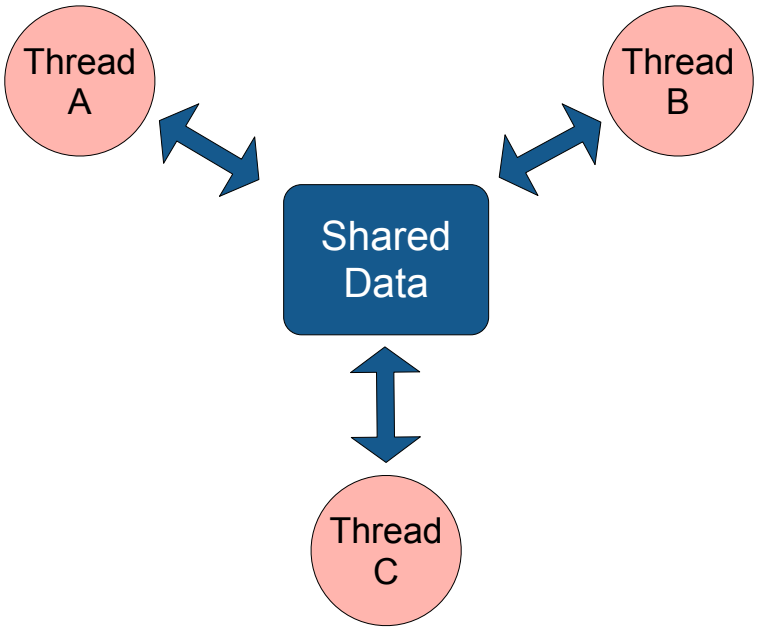
Synchronization



ATM Program

```
// get funds in account
int total = get_account_total();

// get amount to withdraw
int withdraw = get_withdraw_amount();

// check for sufficient funds
if (total >= withdraw) {

   // withdraw funds from account
   update_account_total(total - withdraw);
   dispense_money(withdraw);

}
```

Too Much Milk

```
if (noMilk) {
  buy milk;
}
```

```
if (noMilk) {
  buy milk;
}
```

Time	You	Your Roommate
3:00	Arrive home	
3:05	Look in fridge, no milk	
3:10	Leave for grocery store	
3:15		Arrive home
3:20	Arrive at grocery store	Look in fridge, no milk
3:25	Buy milk	Leave for grocery store
3:35	Arrive home, put milk in fridge	
3:45		Buy milk
3:50		Arrive home with milk
3:50		Too much milk!

Too Much Milk: Solution 1?

Thread A

```
if (noMilk & NoNote) {
    leave note;
    buy milk;
    remove note;
}
```

Thread B

```
if (noMilk & NoNote) {
    leave note;
    buy milk;
    remove note;
}
```

Too Much Milk: Solution 2?

Thread A

```
leave note A;
if (noNote B) {
    if (noMilk) {
        buy milk;
    }
}
remove note A;
```

Thread B

```
leave note B;
if (noNote A) {
    if (noMilk) {
        buy milk;
    }
}
remove note B;
```


Too Much Milk: Solution 3?

Thread A

```
leave note A;
while (note B) {
    do nothing;
}
if (noMilk) {
    buy milk;
}
remove note A;
```

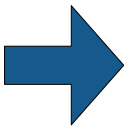
Thread B

```
leave note B;
if (noNote A) {
    if (noMilk) {
        buy milk;
    }
}
remove note B;
```

Critical Sections

```
...
arriveHome();
```

Critical Section



```
if (noMilk) {
  buy milk;
}
```

```
makeCoffee();
...
```

Property of **mutual exclusion**

Mutex Locks

Too Much Milk with Locks

Thread A

```
1   lock.acquire();
2   if (noMilk) {
3       buy milk;
4   }
5   lock.release();
```

Thread B

```
1    lock.acquire();
2    if (noMilk) {
3        buy milk;
4    }
5    lock.release();
```


Implementing Locks: Interrupts (version 1)

```
class Lock {
  public:
    void acquire();
    void release();
}
```

```
Lock::acquire() {
  disable interrupts;
}
```

```
Lock::release() {
  enable interrupts;
}
```

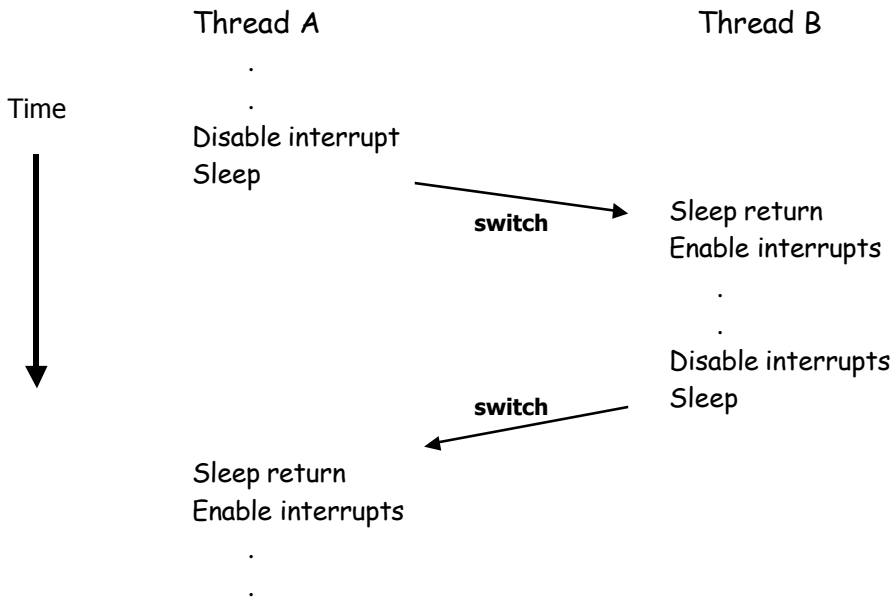
Implementing Locks: Interrupts (version 2)

```
class Lock {
  public:
    void acquire();
    void release();
  private:
    int value = FREE;
    Queue Q = empty;
}
```

```
Lock::acquire() {
  disable interrupts;
  if (value == FREE) {
     value = BUSY;
  } else {
     add curThread to Q;
     put curThread to sleep;
  }
  enable interrupts;
}
```

```
Lock::release() {
  disable interrupts;
  if queue not empty {
     take thread T off Q;
     put T on ready queue;
  } else {
     value = FREE;
  }
  enable interrupts;
}
```


Interrupt Disable/Enable Pattern



Implementing Locks: Atomic Test&Set

```
class Lock {
  public:
    void acquire();
    void release();
  private:
    int value = FREE; // FREE = 0
                      // BUSY = 1
}
```

```
Lock::acquire() {
  while (test&set(value) == BUSY) {
    // do nothing
  }
}
```

```
Lock::release() {
  value = FREE;
}
```


Minimizing Busy-Waiting

```
class Lock {
  public:
    void acquire();
    void release();
  private:
    int value = FREE;
    int guard = 0;
    Queue Q = empty;
}
```

```
Lock::acquire() {
  while (test&set(guard) == 1) {
    // do nothing
  }
  if (value == FREE) {
    value = BUSY;
    guard = 0;
  } else {
    put curThread on Q;
    guard = 0 & put curThread to sleep;
  }
}
```

```
Lock::release() {
  while (test&set(guard) == 1) {
    // do nothing
  }
  if Q is not empty {
    take T off Q;
    put T on ready queue;
  } else {
    value = FREE;
  }
  guard = 0;
}
```