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

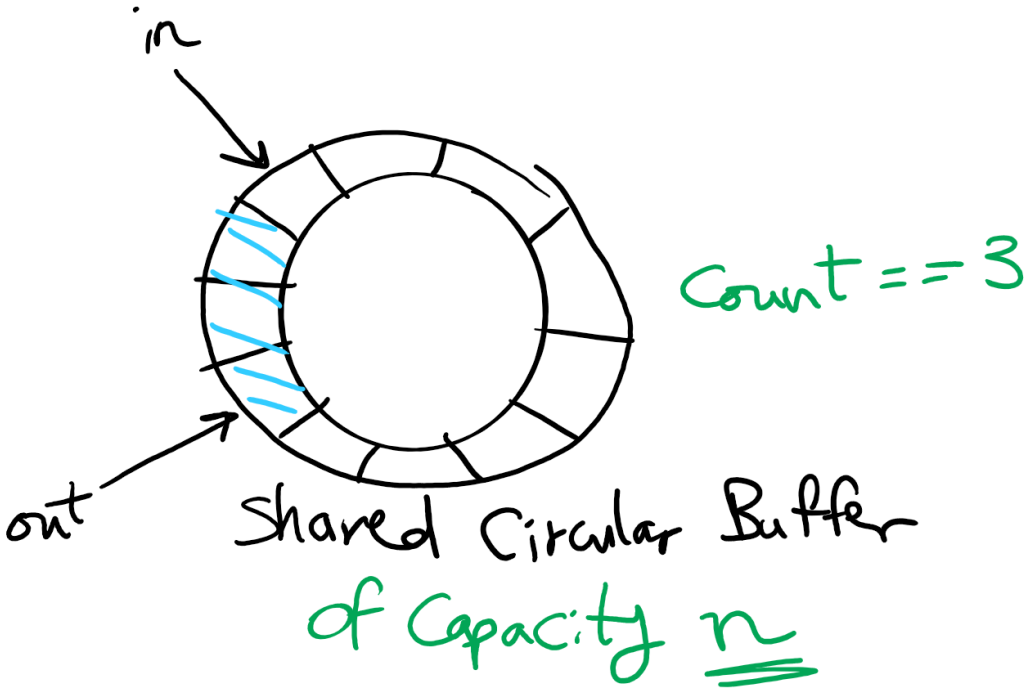
Previous lecture …

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

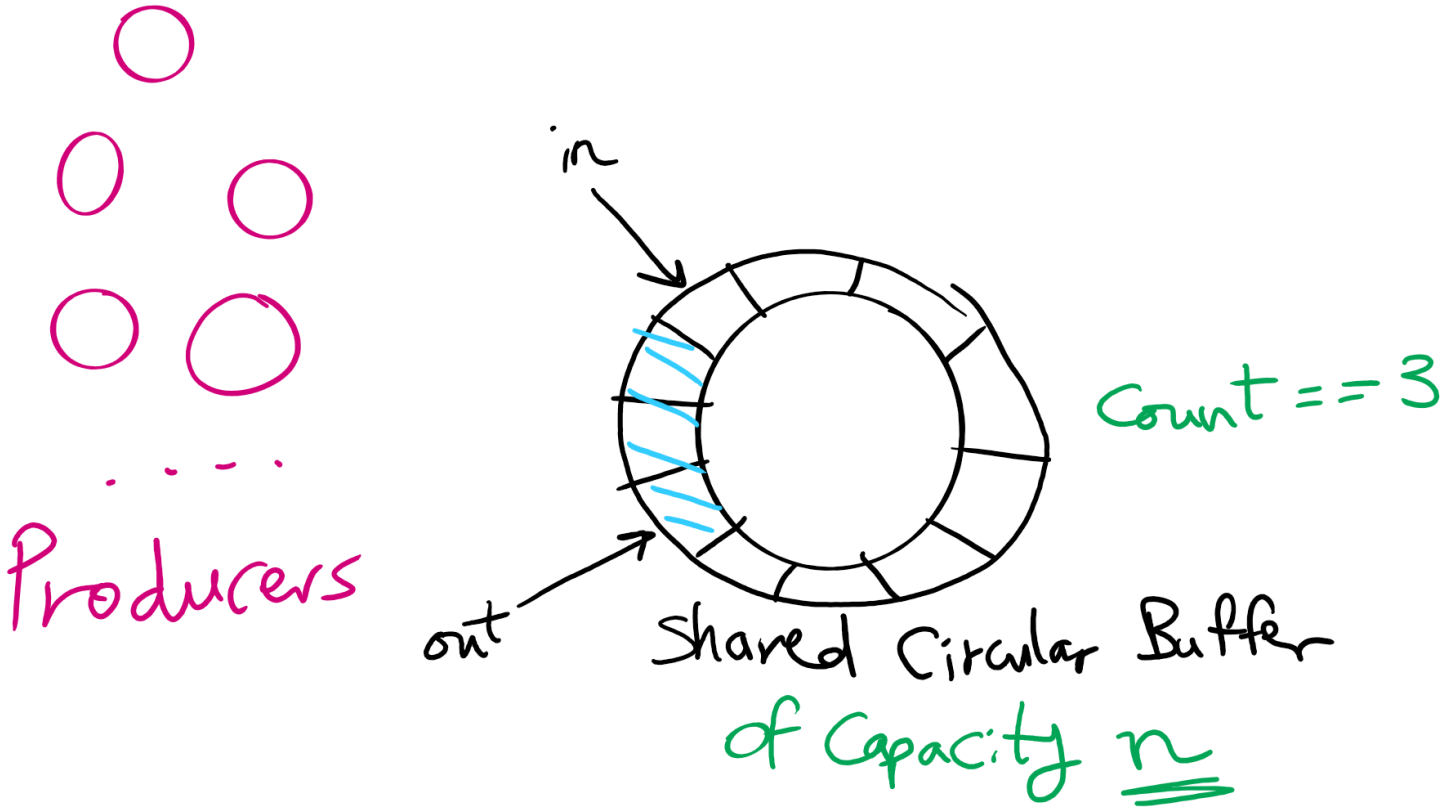
Problem of the Day

- 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

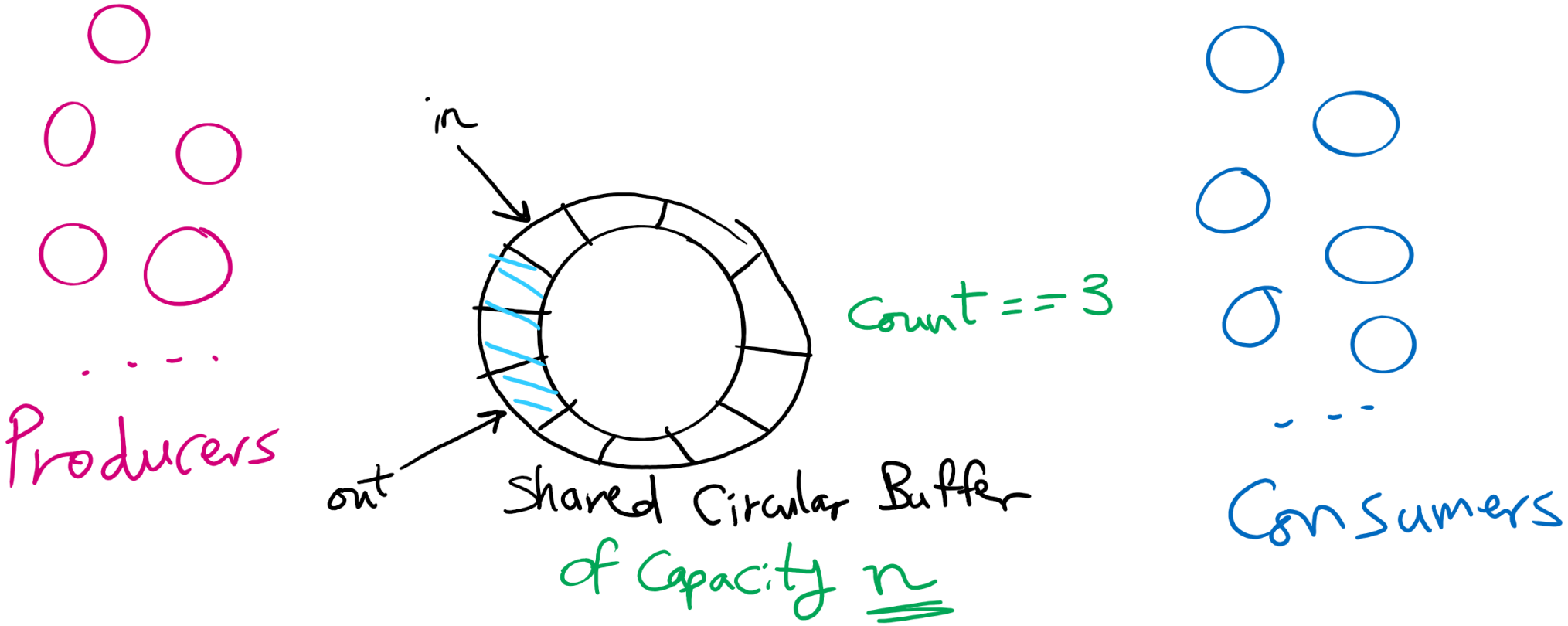
Produces Consumers Problem



Produces Consumers Problem



Produces Consumers Problem

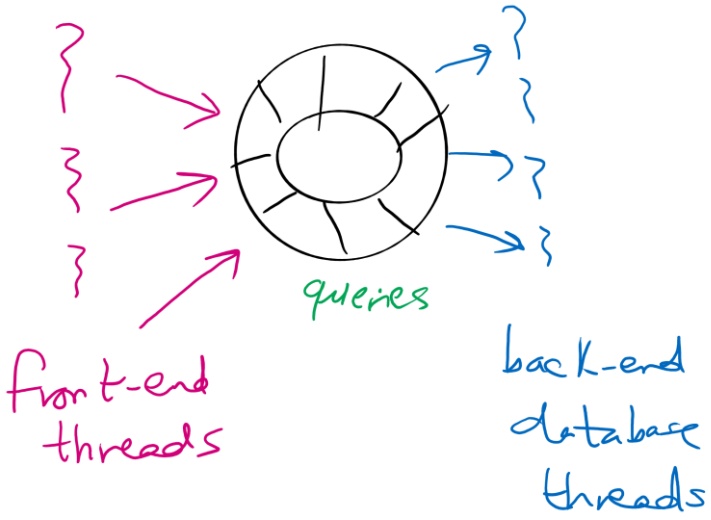


Producers Consumers Problem is everywhere!

- 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);

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)
```

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

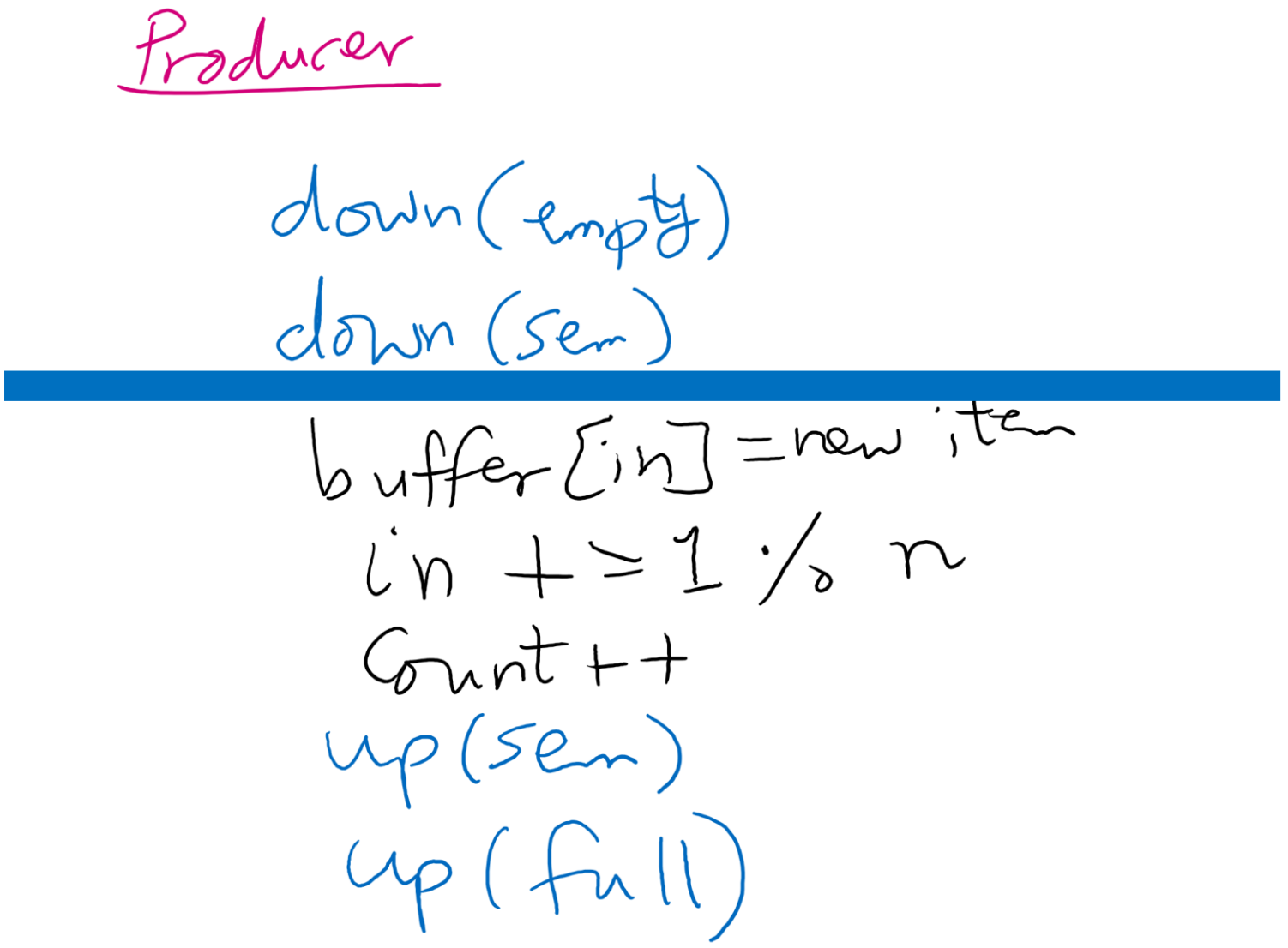
```
down(full)
down(sem)
item = buffer[out]
out += 1 % n
Count--
up(sem)
up(empty)
```

Let’s trace the solution!

Let’s define some events first

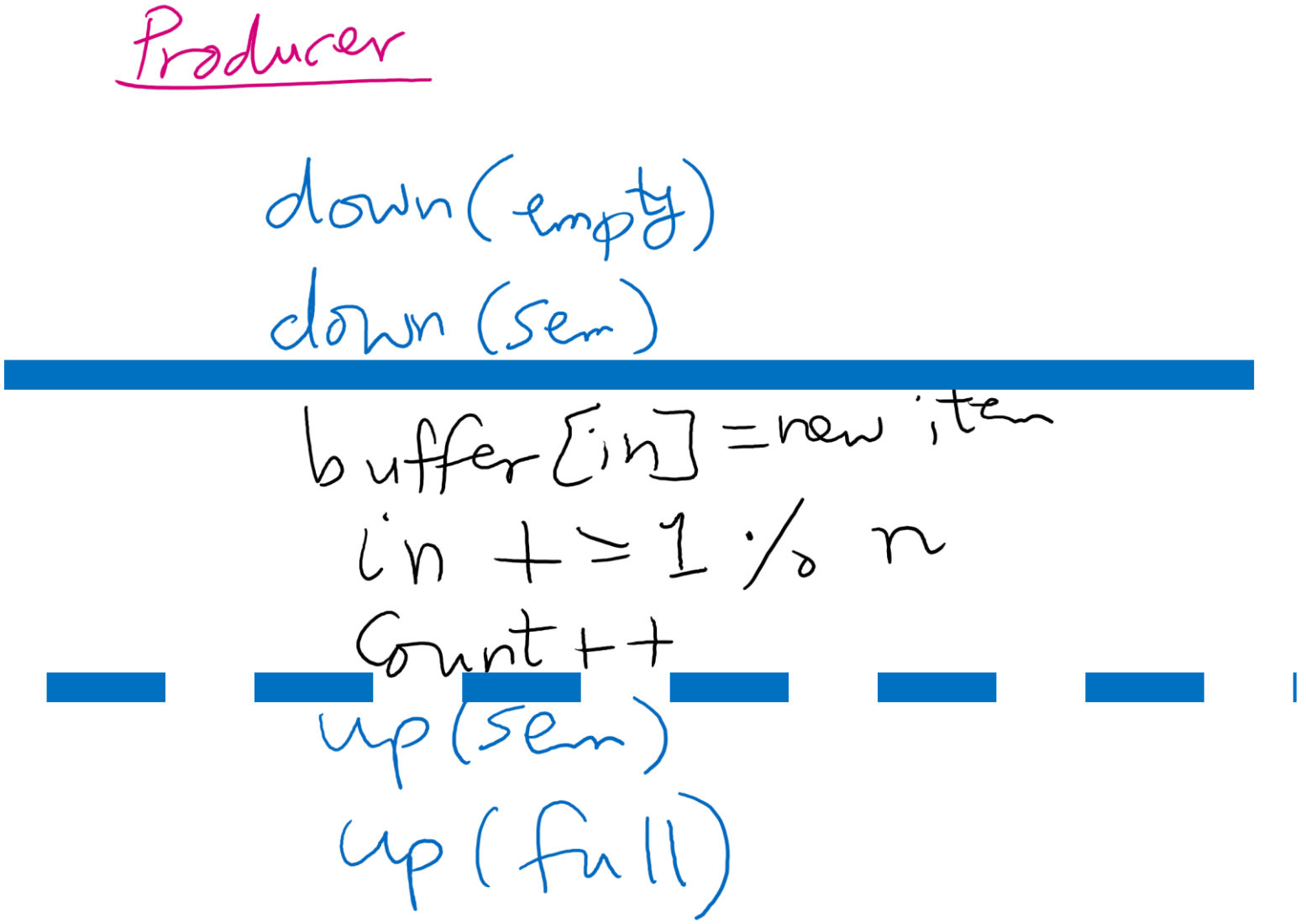
Producer arrives

Moves as far *as possible* until the solid line



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

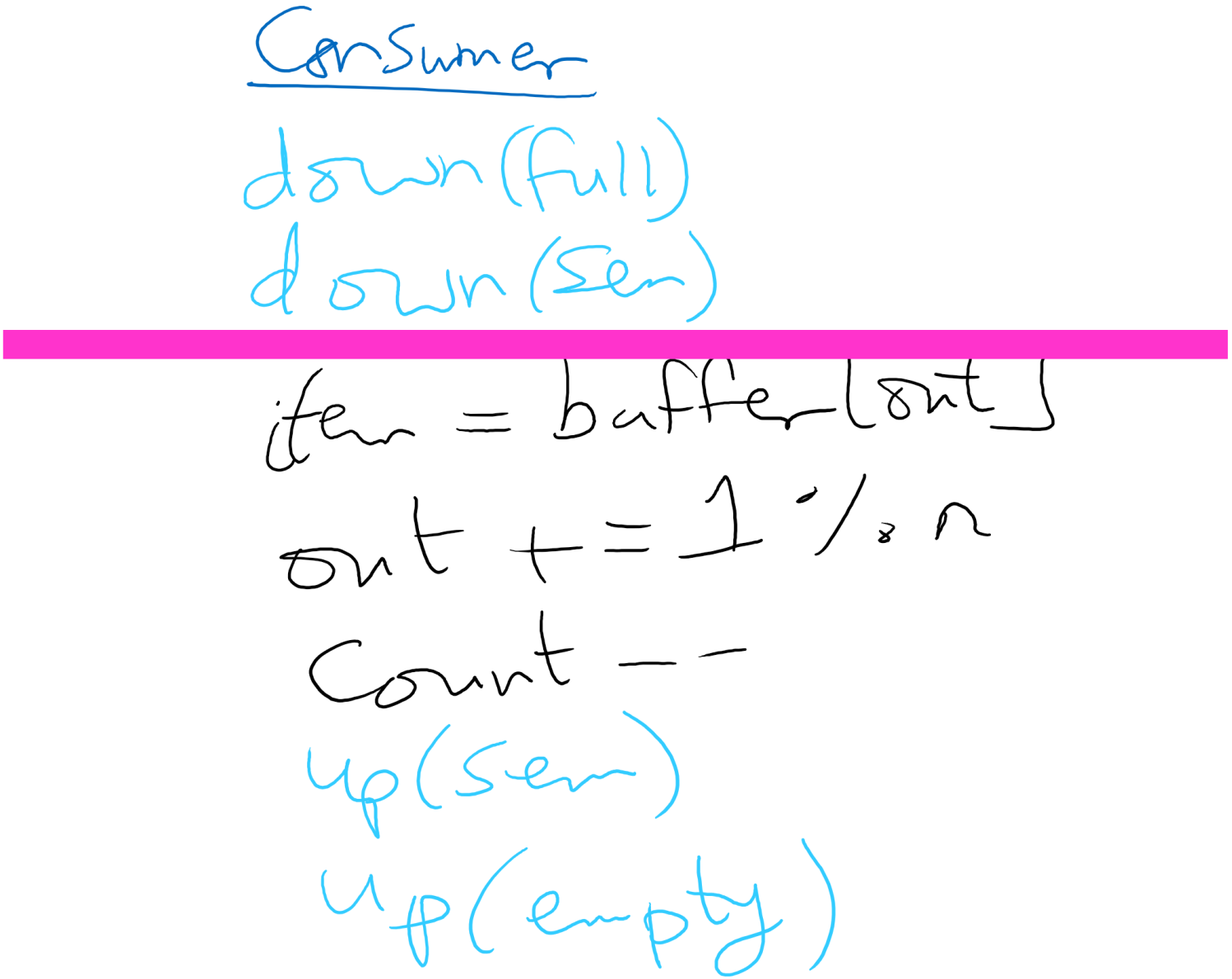
count++

up(sem)

up(full)

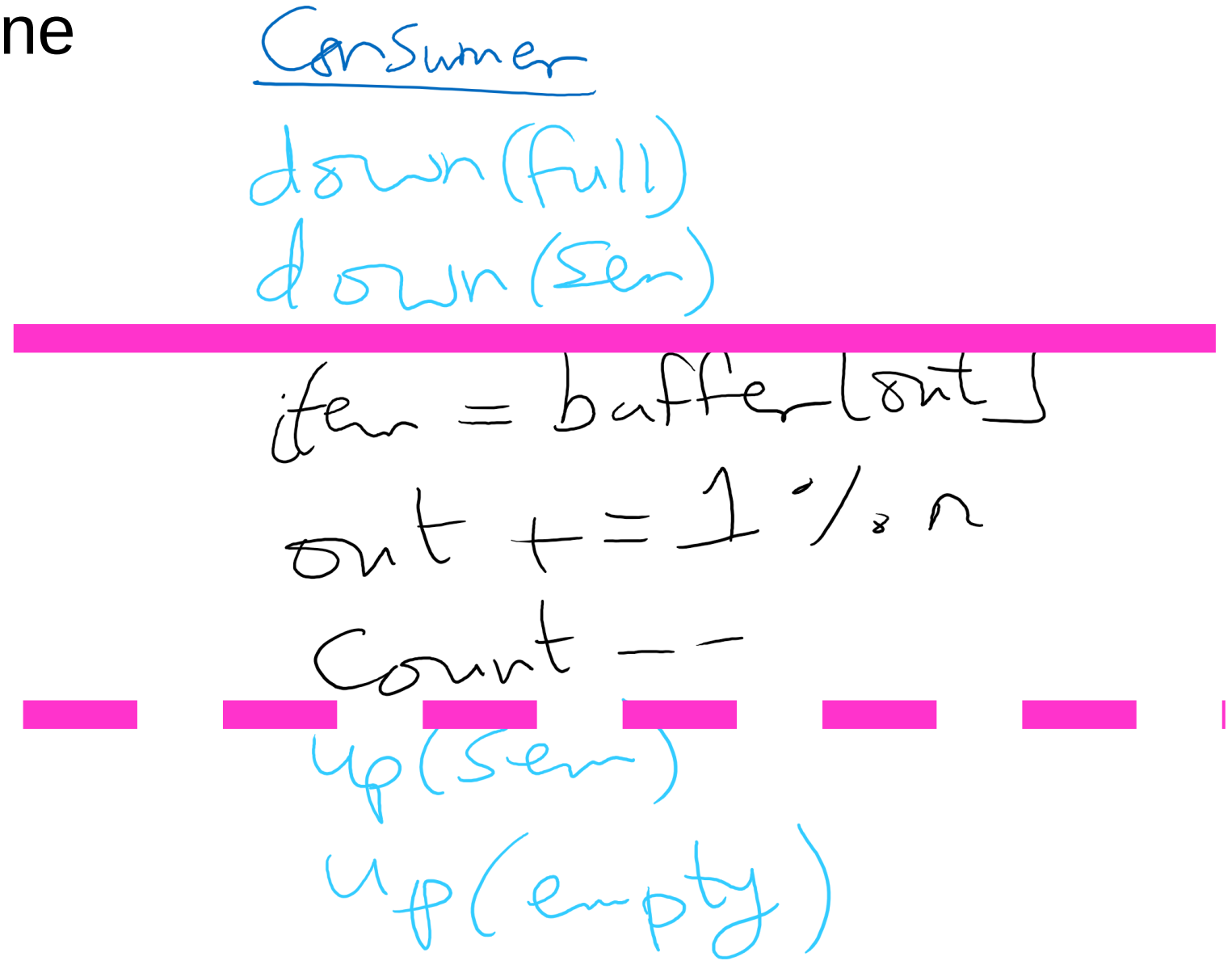
Consumer arrives

Moves as far as possible until the solid line



Consumer 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

Consumer

down(full)

down(sem)

item = buffer[out]

out += 1 % n

count--

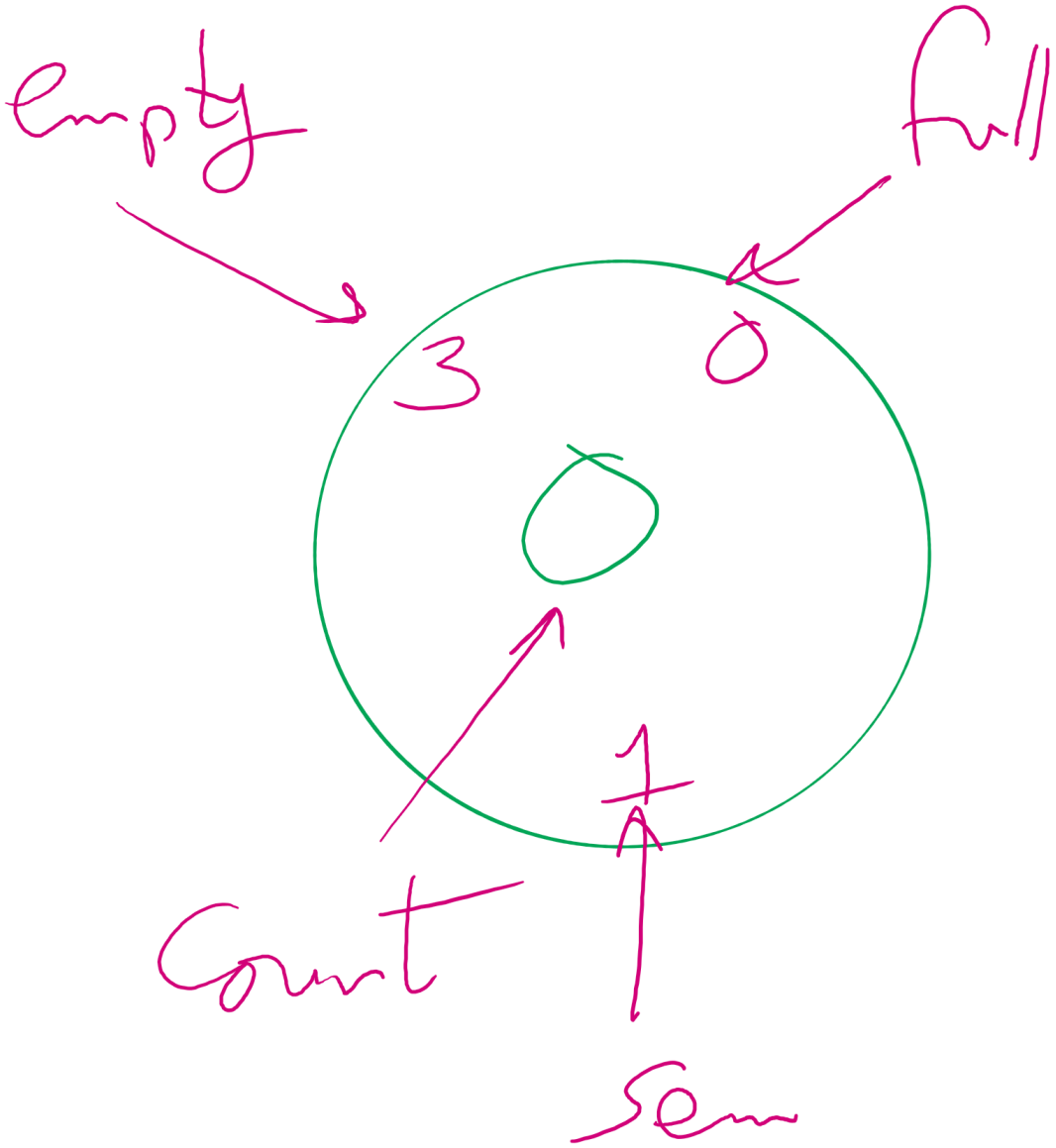
up(sem)

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

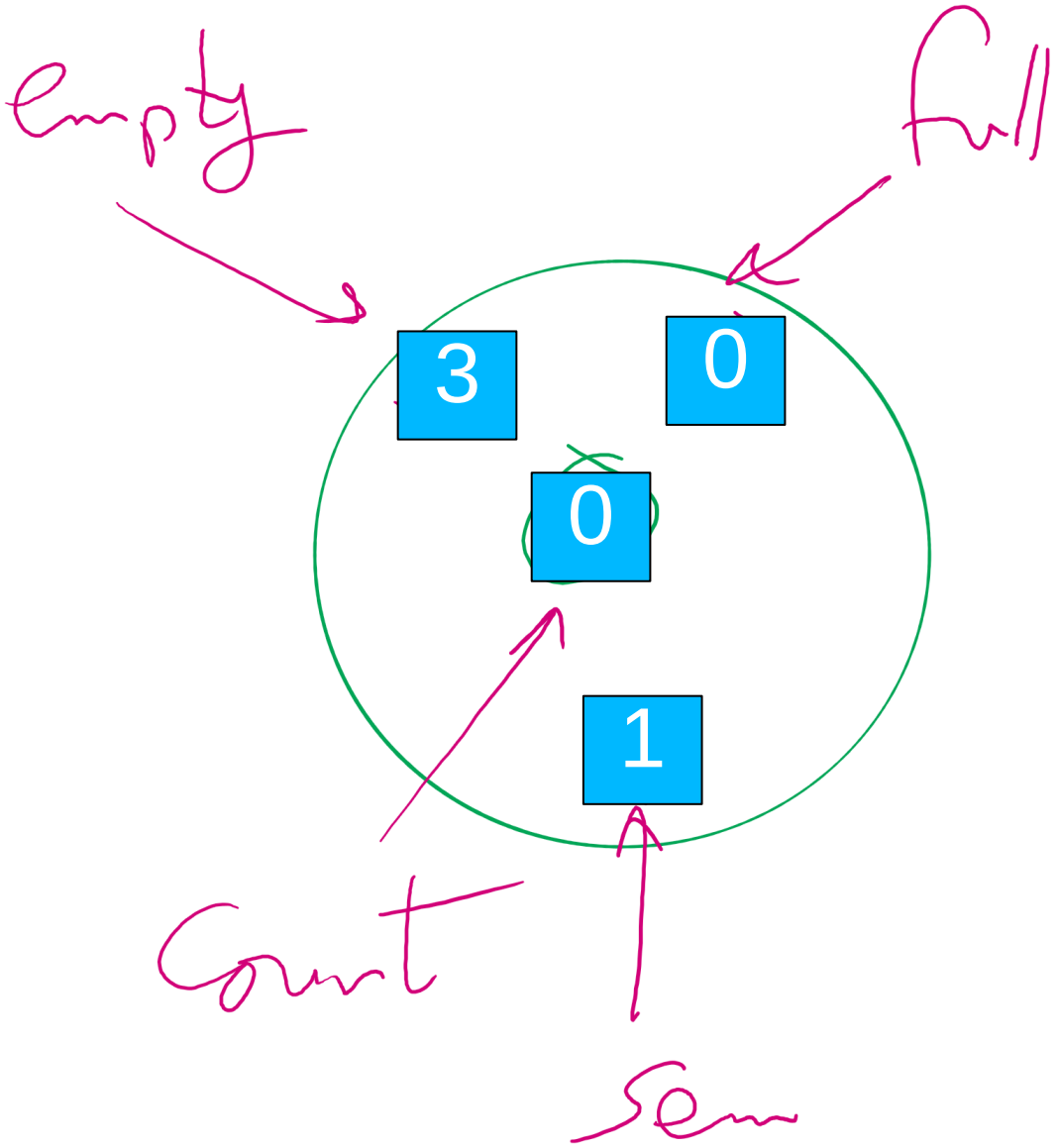


Example 1

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

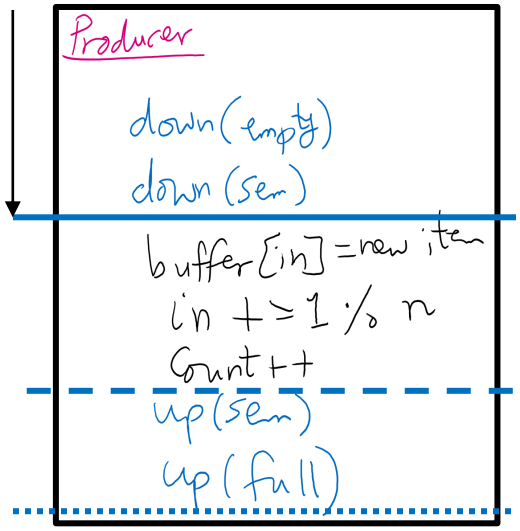
Initial state

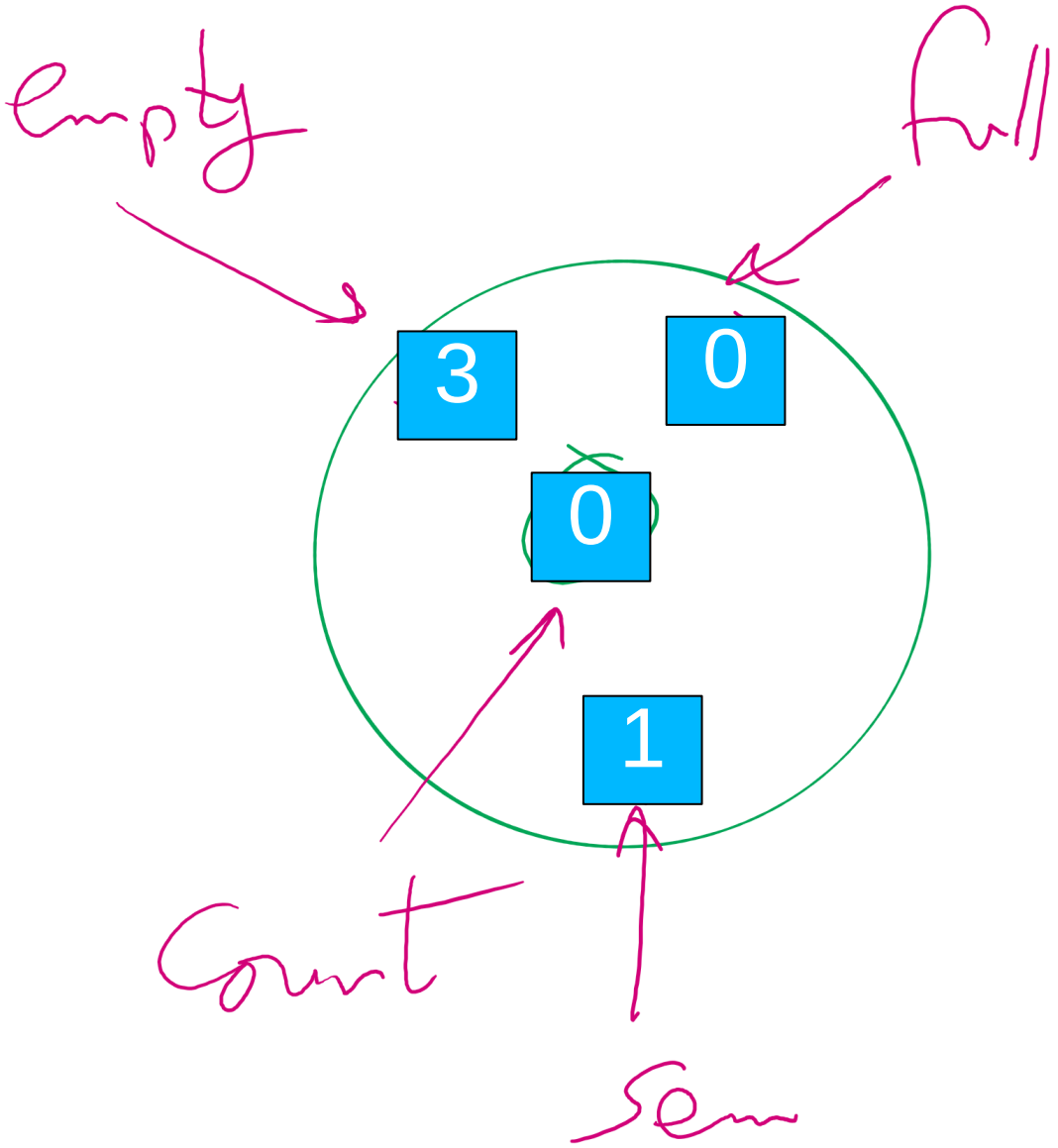


n == 3

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

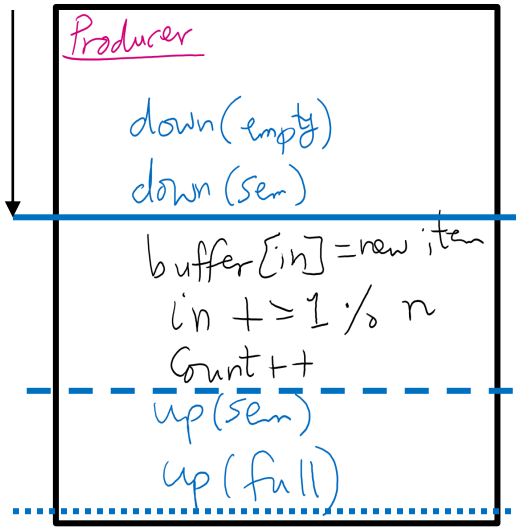


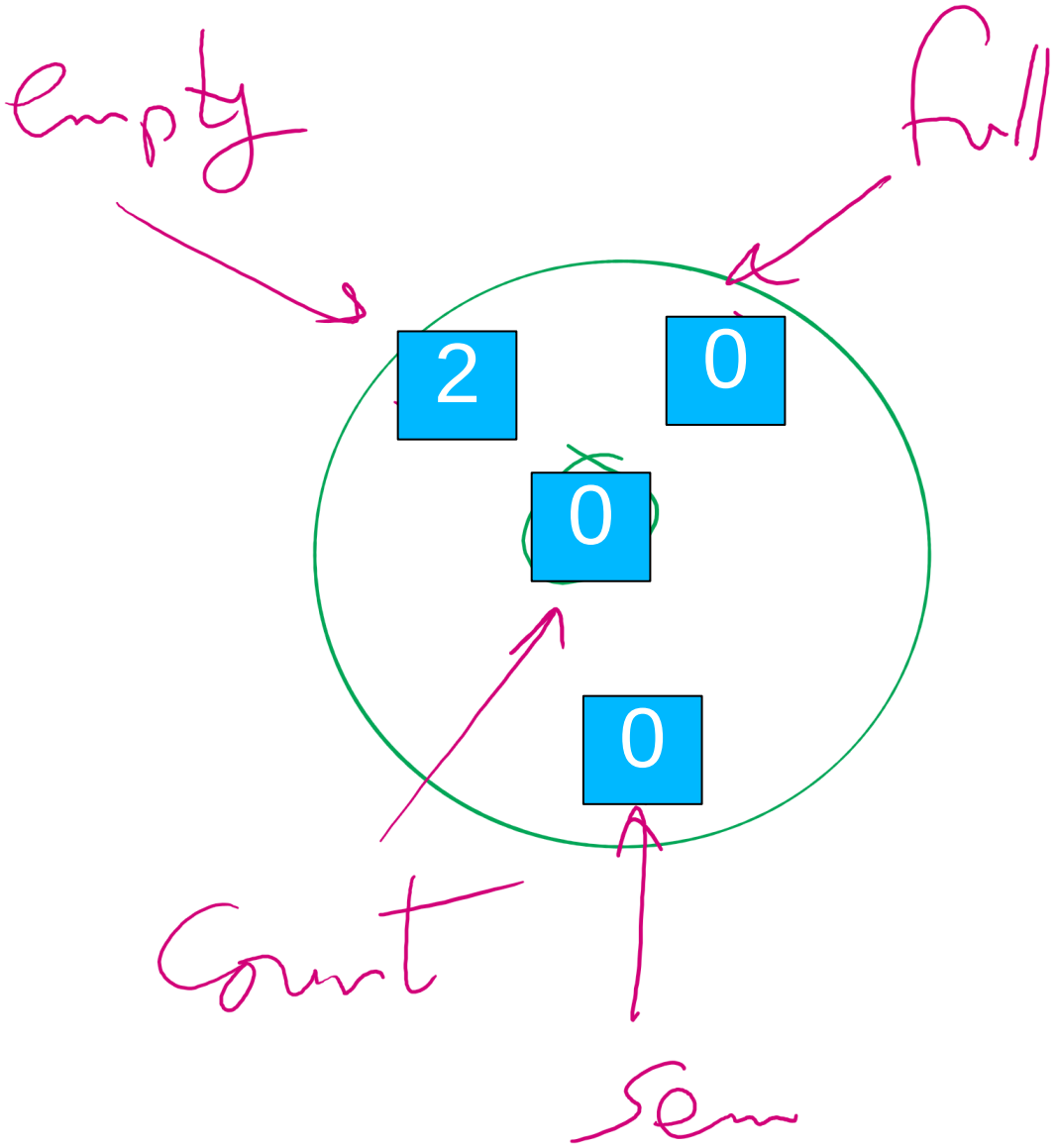


n == 3

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

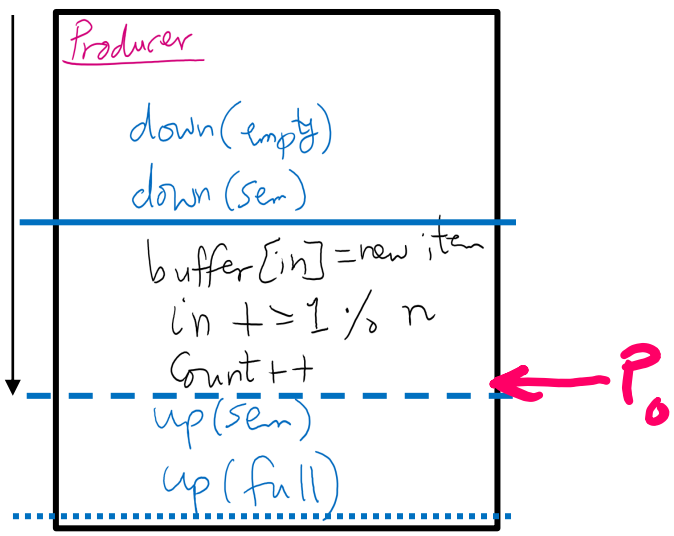


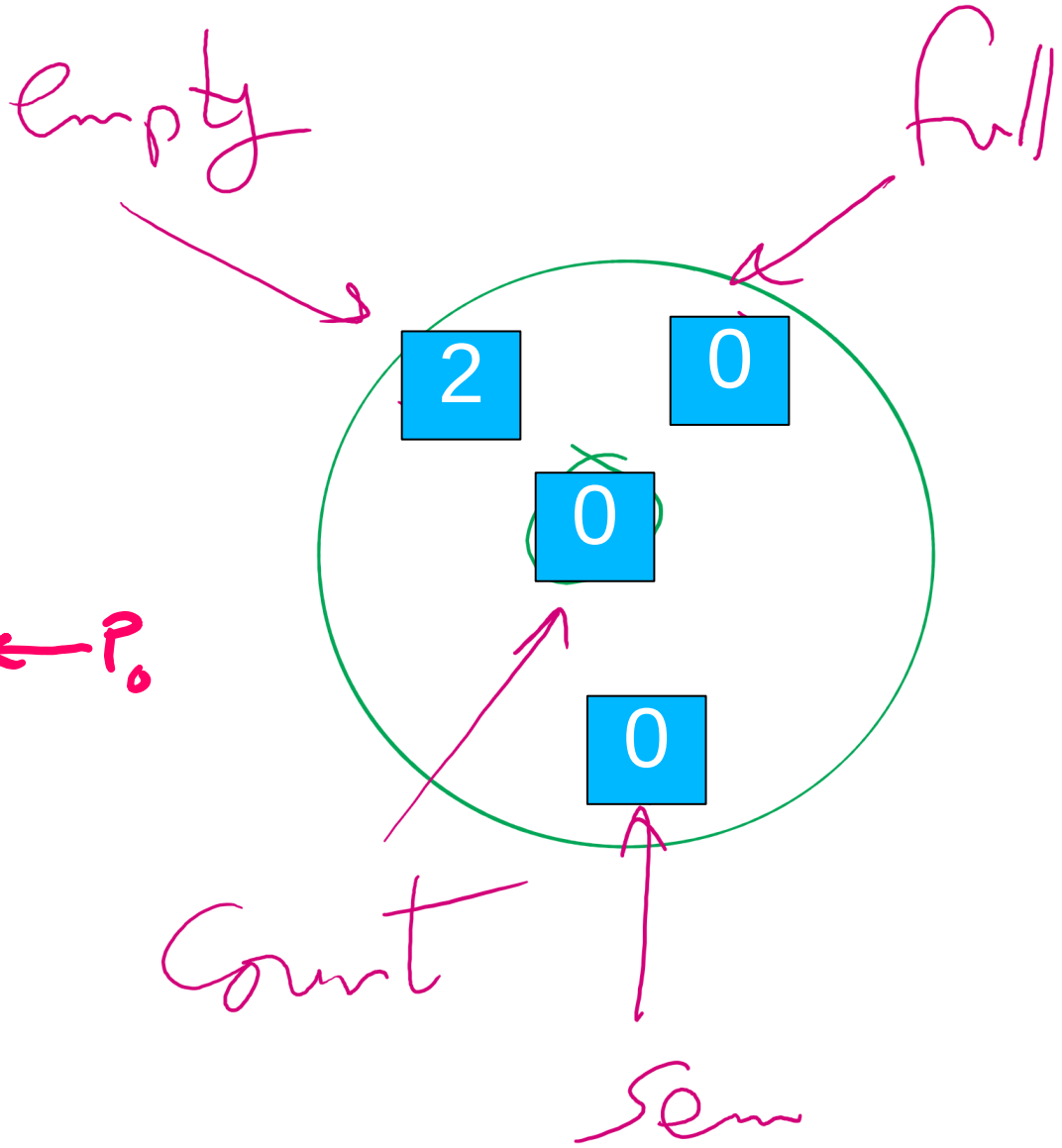


n == 3

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

P0 enters

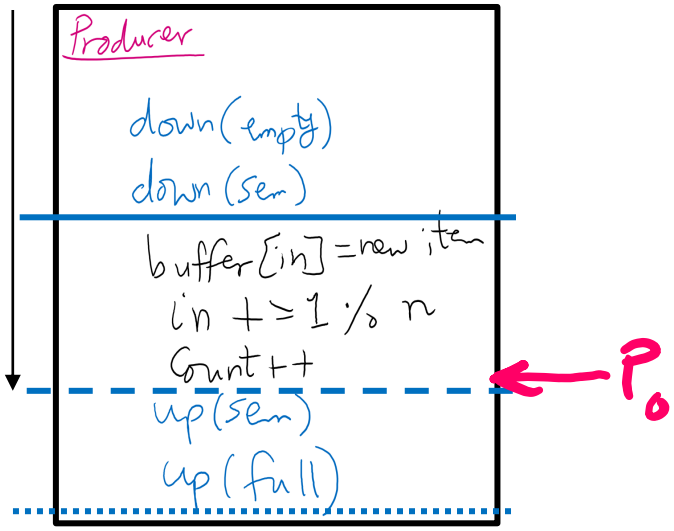


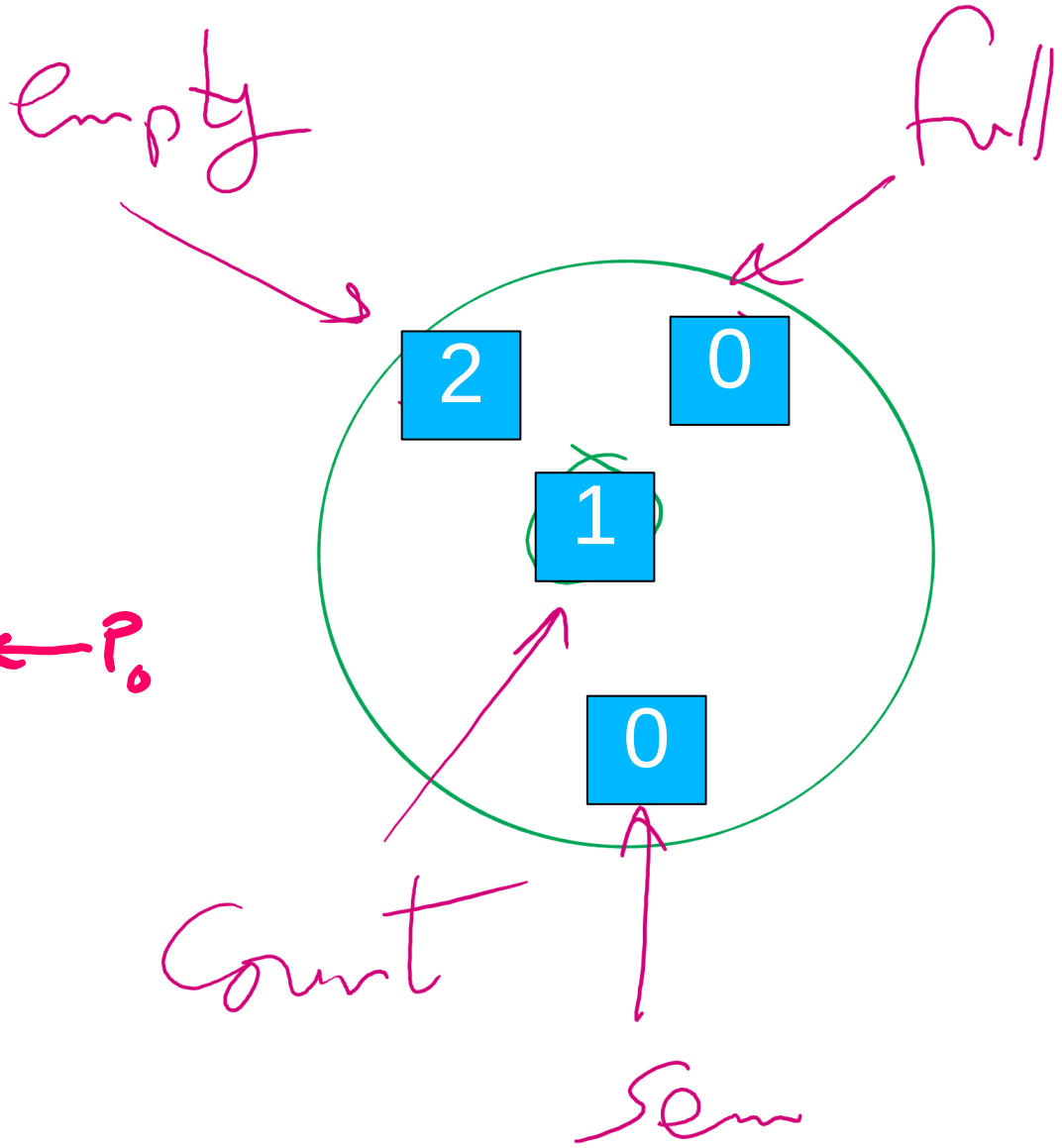


n == 3

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

P0 enters

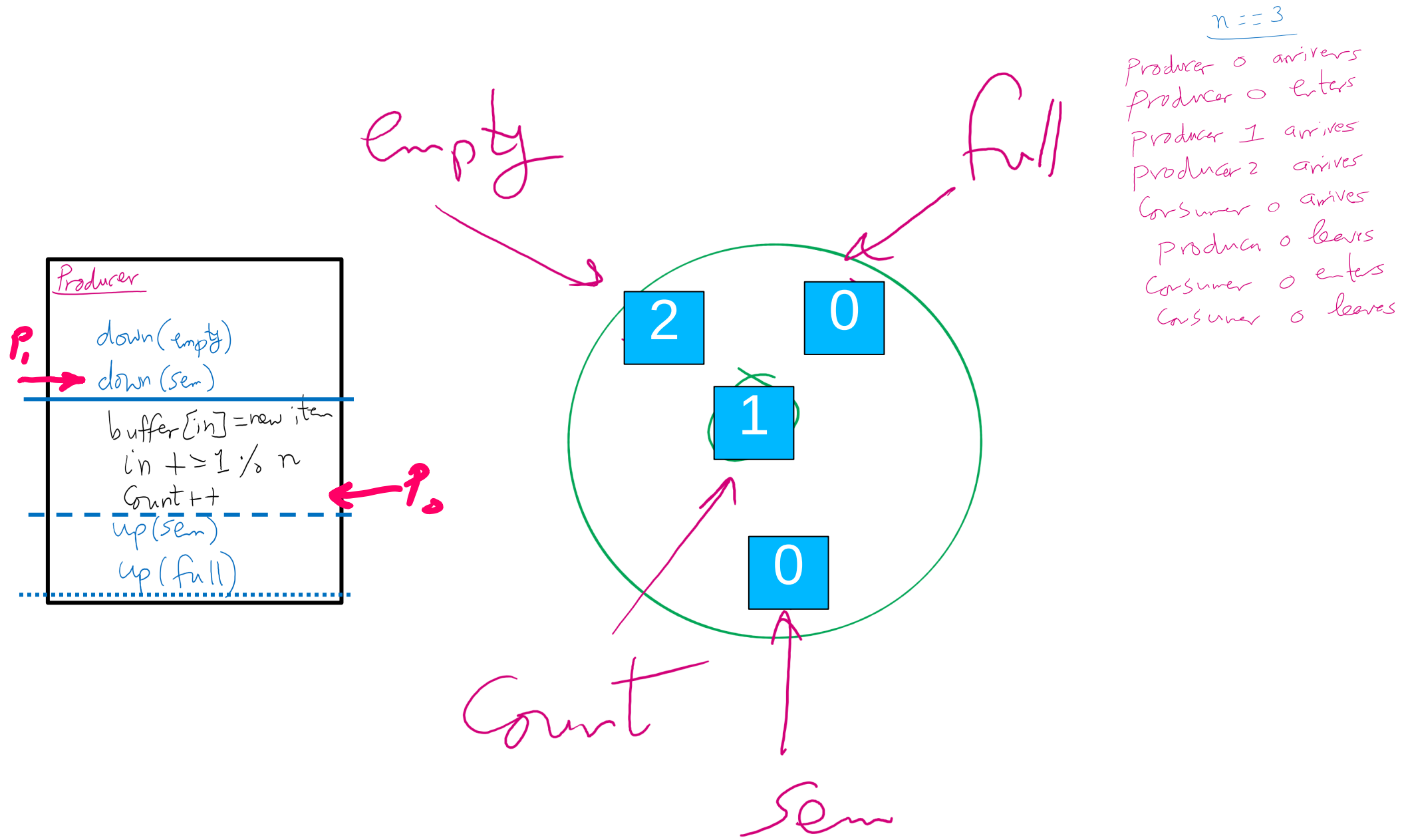




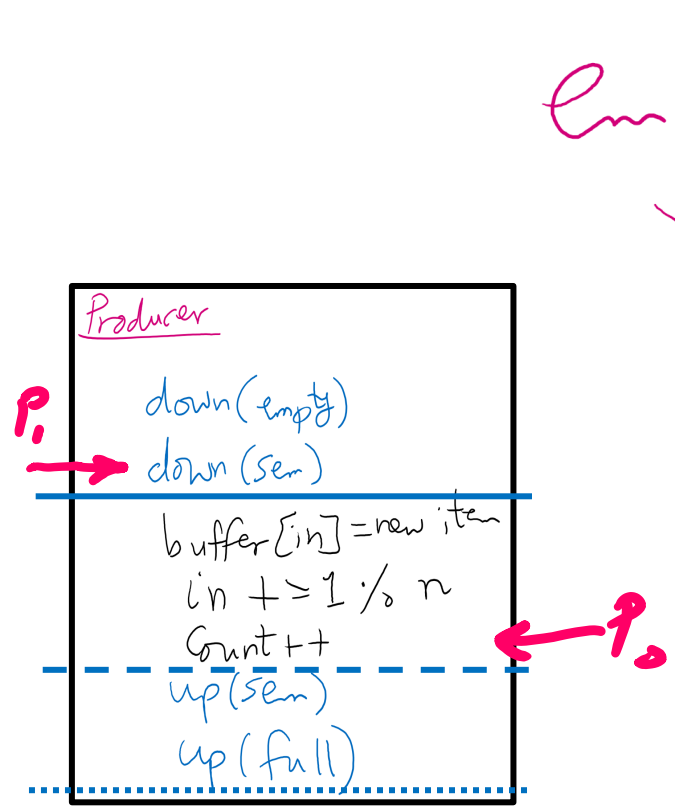
n == 3

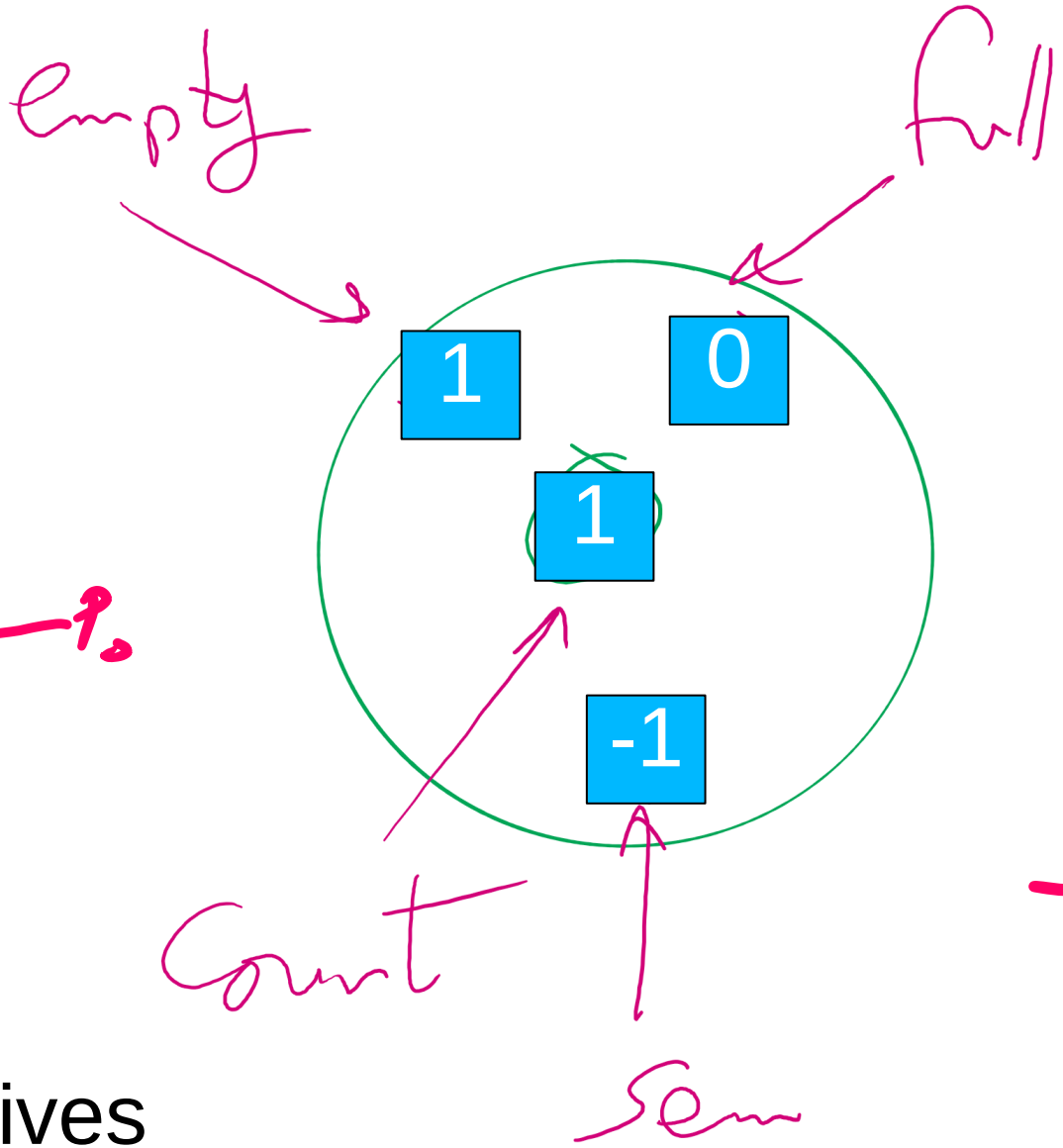
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

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

P1 arrives





n == 3

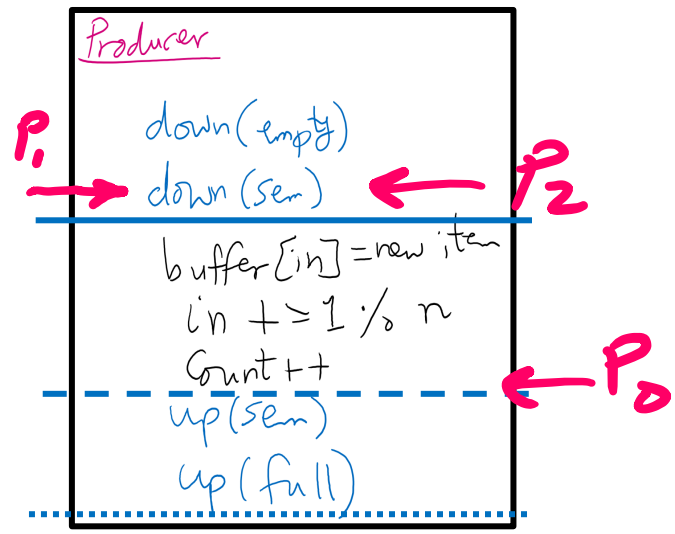
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

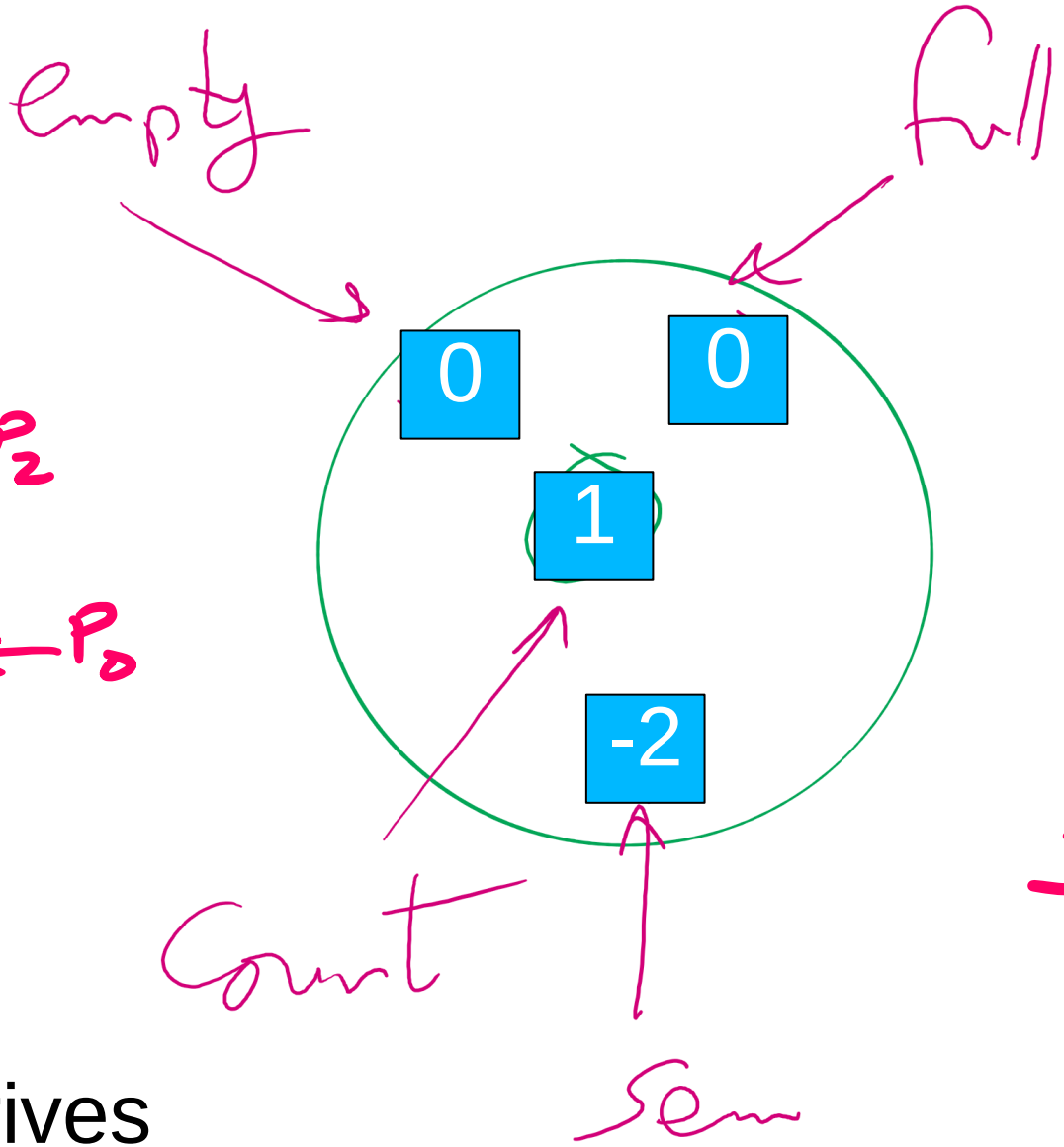
Sem Queue

P1

Next, P2 arrives

P2 arrives

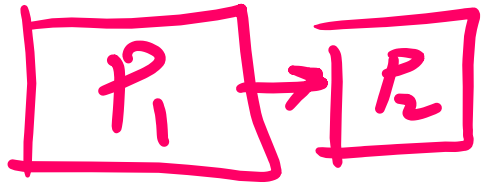




n == 3

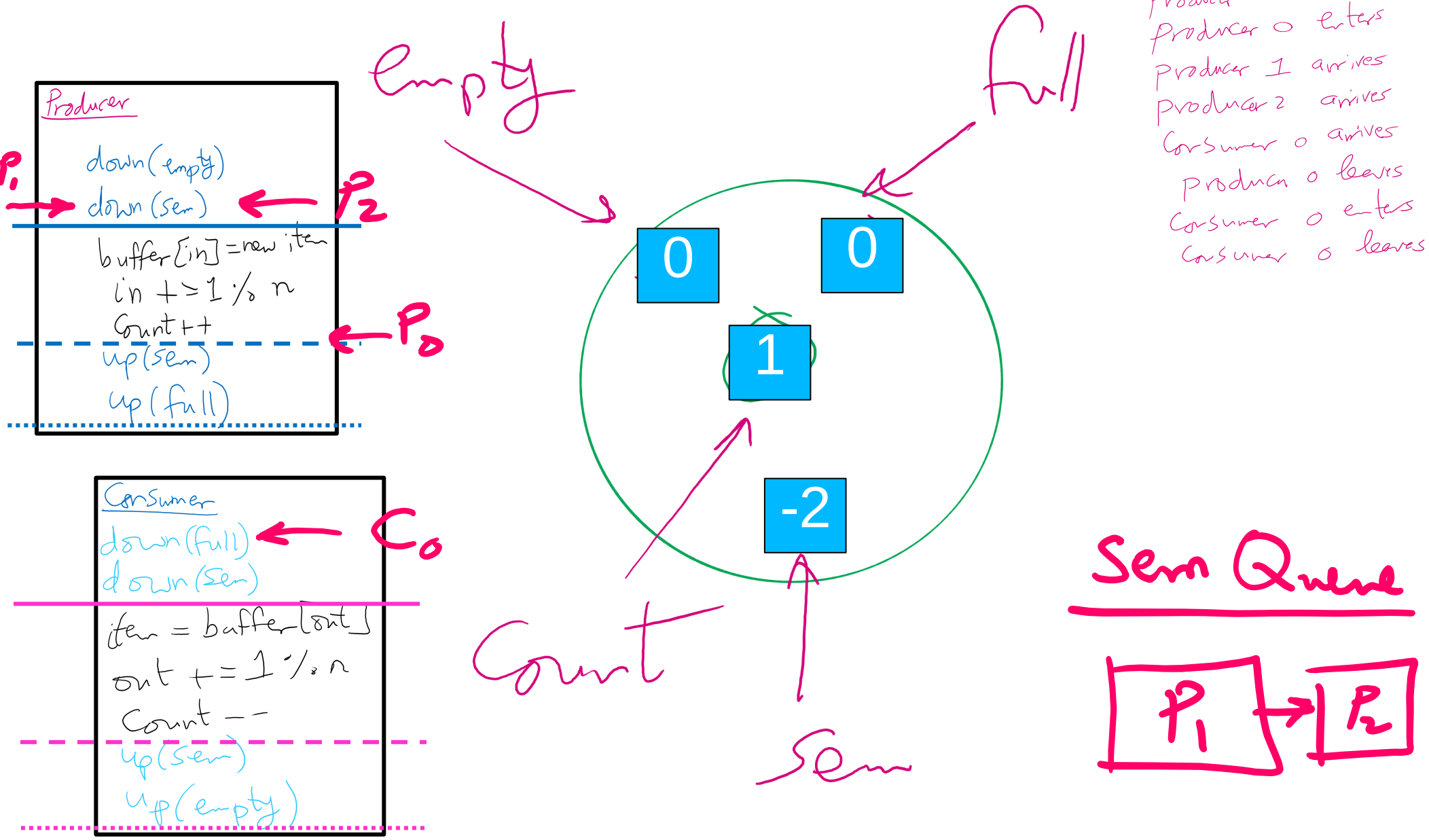
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

Sem Queue

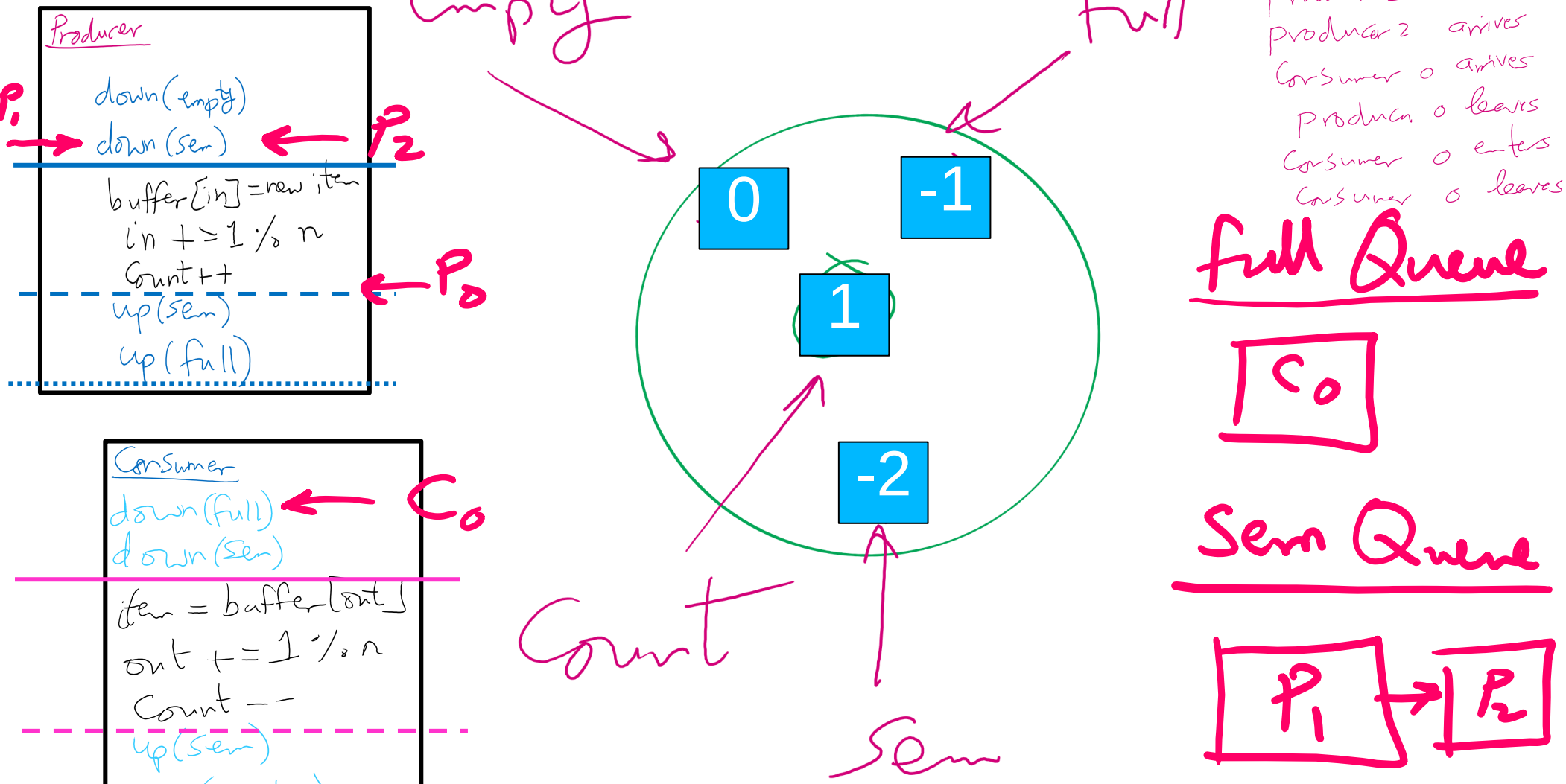


Next, C0 arrives

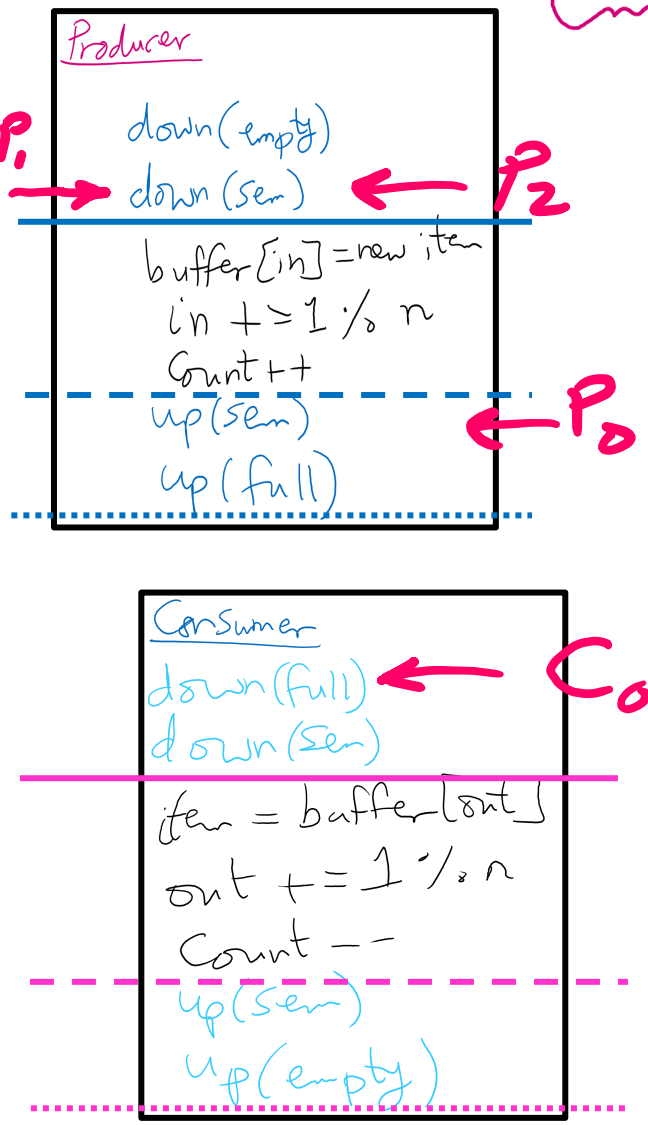
C0 arrives

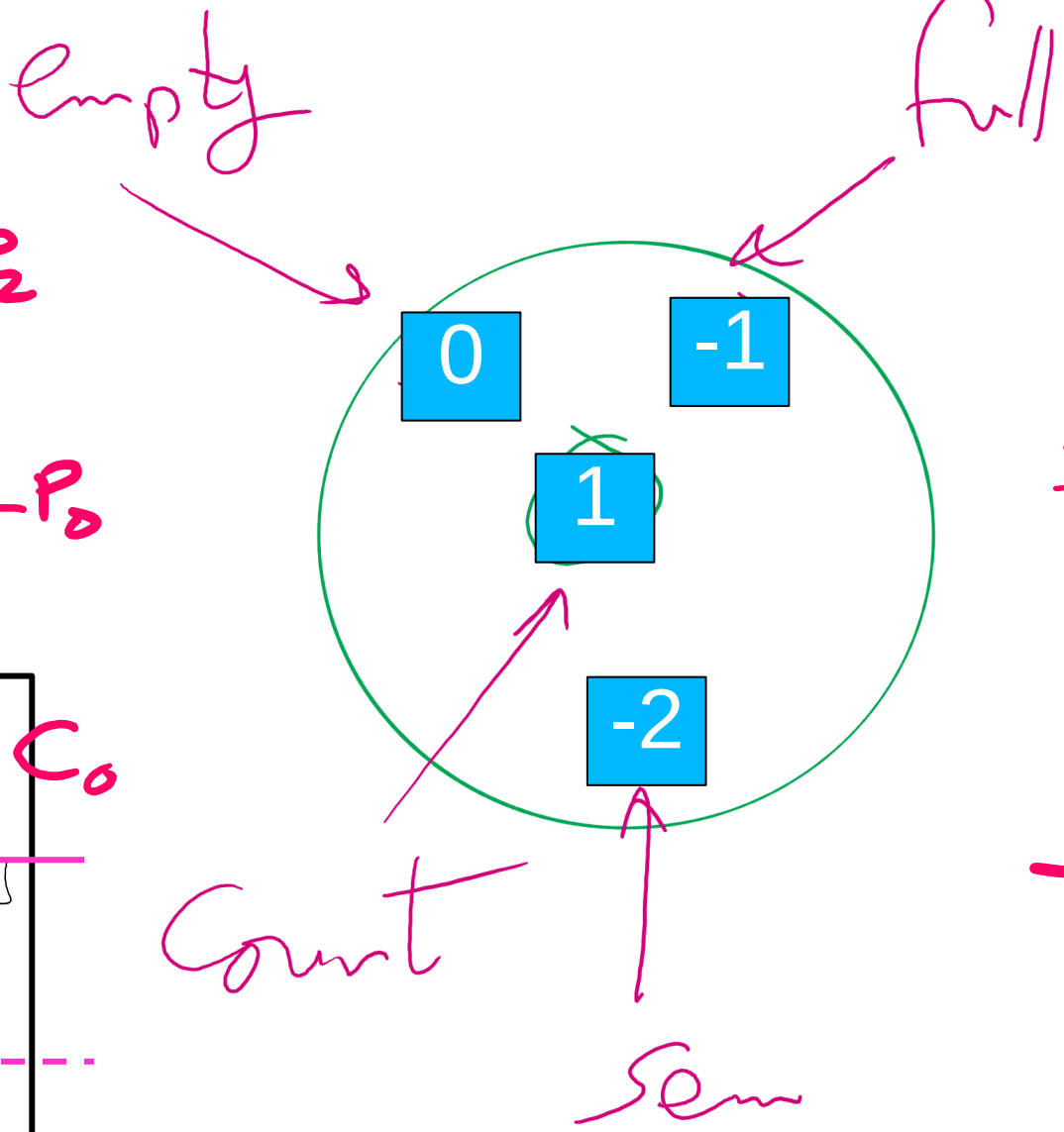

n == 3
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P1
P2
P0
Consumer
down(full)
down(sem)
item = buffer[out]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

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



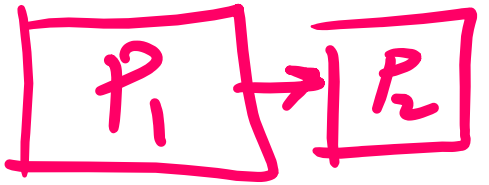


n == 3

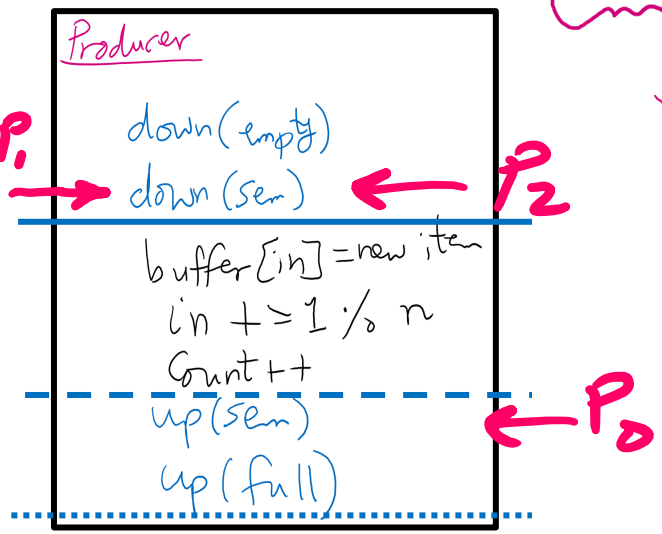
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

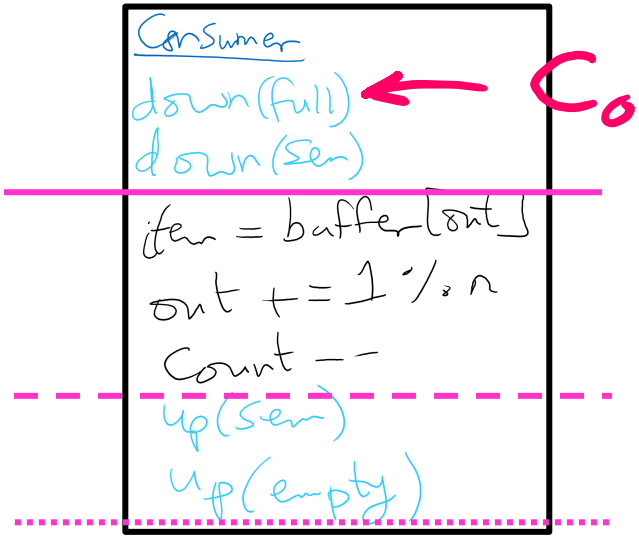
full Queue

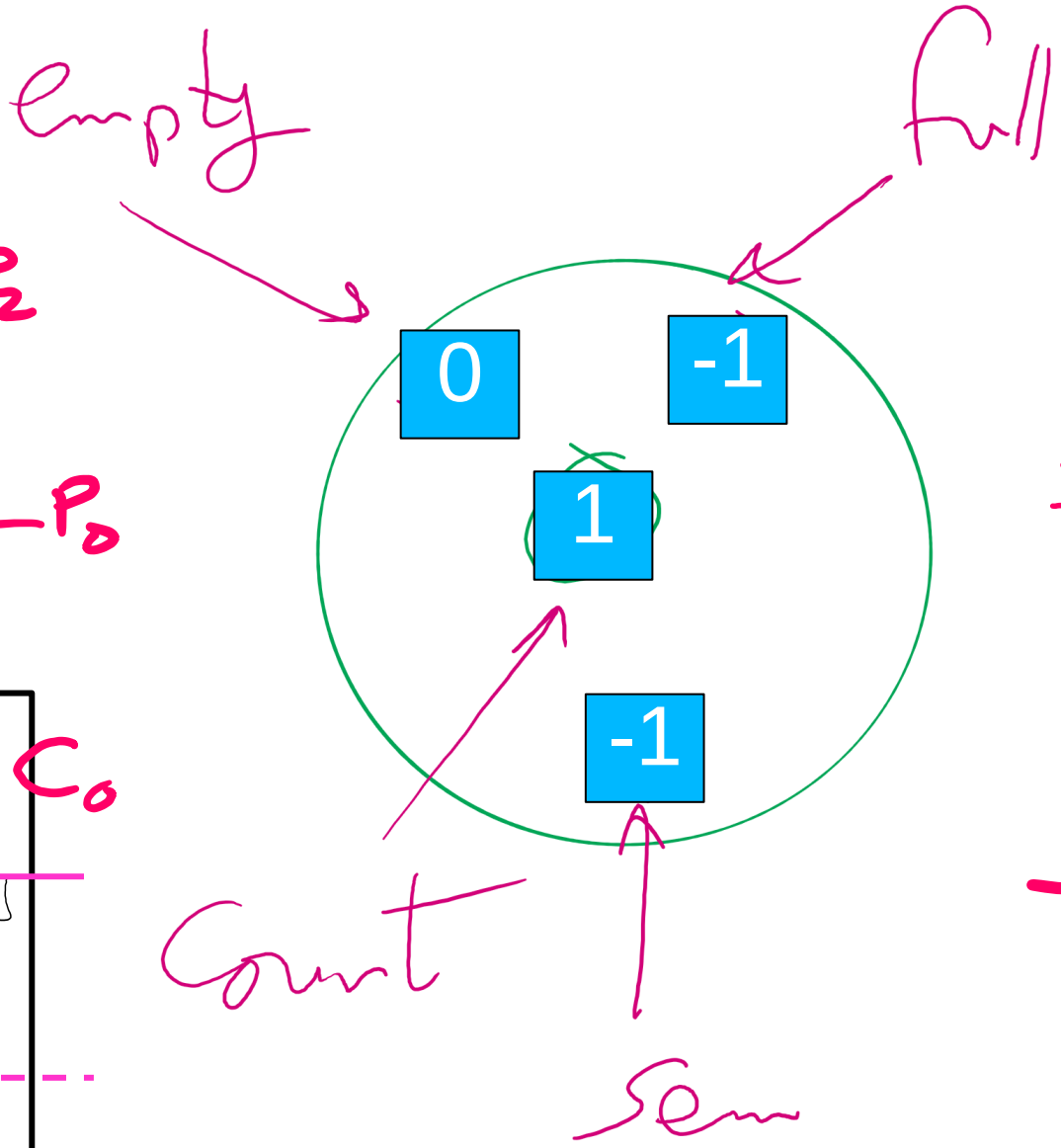
Sem Queue



P0 leaves





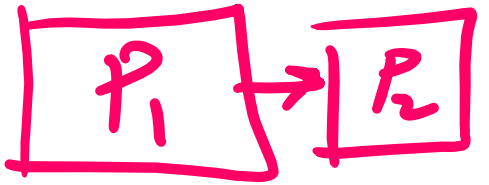


n == 3

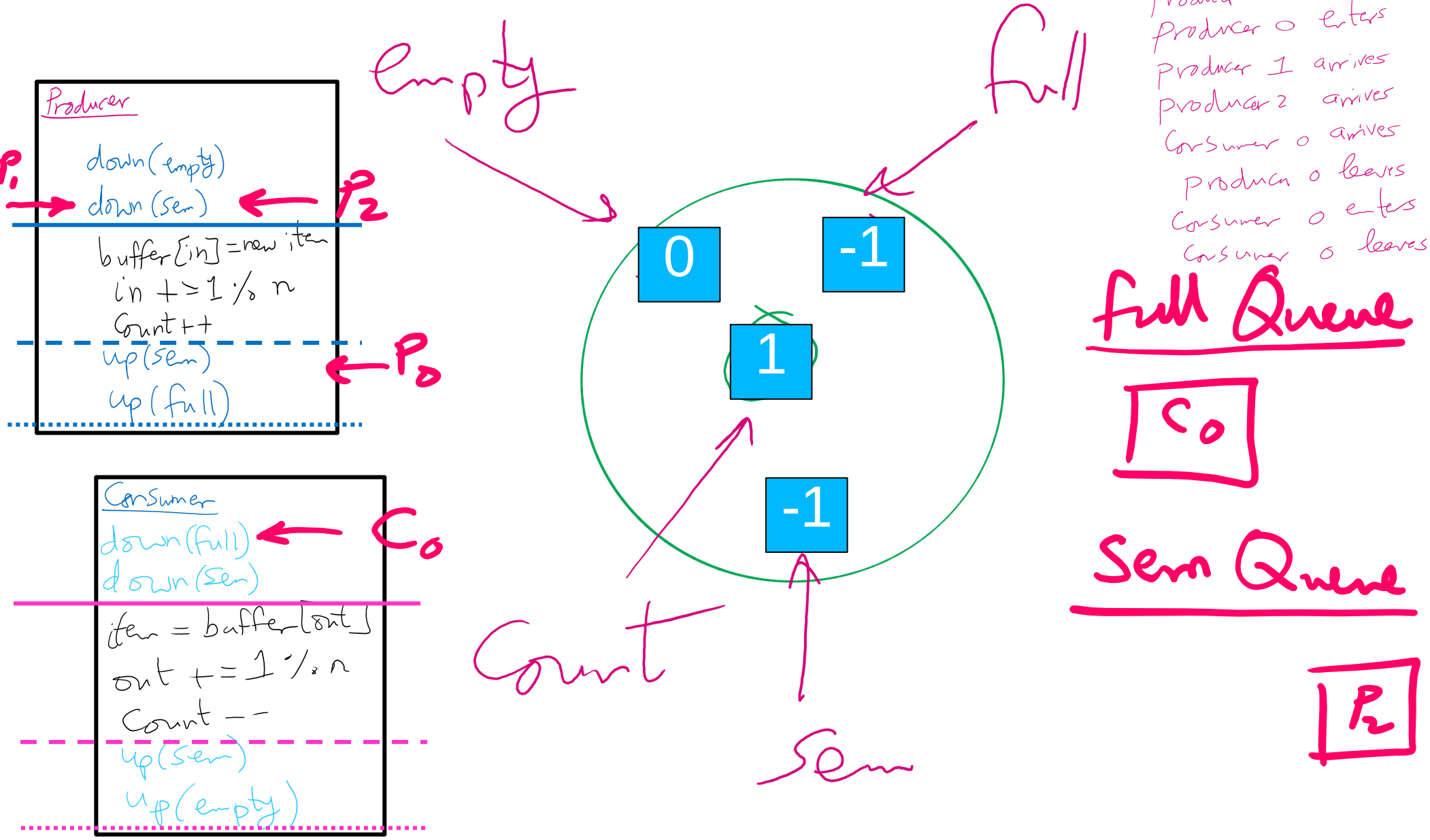
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves

full Queue

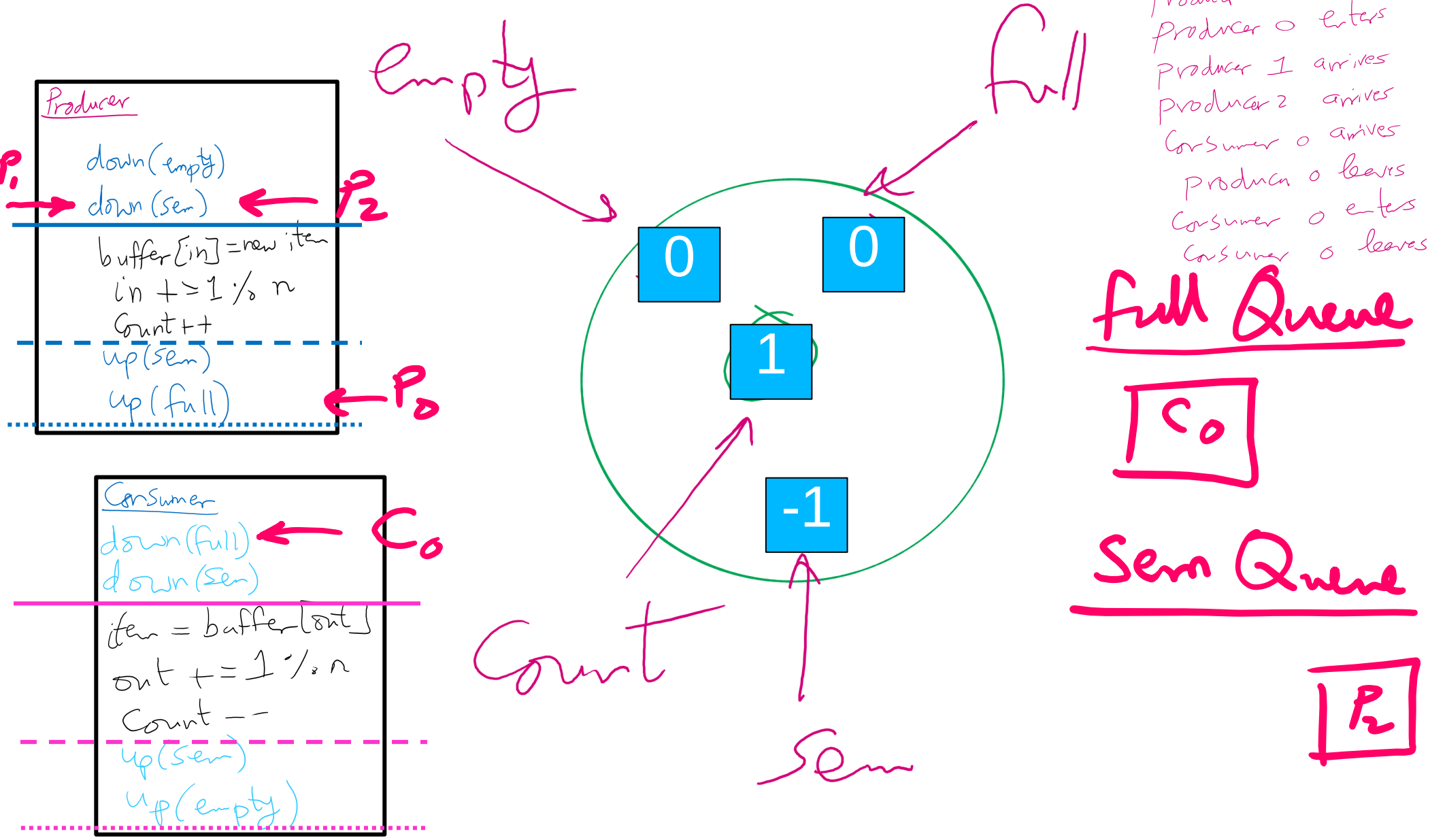
Sem Queue



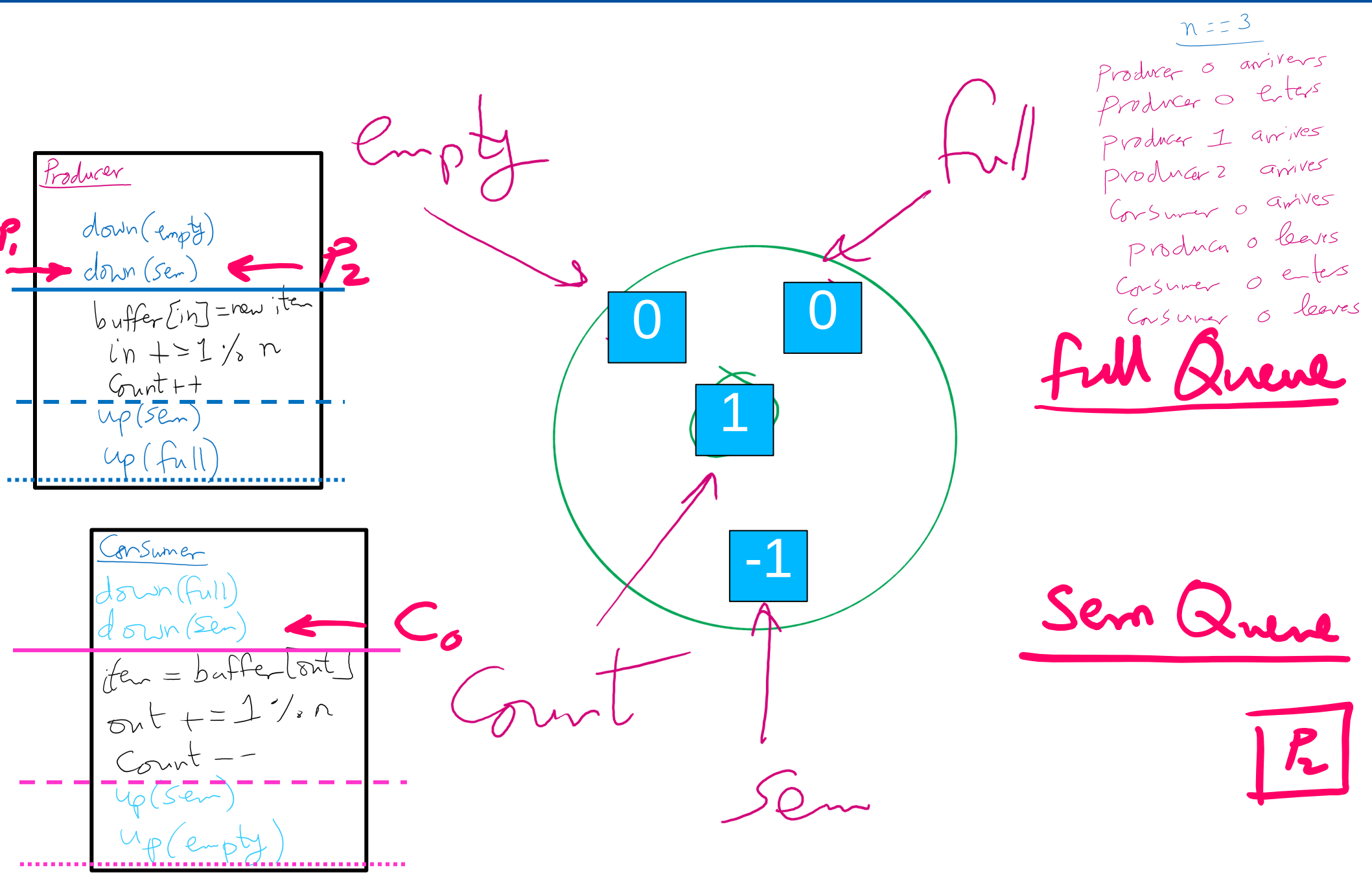
P0 leaves


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
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
P2

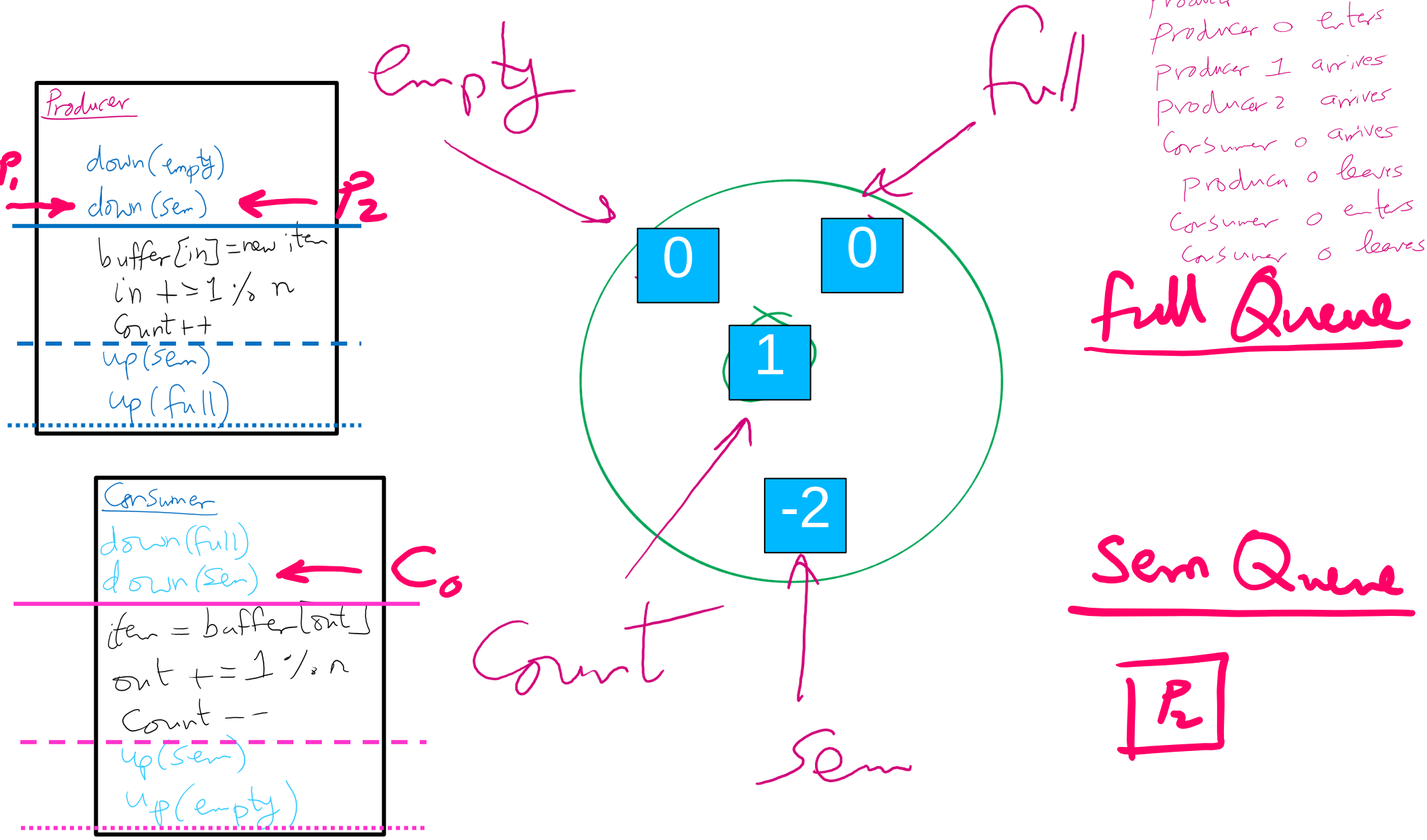
P0 leaves


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
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
P2

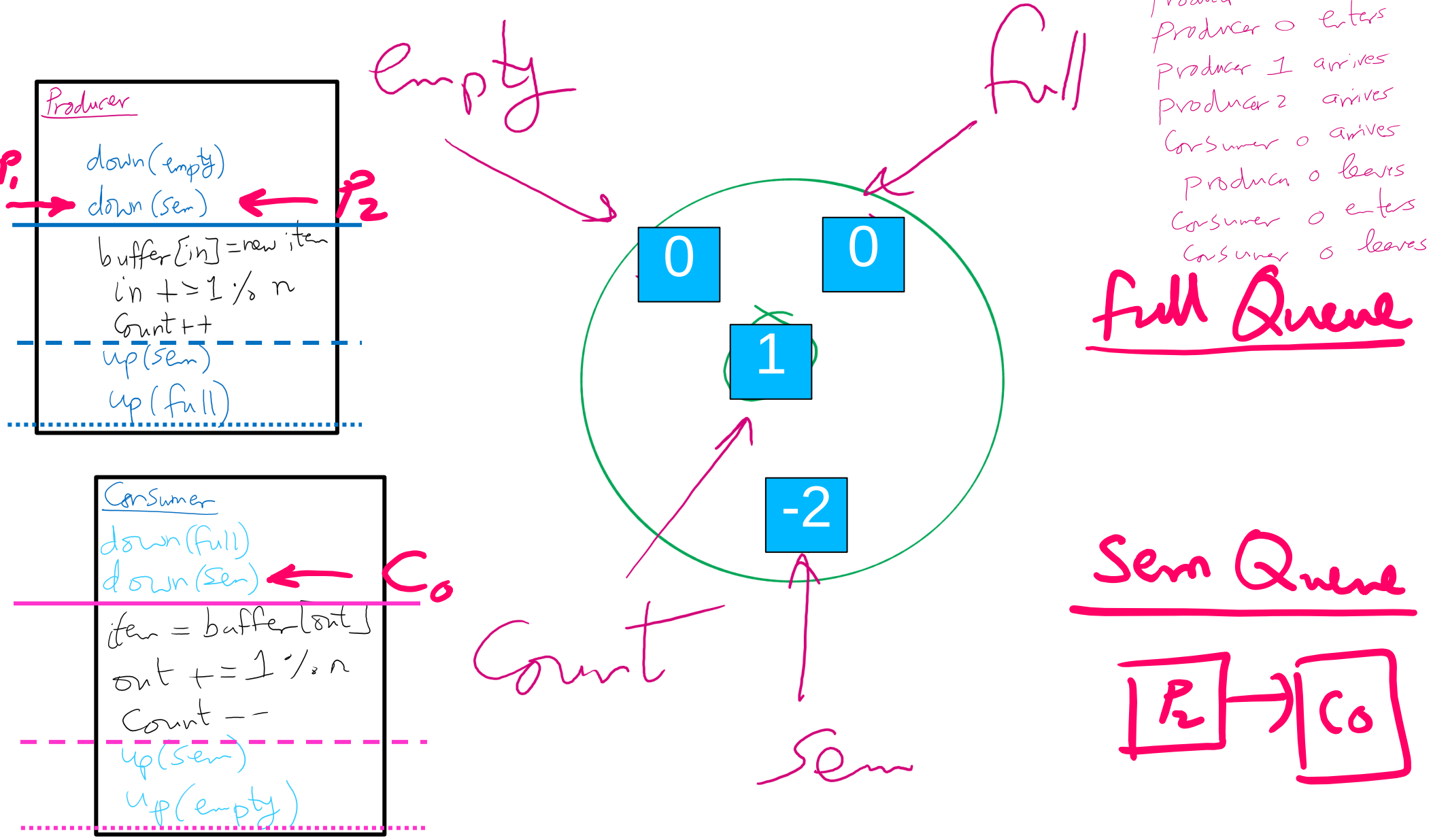
C0 enters


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
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
Count++
up(sem)
up(full)
P1
P2
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

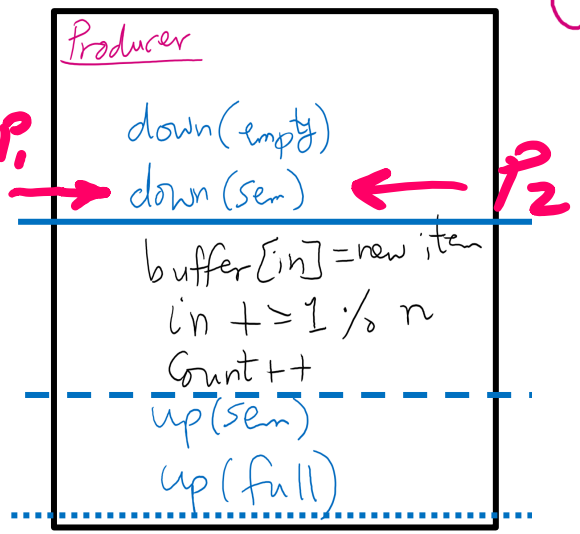
C0 enters

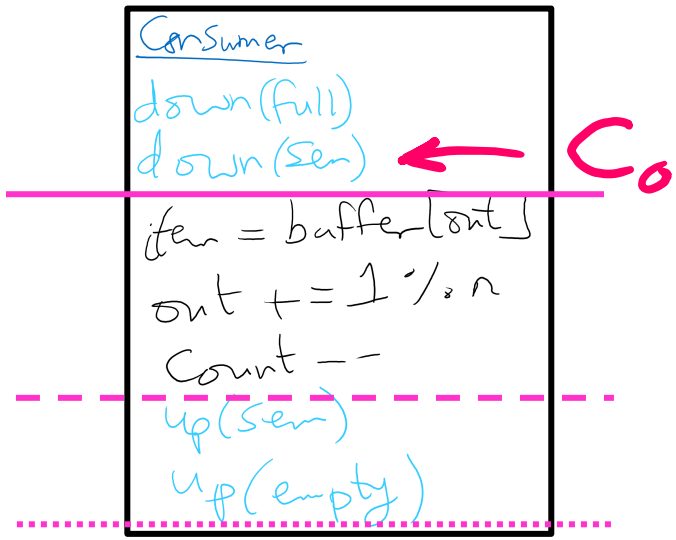

n == 3
Producer 0 arrives
Producer 0 enters
Producer 1 arrives
Producer 2 arrives
Consumer 0 arrives
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
full Queue
Sem Queue
P2
empty
full
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
Count++
up(sem)
up(full)
P1
P2
Consumer
down(full)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

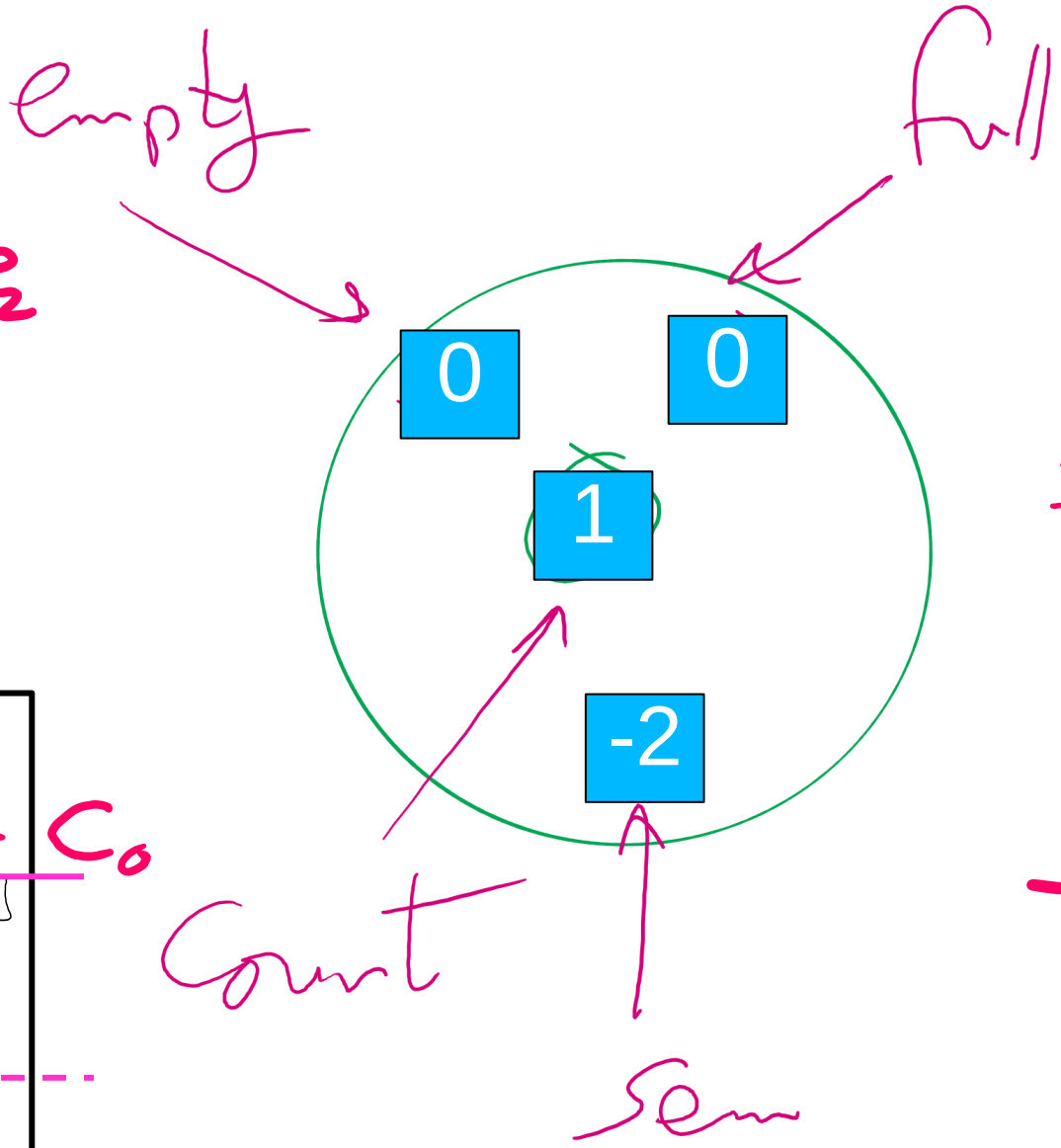
C0 enters


Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P1
P2
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 arives

Producer 0 arives

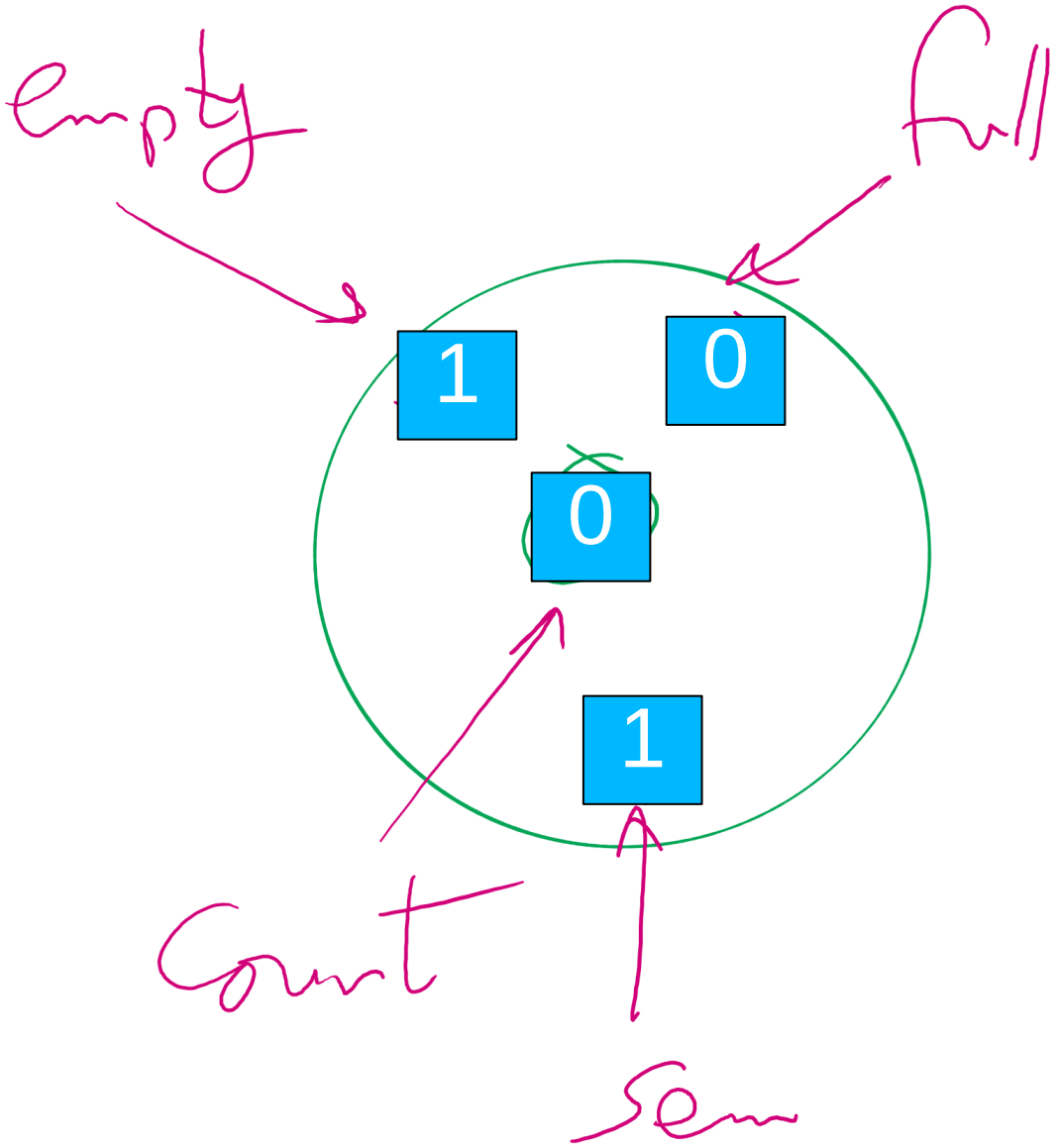
Producer 0 enters

Producer 0 leaves

Consumer 0 enters

Consumer 0 leaves

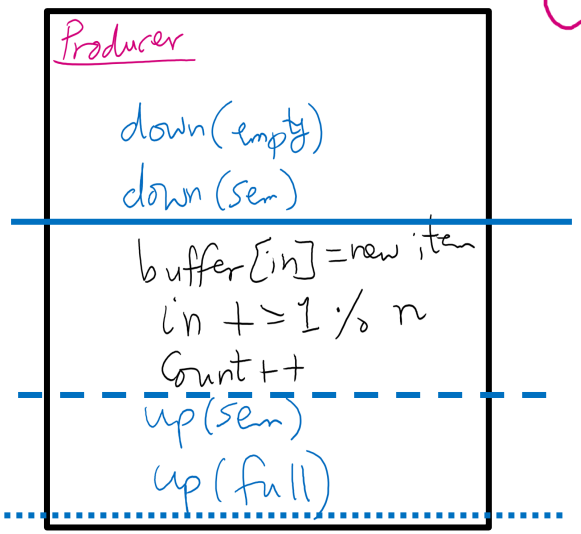
Initial state

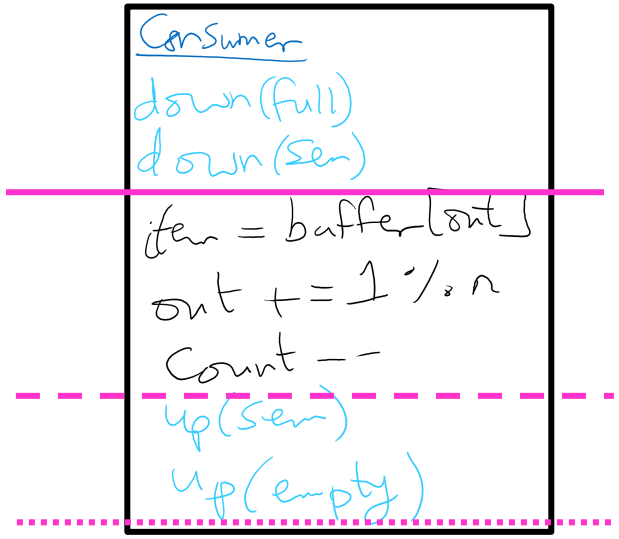


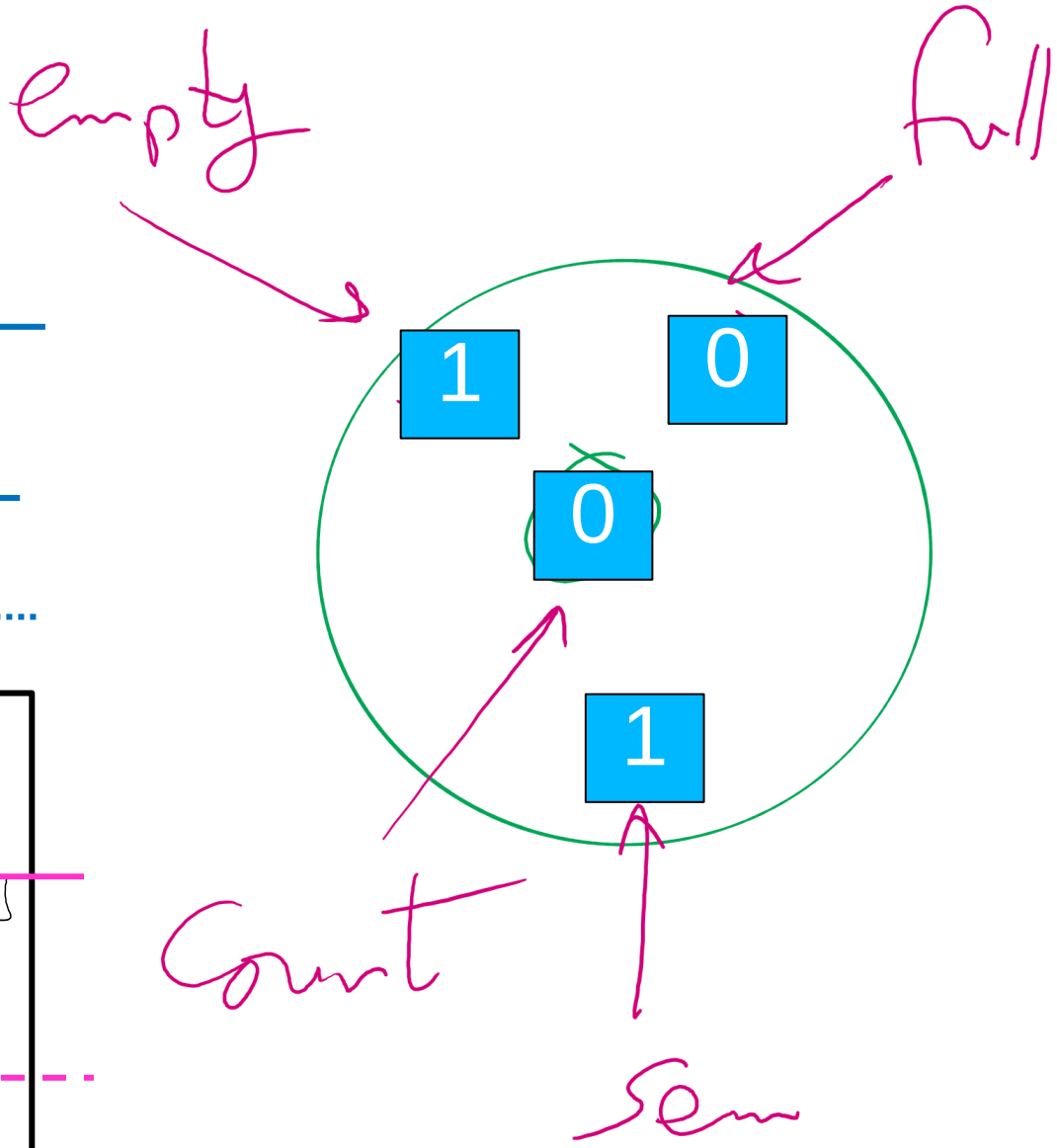
n == 1

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

C0 arrives



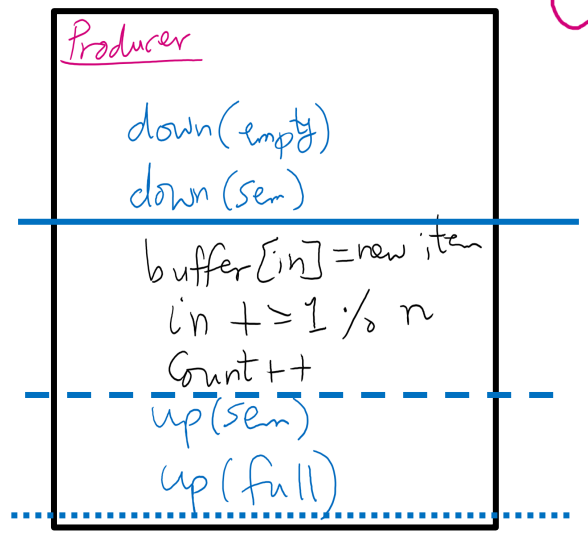


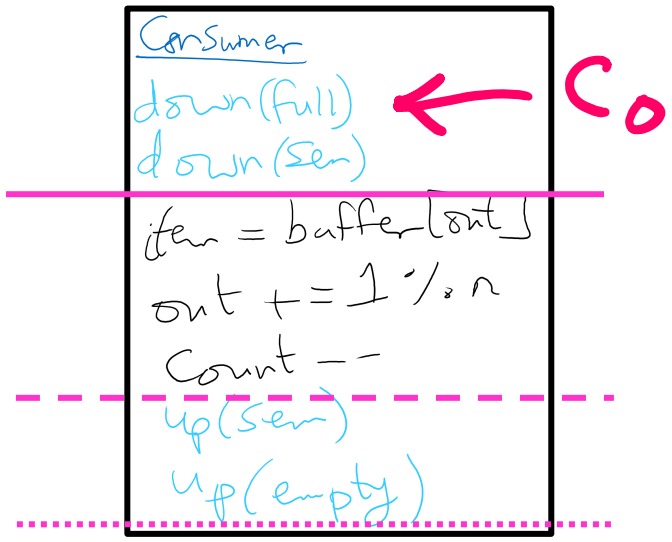


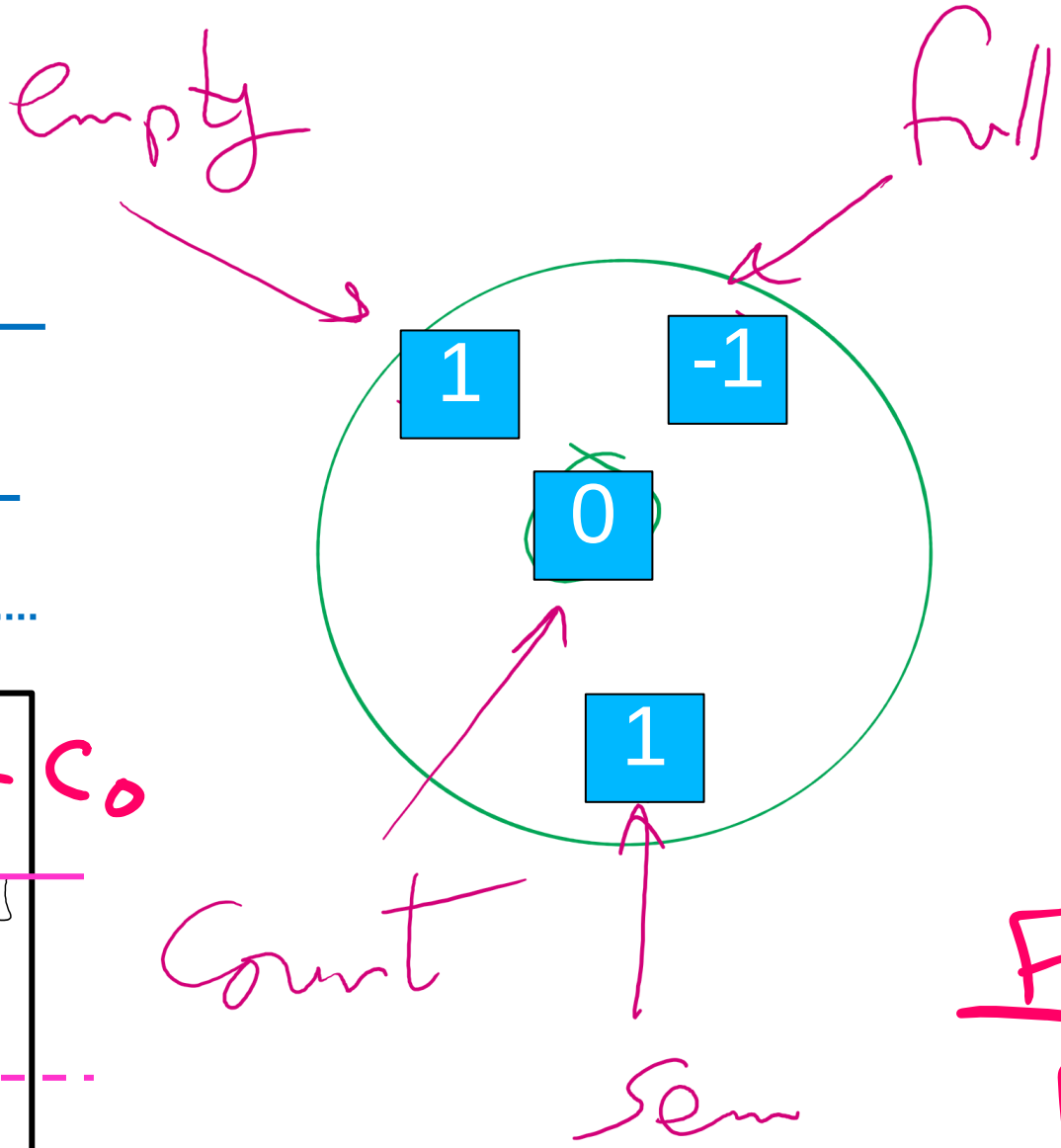
n == 1

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

C0 arrives







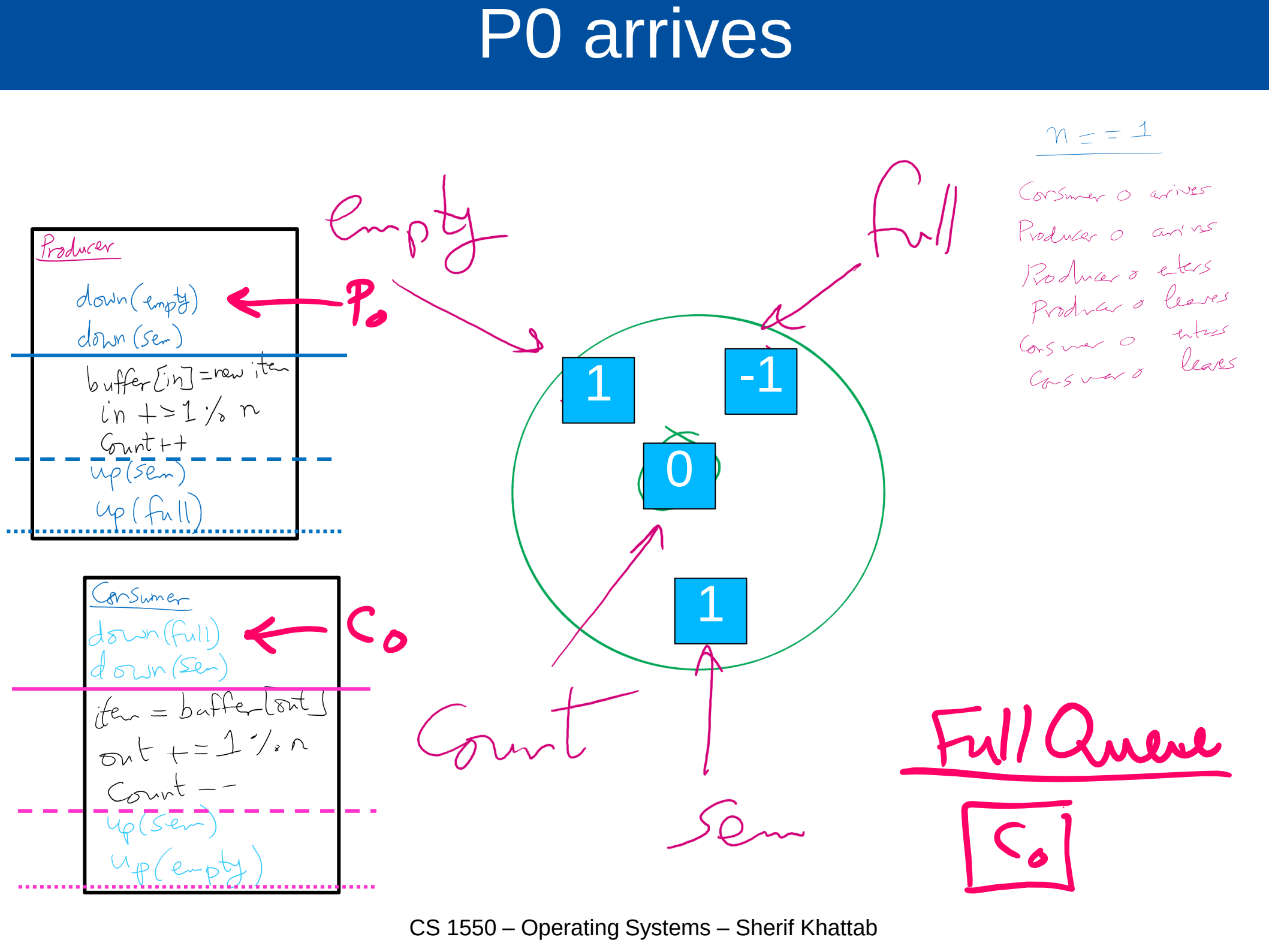
n == 1

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

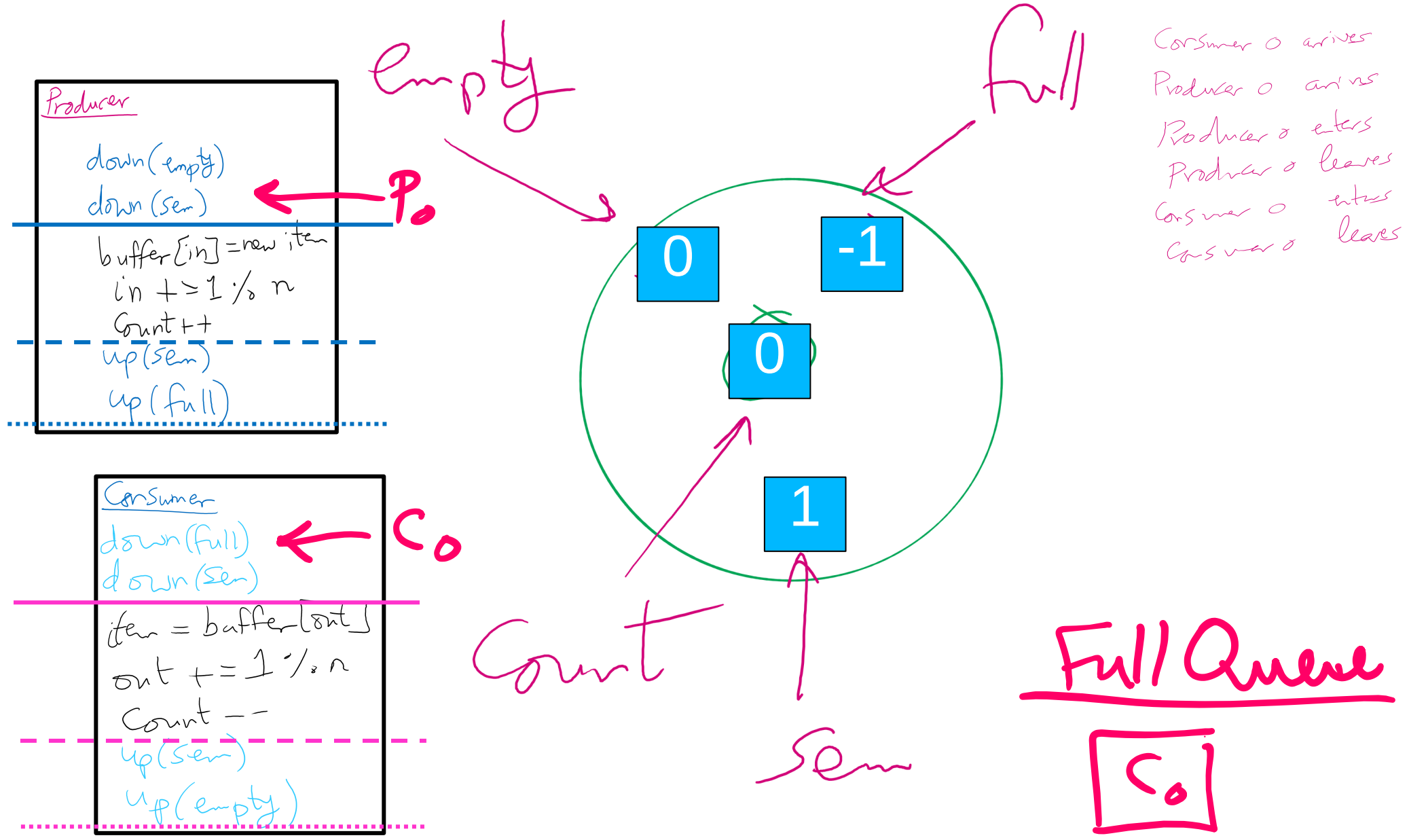
Full Queue

C0

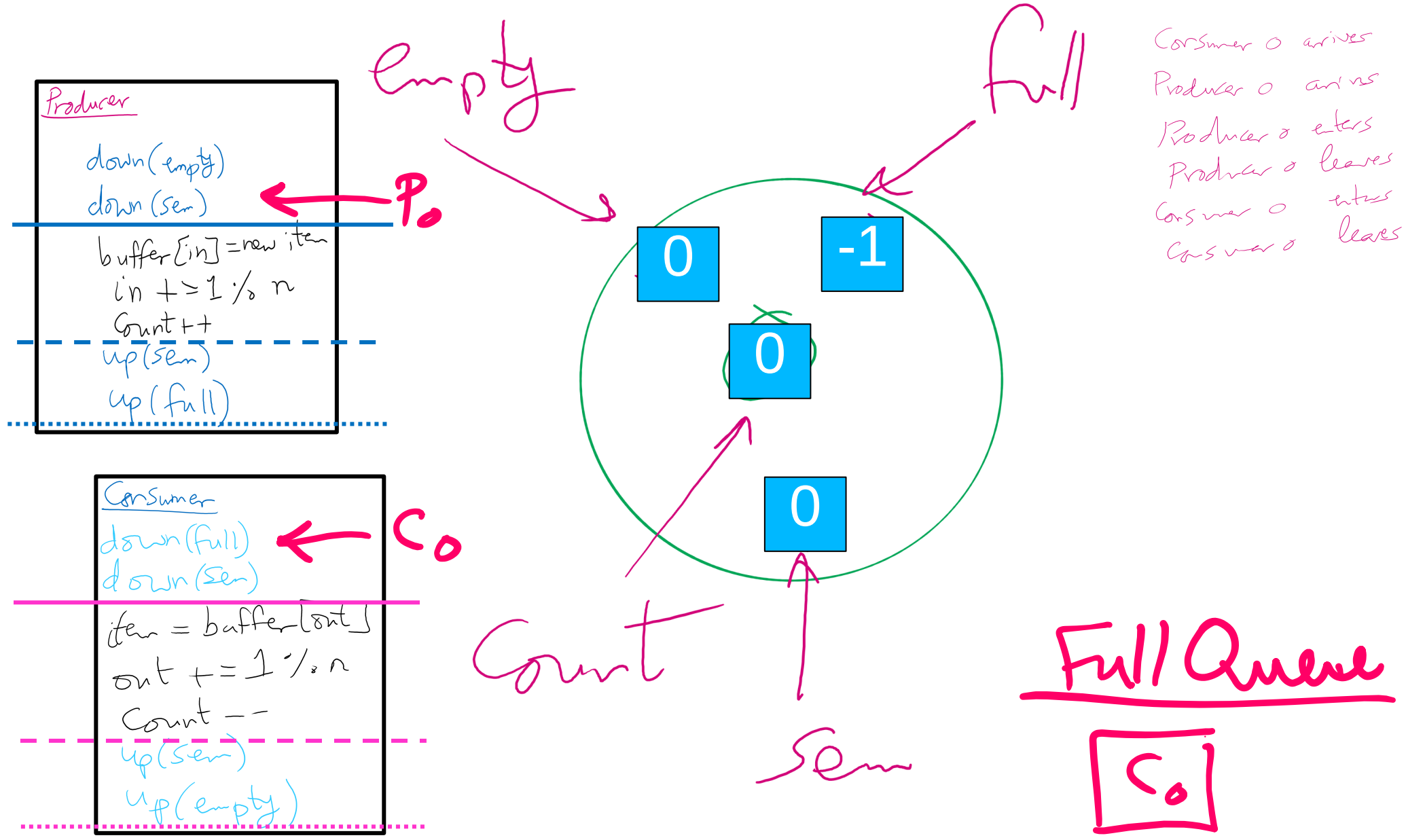
P0 arrives


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

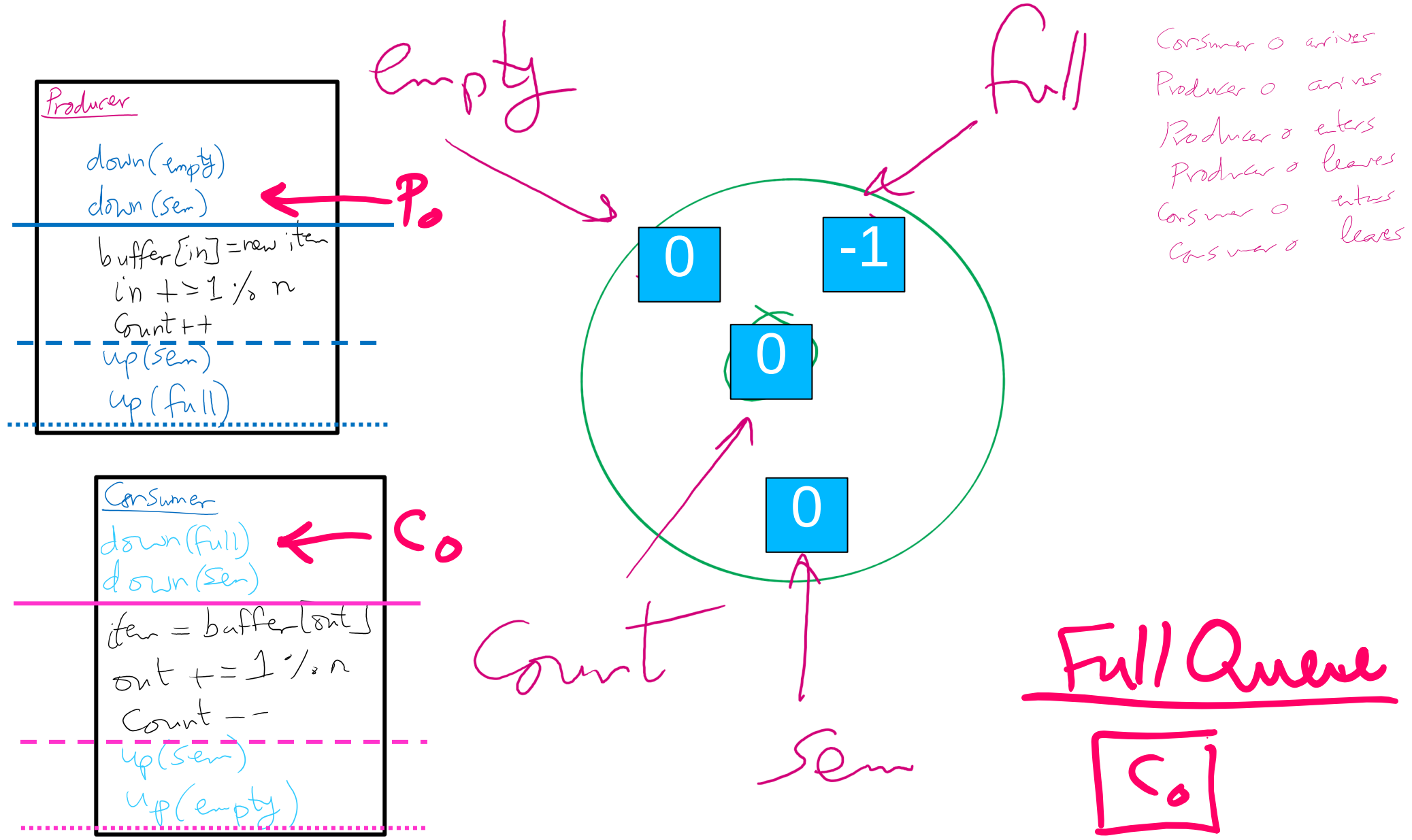
P0 arrives


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

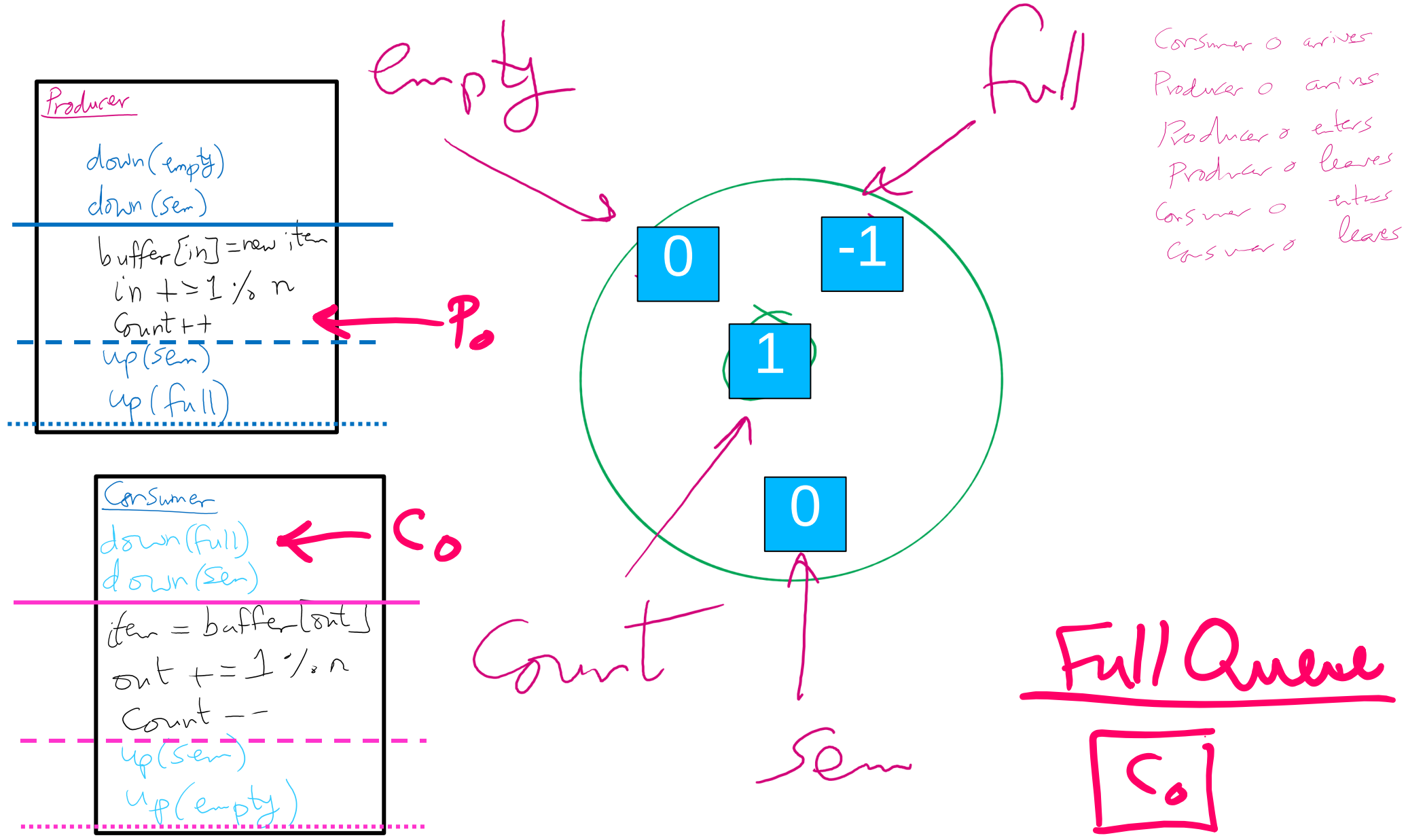
P0 arrives


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

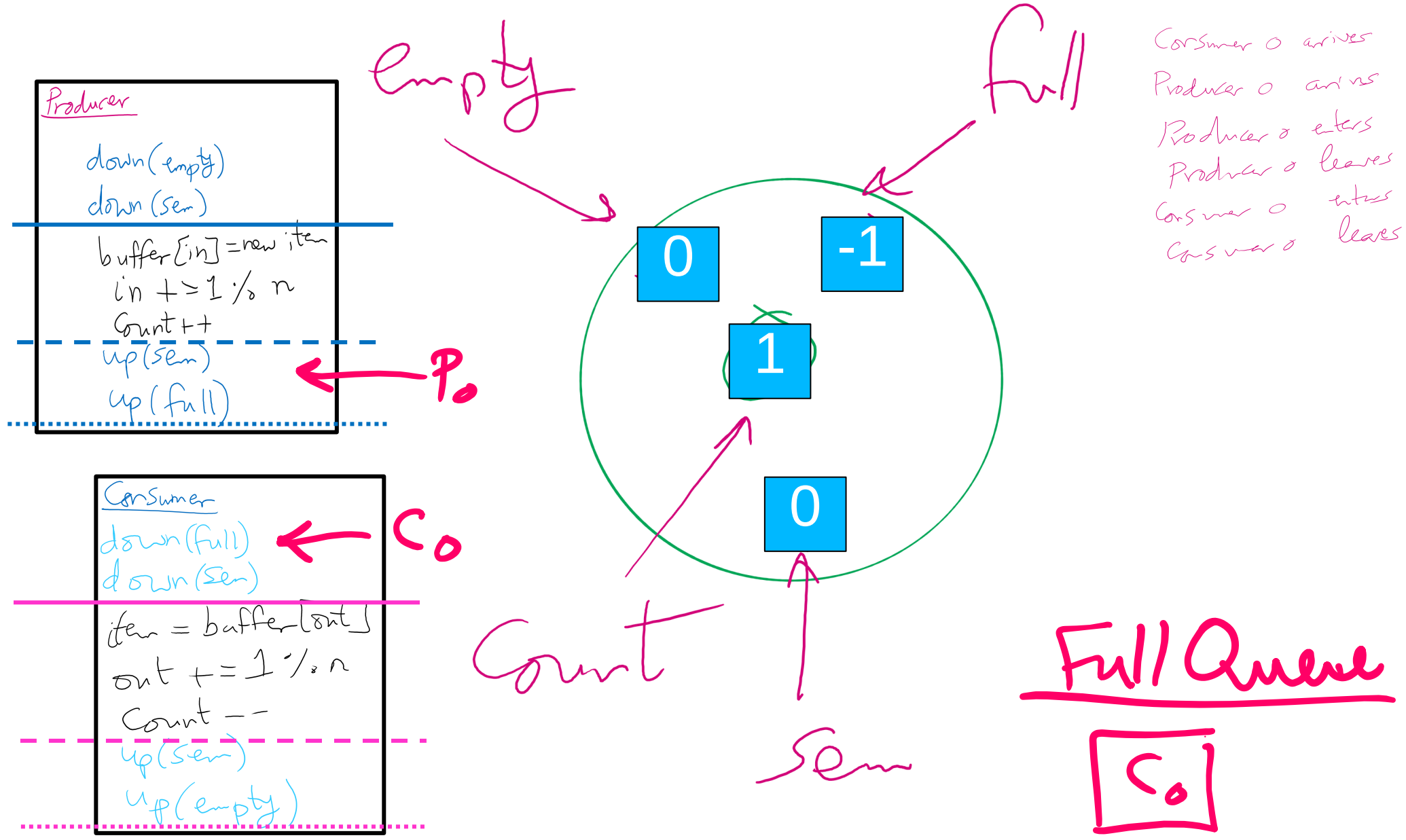
P0 enters


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

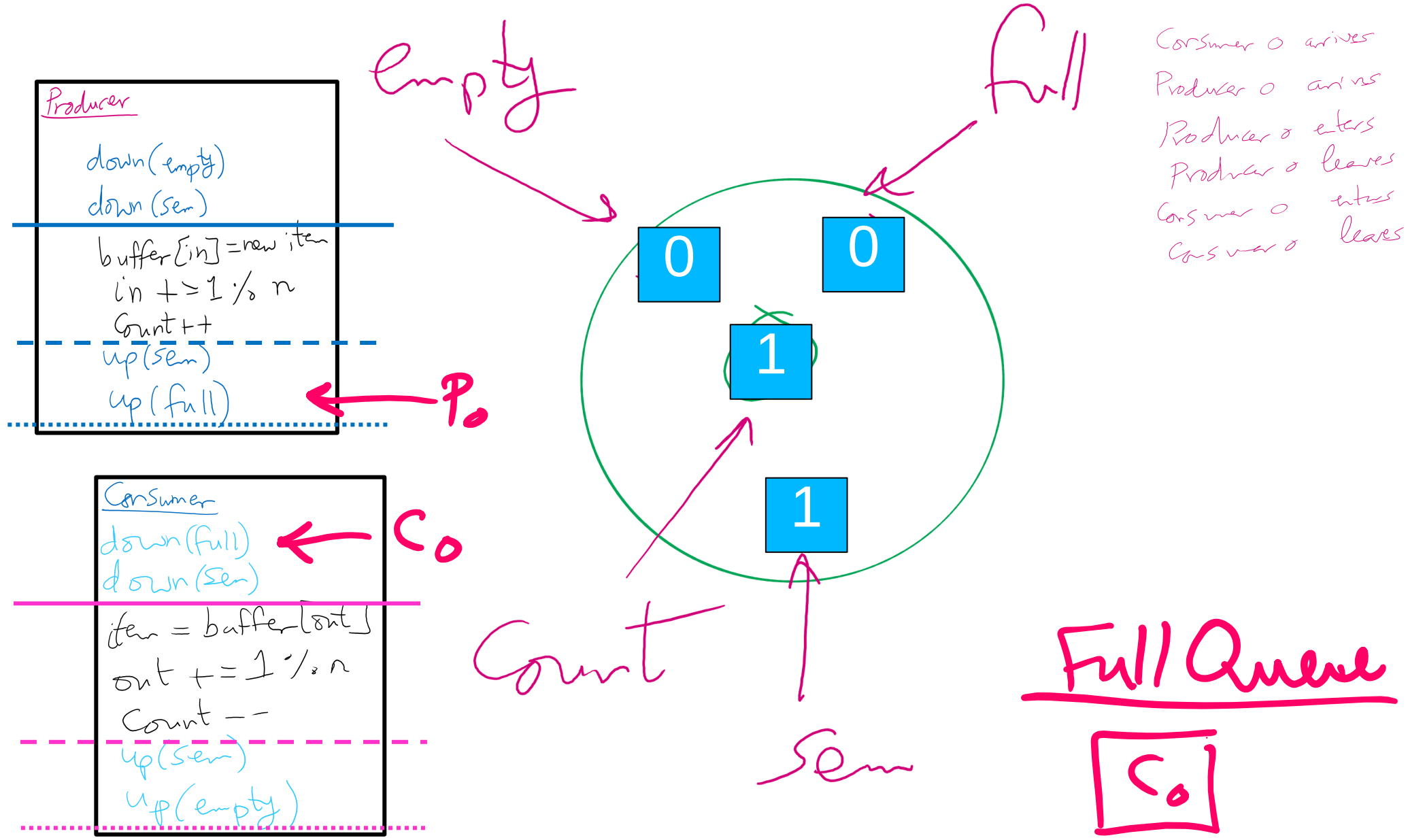
P0 enters


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

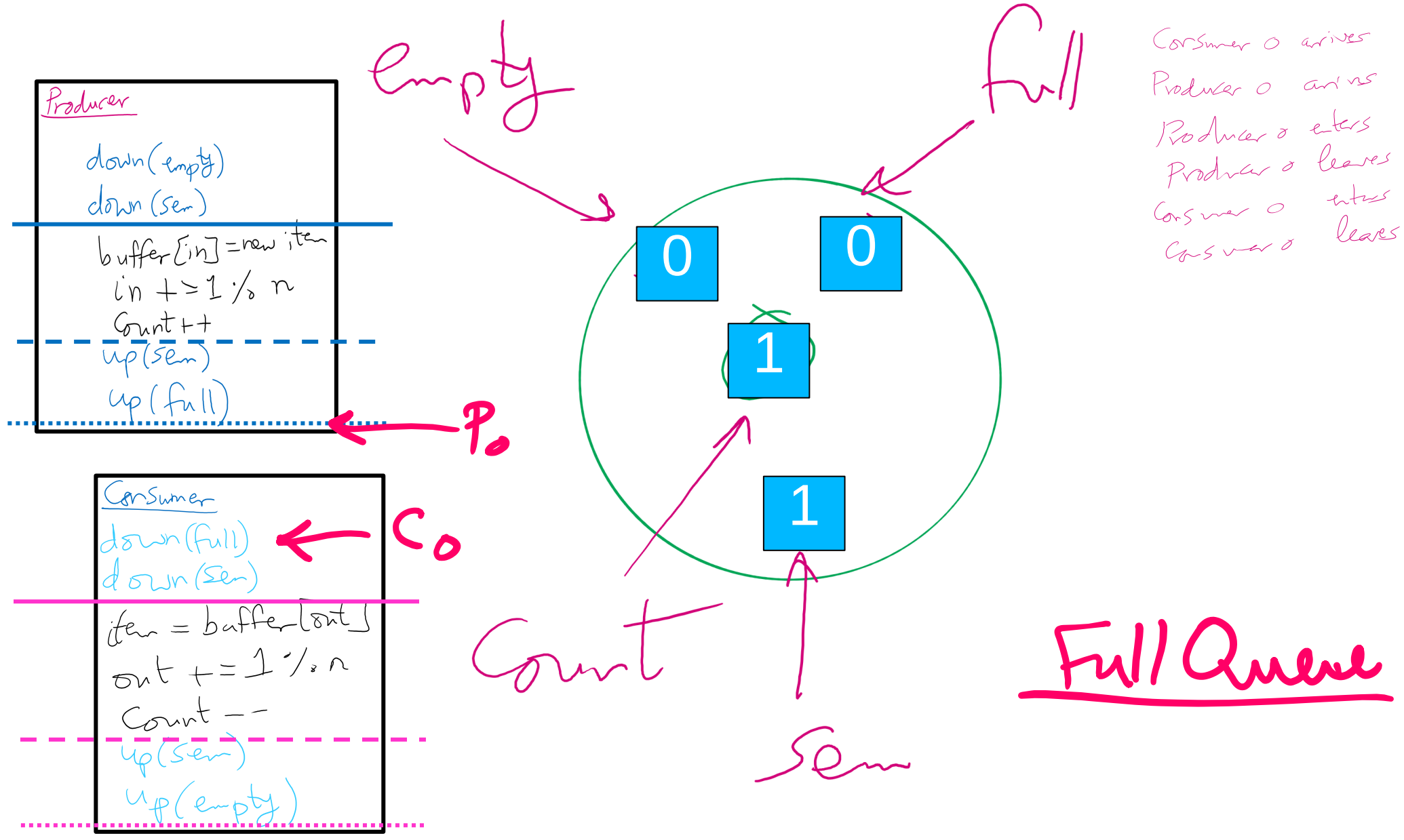
P0 leaves


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

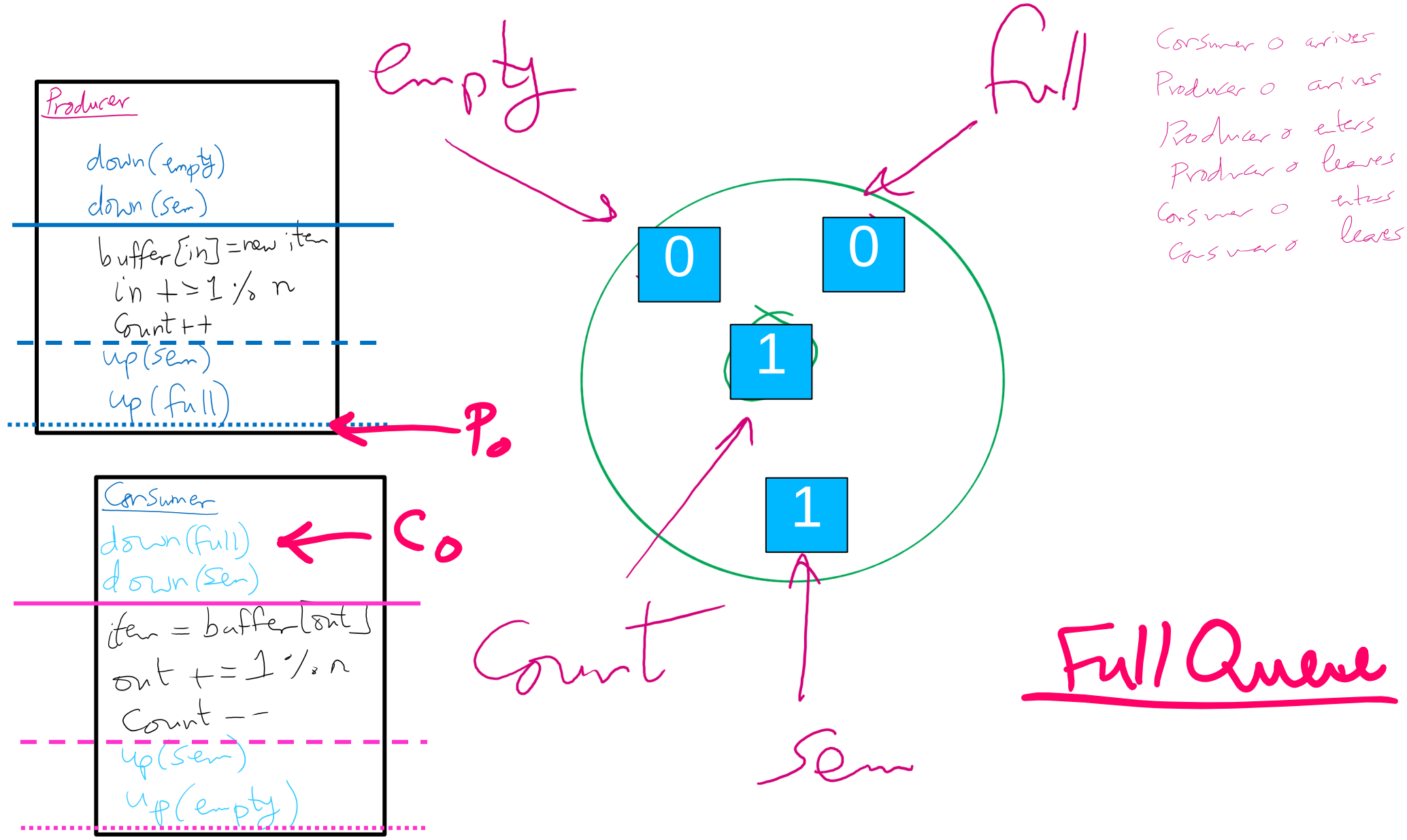
P0 leaves


n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
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

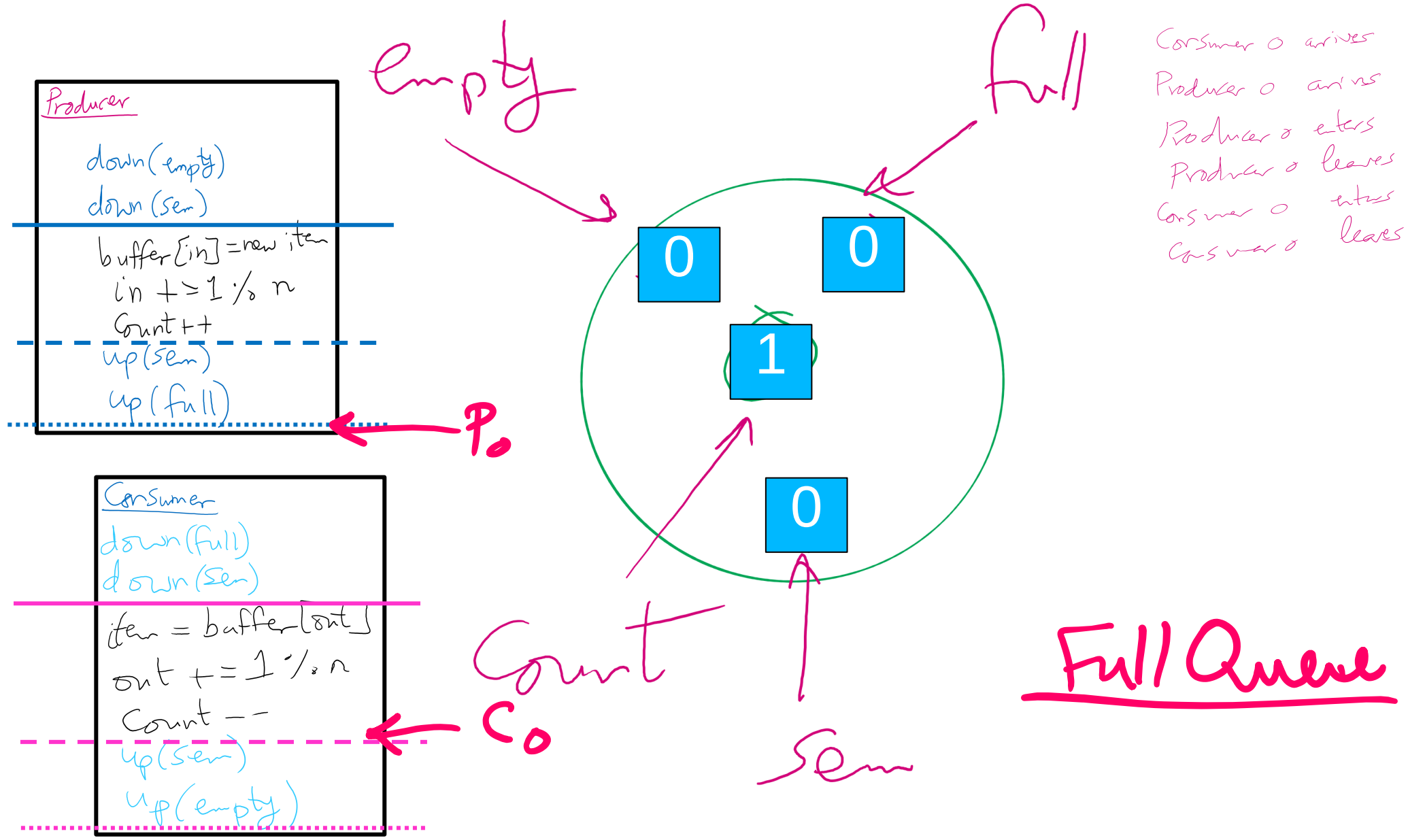
P0 leaves


n == 1
Consumer 0 arrives
Producer 0 arrives
Producer 0 enters
Producer 0 leaves
Consumer 0 enters
Consumer 0 leaves
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

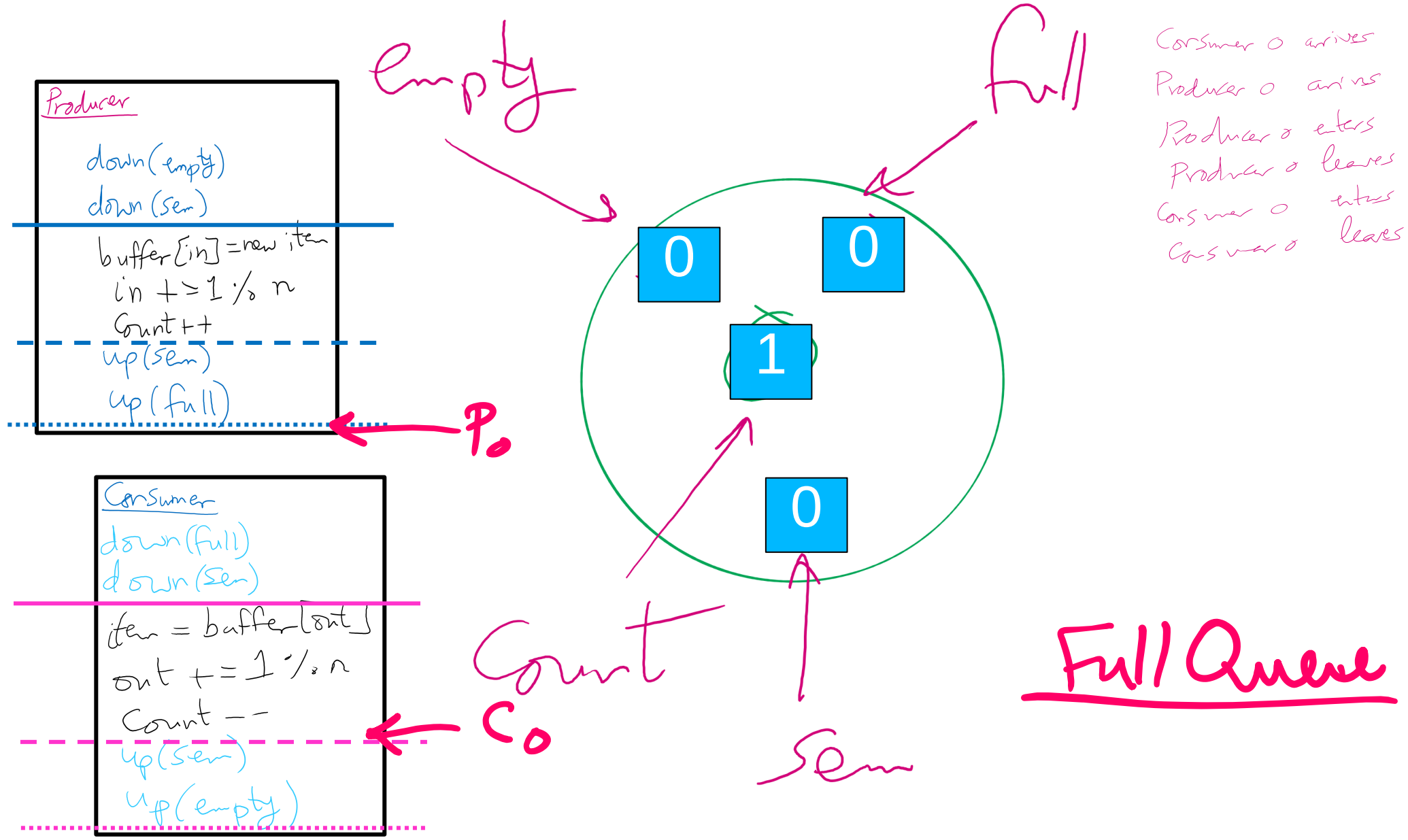
C0 enters


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

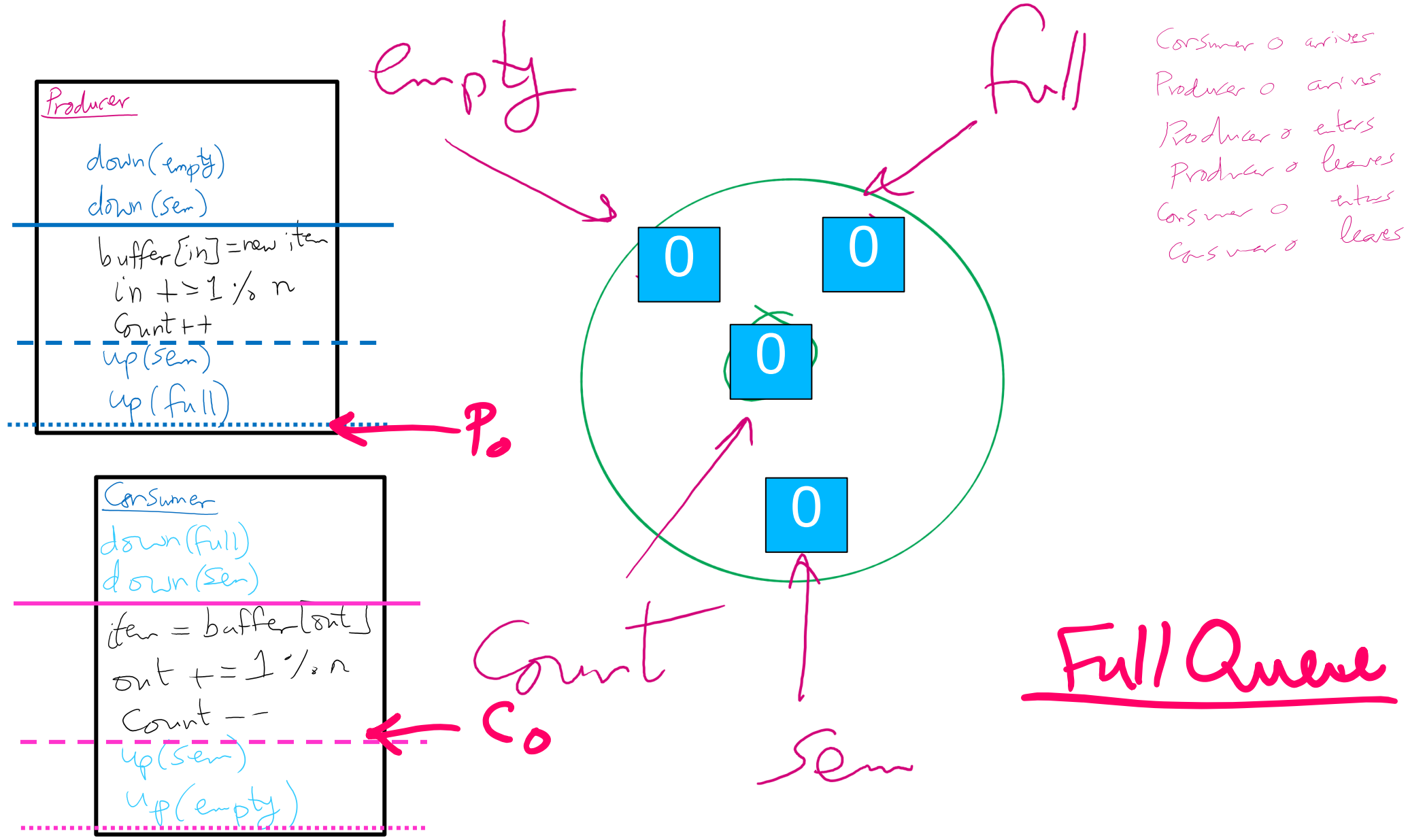
C0 enters


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
0
count
sem
Full Queue

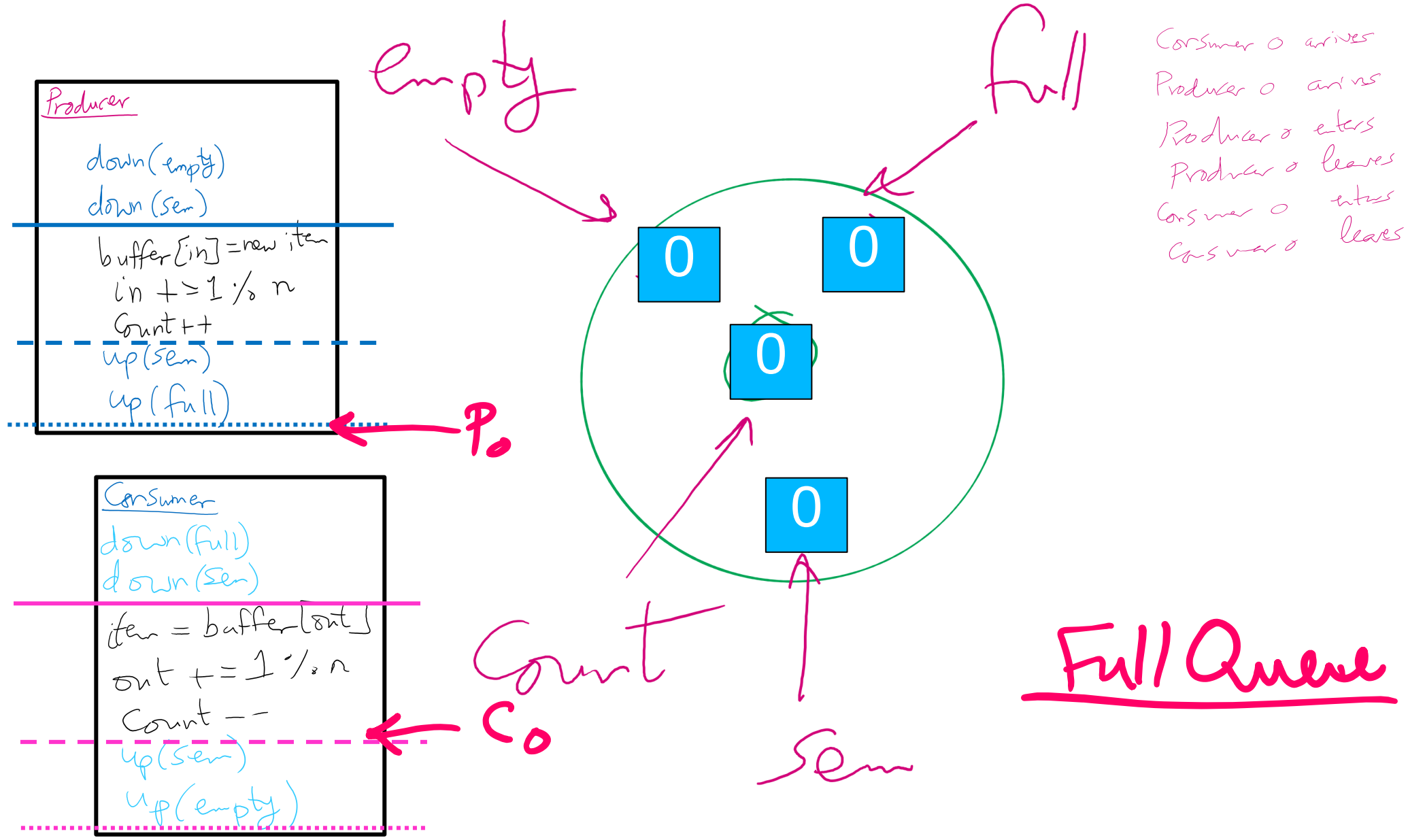
Can C0 enter?


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

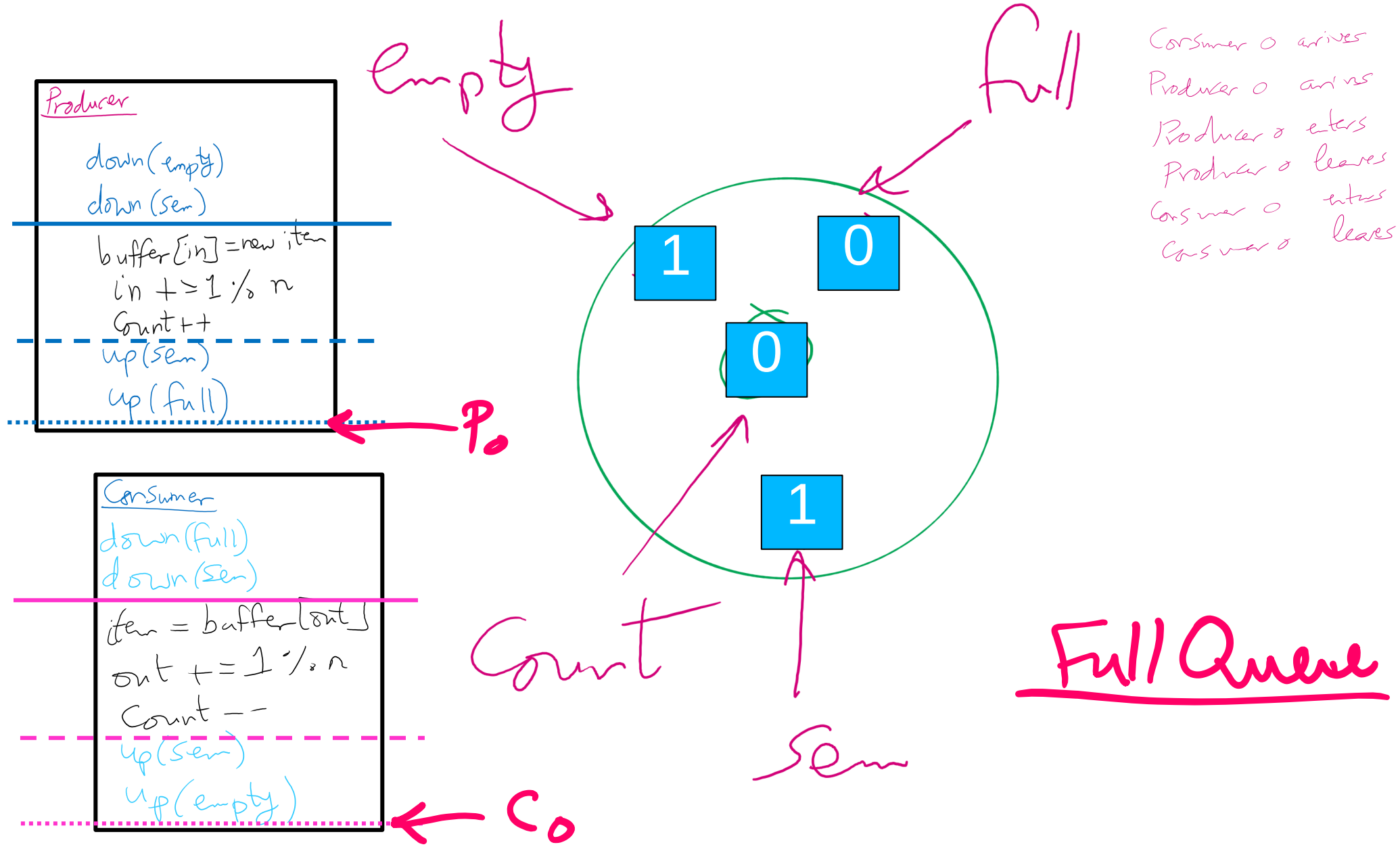
C0 enters


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

Example 3

$n == 2$

C_0 arrives

C_1 arrives

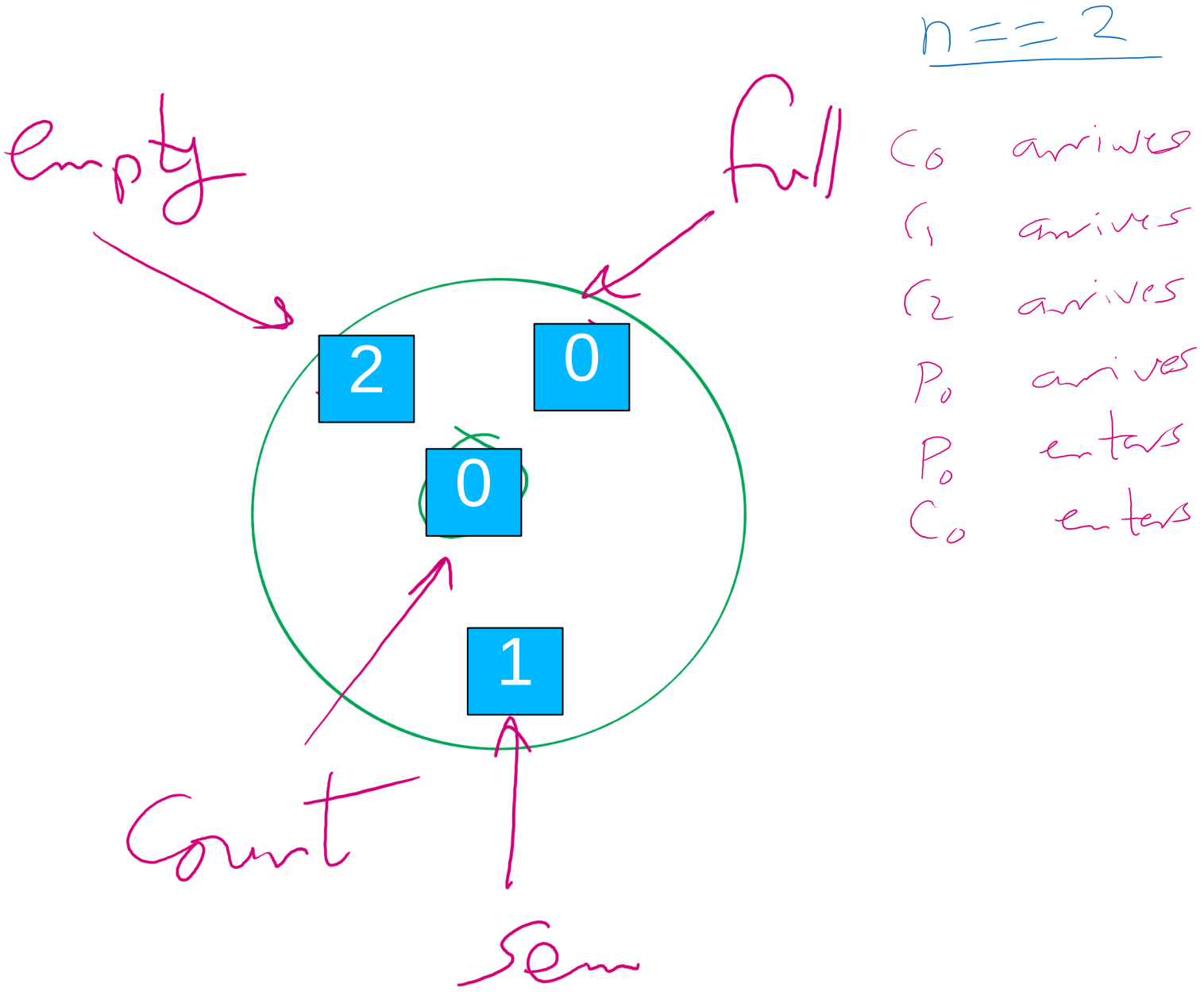
C_2 arrives

P_0 arrives

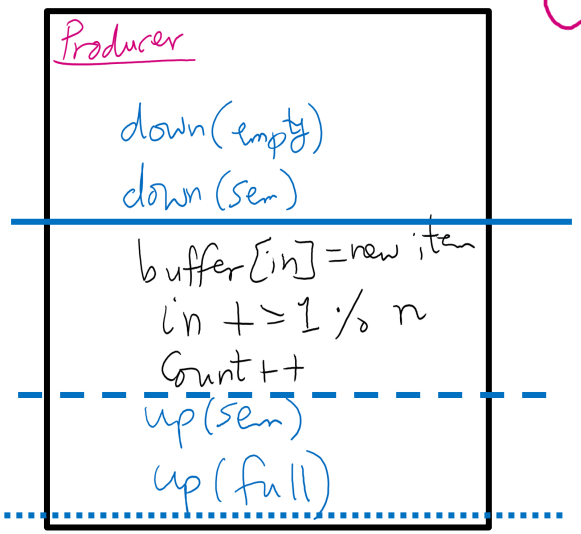
P_0 enters

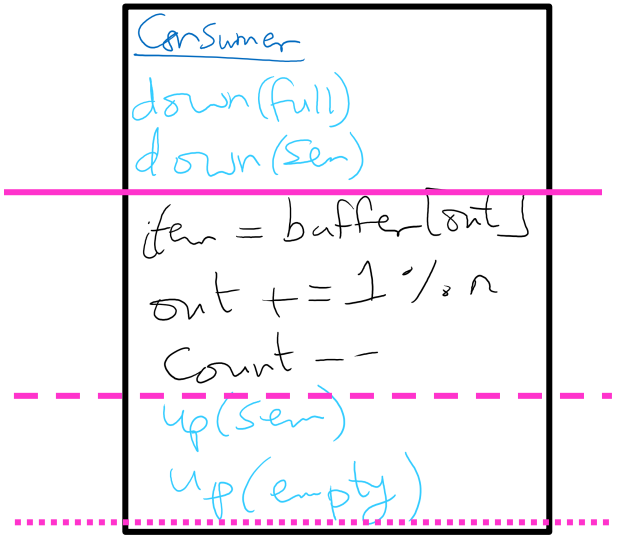
C_0 enters

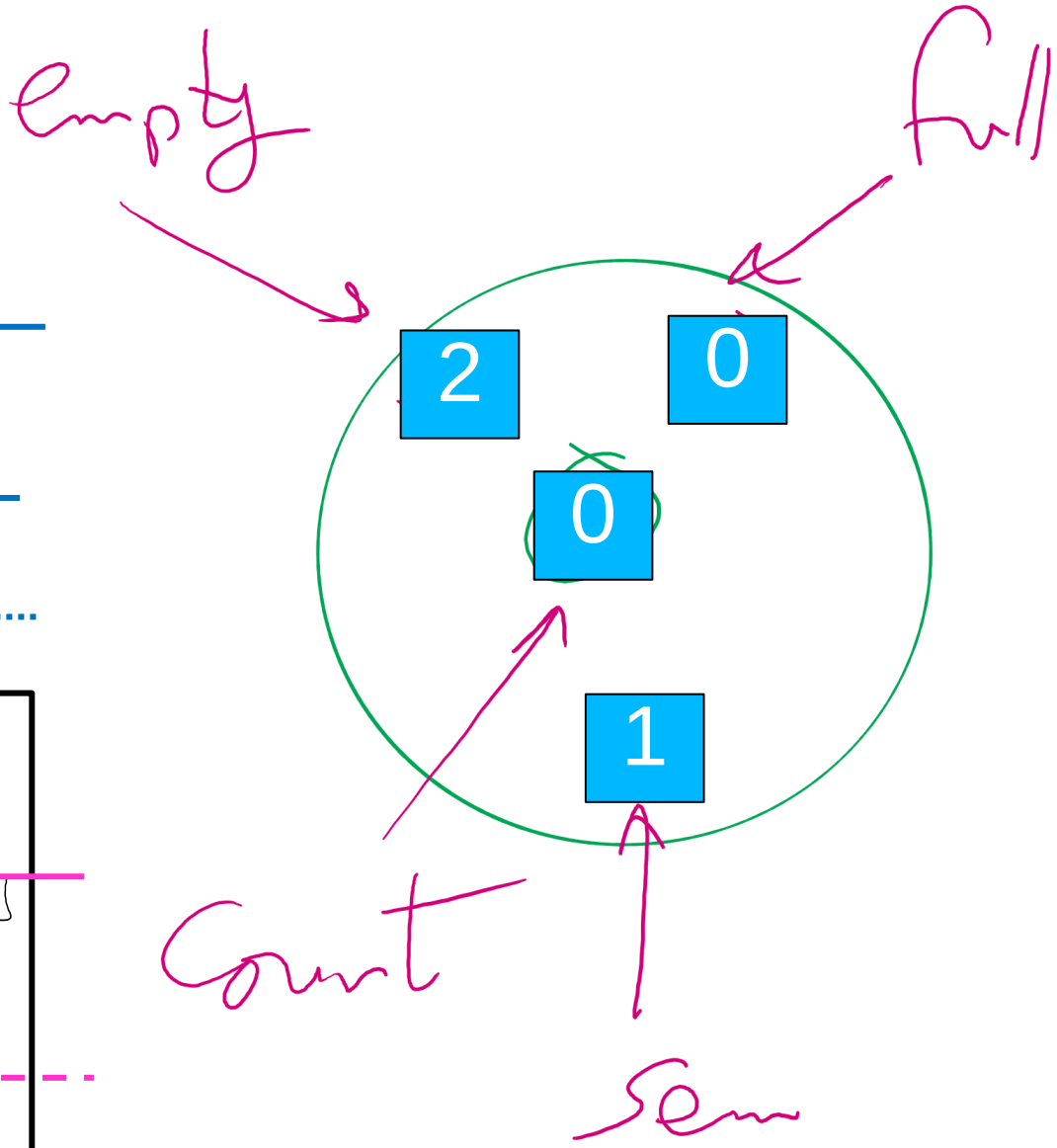
Initial state



C0 arrives



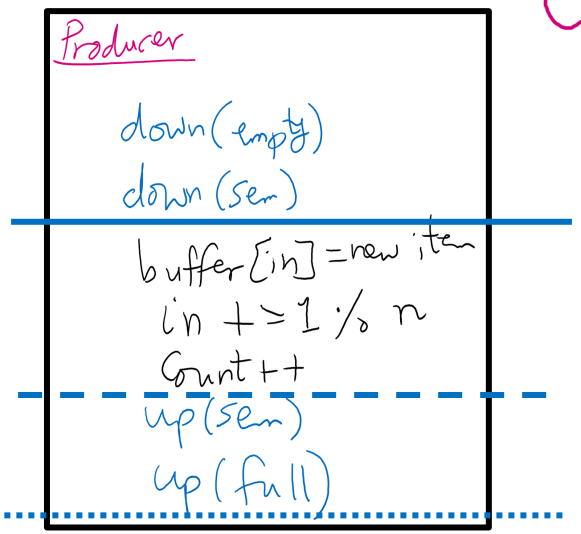


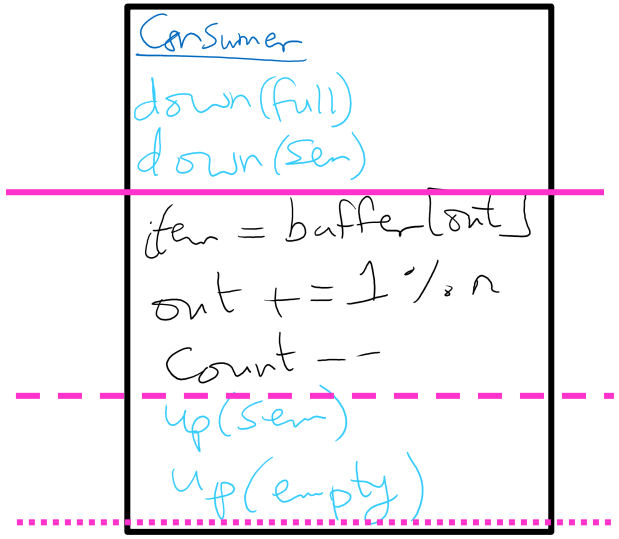


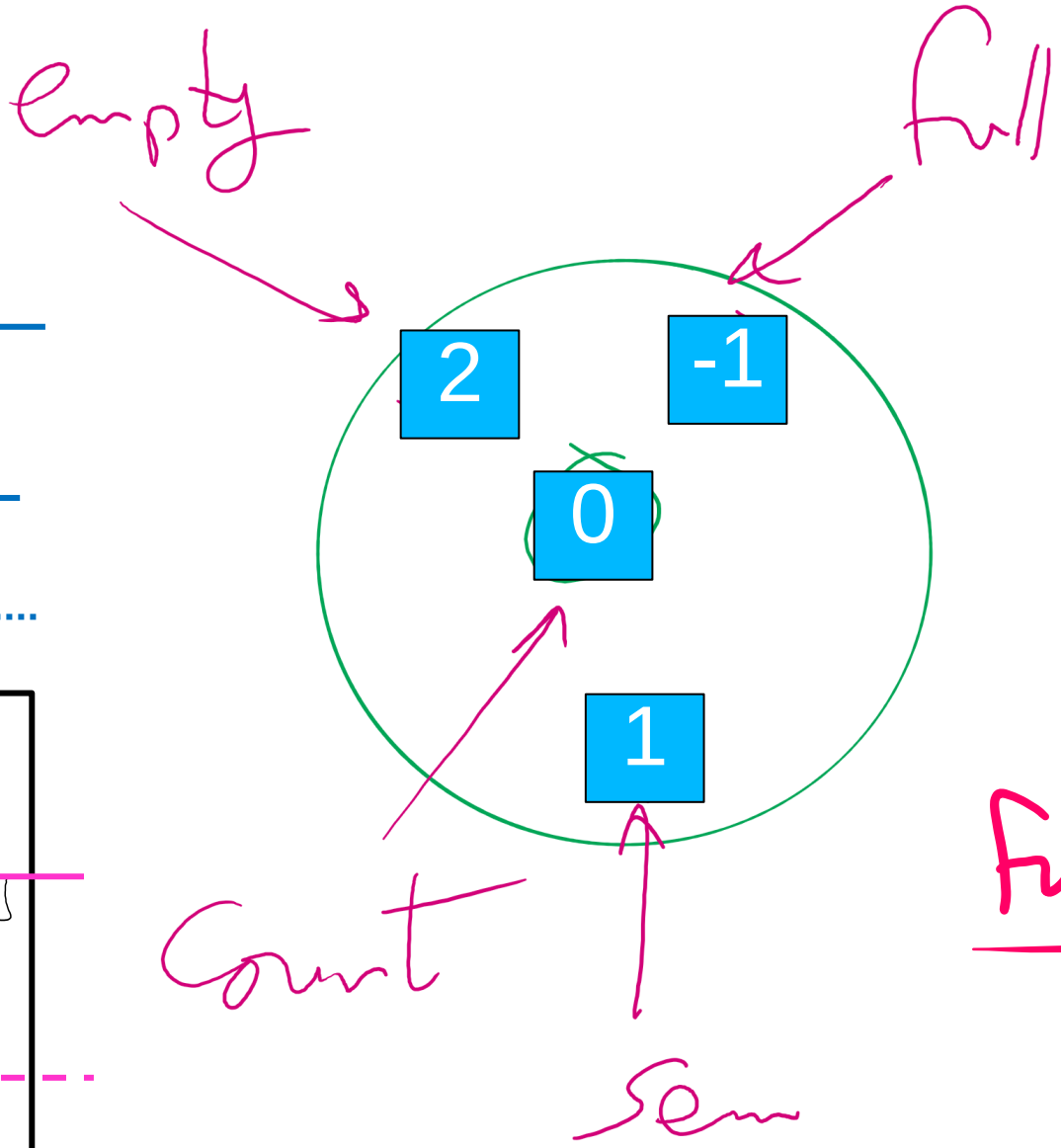
n == 2

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

C0 arrives



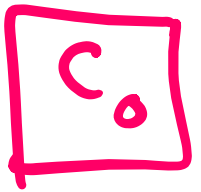




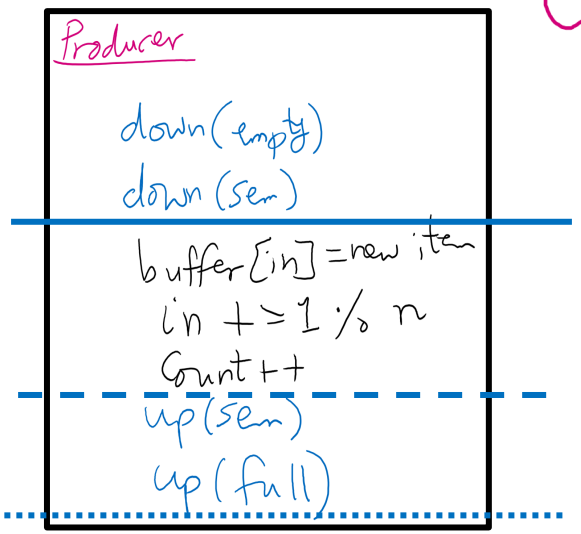
n == 2

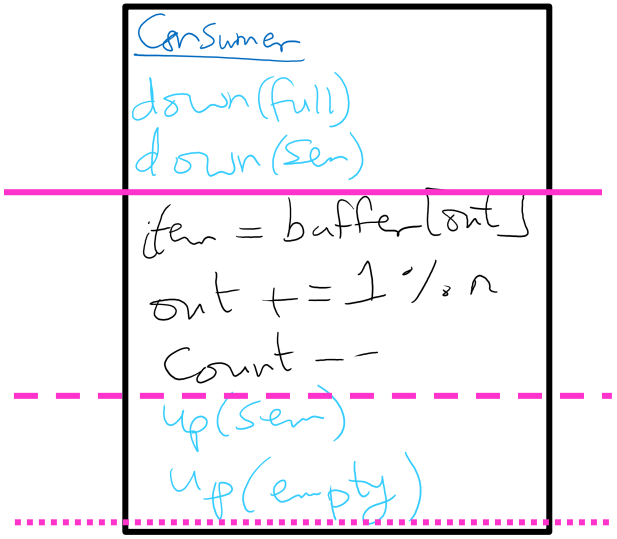
C0 arrives
C1 arrives
C2 arrives
P0 arrives
P0 enters
C0 enters

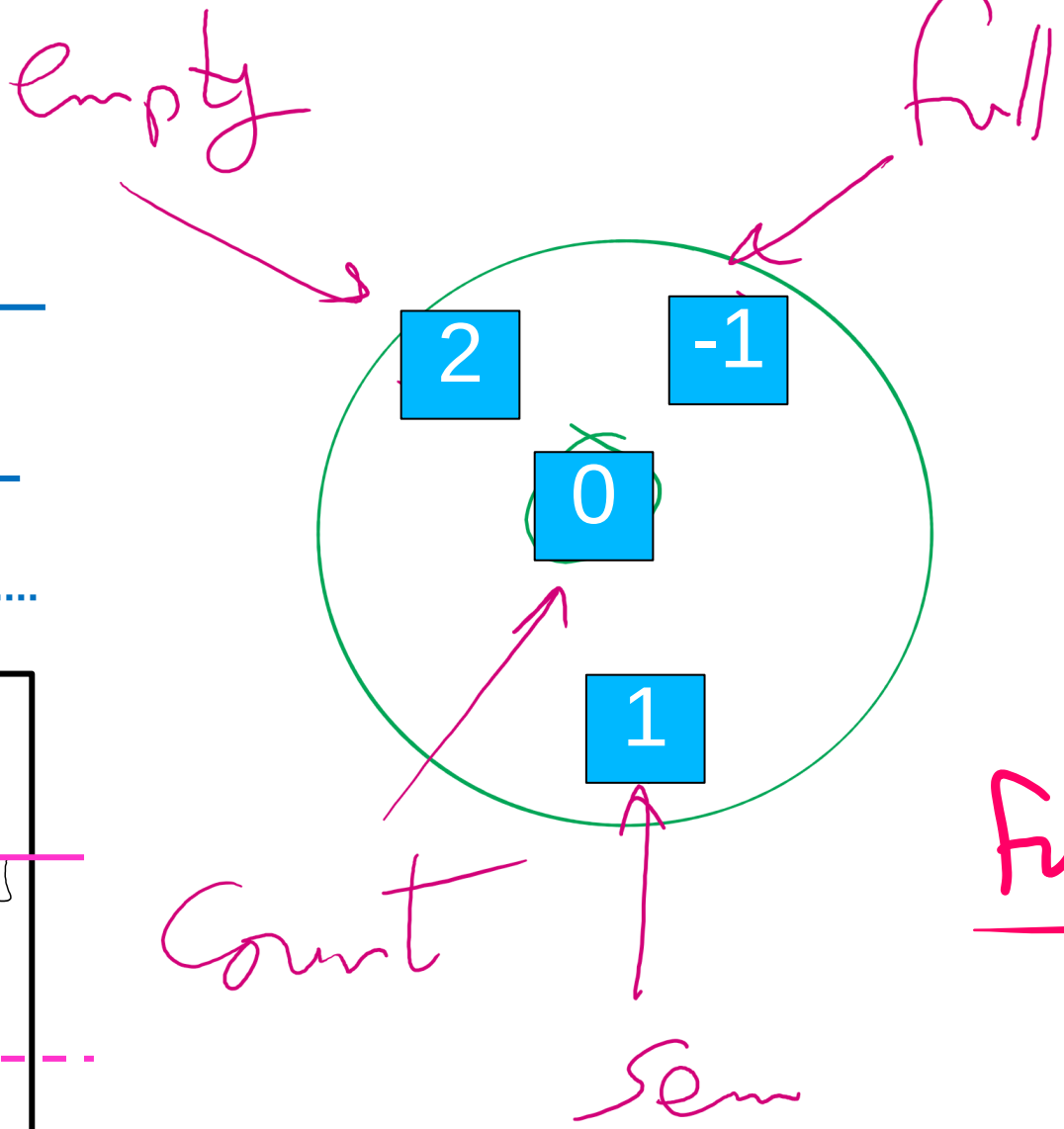
Full Queue



C1 arrives



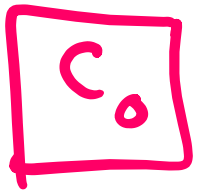




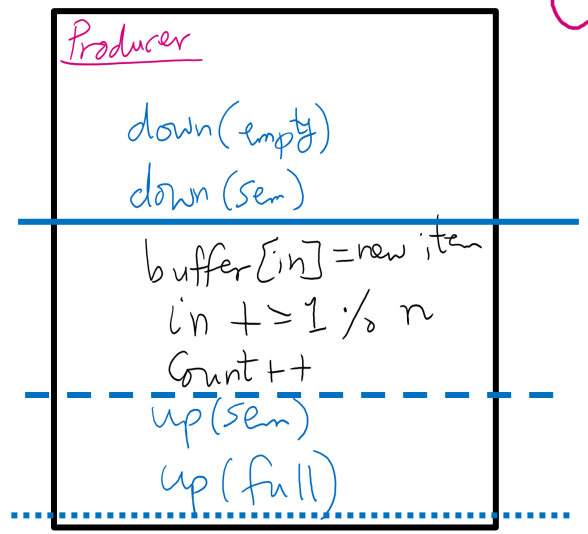
n == 2

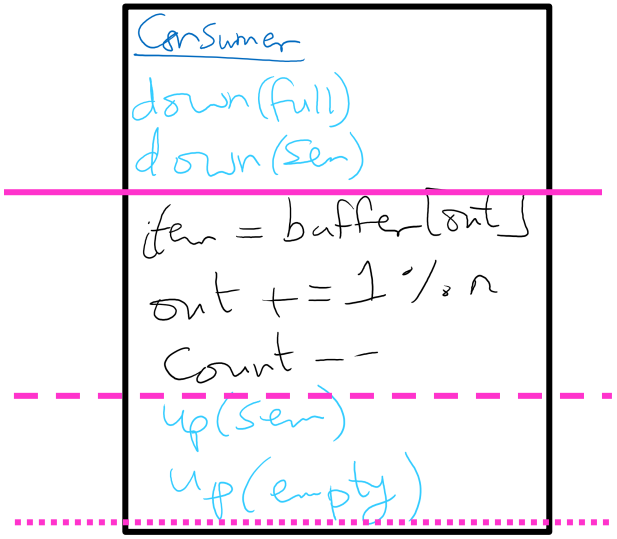
C_0 arrives
C_1 arrives
C_2 arrives
P_0 arrives
P_0 enters
C_0 enters

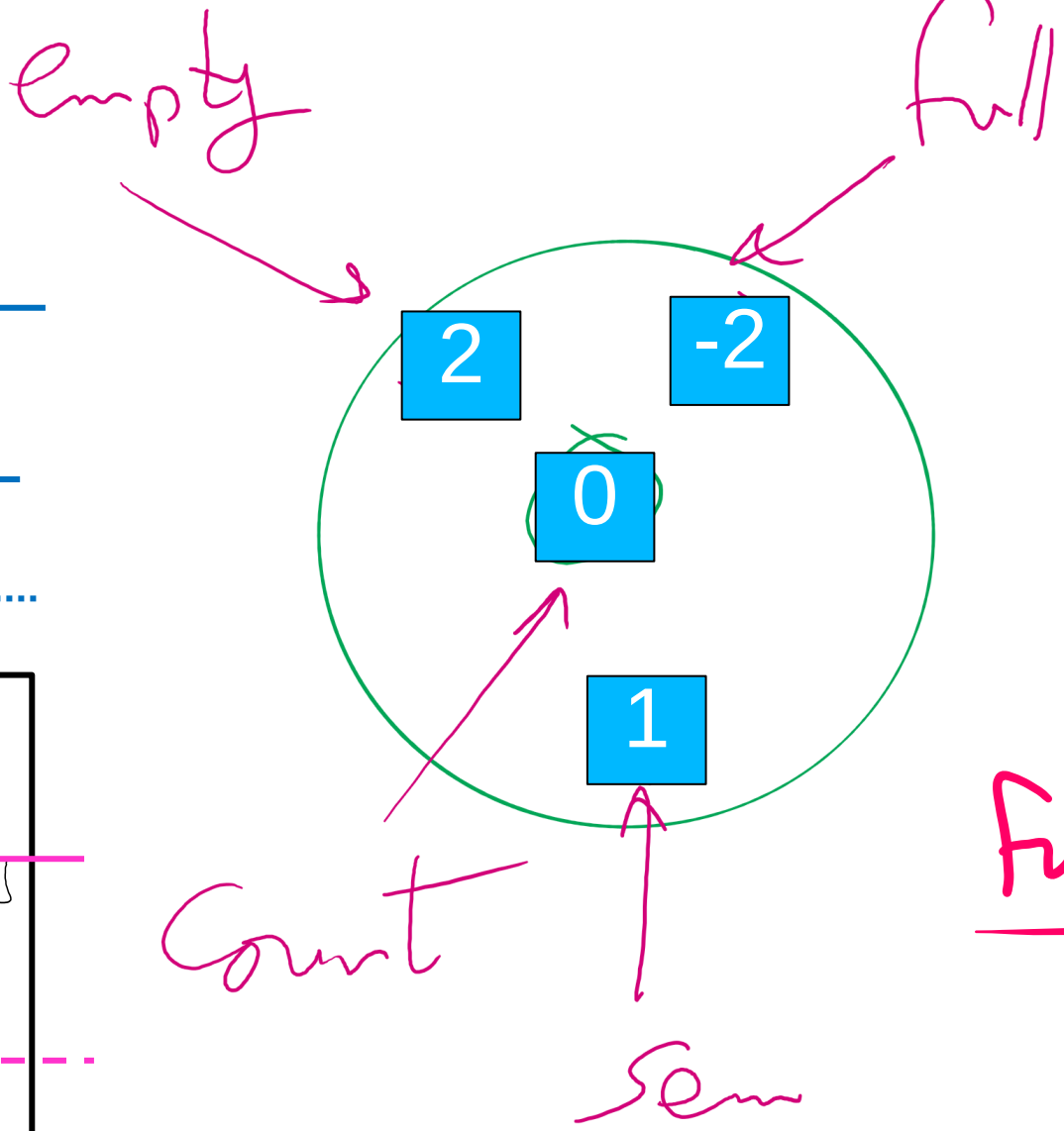
Full Queue



C1 arrives



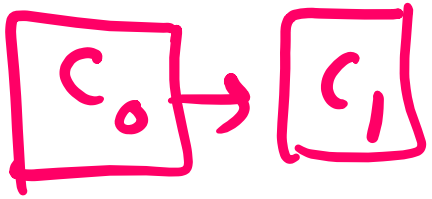




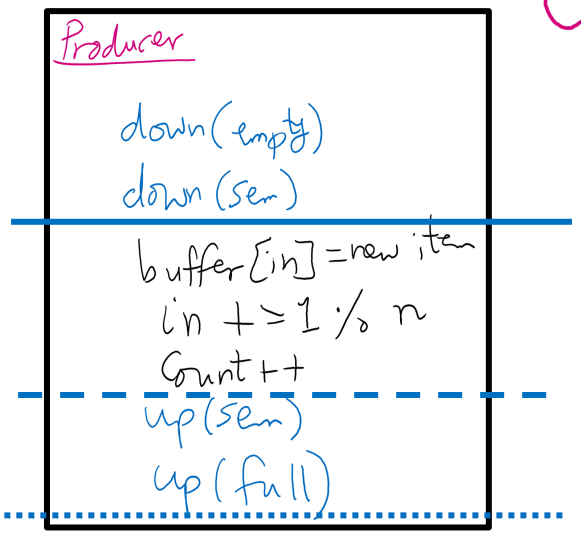
n == 2

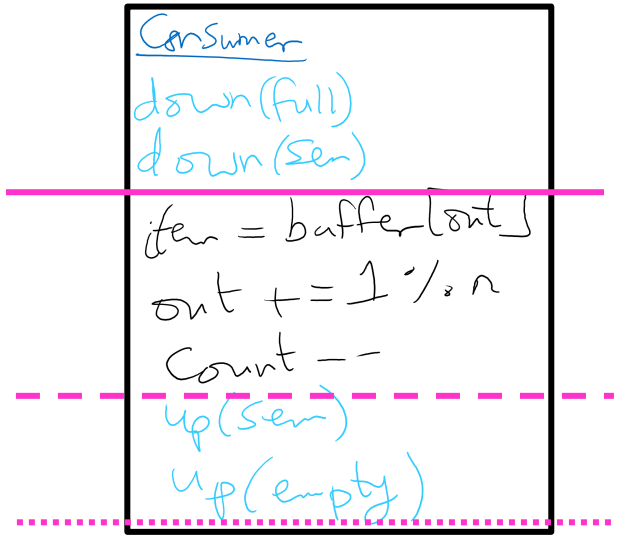
C_0 arrives
C_1 arrives
C_2 arrives
P_0 arrives
P_0 enters
C_0 enters

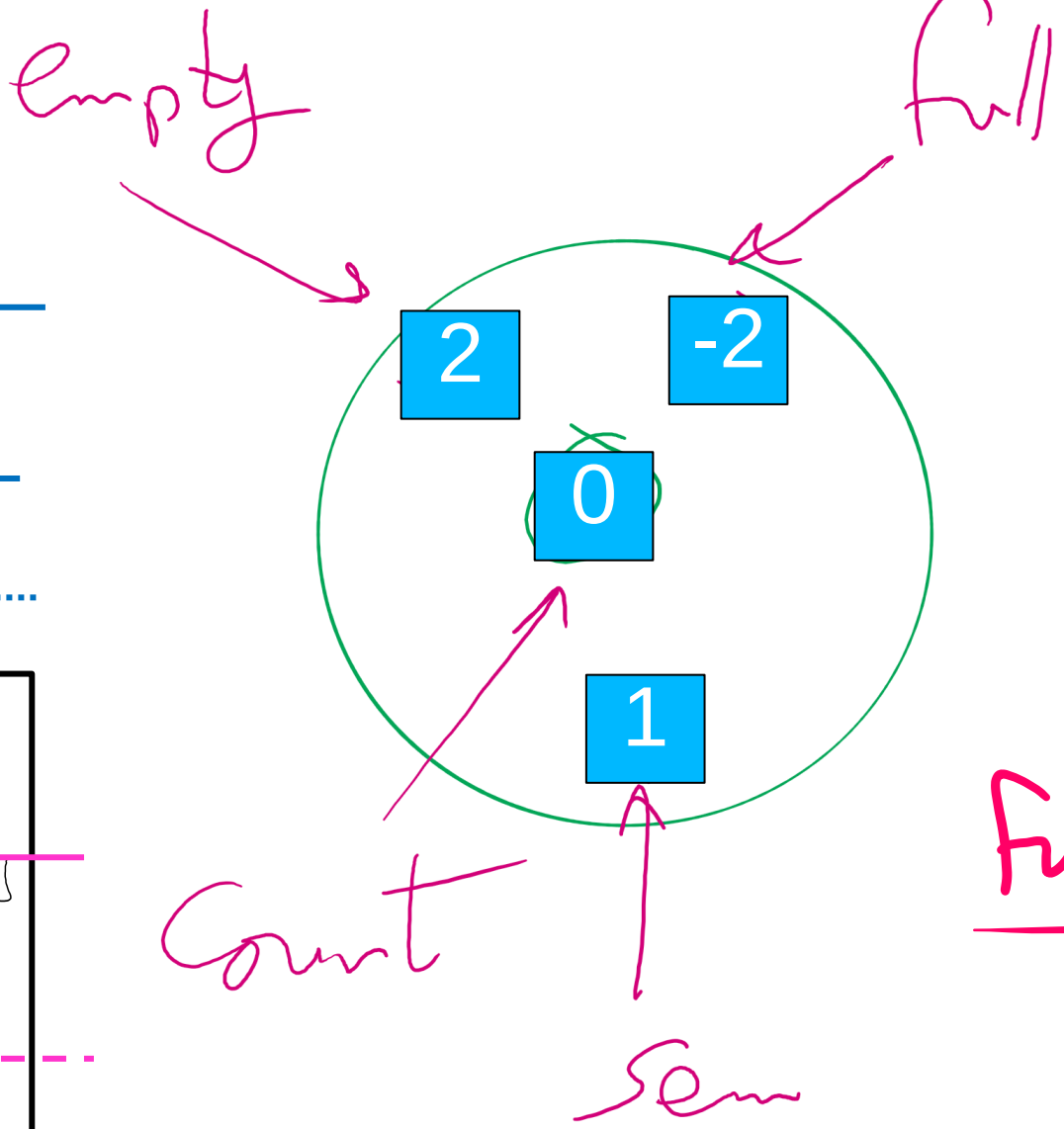
Full Queue



C2 arrives



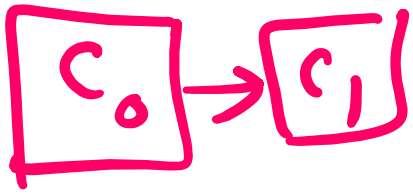




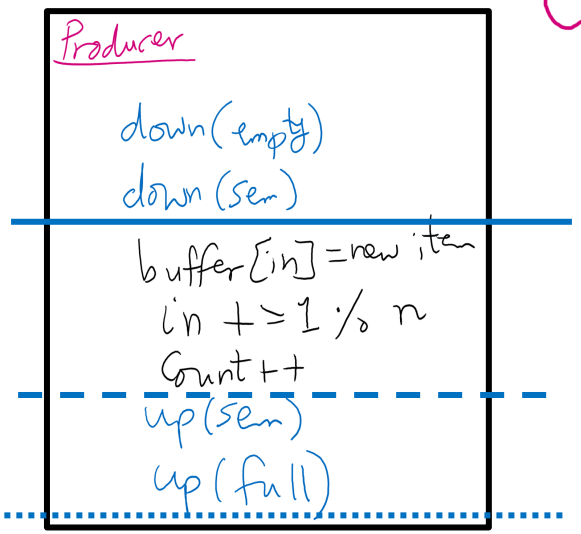
n == 2

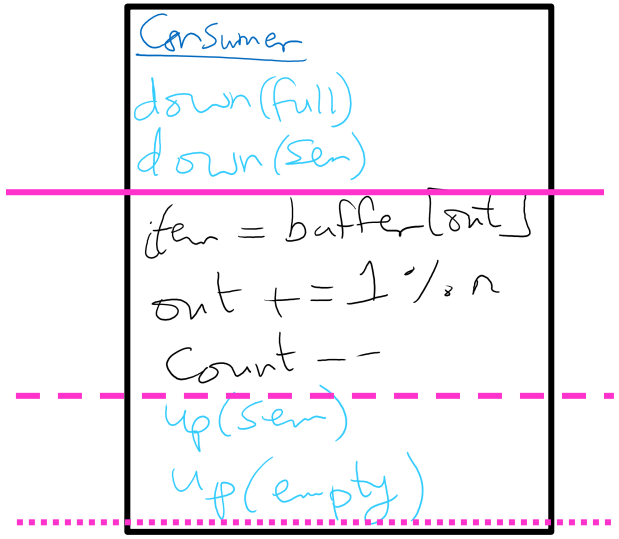
C_0 arrives
C_1 arrives
C_2 arrives
P_0 arrives
P_0 enters
C_0 enters

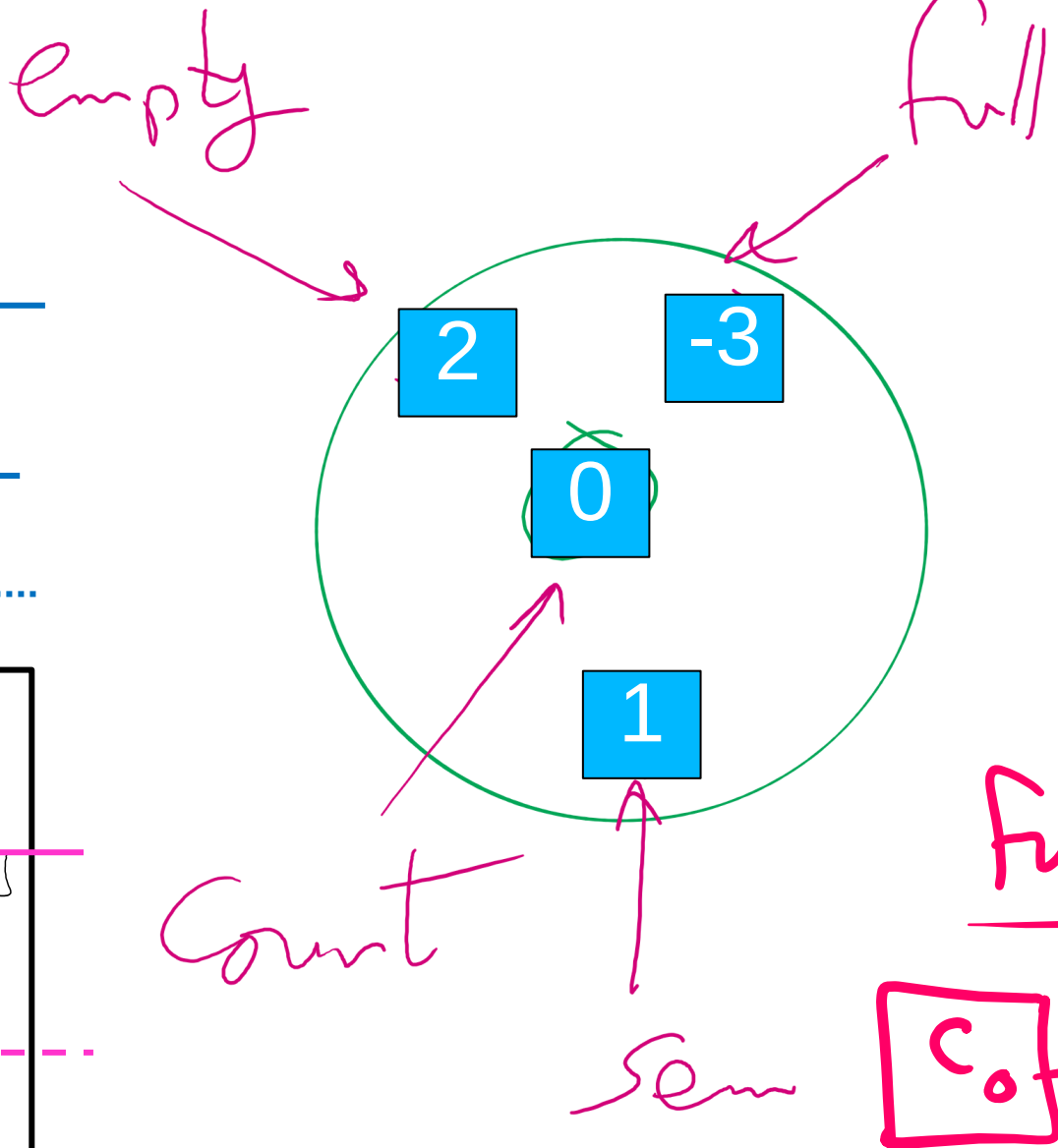
Full Queue



C2 arrives



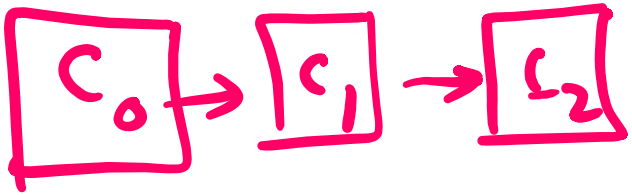




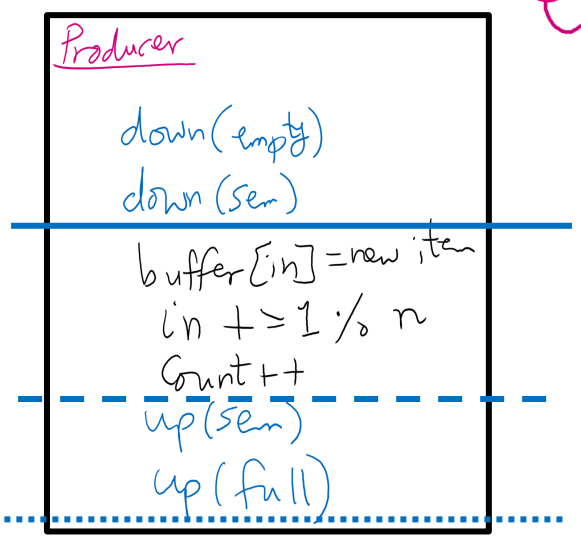
n == 2

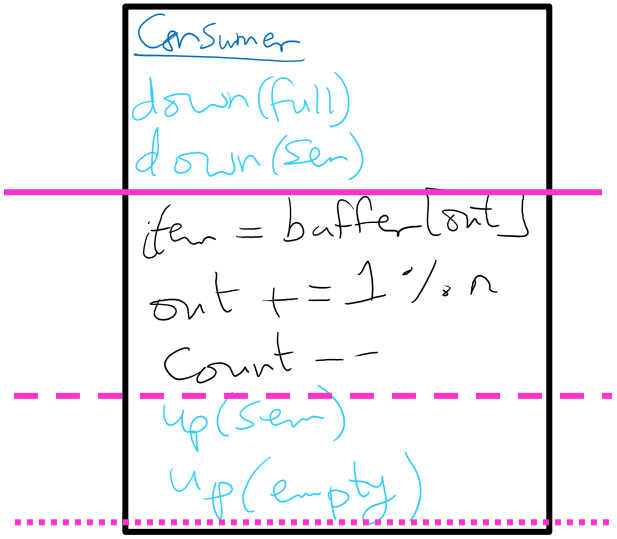
C_0 arrives
C_1 arrives
C_2 arrives
P_0 arrives
P_0 enters
C_0 enters

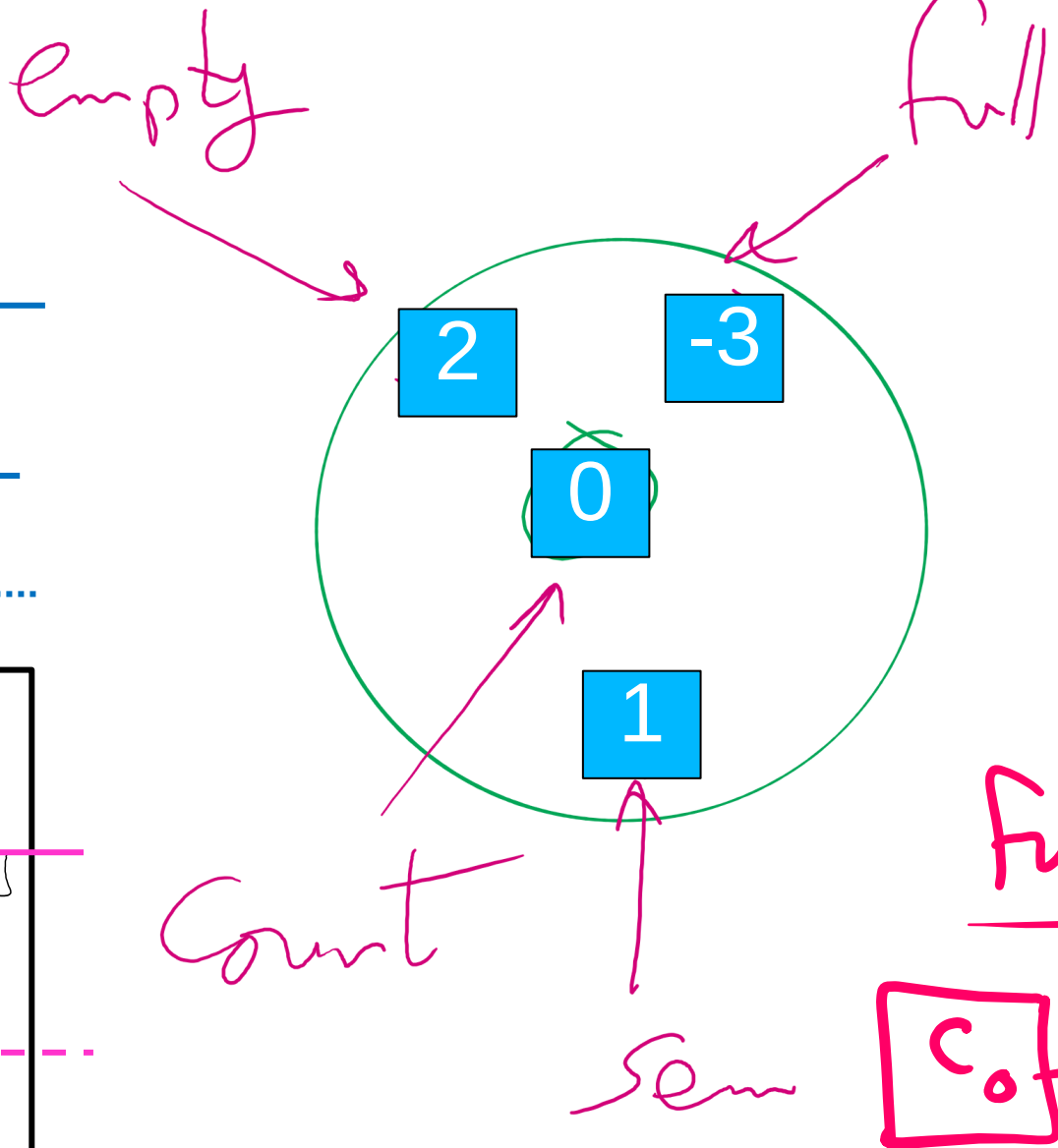
Full Queue



P0 arrives



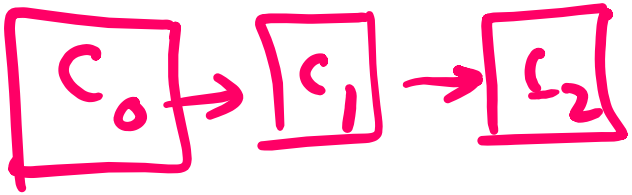




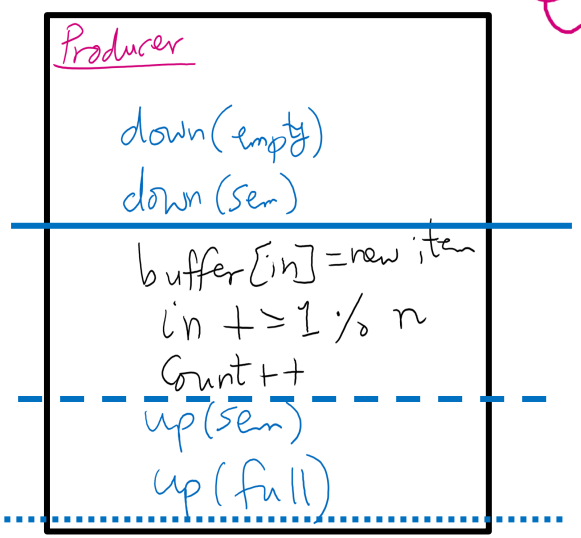
n == 2

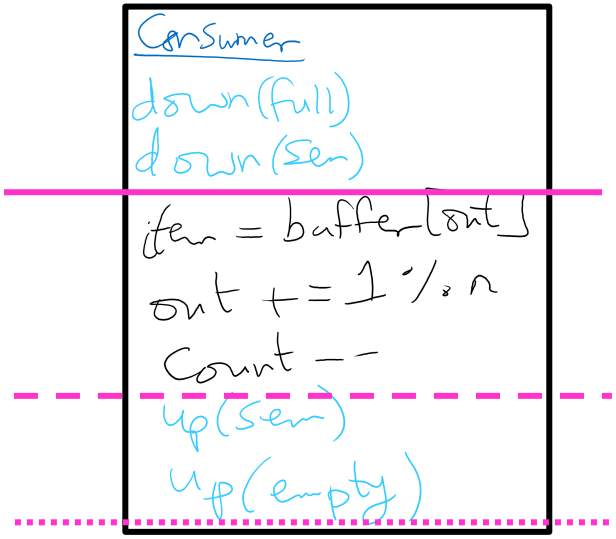
C_0 arrives
C_1 arrives
C_2 arrives
P_0 arrives
P_0 enters
C_0 enters

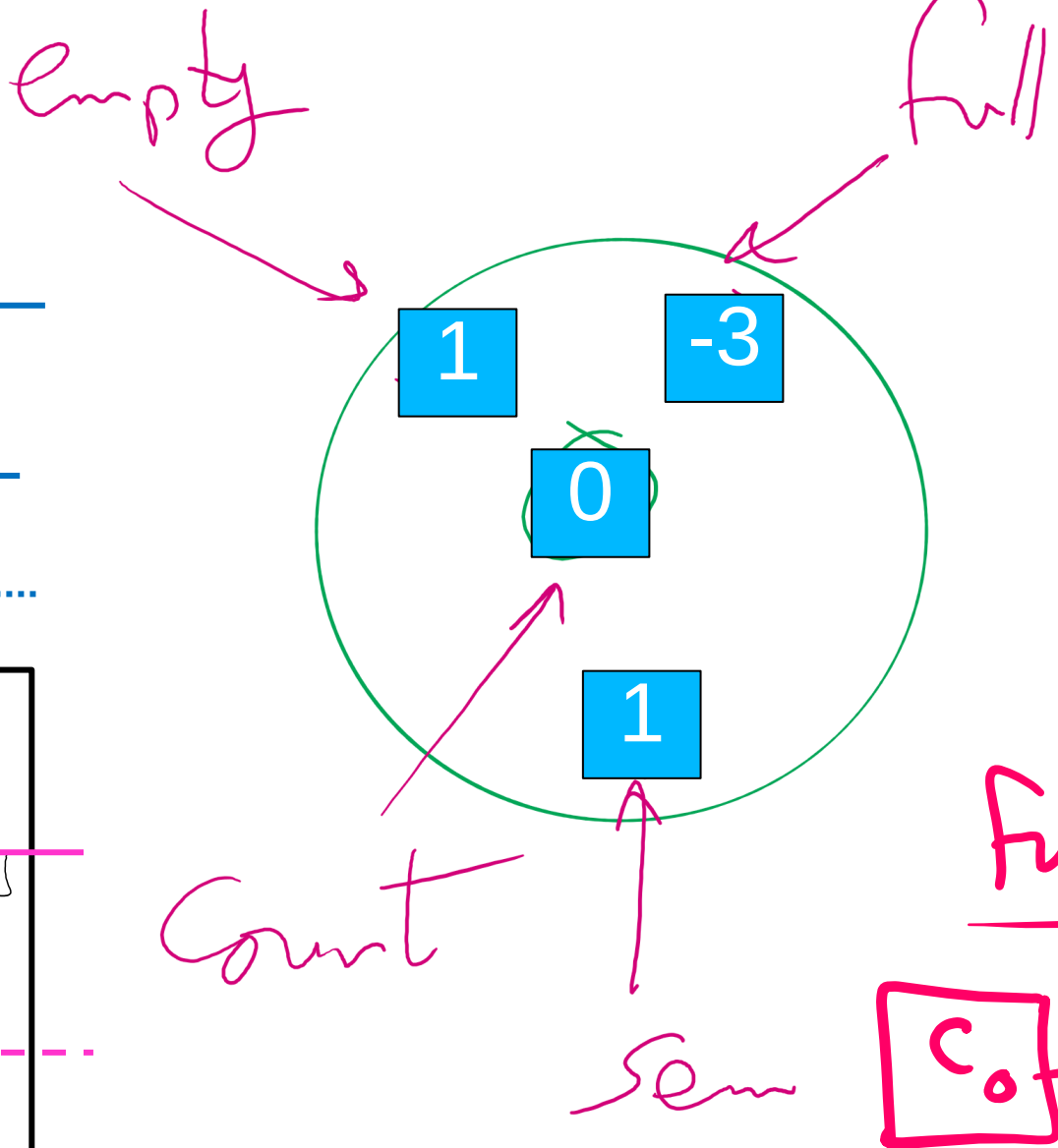
Full Queue



P0 arrives



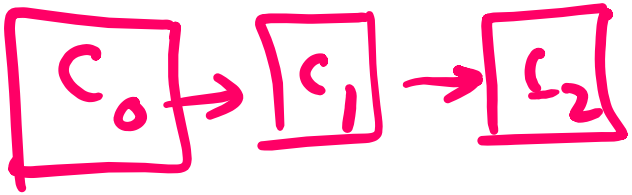




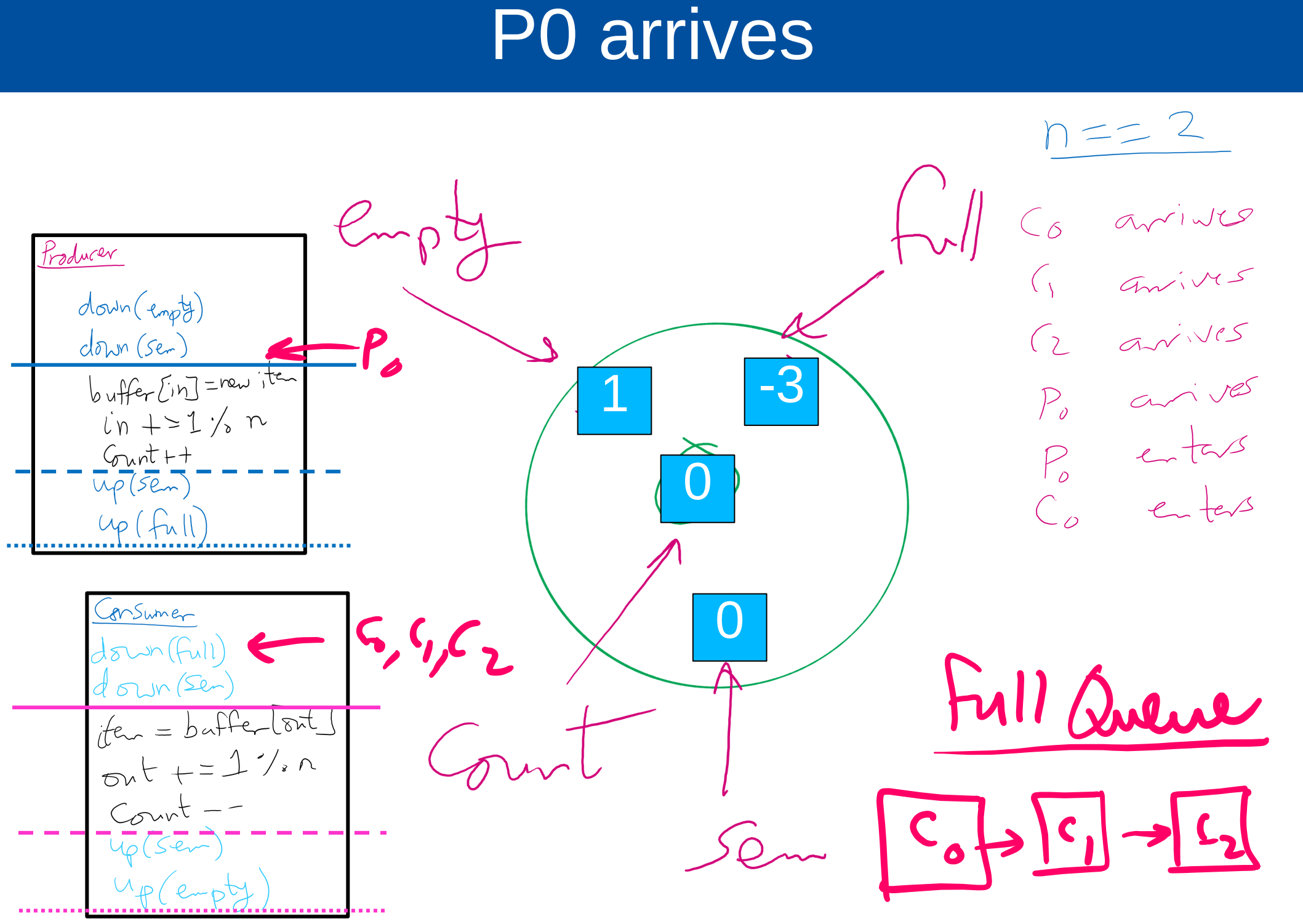
n == 2

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

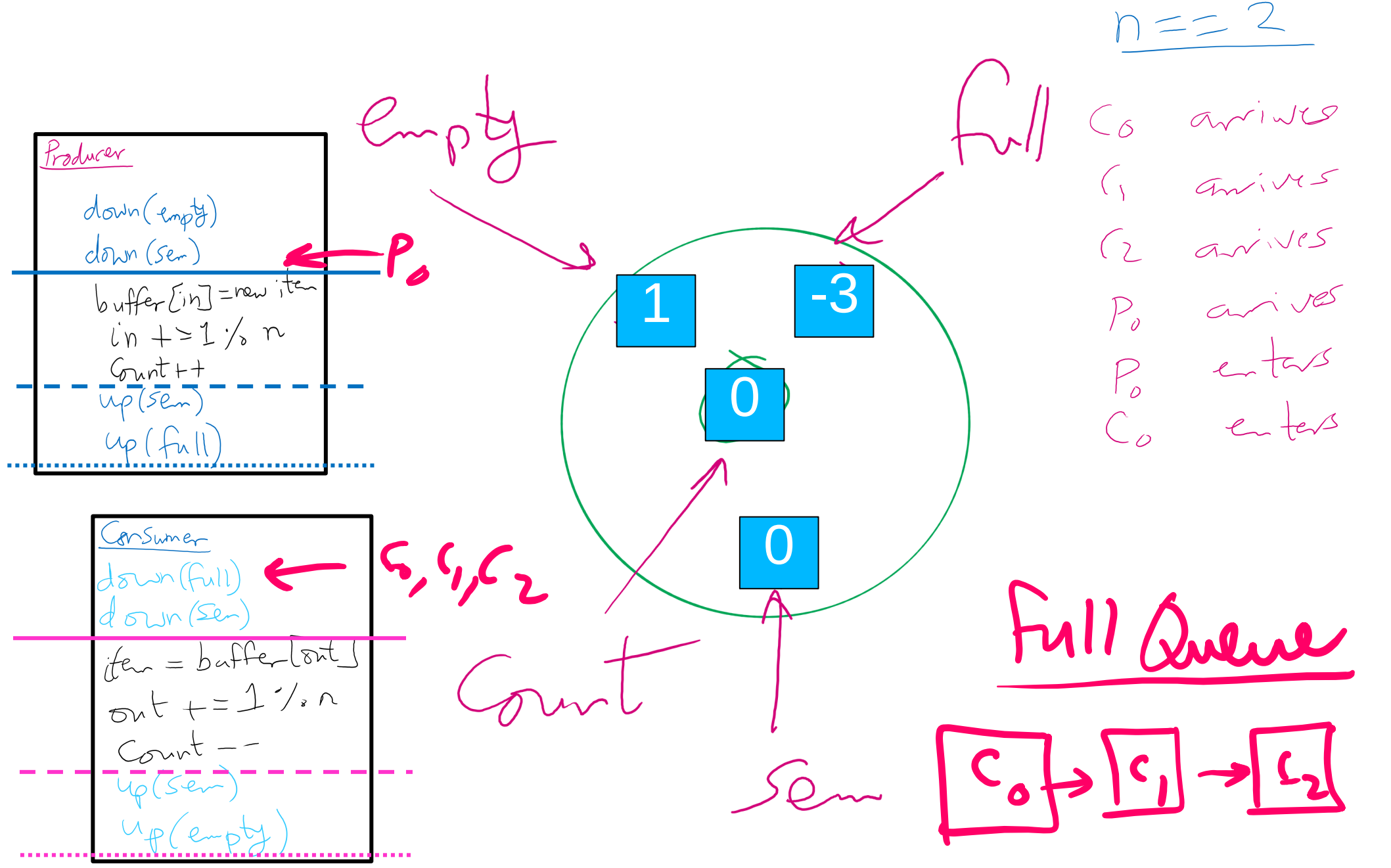
Full Queue



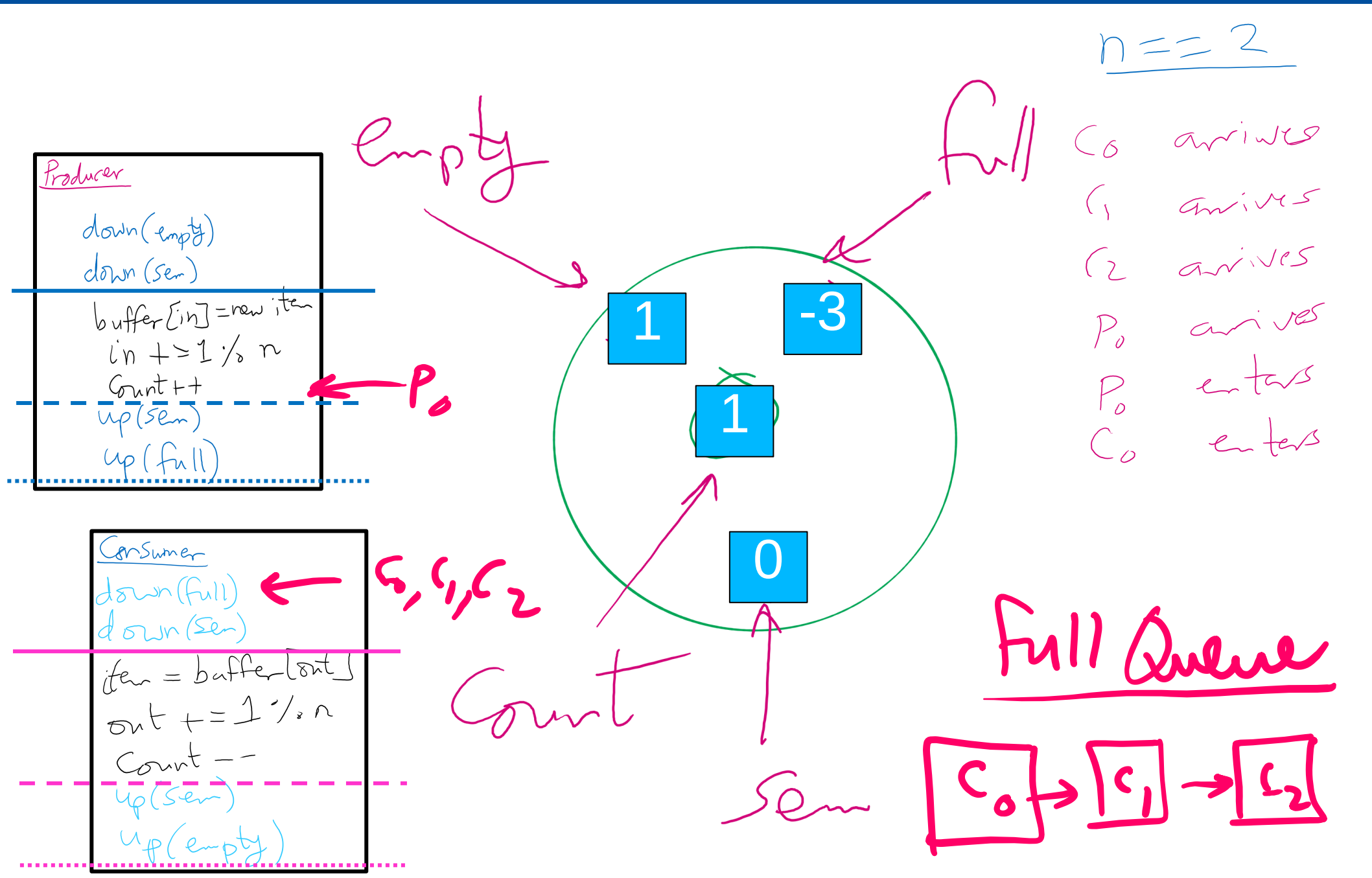
P0 arrives


n == 2
C0 arrives
C1 arrives
C2 arrives
P0 arrives
P0 enters
C0 enters
empty
full
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
1
-3
0
0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0, C1, C2
count
sem
Full Queue
C0
C1
C2

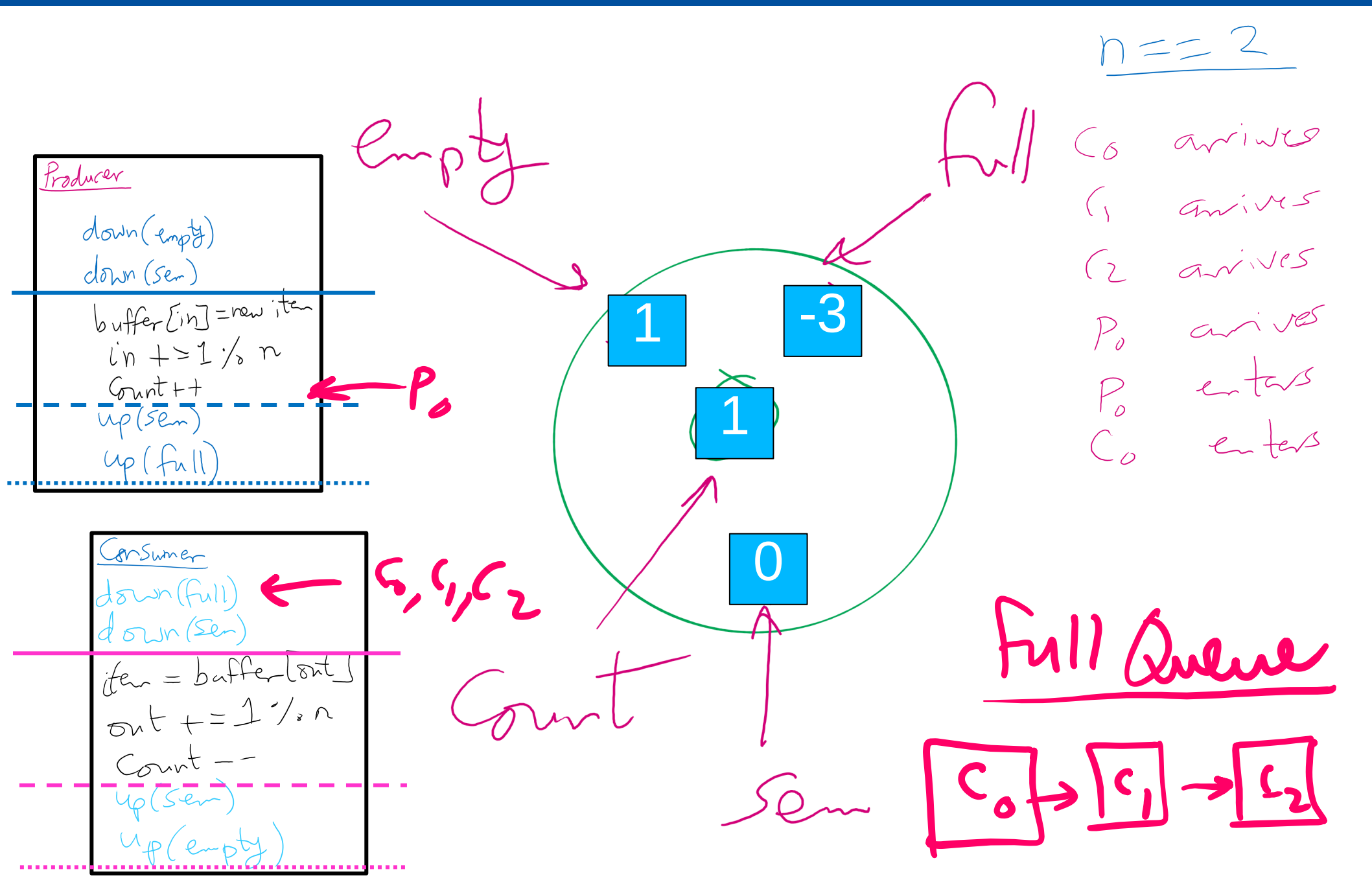
P0 enters


n == 2
C0 arrives
C1 arrives
C2 arrives
P0 arrives
P0 enters
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, C1, C2
empty
full
1
-3
0
0
Count
sem
Full Queue
C0
C1
C2

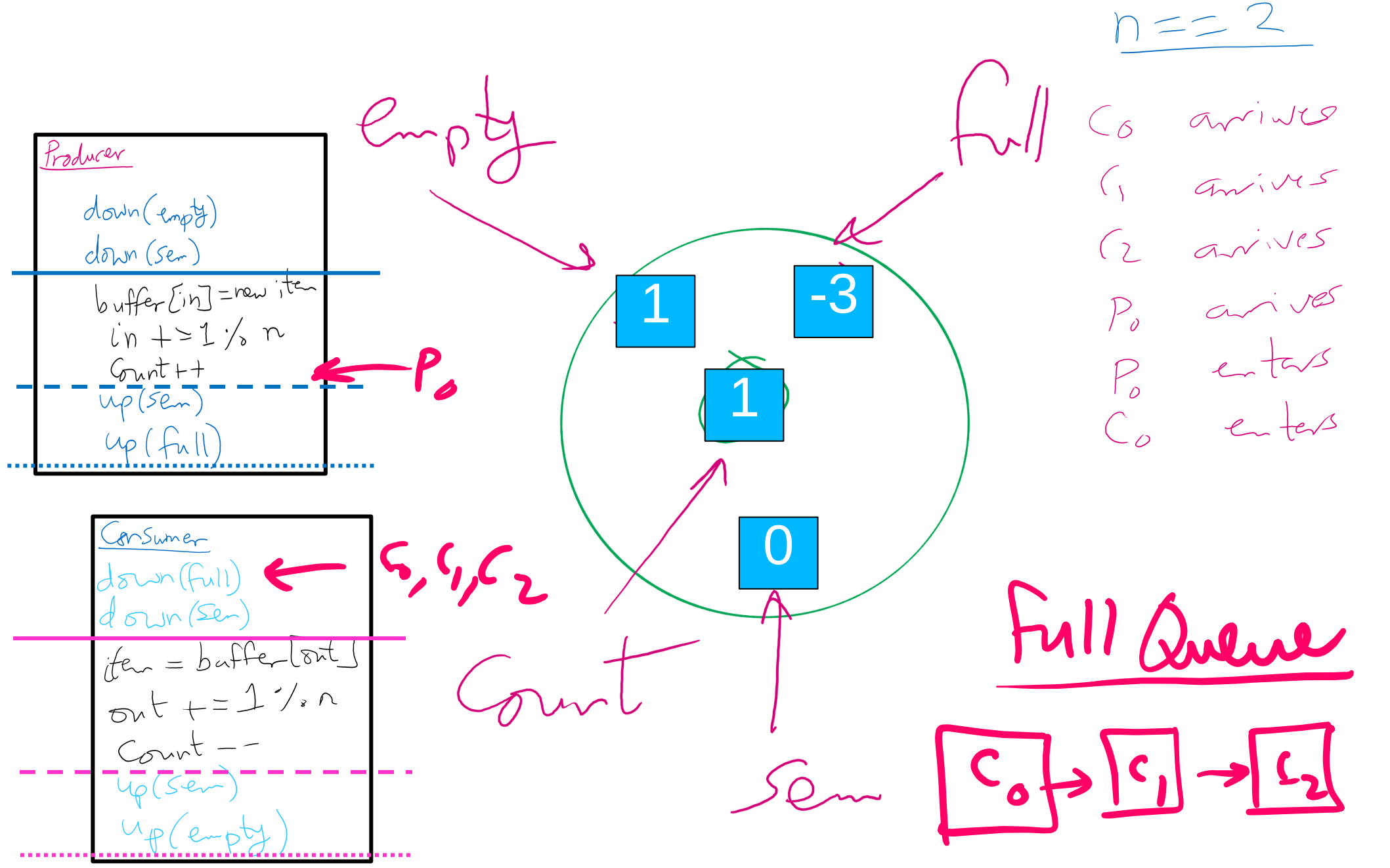
P0 enters


n == 2
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, C1, C2
1
-3
1
0
count
sem
C0 arrives
C1 arrives
C2 arrives
P0 arrives
P0 enters
C0 enters
Full Queue
C0
C1
C2

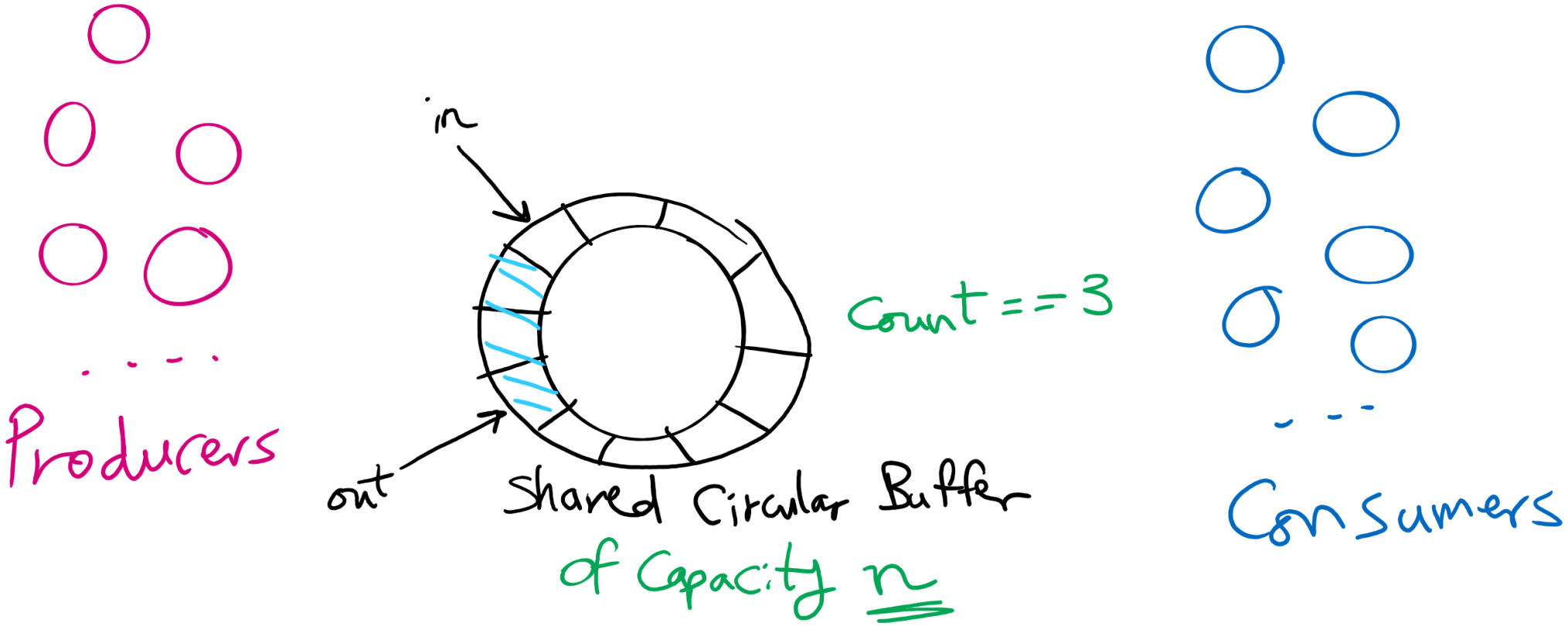
C0 enters


n == 2
C0 arrives
C1 arrives
C2 arrives
P0 arrives
P0 enters
C0 enters
empty
full
Producer
down(empty)
down(sem)
buffer[in] = new item
in += 1 % n
count++
up(sem)
up(full)
P0
1
-3
1
0
Consumer
down(full)
down(sem)
item = buffer[out]
out += 1 % n
count--
up(sem)
up(empty)
C0, C1, C2
count
sem
Full Queue
C0
C1
C2

Can C0 enter?


n == 2
C0 arrives
C1 arrives
C2 arrives
P0 arrives
P0 enters
C0 enters
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, C1, C2
1
-3
1
0
count
sem
Full Queue
C0
C1
C2

Produces Consumers Problem



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)
```

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
```

Warning!

It is easy to make mistakes when using semaphores

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 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