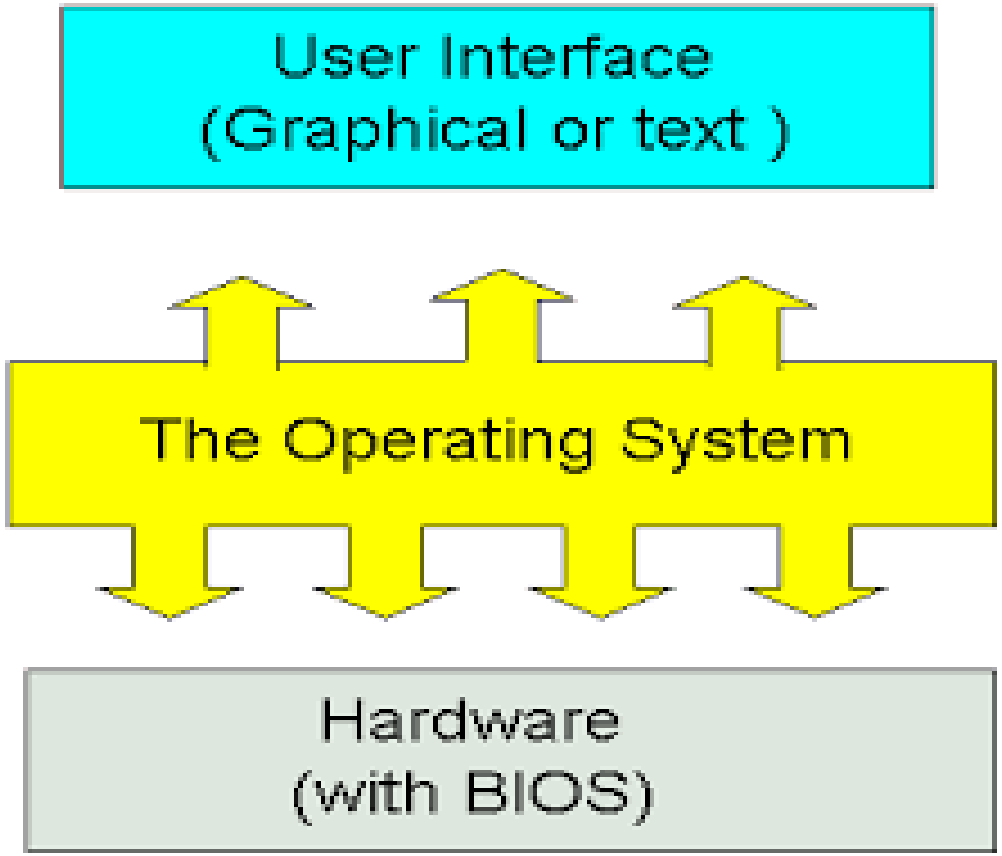
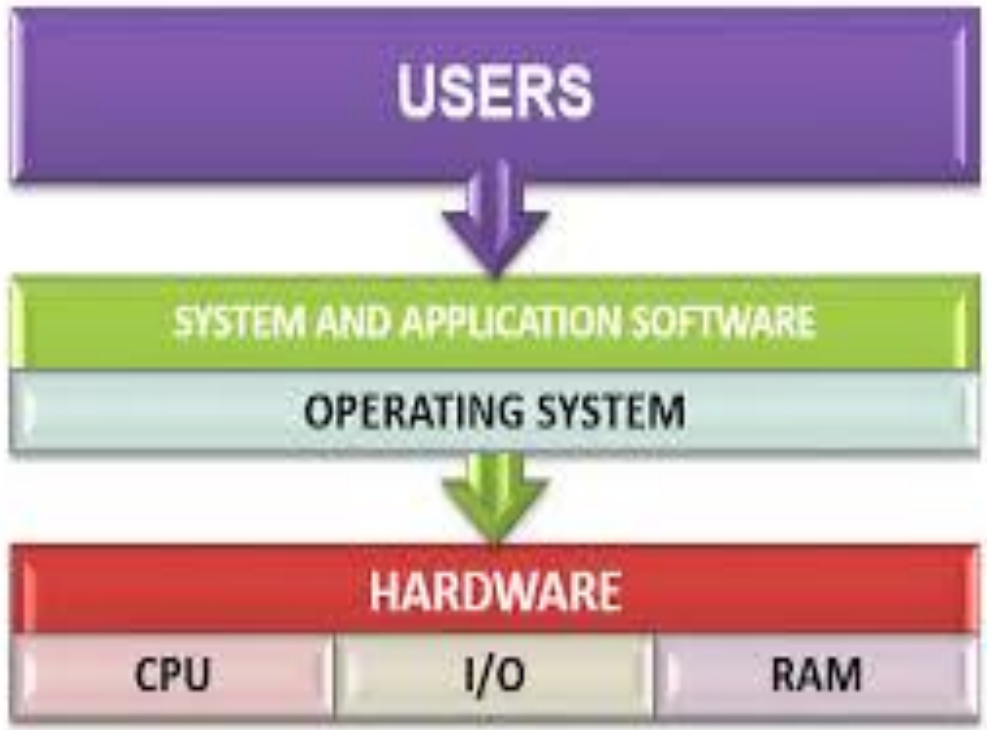
UNIT – I
OVERVIEW OF OPERATING SYSTEM

- Operating System Objectives and Functions
- The Evolution of Operating Systems
- Developments Leading to Modern Operating Systems
- Virtual Machines
- BASH Shell scripting: Basic shell commands, shell as a scripting language

Operating System…? ..Interface…



OS as a Interface [W1]



OS As a Interface [W2]

Operating System Objectives and Functions

Objective

Use the computer hardware in an efficient manner.

Functions

- Execute user programs and make solving user problems easier.
- Make the computer system convenient to use

Functions of Operating Systems

- Memory Management
- Device Management
- Processor Management
- File Management
- Controls System Performance
- Security
- Error Detection
- Coordination among Software and Users
- Job accounting

Services offered by Operating Systems

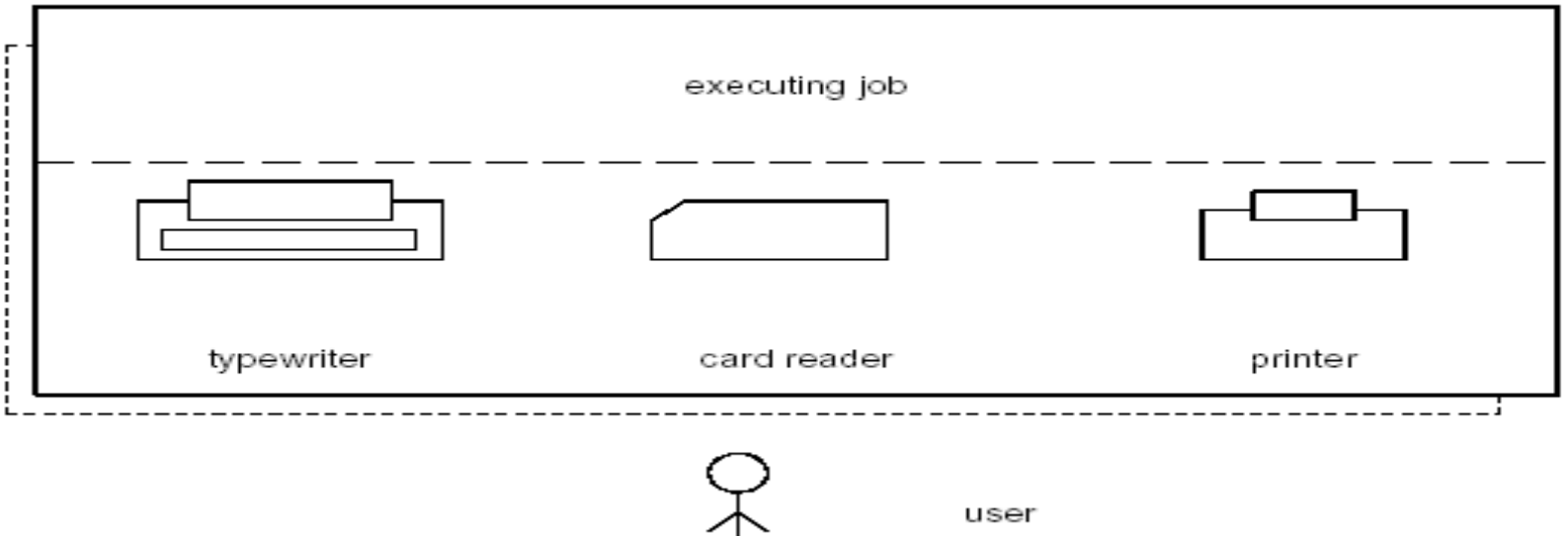
User Interface
Program Execution
File system manipulation
Input / Output Operations
Communication
Resource Allocation
Error Detection
Accounting
Security and protection

Evolution of Operating Systems

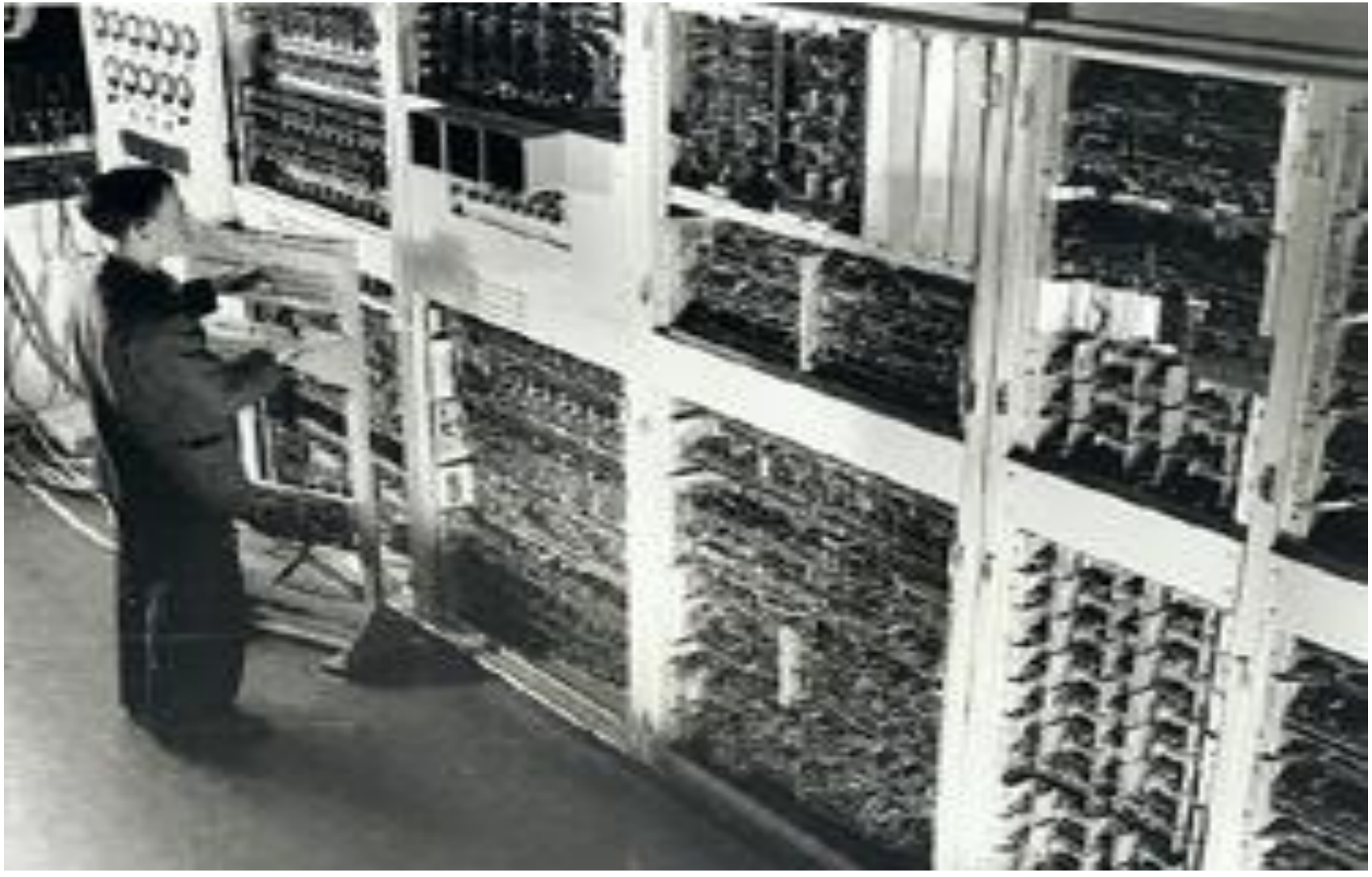
- Early Systems (1950)
- Simple Batch Systems (1960)
- Multiprogrammed Batch Systems (1970)
- Time-Sharing and Real-Time Systems (1970)
- Personal/Desktop Computers (1980)
- Multiprocessor Systems (1980)
- Networked/Distributed Systems (1980)
- Web-based Systems (1990)

Early Systems

- Structure
  - Single user system.
  - Programmer/User as operator (Open Shop).
  - Large machines run from console.
  - Paper Tape or Punched cards.



Example of an early computer system[T1]



Serial Processing

Earliest Computers:

- No operating system
  - programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to the computer in “series”

Problems:

- Scheduling:
  - most installations used a hardcopy sign-up sheet to reserve computer time
    - time allocations could run short or long, resulting in wasted computer time
- Setup time
  - a considerable amount of time was spent just on setting up the program to run

Simple Batch Systems

- Early computers were very expensive
  - important to maximize processor utilization
- Monitor
  - user no longer has direct access to processor
  - job is submitted to computer operator who batches them together and places them on an input device
  - program branches back to the monitor when finished

Characteristics of Early Systems

- Early software: Assemblers, Libraries of common subroutines (I/O, Floating-point), Device Drivers, Compilers, Linkers.
- Need significant amount of setup time.
- Extremely slow I/O devices.
- Very low CPU utilization.
- But computer was very secure.

Simple Batch Systems

- Use of high-level languages, magnetic tapes.
- Jobs are batched together by type of languages.
- An operator was hired to perform the repetitive tasks of loading jobs, starting the computer, and collecting the output (Operator-driven Shop).
- It was not feasible for users to inspect memory or patch programs directly.

Operation of Simple Batch Systems

- The user submits a job (written on cards or tape) to a computer operator.
- The computer operator place a batch of several jobs on an input device.
- A special program, the monitor, manages the execution of each program in the batch.
- Monitor utilities are loaded when needed.
- “Resident monitor” is always in main memory and available for execution.

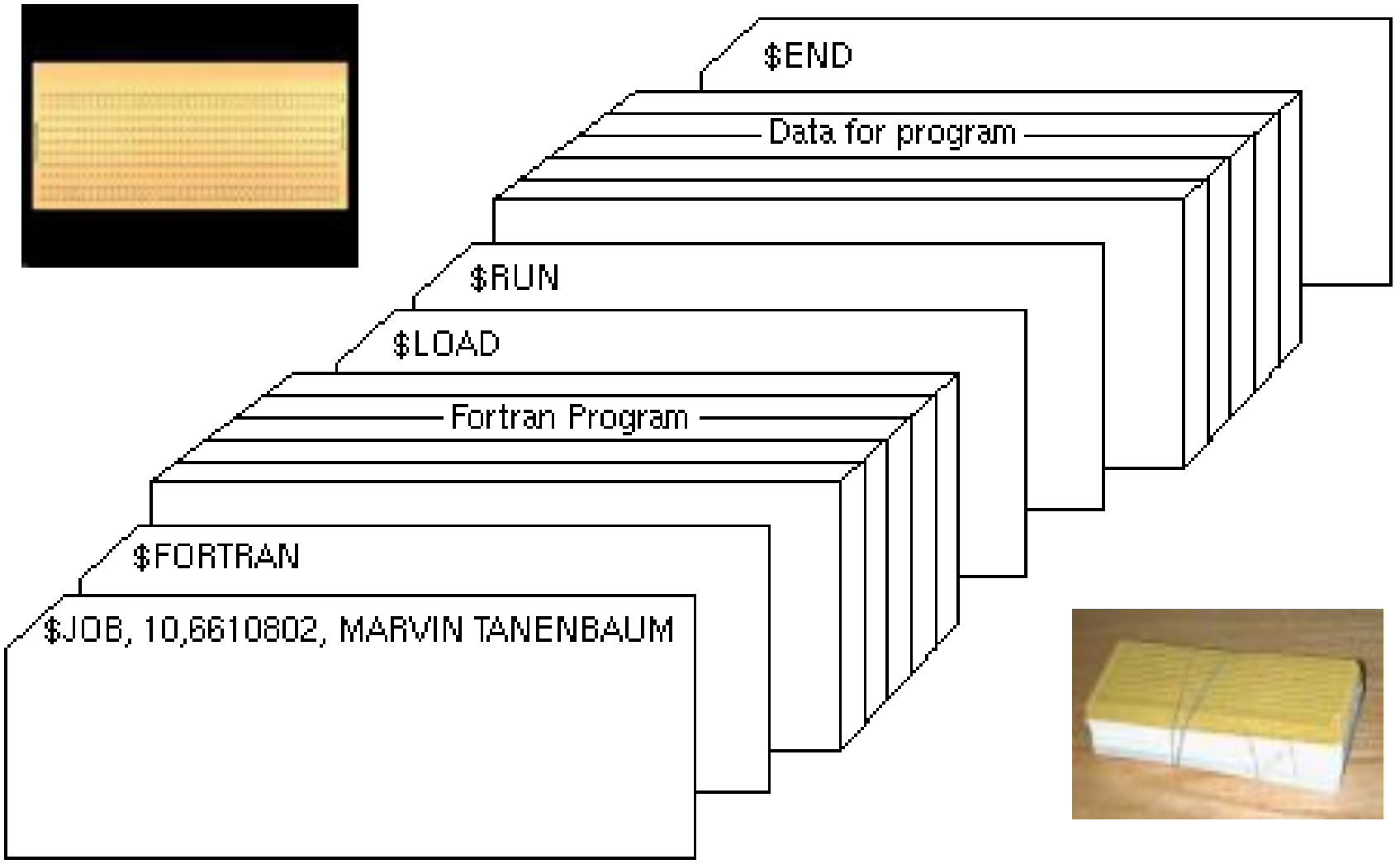
Idea of Simple Batch Systems

- Reduce setup time by batching similar jobs.
- Alternate execution between user program and the monitor program.
- Rely on available hardware to effectively alternate execution from various parts of memory.
- Use Automatic Job Sequencing – automatically transfer control from one job when it finishes to another one.

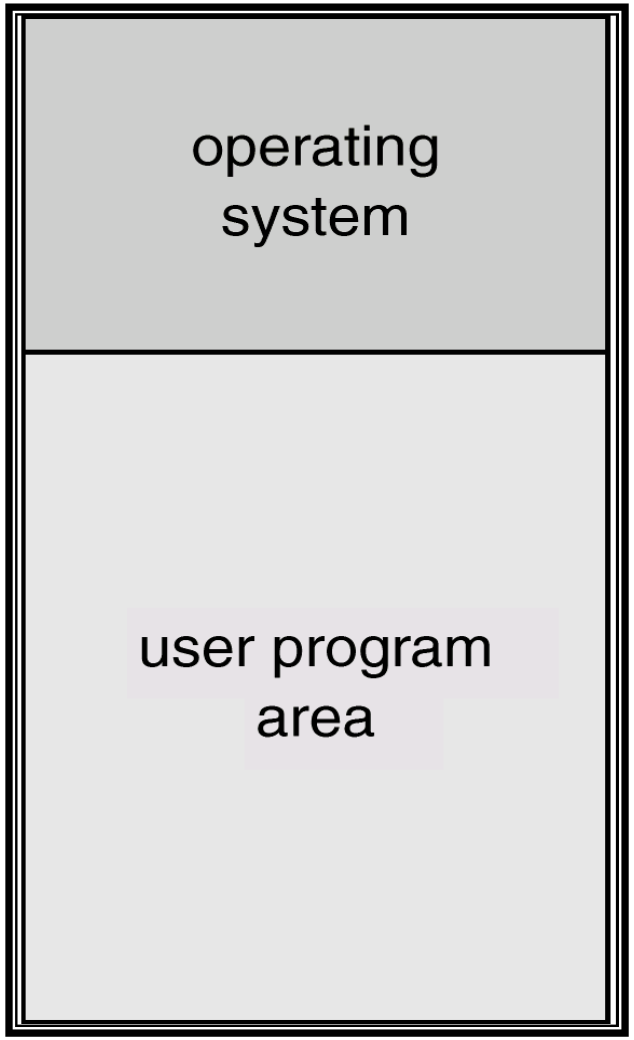
Control Cards

- Problems:
  - 1. How does the monitor know about the nature of the job (e.g., Fortran versus Assembly) or which program to execute?
  - 2. How does the monitor distinguish:
    (a) job from job?
    (b) data from program?
- Solution: Introduce Job Control Language (JCL) and control cards.

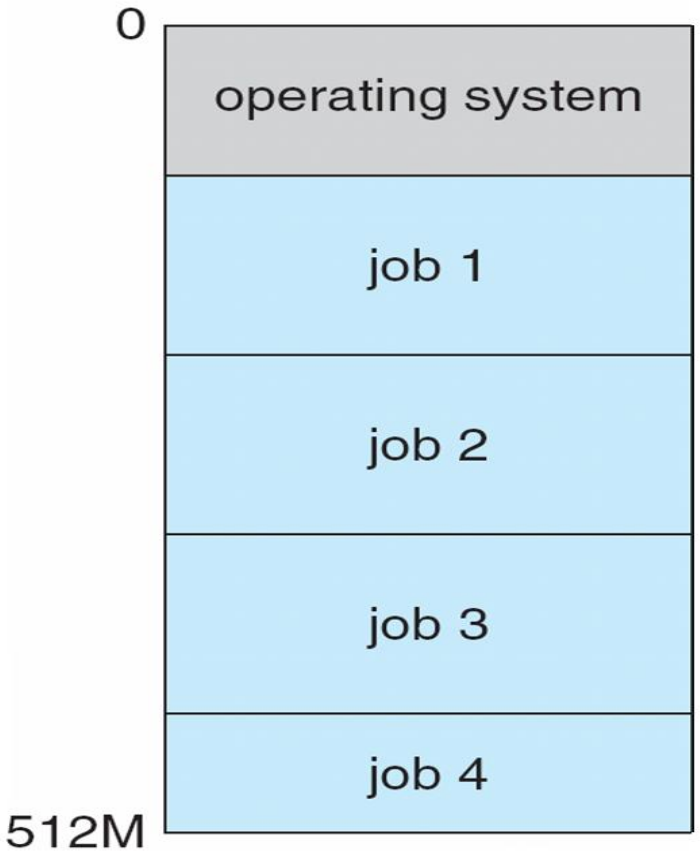
Example card deck of a Job



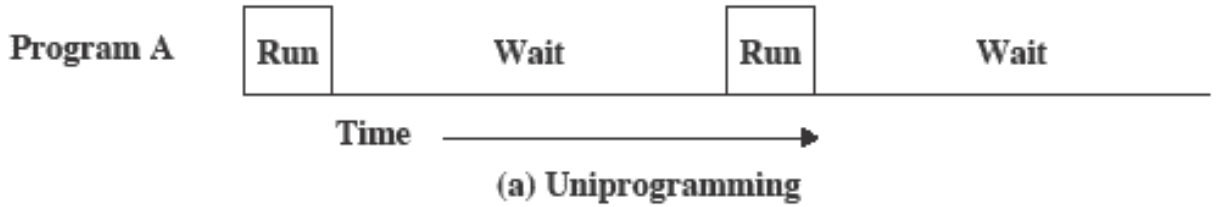
Memory Layout for Uniprogramming



Memory Layout for Batch Multiprogramming



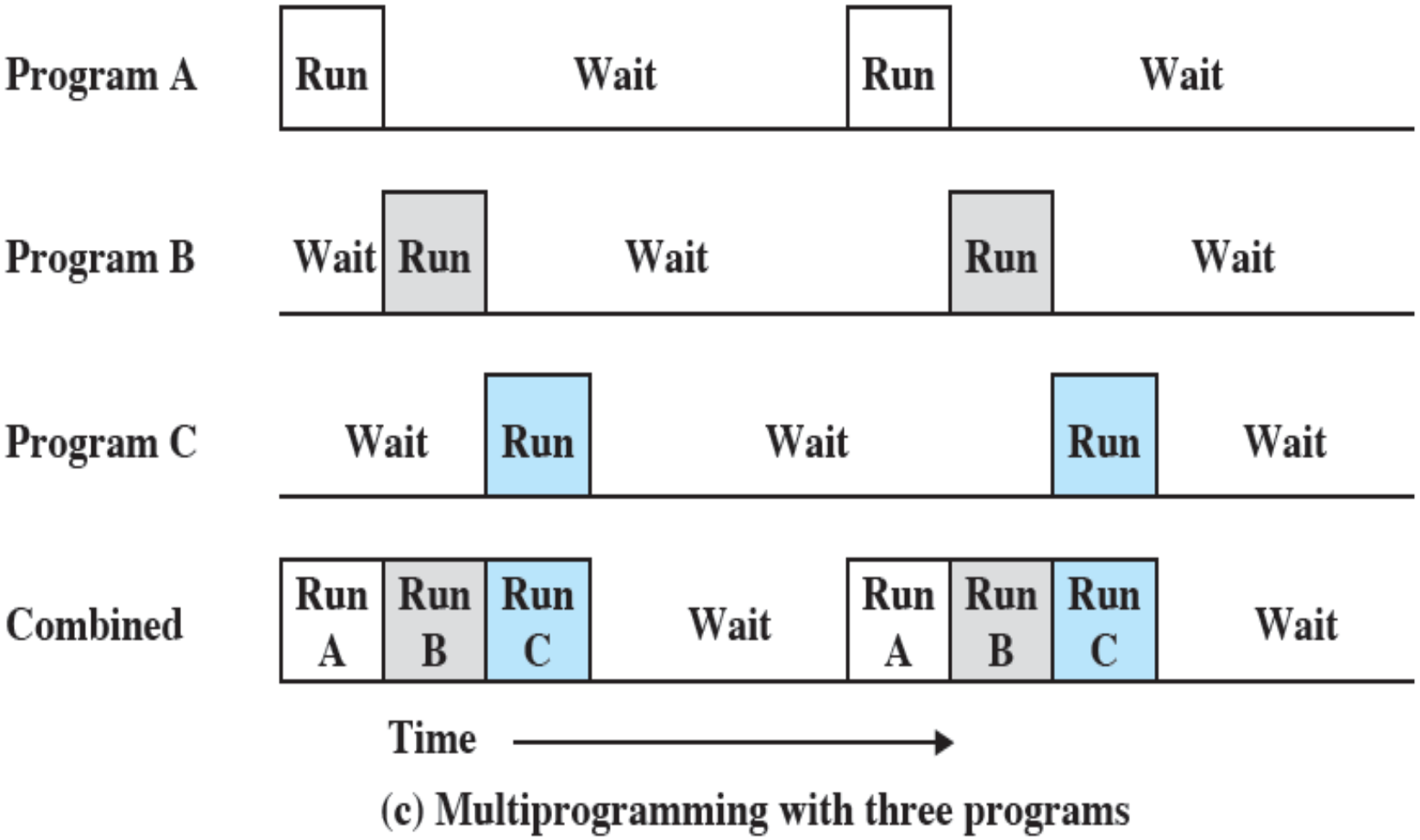
Uniprogramming



(a) Uniprogramming

- The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding

Multiprogramming

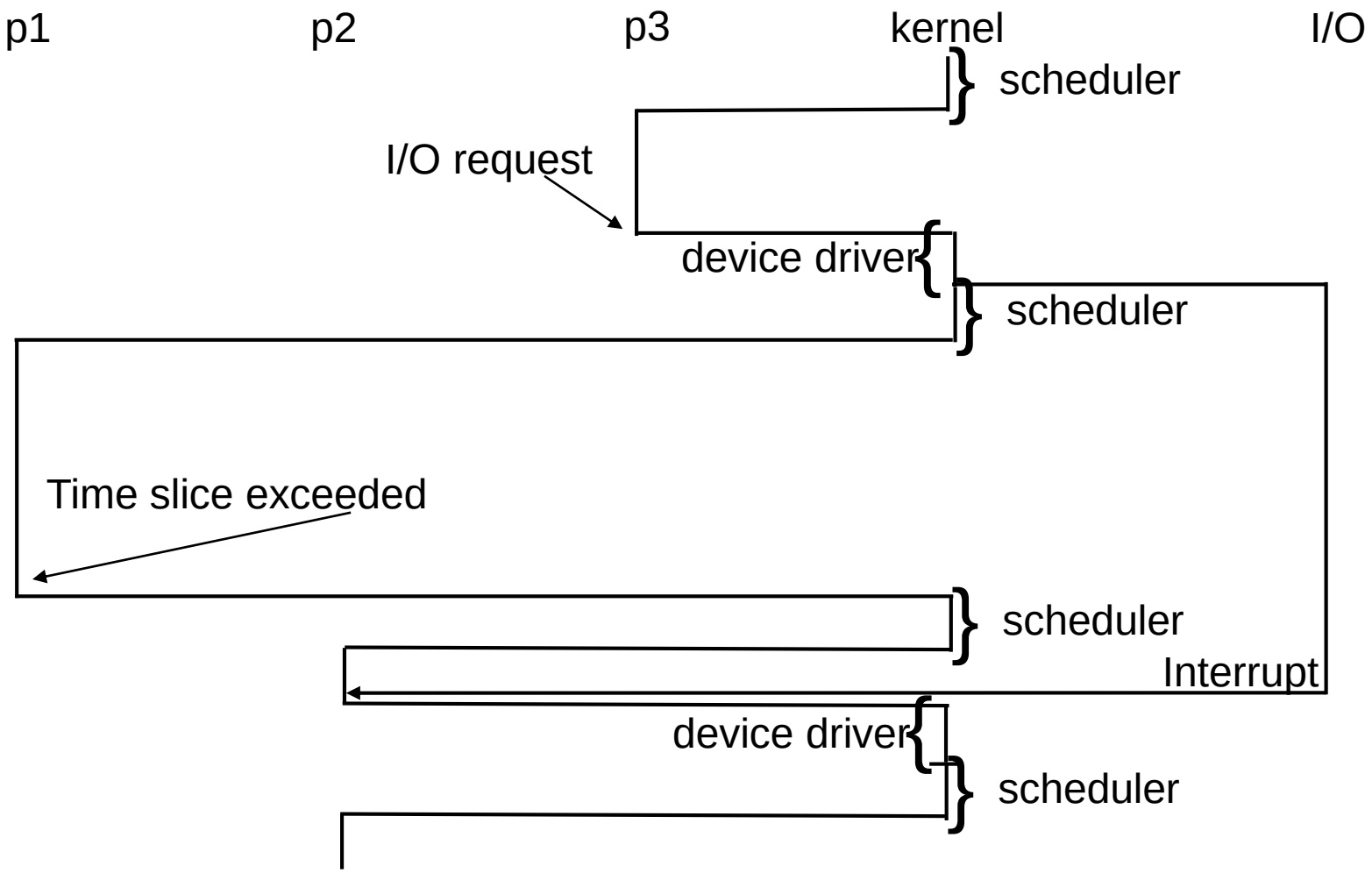


(c) Multiprogramming with three programs

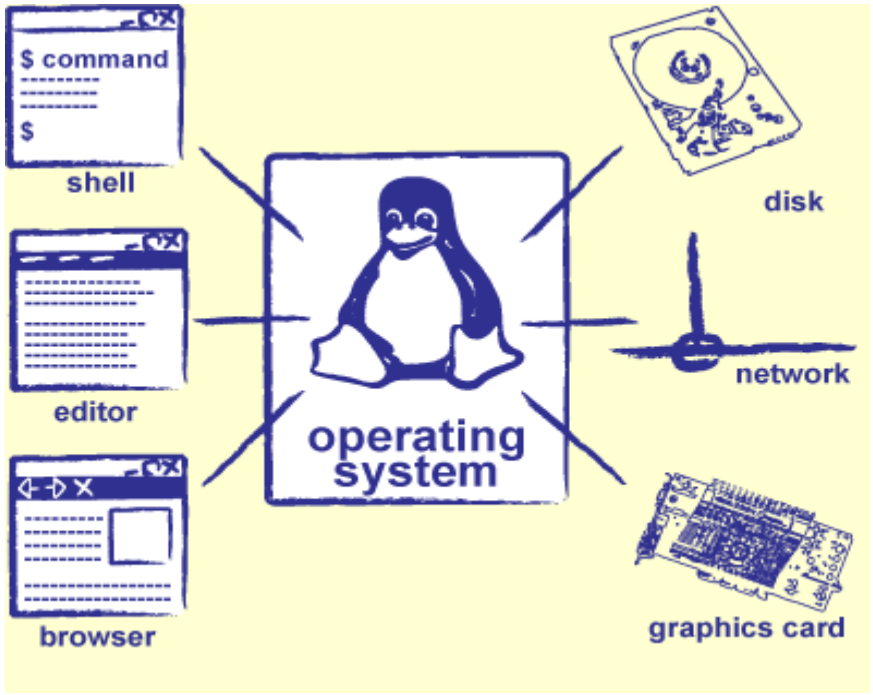
Why Multiprogramming?

- Multiprogramming (also known as Multitasking) needed for efficiency:
  - Single user cannot keep CPU and I/O devices busy at all times.
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute.
  - A subset of total jobs in system is kept in memory.
  - One job selected and run via job scheduling.
  - When it has to wait (for I/O for example), OS switches to another job.

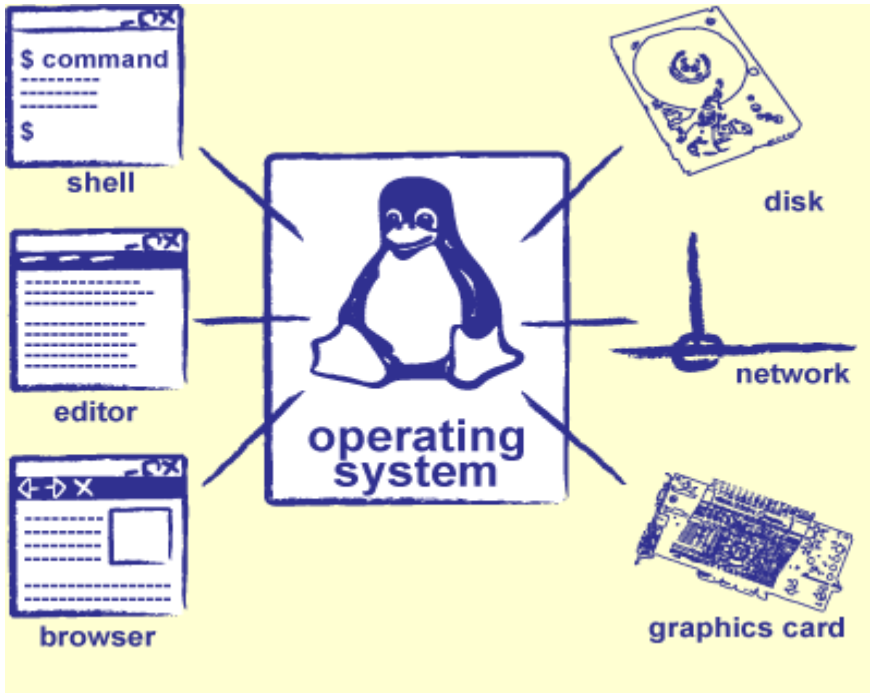
Example of Multiprogramming



Time Sharing



Time Sharing [W3]

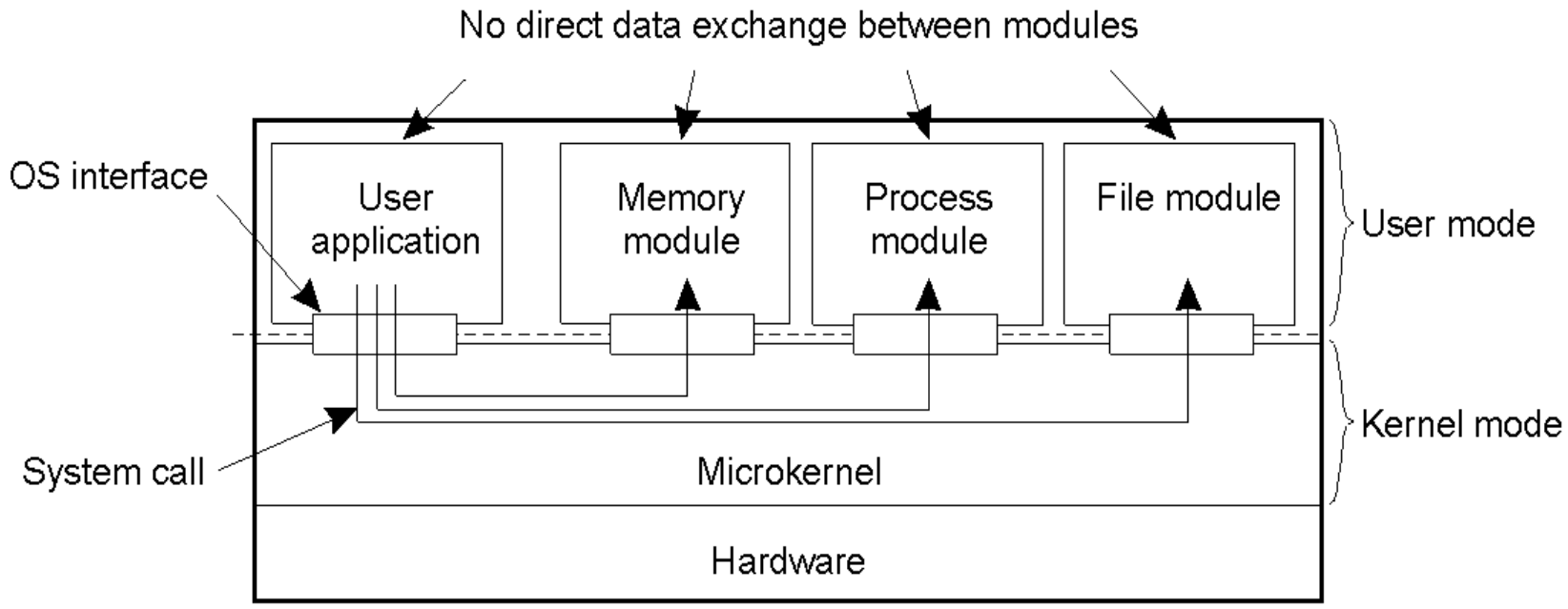


- Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time.
- Time-sharing or multitasking is a logical extension of multiprogramming.
- Processor's time which is shared among multiple users simultaneously is termed as time-sharing.
- The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the *objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.*

- Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently.
- Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if **n** users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

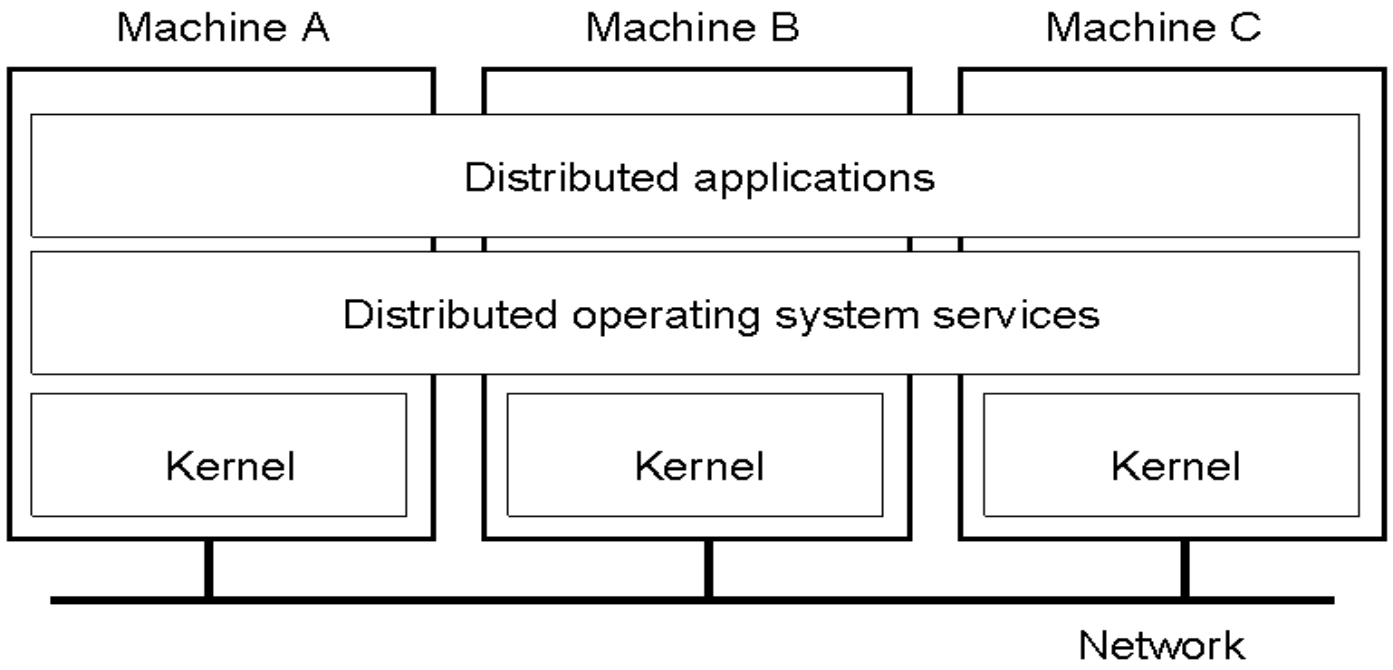
- Difference between Multiprogramming, multitasking, multithreading and multiprocessing
- **Multiprogramming –** A computer running more than one program at a time (like running Excel and Firefox simultaneously).
- **Multiprocessing –** A computer using more than one CPU at a time.
- **Multitasking –** Tasks sharing a common resource (like 1 CPU).
- **Multithreading** is an extension of multitasking.

Uniprocessor Operating System



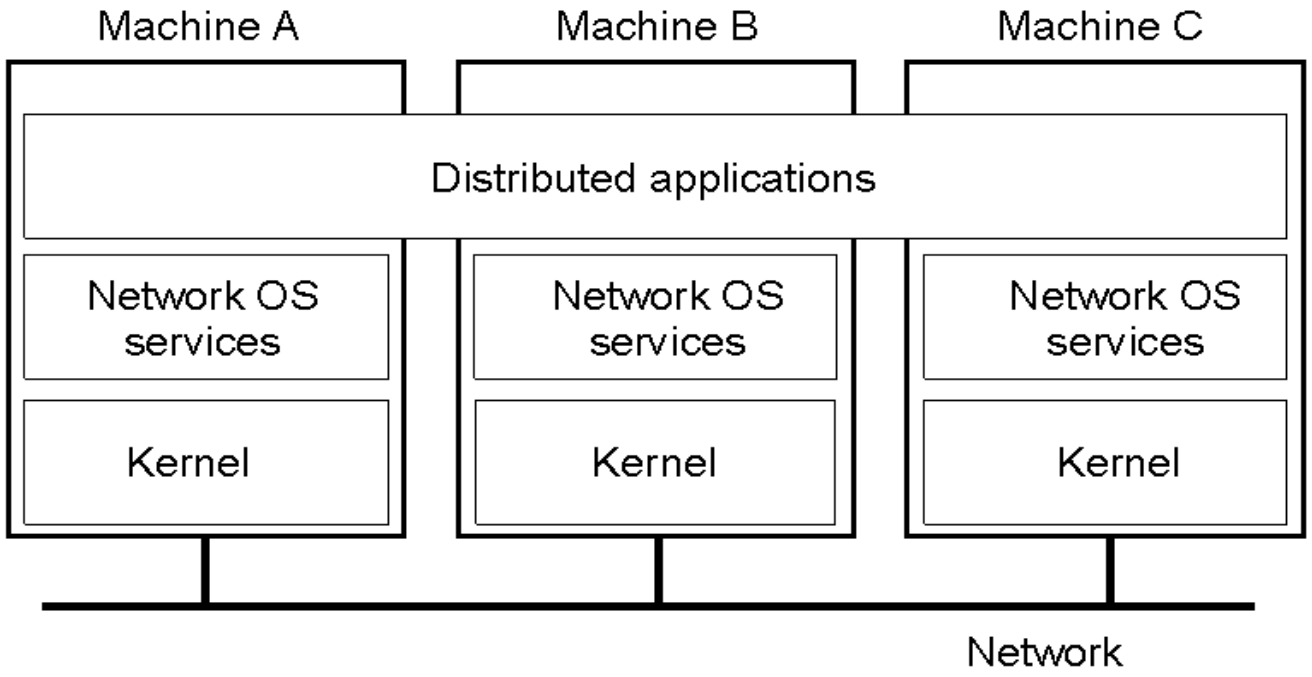
Separating applications from operating system code through a microkernel.

Distributed Operating System



Tightly-coupled operating system for multi-processors and homogeneous multi-computers. Strong transparency.

Network Operating System



Loosely-coupled operating system for heterogeneous multi-computers (LAN and WAN). Weak transparency.

DOS: characteristics

- **Distributed Operating Systems**
  - **Allows a multiprocessor or multicomputer network resources to be integrated as a single system image**
  - **Hide and manage hardware and software resources**
  - **provides transparency support**
  - **provide heterogeneity support**
  - **control network in most effective way**
  - **consists of low level commands + local operating systems + distributed features**
  - **Inter-process communication (IPC)**

DOS: characteristics

- **remote file and device access**
- **global addressing and naming**
- **trading and naming services**
- **synchronization and deadlock avoidance**
- **resource allocation and protection**
- **global resource sharing**
- **deadlock avoidance**
- **communication security**
- **no examples in general use but many research systems: Amoeba, Chorus etc.**

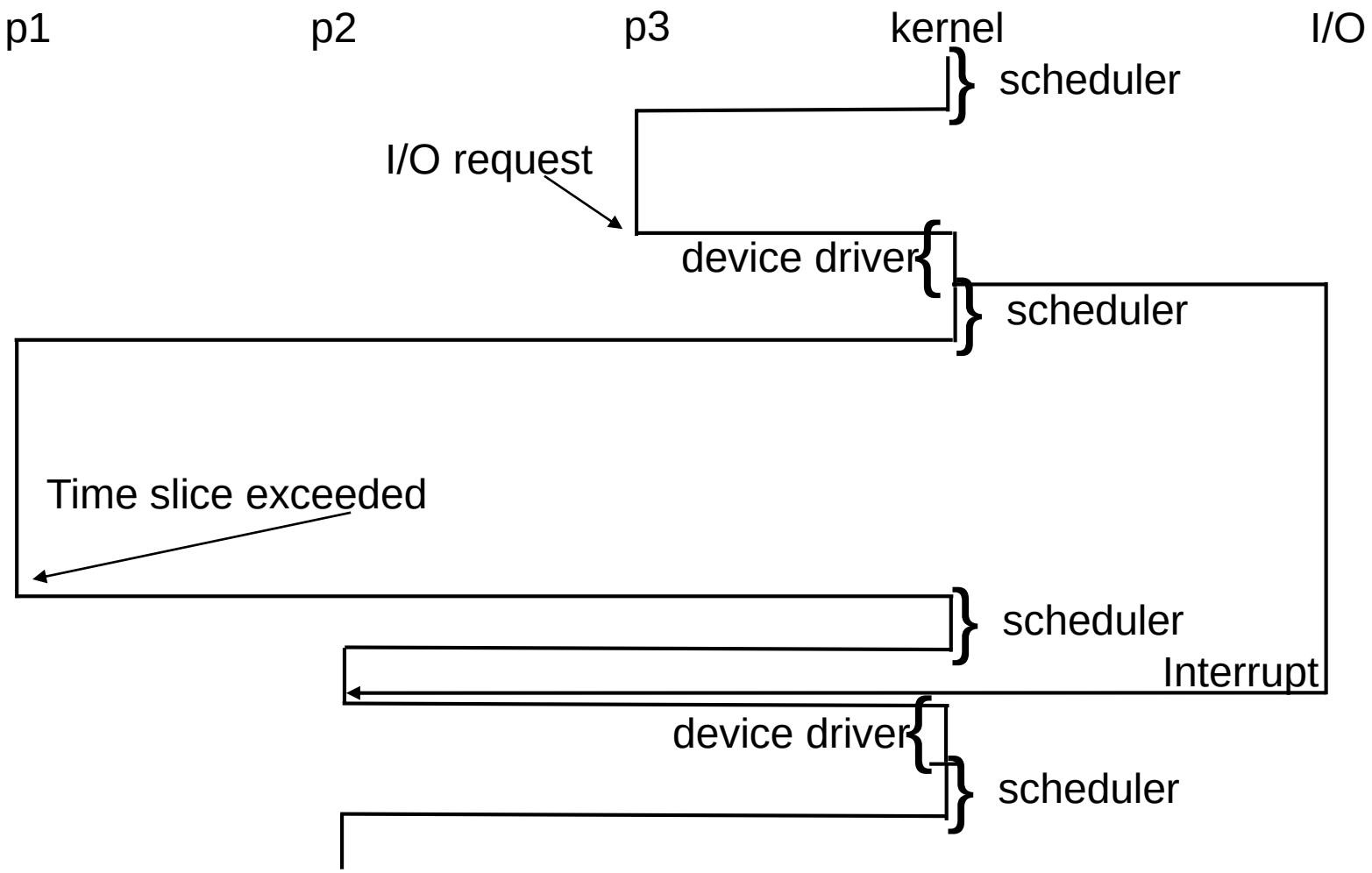
NOS: characteristics

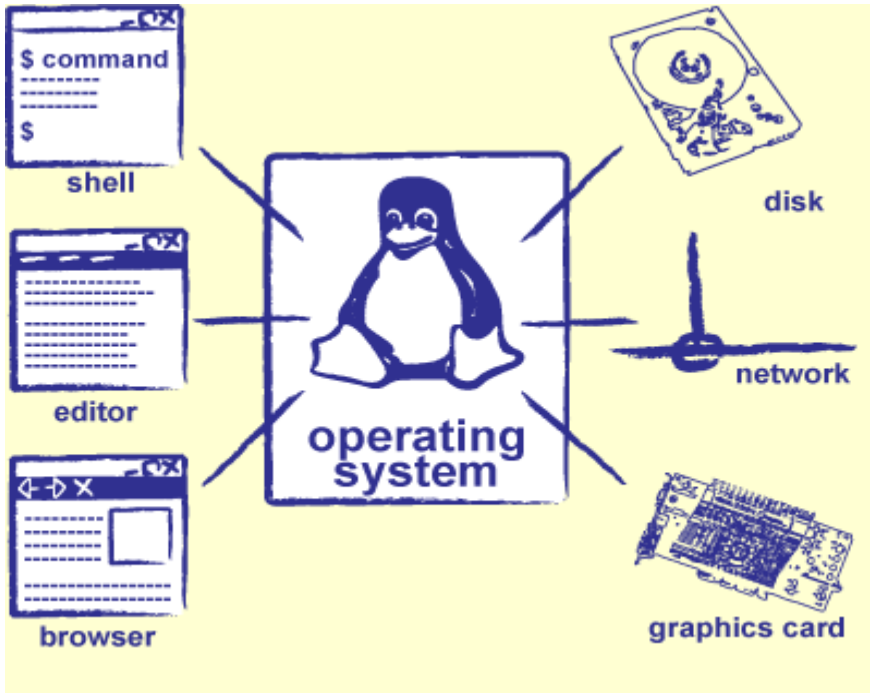
- **Network Operating System**
  - extension of centralized operating systems
  - offer local services to remote clients
  - each processor has own operating system
  - user owns a machine, but can access others (e.g. rlogin, telnet)
  - no global naming of resources
  - system has little fault tolerance
  - e.g. UNIX, Windows NT, 2000

Network v.s. Distributed Operating Systems

Features	**Network OS**	**Distributed OS**
SSI (Single System Image)	NO Ssh, sftp, no view of remote memory	YES Process migration, NFS, DSM (Distr. Shared memory)
Autonomy	High Local OS at each computer No global job coordination	Low A single system-wide OS Global job coordination
Fault Tolerance	Unavailability grows as faulty machines increase.	Unavailability remains little even if fault machines increase.

Example of Multiprogramming





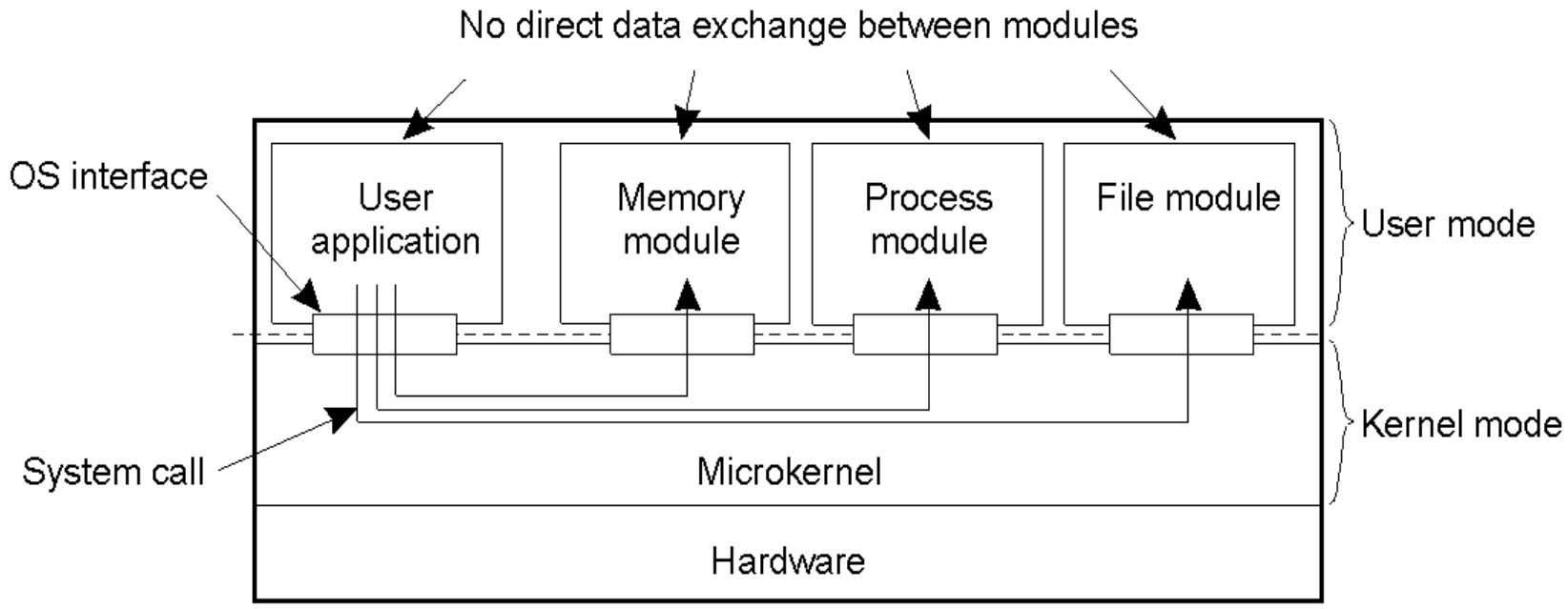
Time Sharing [3]

- Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time.
- Time-sharing or multitasking is a logical extension of multiprogramming.
- Processor's time which is shared among multiple users simultaneously is termed as time-sharing.
- The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the *objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.*

- Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently.
- Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if **n** users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

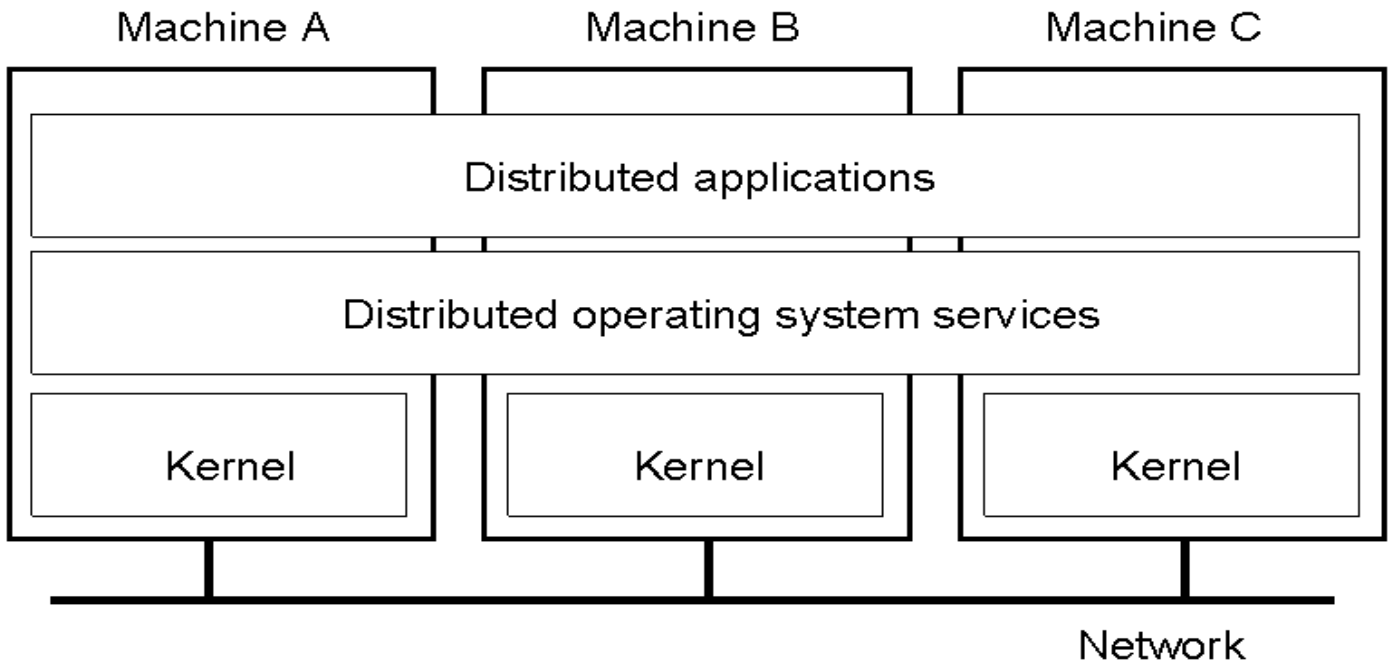
- Difference between Multiprogramming, multitasking, multithreading and multiprocessing
- **Multiprogramming –** A computer running more than one program at a time (like running Excel and Firefox simultaneously).
- **Multiprocessing –** A computer using more than one CPU at a time.
- **Multitasking –** Tasks sharing a common resource (like 1 CPU).
- **Multithreading** is an extension of multitasking.

Uniprocessor Operating System



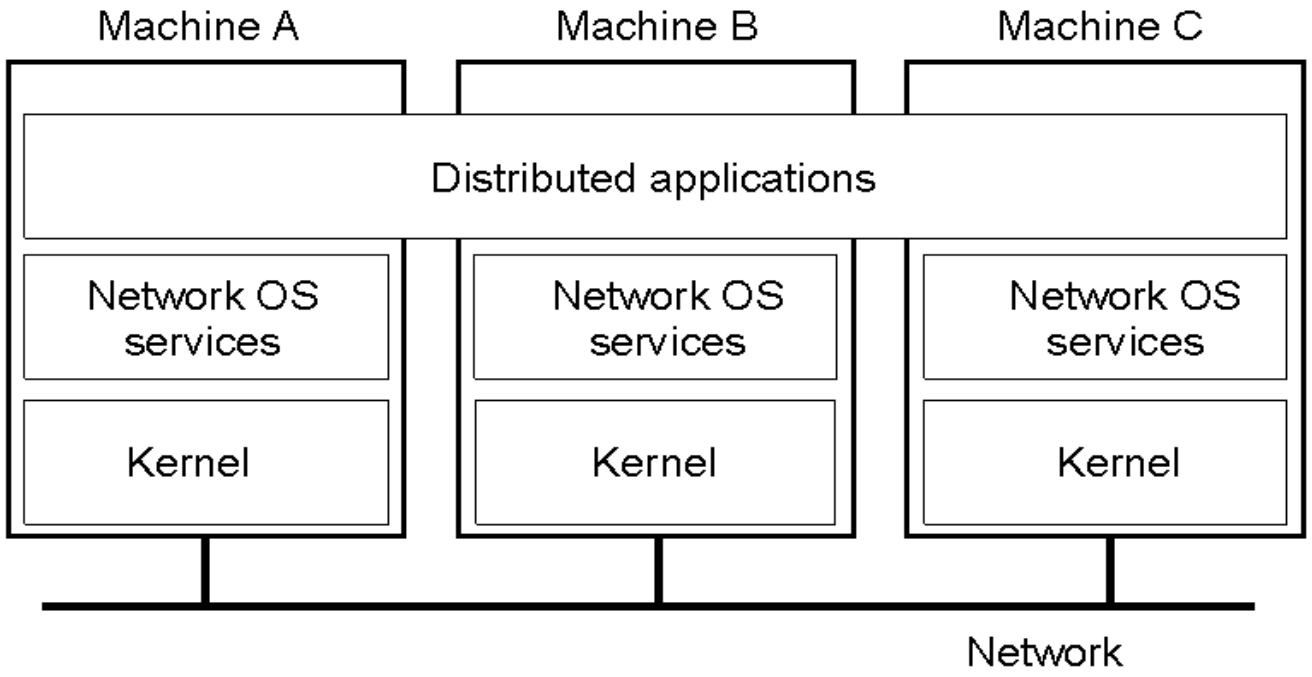
Separating applications from operating system code through a microkernel.

Distributed Operating System



Tightly-coupled operating system for multi-processors and homogeneous multi-computers. Strong transparency.

Network Operating System



Loosely-coupled operating system for heterogeneous multi-computers (LAN and WAN). Weak transparency.

DOS: characteristics (1)

- **Distributed Operating Systems**
  - **Allows a multiprocessor or multicomputer network resources to be integrated as a single system image**
  - **Hide and manage hardware and software resources**
  - **provides transparency support**
  - **provide heterogeneity support**
  - **control network in most effective way**
  - **consists of low level commands + local operating systems + distributed features**
  - **Inter-process communication (IPC)**

DOS: characteristics (2)

- **remote file and device access**
- **global addressing and naming**
- **trading and naming services**
- **synchronization and deadlock avoidance**
- **resource allocation and protection**
- **global resource sharing**
- **deadlock avoidance**
- **communication security**
- **no examples in general use but many research systems: Amoeba, Chorus etc.**

NOS: characteristics

- **Network Operating System**
  - extension of centralized operating systems
  - offer local services to remote clients
  - each processor has own operating system
  - user owns a machine, but can access others (e.g. rlogin, telnet)
  - no global naming of resources
  - system has little fault tolerance
  - e.g. UNIX, Windows NT, 2000

Network v.s. Distributed Operating Systems

Features	**Network OS**	**Distributed OS**
SSI (Single System Image)	NO Ssh, sftp, no view of remote memory	YES Process migration, NFS, DSM (Distr. Shared memory)
Autonomy	High Local OS at each computer No global job coordination	Low A single system-wide OS Global job coordination
Fault Tolerance	Unavailability grows as faulty machines increase.	Unavailability remains little even if fault machines increase.

References :

Text Books :

[1] William Stallings, Operating System: Internals and Design Principles, Prentice Hall, 8th Edition,2014, ISBN-10: 0133805913

[2] Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, Operating System Concepts, John Wiley & Sons ,Inc., 9th Edition,2012, ISBN 978-1-118-06333-0

Web :

[1] https://homepage.cs.uri.edu/

[2] https://electricalfundablog.com

[3] https://bansalwiki.blogspot.com/2013/01/time-sharing-operating-system.html

[4]https://vajiramias.com/current-affairs/compatible-time-sharing-system-ctss/5d2ad8ba1d5def60850adbc6/

References :

[1] https://homepage.cs.uri.edu/

[2] https:\\electricalfundablog.com

[3]https://bansalwiki.blogspot.com/2013/01/time-sharing-operating-system.html

[4] https://vajiramias.com/current-affairs/compatible-time-sharing-system-ctss/5d2ad8ba1d5def60850adbc6/

References :

Text Books :

[1] William Stallings, Operating System: Internals and Design Principles, Prentice Hall, 8th Edition,2014, ISBN-10: 0133805913

[2] Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, Operating System Concepts, John Wiley & Sons ,Inc., 9th Edition,2012, ISBN 978-1-118-06333-0

Web :

[1] https://homepage.cs.uri.edu/

[2] https://electricalfundablog.com

[3] https://bansalwiki.blogspot.com/2013/01/time-sharing-operating-system.html

[4]https://vajiramias.com/current-affairs/compatible-time-sharing-system-ctss/5d2ad8ba1d5def60850adbc6/