



BITS Pilani
Pilani Campus

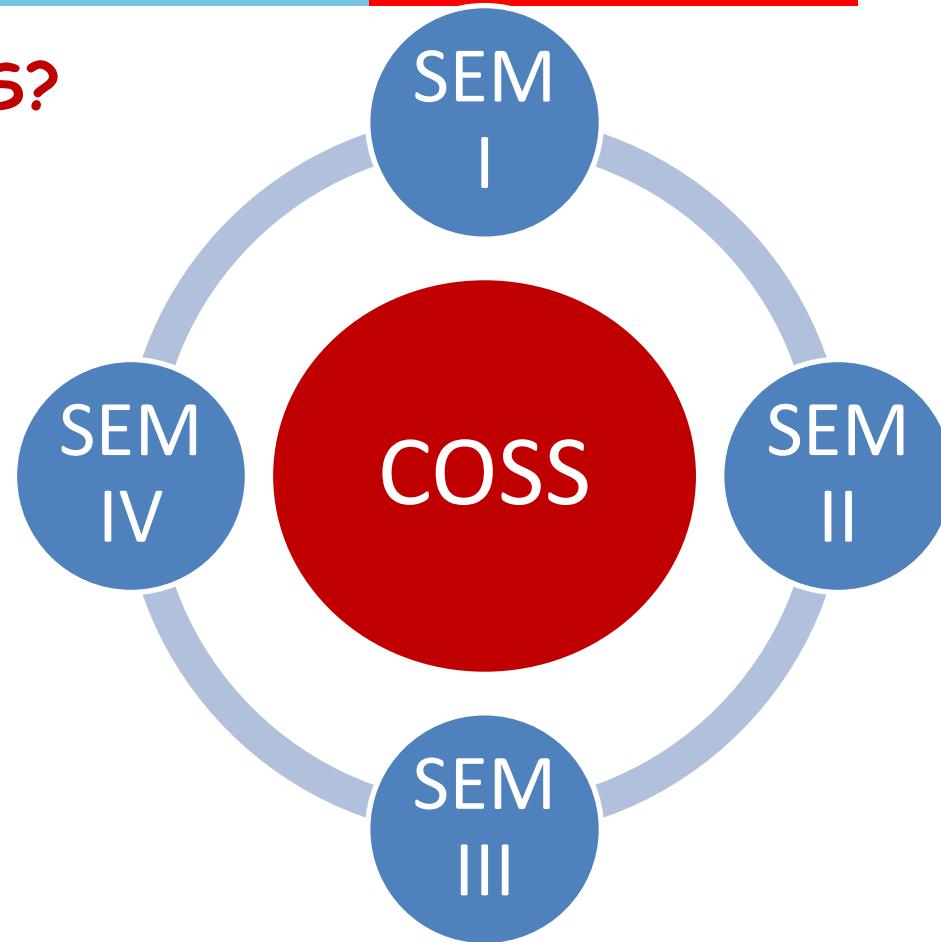
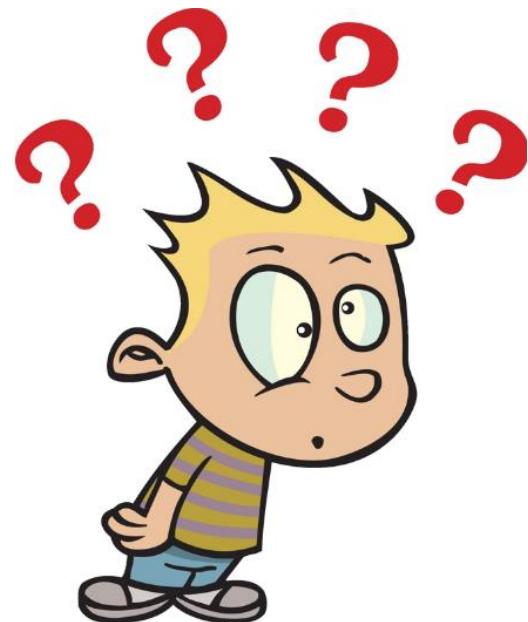
Computer Organization and Software Systems

CONTACT SESSION 1

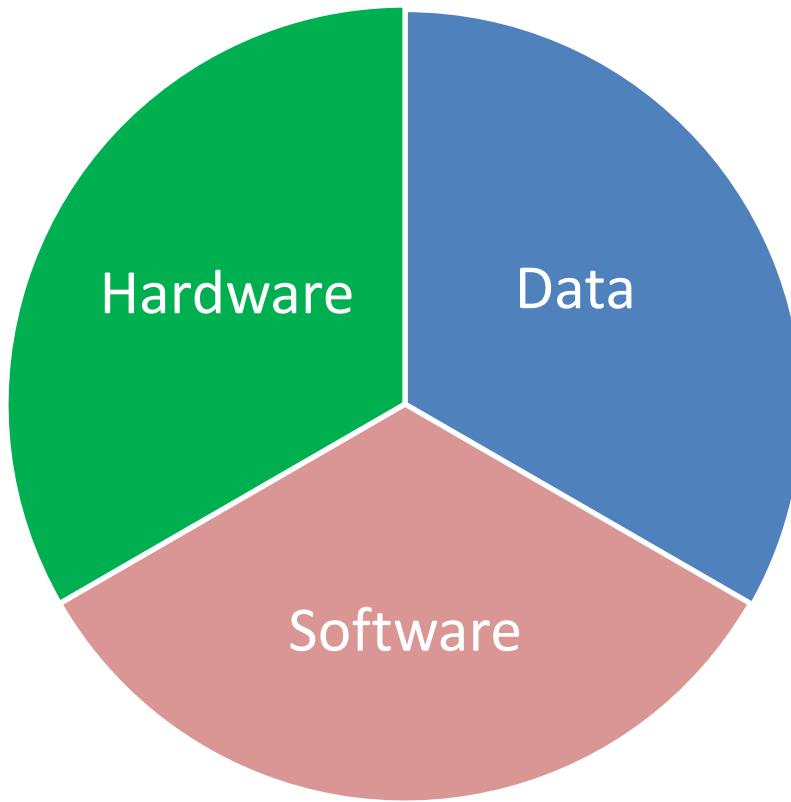
Prepared by Dr. Lucy J. Gudino
Instructor Prof. C R Sarma

Introduction

Why Study **COSS**?



Introduction



Data analytics: is the process of examining **data** sets in order to draw conclusions about the information they contain, increasingly with the aid of **specialized systems** and **software**.

Text Books and Reference Books

Text Books:

- (T1) W. Stallings, *Computer Organization & Architecture*, PHI, 10th ed., 2010.
- (T2) A Silberschatz, Abraham and others, *Operating Systems Concepts*, Wiley Student Edition, 8th Edition

Reference Books:

- (R1) Patterson, David A & J L Hennenssy, *Computer Organization and Design – The Hardware/Software Interface*, Elsevier, 5th Ed., 2014.
- (R2) Randal E. Bryant, David R. O'Hallaron, *Computer Systems – A Programmer's Perspective*, Pearson, 3rd Ed, 2016.
- (R3) Tanenbaum, *Modern Operating Systems*: Pearson New International Edition, Pearson Education, 2013 (Pearson Online)
- (R4) Stallings, *Operating Systems: Internals and Design Principles* : International Edition, Pearson Education, 2013 (Pearson Online)

Evaluation Scheme

5 unit course.

Sl No.	Evaluation Component	Duration	Weightage %	Nature of Component
1	Mid Sem Exam	90 min	30%	Open Book
2	Comprehensive Examination	180 min	40%	OB
3	Quiz - 2	-----	5%	OB
4	Assignments	---	25%	OB

Assignments

- Two assignments:
 - One pre-midsem exam : 12%
 - One post-midsem : 13%
- Lab based
- Simulator to be used : CPU-OS simulator
 - Open source tool
https://drive.google.com/open?id=12YUK52RQ-JhPOddj6CD_oifW4sTMbsBI
 - Virtual lab (Platifi)

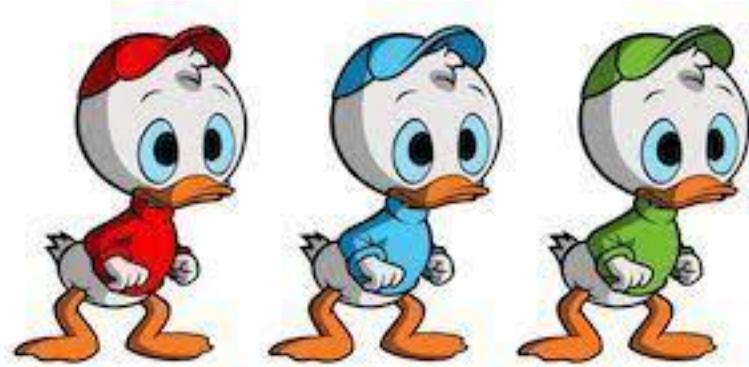
Assignment should not be

FILL IT.



SHUT IT.

FORGET IT.



General Instructions

1. Always use note book for writing important points and for solving problems
2. Use chat box for writing subject related questions
3. Do not repeat the questions on chat box. Questions will be answered during last 10 minutes of the session
4. Unanswered questions will be put up on the canvas forum

Today's Session

Contact Hour	List of Topic Title	Text/Ref Book/external resource
1-2	<p>Introduction to Computer Systems</p> <ul style="list-style-type: none"> • Hardware Organization of a computer • Basic uniprocessor architecture • Instruction Cycle State Diagram • Operating System role in Managing Hardware • Running a Hello Program 	T1

Definition of a Computer

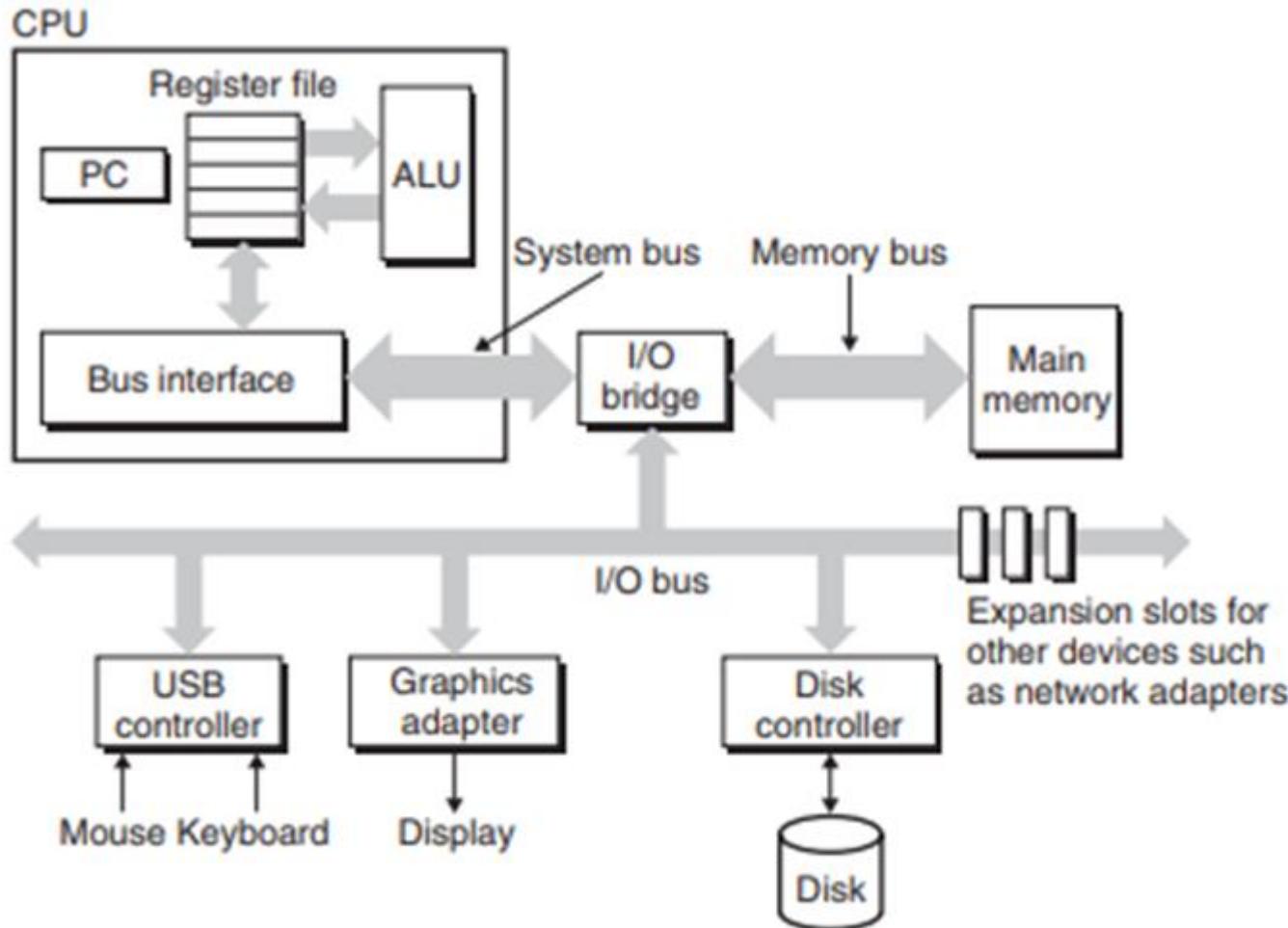
- Is a complex system
- Is a programmable device
- Must be able to **process** data
- Must be able to **store** data
- Must be able to **move** data
- Must be able to **control** above three functions

Computer System



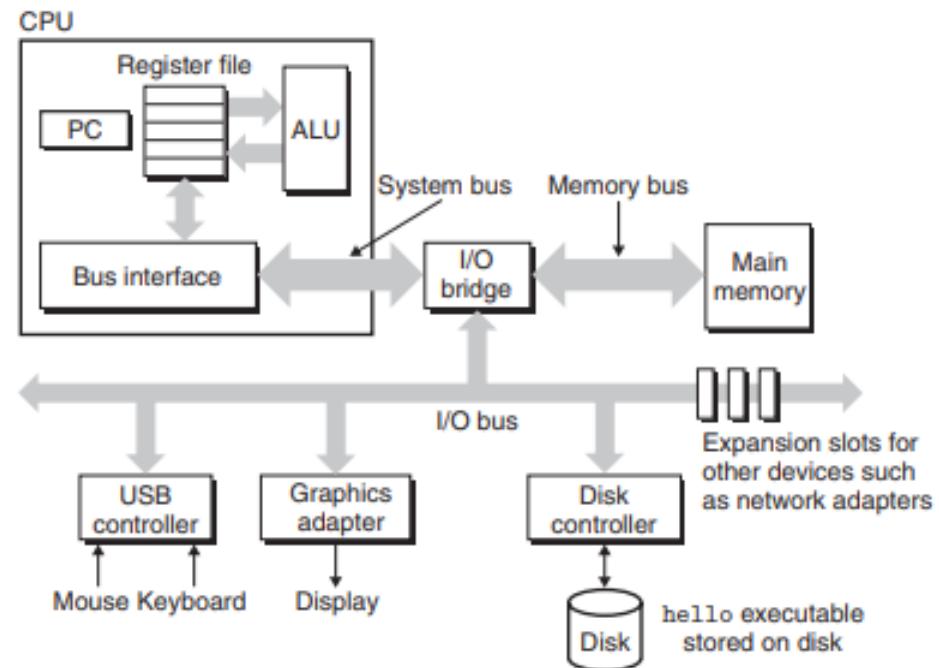
- Hardware
 - Central Processing Unit (CPU)
 - Memory
 - I/O devices
- Software
 - System Software
 - System Management Software
 - Tools and Utilities for Developing the software
 - Application Software
 - General Purpose Software
 - Specific Purposed Software

Hardware Organization of a computer



Von Neumann Architecture

- Three key concepts:
 - Data and instructions are stored in a single read - write memory
 - The contents of this memory are addressable by location, without regard to the type of data contained there
 - Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next

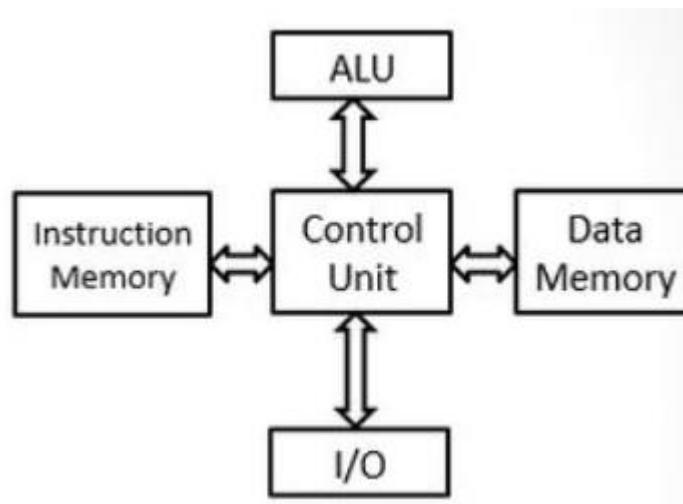


Von Neumann Architecture...

- Stored-program computers have the following characteristics:
 - Three hardware systems:
 - A central processing unit (CPU)
 - A main memory system
 - An I/O system
 - The capacity to carry out sequential instruction processing.
 - A single path between the CPU and main memory.
 - This single path is known as the **von Neumann bottleneck**.
 - Side effect : reduced throughput (Data Rate)

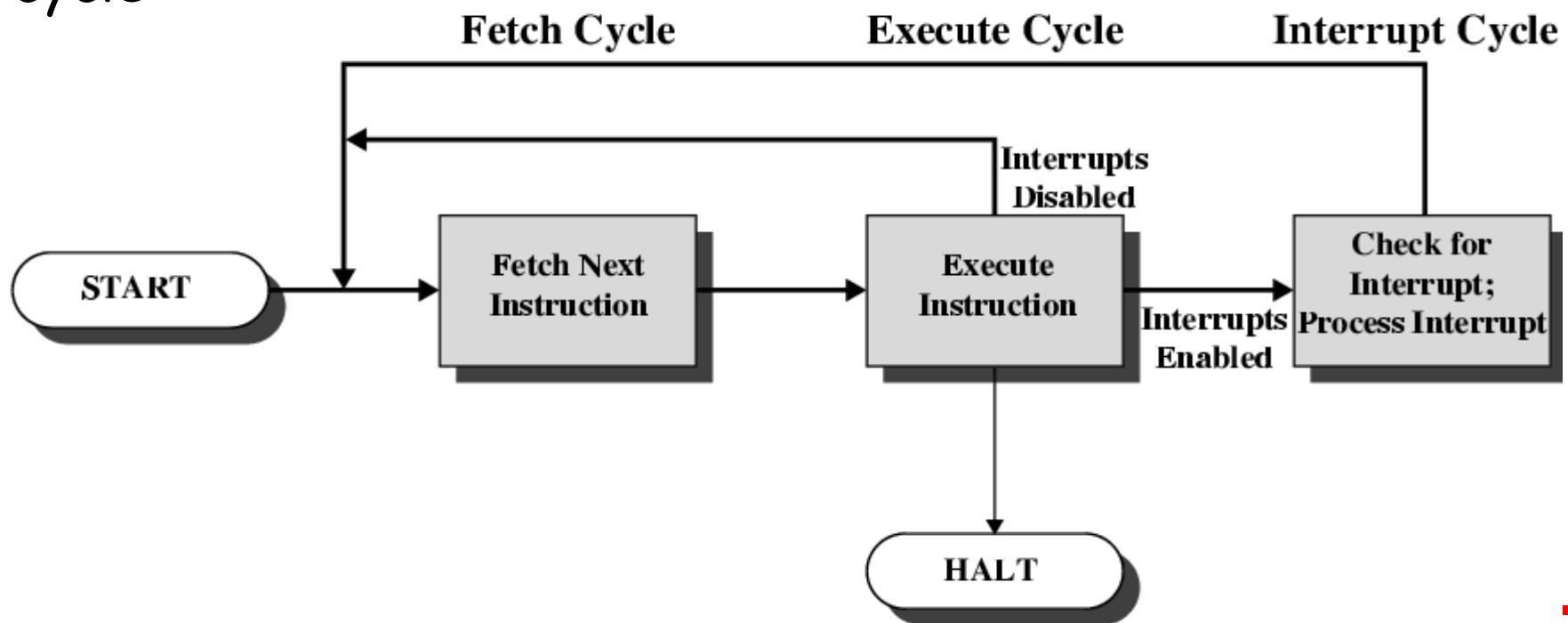
Harvard Architecture

- Uses two memory systems and two separate busses
 - Instruction Memory
 - Data Memory



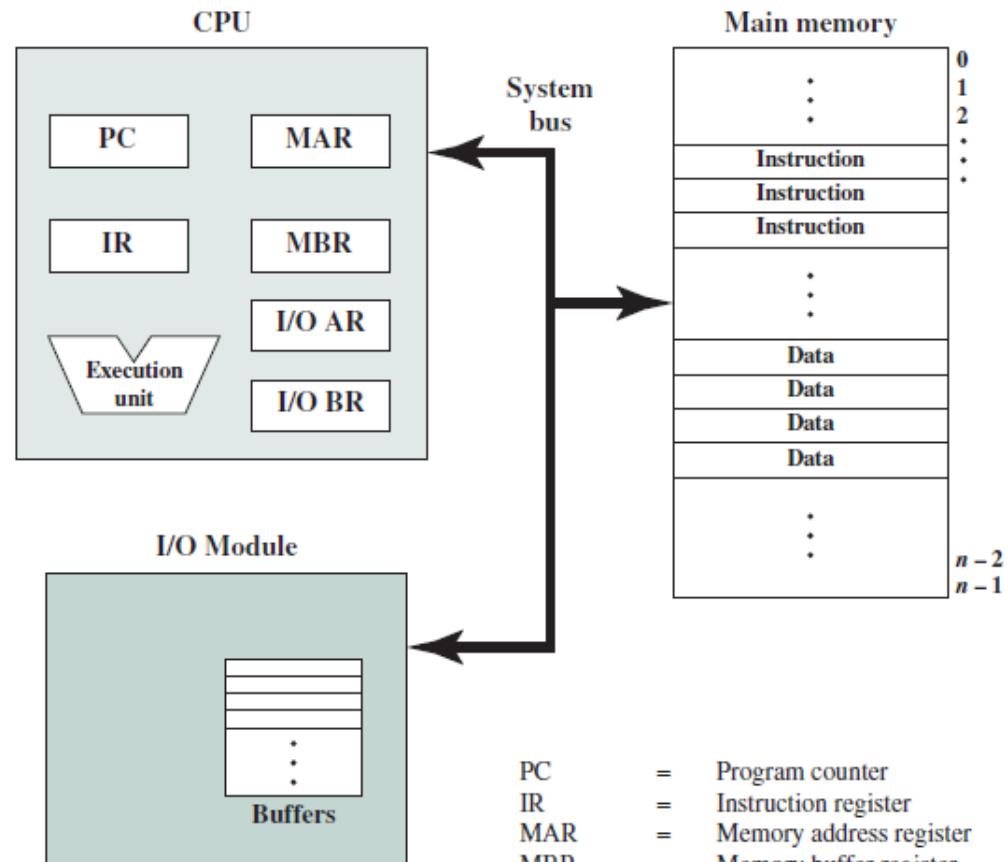
Instruction Cycle Diagram

- Instruction execution : Two steps:
 - Fetch
 - Execute
- Interrupt: Interrupt is checked at the end of Instruction cycle



Fetch Cycle

- Program Counter (PC) holds address of next instruction to be fetched
- Processor fetches instruction from memory location pointed to by PC
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions during execution cycle
- Increment PC
 - Unless told otherwise



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

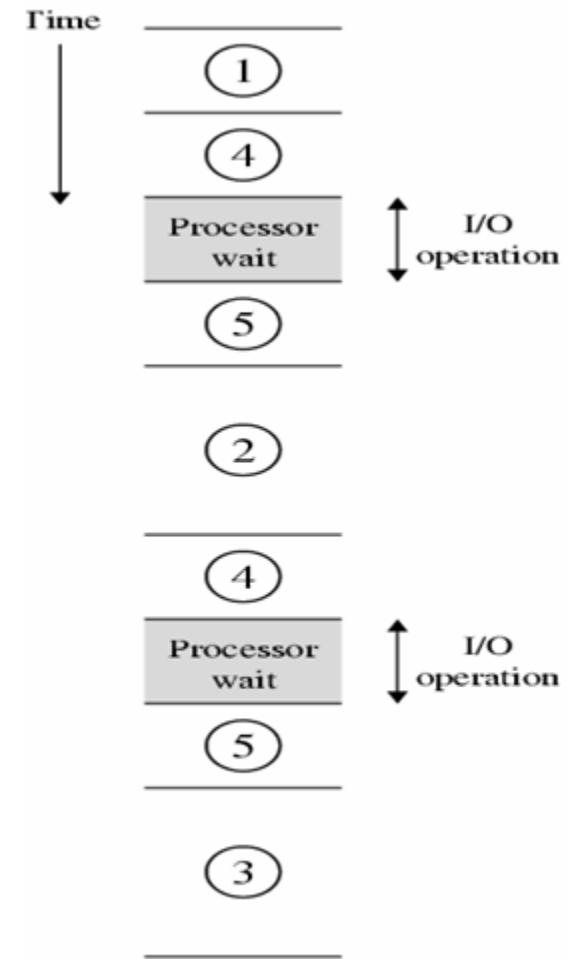
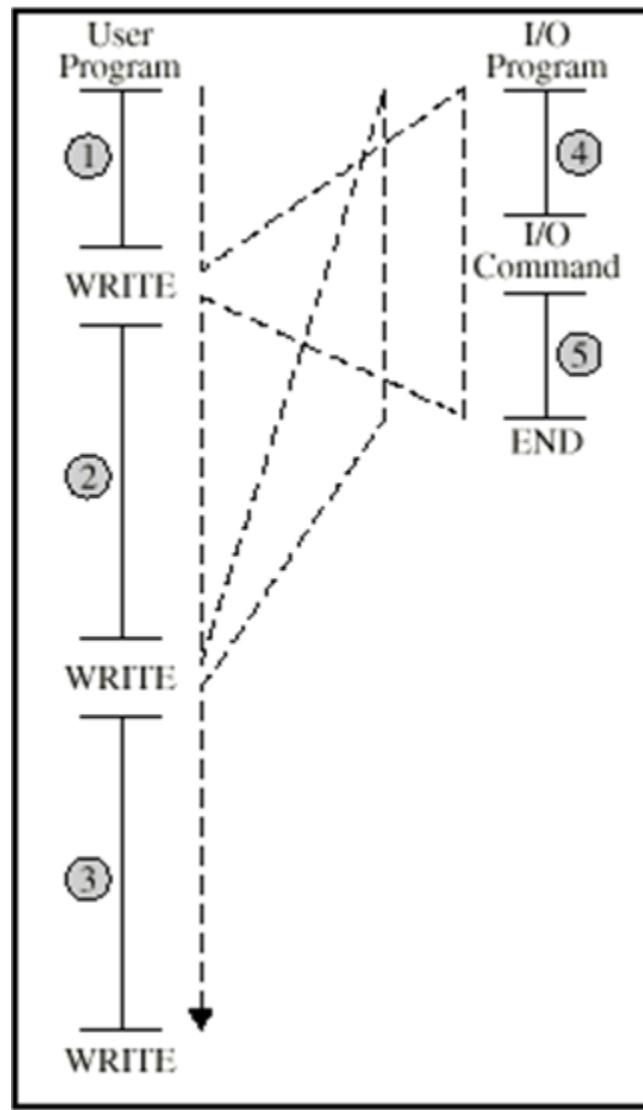
Execute Cycle

- Processor - memory
 - Data transfer between CPU and main memory
- Processor - I/O
 - Data transfer between CPU and I/O module
- Data processing
 - Some arithmetic or logical operation on data
- Control
 - Alteration of sequence of operations
 - e.g. jump
- Combination of above

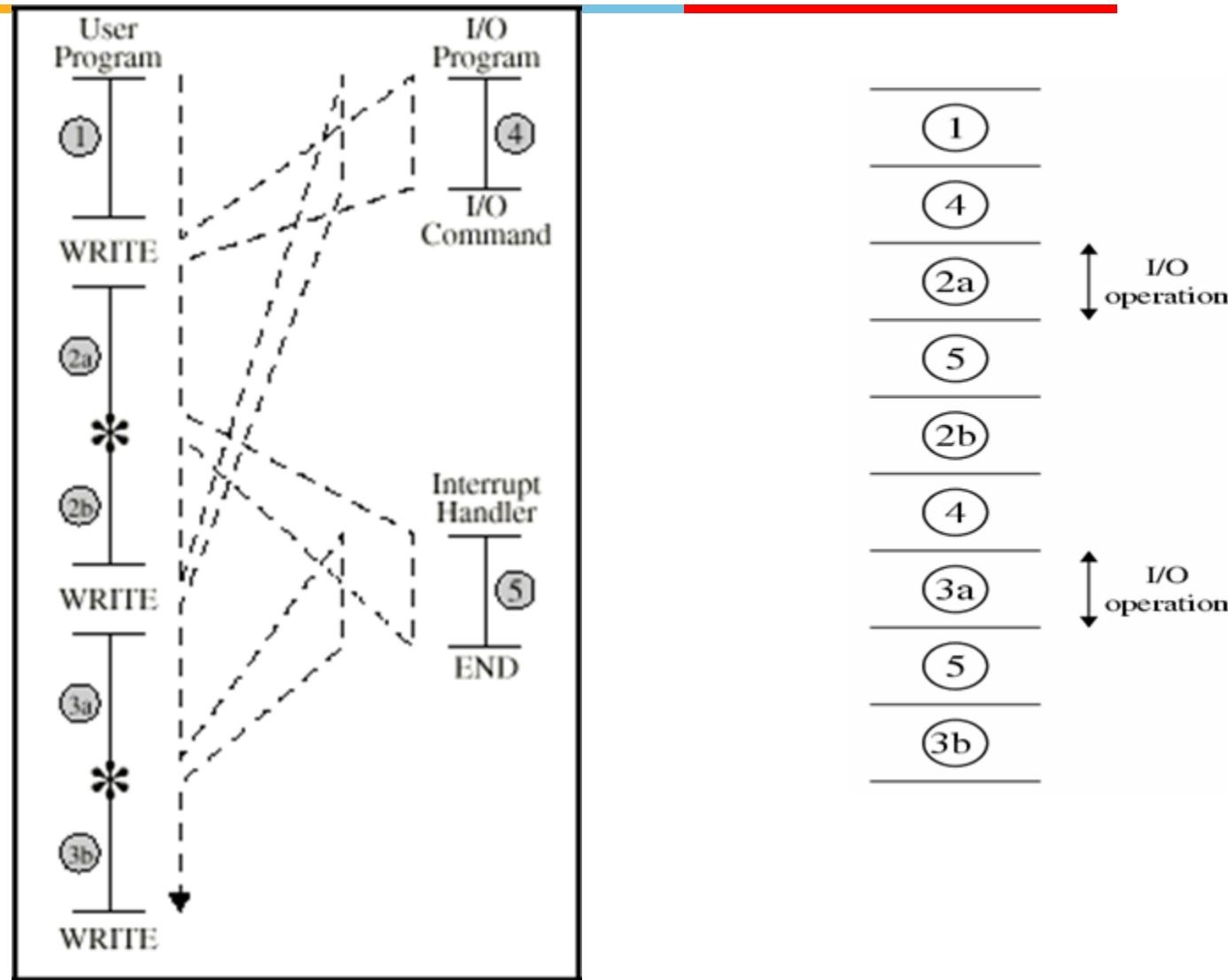
Interrupt Cycle

- Interrupts: Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Interrupts enhances processing efficiency
- Processor checks for interrupt
 - Indicated by an interrupt signal
- If no interrupt, fetch next instruction
- If interrupt pending:
 - Suspend execution of current program
 - Save context
 - Set PC to start address of interrupt handler routine
 - Process interrupt
 - Restore context and continue interrupted program

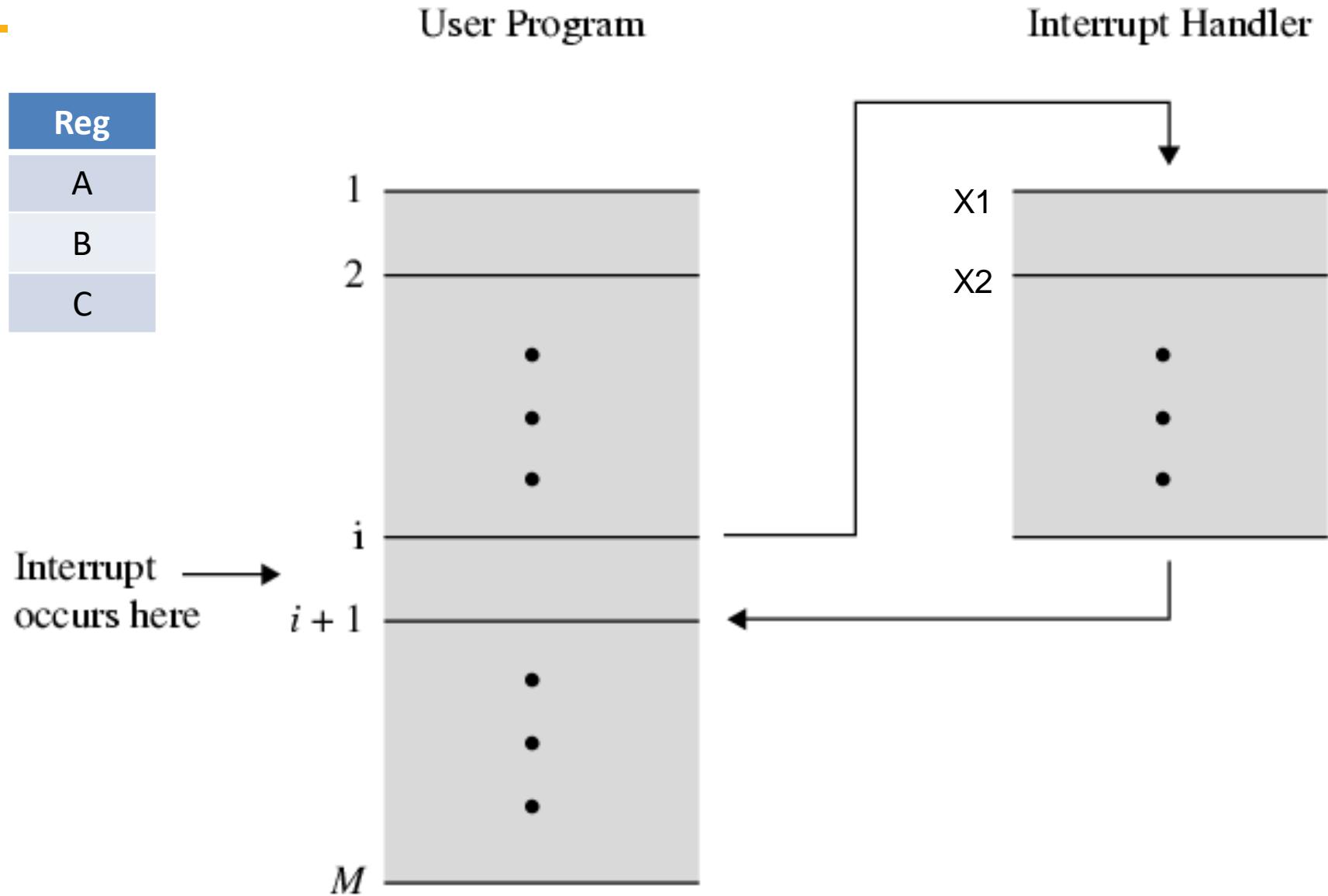
Program Flow Control (No Interrupts)



Program Flow Control (With Interrupts)



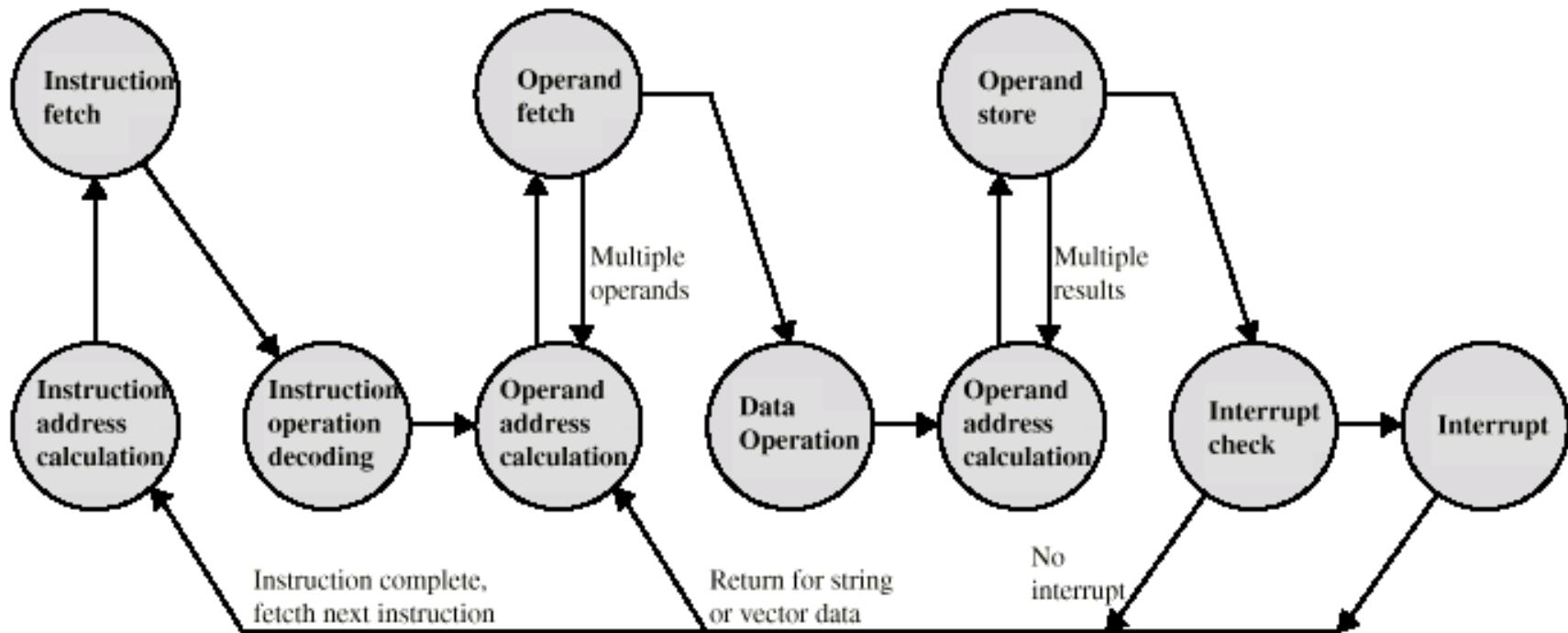
Transfer of Control via Interrupts



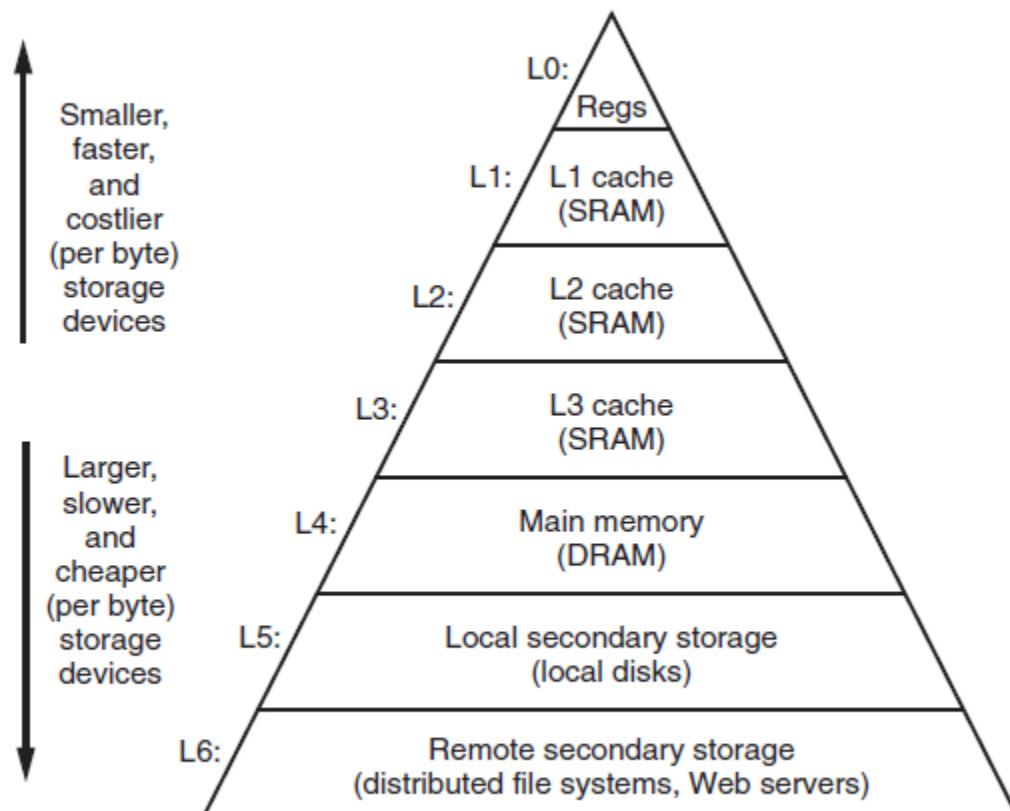
Types of Interrupts

- Types of interrupts:
 - Program
 - e.g. overflow, division by zero
 - Timer
 - Generated by internal processor timer
 - Used in pre-emptive multi-tasking
 - I/O
 - from I/O controller
 - Hardware failure
 - e.g. memory parity error

Instruction Cycle - State Diagram



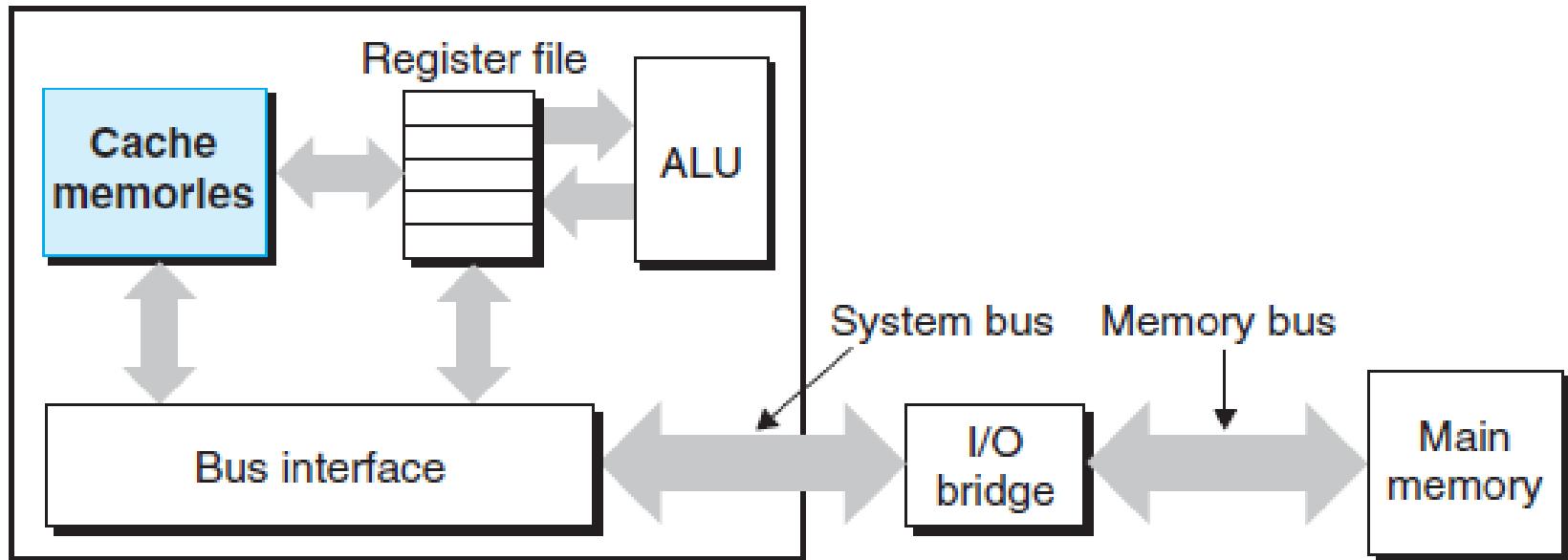
Memory Hierarchy



An example of a memory hierarchy.

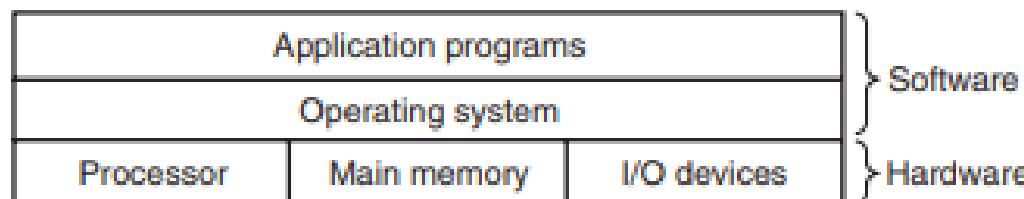
Role of Cache Memory

CPU chip



Operating System

- collection of software/ Program that acts as an intermediary between an user of a computer and the computer hardware.
- is a program that helps to run all the other programs
- Three main functions:
 - Resource management
 - Establish an user interface
 - Execute and provide services for application software



Layered view of a
computer system.

Main objectives

- Convenience
- Efficiency
- Ability to evolve and offer new services
- Maximize System performance
- Protection and access control
- Footprint of OS should be small

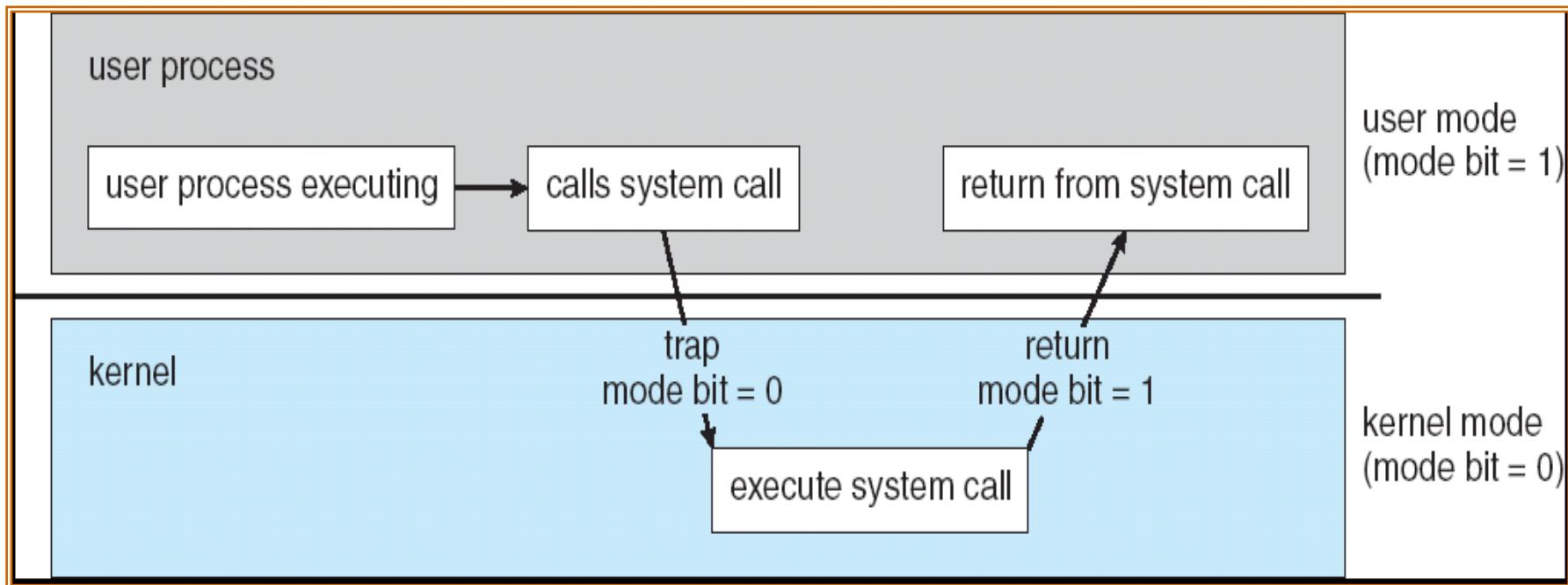
Important Note

- “The one program running at all times on the computer” is the **kernel**. Everything else is either a system program (ships with the operating system) or an application program
- **bootstrap program** is loaded at power-up or reboot
 - Typically stored in ROM or EPROM, generally known as **firmware**
 - Initializes all aspects of system
 - Loads operating system kernel and starts execution

Operating System Operations

- Dual-mode operation
 - User mode
 - Kernel mode (also known as System Mode / Supervisor mode/ privileged mode)
- User mode(1):
 - user program executes in user mode
 - certain areas of memory are protected from user access
 - certain privileged instructions may not be executed
- Kernel Mode (0)
 - privileged instructions may be executed
 - protected areas of memory may be accessed

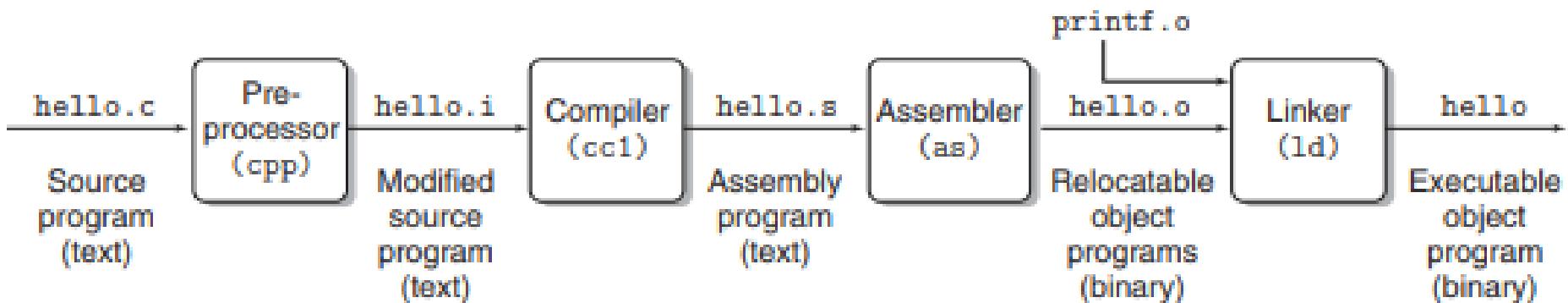
Transition from user to kernel mode



Running a Hello.c Program

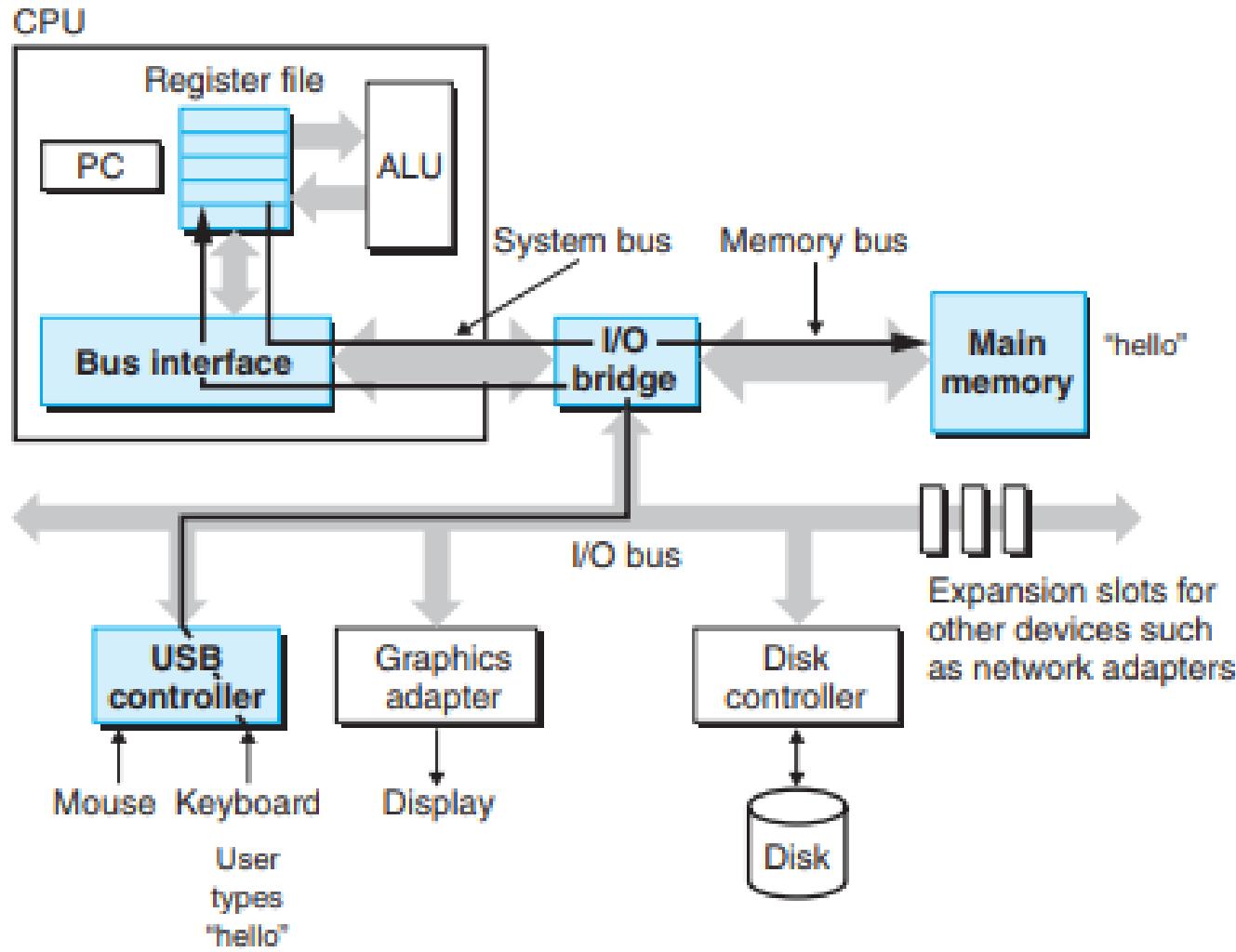
```
#include <stdio.h>

int main()
{
    printf("hello, world\n");
}
```

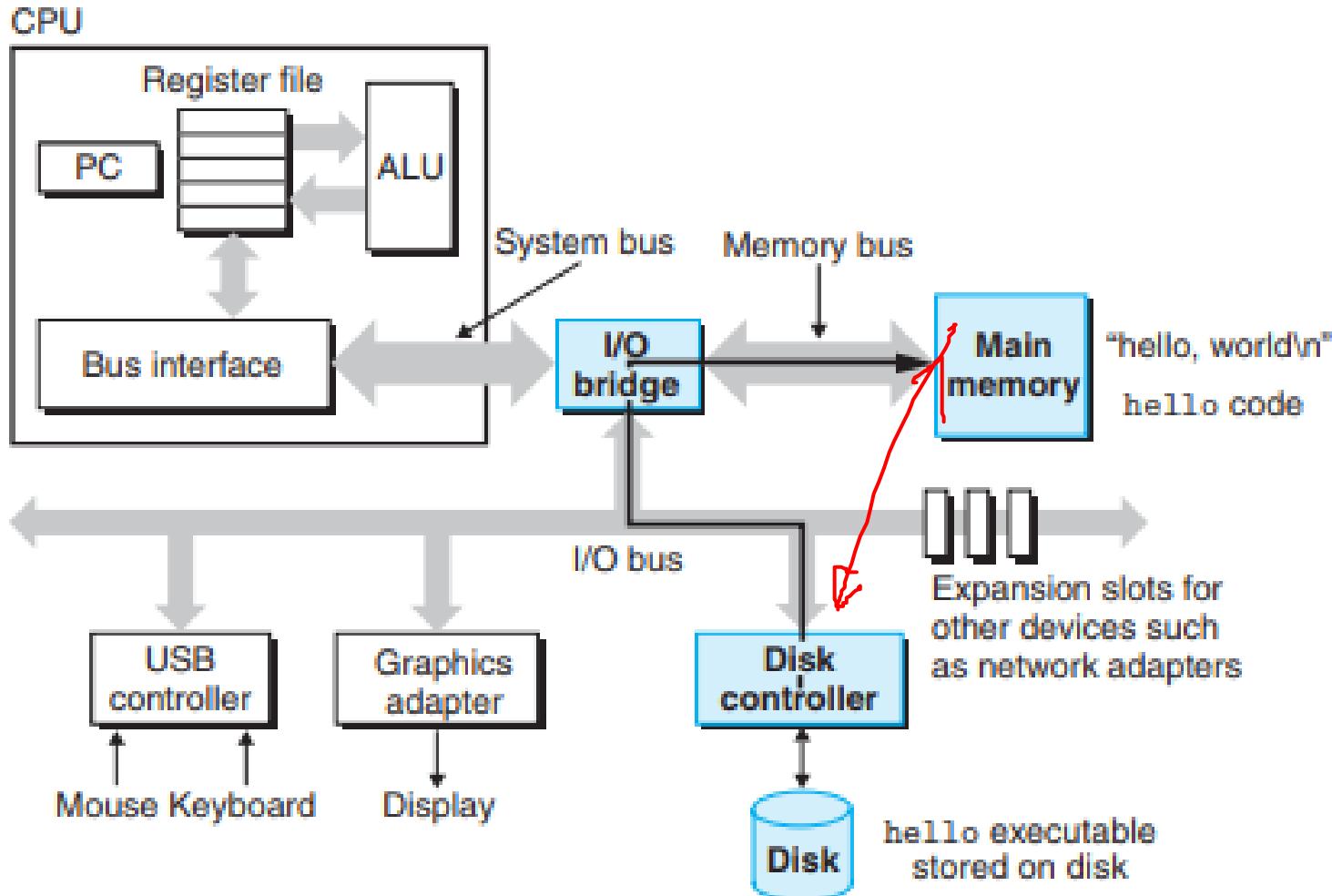


The compilation system.

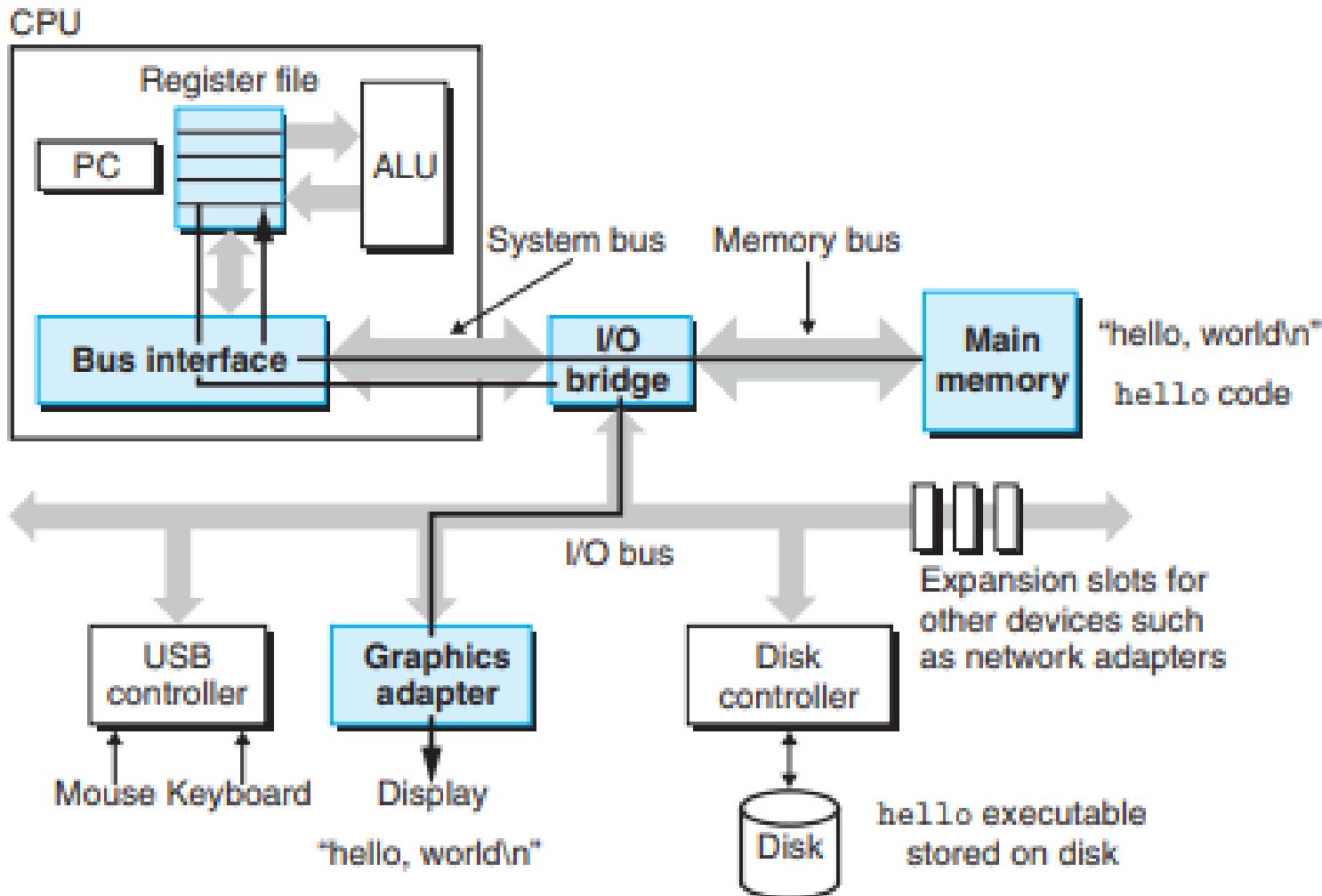
Reading ./hello command from Keyboard



Loading the executable from disk into main memory



Writing the output string from memory to the display





Why do we need to know how compilation works?

- Optimizing program performance.
- Understanding link-time errors
- Avoiding security holes.

Lab Activity

DH News: Latest & Breaking News, L x | Gmail Inbox (27,947) - lucy.gudino@pilani.bits-pilani.ac.in x | e-Learning Portal x | Wilp CS-IS Lab x +

Not secure | bitscsis.vlabs.platifi.com/index1.html#!/resources

BITS - Pilani Virtual Lab

Resources

Home / Computer Organization And Software Systems

- LabCapsule1-Introduction To CPU-OS Simulator
- LabCapsule2-Instruction Set Of CPU-OS Simulator
- LabCapsule3-Cache Memory
- LabCapsule4-Pipeline
- LabCapsule5-Process Management
- LabCapsule6-Multithreading
- LabCapsule7-Synchronization
- LabCapsule8-Deadlock

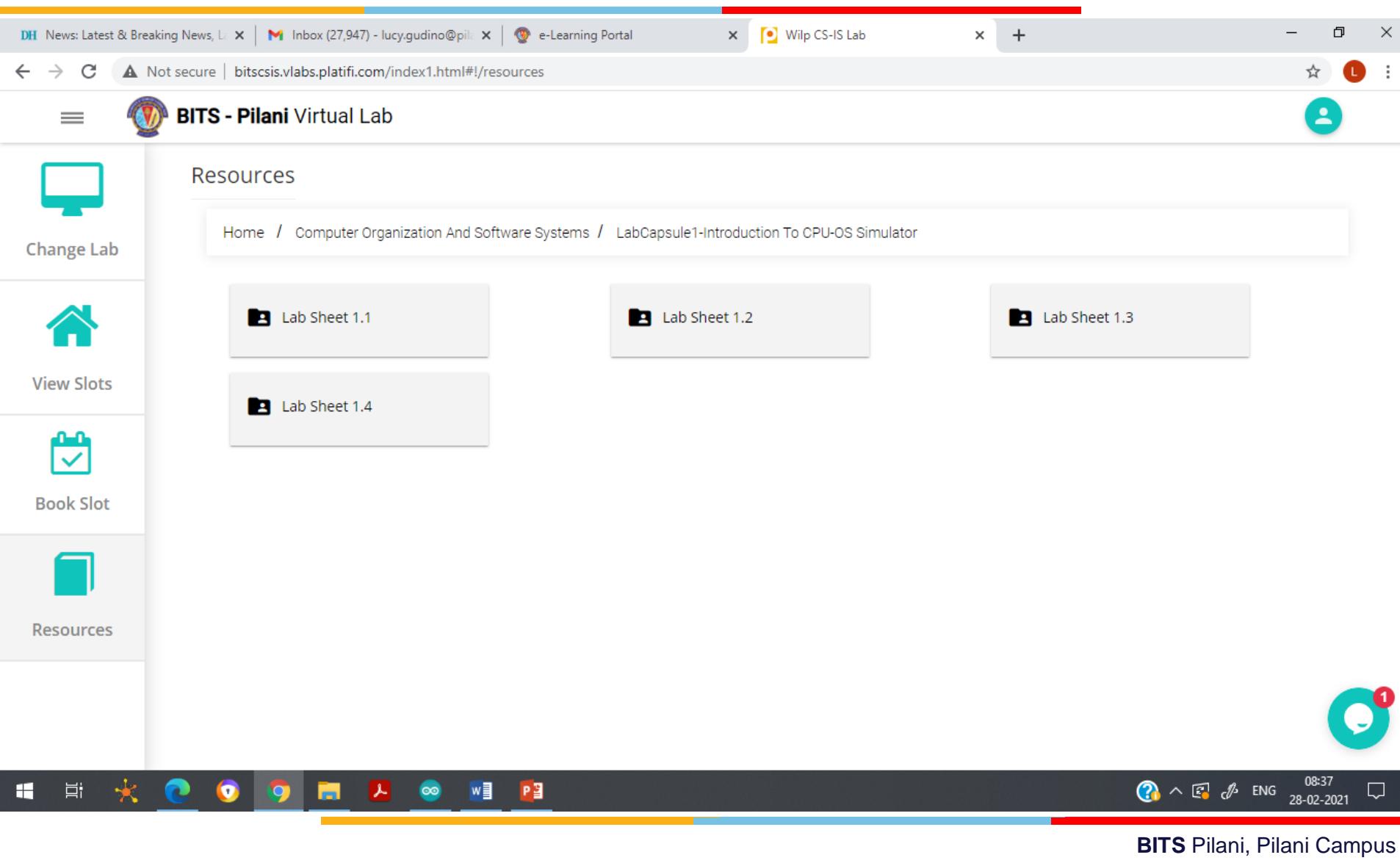
Welcome to our site, if you need help simply reply to this message, we are online and ready to help.
Customer Support just now

Type here and press enter..

08:36 28-02-2021 ENG

BITS Pilani, Pilani Campus

Contd...



DH News: Latest & Breaking News, Li × | Gmail Inbox (27,947) - lucy.gudino@pilani.ac.in × | e-Learning Portal × | Wilp CS-IS Lab × | +

Not secure | bitscsis.vlabs.platifi.com/index1.html#!/resources

BITS - Pilani Virtual Lab

Resources

Home / Computer Organization And Software Systems / LabCapsule1-Introduction To CPU-OS Simulator

Lab Sheet 1.1

Lab Sheet 1.2

Lab Sheet 1.3

Lab Sheet 1.4

Change Lab

View Slots

Book Slot

Resources

08:37 28-02-2021 ENG

BITS Pilani, Pilani Campus

Contd...

Screenshot of a web browser showing the BITS - Pilani Virtual Lab interface.

The browser tabs are:

- DH News: Latest & Breaking News, L ...
- Inbox (27,947) - lucy.gudino@pil...
- e-Learning Portal
- Wilp CS-IS Lab

The address bar shows: bitscsis.vlabs.platifi.com/index1.html#/resources

The page title is: BITS - Pilani Virtual Lab

The main content area displays:

- Resources
- Breadcrumbs: Home / Computer Organization And Software Systems / LabCapsule1-Introduction To CPU-OS Simulator / Lab Sheet 1.1
- Two items:
 - Lab Sheet 1.1_Introducti... (document icon)
 - Lab Video (video camera icon)

The left sidebar menu includes:

- Change Lab
- View Slots
- Book Slot
- Resources

A teal circular notification badge in the bottom right corner indicates 1 unread message.

The taskbar at the bottom shows various pinned icons and the system tray with the date and time (08:37, 28-02-2021).



BITS Pilani
Pilani Campus

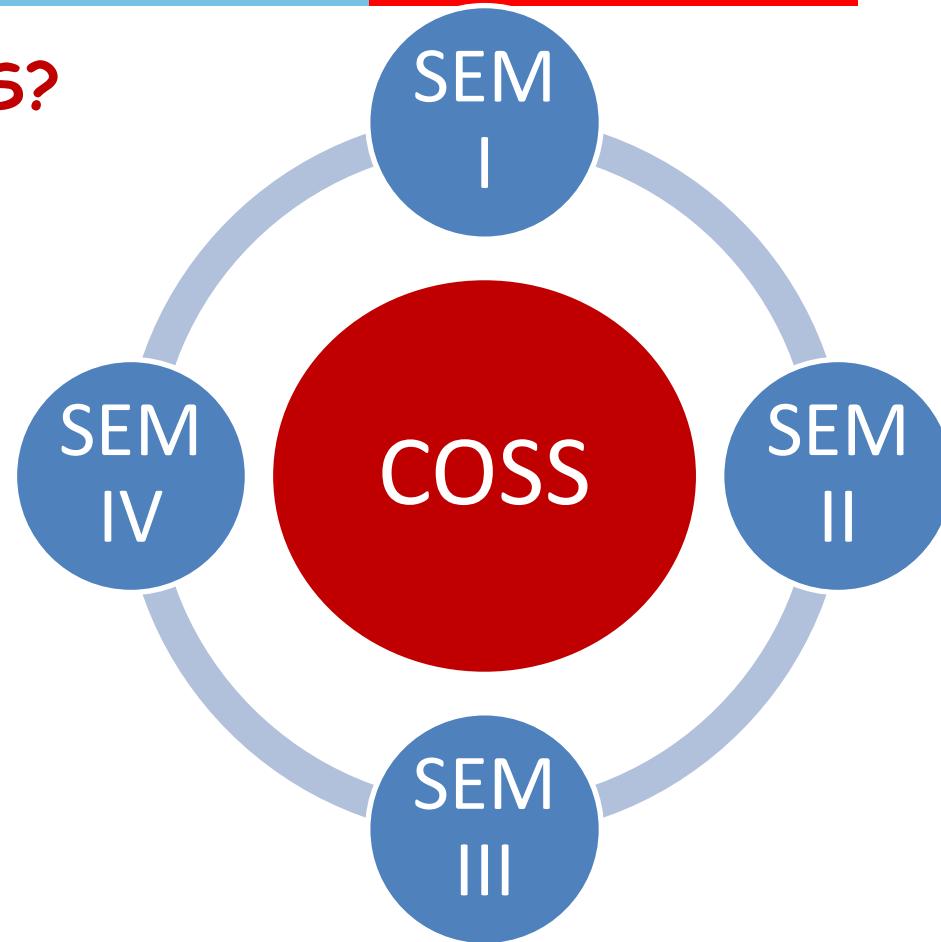
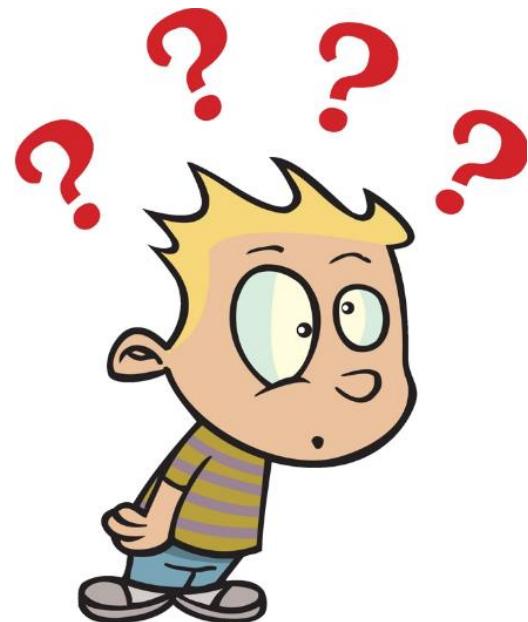
Computer Organization and Software Systems

CONTACT SESSION 1

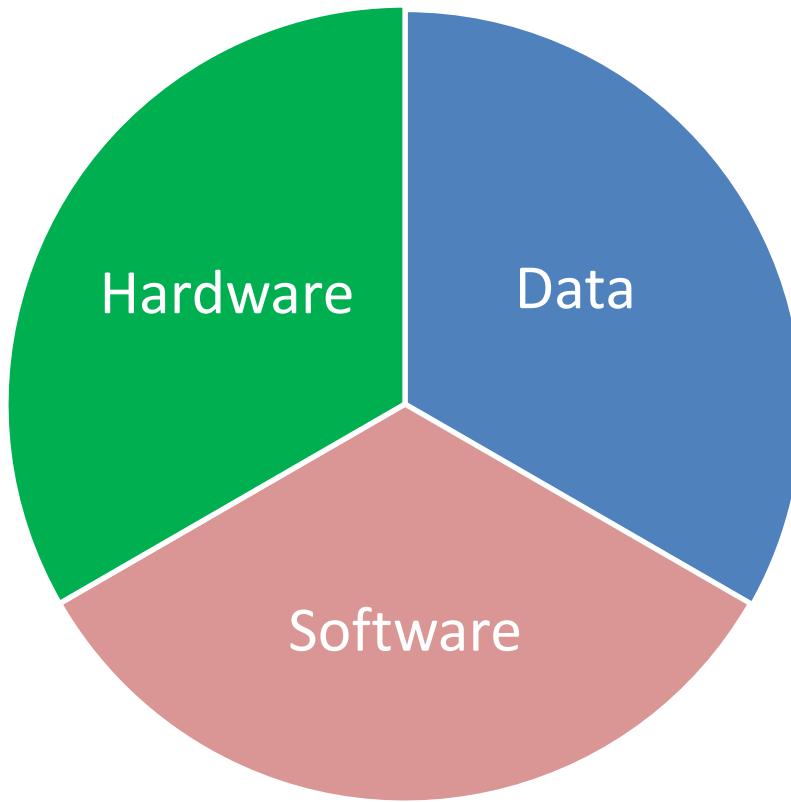
Prepared by Dr. Lucy J. Gudino
Instructor Prof. C R Sarma

Introduction

Why Study **COSS**?



Introduction



Data analytics: is the process of examining **data sets** in order to draw conclusions about the information they contain, increasingly with the aid of **specialized systems** and **software**.

Text Books and Reference Books

Text Books:

- (T1) W. Stallings, *Computer Organization & Architecture*, PHI, 10th ed., 2010.
- (T2) A Silberschatz, Abraham and others, *Operating Systems Concepts*, Wiley Student Edition, 8th Edition

Reference Books:

- (R1) Patterson, David A & J L Hennenssy, *Computer Organization and Design – The Hardware/Software Interface*, Elsevier, 5th Ed., 2014.
- (R2) Randal E. Bryant, David R. O'Hallaron, *Computer Systems – A Programmer's Perspective*, Pearson, 3rd Ed, 2016.
- (R3) Tanenbaum, *Modern Operating Systems*: Pearson New International Edition, Pearson Education, 2013 (Pearson Online)
- (R4) Stallings, *Operating Systems: Internals and Design Principles* : International Edition, Pearson Education, 2013 (Pearson Online)

Evaluation Scheme

5 unit course.

Sl No.	Evaluation Component	Duration	Weightage %	Nature of Component
1	Mid Sem Exam	90 min	30%	Open Book
2	Comprehensive Examination	180 min	40%	OB
3	Quiz - 2	-----	5%	OB
4	Assignments	---	25%	OB

Assignments

- Two assignments:
 - One pre-midsem exam : 12%
 - One post-midsem : 13%
- Lab based
- Simulator to be used : CPU-OS simulator
 - Open source tool
https://drive.google.com/open?id=12YUK52RQ-JhPOddj6CD_oifW4sTMbsBI
 - Virtual lab (Platifi)

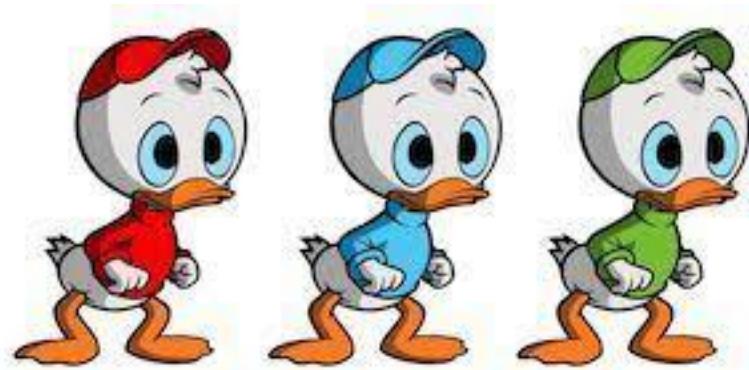
Assignment should not be

FILL IT.



SHUT IT.

FORGET IT.



General Instructions

1. Always use note book for writing important points and for solving problems
2. Use chat box for writing subject related questions
3. Do not repeat the questions on chat box. Questions will be answered during last 10 minutes of the session
4. Unanswered questions will be put up on the canvas forum

Today's Session

Contact Hour	List of Topic Title	Text/Ref Book/external resource
1-2	<p>Introduction to Computer Systems</p> <ul style="list-style-type: none"> • Hardware Organization of a computer • Basic uniprocessor architecture • Instruction Cycle State Diagram • Operating System role in Managing Hardware • Running a Hello Program 	T1

Definition of a Computer

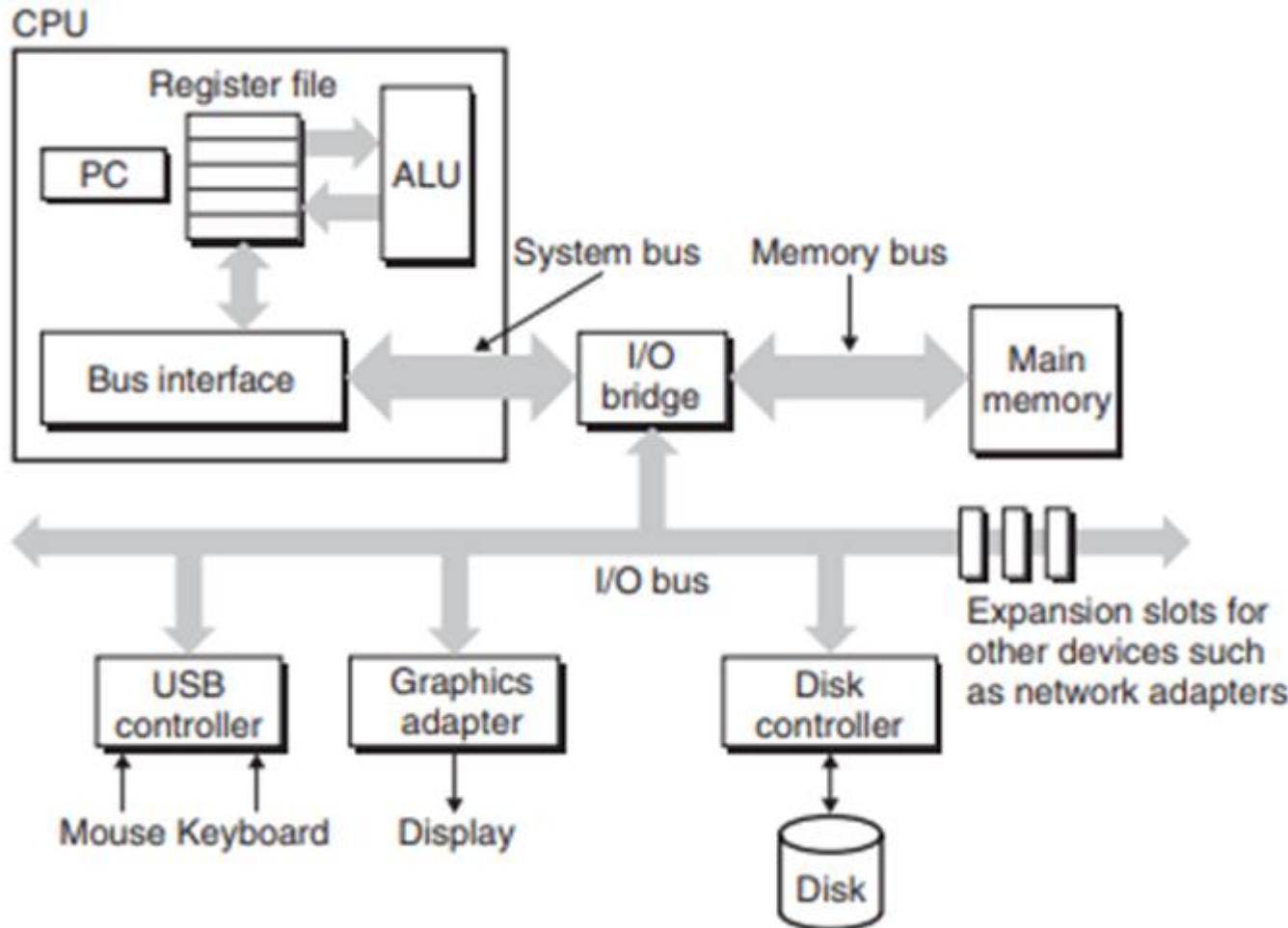
- Is a complex system
- Is a programmable device
- Must be able to **process** data
- Must be able to **store** data
- Must be able to **move** data
- Must be able to **control** above three functions

Computer System



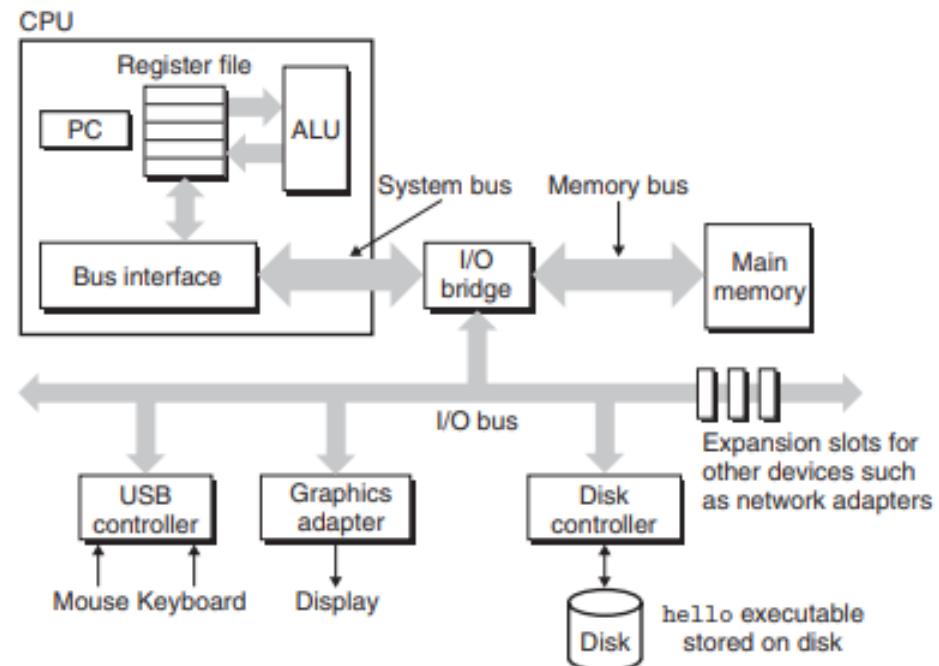
- Hardware
 - Central Processing Unit (CPU)
 - Memory
 - I/O devices
- Software
 - System Software
 - System Management Software
 - Tools and Utilities for Developing the software
 - Application Software
 - General Purpose Software
 - Specific Purposed Software

Hardware Organization of a computer



Von Neumann Architecture

- Three key concepts:
 - Data and instructions are stored in a single read - write memory
 - The contents of this memory are addressable by location, without regard to the type of data contained there
 - Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next

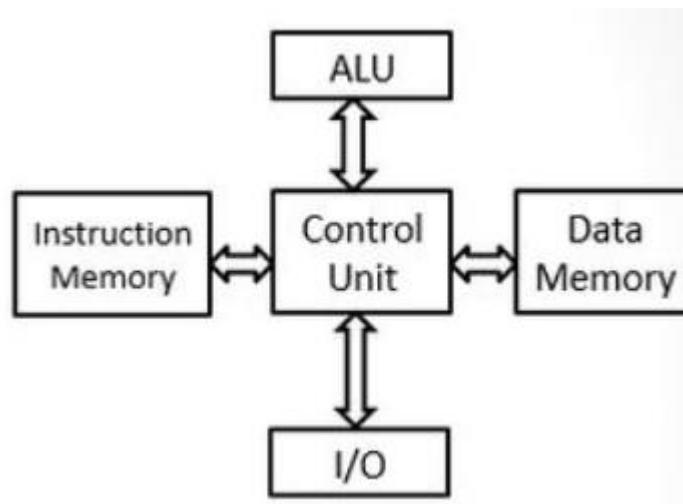


Von Neumann Architecture...

- Stored-program computers have the following characteristics:
 - Three hardware systems:
 - A central processing unit (CPU)
 - A main memory system
 - An I/O system
 - The capacity to carry out sequential instruction processing.
 - A single path between the CPU and main memory.
 - This single path is known as the **von Neumann bottleneck**.
 - Side effect : reduced throughput (Data Rate)

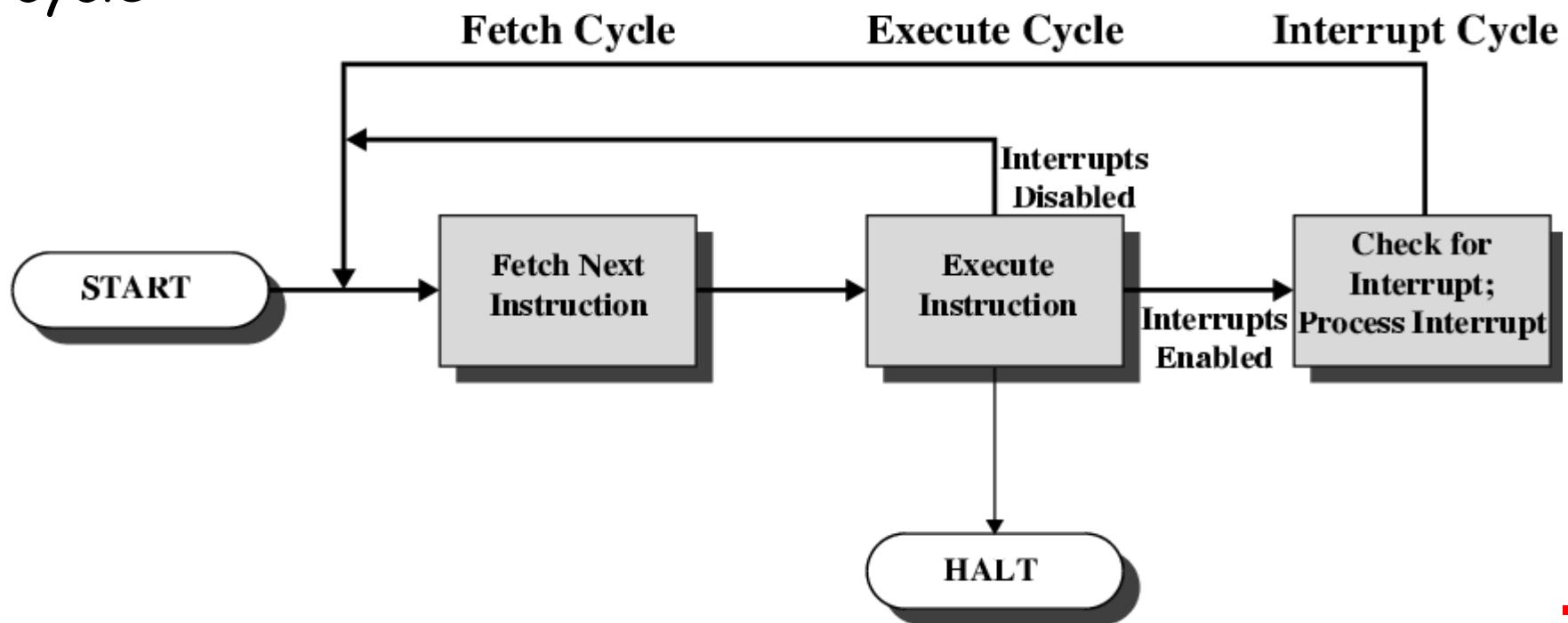
Harvard Architecture

- Uses two memory systems and two separate busses
 - Instruction Memory
 - Data Memory



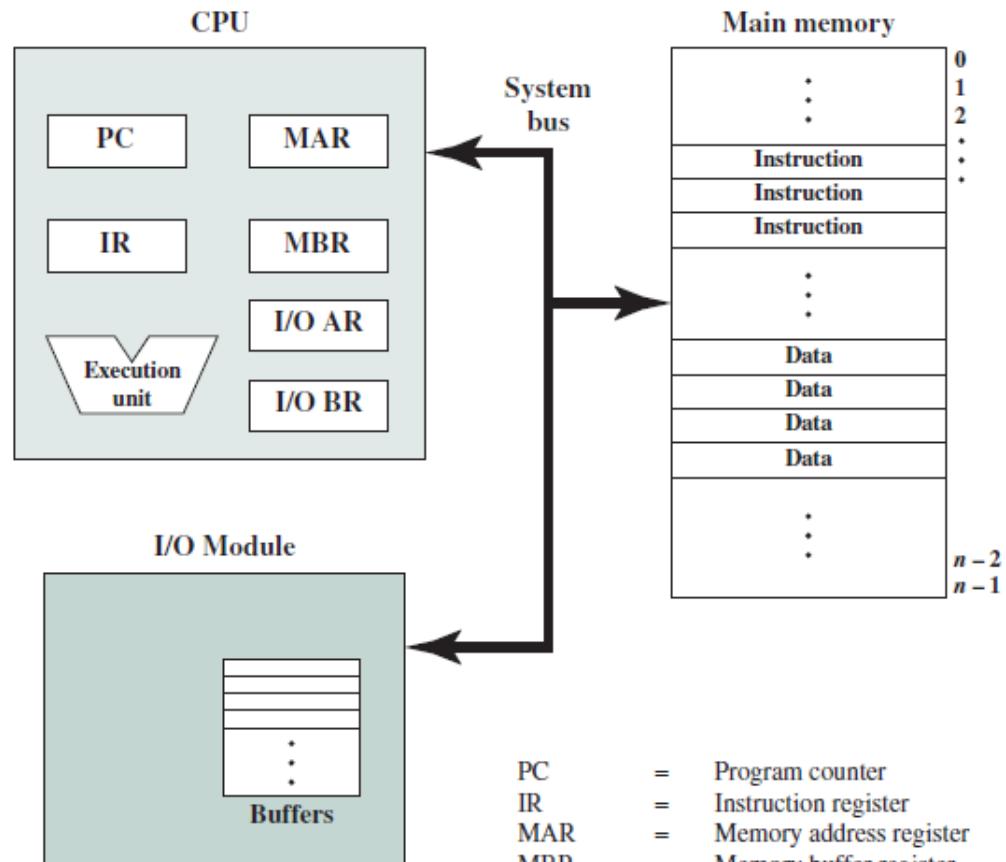
Instruction Cycle Diagram

- Instruction execution : Two steps:
 - Fetch
 - Execute
- Interrupt: Interrupt is checked at the end of Instruction cycle



Fetch Cycle

- Program Counter (PC) holds address of next instruction to be fetched
- Processor fetches instruction from memory location pointed to by PC
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions during execution cycle
- Increment PC
 - Unless told otherwise



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

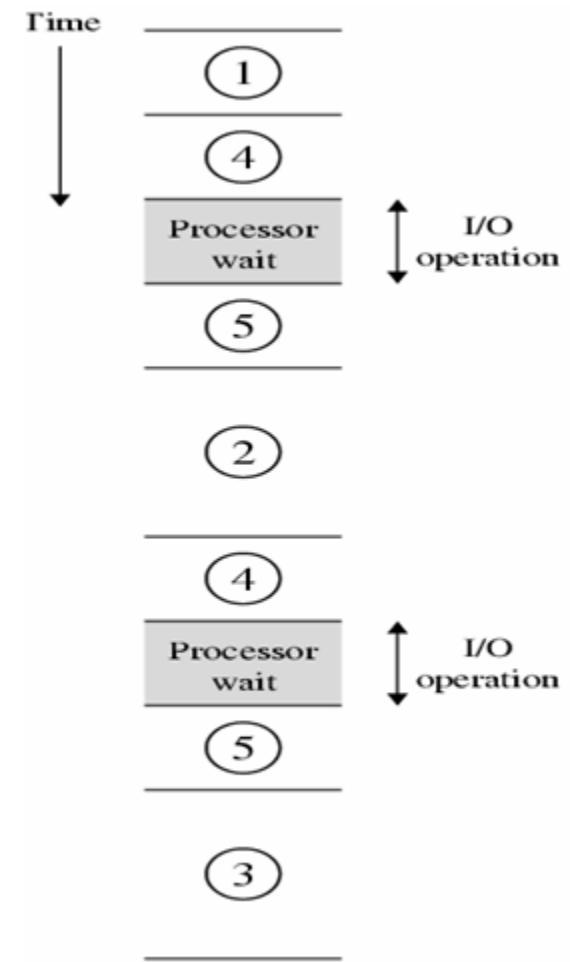
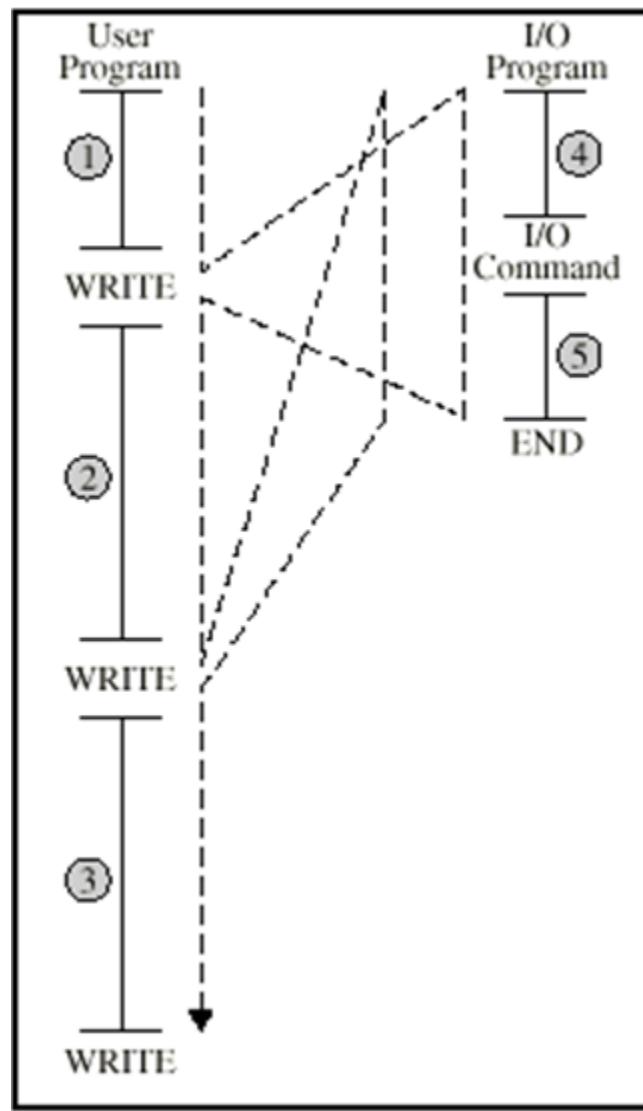
Execute Cycle

- Processor - memory
 - Data transfer between CPU and main memory
- Processor - I/O
 - Data transfer between CPU and I/O module
- Data processing
 - Some arithmetic or logical operation on data
- Control
 - Alteration of sequence of operations
 - e.g. jump
- Combination of above

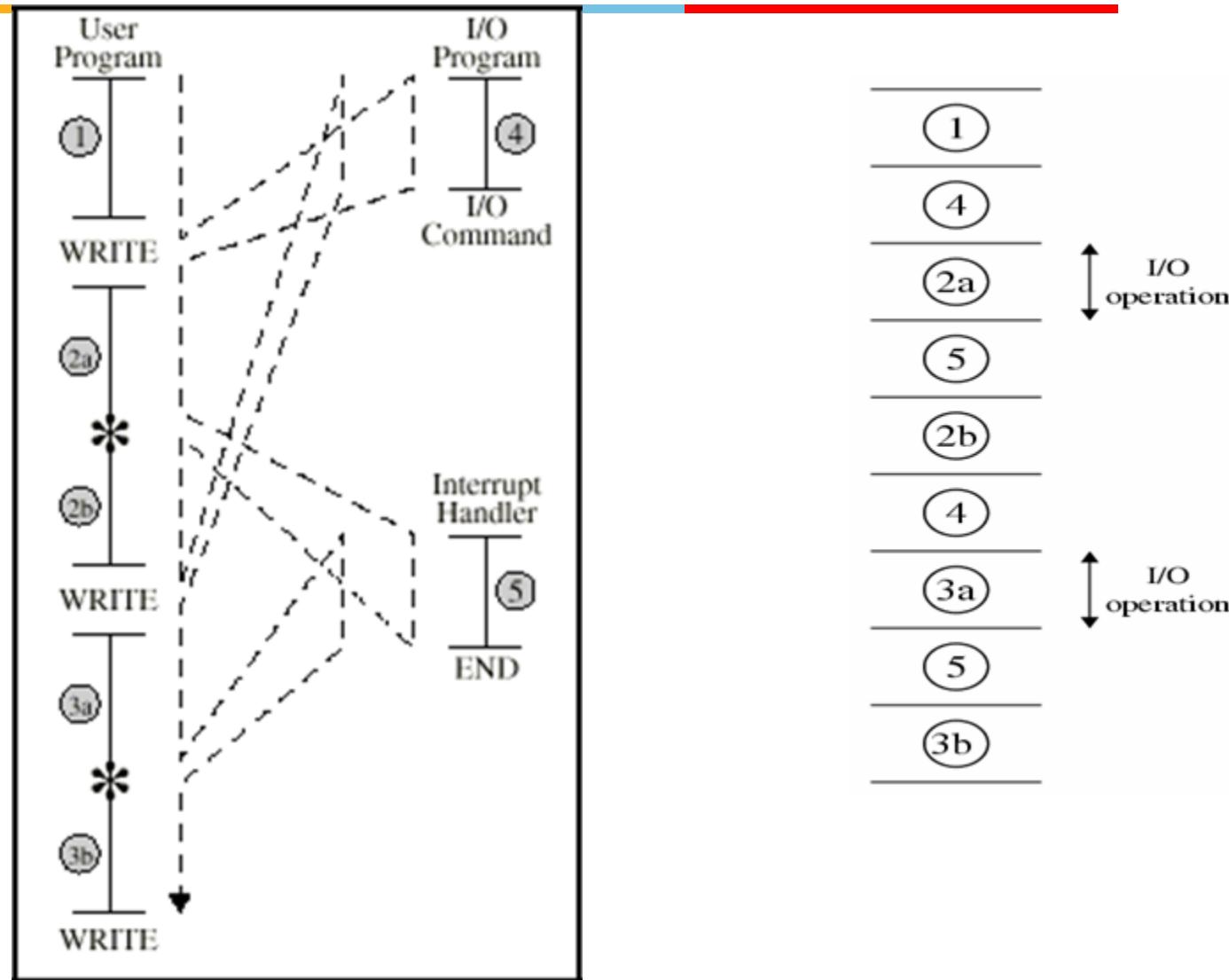
Interrupt Cycle

- Interrupts: Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Interrupts enhances processing efficiency
- Processor checks for interrupt
 - Indicated by an interrupt signal
- If no interrupt, fetch next instruction
- If interrupt pending:
 - Suspend execution of current program
 - Save context
 - Set PC to start address of interrupt handler routine
 - Process interrupt
 - Restore context and continue interrupted program

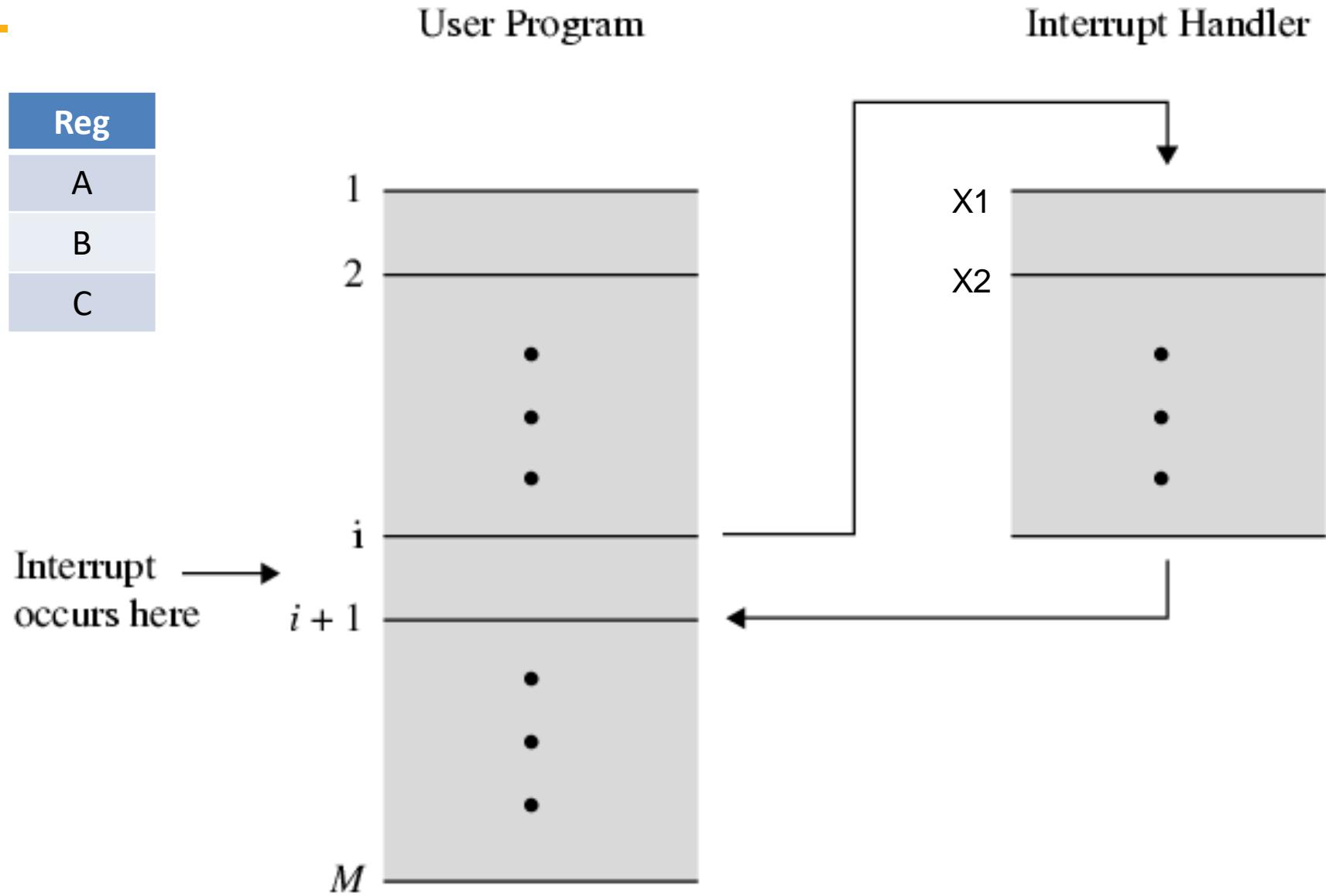
Program Flow Control (No Interrupts)



Program Flow Control (With Interrupts)



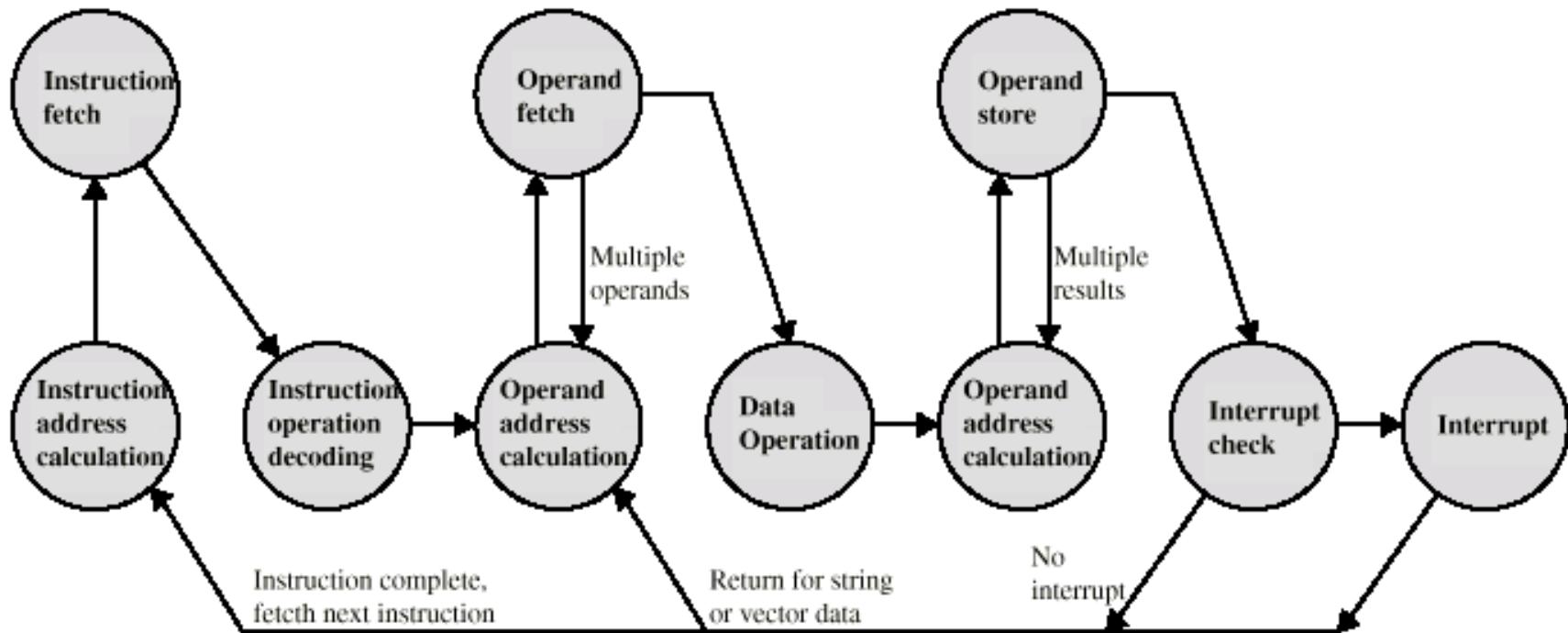
Transfer of Control via Interrupts



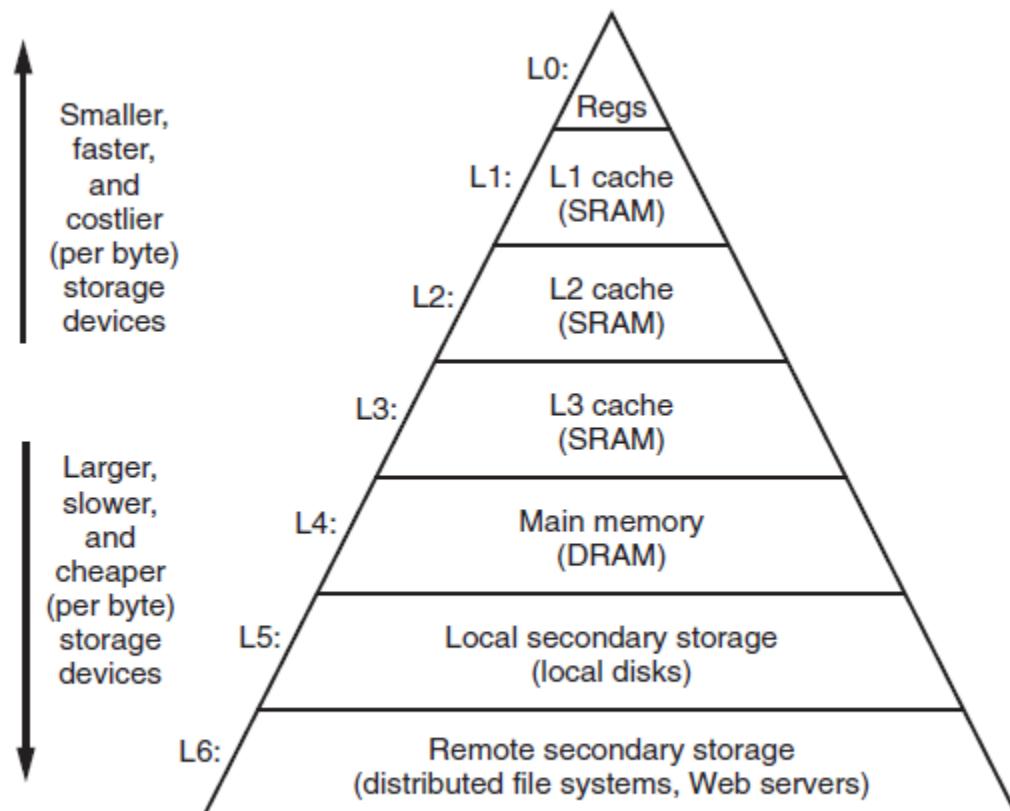
Types of Interrupts

- Types of interrupts:
 - Program
 - e.g. overflow, division by zero
 - Timer
 - Generated by internal processor timer
 - Used in pre-emptive multi-tasking
 - I/O
 - from I/O controller
 - Hardware failure
 - e.g. memory parity error

Instruction Cycle - State Diagram

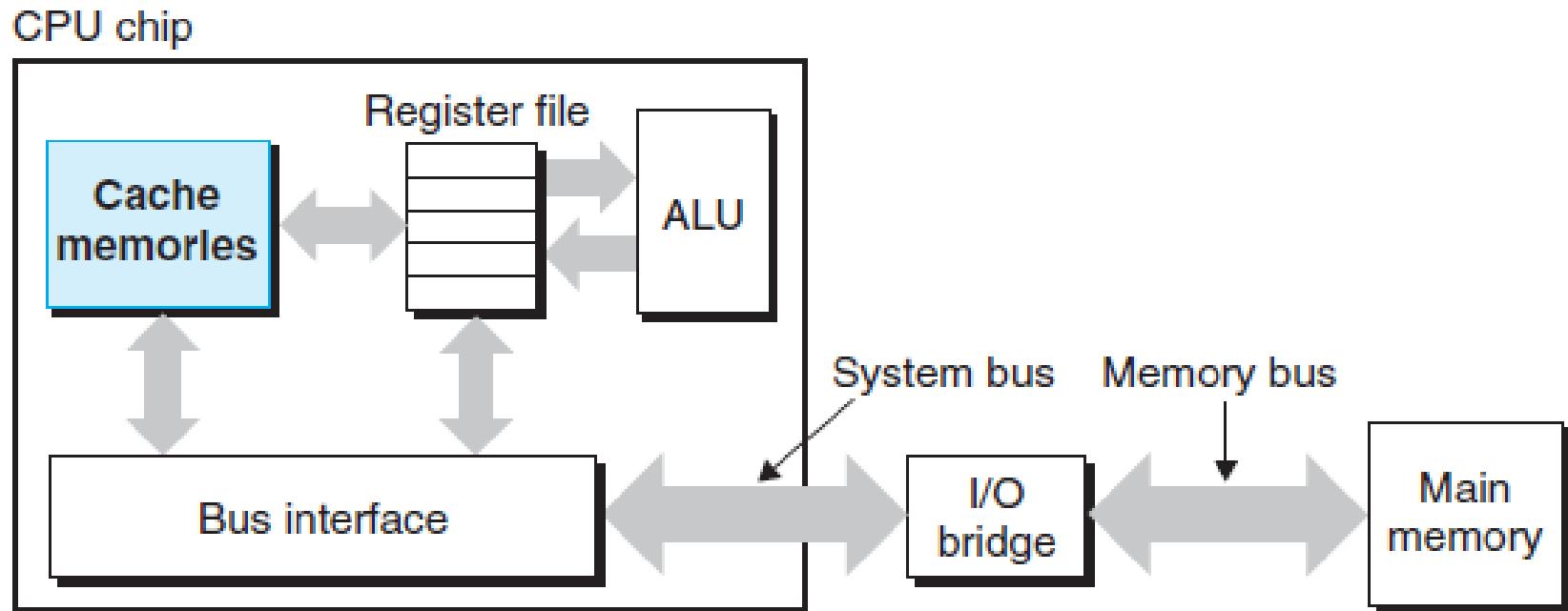


Memory Hierarchy



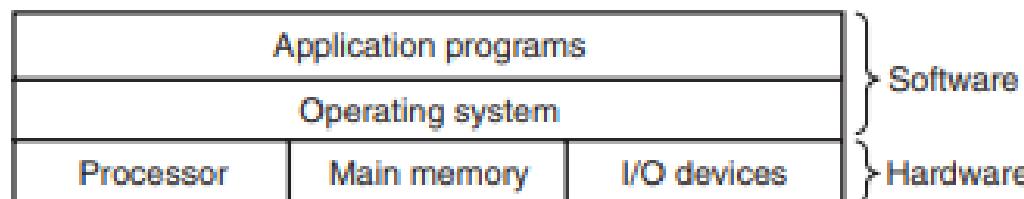
An example of a memory hierarchy.

Role of Cache Memory



Operating System

- collection of software/ Program that acts as an intermediary between an user of a computer and the computer hardware.
- is a program that helps to run all the other programs
- Three main functions:
 - Resource management
 - Establish an user interface
 - Execute and provide services for application software



Layered view of a
computer system.

Main objectives

- Convenience
- Efficiency
- Ability to evolve and offer new services
- Maximize System performance
- Protection and access control
- Footprint of OS should be small

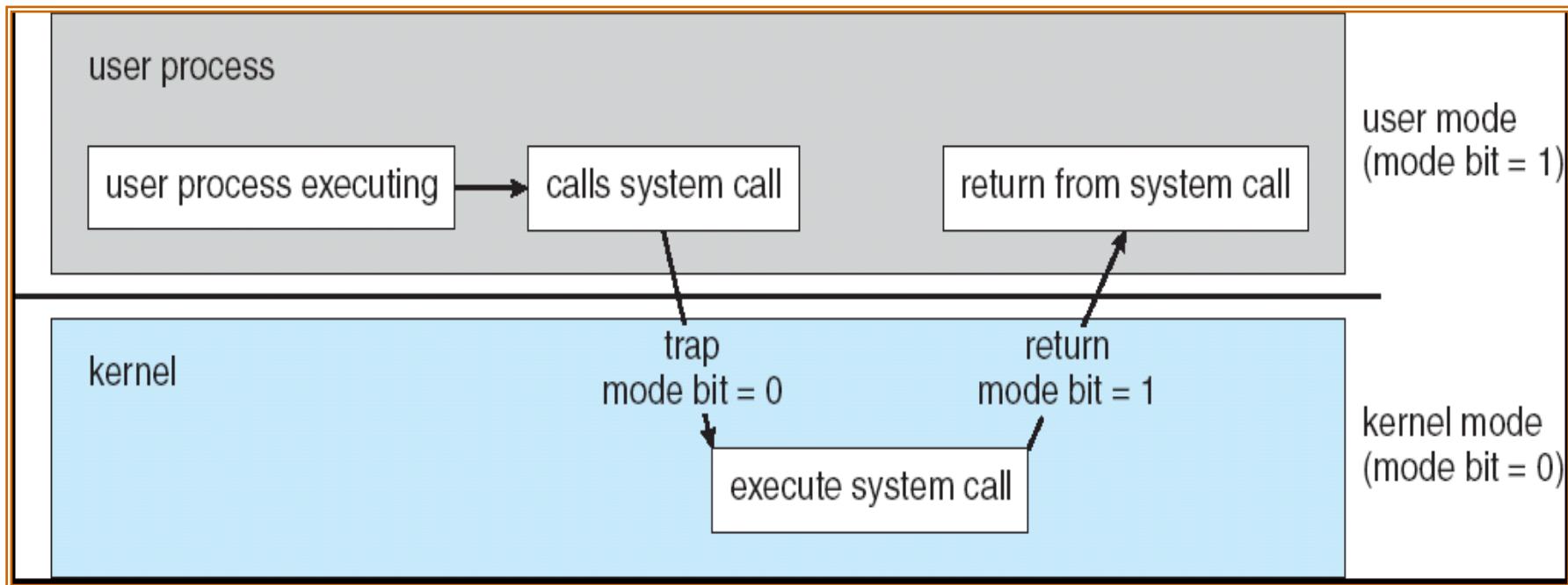
Important Note

- “The one program running at all times on the computer” is the **kernel**. Everything else is either a system program (ships with the operating system) or an application program
- **bootstrap program** is loaded at power-up or reboot
 - Typically stored in ROM or EPROM, generally known as **firmware**
 - Initializes all aspects of system
 - Loads operating system kernel and starts execution

Operating System Operations

- Dual-mode operation
 - User mode
 - Kernel mode (also known as System Mode / Supervisor mode/ privileged mode)
- User mode(1):
 - user program executes in user mode
 - certain areas of memory are protected from user access
 - certain privileged instructions may not be executed
- Kernel Mode (0)
 - privileged instructions may be executed
 - protected areas of memory may be accessed

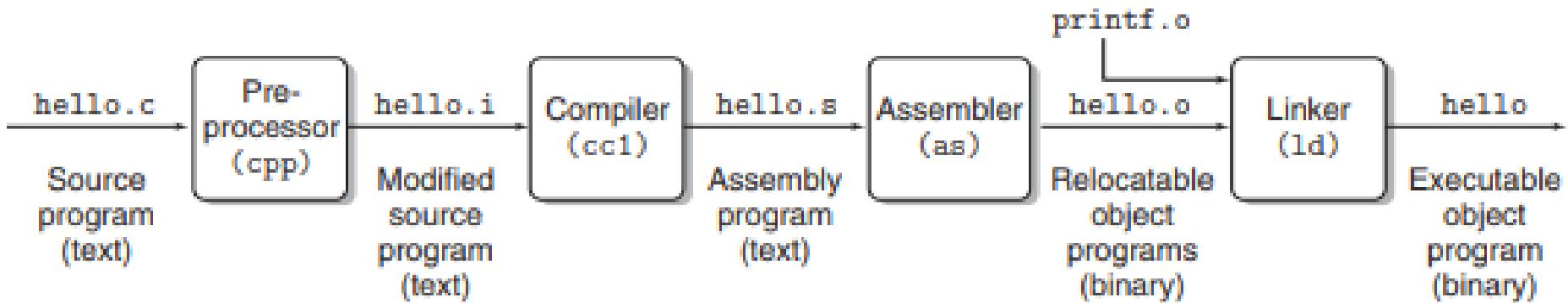
Transition from user to kernel mode



Running a Hello.c Program

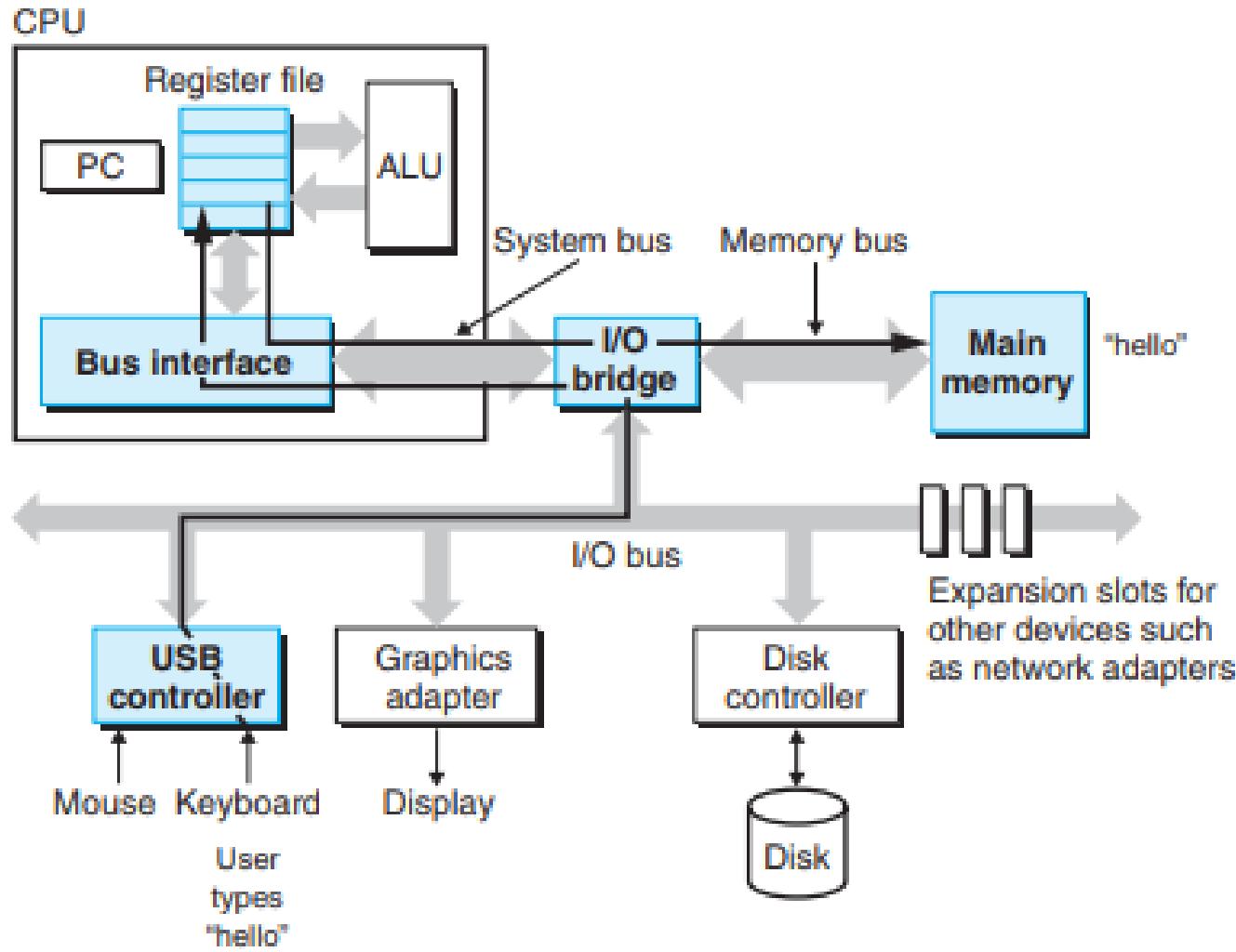
```
#include <stdio.h>

int main()
{
    printf("hello, world\n");
}
```

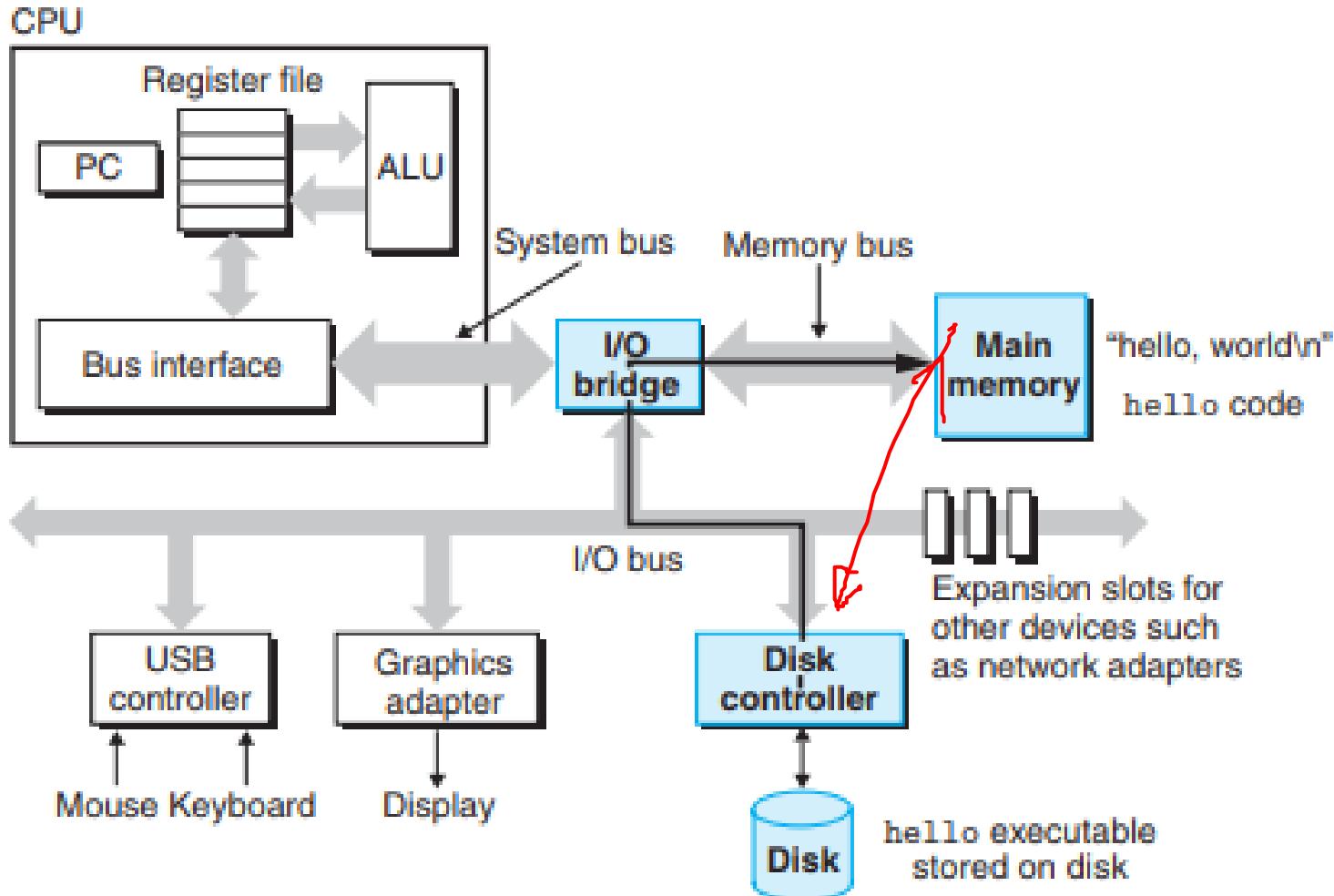


The compilation system.

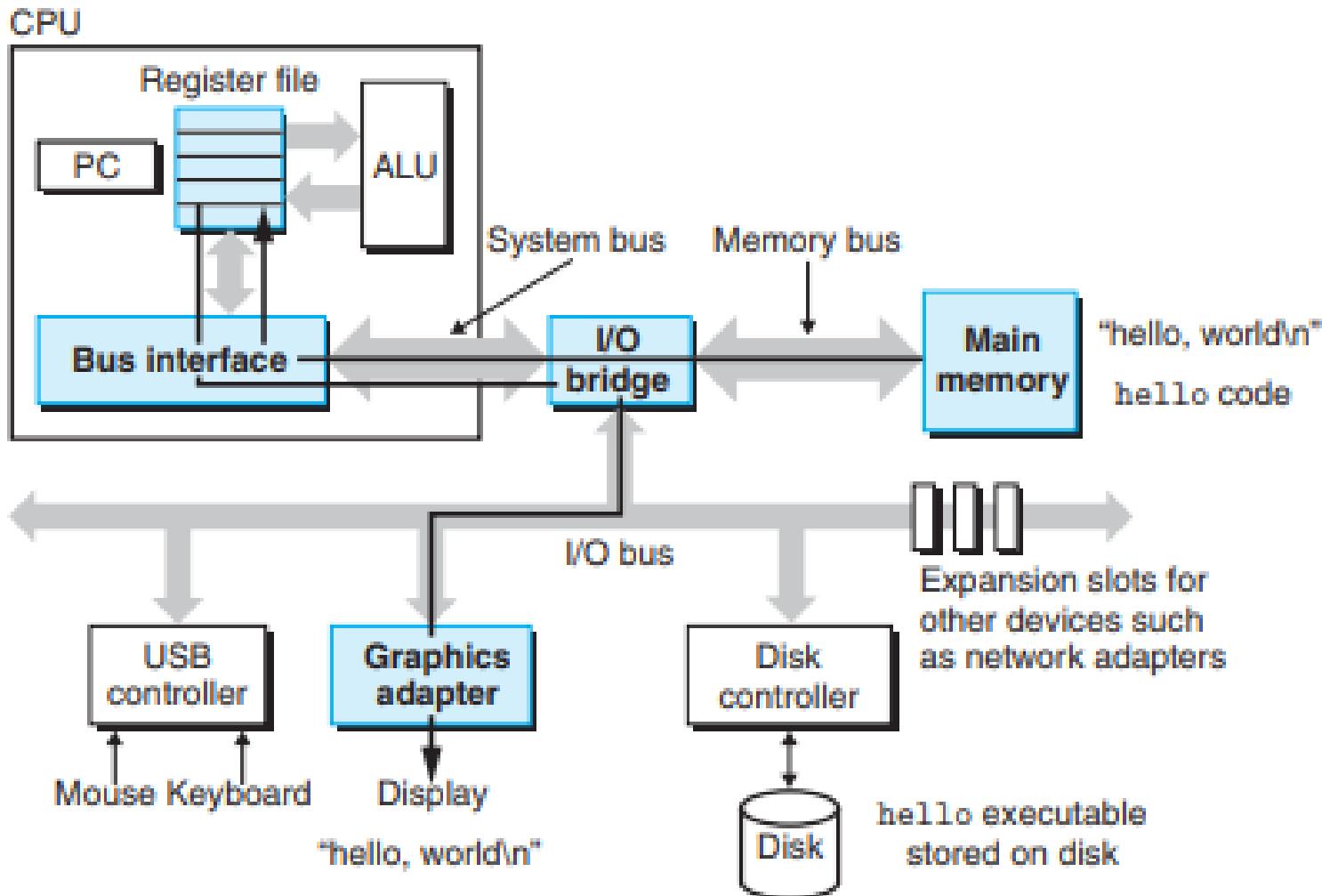
Reading ./hello command from Keyboard



Loading the executable from disk into main memory



Writing the output string from memory to the display





Why do we need to know how compilation works?

- Optimizing program performance.
 - Understanding link-time errors
 - Avoiding security holes.
-

Lab Activity

DH News: Latest & Breaking News, L x | Gmail Inbox (27,947) - lucy.gudino@pilani.bits-pilani.ac.in x | e-Learning Portal x | Wilp CS-IS Lab x +

Not secure | bitscsis.vlabs.platifi.com/index1.html#!/resources

BITS - Pilani Virtual Lab

Resources

Home / Computer Organization And Software Systems

- LabCapsule1-Introduction To CPU-OS Simulator
- LabCapsule2-Instruction Set Of CPU-OS Simulator
- LabCapsule3-Cache Memory
- LabCapsule4-Pipeline
- LabCapsule5-Process Management
- LabCapsule6-Multithreading
- LabCapsule7-Synchronization
- LabCapsule8-Deadlock

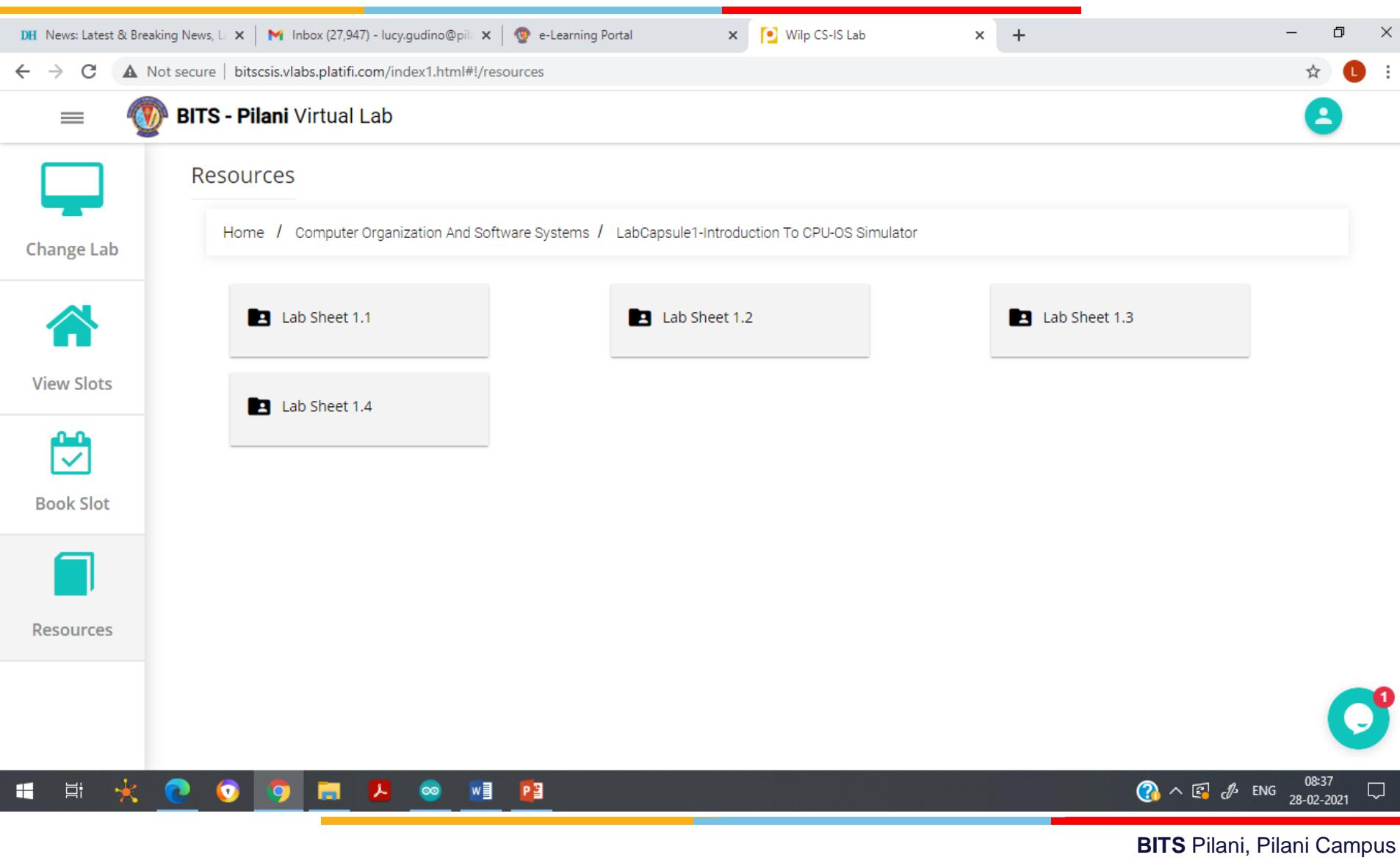
Welcome to our site, if you need help simply reply to this message, we are online and ready to help.
Customer Support just now

Type here and press enter..

08:36 28-02-2021 ENG

BITS Pilani, Pilani Campus

Contd...



DH News: Latest & Breaking News, Li × | Gmail Inbox (27,947) - lucy.gudino@pilani.ac.in × | e-Learning Portal × | Wilp CS-IS Lab × | +

Not secure | bitscsis.vlabs.platifi.com/index1.html#!/resources

BITS - Pilani Virtual Lab

Resources

Home / Computer Organization And Software Systems / LabCapsule1-Introduction To CPU-OS Simulator

Lab Sheet 1.1

Lab Sheet 1.2

Lab Sheet 1.3

Lab Sheet 1.4

Change Lab

View Slots

Book Slot

Resources

08:37 28-02-2021 ENG

BITS Pilani, Pilani Campus

Contd...

Screenshot of a web browser showing the BITS - Pilani Virtual Lab interface.

The browser tabs are:

- DH News: Latest & Breaking News, L ...
- Inbox (27,947) - lucy.gudino@pil...
- e-Learning Portal
- Wilp CS-IS Lab

The address bar shows: bitscsis.vlabs.platifi.com/index1.html#/resources

The page title is: BITS - Pilani Virtual Lab

The main content area displays:

- Resources
- Home / Computer Organization And Software Systems / LabCapsule1-Introduction To CPU-OS Simulator / Lab Sheet 1.1
- Lab Sheet 1.1_Introducti... (document icon)
- Lab Video (video camera icon)

The left sidebar menu includes:

- Change Lab
- View Slots
- Book Slot
- Resources

A teal circular notification badge in the bottom right corner indicates 1 unread message.

The taskbar at the bottom shows various pinned application icons and system status indicators.



BITS Pilani
Pilani Campus

Computer Organization and Software Systems

CONTACT SESSION 2

Prepared By: Dr. Lucy J. Gudino
Instructor: Prof. C R Sarma

Today's Class

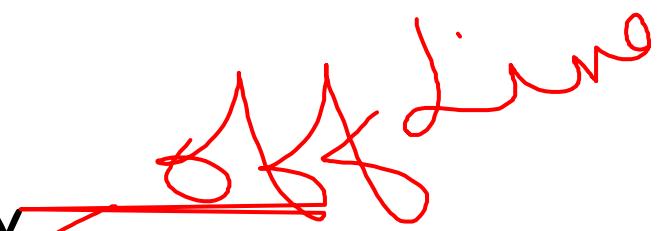
Contact Hour	List of Topic Title	Text/Ref Book/external resource
3-4	Memory Organization - Internal Memory - External Memory	T1, R2



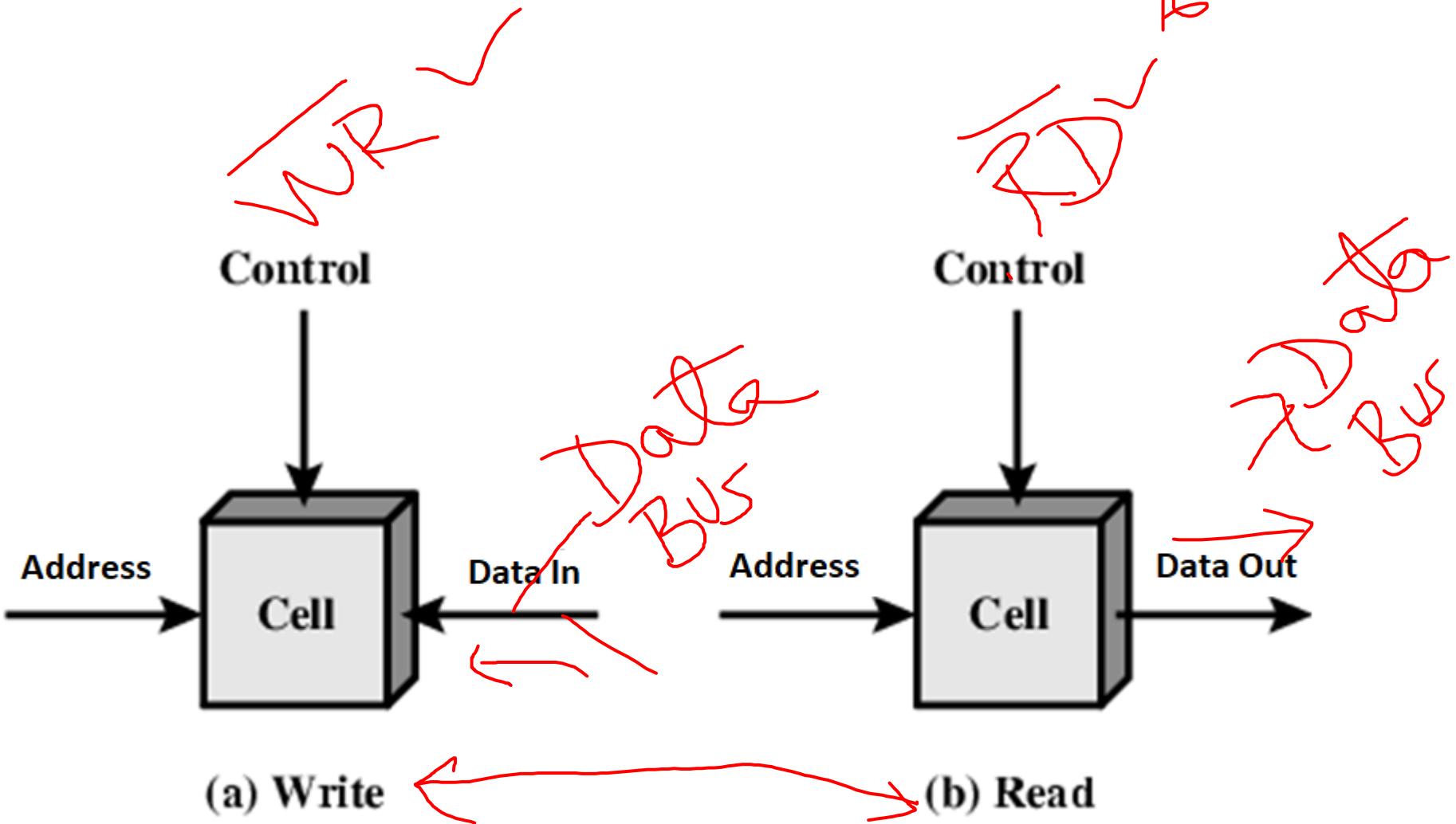
BITS Pilani
Pilani Campus

Memory Organization

Internal Memory Organization

- Internal Memory
 - Also known as main memory or Primary Memory
 - Small data storage but quick access.
 - Examples : RAM, ROM
 - External Memory
 - Also Known as secondary Memory
 - Huge data stored persistently
 - Examples: hard disk, solid state drives, USB flash drives etc.
- 

Semiconductor Memory



Random-Access Memory (RAM)

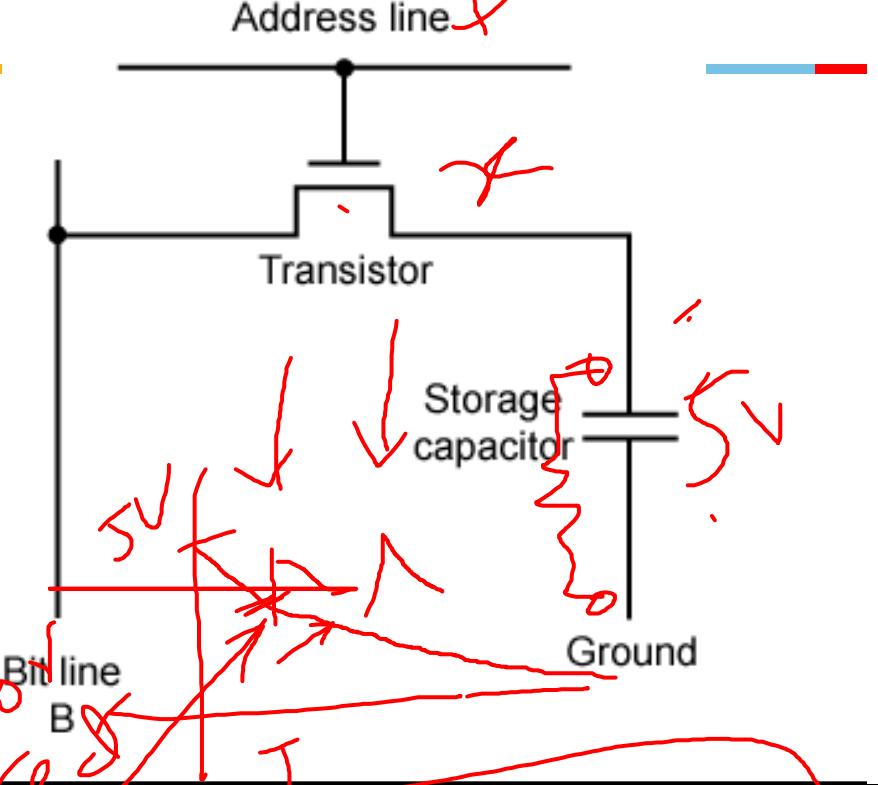
- Key features
 - RAM is traditionally packaged as a chip.
 - Basic storage unit is normally a cell (one bit per cell).
 - Multiple RAM chips form a memory.
- RAM comes in two varieties:
 - SRAM (Static RAM)
 - DRAM (Dynamic RAM)
- SRAM and DRAM are volatile memories
 - Loose information if powered off.

SRAM vs DRAM Summary



~~DRAM~~

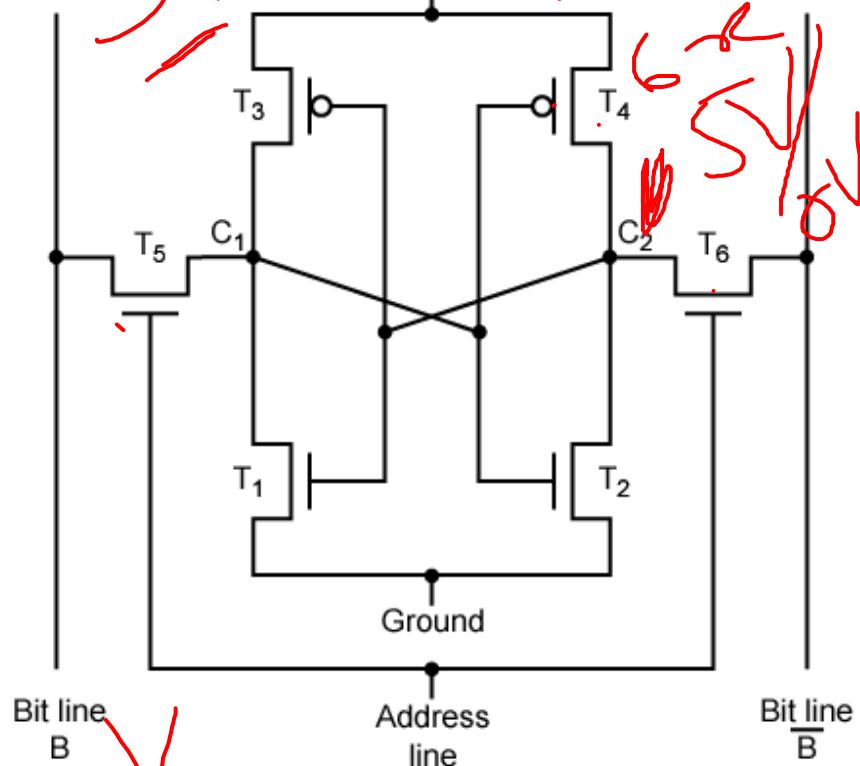
Address line



~~SRAM~~

dc voltage

~~TX~~



Trans. time
per bit

Access time

Needs refresh?

Needs EDC?

Cost

Applications

SRAM

4 to 6

1X

No

Maybe

100x

Cache

DRAM

1

10X

Yes

Yes

1X

Main memories,
frame buffers

Read Only Memory

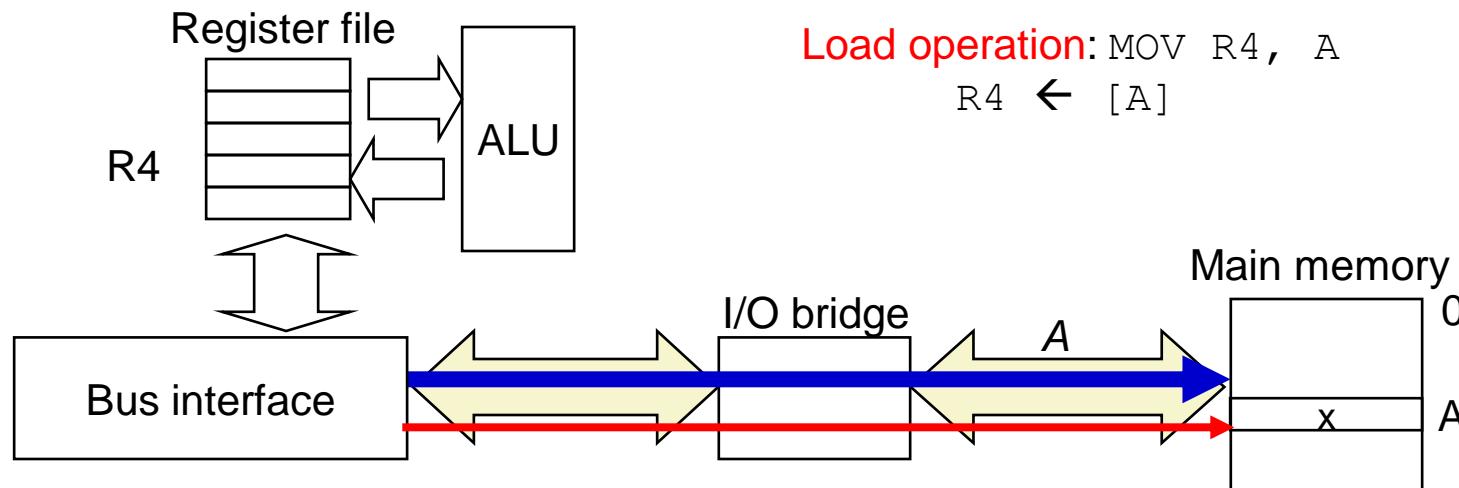
- Permanent Storage and Nonvolatile Memories
 - Read Only Memory Variants:
 - Read-only memory (**ROM**): programmed during production
 - Programmable ROM (**PROM**): can be programmed once
 - Erasable PROM (**EPROM**): can be bulk erased (UV, X-Ray)
 - Electrically erasable PROM (**EEPROM**): electronic erase capability
 - Flash memory: EEPROMs. with partial (block-level) erase capability
 - Wears out after about 100,000 erasing
 - Firmware
-

Applications

- Storing fonts for printers
- Storing sound data in musical instruments
- Video game consoles
- Implantable Medical devices.
- High definition Multimedia Interfaces(HDMI)
- BIOS chip in computer
- Program storage chip in modem, video card and many electronic gadgets, controllers for disks, network cards,

Memory Read Operation (1)

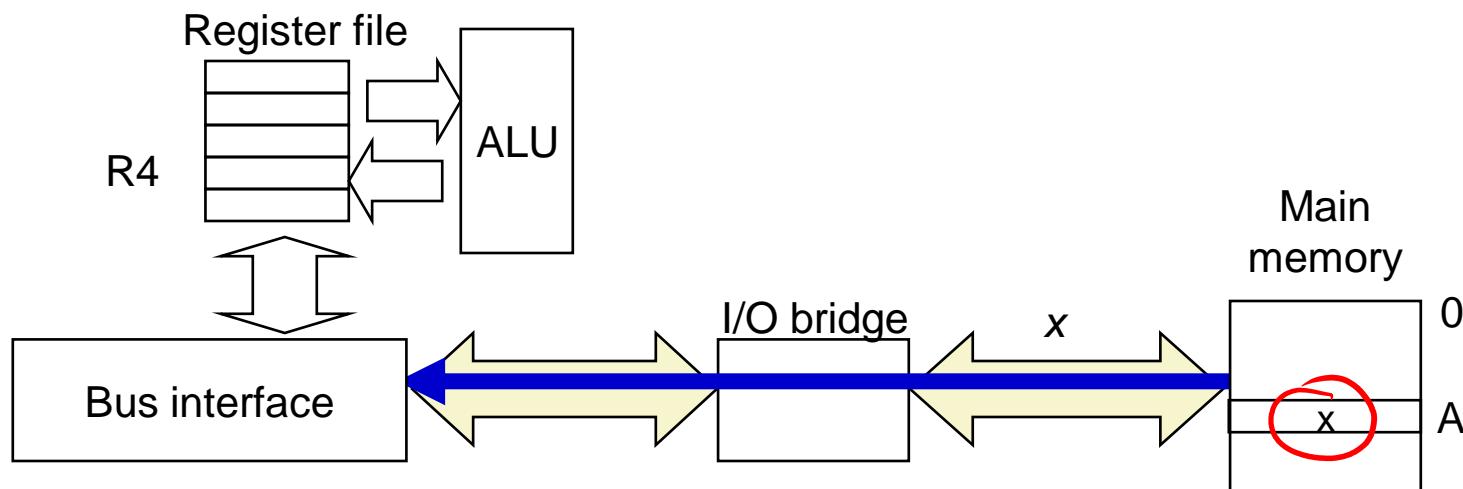
CPU places **address A** and then **read control signal** on the memory bus



Memory Read Operation (2)

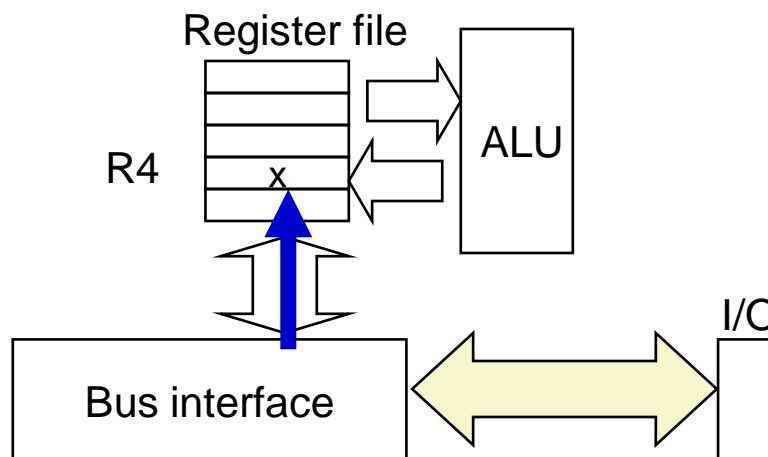
Main memory reads A from the memory bus, retrieves word x , and places it on the bus

Load operation: $\text{MOV } R4, A$
 $R4 \leftarrow [A]$



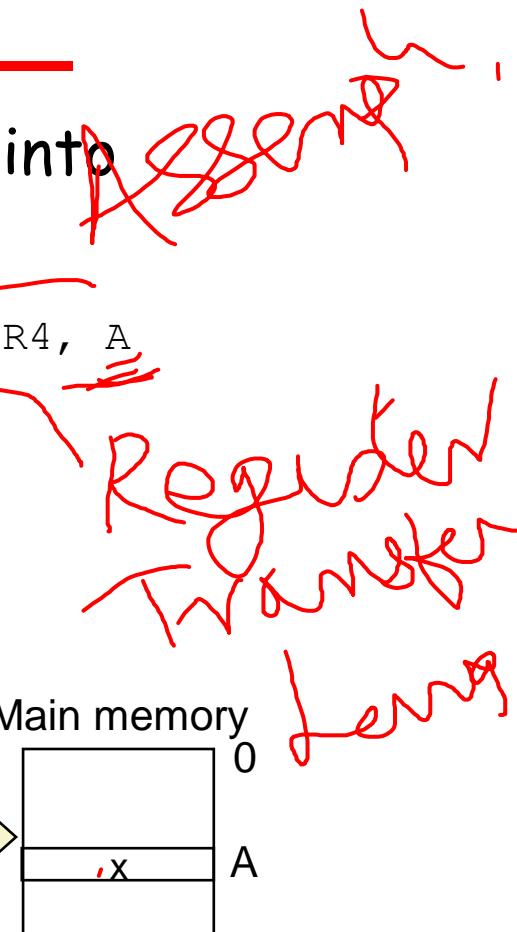
Memory Read Operation (3)

CPU read word x from the bus and copies it into register R4.



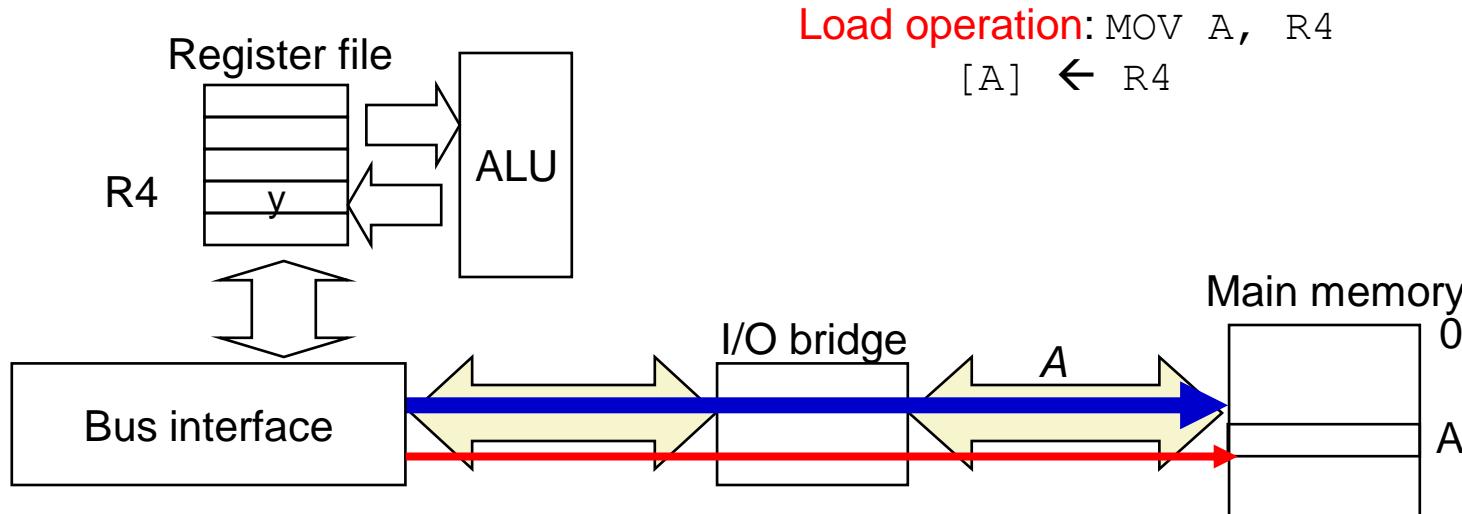
Load operation: $MOV\ R4, [A]$

$R4 \leftarrow [A]$



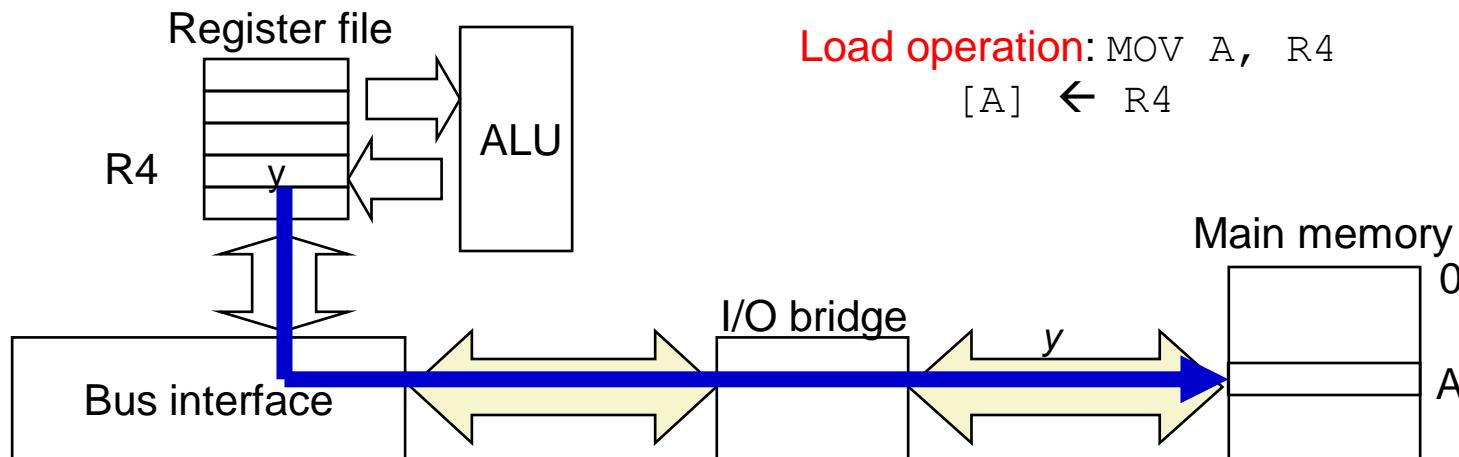
Memory Write Operation (1)

CPU places **address A** and **WRITE** control signal on bus.
 Main memory reads them and waits for the corresponding data word to arrive.



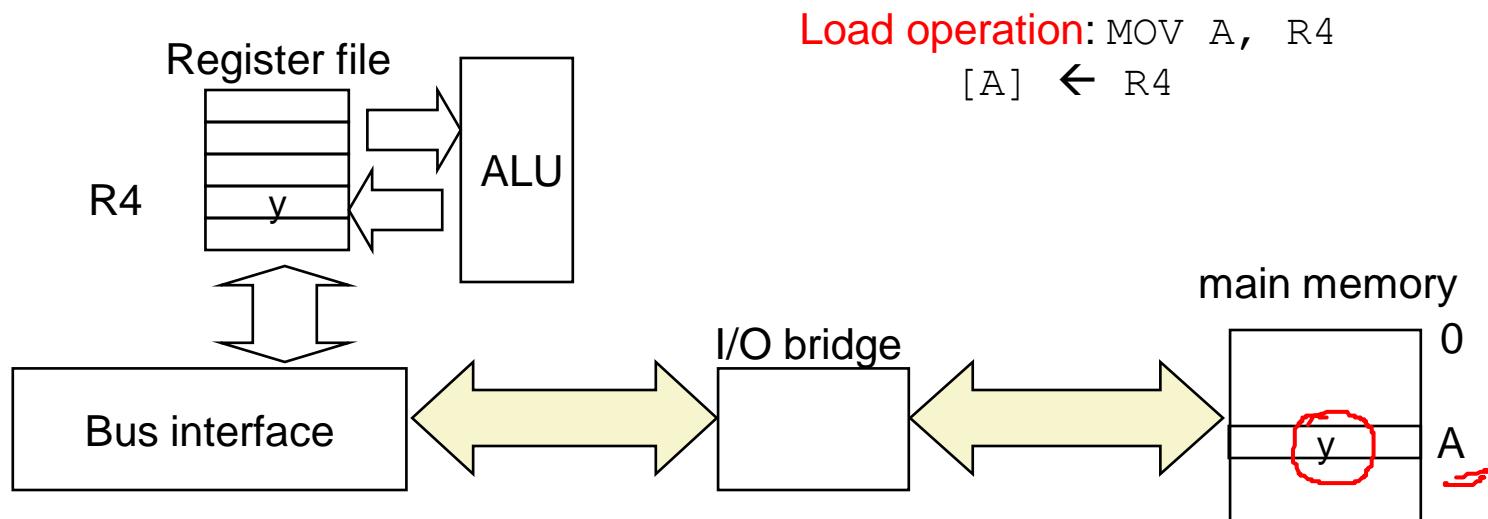
Memory Write Operation (2)

CPU places data word y on the bus



Memory Write Operation (3)

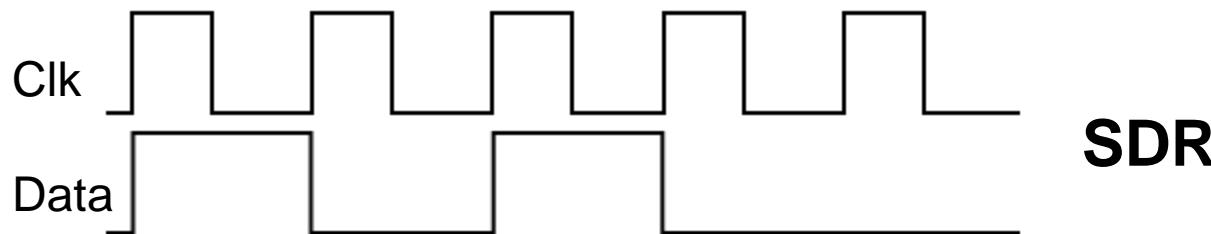
Main memory reads data word y from the bus and stores it at address A .



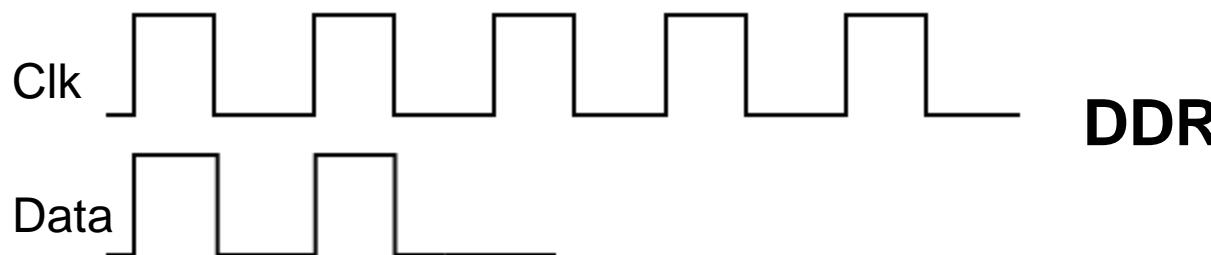
SDR Vs DDR

SDR → Single Data Rate

DDR → Double Data Rate

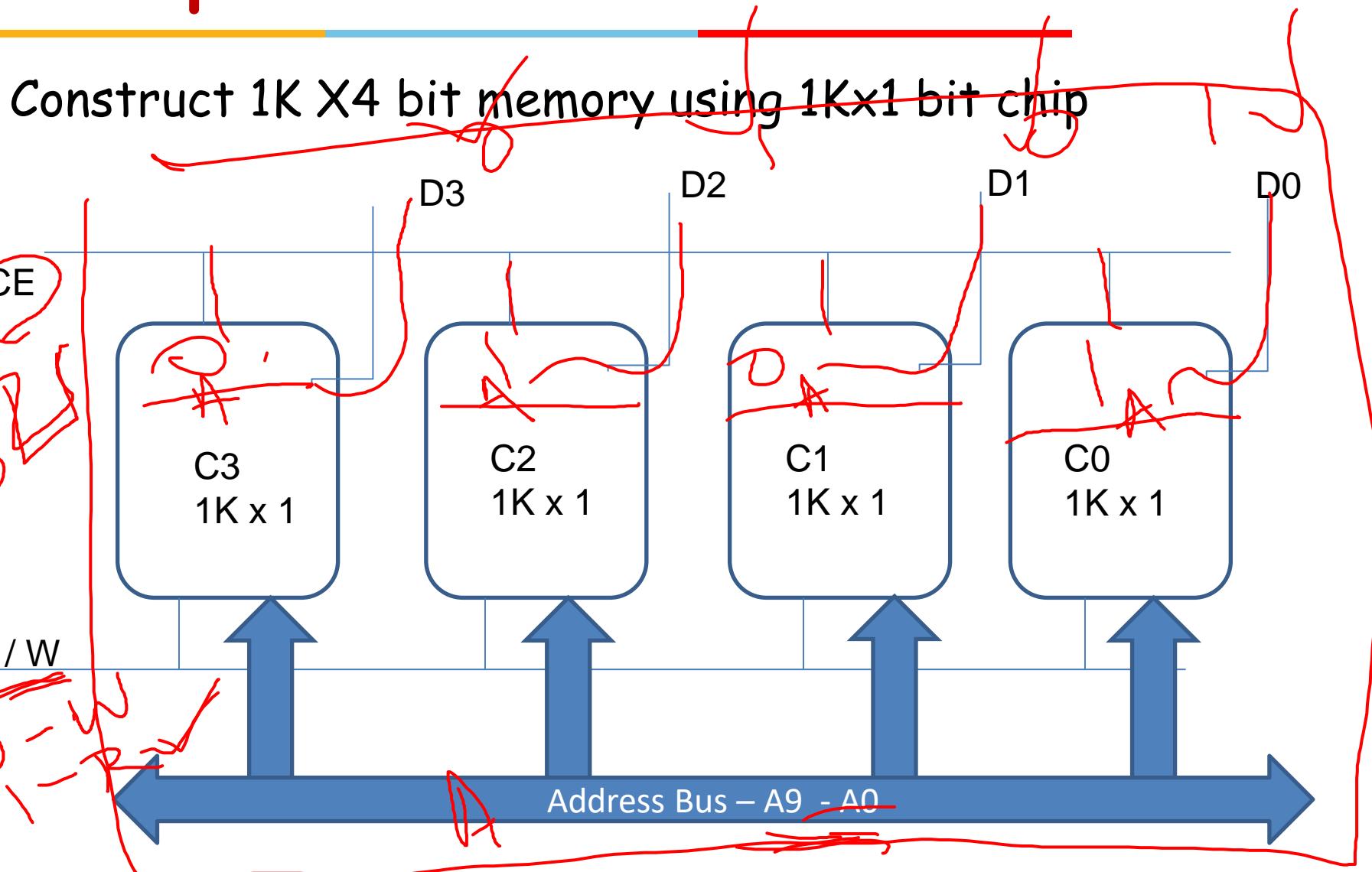


SDR



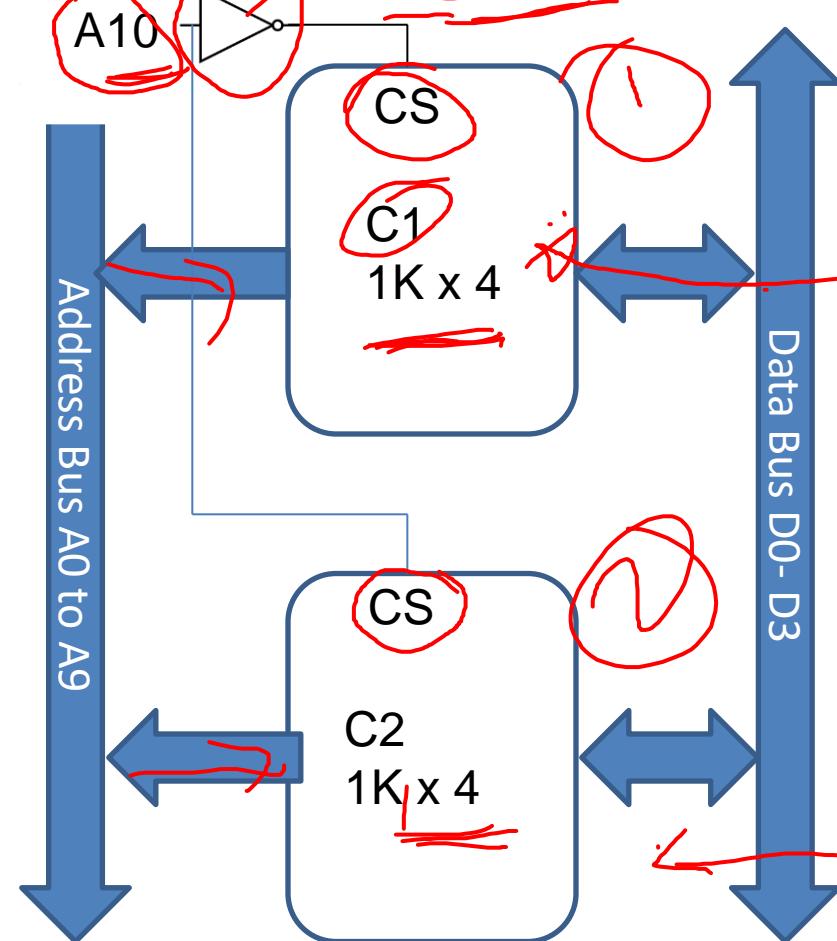
DDR

Typical Memory Connection Examples



Typical Memory Connection Examples

~~Construct 2K X4 bit memory using 1Kx4 bit chip~~

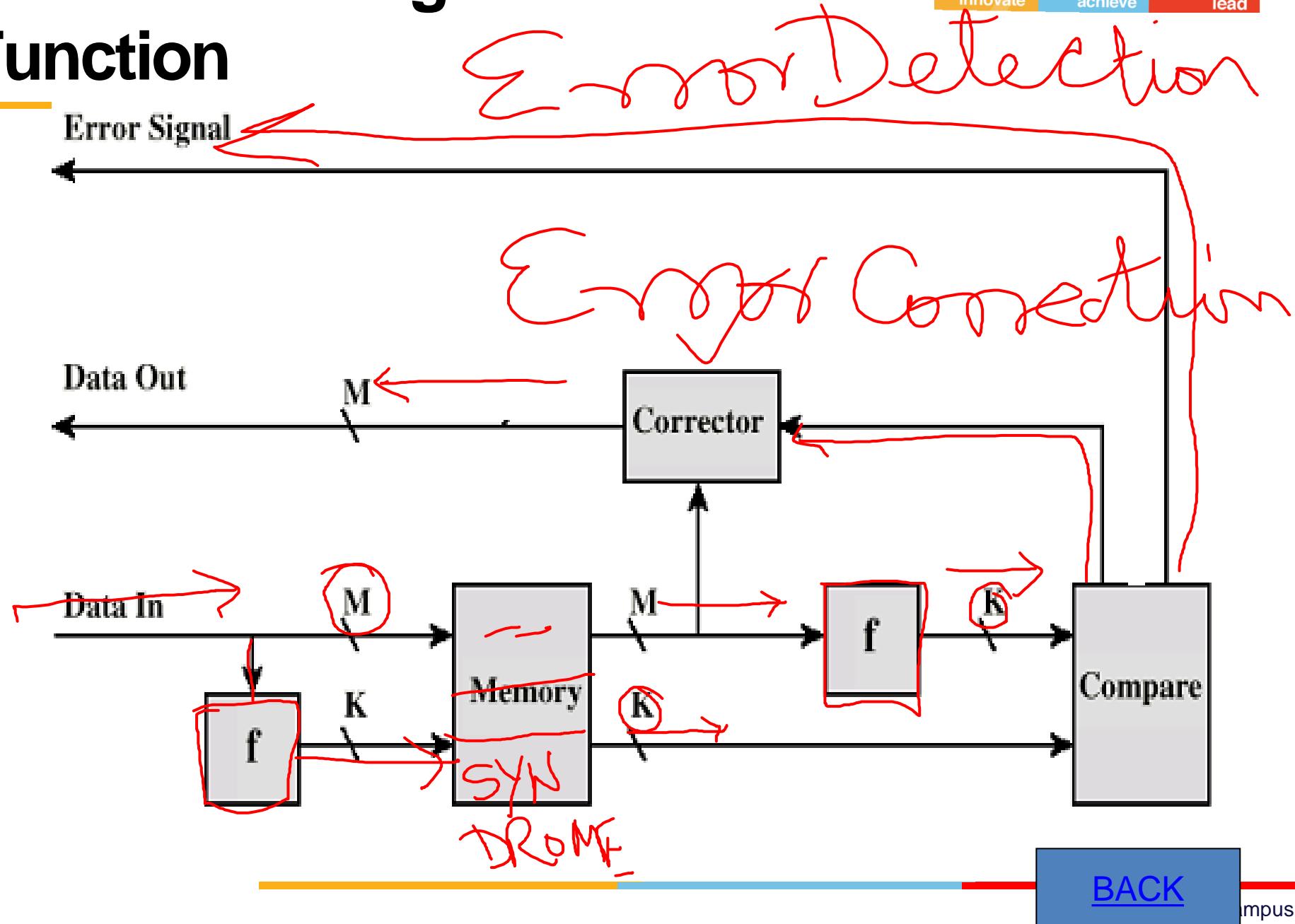


Error Correction

- Hard Failure
 - Caused by harsh environmental abuse or manufacturing defects or wear
 - Memory cell is permanently stuck at 0 or 1
 - Permanent defect
- Soft Error
 - Random, non-destructive
 - alters the contents of one or more memory cells without damaging the memory.
 - No permanent damage to memory
 - Caused by power supply problems
- Detected using Hamming error correcting code

Error Correcting Code

Function



Error Correcting Code

Function

- The comparison logic receives as input two K-bit values. A bit-by-bit comparison is done by taking the exclusive-OR of the two inputs. The result is called the *syndrome word*.
- The comparison yields one of three results
 - No errors are detected.
 - An error is detected and it is possible to correct the error
 - An error is detected but it is not possible to correct it.

Hamming Code.....

- What should be the length of the code K ?
- Result of comparison is known as syndrome word
- length of the syndrome word is K bits, Length of Data is "M" bits
- Length of K should satisfy

$$2^k - 1 \geq M + K$$

$2^3 - 1 = 7$

$M = 3$

$K = 4$

$M = 2$

$K = 3$

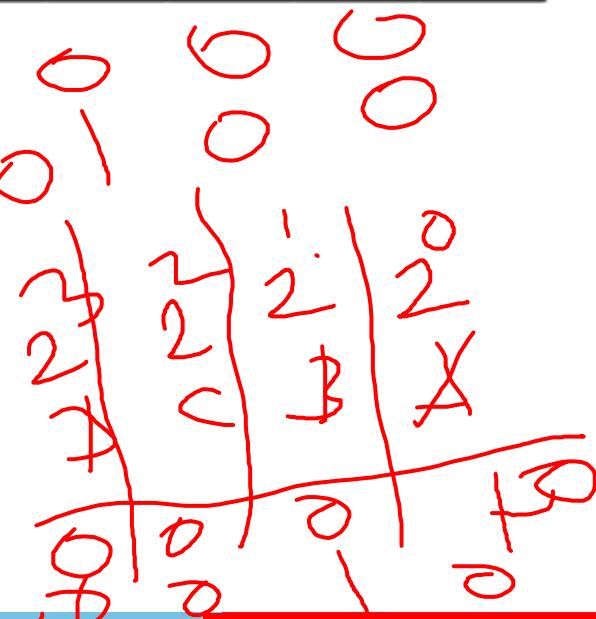
Hamming code....

- Generate 4-bit syndrome for an 8 bit data word with following characteristics
 - If the syndrome contains all 0's, no error has been detected.
 - If the syndrome contains one and only one bit set to one, then an error has occurred in one of the 4 check bits. No correction is needed
 - If the syndrome contains more than one bit set to 1, then the numerical value of the syndrome indicates the position of the data bit in error. This data bit is inverted to correction

Layout of Data and Check bits

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C4				C3		C2	C1

- $C_1 = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7$
- $C_2 = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_7$
- $C_3 = D_2 \oplus D_3 \oplus D_4 \oplus D_8$
- $C_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_8$



Problem 1

Consider the data + code k is as follows:

110101011101

Find out if there is an error. If so which bit is having error?

- $C_1 = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7$
- $C_2 = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_7$
- $C_3 = D_2 \oplus D_3 \oplus D_4 \oplus D_8$
- $C_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_8$

Problem 1 - Solution

Consider the data + check bit is as follows:

110101011101

Find out if there is an error. If so which bit is having

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C4				C3		C2	C1

Problems1 - Solution (Contd..)

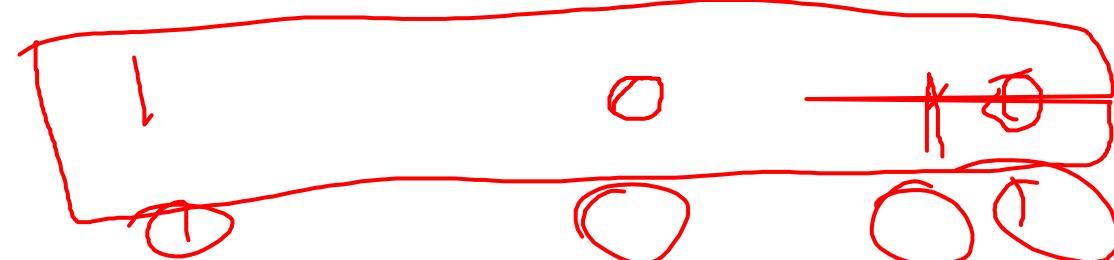
Consider the data + check bit is as follows:

110101011101

Find out if there is an error. If so which bit is having

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit	()	0)	C4)	0)	(C3)	0	C1

1 1 0 1 0 1 0 1 1 1 0 1

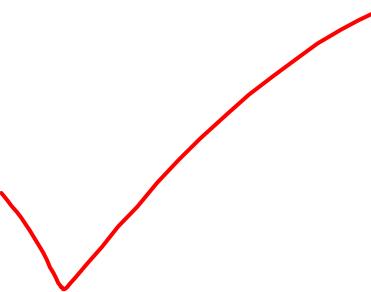


Problem 2

- $C_1 = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7$
- $C_2 = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_7$
- $C_3 = D_2 \oplus D_3 \oplus D_4 \oplus D_8$
- $C_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_8$

data : 10101100 ($M = 8$)

Compute Check Bits



Types of External Memory

- Magnetic Disk
 - RAID
 - Removable
- Optical
 - CD-ROM
 - CD-Recordable (CD-R)
 - CD-R/W
 - DVD
- Magnetic Tape

Magnetic Disk Drive

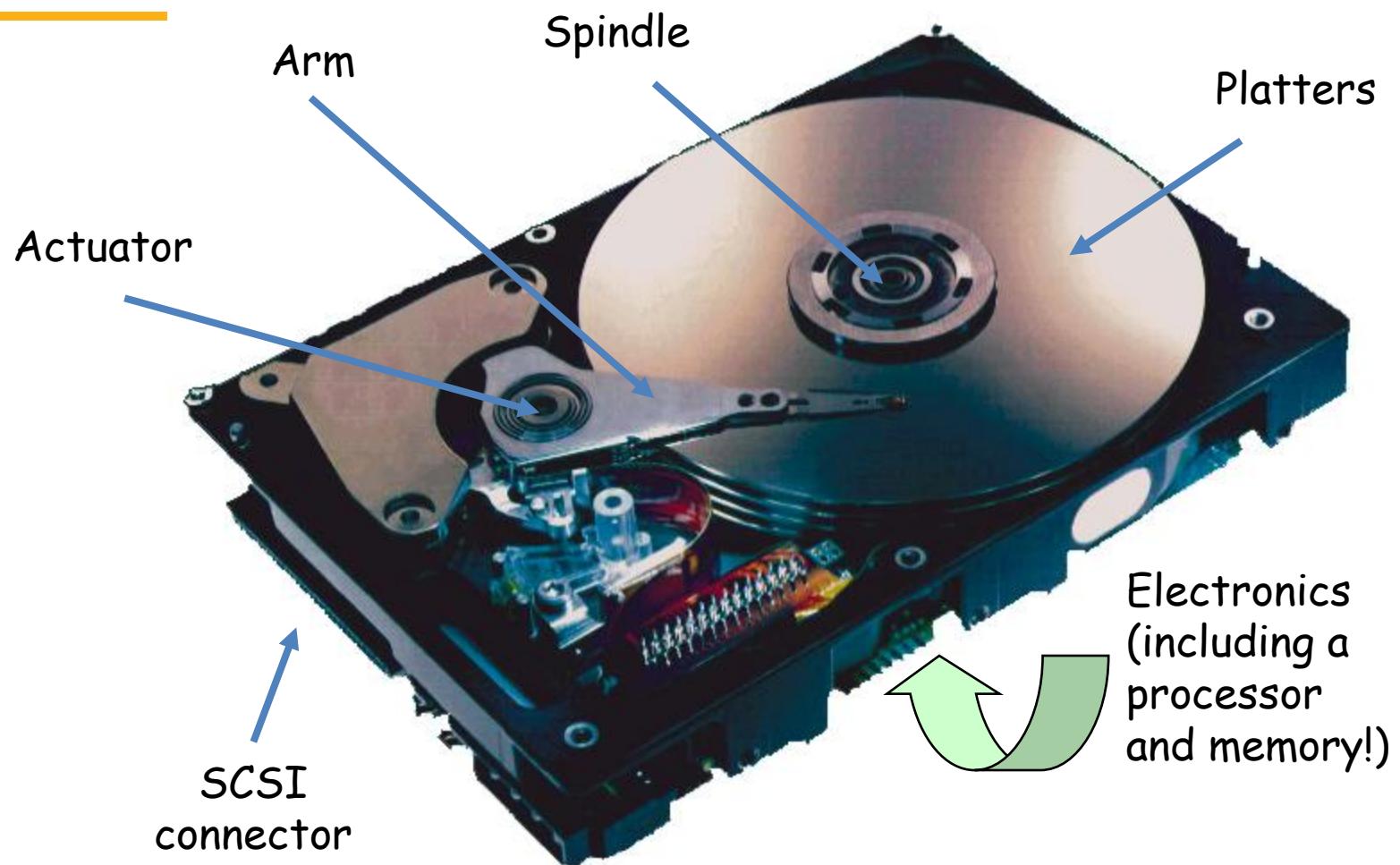
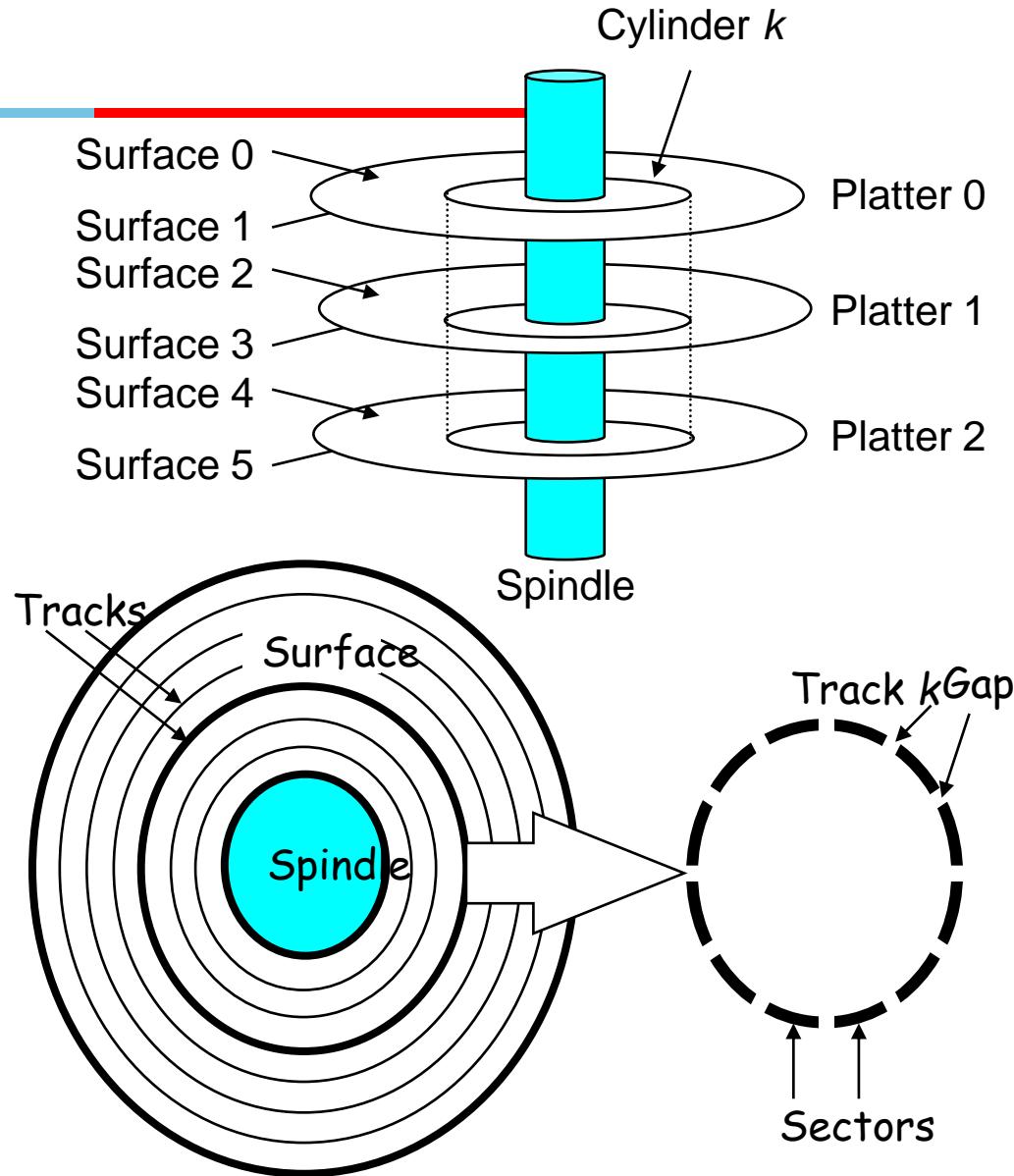


Image courtesy of Seagate Technology

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**.



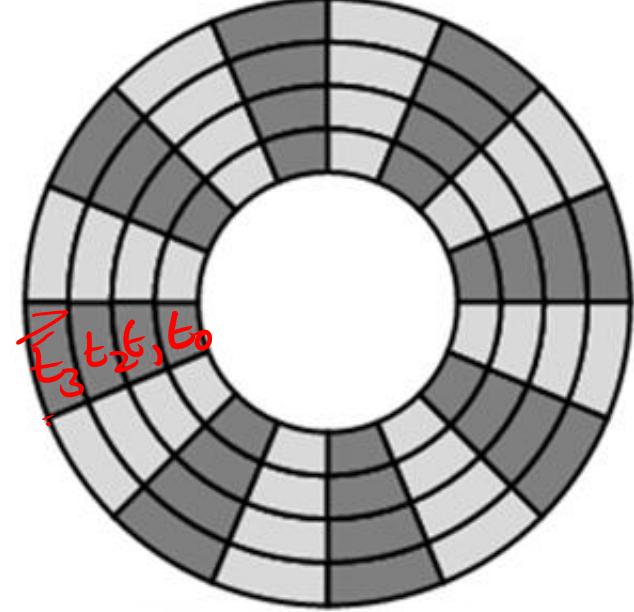
Disk Capacity

- **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.

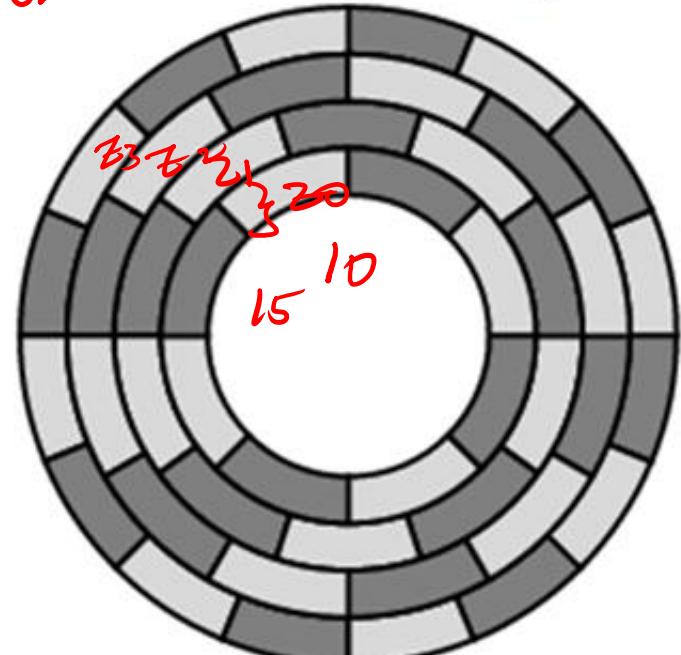
Recording zones

512 byte

- Modern disks partition tracks into disjoint subsets called **recording zones**
 - Each track in a zone has the same number of sectors, determined by the circumference of innermost track.
 - Each zone has a different number of sectors/track, outer zones have more sectors/track than inner zones.
 - So we use **average** number of sectors/track when computing capacity.



10 track Without Recording Zones



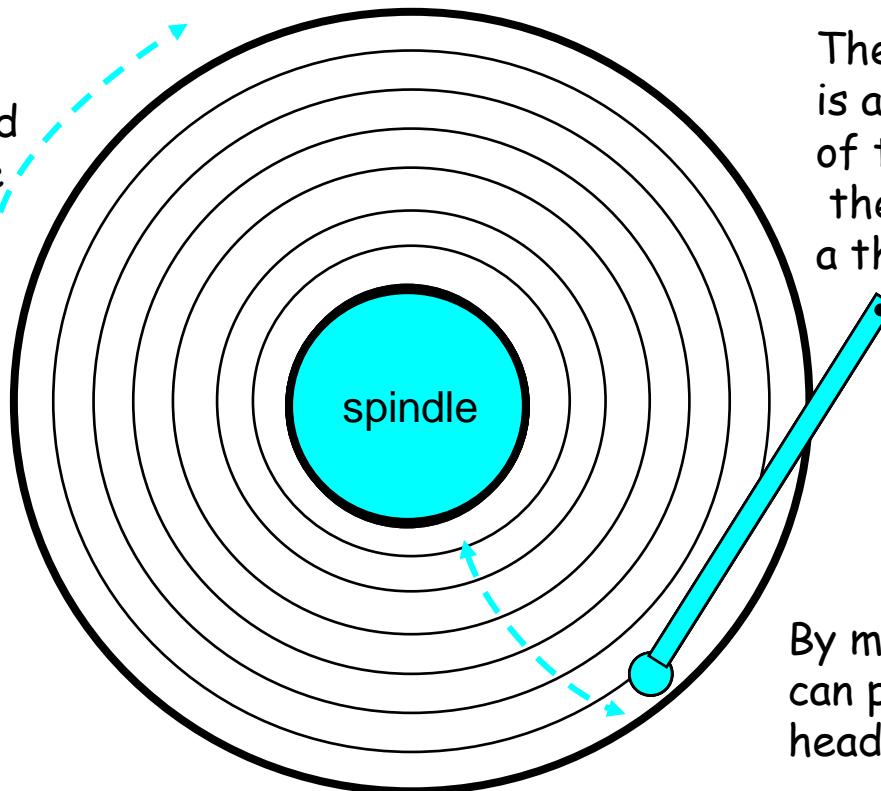
With Recording Zones

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 (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)

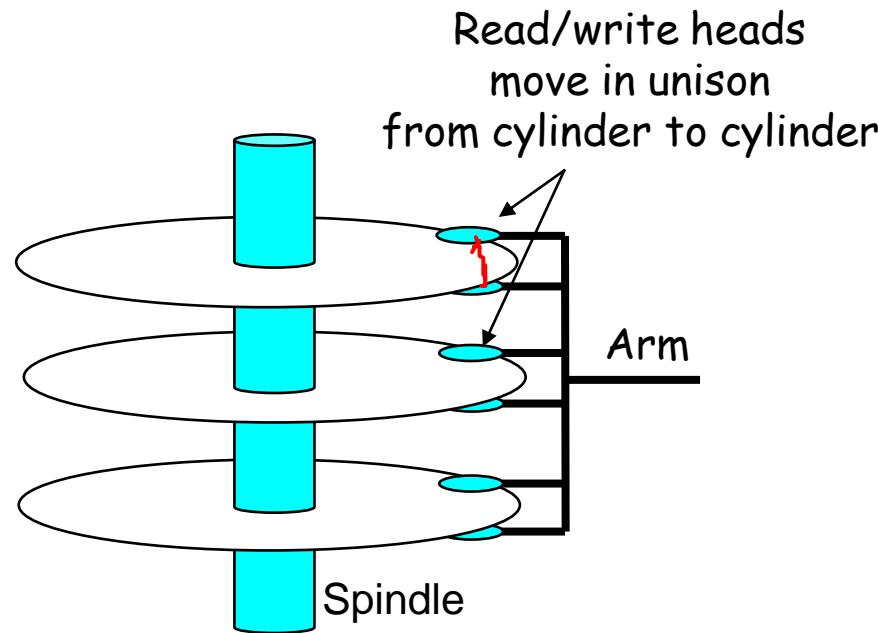
The disk surface spins at a fixed rotational rate



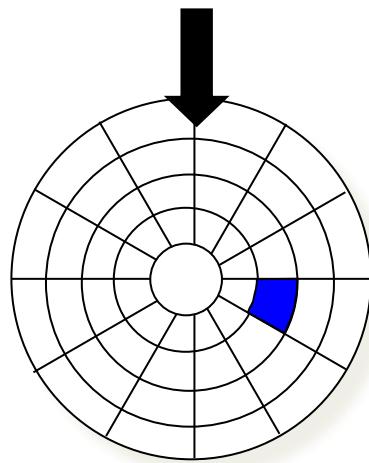
The read/write head is attached to the end of the arm and flies over the disk surface on a thin cushion of air.

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

Disk Operation (Multi-Platter View)

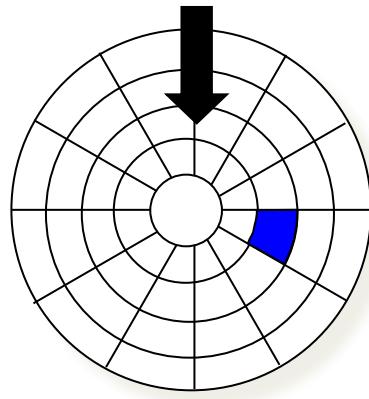


Disk Access



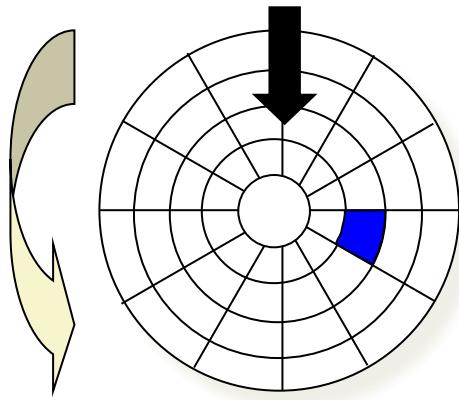
Need to access a sector
colored in blue

Disk Access



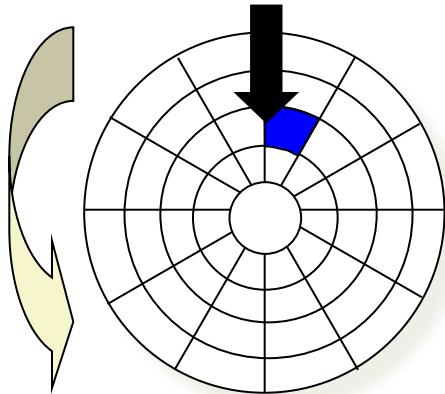
Head in position above a track

Disk Access



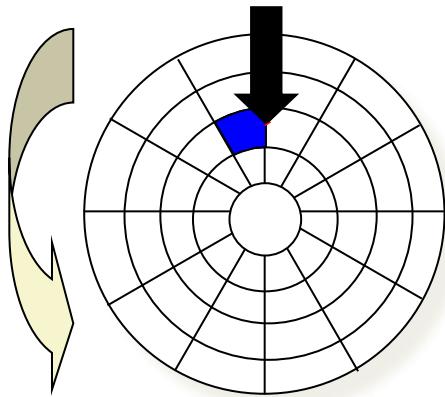
Rotate the platter in counter-clockwise direction

Disk Access - Read



About to read blue sector

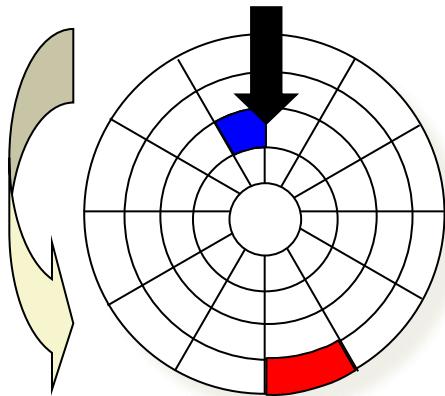
Disk Access - Read



After BLUE
read

After reading blue sector

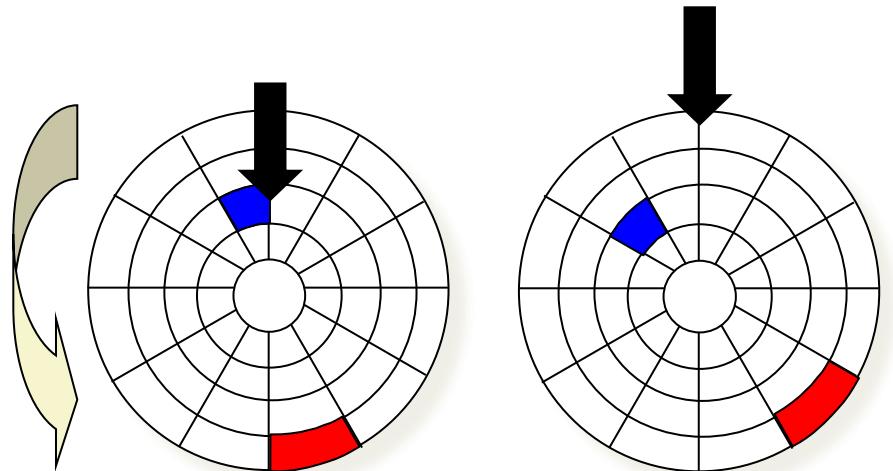
Disk Access - Read



After BLUE
read

Red request scheduled next

Disk Access - Seek

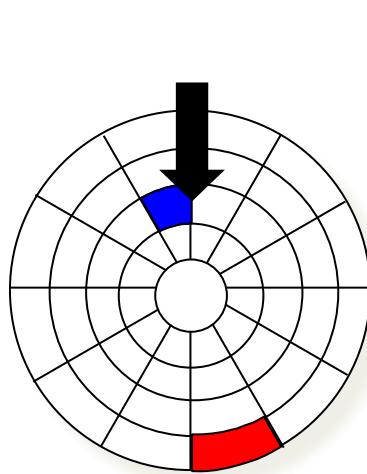


After BLUE
read

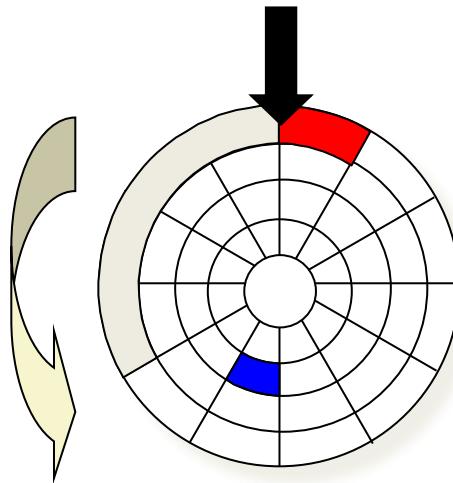
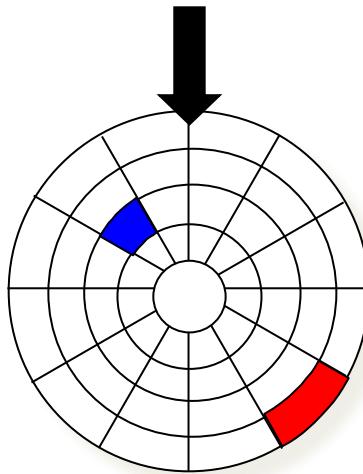
Seek for RED

Seek to red's track

Disk Access - Rotational Latency



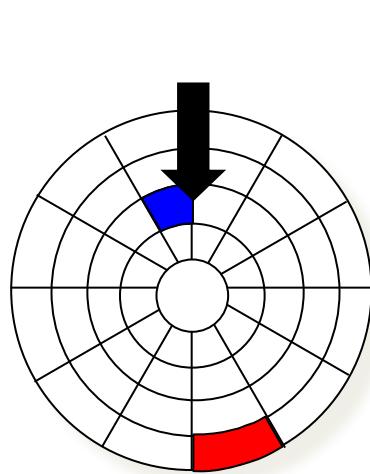
After BLUE Seek for RED
read



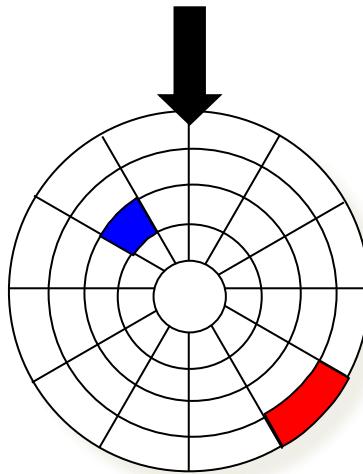
Rotational latency

Wait for red sector to rotate around

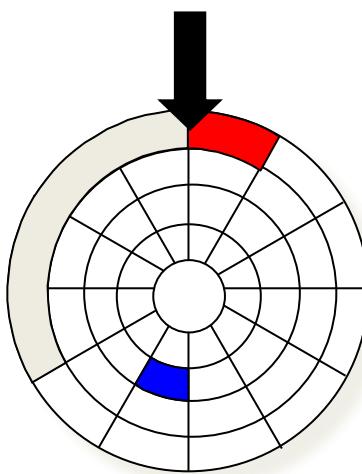
Disk Access - Read



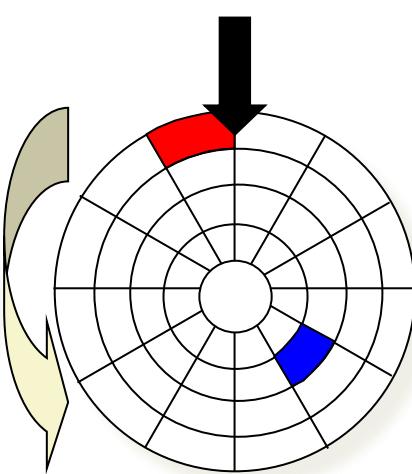
After BLUE
read



Seek for RED

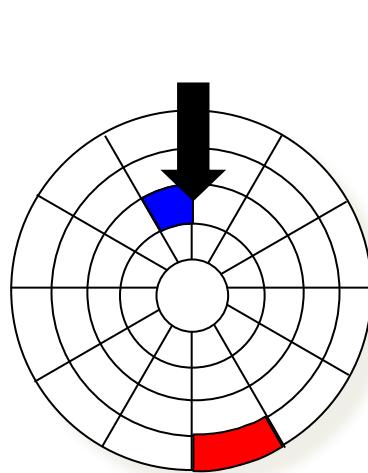


Rotational latency After RED read

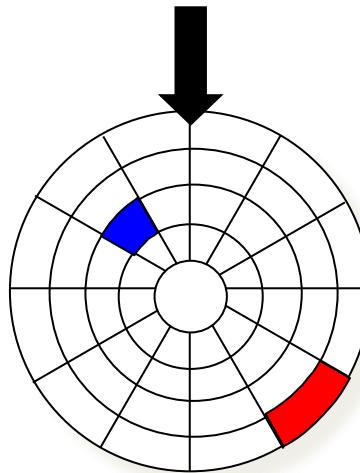
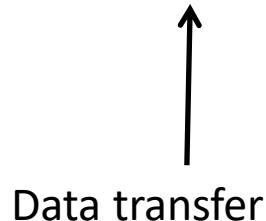


Complete read of red

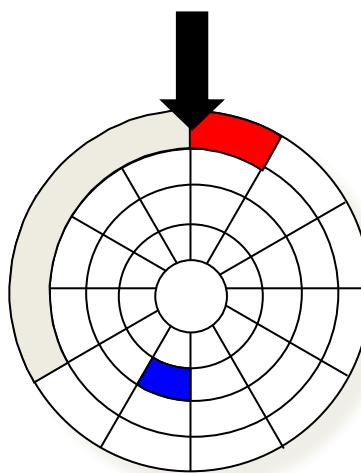
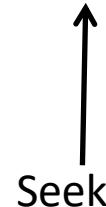
Disk Access - Access Time Components



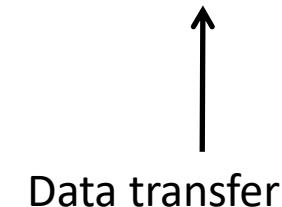
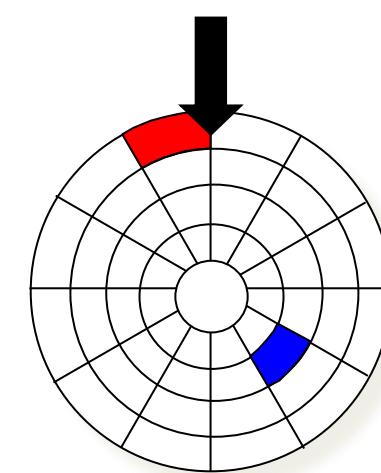
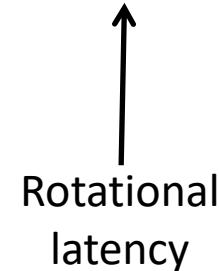
After BLUE
read



Seek for RED



Rotational latency After RED read





BITS Pilani
Pilani Campus

Computer Organization and Software Systems

CONTACT SESSION 3

Prepared By: Dr. Lucy J. Gudino
Instructor: Prof. C R Sarma

Last Class

Contact Hour	List of Topic Title	Text/Ref Book/external resource
3-4	Memory Organization - Internal Memory - External Memory (HDD)	T1, R2

Today's Class

Contact Hour	List of Topic Title	Text/Ref Book/external resource
5-6	<p>Memory Organization</p> <ul style="list-style-type: none"> - External Memory (RAID, SSD) <p>Cache Memory Organization</p> <ul style="list-style-type: none"> - Locality - Locality of Reference to Program Data - Locality of instruction fetches 	T1, R2

RAID

- RAID - Redundant Array of Independent Disks
- Variety of ways in which the data can be organized
- Need for RAID:
 - Parallel I/O
 - Reliability
 - Redundancy through multiple inexpensive disks
- Consists of 7 levels (0-6) each acts as a design Architectures

Common Characteristics

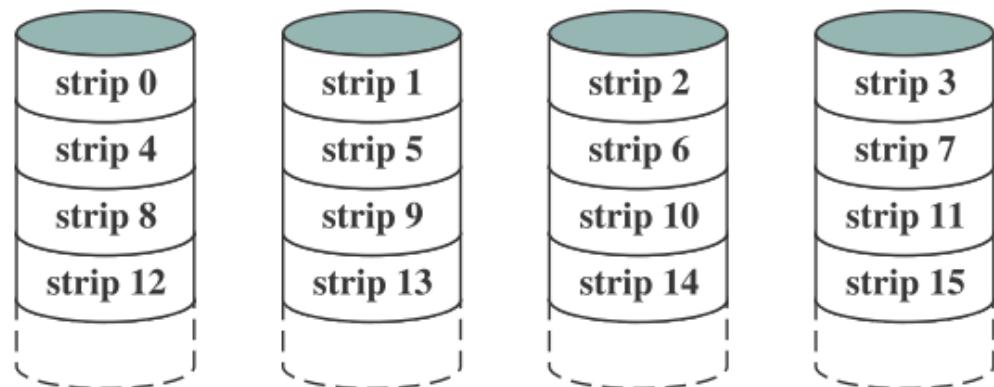
1. RAID is a set of physical disk drives viewed by the operating system as a single logical drive.
2. Data are distributed across the physical drives of an array in a scheme known as striping.
3. Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.

Categories

- RAID category
 - Striping (Level 0)
 - Mirroring (Level 1)
 - Parallel access (Level 2,3)
 - Independent access (Level 4,5,6)

RAID Level 0

- Is not a true member of RAID family - No Redundancy
- Data are distributed across all of the disks in the array
- IO requests can be issued in Parallel
- Reduce IO queuing time
- The data are striped

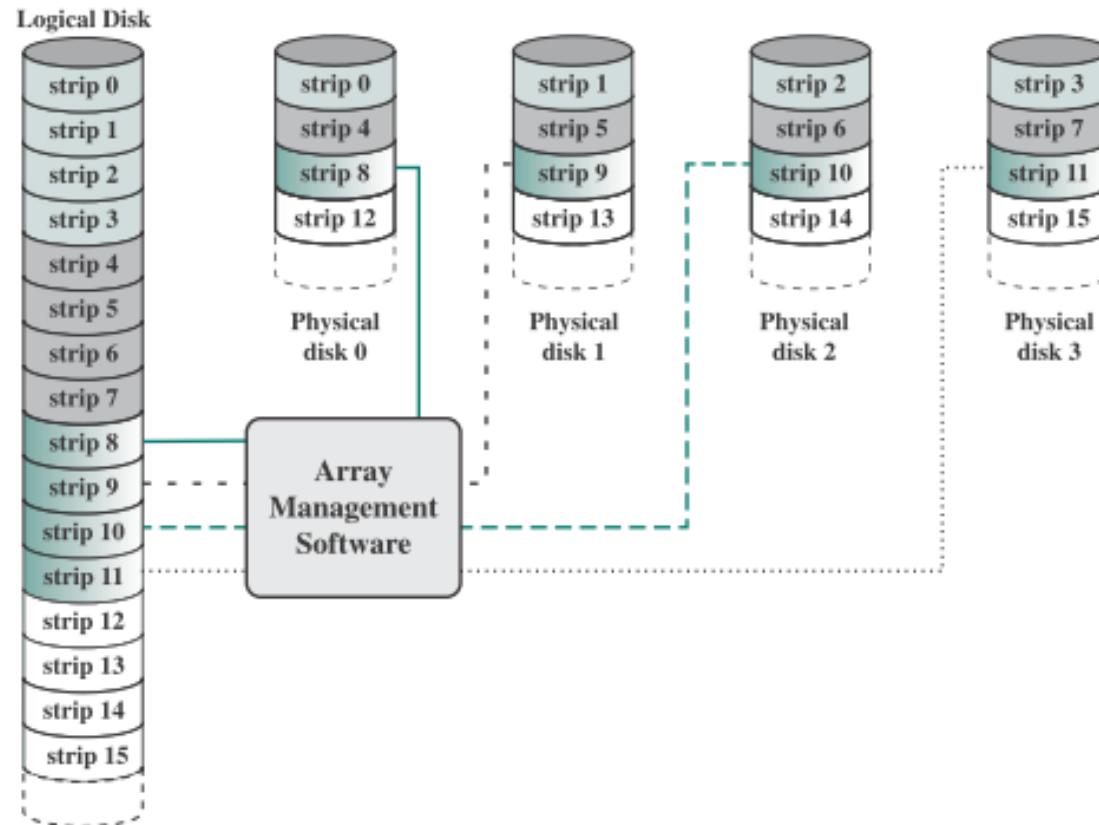


(a) RAID 0 (Nonredundant)

RAID level 0 configuration

RAID 0 for

- high data transfer capacity
- high I/O request rate

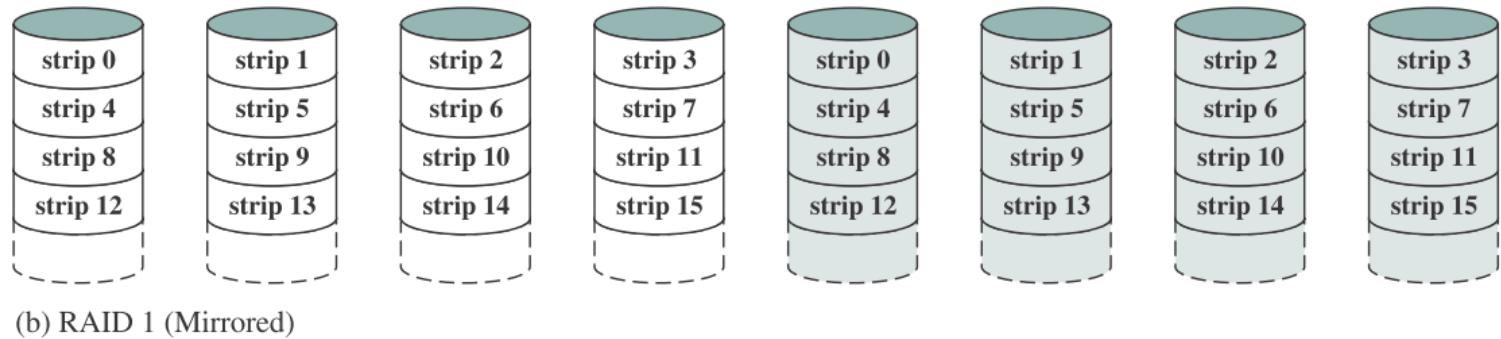


RAID Level 0

Level	Advantages	Disadvantages	Applications
0	<ul style="list-style-type: none"> I/O performance is greatly improved by spreading the I/O load across many channels and drives No parity calculation overhead is involved Very simple design Easy to implement 	The failure of just one drive will result in all data in an array being lost	<ul style="list-style-type: none"> Video production and editing Image Editing Pre- press applications Any application requiring high bandwidth

RAID Level 1

- Mirroring: redundancy is achieved by duplicating all the data
- Each logical strip is mapped to two separate physical disks
- Every disk in the array has a mirror disk that contains the same data



Advantages of RAID 1 organization



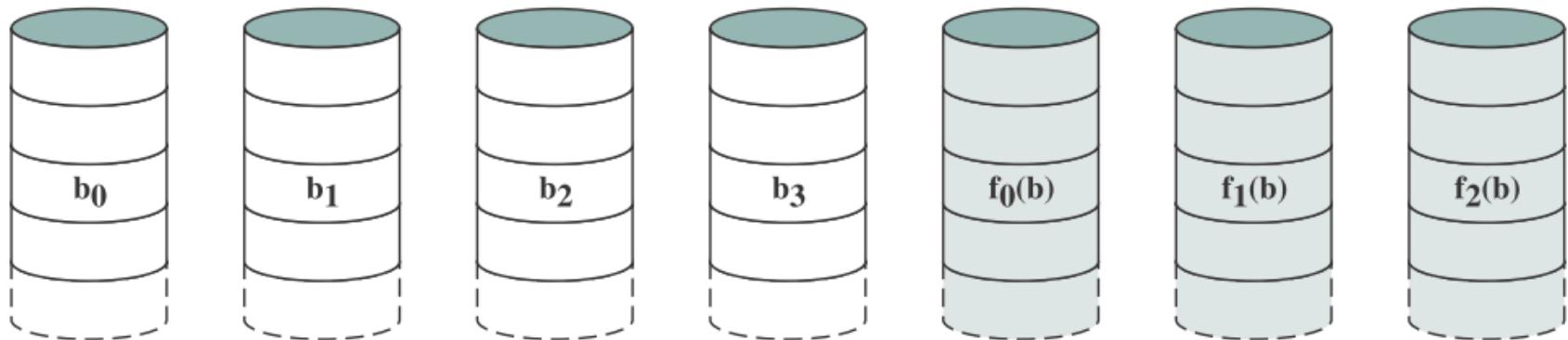
1. A read request can be serviced by either of the two disks that contains the requested data, whichever one involves the minimum seek time plus rotational latency.
2. A write request requires that both corresponding strips be updated, but this can be done in parallel. Thus, the write performance is dictated by the slower of the two writes (i.e., the one that involves the larger seek time plus rotational latency).
3. Recovery from a failure is simple. When a drive fails, the data may still be accessed from the second drive.

RAID Level 1

Level	Advantages	Disadvantages	Applications
1	<ul style="list-style-type: none"> • 100% redundancy of data means no rebuild is necessary in case of a disk failure, just a copy to the replacement disk • Under certain circumstances, RAID 1 can sustain multiple simultaneous drive failures • Simplest RAID storage subsystem design 	Highest disk overhead of all RAID types (100%)—inefficient	<ul style="list-style-type: none"> • Accounting • Payroll • Financial • Any application requiring very high availability

RAID Level 2

- Use parallel access technique
- In a parallel access array, all member disks participate in the execution of every I/O request
- Spindles of the individual drives are synchronized



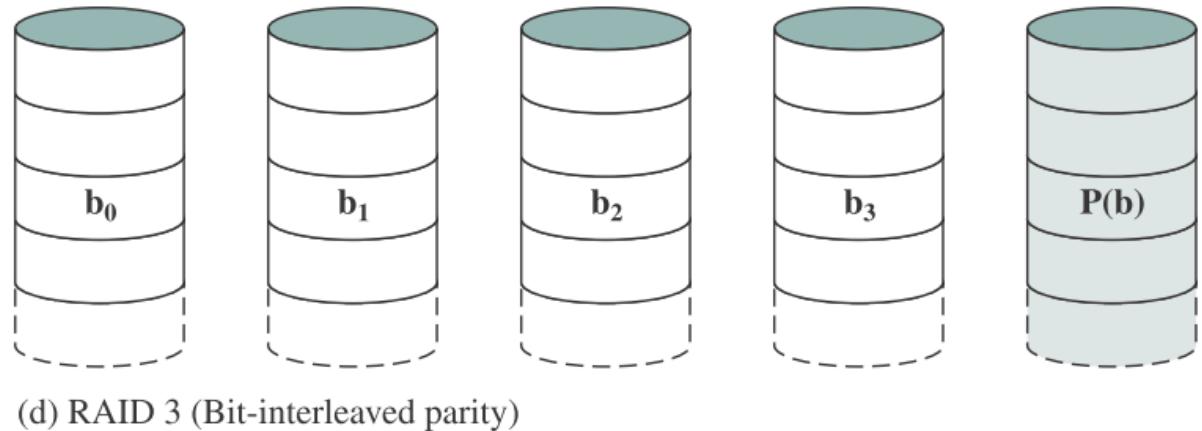
(c) RAID 2 (Redundancy through Hamming code)

RAID Level 2

Level	Advantages	Disadvantages	Applications
2	<ul style="list-style-type: none"> Extremely high data transfer rates possible The higher the data transfer rate required, the better the ratio of data disks to ECC disks Relatively simple controller design compared to RAID levels 3, 4, & 5 	<ul style="list-style-type: none"> Very high ratio of ECC disks to data disks with smaller word sizes— inefficient Entry level cost very high— requires very high transfer rate requirement to justify 	<ul style="list-style-type: none"> No commercial implementations exist/not commercially viable

RAID Level 3

- RAID 3 is organized in a similar fashion to RAID 2
- RAID 3 requires only a single redundant disk
- RAID 3 employs parallel access, with data distributed in small strips
- Parity bit is computed for the set of individual bits in the same position on all of the data disks



RAID Level 3

- Redundancy: Event of a drive failure - parity drive is accessed and data is reconstructed from the remaining devices
 - Missing data can be restored on the new drive
- Data reconstruction is simple

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

- Suppose drive $X1$ fails

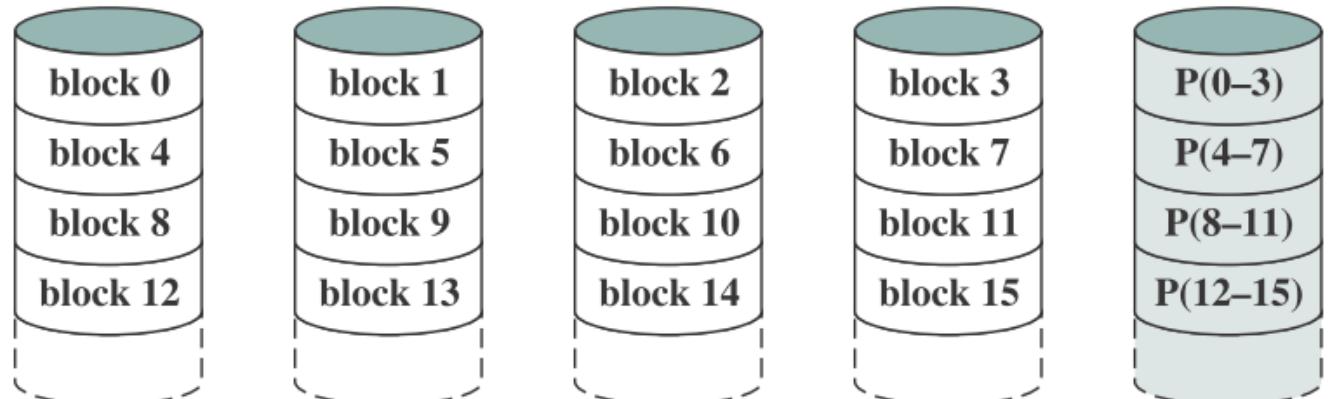
$$X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$$

RAID Level 3

Level	Advantages	Disadvantages	Applications
3	<ul style="list-style-type: none"> Very high read data transfer rate Very high write data transfer rate Disk failure has an insignificant impact on throughput Low ratio of ECC (parity) disks to data disks means high efficiency 	<ul style="list-style-type: none"> Transaction rate equal to that of a single disk drive at best (if spindles are synchronized) Controller design is fairly Complex 	<ul style="list-style-type: none"> Video production and live streaming Image editing Video editing Prepress applications Any application requiring high throughput

RAID Level 4

- RAID levels 4 through 6 make use of an independent access technique
- Each member disk operates independently, so that separate I/O requests run in parallel



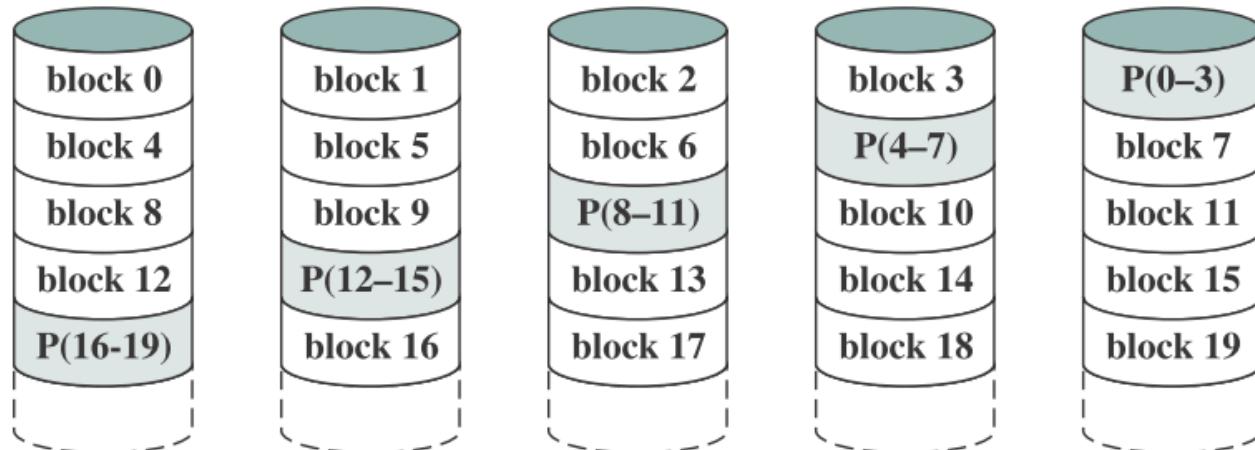
(e) RAID 4 (Block-level parity)

RAID Level 4

Level	Advantages	Disadvantages	Applications
4	<ul style="list-style-type: none"> • Very high Read data transaction rate • Low ratio of ECC (parity) disks to data disks means high efficiency 	<ul style="list-style-type: none"> • Quite complex controller design • Worst write transaction rate and Write aggregate transfer rate • Difficult and inefficient data rebuild in the event of disk failure 	<ul style="list-style-type: none"> • No commercial implementations exist/not commercially viable

RAID Level 5

RAID 5 is organized in a similar fashion to RAID 4.
The difference is that RAID 5 distributes the parity strips across all disks



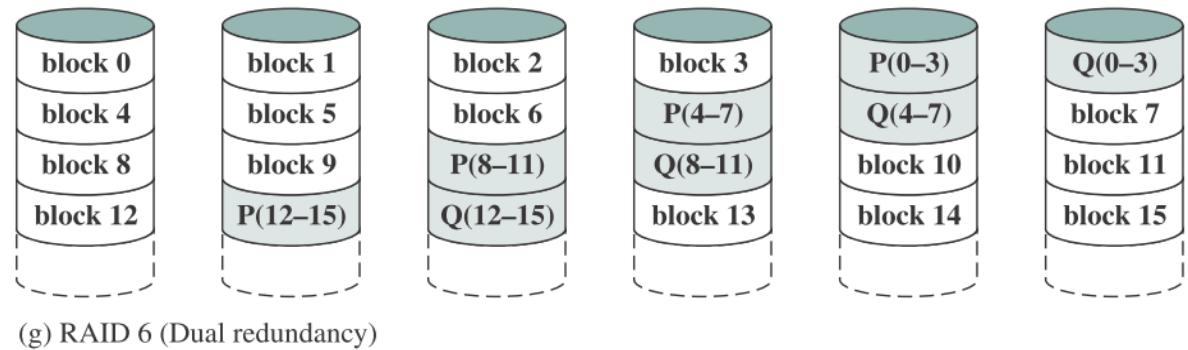
(f) RAID 5 (Block-level distributed parity)

RAID Level 5

Level	Advantages	Disadvantages	Applications
5	<ul style="list-style-type: none"> • Highest Read data transaction rate • Low ratio of ECC (parity) disks to data disks means high efficiency • Good aggregate transfer rate 	<ul style="list-style-type: none"> • Most complex controller design • Difficult to rebuild in the event of a disk failure (as compared to RAID level 1) 	<ul style="list-style-type: none"> • File and application servers • Database servers • Web, e-mail, and news servers • Intranet servers • Most versatile RAID level

RAID Level 6

- Two different parity calculations are carried out and stored in separate blocks on different disks
- RAID 6 array whose user data require N disks consists of N + 2 disks
- One of the two is the exclusive- OR calculation used in RAID 4 and 5
- Other is an independent data check algorithm
- Possible to regenerate data even if two disks containing user data fail



RAID Level 6

Level	Advantages	Disadvantages	Applications
6	<ul style="list-style-type: none">Provides for an extremely high data fault tolerance and can sustain multiple simultaneous drive failures	<ul style="list-style-type: none">More complex controller designController overhead to compute parity addresses is extremely high	<ul style="list-style-type: none">Perfect solution for mission critical applications

Solid state drives

SSDs have the following advantages over HDDs:

- High- performance input/output operations per second (IOPS): Significantly increases performance I/O subsystems.
- Durability: Less susceptible to physical shock and vibration.
- Longer lifespan: SSDs are not susceptible to mechanical wear.
- Lower power consumption: SSDs use considerably less power than comparable- size HDDs.
- Quieter and cooler running capabilities: Less space required, lower energy costs, and a greener enterprise.
- Lower access times and latency rates: Over 10 times faster than the spinning disks in an HDD.

SSD Organization

SSD contains the following components:

- Controller: Provides SSD device level interfacing and firmware execution.
- Addressing: Logic that performs the selection function across the flash memory components.
- Data buffer/cache: High speed RAM memory components used for speed matching and to increased data throughput.
- Error correction: Logic for error detection and correction.
- Flash memory components: Individual NAND flash chips.

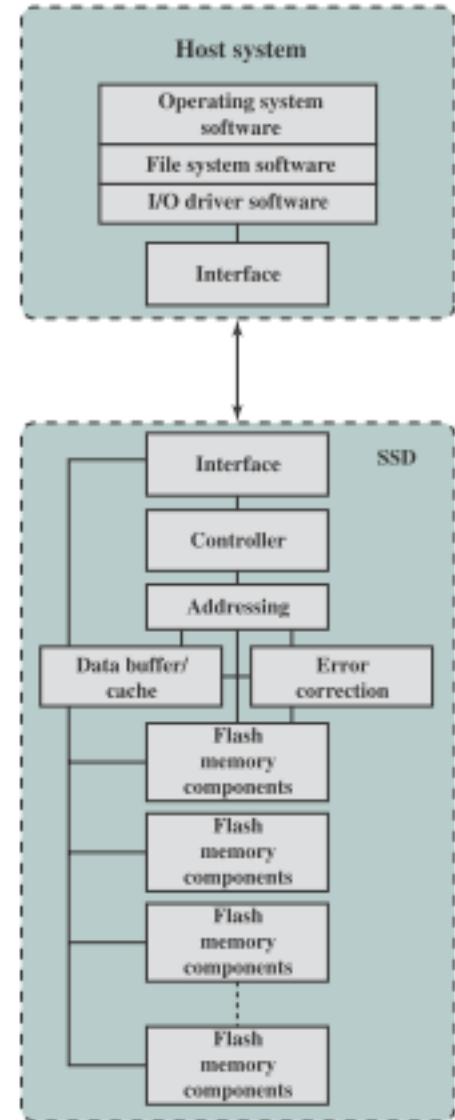


Figure 6.8 Solid State Drive Architecture

Issues with SSD

Two practical issues:

SSD performance has a tendency to slow down as the device is used

Memory becomes unusable after a certain number of writes

First Issue

Files are stored on disk as a set of pages, typically 4 KB in length

Pages are not necessarily stored as a contiguous set of pages on the disk

Flash memory is accessed in blocks, block size of 512 KB, (128 pages per block)

Write a page onto a flash memory:

1. The entire block must be read from the flash memory and placed in a RAM buffer. Then the appropriate page in the RAM buffer is updated.
2. Before the block can be written back to flash memory, the entire block of flash memory must be erased— it is not possible to erase just one page of the flash memory.
3. The entire block from the buffer is now written back to the flash memory.

Second Issue

Flash memory becomes unusable after a certain number of writes.

Typical limit is 100,000 writes

Comparison of Solid State Drives and Disk Drives

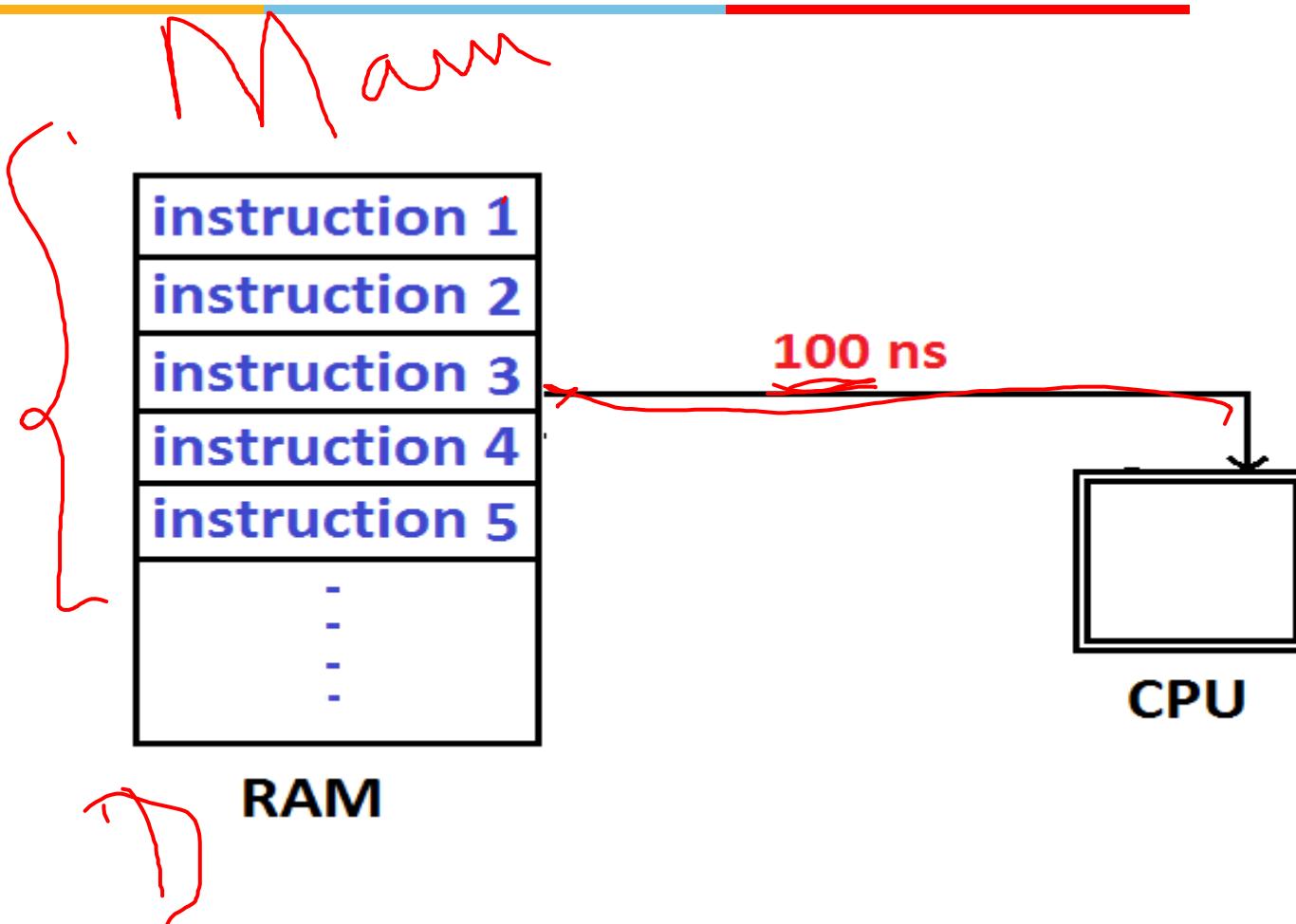


	NAND Flash Drives	Seagate Laptop Internal HDD
File copy/write speed	200–550 Mbps	50–120 Mbps
Power draw/battery life	Less power draw, averages 2–3 watts, resulting in 30+ minute battery boost	More power draw, averages 6–7 watts and therefore uses more battery
Storage capacity	Typically not larger than 512 GB for notebook size drives; 1 TB max for Desktops	Typically around 500 GB and 2 TB max for notebook size drives; 4 TB max for desktops
Cost	Approx. \$0.50 per GB for a 1-TB drive	Approx. \$0.15 per GB for a 4-TB drive

Memory Hierarchy

- Registers
 - In CPU
- Internal or Main memory
 - May include one or more levels of cache
 - "RAM"
- External memory
 - Backing store

Performance enhancement - Motivation



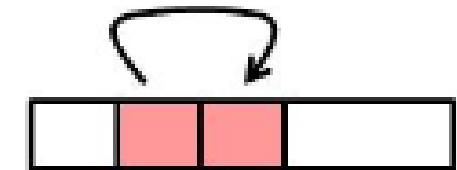
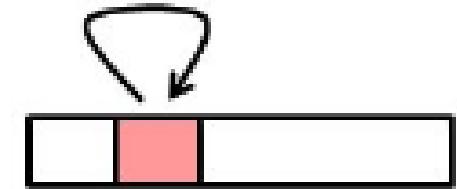
Performance enhancement - Motivation



Locality of Reference

During the course of the execution of a program, memory references tend to cluster

- **Temporal locality:** Locality in time
 - If an item is referenced, it will tend to be referenced again soon
- **Spatial locality:** Locality in space
 - If an item is referenced, items whose addresses are close by will tend to be referenced soon.



Example

```
product = 1;  
for ( i = 0; i < n-1; i++)  
    product = product * a[i] ;
```

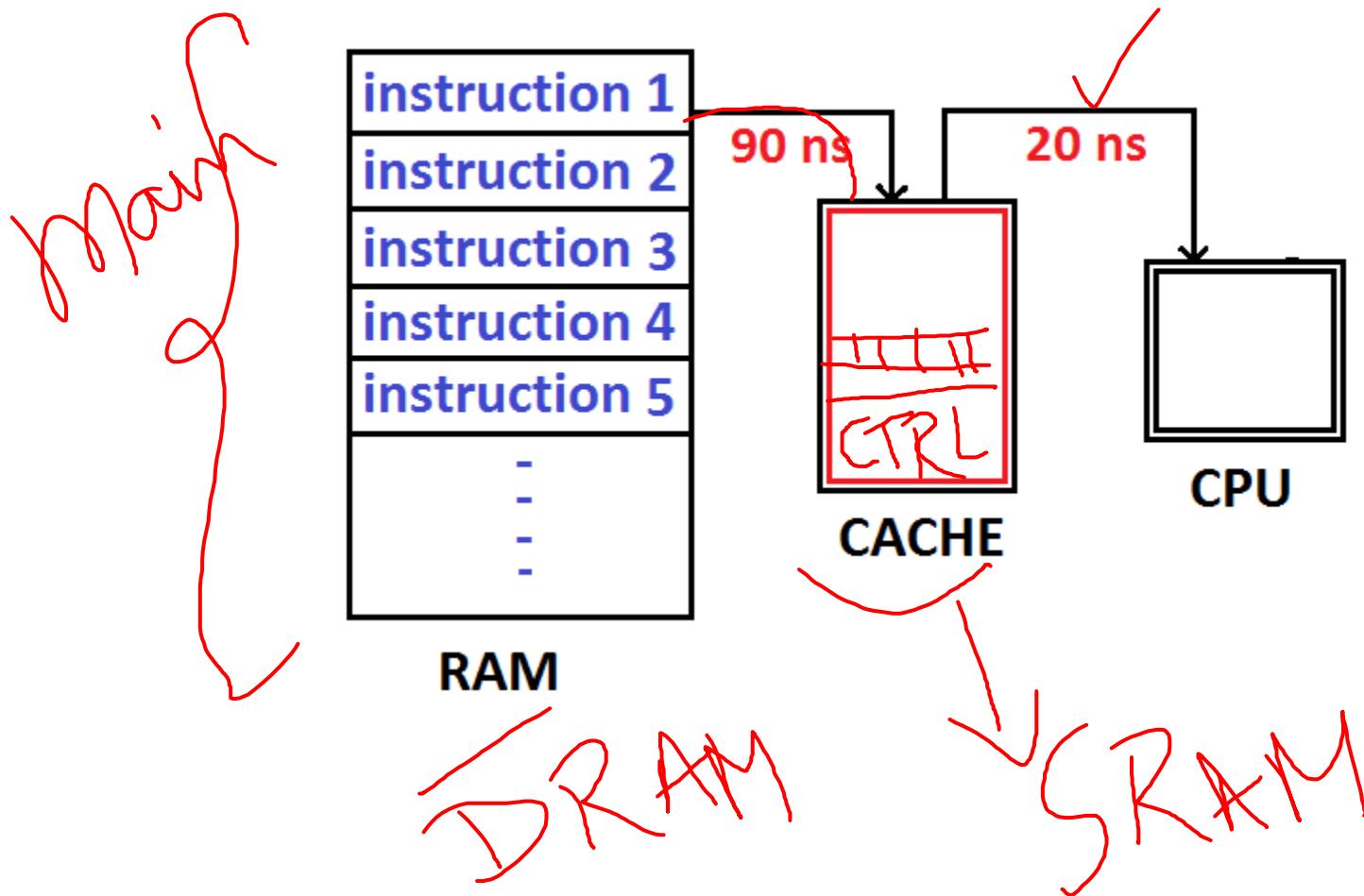
Data :

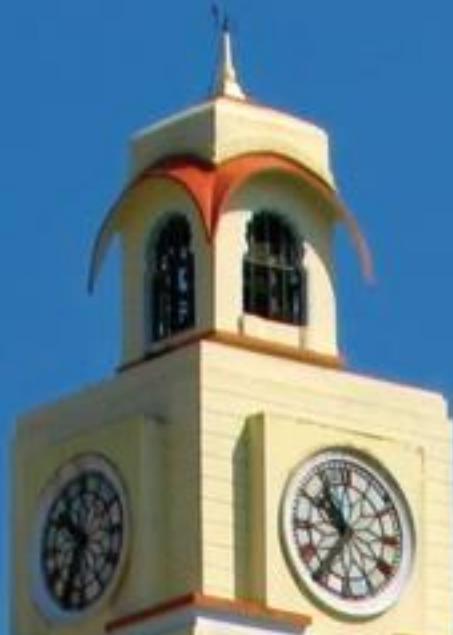
- Access array elements in succession - spatial locality
- Reference to "product" in each iteration - Temporal locality

Instructions :

- Reference instructions in sequence : Spatial locality
- Looping through : Temporal locality

Performance enhancement - Motivation



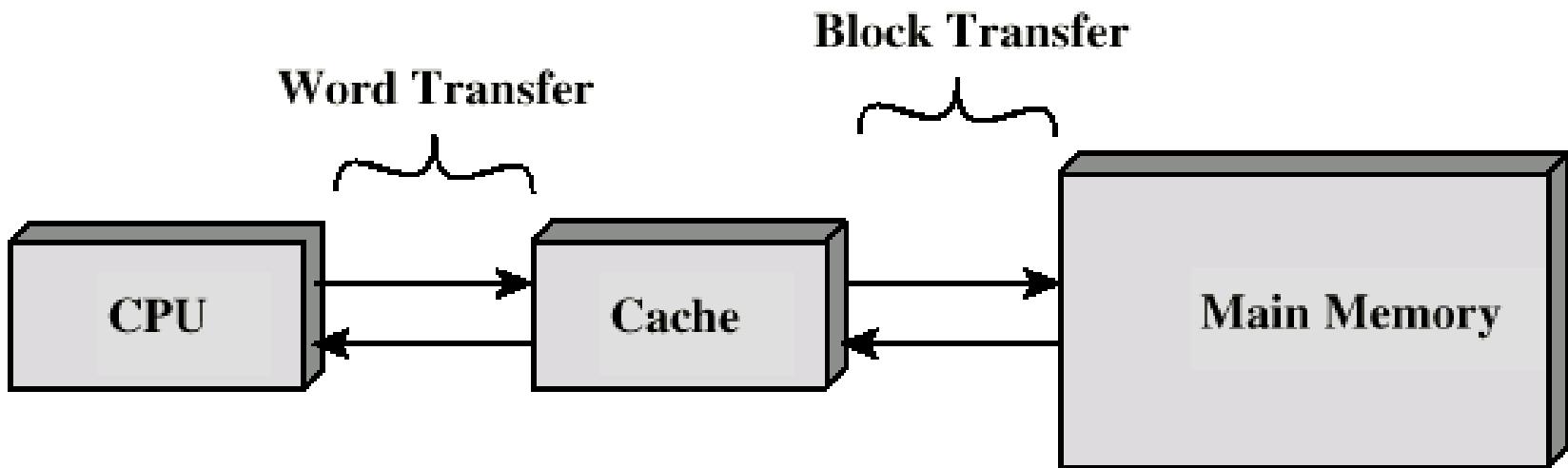


BITS Pilani
Pilani Campus

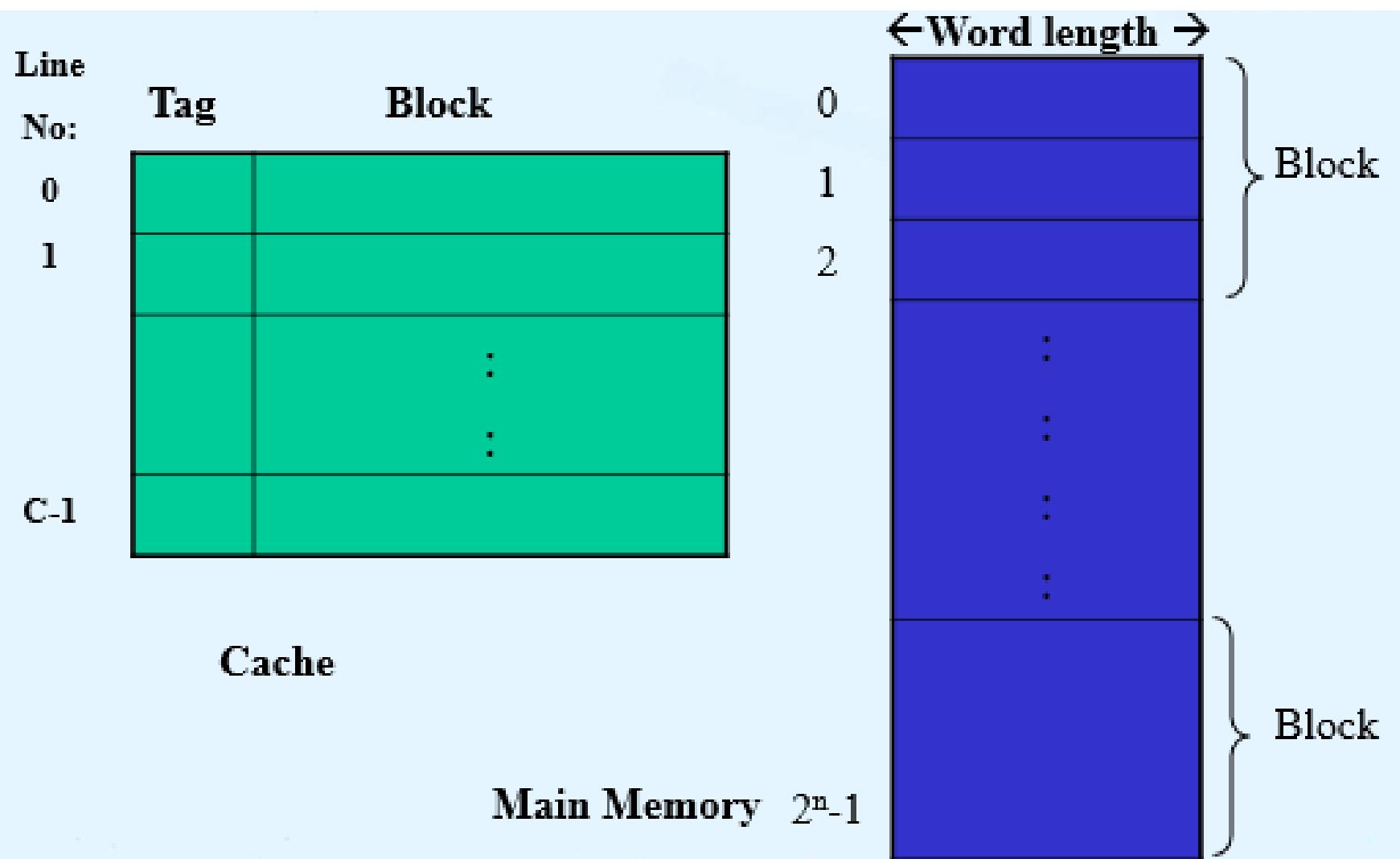
Cache

Cache

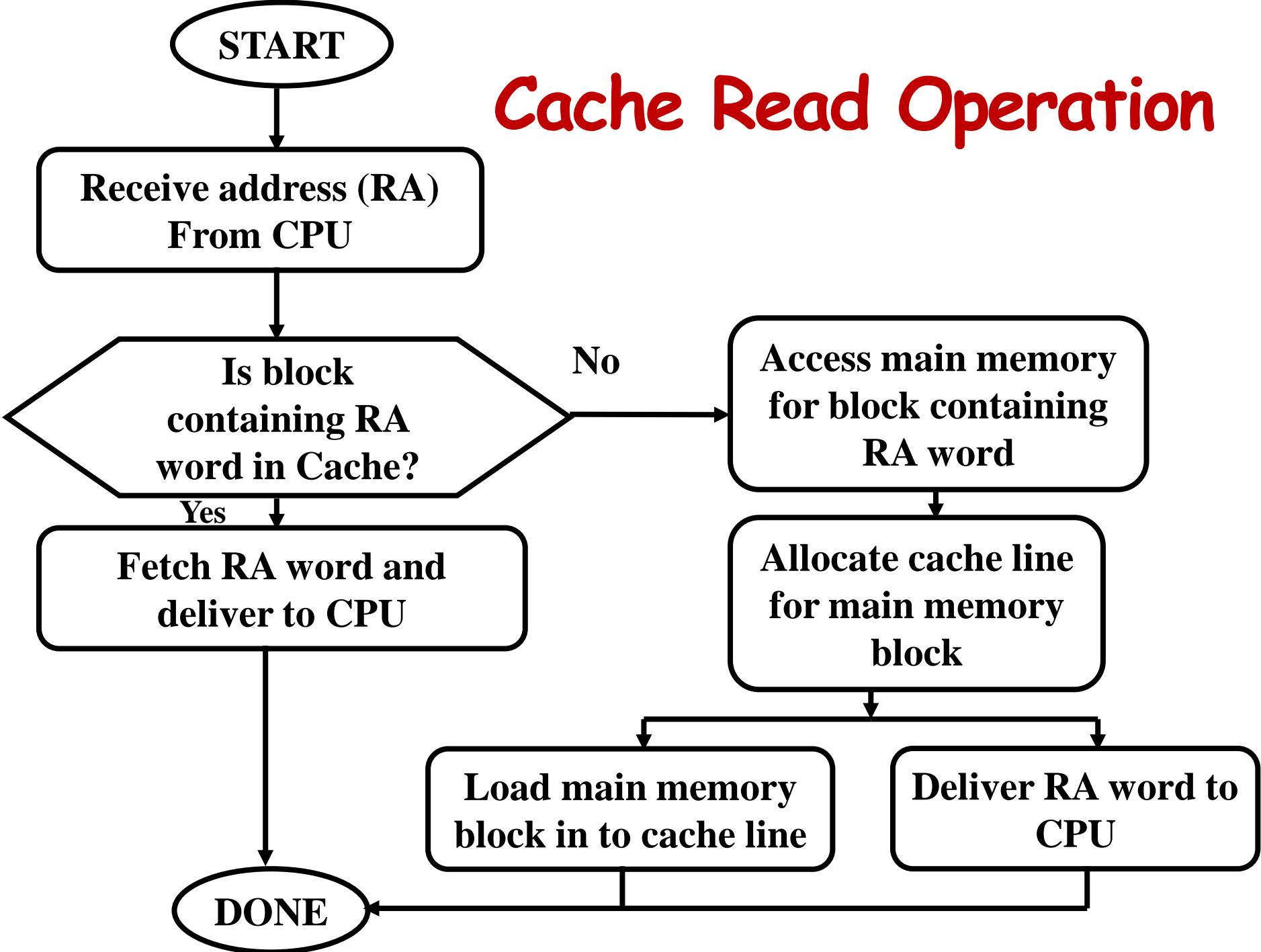
- Small, fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or separate module



Cache and Main Memory Structure



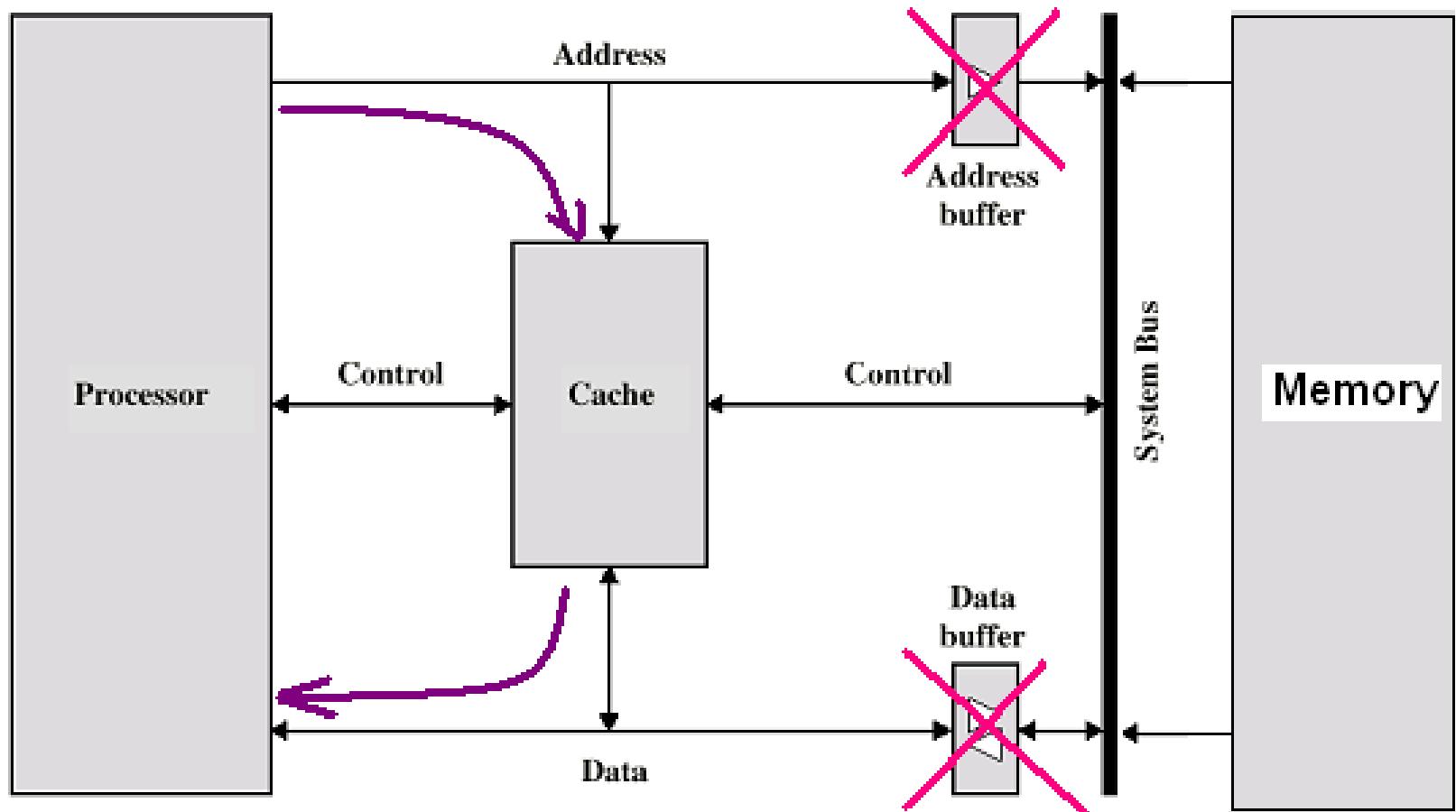
Cache Read Operation



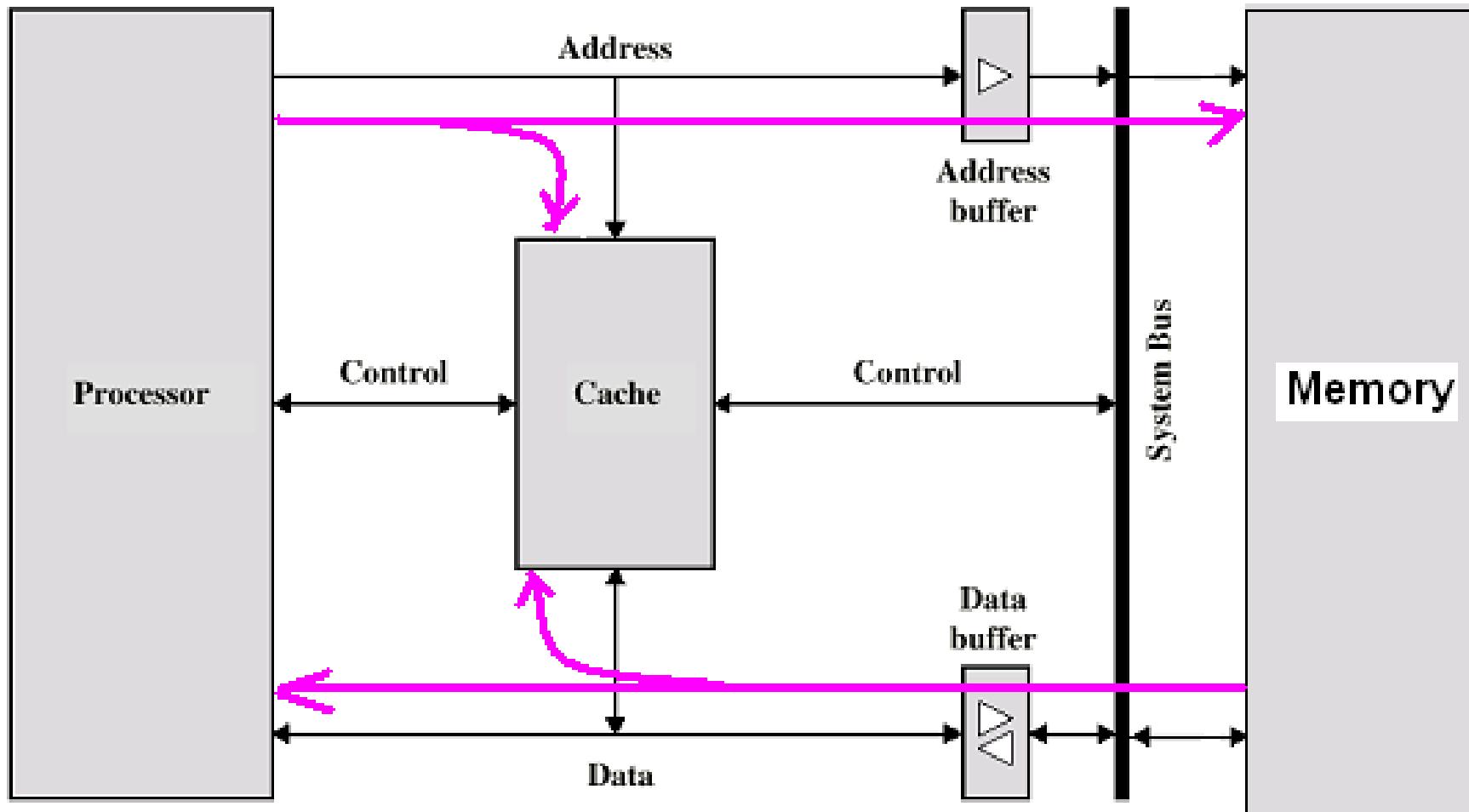
Performance of cache

-
- Hit ratio : Number of Hits / total references to memory
 - Hit
 - Miss

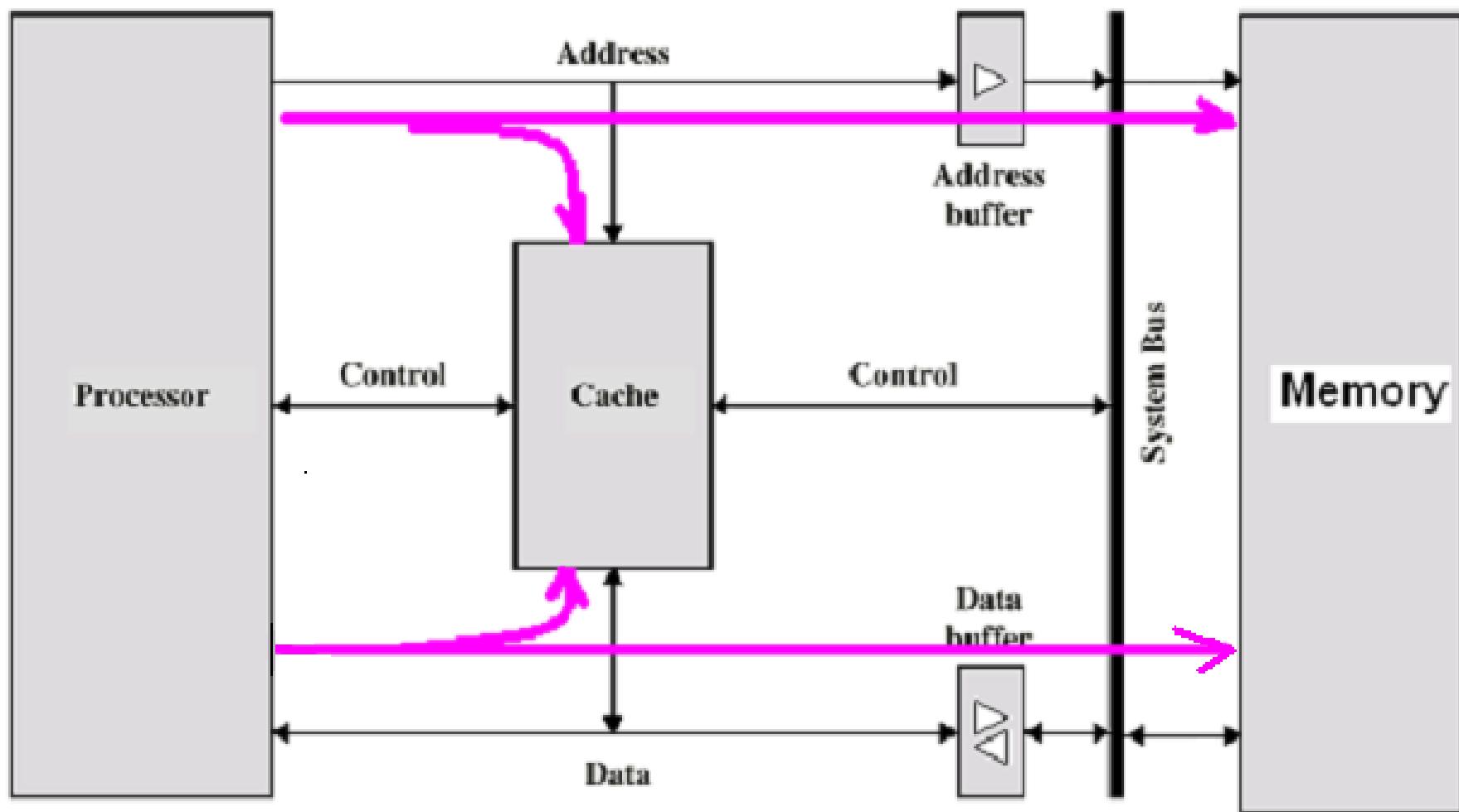
Read Hit



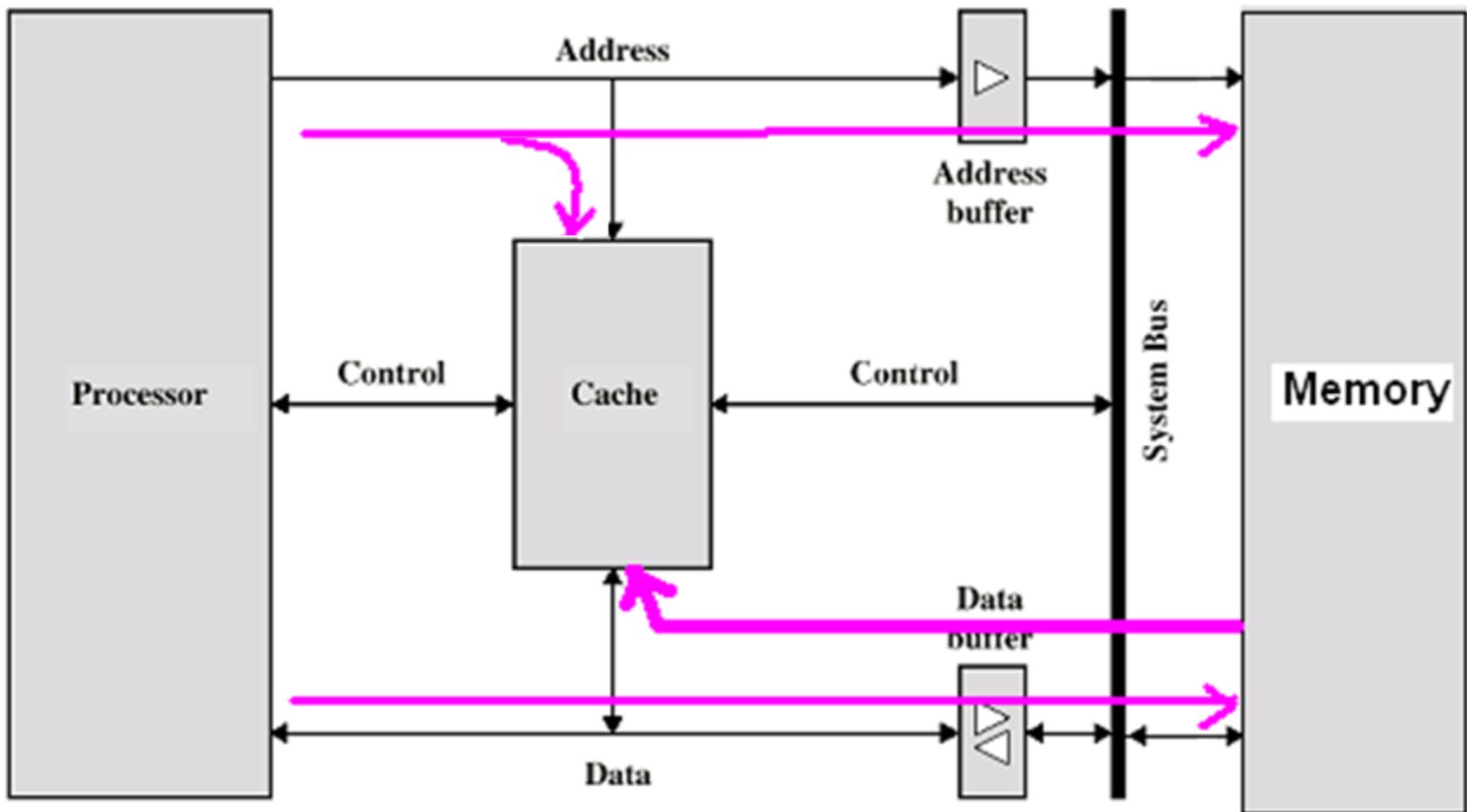
Read Miss



Write hit



Write miss





BITS Pilani
Pilani Campus

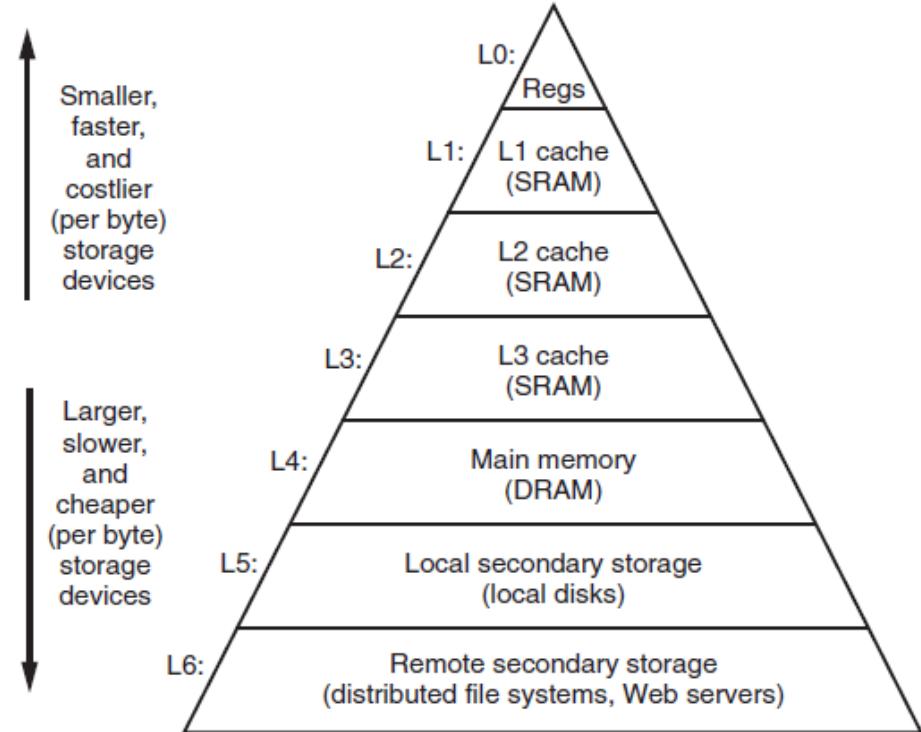
Computer Organization and Software Systems

CONTACT SESSION 4

Dr. Lucy J. Gudino
WILP & Department of CS & IS

The Bottom Line

- How much?
 - Capacity
- How fast?
 - Time is money
- How expensive?



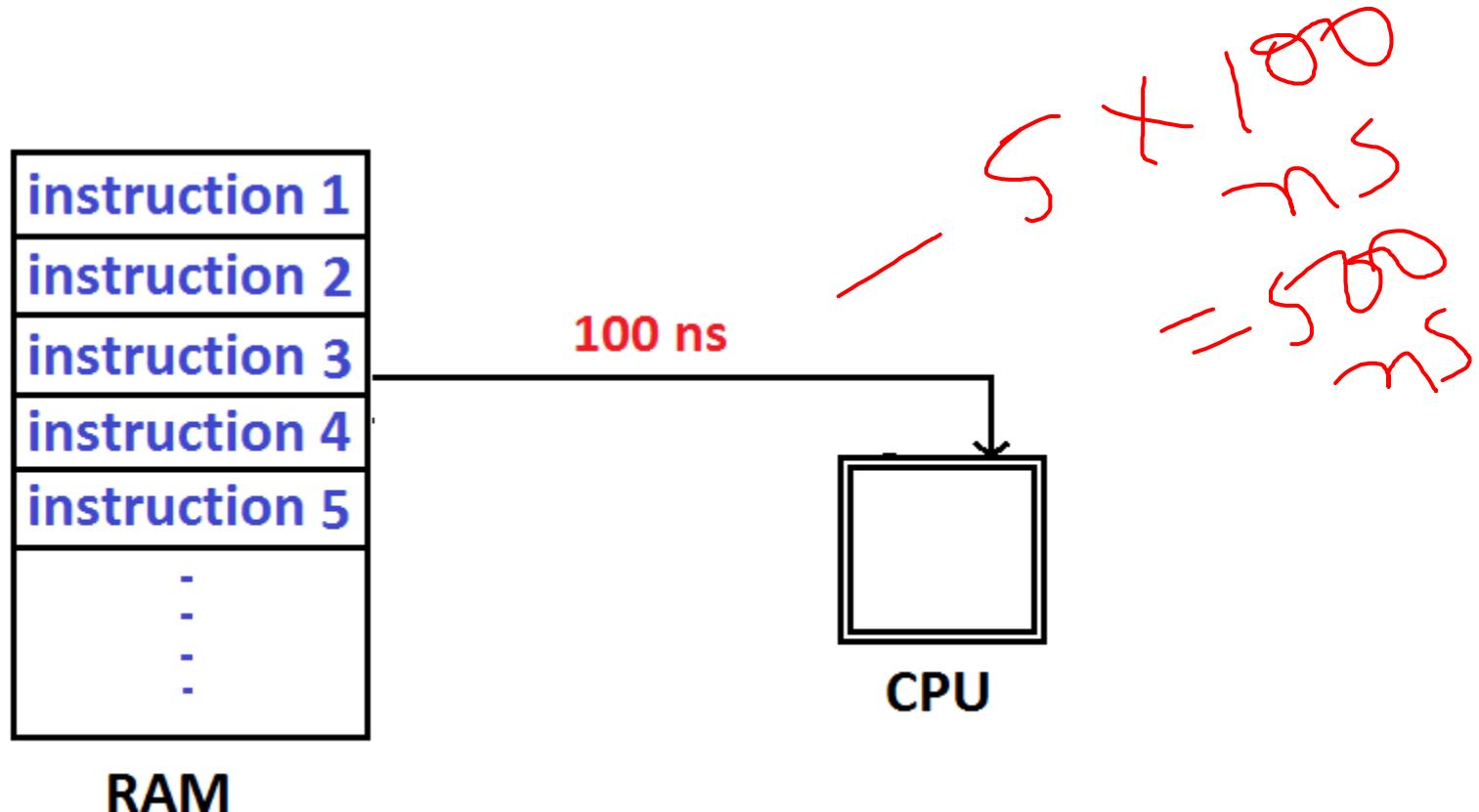
An example of a memory hierarchy.

- Faster access time, ~~highest~~ cost per bit
- Greater capacity, ~~lower~~ cost per bit
- Greater capacity, ~~Still Lower~~ ~~access time~~

Memory Hierarchy

- Registers
 - In CPU
- Internal or Main memory
 - May include one or more levels of cache
 - "RAM"
- External memory
 - Backing store

Performance enhancement - Motivation



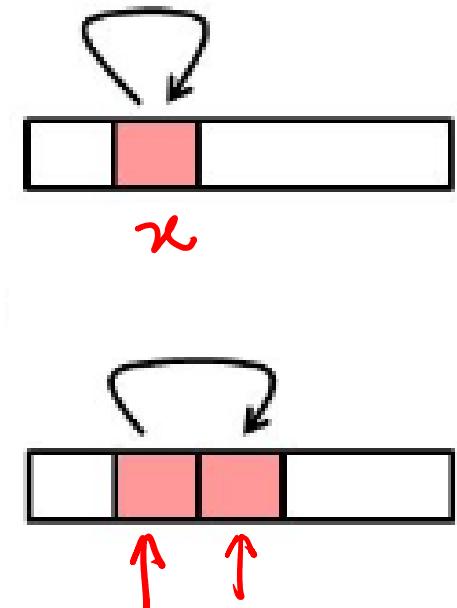
Performance enhancement - Motivation



Locality of Reference

During the course of the execution of a program, memory references tend to cluster

- **Temporal locality:** Locality in time
 - If an item is referenced, it will tend to be referenced again soon
- **Spatial locality:** Locality in space
 - If an item is referenced, items whose addresses are close by will tend to be referenced soon.

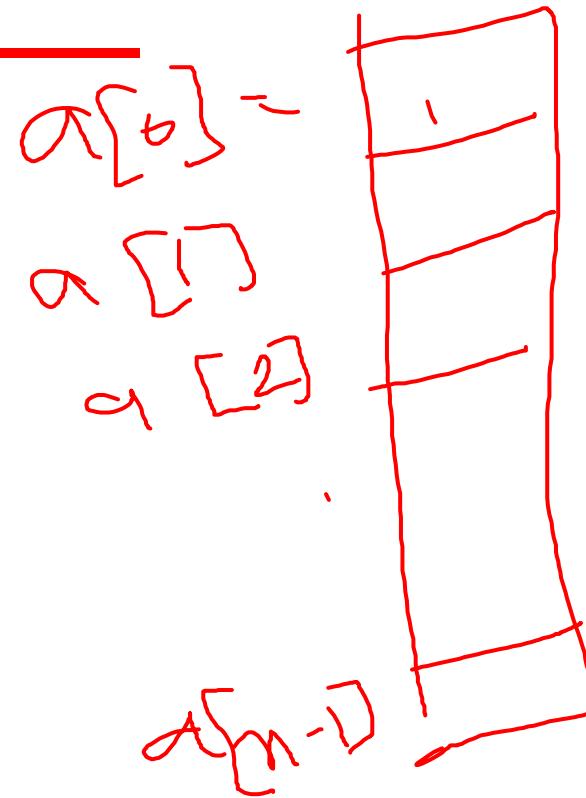


Example

```

product = 1;
for ( i = 0; i < n-1; i++)
    product = product * a[i] ;

```



Data :

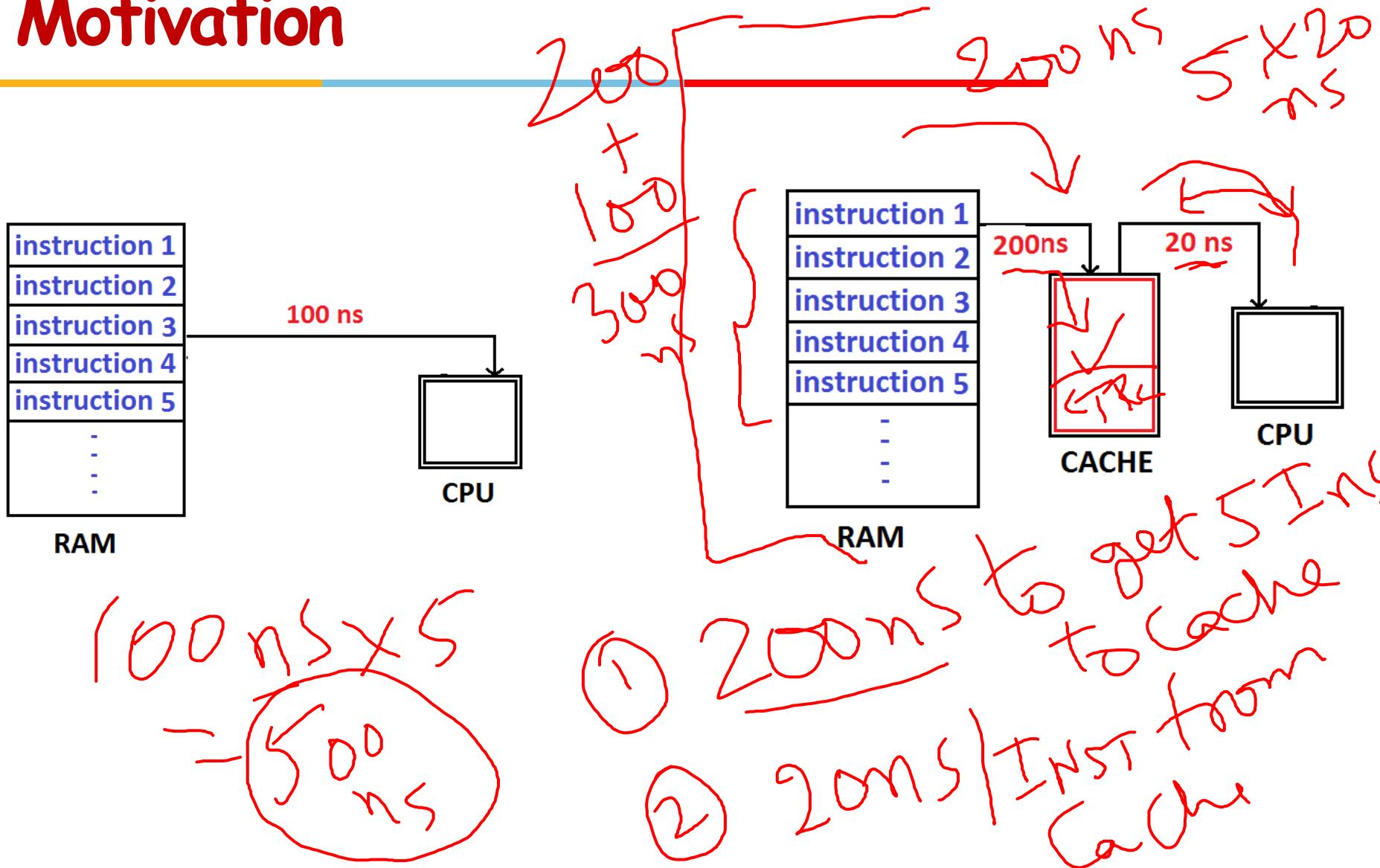
- Access array elements in succession -
spatial locality
- Reference to "product" in each iteration
- Temporal locality

Instructions :

- Reference instructions in sequence :
Spatial locality
- Looping through : Temporal locality

n times product

Performance enhancement - Motivation





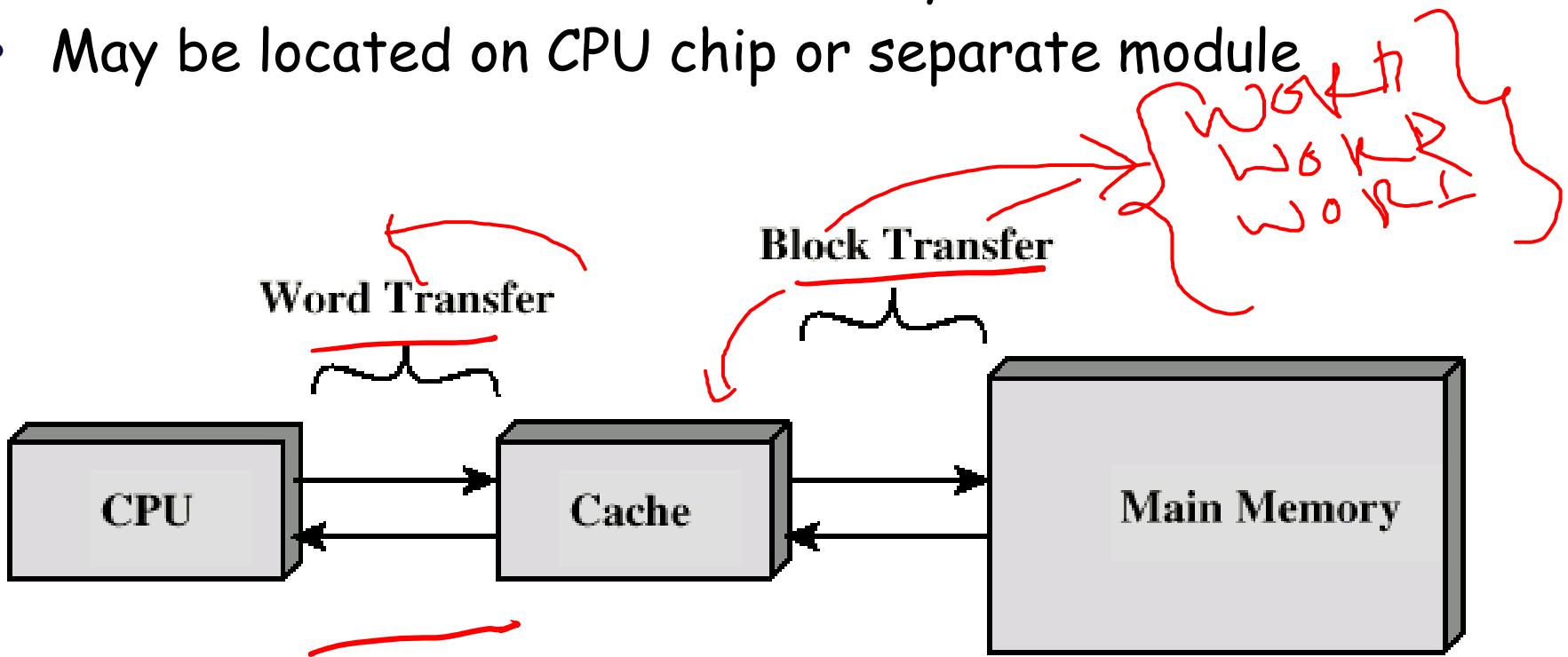
BITS Pilani
Pilani Campus

Cache

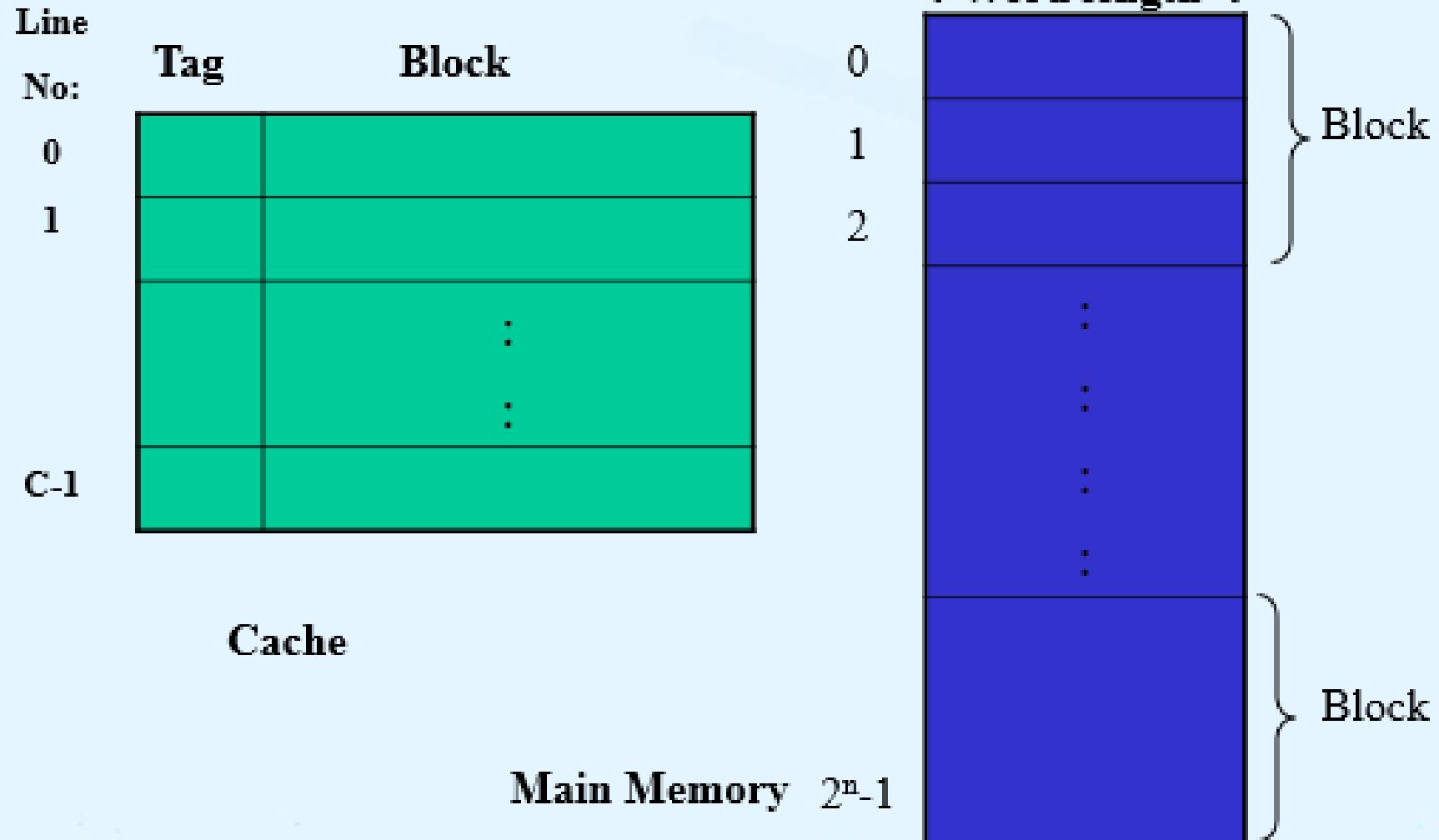


Cache

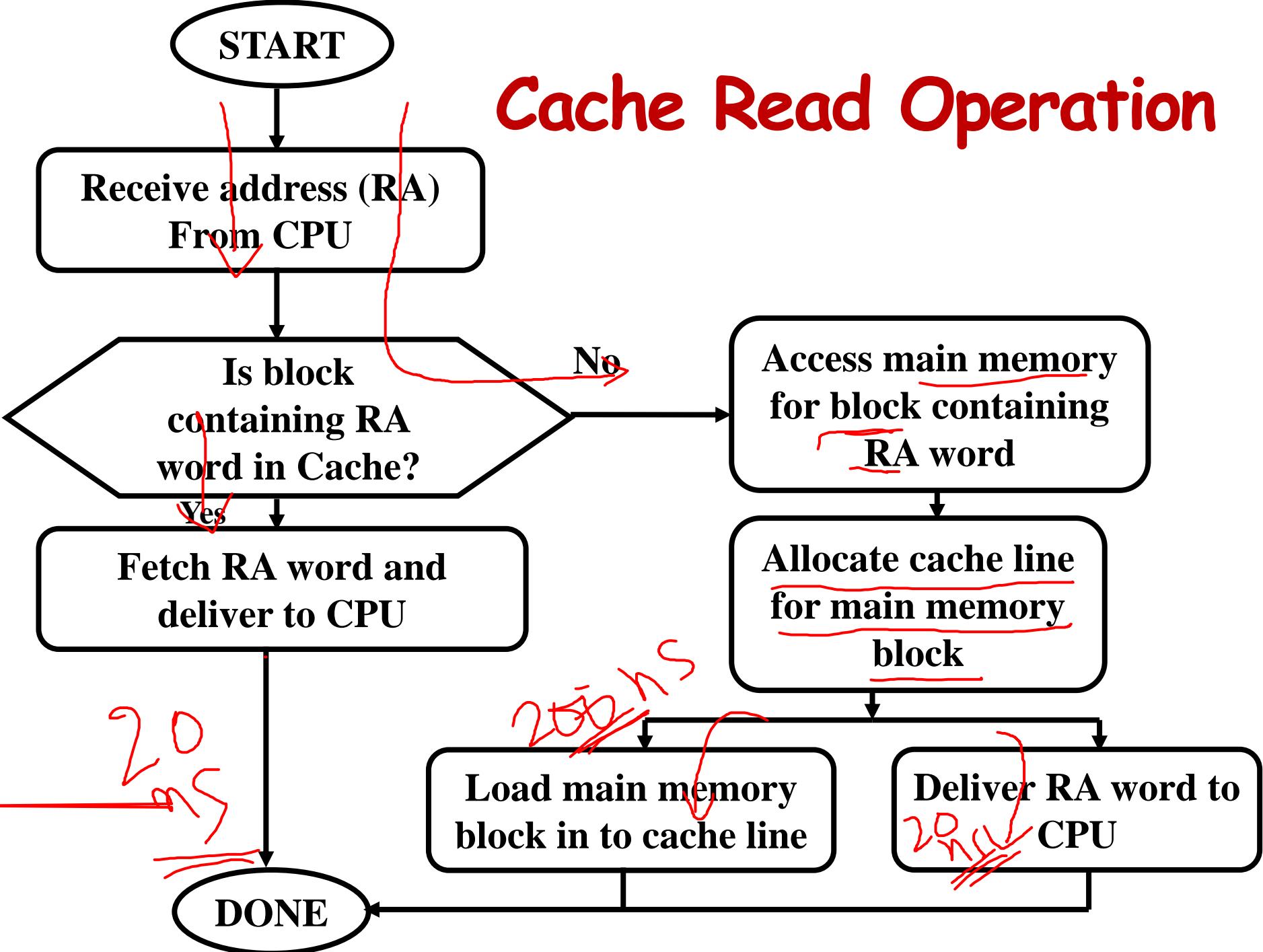
- Small, fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or separate module



Cache and Main Memory Structure



Cache Read Operation



Performance of cache

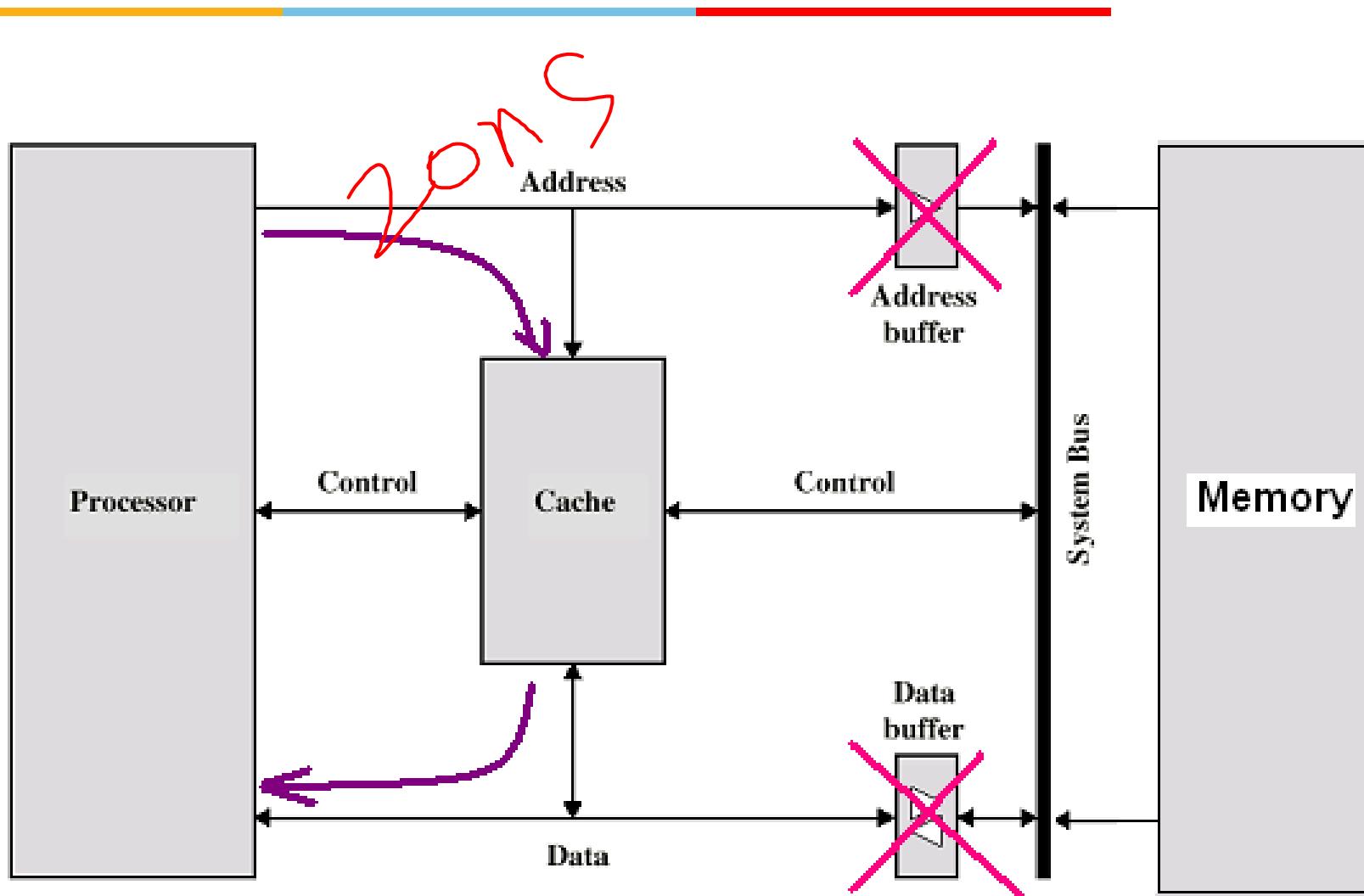
- Hit ratio : Number of Hits / total references to memory

90

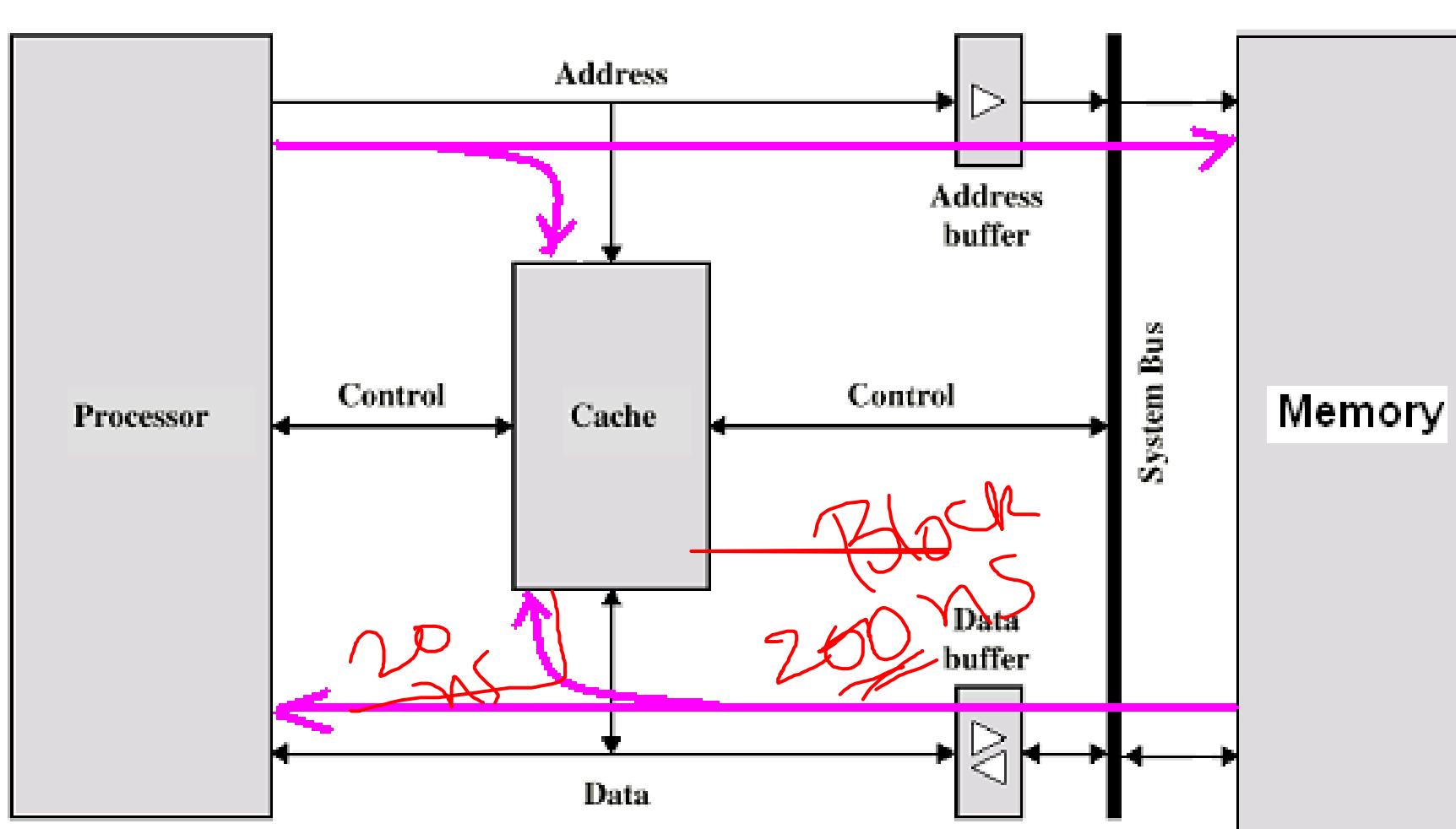
$$= \frac{90}{100} = 0.9$$

- Hit ✓ — 90
- Miss — 10

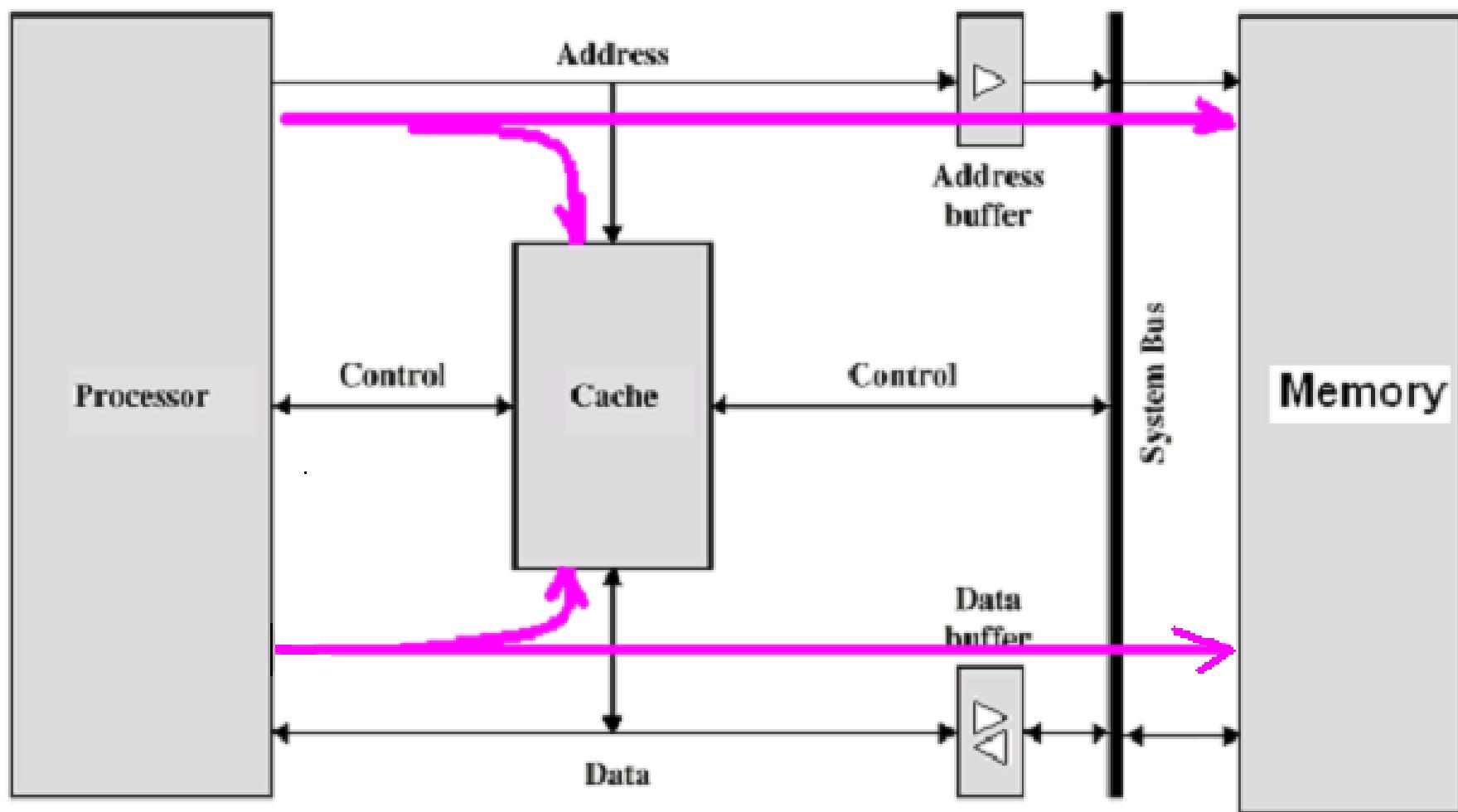
Read Hit



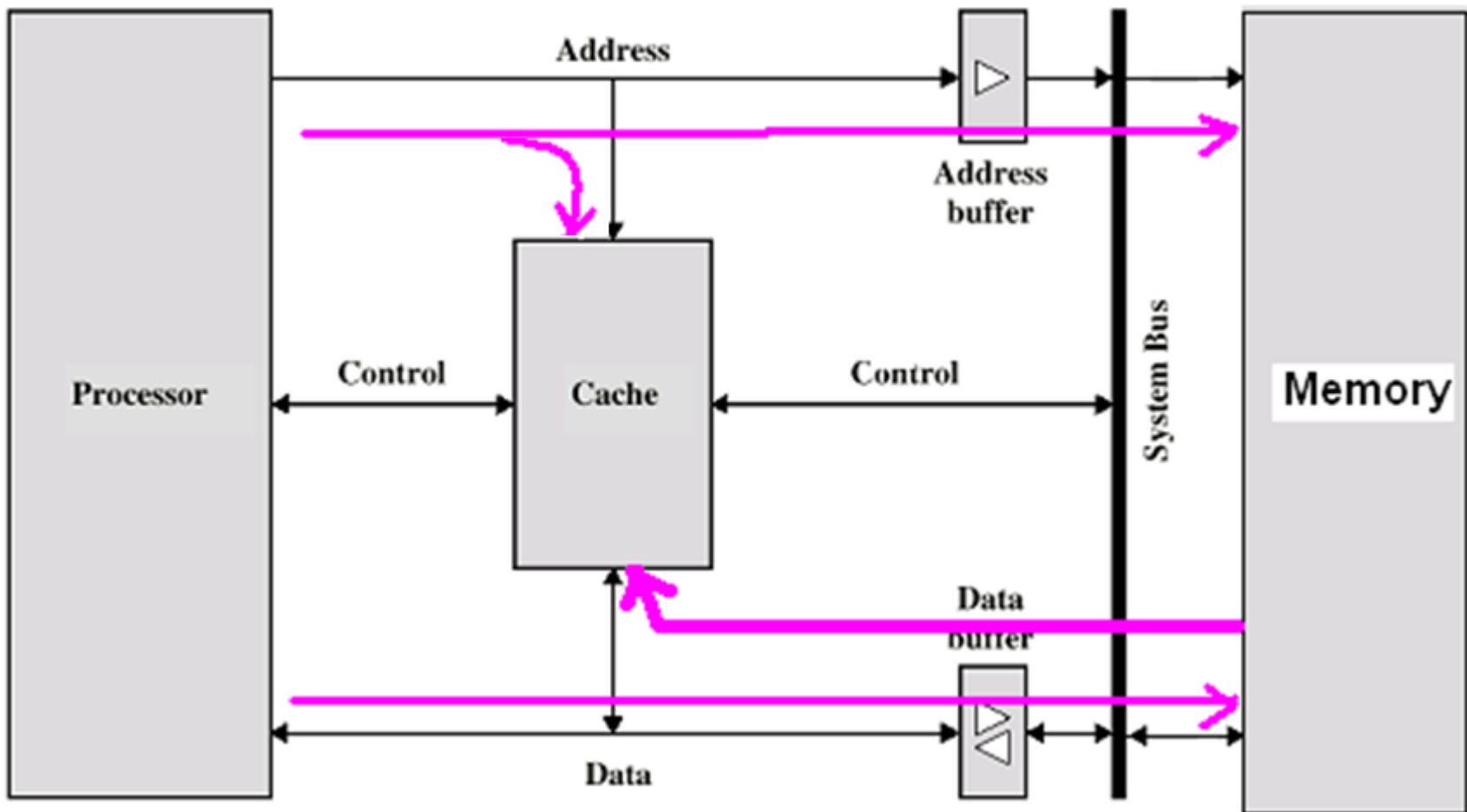
Read Miss



Write hit



Write miss

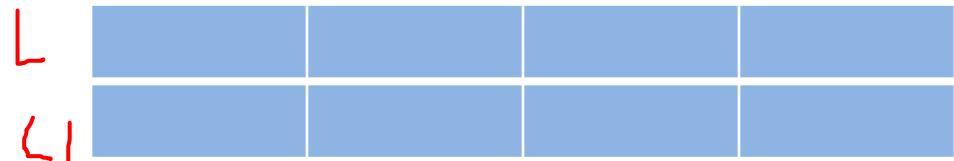
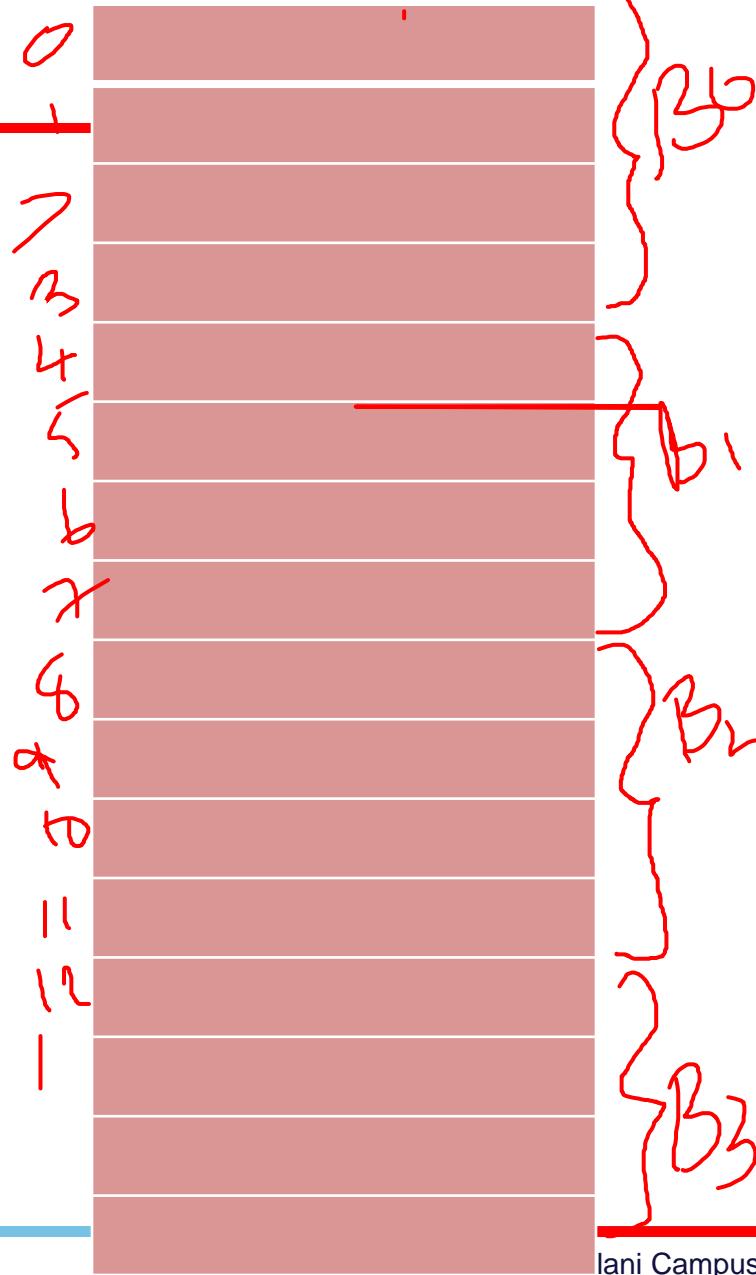


Mapping Function

- How memory blocks are mapped to cache lines
- Three types
 - Direct mapping
 - Associative mapping
 - Set Associative mapping

Direct Mapped Cache

- 16 Bytes main memory
 - How many address bits are required?
- Memory block size is 4 bytes
- Cache of 8 Byte
 - How many ~~cache~~ lines?
 - cache contains 2 lines (4 bytes per Line)



Direct Mapped Cache

- Each block of main memory maps to only one cache line
 - i.e. if a block is in cache, it must be in one specific place
 - $i = j \bmod m$

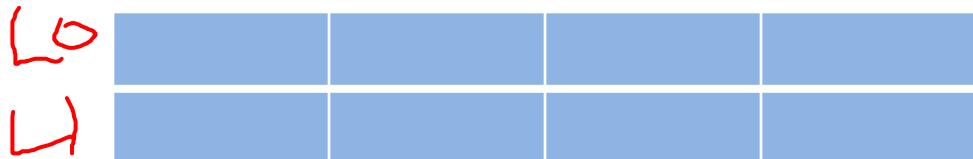
where i = cache line number
 j = main memory block no.
 m = no.of lines in the cache

$$\begin{aligned} J=0, L &= 0^{\circ}/2=0 \\ J=1, L &= 1^{\circ}/2=1 \\ J=2, L &= 2^{\circ}/2=0 \\ J=3, L &= 3^{\circ}/2=1 \end{aligned}$$

Direct Mapped Cache

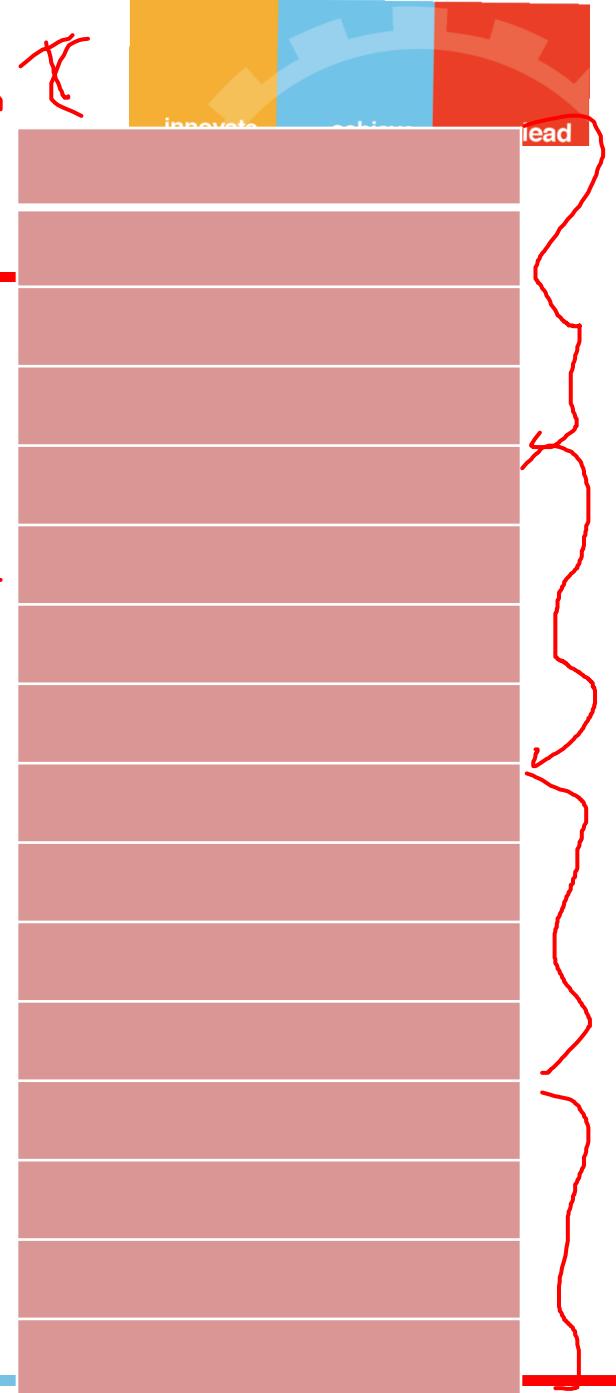
$$i = j \bmod m$$

$$i = 2^m \bmod 2$$

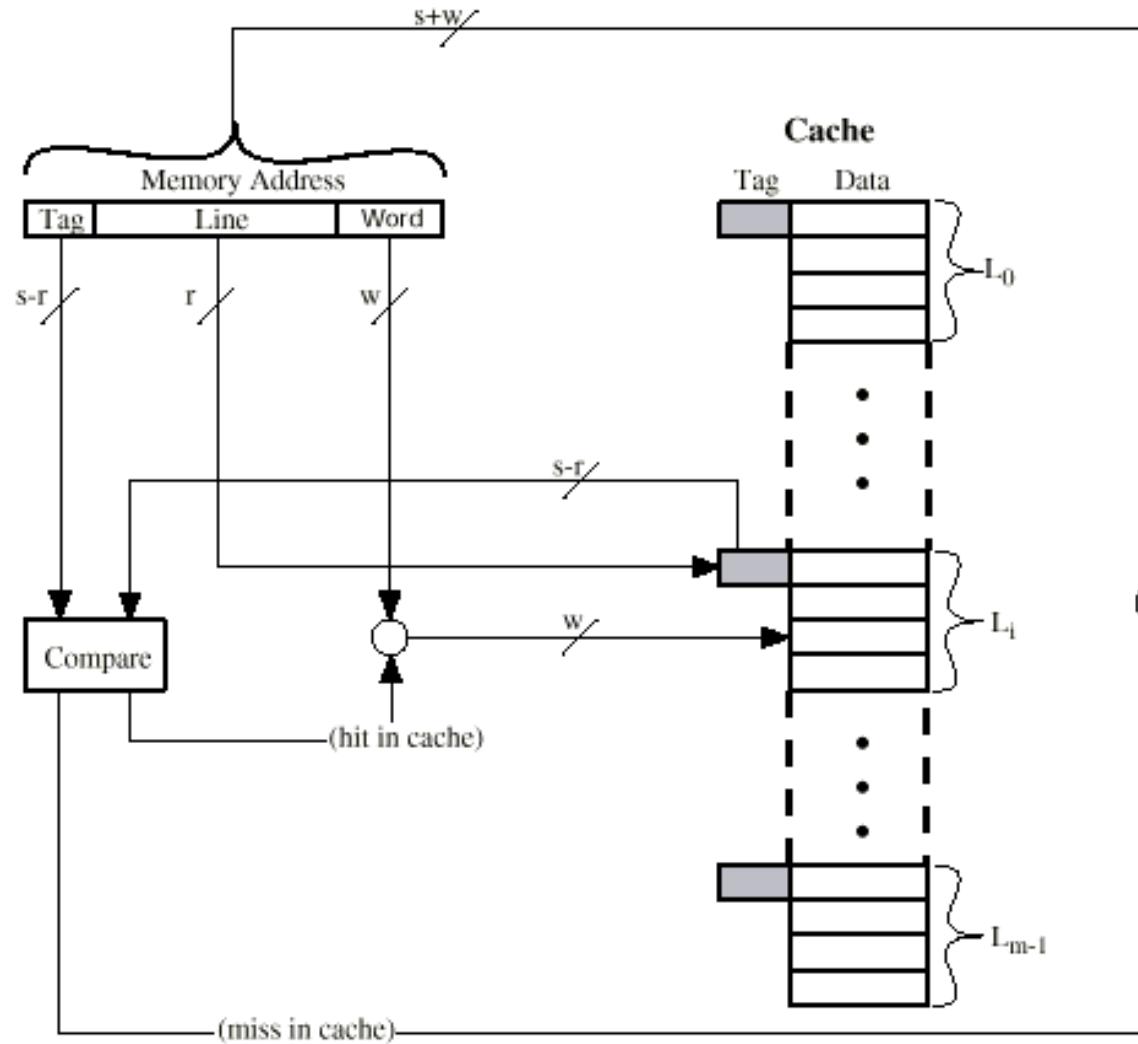


- Address is split in three parts:

- Tag
- Line
- Word



Direct Mapped Cache Organization



Direct mapped cache - Summary



$$15, w=2 \rightarrow s$$

Address length = $(s+w)$ bits

$$13+2$$

$$\frac{15}{2} = 32 \text{ KW}$$

Number of addressable units = 2^{s+w} words or bytes

Block size = line size = 2^w words or bytes

$$2^2 = 4$$

Number of blocks in main memory = $\frac{2^{s+w}}{2^w} = 2^s = 2^{15}/2^2$

Number of lines in cache = $m = 2^r = 64$

$$= 2^{15}$$

Size of tag = $(s-r)$ bits

$$13 - 6 = 7$$

$$256 \times (4W)$$

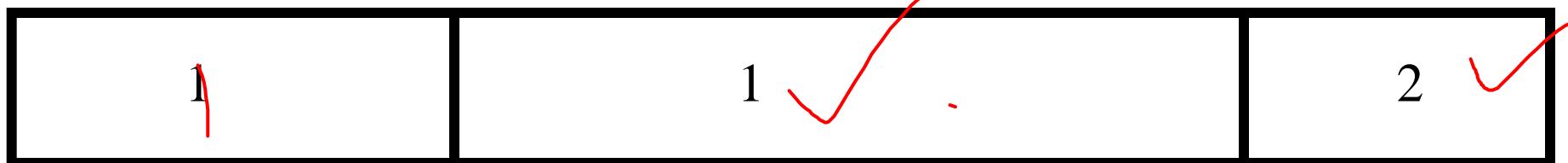
Direct mapped cache - Summary

innovate

achieve

lead

16 Byte Main Memory = 2⁴
16 Byte Cache = 4
Block = 4



Tag s-r

Line or Slot r

Word w

Direct mapped cache-pros & cons

- Simple
- Inexpensive
- Fixed location for given block
 - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

Problem 1 : Direct Mapped Cache

Given :

- Cache of 64KByte, Cache block of 4 bytes
- 16MBytes main memory



Find out

- a) Number of bits required to address the main memory
- b) Number of blocks in main memory
- c) Number of cache lines
- d) Number of bits required to identify a word (byte) in a block
- e) Number of bits to identify a block
- f) Tag, Line, Word

Solution 1



Given :

- Cache of 64kByte, Cache block of 4 bytes
- 16MBytes main memory

Find out

a) Number of bits required to address the main memory

$$2^{24}$$

b) Number of blocks in main memory

$$4 \text{ MB} / 4 \text{ KB Block}$$

c) Number of cache lines

$$(64 \text{ KB}) / 4 \text{ B} = 16 \text{ K lines}$$

Solution 1

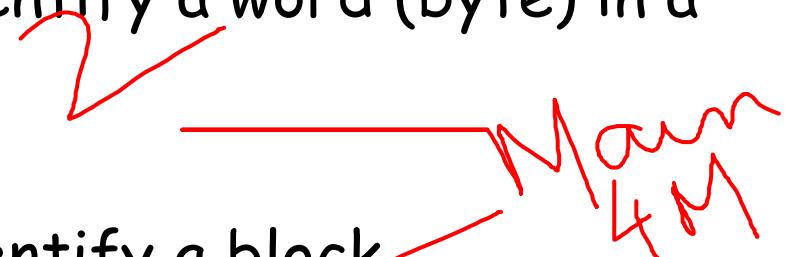


Given :

- Cache of 64kByte, Cache block of 4 bytes
- 16MBytes main memory

Find out

d) Number of bits required to identify a word (byte) in a block?

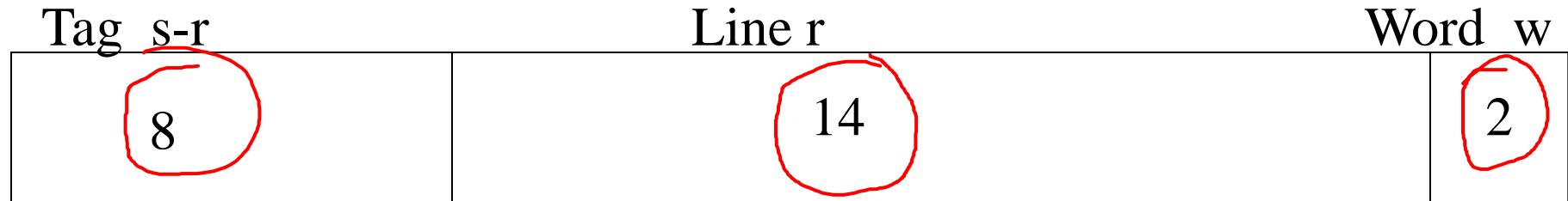


e) Number of bits required to identify a block

A red arrow points from the handwritten text '16K lines' to a circled number '14'.

$$16K \text{ lines} = 14$$

f) Tag, Line, Word



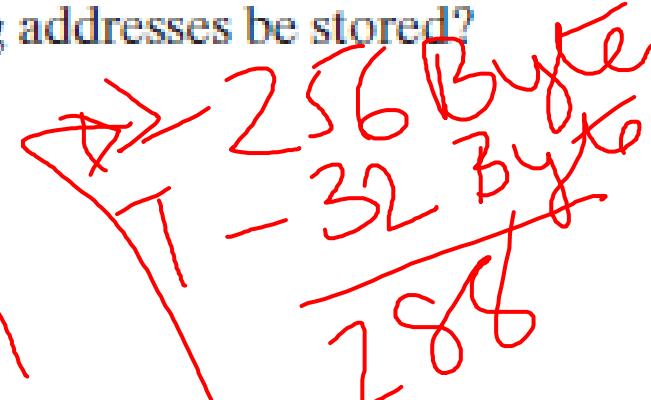
Problem 2

$$T=8L=5W=3$$

Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

- a. How is a 16-bit memory address divided into tag, line number, and byte number?
- b. Into what line would bytes with each of the following addresses be stored?

0001	0001	0001	1011	3
1100	0011	0011	0100	6
1101	0000	0001	1101	3
1010	1010	1010	1010	21



- c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?
- d. How many total bytes of memory can be stored in the cache?
- e. Why is the tag also stored in the cache?

Tag
32+8 bits

+ 256 bytes

Solution 2

Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

- a. How is a 16-bit memory address divided into tag, line number, and byte number?

- b. Into what line would bytes with each of the following addresses be stored?

0001 0001 0001 1011

1100 0011 0011 0100

1101 0000 0001 1101

1010 1010 1010 1010

Solution 2

Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

- c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?

0001 1010 0001 1**000**

0001 1010 0001 1**001**

0001 1010 0001 1**010** : given in the problem

0001 1010 0001 1**011**

0001 1010 0001 1**100**

0001 1010 0001 1**101**

0001 1010 0001 1**110**

0001 1010 0001 1**111**

Solution 2

Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

d. How many total bytes of memory can be stored in the cache?

28672

~~W32 × 8 ⇒ 2¹⁶ BY⁶~~

e. Why is the tag also stored in the cache?

Referline

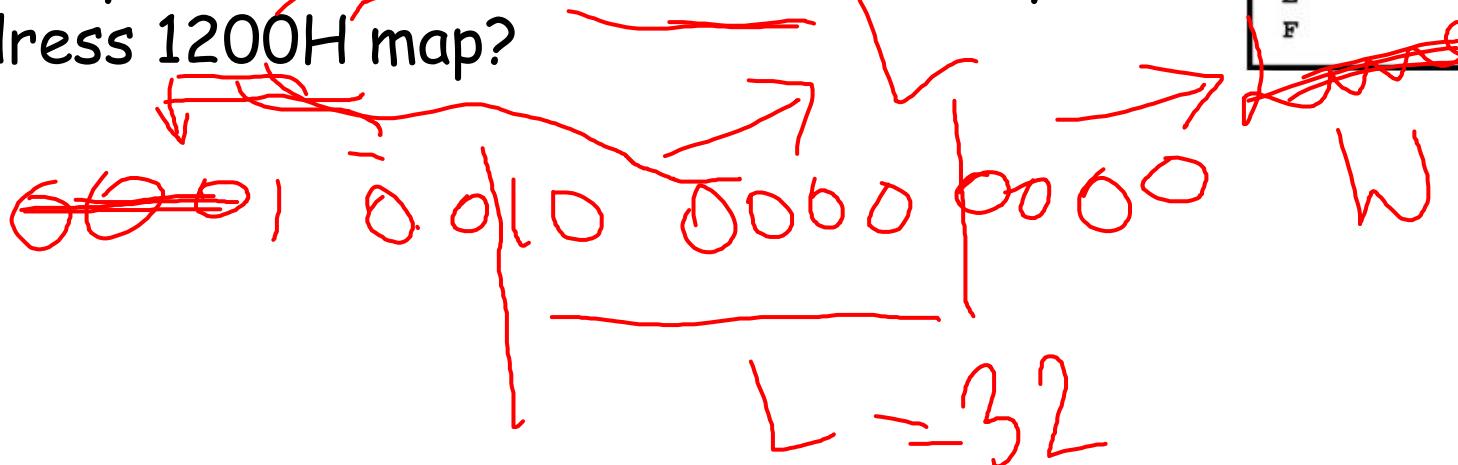
Problem 3

~~F3~~ L=6 W=4

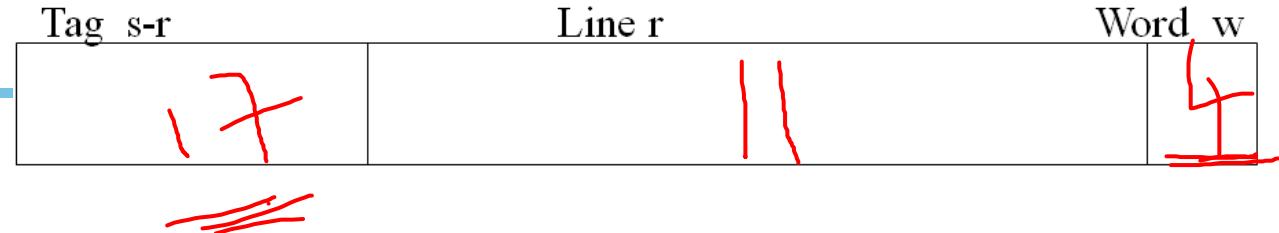
	Hexadecimal	Binary	Decimal
0	0000	0000	0
1	0001	0001	1
2	0010	0010	2
3	0011	0011	3
4	0100	0100	4
5	0101	0101	5
6	0110	0110	6
7	0111	0111	7
8	1000	1000	8
9	1001	1001	9
A	1010	1010	10
B	1011	1011	11
C	1100	1100	12
D	1101	1101	13
E	1110	1110	14
F	1111	1111	15

~~SK = 310~~

Consider a direct-mapped cache with 64 cache lines and a block size of 16 bytes and main memory of 8K (Byte addressable memory). To what line number does byte address 1200H map?



Problem 4



The system uses a L1 cache with direct mapping and 32-bit address format is as follows:

bits 0 - 3 = offset (word) .

bits 4 - 14 = index bits (Line)

bits 15 - 31 = tag ~~17~~

a) What is the size of cache line?

$$2^4 = 16$$

b) How many Cache lines are there?

$$2^{11} = 2K$$

c) How much space is required to store the tags in the L1 cache?

$$17 \times 2K = 17 + 2048$$

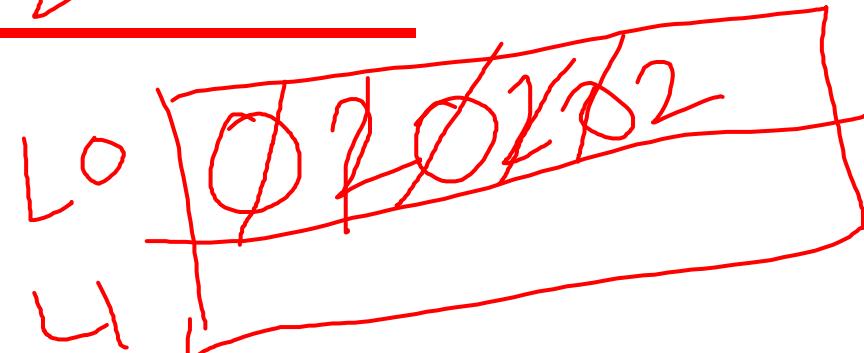
d) What is the total Capacity of cache including tag storage?

$$2^{11} + 16 + (17 \times 2K) = 2048$$

Problem 5

$$\begin{array}{r} 0 \\ 2 \\ \hline 0 & 0 \\ 0 & 2 \\ \hline 2 & 0 \\ 0 & 2 \\ \hline \end{array}$$

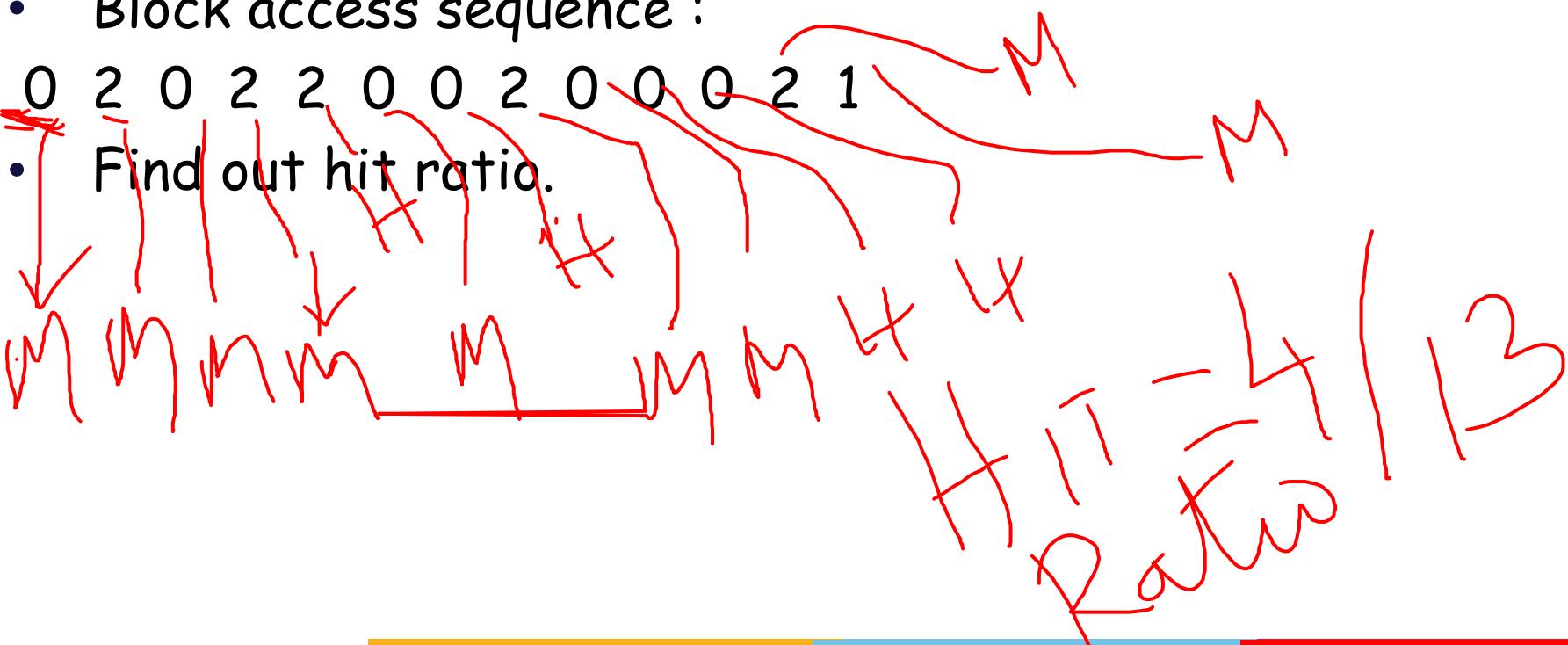
- 16 Bytes main memory,
Memory block size is 4 bytes,
Cache of 8 Byte (cache is 2
lines of 4 bytes each)



- Block access sequence :

0 2 0 2 2 0 0 2 0 0 0 2 1

- Find out hit ratio.



End of Session 4

Until next

Session 5



BITS Pilani
Pilani Campus

Computer Organization and Software Systems

CONTACT SESSION 5

Prepared By: Dr. Lucy J. Gudino
Instructor: <Prof. C R Sarma>

Problem 2 - Direct Mapping

Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

- a. How is a 16-bit memory address divided into tag, line number, and byte number?
- b. Into what line would bytes with each of the following addresses be stored?

0001	0001	0001	1011
1100	0011	0011	0100
1101	0000	0001	1101
1010	1010	1010	1010
- c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?
- d. How many total bytes of memory can be stored in the cache?
- e. Why is the tag also stored in the cache?

Solution 2

Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

- a. How is a 16-bit memory address divided into tag, line number, and byte number?

- b. Into what line would bytes with each of the following addresses be stored?

0001 0001 0001 1011

1100 0011 0011 0100

1101 0000 0001 1101

1010 1010 1010 1010

Solution 2

Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

- c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?

0001 1010 0001 1**000**

0001 1010 0001 1**001**

0001 1010 0001 1**010** : given in the problem

0001 1010 0001 1**011**

0001 1010 0001 1**100**

0001 1010 0001 1**101**

0001 1010 0001 1**110**

0001 1010 0001 1**111**

Solution 2

Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

- d. How many total bytes of memory can be stored in the cache?

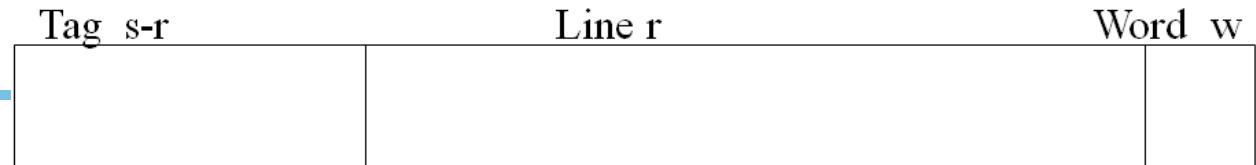
- e. Why is the tag also stored in the cache?

Problem 3

Consider a direct-mapped cache with 64 cache lines and a block size of 16 bytes and main memory of 8K (Byte addressable memory). To what line number does byte address 1200H map?

	Hexadecimal	Binary	Decimal
0	0000	0000	0
1	0001	0001	1
2	0010	0010	2
3	0011	0011	3
4	0100	0100	4
5	0101	0101	5
6	0110	0110	6
7	0111	0111	7
8	1000	1000	8
9	1001	1001	9
A	1010	1010	10
B	1011	1011	11
C	1100	1100	12
D	1101	1101	13
E	1110	1110	14
F	1111	1111	15

Problem 4



The system uses a L1 cache with direct mapping and 32-bit address format is as follows:

bits 0 - 3 = offset (word)

bits 4 - 14 = index bits (Line)

bits 15 - 31 = tag

- What is the size of cache line?
- How many Cache lines are there?
- How much space is required to store the tags in the L1 cache?
- What is the total Capacity of cache including tag storage?

Problem 5

- 16 Bytes main memory,
Memory block size is 4 bytes,
Cache of 8 Byte (cache is 2
lines of 4 bytes each)
- Block access sequence :

0 2 0 2 2 0 0 2 0 0 0 2 1

• Find out hit ratio.

L0



HR=4/13

Problem 6

KB = 10
MB = 2

Suppose a 1024-byte cache has an access time of 0.1 microseconds and the main memory stores 1 Mbytes, with an access time of 1 microsecond. A referenced memory block that is not in cache must be loaded into cache.

Answer the following questions:

- What is the number of bits needed to address the main memory?
- If the cache hit ratio is 95%, what is the average access time for a memory reference?

Solution 6

a) 20 bits

b) If the cache hit ratio is 95%, what is the average access time for a memory reference?

Avg access time = hit ratio * cache access +

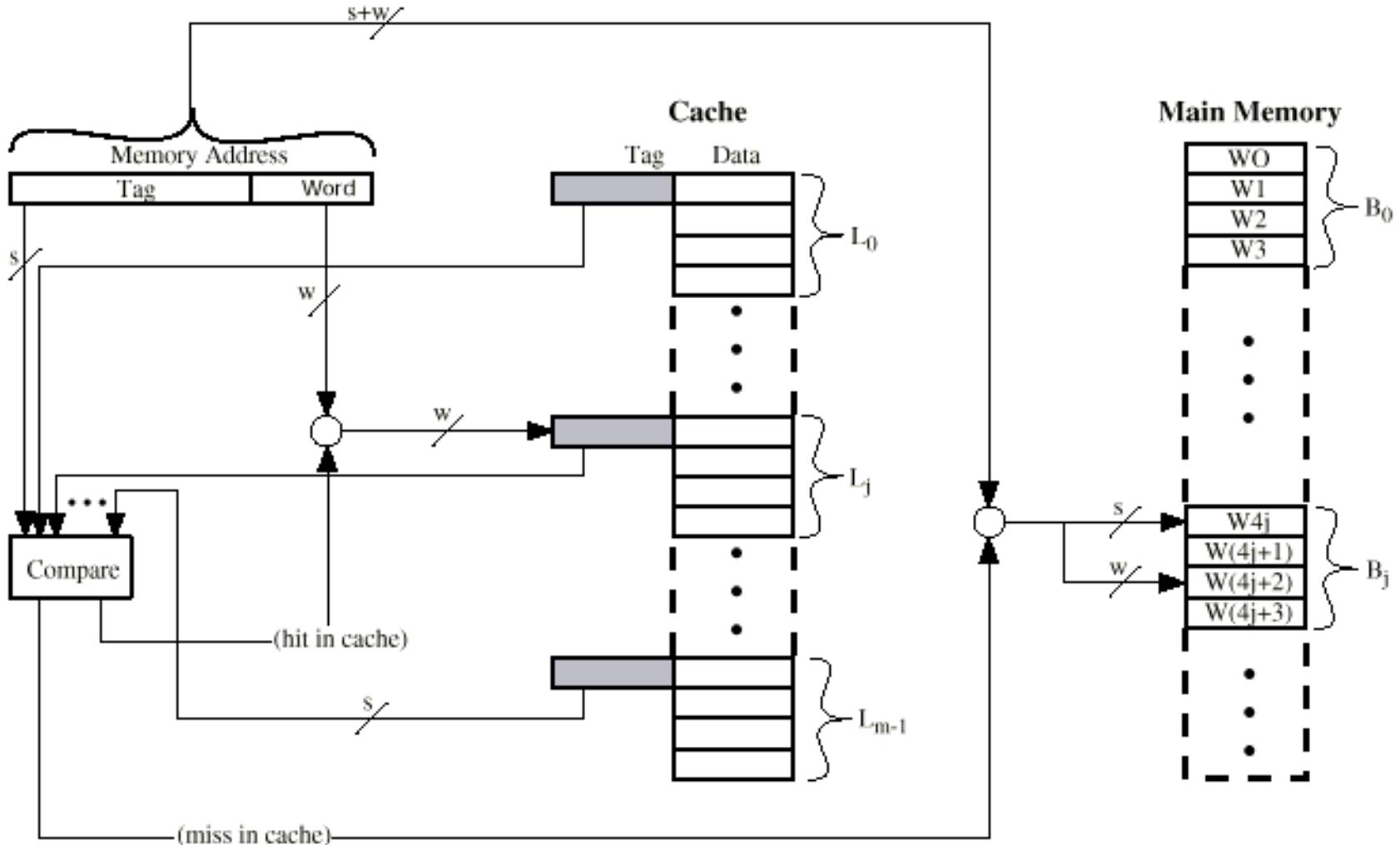
(1- hit ratio) * (cache access + memory access)

$$\begin{aligned}
 &= 0.95 \times \overline{0.1\mu s} + 0.05(\cancel{0.1\mu s}) \\
 &= 0.95 \times 0.1 + 0.05 \\
 &\approx 0.095 + 0.05
 \end{aligned}$$

Associative Mapping

- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line's tag is examined for a match
- Cache searching gets expensive

Associative Cache Organization



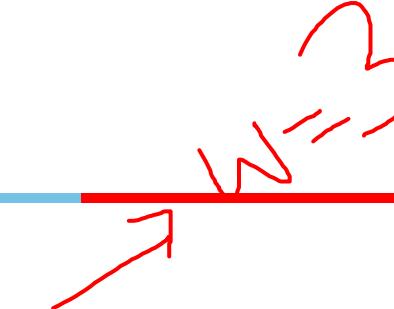
Associative Mapping Summary

- Address length = $(s + w)$ bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^w words or bytes
- Number of blocks in main memory = $2^{s+w}/2^w = 2^s$
- Number of lines in cache = undetermined
- Size of tag = s bits

Problem 7

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory



$$\begin{aligned} & \text{Cache size: } 128 \text{ kByte} = 128 \times 1024 \times 8 \text{ bits} \\ & \text{Main memory size: } 32 \text{ MBytes} = 32 \times 1024 \times 1024 \times 8 \text{ bits} \\ & \text{Block size: } 8 \text{ bytes} = 8 \times 8 \text{ bits} \\ & \text{Number of blocks: } \frac{32 \times 1024 \times 1024 \times 8}{8 \times 8} = 2^{25} \end{aligned}$$

Find out

- a) Number of bits required to address the memory 2²⁵
- b) Number of blocks in main memory 2²²
- c) Number of cache lines $\frac{128 \times 1024 \times 8}{8 \times 8} = 16$
- d) Number of bits required to identify a word (byte) in a block? 3
- e) Tag, Word $(25-3), 3 = 22, 3$

Problem 8

Cache of 64KByte, Cache block of 4 bytes and 16 M Bytes main memory and associative mapping.

Fill in the blanks:

Number of bits in main memory address = 24

Number of lines in the cache memory = 16k

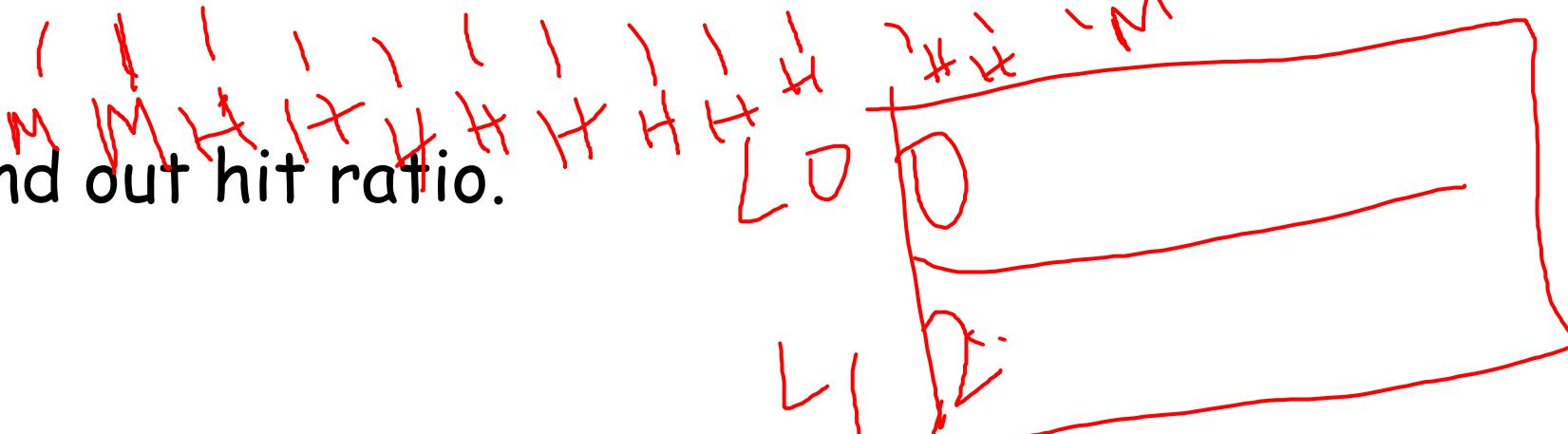
Word bits = 2

Tag bits = 22

Problem 9

- 16 Bytes main memory, Memory block size is 4 bytes, Cache of 8 Byte (cache is 2 lines of 4 bytes each) and associative mapping
- Block access sequence :

0 2 0 2 2 0 0 2 0 0 0 2 1



- Find out hit ratio.

Set Associative Mapping

- Cache is divided into a number of sets (v sets each with k lines)

Tag	Set	Word
-----	-----	------

- $m = v * k$

$$i = j \text{ modulo } v$$

where $i = \text{cache set number}$

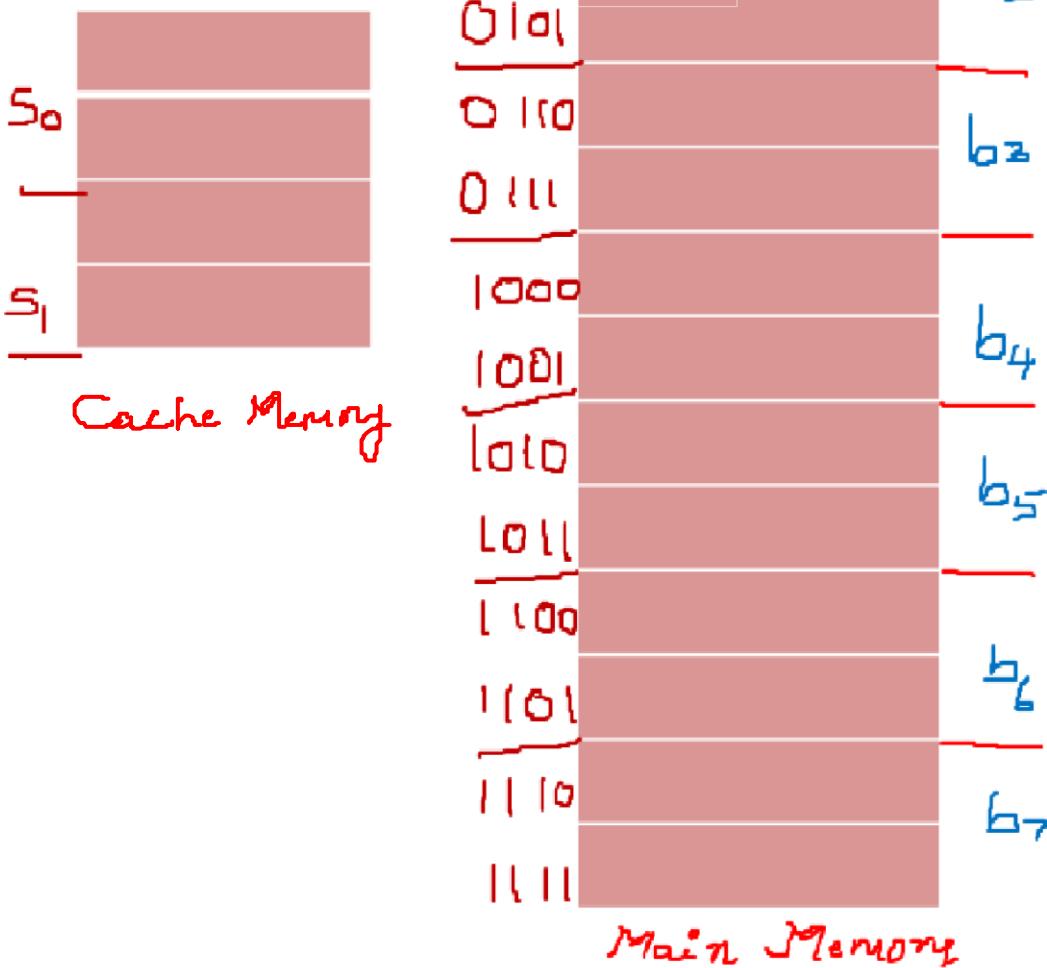
$j = \text{main memory block number}$

$m = \text{number of lines in the cache}$

- Each set contains ' k ' number of lines
- A given block maps to any line in a given set
 - e.g. Block B can be in any line of set i
- e.g. 2 lines per set
 - 2 way set associative mapping
 - A given block can be in one of 2 lines in only one set

Example

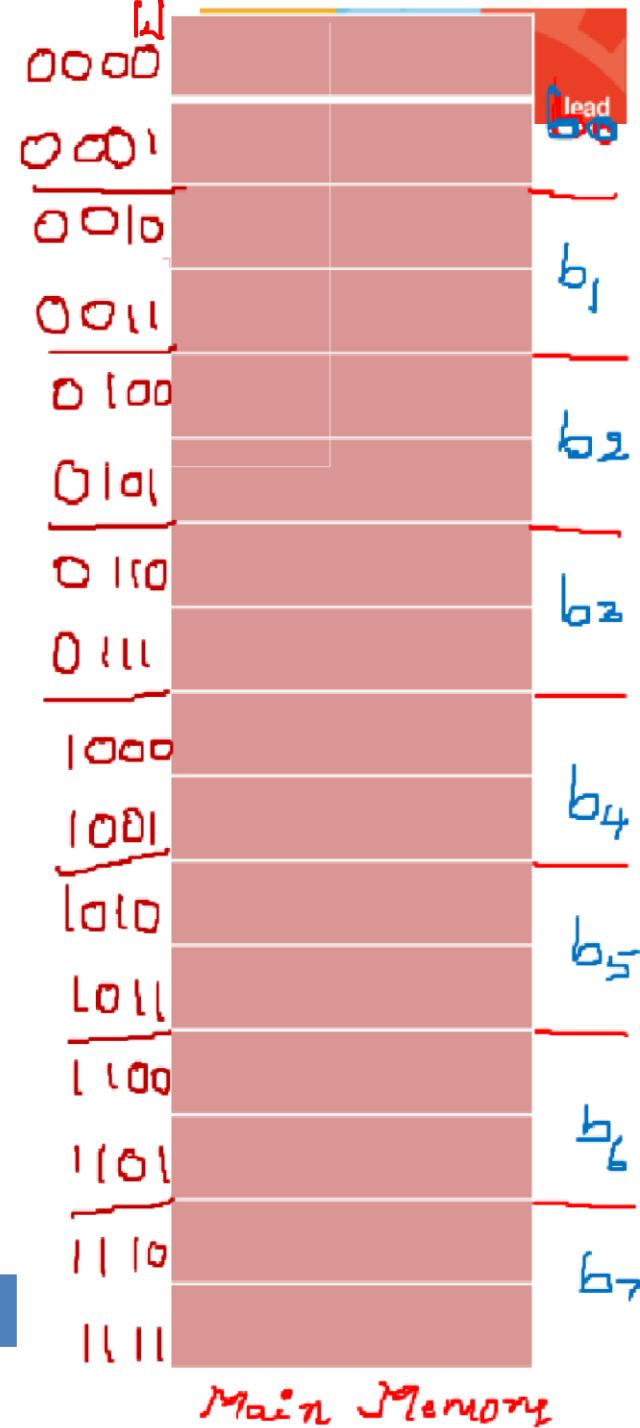
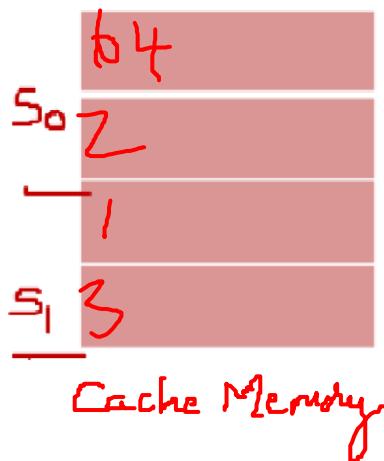
- 16 Bytes main memory, Block Size is 2 Bytes,
- Cache of 8 Bytes, 2 way set associative cache
 - # address bits
 - Cache line size
 - # main memory blocks
 - # Number of cache lines
 - # lines per set
 - # of sets



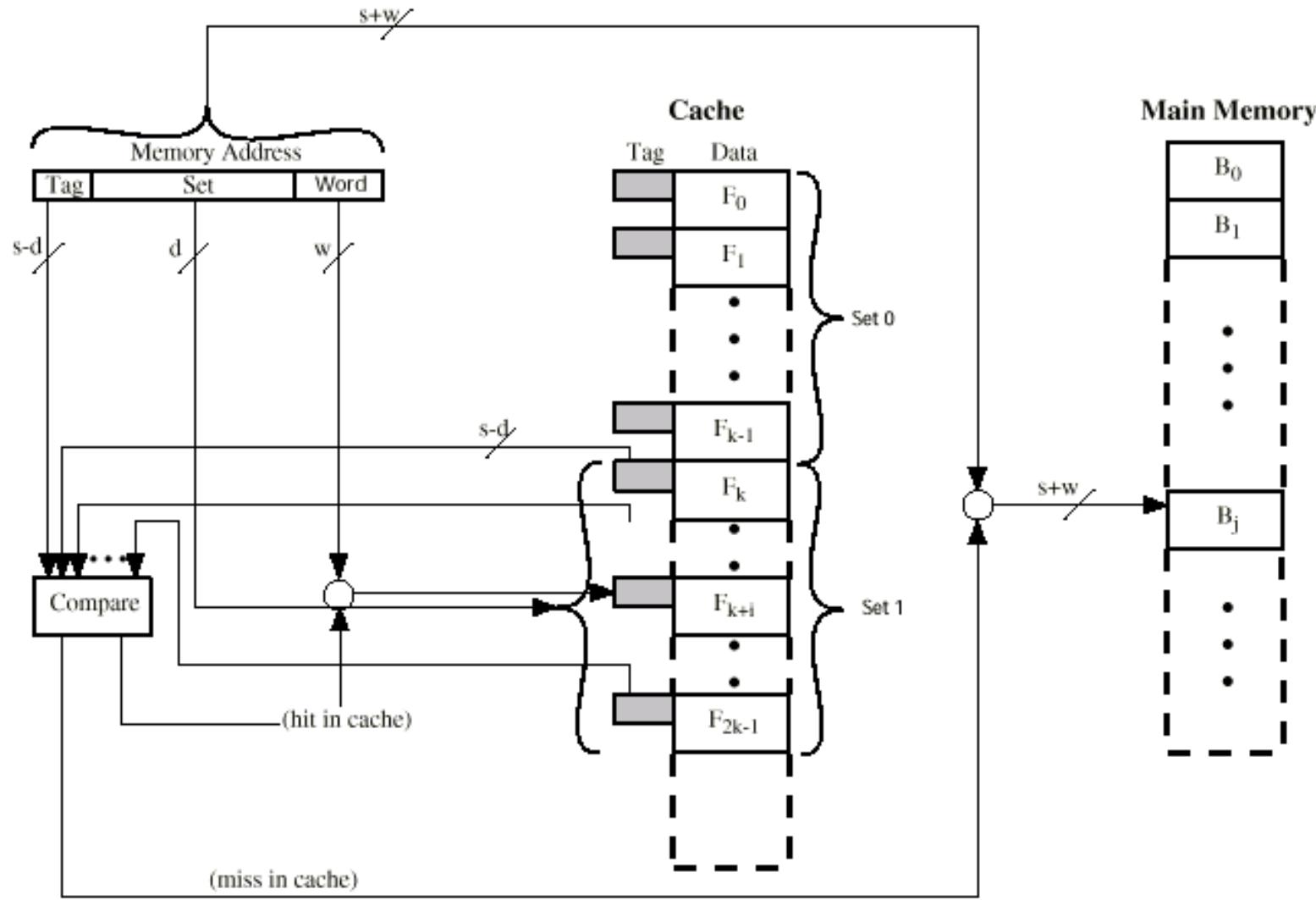
Example - Mapping Function

$i = j \text{ modulo } v$	Set #
0%2	0
1%2	1
2%2	
3%2	
4%2	
5%2	
6%2	
7%2	

TAG | SET | WORD



Set Associative Cache Organization



Set Associative Mapping Summary

10 5 Sets, $d = 3$

Address length = $(s + w)$ bits

Number of addressable units = 2^{s+w} words or bytes

Block size = line size = 2^w words or bytes $= 4 = 2^2$

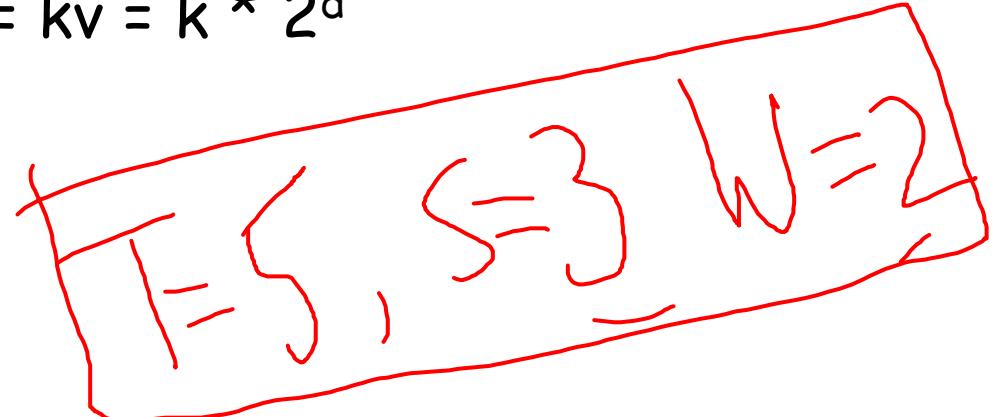
Number of blocks in main memory = 2^s

Number of lines in set = k

Number of sets = $v = 2^d$

Number of lines in cache = $kv = k * 2^d$

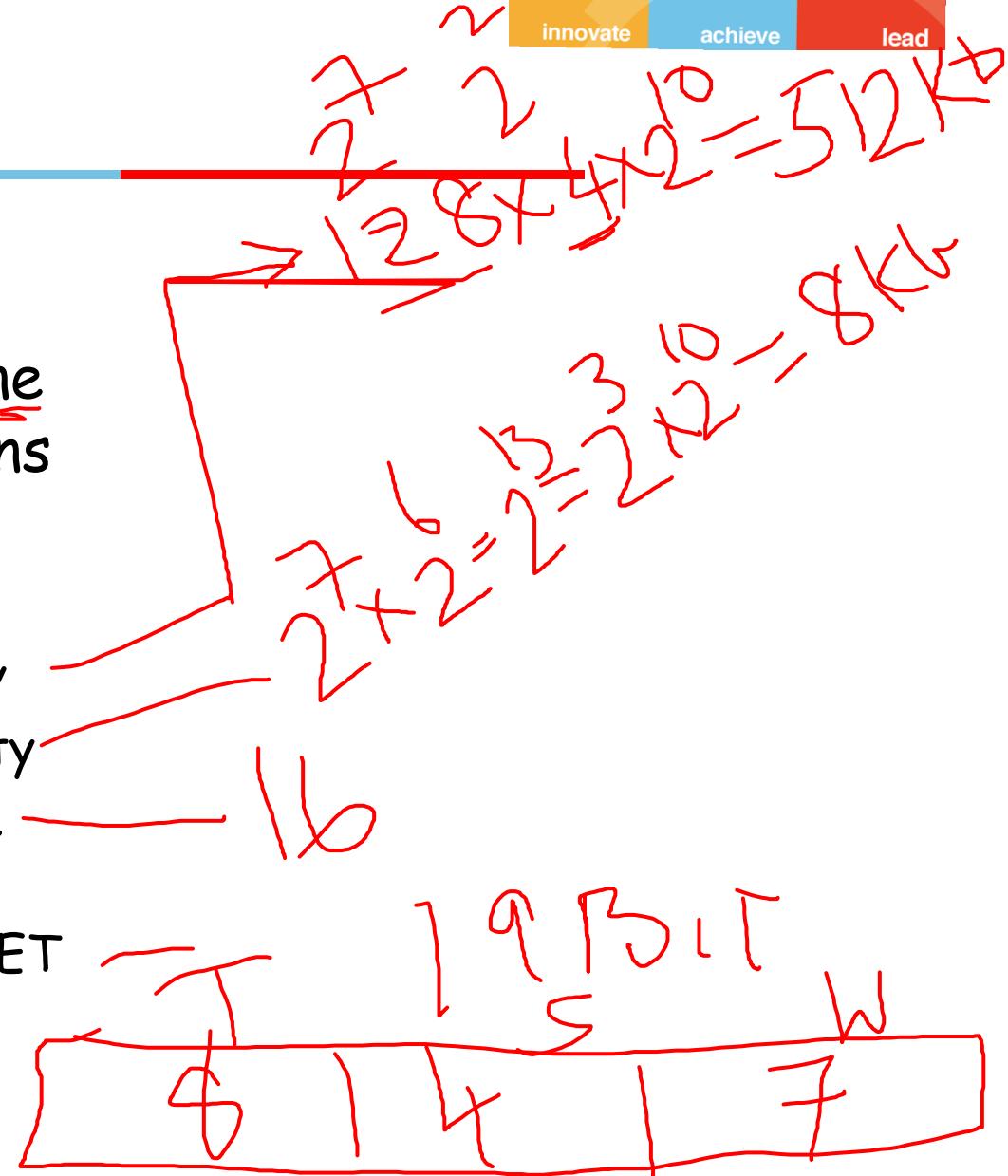
Size of tag = $(s - d)$ bits



Problem 1

A set-associative cache consists of 64 lines, or slots, divided into four-line sets. Main memory contains 4K blocks of 128 bytes each. Find out

- Total main memory capacity
- Total cache memory capacity
- Total number of sets in the cache
- Number of bits for TAG, SET and word

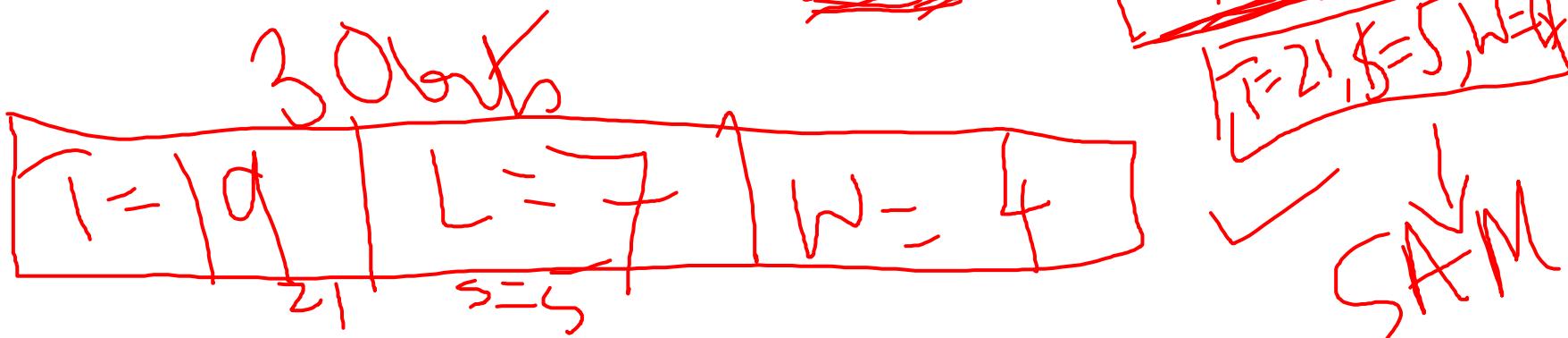


Home work

pls update
inno achieve lead

A computer has an 8 GByte memory with 64 bit word sizes. Each block of memory stores 16 words. The computer has a direct-mapped cache of 128 blocks. The computer uses word level addressing. What is the address format? If we change the cache to a 4-way set associative cache, what is the new address format?

$$\begin{aligned} \text{Addr} &= 33 \text{ bits (StW)} \\ &= 8 \cancel{\text{Gb}} = 1 \cancel{\text{Gb}} = 2^{30} \text{ W} \end{aligned}$$

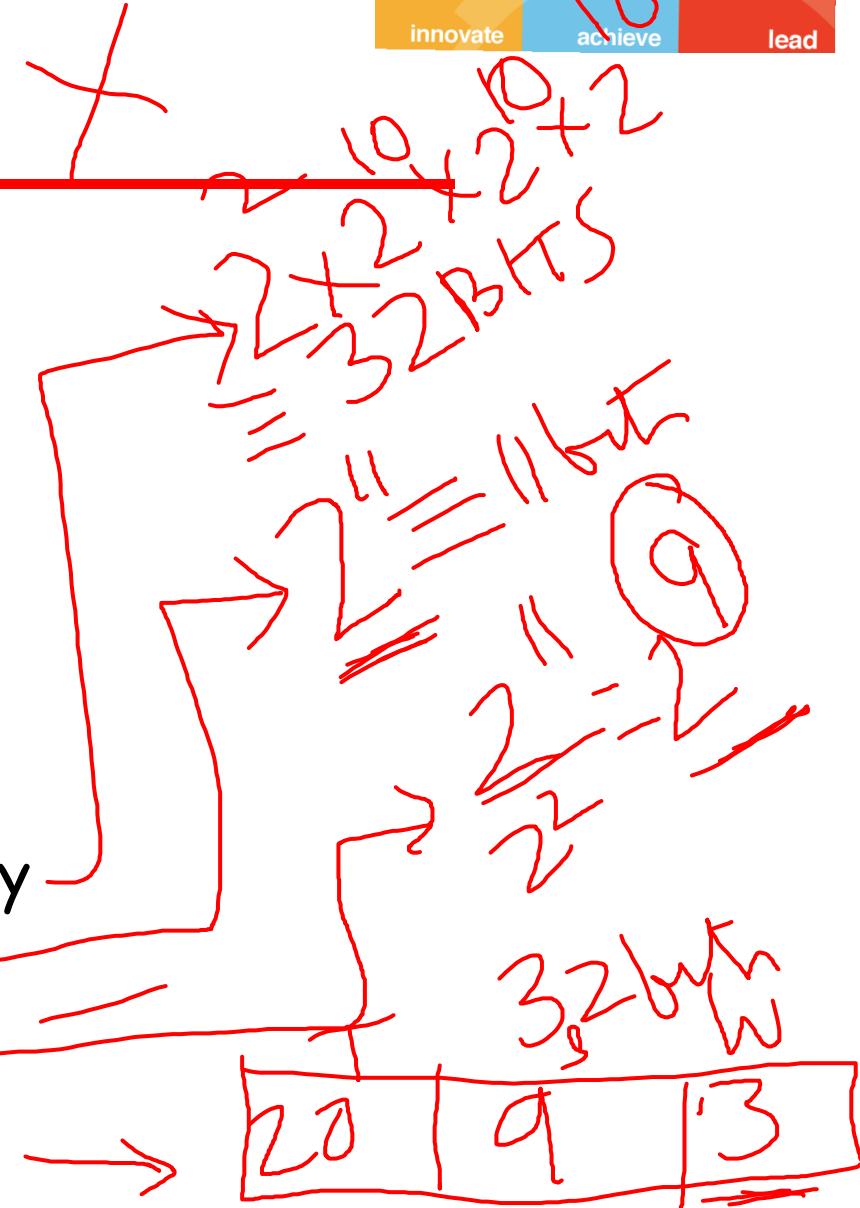


Problem 2 (1/2)

A 4-way set-associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. Find out address format.

Find out:

- 1) # bits to address main memory
- 2) # cache lines
- 3) # sets
- 4) Main Memory Address Format





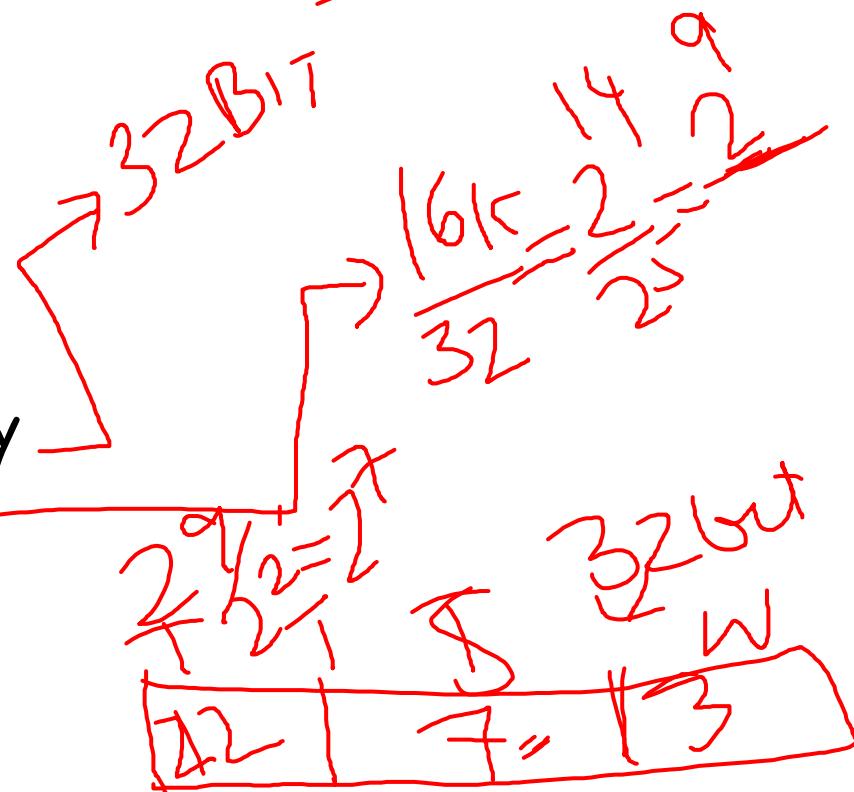
Problem 2 (2/2)

A 4-way set-associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. Find out address format.

Find out:

- 1) # bits to address main memory
- 2) # cache lines
- 3) # sets
- 4) Main Memory Address Format

$$\begin{aligned}
 \text{Block} &= 8 \text{ bytes} \\
 &= 8 + 4 \\
 &= 32 \text{ bytes} \\
 &= 32 \text{ bits}
 \end{aligned}$$





Home work

A computer has an 8 GByte memory with 64 bit word sizes. Each block of memory stores 16 words. The computer has a direct-mapped cache of 128 blocks. The computer uses **word level addressing**. What is the address format? If we change the cache to a 4-way set associative cache, what is the new address format?

Replacement Algorithms (1/3)



Direct mapped cache

- No choice
- Each block maps to one line and replace that line

Replacement Algorithms (2/3)

- Needed in Associative & Set Associative mapped cache
- Hardware implemented algorithm (speed)
- Methods:
 - Least Recently Used (LRU) ✓
 - Least Frequently Used (LFU) ✓
 - First In First Out (FIFO) ✓
 - Random ✓

Replacement Algorithms (3/3)

- **Least Recently used (LRU):** Replace the block that has been in the cache longest with no reference to it
 - e.g. 2 way set associative
 - Uses "USE" bits
 - Most effective method
- **Least frequently used:** Replace block which has had fewest hits
 - Uses counter with each line
- **First in first out (FIFO):** Replace block that has been in cache longest
 - Round robin or circular buffer technique
- **Random**

Problem 3

Consider a reference pattern that accesses the sequence of blocks 0, 4, 0, 2, 1, 8, 0, 1, 2, 3, 0, 4. Assuming that the cache uses associative mapping, find the hit ratio for a cache with four lines

- a) LRU
 - b) LFU
 - c) FIFO
-

Problem 2 - LRU

8
~~7~~
 5 | 12

Ref	0	4	0	2	1	8	0	1	2	3	0	4
time	0	1	2	3	4	5	6	7	8	9	10	11
L0	0	0	0	Q	0	0	0	0	0	0	0	0
L1	4	4	4	4	8	8	8	8	3	3	3	3
L2		2	2	2	2	2	2	2	2	2	2	2
L3			1		1	1	1	1	1	1	1	4
H/M	M	M	H	M	M	M	H	H	H	M	H	M

Problem 2 - LFU

5 | 12

|

✓

Ref	0	4	0	2	1	8	0	1	2	3	0	4
L0	0	0	0, 0,	0,	0,	0,	0, 0,	0, 0,	0, 0,	0, 0,	0, 0,	0, 0,
L1		4	4, 0	4, 0	4, 0	8, 0	8, 0	8, 0	8, 0	3, 0	3, 0	4, 0
L2			2, 0	2, 0	2, 0	2, 0	2, 0	2, 0	2, 1	2, 1	2, 1	2, 1
L3			1, 0	1, 0	1, 0	1, 0	1, 0	1, 0	1, 1	1, 1	1, 1	1, 1
H/M	M	M	H	M	M	H	H	H	H	M	H	M

M
F

H
F

M
F

F
H

M
F

Problem 2 - FIFO



Ref	0	4	0	2	1	8	0	1	2	3	0	4
time	0	1	2	3	4	5	6	7	8	9	10	11
L0	0	0	0	0	0	8	8	8	8	8	8	8
L1		4	4	4	4	4	0	0	0	0	0	0
L2			2	2	2	2	2	2	2	3	3	3
L3				1	1	1	1	1	1	1	1	4
H/M	F.	F.	F	F	F	F	F	F	F	F	F	F



BITS Pilani
Pilani Campus

COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS SESSION 5

Prepared By: Dr. Lucy J. Gudino
Instructor: <Prof. C R Sarma>

Replacement Algorithms (2/3)

- Needed in Associative & Set Associative mapped cache
- Hardware implemented algorithm (speed)
- Methods:
 - Least Recently Used (LRU)
 - Least Frequently Used (LFU)
 - First In First Out (FIFO)
 - Random

Replacement Algorithms (3/3)

- **Least Recently used (LRU):** Replace the block in the set that has been in the cache longest with no reference to it
 - e.g. 2 way set associative
 - Uses "USE" bits
 - Most effective method
- **Least frequently used:** Replace block which has had fewest hits
 - Uses counter with each line
- **First in first out (FIFO):** Replace block that has been in cache longest
 - Round robin or circular buffer technique
- **Random**

Problem 2

Consider a reference pattern that accesses the sequence of blocks 0, 4, 0, 2, 1, 8, 0, 1, 2, 3, 0, 4. Assuming that the cache uses associative mapping, find the hit ratio with a cache of four lines

- a) LRU ✓
 - b) LFU
 - c) FIFO
-

Problem 2 - LRU (Completed)

Ref B11C	0	4	0	2	1	<u>8</u>	0	1	2	3	0	4
time	0	1	2	3	4	5	6	7	8	9	10	11
L0	0 ₀	0 ₀	0 ₂	0 ₂	0 ₂	0 ₂	0 ₆	0 ₆	0 ₆	0 ₆	0 ₁₀	0 ₁₀
L1		4 ₁	4 ₁	4 ₁	4 ₁	8 ₅	8 ₅	8 ₅	8 ₅	3 ₉	3 ₉	3 ₉
L2			2 ₃	2 ₈	2 ₈	2 ₈	2 ₈					
L3				1 ₄	1 ₄	1 ₄	1 ₇	4 ₁₁				
H/M	M	M	H	M	M	M	H	H	H	M	H	M

5 / 12

Problem 2 - LFU

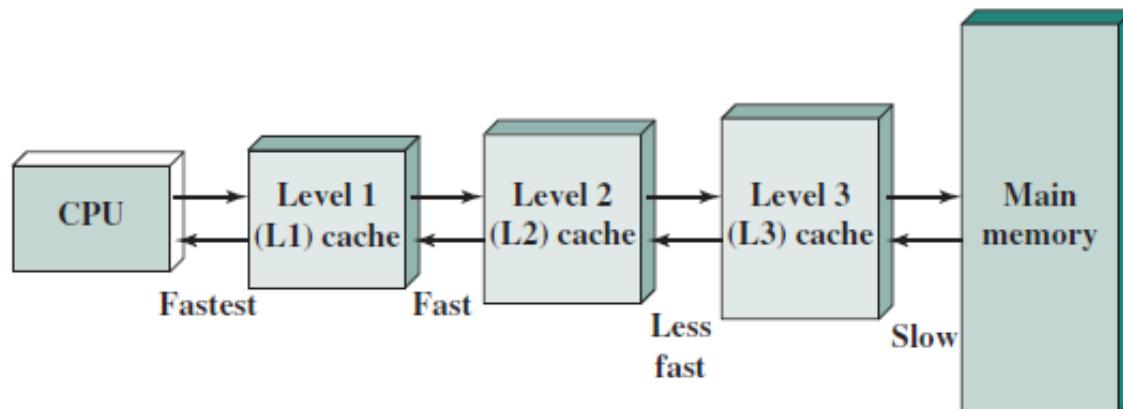
Ref	0	4	0	2	1	8	0	1	2	3	0	4
L0												
L1												
L2												
L3												
H/M												

Problem 2 - FIFO

Ref	0	4	0	2	1	8	0	1	2	3	0	4
time	0	1	2	3	4	5	6	7	8	9	10	11
L0												
L1												
L2												
L3												
H/M												

Issues with Writes

- Multiple copies of data exist:
 - L1, L2, L3, Main Memory, Disk
- What to do on a write-hit?
 - **Write-through** (write immediately to memory)
 - **Write-back** (defer write to memory until replacement of line)
 - Need a dirty bit (line different from memory or not)

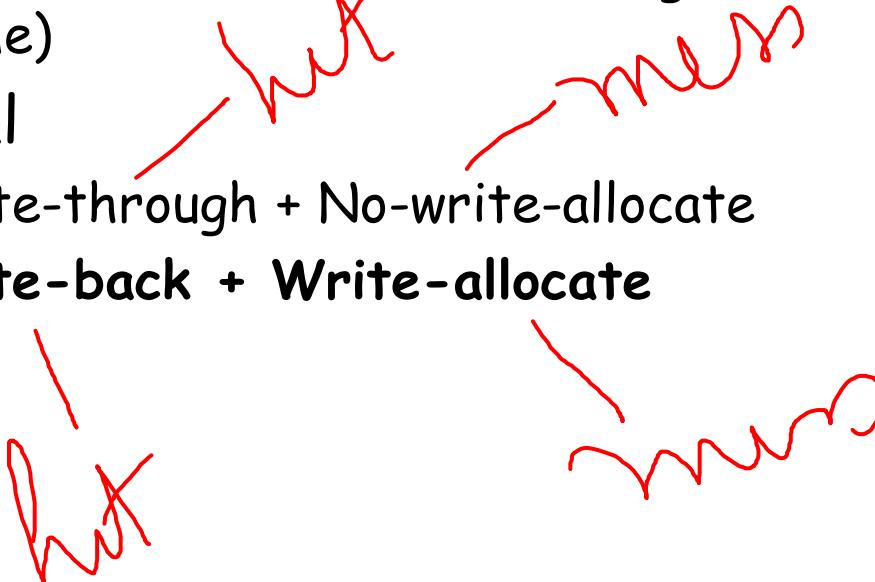


(b) Three-level cache organization

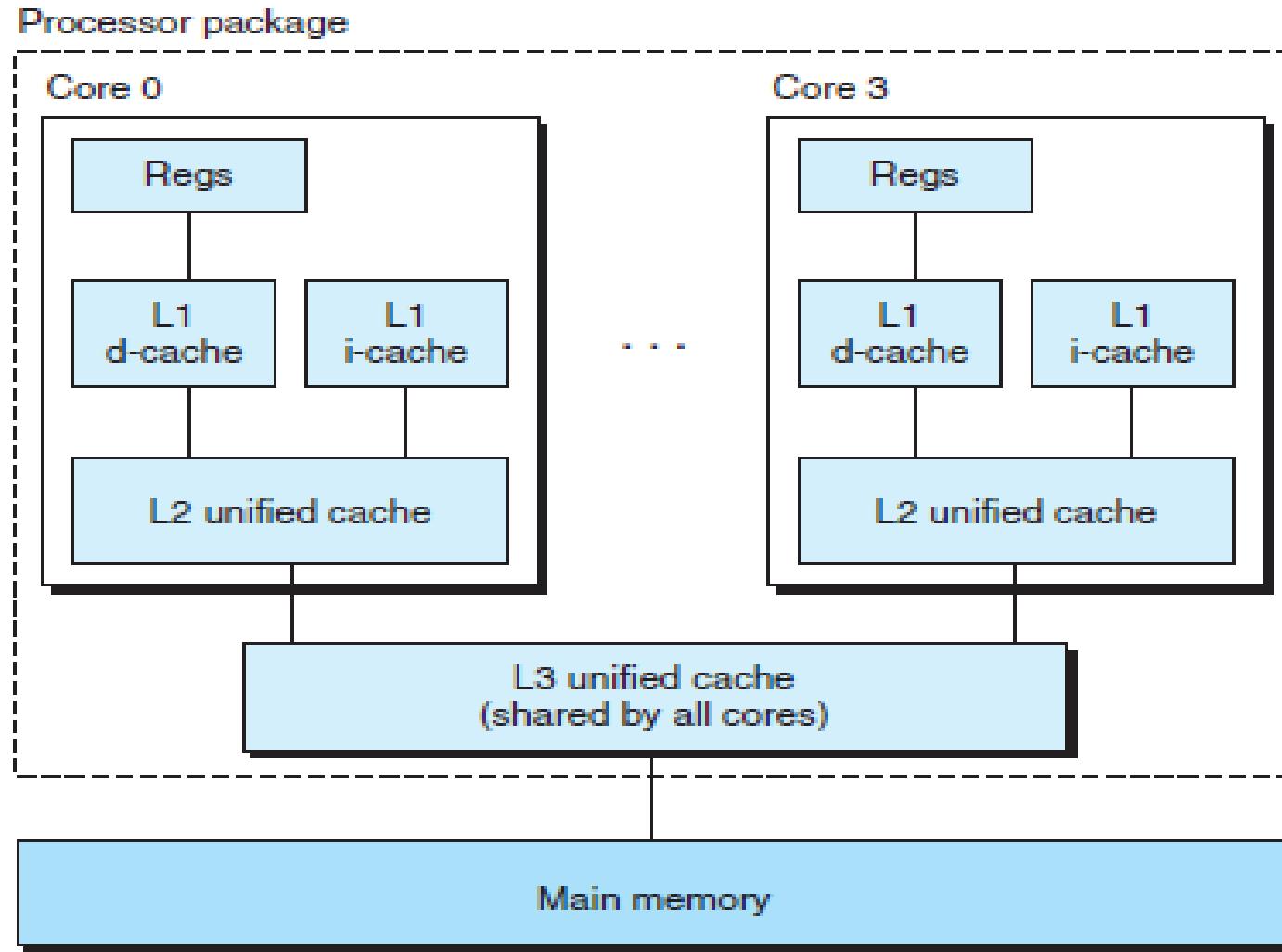
Cache and Main Memory

Issues with Writes

- What to do on a write-miss?
 - Write-allocate (load into cache, update line in cache)
 - Good if more writes to the location follow
 - No-write-allocate (writes straight to memory, does not load into cache)
- Typical
 - Write-through + No-write-allocate
 - Write-back + Write-allocate



Intel Core i7 Cache Hierarchy



Intel Core i7 Cache Hierarchy



Cache type	Access time (cycles)	Cache size (C)	Assoc. (E)	Block size (B)	Sets (S)
L1 i-cache	4	32 KB	8	64 B	64
L1 d-cache	4	32 KB	8	64 B	64
L2 unified cache	11	256 KB	8	64 B	512
L3 unified cache	30–40	8 MB	16	64 B	8192

Characteristics of the Intel Core i7 cache hierarchy.

Performance Impact of Cache Parameters

- Associativity :
 - higher associativity → more complex hardware
 - Higher Associativity → Lower miss rate
 - Higher Associativity → reduces average memory access time (AMAT)
- Cache Size
 - Larger the cache size → Lower miss rate
 - Larger the cache size → reduces average memory access time (AMAT)
- Block Size:
 - Smaller blocks do not take maximum advantage of spatial locality.

Revisiting Locality of reference

```
1 int sumvec(int v[N])
2 {
3     int i, sum = 0;
4
5     for (i = 0; i < N; i++)
6         sum += v[i];
7
8 }
```

Does this function have good locality?

Sum=Sum+ $v[i]$

N=8								
Address	0	1	2	3	4	5	6	7
Contents	V[0]	V[1]	V[2]	V[3]	V[4]	V[5]	V[6]	V[7]
Access Order	1	2	3	4	5	6	7	8

Stride k – reference pattern

Byte Addressable
memory and word
length is 1 byte

innovate

achieve

lead

i	Address
0	0000
1	0001
2	0002
3	0003
4	0004
5	0005
6	0006
7	0007
8	0008
9	0009
10	000A
11	000B
12	000C

Stride 1

Address difference
Stride = -----
Word Length

i	Address
0	0000
1	0001
2	0002
3	0003
4	0004
5	0005
6	0006
7	0007
8	0008
9	0009
10	000A
11	000B
12	000C

Stride 2

Stride k - reference pattern

Byte Addressable
memory and word
length is 2 bytes

i	Address
0	0000
1	0002
2	0004
3	0006
4	0008
5	000A
6	000C
7	000E
8	0010
9	0012
10	0014
11	0016
12	0018

Stride 1

i	Address
0	0000
1	0002
2	0004
3	0006
4	0008
5	000A
6	000C
7	000E
8	0010
9	0012
10	0014
11	0016
12	0018

Stride 2

Address difference
Stride = -----
Word Length

Revisiting Locality of reference

```

1 int sumarrayrows(int a[M][N])
2 {
3     int i, j, sum = 0;
4
5     for (i = 0; i < M; i++)
6         for (j = 0; j < N; j++)
7             sum += a[i][j];
8
9     return sum;
}
  
```

Does this function have good locality?

MxN	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6

M = 2, N=3

Address(Index)	0	1	2	3	4	5
Contents	A[0][0]	A[0][1]	A[0][2]	A[1][0]	A[1][1]	A[1][2]
Access Order	1	2	3	4	5	6

Revisiting Locality of reference

```

1 int sumarraycols(int a[M] [N])
2 {
3     int i, j, sum = 0;
4
5     for (j = 0; j < N; j++)
6         for (i = 0; i < M; i++)
7             sum += a[i] [j];
8
9 }
```

Does this function have good locality?

M = 2, N=3							NxM	[0]	[1]
Address(Index)	0	1	2	3	4	5	[0]	1	2
Contents	A[0][0]	A[1][0]	A[2][0]	A[1][0]	A[1][1]	A[2][1]	[1]	3	4
Access Order	1	3	5	2	4	6	[2]	5	6

Writing Cache Friendly Code

- Make the common case go fast
 - Focus on the inner loops of the core functions
- Minimize the misses in the inner loops
 - Repeated references to variables are good (**temporal locality**)
 - Stride-1 reference patterns are good (**spatial locality**)

Example 1

```
int sumarrayrows(int a[4][4])
{
    int i, j, sum = 0;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            sum += a[i][j];
    return sum;
}
```

Assumption:

- The cache has a block size of 4 words each, 2 cache lines
- Word size 4 bytes.
- C stores arrays in row-major order

Example 1(Contd..)

```
int sumarrayrows(int a[4][4])
{
    int i, j, sum = 0;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            sum += a[i][j];
    return sum;
}
```

[0] L |

+	9
2	0
3	1
4	2
5	
6	
7	
8	

A[i][j]	J = 0	J = 1	J = 2	J = 3
i = 0	1M	2H	2H	4H
i = 1	5M	6H	7H	8H
i = 2	9M	10H	11H	12H
i = 3	13M			

4M
12H

✓ A(0,0) = 1
✓ A(0,1) = 2
- A(0,2) = 3
A(0,3) = 4
A(1,0) = 5
A(1,1) = 6
A(1,2) = 7
A(1,3) = 8
A(2,0) = 9
A(2,1) = 10
A(2,2) = 11
A(2,3) = 12
A(3,0) = 13
A(3,1) = 14
A(3,2) = 15
A(3,3) = 16

Example 2:

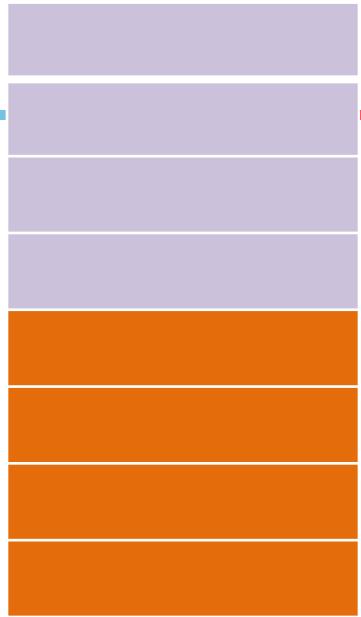
```
int sum_array(int a[4][4])
{
    int i, j, sum = 0;
    for (j = 0; j < 4; j++)
        for (i = 0; i < 4; i++)
            sum += a[i][j];
    return sum;
}
```

Assumption:

- The cache has a block size of 4 words each, 2 cache lines
- Word size 4 bytes.
- ²¹ C stores arrays in row-major order

Example 1(Contd..)

```
int sum_array(int a[4][4])
{
    int i, j, sum = 0;
    for (j = 0; j < 4; j++)
        for (i = 0; i < 4; i++)
            sum += a[i][j];
    return sum;
}
```



A[i][j]	J = 0	J = 1	J = 2	J = 3
i = 0				
i = 1				
i = 2				
i = 3				

A(0,0) = 1
A(0,1) = 2
A(0,2) = 3
A(0,3) = 4
A(1,0) = 5
A(1,1) = 6
A(1,2) = 7
A(1,3) = 8
A(2,0) = 9
A(2,1) = 10
A(2,2) = 11
A(2,3) = 12
A(3,0) = 13
A(3,1) = 14
A(3,2) = 15
A(3,3) = 16

Home Work - Which one is better ?



Program 1:

```
for (int i = 0; i < n; i++) {  
    z[i] = x[i] - y[i];  
    z[i] = z[i] * z[i];  
}
```

Program 2:

```
for (int i = 0; i < n; i++) {  
    z[i] = x[i] - y[i];  
}  
for (int i = 0; i < n; i++) {  
    z[i] = z[i] * z[i];  
}
```



CISC Instruction Set (Intel x86 as an example)

BITS Pilani
Pilani Campus

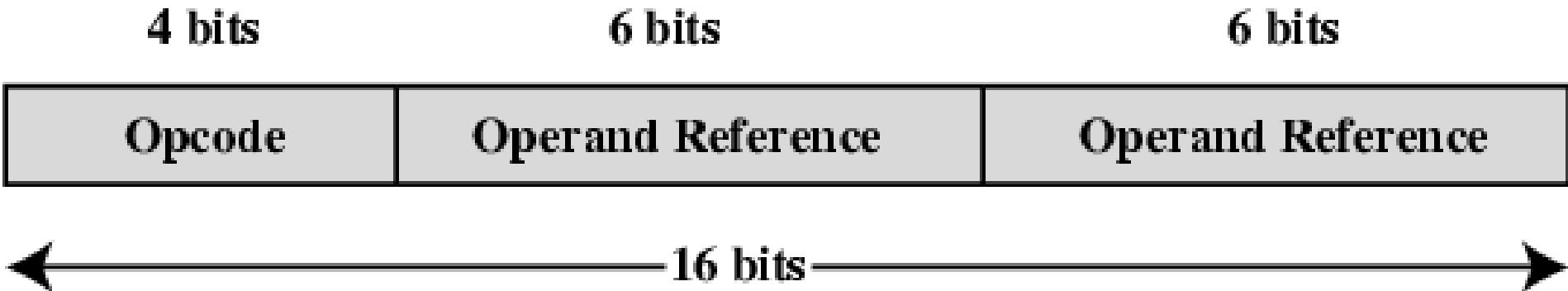
Today's Session

Contact Hour	List of Topic Title	Text/Ref Book/external resource
9-10	<ul style="list-style-type: none"> • Instruction Set Architecture - CISC Vs RISC • CISC Instruction Set (Intel x86 as an example) <ul style="list-style-type: none"> • Machine Instruction Characteristics • Types of Operands • Types of Operations • Addressing Modes • Instruction Formats 	T1

Introduction

- What is an Instruction Set?
 - The complete collection of instructions that are understood by a CPU
- Elements of an Instruction
 - Operation code (Op code)
 - Source Operand reference
 - Result Operand reference
 - Next Instruction Reference
- Source and Destination Operands can be found in four areas
 - Main memory (or virtual memory or cache)
 - CPU register
 - I/O device
 - Immediate

Simple Instruction Format



- During instruction execution, an instruction is read into an instruction register (IR) in the processor.
- The processor must be able to extract the data from the various instruction fields to perform the required operation.
- Opcodes are represented by abbreviations, called **mnemonics**

Example: ADD AX, BX → Add instruction

Instruction Types

- Data processing : Arithmetic and logic instructions
- Data storage (main memory) : Movement of data into or out of register and or memory locations
- Data movement (I/O) : I/O instructions
- Program flow control : Test and branch instructions

Number of Addresses (1/2)

- 3 addresses
 - Result, Operand 1, Operand 2
 - $c = a + b$; add c, a, b
 - May be a forth - next instruction (usually implicit)
 - Needs very long words to hold everything
- 2 addresses
 - One address doubles as operand and result
 - $a = a + b$: add a, b
 - Reduces length of instruction
 - The original value of a is lost.

Number of Addresses (2/2)

- 1 address
 - Implicit second address
 - Usually a register (accumulator)
 - Common on early machines

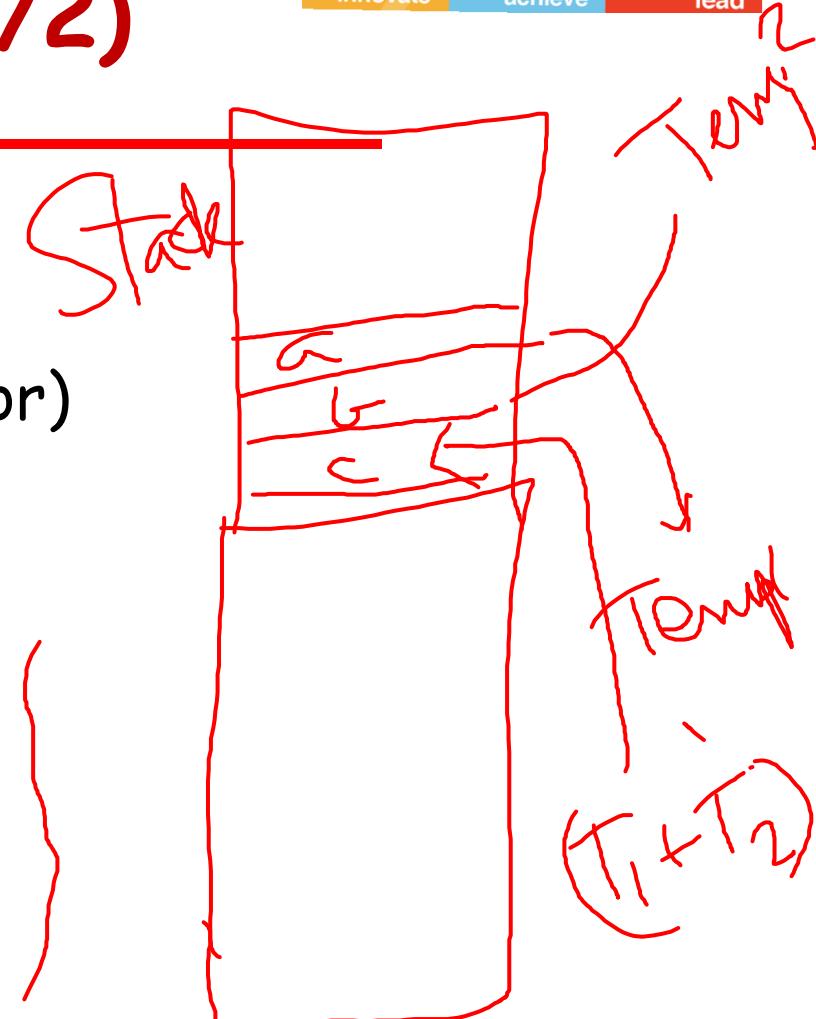
- 0 (zero) addresses
 - All addresses implicit
 - Uses a stack
 - e.g. $c = a + b$

push a

push b –

add –

pop c



Example

$$\text{Execute } Y = \frac{A - B}{C + (D \times E)}$$

(Red annotations: A and B are crossed out, C + (D × E) is underlined)

Instruction	Comment
SUB Y, A, B	$Y \leftarrow A - B$
MPY T, D, E	$T \leftarrow D \times E$
ADD T, T, C	$T \leftarrow T + C$
DIV Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions

Instruction	Comment
LOAD D	$AC \leftarrow D$
MPY E	$AC \leftarrow AC \times E$
ADD C	$AC \leftarrow AC + C$
STOR Y	$Y \leftarrow AC$
LOAD A	$AC \leftarrow A$
SUB B	$AC \leftarrow AC - B$
DIV Y	$AC \leftarrow AC \div Y$
STOR Y	$Y \leftarrow AC$

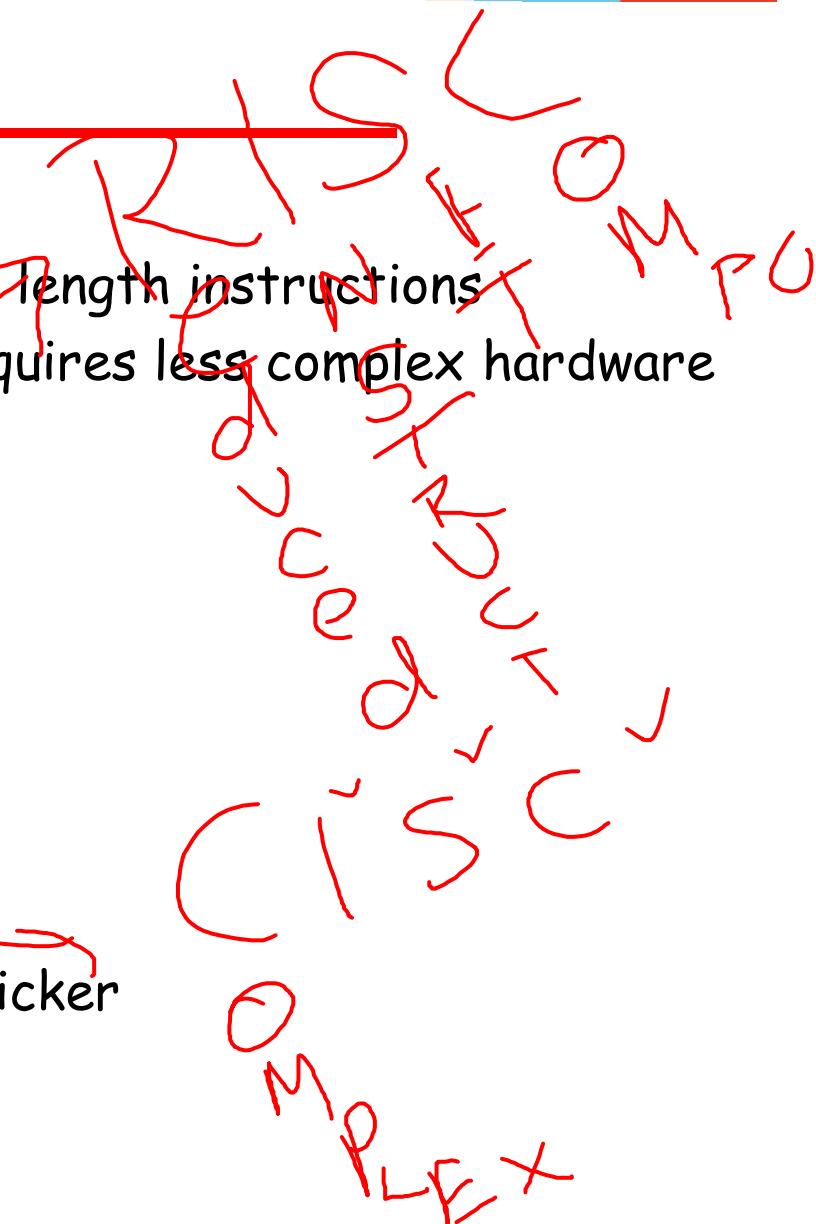
(c) One-address instructions

Instruction	Comment
MOVE Y, A	$Y \leftarrow A$
SUB Y, B	$Y \leftarrow Y - B$
MOVE T, D	$T \leftarrow D$
MPY T, E	$T \leftarrow T \times E$
ADD T, C	$T \leftarrow T + C$
DIV Y, T	$Y \leftarrow Y \div T$

(b) Two-address instructions

How Many Addresses

- Fewer addresses
 - More Primitive instructions, shorter length instructions
 - Less complex instructions, hence requires less complex hardware
 - More instructions per program
 - Longer programs
 - More complex programs
 - Longer execution time
- Multiple address instructions
 - Lengthy instructions
 - More registers
 - Inter-register operations are quicker
 - Fewer instructions per program

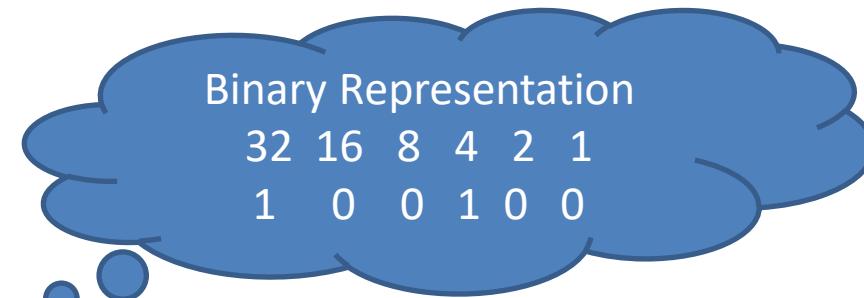


Instruction set Design Decisions

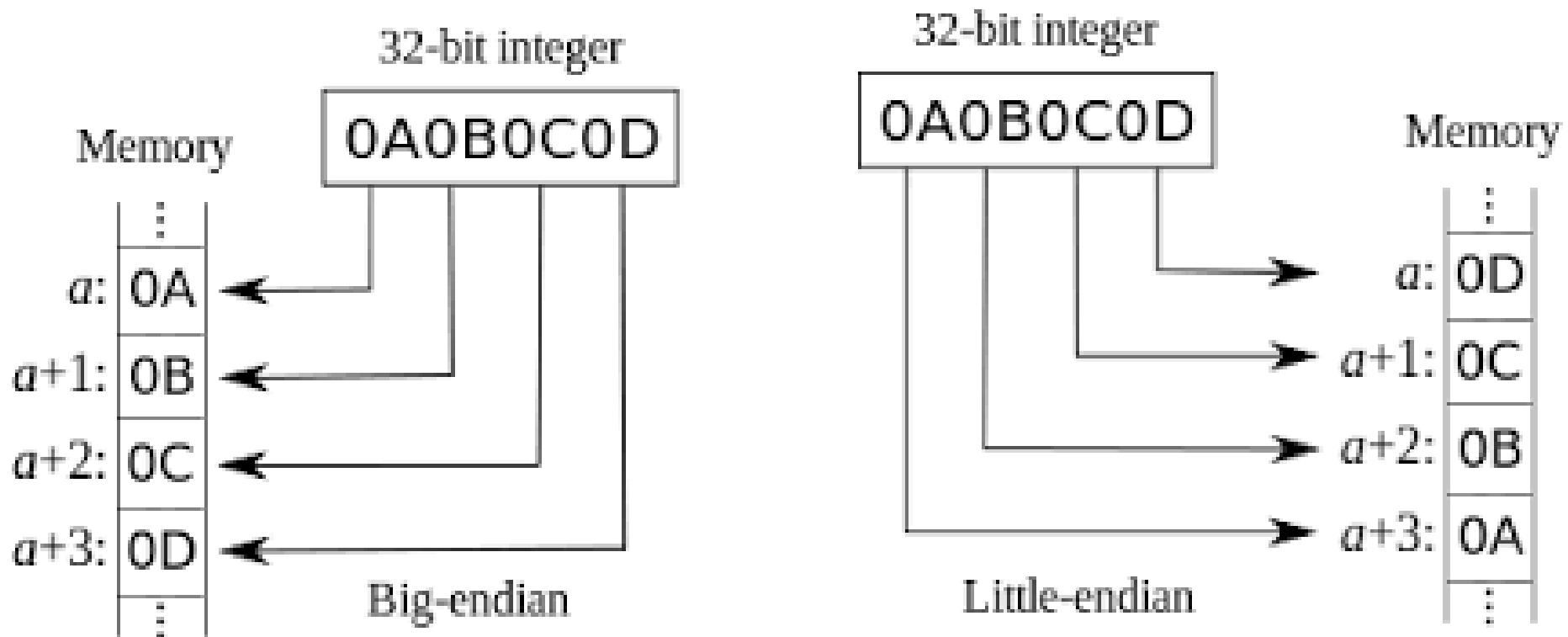
- Operation repertoire
 - How many ops?
 - What can they do?
 - How complex are they?
- Data types
- Instruction formats
 - Length of op code field
 - Number of addresses
- Registers
 - Number of CPU registers available
 - Which operations can be performed on which registers?
- Addressing modes

Types of Operand

- Machine instructions operate on data
- General categories of data
 - Addresses
 - Numbers
 - Binary integer or binary fixed point, floating point, decimal
 - Characters
 - ASCII etc.
 - Logical Data
 - Bits or flags
- Packed Decimal
 - 36 : 0011 0110



Byte Ordering

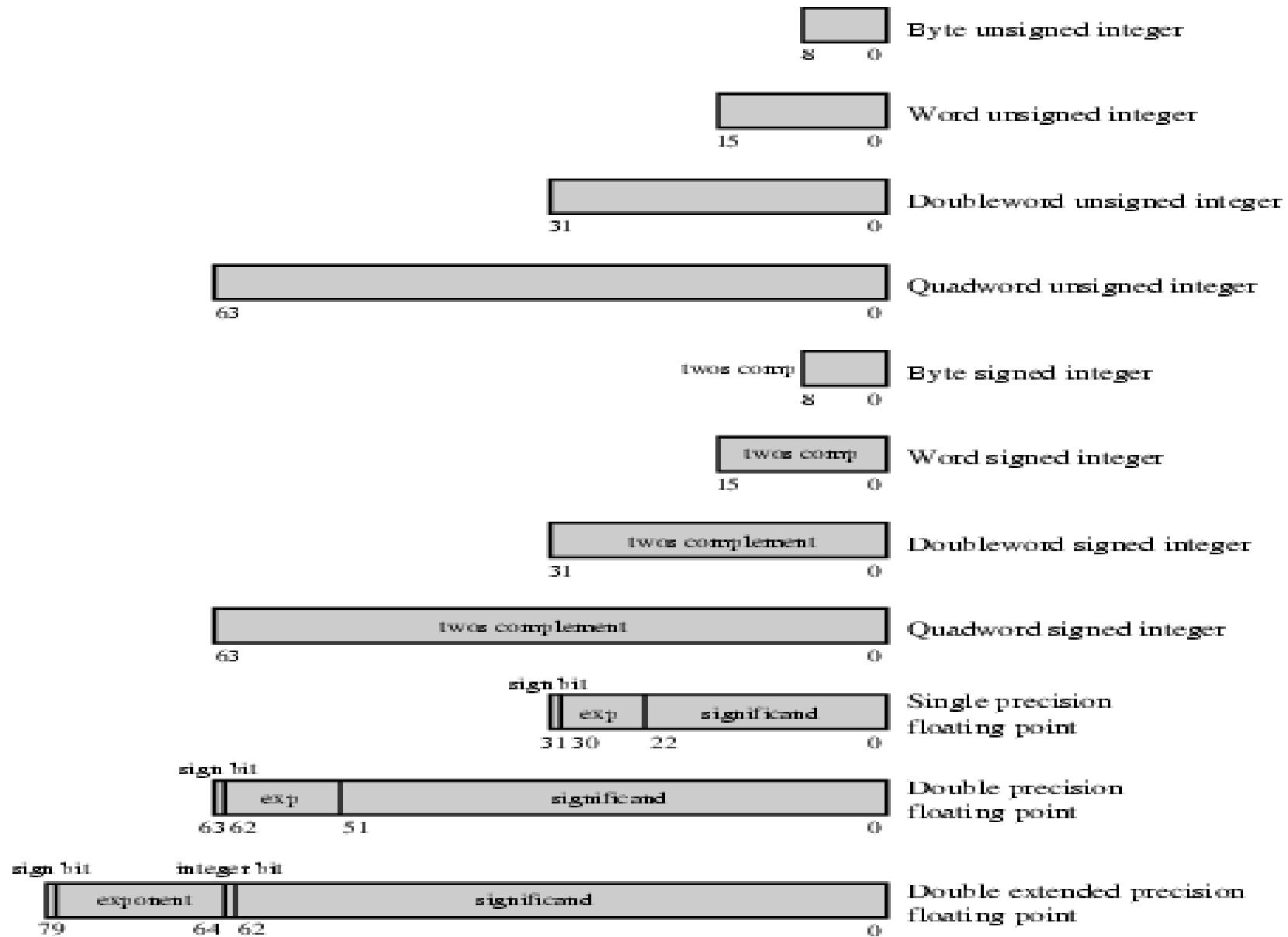


x86 Data Types

- General - Byte, Word, double word, quadword, double quad word - arbitrary binary contents
 - Integer - signed binary using two's complement representation
 - Ordinal - unsigned integer
 - Unpacked BCD - One digit per byte
 - Packed BCD - 2 digits per byte
 - Near Pointer - 16/32 bit offset within segment
 - Far pointer - 16/32 bit offset outside segment
 - Bit field : A contiguous sequence of bits in which the position of each bit is considered as an independent unit.
 - Bit and Byte String
 - Floating Point
-

Data Type	Description
General	Byte, word (16 bits), doubleword (32 bits), quadword (64 bits), and double quadword (128 bits) locations with arbitrary binary contents.
Integer	A signed binary value contained in a byte, word, or doubleword, using two's complement representation.
Ordinal	An unsigned integer contained in a byte, word, or doubleword.
Unpacked binary coded decimal (BCD)	A representation of a BCD digit in the range 0 through 9, with one digit in each byte.
Packed BCD	Packed byte representation of two BCD digits; value in the range 0 to 99.
Near pointer	A 16-bit, 32-bit, or 64-bit effective address that represents the offset within a segment. Used for all pointers in a nonsegmented memory and for references within a segment in a segmented memory.
Far pointer	A logical address consisting of a 16-bit segment selector and an offset of 16, 32, or 64 bits. Far pointers are used for memory references in a segmented memory model where the identity of a segment being accessed must be specified explicitly.
Bit field	A contiguous sequence of bits in which the position of each bit is considered as an independent unit. A bit string can begin at any bit position of any byte and can contain up to 32 bits.
Bit string	A contiguous sequence of bits, containing from zero to $2^{32} - 1$ bits.
Byte string	A contiguous sequence of bytes, words, or doublewords, containing from zero to $2^{32} - 1$ bytes.
Floating point	See Figure 10.4.
Packed SIMD (single instruction, multiple data)	Packed 64-bit and 128-bit data types

x86 Numeric Data Formats





End of Session 5

Types of Operation

- Data Transfer
- Arithmetic
- Logical
- Conversion
- I/O
- System Control
- Transfer of Control

Data Transfer

- Specify
 - Source
 - Destination
 - Amount of data
- Action:
 1. Calculate the memory address, based on the address mode
 2. If the address refers to virtual memory, translate from virtual to real memory address.
 3. Determine whether the addressed item is in cache.
 4. If not, issue a command to the memory module.

Arithmetic

- Add, Subtract, Multiply, Divide
- May include
 - Absolute value ($|a|$)
 - Increment ($a++$)
 - Decrement ($a--$)
 - Negate ($-a$)
- Signed Integer

Logical

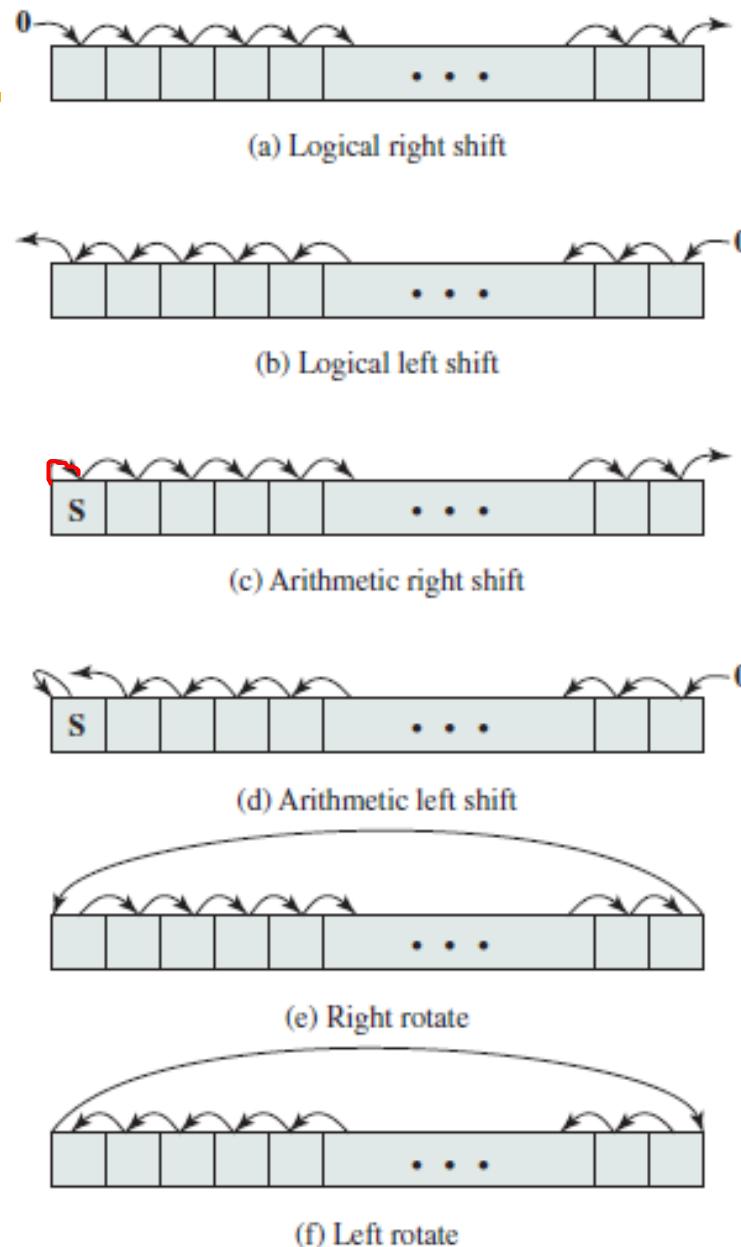
- Bitwise operations
- AND, OR, NOT

INNOVATE
ACHIEVE
LEAD

Basic Logical Operations

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P = Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

Shift and Rotate Operations



Input	Operation	Output
10101101	Logical right shift (3 bits)	10101101=> 00010101
10101101	Logical left shift (3 bits)	10101101=> 01101000
10101101	Arithmetic right shift (3 bits)	10101101=> 11110101
10101101	Arithmetic left shift (3 bits)	10101101=> 11101000
10101101	Right rotate (3 bits)	10101101=> 10110101
10101101	Left rotate (3 bits)	10101101=> 01101101

Conversion

E.g. Binary to Decimal

Input/Output

- May be specific instructions (I/O-Mapped I/O)
- May be done using data movement instructions (memory mapped)
- May be done by a separate controller (DMA)

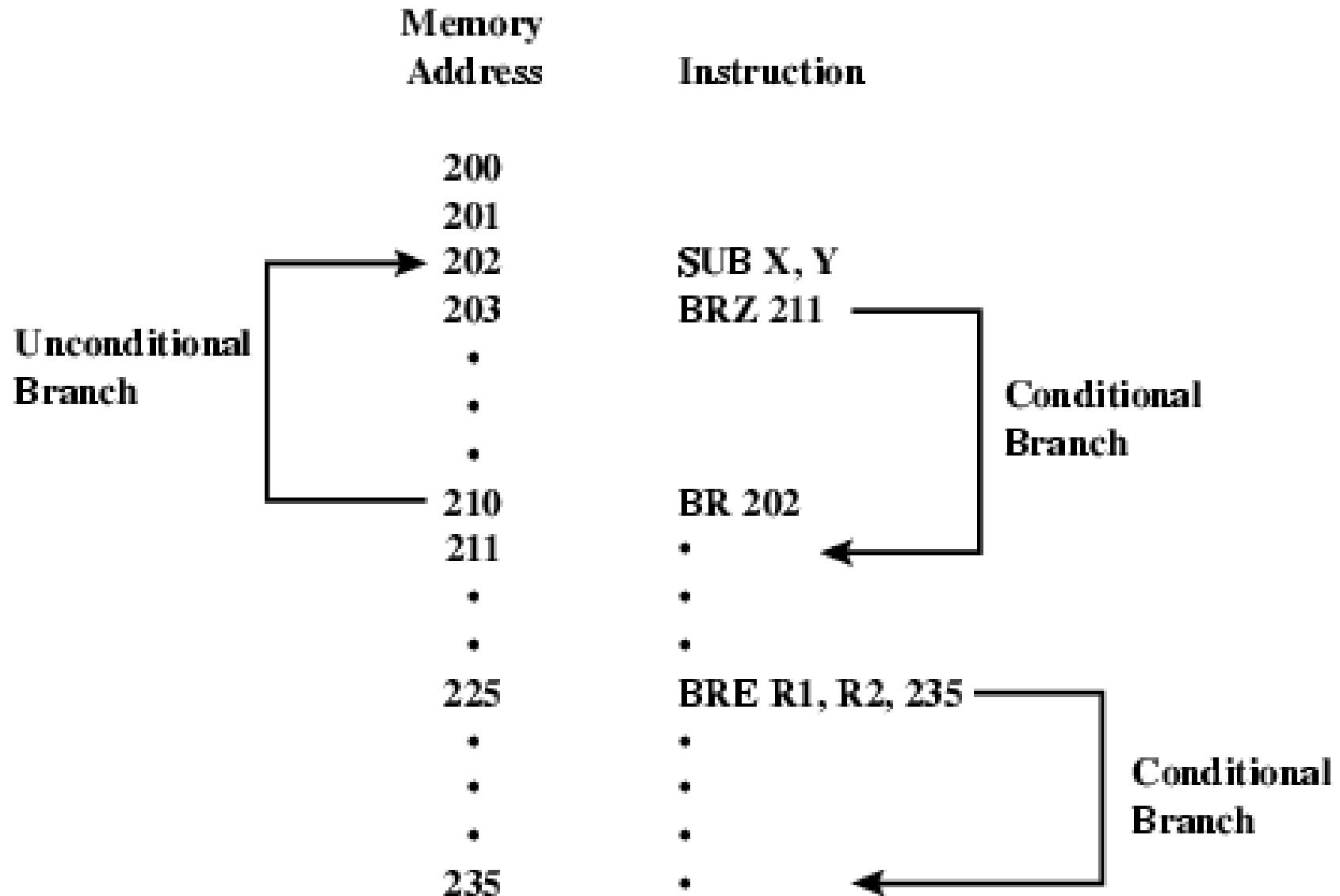
Systems Control

- Privileged instructions
- CPU needs to be in specific state
 - User Mode
 - Kernel mode
- For operating systems use

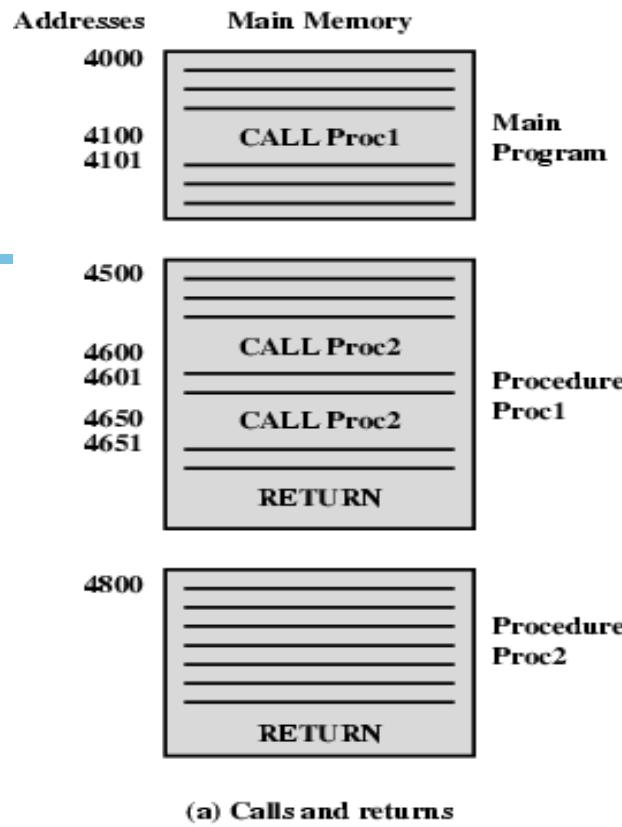
Transfer of Control

- Jump / Branch (Unconditional / Conditional)
 - e.g. jump to x if result is zero
 - Skip (Unconditional / Conditional)
 - skip (unconditional) : Increment to skip next instruction
 - e.g. increment and skip if zero
 - ISZ Register1
Branch xxxx
ADD A
 - Subroutine call
 - interrupt call
-

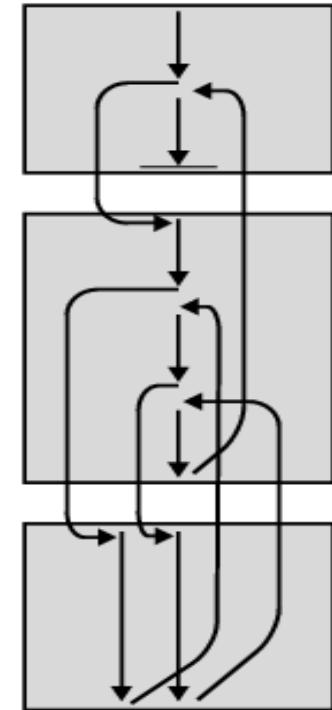
Branch / Jump Instruction



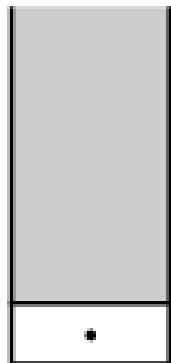
Use of Stack



(a) Calls and returns



(b) Execution sequence



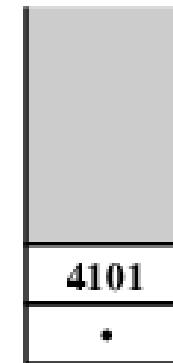
(a) Initial stack contents



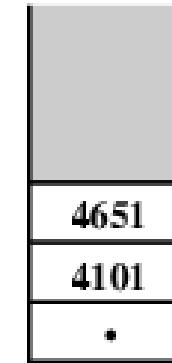
(b) After
CALL Proc1



(c) Initial
CALL Proc2



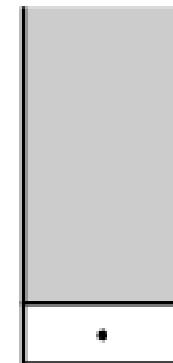
(d) After
RETURN



(e) After
CALL Proc



(f) After
RETURN



(g) After
RETURN

Addressing Modes

- Addressing modes refers to the way in which the operand of an instruction is specified
- Types:
 - Immediate
 - Direct
 - Indirect
 - Register
 - Register Indirect
 - Displacement (Indexed)
 - Stack

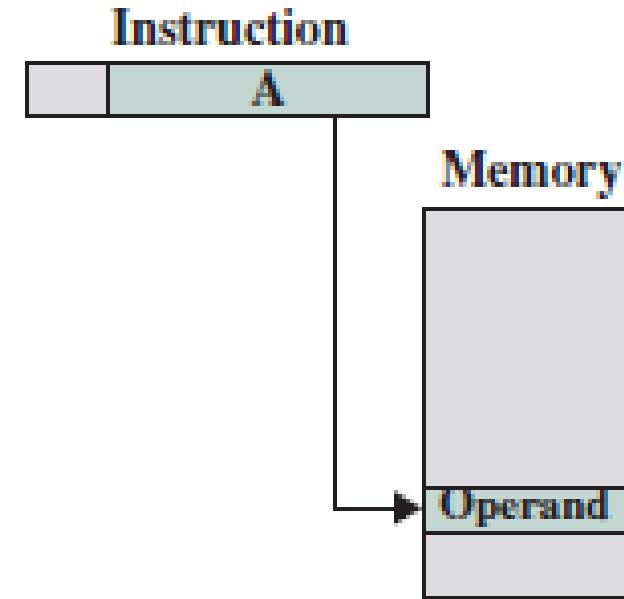
Immediate Addressing

- Operand is specified in the instruction itself
- e.g. ADD #5
 - Add 5 to contents of accumulator
 - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range



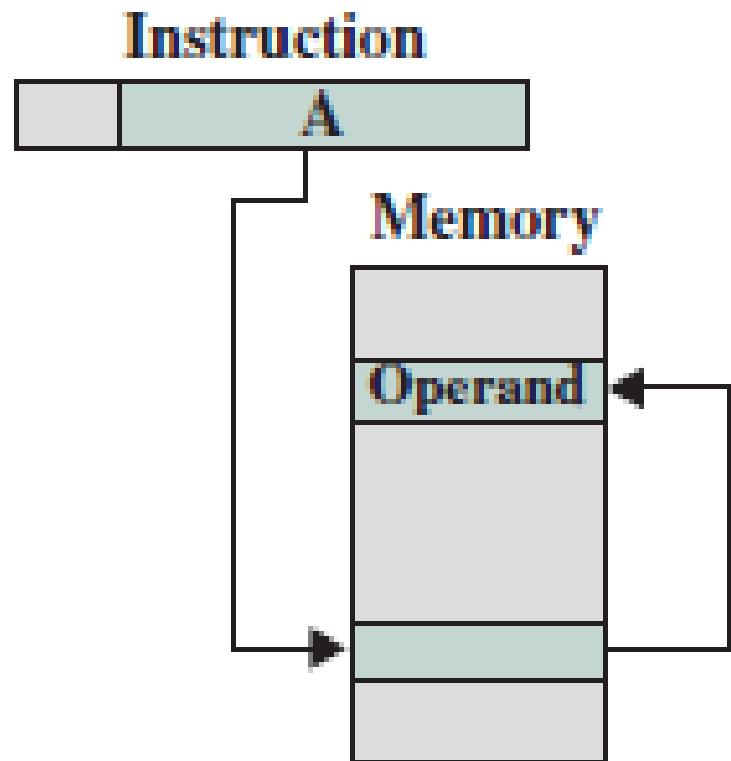
Direct Addressing

- Address of the operand is specified in the instruction
- Effective address (EA) = address field (A)
- e.g. ADD A
 - Add contents of memory cell whose address is A to accumulator
 - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space



Indirect Addressing

- Memory cell pointed to by address field of the instruction contains the address of (pointer to) the operand
- $EA = (A)$
 - Look in A , find address and look there for operand
- e.g. ADD (A)
 - Add contents of cell pointed to by contents of A to accumulator

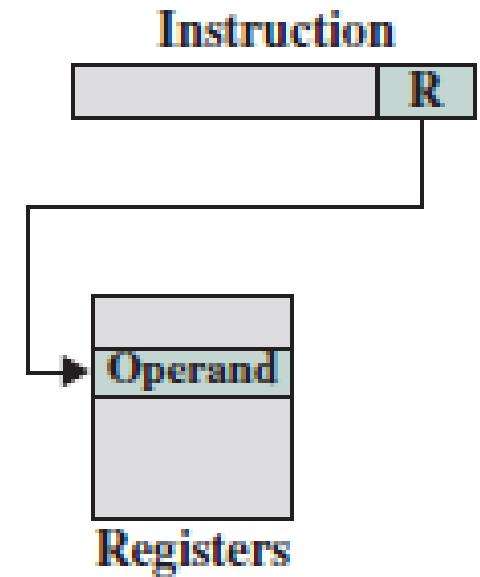


Indirect Addressing...

- Large address space
- 2^n where n = word length
- May be nested, multilevel, cascaded
 - e.g. EA = ((A))
- Multiple memory accesses to find operand
- Slower

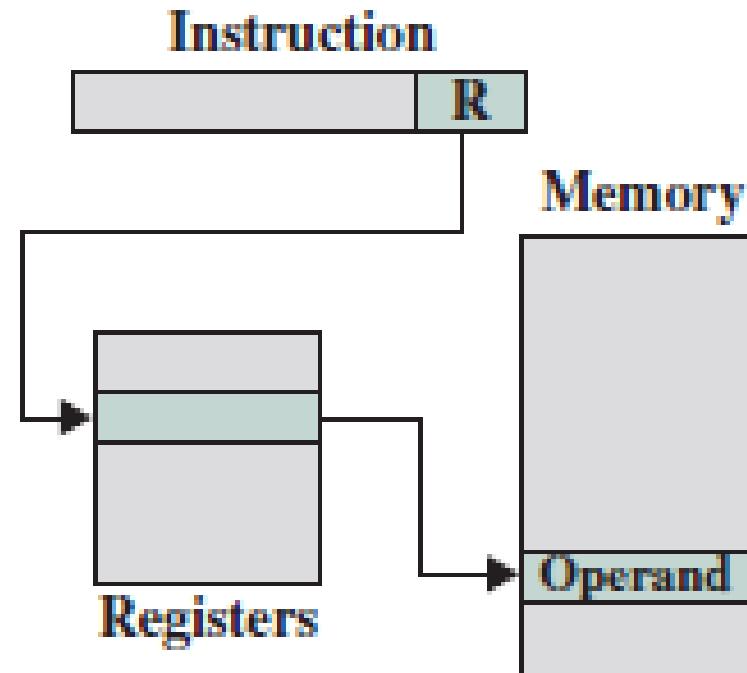
Register Addressing

- Operand is held in register named in address field
- $EA = R$
- Limited number of registers
- Very small address field needed
 - Shorter instructions
 - Faster instruction fetch
- No memory access hence Very fast execution but very limited address space
- Multiple registers helps in improving performance
 - Requires good assembly programming or compiler writing
 - C programming : `register int a;`



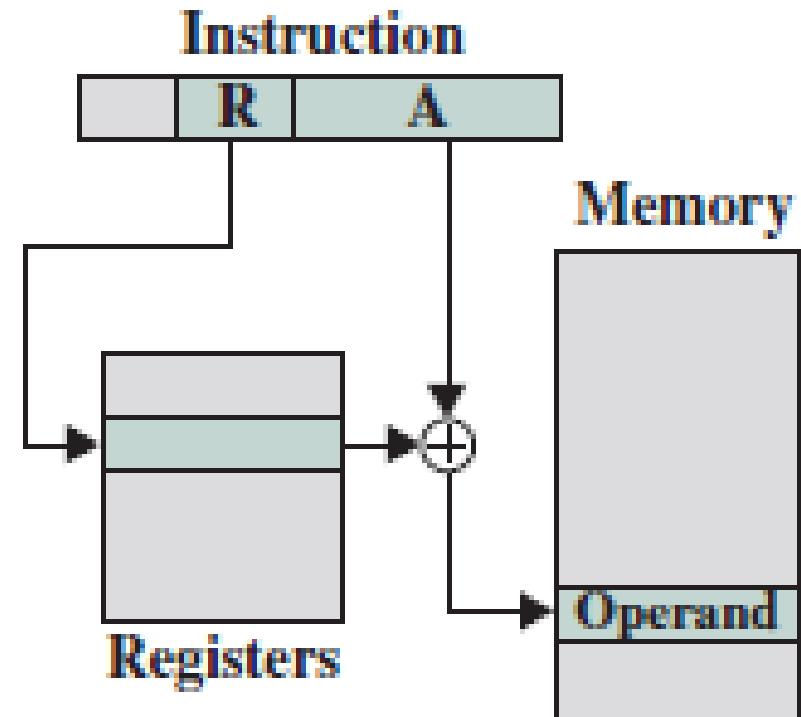
Register Indirect Addressing

- Similar to indirect addressing
- $EA = (R)$
- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One memory access compared indirect addressing



Displacement Addressing

- $EA = A + (R)$
- Address field hold two values
 - A = base value
 - R = register that holds displacement
 - or vice versa
- Three variants:
 - Relative addressing
 - Base register addressing
 - Indexing



Relative Addressing

- Also known as PC relative addressing
- A version of displacement addressing
- R = Program counter, PC
- $EA = A + (PC)$
- Relative addressing exploits the concept of locality
 - If most memory references are relatively near to the instruction being executed, then the use of relative addressing saves address bits in the instruction.

Base-Register Addressing

- The referenced register "R" contains a main memory address
- address field contains a displacement A
- R may be explicit or implicit
- e.g. segment registers in 80x86

Indexed Addressing

- The address field references a main memory address A
- The referenced register R contains a positive displacement from that address.
- $EA = A + R$
- Good for accessing arrays
 - $EA = A + R$
 - R++

Auto Indexing

- Auto indexing incase certain registers are devoted exclusively to indexing

$$EA = A + (R)$$

$$(R) \leftarrow (R) + 1$$

Example: LODSB : Load byte at DS:[SI] into AL. Update SI.

AL = DS:[SI]

SI is incremented or decremented based on direction flag.

D = 0 → increment SI

D = 1 → decrement SI

- Two types:

- Postindex

- Preindex

Post-indexing

- indexing is performed after the indirection
$$EA = (A) + (R)$$
- Steps:
 1. The contents of the address field are used to access a memory location containing a direct address.
 2. Address is then indexed by the register value
- Use:
 - for accessing one of a number of blocks of data of a fixed format

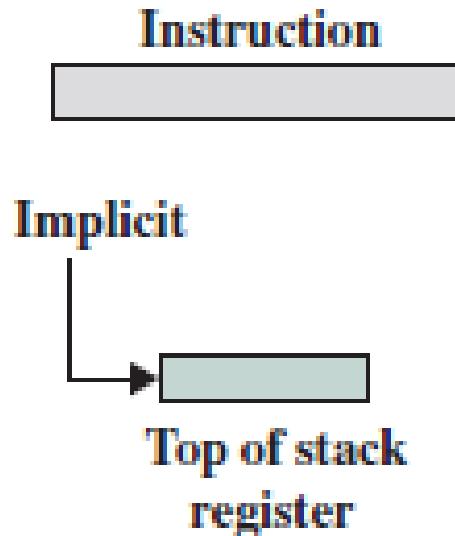
Pre-indexing



- An address is calculated as with simple indexing
$$EA = (A + (R))$$
- Use:
 - to construct a multiway branch table

Stack Addressing

- Operand is (implicitly) on top of stack
- e.g.
 - ADD Pop top two items from stack and add, push the result on stack top



x86 Addressing Modes

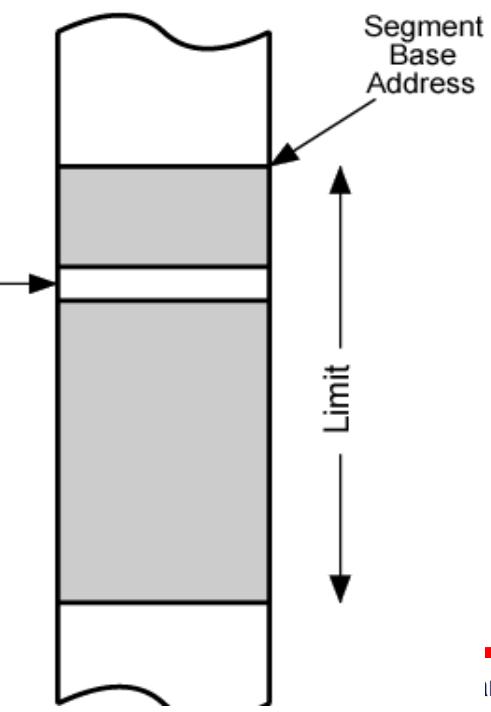
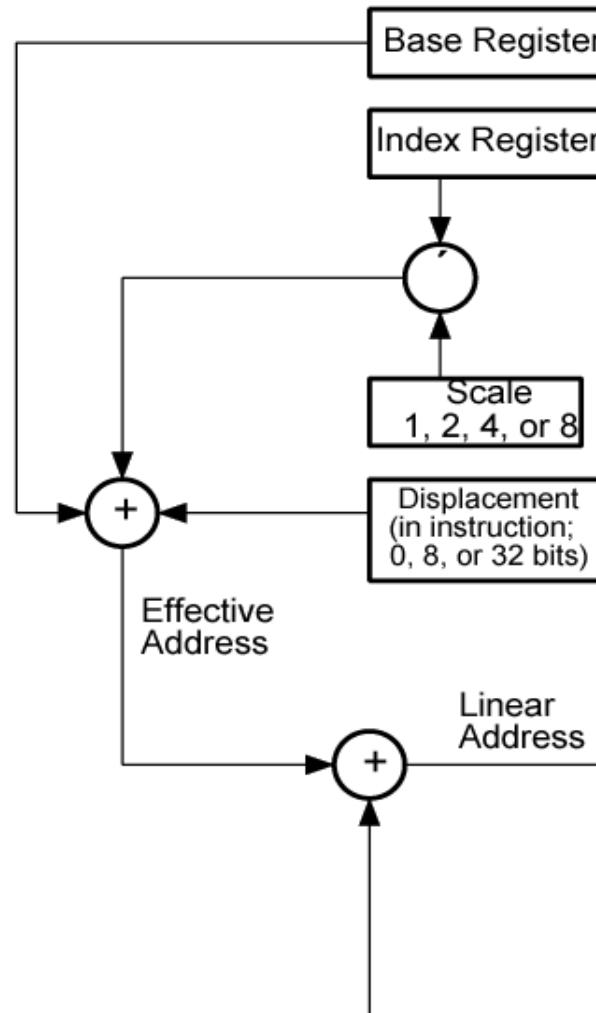
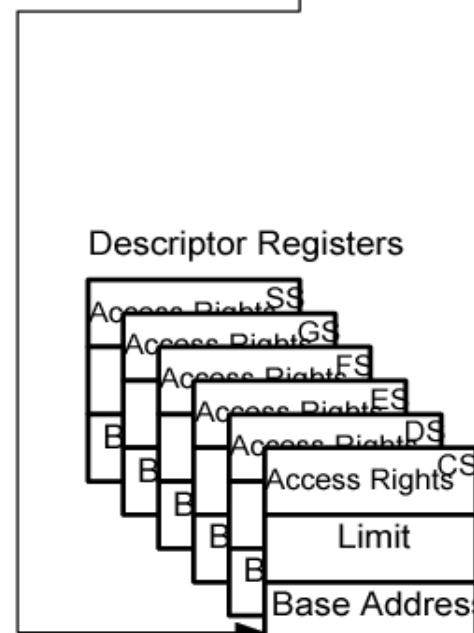
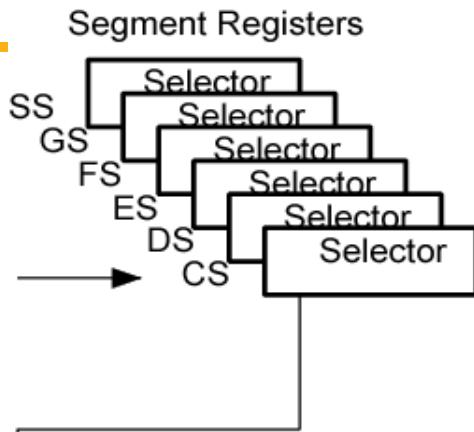
Virtual or effective address is offset into segment

- Starting address plus offset gives linear address
- This goes through page translation if paging enabled

12 addressing modes available

- Immediate
- Register operand
- Displacement
- Base
- Base with displacement
- Scaled index with displacement
- Base with index and displacement
- Base scaled index with displacement
- Relative

x86 Addressing Mode Calculation



Instruction Formats

- Layout of bits in an instruction
- Includes opcode
- Includes (implicit or explicit) operand(s)
- Usually more than one instruction format in an instruction set

Instruction Length

Affected by and affects:

- Memory size
- Memory organization
- Bus structure
- CPU complexity
- CPU speed

Trade off between powerful instruction repertoire and saving space

Allocation of Bits

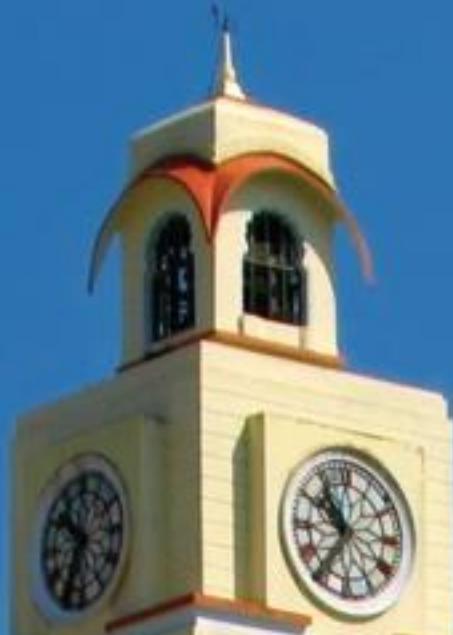
- Number of addressing modes
 - Number of operands
 - Register versus memory
 - Number of register sets
 - Address range
 - Address granularity
-



BITS Pilani
Pilani Campus

COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS SESSION 7

Prepared By: Prof. Lucy J Gudino
Instructor: Prof. C R Sarma



CISC Instruction Set (Intel x86 as an example)

BITS Pilani
Pilani Campus

Today's Session

Contact Hour	List of Topic Title	Text/Ref Book/external resource
13-14	<ul style="list-style-type: none"> • Instruction Set Architecture - CISC • CISC Instruction Set (Intel x86 as an example) Machine Instruction Characteristics, Types of Operands, Types of Operations, Addressing Modes, Instruction Formats • Control Unit Design Hardwired vs microprogrammed control unit 	T1, R1

Types of Operation

- Data Transfer
- Arithmetic
- Logical
- Conversion
- I/O
- System Control
- Transfer of Control

Data Transfer

- Specify
 - Source
 - Destination
 - Amount of data
- Action:
 1. Calculate the memory address, based on the address mode
 2. If the address refers to virtual memory, translate from virtual to real memory address.
 3. Determine whether the addressed item is in cache.
 4. If not, issue a command to the memory module.

Arithmetic

- Add, Subtract, Multiply, Divide
- May include
 - Absolute value ($|a|$)
 - Increment ($a++$)
 - Decrement ($a--$)
 - Negate ($-a$)
- Signed Integer

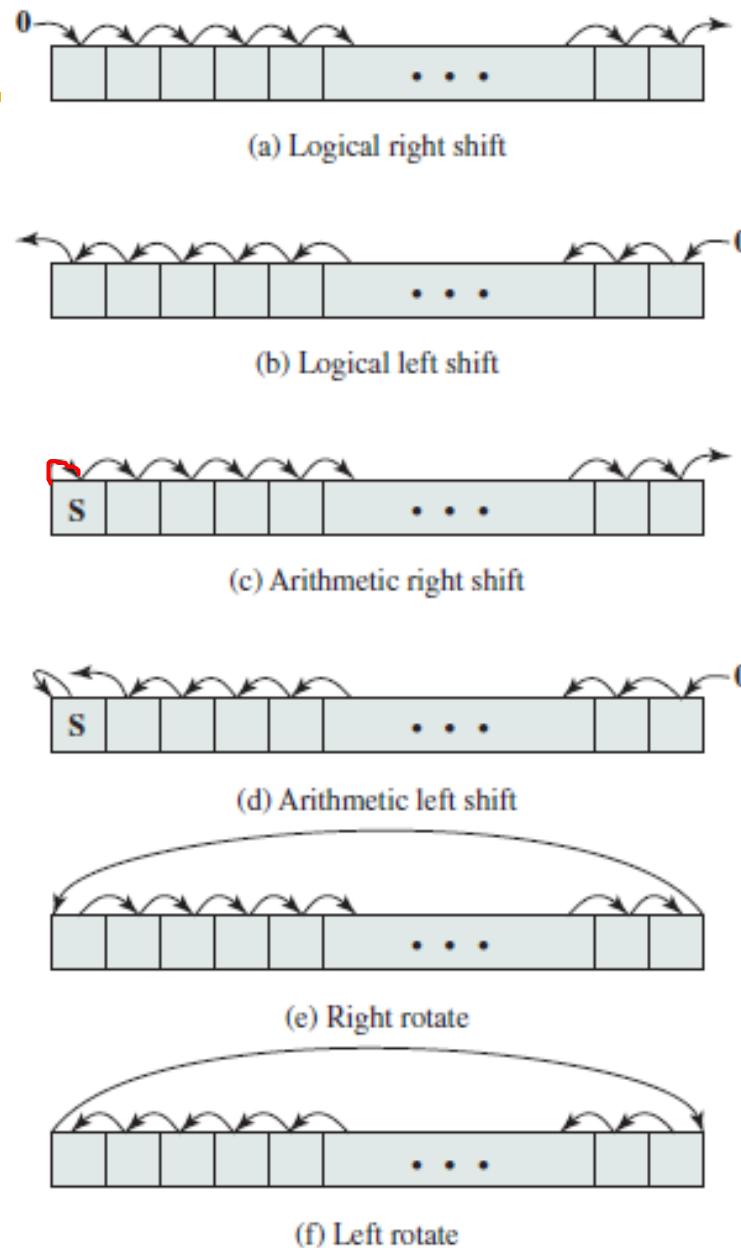
Logical

- Bitwise operations
- AND, OR, NOT

Basic Logical Operations

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P = Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

Shift and Rotate Operations



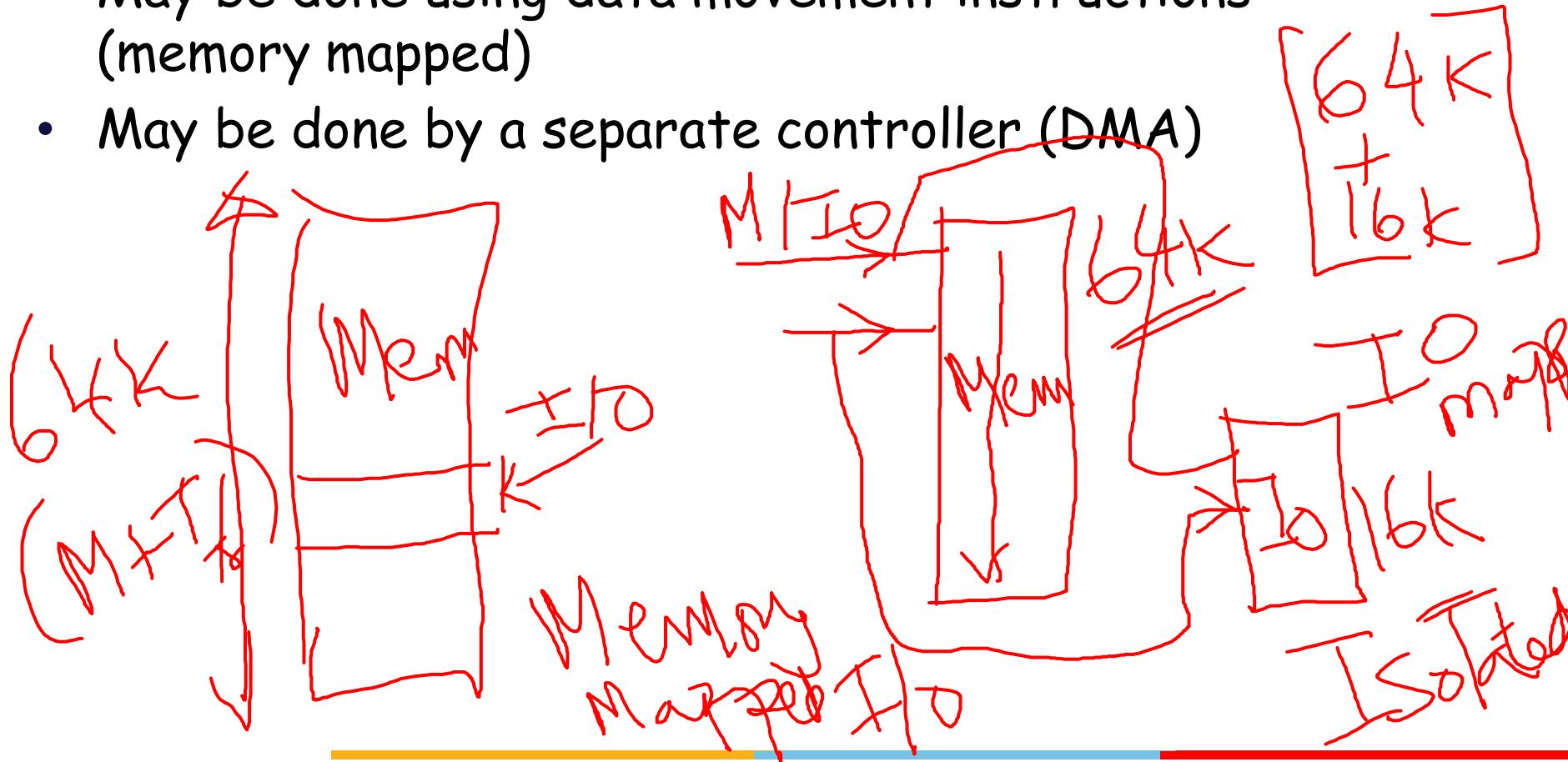
Input	Operation	Output
10101101	Logical right shift (3 bits)	10101101=> 00010101
10101101	Logical left shift (3 bits)	10101101=> 01101000
10101101	Arithmetic right shift (3 bits)	10101101=> 11110101
10101101	Arithmetic left shift (3 bits)	10101101=> 11101000
10101101	Right rotate (3 bits)	10101101=> 10110101
10101101	Left rotate (3 bits)	10101101=> 01101101

Conversion

E.g. Binary to Decimal

Input/Output

- May be specific instructions (I/O-Mapped I/O)
- May be done using data movement instructions (memory mapped)
- May be done by a separate controller (DMA)



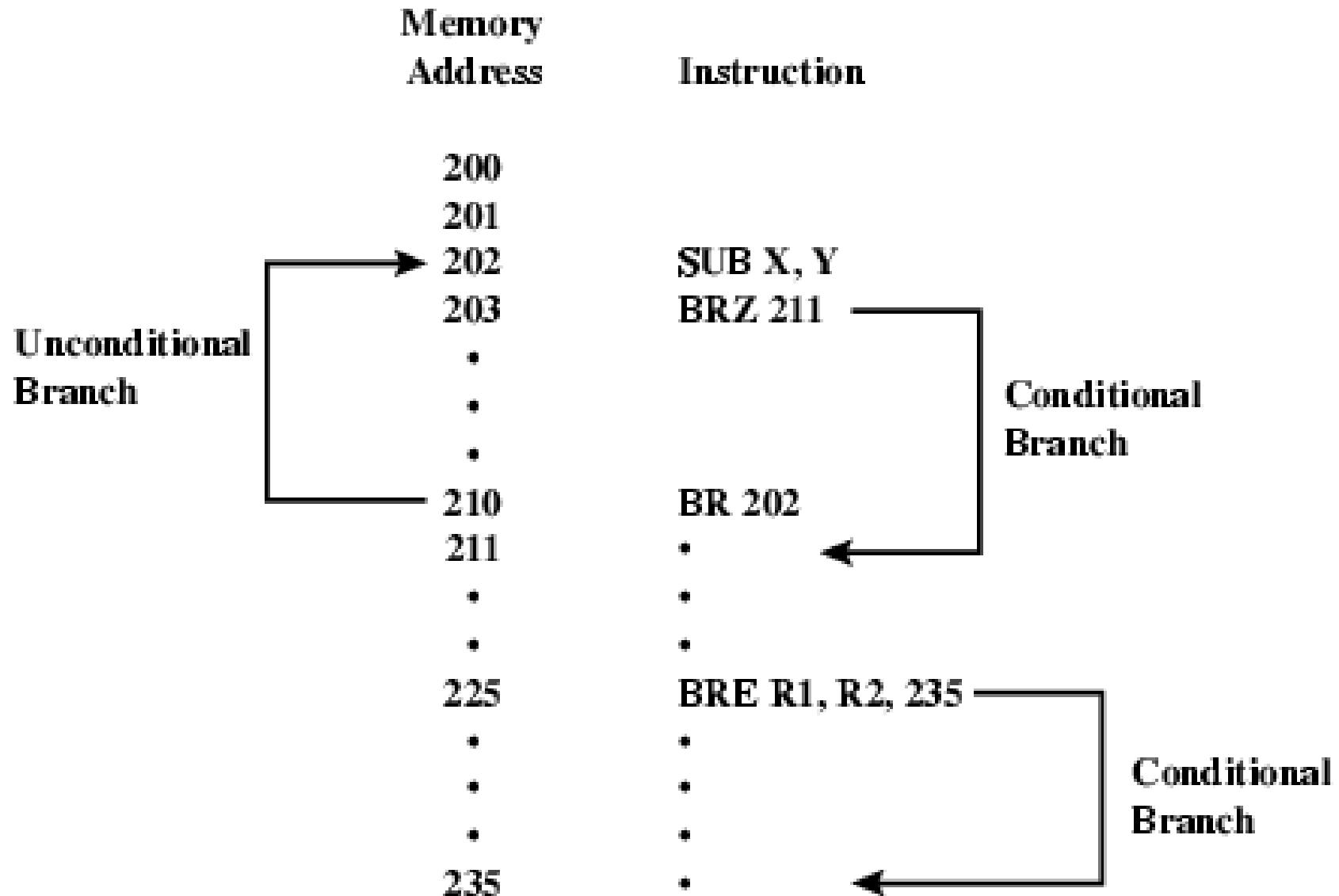
Systems Control

- Privileged instructions
- CPU needs to be in specific state
 - User Mode
 - Kernel mode
- For operating systems use

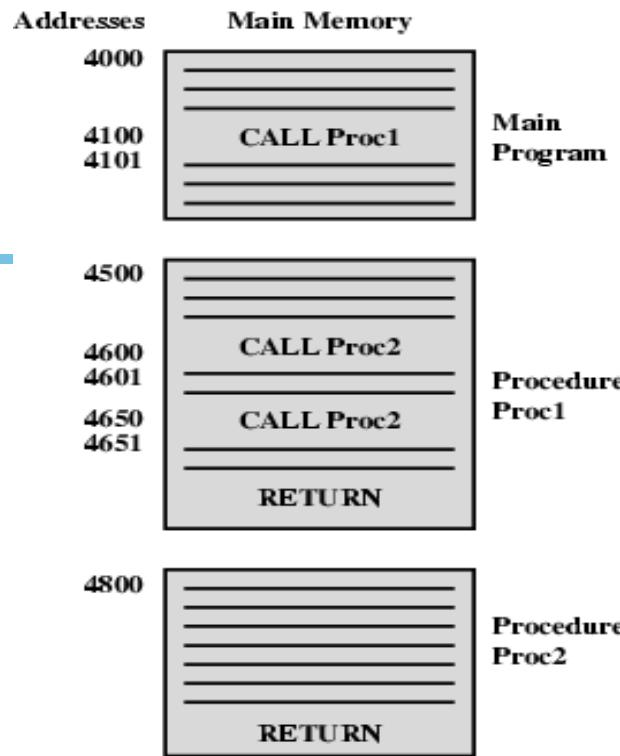
Transfer of Control

- Jump / Branch (Unconditional / Conditional)
 - e.g. jump to x if result is zero
- Skip (Unconditional / Conditional)
 - skip (unconditional) : Increment to skip next instruction
 - e.g. increment and skip if zero
- ISZ Register1
Branch xxxx
ADD A
- Subroutine call
- interrupt call

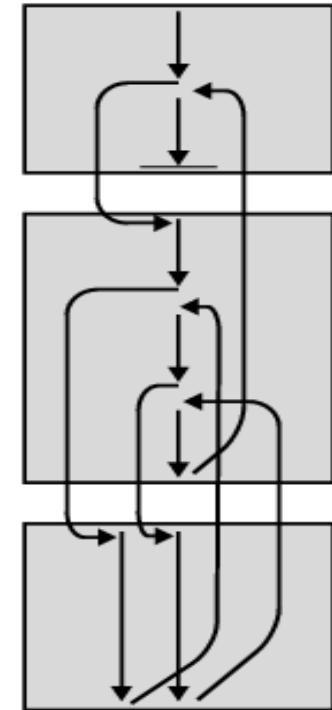
Branch / Jump Instruction



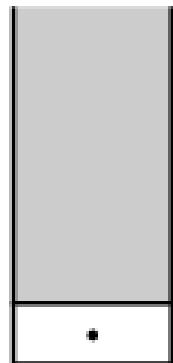
Use of Stack



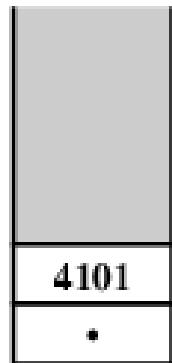
(a) Calls and returns



(b) Execution sequence



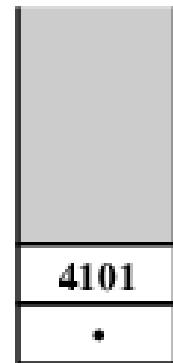
(a) Initial stack contents



(b) After CALL Proc1



(c) Initial CALL Proc2



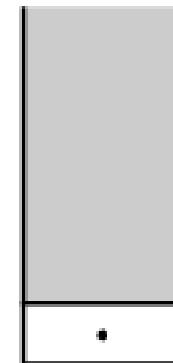
(d) After RETURN



(e) After CALL Proc2



(f) After RETURN



(g) After RETURN

Addressing Modes

- Addressing modes refers to the way in which the operand of an instruction is specified
- Types:
 - Immediate
 - Direct
 - Indirect
 - Register
 - Register Indirect
 - Displacement (Indexed)
 - Stack

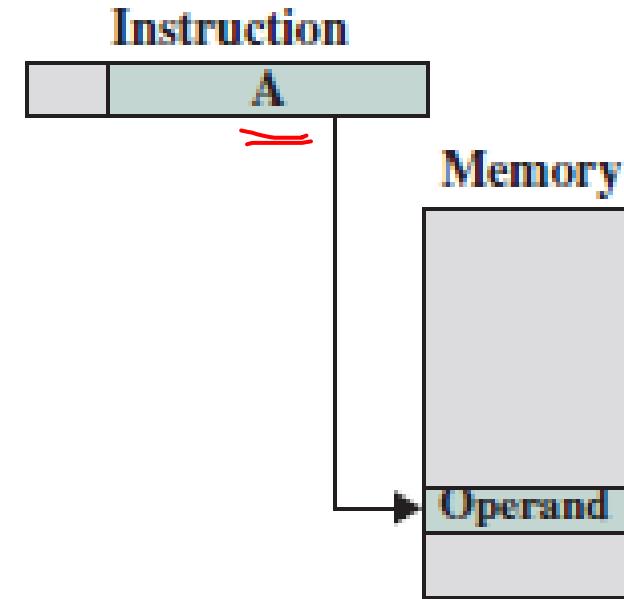
Immediate Addressing

- Operand is specified in the instruction itself
- e.g. ADD #5
 - Add 5 to contents of accumulator
 - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range



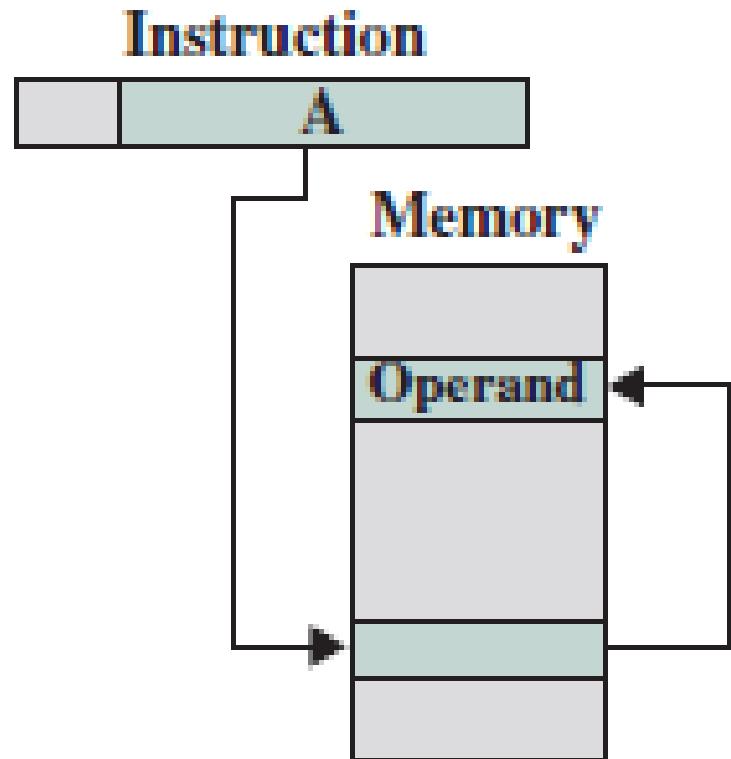
Direct Addressing

- Address of the operand is specified in the instruction
- Effective address (EA) = address field (A)
- e.g. ADD A
 - Add contents of memory cell whose address is A to accumulator
 - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space



Indirect Addressing

- Memory cell pointed to by address field of the instruction contains the address of (pointer to) the operand
- $EA = (A)$
 - Look in A , find address and look there for operand
- e.g. ADD (A)
 - Add contents of cell pointed to by contents of A to accumulator

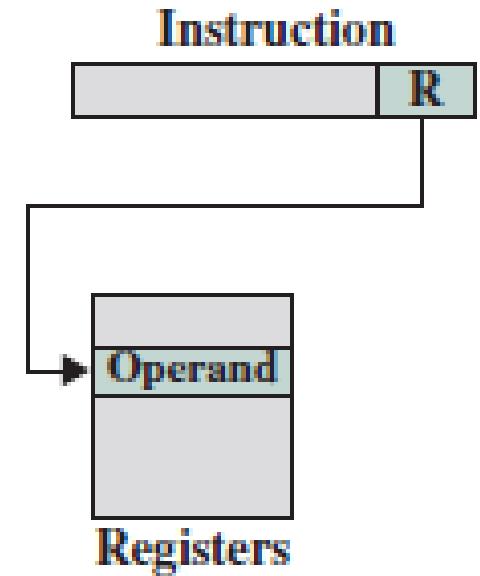


Indirect Addressing...

- Large address space
- 2^n where n = word length
- May be nested, multilevel, cascaded
 - e.g. EA = (((A)))
- Multiple memory accesses to find operand
- Slower

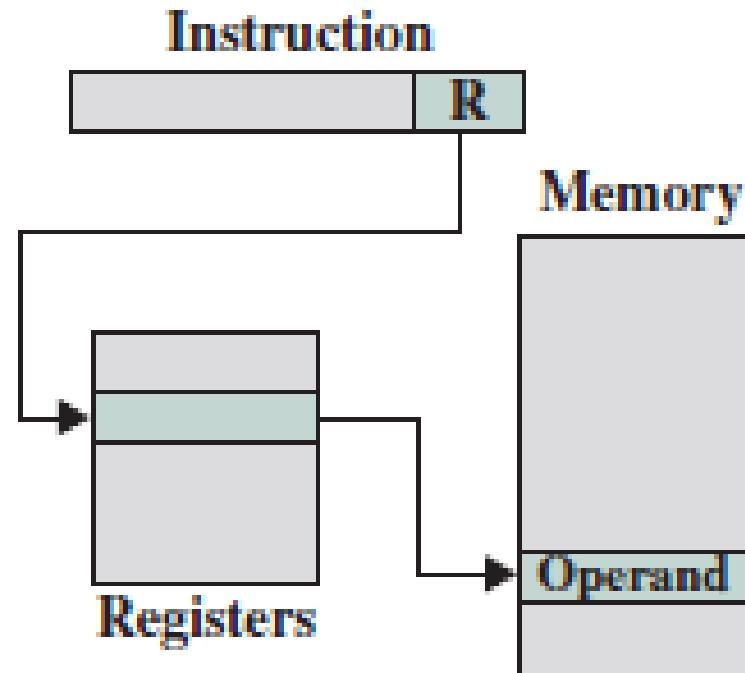
Register Addressing

- Operand is held in register named in address field
- $EA = R$
- Limited number of registers
- Very small address field needed
 - Shorter instructions
 - Faster instruction fetch
- No memory access hence Very fast execution but very limited address space
- Multiple registers helps in improving performance
 - Requires good assembly programming or compiler writing
 - C programming : `register int a;`



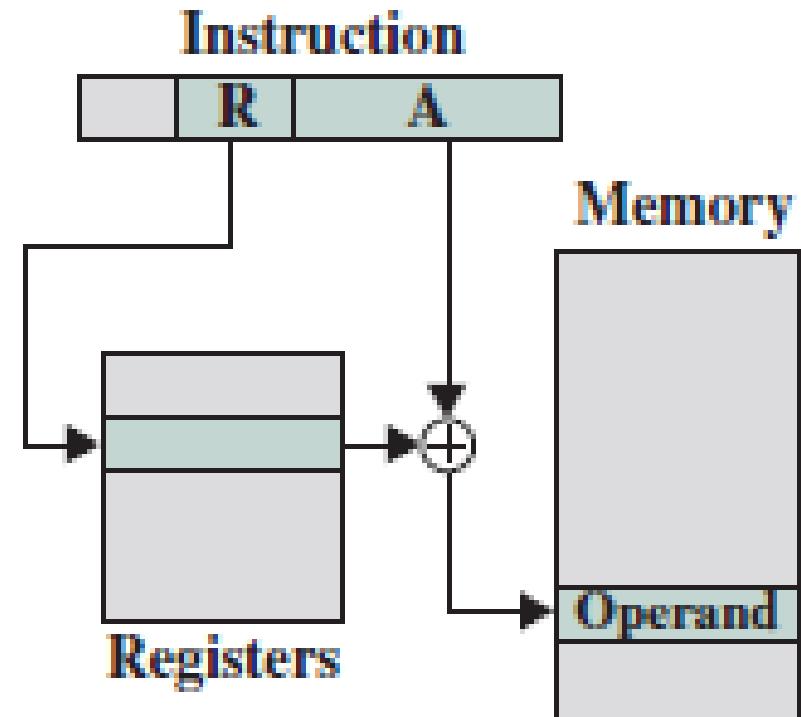
Register Indirect Addressing

- Similar to indirect addressing
- $EA = (R)$
- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One memory access compared indirect addressing



Displacement Addressing

- $EA = A + (R)$
- Address field hold two values
 - A = base value
 - R = register that holds displacement
 - or vice versa
- Three variants:
 - Relative addressing
 - Base register addressing
 - Indexing



Relative Addressing

- Also known as PC relative addressing
- A version of displacement addressing
- R = Program counter, PC
- $EA = A + (PC)$
- Relative addressing exploits the concept of locality
 - If most memory references are relatively near to the instruction being executed, then the use of relative addressing saves address bits in the instruction.

Base-Register Addressing

- The referenced register "R" contains a main memory address
- address field contains a displacement A
- R may be explicit or implicit
- e.g. segment registers in 80x86

Indexed Addressing

- The address field references a main memory address A
- The referenced register R contains a positive displacement from that address.
- $EA = A + R$
- Good for accessing arrays
 - $EA = A + R$
 - R++

Auto Indexing

- Auto indexing incase certain registers are devoted exclusively to indexing

$$EA = A + (R)$$

$$(R) \leftarrow (R) + 1$$

Example: LODSB : Load byte at DS:[SI] into AL. Update SI.

AL = DS:[SI]

SI is incremented or decremented based on direction flag.

D = 0 → increment SI

D = 1 → decrement SI

- Two types:

- Postindex

- Preindex

Post-indexing

- indexing is performed after the indirection
$$EA = (A) + (R)$$
- Steps:
 1. The contents of the address field are used to access a memory location containing a direct address.
 2. Address is then indexed by the register value
- Use:
 - for accessing one of a number of blocks of data of a fixed format

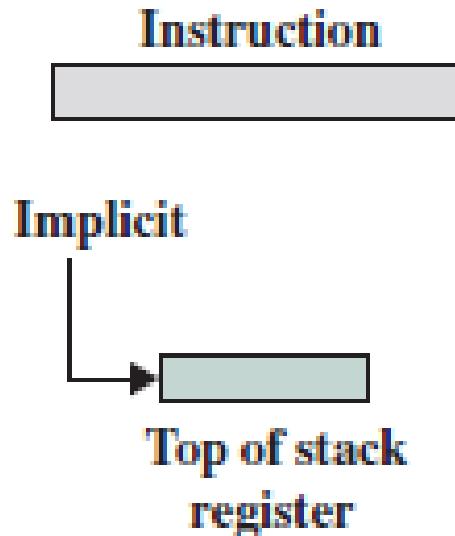
Pre-indexing



- An address is calculated as with simple indexing
$$EA = (A + (R))$$
- Use:
 - to construct a multiway branch table

Stack Addressing

- Operand is (implicitly) on top of stack
- e.g.
 - ADD Pop top two items from stack and add, push the result on stack top



x86 Addressing Modes

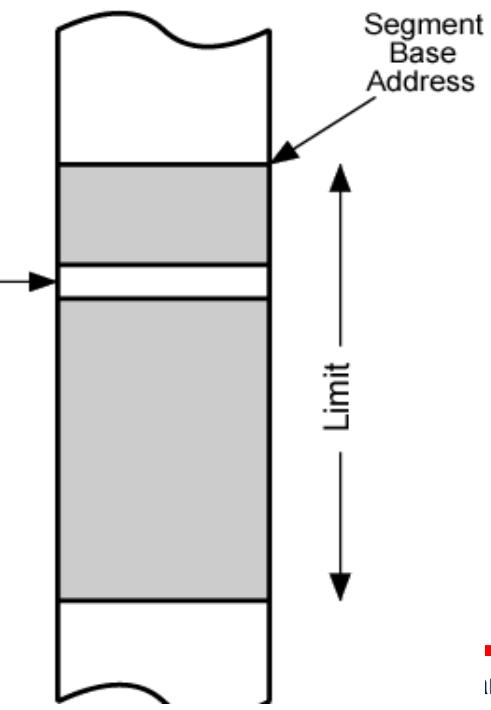
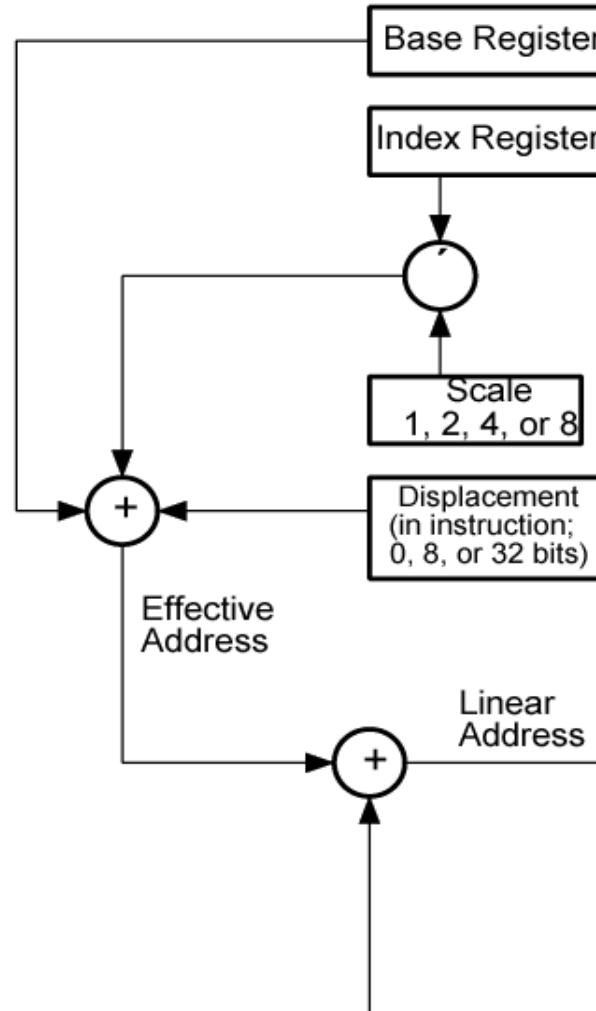
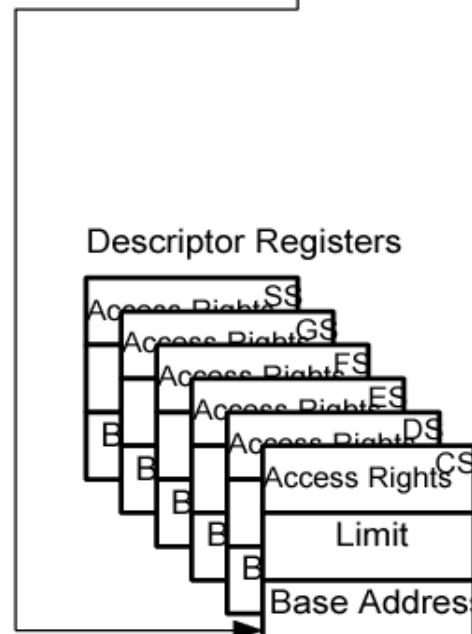
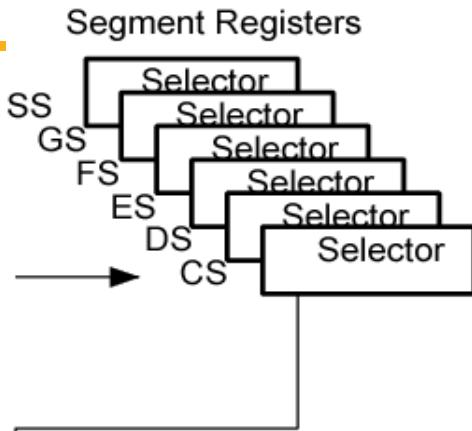
Virtual or effective address is offset into segment

- Starting address plus offset gives linear address
- This goes through page translation if paging enabled

12 addressing modes available

- Immediate
- Register operand
- Displacement
- Base
- Base with displacement
- Scaled index with displacement
- Base with index and displacement
- Base scaled index with displacement
- Relative

x86 Addressing Mode Calculation



Instruction Formats

- Layout of bits in an instruction
- Includes opcode
- Includes (implicit or explicit) operand(s)
- Usually more than one instruction format in an instruction set

Instruction Length

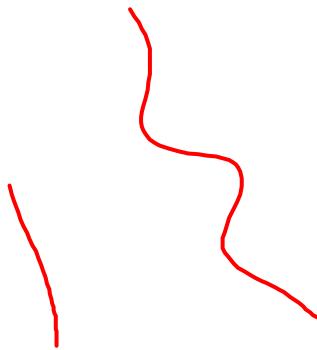
Affected by and affects:

- Memory size
- Memory organization
- Bus structure
- CPU complexity
- CPU speed

Trade off between powerful instruction repertoire and saving space

Allocation of Bits

- Number of addressing modes
 - Number of operands
 - Register versus memory
 - Number of register sets
 - Address range
 - Address granularity
-



End of Session #7



BITS Pilani
Pilani Campus

COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS SESSION 8

Prepared by: Dr. Lucy J Gudino
Instructor: Prof. C R Sarma

Instruction Length

Affected by and affects:

- Memory size
- Memory organization
- Bus structure
- CPU complexity
- CPU speed

Trade off between powerful instruction repertoire and saving space

Allocation of Bits

- Number of addressing modes
 - Number of operands
 - Register versus memory
 - Number of register sets
 - Address range
 - Address granularity
-



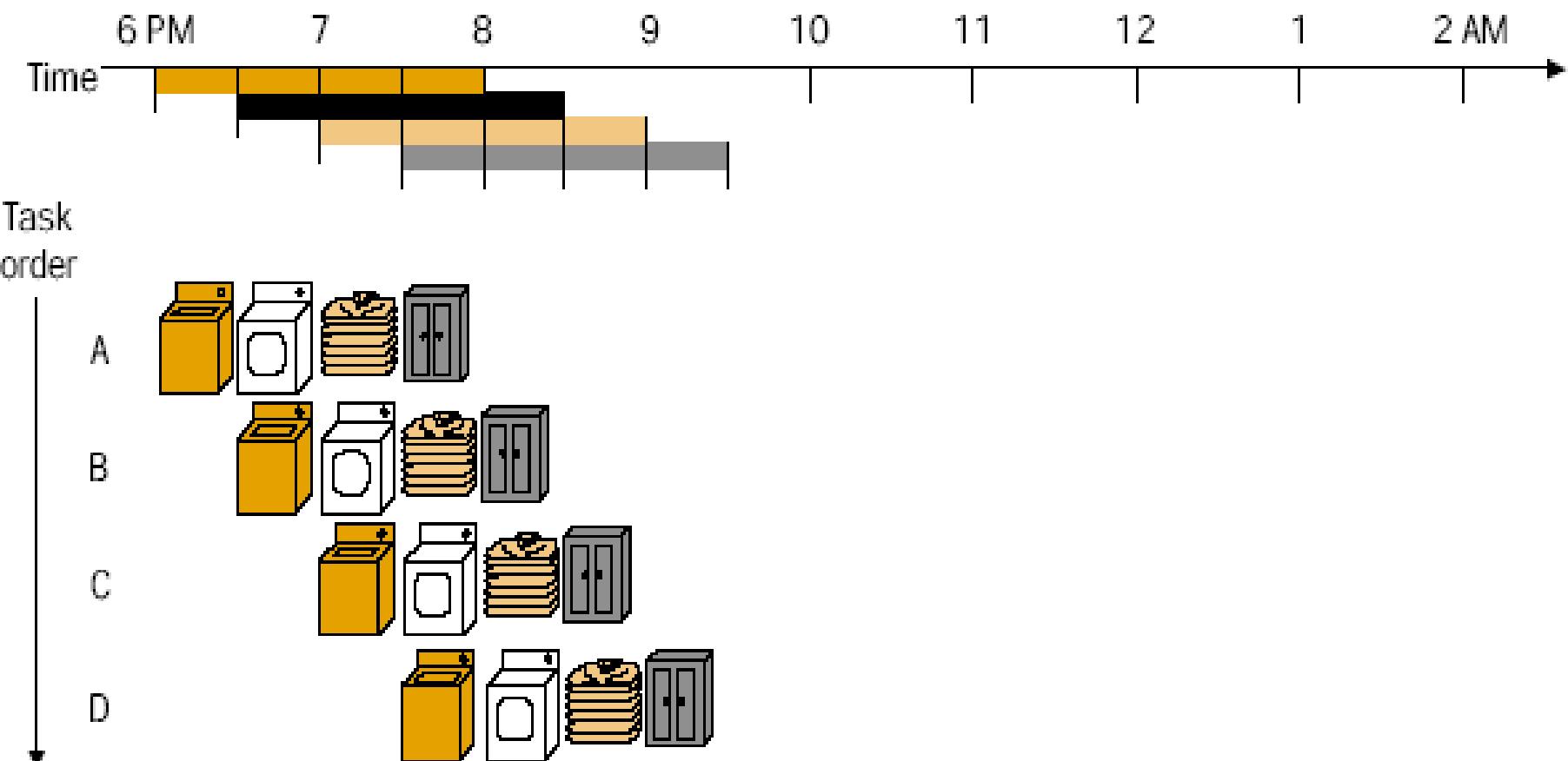
Pipeline

BITS Pilani
Pilani Campus

Laundry System



Laundry System.....



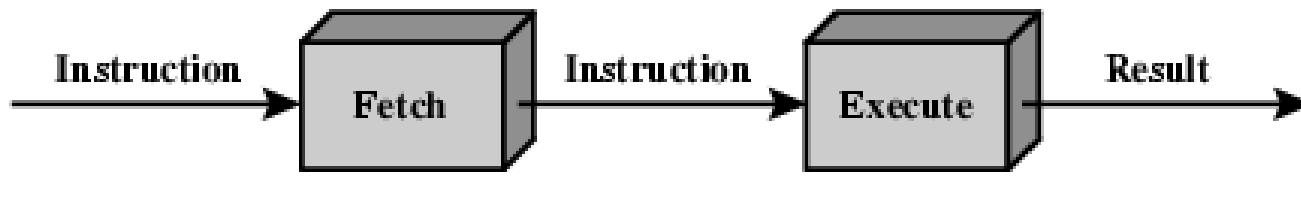
Pipelining

- An overlapped parallelism: overlapped execution of multiple operations
- Pipelining
 - Subdivide the input task into a sequence of subtasks
 - Specialized hardware stages
 - Concurrent operation of all the stages

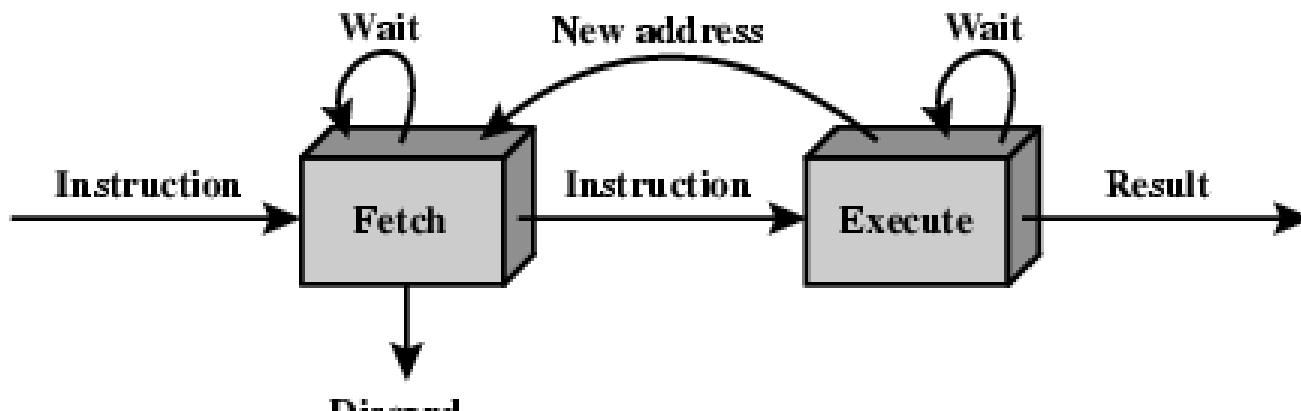
Two segment instruction pipeline

- Contains
 - Instruction Fetch (IF)
 - Execute (EX)
- Example: 8086 microprocessor
- Instruction Fetch unit is implemented by means of first in first out buffer (Queue)

Two Stage Pipeline...



(a) Simplified view



(b) Expanded view

Issues

1. The execution time will generally be longer than the fetch time
2. A conditional branch instruction makes the address of the next instruction to be fetched unknown.

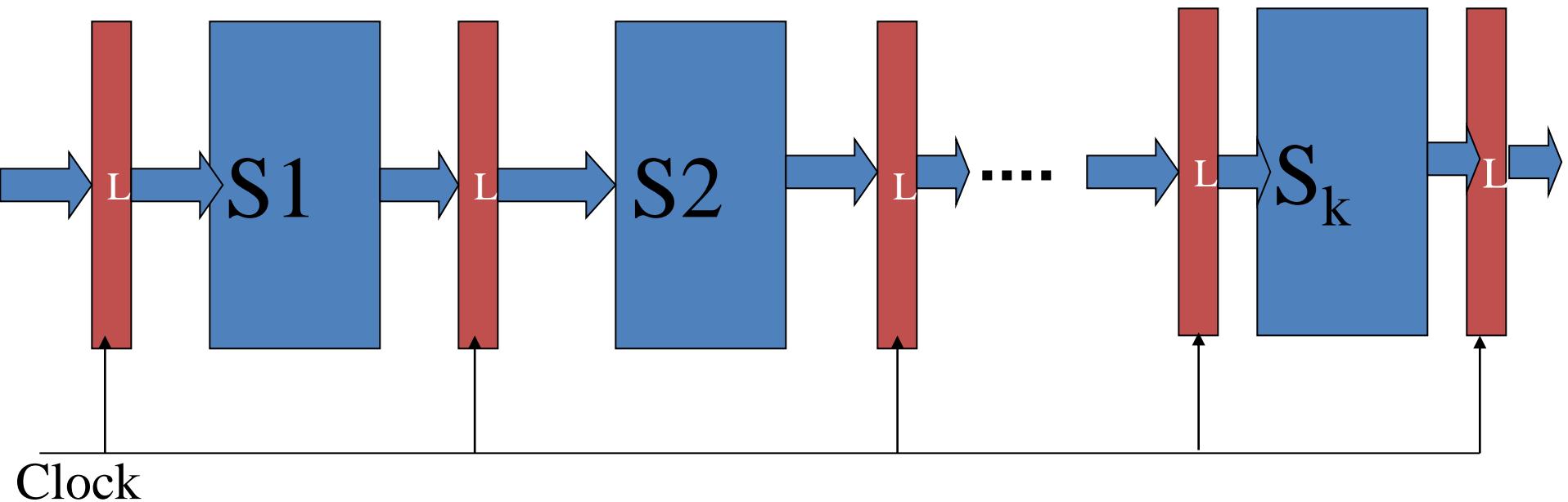
Four Segment instruction pipeline

- Contains
 - FI : fetch instruction
 - DA : Decode instruction and calculate effective address
 - FO :Fetch operand
 - EX: Execute instruction

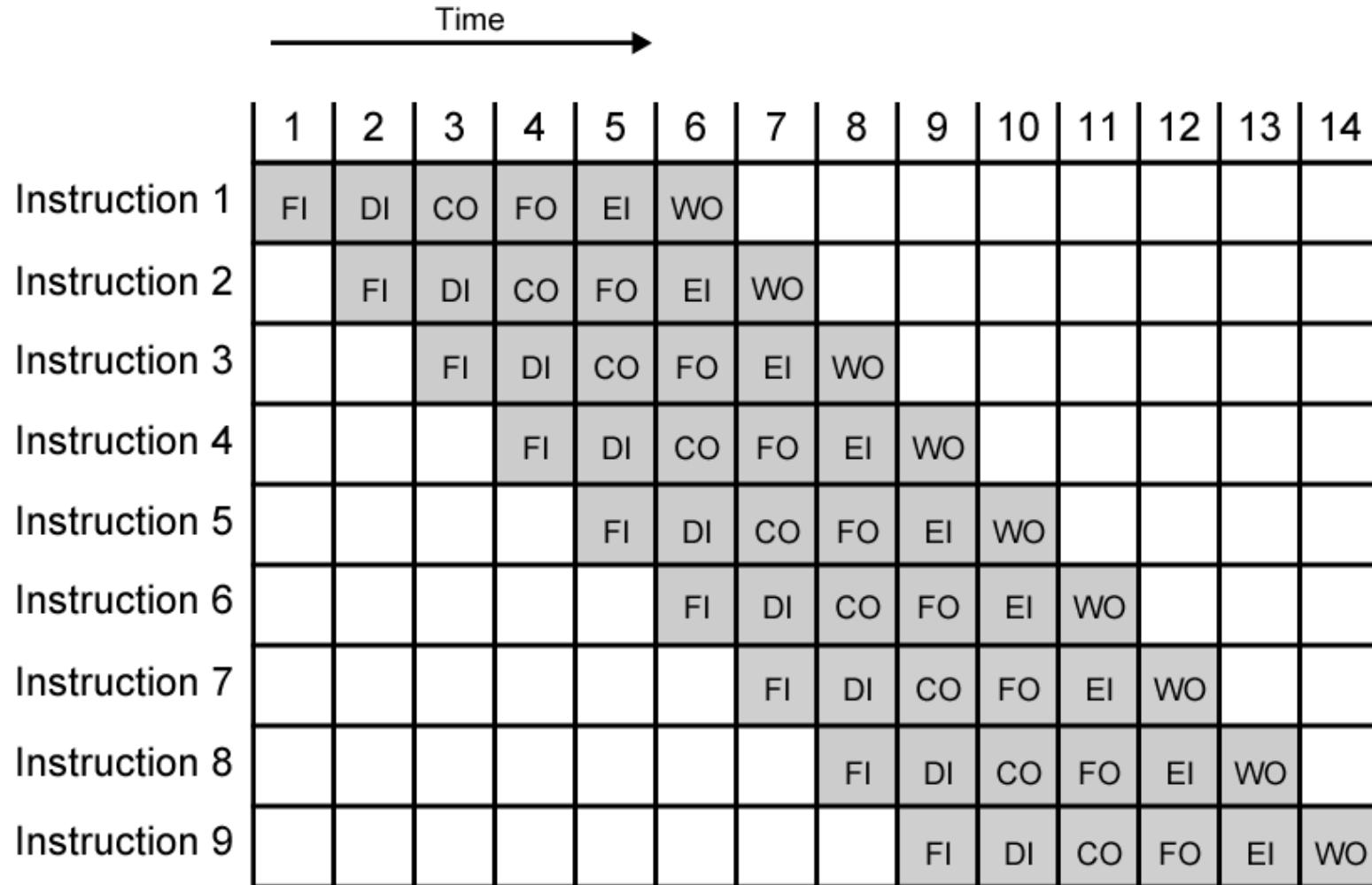
Six stage pipeline

- Contains
 - FI : fetch instruction
 - DI : Decode instruction
 - CO : calculate effective address
 - FO :Fetch operand
 - EI : Execute instruction
 - WO : Write Operand
-

Structure of a pipeline



Timing Diagram for Instruction Pipeline Operation



Classification

- Arithmetic pipelining
- Instruction pipelining
- Processor pipelining
- Unifunction and multifunction pipelining
- Static and Dynamic pipelining
- Scalar and Vector pipelining

Arithmetic pipelining

- Arithmetic and logic units of a computer can be segmented for pipeline operations
- Usually found in high speed computers
- Example:
 - Star 100 → 4 stage ✓
 - TI-ASC → 8 stage ✓
 - Cray-1 → 14 stage ✓
 - Cyber 205 → 26 stages
 - Intel Cooper Lake (3rd Gen Intel Xeon) = 14 stages
- Floating point adder pipeline

$$X = A * 2^a$$

$$Y = B * 2^b$$

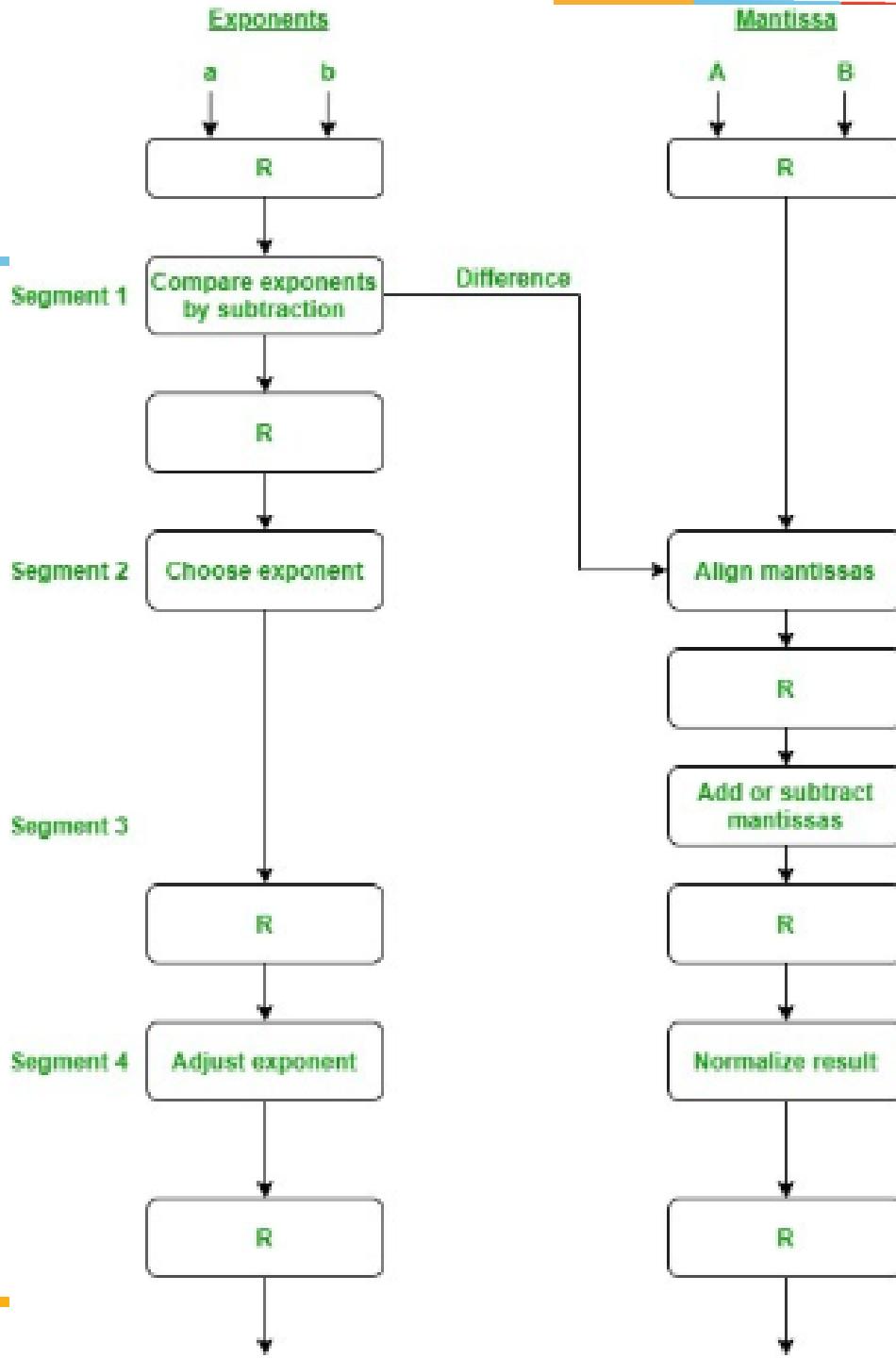
Floating point addition

1. Compare the exponents
2. Align the mantissas
3. Add or subtract the mantissas
4. Normalize the result

Numerical Example:

$$X = 0.9504 \times 10^3$$

$$Y = 0.8200 \times 10^2$$

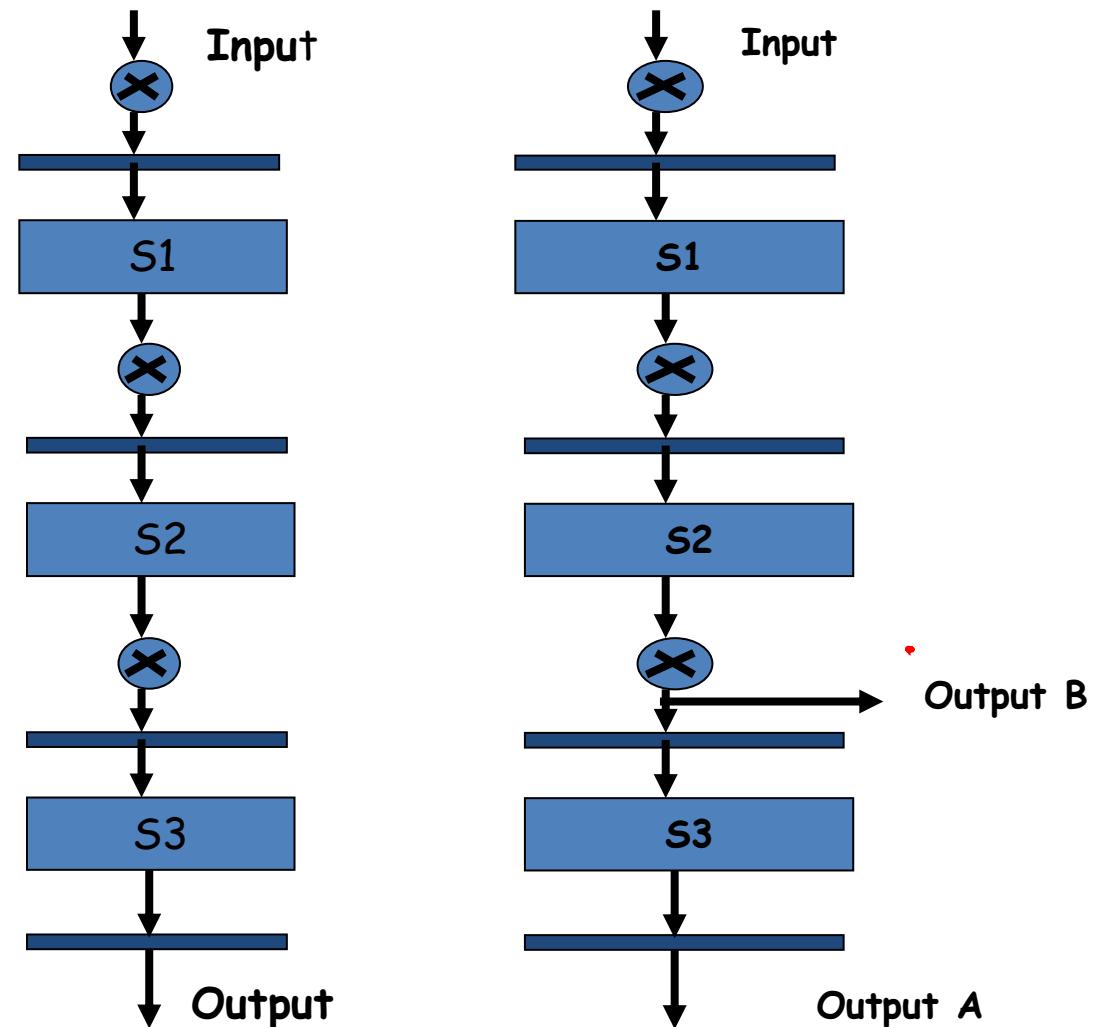


Instruction pipelining

- The execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode.....of subsequent instructions
- Sequence of steps followed in most general purpose computer to process instruction
 1. IF: Fetch the instruction from memory
 2. ID: Decode the instruction
 3. CO: Calculate the effective address
 4. FO: Fetch the operands from memory
 5. EI: Execute the instruction
 6. WO: Store the result in the proper place

Unifunction and multifunction pipelining

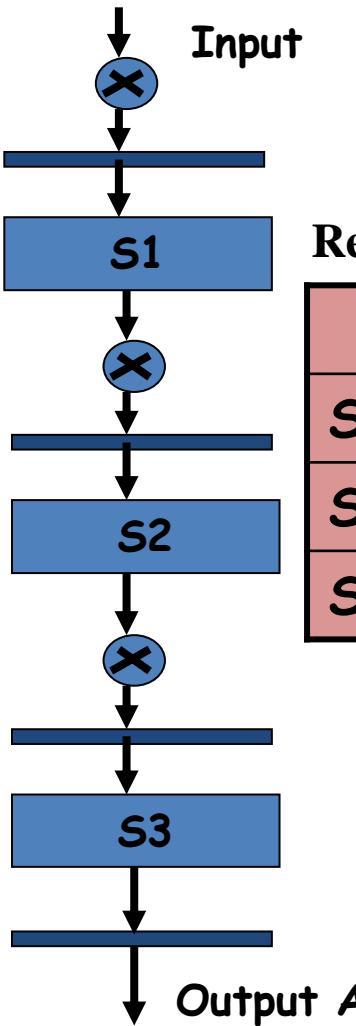
- Unifunction
 - Pipeline with a fixed and dedicated function
 - Ex: Floating point adder
- Multifunction
 - Pipeline may perform different functions



Reservation Table

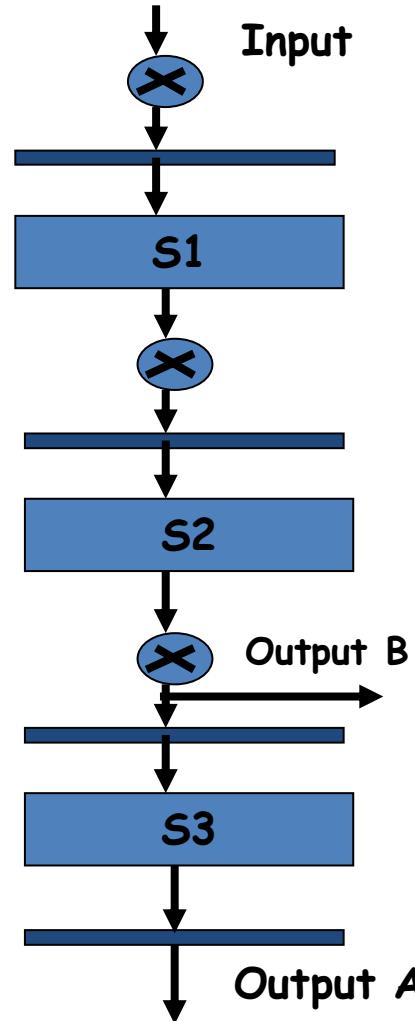
- Is a two dimensional chart
- Used to show how successive pipeline stages are utilized or reserved

Uni-function Vs Multifunction



Reservation Table A

	T0	T1	T2
S1	A		
S2		A	
S3			A



Reservation Table A

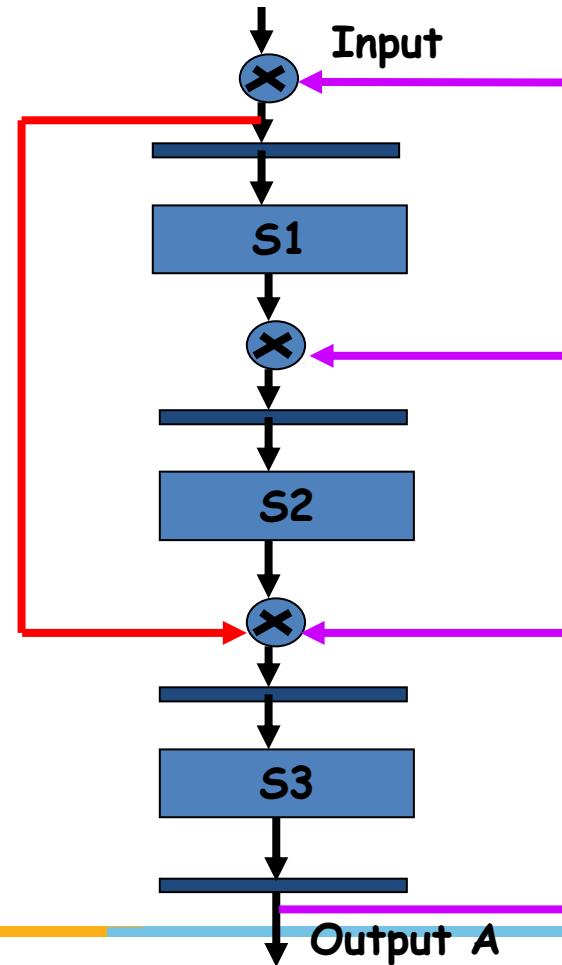
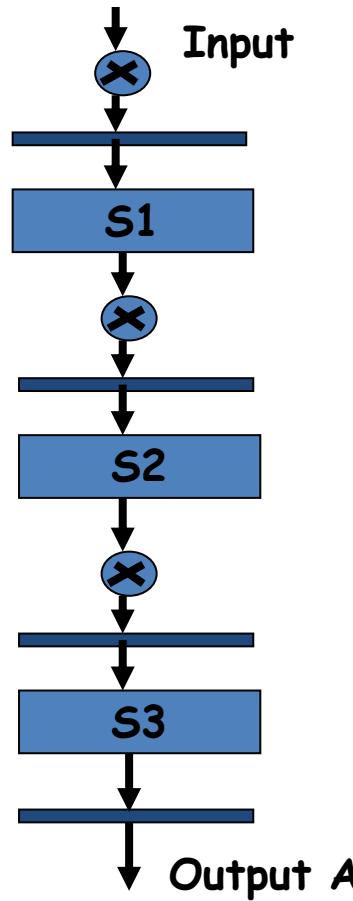
	T0	T1	T2
S1	A		
S2		A	
S3			A

Reservation Table B

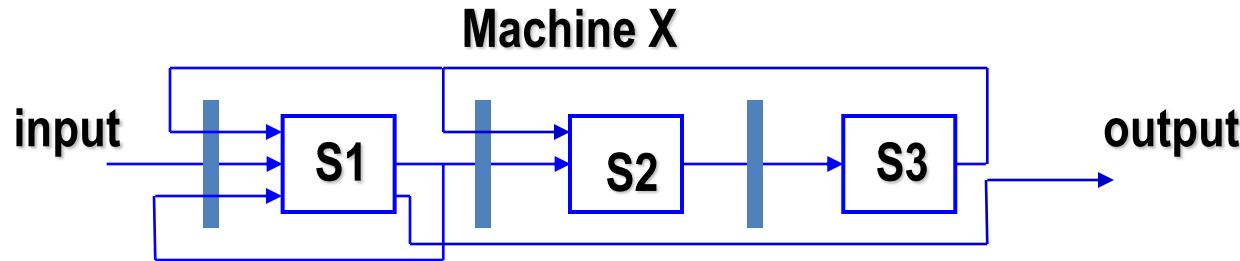
	T0	T1
S1	B	
S2		B

Linear and Nonlinear Pipelines

- Linear Pipeline: Without feed forward and feed back connection
- Nonlinear Pipeline with feed forward and/or feed back connection



Reservation Table

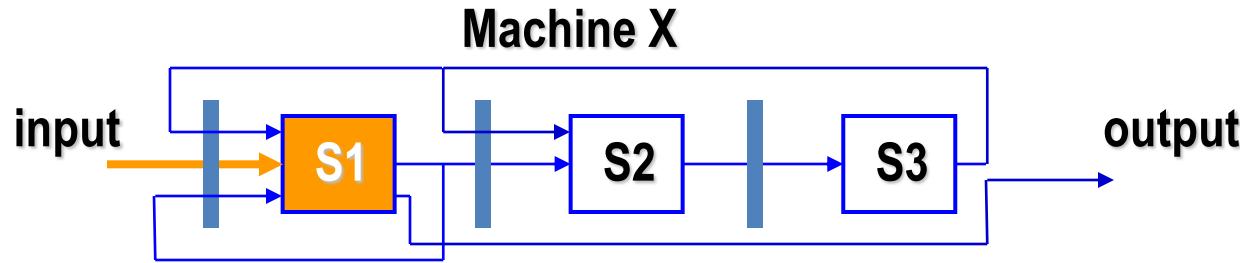


Reservation Table

Time →

Stage ↓	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Reservation Table

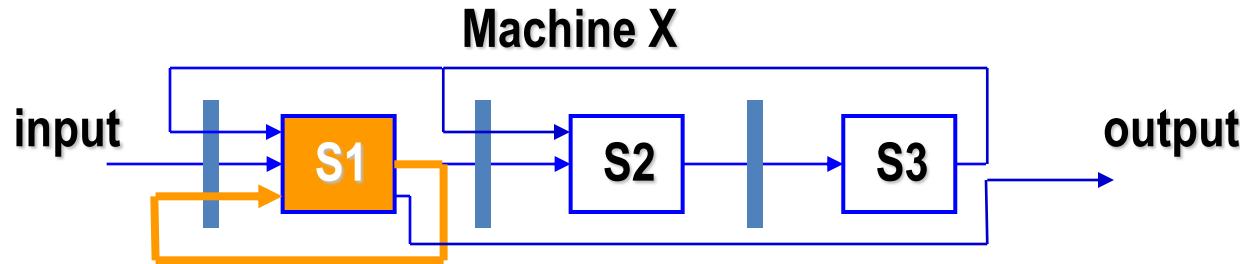


Reservation Table

Time →

	0	1	2	3	4	5	6	7
Stage ↓	X	X					X	X
S1								
S2			X		X			
S3				X		X		

Reservation Table

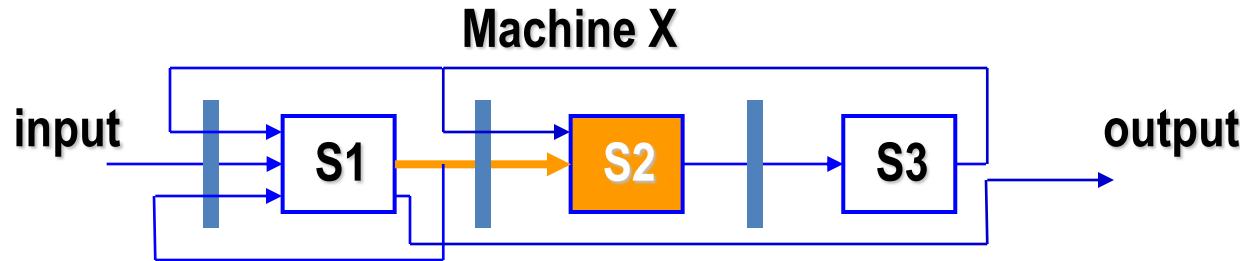


Reservation Table

Time →

	0	1	2	3	4	5	6	7
Stage ↓	X	X					X	X
S1								
S2			X		X			
S3				X		X		

Reservation Table



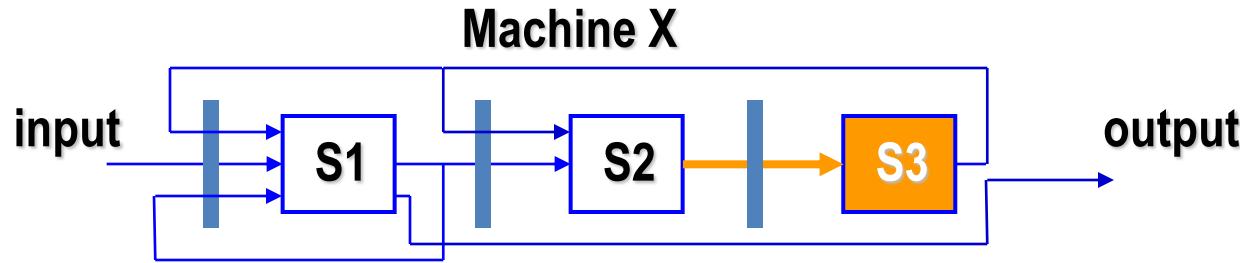
Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Stage ↓

Reservation Table

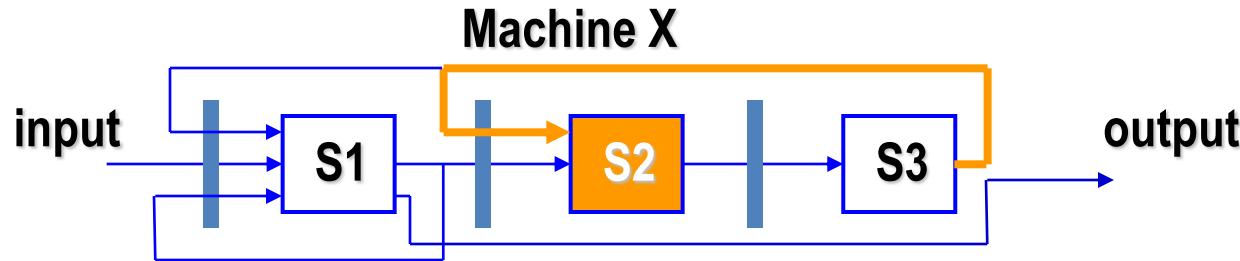


Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X			X	

Reservation Table

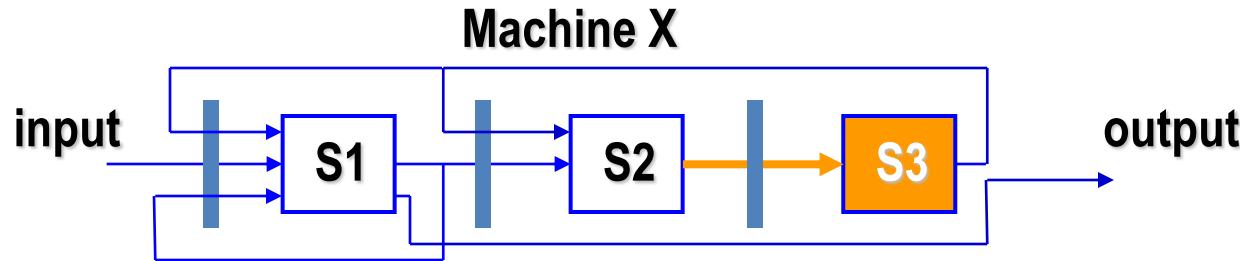


Reservation Table

Time →

	0	1	2	3	4	5	6	7
Stage ↓	X	X					X	X
S1								
S2			X		X			
S3				X		X		

Reservation Table

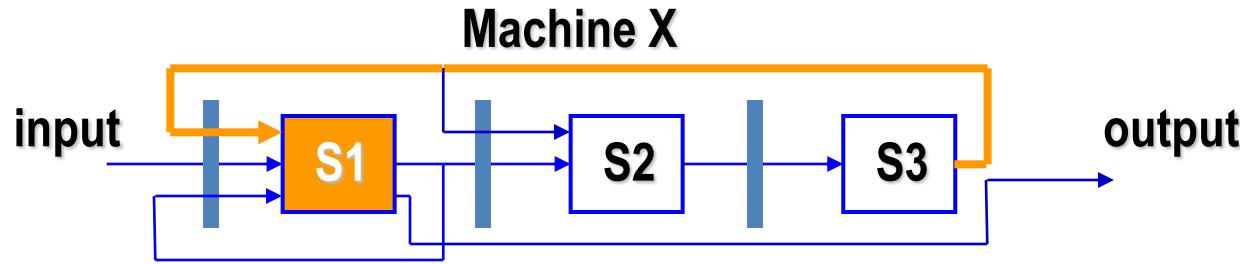


Reservation Table

Time →

	0	1	2	3	4	5	6	7
Stage ↓	X	X					X	X
S1								
S2			X		X			
S3				X		X		

Reservation Table

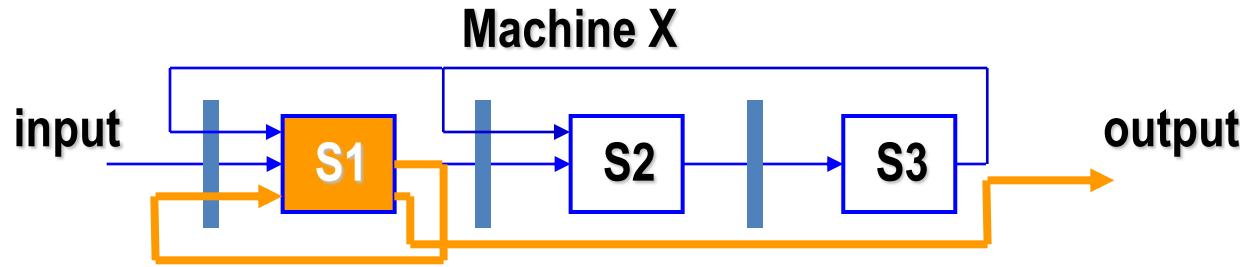


Reservation Table

Time →

	0	1	2	3	4	5	6	7
Stage ↓	X	X					X	X
S1								
S2			X		X			
S3				X		X		

Reservation Table

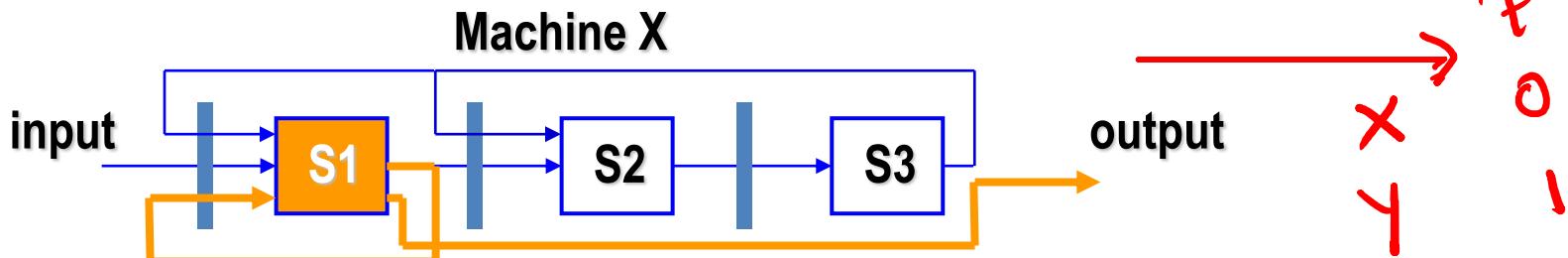


Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Reservation Table



Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

time-space ↓

A horizontal number line starting at 0 and ending at 19. The numbers are labeled above the line. Red 'X' marks are placed at several integer positions: 0, 1, 2, 3, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, and 19. A red circle highlights the 'X' at position 5. Another red circle highlights the 'X' at position 4, which is also circled.

Static and Dynamic pipelining

- Dynamic pipeline allows more frequent changes in its configuration
- Require more elaborate sequencing and control mechanisms

Scalar and Vector pipelining

- Based on the operand types or instruction type
- Scalar pipeline processes scalar operands
- Vector pipeline operate on vector data and instructions.

Important Terms

Clock period: τ

τ_i : time delay of S_i stage

τ_L : time delay of latch

$$\tau = \max\{\tau_i\} + \tau_L$$

Pipeline processor frequency $f = 1/\tau$

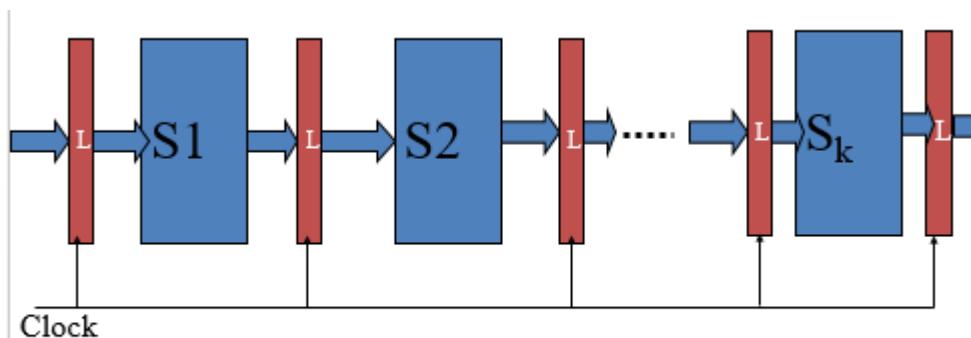


Fig: Structure of a pipeline

Important Terms

Time taken to complete n tasks by k stage pipeline is

$$T_k = [k + (n-1)]\tau$$

Time taken by the nonpipelined processor ?

$$T_1 = k * n * \tau$$

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Important Terms

- Speedup: speedup of a k-stage linear-pipeline over an equivalent nonpipelined processor

$$\begin{aligned}
 S_k &= \frac{T_1}{T_k} \\
 &= \frac{n * k * \tau}{[k + (n - 1)]\tau} \\
 &= \frac{n * k}{[k + (n - 1)]}
 \end{aligned}$$

- The maximum speedup is $S_k \rightarrow k$ when $n \rightarrow \text{INF}$
- Maximum speedup is very difficult to achieve because of data dependencies between successive tasks, program branches, interrupts etc.

Important Terms

Efficiency: the ratio of actual speedup to ideal speedup

$$\eta = \frac{n.k}{k.[k + (n - 1)]}$$

$$= \frac{n}{[k + (n - 1)]}$$

- Maximum efficiency
 $\eta \rightarrow 1$ as $n \rightarrow \infty$
- Implies that the larger the number of tasks flowing through the pipeline, the better is its efficiency
- In steady state of a pipeline, we have $n \gg k$, then efficiency should approach 1
- However, this ideal case may not hold all the time because of program branches and interrupts and data dependencies

Important Terms

Throughput : The number of tasks that can be completed by a pipeline per unit time

$$H_k = \frac{n}{[k + (n - 1)]\tau} = \frac{nf}{[k + (n - 1)]} = \eta f$$

Problem 1

$$\frac{(k + (n-1))\tau}{(6 + 7)\tau} = 13 \text{ C}$$

Draw a space-time diagram for a six segment pipeline showing the time it takes to process eight tasks

Determine the number of clock cycles that it takes to process 200 tasks in a six segment pipeline

	1	2	3	4	5	6	7	8	9	10	11	12	13
S1	T1	T2	T3	T4	T5	T6	T7	T8					
S2		T1	T2	T3	T4	T5	T6	T7	T8				
S3			T1	T2	T3	T4	T5	T6	T7	T8			
S4				T1	T2	T3	T4	T5	T6	T7	T8		
S5					T1	T2	T3	T4	T5	T6	T7	T8	
S6						T1	T2	T3	T4	T5	T6	T7	18

Problem 2

Assume each task is subdivided in to 6 subtasks and clock cycle is 10 microseconds.

- Determine the number of clock cycles that is taken to process 50 tasks.
- Determine the number of clock cycles that is taken to process 50 tasks in non-pipeline processor
- Compute speed up, efficiency and throughput

Pipeline Hazards / Conflicts

- **Resource Hazard**: access to same resource by two segments at the same time
- **Data Hazard** : an instruction depends on the result of a previous instruction, but this result is not yet available
- **Control Hazard**: arise from branch and other instructions that change the value of PC

Resource Hazard

Instruction

	Clock cycle								
	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2		FI	DI	FO	EI	WO			
I3			FI	DI	FO	EI	WO		
I4				FI	DI	FO	EI	WO	



Problem



Consider the following code. Assume that initial contents of all the registers is zero.

```
START:    MOV R4, #1  
          MOV R3, #2  
          MOV R1, #2  
          MOV R2, #3  
          ADD R3, R1, R2  
          SUB R4, R3, R2
```

- Write timing of instruction pipeline with FIVE stages
- What is the content of various registers after the execution of program?

Data Dependency Conflict

START:

```
MOV R4, #1  
MOV R3, #2  
MOV R1, #2  
MOV R2, #3  
ADD R3, R1, R2  
SUB R4, R3, R2
```

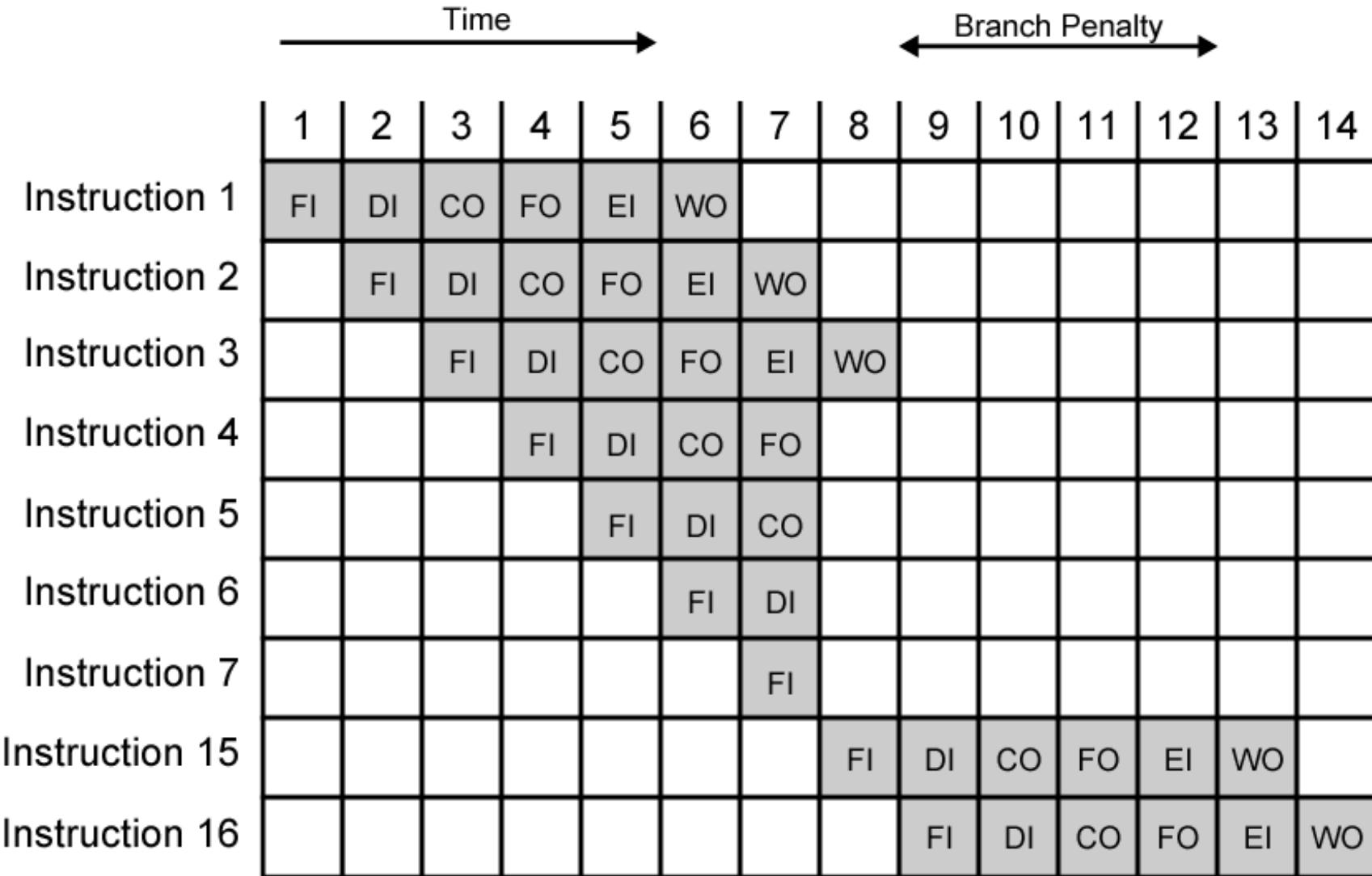
time	1	2	3	4	5	6	7	8	9	10
I1	FI	DI	FO	EI	WO					
I2		FI	DI	FO	EI	WO				
I3			FI	DI	FO	EI	WO			
I4				FI	DI	FO	EI	WO		
I5					FI	DI	FO	EI	WO	
I6						FI	DI	FO	EI	WO

Problem

```
START: ADD R3, R1, R2
       SUB R4, R1, R2
       JNZ NEXT
       MUL R5, R1, R2
       ADD R6, R3, R4
       :
NEXT:  ST R3, LOCN1
       HLT
```

Write time – space diagram for instruction pipeline

The Effect of a Conditional Branch on Instruction Pipeline Operation



Dealing with Branches

Multiple Streams

Prefetch Branch Target

Loop buffer

Branch prediction

Delayed branching

Multiple Streams

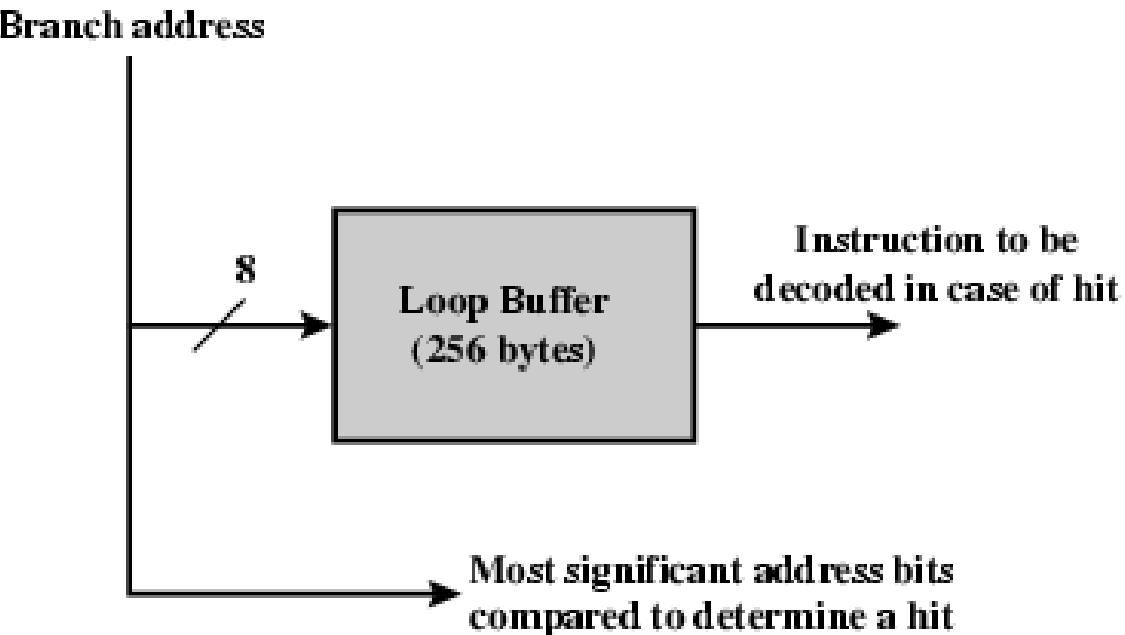
- Have two pipelines
- Pre-fetch each branch into a separate pipeline
- Use appropriate pipeline
- Issues :
 - Leads to bus & register contention
 - Multiple branches lead to further pipelines being needed

Prefetch Branch Target

- Target of branch is prefetched in addition to instructions following branch
- Keep target until branch is executed
- Used by IBM 360/91

Loop Buffer

- Very fast memory
- Maintained by fetch stage of pipeline
- Check buffer before fetching from memory
- Very good for small loops or jumps
- Working principle is similar to cache
- Used by CRAY-1



Branch Prediction (1)

- Predict never taken
- Predict always taken
- Predict by opcode
- Taken/not taken switch
- Branch history table

Branch Prediction (2)

- Predict never taken
 - Assume that jump will not happen
 - Always fetch next instruction
- Predict always taken
 - Assume that jump will happen
 - Always fetch target instruction
- Predict by Opcode
 - Some instructions are more likely to result in a jump than others
 - Can get up to 75% success

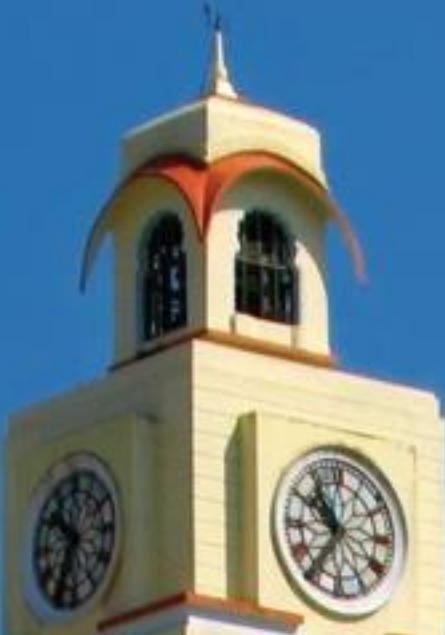


BITS Pilani
Pilani Campus

COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS SESSION 9

Prepared By: Dr. Lucy J. Gudino
Instructor : Prof. C R Sarma
WILP & Department of CS & IS





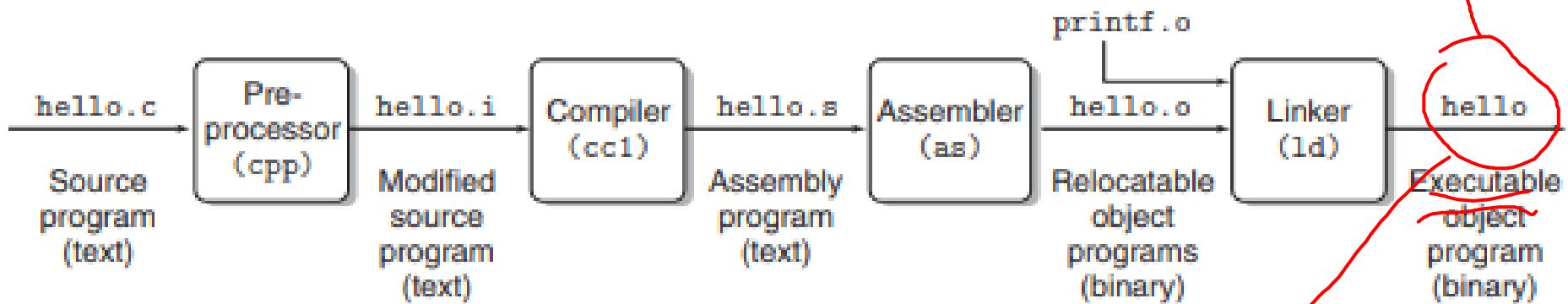
BITS Pilani
Pilani Campus

Process Management

Prepared By: Dr. Lucy J. Gudino
Instructor: <Prof. C R Sarma>

Introduction

- Program Execution

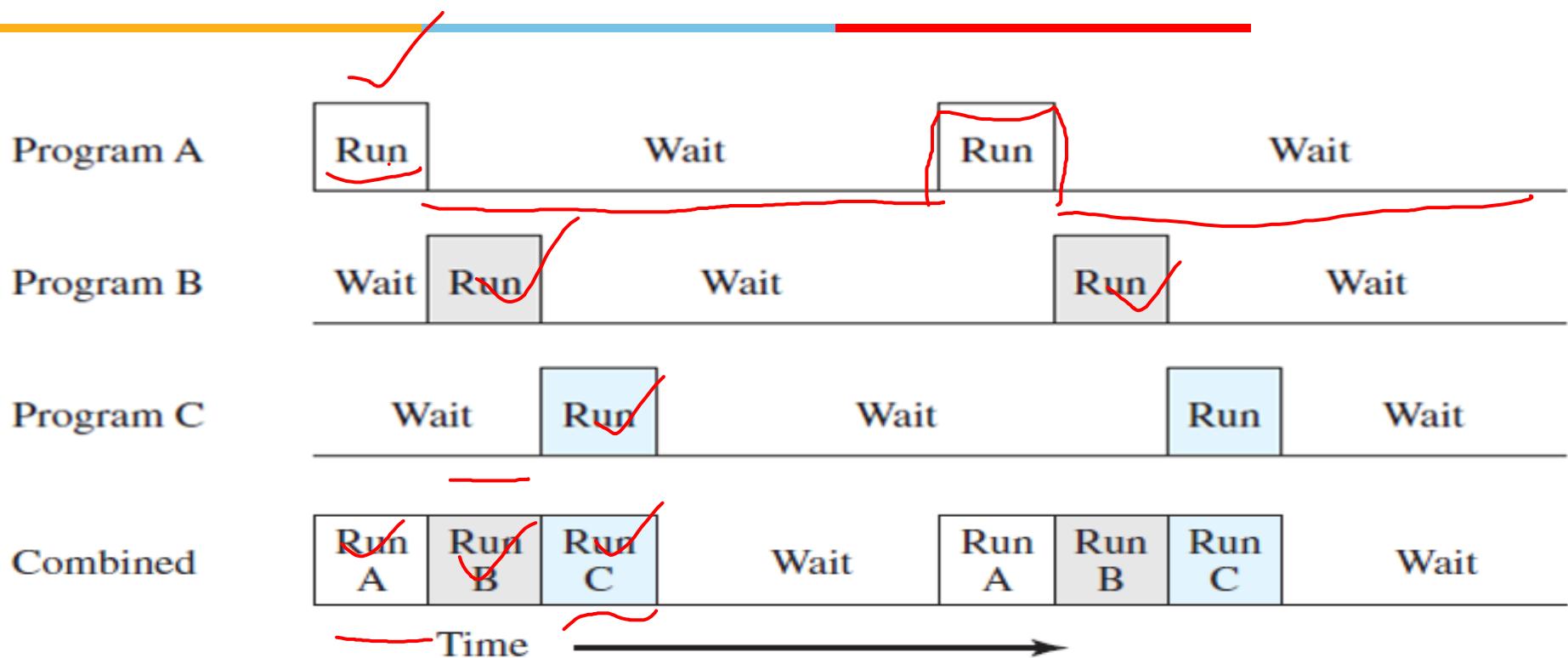


The compilation system.

Process

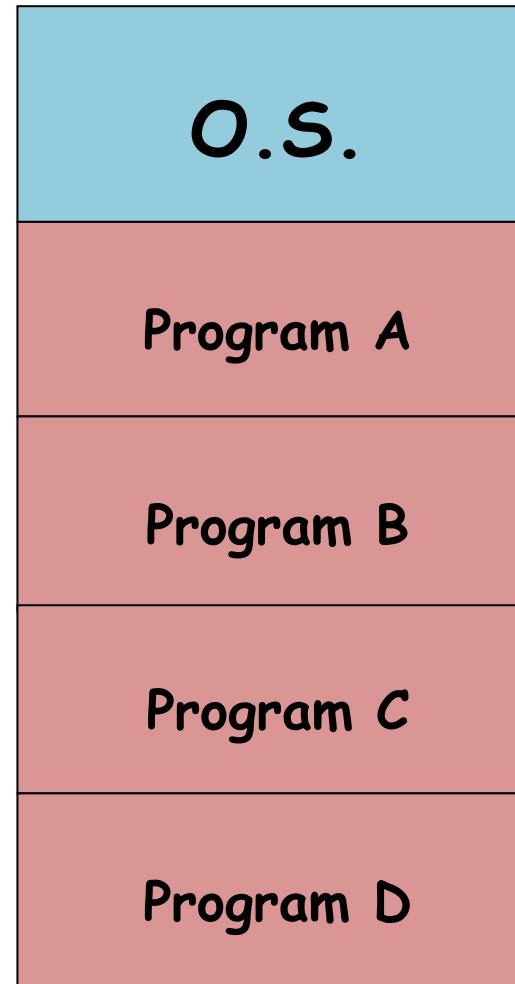
Job Queue

Multiprogramming vs Timesharing



Multiprogramming vs Timesharing

Memory layout of a
Multiprogramming system



Multiprogramming vs Timesharing



- Time sharing Logical extension of multiprogramming systems
- Also known as multitasking
- Each user program is given a time slot to execute in round robin fashion
- Pseudo parallelism
- Time shared system can run several programs at the same time, so it is also called multiprogramming system. But multiprogramming is not a time shared system.
- Allows many users to share the computer resources simultaneously

Process

- ✓ A program in execution
- ✓ Needs resources : CPU, memory, files, I/O devices
- ✓ A program becomes process when executable file loaded into memory
- ✓ Process execution must progress in sequential fashion
- ✓ word processor, a Web browser and an e-mail package are different processes.
- ✓ **Types:** System processes and user processes

Process in Memory

- The **text section**: comprises the executable code.
- The **data section**: stores **global and static variables**, allocated and initialized prior to executing program.
- The **heap**: is used for **dynamic memory allocation**.
- The **stack**: is used for **local variables**.
Space on the stack is reserved for local variables when they are declared and the space is freed up when the variables go out of scope. stack is also used for **function return values**.

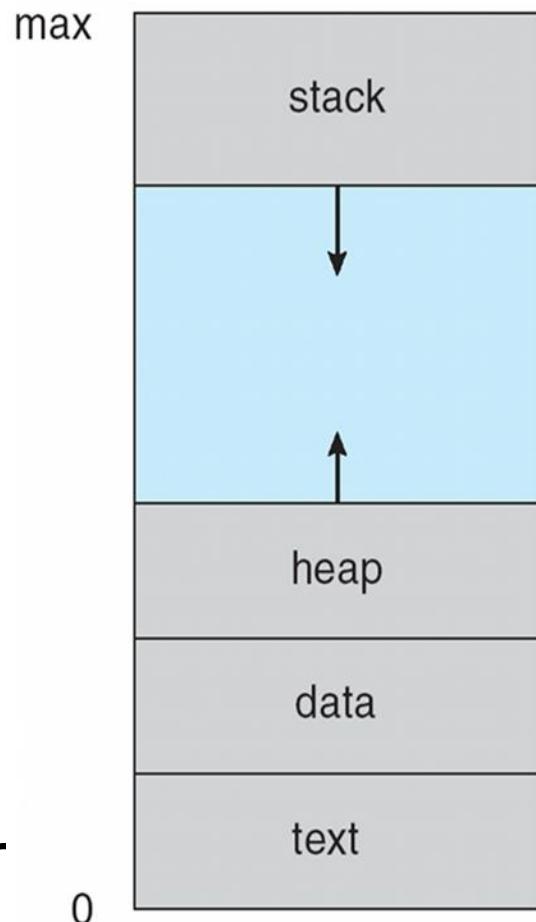
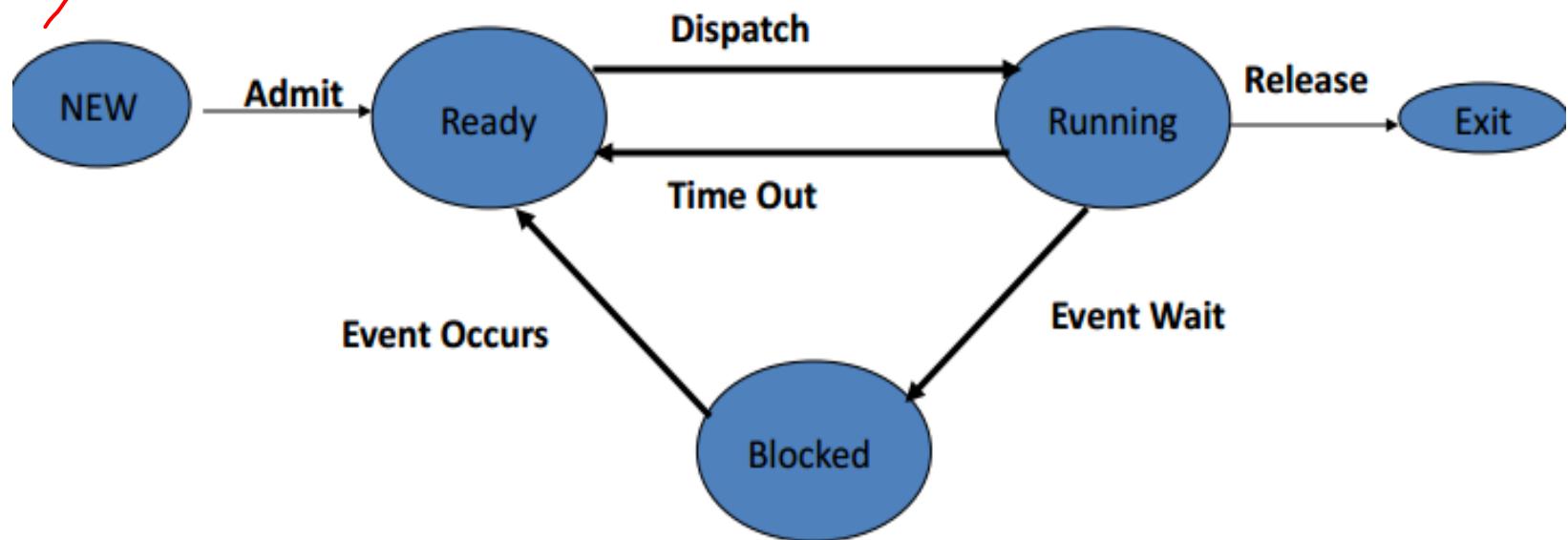


Diagram of Process State

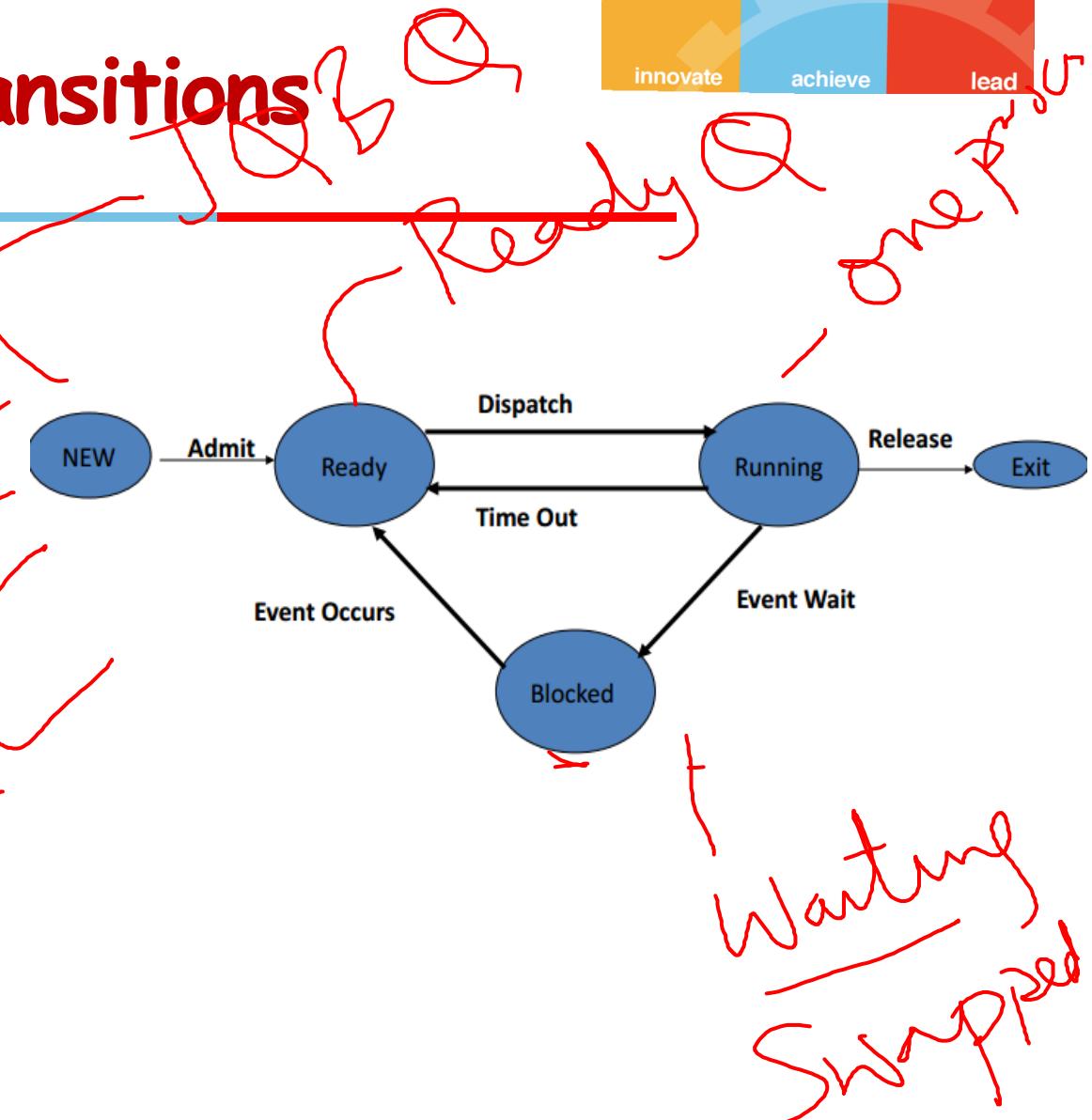


Process State

- Processes may be in one of 5 states
- **New** - The process is in the stage of being **created**.
- **Ready** - The process is **waiting to be assigned** to a processor
- **Running** - Instructions are being **executed** by the CPU.
- **Waiting/Blocked** - The process is **waiting for some resource** to become available. For example: keyboard input, disk access request etc.
- **Terminated** - The process has **finished** its task

Valid State Transitions

- Null to new
- New to ready
- Ready to running
- Running to exit
- Running to ready
- Running to blocked
- Blocked to ready
- Ready to exit
- Blocked to exit



Reasons for Process Creation

- New batch job
- Interactive logon
- Created by OS to provide a service
- Spawned by existing process

Reasons for Process Termination

- Normal Completion
 - Time Limit Exceeded
 - Memory Unavailable
 - Bounds Violation
 - Protection Error
 - Arithmetic Error
 - I/O Failure
 - Invalid and Privileged Instruction
 - Operator or OS Intervention
 - Parent termination
 - Parent request
-

Process Control Block

Each process has a PCB, which stores process-specific information like

- ✓ **Process Number** - Process Identifier
- ✓ **Process State** - Running, waiting, etc.
- ✓ **Pointer** - to parent, child (if any)
- ✓ **Priority** — *Low Number Hi Priority*
- ✓ **Program counter** - location of instruction next to execute
- ✓ **CPU registers** - contents of all process-centric registers
- ✓ **Accounting information** - CPU used, clock time elapsed since start, time limits

Process ID
State
Pointer
Priority
Program counter
CPU registers
I/O information
Accounting information
etc....

Process Context Switch

- To move from one process to another on a context switch, we have to
 - **save the context** of the current process
 - **select the next** process to run
 - **restore the context** of this new process.
- **Context of a process**

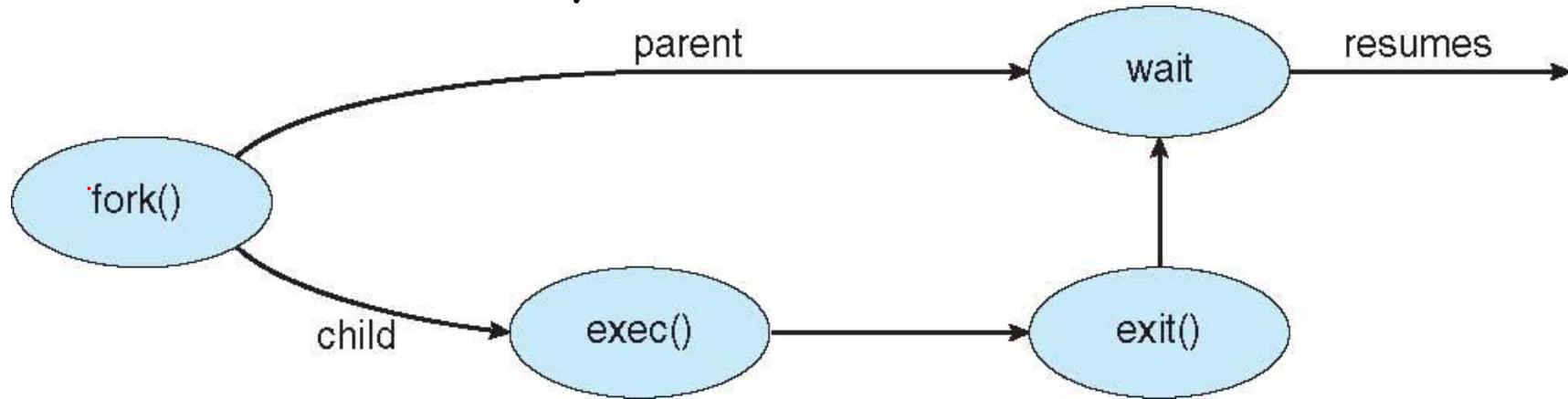
Program Counter, Stack Pointer, Registers, Code + Data + Stack (also called Address Space), Other state information maintained by the OS for the process (open files, scheduling info, I/O devices being used etc.)

Operations on Processes

- Process Creation
- Process Termination
- Process Preemption
- Blocking

Operations on Processes - Process Creation

- Created using system calls
 - Example : fork or spawn
 - Parent Vs Child
 - Each process stores PID → Integer, and PPID
- There are two options for the parent process after creating the child:
 - ✓ Wait for the child process to terminate
 - ✓ Run concurrently with the child



Contd...

- Two possibilities for the address space of the child relative to the parent:
 - ✓ The child may be an **exact duplicate of the parent**
 - ✓ The child process may have **a new program loaded into its address space**

Operations on Processes – Process Preemption and Blocking

- Preemption : temporarily interrupting the running process
- Process blocking: Running process goes for an I/O

Operations on Processes

- Processes may request their own termination by making the `exit()` system call
 - When a process terminates, all of its system resources are freed up, open files flushed and closed, etc.
 - The process termination status and execution times are returned to the parent
 - Orphan process
 - Zombie process.
- Parent gets killed / Terminated*
- Parent exits w/o waiting for child*

fork() system call

- To create new process
- Parent process : process which invoked fork()
- A child process uses the same pc(program counter), same CPU registers, same open files which are used in the parent process.
- fork() takes no parameters and returns an integer value
 - **Negative Value:** creation of a child process was unsuccessful.
 - **Zero:** Returned to the newly created child process.
 - **Positive value:** Returned to parent or caller. The value contains process ID of newly created child process.

Example 1

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    fork();
    printf ("hello\n");
    return (0);
}
```

```
$ gedit fork.c
$ gcc -o fork_EX1 fork_Ex1.c
$ ./fork_Ex1
hello
hello
$
```

Example 2

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int x;
    x = fork();
    if (x == 0 )
        printf ("Child Process :%d", x);
    else
        printf ("I am parent : %d", x);
    return (0);
}
```

exec

```
$ gcc -o fork_Ex3 fork_Ex3.c
$ ./fork_Ex3
```

Child Process : 0

I am Parent : 1234

\$

Example 3

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
```

```
int main()
```

```
{
```

```
fork();
```

```
fork();
```

```
fork();
```

```
printf("Hello");
```

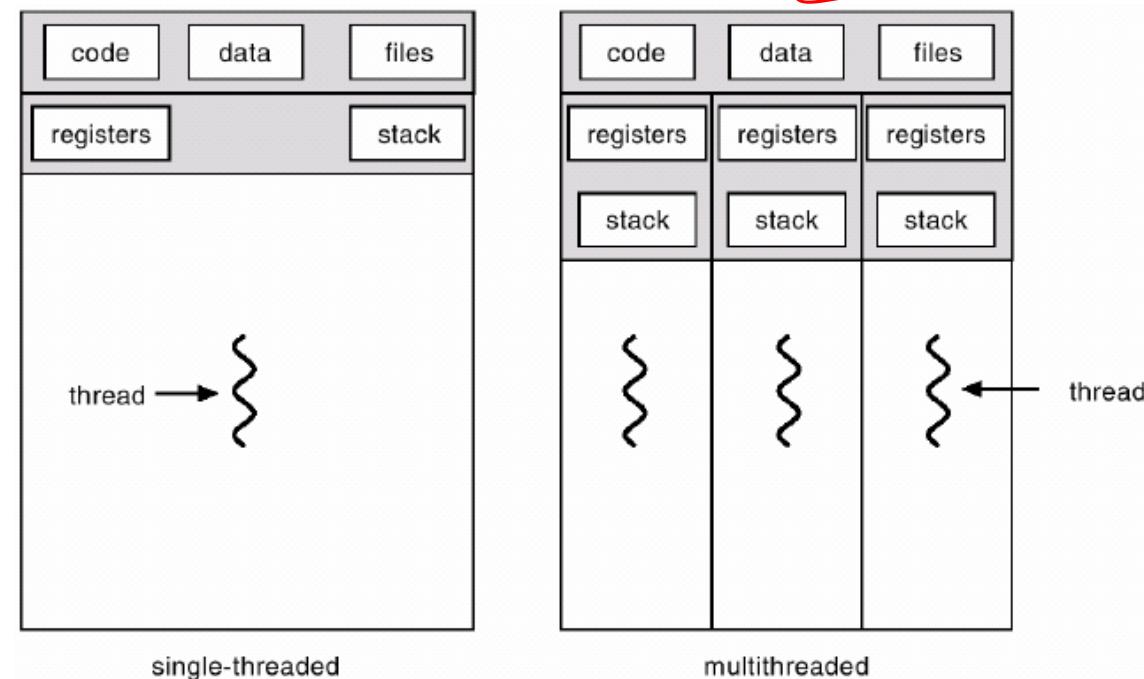
```
return (0);
```

```
}
```

```
SS Hello
```

Threads

- A thread is a **single sequence stream** within in a process
- A thread has a **program counter (PC)**, a **register set**, and **a stack space**.



Process Vs. Threads

Process

1. Considered Heavy weight
2. Process creation is costly in terms of resources
3. Program executing as process are relatively slow
4. Process cannot access the memory area belonging to another process
5. Process switching is time consuming
6. One process can contain several threads

Threads

1. Considered light weight
2. Thread creation is very economical
3. Program executing as threads are comparatively faster
4. Thread can access the memory area belonging to another thread within same process
5. Thread switching is faster
6. One thread can belong to exactly one process

Process vs. Thread

Similarities

- Like processes threads share CPU and **only one thread active** (running) at a time.
- Like processes, threads within a processes, **execute sequentially**.
- Like processes, thread can **create children**.
- And like process, only when **one thread is blocked**, another thread can run.

Process vs. Thread contd.

Differences

- Unlike processes, threads are not independent of one another.
- Unlike processes, all threads can access every address in the task .
- Unlike processes, threads are created to assist one other.

Types of Threads

Threads are implemented in following two ways

- **User Level Threads** -- User managed threads.
- **Kernel Level Threads** -- Operating System managed threads.



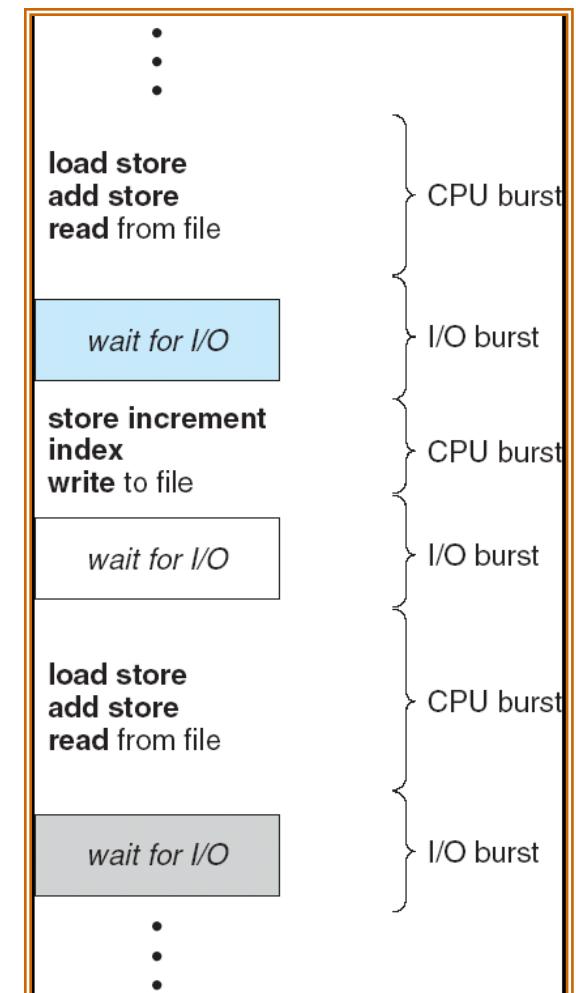
Process Scheduling

Scheduling: Basic Concepts

- **Scheduling:** The assignment of CPU to a process
 - **Preemption:** Removal of CPU control from a process
 - **Context Switch:** Switching the CPU to another process
 - Save the state of old process
 - Load the saved state of new process
 - **PCB** is used to save the information /context of a process
-

Scheduling: Basic Concepts...

- Maximum CPU utilization obtained with multiprogramming
- Process execution consists of a cycle of CPU execution and I/O wait
 - CPU-I/O Burst Cycle
- CPU burst distribution



Process scheduling

- **Job queue** - set of all processes in the system
- **Ready queue** - set of all processes residing in main memory, ready and waiting to execute
- **Device queues** - set of processes waiting for an I/O device
- Processes migrate among the various queues

Types of Scheduler

- Long-term scheduler
- Medium-term scheduler
- Short-term scheduler
- I/O scheduler

Long Term Schedulers

- Also known as Job Scheduler
- Decides which processes should be brought into the ready queue.
- The long-term scheduler controls the **degree of multiprogramming**
 - Main Goal : to have good mix of CPU bound and I/O bound processes
- Executes infrequently, maybe only when process leaves system
- Does not exist on most modern timesharing systems

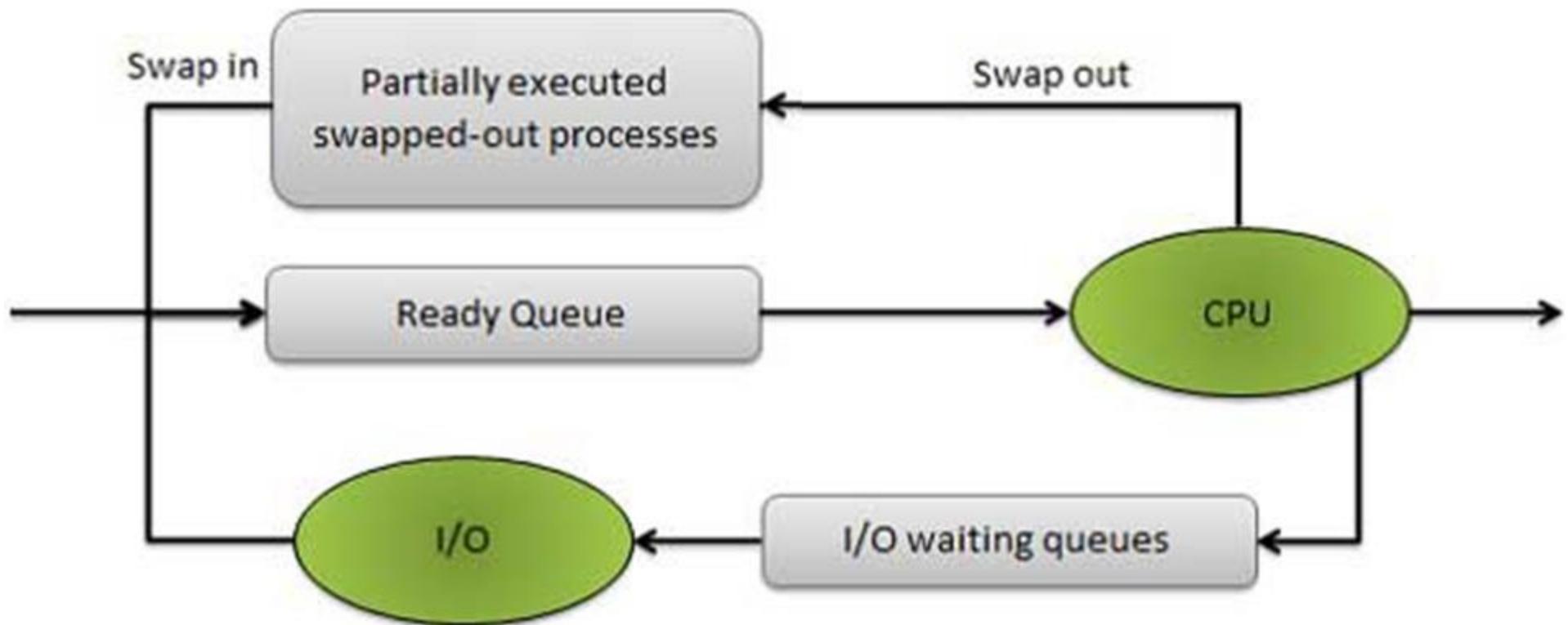
Short Term Scheduler

- Also known as CPU Scheduler or dispatcher
- selects from among the processes that are ready to execute and allocates the CPU to one of them
- Executes frequently
- Runs whenever
 - Process is created and brought in to ready queue
 - Process is terminated
 - Process switches from running to blocked (wait)
 - When interrupt occurs

Short - Term Scheduler...

- Main Goals:
 - Minimize response time
 - Maximize throughput
 - Minimize overhead such as OS overhead, context switching etc.
 - Efficient use of resources
 - Fairness - Share CPU and other resources in an equitable fashion

Addition of Medium Term Scheduling



Contd...

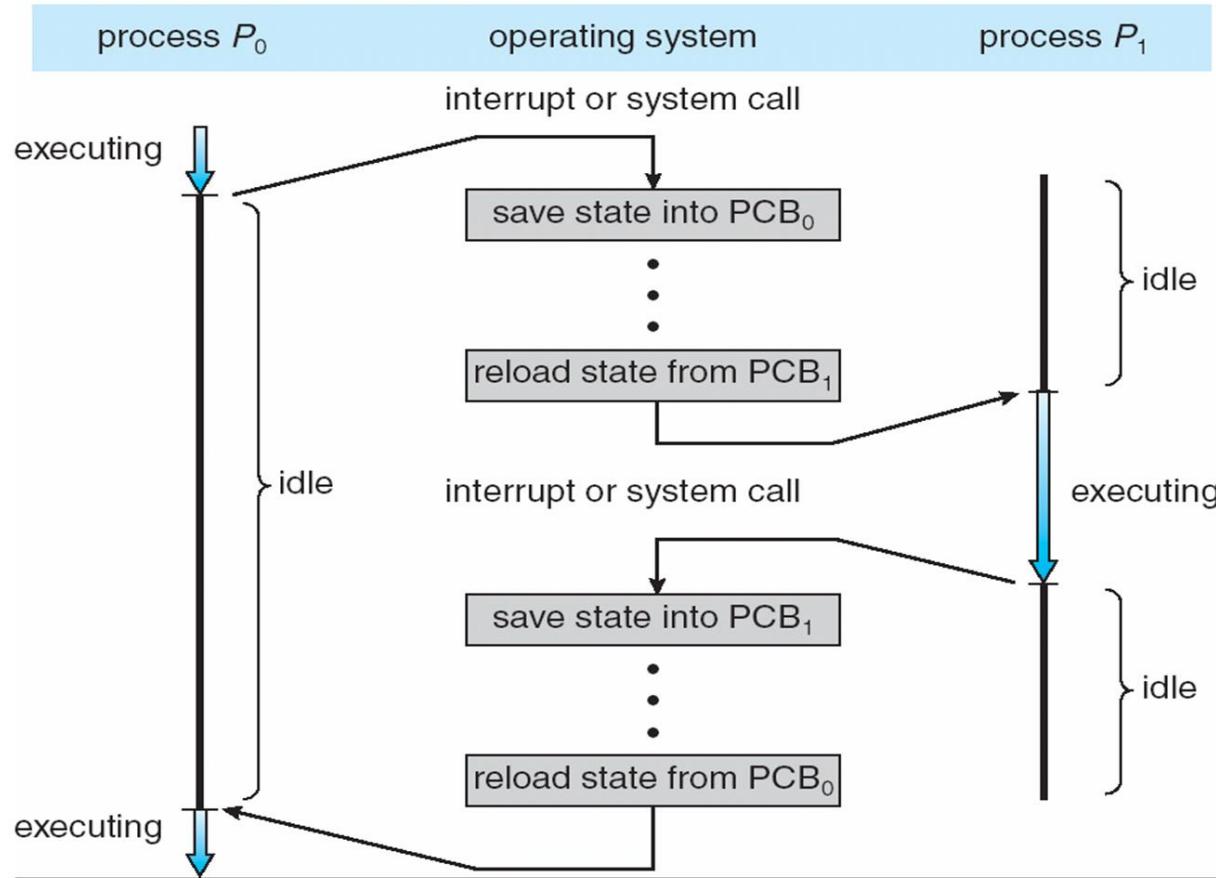
The mid-term scheduler may decide to swap out

1. a process which has not been active for some time
2. a process which has a low priority
3. a process which is page faulting frequently
4. a process which is taking up a large amount of memory in order to free up main memory for other processes, swapping the process back in later when more memory is available,
5. when the process has been unblocked and is no longer waiting for a resource.

I/O Scheduling

The decision as to which process's pending I/O request shall be handled by an available I/O device

CPU Switch From Process to Process







OPERATING SYSTEM SESSION 10

Prepared By: Dr. Lucy J. Gudino
Instructor: <Prof. C R Sarma>



BITS Pilani
Pilani Campus



Process Scheduling-2

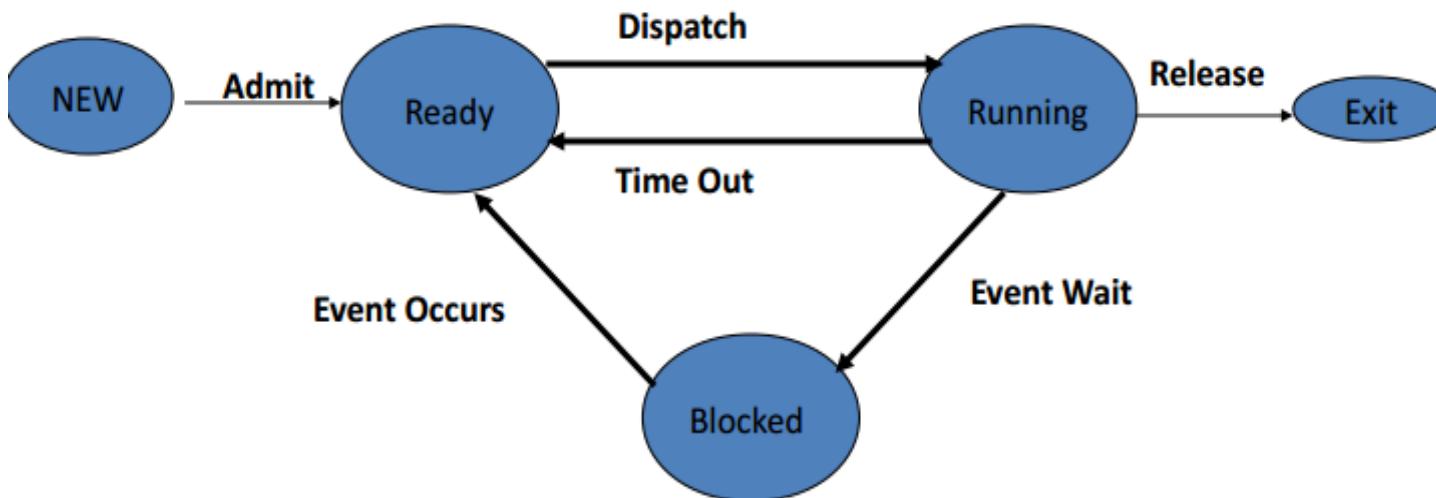
Scheduling Algorithms

I/O Scheduling

The decision as to which process's pending I/O request shall be handled by an available I/O device

CPU Scheduler

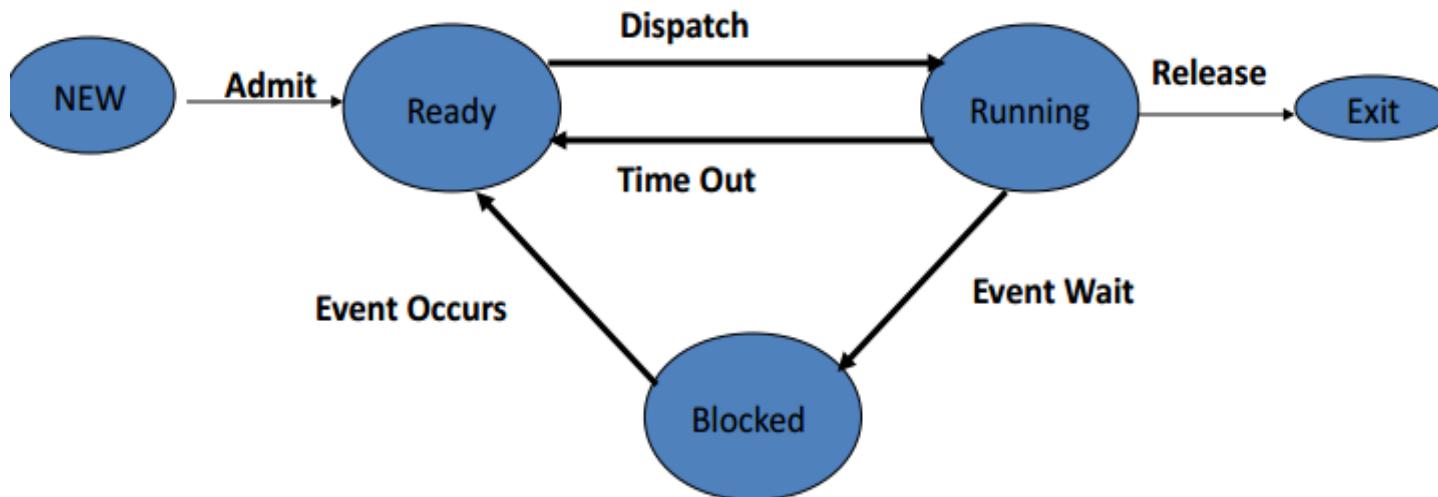
- CPU scheduling decisions may take place when a process:
 - 1. Switches from running to waiting(blocked) state
 - 2. Switches from running to ready state
 - 3. Switches from waiting (blocked) to ready state
 - 4. Switches from running to terminate state



CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
 - 1. Switches from running to waiting (blocked) state
 - 2. Switches from running to ready state
 - 3. Switches from waiting (blocked) to ready
 - 4. Switches from running to terminate state

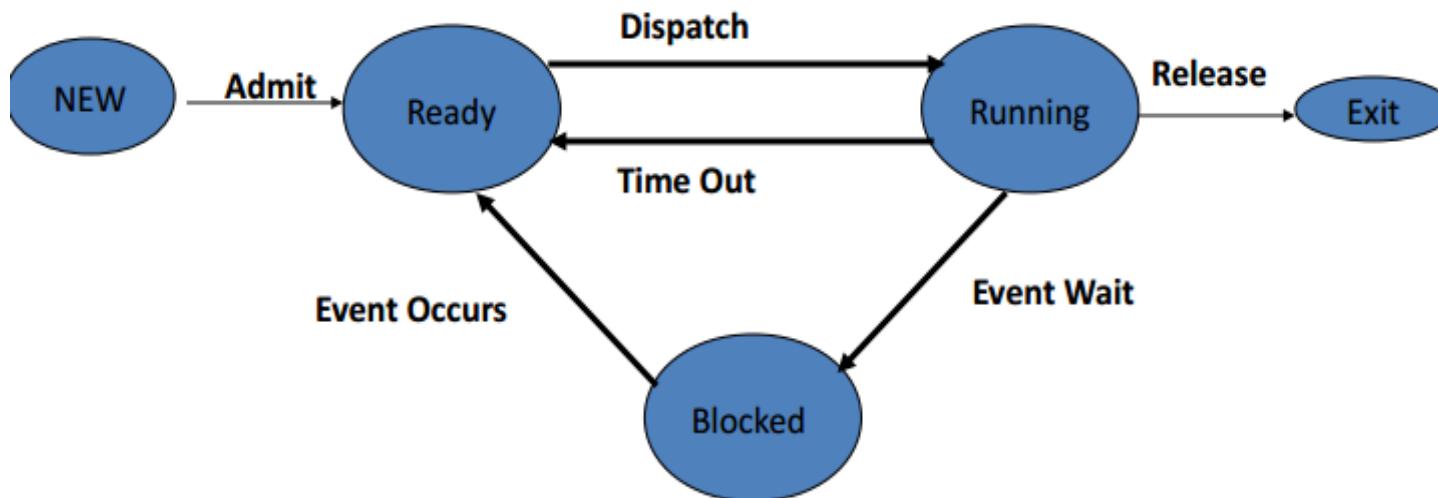
No choice! New process must be selected



CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
 - 1. Switches from running to waiting state
 - 2. Switches from running to ready state
 - 3. Switches from waiting to ready
 - 4. Terminates

There is a choice!



Preemptive and Non-preemptive scheduling

- Non-preemptive scheduling
 - A new process is selected to run either
 - when a process terminates or
 - when an explicit system request causes a wait state
 - (e.g., I/O or wait for child)
- Preemptive scheduling
 - New process selected to run when
 - An interrupt occurs
 - When new processes become ready

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler;
- Function of Dispatcher :
 - switching context
 - switching to user mode
 - jumping to the proper location in the user program to restart that program

Dispatch latency - time it takes for the dispatcher to stop one process and start another running

Good Scheduling Algorithm

- Be fair : don't allow process to starve
- Be efficient : maximize CPU utilization
- Maximize throughput
 - Throughput : number of processes completed in unit time
- Minimize response time
 - Response time : time measured from process creation to the time of first output (response).
- Minimize waiting time
 - Waiting time : the amount of time a process has been waiting in the ready queue

Good Scheduling Algorithm...

- Be predictable : a given job should take about the same amount of time to run when run multiple times
- Minimize overhead : Keep scheduling time and context switch time at a minimum
- Maximize resource use : favor processes that will use underutilized resources
- Avoid indefinite postponement : every process should get a chance to run eventually
- Enforce priorities
- Degrade gracefully : as the system becomes more heavily loaded, performance should deteriorate gradually, not abruptly

Performance Metrics

- CPU utilization - keep the CPU as busy as possible.
CPU utilization vary from 0 to 100. It varies from 40 (lightly loaded) to 90 (heavily loaded)
 - Throughput - Number of processes that complete their execution per time unit.
 - Turnaround time - Amount of time to execute a particular process (interval from time of submission to time of completion of process).
 - Response time
 - Waiting time
-

Scheduling Algorithms

1. First Come First Served (FCFS)
2. Shortest Job First (SJF)
3. Shortest Remaining Time First (SRTF)
4. Priority Scheduling
5. Round Robin Scheduling
6. Multi level Feedback Queue



BITS Pilani
Pilani Campus

Problems



Scheduling Algorithm - FCFS

- Simplest scheduling algorithm.
 - Non-preemptive type of scheduling.
 - Process which requests the CPU first is allocated the CPU first.
 - The implementation of FCFS is managed with a FIFO queue.
-

FCFS: Example 1



- Draw Gantt chart
- Compute the average wait time, TAT and RT for processes

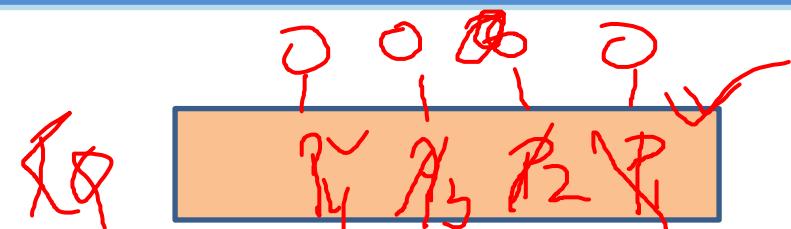
Process	AT	BT	FT	TAT	WT	RT
P1	0	7	7	7	0	0
P2	0	3	10	10	7	7
P3	0	4	14	14	10	10
P4	0	6	20	20	14	14

TC

AVG

P1 | P2 | P3 | P4

0 7 10 14 20



FCFS: Example 2

- Draw Gantt chart
- Compute the average wait time, TAT and RT for processes

Process	AT	BT	FT	TAT	WT	RT
P1	0	7	7	7	0	0
P2	8	3	20	12	9	17
P3	3	4	11	8	4	7
P4	5	6	17	12	6	11

$55/4$ $59/4$ $19/4$ $35/4$ RT



P2 P4 P3 P1

FCFS: Example 3

- Draw Gantt chart
- Compute the average wait time, TAT and RT for processes

Process	AT	BT	FT	TAT	WT	RT
P1	0	2	2	2	0	0
P2	3	1	4	1	0	0
P3	5	5	10	5	0	0



FCFS Drawbacks

- A process that does not perform any I/O will monopolize the processor (Convoy Effect).
- Favors CPU-bound processes:
 - I/O-bound processes have to wait until CPU-bound process completes.
 - They may have to wait even when their I/O are completed (poor device utilization).

SJF: Shortest Job First

- The job with the shortest next CPU burst time is selected
- Associates with each process the length of its **next CPU burst**.
- Use these lengths to schedule the process with the shortest time.
- If two processes have the same **next CPU burst**, **FCFS** is used to break the tie.
- also known as “shortest next CPU burst algorithm”

SJF: Shortest Job First → Types

- Two schemes:
 - nonpreemptive - once CPU given to the process, it cannot be preempted until completes its CPU burst
 - preemptive - if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal - gives minimum average waiting time for a given set of processes

SJF (non-preemptive): Example 4

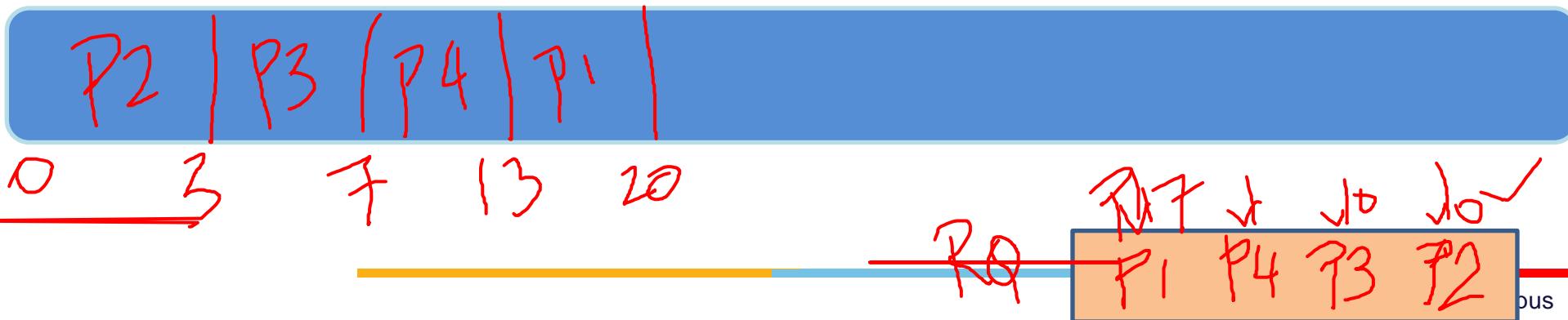


Draw Gantt chart

Compute the average wait time, TAT and RT for processes

Process	AT	BT	FT	TAT	WT	RT
P1	0	7	20 P10	20	13	13
P2	0	3	3	3	0	0
P3	0	4	7	7	3	3
P4	0	6	13	13	7	7

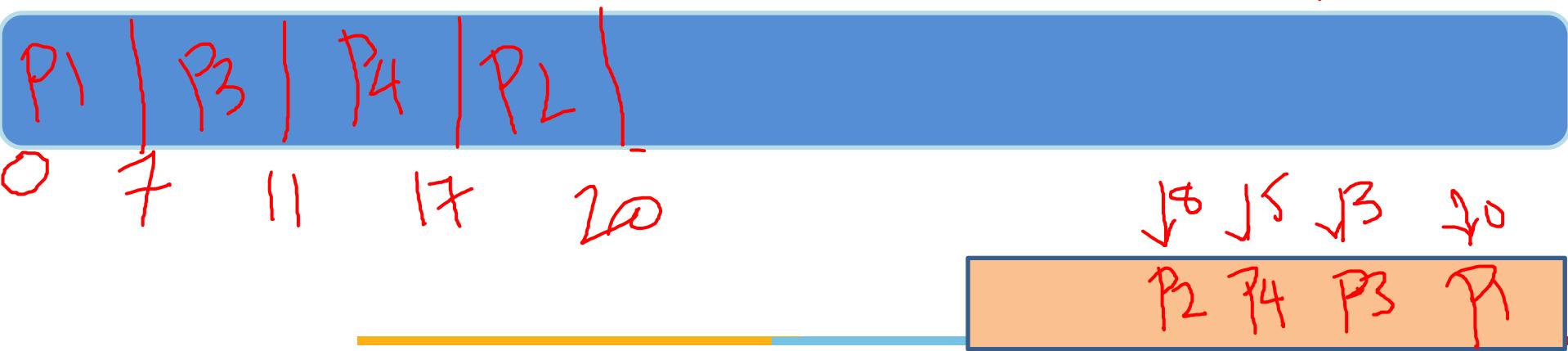
Σ34ms



SJF (non-preemptive): Example 5

- Draw Gantt chart
- Compute the average wait time, TAT and RT for processes

Process	AT	BT	FT	TAT	WT	RT
P1	0	7	7	0.7	0	0
P2	8	3	11	12	9	17
P3	3	4	11	8	4	7
P4	5	6	17	12	6	11



SJF (Preemptive) / SRTF: Example 6



- Draw Gantt chart
- Compute the average wait time, TAT and RT for processes

Process	AT	BT	FT	TAT	WT	RT
P1	0	7	9	9	02	0
P2	8	3	12	4	1	5
P3	3	2	5	2	0	3
P4	5	6	18	13	6	12



SJF...

- Optimal: Shortest average wait time
- It's unfair !!
 - Continuous stream of short jobs will starve long job
- SJF scheduling is used frequently in long-term scheduling.
 - user needs to estimate the process time
- Short term Scheduling: Need to know the execution time of a process
 - May have an estimate, to predict next CPU burst
- Jobs are organized in order of estimated completion time

SJF Drawbacks

- Possibility of starvation for longer processes as long as there is a steady supply of shorter processes.
- CPU bound process gets lower priority but a process doing no I/O could still monopolize the CPU if it is the first one to enter the system.

Note: SJF implicitly incorporates priorities: shortest jobs are given preferences.

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer \equiv highest priority)
 - Preemptive
 - nonpreemptive
- SJF is a special case of priority scheduling where priority is the predicted next CPU burst time
- Problem \equiv Starvation - low priority processes may never execute
- Solution \equiv Aging - as time progresses increase the priority of the process

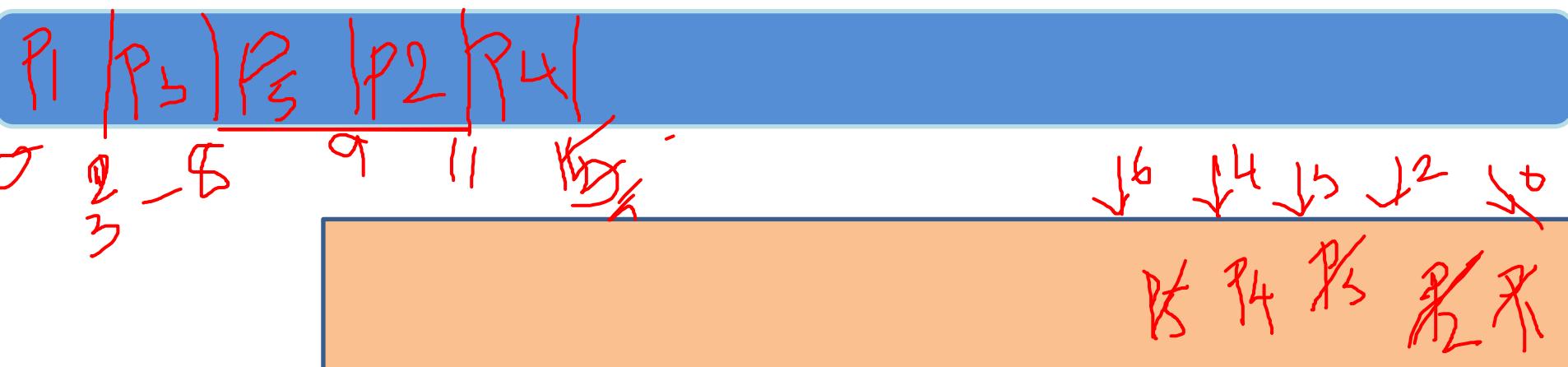
Note: Some use larger integer \equiv highest priority)

Priority (non-preemptive): Example 8



- Draw Gantt chart (Lower Number Higher priority,)
- Compute the average wait time, TAT and RT for processes

Process	AT	BT	Pri	FT	TAT	WT	RT
P1	0	3	5	3	3	0	0
P2	2	2	3	11	9	7	9
P3	3	5	2	5	5	0	0
P4	4	4	4	15	11	7	11
P5	6	1	1	9	3	2	8



Priority (preemptive): Example 8



- Draw Gantt chart
- Compute the average wait time, TAT and RT for processes

Process	AT	BT	Pri	FT	TAT	WT	RT
P1	0	3	5	15	15	12	0
P2	2	2	3	10	8	6	0
P3	3	5	2	9	6	3	0
P4	4	4	4	14	10	6	6
P5	6	1	1	7	1	0	0



Sequence: P1, P2, P3, P5, P3, P2, P4, P1

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer \equiv highest priority)
 - Preemptive
 - nonpreemptive
- SJF is a special case of priority scheduling where priority is the predicted next CPU burst time
- Problem \equiv Starvation - low priority processes may never execute
- Solution \equiv Aging - as time progresses increase the priority of the process

Note: Some use larger integer \equiv highest priority

Priority (non-preemptive): Example 7



- Draw Gantt chart (Lower Number Higher priority,)
- Compute the average wait time, TAT and RT for processes

Process	AT	BT	Pri	FT	TAT	WT	RT
P1	0	3	5				
P2	2	2	3				
P3	3	5	2				
P4	4	4	4				
P5	6	1	1				

Priority (preemptive): Example 8



- Draw Gantt chart
- Compute the average wait time, TAT and RT for processes

Process	AT	BT	Pri	FT	TAT	WT	RT
P1	0	3	5				
P2	2	2	3				
P3	3	5	2				
P4	4	4	4				
P5	6	1	1				

Round Robin (RR)

- Each process gets a small unit of CPU time (time quantum), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q , then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance
 - q large \Rightarrow FCFS
 - q small \Rightarrow q must be large with respect to context switch, otherwise overhead is too high

Round Robin: Example 9

- Draw Gantt chart , Time Quantum = 4
- Compute the average wait time, TAT and RT for processes

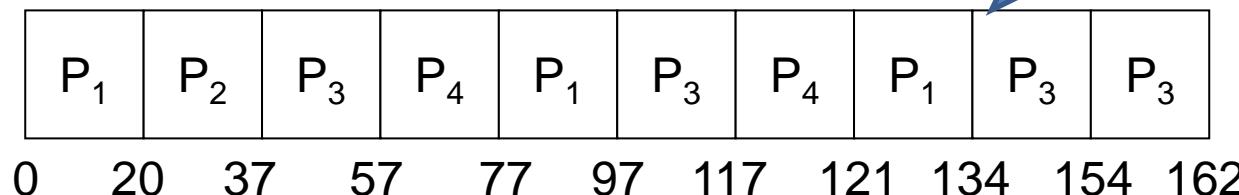
Process	AT	BT	FT	TAT	WT	RT
P1	5	5				
P2	4	6				
P3	3	7				
P4	1	9				
✓ P5	2	2				
✓ P6	6	3				



Example 10 : RR with Time Quantum = 20

<u>Process</u>	<u>Burst Time</u>
P_1	53
P_2	17
P_3	68
P_4	24

The Gantt chart is:

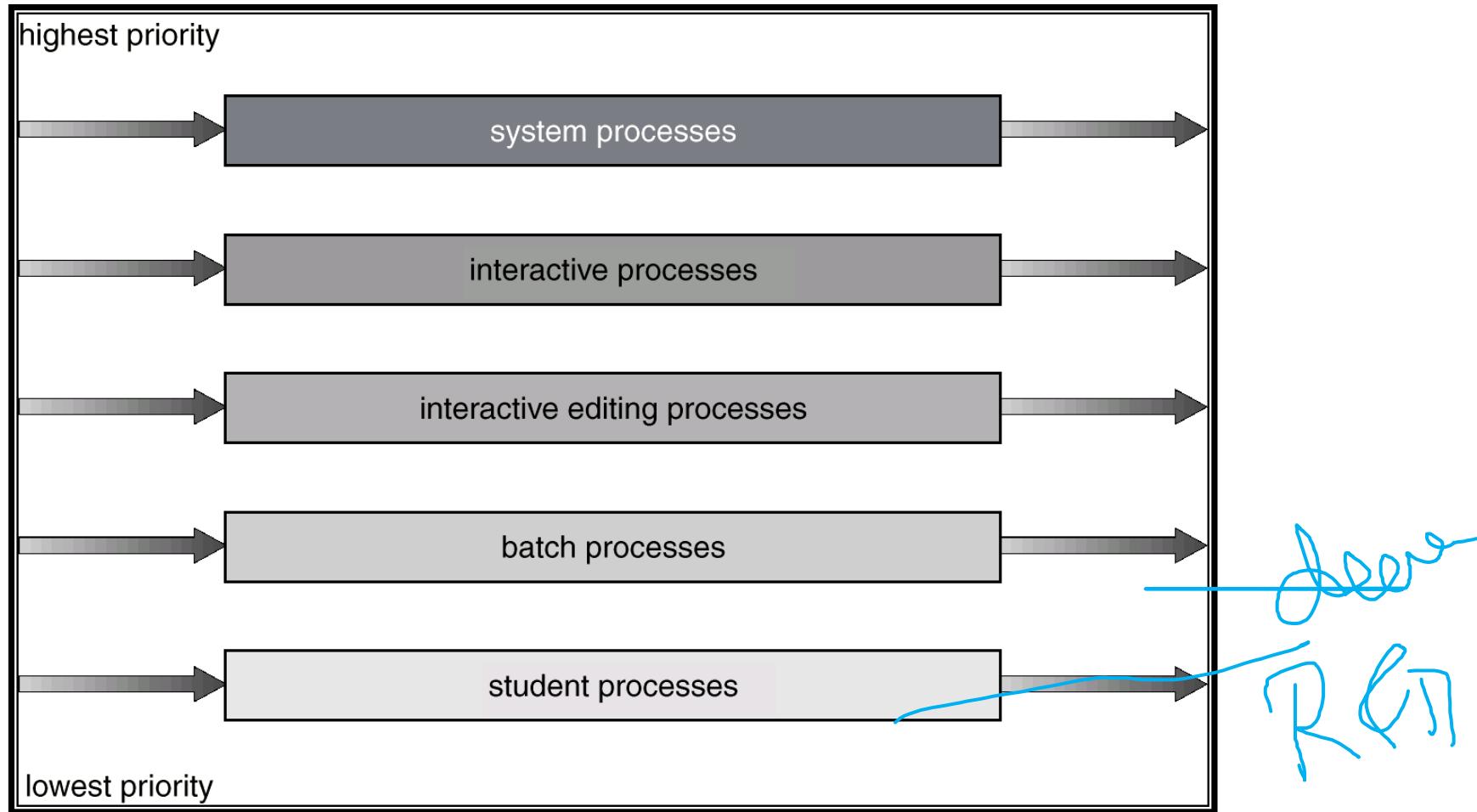


Typically, higher average turnaround than SJF, but better response

Multilevel Queuing

- Process classification based on response-time requirements and scheduling needs
 - Foreground processes ✓
 - background processes ✓
- Foreground processes may have priority (externally defined) over background processes
- Multilevel queue scheduling algorithm
 - Partition the ready queue into several separate queues
 - Each queue has its own scheduling algorithm
 - Example: foreground and background processes.
 - The foreground queue scheduled by an RR algorithm
 - background queue is scheduled by an FCFS algorithm.
- Process assignment to queue based on
 - Memory size, priority of process or process type

Multilevel Queue Scheduling



Multilevel Feedback Queue

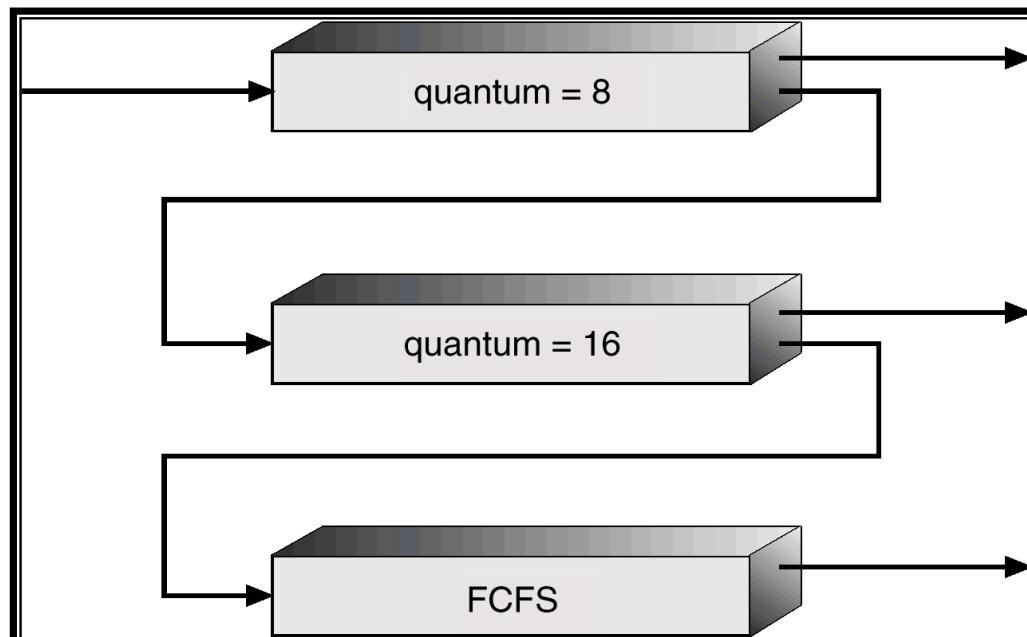
- Disadvantage of Multilevel queue : Inflexible
- **Multilevel Feedback Queue** : A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
 - number of queues
 - scheduling algorithms for each queue
 - method used to determine when to upgrade a process
 - method used to determine when to demote a process
 - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:

Q_0 – RR - time quantum 8 ms
 Q_1 – RR- time quantum 16ms
 Q_2 – FCFS

- Scheduling
 - A new process enters queue Q_0 . When it gains CPU, process receives 8 milliseconds. If it does not finish in 8 milliseconds, process is moved to queue Q_1 .
 - At Q_1 process receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 .





BITS Pilani
Pilani Campus

COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS

SESSION 11

Prepared: Dr. Lucy J. Gudino

Instructor: Prof. C R Sarma

WILP & Department of CS & IS

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Scheduling Algorithms : FCFS, SJF, SRTF Priority and Round-Robin Scheduling	T2

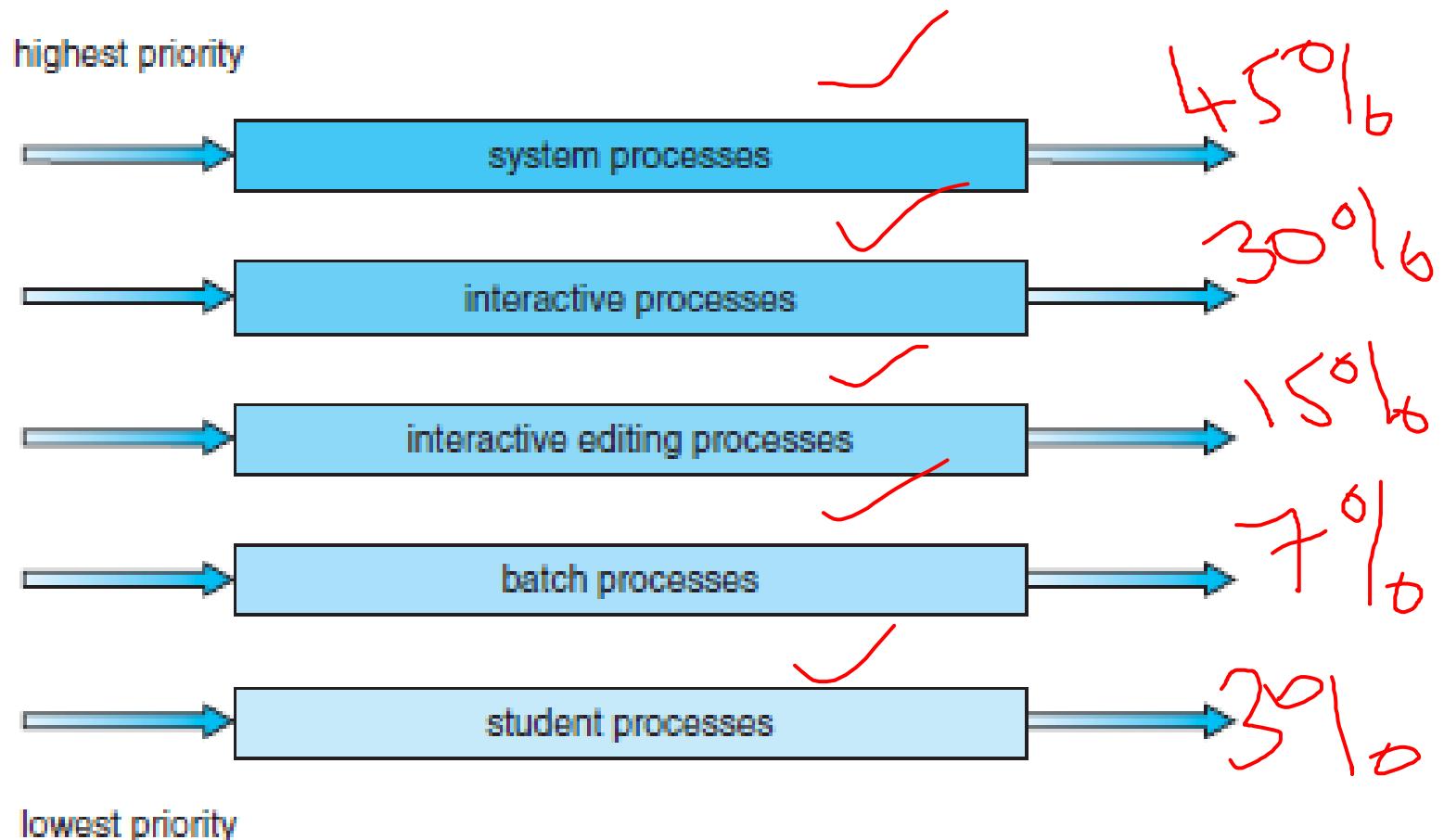
Today's Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Scheduling algorithms (RR and Multilevel scheduling)• Scheduling with I/O• Process Coordination• Deadlock (Introduction)	T2

Multilevel Queuing

- Process classification based on response-time requirements and scheduling needs
 - Foreground processes ✓ *interactive*
 - background processes ✓ *batch*
- Foreground processes may have priority (externally defined) over background processes
- Multilevel queue scheduling algorithm
 - Partition the ready queue into several separate queues
 - Each queue has its own scheduling algorithm
 - Example: foreground and background processes.
 - The foreground queue scheduled by an RR algorithm
 - background queue is scheduled by an FCFS algorithm.
- Process assignment to queue based on
 - Memory size, priority of process or process type

Multilevel Queue Scheduling



aging



Multilevel Feedback Queue

- Disadvantage of Multilevel queue : Inflexible
- **Multilevel Feedback Queue** : A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
 - number of queues
 - scheduling algorithms for each queue ✓
 - method used to determine when to upgrade a process
 - method used to determine when to demote a process
 - method used to determine which queue a process will enter when that process needs service

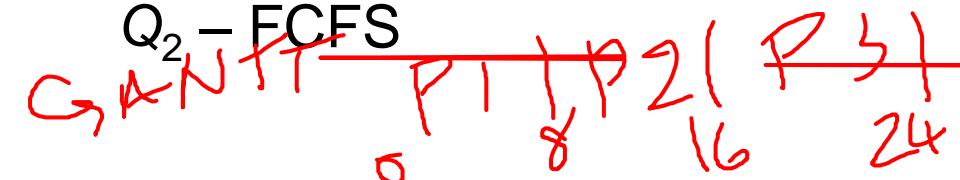
Example of Multilevel Feedback Queue

- Three queues:

Q_0 – RR - time quantum 8 ms

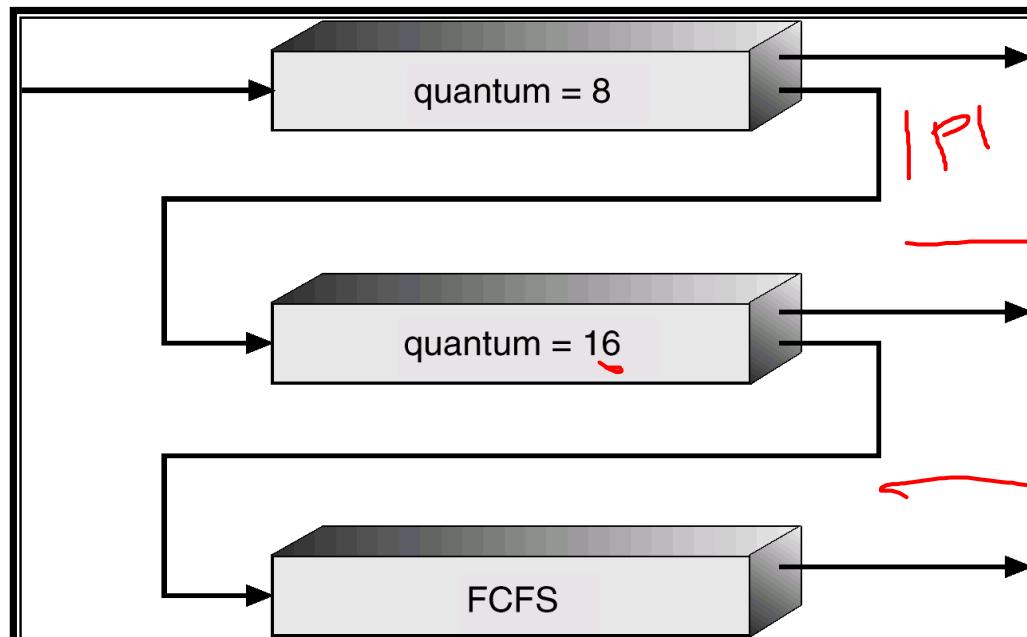
Q_1 – RR- time quantum ~~16ms~~

Q₂ - FCFS



~~Scheduling~~

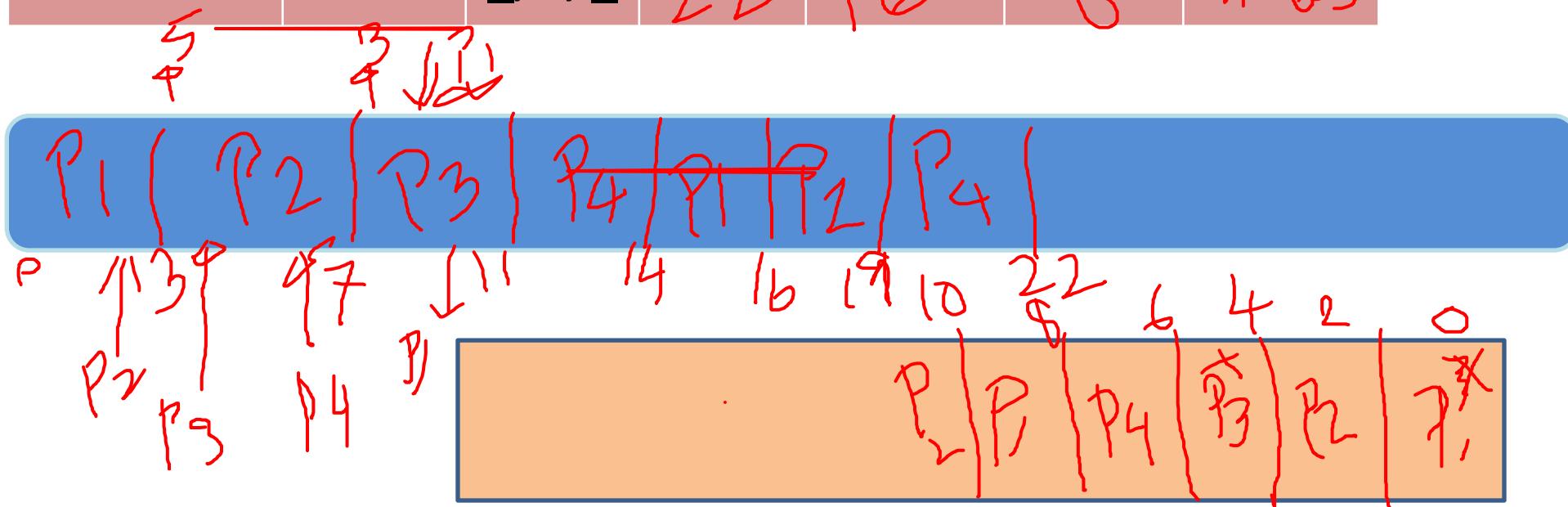
- A new process enters queue Q_0 . When it gains CPU, process receives 8 milliseconds. If it does not finish in 8 milliseconds, process is moved to queue Q_1 .



- At Q_1 , process receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 .

FCFS with I/O

Process	AT	BT & I/O	FT	TAT	WT	RT
X P1	0	<u>3, 5, 2</u>	16	16	6	160
X P2	2	<u>4, 3, 3</u>	19	17	7	3
X P3	4	<u>4</u>	11	7	3	7-4=3
P4	6	<u>3, 2, 3</u>	22	16	8	11-6=5



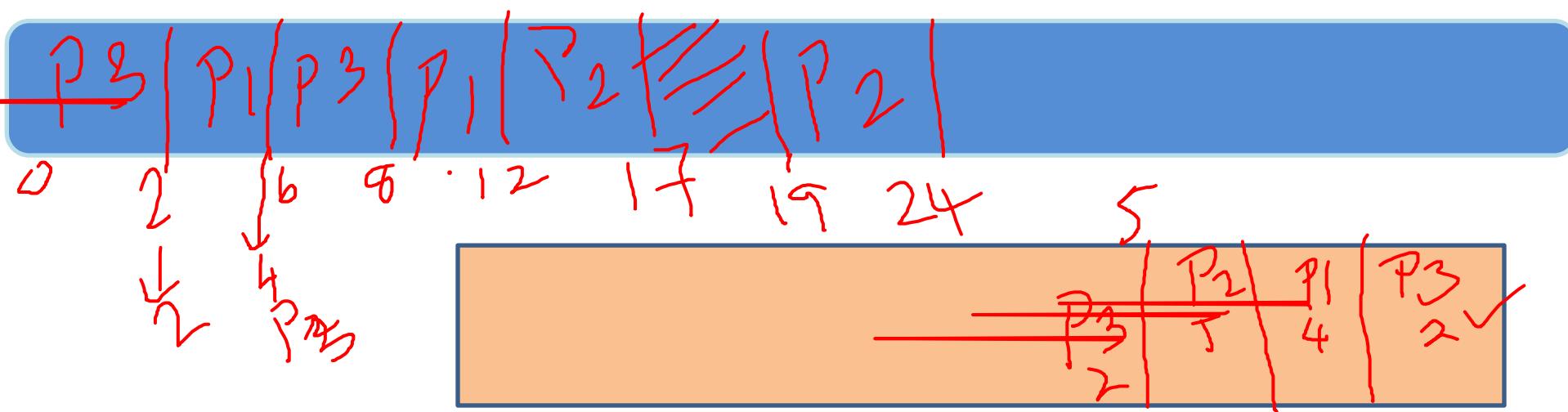
SJF (Non Pre-emptive) with IO

innovate

achieve

lead

Process	AT	CPU- BT & I/O BT	Total BT	FT	TAT	WT	RT
P1 X	0	(4 - 2 - 4)	10	12	12	2	2
P2 *	0	(5 - 2 - 5)	12	24	24	12	12
P3 *	0	(2 - 2 - 2)	6	8	8	2	0



SJF (Pre-emptive) with IO

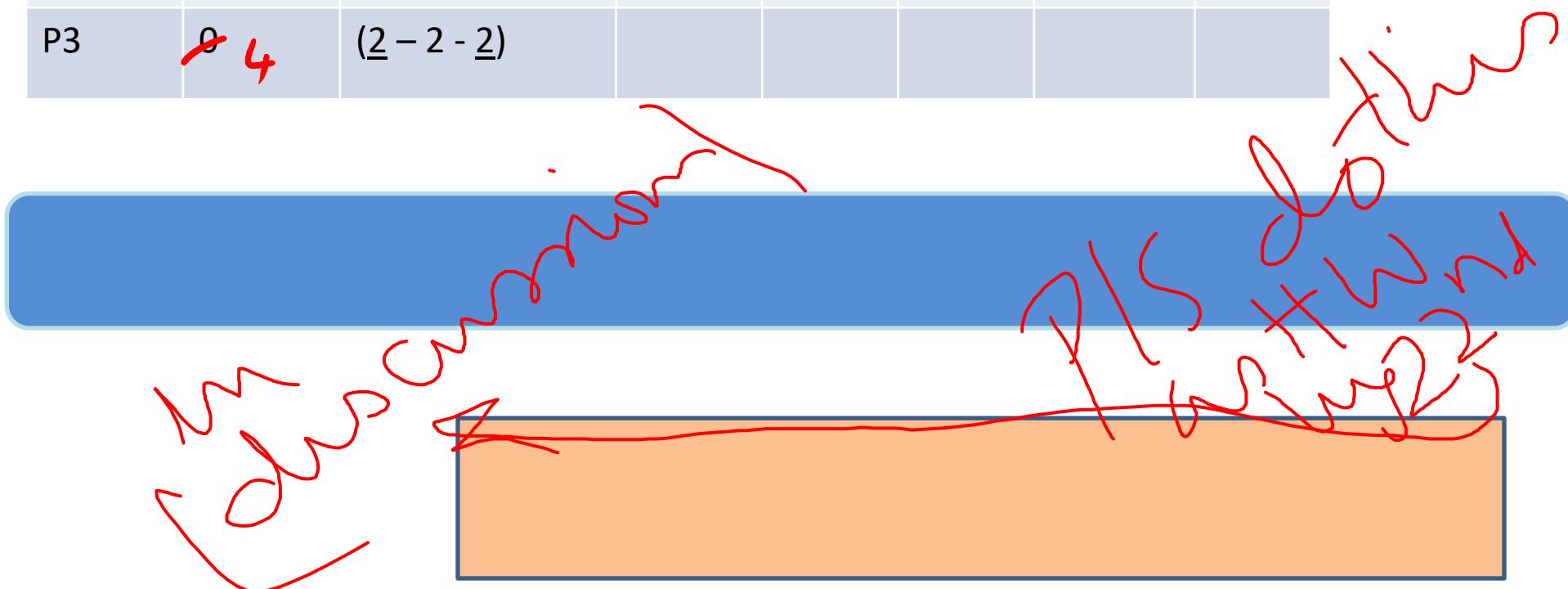
Process	AT	CPU- BT & I/O BT	Total BT	FT	TAT	WT	RT
P1	0	(<u>4</u> - 2 - <u>4</u>) 4	10	12	12	2	
P2	0	(<u>5</u> - 2 - <u>5</u>)	12	24	24	12	
P3	0	(<u>2</u> - 2 - <u>2</u>)	6	8	8	2	



SJF (Pre-emptive) with IO - Hw

*Priozitn
RR*

Process	AT	CPU- BT & I/O BT	Total BT	FT	TAT	WT	RT
P1	0	(<u>4</u> – 2 – <u>4</u>)					
P2	<u>0</u> <u>2</u>	(<u>5</u> – 2 – <u>5</u>)					
P3	<u>0</u> <u>4</u>	(<u>2</u> – 2 - <u>2</u>)					





BITS Pilani
Pilani Campus

Process Synchronization

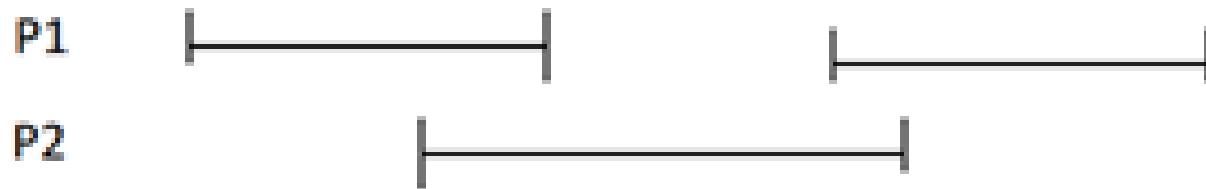
Background

- ❖ Processes can execute concurrently
 - ❖ May be interrupted at any time, partially completing execution
- ❖ Uniprocessor Vs Multiprocessor

uniprocessor system |



multiprocessor system



Contd...

- ❖ Concurrent access to shared data may result in data inconsistency

- ❖ Example

```
Procedure echo;  
Var out,in:Character;  
Begin  
    input (in, keyboard);  
    out:=in;  
    output(out,Display)  
End
```

- ❖ Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes

Example (Contd...)

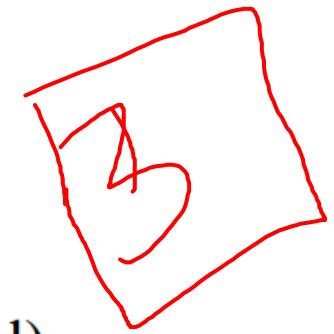
IN OUT = 3

Process P0

1. input (in, keyboard);
2. -----
3. -----
4. -----
5. out:=in;
6. output(out, Display)

Process P1

1. -----
2. input (in, keyboard);
3. out:=in;
4. output(out, Display)



Process Synchronization

- Process **Synchronization** means sharing system resources by processes in such a way that, concurrent access to shared data is handled thereby minimizing the chance of inconsistent data
 - A **Critical Section** is a code segment that accesses shared variables or resources and has to be executed as an atomic action.
 - Successful use of concurrency requires -
 - Ability to define critical section and
 - Enforce mutual exclusion
-

Process structure

Process P_i

do {

:

ENTRY SECTION

critical section

Exit SECTION

remainder section

} while (TRUE);

Main requirements

Three requirements

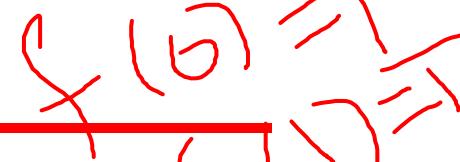
- **A mutual exclusion** : When one process is using a shared modifiable data, the other processes will be excluded from doing the same thing
 - **Progress** : when no process in critical section, any process that makes a request is allowed to enter critical section without any delay
 - **Bounded Waiting** : Processes requesting critical section should not be delayed indefinitely (no deadlock, starvation)
- ❖ No assumption should be made about relative execution speed of processes or number of processes
- ❖ A process remains inside critical section for a finite amount of time

Critical Section Handling in OS



- ❖ Two approaches depending on if kernel is preemptive or non-preemptive
 - ❖ **Preemptive** - allows preemption of process when running in kernel mode
 - ❖ **Non-preemptive** - runs until exits kernel mode, blocks, or voluntarily yields CPU
 - ❖ Essentially free of race conditions in kernel mode
- ❖ *Why then anyone would prefer preemptive kernel?????*

Critical Section → Peterson's solution



- Good algorithmic description of solving the problem
- Two process solution
- The two processes share two variables:
 - int turn;
 - Boolean flag[2]
- The variable **turn** indicates whose turn it is to enter the critical section
- The **flag** array is used to indicate if a process is ready to enter the critical section. **flag[i] = true** implies that process P_i is ready!

Algorithm

Process P_i

do {

flag[i] = true;

turn = j;

while (flag[j] && turn == j);

critical section

flag[i] = false;

} while (true);

Three requirements:

1. Mutual exclusion is preserved
2. Progress requirement is satisfied
3. Bounded-waiting requirement is met

Algorithm

$turn = 0$

$flag = \{F, F\}$

t	P0	P1
1	$flag[0] = true;$ $turn = 1;$	
2		$flag[1] = true;$ $turn = 0;$
3	$while (flag[1] \&& turn == 1);$ critical section	
4		$while (flag[0] \&& turn == 0);$ critical section
5	Critical Section... $flag[0] = false$	
6		$while (flag[0] \&& turn == 0);$ critical section $flag[1] = false;$

Semaphores

- is a variable which is treated in a special way
- allow processes to make use of critical section in exclusion to other processes
- process wanting to access critical section locks semaphore and releases lock on exit.
- is a synchronization tool

Contd...

-
- Basic properties of semaphore:
 - semaphore S - integer variable with non negative values
 - can only be accessed via two operations

wait (S):

```
while  $S \leq 0$  do no-op;  
       $S--;$ 
```

signal (S):

```
 $S++;$ 
```

- Semaphore operation is atomic and indivisible
 - wait and signal operations are carried out without interruption

Types of semaphore

- Binary semaphore : can have two values 0 and 1
 - Also known as **mutex locks**, as they are locks that provide mutual exclusion.
- *Counting Semaphore* : integer value can range over an unrestricted domain.

Critical Section of n Processes

Shared data:
semaphore S ;

//initially $S = 1$

wait (S):

while $S \leq 0$ do *no-op*;
 $S--;$

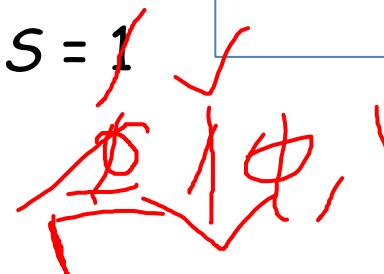
signal (S):

$S++;$

Process P_0 :
do {
 remainder section

 wait(S); — (Wants)
 critical ✓
 signal(S);

 remainder section
} while (1);



Process P_1 :
do {
 remainder section

wait(S); —
 critical section
 signal(S);
 remainder section
} while (1);

Process synchronization

- Semaphore can be used to solve various synchronization problems
- Example : two concurrent processes : P1 (with statement S1) and P2 (with statement S2)
 - requirement : **statement s2 should be executed only after statement s1 is executed**
 - semaphore variable : **synch** initialized to zero.

P1:

S1;
signal (synch);

P2:

wait (synch);
S2;

wait (synch):
while $synch \leq 0$ do no-
op;

signal (synch):
 $synch--;$
 $synch++;$

Contd...

- Main disadvantage : Busy Waiting → waiting wastes CPU cycles
- semaphore is also called a **spinlock** because the process "spins" while waiting for the lock.
- Solution: on finding zero semaphore value (binary semaphore), the process can block itself instead of busy waiting
- The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state.
- Then control is transferred to the CPU scheduler, which selects another process to execute.

Contd...

- A process that is blocked, waiting on a semaphore S , should be restarted when some other process executes a `signal()` operation.
- The process is restarted by a wakeup () operation, which changes the process from the waiting state to the ready state.
- The process is then placed in the ready queue.

Consumer - Producer Problem

- Also known as bounded buffer problem
- Two processes - consumer and producer
- Consumer and Producer processes share a common, fixed size buffer
- Producer process : generates data and puts it in the buffer
- Consumer process: consumes data from the buffer
- **Problem statement :** When a producer is placing an item in the buffer, then at the same time consumer should not consume any item.

mutex = 1

Full = 0 → Initially, all slots are empty.

Empty = n → All slots are empty

Producer

```
do{  
    //produce an item  
    wait(empty);  
    wait(mutex);  
        //place in buffer  
    signal(mutex);  
    signal(full);  
  
}while(true)
```

Consumer

```
do{  
    wait(full);  
    wait(mutex);  
        // remove item from buffer  
    signal(mutex);  
    signal(empty);  
        // consumes item  
  
}while(true)
```

Deadlock

EXAMPLES:

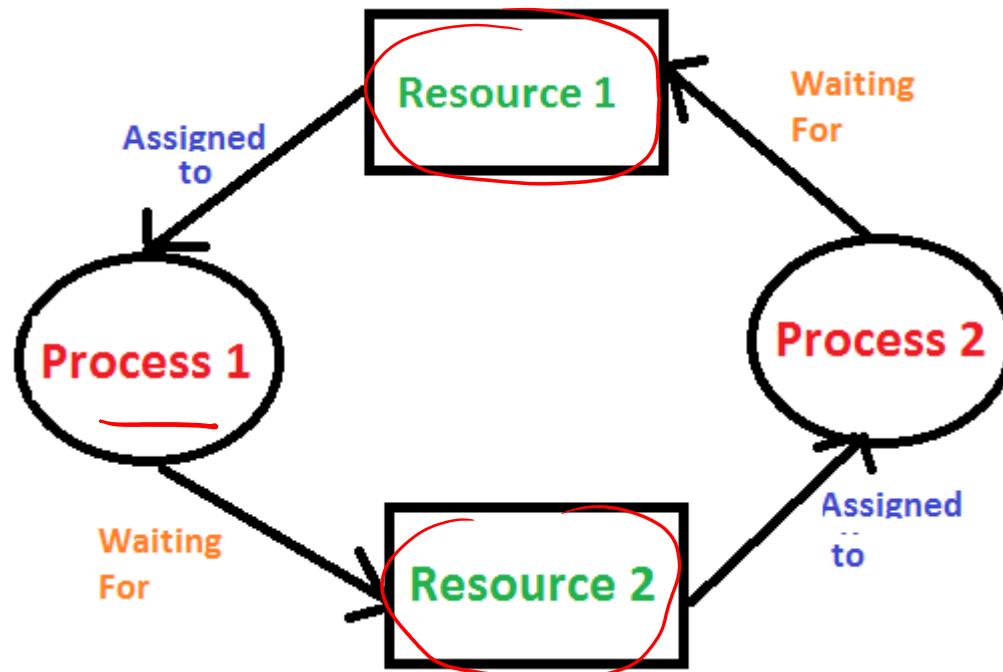
"It takes money to make money".

"You can't get a job without experience; you can't get experience without a job."

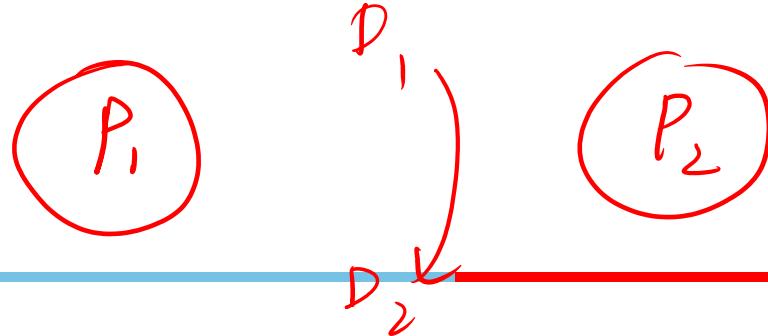


The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.



Contd...



- Example
 - System has 2 disk drives D1 and D2
 - P_1 and P_2 each hold one disk drive and each needs another one.
- Finite number of resources
 - Example : Memory space, CPU, files, I/O devices - printers, monitor, DVD drives
- Resource types and instances
 - system with 3 printers → Resource Type: printer and Number of instances : 3
- Process must request a resource before using it and must release the resource after using it

Contd...

- Each process utilizes a resource as follows:
 - request ✓
 - use ✓
 - release ✓
- Device : request () and release()
- File: open() and close ()
- Memory: allocate () and free ()

Deadlock Characterization

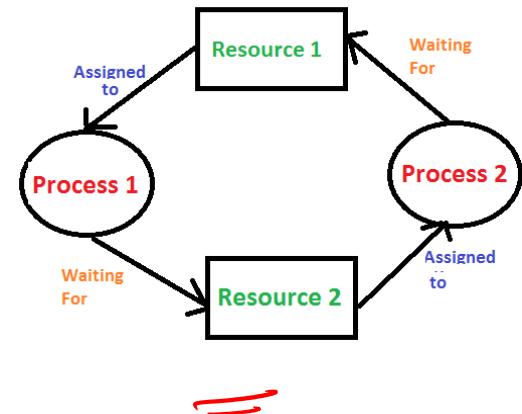
Mutual exclusion: only one process at a time can use a resource.

Hold and wait: a process holding at least one resource and is waiting to acquire additional resources held by other processes.

No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task.

Circular wait: there exists a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_n , and P_n is waiting for a resource that is held by P_0 .

$P_0 \rightarrow P_1 \rightarrow P_2 \dots \rightarrow P_n \rightarrow P_0$



$P_0 \rightarrow P_1 \rightarrow P_2 \dots \rightarrow P_n \rightarrow P_0$

NECESSARY CONDITIONS

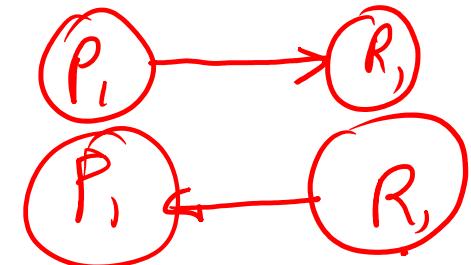
ALL of these four must happen simultaneously for a deadlock to occur

Resource-Allocation Graph

A set of vertices V and a set of edges E .

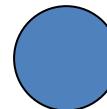
V is partitioned into two types:

- $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system.
 - $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system.
- ✓ request edge - directed edge $P_i \rightarrow R_j$
- ✓ assignment edge - directed edge $R_j \rightarrow P_i$

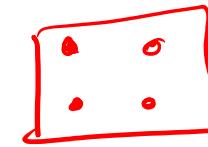
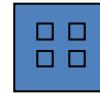


Resource-Allocation Graph (Cont.)

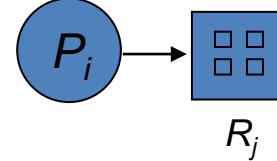
Process



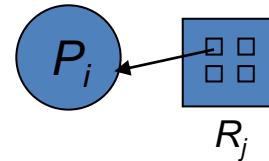
Resource Type with 4 instances



P_i requests instance of R_j



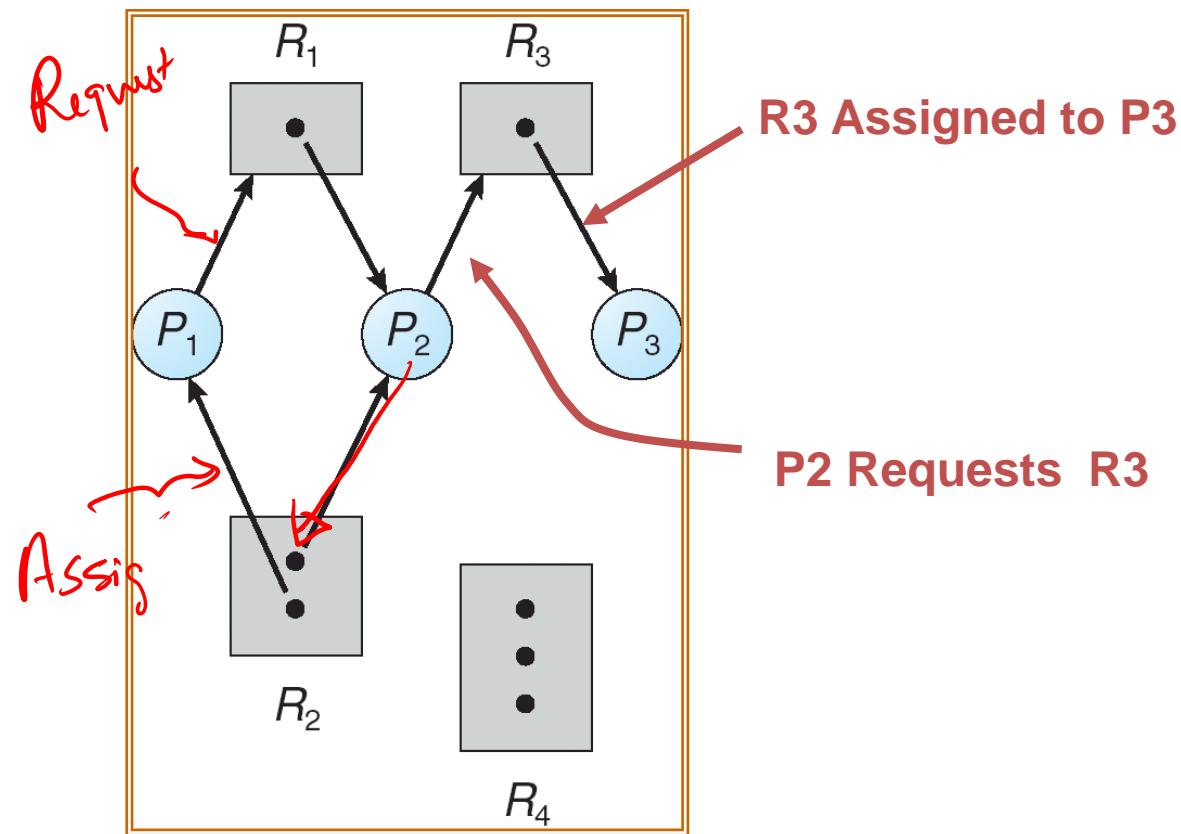
P_i is holding an instance of R_j



ME, HW, no preemption,
~~Circular wait~~



Example 1



Example 2

$P_1 \rightarrow R_1 \rightarrow P_2 \quad X$

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

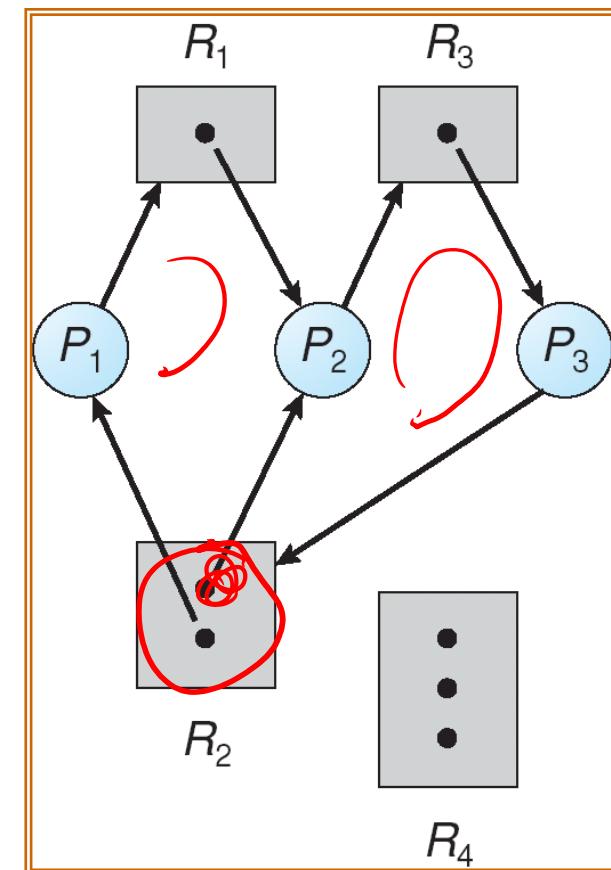
$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

Cycle 1:

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

Cycle 2:

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$



Basic Facts

- If graph contains no cycles \Rightarrow no deadlock.
- If graph contains a cycle \Rightarrow
 - if only one instance per resource type, then there is a deadlock.
a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock
 - if several instances per resource type, there **may be** a deadlock.
a cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock

Graph With A Cycle But No Deadlock



Cycle 1:

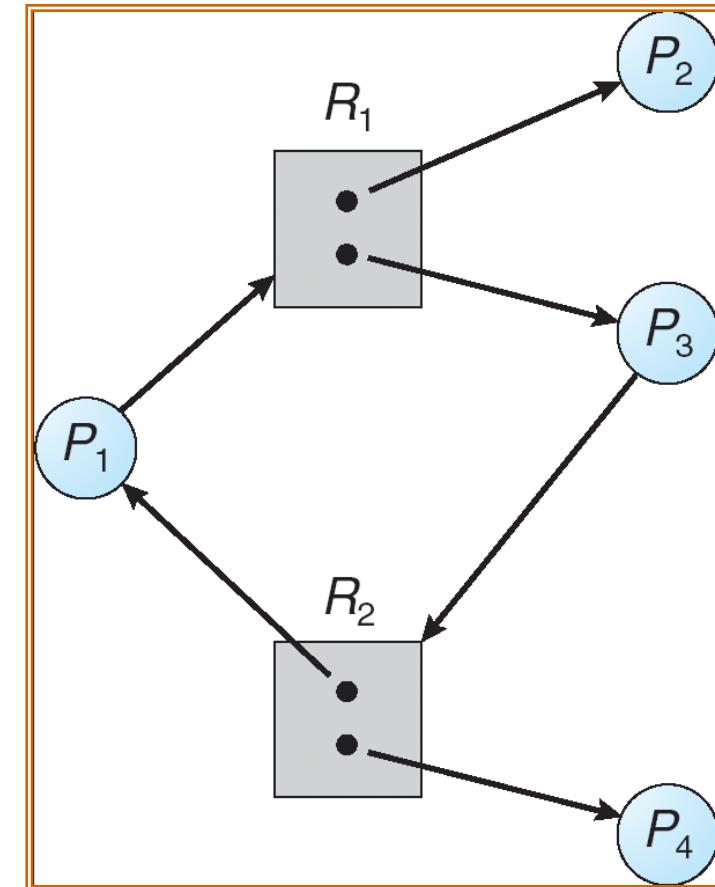
P1 → R1 → P3 → R2 → P1

But there is no deadlock.

Conclusion:

No Cycle: No deadlock

Cycle : may or may not be deadlock



Methods for Handling Deadlocks



- Deadlock Prevention and Deadlock avoidance:
 - ensuring that the system will never enter a deadlock state.
- Deadlock detection and recovery:
 - Allow the system to enter a deadlock state, detect it, and recover
 - Ignore the deadlock problem altogether and pretend that deadlocks never occur in the system.

Deadlock Prevention Vs. Deadlock avoidance

Deadlock Prevention: ME, W_XH, NP, CW

- The goal is to ensure that at least one of the necessary conditions for deadlock can never hold.
- The system does not require additional apriori information regarding the overall potential use of each resource for each process.

Deadlock avoidance:

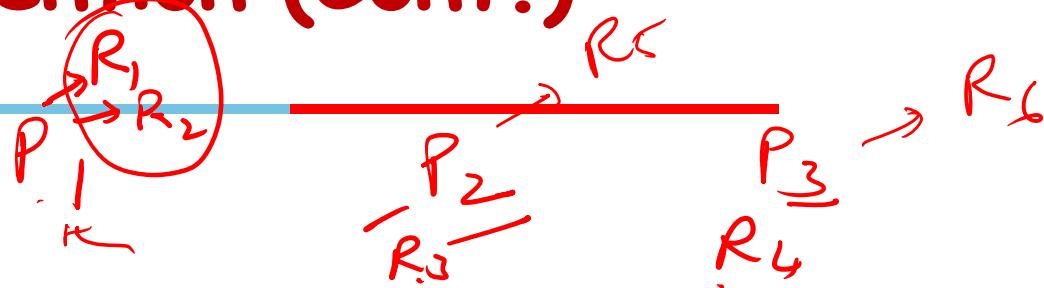
- The goal for deadlock avoidance is to the system must not enter an unsafe state.
- The system requires additional apriori information regarding the overall potential use of each resource for each process.

Deadlock Prevention

ME

- Mutual Exclusion - Sharable vs nonsharable resources
 - must hold for nonsharable resources . Example : Printer
 - not required for sharable resources. Example: Read only files
- Hold and Wait - must guarantee that whenever a process requests a resource, it does not hold any other resources.
 - ✓ – Protocol 1: requires each process to request and be allocated all its resources before it begins execution
 - No partial resource allocation
 - ✓ – Protocol 2: Allow process to request resources only when the process has none.
 - may request resources and use them, needs some more then release first.
 - Low resource utilization; starvation possible.

Deadlock Prevention (Cont.)



- **No Preemption -**

Protocol 1:

- If a process is holding some resources and requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
- Preempted resources are added to the list of resources
- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

Protocol 2:

- If a process requests some resources, check whether they are available. If so, allocate them.
- If not, check whether they are allocated to some other process that is waiting for additional resources. If so, preempt the desired resources from the waiting process and allocate them to the requesting process.

Deadlock Prevention (Cont.)

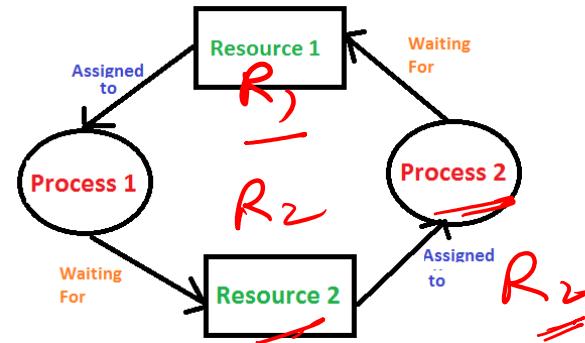
- **Circular Wait** - impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

$$F: R \rightarrow N$$

protocol to be followed is $F(R_i) < F(R_j)$



Main Drawback: low device utilization and reduced system throughput



Deadlock Avoidance

- Each process provides OS with information about its requests and releases for resources R_i
- Advantage: Simplest and most useful model
- Disadvantage : knowing all future requests and releases is difficult
- A resource allocation state is defined by
 - # of available resources
 - # of allocated resources to each process
 - maximum demands by each process
- On process request for a resource, OS needs to check whether the system is in **safe state** or not

safe state

Deadlock Avoidance...

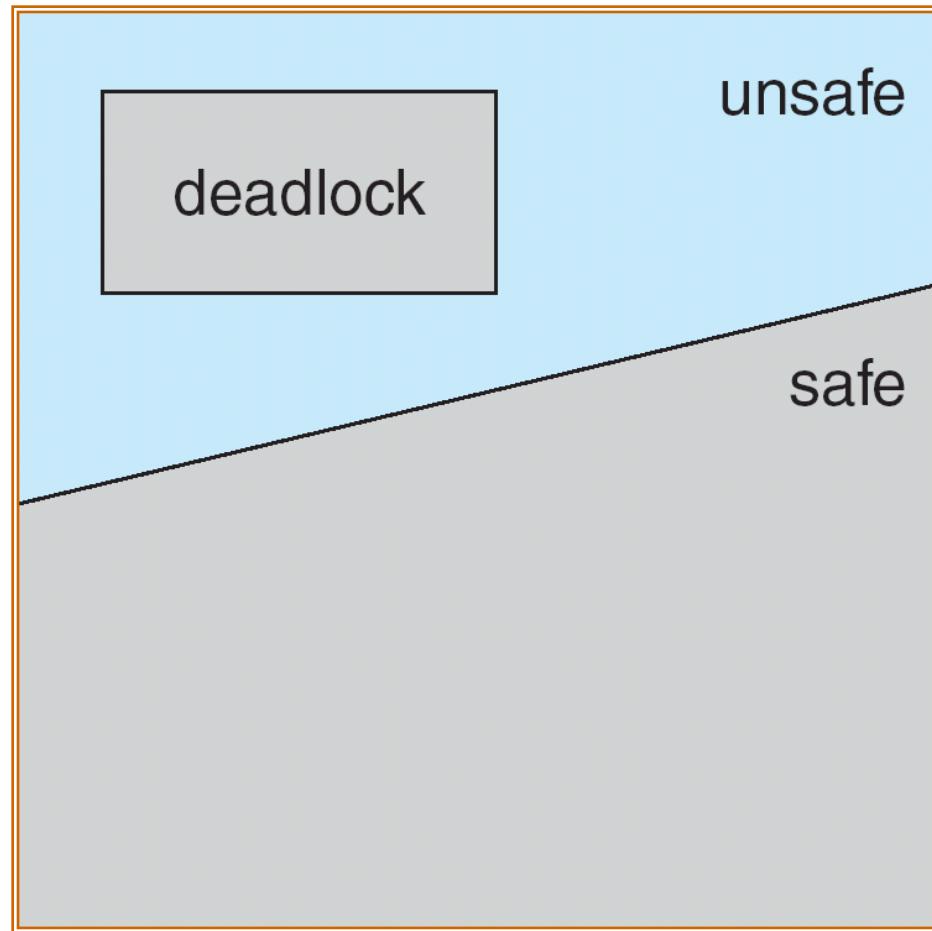
- A state is safe if a sequence of processes exist such that there are enough resources for the first to finish, and as each finishes and releases its resources that are enough for the next one to finish P_i
- That is:
 - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
 - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
 - When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

Basic Facts

- If a system is in safe state \Rightarrow no deadlocks.
- If a system is in unsafe state \Rightarrow possibility of deadlock.
- Avoidance \Rightarrow ensure that a system will never enter an unsafe state.

NOTE: All deadlocks are unsafe, but all unsafes are NOT deadlocks.

Safe, Unsafe, Deadlock State



Avoidance algorithms

- Single instance of a resource type → Use a resource-allocation graph
- Multiple instances of a resource type → Use the Banker's algorithm

Resource-Allocation Graph Scheme

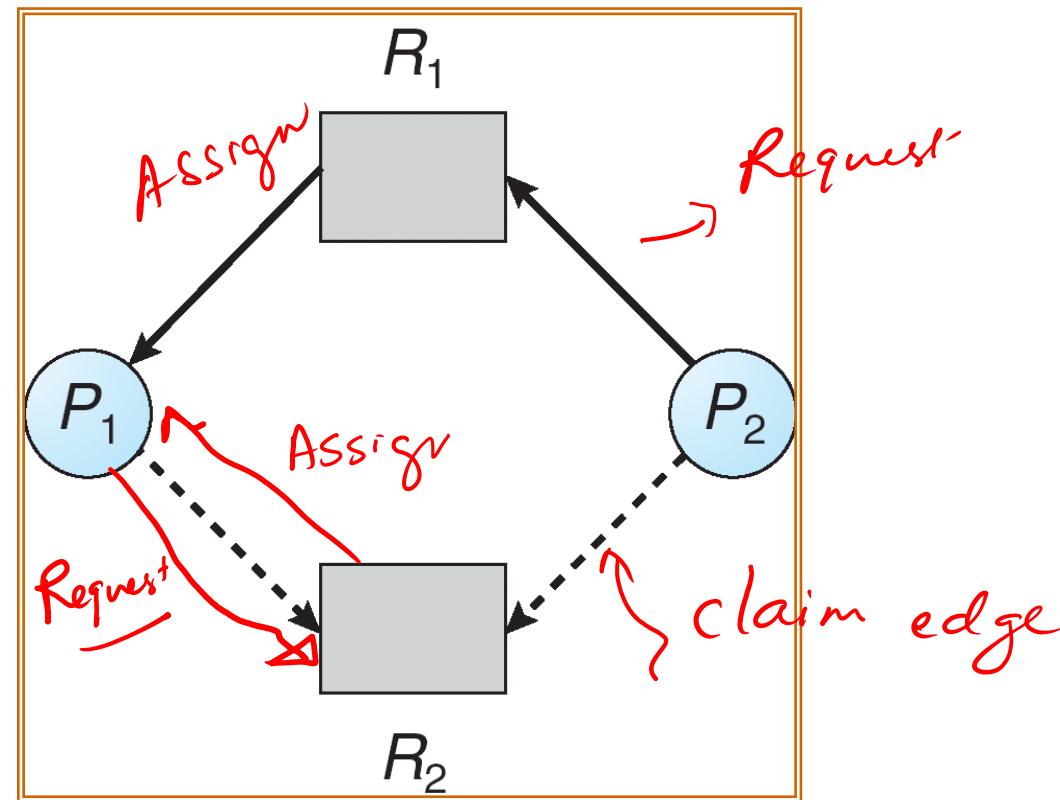
- Two edges : Request and assignment edge
- *Claim edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j ; some time in future*
 - represented by a dashed line.
- Claim edge converted to request edge when a process requests a resource.
- Request edge converted to an assignment edge when the resource is allocated to the process.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- **Resources must be claimed *a priori* in the system.**

Resource-Allocation Graph

$t=0$
 $P_1 R_1 R_2$

$t=1$
 $P_1 \leftarrow R_1$

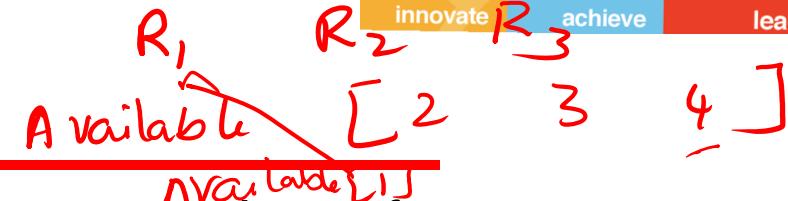
$t=2$
 $P_1 \leftarrow R_2$



Banker's Algorithm

- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm



Let n = number of processes, and m = number of resources types.

Available: Vector of length m . If $\text{Available}[j] = k$, there are k instances of resource type R_j available.

Max: $n \times m$ matrix. If $\text{Max}[i,j] = k$, then process P_i may request at most k instances of resource type R_j .

Allocation: $n \times m$ matrix. If $\text{Allocation}[i,j] = k$ then P_i is currently allocated k instances of R_j .

Need: $n \times m$ matrix. If $\text{Need}[i,j] = k$, then P_i may need k more instances of R_j to complete its task.

$$\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j].$$

Example of Banker's Algorithm

5 processes P_0 through P_4 :

3 resource types:

~~A (10 instances), B (5 instances), and C (7 instances).~~

Snapshot at time T_0 :

	<u>Allocation</u>			<u>Max</u>			<u>Available</u>			<u>Need</u>		
	<u>A</u>	<u>B</u>	<u>C</u>	<u>A</u>	<u>B</u>	<u>C</u>	<u>A</u>	<u>B</u>	<u>C</u>	<u>A</u>	<u>B</u>	<u>C</u>
P_0	0	1	0	7	5	3	3	3	2	7	4	3
P_1	2	0	0	3	2	2	1	2	2	6	0	0
P_2	3	0	2	9	0	2	0	1	1	4	3	1
P_3	2	1	1	2	2	2						
P_4	0	0	2	4	3	3						

$$\{10 \ 5 \ 7\} - [25] \Rightarrow 3 \ 3 \ 2$$



BITS Pilani
Pilani Campus

COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS

SESSION 11

Prepared: Dr. Lucy J. Gudino

Instructor: Prof. C R Sarma

WILP & Department of CS & IS

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Scheduling Algorithms : FCFS, SJF, SRTF Priority and Round-Robin Scheduling	T2

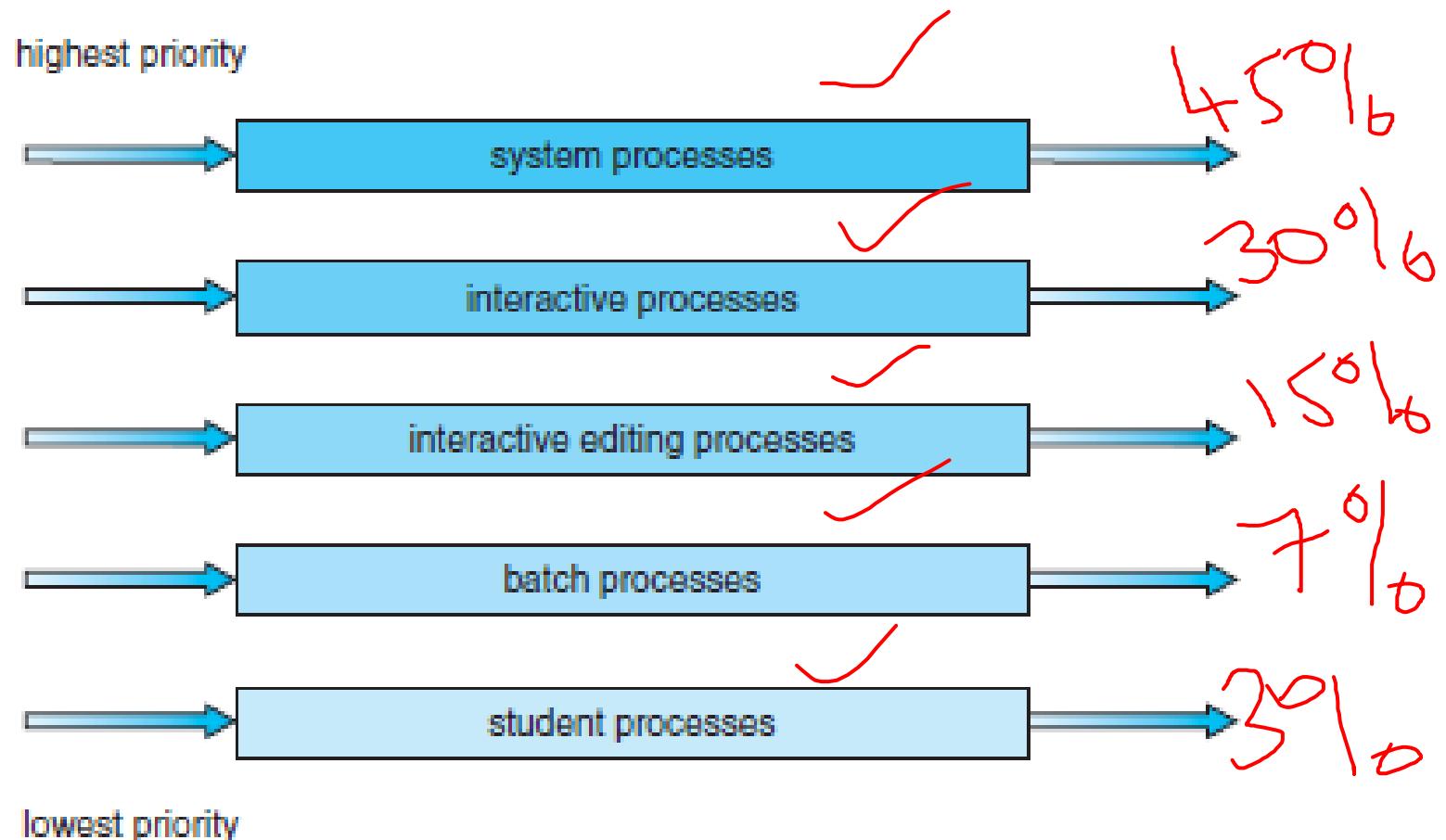
Today's Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Scheduling algorithms (RR and Multilevel scheduling)• Scheduling with I/O• Process Coordination• Deadlock (Introduction)	T2

Multilevel Queuing

- Process classification based on response-time requirements and scheduling needs
 - Foreground processes ✓ *interactive*
 - background processes ✓ *batch*
- Foreground processes may have priority (externally defined) over background processes
- Multilevel queue scheduling algorithm
 - Partition the ready queue into several separate queues
 - Each queue has its own scheduling algorithm
 - Example: foreground and background processes.
 - The foreground queue scheduled by an RR algorithm
 - background queue is scheduled by an FCFS algorithm.
- Process assignment to queue based on
 - Memory size, priority of process or process type

Multilevel Queue Scheduling



aging



Multilevel Feedback Queue

- Disadvantage of Multilevel queue : Inflexible
- **Multilevel Feedback Queue** : A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
 - number of queues
 - scheduling algorithms for each queue ✓
 - method used to determine when to upgrade a process
 - method used to determine when to demote a process
 - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:

Q_0 - RR - time quantum 8 ms

Q_1 - RR - time quantum 16 ms

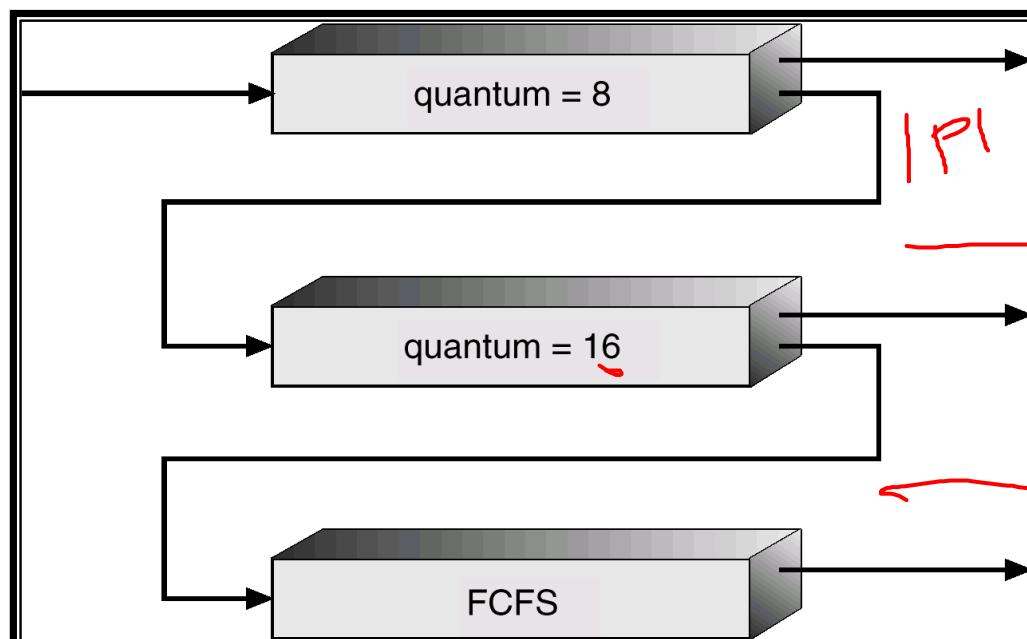
Q_2 - FCFS

~~P1(20), P2(25), P3(16), P4(20)~~

Scheduling

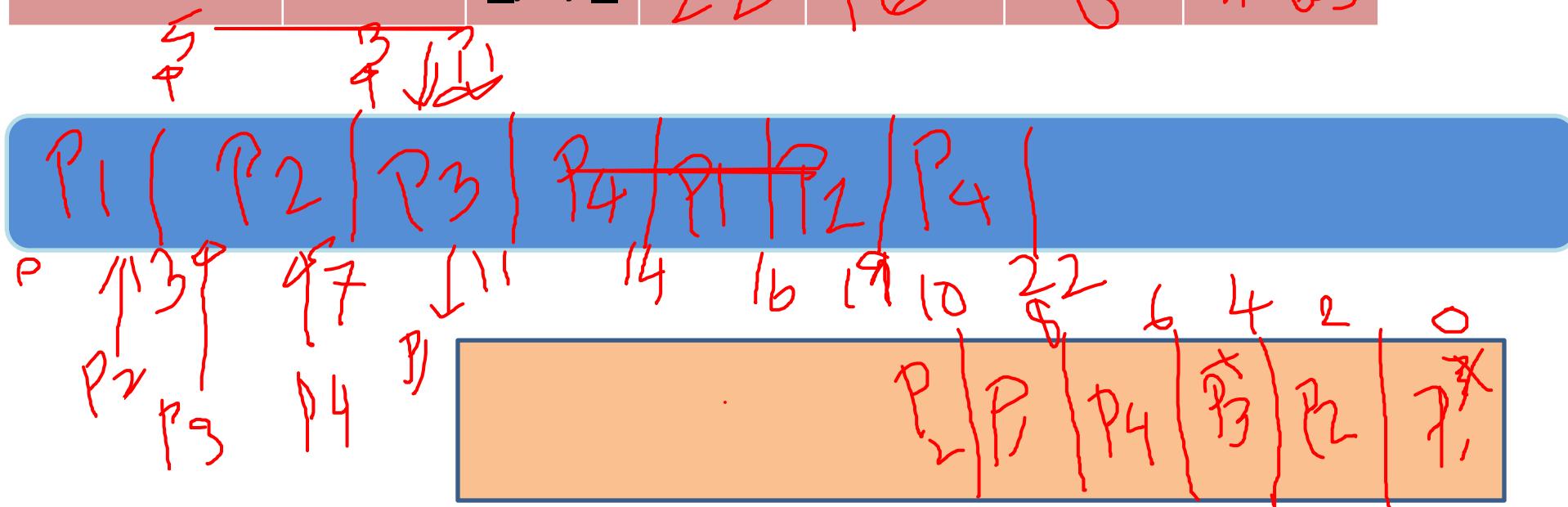
- A new process enters queue Q_0 . When it gains CPU, process receives 8 milliseconds. If it does not finish in 8 milliseconds, process is moved to queue Q_1 .

- At Q_1 process receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 .



FCFS with I/O

Process	AT	BT & I/O	FT	TAT	WT	RT
X P1	0	<u>3, 5, 2</u>	16	16	6	160
X P2	2	<u>4, 3, 3</u>	19	17	7	3
X P3	4	<u>4</u>	11	7	3	7-4=3
P4	6	<u>3, 2, 3</u>	22	16	8	11-6=5



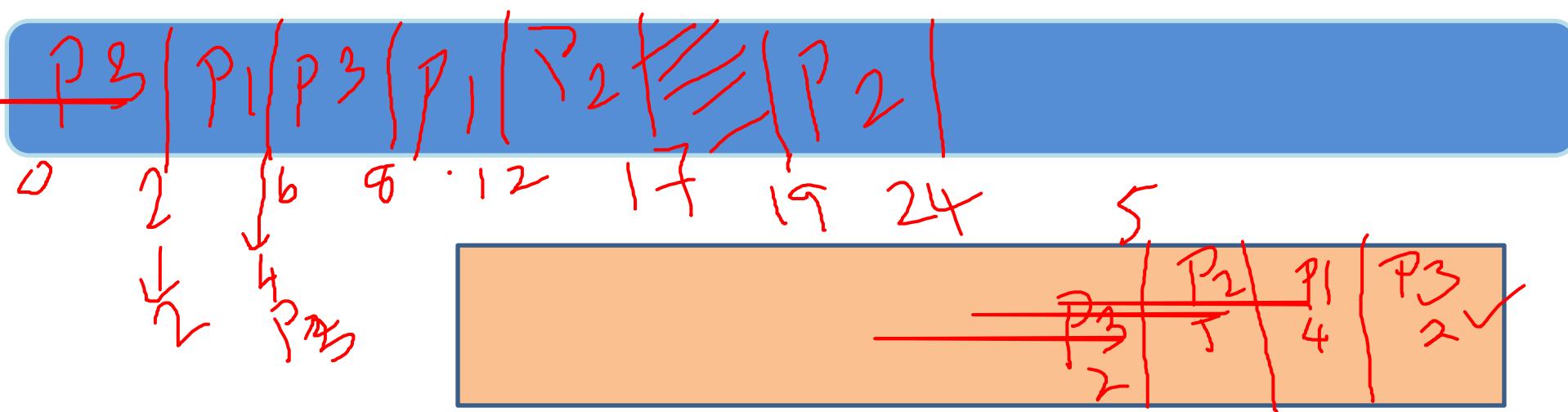
SJF (Non Pre-emptive) with IO

innovate

achieve

lead

Process	AT	CPU- BT & I/O BT	Total BT	FT	TAT	WT	RT
P1 X	0	(4 - 2 - 4)	10	12	12	2	2
P2 *	0	(5 - 2 - 5)	12	24	24	12	12
P3 *	0	(2 - 2 - 2)	6	8	8	2	0



SJF (Pre-emptive) with IO

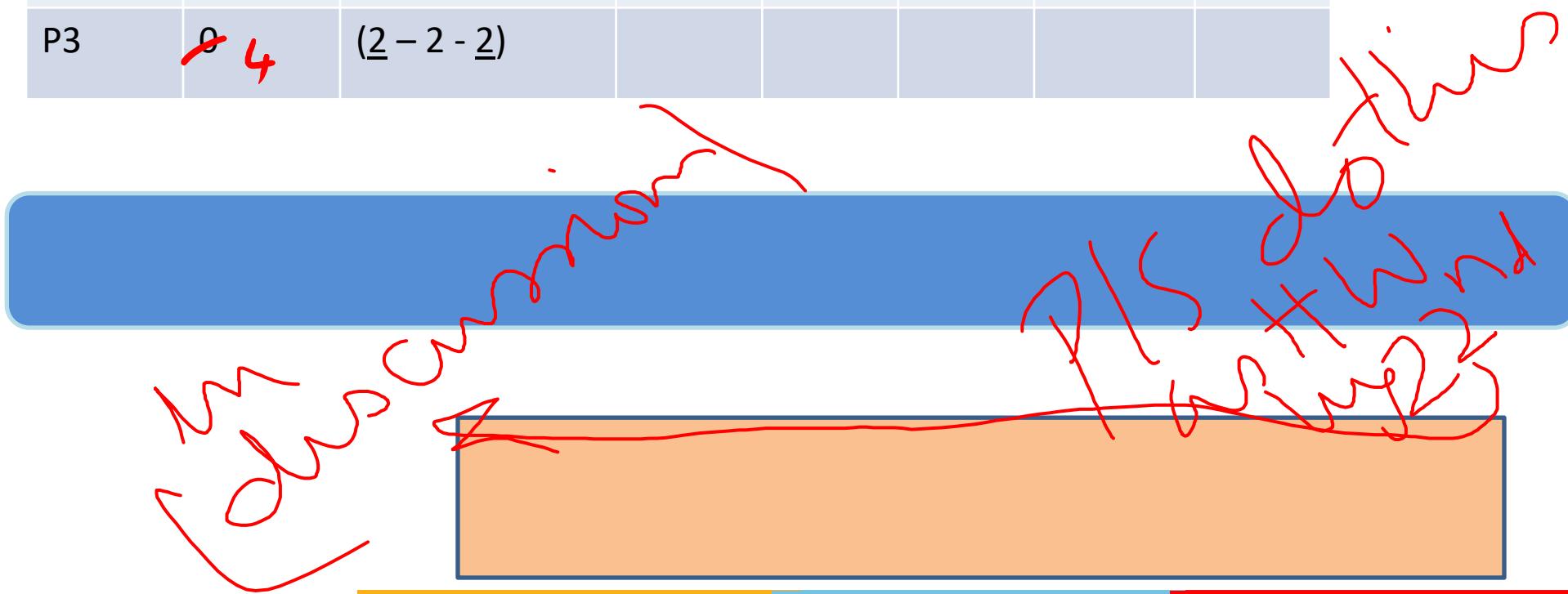
Process	AT	CPU- BT & I/O BT	Total BT	FT	TAT	WT	RT
P1	0	(<u>4</u> - 2 - <u>4</u>) 4	10	12	12	2	
P2	0	(<u>5</u> - 2 - <u>5</u>)	12	24	24	12	
P3	0	(<u>2</u> - 2 - <u>2</u>)	6	8	8	2	



SJF (Pre-emptive) with IO - Hw

*Priozitn
RR*

Process	AT	CPU- BT & I/O BT	Total BT	FT	TAT	WT	RT
P1	0	(<u>4</u> – 2 – <u>4</u>)					
P2	<u>0</u> <u>2</u>	(<u>5</u> – 2 – <u>5</u>)					
P3	<u>0</u> <u>4</u>	(<u>2</u> – 2 - <u>2</u>)					





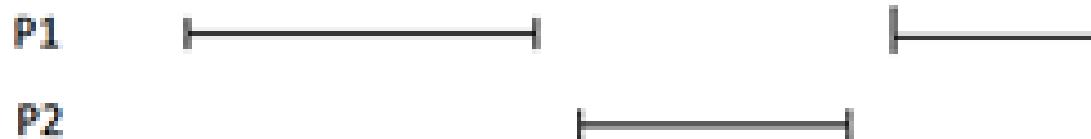
BITS Pilani
Pilani Campus

Process Synchronization

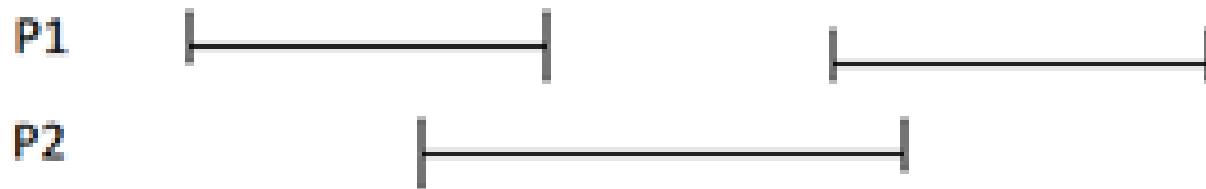
Background

- ❖ Processes can execute concurrently
 - ❖ May be interrupted at any time, partially completing execution
- ❖ Uniprocessor Vs Multiprocessor

uniprocessor system |



multiprocessor system



Contd...

- ❖ Concurrent access to shared data may result in data inconsistency

- ❖ Example

```
Procedure echo;  
Var out,in:Character;  
Begin  
    input (in, keyboard);  
    out:=in;  
    output(out,Display)  
End
```

- ❖ Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes

Example (Contd...)

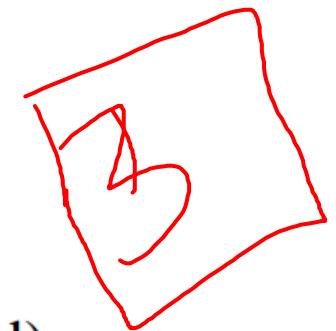


Process P0

1. input (in, keyboard);
2. -----
3. -----
4. -----
5. out:=in;
6. output(out, Display)

Process P1

1. -----
2. input (in, keyboard);
3. out:=in;
4. output(out, Display)



Process Synchronization

- Process **Synchronization** means sharing system resources by processes in such a way that, concurrent access to shared data is handled thereby minimizing the chance of inconsistent data
 - A **Critical Section** is a code segment that accesses shared variables or resources and has to be executed as an atomic action.
 - Successful use of concurrency requires -
 - Ability to define critical section and
 - Enforce mutual exclusion
-

Process structure

Process P_i

do {

:

ENTRY SECTION

critical section

Exit SECTION

remainder section

} while (TRUE);

Main requirements

Three requirements

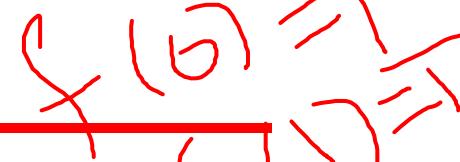
- **A mutual exclusion** : When one process is using a shared modifiable data, the other processes will be excluded from doing the same thing
 - **Progress** : when no process in critical section, any process that makes a request is allowed to enter critical section without any delay
 - **Bounded Waiting** : Processes requesting critical section should not be delayed indefinitely (no deadlock, starvation)
- ❖ No assumption should be made about relative execution speed of processes or number of processes
- ❖ A process remains inside critical section for a finite amount of time

Critical Section Handling in OS



- ❖ Two approaches depending on if kernel is preemptive or non-preemptive
 - ❖ **Preemptive** - allows preemption of process when running in kernel mode
 - ❖ **Non-preemptive** - runs until exits kernel mode, blocks, or voluntarily yields CPU
 - ❖ Essentially free of race conditions in kernel mode
- ❖ *Why then anyone would prefer preemptive kernel?????*

Critical Section → Peterson's solution



- Good algorithmic description of solving the problem
- Two process solution
- The two processes share two variables:
 - int turn;
 - Boolean flag[2]
- The variable **turn** indicates whose turn it is to enter the critical section
- The **flag** array is used to indicate if a process is ready to enter the critical section. **flag[i] = true** implies that process P_i is ready!

Algorithm

Process P_i

do {

flag[i] = true;

turn = j;

while (flag[j] && turn == j);

critical section

flag[i] = false;

} while (true);

Three requirements:

1. Mutual exclusion is preserved
2. Progress requirement is satisfied
3. Bounded-waiting requirement is met

Algorithm

$turn = 0$

$flag = \{F, F\}$

t	P0	P1
1	$flag[0] = true;$ $turn = 1;$	
2		$flag[1] = true;$ $turn = 0;$
3	$while (flag[1] \&& turn == 1);$ critical section	
4		$while (flag[0] \&& turn == 0);$ critical section
5	Critical Section... $flag[0] = false$	
6		$while (flag[0] \&& turn == 0);$ critical section $flag[1] = false;$

Semaphores

- is a variable which is treated in a special way
- allow processes to make use of critical section in exclusion to other processes
- process wanting to access critical section locks semaphore and releases lock on exit.
- is a synchronization tool

Contd...

- Basic properties of semaphore:
 - semaphore S - integer variable with non negative values
 - can only be accessed via two operations
 - wait (S):***
while $S \leq 0$ do no-op;
 $S--;$
 - signal (S):***
 $S++;$
- Semaphore operation is atomic and indivisible
 - wait and signal operations are carried out without interruption

Types of semaphore

- Binary semaphore : can have two values 0 and 1
 - Also known as **mutex locks**, as they are locks that provide mutual exclusion.
- *Counting Semaphore* : integer value can range over an unrestricted domain.

Critical Section of n Processes

Shared data:
semaphore S :

//initially $S = 1$

wait (S):

while $S \leq 0$ do *no-op*;
 $S--;$

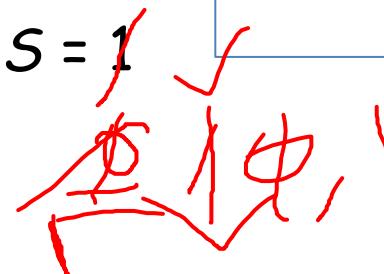
signal (S):

$S++;$

Process P_0 :
do {
 remainder section

 wait(S); — (Wants)
 critical ✓
 signal(S);

 remainder section
} while (1);



Process P_1 :
do {
 remainder section

wait(S); —
 critical section
~~signal(S);~~

 remainder section
} while (1);

Process synchronization

- Semaphore can be used to solve various synchronization problems
- Example : two concurrent processes : P1 (with statement S1) and P2 (with statement S2)
 - requirement : **statement s2 should be executed only after statement s1 is executed**
 - semaphore variable : **synch** initialized to zero.

P1:

S1;
signal (synch);

P2:

wait (synch);
S2;

wait (synch):
while $synch \leq 0$ do no-
op;

signal (synch):
 $synch--;$
 $synch++;$

Contd...

- Main disadvantage : Busy Waiting → waiting wastes CPU cycles
- semaphore is also called a **spinlock** because the process "spins" while waiting for the lock.
- Solution: on finding zero semaphore value (binary semaphore), the process can block itself instead of busy waiting
- The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state.
- Then control is transferred to the CPU scheduler, which selects another process to execute.

Contd...

- A process that is blocked, waiting on a semaphore S , should be restarted when some other process executes a `signal()` operation.
- The process is restarted by a wakeup () operation, which changes the process from the waiting state to the ready state.
- The process is then placed in the ready queue.

Consumer - Producer Problem

- Also known as bounded buffer problem
- Two processes - consumer and producer
- Consumer and Producer processes share a common, fixed size buffer
- Producer process : generates data and puts it in the buffer
- Consumer process: consumes data from the buffer
- **Problem statement :** When a producer is placing an item in the buffer, then at the same time consumer should not consume any item.

mutex = 1

Full = 0 → Initially, all slots are empty.

Empty = n → All slots are empty

Producer

```
do{  
    //produce an item  
    wait(empty);  
    wait(mutex);  
        //place in buffer  
    signal(mutex);  
    signal(full);  
  
}while(true)
```

Consumer

```
do{  
    wait(full);  
    wait(mutex);  
        // remove item from buffer  
    signal(mutex);  
    signal(empty);  
        // consumes item  
  
}while(true)
```

Deadlock

EXAMPLES:

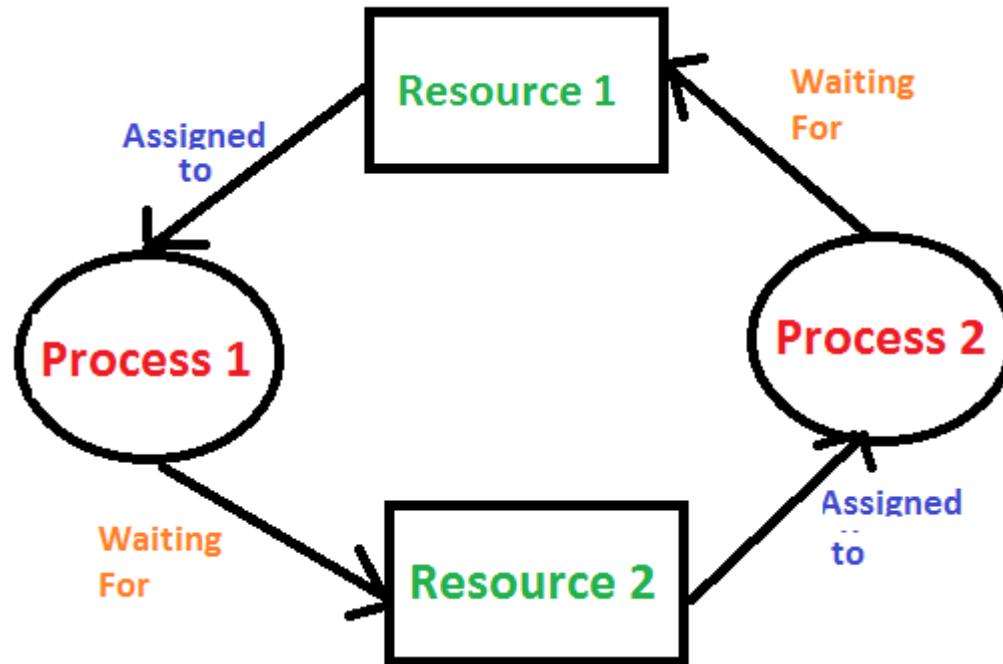
"It takes money to make money".

"You can't get a job without experience; you can't get experience without a job."



The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.



Contd...

- Example
 - System has 2 disk drives D1 and D2
 - P_1 and P_2 each hold one disk drive and each needs another one.
- Finite number of resources
 - Example : Memory space, CPU, files, I/O devices - printers, monitor, DVD drives
- Resource types and instances
 - system with 3 printers → Resource Type: printer and Number of instances : 3
- Process must request a resource before using it and must release the resource after using it

Contd...

- Each process utilizes a resource as follows:
 - request
 - use
 - release
- Device : request () and release()
- File: open() and close ()
- Memory: allocate () and free ()

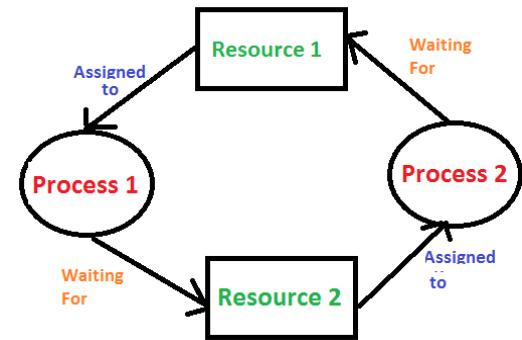
Deadlock Characterization

Mutual exclusion: only one process at a time can use a resource.

Hold and wait: a process holding at least one resource and is waiting to acquire additional resources held by other processes.

No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task.

Circular wait: there exists a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2, \dots, P_{n-1} is waiting for a resource that is held by P_n , and P_n is waiting for a resource that is held by P_0 .



NECESSARY CONDITIONS

ALL of these four must happen simultaneously for a deadlock to occur

Resource-Allocation Graph

A set of vertices V and a set of edges E .

V is partitioned into two types:

- $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system.
- $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system.

request edge - directed edge $P_i \rightarrow R_j$

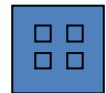
assignment edge - directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

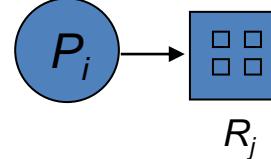
Process



Resource Type with 4 instances

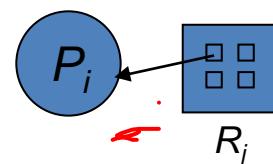


P_i requests instance of R_j



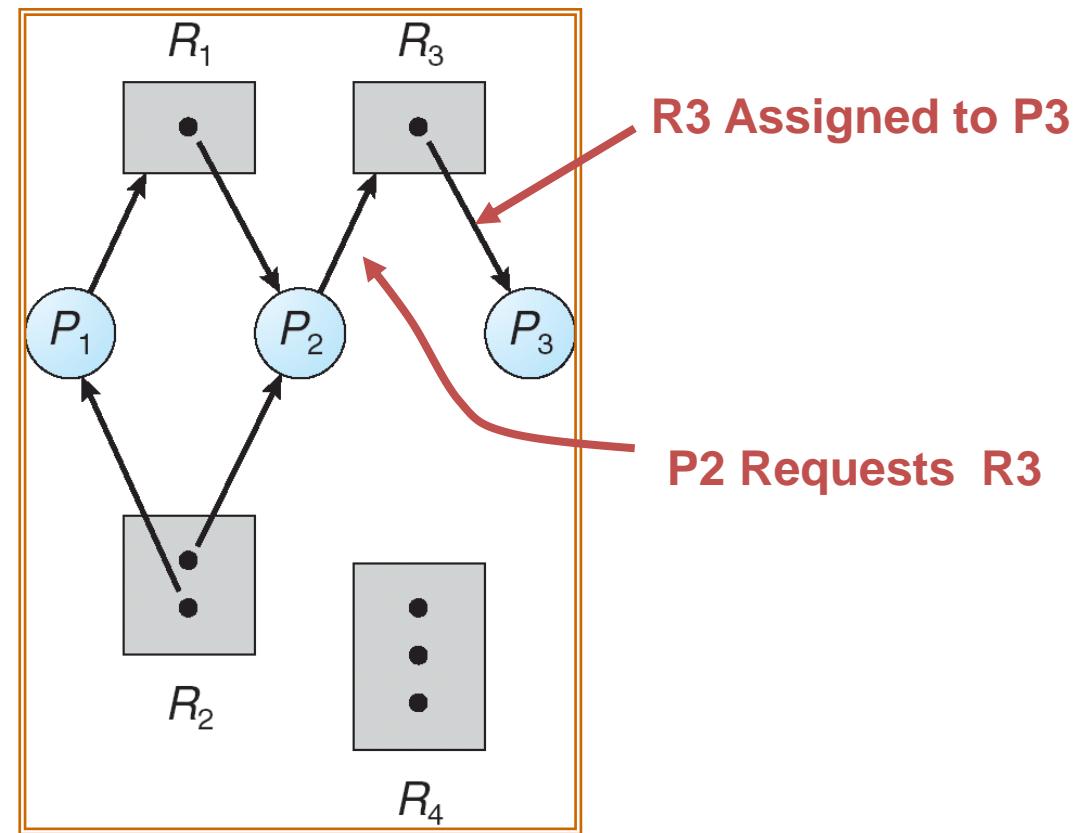
P_i requested
 R_j

P_i is holding an instance of R_j



$P_i \leftarrow R_j$

Example 1



Example 2

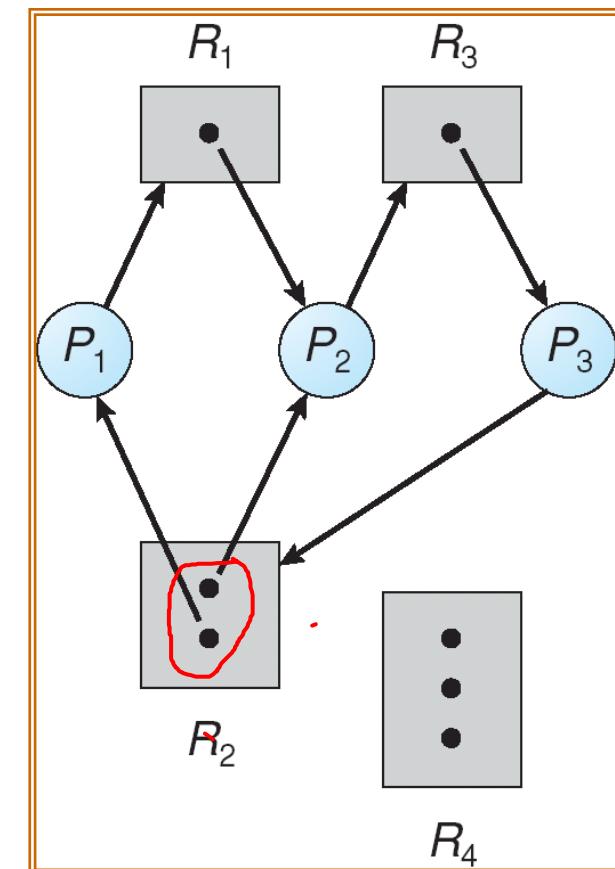
~~not cycle~~



Cycle 1:
 $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

Cycle 2:
 $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

— Cycle



Basic Facts

- If graph contains no cycles \Rightarrow no deadlock.
- If graph contains a cycle \Rightarrow
 - if only one instance per resource type, then there is a deadlock.
a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock
 - if several instances per resource type, there **may be** a deadlock.
a cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock

Graph With A Cycle But No Deadlock



Cycle 1:

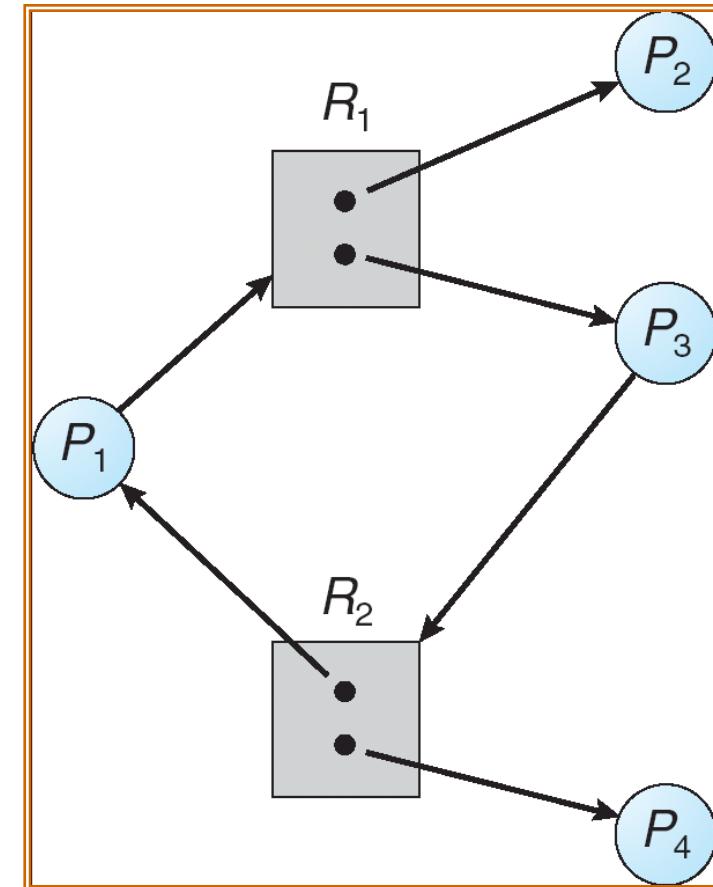
P1 → R1 → P3 → R2 → P1

But there is no deadlock.

Conclusion:

No Cycle: No deadlock

Cycle : may or may not be deadlock



Methods for Handling Deadlocks



- Deadlock Prevention and Deadlock avoidance:
 - ensuring that the system will never enter a deadlock state.
- Deadlock detection and recovery:
 - Allow the system to enter a deadlock state, detect it, and recover
- Ignore the deadlock problem altogether and pretend that deadlocks never occur in the system.

Deadlock Prevention Vs. Deadlock avoidance

Deadlock Prevention:

- The goal is to ensure that at least one of the necessary conditions for deadlock can never hold.
- The system does not require additional apriori information regarding the overall potential use of each resource for each process.

Deadlock avoidance:

- The goal for deadlock avoidance is to the system must not enter an unsafe state.
- The system requires additional apriori information regarding the overall potential use of each resource for each process.

Deadlock Prevention

- **Mutual Exclusion** - Sharable vs nonsharable resources
 - must hold for nonsharable resources . Example : Printer
 - not required for sharable resources. Example: Read only files
- **Hold and Wait** - must guarantee that whenever a process requests a resource, it does not hold any other resources.
 - Protocol 1: requires each process to request and be allocated all its resources before it begins execution
 - No partial resource allocation
 - Protocol 2: Allow process to request resources only when the process has none.
 - may request resources and use them, needs some more then release first.
 - Low resource utilization; starvation possible.

Deadlock Prevention (Cont.)

- **No Preemption -**

Protocol 1:

- If a process is holding some resources and requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
- Preempted resources are added to the list of resources
- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

Protocol 2:

- If a process requests some resources, check whether they are available. If so, allocate them.
- If not, check whether they are allocated to some other process that is waiting for additional resources. If so, preempt the desired resources from the waiting process and allocate them to the requesting process.

Deadlock Prevention (Cont.)

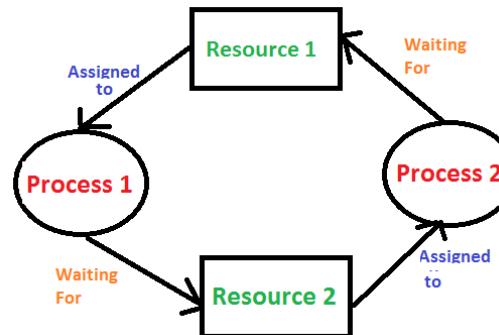
- **Circular Wait** - impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

$$F: R \rightarrow N$$

protocol to be followed is $F(R_i) < F(R_j)$



Main Drawback: low device utilization and reduced system throughput



Deadlock Avoidance

- Each process provides OS with information about its requests and releases for resources R_i
- Advantage: Simplest and most useful model
- Disadvantage : knowing all future requests and releases is difficult
- A resource allocation state is defined by
 - # of available resources
 - # of allocated resources to each process
 - maximum demands by each process
- On process request for a resource, OS needs to check whether the system is in **safe state** or not

safe state

Deadlock Avoidance...

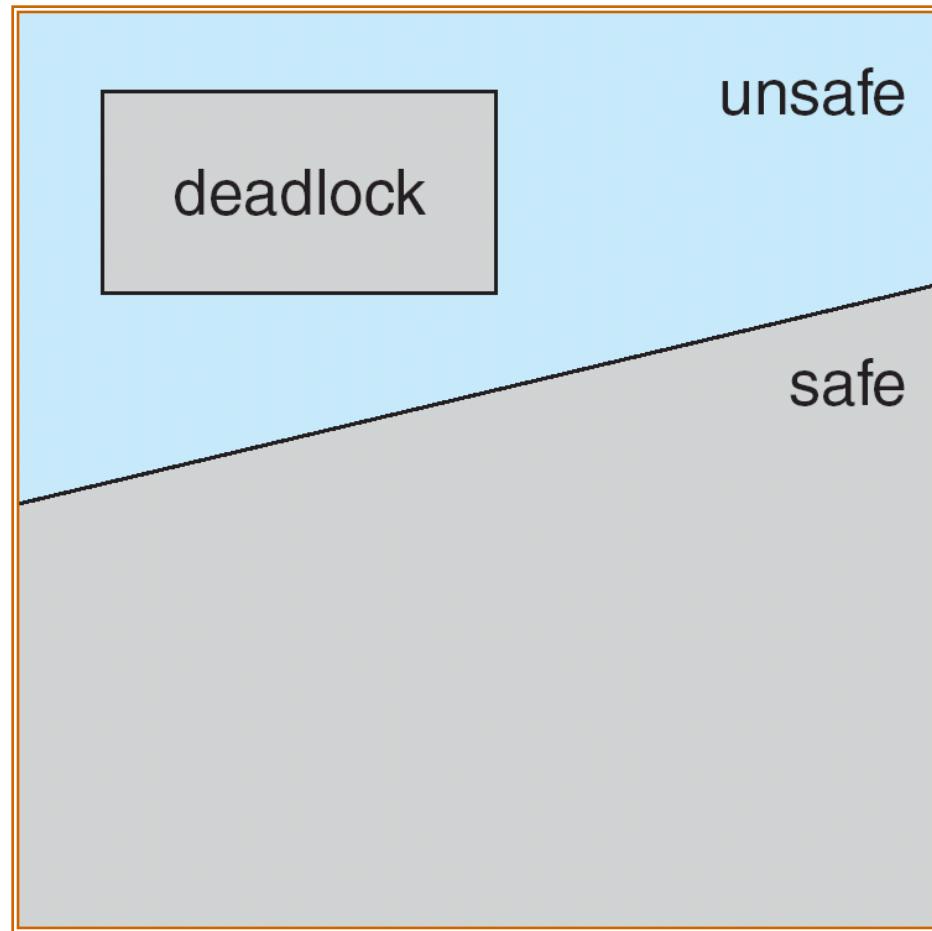
- A state is safe if a sequence of processes exist such that there are enough resources for the first to finish, and as each finishes and releases its resources that are enough for the next one to finish
- That is:
 - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
 - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
 - When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

Basic Facts

- If a system is in safe state \Rightarrow no deadlocks.
- If a system is in unsafe state \Rightarrow possibility of deadlock.
- Avoidance \Rightarrow ensure that a system will never enter an unsafe state.

NOTE: All deadlocks are unsafe, but all unsafes are NOT deadlocks.

Safe, Unsafe, Deadlock State



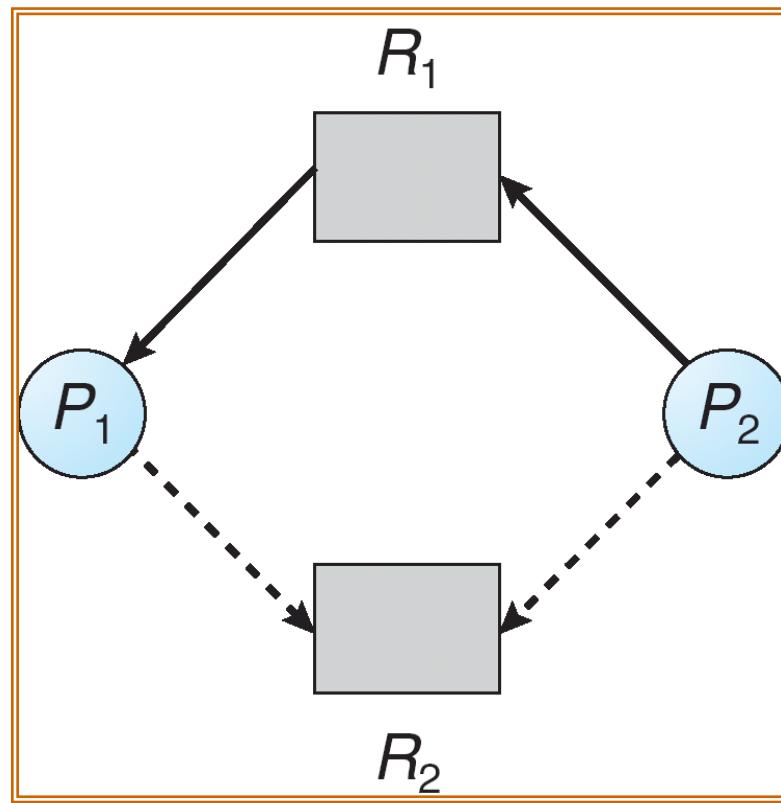
Avoidance algorithms

- Single instance of a resource type → Use a resource-allocation graph
- Multiple instances of a resource type → Use the Banker's algorithm

Resource-Allocation Graph Scheme

- Two edges : Request and assignment edge
- *Claim edge* $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j ; some time in future
 - represented by a dashed line.
- Claim edge converted to request edge when a process requests a resource.
- Request edge converted to an assignment edge when the resource is allocated to the process.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- **Resources must be claimed *a priori* in the system.**

Resource-Allocation Graph



Banker's Algorithm

- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm



$$n=5, m=3, A = \{0, 1, 2\}, B = \{5, 3, 2\}$$

Let n = number of processes, and m = number of resources types.

Available: Vector of length m . If $\text{Available}[j] = k$, there are k instances of resource type R_j available.

Max: $n \times m$ matrix. If $\text{Max}[i,j] = k$, then process P_i may request at most k instances of resource type R_j .

Allocation: $n \times m$ matrix. If $\text{Allocation}[i,j] = k$ then P_i is currently allocated k instances of R_j .

Need: $n \times m$ matrix. If $\text{Need}[i,j] = k$, then P_i may need k more instances of R_j to complete its task.

$$\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j].$$

Example of Banker's Algorithm

5 processes P_0 through P_4 :

3 resource types:

A (10 instances), B (5 instances), and C (7 instances)

Snapshot at time T_0 :

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
P_0	0 1 0	7 5 3	3 3 2 X	7 4 3
P_1	2 0 0	3 2 2	5 3 2 X	1 2 2 =
P_2	3 0 2	9 0 2	7 4 1 +	6 0 0
P_3	-2 1 1	2 2 2	7 4 5 +	5 1 1
P_4	0 0 2	4 3 3	7 5 5	4 3 1
			7 5 7	
				[P1, P3, P4, P0, P2]



BITS Pilani
Pilani Campus

COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS

SESSION 13

Prepared: Dr. Lucy J. Gudino

Instructor: Prof. C R Sarma

WILP & Department of CS & IS



Today's Session

List of Topic Title	Text/Ref Book/external resource
<ul style="list-style-type: none">• Deadlock	T2

Avoidance algorithms

- Single instance of a resource type → Use a resource-allocation graph
- Multiple instances of a resource type → Use the Banker's algorithm

Banker's Algorithm

- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm



Let n = number of processes, and m = number of resources types.

Available: Vector of length m . If $\text{Available}[j] = k$, there are k instances of resource type R_j available.

Max: $n \times m$ matrix. If $\text{Max}[i,j] = k$, then process P_i may request at most k instances of resource type R_j .

Allocation: $n \times m$ matrix. If $\text{Allocation}[i,j] = k$ then P_i is currently allocated k instances of R_j .

Need: $n \times m$ matrix. If $\text{Need}[i,j] = k$, then P_i may need k more instances of R_j to complete its task.

$$\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j].$$

Example of Banker's Algorithm

5 processes P_0 through P_4 :

3 resource types:

A (10 instances), B (5 instances), and C (7 instances).

Snapshot at time T_0 :

	<u>Allocation</u>			<u>Max</u>			<u>Available</u>			<u>Need</u>		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3				3	4	7
P_1	2	0	0	3	2	2						
P_2	3	0	2	9	0	2						
P_3	2	1	1	2	2	2						
P_4	0	0	2	4	3	3						

Safety Algorithm

1. Let **Work** and **Finish** be vectors of length m and n , respectively. Initialize:

Work = Available

Finish [i] = false for $i = 0, 1, \dots, n-1$

2. Find an i such that both: $+ (0) ((1)(\cdot))$.

(a) $\text{Finish}[i] = \text{false}$

(b) $\text{Need}_i \leq \text{Work}$

If no such i exists, go to step 4.

$\begin{cases} T & \\ F & \end{cases}$

3. $\text{Work} = \text{Work} + \underline{\text{Allocation}}$;

$\text{Finish}[i] = \text{true}$

go to step 2.

4. If $\text{Finish}[i] == \text{true}$ for all i , then the system is in a safe state.

\approx

Example (Cont. 1/3)

The content of the matrix Need is defined to be Max - Allocation.

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

The system is in a safe state since the sequence $\langle \rangle$ satisfies safety criteria.

Example (Cont. 2/3)

The content of the matrix Need is defined to be Max - Allocation.

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2 X	7 4 3 ✓
P_1	2 0 0	3 2 2	5 3 2 X	1 2 2 ✓
P_2	3 0 2	9 0 2	7 4 3 X	6 0 0 ✓
P_3	2 1 1	2 2 2	7 4 5 X	0 1 1 ✓
P_4	0 0 2	4 3 3	7 5 5 X 1 0 5 X	4 3 1 ✓

The system is in a safe state since the sequence $\langle \rangle$ satisfies safety criteria.

P_1, P_3, P_4, P_0, P_2

Example (Cont. 3/3)

The content of the matrix *Need* is defined to be *Max - Allocation*.

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

The system is in a safe state since the sequence $\langle \rangle$ satisfies safety criteria.

Example (Cont. 3/3)

The content of the matrix *Need* is defined to be *Max - Allocation*.

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

The system is in a safe state since the sequence $\langle \rangle$ satisfies safety criteria.

Example: P_1 Request (1,0,2)

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Two steps:

1. Run Resource - Request Algorithm
2. Check whether the system is safe using safety algorithm

Resource-Request Algorithm for



Process P_i

Request_i = request vector for process P_i . If $\text{Request}_i[j] = k$ then process P_i wants k instances of resource type R_j .

1. If $\text{Request}_i \leq \text{Need}_i$, go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $\text{Request}_i \leq \text{Available}$, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$$\text{Available} = \text{Available} - \text{Request}_i;$$

$$\text{Allocation}_i = \text{Allocation}_i + \text{Request}_i;$$

$$\text{Need}_i = \text{Need}_i - \text{Request}_i;$$

- If safe \Rightarrow the resources are allocated to P_i .
- If unsafe $\Rightarrow P_i$ must wait, and the old resource-allocation state is restored

Example: P_1 Request (1,0,2)

Pretend to allocate requested resources to P_i by modifying the state as follows:

$$\text{Available} = \text{Available} - \text{Request}_i;$$

$$\text{Allocation}_i = \text{Allocation}_i + \text{Request}_i;$$

$$\text{Need}_i = \text{Need}_i - \text{Request}_i;$$

Check that Request \leq need

Request \leq Available (that is, (1,0,2) \leq (3,3,2) \Rightarrow true

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
P_0	ABC	ABC	ABC	ABC
P_1	010	753	332	743
P_1	300	322	230	122
P_2	302	902		600
P_3	211	222		011
P_4	002	433		431

X B C
3 3 2 +
2 3 0 +
5 3 2 +
7 4 3 X
7 4 5
7 5 5
(10 5 7)

P_1, P_3, P_4, P_0, P_2

Example: P_1 Request (1,0,2)

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>	
	A B C	A B C	A B C	-----
P_0	0 1 0	7 4 3	2 3 0	
P_1	3 0 2	0 2 0		
P_2	3 0 2	6 0 0		
P_3	2 1 1	0 1 1		
P_4	0 0 2	4 3 1		

Example: P_1 Request (1,0,2)

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>	
	A B C	A B C	A B C	-----
P_0	0 1 0	7 4 3	2 3 0	
P_1	3 0 2	0 2 0		
P_2	3 0 2	6 0 0		
P_3	2 1 1	0 1 1		
P_4	0 0 2	4 3 1		



Check This...



Can request for $(3,3,0)$ by P_4 be granted?

Can request for $(0,2,0)$ by P_0 be granted?

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

give can't
Sequence
switch
and

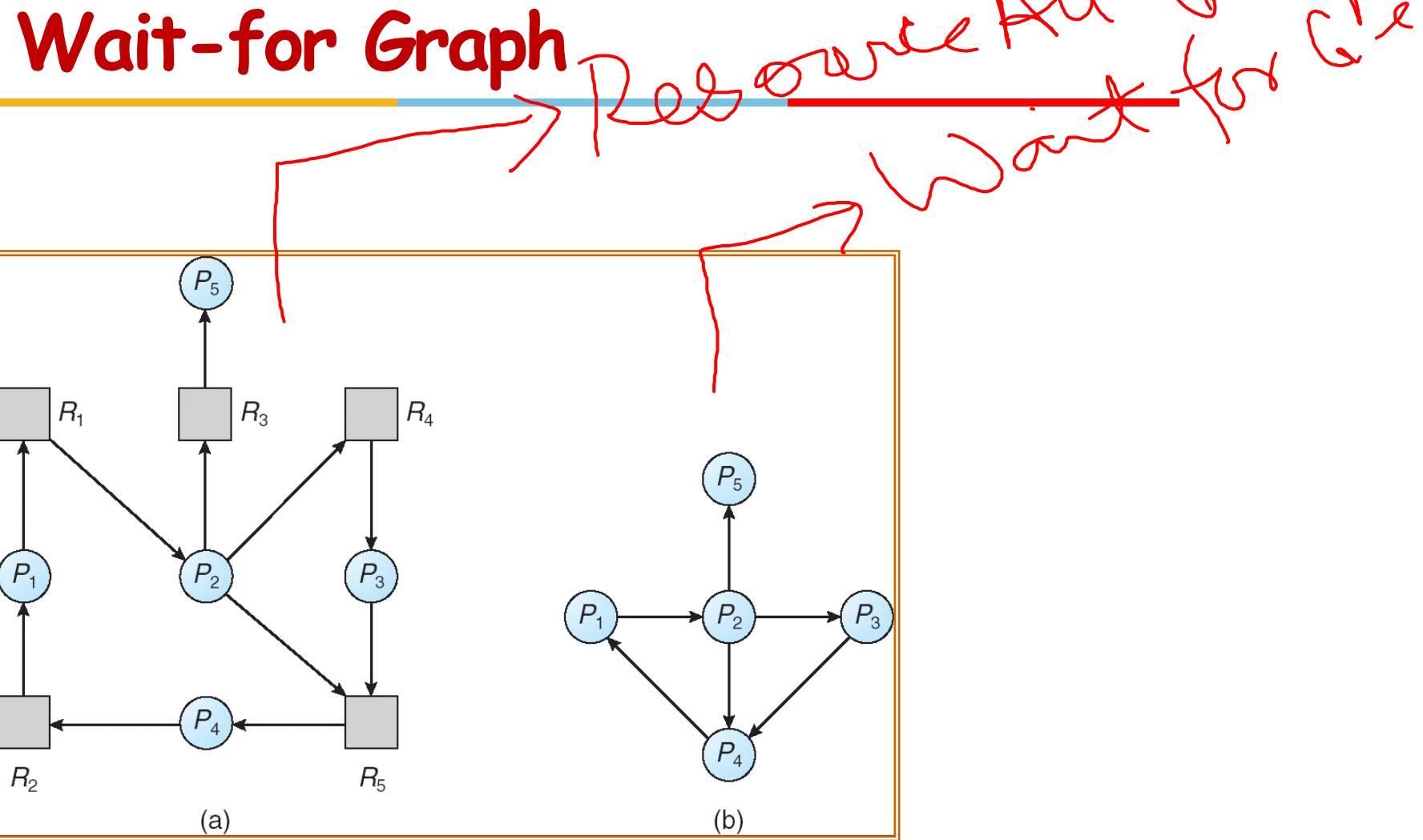
Deadlock Detection

- Allow system to enter deadlock state
 - Detection algorithm
 - Recovery scheme

Single Instance of Each Resource Type

- Maintain wait-for graph
 - Nodes are processes.
 - $P_i \rightarrow P_j$ if P_i is waiting for P_j to release the resource
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

Resource-Allocation Graph and Wait-for Graph



Corresponding wait-for graph

Several Instances of a Resource Type

Available: A vector of length m indicates the number of available resources of each type.

Allocation: An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.

Request: An $n \times m$ matrix indicates the current request of each process. If $\text{Request}[i,j] = k$, then process P_i is requesting k more instances of resource type R_j .

Detection Algorithm

1. Let $Work$ and $Finish$ be vectors of length m and n , respectively
 Initialize:

- (a) $Work = Available$
- (b) For $i = 1, 2, \dots, n$, if $Allocation_i \neq 0$, then
 $Finish[i] = \text{false}$; otherwise,
 $Finish[i] = \text{true}$.

2. Find an index i such that both:

- (a) $Finish[i] == \text{false}$
- (b) $Request_i \leq Work$

If no such i exists, go to step 4.

3. $Work = Work + Allocation$;
 $Finish[i] = \text{true}$
 go to step 2.

4. If $Finish[i] == \text{false}$, for some i , $1 \leq i \leq n$, then the system is in deadlock state.
 Moreover, if $Finish[i] == \text{false}$, then P_i is deadlocked.

Example of Detection Algorithm

Five processes P_0 through P_4 ; three resource types

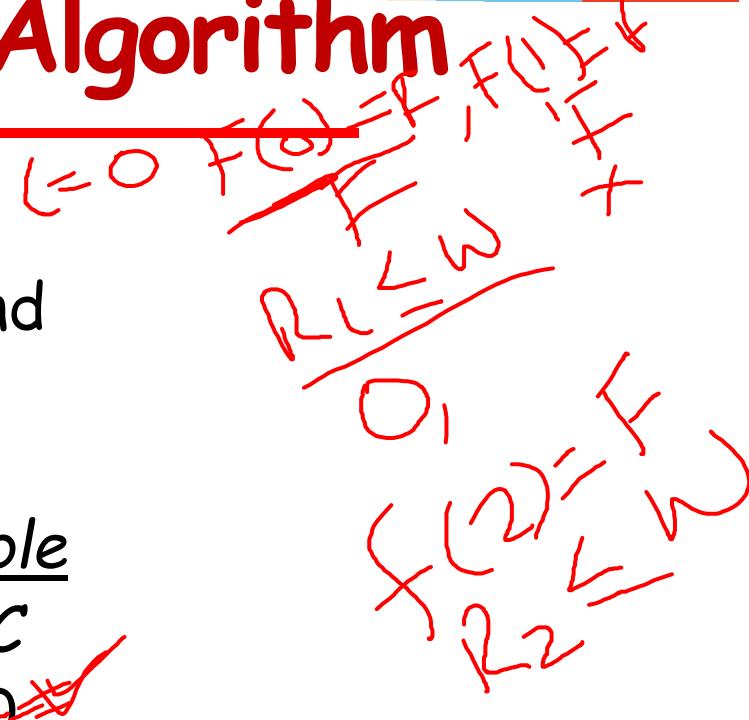
A (7 instances), B (2 instances), and C (6 instances).

Snapshot at time T_0 :

Allocation Request Available

	<u>A</u>	<u>B</u>	<u>C</u>	<u>A</u>	<u>B</u>	<u>C</u>	<u>A</u>	<u>B</u>	<u>C</u>
P_0	✓	✓		0	1	0	0	0	0
P_1	✗	.		2	0	0	0	0	0
P_2	✓	✓		3	0	3	3	3	3
P_3	✓			2	1	1	5	2	4
P_4	✓			0	0	2	5	2	6

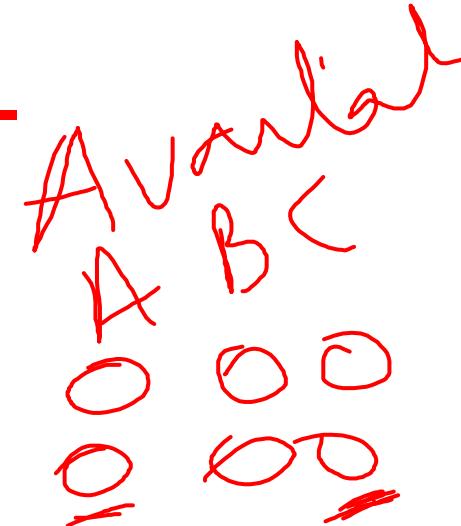
Sequence $\langle P_0, P_2, P_3, P_4, P_1 \rangle$ will result in $\text{Finish}[i] = \text{true}$ for all i .



Example (Cont.)

P_2 requests an additional instance of type C.

	<u>Request</u>		
	A	B	C
P_0	0	0	0
P_1	2	0	1
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2



State of system?

- Can reclaim resources held by process P_0 but insufficient resources to fulfill other processes' requests.
- Deadlock exists, consisting of processes P_1, P_2, P_3 , and P_4 .

Recovery from Deadlock: Process Termination

Abort all deadlocked processes.

Abort one process at a time until the deadlock cycle is eliminated.

In which order should we choose to abort?

- Priority of the process.
- How long process has computed, and how much longer to completion.
- Resources the process has used.
- Resources process needs to complete.
- How many processes will need to be terminated.
- Is process interactive or batch?



Memory Management

Memory Management Requirements



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

Memory Management

Requirement 1: Relocation

- Program must be brought into memory and placed within a process for it to be executed
- *Input queue* - collection of processes on the disk that are waiting to be brought into memory to run the program.
- The programmer does not know where the program will be placed in memory when it is executed,
 - it may be swapped to disk and return to main memory at a different location (relocated)
- Memory references must be translated to the actual physical memory address
- **Address binding** : Mapping from one address space to another

Logica
Physical

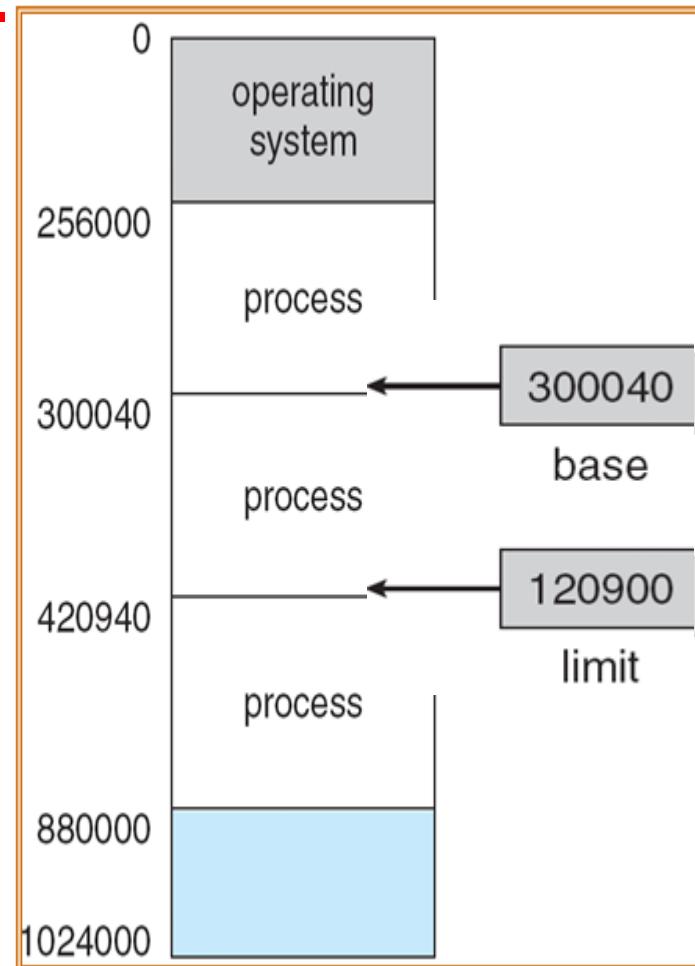
Memory Management Requirement 1: Relocation



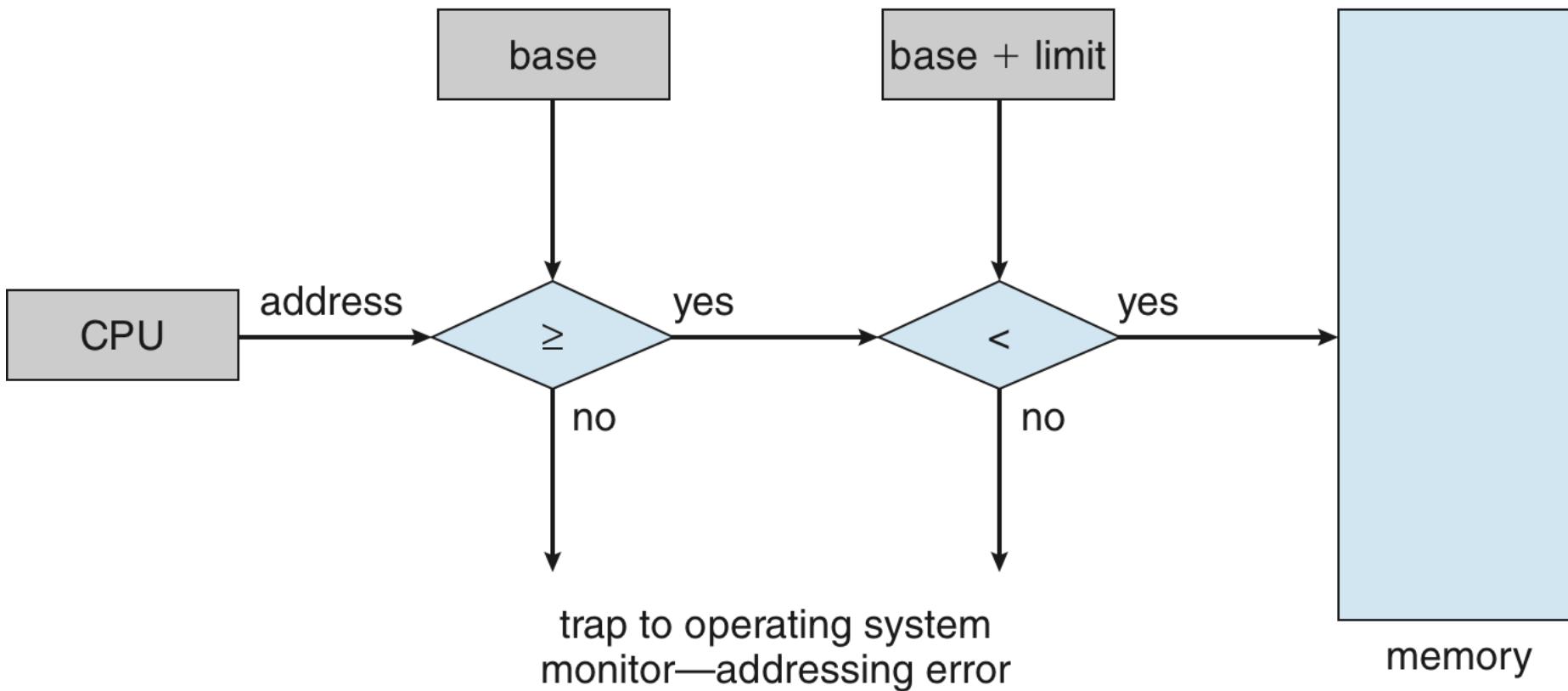
- Address binding of instructions and data to memory addresses can happen at three different stages.
 - **Compile time:** If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.
 - **Load time:** Compiler must generate relocatable code if memory location is not known at compile time.
 - Final address binding will be done at load time
 - If starting address changes then reload the program
 - **Execution time:** Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers).

Memory Management Requirement 2: Protection

- Each process has a separate memory space
 - security ?
- provide security using 2 registers
 - base register
 - limit register
- The base register holds the smallest legal physical memory address; the limit register specifies the size of the range.
- Need a hardware for address comparison



Hardware Support for Relocation and Limit Registers



Points to be noted

- Illegal access results in trap to OS → fatal error
- Special privileged instructions are used to load base and limit registers
- The operating system has unrestricted access to both operating system and users' memory.

Memory Management

Requirement 3: Sharing

- Allow several processes to access the same portion of memory
- Better to allow each process access to the same copy of the program rather than have their own separate copy

Memory Management Requirement4: Logical Organization

- Memory is organized linearly (usually)
- Programs are written in modules
 - Modules can be written and compiled independently
 - Different degrees of protection given to modules (read-only, execute-only)
 - Share modules among processes
- Logical Organization
- Segmentation helps here

Memory Management Requirement 5: Physical Organization



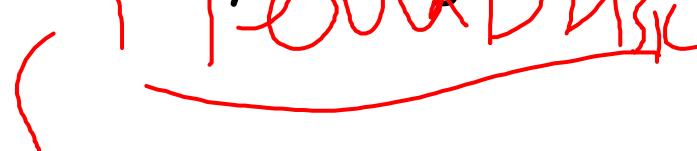
- Two Level organization : Main memory and Secondary Memory

- Main memory : Faster access at relatively high cost, volatile, smaller capacity

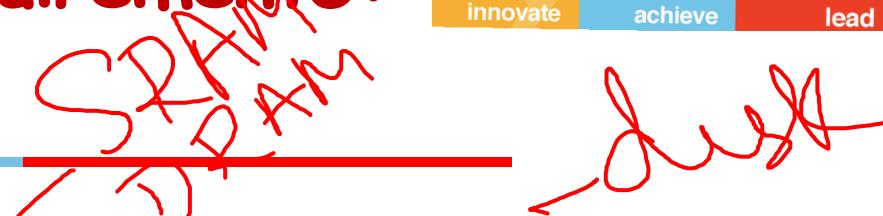
- Secondary memory: Slower, cheaper, nonvolatile, large capacity

- Cannot leave the programmer with the responsibility to manage memory

- Memory Management Unit



Secondary



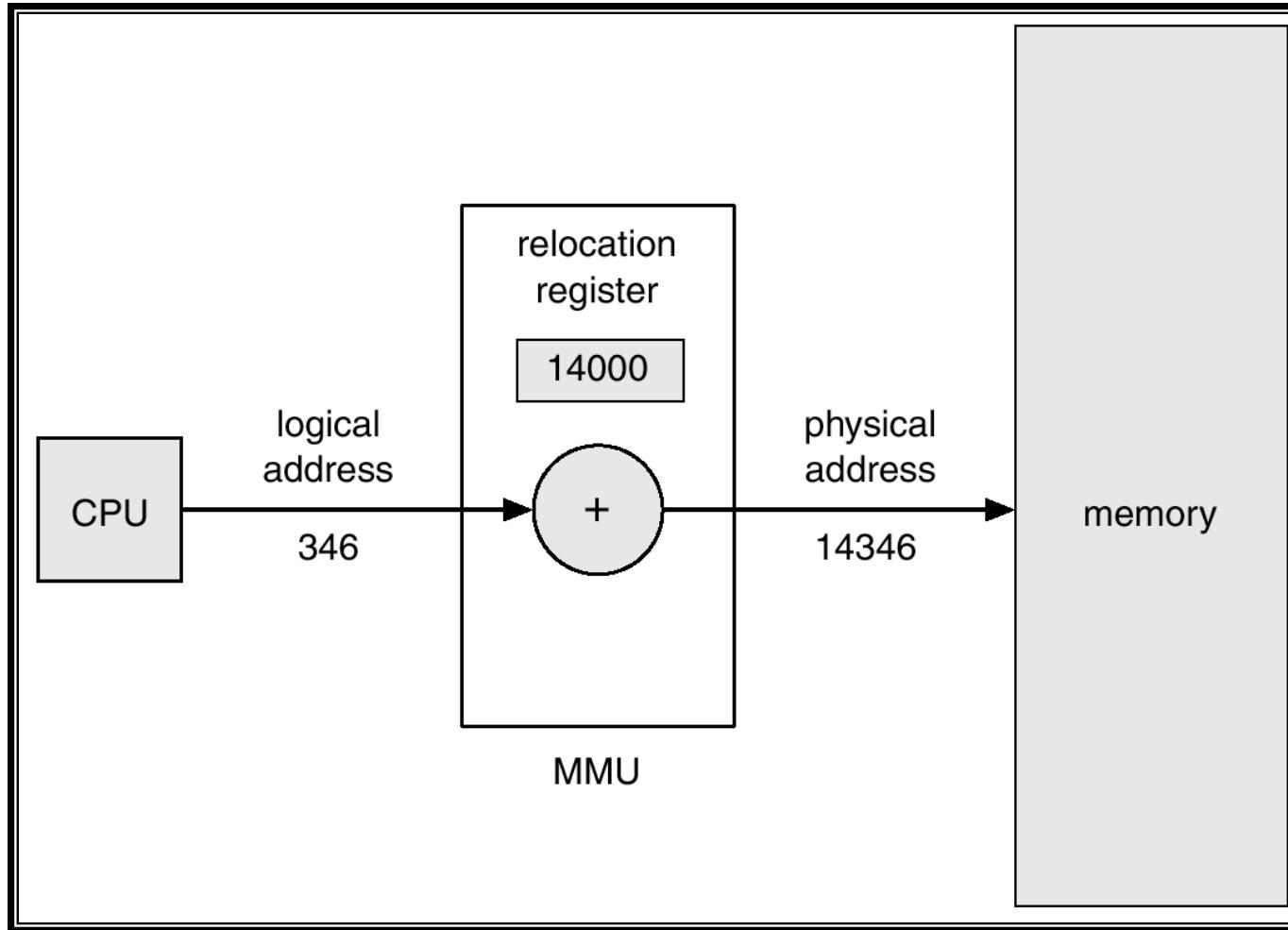
Logical vs. Physical Address Space

- *Logical address* - generated by the CPU; also referred to as *virtual address*.
- *Physical address* - MAR register- address seen by the memory unit.

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address.
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.
- Two different types of addresses:
 - Logical: range 0 to max.
 - Physical: range $R+0$ to $R+max$; where R is a relocation register value.
- Note: The user generates only logical addresses and thinks that the process runs in locations 0 to max.
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.

Dynamic relocation using a relocation register





BITS Pilani
Pilani Campus

COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS SESSION 14

Prepared by: Dr. Lucy J. Gudino
WILP & Department of CS & IS

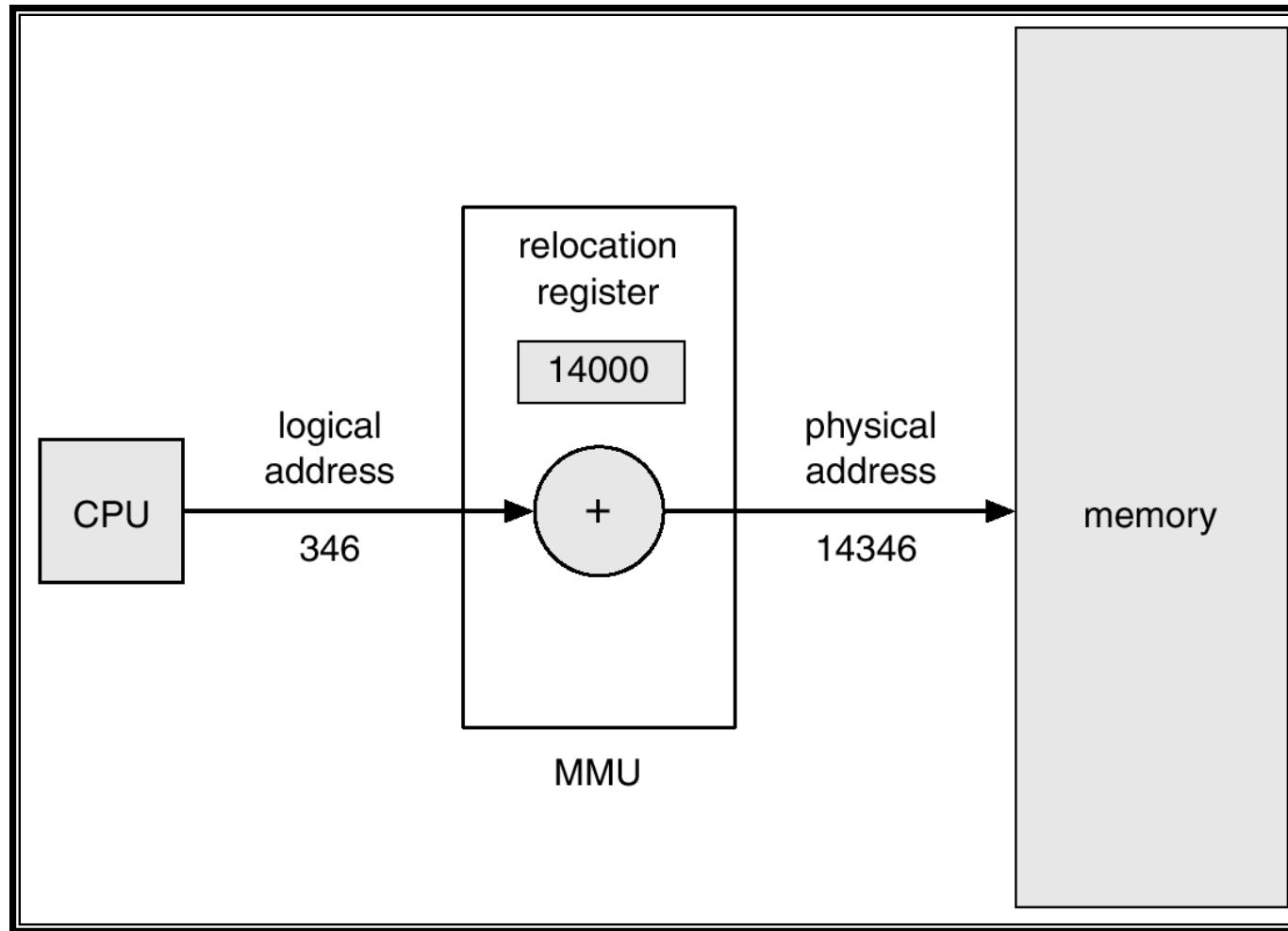
Today's Session

List of Topic Title	Text/Ref Book/external resource
• Memory Management...	T2

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address.
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.
- Two different types of addresses:
 - Logical: range 0 to max.
 - Physical: range $R+0$ to $R+max$; where R is a relocation register value.
- Note: The user generates only logical addresses and thinks that the process runs in locations 0 to max.
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.

Dynamic relocation using a relocation register



Memory Management Techniques



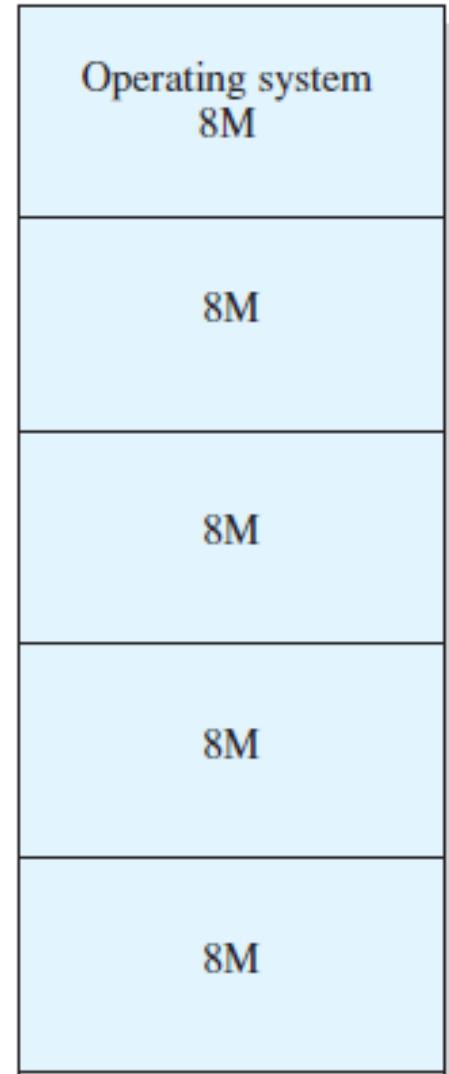
- Fixed Partitioning
- Dynamic Partitioning
- Simple Paging
- Simple Segmentation
- Virtual Memory Paging
- Virtual Memory Segmentation

Fixed partition

- Main memory is divided into a number of static partitions at system generation time.
- A process may be loaded into a partition of equal or smaller size.
- Simple to implement
- The degree of multiprogramming is bound by the number of partitions
- Little operating system overhead
- EX: IBM OS/360 → MFT : Multiprogramming with Fixed Number of Tasks

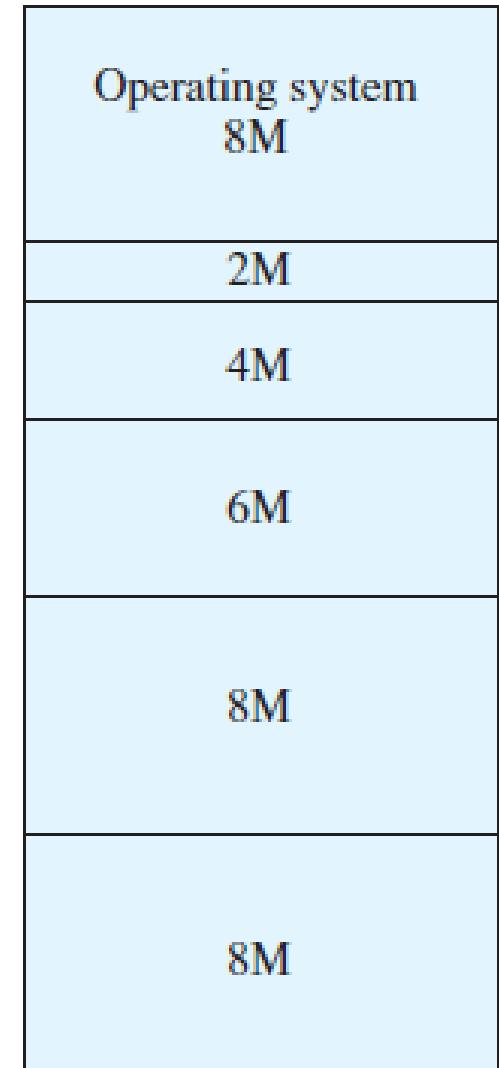
Fixed partition (Contd...)

- Two difficulties:
 - A program may be too big to fit into a partition :
 - Use overlays
 - Inefficient main memory utilization
→ Internal Fragmentation
- which process to swap out is based on scheduling



Fixed partition (Contd...)

- Unequal size partitions
 - provides a degree of flexibility
- Two ways to assign processes to partitions
 - One process queue per partition
 - Single queue



Fixed partition (Contd...)

- Unequal size partitions : Pros and Cons :
 - One process queue per partition
 - Lesser internal fragmentation
 - Not optimum solution
 - Single queue
 - Optimum
- Disadvantages of fixed partition:
 - The number of partitions specified at system generation time limits the number of active (not suspended) processes in the system.
 - it is an inefficient technique: small jobs will not utilize partition space efficiently

Example : Assembler

- Two passes : Pass 1 and Pass 2
 - Pass 1: Constructs Symbol Table ✓
 - Pass 2: Generates Machine code ✓



	Size
Pass 1	70K
Pass 2	80K
Symbol Table	25K
Common Routines	25K
Total	200K

Handwritten annotations in red:

- A red circle highlights the value "70K" under "Pass 1".
- A red circle highlights the value "80K" under "Pass 2".
- A red circle highlights the value "200K" under "Total".
- Red numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 are drawn around the table, with arrows pointing from some of them to the circled values.

Example : Assembler

- Overlay A: symbol table, common_routines, and Pass1.
- Overlay B: symbol table, common routines, and Pass2.
- Add overlay driver 10k

~~70K~~ (130)

~~25K~~

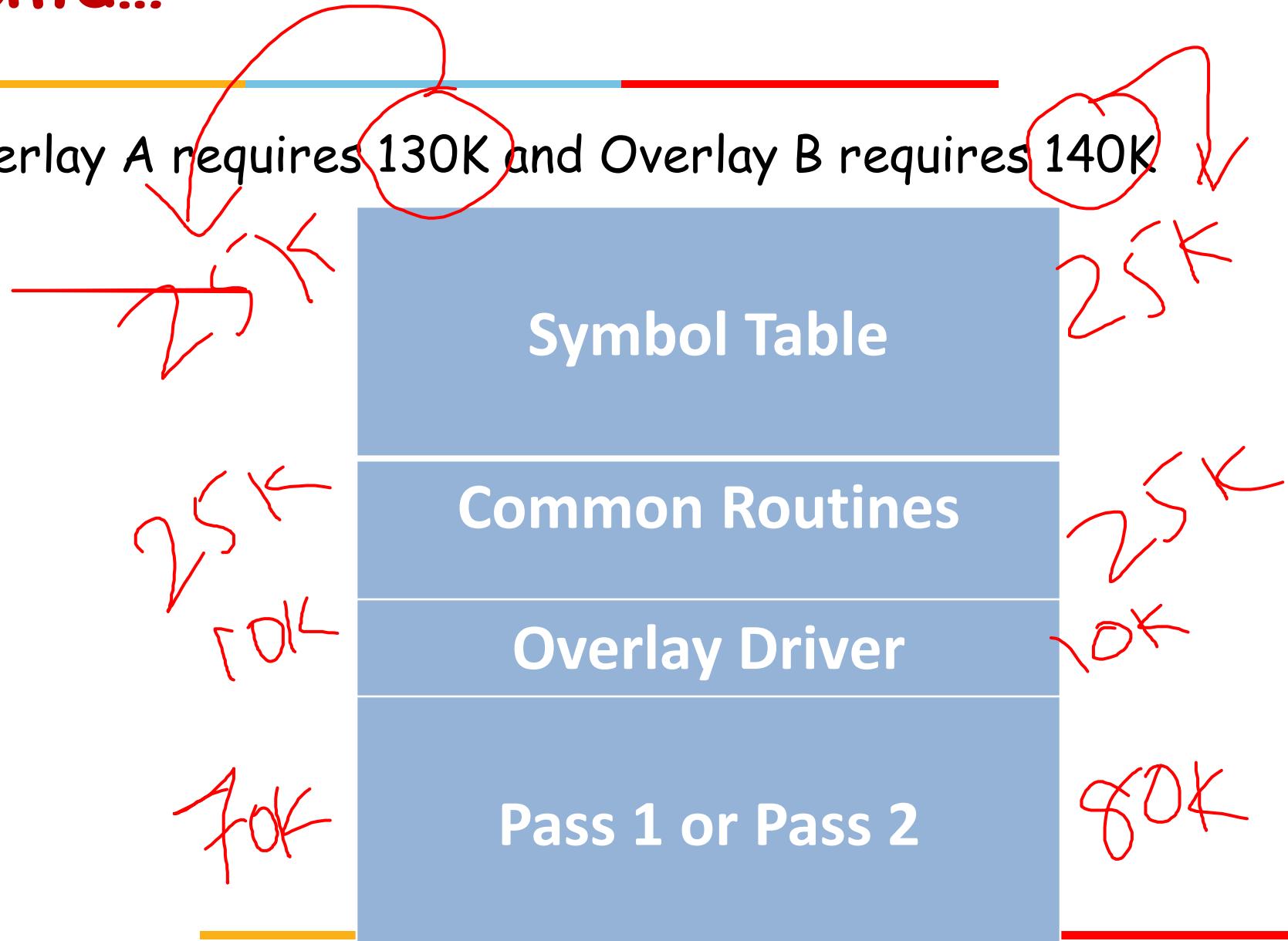
~~44K~~ (46K)

-

	Size
Pass 1	70K
Pass 2	80K
Symbol Table	<u>25K</u>
Common Routines	<u>25K</u>
Total	<u>200K</u>

Contd...

Overlay A requires 130K and Overlay B requires 140K

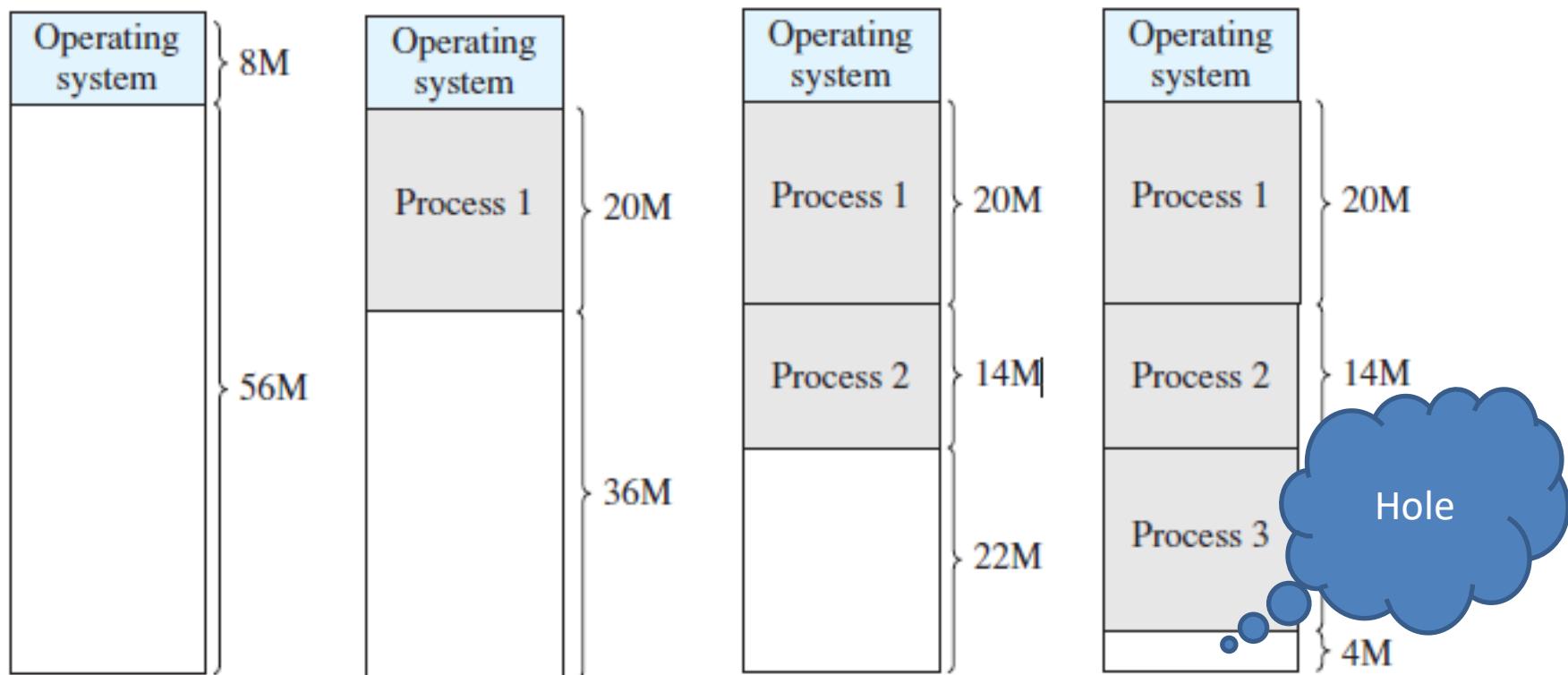


Overlays : Summary

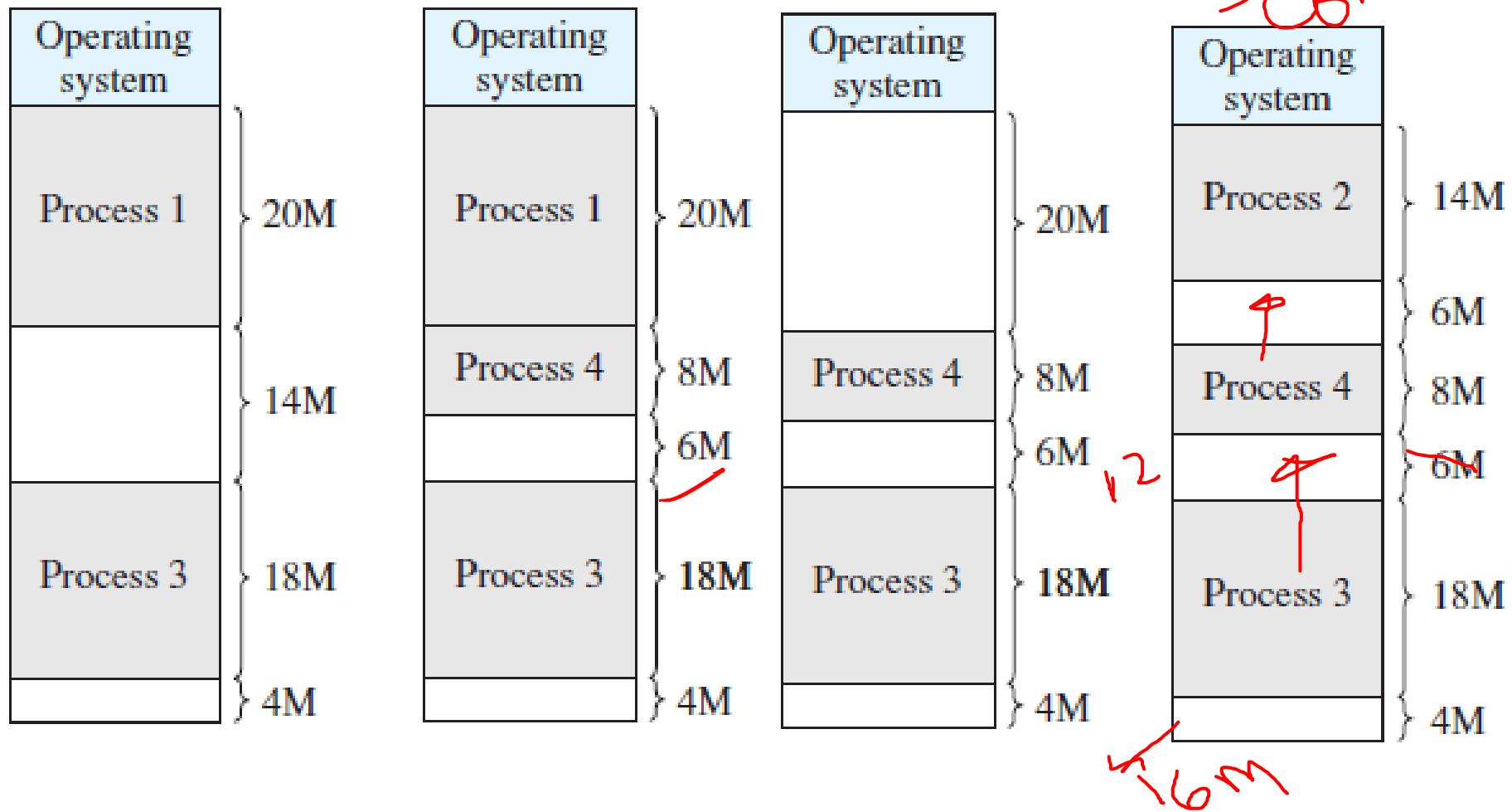
- If a process is larger than the amount of memory, a technique called overlays can be used.
- Overlays is to keep in memory only those instructions and data that are needed at any given time.
- When other instructions are needed, they are loaded into space that was occupied previously by instructions that are no longer needed.
- Overlays are implemented by user, no special support needed from operating system, programming design of overlay structure is complex.

Dynamic Partitioning or Variable partition allocation

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

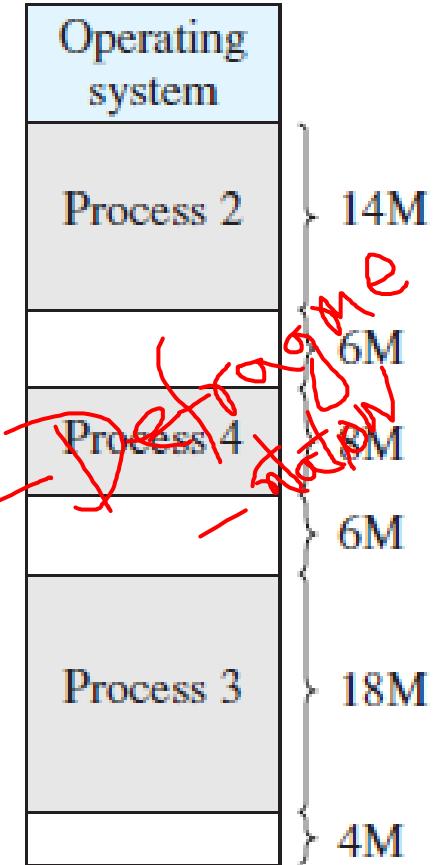


Contd...



Cont.

- Hole - block of available memory;
- When a process arrives, it is allocated memory from a hole large enough to accommodate it.
- Holes of various size are scattered throughout memory.
- External Fragmentation
- Compaction: technique to overcome external fragmentation
- Compaction is a time consuming process and wasteful of processor time
- Operating system maintains information about
 - a) allocated partitions b) free partitions (hole)

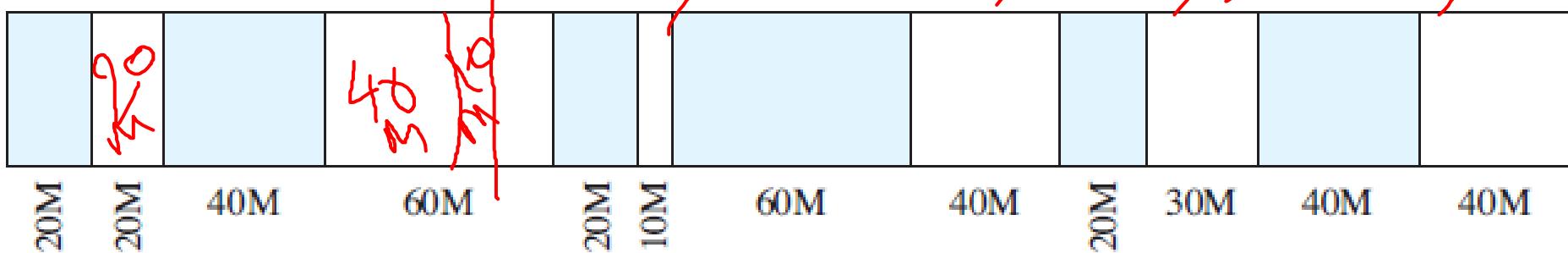


Dynamic Storage-Allocation

- 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 1 - First Fit

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



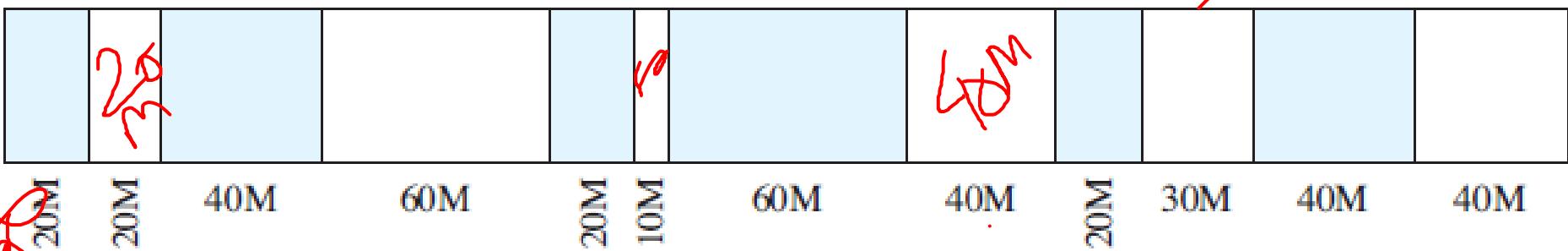
The shaded areas are allocated blocks; the white areas are free blocks. The next three memory requests are for 40M, 20M, and 10M. Indicate the starting address for each of the three blocks using the following placement algorithms:

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

Problem : Best fit

130/3

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



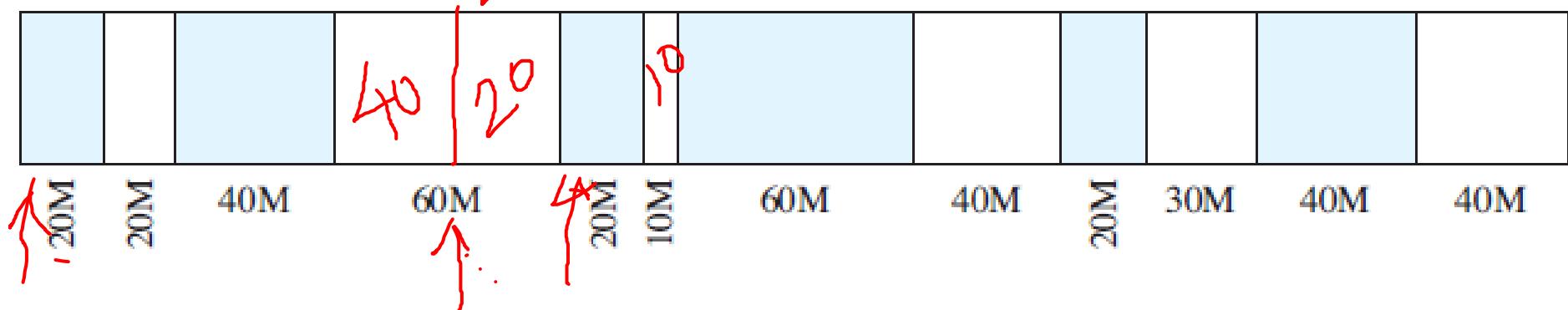
The shaded areas are allocated blocks; the white areas are free blocks. The next three memory requests are for 40M, 20M, and 10M. Indicate the starting address for each of the three blocks using the following placement algorithms:

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

Problem - Next Fit

~~30%~~

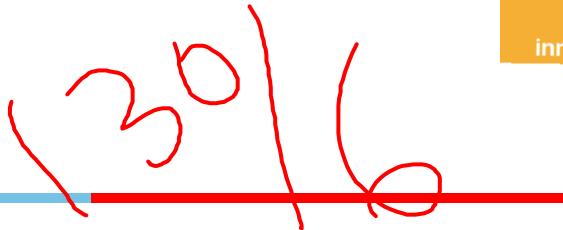
A dynamic partitioning scheme is being used, and the following is the memory configuration at a given point in time:



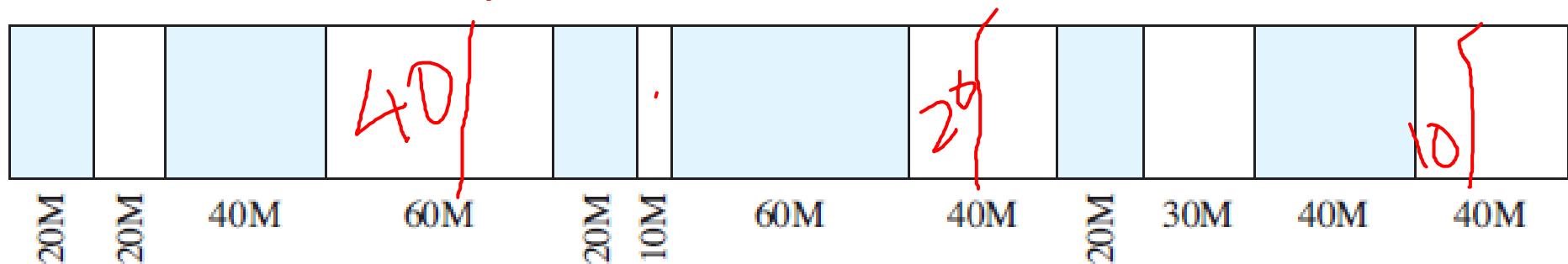
The shaded areas are allocated blocks; the white areas are free blocks. The next three memory requests are for 40M, 20M, and 10M. Indicate the starting address for each of the three blocks using the following placement algorithms:

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

Problem: worst fit



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

Problem 2 (HW)

→ DISCUSSED

Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)?



300KB 600KB 350KB 200KB 750KB 125KB

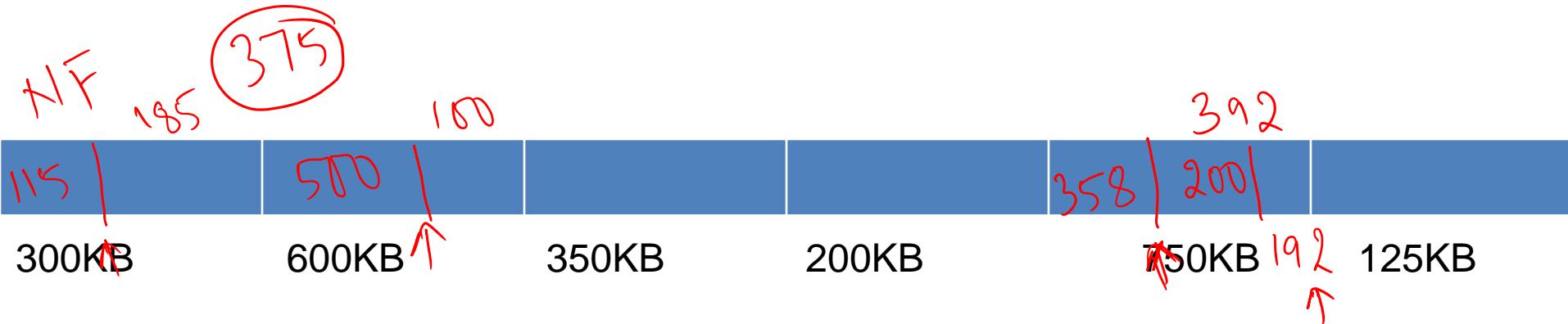
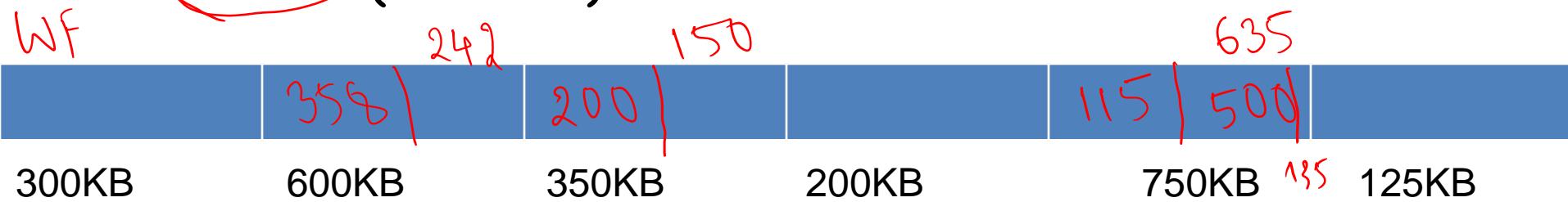


300KB 600KB 350KB 200KB 750KB 125KB



Problem (Contd...)

Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and ~~worst-fit~~ algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)?



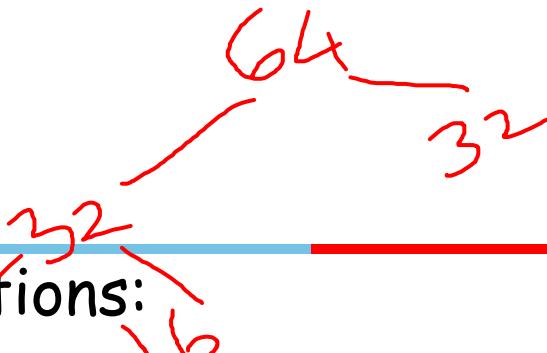
Fragmentation

- **fragmentation** is a phenomenon in which storage space is used inefficiently, reducing capacity and often performance.
 - **External Fragmentation** - total memory space exists to satisfy a request, but it is not contiguous.
 - **Internal Fragmentation** - allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
- Reduce external fragmentation by compaction
 - Shuffle memory contents to place all free memory together in one large block.
 - Compaction is possible only if relocation is dynamic, and is done at execution time.

Buddy System

- The **buddy system** is a memory allocation and management algorithm
- It manages memory in **power of two** increments.
- Splitting memory into halves and to try to give a best fit

Contd...



- Provides two operations:
 - ~~Allocate(A, 2^k)~~: Allocates a block of 2^k and marks it as allocated
 - Free(A): Marks the previously allocated block A as free and merge it with other blocks to form a larger block
- Algorithm: Assume that a process A of size "X" needs to be allocated
 - If $2^{k-1} < X \leq 2^k$: Allocate the entire block 2^k
 - Else: Recursively divide the block equally and test the condition at each time, when it satisfies, allocate the block.

Problem 1

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)



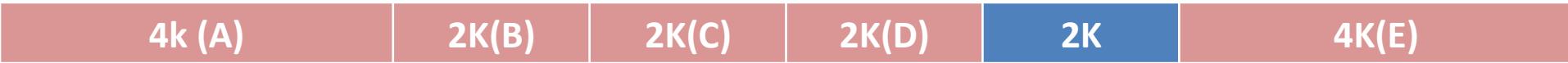
- Allocate (D: 1.9K)



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)

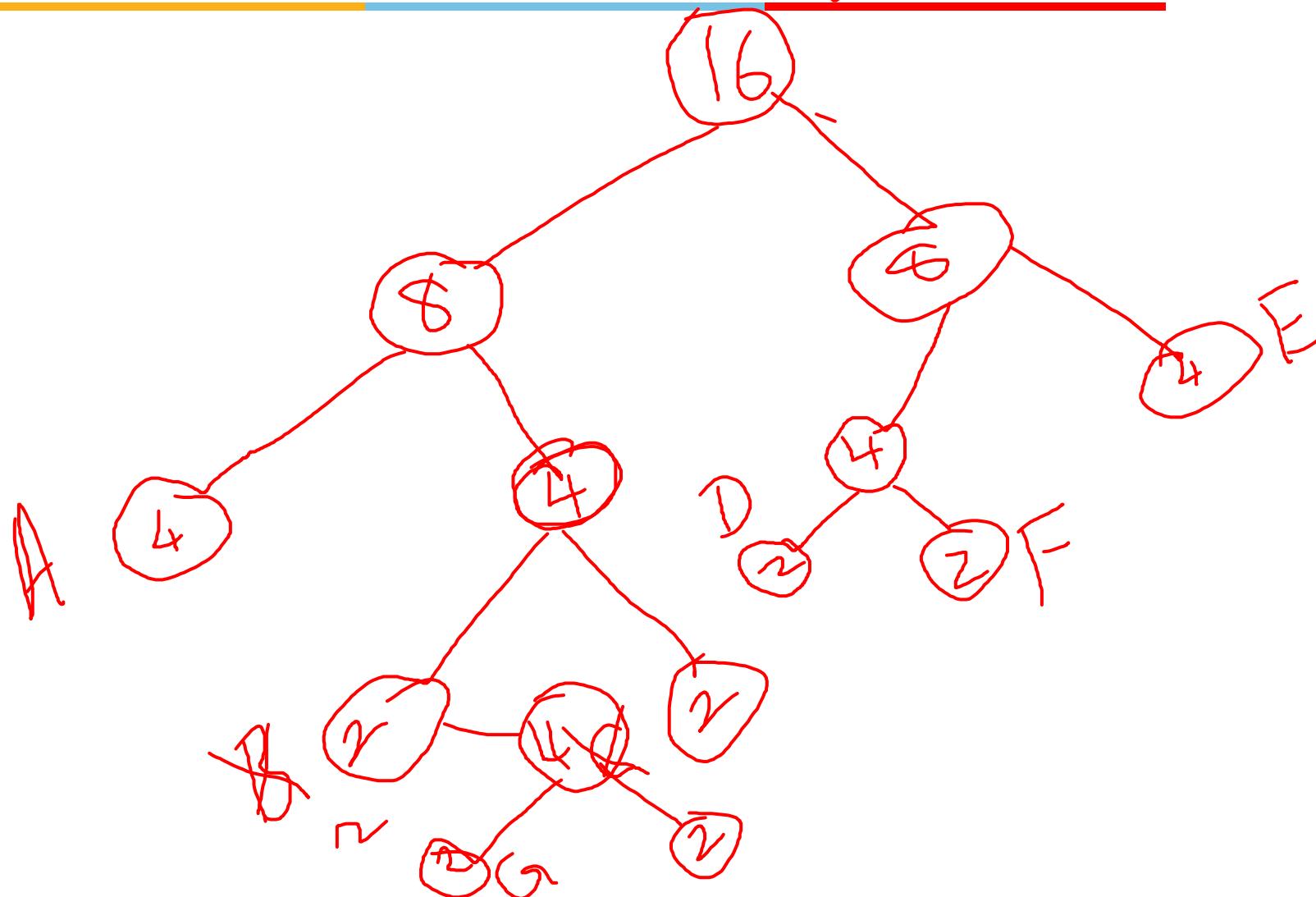


- Allocate (G: 1.8K)

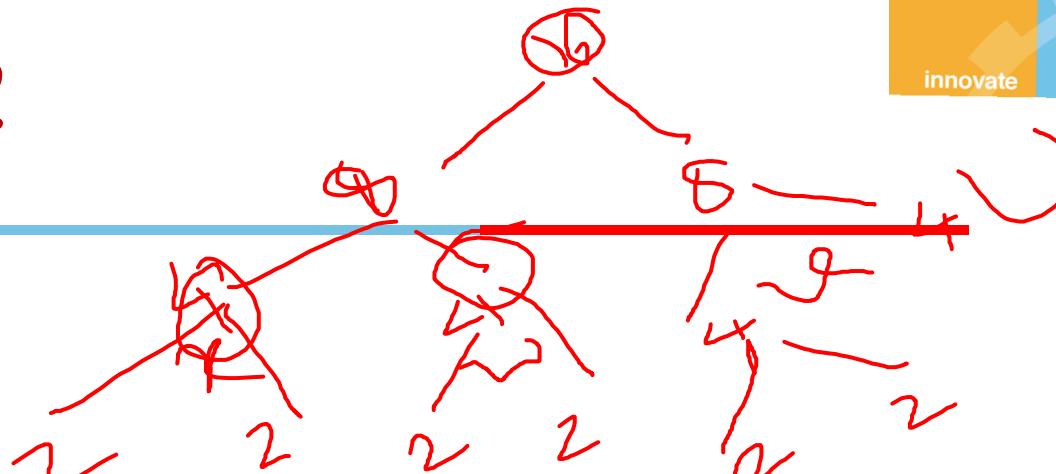


Tree Structure

- 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)



Problem 2



- Consider the state of memory after allocating 5 processes A, B, C, D and E



- What is the state of memory after freeing process D?



- What is the state of memory after freeing Process C?



Advantages and Disadvantages

Advantage -

- Easy to implement a buddy system (Linux)
- Allocates block of correct size
- It is easy to merge adjacent holes
- Fast to allocate memory and de-allocating memory

Disadvantage -

- It requires all allocation unit to be powers of two
- It leads to internal fragmentation

Important Note

- Fixed size and dynamic partitions are inefficient
 - Fixed size → internal fragmentation
 - Dynamic → external fragmentation
- Paging helps to reduce internal fragmentation and completely eliminates external fragmentation

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 addresses.

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	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(a) Fifteen available frames

	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

(b) Load process A

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

(c) Load process B

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

(d) Load process C

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

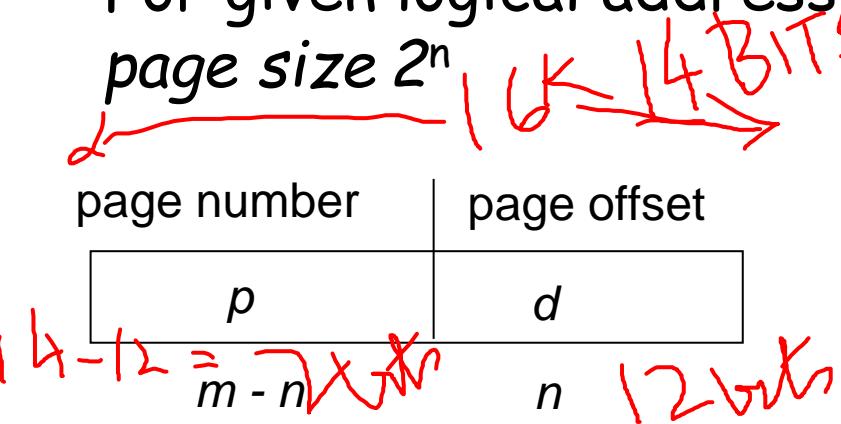
(e) Swap out B

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

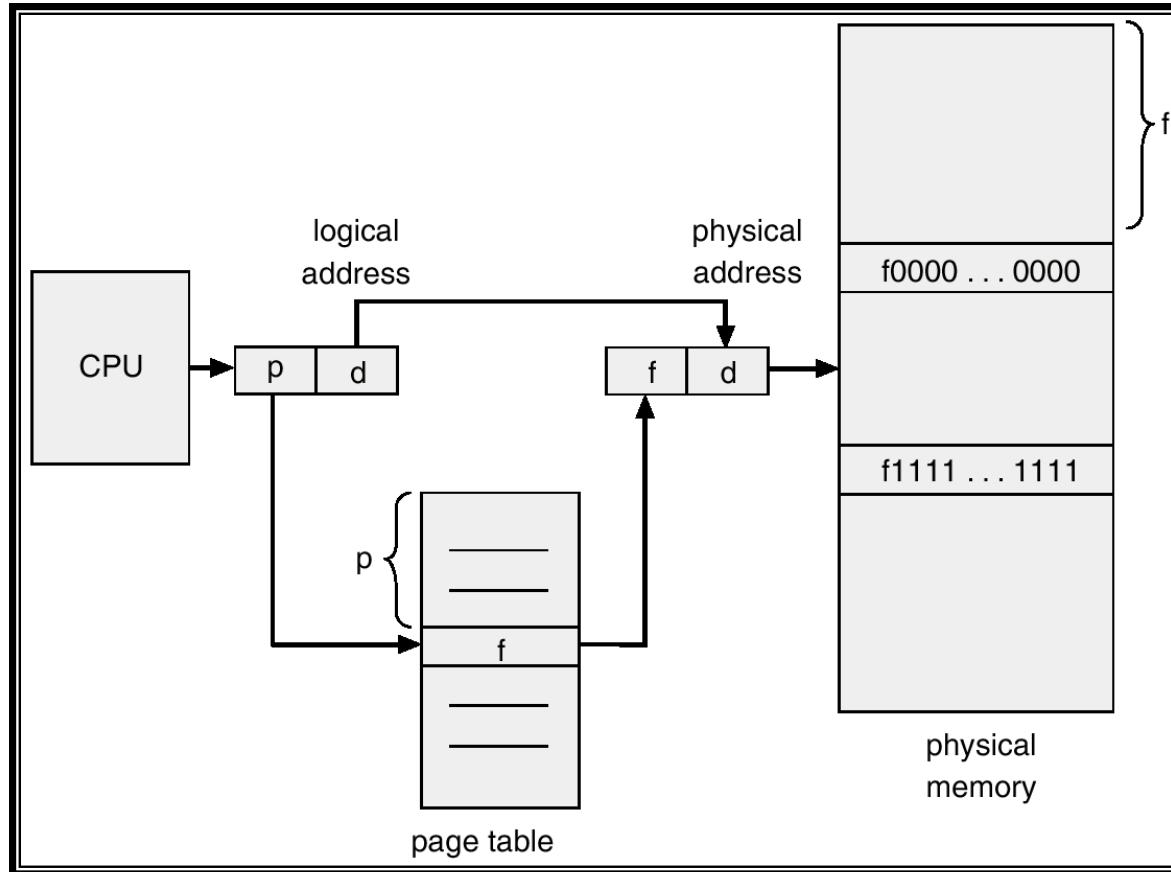
(f) Load process D

Address Translation Scheme

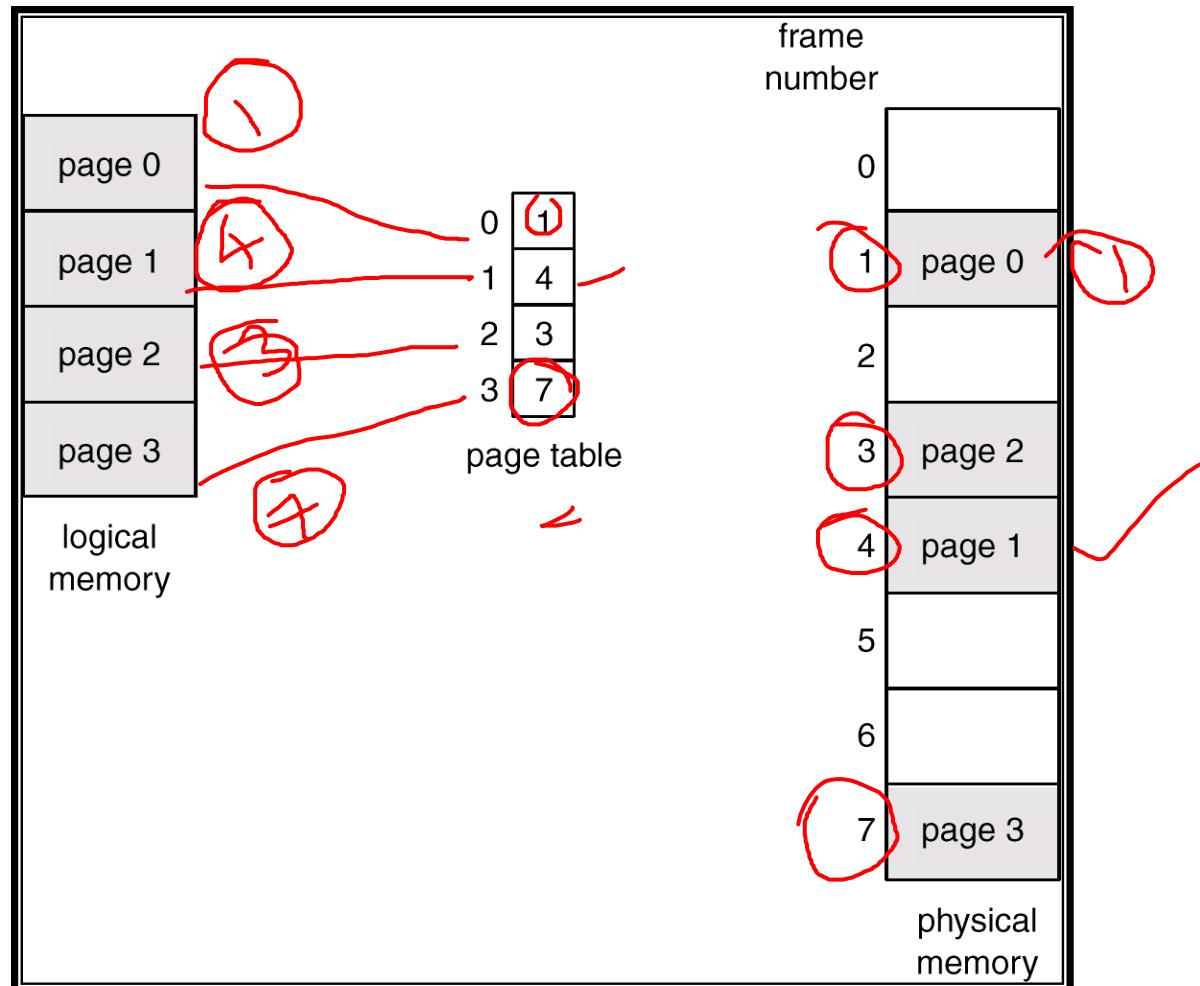
- Address generated by CPU is divided into:
 - Page number (p) - used as an index into a page table. Page table contains base address of each page in physical memory.
 - Page offset (d) - combined with base address to define the physical memory address that is sent to the memory unit.
 - For given logical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



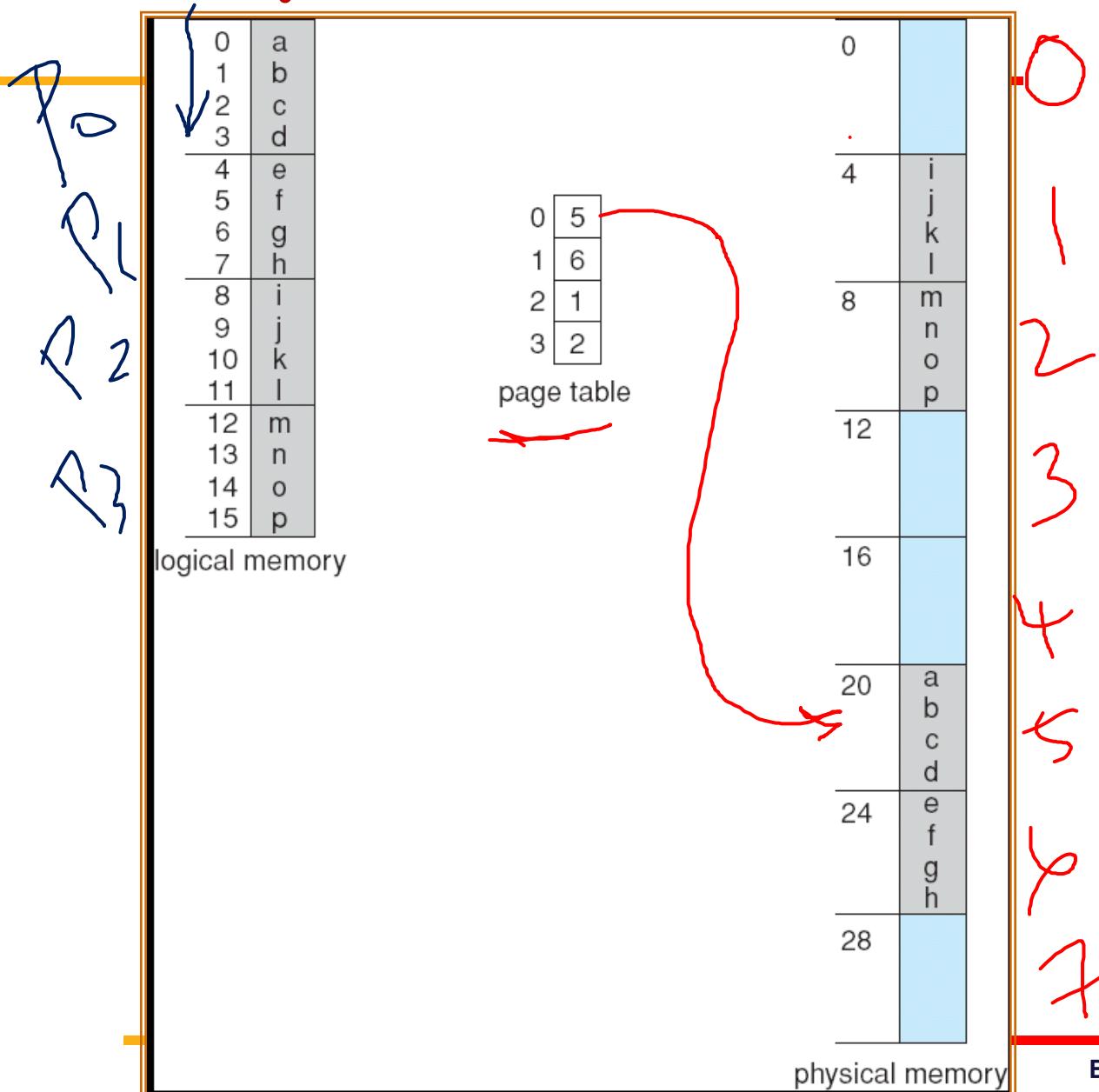
Address Translation Architecture



Paging Example



Paging Example



Important points....

- Paging is a form of dynamic relocation
- No external fragmentation but may have some internal fragmentation
- Small or large page size ?
- Page size determines
 - Memory wastage due to internal fragmentation
 - Size of the page table for a process
 - disk I/O data transfer
- small page : increases paging table overhead !!!!, internal fragmentation decreases
- large page : Decreases paging table overhead, internal fragmentation increases, disk I/O is more efficient when the number of data being transferred is larger

Contd...

- page size is usually 4 KB to 8 KB
 - Needs to maintain the allocation details of main memory → frame table
-

Problem

Consider a logical address space of 64 pages of 1,024 words each, mapped onto a physical memory of ~~32~~ frames.

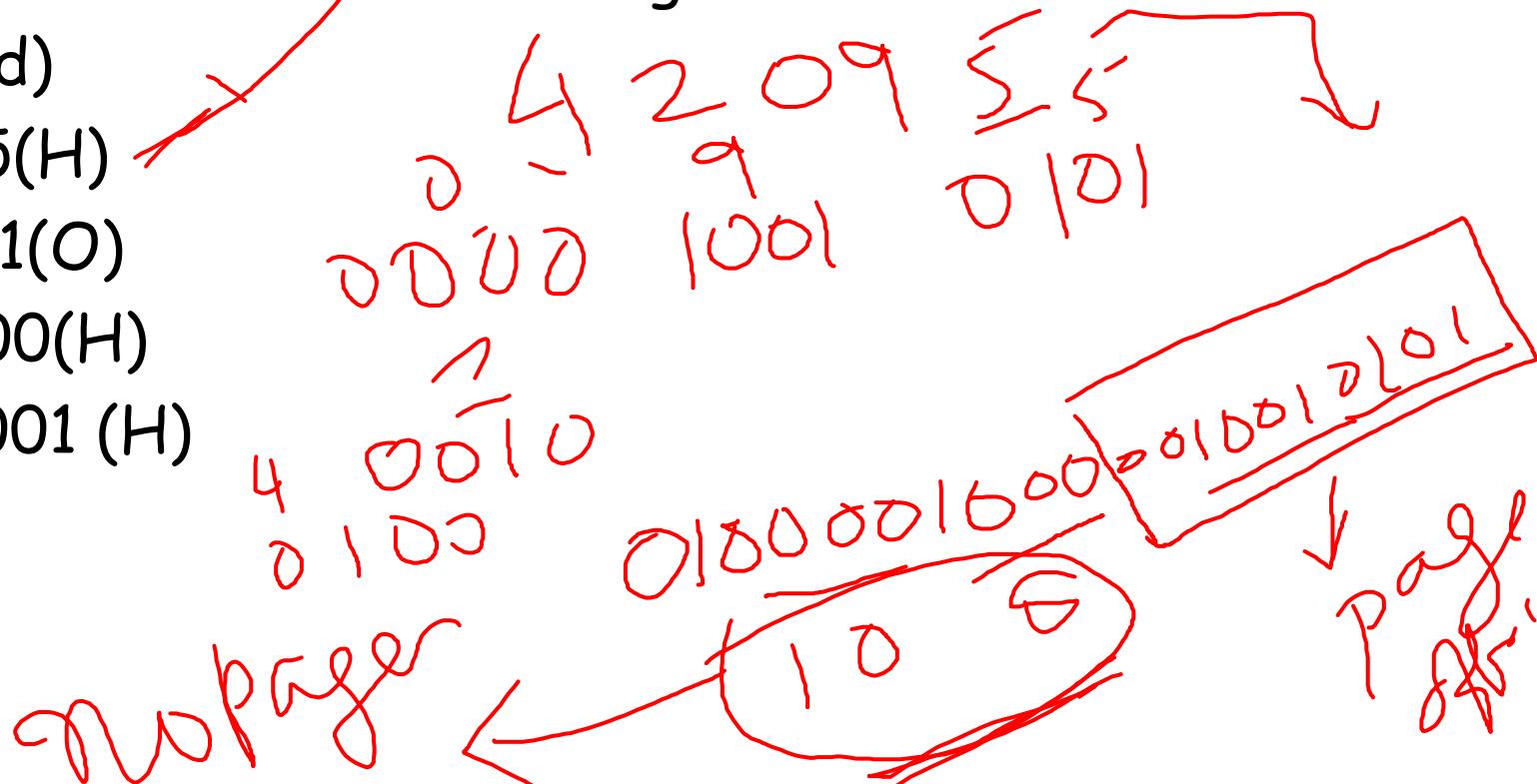
- a. How many bits are there in the logical address? ~~16~~
- b. How many bits are there in the physical address? ~~15~~

$32 - 1$ { bits
 10 bits frame = page size
 10 bits offset, subframe

Problem

Assuming a 1-KB page size, what are the page numbers and offsets for the following address references

- a. 3085(d)
- b. 42095(H)
- c. 215201(O)
- d. 650000(H)
- e. 2000001 (H)





Virtual Memory

Virtual Memory

What is the motivation behind Virtual Memory?

- Programs often have code to handle unusual error conditions which is almost never executed.
- Arrays, lists, and tables are often allocated more memory than they actually need.
- Certain options and features of a program may be used rarely
- Another motivation is Principle of locality.
- trashing is a condition where system spends more time in swapping than executing instructions

Background

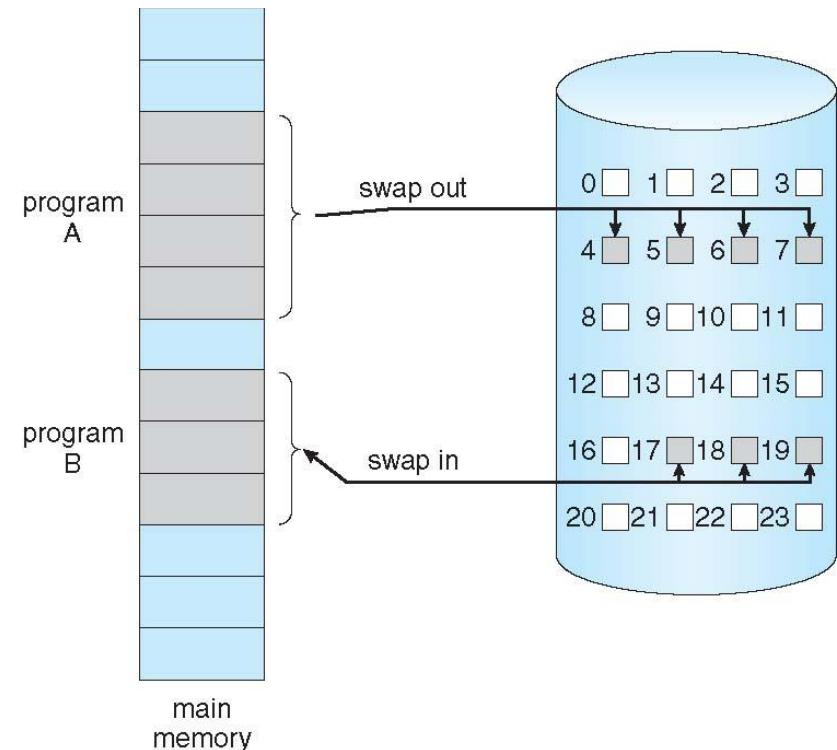
- ❖ **Virtual memory** - separation of user logical memory from physical memory
 - ❖ Only part of the program needs to be in memory for execution
 - ❖ Logical address space can therefore be much larger than physical address space
 - ❖ Allows address spaces to be shared by several processes
 - ❖ More programs running concurrently
 - ❖ Less I/O needed to load or swap processes

Virtual address space - logical view of how process is stored in memory

- Usually start at address 0, contiguous addresses until end of space
- Meanwhile, physical memory organized in page frames
- MMU must map virtual to physical

Demand Paging

- ❖ Bring a page into memory only when it is needed
 - ❖ Less I/O needed, no unnecessary I/O
 - ❖ Less memory needed
 - ❖ Faster response
 - ❖ More users
- ❖ Page is needed \Rightarrow reference to it
 - ❖ invalid reference \Rightarrow abort
 - ❖ not-in-memory \Rightarrow bring to memory
- ❖ **Lazy Swapper, Pager**



Demand Paging

- ❖ Pager guesses which pages will be used before swapping out again
- ❖ Pager brings in only those pages into memory
- ❖ How to determine that set of pages?
 - ❖ Need new MMU functionality to implement demand paging
 - ❖ If pages needed are already **memory resident**
 - ❖ If page needed and not memory resident
 - ❖ Need to detect and load the page into memory from storage

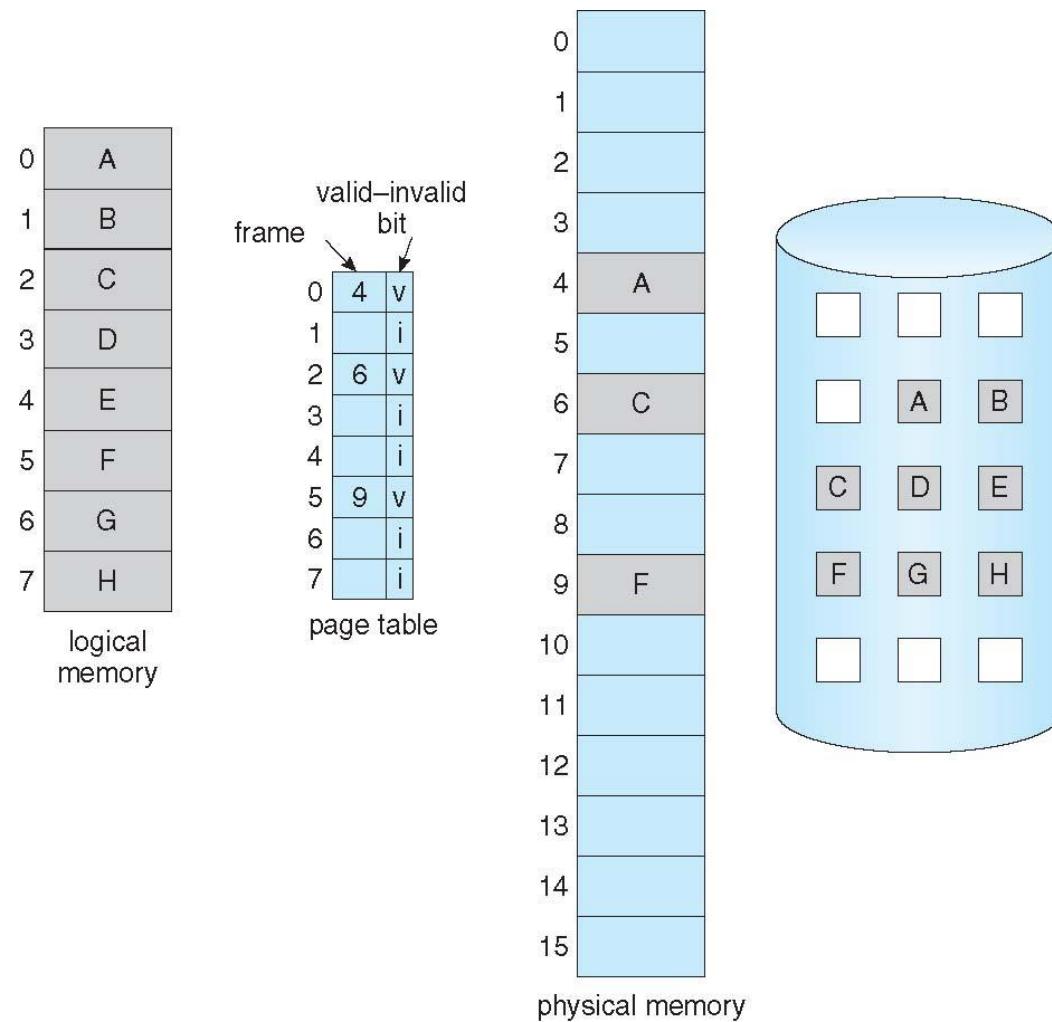
Valid-Invalid Bit

- ❖ With each page table entry a valid-invalid bit is associated
- ❖ **v** \Rightarrow legal and in-memory - **memory resident**
- ❖ **i** \Rightarrow either illegal or not-in-memory
- ❖ Initially valid-invalid bit is set to **i** on all entries
- ❖ During MMU address translation, if valid-invalid bit in page table entry is **i** \Rightarrow page fault

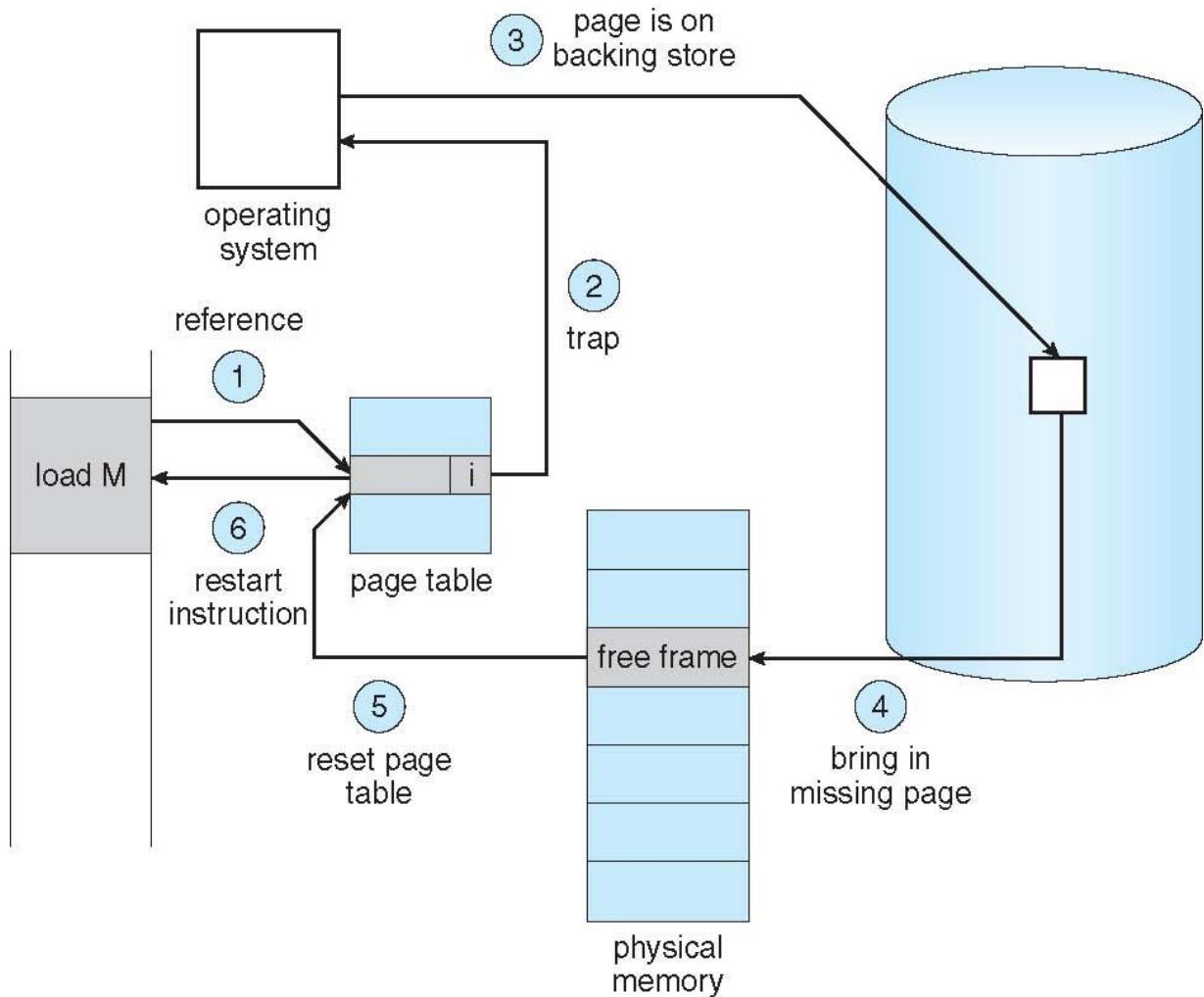
Frame #	valid-invalid bit
	v
	v
	v
	i
...	
	i
	i

page table

Valid-Invalid Bit



Page Fault



Page Fault

- ❖ If there is a reference to a page, first reference to that page will trap to operating system: **page fault**
 - ❖ Operating system looks at an internal table (usually associated with PCB) to decide:
 - ❖ Invalid reference \Rightarrow abort
 - ❖ Just not in memory
 - ❖ Find free frame
 - ❖ Swap page into frame via disk operation
 - ❖ Reset page table to indicate page now in memory
Set valid/invalid bit = **v**
 - ❖ Restart the instruction that caused the page fault
-

Demand Paging

- ❖ Pure Demand Paging
- ❖ Locality of reference - tendency of a processor to access the same set of memory locations repetitively over a short period of time
- ❖ Hardware support needed for demand paging
 - ❖ Page table with valid / invalid bit
 - ❖ Secondary memory (swap device with **swap space**)
 - ❖ Instruction restart

Page Replacement

- ❖ Use **modify (dirty) bit** to reduce overhead of page transfers
 - only modified pages are written to disk
- ❖ Page replacement completes separation between logical memory and physical memory - large virtual memory can be provided on a smaller physical memory
- ❖ Find the location of the desired page on disk
- ❖ Find a free frame:
 - If there is a free frame, use it
 - If there is no free frame, use a page replacement algorithm to select a **victim frame**
 - Write victim frame to disk if dirty
- ❖ Bring the desired page into the (newly) free frame; update the page and frame tables
- ❖ Continue the process by restarting the instruction that caused the page fault

Problem

Assuming a 1-KB page size, what are the page numbers and offsets for the following address references

- a. 3085(d)
- b. 42095(H)
- c. 215201(O)
- d. 650000(H)
- e. 2000001 (H)

Logical address (decimal)	Logical address (binary)	Page # (22 bits) (binary)	Offset (10 bits) (binary)	Page # decimal	Offset decimal
3085	0000000000000000000110000001101	000000000000000000011	0000001101	3	13
42095	00000000000000001010010001101111	0000000000000000101001	0001101111	41	111
215201	000000000000110100100010100001	00000000000011010010	0010100001	210	161
650000	00000000001001110101100010000	0000000000100111010	1100010000	634	784
2000001	0000000000100000000000000001	00000000001000000000	0000000001	512	1

Problem

Consider a logical address space of 64 pages of 1,024 words each, mapped onto a physical memory of 32 frames.

- a. How many bits are there in the logical address?
- b. How many bits are there in the physical address?

Problem

Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.

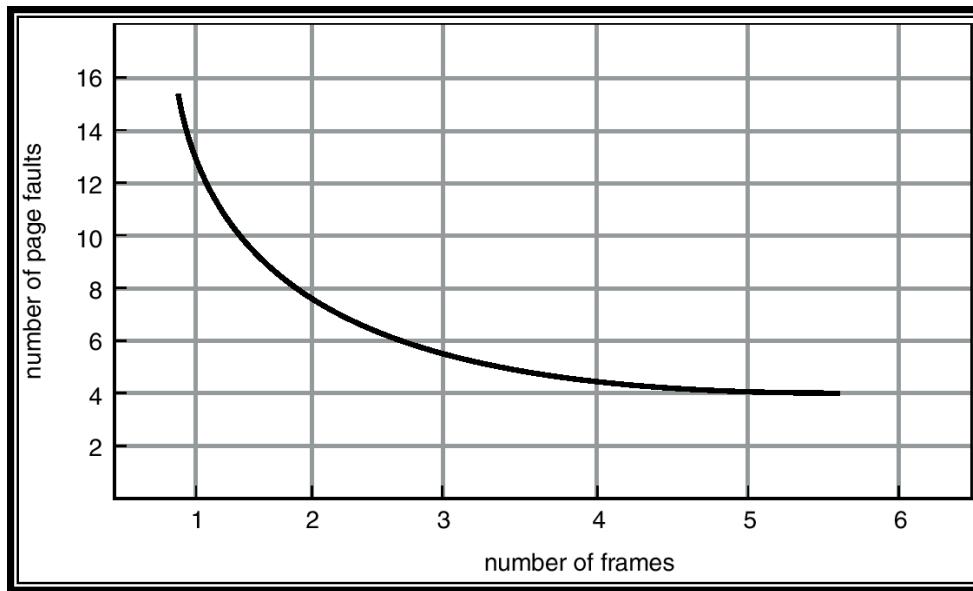
- a. How many bits are required in the logical address?
- b. How many bits are required in the physical address?

Problem

Consider a computer system with a 32-bit logical address and 4-KB page size. The system supports up to 512 MB of physical memory. Find out #pages and #frames.

Page Replacement Algorithms

- Want lowest page-fault rate.

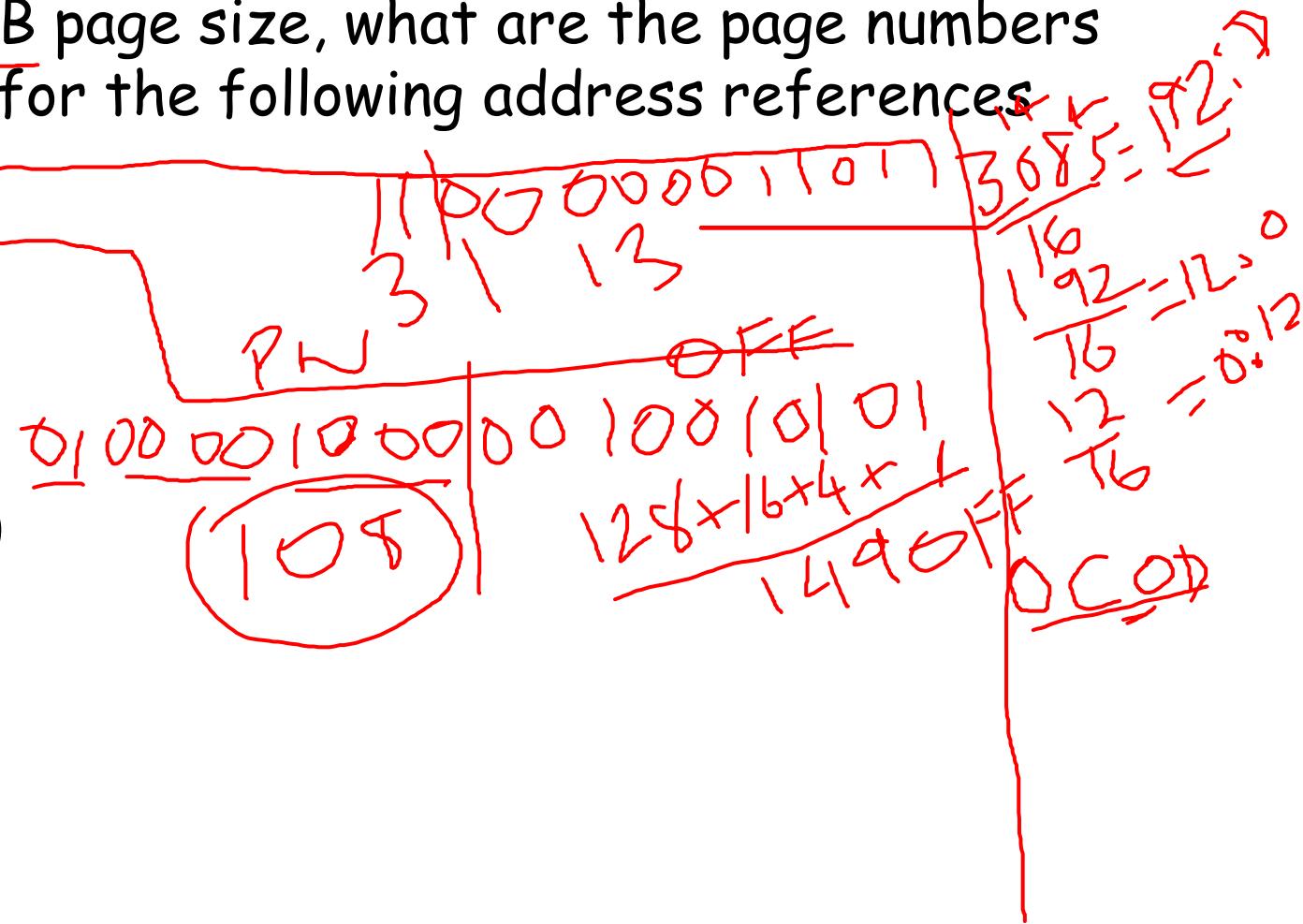


- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.

Problem

Assuming a 1-KB page size, what are the page numbers and offsets for the following address references

- a. 3085(d)
- b. 42095(H)
- c. 215201(O)
- d. 650000(H)
- e. 2000001 (H)





Virtual Memory

Introduction

- How to increase the degree of multiprogramming ?
- Virtual memory is a technique that allows the execution of processes that are not completely in memory.
- Motivation:
 - Programs often have code to handle unusual error conditions which is almost never executed.
 - Arrays, lists, and tables are often allocated more memory than they actually need.
 - Certain options and features of a program may be used rarely
 - Principle of locality
 - thrashing is a condition where system spends more time in swapping than executing instructions

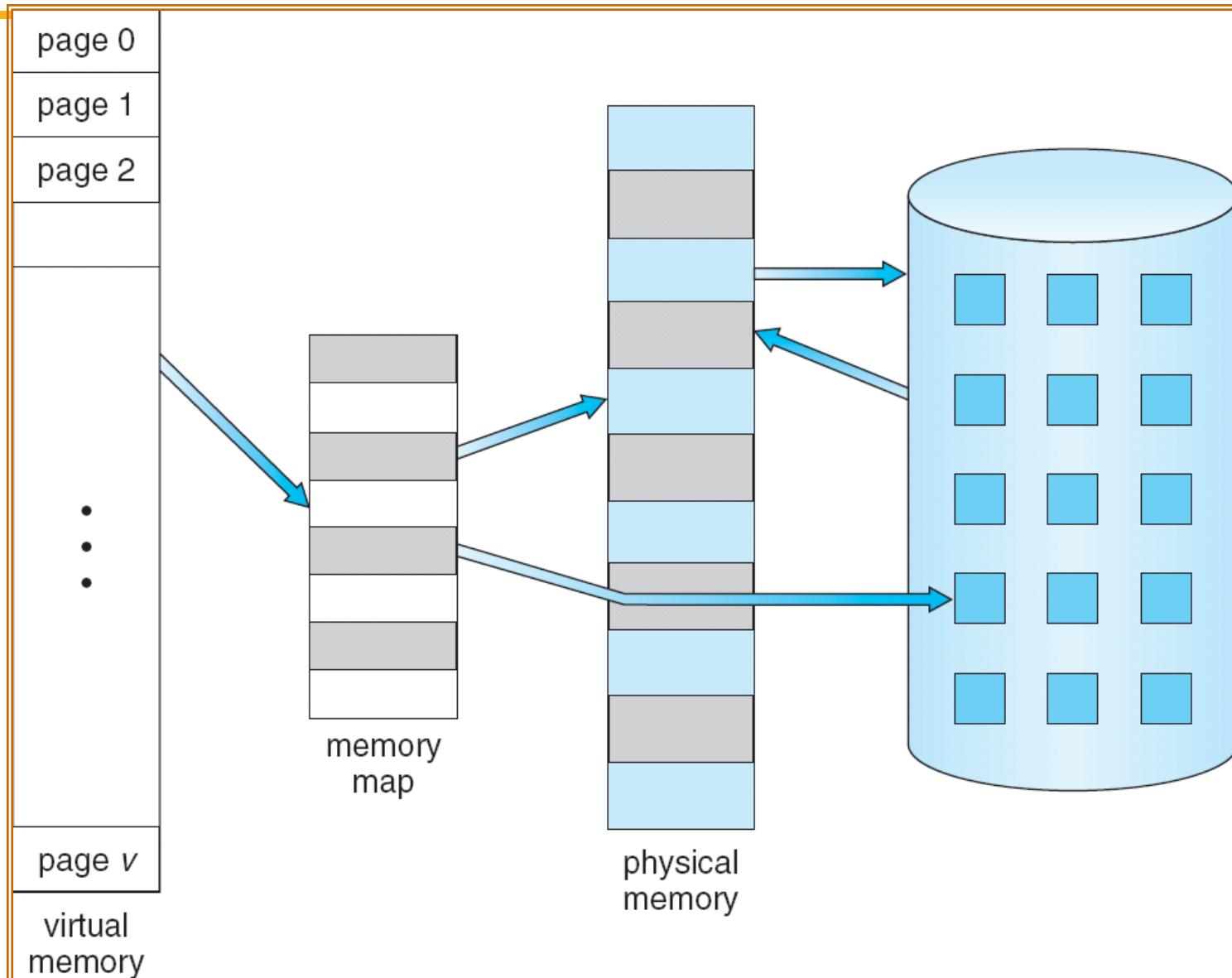
Contd...

- Advantages:
 - A program would no longer be constrained by the amount of physical memory that is available
 - Because each user program could take less physical memory, more programs could be run at the same time
 - increases the CPU utilization and throughput
 - Less I/O would be needed to load or swap each user program into memory, so each user program would run faster

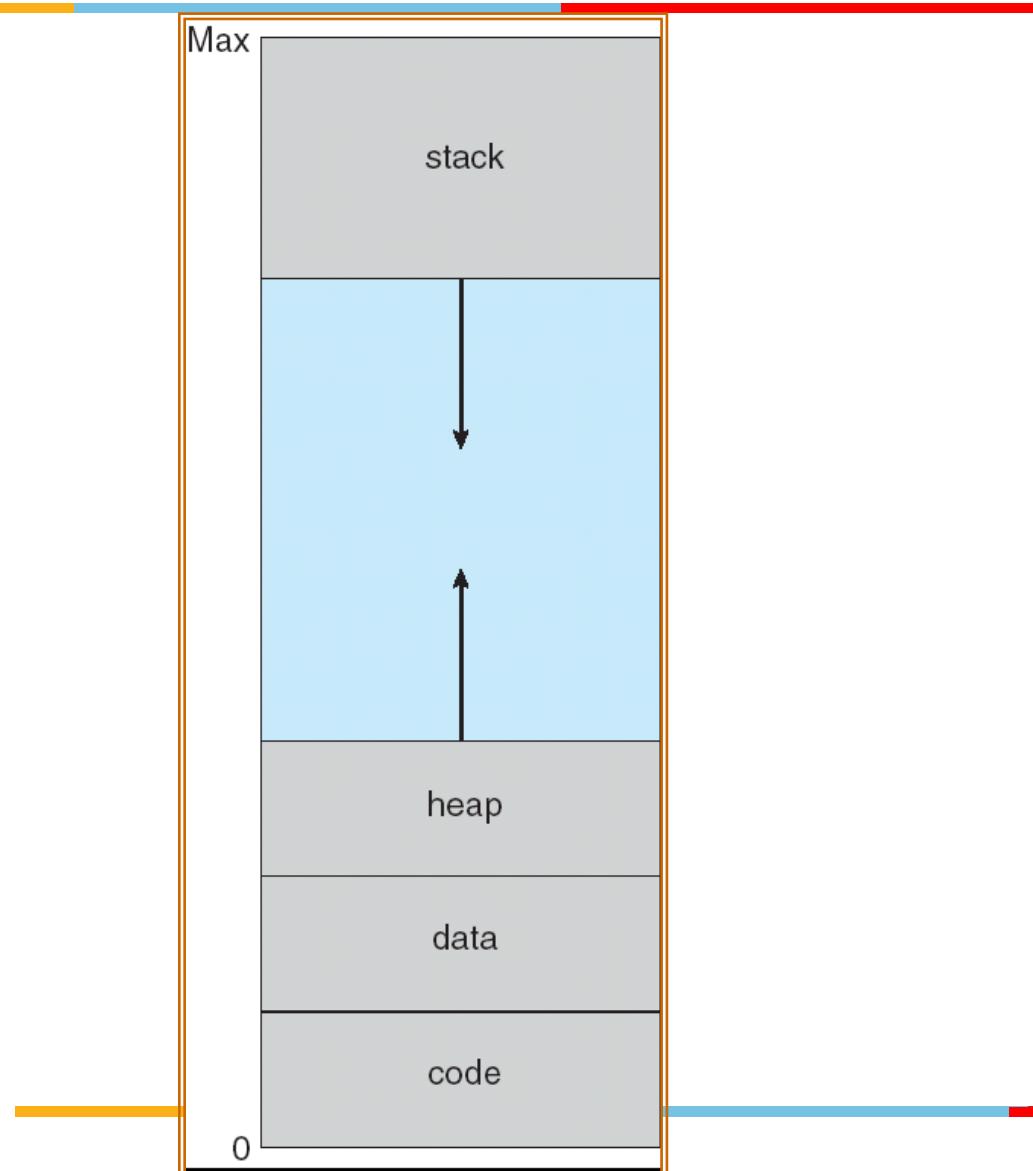
Background

- **Virtual memory** - separation of user logical memory from physical memory.
 - Only part of the program needs to be in memory for execution
 - Logical address space can therefore be much larger than physical address space
 - Allows address spaces to be shared by several processes
 - Allows for more efficient process creation

Virtual Memory That is Larger Than Physical Memory

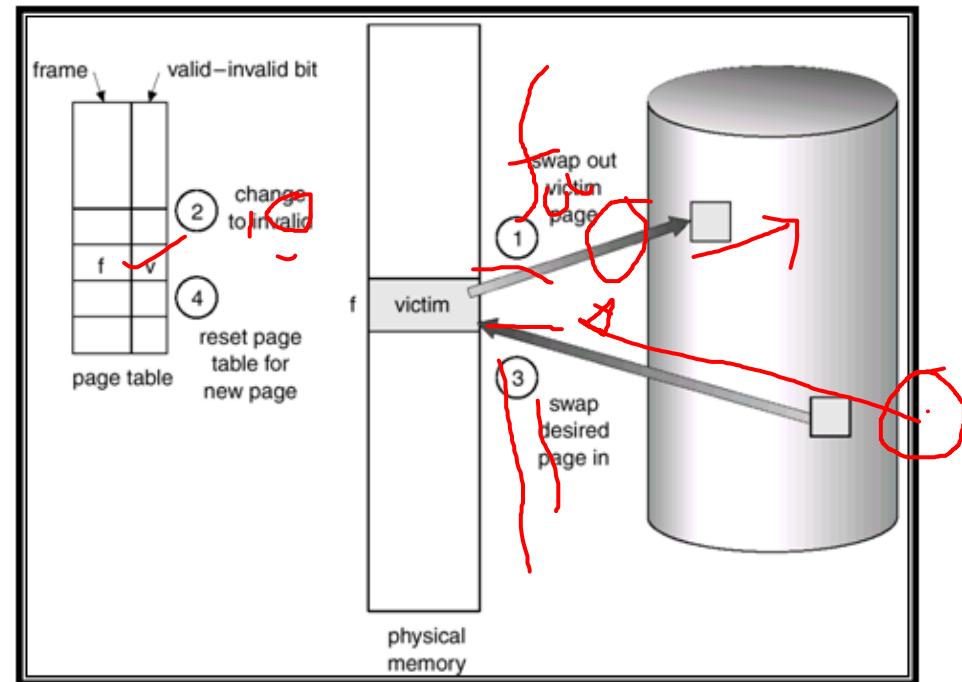


Virtual-address Space



What happens if there is no free frame?

- Page replacement - find some page in memory, but not really in use, swap it out
 - use a page replacement victim frame
 - performance - want a in minimum number of
- Same page may be brou



Contd...

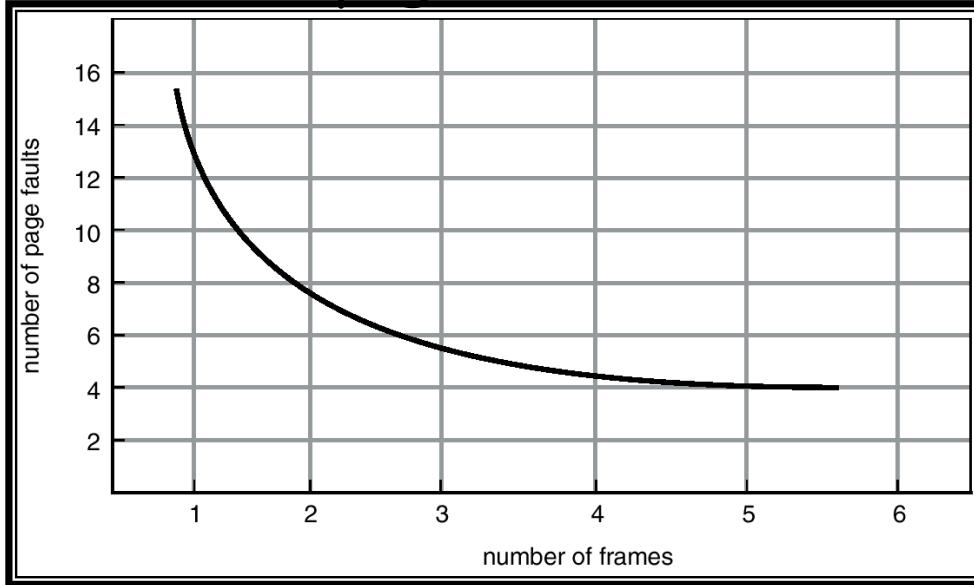
- if no frames are free, two page transfers (one out and one in) are required.
- Main drawback : doubles the page-fault service time and increases the effective access time accordingly.
- Use **modify (dirty)** bit to reduce overhead of page transfers - only modified pages are written to disk
- Demand paging requires two algorithms:
 - frame - allocation algorithm
 - page - replacement algorithm
- Frame allocation algorithm : handles frame allocation to each process.
- page - replacement algorithm : deals with the frames that are to be replaced

Procedure

1. Find the location of the desired page on the disk.
 2. Find a free frame:
 - a) If there is a free frame, use it.
 - b) If there is no free frame, use a page-replacement algorithm to select a victim frame.
 - c) Write the victim frame to the disk; change the page and frame tables accordingly.
 3. Read the desired page into the newly freed frame; change the page and frame tables.
 4. Restart the user process.
-

Page Replacement Algorithms

- Want lowest page-fault rate.



- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- In all our examples, the reference string is 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

First-In-First-Out (FIFO) Algorithm

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

3 frames (3 pages can be in memory at a time per process)



1	1	4	5
2	2	1	3
3	3	2	4

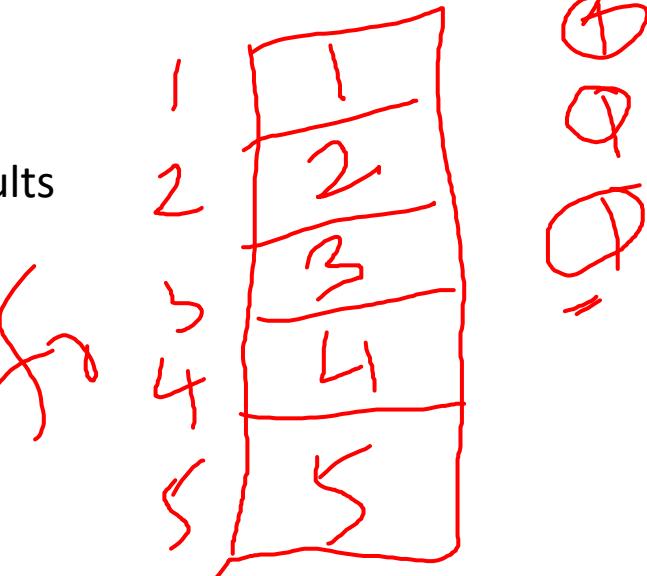
9 page faults

4 frames



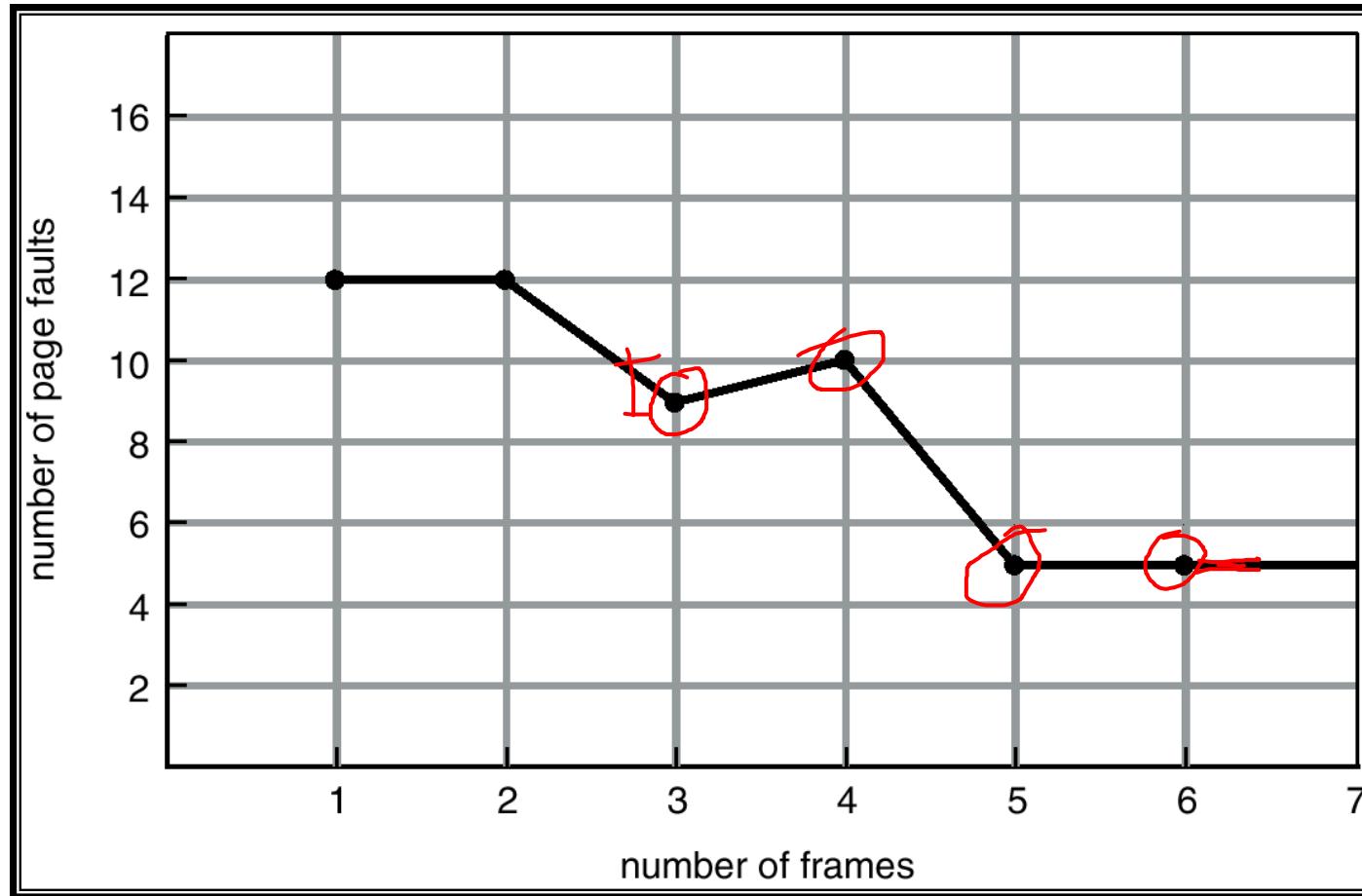
1	1	5	4
2	2	1	5
3	3	2	
4	4	3	

10 page faults



FIFO Replacement - Belady's Anomaly
– more frames \Rightarrow more page faults

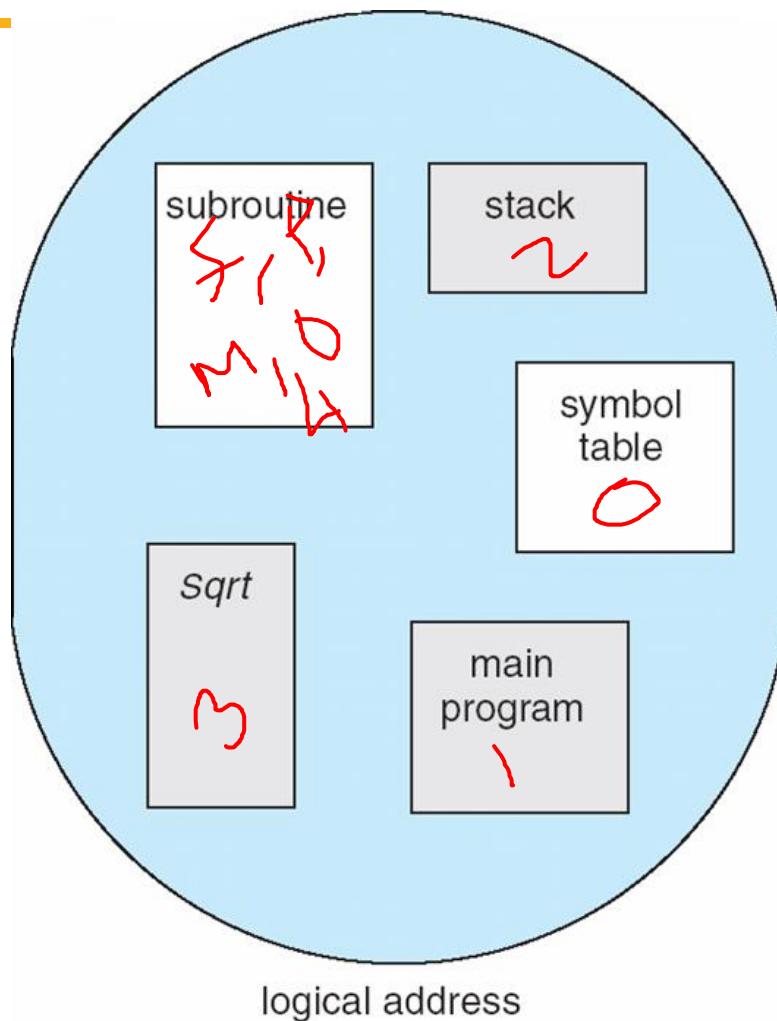
FIFO Illustrating Belady's Anomaly



Segmentation

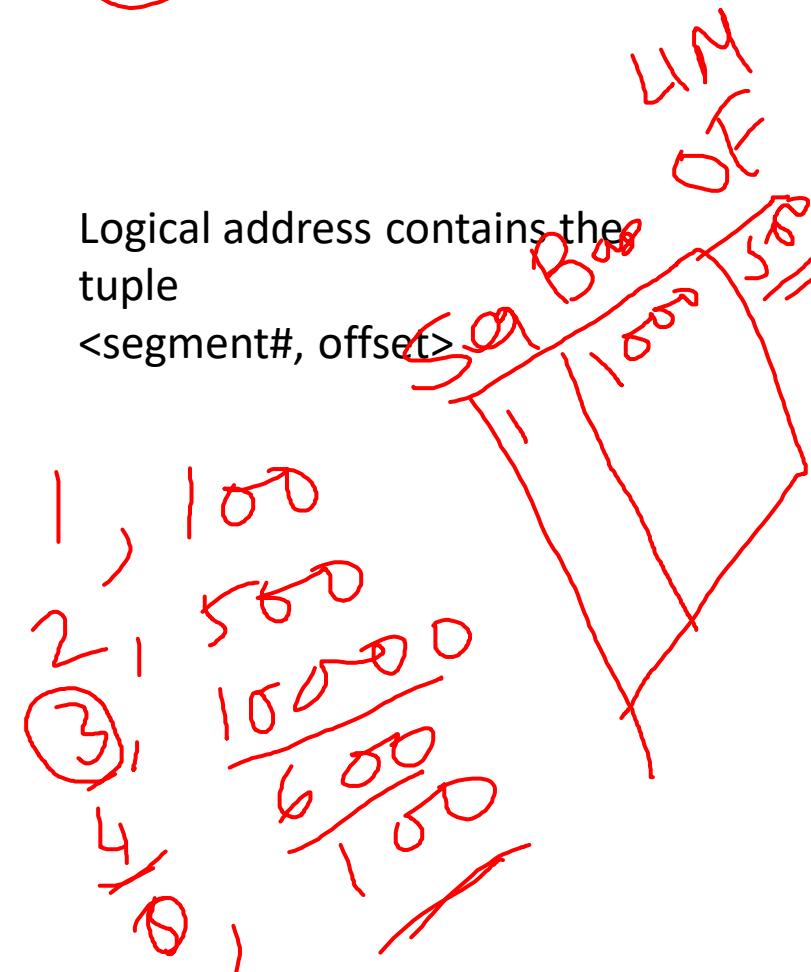
- Memory-management scheme that supports **user view** of memory
- A program is a collection of modules / segments
 - A segment is a logical unit such as:
 - main program, procedure / function / method,
 - object, local variables, global variables
 - common block, stack
 - symbol table, arrays
- Two Types : Simple Segmentation and Virtual memory segmentation

User's View of a Program



① = 1000 to 1500

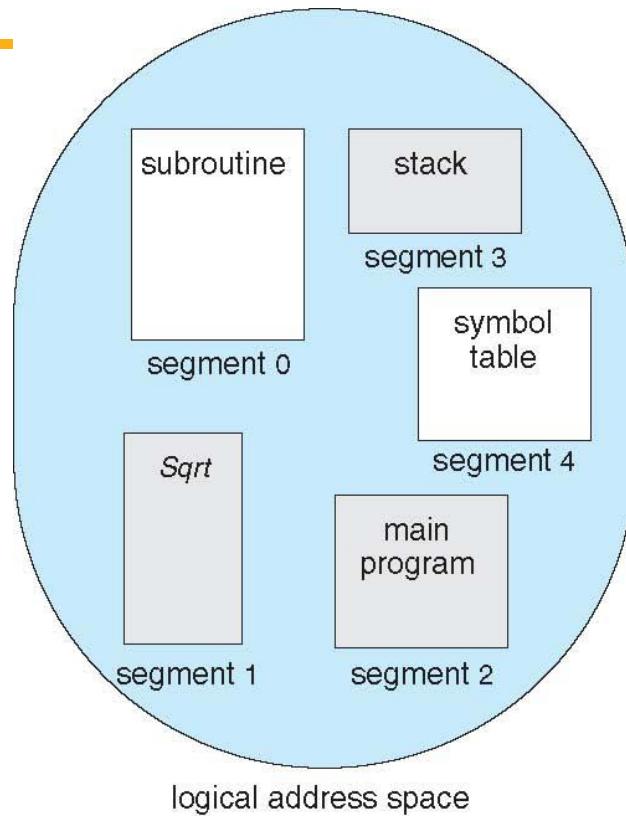
Logical address contains the tuple
<segment#, offset>



Segmentation Architecture

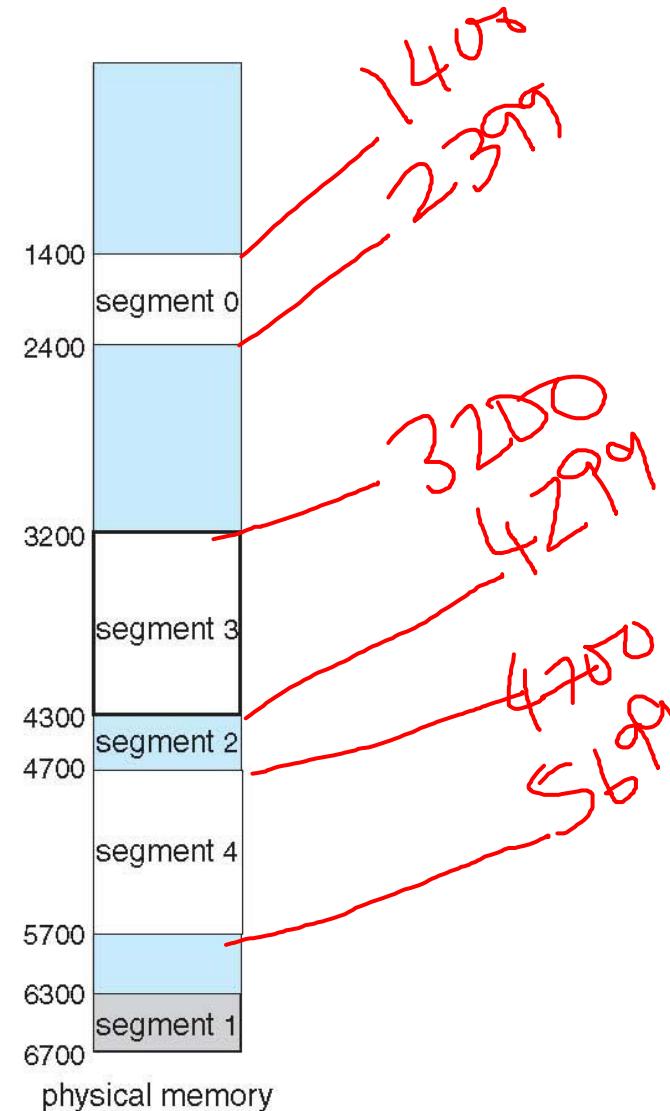
- **Segment table** - maps two-dimensional logical address to physical address;
- Each table entry has:
 - **base** - contains the starting physical address where the segments reside in memory
 - **limit** - specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;
segment number **s** is legal if **$s < STLR$**

Example of Segmentation

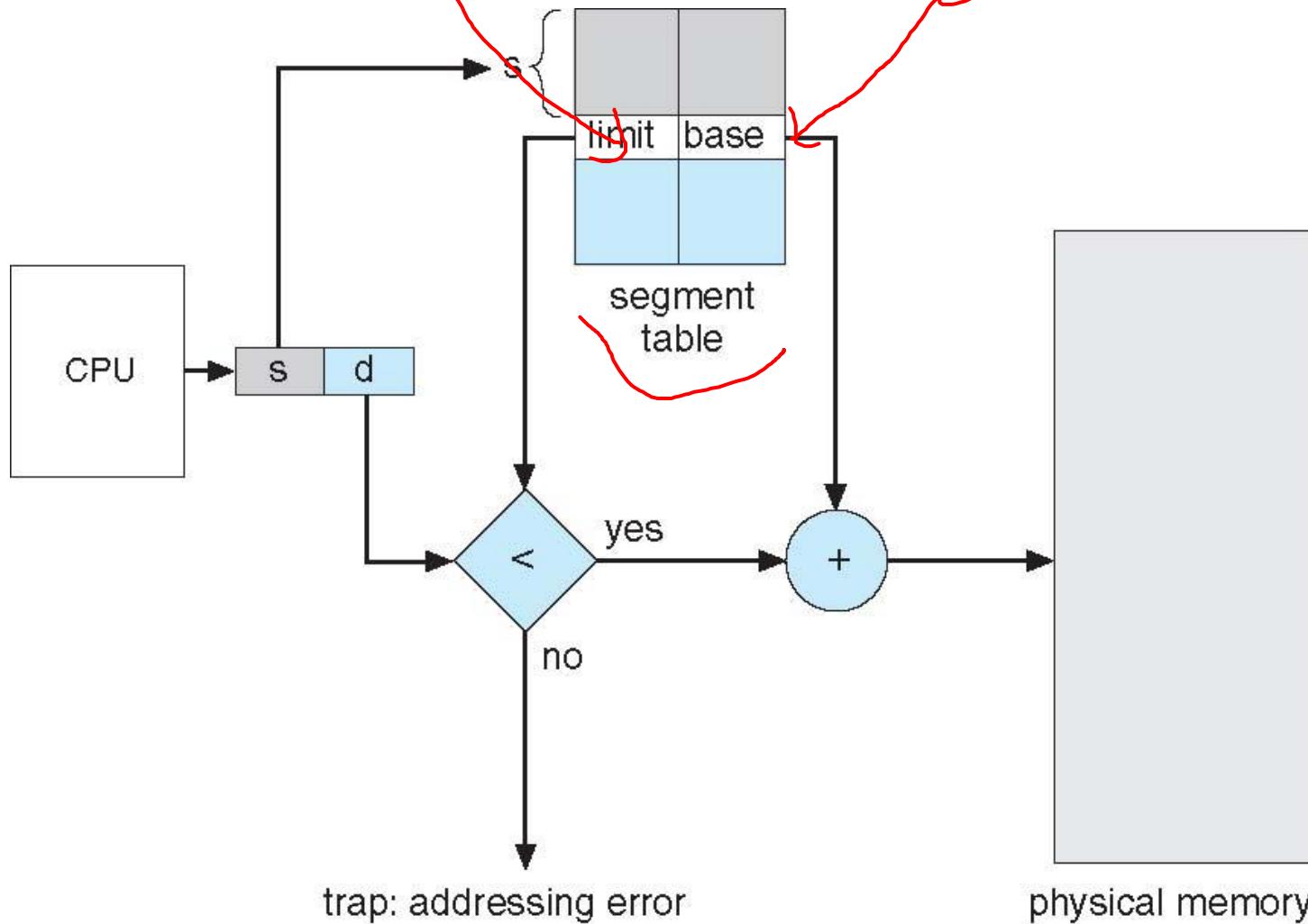


segment table

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



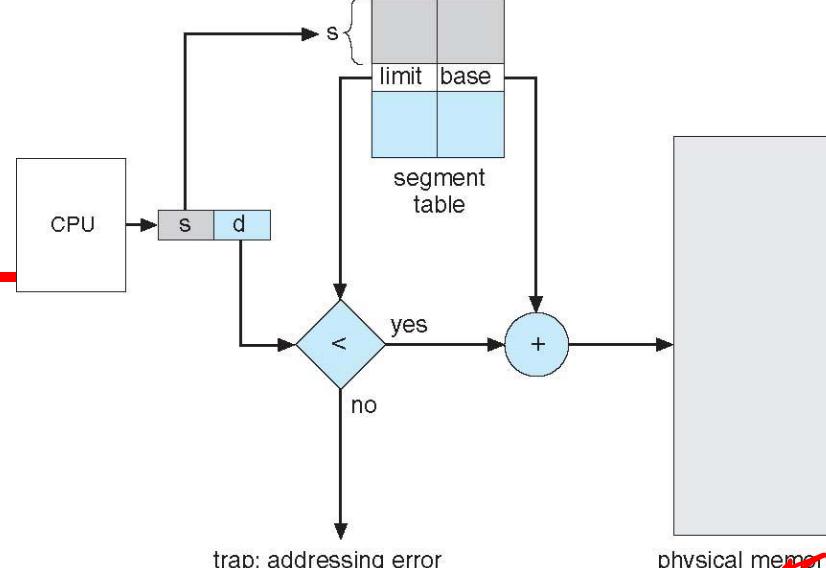
Segmentation Hardware



Problem: Segmentation

Consider the following segment table:

Segment	Base	Length(Limit)
0	128	512
<u>1</u>	8192	2048
2	1024	4096
3	16384	8192 → 245765
4	32768	1024
5	65536	16384



$$\begin{array}{r}
 \textcircled{1} - 128 + 430 = 558 \\
 2 - 16384 + 5024 = 1408
 \end{array}$$

What are the physical addresses for the following logical addresses (s, d)?

- a) 0, 430
- b) ~~1, 2056~~
- c) ~~2, 5024~~
- d) 3, 7024

Example 2

0100 Addr
0100 PageNum

STRE

Consider the following sequence of address

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611,
0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101,
0609, 0102, 0105

Assume 100 byte page.

Find out the reference string.

(1, 4, 1, 6) | 1, 1, 1, 1, 1, 1

String of
page number

Replacement Algorithm

~~a = 15~~

Given page reference string: 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3

Compare the number of page faults for ~~FIFO~~ Optimal and LRU page replacement algorithm

Solution : FIFO

~~Page Faults~~ ~~10~~ == 5

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3
1, 1, 1, 1,	1, 1, 5, 5, 5, 5, 5,	3, 3, 3, 3,												
2, 2, 2, 2,	2, 2, 6, 6, 6, 6, 6,	7, 7, 7, 7,												
3, 3, 3, 3,	3, 3, 3, 3, 2, 2, 2,	2, 2, 2, 2,												
	4, 4, 4, 4, 4, 4, 1, 1, 1, 1, 1, 1, 1,	6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,												
F F F F														

Problem3: Replacement Algorithm Will be done in webinar on 21-03-2023

Given page reference string: 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3

Compare the number of page faults for FIFO, Optimal and LRU page replacement algorithm

Solution : Optimal \rightarrow Replace page that will not be used for longest period of time

Problem3: Replacement Algorithm



Will be done in webinar on 21-03-2023

Given page reference string: 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2

Compare the number of page faults for FIFO, Optimal and LRU page replacement algorithm

Solution : LRU

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3



BITS Pilani

Pilani Campus

Optimizing Program Performance

Introduction

- What are the key characteristics of good programming?
- Five elements of a program
 - Variables
 - Loops
 - Conditionals
 - Input/output
 - Functions / methods
- How to write an efficient program?
 - Select an appropriate set of algorithms and data structures
 - Divide the given task and execute the subtasks parallel.
 - Use optimized compiler

Optimizing Compilers

- What is optimizing compiler?
- Steps in optimizing a program
 - Eliminate unnecessary work
 - Instruction level parallelism

Capabilities and Limitations of Optimizing Compilers

Memory Reuse

```
void twiddle1(int *xp, int *yp)
{
    *xp += *yp;
    *xp += *yp;
}

void twiddle2(int *xp, int *yp)
{
    *xp += 2* *yp;
}
```

$\cancel{\text{XP}} = \cancel{\text{XP}} + \cancel{\text{YP}}$
 $\cancel{\text{XP}} = \cancel{\text{XP}} + \cancel{\text{YP}}$
 $\cancel{\text{XP}} = \cancel{\text{XP}} + 2\cancel{\text{YP}}$

Memory Reuse

```
void twiddle1(int *xp )
{
    *xp += *xp;
    *xp += *xp;
}

void twiddle2(int *xp )
{
    *xp += 2* *xp;
}
```

$\cancel{\text{XP}} = \cancel{\text{XP}} + \cancel{\text{XP}}$
 $\cancel{\text{XP}} = \cancel{\text{XP}} + \cancel{\text{XP}}$
 $\cancel{\text{XP}} = \cancel{\text{XP}} + 2\cancel{\text{XP}}$

Code Optimization blocker - Memory Aliasing



```
x = 1000; y = 3000;  
*q = y; /* 3000 */  
*p = x; /* 1000 */  
t1 = *q; /* 1000 or 3000 */
```

```
x = 1000; y = 3000;  
*p = x; /* 1000 */  
t1 = *q; /* 1000 */
```

Example 1

Consider the following procedure to swap two values:

```
1  /* Swap value x at xp with value y at yp */
2  void swap(int *xp, int *yp)
3  {
4      *xp = *xp + *yp; /* x+y          */
5      *yp = *xp - *yp; /* x+y-y = x */
6      *xp = *xp - *yp; /* x+y-x = y */
7 }
```

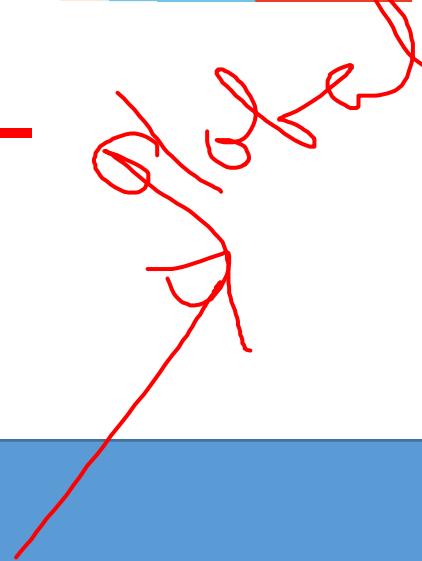
If this procedure is called with xp equal to yp, what effect will it have?

Code Optimization blocker - Function Calls



```
int f();  
  
int func1() {  
    return f() + f() + f() + f();  
}
```

```
int func2() {  
    return 4*f();  
}
```



```
int counter = 0;  
  
int f() {  
    return counter++;  
}
```

Optimization techniques

- Code Movement
- Dead Code Elimination
- Strength Reduction
- Common Expression Elimination
- Compile time evaluation
 - Constant Folding
 - Constant Propagation

Code Movement

Move the code fragment outside the loop as it won't have any difference if it is performed inside the loop repeatedly or outside the loop once.

Source Code:

```
for ( x = 0 ; x < n ; x++ ) {  
    temp = sum + 10;  
    a[x] = a[x] + x;  
}
```

Optimized Code:

```
temp = sum + 10;  
for ( x = 0 ; x < n ; x++ ) {  
    a[x] = a[x] + x;  
}
```

Contd...

```
void set_row(double *a, double *b,
    long i, long n)
{
    long j;
    for (j = 0; j < n; j++)
        a[n*i+j] = b[j];
}
```



```
long j;
int ni = n*i;
for (j = 0; j < n; j++)
    a[ni+j] = b[j];
```

Dead Code Elimination

Remove code fragment which does not affect the program results.

```
int fun(void) {  
    int x = 4;  
    int y = 5; /* Assignment to dead variable */  
    int z;  
    z = x + 4;  
    return z;  
    x = 4; /* Unreachable code */  
    return 0;  
}
```

```
int fun(void) {  
    int x = 4;  
    int z;  
    z = x + 4;  
    return z;  
}
```

Strength Reduction

Replace complex instructions with cheaper expressions.

Source Code :

```
for ( x = 0; x < n ; x++ ){
    y = z * 2;
}
```

Optimized Code :

```
for ( x = 0; x < n ; x++ ){
    y = z + z;
}
```

Common Expression Elimination



- Eliminate the expression which is appearing repeatedly in the code

Source code:

```
x1 = y * 2 + z;  
x2 = y * 2 - z;
```

```
Temp = y * 2;
```

```
x1 = Temp + z;  
x2 = Temp - z;
```

Compile Time Evaluation

- Constant Folding: Process of evaluating the expressions whose values are constant at compile time.

Example :

Source code :

```
x = y + 2 + z - 3;
```

Optimized code :

```
x = y + z - 1;
```

- Constant Propagation: Process of substituting the values of known constants in the expressions at compile time

Source Code :

```
int x = 10;  
int y = x + 10 + x/2 ;  
return y + x;
```

Optimized Code 1:

```
int x = 10;  
int y = 10 + 10 + 10 /2;  
return y + 10;
```

Optimized Code 2:

```
int x = 10;  
int y = 25;  
return 35;
```

Loop Unrolling

- Also known as loop unwinding
- Tries to transform the loop so that program execution improves at the cost of size

Source Code:

```
while ( x <= 100) {  
    a[x] = x+10;  
    x++;  
}
```

Optimized Code :

```
while ( x <= 100) {  
    a[x] = x+10;  
    x++;  
    a[x] = x+10;  
    x++;  
}
```

Example

```
int x;
for (x = 0; x < 100; x++)
{
    delete(x);
}
```

```
int x;
for (x = 0; x < 100; x += 5 )
{
    delete(x);
    delete(x + 1);
    delete(x + 2);
    delete(x + 3);
    delete(x + 4);
}
```

Experiment 1: Redundant Code

```
program Ex1
    n = 5
    for i = 1 to 6
        n = n + 1
        if n = 3 then
            n = 0
        end if
    next
end
```

- a. Compile the program and Observe the following:
 - i. Note down the code size in Binary Code
-> Show button-> Show instructional Stats
 - ii. Note the extra **mov** instructions in the assembly code generated

Experiment 1: Redundant Code

```
program Ex1
    n = 5
    for i = 1 to 6
        n = n + 1
        if n = 3 then
            n = 0
        end if
    next
end
```

- b.
- In the COMPILER frame click on the Enable "Optimizer" check box. Click on the "Redundant Code" check box in the Optimizer window. Compile the source again. Observe the following:
- i. Note down the code size in Binary Code
-> Show button -> Show instructional Stats
 - ii. Note the lean and mean set of instructions in the assembly code generated

Experiment 2: Constant Folding

```
program Ex4  
n = 1 + 7 - 9  
End
```

Repeat what was done in experiment 1 and notice the changes.
a. Next check the "Constant Folding" check box.

Experiment : 3

Strength Reduction

program Ex5

i = 3

n = i * 16

end

Repeat what was done in experiment 1 and notice the changes.

a. Next check the Strength Reduction check box

Experiment 4: Loop Unrolling

```
program Ex6
    for p = 1 to 100
        r = r + 2
    next
end
```

Repeat what was done in experiment 1 and notice the changes.

- a. Next check the Loop Unrolling check box
Compile time is more, Code size is large compared to un-optimized code
- b. Execute the program with and without optimiser and see the execution time difference



BITS Pilani
Pilani Campus

COMPUTER ORGANIZATION & SOFTWARE SYSTEMS

SESSION 16

Prepared: Dr. Lucy J. Gudino

Instructor: Prof. C R Sarma

WILP & Department of CS & IS



Multiprocessor Organizations

BITS Pilani
Pilani Campus

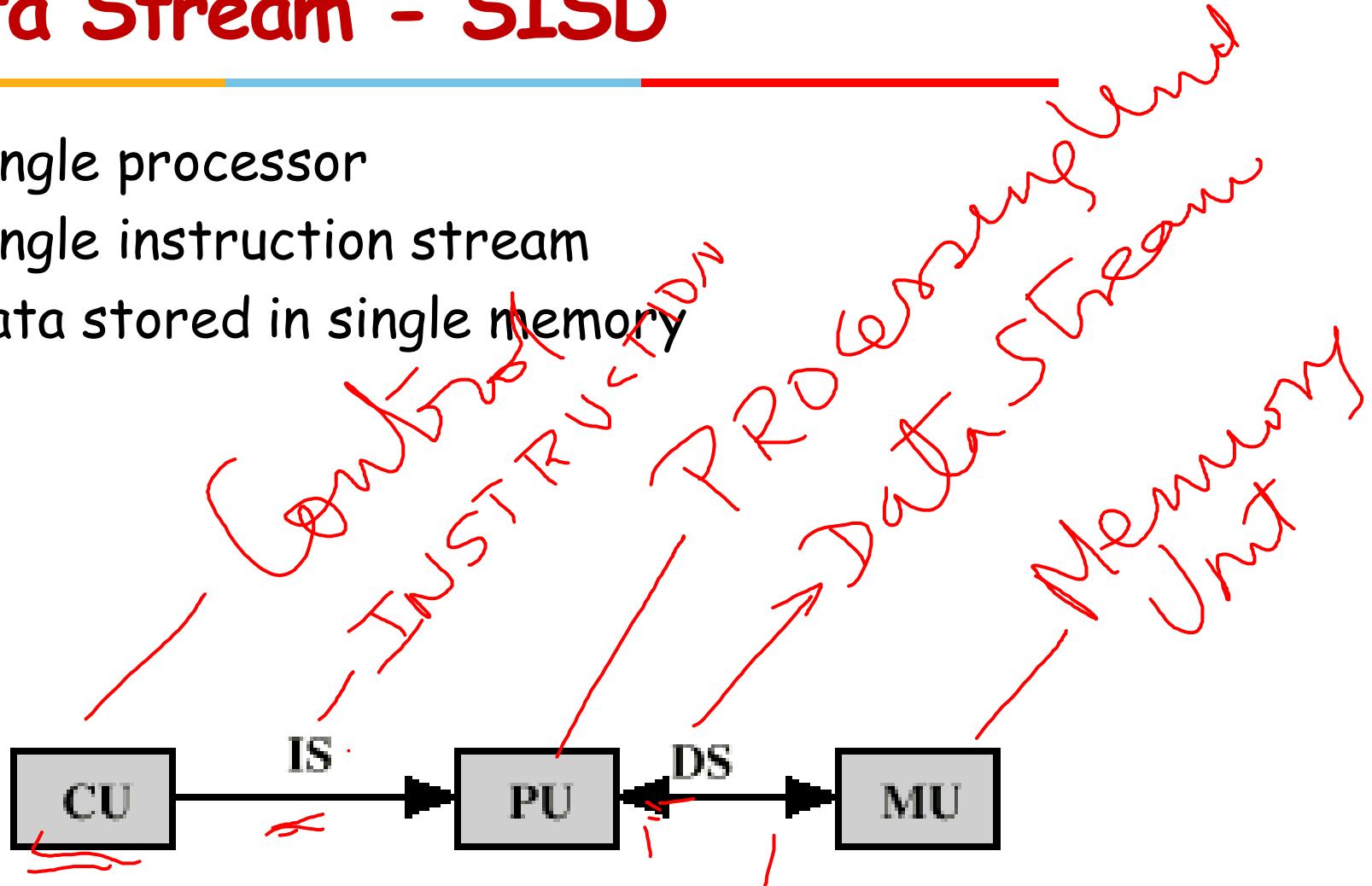
Multiprocessor Organization

Multicore

- Multiprocessor Vs Multicomputer system
- Flynn's Classification TAXONOMY
 - Single instruction, single data stream - SISD
 - Single instruction, multiple data stream - SIMD
 - Multiple instruction, single data stream - MISD
 - Multiple instruction, multiple data stream- MIMD

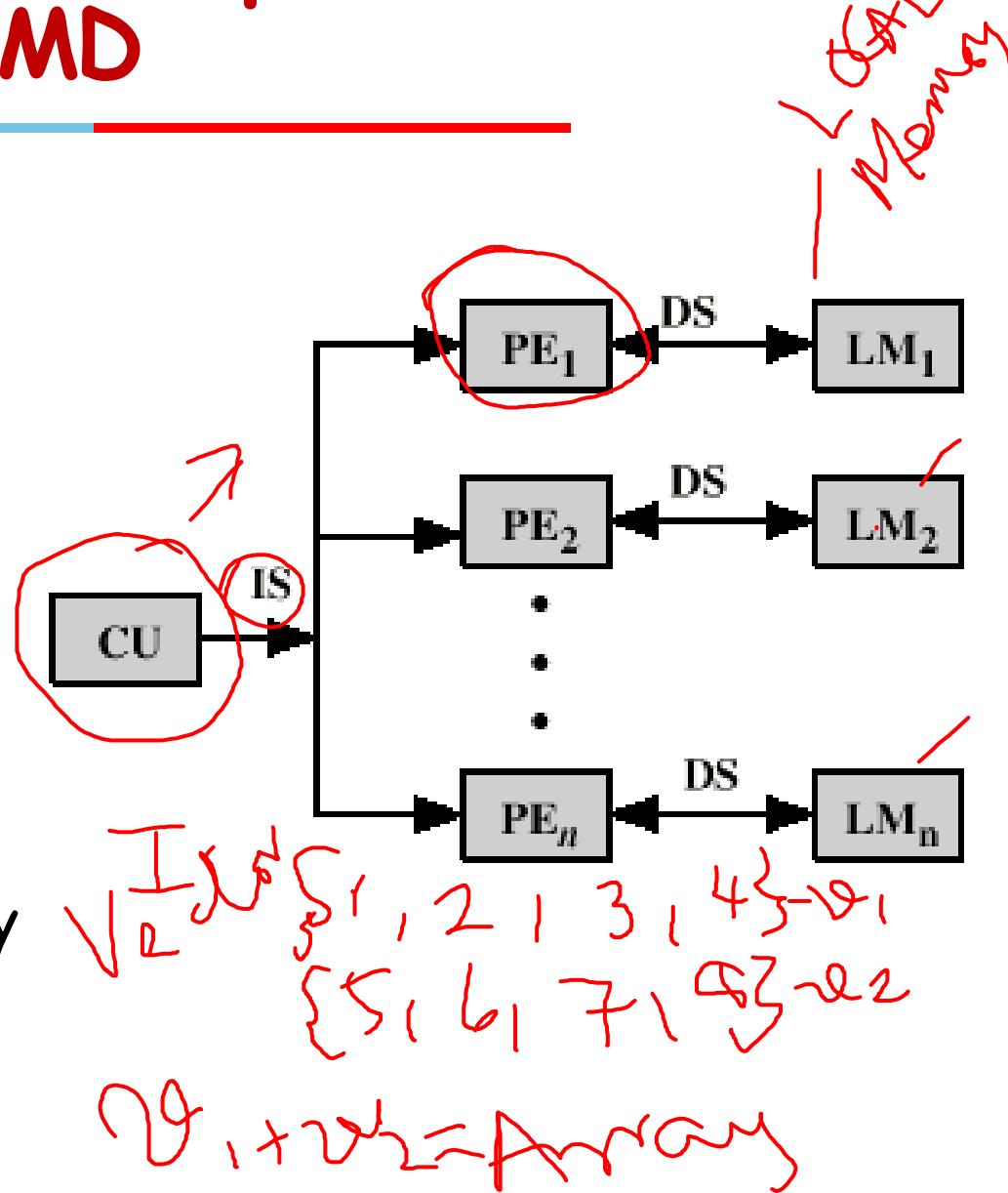
Single Instruction, Single Data Stream - SISD

- Single processor
- Single instruction stream
- Data stored in single memory



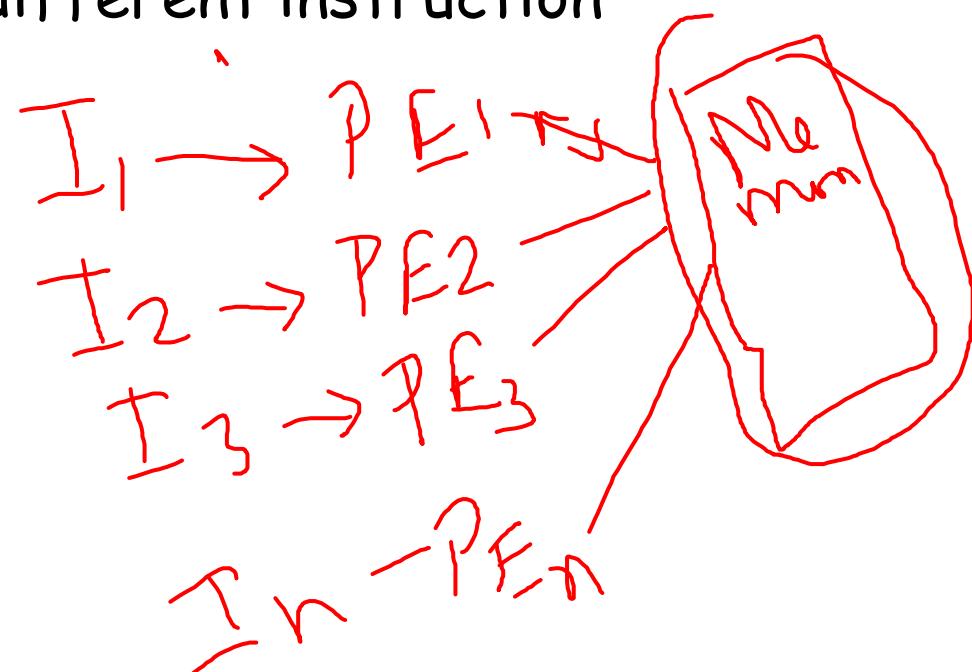
Single Instruction, Multiple Data Stream - SIMD

- Single machine instruction
- Controls simultaneous execution
- Number of processing elements
- Each processing element has associated data memory
- Each instruction executes on different set of data by different processors
- Vector and array processors



Multiple Instruction, Single Data Stream - MISD

- Single Sequence of data for multiple instruction stream
- Transmitted to set of processors
- Each processor executes different instruction sequence
- Never been implemented

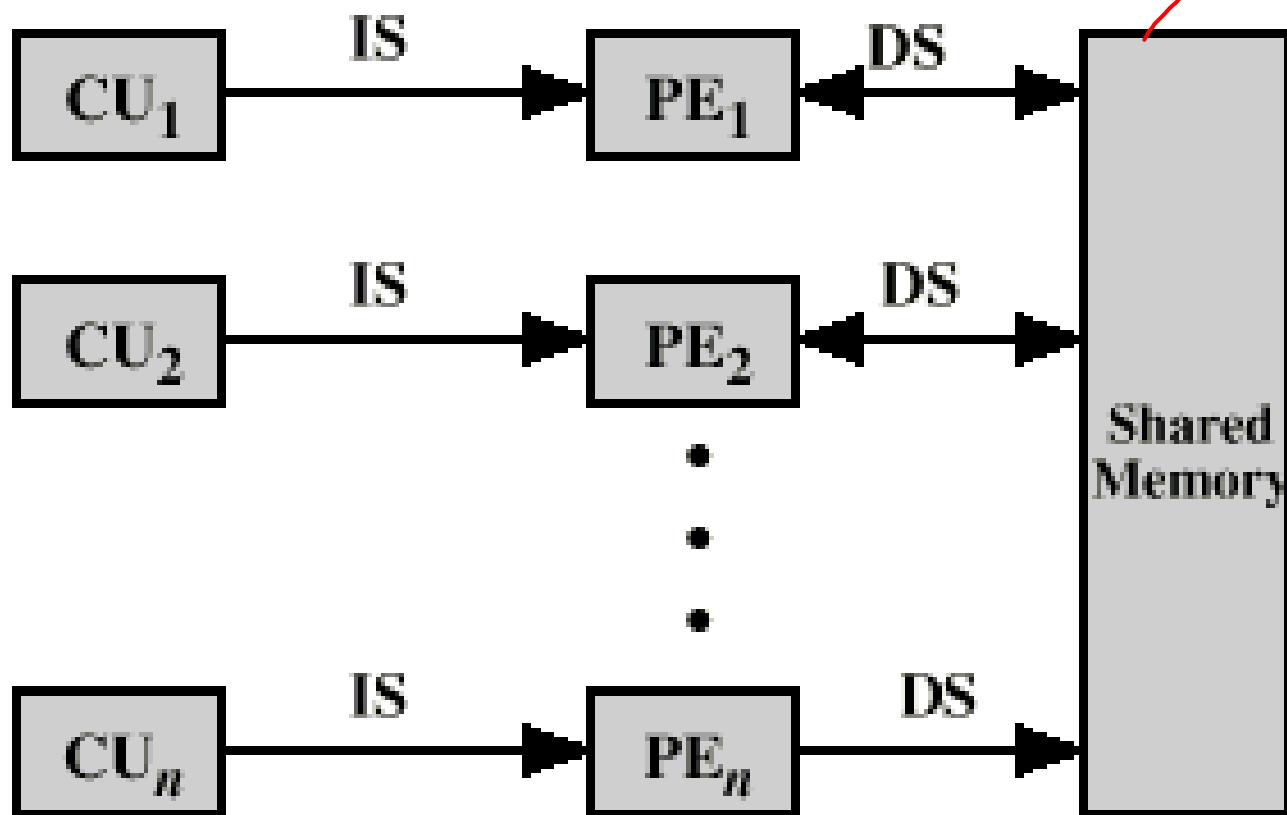


Multiple Instruction, Multiple Data Stream- MIMD



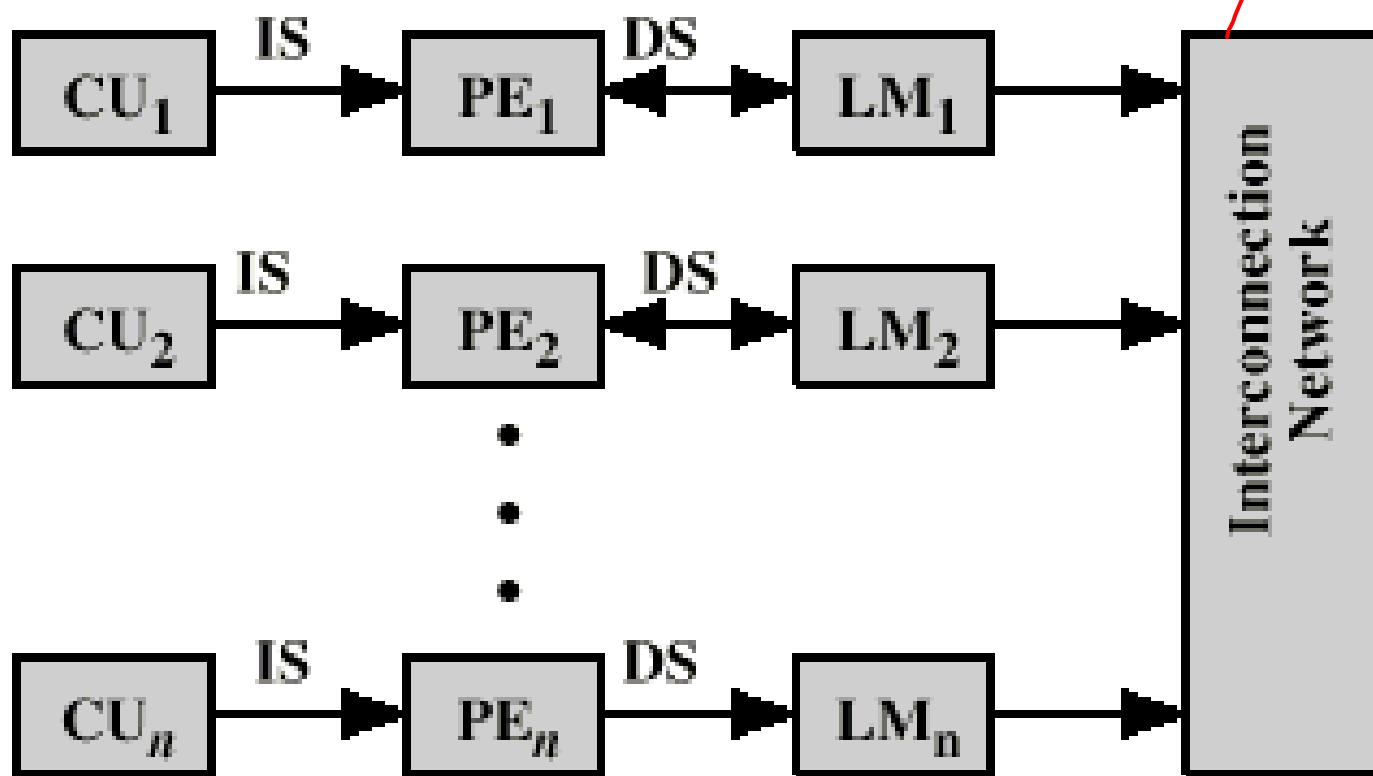
- Set of processors
- Simultaneously execute different instruction sequences
- Different sets of data
- Examples
 - SMPs → Symmetric Multiprocessor
 - Clusters
 - NUMA → Non-Uniform Memory Systems
- Two Types :
 - Tightly Coupled → Shared Memory Systems
 - Loosely Coupled Systems → Distributed Memory Systems

Parallel Organizations - MIMD Shared Memory

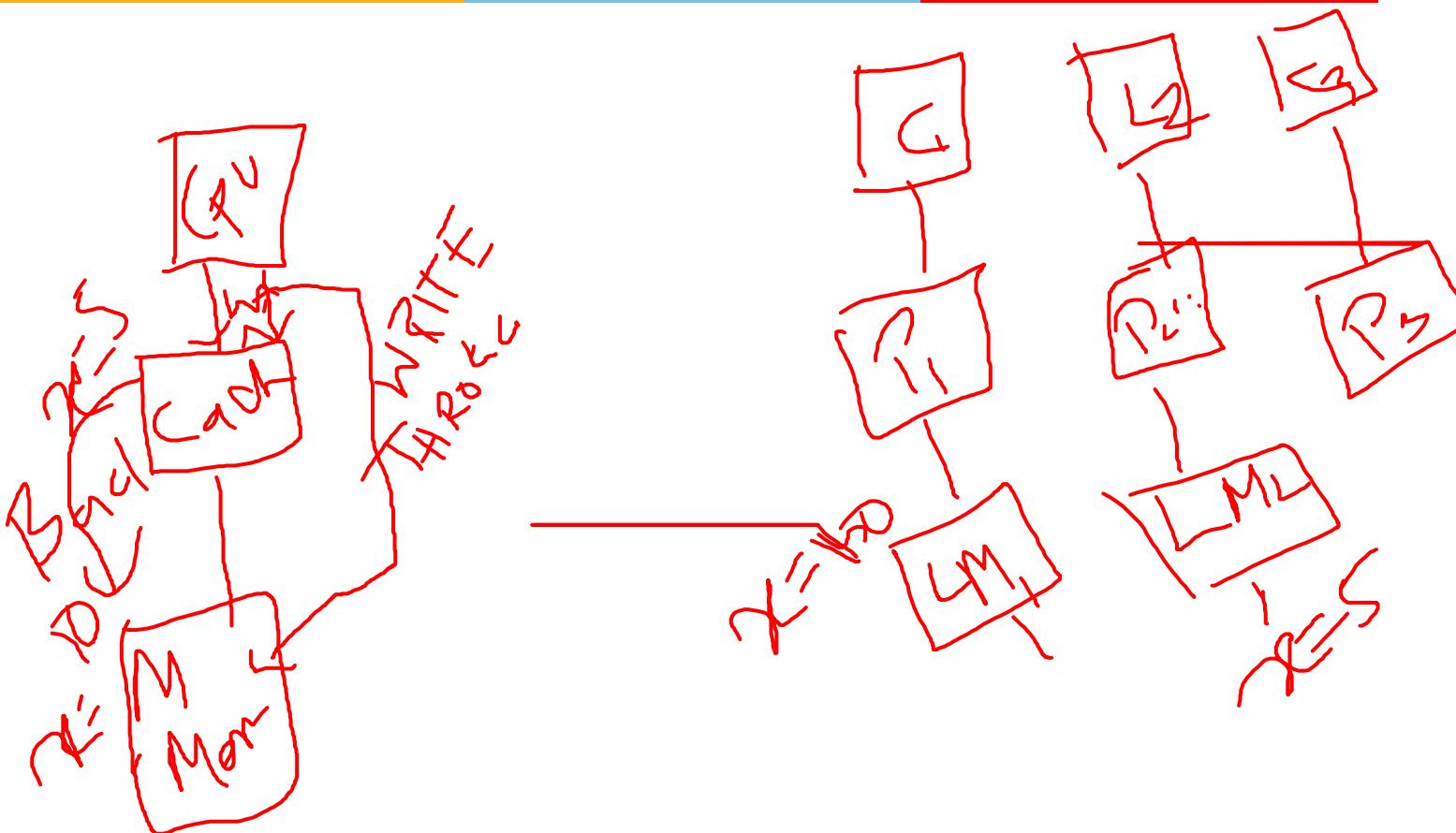


Parallel Organizations - MIMD

Distributed Memory



Write through and Write Back Policy



Symmetric Multiprocessors

~~SMP~~

A stand alone computer with the following characteristics

- Two or more similar processors of comparable capacity
- Processors share same memory and I/O
- Processors are connected by a bus or other internal connection
- Memory access time is approximately the same for each processor
- All processors share access to I/O
 - Either through same channels or different channels giving paths to same devices
- All processors can perform the same functions (hence symmetric)
- System controlled by integrated operating system
 - providing interaction between processors
 - Interaction at job, task, file and data element levels

SMP Advantages

Performance

- If some work can be done in parallel

Availability

- Since all processors can perform the same functions, failure of a single processor does not halt the system

Incremental growth

- User can enhance performance by adding additional processors

Scaling

- Vendors can offer range of products based on number of processors

Cluster



- Alternative to SMP
- A group of interconnected whole computers
- Working together as unified resource
- Illusion of being one machine
- Each computer called a node
- High performance
- High availability
- Server applications

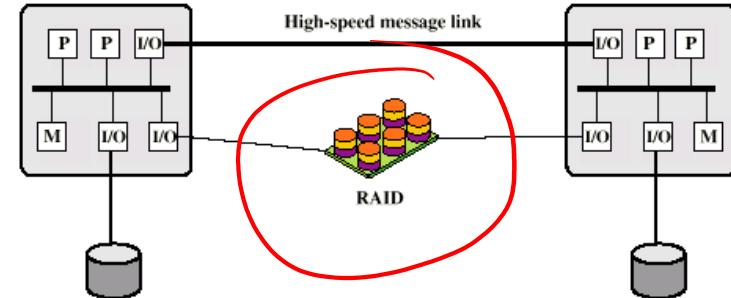
TPC-C
DISTRIBUTED
X11

Cluster Benefits

- Absolute scalability
- Incremental scalability
- High availability
- Superior price/performance



Standby Server, No Shared Disk



Shared Disk

Cloud Computing

- Provides Computing, Networking and storage on demand
- The main advantage of cloud computing is that it provides wide range of technologies so that one can innovate faster and build anything that one can imagine.
- Definition:
—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

- 3 cloud service models or service types for any cloud platform
- 4 deployment models
- 5 essential characteristics of cloud computing infrastructure

NIST
NF

Characteristics of Cloud Computing

5 Essential Characteristics of Cloud Computing

Ref: The NIST Definition of Cloud Computing

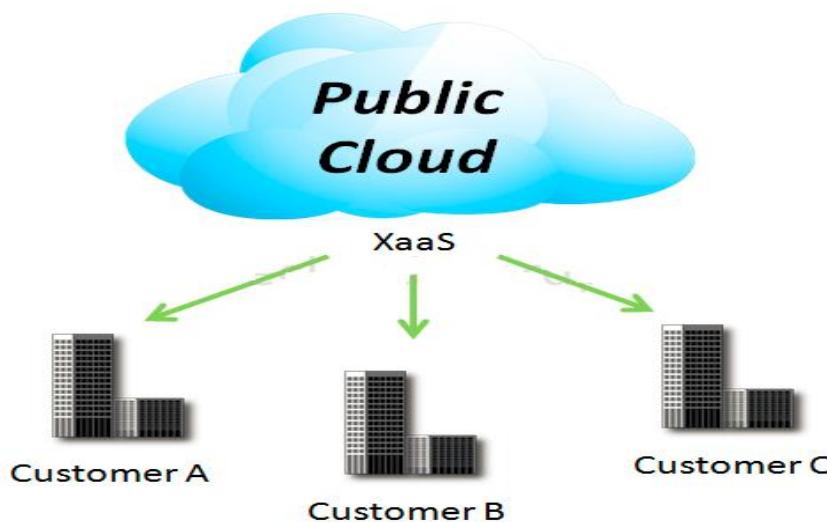
<http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>



Source: <http://aka.ms/532>

4 Deployment models

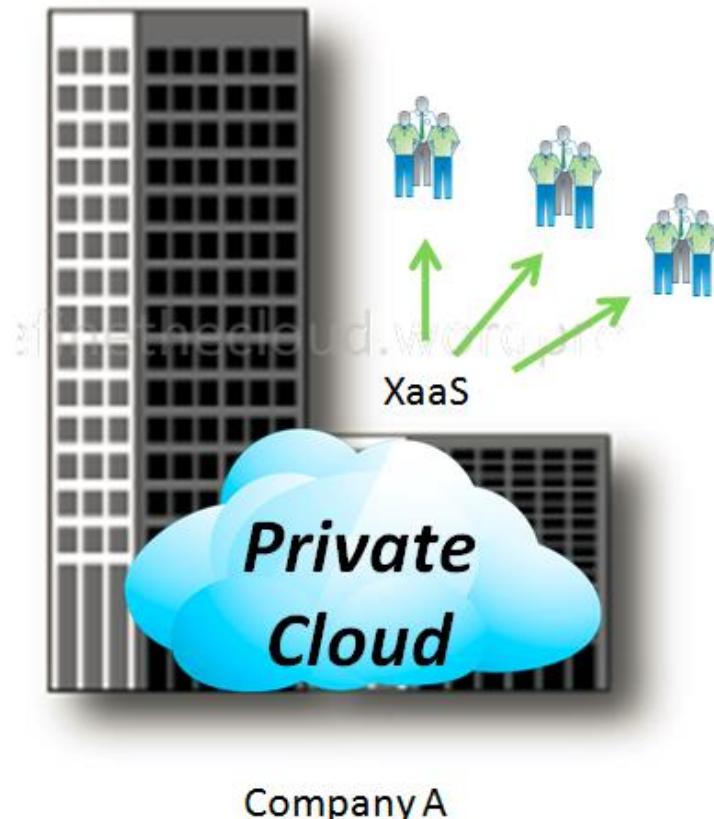
1. Public Cloud



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

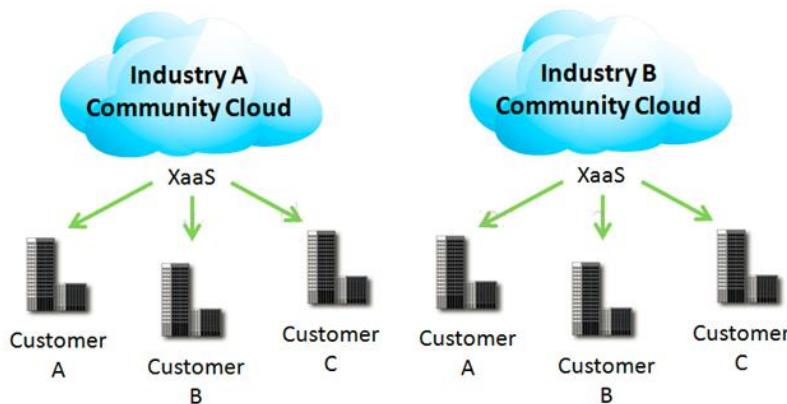
2. Private Cloud



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

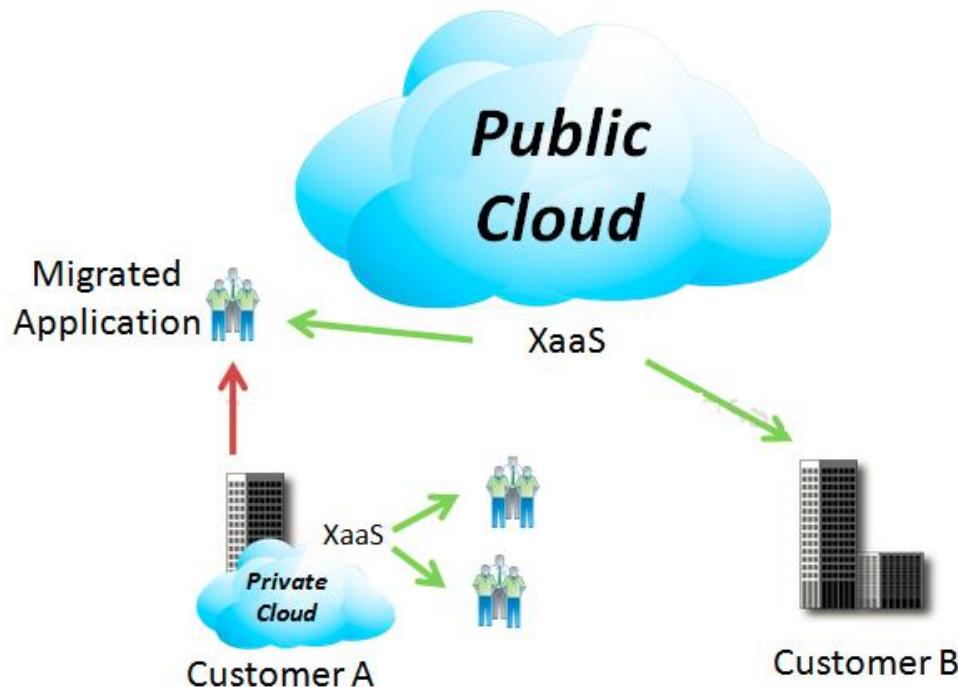
3. Community Cloud



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.

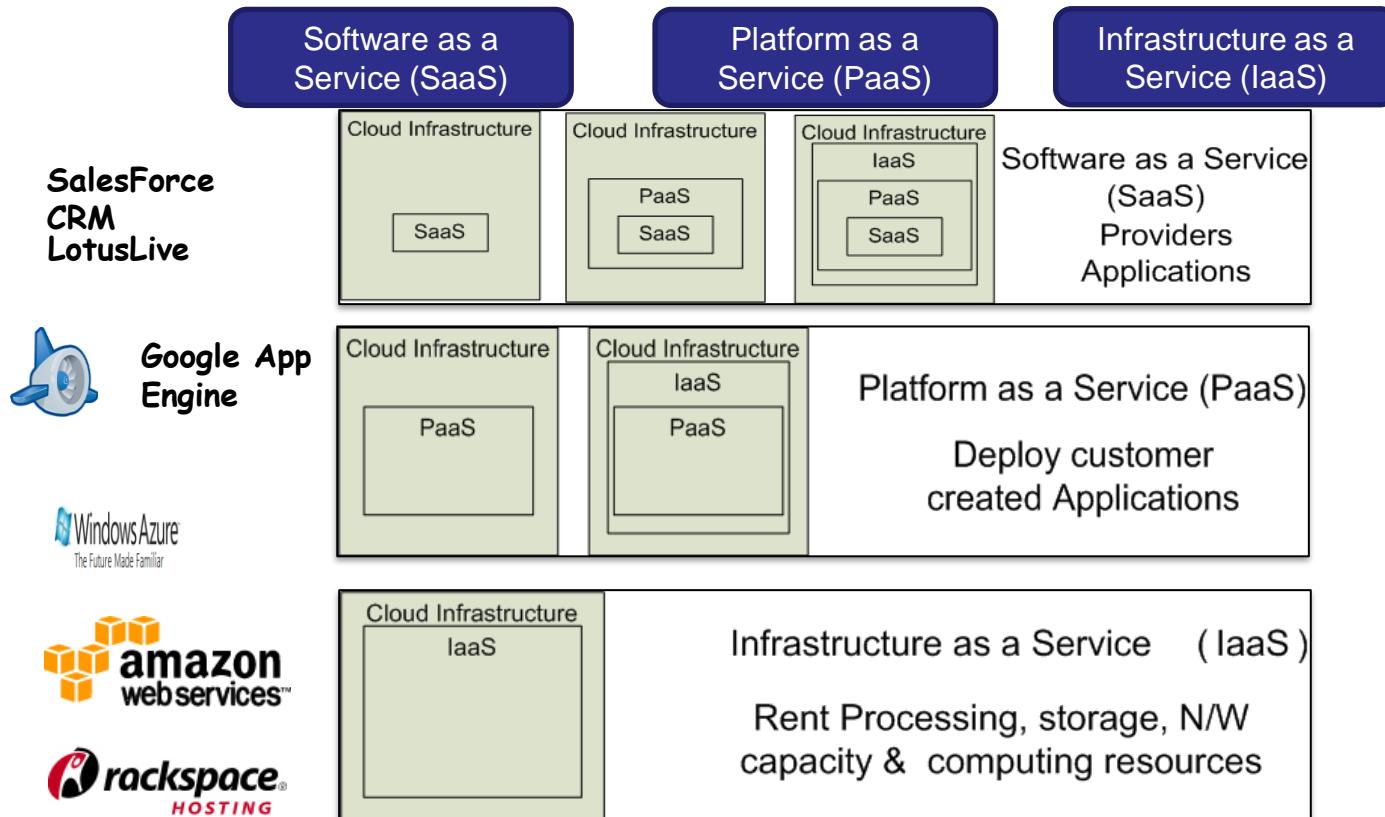
4 Deployment models

4. 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

3 Cloud Service Models



? Q&A

That's all Folks!!!

The Contact Sessions are OVER!

Prepare well for the Comprehensive exam

Study the text books

Refer the Reference books

Presentation is for Guidance purpose use judiciously

All the Best for Preparation and Best of LUCK for EXAM!



BITS Pilani
Pilani Campus



Webinar-1

Performance Assessment Error Correction CPU-OS Simulator



BITS Pilani
Pilani Campus

Performance Assessment

(R1: 1.4, T1: 2.3)

- MIPS Rate, Amdahl's Law



Performance Assessment-Summary

- If you were running a program on two different processors, we would say that the faster is the one that gets the job done first.
- **Execution time:** The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution

$$\text{Performance}_x = \frac{1}{\text{Execution time}_x}$$

This means that for two computers X and Y, if the performance of X is greater than the performance of Y, we have

$$\text{Performance}_x > \text{Performance}_y$$

$$\frac{1}{\text{Execution time}_x} > \frac{1}{\text{Execution time}_y}$$

$$\text{Execution time}_y > \text{Execution time}_x$$

Performance Assessment-Summary

- All computers are governed by a **clock** that determines when events take place in the hardware.
- These discrete time intervals are called **clock cycles**.
- The rate of clock pulses is known as the **clock rate**, or clock speed (Hertz) which is the inverse of the **clock period**. For example, 1 GHz processor receives 1 billion pulse / sec

CPU Performance and Its Factor

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock cycle time}}$$

Alternatively, because clock rate and clock cycle time are inverses,

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

Performance Assessment-Summary

Instruction Performance

$\text{CPU clock cycles} = \text{Instructions for a program} \times \frac{\text{Average clock cycles per instruction}}$

- The term **clock cycles per instruction**, which is the average number of cycles each instruction takes to execute, is often abbreviated as **CPI**.

$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$

or, since the clock rate is the inverse of clock cycle time:

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

Performance Assessment -

Millions of Instructions per Second (MIPS) Rate

Million Instructions Per Second (MIPS) is the common measure of performance for a processor is the rate at which instructions are executed, expressed as MIPS or referred to as MIPS rate

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$\text{MIPS} = \frac{\frac{\text{Instruction count}}{\text{Instruction count} \times \text{CPI}} \times 10^6}{\text{Clock rate}} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

Problem - 1

A benchmark program runs on a system having clock rate of 40MHz. The program consists of 100000 executable instructions with following instruction mix and clock cycle count for each instruction type.

Instruction Type	Instruction Count (IC)	Cycles Per Instruction (CPI)
Integer Arithmetic	45000	1
Data Transfer	32000	2
Floating Point	15000	2
Control Transfer	8000	2

Determine the effective CPI, MIPS and execution time for the program

Solution -1

Given Data :

- Clock speed of the processor = 40MHz
- No. of Instructions the executed program consists of = 100000

$$CPI = \frac{\text{Instruction count} \times \text{Cycles per second}}{\text{Number of instructions the executed program consists}}$$

$$\bullet CPI = \frac{(45000 * 1) + (32000 * 2) + (15000 * 2) + (8000 * 2)}{100000} = \frac{155000}{100000} = 1.55$$

Solution -1

- To calculate MIPS

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$\text{MIPS} = \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- $\text{MIPS} = (40 \times 10^6) / (1.55 \times 10^6)$
- $= 25.8$

Solution -1

- To calculate Execution Time
- Execution Time = Instruction Count \times CPI \times Cycle Time
 $= I_c \times CPI \times (1/f)$
 $= (100000 \times 1.55) / (40 \times 10^6)$
 $= 0.003875$
 $= 3.875 \text{ ms}$

Problem - 2

A Netcom systems developed two computer systems C1 and C2, where C1 has machine instructions for floating point (FP) operations as part of its processor ISA and C2 does NOT have floating point instructions as part of its processor ISA. Since C2 does not have floating point instructions, all floating-point instructions will be implemented in Software level with non-FP instructions. You can assume that both systems are operating at a clock speed of 300 Mhz. We are trying to run the SAME program in both the systems which has the following proportion of commands:

Instruction type	% instructions in Program	Instruction Duration (Number of clock periods CPI)	
		C1	C2
Addition of FP numbers	16%	6	20
Multiplication of FP numbers	10%	8	32
Division of FP numbers	8%	10	66
Misc. Instructions (non-FP)	66%	3	3

Solution-2

a) Find the MIPS for both C1 and C2.

For C1:

- CPI =

$$0.16*6 + 0.1*8 + 0.08*10 + 0.66*3 = 4.54$$

- MIPS =

$$300 * 10^6 / (4.54 * 10^6) = 66.08$$

For C2:

- CPI =

$$0.16*20 + 0.1 * 32 + 0.08 * 66 + 0.66 * 3 = 13.66$$

- MIPS =

$$300 * 10^6 / (13.66 * 10^6) = 21.96$$

Solution-2 Contd...

b) Assume that there are 9000 instructions in the program that is getting executed on C1 and C2. What will be the CPU program execution time on each system C1 and C2 ?

- CPU time for C1 of the program execution

$$\begin{aligned}&= \text{No of instructions} * \text{CPI} / \text{Clock rate} \\&= 9000 * 4.54 / (300 * 10^6) \\&= 0.136 \text{ ms}\end{aligned}$$

- CPU time for C2 of the program execution

$$\begin{aligned}&= \text{No of instructions} * \text{CPI} / \text{Clock rate} \\&= 9000 * 13.66 / (300 * 10^6) \\&= 0.41 \text{ ms}\end{aligned}$$

Solution-2 Contd...

c) For the two systems to have the fastest speed and at the same time have equal speed, what would be the possible mixture of the instructions that would be required in the program? WHY?

- For both C_1 and C_2 should be equally fast,
 - Have a program that does NOT have any floating point instructions as CPI for non-floating point instructions is same between C_1 and C_2 .

Problem -3 (Home work)

- Consider two different machines, with two different instruction sets, both of which have a clock rate of 200 MHz. The following measurements are recorded on the two machines running a given set of benchmark programs:

Instruction Type	Instruction Count (millions)	Cycles per Instruction
Machine A		
Arithmetic and logic	8	1
Load and store	4	3
Branch	2	4
Others	4	3
Machine B		
Arithmetic and logic	10	1
Load and store	8	2
Branch	2	4
Others	4	3

- Determine the effective CPI, MIPS rate, and execution time for each machine.

Amdhal's Law

- Deals with the potential speedup of a program using multiple processors (parallel) compared to a single processor.
- It states that if P is the proportion of a program that can be made parallel and $(1-P)$ is the proportion that cannot be parallelized (remains sequential), then the maximum speed up that can be achieved by using N processors is:

$$\text{SpeedUp}(P, N) = \frac{1}{(1-P) + \frac{P}{N}}$$

- As N tends to infinity, the maximum speedup tends to $1/(1-P)$

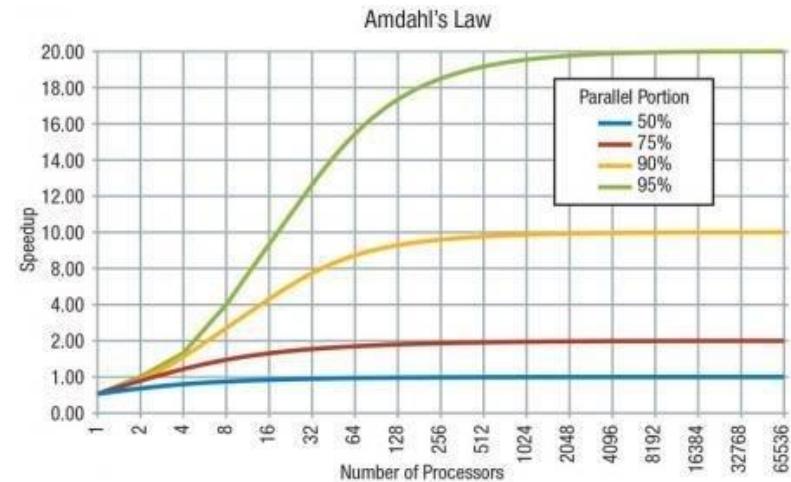
Amdhal's Law

Speedup = Time to execute program on a single processor

Time to execute program on N parallel processors

$$= \frac{T(1-P) + TP}{T(1-P) + TP/N}$$

$$SpeedUp(P, N) = \frac{1}{(1-P) + \frac{P}{N}}$$



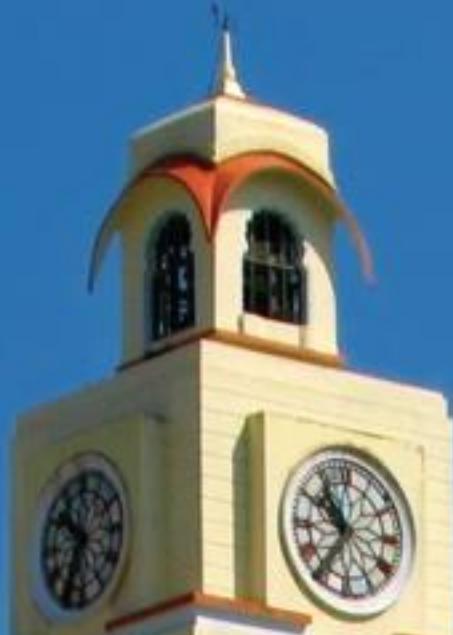
- As N tends to infinity, the maximum speedup tends to $1/(1-P)$

Problem -4 (Amdhal's Law)

- A programmer is given the job to write a program on a computer with processor having speedup factor 3.8 on 4 processors. He makes it 95% parallel and goes home dreaming of a big pay raise. Using Amdahl's law, and assuming the problem size is the same as the serial version, and ignoring communication costs, what is the speedup factor that the programmer will get?
- **Solution:**
- Given Data: No. of Processors (N) = 4
- Proportion of parallelism = 95%
- Speed up = $1/[(1-P)+P/N]$
= $1/[(1-0.95)+0.95/4]$
= 3.47

Problem - 5 (Home Work)

- A programmer has parallelized 99% of a program, but there is no value in increasing the problem size, i.e., the program will always be run with the same problem size regardless of the number of processors or cores used. What is the expected speedup on 20 processors?

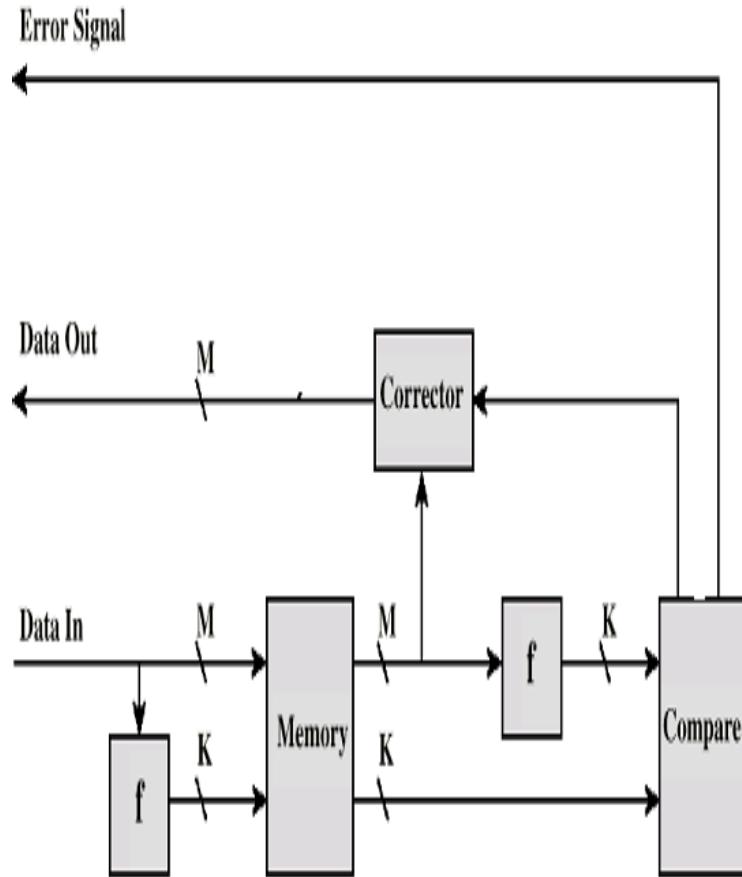


Hamming Code



BITS Pilani
Pilani Campus

Error Correcting Code Function



The comparison logic receives as input two K-bit values. A bit-by-bit comparison is done by taking the exclusive-OR of the two inputs. The result is called the syndrome word.

The comparison yields one of three results

- No errors are detected.
- An error is detected and it is possible to correct the error
- An error is detected but it is not possible to correct it.

Hamming Code.....

What should be the length of the code K

- length of the syndrome word is K bits,
- Length of Data is "M" bits
- Length of K should satisfy

$$2^k - 1 \geq M + K$$

and K has a range between 0 and $2^k - 1$

Problem 1

Let us assume data transmitted is 1011, data corruption in Semiconductor memory has resulted in altering the content of a single memory cell. How do we detect and correct the error using Hamming Code?

Step 1 : Check the length of Check bits required to form Tx word

$$\text{Data} = 1011 \ (\text{M}=4)$$

$$2^k - 1 \geq M + K$$

Here $M = 4$. at least K value should be 3

Hence there are in total 7 bits for transmission

Problem 1

Step 2: How to calculate Check bits and their position

Now we know there are total 7 bits for transmission

Data bits - D₁, D₂, D₃, D₄

Check bits - C₁, C₂ and C₃

— — — — — — —

Problem 1

Step 2: How to calculate Check bits and their position

C3	C2	C1	Position	Bits

Position	7	6	5	4	3	2	1
Bits							
Data							
bits							

- $C1 = D1 \oplus D2 \oplus D4$
- $C2 = D1 \oplus D3 \oplus D4$
- $C3 = D2 \oplus D3 \oplus D4$
- $C1 = 1 \oplus 1 \oplus 1 = 1$
- $C2 = 1 \oplus 0 \oplus 1 = 0$
- $C3 = 1 \oplus 0 \oplus 1 = 0$

Problem 1

Step 2: How to calculate Check bits and their position

C3	C2	C1	Position	Bits
0	0	0		X
0	0	1	1	C1
0	1	0	2	C2
0	1	1	3	D1
1	0	0	4	C3
1	0	1	5	D2
1	1	0	6	D3
1	1	1	7	D4

- $C1 = D1 \oplus D2 \oplus D4$
- $C2 = D1 \oplus D3 \oplus D4$
- $C3 = D2 \oplus D3 \oplus D4$
- $C1 = 1 \oplus 1 \oplus 1 = 1$
- $C2 = 1 \oplus 0 \oplus 1 = 0$
- $C3 = 1 \oplus 0 \oplus 1 = 0$

				2^2		2^1	2^0
Position	7	6	5	4	3	2	1
Bits	D4	D3	D2	C3	D1	C2	C1
Data bits	1	0	1	0	1	0	1

Problem 1 - Data bit corrupted

Step 3: Let us assume the corrupted data received is **1111**. Calculate Check bits for the received word

				2^2		2^1	2^0
Position	7	6	5	4	3	2	1
Bits	D4	D3	D2	C3	D1	C2	C1
Data bits	1	1	1	1	1	1	1

- $C1 = D1 \oplus D2 \oplus D4$
- $C2 = D1 \oplus D3 \oplus D4$
- $C3 = D2 \oplus D3 \oplus D4$
- *Calculated Check bit*
- $C1 = 1 \oplus 1 \oplus 1 = 1$
- $C2 = 1 \oplus 1 \oplus 1 = 1$
- $C3 = 1 \oplus 1 \oplus 1 = 1$

Step 4: Compare Check bits to generate Syndrome

Previous checkbits: 001

Calculated check bits : \oplus 111

Syndrome : **110**

Toggle the $(110)_2$ or 6th position bit to get the correct data.

Here Corrected data is **1010111**. Extracting Data it yields 1011

Problem 1 - Check bit is corrupted

Step 3: Let us assume the data received is same 1011.
Calculate Check bits for the received word

				2^2		2^1	2^0
Position	7	6	5	4	3	2	1
Bits	D4	D3	D2	C3	D1	C2	C1
Data bits	1	0	1	0	1	0	0

- $C1 = D1 \oplus D2 \oplus D4$
- $C2 = D1 \oplus D3 \oplus D4$
- $C3 = D2 \oplus D3 \oplus D4$
- $C1 = 1 \oplus 1 \oplus 1 = 1$ (**0**)
- $C2 = 1 \oplus 0 \oplus 1 = 0$
- $C3 = 1 \oplus 0 \oplus 1 = 0$

Step 4: Compare Check bits to generate Syndrome

Computed checkbits: 001

Received check bits : \oplus 000

Syndrome : 001

Change in Single bit indicates error in one of the check bits. Only error signal is generated.

Problem 1. Conclusion...

- If the syndrome contains all 0s, no error has been detected.
- If the syndrome contains one and only one bit set to 1, then an error has occurred in one of the 3 check bits. No correction is needed.
- If the syndrome contains more than one bit set to 1, then the numerical value of the syndrome indicates the position of the data bit in error. This data bit is inverted for correction.

Problem 2 - Homework

Consider a 12 bit (Data + Check bit) Code 111101011101
Find out if there is an error. If so which bit is having error?

Layout of Data and Check bits

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C4				C3		C2	C1

- $C1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7$
- $C2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7$
- $C3 = D2 \oplus D3 \oplus D4 \oplus D8$
- $C4 = D5 \oplus D6 \oplus D7 \oplus D8$

\oplus : Bitwise XOR operation



CPU-OS Simulator

BITS Pilani
Pilani Campus

CPU OS Simulator

- Components of CPU Simulator
- Execute a program
- Cache updates
- Assignment

CPU-OS Simulator Interface

innovate

achieve

lead

CPU Simulator: CPU 0 [YASMIN: CPU-OS Simulator, Version: 7.5.50, Copyright @ 2006-2013, Besim Mustafa, Edge Hill University, UK]

Cache - Pipeline	Execution Unit
<p>Pipeline</p> <p><input checked="" type="radio"/> Single pipeline <input type="radio"/> Dual pipeline</p> <p>Select pipeline</p> <p><input style="border: 1px solid #0070C0; padding: 5px; width: 150px; height: 30px; margin-top: 10px;" type="button" value="SHOW PIPELINE..."/></p>	
<p>Cache</p> <p>Select cache type</p> <p><input style="border: 1px solid #0070C0; padding: 5px; width: 150px; height: 30px; margin-top: 10px;" type="button" value="SHOW CACHE..."/></p>	

SPECIAL CPU REGISTERS					
PC	0	SR	0		
SP	8096	BR	0		
SR Status Flag					
OV	<input type="checkbox"/>	Z	<input type="checkbox"/>		
	N	<input type="checkbox"/>			
IR					
MAR	0				
MDR					
CPU Mode					
User					
Kernel		<input checked="" type="radio"/>			

Reg	Val (D)	C	Val (D)	
R06	0			
R07	0			
R08	0			
R09	0			
R10	0			
R11	0			
R12	0			
R13	0			
R14	0			
R15	0			
R16	0			
R17	0			
R18	0			
R19	0			
R20	0			
R21	0			
R22	0			
R23	0			
R24	0			
R25	0			
R26	0			
R27	0			
R28	0			
R29	0			
R30	0			
R31	0			

Program Instructions Optimize - Assemble		
New Program Program Name <input type="text"/> Base Address <input type="text"/>	Pages 1 <input type="button" value="▼"/> ADD <input type="button" value="SAVE..."/> <input type="button" value="LOAD..."/> <input type="button" value="COPY TO CLIPBOARD"/>	Programs Program List ALL <input type="button" value="▼"/> Base Address <input type="checkbox"/> -1

The screenshot shows the 'Program Control' tab selected. It includes buttons for 'STEP', 'RUN', and 'STOP'. A slider labeled 'Fast' at the top and 'Slow' at the bottom controls the execution speed. To the right are links to 'CPU View', 'CPU Help', 'RESET PROGRAM', and 'SHOW PCB...'.

The screenshot shows the 'Advanced' tab selected in a 'New CPU' dialog box. The interface includes tabs for 'CPU', 'Advanced', and 'New CPU'. Below the tabs are five buttons: 'COMPILER...', 'OS 0...', 'INPUT OUTPUT...', 'VIRTUAL OS...', and 'INTERRUPTS...'. The 'COMPILER...' button is highlighted with a blue border.

Registers	Program Stack	Watch
<div style="display: flex; justify-content: space-around;"> Reg Value <input type="button" value="CHANGE"/> <input type="button" value="RESET ALL"/> </div> <hr/> <div style="display: flex; align-items: center;"> Show Reg Access Status <input type="checkbox"/> </div> <div style="display: flex; align-items: center;"> Select Register Set Size <input type="button" value="32"/> </div>		

How to Simulate and Run the program



Simulator Teaching Language (STL)

```
program SelectionSort
VAR a array(10) INTEGER
a(1)=15
a(2)=20
a(3)=8
a(4)=100
a(5)=30
for n = 1 to 5
writeln("num",a(n))
next
for i = 1 to 5
for j = i+1 to 5
p = a(i)
q = a(j)
if p > q then
x = p
a(i) = q
a(j) = x
end if
next
writeln("Sorted Array in ascending order")
for n = 1 to 5
writeln("num",a(n))
next
end
```

How to Simulate and Run the program



Simulator Teaching Language (STL)

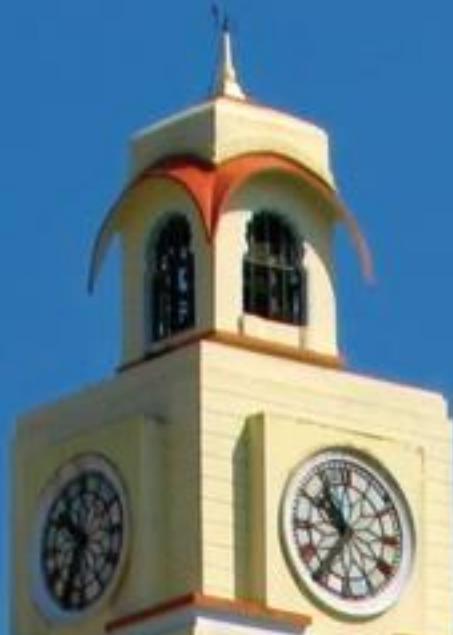
- a) Execute the above program by setting block size to 2, 4, 8, 16 and 32 for cache size = 8, 16 and 32. Record the observation in the following table.

Block Size	Cache size	# Hits	# Misses	% Miss Ratio	% Hit Ratio
2	8				
4					
8					
2	16				
4					
8					

- b) Plot a single graph of Cache hit ratio Vs Block size with respect to cache size = 8, 16 and 32. Comment on the graph that is obtained.

Resources

- Link to download CPU-OS Simulator
- https://drive.google.com/drive/folders/12YUK52RQ-JhPOddj6CD_oifW4sTMbsBl?usp=sharing
- Introduction to CPU Simulator
- https://www.youtube.com/watch?v=izcvk0x7kKM&list=PLGz_KyS7cuuWGqFbBITk7Nr4PoVmAWkEO
- Installation on Mac OS
- <https://www.youtube.com/watch?v=2vBjXUbvCKs>



BITS Pilani
Pilani Campus

Webinar-2 Cache Memory

by

VIVEKANANDA M R

vivekanandamr@wilp.bits-pilani.ac.in



BITS Pilani
Pilani Campus

Direct Mapping



Direct Mapped Cache - Summary

- Each block of main memory maps to only one cache line
 - if a block is in cache, it must be in one specific place
 - $i = j \text{ modulo } m$
where i = cache line number
 j = main memory block no.
 m = no.of lines in the cache
- Address is split in three parts:
 - Tag
 - Line
 - Word

Problem- 1: Direct mapping

Consider a cache memory of 8192KB with line size of 128B. Find the number of bits required for TAG, LINE and WORD fields of the main memory address 0xFEEDFOOD?

Solution-1:

- No. of bits needed for the Main Memory Address
= 32 bits
- Block Size
= 128Bytes
- No. of bits to represent WORD
 $128 \text{ Bytes} = 2^7 \Rightarrow 7 \text{ bits}$

Problem- 1: Direct mapping

Consider a cache memory of 8192KB with line size of 128B. What is the tag, line and word for the main memory address 0xFEEDFOOD?

Solution-1

- Cache Size = 8192 KB
- Total No. of Cache Lines
 - = Cache size / Line size
 - = $8192 \text{ KB}/128\text{B} = (2^{13}.2^{10}) / 2^7 = 2^{16} = 64\text{K lines}$
- Total bits to represent line field
 - = 16 bits
- Total bits to represent Tag filed is:
 - = ~~32 - Line - Word = 9 bits~~

Problem- 1: Direct mapping

Given Address : 0xFEEDFOOD



BITS Pilani
Pilani Campus

Associative Mapping

Associative Mapping - Summary

- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line's tag is examined for a match
- Cache searching gets expensive

Problem 2: Fully Associative

A system has main memory of size 128Byte with on chip cache 32 Bytes and block size of 8 Bytes, with the system having fully associative cache mapping, find the following:

- a) The number of main memory address bits.

Main Memory Size = 128 Bytes = 2^7 Bytes

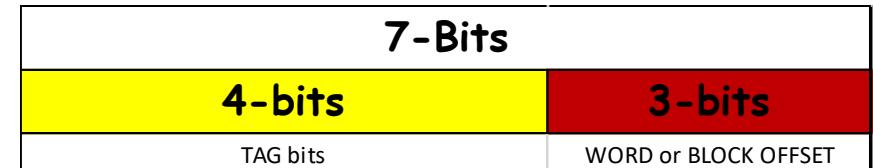
Thus, the size of Physical Address = 7 bits.

- b) The number of Tag bits and Word bits.

Block size = 8 Bytes = 2^3 Bytes

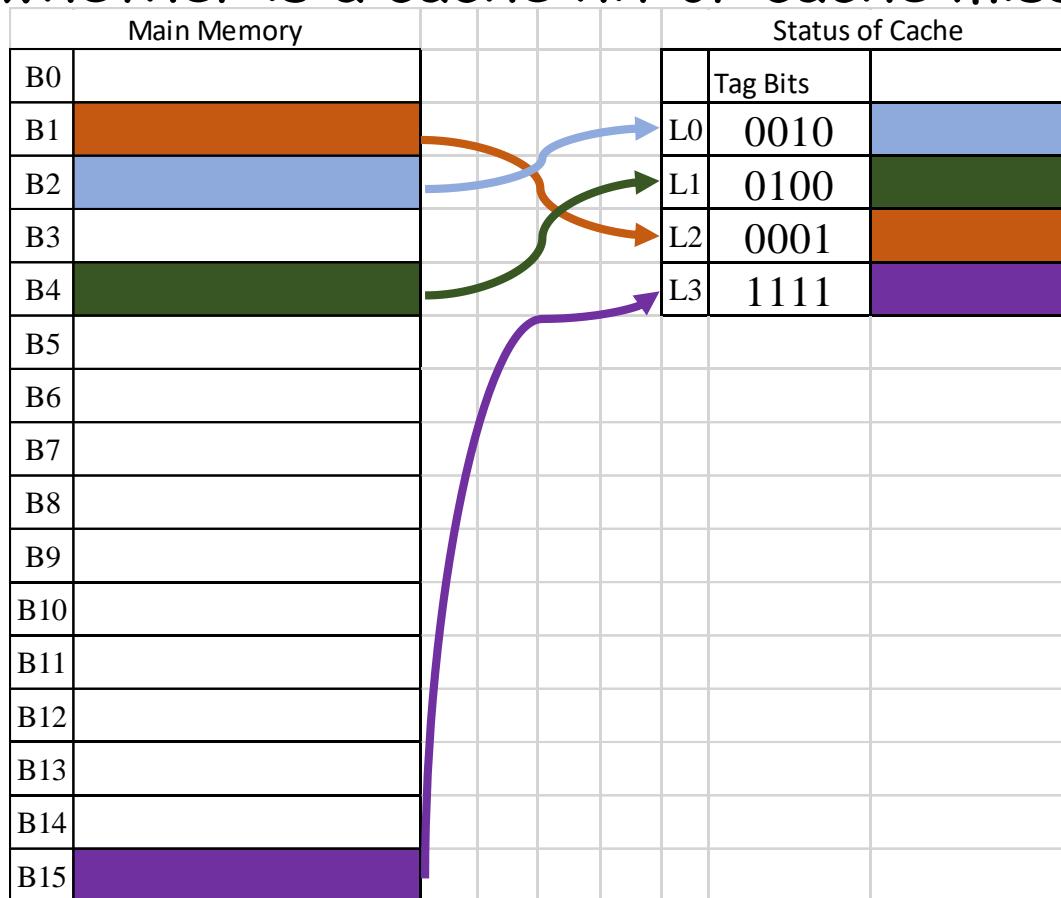
of bits for WORD offset field = 3 bits

of Tag bits = 7 - WORD offset = $7-3 = 4$ bits



Fully Associative

c) If the CPU request the addresses 0001100, 0011001 find whether is a cache hit or cache miss for each of the address.



0001100

0001100 - Hit

0011001

0011001-Miss

Problem 3: Fully Associative

An 8KB associative cache has a line size of 32 Bytes. Main memory size is 1GB. Find the number of TAG bits and number of comparators required for search.

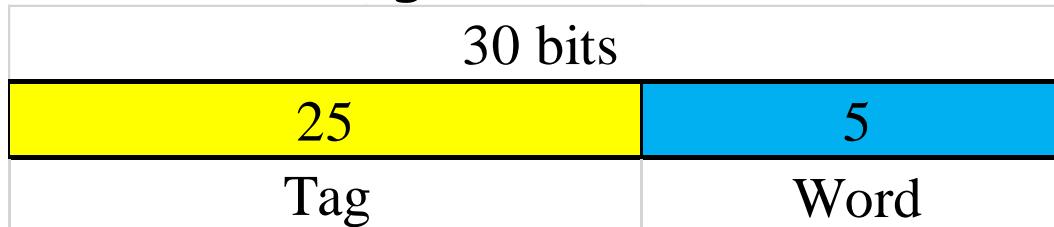
Given Data:

$$\text{Cache size} = 8\text{KB} = 2^3 \cdot 2^{10} = 2^{13} \text{ Bytes}$$

$$\text{Block Size} = 32 \text{ Bytes} = 2^5 \text{ Bytes}$$

$$\text{Memory size} = 1\text{GB} = 2^{30} \text{ Bytes}$$

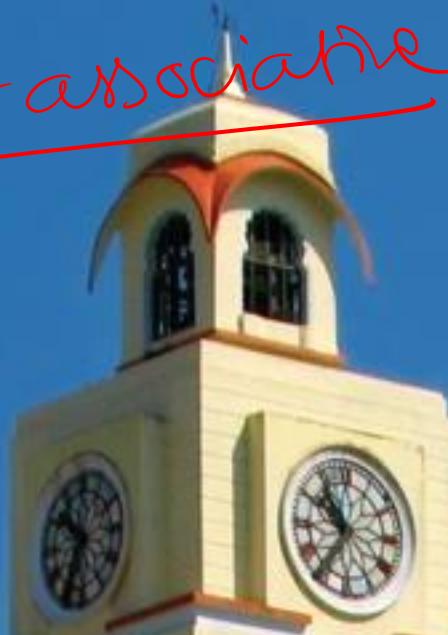
- # of bits for main memory address = 30 bits
- # of bits for Word offset field = 5 bits
- # of bits for Tag field = $30 - 5 = 25$ Bits



Problem 3: Fully Associative

- # of cache lines
= (Cache Size)/(Line Size) = $2^{13}/2^5 = 2^8$ lines = 256 lines
- # of Comparators required = # of Cache lines = 256
- Size of comparator = size of tag bits
= 25-bit comparator

K-Say Set associative



BITS Pilani
Pilani Campus

Set Associative Mapping

Lakshmikantha G C

Set Associative - Summary

- Makes use of advantages of Direct Mapped and Associative mapping
 - Block is mapped to a set (Direct mapping)
 - Within the block can be placed in any line (Associative)
 - K-Way associative cache \rightarrow K lines in a set
 - Mapping function : $i = j \% v$
- Where i = cache set number j = main memory block number v = number of sets in the cache
- 2-Way Set



Problem : Set Associative

A set associative cache memory consists of 128 lines divided into four line sets. The main memory consists of 16,384 blocks and each block contains 256 eight bit words.

a) How many bits are required for addressing the main memory?

memory?

Size of main memory = 16384 blocks

= $16384 \times 256 \times 8$ bits

= $16KB \times 256B$

= $2^{14} \times 2^8$

= 2^{22} bytes

~~2^{22} bytes~~

$\leftarrow 22$ bits

\rightarrow

B_0

M

B_0

\vdots

L_n

G

16384

127

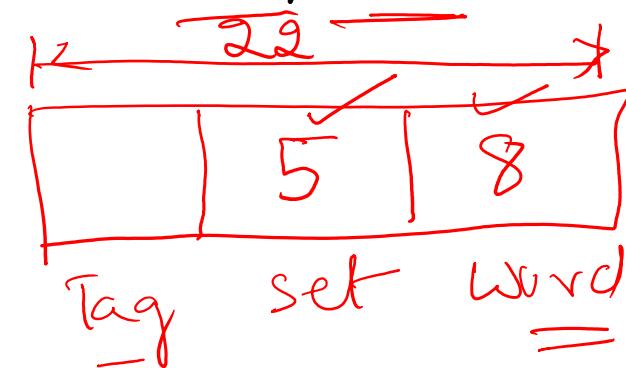
Problem : Set Associative

A set associative cache memory consists of 128 lines divided into four line sets. The main memory consists of 16,384 blocks and each block contains 256 eight bit words.

b) How many bits are needed to represent the TAG, SET and WORD fields?

$$\begin{array}{l} \text{Block size} = 256 \text{ bytes} \\ \text{Block size} = 2^8 \text{ bytes} \end{array}$$

$$22 - (5 + 8)$$



of bits in WORD offset = 8 bits

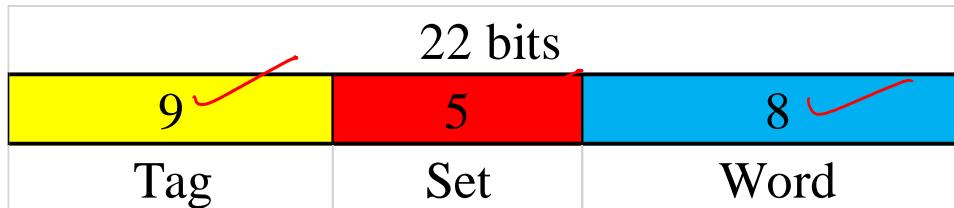
$$\begin{array}{l} \text{Number of sets in cache} = \frac{\text{Number of lines in cache}}{\text{Set size}} \\ = \frac{128 \text{ Lines}}{4 \text{ line set}} = 32 \text{ sets} \\ = 2^5 \text{ sets} \end{array}$$

of bits in set number = 5 bits

Problem : Set Associative

Number of bits in TAG = Number of bits in physical address -
 (Number of bits in set number + Number of bits in word)
 = 22 bits - (5 bits + 8 bits)
 = 9 bits

Thus, Number of bits in TAG = 9 bits

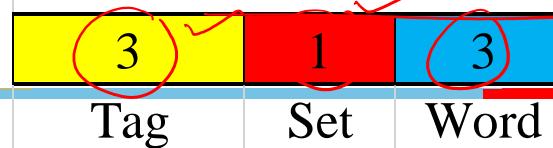


Problem : Set Associative

Add xcm

A system has a main memory of 128 Bytes and a cache size of 32 Bytes with 8 Bytes per cache block. Assume that the size of each memory word is 1 byte. if the cache is organized as a 2-way set-associative cache. Find out TAG, SET, and WORD field bits.

- Block size = 8 bytes = 2^3 Bytes \rightarrow WORD = 3bits
- Cache size = 32 Bytes = 2^5 Bytes
- Number of lines = Cache size / Block size
 $= 2^5 / 2^3$ Bytes = $2^2 \rightarrow 4$
- Number of cache lines per set = 2 (2-way set associative)
- Number of Sets = Number of lines/lines per Set = $4 / 2 = 2$
- # bits for identifying a SET = 1
- Total number of address bits = 7 (128 Bytes MM = 2^7)
- TAG = $7 - (3 + 1) = 3$ bits



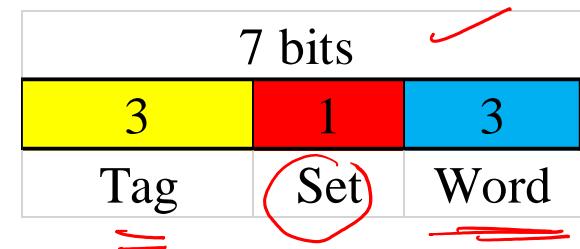
Problem : Set Associative

b) When a program is executed, the processor requests data from the following word addresses:

~~0001010 0010010~~

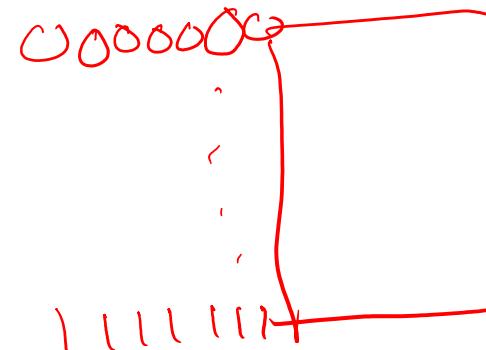
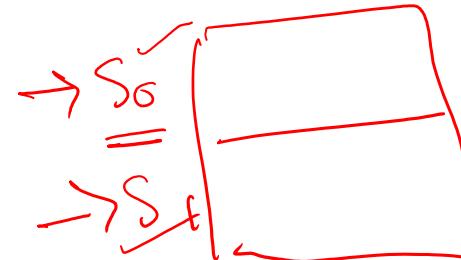
For each of the above addresses, find out for which set it will map.

- For the word address: ~~0001010~~
~~000 1 010~~ → Set-1



- For the word address: ~~0010010~~

~~000 0 010~~ → Set-0





BITS Pilani
Pilani Campus

Replacement Algorithms

Replacement Algorithms (2/3)

- Needed in Associative & Set Associative mapped cache
- Methods:
 - Least Recently Used (LRU) ←
 - Least Frequently Used (LFU)
 - First In First Out (FIFO)
 - Random

Replacement Algorithms (3/3)

- ~~Least Recently used (LRU)~~: Replace the block in the set that has been in the cache longest with no reference to it
- ~~Least frequently used~~: Replace block which has had fewest hits
 - Uses counter with each line
- ~~First in first out (FIFO)~~: Replace block that has been in cache longest
 - Round robin or circular buffer technique
- ~~Random~~

ARM

1866

Problem

Consider a reference pattern that accesses the sequence of blocks 4, 7, 6, 1, 7, 6, 1, 2, 7, 2, 5, 4. Assuming that the cache uses associative mapping, find the hit ratio with a cache of four lines.

- a) LRU
- b) LFU
- c) FIFO

LRU

Replacement ←

Ref	4	7	6	1	7	6	1	2	7	2	5
time	0	1	2	3	4	5	6	7	8	9	10
L0	4 ₀	2 ₇	2 ₇	2 ₉	2 ₉						
L1		7 ₁	7 ₁	7 ₁	7 ₄	7 ₄	7 ₄	7 ₄	7 ₈	7 ₈	7 ₈
L2			6 ₂	6 ₂	6 ₂	6 ₅	5 ₁₀				
L3				1 ₃	1 ₃	1 ₃	1 ₆				
H/M	M	M	M	M	(H)	(H)	(H)	M	(H)	(H)	M

Cache Miss → 6 total = 12
Cache Hit → 5

LFU Counter ←

Diagram illustrating the LFU Counter mechanism. A horizontal bar at the top is divided into three colored segments: yellow (left), light blue (middle), and red (right). Red arrows point downwards from the center of each segment to the corresponding row in the memory table below. Red circles highlight the values in the 6th, 11th, and 12th columns of the table.

Ref	4	7	6	1	7	6	1	2	7	2	5
L0	4 ₁	2 ₁	2 ₁	2 ₂	2 ₂						
L1		7 ₁	7 ₁	7 ₁	7 ₂	7 ₂	7 ₂	7 ₂	7 ₃	7 ₃	7 ₃
L2			6 ₁	6 ₁	6 ₁	6 ₂	5 ₁				
L3				1 ₁	1 ₁	1 ₁	1 ₂				
H/M	M	M	M	M	H	H	H	M	N/A	H	M

FIFO



Replace



Ref	4	7	6	1	7	6	1	2	7	2	5
time	0	1	2	3	4	5	6	7	8	9	10
L0	4 ₀	2 ₇	2 ₇	2 ₇	2 ₇						
L1		7 ₁	5 ₁₀								
L2			6 ₂								
L3				1 ₃							
H/M	M	M	M	M	(H)(C)	(H)	(H)	M	(H)	(H)	M



K-way

Problem : Set Associative (LRU)

S₀
S₁

Consider a 2 - way set associative cache with 4 cache lines (0-3). The memory block requests are in the order-
4, 3, 25, 8, 4, 6, 25, 8, 16, 35, 4

If **LRU replacement** policy is used, calculate the hit ratio and miss ratio

Problem : Set Associative (LRU)

	4	3	25	8	4	6	25	8	16	35	4
Time	0	1	2	3	4	5	6	7	8	9	10
Set 0	L0										
	L1										
Set 1	L2										
	L3										
	H/M										

Problem : Set Associative (LRU)

innovate

achieve

lead

BN (j)	4	3	25	8	4	6	25	8	16	35	4
#of Sets(v)	2	2	2	2	2	2	2	2	2	2	2
$i=j \% v$	0	1	1	0	0	0	1	0	0	1	0
Time	4	3	25	8	4	6	25	8	16	35	4
L0	4_0	4_0	4_0	4_0	4_4	4_4	4_4	8_7	8_7	8_7	4_{10}
Set 0	4_0	4_0	4_0	4_0	8_3	8_3	6_5	6_5	6_5	35_9	35_9
L1	3_1	3_1	25_2	25_2	25_2	25_2	25_6	25_6	25_6	25_6	25_6
Set 1	M	M	M	M	H	M	H	M	M	M	M
H/M											

Problem : Set Associative (FIFO)

Consider a 2 - way set associative cache with 4 cache lines (0-3). The memory block requests are in the order-

4, 3, 25, 8, 4, 6, 25, 8, 16, 35, 4

If FIFO replacement policy is used, calculate the hit ratio and miss ratio

Problem : Set Associative (FIFO)

		4	3	25	8	4	6	25	8	16	35	4
	Time	0	1	2	3	4	5	6	7	8	9	10
Set 0	L0											
	L1											
Set 1	L2											
	L3											
	H/M											

Problem : Set Associative (FIFO)

BN (j)	4	3	25	8	4	6	25	8	16	35	4
#of Sets(v)	2	2	2	2	2	2	2	2	2	2	2
i=j%v	0	1	1	0	0	0	1	0	0	1	0

	4	3	25	8	4	6	25	8	16	35	4
Time	0	1	2	3	4	5	6	7	8	9	10
L0	4_0	4_0	4_0	4_0	4_0	6_5	6_5	6_5	6_5	6_5	4_{10}
Set 0											
L1				8_3	8_3	8_3	8_3	8_3	16_8	16_8	16_8
Set 1				3_1	3_1	3_1	3_1	3_1	3_1	35_9	35_9
L3				25_2	25_2	25_2	25_2	25_2	25_2	25_2	25_2
H/M	M	M	M	M	H	M	H	H	M	M	M

Home work Problem :

A Research center wants to check whether implementing cache replacement using two existing cache memory replacement algorithms LFU and FIFO would help reduce the miss rate. The proposed algorithm would work in two phases. The first 6 clocks (0-5) follow LFU and the next 6 clocks (6-11) follow FIFO. The main memory block sequence is 0, 4, 0, 2, 1, 5, 0, 1, 2, 5, 0, 2

What is the Hit Ratio for the proposed new replacement algorithm?

Justify your answer by filling in the following table. In LFU, in case of a tie between cache lines for replacement, select the line which has been there for a longer time in the cache.

Problem : Set Associative (FIFO)

Ref	0	4	0	2	1	5	0	1	2	5	0	2
time	0	1	2	3	4	5	6	7	8	9	10	11
L0												
L1												
L2												
L3												
H/M												

Thank You



BITS Pilani
Pilani Campus

- Dr Selva Kumar and Dr H P Srinivasa



COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS

WEBINAR 4 – BUDDY SYSTEMS , DEADLOCKS & SEMAPHORES

Topics

- Buddy System
 - Deadlock Avoidance Algorithm
 - Banker's Algorithm
 - Resource Request Algorithm
 - Deadlock Detection Algorithm
 - Semaphores
-

Buddy System

- The **buddy system** is a memory allocation and management algorithm.
- It manages memory in **power of two increments**.
- Splitting memory into halves and to try to give a best fit.
- Provides two operations:
 - $\text{Allocate}(2^k)$: Allocates a block of 2^k and marks it as allocated
 - $\text{Free}(A)$: Marks the previously allocated block A as free and merge it with other blocks to form a larger block.
- Algorithm: Assume that a process A of size “X” needs to be allocated
 - **If $2^{K-1} < X \leq 2^K$:** Allocate the entire block
 - **Else:** Recursively divide the block equally and test the condition at each time, when it satisfies, allocate the block.

Problem : Buddy System

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)

Problem : Buddy System

- 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) | |



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) | |

Problem : Buddy System

- Allocate (E: 3.2K)



- Free (C)



- Free (B) - coalescing



- Allocate (F: 1.6K)



- Allocate (G: 1.8K)



Advantages and Disadvantages

Advantage

- Easy to implement a buddy system
- Allocates block of correct size
- It is easy to merge adjacent holes
- Fast to allocate memory and de-allocating memory

Disadvantage

- It requires all allocation unit to be powers of two
- It leads to internal fragmentation

Banker's Safety Algorithm

1. Let **Available** and **Finish** be vectors of length m and n respectively, where m and n represents number of processes and resources respectively. Initialize:

Finish [i] = false for $i = 0, 1, \dots, n-1$

2. Find an i such that both:

(a) **Finish [i] = false**

(b) **Need_i ≤ Available**

If no such i exists, go to step 4

3. **Available = Available + Allocation_i**

Finish[i] = true

go to step 2

4. If **Finish [i] == true** for all i , then the system is in a safe state

Problem : Banker's Algorithm

Apply Banker's algorithm for the following and find out whether the system is in safe state or not.

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	4	0	0	1	6	0	1	2	3	2	1	1
P1	1	1	0	0	2	7	5	0				
P2	1	2	5	4	2	3	5	6				
P3	0	6	3	3	1	6	5	3				
P4	0	2	1	2	1	6	5	6				
Total	9	13	10	11								

Need Matrix

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	4	0	0	1	6	0	1	2	3	2	1	1
P1	1	1	0	0	2	7	5	0				
P2	1	2	5	4	2	3	5	6				
P3	0	6	3	3	1	6	5	3				
P4	0	2	1	2	1	6	5	6				

Need = Max - Allocation

	A	B	C	D
P0	2	0	1	1
P1	1	6	5	0
P2	1	1	0	2
P3	1	0	2	0
P4	1	4	4	4

Problem : Banker's Algorithm

Process	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	4	0	0	1	6	0	1	2	3	2	1	1	2	0	1	1
P1	1	1	0	0	2	7	5	0	7	2	1	2	1	6	5	0
P2	1	2	5	4	2	3	5	6					1	1	0	2
P3	0	6	3	3	1	6	5	3					1	0	2	0
P4	0	2	1	2	1	6	5	6					1	4	4	4

Step 1:

$$\text{Work} = \text{Available} = 3 \ 2 \ 1 \ 1$$

0 1 2 3 4

Finish[i] =

F	F	F	F	F
---	---	---	---	---

Step 2:

For i = 0

Finish[0] = F & Need[0] <= Work

2 0 1 1 <= 3 2 1 1 (T)

P0 -> Safe sequence

Step 3:

$$\text{Work} = \text{Work} + \text{Allocation}[0]$$

$$\text{Work} = 3 \ 2 \ 1 \ 1 + 4 \ 0 \ 0 \ 1 = 7 \ 2 \ 1 \ 2$$

0 1 2 3 4

Finish[i] =

T	F	F	F	F
---	---	---	---	---

Step 2:

For i = 1

Finish[1] = F & Need[1] <= Work

1 6 5 0 <= 7 2 1 2 (F)

P1 -> Wait

Problem : Banker's Algorithm

Process	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	4	0	0	1	6	0	1	2	3	2	1	1	2	0	1	1
P1	1	1	0	0	2	7	5	0	7	2	1	2	1	6	5	0
P2	1	2	5	4	2	3	5	6	8	4	6	6	1	1	0	2
P3	0	6	3	3	1	6	5	3	8	10	9	9	1	0	2	0
P4	0	2	1	2	1	6	5	6					1	4	4	4

Step 2:

For i = 2

Finish[2] = F & Need[2] <= Work

1 1 0 2 <= 7 2 1 2 (T)

P2 -> Safe sequence

Step 3:

Work = Work + Allocation[2]

Work = 7 2 1 2 + 1 2 5 4 = 8 4 6 6

0 1 2 3 4

Finish[i] =

T	F	T	F	F
---	---	---	---	---

Step 2:

For i = 3

Finish[3] = F & Need[3] <= Work

1 0 2 0 <= 8 4 6 6 (T)

P3 -> Safe sequence

Step 3:

Work = Work + Allocation[3]

Work = 8 4 6 6 + 0 6 3 3 = 8 10 9 9

0 1 2 3 4

Finish[i] =

T	F	T	T	F
---	---	---	---	---

Problem : Banker's Algorithm

Process	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	4	0	0	1	6	0	1	2	3	2	1	1	2	0	1	1
P1	1	1	0	0	2	7	5	0	7	2	1	2	1	6	5	0
P2	1	2	5	4	2	3	5	6	8	4	6	6	1	1	0	2
P3	0	6	3	3	1	6	5	3	8	10	9	9	1	0	2	0
P4	0	2	1	2	1	6	5	6	8	12	10	11	1	4	4	4

Step 2:

For i = 4

Finish[4] = F & Need[4] <= Work

1 4 4 4 <= 8 10 9 9 (T)

P4 -> Safe sequence

Step 3:

Work = Work + Allocation[4]

Work = 8 10 9 9 + 0 2 1 2 = 8 12 10 11

0 1 2 3 4

Finish[i] = T F T T T

Step 2:

For i = 1

Finish[1] = F & Need[1] <= Work

1 6 5 0 <= 8 12 10 11 (T)

P1-> Safe sequence

Step 3:

Work = Work + Allocation[3]

Work = 8 12 10 11 + 1 1 0 0 = 9 13 10 11

0 1 2 3 4

Finish[i] = T T T T T

Safe sequence is <P0, P2, P3, P4, P1>

Resource-Request Algorithm

Request_i = request vector for process P_i .

If $\text{Request}_i[j] = k$ then process P_i wants k instances of resource type R_j

1. If $\text{Request}_i \leq \text{Need}_i$, go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If $\text{Request}_i \leq \text{Available}$, go to step 3. Otherwise P_i must wait, since resources are not available
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$$\text{Available} = \text{Available} - \text{Request}_i;$$

$$\text{Allocation}_i = \text{Allocation}_i + \text{Request}_i;$$

$$\text{Need}_i = \text{Need}_i - \text{Request}_i;$$

- | If safe \Rightarrow the resources are allocated to P_i
- | If unsafe $\Rightarrow P_i$ must wait, and the old resource-allocation state is restored

Problem: Resource- Request Algorithm

Apply Banker's algorithm for the following and find out whether the system is in safe state or not. If process P1 requests for additional resources (1,2,0,0) will the system go to unsafe state or not. Check using Resource-Request algorithm.

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	4	0	0	1	6	0	1	2	3	2	1	1
P1	1	1	0	0	2	7	5	0				
P2	1	2	5	4	2	3	5	6				
P3	0	6	3	3	1	6	5	3				
P4	0	2	1	2	1	6	5	6				

Problem: Resource- Request Algorithm

Process	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	4	0	0	1	6	0	1	2	3	2	1	1	2	0	1	1
P1	1	1	0	0	2	7	5	0					1	6	5	0
P2	1	2	5	4	2	3	5	6					1	1	0	2
P3	0	6	3	3	1	6	5	3					1	0	2	0
P4	0	2	1	2	1	6	5	6					1	4	4	4

Step 1:
 $\text{Request}[1] \leq \text{Need}[1]$
 $1\ 2\ 0\ 0 \leq 1\ 6\ 5\ 0$

Step 2:
 $\text{Request}[1] \leq \text{Available}$
 $1\ 2\ 0\ 0 \leq 3\ 2\ 1\ 1$

Step 3:
 $\text{Allocation}[1] = \text{Allocation}[1] + \text{Request}[1] = 1\ 1\ 0\ 0 + 1\ 2\ 0\ 0 = 2\ 3\ 0\ 0$
 $\text{Available} = \text{Available} - \text{Request}[1] = 3\ 2\ 1\ 1 - 1\ 2\ 0\ 0 = 2\ 0\ 1\ 1$
 $\text{Need}[1] = \text{Need}[1] - \text{Request}[1] = 1\ 6\ 5\ 0 - 1\ 2\ 0\ 0 = 0\ 4\ 5\ 0$

Problem: Resource- Request Algorithm

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	4	0	0	1	2	0	1	1	2	0	1	1
P1	2	3	0	0	0	4	5	0				
P2	1	2	5	4	1	1	0	2				
P3	0	6	3	3	1	0	2	0				
P4	0	2	1	2	1	4	4	4				

Problem: Resource- Request Algorithm

Process	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	4	0	0	1	6	0	1	2	2	0	1	1	2	0	1	1
P1	2	3	0	0	2	7	5	0	6	0	1	2	0	4	5	0
P2	1	2	5	4	2	3	5	6					1	1	0	2
P3	0	6	3	3	1	6	5	3					1	0	2	0
P4	0	2	1	2	1	6	5	6					1	4	4	4

Step 1:

$$\text{Work} = \text{Available} = 2\ 0\ 1\ 1$$

0 1 2 3 4

Finish =



Step 3:

$$\text{Work} = \text{Work} + \text{Allocation}[0]$$

$$\text{Work} = 2\ 0\ 1\ 1 + 4\ 0\ 0\ 1 = 6\ 0\ 1\ 2$$

0 1 2 3 4

Finish =



Step 2:

For i=0

Finish[0] = F & Need[0] <= Work

$$2\ 0\ 1\ 1 \leq 2\ 0\ 1\ 1 \text{ (T)}$$

P0 -> Safe sequence

Step 2:

For i=1

Finish[1] = F & Need[1] <= Work

$$0\ 4\ 5\ 0 \leq 6\ 0\ 1\ 2 \text{ (F)}$$

P1 -> Wait

Problem: Resource- Request Algorithm

Process	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	4	0	0	1	6	0	1	2	3	2	1	1	2	0	1	1
P1	1	1	0	0	2	7	5	0	6	0	1	2	1	6	5	0
P2	1	2	5	4	2	3	5	6					1	1	0	2
P3	0	6	3	3	1	6	5	3					1	0	2	0
P4	0	2	1	2	1	6	5	6					1	4	4	4

Step 2:
 For i=2
 $\text{Finish}[2] = \text{F} \& \text{Need}[2] \leq \text{Work}$
 $1\ 1\ 0\ 2 \leq 6\ 0\ 1\ 2 \text{ (F)}$
P2 -> Wait

Step 2:
 For i=4
 $\text{Finish}[4] = \text{F} \& \text{Need}[4] \leq \text{Work}$
 $1\ 4\ 4\ 4 \leq 6\ 0\ 1\ 2 \text{ (F)}$
P4 -> Wait

Step 2:
 For i=3
 $\text{Finish}[3] = \text{F} \& \text{Need}[3] \leq \text{Work}$
 $1\ 0\ 2\ 0 \leq 6\ 0\ 1\ 2 \text{ (F)}$
P3 -> Wait

The system is in unsafe state

Deadlock Detection Algorithm

1. Let \mathbf{Work} and \mathbf{Finish} be vectors of length m and n , respectively Initialize:
 - (a) $\mathbf{Work} = \mathbf{Available}$
 - (b) For $i = 1, 2, \dots, n$, if $\mathbf{Allocation}_i \neq \mathbf{0}$, then
 $\mathbf{Finish}[i] = \text{false}$; otherwise, $\mathbf{Finish}[i] = \text{true}$
2. Find an index i such that both:
 - (a) $\mathbf{Finish}[i] == \text{false}$
 - (b) $\mathbf{Request}_i \leq \mathbf{Work}$
If no such i exists, go to step 4
3. $\mathbf{Work} = \mathbf{Work} + \mathbf{Allocation}_i$
 $\mathbf{Finish}[i] = \text{true}$
go to step 2
4. If $\mathbf{Finish}[i] == \text{false}$, for some i , $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $\mathbf{Finish}[i] == \text{false}$, then P_i is deadlocked

Problem : Deadlock Detection Algorithm

Consider the following snapshot of the system with four processes P0 to P3 and 3 resource types A(5 Instances), B(3Instances), and C(8 Instances).

Answer the following questions with reference to Deadlock Detection Algorithm.

- Check whether the system is in deadlock or not.
- If the system is in safe state then give safe sequence or else provide the process number(s) which is causing the deadlock.

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	1	0	2	0	0	1	0	0	0
P1	2	1	1	1	0	2			
P2	1	0	3	0	0	0			
P3	1	2	2	3	3	0			

Problem : Deadlock Detection Algorithm

Initialization:

Work = Available = 0 0 0

0	1	2	3
---	---	---	---

Finish =

F	F	F	F
---	---	---	---

Process	Allocation			Request		
	A	B	C	A	B	C
P0	1	0	2	0	0	1
P1	2	1	1	1	0	2
P2	1	0	3	0	0	0
P3	1	2	2	3	3	0

Step 2:

For i=0

Finish[0] = F & Request[0] <= Work

0 0 1 <= 0 0 0 (F)

P0 -> Wait

Step 2:

For i=1

Finish[1] = F & Request[1] <= Work

1 0 2 <= 0 0 0 (F)

P1 -> Wait

Step 2:

For i=2

Finish[2] = F & Request[2] <= Work

0 0 0 <= 0 0 0 (T)

P2 -> Safe sequence

Step 3:

Work = Work + Allocation[2]

Work = 0 0 0 + 1 0 3 = 1 0 3

0	1	2	3
---	---	---	---

F	F	T	F
---	---	---	---

Problem : Deadlock Detection Algorithm

Step 2:

For i=3

Finish[3] = F & Request[3] <= Work
 $3 \ 3 \ 0 \leq 1 \ 0 \ 3 \text{ (F)}$

P3 -> Wait

Process	Allocation			Request		
	A	B	C	A	B	C
P0	1	0	2	0	0	1
P1	2	1	1	1	0	2
P2	1	0	3	0	0	0
P3	1	2	2	3	3	0

Step 2:

For i=0

Finish[0] = F & Request[0] <= Work
 $0 \ 0 \ 1 \leq 1 \ 0 \ 3 \text{ (T)}$

P0 -> Safe sequence

Step 3:

Work = Work + Allocation[0]

Work = $1 \ 0 \ 3 + 1 \ 0 \ 2 = 2 \ 0 \ 5$

0 1 2 3

Finish =

T	F	T	F
---	---	---	---

Step 2:

For i=1

Finish[1] = F & Request[1] <= Work
 $1 \ 0 \ 2 \leq 2 \ 0 \ 5 \text{ (T)}$

P1 -> Safe sequence

Step 3:

Work = Work + Allocation[1]

Work = $2 \ 0 \ 5 + 2 \ 1 \ 1 = 4 \ 1 \ 6$

0 1 2 3

Finish =

T	T	T	F
---	---	---	---

Problem : Deadlock Detection Algorithm

Step 2:

For i=3

Finish[3] = F & Request[3] <= Work

$$3 \ 3 \ 0 \leq 4 \ 1 \ 6 \ (\text{F})$$

P3 -> Wait

Process	Allocation			Request		
	A	B	C	A	B	C
P0	1	0	2	0	0	1
P1	2	1	1	1	0	2
P2	1	0	3	0	0	0
P3	1	2	2	3	3	0

The system is in unsafe state and Process P3 causes deadlock

Semaphore

Synchronization tool that provides more sophisticated ways (than Mutex locks) for process to synchronize their activities.

Semaphore **S** – integer variable

Can only be accessed via two indivisible (atomic) operations

- **wait()** and **signal()**

Definition of the **wait()** operation

```
wait(S) {  
    while (S <= 0)  
        ; // busy wait - block calling process  
    S--;  
}
```

Definition of the **signal()** operation

```
signal(S) {  
    S++;  
}
```

Problem : Semaphores

The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as $S_0 = 1$, $S_1 = 0$, $S_2 = 0$. Find out how many times Process P0 will print “0”. Assume the order of execution as P0, P1, P2, P0, P1.

Process P0:

```
while(true)
{
    wait(S0);
    printf( "0");
    signal(S1);
    signal(S2);
}
```

Process P1:

```
wait(S1);
signal(S0);
```

Process P2:

```
wait(S2);
signal(S0);
```

Problem : Semaphores

```
wait(S) {
    while (S <= 0)
        ; // busy wait
    S--;
}
```

```
signal(S) {
    S++;
}
```

Process P0:

```
while(true)
{
    wait(S0);
    printf( "0");
    signal(S1);
    signal(S2);
}
```

Process P1:

```
wait(S1);
signal(S0);
```

Process P2:

```
wait(S2);
signal(S0);
```

$$S0 = 1, S1 = 0, S2 = 0$$

Timeline	S0	S1	S2	Print
P0	0	1	1	0
P1	1	0		
P2	2		0	
P0	1	1	1	0
P1	2	0		

Problem : Semaphores

Consider two processes A and B. Two semaphore variables S and T are used to synchronize the processes. S is initialized to 0 and T is initialized to 1. Processes are scheduled in the following order: A B A B B A A B A

What will be printed on the screen?

Process A

```
While(1)
{
    wait(S);
    Print 'P';
    Print 'P';
    Signal(T);
}
```

Process B

```
While(1)
{
    wait(T);
    Print 'I';
    Print 'I';
    Signal(S);
}
```

Problem : Semaphores

```
wait(S)
{
    while (S <= 0)
        ; // busy wait
    S--;
}
```

```
signal(S)
{
    S++;
}
```

Initially: S = 0, T = 1

Timeline	S	T	Print
A			
B	1	0	
A	0	1	PP
B	1	0	
B			
A	0	1	PP
A			
B	1	0	
A	0	1	PP

Process A

```
While(1)
{
    wait(S);
    Print 'P';
    Print 'P';
    Signal(T);
}
```

Process B

```
While(1)
{
    wait(T);
    Print 'I';
    Print 'I';
    Signal(S);
}
```



Questions ?



BITS Pilani
Pilani Campus

Thank you.





COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS WEBINAR



BITS Pilani
Pilani Campus

Dr. Lucy J. Gudino

Agenda

-
- Replacement algorithms (Optimal and LRU)
 - Pipeline collision avoidance strategy

FIFO → ~~15~~
15

Problem 1: Replacement Algorithm

Given page reference string: 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3

Compare the number of page faults for FIFO, Optimal and LRU page replacement algorithm

7/12

Solution : Optimal → Replace page that will not be used for longest period of time

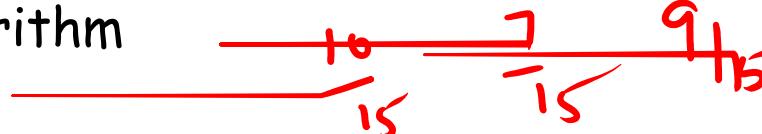
	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3
f_0	1	1	1	1	1	1	1	1	1	1	1	1	7	7	7
f_1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
f_2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
f_3				4	4	4	5	6	6	6	6	6	6	6	6

Problem 2: Replacement Algorithm

Given page reference string: 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2

Compare the number of page faults for FIFO, Optimal and LRU page replacement algorithm

Solution : LRU



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	1	2	3	4	2	1	5	6	2	1	2	3	7	6	2
f ₀	1 ₅	1 ₅	1 ₅	1 ₉	1 ₉	1 ₉	1 ₉	1 ₉	6 ₁₃	6 ₁₃					
f ₁	2 ₁	2 ₁	2 ₁	2 ₄	2 ₄	2 ₄	2 ₄	2 ₈	2 ₈	2 ₁₀	2 ₁₄				
f ₂	3 ₂	5 ₆	3 ₁₁	3 ₁₁	3 ₁₁	3 ₁₁	3 ₁₄								
f ₃		4 ₃	4 ₃	4 ₃	4 ₃	6 ₇	7 ₁₂	7 ₁₂	7 ₁₂	7 ₁₂	H				

Problem 1

Consider the following reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

For Optimal and LRU replacement algorithm with number of frames = 2, 3, 4, check whether it exhibits Belady's Anomaly or not.

Belady's anomaly : more frames \Rightarrow more page faults

$\pi = 2$

Optimal

$9/12$

1	2	3	4	1	2	5	1	2	3	4	5
f_0	1	1	1	1	1	1	1	2	3	4	4
f_1	2	3	4	4	2	5	5	5	5	5	5

M M M M H M M H M M M H

3

$7/12$

1	2	3	4	1	2	5	1	2	3	4	5
f_0	1	1	1	1	1	1	1	1	3	4	4
f_1	2	2	2	2	2	2	2	2	2	2	2
f_2		3	4	4	4	5		5	5	5	5

M M M M H M M H M M M H

$\frac{6}{12}$

1	2	3	4	1	2	5	1	2	3	4	5
f_0											
f_1											
f_2											
f_3											

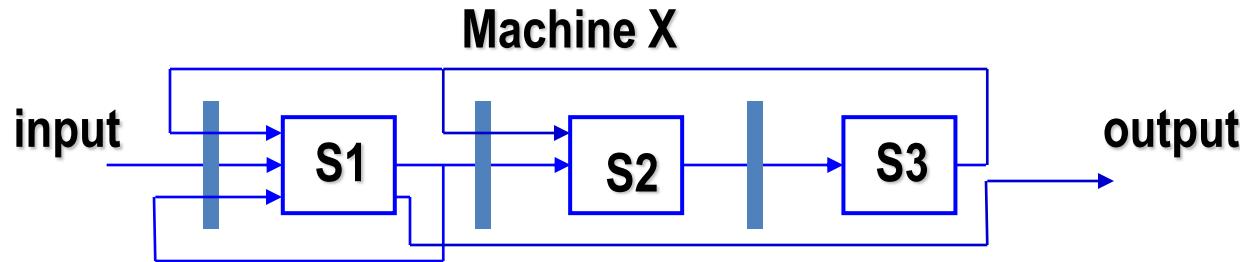
M M M M H H H H H H H H

LRU

Pipeline

- Linear vs Nonlinear pipelines
- Reservation Table.

Reservation Table

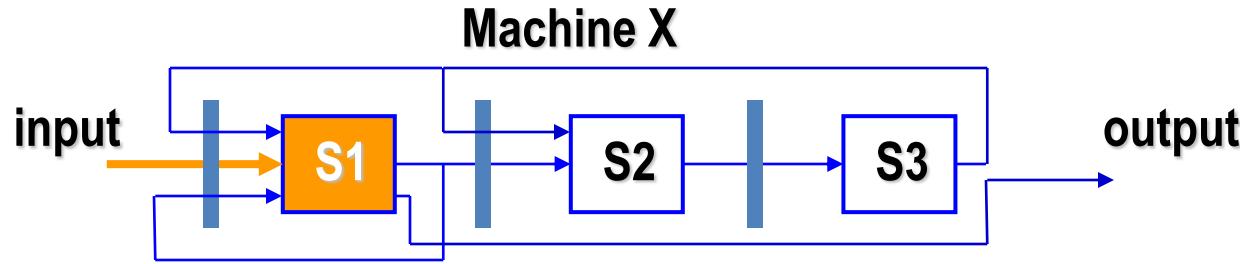


Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Reservation Table



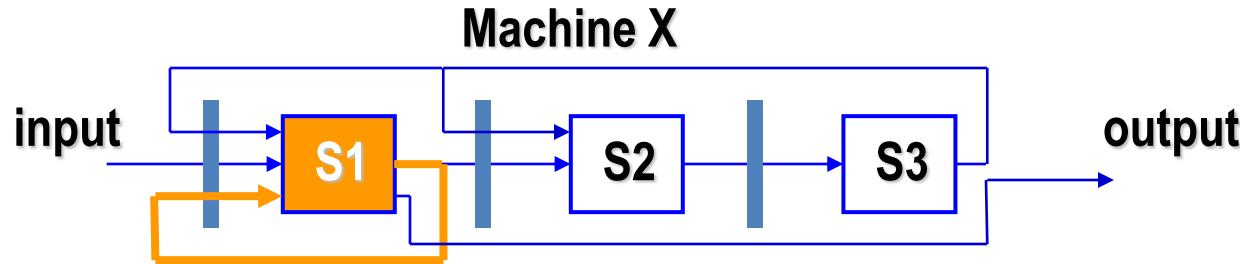
Reservation Table

Time →

Stage ↓

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Reservation Table

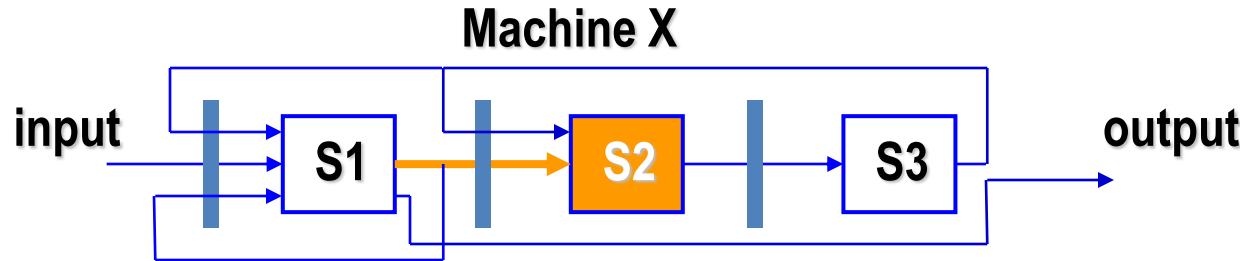


Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Reservation Table



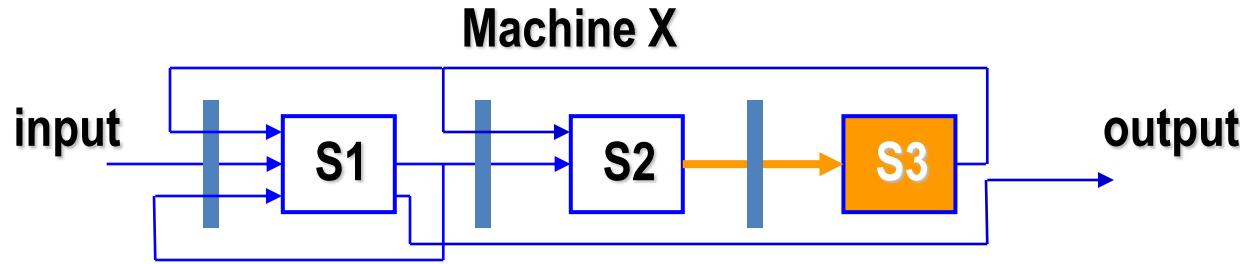
Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Stage ↓

Reservation Table

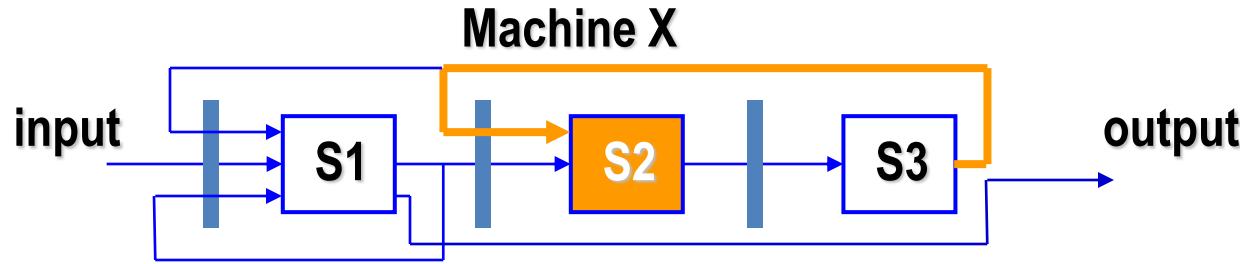


Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X			X	

Reservation Table

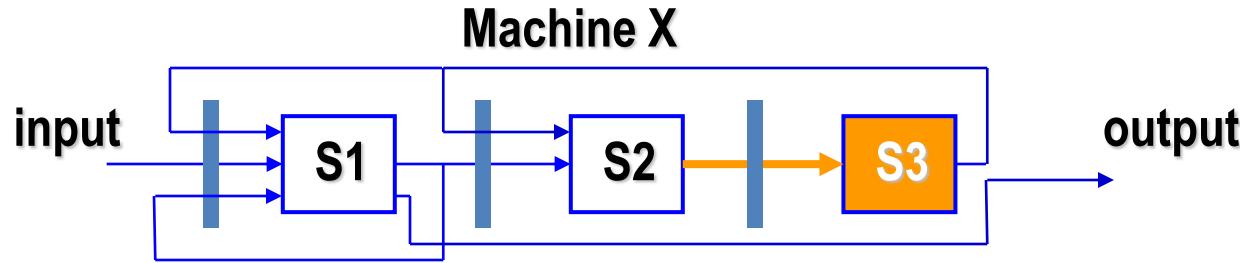


Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Reservation Table

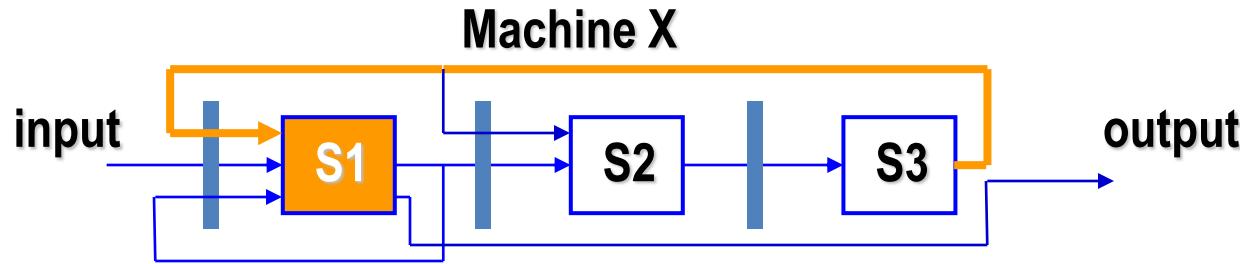


Reservation Table

Time →

	0	1	2	3	4	5	6	7
Stage ↓	X	X					X	X
S1								
S2			X		X			
S3				X		X		

Reservation Table

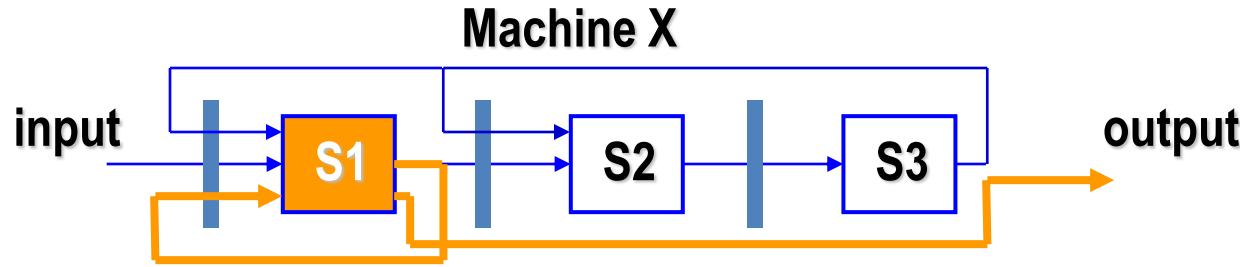


Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Reservation Table



Reservation Table

Time →

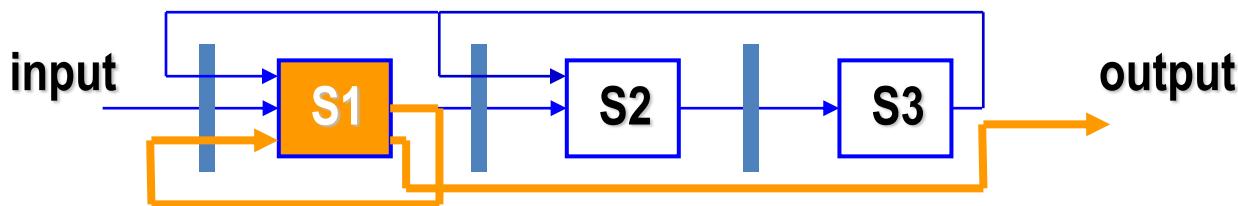
Stage ↓

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Reservation Table

X Y

Machine X



Reservation Table

Time →

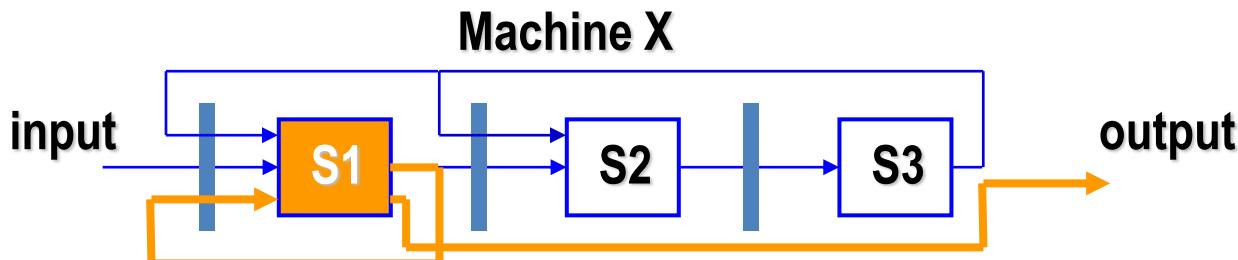
	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
X	X					X	X												
		X		X															
			X		X														

Stage →

Reservation Table



Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2				X		X		
S3					X		X	

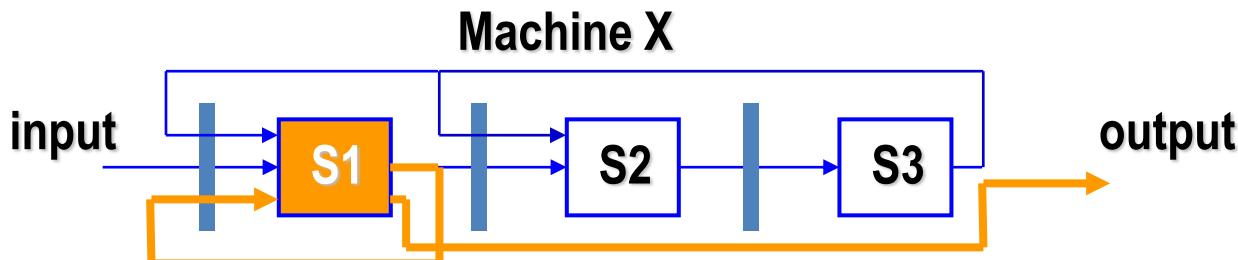


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
X	X	Y	Y			X	X												
		X		X Y															
			X		X														

Stage →

Stage →

Reservation Table



Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2				X		X		
S3					X		X	



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
X	X		Y	Y		X	X		Y	Y									
		X		X	Y		Y												
			X		X	Y		Y											

Stage →

Stage →



BITS Pilani
Pilani Campus

Job Sequencing and Collision Prevention

Terms and definitions

- Collision: when two or more initiations attempt to use the same stage at the same time
- Initiation : Launching of an operation
- Latency: The number of cycles that elapse between two initiations or number of time units between two initiations
- Latency Sequence: The sequence of latencies between successive initiations
- Latency cycle: A latency sequence that repeats itself
- Procedure to choose a latency sequence is called Control Strategy

Collision

In a static pipeline, all initiations are characterized by the same reservation table

In a dynamic pipeline successive initiations may be characterized by a set of reservation tables

Reservation Table

	0	1	2	3	4	5	6	7	8
S1	X								X
S2		X	X					X	
S3				X					
S4					X	X			
S5							X	X	

Collision Control

A collision occurs when two tasks are initiated with a latency equal to the column distance between two X's on some row of the reservation table



	0	1	2	3	4	5	6	7	8	Latency
S1	X								X	8
S2		X	X					X		1,6,5
S3				X						-
S4					X	X				1
S5							X	X		1

Forbidden set of latencies

- The set of column distances $F = \{L_1, L_2, \dots, L_r\}$, between all possible pairs of X's on each row of reservation table
- Forbidden set contains all possible latencies that cause collision between two initiations

Collision

$$F = \{1, 5, 6, 8\}$$

	0	1	2	3	4	5	6	7	8	Latency
S1	X								X	8
S2		X	X					X		1,6,5
S3				X						-
S4					X	X				1
S5							X	X		1

Collision Vector

- Is a binary vector
- $C = (C_n, \dots, C_2, C_1)$
where $C_i = 1$ if $i \in F$

Collision Vector



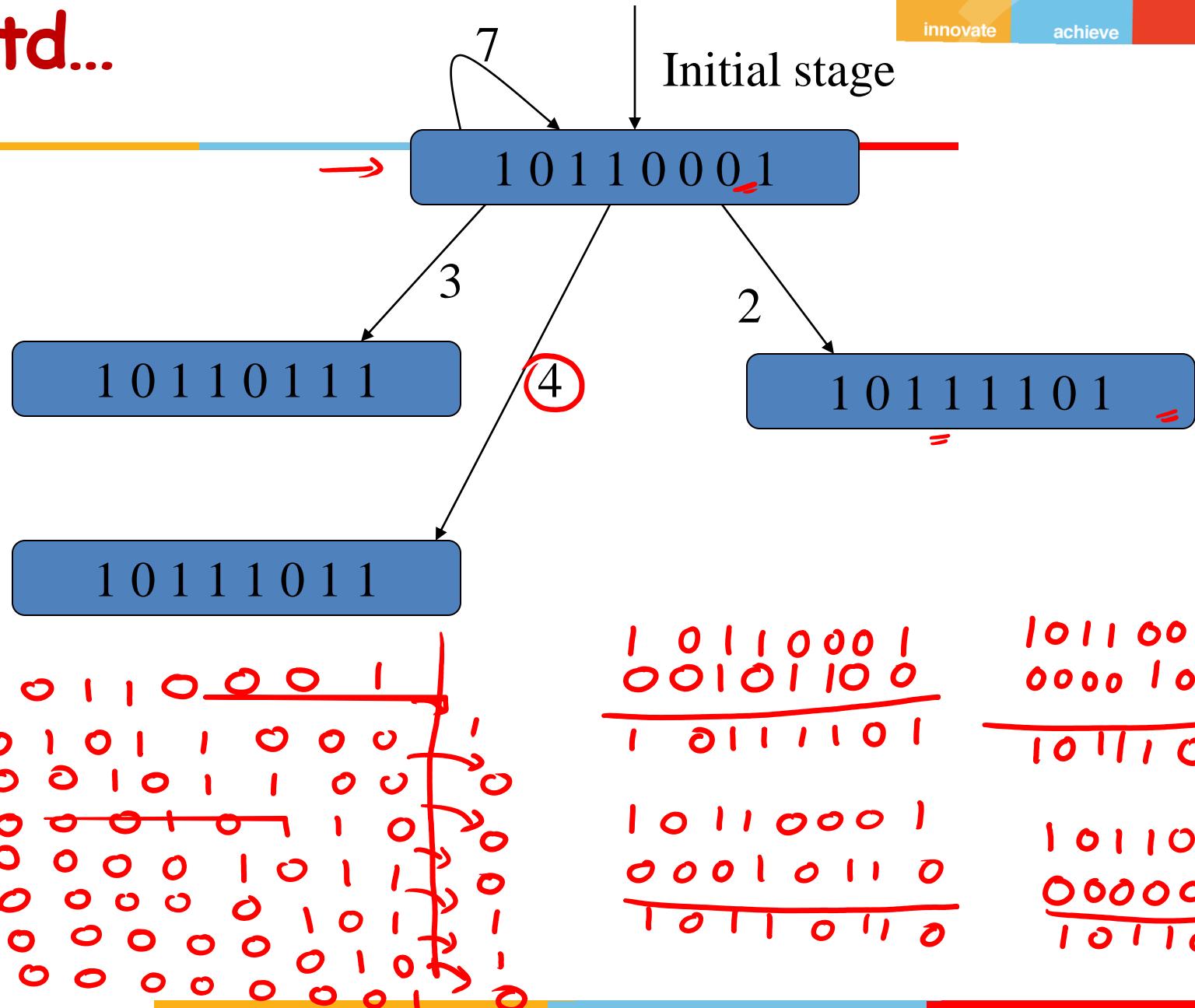
	0	1	2	3	4	5	6	7	8
S1	X								
S2		X	X					X	
S3				X					
S4					X	X			
S5							X	X	

✓ $F = \{1, 5, 6, 8\}$ $(3, 4)$

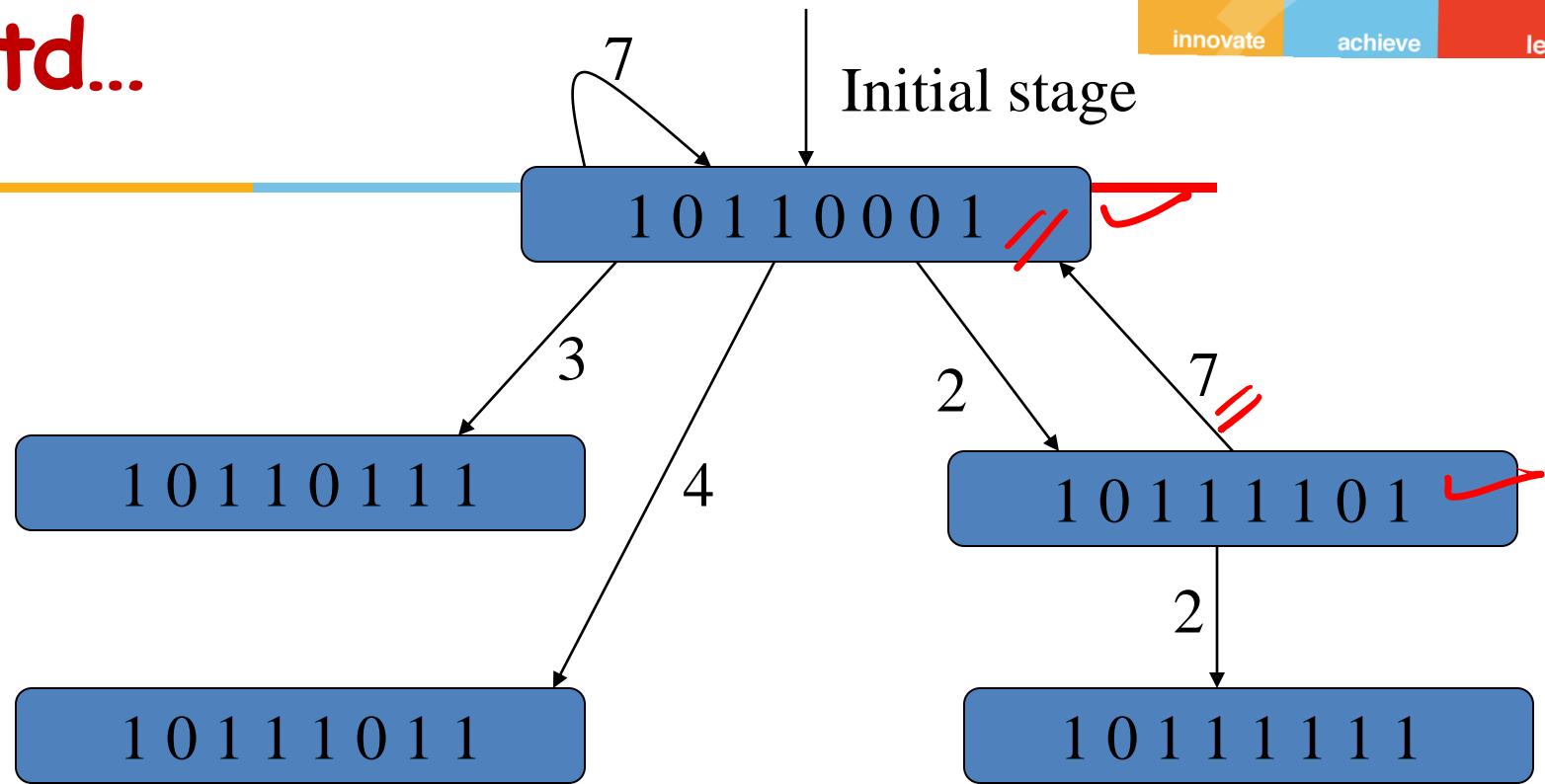
$C = (C_8 \ C_7 \ C_6 \ C_5 \ C_4 \ C_3 \ C_2 \ C_1)$

$C = | \ 0 \ | \ | \ 0 \ 0 \ 0 \ |$

Contd...



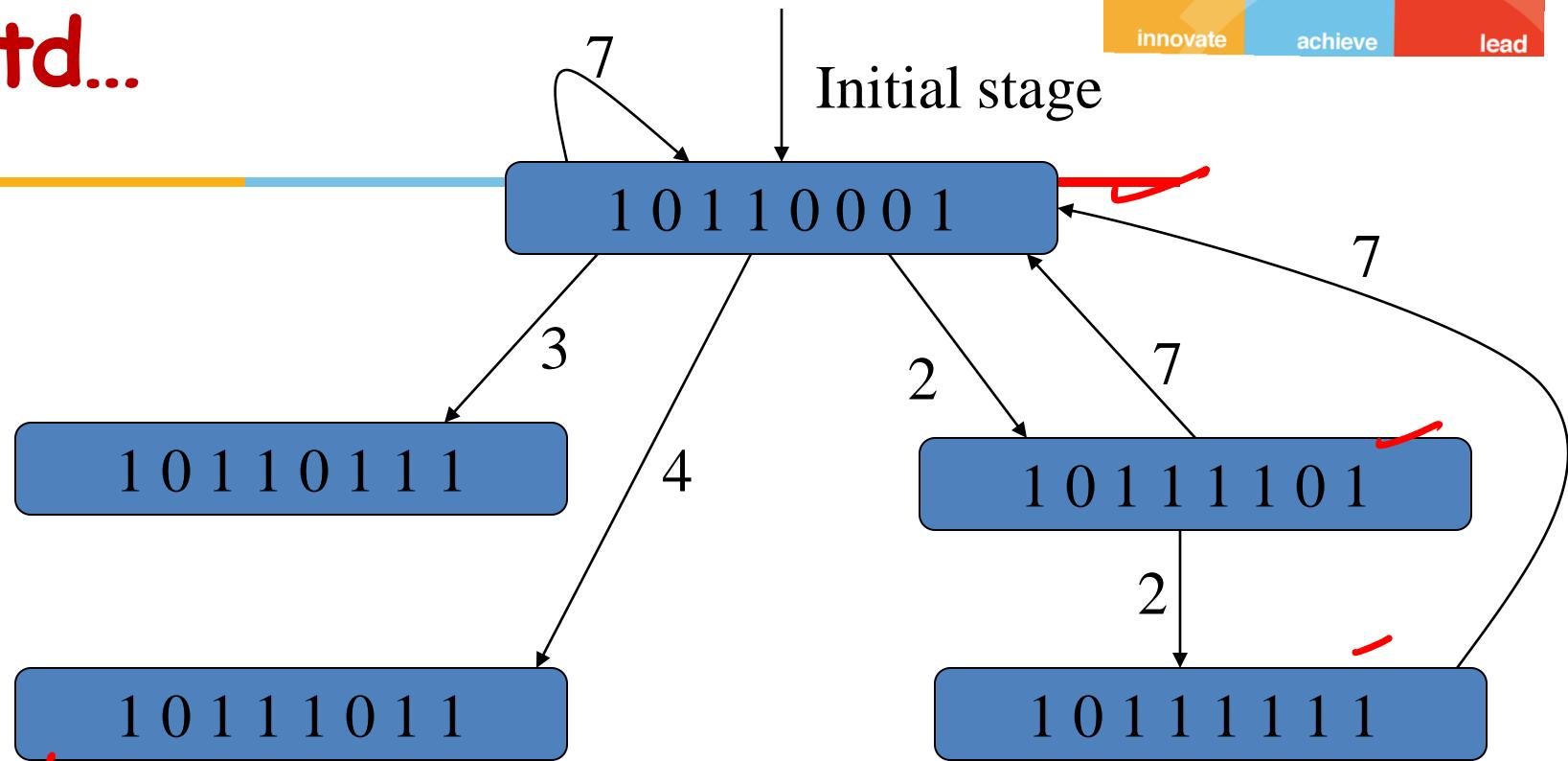
Contd...



$t=1$ 10111101
 $t=2$ 01011110
 $t=3$ 00101111
 $t=4$ 00010111
 $t=5$ 00001011
 $t=6$ 00000101
 $t=7$ 00000010

$\begin{array}{r} 10110001 \\ 00101111 \\ \hline 10111111 \end{array}$
 $\begin{array}{r} 10110001 \\ 00000001 \\ \hline 10110001 \end{array}$

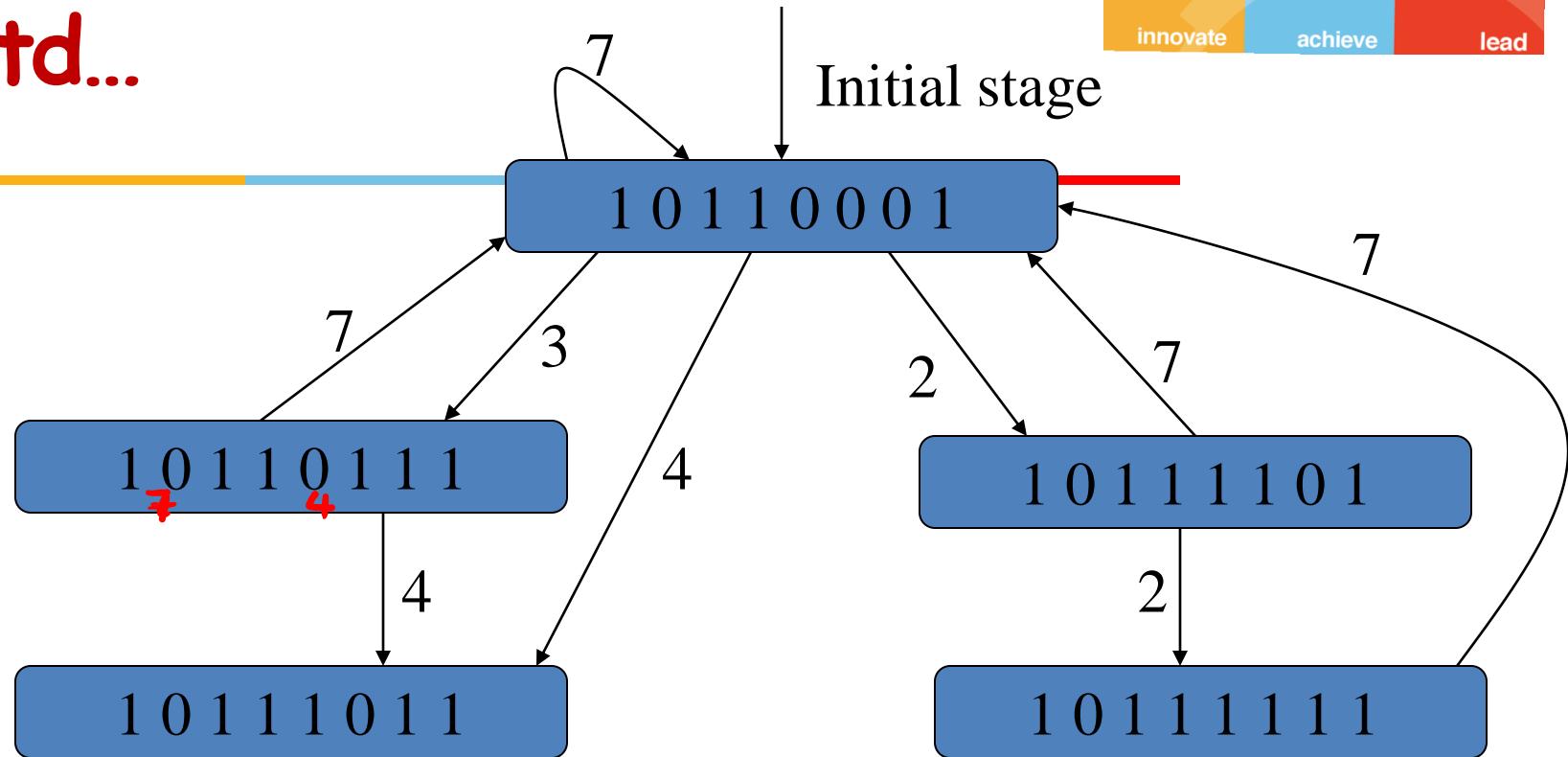
Contd...



$t=0$ 10111111
 $t=6$ 00000001 → 1
 $t=7$ 00000001 → 0
 $t=8$ 00000000

$$\begin{array}{r}
 10110001 \\
 00000001 \\
 \hline
 10110001
 \end{array}$$

Contd...

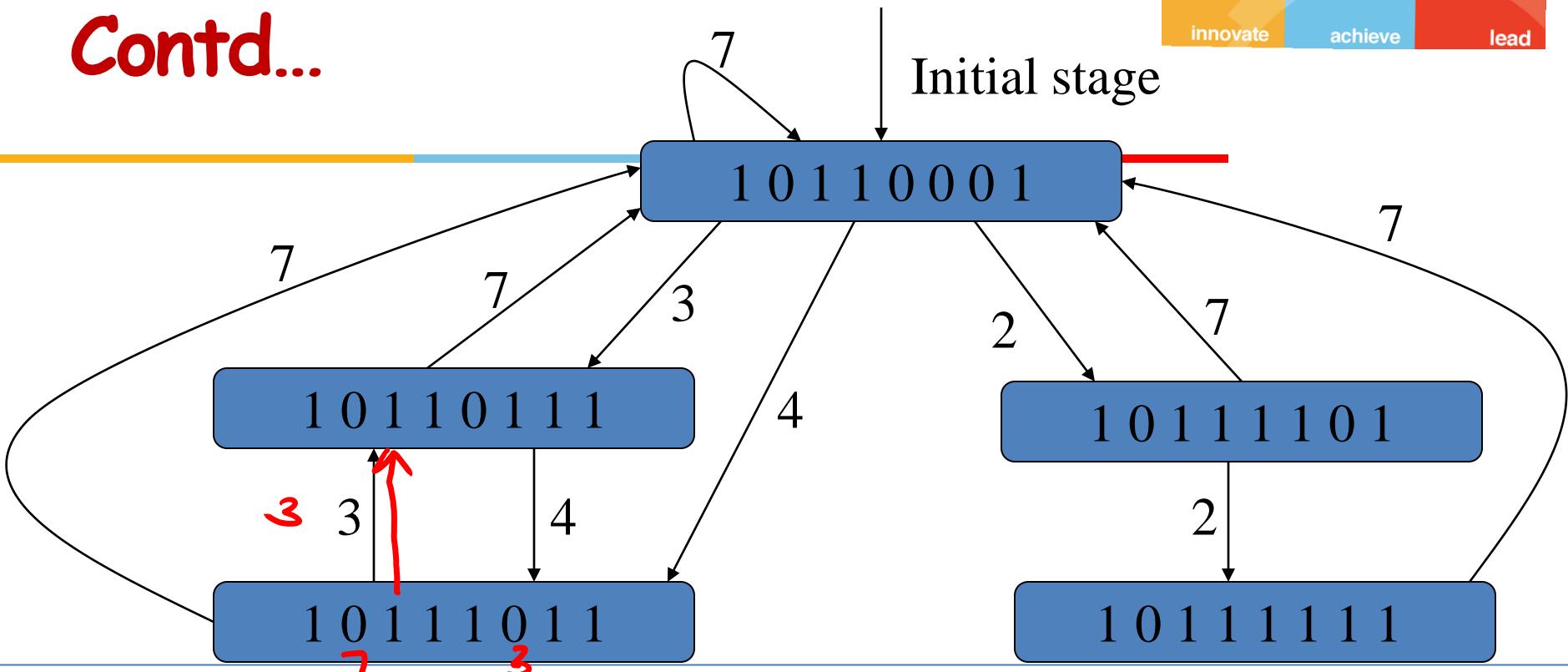


$$\begin{array}{r}
 t=4 \quad 10110111 \\
 00001011 \\
 \hline
 t=7 \quad 00000001
 \end{array}$$

$$\begin{array}{r}
 10110001 \\
 00001011 \\
 \hline
 10111011
 \end{array}$$

$$\begin{array}{r}
 10110001 \\
 00000001 \\
 \hline
 10110001
 \end{array}$$

Contd...



$t=3$

$$\begin{array}{r}
 10111011 \\
 00010111 \rightarrow 0 \\
 00000001 \rightarrow 0
 \end{array}$$

$t=7$

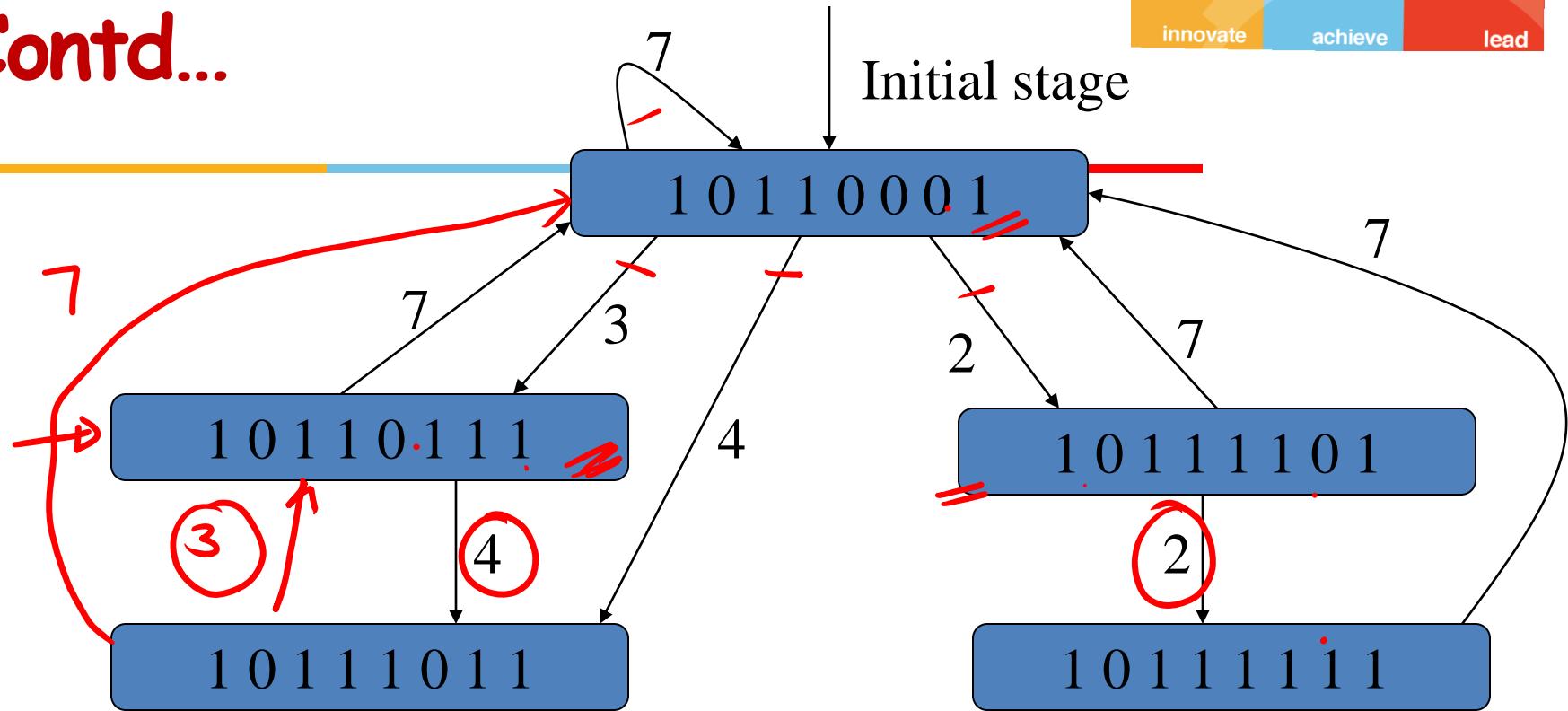
$$\begin{array}{r}
 10110001 \\
 00010111 \\
 \hline
 10110111
 \end{array}$$

$$\begin{array}{r}
 10110001 \\
 00000001 \\
 \hline
 00110001
 \end{array}$$

Important Terms

- Average latency of a latency cycle is obtained by dividing the sum of all latencies by the number of latencies along the cycle
- Constant cycle: is a latency cycle which contains only one latency value
- Simple Cycle: is a latency cycle in which each state appears only once
- Greedy Cycle: whose edges are all made with minimum latencies from their respective starting states
- Minimum average latency
- Throughput = inverse of MAL

Contd...



$$\frac{\text{Simple cycle:}}{(2,7)} = 4.5$$

$$(2,2,7) = 3.6$$

$$(4,3,7) = 4.6$$

$$\begin{aligned} (3,4) &= 3.5 \\ (3,7) &= 5 \end{aligned}$$

$$(7) = 7$$

$$\begin{aligned} (4,7) &= 5.5 \\ (4,3,7) &= 4.6 \end{aligned}$$

$$\begin{aligned} (2,1,7) &\Rightarrow 3.6 \\ (3,4) &\Rightarrow 3.5 \end{aligned}$$

Constant cycle - {1}

Greedy Cycle

$$(2,2,7)$$

$$(4,3)$$

$$(3,4)$$

Contd...

