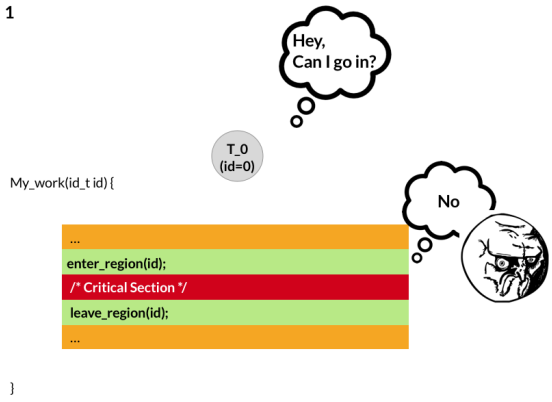
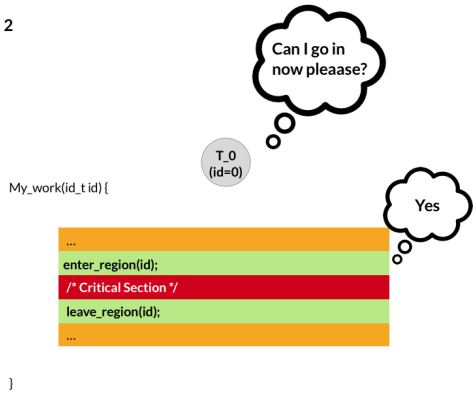
Vocabularies

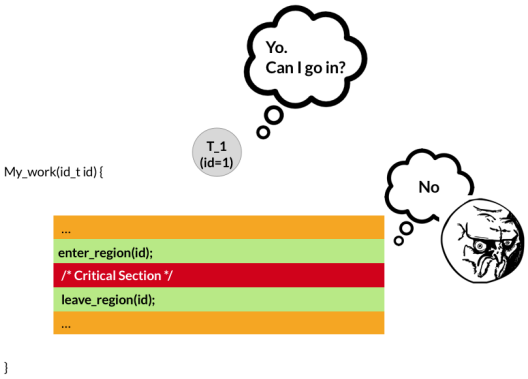
- **Peterson's Algorithm**
  - is a concurrent programming algorithm for mutual exclusion that allows two or more processes to share a single-use resource without conflict, using only shared memory for communication



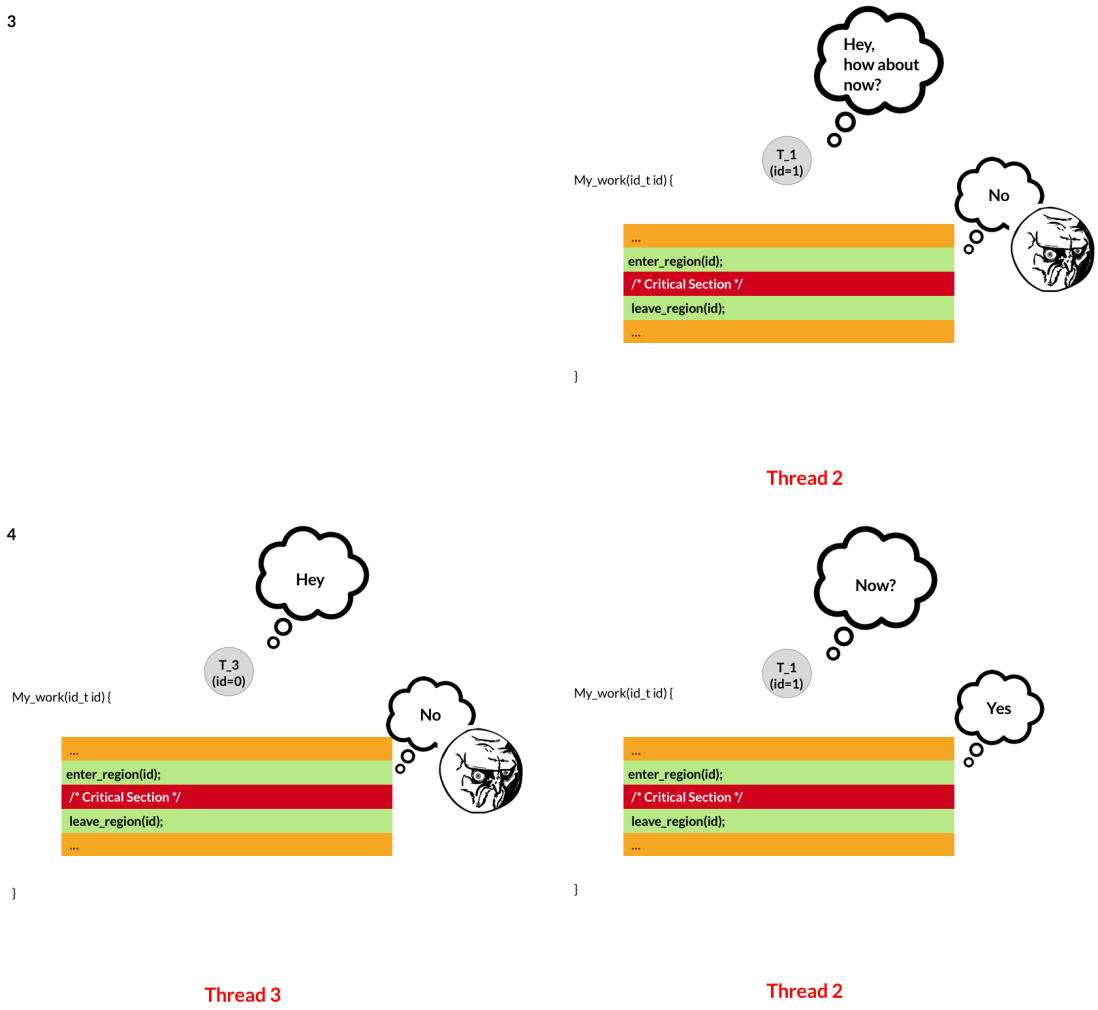
Thread 1



Thread 1



Thread 2



- **Lamport's Bakery Algorithm**
  - Is one of the simplest known solutions to the mutual exclusion/critical section problem for the general case of N process
- **Synchronization**
  - Is the concurrent execution of two or more threads that share critical resource to avoid critical resource use conflicts
- **Disable Interrupts**
  - Is a type of interrupt that postpones interrupts until a later time
- **Spin Lock**
  - Is a loop that keeps a thread from going beyond the loop till a certain condition is met

```
1     while(cantGoOn) {};
```

- **Priority Inversion**
  - Is a problem a low priority process acquiring a resource that a high priority process needs, and then being preempted by a medium priority process, so the high priority process is blocked on the resource while the medium priority one finishes

    Example

    Mars Pathfinder Rover

- **Sleep Lock**
  - Is a type of thread where locking condition is achieved by putting thread to sleep (into "blocked" state) while waiting to acquire a lock lock

```
wait_event(queue, condition)
wake_up(wait_queue_head_t *queue);
```

- **Condition variables**
  - Is an explicit queue that threads can put themselves on when some state of execution (i.e., some condition) is not as desired (by waiting on the condition); some other thread, when it changes said state, can then wake one (or more) of those waiting threads and thus allow them to continue (by signaling on the condition)
- **Semaphores**
  - Is a variable or abstract data type used to control access to a common resource by multiple processes in a concurrent system such as a multitasking operating system.


```
wait_event(queue, condition)
wake_up(wait_queue_head_t *queue);
```

- **Signal**
  - Is a function that unblock one threads currently blocked on the specified condition variable
- **Broadcast**
  - Is a function that unblock all threads currently blocked on the specified condition variable

1 Lecture Video

1.1 Synchronization Hardware

- We are not going to use peterson's algorithm or lampert's bakery algorithm to solve critical section problem
- We will be using a lot of conditional variables

1.2 Disabling Interrupts

```
boolean TAS(boolean *lock) {
  boolean old = *lock;
  *lock = True;
  return old;
}
```

```
boolean TAS(boolean *lock) {
    if(*lock == False) {
        *lock = True;
        return False;
    } else
        return True;
}
```

- Is used within the operating system
  - Is used in very short critical section
  - Disabling interrupts in OS is used sometimes
  - Disabling interrupts in OS works only on uniprocessor
- Is poor solution for user-level programs - what could go wrong here?
  - **Professor Reid:** Interrupts may never be enabled again
- Disabling interrupts not sufficient on a uniprocessor why?
  - **Professor Reid:** Because interrupts work on per process basis (so it disables interrupts on one processor and not the other)

1.3 Atomic instructions: Test and Set

```
boolean TAS(boolean *lock) {
  boolean old = *lock;
  *lock = True;
  return old;
}
```

```
boolean TAS(boolean *lock) {
    if(*lock == False) {
        *lock = True;
        return False;
    } else
        return True;
}
```

- Is hardware-based
- The code is for definition and not implementation (it may look different)
- **Professor Reid:** It modifies the value of the lock variable, and returns the value of what it was just set before

1.4 Lock Implementation

- Called **spin lock** becauses it uses busy waiting
- Consumes CPU cycles
- Is like constantly knocking the door until someone opens it

1.5 Other Implementations

- starvation could occur because everyone is trying to knock on the door.
  - We don't know who will get in
  - It's possible that a thread will be locked out from getting in critical section indefinitely
- Dead lock
  - Will come back later

1.6 Sleeps Locks

- Is an alternative to spin lock
- A thread is put inside queue
  - In linux, this is called **wait queue**
- Works by putting a thread to sleep (into a blocked state)
- A thread would wake up when another thread releases the lock
  - Sometimes on multiple threads blocking on that lock

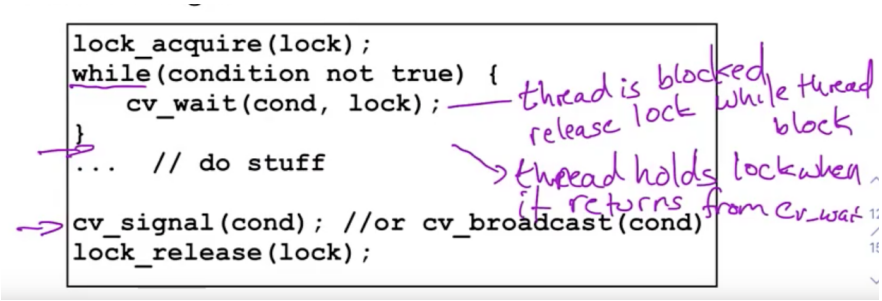
1.7 Next: Higher Level Abstractions

- Locks
  - Lock is a very simple abstraction
  - Has an ability to lock and unlock
  - Is highly useful in case where we want one thread to modify value of a varaible at a time, this is the perfect mechanism
- Conditional Variables
  - Assignment 2 is entirely based on conditional variable and locks
  - idea: Thread needs to wait until something happens
  - example
    - Thread waiting to put some piece of data (producer / consumer problem)
    - Thread waiting until some kind of action occurs
  - Key: sets up a condition that we want to test
    - condition is true: continue
    - condition is false: wait for a while until we are told OK, the condition is true and you can proceed
  - Condition can be any boolean expression
    - Condition can be true
  - Will be something that will be talked for a long time
- Semaphore
  - Conditional variable can be implemented using semaphores
  - Semaphores can be implemented using conditional variable
  - From reasoning point of view, conditional variable is more easier and more widely used than semaphores
  - Is covered little in this course

1.8 Conditional Variable

- Conditional variable is also a variable
- Is always associated with a lock
- Pattern
  - Acquire the lock
  - Check for the condition (false: call `cv_wait`, true: call `cv_signal or cv_broadcast`)

1.9 Using Conditional Variable



- Is always put together with lock

Important Thread that acquire does not proceed until the lock is free