Listing 5.3

```
#include <iostream>

void foo(int a,int b)
{
    std::cout<<a<<","<<b<<std::endl;
}

int get_num()
{
    static int i=0;
    return ++i;
}

int main()
{
    foo(get_num(),get_num());
}
```

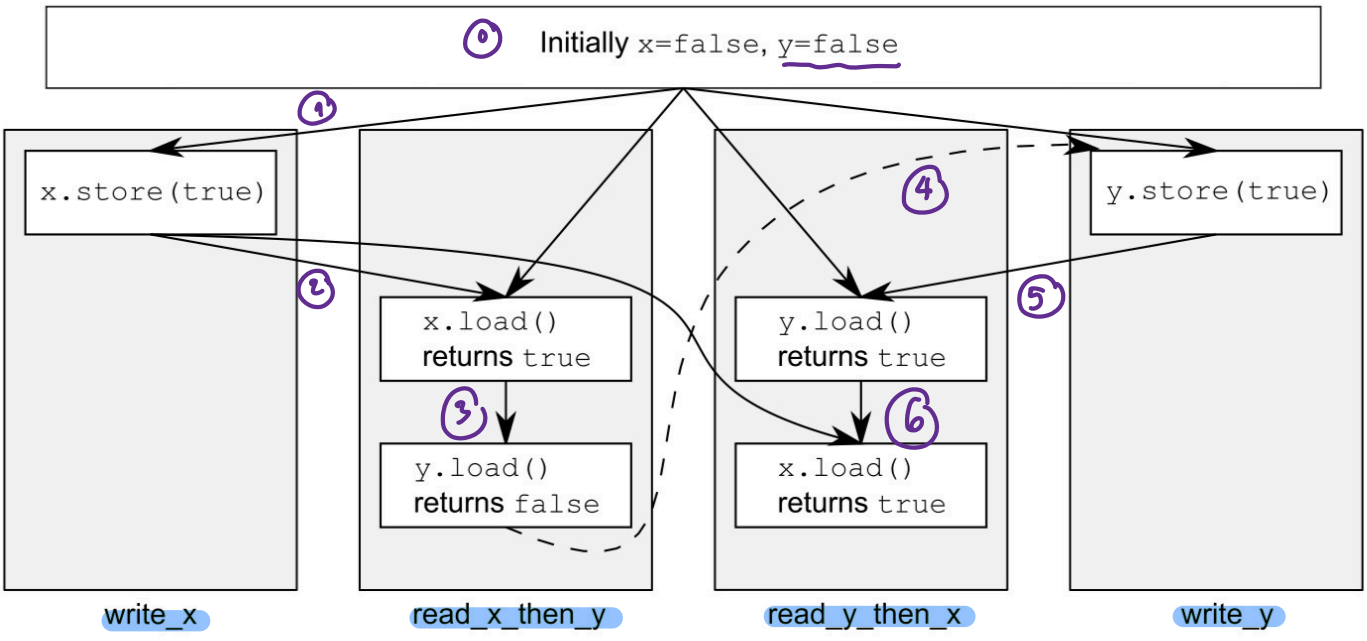


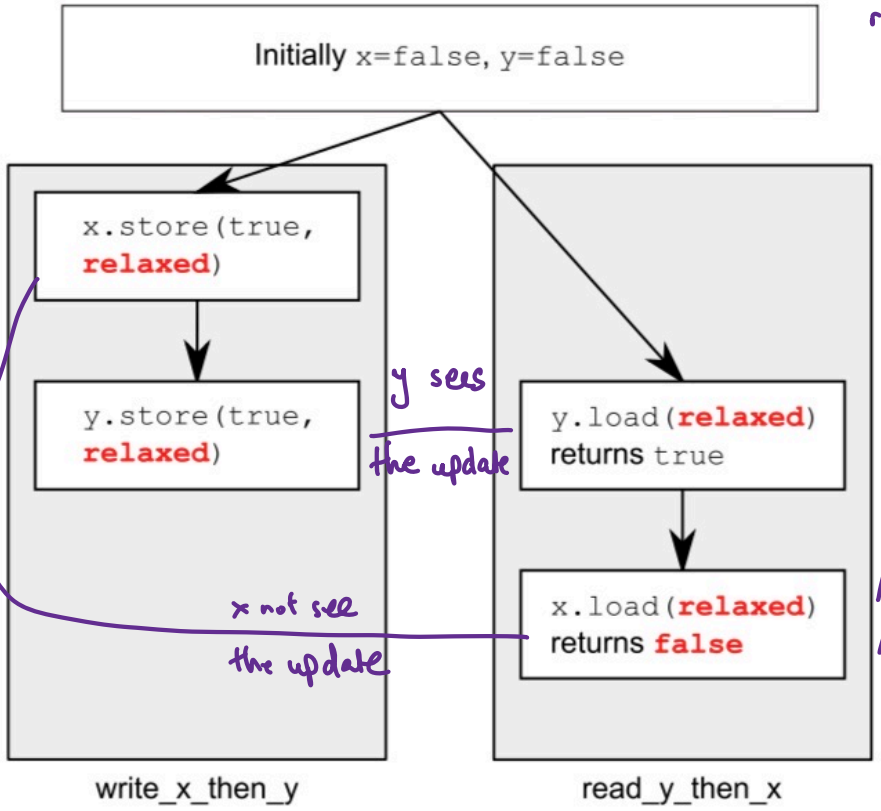
Figure 5.3 Sequential consistency and happens-before

↳ slows your program down.

Listing 5.4

```
#include <atomic>
#include <thread>
#include <assert.h>

std::atomic<bool> x,y;
std::atomic<int> z;

void write_x()
{
    x.store(true,std::memory_order_seq_cst);
}

void write_y()
{
    y.store(true,std::memory_order_seq_cst);
}

void read_x_then_y()
{
    while(!x.load(std::memory_order_seq_cst));
    if(y.load(std::memory_order_seq_cst))
        ++z;
}

void read_y_then_x()
{
    while(!y.load(std::memory_order_seq_cst));
    if(x.load(std::memory_order_seq_cst))
        ++z;
}

int main()
{
    x=false;
    y=false;
    z=0;
    std::thread a(write_x);
    std::thread b(write_y);
    std::thread c(read_x_then_y);
    std::thread d(read_y_then_x);
    a.join();
    b.join();
    c.join();
```

```
    d.join();
    assert(z.load()!=0);
}
```



relaxed ordering

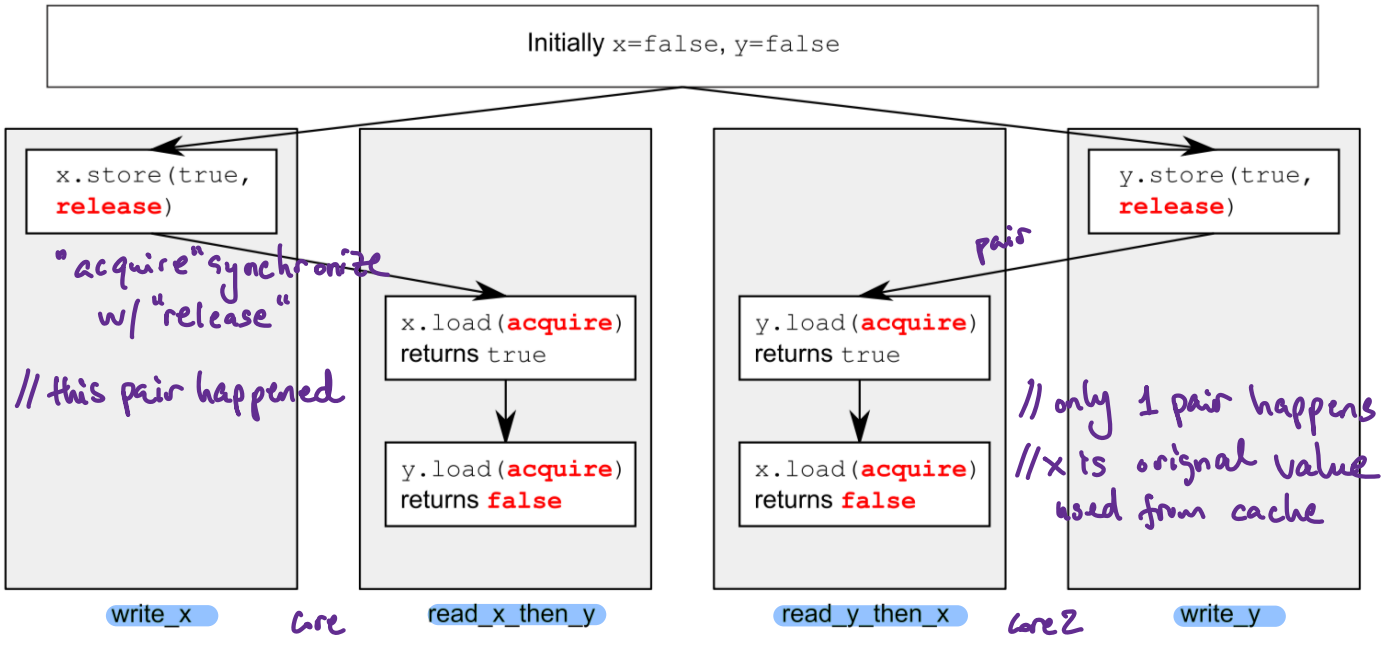
y sees the update

x not see the update

// original value
// that's totally allowable for relaxed ordering.

Figure 5.4 Relaxed atomics and happens-before

// runs much more quickly but the order isn't cared



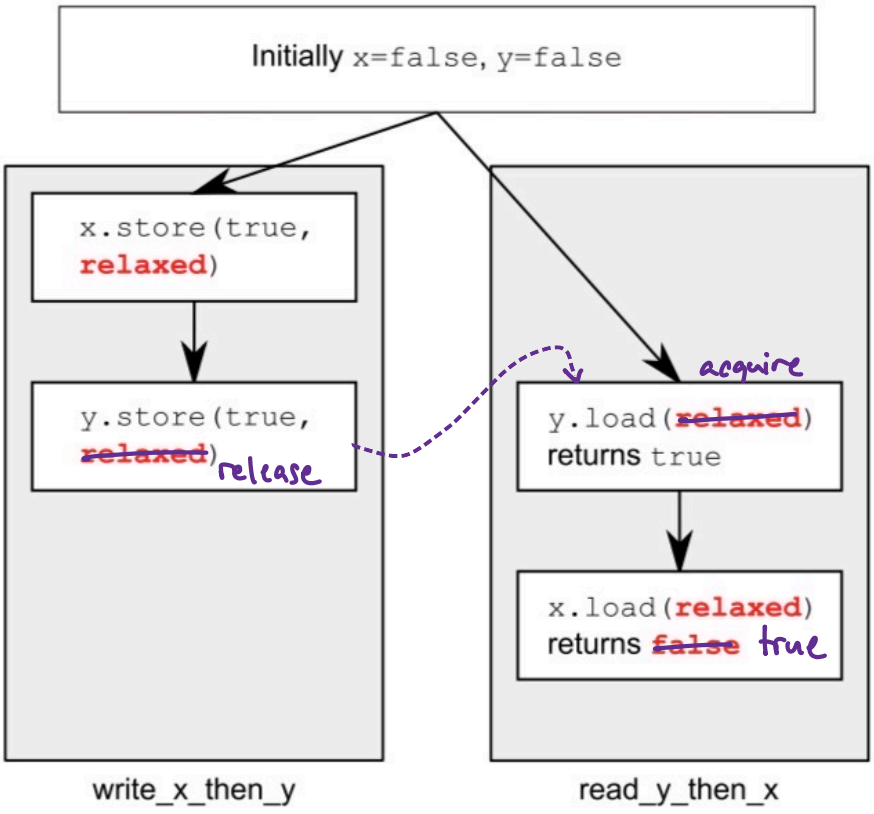
"acquire" synchronize w/ "release"

// this pair happened

pair

// only 1 pair happens
// x is original value used from cache

Figure 5.6 Acquire-release and happens-before



b/c synchronization of y.store/load w/ release/acquire.
x.store happens before x.load, so "relaxed" sees the update.

release/acquire and

Figure 5.4 Relaxed atomics and happens-before