CS & IT ENGINEERING

Operating System

Process Synchronization

Lecture No. 7

By- Dr. Khaleel Khan Sir

TOPICS TO BE COVERED

- Blocking Mechanisms :
- Sleep-Wakeup
- Semaphore *

Q.

Fetch_And_Add (X, i) is an atomic Read-Modify-write instruction that reads the value of memory location X, increments it by the value i and returns the old value of X, It is used in the pseudocode shown below to implement a busy-wait lock. L is an unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

```
AcquireLock(L)
{
  While (Fetch_And_Add (L,1));
   L=1;
}
ReleaseLock(L)
{
  L=0;
}
```

P2; P3 . . . . .

int = 2B

32767

+

500000

<CS (P1)>

(race Situation)

<MCQ> | <MSQ>

1 L

$\boxed{\cancel{0}}$

CS is free

This implementation

[✓ A.] fails as L can overflow

[✓ B.] fails as L can take on a non-zero value when the lock is actually available

1

C. works correctly but may starve some processes

D. works correctly without starvation

```
int Fetch-And-Add(x,i)
{
        int rv;
        rv = x;        Atomic
        x = x+i;       exec
        return(rv);
}
```

```
int lock=0;

Entry   while(Fetch-And-Add(&Lock,1));
        Pre →  L=1;

        <CS>

Exit    L=0
```

RQ: P1 P2 P3

lock: ~~0~~ ~~1~~ 2

CS: P1

int : 2B

<32767> →

Normal

I:

t1 : P1 : F_A_A : → 0

t2 : P2 : F_A_A : → 1

II:

lock: ~~0~~ 1 ~~0~~ ~~2~~ (1)

t1 : P1 : F-A-A : 0 ✓ : <CS> . . . Pre

t2 : P2 : F-A-A : 1 ; Pre : <L=1>

t3 : P1 : <CS>;

t4 : P2 : L=1 ; FAA: 1

II : Blocking Mechanisms / Non-Busy-waiting

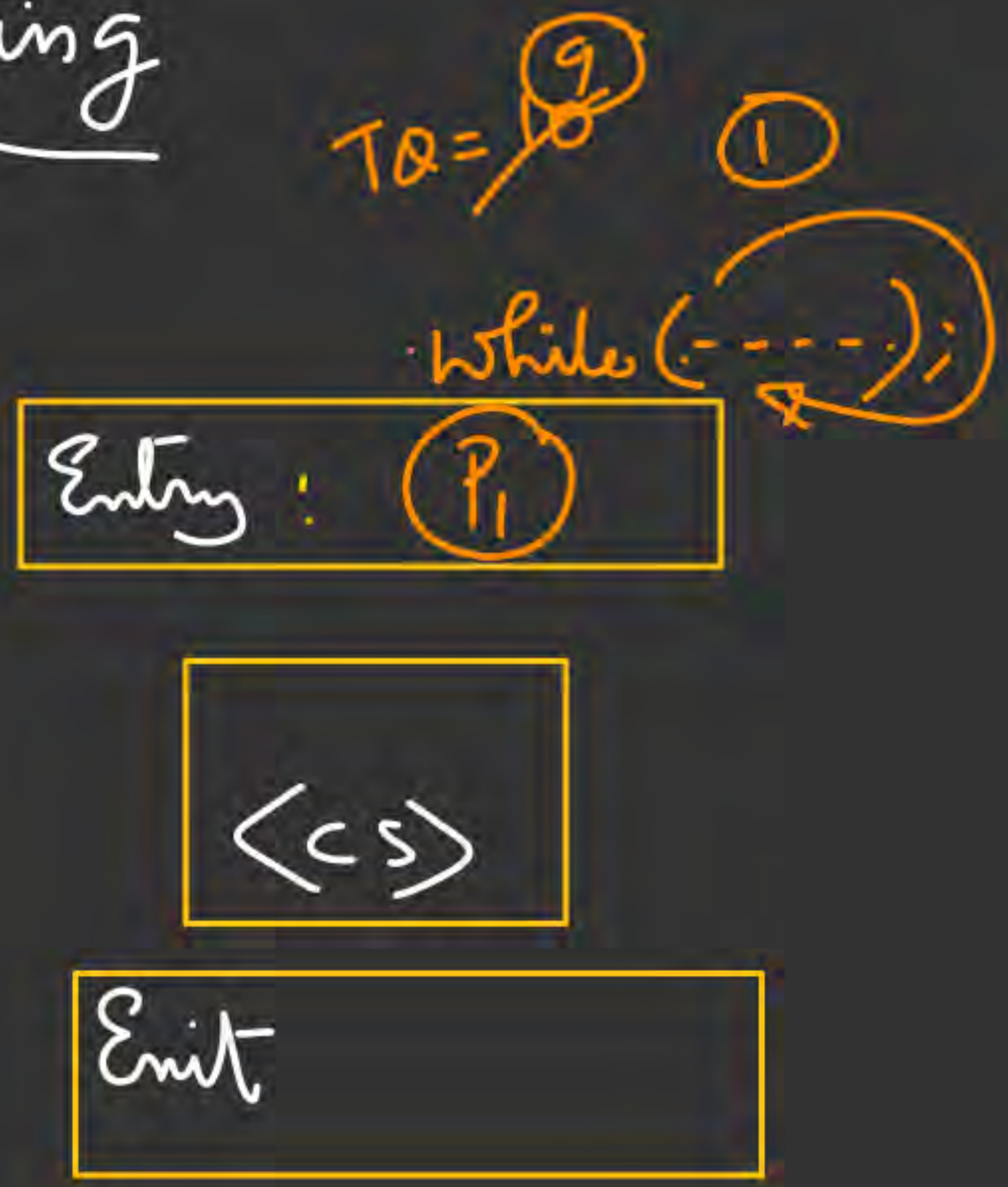
< is to avoid wastage of cpu time > in the form of loops ;

: if - then - else

: Sleep-wakeup

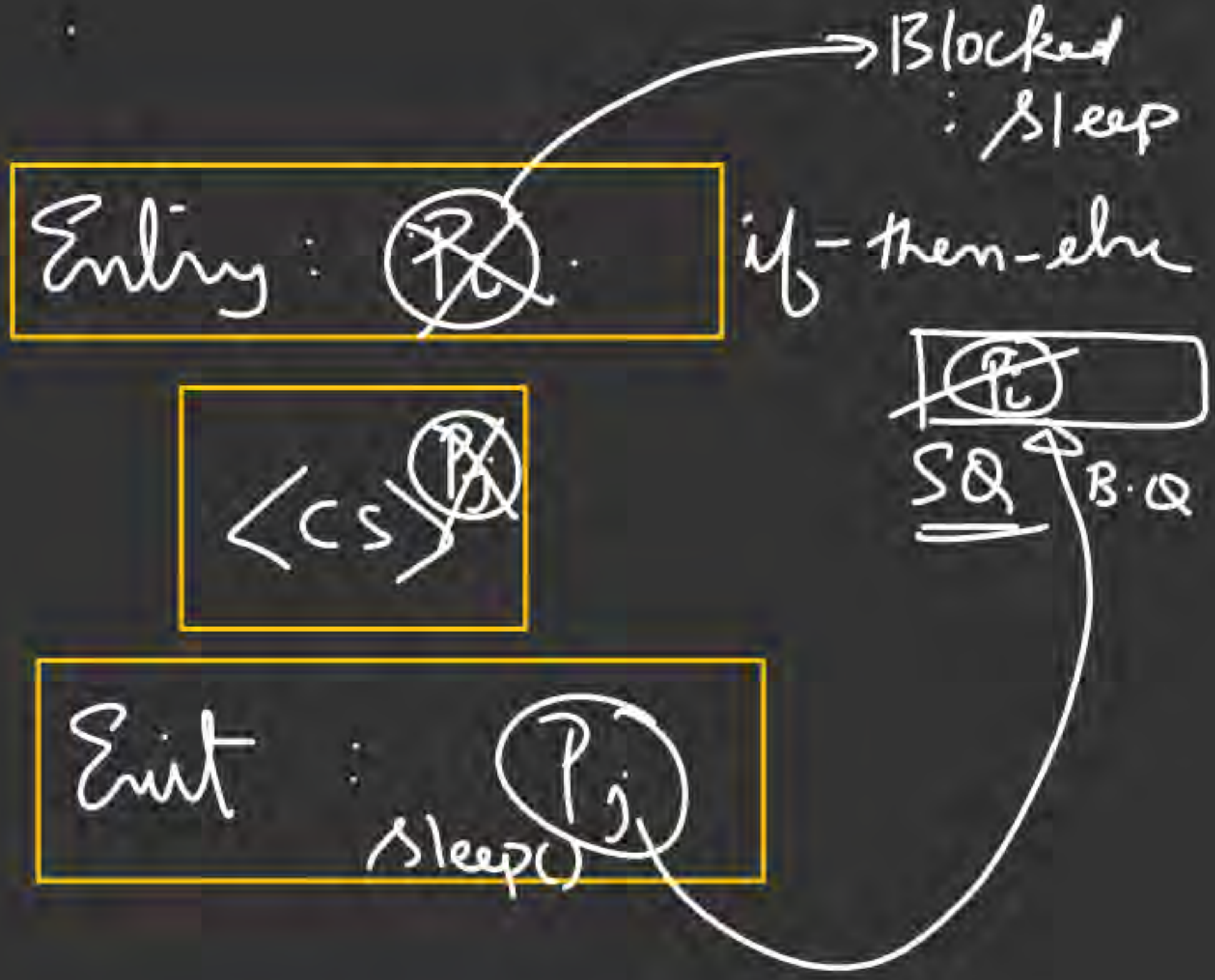
: Semaphores

: Monitors



① Sleep() & wakeup()

- → Blocking Construct
- → Multi-Process Solⁿ
- → are OS primitives
- → when Process executes Sleep() :→ gets Blocked till some other process wakes it up (wakeup())

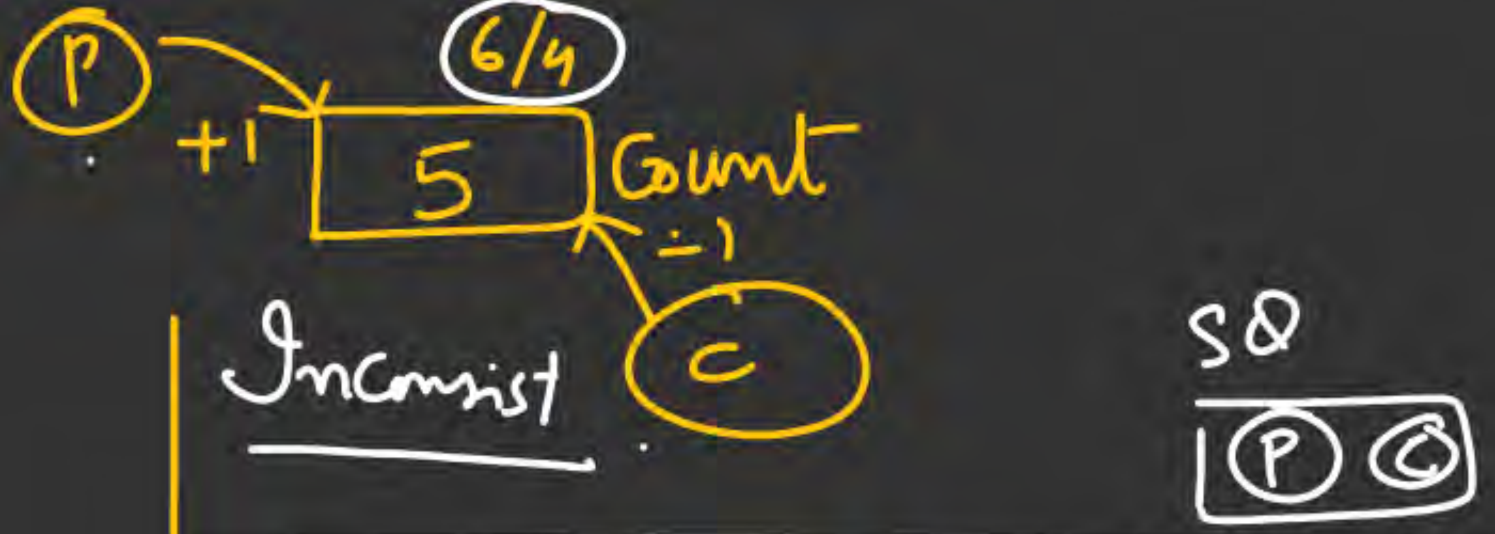


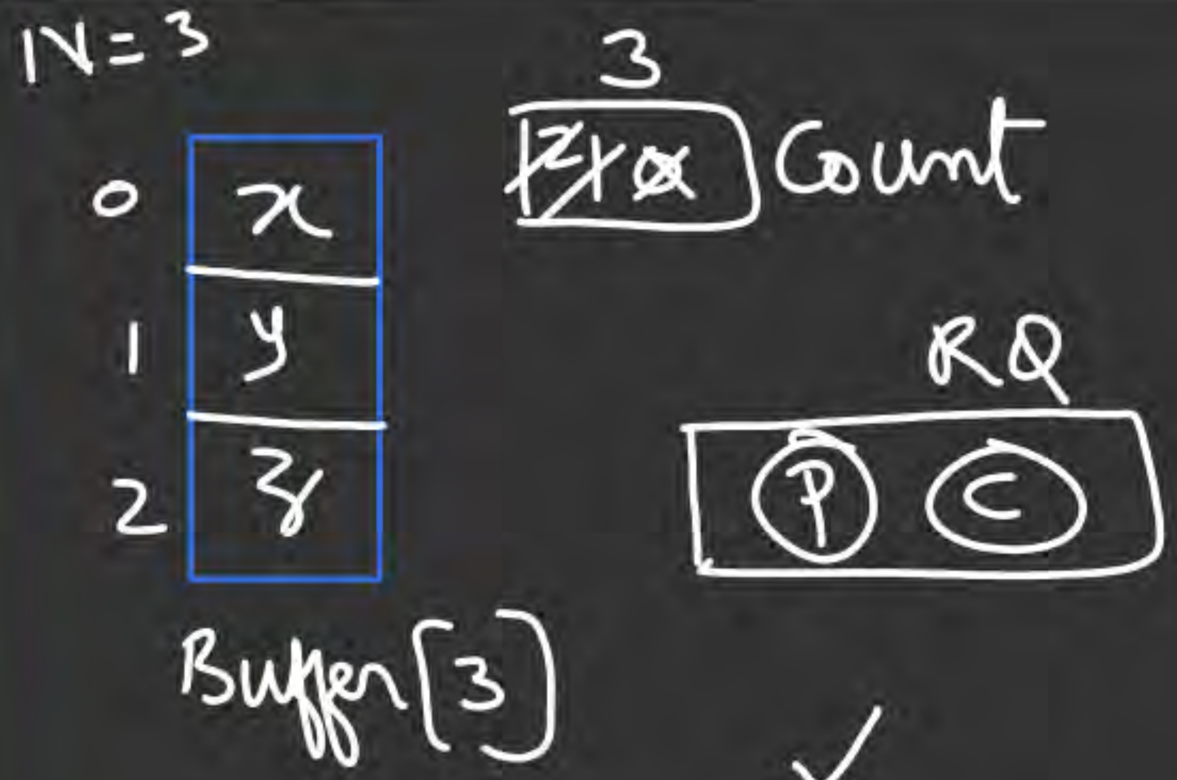
Producer-Consumer using Sleep() & wakeup()

```
#define N 100
int Buffer[N];
int Count = 0

void Producer(void)
{
    int itemp, in=0;
    while (1)
    {
      a) itemp=Produce_item();
      b) if(Count==N) Sleep();
      c) Buffer[in]=itemp;
      d) in=(in+1)%N;
      e) Count=Count+1;
      f) if(Count==1) wakeup(Consumer);
    }
}
```

```
void Consumer(void)
{
    int itemc, out=0;
    while(1)
    {                       L; C; J; sleep
      a) if (Count==0) Sleep();
      b) itemc=Buffer[out];
      c) out=(out+1)%N;
      d) Count=count-1;
      e) if (Count==(N-1)) wakeup(Producer);
      f) Process_item(itemc);
    }
}
```





t1: C: L; C; J; Pre; <sleep>

t2: P: - - - - - Sleep(); ✓

t3: C: Sleep();

Inconsistency

Deadlock

** SEMAPHORES:

→ Blocking Contructs

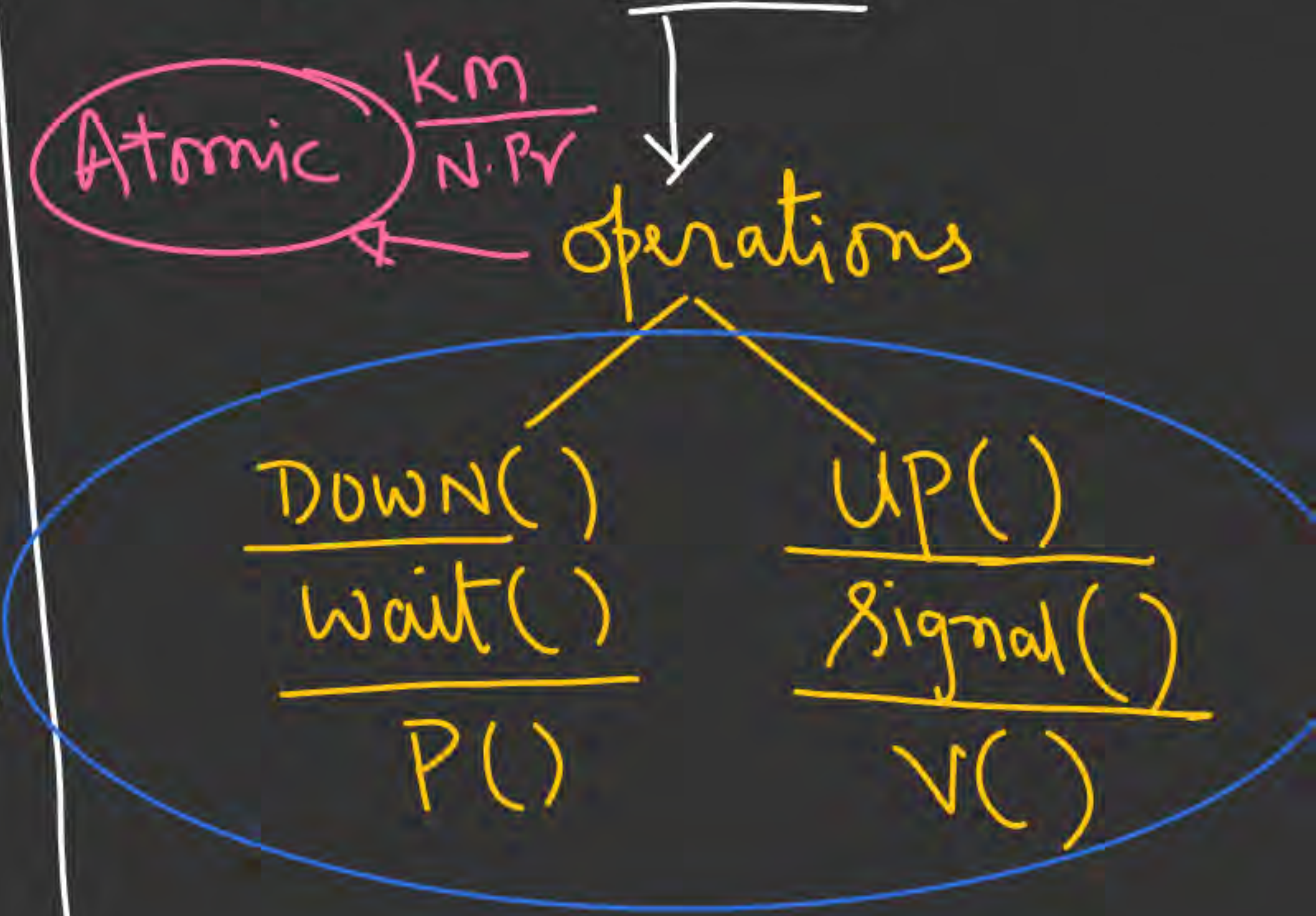
→ OS based
< : is an OS resource (S/w) >

→ General purpose utility
{ — < CS >
— Requirements of classical IPC Problems
— Concurrency Mechanisms
}

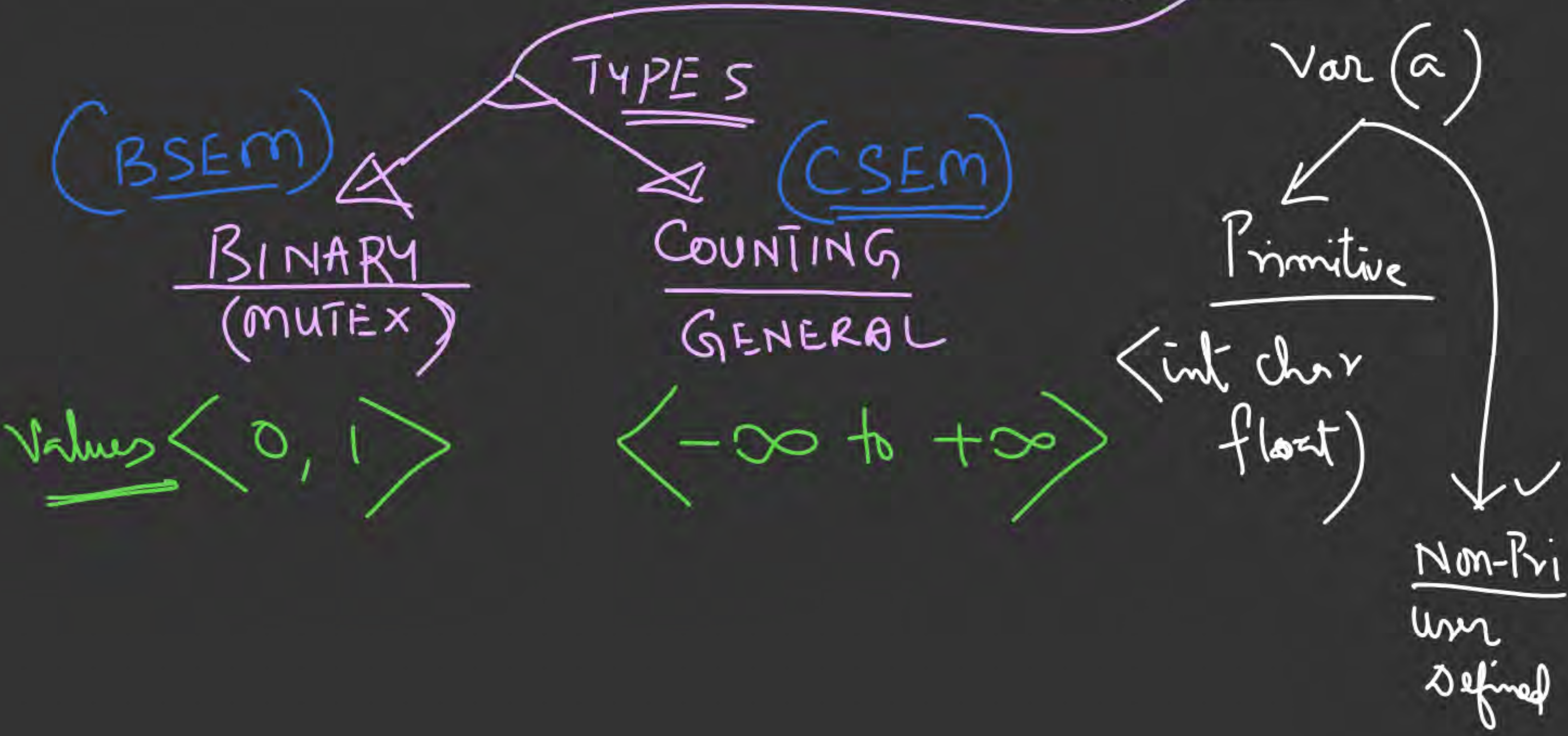
+ Practically In use tom (OS)

→ It was proposed & Impl. by E. Dijkstra <Dutch>

→ Semaphore is Implemented as an A.D.T



Defn : is a variable ⟨ADT : SEM⟩ that takes only integer values;



TYPES

(BSEM)

BINARY

(MUTEX)

Values ⟨0, 1⟩

(CSEM)

COUNTING

GENERAL

⟨−∞ to +∞⟩

Var (a)

Primitive

⟨int char float⟩

Non-Pri

User Defined

THANK YOU
GW
SOLDIERS !