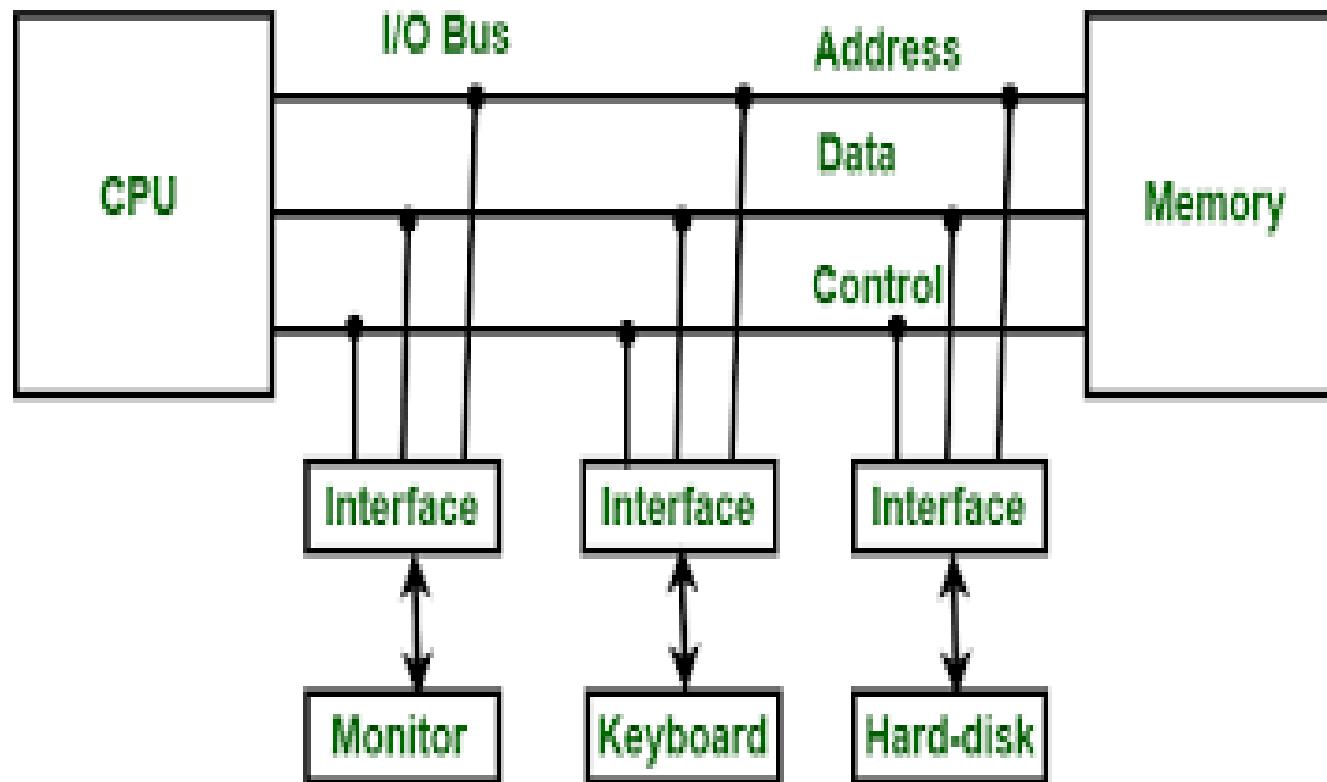
The File, Character, and Block Device Interfaces



The File, Character, and Block Device Interfaces

- There are block devices that are connected to device controllers and managed by device drivers.
- All block device drivers have same block device interface.
- All character device drivers have same character device interface
- The file system uses the block device interface and exports the file interface for files.
- The I/O system uses block device interface and character device interface and exports the file interface for devices.
- The user only sees the file interface for both files and devices.

The File, Character, and Block Device Interfaces



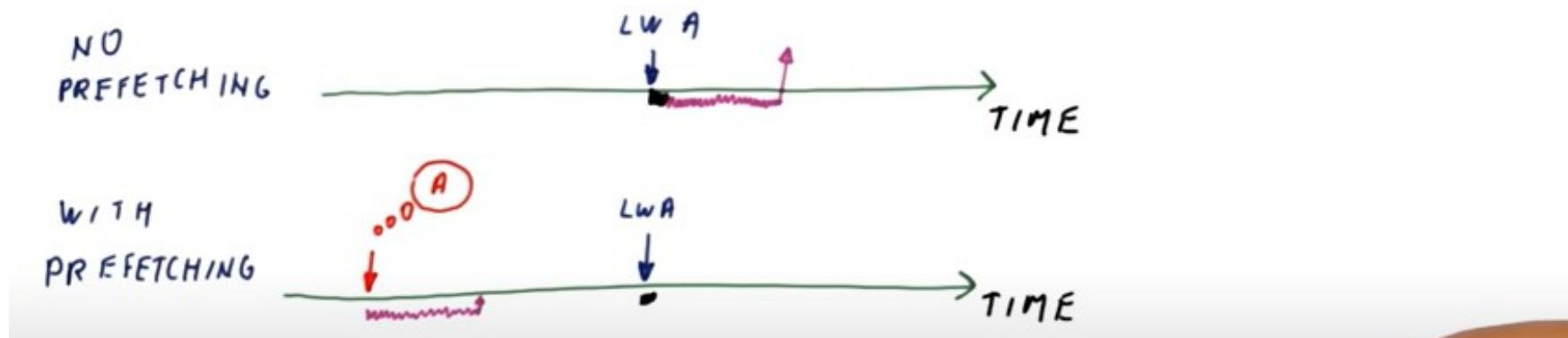
Mechanisms for Improving I/O Performance

- Prefetching

- Prefetching is a technique for speeding up fetch operations by beginning a fetch operation whose result is expected to be needed soon.
- Prefetching allows a browser to silently fetch the necessary resources needed to display content that a user might access in the near future.
- The browser stores these resources in its cache enabling it to deliver the requested data faster.

Mechanisms for Improving I/O Performance

- Prefetching

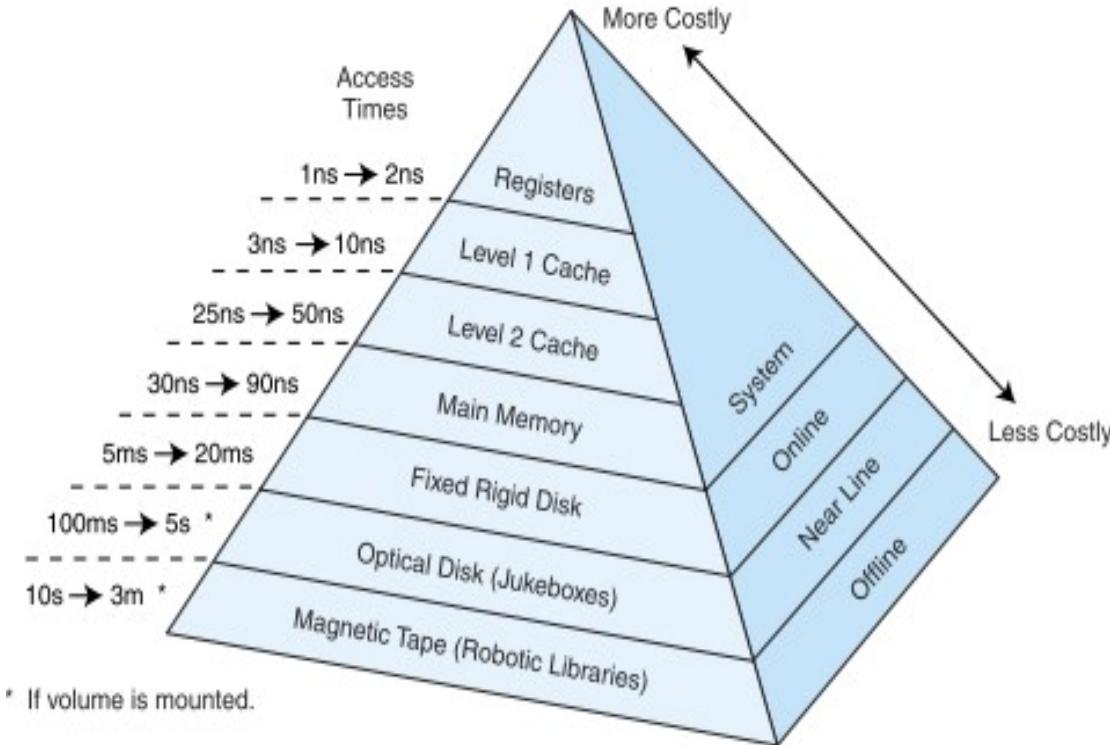


1. Stream buffer
2. Stride pre-fetcher
3. Correlating pre-fetcher

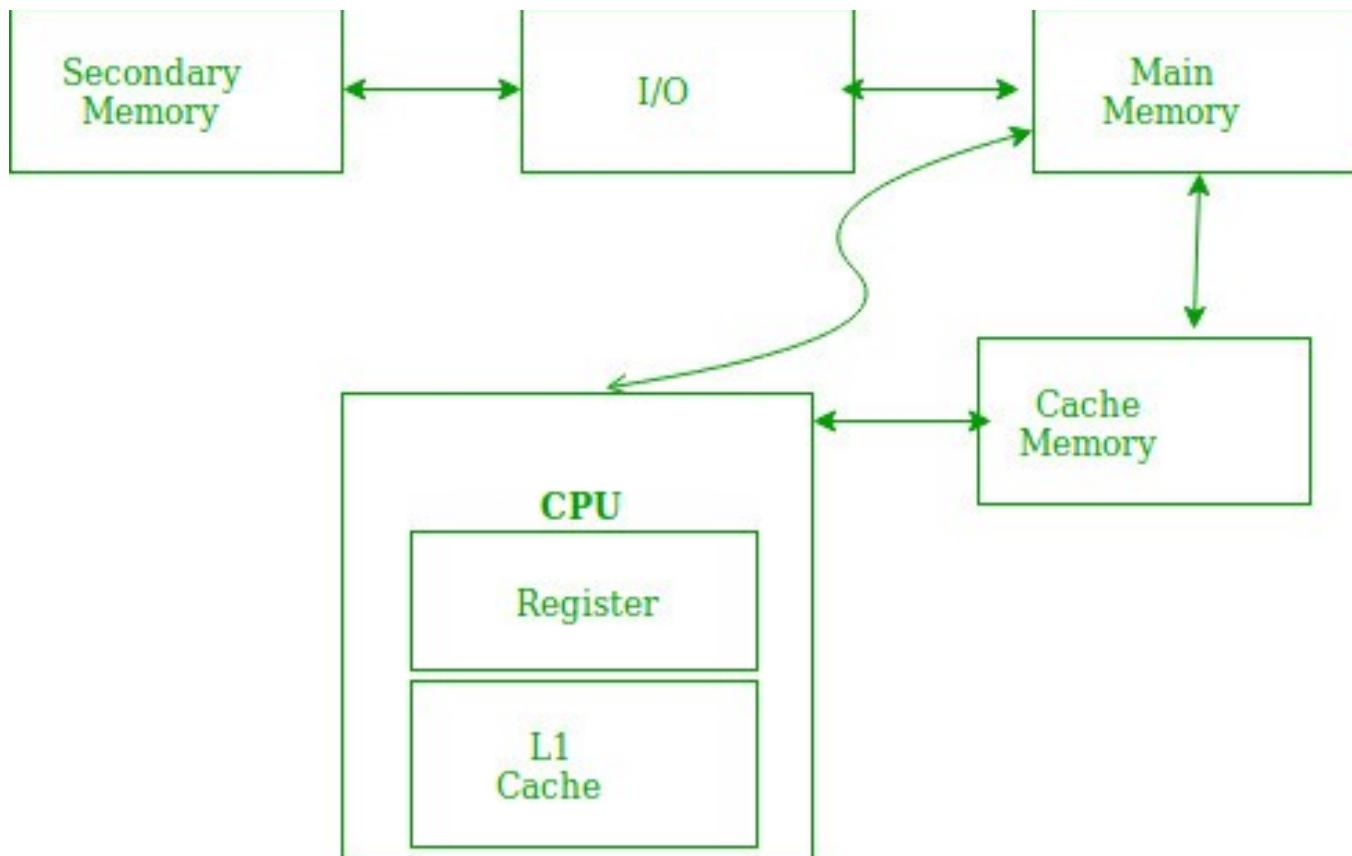
Caching

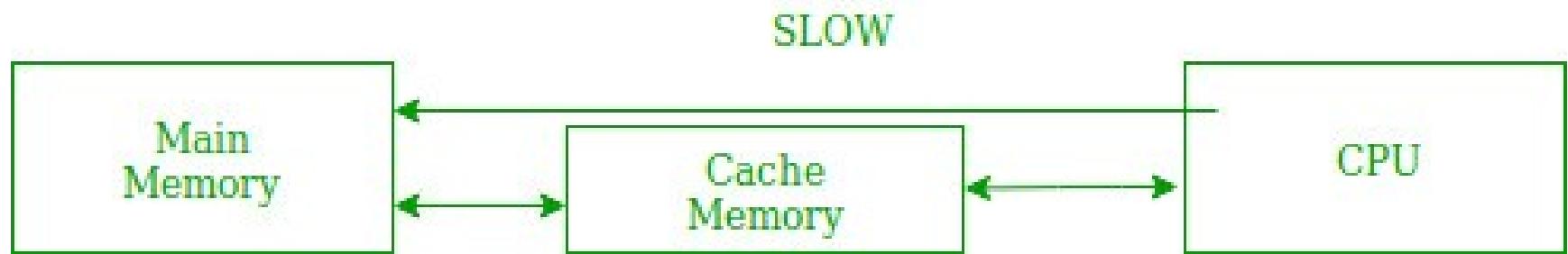
- **Caching** - storing data in a separate disk (very fast speed disk).
- The data which is to be used many times results in wastage of time if it is in hard disk but storing the data in cache **reduces this time wastage**.
- Cache is in processor and can be also **implemented with RAM and disk**.
- **Read and write is same** as the normal storage
- **Example** – Cache is used in system to **speed up the access of data** frequently used.

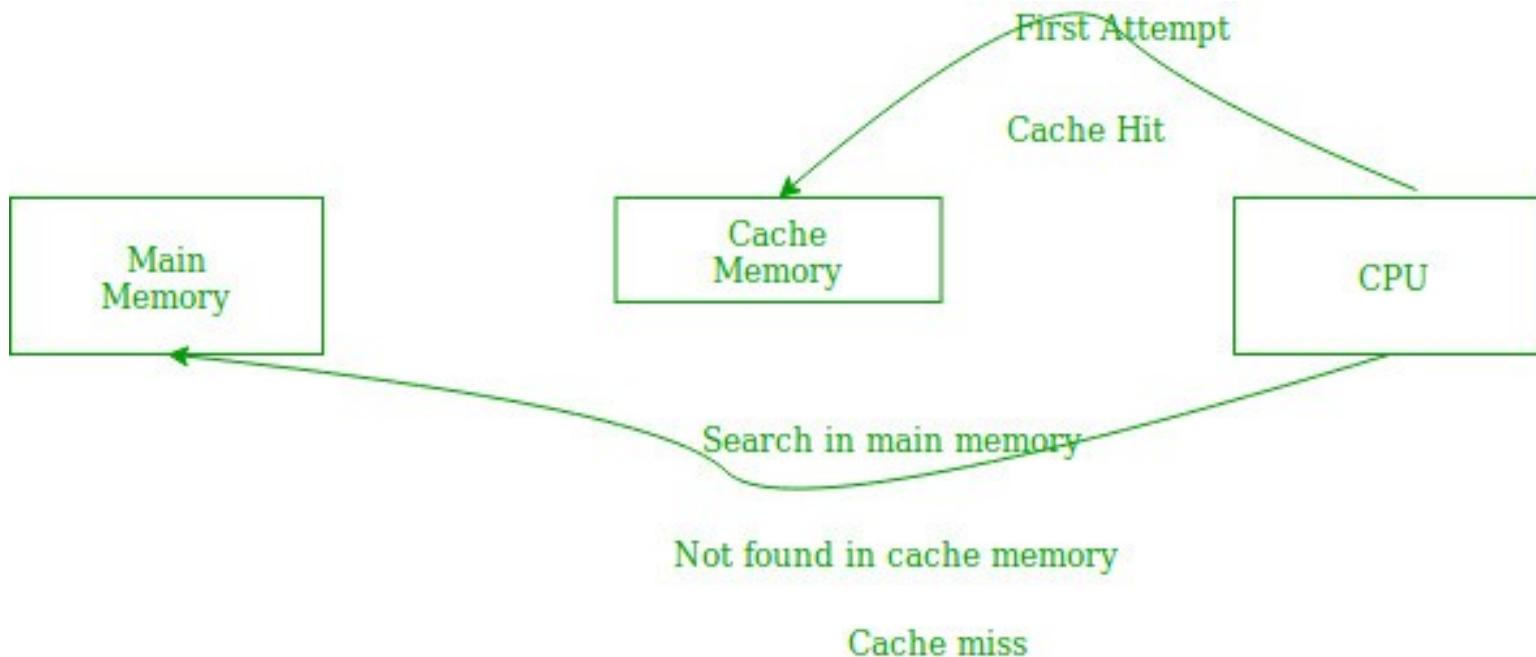
Computer Memory Hierarchy



Source: Null, Linda and Lobur, Julia (2003). *Computer Organization and Architecture* (p. 236). Sudbury, MA: Jones and Bartlett Publishers







Principle of Locality

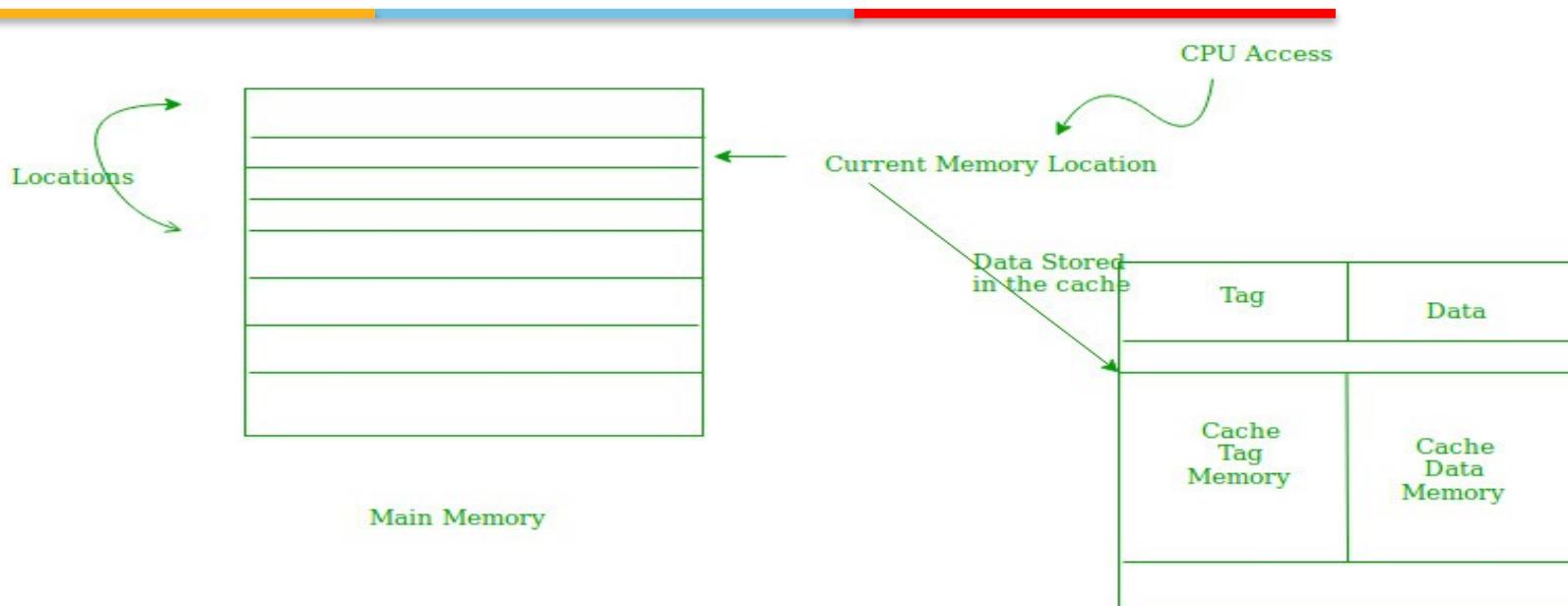
locality of reference, also known as the **principle of locality**, is the tendency of a processor to access the same set of memory locations repetitively over a short period of time.

- .

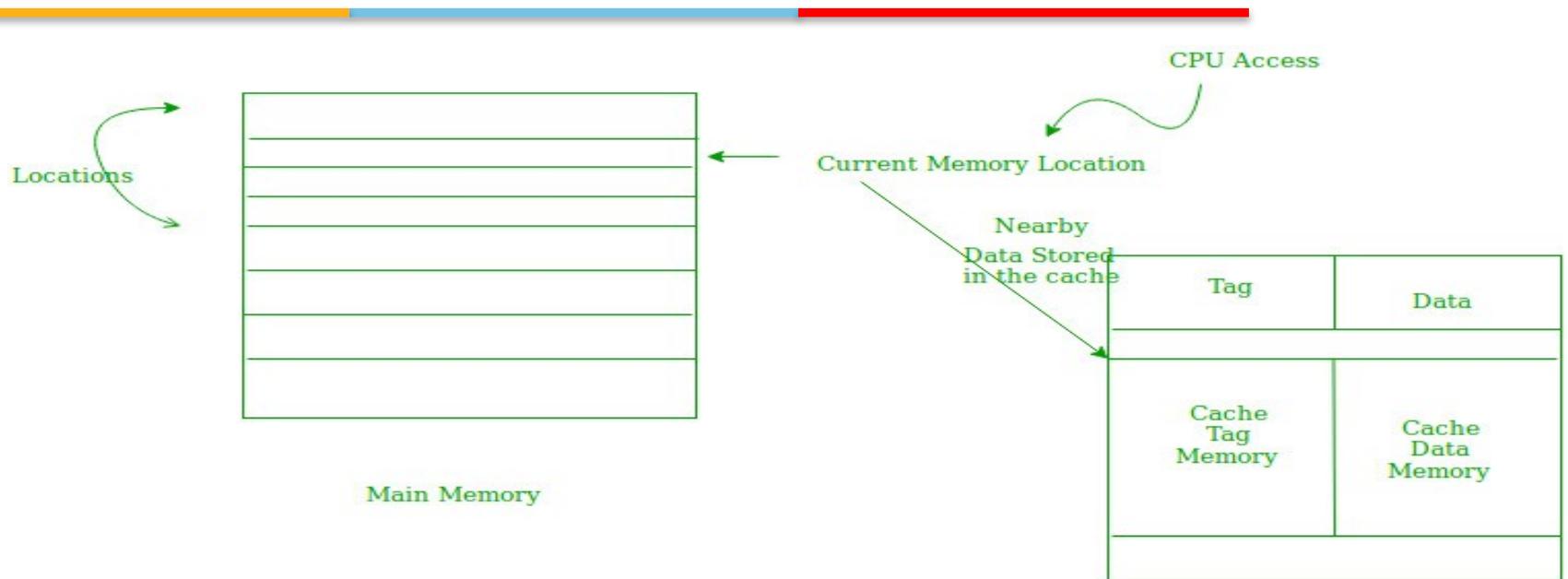
In case of loops in program control processing unit repeatedly refers to the set of instructions that constitute the loop.

In case of subroutine calls, everytime the set of instructions are fetched from memory.

References to data items also get localized that means same data item is referenced again and again



- **Temporal locality** means current data or instruction that is being fetched may be needed soon. So we should store that data or instruction in the cache memory so that we can avoid again searching in main memory for the same data



Spatial locality means instruction or data near to the current memory location that is being fetched, may be needed soon in the near future. This is slightly different from the temporal locality. Here we are talking about nearly located memory locations while in temporal locality we were talking about the actual memory location that was being fetched.

Cache Performance

1. The performance of the cache is measured in terms of hit ratio.
2. When CPU refers to memory and find the data or instruction within the Cache Memory, it is known as cache hit.
3. If the desired data or instruction is not found in the cache memory and CPU refers to the main memory to find that data or instruction, it is known as a cache miss.

$$\begin{aligned}\text{Hit} + \text{Miss} &= \text{Total CPU} \\ &\text{Reference} \\ \text{Hit Ratio}(h) &= \text{Hit} / \\ &(\text{Hit} + \text{Miss})\end{aligned}$$

- Average access time of any memory system consists of two levels: Cache and Main Memory. If T_c is time to access cache memory and T_m is the time to access main memory then we can write:

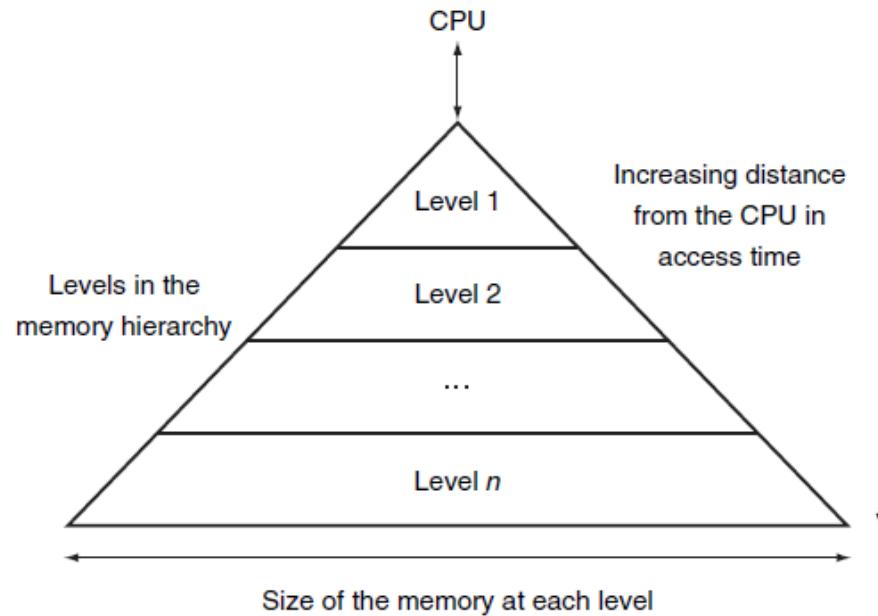
$$T_{avg} = \text{Average time to access memory}$$
$$T_{avg} = h * T_c + (1-h)*(T_m + T_c)$$

Memory Hierarchy..

- A memory hierarchy consists of multiple levels of memory with different speeds and sizes.
- The faster memories are more expensive per bit than the slower memories and thus are smaller.
- The goal is to present the user with as much memory as is available in the cheapest technology, while providing access at the speed offered by the fastest memory
- A memory hierarchy can consist of multiple levels, but data is copied between only two adjacent levels at a time

- The upper level—the one closer to the processor—is smaller and faster than the lower level, since the upper level uses technology that is more expensive
- If the data requested by the processor appears in some block in the upper level, this is called a **hit**
- If the data is not found in the upper level, the request is called a **miss**.
- **block (or line)** The minimum unit of information that can be either present or not present in a cache.
- **hit rate** The fraction of memory accesses found in a level of the memory hierarchy.(Also called hit ratio)
- **miss rate** The fraction of memory accesses not found in a level of the memory hierarchy (1-hit rate)
- **miss penalty** The time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, and then pass the block to the requestor.

Memory Hierarchy..

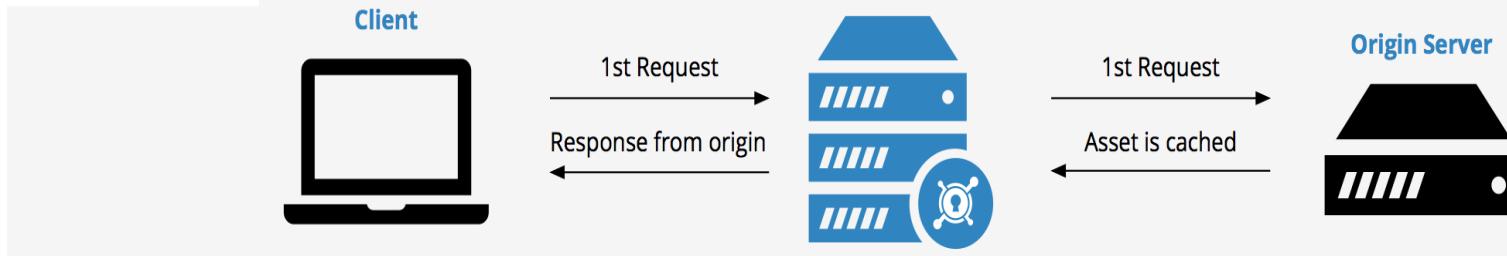


Structure of Memory Hierarchy: As the distance from processor increases, so does the size

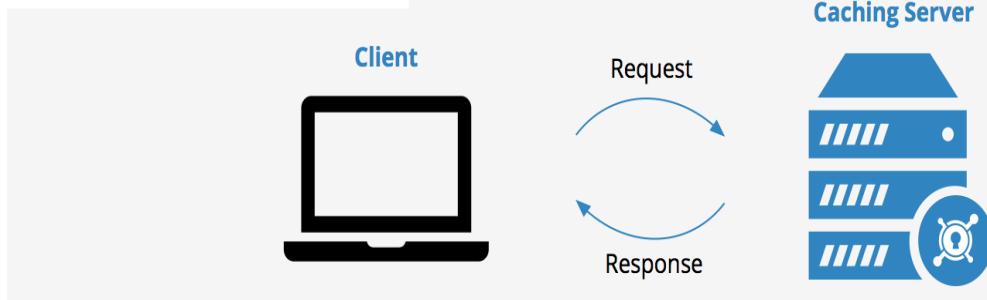
Cache Memory

A cache is a smaller, faster memory, located closer to a processor core, which stores copies of the data from frequently used main memory locations.

1st Request



Subsequent Requests



Cache Definition

Buffering

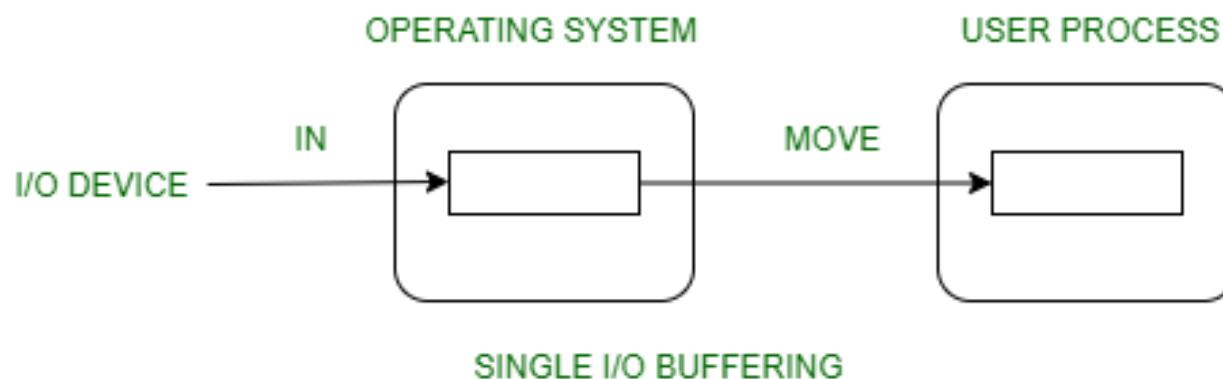
- A buffer is a memory area that stores data being transferred between two devices or between a device and an application
- Buffering is done to deal effectively with a speed mismatch between the producer and consumer of the data stream.
- This process of data transfer is not instantaneous, therefore it needs another buffer in order to store additional incoming data.
- Buffering also provides variations for devices that have different data transfer sizes.

Uses of I/O Buffering

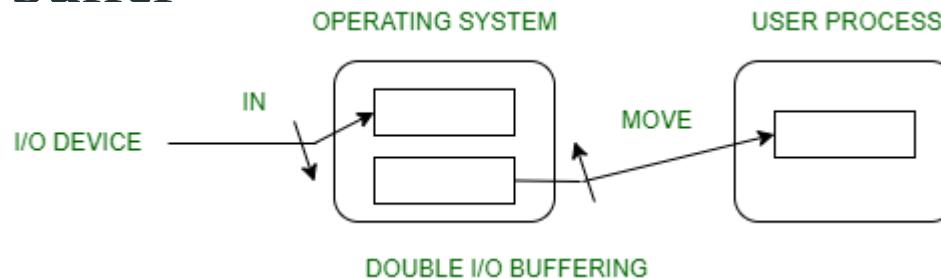
- A buffer is produced in main memory to **heap up the bytes received** from modem.
- Provides variations for devices that have different data transfer sizes.
- The use of two buffers disintegrates the producer and the consumer of the data, thus **minimizes the time requirements** between them.

Types of I/O Buffering Techniques

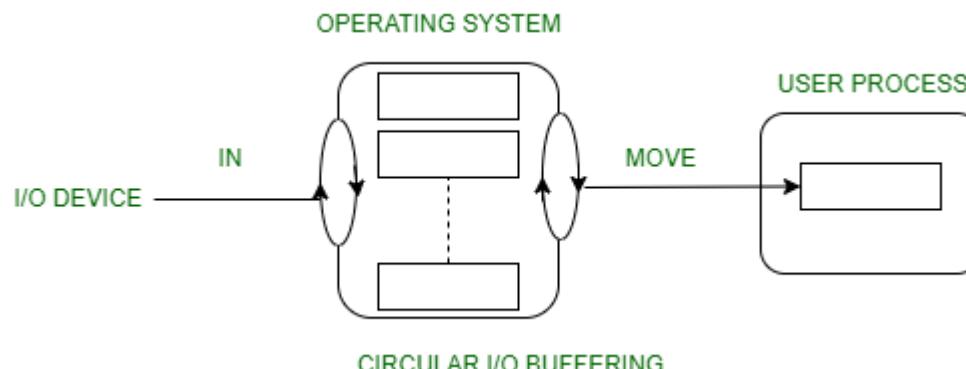
- **Single buffer** - A buffer is provided by the operating system to the system portion of the main memory.



- **Double buffer**



- **Circular buffer** - When more than two buffers are used, the collection of buffers is itself referred to as a circular buffer. (producer-consumer)





BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 11

Course design by - Dr. Lucy J. Gudino &

Dr Chandrasekhar

Faculty - Dr Thangakumar J

WILP & Department of CS & IS



BITS Pilani
Pilani Campus

Operating Systems - File as a Logical Abstraction of Storage



Last Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• I/O System Software<ul style="list-style-type: none">◦ Device Drivers◦ Device Driver Interfaces<ul style="list-style-type: none">▪ The Block Device Interfaces▪ The Character Device Interfaces◦ Device Numbers• Unification of Files and I/O Devices• Mechanisms for improving I/O performance<ul style="list-style-type: none">◦ Prefetching◦ Caching◦ Buffering	<p>T1 (15.1, 15.4)</p> <p>T1 (15.5)</p>

File as a Logical Abstraction of Storage



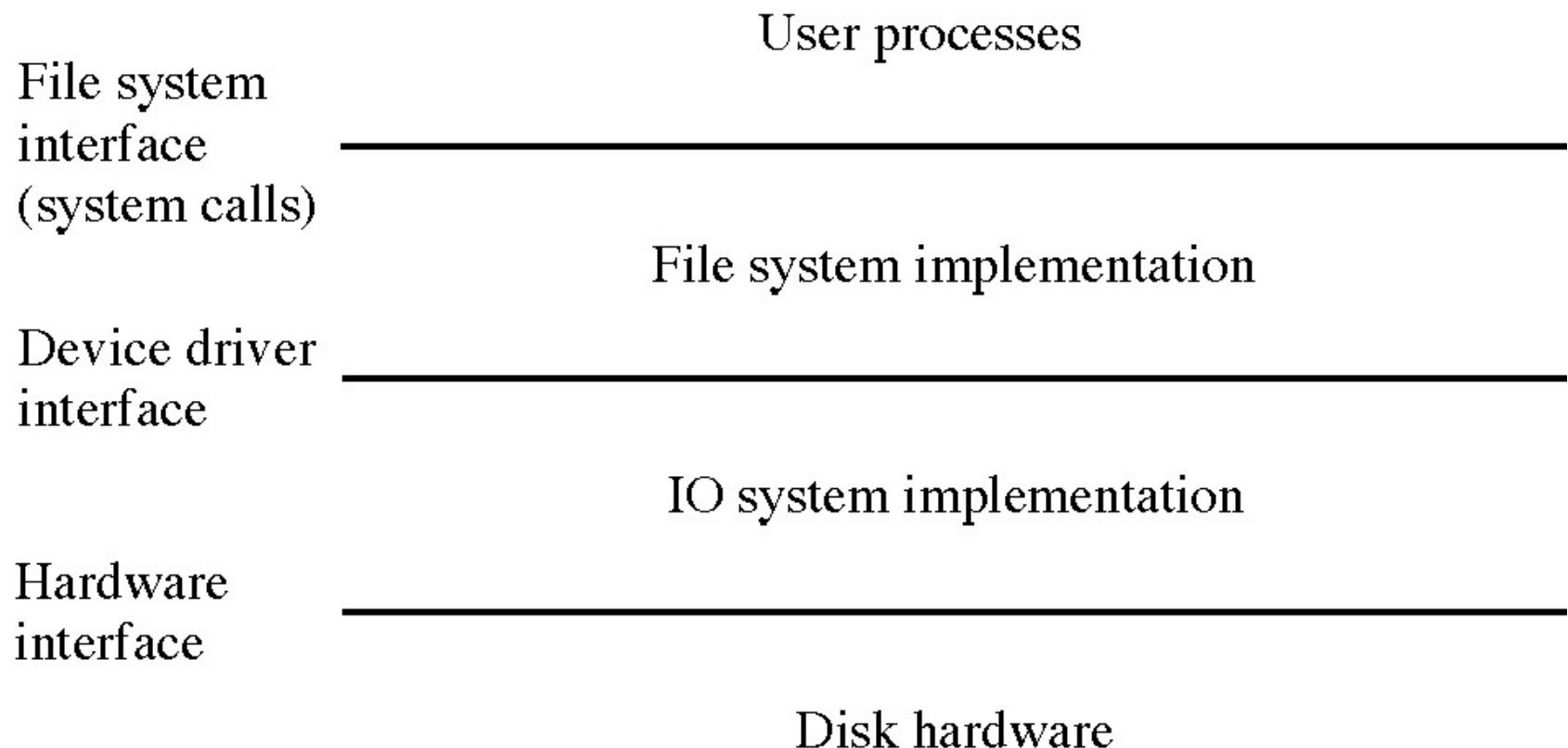
Today's topic

- Levels in the file system
- File storage on hard disks, CD ROMs and Flash ROMs
- File Vs Databases

FILES

- A **File** – a sequence of bytes of arbitrary length (0 to large upper limit)
- It has **name** (a character string) which shows the fits into a hierarchical structure.
- Implemented by the OS to provide **persistent storage**.

LEVELS IN THE FILE SYSTEM



- Figure shows the levels of file system abstraction.
- Logical specification of the system call interface to the file system objects and operations.
- Implementation of the file system interface on the I/O system interface
- Implementation of the I/O system interface on the disk hardware

Views of a File at Each Level

User process

The user process sees a file that is an array of bytes

File "mydata"

Byte 872

File system

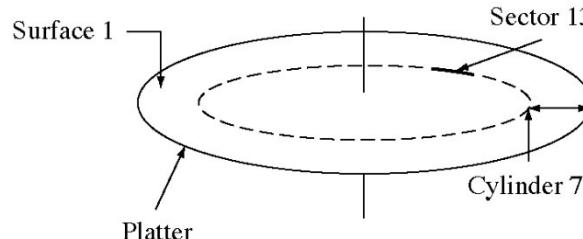
The file system sees a logical disk that is an array of sectors

Disk/dev/fu4a

Sector 1214

Device driver

The disk driver sees a disk of platters, cylinders, and sectors



Disk controller

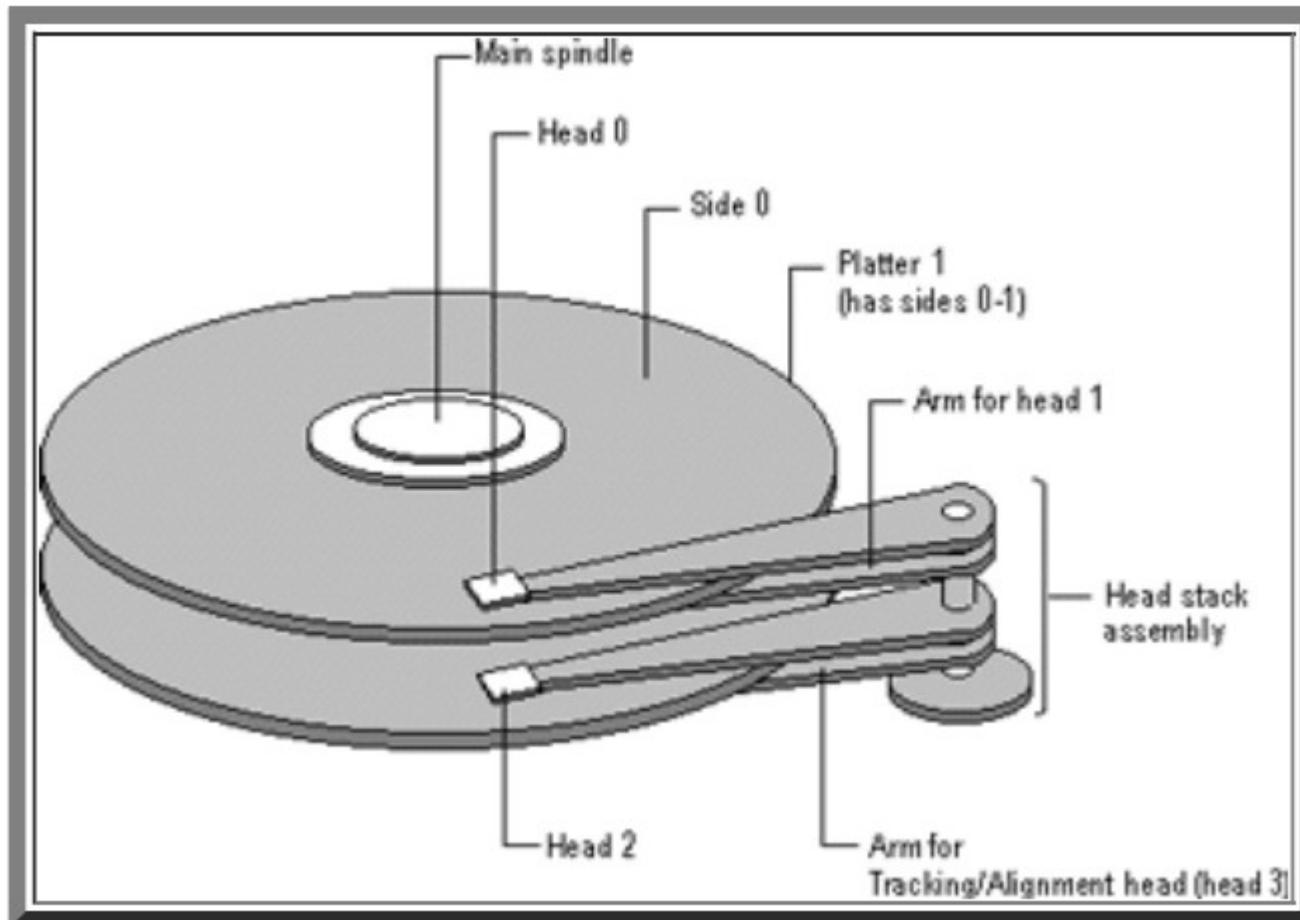
Disk

- Figure shows the views of files at each of the levels of file system abstraction
- Many possible variations in how the file abstraction can be presented to the user processes
- Many important design issues in file system interface design

Hard Disk Drive (HDD)

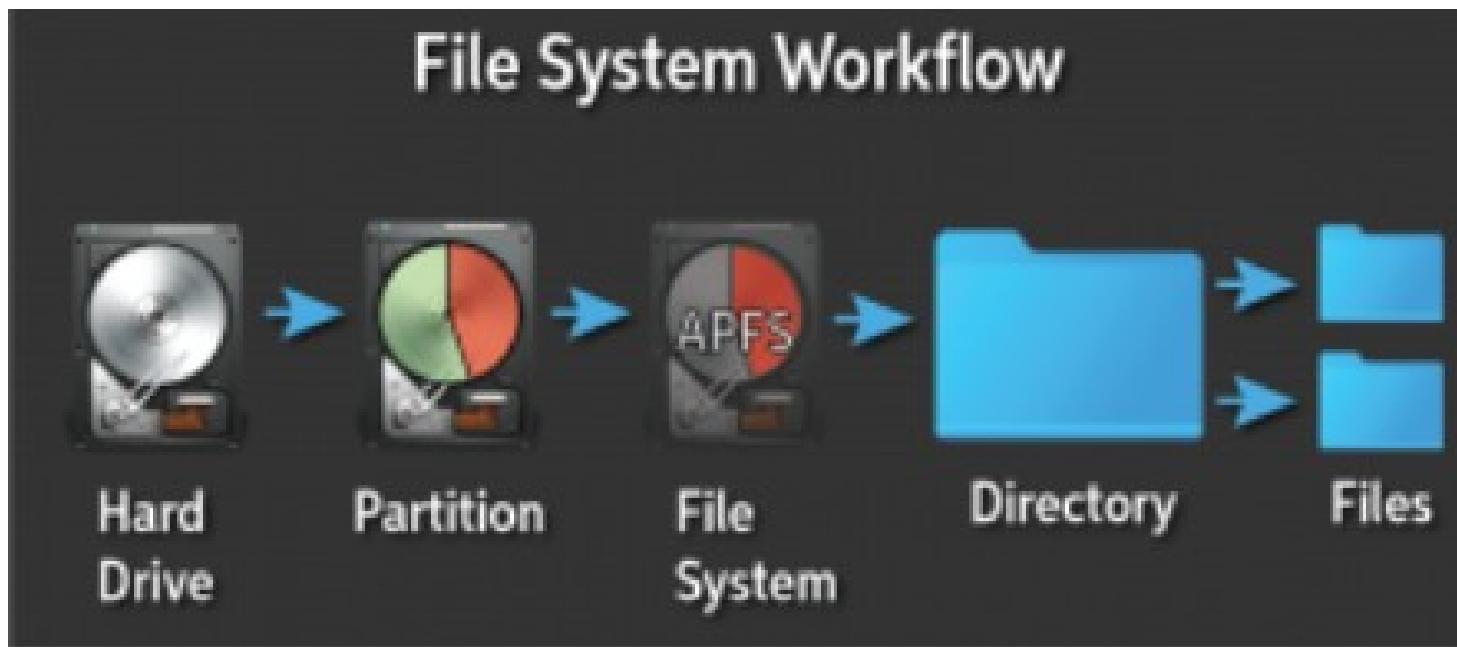
- A non-volatile data storage device
- Usually installed inside desktop computers, mobile devices, consumer electronics
- Used either as the primary or secondary storage device
- External hard drives can be used to expand the storage capacity of a computer or to act as a portable device to back up data.
- Users can store data from multiple devices and physically bring that data with them wherever they go.

Structure of a Hard Disk



- Hard disk consists of a **number of platters**
- The platters rotate at a very high speed (5400 RPM to 10,000 RPM)
- Disk (read/write) heads move over the platter **surface to read and write** (magnetize) data bits
- The disk head can read or write data only when the desired disk surface area is under the disk head.
- Each platter is logically divided into a **number of tracks**
- Each track is logically divided into a **number of sectors**
- Each sector is uniquely identified by its **sector number**
- Disk sectors are usually **512 bytes in size.**

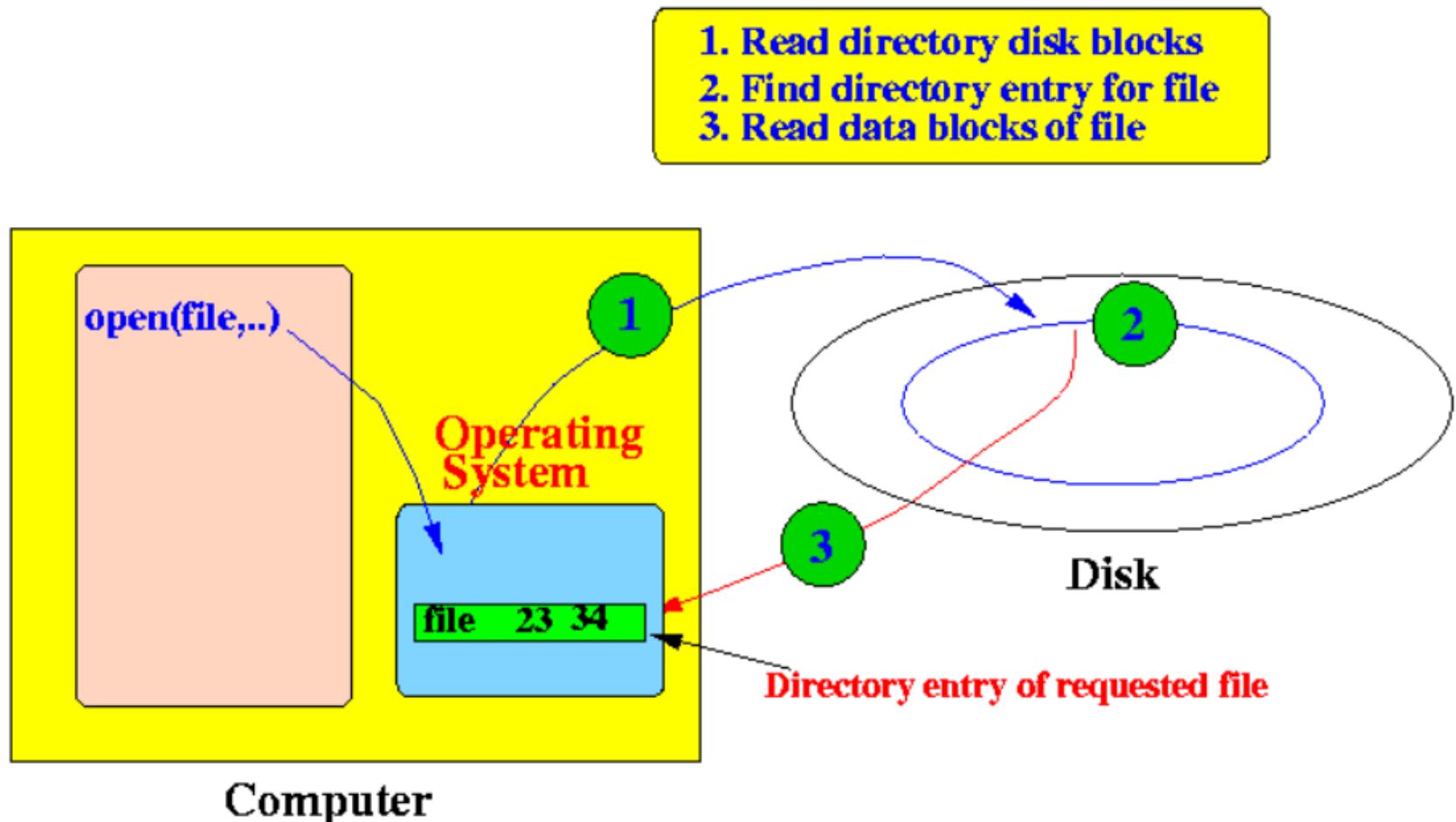
Workflow of a File System



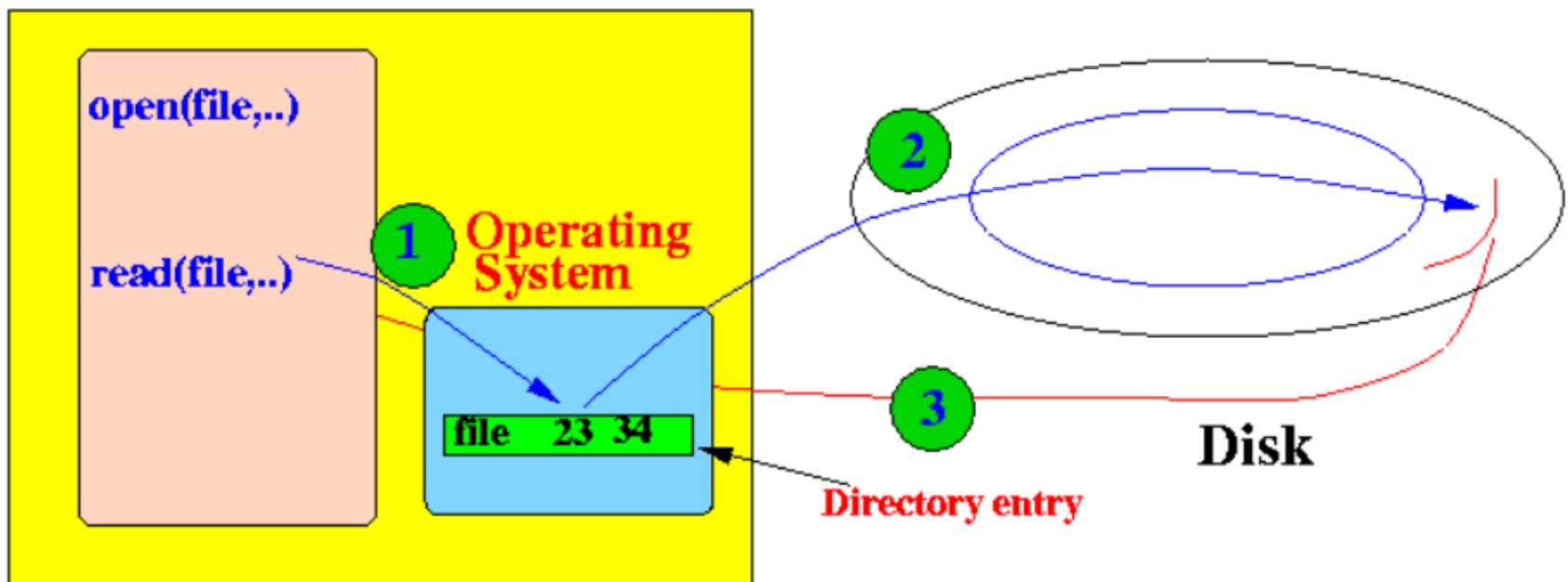
File Storage on Hard Disks

- Each computer system has a (built-in) file system
 - File systems used by Microsoft Windows are: **FAT** (File Allocation Table), **FAT32** and **NTFS**
 - File systems used by Solaris UNIX are: **UFS** (UFS File System) and **ZFS** (ZFS File System)
- The file system always reserve a number of disk blocks (called: directory disk blocks) to store information about the files on the disk
 - The **directory disk blocks** contains a large number of directory entries

Accessing Data Stored in Files on a Hard Disk

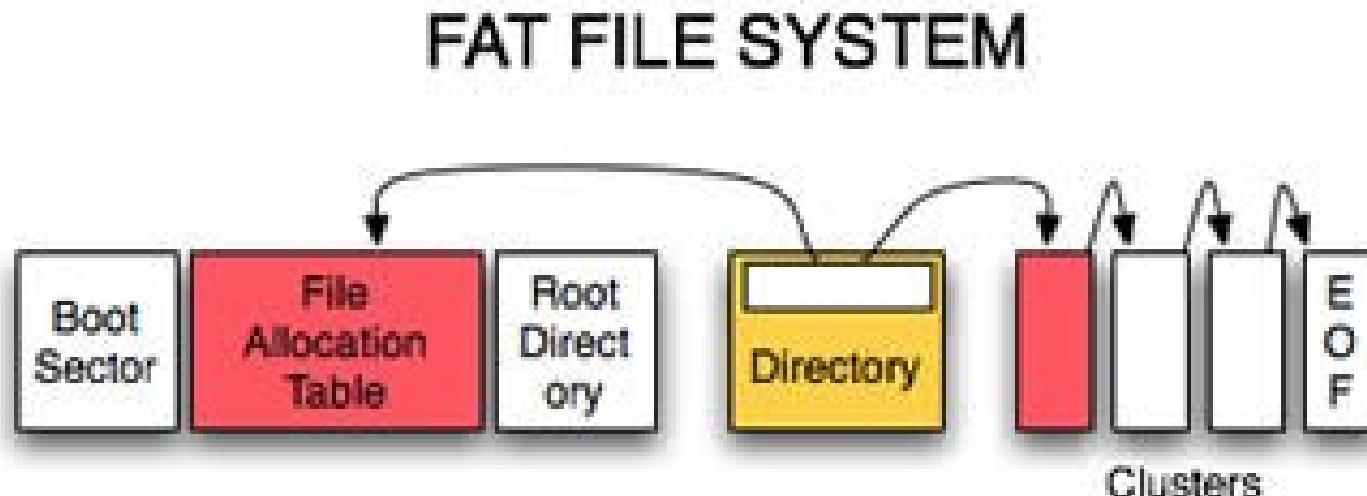


1. Find the block number to read
2. Locate the disk sectors of the block
3. Read data from disk at proper sector



File Allocation Table (FAT)

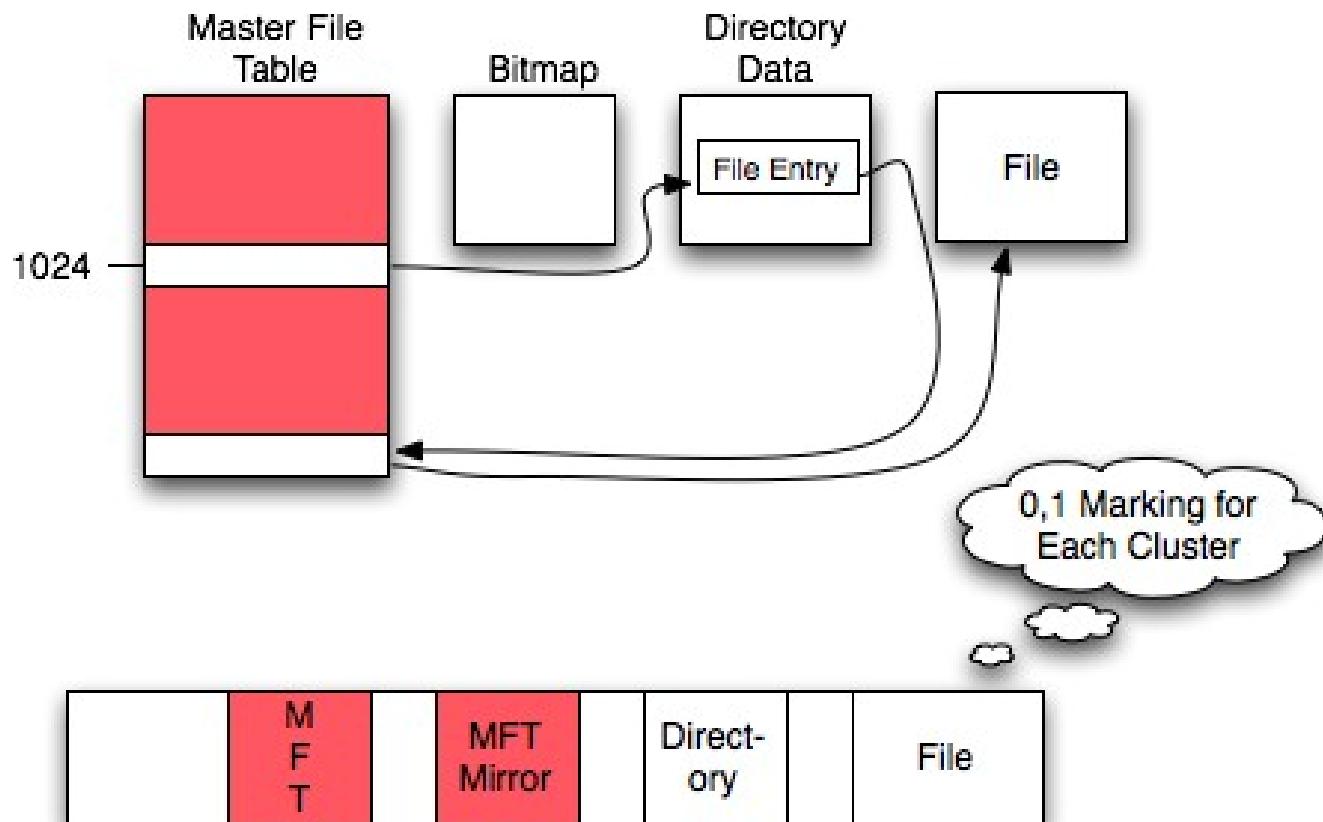
- Simple, handy and robust
- The file system originates from its prominent use of an index table that is allocated at the time of formatting
- Various versions of FAT file systems are FAT 12, FAT 16 and FAT 32



NTFS

- The NTFS - **New Technology File System**
- While accessing (read) files on NTFS partitions on Mac-based systems, unable to transfer data from a Mac to that partition.
- NTFS is **compatible** with some media devices, but it is necessary to **check** the device's owner's manual to see if it's **compatible**.

NTFS FILE SYSTEM

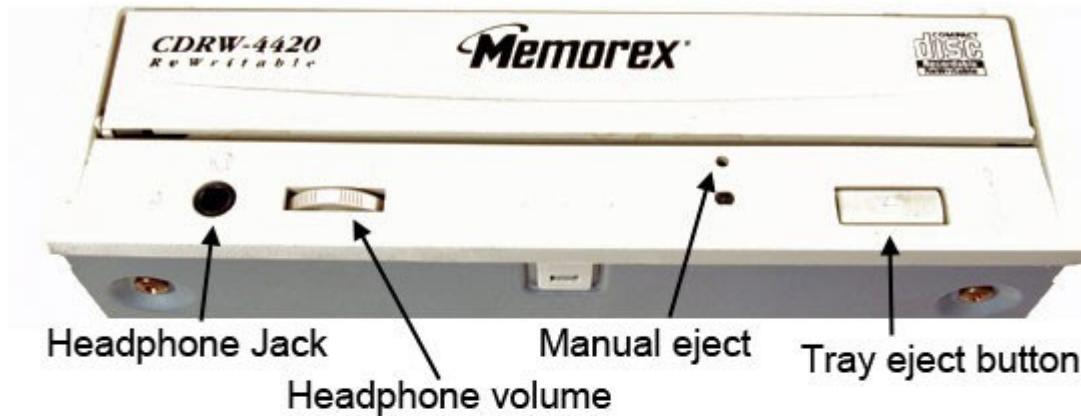


Compact Disc Read Only Memory (CD-ROM)

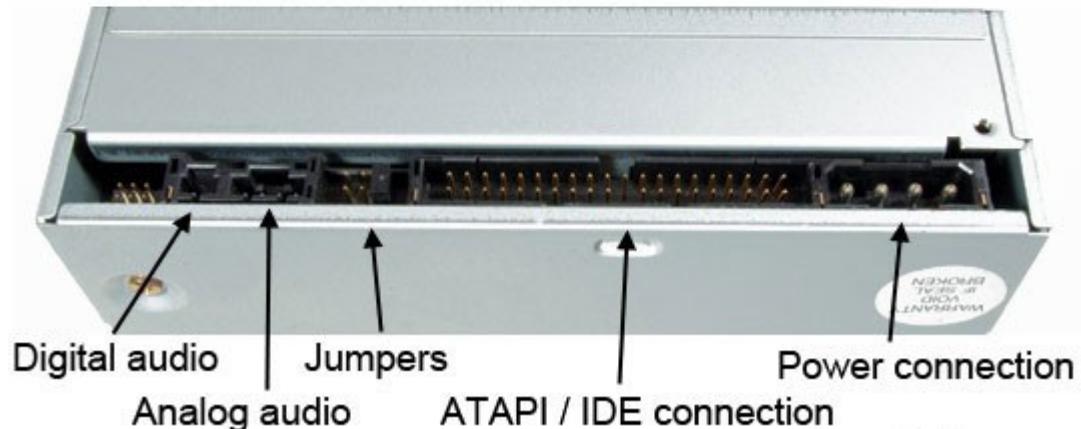


Compact Disc Read Only Memory (CD-ROM)

Front of Disc Drive



Back of Disc Drive



Compact Disc (CD) File System

- Features of **Compact Disc File System (CDFS)** are:
 - It is a file system for **read-only and write-once CD-ROMs**
 - It exports all tracks and boot images on a CD as normal files
 - CDFS provides a wide range of services which include creation, replacing, renaming, or deletion of files on write-once media
 - It uses a **VCACHE driver** to control the CD-ROM disc cache allowing for a smoother playback
 - It includes several disc properties like **volume attributes, file attributes, and file placement**

Types of CDFS

There are **different versions** of Compact Disk File System:

- 1. Clustered operated system. (can be Global or Grid)**
- 2. Flash operated**
- 3. Object file system**
- 4. Semantic**
- 5. Steganographic process**
- 6. Versioning**
- 7. Synthetic operated system**

Advantages of CDFS

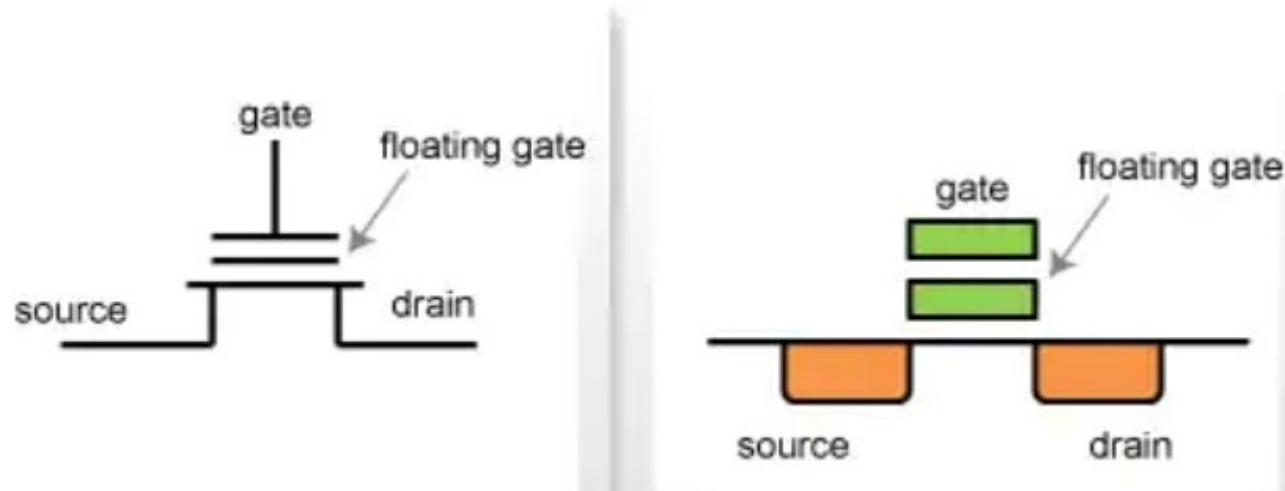
1. Very easy to transfer files from CDs to PC and vice versa.
2. CDFS supports cross-platform compatibility.
3. The CDs are very cheap to produce and so is the system.
4. The system uses a very fast means to transfer data.
5. Most computers read CDs so It is a huge success for Compact Disk File System.

Flash ROM

- A flash ROM is a type of **non-volatile memories** (flash)
- A flash ROM **sets data of all cells** (cell matrix) in a **memory to 1** at once.
- That is, it gets electrons out from the floating gate of all the memory cells at a stretch, and leaves holes.
- ‘Flash’ is named from the **image of momentary flash of lighting**.

Memory Cell of a Flash ROM

- A floating gate is like a pocket for putting in coins.
- If an electron is put in there, the electron will stay in there.
- This electron remains in the floating gate even after a microcontroller is turned off. It is the basic mechanism of a non-volatile memory.



The symbols of the cell of a flash ROM

The sectional view of the cell of a flash ROM

Flash Memory

Advantages

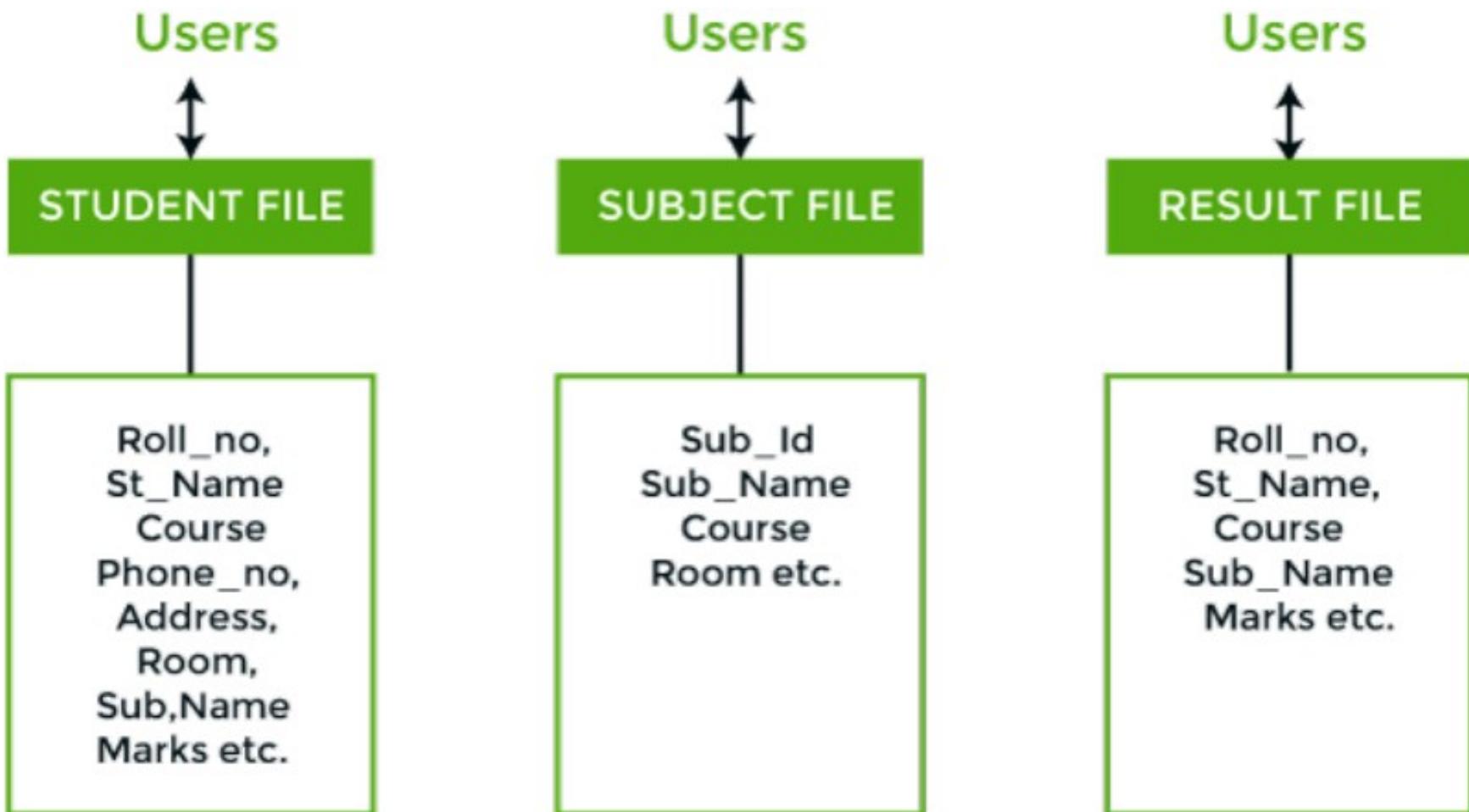
- Faster read and write compared to traditional hard disk drives.
- Smaller size.
- Less prone to damage.
- Cheaper than traditional drives in small storage capacities.
- Uses less power than traditional hard disk drives.

Disadvantages

- Flash memory cells have a limited number of write and erase cycles before failing.
- Smaller size devices, such as flash drives make them easier to lose
- May require a special version of a program to run on a flash-based drive to protect from prematurely wearing out the drive.

File System Vs Databases

- **File based systems** - traditional based approach, a decentralized approach was taken where each department stored and controlled its own data (data processing specialist)
- They create the necessary computer file structures, and manage the data within structures and design some application programs that create reports based on file data.



DataBase Management System (DBMS)

- **A database approach** - a well-organized collection of data that are related in a meaningful way which can be accessed by different users but stored only once in a system.
- The **DBMS system operations** are:
 - Insertion**,
 - Deletion**,
 - Selection**,
 - Sorting** etc.

STUDENT'S FILE

Roll_no,
St_Name
Course
Phone_no,
Address,
Room,
Sub,Name
Marks etc.

SUBJECT FILE

Sub_Id
Sub_Name
Course
Room etc.

RESULT FILE

Roll_no,
St_Name,
Course
Sub_Name
Marks etc.

File System and DBMS

	File System	DBMS
Meaning	The user has to write the procedures for managing the database	The user is not required to write the procedures
Sharing of data	It may be of different formats, so it isn't easy to share data.	Data sharing is easy
Data	Provides the detail of the data representation and storage of data	An abstract view of data that hides the details
Security and Protection	It isn't easy to protect a file under the file system	Good protection mechanism
Recovery Mechanism	The file system doesn't have a crash mechanism	DBMS protects the user from system failure

	File System	DBMS
Manipulation Techniques	Can't efficiently store and retrieve the data	Sophisticated techniques to store and retrieve the data
Concurrency Problems	Concurrent access has many problems like redirecting the file while deleting some information or updating some information	Concurrent access of data using some form of locking
Cost	Cheaper to design	Expensive to design
Data Redundancy and Inconsistency	There exists a lot of duplication of data	Problems are controlled
Structure	Simple structure	Complex to design

	File System	DBMS
Data Redundancy and Inconsistency	There exists no Data Independence	Data Independence exists, and it can be of two types. Logical Data Independence Physical Data Independence
Integrity Constraints	Difficult to implement	Easy to apply
Data Models	No concept of data models exists	Three types of data models exist: Hierarchical, Network and Relational data models
Flexibility	Less	More
Examples	Cobol, C++ etc	Oracle, SQL Server, Sybase etc

What is a file system?

- Speaking broadly, a file system is the logical means for an operating system to store and retrieve data on the computers hard disks, be they local drives, network-available volumes, or exported shares in a storage area network (SAN)

What is a file system?

A file system can refer to the methods and data structures that an operating system uses to keep track of files on a disk or partition

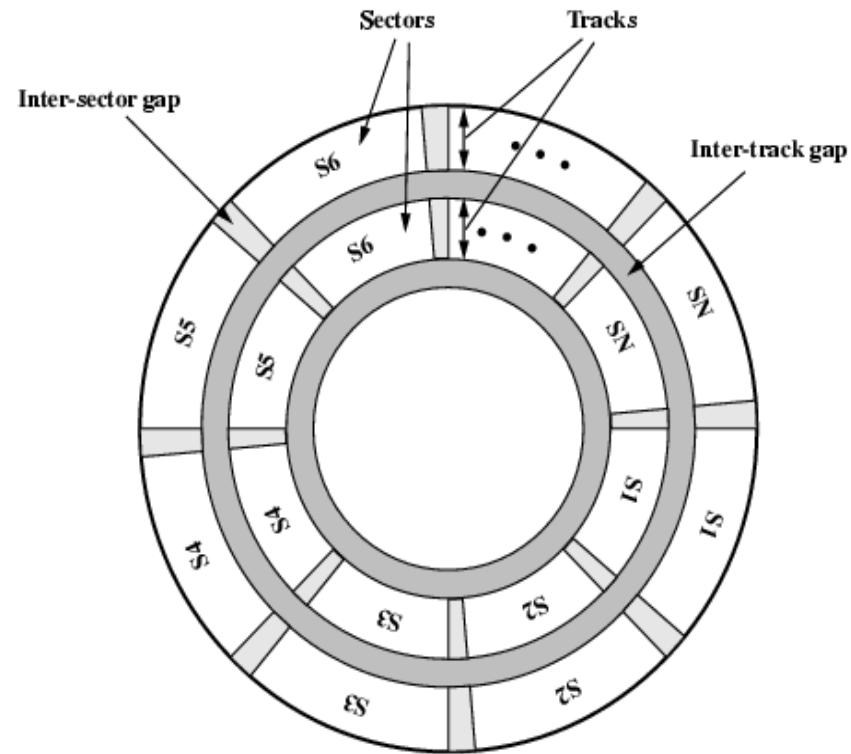
- Linux keeps regular files and directories on block devices such as disks
- A Linux installation may have several physical disk units, each containing one or more file system types
- Partitioning a disk into several file system instances makes it easier for administrators to manage the data stored there

Magnetic Disk

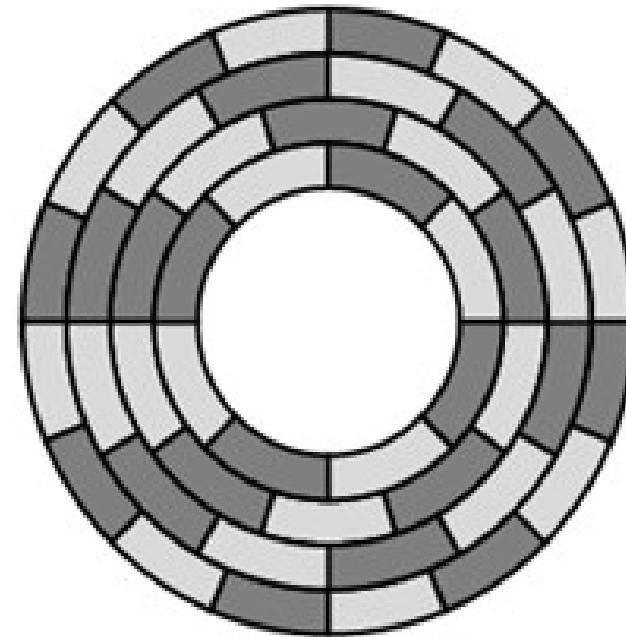
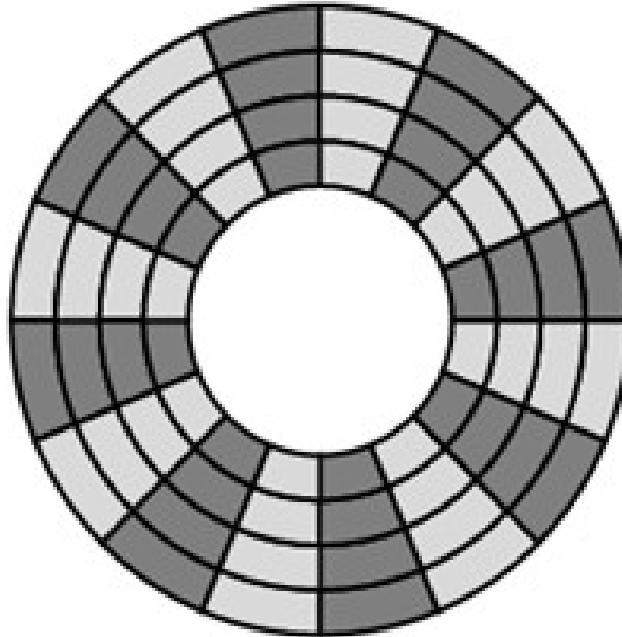


Data Organization on Disk

- Concentric rings called tracks
 - Gaps between tracks
 - Same number of bits per track
 - Constant angular velocity
- Tracks divided into sectors
- Minimum block size is one sector
- Disk rotates at const. angular velocity
 - Gives pie shaped sectors
- Individual tracks and sectors addressable



Multi Zone Recording Disks



Disk storage capacity in a CAV system is limited by the maximum recording density that can be achieved on the innermost track. (constant angular velocity)

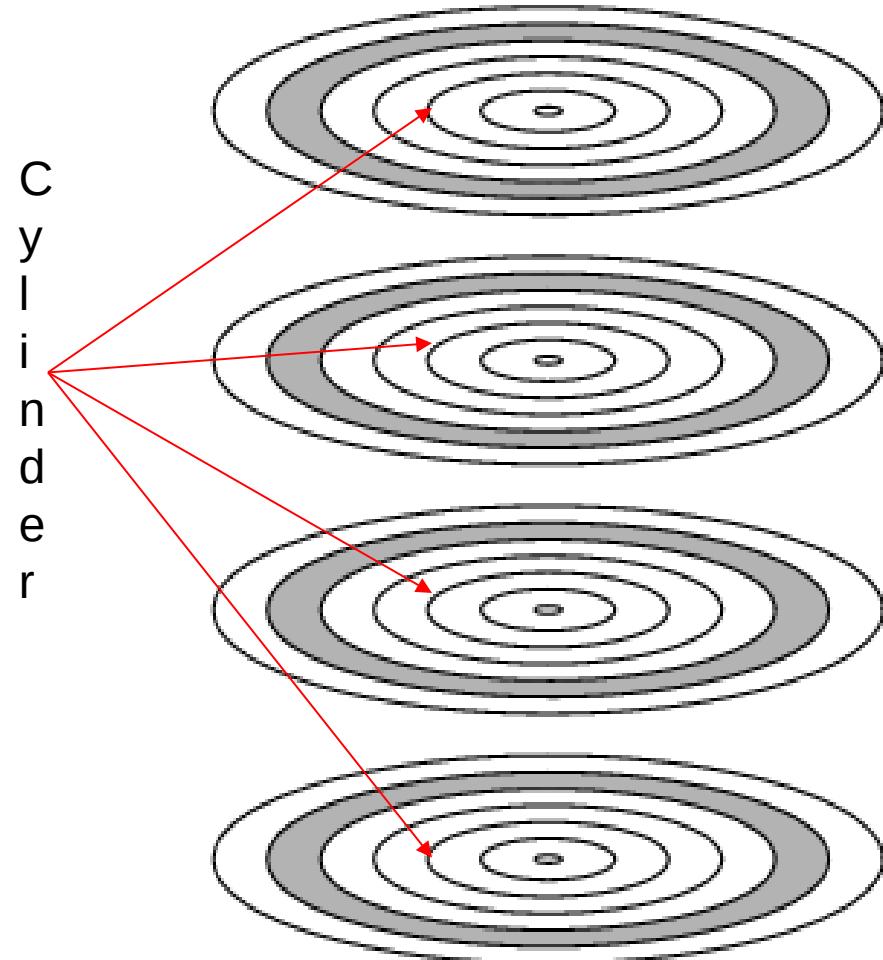
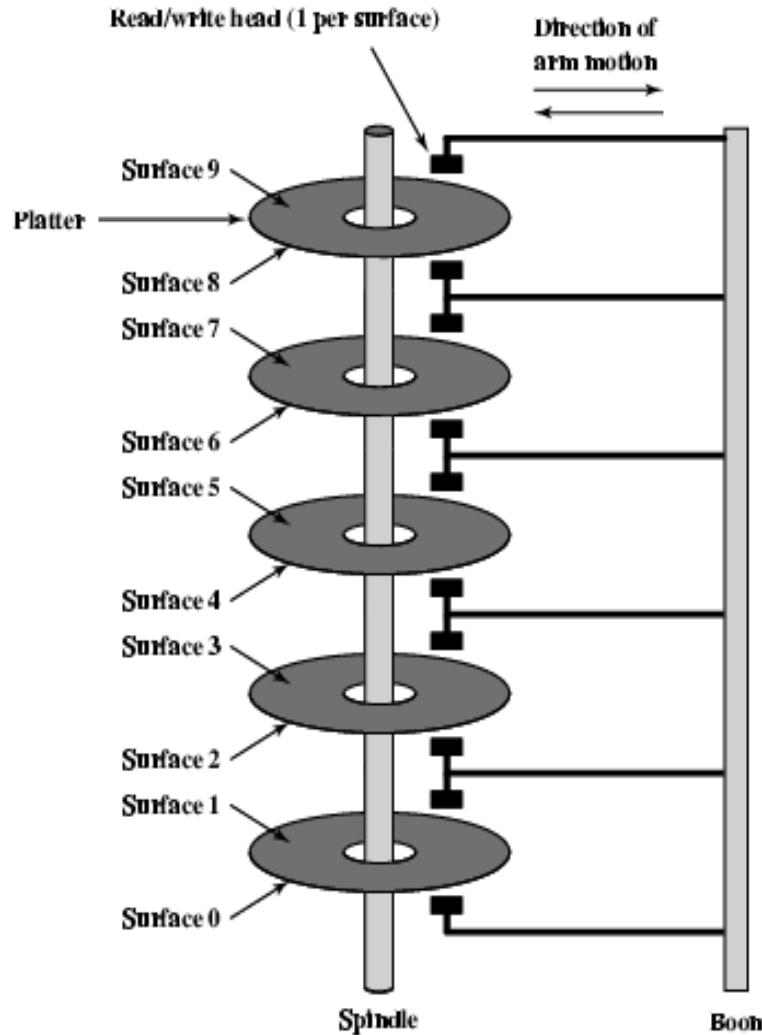
Read and Write Mechanisms-1

- Recording & retrieval of data via conductive coil which is called a **head**
- May be single read/write head or separate ones
- During read/write, head is stationary, platter rotates
- **Write**
 - Electricity flowing through coil produces magnetic field
 - Electric pulses sent to head
 - Magnetic pattern recorded on surface below

Disk Characteristics

- Fixed (rare) or movable head
 - Fixed head
 - One read/write head per track mounted on fixed ridged arm
 - Movable head
 - One read/write head per side mounted on a movable arm
- Removable or fixed disk
- Single or double (usually) sided
- Head mechanism
 - Contact (Floppy), Fixed gap, Flying (Winchester)
- Single or multiple platter

Multiple Platters Tracks and Cylinders



Capacity

- Vendors express capacity in units of gigabytes (GB), where $1\text{ GB} = 10^9\text{ Byte}$
- Capacity is determined by these technology factors:
 - **Recording density (bits/in)**: number of bits that can be squeezed into a 1 inch segment of a track.
 - **Track density (tracks/in)**: number of tracks that can be squeezed into a 1 inch radial segment.
 - **Areal density (bits/in²)**: product of recording and track density.
- Modern disks partition tracks into disjoint subsets called recording zones

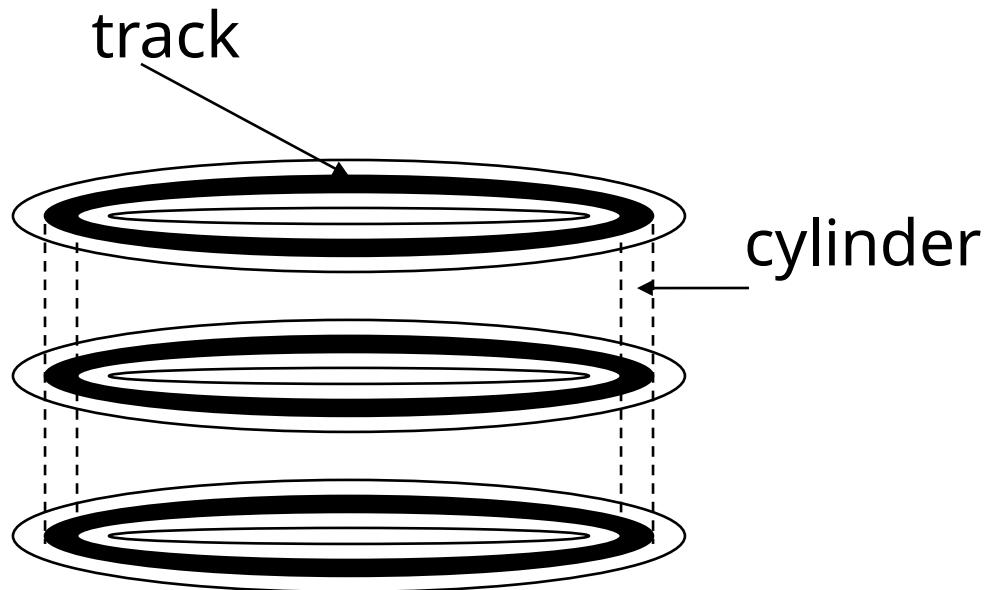
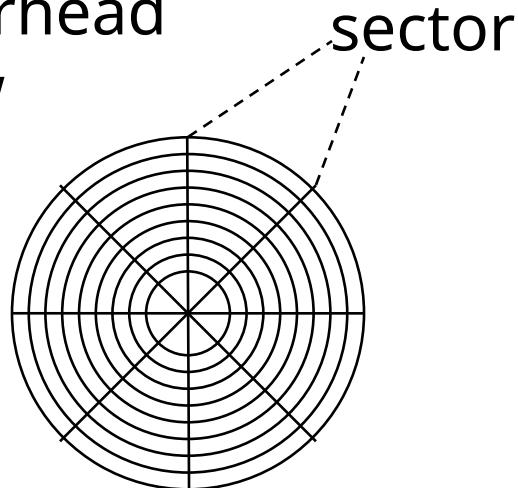
Computing Disk Capacity



- Capacity = (# bytes/sector) x (avg. # sectors/track) x (# tracks/surface) x (# surfaces/platter) x (# platters/disk)
- Example:
 - 512 bytes/sector, 300 sectors/track (average)
 - 20,000 tracks/surface, 2 surfaces/platter
 - 5 platters/disk
 - Capacity = $512 \times 300 \times 20000 \times 2 \times 5 = 30.72\text{GB}$

What is a file system?

Overhead view



Disk blocks are composed of one or more contiguous sectors

The same track on each platter in a disk makes a cylinder; partitions are groups of contiguous cylinders

What is a file system?

- File system instances reside on partitions
- Partitioning is a means to divide a single hard drive into many logical drives
- A partition is a contiguous set of blocks on a drive that are treated as an independent disk
- A partition table is an index that relates sections of the hard drive to partitions

The UNIX File System

- **The root directory has many subdirectories. The following table describes some of the subdirectories contained under root**

Directory	Content
/bin	Common programs , shared by the system, the system administrator and the users.
/dev	Contains references to all the CPU peripheral hardware , which are represented as files with special properties.
/etc	Most important system configuration files are in /etc, this directory contains data similar to those in the Control Panel in Windows
/home	Home directories of the common users .
/lib	Library files , includes files for all kinds of programs needed by the system and the users.
/sbin	Programs for use by the system and the system administrator.
/tmp	Temporary space for use by the system , cleaned upon reboot, so don't use this for saving any work!
/usr	Programs, libraries, documentation etc. for all user-related programs.
/var	Storage for all variable files and temporary files created by users, such as log files, the mail queue, the print spooler area, space for temporary storage of files .

UNIX and POSIX file attributes

- **File type** : Type of file
- **Access permission** : The file access permission (owner, group, others)
- **Hard link count** : Number of hard links of a file
- **UID** : The file owner user ID
- **GID** : The file group ID
- **File size** : The file size in bytes
- **Last access time** : The time the file was last accessed
- **Last modify time** : The time the file was last modified
- **Last change time** : The time the file access permission

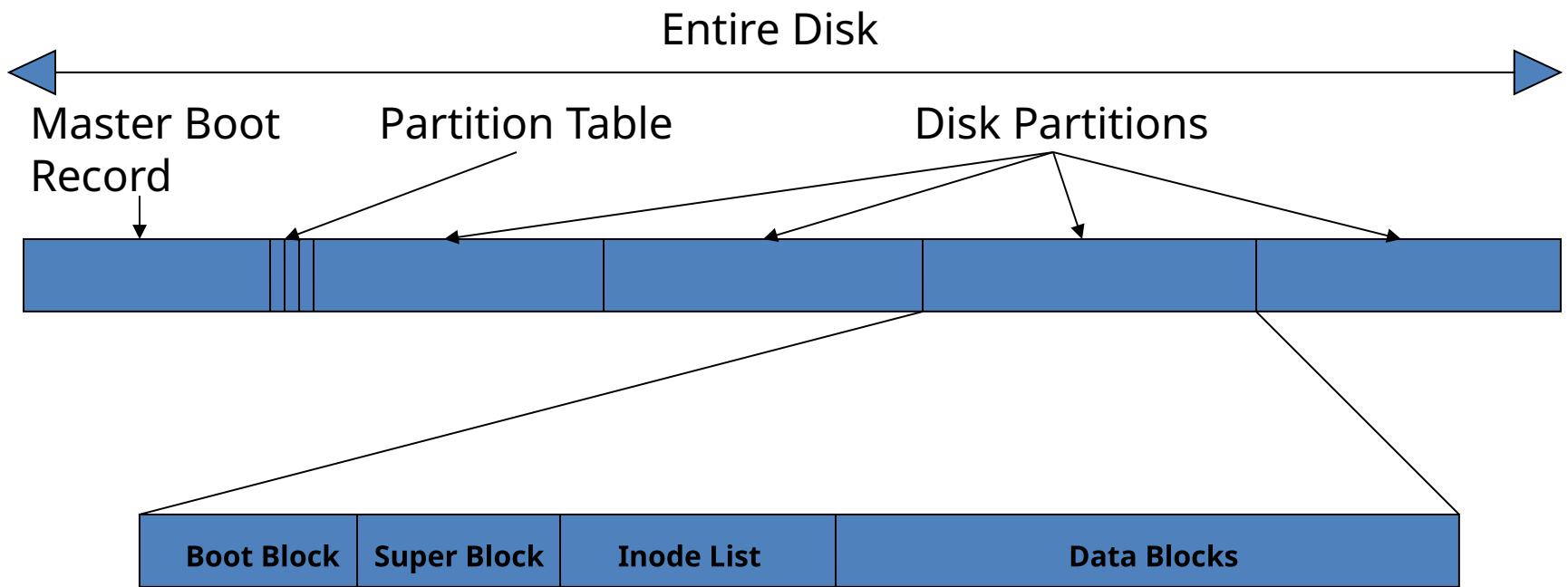
UID ,GID or hard link count was last changed

- **Inode number** : The system inode number of the file

INTERNAL REPRESENTATION OF FILES

- Inodes
- Structure of a regular file
- Directories
- Conversion of a path name to an inode
- Super block
- Inode assignment to a new file
- Allocation of disk blocks
- Other file types

File System Implementation (Disk Layout)



A Possible File System Instance Layout

File system layout

- The central structural concepts of a file system type are:
 - Boot Block
 - Super Block
 - Inode List
 - Data Block



Boot Block

Boot Block:

- Occupies the beginning of a file system
- Typically residing at the first sector, it may also contain the bootstrap code that is read into the machine at boot time
- Although only one boot block is required to boot the system, every file system may contain a boot block

Super Block

Super Block:

- Describes the state of a file system
- How large it is
- How many files it can store
- Where to find free space in the file system
- Consists of following fields
 - List of free Inodes in the file system
 - Index of the next free Inode in the free Inode list
 - Lock fields for the free block and free Inode lists
 - A flag indicating that the super block has modified.

Kernel periodically writes super block into the disk for consistency.



Super Block

- Consists of following fields
 - Size of the file system
 - Number of free blocks in the file system
 - List of free blocks available in the file system
 - Index of the next free block in the free block list
 - Size of the Inode list
 - Number of free Inodes in the file system



Inode List

Inode List:

- An inode is the internal representation of a file
 - contains the description of the disk layout of the file data
 - file owner
 - permissions
- The inode list contains all of the inodes present in an instance of a file system



Data Blocks

Data Blocks:

- Contain the file data in the file system
- Additional administrative data
- An allocated data block can belong to one and only one file in the file system



What is a file system?

So what is a file system?

- A file system is a set of abstract data types that are implemented for the storage, hierarchical organization, manipulation, navigation, access, and retrieval of data

Inodes in UNIX system

- UNIX system has an **inode table** which keeps track of all files
- Each entry in inode table is an **inode record**
- **_inode record** contains all attributes of file including **inode number** and **physical address** of file
- Information of a file is accessed using its **inode number**
- **Inode number** is unique within a **file system**
- A file record is identified by a **file system ID** and **inode number**
- **Inode record** does not contain the **name of the file**
- The mapping of filenames to inode number is done via **directory files**

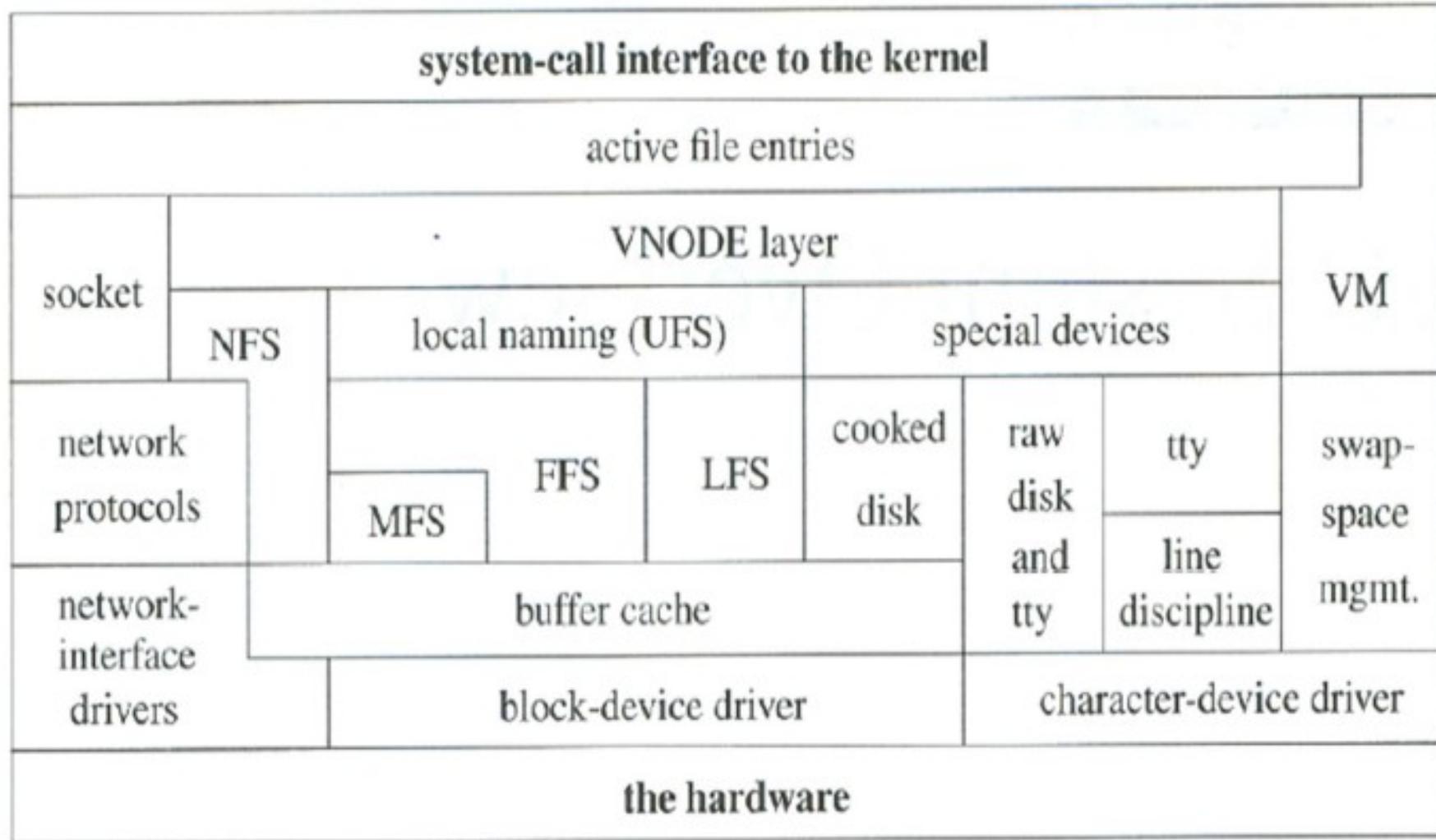
What is a file system?

- The kernel deals on a logical level with file systems rather than with disks
- The separate file systems that the system may use are not accessed by device identifiers
- Instead they are combined into a single hierarchical tree structure that represents the file systems as one whole single entity

File system consistency check

- During bootstrap operation, all the file systems are checked for self – consistency and attached to the root system
 - UNIX command: fsck
- To find the available space on the mounted file systems use UNIX command df (disc free space)

Kernel I/O structure



- Fast File System (FFS): It assumes that the average file size is small and that some files (those in the same directory) will be accessed all at once. Given these assumptions, it divides the disk surface in different cylinder groups and lays out blocks that may be accessed together in the same group. This reduces expensive seeks.
- Log-structured File System (LFS): It assumes that the system bottleneck are writes to the disk, so it packs all writes in a concrete area of the disk, treating the overall space as an append-only log (no seeks required).
- Memory File System (MFS): Reserves a contiguous region of memory and lays out blocks in it. This is why MFS is considered inefficient, because it is using the concepts of an on-disk file-system over memory.

I/O in Unix - Devices

- I/O system used for
 - I/O devices, Network communication and Virtual Memory (Swap space)

```
[SysPgm@labserver ~] $ ls -l /dev/ ↵
```

- Observe File types
 - c for Character device driver
 - b for Block device driver
 - l for symbolic link
- Observe file size [Replaced by 2 integer numbers]
 - Major device number and minor device number

I/O in Unix - Devices

- Block Devices and Character Devices
 - Character devices – i.e. devices treated as byte streams
 - Include terminals, line printers, main memory apart from disks.
 - Unstructured or raw interface – no buffering by kernel.
 - Block Devices – i.e. block-addressable devices
 - E.g. Disks and Tapes
 - Structured interface – block-sized accesses – buffering done by kernel.

I/O in Unix - Devices

Device Identification – major device #, minor device #

- Major Device # refers to the type of device and hence is used to identify a device driver.
 - When a device file is opened, Linux examines its major number and forwards the call to the driver registered for that device.
 - Minor Device # refers to a specific device and is interpreted by a device driver.
 - Minor number is used to identify the specific device instance if the driver controls more than one device of a type.
- Block Device table and Character Device Table

File Systems and inodes

Every file is associated with a table that contains all that you could possibly need to know about a file—except its name and contents. This table is called the **inode** (shortened from index node) and is accessed by the **inode number**. The inode contains the following attributes of a file:

- File type (regular, directory, device, etc.).
- File permissions (the nine permissions and three more).
- Number of links (the number of aliases the file has).
- The UID of the owner.
- The GID of the group owner.
- File size in bytes.
- Date and time of last modification.
- Date and time of last access.
- Date and time of last change of the inode.
- An array of pointers that keep track of all disk blocks used by the file.

File Systems and inodes

The **ls** command reads the inode to fetch a file's attributes, and it can list most of them using suitable options. One of them is the **-i** (inode) option that tells you the inode number of a file:

```
$ ls -il tulec05
9059 -rw-r--r-- 1 kumar    metal      51813 Jan 31 11:15 tulec05
```

The file *tulec05* has the inode number 9059. No other file *in the same file system* can have this number unless the file is removed. When that happens, the kernel will allocate this inode number to a new file.

Note: The inode contains all attributes of a file except the filename. The filename is stored in the directory housing the file. The inode number is also not stored in the inode, but the kernel can locate any inode by its position since its size is fixed.

Hard Links

Why is the filename not stored in the node? So that a file can have multiple filenames. When that happens, we say the file has more than one **link**. We can then access the file by any of its links. All names provided to a single file have one thing in common; they all have the same inode number.

The link count is displayed in the second column of the listing. This count is normally 1 (as shown in the previous listing), but the following files have two links:

```
-rwxr-xr--  2 kumar  metal   163 Jul 13 21:36 backup.sh
-rwxr-xr--  2 kumar  metal   163 Jul 13 21:36 restore.sh
```

All attributes seem to be identical, but the “files” could still be copies (which have different inode numbers). It’s the link count that seems to suggest that the “files” are linked to each other. But this can only be confirmed by using the **-i** option to **ls**:

```
$ ls -li backup.sh restore.sh
```

```
478274 -rwxr-xr--  2 kumar  metal   163 Jul 13 21:36 backup.sh
478274 -rwxr-xr--  2 kumar  metal   163 Jul 13 21:36 restore.sh
```

In: Creating Hard Links

A file is linked with the **ln** (link) command, which takes two filenames as arguments. The command can create both a *hard* and a *soft* link (discussed later) and has a syntax similar to the one used by **cp**. The following command (hard) links **emp.1st** with **employee**:

```
ln emp.1st employee
```

employee must not exist

The **-i** option to **ls** shows that they have the same inode number, meaning that they are actually one and the same file:

```
$ ls -li emp.1st employee
```

```
29518 -rwxr-xr-x 2 kumar metal
```

inode

```
915 May 4 09:58 emp.1st
```

inode

```
915 May 4 09:58 employee
```

inode

The link count, which is normally one for unlinked files, is shown to be two. You can link a third filename, **emp.dat**, and increase the number of links to three:

```
$ ln employee emp.dat ; ls -l emp*
```

```
29518 -rwxr-xr-x 3 kumar metal
```

```
915 May 4 09:58 emp.dat
```

```
29518 -rwxr-xr-x 3 kumar metal
```

```
915 May 4 09:58 emp.1st
```

```
29518 -rwxr-xr-x 3 kumar metal
```

```
915 May 4 09:58 employee
```

WHERE TO USE HARD LINKS

1. In data/ foo.txt input_files
2. Links provide some protection against accidental deletion
3. Because of links, we don't need to maintain two programs as two separate disk files.

LIMITATIONS OF HARD LINKS

1. We cant have two linked filenames in two file systems.
2. We cant link a directory even within the same file system

This can be solved by using symbolic links (soft links)

Symbolic Links

We have seen how links let us have multiple names for a file. These links are often called **hard links**, and have two limitations:

- You can't have two linked filenames in two file systems. In other words, you can't link a filename in the `/usr` file system to another in the `/home` file system.
- You can't link a directory even within the same file system.

This serious limitation was overcome when **symbolic links** made their entry. Until now, we have divided files into three categories (ordinary, directory and device). The symbolic link is the fourth file type considered in this text. Unlike the hard link, a symbolic link doesn't have the file's contents, but simply provides the pathname of the file that actually has the contents. Being more flexible, a symbolic link is also known as a **soft link**. Windows shortcuts are more like symbolic links.

Symbolic Links

The **ln** command creates symbolic links also, except that you have to use the **-s** option. This time the listing tells you a different story:

```
$ ln -s note note.sym
$ ls -li note note.sym
9948 -rw-r--r-- 1 kumar  group          80 Feb 16 14:52 note
9952 1rwxrwxrwx 1 kumar  group          4 Feb 16 15:07 note.sym -> note
```

You can identify symbolic links by the character **l** (el) seen in the permissions field. The pointer notation **-> note** suggests that **note.sym** contains the pathname for the filename **note**. It's **note**, and not **note.sym**, that actually contains the data. When you use **cat note.sym**, you don't actually open the symbolic link, **note.sym**, but the file the link points to. Observe that the size of the symbolic link is 4; this is the length of the pathname it contains (**note**).

It's important you realize that this time we indeed have two “files”, and they are not identical. Removing **note.sym** won't affect us much because we can easily recreate the link. But if we remove **note**, we would lose the file containing the data. In that case, **note.sym** would point to a nonexistent file and become a *dangling* symbolic link.

Thank you

Q & A



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN

SESSION 12

Course design by - Dr. Lucy J. Gudino &

Dr Chandrasekhar

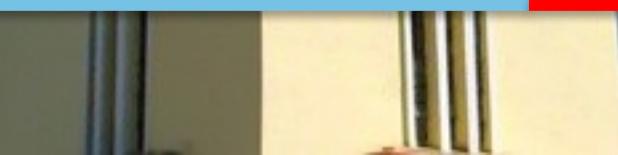
Faculty - Dr Thangakumar J

WILP & Department of CS & IS



BITS Pilani
Pilani Campus

Operating Systems - File as a Logical Abstraction of Storage



Last Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Levels in the file system• File storage on hard disks, CD ROMs and Flash ROMs• File Vs Databases	T1(16.1.2-16.1.3)

File as a Logical Abstraction of Storage



Today's topic

- File Access Methods – Effect of Locality
- File System Implementation
- Example (UNIX or Linux File System/
Windows NTFS/Mobile file System/ Flash
Memory File System)

File Access Methods - Effect of Locality

- **Problem of old UNIX file system** - treated the disk like it was a random-access memory; data was spread all over the place so had a **real and expensive positioning costs**.
- **Getting fragmented**
- Smaller blocks were good because they **minimized internal fragmentation**, but bad for transfer as each block might require a **positioning overhead** to reach it.

Fast File System (FFS)

- To design the file system structures and allocation policies to be “disk aware”; improve performance.
- Keeping the same interface to the file system, but changing the internal implementation leads to performance and reliability.

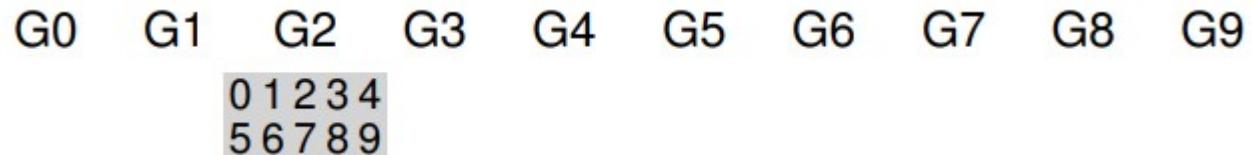
Organizing Structure: The Cylinder Group

- FFS divides the disk into a bunch of groups known as cylinder groups (block groups)
- FFS can ensure that accessing one after the other will not result in long seeks across the disk.
- It allocates files and directories within each of these groups.
- A per-group inode bitmap (ib) and data bitmap (db) serve this role for inodes and data blocks in each group.



The Large-File Exception

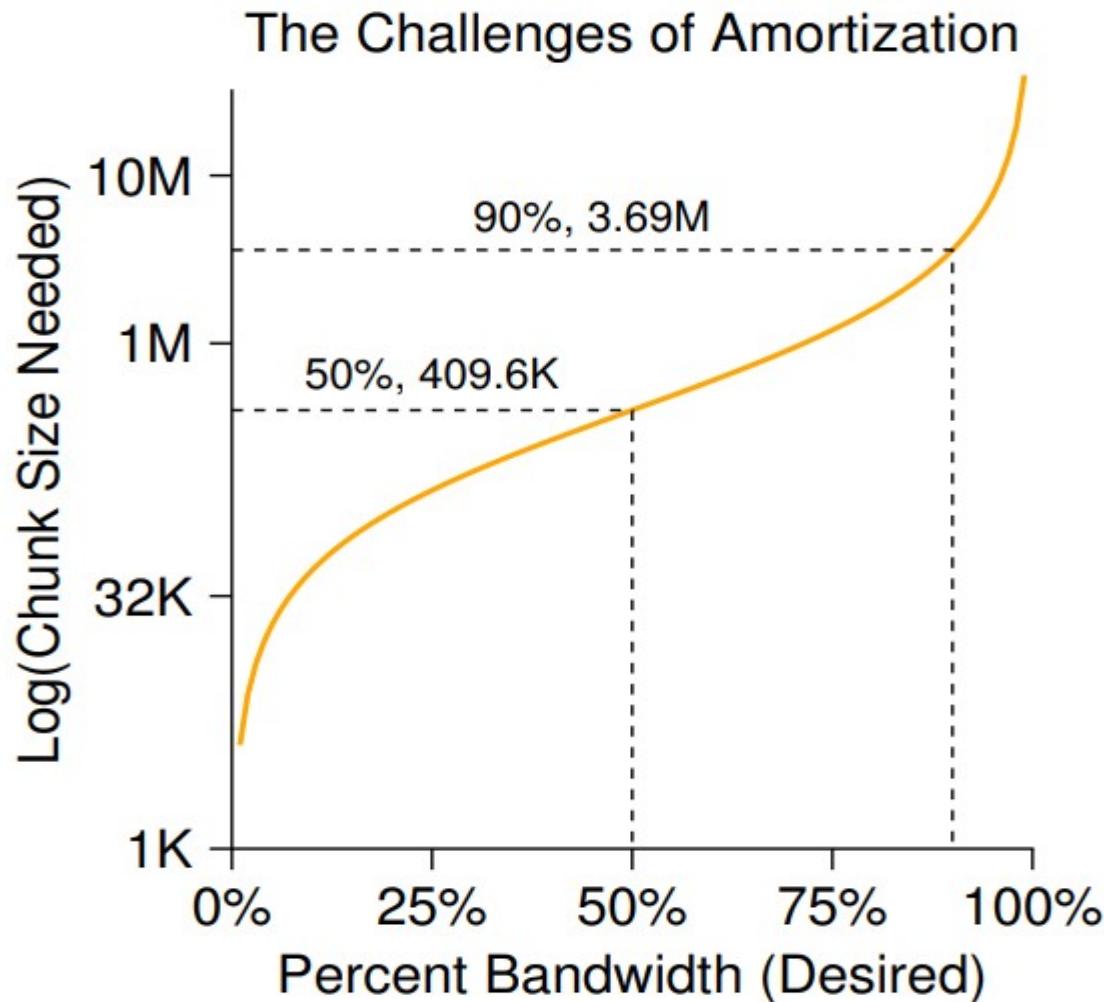
- File would entirely fill the block group it is first placed within.
- Filling a block group is undesirable, as it prevents subsequent “related” files from being placed within this block group, and thus may hurt file-access locality.
- The depiction of FFS **without the large-file exception**:



- With the large-file exception, the file spread across the disk in chunks:



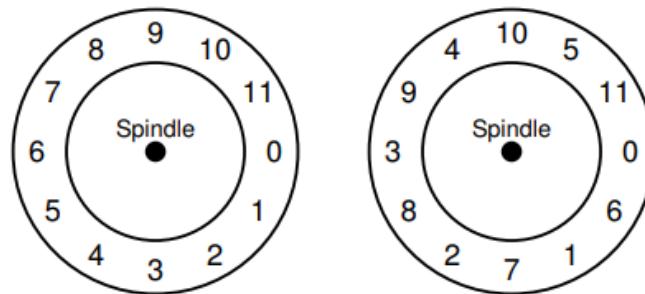
Amortization: How Big Do Chunks Have To Be?



- Many files were 2KB or so in size back then, and using 4KB blocks, while good for transferring data, was not so good for space efficiency.
- This **internal fragmentation** leads to roughly half the disk being wasted for a typical file system.
- Introduced **sub-blocks**, which were **512-byte little blocks** that the file system could allocate to files
- FFS will find a 4KB block, **copy the sub-blocks into it**, and **free the sub-blocks for future use.**

FFS: Standard Versus Parameterized Placement

- **Parameterization** - FFS was smart enough to figure out for a particular disk how many blocks it should skip in doing layout to avoid the extra rotations
- Modern disks internally read the entire track in and buffer it in an internal disk cache – **track buffer**
- **Symbolic links** allow the user to create an “alias” to any other file or directory on a system and thus are much more flexible.
- FFS also introduced an atomic **rename()** operation for renaming files.



FILE SYSTEM IMPLEMENTATION

- To implement the file system, there is a need to implement three objects the file system provides for users:

–Files

–Open Files

–Directories

- Each one is implemented as a data structure and a set of procedures to manipulate the data structure.

File Descriptor

- **File descriptor** is simply an index into the file descriptor table. For each process in our operating system, there is a **process control block**(PCB). PCB keeps track of the context of the process. So one of the fields within this is an array called file descriptor table.

- This array keeps track of all the resources that the process owns and can operate on. The file descriptor table holds pointers to the resources. Resources could be
 - File
 - Terminal I/O
 - Pipes
 - Socket for the communication channel between machines
 - Any devices

- So if any process opens or uses any resources will have an entry in the file descriptor table. Any process when it first starts is given access to three resources.
- These resources are:
- stdin
- stdout
- stderr

- Suppose we are working on the terminal then input for it is stdin, the output is stdout and the error is stderr.
- And if we open any other terminal then both are two different processes now. The newly opened terminal will have its own stdin, stdout because it's a different process.

File Descriptor

File Descriptor

- A number to describe an open file or I/O resource in system
 - Unique non-negative integer
- Describes resource and how it can be accessed
- When a process requests for resource:
 - Kernel grants access (if valid)
 - Creates entry in global file table
 - Provides process with location of that entry

- The file descriptor is just an integer that you get from the **open()** system call.
- Example of file descriptor:

int fd = open(filePath, mode);

File Pointer

- **File pointer** is a pointer returned by **fopen()** library function. It is used to identify a file. It is passed to a **fread()** and **fwrite()** function.
- Example of file pointer:

```
FILE *fp;  
fp = fopen("sample.txt,"a");  
fprintf(fp, "Welcome to GFG");  
fclose(fp);
```

File System Data Structures

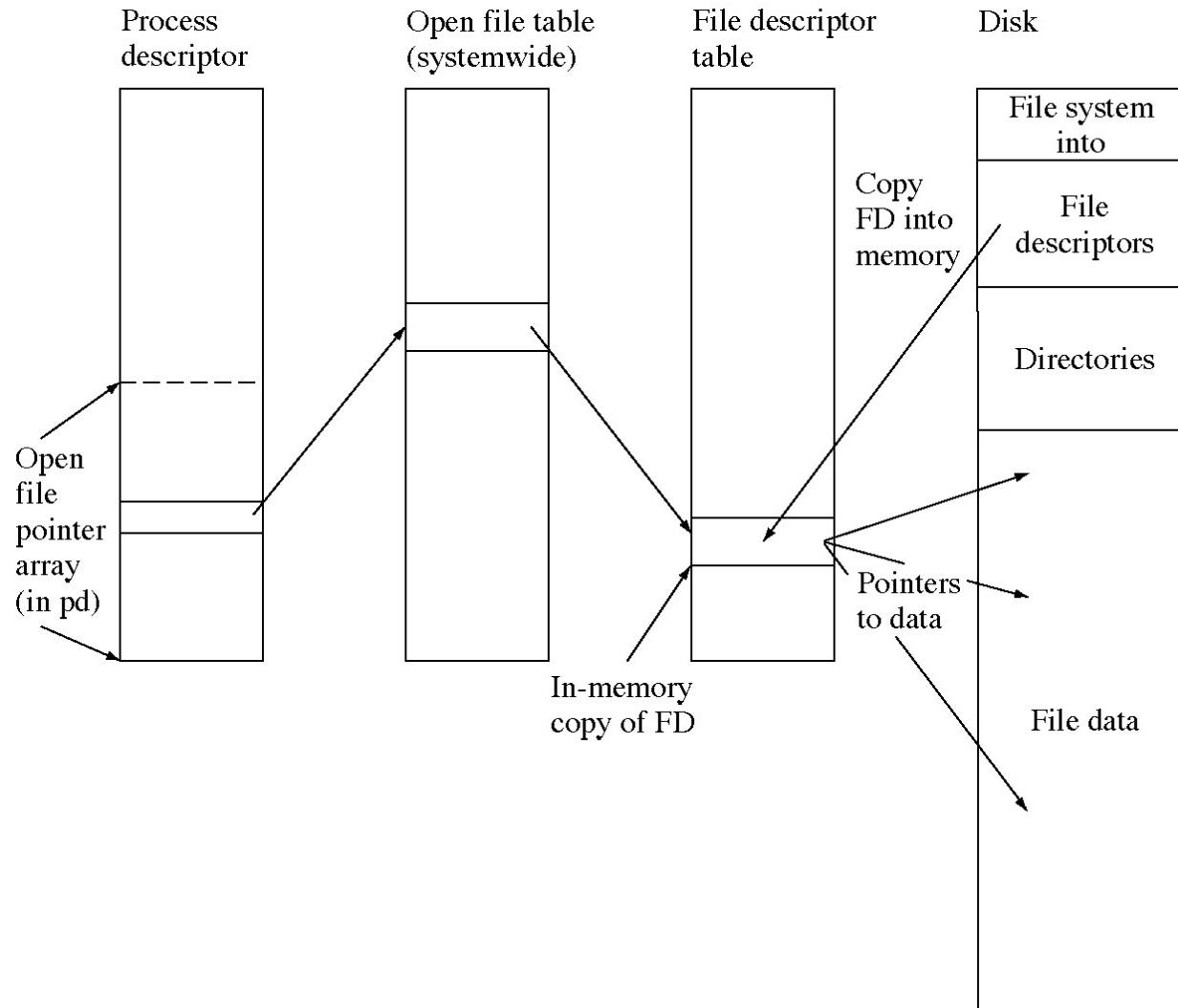
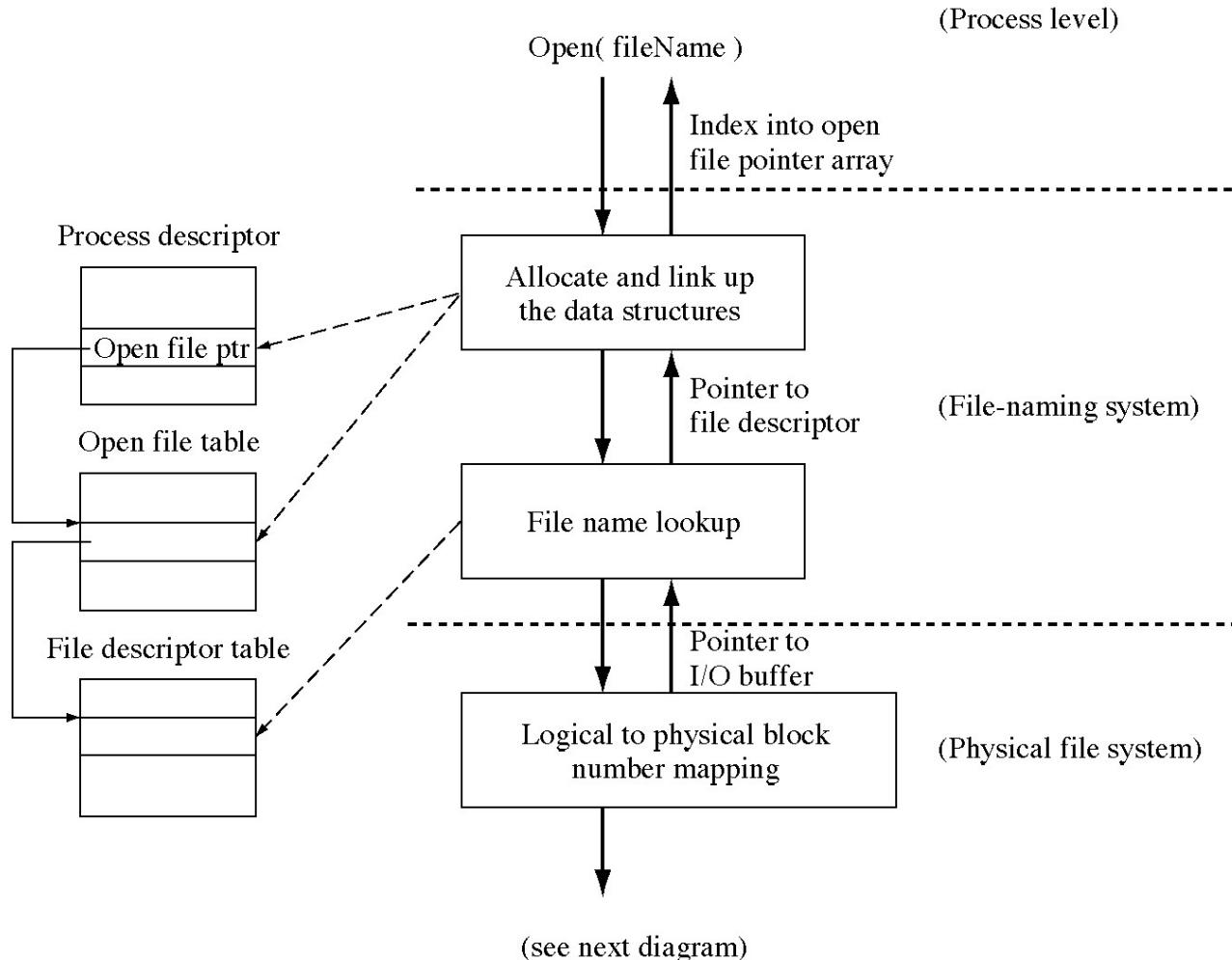


Figure shows the main data structures in a file system and how they are connected to each other. The “open file table” contains a structure for each open file. A file is opened **by** a process and is connected with that process. **Open File Pointer Table (or per process open file table)** - When a file is opened, the open file structure is allocated and a pointer to that structure is placed in a table in the process descriptor of the process opening the file.

- **File Descriptor** – the data structure that represents the file (contains all the meta-data about the file).

File System Control Flow for an Open



- Figure shows the control flow for the **open** system call. **The left side** – data structure affected;
- **right side** – file system modules that are called.
- **The vertical arrows** – connects the modules represent procedure calls and returns.
- **The dashed arrows** – data structures the modules access and modify.

File System Control Flow for a Read

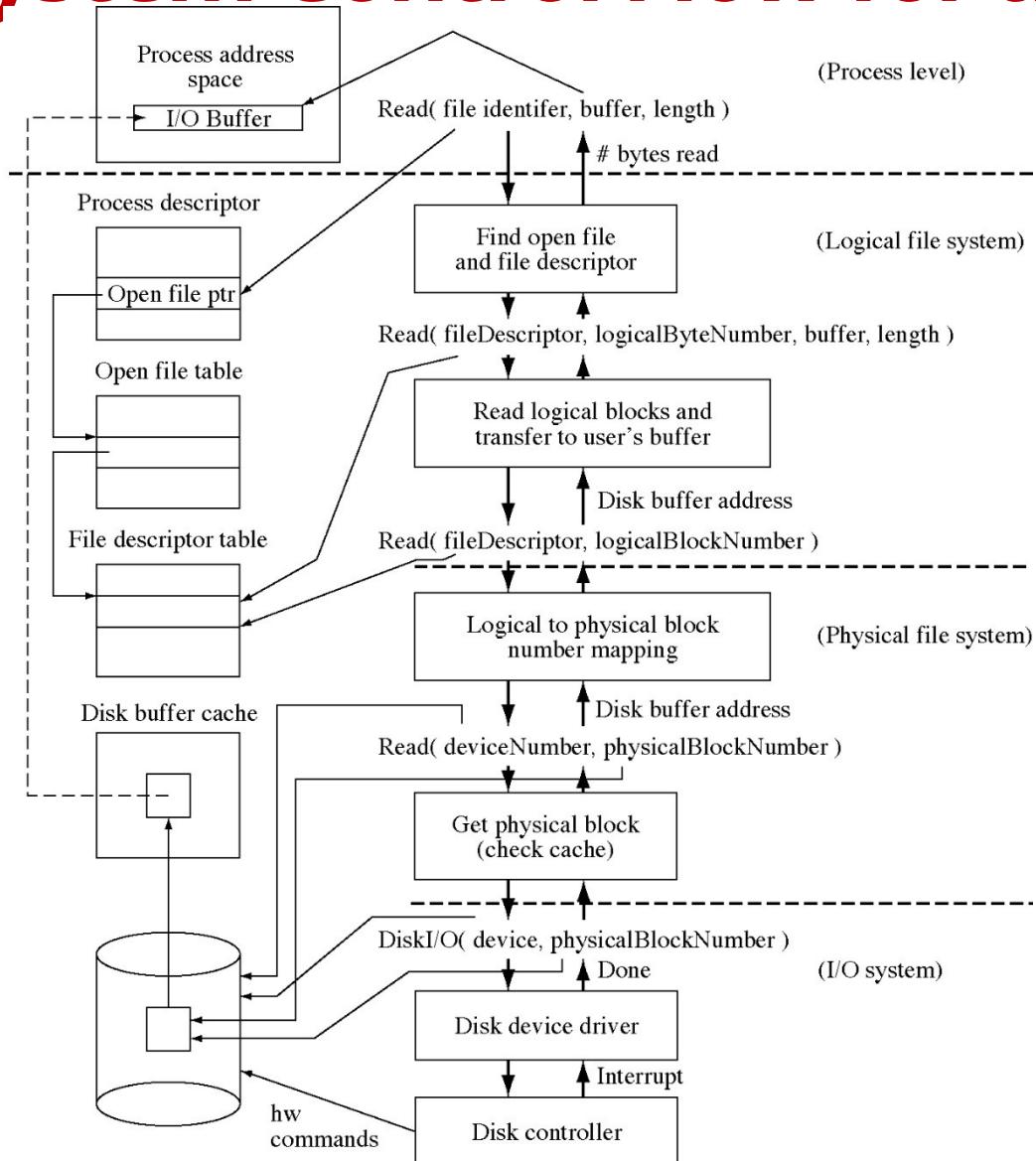


Figure shows the control flow for the **read** system call. **The left side** – the data structures and how they linked together.

The right side – the file system modules that use these data structures.

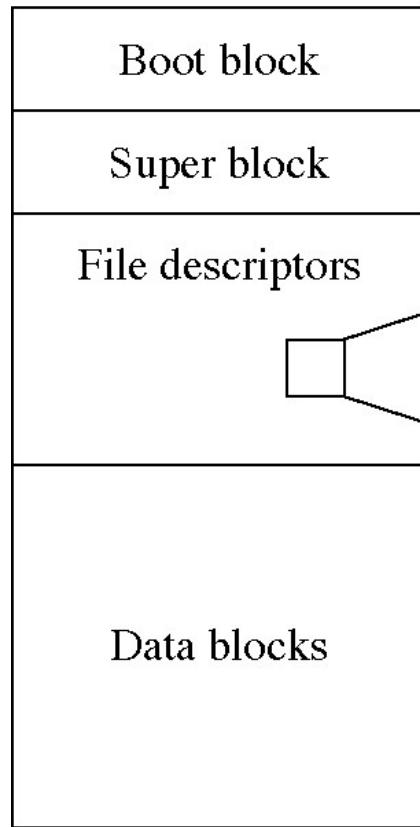
The downward arrows – procedure calls. **The upward arrows** – represents the return from the procedure call.

The lines from right side to the left side – indicate which data structures are accessed and modified **by** the various modules.

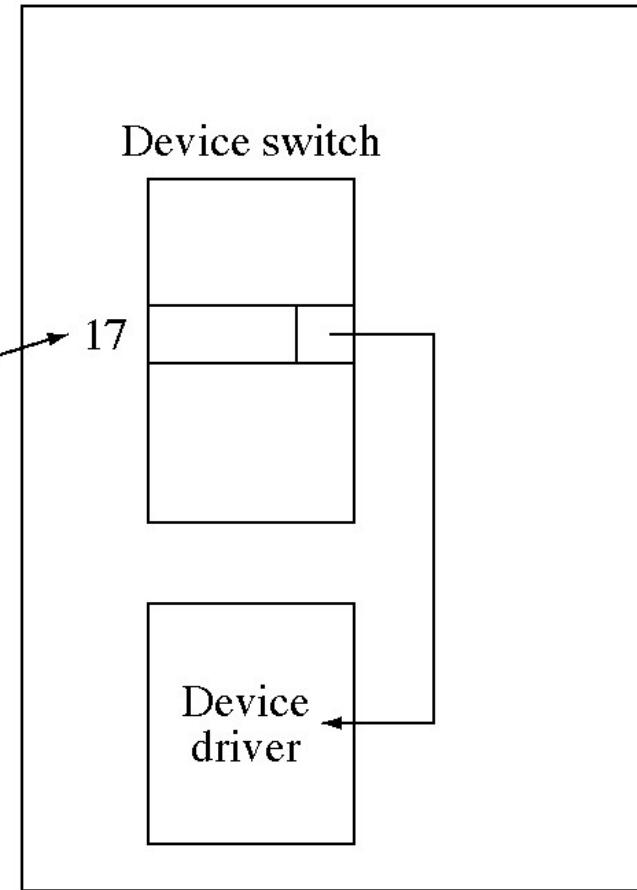
The logical file system – files at the logical level (logical **byte offsets, logical blocks, and logical block numbers**). **The physical file system** - physical blocks. **The I/O system** – devices.

Connecting Files and Devices through the Device Switch

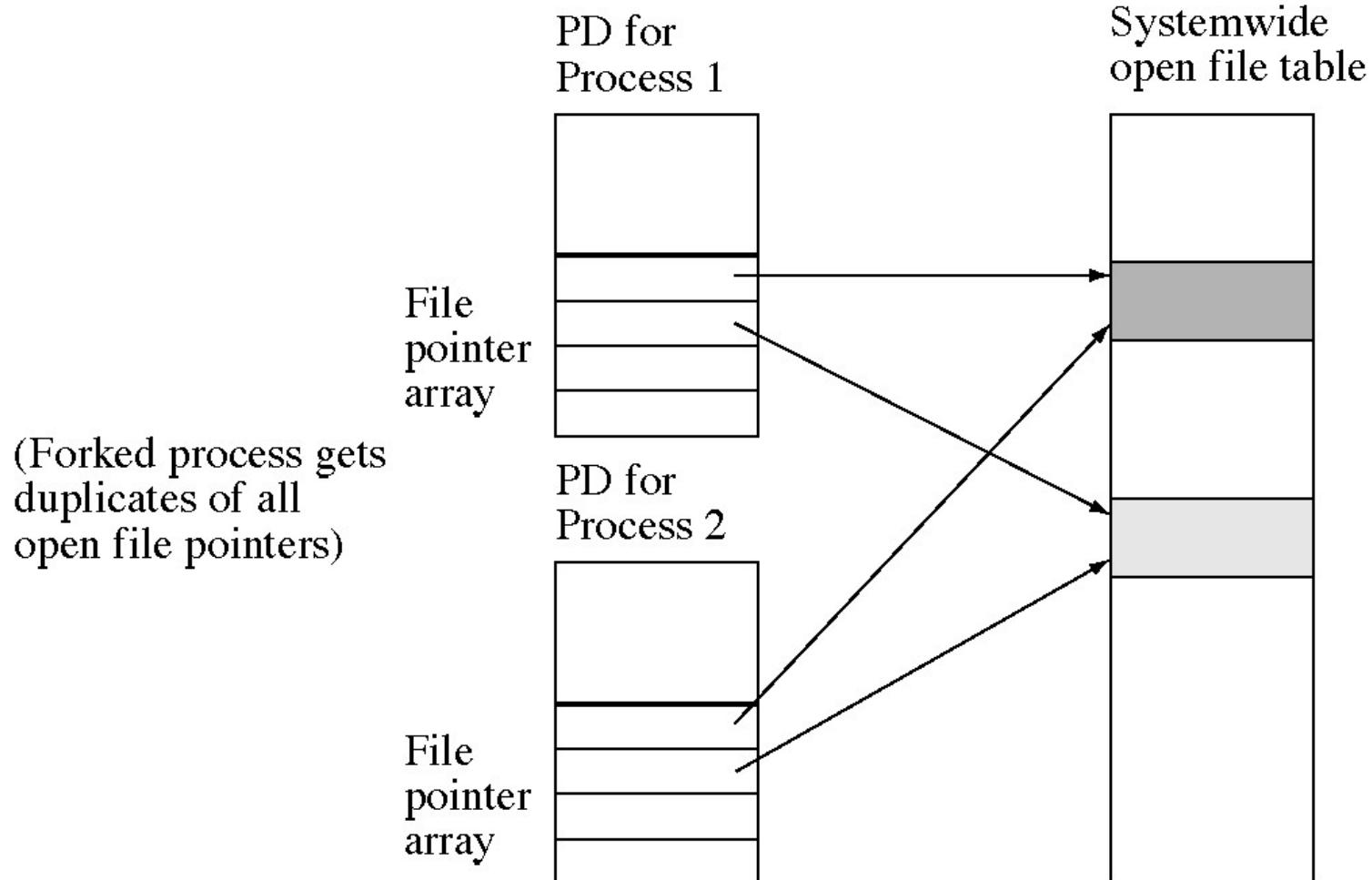
File system



Device switch



The Effect of a Fork on the File System Data Structure



System Call Data Structure Changes

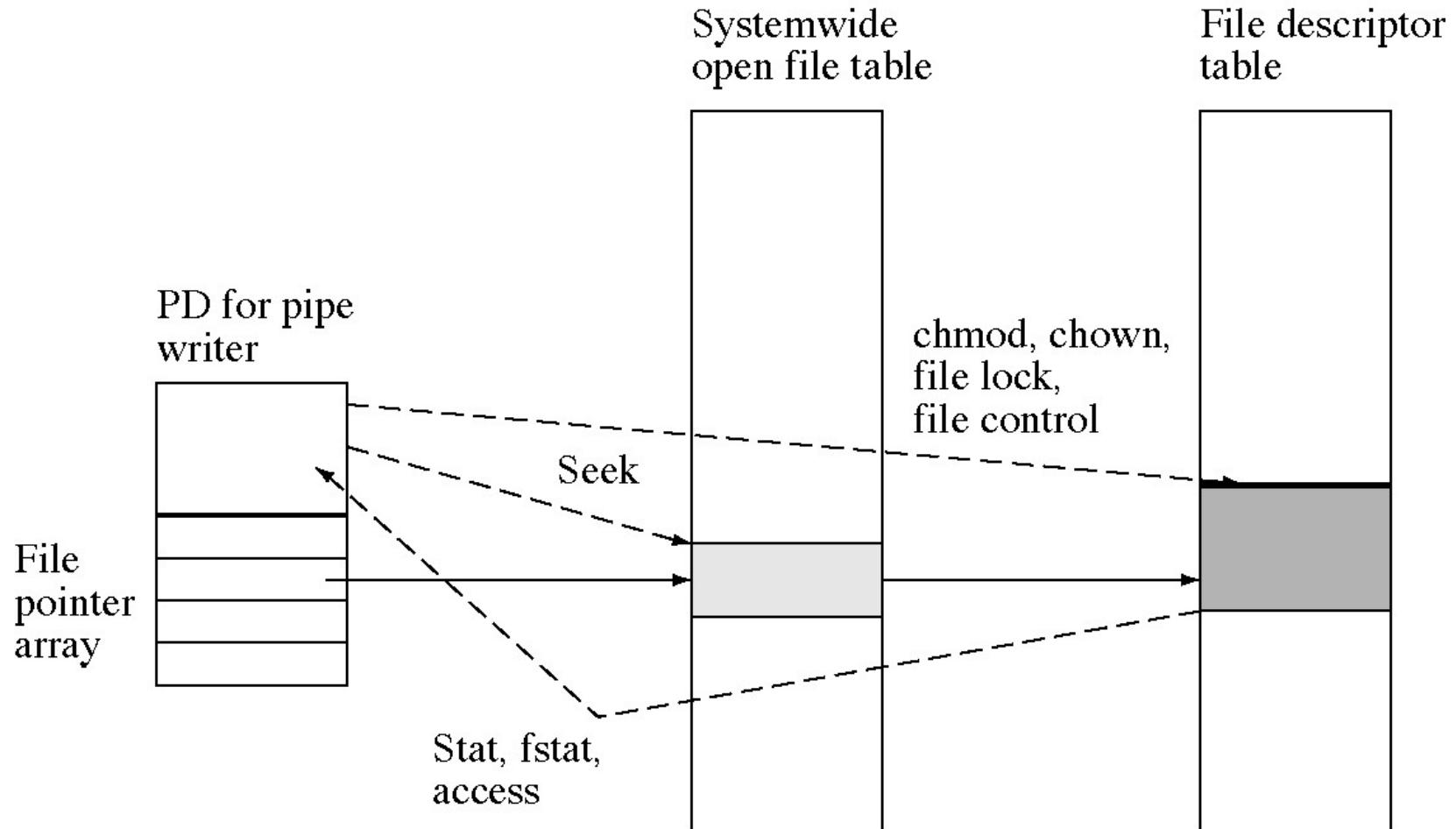


Figure shows the effect of **stat**, **fstat**, **access**, **chmod**, **chown**, and **seek** system calls on the file system data structure.

The stat an fstat – find the file descriptor and return the meta-data in it.

The access system – finds the file descriptor and checks the protection information.

The chmod and chown system call – change the meta data in the file descriptor and arrange for it to be written back out to the disk.

The seek system call – changes the file pointer in the open file structure.

The file control and file lock system calls - change the information in the file descriptor.

Stat command

stat is a command which gives information about the file and filesystem. Stat command gives information such as the size of the file, access permissions and the user ID and group ID, birth time access time of the file.

syntax of using stat command:

stat --options filenames

Stat command

-L, --dereference follow links

-f, --file-system display file system status
instead of file status

-c --format=FORMAT use the specified
FORMAT instead of the default; output a
newline after each use of FORMAT

Information we get from stat

- **File:** The file name, If the provided file is a symlink, then the name will be different.
- **Size:** The size of a given file in Bytes.
- **Blocks:** Total number of allocated blocks to the file on hard disk.
- **File type:** Regular files, special files, directories, or symbolic links.
- **Device:** Device number in hexadecimal format.
- **Inode:** Inode number of the file.
- **Links:** Number of hard links of the file.
- **Access:** The last time at which the file was accessed.
- **Modify:** The last time at which file was modified.
- **Change:** The last time the at which file's attribute or content was changed.
- **Birth:** The time at which the file was created.

du Command

du command displays the number of blocks used for files. If the *File* parameter specified is actually a directory, all files within the directory are reported on. If no *File* parameter is provided, the **du** command uses the files in the current directory.

Notes:

- Files with multiple links are counted and written for only one entry.
- Block counts are based only on file size

du Command

Flags

- a Displays disk usage for each file specified, or displays the individual disk usage for each file in a directory.
- k Calculates the block count in 1024-byte units rather than the default 512-byte units.
- l Allocates blocks evenly among the links for files with multiple links. By default, a file with two or more links is counted only once.
- s Displays the total disk usage for all specified files, or displays the total disk usage for all files in a directory.

syntax of du command

du [OPTION]... [FILE]...

df Command

- df command (short for disk filesystem) is used to show disk utilization for a Linux system.
- **Display information of device name, total blocks, total disk space, used disk space, available disk space and mount points on a file system**
- Display all the file system, use -a option. df -a
- Use -h option to display size in power of 1024 /home/HCL df -h
- Use -H option to display sizes in power of 1000 /home/HCL df -H
- Use -T option to display file type /home/HCL df -T

df Command

To display information about /boot file system,

```
df /boot
```

We can display information in various block format. For instance see 1M-byte blocks, run:

```
df -m
```

```
df -m /boot
```

Show statistics about the number of free and used inodes

Pass the -i as follows to display index node (inodes) info:

```
df -i
```

```
df -i -H
```

```
df -i -H -T
```

Inodes

Each file is represented by a structure, called an inode.

Each inode contains the description of the file: file type, access rights, owners, timestamps, size, pointers to data blocks.

The addresses of data blocks allocated to a file are stored in its inode. When a user requests an I/O operation on the file, the kernel code converts the current offset to a block number, uses this number as an index in the block addresses table and reads or writes the physical block.

Directories

- Directories are structured in a hierarchical tree. Each directory can contain files and subdirectories.
- Directories are implemented as a special type of files. Actually, a directory is a file containing a list of entries. Each entry contains an inode number and a file name. When a process uses a pathname, the kernel code searches in the directories to find the corresponding inode number. After the name has been converted to an inode number, the inode is loaded into memory and is used by subsequent requests.

Links

The inode contains a field containing the number associated with the file.

Adding a link simply consists in creating a directory entry, where the inode number points to the inode, and in incrementing the links count in the inode.

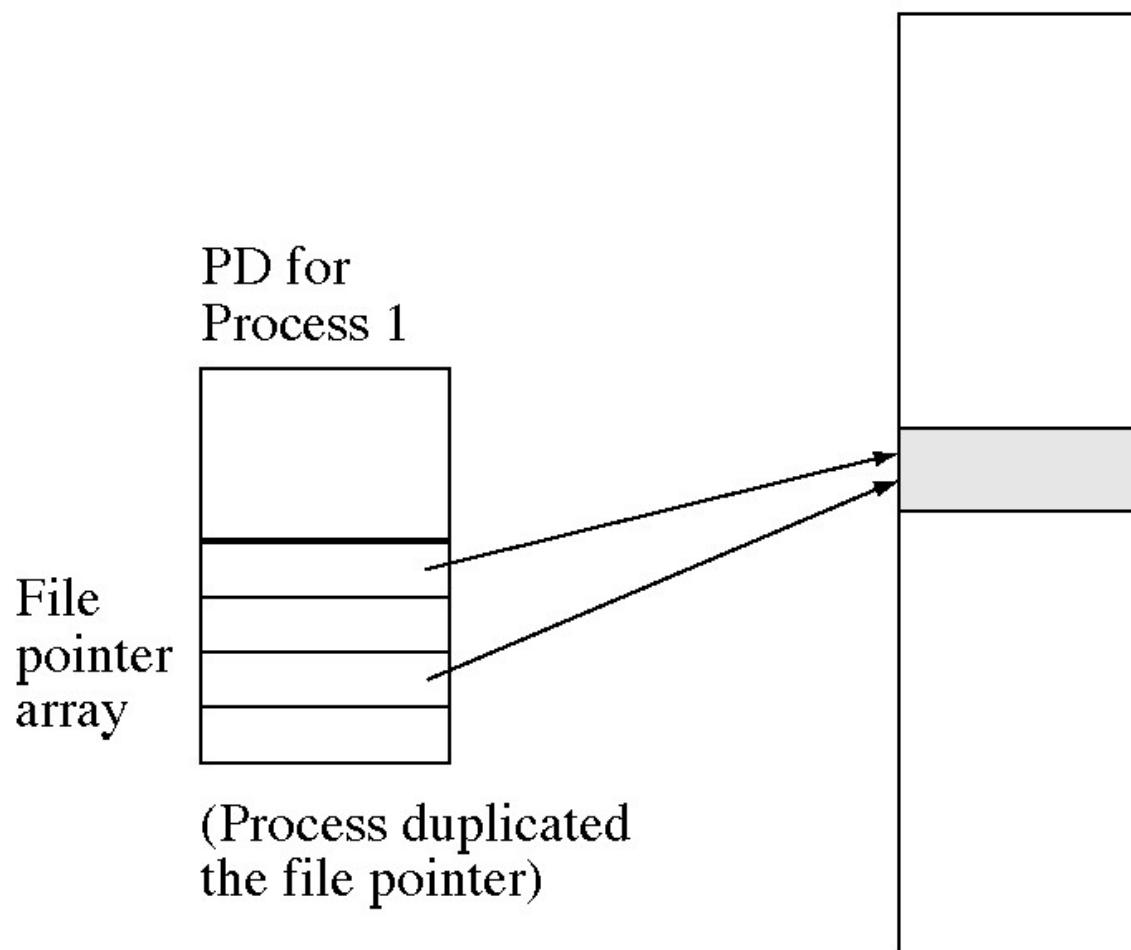
When a link is deleted, i.e. when one uses the `rm` command to remove a filename, the kernel decrements the links count and deallocates the inode if this count becomes zero.

Device special files

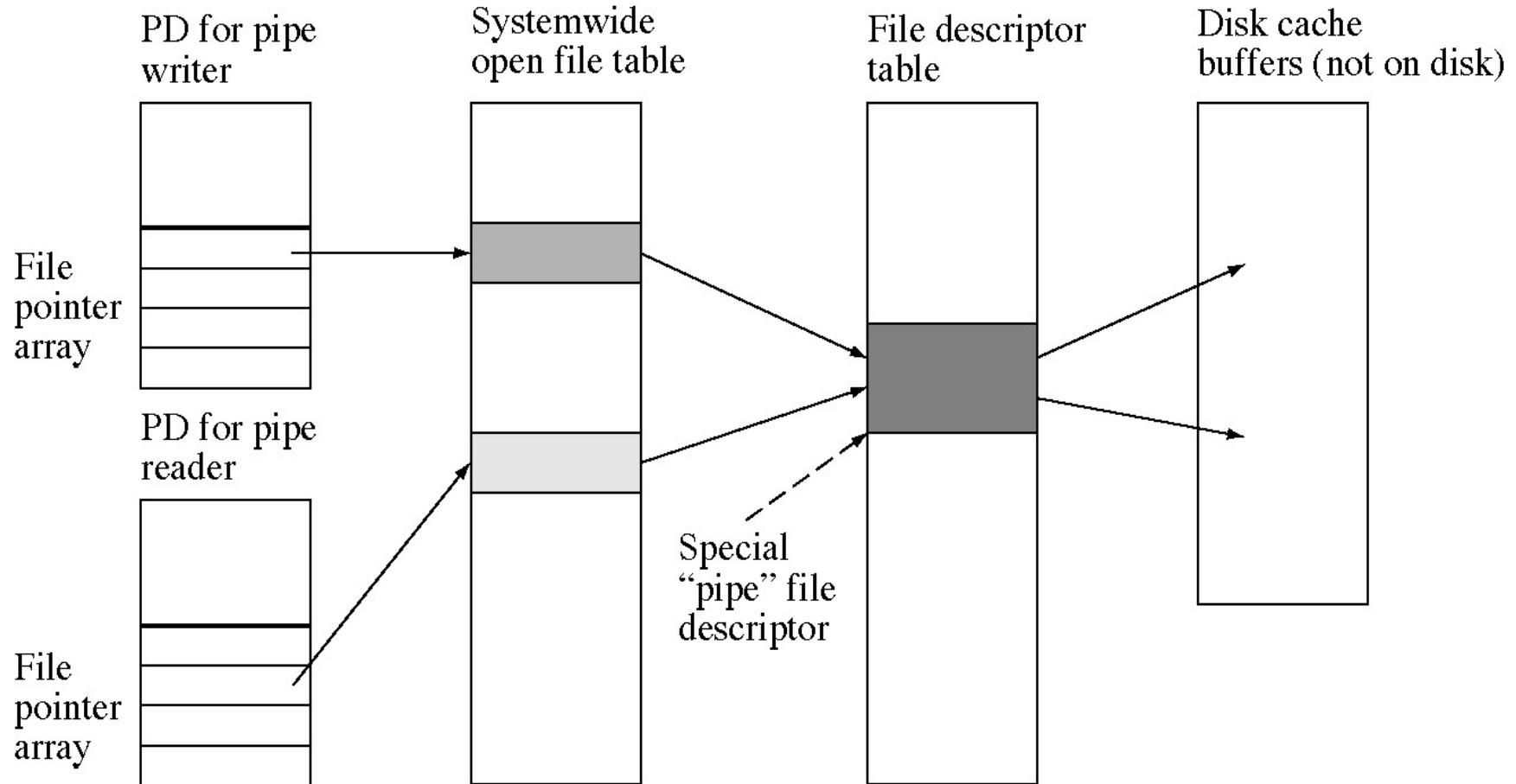
- Devices can be accessed via special files. A device special file does not use any space on the filesystem. It is only an access point to the device driver.
- Two types of special files exist: character and block special files. The former allows I/O operations in character mode while the later requires data to be written in block mode via the buffer cache functions.
- When an I/O request is made on a special file, it is forwarded to a device driver. A special file is referenced by a major number, which identifies the device type, and a minor number, which identifies the unit.

Effects of Duplicate System Call

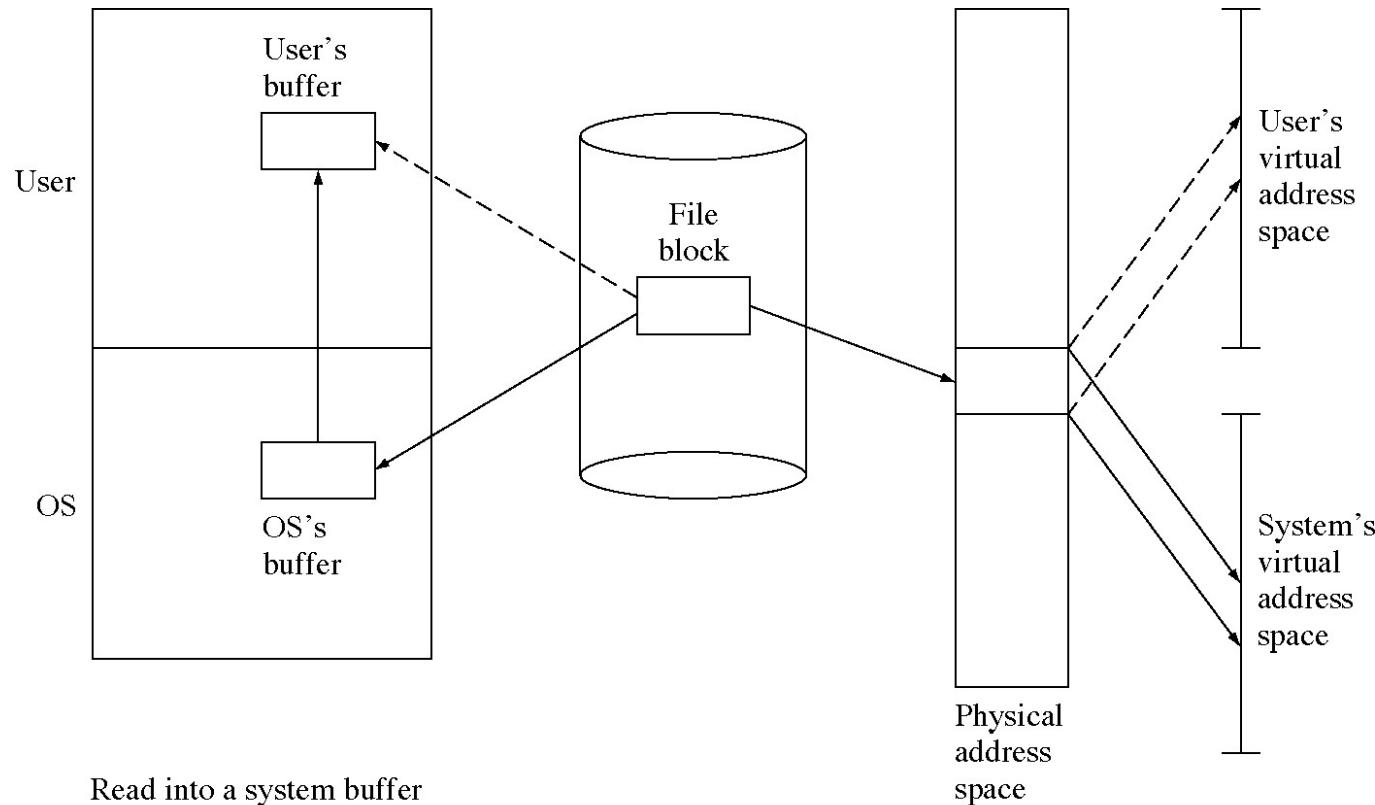
Systemwide
open file table



Effects of Pipe System Call



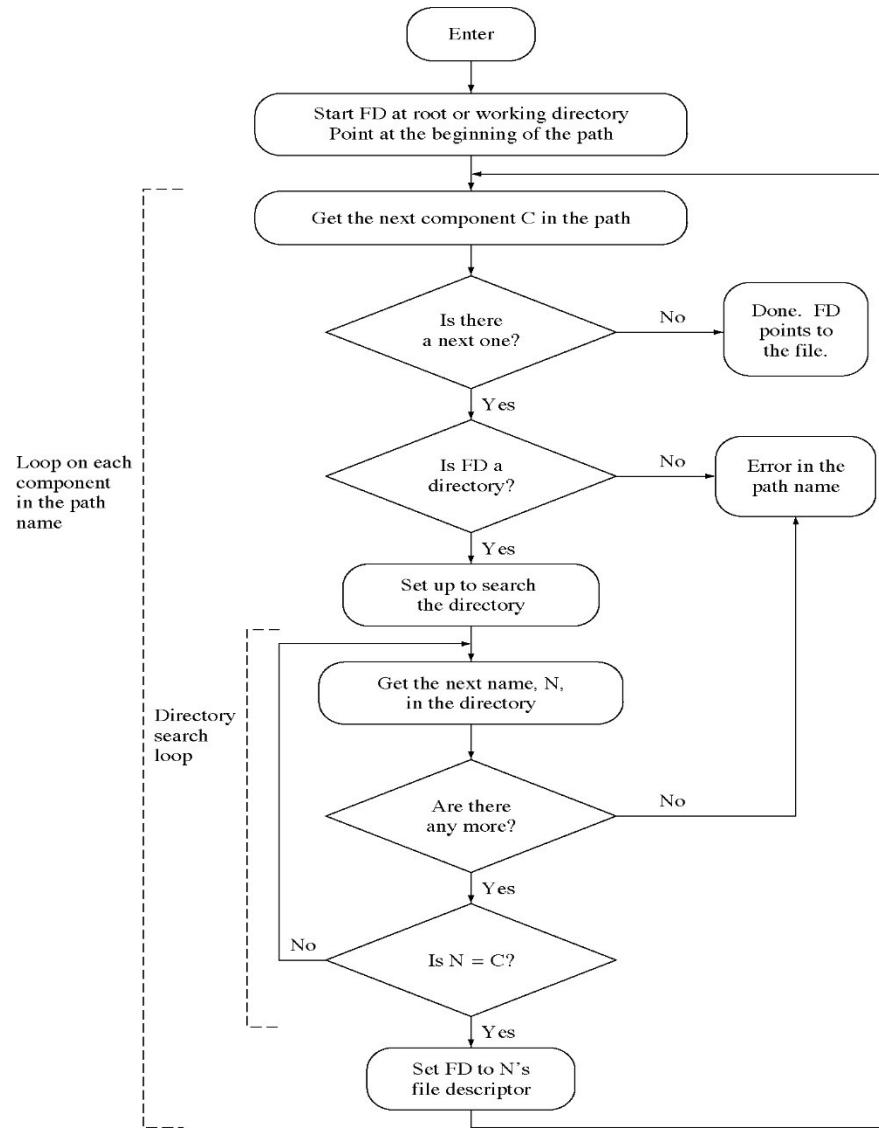
Avoiding Copying I/O Data



Read into a system buffer and then copy to the user's buffer, or read directly into the user's buffer.

Copy into a page in the system's virtual address space and then map it into the user's virtual address space.

Path Name Lookup Algorithm



UNIX FILE SYSTEM

- Developed by

Bell Telephone

Laboratories for the

Digital Equipment

Corporation PDP line

of minicomputers

- Most of the UNIX interface is an official standard

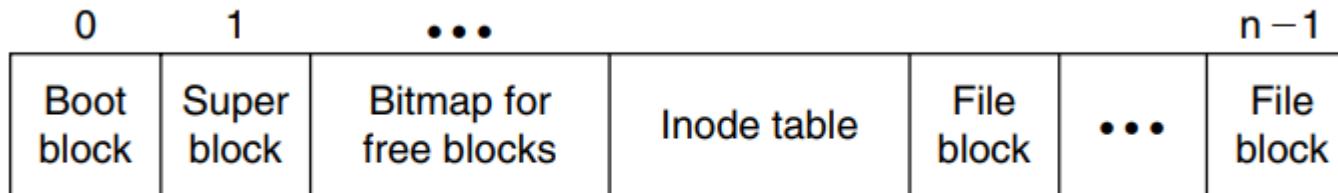
Procedure	Brief Description
OPEN (<i>name, flags, mode</i>)	Open file <i>name</i> . If the file doesn't exist and <i>flags</i> is set, create file with permissions <i>mode</i> . Set the file cursor to 0. Returns a file descriptor.
READ (<i>fd, buf, n</i>)	Read <i>n</i> bytes from the file at the current cursor and increase the cursor by the number of bytes read.
WRITE (<i>fd, buf, n</i>)	Write <i>n</i> bytes at the current cursor and increase the cursor by the bytes written.
SEEK (<i>fd, offset, whence</i>)	Set the cursor to <i>offset</i> bytes from beginning, end, or current position.
CLOSE (<i>fd</i>)	Delete file descriptor. If this is the last reference to the file, delete the file.
FSYNC (<i>fd</i>)	Make all changes to the file durable.
STAT (<i>name</i>)	Read metadata of file.
CHMOD, CHOWN, etc.	Various procedures to set specific metadata.
RENAME (<i>from_name, to_name</i>)	Change name from <i>from_name</i> to <i>to_name</i>
LINK (<i>name, link_name</i>)	Create a hard link <i>link_name</i> to the file <i>name</i> .
UNLINK (<i>name</i>)	Remove <i>name</i> from its directory. If <i>name</i> is the last name for a file, remove file.
SYMLINK (<i>name, link_name</i>)	Create a symbolic name <i>link_name</i> for the file <i>name</i> .
MKDIR (<i>name</i>)	Create a new directory named <i>name</i> .
CHDIR (<i>name</i>)	Change current working directory to <i>name</i> .
CHROOT (<i>name</i>)	Change the default root directory to <i>name</i> .
MOUNT (<i>name, device</i>)	Graft the file system on <i>device</i> onto the name space at <i>name</i> .
UNMOUNT (<i>name</i>)	Unmount the file system at <i>name</i> .

The Naming Layers of the UNIX File System

Layer	Purpose	
Symbolic link layer	Integrate multiple file systems with symbolic links.	↑ user-oriented names
Absolute path name layer	Provide a root for the naming hierarchies.	
Path name layer	Organize files into naming hierarchies.	↓
File name layer	Provide human-oriented names for files.	machine-user interface
Inode number layer	Provide machine-oriented names for files.	↑ machine-oriented names
File layer	Organize blocks into files.	
Block layer	Identify disk blocks.	↓

Disk layout for a simple file system

- Bottom layer the UNIX file system names some physical device (magnetic disk, flash disk, or magnetic tape) that can store data durably.
- The storage is divided into fixed-size units, called **blocks**.
- In **version 6**, the UNIX file system used **512-byte blocks**, but modern UNIX file systems often use **8-kilobyte blocks**.



- A **file** is a linear array of bytes of arbitrary length.
- The file system needs to record in some way which blocks belong to each file.
- To support this requirement, the UNIX file system creates an index node (**inode**).
- The inode for a file is a context in which the various blocks of the file are named by **integer block numbers**.
- To implement the **naming layer** is to employ a table that directly contains all inodes, indexed by **inode number**.

- **Directory** - The most visible component of the durable object naming scheme

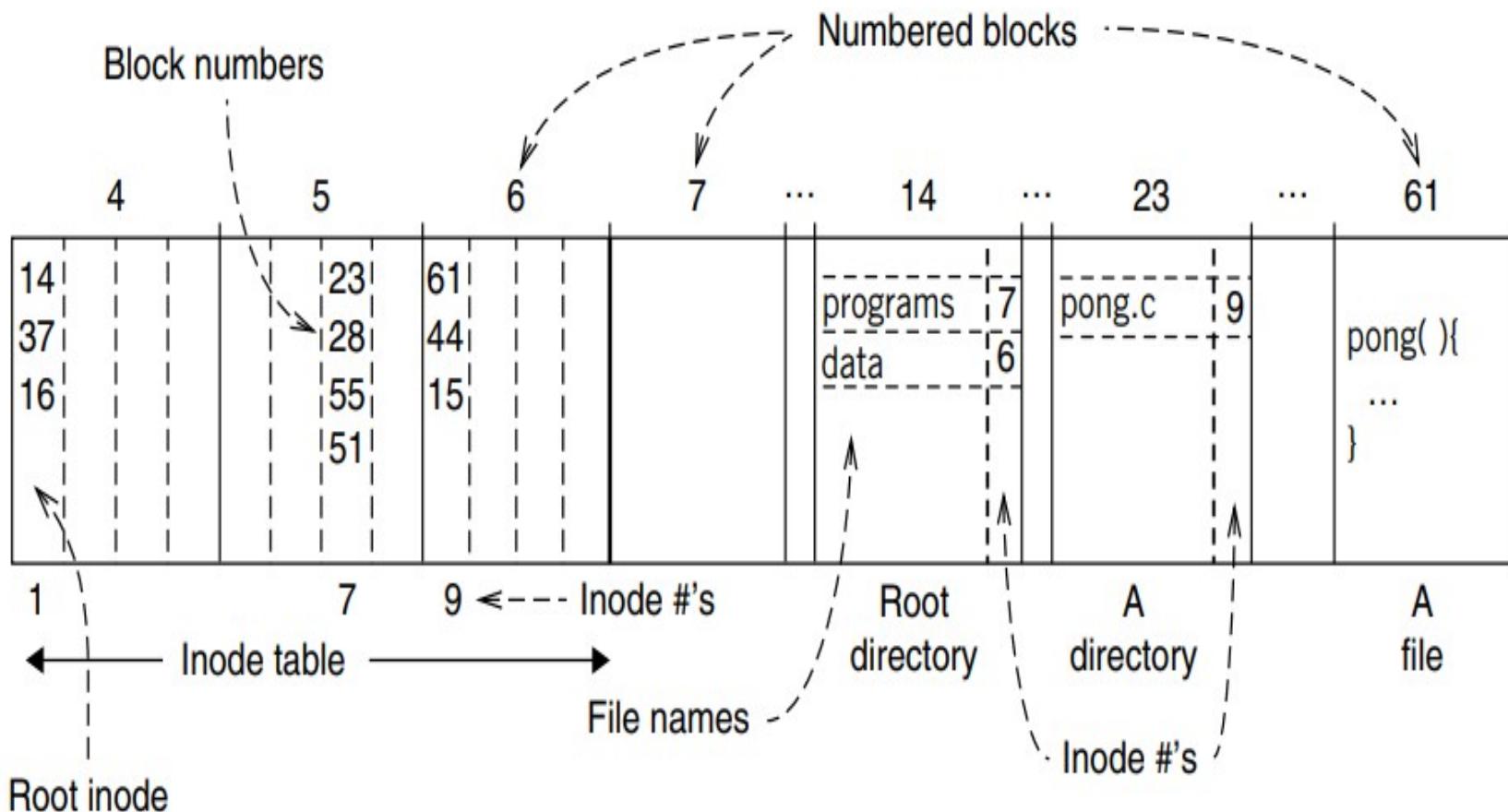
- Figure is a directory with two file names (“program” and “paper”), which are mapped to inode numbers 10 and 12, respectively
- In **Version 6**, the maximum length of a name is 14 bytes, and the entries in the table have a fixed length of 16 bytes

```

• \b procedure LOOKUP (character string filename, integer dir) \b returns integer
  block instance b
  inode instance i \leftarrow INODE_NUMBER_TO_INODE (dir)
  \b if i.type \neq DIRECTORY \b then \b return \b FAILURE
  \b for offset \b from 0 \b to i.size - 1 \b do
    b \leftarrow INODE_NUMBER_TO_BLOCK (offset, dir)
    \b if STRING_MATCH (filename, b) \b then
      \b return \b INODE_NUMBER (filename, b)
    offset \leftarrow offset + BLOCKSIZE
  \b return \b FAILURE

```

Example disk layout for a UNIX file system



The UNIX Naming Layers, with Details of the Naming Scheme of each Layer

Layer	Names	Values	Context	Name-Mapping Algorithm	
Symbolic link	Path names	Path names	The directory hierarchy	PATHNAME_TO_GENERAL_PATH	↑
Absolute path name	Absolute path names	Inode numbers	The root directory	GENERALPATH_TO_INODE_NUMBER	user-oriented names
Path name	Relative path names	Inode numbers	The working directory	PATH_TO_INODE_NUMBER	↓
File name	File names	Inode numbers	A directory	NAME_TO_INODE_NUMBER	machine-user interface
Inode number	Inode numbers	Inodes	The inode table	INODE_NUMBER_TO_INODE	↑
File	Index numbers	Block numbers	An inode	INDEX_TO_BLOCK_NUMBER	machine-oriented names
Block	Block numbers	Blocks	The disk drive	BLOCK_NUMBER_TO_BLOCK	↓

Implementing the File System API

- **File descriptor (fd)** - **READ, WRITE, and CLOSE** use to name the file

```
procedure OPEN (character string filename, flags, mode)
  inode_number ← PATH_TO_INODE_NUMBER (filename, wd)
  if inode_number = FAILURE and flags = O_CREATE then          // Create the file?
    inode_number ← CREATE (filename, mode)                  // Yes, create it.
  if inode_number = FAILURE then
    return FAILURE
  inode ← INODE_NUMBER_TO_INODE (inode_number)
  if PERMITTED (inode, flags) then // Does this user have the required permissions
    file_index ← INSERT (file_table, inode_number)
    fd ← FIND_UNUSED_ENTRY (fd_table) // Find entry in file descriptor table
    fd_table[fd] ← file_index          // Record file index for file descriptor
    return fd                          // Return fd
  else return FAILURE                  // No, return a failure
```

- **READ** is implemented as follows:

```
procedure READ (fd, character array reference buf, n)
  file_index ← fd_table[fd]
  cursor ← file_table[file_index].cursor
  inode ← INODE_NUMBER_TO_INODE (file_table[file_index].inode_number)
  m = MINIMUM (inode.size – cursor, n)
  atime of inode ← NOW ()
  if m = 0 then return END_OF_FILE
  for i from 0 to m – 1 do {
    b ← INODE_NUMBER_TO_BLOCK (i, inode_number)
    COPY (b, buf, MINIMUM (m – i, BLOCKSIZE))
    i ← i + MINIMUM (m – i, BLOCKSIZE)
  }
  file_table[file_index].cursor ← cursor + m
  return m
```

REFERENCES

- **Data, S. I. Locality and The Fast File System.**

RECAP

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• I/O System Hardware<ul style="list-style-type: none">◦ Device Controllers◦ Types of devices : Terminal Devices, Communication Devices and Disk Devices• Data transfer schemes• Programmed I/O, Interrupt Driven I/O, DMA	<p>T1 (14.1 – 14.4)</p> <p>R6 (7.3-7.5)</p>

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none"> • I/O System Software <ul style="list-style-type: none"> ◦ Device Drivers ◦ Device Driver Interfaces <ul style="list-style-type: none"> ▪ The Block Device Interfaces ▪ The Character Device Interfaces ◦ Device Numbers 	T1 (15.1, 15.4)
<ul style="list-style-type: none"> • Unification of Files and I/O Devices • Mechanisms for improving I/O performance <ul style="list-style-type: none"> ◦ Prefetching ◦ Caching ◦ Buffering 	T1 (15.5)
	Slide 56

- Levels in the file system
- File storage on hard disks, CD ROMs and Flash ROMs
- File Vs Databases



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN SESSION 13

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Balamurali Shankar
WILP & Department of CS & IS



BITS Pilani
Pilani Campus

Operating Systems – Software Design



Last Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• File Access Methods – Effect of Locality• File System Implementation• Example (UNIX or Linux File System/ Windows NTFS/Mobile file System/ Flash Memory File System)	T2 and Class Notes

Software Design

Today's topic

- **The General Concept of Abstraction**
- **Types of Abstractions in Software Engineering**
 - Data Abstraction
 - Control Abstraction
- **Design Styles in Programming and Programming Languages: Procedural Design vs. OO Design**
 - Object Oriented Abstraction
 - The Evolution of the Object Model
 - Foundations of the Object Model
 - Elements of the Object Model
 - Applying the Object Model
 - Abstractions in OOP

The General Concept of Abstraction

- **Abstraction** - a method to' hide irrelevant details and represent only the essential features
- *An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer*
- It allows managing complex systems
- Two types of abstraction:
 - **Data abstraction** - hiding the details about the data
 - **Control abstraction** - hiding the implementation details

```
#include <iostream>
using namespace std;
class implementAbstraction
{
    private:
        int a, b;
public:
    // method to set values of
    // private members
    void set(int x, int y)
    {
        a = x;
        b = y;
    }
    void display()
    {
        cout<<"a = " <<a << endl;
        cout<<"b = " << b << endl;
    }
};

int main()
{
    implementAbstraction obj;
    obj.set(10, 20);
    obj.display();
    return 0;
}
```

```
// Abstract class
abstract class Vehicle {
    // Abstract method (does not have a body)
    public abstract void VehicleSound();
    // Regular method
    public void honk() { System.out.println("honk honk"); }
}

// Subclass (inherit from Vehicle)
class Car extends Vehicle {
    public void VehicleSound()
    {
        // The body of VehicleSound() is provided here
        System.out.println("kon kon");
    }
}

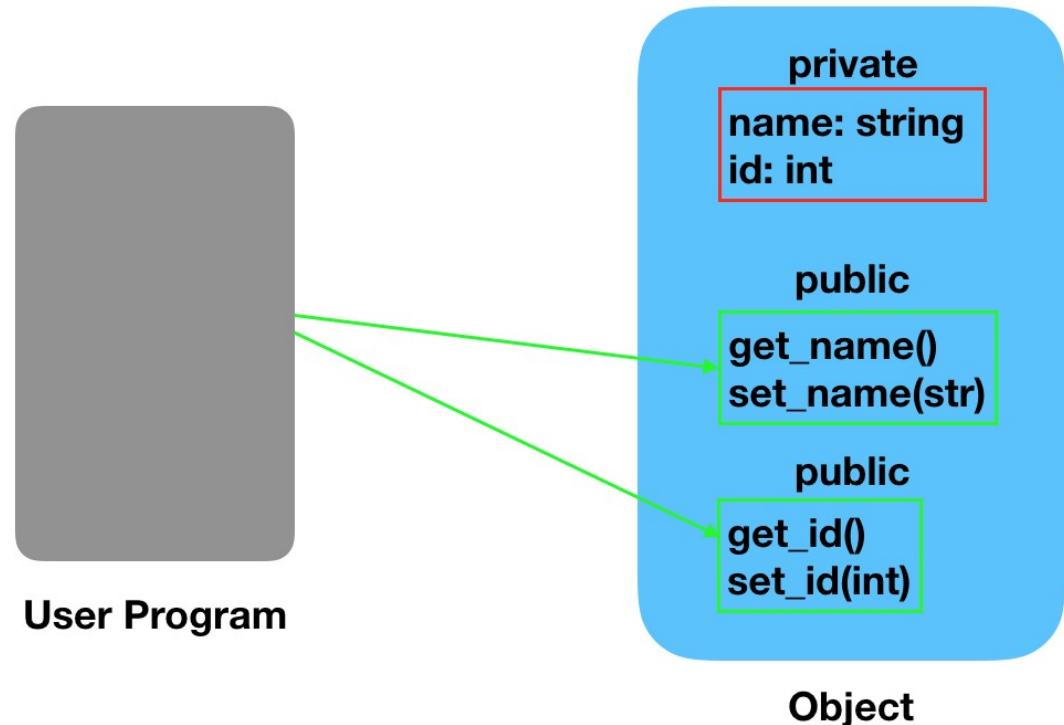
class Main {
    public static void main(String[] args)
    {
        // Create a Car object
        Car myCar = new Car();
        myCar.VehicleSound();
        myCar.honk();
    }
}
```

Advantages of Abstraction

1. It reduces the complexity of viewing the things.
2. Avoids code duplication and increases reusability.
3. Helps to increase the security of an application or program as only important details are provided to the user.

Data Abstraction

Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

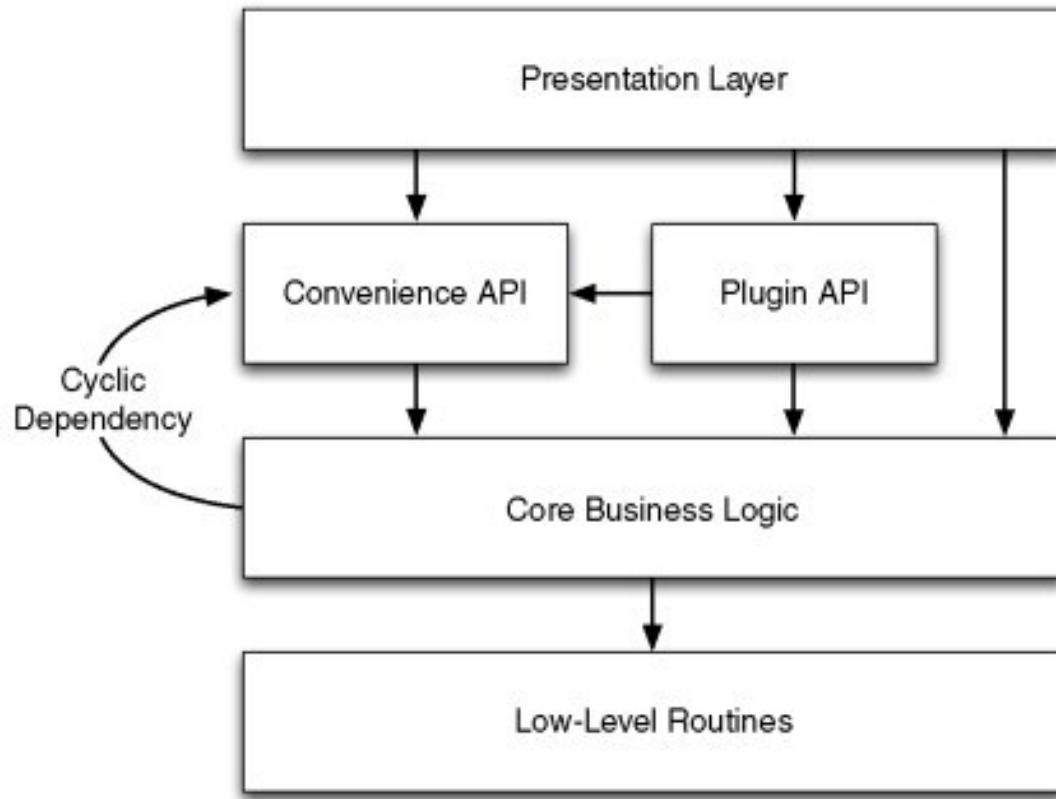


Advantages of Data Abstraction

- Helps the user to avoid writing the low level code
- Avoids code duplication and increases reusability.
- Can change internal implementation of class independently without affecting the user.
- Helps to increase security of an application or program as only important details are provided to the user.

Control Abstraction

- This refers to the software part of abstraction wherein the program is simplified, and unnecessary execution details are removed.
- The **advantage** of control abstraction is that it makes code a lot cleaner and **more secure**.
- It is used to build new functionalities and combines control statements into a single unit
- **Higher-order functions, closures, and lambdas** are few preconditions for control abstraction.
- Highlights more on how a particular functionality can be achieved rather than describing each detail.
- Forms the **main unit of structured programming**.



Procedural Programming

- It can be defined as a programming model which is derived from structured programming, based upon the **concept of calling procedure**.
- Procedures as (routines, subroutines or functions), simply consist of a series of computational steps to be carried out.
- During a program's execution, any given procedure might be called at any point, including by other procedures or itself.

Procedural Oriented Programming	Object-Oriented Programming
The program is divided into small parts called functions .	The program is divided into small parts called objects .
Top-down approach	Bottom-up approach
Adding new data and functions is not easy	Adding new data and function is easy
Less secure	More secure
There is no concept of data hiding and inheritance	The concept of data hiding and inheritance is used.
The function is more important than the data.	Data is more important than function
Examples: C, FORTRAN, Pascal, Basic, etc.	Examples: C++, Java, Python, C#, etc.

Object Oriented Abstraction

- In Object-oriented programming, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user.
- It allows users not to get overwhelmed by the hidden logic that makes the users complex logic.
- Thus, OOPs abstracts hidden logic so that the user is not required to understand or even think about it.

The Evolution of the Object Model

- First-Generation Languages (1954-1958)

FORTRANI	Mathematical expressions
ALGOL 58	Mathematical expressions
Flowmatic	Mathematical expressions
IPL V	Mathematical expressions

- Second-Generation Languages (1959~1961)

FORTRANII	Subroutines, separate compilation
ALGOL 60	Block structure, data types
COBOL	Data description, file handling
Lisp	List processing, pointers, garbage collection

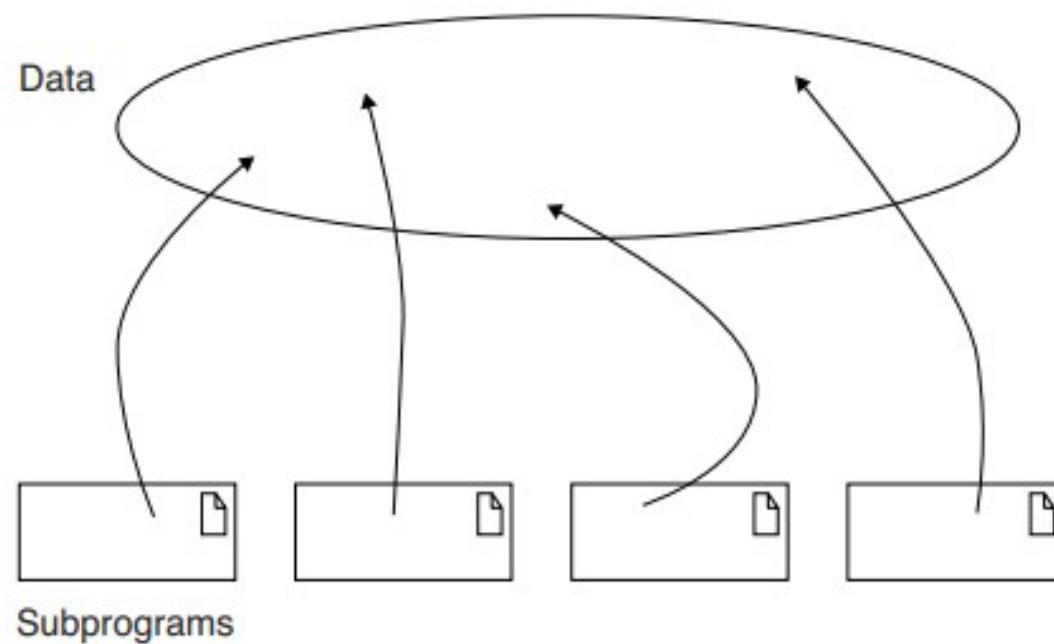
- Third-Generation Languages (1962-1970)

PL/1	FORTRAN + ALGOL + COBOL
ALGOL 68	Rigorous successor to ALGOL 60
Pascal	Simple successor to ALGOL 60
Simula	Classes, data abstraction

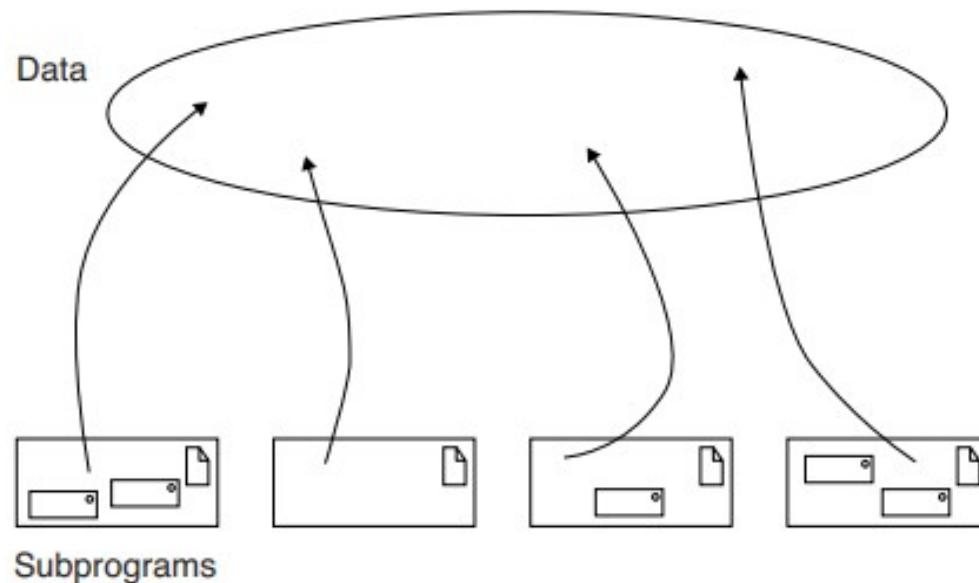
- The Generation Gap (1970-1980)

Many different languages were invented, but few endured [2].

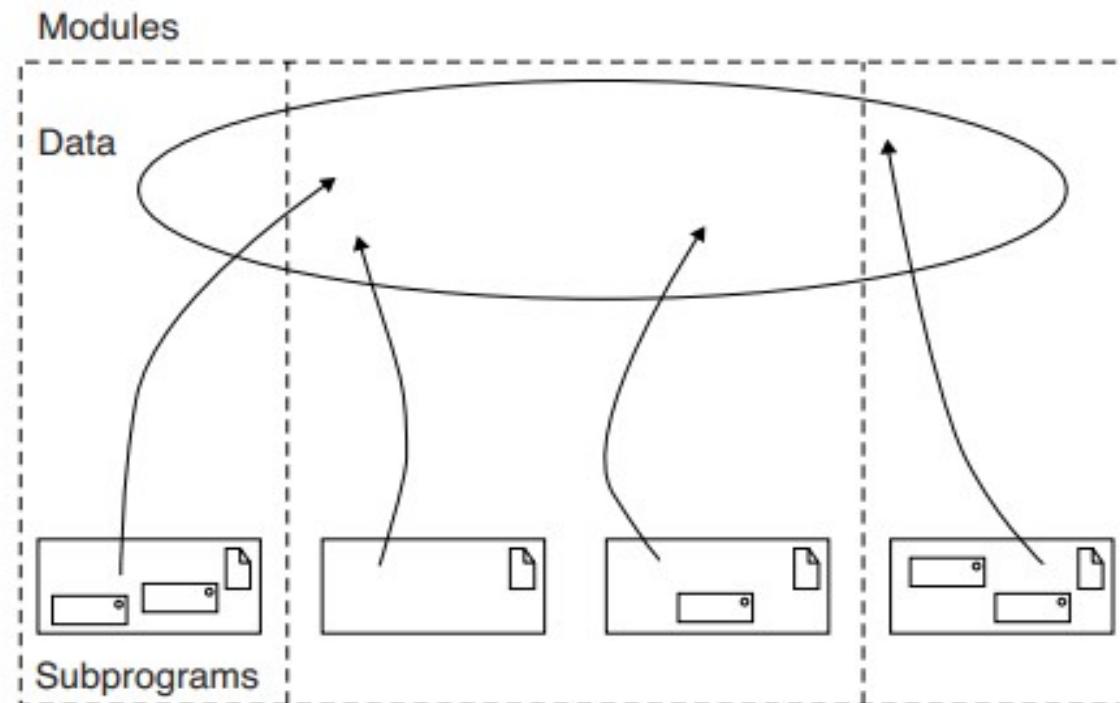
The Topology of First and Early Second-Generation Programming Languages



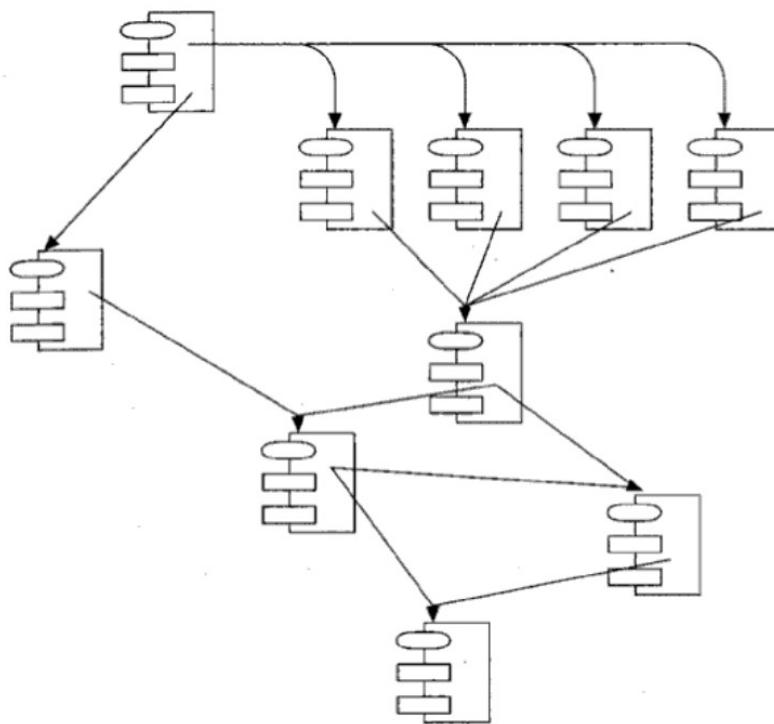
The Topology of Late Second and Early Third-Generation Programming Languages



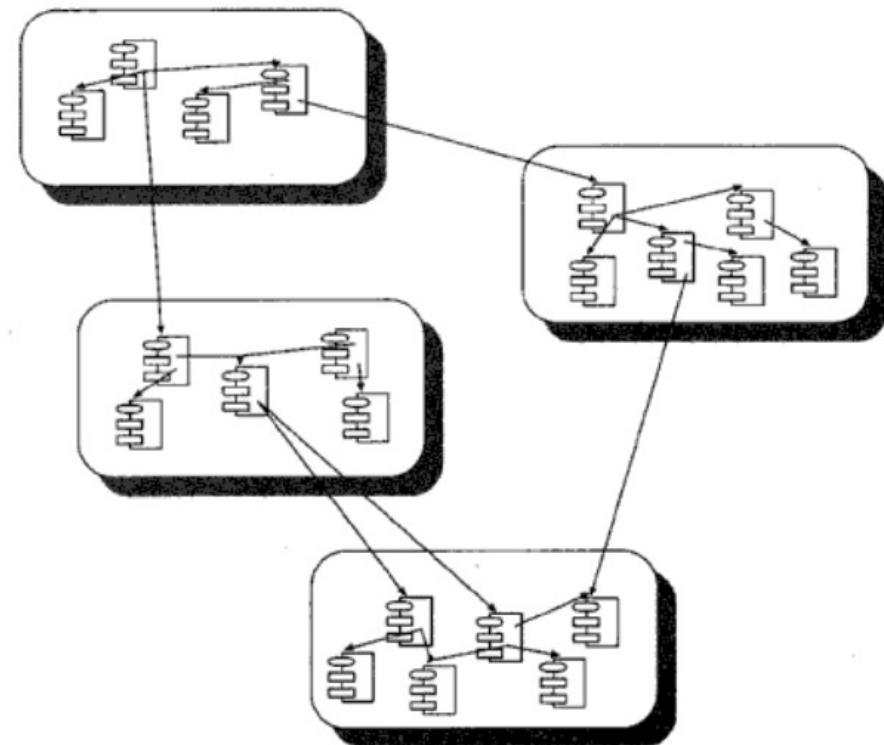
The Topology of Late Third-Generation Programming Languages



The Topology of Small- to Moderate-Sized Applications using Object-Based and Object-Oriented Programming Languages



The Topology of Large Applications Using Object-Based and Object-Oriented Programming Languages



Foundations of the Object Model

- Object-oriented analysis and design represents an evolutionary development
- A **Smalltalk programmer** - methods
- A **C++ programmer** - virtual member functions
- A **CLOS programmer** - generic functions.
- An **Object Pascal programmer** -type coercion
- An **Ada programmer** calls the same thing a type conversion.
- ***Object-oriented Programming*** – *a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.*

- A language is object-oriented if and only if it satisfies the following requirements:
 - It supports objects that are data abstractions with an interface of named operations and a hidden local state.
 - Objects have an **associated type** [class].
 - Types [classes] may **inherit attributes from supertypes** [superclasses]
- ***Object-oriented Design*** - *a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design*
- ***Object-oriented Analysis*** - *a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain.*

Contributions to the Foundation of the Object Model

- "Advances in computer architecture, including capability systems and hardware support for operating systems concepts
- Advances in programming languages, as demonstrated in Simula, Smalltalk, CLU, and Ada
- Advances in programming methodology, including modularization and "information hiding"
- Advances in database models
- Research in artificial intelligence
- Advances in philosophy and cognitive science

Elements of the Object Model

- There are five main kinds of programming styles:
 - **Procedure-oriented** Algorithms
 - **Object-oriented** Classes and objects
 - **Logic-oriented** Goals, often expressed in a predicate calculus
 - **Rule-oriented** If-then rules
 - **Constraint-oriented** Invariant relationships
- There is no single programming style that is best for all kinds of applications
- There are four major elements of this model: **Abstraction, Encapsulation, Modularity & Hierarchy.**

- **Encapsulation** is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation.
- **Modularity** is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.
- **Hierarchy** is a ranking or ordering of abstractions.
- **Concurrency** is the property that distinguishes an active object from one that is not active.
- **Persistence** is the property of an object through which its existence transcends time (i.e. the object continues to exist after its creator ceases to exist) and/or space (i. e. the objects location moves from the address space in which it was created).

```
// c++ program to explain
// Encapsulation

#include<iostream>
using namespace std;

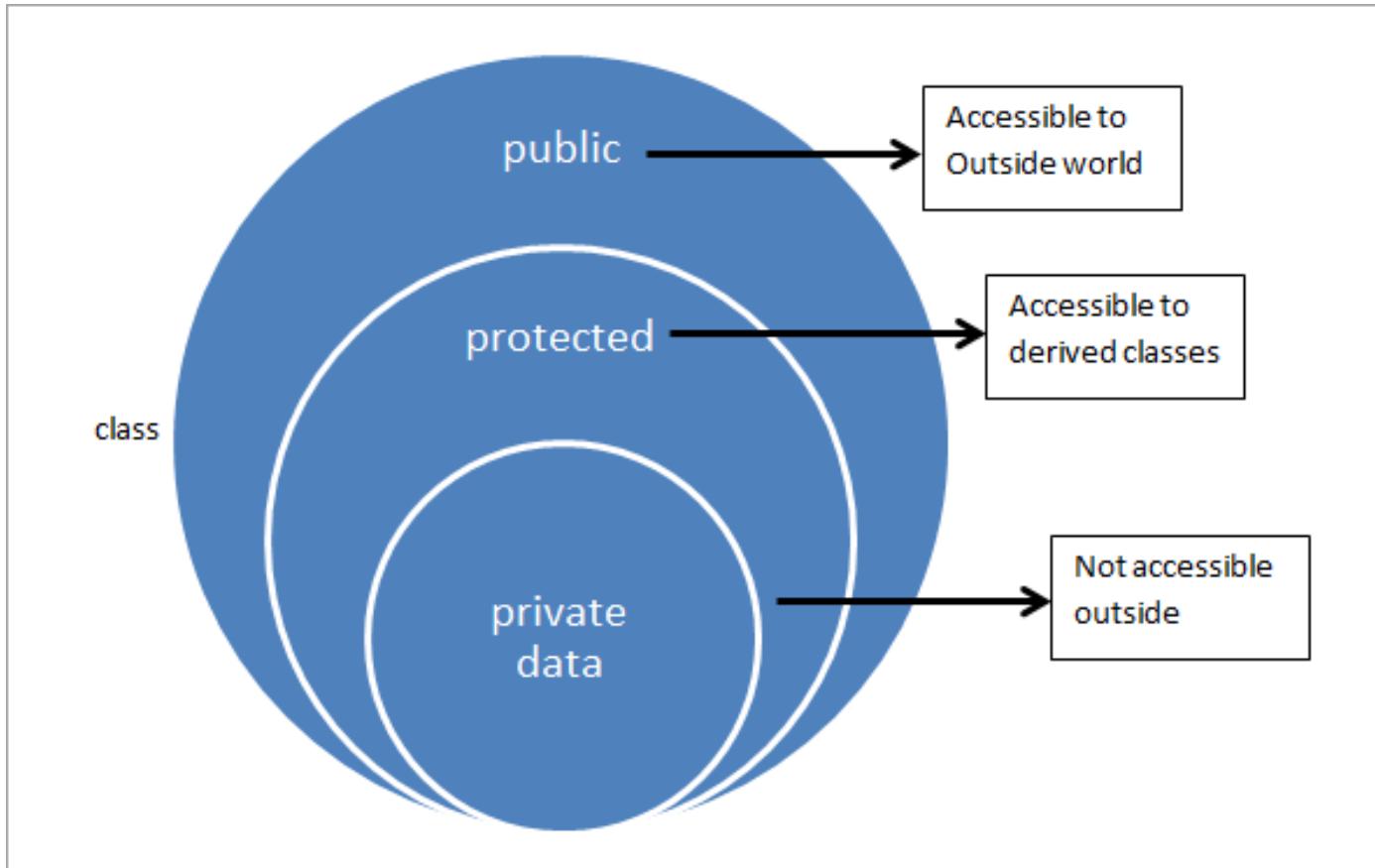
class Encapsulation
{
    private:
        // data hidden from outside world
        int x;

    public:
        // function to set value of
        // variable x
        void set(int a)
        {
            x = a;
        }

        // function to return value of
        // variable x
        int get()
        {
            return x;
        }
};

/
```

Encapsulation



Applying the Object Model

- The use of the object model helps us to exploit the expressive power of object-based and object-oriented programming languages.
- Encourages the reuse not only of software but of entire designs, leading to the creation of reusable application frame-works
- Produces systems that are built upon stable intermediate forms, which are more resilient to change.

Applications of the Object Model

Air traffic control
Animation
Avionics
Banking and insurance software
Business data processing
Chemical process control
Command and control systems
Computer aided design
Computer aided education
Computer integrated manufacturing
Databases
Document preparation
Expert systems
Film and estage storyboarding
Hipermedia
Image recognition

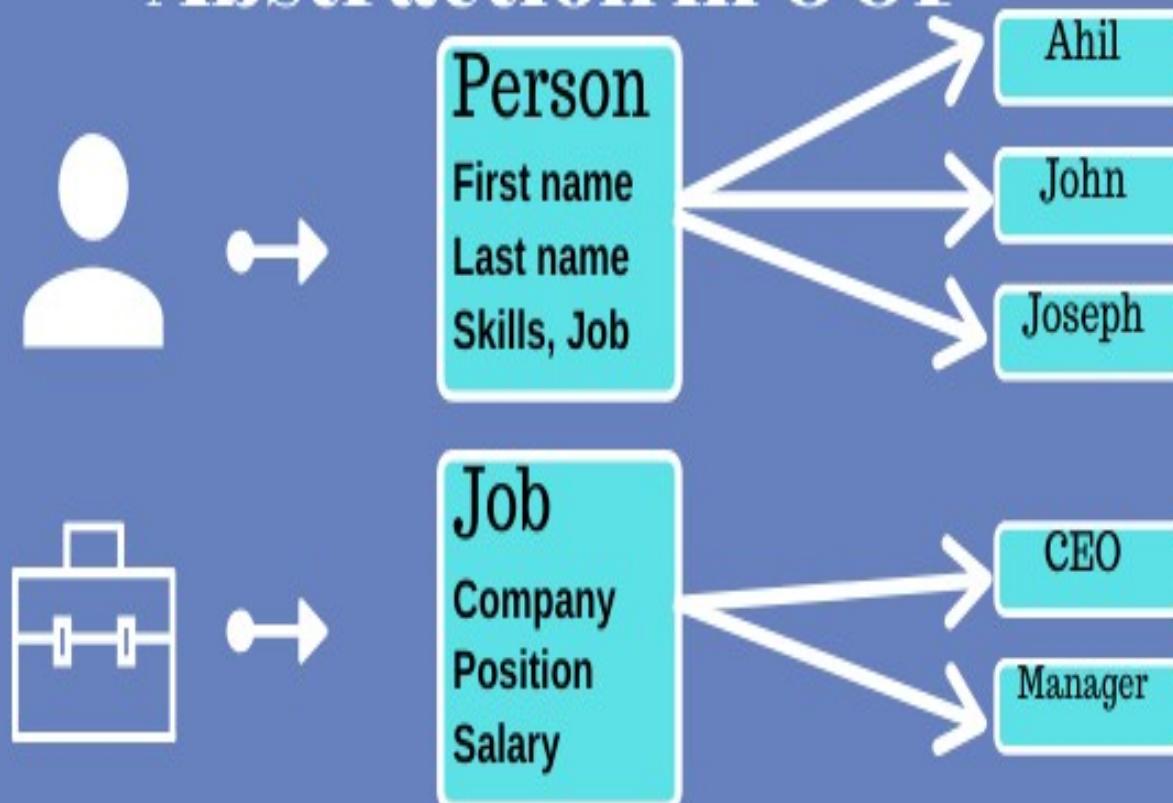
Investment strategiei
Mathematical analysis
Medical electronics
Music composition
Office automation
Operating systems
Petroleum engineering
Reusable software components
Robotics
Software development environments
Space station software
Spacecraft and aircraft simulation
Telecommunications
Telemetry systems
User interface design
VLSI design

Abstractions in OOP

- **Objects** are the building blocks of Object-Oriented Programming.
- An object contains **some properties and methods**.
- It is possible to hide them from the outer world through access modifiers.
- Provide access only for required functions and properties to the other programs.
- This is the general procedure to implement abstraction in OOPS.
- **Abstract Class** is a type of class in OOPs, that declare one or more abstract methods.
- **Abstract Method** is a method that has just the method definition but does not contain implementation.

Abstraction in OOP

© www.learnsimpli.com



When to use Abstract Methods & Abstract Class?

- **Abstract methods** are mostly declared where two or more subclasses are also doing the same thing in different ways through different implementations.
- It extends the same Abstract class and offers different implementations of the abstract methods.
- **Abstract classes** help to describe generic types of behaviors and object-oriented programming class hierarchy.
- It describes subclasses to offer implementation details of the abstract class.

Difference between Abstraction and Encapsulation

Abstraction

Abstraction in Object Oriented Programming solves the issues at the design level.

Abstraction in Programming is about hiding unwanted details while showing most essential information.

Data Abstraction in Java allows focussing on what the information object must contain

Encapsulation

Encapsulation solves it implementation level.

Encapsulation means binding the code and data into a single unit.

Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons.



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN SESSION 14

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Dr Thangakumar J
WILP & Department of CS & IS



BITS Pilani
Pilani Campus

Operating Systems – Software Design (Cont..)



Last Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none"> • The General Concept of Abstraction • Types of Abstractions in Software Engineering <ul style="list-style-type: none"> • Data Abstraction • Control Abstraction • Design Styles in Programming and Programming Languages: Procedural Design vs. OO Design <ul style="list-style-type: none"> • Object Oriented Abstraction • The Evolution of the Object Model • Foundations of the Object Model • Elements of the Object Model • Applying the Object Model • Abstractions in OOP 	<p>R4 (2.1, 2.2, 2.3, 2.4) and Class Notes</p>

Software Design

Today's topic

- Inheritance and Delegation; Design with inheritance and delegation; Comparison: Program Composition – Object Hierarchy vs. Class Hierarchy
- OO Design and Design Patterns
- Design of Complex Software (Compilers/ OS/ Web Server as an example).

When to use Inheritance & Delegation?

Inheritance should only be used when:

- Both classes are in the same logical domain
- The subclass is a proper subtype of the superclass
- The superclass's implementation is necessary or appropriate for the subclass
- The enhancements made by the subclass are primarily additive.

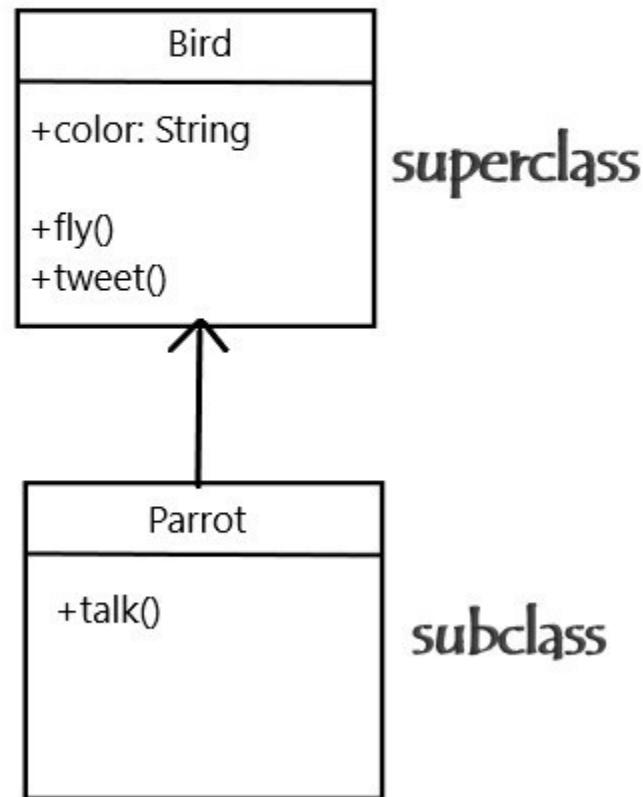
Delegation should only be used:

1. To represent or refer to one or more functions.
2. To define call-back methods.
3. To consume a delegate, we need to create an object to delegate.

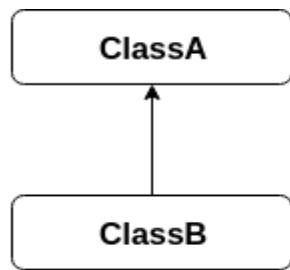
Inheritance

- Inheritance defines a relationship among classes, wherein one class shares the structure or behavior defined in one or more classes.
- It represents a **hierarchy of abstractions**, in which a subclass inherits from one or more super classes.
- Typically, a subclass augments or redefines the existing structure and behavior of its super classes.
- It implies a **generalization/ specialization hierarchy**, wherein a subclass specializes the more general structure or behavior of its super classes.

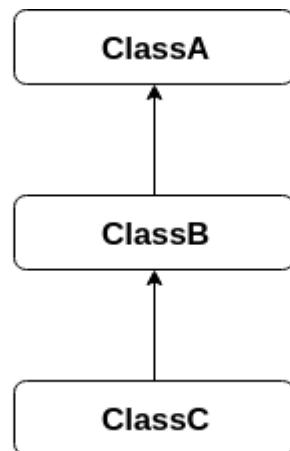
Inheritance



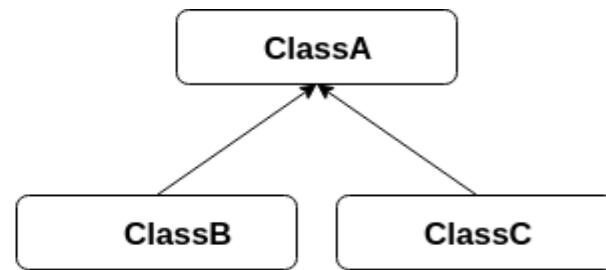
Types of Inheritance



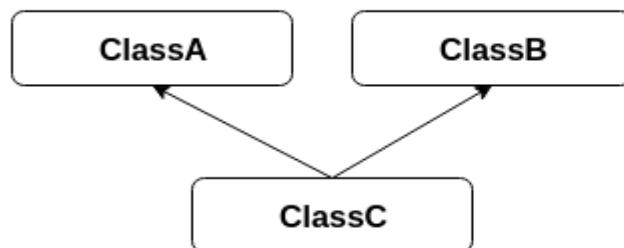
Single Inheritance



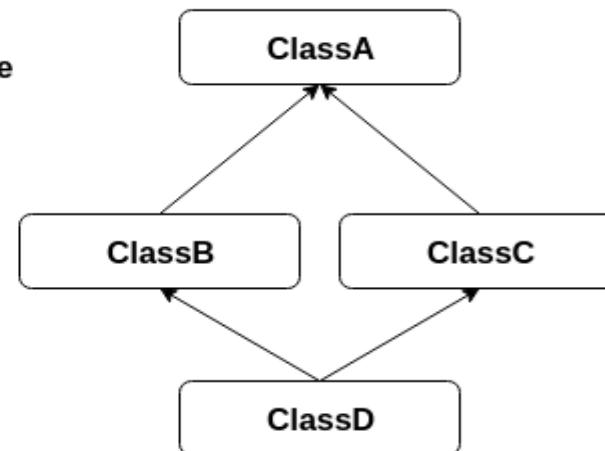
Multilevel Inheritance



Hierarchical Inheritance

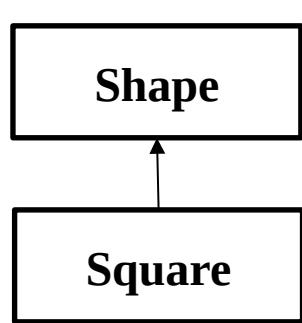


Multiple Inheritance

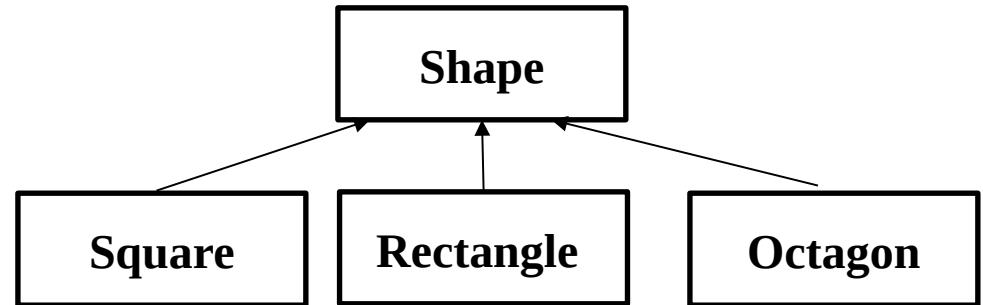


Hybrid Inheritance

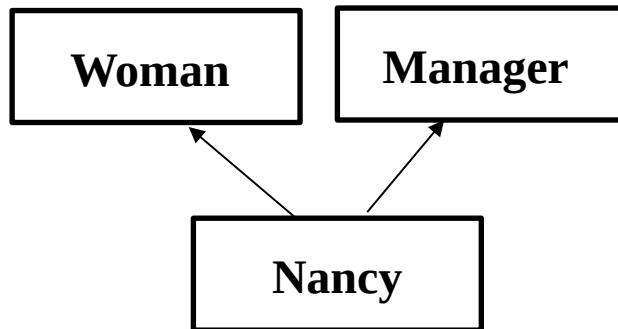
Example



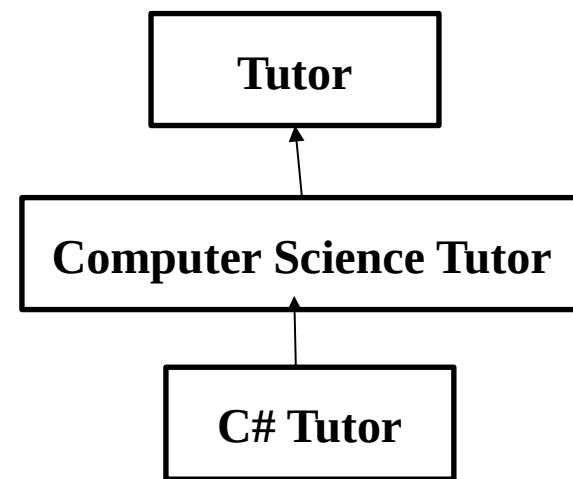
Single Inheritance



Hierarchical Inheritance



Multiple Inheritance



Multi-Level Inheritance

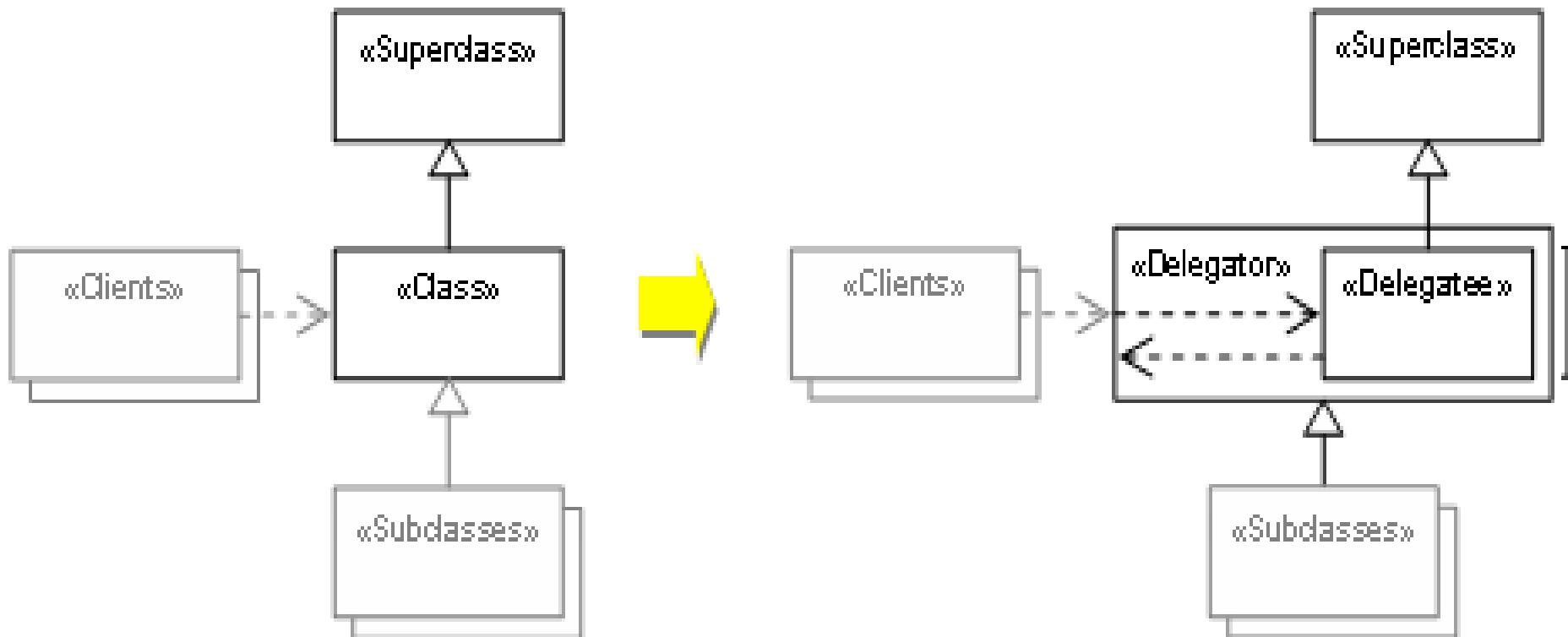
Use of Inheritance

- Used during object design
- Reduce redundancy
- Extensibility
- Specialization or generalization
- Used during requirements analysis

Delegation

- **Delegation** - objects delegate their behavior to related objects. i.e., The act of one object forwarding to another object an operation to be performed on behalf of the first object.
- **Advantage** - run-time flexibility
- **Disadvantage** - delegation is not directly supported by most popular object-oriented languages

Replacing Inheritance by Delegation

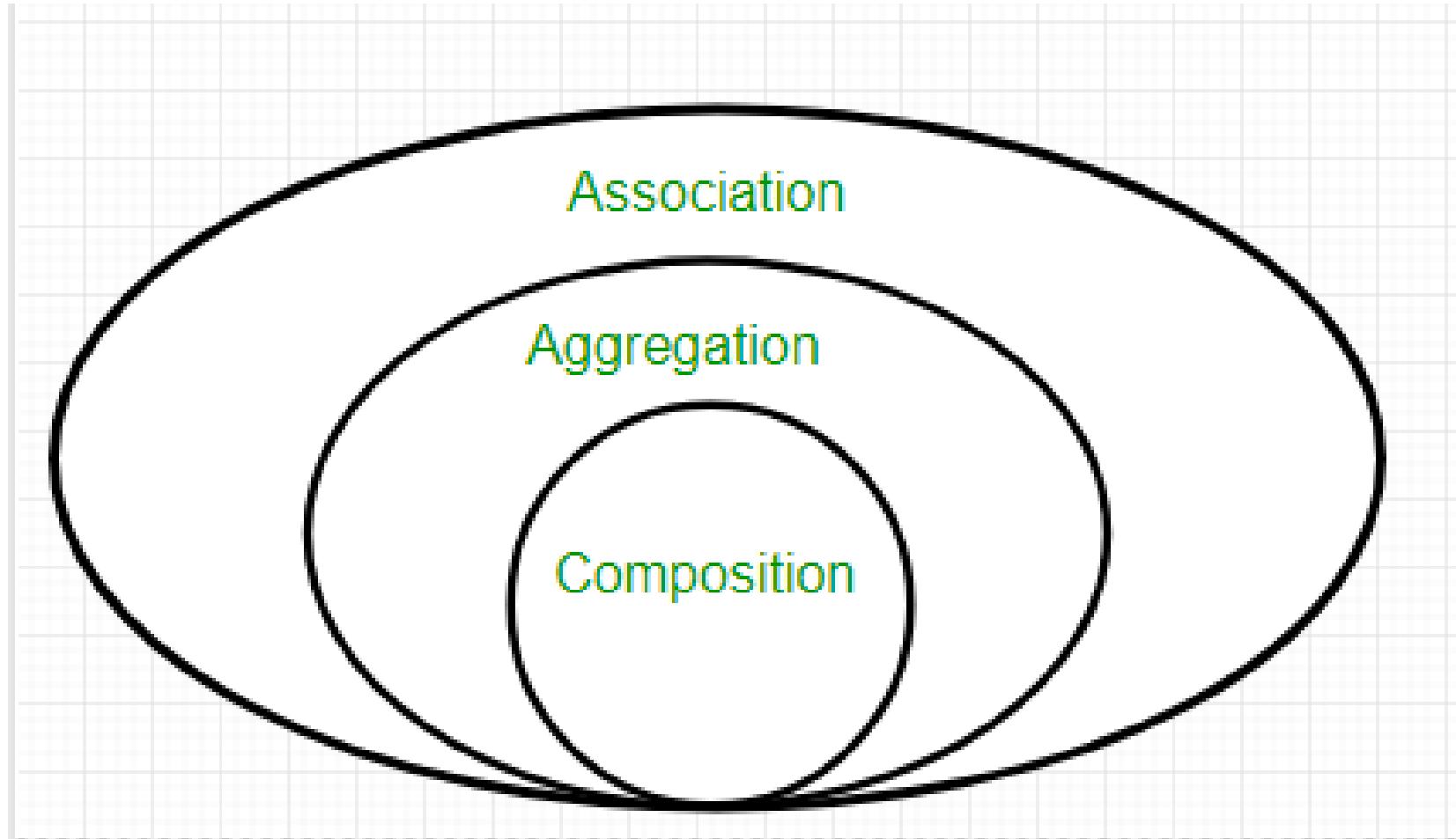


Delegation	Inheritance
Code sharing organization of objects	Code sharing organization of classes
Example: Java 1.1	Java 1.0
Adv: Flexibility, the reused class can be changed without changing reusing class	Adv: straightforward to use, Callbacks do not have to be public
Disadv: inefficiency	Disadv: exposes the subclass to the detail of its parent class

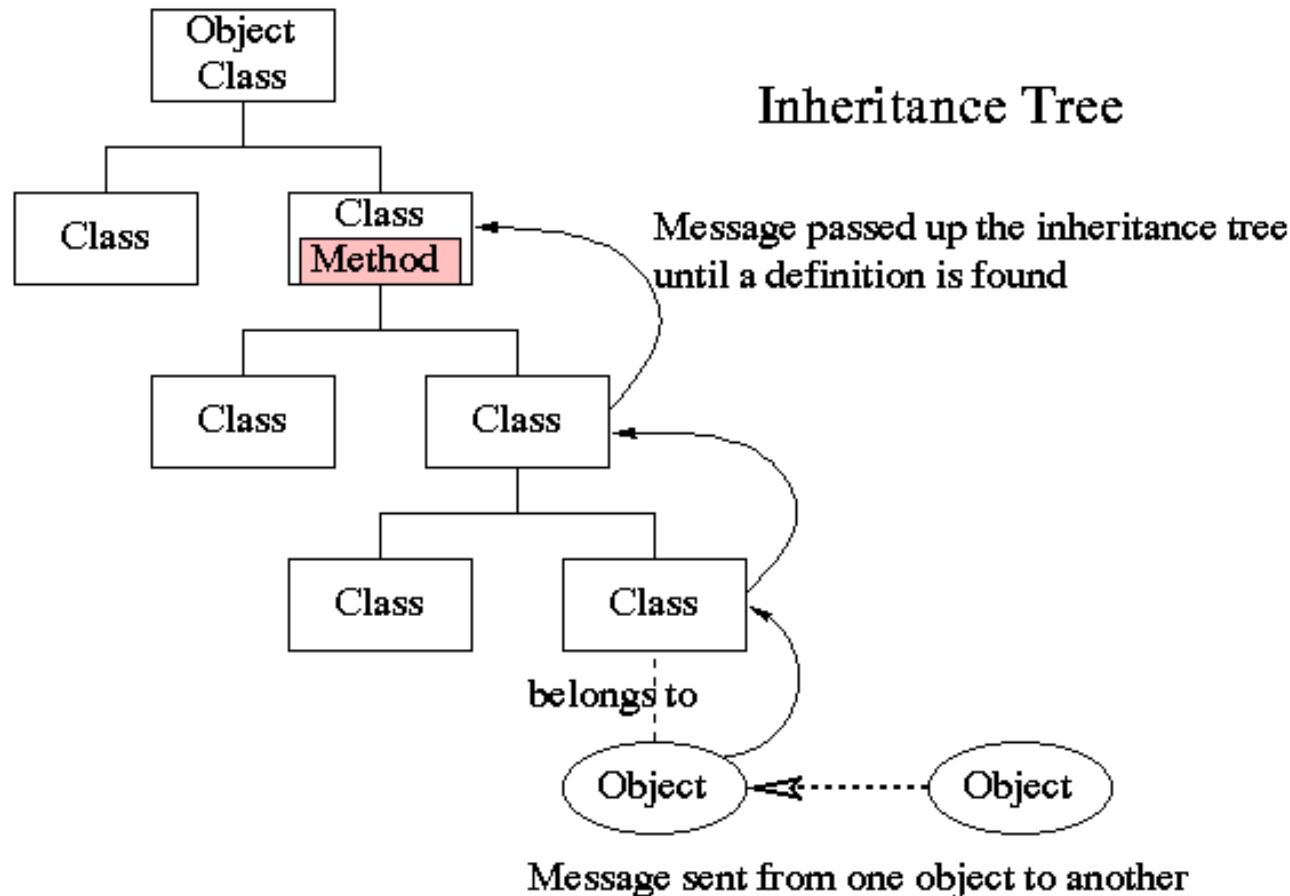
Program Composition

- **Composition** - describes a class that references one or more objects of other classes in instance variables. This allows to model a **has-a-relationship** association between objects.
- **Benefits** - reuse existing code, design clean APIs and change the implementation of a class used in a composition without adapting any external clients

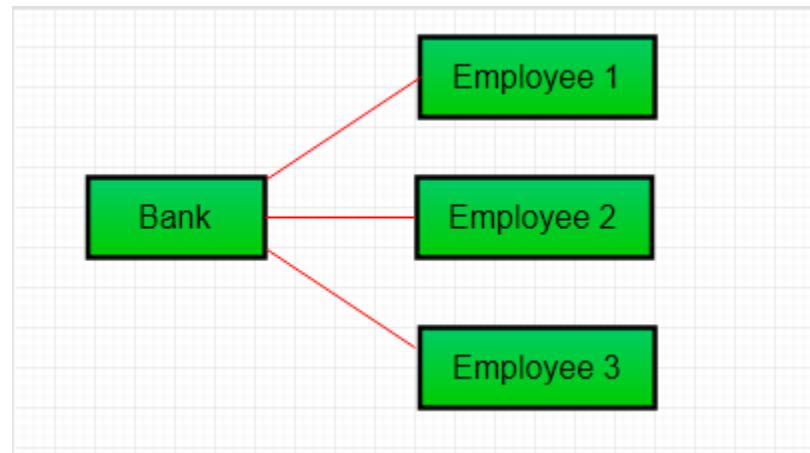
Program Composition



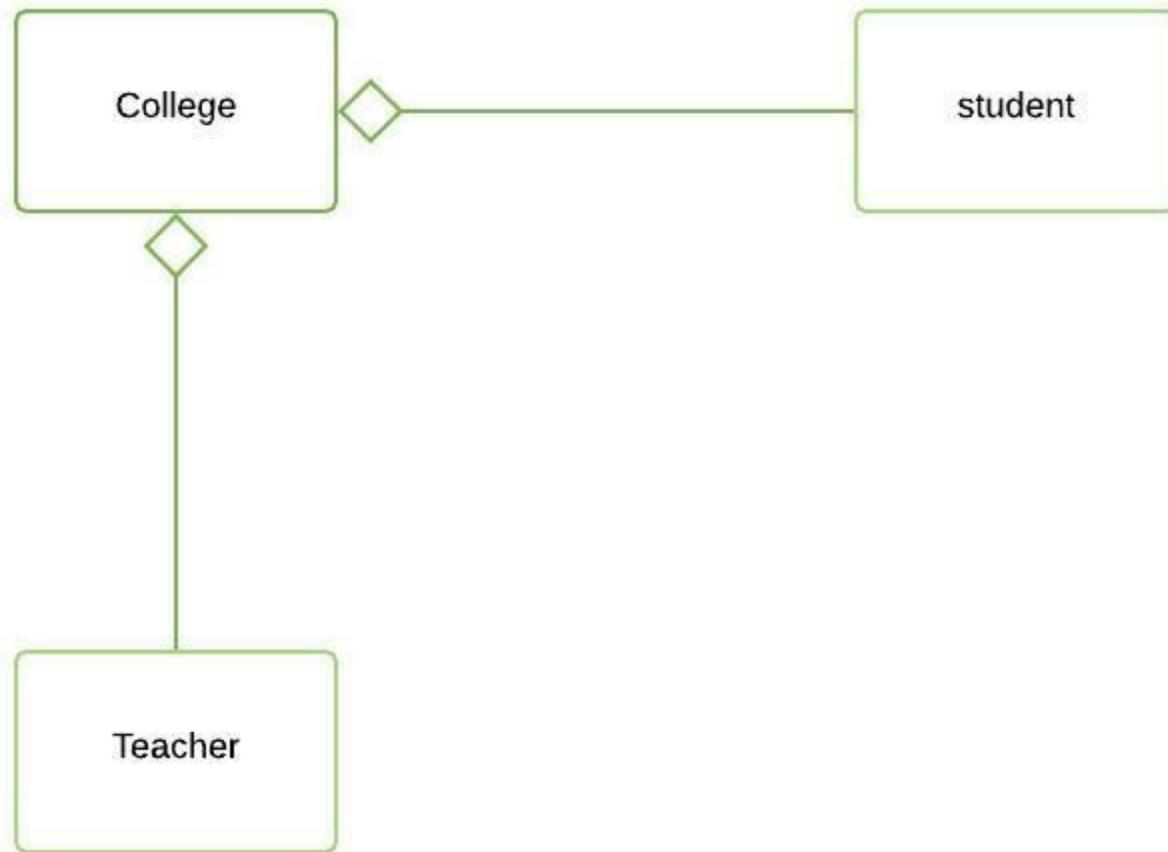
Class Hierarchy



Aggregation



Association



Uses of Hierarchy

- An **object hierarchy** shows the **order and the linkage objects** possess when they link themselves
- A **class hierarchy** is an assembly of classes, a class hierarchy is popularly known as the **hierarchy tree**, it shows the **relationship and order of objects**.
- The depth of a **hierarchy** does not hinder the inheritance of a class variable and method down the hierarchical level

Object-oriented Design

- The users must be able to use objects of subclasses via references to base classes without noticing any difference.
- When using an object through its base class interface, the object of a subclass must not expect the user to obey preconditions that are stronger than those required by the base class.
- The five basic concepts of object-oriented design are the implementation level features that are built into the programming language: Object/Class, information hiding, inheritance, OOP and polymorphism.

Design Patterns

- Template solutions that developers have refined over time to solve a range of recurring problems.
- Elements of design patterns are:
 - **Name** : identifies pattern
 - **Problem description**: describes when to apply the pattern in terms of the problem and context
 - **Solution**: describes elements that make up the design, their relationship, responsibilities, and collaborations.
 - **Consequences**: results and trade-offs of applying the pattern

Gang of four in Design Patterns

- Design patterns are primarily based on the following principles of object orientated design.
 - Program to an interface not an implementation
 - Favor object composition over inheritance

Usage of Design Pattern

Design Patterns have two main usages in software development.

Common platform for developers

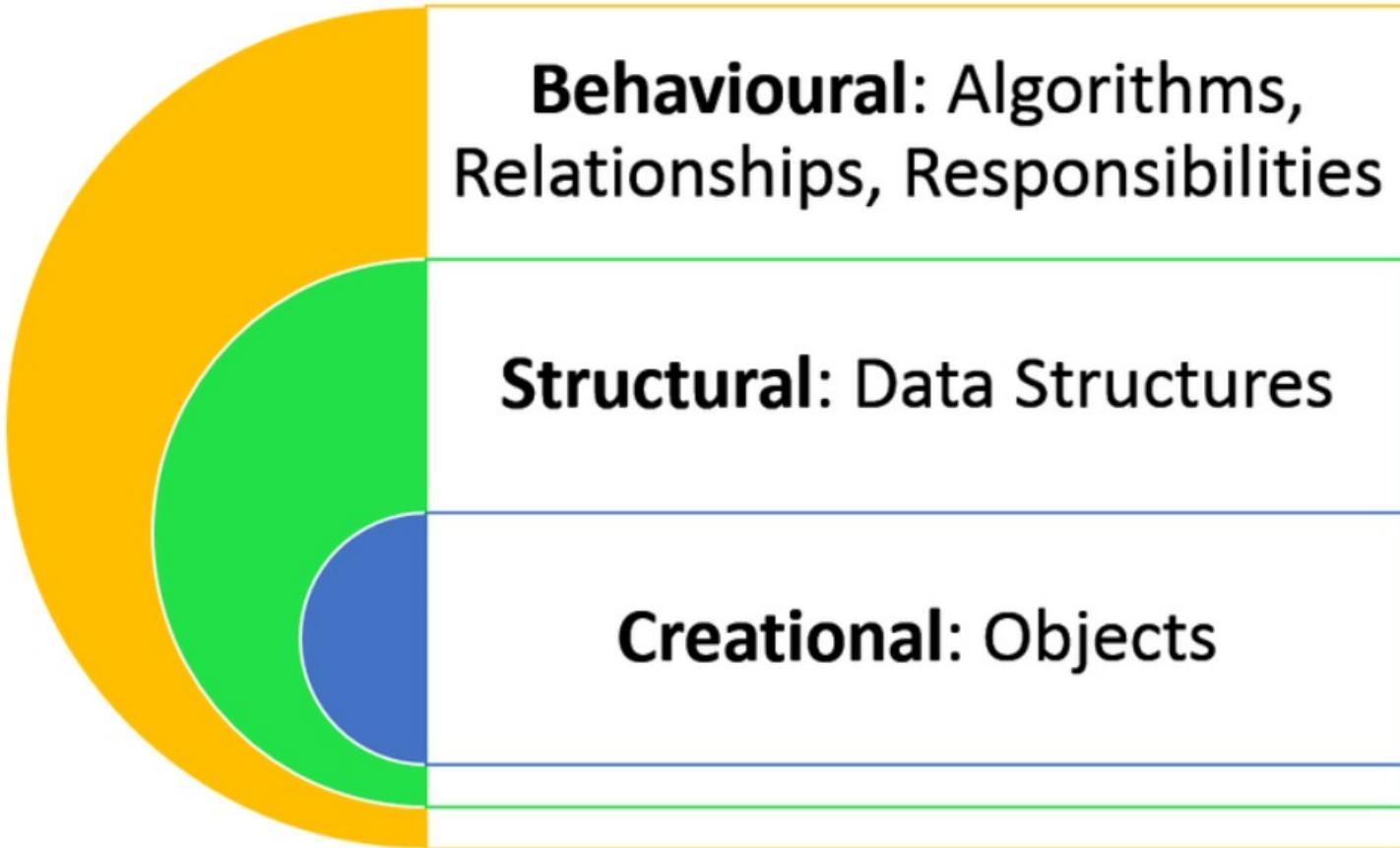
- Provide a standard terminology and are specific to particular scenario.

Best Practices

- Evolved over a long period of time and they provide best solutions to certain problems faced during software development.
- Learning these patterns helps un-experienced developers to learn software design in an easy and faster way.

Types of Design Patterns

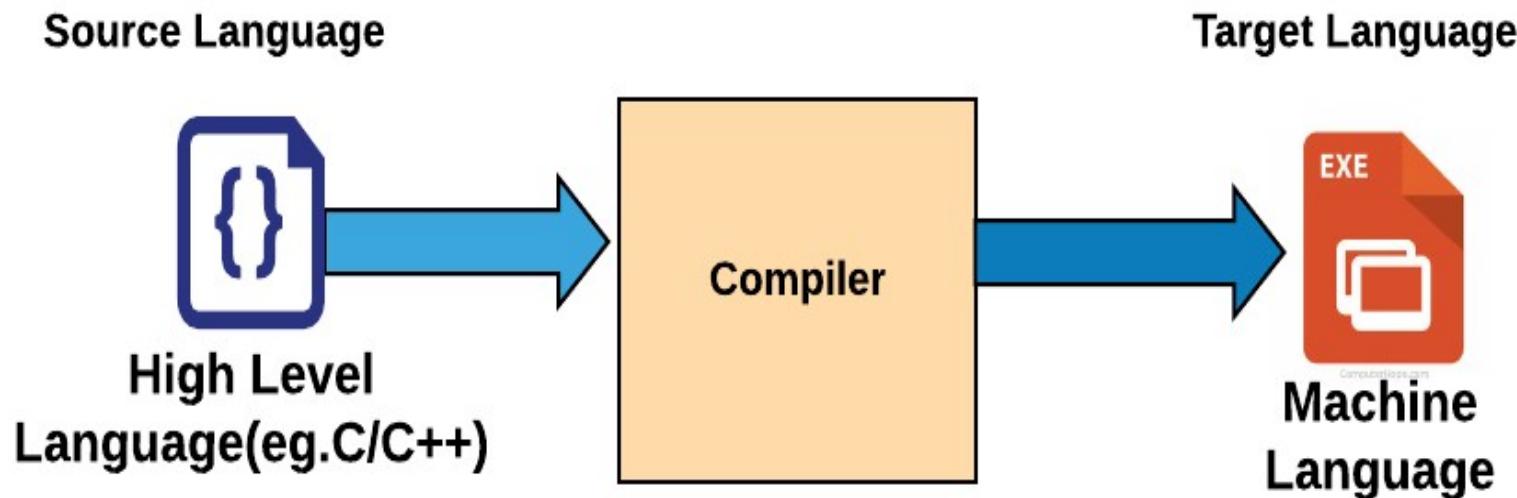
Design Patterns



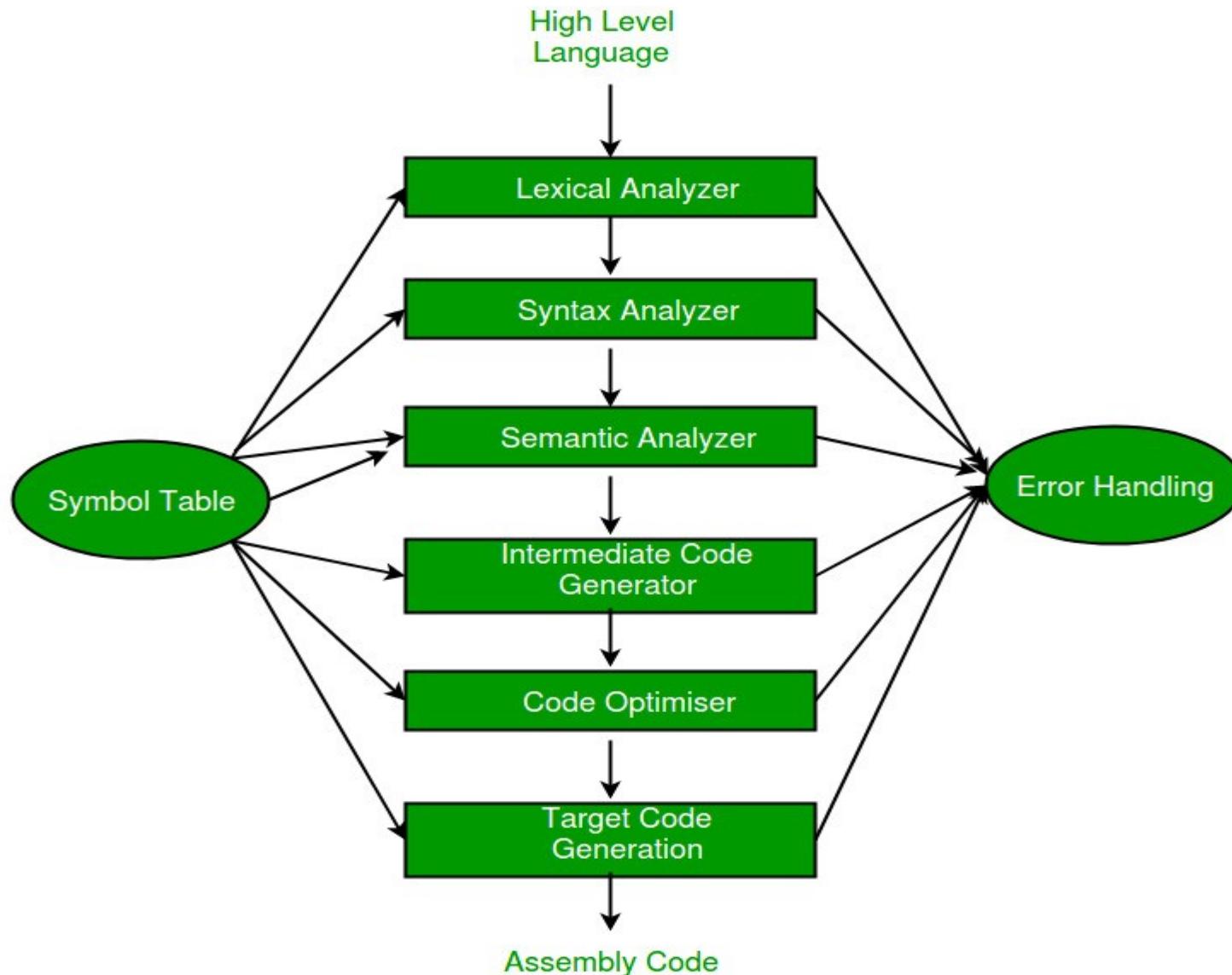
Advantages of Design Patterns

- More stable,
- Robust and
- Scalable
- Reusable in multiple projects
- Provide the solutions that help to define the system architecture
- Provide transparency to the design of an application

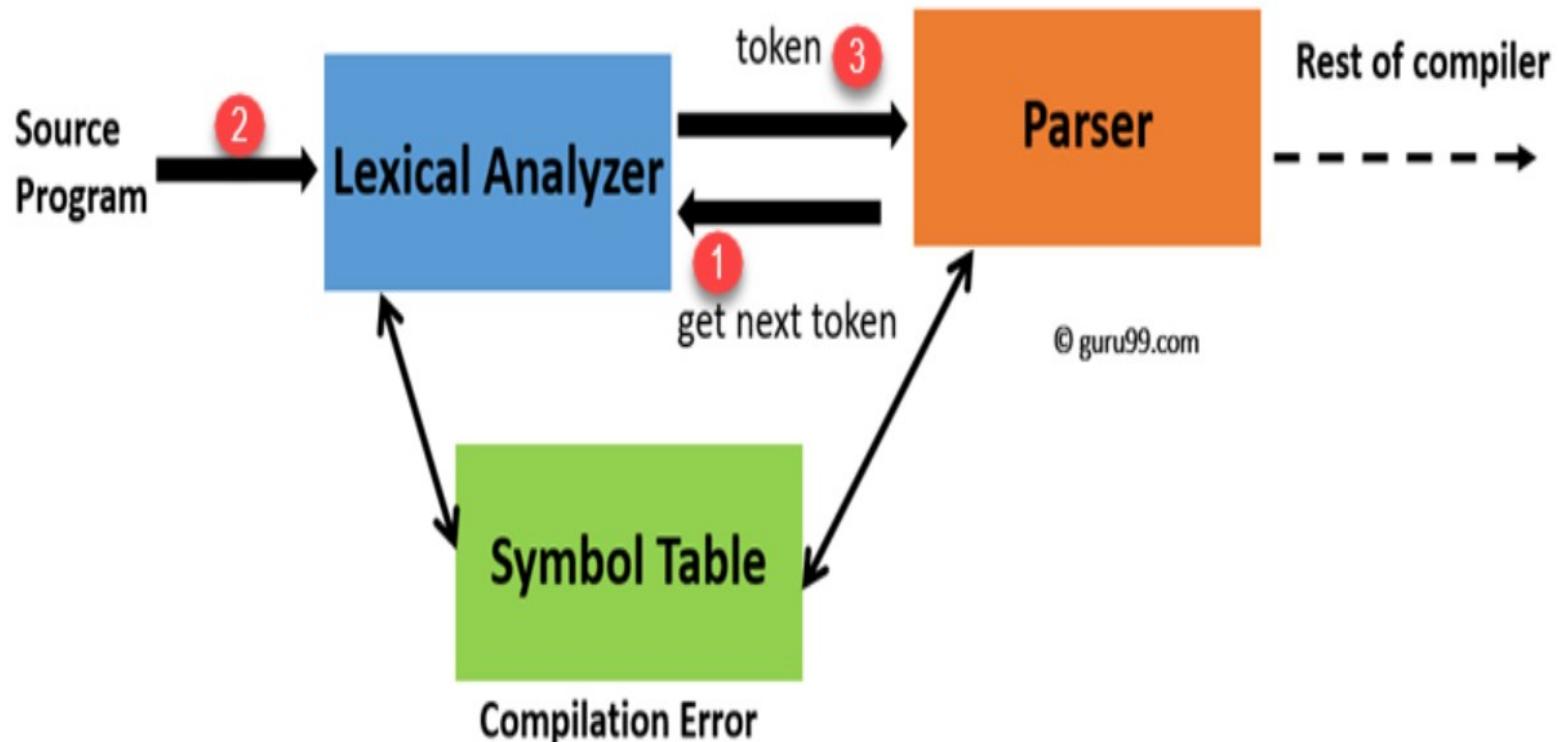
Design of Complex Software - Compiler



Phases of a Compiler



Lexical Analyzer



Example of Lexical Analyzer in Compiler

- A statement through lexical analyzer –
- **a = b + c ;** It will generate token sequence like this:
- **id=id+id;** Where each **id** refers to it's variable in the symbol table referencing all details

```
int main()
{
// 2 variables
int a, b;
a = 10;
return 0;
}
```

- All the valid tokens are:

```
'int' 'main' '(' ')' '{' 'int' 'a' ',' 'b' ';' 'a' '=' '10' ';' 'return' '0' ';' '}'
```

Example of Syntax Analyzer in Compiler

- It checks the syntactical structure of the given input.
- The parse tree is constructed by using the pre-defined Grammar of the language and the input string.
- If the given input string can be produced with the help of the syntax tree (in the derivation process), the input string is found to be in the correct syntax. if not, error is reported by syntax analyzer.

S -> cAd

A -> bc|a

And the input string is “cad”

- Now the parser attempts to construct syntax tree from this grammar for the given input string. It uses the given production rules and applies those as needed to generate the string.

Example of Semantic Analyzer in Compiler

- It uses **syntax tree and symbol table** to check whether the given program is semantically consistent with language definition.
- It gathers type information and stores it in either syntax tree or symbol table.
- This type information is subsequently used by compiler during intermediate-code generation.

```
float x = 10.1;  
float y = x*30;
```

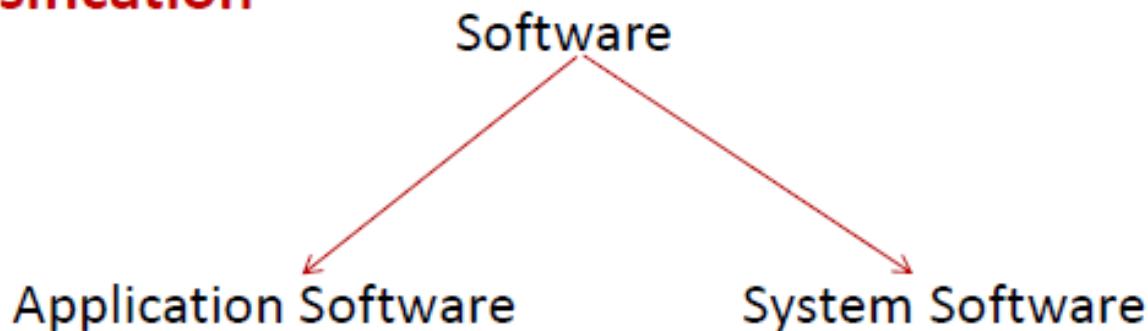
- In the above example integer 30 will be type casted to float 30.0 before multiplication, by semantic analyzer.

- Assemblers
- Assembler's functions
- Assembler directives
- Object code structure
- Object code
- Two pass assembler
- Data structures for two pass assembler
- Operation code table
- Symbol table and location counter

Software

- software, is a collection of computer programs and related data that provide the instructions telling a computer *what to do* and *how to do it*

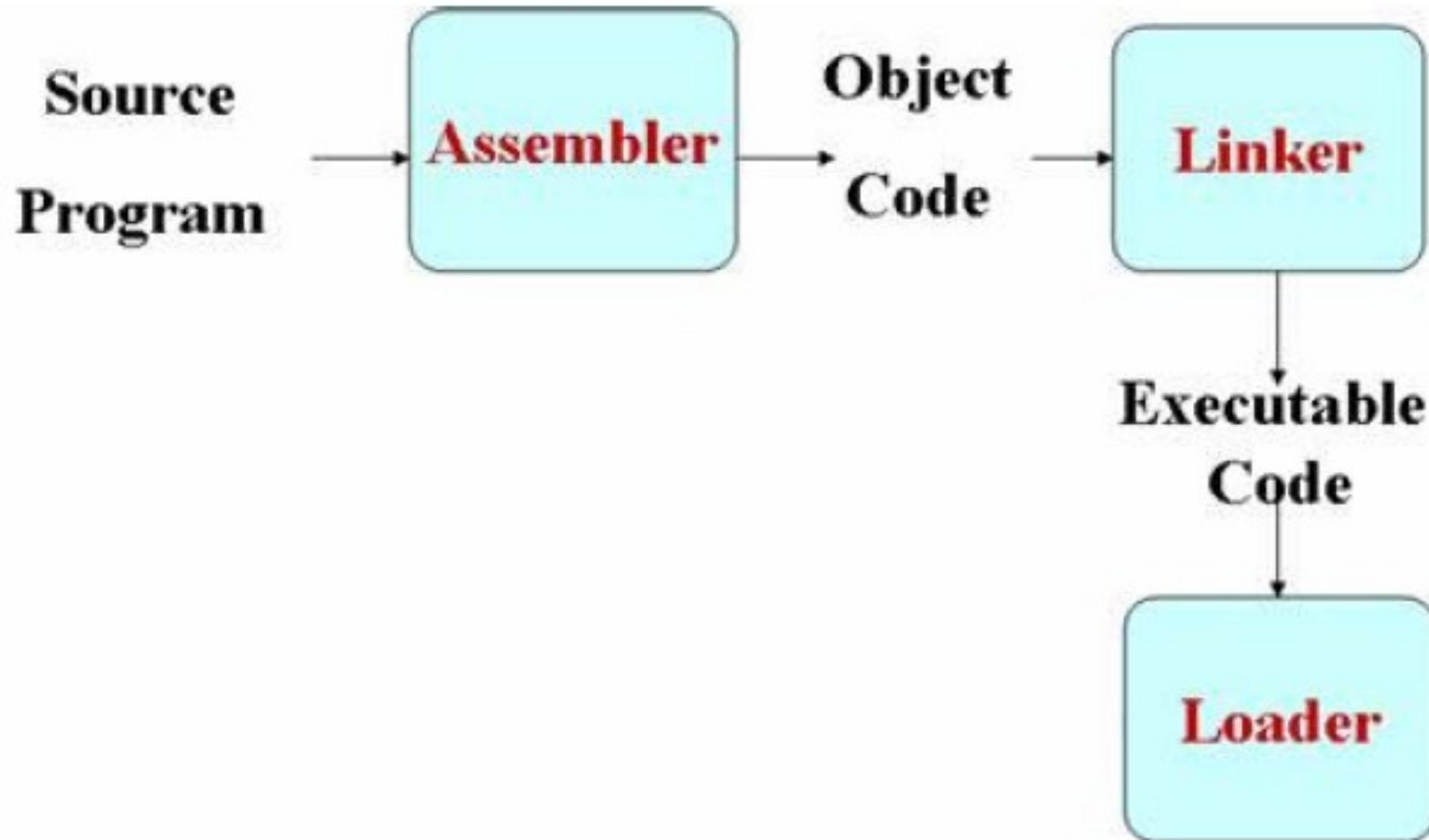
Classification



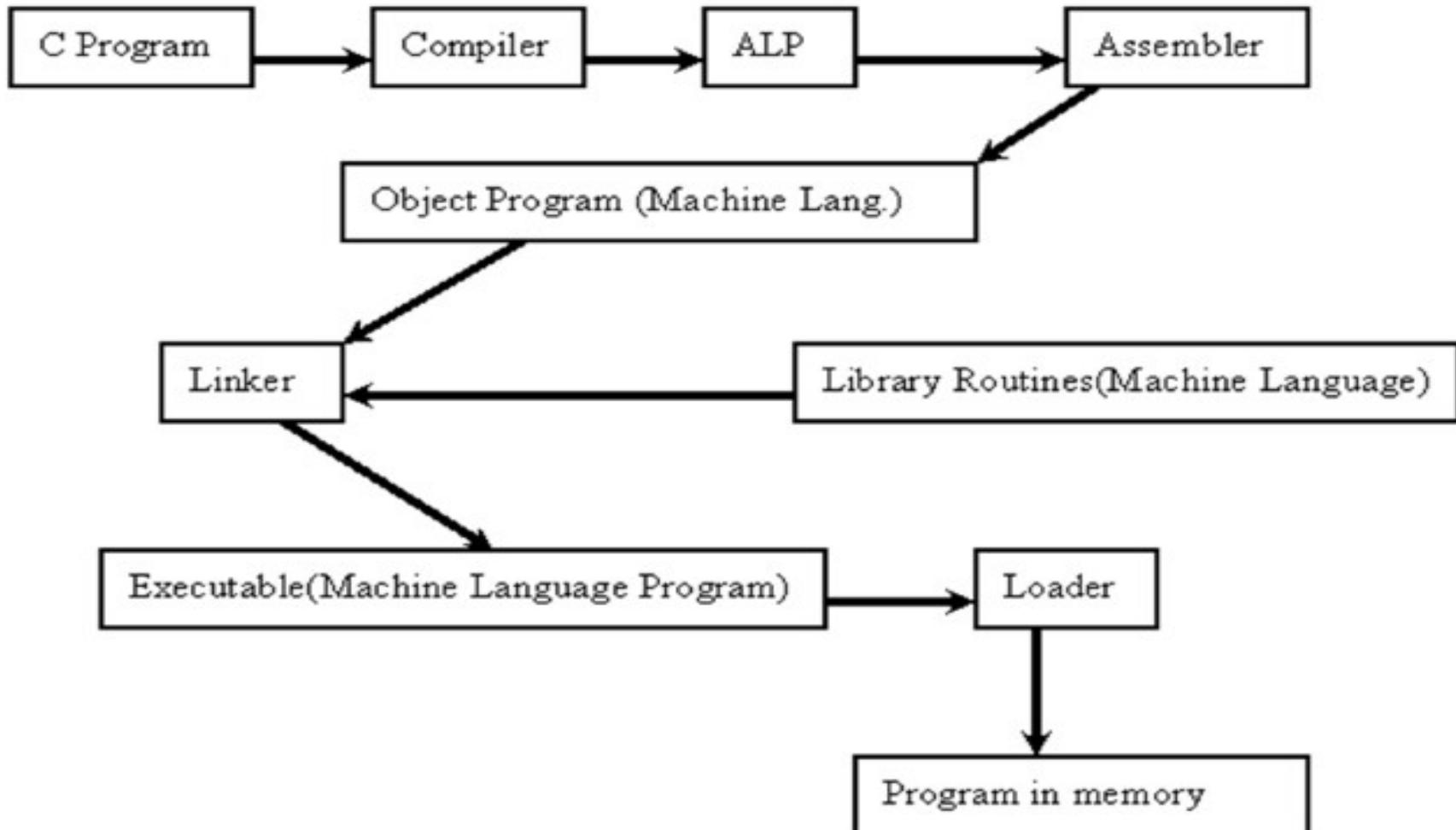
System software

- System software is computer software designed to operate the computer hardware and to provide a platform for running application software.
- Examples :
 - Loaders
 - Assembler
 - Linkers
 - Operating system
 - Compilers
 - Editors

Assemblers



Assemblers



Assembler's functions

- Convert mnemonic operation codes to their machine language equivalents
- Convert symbolic operands to their equivalent machine addresses
- Build the machine instructions in the proper format
- Convert the data constants to internal machine representations
- Write the object program and the assembly listing

Pseudo code

```
Program copy {
    save return address;
cloop: call subroutine RDREC to read one record;
    if length(record)=0 {
        call subroutine WRREC to write EOF;
    } else {
        call subroutine WRREC to write one record;
        goto cloop;
    }
    load return address
    return to caller
}
```

Pseudo code

```
Subroutine RDREC {  
    clear A, X register to 0;  
    rloop: read character from input device to A register  
    if not EOR {  
        store character into buffer[X];  
        X++;  
        if X < maximum length  
            goto rloop;  
    }  
    store X to length(record);  
    return  
}
```

EOR:
character x'00'

Pseudo code

```
Subroutine WDREC {  
    clear X register to 0;  
    wloop: get character from buffer[X]  
        write character from X to output device  
        X++;  
        if X < length(record)  
            goto wloop;  
    return  
}
```

Assembler Directives

- Pseudo-Instructions
 - Not translated into machine instructions
 - Providing information to the assembler
- Basic assembler directives
 - START Specify name and starting address for the program
 - END end of the program
 - BYTE generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant
 - WORD generate one-word integer constant
 - RESB reserve the indicated number of bytes for a data area
 - RESW reserve the indicated number of words for a data area

Program - takes input from input device

Line	Loc	Label	Inst	Arg	Obj code
5	1000	COPY	START	1000	
10	1000	FIRST	STL	RETADR	141033
15	1003	CLOOP	JSUB	RDREC	482039
20	1006		LDA	LENGTH	001036
25	1009		COMP	ZERO	281030
30	100C		JEQ	ENDFIL	301015
35	100F		JSUB	WRREC	482061
40	1012		J	CLOOP	3C1003
45	1015	ENDFIL	LDA	EOF	00102A
50	1018		STA	BUFFER	0C1039
55	101B		LDA	THREE	00102D
60	101E		STA	LENGTH	0C1036
65	1021		JSUB	WRREC	482061

Program - takes input from input device

Line	Loc	Label	Inst	Arg	Obj code
70	1024		LDL	RETADR	081033
75	1027		RSUB		4C0000
80	102A	EOF	BYTE	C'EOF'	454F46
85	102D	THREE	WORD	3	000003
90	1030	ZERO	WORD	0	000000
95	1033	RETADR	RESW	1	
100	1036	LENGTH	RESW	1	
105	1039	BUFFER	RESB	4096	
110	SUB ROUTINE TO READ RECORD INTO BUFFER				
115					
120					
125	2039	RDREC	LDX	ZERO	041030
130	203C		LDA	ZERO	001030

Assembler

- Translation of source program to object code requires following function
 1. Convert mnemonics operation codes to their machine language equivalents
 - E.g: Translate STL to 14
 2. Convert symbolic operands to their equivalent machine address
 - E.g: Translate RETADR to 1033
 3. Build the machine instructions in the proper format
 4. Convert the data constants specified in the source program into their internal machine representations
 - Translate EOF to 454F46
 5. Write the object program and assembly listing

Assembler

- All the steps except step 2 can easily be accomplished by sequential processing of the source program, one line at a time
- Step 2 can not be achieved by sequential processing
 - Contains forward reference
 - Assembler has to make 2 passes to resolve this
 - Pass 1 for label definition and assign address
 - Pass 2 for translation

Assembler

- Assembler functionality
 - Translate source program to object code
 - Process statements called assembler directives
 - Assembler directives are not translated into machine instructions
 - Assembler directives provide instructions to the assembler itself
 - Example: BYTE, WORD, RESB, RESW, START, END

Object Program

Simple object program format contains 3 types of records

- Header
 - Starting address, program name and length
- Text
 - Translated instructions and data of the program together with an indication of the addresses where these are to be loaded
- End
 - Marks the end of the program and specifies the address

in the program where the execution is to begin

Object Program

Header

Col. 1	H	11 0011 001000 00100100
Col. 2~7	Program name	
Col. 8~13	Starting address (hex)	T 001000 1E 141033 482039 001036 281030 301015 482061 ...
Col. 14-19	Length of object program in bytes (hex)	T 00101E 15 0C1036 482061 081044 4C0000 454E46 000003

Text

Col.1	T		T	002039	1E	041030	001030	E0205D	30203F	D8205D	281030	...
Col.2^7	Starting address in this record (hex)											
Col. 8^9	Length of object code in this record in bytes (hex)											
Col. 10^69	Object code (69-10+1)/6=10 instructions		T	002057	1C	101036	4C0000	F1	001000	041030	E02079	...

End

Col.1	E						
Col.2~7	E	Address of first executable instruction (hex)					

Object Code

H COPY 001000 00107A

T 001000 1E 141033 482039 001036 281030 301015 482061 ...

T 00101E 15 0C1036 482061 081044 4C0000 454F46 000003 ...

T 002039 1E 041030 001030 E0205D 30203F D8205D 281030 ...

T 002057 1C 101036 4C0000 F1 001000 041030 E02079 ...

T 002073 07 382064 4C0000 05

E 001000

Object Code

```
HCOPY 00100000107A
T0010001E1410334820390010362810303010154820613C100300102A0C103900102D
T00101E150C10364820610810334C0000454F46000003000000
T0020391E041030001030E0205D30203FD8205D2810303020575490392C205E38203F
T0020571C1010364C0000F1001000041030E02079302064509039DC20792C1036
T002073073820644C000005
E001000
```

Difficulties: Forward Reference

Forward reference: reference to a label that is defined later in the program.

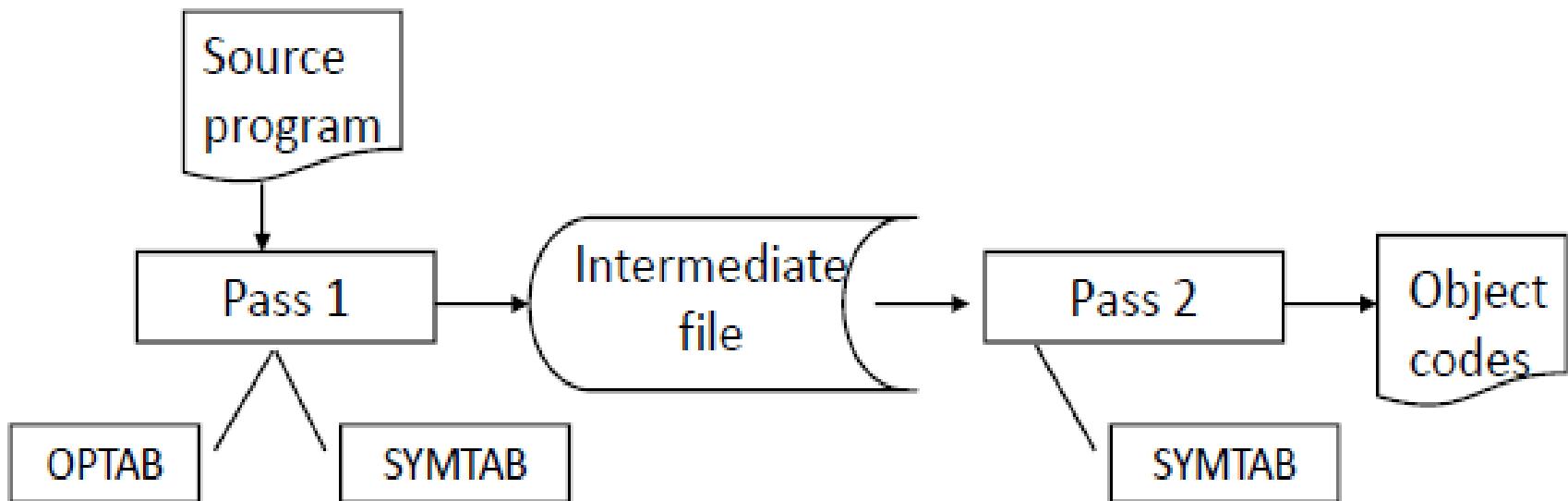
<u>Loc</u>	<u>Label</u>	<u>Operator</u>	<u>Operand</u>	
1000	FIRST	STL	RETADR	
1003	CLOOP	JSUB	RDREC	
...
1012		J	CLOOP	
...
1033	RETADR	RESW	1	

Two Pass Assembler

- **Pass 1**
 - **Assign addresses to all statements in the program**
 - **Save the values assigned to all labels for use in Pass 2**
 - **Perform some processing of assembler directives**
- **Pass 2**
 - **Assemble instructions**
 - **Generate data values defined by BYTE, WORD**
 - **Perform processing of assembler directives not done in Pass 1**
 - **Write the object program and the assembly listing**

Two Pass Assembler

Read from input line
LABEL, OPCODE, OPERAND



Data Structures

- **Operation Code Table (OPTAB)**
- **Symbol Table (SYMTAB)**
- **Location Counter(LOCCTR)**

Data Structures

- Operation Code Table (OPTAB)
 - Used to lookup mnemonic operation codes and translate them to their machine language equivalents
 - If variable length instruction format: OPTAB contains length of the instruction as well
 - Organized as hash table with mnemonic operation code as key
 - OPTAB is a static table

Data Structures

- Symbol Table (SYMTAB)
 - Used to store values (addresses) assigned to labels
 - Contains the name and value (address) for each label in the source program, together with flags to indicate error conditions
 - label name, value, flag, (type, length) etc
 - Organized as hash table for efficiency of insertion & retrieval
 - Dynamic table (insert, delete, search)

COPY	1000
FIRST	1000
CLOOP	1003
ENDFIL	1015
EOF	1024
THREE	102D
ZERO	1030
RETADR	1033
LENGTH	1036
BUFFER	1039
RDREC	2039

Data Structures

- Location Counter(LOCCTR)
 - Used to help in the assignment of addresses
 - Initialized to the address specified in the START statement
 - After each source statement is processed, the length of the assembled instruction or data area to be generated is added to LOCCTR

Algorithm for pass1 Assembler

Pass 1:

```
begin
  read first input line
  if OPCODE = 'START' then
    begin
      save #[OPERAND] as starting address
      initialize LOCCTR to starting address
      write line to intermediate file
      read next input line
    end {if START}
  else
    initialize LOCCTR to 0
  while OPCODE ≠ 'END' do
    begin
      if this is not a comment line then
        begin
          if there is a symbol in the LABEL field then
            begin
              search SYMTAB for LABEL
              if found then
                set error flag (duplicate symbol)
              else
                insert (LABEL,LOCCTR) into SYMTAB
            end {if symbol}
          search OPTAB for OPCODE
```

Algorithm for pass1 Assembler

```
if found then
    add 3 {instruction length} to LOCCTR
else if OPCODE = 'WORD' then
    add 3 to LOCCTR
else if OPCODE = 'RESW' then
    add 3 * #[OPERAND] to LOCCTR
else if OPCODE = 'RESB' then
    add #[OPERAND] to LOCCTR
else if OPCODE = 'BYTE' then
    begin
        find length of constant in bytes
        add length to LOCCTR
    end {if BYTE}
else
    set error flag (invalid operation code)
end {if not a comment}
write line to intermediate file
read next input line
end {while not END}
write last line to intermediate file
save (LOCCTR - starting address) as program length
end {Pass 1}
```

Algorithm for pass 2

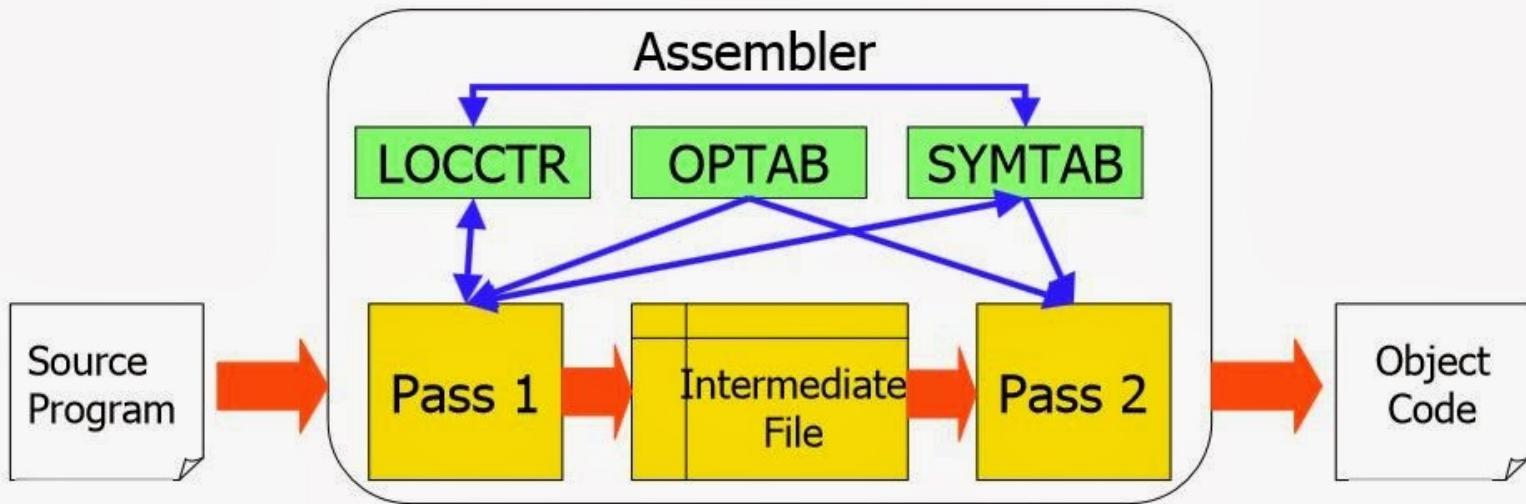
Pass 2:

Assembler

```
begin
    read first input line {from intermediate file}
    if OPCODE = 'START' then
        begin
            write listing line
            read next input line
        end {if START}
    write Header record to object program
    initialize first Text record
    while OPCODE ≠ 'END' do
        begin
            if this is not a comment line then
                begin
                    search OPTAB for OPCODE
                    if found then
                        begin
                            if there is a symbol in OPERAND field then
                                begin
                                    search SYMTAB for OPERAND
                                    if found then
                                        store symbol value as operand address
                                    else
                                        begin
                                            store 0 as operand address
                                            set error flag (undefined symbol)
                                        end
                                end {if symbol}
                            else
                        end
                end
        end
```

Algorithm for pass 2 Assembler

```
        store 0 as operand address
        assemble the object code instruction
end {if opcode found}
else if OPCODE = 'BYTE' or 'WORD' then
        convert constant to object code
if object code will not fit into the current Text record then
        begin
            write Text record to object program
            initialize new Text record
        end
        add object code to Text record
    end {if not comment}
    write listing line
    read next input line
end {while not END}
write last Text record to object program
write End record to object program
write last listing line
end {Pass 2}
```



OPTAB

Mnemonic	Opcode
LDX	04
LDA	00
ADD	18
STA	0C
RSUB	4C
TIX	2C
JLT	38

SYMTAB

Name	LOCCTR
FIRST	1000
LOOP	1006
TABLE	1015
COUNT	1018
ZERO	101B
TOTAL	101E

Assembler Design

- Machine Dependent Assembler Features
 - instruction formats and addressing modes
 - program relocation
- Machine Independent Assembler Features
 - literals
 - symbol-defining statements
 - expressions
 - program blocks
 - control sections and program linking

Machine-dependent Assembler Features

- ✓ Instruction formats and addressing modes
- ✓ Program relocation

Instruction Format and Addressing Mode

- SIC/XE
 - PC-relative or Base-relative addressing: op m
 - Indirect addressing: op @m
 - Immediate addressing: op #c
 - Extended format: +op m
 - Index addressing: op m,x
 - register-to-register instructions
 - larger memory -> multi-programming (program allocation)

Translation

- Register translation
 - register name (A, X, L, B, S, T, F, PC, SW) and their values (0,1, 2, 3, 4, 5, 6, 8, 9)
 - preloaded in SYMTAB
- Address translation
 - Most register-memory instructions use program counter relative or base relative addressing
 - Format 3: 12-bit address field
 - base-relative: 0~4095
 - pc-relative: -2048~2047
 - Format 4: 20-bit address field

Assembler Design

Options

- One-pass assemblers
- Multi-pass assemblers
- Two-pass assembler with overlay structure

Two-Pass Assembler with Overlay Structure

- For small memory
 - pass 1 and pass 2 are never required at the same time
 - three segments
 - root: driver program and shared tables and subroutines
 - pass 1
 - pass 2
 - tree structure
 - overlay program

One-Pass Assemblers

- Main problem
 - forward references
 - data items
 - labels on instructions
- Solution
 - data items: require all such areas be defined before they are referenced
 - labels on instructions: no good solution

One-Pass Assemblers

- Main Problem
 - forward reference
 - data items
 - labels on instructions
- Two types of one-pass assembler
 - load-and-go
 - produces object code directly in memory for immediate execution
 - the other
 - produces usual kind of object code for later execution

Load-and-go Assembler

- Characteristics
 - Useful for program development and testing
 - Avoids the overhead of writing the object program out and reading it back
 - Both one-pass and two-pass assemblers can be designed as load-and-go.
 - However one-pass also avoids the overhead of an additional pass over the source program
 - For a load-and-go assembler, the actual address must be known at assembly time, we can use an absolute program

Forward Reference in One-pass Assembler

- For any symbol that has not yet been defined
 1. omit the address translation
 2. insert the symbol into SYMTAB, and mark this symbol undefined
 3. the address that refers to the undefined symbol is added to a list of forward references associated with the symbol table entry
 4. when the definition for a symbol is encountered, the proper address for the symbol is then inserted into any instructions previous generated according to the forward

Load-and-go Assembler (Cont.)

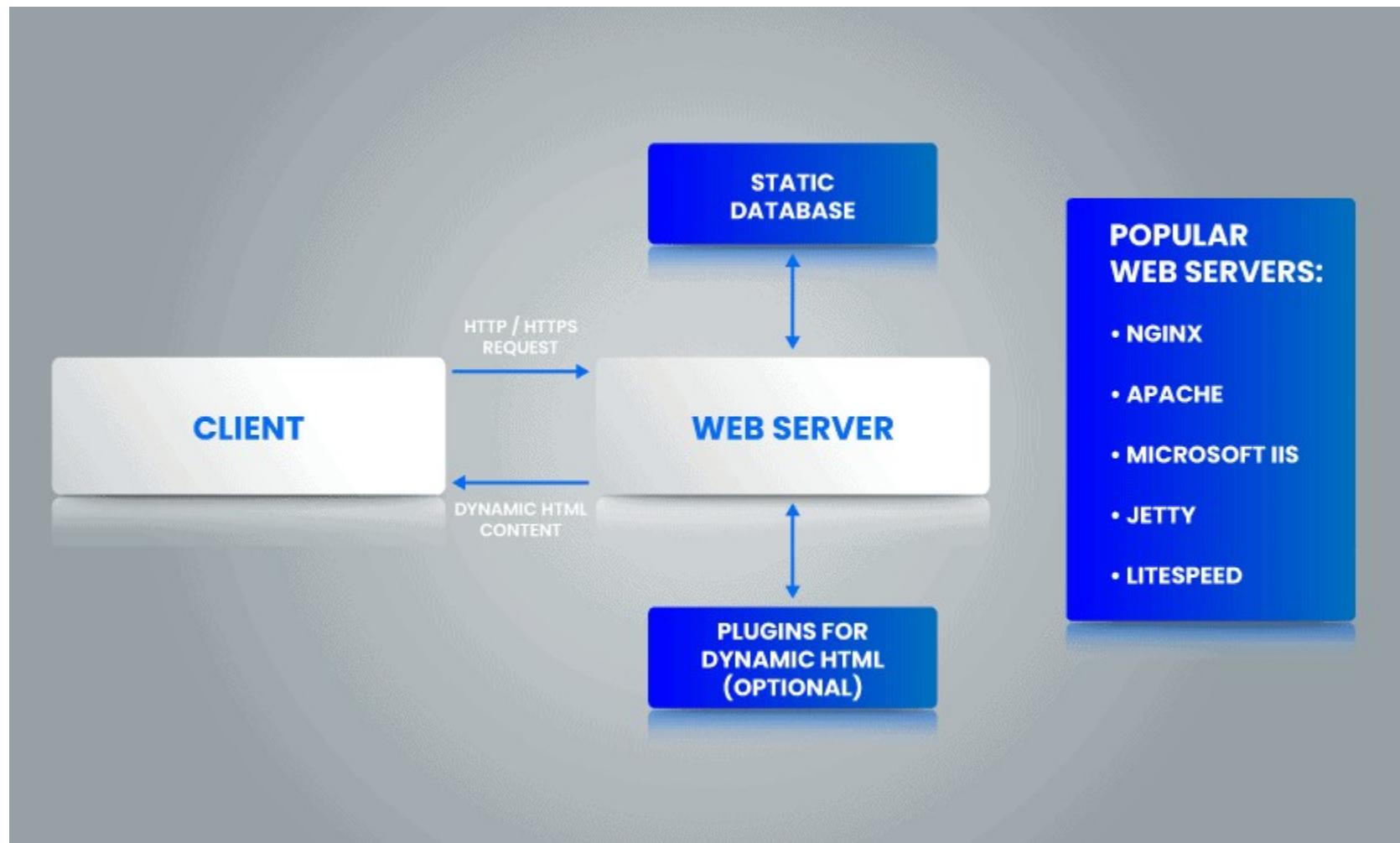
- At the end of the program
 - any SYMTAB entries that are still marked with * indicate undefined symbols
 - search SYMTAB for the symbol named in the END statement and jump to this location to begin execution
- The actual starting address must be specified at assembly time

Producing Object Code

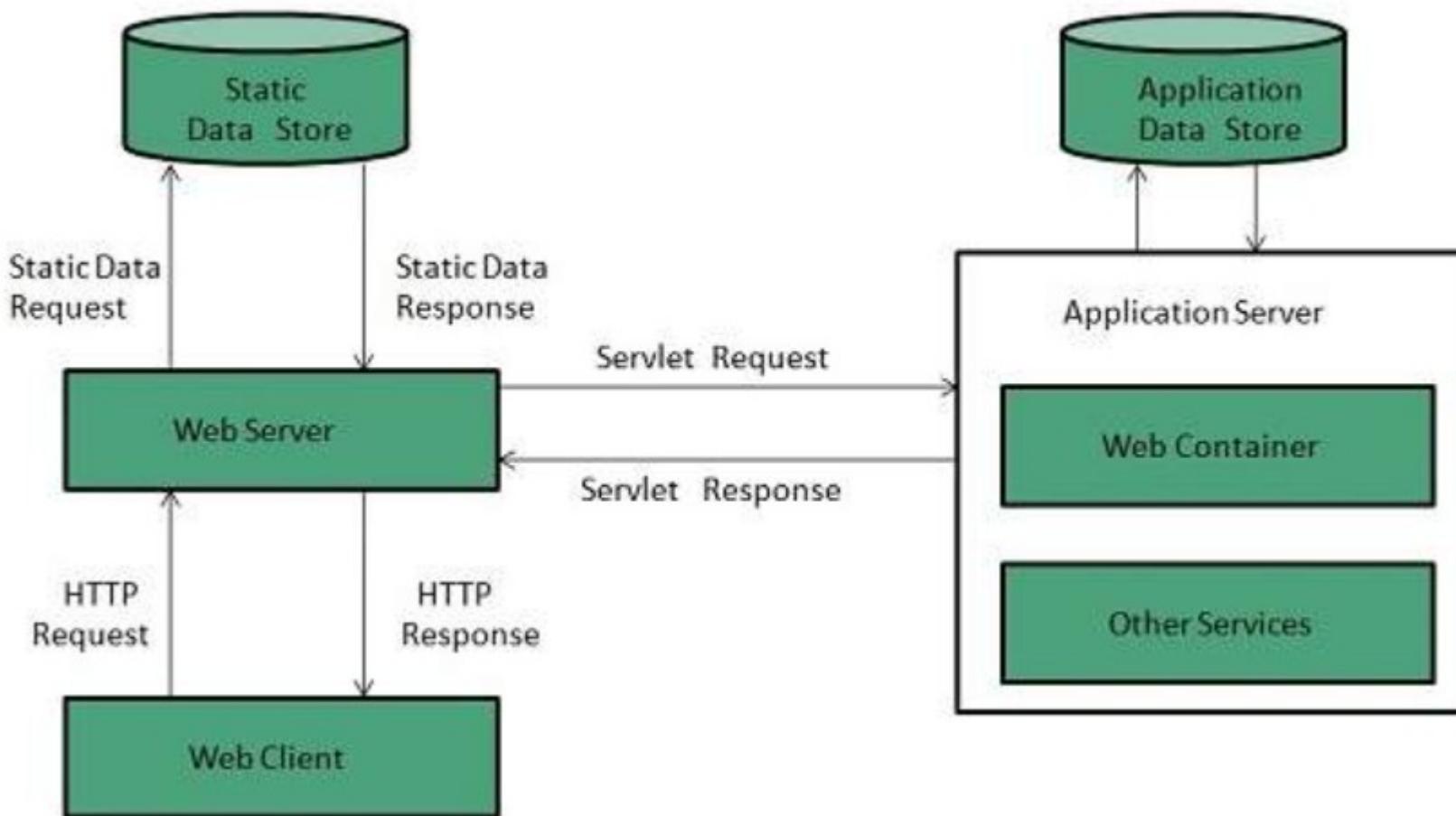
- When external working-storage devices are not available or too slow (for the intermediate file between the two passes)
- Solution:
 - When definition of a symbol is encountered, the assembler must generate another Tex record with the correct operand address
 - The loader is used to complete forward references that could not be handled by the assembler
 - The object program records must be kept in their original order when they are presented to the loader

Web Server

- Web servers are where web content is stored and kept available for users to access at any time.
- The basic objective is to **store, process and deliver web pages** to the users.
- Only delivers static content via HTML; A hypertext document that displays information on a browser.



Architecture of Web Server



Dynamic vs Static Web Servers

- **A Static Web Server** - consist of a computer and HTTP software, the sever will send hosted files as is to a browser.
- **Dynamic Web Server** - consist of a web server and other software such as an application server and database; the application server can be used to update any hosted files before they are sent to a browser.

Top Web Server Software on the Market

- Apache HTTP Server
- Microsoft Internet Information Services (IIS)
- Nginx
- Lighttpd
- Sun Java System Web Server

Challenges in Webserver Design

- User Interface And User Experience
- Scalability
- Performance
- Knowledge of Framework and Platforms
- Security

Web Server Security Practices

- A reverse proxy is designed to hide an internal server and act as an intermediary for traffic originating on an internal server.
- Access restriction through processes like limiting the web host's access to infrastructure machines or using Secure Socket Shell (SSH).
- Keeping web servers patched and up to date to help ensure the web server isn't susceptible to vulnerabilities.
- Network monitoring to make sure there isn't any or unauthorized activity.
- Using a firewall and SSL as firewalls can monitor HTTP traffic while having a Secure Sockets Layer (SSL) can help keep data secure.

- System calls related to process
 - fork
 - wait
 - exec
 - exit
 - signal
 - kill
 - raise

Process summary

- Process is a program in execution
 - Program is passive entity and Process is active entity
- In multi programming / multi tasking environment number of processes can reside in the system
- At any instance of time processes can be in different state such as Ready, Blocked, Running etc. and processes move from one state to another state depending upon certain conditions
- Processes use available system resources
- OS is responsible for managing all the processes in the system

Process related system calls – fork

- `pid_t fork(void)`
 - Duplicates the current process, creating a new entry in the process table with many of the same attributes as the parent process.
 - The new process is almost identical to the original, executing the same code but with its own data space, environment and file descriptor.
 - Uses copy on write technique
 - Kernel internally invokes `do_fork()` function (in `fork.c`)
 - `do_fork(SIGCHLD, regs.esp, ®s, 0);`

Process related system calls –

fork

- The fork () system call does the following in a UNIX system
 - Allocate slot in the process table for the new process
 - Assigns a unique process id to the new process
 - Make a copy of the process image of the parent, with the exception of shared memory
 - It increases counters for any files owned by the parent, to reflect that an additional process now also owns these files.
 - It assigns the child process to a ready to run state
 - It returns the ID number of the child to the parent process and a 0 value to the child process.

Process related system calls – fork

- All these works is done in Kernel space of parent process.
- After completing these functions, OS will do the following operations as a part of dispatcher routine
- Control returns to the user mode at the point of the fork call of the parent.
- Transfer control to the child process. The child process begins executing at the same point in the code as the parent, namely at the return from the fork call.
- Transfer control to another process. Both child and parent are left in the ready to run state.
- If fork system call fails, the return value to parent (no child will be created) will be -1.

Process Creation Example

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
int main(){
    pid_t pid;
    printf("fork program starting
    \n");
    pid = fork();
    if (pid < 0 ){
        perror("fork failed\n");
        exit(1);
    }
    else if (pid == 0){
        printf(" This is from child
        process My PID is %d and my
        Parent PID is %d\n",
        getpid(),getppid());
    }
    else {
        printf("This is from parent
        process My PID is %d and my
        Child's PID is %d\n",
        getpid(),pid);
    }
    return 0;
}
```

Calling fork()

Calling fork()

Consider the following example in which we have used the fork() system call to create a new child process:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    fork();
    printf("Using fork() system call\n");
    return 0;
}
```

OUTPUT:

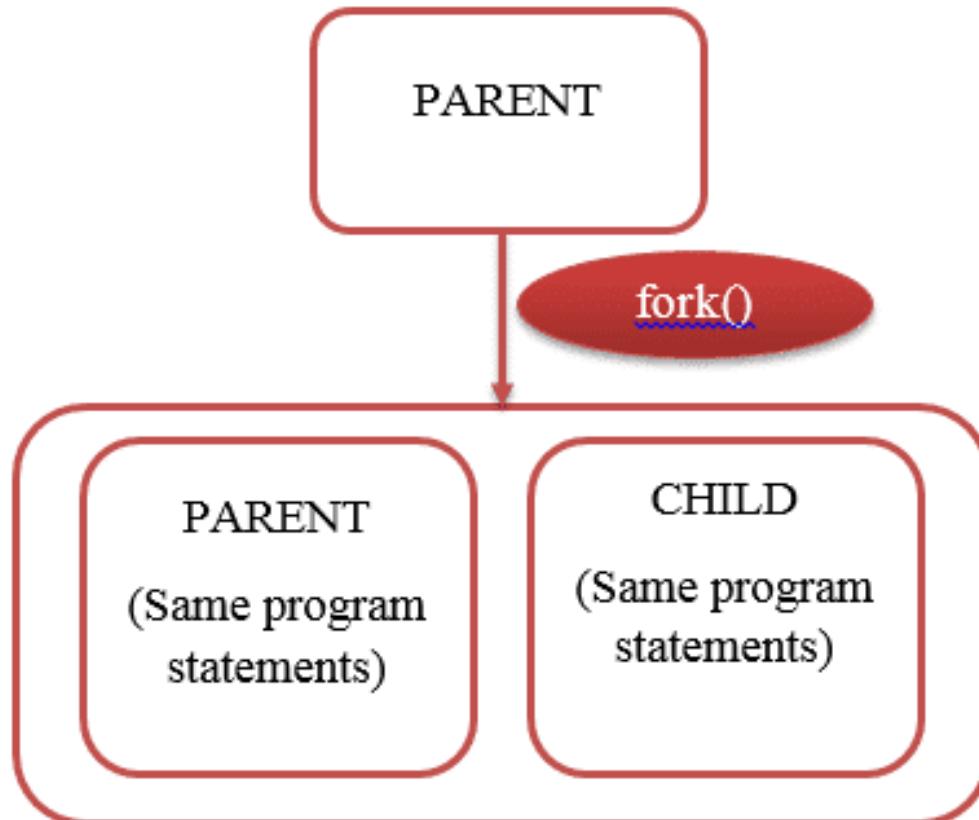
```
sysads@linuxhint $ gcc fork.c -o fork
sysads@linuxhint $ ./fork
```

Using fork() system call

Using fork() system call

In this program, we have used fork(), this will create a new child process. When the child process is created, both the parent process and the child process will point to the next instruction (same Program Counter).

That is 2^n times, where n is the number of fork() system calls.



Calling fork()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    fork();
    fork();
    fork();
    fork();
    printf("Using fork() system call\n");
    return 0;
}
```

Output:

```
sysads@linuxhint $ gcc fork.c -o fork
sysads@linuxhint $ ./fork
```

Now the total number of process created are $2^4 = 16$ and we have our print statement executed 16 times.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    pid_t p;
    p = fork();
    if(p== -1)
    {
        printf("There is an error while calling
fork()\n");
    }
    if(p== 0)
    {
        printf("We are in the child process\n");
    }
    else
    {
        printf("We are in the parent process\
");
    }
    return 0;
}
```

Calling fork()

OUTPUT:

sysads@linuxhint \$ gcc fork.c -o
fork

sysads@linuxhint \$./fork
We are in the parent process
We are in the child process

Calling fork()

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
//main function begins
int main(){
    pid_t p= fork(); //calling of fork
    system call
    if(p==-1)
        printf("Error occurred while calling
fork()");
    else if(p==0)
        printf("This is the child process");
    else
        printf("This is the parent process");
    return 0;
}
```

Output:
This is the parent process

This is the child process



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN SESSION 15

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Dr Thangakumar J



BITS Pilani
Pilani Campus

Operating Systems – Abstraction vs. Realization

Last Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Inheritance and Delegation; Design with inheritance and delegation; Comparison: Program Composition – Object Hierarchy vs. Class Hierarchy• OO Design and Design Patterns• Design of Complex Software (Compilers/ OS/ Web Server as an example).	R4

OS: Abstraction vs. Realization

Today's
topic

Operating System Structure

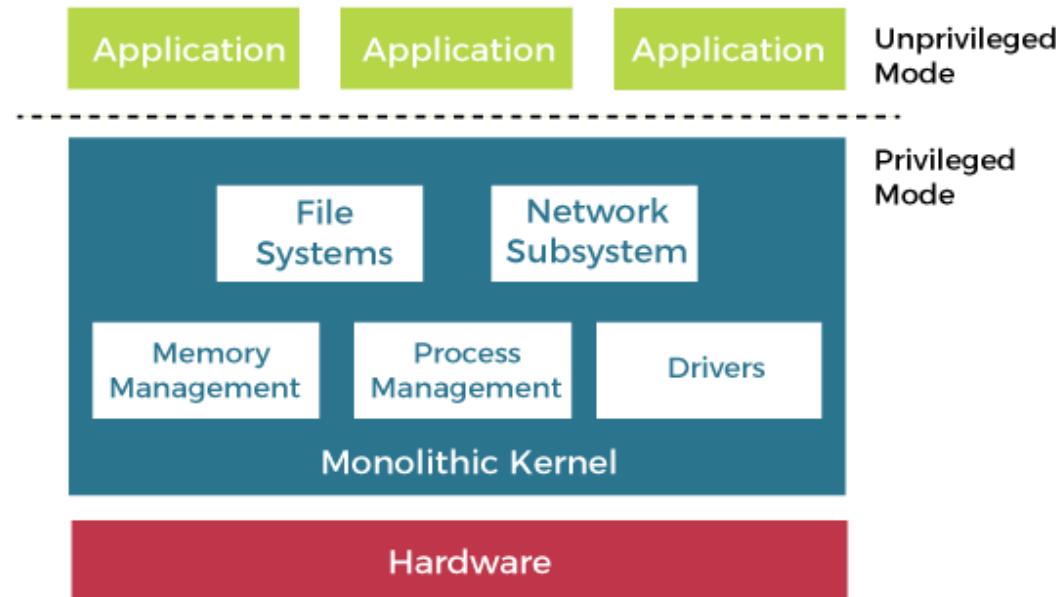
- Monolithic Systems
- Layered Systems
- Microkernels
- Client Server Model
- Virtual Machines

Operating System Structure

- **The six designs are:**
 - Monolithic systems,
 - Layered systems,
 - Microkernels,
 - Client-server systems,
 - Virtual machines, and
 - Exokernels

Monolithic Systems

Monolithic Kernel System



Monolithic Systems

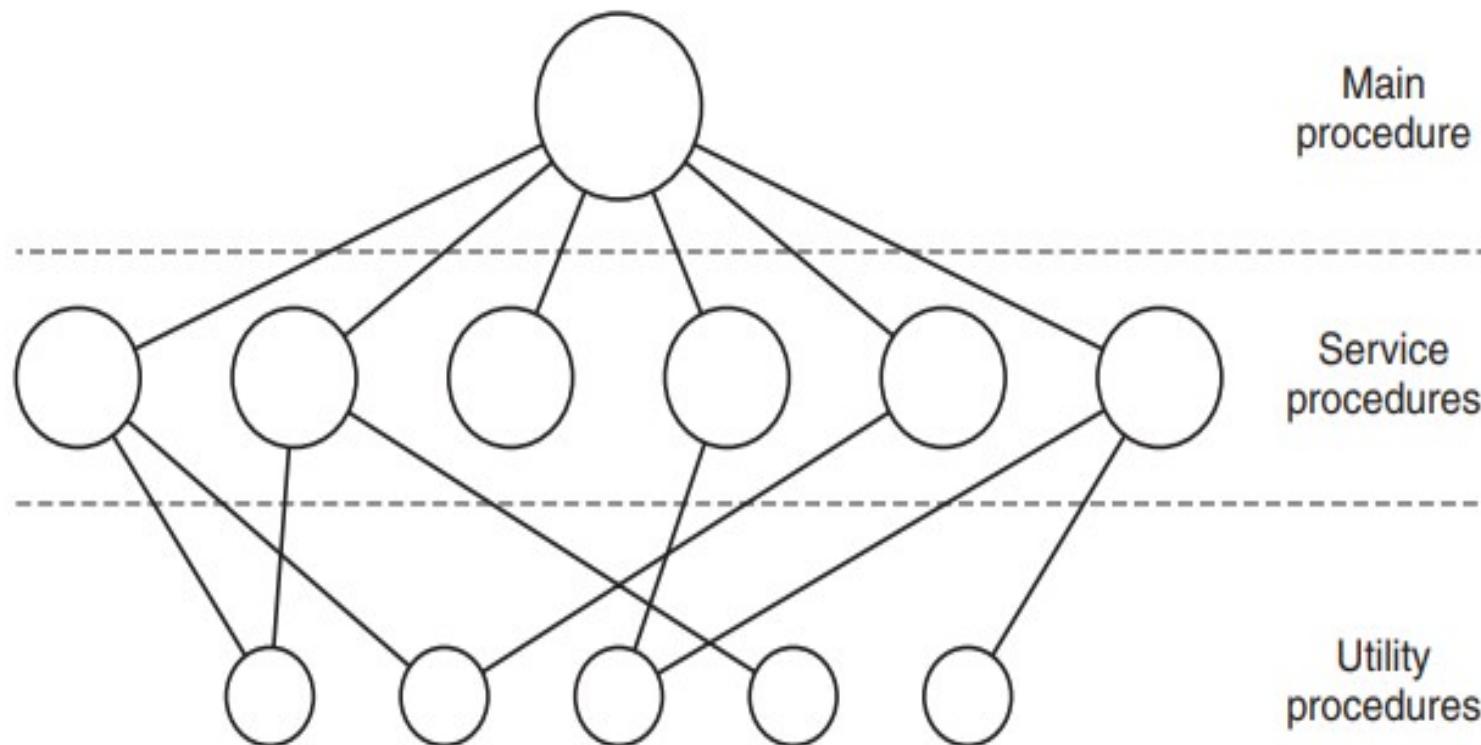
- The entire OS runs as a **single program** in kernel mode
- The OS - a collection of procedures, linked together into a single large executable binary program
- Each procedure in the system is free to call any other one; **efficient** but leads to a system **unwieldy and difficult** to understand
- A crash in any of these procedures will take down the entire OS

- It is possible to have some structure.
- The services (system calls) provided by the OS are requested by putting the parameters in a well-defined place (e.g., on the stack) and then executing a trap instruction.
- This instruction switches the machine from user mode to kernel mode and transfers control to the OS as in figure.
- The OS then fetches the parameters and determines which system call is to be carried out.
- It indexes into a table that contains in slot k a pointer to the procedure that carries out system call k .

A basic structure for the operating system:

- 1. A main program that invokes the requested service procedure.**
 - 2. A set of service procedures that carry out the system calls.**
 - 3. A set of utility procedures that help the service procedures.**
- For each system call, there is one service procedure that takes care of it and executes it.
 - The utility procedures do things that are needed by several service procedures, such as fetching data from user programs.
 - This division of the procedures into **three layers** is shown in figure.

A Simple Structuring Model for a Monolithic System



- System calls related to process
 - fork
 - wait
 - exec
 - exit
 - signal
 - kill
 - raise

Process summary

- Process is a program in execution
 - Program is passive entity and Process is active entity
- In multi programming / multi tasking environment number of processes can reside in the system
- At any instance of time processes can be in different state such as Ready, Blocked, Running etc. and processes move from one state to another state depending upon certain conditions
- Processes use available system resources
- OS is responsible for managing all the processes in the system

Process related system calls – fork

- `pid_t fork(void)`
 - Duplicates the current process, creating a new entry in the process table with many of the same attributes as the parent process.
 - The new process is almost identical to the original, executing the same code but with its own data space, environment and file descriptor.
 - Uses copy on write technique
 - Kernel internally invokes `do_fork()` function (in `fork.c`)
 - `do_fork(SIGCHLD, regs.esp, ®s, 0);`

Process related system calls –

fork

- The fork () system call does the following in a UNIX system
 - Allocate slot in the process table for the new process
 - Assigns a unique process id to the new process
 - Make a copy of the process image of the parent, with the exception of shared memory
 - It increases counters for any files owned by the parent, to reflect that an additional process now also owns these files.
 - It assigns the child process to a ready to run state
 - It returns the ID number of the child to the parent process and a 0 value to the child process.

Process related system calls – fork

- All these works is done in Kernel space of parent process.
- After completing these functions, OS will do the following operations as a part of dispatcher routine
- Control returns to the user mode at the point of the fork call of the parent.
- Transfer control to the child process. The child process begins executing at the same point in the code as the parent, namely at the return from the fork call.
- Transfer control to another process. Both child and parent are left in the ready to run state.
- If fork system call fails, the return value to parent (no child will be created) will be -1.

Process Creation Example

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
int main(){
    pid_t pid;
    printf("fork program starting
    \n");
    pid = fork();
    if (pid < 0 ){
        perror("fork failed\n");
        exit(1);
    }
    else if (pid == 0){
        printf(" This is from child
        process My PID is %d and my
        Parent PID is %d\n",
        getpid(),getppid());
    }
    else {
        printf("This is from parent
        process My PID is %d and my
        Child's PID is %d\n",
        getpid(),pid);
    }
    return 0;
}
```

Calling fork()

Calling fork()

Consider the following example in which we have used the fork() system call to create a new child process:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    fork();
    printf("Using fork() system call\n");
    return 0;
}
```

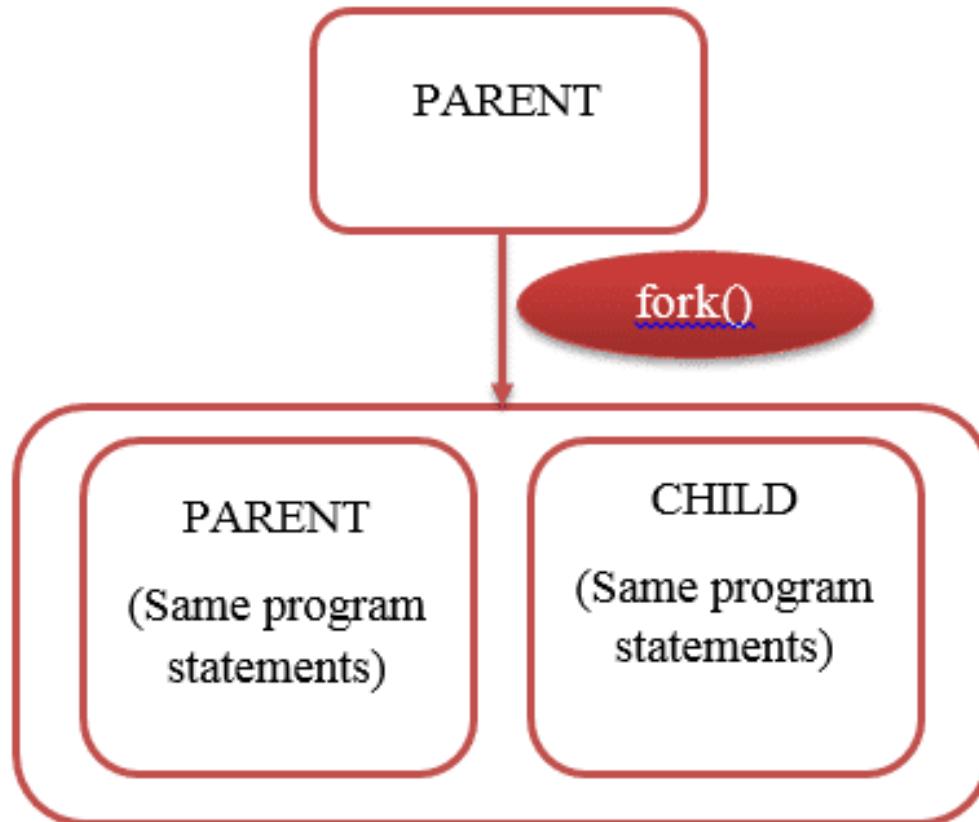
OUTPUT:

```
sysads@linuxhint $ gcc fork.c -o fork
sysads@linuxhint $ ./fork
```

Using fork() system call

Using fork() system call

In this program, we have used fork(), this will create a new child process. When the child process is created, both the parent process and the child process will point to the next instruction (same Program Counter). That is 2^n times, where n is the number of fork() system calls.



Calling fork()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    fork();
    fork();
    fork();
    fork();
    printf("Using fork() system call\n");
    return 0;
}
```

Output:

```
sysads@linuxhint $ gcc fork.c -o fork
sysads@linuxhint $ ./fork
```

Now the total number of process created are $2^4 = 16$ and we have our print statement executed 16 times.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    pid_t p;
    p = fork();
    if(p== -1)
    {
        printf("There is an error while calling
fork()\n");
    }
    if(p== 0)
    {
        printf("We are in the child process\n");
    }
    else
    {
        printf("We are in the parent process\
");
    }
    return 0;
}
```

Calling fork()

OUTPUT:

sysads@linuxhint \$ gcc fork.c -o
fork

sysads@linuxhint \$./fork
We are in the parent process
We are in the child process

Calling fork()

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
//main function begins
int main(){
    pid_t p= fork(); //calling of fork
    system call
    if(p==-1)
        printf("Error occurred while calling
fork()");
    else if(p==0)
        printf("This is the child process");
    else
        printf("This is the parent process");
    return 0;
}
```

Output:
This is the parent process

This is the child process

Advantages

1. The monolithic kernel runs quickly because of memory management, file management, process scheduling, etc. These are all implemented in the same address space.
2. Its structures are easy and simple. The kernel contains all of the components required for processing.
3. All of the components may interact directly with each other's and also with the kernel.
4. It works better for performing smaller tasks because it may handle limited resources.

Disadvantages

5. Monolithic OS has more tendency to generate errors and bugs. It's because user programs use the same address spaces as the kernel.
6. It isn't easy to port code written in the monolithic operating system.
7. It is very difficult to add and remove features from a monolithic operating system. All of the code must be modified and recompiled to add or remove a feature.

Layered Systems

- THE system built at the Technische Hogeschool Eindhoven in the Netherlands by E. W. Dijkstra (1968) and his students.
- The system was a simple batch system for a Dutch computer, the Electrologica X8, which had 32K of 27-bit words (bits were expensive back then).
- The system had **six layers**, as shown in figure.
- **Layer 0** - allocation of the processor, switching between processes when interrupts occurred or timers expired
- **Above layer 0** - sequential processes; be programmed without having to worry about the fact that multiple processes were running on a single processor.

Recap - Structure of the operating system

Layer	Function
5	The operator
4	User programs
3	Input/Output Management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

- **Layer 1** - memory management
- It allocated space for processes in main memory and on a 512K word drum used for holding parts of processes (pages) for which there was no room in main memory
- **Above layer 1** - processes did not have to worry about whether they were in memory or on the drum;
- **The layer 1 software** - pages were brought into memory at the moment they were needed and removed when they were not needed.
- **Layer 2** - communication between each process and the operator console; each process effectively had its own operator console.

- **Layer 3** - managing the I/O devices and buffering the information streams to and from them.
- **Above layer 3** - each process deal with abstract I/O devices with nice properties.
- **Layer 4** - the user programs were found.
- They did not have to worry about process, memory, console, or I/O management.
- The **system operator process** was located in **layer 5**.
- **MULTICS** - a series of concentric rings, with the inner ones being more privileged than the outer ones

- When a procedure in an outer ring wanted to call a procedure in an inner ring, it had to make the equivalent of a system call, i.e., a **TRAP instruction** whose parameters were carefully checked for validity before the call was allowed to proceed.
- Although the entire OS was part of the address space of each user process in MULTICS, the hardware made it possible to designate individual procedures (memory segments, actually) as protected against reading, writing, or executing.
- The **THE layering scheme** - a design aid.
- **Advantage of the ring mechanism** - easily be extended to structure user subsystems

Advantages

1. Easy Debugging

It is very simple to debug because the layers are discrete. If an error happens in the CPU scheduling layer, the developer may only debug that layer.

2. Modularity

This design supports modularity because each layer only executes tasks it is scheduled to perform.

3. Abstraction

Each layer is concerned with its own set of functions. As a result, the functions and implementations of the other layer are abstract to it.

4. Easy update

A modification in one layer does not affect the other layers.

Disadvantages

1. Complex and better implementation

Layer layout is important because a layer can utilize the services of the layers below it. For example, the backup storage layer uses the memory management layer's services, so it must be stored beneath the memory management layer.

2. Slower in execution

When one layer wishes to interact with another, it sends a request that must traverse all layers between the two layers to be fulfilled. It enhances response time, which is faster than the Monolithic system. As a result, increasing the number of layers may lead to a very inefficient design.

Summary

MONOLITHIC OS VERSUS LAYERED OS

MONOLITHIC OS

An OS architecture in which the entire operating system works in the kernel space

There are three main layers

LAYERED OS

An OS architecture that is divided into a number of layers, each layer performing a specific functionality

There are more layers

Visit www.PEDIAA.com

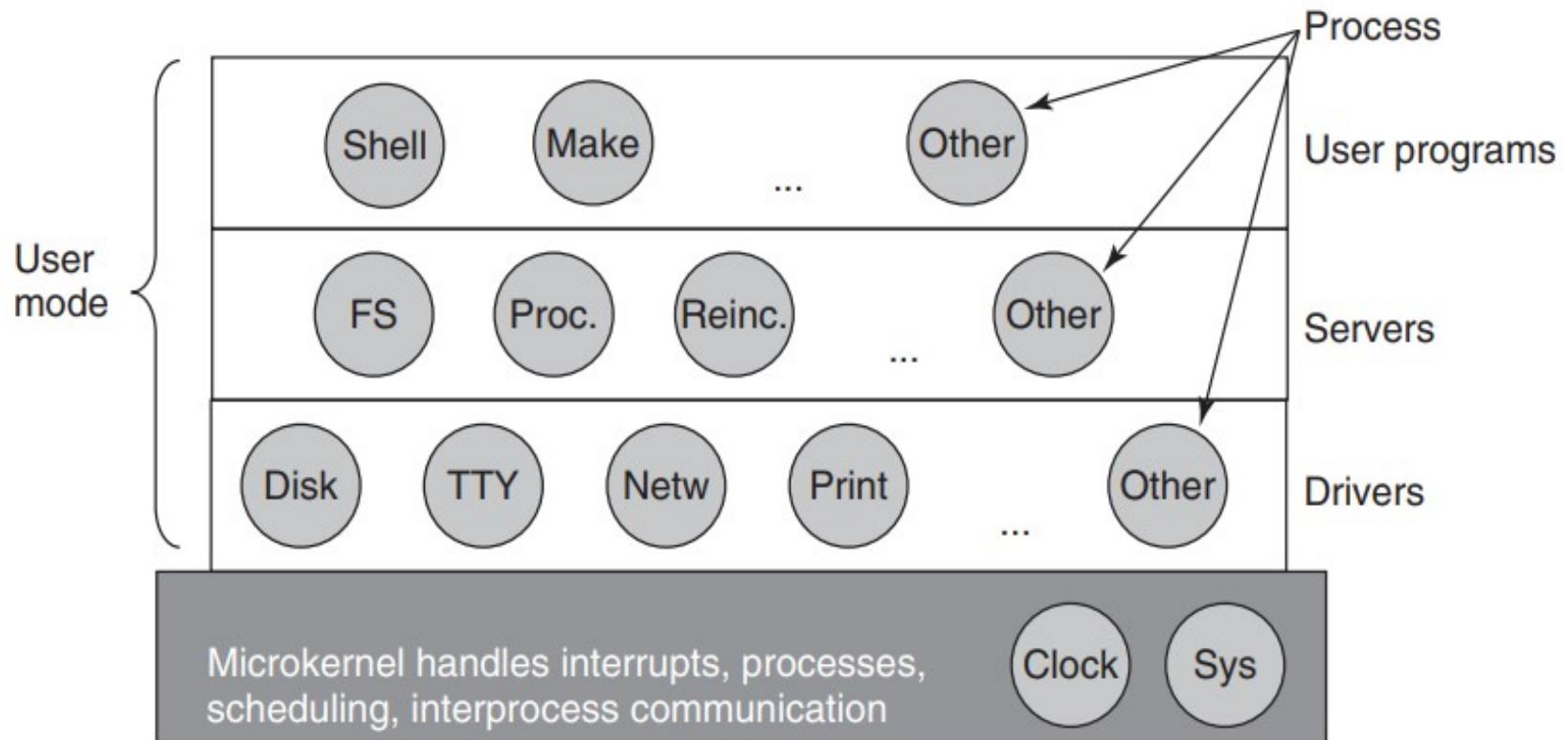
Microkernels

- Various researchers have repeatedly studied the number of bugs per 1000 lines of code
- **Bug density** - module size, module age, and more; but a ballpark figure for serious industrial systems is between two and ten bugs per thousand lines of code.
- A **monolithic OS** of five million lines of code is likely to contain between **10,000 and 50,000 kernel bugs**
- **Microkernel design** - to achieve **high reliability** by splitting the OS into small, well-defined modules; **runs in kernel mode** and the rest run as relatively powerless ordinary user processes.

- By running each device driver and file system as a separate user process, a bug in one of these can crash that component, but cannot crash the entire system.
- A bug in the audio driver will cause the sound to be garbled or stop, but will not crash the computer.
- With the exception of OS X, which is based on the Mach microkernel, common desktop OS do not use microkernels.
- They are dominant in real-time, industrial, avionics, and military applications that are mission critical and have very high reliability requirements.

- The better-known microkernels include: **Integrity, K42, L4, PikeOS, QNX, Symbian, and MINIX 3**
- **MINIX 3** - the idea of modularity to the limit, breaking most of the operating system up into a number of independent user-mode processes
- MINIX 3 is a **POSIX-conformant, open source system**.
- It is only about **12,000 lines of C** and some **1400 lines of assembler** for very low-level functions such as catching interrupts and switching processes.
- The **device driver for the clock is in the kernel** because the scheduler interacts closely with it.
- The other device drivers run as separate user processes.

Simplified Structure of the MINIX System



- Outside the kernel, the system is structured as three layers of processes all running in user mode.
- The **lowest layer** - the device drivers.
- They run in user mode, they do not have physical access to the I/O port space and cannot issue I/O commands directly.
- **To program an I/O device**, the driver builds a structure telling which values to write to which I/O ports and makes a kernel call telling the kernel to do the write.
- This approach means that the kernel can check to see that the driver is writing (or reading) from I/O it is authorized to use.
- A **buggy audio driver** cannot accidentally write on the disk.

- One or more file servers manage the file system(s), the process manager creates, destroys, and manages processes, and so on.
- User programs obtain OS services by sending short messages to the servers asking for **the POSIX system calls**.
- **Eg:** a process needing to do a read sends a message to one of the file servers telling it what to read.
- **The reincarnation server** - to check if the other servers and drivers are functioning correctly
- In the event that a faulty one is detected, it is automatically replaced without any user intervention.
- The system is **self-healing** and can **achieve high reliability**

- Drivers can touch only authorized I/O ports; but access to kernel calls - controlled on a per-process basis, as is the ability to send messages to other processes.
- Processes can grant **limited permission** for other processes to have the kernel access their address spaces.
- **A simple scheduling algorithm** - to assign a numerical priority to every process and then have the kernel run the highest-priority process that is runnable.
- The policy—assigning priorities to processes—can be done by user-mode processes.
- Policy and mechanism can be **decoupled** and the kernel can be made smaller.

Introduction – About Minix

- Minix – Mini Unix (Minix) basically, a UNIX - compatible operating system.
- Minix is small in size, with microkernel-based design.
- Minix has been kept (relatively) small and simple.
- Minix is small, it is nevertheless a preemptive, multitasking operating system.

POSIX (Portable Operating System Interface) is a set of standard operating system interfaces based on the Unix operating system.

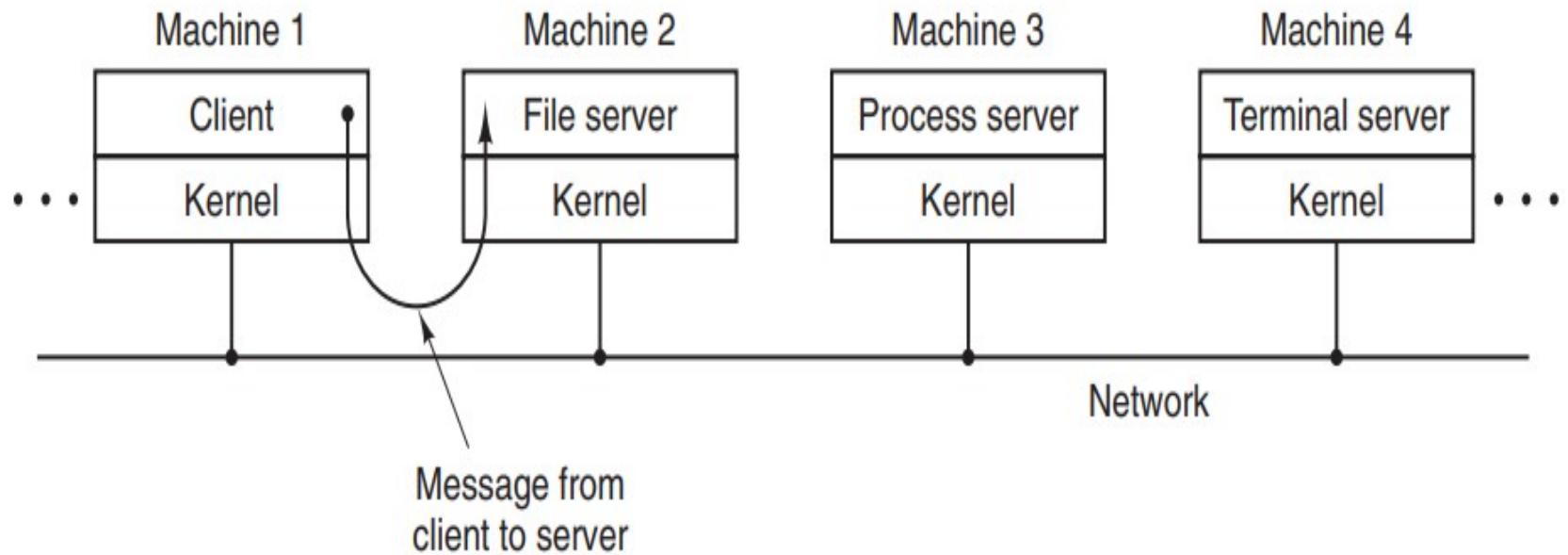
The need for Standardization arose because enterprises using computers wanted to be able to develop programs that could be moved among different manufacturer's computer systems without having to be recoded.

Unix was selected as the basis for a standard system interface partly because it was "manufacturer-neutral." However, several major versions of Unix existed so there was a need to develop a common denominator system.

Client-Server Model

- **Client-server model** - two classes of processes, the servers, each of which provides some service, and the clients, which use these services
- Lowest layer – **microkernel**
- The essence is the presence of client processes and server processes.
- Communication between clients and servers - **message passing**
- The client-server model is an abstraction that can be used for a single machine or for a network of machines.
- Many systems involve users at their home PCs as clients and large machines elsewhere running as servers; Web operates this way.

The Client-Server Model over a Network



Virtual Machines

- The initial releases of **OS/360** were strictly **batch systems**
- The **official IBM timesharing system**, **TSS/360**, so big and slow that few sites converted to it
- **The z/VM** - widely used on IBM's current mainframes
- **The zSeries** - heavily used in large corporate data centers
- For example, as **e-commerce servers** that handle hundreds or thousands of transactions per second and use databases whose sizes run to millions of gigabytes.

VM/370

- Originally called CP/CMS (Control Program/Cambridge Monitor System) and later renamed VM/370
- A timesharing system provides multiprogramming and an extended machine with a more convenient interface than the bare hardware.
- The essence of VM/370 is to completely separate these two functions.
- **Virtual machine monitor** - runs on the bare hardware and does the multiprogramming; but several virtual machines to the next layer up.
- They are exact copies of the bare hardware, including kernel/user mode, I/O, interrupts, and everything else the real machine has.

The Structure of VM/370 with CMS

- Different virtual machines can run different operating systems
- **Original IBM VM/370 system** – some ran **OS/360** or one of the other **large batch** or **transaction-processing operating systems**; others ran a single-user, interactive system called **CMS** (Conversational Monitor System) for interactive timesharing users
- **CMS program** executed a system call - trapped to the operating system in its own virtual machine - run on a real machine instead of a virtual one
- Issue the normal **hardware I/O instructions** for reading its virtual disk or whatever was needed to carry out the call

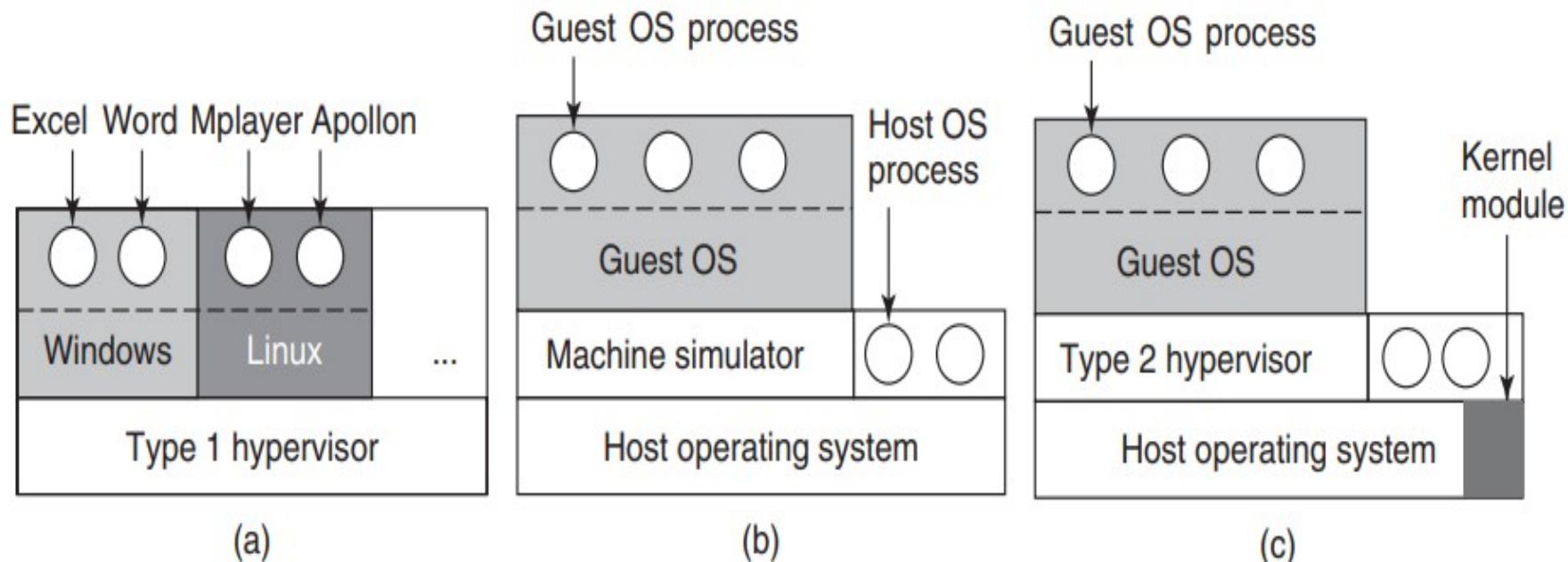
- **I/O instructions** were trapped by **VM/370**; then performed them as part of its simulation of the real hardware
- **z/VM** - used to run **multiple complete operating systems** rather than stripped-down single-user systems like CMS
- **For example**, the **zSeries** is capable of running one or more **Linux virtual machines** along with traditional IBM operating systems.

Virtual Machines Rediscovered

- **IBM, Oracle and Hewlett-Packard** - virtual-machine support to their high-end enterprise servers; virtualization has largely been ignored in the PC world until recently
- Many companies run their mail servers, Web servers, FTP servers, and other servers on separate computers, sometimes with different operating systems.
- **Virtualization**, a way to run them all on the same machine without having a crash of one server bring down the rest.

- **Virtualization** - popular in the Web hosting world
- Without virtualization - Web hosting customers are forced to choose between shared hosting and dedicated hosting
- End users who want to run two or more operating systems at the same time, say **Windows and Linux**, because some of their favourite application packages run on one and some run on the other.
- **Virtual machine monitor** - renamed type 1 hypervisor
- It requires more keystrokes than people are prepared to put up with now.

(a) A type 1 hypervisor (b) A pure type 2 hypervisor (c) A practical type 2 hypervisor



- The research papers - commercial products (e.g., VMware Workstation and Xen) and a revival of interest in virtual machines.
- Besides VMware and Xen, popular hypervisors today include KVM (for the Linux kernel), VirtualBox (by Oracle), and Hyper-V (by Microsoft).
- **Binary translation** - improve the resulting systems; to add a kernel module to do some of the heavy lifting
- **Type 2 hypervisors** - all commercially available hypervisors, such as VMware Workstation, use this hybrid strategy

- **Type 2** makes uses of a **host operating system and its file system** to create processes, store files, and so on
- A type 1 hypervisor has no underlying support and must perform all these functions itself.
- After a type 2 hypervisor is started, it **reads the installation CD-ROM** (or CDROM image file) for the chosen guest operating system and installs the guest OS on a virtual disk, which is just a big file in the host operating system's file system.
- Type 1 hypervisors cannot do this because there is **no host operating system to store files on**. They must manage their own storage on a raw disk partition.
- **Paravirtualization** - a different approach to handling control instructions is to modify the operating system to remove them

The Java Virtual Machine

- When Sun Microsystems invented the Java programming language, it also invented a virtual machine (i.e., a computer architecture) called the JVM (Java Virtual Machine).
- The Java compiler produces code, is executed by a software JVM interpreter
- **Advantage** - the JVM code can be shipped over the Internet to any computer that has a JVM interpreter and run there
- If the interpreter is implemented properly, which is not completely trivial, incoming JVM programs can be checked for safety and then executed in a protected environment so they cannot steal data or do any damage

Exokernels

- Giving each user a subset of the resources
- One virtual machine might get **disk blocks 0 to 1023**, the next one might get blocks **1024 to 2047**, and so on.
- **Exokernel** - a program running in kernel mode at the bottom layer
- Allocate **resources to virtual machines** and then check attempts to use them to make sure no machine is trying to use somebody else's resources
- Each user-level virtual machine can run its own OS, as on **VM/370** and the **Pentium virtual 8086s**, except that each one is restricted to using only the resources it has asked for and been allocated.

Advantage of the Exokernel Scheme

- It saves a layer of mapping
- In the other designs, each virtual machine thinks it has its own disk, with blocks running from 0 to some maximum, so the virtual machine monitor must maintain tables to remap disk addresses (and all other resources).
- With the exokernel, this remapping is not needed.
- The exokernel need only keep track of which virtual machine has been assigned which resource.
- Separating the multiprogramming (in the exokernel) from the user operating system code (in user space) but with less overhead



BITS Pilani
Pilani Campus



THANK YOU



BITS Pilani
Pilani Campus

COMPUTING AND DESIGN SESSION 16

Course design by - Dr. Lucy J. Gudino &
Dr Chandrasekhar
Faculty - Thangakumar J



BITS Pilani
Pilani Campus

Operating Systems – Abstraction vs. Realization (Cont..)



Last Session

List of Topic Title	Text/Ref Book/external resource
Operating System Structure <ul style="list-style-type: none">• Monolithic Systems• Layered Systems• Microkernels• Client Server Model• Virtual Machines	R1 (1.7)

OS: Abstraction vs. Realization

(Cont..)

Today's
topic

- Principle of Virtualization
- Benefits and features
- Building Blocks
- Types of Virtual Machines and Implementations
- Virtualization and Operating System Components

Recap of concepts

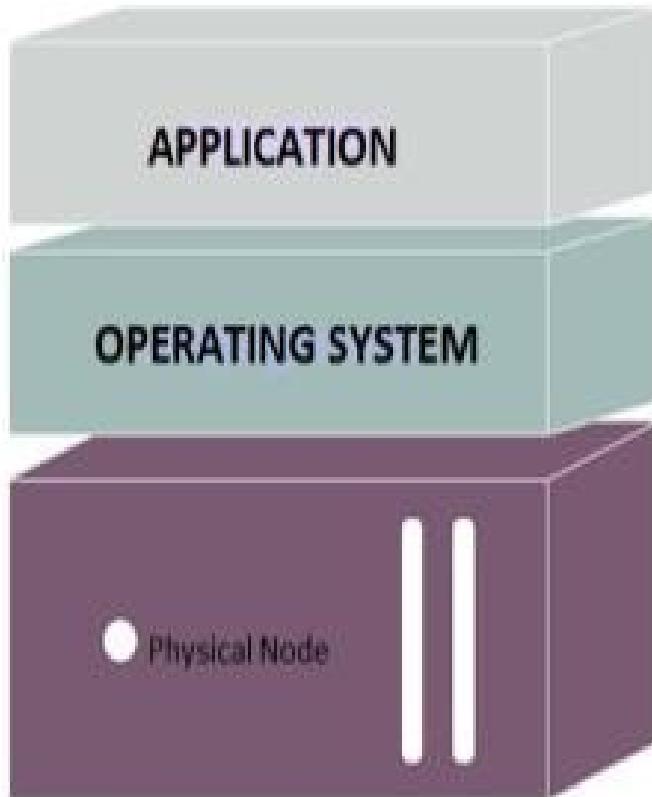
A virtual machine is a **computer file, typically called an image, that behaves like an actual computer**. It can run in a window as a separate computing environment, often to run a different operating system—or even to function as the user's entire computer experience—as is common on many people's work computers.

Example

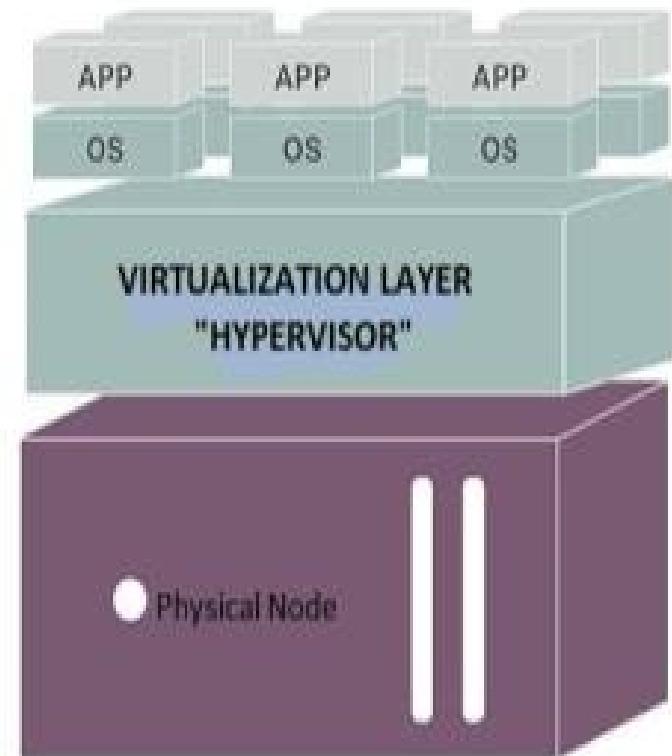
A process virtual machine **allows you to run a single process as an application on a host machine**. An example of a process virtual machine is the Java Virtual Machine (JVM) which allows any system to run Java applications as if they were native to the system.

VIRTUALIZATION

- The provision of a set of computing resources or their logical combination, abstracted from the hardware while providing logical isolation of computing processes running on the same physical resource.
- This includes operating systems, hardware platforms (computers and servers), storage arrays (file storage), and computer network resources.
- Virtualization is realized with the help of software – a **hypervisor**.
- It manages the physical resources of a computing machine and distributes those resources among several different operating systems, allowing them to run simultaneously.



TRADITIONAL ARCHITECTURE

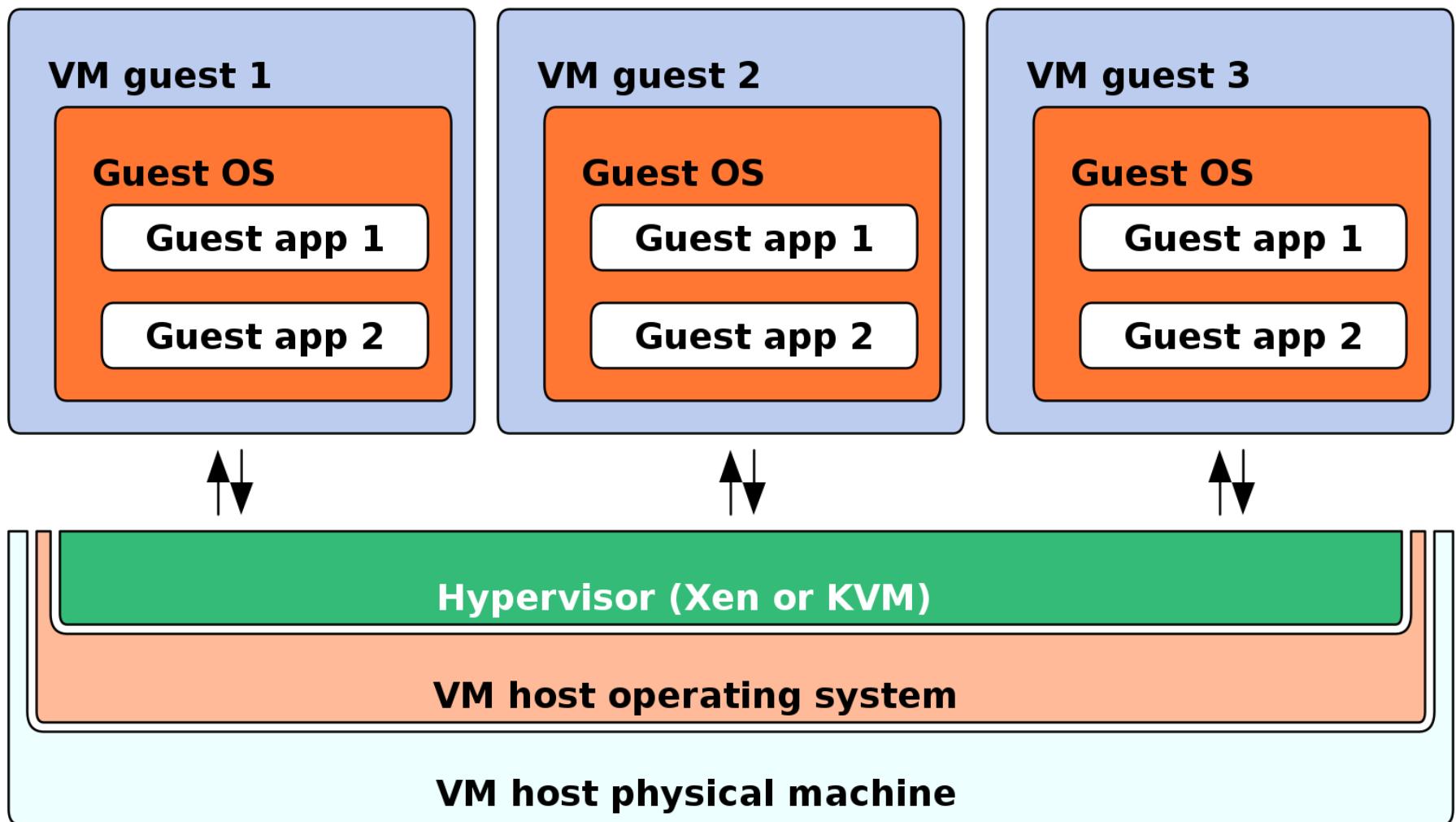


VIRTUAL ARCHITECTURE

- The hypervisor creates multiple copies, clones of its hardware resources, out of a single physical computer.
- A guest operating system can be installed on each virtual machine, not tied to the host hardware.
- A hypervisor allows **virtualizing desktops, applications, storage systems, and networks.**
- It can be either installed directly on the hardware or operate as a layer installed in the operating system between the hardware and the virtual machine.
 - **A virtualization host** is a physical server with a hypervisor running on it.
 - **A host system** is the physical machine OS that has the installed hypervisor.
 - **A guest operating system** is an OS running inside the virtual machine.
 - **A virtual machine** is a virtual copy of real hardware, an application that functions as a physical computer or server.

PRINCIPLE OF VIRTUALIZATION

- A technique of how to separate a service from the underlying physical delivery of that service.
- The process of **creating a virtual version** of something like computer hardware.
- **Multiple OS and applications** can run on same machine and its same hardware at the same time, **increasing the utilization** and **flexibility** of hardware.
- It allows to **share** a single physical instance of a resource or an application among multiple customers and organizations at one time.



Types of Virtualization

- **Application Virtualization:** helps a user to have remote access of an application from a server. The server stores all personal information and other characteristics of the application but can still run on a local workstation through the internet.
- **Network Virtualization:** provides a facility to create and provision virtual networks—logical switches, routers, firewalls, load balancer, Virtual Private Network (VPN), and workload security within days or even in weeks.
- **Desktop Virtualization:** allows the users' OS to be remotely stored on a server in the data centre. The user to access their desktop virtually, from any location by a different machine.

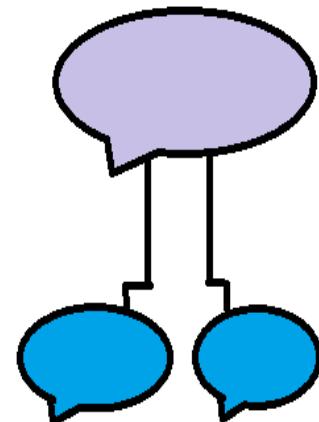
TYPES OF VIRTUALIZATION



DESKTOP
VIRTUALIZATION



APPLICATION
VIRTUALIZATION



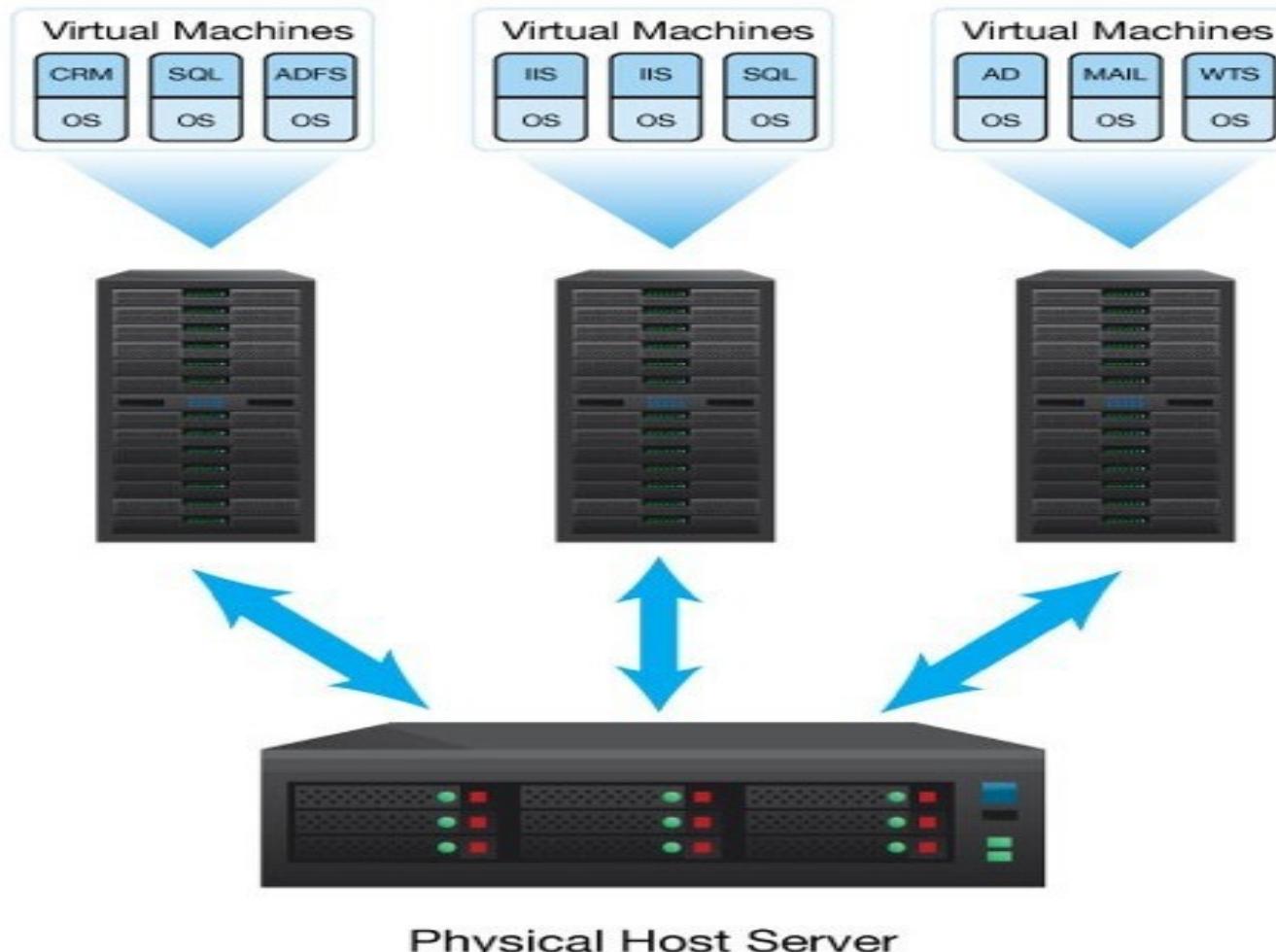
NETWORK
VIRTUALIZATION



STORAGE
VIRTUALIZATION

- **Storage Virtualization:** an array of servers that are managed by a virtual storage system
- **Server Virtualization:** the central-server(physical server) is divided into multiple different virtual servers by changing the identity number, processors. So, each system can operate its own operating systems in isolate manner.
- **Data virtualization:** the data is collected from various sources and managed that at a single place without knowing more about the technical information. big giant companies are providing their services like **Oracle**, **IBM**, **At scale**, **Cdata**, etc.

Server Virtual Machine



BENEFITS OF VIRTUALIZATION

1. **More flexible and efficient allocation of resources.**
2. **Enhance development productivity.**
3. **It lowers the cost of IT infrastructure.**
4. **Remote access and rapid scalability.**
5. **High availability and disaster recovery.**
6. **Pay peruse of the IT infrastructure on demand.**
7. **Enables running multiple operating systems.**

Virtual Machines

- It is a fake computer system operating on hardware.
- It partially uses the hardware of your system (like CPU, RAM, disk space, etc.) but **its space is completely separated from main system.**
- **Virtual Machine Language:** can be understood by different operating systems; **platform-independent**
- To use code that can be executed on different types of operating systems like (Windows, Linux, etc), the virtual machine language will be helpful.

Virtual Machine Components

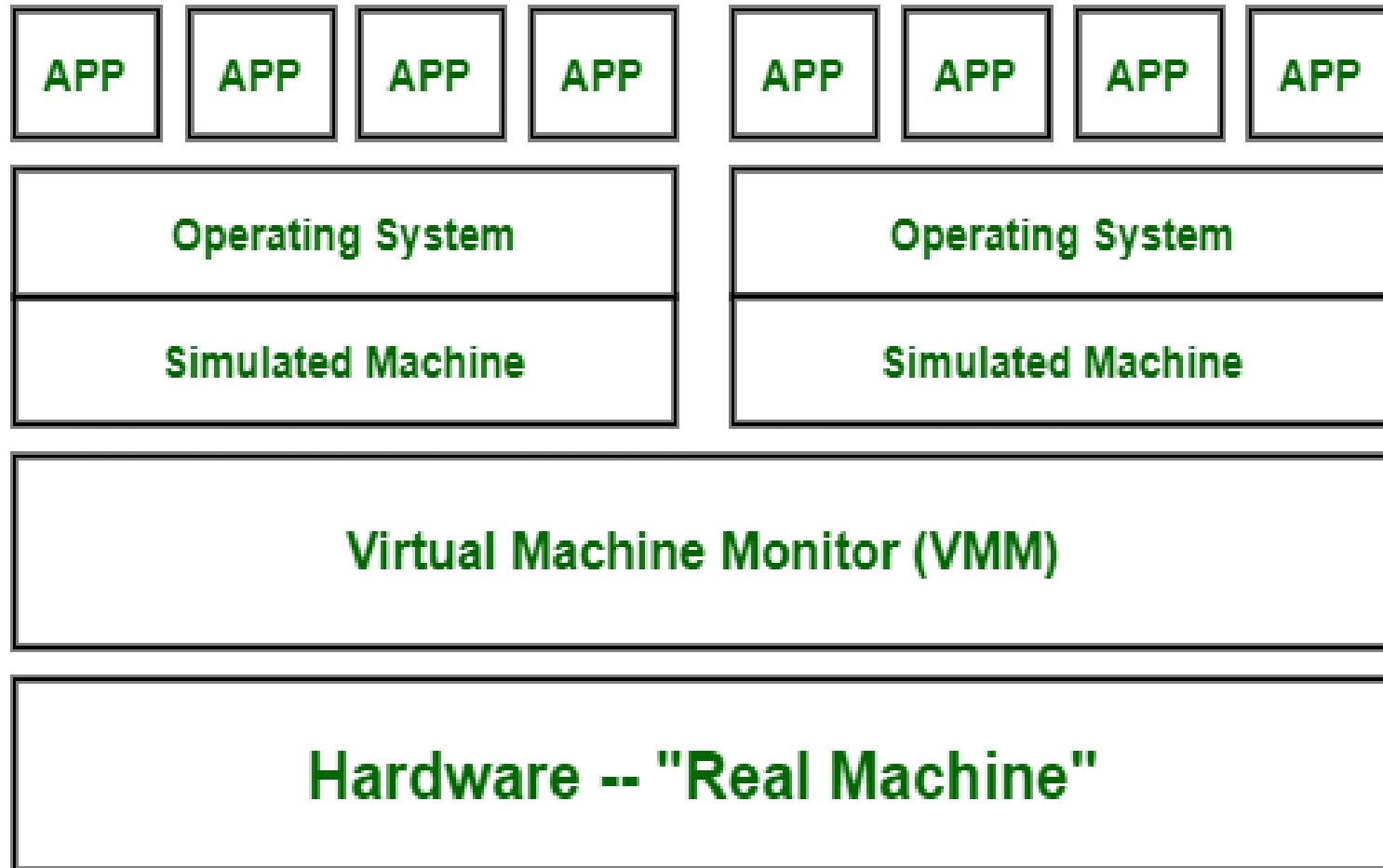
- **Operating System**
- **VMware Tools**
- **Compatibility Setting**
- **Hardware Devices**

TYPES OF VIRTUAL MACHINES AND IMPLEMENTATIONS

1. System Virtual Machine

- It gives us **complete system platform** and gives the execution of the complete virtual operating system.
- Provides an environment for an OS to be installed completely.

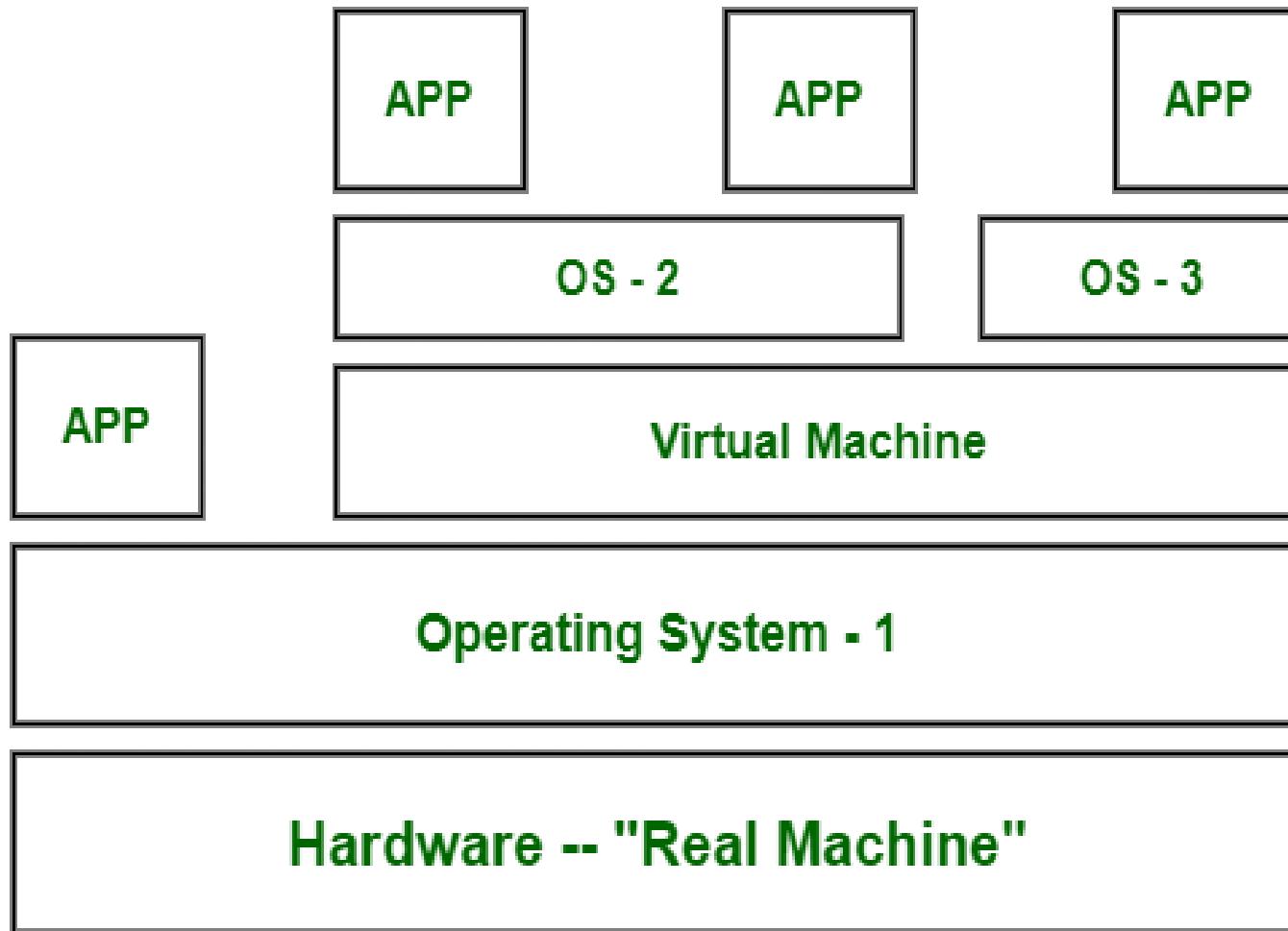
System Virtual Machine



2. Process Virtual Machine

- It does not provide with the facility to install the virtual operating system completely.
- It creates **virtual environment** of OS while using some app or program and this environment will be destroyed as soon as we exit from that app.

Process Virtual Machine



How are Virtual Machines Implemented?

- Virtual Machines are written in portable and efficient programming language (C or C++).
- For performance critical components, assembly language is used.
- Some virtual machines (Lisp, Forth, Smalltalk) are largely written in language itself.
- Most Java VM implementations consists of a mixture of C/C++ and assembly code.

Advantages - Virtual Machines

- Provide **software compatibility** to the software running on it
- Provides **isolation** between different types of operating systems and processes
- Provide **encapsulation** and software present on a virtual machine can be modified and controlled
- Provide various features like no dual booting, transfer of files between virtual machines, error in one OS doesn't affect the other OS present on the host, fresh OS can be added easily.
- Provide **good software management** like these can run a complete software stack of the host machine, run legacy OS, etc..

VIRTUALIZATION AND OPERATING SYSTEM COMPONENTS

- It refers to an **operating system feature in which the kernel enables the existence of various isolated user-space instances**.
- The installation of virtualization software refers to **Operating system-based virtualization**.
- It is installed over a pre-existing operating system and that operating system is called **the host operating system**.
- Virtualization software converts hardware IT resources that require unique software for operation into virtualized IT resources.

How Operating System Virtualization Works?

- The OS manages all the software and hardware of the computer.
- Several different computer programs can run at the same time.
- This is done by using the CPU of the computer.
- With the combination of few components of the computer which is coordinated by the operating system, every program runs successfully.

VM
Guest operating system and application software

VM
Guest operating system and application software

VM
Guest operating system and application software

Virtual machine management

Hardware (virtualization host)

- Operating system-based services are: **Backup and Recovery, Security Management, Integration to Directory Services.**
- Various major **operations of Operating System Based Virtualization** are:
 1. *Hardware capabilities can be employed, such as the network connection and CPU.*
 2. *Connected peripherals with which it can interact, such as webcam, printer, keyboard, or Scanners.*
 3. *Data that can be read or written, such as files, folders, and network shares.*
- **Operating system-based virtualization** can raise **demands and problems** related to **performance overhead**, such as:
 1. *The host operating system employs CPU, memory, and other hardware IT resources.*
 2. *Hardware-related calls from guest operating systems need to navigate numerous layers to and from the hardware, which shrinkage overall performance.*
 3. *Licenses are frequently essential for host operating systems, in addition to individual licenses for each of their guest operating systems.*

Types of OS Virtualization

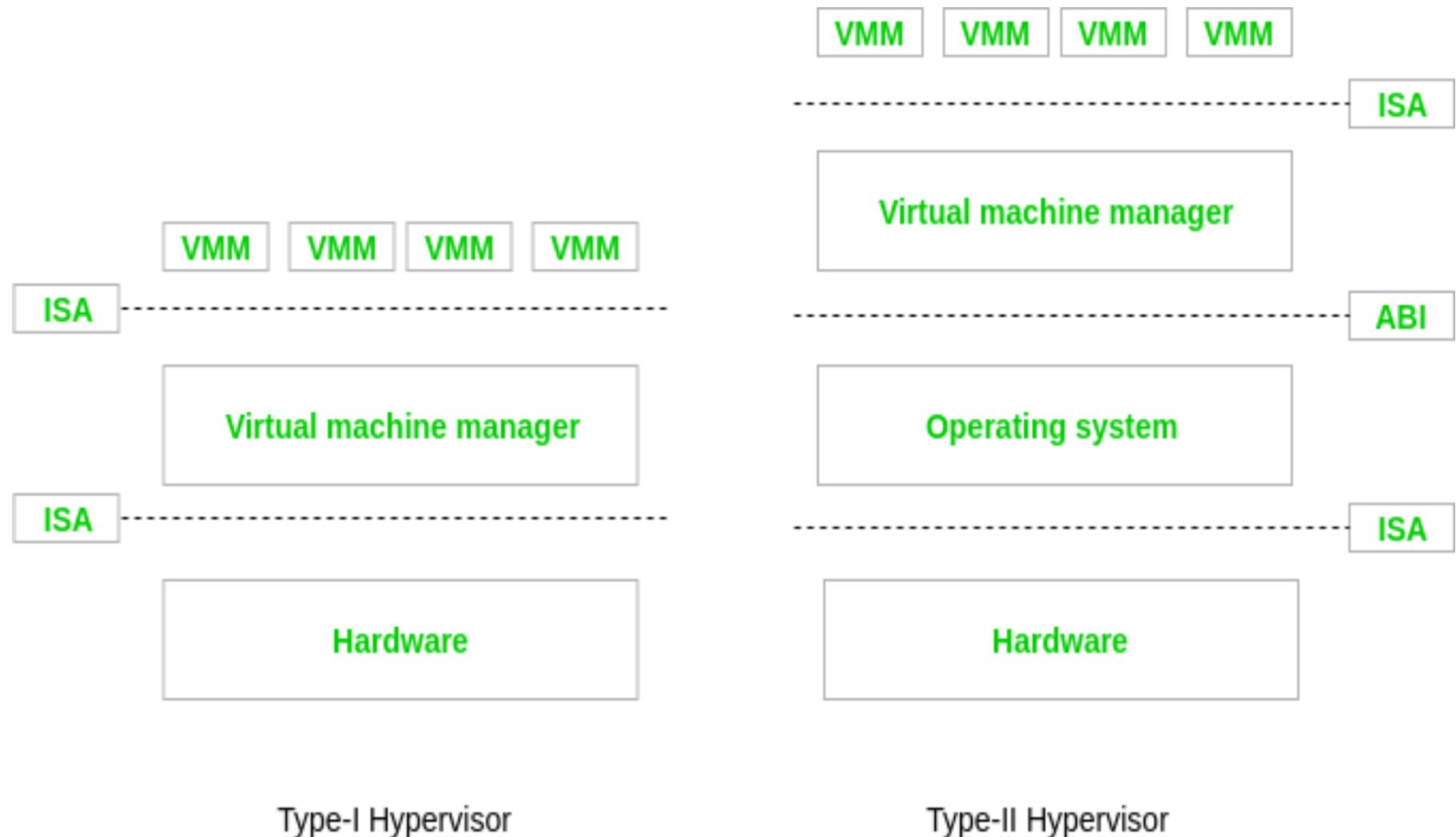
- **Linux Operating System virtualization:** VMware Workstation software is used to virtualize Linux systems.
- In addition, to install any software by the means of virtualization the user will need VMware software to install first.
- **Windows Operating System virtualizations:** This type of virtualization is similar to the above to install any software there is a need to install VMware software firstly.

Advantages of OS Virtualization

- Eliminates the use of physical space which utilizes by the IT system
- No hardware required the maintenance will be less; save both time and money.
- Lower power consumption, lower cooling requirement, low maintenance, and more electricity savings.
- It allows the companies to make enhancement in terms of efficiency to use the server hardware and thus there is a greater return on investment (ROI) on the purchase and greater operational works.
- Quick deployment capability

Hardware Based Virtualization

- **Hardware based virtualization** - A platform virtualization approach that allows efficient full virtualization with the help of hardware capabilities, primarily from the host processor.
- **Hardware-level virtualization** - An abstract execution environment in terms of computer hardware in which guest OS can be run
- A fundamental component of hardware virtualization is the **hypervisor, or virtual machine manager (VMM)**.
- Two types of Hypervisors:
 - **Type-I hypervisors**
 - **Type-II hypervisors**



Type-I Hypervisor

Type-II Hypervisor

Advantages of Hardware-based Virtualization

- Reduces the maintenance overhead of paravirtualization
- Convenient to attain enhanced performance
- More Efficient Resource Utilization
- Lower Overall Costs Because Of Server Consolidation
- Increased Uptime Because Of Advanced Hardware Virtualization Features
- Increased IT Flexibility

Disadvantages of Hardware-based Virtualization

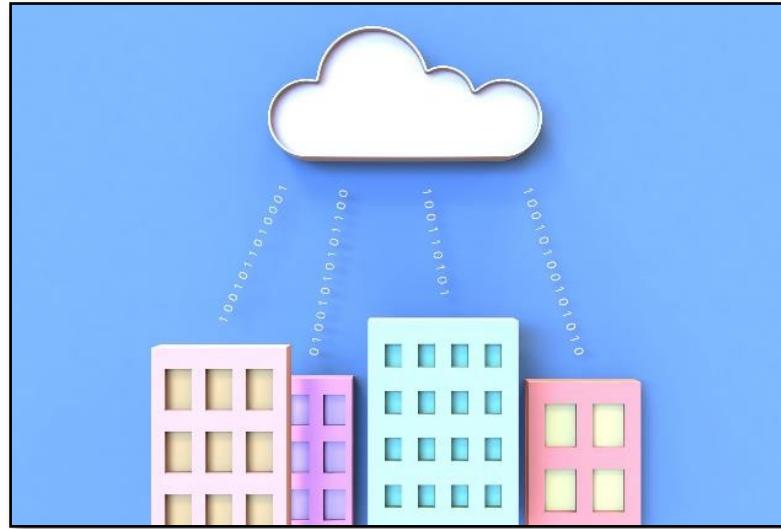
- Hardware-based virtualization requires explicit support in the host CPU, which may not be available on all x86/x86_64 processors.
- A “pure” hardware-based virtualization approach, including the entire unmodified guest operating system, involves many VM traps, and thus a rapid increase in CPU overhead occurs which limits the scalability and efficiency of server consolidation.
- This performance hit can be mitigated by the use of para-virtualized drivers; the combination has been called “**hybrid virtualization**”.

Quick overview of cloud

- Part 1: What is Cloud Computing?
- Part 2: Six Advantages of Cloud Computing
- Part 3: What is Amazon Web Services (AWS)?
- Part 4: The AWS Cloud Adoption Framework (CAF)

What is Cloud Computing?

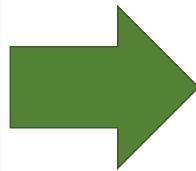
Cloud computing is the **on-demand** delivery of compute power, database storage, applications, and other IT resources through a cloud services platform **via the internet** with **pay-as-you-go** pricing.



Before Cloud Computing



Cloud computing enables you to **stop thinking of your infrastructure as hardware**, and instead **think of it (and use it) as software**.



Before Cloud Computing



- 💡 Hardware solutions are **physical**. This means they require:
 - 💡 Space
 - 💡 Staff
 - 💡 Physical security
 - 💡 Planning
 - 💡 Capital expenditure
- 💡 Guess at theoretical maximum peaks
 - 💡 Is there enough resource capacity?
 - 💡 Do we have sufficient storage?

What if your needs change?

You have to go through the **time, effort, and cost** required to change all these.

Utilizing Cloud Computing



Software is flexible.

If your needs change, your software can change much more **quickly, easily, and cost-effectively** than your hardware.

Three Models of Cloud Computing

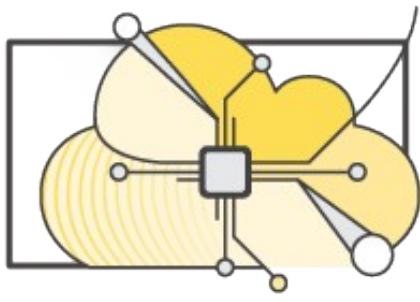


IaaS
Infrastructure
as a Service

PaaS
Platform
as a Service

SaaS
Software
as a Service

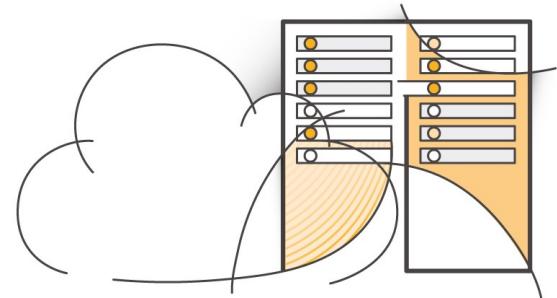
Three Cloud Deployment Models



All-In Cloud

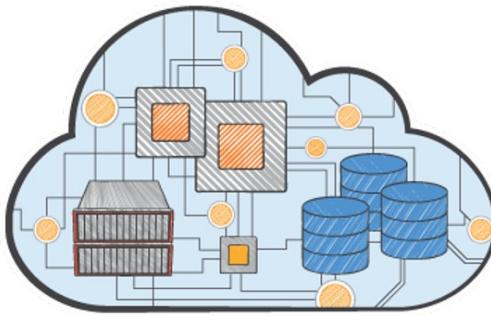


Hybrid



**Private Cloud
(On-premises)**

All-In Cloud versus On-Premises



All-In Cloud

- No upfront investment
- Low ongoing costs
- Focus on innovation
- Flexible capacity
- Speed and agility
- Global reach on demand



On-Premises

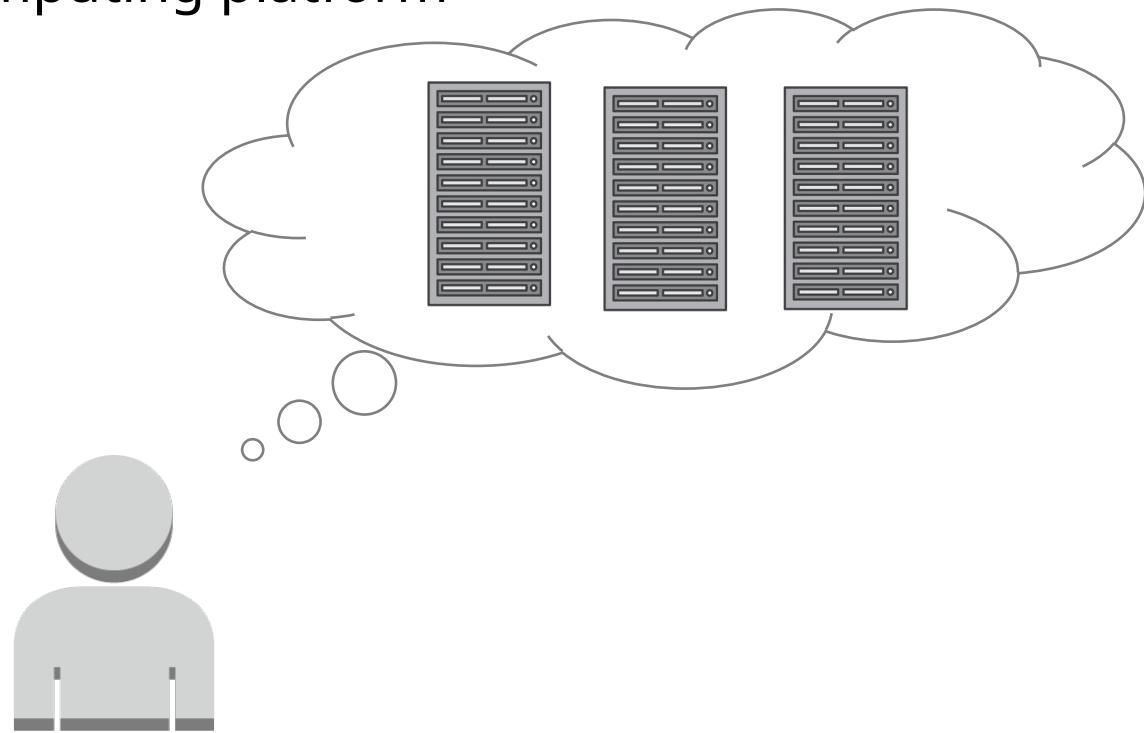
- Large initial purchase
- Labor, patches, and upgrade cycles
- Systems administration
- Fixed capacity
- Long procurement cycle and setup
- Limited geographic regions

What can you do in the cloud?



You can use a cloud computing platform for:

- Application Hosting
- Backup and Storage
- Content Delivery
- Websites
- Enterprise IT
- Databases



Important Cloud Terminology



💡 **High Availability (Highly Available):**

- 💡 Accessible when you need it

💡 **Fault Tolerance (Fault Tolerant):**

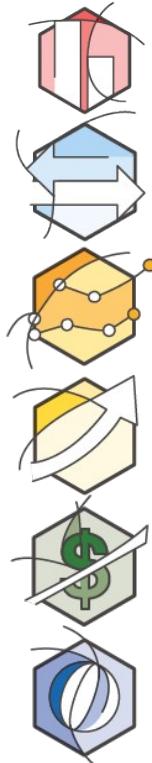
- 💡 Ability to withstand a certain amount of failure and still remain functional

💡 **Scalability (Scalable):**

- 💡 Ability to easily grow in size, capacity, and/or scope when required
- 💡 Growth is (usually) based on demand

💡 **Elasticity (Elastic):**

- 💡 Ability to grow (scale) when required and to reduce in size when resources are no longer needed



Trade **capital expense** for **variable expense**.

Benefit from **massive economies of scale**.

Eliminate guessing on your capacity needs.

Increase **speed** and **agility**.

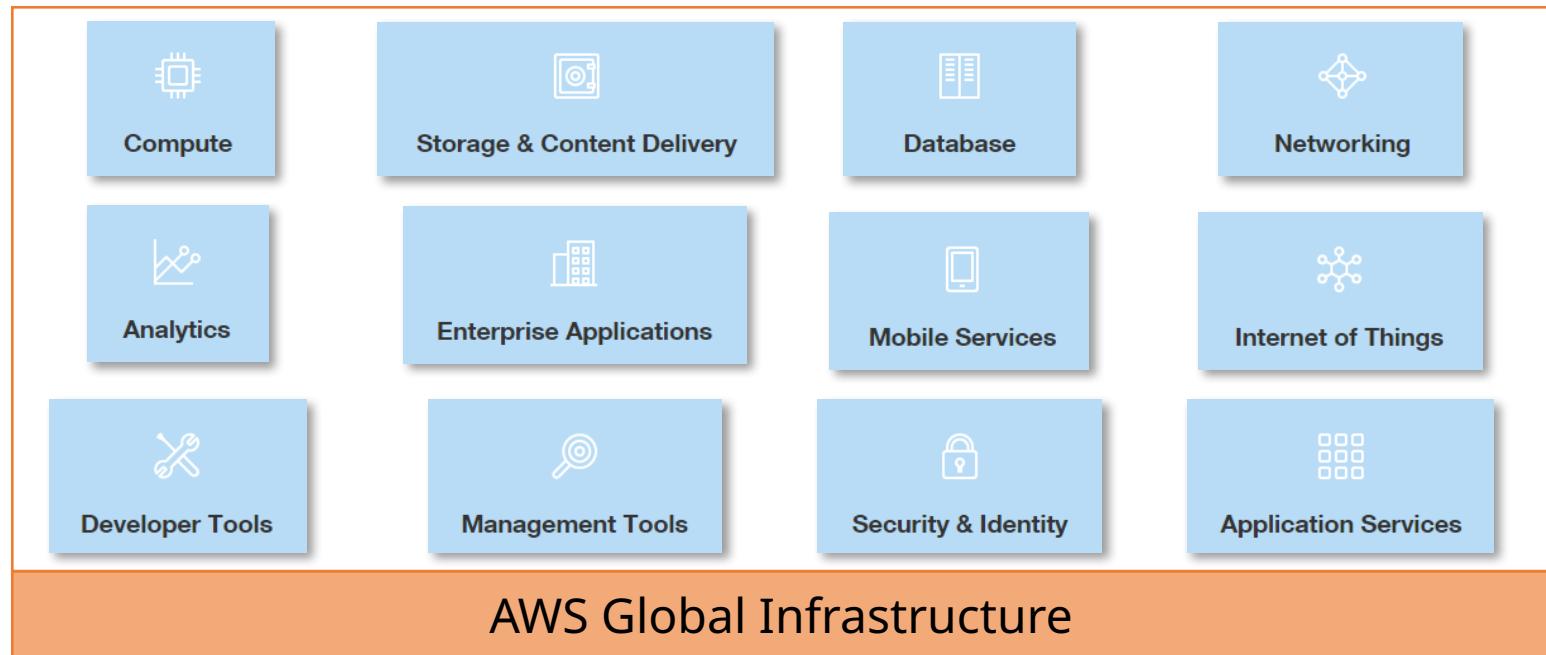
Stop spending money to run and maintain data centers.

Go global in minutes.

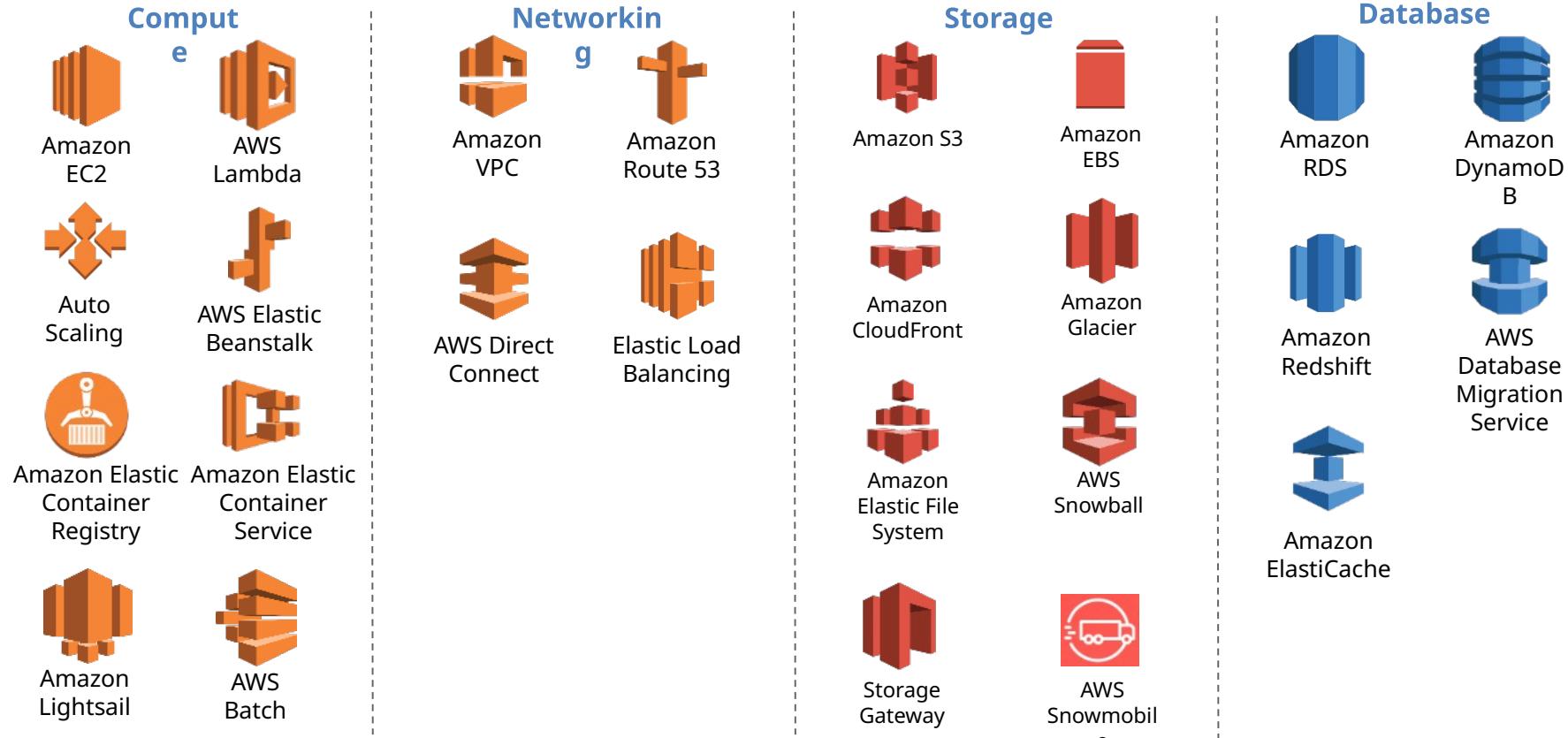
What is AWS?



AWS is a **secure cloud platform with more than 165 different services** that include solutions for:



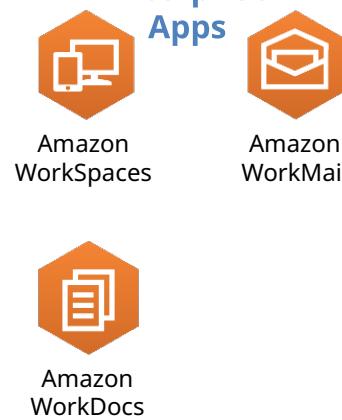
AWS by Category: Core Services



Analytics



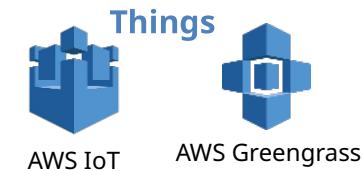
Enterprise Apps



Mobile Services



Internet of Things



AWS by Category

Developer Tools	Management Tools	Security & Identity	App Services
 AWS CodeCommit	 AWS CodeDeploy	 AWS Identity and Access Management	 Amazon API Gateway
 AWS CodePipeline	 AWS CloudTrail	 Amazon Inspector	 Amazon CloudSearch
 AWS X-Ray	 AWS OpsWorks	 AWS Key Management Service	 Amazon SES
	 AWS Trusted Advisor	 AWS Organizations	 AWS WAF
		 AWS Certificate Manager	 AWS Shield
			 Amazon SQS
			 Amazon AppStream
			 Amazon Elastic Transcoder
			 Amazon SNS
			 Amazon SWF

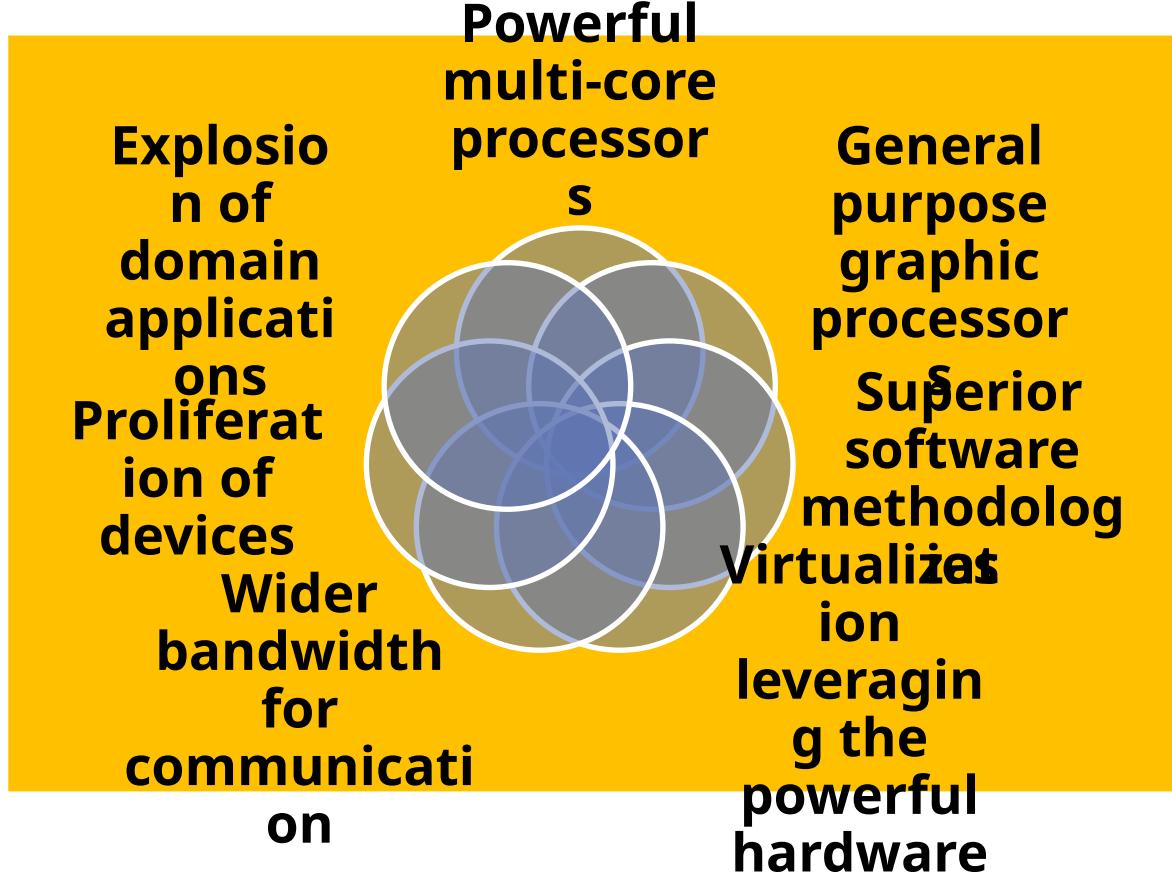
Access to AWS Services



- AWS Management Console
 - Access on the go with AWS Console Mobile App
- AWS Command Line Interface (AWS CLI)
- Software Development Kits (SDK)

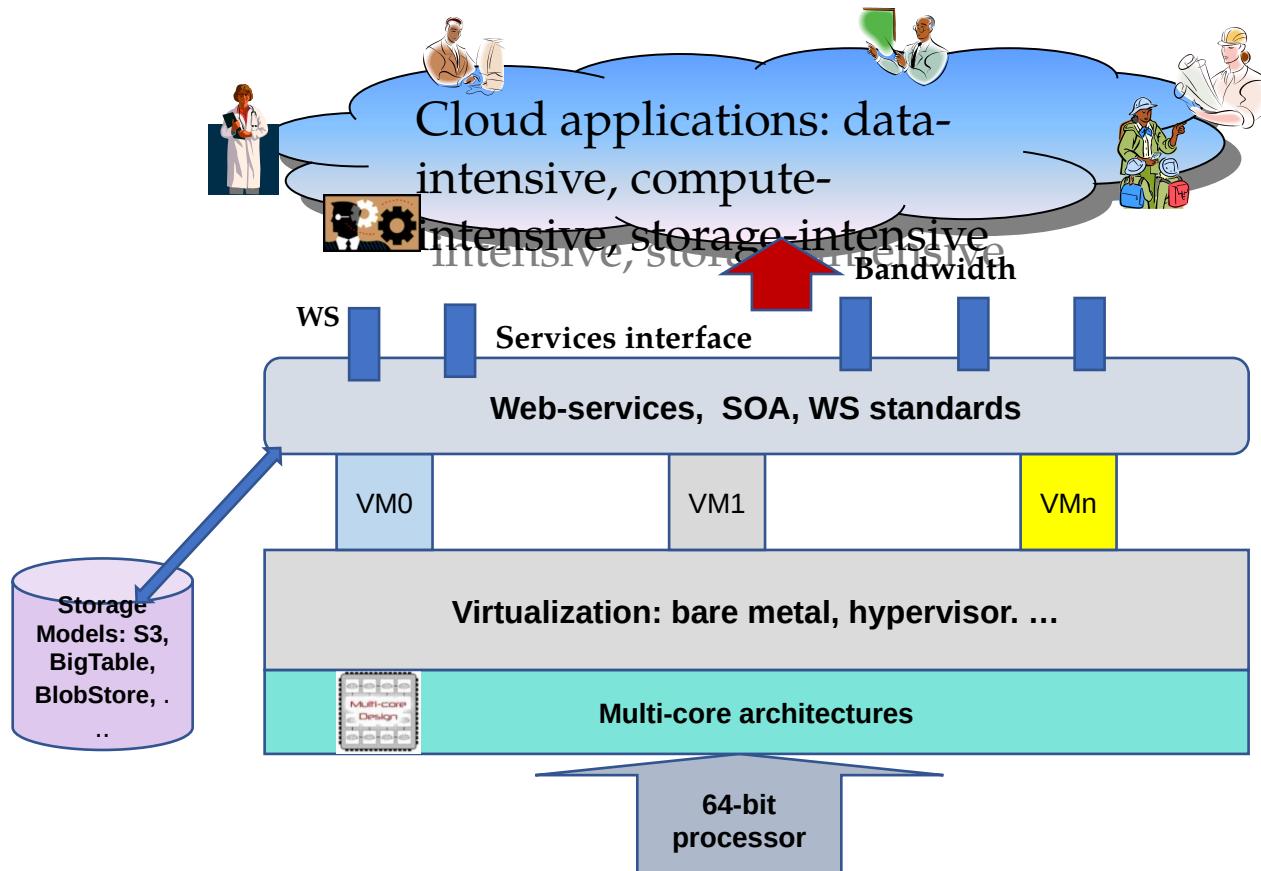


Motivation



- 1. Web Scale Problems**
 - 2. Web 2.0 and Social Networking**
 - 3. Information Explosion**
 - 4. Mobile Web**
-

Technology Advances



What is Cloud Computing?

Cloud Computing is a general term used to describe a new class of network based computing that takes place over the Internet,

- basically a step on from Utility Computing
 - a collection/group of integrated and networked hardware, software and Internet infrastructure (called a platform).
 - Using the Internet for communication and transport
- provides hardware, software and networking services to clients

These platforms hide the complexity and details of the underlying infrastructure from users and applications by providing very simple graphical interface or API (Applications Programming Interface).

What is Cloud Computing cont....

In addition, the platform provides on demand services, that are always on, anywhere, anytime and any place.

Pay for use and as needed, elastic

- scale up and down in capacity and functionalities
- The hardware and software services are available to
- general public, enterprises, corporations and businesses markets

Drivers for the new Platform

Generational Shift of Computing Platform

Technology	Economic	Business	
	Centralized compute & storage, thin clients	Optimized for efficiency due to high cost	High upfront costs for hardware and software
	PCs and servers for distributed compute, storage, etc.	Optimized for agility due to low cost	Perpetual license for OS and application software
	Large DCs, commodity HW, scale-out, devices	Order of magnitude better efficiency and agility	Pay as you go, and only for what you use

<http://blogs.technet.com/b/yungchou/archive/2011/03/03/chou-s-theories-of-cloud-computing-the-5-3-2-principle.aspx>

Cloud Summary



- Shared pool of configurable computing resources
 - On-demand network access
 - Provisioned by the Service Provider
-

Cloud Computing: Definition

The US National Institute of Standards (NIST) defines cloud computing as follows:

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

3-4-5 rule of Cloud Computing

NIST specifies 3-4-5 rule of Cloud Computing

- 3** cloud service models or service types for any cloud platform
- 4** deployment models
- 5** essential characteristics of cloud computing infrastructure

Characteristics of Cloud Computing

5 Essential Characteristics of Cloud Computing

Ref: The NIST Definition of Cloud Computing

<http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>

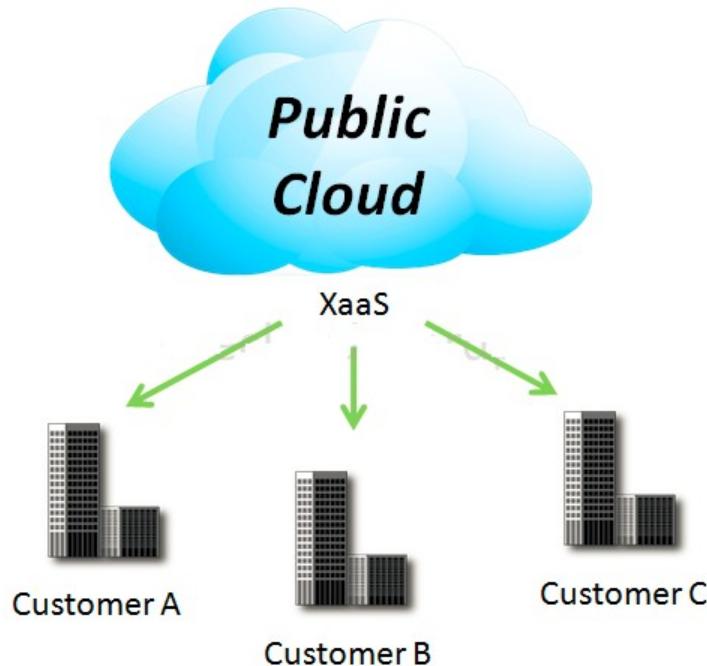


Source: <http://aka.ms/532>

- On demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

4 Deployment Models

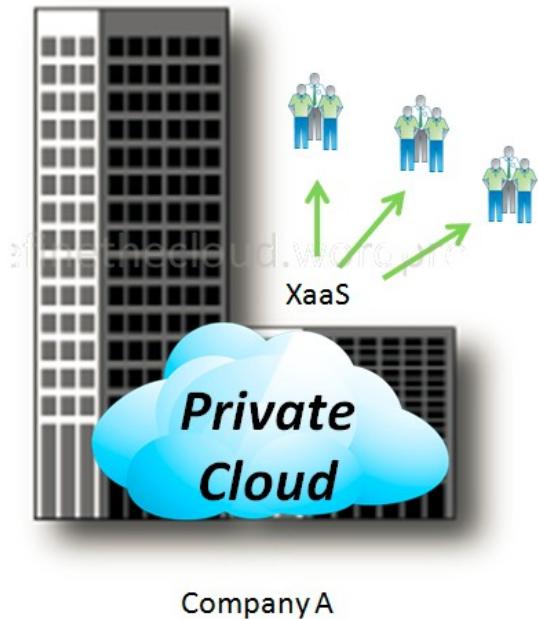
1. Public Cloud



Mega-scale cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

4 Deployment Models

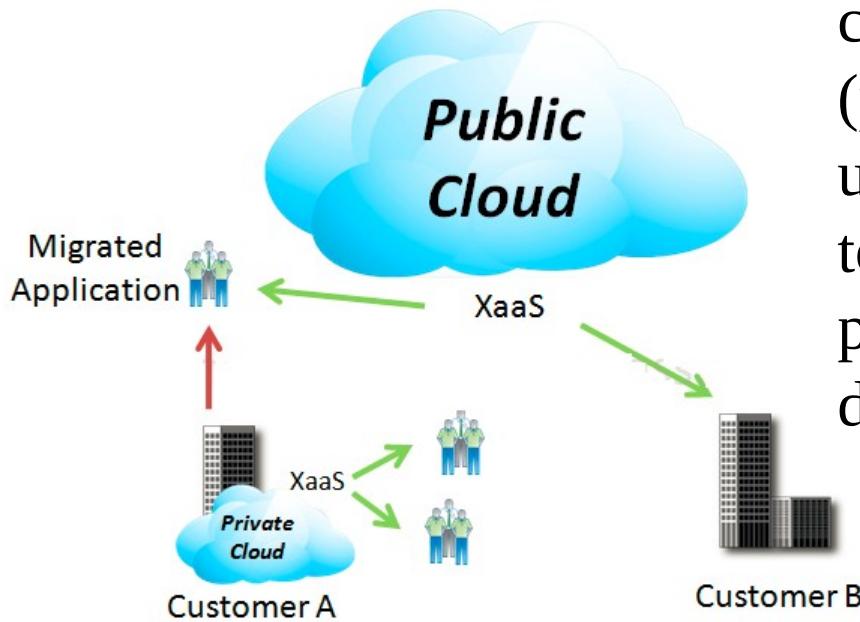
2. Private Cloud



The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

4 Deployment Models

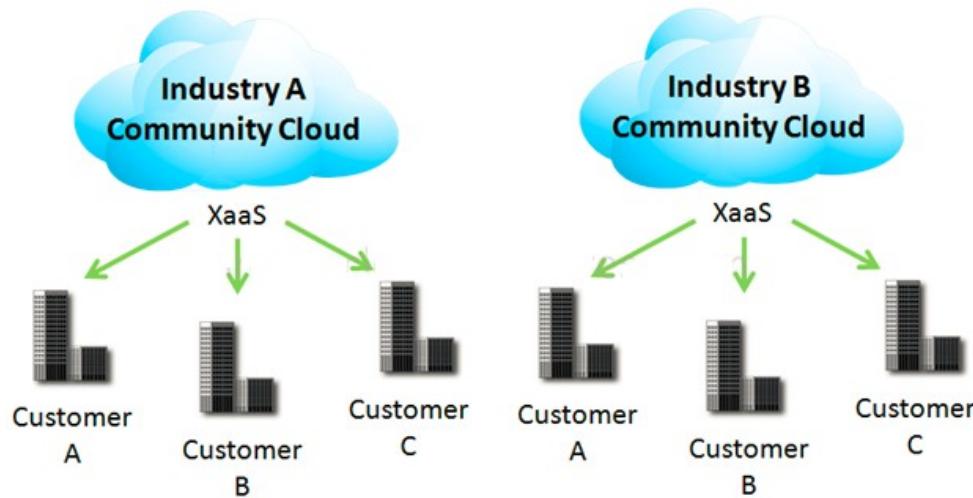
3. Hybrid Cloud



The cloud infrastructure is a composition of two or more clouds (private or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability

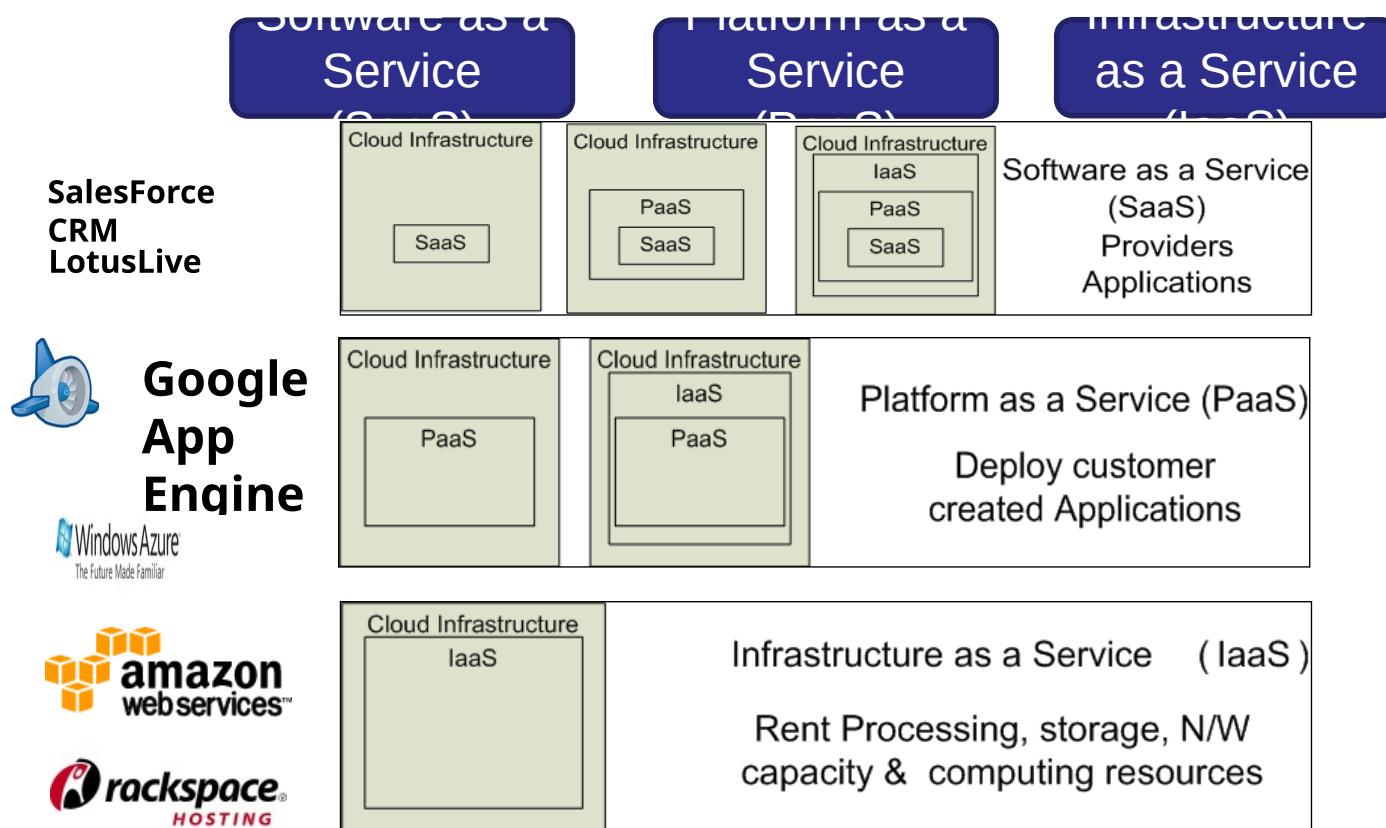
4 Deployment Models

4. Community Cloud



Community Clouds are when an ‘infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise’ according to NIST. A community cloud is a cloud service shared between multiple organizations with a common tie/goal/objective. E.g. OpenCirrus

3 Cloud Service Models



Software as a Service (SaaS)

Software as a service features a complete application

offered as a service on demand.

A single instance of the software runs on the cloud and services multiple end users or client organizations.

E.g. salesforce.com , Google Apps

Platform as a Service

Platform as a service encapsulates a layer of software and provides it as a service that can be used to build higher-level services.

2 Perspectives for PaaS :-

- 1. Producer:-** Someone producing PaaS might produce a platform by integrating an OS, middleware, application software, and even a development environment that is then provided to a customer as a service.
- 2. Consumer:-** Someone using PaaS would see an encapsulated service that is presented to them through an API. The customer interacts with the platform through the API, and the platform does what is necessary to manage and scale itself to provide a given level of service.

Virtual appliances can be classified as instances of PaaS.

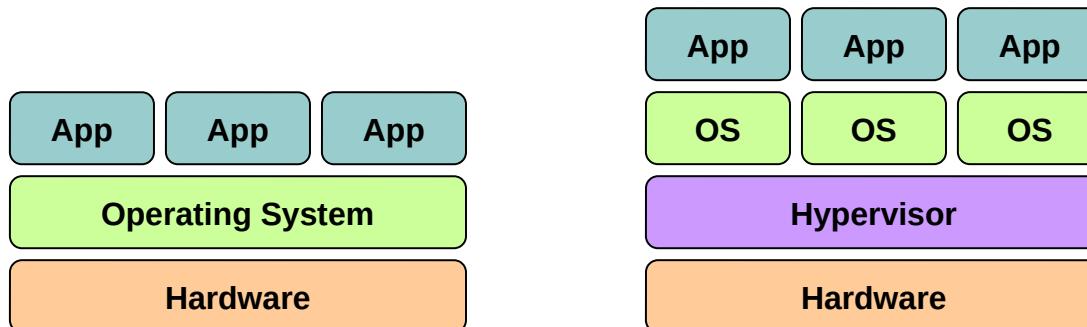
Infrastructure as a Service

Infrastructure as a service delivers basic storage and computing capabilities as standardized services over the network.

Servers, storage systems, switches, routers , and other systems are pooled and made available to handle workloads that range from application components to high-performance computing applications.

Cloud Infrastructures

Key Technology is Virtualization



Virtualization plays an important role as an enabling technology for datacentre implementation by abstracting compute, network, and storage service platforms from the underlying physical hardware

Cloud Providers Characteristics

- **Provide on-demand provisioning of computational resources**
- **Use virtualization technologies to lease these resources**
- **Provide public and simple remote interfaces to manage those resources**
- **Use a pay-as-you-go cost model, typically charging by the hour**
- **Operate data centers large enough to provide a seemingly unlimited amount of resources to their clients**

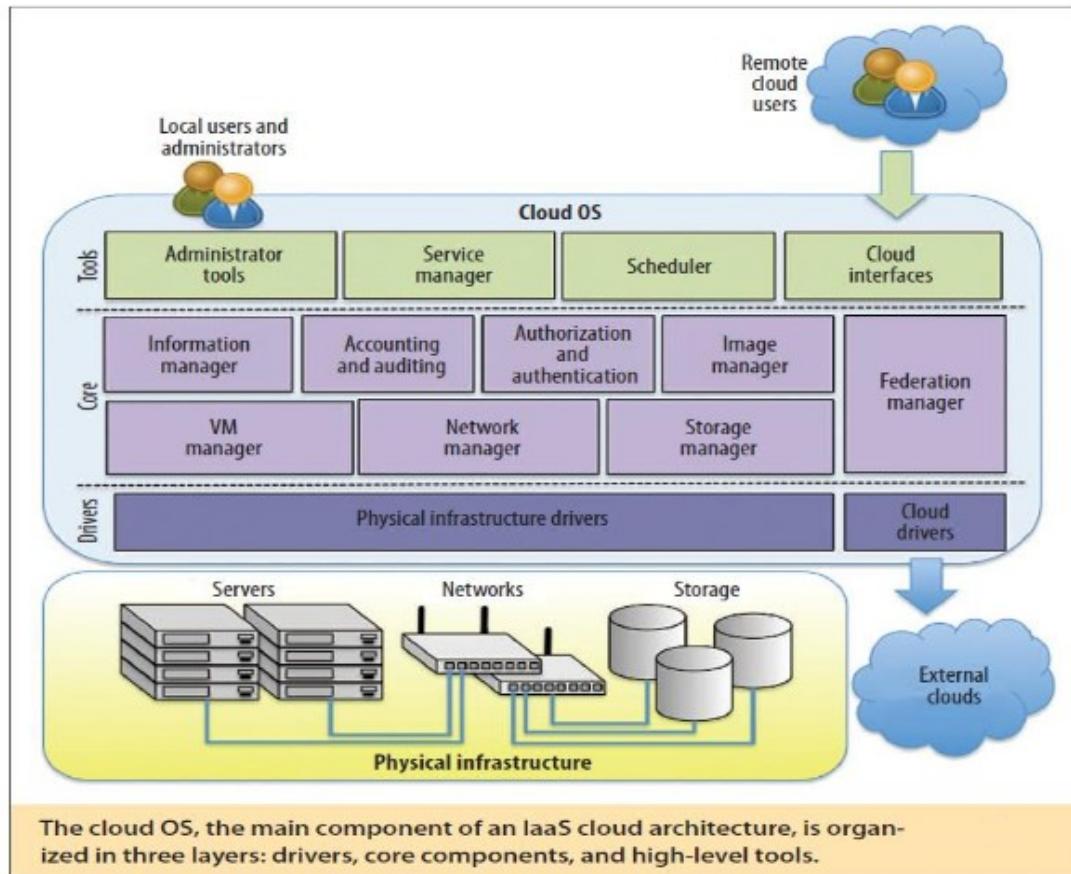
Management of Virtualized Resources

Distributed Management of Virtual Machines

Reservation-Based Provisioning of Virtualized Resources

Provisioning to Meet SLA Commitments

The Cloud OS



The cloud operating system is responsible for:

1. managing the physical and virtual infrastructure,
2. orchestrating and commanding service provisioning and deployment
3. providing federation capabilities for accessing and deploying virtual resources in remote cloud infrastructures



BITS Pilani
Pilani Campus



THANK YOU