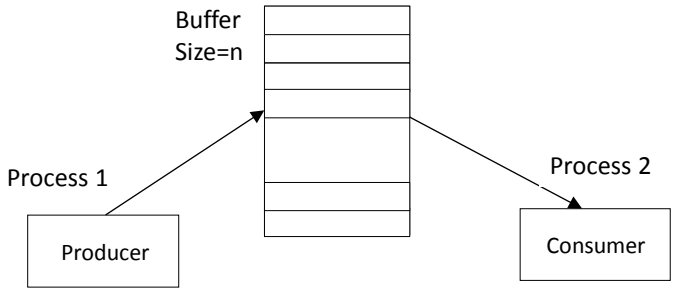
The Producer/Consumer Problem

Process 1 produces data items and stores them in a buffer that can hold up to n items. If it tries to write while buffer is full, it should be blocked. Process 2 independently read and remove items from buffer. If buffer is empty, Process 2 should be blocked.



The Producer/Consumer Problem

We use two semaphores, empty (initially = n) and full (initially =0).

Producer code

```
..
..
down(empty);
Write_item;
up(full);
..
..
```

Consumer code

```
..
..
down(full);
Read_item;
up(empty);
..
..
```

The Producer/Consumer Problem

If producer and consumer may access the same item simultaneously.

Producer code

```
..
..
down(empty);
down(s);
Write_item;
up(s);
up(full);
..
..
```

Consumer code

```
..
..
down(full);
down(s);
Read_item;
up(s);
up(empty);
..
..
```


The Producer/Consumer Problem

Is this solution also correct?

Producer code

```
..
..
down(s);
down(empty);
Write_item;
up(full);
up(s);
..
..
```

Consumer code

```
..
..
down(s);
down(full);
Read_item;
up(empty);
up(s);
..
..
```

Signaling using Semaphores

Consider three processes running independently at unknown speeds.

Process 1	Process 2	Process 3
…	…	…
`fn1()`	`fn2()`	`fn3()`
…	…	…

How to use semaphores such that `fn2()` in process 2 cannot be executed until `fn1()` ends execution, and `fn3()` in process 3 cannot be executed until `fn2()` ends execution.