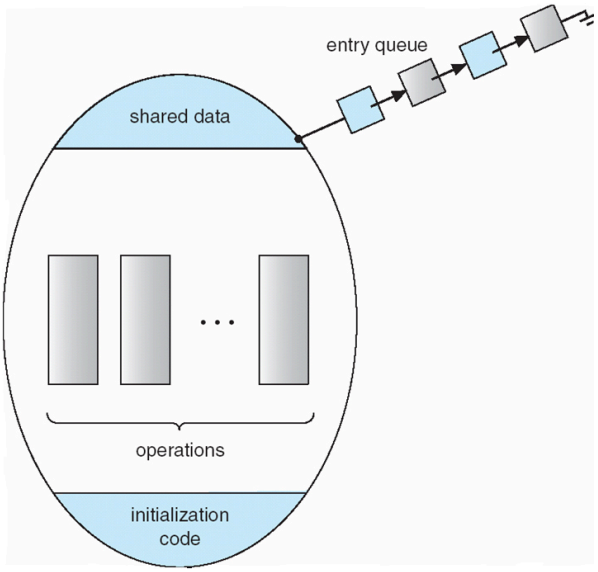
Monitors



Thread-Safe Queue with Monitors

```
class Queue {

  private ...; // queue data

  public synchronized void add(Object item) {
      put item on queue;
  }

  public synchronized Object remove() {
      remove item;
      return item;
  }

}
```

Producers/Consumers with Monitors

```
class Queue {

  private ...; // queue data


  public synchronized void add(Object item) {
    put item on queue;
    this.notify(); // wake up waiting thread, aka Signal
  }


  public synchronized Object remove() {
    while queue is empty {
      this.wait(); // give up lock and sleep
    }
    remove and return item;
  }


}
```


C++ Style Monitors

```
class Queue {

  public:
    add();
    remove();

  private:
    Lock lock;
    ConditionalVariable cv;
    Queue data;

}
```

```
Queue::add(item) {
  lock->acquire();

  put item on queue;
  cv->signal();

  lock->release();
}
```

```
Queue::remove()
  lock->acquire();

  while queue is empty {
    cv->wait(lock); // release lock & sleep
  }
  remove item from queue;

  lock->release();
  return item;
}
```

Semaphore using a Monitor

```
Semaphore(int initialVal) {

    int counter = initialVal;
    Lock lock;
    ConditionVariable notZero;

}
```

```
Semaphore::Wait() {
    lock.acquire();

    while (counter == 0) {
        notZero.wait();

    }
    counter--;

    lock.release();
}
```

```
Semaphore::Signal() {
    lock.acquire();

    counter++;
    notZero.signal();

    lock.release();
}
```