COMPUTER SCIENCE

CLASSICAL IPC PROBLEMS 08

Dr. KHALEEL KHAN SIR

TOPICS TO BE COVERED

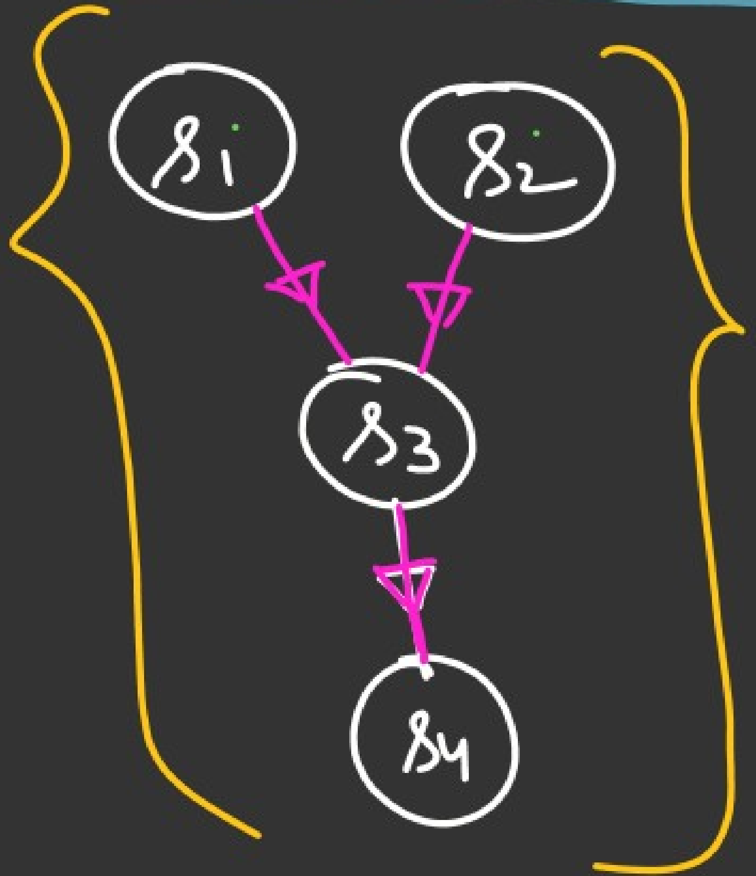
- Classical IPC Problems

c/c++ Sequentially/Parallely
Concurrent

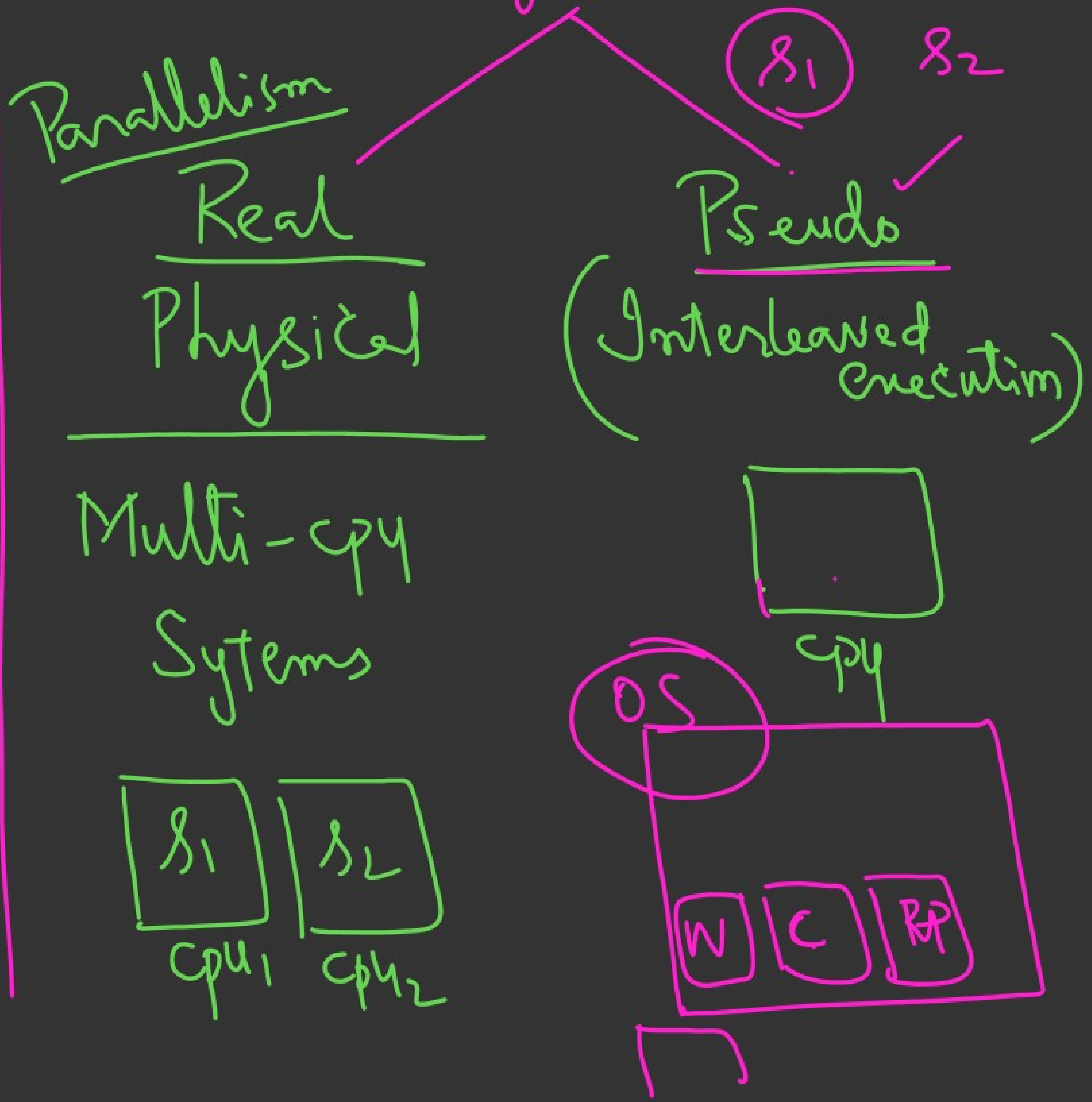
```
{        L, L, A, S
  S1: a=b+c;
  S2: d=e*f;
  S3: k=a+d;
  S4: l=k*m;
}
```

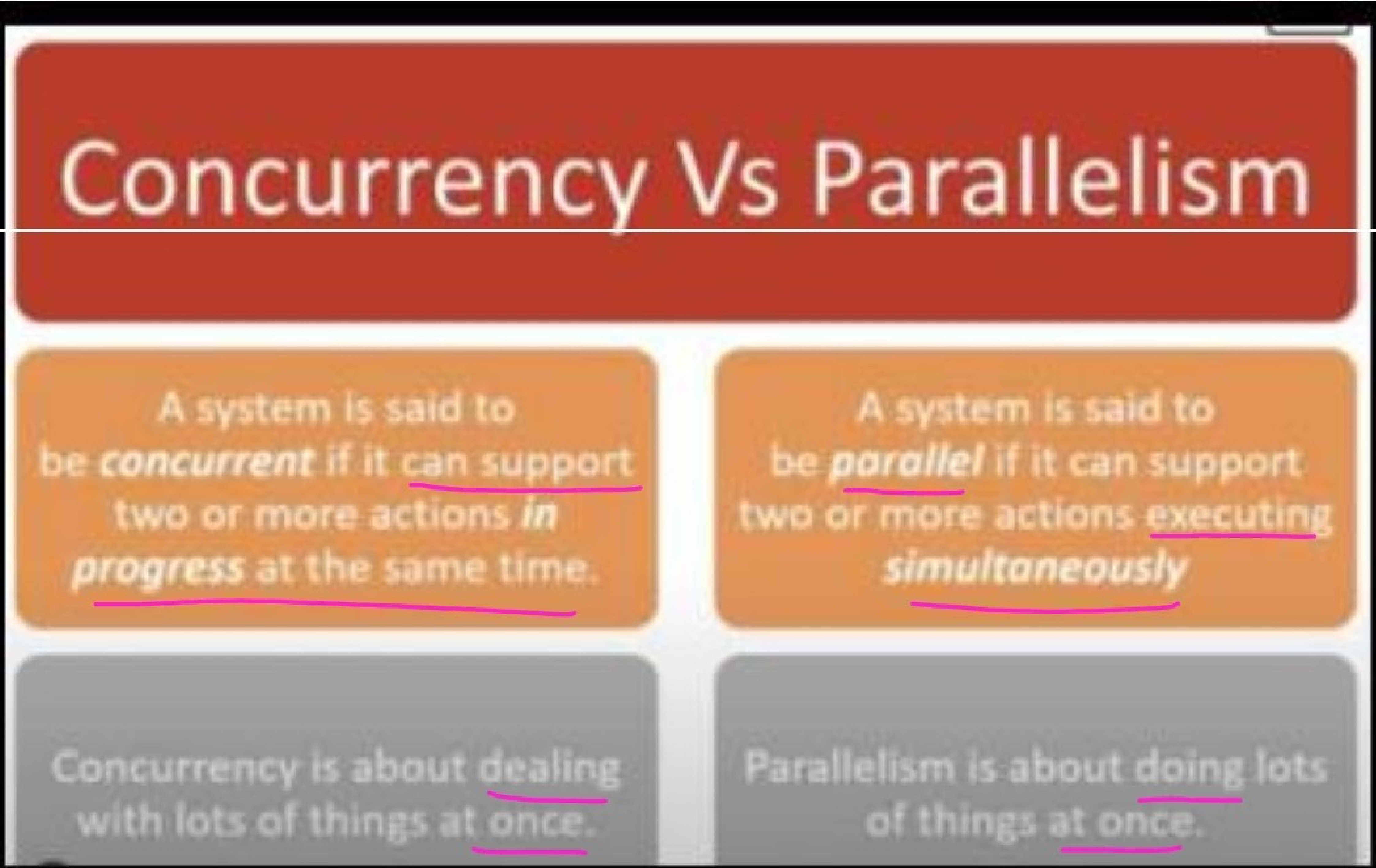
Independent

Precedence Graph $G=(V,E)$



Types of Concurrency





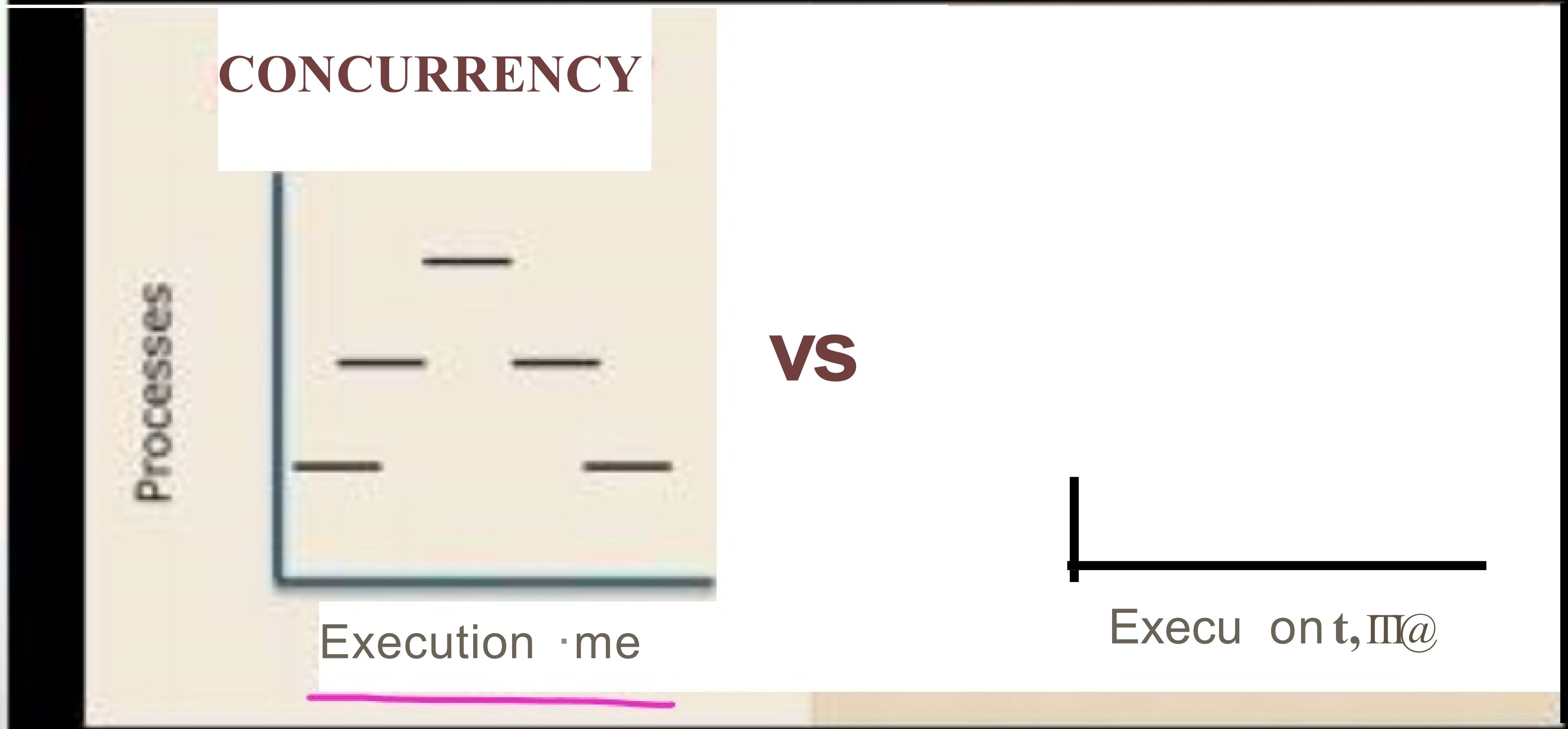
Concurrency Vs Parallelism

A system is said to be ***concurrent*** if it can support two or more actions ***in progress*** at the same time.

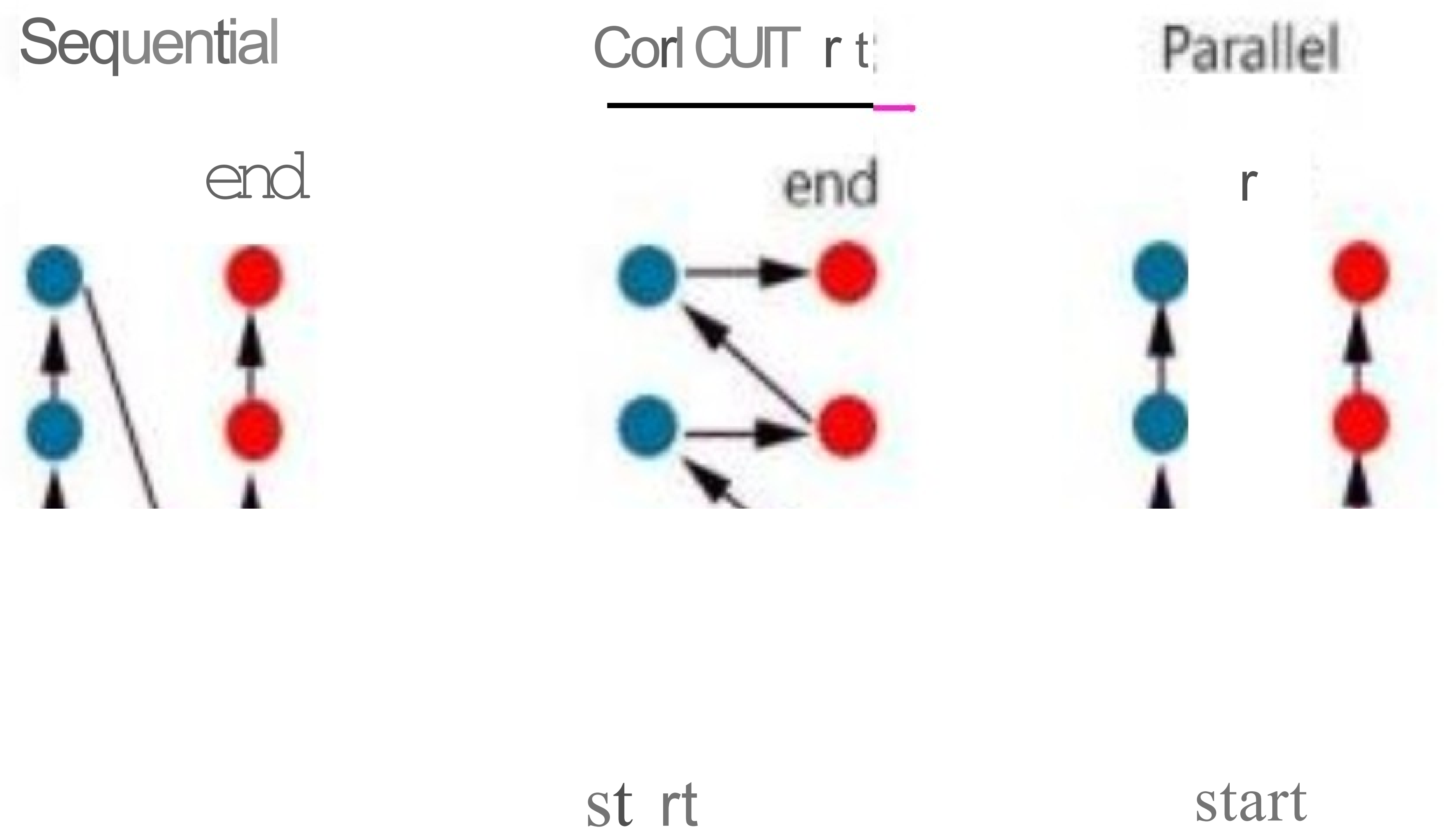
A system is said to be ***parallel*** if it can support two or more actions executing ***simultaneously***

Concurrency is about dealing with lots of things at once.

Parallelism is about doing lots of things at once.

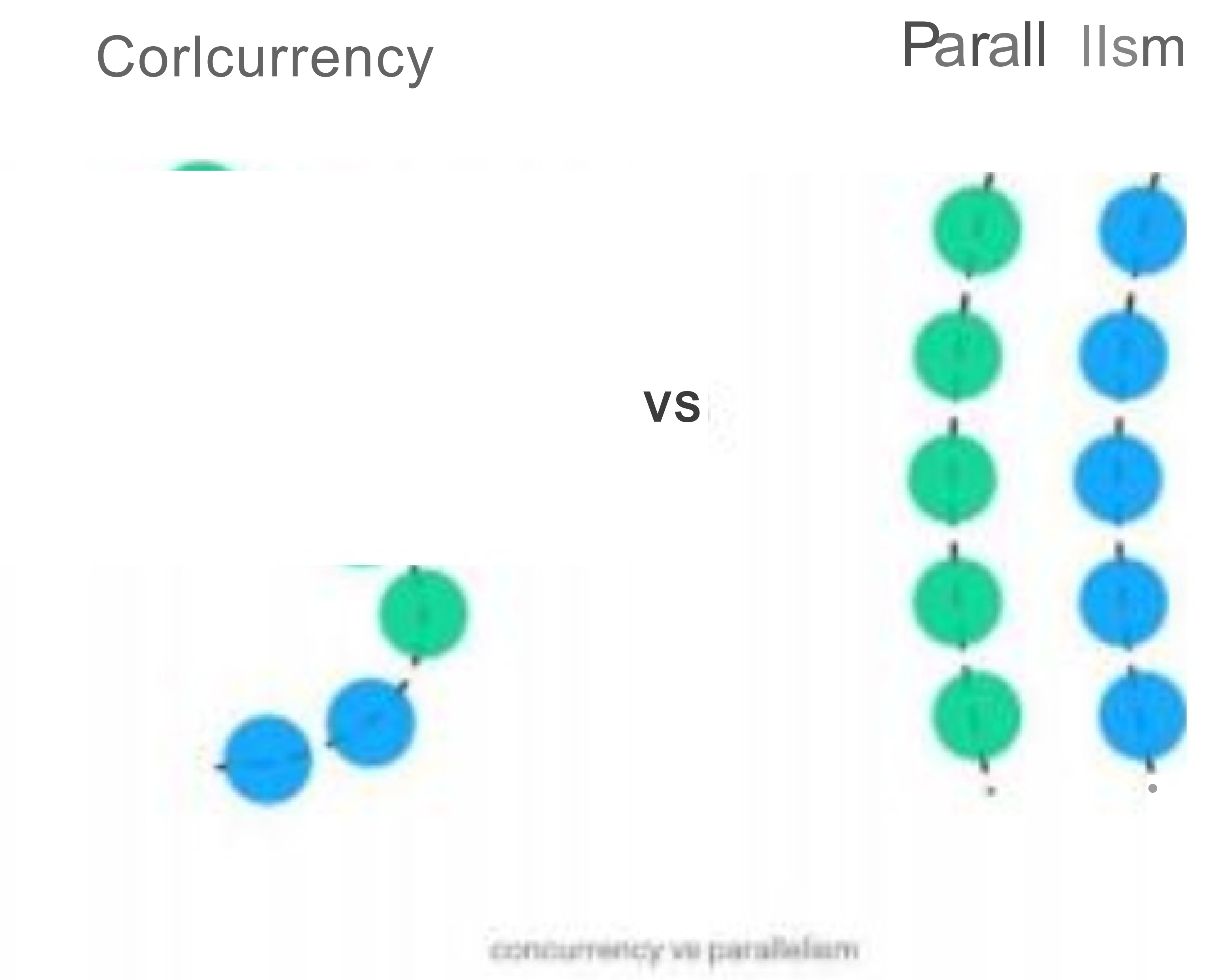


dealing vs doing

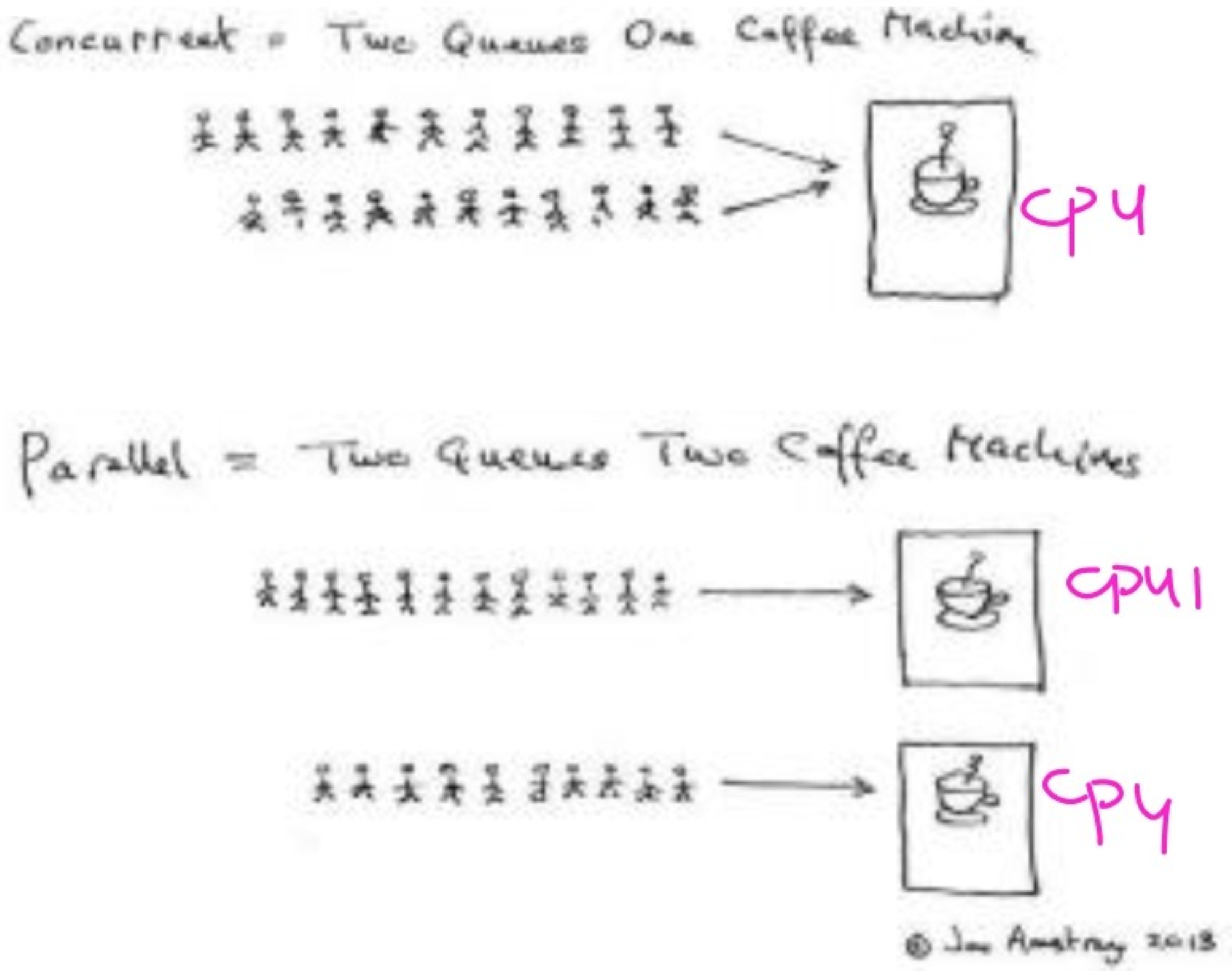


Concurrency is abo 1 o aling 1li h lots of things at onr&l arallelism is about oing lo s o hings at on e. - ob Pike

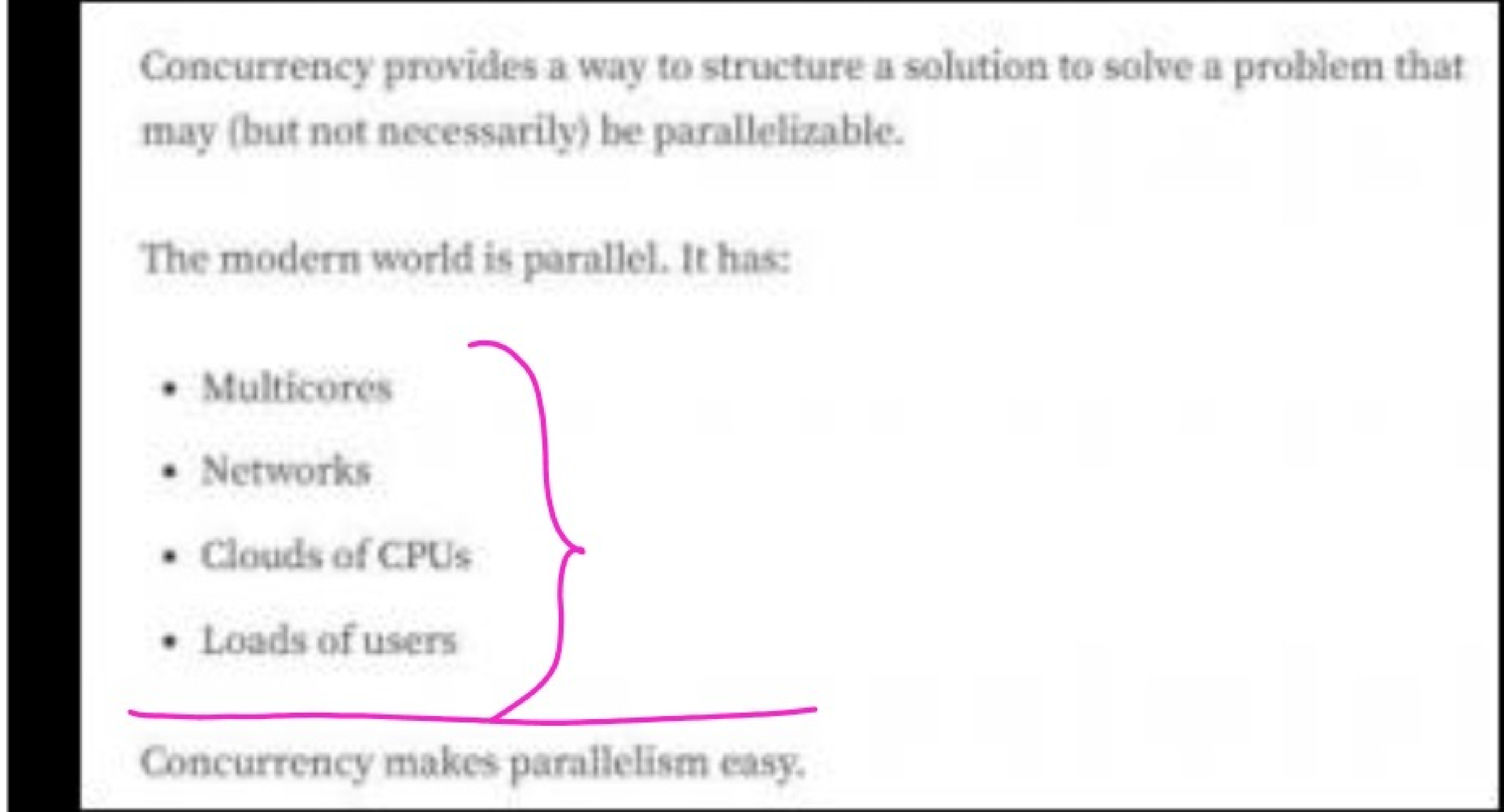
Concurrency Is not Parallelism



An analogy


Concurrent = Two Queues One Coffee Machine
CPU
Parallel = Two Queues Two Coffee Machines
CPU1
CPU
© Joe Armstrong 2013

·cone r ency 1sabouts ructure. parallehsm 1sabout e ec t1on'



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

$$\boxed{S_1 : a = b + c; \quad S_2 : d = e * f;}$$

$$\boxed{S_3 : a = b + \underline{c}; \quad S_4 : d = b \underline{* c};}$$

$$S_5 : \underset{}{\overset{}{a}} = b + c; \quad d = a * f; \quad \times$$

$$\left\{ \begin{array}{l} S_8 : \underline{d} = b + c; \\ S_9 : \underline{d} = k * f; \end{array} \right\} \times$$

Two Stmnts are Concurrent

↳ Independent

① No Shared Variables

② output of one Stmt Should not Serve as IP to other Stmt

Let 'S' : be a Statement

S < Read Set [R(S)]
  write Set [W(S)]

S : a = b+c

R() = {b, c}

W() = {a}

a = a + (++b * ++c)

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

R() = {a, b, c}

W() = {a, b, c}

```
int x;

Scanf("%d", &x);   "S"
```

R() = ∅

W() = {x}

```
printf("%d", x)
```

R() = {x}

W() = ∅

Concurrency Conditions

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

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

Bernstein's Conc. Cond's

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 $\emptyset$

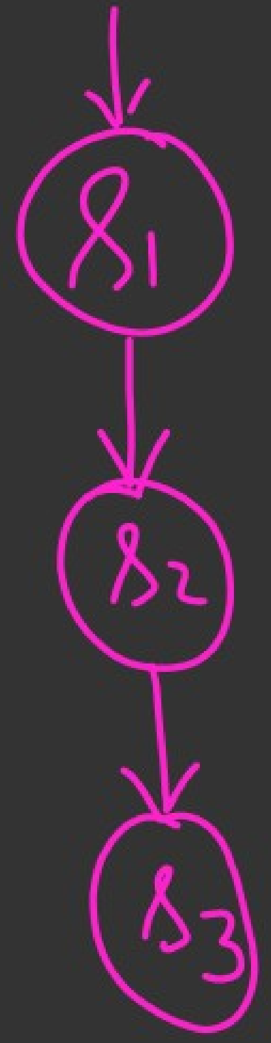
Concurrency Mechanisms/constructs

1. Parbegin - Parend / Cobegin - Coend

sequential construct

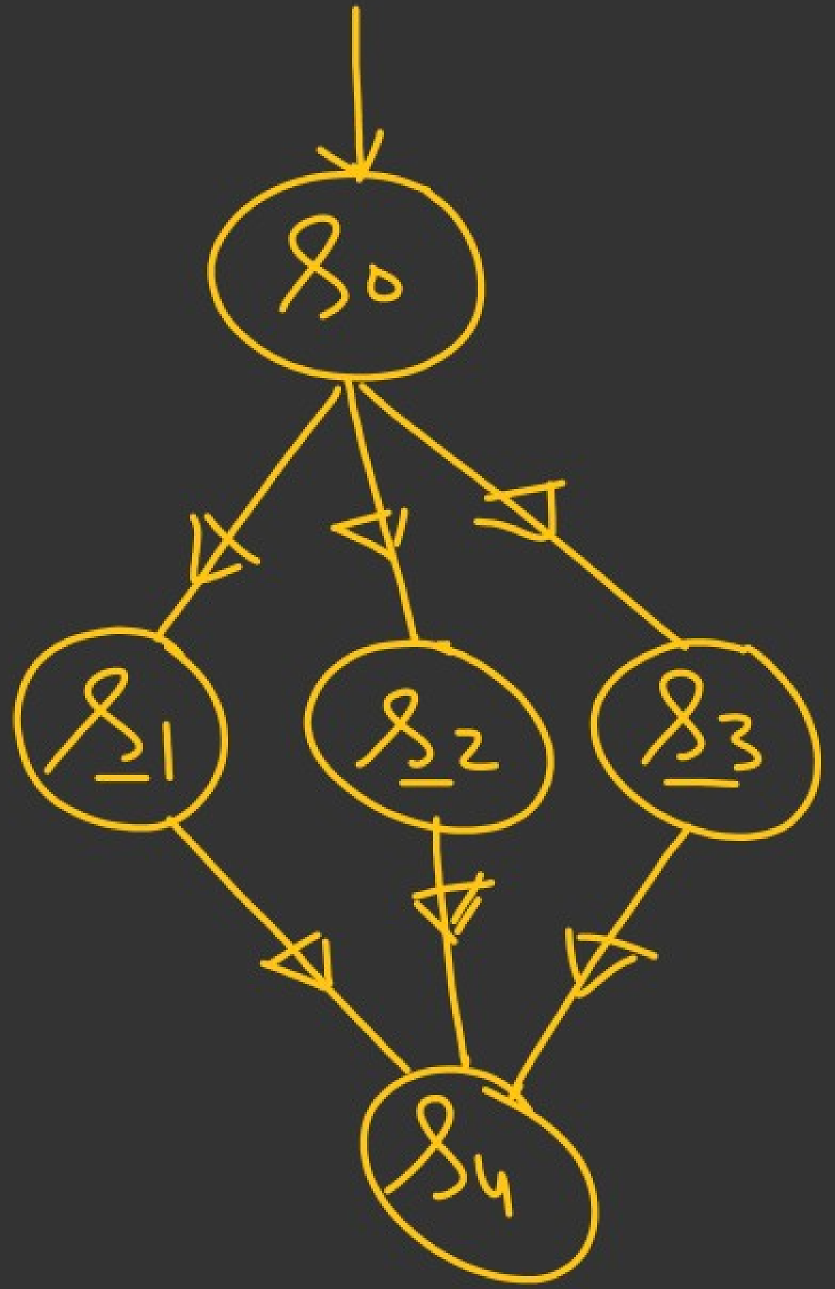
```
begin
    S1;
    S2;
    S3;
end
```

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



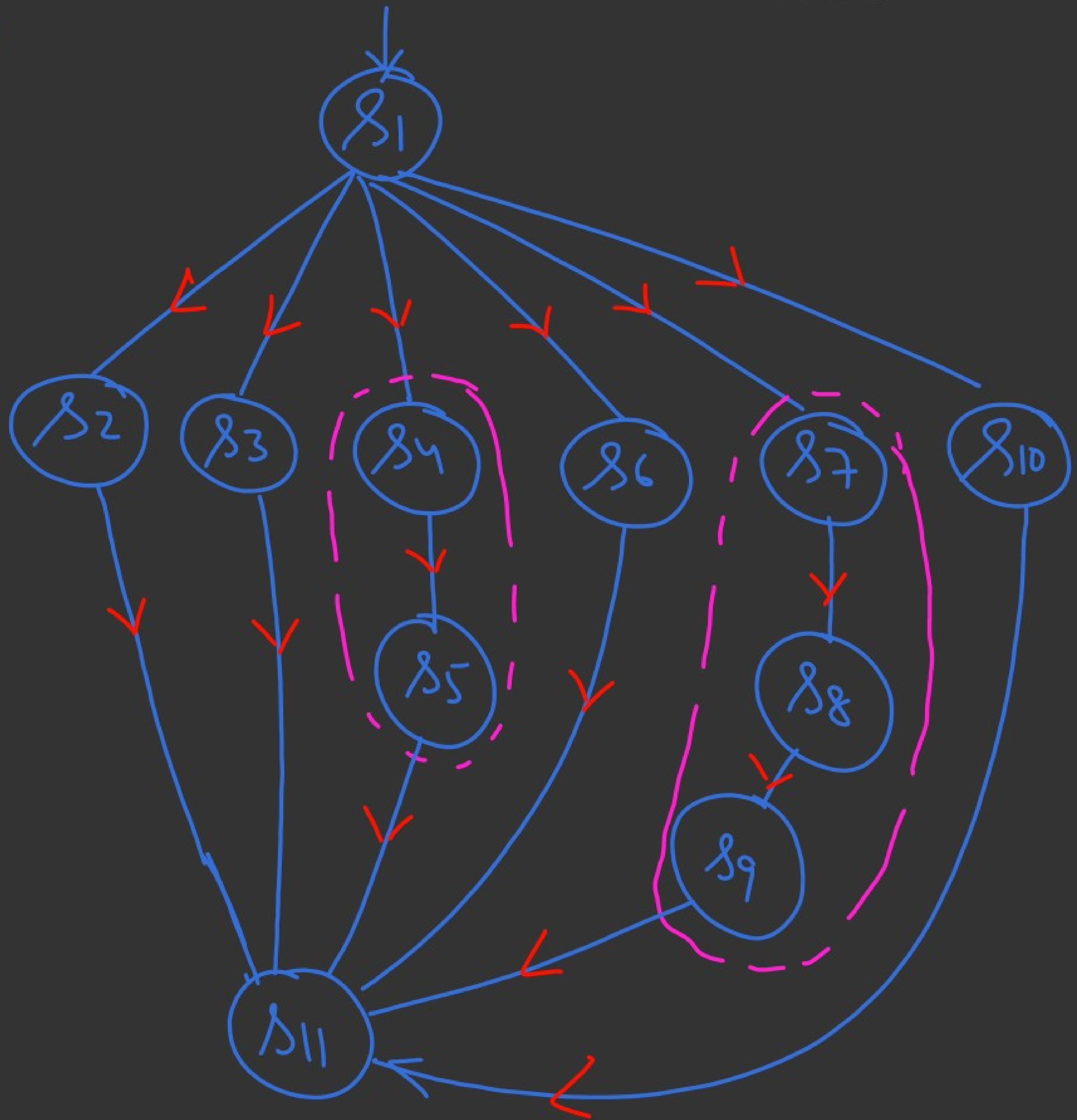
```
S0;
Parbegin
    S1;
    S2;
    S3;
Parend
S4;
```

(Conc. Program)

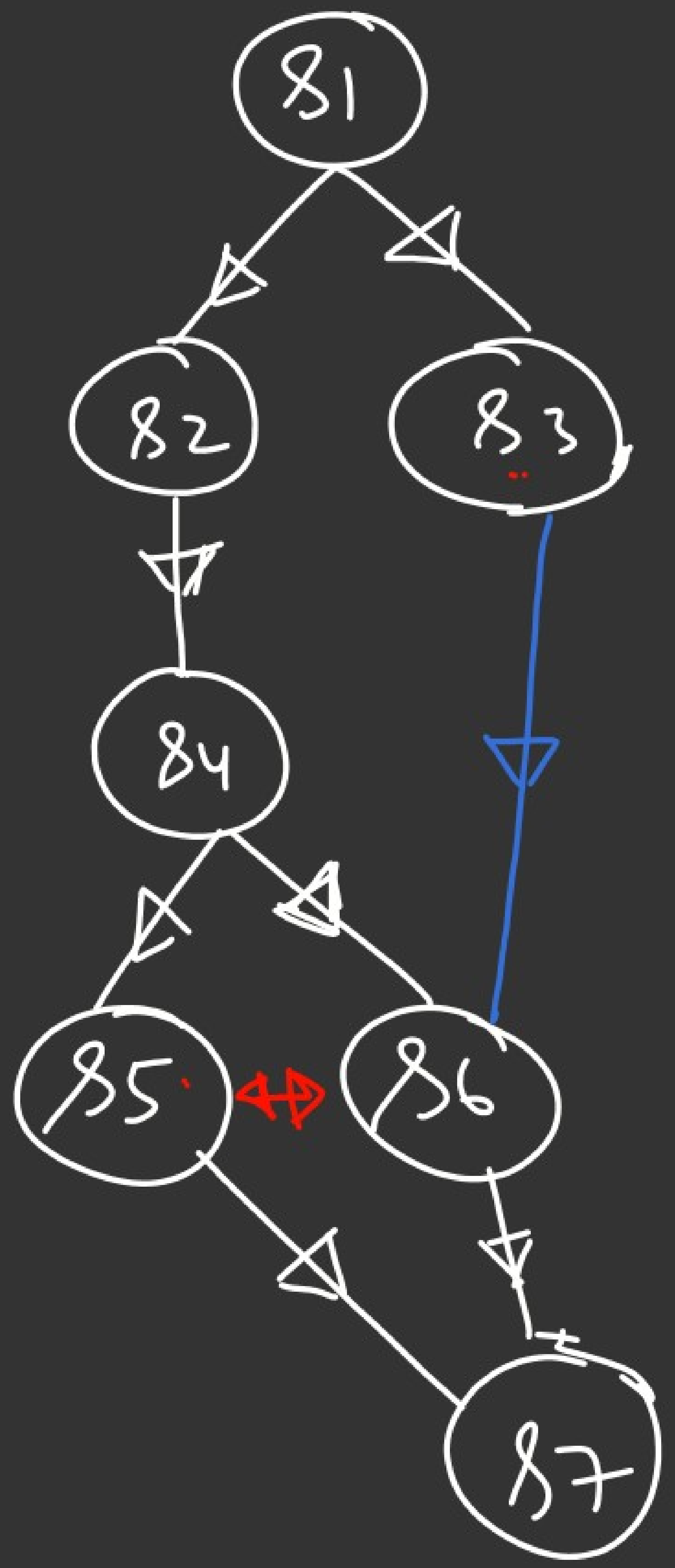


→ Parbegin - Parend Can be nested with begin-end

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



This Graph is non Implementable with Parbegin-Parend



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

X 6:45pm

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

THANK YOU!