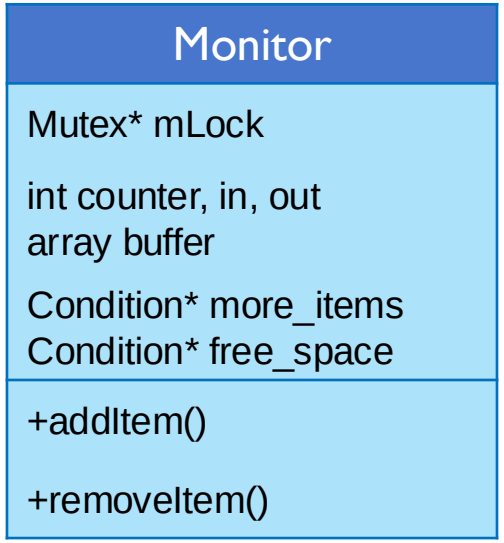
OPERATING SYSTEMS

PRODUCER/CONSUMER MONITOR

- Shared Variables:
  - Counter
  - Buffer
  - In
  - Out
- Condition Variables:
  - free_space
  - more_items
- Is 'item' shared?
  - It is *not* shared, produced and consumed locally.

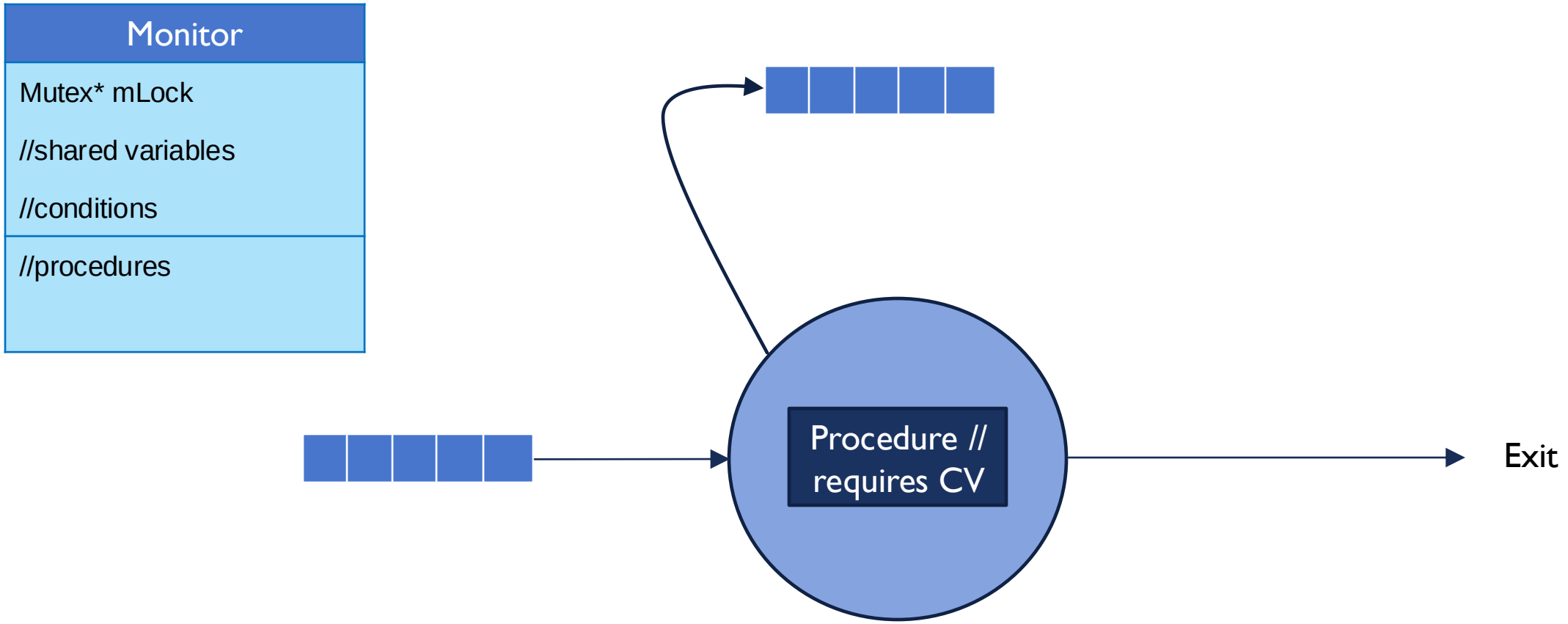



```
in = out = 0;
```

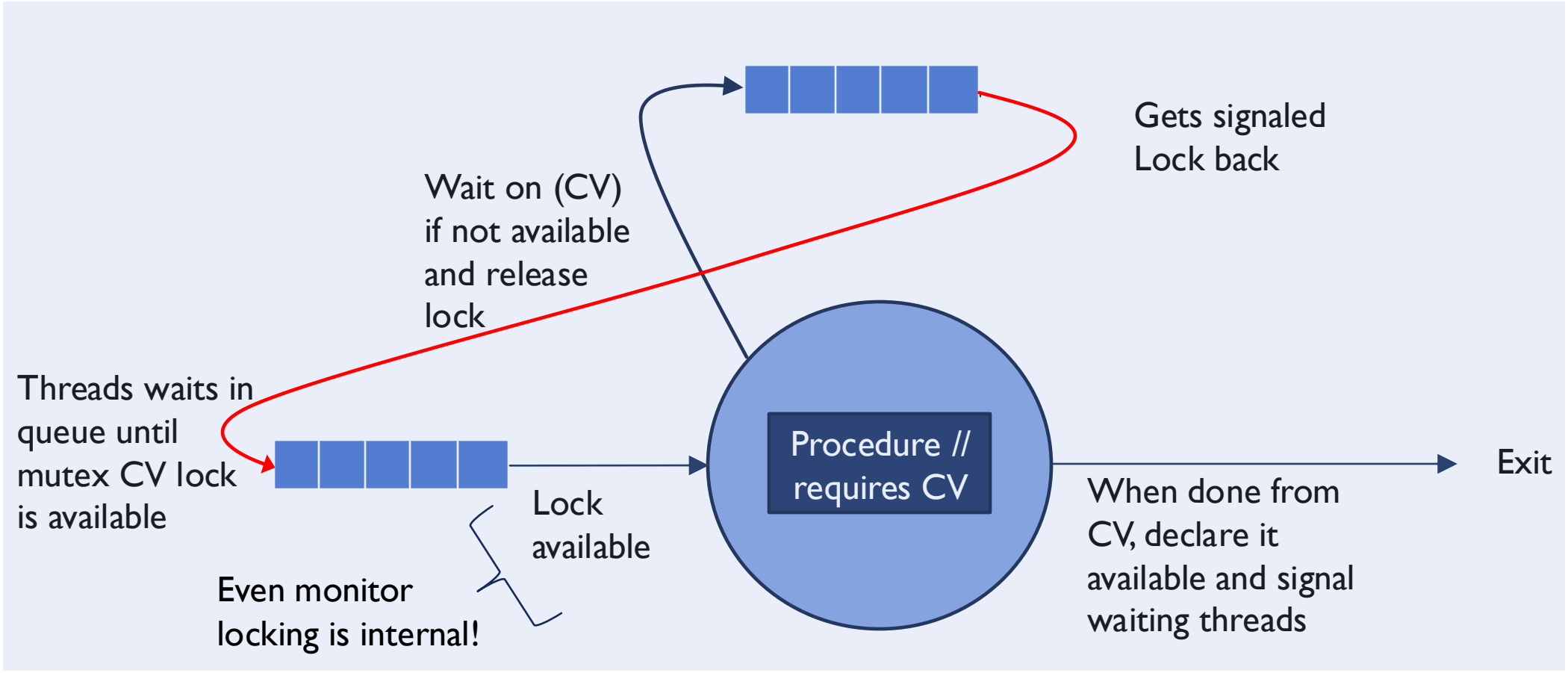
```
while (true) {
item = produce_item;
while
(counter == BUFFER_SIZE) {}/* do nothing */;
buffer[in] = item;
in = (in + 1) % BUFFER_SIZE;
counter++;
}
```

```
while (true) {
while (counter == 0) {}/* do nothing
item = buffer[out];
out = (out + 1) % BUFFER_SIZE;
counter--;
consume_item(item);
```

MONITOR USAGE FLOW CHART



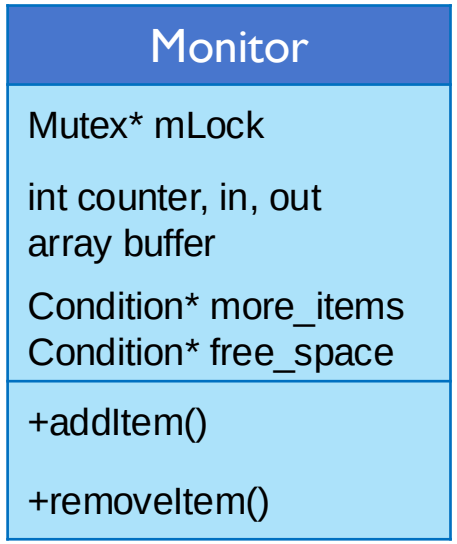
MONITOR USAGE FLOW CHART



This is all internal

PRODUCER/CONSUMER MONITOR

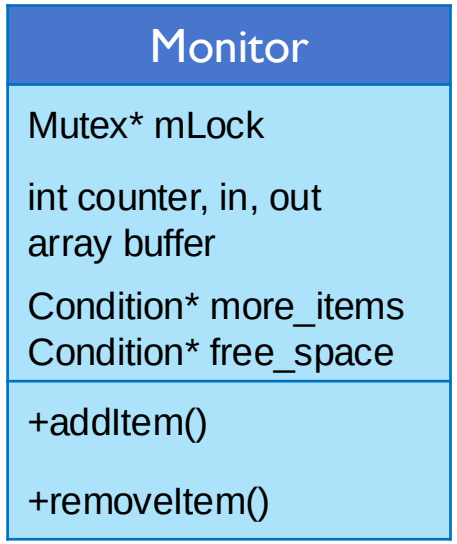
- Shared Variables:
  - Counter
  - Buffer
  - In
  - Out
- Condition Variables:
  - free_space
  - more_items
- Is 'item' shared?
  - It is *not* shared, produced and consumed locally.




```
monitor ProducerConsumer
{
   int itemCount = 0;
   condition free_space;
   condition more_items;
   mutex mLock;

method remove(item) {

}

method add(item) {

}
```

PRODUCER/CONSUMER MONITOR

- Shared Variables:
  - Counter
  - Buffer
  - In
  - Out
- Condition Variables:
  - free_space
  - more_items
- Is 'item' shared?
  - It is *not* shared, produced and consumed locally.



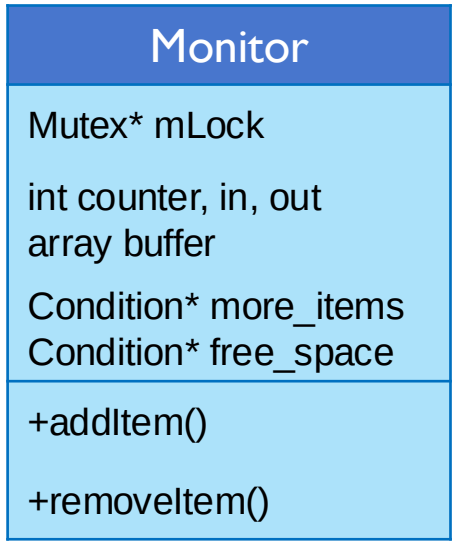

```
monitor ProducerConsumer
{
   int itemCount = 0;
   condition free_space;
   condition more_items;
   mutex mLock;
```

```
method remove(item) {

        mLock.Lock ()

        while(bufferLength == 0)
          more_items.Wait(&mLock)
        end

        -- remove item

        --signal relevant condition

        free_space.Signal (&mLock)
        mLock.Unlock ()
}
```

```
method add(item) {

}
```

PRODUCER/CONSUMER MONITOR

- Shared Variables:
  - Counter
  - Buffer
  - In
  - Out
- Condition Variables:
  - free_space
  - more_items
- Is 'item' shared?
  - It is *not* shared, produced and consumed locally.




```
monitor ProducerConsumer
{
   int itemCount = 0;
   condition free_space;
   condition more_items;
   mutex mLock;
```

```
method remove(item) {

        mLock.Lock ()

        while(bufferLength == 0)
          more_items.Wait(&mLock)
        end

        -- remove item

        --signal relevant condition

        free_space.Signal (&mLock)
        mLock.Unlock ()
}
```

```
method add(item) {

        mLock.Lock ()

        while(bufferLength == SIZE)
          free_space.Wait(&mLock)
        end

        -- add item

        --signal relevant condition

        more_items.Signal (&mLock)
        mLock.Unlock ()
}
```