Threads

Race conditions

Atomic Variables

Critical sections

BTW

- Read the week 9 advice on course website

Race Conditions

- Launch 2 threads, the outcome depends on which finishes first
  - Spurious, tough to reproduce (may need exacting set of conditions)
  - Because of this non-determinism they are **Tough to debug**

Each of these instructions
are really 3 assembly instructions

```
//A global int
int i=0;

//Thread 1
i++;

//Thread 2
i--;
```

See https://github.com/CNUClasses/thread_problem_atomic_solution.g

Race Conditions

- Launch 2 threads, the outcome depends on which finishes first
  - Spurious, tough to reproduce (may need exacting set of conditions)
  - Because of this non-determinism they are **Tough to debug**

```
//A global int
int i=0;
```

Each of these instructions are really 3 assembly instructions
The problem is you can't guarantee that all three will run to completion without being interrupted.
Want an atomic operation; a sequence of 1 or more operations that appear indivisible. No other process can see an intermediate state, once started cannot be interrupted

```
//Thread 1
i++;
```

```
//Thread 2
i--;
```

See https://github.com/CNUClasses/thread_problem_atomic_solution.g

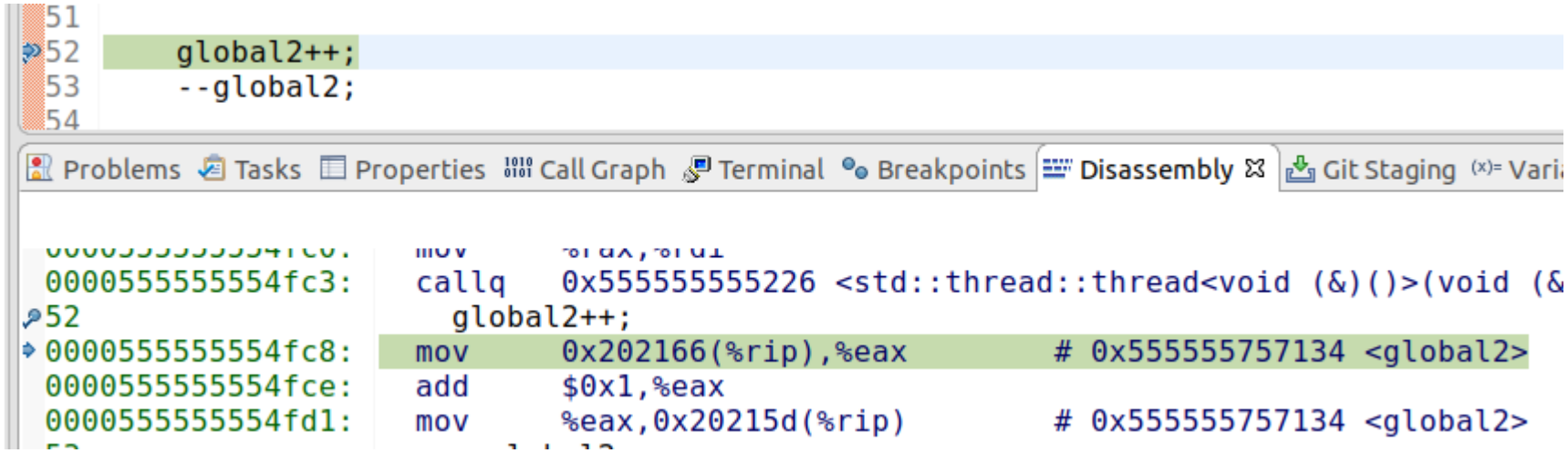
So, is this atomic?

```
global2++;
--global2;
```

See https://github.com/CNUClasses/thread_problem_atomic_solution.g

No, its 3, interruptible, machine instructions

```
global2++;
--global2;
```



Go to this project and demo non deterministic behaviour

↓

See https://github.com/CNUClasses/thread_problem_atomic_solution.g

OK make it atomic

Go From this

```
#include <thread>

using namespace std;
const int NUMB_TIMES = 100000;

//global variable
int global2 = 0;
```

OK make it atomic

Go From this

```
#include <thread>

using namespace std;
const int NUMB_TIMES = 100000;

//global variable
int global2 = 0;
```

To this

```
#include <thread>
#include <atomic>

using namespace std;
const int NUMB_TIMES = 100000;

//atomic variable
std::atomic<int> global2(0);
```

OK make it atomic

Go From this

```
#include <thread>

using namespace std;
const int NUMB_TIMES = 100000;

//global variable
int global2 = 0;
```

To this

```
#include <thread>
#include <atomic>

using namespace std;
const int NUMB_TIMES = 100000;

//atomic variable
std::atomic<int> global2(0);
```

Atomic types are types that encapsulate a value whose access is guaranteed to not cause data races and can be used to synchronize memory accesses among different threads.

OK make it atomic

Go From this

```
#include <thread>

using namespace std;
const int NUMB_TIMES = 100000;

//global variable
int global2 = 0;
```

To this

```
#include <thread>
#include <atomic>

using namespace std;
const int NUMB_TIMES = 100000;

//atomic variable
std::atomic<int> global2(0);
```

Go to this project and demo atomic solution

See https://github.com/CNUClasses/thread_problem_atomic_solution.g

But...

- Atomics protect single lines of code only.
- What if you have 3 lines that must be uninterruptable?

But...

- Atomics protect single lines of code only.
- What if you have 3 lines that must be uninterruptable? Like this

```
//starting balance
int bal =50;

void withdrawmoney(int amt){
    if (bal>amt){
        cout<<"approved!"<<endl;
        bal -=amt;
    }
    else
        cout<<"denied!";
}
```

```
int main() {
    thread t1(withdraw, 40);
    thread t2(withdraw, 25);
```

Go to this project to see this code

See https://github.com/CNUClasses/Thread_Race_condition.git

But...

- Atomics protect single lines of code only.
- What if you have 3 lines that must be uninterruptable? Like this

```
//starting balance
int bal =50;

void withdrawmoney(int amt){
    if (bal>amt){
        cout<<"approved!"<<endl;
        bal -=amt;
    }
    else
        cout<<"denied!";
}



int main() {
    thread t1(withdraw, 40);
    thread t2(withdraw, 25);
```

What happens if you are interrupted right after the if conditional check

But...

- Atomics protect single lines of code only.
- What if you have 3 lines that must be uninterruptable? Like this

```
//starting balance
int bal =50;

void withdrawmoney(int amt){
    if (bal>amt){
        cout<<"approved!"<<endl;
        bal -=amt;
    }
    else
        cout<<"denied!";
}
```

What happens if you are interrupted right after the if conditional check
Will not help to make bal an atomic (why?)

```
int main() {
    thread t1(withdraw, 40);
    thread t2(withdraw, 25);
```

But…

- Atomics protect single lines of code only.
- What if you have 3 lines that must be uninterruptable? Like this

```
//starting balance
int bal =50;

void withdrawmoney(int amt){
    if (bal>amt){
        cout<<"approved!"<<endl;
        bal -=amt;
    }
    else
        cout<<"denied!";
}
```

What is needed is to make these three lines uninterruptable

```
int main() {
    thread t1(withdraw, 40);
    thread t2(withdraw, 25);
```

But…

- Atomics protect single lines of code only.
- What if you have 3 lines that must be uninterruptable? Like this

```
//starting balance
int bal =50;

void withdrawmoney(int amt){
    if (bal>amt){
        cout<<"approved!"<<endl;
        bal -=amt;
    }
    else
        cout<<"denied!";
}


int main() {
    thread t1(withdraw, 40);
    thread t2(withdraw, 25);
```

What is needed is to make these three lines uninterruptable
We call this a “critical section”

Critical Section: Code that accesses a shared resource, that must complete without interruption. BTW make them as small as possible

But…

- Atomics protect single lines of code only.
- What if you have 3 lines that must be uninterruptable? Like this

```
//starting balance
int bal =50;

void withdrawmoney(int amt){
    if (bal>amt){
        cout<<"approved!"<<endl;
        bal -=amt;
    }
    else
        cout<<"denied!";
}



int main() {
    thread t1(withdraw, 40);
    thread t2(withdraw, 25);
```

What is needed is to make these three lines uninterruptable
We call this a “critical section”

Critical Section: Code that accesses a shared resource, that must complete without interruption. BTW make them as small as possible

Question: Can you have a critical section in a single threaded environment?

But…

- Atomics protect single lines of code only.
- What if you have 3 lines that must be uninterruptable? Like this

```
//starting balance
int bal =50;

void withdrawmoney(int amt){
    if (bal>amt){
        cout<<"approved!"<<endl;
        bal -=amt;
    }
    else
        cout<<"denied!";
}



int main() {
    thread t1(withdraw, 40);
    thread t2(withdraw, 25);
```

What is needed is to make these three lines uninterruptable
We call this a “critical section”

Critical Section: Code that accesses a shared resource, that must complete without interruption. BTW make them as small as possible

Question: Can you have a critical section in a single threaded environment? No

Critical Section

- Critical Section: Code that accesses a shared resource, where only 1 thread can be at a time.
- Make them as small as possible! Why? Because in the critical section you potentially go from a multithreaded application, to a single threaded application where the other threads are blocked waiting to get in.

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    thread t1(fun);

    int i=g;

    i++;

    g=i;

    t1.join();
:
}
```

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    //thread t1(fun);

    int i=g;

    i++;

    g=i;

    //t1.join();
:
}
```

If no threads?

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    //thread t1(fun);

    int i=g;

    i++;

    g=i;

    //t1.join();
:
}
```

If no threads?

If no threads then single threaded, no critical sections.

Where are critical sections in the following code?

```
int g=0;

void fun(){
    int a=g;
}

int main(){
    thread t1(fun);

    int i=g;

    i++;

    g=i;

    t1.join();:
}
```

If no threads?
If no threads then single threaded, no critical sections.

If fun() just reads g?

Where are critical sections in the following code?

```
int g=0;

void fun(){
    int a=g;
}

int main(){
    thread t1(fun);

    int i=g;

    i++;

1   g=i;

    t1.join();:
}
```

If no threads?
If no threads then single threaded, no critical sections.

If fun() just reads g?
g is being written at 1 If 1 write then all reads and writes need protection

Where are critical sections in the following code?

```
int g=0;

void fun(){
    int a=g;
}

int main(){
    thread t1(fun);

    int i=g;

    i++;

    g=i;

    t1.join();:
}
```

1

If no threads?
If no threads then single threaded, no critical sections.

If fun() just reads g?
g is being written at 1 If 1 write then all reads and writes need protection
See code in the rounded rectangle for critical sections

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    thread t1(fun);

    int i=g;

    i++;

    g=i;

    t1.join();:
}
```

1

If no threads?
If no threads then single threaded, no critical sections.

If fun() just reads g?
g is being written at 1 If 1 write then all reads and writes need protection
Will these smaller critical sections work (note fun changes)?

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    thread t1(fun);

    int i=g;

    i++;

    g=i;

    t1.join();:
}
```

1

If no threads?
If no threads then single threaded, no critical sections.

If fun() just reads g?
g is being written at
need protection

1

If 1 write then all reads and writes

Will these smaller critical sections work? (note fun changes)

NO

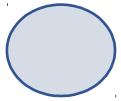
In main thread, if i receives g, then increment i,
Then t1 changes g, then main thread writes
i back to g? Answer: you overwrite t1s changes

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    thread t1(fun);

    int i=g;

    i++;

    g=i;

    t1.join();:
}
```

1

If no threads?
If no threads then single threaded, no critical sections.

If fun() just reads g?
g is being written at  If 1 write then all reads and writes need protection
See code in the rounded rectangle for critical sections

If thread starts in position 1

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    thread t1(fun);

    int i=g;

    i++;

    g=i;

    t1.join();:
}
```

1

If no threads?
If no threads then single threaded, no critical sections.

If fun() just reads g?
g is being written at
need protection
See code in the rounded rectangle for critical sections

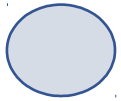
If 1 write then all reads and writes

If thread starts in position 1

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    int i=g;

2 →  thread t1(fun);

    i++;

    g=i;

    t1.join();:
}
```

If no threads?
If no threads then single threaded, no critical sections.

If fun() just reads g?
g is being written at  If 1 write then all reads and writes need protection
See code in the rounded rectangle for critical sections

If thread starts in position 2

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    int i=g;

    thread t1(fun);

    i++;

    g=i;

    t1.join();:
}
```

2

If no threads?
If no threads then single threaded, no critical sections.

If fun() just reads g?
g is being written at
need protection

If 1 write then all reads and writes

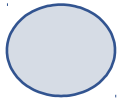
See code in the rounded rectangle for critical sections

If thread starts in position 2

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    int i=g;

    i++;

    g=i;

3 → thread t1(fun);

    t1.join();:
}
```

If no threads?
If no threads then single threaded, no critical sections.

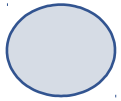
If fun() just reads g?
g is being written at  If 1 write then all reads and writes need protection
See code in the rounded rectangle for critical sections

If thread starts in position 3

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    int i=g;

    i++;

    g=i;

3 → thread t1(fun);

    t1.join();:
}
```

If no threads?
If no threads then single threaded, no critical sections.

If fun() just reads g?
g is being written at  If 1 write then all reads and writes need protection
See code in the rounded rectangle for critical sections

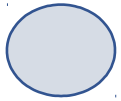
If thread starts in position 3

As written, only t1 will access g when the application
Is multithreaded. So there are no critical sections for position 3 as the global is never accessed in a multithreaded environment.

Where are critical sections in the following code?

```
int g=0;

void fun(){
    g++;
}

int main(){
    int i=g;

    i++;

    g=i;

3   thread t1(fun);
    g--;

    t1.join();:
}
```

If no threads?
If no threads then single threaded, no critical sections.

If fun() just reads g?
g is being written at  If 1 write then all reads and writes need protection
See code in the rounded rectangle for critical sections

If thread starts in position 3

As written, only t1 will access g when the application
Is multithreaded. So there are no critical sections for position 3 as the global is never accessed in a multithreaded environment

But it only takes a slight change to the code to cause problems
These types of changes often occur over the lifetime of the codebase

BTW…When only reading global variables

- If all you do is read a global variable, then there is no critical section and no need to protect access to the global variable.

```
//A global int
int i=0;
```

```
//Thread 1
int j=i;
```

```
//Thread 2
int k=i;
```

BTW… When only reading global variables

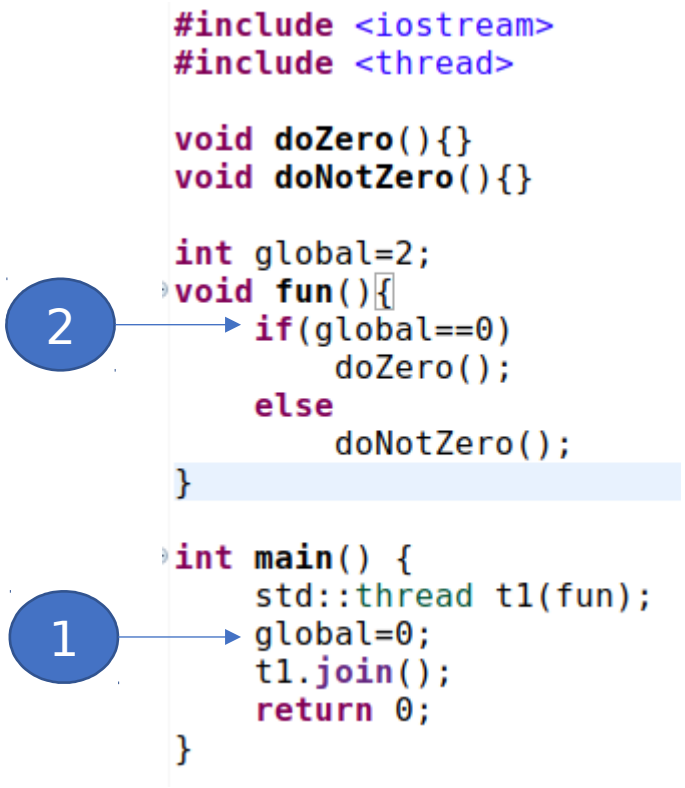
- If all you do is read a global variable, then there is no critical section and no need to protect access to the global variable.
- BUT, if you write a global variable at all. Even if just 1 write and 10000 reads.
- Then all 10001 operations are critical and all 10001 must be protected.

```
//A global int
int i=0;
```

```
//Thread 1
int j=i;
```

```
//Thread 2
int k=i;
```

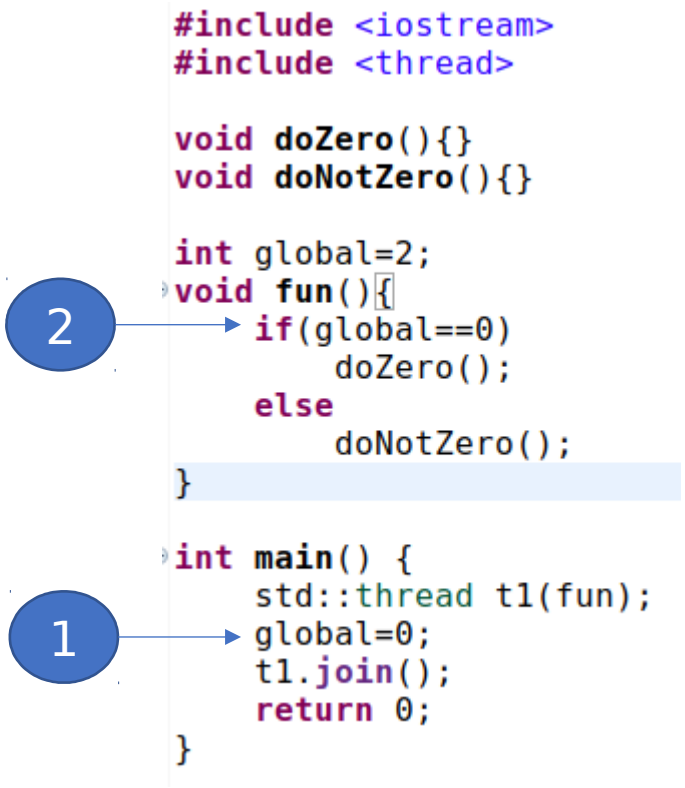
Race Condition again- what happens here?



- Do you execute doZero() or doNotZero()
- If 1 happens before 2
  - Then doZero()
- If 2 happens before 1
  - Then doNotZero()

How can you tell what happens?

Race Condition again- what happens here?



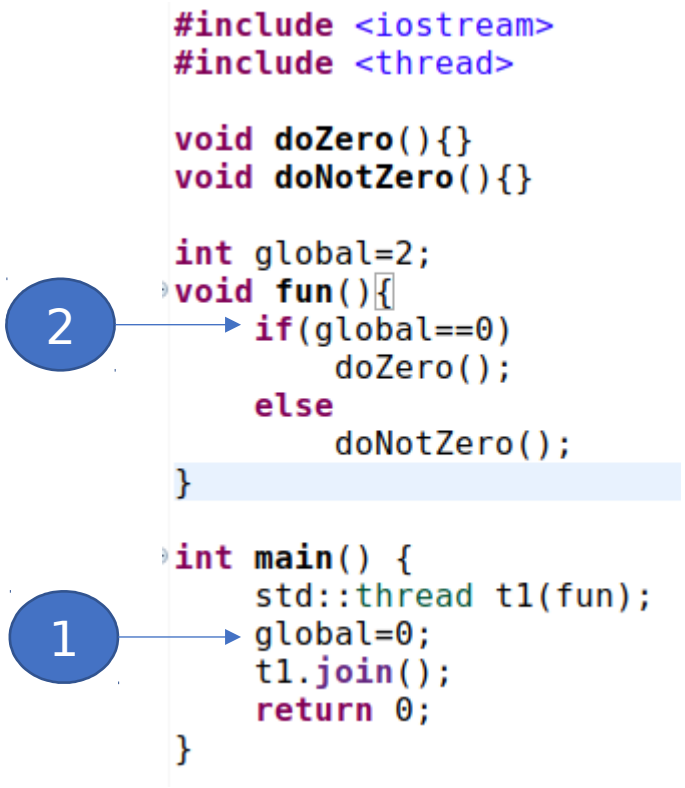

```
#include <iostream>
#include <thread>

void doZero(){}
void doNotZero(){}

int global=2;
void fun(){
    if(global==0)
        doZero();
    else
        doNotZero();
}

int main() {
    std::thread t1(fun);
    global=0;
    t1.join();
    return 0;
}
```

- Do you execute doZero() or doNotZero()
- If 1 happens before 2
  - Then doZero()
- If 2 happens before 1
  - Then doNotZero()

How can you tell what happens?

You cannot as written.

Race Condition again- what happens here?



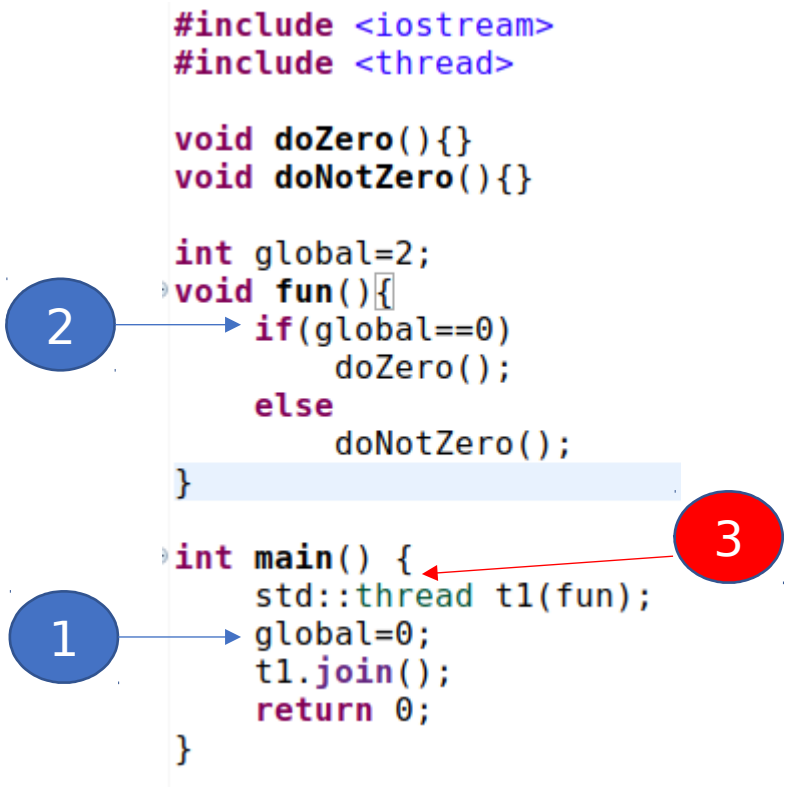
- Do you execute doZero() or doNotZero()?
- If 1 happens before 2
  - Then doZero()
- If 2 happens before 1
  - Then doNotZero()

How can you tell what happens?

You cannot as written.

You can however use condition variables to impose an order of your choice (later)

Race Condition again- what happens here?



- Do you execute doZero() or doNotZero()?
- If 1 happens before 2
  - Then doZero()
- If 2 happens before 1
  - Then doNotZero()

How can you tell what happens?

You cannot as written.

You can however use condition variables to impose an order of your choice (later)

Or move 1 to position 3

Race Condition again- A bogus solution

```
#include <iostream>
#include <thread>
#include <chrono>

void doZero(){}
void doNotZero(){}

int global=2;
void fun(){
    if(global==0)
        doZero();
    else
        doNotZero();
}

int main() {
    std::thread t1(fun);

    //when you see delays like this in the code with
    //comments like "wait for deposit to occur first"
    //or "wait for system stabalization" be very
    //suspicious of the code quality since this often means the
    //original developer has no idea how to coordinate thread activities
    //hint (use condition variables- coming soon)
    std::this_thread::sleep_for(std::chrono::milliseconds(500));    global=0;

    t1.join();
    return 0;
}
```

PSA- you may see code that "fixes" this with delays (see left). This is a cheesy, non scalable solution. (Why?)

DO NOT DO THIS!

Summary

- Race conditions- where they occur, learn to recognize them
- Atomics and problems they solve (single line only)
- Critical Sections- an area of code where only 1 thread can be at a time. Learn how to recognize them, make them small (since only one thread should be in them at a time)
  - Question- If you launch no threads, can you have critical sections?
  - Question-if you only read global variables, can you have critical sections?