CSE 330: Operating Systems **Class: 15** **Date: 10/6**

Fall 2016

Note Title

Semaphores

→ critical sections (locking)

→ synchronization

Monitor → critical section

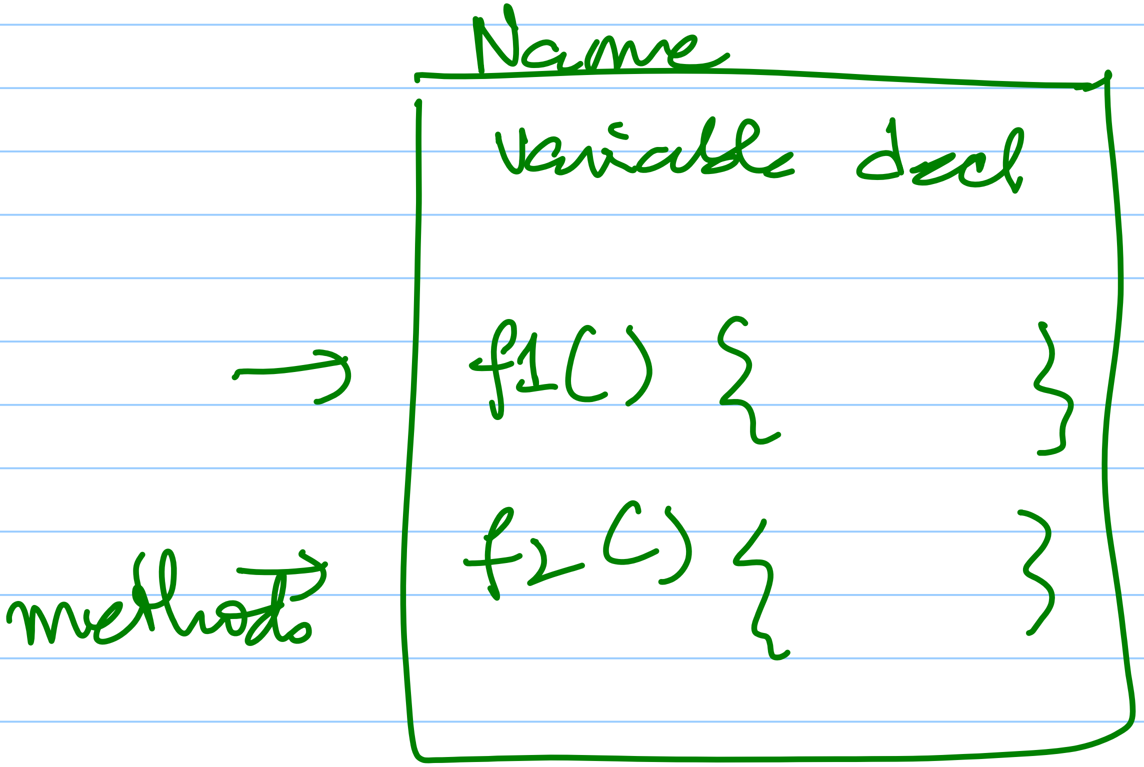
→ structured (object oriented)

Synchonization

→ conditions

↳ variables in a monitor

monitor → class



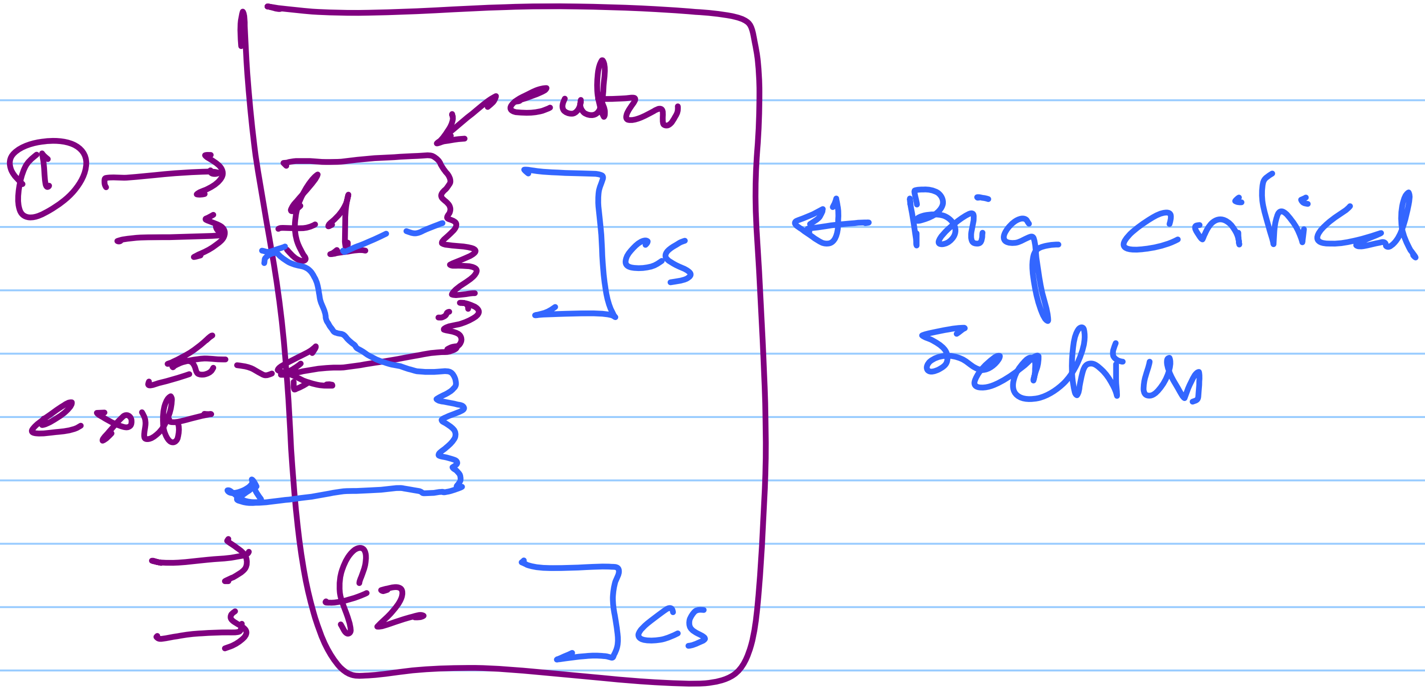
threads can call monitor methods

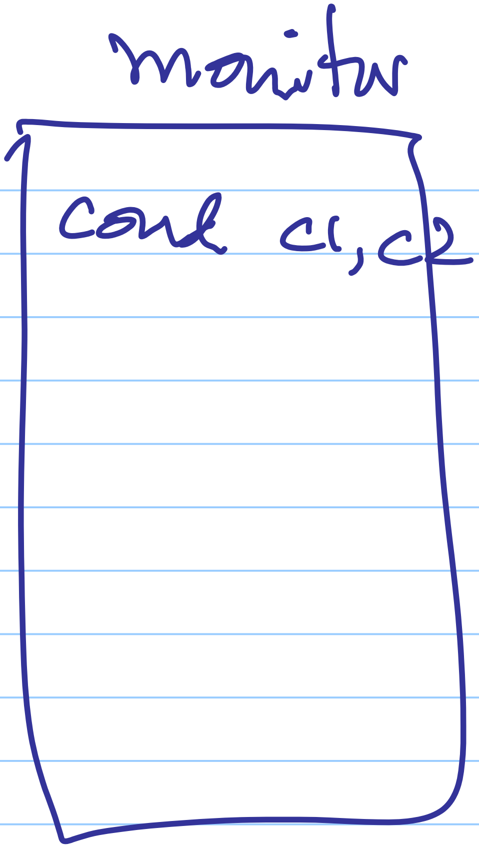
Name.f1()
Name.f2()

Monitor methods

- → can be called by multiple threads – concurrently

- → **BUT** only 1 thread is allowed to execute inside one monitor @ any point in time


1
entry
f1
cs
exit
f2
cs
Big critical section



```
cond c1, c2
```

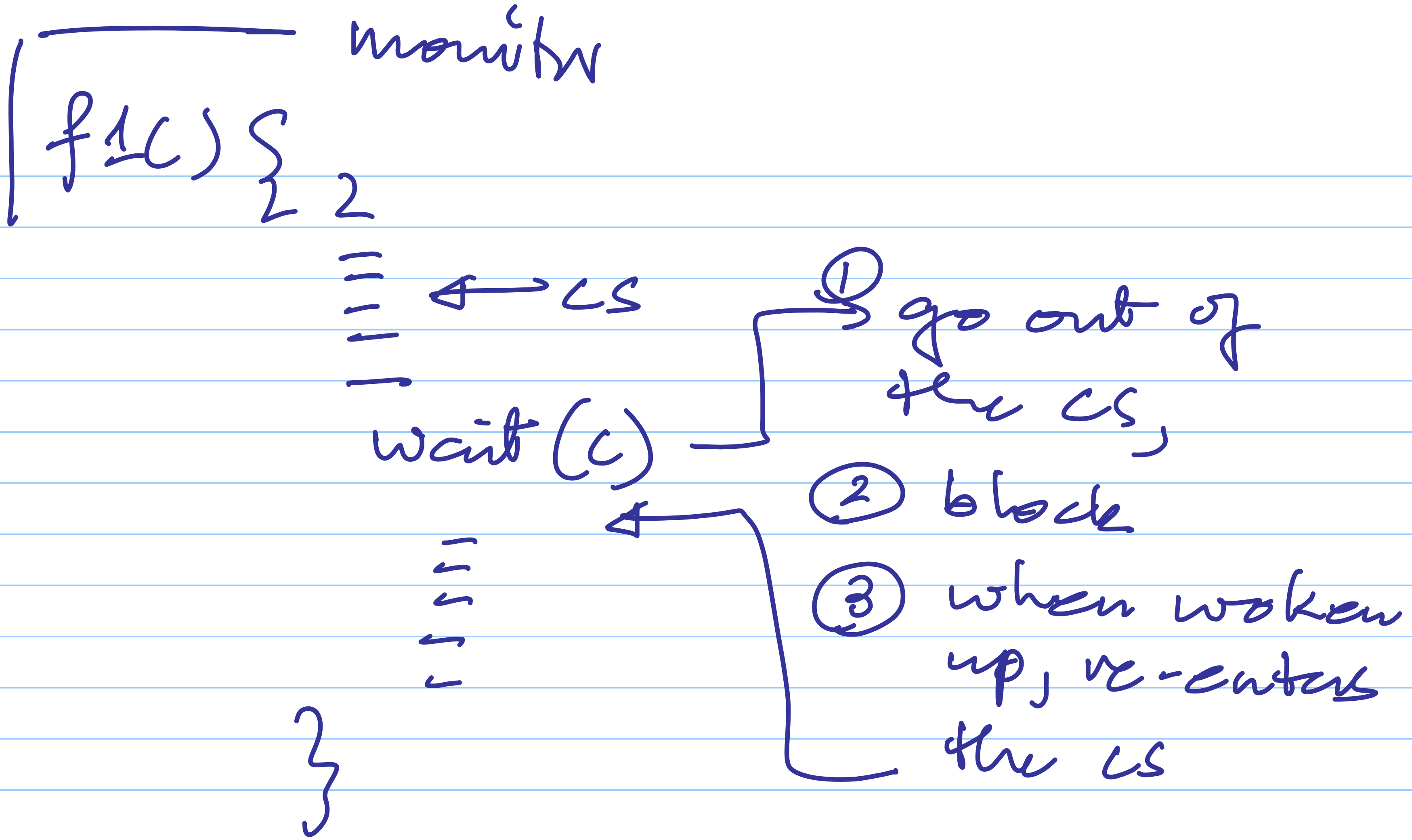
← condition variables declared inside the monitor

→ has no value (!)

2 functions are defined on a condition

1. wait(c) → block (unconditional block ... "forever")

2. signal(c)



① go out of the cs,

② block

③ when woken up, re-enters the cs

```
f1(){ -
      -
      -
      -
   signal(c)
      -
      -
      -
      -
  }
```

signal(c) → wake up a thread blocked on that condition - if any

↳ if no thread is blocked

→ NO_OP

if T_1 signals T_2 → T_2 is to wakeup.

option 1

T_1 continues to execute and T_2 has to wait till T_1 exits the monitor or T_1 blocks.

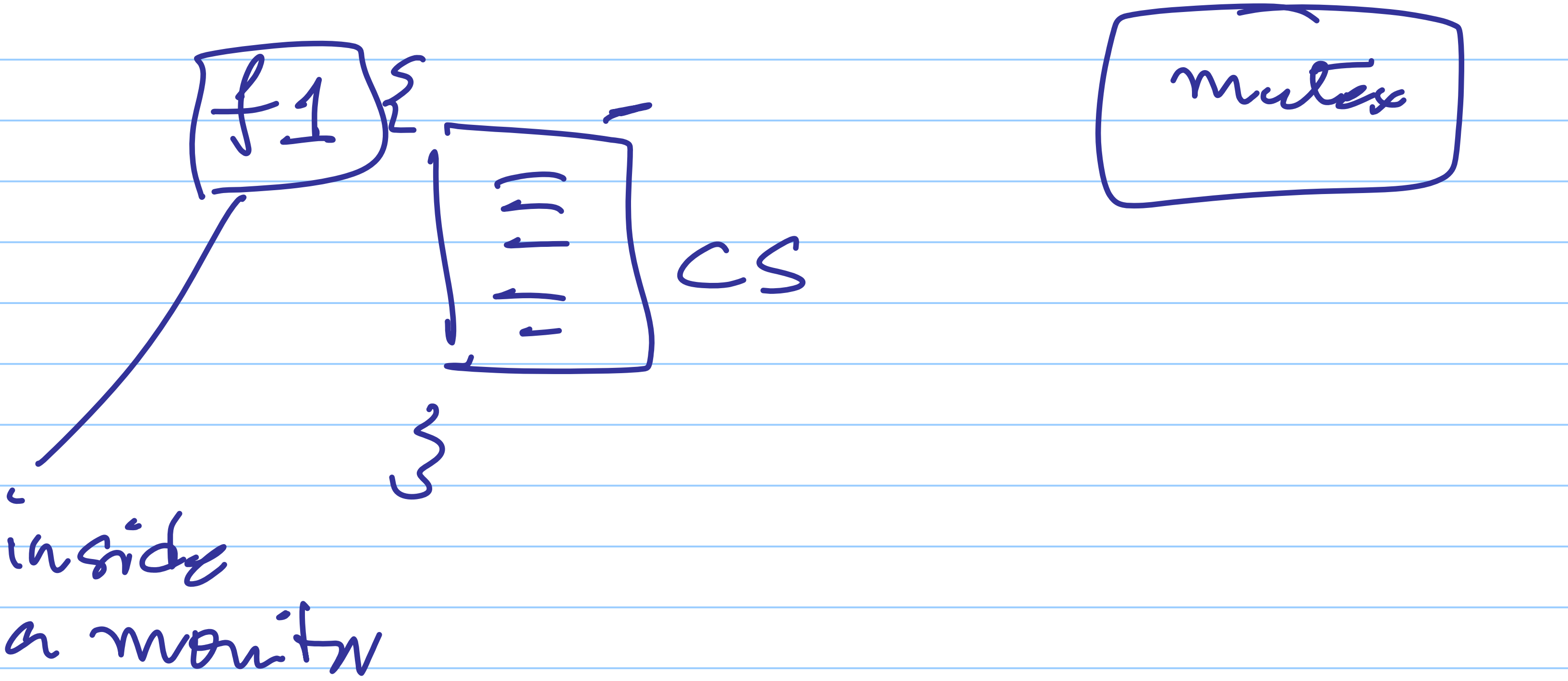
pthreads →

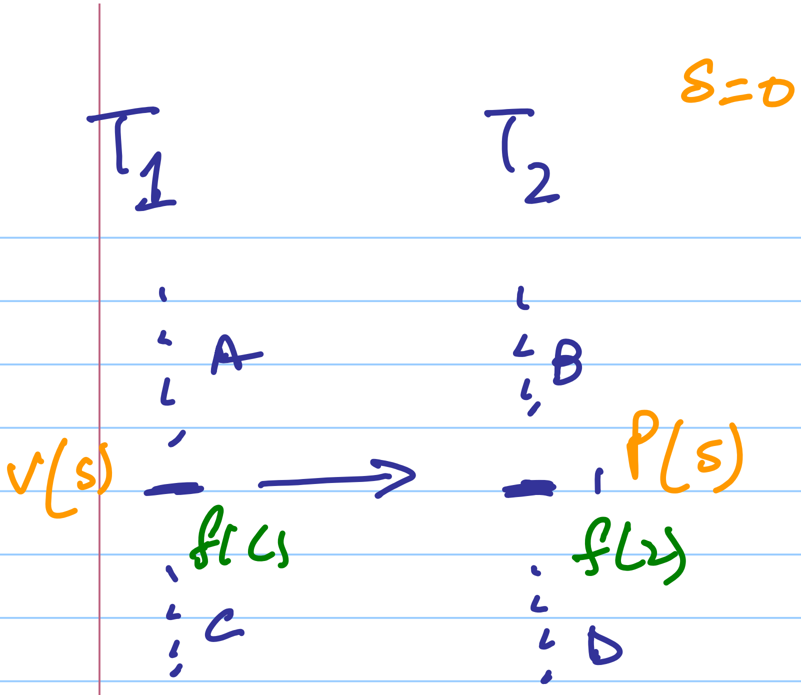
option 2 →

(T_2 has priority over other threads)

→ T_1 blocks. T_2 executes, T_1 executes when T_2 leaves/blocks.

programming with monitors





A → D

```
sync_mon
  f1() {
        signal(c)
        }
  f2() {
        wait(c)
        }
```

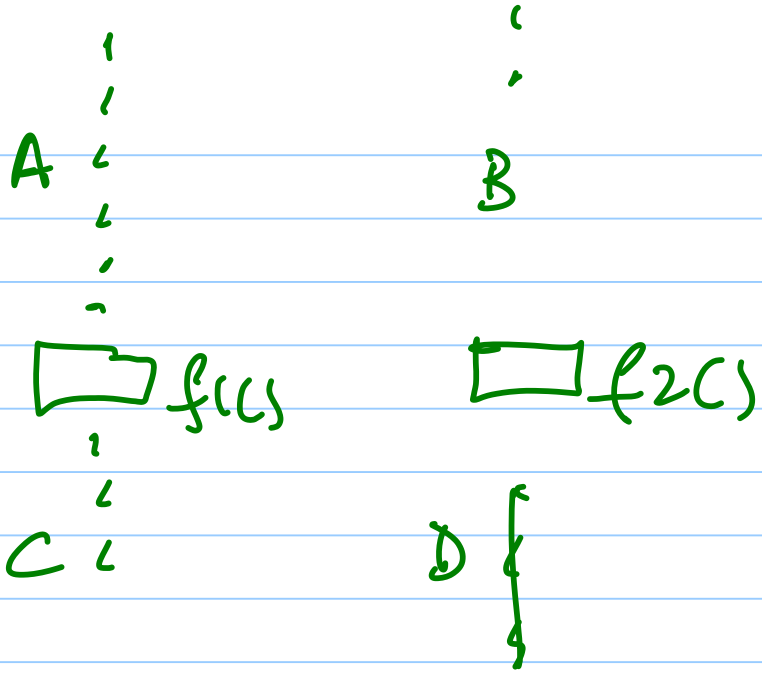
⋮

signal(c_1)
wait (c_2)

(

wait (c_1)
signal(c_2)

X

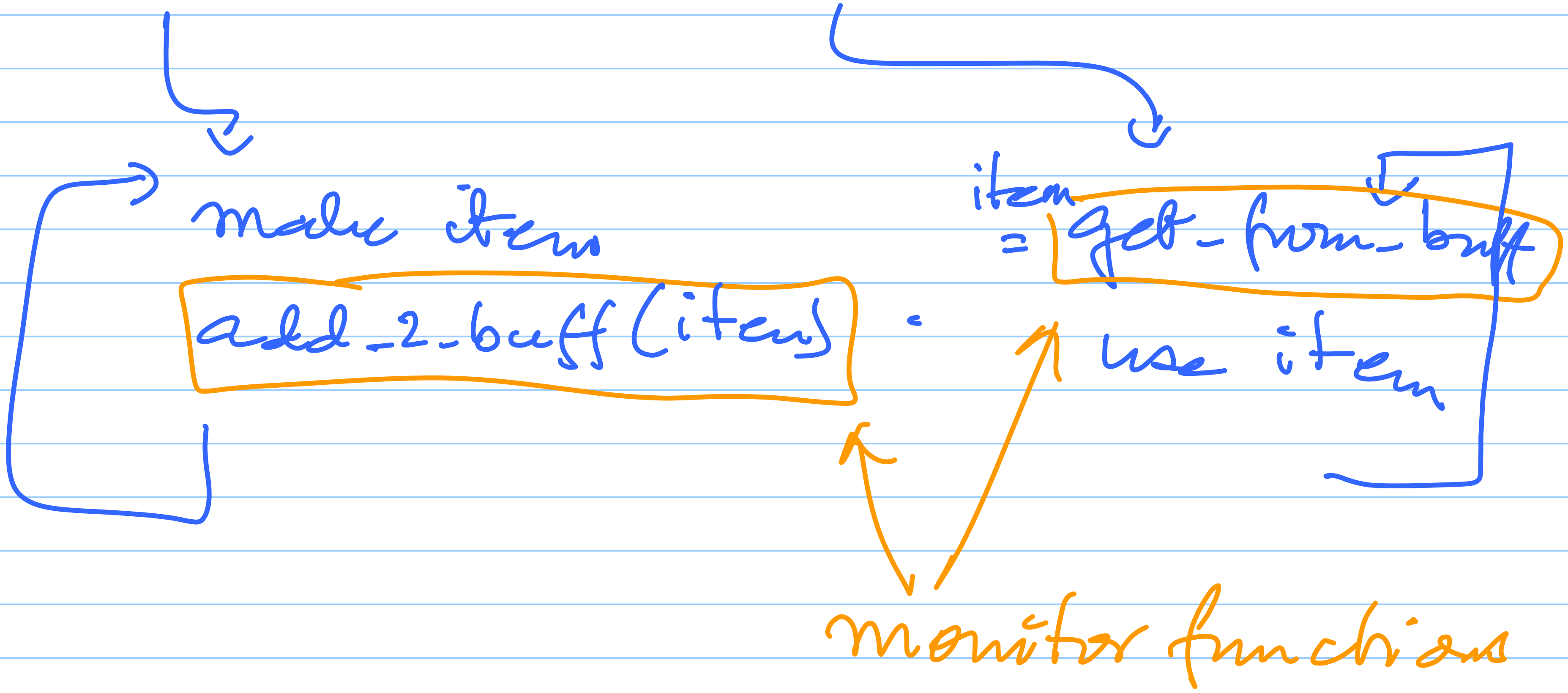


```
int done = 0;
cond c;

f1 { done = 1;
     signal (c); }

f2 { if (done != 1)
       wait (c) }
```

Producer, consumer



make item

add_2_buff(item)

item = get_from_buff

use item

monitor functions

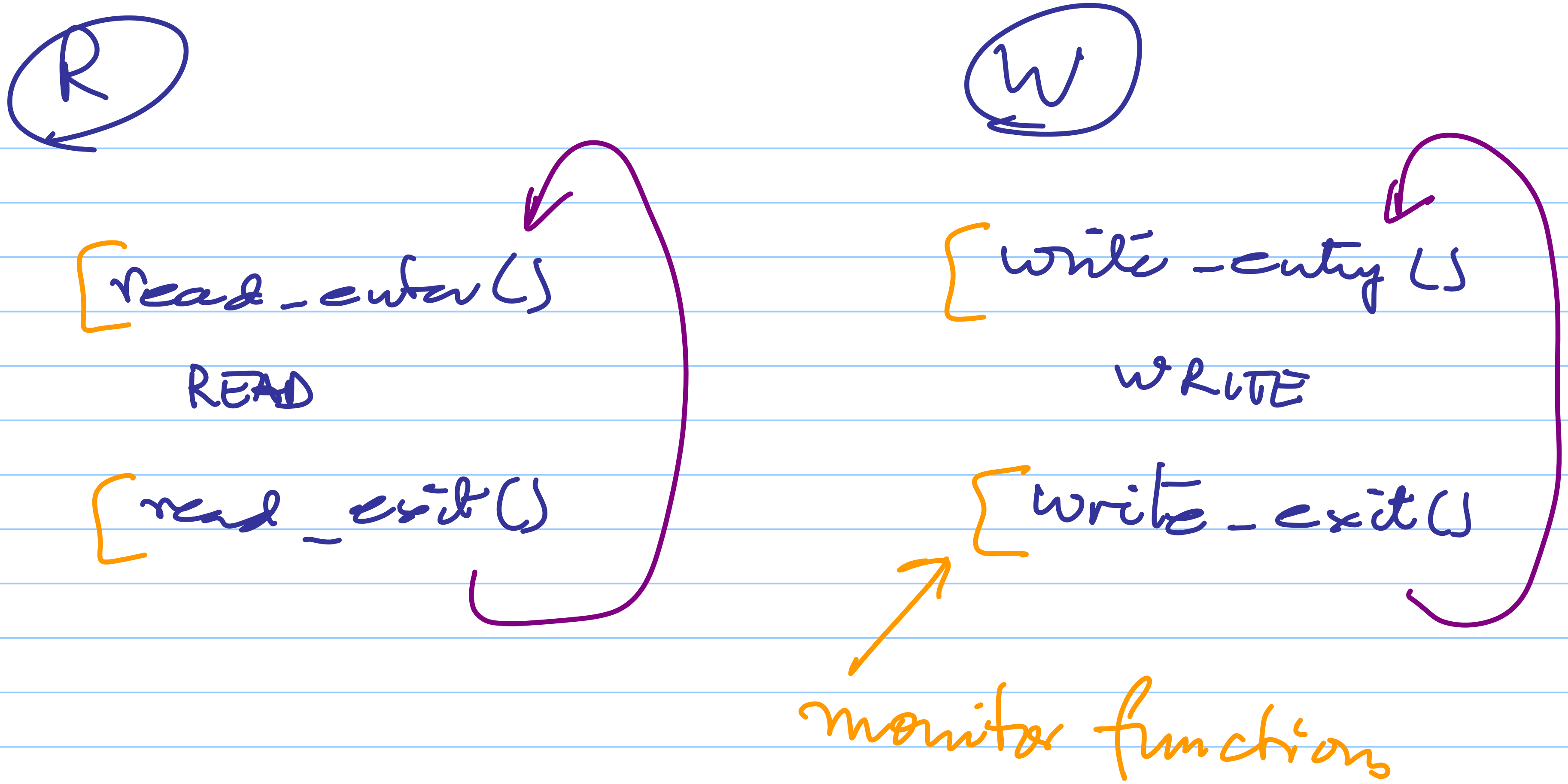
MON

buffer[N], in, out = 0 count = 0
[item] [int] [int]

Conditions → prod, cons

```
add2buff()
{ if (count==N) ● wait (prod)
    count++
    buff[in] = item
    in = (in+1) % N
  ● signal (cons);
}
```

```
get from buff()
{
  if (count == 0)
     ● wait (cons)
     count --
  item=buff[out] u
  out =(out++)%N
● Signal (prod)
}
```



R

read_enter()

READ

read_exit()

W

write_entry()

WRITE

write_exit()

monitor functions

```
READ_ENTER
{ if((wc>0) OR(wwc>0))
        { rwc++  wait (R)  rwc--}
    rc++ ;  signal (R) ;
 }

READ_EXIT
          rc--
    {   if (rc==0) { signal (w) }
        }
```

```
write_enter()
  { if (rc >0) or(wc>0)
          {wwc++; wait(w); wwc--}
      wc++ ↓
    }
  write exit  wc--
    { if(rwc >0) signal(R);
          else signal(w); }
```

```
pickup
while !(chop[LEFT] & chop[RIGHT])
                         wait (self[i]);
  chp[LEFT] = chp[Right] = false

  putdown      chp[L] = chp[R] = true
               signal[L] ; signal[R)
```