Previously, on CS4410...

Reader/Writer Lock Specification

```
1   def RWlock() returns lock:
2       lock = { .nreaders: 0, .nwriters: 0 }
3
4   def read_acquire(rw):
5       atomically when rw→nwriters == 0:
6           rw→nreaders += 1
7
8   def read_release(rw):
9       atomically rw→nreaders -= 1
10
11  def write_acquire(rw):
12      atomically when (rw→nreaders + rw→nwriters) == 0:
13          rw→nwriters = 1
14
15  def write_release(rw):
16      atomically rw→nwriters = 0
```

lock implemented in terms of checks on two variables that must be updated atomically!

Better to assert $rw \rightarrow nreaders > 0$

Busy-Waiting Implementation

```
1   from synch import Lock, acquire, release
2
3   def RWlock() returns lock:
4       lock = { .lock: Lock(), .nreaders: 0, .nwriters: 0 }
5
6   def read_acquire(rw):
7       acquire(?rw→lock)
8       while rw→nwriters > 0:        } Busy
9           release(?rw→lock)         } waiting
10          acquire(?rw→lock)         }
11      rw→nreaders += 1
12      release(?rw→lock)
13
14  def read_release(rw):
15      acquire(?rw→lock)
16      rw→nreaders -= 1
17      release(?rw→lock)
18
19  def write_acquire(rw):
20      acquire(?rw→lock)
21      while (rw→nreaders + rw→nwriters) > 0:
22          release(?rw→lock)
23          acquire(?rw→lock)
24      rw→nwriters = 1
25      release(?rw→lock)
26
27  def write_release(rw):
28      acquire(?rw→lock)
29      rw→nwriters = 0
30      release(?rw→lock)
```

To ensure that nreaders and nwriters are updated atomically, we need to access them in mutual exclusion!

Hence, the implementation of the RWlock includes a mutex lock – to protect accesses to nreaders and nwriters

Waiting with Semaphores

```
1    import synch
2
3    condition = BinSema(True)
4
5    def T0():
6        acquire(?condition)
7
8    def T1()
9        release(?condition)
10
11
12   spawn(T0)
13   spawn(T1)
```

By initializing a semaphore to "acquired" (i.e., True) we can force a thread to wait

What else can we do with binary semaphores?

Conditional Critical Sections

- A critical section with an associated condition
  - queue.get(), but wait until queue is not empty
    - don't want two threads to run code at the same time
    - don't want any thread to run queue.get() when the queue is empty
  - print(), but wait until printer is idle
  - RW.read_acquire(), but only when there are no writers in the critical section

One Critical Section, multiple conditions

- Some conditional critical sections can have multiple conditions:
  - R/W lock
    - readers are waiting for writers to leave
    - writers are waiting for readers and writers to leave
  - bounded queue
    - dequeuers waiting for queue to be not empty
    - enqueuers waiting for queue to be not full
  - ...

High level idea: selective baton passing

- To execute inside the CS, thread needs the baton
- Threads can be waiting for various conditions
  - while they do, they don't hold the baton
- When a thread with the baton leaves the CS, it checks whether there are threads waiting for a condition that now holds
- If so, it passes the baton to one such thread
- If not, the CS is vacated, and the baton can be picked up by another thread when it comes along

Split Binary Semaphores

Hoare 1973

- Implement baton passing with multiple binary semaphores
- $\mathcal{N}$ conditions require $\mathcal{N} + 1$ binary semaphores
  - one of each condition
  - one to enter the CS in the first place

Split Binary Semaphores

Hoare 1973

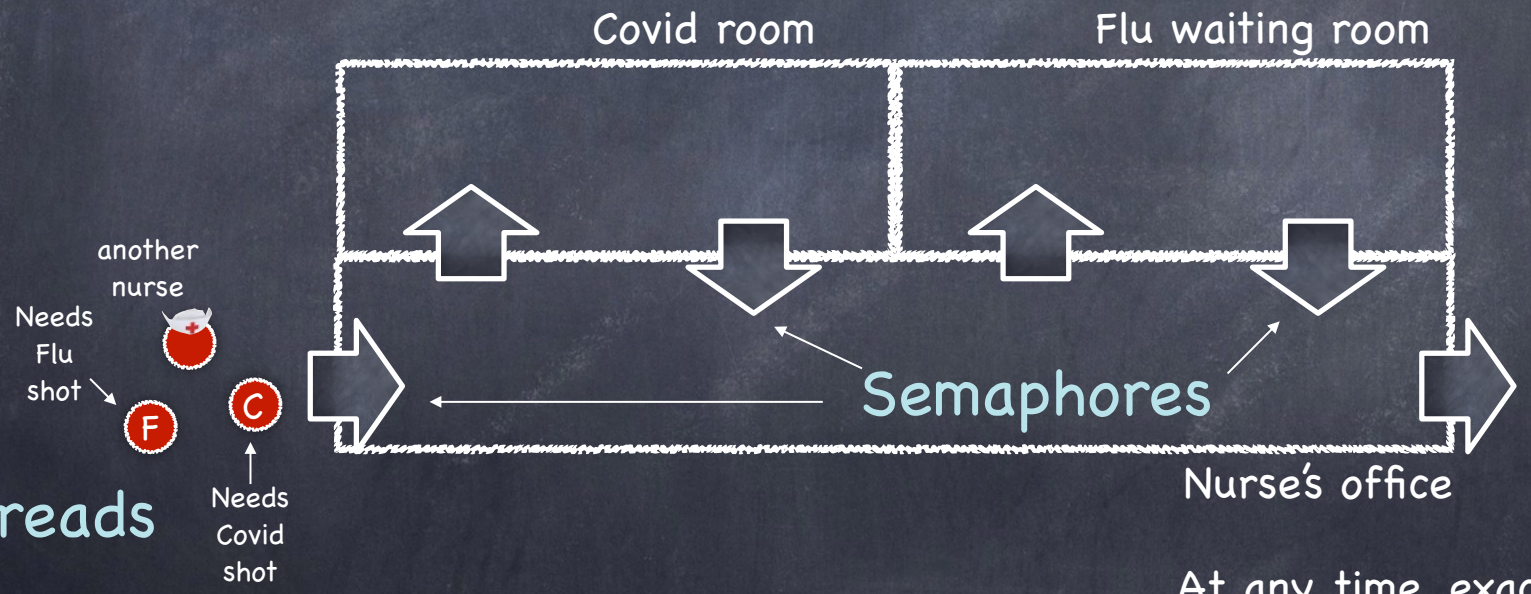
- Invariant: At most one of these semaphores is released (i.e., its value is False)
  - If all are acquired (True), baton held by some thread (some thread in CS)
  - If one is released (False), no thread holds baton (CS is empty)
    - if it is the "entry" semaphore, no thread is waiting on a condition that holds—any thread can enter CS
    - if it is one of the condition semaphores, some thread waiting on that condition can enter CS

Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)



Nurse's office: critical section

Rooms: waiting conditions

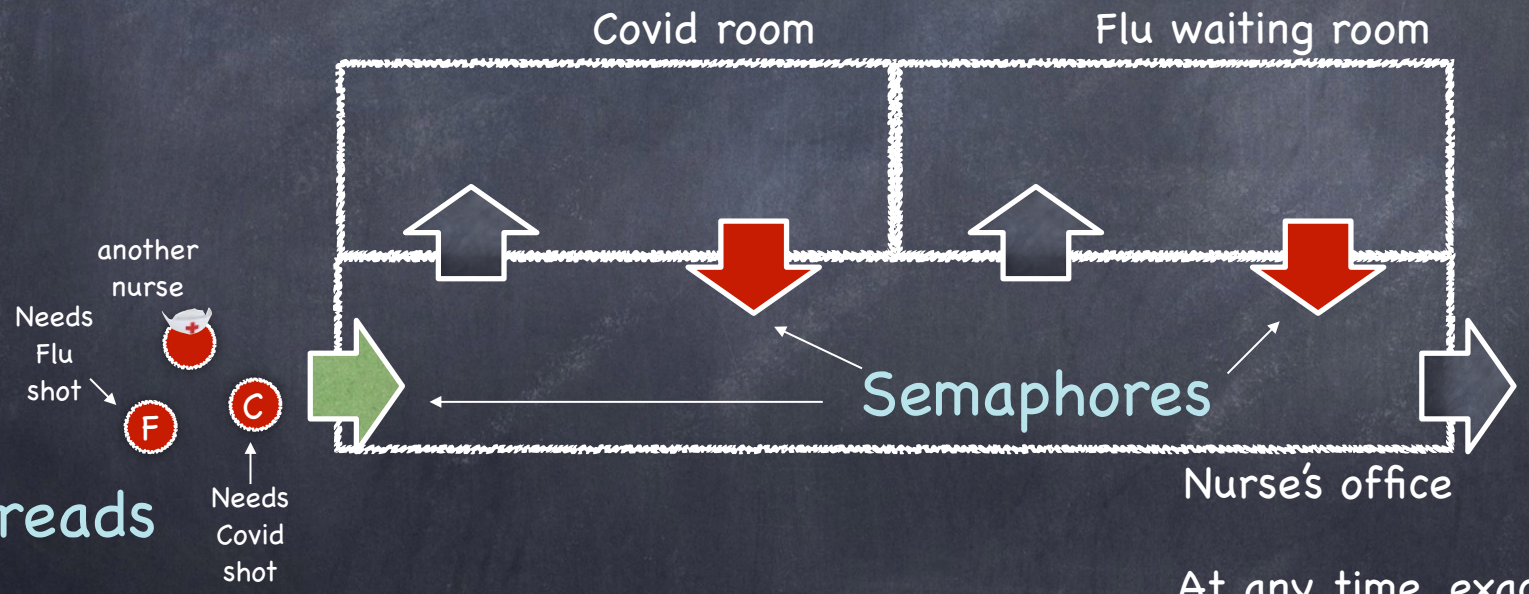
At any time, exactly one semaphore or thread is green

Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)



Nurse's office: critical section

Rooms: waiting conditions

At any time, exactly one semaphore or thread is green

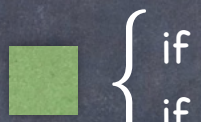
(and thus, at most one semaphore is green (Invariant))

What this models

- Reader/writer lock
  - Nurse's office: critical section
  - Waiting Room 1: readers waiting for writer to leave
  - Waiting Room 2: writers waiting for readers and writer to leave
- Bounded queue
  - Nurse's office: critical section
  - Waiting Room 1: dequeuers waiting for non-empty queue
  - Waiting Room 2: enqueuers waiting for non-full queue
- ...

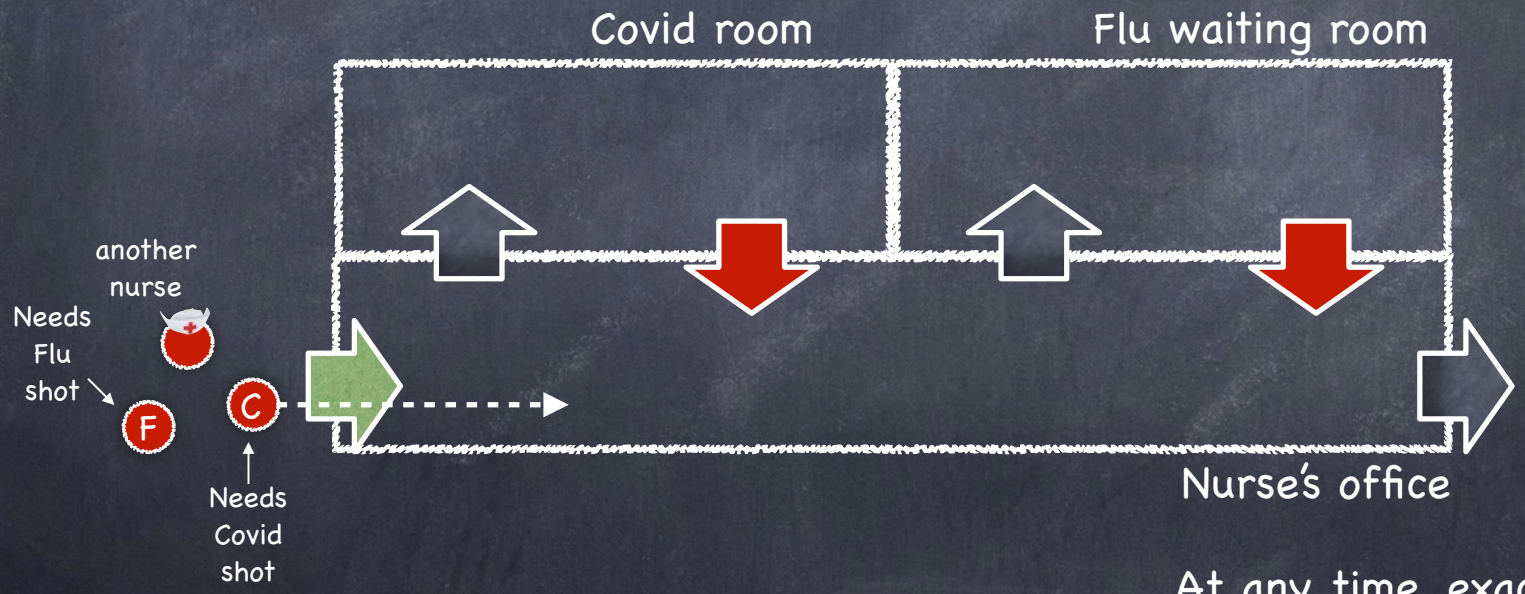
Jabs…

Nurse administers C and F vaccines, one patient at a time



if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)



Nurse's office: critical section

Rooms: waiting conditions

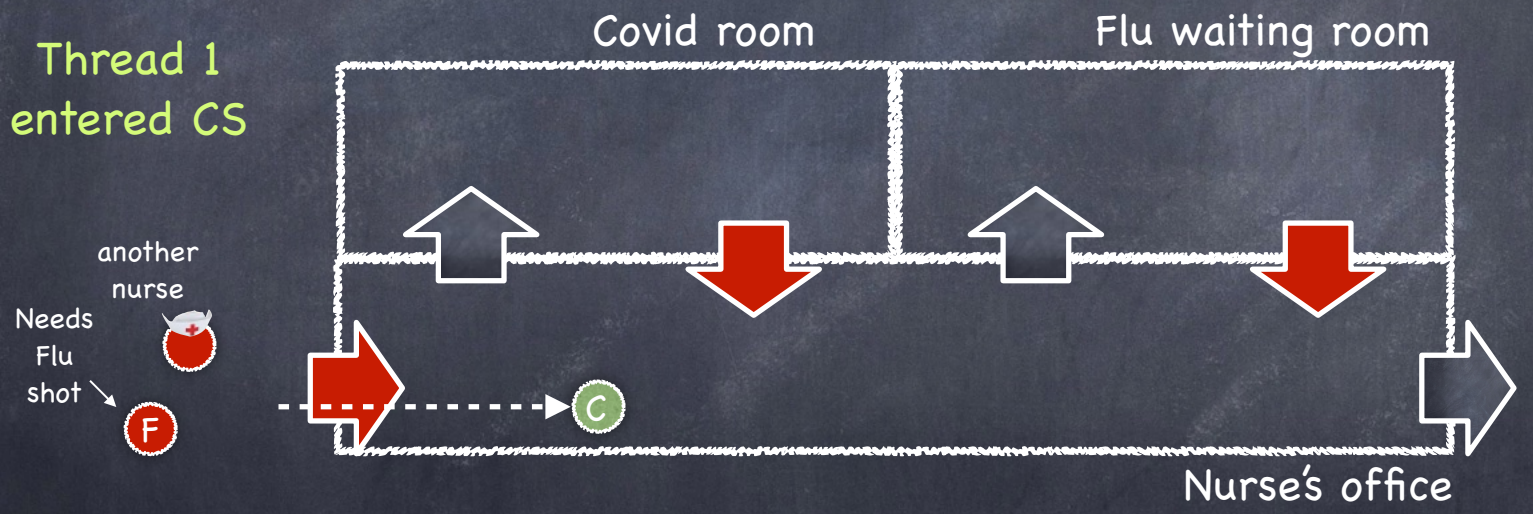
At any time, exactly one semaphore or thread is green

Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)



Nurse's office: critical section

Rooms: waiting conditions

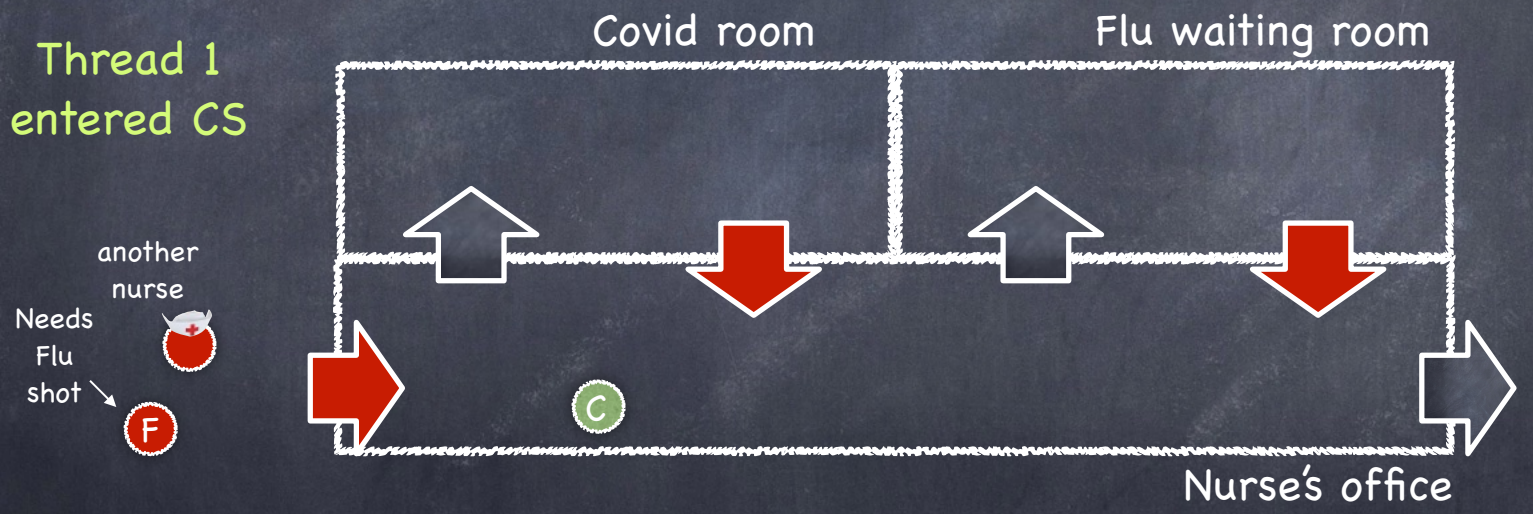
At any time, exactly one semaphore or thread is green

Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)



Nurse's office: critical section

Rooms: waiting conditions

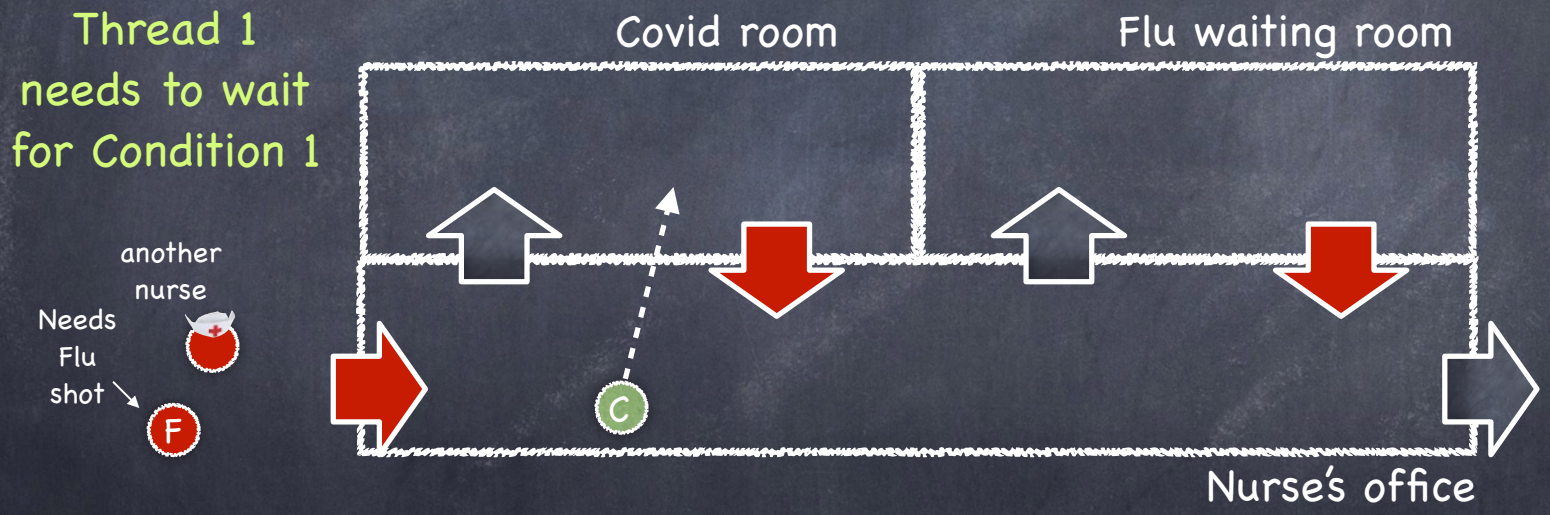
At any time, exactly one semaphore or thread is green

Jabs...

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)



Nurse's office: critical section

Rooms: waiting conditions

At any time, exactly one semaphore or thread is green

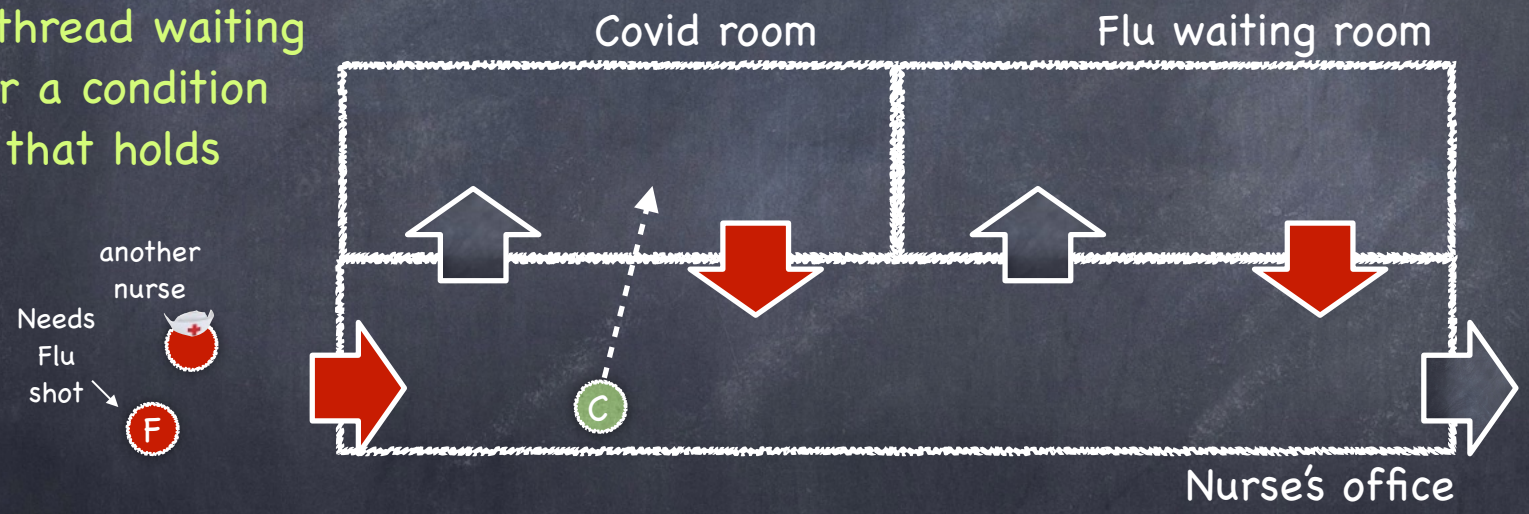
Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)

No thread waiting for a condition that holds



Nurse's office: critical section

Rooms: waiting conditions

At any time, exactly one semaphore or thread is green

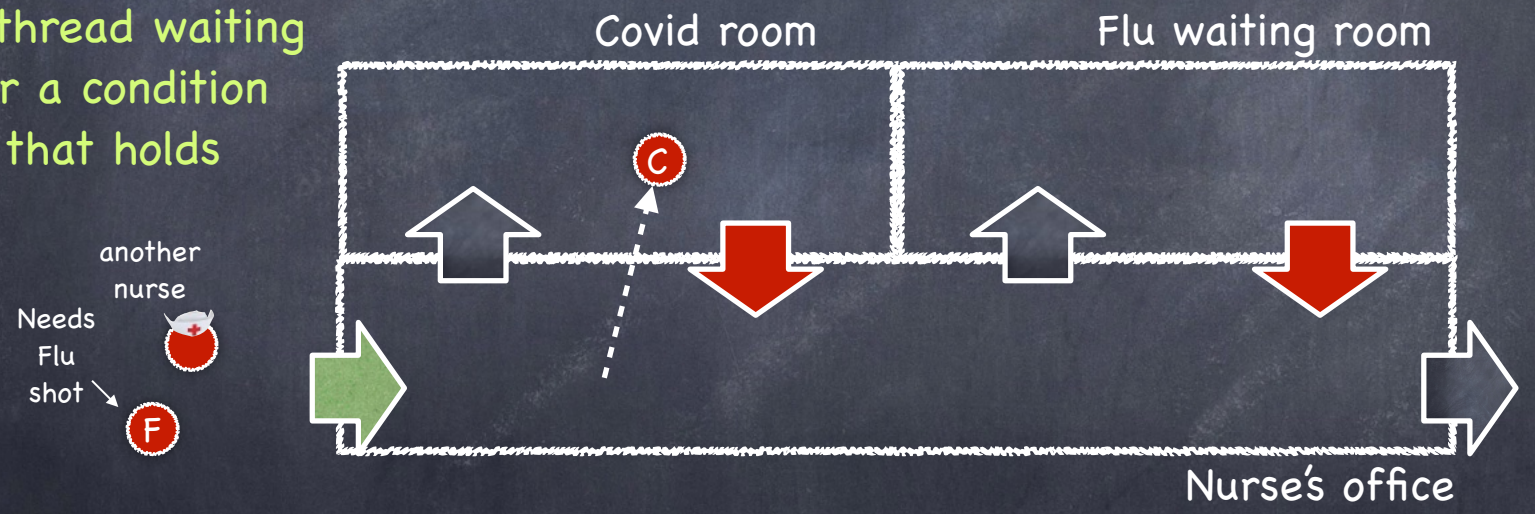
Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)

No thread waiting for a condition that holds



Nurse's office: critical section

Rooms: waiting conditions

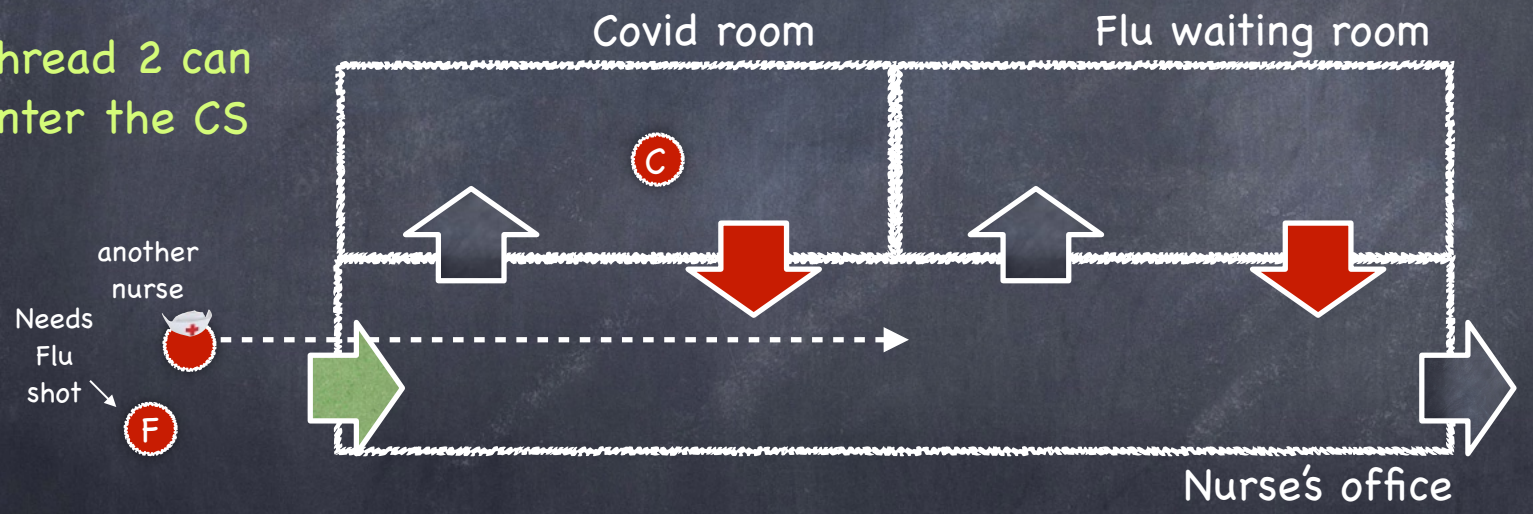
At any time, exactly one semaphore or thread is green

Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)



Nurse's office: critical section

Rooms: waiting conditions

At any time, exactly one semaphore or thread is green

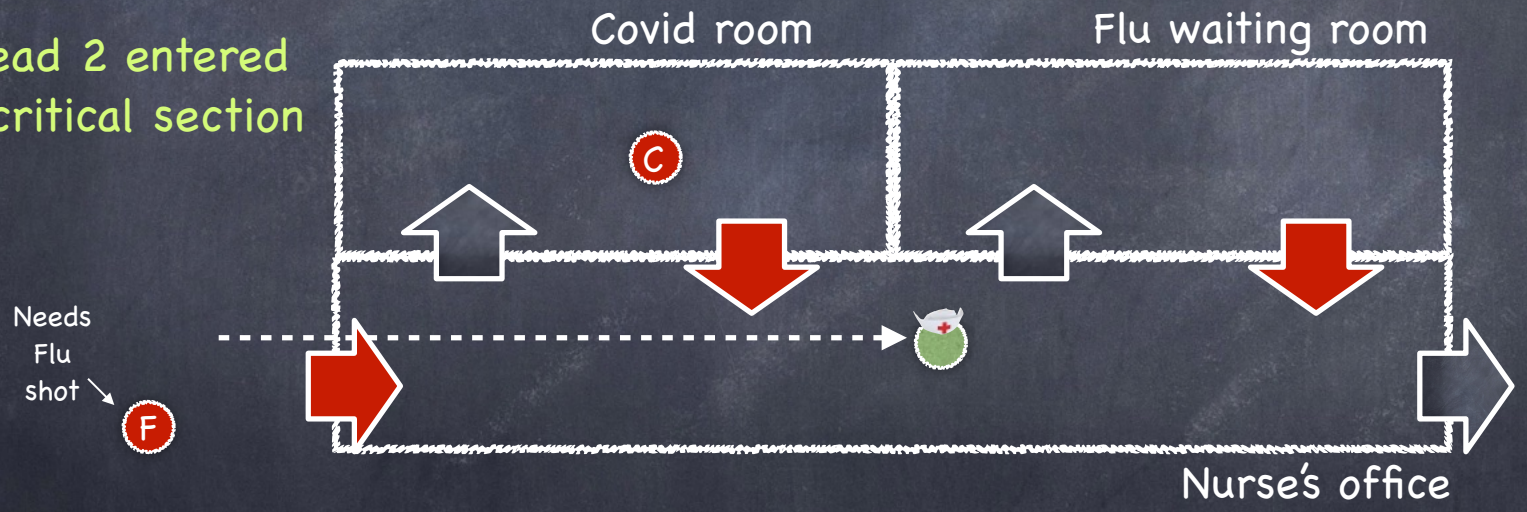
Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)

Thread 2 entered the critical section



Nurse's office: critical section

Rooms: waiting conditions

At any time, exactly one semaphore or thread is green

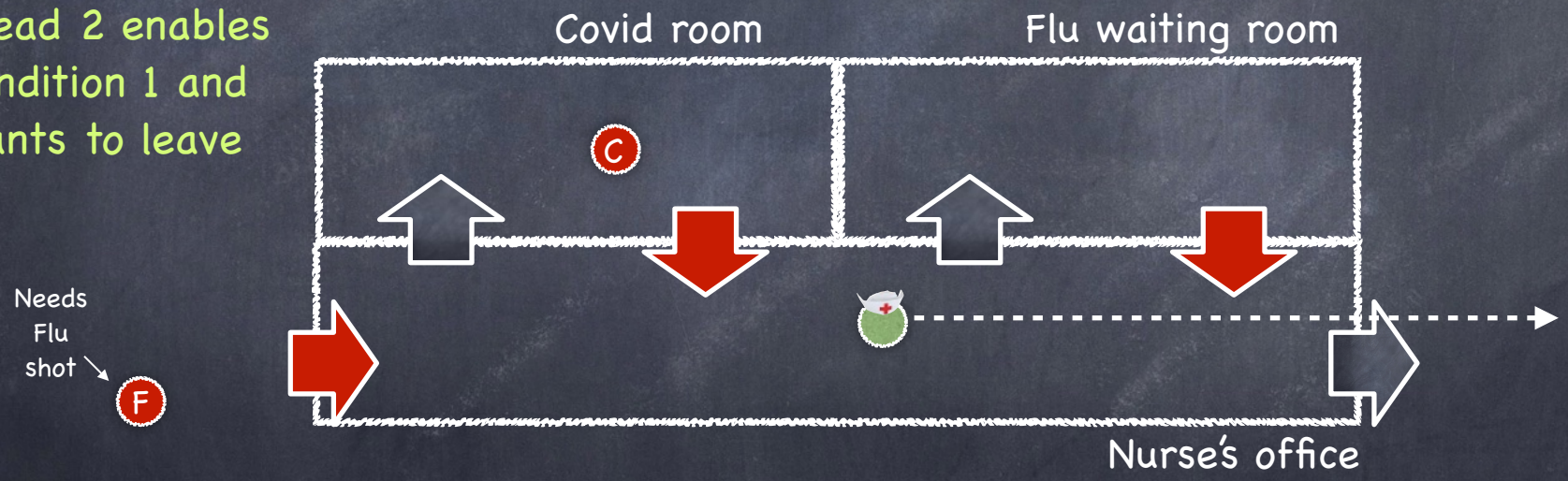
Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)

Thread 2 enables Condition 1 and wants to leave



Nurse's office: critical section

Rooms: waiting conditions

At any time, exactly one semaphore or thread is green

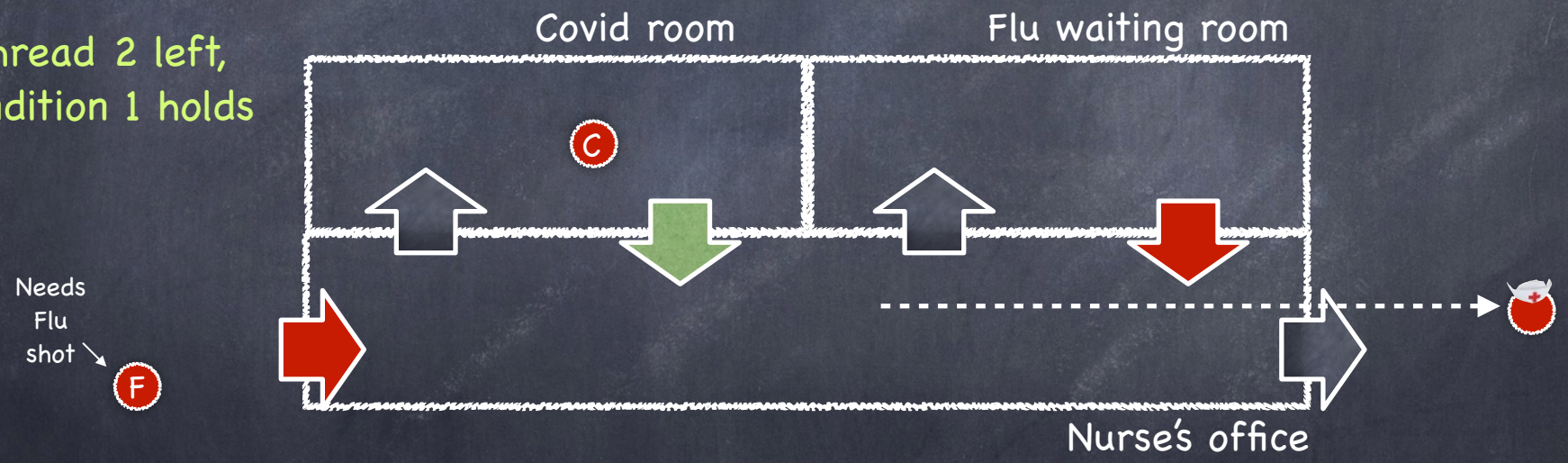
Jabs…

Nurse administers C and F vaccines, one patient at a time

- if thread, in CS
- if sema, released (False)

- if thread, outside CS
- if sema, acquired (True)

Thread 2 left,
Condition 1 holds



Nurse's office: critical section

Rooms: waiting conditions

At any time, exactly one semaphore or thread is green

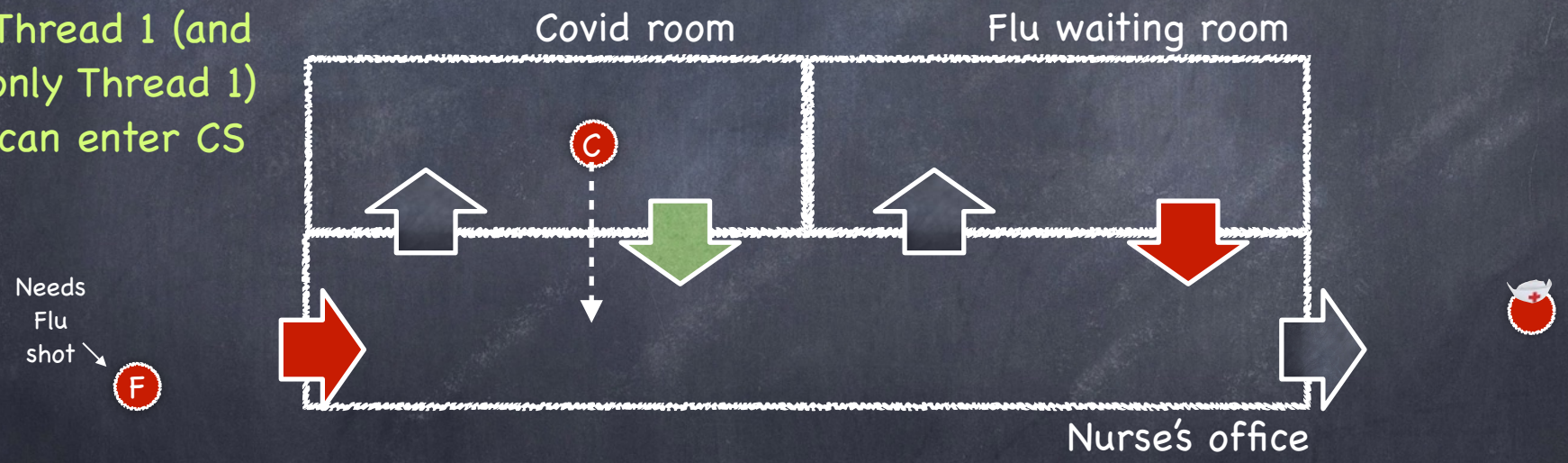
Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)

Thread 1 (and only Thread 1) can enter CS



Nurse's office: critical section

Rooms: waiting conditions

At any time, exactly one semaphore or thread is green

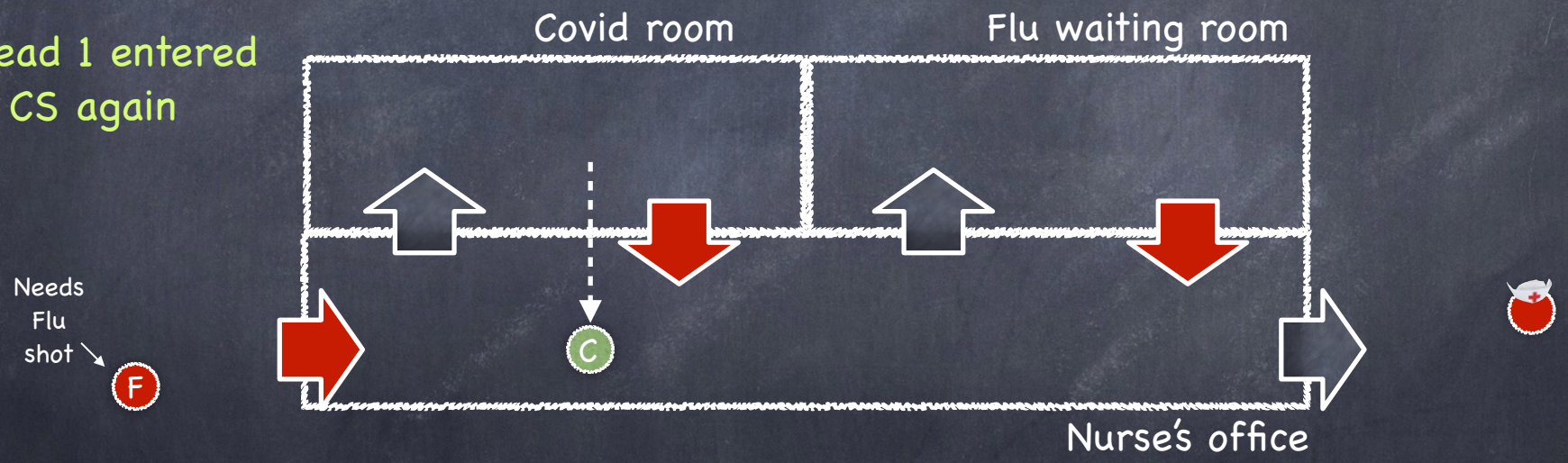
Jabs...

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)

Thread 1 entered CS again



Nurse's office: critical section

Rooms: waiting conditions

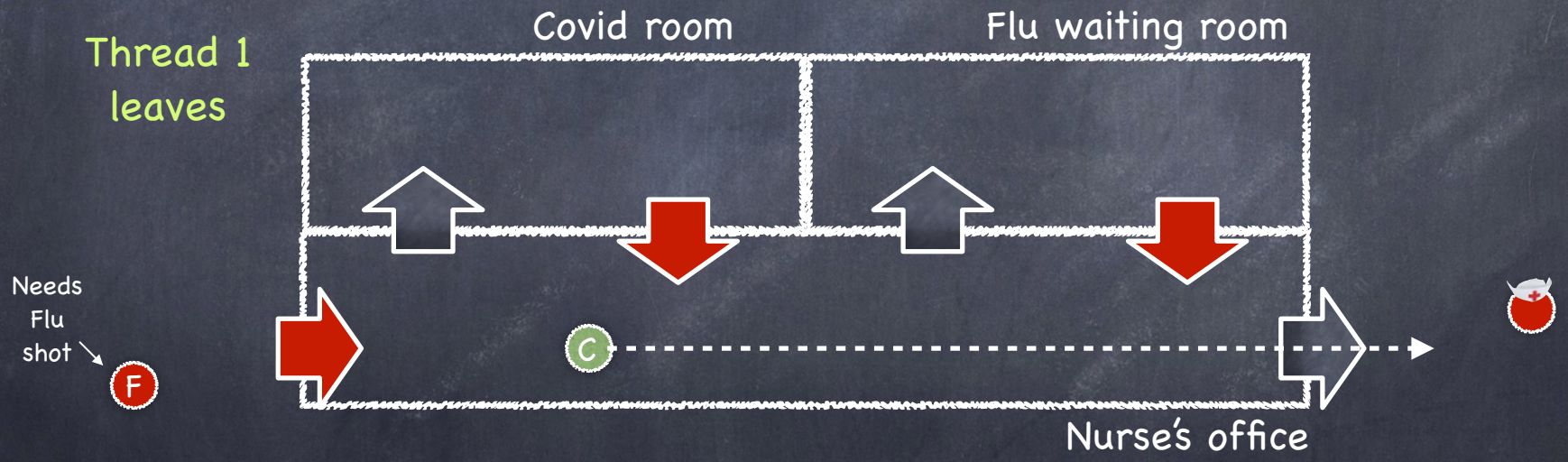
At any time, exactly one semaphore or thread is green

Jabs...

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)



Nurse's office: critical section

Rooms: waiting conditions

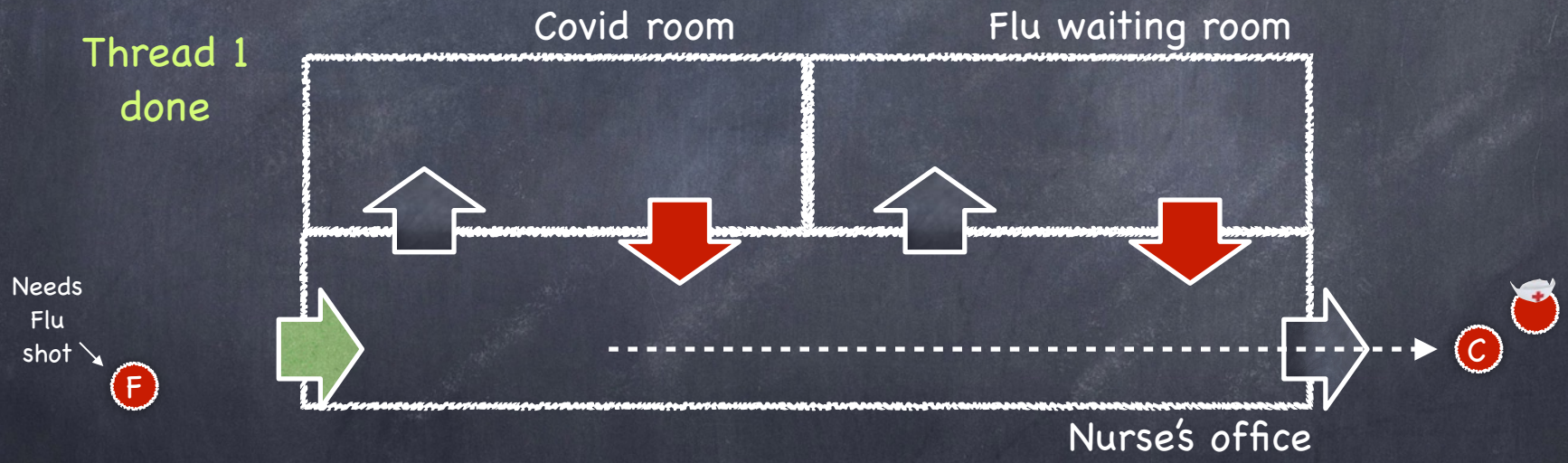
At any time, exactly one semaphore or thread is green

Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)



Nurse's office: critical section

Rooms: waiting conditions

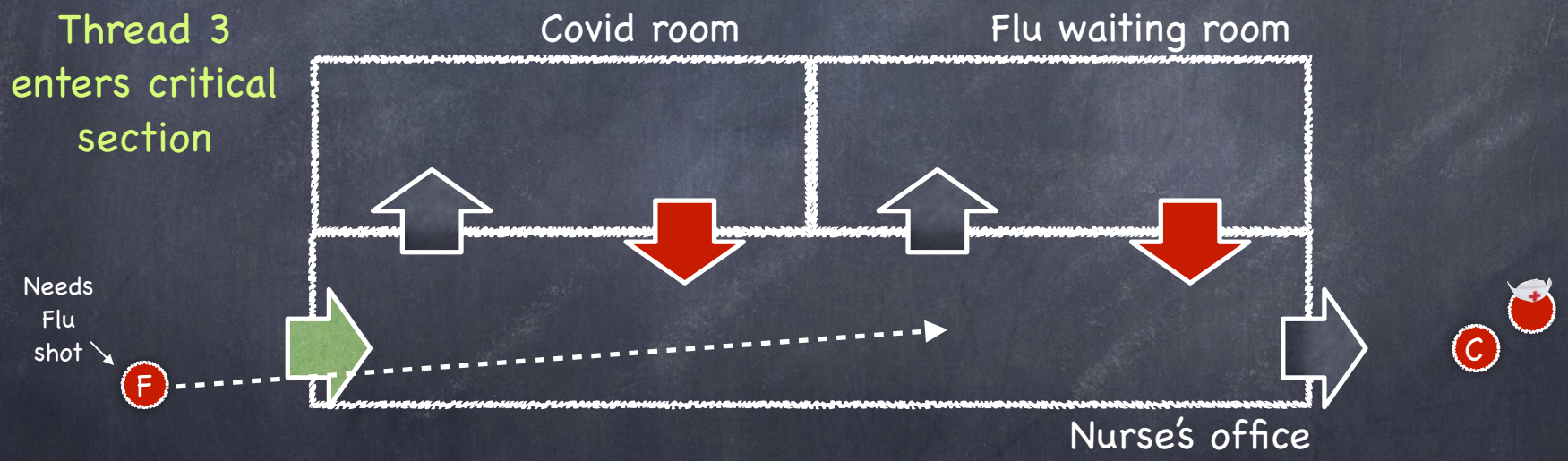
At any time, exactly one semaphore or thread is green

Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)



Nurse's office: critical section

Rooms: waiting conditions

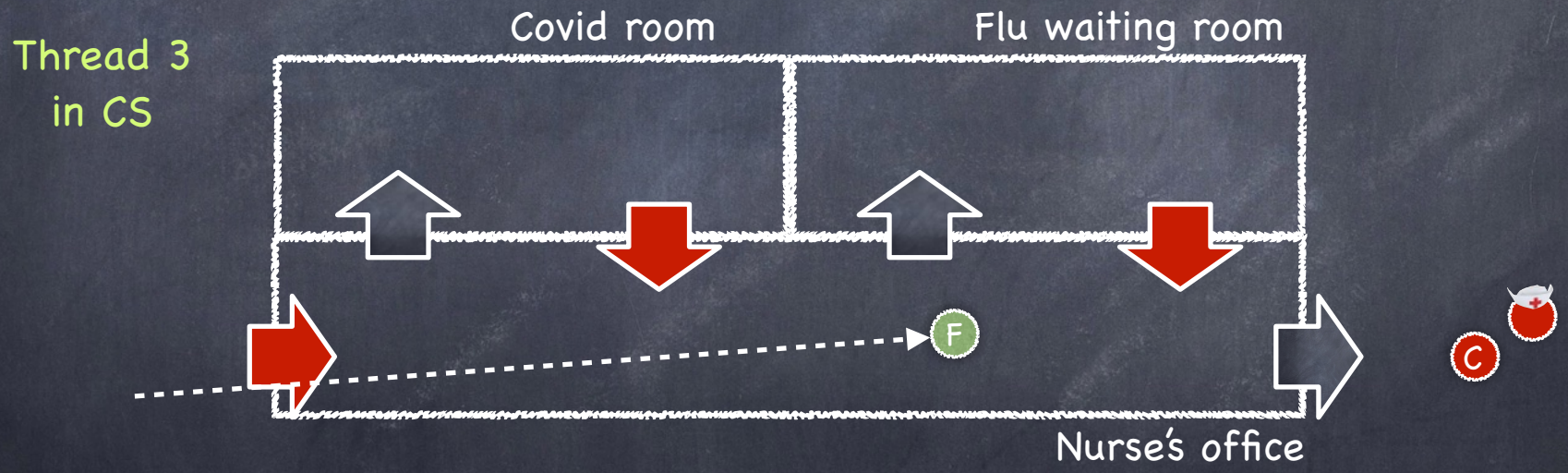
At any time, exactly one semaphore or thread is green

Jabs…

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)



Nurse's office: critical section

Rooms: waiting conditions

At any time, exactly one semaphore or thread is green

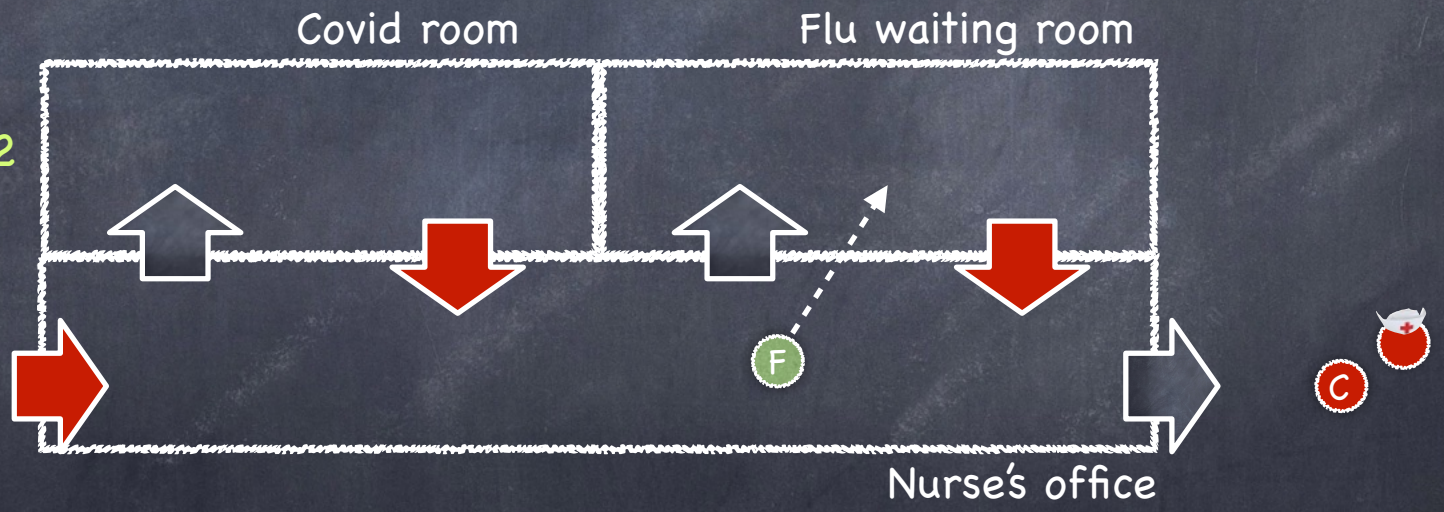
Jabs...

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)

Thread 3 needs to wait for Condition 2



Nurse's office: critical section

Rooms: waiting conditions

At any time, exactly one semaphore or thread is green

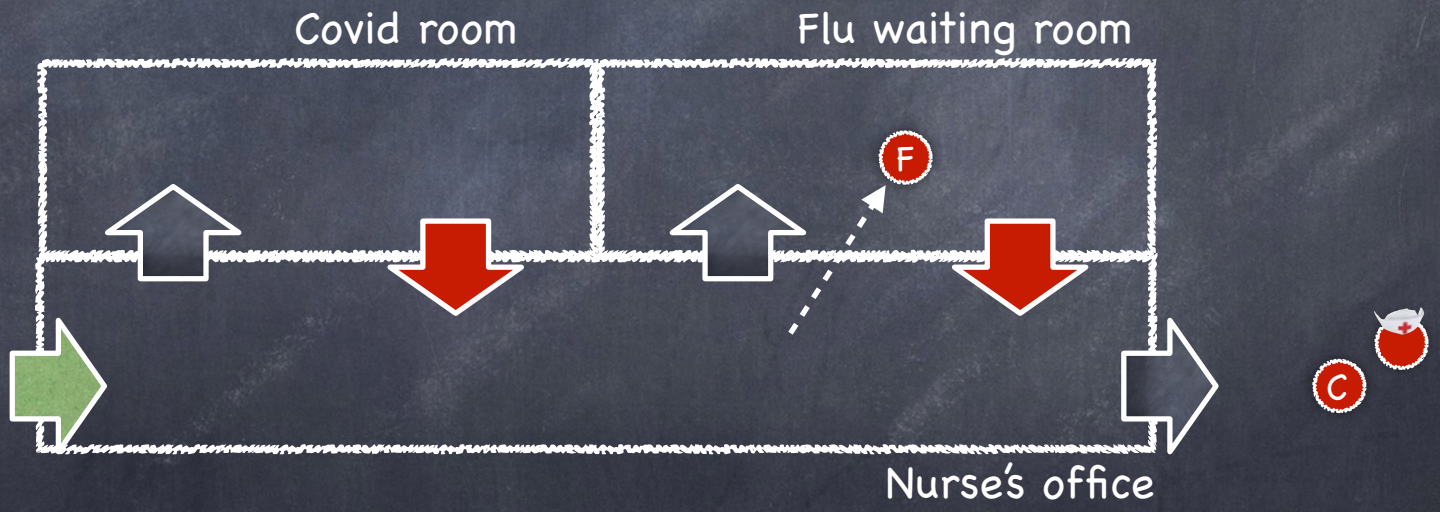
Jabs...

Nurse administers C and F vaccines, one patient at a time

if thread, in CS
if sema, released (False)

if thread, outside CS
if sema, acquired (True)

Thread 3 waiting for Condition 2



Nurse's office: critical section

Rooms: waiting conditions

At any time, exactly one semaphore or thread is green

Reader/Writer Lock Specification (again)

```
1   def RWlock() returns lock:
2       lock = { .nreaders: 0, .nwriters: 0 }
3
4   def read_acquire(rw):
5       atomically when rw→nwriters == 0:
6           rw→nreaders += 1
7
8   def read_release(rw):
9       atomically rw→nreaders -= 1
10
11  def write_acquire(rw):
12      atomically when (rw→nreaders + rw→nwriters) == 0:
13          rw→nwriters = 1
14
15  def write_release(rw):
16      atomically rw→nwriters = 0
```

Better to assert $rw \rightarrow nreaders > 0$

Reader/Writer Lock: Implementation

```
1   from synch import BinSema, acquire, release
2
3   def RWlock() returns lock:
4       lock = {
5               .nreaders: 0, .nwriters: 0, .mutex: BinSema(False),
6               .r_gate: { .sema: BinSema(True), .count: 0 },
7               .w_gate: { .sema: BinSema(True), .count: 0 }
8           }
```

Accounting

- $nreaders$: #readers in the CS
- $r\_gate\,.\,count$: #readers waiting to enter CS
- $nwriters$: #writers in the CS
- $w\_gate\,.\,count$: #writers waiting to enter CS

Invariants

- If n readers in the critical section, then $nreaders \geq n$
- If n writers in the critical section, then $nwriters \geq n$
- $\vee\ (nreaders \geq 0 \wedge nwriters = 0)$
  $\vee\ (nreaders = 0 \wedge nwriters = \leq 1)$

Reader/Writer Lock: Implementation

```
18      def read_acquire(rw):
19          acquire(?rw→mutex)
            if rw→nwriters > 0:
21              rw→r_gate.count += 1; release_one(rw)
                acquire(?rw→r_gate.sema); rw→r_gate.count −= 1
23          rw→nreaders += 1
24          release_one(rw)
25
26      def read_release(rw):
            acquire(?rw→mutex); rw→nreaders −= 1; release_one(rw)
```

enter main gate

waiting condition

leave

enter reader gate

entering RW CS

leave: let others try too

Note: acquire and release operations alternate

no special waiting condition

Reader/Writer Lock: Implementation

```
29      def write_acquire(rw):
30          acquire(?rw→mutex)
31          if (rw→nreaders + rw→nwriters) > 0:
32              rw→w_gate.count += 1; release_one(rw)
33              acquire(?rw→w_gate.sema); rw→w_gate.count −= 1
34          rw→nwriters += 1
35          release_one(rw)
36
37      def write_release(rw):
38          acquire(?rw→mutex); rw→nwriters −= 1; release_one(rw)
```

enter main gate

waiting condition

enter writer gate

Similar structure to read_acquire()

Reader/Writer Lock: Implementation

when leaving the critical section:

```
10      def release_one(rw):
11          if (rw→nwriters == 0) and (rw→r_gate.count > 0):
12              release(?rw→r_gate.sema)
13          elif ((rw→nreaders + rw→nwriters) == 0) and (rw→w_gate.count > 0):
14              release(?rw→w_gate.sema)
15          else:
16              release(?rw→mutex)
```

If no writers in the Critical Section and there are readers waiting

then let a reader in!

Reader/Writer Lock: Implementation

when leaving the critical section:

```
10      def release_one(rw):
11          if (rw→nwriters == 0) and (rw→r_gate.count > 0):
12              release(?rw→r_gate.sema)
13          elif ((rw→nreaders + rw→nwriters) == 0) and (rw→w_gate.count > 0):
14              release(?rw→w_gate.sema)
15          else:
16              release(?rw→mutex)
```

If no writers in the Critical Section and there are readers waiting

then let a reader in!

Reader/Writer Lock: Implementation

when leaving the critical section:

```
10      def release_one(rw):
11          if (rw→nwriters == 0) and (rw→r_gate.count > 0):
12              release(?rw→r_gate.sema)
13          elif ((rw→nreaders + rw→nwriters) == 0) and (rw→w_gate.count > 0):
14              release(?rw→w_gate.sema)
15          else:
16              release(?rw→mutex)
```

If no readers nor writers in the Critical Section and there are writers waiting

then let a writer in!

Reader/Writer Lock: Implementation

when leaving the critical section:

```
10      def release_one(rw):
11          if (rw→nwriters == 0) and (rw→r_gate.count > 0):
12              release(?rw→r_gate.sema)
13          elif ((rw→nreaders + rw→nwriters) == 0) and (rw→w_gate.count > 0):
14              release(?rw→w_gate.sema)
15          else:
16              release(?rw→mutex)
```

If no readers nor writers in the Critical Section and there are writers waiting

then let a writer in!

Reader/Writer Lock: Implementation

when leaving the critical section:

```
10      def release_one(rw):
11          if (rw→nwriters == 0) and (rw→r_gate.count > 0):
12              release(?rw→r_gate.sema)
13          elif ((rw→nreaders + rw→nwriters) == 0) and (rw→w_gate.count > 0):
14              release(?rw→w_gate.sema)
15          else:
16              release(?rw→mutex)
```

Otherwise...

let anyone in!

Reader/Writer Lock: Implementation

when leaving the critical section:

```
10      def release_one(rw):
11          if (rw→nwriters == 0) and (rw→r_gate.count > 0):
12              release(?rw→r_gate.sema)
13          elif ((rw→nreaders + rw→nwriters) == 0) and (rw→w_gate.count > 0):
14              release(?rw→w_gate.sema)
15          else:
16              release(?rw→mutex)
```

Can these two conditions be reversed?

What is the effect of that?

Reader/Writer Lock: Implementation

when leaving the critical section:

```
10      def release_one(rw):
11          if (rw→nwriters == 0) and (rw→r_gate.count > 0):
12              release(?rw→r_gate.sema)
13          elif ((rw→nreaders + rw→nwriters) == 0) and (rw→w_gate.count > 0):
14              release(?rw→w_gate.sema)
15          else:
16              release(?rw→mutex)
```

What happens if multiple readers are waiting and a writer leaves?

Does it let all the readers in or just one?

Reader/Writer Lock: Implementation

```
18      def read_acquire(rw):
19          acquire(?rw→mutex)
20          if rw→nwriters > 0:
21              rw→r_gate.count += 1; release_one(rw)
22              acquire(?rw→r_gate.sema); rw→r_gate.count −= 1
23          rw→nreaders += 1
24          release_one(rw)
25
26      def read_release(rw):
27          acquire(?rw→mutex); rw→nreaders −= 1; release_one(rw)
```

A Hierarchy of Critical Sections

- Again, we have two different critical sections...
- ...that occur at different levels of abstraction
  - the first relies a R/W lock
    - protects access to some shared object (say, a DB)
    - allows multiple readers in the CS
  - the second relies on split binary semaphores
    - protects the shared variables ($nreaders$, $r\_gate.count$, etc) and implements the conditions we use to implement R/W locks
    - allows only one thread at a time in its CS

Starvation

- Our R/W implementation can starve writers
- Change the waiting and release conditions:
  - when a reader tries to enter CS, wait if there is
    - a writer in CS or
    - writers at the write gate waiting to enter CS
  - exiting reader prioritizes releasing a waiting writer
  - exiting writer prioritizes releasing a waiting reader

See Chapter 17 in the Harmony book

Conditional Critical Sections

- We know of two ways to implement them:

Busy Waiting	Split Binary Semaphores
Wait for condition in loop, acquiring lock before testing for condition, and releasing it if condition does not hold	Use a collection of binary semaphores and keep track of state, including information about waiting threads
Easy to understand the code	State tracking is complicated
OK-ish for true multi-core, but bad for virtual threads	Good for both multicore and virtual threading

Language support?

- Can the programming language be more helpful here?
  - Offer some helpful syntax
  - or at least some library support

Enter Monitors

- Collect shared data into an object/module
- Define methods for accessing shared data
- Separate the concerns of mutual exclusion and condition synchronization
- Monitors are comprised of
  - one mutex lock, and
  - zero or more condition variables for managing concurrent access to shared data

Condition Variables

- An abstraction for conditional synchronization associated with a monitor
- Enable threads to wait for a given condition to hold while inside the monitor (after releasing the monitor lock) and be alerted when the condition holds
- Condition variable is a misnomer
  - can neither be read nor set to a value
  - think of a condition variable as a label associated with a condition and a queue
  - threads wait in the queue (inside the monitor) until notified that condition holds

Resource Variables

- Each condition variable should be associated with a resource variable (RV) tracking the state of the resource that determines whether the condition holds
  - e.g., tin a bounded buffer he number of buffer slots that have been filled
  - It is your job to maintain the RV!
- Check its RV before calling wait() on a condition variable to ensure the resource is truly unavailable
- Once the resource is available, claim it (subtract the amount you are using!)
- Before notifying you are releasing a resource, indicate it has become available by increasing the corresponding RV

Two Types of Monitors

Hoare Monitors

Tony Hoare

Mesa Monitors

Butler Lampson

Different semantics as to what happens when a thread waiting on a condition is alerted that the condition holds