Mutex

* lock()/unlock() methods
* We don’t keep track of who is holding the lock--we just use the Honor System
  + Any crazy thread can call unlock() and wreak havoc
* We can deadlock with ourselves if we call lock() while already holding the lock
* Test-and-set works nicely here

Reentrant Mutex

* lock()/unlock() methods
* We keep track of which thread is the current lock holder
* If the current lock holder asks to lock again, we just increase the depth of the lock (no deadlocking here!) [like synchronized does in Java]
* Calls to unlock() verify that the calling thread is indeed the current lock holder.
  + If so, depth is decremented.
  + If depth is now 0, lock is released for other threads to get
* Compare-and-swap works nicely here

Spin Lock

* Thread tries to stay on the core/CPU in the "running" state and checks again and again and again very quickly for the lock to become available (no context switch into a blocked state).
* Potentially, the thread spinning waiting for the lock *might* get preempted and have a context switch back to "ready to run". But this should be rare because the spin should be for a very short time.
* Should only be used when the time waiting to get the lock is very short.
* Thread does NOT move to the "blocked" state of "needs lock".

Priority Inversion

* When a low-priority thread is holding a lock that a higher priority thread needs
* The higher priority thread is blocked and effectively has its priority lowered to the low priority thread’s level.
* Mitigation:
  + Temporarily raise the priority of the low priority thread up to the match high priority thread’s priority until it releases the lock.