OPERATING SYSTEM REVISION

COMPLETE REVISION

Vishvadeep Gothi

Highlights

- AIR: 19, 119, 440, 682
- ME IISc Bangalore
- Mtech, BITS Pilani (Data Science)

Are You Ready?

Lets Begin the Unlimited Learning

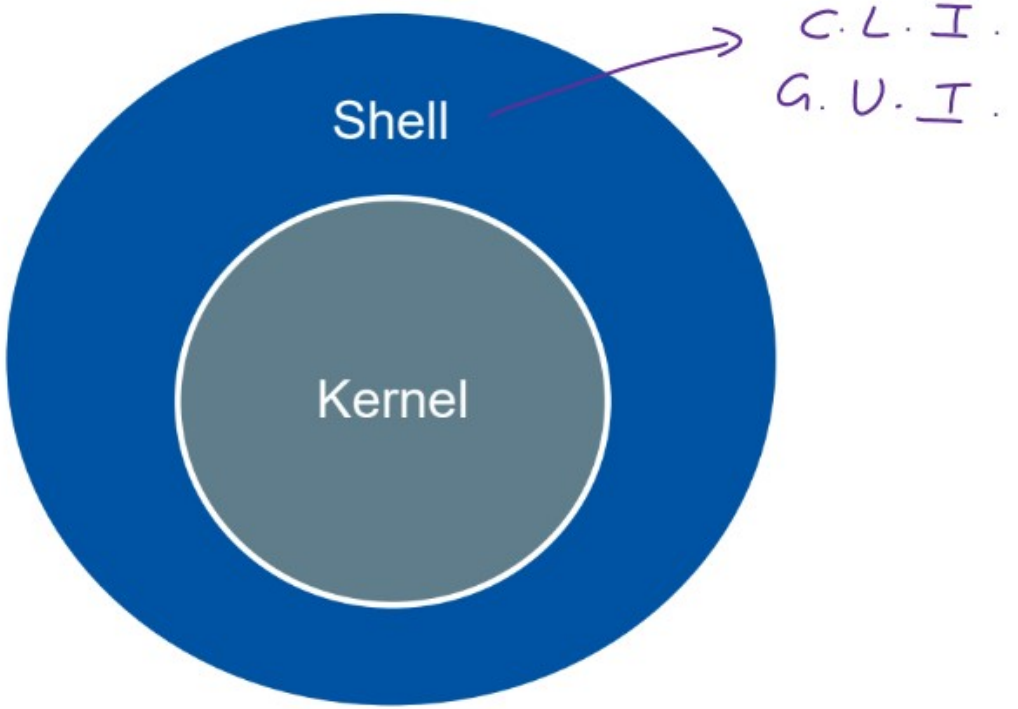
Operating System

- Software abstracting hardware
- Interface between user and hardware
- Set of utilities to simplify application development/execution
- Control program
- Acts like a government

Services of Operating Systems

- User Interface
- Program Execution
- I/O Operation
- File-System Manipulation
- Communication (Inter-process Communication)
- Error Detection
- Resource Allocation
- Accounting
- Protection & Security

Parts of Operating Systems



unacademy

System Call

A system call is a way for programs to interact with the operating system

Dual Mode of Operation

2 modes:

- User Mode (mode bit = 1)
- Kernel/System/Supervisor/Privileged Mode (mode bit = 0)

Dual Mode of Operation

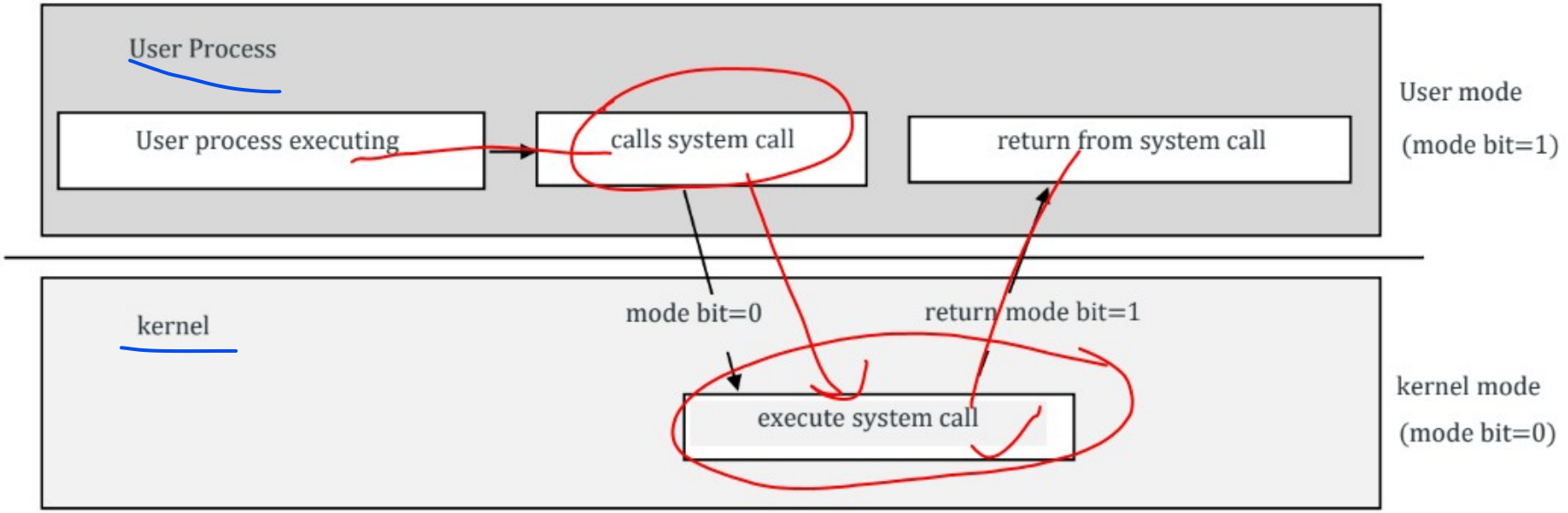


Fig. Transition from user to kernel mode.

unacademy

Types of Operating Systems

1. Uni-programming OS
2. Multiprogramming OS
3. Multi-Tasking OS
4. Multi-User OS
5. Multi-Processing OS
6. Embedded OS
7. Real-Time OS
8. Hand-held Device OS

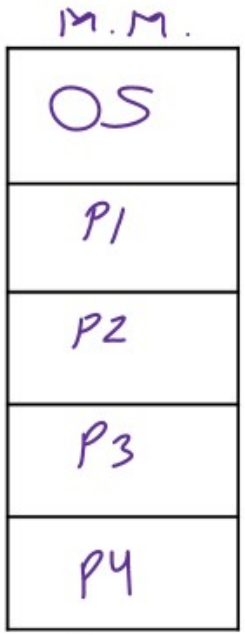
Uni-programming OS

OS allows only one process to reside in main memory

Multi-programming OS

↗ Non-preemptive
↘ preemptive

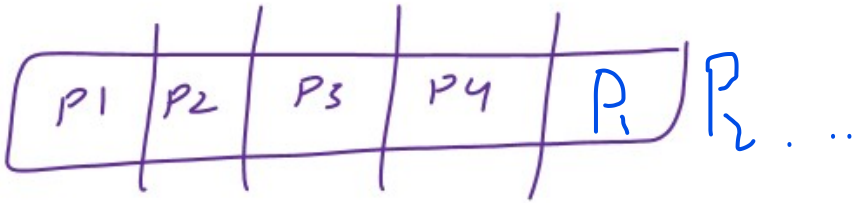
OS allows multiple processes to reside in main memory



Degree of multiprogramming
↳ no. of processes in mm.

Multi-tasking OS

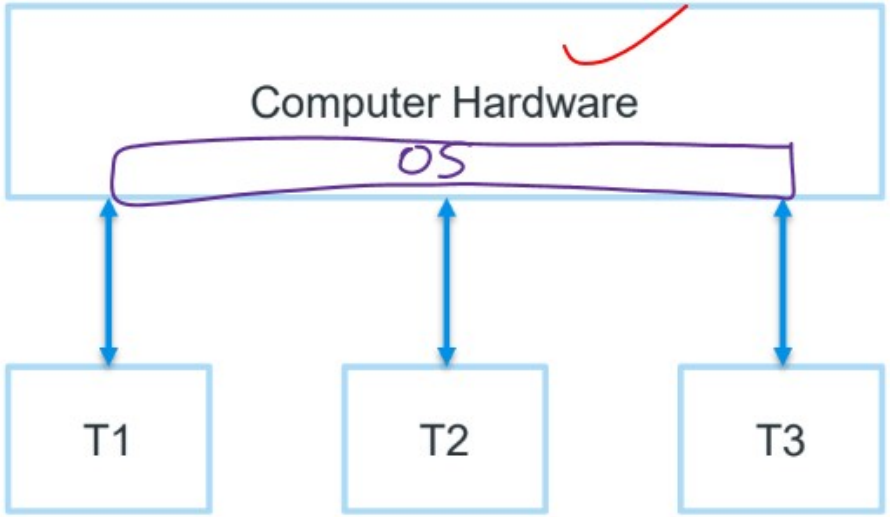
Extension of multi-programming OS in which processes execute in round-robin fashion.



unacademy

Multi-User OS

This OS allows multiple users to access single system simultaneously

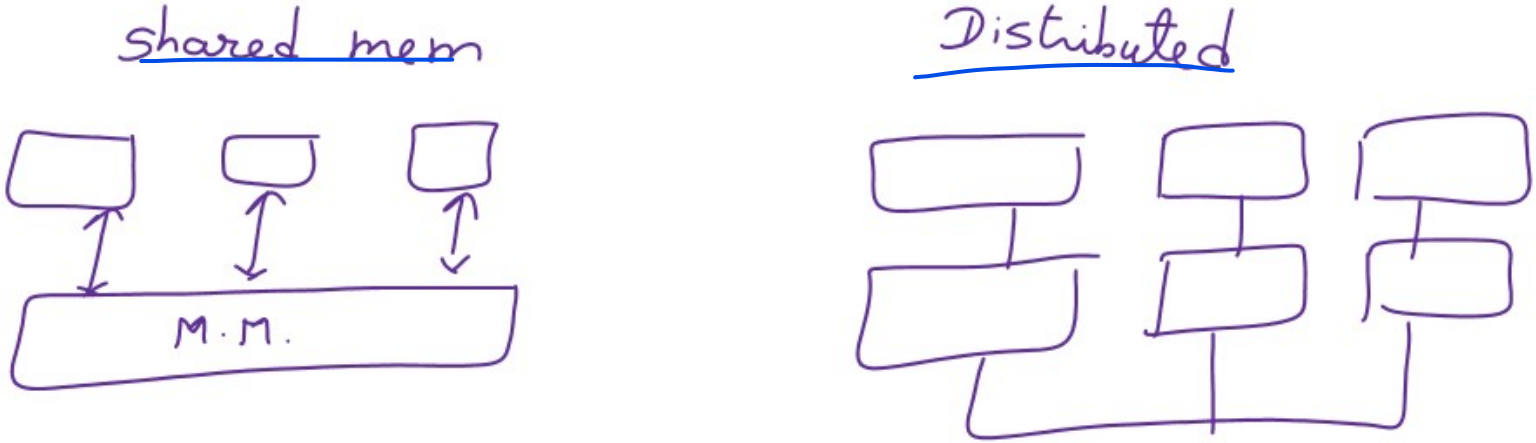


Minix
Linux
Unix

unacademy

Multi-Processing OS

This OS is used in computer systems with multiple CPUs



unacademy

Embedded OS

An OS for embedded computer systems.

- Designed for a specific purpose, to increase functionality and reliability for achieving a specific task.

Real-Time OS

Real time operating systems (RTOS) are used in environments where a large number of events, mostly external to the computer system, must be accepted and processed in a short time or within certain deadlines.

unacademy

Hand-held Device OS

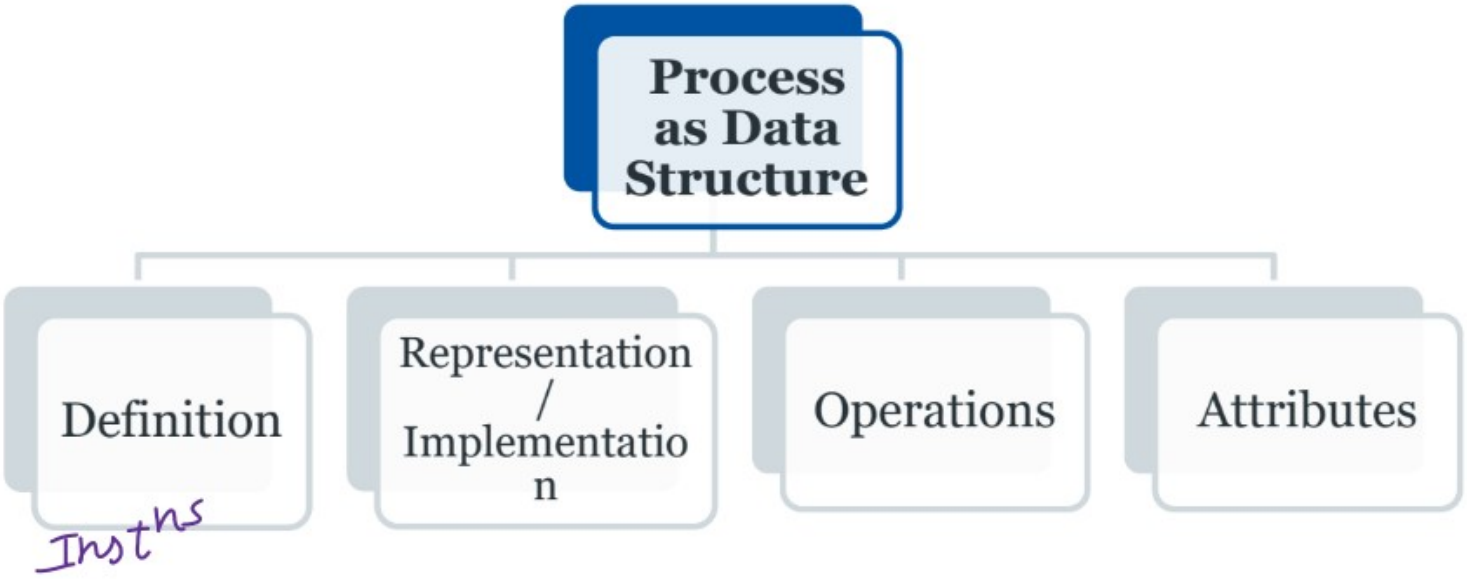
OS used in hand-held devices.

Process

Process = Prog. + Run-time Activity

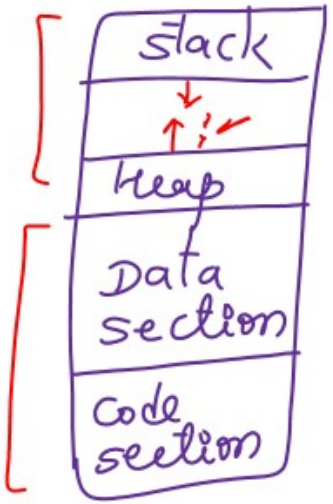
- **Process:**
  - Program under execution
  - An instance of a program
  - Schedulable/Dispatchable unit (CPU)
  - Unit of execution (CPU)
  - Locus of control (OS)

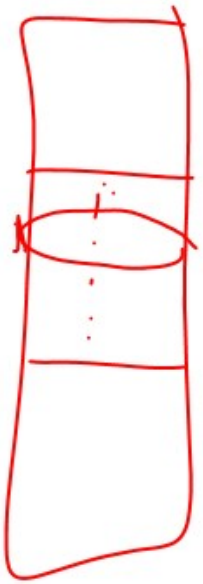
Process as Data Structure



unacademy

Representation of a Process

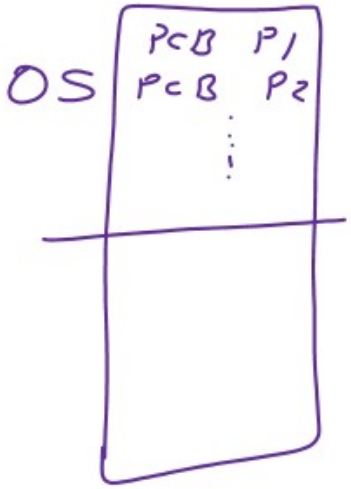




unacademy

Attributes of a Process

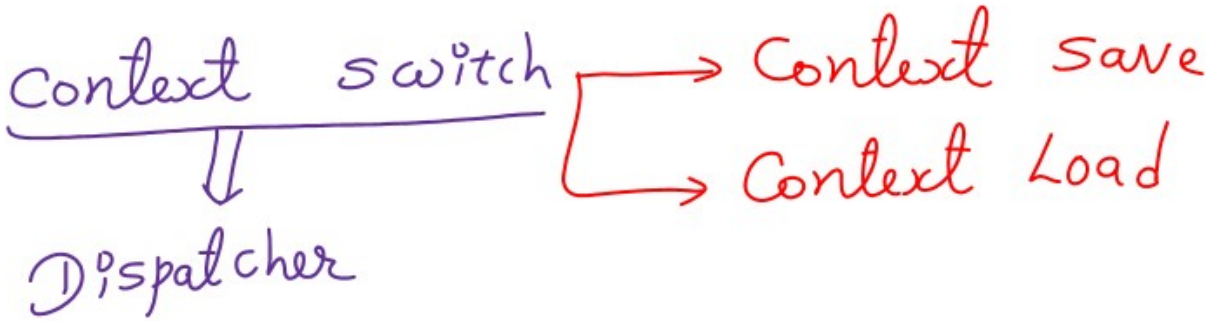
- PID
- PC
- GPR
- List of Devices
- Type
- Size
- Memory Limits
- Priority
- State
- List of Files



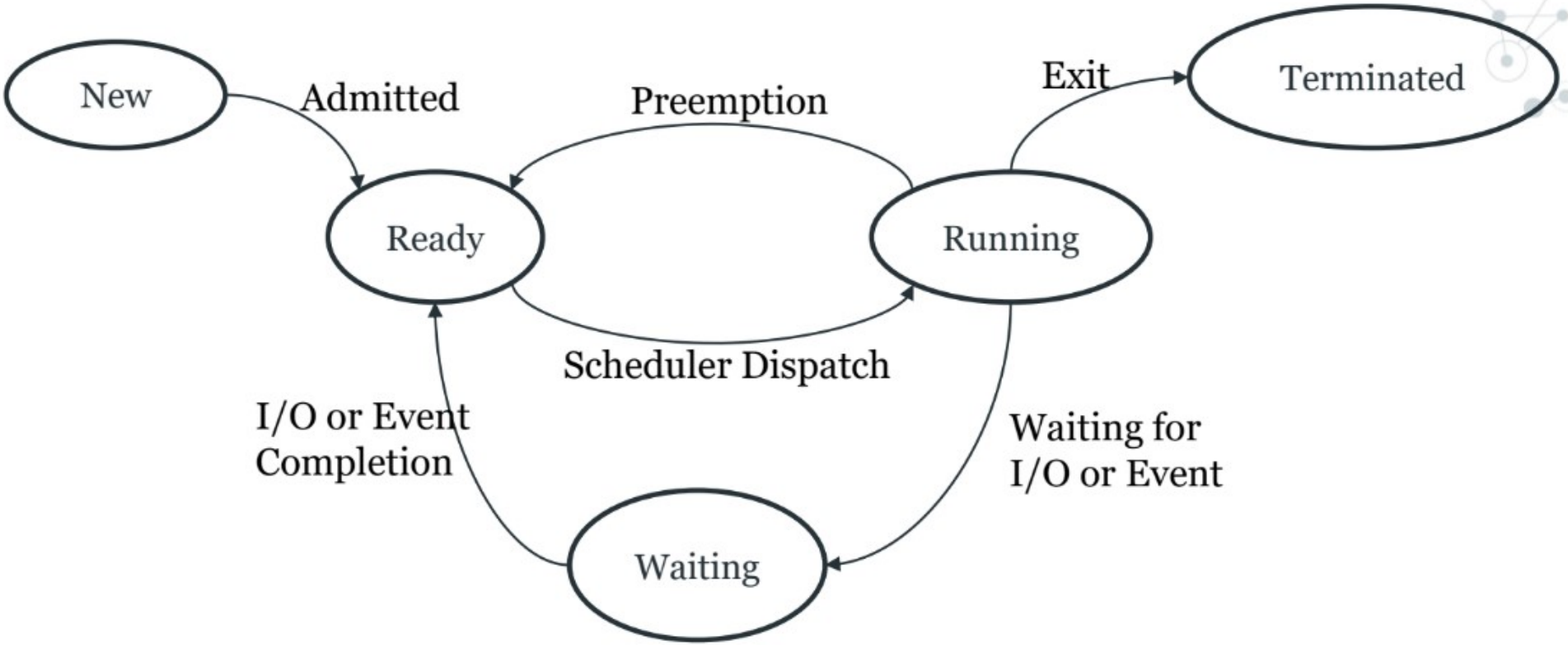
unacademy

Context

The content of PCB of a process are collectively know as 'Context' of that process



Process States



unacademy

Process States

New: All installed processes are in known to be in new state

Ready: All processes which are waiting to run on CPU are known to be in ready state

Running: A process which is running on CPU has its state as running

Terminated: A completed process has its state as terminated

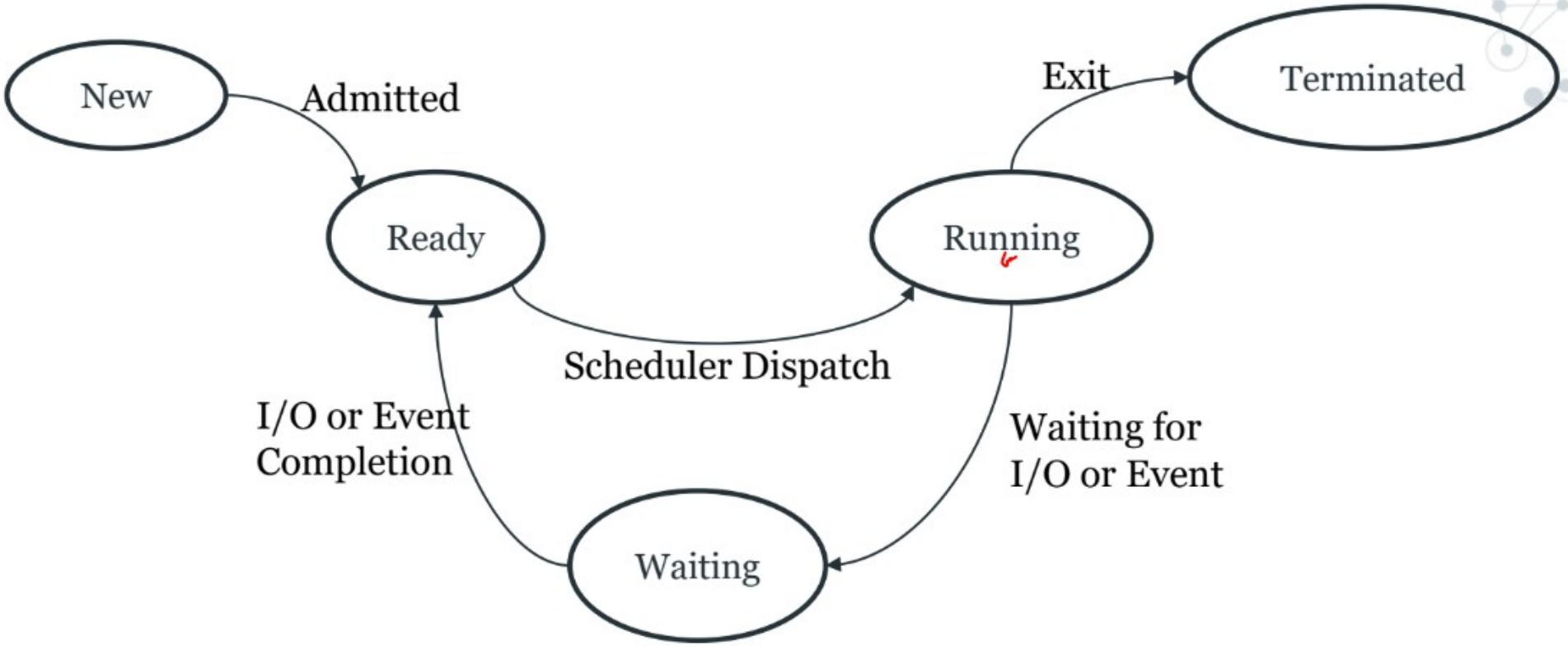
Blocked: All processes which are waiting for any IO or event

Process States Transitions

2 Transitions are voluntary:

- Running to Terminated
- Running to Blocked

Process States: Non-preemptive



unacademy

CPU vs IO Bound Process

CPU Bound: If the process is intensive in terms of CPU operations

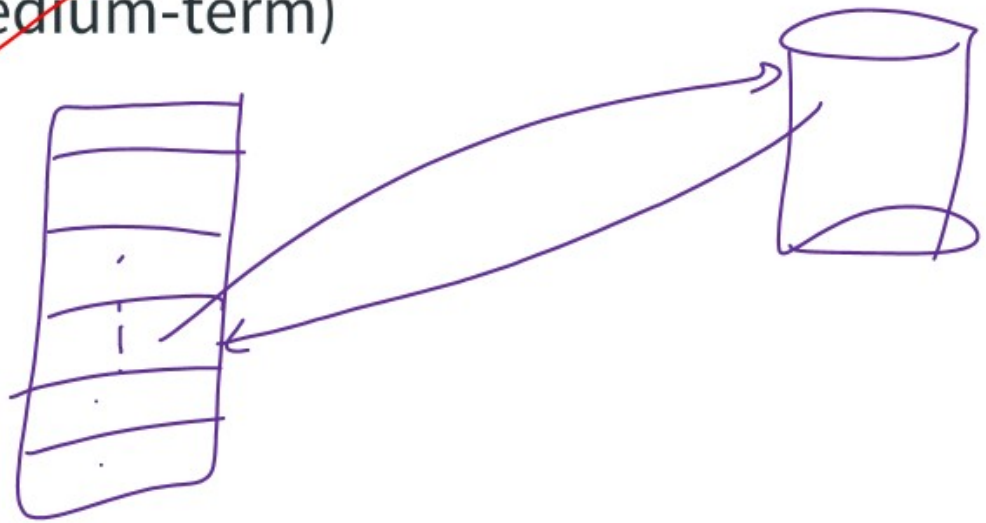
IO Bound: If the process is intensive in terms of IO operations

Scheduling Queues

- Job Queue
- Ready Queue
- Device Queue

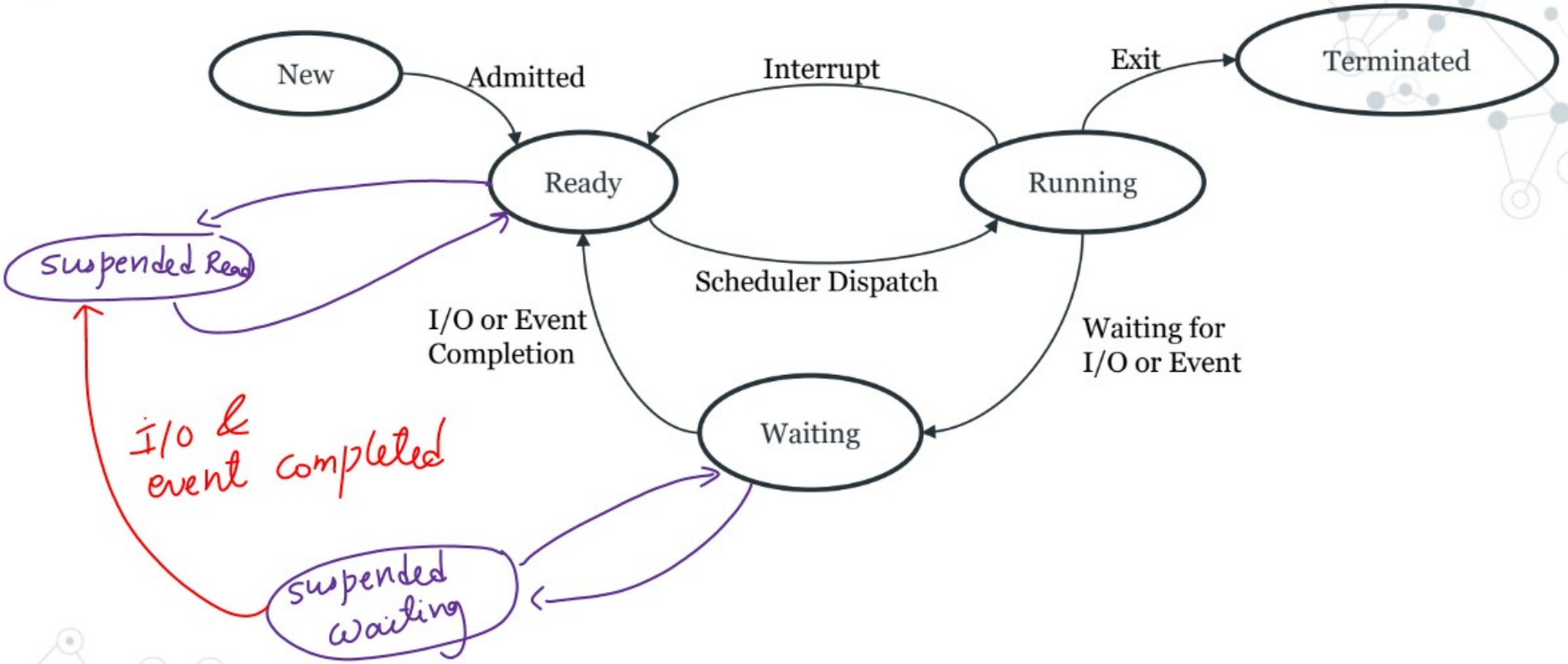
Types of Schedulers

1. Long-Term Scheduler (Job)
2. Short-Term Scheduler (CPU)
3. Mid-Term Scheduler (Medium-term)



unacademy

Updated Process States



unacademy

CPU Scheduling

Function:

Make a selection

Goal

- Minimize Wait time and Turn-around time
- Maximize CPU utilization (Throughput)
- Fairness

unacademy

Scheduling Times

1. Arrival Time (AT) ✓
2. Burst Time (BT) / service time
3. Completion Time(CT)
4. Turnaround Time (TAT) = CT − AT
5. Waiting Time (WT) = TAT − BT
6. Response Time (RT):
7. Throughput:

CPU Scheduling: Types

CPU Scheduling:

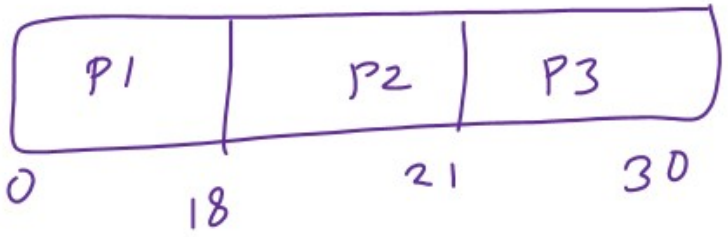
1. Preemptive
2. Non-preemptive

FCFS (First Come First Serve)

- Criteria: A.T.
  - Tie-breaker: smaller process_id
- Type: Non-preemption

FCFS (First Come First Serve)

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	18	18	18	0
P2	0	3	21	21	18
P3	0	9	30	30	21



$$\text{avg TAT} = \frac{69}{3} = 23$$

$$\text{avg WT} = \frac{39}{3} = 13$$

unacademy

Convoy Effect

↓

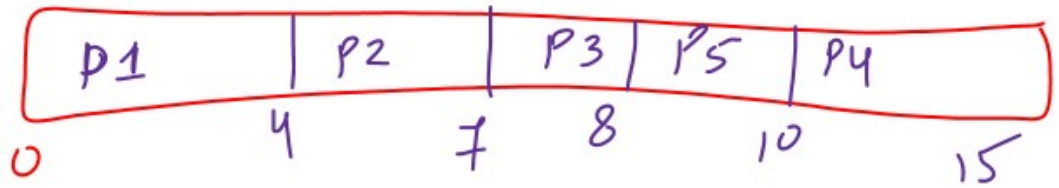
only in FCFS

SJF (Shortest Job First)

- Criteria: B.T.
  - Tie-breaker: FCFS
- Type: non-preemptive

SJF (Shortest Job First)

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	4			
P2	4	3			
P3	6	1			
P4	3	5			
P5	8	2			



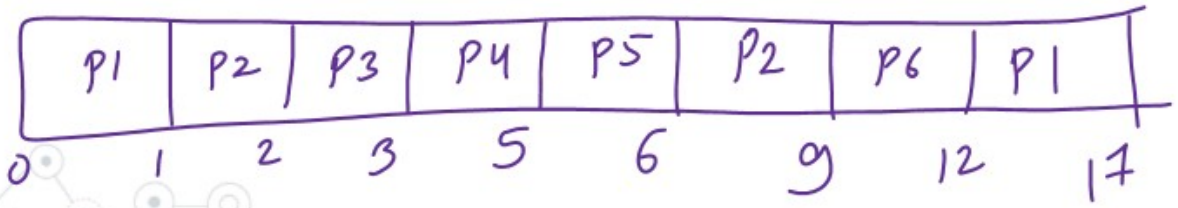
unacademy

SRTF (Shortest Remaining Time First)

- Criteria: BT
  - Tie-breaker: FCFS
- Type: Preemptive

SRTF (Shortest Remaining Time First)

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	6			
P2	1	4			
P3	2	1			
P4	3	2			
P5	4	1			
P6	5	3			



unacademy

Problems with SJF & SRTF

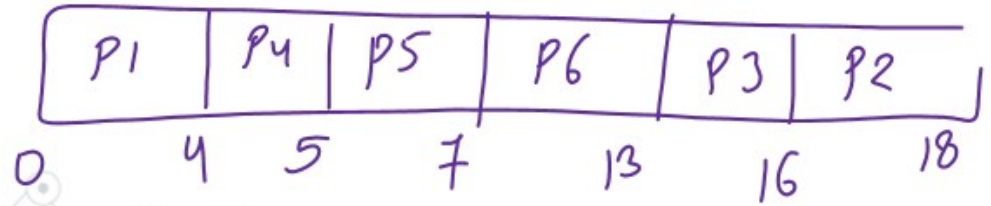
1. Starvation
2. No fairness
3. Practical Implementation is not possible

Priority Based Scheduling

- Criteria: Priority
  - Tie-breaker: Given in Question
- Type: Non-preemptive , preemptive
- Priority:
  - Static
  - Dynamic

Priority Scheduling: Non-Preemptive

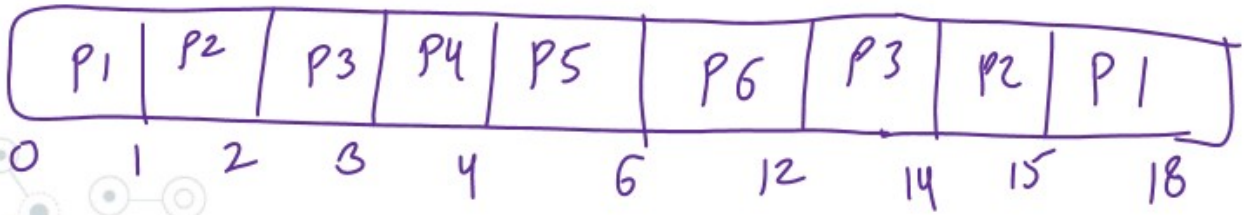
Process	Arrival Time	Burst Time	Priority	Completion Time	Turnaround Time	Waiting Time
P1	0	4	4			
P2	1	2	5			
P3	2	3	6			
P4	3	1	10(Highest)			
P5	4	2	9			
P6	5	6	7			



unacademy

Priority Scheduling: Preemptive

Process	Arrival Time	Burst Time	Priority	Completion Time	Turnaround Time	Waiting Time
P1	0	~~4~~ 3	4			
P2	1	~~2~~ 1	5			
P3	2	~~3~~ 2	6			
P4	3	~~1~~ 0	10(Highest)			
P5	4	~~2~~	9			
P6	5	6	7			



$\frac{6}{18} = \frac{1}{3}$

unacademy

Starvation

Soln ⇒ Aging (dynamic priority)

Round-Robin

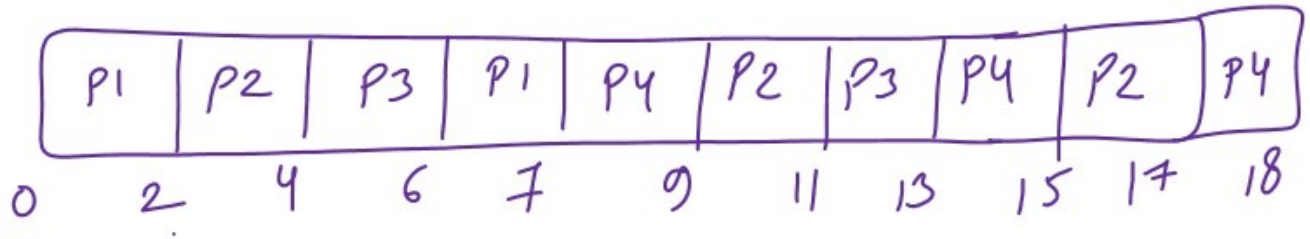
- Criteria: A.T. + Q.
  - Tie-breaker: process_id smaller
- Type: Preemptive

Round-Robin

R.Q. = P1, P2, P3, P1, P4, P2, P3

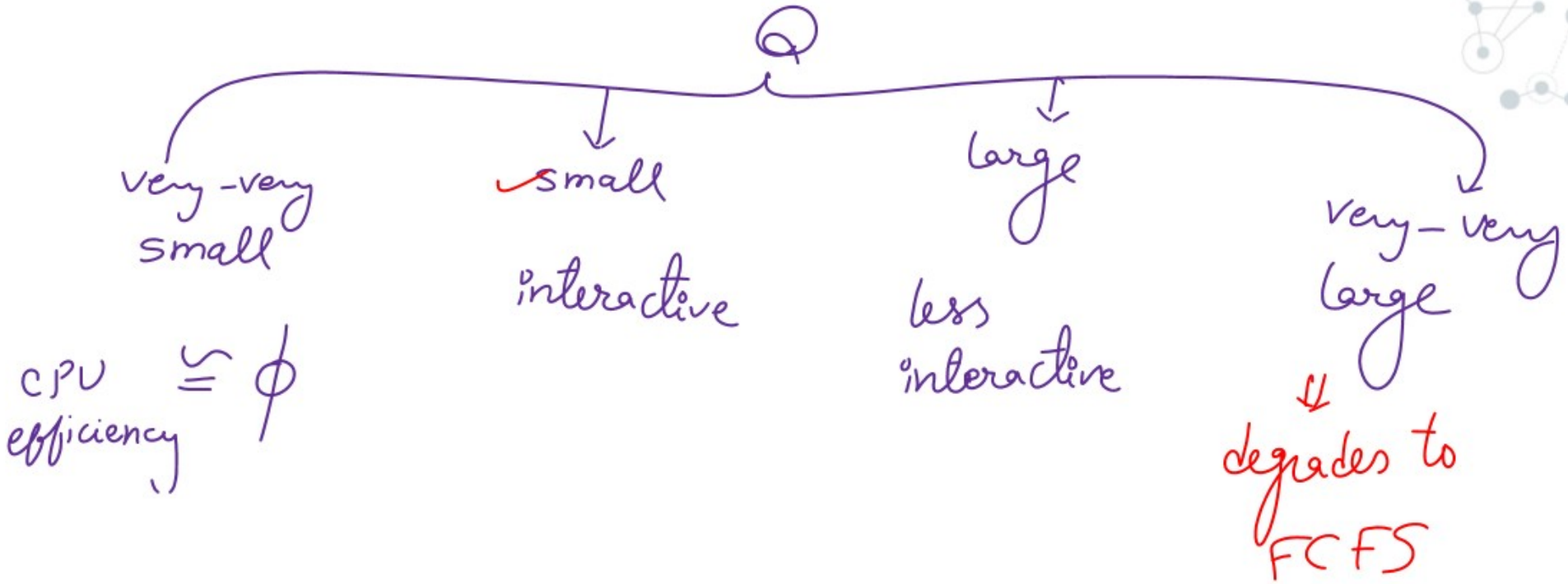
Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	3			
P2	1	6			
P3	2	4			
P4	3	5			

Q=2



unacademy

What Should Be the Quantum Value?



Multilevel Queue (MLQ) Scheduling

- System Processes
- Foreground Processes
- Background Processes

Scheduling Among Queues

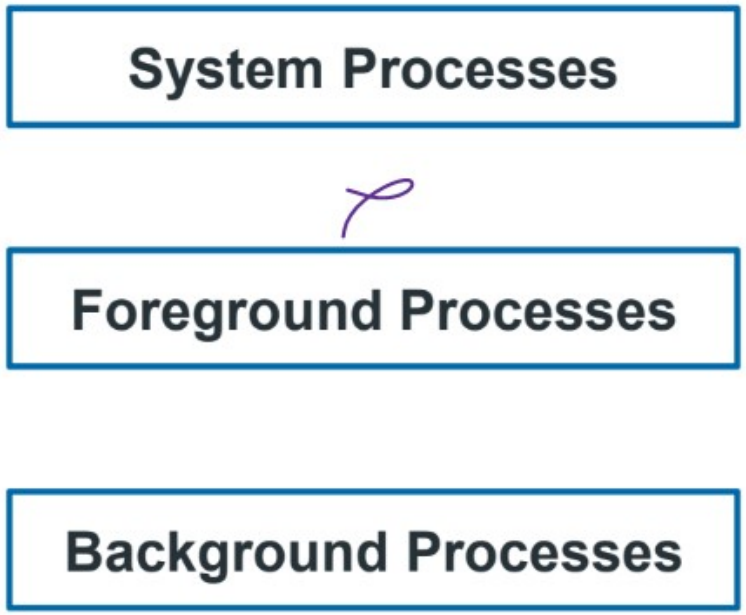
1. Fixed priority preemptive scheduling method
2. Time slicing

Multilevel Queue (MLQ) Scheduling

Disadvantages:

1. Some processes may starve for CPU if some higher priority queues are never becoming empty.
2. It is inflexible in nature.

Multilevel Feedback Queue (MLFQ) Scheduling



unacademy

Multilevel Feedback Queue (MLFQ) Scheduling

Disadvantage:

1. Some processes may starve for CPU if some higher priority queues are never becoming empty.

Advantage:

1. Flexible

FCFS

Advantages:

1. Easy to implement
2. No complex logic
3. No starvation

Disadvantages:

1. No option of Preemption
2. Convoy effect makes the system slow

SJF

Advantages:

1. Minimum average waiting time among non-preemptive scheduling
2. Better throughput in continue run

Disadvantages:

1. No practical implementation because Burst time is not known in advance
2. No option of Preemption
3. Longer Processes may suffer from ~~deadlock~~ starvation

unacademy

SRTF

Advantages:

1. Minimum average waiting time among all scheduling algorithm
2. Better throughput in continue run

Disadvantages:

1. No practical implementation because Burst time is not known in advance
2. Longer Processes may suffer from starvation

unacademy

Priority Based Algorithm

Advantages:

1. Better response for real time situations

Disadvantages:

1. Low Priority Processes may suffer from ~~deadlock~~ starvation

Round Robin Algorithm

Advantages:

1. All processes execute one by one, so no starvation
2. Better interactiveness
3. Burst time is not required to be known in advance

Disadvantages:

1. Average waiting time and turnaround time are more
2. Can degrade to FCFS

Types of Processes

1. Independent
2. Cooperating/Coordinating/Communicating

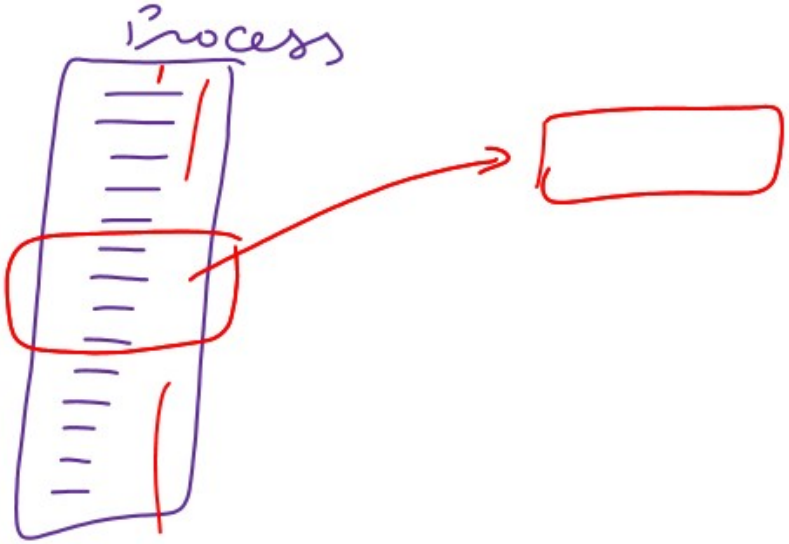
unacademy

Problems Without Synchronization

- Problems without Synchronization:
  - Inconsistency
  - Loss of Data
  - Deadlock

Critical Section

The critical section is a code segment where the shared variables can be accessed



Race Condition

A race condition is an undesirable situation, it occurs when the final result of concurrent processes depends on the sequence in which the processes complete their execution.

Critical Section

The critical section is a code segment where the shared variables can be accessed

Solution of Critical Section Problem

- Requirements of Critical Section problem solution:
  1. Mutual Exclusion
  2. Progress
  3. Bounded Waiting

Solution 1: Using Lock

```
Boolean lock=false;

while(true)
{
    while(lock);
    lock=true;
      CS
    lock=false;
      RS;
}
```

```
while(true)
{
    while(lock);
    lock=true;
      CS
    lock=false;
      RS;
}
```

No M.E. ✗

✓ Progress

No B.W. ✗

Solution 2: Using Turn

```
int turn=0;

P0
while(true)
{
   while(turn!=0);
     CS
   turn=1;
     RS;
}
```

```
P1
while(true)
{
   while(turn!=0);
     CS
   turn=0;
     RS;
}
```

M. E. ✓
Progress ✗
B. W. ✓

strict alternatn

Solution 3: Peterson's Solution

```
Boolean Flag[2];
int turn;
```

false
false

✓M.E.
✓Progress
✓B.W.

```
while(true) {
    Flag[i]=true;
    turn=j;
    while(Flag[j] && turn==j) ;
        CS
    Flag[i]=False;
        RS;
}
```

```
while(true) {
    Flag[j]=true;
    turn=i;
    while(Flag[j] && turn==i) ;
        CS
    Flag[j]=False;
        RS;
}
```

unacademy

Synchronization Hardware

1. TestAndSet()
2. Swap()

TestAndSet()

Returns the current value flag and sets it to true

TestAndSet()

```
Boolean Lock=False;

boolean TestAndSet(Boolean *trg)
{
    boolean rv = *trg;
    *trg = True;
    Return  rv;
}
```

```
while(true)
{
   while(TestAndSet(&Lock));
         CS
   Lock=False;
}
```

Swap()

local

global

```
Boolean Key, Lock=False;
```

```
void Swap(Boolean *a, Boolean *b)
{
   boolean temp = *a;
   *a=*b;
   *b=temp;
}
```

```
while(true
{
   Key = True;
   while (key==True)
        Swap(&Lock, &Key);
   CS
   Lock=False;
      RS
}
```

Synchronization Tools

1. Semaphore
2. Monitor

Semaphore

- Integer value which can be accessed using following functions only
  - wait() / P() / Degrade()
  - signal() / V() / Upgrade()

wait() & signal()

```
wait(S)
{
   while(S<=0);
   S--;
}
```

```
signal(S)
{
   S++;
}
```

S = X - 1

signal (S)

Question 1

Ans = 10

Consider a semaphore S, initialized with value 1. Consider 10 processes P1, P2 …. P10. All processes have same code as given below but, one process P10 has signal(S) in place of wait(S). If all processes can execute multiple times, then maximum number of processes which can be in critical section together ?

```
while(True)
{
  wait(S)
     C.S.
  signal(s)
}
```

P10

signal (S)

C.S.

unacademy

Question 2

Consider a semaphore S, initialized with value 1. Consider 10 processes P1, P2 …. P10. All processes have same code as given below but, one process P10 has signal(S) and wait(S) swapped. If all processes can execute only one time, then maximum number of processes which can be in critical section together ?

```
while(true)
{
  wait(S)
     C.S.
  signal(s)
}
```

P1

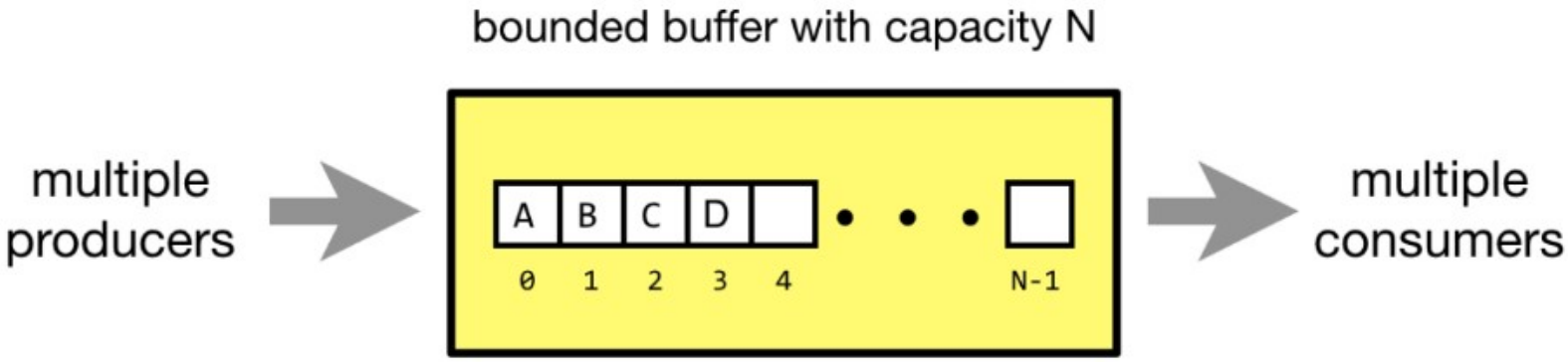
signal(S)

C.S.

signal(S)

Ans = 3

Bounded Buffer Problem



unacademy

Bounded Buffer Problem

- Known as producer-consumer problem also
- Buffer is the shared resource between producers and consumers

- Solution:
  - Producers must block if the buffer is full
  - Consumers must block if the buffer is empty

Bounded Buffer Problem: Solution

- Variables:
  - Mutex: Binary Semaphore to take lock on buffer (Mutual Exclusion) = 1
  - Full: Counting Semaphore to denote the number of occupied slots in buffer = 0
  - Empty: Counting Semaphore to denote the number of empty slots in buffer = N

Producer-Consumer Code

```
wait(Empty)
wait(mutex)
    // Add item
signal(mutex)
signal(Full)
```

```
wait(Full)
wait(mutex)
    // Remove item
signal(mutex)
signal(Empty)
```

Reader-Writer Problem

Consider a situation where we have a file shared between many people:

- If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her
- However, if some person is reading the file, then others may read it at the same time

- Solution:
  - If writer is accessing the file, then all other readers and writers will be blocked
  - If any reader is reading, then other readers can read but writer will be blocked

Reader-Writer Problem: Solution

- Variables:
  - mutex: Binary Semaphore to provide Mutual Exclusion
  - wrt: Binary Semaphore to restrict readers and writers if writing is going on
  - readcount: Integer variable, denotes number of active readers
- Initialization:
  - mutex: 1
  - wrt: 1
  - readcount: 0

Reader() Writer() Process

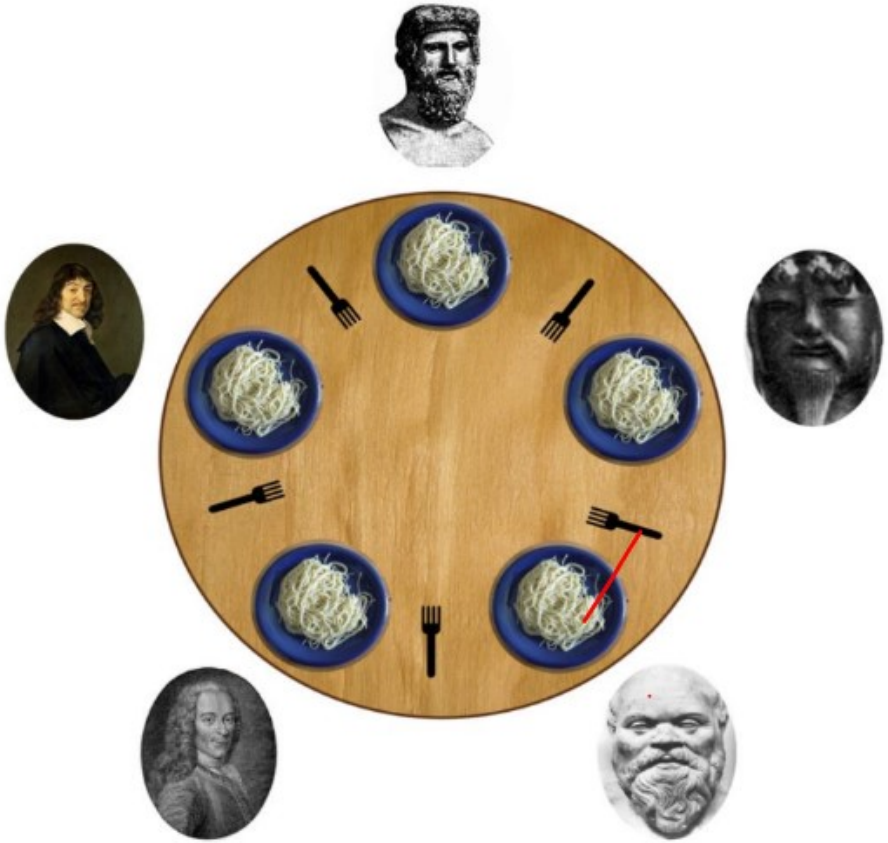
writer:-

```
wait (wrt)
    // write
signal (wrt)
```

Reader:-

```
wait (mutex)
 Rc ++
if (RC == 1)
      wait (wrt)
signal (mutex)
    //Read
wait (mutex)
 RC -- ;
if (RC == 0)
    signal (wrt)
signal (mutex)
```

Dining Philosopher Problem



unacademy

Dining Philosopher Problem

- K philosophers seated around a circular table
- There is one chopstick between each philosopher
- A philosopher may eat if he can pick up the two chopsticks adjacent to him
- One chopstick may be picked up by any one of its adjacent followers but not both

Dining Philosopher Problem: Solution

```
semaphore cs[k]

wait (cs[i])
wait (cs[(i+1) mod k])

// Eating
signal (cs[i])
signal (cs[(i+1) mod k])
```

Dining Philosopher Problem: Solution

Some of the ways to avoid deadlock are as follows –

1. There should be at most (k–1) philosophers on the table
2. A philosopher should only be allowed to pick their chopstick if both are available at the same time
3. One philosopher should pick the left chopstick first and then right chopstick next; while all others will pick the right one first then left one

LET'S TAKE A BREAK

SEE YOU IN 10MIN