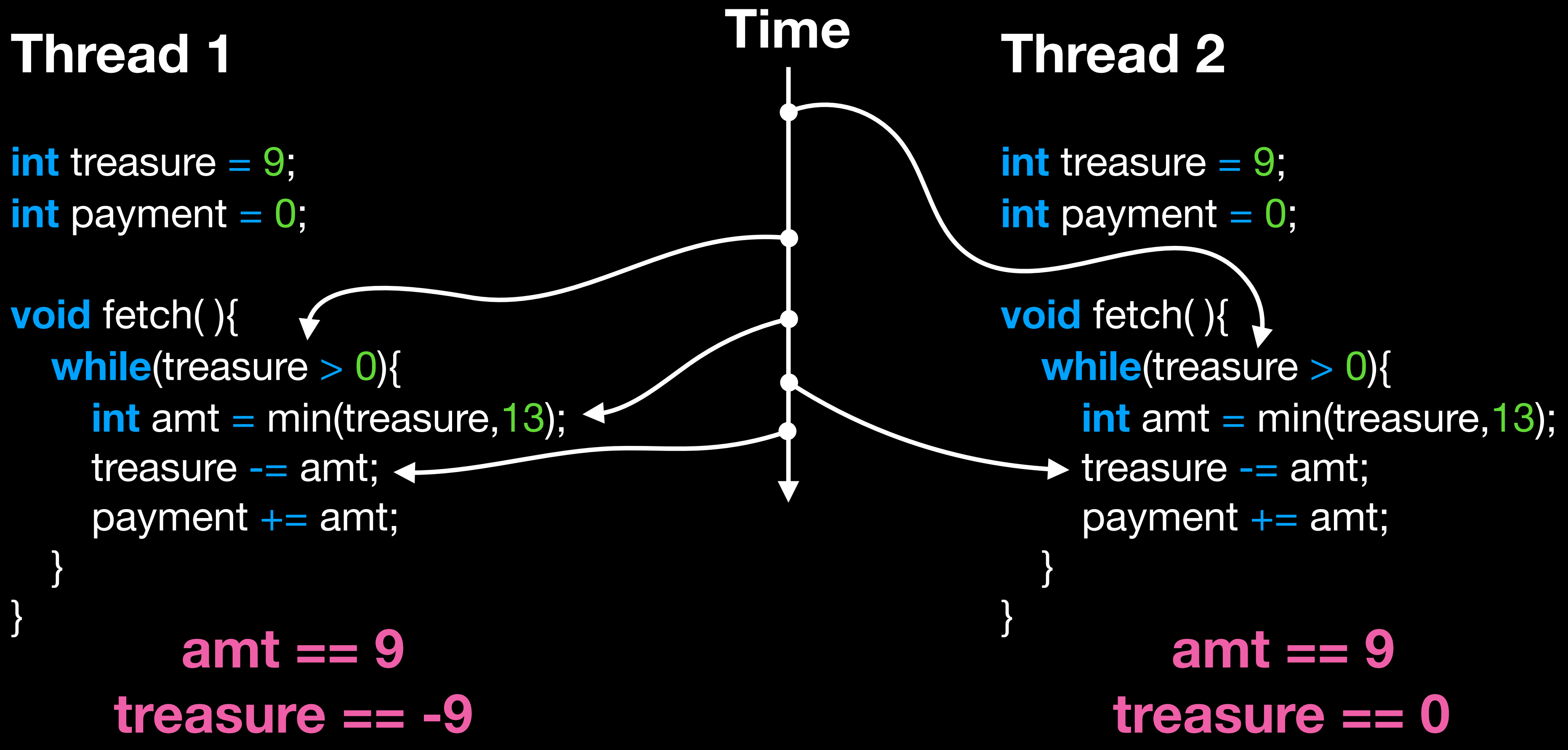
Stop go

Threads

thread



Thread 1

```
int treasure = 9;
int payment = 0;

void fetch( ){
   while(treasure > 0){
      int amt = min(treasure,13);
      treasure -= amt;
      payment += amt;
   }
}
```

amt == 9

treasure == -9

Thread 2

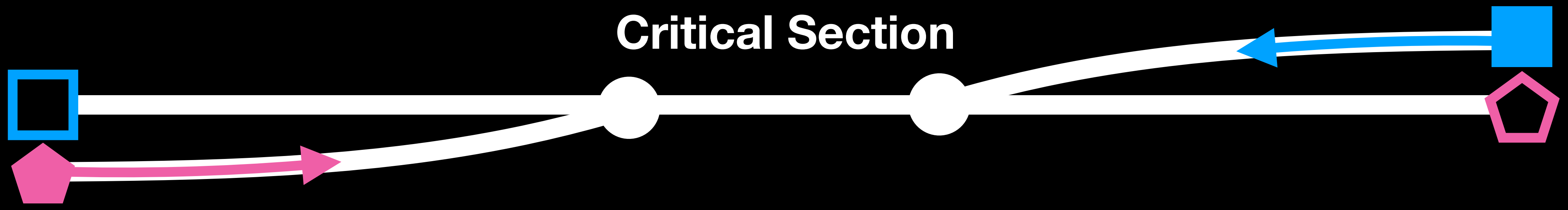
```
int treasure = 9;
int payment = 0;

void fetch( ){
   while(treasure > 0){
      int amt = min(treasure,13);
      treasure -= amt;
      payment += amt;
   }
}
```

amt == 9

treasure == 0

Race Conditions



Like two trains which need to share a single track
A race to the critical section.
Unpredictable results if both are in the critical section at the same time
Changing shared variables creates a critical section

thread

```
#include <iostream>
#include <thread>
#include <mutex>
using namespace std;
int treasure = 1000;
int payment = 0;
mutex mtx;

void fetch( ){
   while(treasure > 0){
      mtx.lock();
      int amt = min(treasure,13);
      treasure -= amt;
      payment += amt;
      mtx.unlock()
   }
}

int main(){
   thread diver(fetch);
   diver.join();
   cout << payment << endl;
   return 0;
}
```

Must be shared between threads

A **mutex** provides mutual exclusion
like a stop light

lock — before the critical section

unlock — after the critical section

mutex

- Use when modifying shared variables
- Limit the use or threads will be waiting doing nothing
- **Deadlock** is circular waiting that can't be resolved (I'm waiting on your lock and you are waiting on my lock)