CS & IT ENGINEERING

Operating Systems

Process Synchronization/Coordination

Concurrency Mechanisms

(Part - 1)

Lecture No. 8

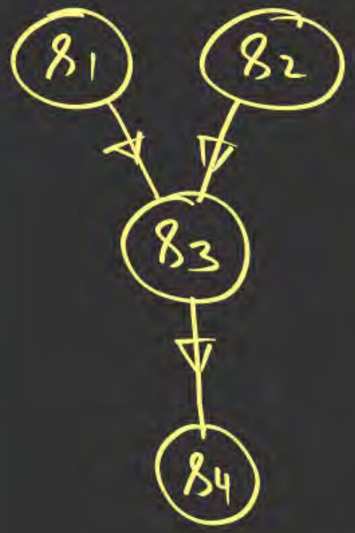
By- Dr. Khaleel Khan sir

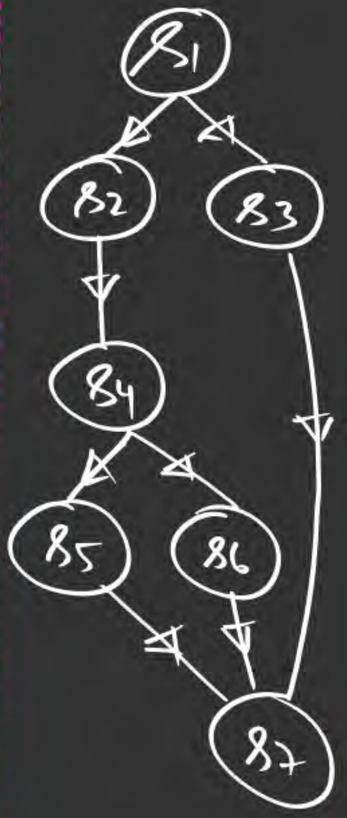
- Concurrency Conditions
- Concurrency Vs Paralellism
- Concurrency Constructs

Concurrency Mechanisms

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

Precedence Graph





Types of Concurrency

- **Real**
  - Physical
    - Multi-CPU
    - Multi-Core
    - Systems
- **Pseudo**
  - Single CPU System

- Interleaved (Pre Emp) execution

Concurrency Vs Parallelism

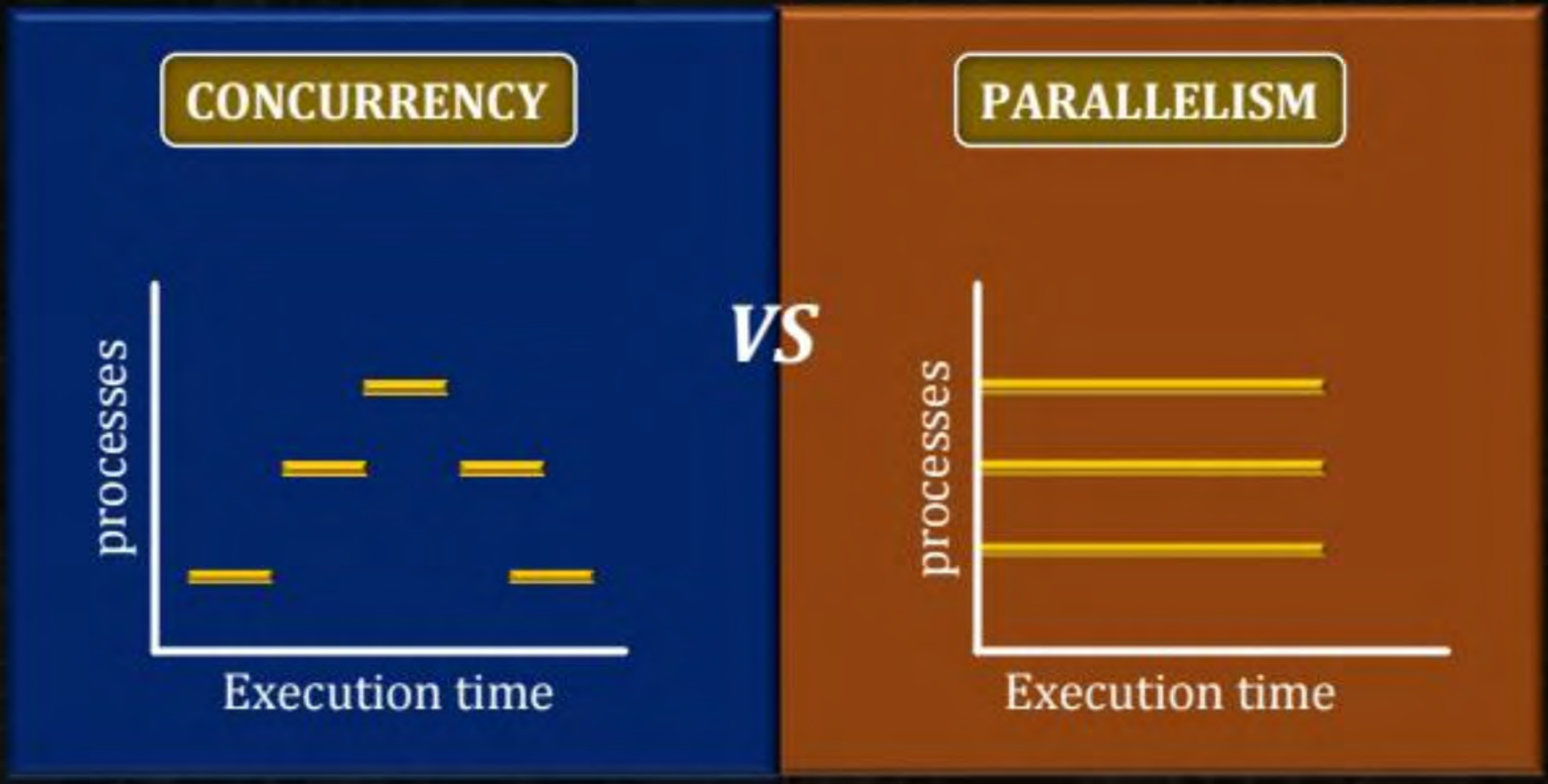
A System is said to be ***Concurrent*** if it can support Two or more actions in ***Progress*** at the same time

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.

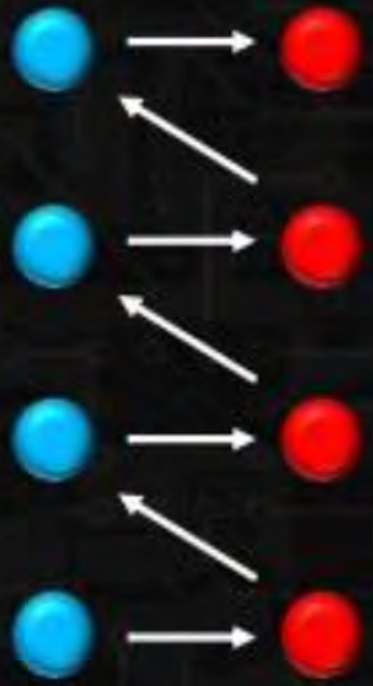
Parallelism is about doing Lots of things at once.

Concurrency Vs Parallelism



Sequential

Concurrent

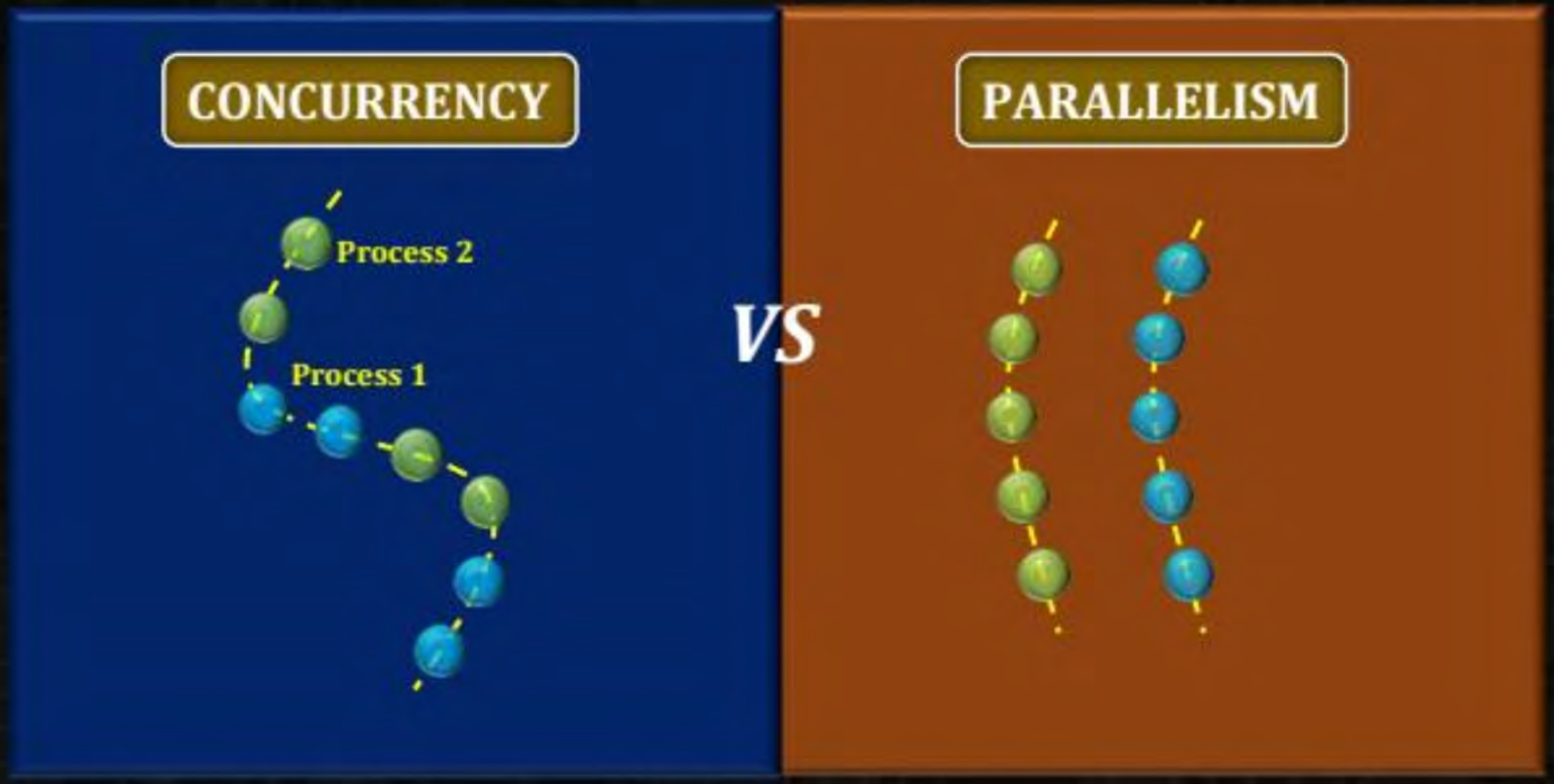


Parallel

"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

“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

Concurrency makes Parallelism easy.

Concurrency Conditions

S_i & S_j : Statements

→ o/p of one statement should not serve as I/p to other;

S < Read_set : R(S)
write_set : W(S)

① R(): {b,c} W(): {a}

S_i : a = b + c;

S_j : d = e * f;

No Common variables

② S_i : a = b + c;

S_j : d = b * c;

S : a += ++b * ++c;

R(S) = {a, b, c}

W(S) = {a, b, c}

int x; R: ∅ W: {x}

S_1 : scanf("%d", &x);
read

int a = 5; R: {a}

S_2 : printf("%d", a); W = ∅

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

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

S_i : $a = b + c;$
S_j : $a = l * m;$ } X

Bernstein's Concurrency Conditions

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

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

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

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

Concurrency Mechanisms/Construct:

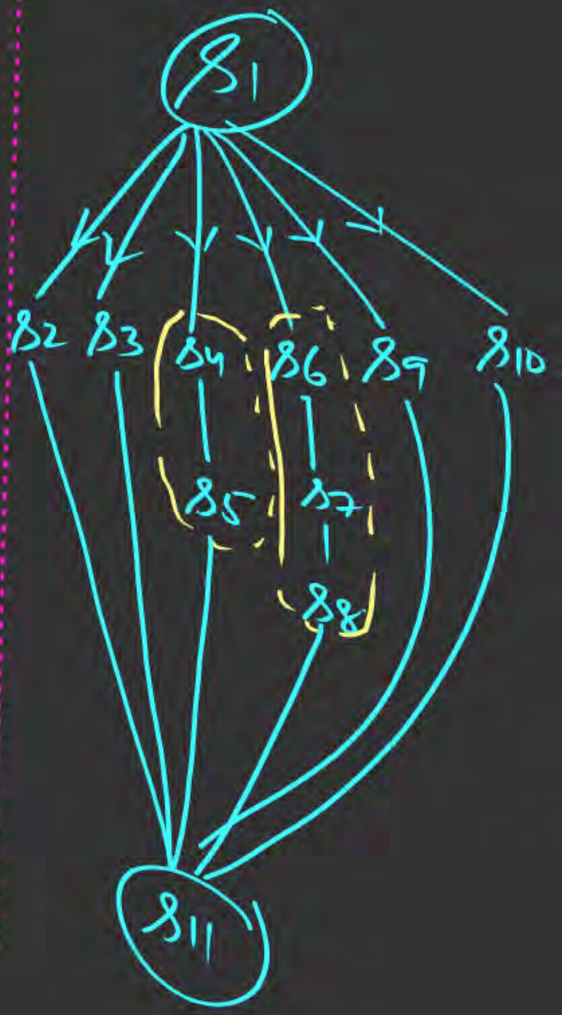
① Parbegin – Parend / Cobegin – Coend

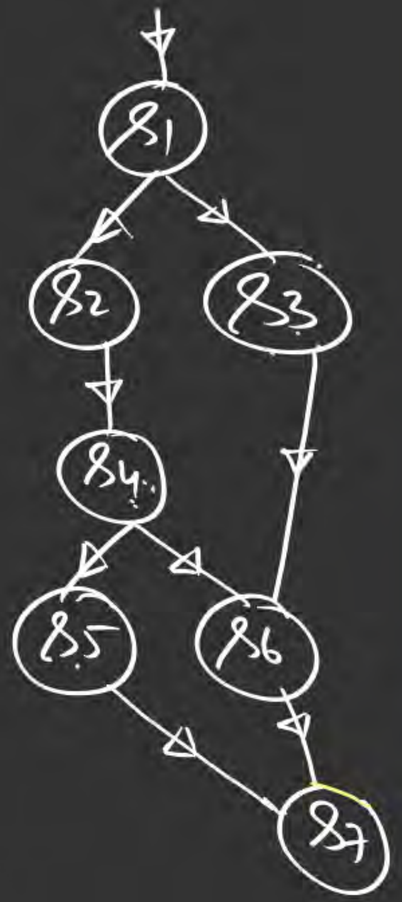
```
begin
  S1;
  S2;
  S3;
end
```

```
{
  S1;
  S2;
  S3;
}
```


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


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






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

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

Parbegin-Parend with Semaphores

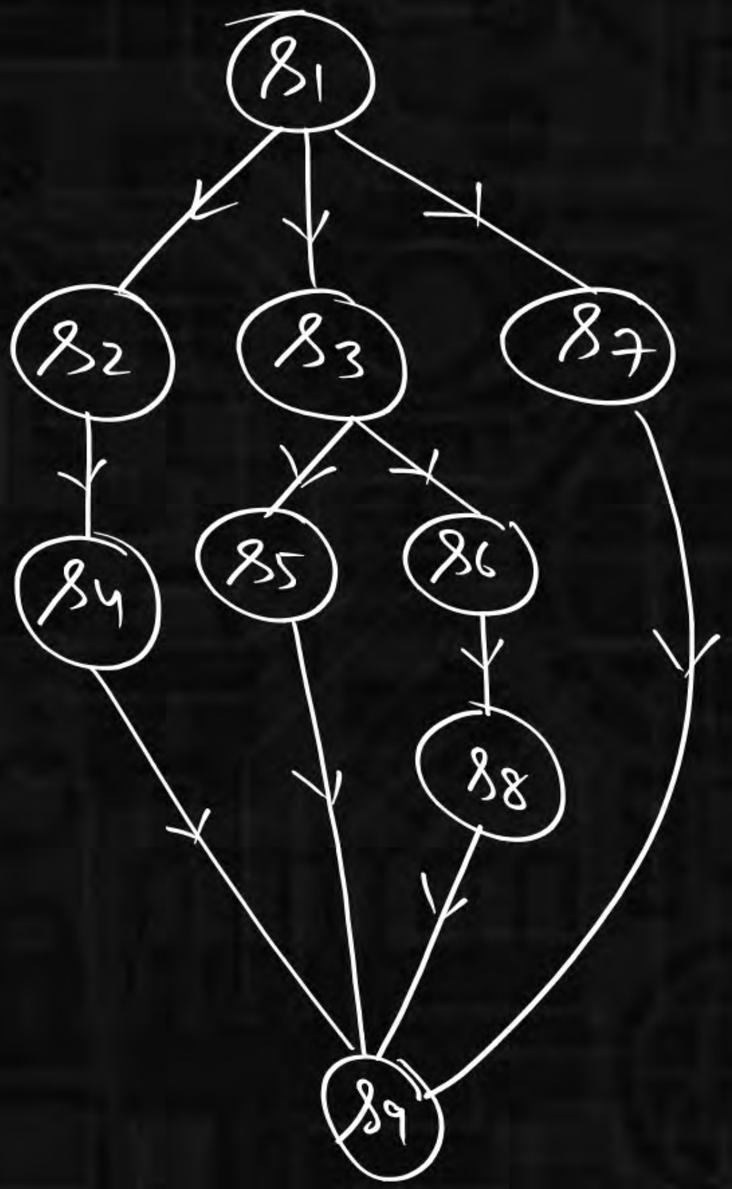
BSEM a,b,c,d,e,f,g = {0}

```
Parbegin
  begin S1; V(a); V(b); end;
  begin P(a); S2; S4; V(c); V(d); end
  begin P(b); S3; V(e); end;
  begin P(c); S5; V(f); end
  begin P(d); P(e); S6; V(g); end
  begin P(f); P(g); S7; end;
Parend
```

Q.

Draw the precedence graph for the concurrent program given below:

```
S1;
Parbegin
    begin
        S2;S4;
    end
    begin
        S3:
        Parbegin
                S5;
        begin
                S6; S8
        end
        Parend
    end;
    S7;
Parend;
S9;
```



Q.

```
void P(void)
{
    A;
    B;
    C;
}
```

```
void Q(void)
{
    D;
    E;
}
```

```
main ()
{
    Parbegin
        PO;
        QO;
    Parend
}
```

Indicate the Valid Output Sequences

P	Q
✓ 1. A B C D E	✓ 2. D E A B C
✓ 3. A D B E C	✗ 4. A E B D C
✗ 5. D C E B A	

Q.

```
int x = 0, y = 0;
Cobegin
    begin                 S1
        1: x = 1;
        2: y = y+x;
    end
    begin                 S2
        3: y = 2;
        4: x = x+3;
    end
Coend
```

Final values of x & y

I) x = 1; y = 2

II) x = 1; y = 3

III) x = 4; y = 6

I: x = 1, y = 2 ? ✗

II: x = 1; y = 3 ✓

x: 1 ø 3 — y: 3 ø 2

S2: 3; 4;
S1: 1; 2

III: x = 4; y = 6; ✓

x: x ø 4 — y: 6 ø 2

S1: 1;
S2: 3; 4;
S1: 2;

Q.

```
int x = 0, y = 20;
Bsem mx = 1; my = 1;
Cobegin
    begin
        P(mx);
        x = x+1;
        V(my);
    end
    begin
        P(my);
        x= y+1;
        V(mx);
    end
Coend
```

Final possible values of x 21, 22

x = 22

int x = 0, y = 20

Cobegin

S1: begin
    x = x+1;
    (Load R, x ; Inc R ; Store x, R)
end

S2: begin
    x = y+1;
end;

Coend

y
20

x
1 ~~0 2~~
21
R

S1 : L ; I

S2 : ✓

S1 : S

∴ Final Possible values of x = 21, 22, 1

Q.

```
integer B = 2;
P1()
{
  1. C = B - 1;
  2. B = 2*C;
}
P2()
{
  3. D = 2*B;
  4. B = D - 1;
}
main ()
{
     Parbegin
          P1()
          P2()
     Parend
}
```

The number of distinct values of B is/are _____ 3 [2,3,4]

RQ

(I) 1; 2; 3; 4; :

B | ~~3~~ | 1 (C) | 4 (D)

(II) 3; 4; 1; 2;

Q.

```
int count = 0;
void test()
{
    int i, n = 5;
    for(i = 1; i<=n; ++i)
    count = count+1;
}
main ()
{   Parbegin
      P1 : test();
      P2 : test();
    Parend
}
```

L, I, S : Re

Exercise :

10

What is the minimum and maximum value of count?

Q.

Consider the following pseudocode, where S is a semaphore initialized to 5 in line#2 and **counter** is a shared variable initialized to 0 in line#1. Assume that the increment operation in line #7 is not atomic

```
1. int counter = 0
2. Semaphore S = init(5);
3. void parop(void)
4. {
5.        wait (S);
6.        wait (S);
7.        counter++;
8.        signal (S);
9.        signal (S);
10.}
```

If five Processes execute the function **parop** concurrently, which of the following program behavior(s) is/are possible?

A. There is a deadlock involving all the Processes

B. The value of counter is 5 after all the Processes successfully complete the execution of parop

C. The value of counter is 1 after all the Processes successfully complete the execution of parop

D. The value of counter is 0 after all the Processes successfully complete the execution of parop

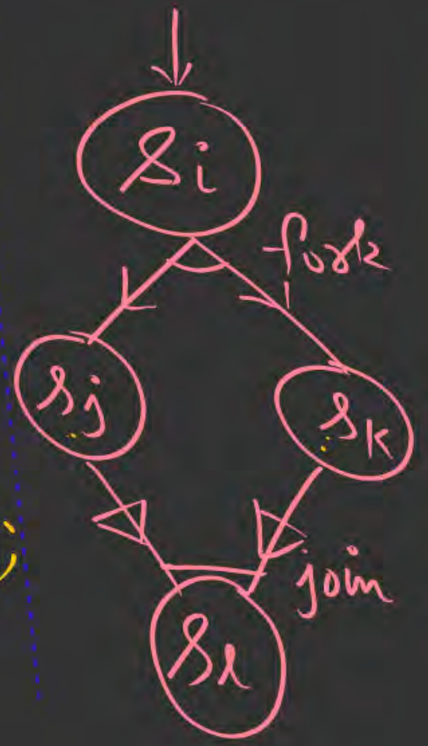
II: Fork and join construct

: Execution of fork L, will start Two Computations concurrently

(i) Statement @ label L
(ii) Next Statement after fork L;

Syntax:

```
Count = 2;
Si;
fork L;
Sj;
goto X;
L: Sk;
X: Join(Count);
Sx;
```




```
Join(int Count)
{
    Count = Count - 1;
    if (Count ≠ 0) exit
    else
        return;
}
```

Atomic


THANK YOU
GW
SOLDIERS !