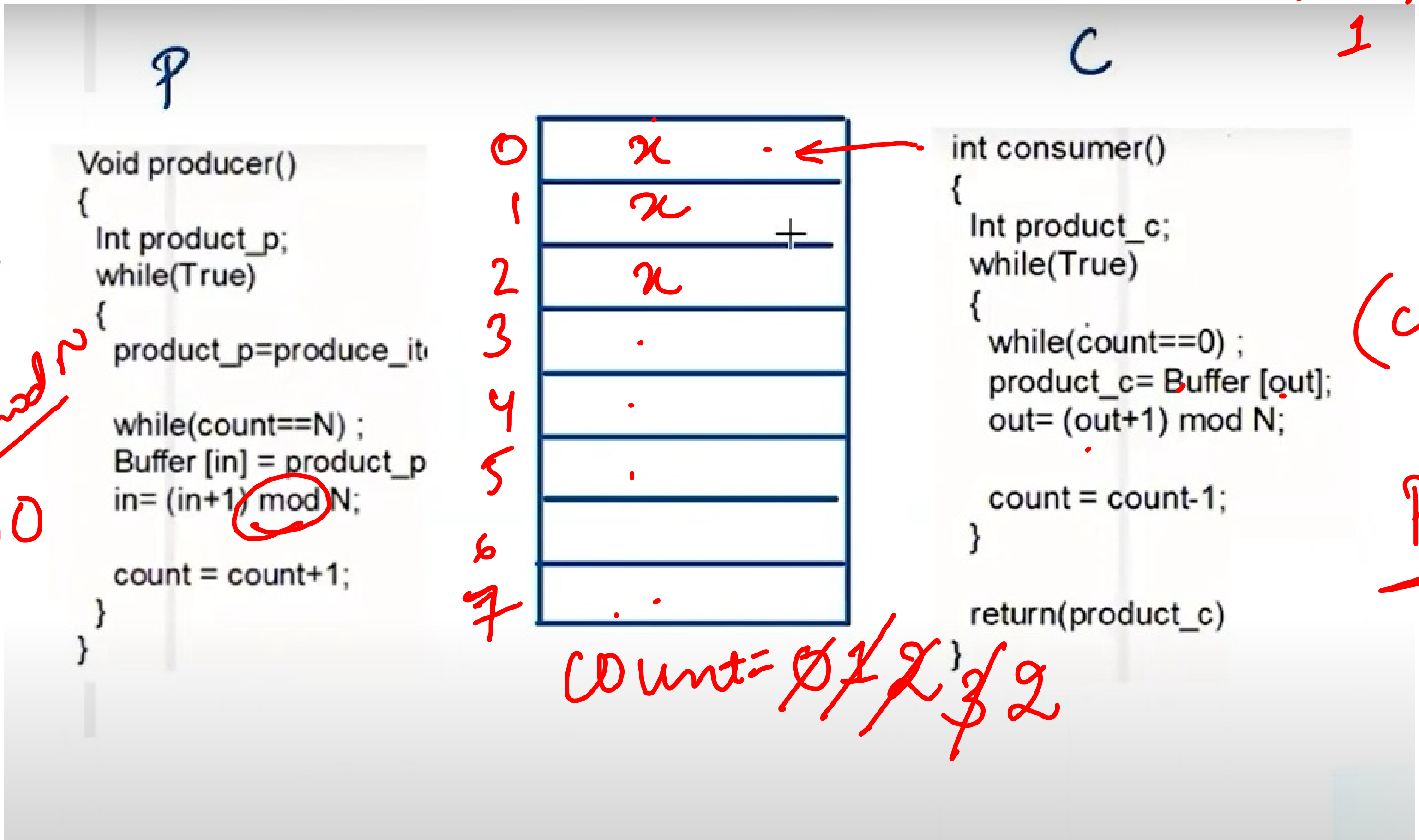
In=$\cancel{0}\cancel{1}\cancel{2}3$

Producer - consumer

Buffer=8

out=$\cancel{0}$ 1



```
Void producer()
{
  Int product_p;
  while(True)
  {
   product_p=produce_it
   
   while(count==N) ;
   Buffer [in] = product_p
   in= (in+1) mod N;
   
   count = count+1;
  }
}
```

```
int consumer()
{
  Int product_c;
  while(True)
  {
   while(count==0) ;
   product_c= Buffer [out];
   out= (out+1) mod N;
   
   count = count-1;
  }
  
  return(product_c)
}
```

count=8

in=(in+1)mod n

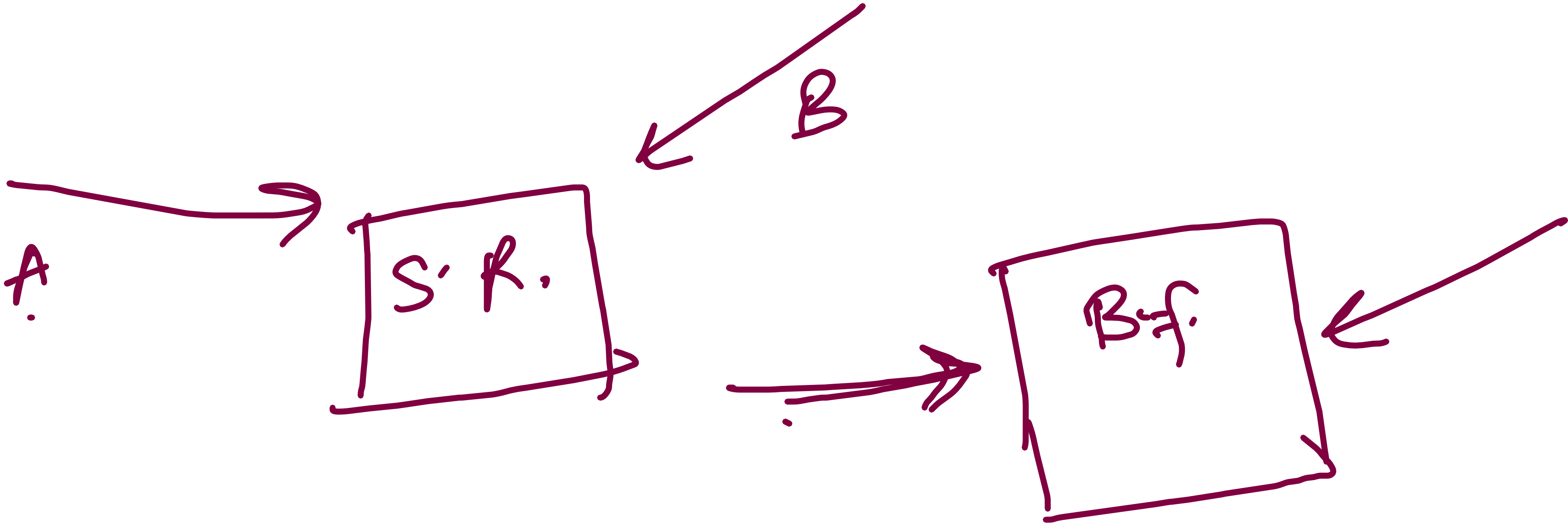
8%8 ⇒ 0

count=$\cancel{0}\cancel{1}\cancel{2}\cancel{3}2$

(count==0);

P→P→P→C

→P


A.
B
S. R.
B.f.

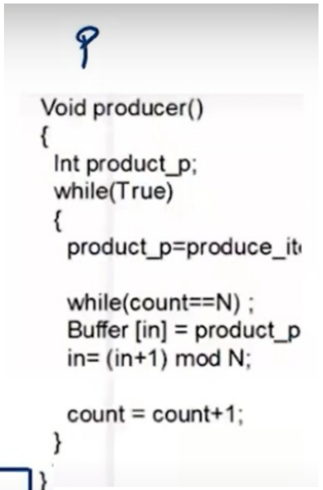
in=~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

out=~~0~~ 1

count+1

temp= count+1

count = temp

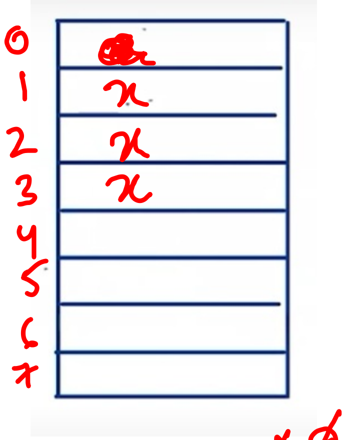


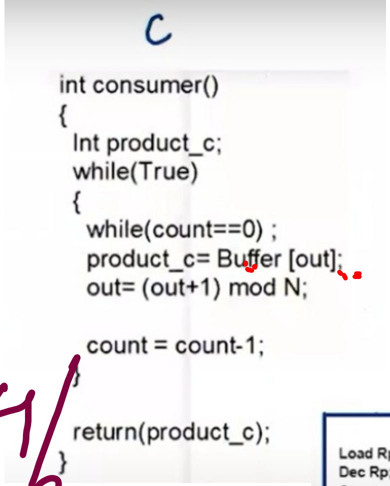
P

```
Void producer()
{
  Int product_p;
  while(True)
  {
    product_p=produce_it

    while(count==N) ;
    Buffer [in] = product_p
    in= (in+1) mod N;

    count = count+1;
  }
}
```





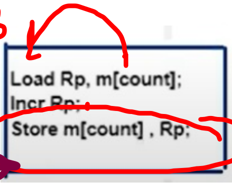
C

```
int consumer()
{
  Int product_c;
  while(True)
  {
    while(count==0) ;
    product_c= Buffer [out];
    out= (out+1) mod N;

    count = count-1;
  }

  return(product_c);
}
```

3

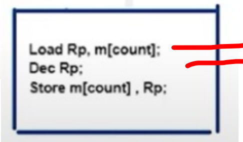


```
Load Rp, m[count];
Incr Rp;
Store m[count] , Rp;
```

pre empt

count=~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 2

Buffer=8=N



```
Load Rp, m[count];
Dec Rp;
Store m[count] , Rp;
```

temp= count

temp = 3-1=2

count

pre empt

2

count+1

temp=3.

temp=3+1=4

count= temp

4

4/2

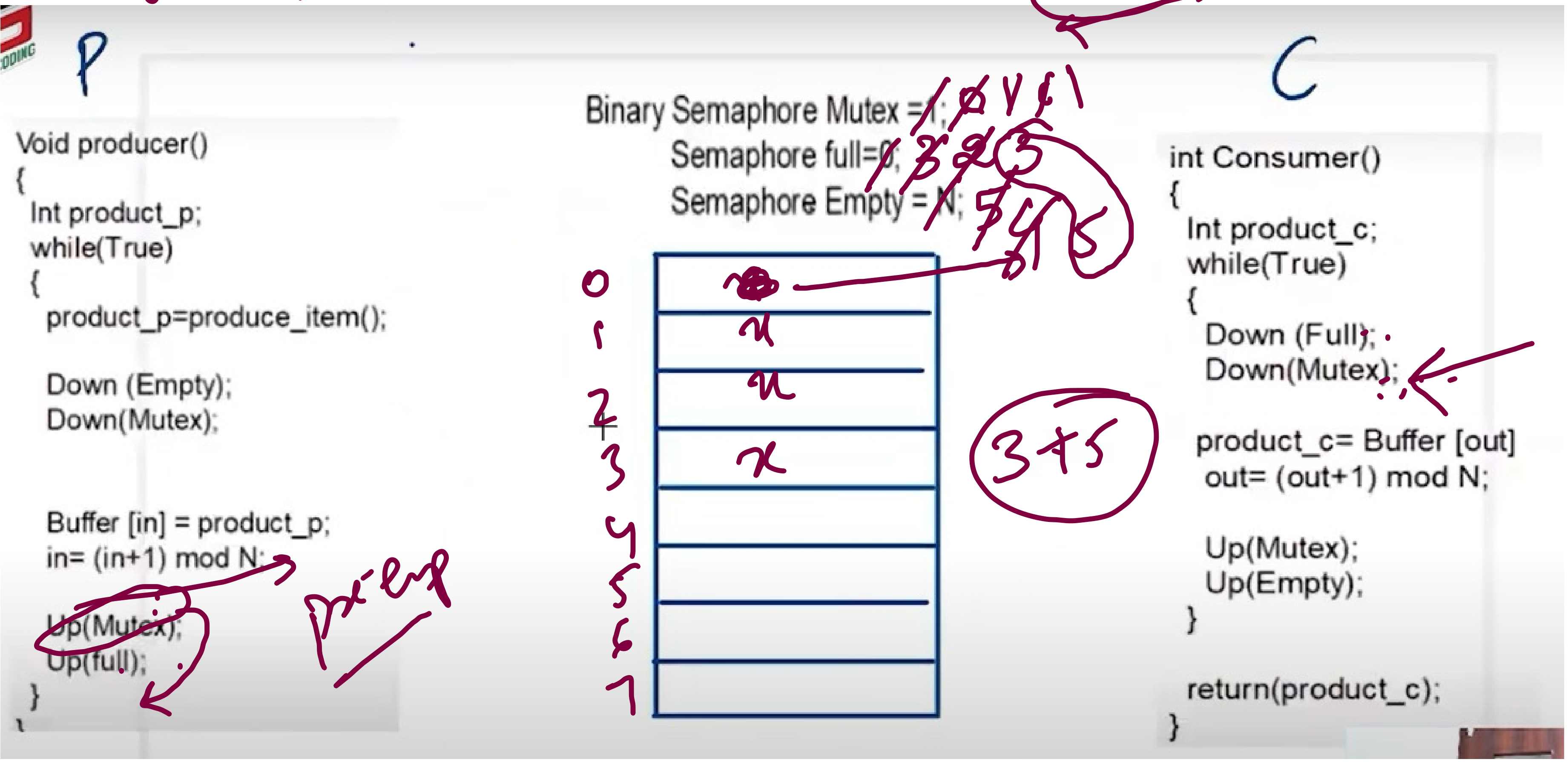
Race condition

4/2

in = ~~0~~ 3 4

C+B

out = ~~0~~ 1



P

```
Void producer()
{
 Int product_p;
 while(True)
 {
  product_p=produce_item();

  Down (Empty);
  Down(Mutex);

  Buffer [in] = product_p;
  in= (in+1) mod N;

  Up(Mutex);
  Up(full);
 }
```

pr-ep

```
Binary Semaphore Mutex =1;
         Semaphore full=0;
         Semaphore Empty = N;
```

~~0~~ 1 ~~0~~ 1

~~1~~ ~~2~~ 3

7 ~~6~~ 5

3+5

C

```
int Consumer()
{
 Int product_c;
 while(True)
 {
  Down (Full);
  Down(Mutex);

  product_c= Buffer [out]
  out= (out+1) mod N;

  Up(Mutex);
  Up(Empty);
 }

 return(product_c);
}
```

1/0

M.S.

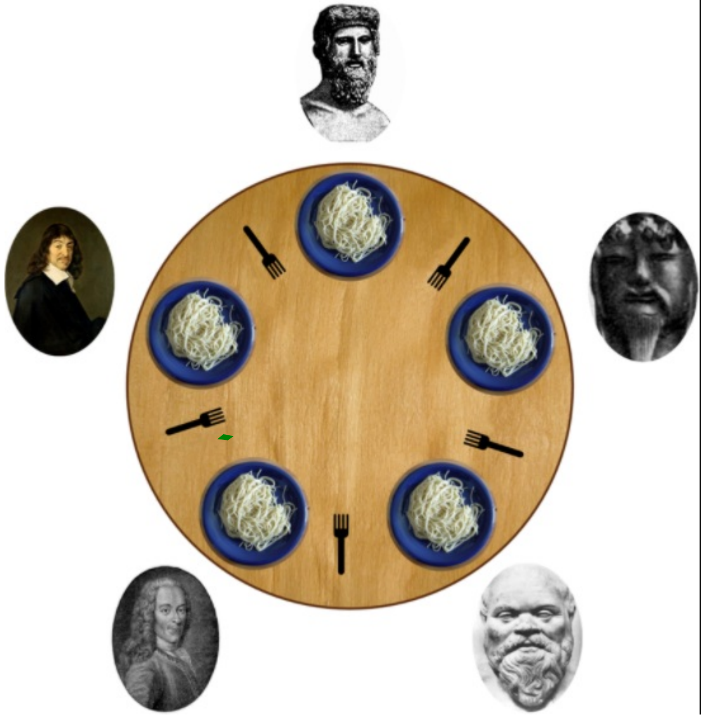
count =

Bigger problem

Dining philosphers

5

0



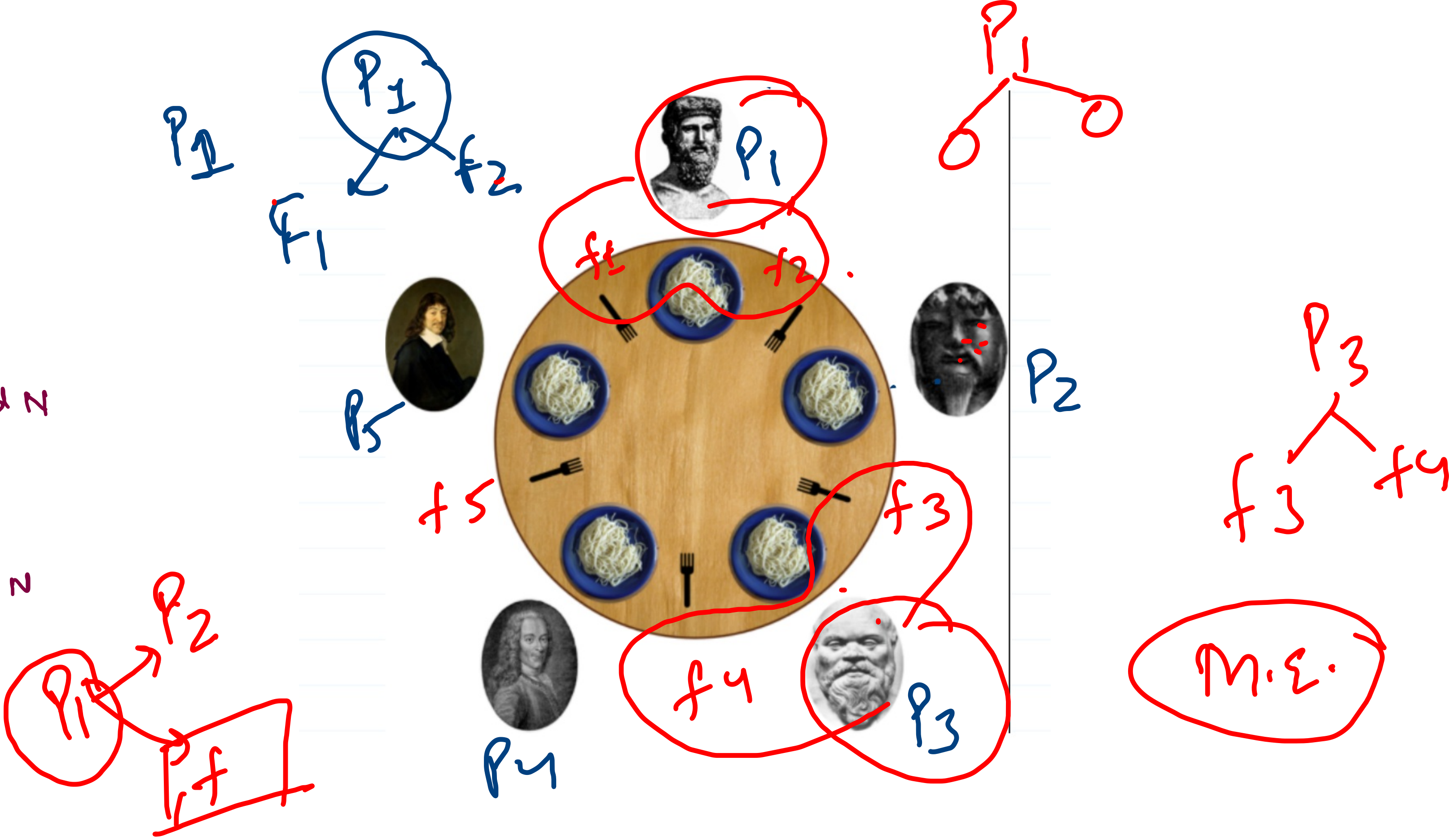
→ Eat

— Think

Semaphores

D.L.

```
void philospher()
{
 while(1)
  {
   Think()
    take_fork(i).
    take_fork (i+1) mod N
    Eating () →
     Put_fork(i)
     Put_fork(i+1) mod N
  }
}
```



```
void philospher()
{
 while(1)
 {
   Think()
   wait(take-fork(Si))
   wait(take-fork(Si+1)%n)
   Eating()
   (Put-fork(i))
  (Put-fork(i+1) mod N)
 }
}
```

S1 S2 S3 S4 S5

~~1~~ ~~0~~ 1 0 ~~1~~ ~~0~~ 1 0 ~~1~~ 0 ~~1~~ 0 ~~1~~ 0

~~P1~~ — ~~f1~~, ~~f2~~

→ P2 — f2 wait state

P3 — F3, F4

P4 → wait state

P5 → f5, ~~wait~~ f1

f0

P0

P1

P2

P3

P4

```
void philospher()
{
  while(1)
   {
     Think()
       wait(take-fork(Si))
       wait(take-fork(Si+1)%8)
      Eating()
Sig  (Put_fork(i))
sig:  ( Put_fork(i+1) mod N )
   }
 }
```

S1	S2	S3	S4	S5
0 ~~1~~	0 ~~1~~	~~1~~ 0	~~1~~ 0	1

I II

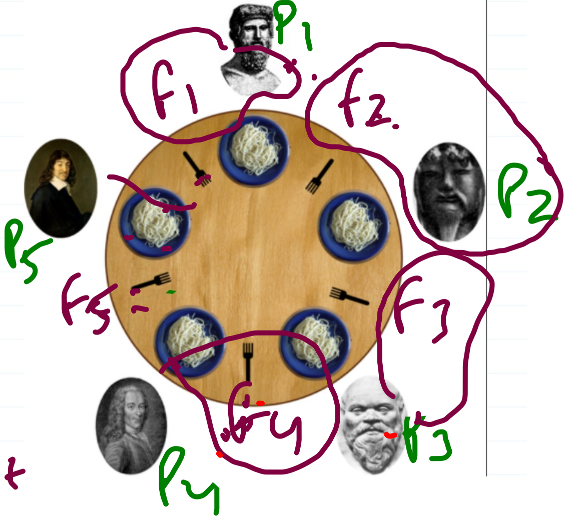
P1 — f1, wait

P2 → f2, wait

P3 – f3 – wait

P4 - f4 - wait

P5 – f5, wait.



P5

P1 — f1, f2

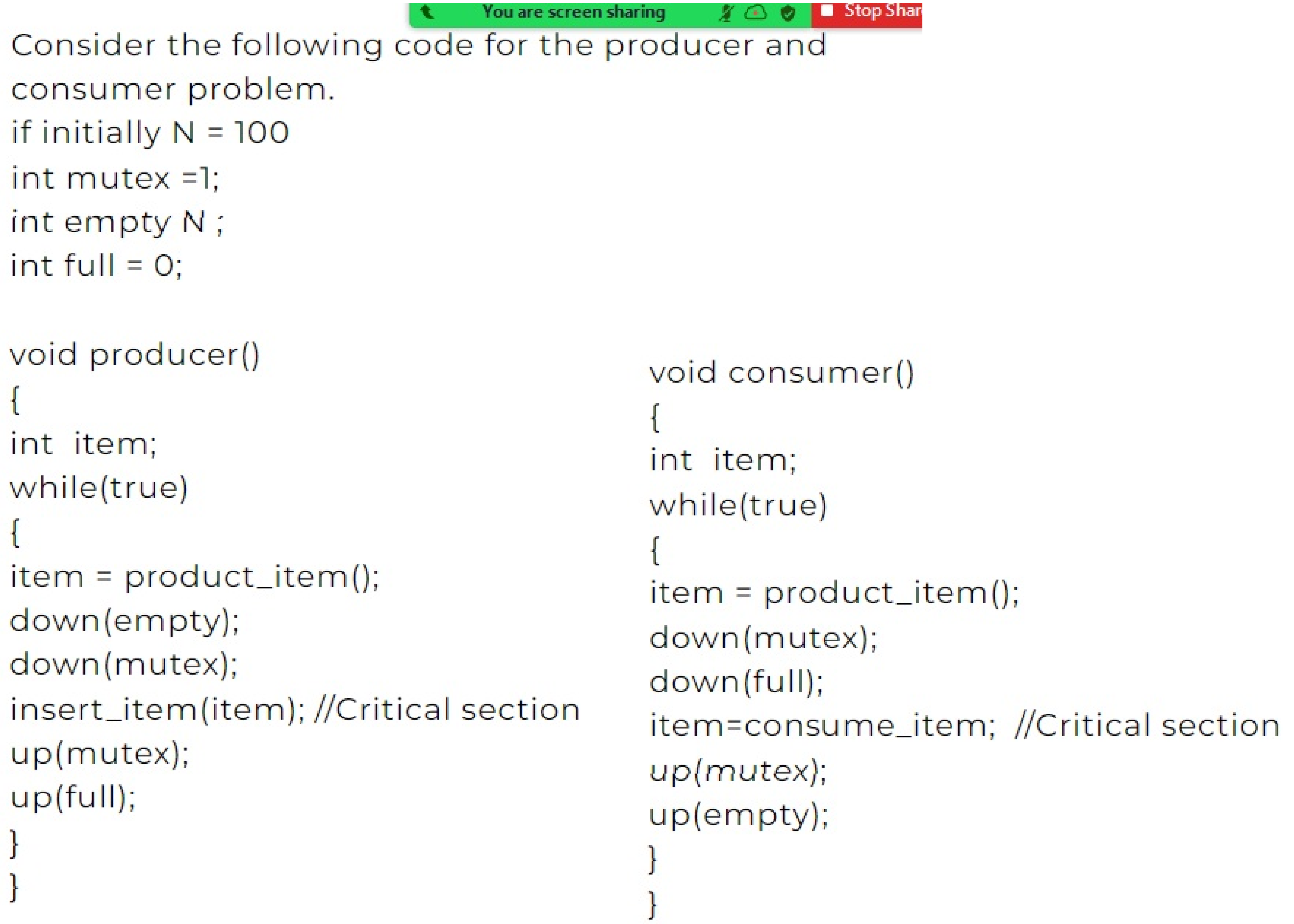
P2 — f2, f3

P3 — f3, f4

P4 — f4, f5.

P5 → X

f1 f5

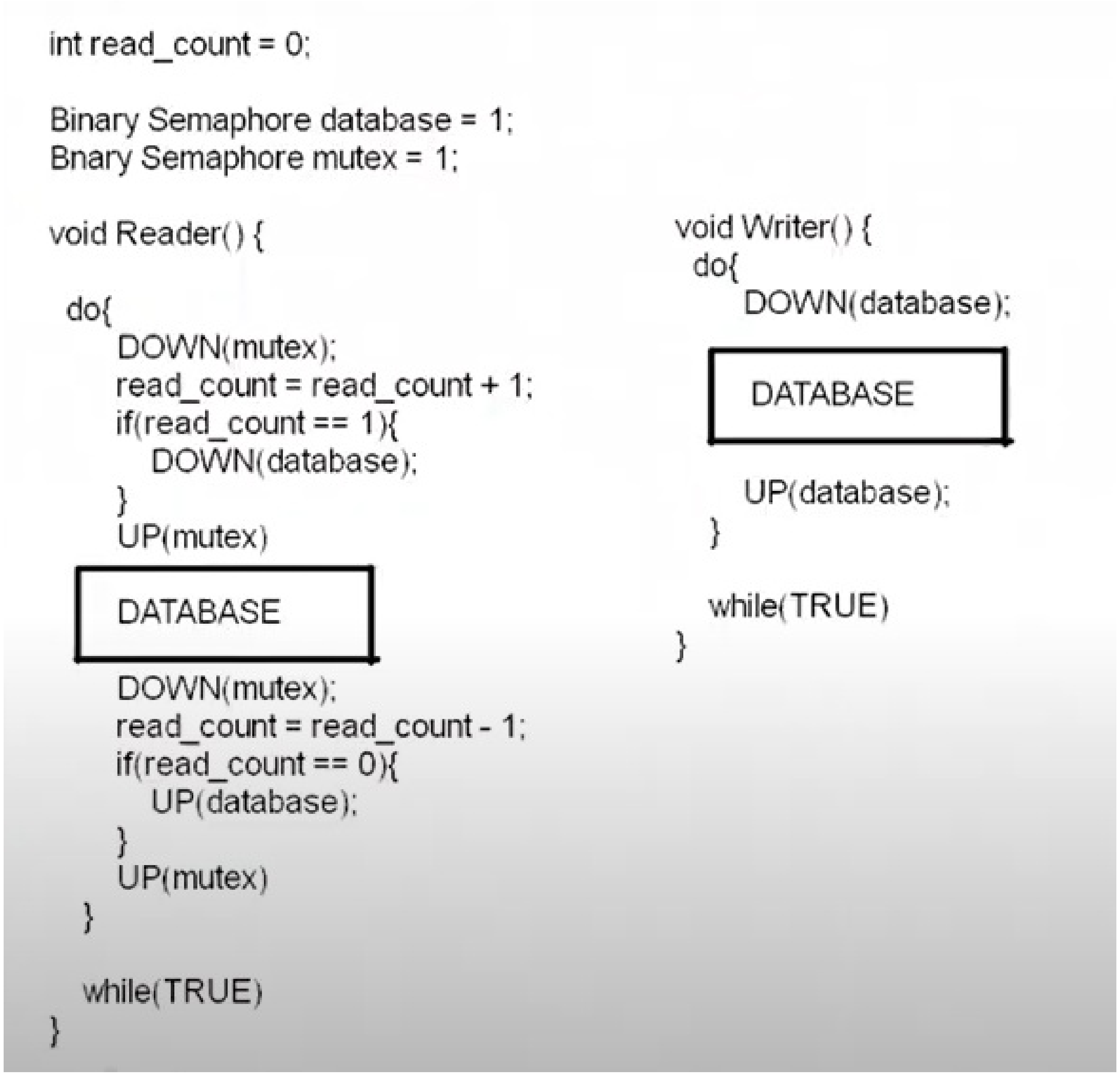


Consider the following code for the producer and consumer problem.

```
if initially N = 100
int mutex =1;
int empty N ;
int full = 0;
```

```
void producer()
{
int  item;
while(true)
{
item = product_item();
down(empty);
down(mutex);
insert_item(item); //Critical section
up(mutex);
up(full);
}
}
```

```
void consumer()
{
int  item;
while(true)
{
item = product_item();
down(mutex);
down(full);
item=consume_item();  //Critical section
up(mutex);
up(empty);
}
}
```



```
int read_count = 0;

Binary Semaphore database = 1;
Bnary Semaphore mutex = 1;

void Reader() {

  do{
      DOWN(mutex);
      read_count = read_count + 1;
      if(read_count == 1){
         DOWN(database);
      }
      UP(mutex)

      DATABASE

      DOWN(mutex);
      read_count = read_count - 1;
      if(read_count == 0){
         UP(database);
      }
      UP(mutex)
   }

   while(TRUE)
}
```

```
void Writer() {
  do{
      DOWN(database);

      DATABASE

      UP(database);
   }

   while(TRUE)
}
```