

## Comments on Nachos Assignment 3

---

---

---

---

---

---

---

---

## EventBarrier

- `EventBarrier` has a binary "memory".
- It has a "broadcast" to notify all waiting threads of an event.
- The broadcast primitive waits until the event is handled.

```
EventBarrier::Wait()
    If the EventBarrier is not in the signaled state, wait for it.

EventBarrier::Signal()
    Signal the event, and wait for all waiters/arrivals to respond.

EventBarrier::Complete()
    Notify EventBarrier that caller's response to the event is complete.
    Block until all threads have responded to the event.
```

---

---

---

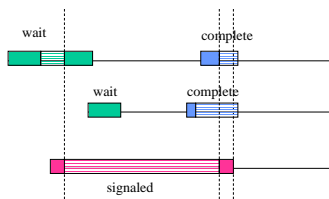
---

---

---

---

---



The question is whether completers have to wait –  
What is that 3<sup>rd</sup> line for? Fix this....

---

---

---

---

---

---

---

---

## The Moat Problem

- Travelers, knights, and troubadours arrive at the castle.
- The castle guard decides when to lower the bridge to allow the arrivals into the castle.
- If the bridge is down, new arrivals may enter immediately without waiting.
- The guard doesn't raise the bridge if there are people on it.

- This can be solved easily using *EventBarrier*.

---

---

---

---

---

---

---

---

## The Moat Problem with EventBarrier

```
EventBarrier gate;

/* Called by knights etc. */
void EnterCastle() {
    gate.Wait();      /* wait for gate to open (if necessary) */
    CrossBridge();
    gate.Complete(); /* tell the guard its OK to close gate */
}

void GuardThread() {
    while (TRUE) {
        /* twiddle thumbs */
        /* watch for arriving travelers */
        /* decide when to open gate */
        WaitForOrderToOpenGate();
        gate.Signal(); /* open gate, wait for travelers to cross, close gate */
        /* gate is closed */
    }
}
```

---

---

---

---

---

---

---

---

## EventBarrier Example

```
EventBarrier channel;

void OutputThread {
    while (TRUE) {
        ComputeDataToSend();
        channel.Wait();
        SendData();
        channel.Complete();
    }
}

void ChannelScheduler() {
    while (TRUE) {
        WaitUntilTimeToOpenChannel();
        channel.Signal(); /* open floodgate for bursts of outgoing data */
        /* channel is closed */
    }
}
```

### Invariants:

1. Output thread never blocks in Wait() if the channel is already open.
2. Channel never closes while a thread is sending data.
3. Each thread sends at most once each time the channel opens.

---

---

---

---

---

---

---

---

## Highway 110 Problem

Highway 110 is a two-lane north-south road that passes across a one-lane bridge. A car can safely enter the bridge if and only if there are no oncoming cars on the bridge.

To prevent accidents, sensors installed at each end of the tunnel notify a controller computer when cars arrive or depart in either direction. The controller uses the sensor input to control signal lights at either end of the bridge.

---

---

---

---

---

---

---

---

## Highway 110 with EventBarrier

EventBarrier north;  
EventBarrier south;

```
void HeadingNorth {
    north.Wait();
    go across one-lane bridge;
    north.Complete();
}

void HeadingSouth {
    south.Wait();
    go across one-lane bridge;
    south.Complete();
}

void BridgeScheduler {
    while (TRUE) {
        north.Signal();
        south.Signal();
        ....
    }
}
```

---

---

---

---

---

---

---

---

## Highway 110 with Locks and Condition Variables

OneVehicle(int direc) //direc is either 0 or 1; giving the direction in which the car is to cross

```
{
    ArriveBridge(direc);
    CrossBridge(direc);
    ExitBridge(direc);
}
```

Variation of this problem has "load limit" restriction of 3 cars at a time

Fairness?

```
ArriveBridge(dir)
{
    bridgeLock->Acquire();
    while (num_on_bridge != 0 &
        dir != direction)
        OKtoGo[dir]->Wait();
    direction = dir;
    num_on_bridge++;
    bridgeLock->Release();
}

ExitBridge(dir)
{
    bridgeLock->Acquire();
    num_on_bridge--;
    if (num_on_bridge == 0) {
        direction = Idir;
        OKtoGo[direction]->Broadcast();
    }
    bridgeLock->Release();
}
```

---

---

---

---

---

---

---

---