CS & IT ENGINEERING

Operating Systems

Process Synchronization/Co-ordination

Concurrency Mechanisms

Lecture No. 11

By- Dr. Khaleel Khan sir

- Concurrency Conditions
- Concurrency Vs Paralellism
- Concurrency Constructs

Q.

First Reader-Writer using Semaphore with Busy Waiting

```
int R = 0, W = 0 ;
BSEM mutex = 1 ;
Void Reader (Void)
{
 L1: P(mutex);
 if (W == 1)
 {
   V(mutex);
   goto L1;
 }
 else
 {
    R = R+1;
   V(mutex);
 }
 <DB_READ>
 P(mutex);
 R = R –1;
 V(mutex);
}
```

```
Void Writer ( Void)
{
 L2: P(mutex);
  if(_____)
  {
     V(mutex);
     goto L2;
  }
  else
  {
     W=1;
     V(mutex);
  }
<DB_WRITE>
P(mutex);
  W=0;
V(mutex);
}
```

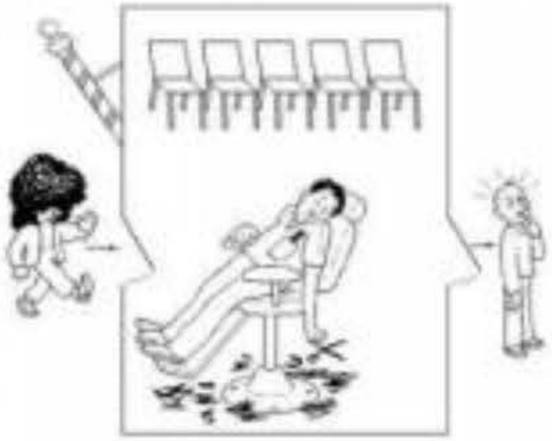
B/w

When Should writer Busy wait ?

(R ≥ 1 or W == 1)

Sleeping Barber Problem

Exercise



- There is one barber, and n chairs for waiting customers
- If there are no customers, then the barber sits in his chair and sleeps (as illustrated in the picture)
- When a new customer arrives and the barber is sleeping, then he will wakeup the barber
- When a new customer arrives, and the barber is busy, then he will sit on the chairs if there is any available, otherwise (when all the chairs are full) he will leave.

Tanenbaum

Barber Shop Simulation


Barber Shop
Waiting Room
Barber has a chair to cut hair of customer. When barber is done with customer then he will cut hair of next customer from waiting room.


Barber Shop
Waiting Room
z
Z
z

Concurrency vs Sequentiality

S_i : High Level Lang. Stmnt

```
begin {
    S1: a = b + c;
    S2: d = e * f;
    S3: k = a + d;
    S4: l = k * 10;
end }
```

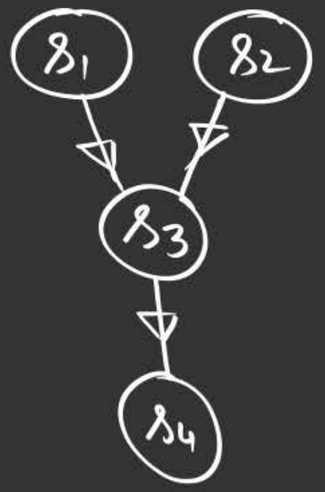
Stmnt

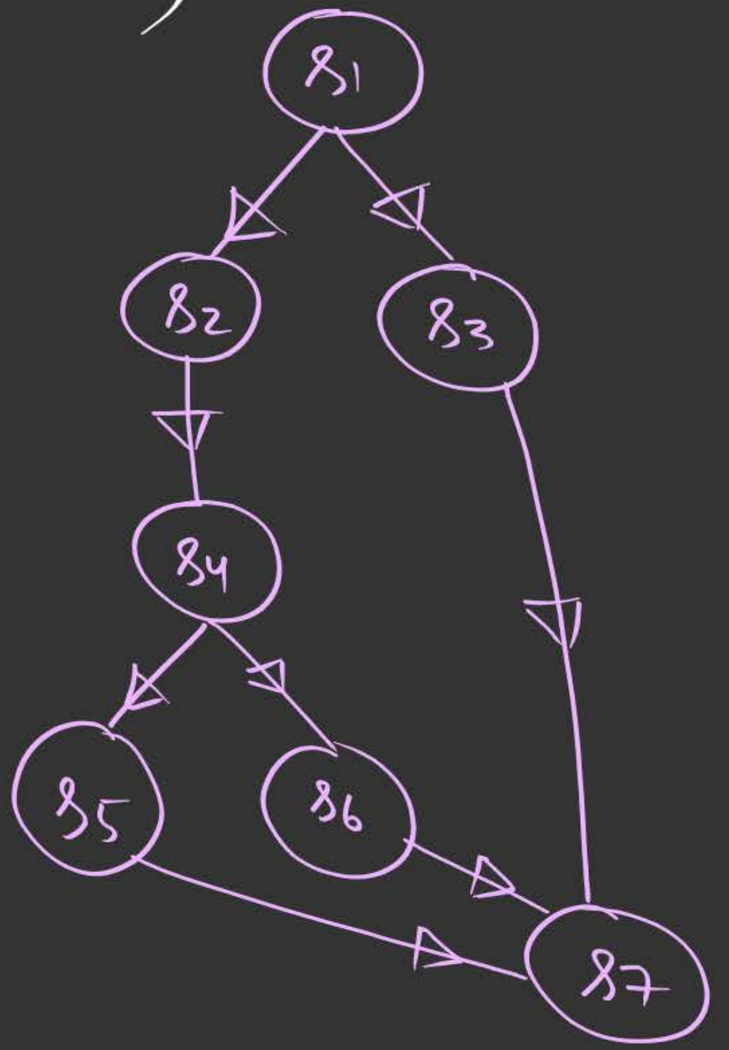
I_1	Load R1, b
I_2	Load R2, c
I_3	Add R1, R2
I_4	Store a, R1

only S_1 S_2 : Concurrently (No dependency)

Precedence Graph (directed)

$G = (V, E)$





Concurrency Vs Parallelism

A System is said to be ***Concurrent*** if it can support Two or more actions in ***Progress*** at the same time

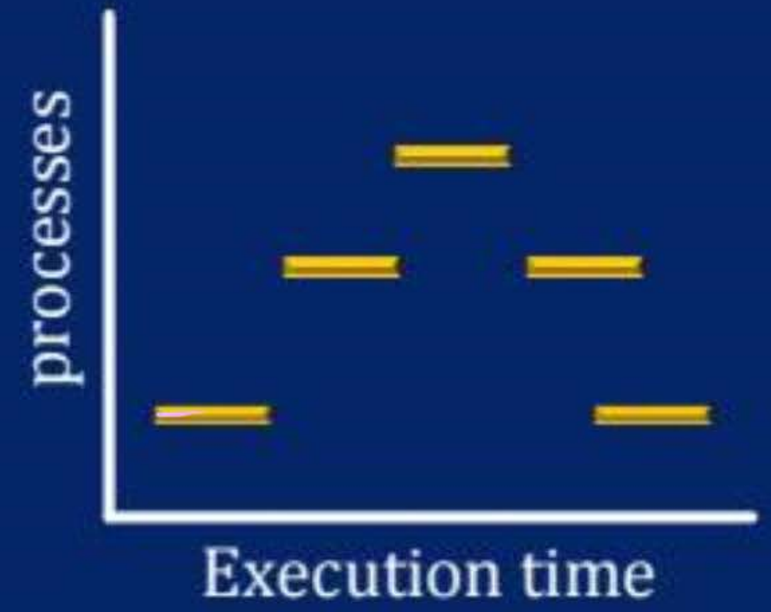
A system is said to be ***Parallel*** if it can support Two or more actions Executing ***simultaneously***

Concurrency is about dealing with lots of things at once.

Parallelism is about doing Lots of things at once.

Concurrency Vs Parallelism

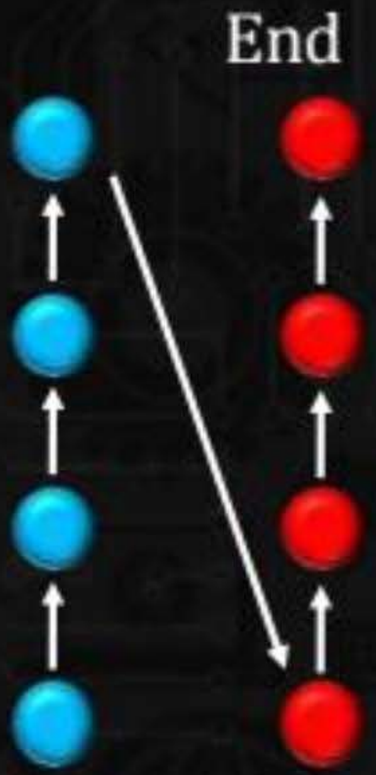
CONCURRENCY



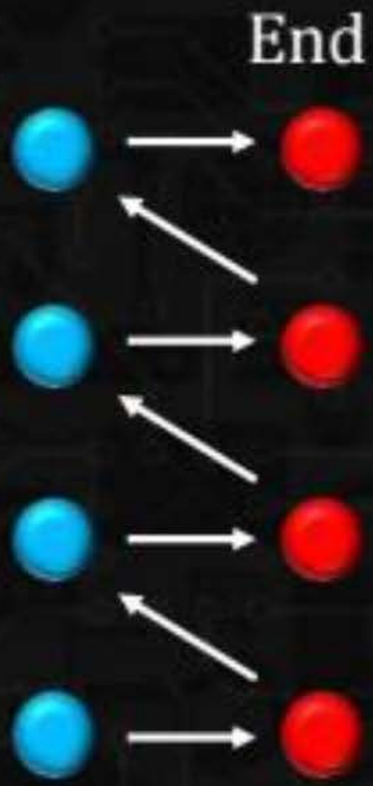
VS

PARALLELISM

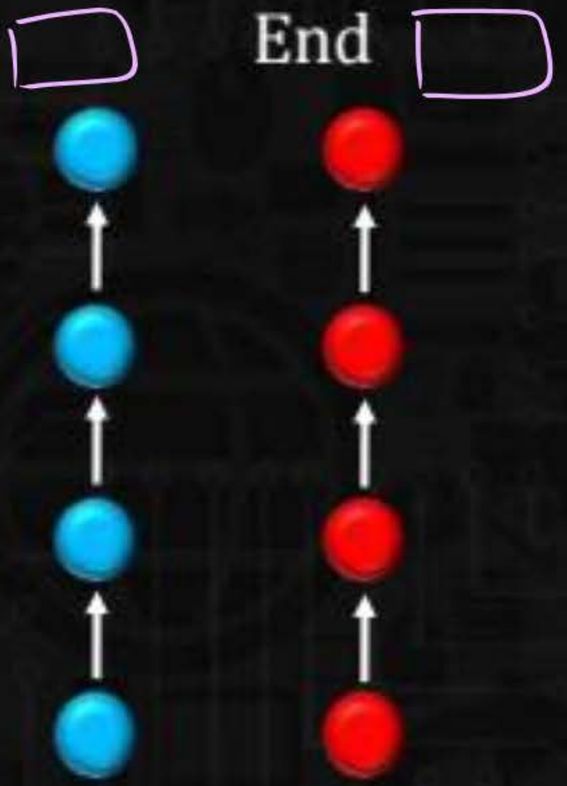
Sequential



Concurrent



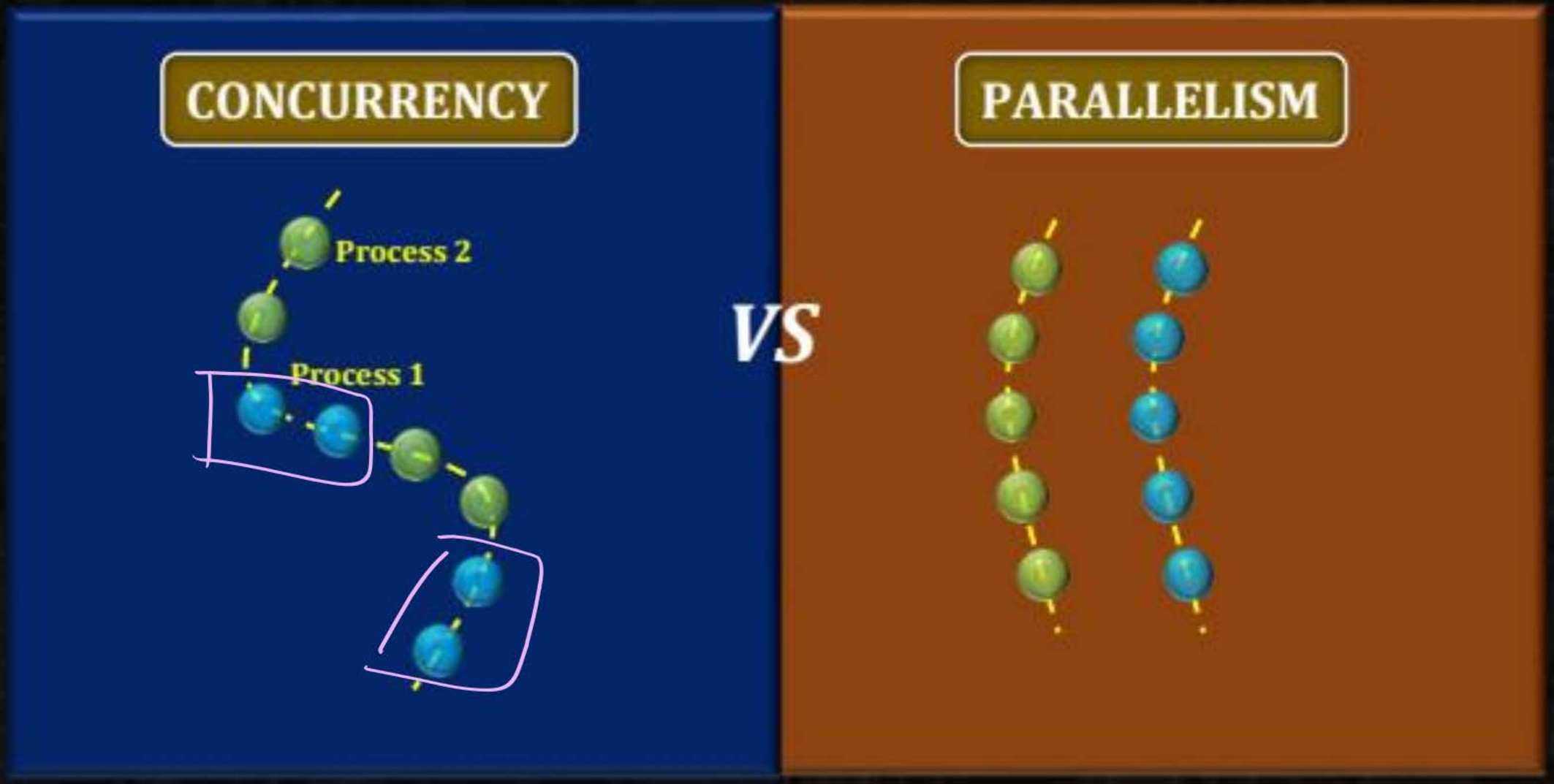
Parallel



"Concurrency is about dealing with lots of things at once.

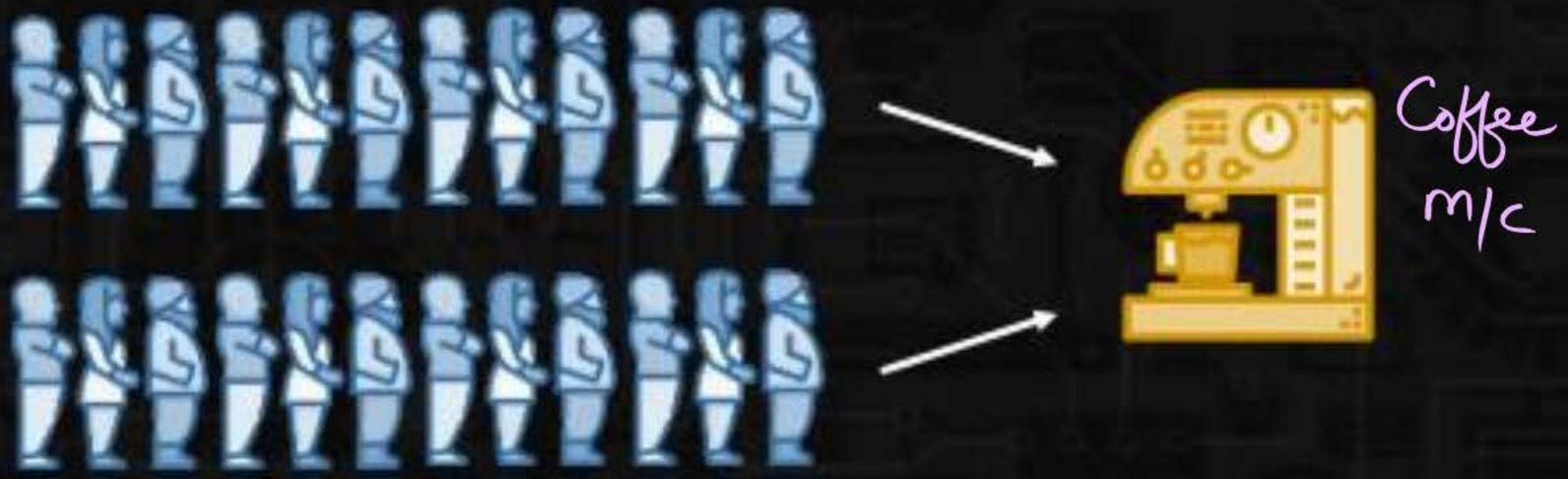
Parallelism is about doing things at once" - Rob Pike

Concurrency is not parallelism

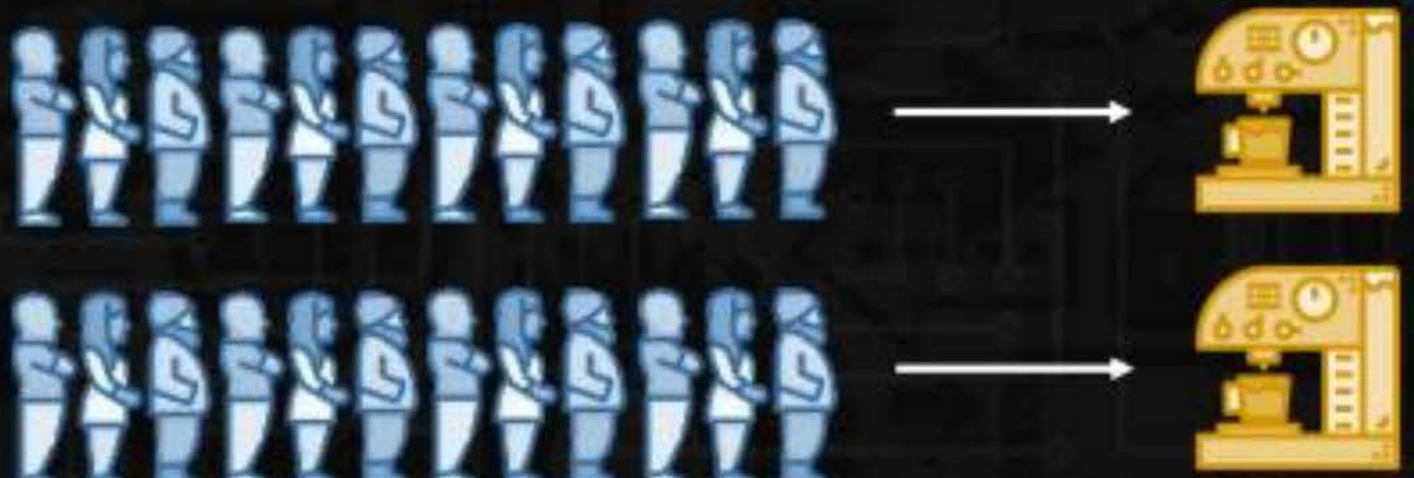


An analogy

Concurrent = Two queues one coffee machine



Parallel = Two Queues two coffee machine



Joe Armstrong 2013

Independent

P1 P2 P3

R.R

CPU

Concurrency

(Process will execute with Pre Emptions)

- Interleaved -

Parallelism is possible with Multiple CPU's / Cores

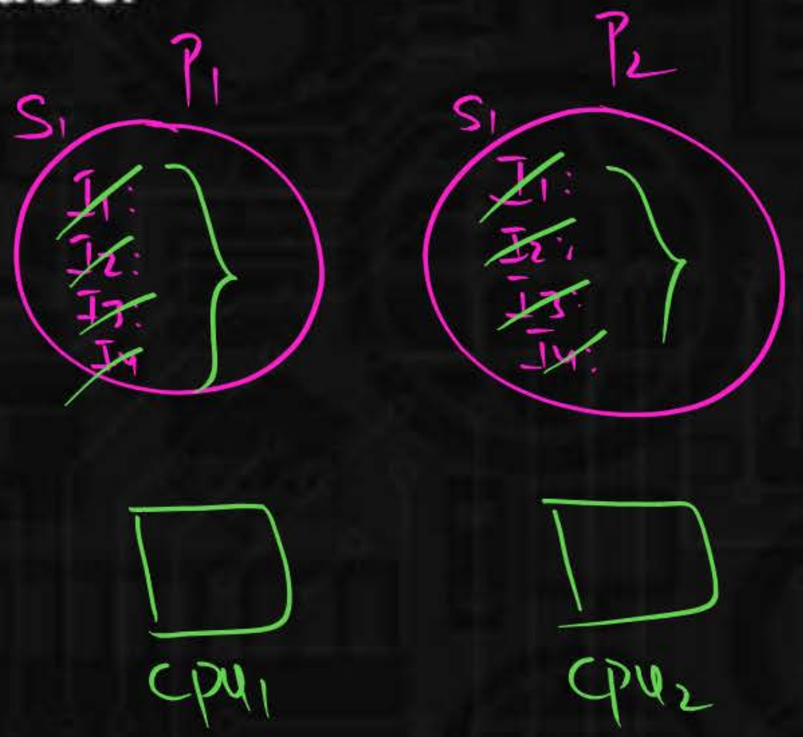
"Concurrency is about structure, Parallelism is about execution"

Concurrency provides a way to structure a solution to solve a problem that may (but not necessarily) be parallelizable.

The modern world is parallel It has:

- Multicores
- Networks
- Clouds of CPUs
- Loads of users

Parallelism



Concurrency makes Parallelism easy.

Types of Concurrency

- Pseudo (one CPU)
  - Interleaved execution among Processes / applications
- Real / Physical (Parallelism)
  - Multiple processing units

Concurrency Conditions

S_i : a = b + c; S_j : d = e * f;

S < R(S): Read Set
    W(S): write Set

2000 | 10 | x

int x = 10;

S: printf("%d", x);

read-set → 10

2000 | ~~5~~ 100 | x

3) int ...

S: Read(x);

R(S): ∅
W(S): {x}

S_i: a = b + c;
S_j: d = b * c;

o/p of one stmnt should not serve as Input to other

1) S: a = b + c
   15 20
   15 | b

2) S: a += ++b * --c;

R() = } ⟨a, b, c⟩ ✓
W() = }

S_i < $R(S_i)$, $W(S_i)$: S_j < $R(S_j)$, $W(S_j)$

Bernstein's Conc. Conditions

Reader-writer

I. $R(S_i) \cap W(S_j) = \emptyset$ (NULL)

II. $W(S_i) \cap R(S_j) = \emptyset$

III. $W(S_i) \cap W(S_j) = \emptyset$

IV. $R(S_i) \cap R(S_j)$ = May or May NOT be $\emptyset$

S_i : a = (b + c) ;

S_j : a = (k * m) ;

a

Inconsistency

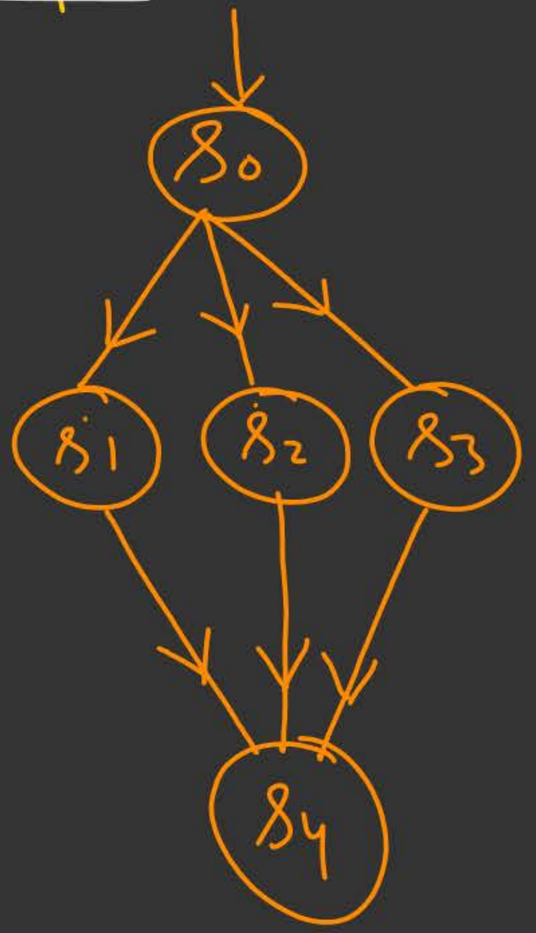
Parallel Concurrency Mechanisms/Constructs

1. Parbegin - Parend / Cobegin - Coend :

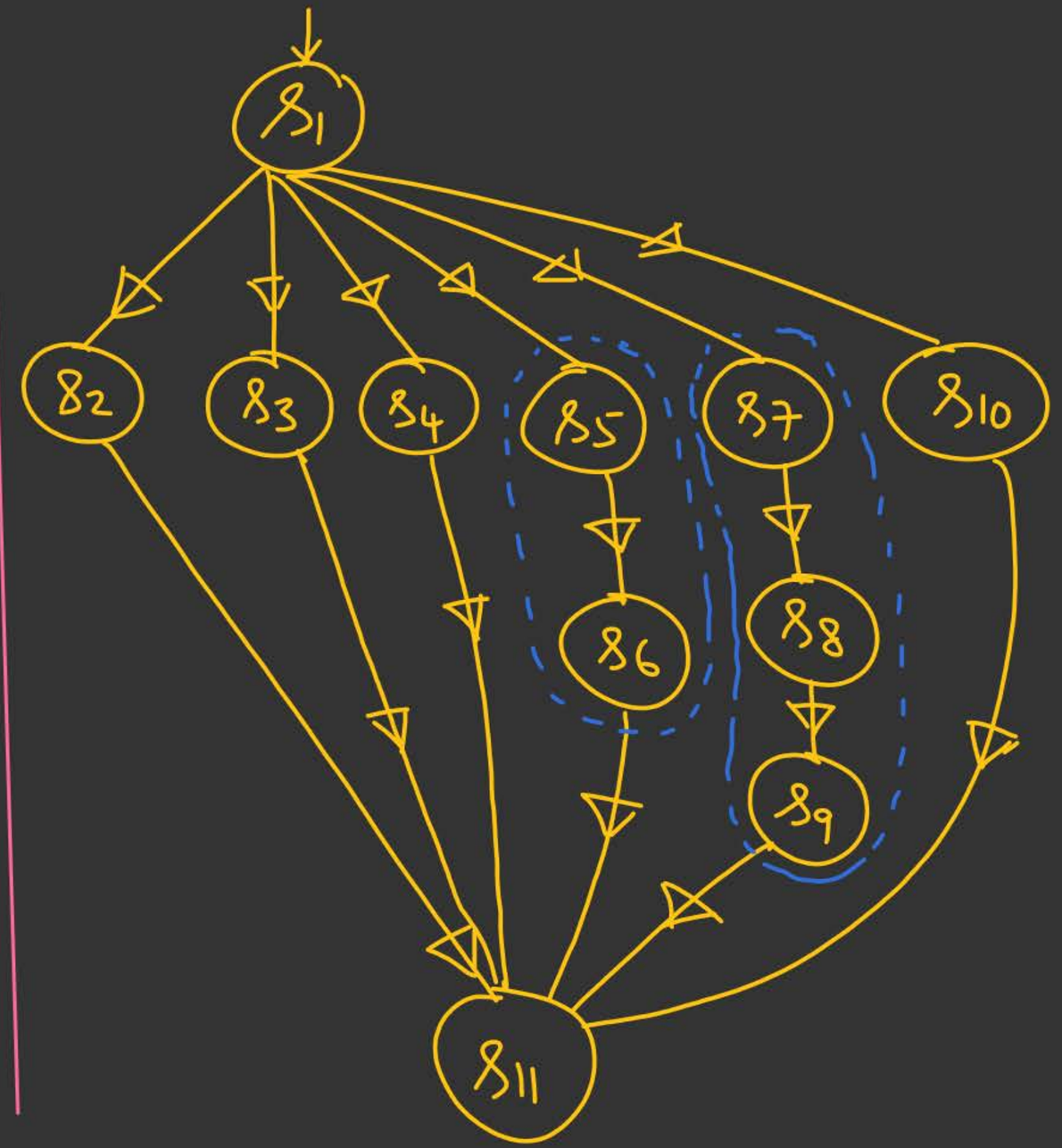
Sequential

```
{ begin
    S1;
    S2;
    S3;
} end
```

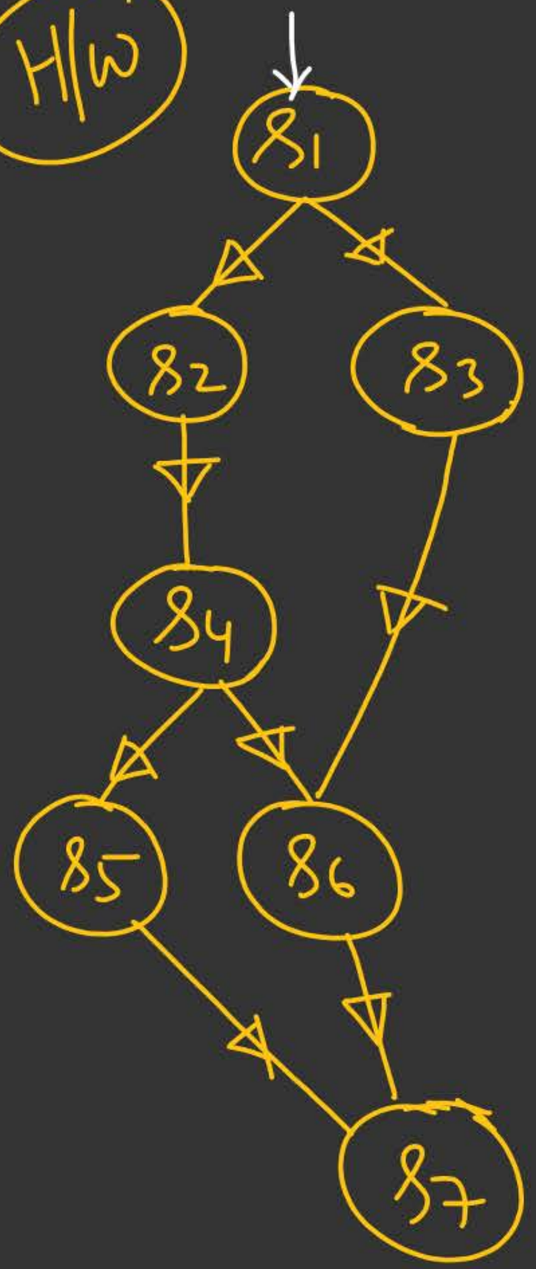

```
S0;
Parbegin
    S1;
    S2;
    S3;
Parend
S4;
```



```
S1;
Parbegin
    • S2; S3;
    • S4;
    • begin S5; S6; end
    • begin S7; S8; S9; end
    • S10;
Parend
    S11;
```



H/W

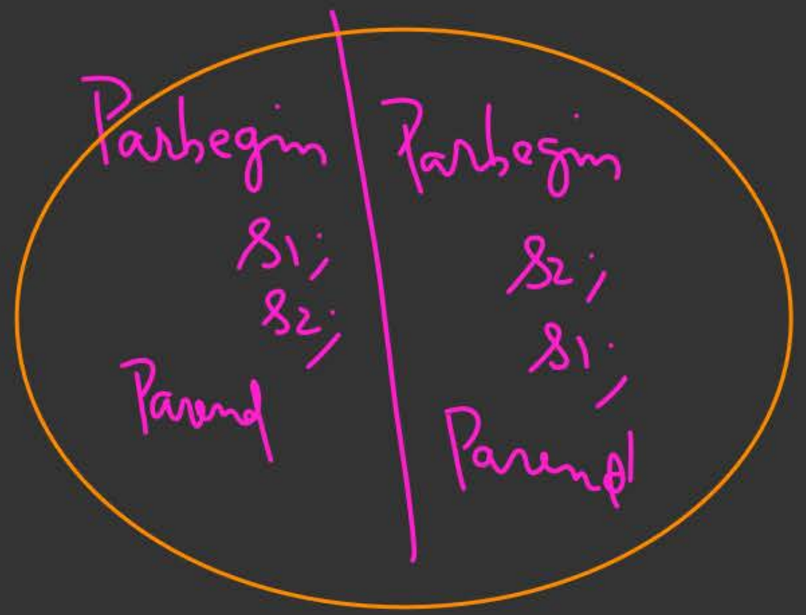



```
S1;
Parbegin
    S3;
    begin
        S2;
        S4;
        Parbegin
            S5;
            S6;
        Parend
    end
    S5;
Parend
S7;
```

Q) How many times Parallel Construct to be used ?

a) 1 b) 2

c) 3 d) None



THANK YOU!