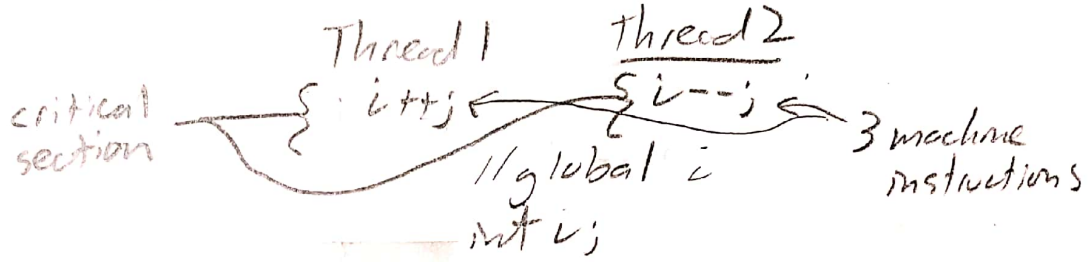
Race Condition - outcome depends on which thread finishes first. Often spurious, + tough to reproduce (may need exacting set of conditions). So usually tough to debug.

{see "Problems that threads can cause" for example race condition}



Atomic Operation sequence of 1 or more operations that appear indivisible. No other process can see an intermediate state or interrupt it.

```
int i;
i++; ← is this atomic? No it compiles to this
-----
mov, eax dword ptr [global (address)] //get it
                                      //increment
add eax, 1
mov dword ptr [global (address)]      //put back
```

can interrupt anywhere in the middle of above

3. {show how this leads to non-deterministic behavior in code}

---

solve with atomic<int> i, all 3 guaranteed to complete in 1 go.

- go thru building a program using threads
- hit breakpoint, show program is HTOP
  show TGID & PID.
- emphasize that you must wait for thread to exit before main thread exits (otherwise thread executing on reclaimed memory)
- BTW - what if you want to stop a thread?
- show 410_stop_threads

but atomics are single line only!
what if you have this? 3 lines must complete. Cannot use atomics!

```
int bal = 50;
void withdraw(int amt){
    if (bal >amt){
        cout << " approved" <<endl;
        bal -= amount;
    }
}
```

if interrupt here what happens?

Critical Section — code that accesses shared resource that must complete w/o interruption!

can be simple like above, can also be complex. Sometimes tricky

Thread 1

```
int j = i;
```

Thread 2

```
int k = i
```

```
// global
int i
```

if all you do is read a variable then no critical section, no need to protect. The first time you write the var, even if 100 reads & 1 write, then all 101 operations are critical & must be protected.

Board examples - critical sections, race conditions

ex

```
int balance = 50;

void withdraw (int amount) {
    if (balance > amount) {
        cout << "approved";
        balance -= amount;
    }
}
```

2 threads start

T1 withdraw 40
T2 withdraw 25
} 40+25 = 65 > 50 !!

sequence	Balance After
T1 starts gets to 3 (past the filter)	50
T1 preempted by T2	50
T2 runs from 1 → 5	25
T1 switched back in	-15

race condition - when program output depends on what gets done first.

race condition

ex.

```
int global = 2;

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

int main(){
    thread myt(fun);
    global = 0;
    myt.join();
}
```

② ③ ①

execute doZero() or doNotZero()?

if ① happens before ②
    doZero();
if ② happens before ①
    doNotZero();

how can you tell?

you cannot as written

you can however use condition variables to synchronize these (label) (or move line 1 to position 3)

PSA you may see code that "fixes" this with delays (sleep_for(), sleep_until()) this is a cheesy non scalable solution. Do not do this

stopped here

2/12/19 1st class

critical section - an area of code where only 1 thread can be at a time.

Question: if you launch no threads, do you have CS's?
Answer: No,

Question: if you only read global vars do you need synchronization?
No, but 1st write then every rdr needs it!

ex.

```
int gi=0;

void fun(){
    gi++;
}

int main(){
// thread t1(fun)
    int i=gi;
    i = i+1;
    gi = i;

t1.join();
```

① start here → // thread t1(fun)

② or → int i=gi;

③ or → i = i+1;

gi = i;

where are critical section(s)

if no thread

if fun just reads gi

if thread started in ① ② or ③