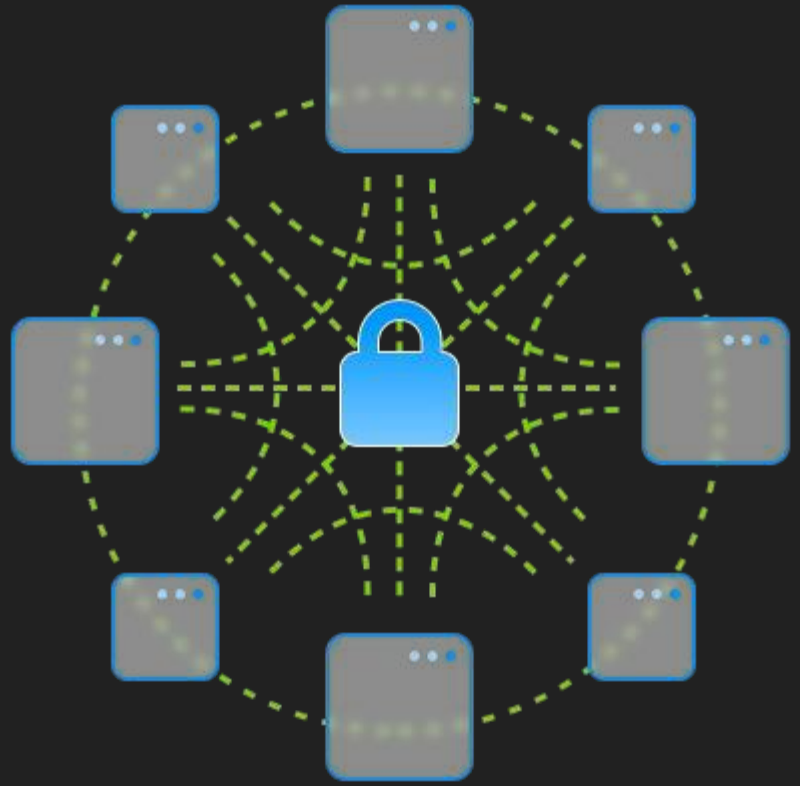
Synchronization



XV-6 Locking Mechanisms

Spin Locks

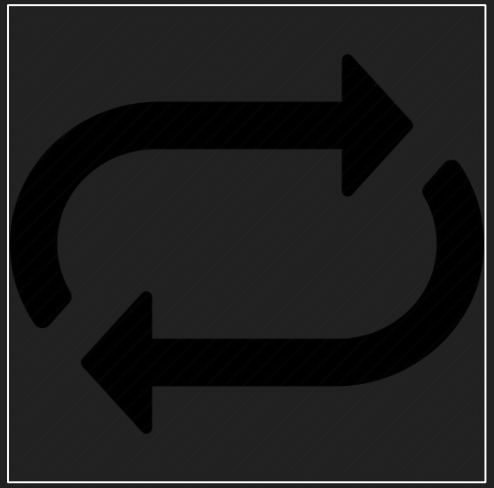
```
struct spinlock {
  uint locked;       // Is the lock held?

  // For debugging:
  char *name;        // Name of lock.
  struct cpu *cpu;   // The cpu holding the lock.
  uint pcs[10];      // The call stack (an array of program counters)
                     // that locked the lock.
};
```

- The major disadvantage: BUSY WAITING!

Reentrant Lock

Implementation

A lock which can be obtained several times by its current holder.



You can modify spinlock in order to satisfy this requirement.

Then you should write a function to test this modified lock.

Reentrant Lock

- First of all, you should know how spin locks work in xv6. (read functions in spinlock.c)
- Then you can change it to be able to acquire lock several times.
- Then you should write a recursive function that acquires lock.

Readers-Writer Locks

- No reader should be kept waiting unless a writer has already obtained permission to use the shared object.
- No BUSY WAITING is allowed!

System Calls

- **rwinit()**

  Initialize a lock.

- **rwtest(uint** pattern**)**

  This is for testing the reader-writer lock. The order of read or write permissions are specified in pattern argument (A binary string which always starts with 1. 0 for readers and 1 for writers.)