



University of
Pittsburgh

Introduction to Operating Systems

CS 1550



Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013**)

Announcements

- Upcoming deadlines
 - Homework 3 is due **this Friday**
 - Lab 1 is due on Tuesday 2/7 at 11:59 pm
 - Project 1 is due on Friday 2/17 at 11:59 pm
 - Discussed in this week's recitations
- AFS Quota
 - You can check it using the command **fs quota**
 - You can increase it from accounts.pitt.edu.
 - Check README of Lab 1

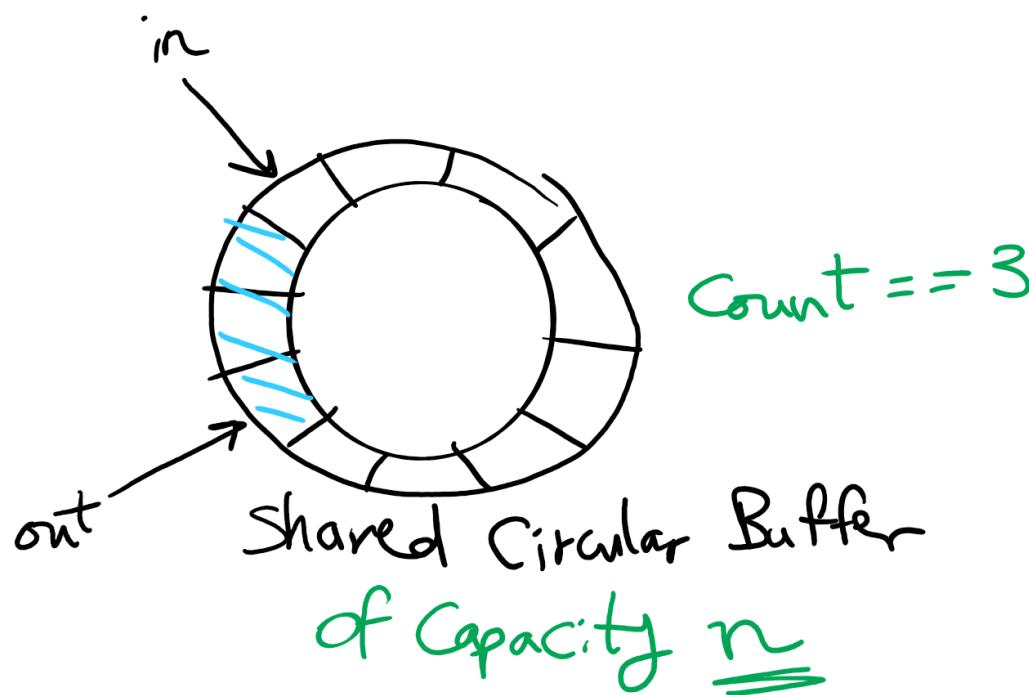
Previous lecture ...

- How processes (threads) are created and terminated
- Tracing programs with fork() calls

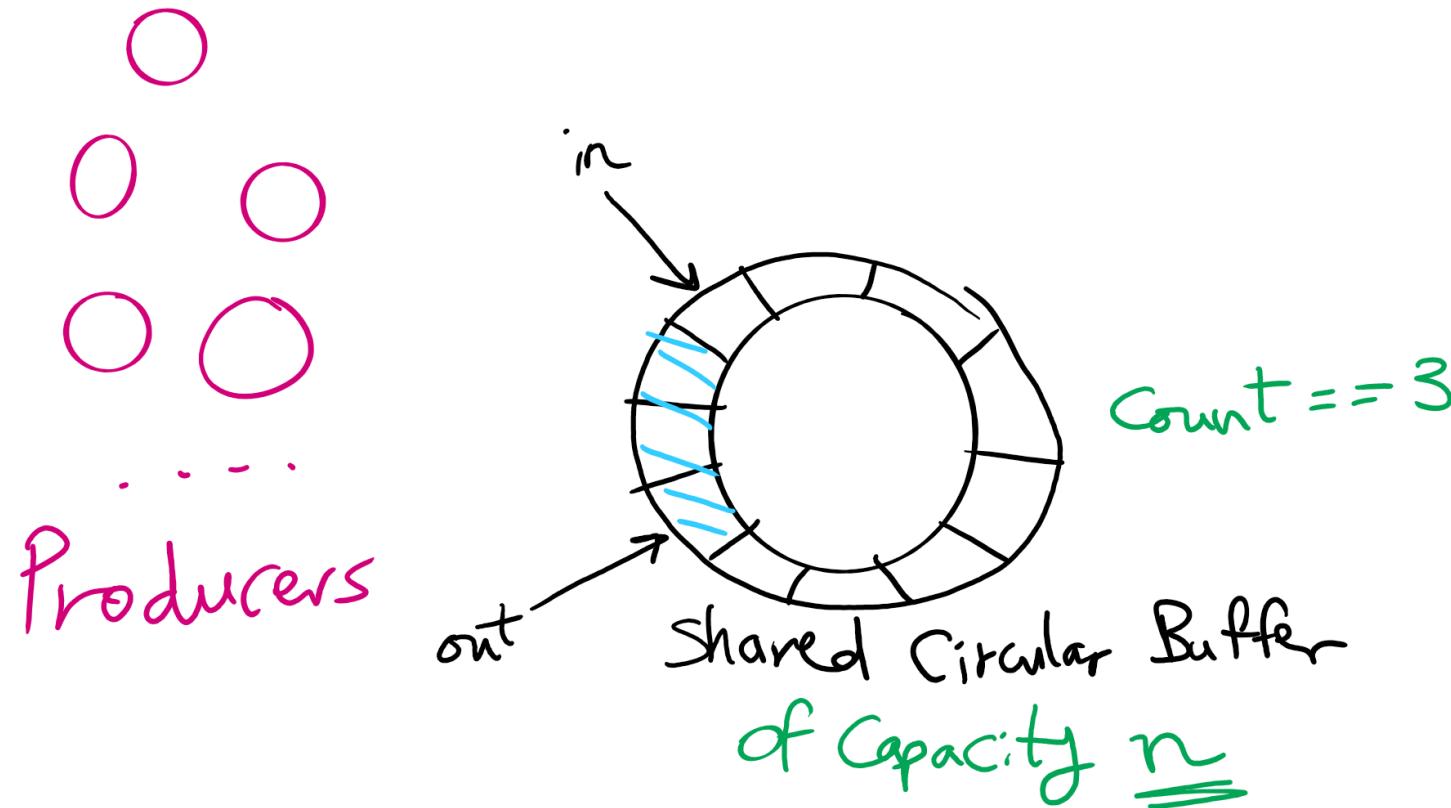
Problem of the Day

- Bounded Buffer Problem
 - aka Producers Consumers Problem
- A shared circular-array buffer with capacity n
- A set of *producer* processes/threads
- As set of *consumer* processes/threads
- Requirements:
 - Never exceed the buffer capacity
 - Producers wait if the buffer is full
 - Consumers wait if the buffer is empty

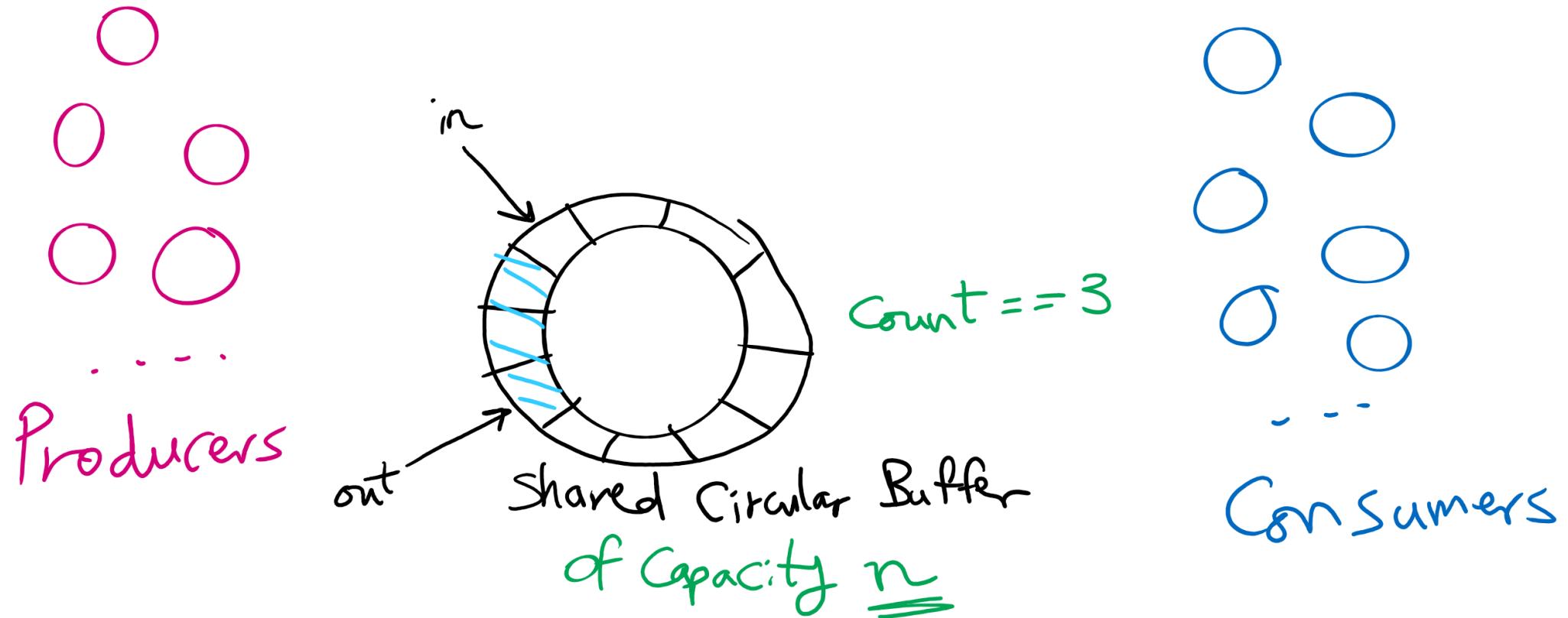
Producers Consumers Problem



Producers Consumers Problem



Producers Consumers Problem



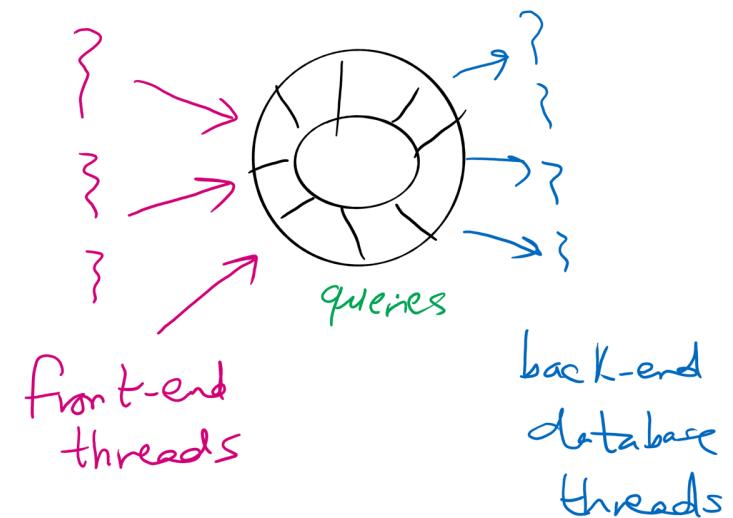
Producers Consumers Problem is everywhere!

- Access to User Interface elements in mobile applications
 - Main UI thread
 - Background threads
 - e.g., download files, check email on mail server
 - Background threads send requests to UI thread to update UI elements
 - Requests are stored in a shared bounded buffer
 - Which threads are the producers?
 - Who threads are the consumers?

Producers Consumers Problem is everywhere!

- Web Server

- front-end processes/threads that interact with the HTTP connection from the client (e.g., browser)
- Back-end processes/threads that execute database queries
- Queries are inserted by front-end processes into a shared buffer
- Which threads are the producers?
- Who threads are the consumers?



Solving Producers Consumers using Semaphores

Semaphore empty($\leq n$), full(0)
Mutex Sem(1);

Solving Producers Consumers using Semaphores

Semaphore empty($\leq n$), full(0)
Mutex Sem(1);

Producer

down(empty)

down(Sem)

buffer[in] = new item

in += 1 % n

Count++

up(Sem)

up(full)

Solving Producers Consumers using Semaphores

Semaphore empty($\leq n$), full(0)
Mutex Sem(1);

Producer

down(empty)

down(Sem)

buffer[in] = new item

in += 1 % n

Count ++

up(Sem)

up(full)

Consumer

down(full)

down(Sem)

item = buffer[out]

out += 1 % n

Count --

up(Sem)

up(empty)

Let's trace the solution!

Let's define some events first

Producer arrives

Moves as far as possible until the solid line

Producer

down(empty)
down(sem)

buffer[in] = new item
in += 1 % n
Count++
up(sem)
up(full)

Producer enters

Moves as far as possible past the solid line and until the dashed line

Producer

down(empty)

down(sem)

buffer[in] = new item

in += 1 % n

Count++

up(sem)

up(full)



Producer leaves

Moves as far as possible until the dotted line

Producer

down(empty)

down(sem)

buffer[in] = new item

in += 1 % n

Count++

up(sem)

up(full)



Consumer arrives

Moves as far as possible until the solid line

Consumer

down(full)

down(sem)

item = buffer[out]

out += 1 % n

Count --

up(sem)

up(empty)

Consumer enters

Moves as far as possible past the solid line and until the dashed line

Consumer

down(full)

down(sem)

item = buffer[out]

out += 1 % n

Count --



up(sem)

up(empty)

Consumer leaves

Moves as far as possible past the dashed line and until the dotted line

Consumer

down(full)

down(sem)

item = buffer[out]

out += 1 % n

Count --

up(sem)

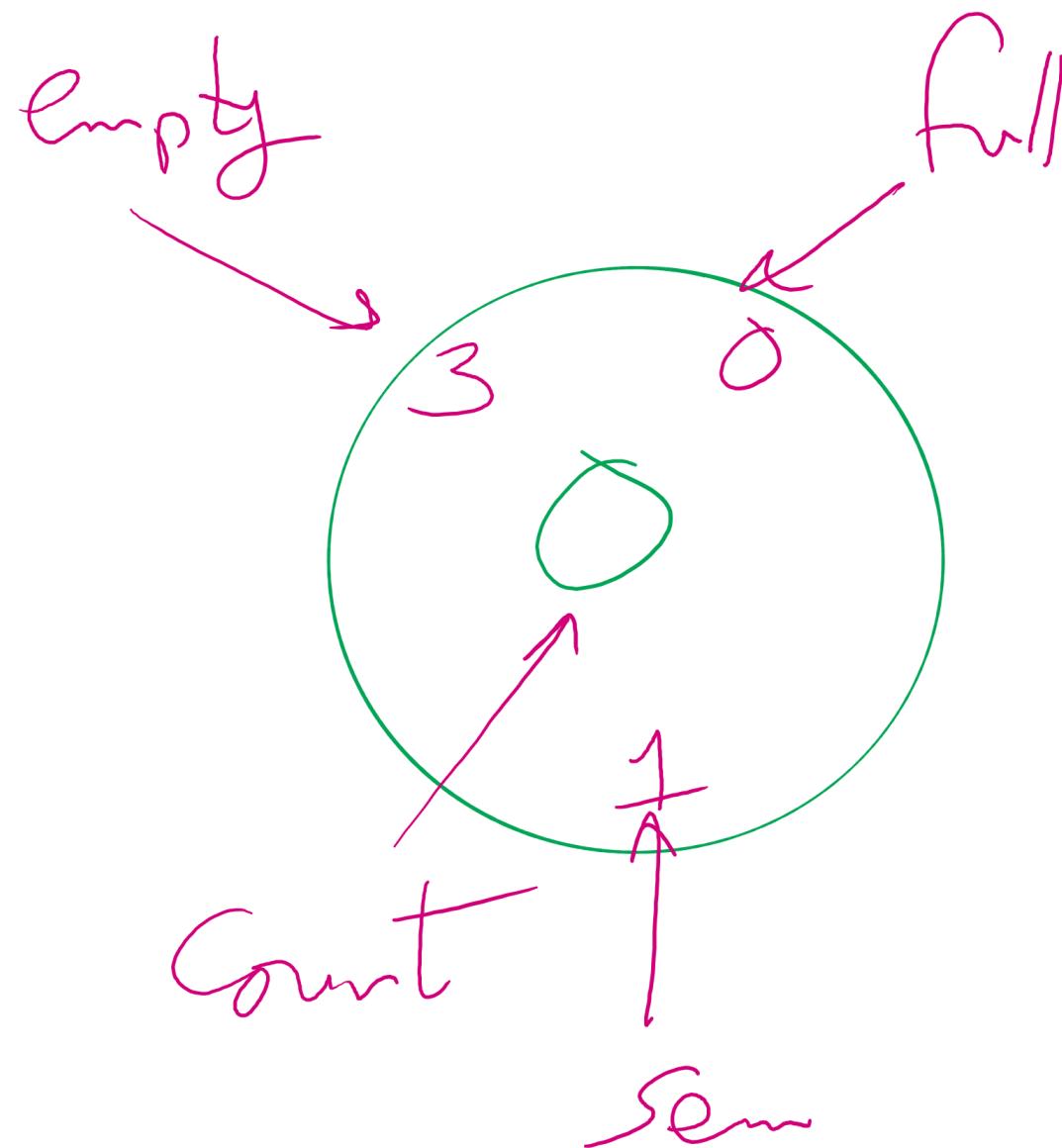
up(empty)



Tracing

Given a sequence of events, is the sequence *feasible*?
If yes, what is the *system state* at the end of the sequence?

System State



Example 1

$$\underline{n = 3}$$

Producer 0 arrives

Producer 0 enters

Producer 1 arrives

producer 2 arrives

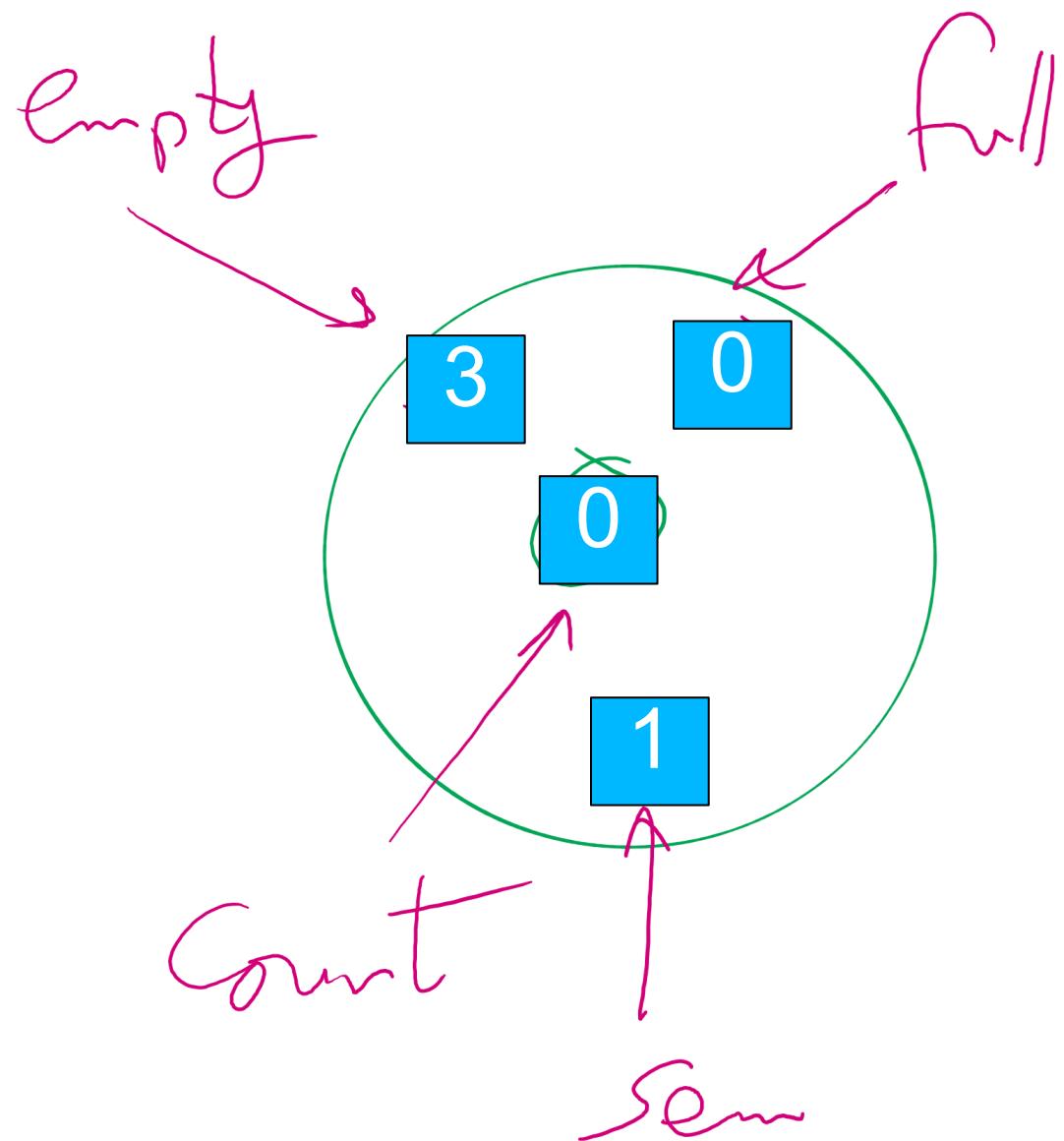
Consumer 0 arrives

producer 0 leaves

Consumer 0 enters

Consumer 0 leaves

Initial state



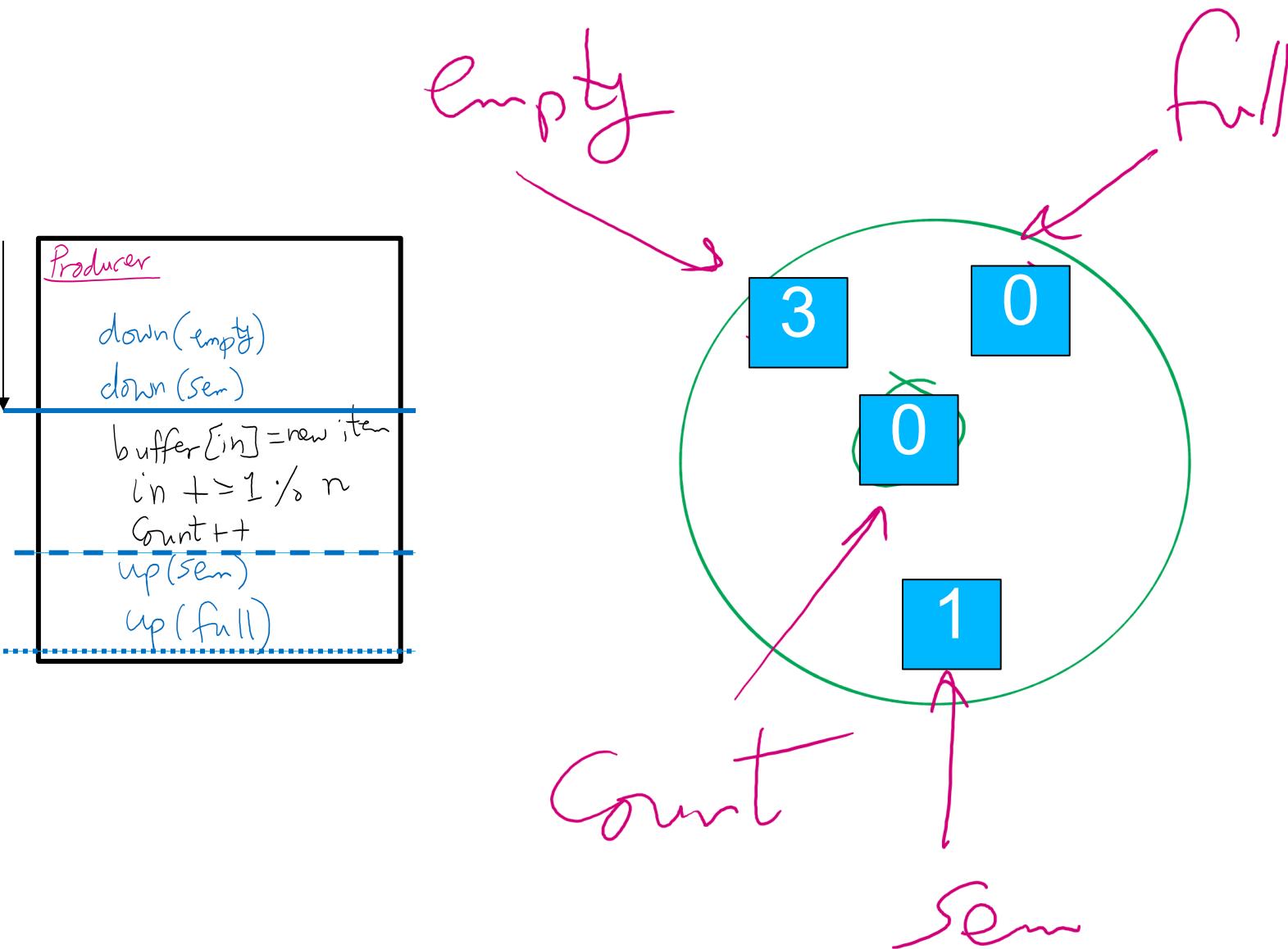
$$n = 3$$

Producer 0 arrives
Producer 0 enters
producer 1 arrives
producer 2 arrives
Consumer 0 arrives
producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

P0 arrives

$n = 3$

Producer 0 arrives
Producer 0 enters
producer 1 arrives
producer 2 arrives
Consumer 0 arrives
producer 0 leaves
Consumer 0 enters
Consumer 0 leaves



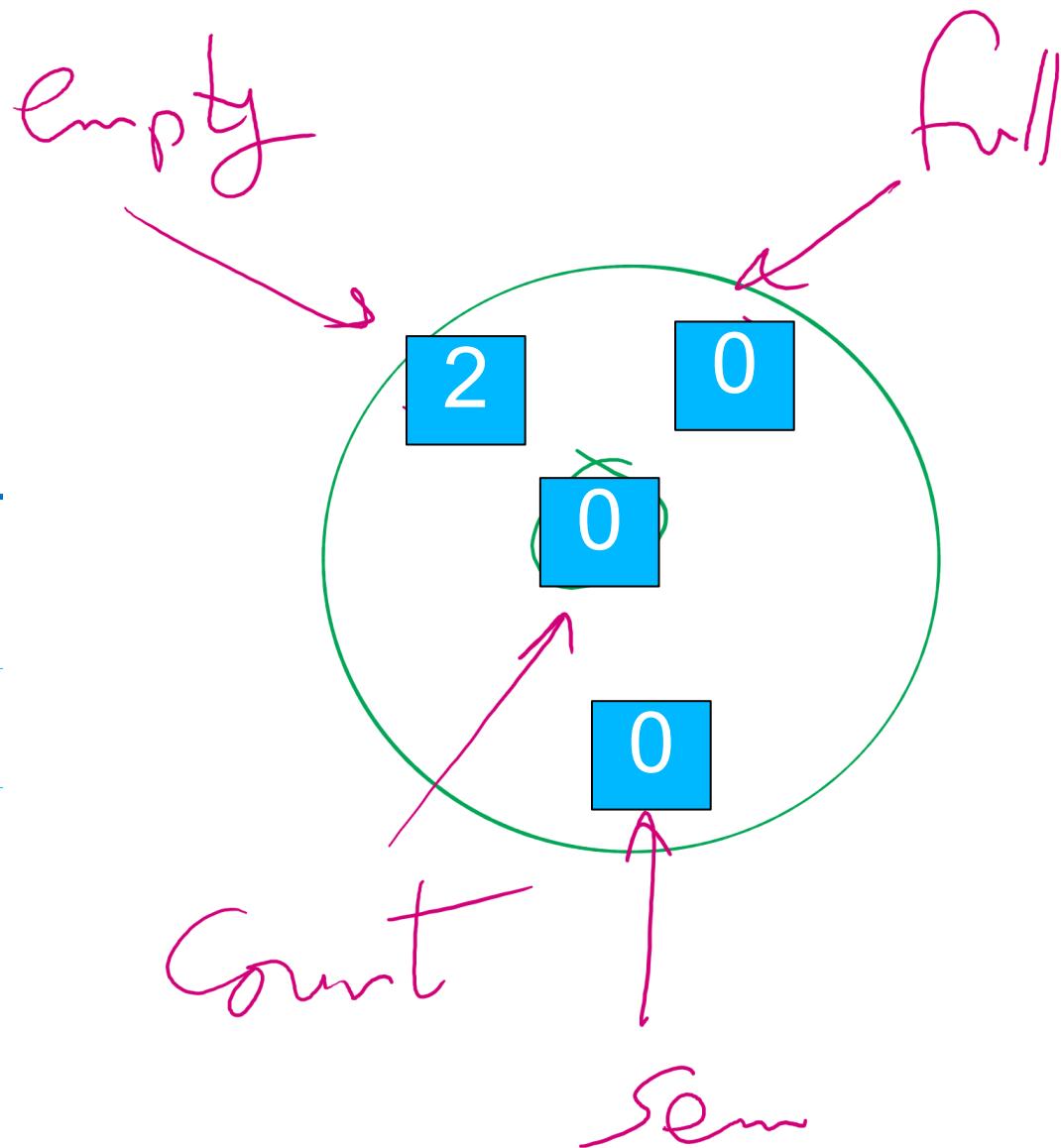
P0 arrives

$n = 3$

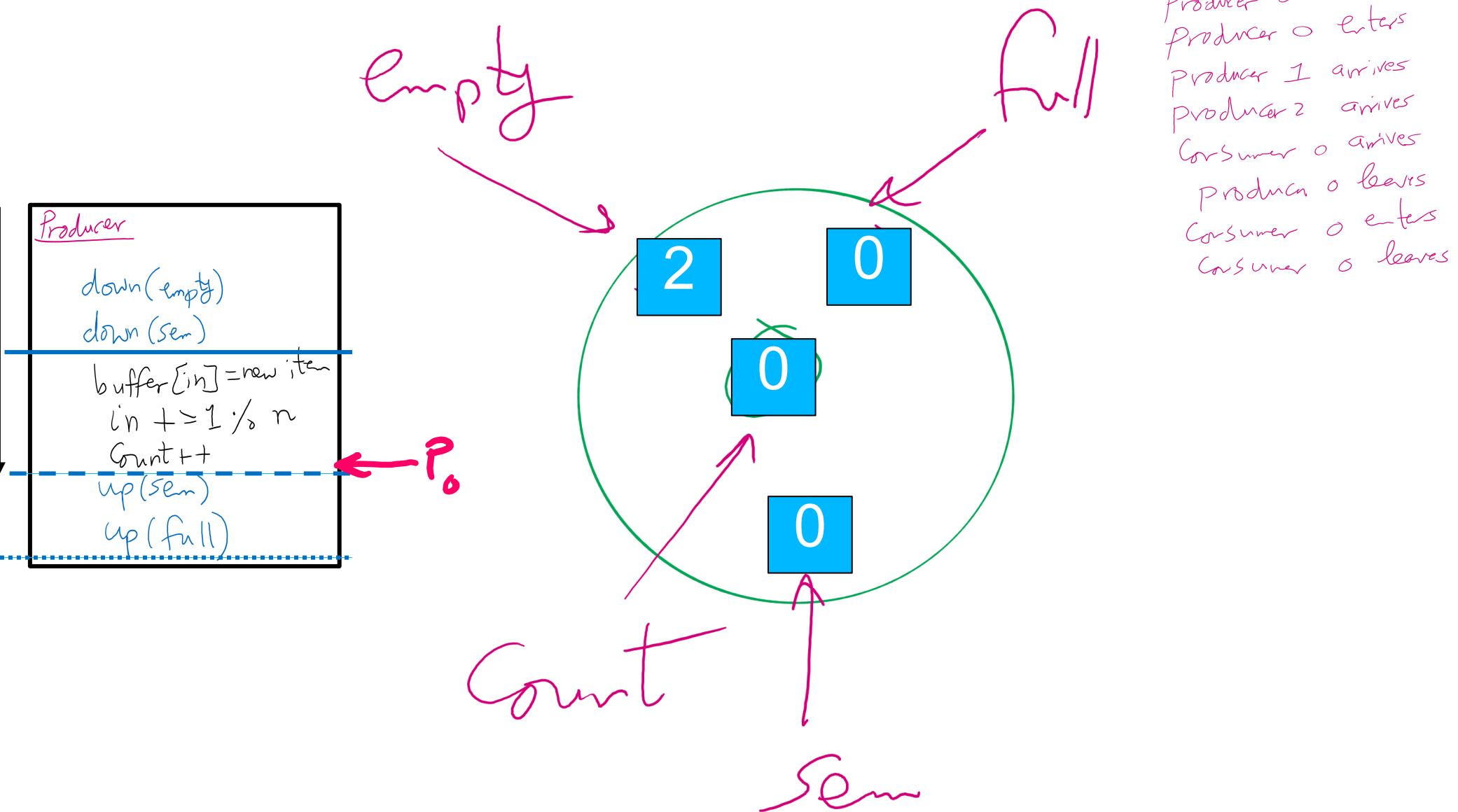
Producer 0 arrives
Producer 0 enters
producer 1 arrives
producer 2 arrives
Consumer 0 arrives
producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

Producer

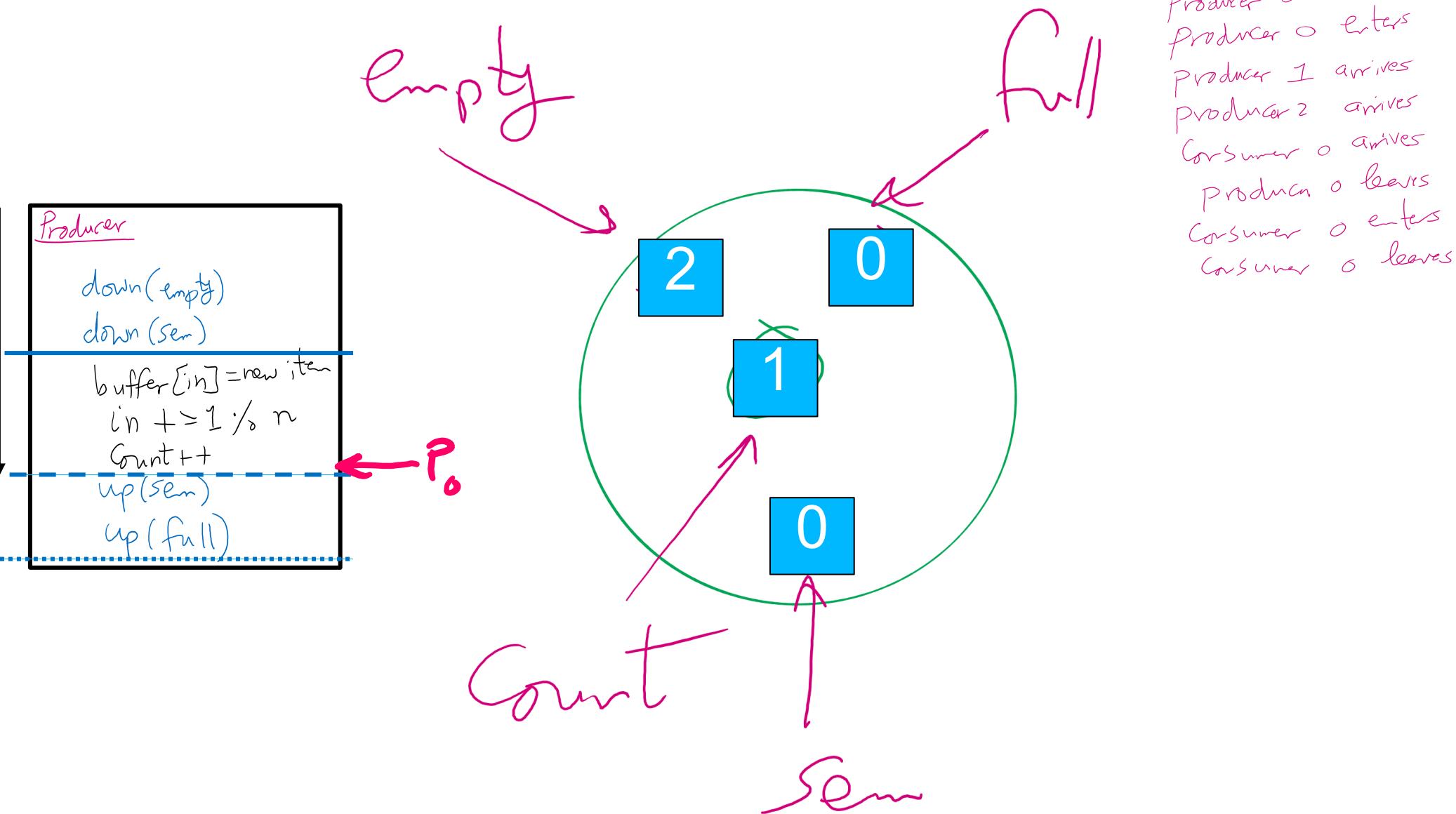
```
down(empty)
down(sem)
buffer[in] = new item
in += 1 /> n
Count ++
up(sem)
up(full)
```



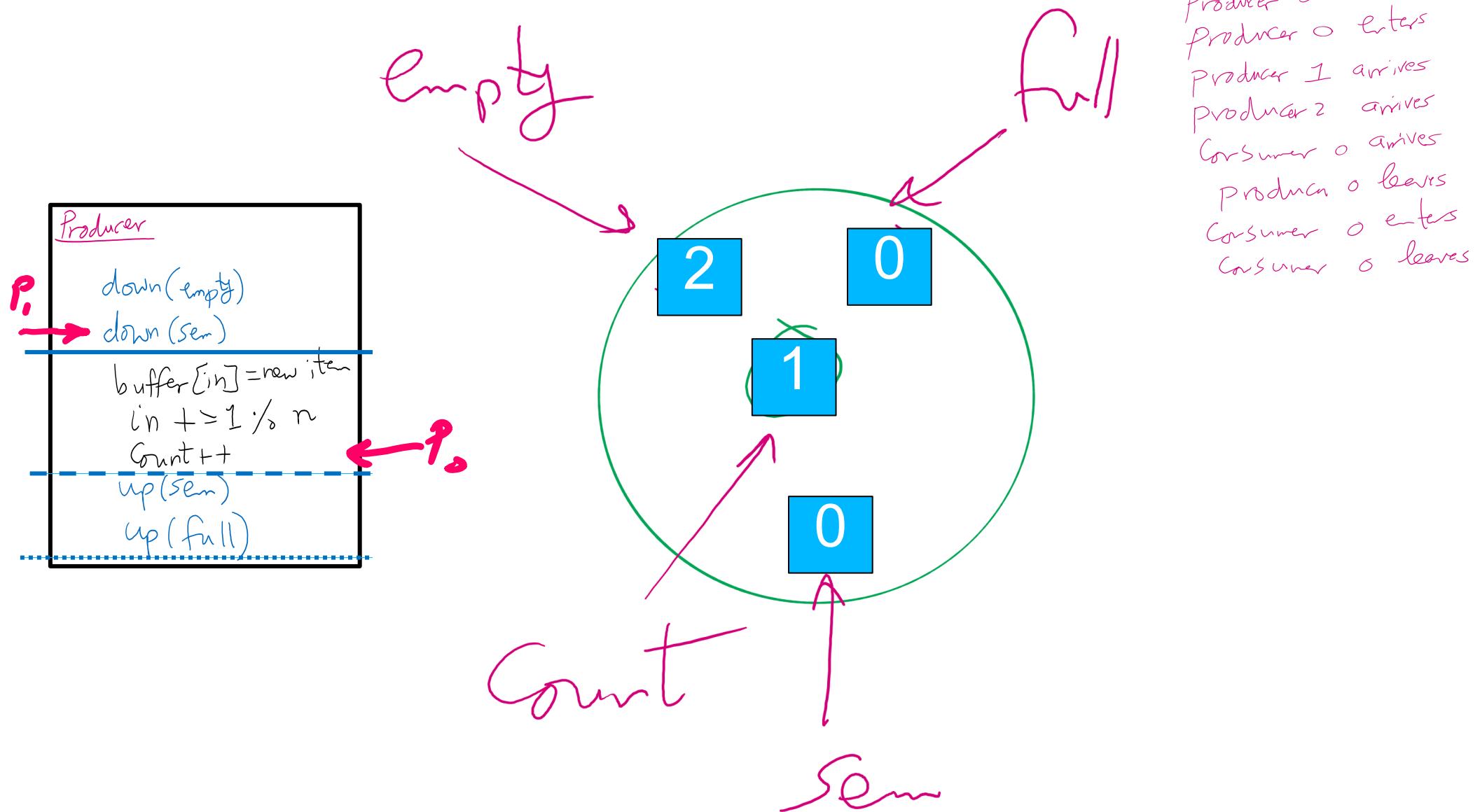
P0 enters



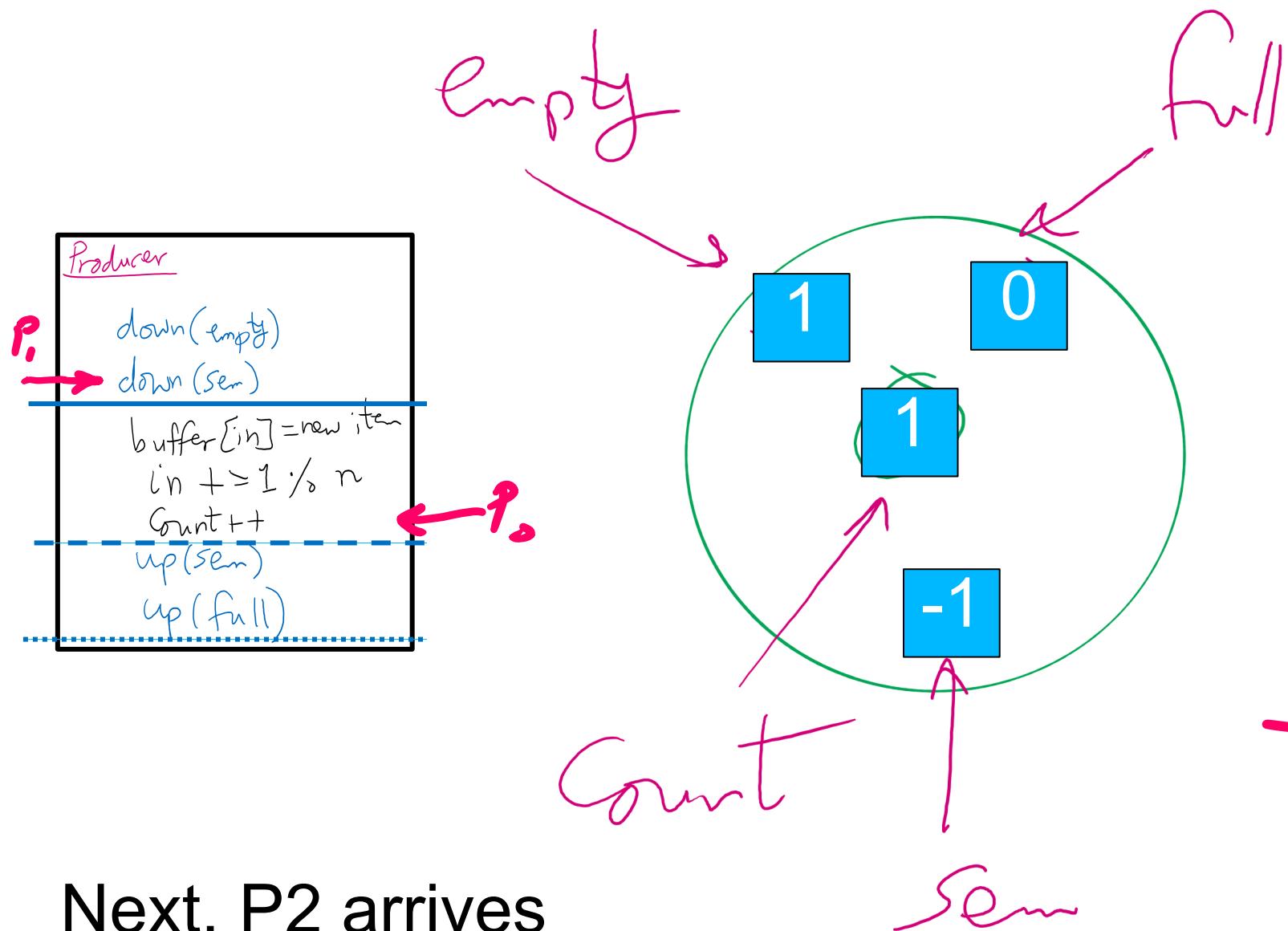
P0 enters



P1 arrives

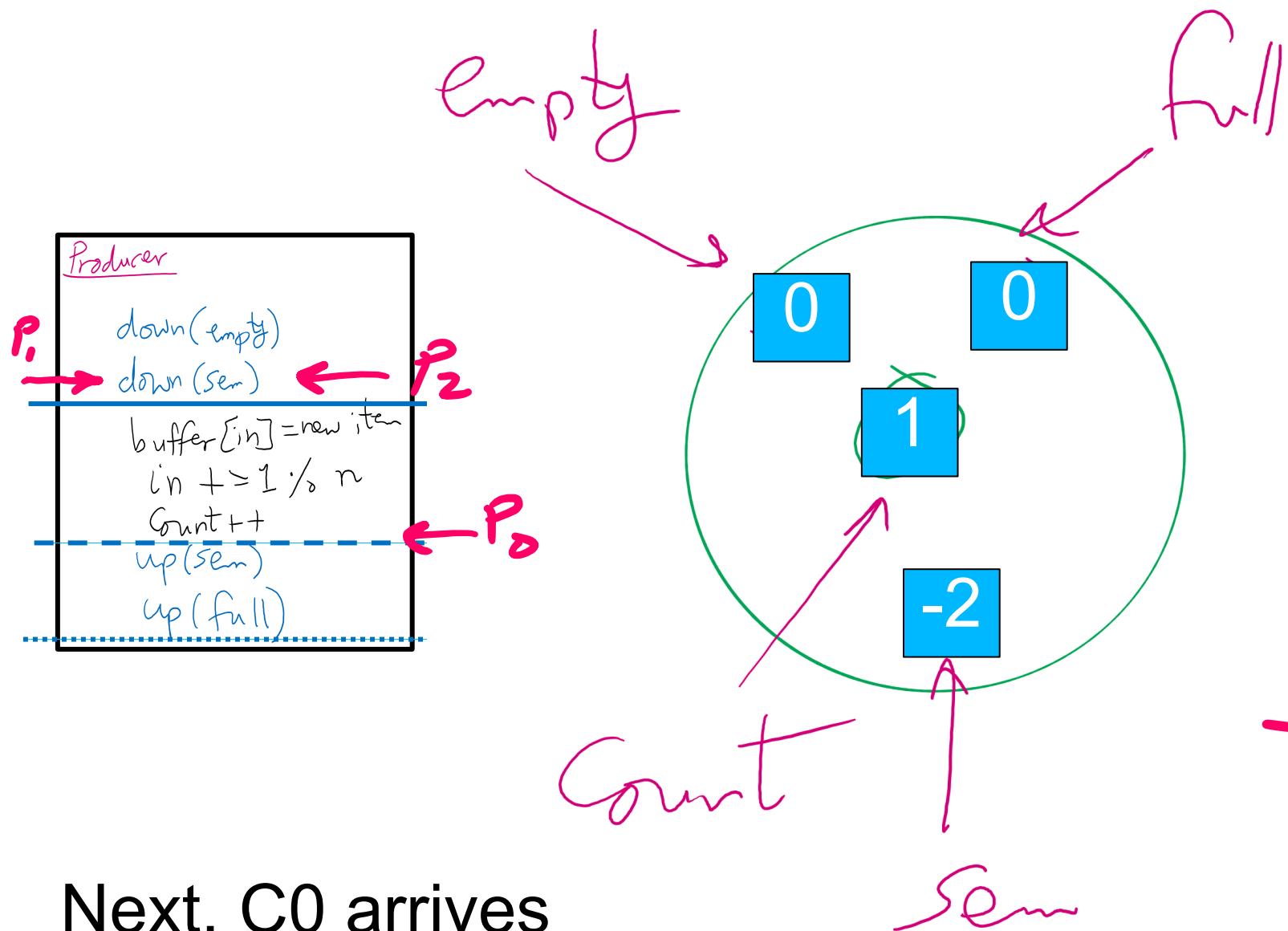


P1 arrives



Next, P2 arrives

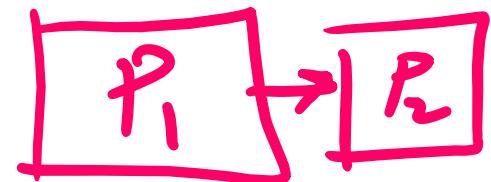
P2 arrives



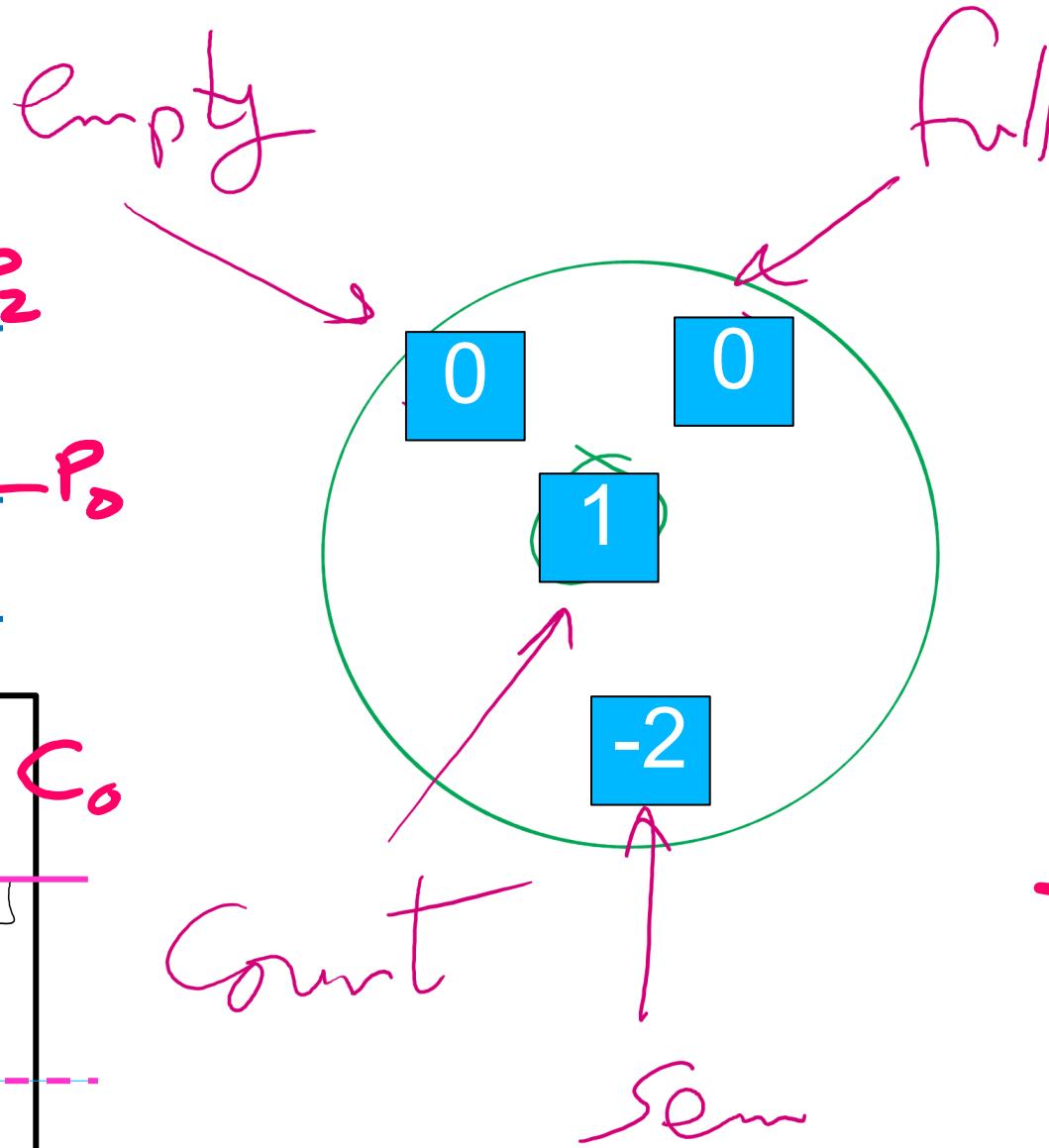
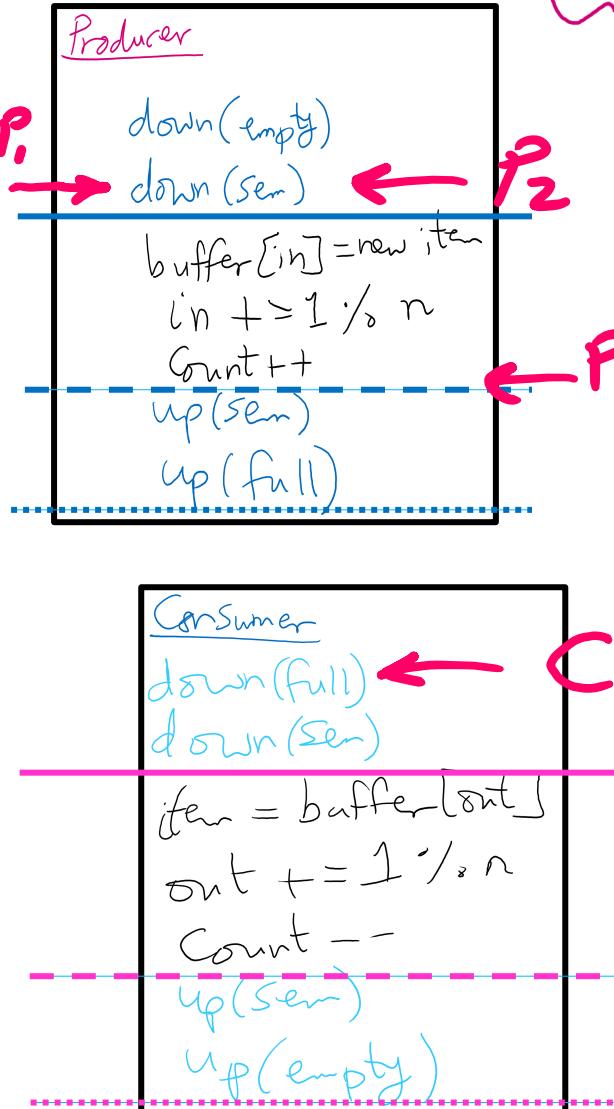
$$n = 3$$

Producer 0 arrives
Producer 0 enters
producer 1 arrives
producer 2 arrives
Consumer 0 arrives
producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

Sem Queue

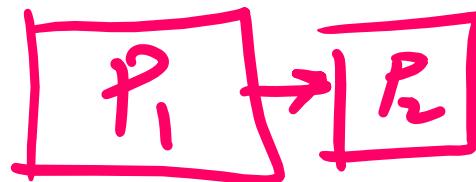


C0 arrives

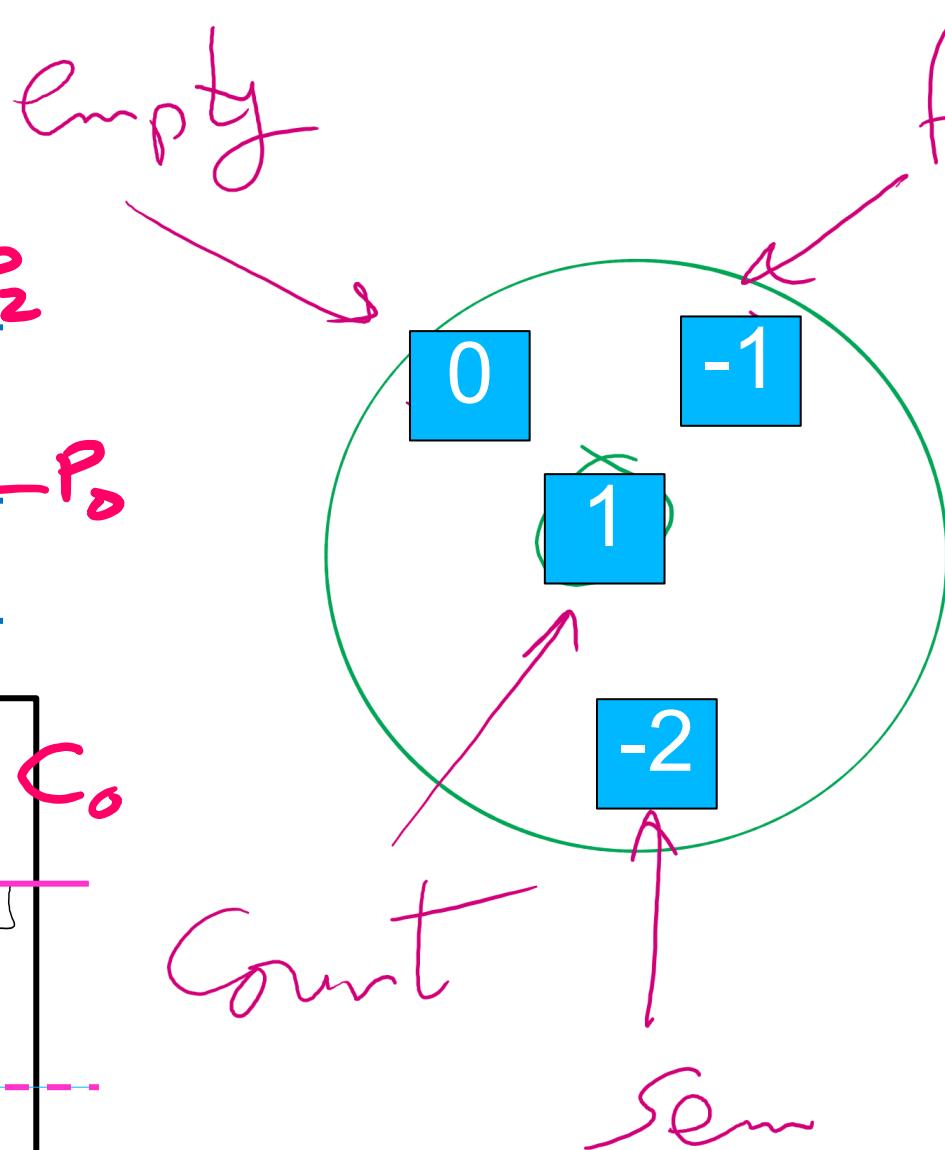
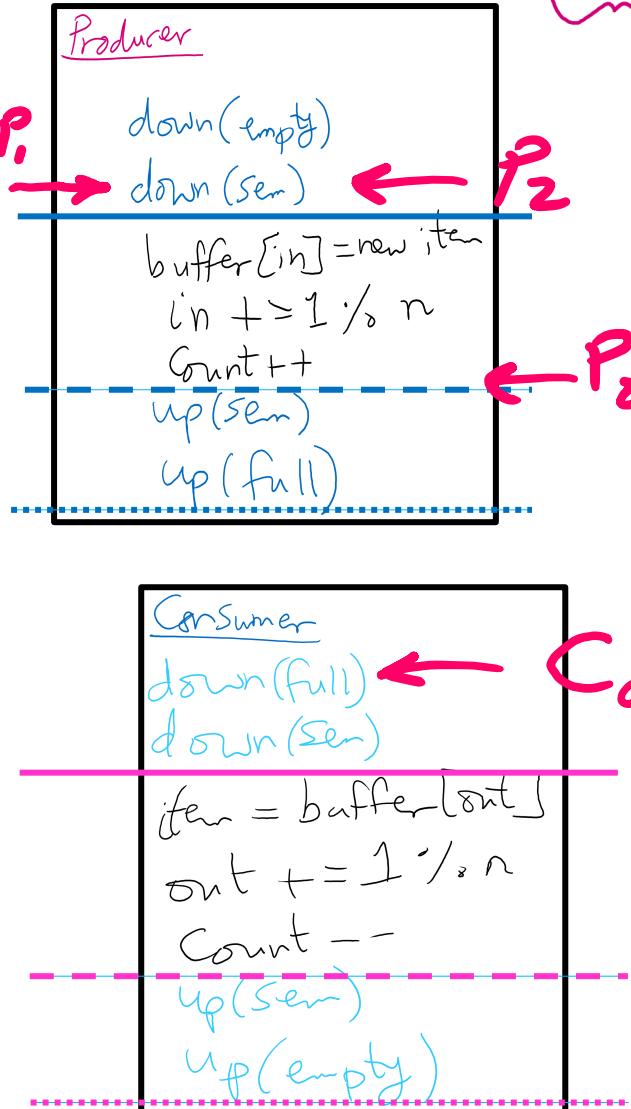


$n = 3$
 Producer 0 arrives
 Producer 0 enters
 Producer 1 arrives
 Producer 2 arrives
 Consumer 0 arrives
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

Sem Queue



C0 arrives



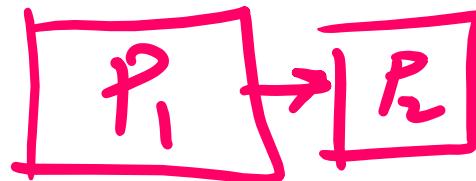
n = 3

Producer 0 arrives
 Producer 0 enters
 Producer 1 arrives
 Producer 2 arrives
 Consumer 0 arrives
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

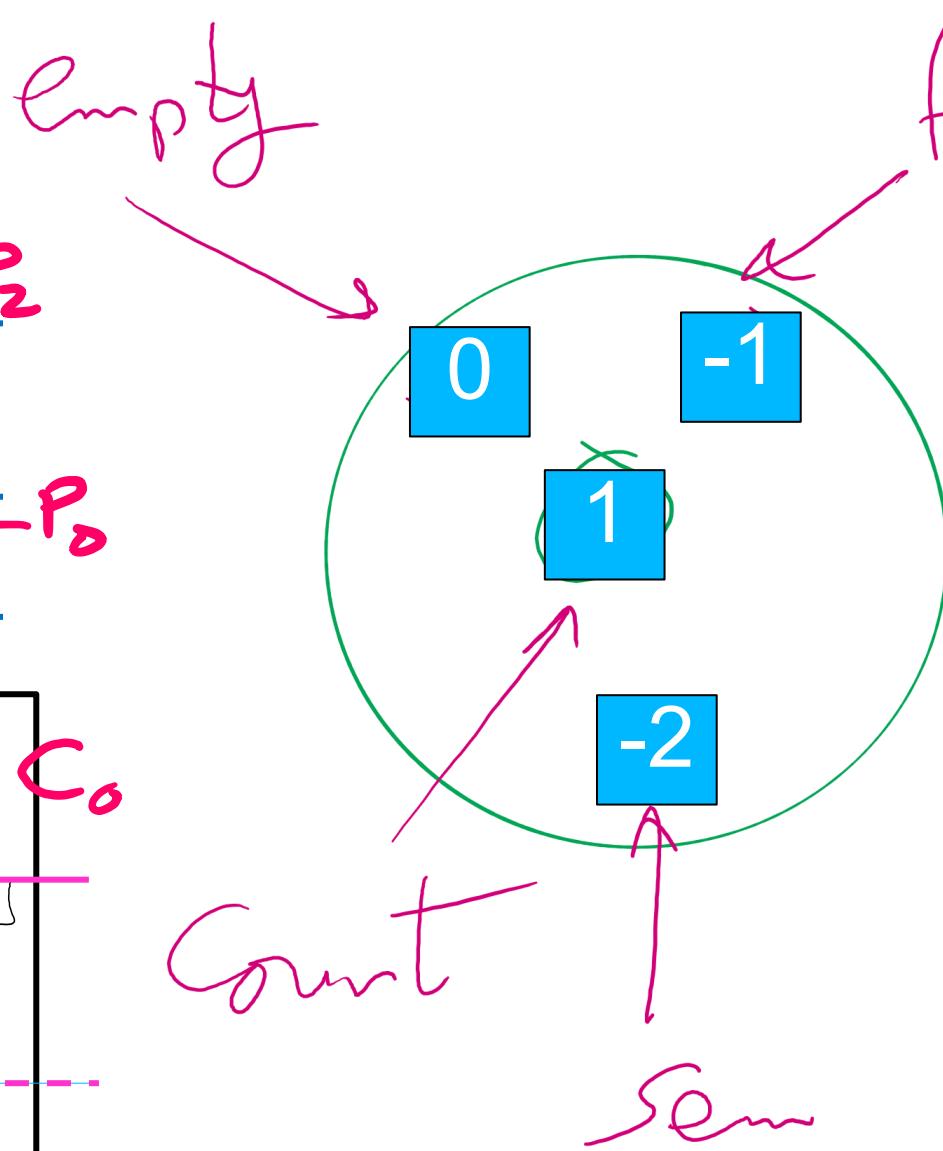
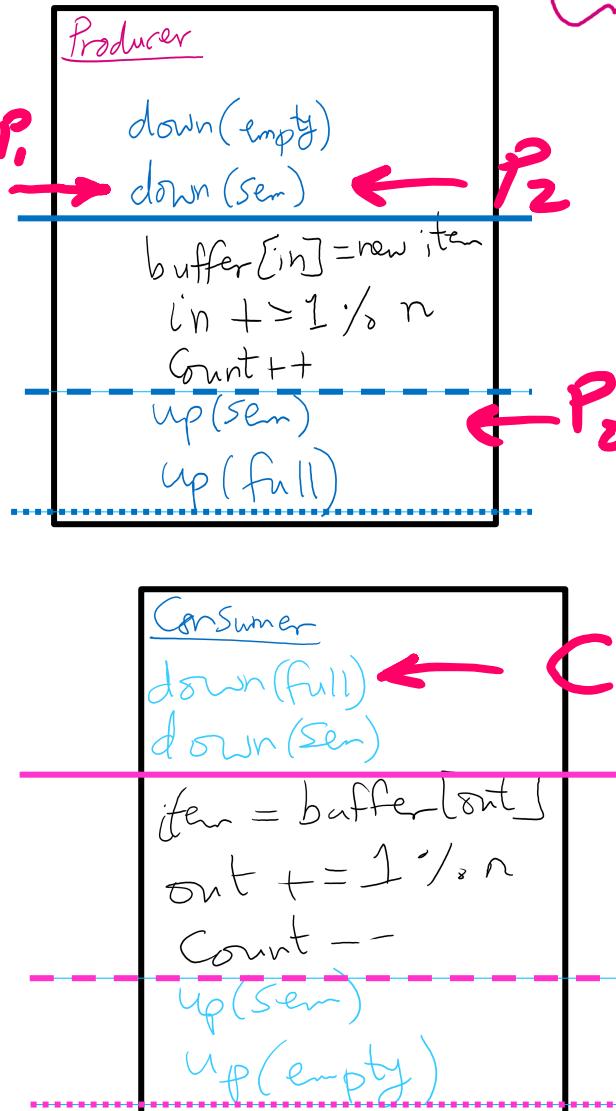
full Queue

| C0 |

Sem Queue



P0 leaves



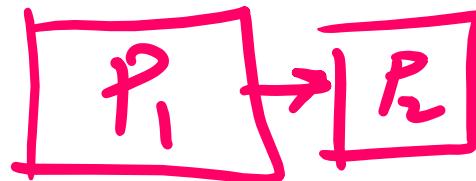
$$n = 3$$

Producer 0 arrives
 Producer 0 enters
 Producer 1 arrives
 Producer 2 arrives
 Consumer 0 arrives
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

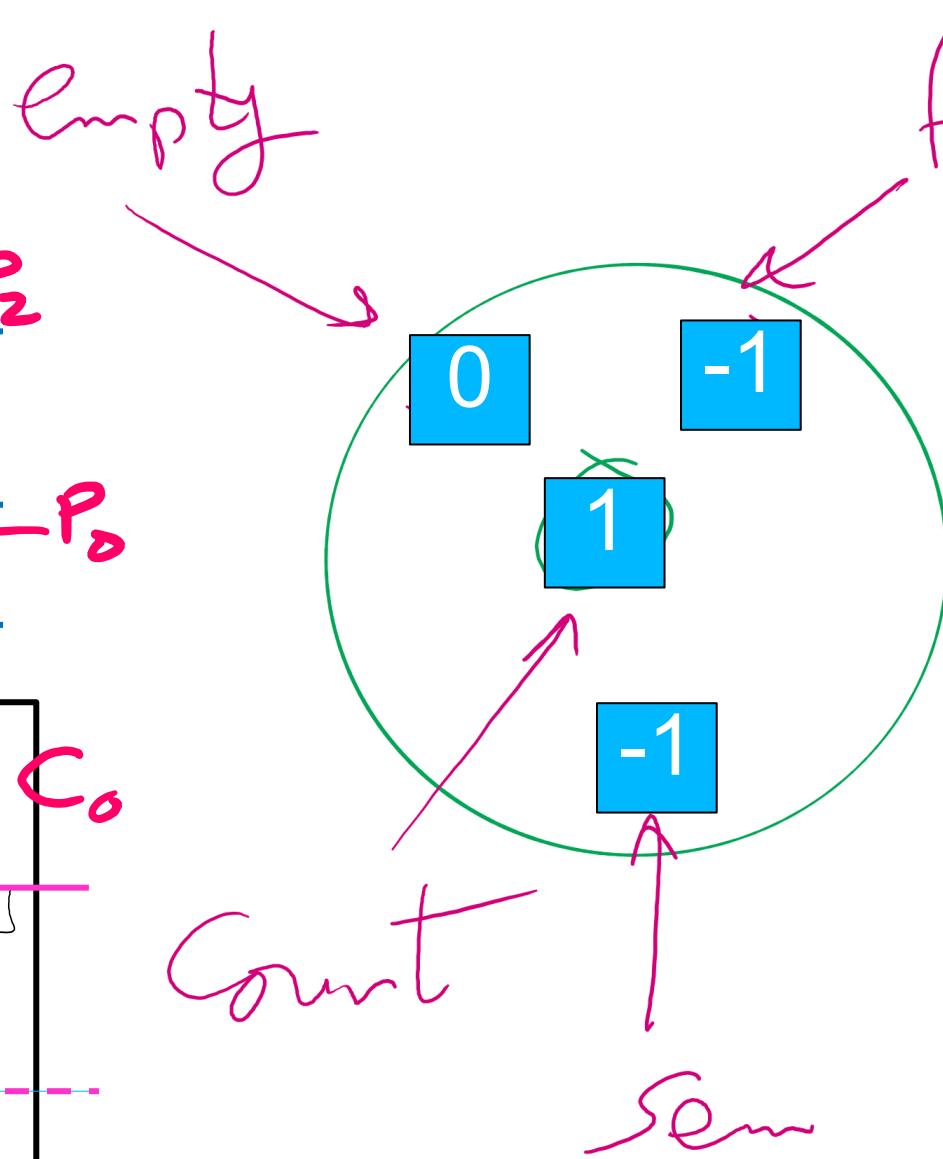
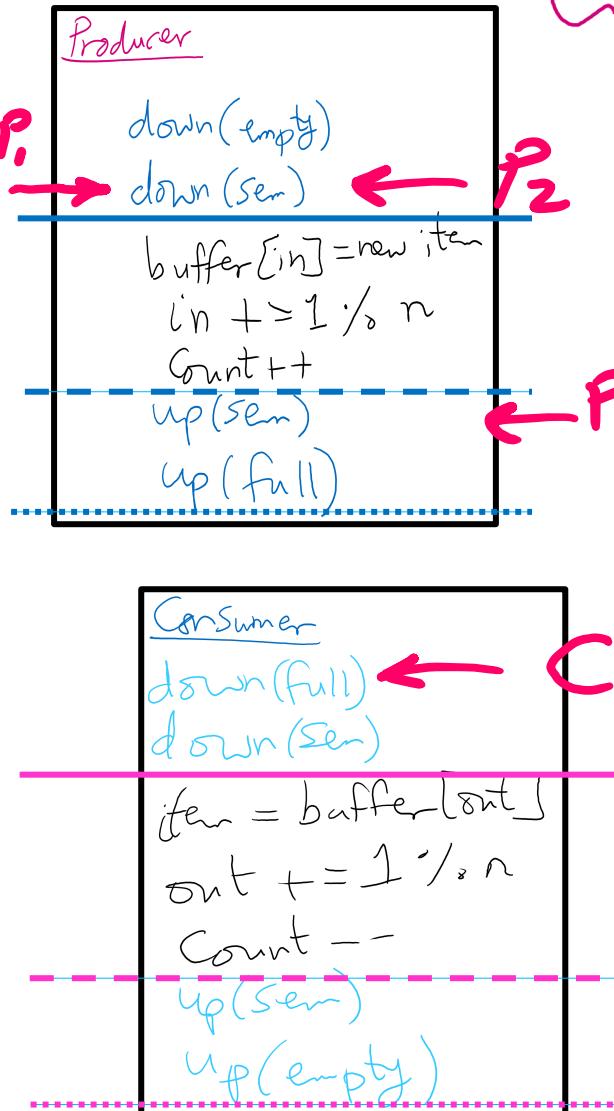
full Queue



Sem Queue



P0 leaves



$n = 3$

Producer 0 arrives

Producer 0 enters

producer 1 arrives

producer 2 arrives

Consumer 0 arrives

producer 0 leaves

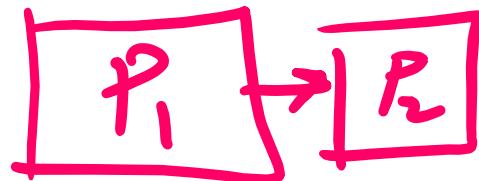
Consumer 0 enters

Consumer 0 leaves

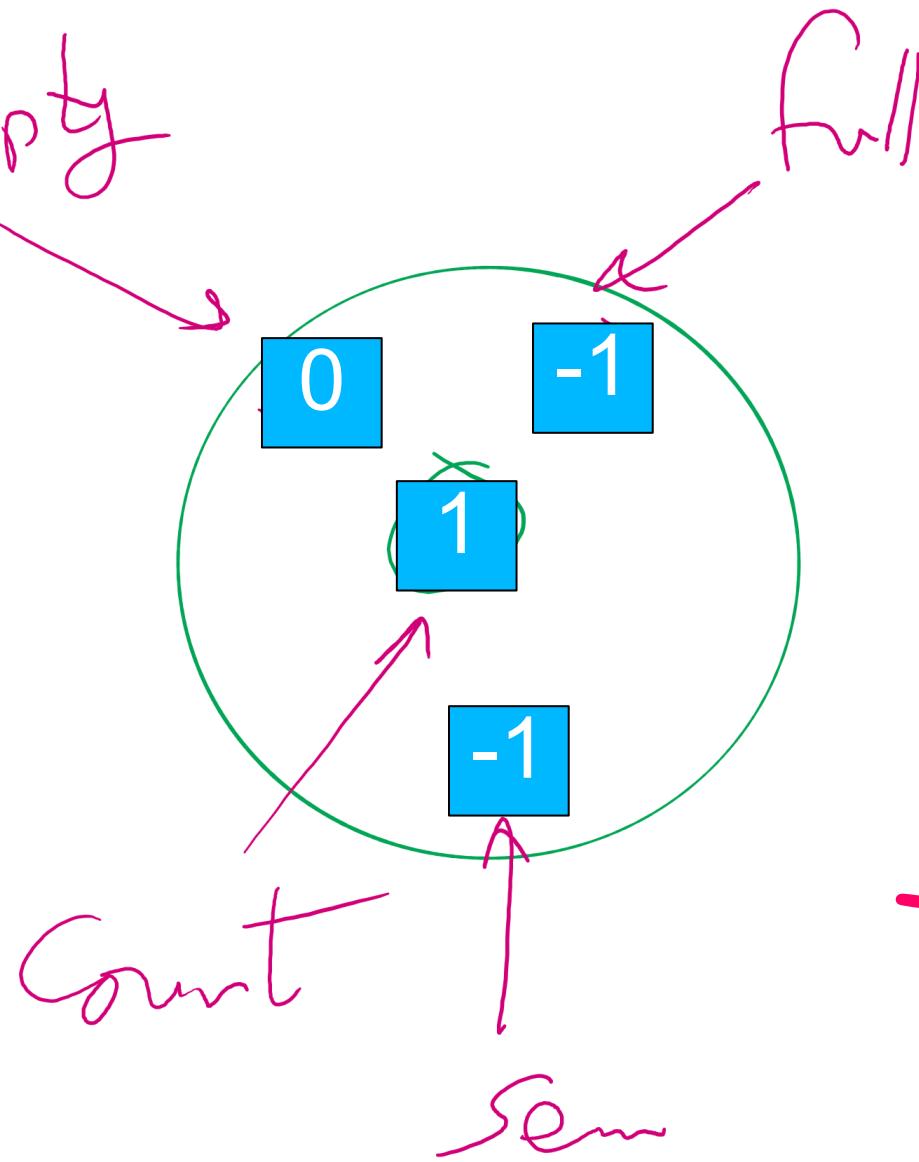
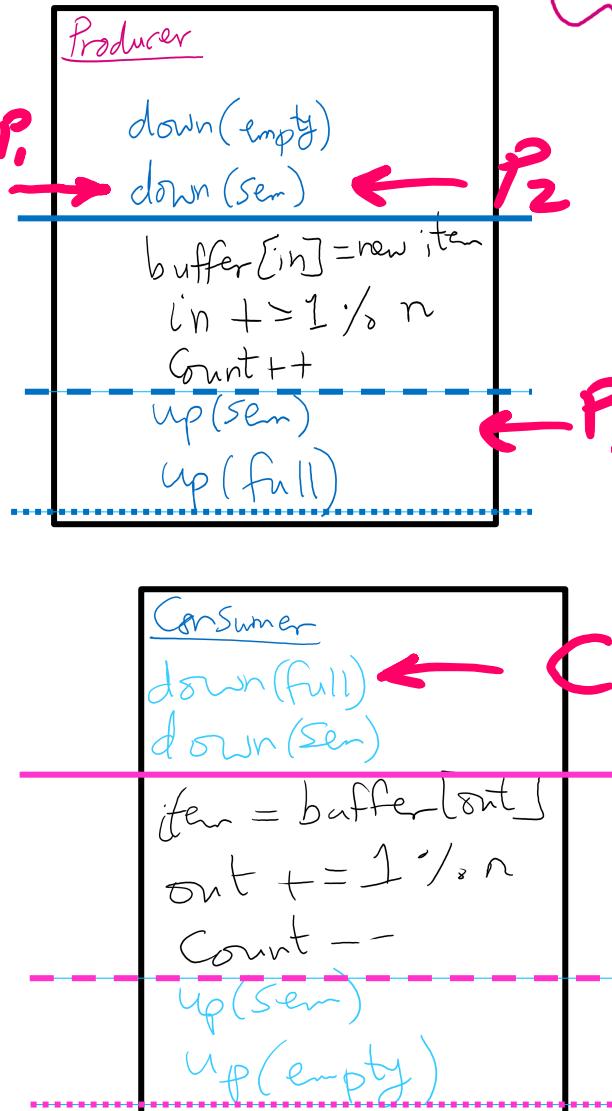
full Queue



Sem Queue



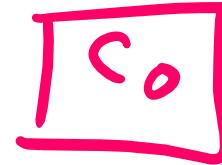
P0 leaves



n = 3

Producer 0 arrives
 Producer 0 enters
 Producer 1 arrives
 Producer 2 arrives
 Consumer 0 arrives
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

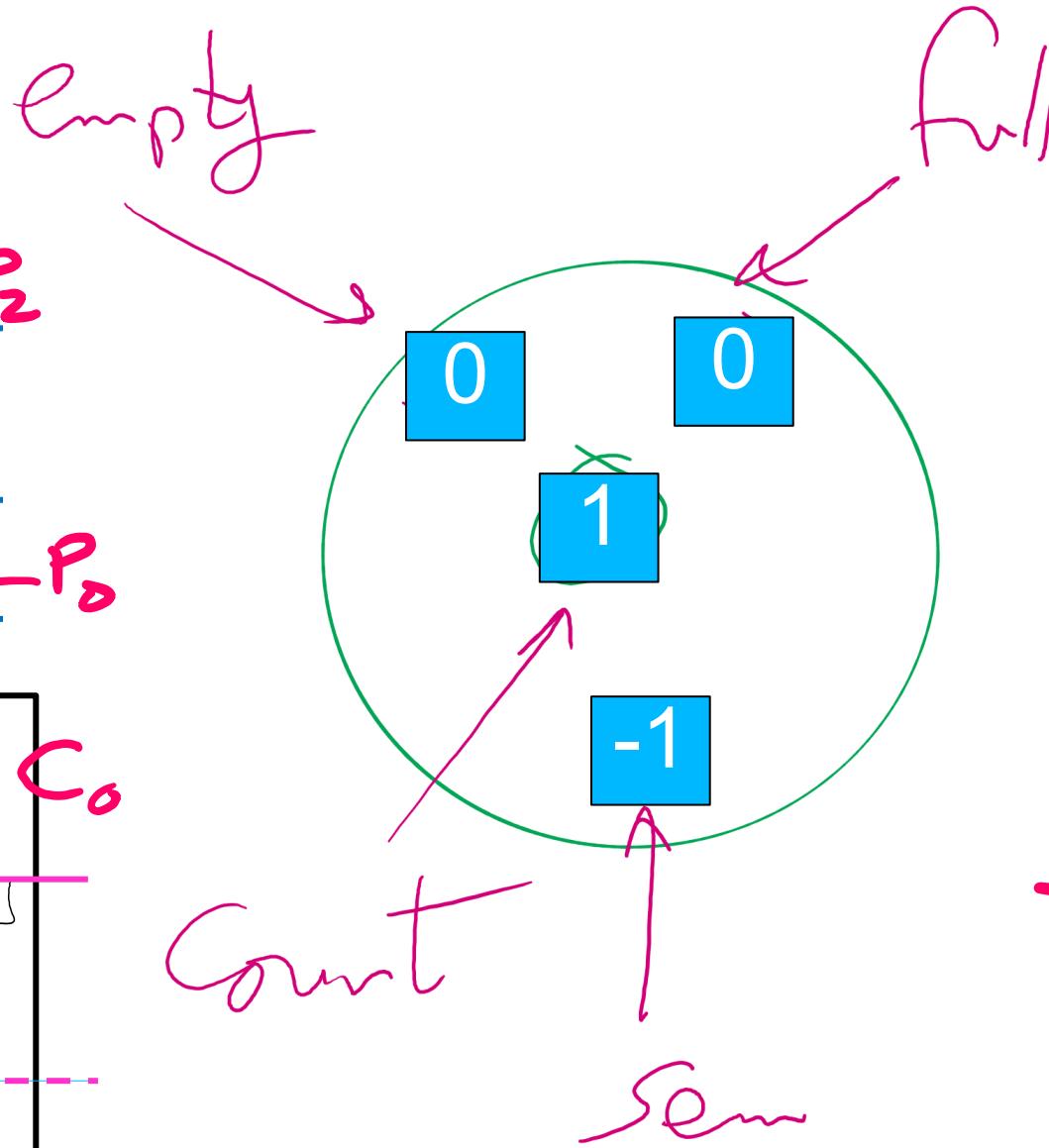
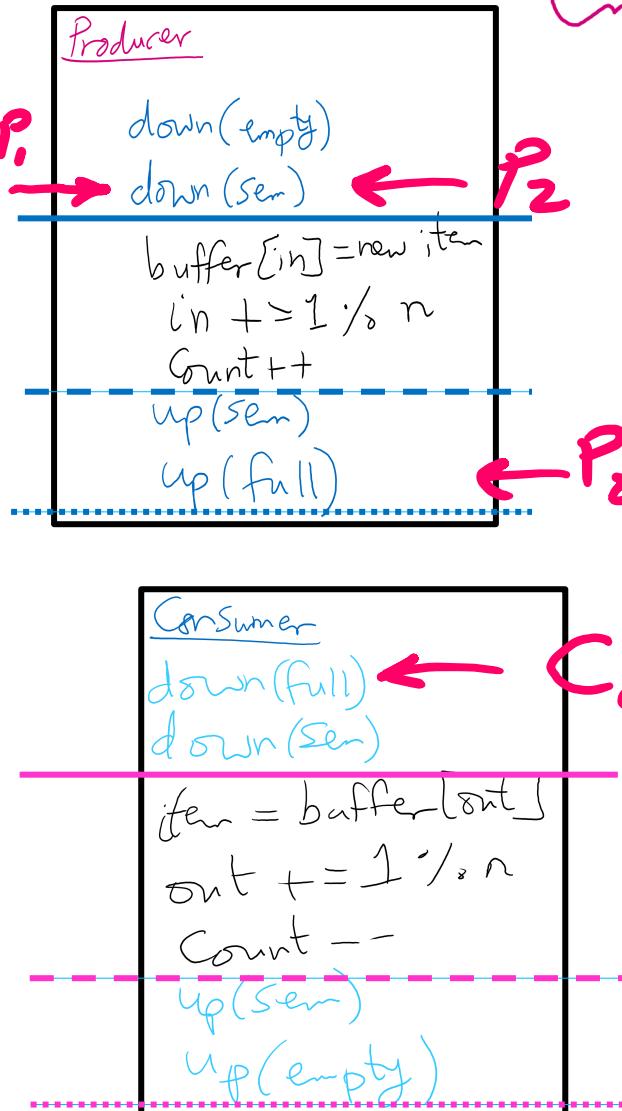
full Queue



Sem Queue



P0 leaves



$n = 3$
 Producer 0 arrives
 Producer 0 enters
 Producer 1 arrives
 Producer 2 arrives
 Consumer 0 arrives
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

full Queue



Sem Queue



C0 enters

Producer

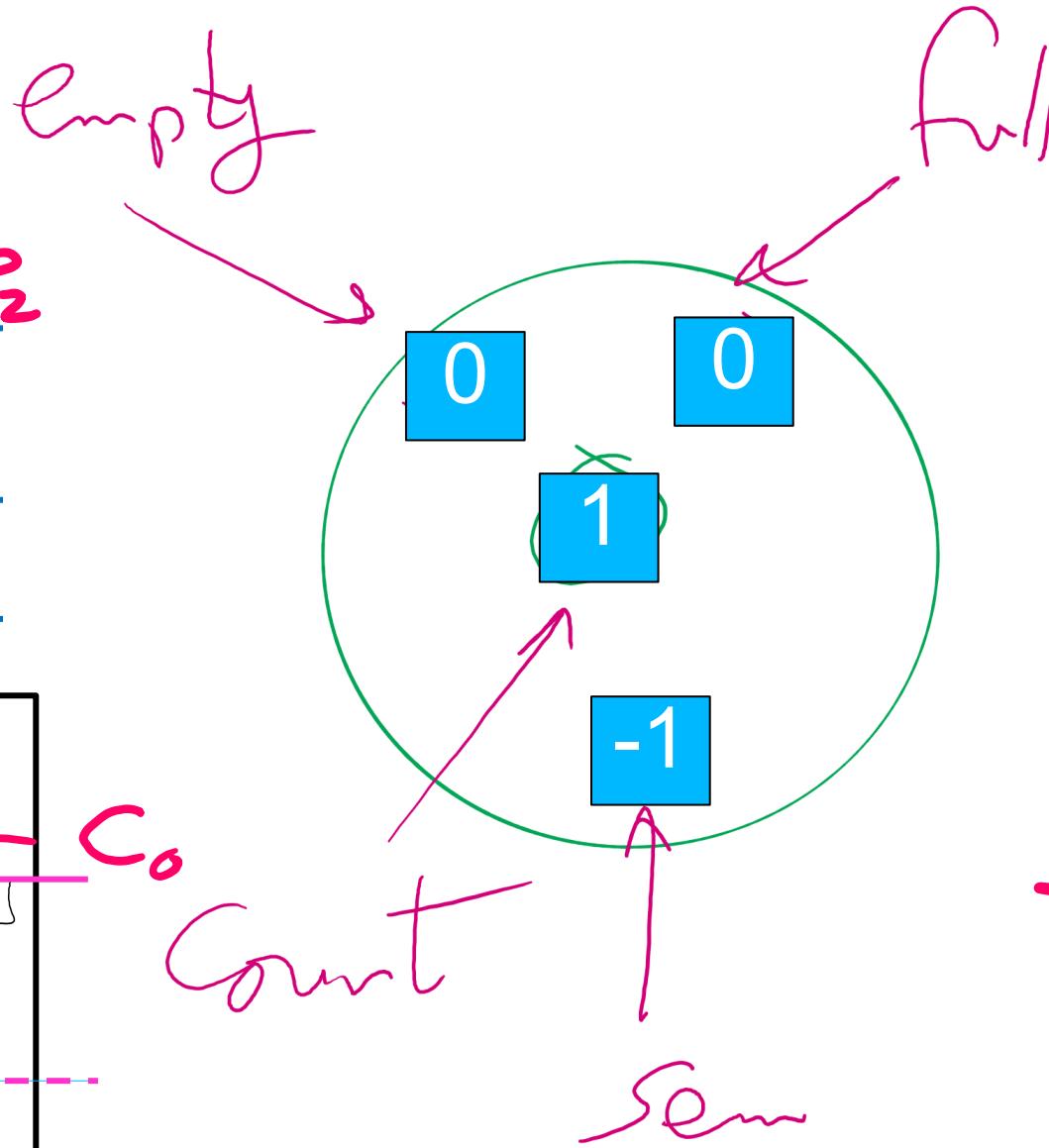
```

P1 →
    down(empty)
    down(sem)
    buffer[in] = new item
    in += 1/n
    Count ++
    up(sem)
    up(full)
  
```

Consumer

```

C0 ←
    down(full)
    down(sem)
    if item = buffer[out]
    out += 1/n
    Count --
    up(sem)
    up(empty)
  
```



$n = 3$
 Producer 0 arrives
 Producer 0 enters
 Producer 1 arrives
 Producer 2 arrives
 Consumer 0 arrives
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

full Queue

Sem Queue



C0 enters

Producer

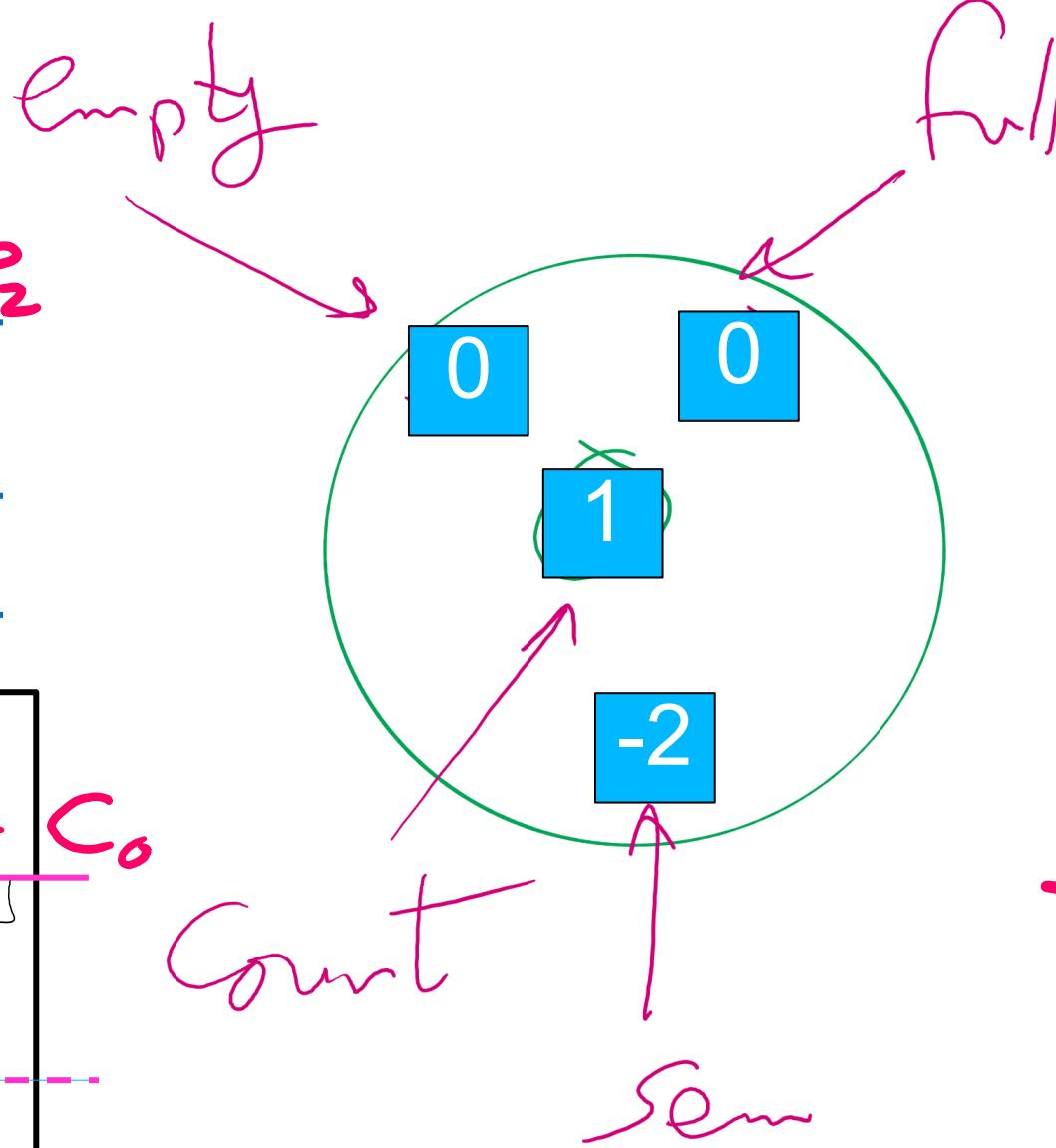
```

P1 →
    down(empty)
    down(sem)
    buffer[in] = new item
    in += 1 / n
    Count ++
    up(sem)
    up(full)
  
```

Consumer

```

C0 ←
    down(full)
    down(sem)
    if item = buffer[out]
    out += 1 / n
    Count --
    up(sem)
    up(empty)
  
```



n = 3

Producer 0 arrives
 Producer 0 enters
 Producer 1 arrives
 Producer 2 arrives
 Consumer 0 arrives
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

full Queue

Sem Queue



C0 enters

Producer

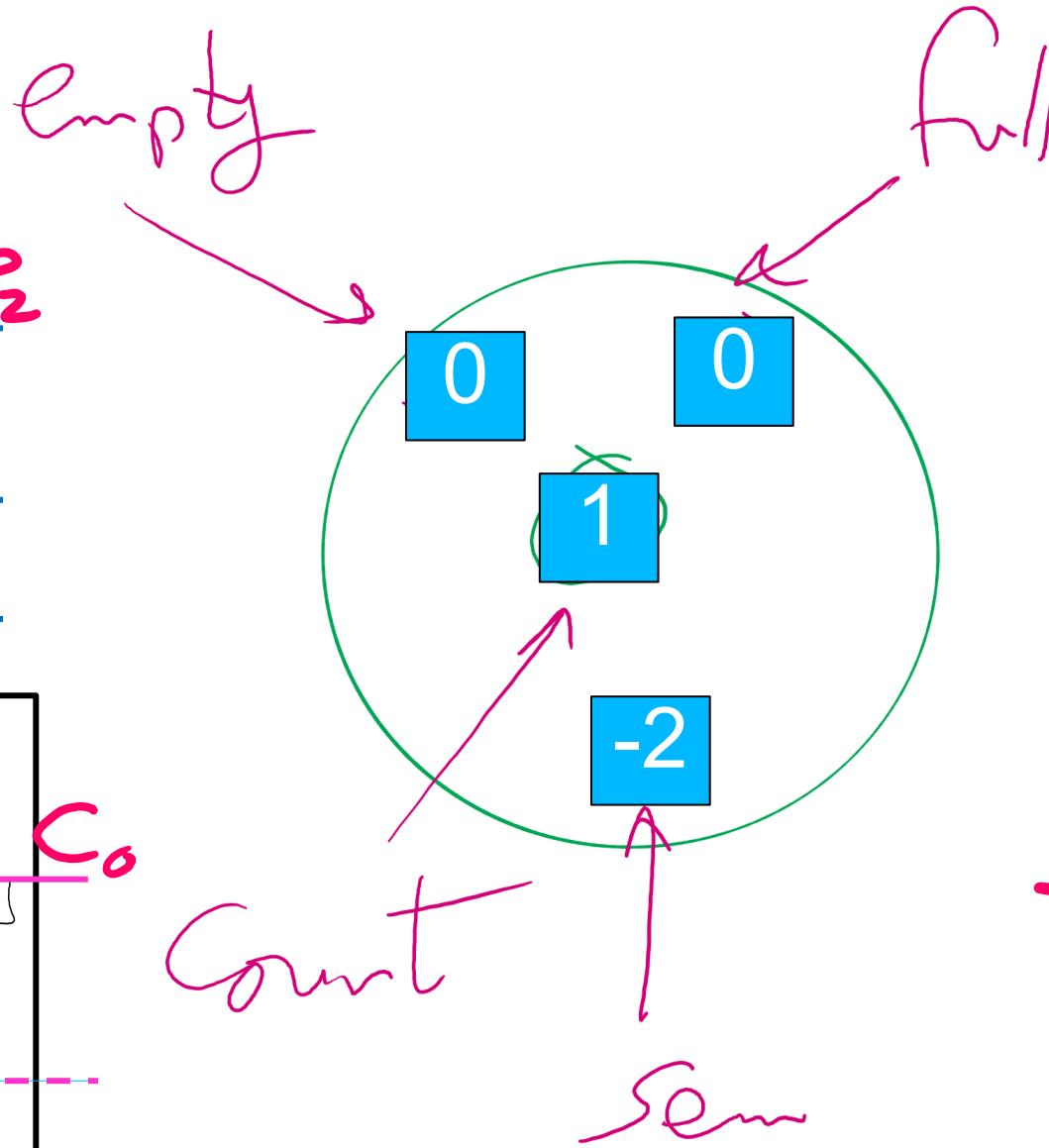
```

P1 → down(empty)
down(sem) ← P2
buffer[in] = new item
in += 1/n
Count ++
up(sem)
up(full)
    
```

Consumer

```

down(full)
down(sem) ← C0
item = buffer[out]
out += 1/n
Count --
up(sem)
up(empty)
    
```



n = 3

Producer 0 arrives

Producer 0 enters

producer 1 arrives

producer 2 arrives

Consumer 0 arrives

producer 0 leaves

Consumer 0 enters

Consumer 0 leaves

full Queue

Sem Queue



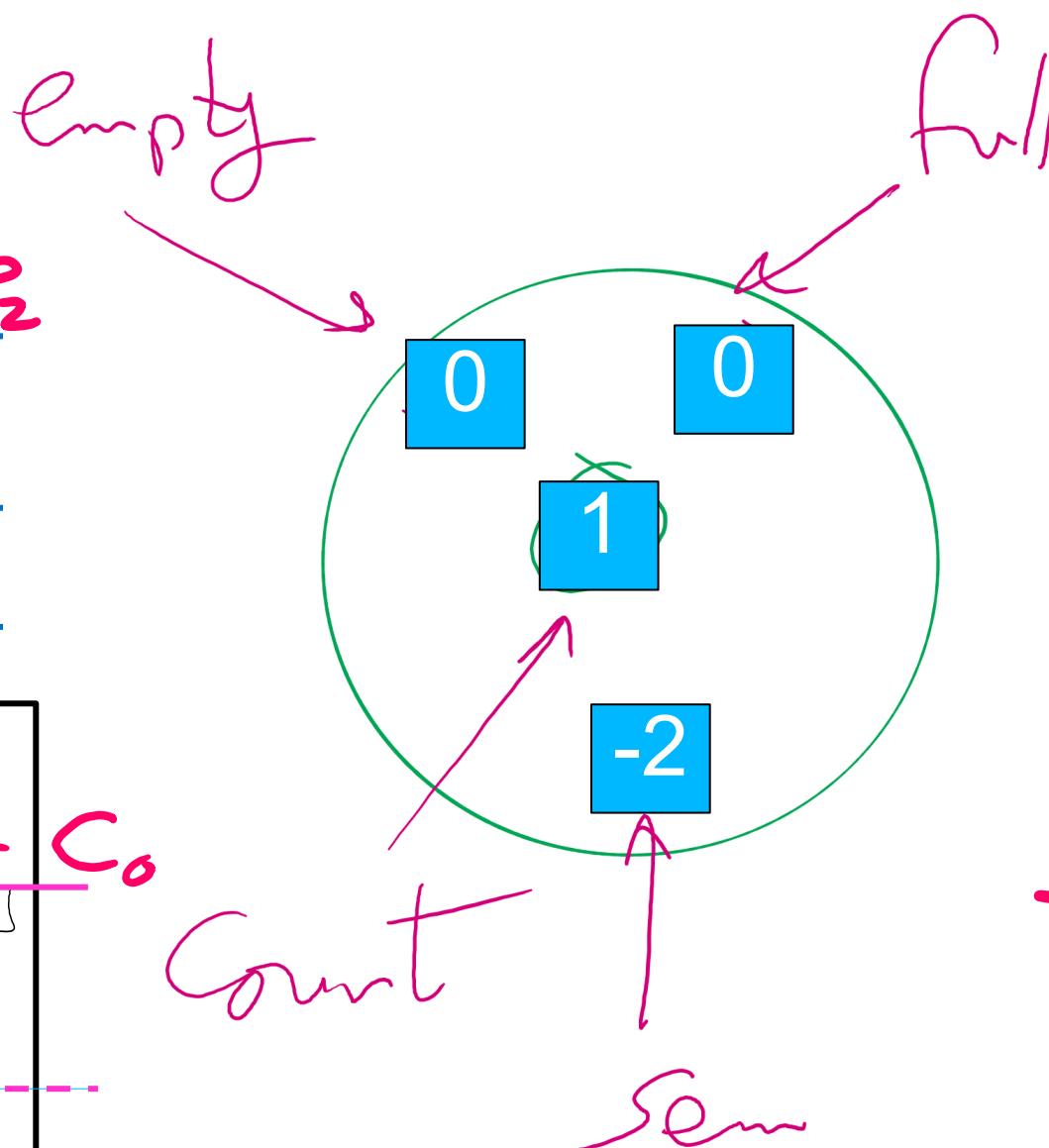
Can C0 enter?

Producer

```
P1 → down(empty)
down(sem) ← P2
buffer[in] = new item
in += 1/n
Count ++
up(sem)
up(full)
```

Consumer

```
down(full)
down(sem) ← C0
item = buffer[out]
out += 1/n
Count --
up(sem)
up(empty)
```



n = 3

Producer 0 arrives
Producer 0 enters
Producer 1 arrives
producer 2 arrives
Consumer 0 arrives
producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

full Queue

Sem Queue



Example 2

$$\underline{n = 1}$$

Consumer 0 arrives

Producer 0 arrives

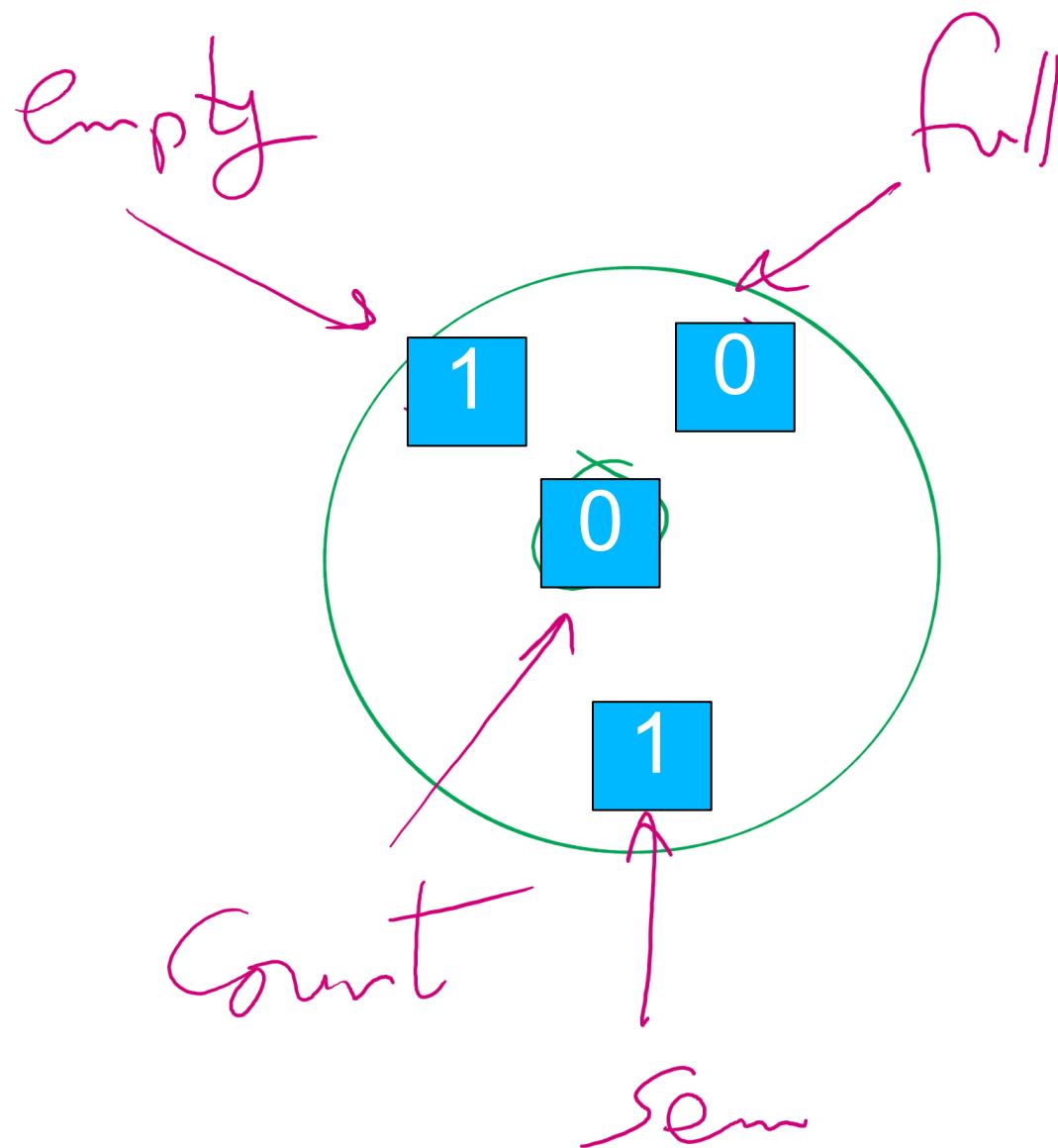
Producer 0 enters

Producer 0 leaves

Consumer 0 enters

Consumer 0 leaves

Initial state



$$n = 1$$

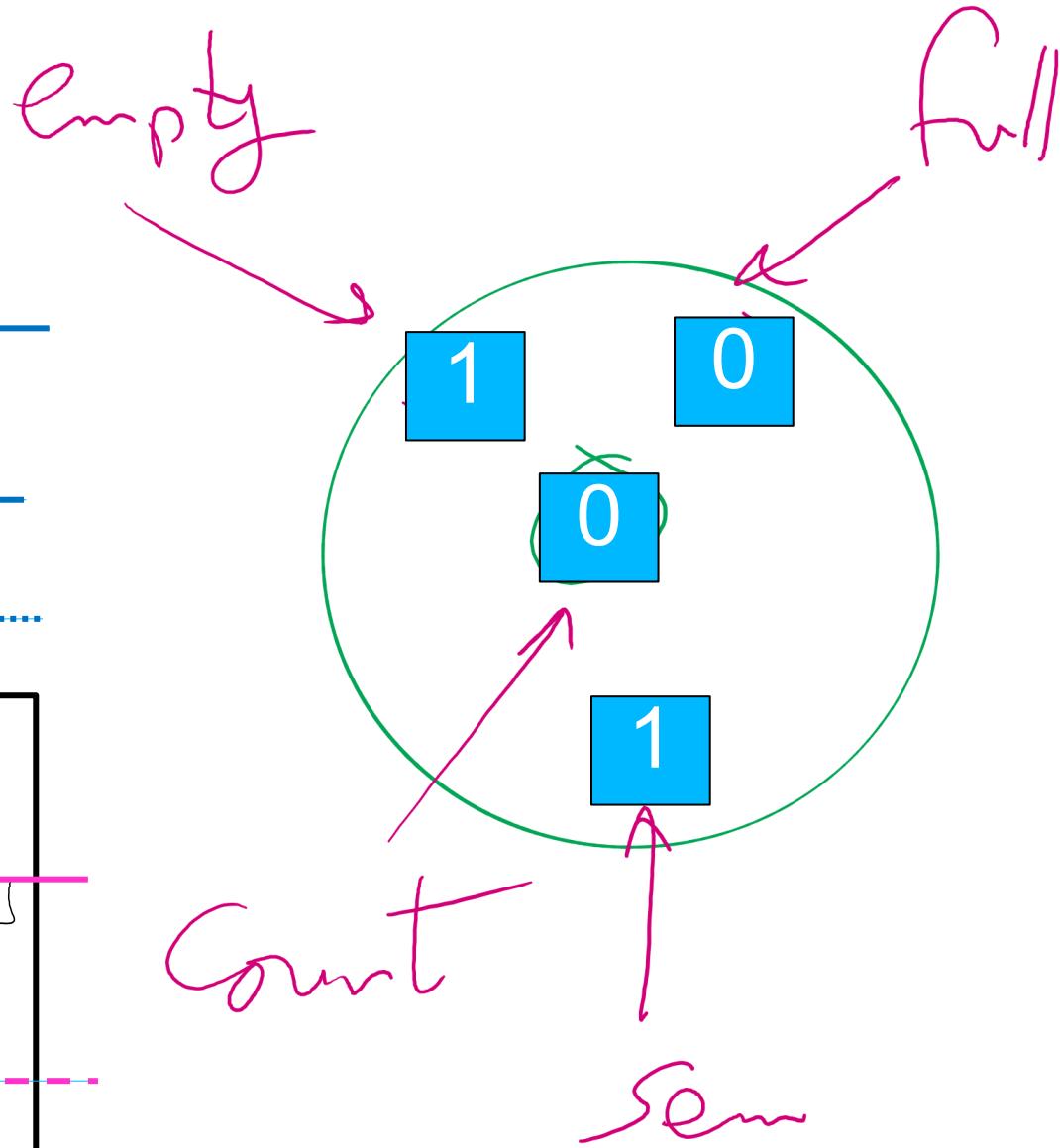
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

C0 arrives

$$n = 1$$

Producer

```
down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)
```



Consumer

```
down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)
```

Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

C0 arrives

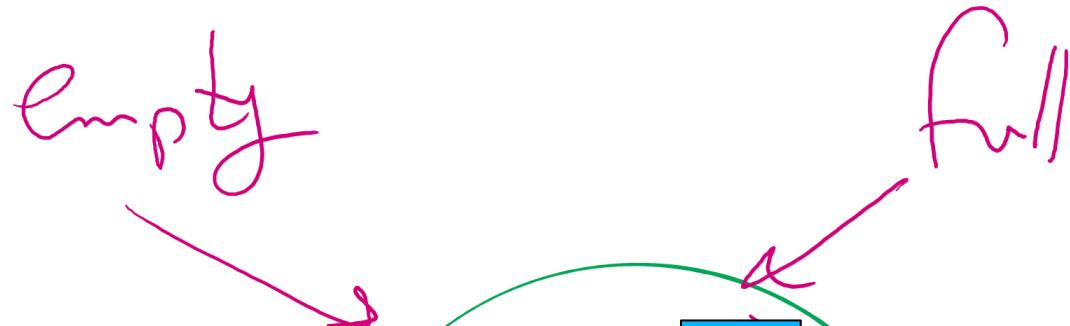
$$n = 1$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
Count ++
up(sem)
up(full)

```



Consumer 0 arrives
 Producer 0 arrives
 Producer 0 enters
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1 % n
Count --
up(sem)
up(empty)

```

Count
 Sem

Full Queue

C0

P0 arrives

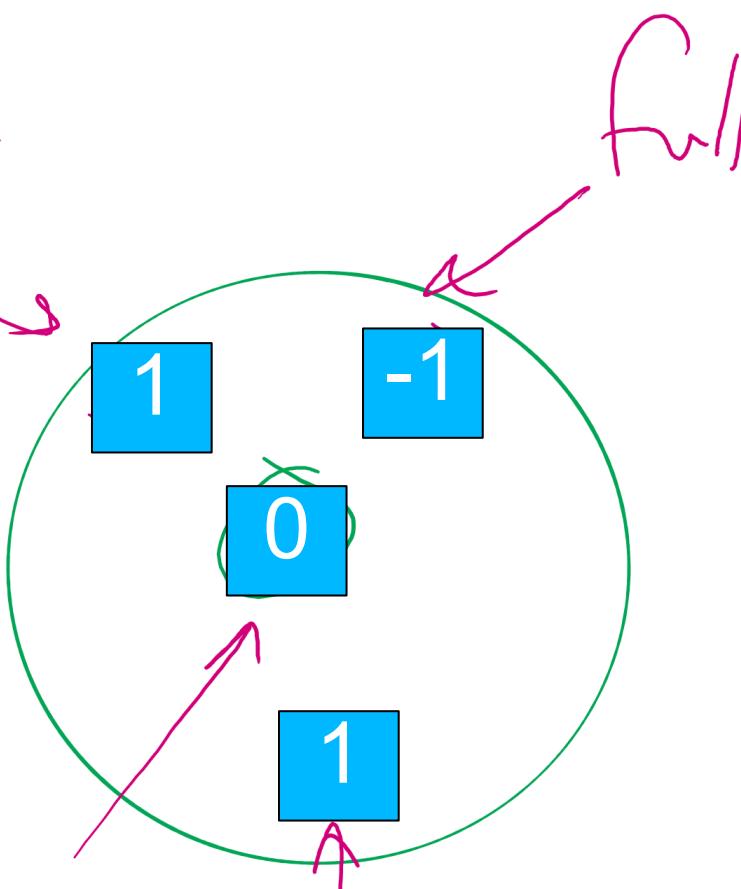
$$n = 1$$

Producer

```
down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)
```

empty

P0



Consumer

```
down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)
```

C0

Count

Sem

Full Queue

C0

Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

P0 arrives

$$n = 1$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

```

empty

P0

full



Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

```

Count

C0

Sem

Full Queue



Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

P0 arrives

$$n = 1$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

```

empty

P0

full



Consumer 0 arrives
 Producer 0 arrives
 Producer 0 enters
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

```

Count

C0

Full Queue



P0 enters

$$n = 1$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

```

empty

P0

full



Consumer 0 arrives
 Producer 0 arrives
 Producer 0 enters
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

```

Count

C0

Sem

Full Queue



P0 enters

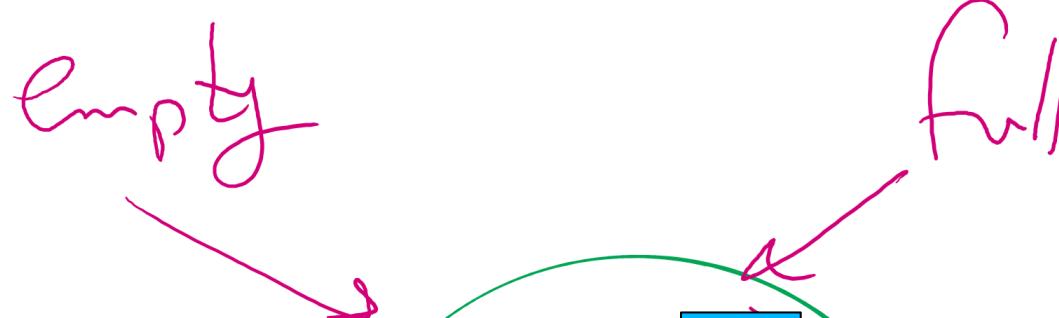
$$n = 1$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

```



Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

```



Consumer 0 arrives
 Producer 0 arrives
 Producer 0 enters
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

Full Queue



P0 leaves

$$n = 1$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

```

empty

P_0



full

Consumer 0 arrives
 Producer 0 arrives
 Producer 0 enters
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

```

C_0

Count

Sem

Full Queue

C_0

P0 leaves

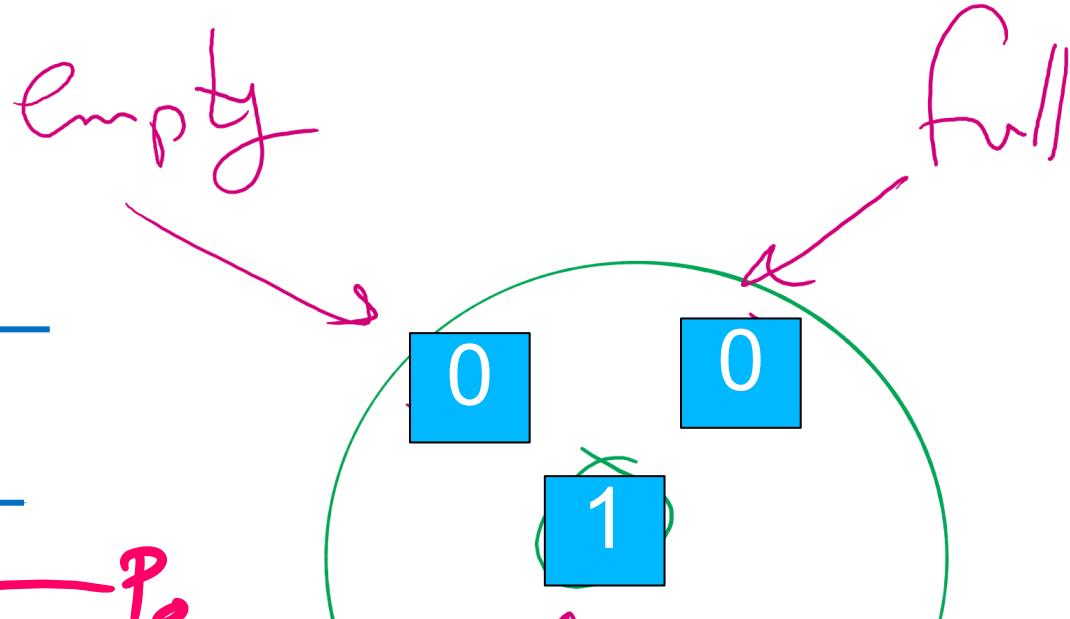
$$n = 1$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1/n
Count ++
up(sem)
up(full)

```



Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1/n
Count --
up(sem)
up(empty)

```

Consumer 0 arrives
 Producer 0 arrives
 Producer 0 enters
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

Full Queue



P0 leaves

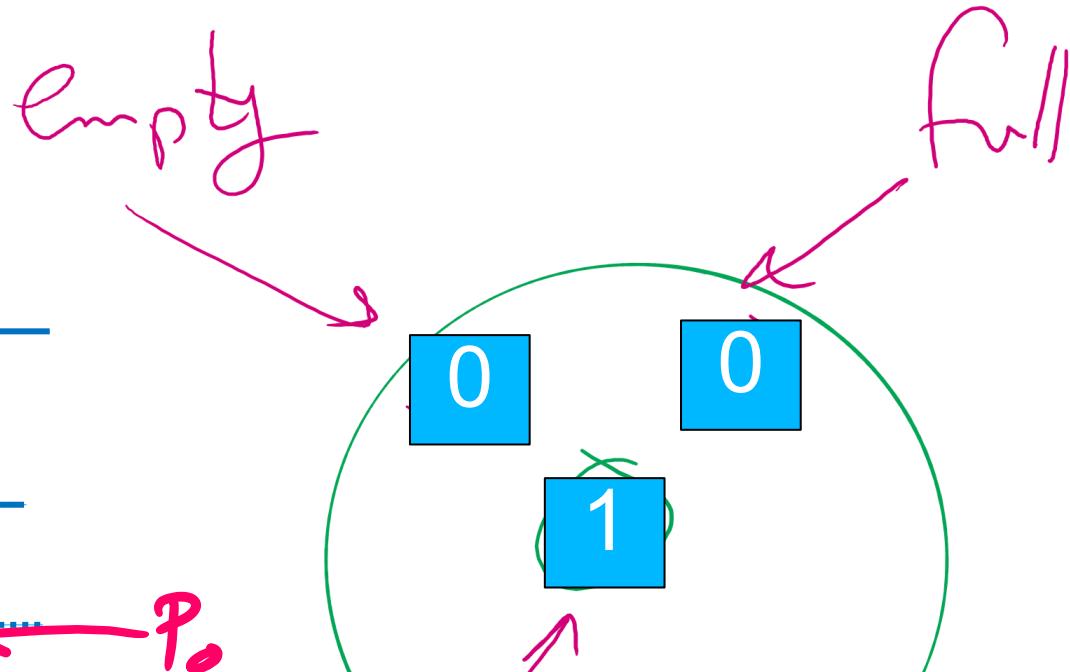
$$n = 1$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1/n
Count ++
up(sem)
up(full)

```



Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1/n
Count --
up(sem)
up(empty)

```

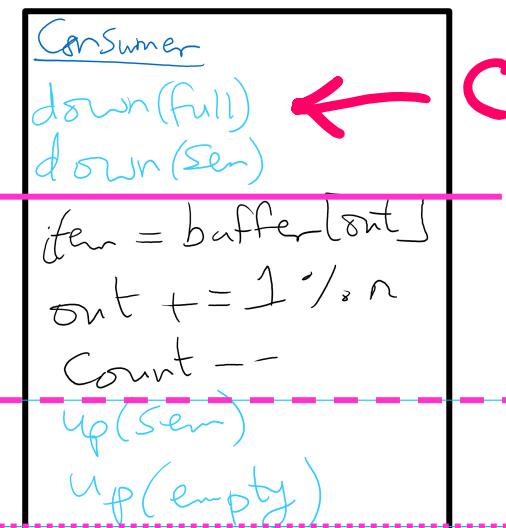
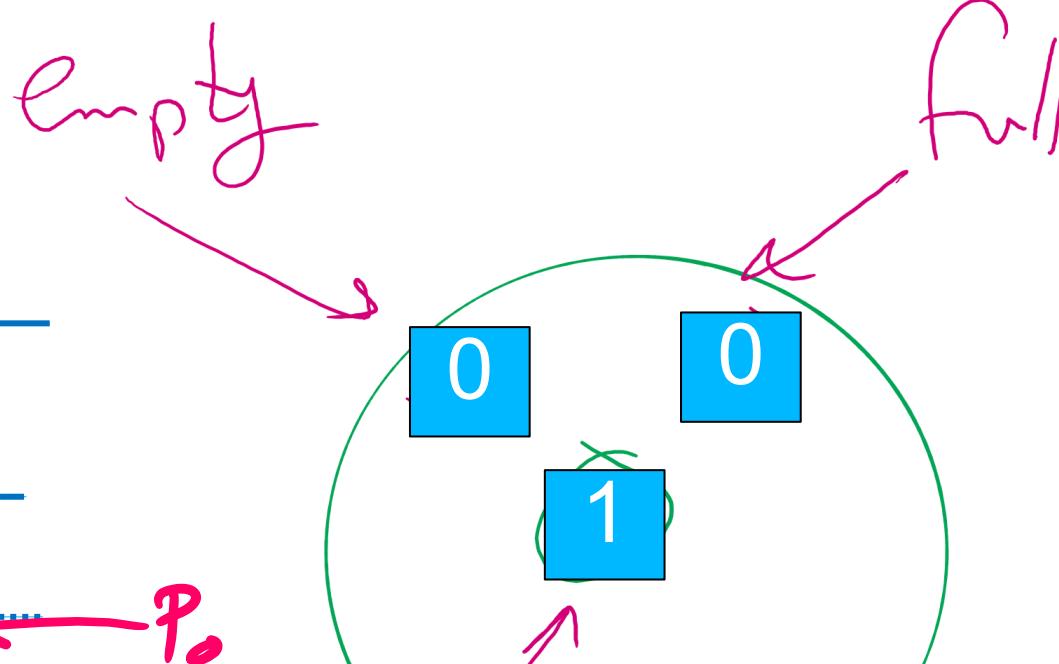
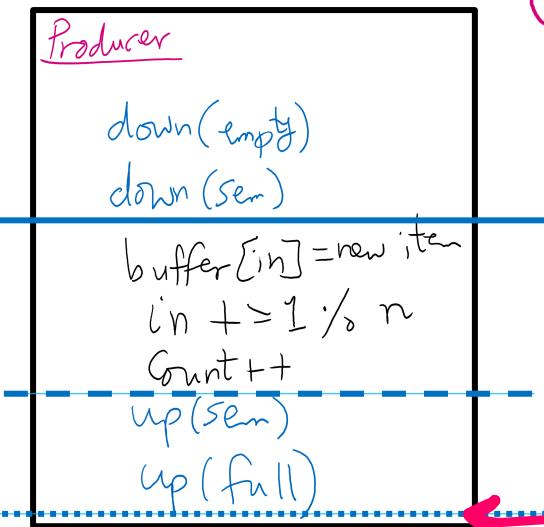
Consumer 0 arrives
 Producer 0 arrives
 Producer 0 enters
 Producer 0 leaves
 Consumer 0 enters
 Consumer 0 leaves

Full Queue

C0 enters

$$n = 1$$

Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves



Count

Sem

Full Queue

C0 enters

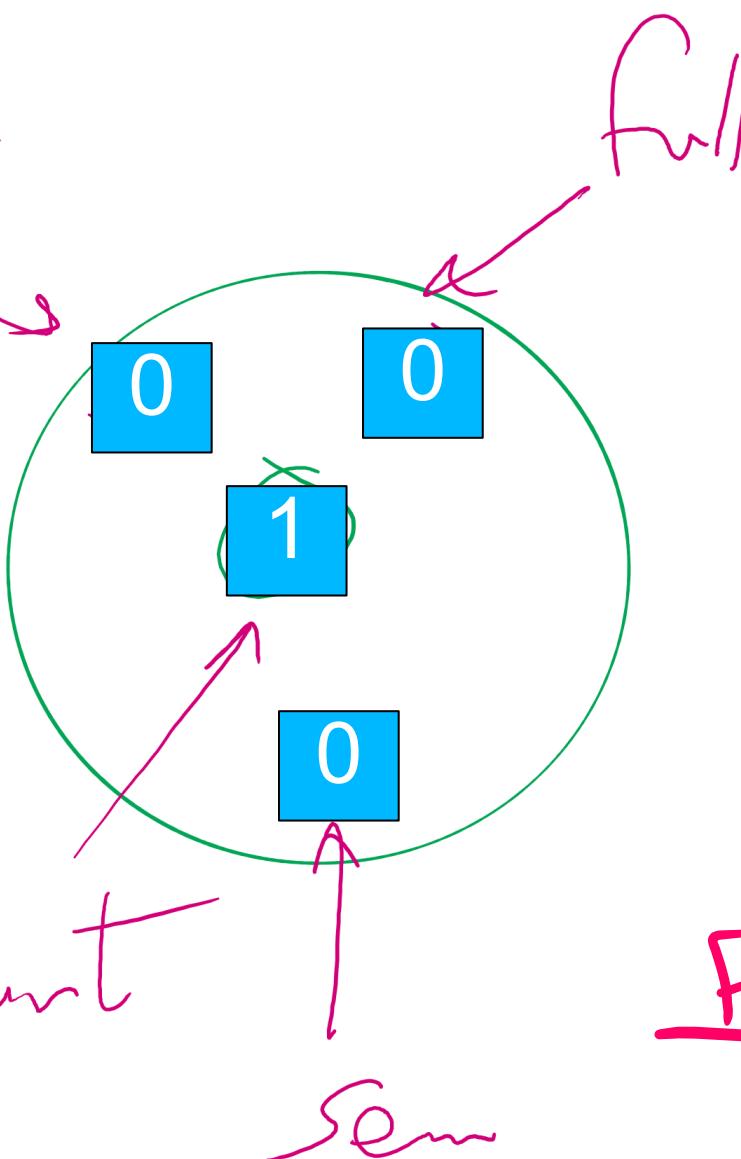
$$n = 1$$

Producer

```
down(empty)
down(sem)
buffer[in] = new item
in += 1/n
Count ++
up(sem)
up(full)
```

empty

P₀



Consumer

```
down(full)
down(sem)
item = buffer[out]
out += 1/n
Count --
up(sem)
up(empty)
```

Count

C₀

Full Queue

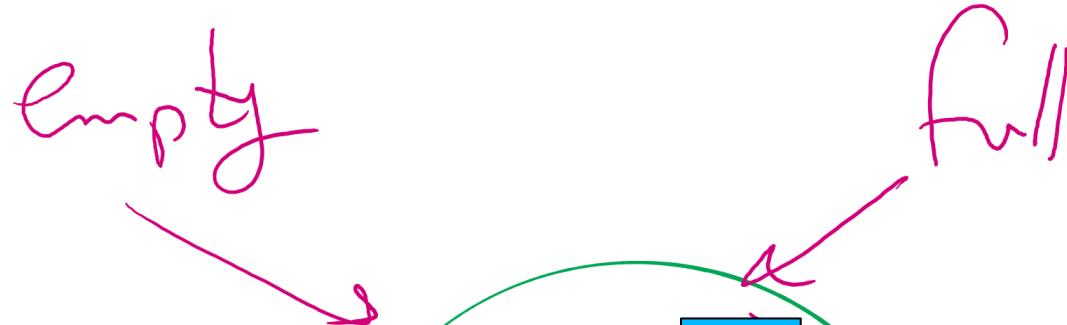
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

Can C0 enter?

$$n = 1$$

Producer

```
down(empty)
down(sem)
buffer[in] = new item
in += 1/n
Count ++
up(sem)
up(full)
```



Consumer

```
down(full)
down(sem)
item = buffer[out]
out += 1/n
Count --
up(sem)
up(empty)
```

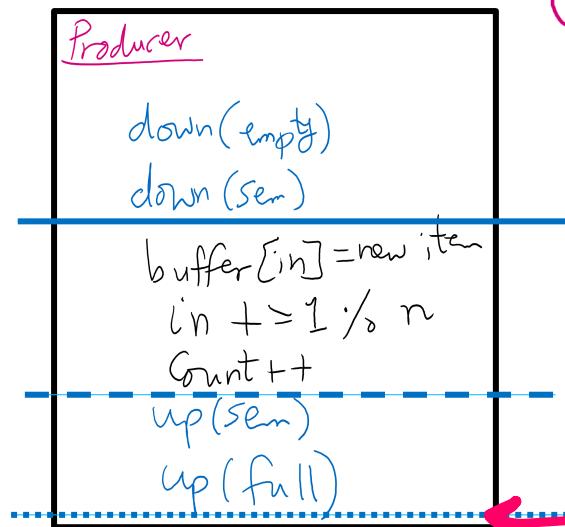


Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

Full Queue

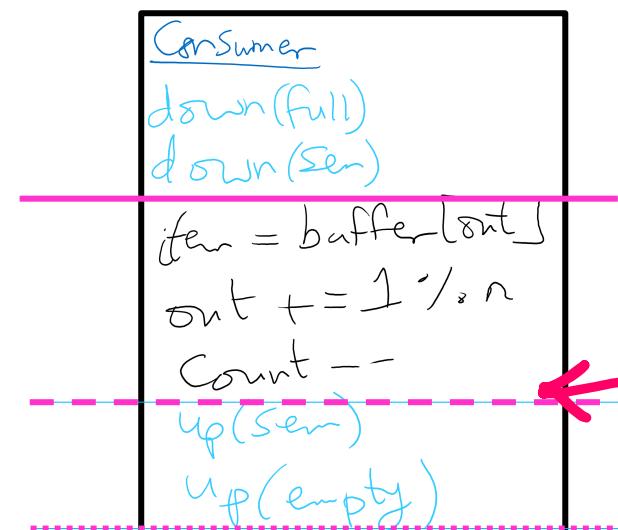
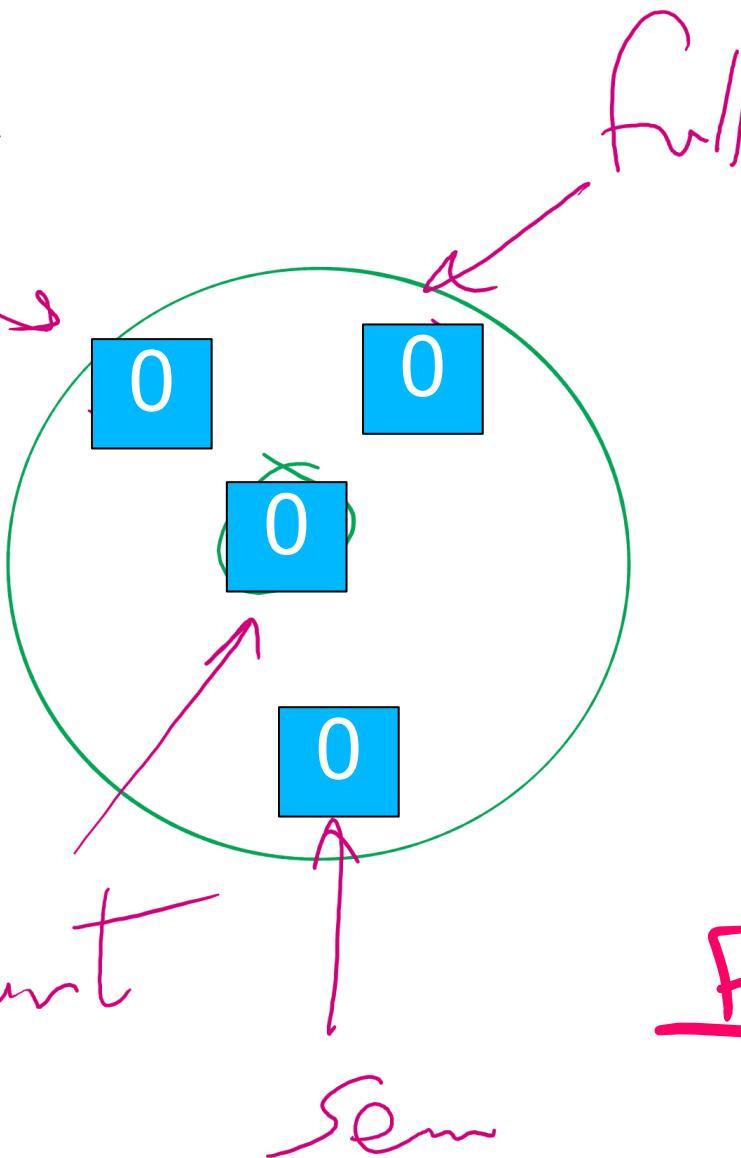
C0 enters

$$n = 1$$



empty

P₀



Count

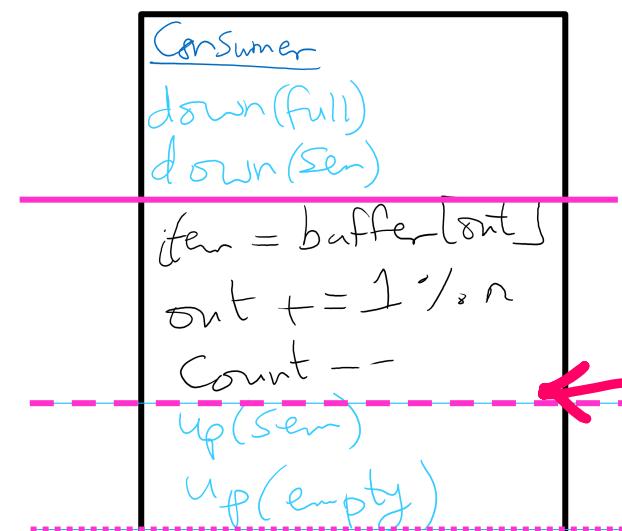
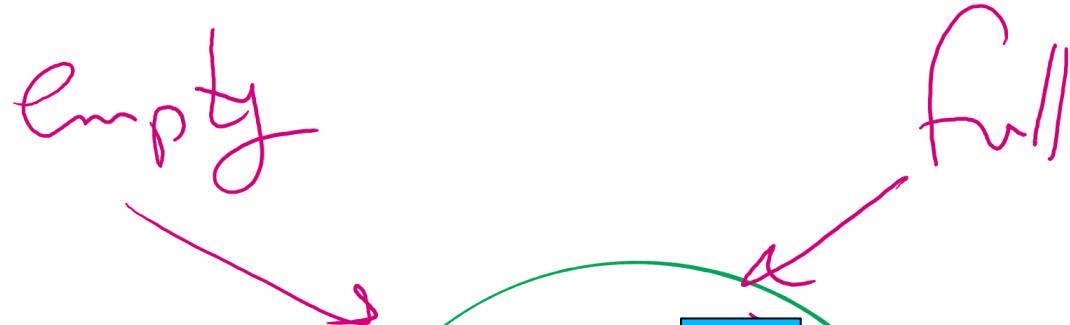
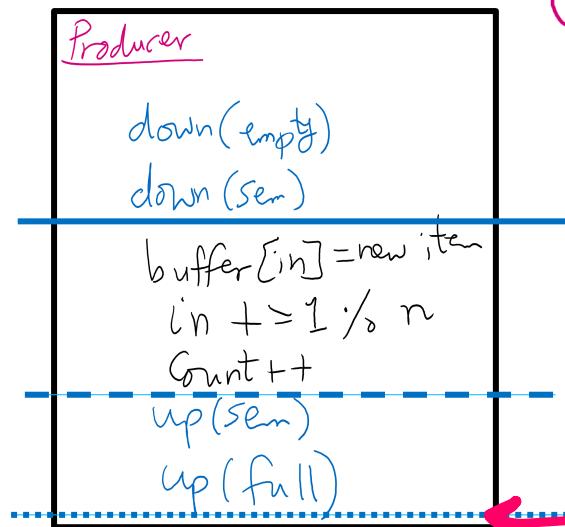
C₀

Full Queue

Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

C0 leaves

$$n = 1$$

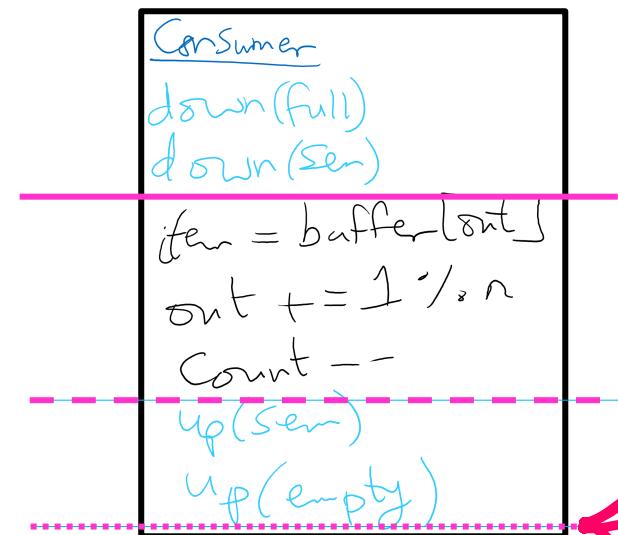
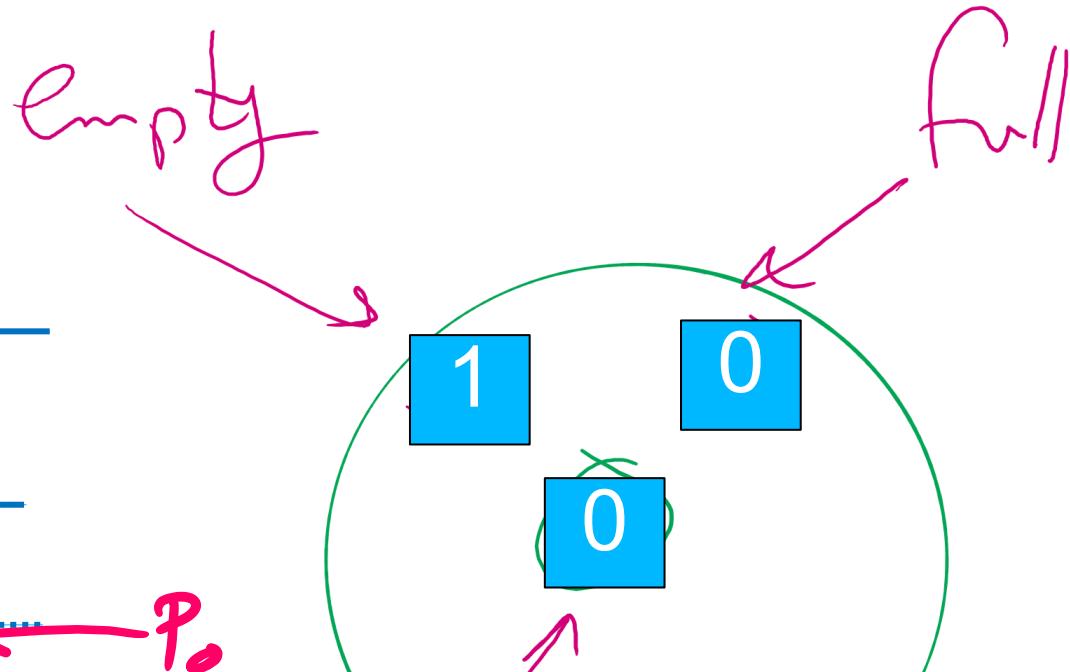
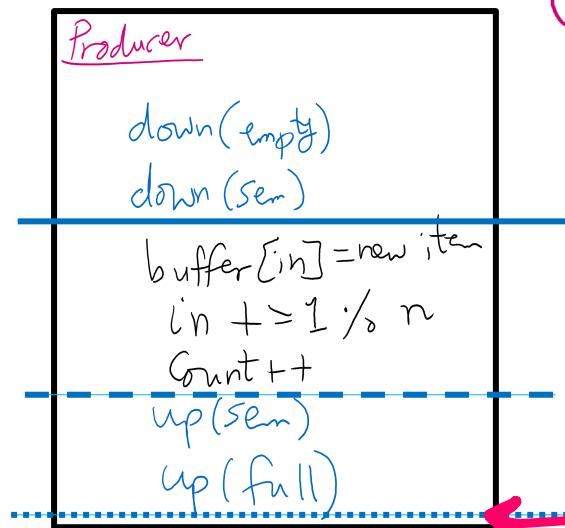


Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

Full Queue

C0 leaves

$$n = 1$$



Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

Example 3

$$\underline{n = 2}$$

C₀ arrives

G₁ arrives

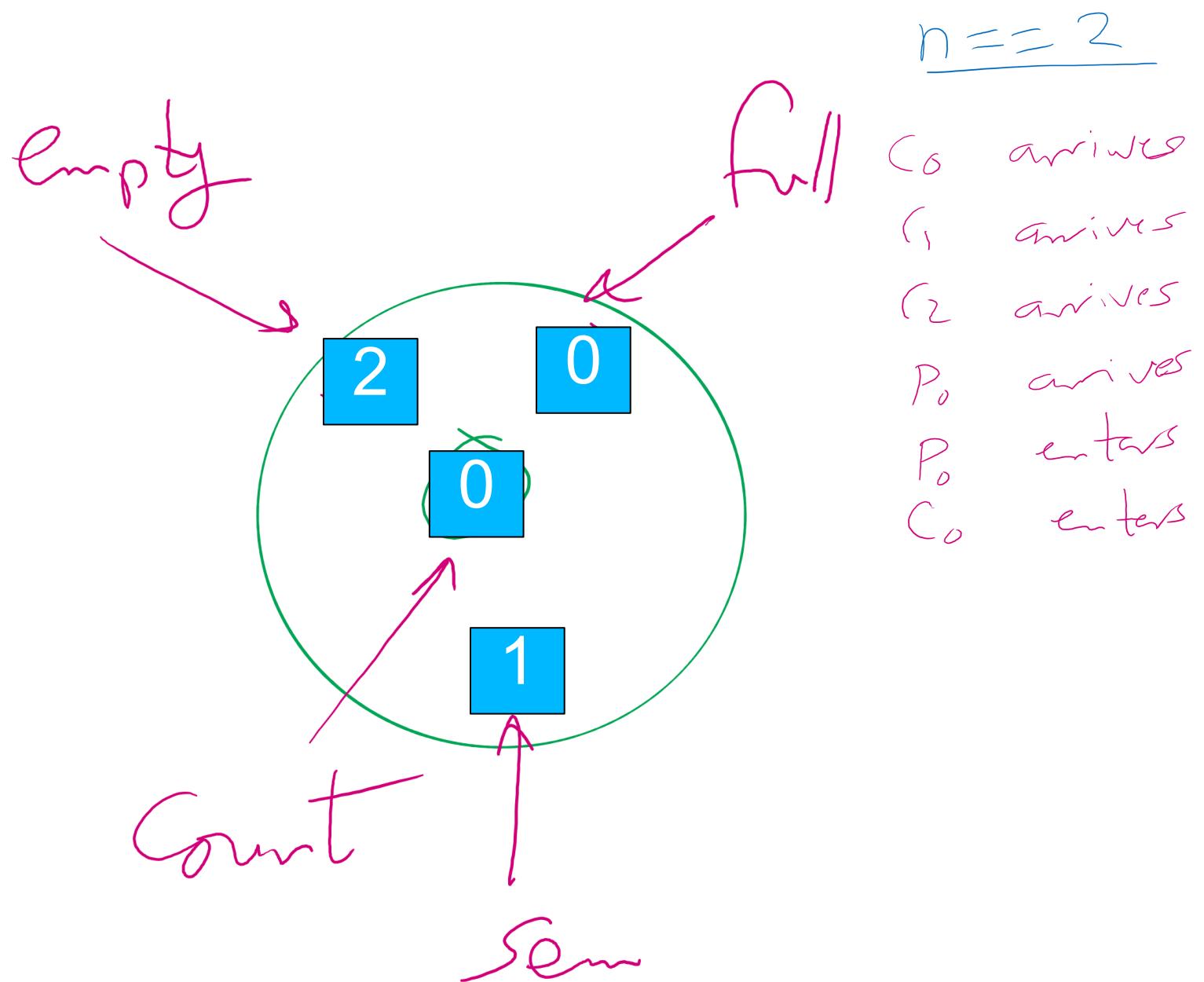
G₂ arrives

P₀ arrives

P₀ enters

C₀ enters

Initial state



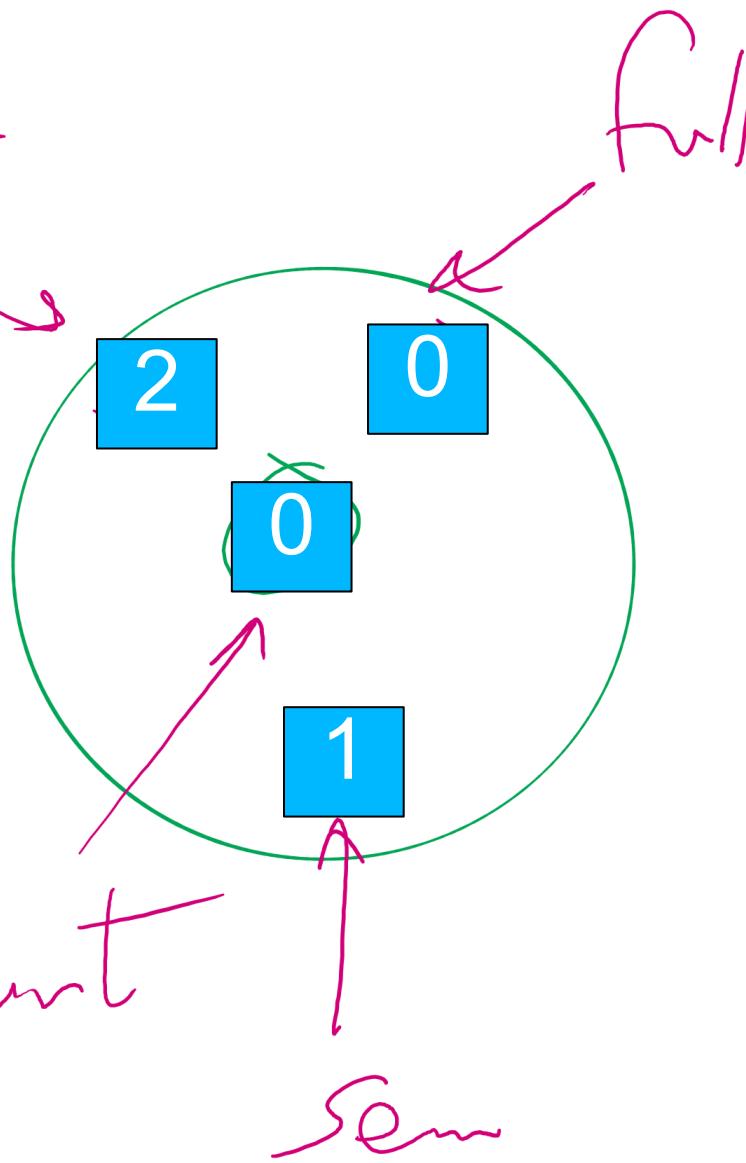
C0 arrives

$$n == 2$$

Producer

```
down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)
```

empty



$n == 2$

C ₀	arrives
C ₁	arrives
C ₂	arrives
P ₀	arrives
P ₀	enters
C ₀	enters

Consumer

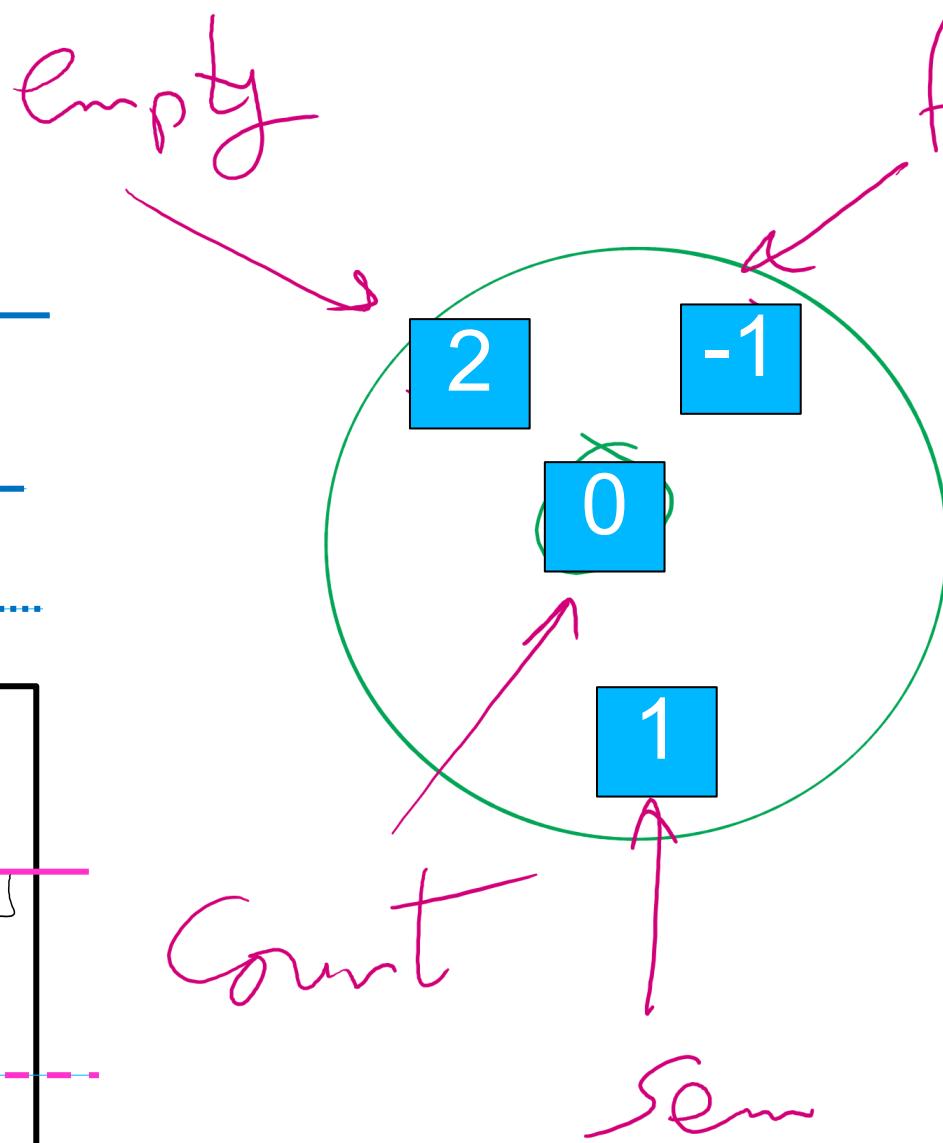
```
down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)
```

C0 arrives

$$n == 2$$

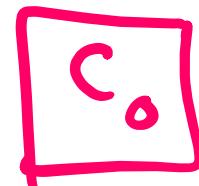
Producer

```
down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)
```



C_0 arrives
 G_1 arrives
 G_2 arrives
 P_0 arrives
 P_0 enters
 C_0 enters

Full Queue



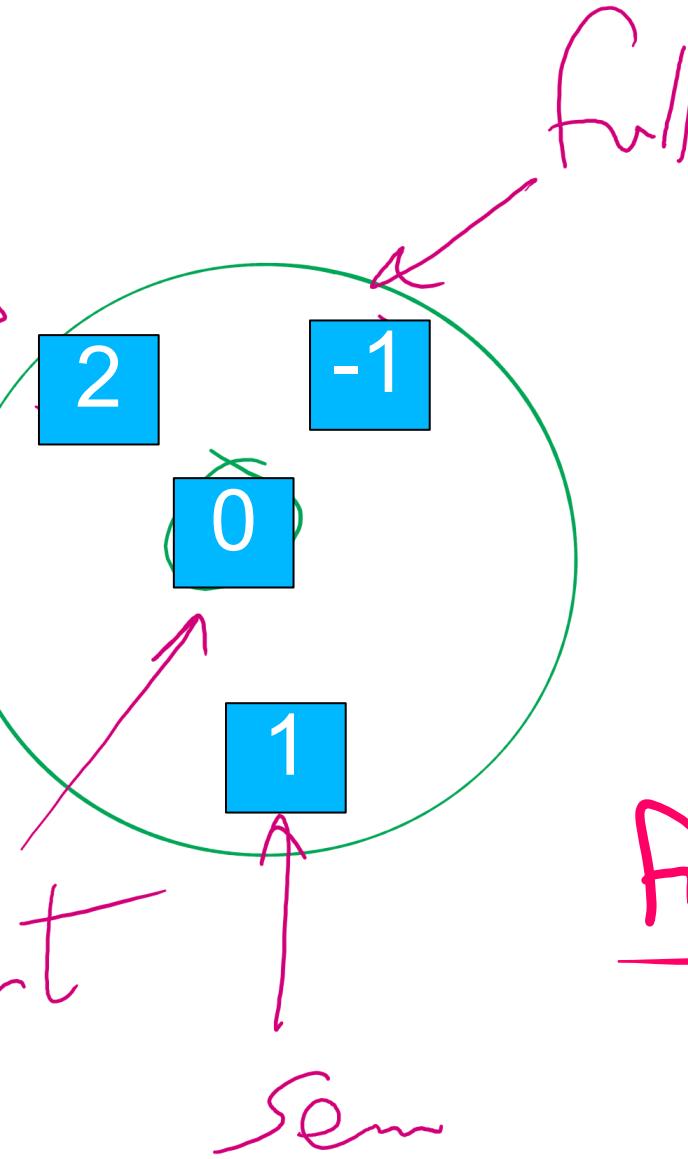
C1 arrives

$n = 2$

Producer

```
down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)
```

empty

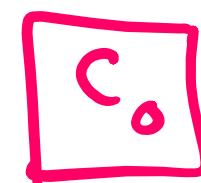


C_0 arrives
 C_1 arrives
 C_2 arrives
 P_0 arrives
 P_0 enters
 C_0 enters

Consumer

```
down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)
```

Full Queue



C1 arrives

$$n == 2$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

```

empty

full

C_0 arrives
 C_1 arrives
 C_2 arrives
 P_0 arrives
 P_0 enters
 C_0 enters

Consumer

```

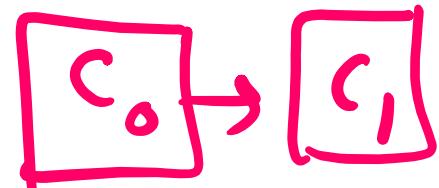
down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

```

Count

Sem

Full Queue



C2 arrives

$$n == 2$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

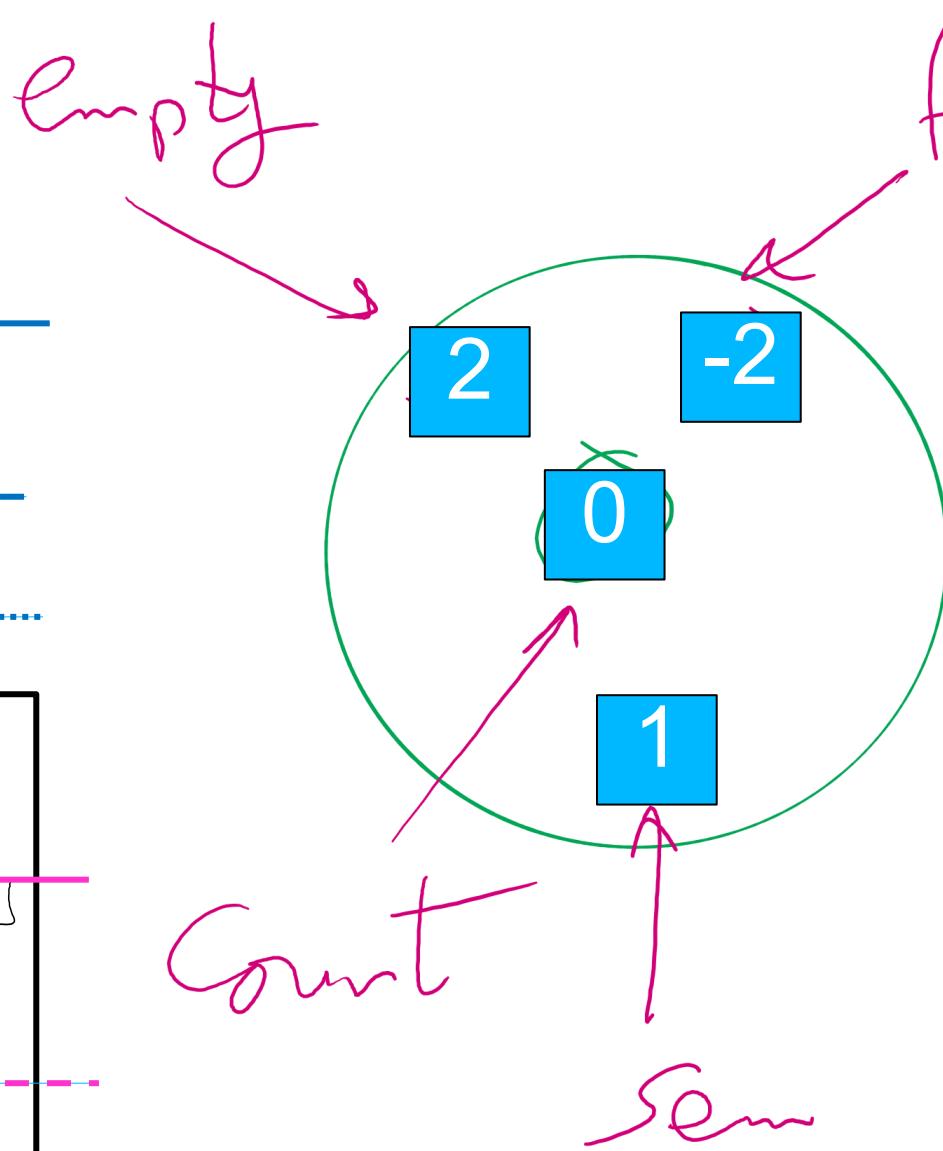
```

Consumer

```

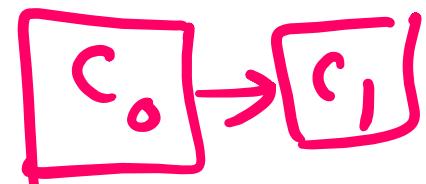
down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

```



C_0 arrives
 C_1 arrives
 C_2 arrives
 P_0 arrives
 P_0 enters
 C_0 enters

Full Queue



C2 arrives

$$n == 2$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
Count ++
up(sem)
up(full)

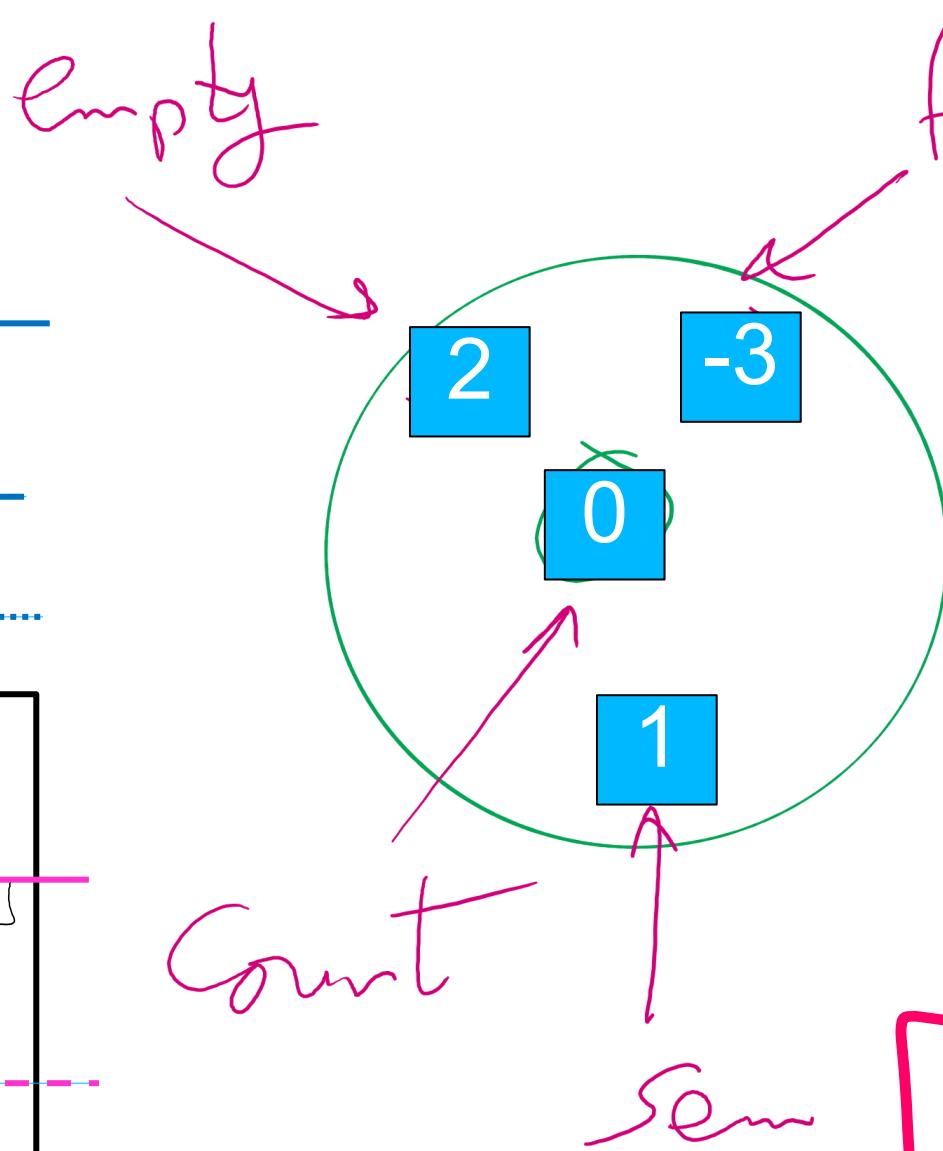
```

Consumer

```

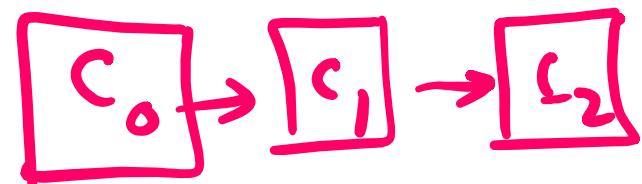
down(full)
down(sem)
item = buffer[out]
out += 1 % n
Count --
up(sem)
up(empty)

```



C_0	arrives
C_1	arrives
C_2	arrives
P_0	arrives
P_0	enters
C_0	enters

Full Queue



P0 arrives

$$n == 2$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
Count ++
up(sem)
up(full)

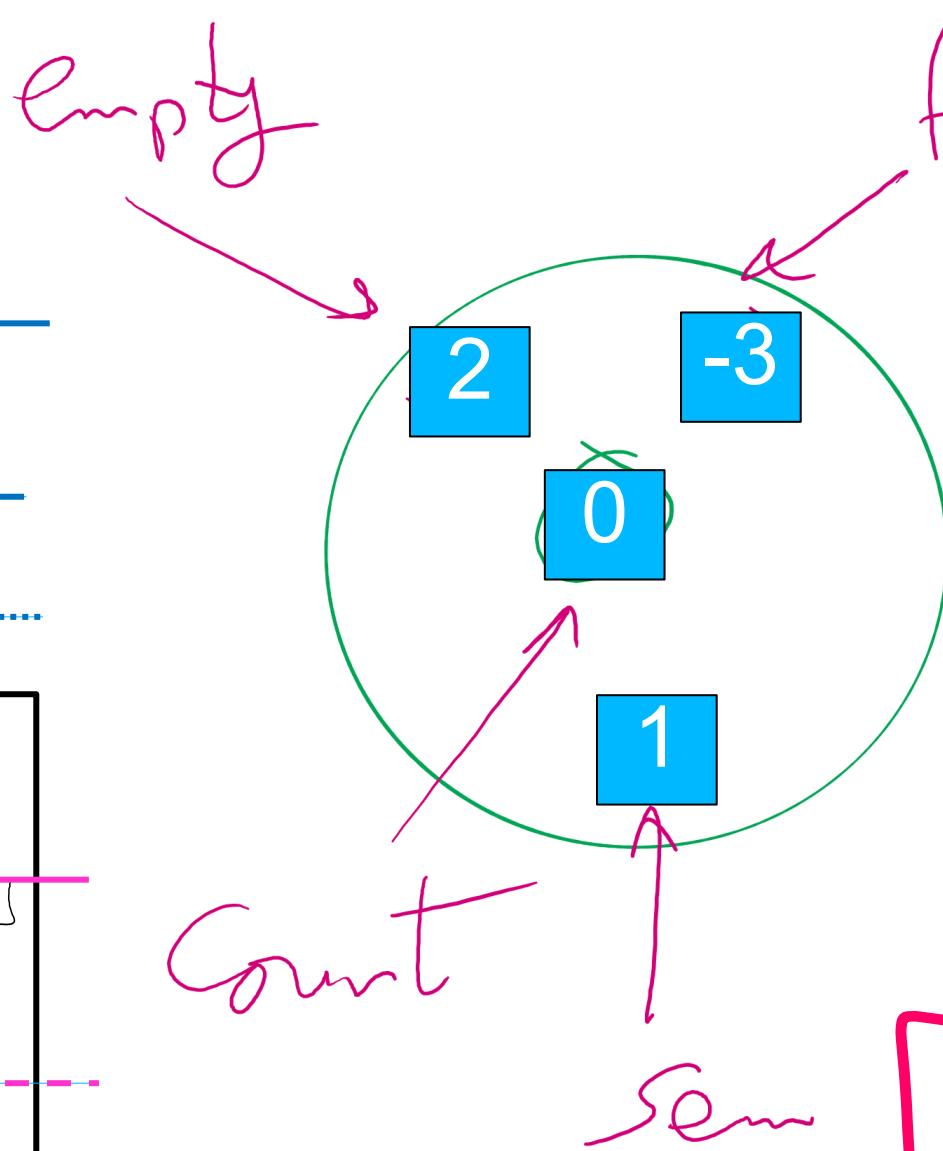
```

Consumer

```

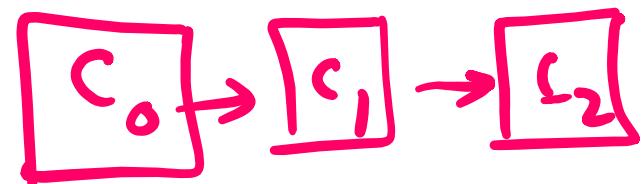
down(full)
down(sem)
item = buffer[out]
out += 1 % n
Count --
up(sem)
up(empty)

```



C_0	arrives
C_1	arrives
C_2	arrives
P_0	arrives
P_0	enters
C_0	enters

Full Queue



P0 arrives

$$n == 2$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

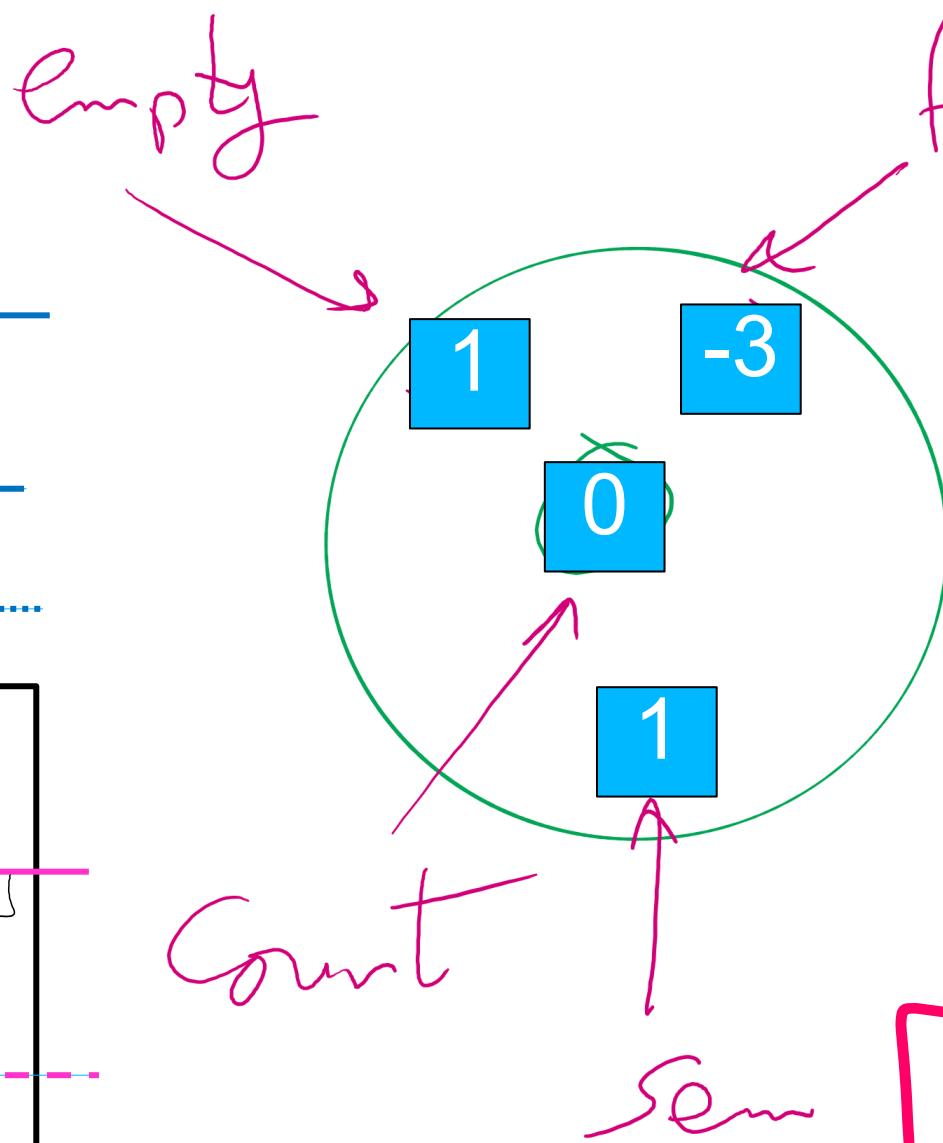
```

Consumer

```

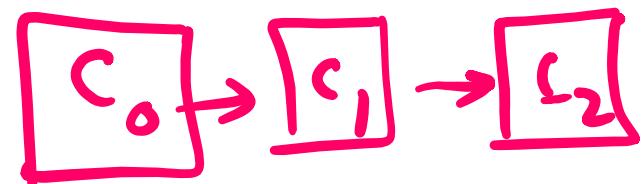
down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

```



C_0 arrives
 C_1 arrives
 C_2 arrives
 P_0 arrives
 P_0 enters
 C_0 enters

Full Queue



P0 arrives

$$n == 2$$

Producer

```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

```

empty

P0



full

C0	arrives
C1	arrives
C2	arrives
P0	arrives
P0	enters
C0	enters

Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

```

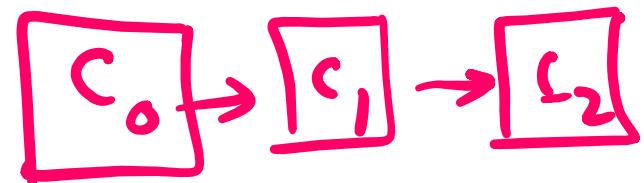
C0, C1, C2

Count



sem

Full Queue



P0 enters

$$n == 2$$

Producer

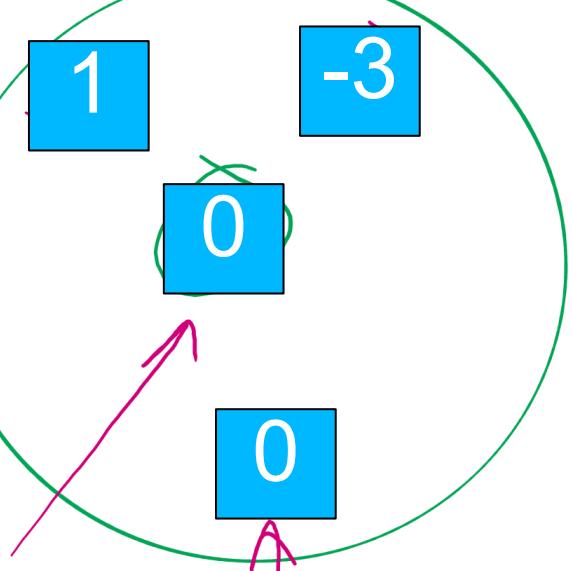
```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

```

empty

P0



C0	arrives
C1	arrives
C2	arrives
P0	arrives
P0	enters
C0	enters

Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

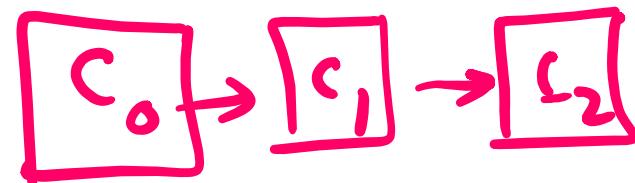
```

C0, C1, C2

Count

Sem

Full Queue



P0 enters

$$n == 2$$

Producer

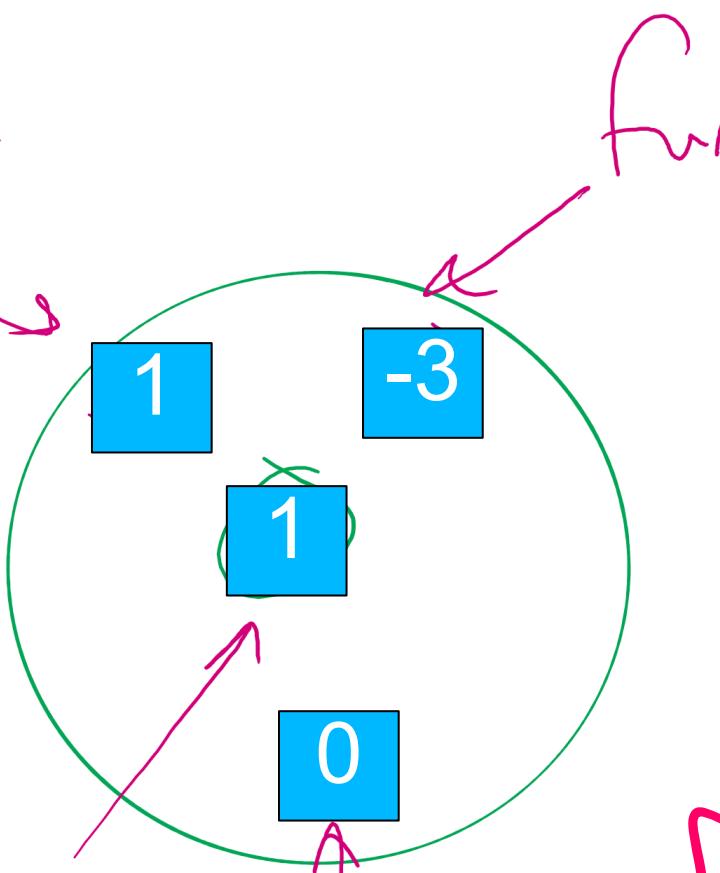
```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

```

empty

P0



C0	arrives
C1	arrives
C2	arrives
P0	arrives
P0	enters
C0	enters

Consumer

```

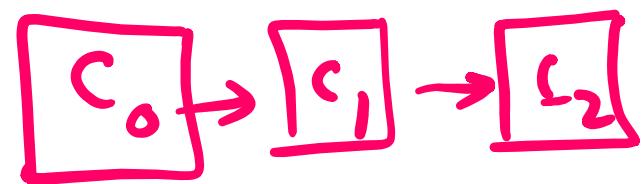
down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

```

C0, C1, C2

Count

Full Queue



C0 enters

$$n == 2$$

Producer

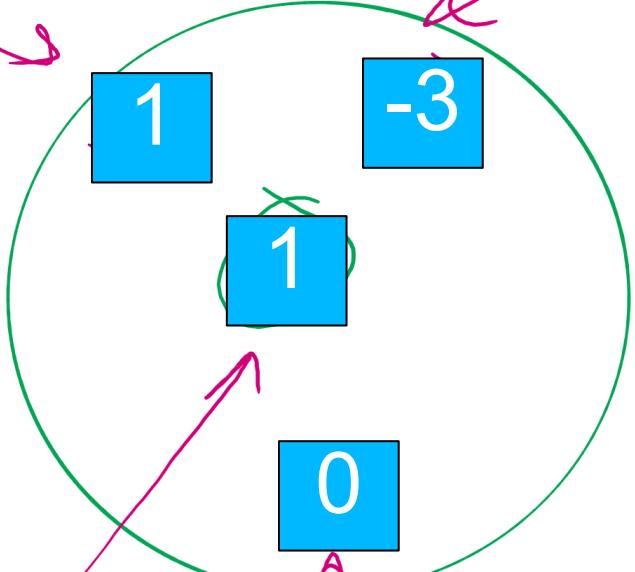
```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

```

empty

P₀



C ₀	arrives
C ₁	arrives
C ₂	arrives
P ₀	arrives
P ₀	enters
C ₀	enters

Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

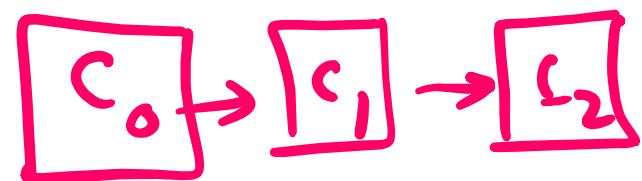
```

C₀, C₁, C₂

Count

Sem

Full Queue



Can C0 enter?

$$n == 2$$

Producer

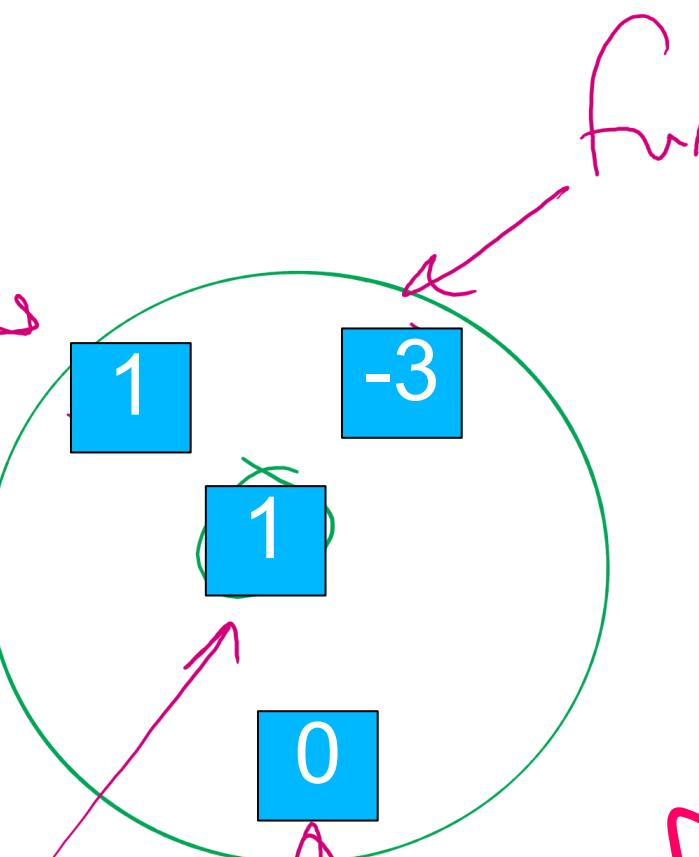
```

down(empty)
down(sem)
buffer[in] = new item
in += 1 / n
Count ++
up(sem)
up(full)

```

empty

P₀



C ₀	arrives
C ₁	arrives
C ₂	arrives
P ₀	arrives
P ₀	enters
C ₀	enters

Consumer

```

down(full)
down(sem)
item = buffer[out]
out += 1 / n
Count --
up(sem)
up(empty)

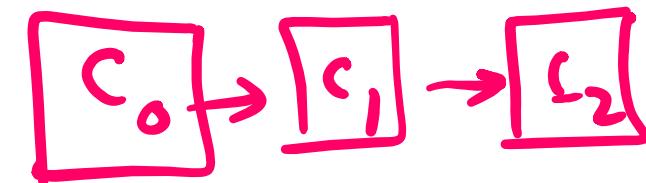
```

C₀, C₁, C₂

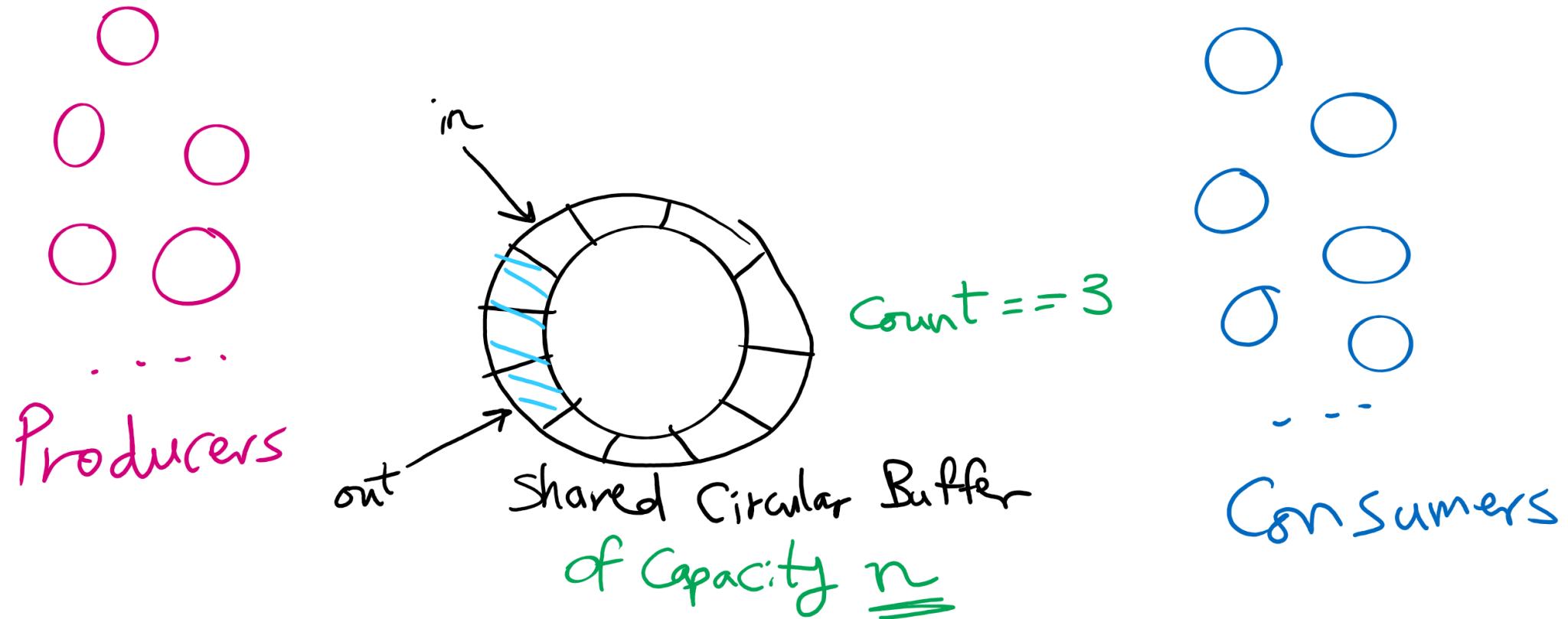
Count

Sem

Full Queue



Producers Consumers Problem



Solving Producers Consumers using Semaphores

Semaphore empty($\leq n$), full(0)
Mutex Sem(1);

Producer

down(empty)

down(Sem)

buffer[in] = new item

in += 1 % n

Count ++

up(Sem)

up(full)

Consumer

down(full)

down(Sem)

item = buffer[out]

out += 1 % n

Count --

up(Sem)

up(empty)

Is this sequence feasible?

n == 3

for (i=0; i<3; i++){

Pi arrives

Pi enters

Pi leaves

}

P3 arrives

C0 arrives

C0 enters

C0 leaves

P3 enters

P3 leaves

Some thoughts

- If we have one producer and one consumer
 - do we need count?
 - do we need the mutex?
- For multiple producers and consumers
 - what benefit do we get if we have one mutex for producers and one for consumers?

Warning!

It is easy to make mistakes when using semaphores

Let's make a “small” change

Semaphore empty($\leq n$), full(0)
Mutex Sem(1);

Producer

down(empty)
down(Sem)

buffer[in] = new item

in += 1 % n

Count ++

up(Sem)

up(full)

Consumer

down(full)
down(Sem)

item = buffer[out]

out += 1 % n

Count --

up(Sem)

up(empty)

Let's make a “small” change

```
Semaphore empty(n), full(0);  
Mutex sem(1);
```

Producer

```
down(sem)  
down(empty)  
buffer[in] = new item  
in = (in + 1) % n  
count++  
up(empty)  
up(sem)
```

Consumer

```
down(full)  
down(sem)  
Item = buffer[out]  
out = (out + 1) % n  
count--  
up(sem)  
up(full)
```

Is this sequence feasible?

n == 3

for (i=0; i<3; i++){

Pi arrives

Pi enters

Pi leaves

}

P3 arrives

C0 arrives

C0 enters

C0 leaves

P3 enters

P3 leaves

Solution

- Condition Variables
 - Yet another construct (Add to Spinlock and Semaphore)
 - Has 3 operations
 - `wait()`
 - `signal()`
 - `broadcast()`
 - Not foreign to us at all
 - Every object variable in Java is a Condition Variable

Solution

- Condition Variables operations must be called while holding a mutex lock
- `wait()`
 - unlock mutex
 - block process
 - when awake, relock mutex
 - when successful, return
- `signal()`
 - wakeup one waiting process in the condition variable's queue if any
- `broadcast()`
 - wakeup all waiting processes in the condition variable's queue if any

Solving Bounded Buffer Using Condition Variables

Mutex Sem;
ConditionVariable CV;

Producer

```
down (Sem)
while (count == n)
    (V.wait ())
    buffer[i] = new item
    i = (i + 1) % n
    count ++
CV.broadcast ()
up (Sem)
```

Consumer

```
down (Sem)
while (count == 0)
    (V.wait ())
item = buffer [at]
at = (at + 1) % n
count --
CV.broadcast ()
up (Sem)
```

Solving Bounded Buffer Using Condition Variables

Mutex Sem;
ConditionVariable CV;

Producer

```
down (Sem)
while (count == n)
    (V.wait ())
    buffer[i] = new item
    i = (i + 1) % n
    count ++
    CV.broadcast ()
up (Sem)
```

Consumer

```
down (Sem)
while (count == 0)
    (V.wait ())
item = buffer [at]
at = (at + 1) % n
count --
CV.broadcast ()
up (Sem)
```