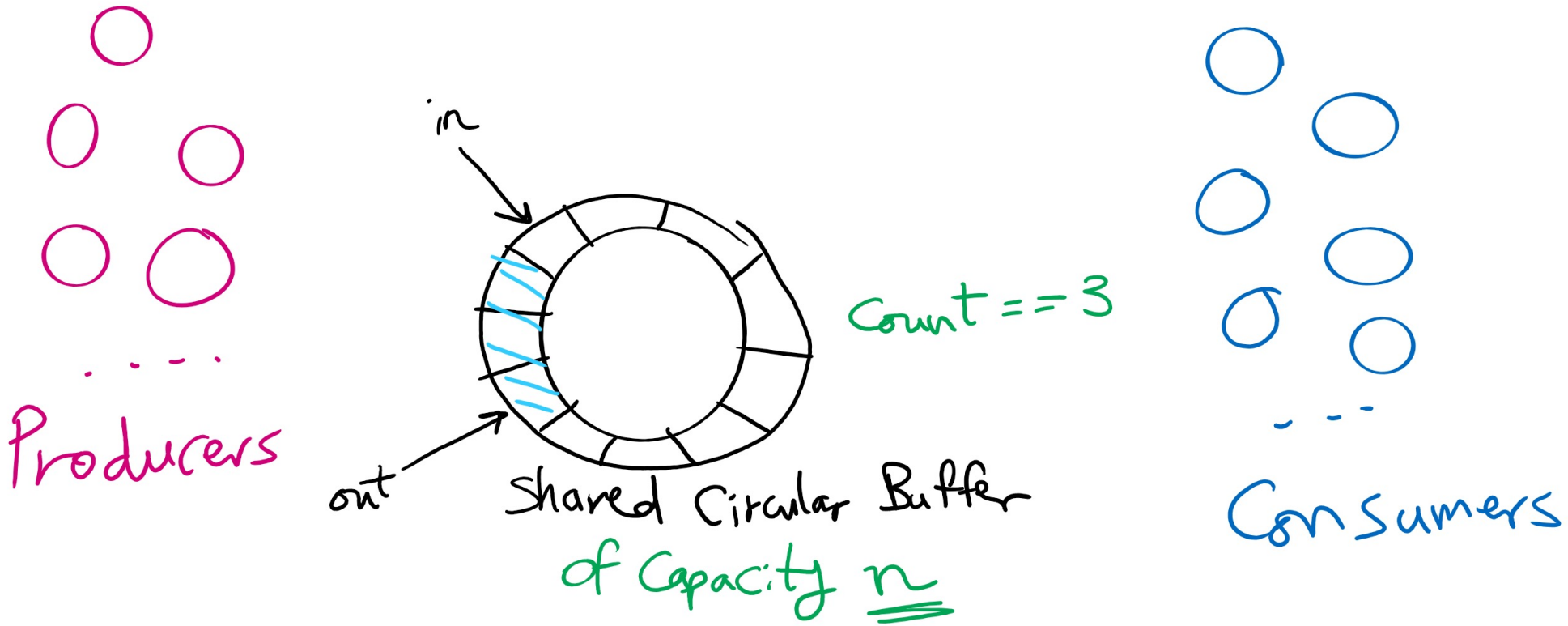
- review producer/consumer problem
- locks and condition variables.

P3a

- `/home/wireman/project3a/p3a/tests/`

Produces Consumers Problem

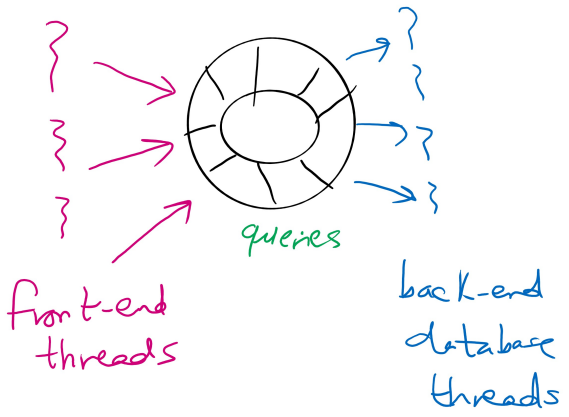


Producers Consumers Problem is everywhere!

- 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(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)
```

How to trace the solution?

Let’s define some events.

Producer arrives

Moves as far as possible until the solid line

Producer

down(empty)

down(sem)

---

buffer[in] = new item

in += 1 % n

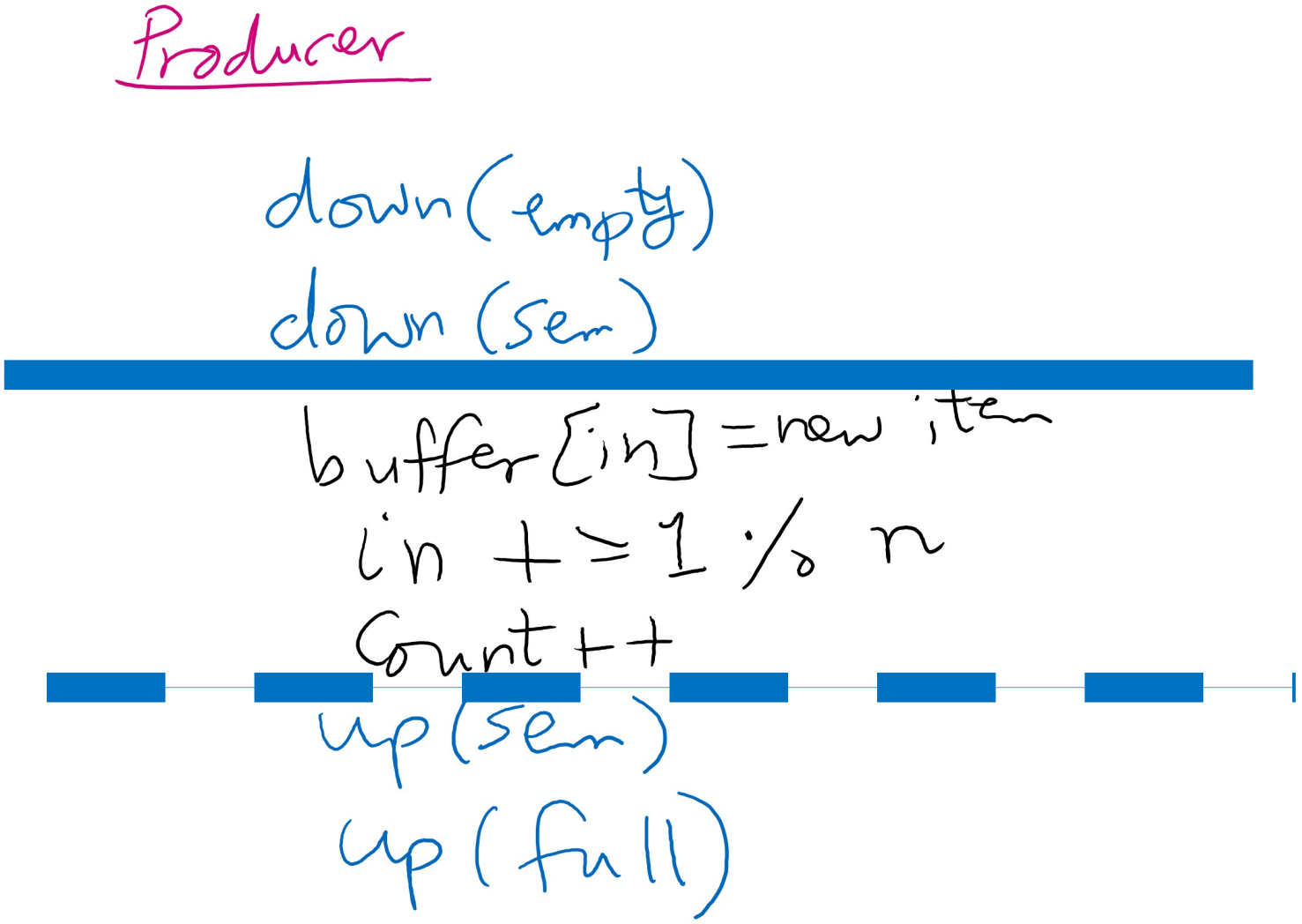
count++

up(sem)

up(full)

Producer enters

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



Producer 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

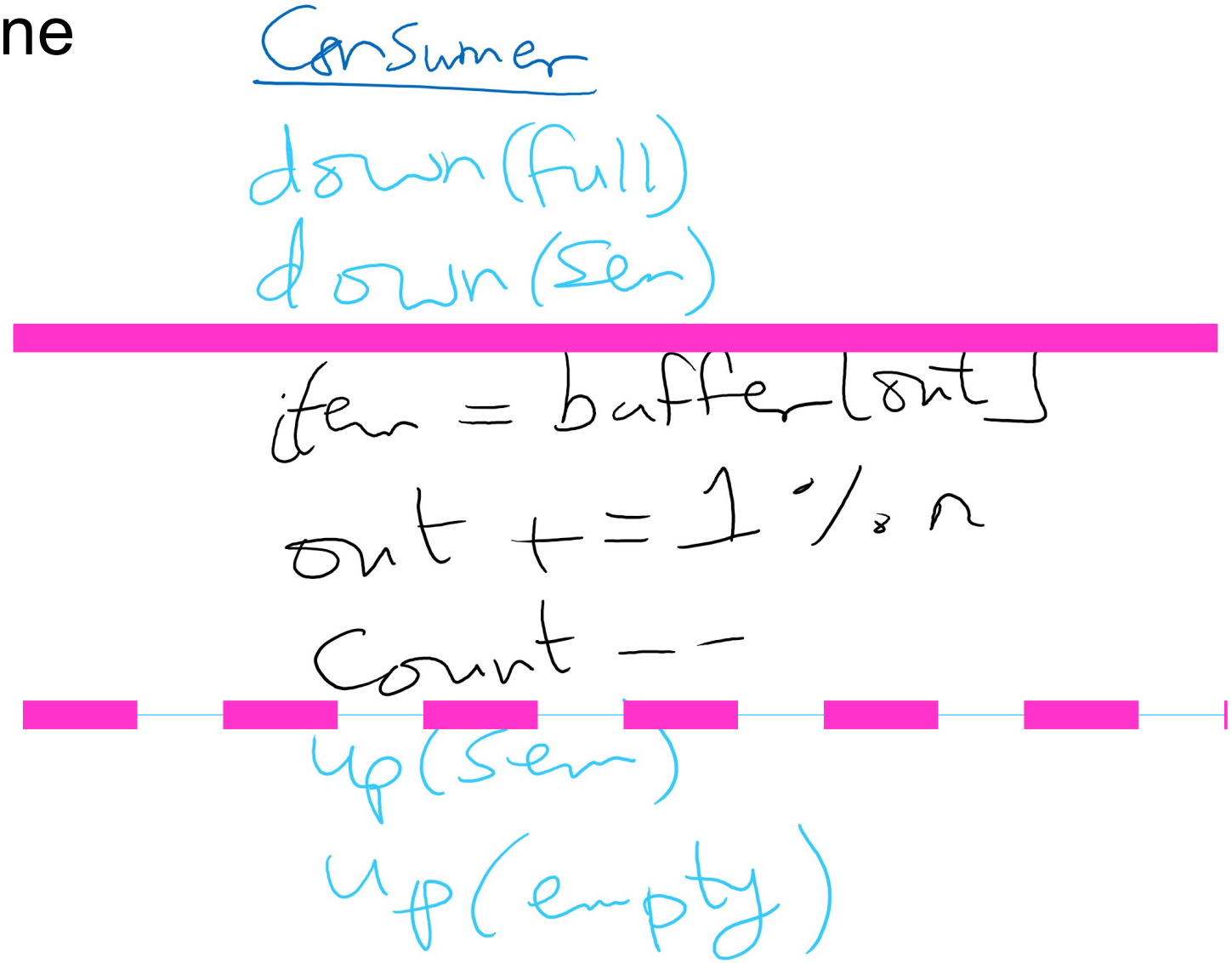
count --

up(sem)

up(empty)

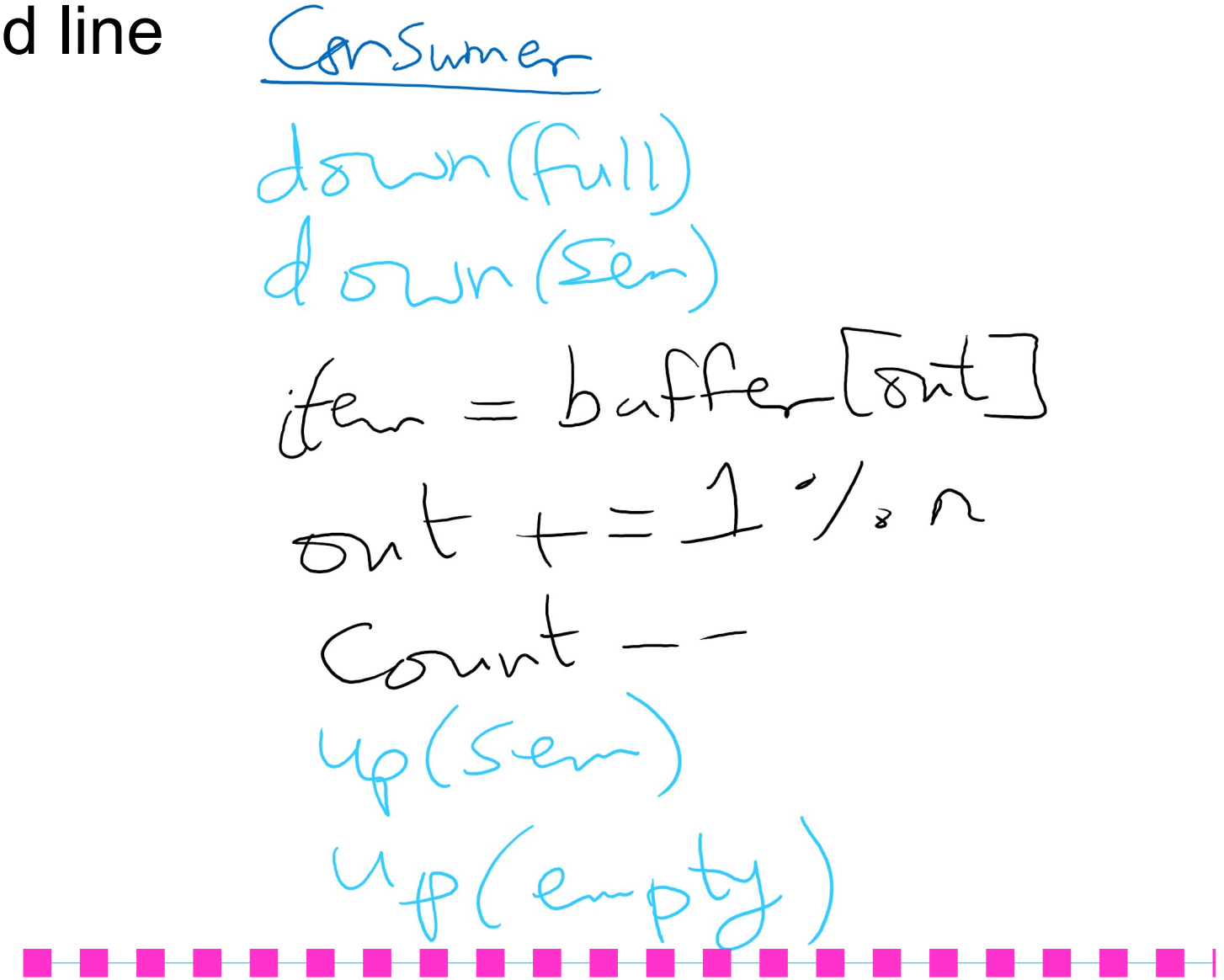
Consumer enters

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



Consumer leaves

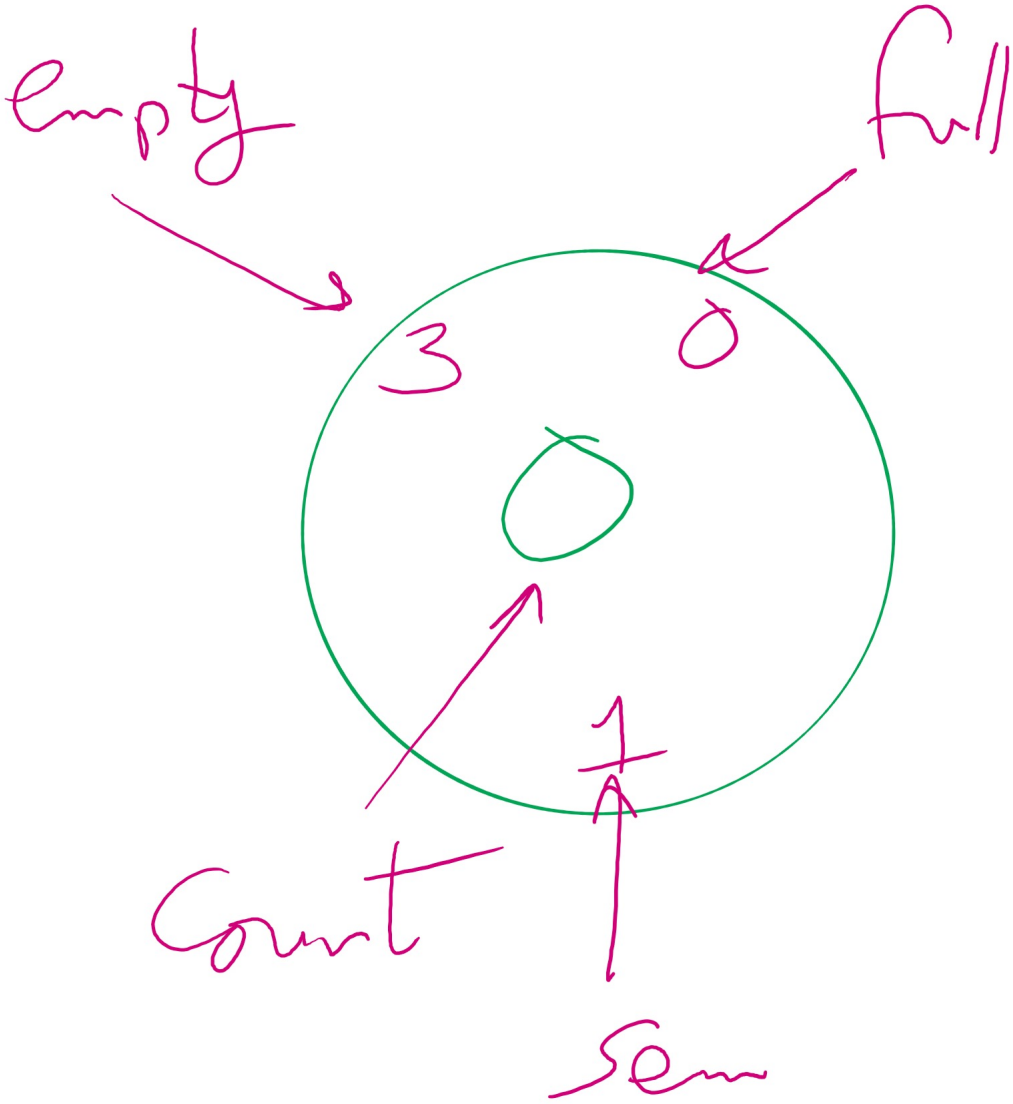
Moves as far as possible past the dashed line and until the dotted line



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

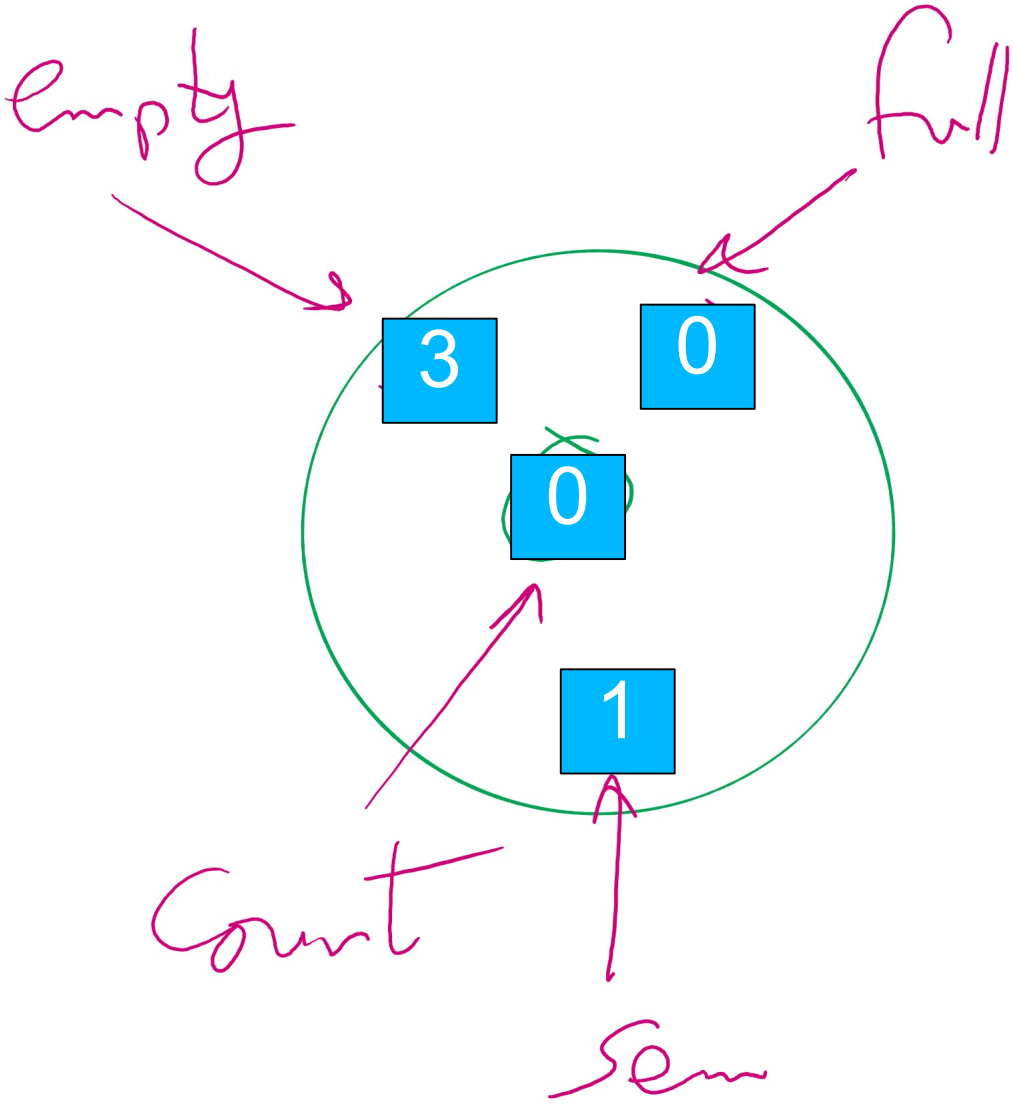


Example 1

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



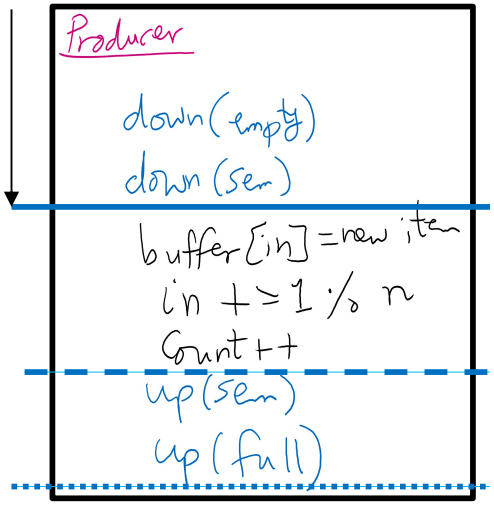
n == 3

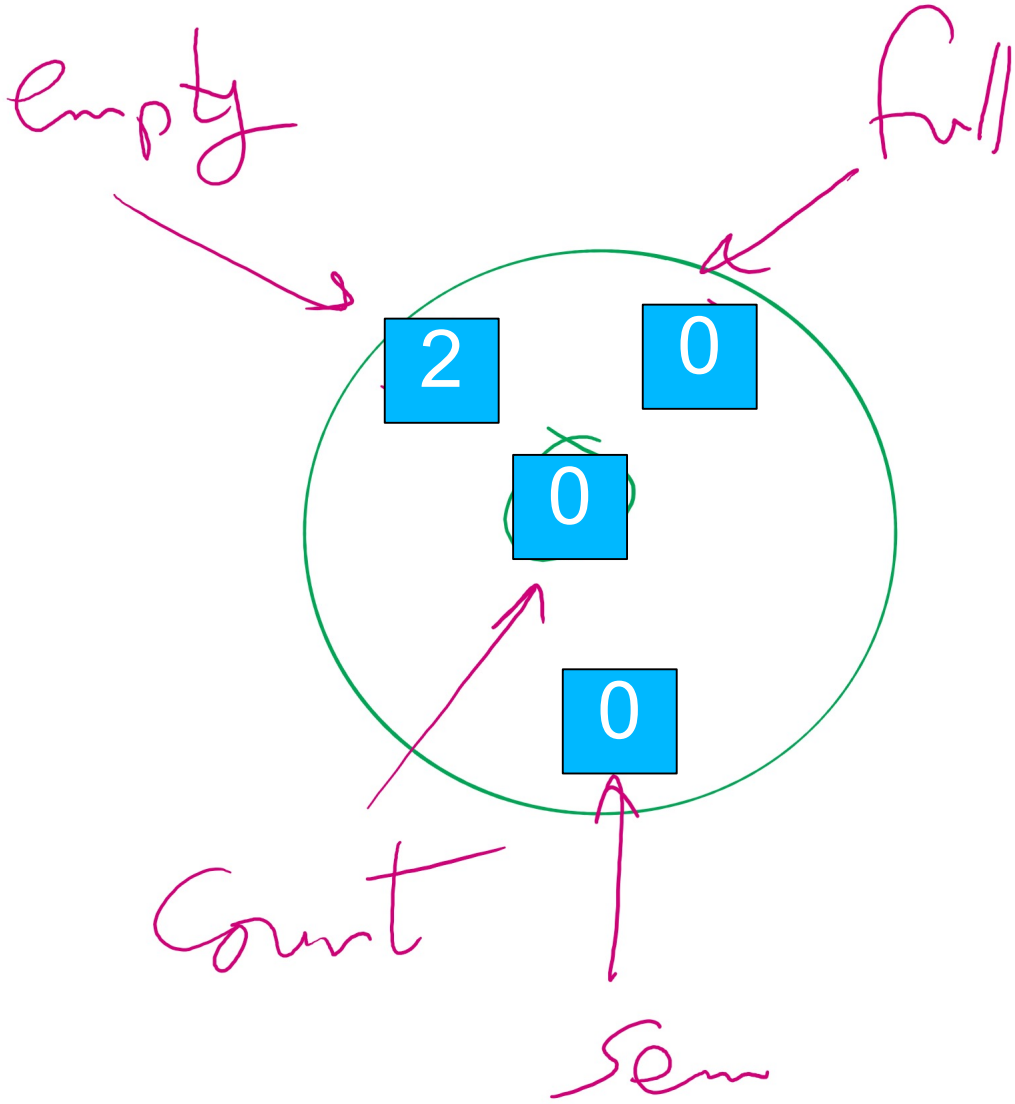
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

P0 arrives

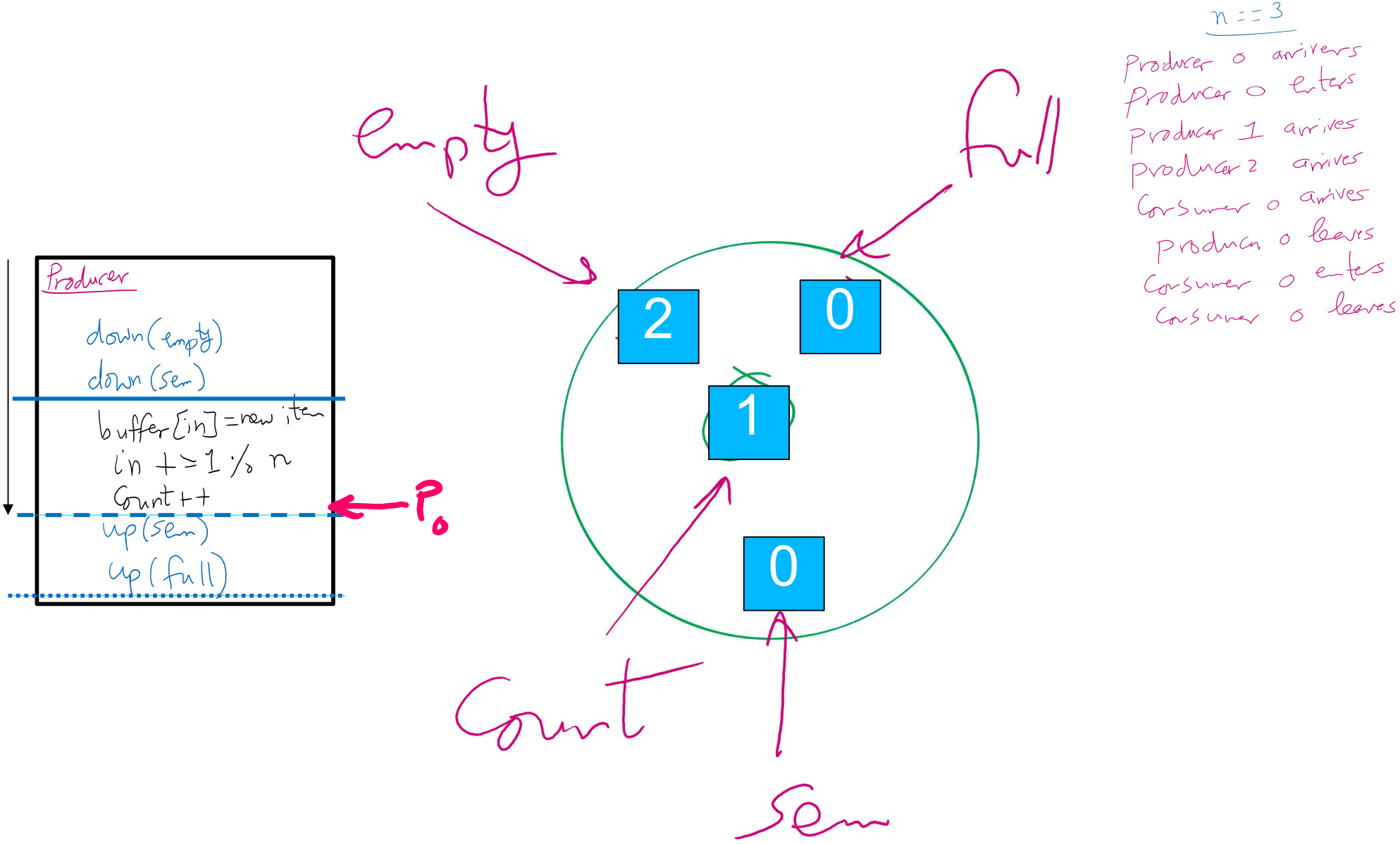
n == 3

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





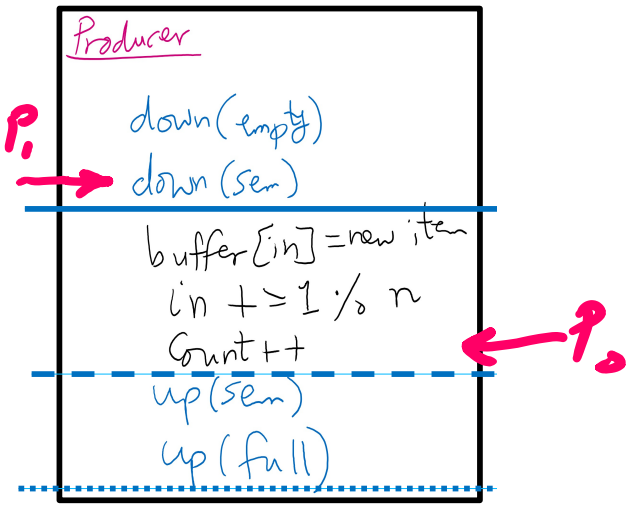
P0 enters

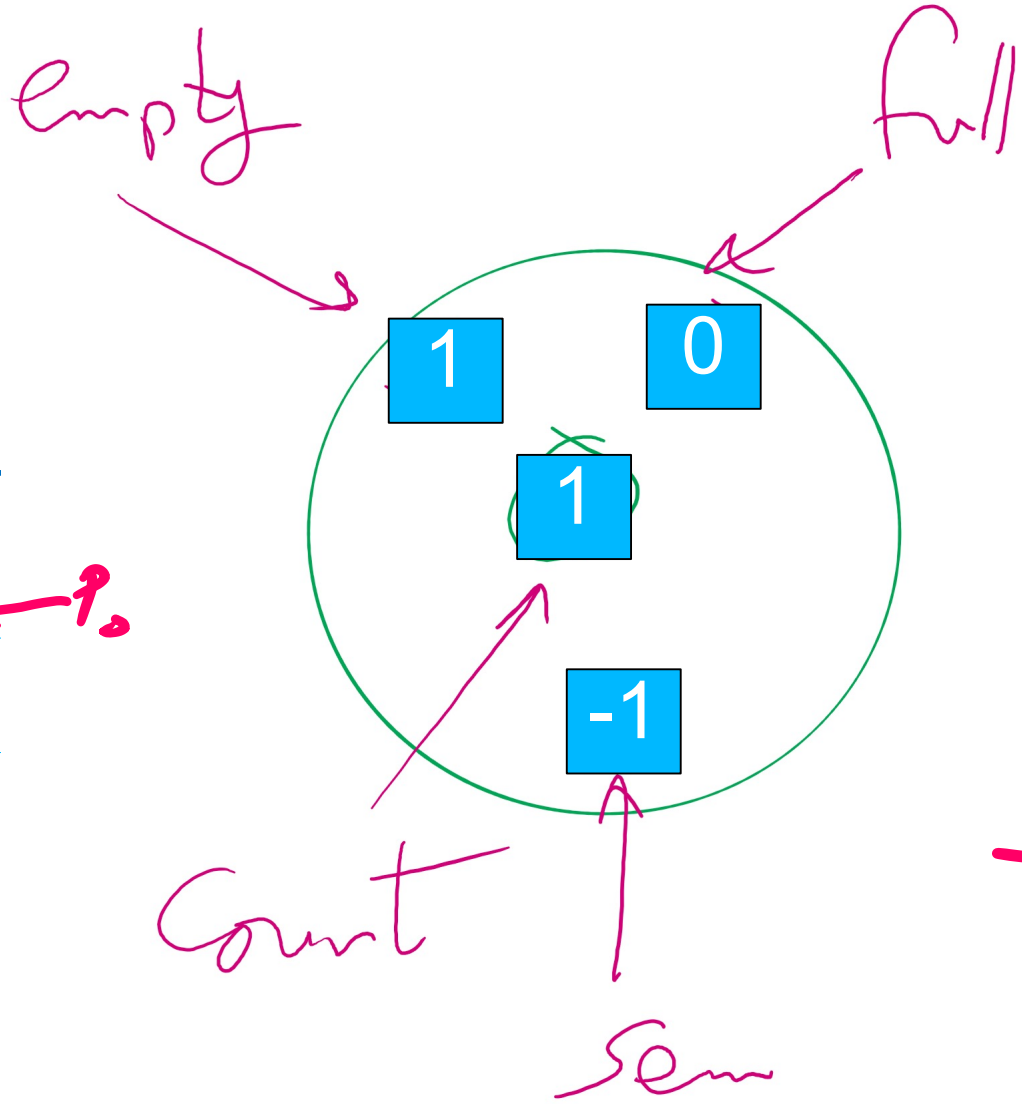

n == 3
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
empty
full
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
2
0
1
0
count
sem

P1 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

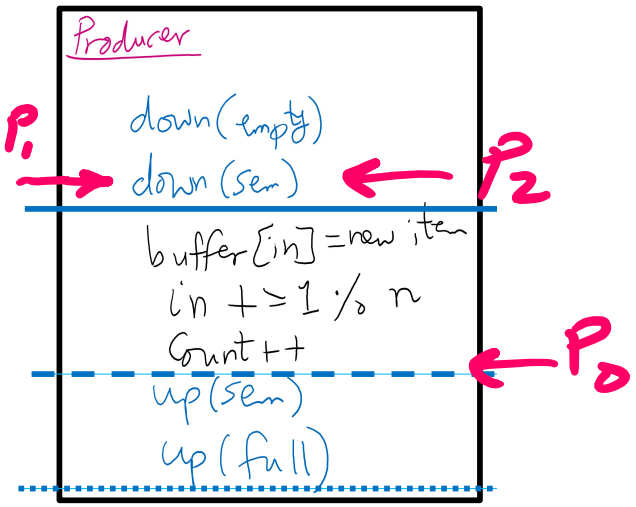


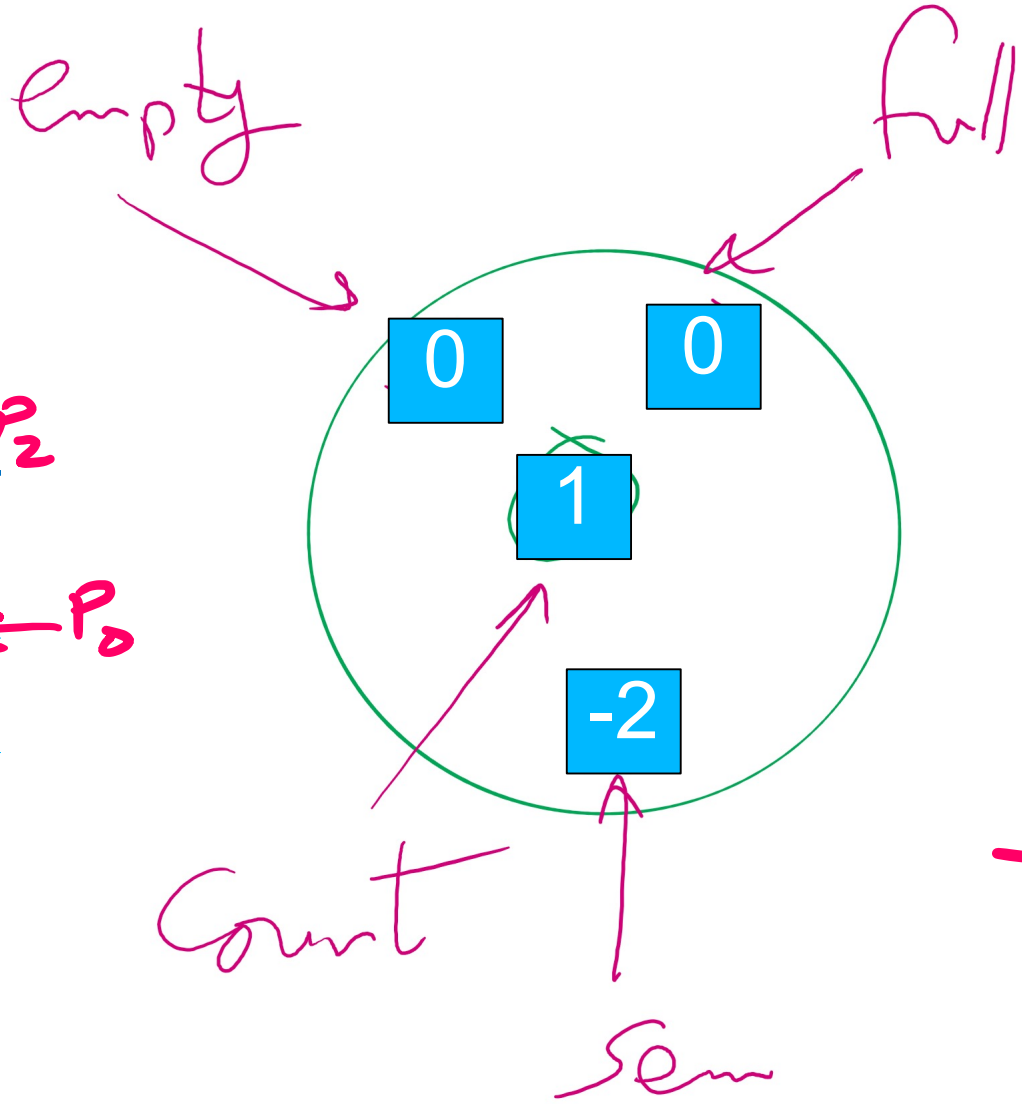


Sem Queue

P1

P2 arrives

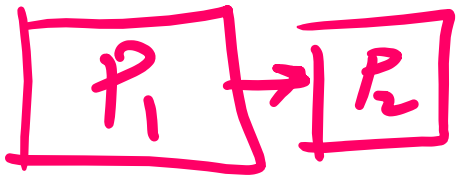




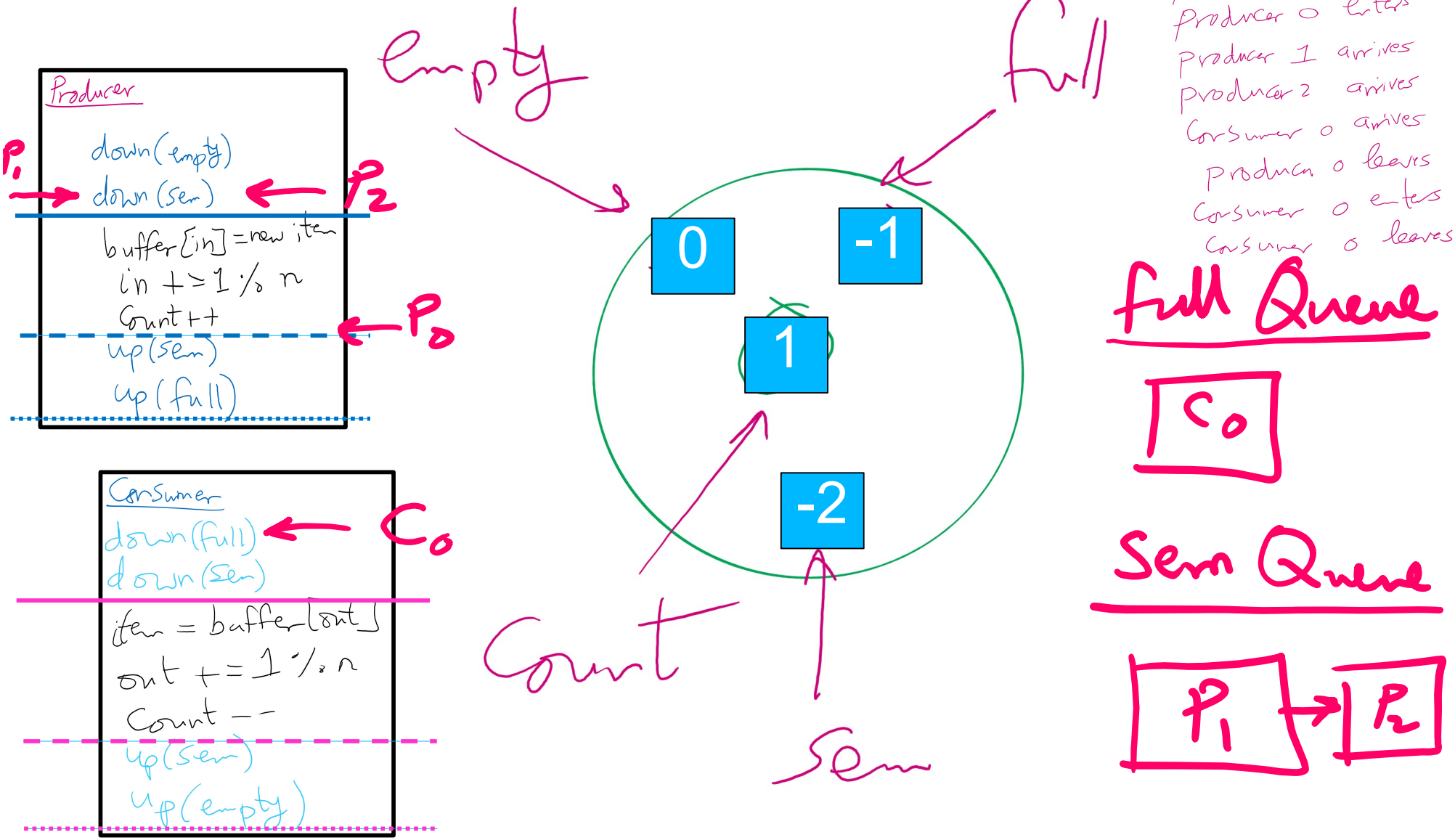
n == 3

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

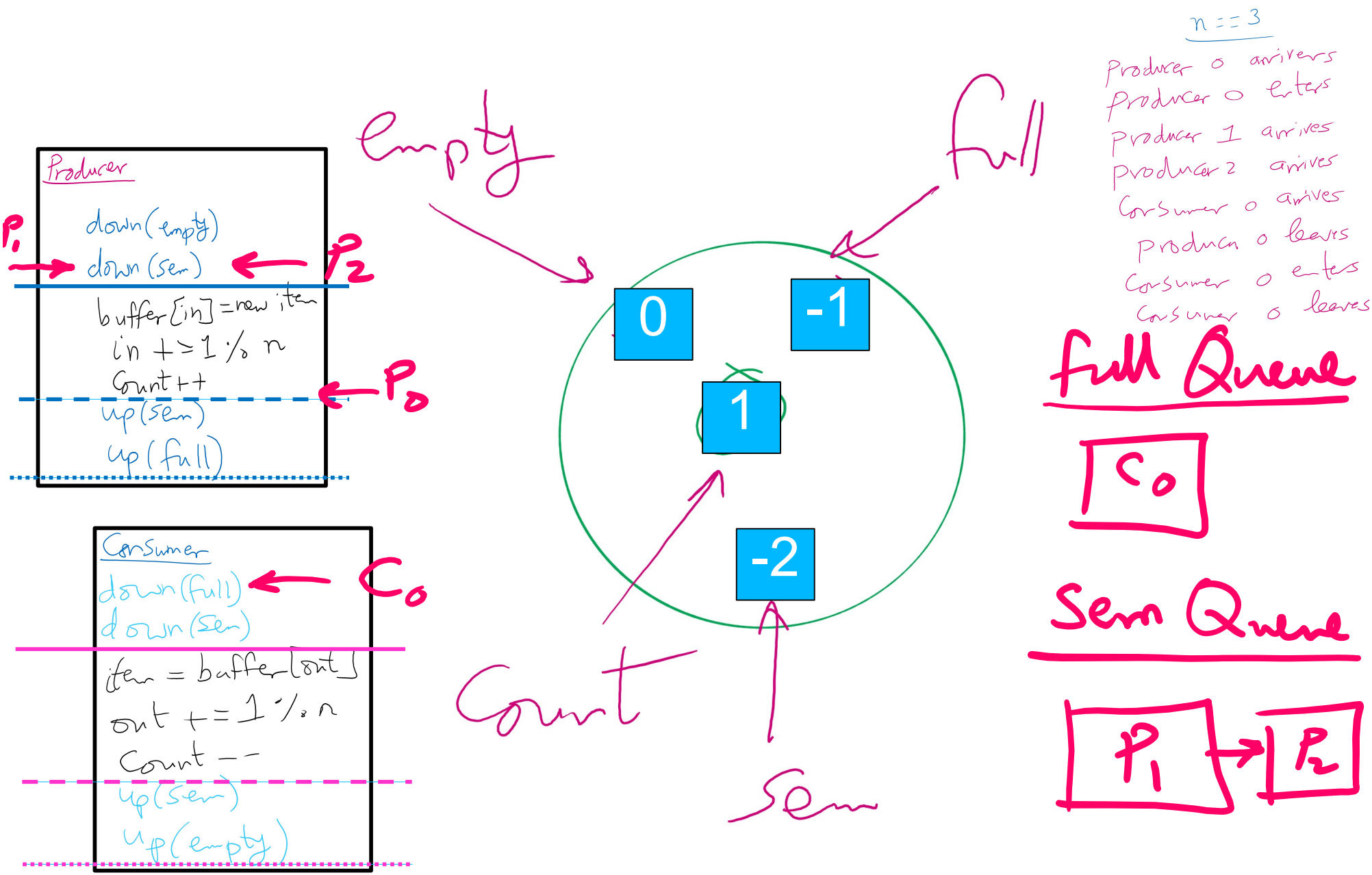
Sem Queue



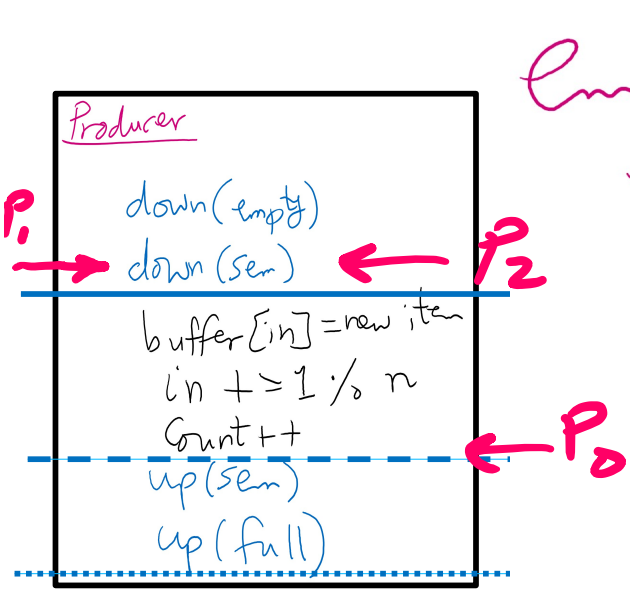
C0 arrives

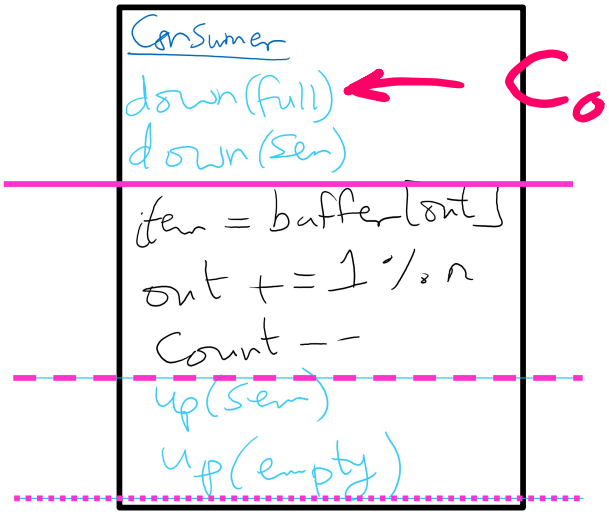

n == 3
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
Count++
up(sem)
up(full)
P1
P2
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
Count--
up(sem)
up(empty)
C0
empty
full
0
-1
1
-2
Count
Sem
full Queue
C0
Sem Queue
P1
P2

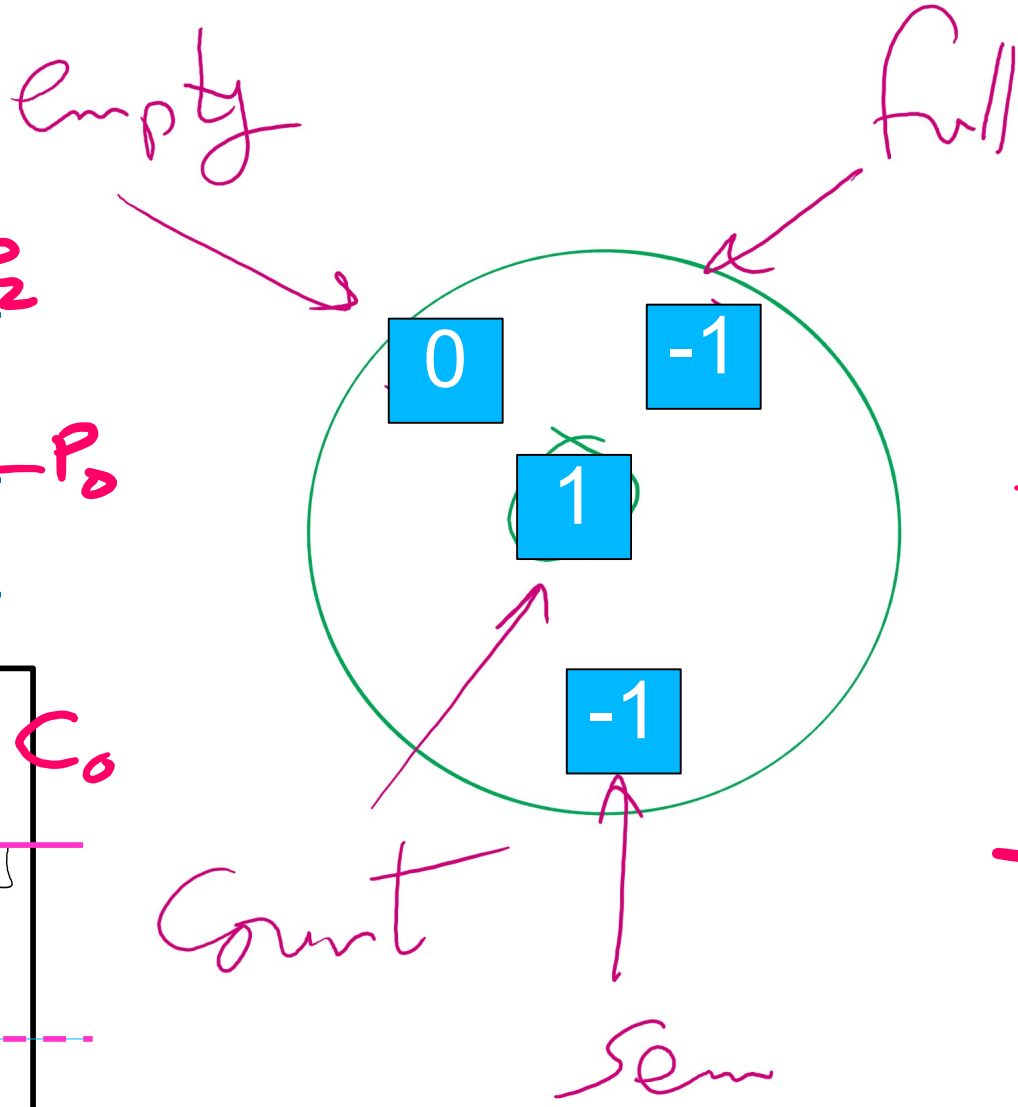
P0 leaves


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
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P1
P2
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
empty
full
0
-1
1
-2
count
sem
full Queue
C0
Sem Queue
P1
P2

P0 leaves





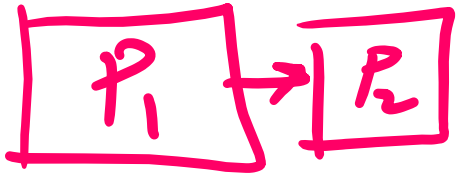


n == 3

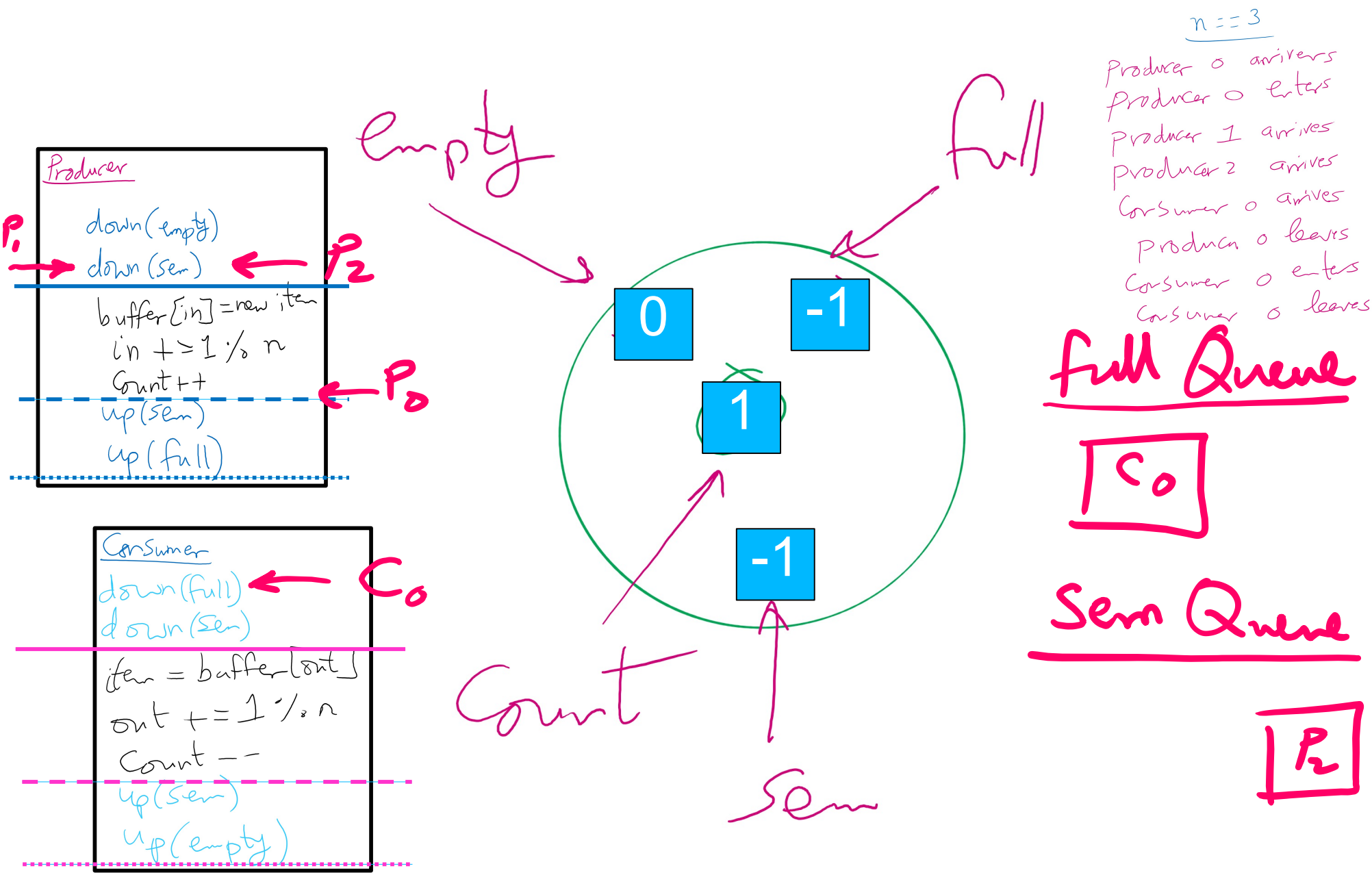
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

full Queue

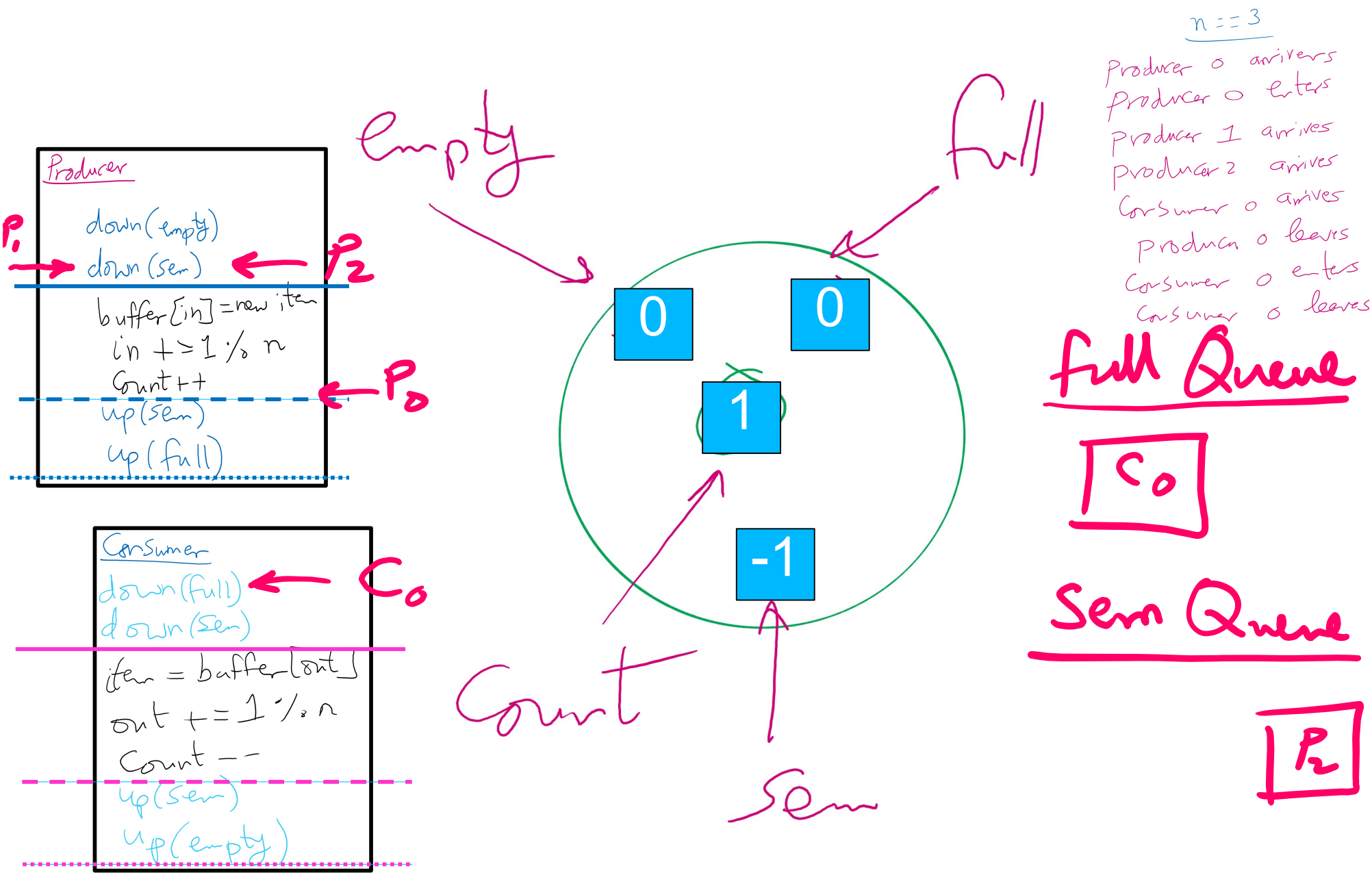
Sem Queue



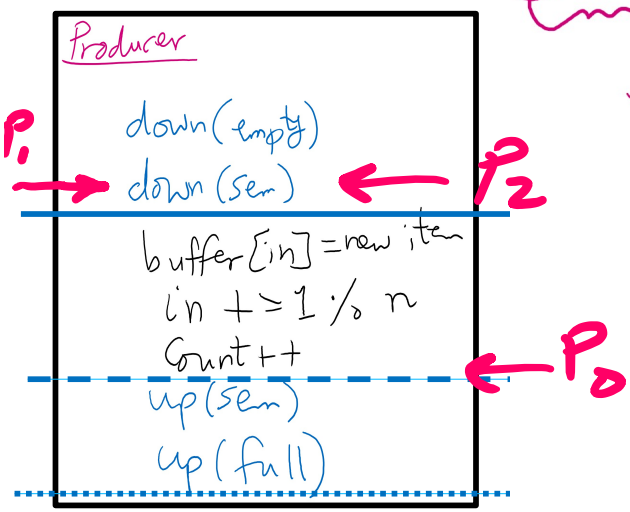
P0 leaves

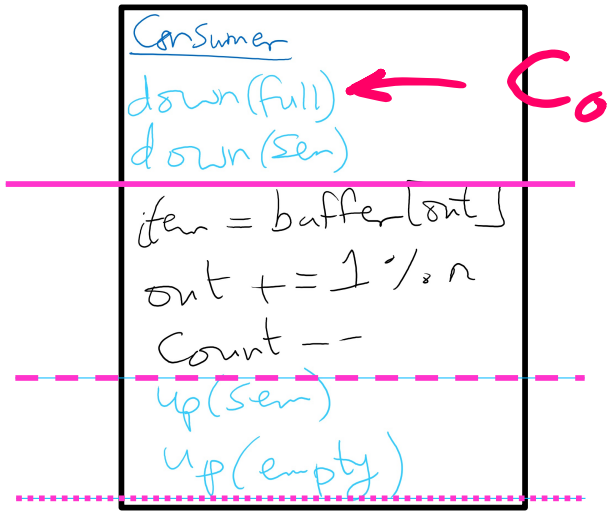

n == 3
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P1
P2
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
empty
full
0
-1
1
-1
count
sem
full Queue
C0
Sem Queue
P2

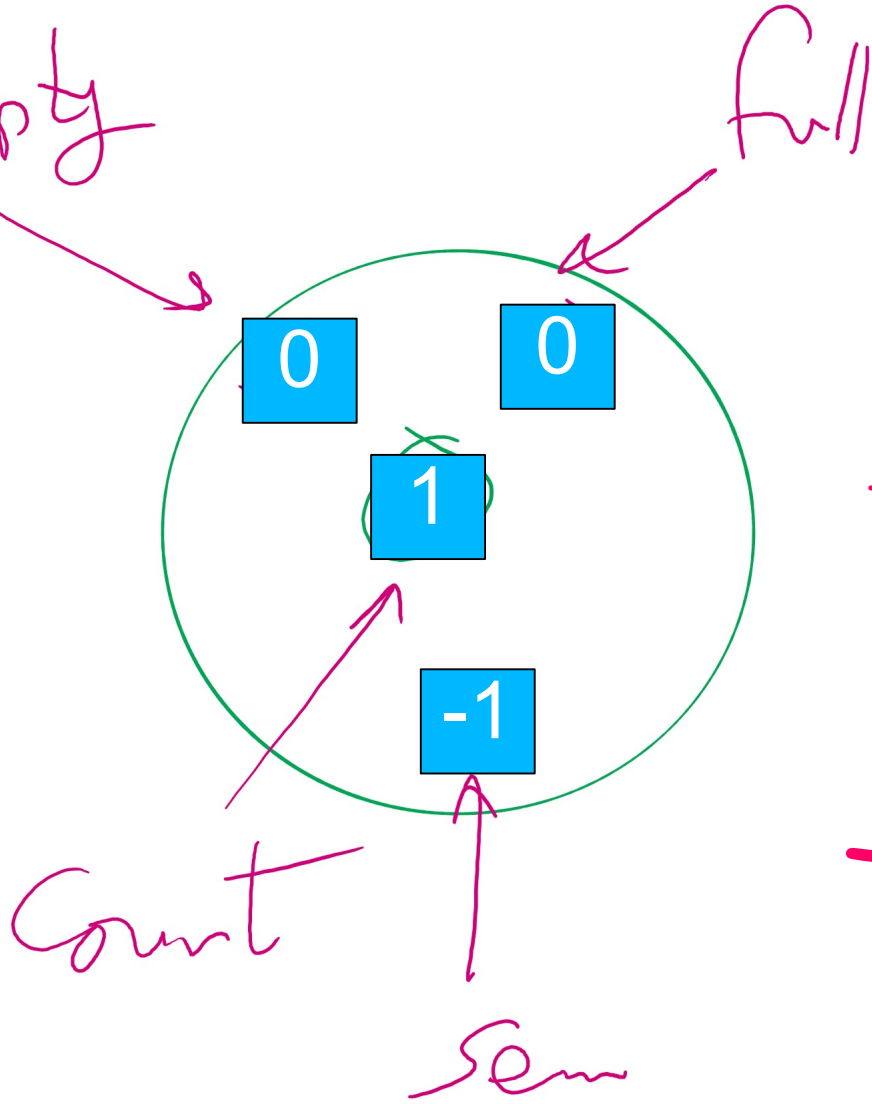
P0 leaves


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
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P1
P2
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
empty
full
0
0
1
-1
count
sem
full Queue
C0
Sem Queue
P2

C0 enters







n == 3

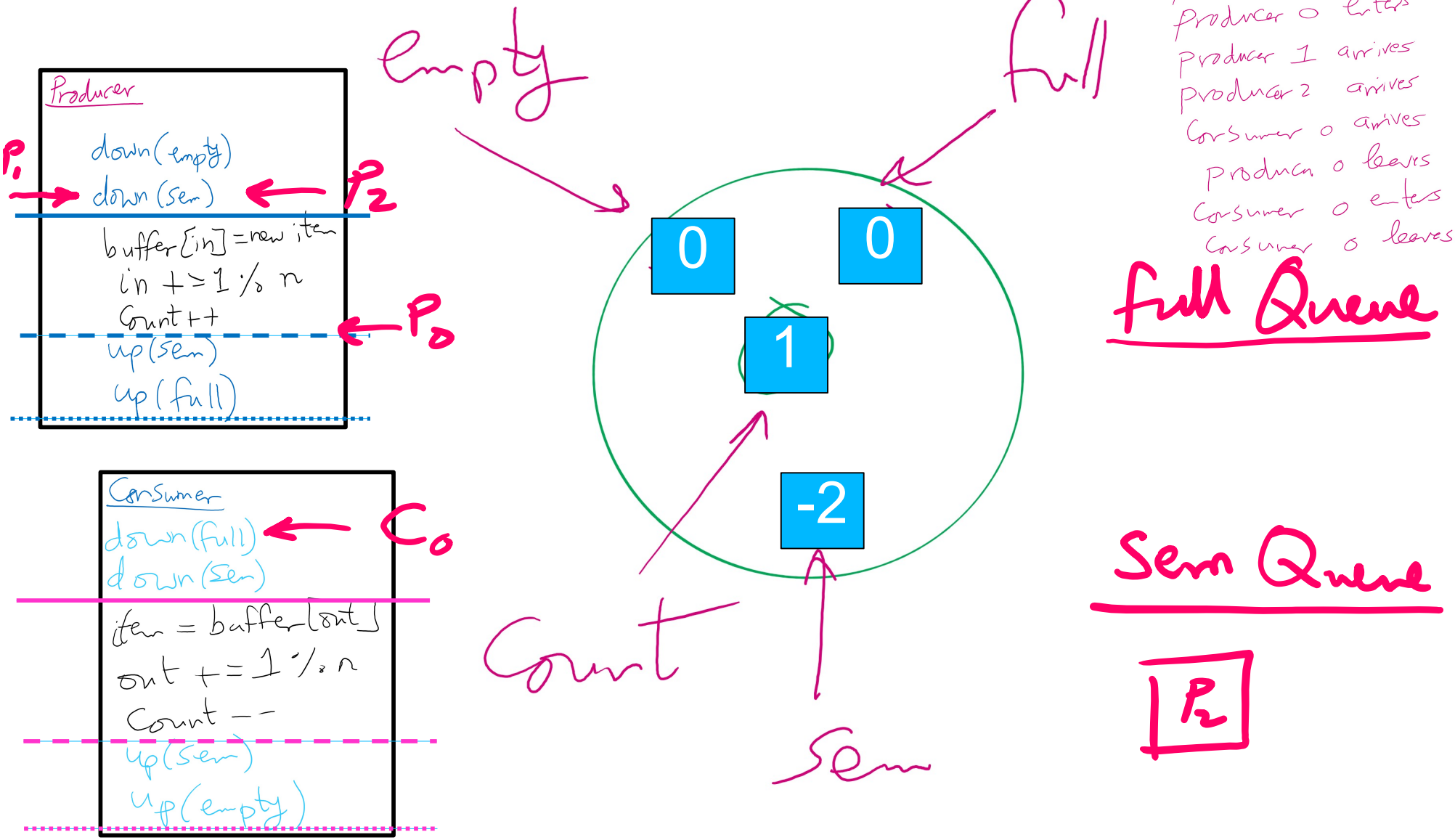
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

full Queue

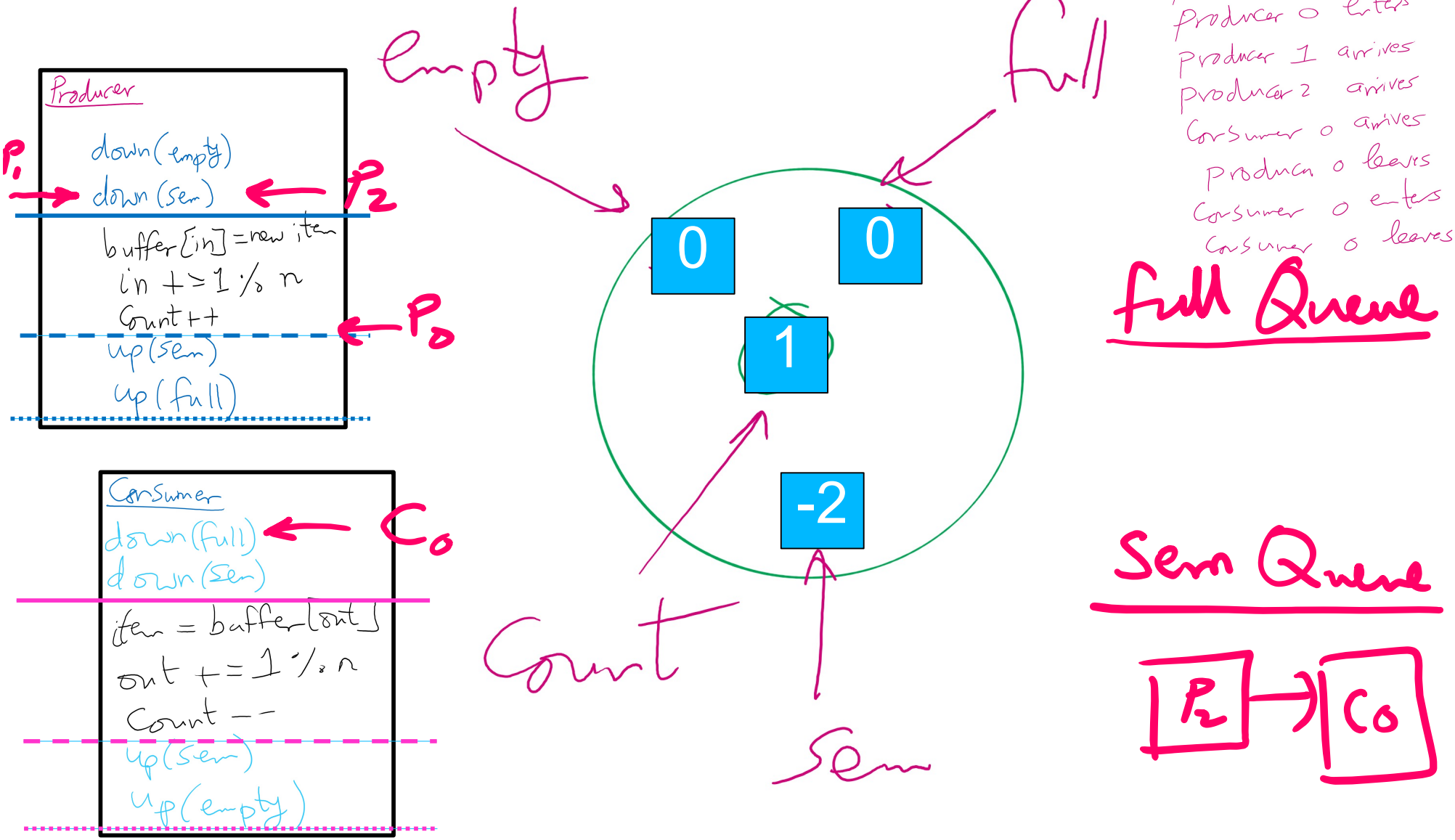
Sem Queue

P2

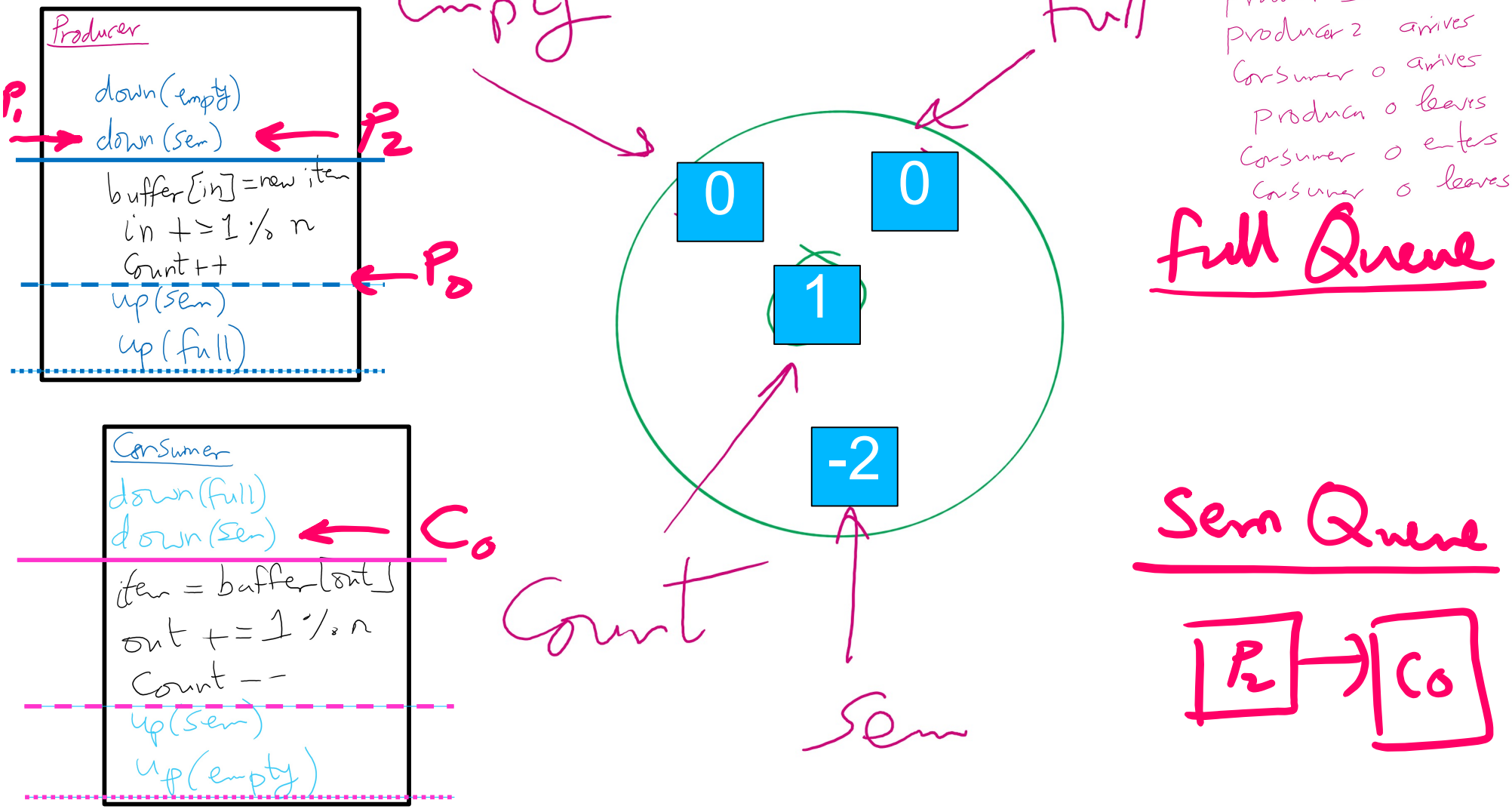
C0 enters


Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P1
P2
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
empty
full
0
0
1
-2
count
sem
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
P2

C0 enters


Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P1
P2
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
empty
full
0
0
1
-2
count
sem
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
P2
C0

Can C0 enter?



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

n == 1

Consumer 0 arrives

Producer 0 arrives

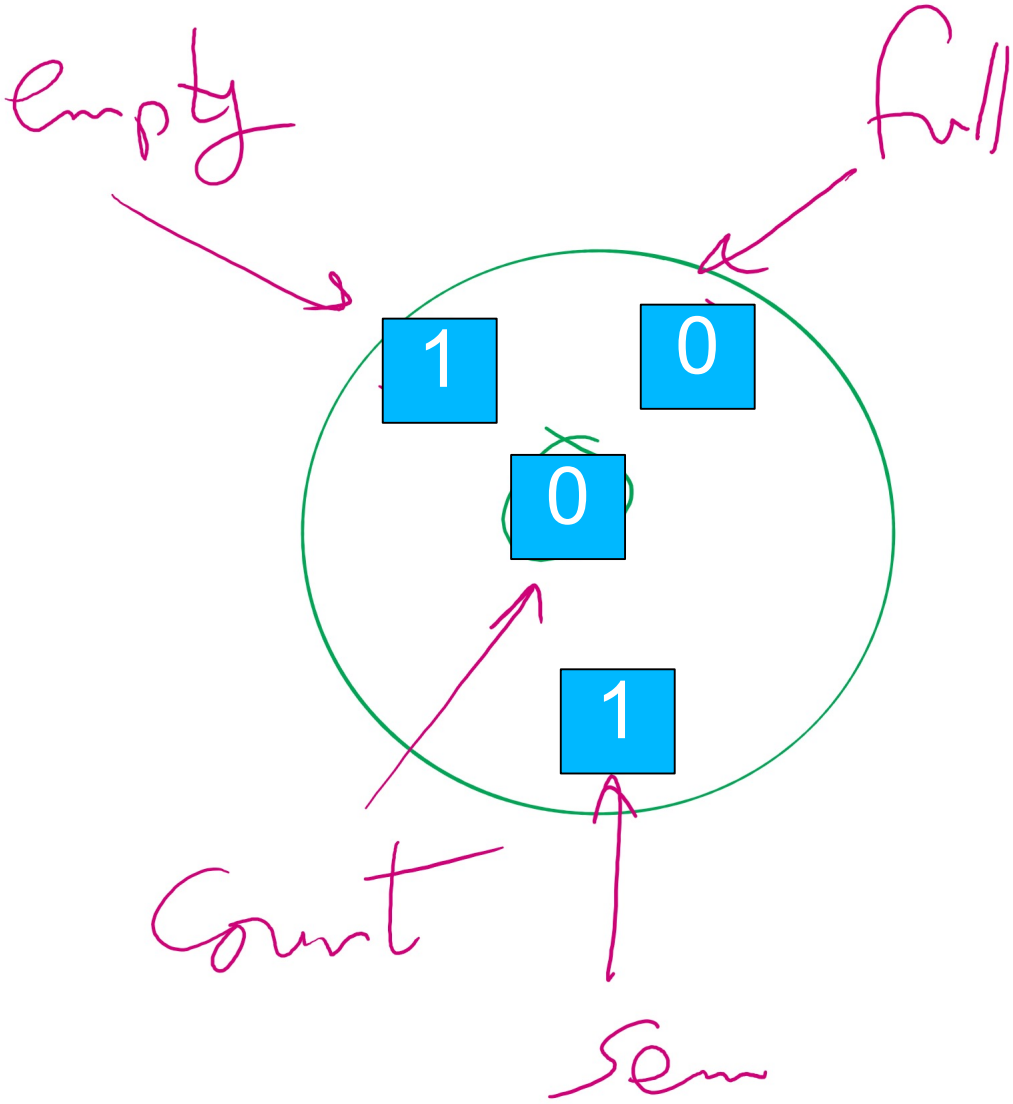
Producer 0 enters

Producer 0 leaves

Consumer 0 enters

Consumer 0 leaves

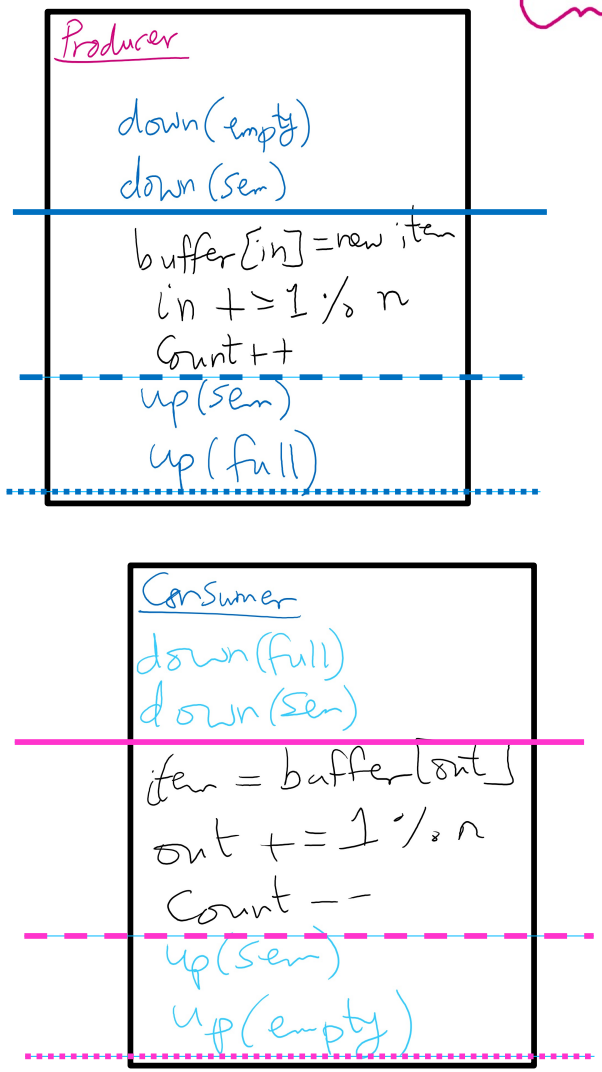
Initial state

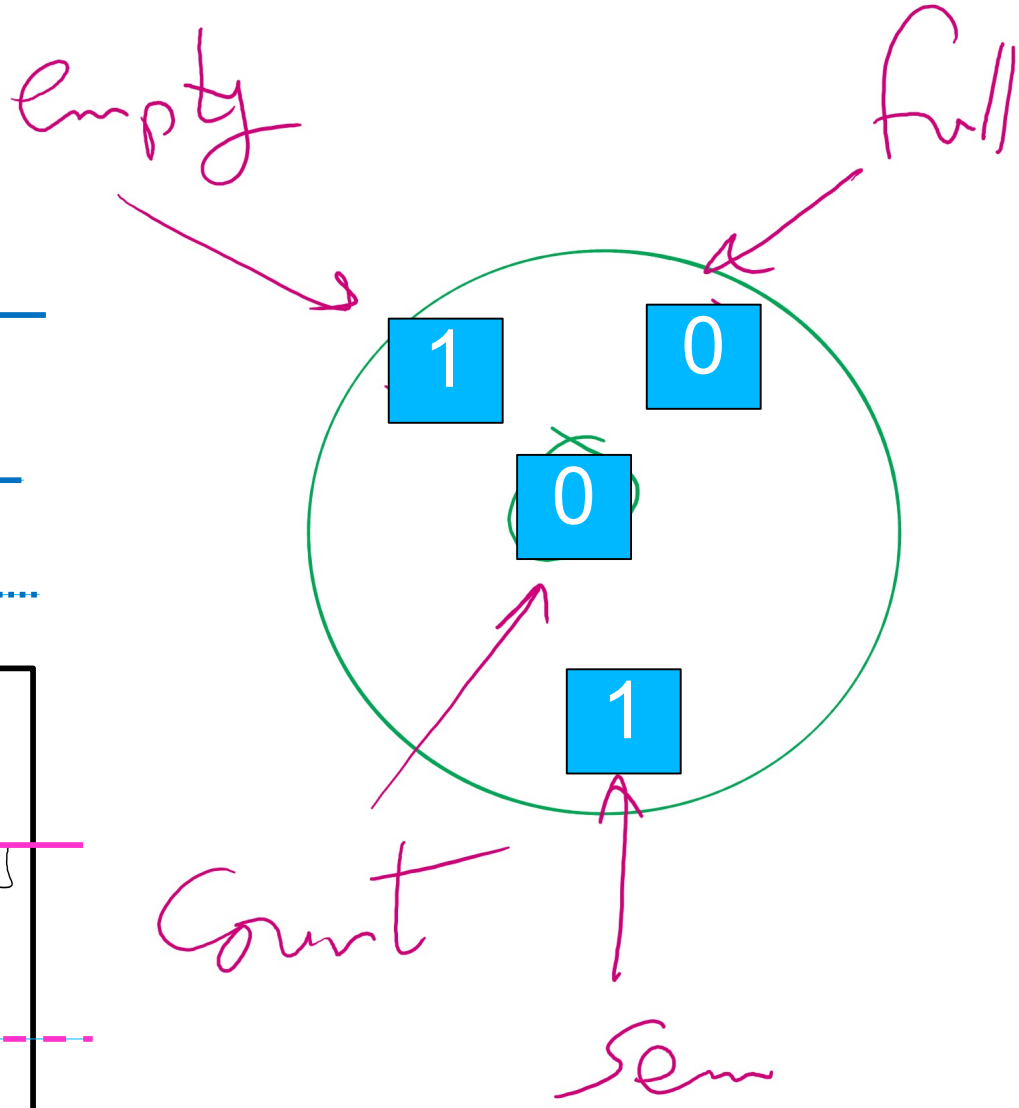


n = = 1

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

C0 arrives

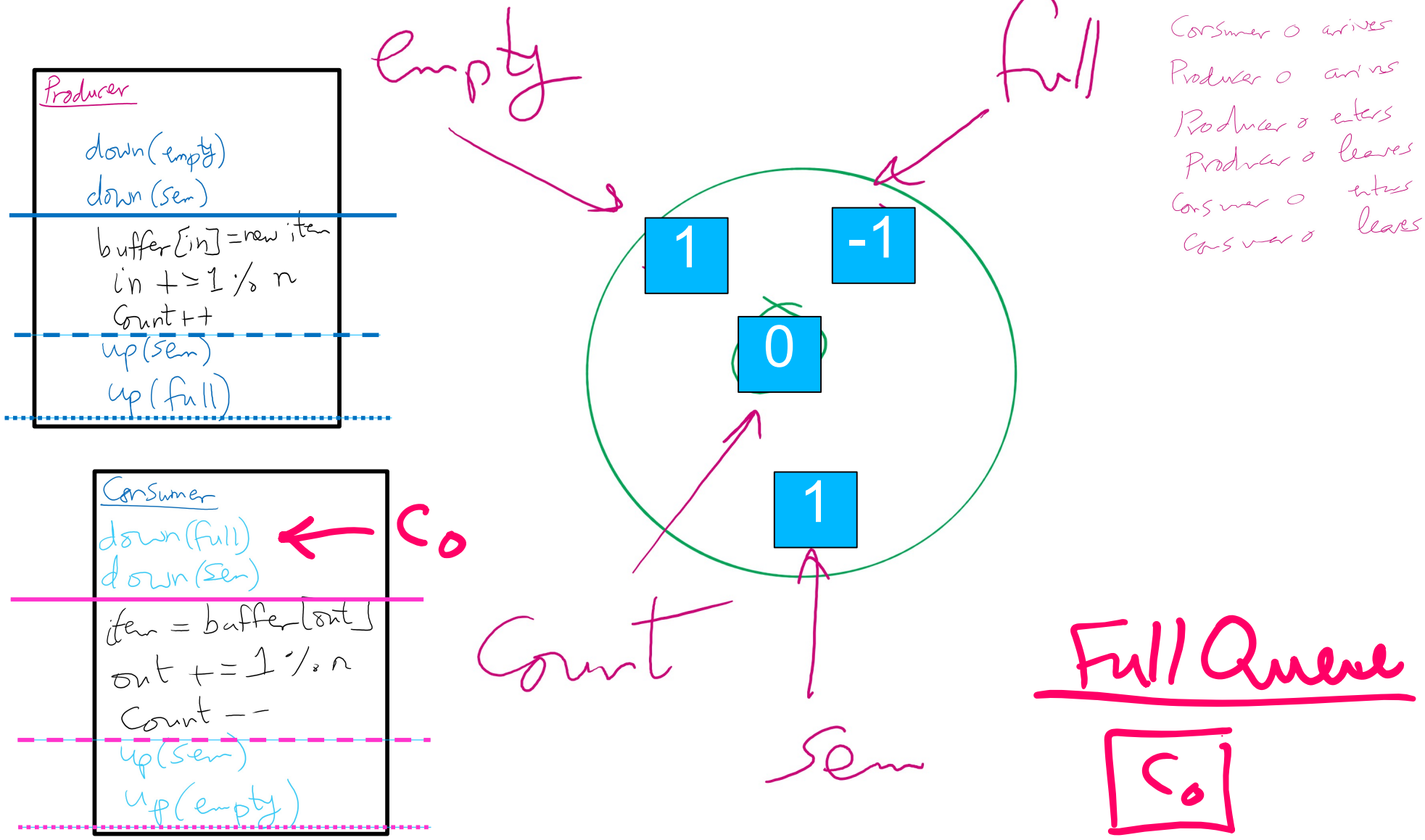




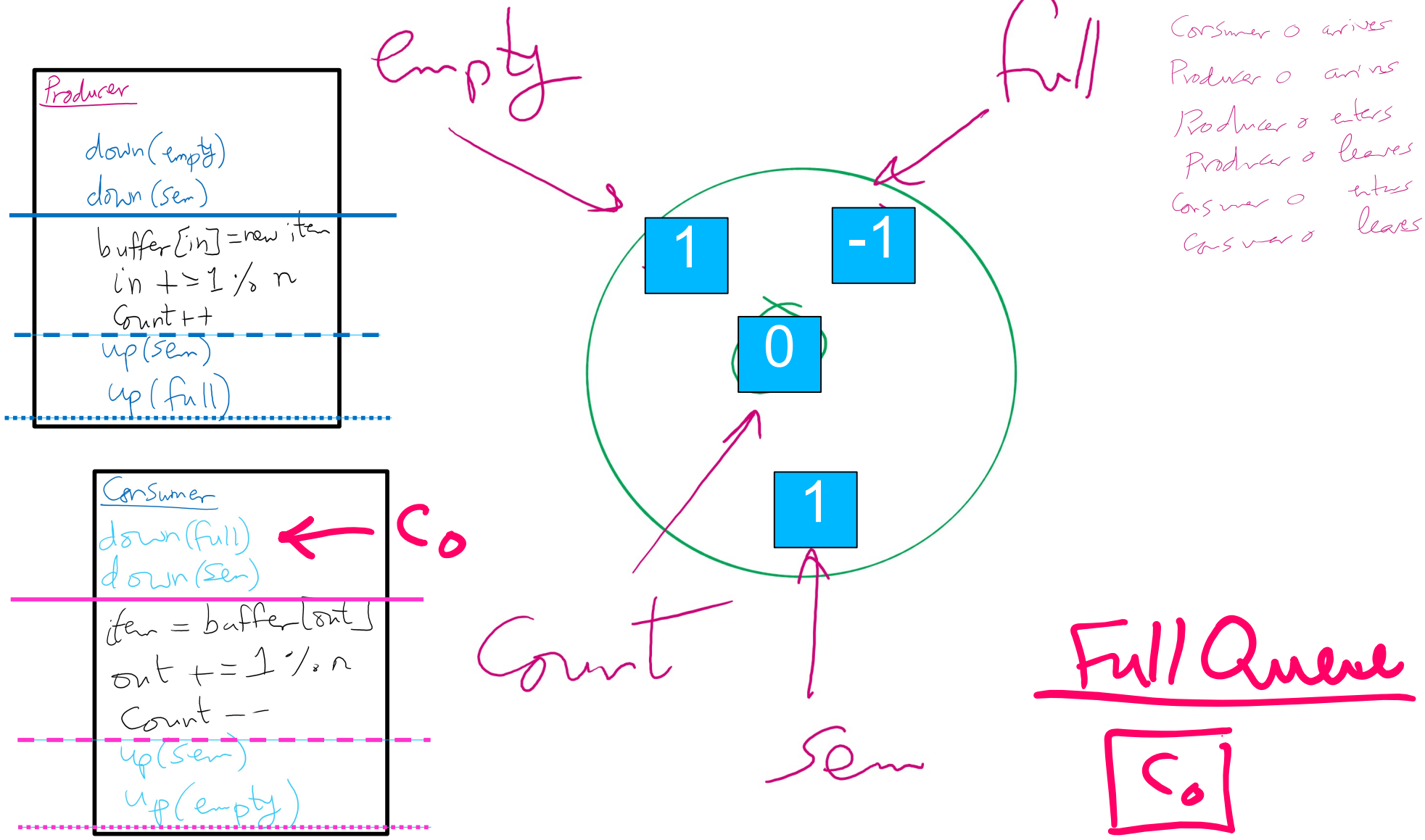
n == 1

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

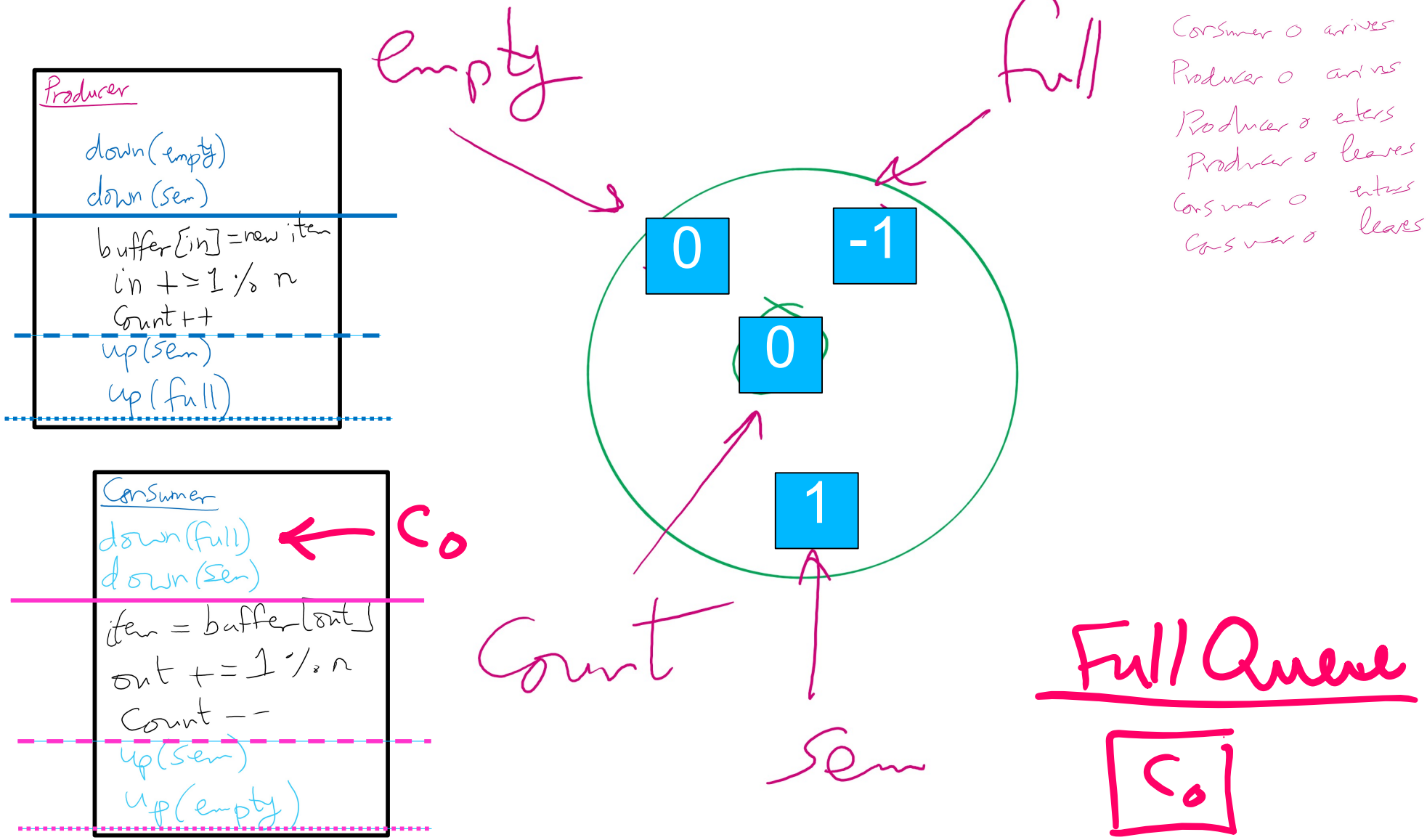
C0 arrives


n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
empty
full
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)
C0
1
-1
0
1
count
sem
Full Queue
C0

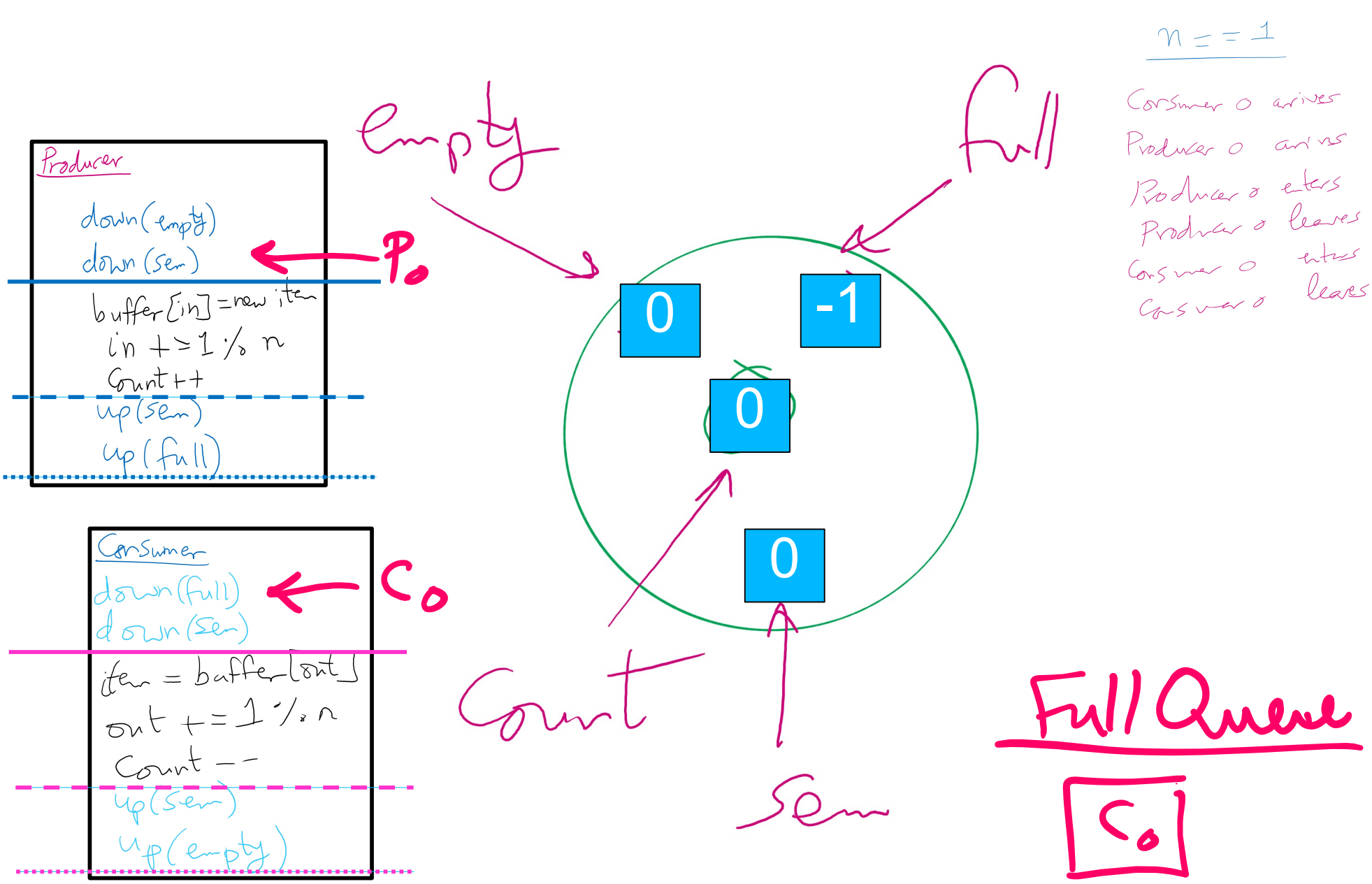
P0 arrives


n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
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)
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
empty
full
1
-1
0
1
count
sem
Full Queue
C0

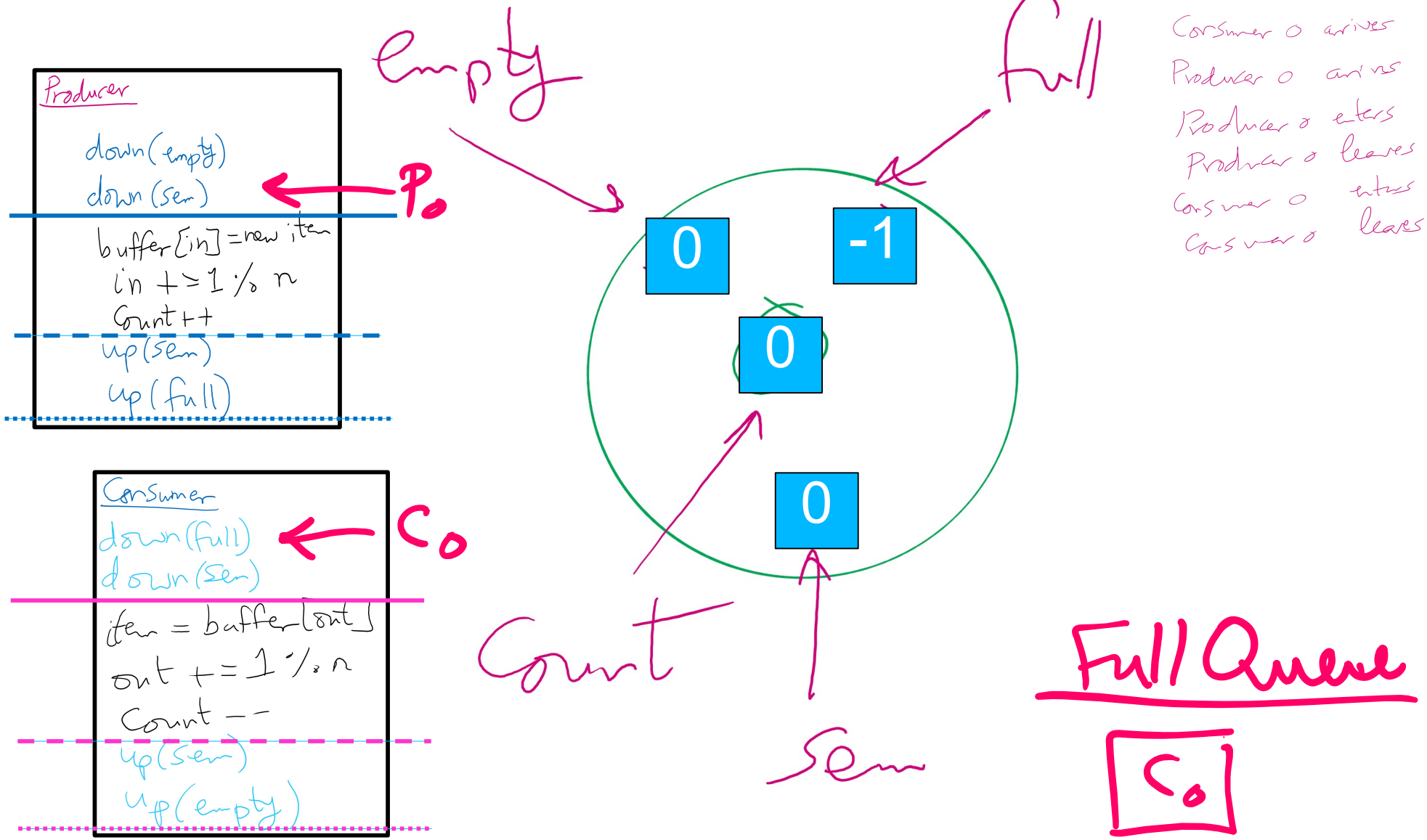
P0 arrives


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)
C0
empty
full
0
-1
0
1
Count
sem
n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
Full Queue
C0

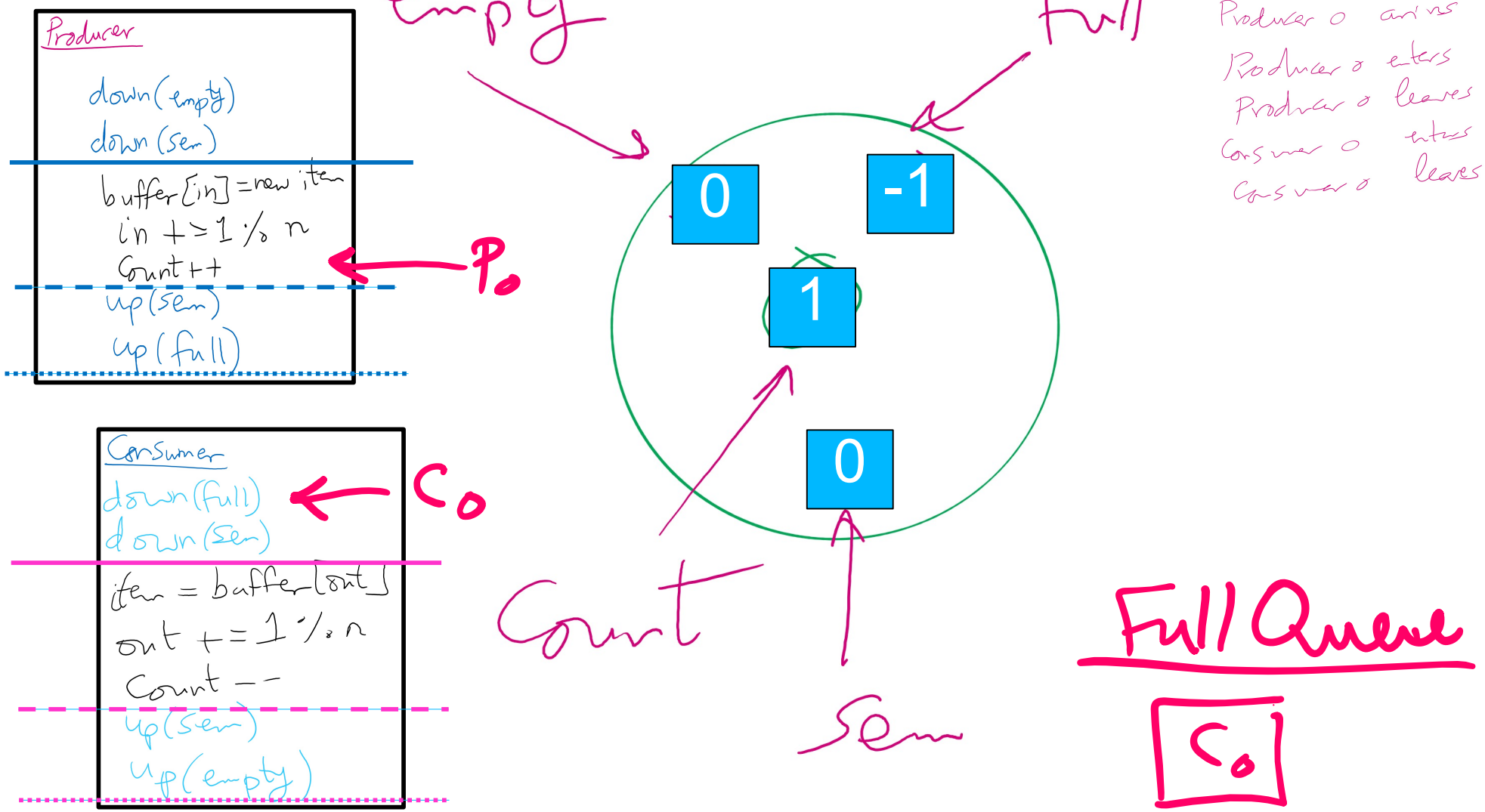
P0 arrives


n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
empty
full
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
0
-1
0
0
count
sem
Full Queue
C0

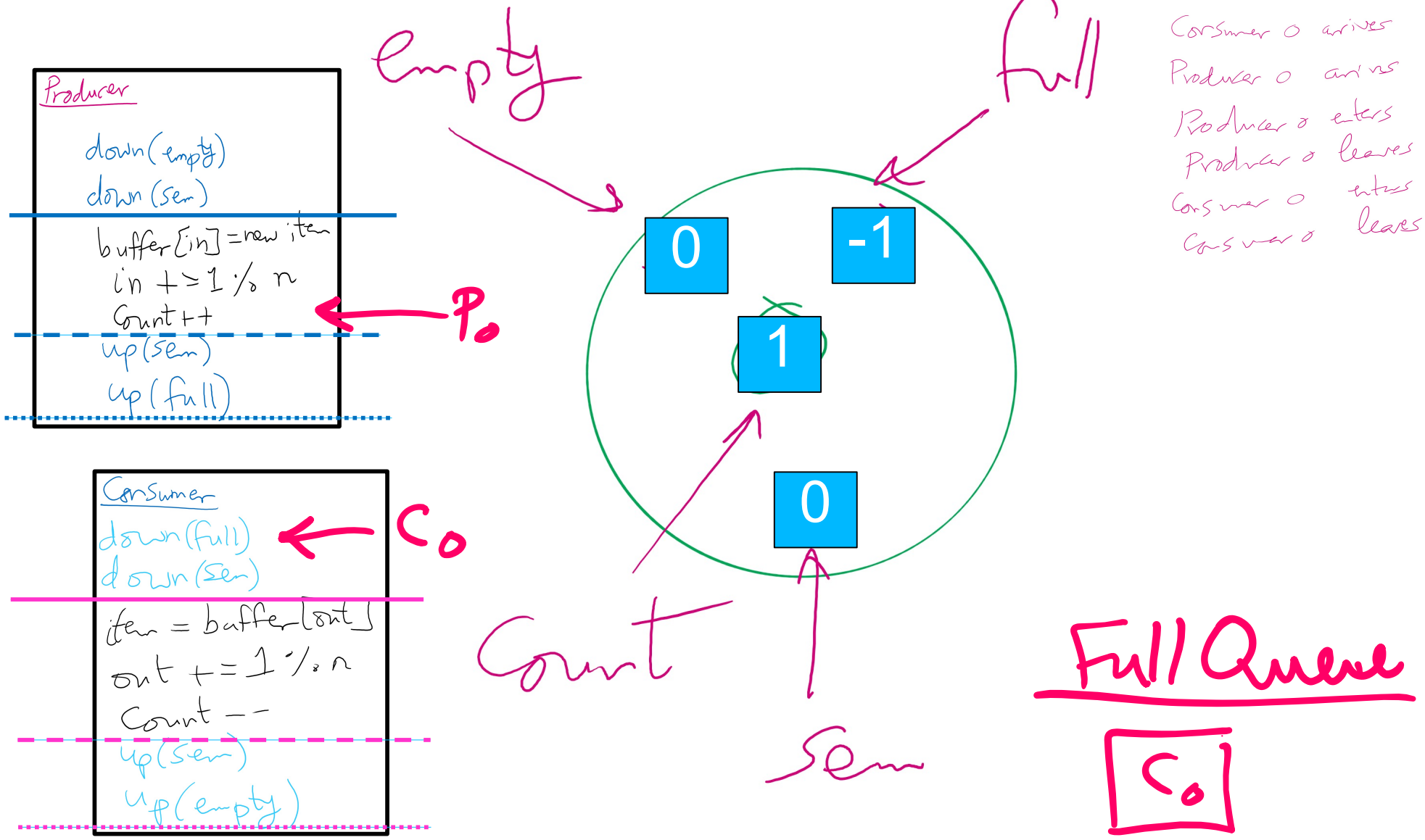
P0 enters


Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
empty
full
0
-1
0
0
count
sem
n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
Full Queue
C0

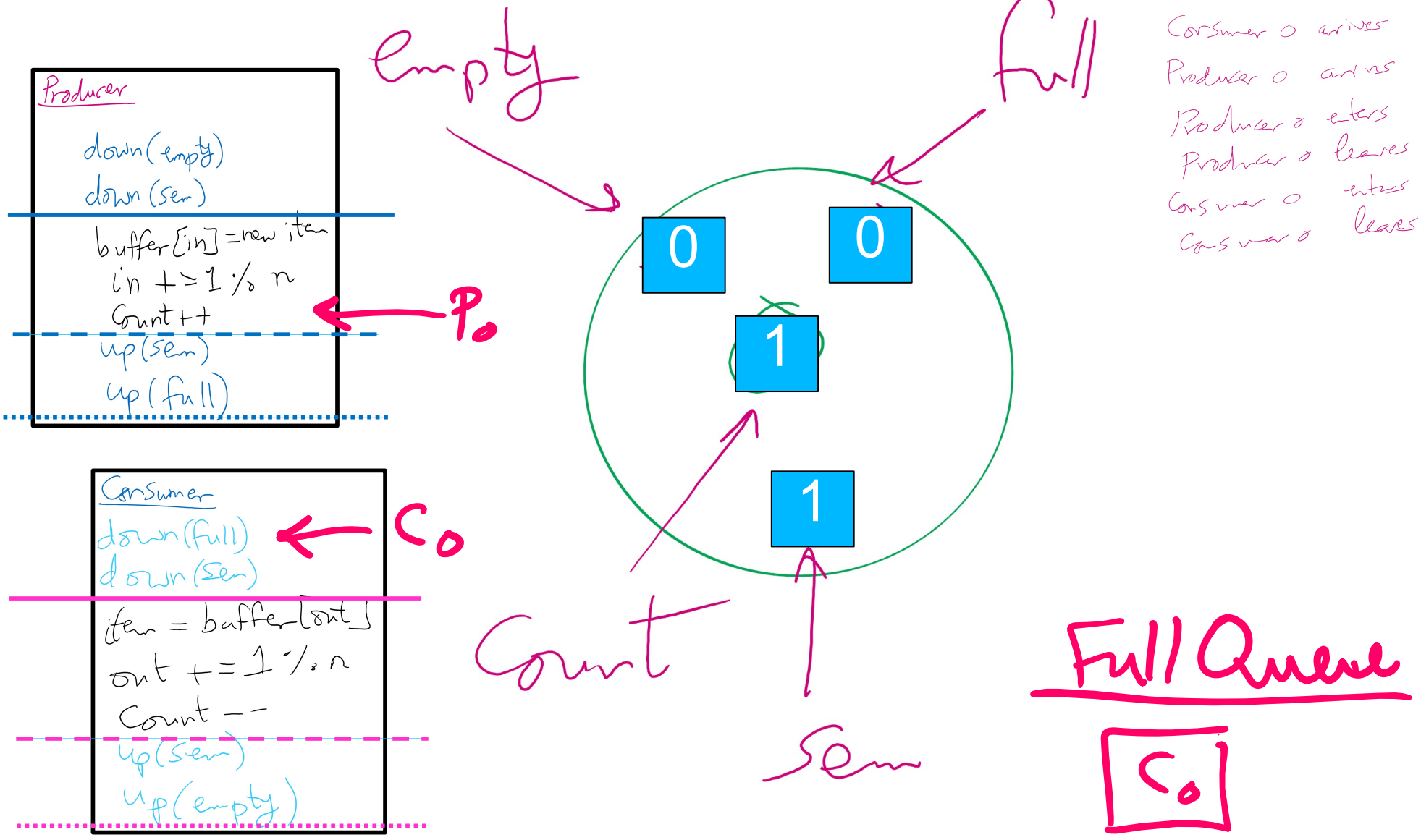
P0 enters


Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
Count++
up(sem)
up(full)
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
Count--
up(sem)
up(empty)
C0
empty
full
0
-1
1
0
Count
sem
n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
Full Queue
C0

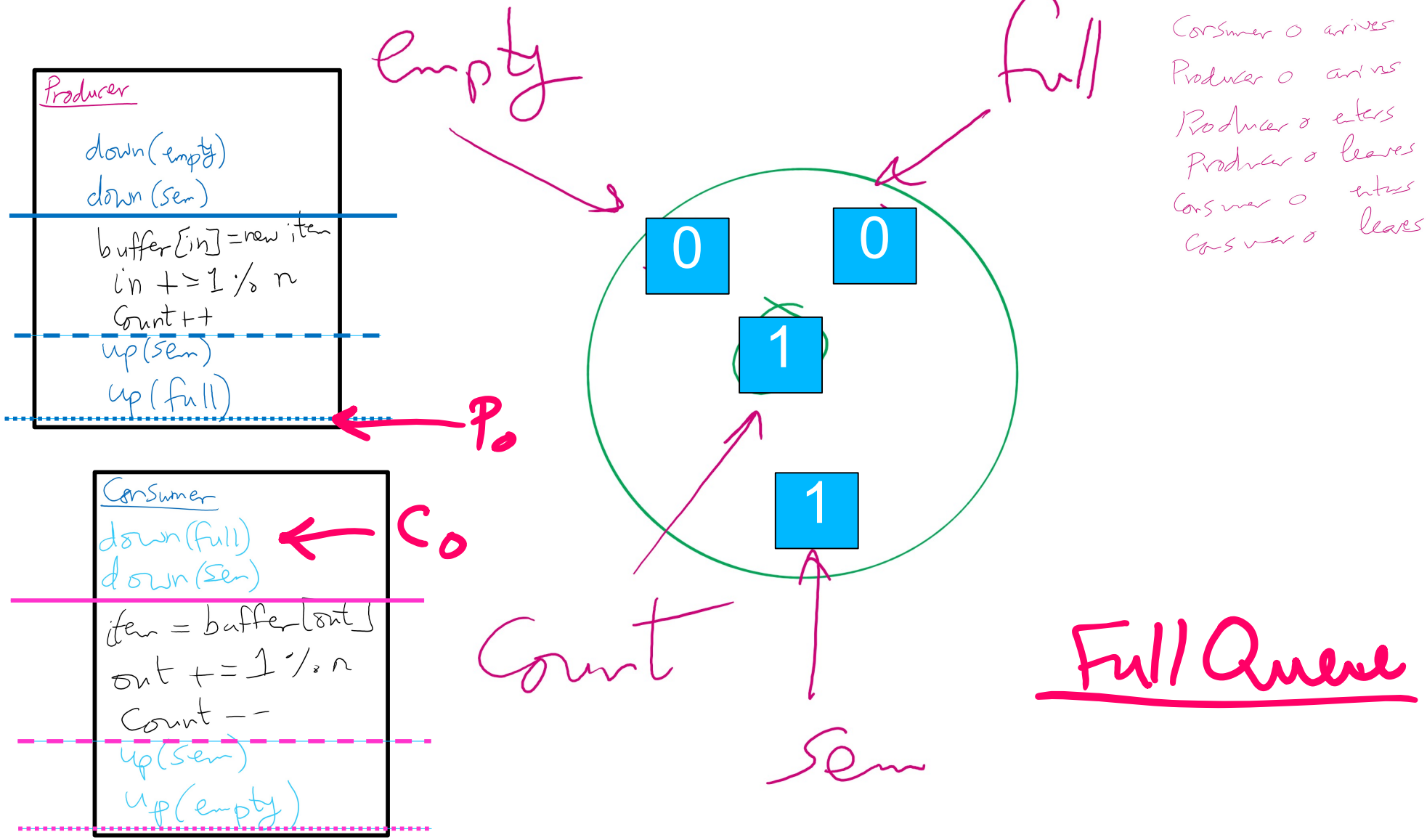
P0 leaves


Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
empty
full
0
-1
1
0
count
sem
n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
Full Queue
C0

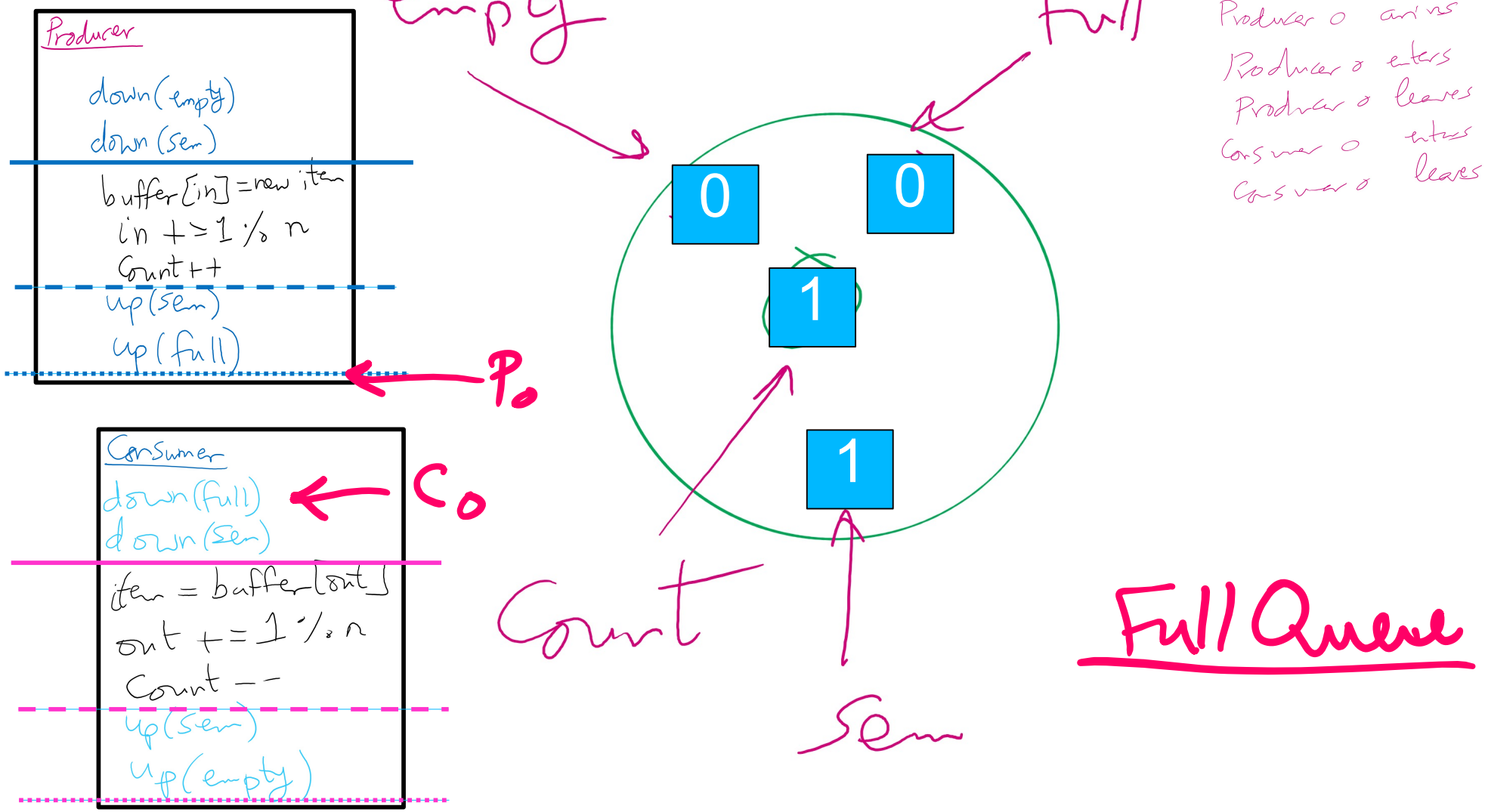
P0 leaves


n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
empty
full
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
0
0
1
1
count
sem
Full Queue
C0

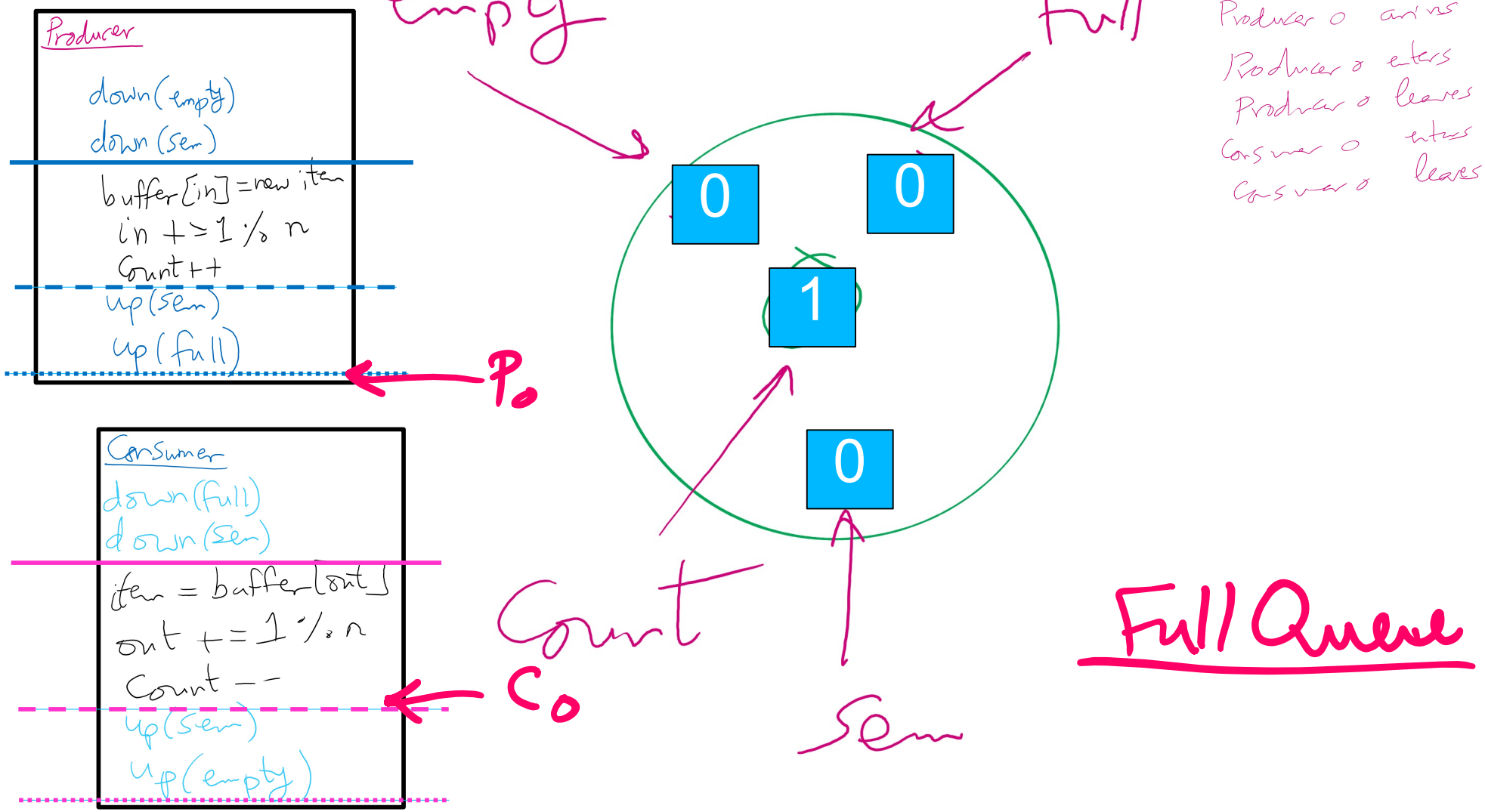
P0 leaves


n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
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
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
empty
full
0
0
1
1
count
sem
Full Queue

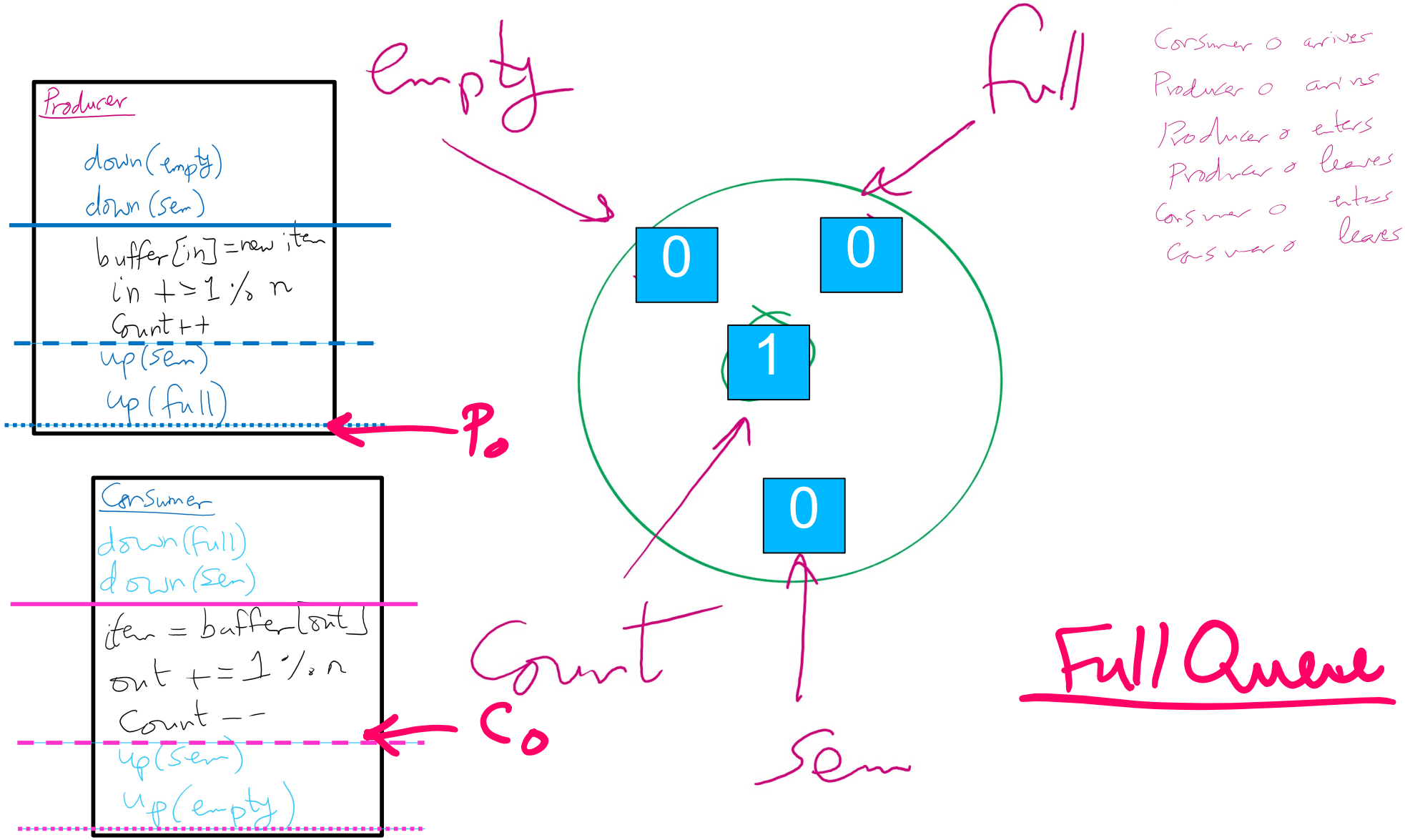
C0 enters


n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
empty
full
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
0
0
1
1
count
sem
Full Queue

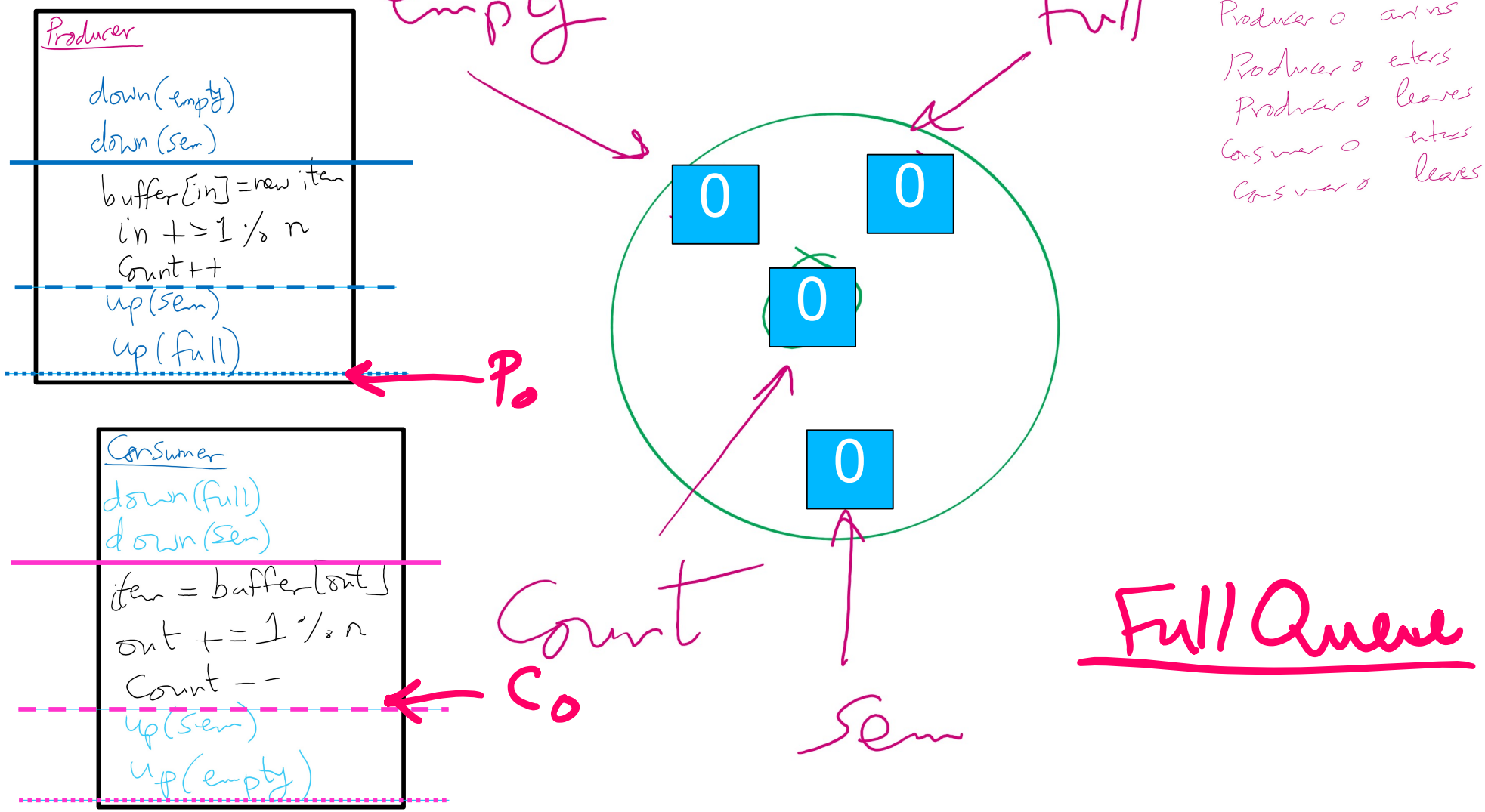
C0 enters


Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
empty
full
0
0
1
0
count
sem
n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
Full Queue

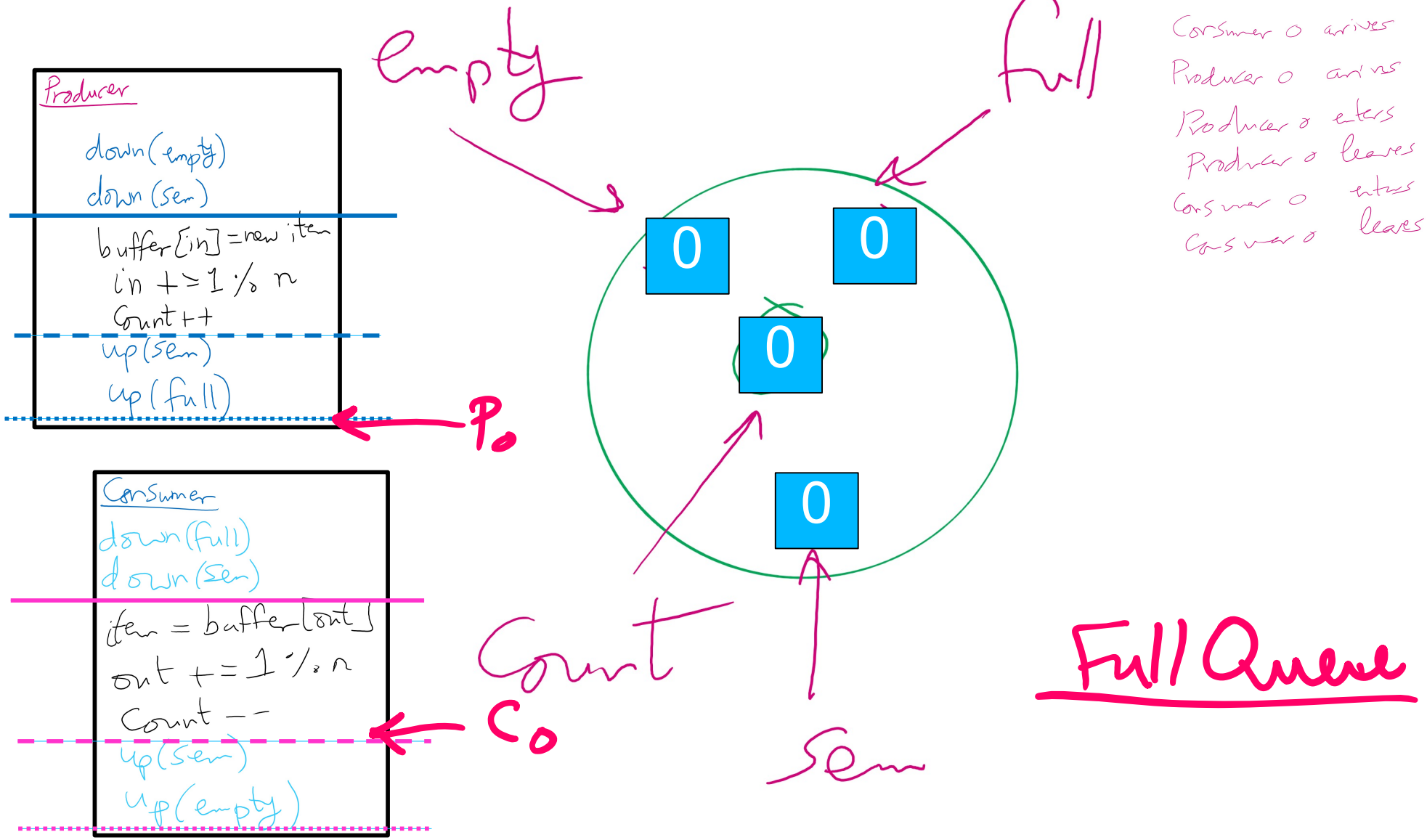
Can C0 enter?


n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
empty
full
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
0
0
1
0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
count
sem
Full Queue

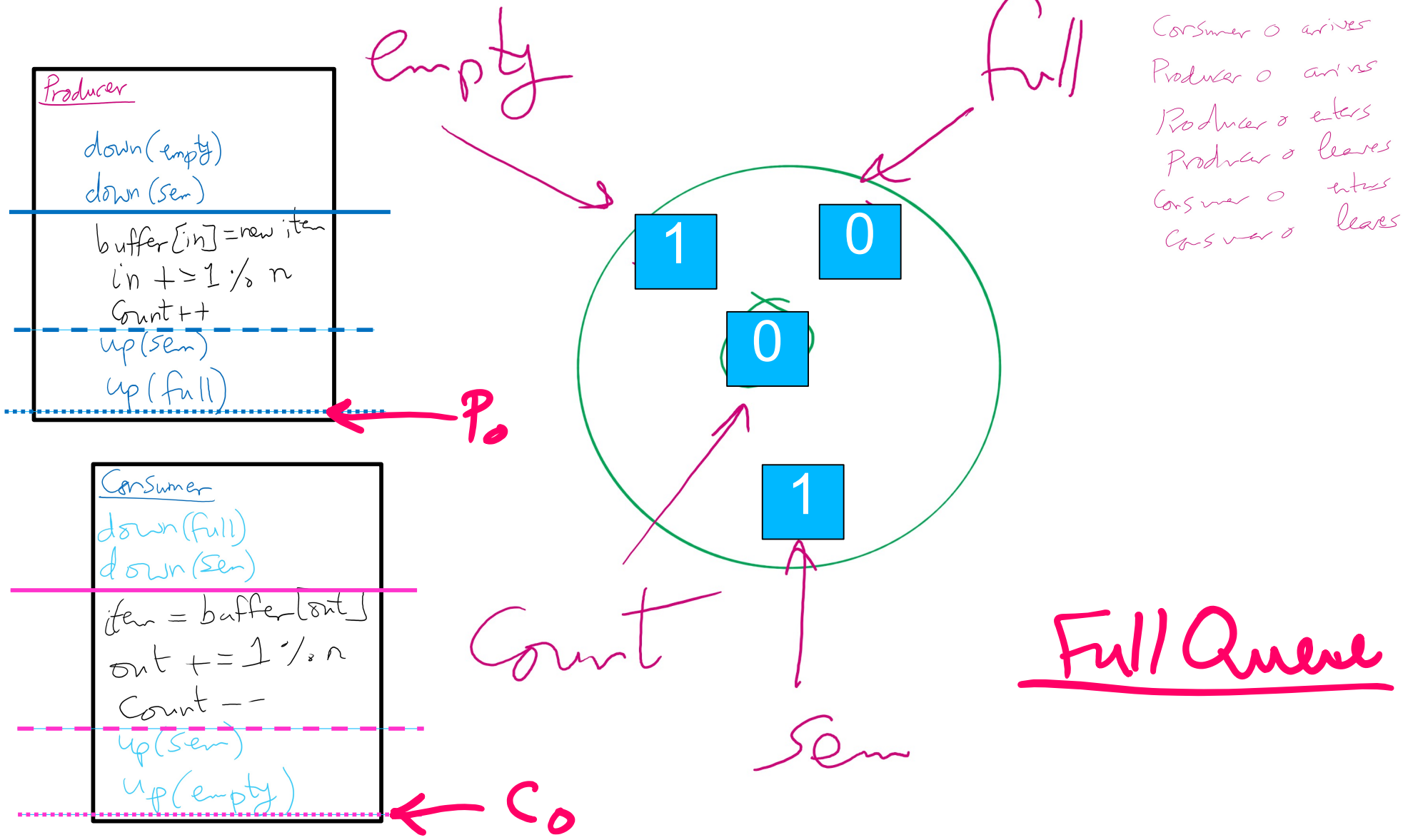
C0 enters


Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
empty
full
0
0
0
0
count
sem
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
empty
full
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0
0
0
0
0
count
sem
Full Queue

C0 leaves


Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
Count++
up(sem)
up(full)
P0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
Count--
up(sem)
up(empty)
C0
empty
full
1
0
0
1
Count
Sem
n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
Full Queue

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?

Let's make a "small" change

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

Solving Bounded Buffer Using Condition Variables

```
Mutex sem;
ConditionVariable CV;
```

Producer

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

Consumer

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