

# HW 12

Turn in a .txt file showing the state of the queues after each step or fill in this, scan it and turn it in.

This homework very much based on the example at

<http://stackoverflow.com/questions/37026/java-notify-vs-notifyall-all-over-again>

It contains a detailed explanation of what happens -- look at it and the solution *after* doing this yourself

- In this homework you will show the status of the program as *wait*, *notify* and *notifyAll* calls are made. Positions in the *wait* and *blocked queue* do not matter.
- This homework will demonstrate how threads wait on locks, how *wait*, *notify* and *notifyAll* work, and why you need a *while* loop to recheck a condition.

# wait, notify and notifyAll

## Let $L$ be the lock

### **notify**

If one or more threads are in the *wait queue*, wake one up and place it into the *blocked queue*. The thread to be woken up is picked arbitrarily. The thread woken up must acquire  $L$  before continuing. The thread executing *notify* must hold the lock  $L$  and continues to hold it until it reaches the end of the synchronized block.

### **notifyAll**

If one or more threads are in the *wait queue*, wake all of them up. All woken up threads will be placed into the *blocked queue* and attempt to acquire  $L$  when it is released by the thread executing the *notifyAll*. At most one will get the lock, all others will continue to be in the *blocked queue* (not the *wait queue*!) The thread executing *notifyAll* must hold the lock  $L$  and continues to hold it until it reaches the end of the synchronized block.

### **Wait**

Put the thread executing *wait* into  $L$ 's *wait queue*. The thread executing *wait* must hold  $L$  and releases it when it executes *wait*.

# First scenario -- this code is part of a class that implements a blocking queue

```
public synchronized void put(Object o) {
    while (buf.size()==MAX_SIZE) {
        wait(); // called if the buffer is full (try/catch removed
                // for brevity)
    }
    buf.add(o);
    notify(); // called in case there are any getters or putters waiting
}

public synchronized Object get() {
    // Y: this is where C2 tries to acquire the lock (i.e. at the
    // beginning of the method)
    while (buf.size()==0) {
        wait(); // called if the buffer is empty (try/catch removed
                // for brevity)
        // X: this is where C1 tries to re-acquire the lock (see below)
    }
    Object o = buf.remove(0);
    notify(); // called if there are any getters or putters waiting
    return o;
}
```

## Queue object

*buf*



lock's *wait queue*



lock's *blocked queue*



There are two kinds of threads -- consumer threads *C1*, *C2*, ..., that remove characters from *buf*, and producer threads *P1*, *P2*, ..., that add characters to *buf*. For our purposes, *buf.size()* returns the number of characters in the buffer.

The buffer *buf* is initially empty.

1. Consumer *C1* enters the synchronized block for the *get* method
2. *buf.size() == 0* is true
3. *wait()* is executed, placing *C1* on the lock's *wait queue*

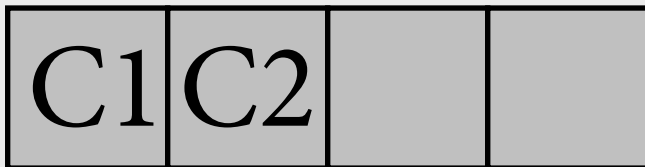
**Show the status of *buf*, lock's *wait queue* and lock's *blocked queue*.**

## Queue object

*buf*



lock's *wait queue*



lock's *blocked queue*

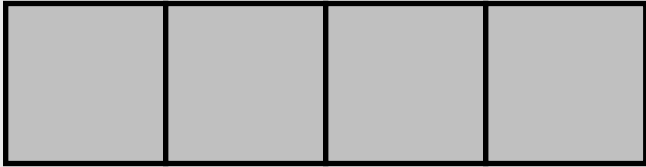


1. Consumer 2 (*C2*) is just about to enter the synchronized block for the *get* method, but has not acquired the lock.
2. Producer *P1* enters the synchronized method *put*, acquires the lock, places the character “*c*” into *buf*, and calls *notify()*.
3. *C1* is woken up by the notify and must reacquire the lock before proceeding. Thus both *C1* and *C2* are competing for the lock.

**Show the status of *buf*, lock's *wait queue* and lock's *blocked queue*.**

## Queue object

*buf*



lock's *wait queue*



lock's *blocked queue*



1. One of *C1* and *C2* is non-deterministically chosen to get the lock. Let's say *C2* gets the lock. It gets to enter the method since *C1* is awake it is put on the *blocked queue*, not back on the *wait queue*.
2. *C2* gets the character and releases the lock which is then acquired by *C1*.

**Is there a character in *buf* for *C1* to get? No**

**What will happen in the program as written?** *C1* will wait until a new producer comes along

**What would have happened if the *while* loop was not in the *get()* method?**

The first item of buffer would have been removed, but no item would have been present

Let's look at a scenario that shows the need for `notifyAll` instead of `notify` in the code.

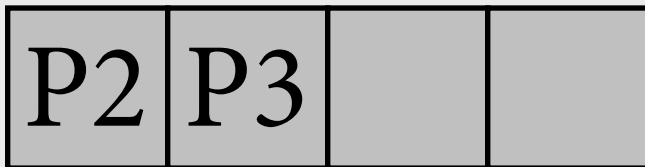


## Queue object

*buf*



lock's *wait queue*



lock's *blocked queue*



To make this easy, assume a buffer size of 1. Producer and consumer threads are named as before. *buf* is initially empty.

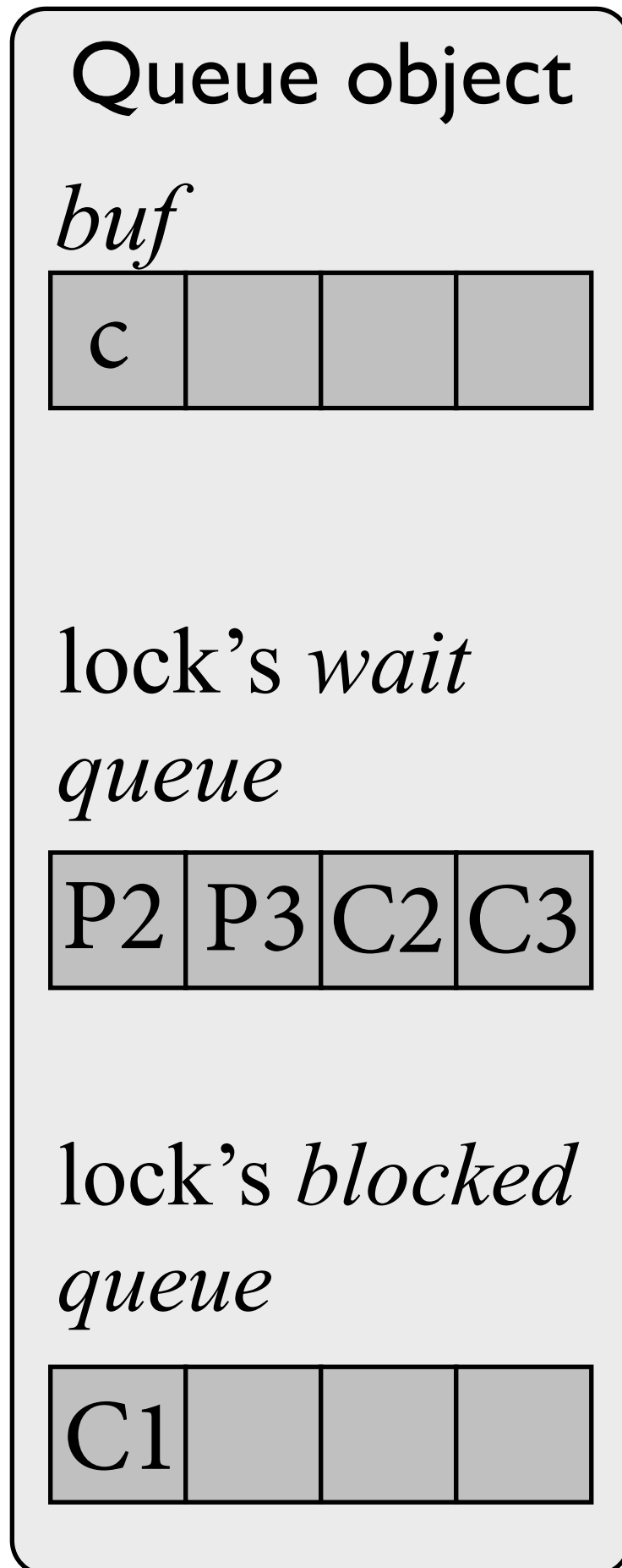
1. *P1* puts a “c” into the buffer.
2. *P2* attempts a put, checks the while loop and performs a *wait()*
3. *P3* attempts a put, checks the *while* loop and performs a *wait()*

**Show the status of *buf*, lock's *wait queue* and lock's *blocked queue*.**

4. The following happen at time step 4:

- a. *C1* attempt to get 1 character and enters the *get* method;
- b. *C2* attempts to get 1 character but blocks on entry to the *get* method;
- c. *C3* attempts to get 1 character but blocks on entry to the *get* method;

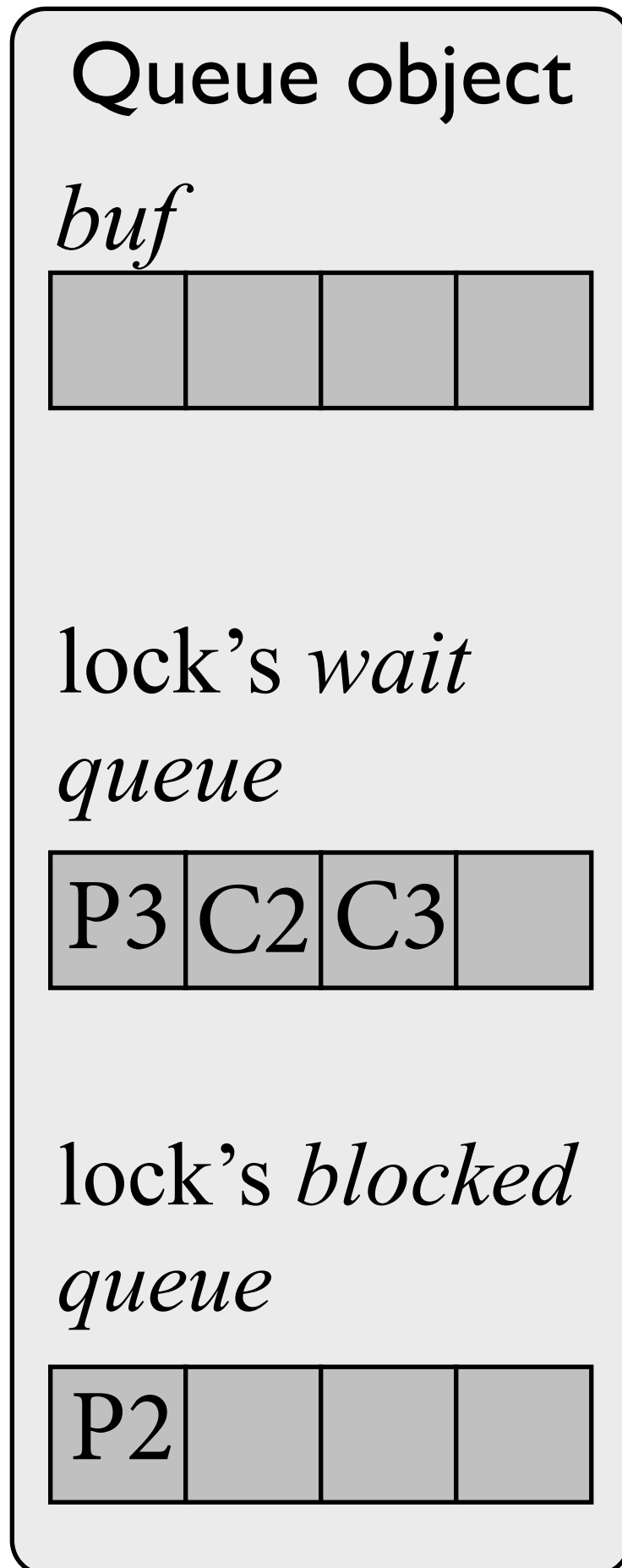
**Show the status of *buf*, lock's *wait queue* and lock's *blocked queue*.**



5. The following happen at time step 5.

- C1* is executing the *get* method, gets the character, calls *notify* and exits the method (releasing the lock and giving *C2* and *C3* a chance to acquire it);
- The *notify* wakes up *P2*
- BUT, *C2* enters the method before *P2* can (*P2* must reacquire the lock), so *P2* blocks on entry to the put method;
- C2* checks the wait loop, sees there are no more characters in the buffer and so it waits (releasing the lock in the process)
- C3* enters the method after *C2*, but before *P2*, checks the wait loop, sees there are no more characters in the buffer, and so it waits

**Show the status of *buf*, lock's *wait queue* and lock's *blocked queue*.**



6. The following happen at time step 6.

- a. Now  $P3$ ,  $C2$  and  $C3$  are all waiting!
- b.  $P2$  acquires the lock, puts a “ $d$ ” in the buffer, calls *notify* and exits the method

**Show the status of *buf*, lock’s *wait queue* and lock’s *blocked queue*.**

Queue object

*buf*

d			
---	--	--	--

lock’s *wait queue*

P3	C2	C3	
----	----	----	--

lock’s *blocked queue*

--	--	--	--

7. The following happens at time step 7.

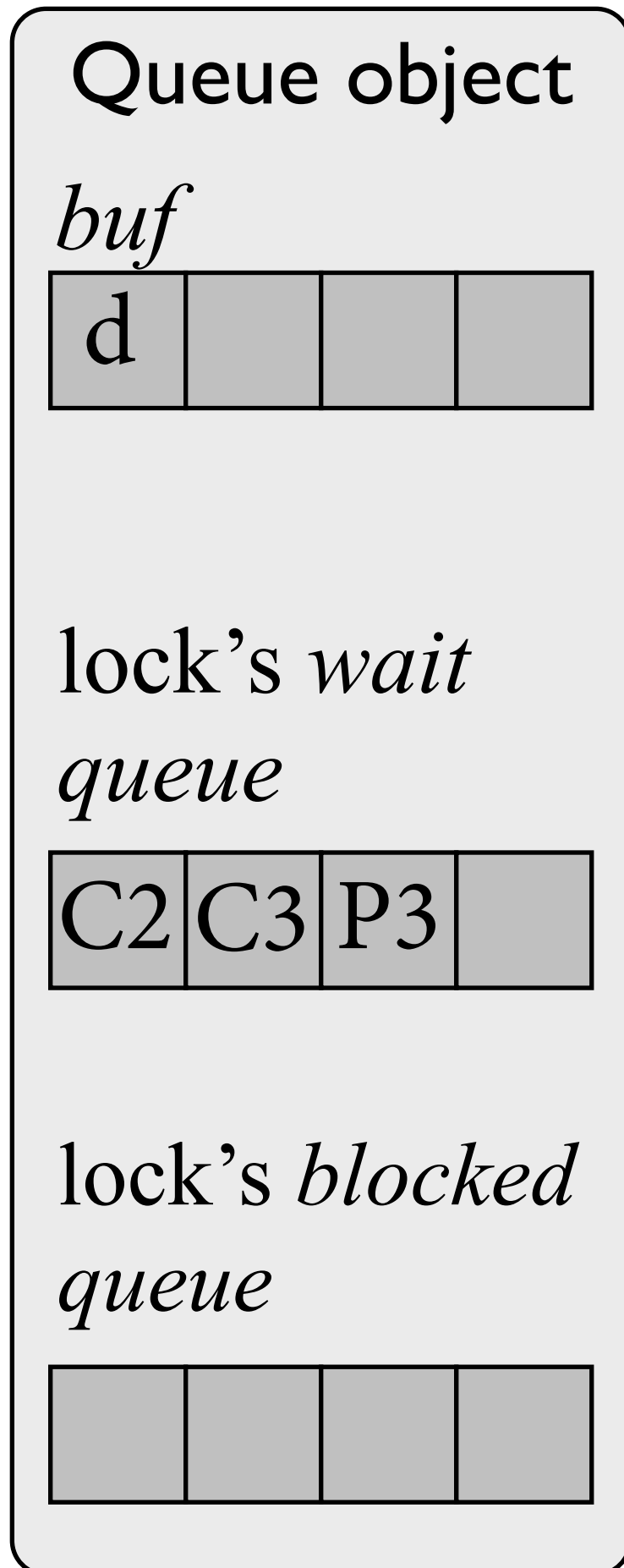
- a.  $P2$ 's notification wakes up  $P3$  (any thread can be woken up)
- b.  $P3$  checks the wait loop condition. There is already a character (" $d$ ") in the buffer and so it waits.

**Show the status of  $buf$ , lock's *wait queue* and lock's *blocked queue*.**

**Is it possible for any thread to be woken up by another notify?** No, it will only notify that specific thread

**What would have happened if in 6b a `notifyAll()` was called?**

C2 would have grabbed the character 'd', leaving C3 and P3 in the wait queue



The correct code. Always use *notifyAll* unless there is a good reason not to.

```
public synchronized void put(Object o) {
    while (buf.size()==MAX_SIZE) {
        wait(); // called if the buffer is full (try/catch removed
                // for brevity)
    }
    buf.add(o);
    notifyAll(); // called in case any getters or putters waiting
}

public synchronized Object get() {
    // Y: this is where C2 tries to acquire the lock (i.e. at the
    // beginning of the method)
    while (buf.size()==0) {
        wait(); // called if the buffer is empty (try/catch removed
                // for brevity)
        // X: this is where C1 tries to re-acquire the lock (see below)
    }
    Object o = buf.remove(0);
    notifyAll(); // called in case any getters or putters waiting
    return o;
}
```