Moore’s Law & Amdahl's Law

Race Conditions

Protecting Code

Deadlock

Building a Thread Pool

Threading a Shared Scan

Moore’s Law & Amdahl's Law

Race Conditions

Protecting Code

Deadlock

Building a Thread Pool

Threading a Shared Scan

Moore’s Law & Amdahl's Law
Race Conditions
Protecting Code
Deadlock
Building a Thread Pool
Threading a Shared Scan

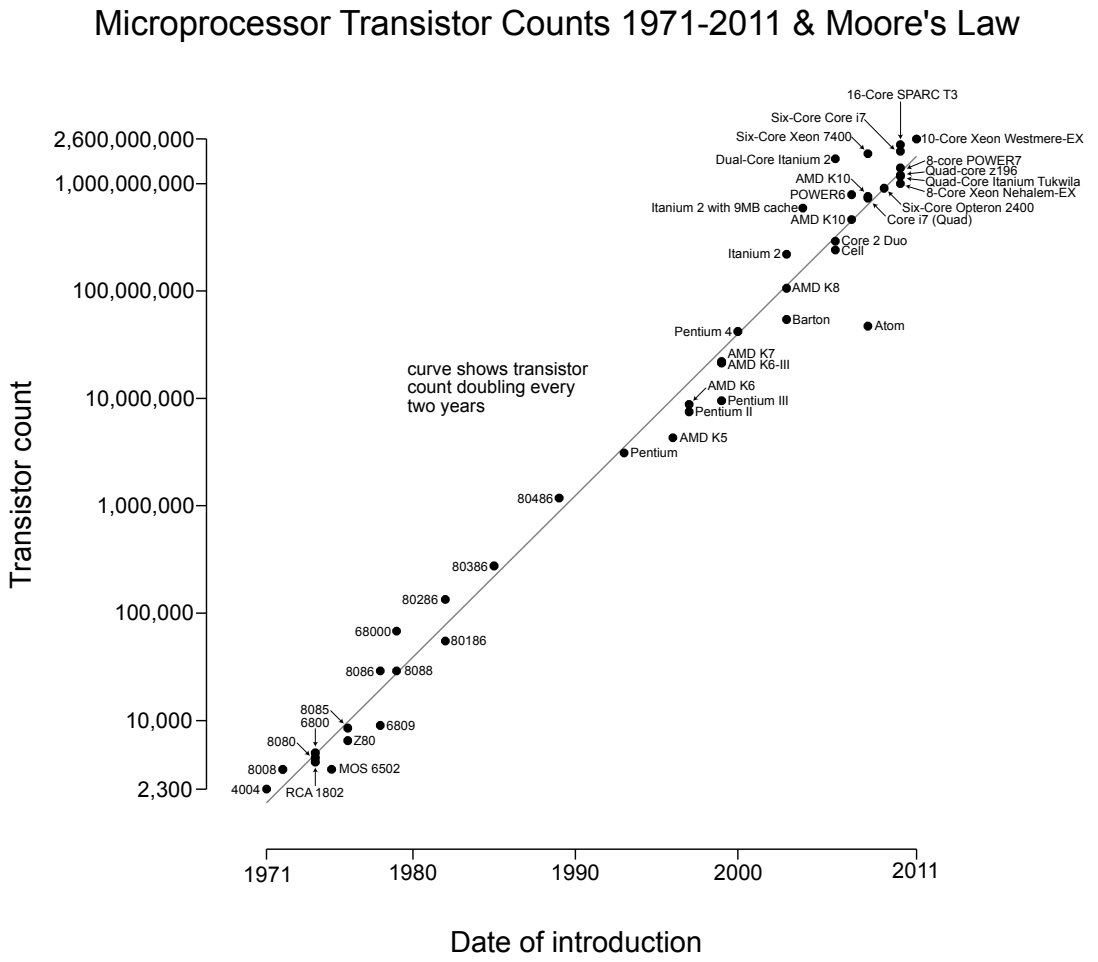
Moore’s Law & Amdahl's Law
Race Conditions
Protecting Code
Deadlock
Building a Thread Pool
Threading a Shared Scan

Moore’s Law & Amdahl's Law
Race Conditions
Protecting Code
Deadlock
Building a Thread Pool
Threading a Shared Scan

Moore’s Law & Amdahl's Law
Race Conditions
Protecting Code
Deadlock
Building a Thread Pool
Threading a Shared Scan

Moore's Law

The number of transistors in a dense integrated circuit doubles approximately every two years.



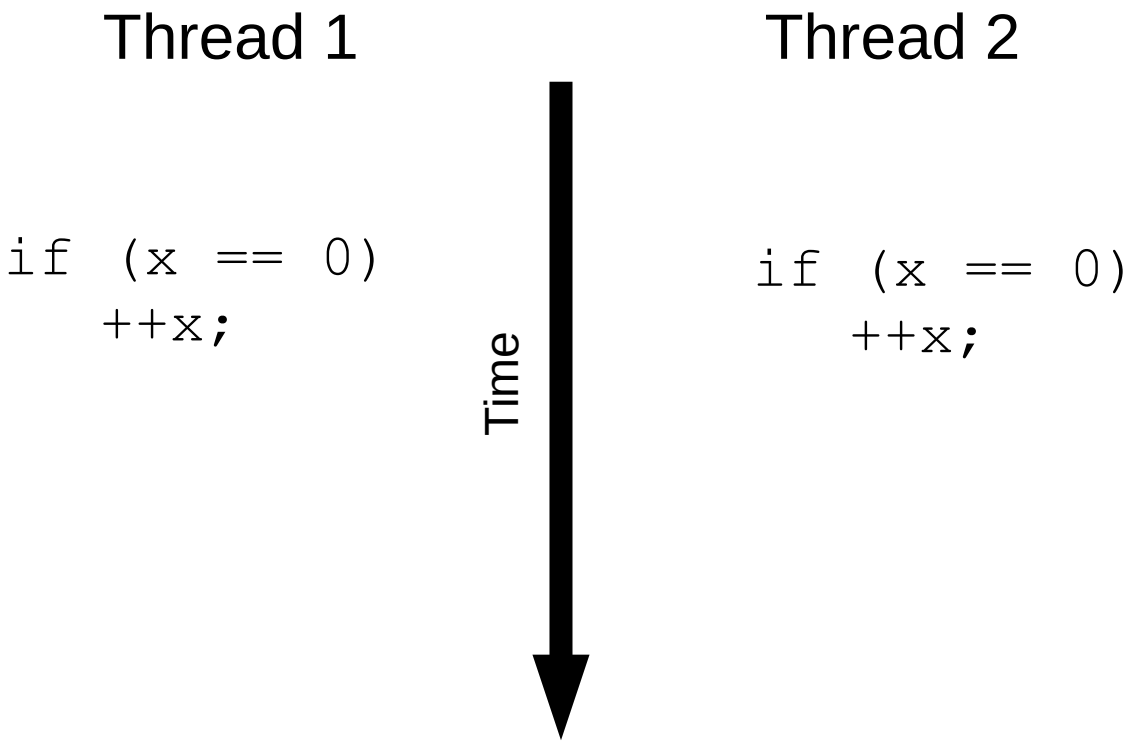
Amdahl's Law

The formula used to calculate the theoretical maximum speed-up for a given workload.

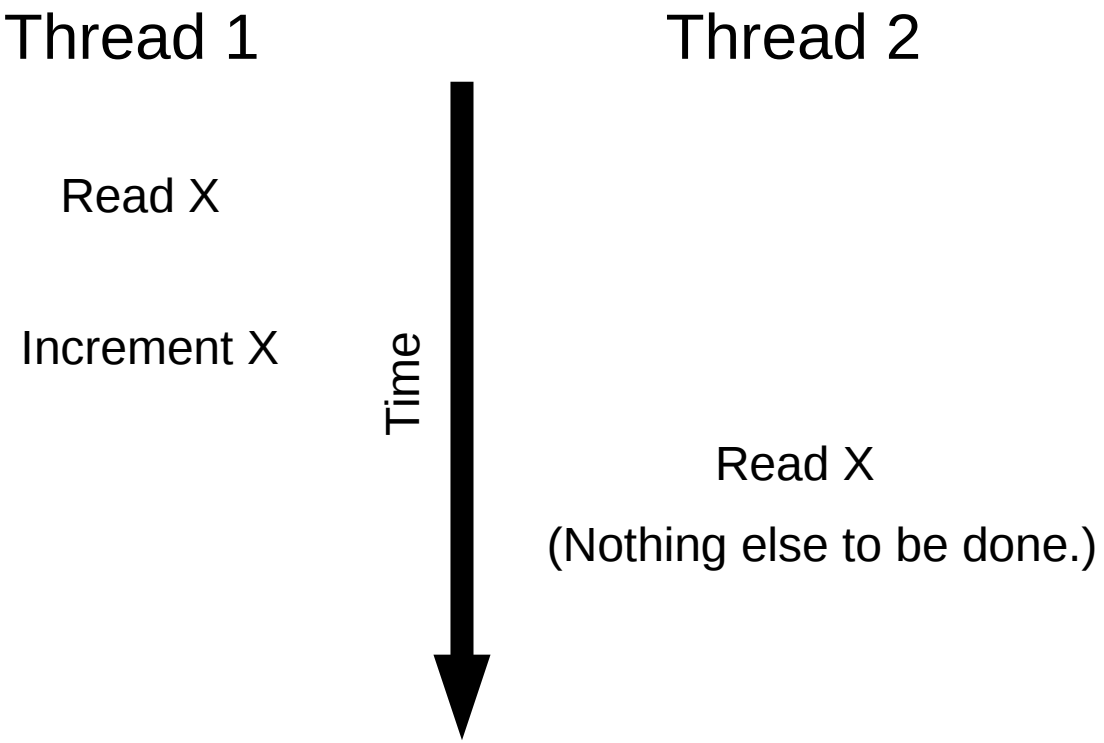
$$S_{latency}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

$$S_{latency}(4) = \frac{1}{(1-0.75) + \frac{0.75}{4}} \approx 2.28$$

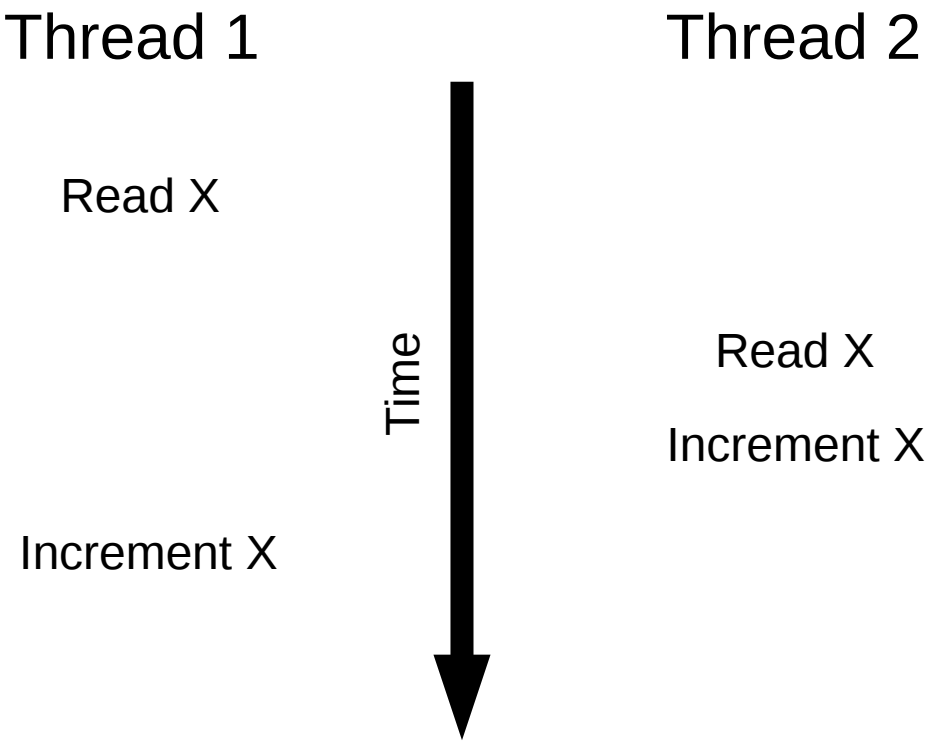
Race Conditions



Race Conditions



Race Conditions



Mutex → mutual exclusion

Atomic Operation

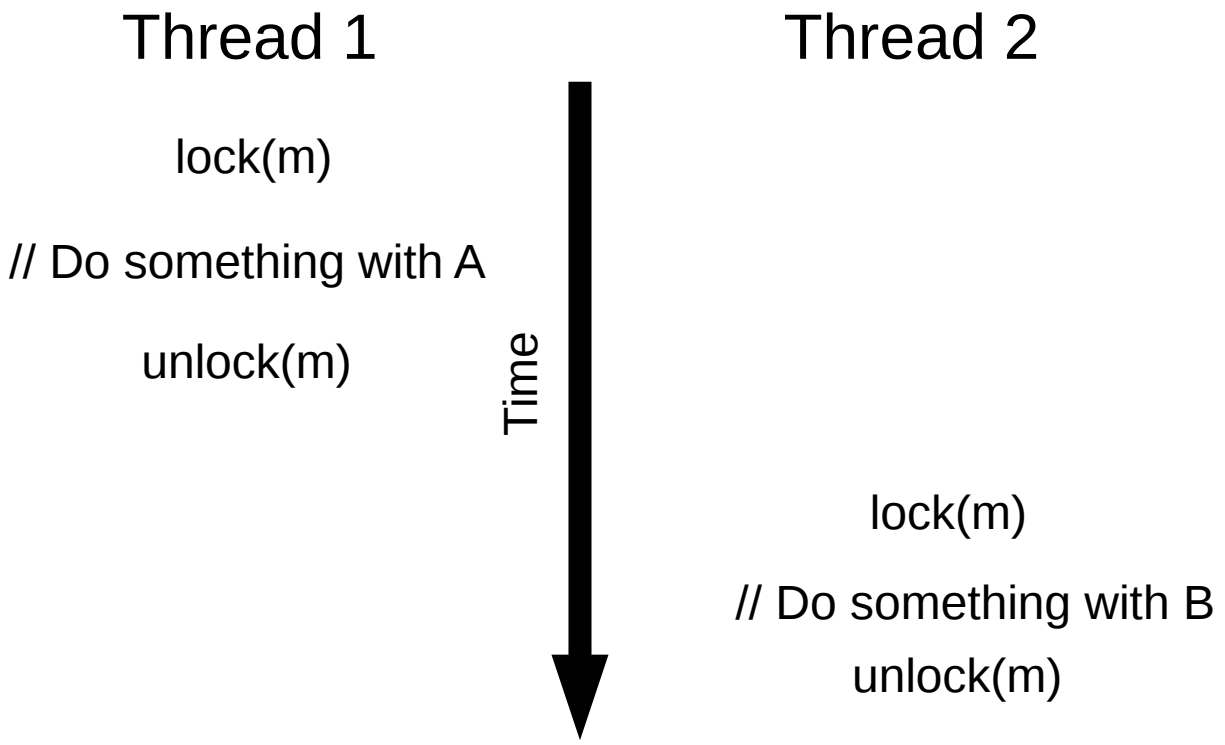
Compare-and-swap (x86 CMPXCHG)

A `pthreads mutex` Snippet

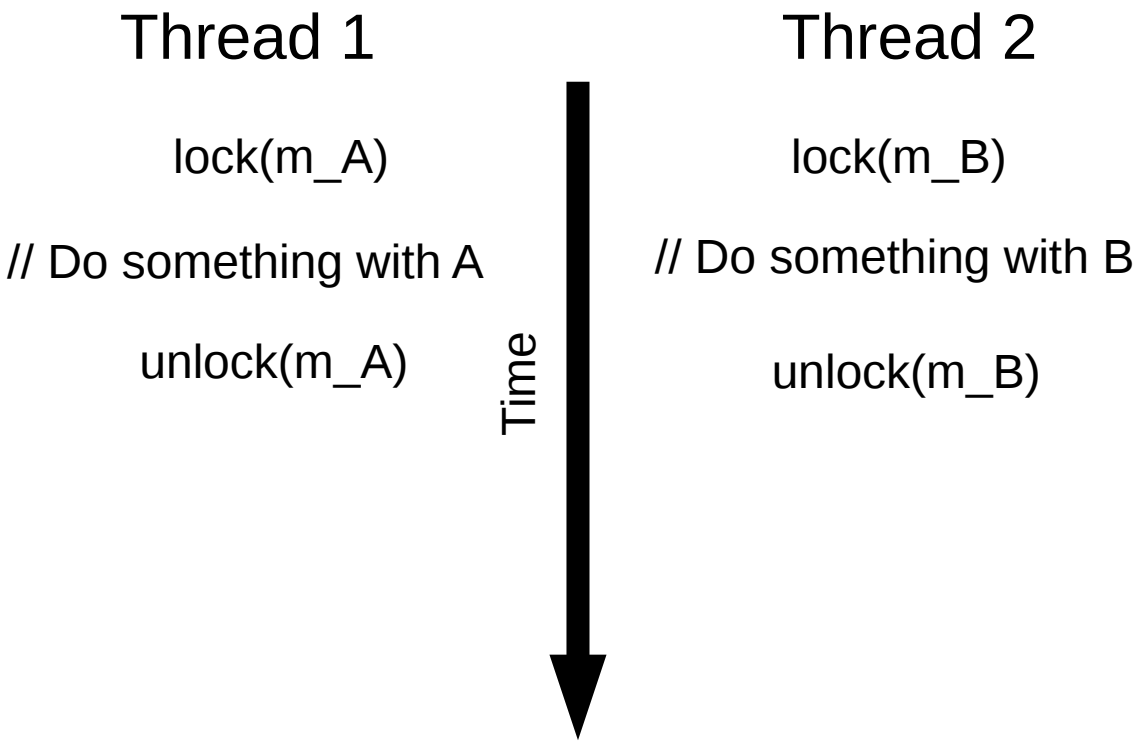
A pthreads mutex Snippet

```
#include <pthread.h>
...
pthread_mutex_t m;

pthread_mutex_init(&m, NULL);
...

pthread_mutex_lock(&m);
printf("We got the lock!");
... // Other critical code
pthread_mutex_unlock(&m);

if (pthread_mutex_trylock(&m)){
    printf("We got the lock!");
    pthread_mutex_unlock(&m);
} else {
    printf("Already locked...");
}
```

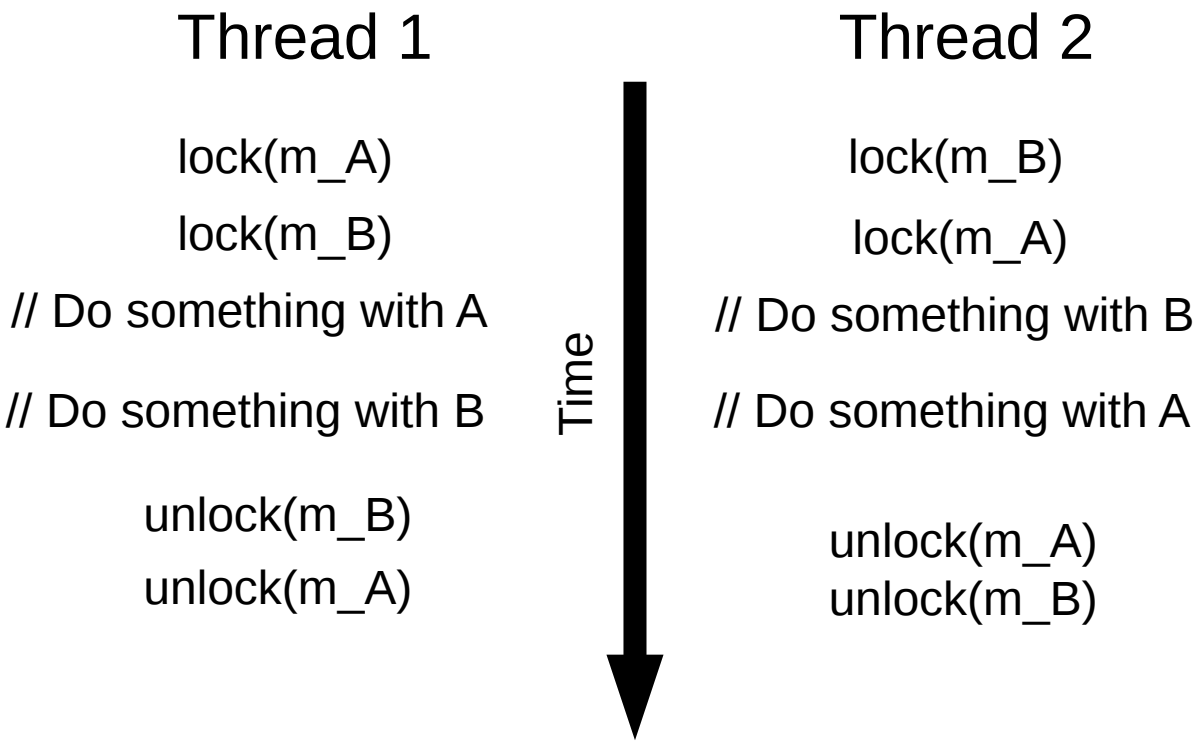
Deadlock


Thread 1
Thread 2
lock(m)
// Do something with A
unlock(m)
Time
lock(m)
// Do something with B
unlock(m)

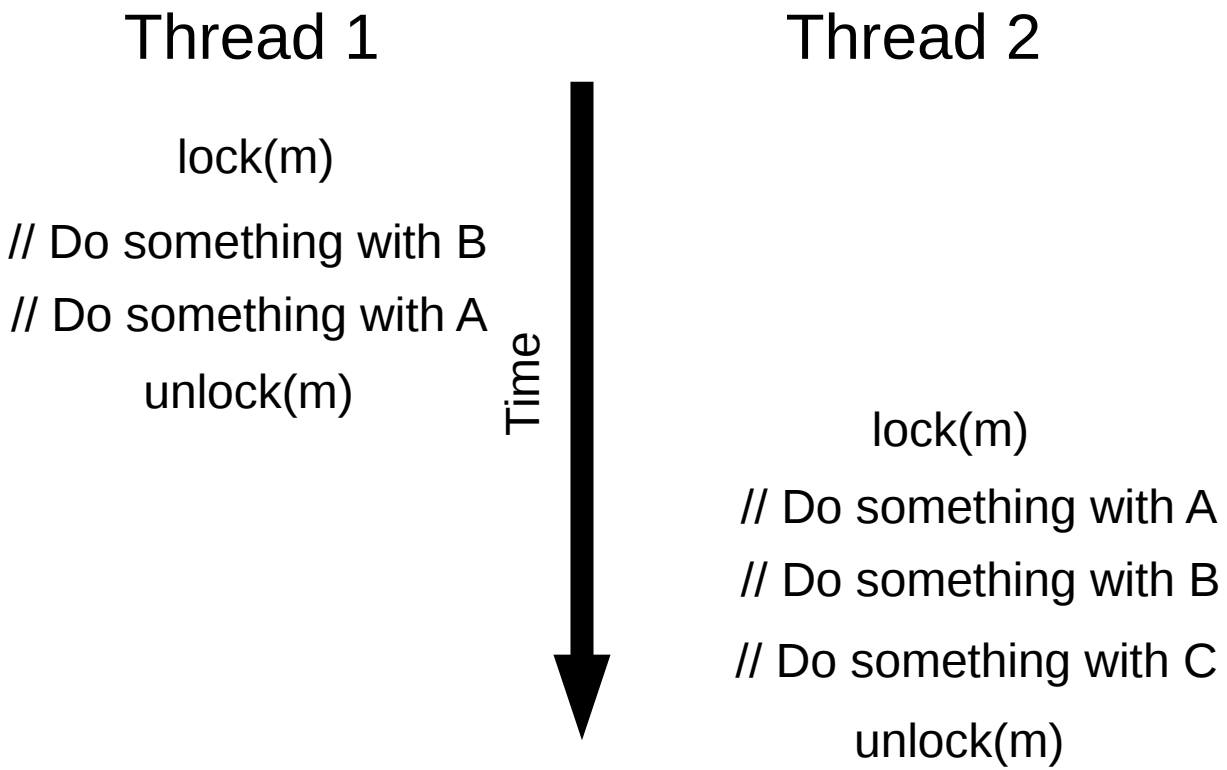
Deadlock


Thread 1
lock(m_A)
// Do something with A
unlock(m_A)
Time
Thread 2
lock(m_B)
// Do something with B
unlock(m_B)

Deadlock


Thread 1
lock(m_A)
lock(m_B)
// Do something with A
// Do something with B
unlock(m_B)
unlock(m_A)
Time
Thread 2
lock(m_B)
lock(m_A)
// Do something with B
// Do something with A
unlock(m_A)
unlock(m_B)

Deadlock


Thread 1
lock(m)
// Do something with B
// Do something with A
unlock(m)
Time
Thread 2
lock(m)
// Do something with A
// Do something with B
// Do something with C
unlock(m)

Implementing a Thread Pool

```
pthread_t

pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                    void *(*start_routine) (void *), void *arg);

pthread_cond_t
pthread_cond_init(pthread_cond_t*,pthread_condattr_t*);
pthread_cond_wait(pthread_cond_t*, pthread_mutex_t)
pthread_cond_signal(pthread_cond_t*)
```

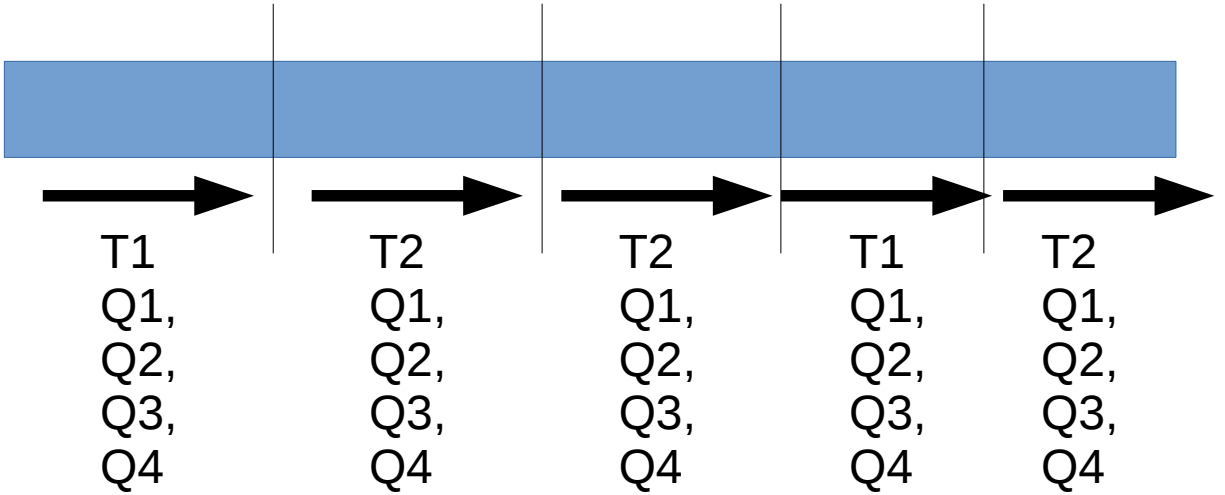
Link with `-lpthread`

Implementing a Thread Pool

```
// Worker thread waits for work
        pthread_mutex_lock(m);
        while ([queue_is_empty])
             pthread_cond_wait(cv, m);
        // Do critical something
        pthread_mutex_unlock(m);
```

```
// Producer thread adds work and signals
        pthread_mutex_lock(m);
        // Add some work to a shared struct
        pthread_cond_signal(cv); // wake up a thread
        pthread_mutex_unlock(m);
```

Threading a Shared Scan



Threading a Shared Scan