CS & IT ENGINEERING

Operating System

Process Synchronization/ Co-ordination

Classical IPC Problems

Lecture No. 10

By- Dr. Khaleel Khan Sir

TOPICS TO BE COVERED

- Producer-Consumer Problem
- Readers – Writer Problem
- Dining Philosopher Problem

Q) Which of the following may result in a process getting Blocked (MSQ)

a) DOWN

b) UP

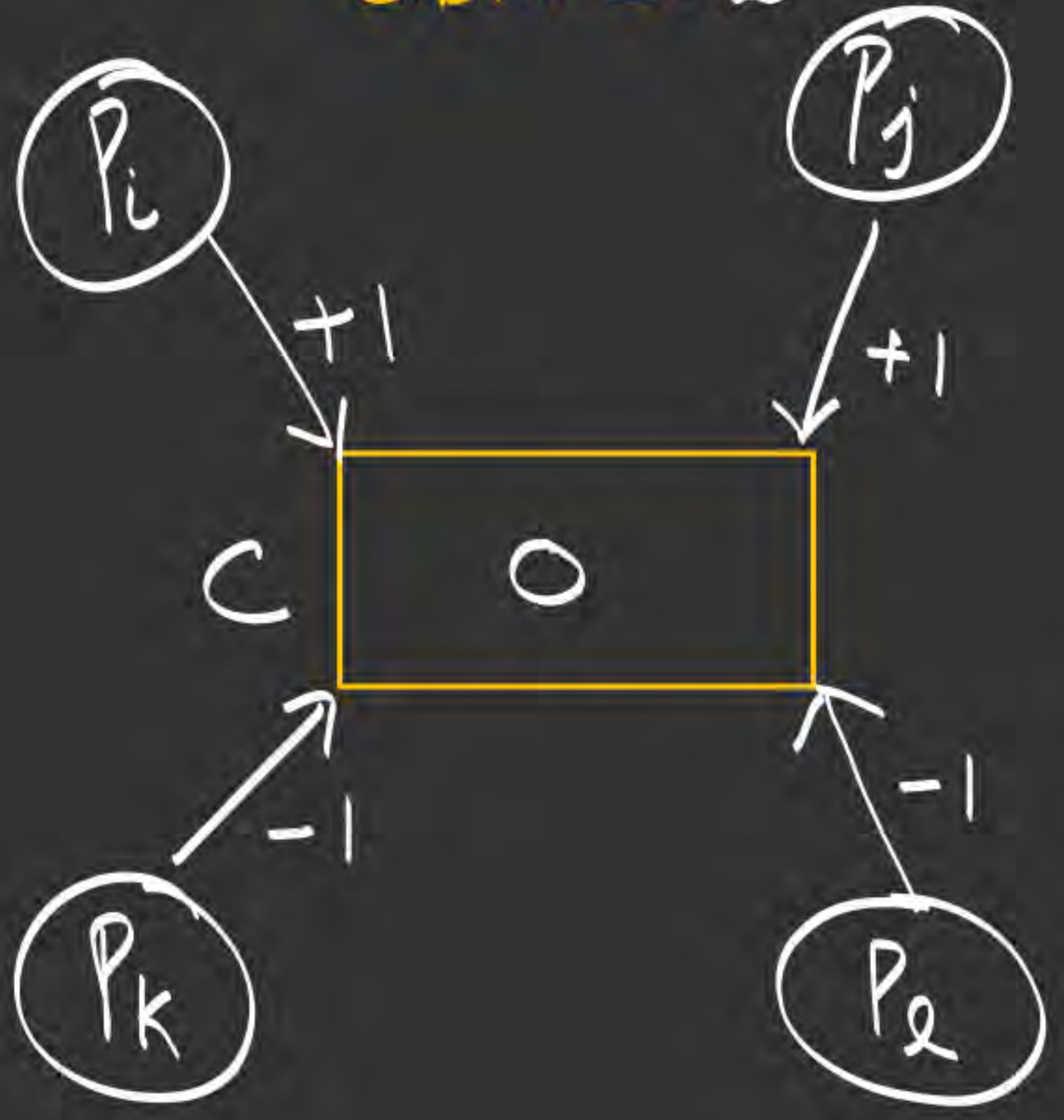
c) System - Call

d) Scheduler Dispatch

$\langle A \quad C \rangle$

$$\text{Ready} \xrightarrow{S\text{-}D} \text{Running}$$

CSEM S=2



P_x $x = i, j, k, l$

{

P(S);

$C = C \pm 1;$ } CS

V(S);

}

Load R_x, C

Inc/Dcr R_x

Store C, R_x

G 2023

→ BSEM : 12

→ CSEM : 7

Max (C) = 2

Min (C) =

a) -2 ✓ b) -1 c) 0 d) None

2

S = ~~2~~ ~~1~~ ~~2~~ ~~1~~ ~~0~~ 1 ~~0~~

P_i : ✓

P_j : L, Inc

P_k : ✓

P_l : ✓

P_j : S : ✓

C $\boxed{\cancel{1}\cancel{0}}$ 2

$\boxed{2\cancel{1}}$ R_j

Classical IPC Problems

1) Producer-Consumer:

```
#define N 100
int Buffer[N];
CSEM Empty = N;
```

<No. of Empty Slots>

```
CSEM full = 0;
```

<No. of full Slots (≈ data items)>

```
BSEM mutex = 1;
```

<used b/w P & C to ensure Mut. Exclusion on Buffer>

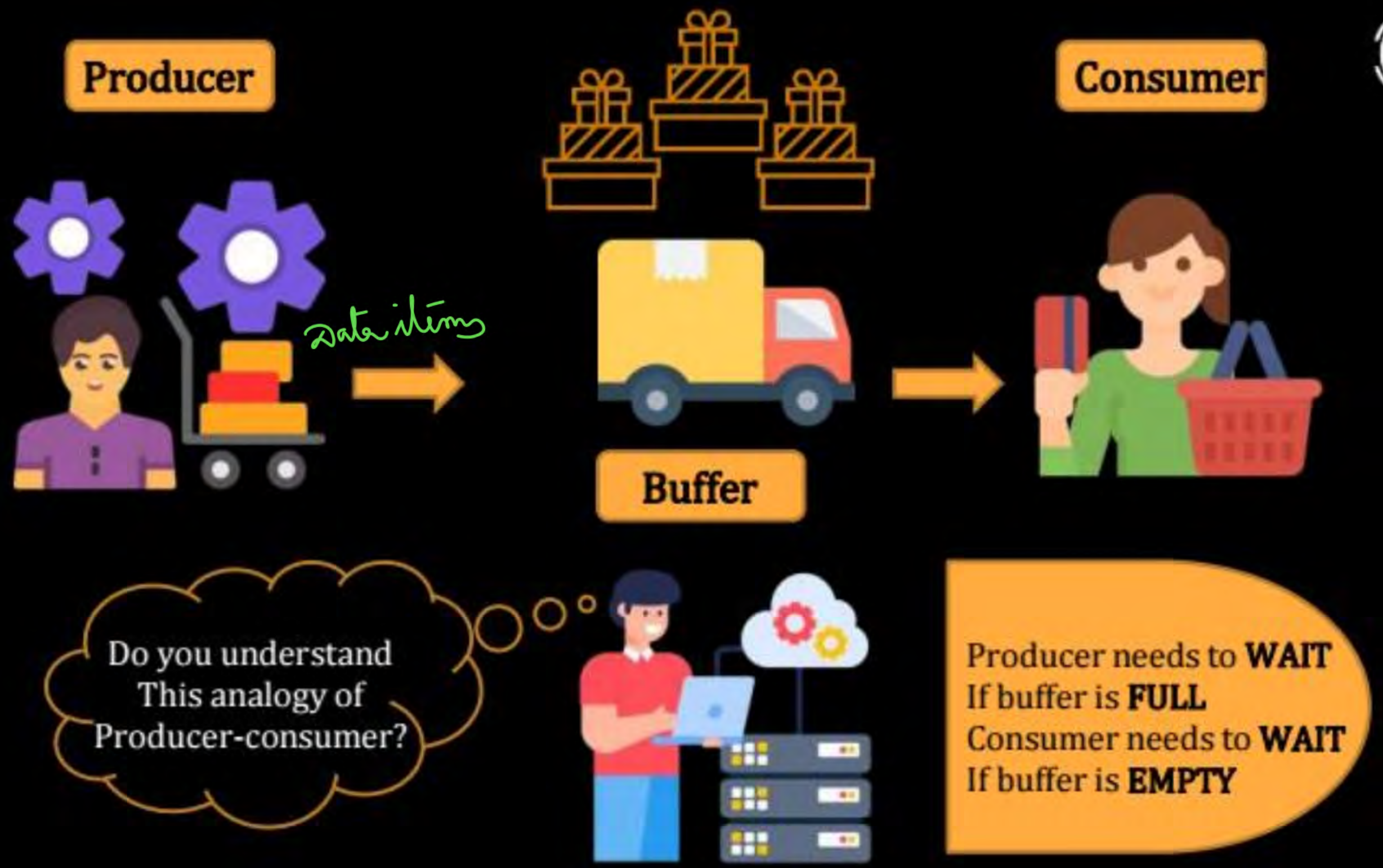
```
void Producer(void)
{
    int itemp, in = 0;
    while (1)
    {
    a) itemp = Produce_item();
    b) DOWN(Empty);
    c) DOWN(mutex);
    d) Buffer[in] = itemp;
    e) in = (in+1) % N;
    f) up(mutex);
    g) up(full);
    }
}
```

```
void Consumer(void)
{
    int itemc, out = 0;
    while (1)
    {
    a) DOWN(full);
    b) DOWN(mutex);
    c) itemc = Buffer[out];
    d) out = (out+1) % N;
    e) up(mutex);
    f) up(Empty);
    g) Process_item(itemc);
    }
}
```

Empty = N

full = ~~0~~ -1

mutex = ~~1~~ 0


Producer
Consumer
data items
Buffer
Do you understand
This analogy of
Producer-consumer?
Producer needs to **WAIT**
If buffer is **FULL**
Consumer needs to **WAIT**
If buffer is **EMPTY**

Consider the following solution to the producer-consumer synchronization problem. The shared buffer size is N. Three semaphores *empty*, *full* and *mutex* are defined with respective initial values of 0, N and 1. Semaphore *empty* denotes the number of available slots in the buffer, for the consumer to read from. Semaphore *full* denotes the number of available slots in the buffer, for the producer to write to. The placeholder variables, denoted by P, Q, R and S, in the code below can be assigned either *empty* or *full*. The valid semaphore operations are: *wait()* and *sigmal()*.

Producer:	Consumer:

`do{`
`    wait(P);` full
`    wait(mutex);`
`    //Add item to buffer`
`    signal(mutex);`
`    signal(Q);` Empty
`}while(1);`
	`do{`
	`    wait(R);` Empty
	`    wait(mutex);`
	`    //Consume item from buffer`
	`    signal(mutex);`
	`    signal(S);` full
	`}while(1);`

Which one of the following assignments to P, Q, R and S will yield the correct solution?

(A) P: *full*, Q: *full*, R: *empty*, S: *empty*

(B) P: *empty*, Q: *empty*, R: *full*, S: *full*

(C) P: *full*, Q: *empty*, R: *empty*, S: *full*

(D) P: *empty*, Q: *full*, R: *full*, S: *empty*

Consider the procedure below for the Producer-Consumer problem which uses semaphores:

```
semaphore n = 0; -1 <No-of Data-items>
semaphore s = 1; 0
void producer()
{
      while(true)
      {
       produce();
       semWait(s); Blocked
       addToBuffer();
       semSignal(s);
       semSignal(n);
      }
}
```

```
void consumer()
{
      while(true)
      {
       semWait(s);
       semWait(n); Blocked
       removeFromBuffer();
       semSignal(s);
       consume();
      }
}
```

Which one of the following is TRUE?

(A) The producer will be able to add an item to the buffer, but the consumer can never consume it.

(B) The consumer will remove no more than one item from the buffer.

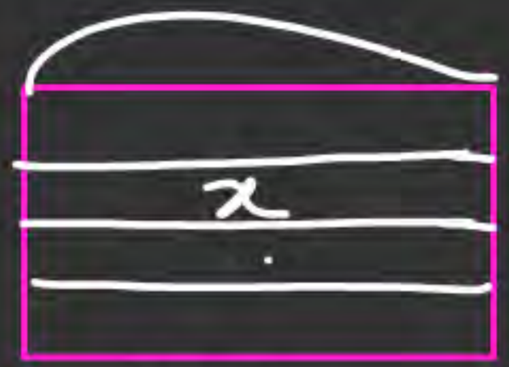
✓ **(C)** Deadlock occurs if the consumer succeeds in acquiring semaphore s when the buffer is empty.

(D) The starting value for the semaphore n must be 1 and not 0 for deadlock-free operation.

2) Reader-Writer Problem

R ✓   W ✓

r_1
r_2
r_3
⋮
r_n



DB
<records>

w_1
w_2
w_3
⋮
w_m

BSEM mutex=1;

```
{
    P(mutex)
      <DB>
    V(mutex);
}
```

First R-W Problem: Starv. to writers

t_1: r_1 & w_1 : Any-one
t_2: r_1 ✓ ; w_1 wait
t_3: r_2 ✓
t_4: r_3 ✓

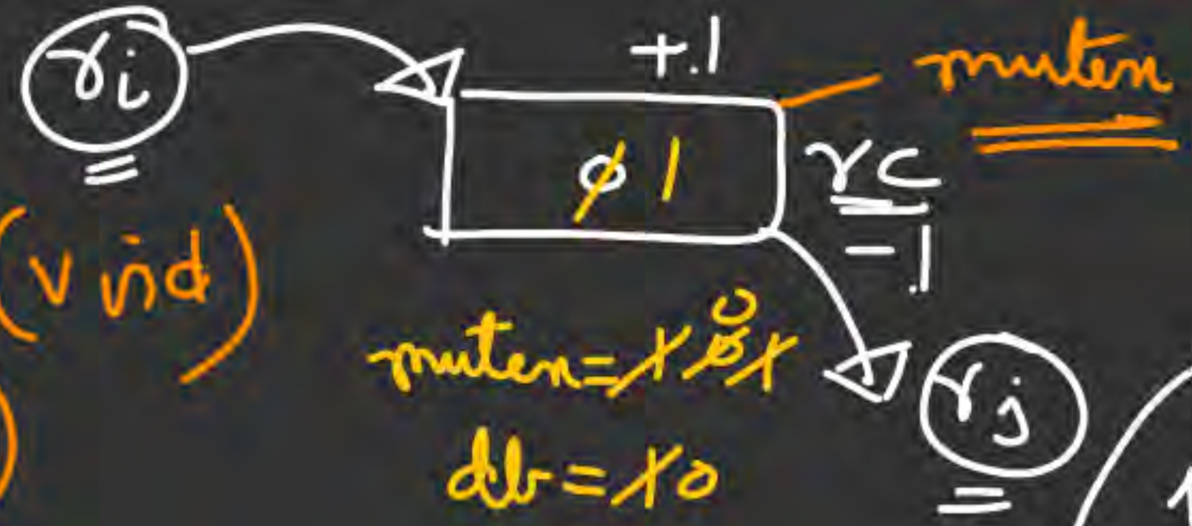
DB
r_1
r_2 r_3

---

t_1: r_1 & w_1 : Any one
t_2: w_1 ✓ ; r_1 wait
t_3: w_2 × ; wait
t_4: w_3 × : wait
t_5: r_2 × : wait

w_1
DB

Impl. of First R-W – using Semaphores



mutex = ~~1~~ ~~0~~ 1

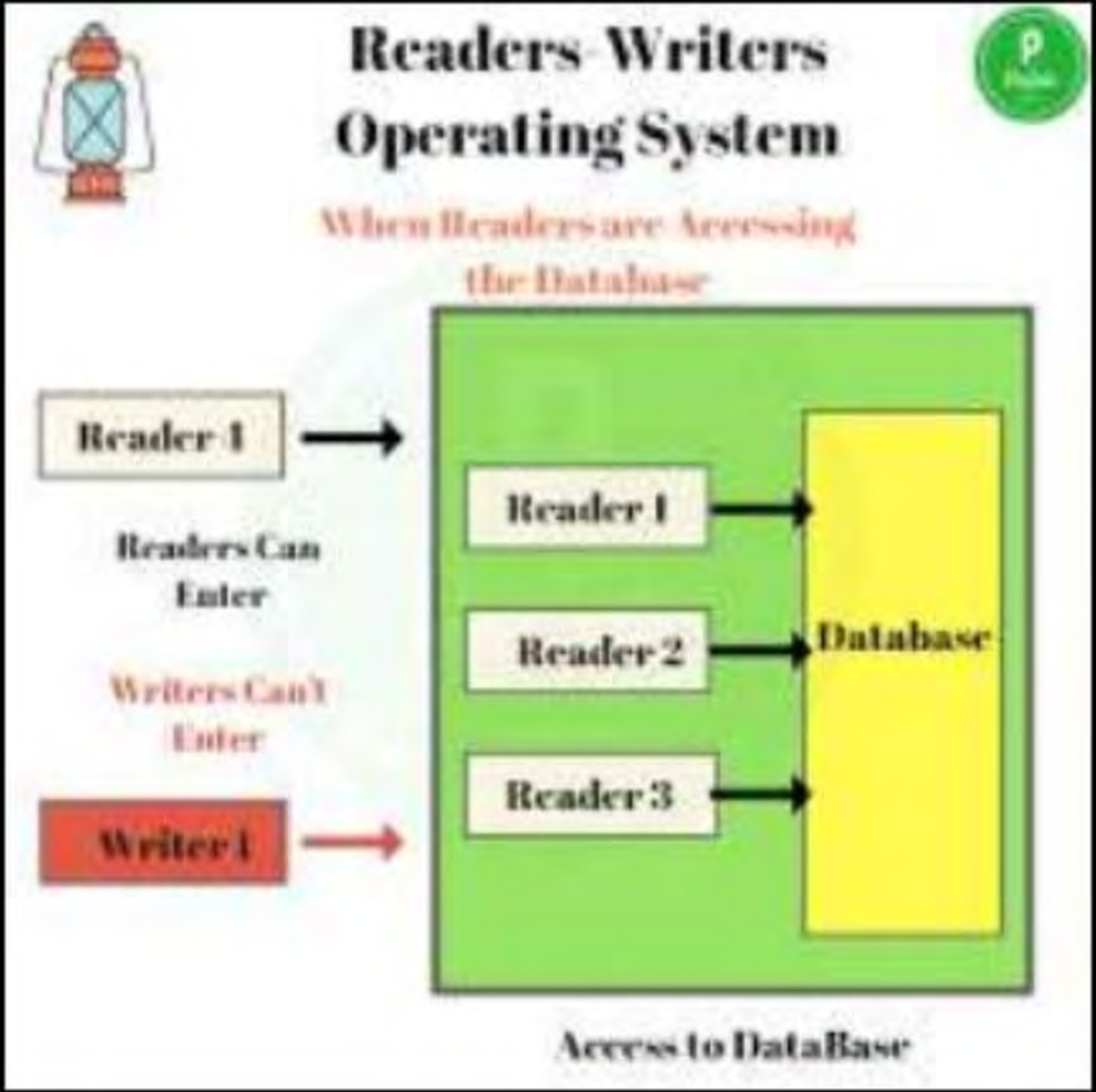
db = ~~1~~ 0

```
int rc = 0;  // Reader's Count //
BSEM mutex = 1;
  <used by R's to update rc>
BSEM db = 1;
  <used by R's & W's to
      access <DB> as CS>
```

```
void writer(void)
{
   while(1)
   {
      a) DOWN(db);
      b) <DB-WRITE>
      c) UP(db);
   }
}
```

```
void Reader(void)
{
   while(1)
   {
      a) DOWN(mutex);
      b)   rc = rc+1;
      c) if(rc==1) DOWN(db);
      d) up(mutex);
      e) <DB-READ>
      f) DOWN(mutex);
      g) rc = rc-1;
      h) if(rc==0) up(db);
   } (i) up(mutex);
}
```

1st reader is the Leader of all other Readers


Readers-Writers
Operating System
When Readers are Accessing
the Database
Reader 1
Readers Can
Enter
Writers Can't
Enter
Writer 1
Reader 1
Reader 2
Reader 3
Database
Access to DataBase


Readers-Writers
Operating System
When Writer is Writing
in the Database
Reader 1
Readers Can't
Enter
Writers Can't
Enter
Writer 2
Writer 1
Database
Access to DataBase

Synchronization in the classical readers and writers problem can be achieved through use of semaphores. In the following incomplete code for readers-writers problem, two binary semaphores mutex and wrt are used to obtain synchronization

db

```
wait (wrt)
writing is performed
signal (wrt)
```

writer

```
wait (mutex)
readcount = readcount + 1
if readcount = 1 then S1
S2
reading is performed
S3
readcount = readcount - 1
if readcount = 0 then S4
signal (mutex)
```

wait (wrt);

Reader

The values of S1, S2, S3, S4, (in that order) are

X (A) signal (mutex), wait (wrt), signal (wrt), wait (mutex)

X (B) signal (wrt), signal (mutex), wait (mutex), wait (wrt)

✓ (C) wait (wrt), signal (mutex), wait (mutex), signal (wrt)

X (D) signal (mutex), wait (mutex), signal (mutex), wait (mutex)

3. DINING PHILOSOPHER'S
Researcher PROBLEM

'N'-Philosophers $(N \geq 2)$

Spaghetti (Noodles)

deadlock.

All → Hungry

```
#define N 5
void Philosopher(int i)
{
    while (1)
    {
        a) Think(i);
        b) Take-fork(i);
        c) Take-fork((i+1)%N);
        d) eat(i);
        e) put-fork(i);
        f) put-fork(i+1)%N;
    }
}
```

Pre → b), c)

$P_0 \leftarrow f_0$
$P_1 \leftarrow f_1$
$P_2 \leftarrow f_2$
$P_3 \leftarrow f_3$
$P_4 \leftarrow f_4$

Without deadlock, for $N = 5$, what is the max # of Philosophers that can be eating ____;

Max. Concurrency

$P_0 : f_0; f_1$ ✓

$P_1 :$ ✗

$P_2 : f_2; f_3$ ✓

$P_3 : f_3$ ✗

$P_4 : f_4; f_0$ ✗

$N = 6$

$= 3$

$P_0 : f_0; f_1;$ ✓

$P_1 : f_1$ ✗

$P_2 : f_2; f_3:$ ✓

$P_3 : f_3;$ ✗

$P_4 : f_4; f_5:$ ✓

$P_5 : f_5$ ✗

How to Prevent/Avoid deadlock in D.P

Semaphore based

→ put Semaphore control on taking & releasing the forks

Non-Semaphore based :

deadlock-free opn

→ out of 'N'-Philosophers,
let $(N-1) \rightarrow \underline{L}-\underline{R}$
& $\underline{1} \rightarrow R-L$

→ Even Numbr'd : L - R
odd " : R - L

Q1) A solution to the Dining Philosophers Problem which avoids deadlock is:

(A) ensure that all philosophers pick up the left fork before the right fork ✗

(B) ensure that all philosophers pick up the right fork before the left fork ✗

(C) ensure that one particular philosopher picks up the left fork before the right fork, and that all other philosophers pick up the right fork before the left fork (N-1) & 1 ✓

L-R RL

(D) None of the above ✗

Let m[0] ....m[4] be mutexes (binary semaphore) and P0 ..... P4 be processes. Suppose each process P[i] executes the following:

Q2)

```
wait (m[i]) ;    wait (m[i+1] mod 4)
CS
release (m[i]);    release (m[i+1] mod 4)
```

Which situation could be came

a) Thrashing

b) Deadlock ✓

c) Starvation but no deadlock

d) None

Q.13

First Reader-Writer using Semaphore with Busy Waiting

H/W

G

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

```
Void Writer ( Void)
    {
     L2: P(mutex);
     if(__________) - ②
      {
        V(mutex);
        goto L2;
      }
        else
        {
           W=1;
           V(mutex);
        }
    <DB_WRITE>
    P(mutex);
        W=0;
    V(mutex);
    }
```

Q

______ ①

______ ②

CONCURRENCY-MECHANISMS

THANK YOU!