Scheduling (Uniprocessor)

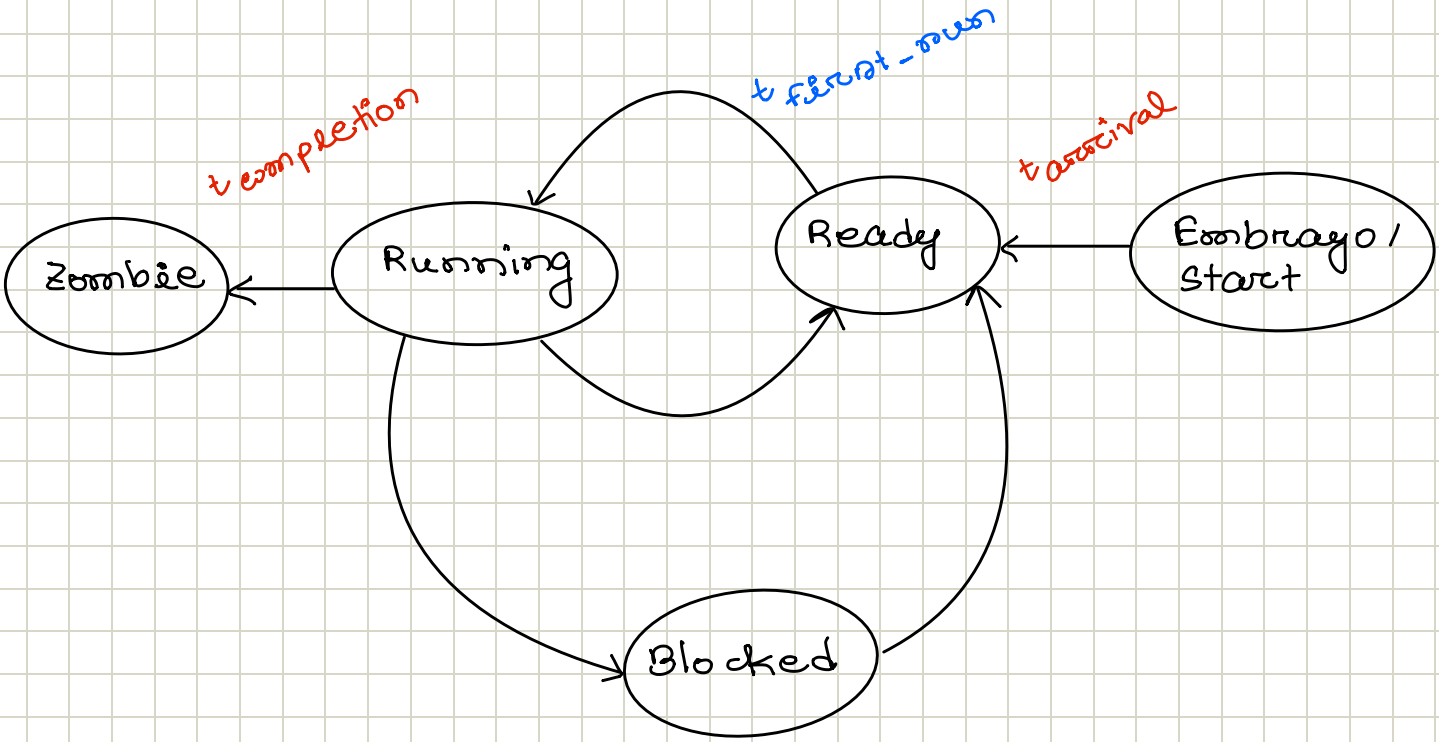
MLFQ (Multi level Feedback Queue)

Turn around time = $t_{completion} - t_{arrival}$

↳ SJF (Shortest Job First), STCF (Shortest Time to completion First)

Response time = $t_{first\_run} - t_{arrival}$

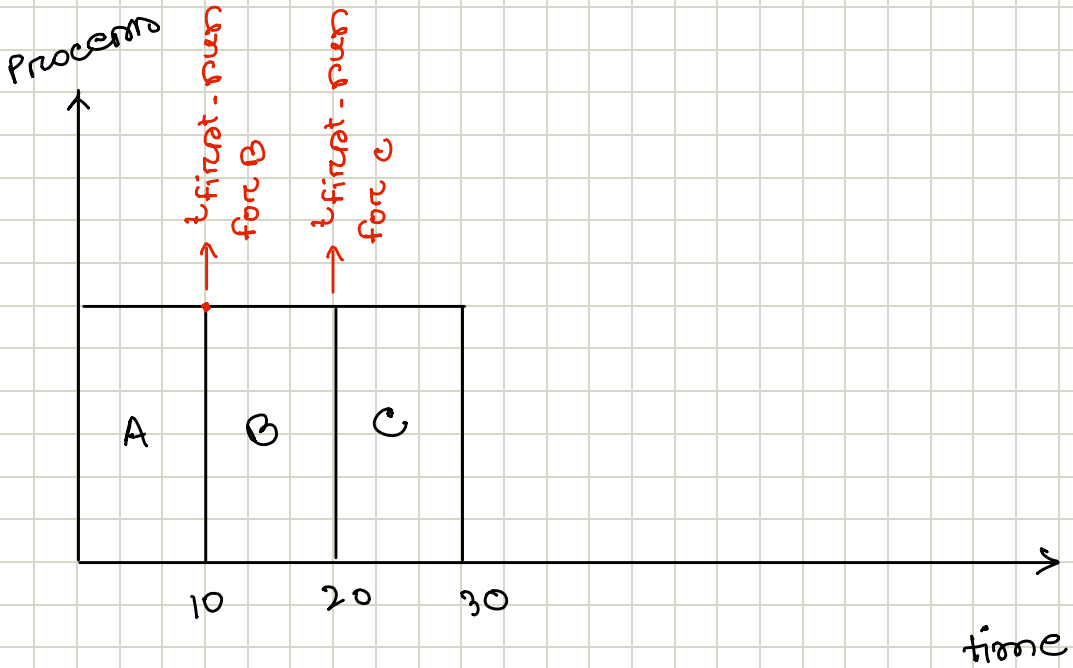
↳ Round Robin



First In First Out (FIFO)

Performs well if all five assumptions are held.

Process	Arrival	Duration / Burst Time
A	0	10
B	0	10
C	0	10



Gantt chart

Average turnaround time = $t_{completion} - t_{arrival}$

$$= \frac{(10-0)+(20-0)+(30-0)}{3}$$

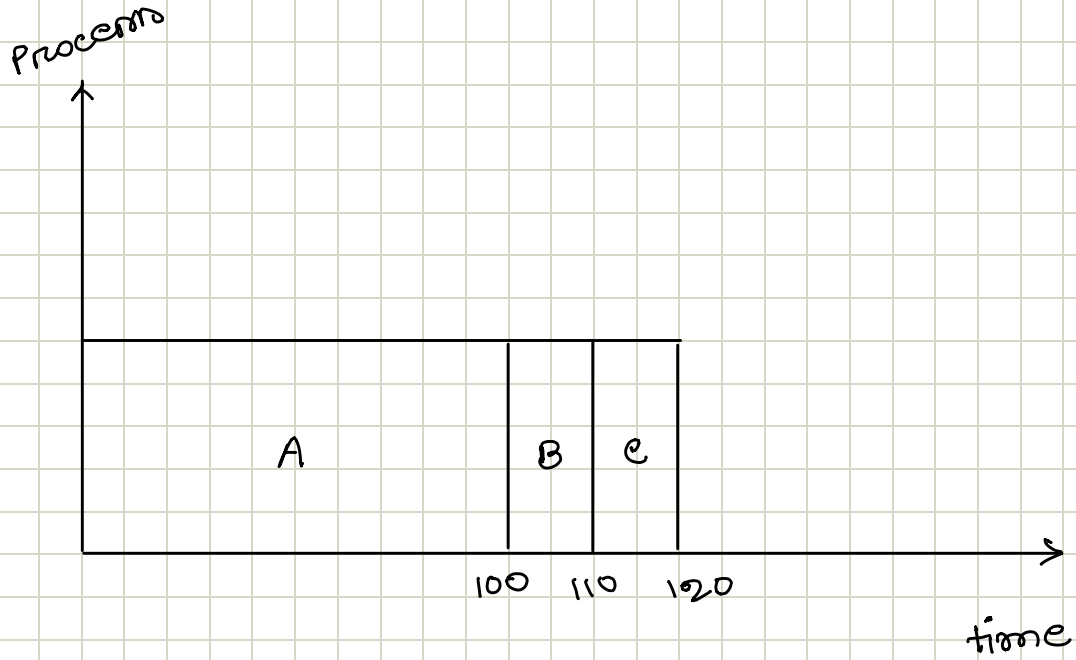
$$= 20$$

Problem of FIFO:

(1) Each job runs for the different time:

Convoy Effect

Process	Arrival	Duration / Burst Time
A	0	100
B	0	10
C	0	10



Gantt chart

Average turnaround time = $t_{completion} - t_{arrival}$

$$= \frac{100 + 110 + 120}{3}$$

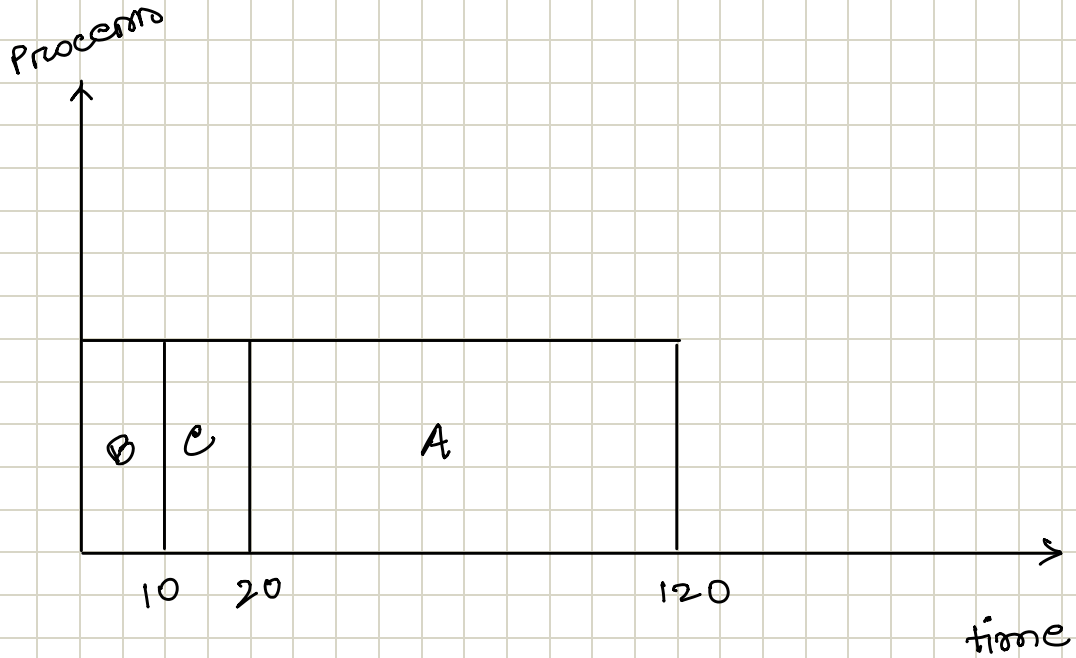
$$= 110$$

Shortest Job First (SJF)

Works well under assumptions 2-5

↳ Sort the jobs in ascending order based on arrival and duration

Process	Arrival	Duration / Burst Time
A	0	100
B	0	10
C	0	10



Gantt chart

Average turnaround time = $t_{completion} - t_{arrival}$

$$= \frac{10 + 20 + 120}{3}$$

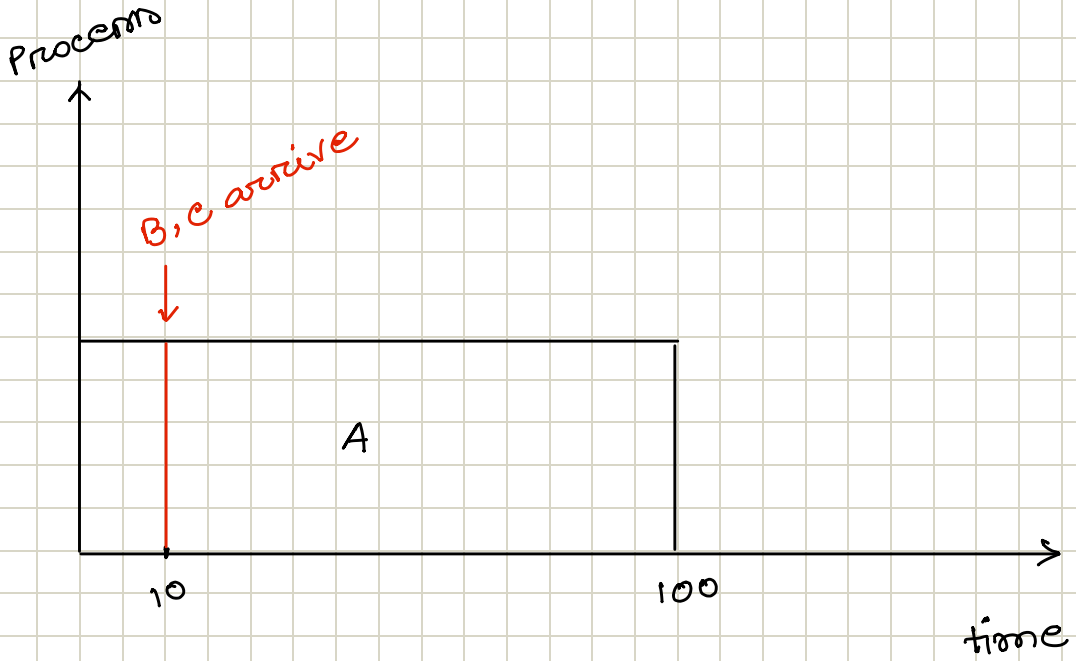
$$= 50$$

- Sort the available process (those who have already come to the system)
- In ascending order based on duration each time before running a process

Problem of SJF:

(2) Jobs arriving at the different time:

Process	Arrival	Duration / Burst Time
A	0	100
B	10	10
C	10	10



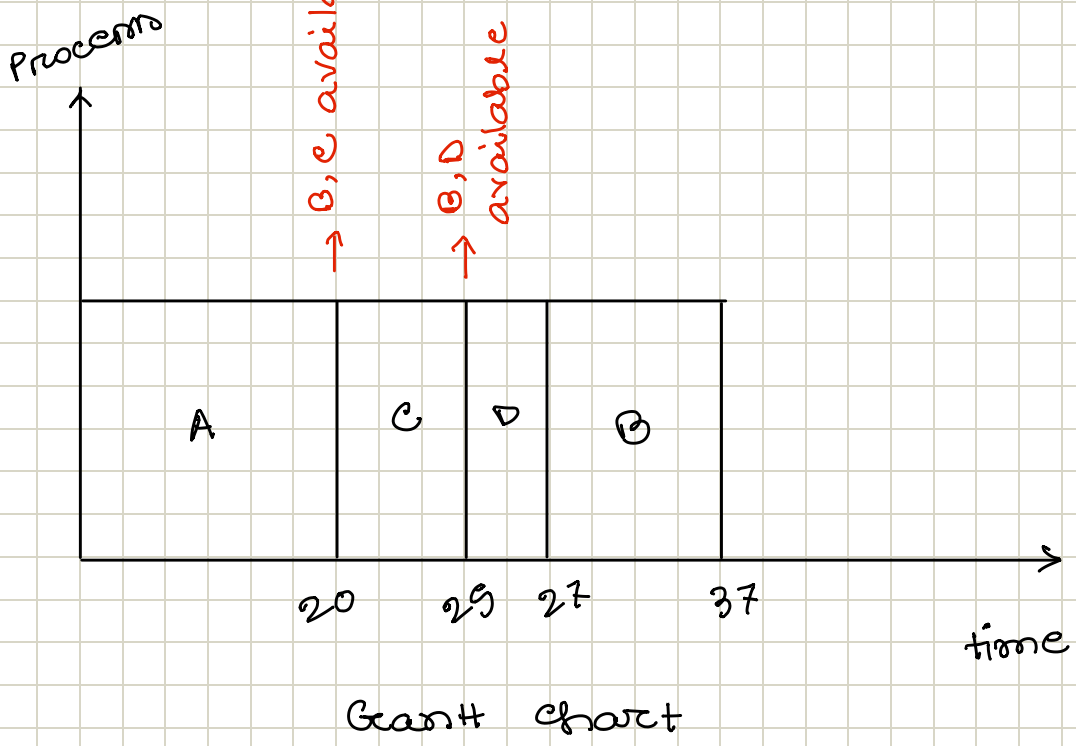
Gantt chart

Average turnaround time = $t_{completion} - t_{arrival}$

$$= \frac{(100-0) + (110-10) + (120-10)}{3}$$

$$= 103.33$$

Process	Arrival	Duration / Burst Time
A	0	20
B	10	10
C	12	5
D	25	2



Gantt chart

Average turnaround time = $t_{completion} - t_{arrival}$

$$= \frac{(20-0)+(25-12)+(27-25)+(37-10)}{3}$$

=

STCF (Preemptive Algorithm)

- Whenever a new process arrives or the running process completes, among the remaining process, take the process with shortest remaing time.

Process	Arrival	Duration / Burst Time
A	2	15
B	10	6
C	15	10
D	20	10
E	20	5

(1) Draw the gantt chart using STCF and calculate average turnaround time.

Rough

Remaining time

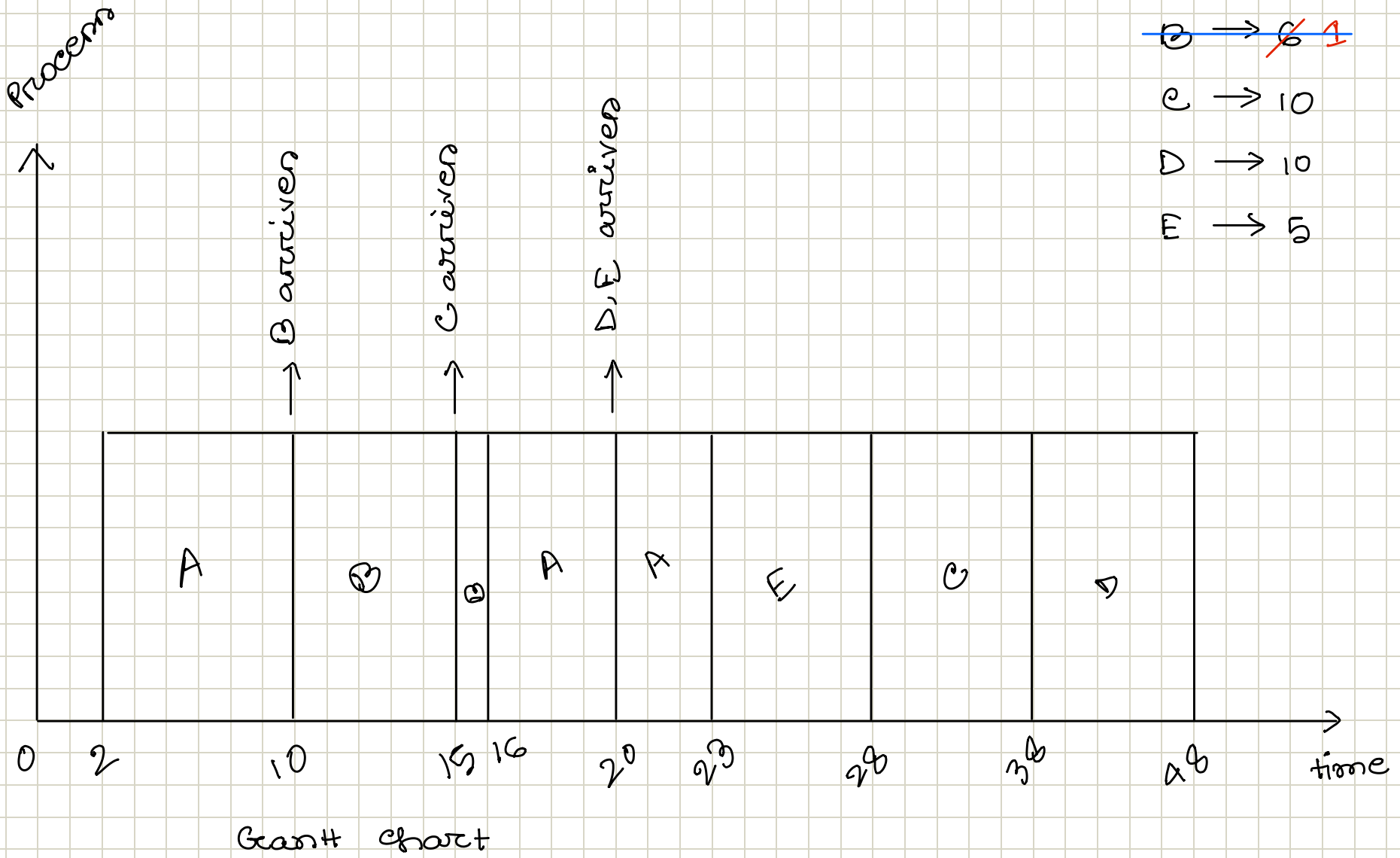
A → ~~7~~ 3

~~B → 6 1~~

C → 10

D → 10

E → 5



Gantt chart

Average turnaround time =

$$\frac{(23-2)+(16-10)+(38-15)+(48-20)+(28-20)}{5}$$

$$= \frac{86}{5} = 17.2 \text{ unit time}$$

Avg. Response time = $\frac{(2-2)+(10-10)+(28-15)+(38-20)+(23-20)}{5}$

$= 6.8$ unit time

Conn: High response time!

SJF (Non-preemptive Algorithm)

- Whenever a process ends, take the smallest burst time duration's job next from the remaining process.

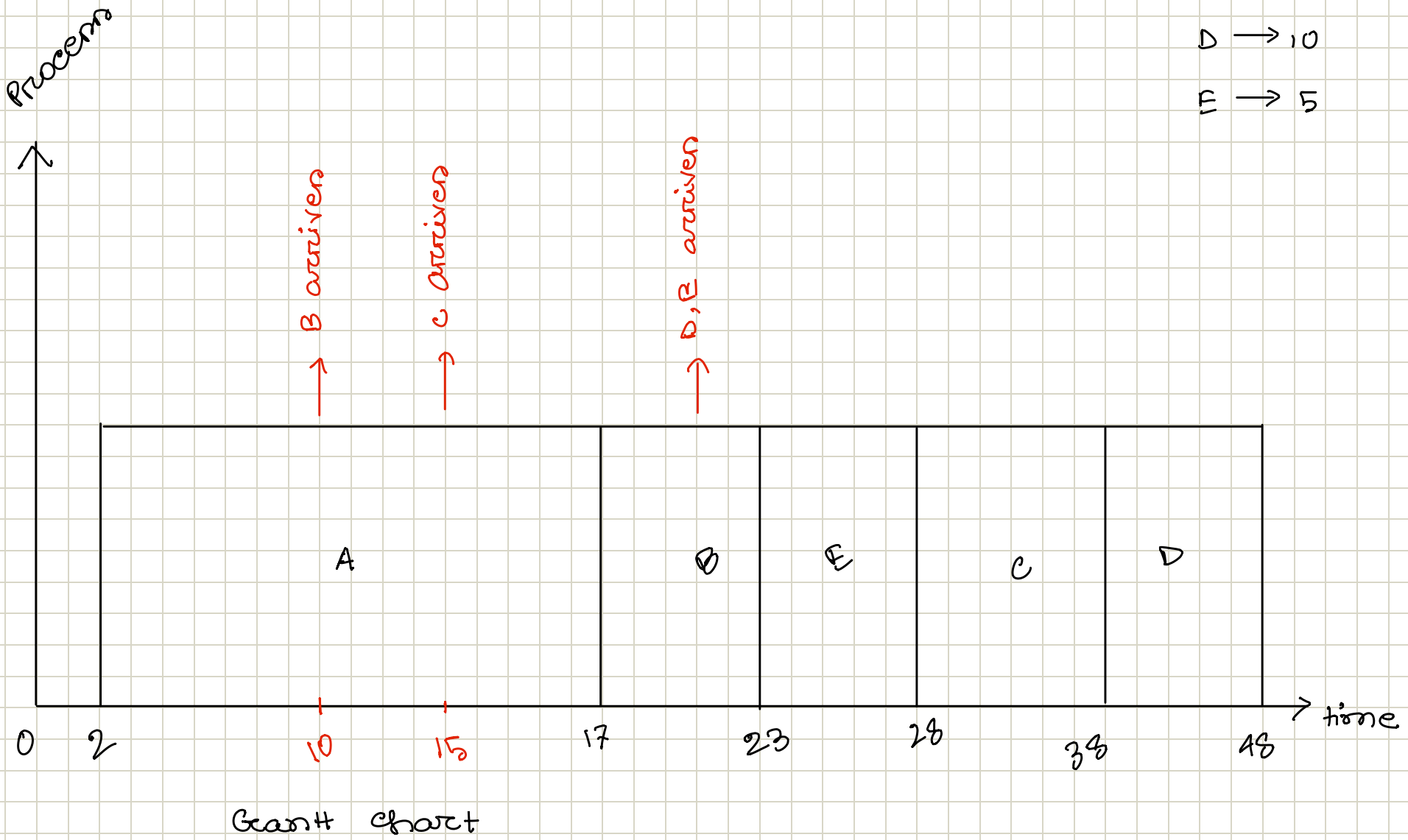
Process	Arrival	Duration / Burst Time
A	2	15
B	10	6
C	15	10
D	20	10
E	20	5

Remaining

C → 10

D → 10

E → 5



Gantt chart

Avg. turnaround time =

$$\frac{(17-2)+(23-10)+(38-15)+(48-20)+(28-20)}{5}$$

= 17.4 unit time

MID Syllabus

(1) Introduction (according to Sumaiya Tabassum Madam's slide)

(2) Process, fork, wait, exec

(3) Scheduling : FIFO, SJF, STCF, RR

(4) IPC : Producer - consumer Problem

CT-02

22 March	8 April
- Scheduling	- Scheduling
- IPC	- IPC
	- Introduction

Time - slice

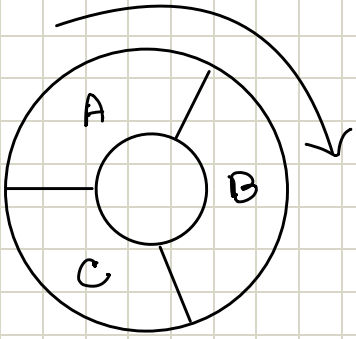
- Smallest unit of time where a periodic interrupt is provided to the operating system.
- Round Robin algorithm works on the basis of time - slice.

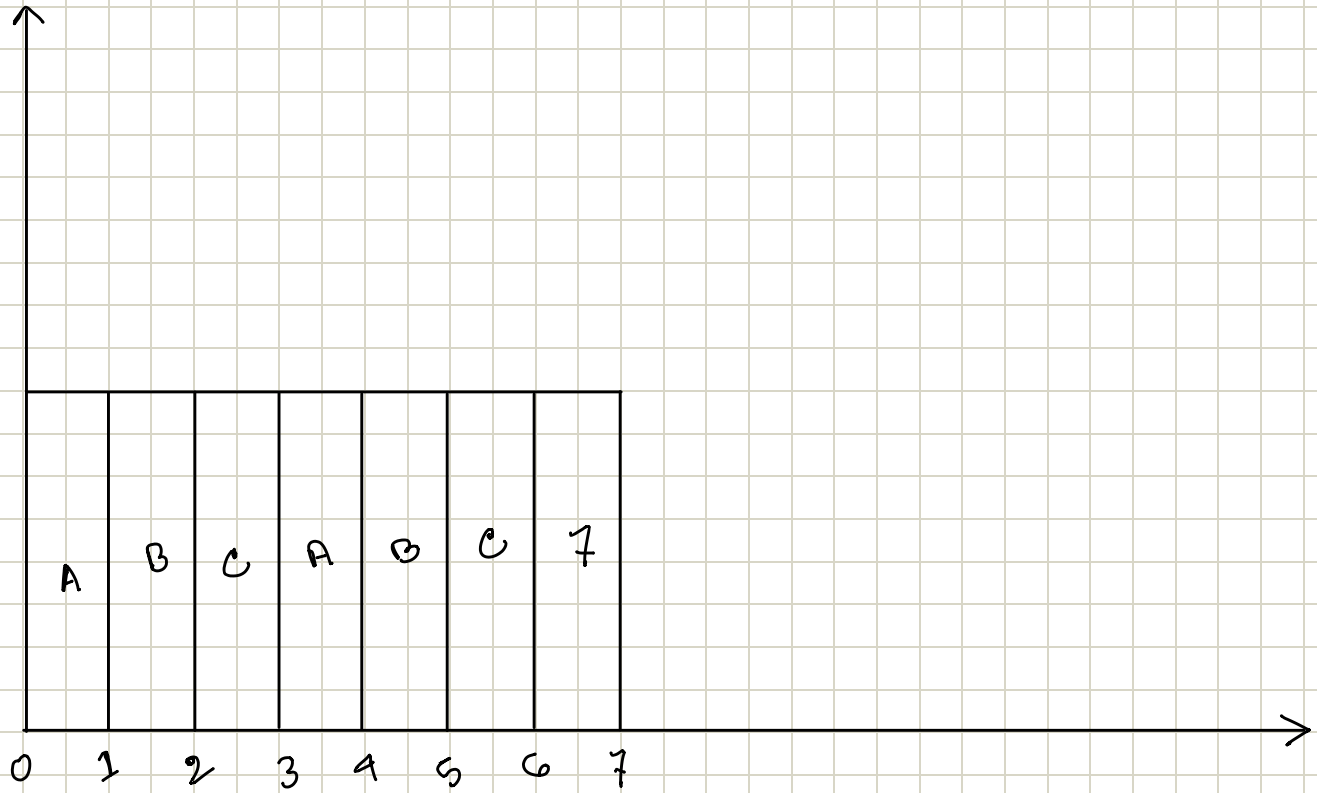
Round Robin

- Focus on response time
- It allows to run a process for a fixed time - slice at each turn.
- Maintains a queue (circular) to keep track of ready processes.

Process	Arrival	Duration (ms)
A	0	2
B	0	3
C	0	2

time - slice = 1ms





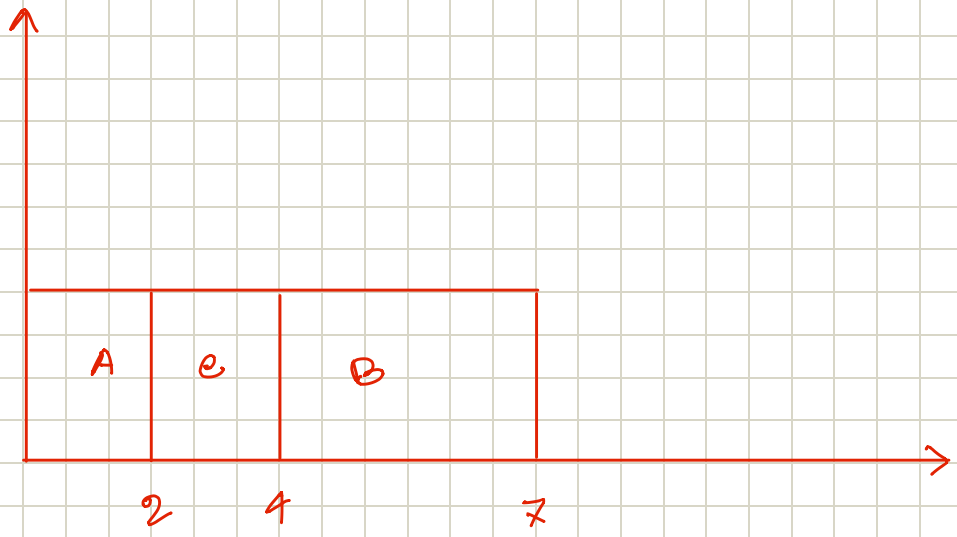
$$\text{Avg. response time} = \frac{(0-0)+(1-0)+(2-0)}{3}$$

$$= 1 \text{ ms}$$

$$\text{Avg. turnaround time} = \frac{(4-0)+(7-0)+(6-0)}{3}$$

$$= \frac{17}{3} = 5.67 \text{ ms}$$

SJF



Avg. t. time $= \frac{2+3+7}{3}$

$= {}^{13}/_{3} = 4.33$ (less & better than Round Robin)

Avg. rt. time $= \frac{0+2+4}{3}$

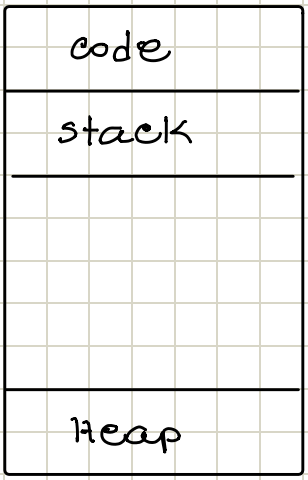
$= 2$ ms (2x than Round Robin)

Concurrency and Inter Process Communication (IPC)

- We can execute a tasks in parallel to reduce the run-time.
  For example, searching, matrix addition etc.
- For this, concept of "thread" is introduced
  - → similar to a process
  - → Share the same memory space
  - → Additionally, has its own id, stacks, registers.

Parallel Matrix Addition:

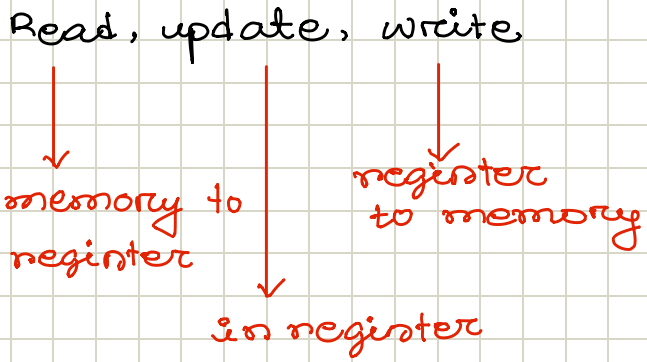
$$\begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 4 & 2 \end{bmatrix}$$



→ shared by all threads

→ Each thread has it's own stack

→ shared by all threads



Mutual Exclusion

- When one thread is in critical-section (using shared data), other threads must wait until that thread exits from critical section. For this purpose, we use lock (also known as mutex, semaphore).

```
lock();
updateSharedData();
unlock();
```

For, Inter Process communication → We need conditional variables (C.V), along with locks.

Purpose of C.V:

- Wait on some condition
- Give signal to one or all waiting threads.

Goes to Blocked State

Producer / Consumer Problem

Producer ⟷ shared space ⟷ consumer

(Mutually exclusive access)

Types of thread:

(1) Producer thread

(2) Consumer thread

- Producer produces when buffer has space
- Consumer consumes when buffer is non-empty

Thread 1 State:

lock variable

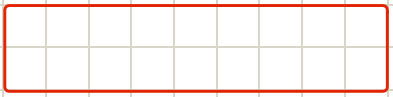
POSIX Thread

```
Mutex_lock(&m);
if(numfull == max) {
```

```
    cond_wait (&cond, &m);
}
cond_signal (&cond);
do_fill (i); // critical section
Mutex_unlock (&m);
```

numfull → currently filled space

Buffer's max size → max



Buffer

Thread 2 State :

```
void * consumer () {
  for (int i=0; i<loops; i++) {
    Mutex_lock (&m);
    if (numfull == 0    ) {
        cond_wait (&cond, &m);
    }
    cond_signal (&cond);
    int tmp = do_get (i); // critical section
    Mutex_unlock (&m);
  }
}
```

- Critical section
- Mutual exclusion
- Synchronization (wait, signal)

2 consumer, 1 producer

- Required two different CVs (conditional variables)

2 consumer, 2 Producer

- Required two unique CVs (conditional variables)

Summary:

(1) Use different conditional variables for different types of thread.

(2) Always check the conditions in while loop