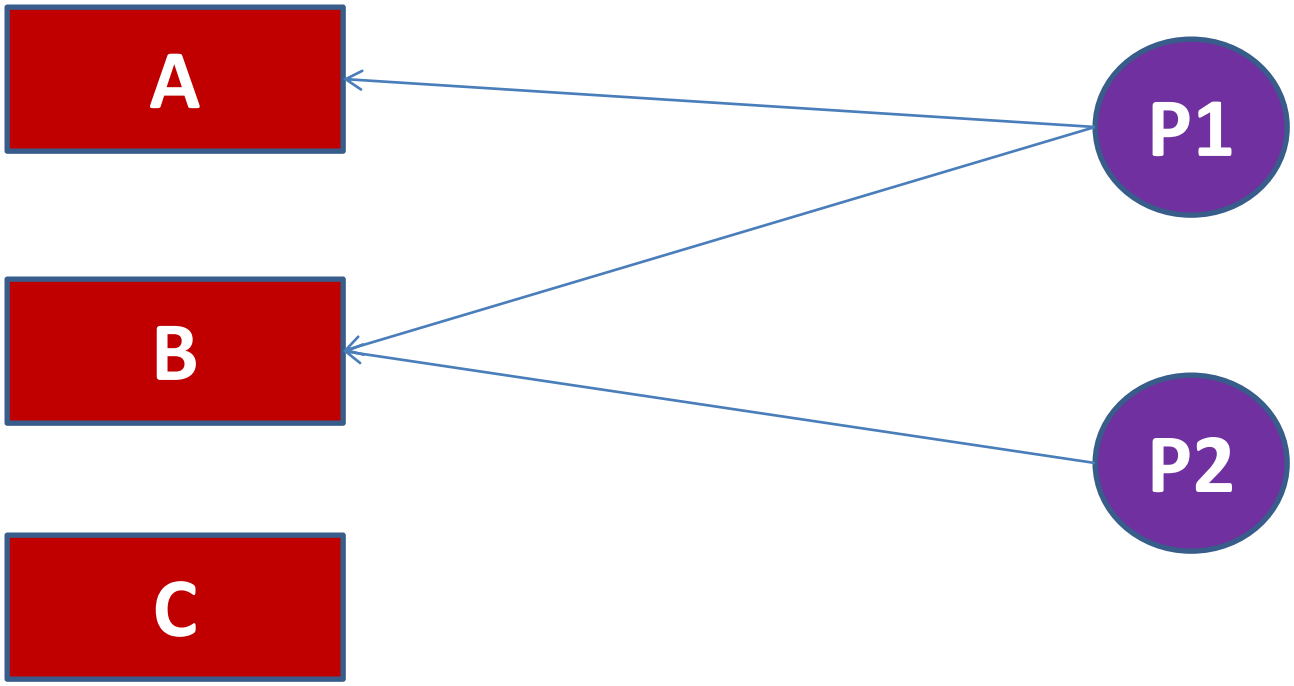
Monitor

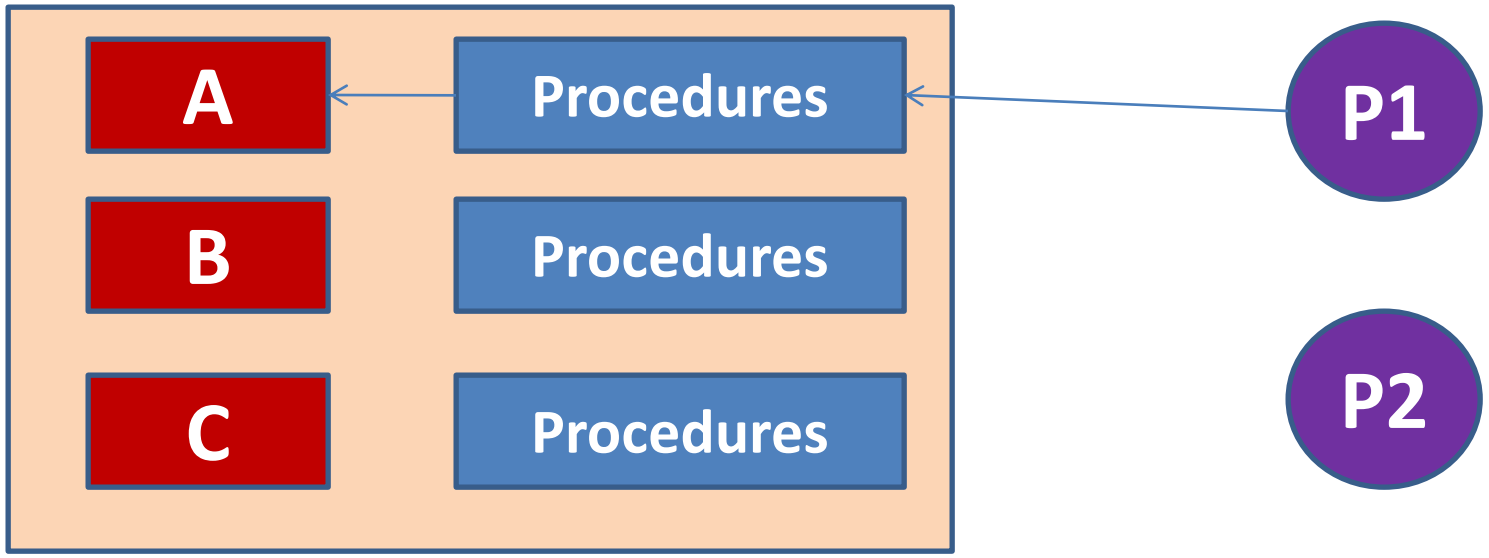
By
Dr. Upasana Pandey
Department of Computer Science & Engineering
IMS Engineering College (College Code:143)

Monitor



When multiple process access shared resources simultaneously, create problem of race condition.

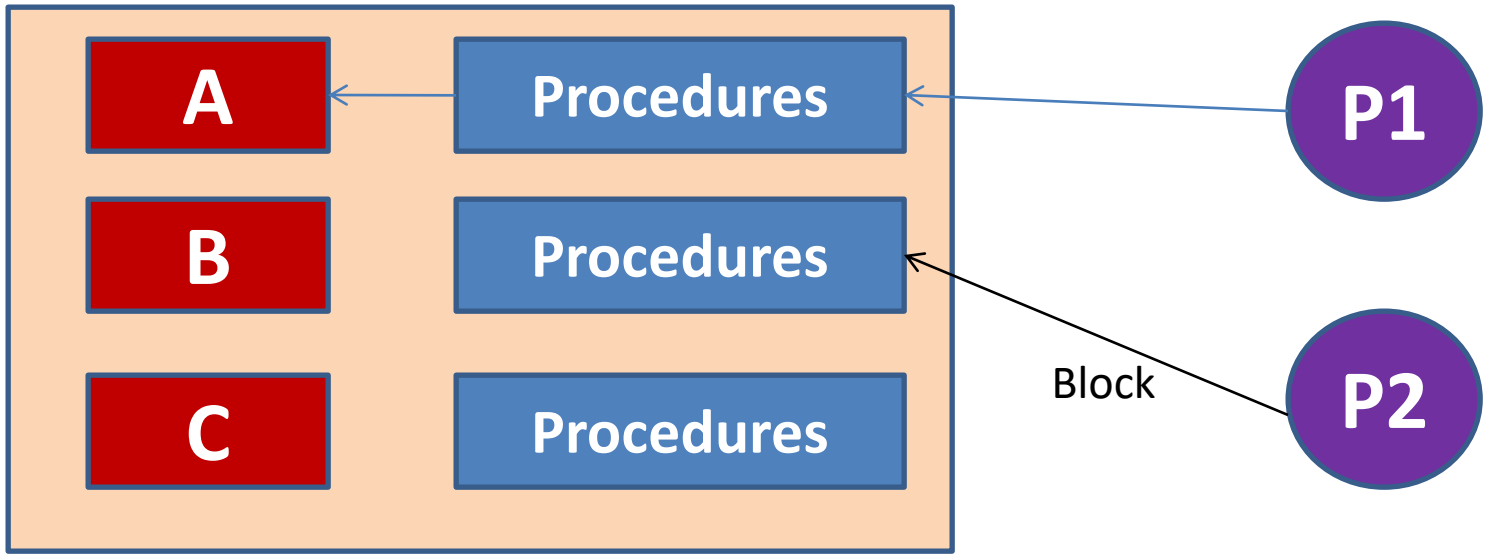
Monitor



A monitor is a module that encapsulates:

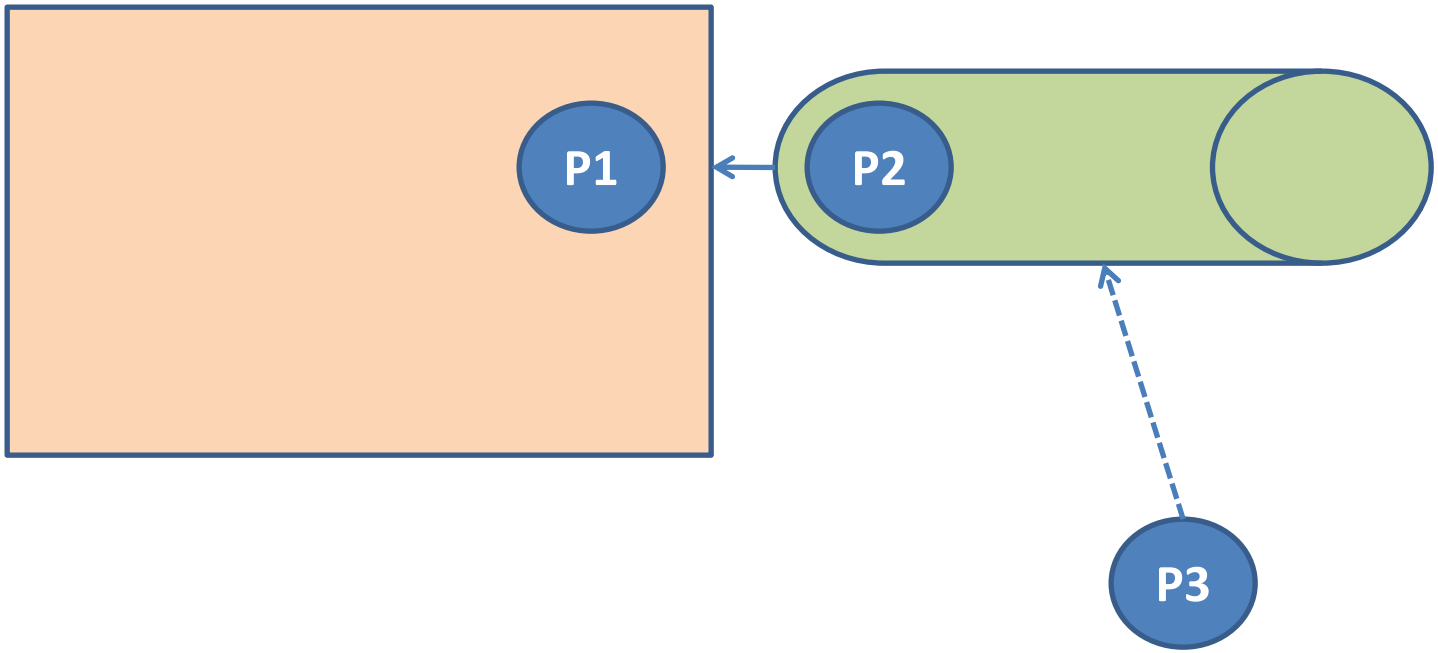
- **Shared data structure.**
- **Procedure that operates on shared data.**
- **Synchronization between concurrent procedure invocation.**

Monitor



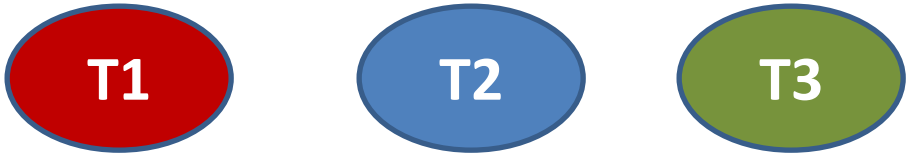
Only one process can enter into monitor at a time.

Monitor

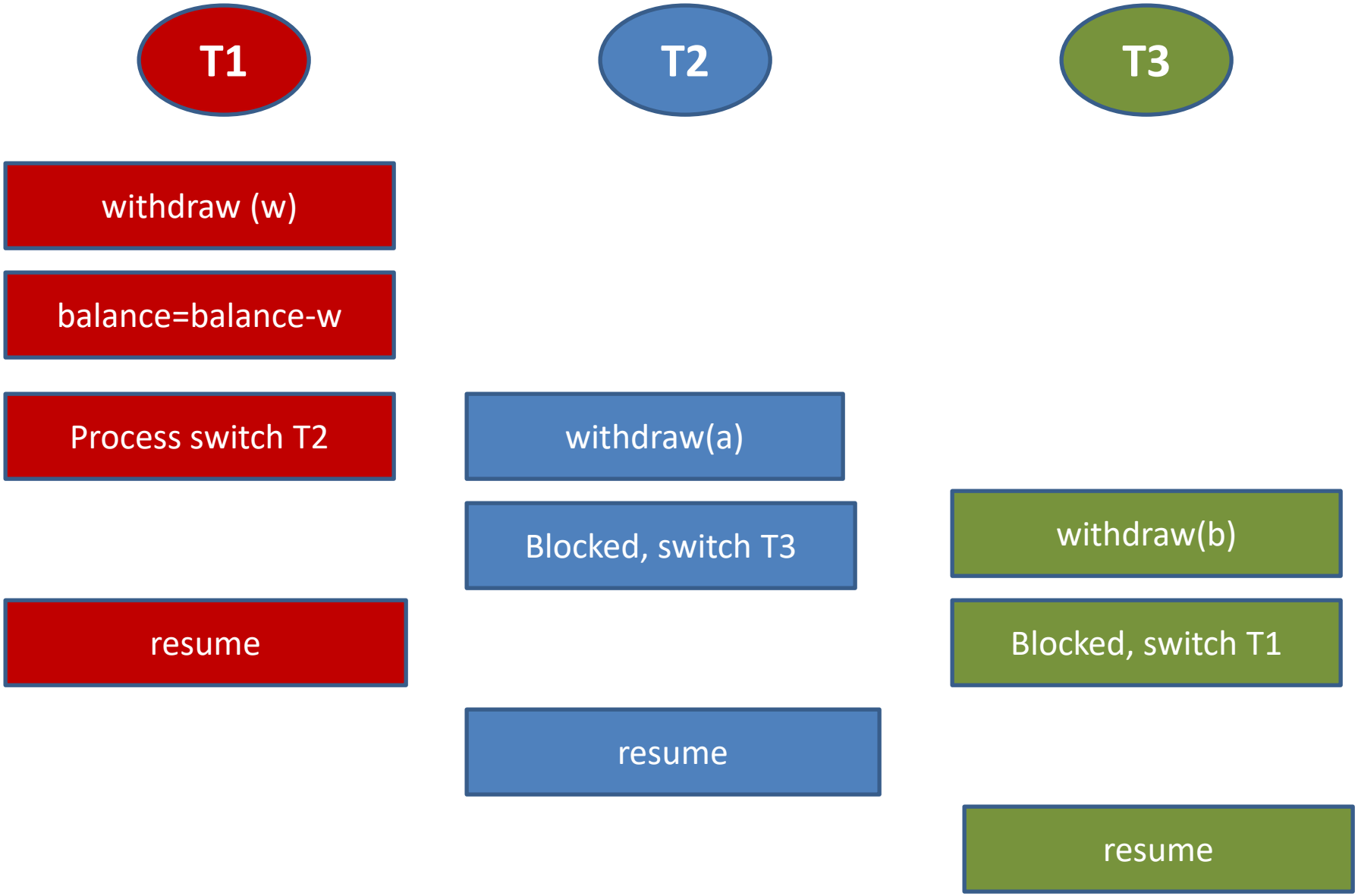


Monitor

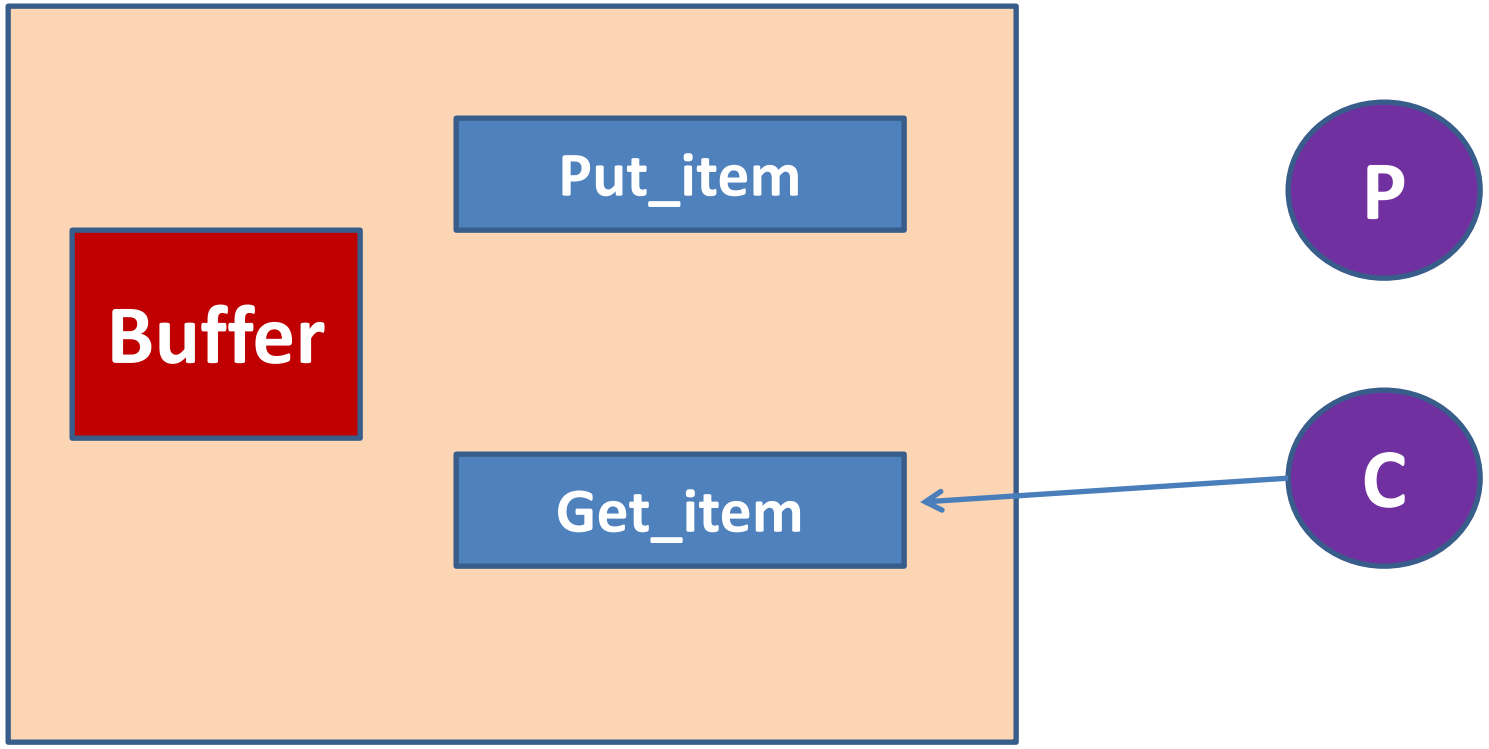
```
Monitor account
{
Double balance;

Withdraw amount()
{
balance=balance-amount
return balance
}

}
```



Monitor


T1
T2
T3
withdraw (w)
balance=balance-w
Process switch T2
withdraw(a)
Blocked, switch T3
withdraw(b)
Blocked, switch T1
resume
resume
resume

Bounded Buffer Problem



Initially buffer is empty.

Consumer entered into monitor for consuming data, buffer is empty, it starts waiting in monitor.

Now, producer wants to produce data, but monitor will not allow it to enter because consumer already entered into monitor.

Solution: Conditional Variable

Conditional variable

- Conditional variable provides synchronization inside the monitor.
- If a process wants to sleep inside the monitor or it allows a waiting process to continue, in that case conditional variable is used in monitor.
- Three operation can be performed:
  - wait, signal and broadcast.

Conditional variable

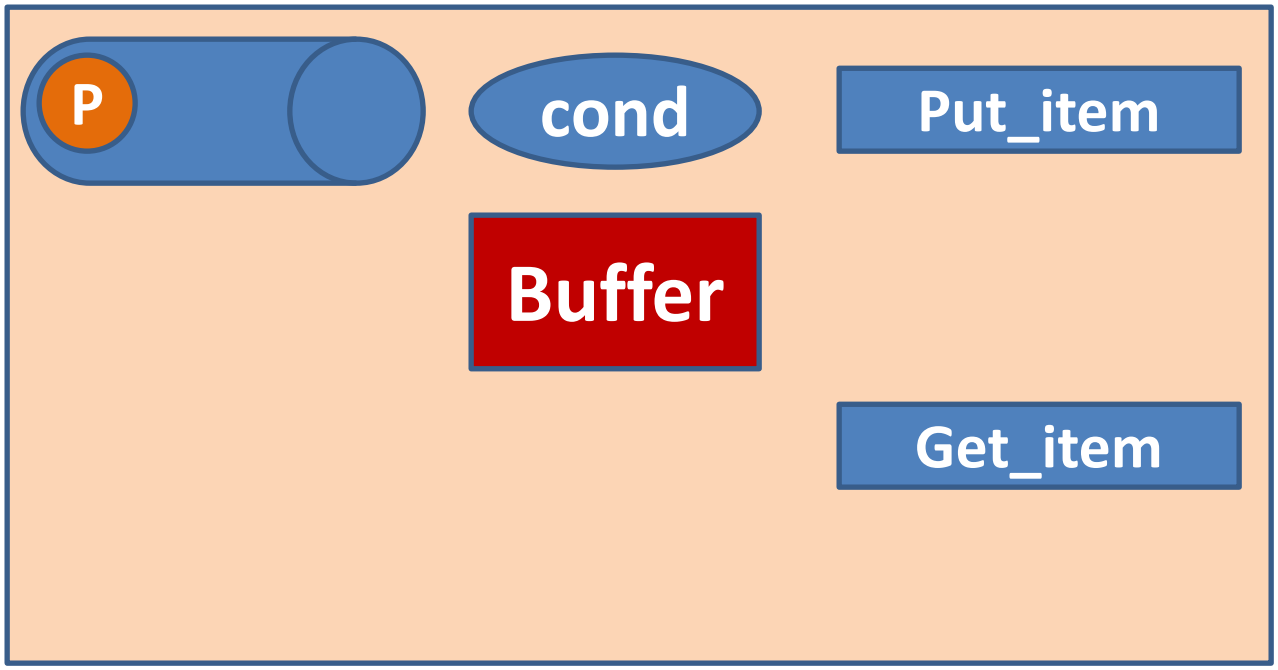
- Wait: if resource is currently not available, current process put to sleep. It releases the lock for monitor.

- Signal: it wakes up one process which are sleeping as a result of wait(). This causes a waiting process to resume immediately. The lock is automatically pass to the waiter, the original process blocked now.

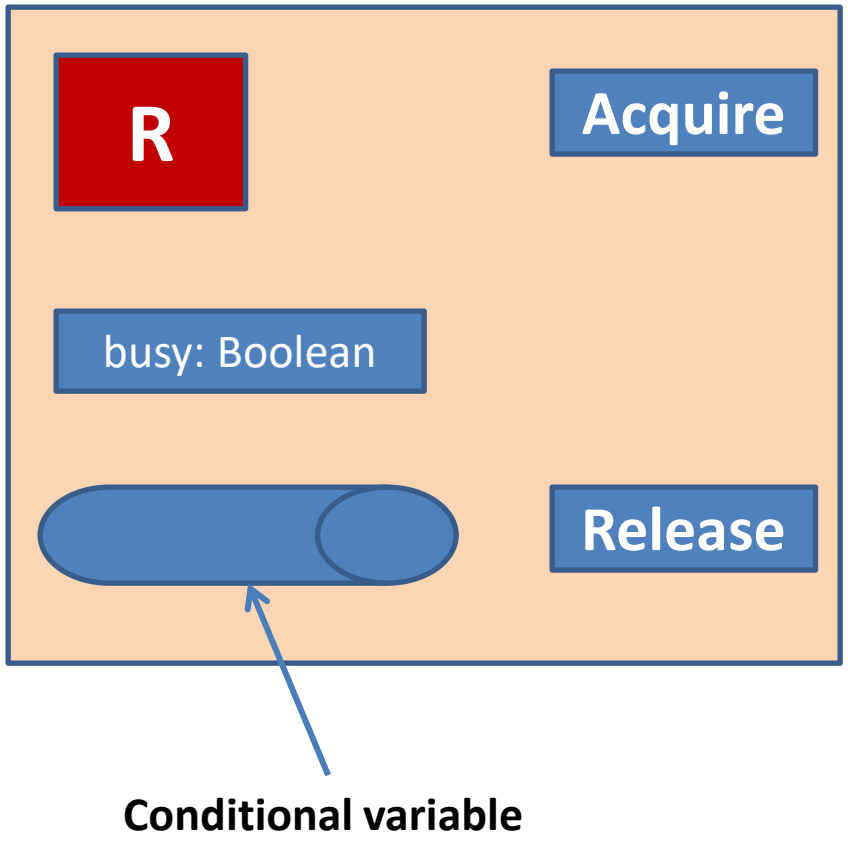
- Broadcast: it signal to all waiting processes.

Conditional variable

Conditionvariable.wait
Conditionvariable.signal

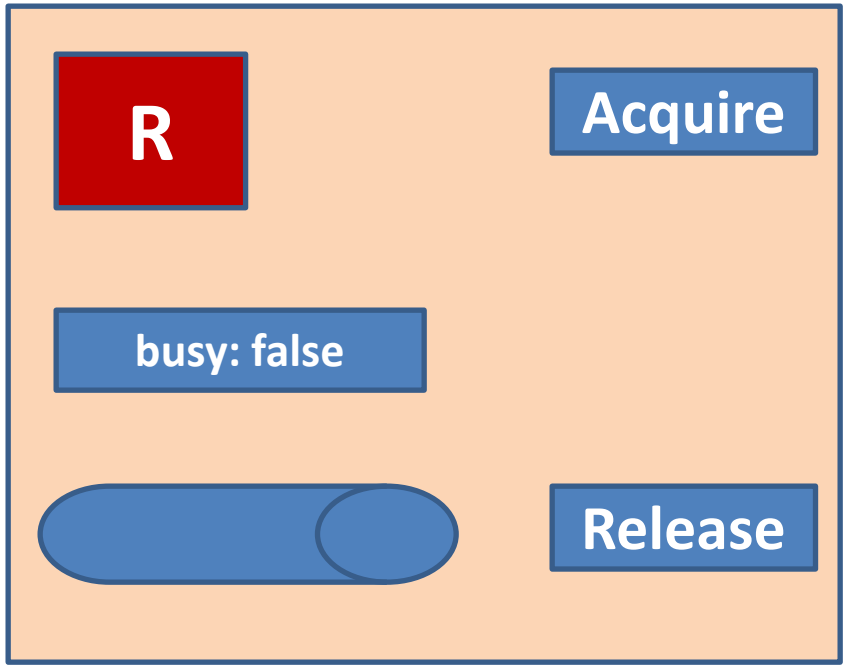


Single Resource Allocation




```
Monitor single_resource
{
boolean busy;
condition nonbusy;

   Acquire()
   {
    if busy then nonbusy.wait
    else busy=true
   }
   Release()
   {
    busy=false;
    Nonbusy.signal;
   }

}
```

Single Resource Allocation



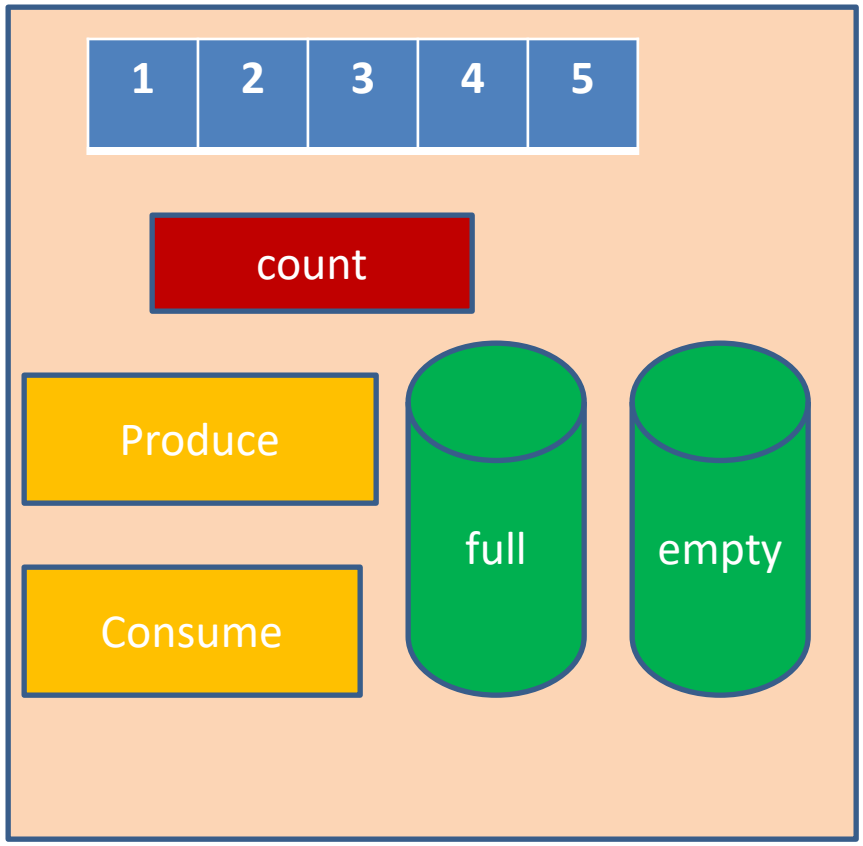
Single Resource

P1	P2
sr.acquire	sr.acquire
…….	……
…….	……
…….	……
sr.release	sr.release

```
acquire()
  {
   if busy then nonbusy.wait
   else busy=true
  }
```

```
release()
  {
   busy=false;
   Nonbusy.signal;
  }
```

Bounded Buffer Problem




```
monitor bb
{
   int data[5];
   int count;
   condition full;
   condition empty;
   produce()
   {
    if count==5 then full.wait
    add item to buffer & increase count value
    empty.signal
    }
    consume()
    {
     if count==0 then empty.wait
     access item from buffer and remove count value
     empty.signal
       }
}
```

Thank You