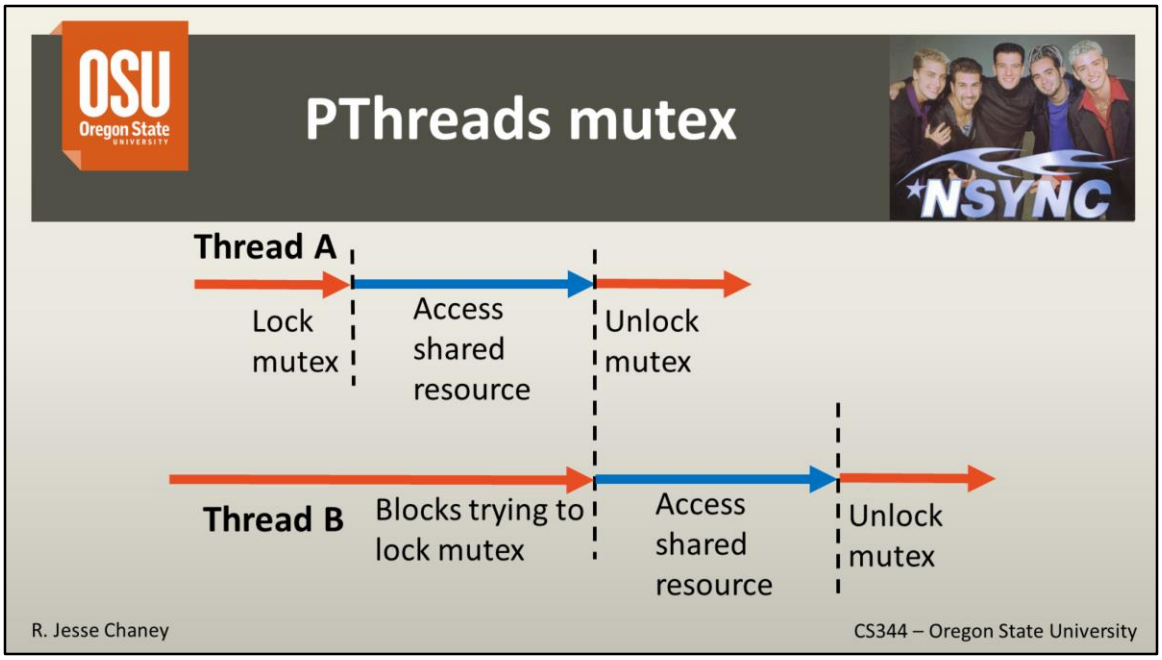
PThreads Synchronization

TLPI chapter 30 and the PThreads tutorial

Mutex
Condition variables
Spin locks
Barriers

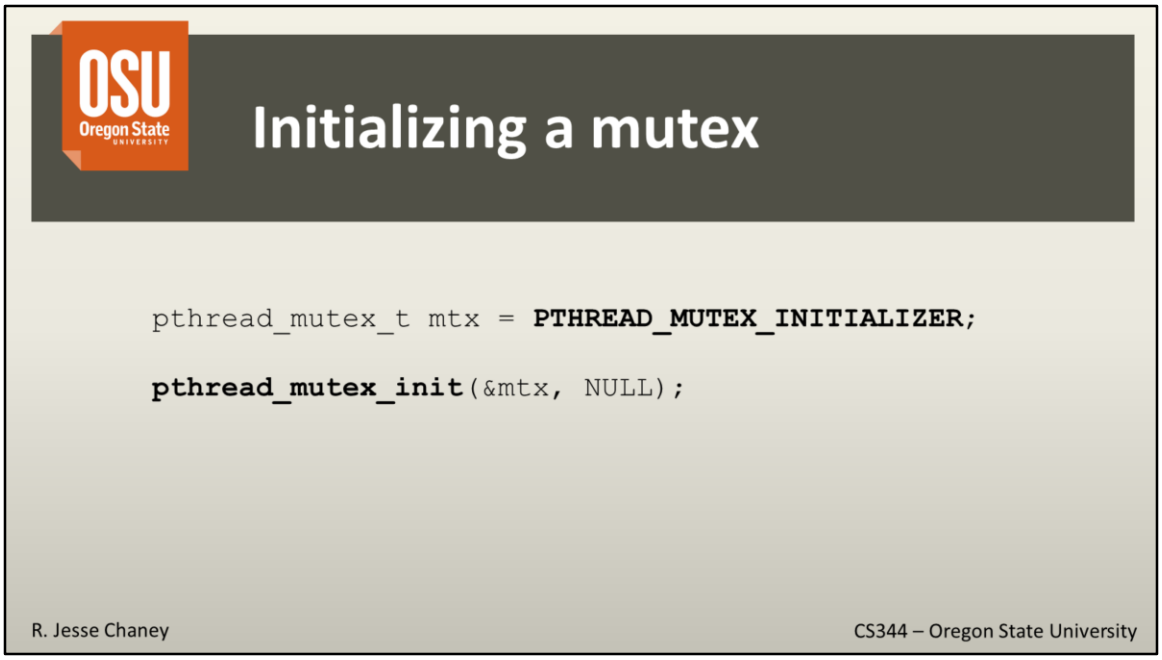
R. Jesse Chaney



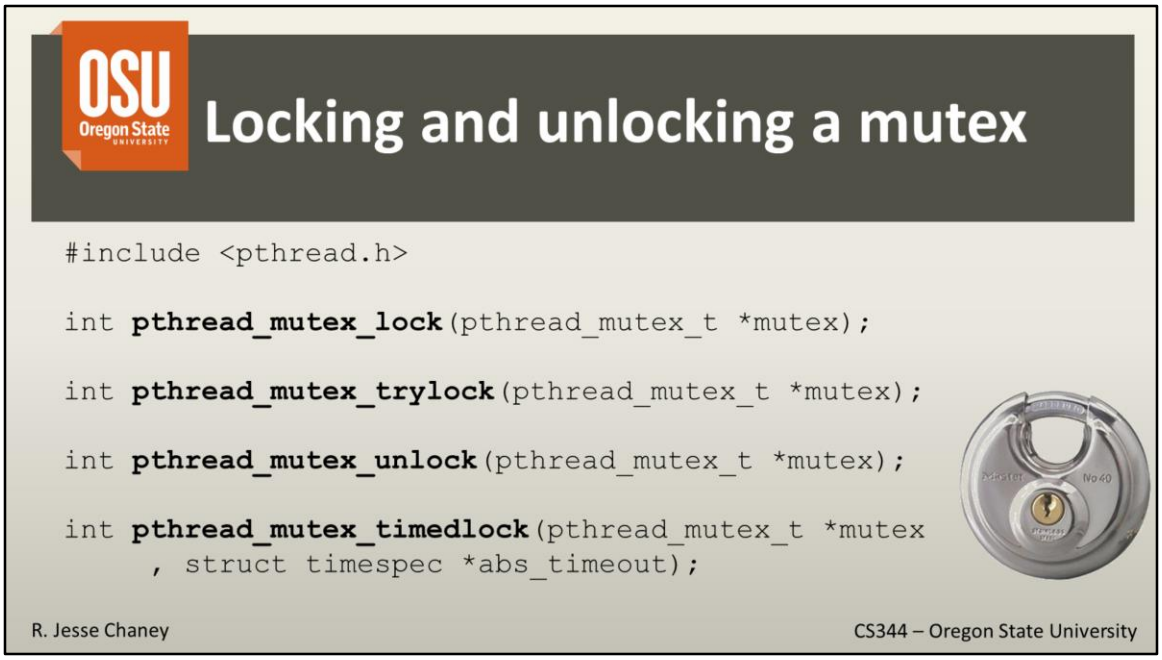
Now that we've have global memory that can be shared easily between threads, how do we protect it from producing conflicting or erroneous results. The first methods we'll review is mutex.

The word mutex stands for mutual exclusion. It allows you to protect resources (variables or sections to code) such that only one thread at a time is accessing it.

Multiple threads can be blocked waiting to lock the mutex. When the mutex is unlocked, one and only one will be able to acquirer the mutex lock.

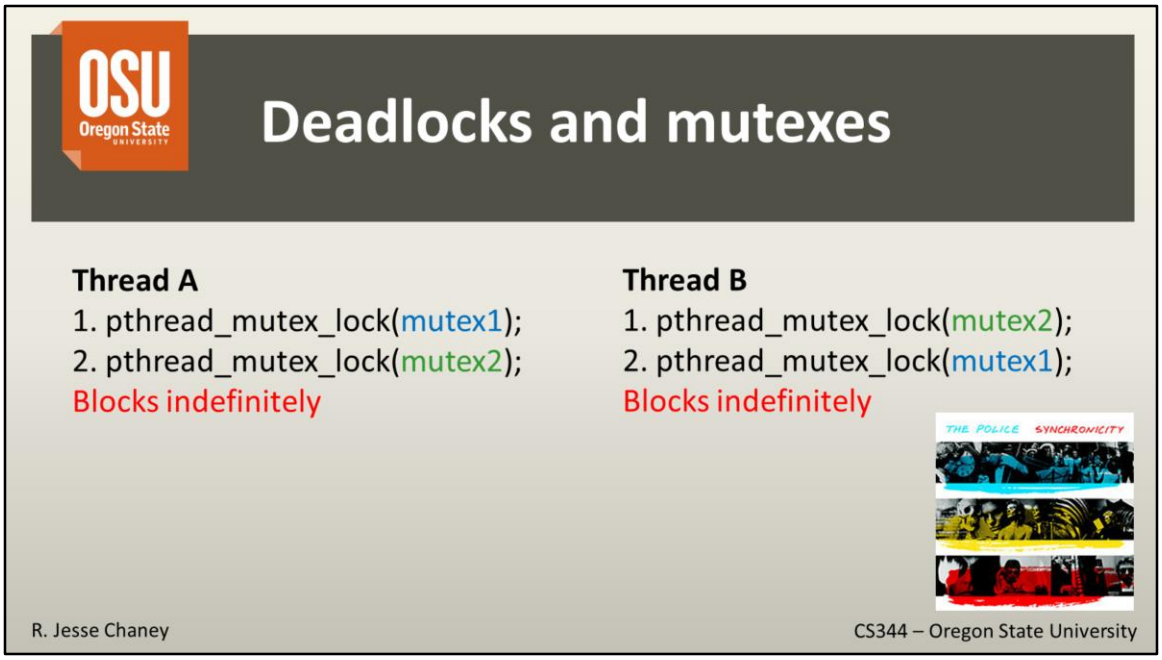


Before you make use of a mutex, you must initialize it. A statically created mutex can be initialized with the `PTHREAD_MUTEX_INITIALIZER` value. A allocated variable must have the function `pthread_mutex_init()` called on it.



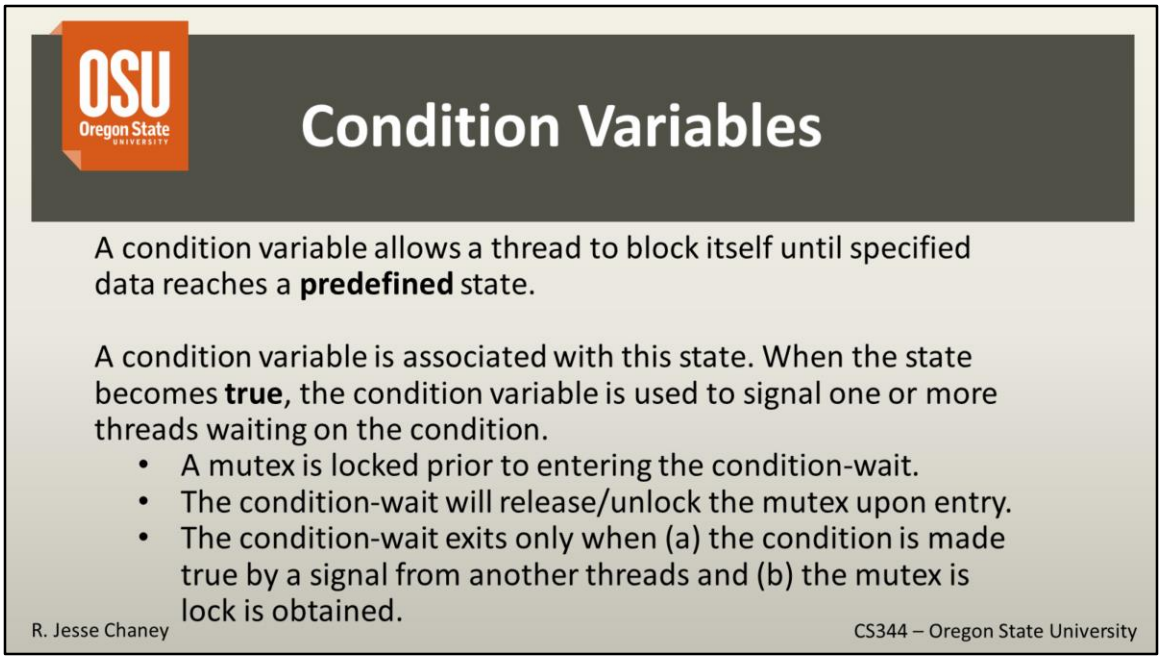
When a thread blocks trying to acquire a mutex, it state is changed to waiting. It therefor consumes very little CPU resources while waiting to acquire the mutex.

Use of the pthread_mutex_timedlock() call also requires #include <time.h>.



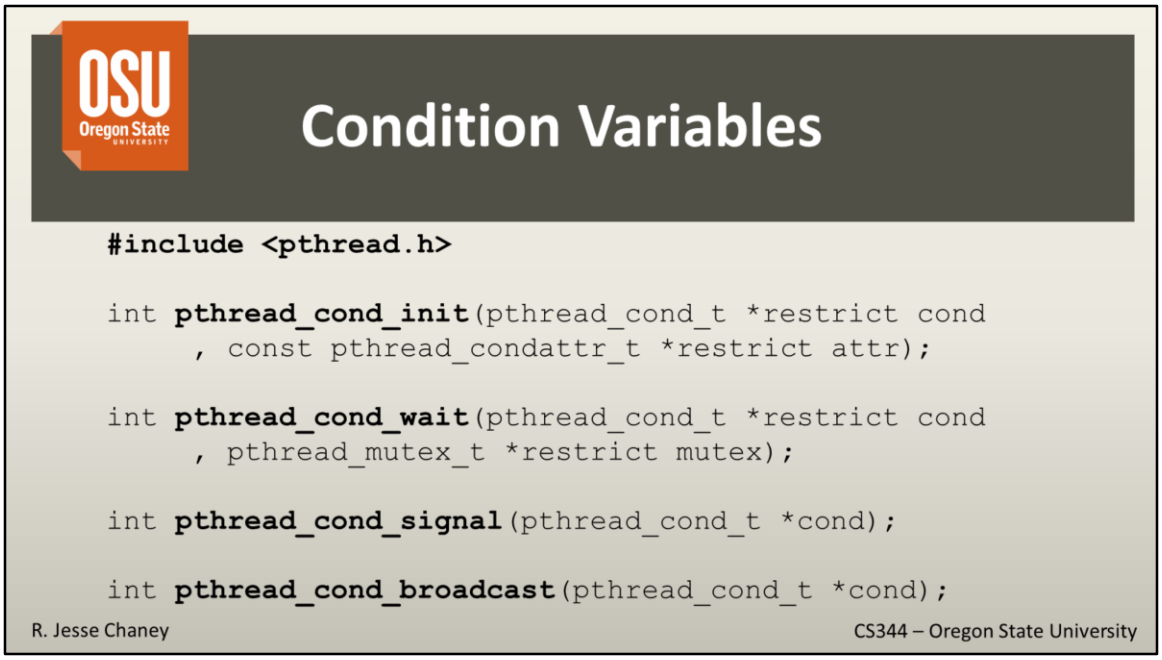
- A single thread may not lock the same mutex twice.
- A thread may not unlock a mutex that it doesn't currently own (i.e., that it did not lock).
- A thread may not unlock a mutex that is not currently locked.

The simplest way to avoid such deadlocks is to define a mutex **hierarchy**. When threads can lock the same set of mutexes, they should always lock them in the same order.



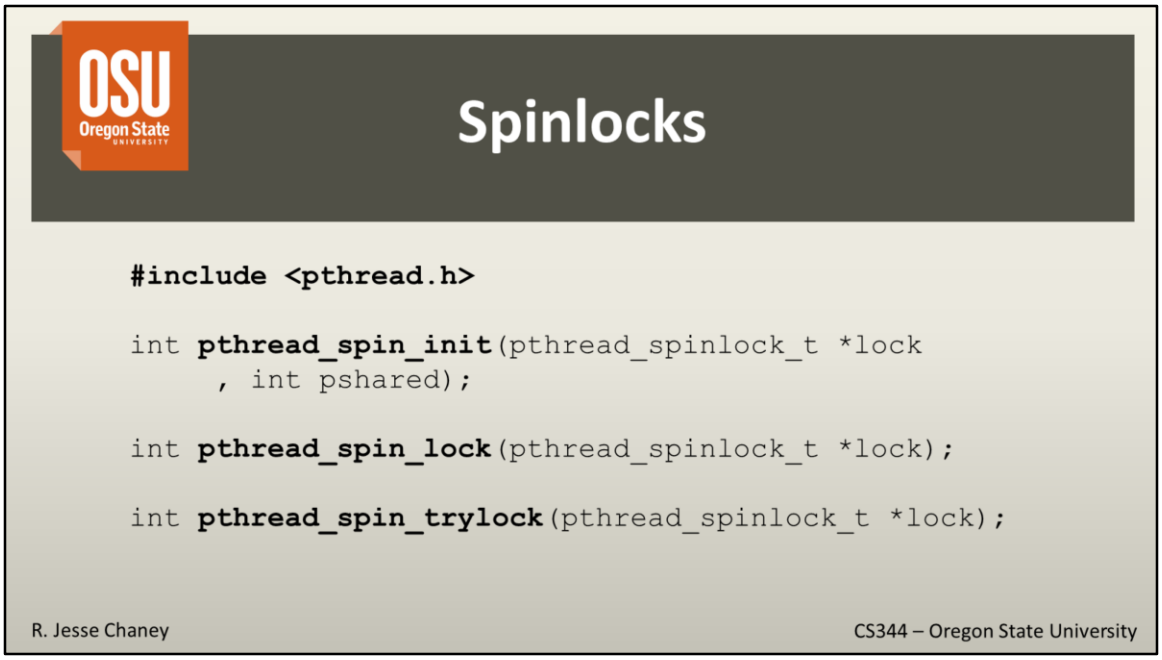
A mutex prevents multiple threads from accessing a shared variable at the same time. A condition variable allows one thread to inform (signal) other threads about changes in the state of a shared variable (or other shared resource) and allows the other threads to wait (block) for such notification.

A mutex is always associated with a condition variable.



Like as mutex must be initialized before it is used, so must a condition variable. Multiple threads can be blocked on the condition variable.

A call to the pthread_cond_signal() function will unblock a single waiting thread. A call to the pthread_cond_broadcast() will signal all threads blocked on the condition variable.



Spin locks consume a larger amount of processor resources to block the thread. When a mutex lock is not available, the thread changes its scheduling state and adds itself to the queue of waiting threads. When the lock becomes available, these steps must be reversed before the thread obtains the lock. While the thread is blocked, it consumes no processor resources. On the other hand, when a thread is waiting on a spin lock, its state remains ready and it will consume CPU resources.

Therefore, spin locks and mutexes can be useful for different purposes. Spin locks might have lower overall overhead for very short-term blocking, and mutexes might have lower overall overhead when a thread will be blocked for longer periods of time.