Introduction to Concurrency

1. Concurrency

Concurrency refers to the execution of **multiple instruction sequences in an overlapping time period**.
In an operating system, concurrency occurs when **multiple processes or threads are in progress at the same time**, allowing better utilization of system resources.

Note: Concurrency does not necessarily mean simultaneous execution. Simultaneous execution is called **parallelism**.

---

2. Thread

A **thread** is:

- A **single sequence stream of execution** within a process
- An **independent path of execution** inside a process
- A **light-weight process**
- A mechanism to achieve concurrency by dividing a process into **independent execution paths**

Examples:

- Multiple tabs running in a web browser
- In a text editor, typing, spell checking, formatting, and auto-saving occur concurrently using multiple threads

---

3. Thread Scheduling

- Threads are scheduled for execution based on **priority and scheduling policy**.
- The **operating system scheduler** assigns CPU time slices to threads.
- Even though threads belong to a process, **CPU allocation is controlled by the OS**, not the runtime environment.

---

4. Thread Context Switching

Thread context switching occurs when the OS switches the CPU from one thread to another **within the same process**.

During thread context switching:

- The **memory address space is not switched** (shared among threads)
- The following are switched:
    - Program Counter (PC)

    - CPU registers
    - Stack
- Thread switching is **faster than process context switching**
- Cache performance may improve due to locality, but **cache contents are not guaranteed to be preserved**

---

5. How Does Each Thread Get Access to the CPU?

- Each thread has its **own Program Counter**
- Based on the **thread scheduling algorithm**, the OS selects a thread for execution
- The OS fetches instructions using the **PC of the selected thread** and executes them on the CPU

---

6. I/O or Time-Quantum Based Context Switching

- Context switching can occur due to:
    - I/O operations
    - Expiry of time quantum
- Each thread has a **Thread Control Block (TCB)**:
    - Similar to a Process Control Block (PCB)
    - Stores thread state information during context switching

---

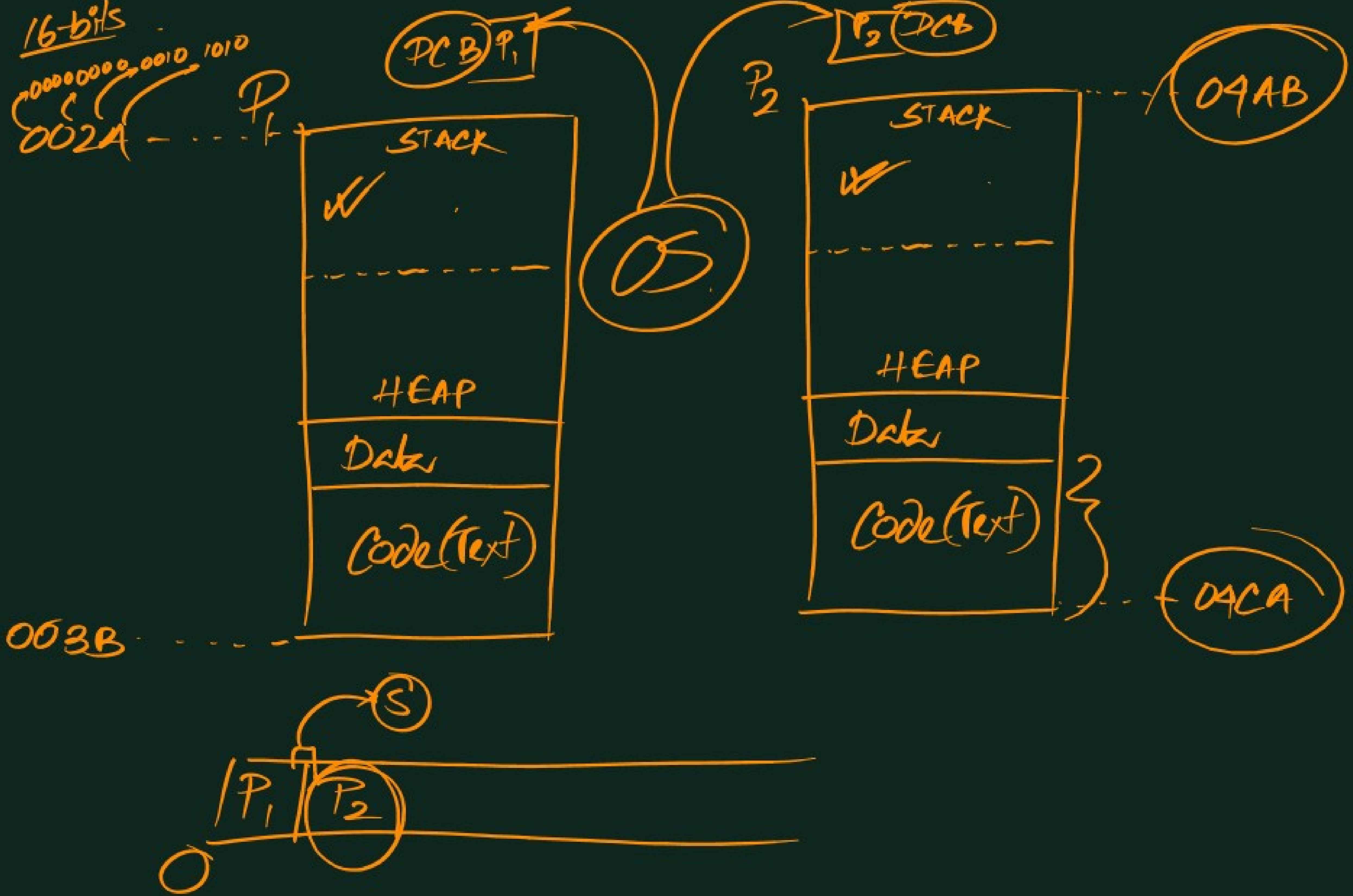
7. Multithreading on a Single-CPU System

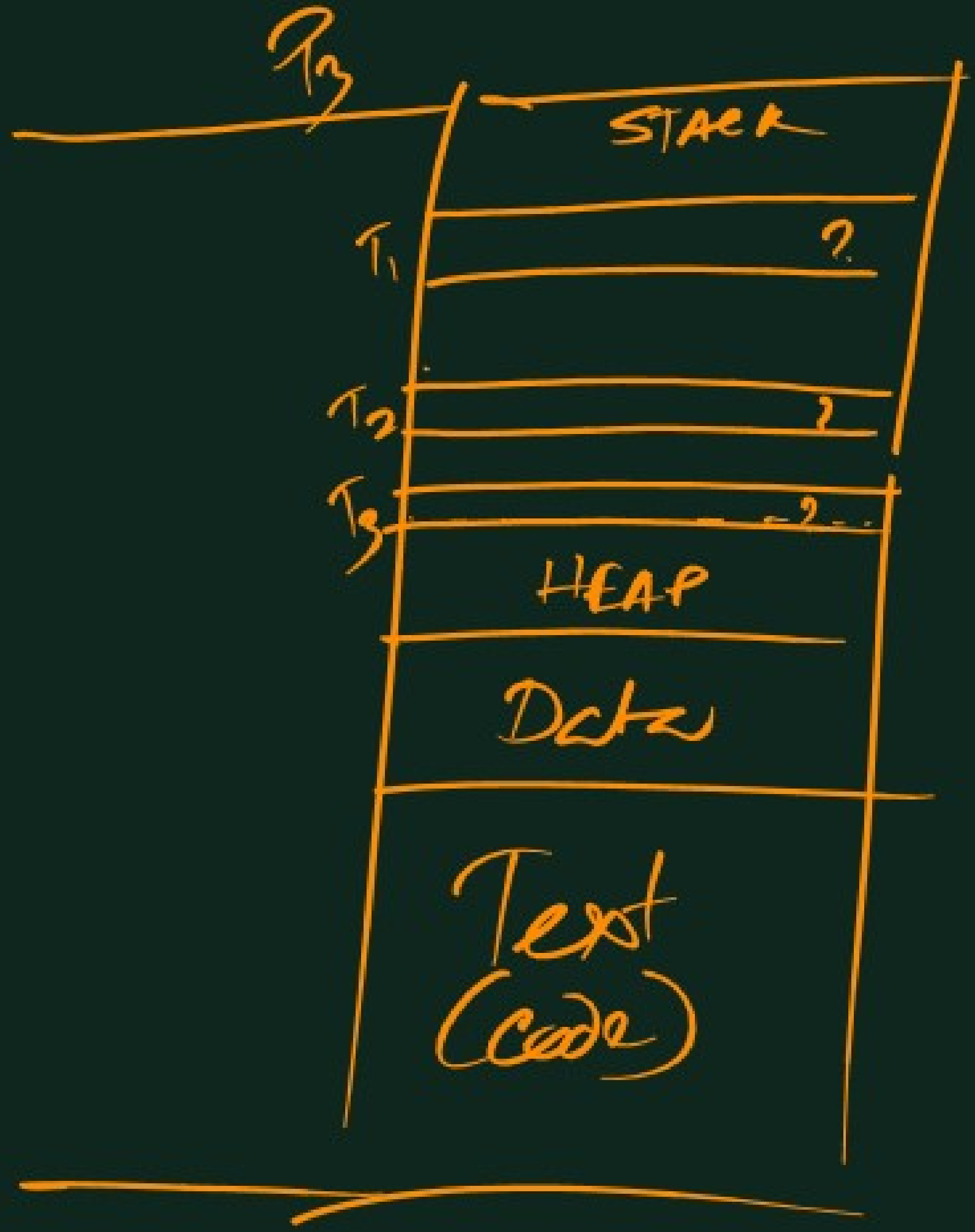
- A single-CPU system **does not gain execution speed or throughput** from multithreading
- Only **one thread executes at a time**
- However, multithreading **improves responsiveness**, especially for I/O-bound applications

---

8. Benefits of Multithreading

- **Responsiveness:** Applications remain active even if one thread is blocked
- **Resource Sharing:** Threads efficiently share process resources
- **Economy:**
    - Thread creation and context switching are cheaper than process creation
    - Process creation requires significant memory and resource allocation
- **Scalability:** Efficient utilization of **multicore and multiprocessor architectures**


16-bits
0000000000101010
002A
003B
P1
STACK
HEAP
Data
Code(Text)
PCB P1
OS
P2 PCB
P2
STACK
HEAP
Data
Code(Text)
04AB
04CA
S
P1
P2
0



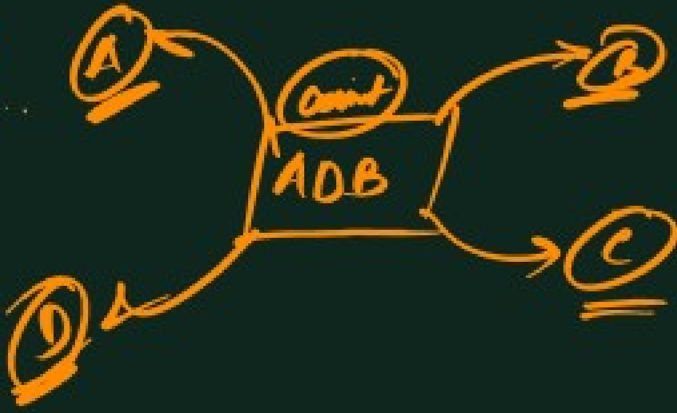
$P_1 : T_1$

$P_1 : T_2$

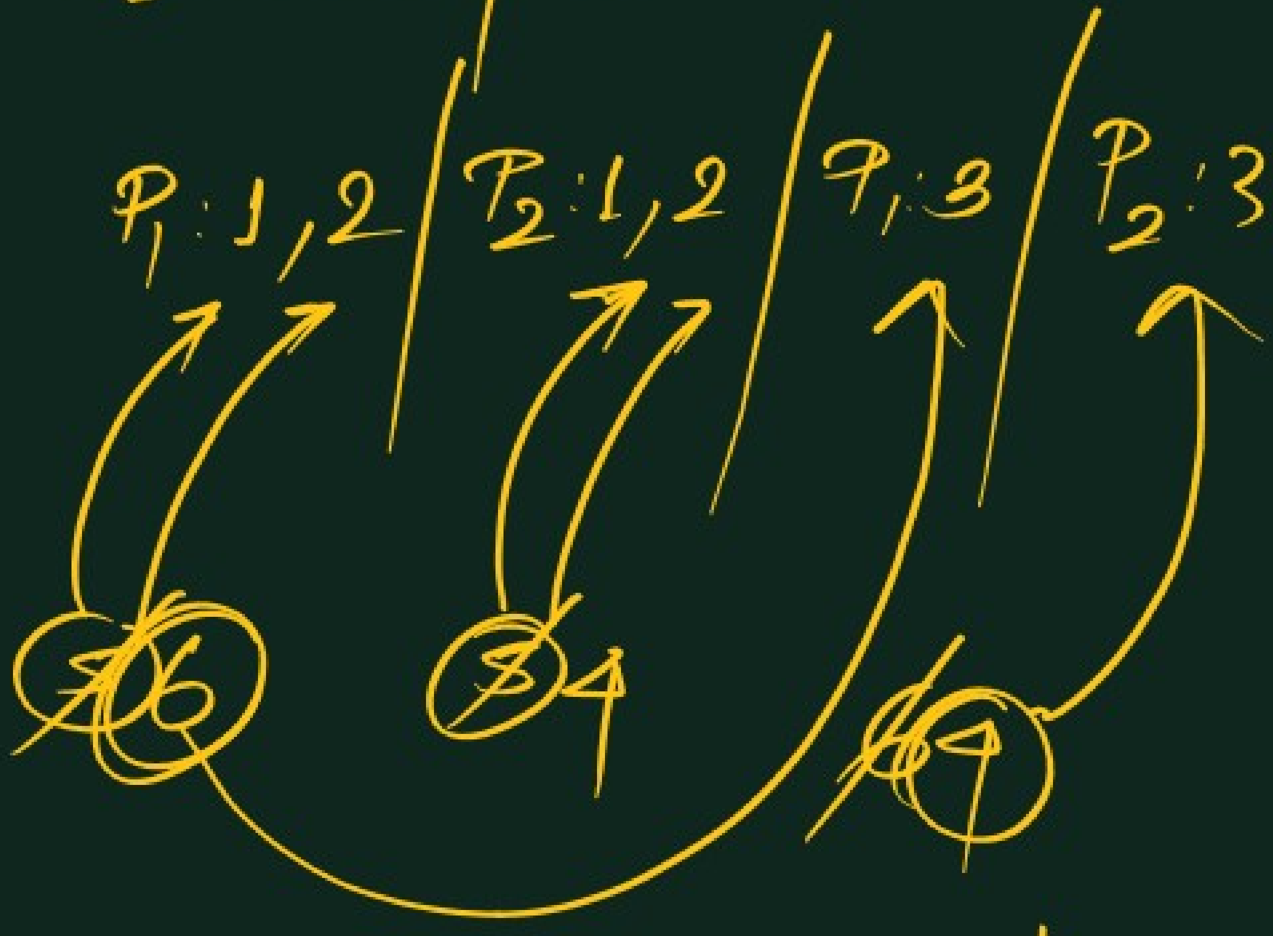
$P_1 : T_3$

Count [~~5~~] ~~6~~ 5

Synchronization



P_1 : 1 2 3 | P_2 : 1 2 3



P_1 : 1, 2 | P_2 : 1, 2, 3 | P_1 : 3

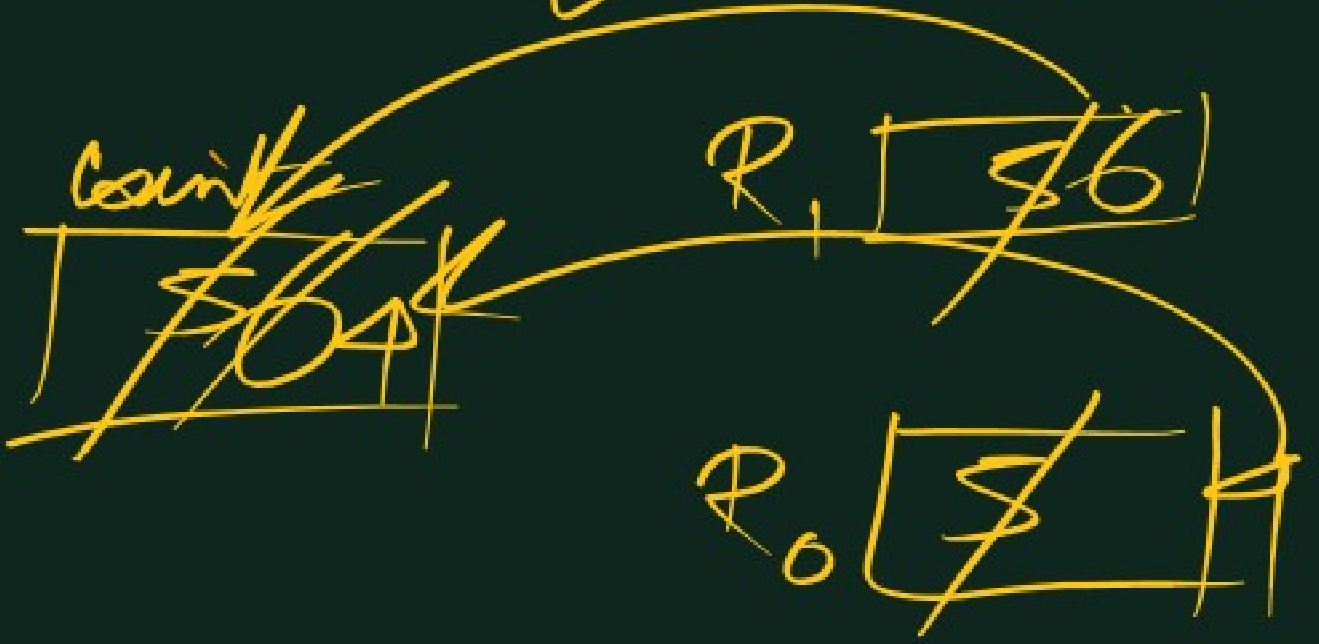
```
P1                  P2
{                   {
  - - - -             - - - -
  - - - -             - - - -
  count++;            Count--;
  - - - -             - - - -
}                   }
```

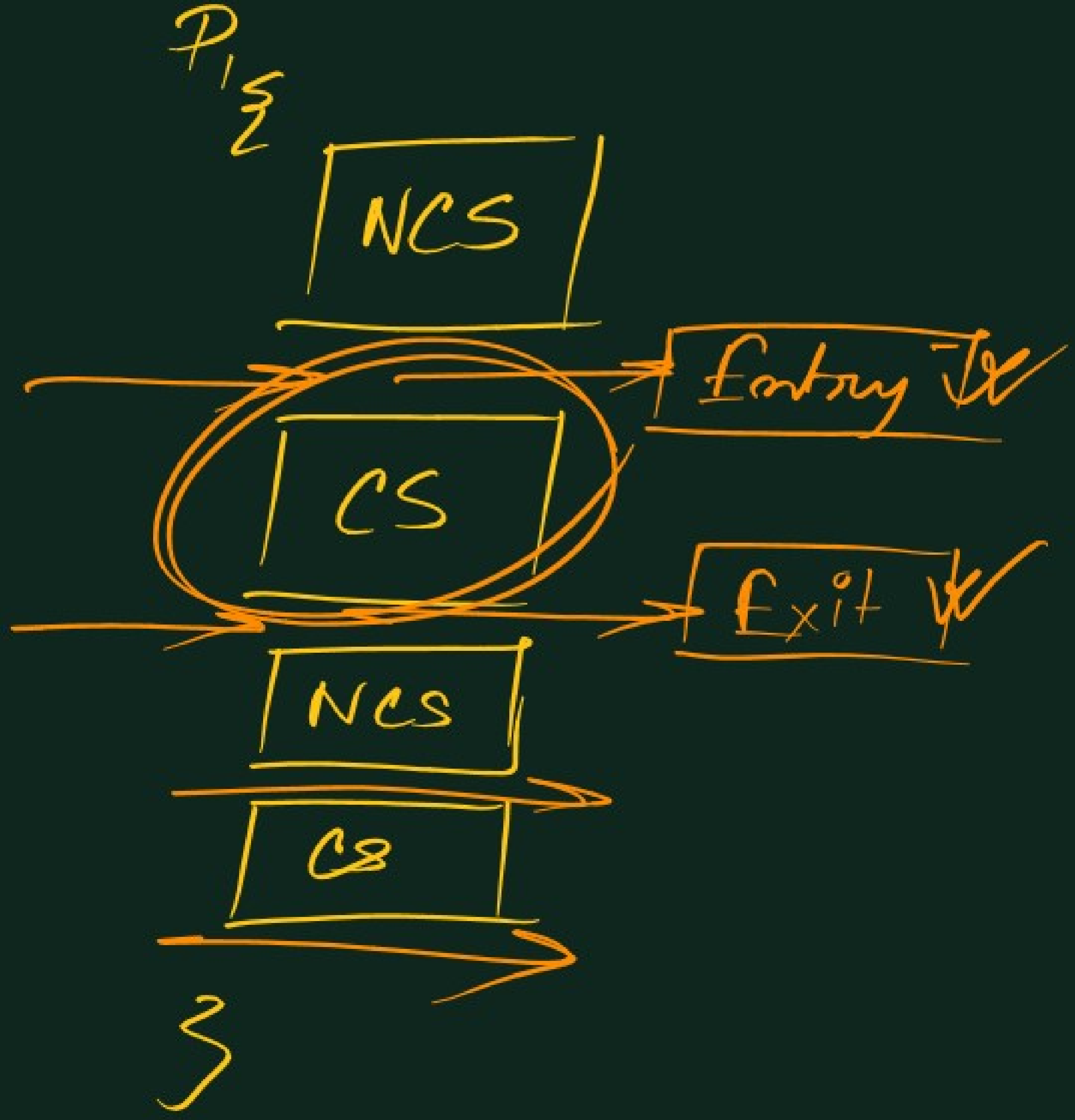
Count [5]

Assembly Language

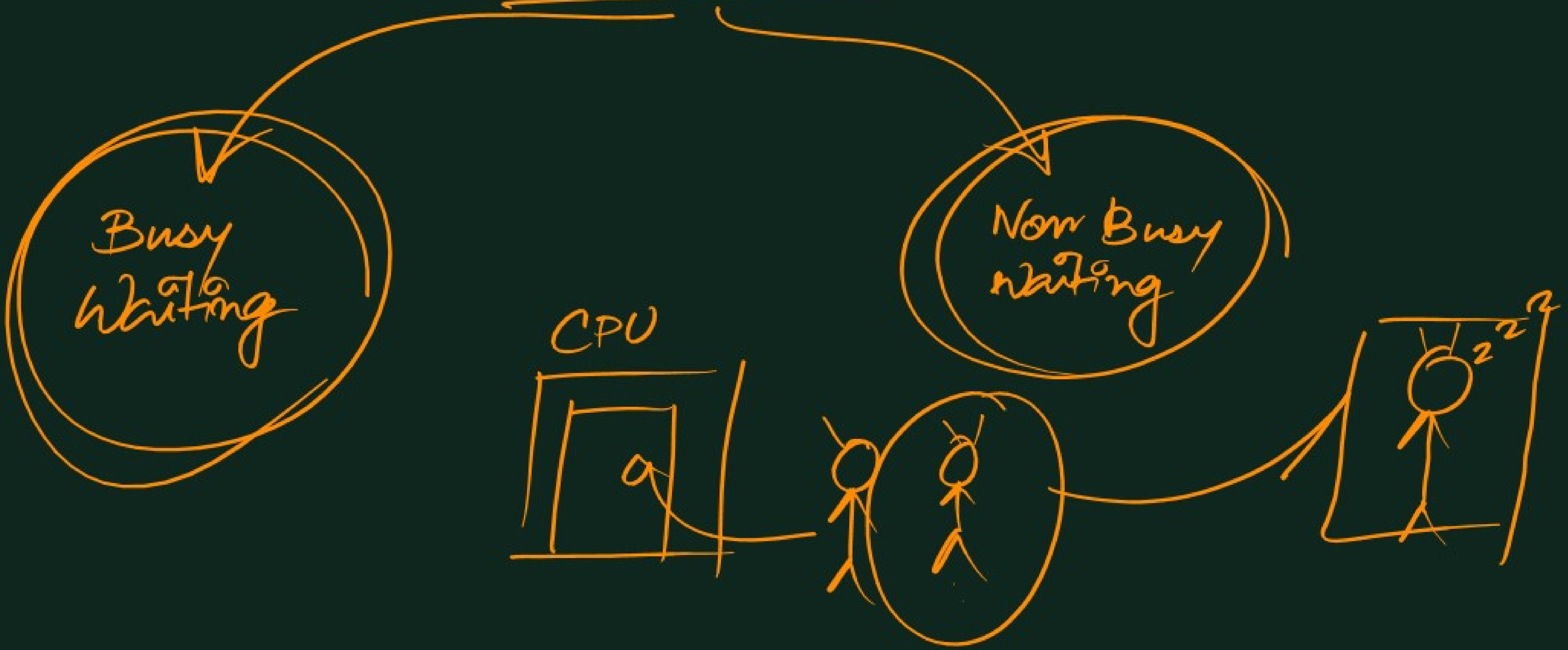
P_1
1) MOV R_1, Count
2) INR R_1
3) MOV Count, R_1

P_2 count--
1) MOV R_0, Count
2) DCR R_0
3) MOV Count, R_0




P1 {
NCS
CS
Entry ✓
Exit ✓
NCS
CS
}

Synchronization Mechanism



(I) Mutual exclusion
(II) Progress

Primary

(III) Bounded waiting *
(IV) Portability / Architectural Neutrality.

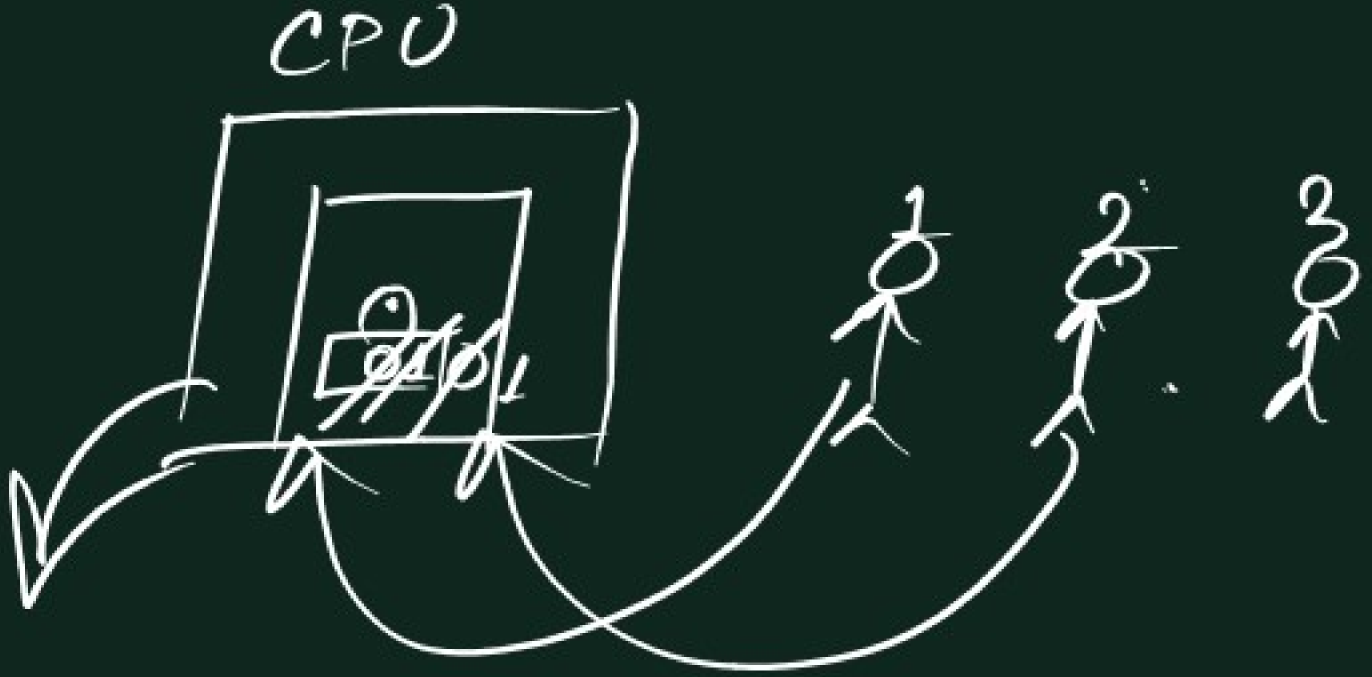
Optional

Criteria for Process synchronization Mechanism.

Entry
```
while (LOCK != 0);
Lock = 1;
```

```
count++;
```
CS

Exit
```
Lock = 0;
```



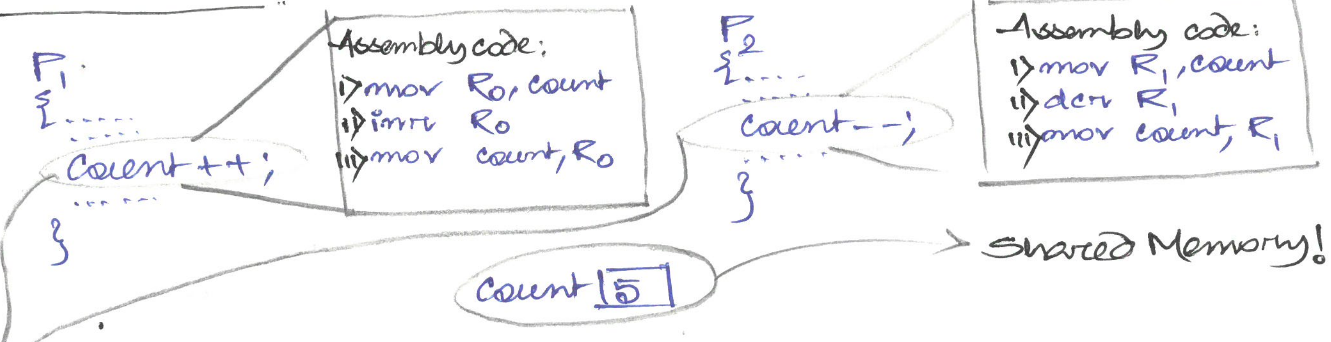
Process Synchronization

RBR
08-9✓

In CN, interprocess communication occurs in between 2 processes residing in 2 different Computers. There the commu--nication is handled by protocol stacks (i.e. ISO-OSI, TCP/IP etc.)

In OS, interprocess communication occurs in between 2 processes residing in the same computer. It happens when 2 or more processes try to compete for some resource or they co-operate eachother to complete some particular task.

Illustration:



```
Assembly code:
i) mov  R0, count
ii) incr  R0
iii) mov  count, R0
```

```
Assembly code:
i) mov R1, count
ii) decr R1
iii) mov count, R1
```

3 situations of execution:

```
P1: I, II, III | P2: I, II, III                      Count = 5
P1: I, II | P2: I, II, III | P1: III                 Count = 6
P1: I, II | P2: I, II | P1: III | P2: III            Count = 4
```

Race condition: The order of exe--cution of instructi--ons defines the results produced.

| → Preemption

Critical section:
It is the part of the program where shared resources are accessed by processes.

As a solution to Race condition, we modify our code segment by introducing an entry section (before the Critical Section) & an exit section (after the critical section) for every critical section in our code. This is known as Synchronization Mechanism.

- **Requirements of Synchronization Mechanisms:**
(A.K.A. Criteria of solving critical section problem)
  - **Primary:** ① Mutual exclusion: Treat the critical section as dressing room, i.e. only one at a time.
    ② Progress: Dog (which is not interested in eating grass) should not block the horse to the room full of grass (Critical section).
  - **Secondary:** ① Bounded waiting: The waiting time for a process before it goes into Critical section should be limited.
    ② Portability / Architectural neutrality.

Synchronization Mechanisms

With Busy waiting

∴ A process, seeing another process accessing the criticalsection will stay busy waiting for the process to come out of the criticalsection so that it can enter.

Without Busy waiting Better

∴ A process, seeing that the criticalsection is preoccupied will not waste CPU time by waiting, rather it'll go and sleep. Then again it will resume and enter the critical section once the previous process gets out of the criticalsection.

• Lock variable:

- Software mechanism implemented in user mode.
- Busy waiting solution.
- can be used even for more than two processes.

Synchronization Mechanism

entry section

```
1. while (LOCK != 0);
2. LOCK = 1
```

Critical Section

exit section

```
LOCK = 0
```

Lock = 0 – CS vacant
= 1 – CS occupied

• Kernelmode: Support needed from OS will have to be taken from OS code, a.k.a. Kernelmode.

• Usermode: Independent user code which doesn't need any support from OS.