Spin Locks and Contention

recap

Using a lock

```
Lock mutex = new LockImpl(...); // lock implementation
...
mutex.lock();
try {
  ...             // body
  } finally {
    mutex.unlock();
  }
```

Peterson – lock for 2 using only read/write

```
class Peterson implements Lock {
  private boolean[] flag = new boolean[2];
  private int victim;
  public void lock() {
    int i = ThreadID.get(); // either 0 or 1
    int j = 1-i;
    flag[i] = true;
    victim = i;
    while (flag[j] && victim == i) {}; // spin
  }
}
```

Program order and seq. consistent memory are necessary – will not work in practice!

TASLock – spin getAndSet

```
public void lock() {
  while (state.getAndSet(true)) {}
}
public void unlock() {
  state.set(false);
}
```

TTASLock – spin read

```
while (true) {
  while (state.get()) {};
  if (!state.getAndSet(true))
    return;
}
```

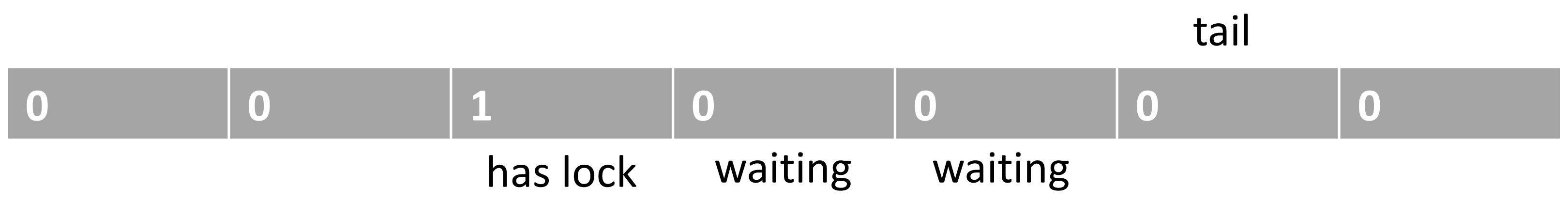
Backoff – wait instead

```
while (true) {
  while (state.get()) {};
  if (!state.getAndSet(true)) {
    return;
  } else {
    backoff.backoff();
  }
}
```

Queue Locks

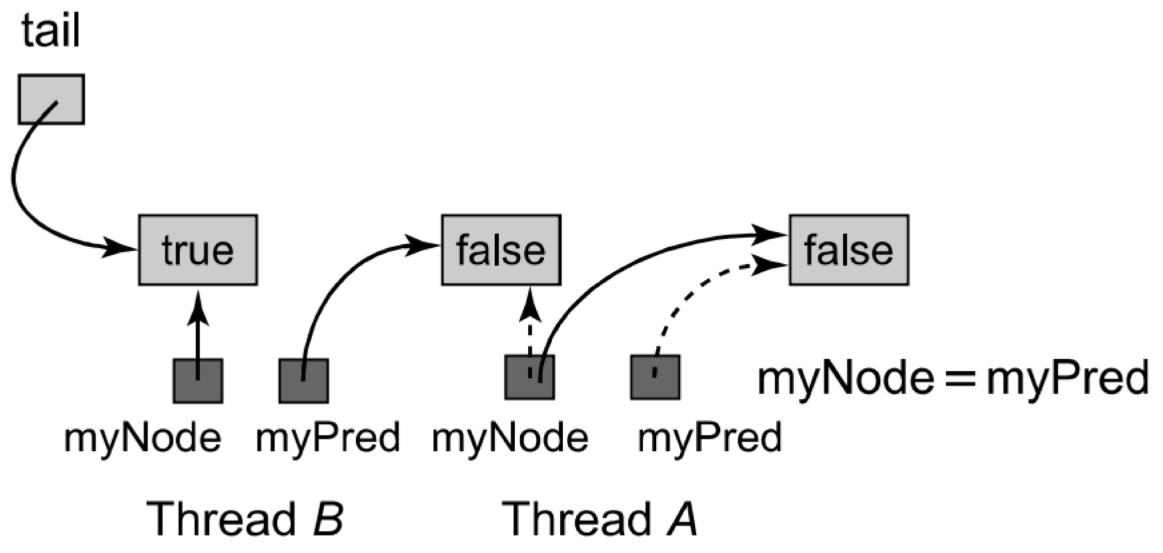
Array Lock – store in an array who has the lock

```
public void lock() {
  int slot = tail.getAndIncrement() % size;
  mySlotIndex.set(slot);
  while (! flag[slot]) {};
}
public void unlock() {
  int slot = mySlotIndex.get();
  flag[slot] = false;
  flag[(slot + 1) % size] = true;
}
```



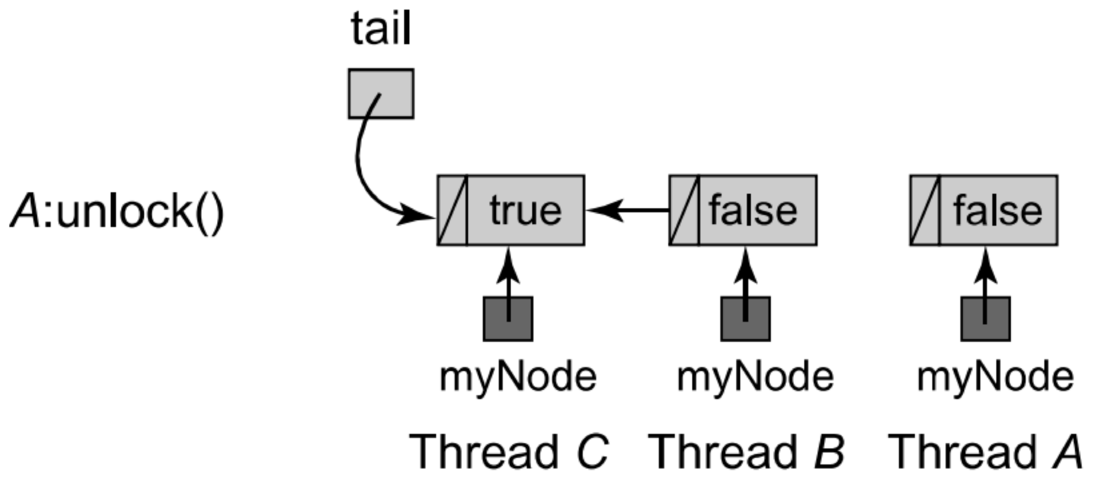
CLH Queue Lock – less space, any number of threads

A:unlock()
B:lock()



MCS Queue Lock – spins locally

but might need to spin to release the lock


tail
A:unlock()
true
false
false
myNode
myNode
myNode
Thread C
Thread B
Thread A