CSE113: Parallel Programming

March 3, 2023

- **Topics**:
  - Finish up barriers
  - Memory consistency models:
    - Total store order
    - Relaxed memory consistency
    - Examples

target machine
TSO (x86)

	L	S
L	NO	different address
S	NO	No

Announcements

- Midterm grades are out
  - Let us know within 1 week if there are issues
- We are working on HW 2 grades and will strive to get them released by Monday
- Homework 4 is out!
  - Should be able to do most of it (part 1 and some of the others) by end of lecture

Previous quiz

Barrier objects have how many API calls?

- ○ 0
- ○ 1
- ○ 2
- ○ 3

Previous quiz

A barrier call emits which of the following events? Check all that apply

- [ ] barrier_lock
- [ ] barrier_arrive
- [ ] barrier_enqueue
- [ ] barrier_leave

Previous quiz

If a program uses both barriers and mutexes, the outcome is deterministic (i.e. the same every time) if there are no data conflicts

- ◯ True
- ◯ False

Example

Previous quiz

Write a few sentences about what you think the best interface for parallel programming is: that is: do you think it is atomics? Mutexes? Concurrent Data Structures? Barriers? Or even maybe the compiler should simply do it all automatically? Or is it some combination of the above? What are the trade-offs involved?

Barriers

- Intuition: threads stop and wait for each other:
  - Threads ***arrive*** at the barrier
  - Threads ***wait*** at the barrier
  - Threads ***leave*** the barrier once all other threads have arrived

Example: say there are 4 threads:

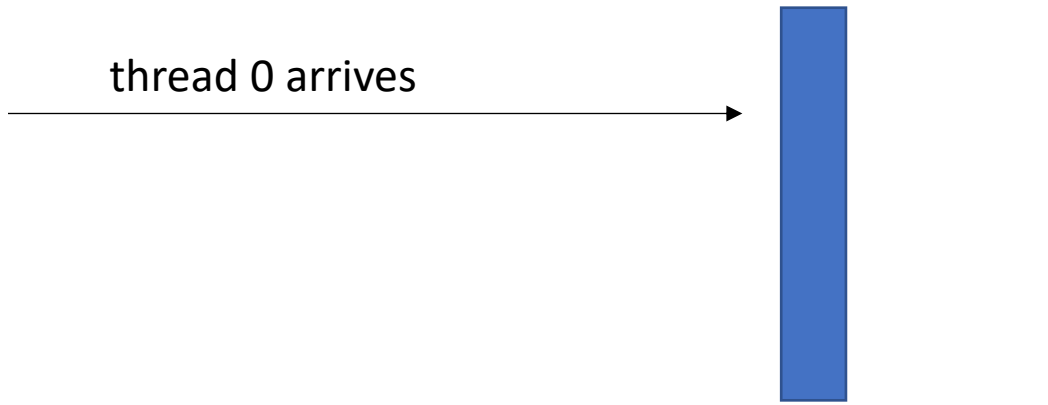
```
barrier();
```

Barriers

- Intuition: threads stop and wait for each other:
  - Threads ***arrive*** at the barrier
  - Threads ***wait*** at the barrier
  - Threads ***leave*** the barrier once all other threads have arrived

Example: say there are 4 threads:

```
barrier();
```

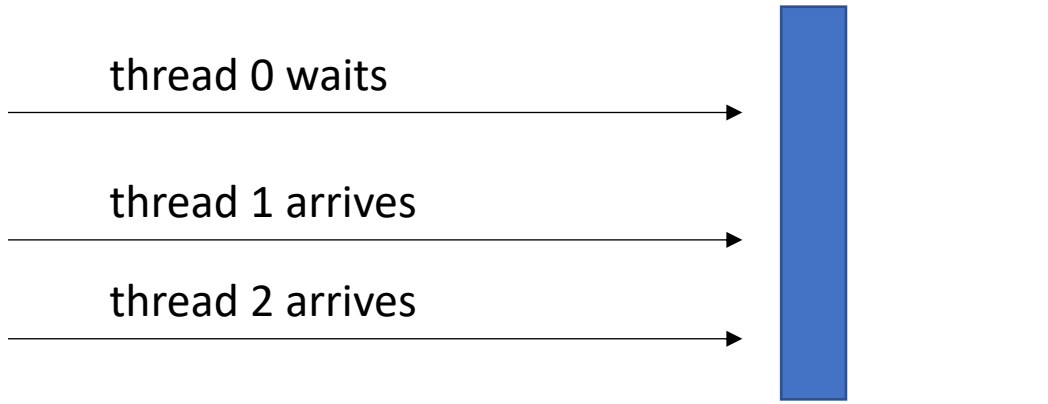


Barriers

- Intuition: threads stop and wait for each other:
  - Threads ***arrive*** at the barrier
  - Threads ***wait*** at the barrier
  - Threads ***leave*** the barrier once all other threads have arrived

Example: say there are 4 threads:

```
barrier();
```

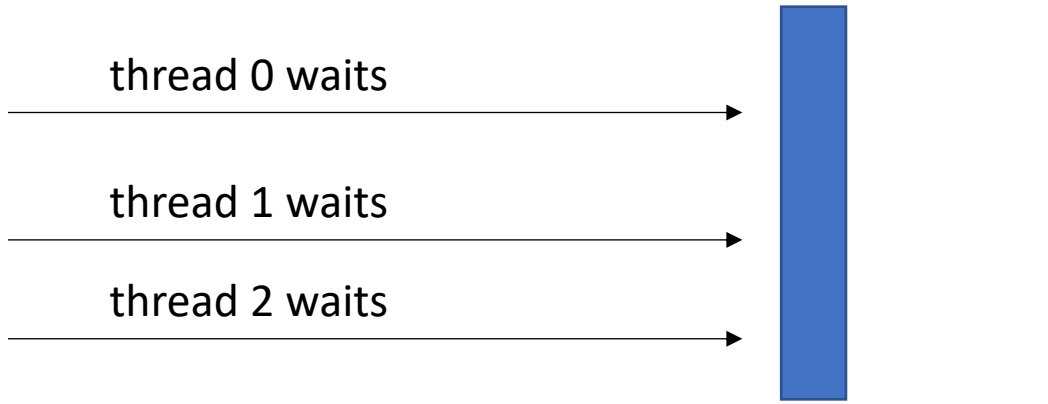


Barriers

- Intuition: threads stop and wait for each other:
  - Threads ***arrive*** at the barrier
  - Threads ***wait*** at the barrier
  - Threads ***leave*** the barrier once all other threads have arrived

Example: say there are 4 threads:

```
barrier();
```

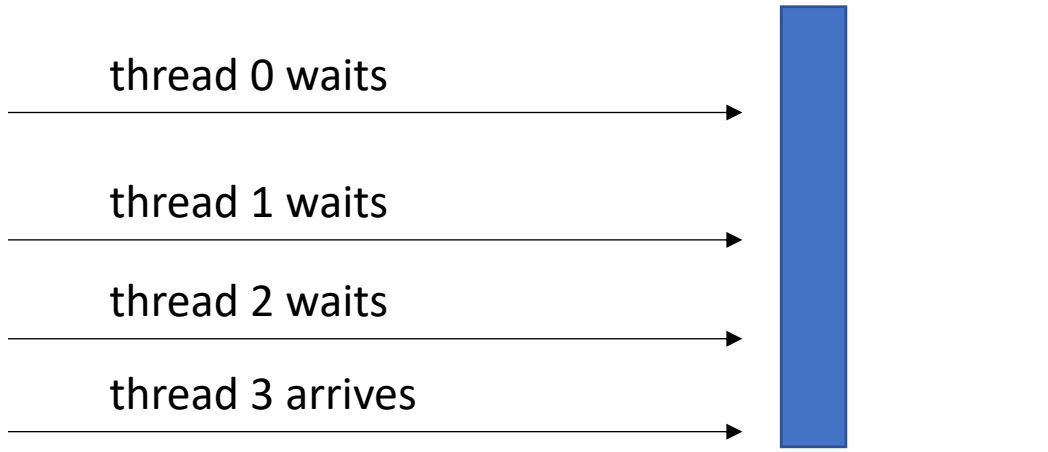


Barriers

- Intuition: threads stop and wait for each other:
  - Threads ***arrive*** at the barrier
  - Threads ***wait*** at the barrier
  - Threads ***leave*** the barrier once all other threads have arrived

Example: say there are 4 threads:

```
barrier();
```

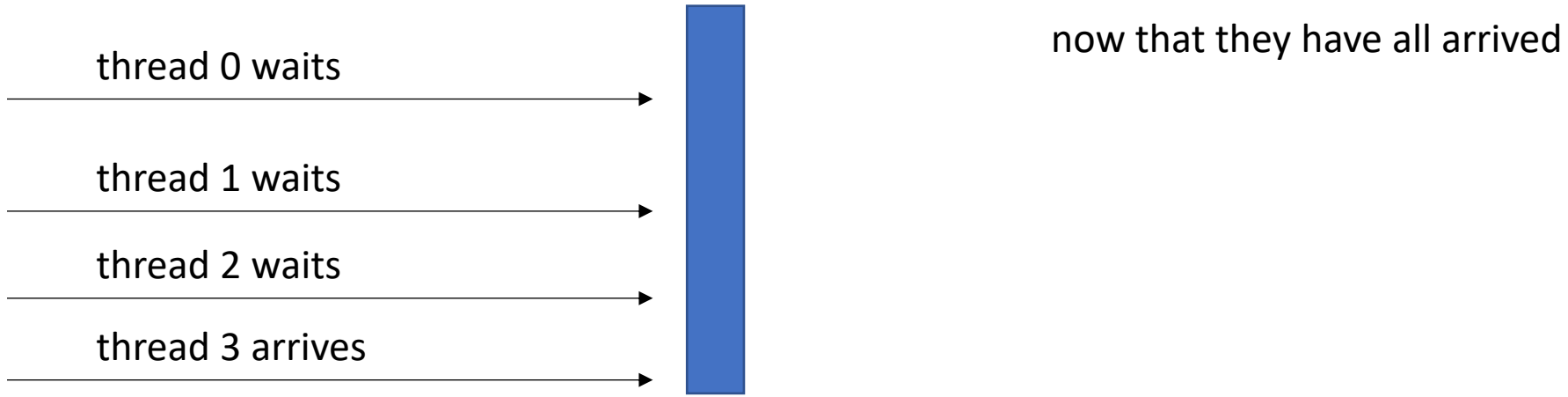


Barriers

- Intuition: threads stop and wait for each other:
  - Threads ***arrive*** at the barrier
  - Threads ***wait*** at the barrier
  - Threads ***leave*** the barrier once all other threads have arrived

Example: say there are 4 threads:

```
barrier();
```

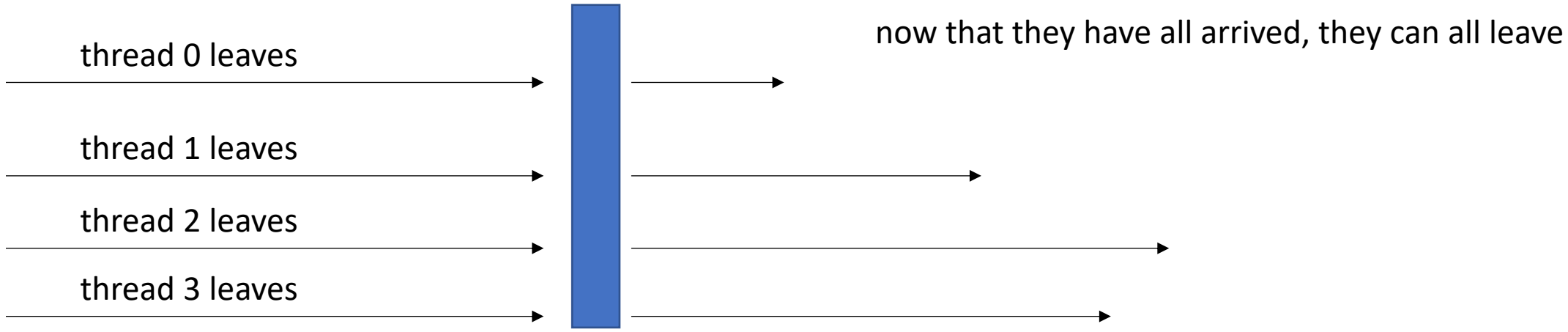


Barriers

- Intuition: threads stop and wait for each other:
  - Threads ***arrive*** at the barrier
  - Threads ***wait*** at the barrier
  - Threads ***leave*** the barrier once all other threads have arrived

Example: say there are 4 threads:

```
barrier();
```



num_threads == 2
counter == 1

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier() {
       int arrival_num = atomic_fetch_add(&counter, 1);
       if (arrival_num == num_threads - 1) {
          counter.store(0);
       }
       else {
         while (counter.load() != 0);
       }
    }
```

Thread 1:

```
B.barrier();
B.barrier();
```

Thread 1 wakes up! Doesn't think its missed anything

arrival_num == 0

arrival_num = 0

in a perfect world,
thread 1 executes now and leaves the barrier

Both threads get stuck here!

Sense Reversing Barrier

- Book Chapter 17
- Alternating ”sense” dynamically

Thread 0:

```
B.barrier();
B.barrier();
```

sync on sense = true

Thread 1:

```
B.barrier();
B.barrier();
```

```
class SenseBarrier {
  private:
    atomic_int counter;
    int num_threads;
    atomic_bool sense;
    bool thread_sense[num_threads];
  public:
    Barrier(int num_threads) {
      counter = 0;
      this->num_threads = num_threads;
      sense = false;
      thread_sense = {true, ...};
    }

     void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
     }
}
```

thread_sense = true

```
num_threads == 2
counter == 0
sense = false
```

thread_sense = true

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
     }
```

Thread 1:

```
B.barrier();
B.barrier();
```

num_threads == 2
counter == 2
sense = false

thread_sense = true
arrival_num = 1

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = true
arrival_num = 0

Thread 1:

```
B.barrier();
B.barrier();
```

num_threads == 2
counter == 2
sense = false

thread_sense = true
arrival_num = 1

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
            counter.store(0);
            sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = true
arrival_num = 0

Thread 1:

```
B.barrier();
B.barrier();
```

num_threads == 2
counter == 0
sense = true

thread_sense = false
arrival_num = 1

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = true
arrival_num = 0

Thread 1:

```
B.barrier();
B.barrier();
```

num_threads == 2
counter == 0
sense = true

thread_sense = false
arrival_num = ?

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = true
arrival_num = 0

Thread 1:

```
B.barrier();
B.barrier();
```

Remember the issue! Thread 1 went to sleep around this time and thread 0 went into the barrier again!

num_threads == 2
counter == 1
sense = true

thread_sense = false
arrival_num = 0

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = true
arrival_num = 0

Thread 1:

```
B.barrier();
B.barrier();
```

num_threads == 2
counter == 1
sense = true

thread_sense = false
arrival_num = 0

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = true
arrival_num = 0

Thread 1:

```
B.barrier();
B.barrier();
```

both are waiting!,
but thread 1 can leave

num_threads == 2
counter == 1
sense = true

thread_sense = false
arrival_num = 0

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = false
arrival_num = 0

Thread 1:

```
B.barrier();
B.barrier();
```

both are waiting!,
but thread 1 can leave

num_threads == 2
counter == 1
sense = true

thread_sense = false
arrival_num = 0

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
      }
```

thread_sense = false
arrival_num = ?

Thread 1:

```
B.barrier();
B.barrier();
```

Thread 1 finishes the barrier

num_threads == 2
counter == 1
sense = true

thread_sense = false
arrival_num = 0

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = false
arrival_num = ?

Thread 1:

```
B.barrier();
B.barrier();
```

Goes into the second barrier

num_threads == 2
counter == 2
sense = true

thread_sense = false
arrival_num = 0

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = false
arrival_num = 1

Thread 1:

```
B.barrier();
B.barrier();
```

Goes into the second barrier

num_threads == 2
counter == 2
sense = true

thread_sense = false
arrival_num = 0

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = false
arrival_num = 1

Thread 1:

```
B.barrier();
B.barrier();
```

Goes into the second barrier

```
num_threads == 2
counter == 0
sense = false
```

thread_sense = false

arrival_num = 0

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = false

arrival_num = 1

Thread 1:

```
B.barrier();
B.barrier();
```

Goes into the second barrier

num_threads == 2
counter == 0
sense = false

thread_sense = false
arrival_num = 0

Thread 0:

```
B.barrier();
B.barrier();
```

```
void barrier(int tid) {
        int arrival_num = atomic_fetch_add(&counter, 1);
        if (arrival_num == num_threads-1) {
           counter.store(0);
           sense = thread_sense[tid];
        }
        else {
          while (sense != thread_sense[tid]);
        }
        thread_sense[tid] = !thread_sense[tid];
    }
```

thread_sense = false
arrival_num = 1

Thread 1:

```
B.barrier();
B.barrier();
```

thread 0 can leave, thread 1 can leave and the barrier works as expected!

Moving on to memory consistency!

- One of my favorite topics!

Moving on to memory consistency!

- One of my favorite topics!
- What do other people think?

> Look, memory ordering pretty much _is_ the rocket science of CS, but the C standards committee basically made it a ton harder by specifying "we have to make the rocket out of duct tape and bricks, and only use liquid hydrogen as a propellant".
>
> Linus

Memory Consistency

- We have been very strict about using atomic types in this class
  - and the methods (.load and .store)
  - why?

  - Architectures do very strange things with memory loads and stores
  - Compilers do too (but we won't talk too much about them today)

  - C++ gives us sequential consistency if we use atomic types and operations
  - What do we remember sequential consistency from?

Sequential consistency for atomic memory

- Let’s play our favorite game:

Global variable:

```
atomic_int x(0);
atomic_int y(0);
```

Thread 0:

```
x.store(1);
y.store(1);
```

Is it possible for
`t0 == 0 and t1 ==1`

Thread 1:

```
int t0 = y.load();
int t1 = x.load();
```

Global variable:

```
atomic_int x(0);
atomic_int y(0);
```

Thread 0:

```
x.store(1);
y.store(1);
```

Is it possible for
`t0 == 0 and t1 ==1`

Thread 1:

```
int t0 = y.load();
int t1 = x.load();
```

```
x.store(1);
```

```
y.store(1);
```

```
int t0 = y.load();
```

```
int t1 = x.load();
```

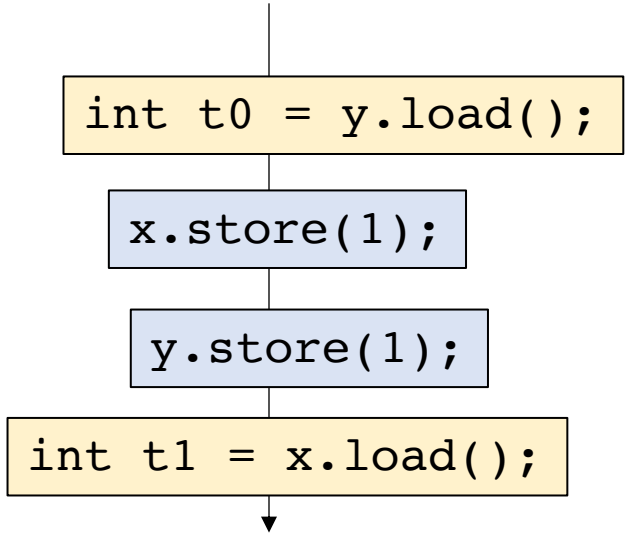
Global variable:

```
atomic_int x(0);
atomic_int y(0);
```

Thread 0:

```
x.store(1);
y.store(1);
```

Is it possible for
`t0 == 0 and t1 ==1`



Thread 1:

```
int t0 = y.load();
int t1 = x.load();
```

Global variable:

```
atomic_int x(0);
atomic_int y(0);
```

How about:

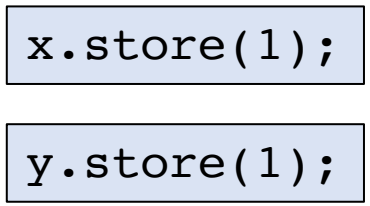
Is it possible for
`t0 == 1 and t1 ==0`

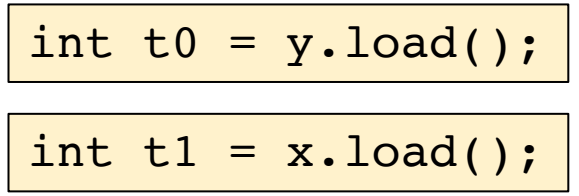
Thread 0:

```
x.store(1);
y.store(1);
```

Thread 1:

```
int t0 = y.load();
int t1 = x.load();
```





Global variable:

```
atomic_int x(0);
atomic_int y(0);
```

How about:

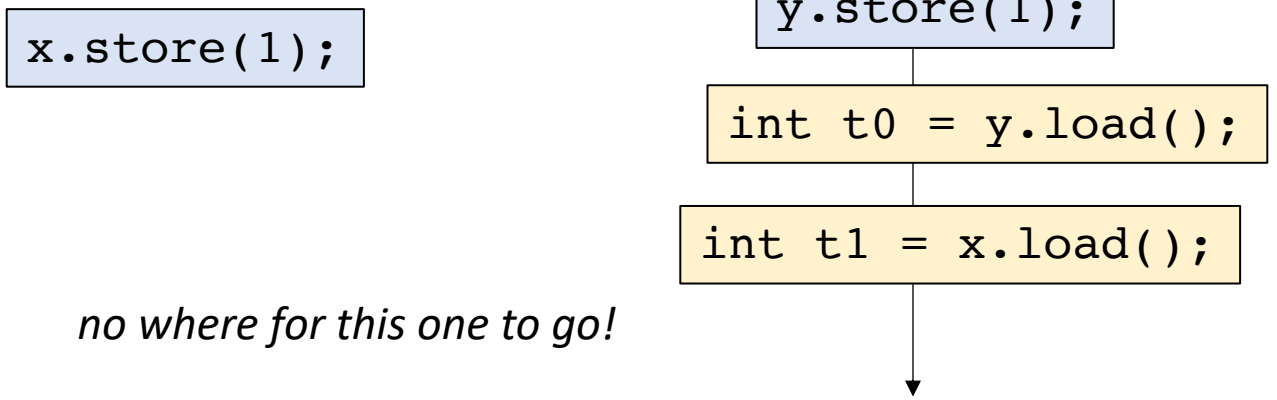
Is it possible for
`t0 == 1 and t1 ==0`

Thread 0:

```
x.store(1);
y.store(1);
```

Thread 1:

```
int t0 = y.load();
int t1 = x.load();
```



no where for this one to go!

Global variable:

```
atomic_int x(0);
atomic_int y(0);
```

Another test
Can `t0 == t1 == 0`?

Thread 0:

```
x.store(1);
int t0 = y.load();
```

Thread 1:

```
y.store(1);
int t1 = x.load();
```

Global variable:

```
atomic_int x(0);
atomic_int y(0);
```

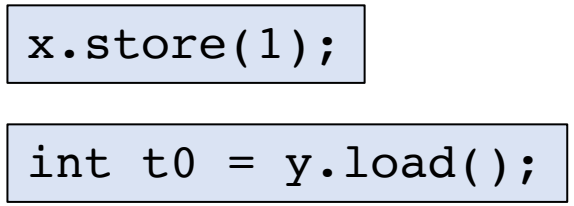
Another test
Can `t0 == t1 == 0`?

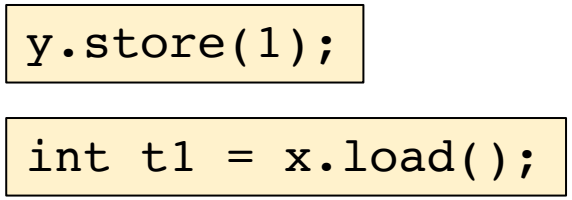
Thread 0:

```
x.store(1);
int t0 = y.load();
```

Thread 1:

```
y.store(1);
int t1 = x.load();
```





Global variable:

```
atomic_int x(0);
atomic_int y(0);
```

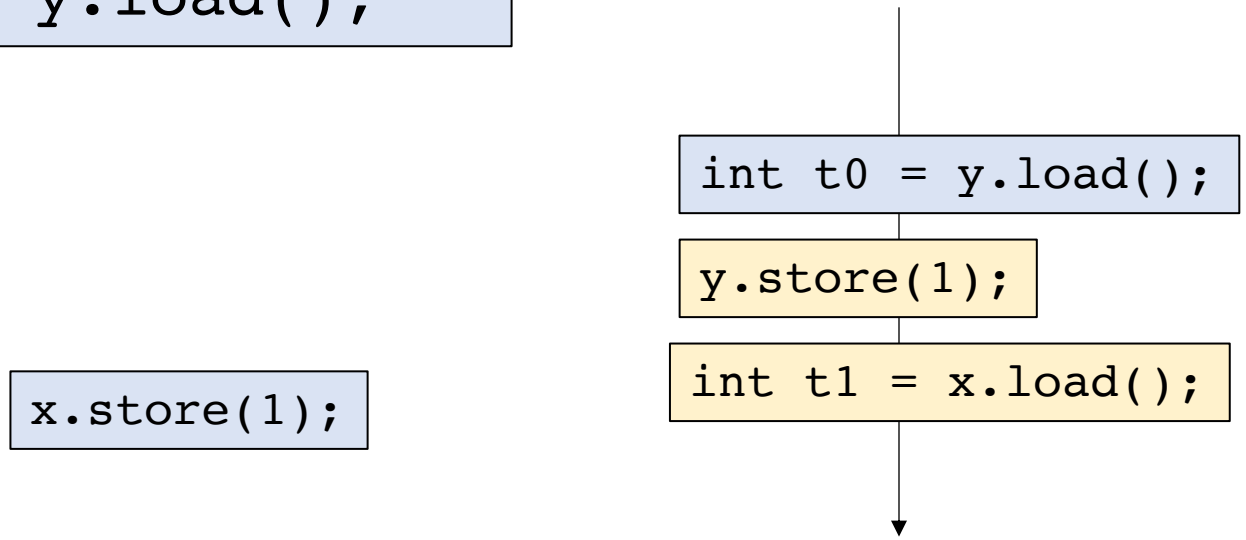
Another test
Can `t0 == t1 == 0`?

Thread 0:

```
x.store(1);
int t0 = y.load();
```

Thread 1:

```
y.store(1);
int t1 = x.load();
```



no place for this one!

C++

- Plain atomic accesses are documented to be sequentially consistent (SC)

- Why wasn’t SC very good for concurrent data structures?
  - Compossibility: two objects that are SC might not be SC when used together

  - Programs contain only 1 shared memory though; no reason to compose different main memories.

What about ISAs?

- Remember, it is important for us to understand how our code executes on the architecture to write high performing programs

- Lets think about x86
  - Instructions:
  - `MOV %t0 [x]` - loads the value at x to register t0
  - `MOV [y] 1` - stores the value 1 to memory location y

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Another test

Can `t0 == t1 == 0`?

Thread 0:

```
mov [x], 1
mov %t0, [y]
```

Thread 1:

```
mov [y], 1
mov %t1, [x]
```

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

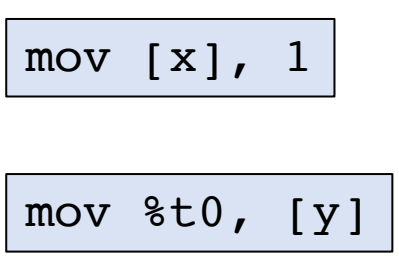
Another test
Can `t0 == t1 == 0`?

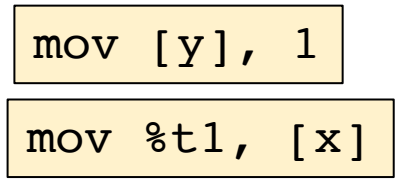
Thread 0:

```
mov [x], 1
mov %t0, [y]
```

Thread 1:

```
mov [y], 1
mov %t1, [x]
```





Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

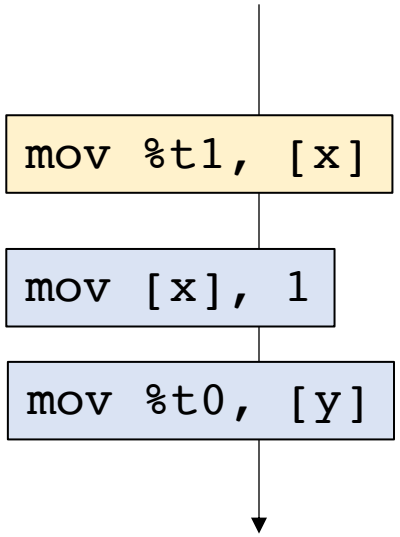
Another test
Can `t0 == t1 == 0`?

Thread 0:

```
mov [x], 1
mov %t0, [y]
```

Thread 1:

```
mov [y], 1
mov %t1, [x]
```




```
mov [y], 1
```

no place for this event!

What if we actually run this code?

- We’d like to be able to compile atomic instructions just to regular ISA loads and stores

Schedule

- Memory consistency models:
  - **Total store order**
  - Relaxed memory consistency
  - Examples

Thread 0:

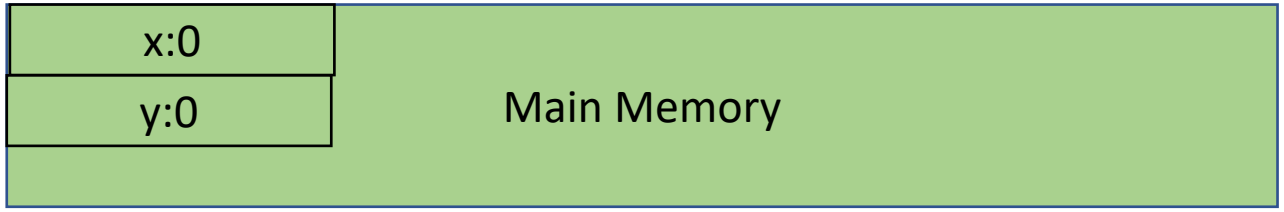
```
mov [x], 1
mov %t0, [y]
```

Core 0

Thread 1:

```
mov [y], 1
mov %t1, [x]
```

Core 1



Thread 0:

```
mov %t0, [y]
```

Core 0

```
mov [x], 1
```

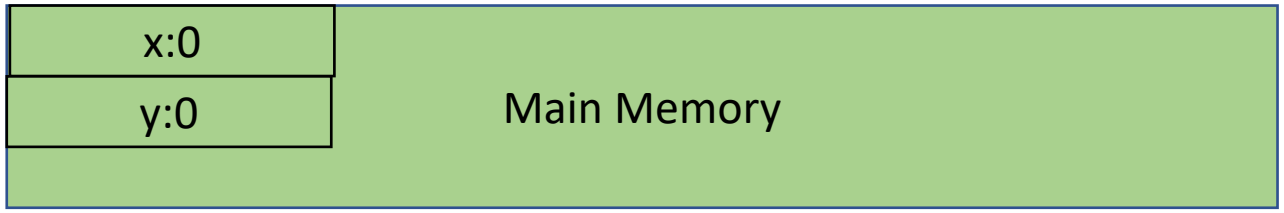
execute first instruction
what happens to the stores?

Thread 1:

```
mov %t1, [x]
```

Core 1

```
mov [y], 1
```



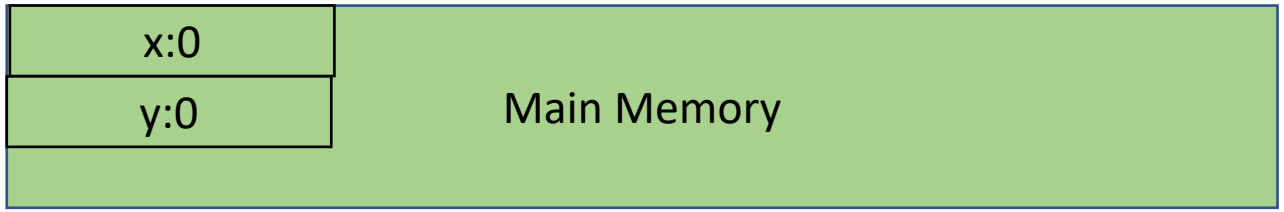
Thread 0:

Thread 1:

X86 cores contain a store buffer; holds stores before going to main memory

```
mov %t0, [y]
```

```
mov %t1, [x]
```



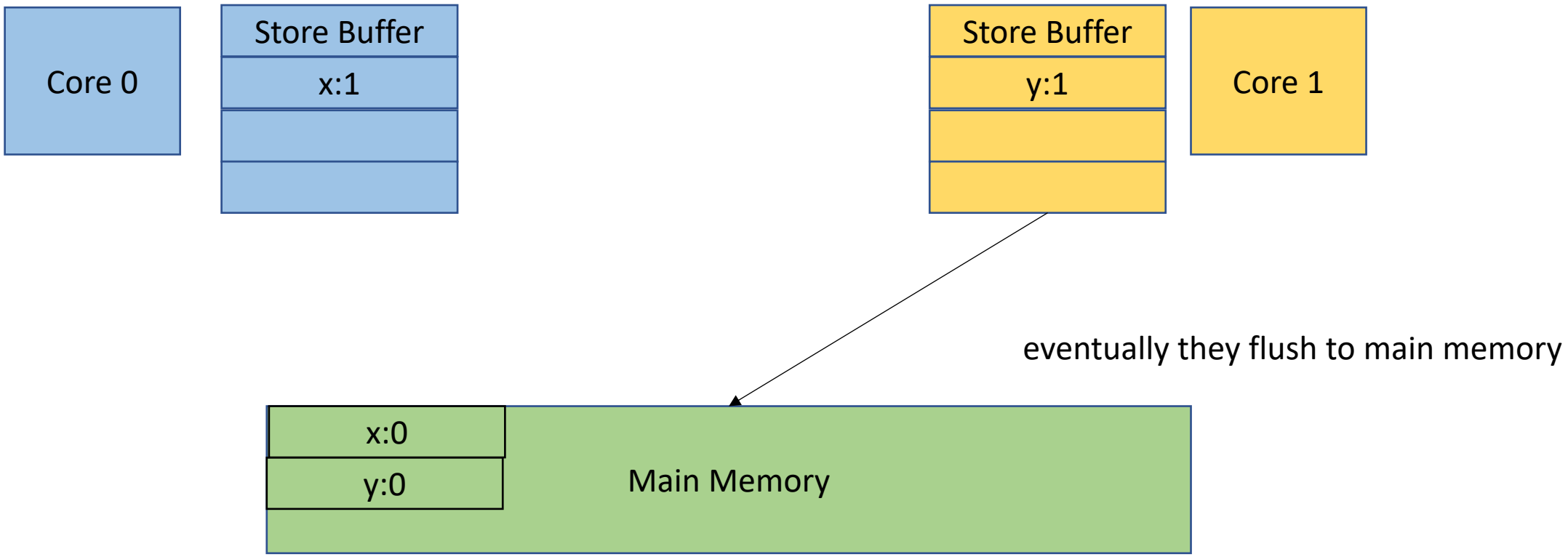
Thread 0:

Thread 1:

X86 cores contain a store buffer; holds stores before going to main memory

```
mov %t0, [y]
```

```
mov %t1, [x]
```



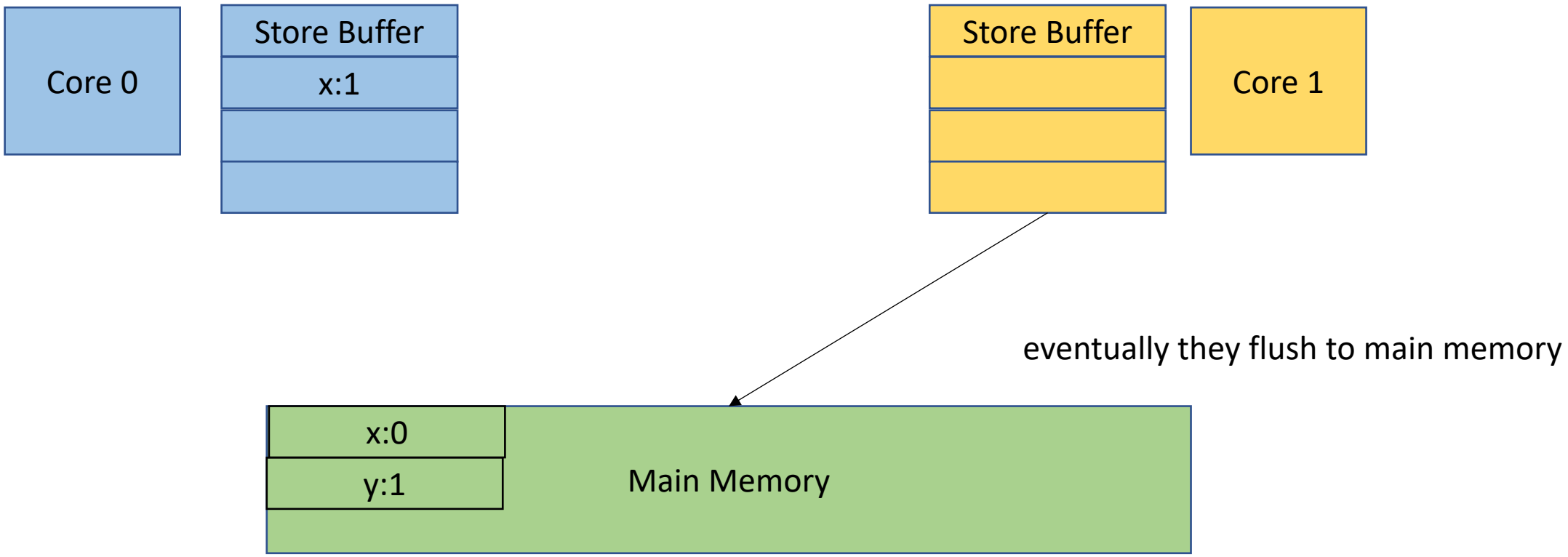
Thread 0:

Thread 1:

X86 cores contain a store buffer; holds stores before going to main memory

```
mov %t0, [y]
```

```
mov %t1, [x]
```



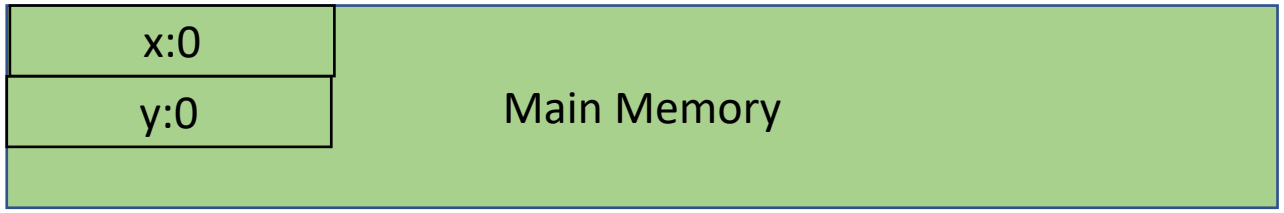
Thread 0:

```
mov [x], 1
mov %t0, [y]
```

rewind

Thread 1:

```
mov [y], 1
mov %t1, [x]
```



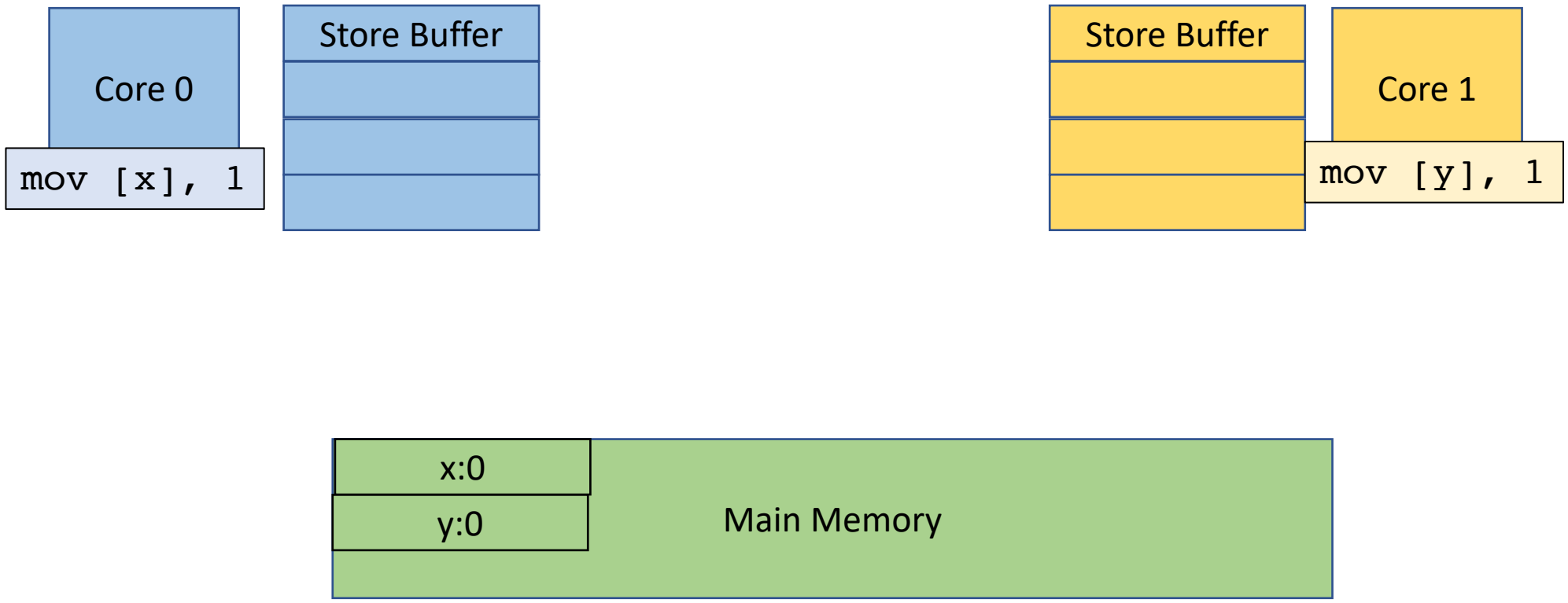
Thread 0:

Thread 1:

```
mov %t0, [y]
```

execute first instruction

```
mov %t1, [x]
```



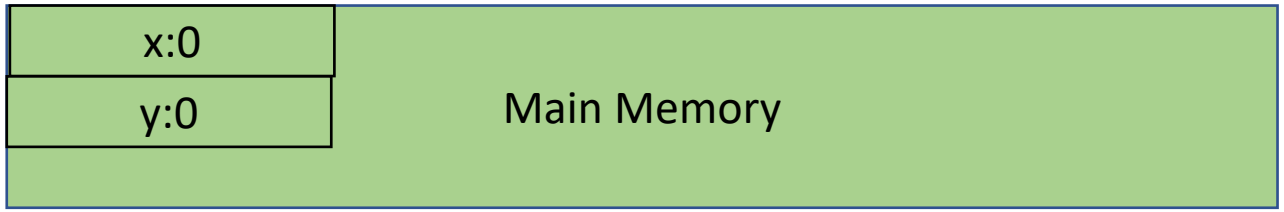
Thread 0:

Thread 1:

```
mov %t0, [y]
```

values get stored in SB

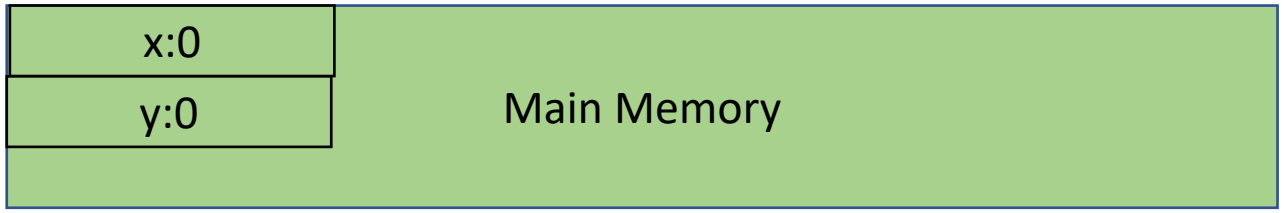
```
mov %t1, [x]
```



Thread 0:

Thread 1:

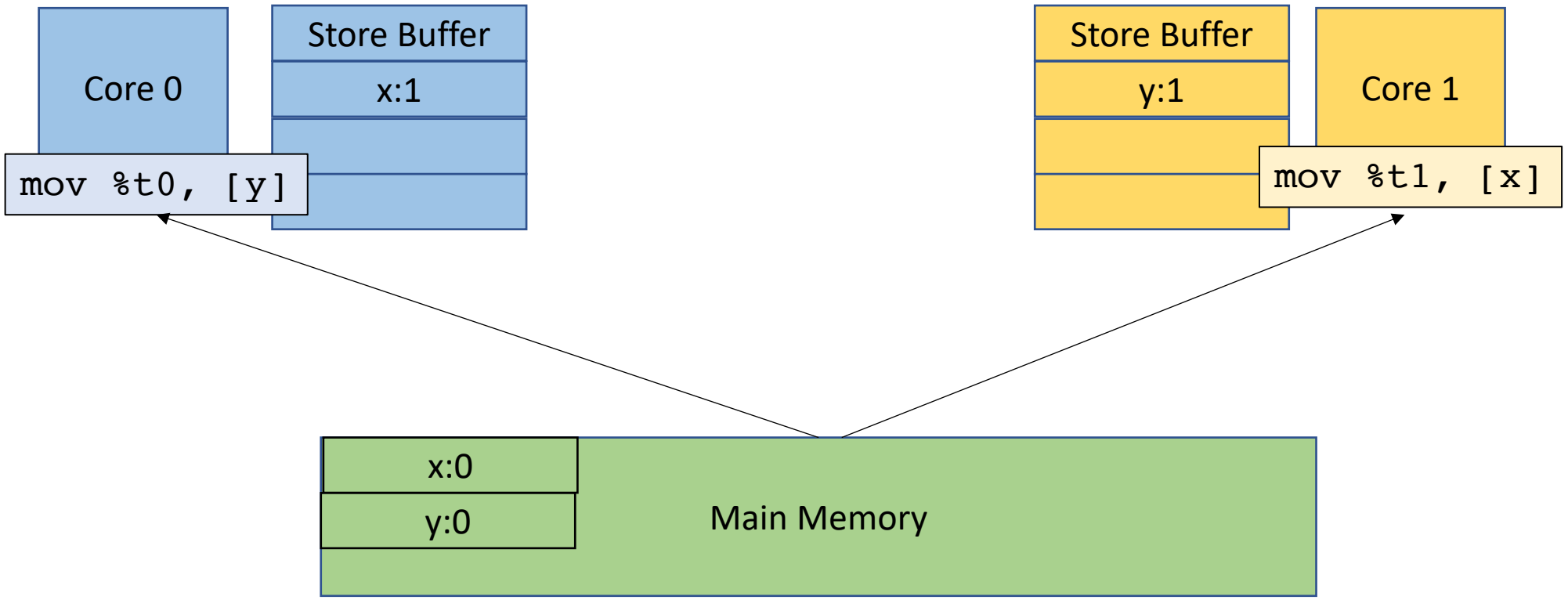
Execute next instruction



Thread 0:

Thread 1:

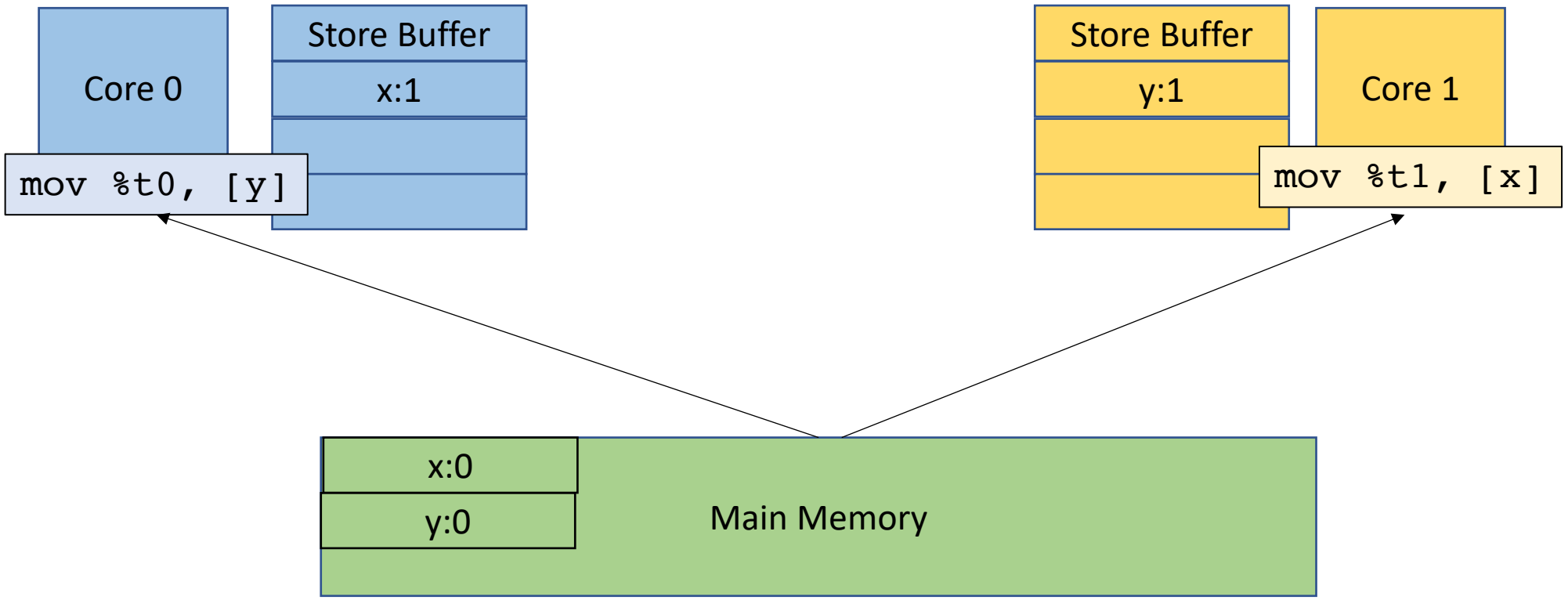
Values get loaded from memory



Thread 0:

Thread 1:

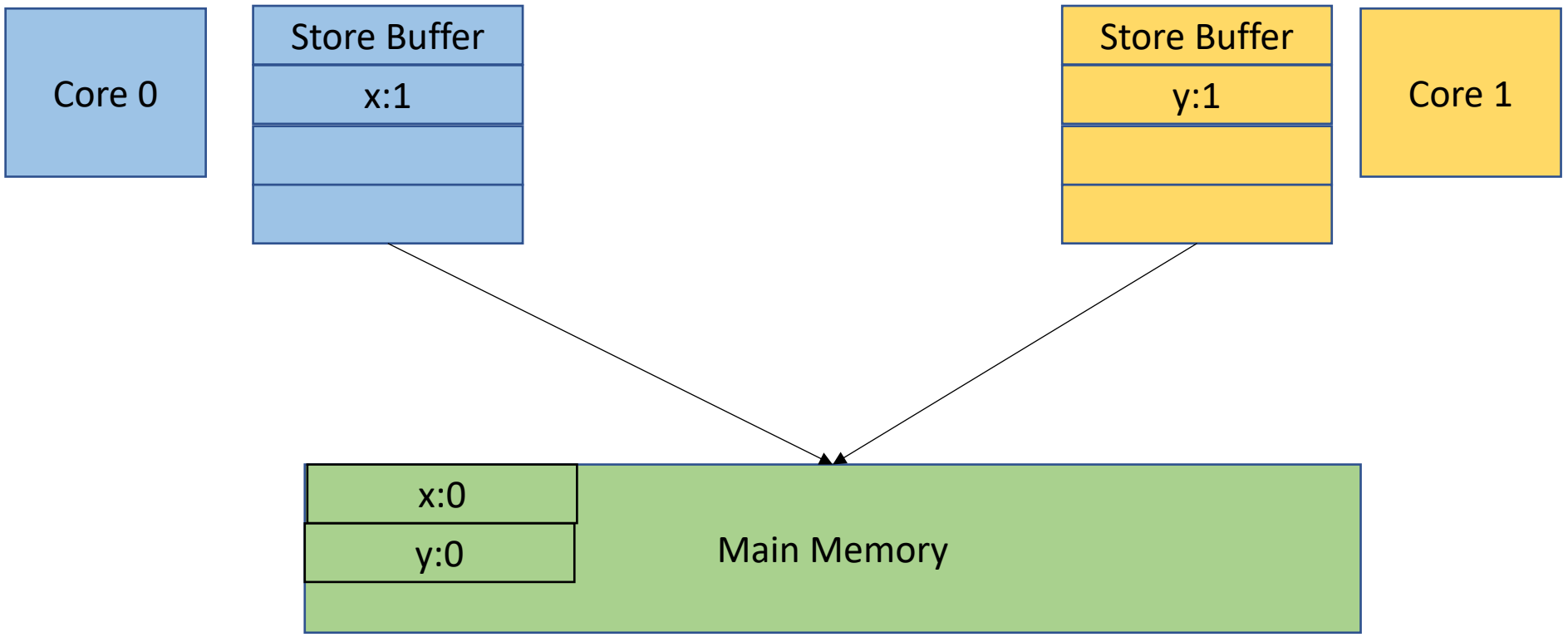
we see `t0 == t1 == 0`!



Thread 0:

Thread 1:

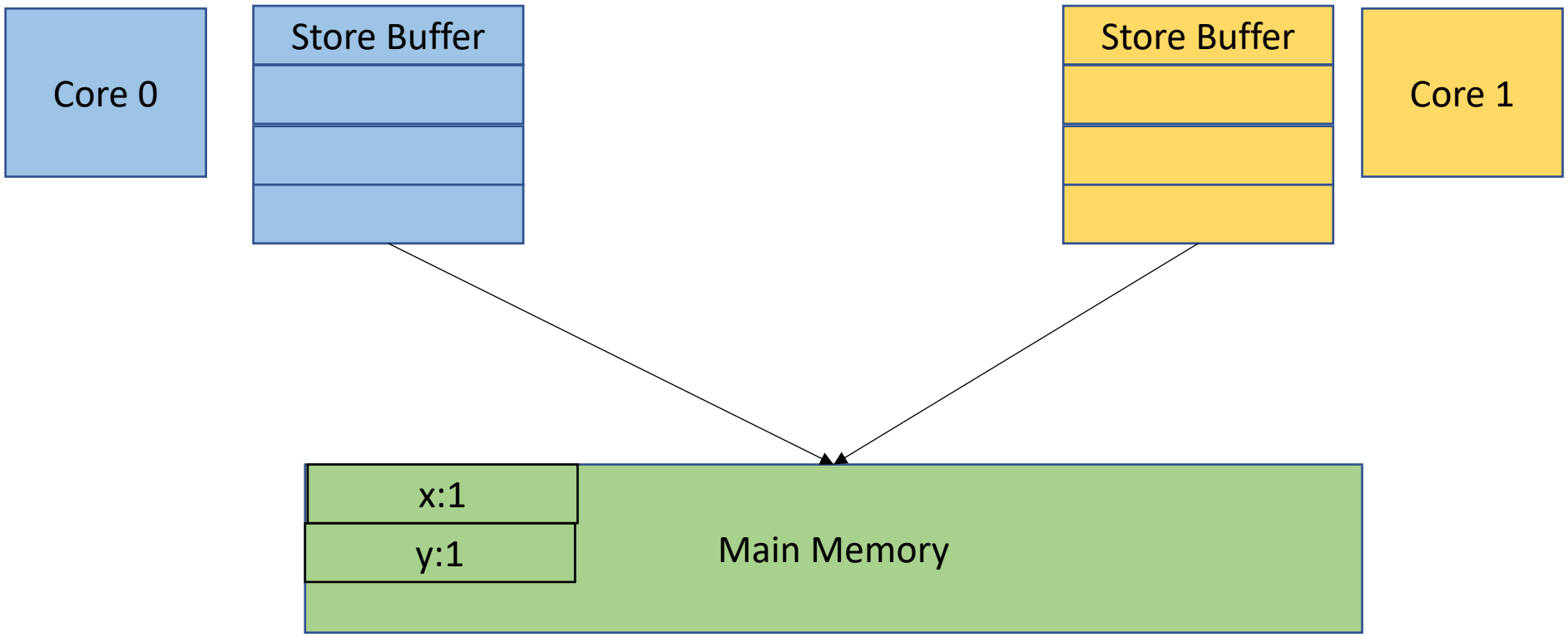
Store buffers are drained eventually



Thread 0:

Thread 1:

Store buffers are drained eventually
but we've already done our loads



Our first relaxed memory execution!

- also known as weak memory behaviors
- An execution that is NOT allowed by sequential consistency
- A memory model that allows relaxed memory executions is known as a relaxed memory model
  - X86 has a relaxed memory model due to store buffering
  - If you restrict yourself to use only default atomic operations, C++ has does NOT have a weak memory model

Litmus tests

- Small concurrent programs that check for relaxed memory behaviors
- Vendors have a long history of under documented memory consistency models
- Academics have empirically explored the memory models
  - Many vendors have unofficially endorsed academic models
  - X86 behaviors were documented by researchers before Intel!

Litmus tests

This test is called “store buffering”

Thread 0:

```
mov [x], 1
mov %t0, [y]
```

Thread 1:

```
mov [y], 1
mov %t1, [x]
```

Can `t0 == t1 == 0`?

Restoring sequential consistency

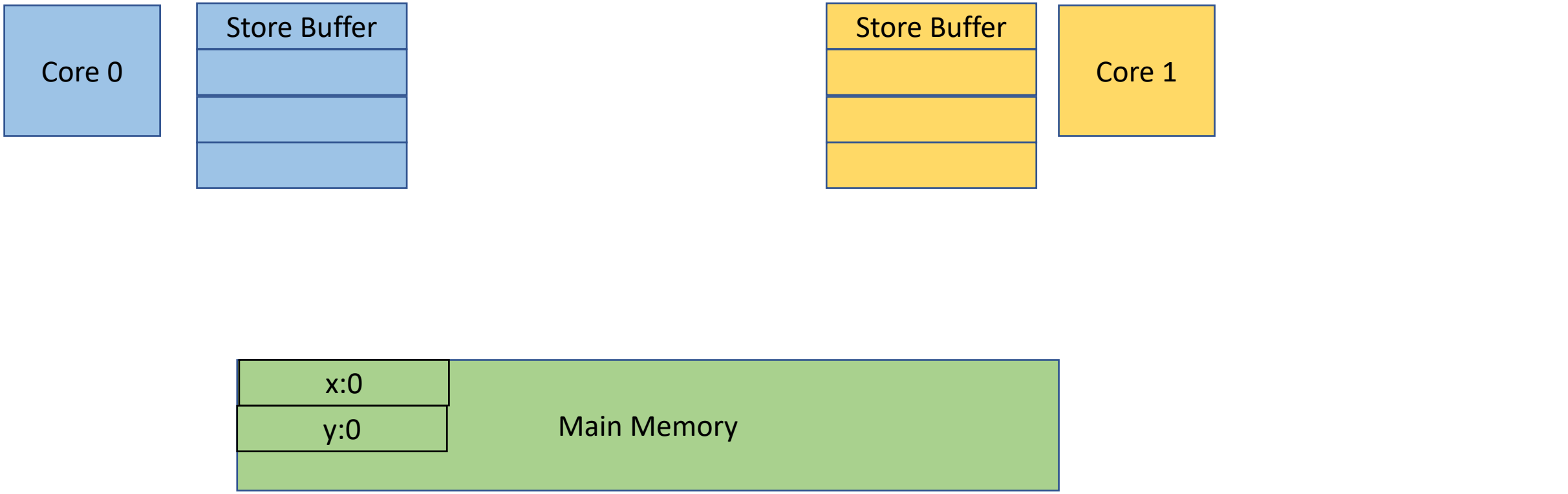
- It is typical that relaxed memory models provide special instructions which can be used to disallow weak behaviors.
- These instructions are called Fences
- The X86 fence is called `mfence`. It flushes the store buffer.

Thread 0:

```
mov [x], 1
mfence
mov %t0, [y]
```

Thread 1:

```
mov [y], 1
mfence
mov %t1, [x]
```



Thread 0:

Thread 1:

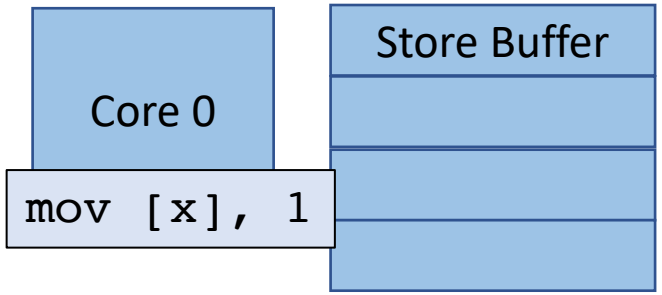
```
mfence
```

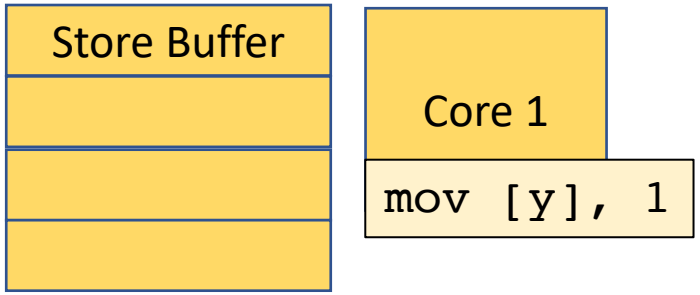
```
mov %t0, [y]
```

Execute first instruction

```
mfence
```

```
mov %t1, [x]
```





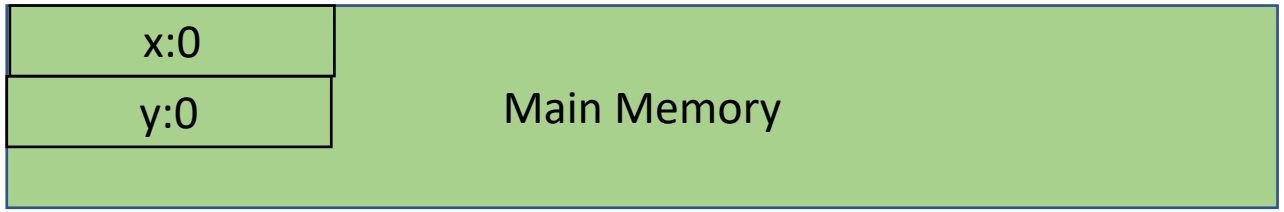
Thread 0:

Thread 1:

```
mfence
mov %t0, [y]
```

Values go into the store buffer

```
mfence
mov %t1, [x]
```



Thread 0:

Thread 1:

Execute next instruction

```
mov %t0, [y]
```

```
mov %t1, [x]
```

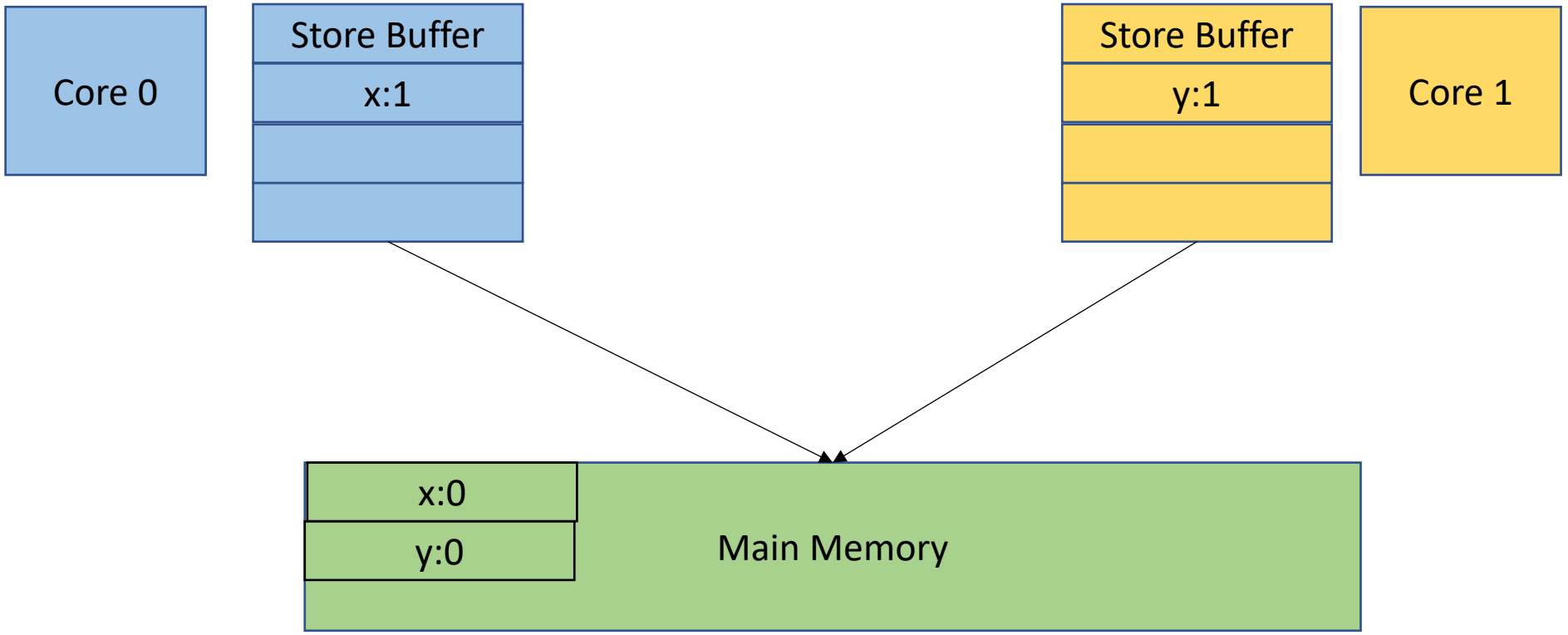
Thread 0:

Thread 1:

store buffers are flushed

```
mov %t0, [y]
```

```
mov %t1, [x]
```



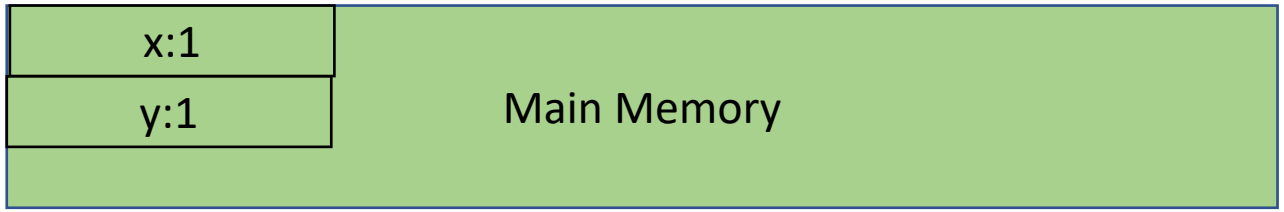
Thread 0:

Thread 1:

store buffers are flushed

```
mov %t0, [y]
```

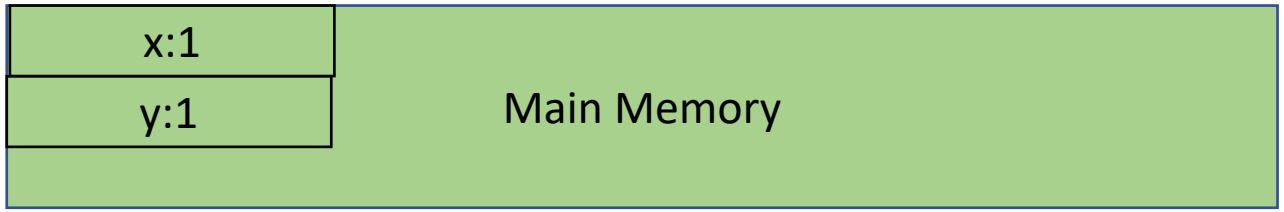
```
mov %t1, [x]
```



Thread 0:

Thread 1:

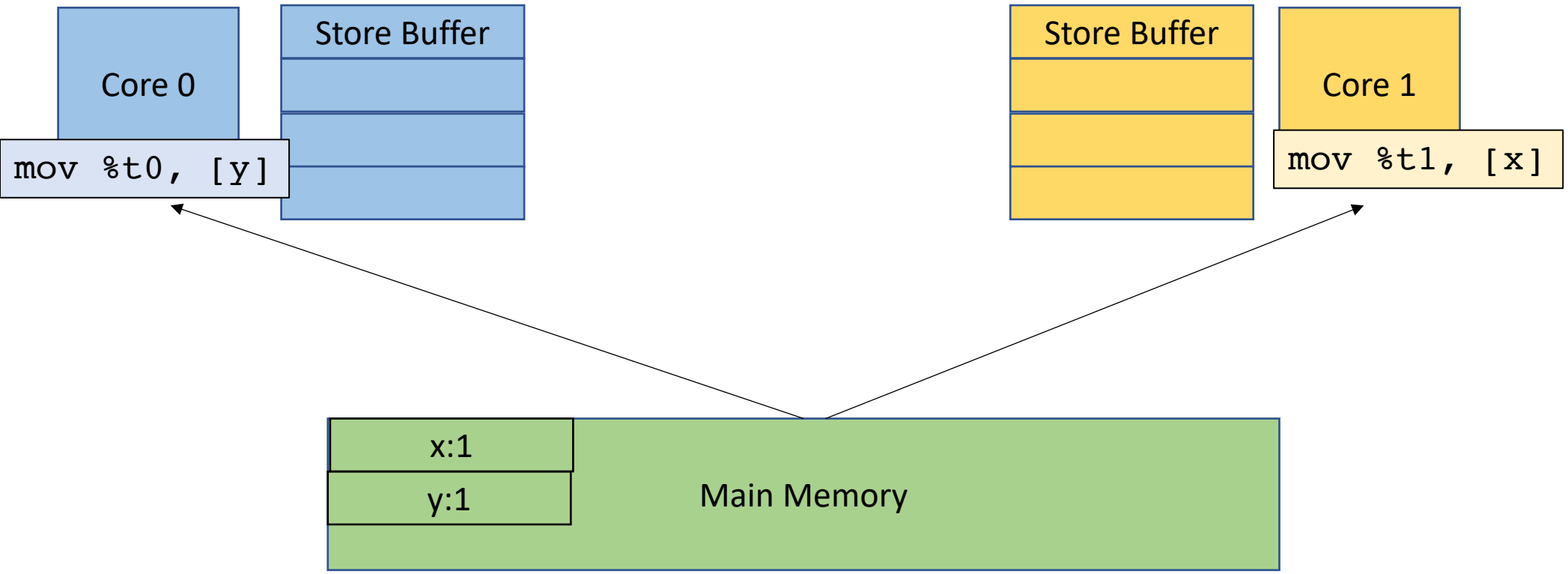
execute next instruction



Thread 0:

Thread 1:

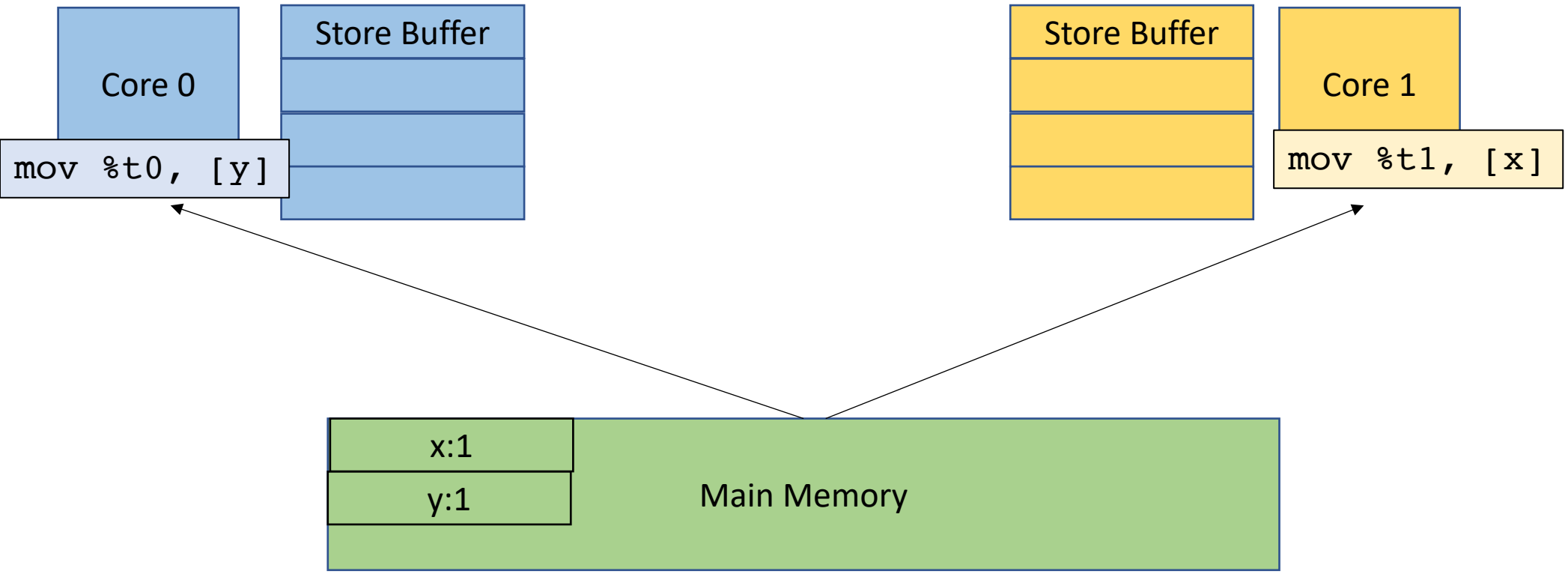
values are loaded from memory



Thread 0:

Thread 1:

We don't get the problematic behavior: `t0 == t1 == 0`



Next example

Thread 0:

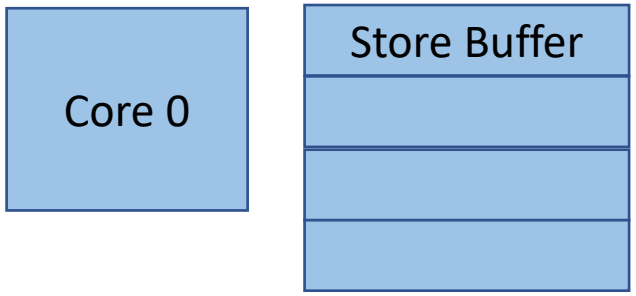
```
mov [x], 1
```

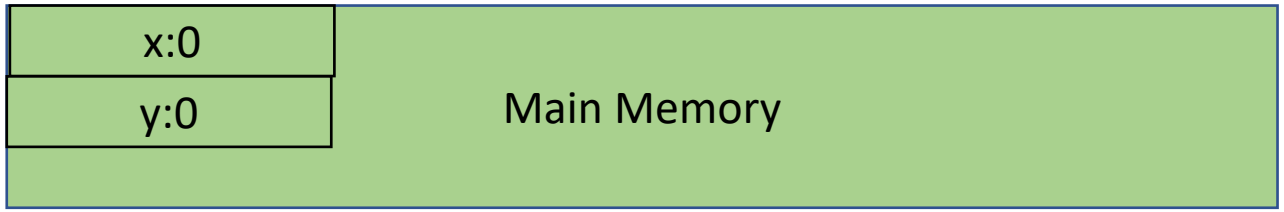
```
mov %t0, [x]
```

single thread
same address

possible outcomes:
t0 = 1
t0 = 0

Which one do you expect?



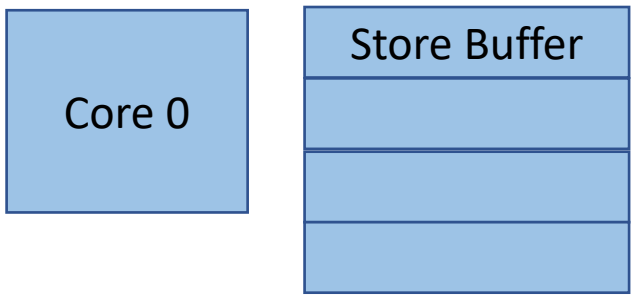


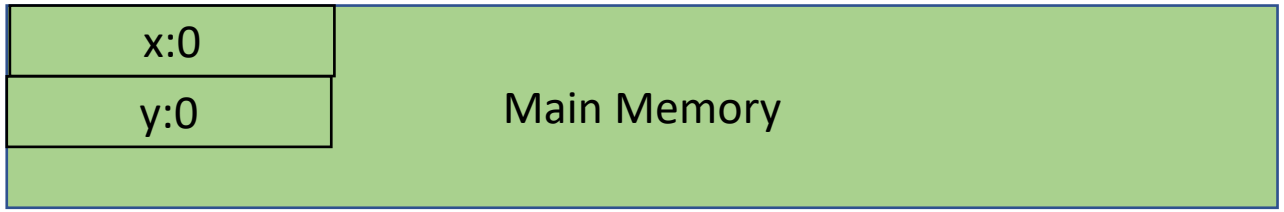
Thread 0:

```
mov [x], 1
```

```
mov %t0, [x]
```

How does this execute?

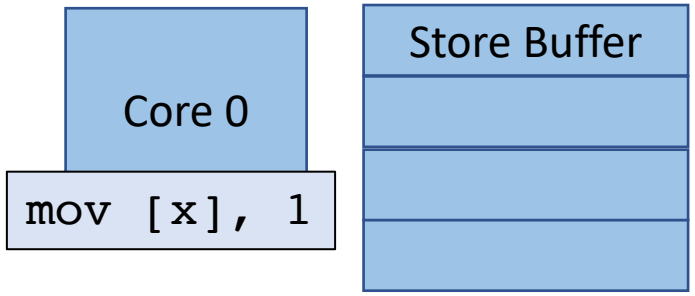


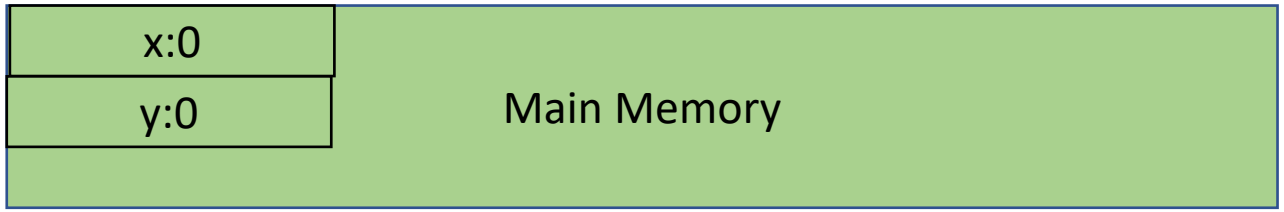


Thread 0:

execute first instruction

```
mov %t0, [x]
```

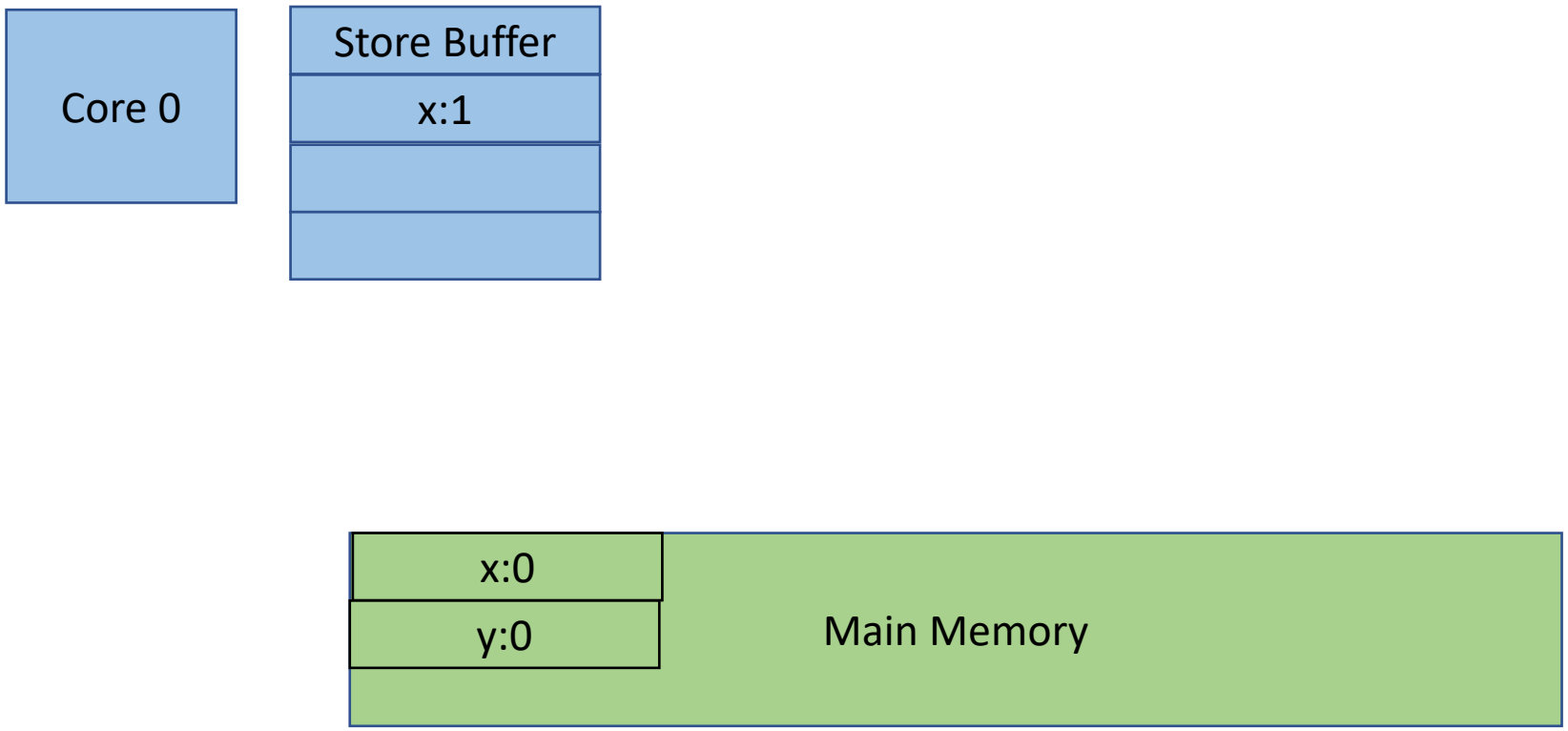




Thread 0:

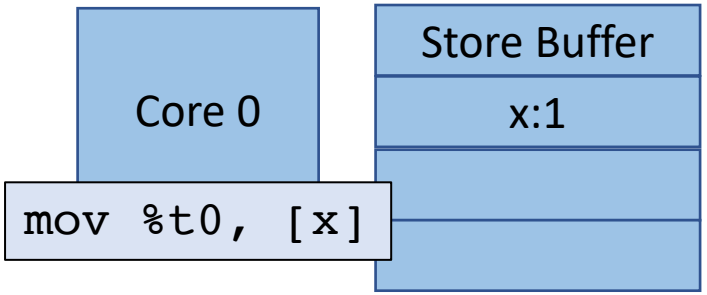
Store the value in the store buffer

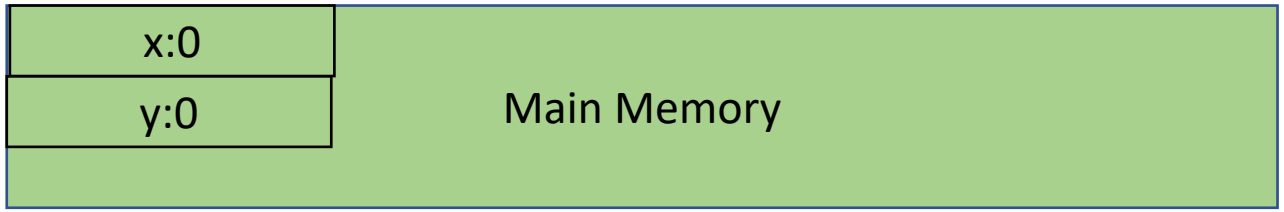
```
mov %t0, [x]
```



Thread 0:

Next instruction

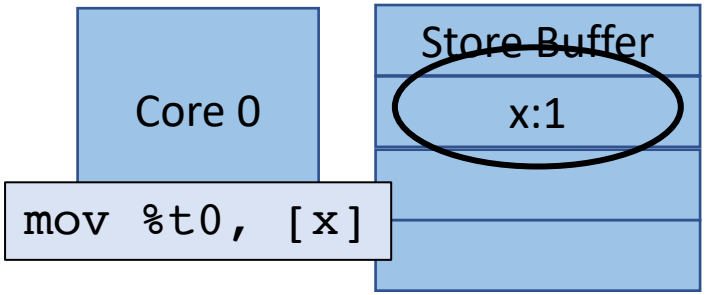


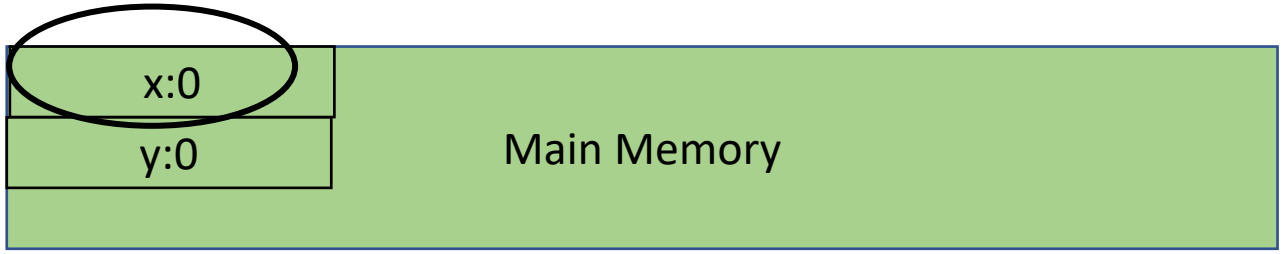


Thread 0:

Where to load??

Store buffer?
Main memory?



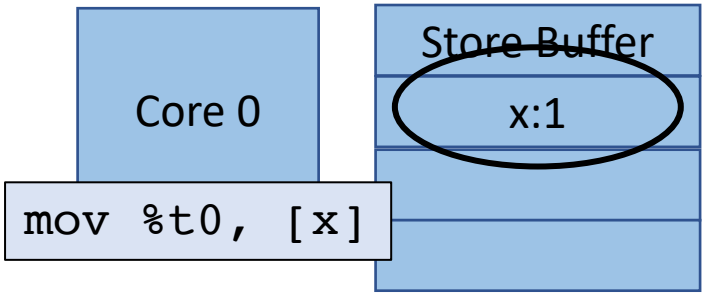


Thread 0:

Where to load??

Threads check store buffer before going to main memory

It is close and cheap to check.



Memory Consistency

- How to specify a relaxed memory model?

- We can do it operationally
  - by constructing a high-level machine and reasoning about operations through the machine.

  - or we can talk about instructions that are allowed to ”break” program order.

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Another test
Can `t0 == t1 == 0`?

Thread 0:

```
mov [x], 1
mov %t0, [y]
```

Thread 1:

```
mov [y], 1
mov %t1, [x]
```

We will annotate instructions with S for store, and L for loads

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Another test
Can `t0 == t1 == 0`?

Thread 0:

```
S:mov [x], 1
L:mov %t0, [y]
```

Thread 1:

```
S:mov [y], 1
L:mov %t1, [x]
```

We will annotate instructions with S for store, and L for loads

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

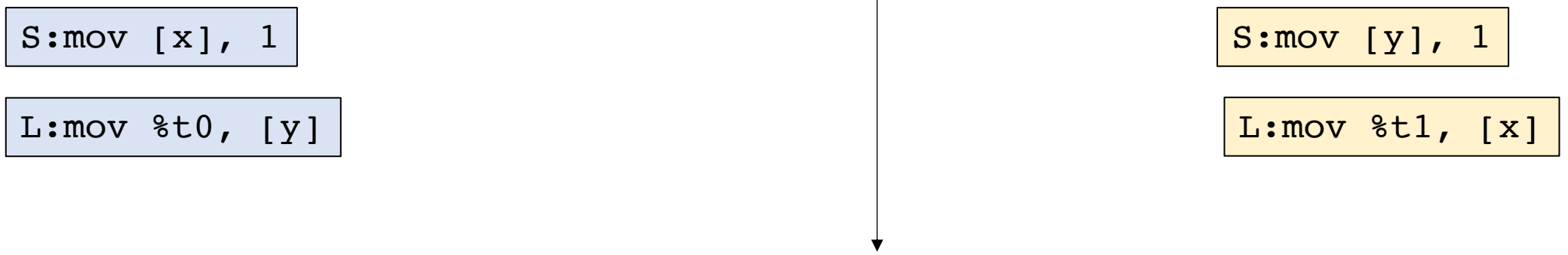
Another test
Can `t0 == t1 == 0`?

Thread 0:

```
S:mov [x], 1
L:mov %t0, [y]
```

Thread 1:

```
S:mov [y], 1
L:mov %t1, [x]
```



Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

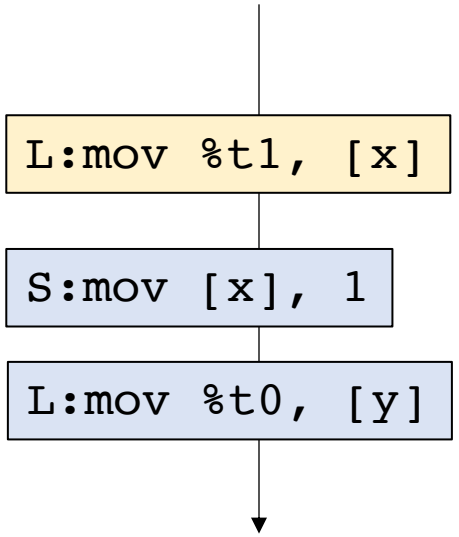
Another test
Can `t0 == t1 == 0`?

Thread 0:

```
S:mov [x], 1
L:mov %t0, [y]
```

Thread 1:

```
S:mov [y], 1
L:mov %t1, [x]
```




```
S:mov [y], 1
```

Now we make a new rule:

S(tores) followed by a L(oad)
do not have to follow program order

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Another test
Can `t0 == t1 == 0`?

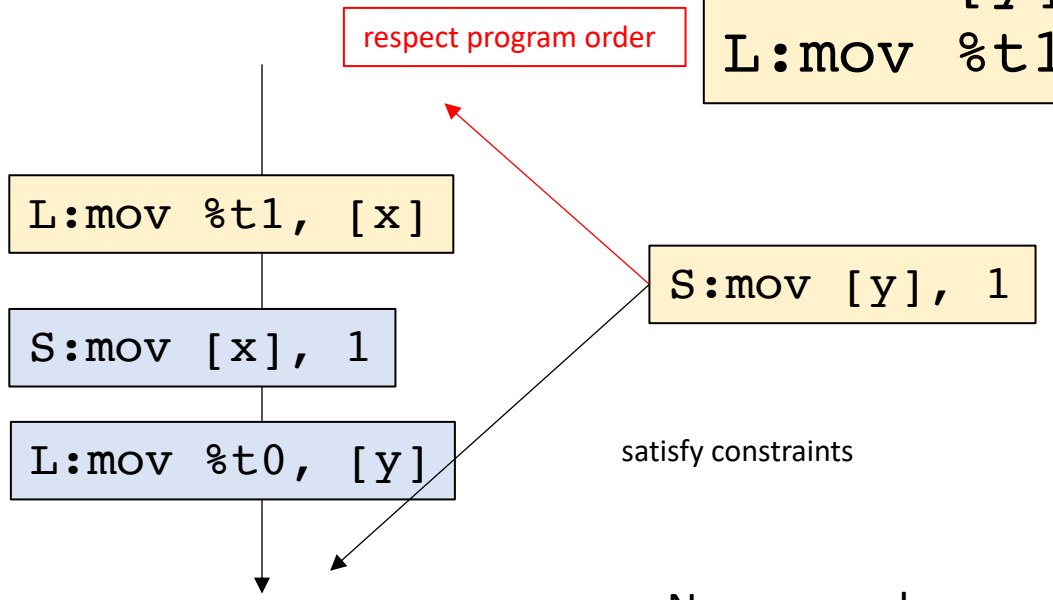
Thread 0:

```
S:mov [x], 1
L:mov %t0, [y]
```

we can ignore this condition!!

Thread 1:

```
S:mov [y], 1
L:mov %t1, [x]
```



Now we make a new rule:

S(tores) followed by a L(oad)
do not have to follow program order

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Another test
Can `t0 == t1 == 0`?

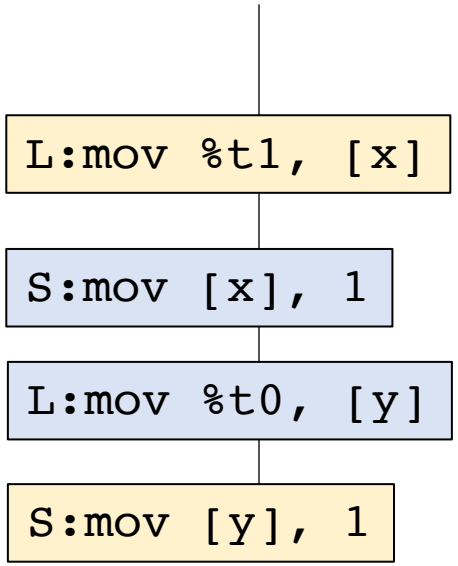
Thread 0:

```
S:mov [x], 1
L:mov %t0, [y]
```

we can ignore this condition!!

Thread 1:

```
S:mov [y], 1
L:mov %t1, [x]
```



Now we can satisfy the condition!

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Another test
Can `t0 == t1 == 0`?

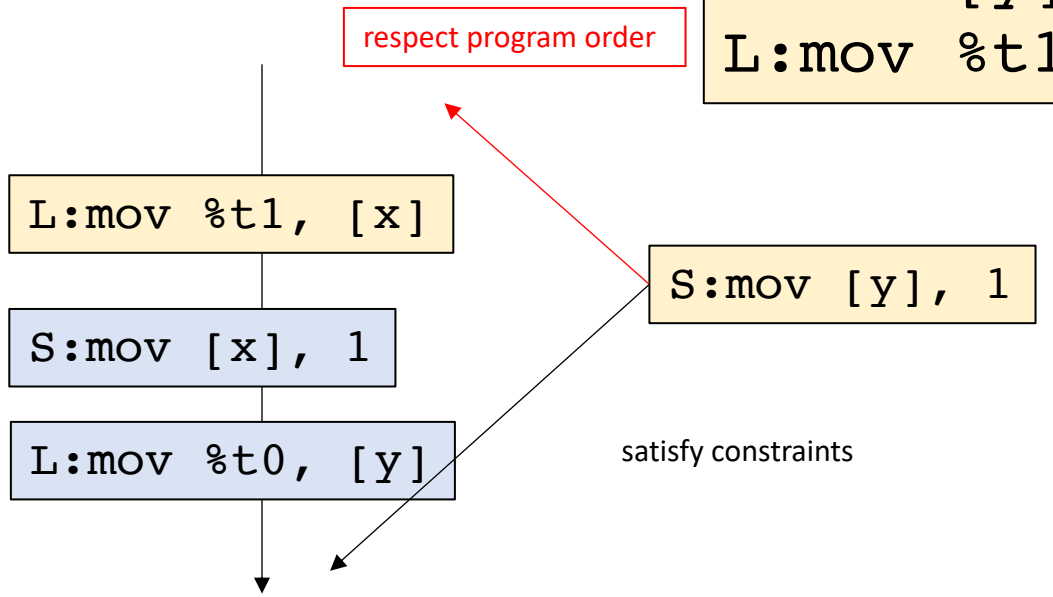
Thread 0:

```
S:mov [x], 1
L:mov %t0, [y]
```

Thread 1:

```
S:mov [y], 1
L:mov %t1, [x]
```

we can ignore this condition!!



Lets peak under the hood here

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Another test
Can `t0 == t1 == 0`?

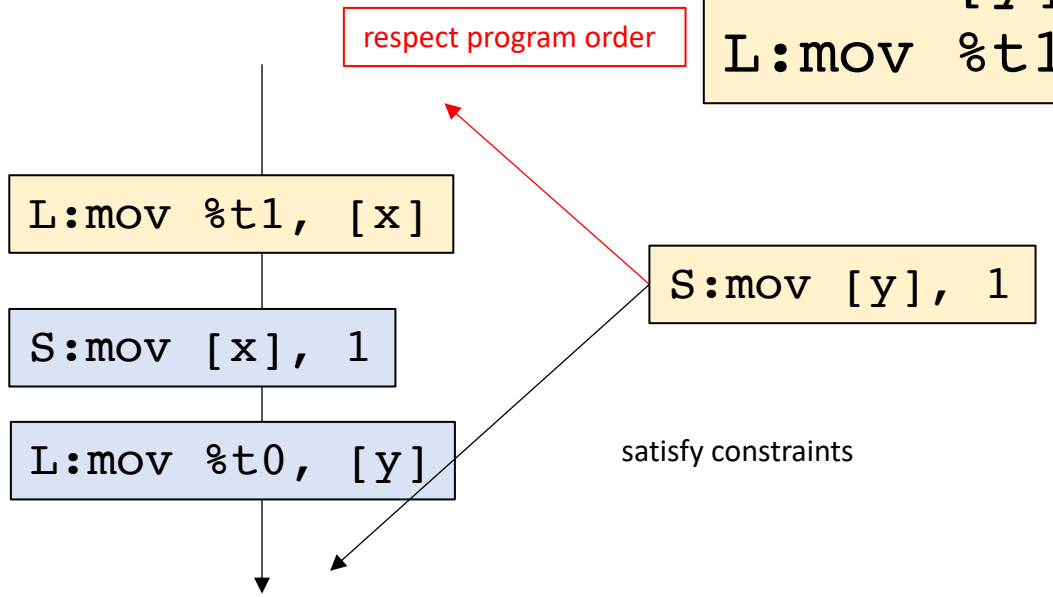
Thread 0:

```
S:mov [x], 1
L:mov %t0, [y]
```

Thread 1:

```
S:mov [y], 1
L:mov %t1, [x]
```

we can ignore this condition!!



Lets peak under the hood here

Global timeline is when the Store operation becomes visible to other threads

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Another test
Can `t0 == t1 == 0`?

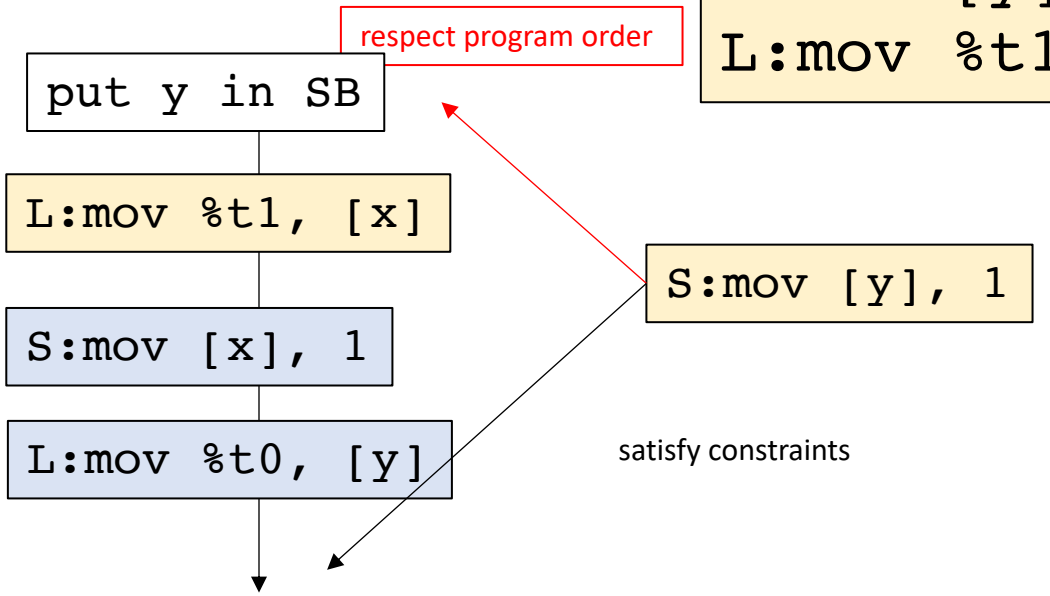
Thread 0:

```
S:mov [x], 1
L:mov %t0, [y]
```

Thread 1:

```
S:mov [y], 1
L:mov %t1, [x]
```

we can ignore this condition!!



Lets peak under the hood here

Global timeline is when the Store operation becomes visible to other threads

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Another test
Can `t0 == t1 == 0`?

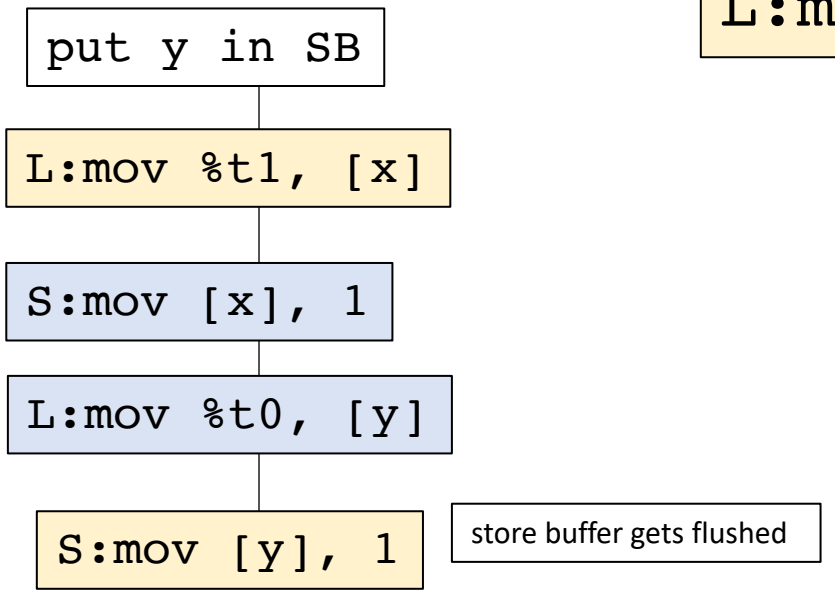
Thread 0:

```
S:mov [x], 1
L:mov %t0, [y]
```

we can ignore this condition!!

Thread 1:

```
S:mov [y], 1
L:mov %t1, [x]
```



Lets peak under the hood here

Global timeline is when the Store operation becomes visible to other threads

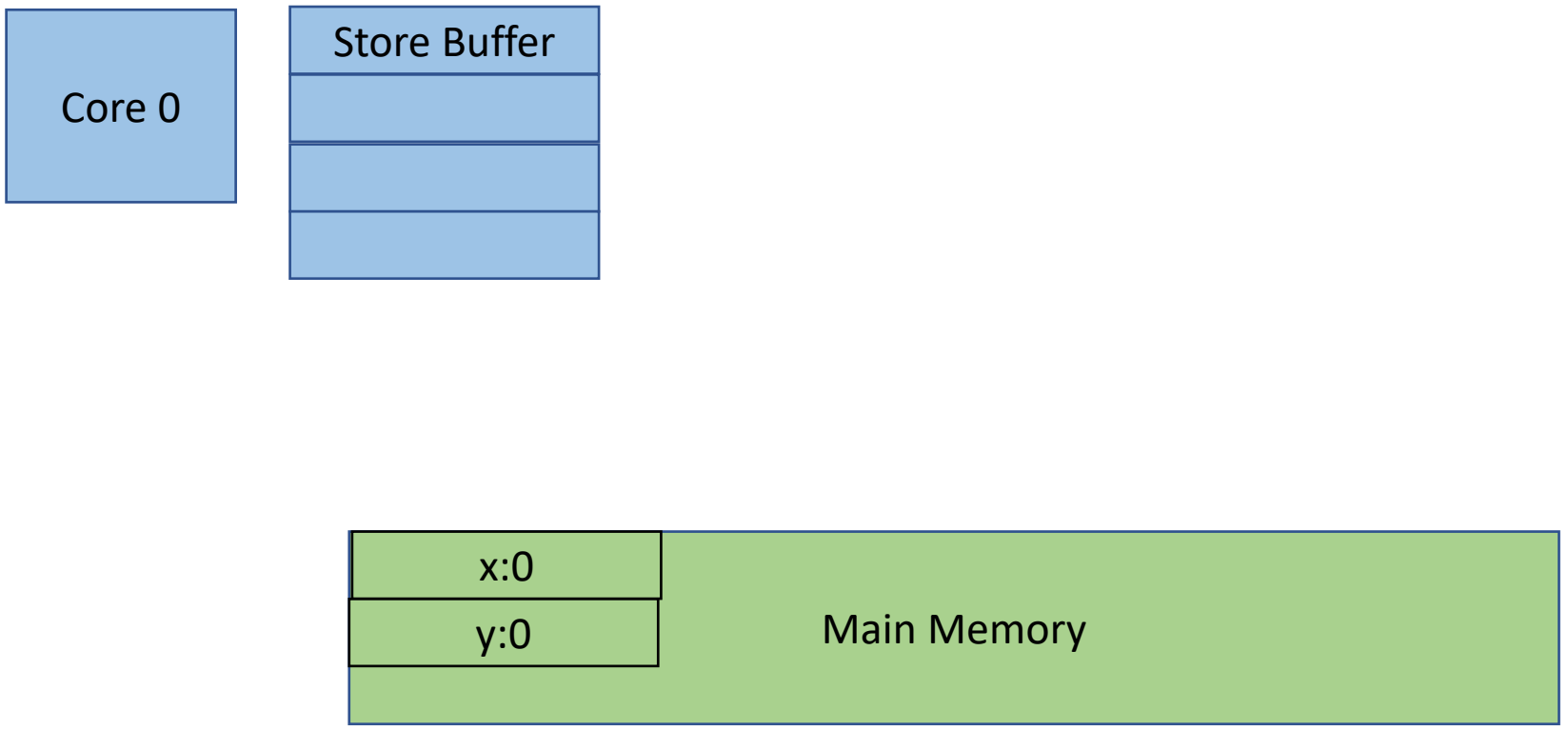
Questions

- Can stores be reordered with stores?

Thread 0:

```
mov [x], 1
```

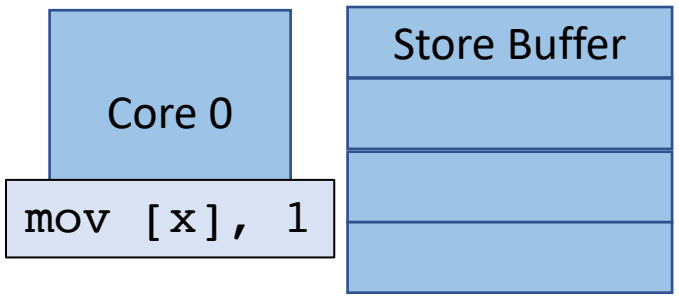
```
mov [y], 1
```

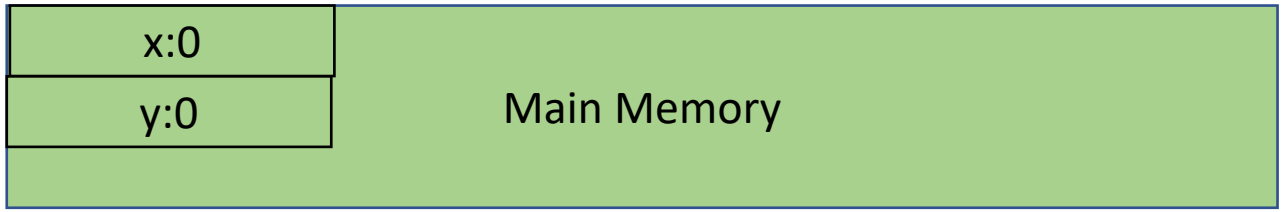


Thread 0:

```
mov [y], 1
```

execute the first instruction

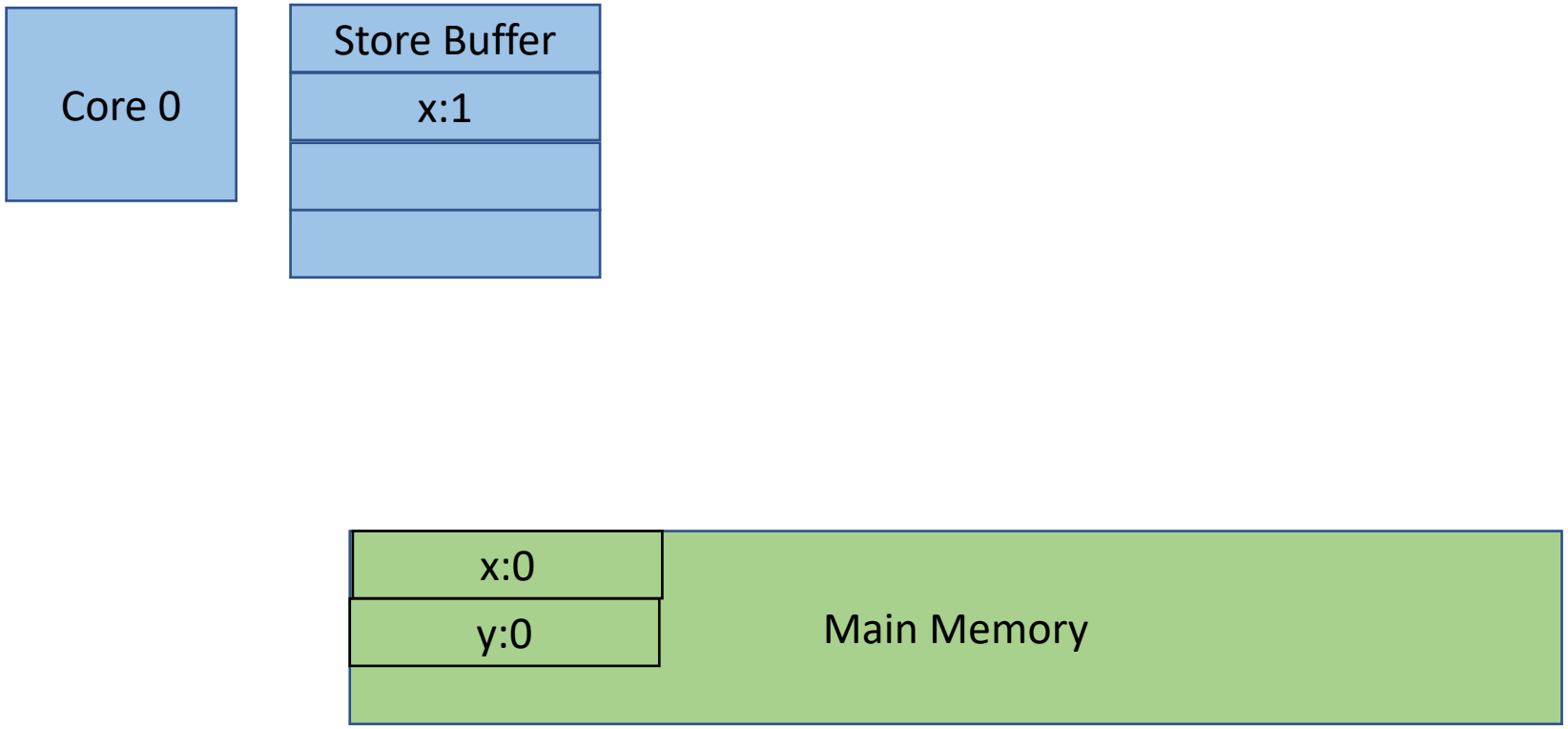




Thread 0:

```
mov [y], 1
```

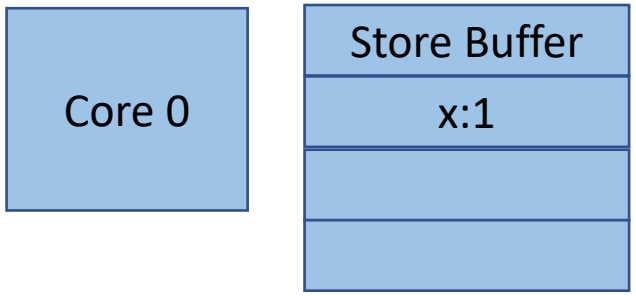
value goes into store buffer

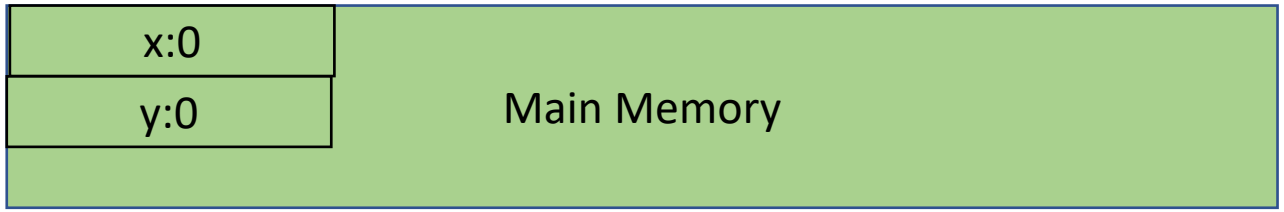


Thread 0:

```
mov [y], 1
```

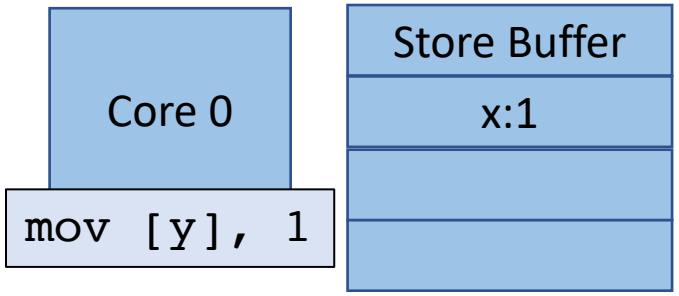
execute next instruction

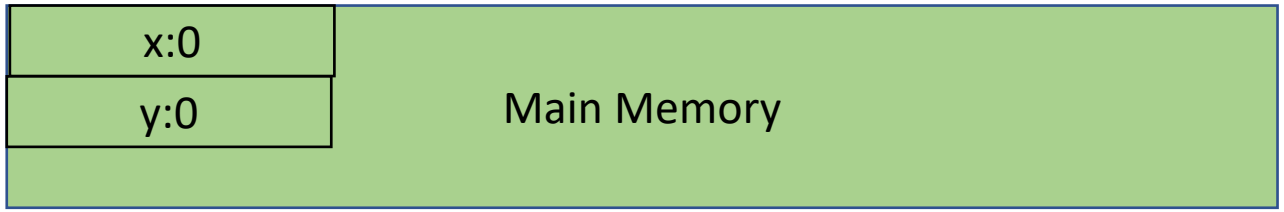




Thread 0:

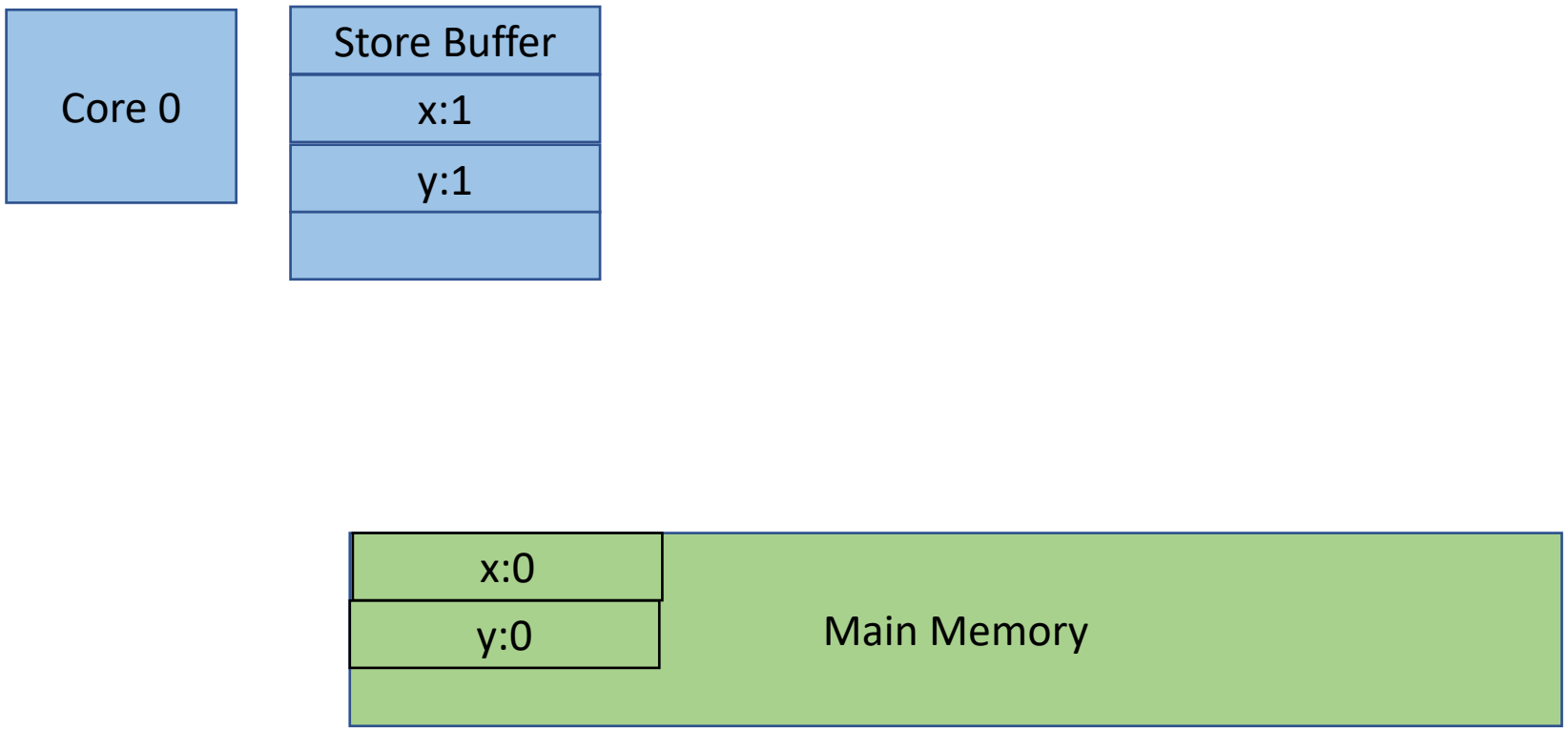
execute next instruction



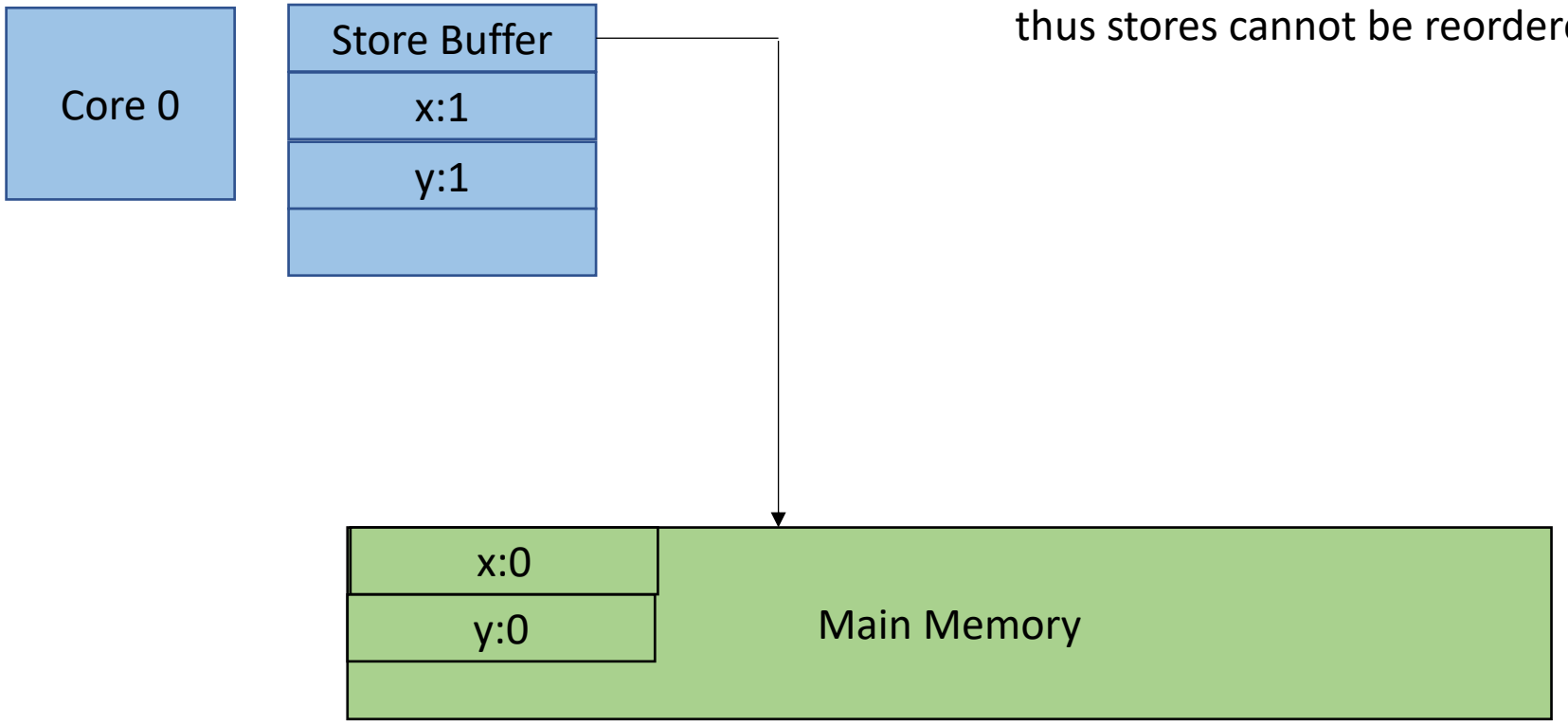


Thread 0:

value goes into the store buffer

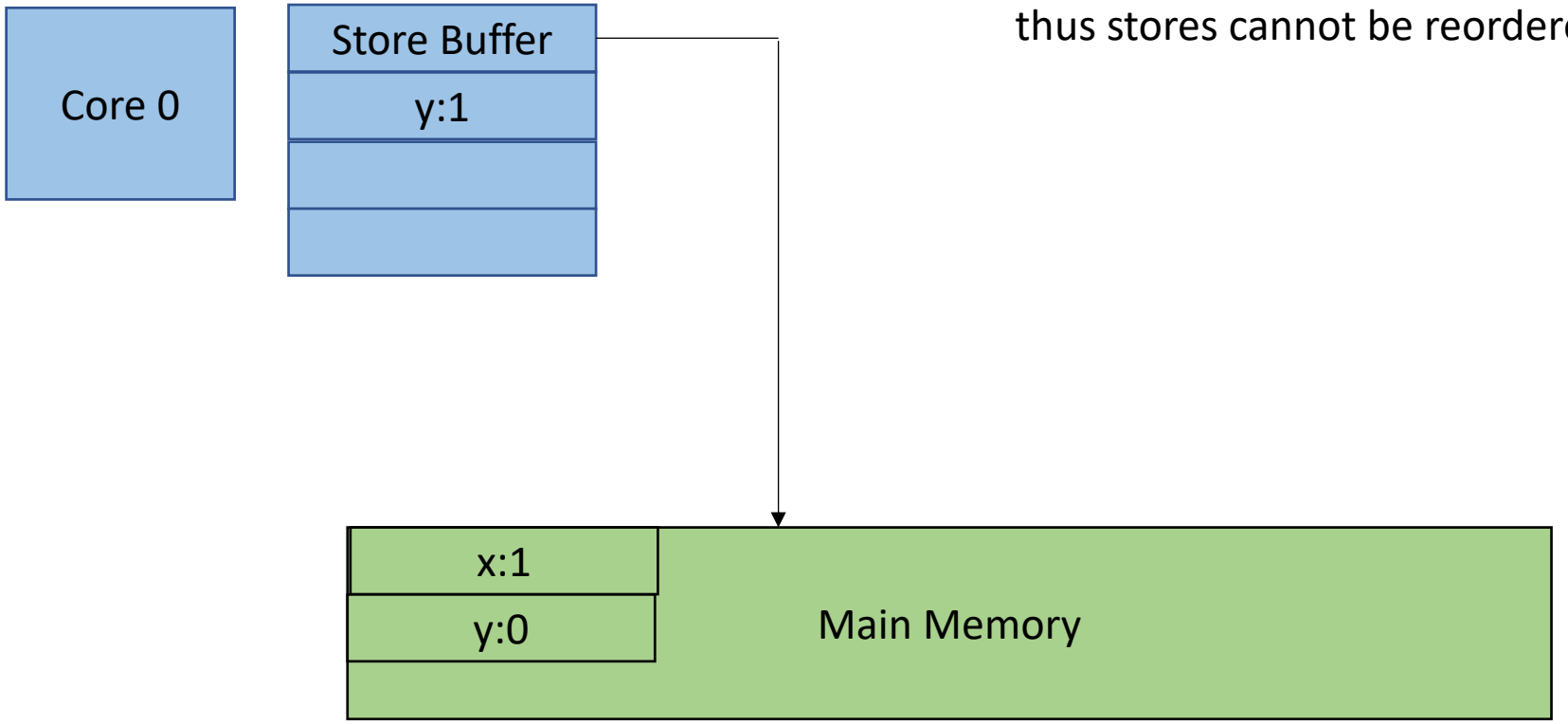


Thread 0:



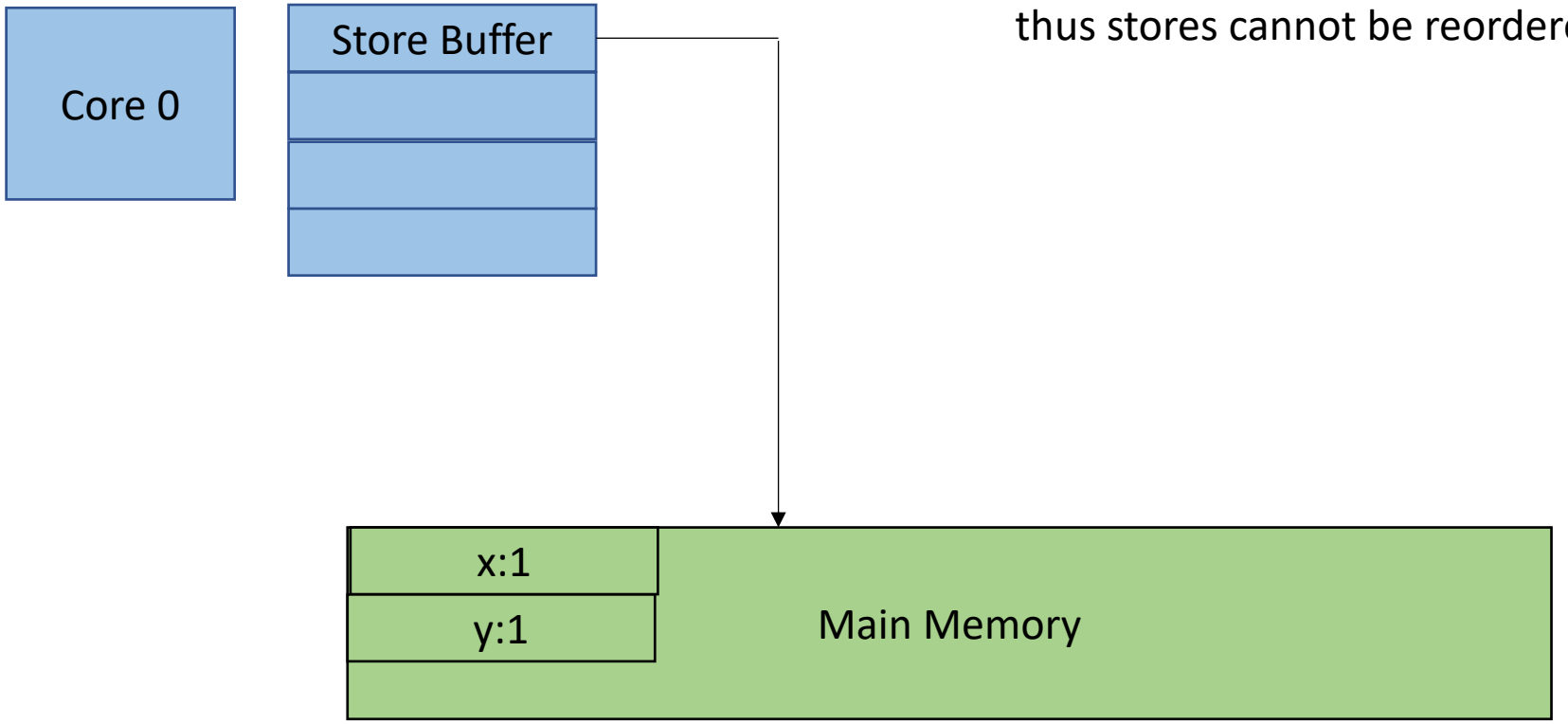
On x86, the store buffer trains in a FIFO way:
thus stores cannot be reordered

Thread 0:



On x86, the store buffer trains in a FIFO way:
thus stores cannot be reordered

Thread 0:



On x86, the store buffer trains in a FIFO way:
thus stores cannot be reordered

Questions

- Can stores be reordered with stores?
- How do we make rules about mfence?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

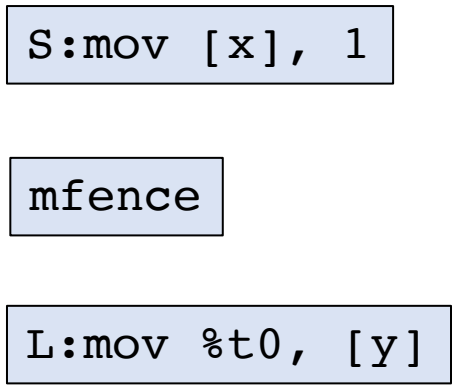
Another test
Can `t0 == t1 == 0`?

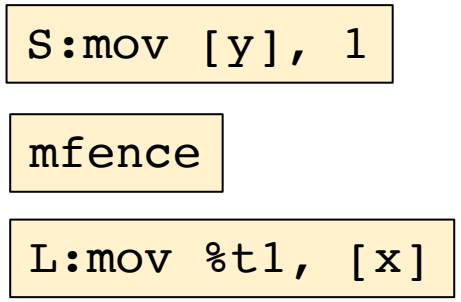
Thread 0:

```
S:mov [x], 1
mfence
L:mov %t0, [y]
```

Thread 1:

```
S:mov [y], 1
mfence
L:mov %t1, [x]
```





Rules: S(tores) followed by a L(oad) do not have to follow program order.

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

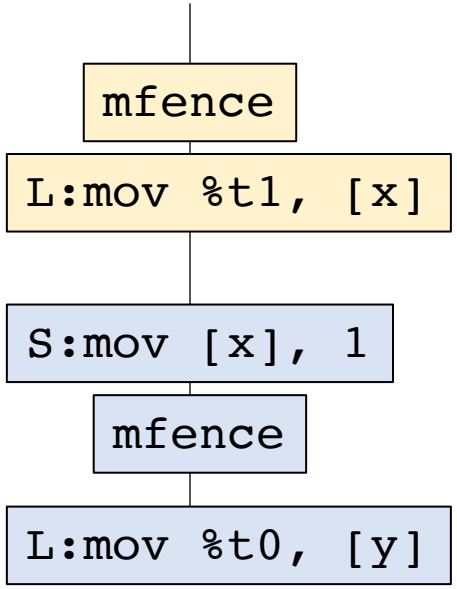
Thread 0:

```
S:mov [x], 1
mfence
L:mov %t0, [y]
```

Another test
Can `t0 == t1 == 0`?

Thread 1:

```
S:mov [y], 1
mfence
L:mov %t1, [x]
```




```
S:mov [y], 1
```

So we can't reorder this instruction at all!

Rules:
S(tores) followed by a L(oad)
do not have to follow program order.

S(tores) cannot be reordered past a fence in program order

Rules

- Are we done?

Rules:
S(tores) followed by a L(oad)
do not have to follow program order.

S(tores) cannot be reordered past a fence
in program order

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Thread 0:

```
S:mov [x], 1
L:mov %t0, [x]
```

```
S:mov [x], 1
```

```
L:mov %t0, [x]
```

Another test
Can `t0 == 0`?

Rules:
S(tores) followed by a L(oad)
do not have to follow program order.

S(tores) cannot be reordered past a fence
in program order

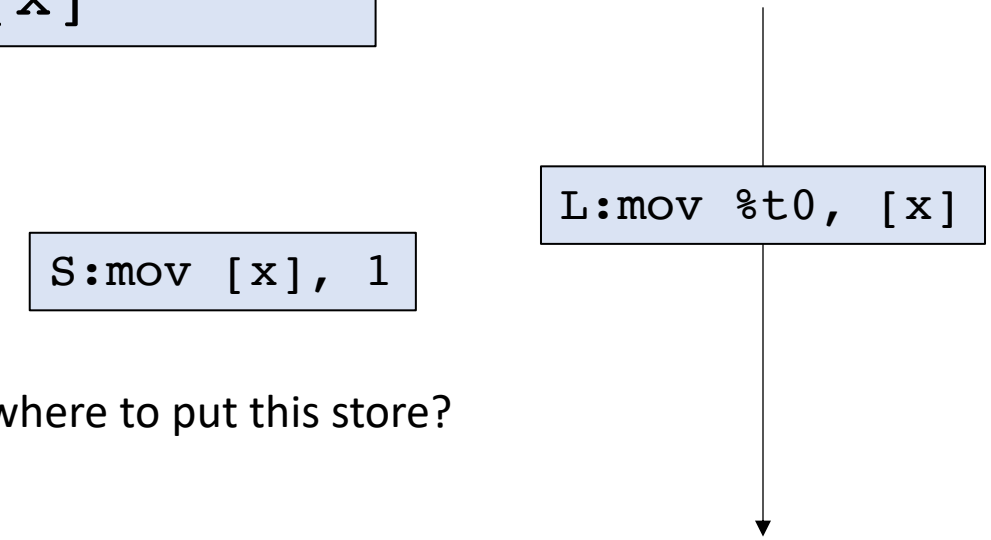
Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Another test
Can `t0 == 0`?

Thread 0:

```
S:mov [x], 1
L:mov %t0, [x]
```



where to put this store?

Rules:
S(tores) followed by a L(oad)
do not have to follow program order.

S(tores) cannot be reordered past a fence
in program order

Global variable:

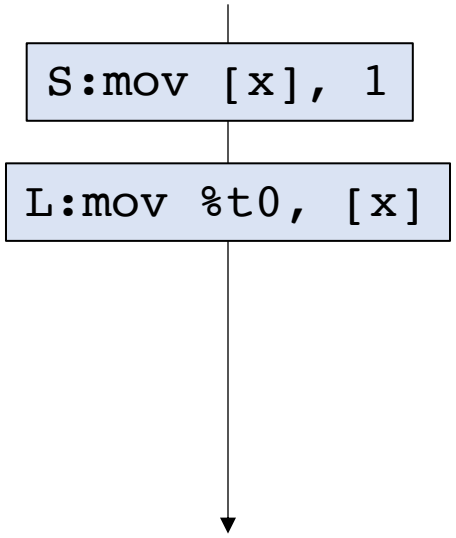
```
int x[1] = {0};
int y[1] = {0};
```

Another test
Can `t0 == 0`?

Thread 0:

```
S:mov [x], 1
L:mov %t0, [x]
```

where to put this store?



Rules:
S(tores) followed by a L(oad)
do not have to follow program order.

S(tores) cannot be reordered past a fence in program order

S(tores) cannot be reordered past L(oads) from the same address

TSO - Total Store Order

Rules:
S(tores) followed by a L(oad)
do not have to follow program order.

S(tores) cannot be reordered past a fence
in program order

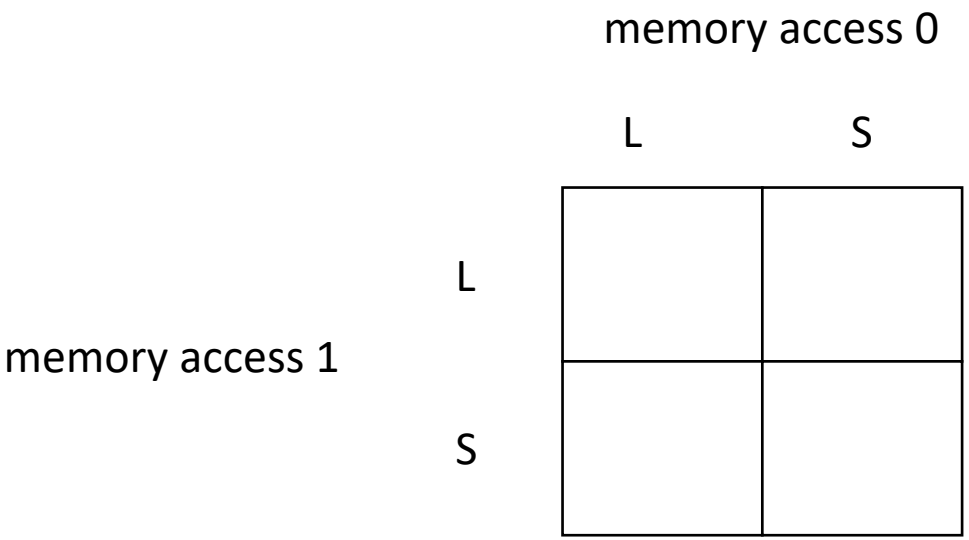
S(tores) cannot be reordered past L(oads)
from the same address

Schedule

- Memory consistency models:
  - Total store order
  - **Relaxed memory consistency**
  - Examples

Other memory models?

- We can specify them in terms of what reorderings are allowed



If memory access 0 appears before memory access 1 in program order, can it bypass program order?

Other memory models?

- We can specify them in terms of what reorderings are allowed

memory access 1 \ memory access 0	L	S
L	NO	NO
S	NO	NO

Sequential Consistency

If memory access 0 appears before memory access 1 in program order, can it bypass program order?

Other memory models?

- We can specify them in terms of what reorderings are allowed

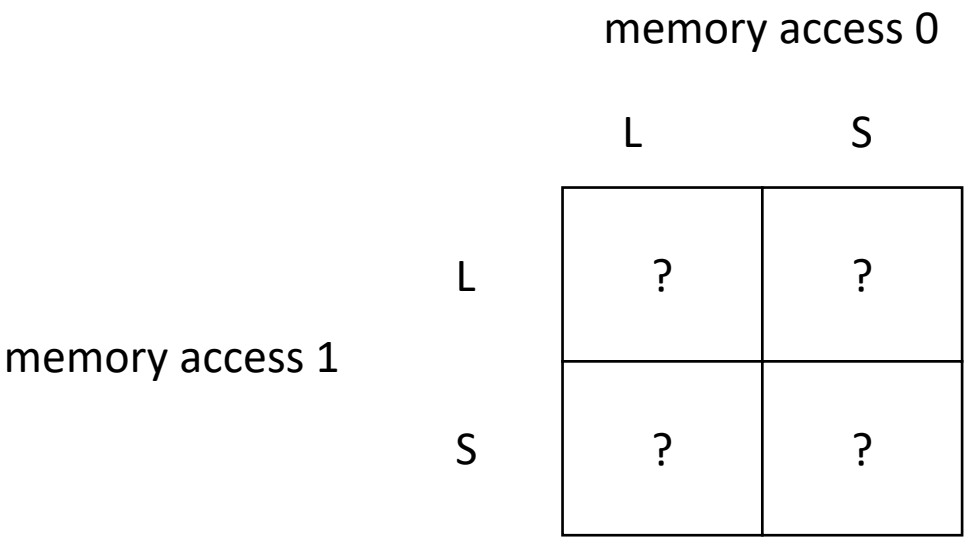
memory access 1 \ memory access 0	L	S
L	NO	Different address
S	NO	NO

TSO - total store order

If memory access 0 appears before memory access 1 in program order, can it bypass program order?

Other memory models?

- We can specify them in terms of what reorderings are allowed



Weaker models?

If memory access 0 appears before memory access 1 in program order, can it bypass program order?

Other memory models?

- We can specify them in terms of what reorderings are allowed

memory access 1 \ memory access 0	L	S
L	NO	Different address
S	NO	Different address

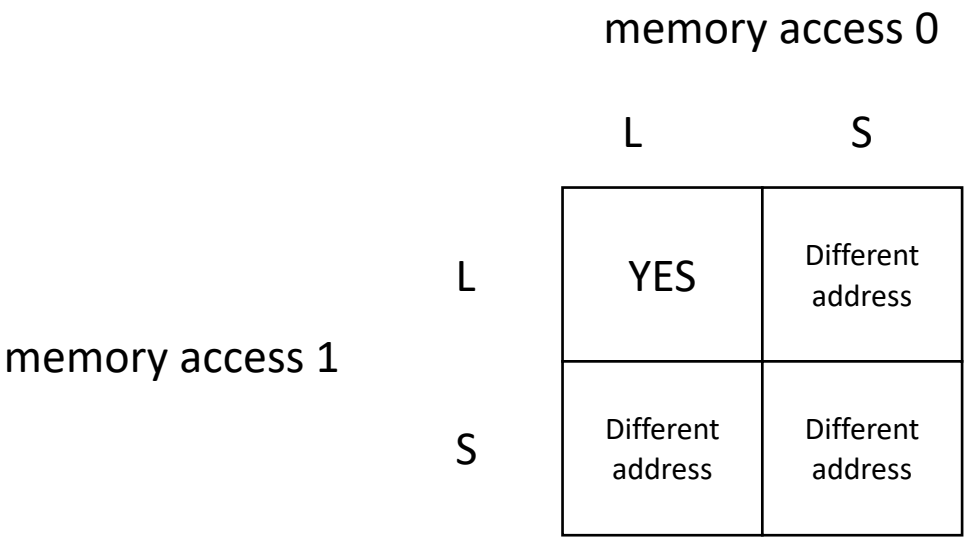
PSO - partial store order

If memory access 0 appears before memory access 1 in program order, can it bypass program order?

Allows stores to drain from the store buffer in any order

Other memory models?

- We can specify them in terms of what reorderings are allowed



RMO - Relaxed Memory Order

If memory access 0 appears before memory access 1 in program order, can it bypass program order?

Very relaxed model!

Other memory models?

- FENCE: can always restore order using fences. Accesses cannot be reordered past fences!

memory access 1 \ memory access 0	L	S
L	NO	NO
S	NO	NO

Any Memory Model

If memory access 0 appears before memory access 1 in program order, and there is a FENCE between the two accesses, can it bypass program order?

Schedule

- Memory consistency models:
  - Total store order
  - Relaxed memory consistency
  - **Examples**

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

First thing: change our syntax to pseudo code

Thread 0:

```
L:mov %t0, [y]
S:mov [x], 1
```

Thread 1:

```
L:mov %t1, [x]
S:mov [y], 1
```

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

First thing: change our syntax to pseudo code
You should be able to find natural mappings
to any ISA

Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

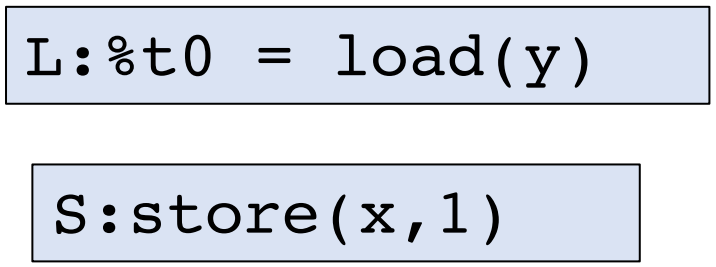
Get out our lego bricks and try for sequential consistency

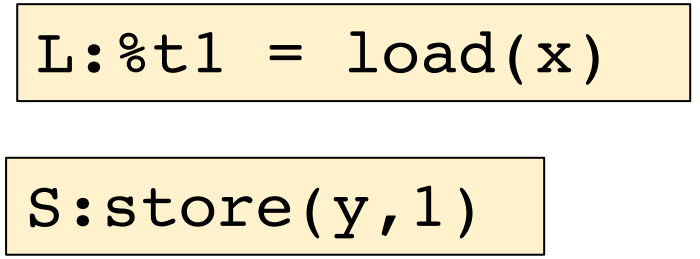
Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```





Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

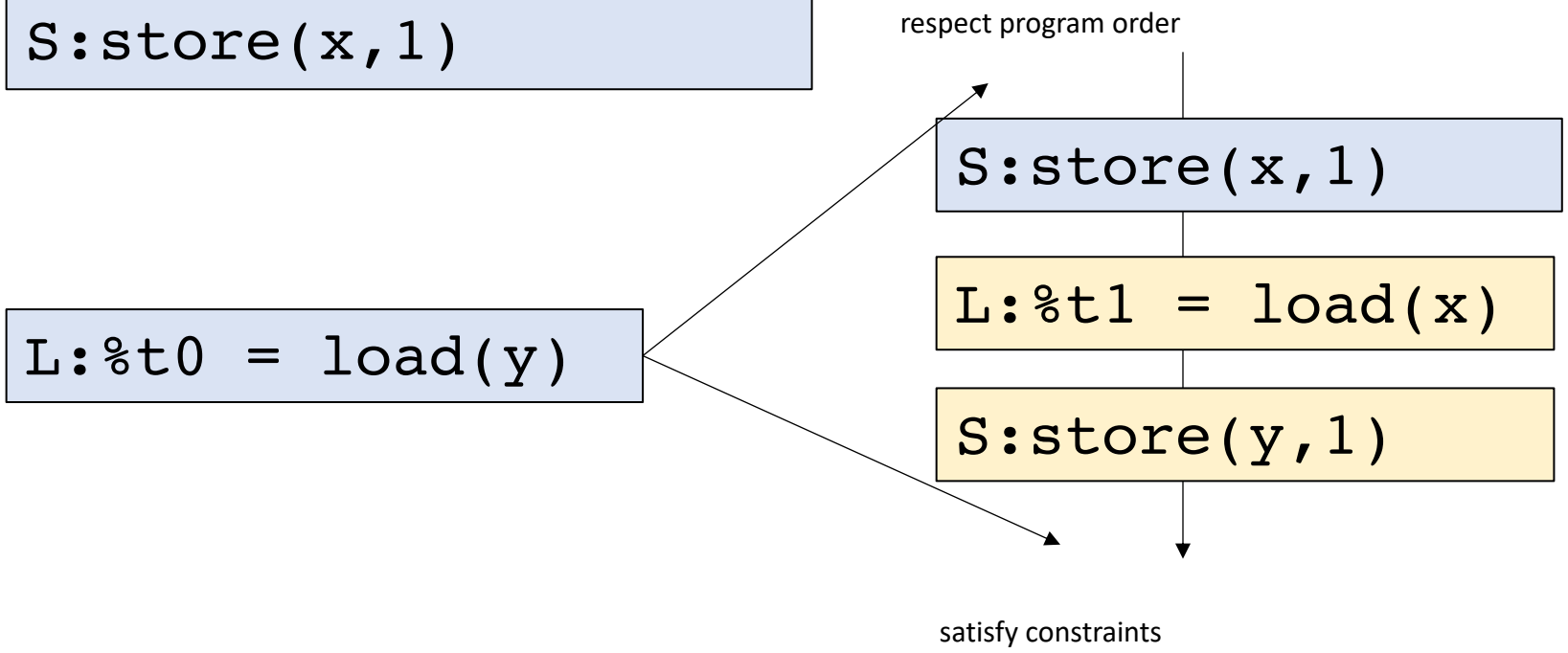
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```



Not allowed under sequential consistency!

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

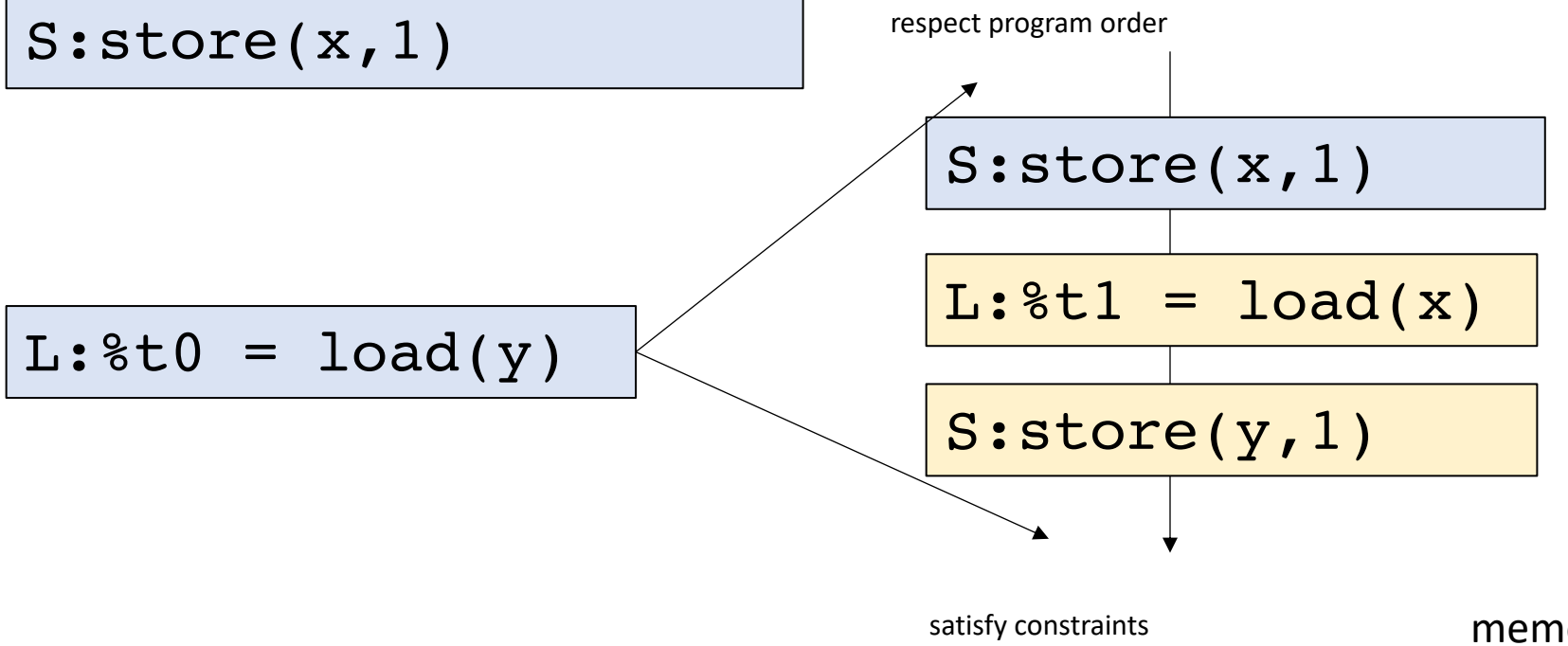
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```



memory access 0

memory access 1	L	S
L	NO	Different address
S	NO	NO

What about TSO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

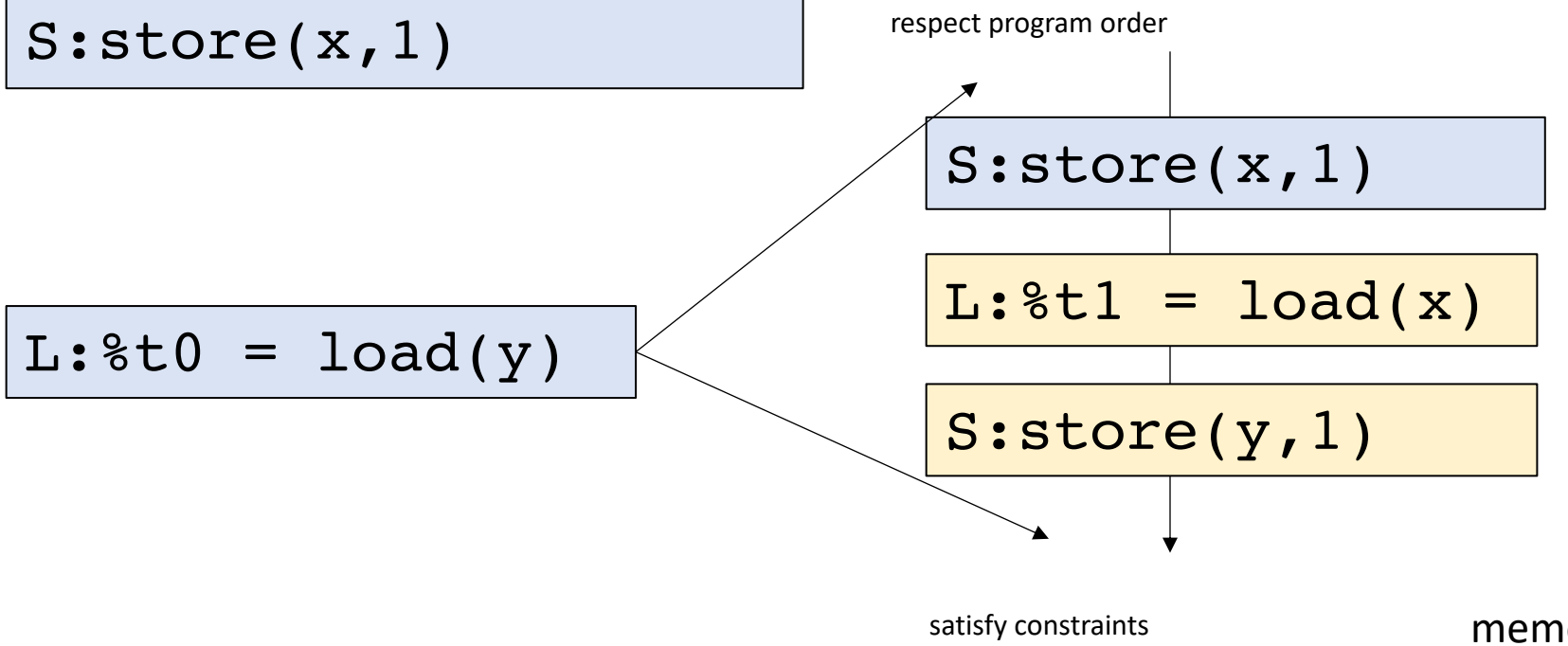
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```



memory access 0

memory access 1

	L	S
L	NO	Different address
S	NO	NO

What about TSO? NOT ALLOWED!

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

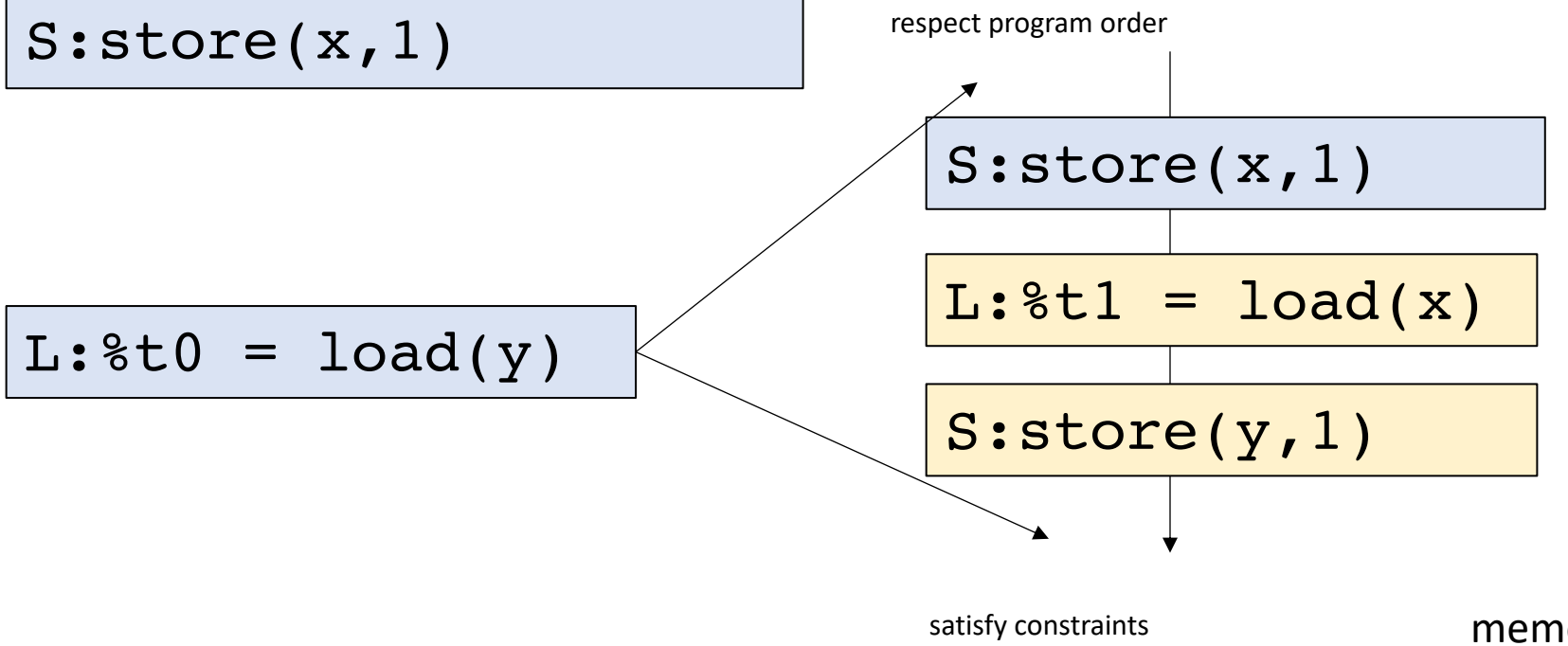
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```



memory access 0

memory access 1

	L	S
L	NO	Different address
S	NO	Different address

What about PSO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

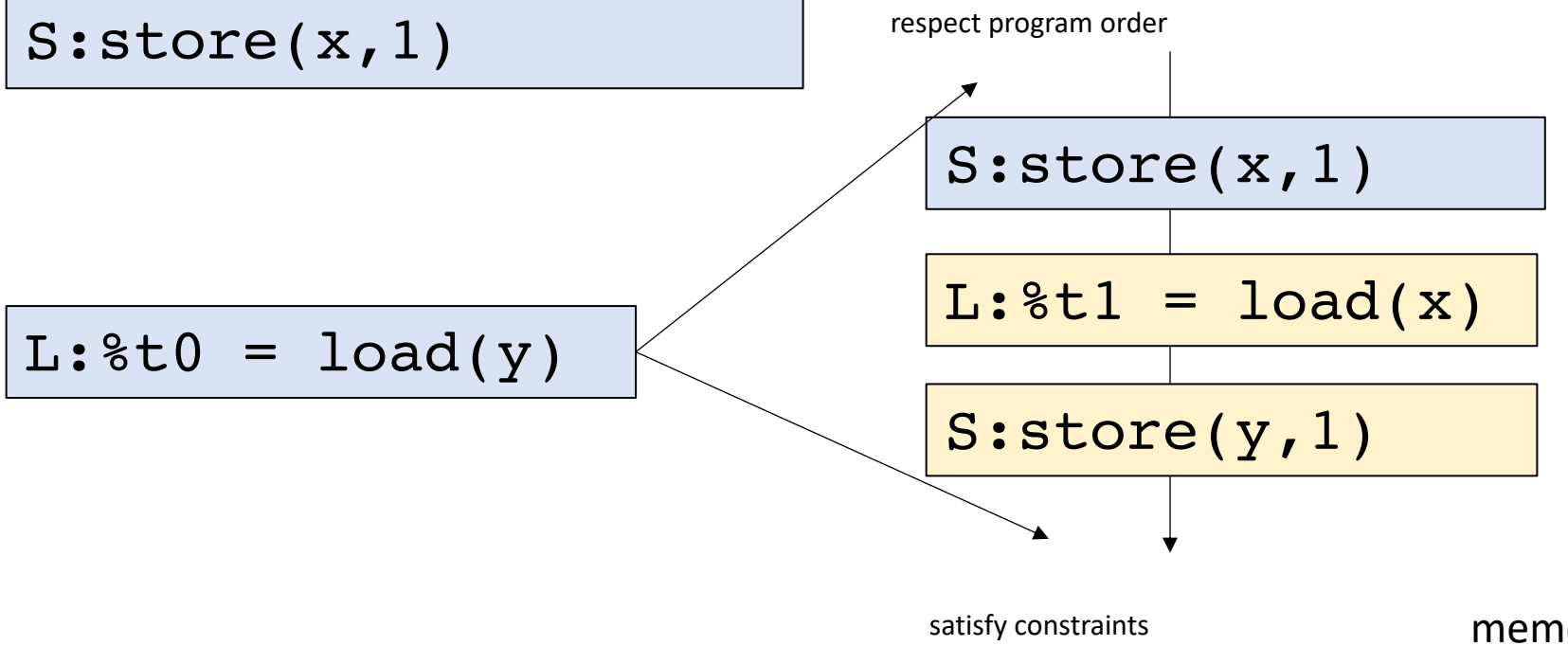
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```



memory access 0

memory access 1

	L	S
L	NO	Different address
S	NO	Different address

What about PSO? NO!

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

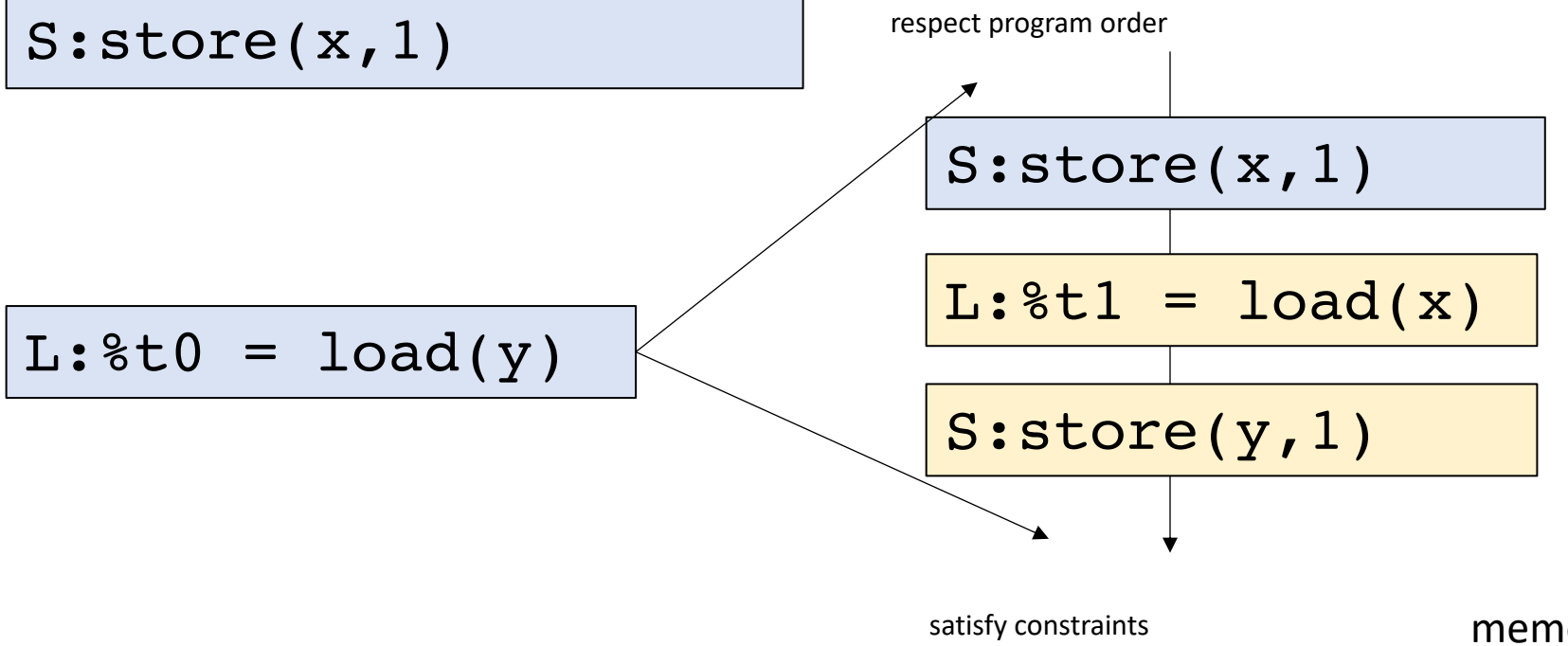
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```



memory access 0

memory access 1

	L	S
L	YES	Different address
S	different address	Different address

What about RMO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

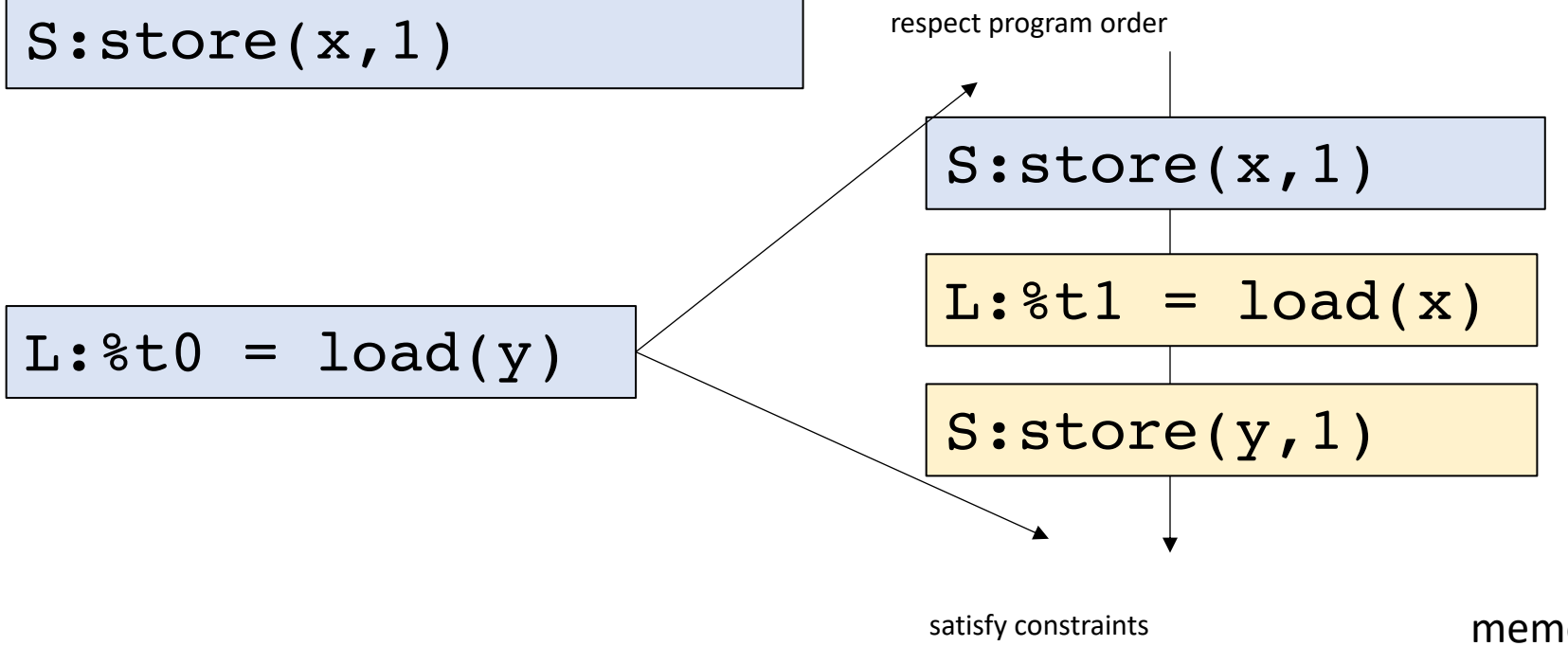
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```



memory access 0

memory access 1	L	S
L	YES	Different address
S	different address	Different address

What about RMO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

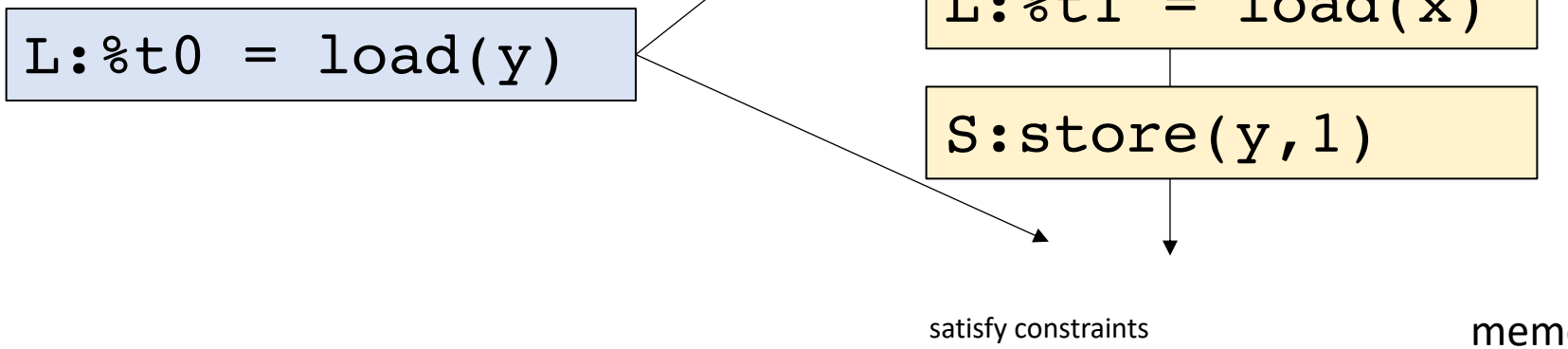
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```



memory access 0

memory access 1	L	S
L	YES	Different address
S	different address	Different address

What about RMO? YES!

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

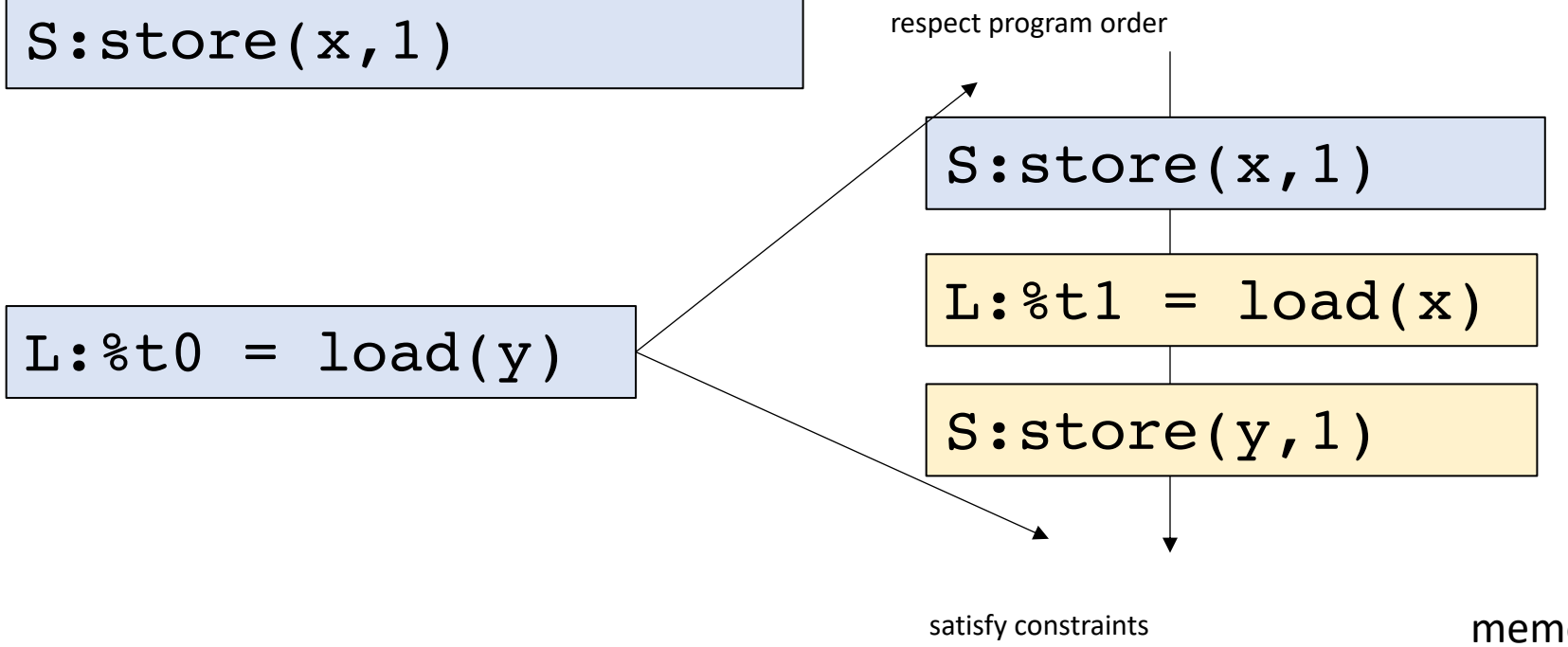
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```



memory access 0

memory access 1	L	S
L	YES	Different address
S	different address	Different address

How do we disallow the behavior in RMO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

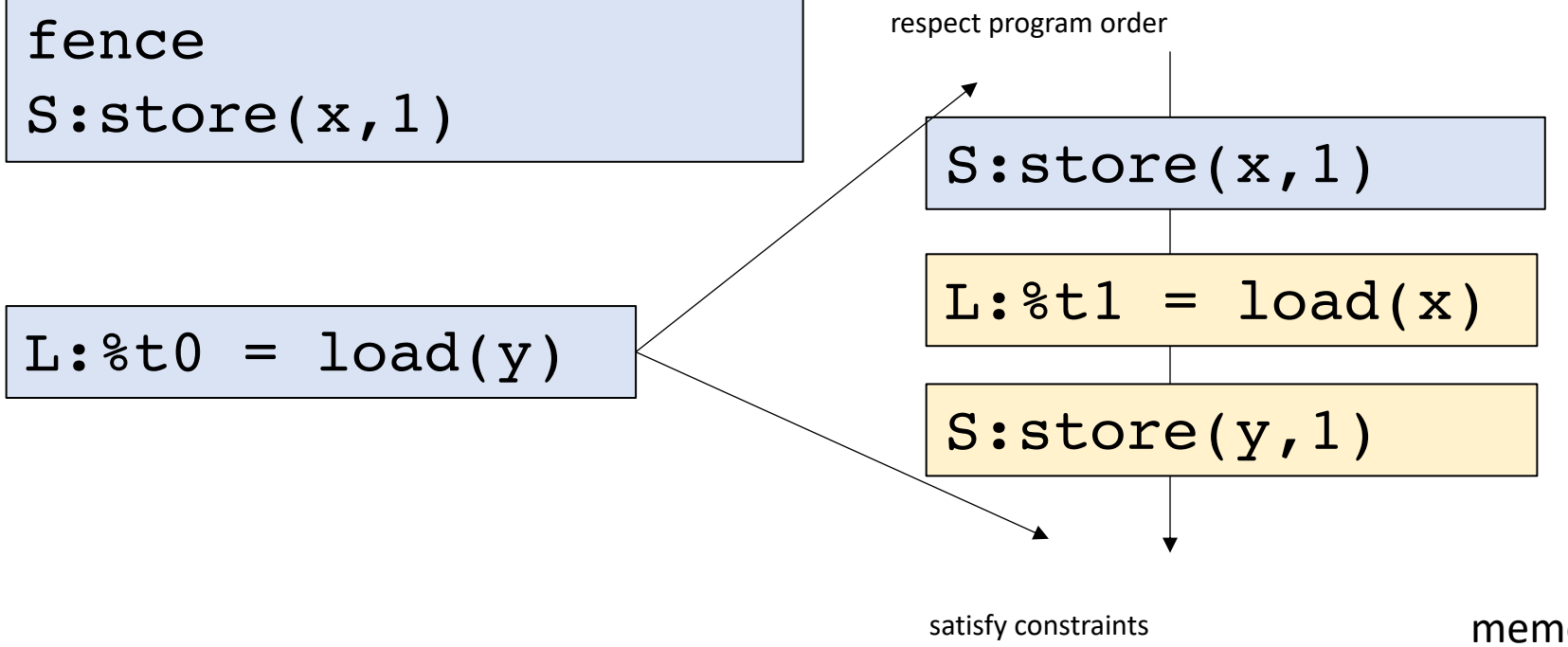
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
fence
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```



memory access 0

memory access 1	L	S
L	YES	Different address
S	different address	Different address

How do we disallow the behavior in RMO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

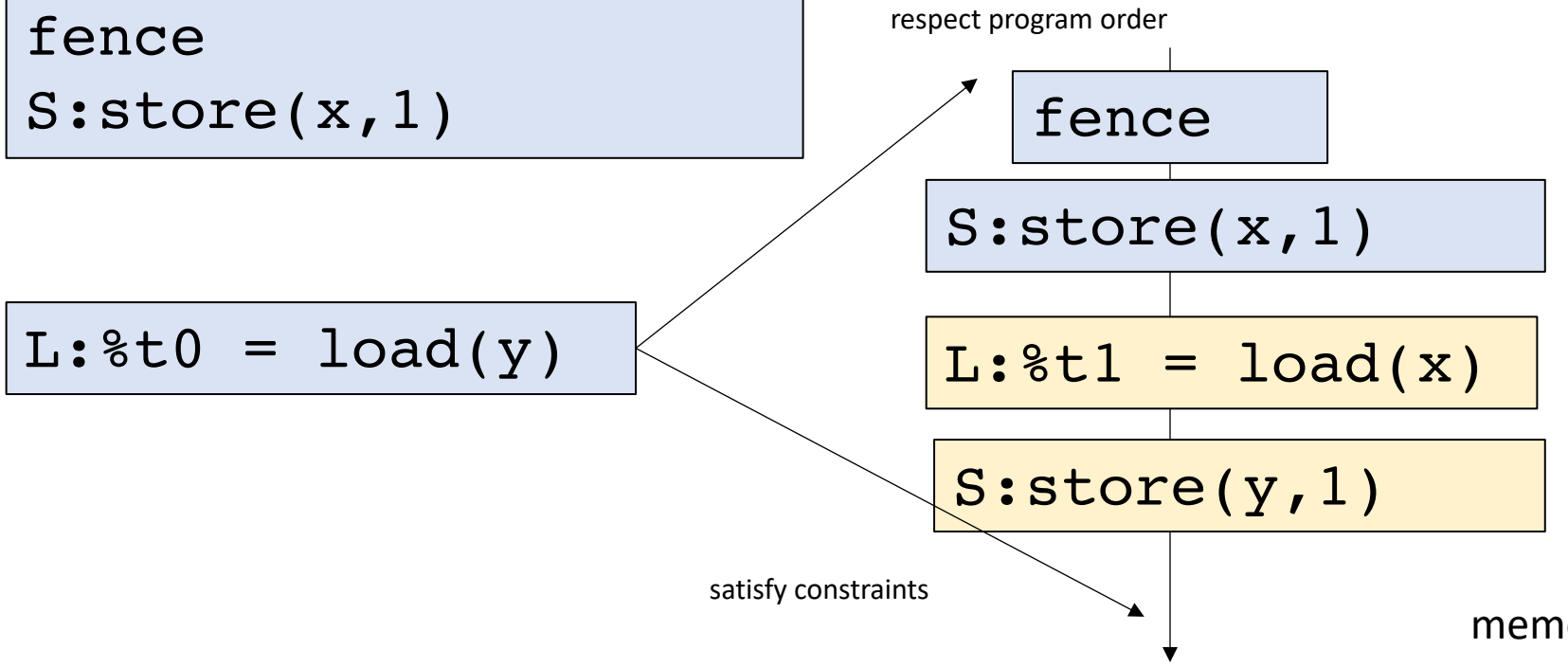
Get out our lego bricks

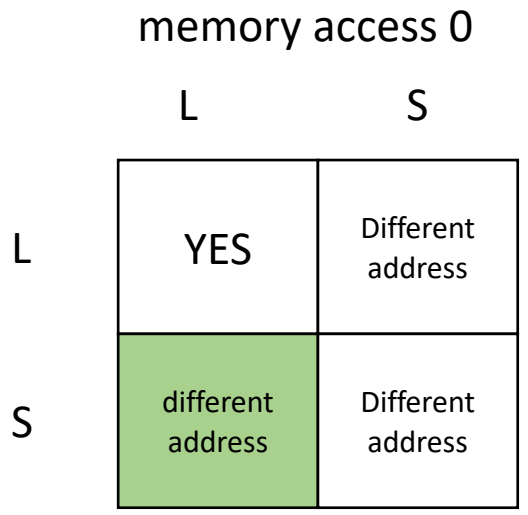
Thread 0:

```
L:%t0 = load(y)
fence
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```





How do we disallow the behavior in RMO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

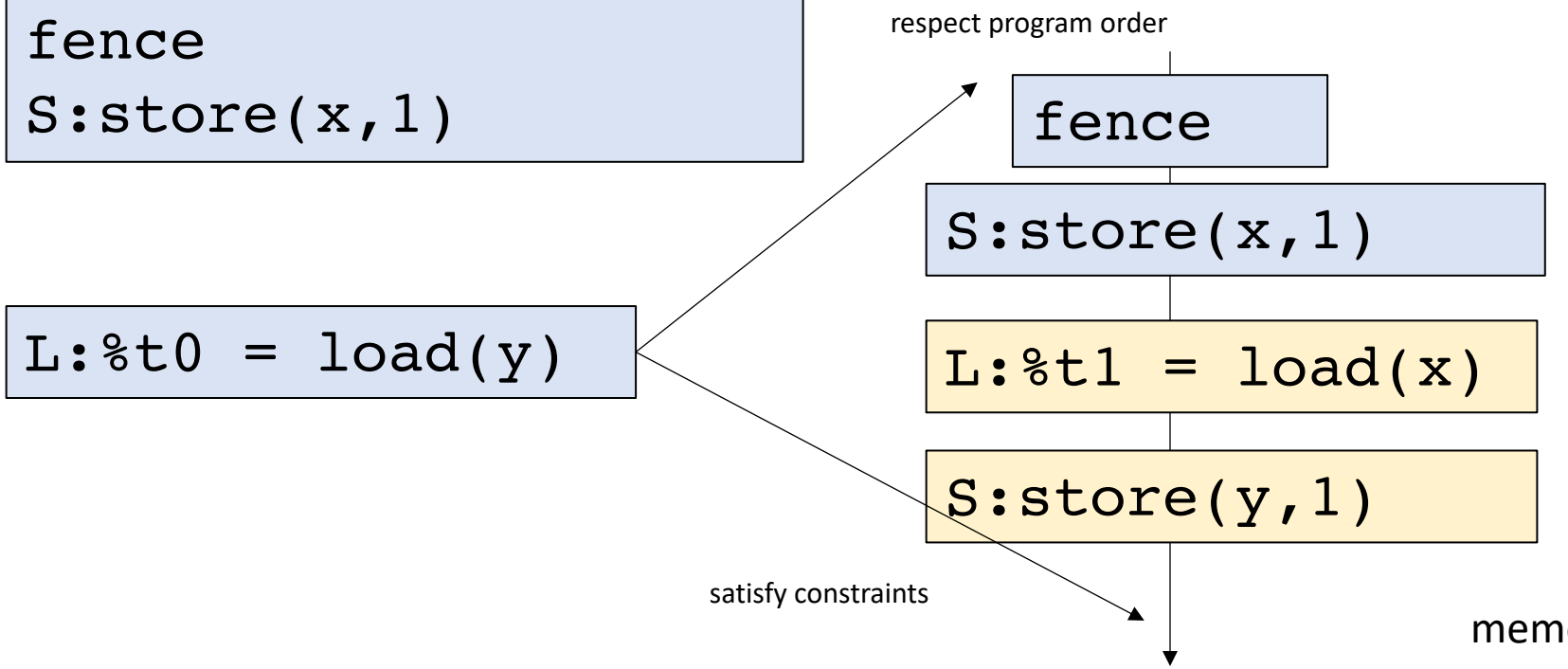
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
fence
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
S:store(y,1)
```



memory access 0

memory access 1	L	S
L	YES	Different address
S	different address	Different address

Now we cannot break program order past the fence!
Are we done?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

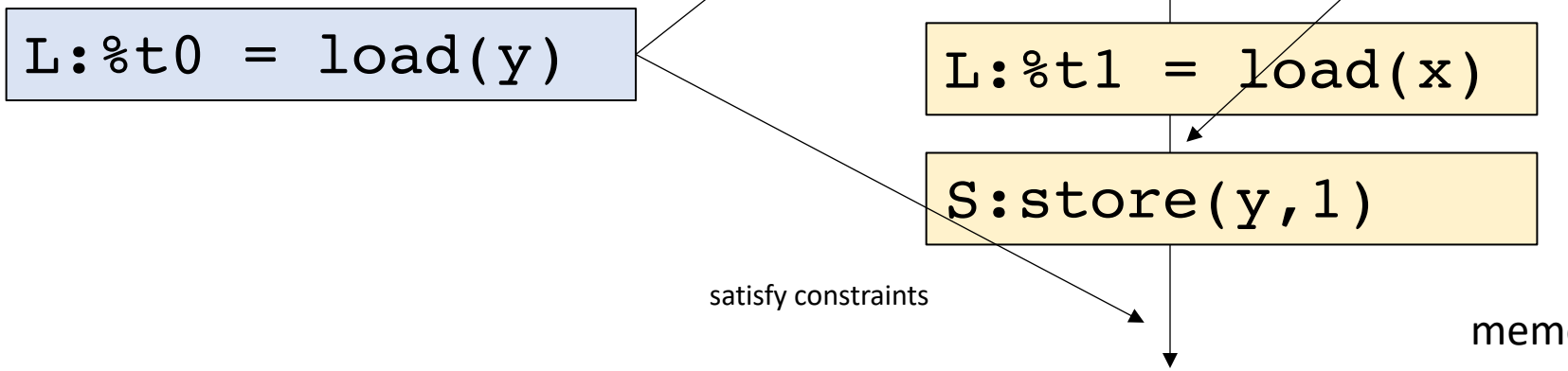
Get out our lego bricks

Thread 0:

```
L:%t0 = load(y)
fence
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
fence
S:store(y,1)
```



memory access 0

memory access 1	L	S
L	YES	Different address
S	different address	Different address

Now we cannot break program order past the fence!
Are we done?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == t1 == 1`?

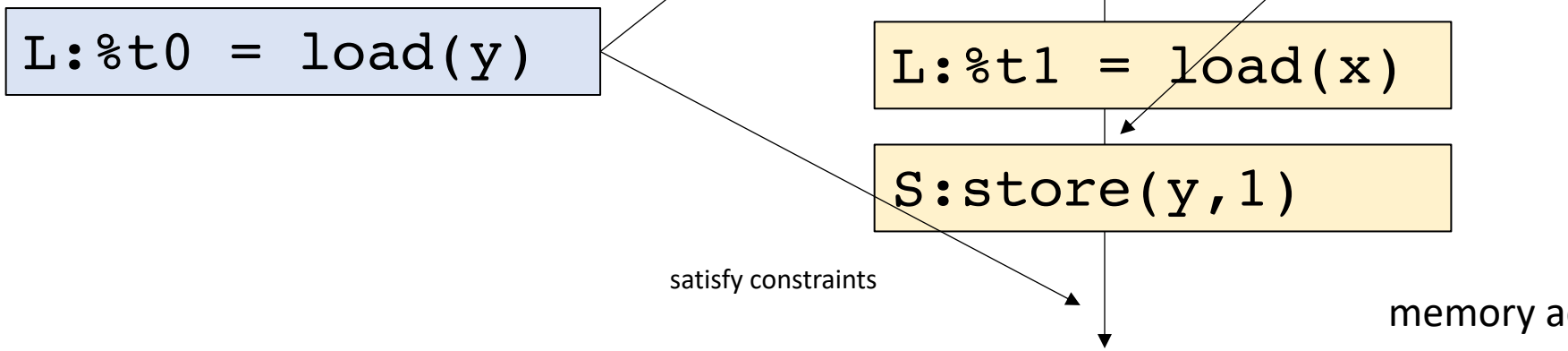
Get out our lego bricks

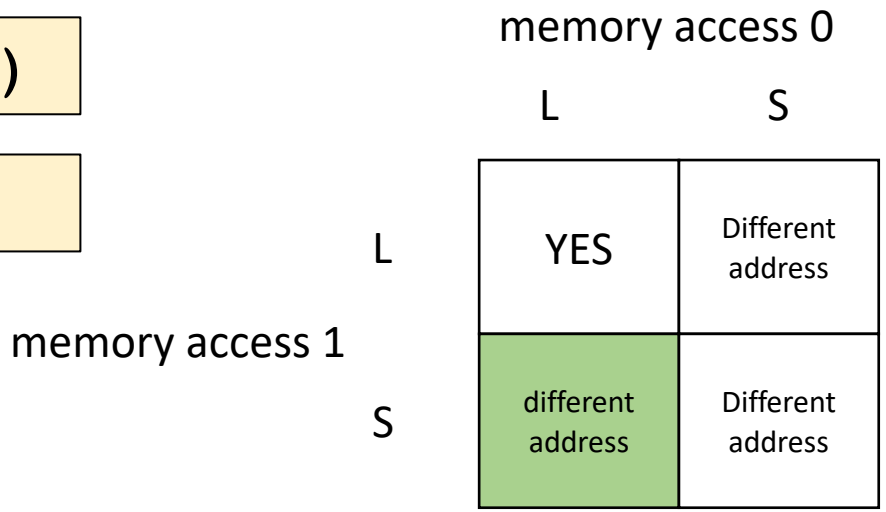
Thread 0:

```
L:%t0 = load(y)
fence
S:store(x,1)
```

Thread 1:

```
L:%t1 = load(x)
fence
S:store(y,1)
```





Now we cannot break program order past the fence!
Are we done? The behavior is no longer allowed

One more example

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == 1 and t1 == 0`?

Thread 0:

```
S:store(x,1)
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```

```
S:store(x,1)
```

```
S:store(y,1)
```

```
L:%t0 = load(y)
```

```
L:%t1 = load(x)
```

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == 1 and t1 == 0`?

start off thinking about sequential consistency

Thread 0:

```
S:store(x,1)
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```

```
S:store(x,1)
```

```
S:store(y,1)
```

```
L:%t0 = load(y)
```

```
L:%t1 = load(x)
```

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == 1 and t1 == 0`?

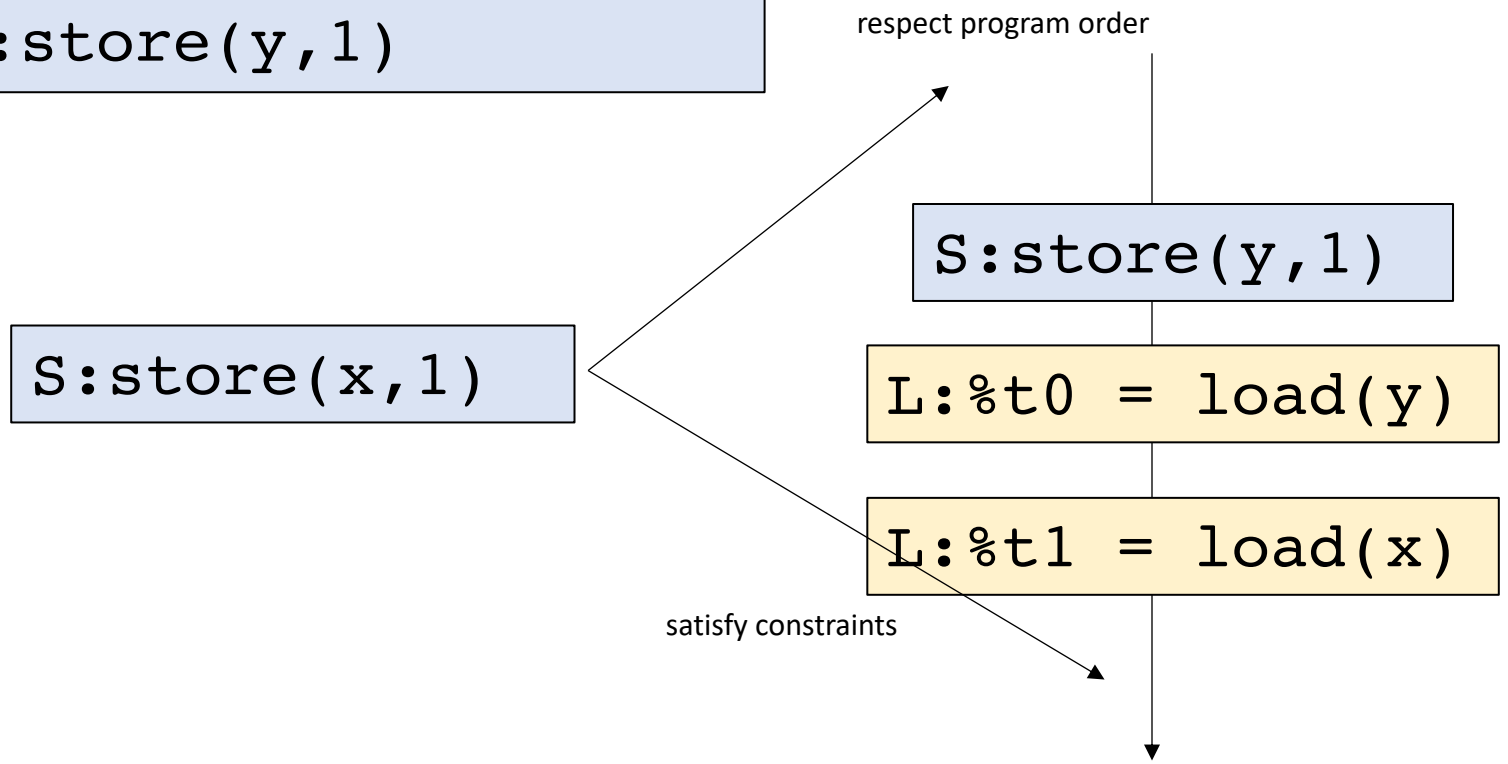
start off thinking about sequential consistency

Thread 0:

```
S:store(x,1)
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```



Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

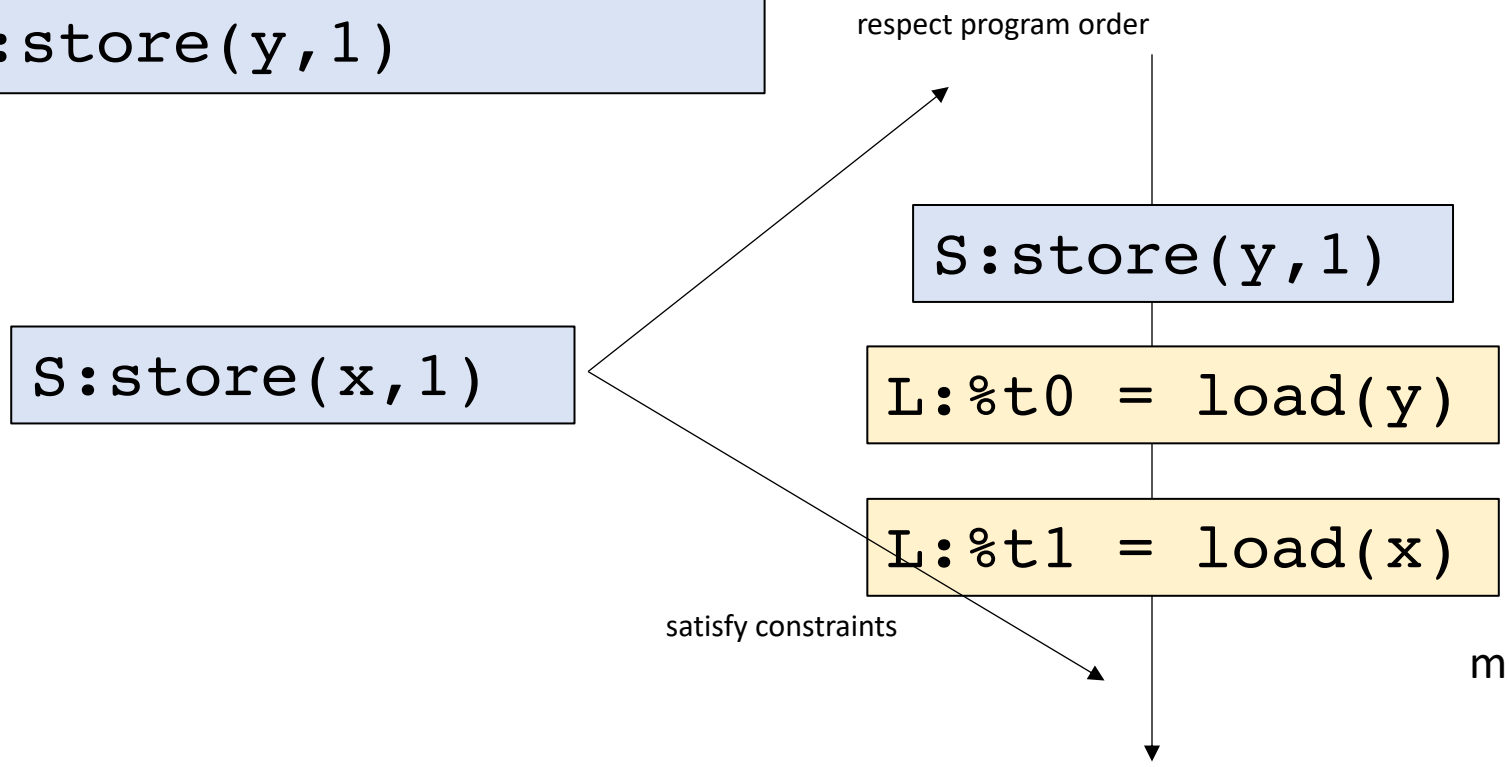
Question: can `t0 == 1 and t1 == 0`?

Thread 0:

```
S:store(x,1)
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```



memory access 0

memory access 1

	L	S
L	NO	Different address
S	NO	NO

What about TSO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

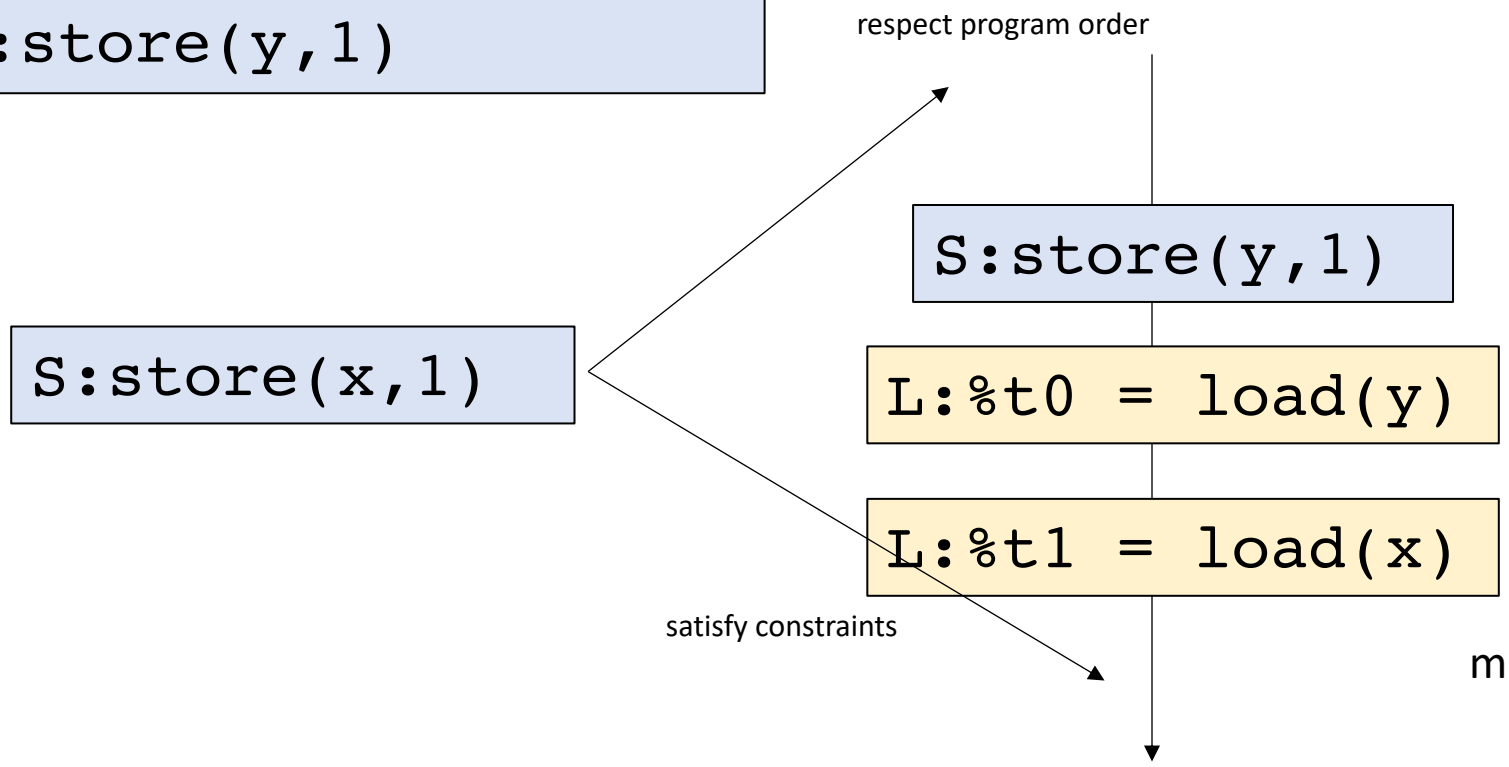
Question: can `t0 == 1 and t1 == 0`?

Thread 0:

```
S:store(x,1)
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```



memory access 0

memory access 1

	L	S
L	NO	Different address
S	NO	NO

What about TSO? NO

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

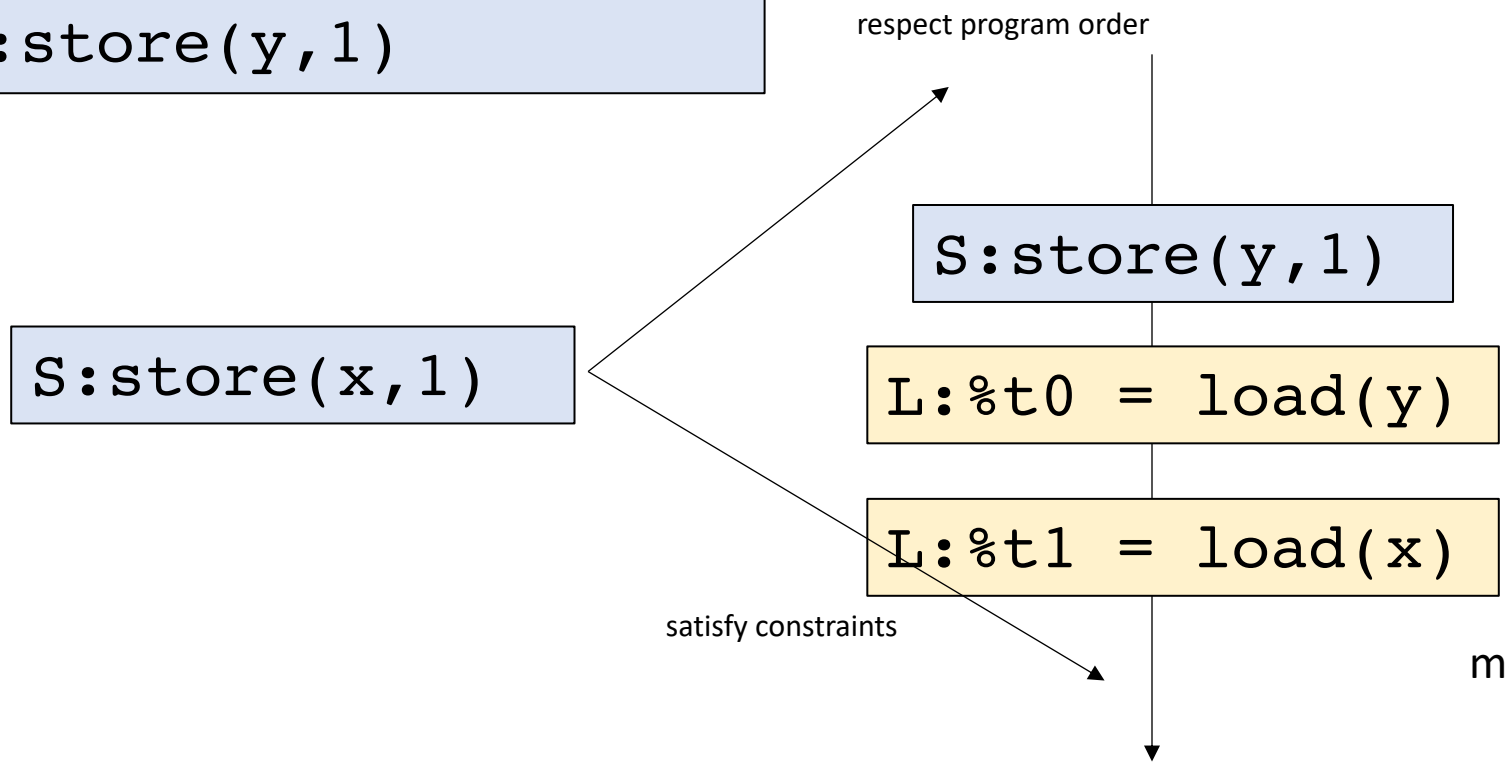
Question: can `t0 == 1 and t1 == 0`?

Thread 0:

```
S:store(x,1)
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```



memory access 0

memory access 1

	L	S
L	NO	Different address
S	NO	Different address

What about PSO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

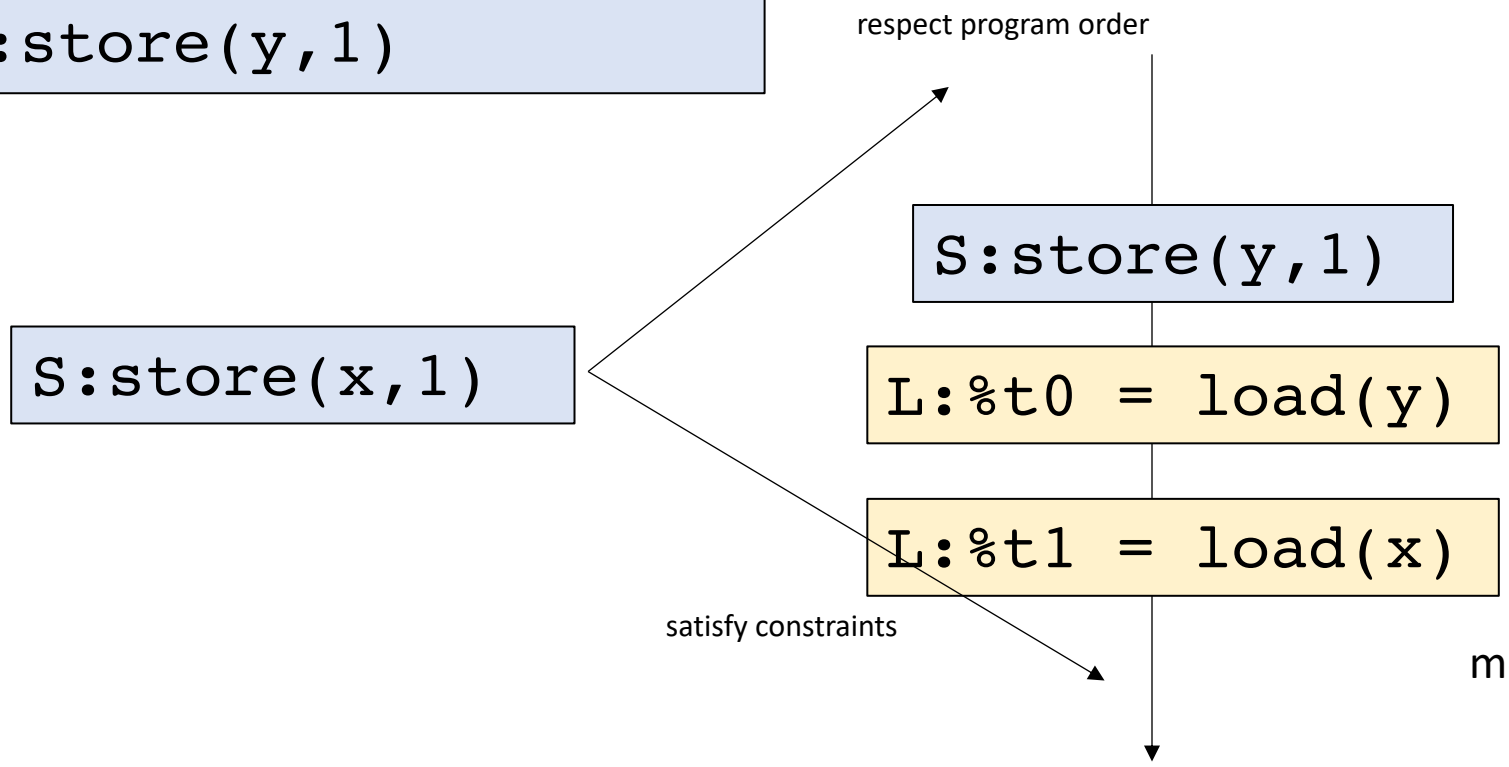
Question: can `t0 == 1 and t1 == 0`?

Thread 0:

```
S:store(x,1)
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```



memory access 0

memory access 1

	L	S
L	NO	Different address
S	NO	Different address

What about PSO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

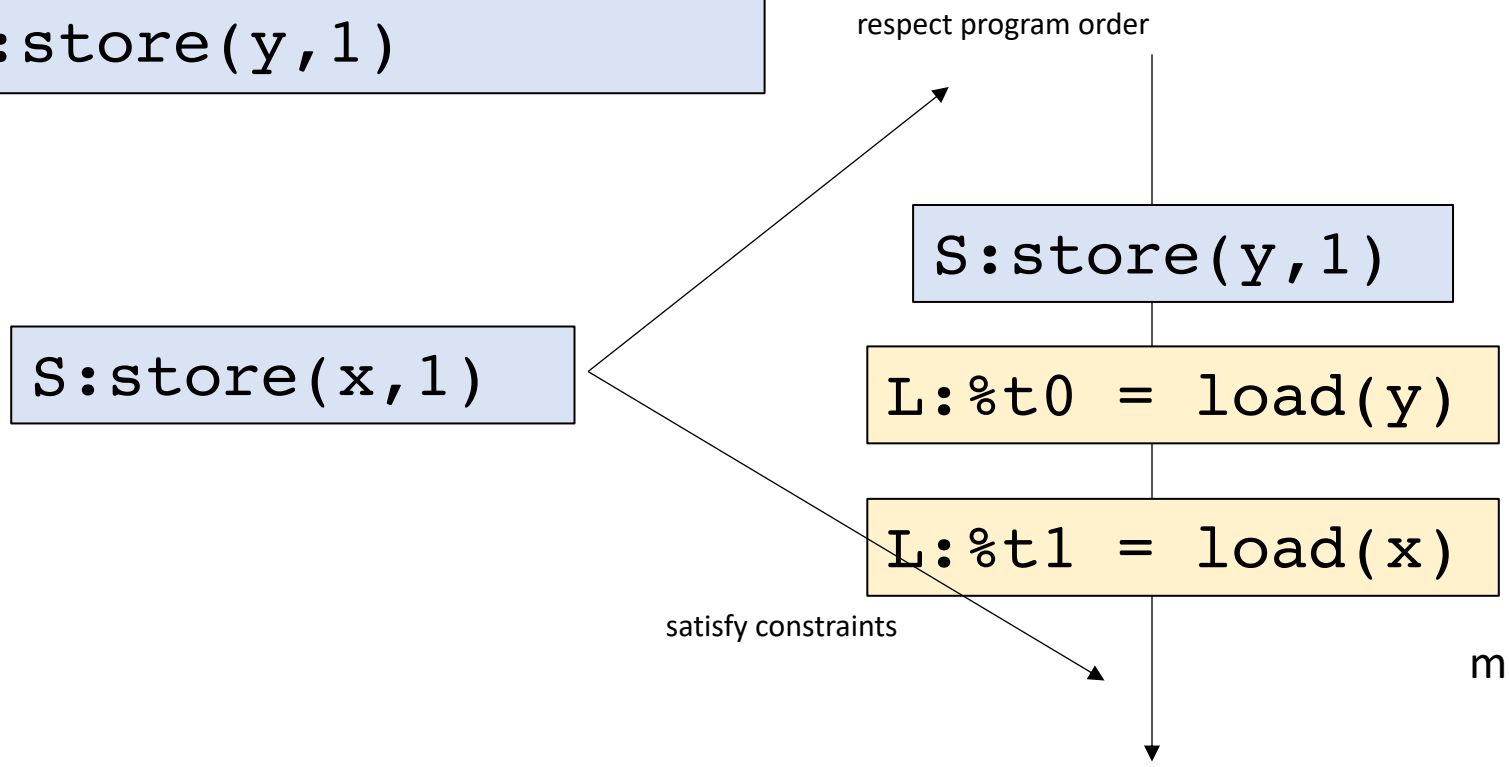
Question: can `t0 == 1 and t1 == 0`?

Thread 0:

```
S:store(x,1)
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```



memory access 0

memory access 1

	L	S
L	NO	Different address
S	NO	Different address

What about PSO? YES

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

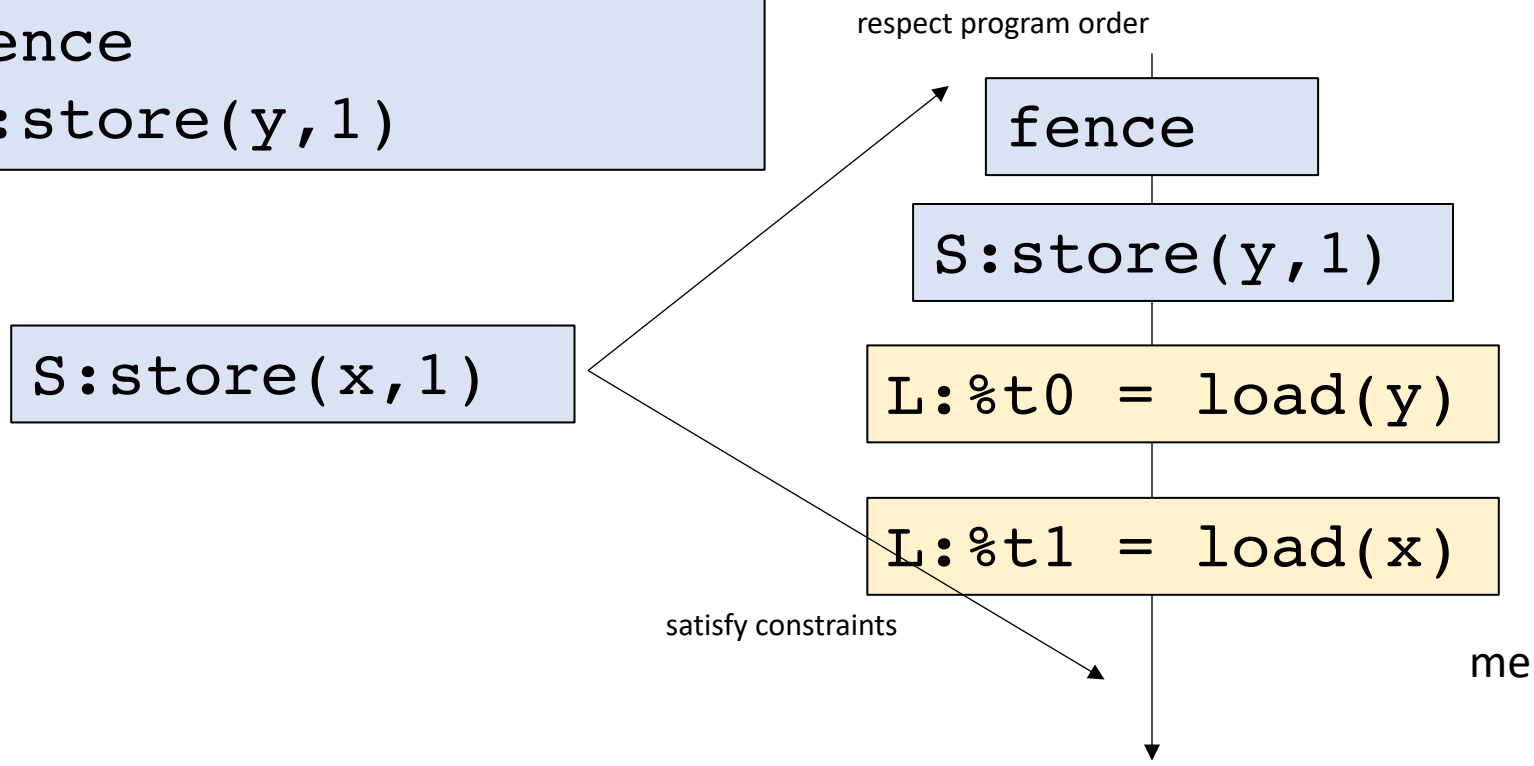
Question: can `t0 == 1 and t1 == 0`?

Thread 0:

```
S:store(x,1)
fence
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```



memory access 0

memory access 1

	L	S
L	NO	Different address
S	NO	Different address

Now it is disallowed in PSO

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

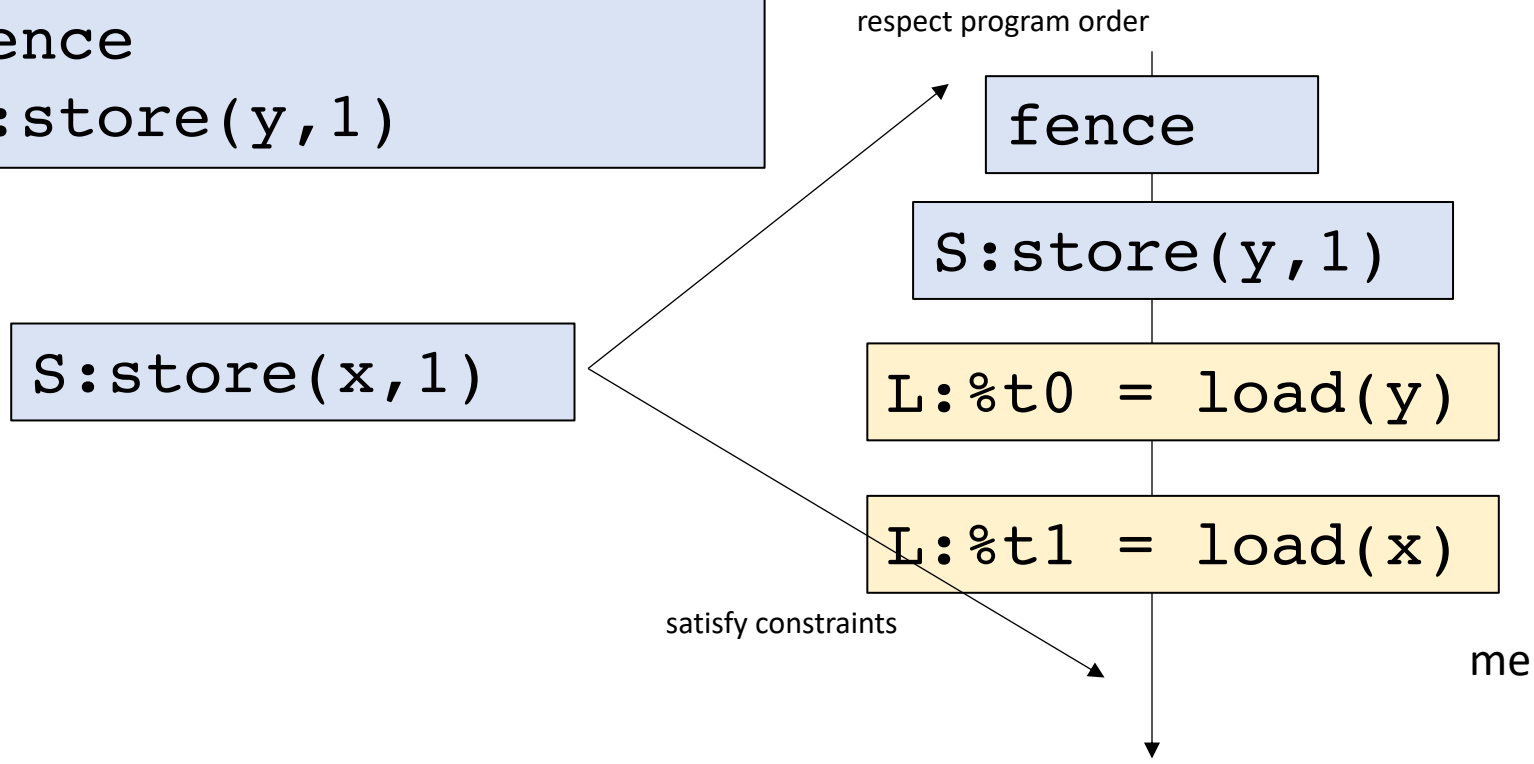
Question: can `t0 == 1 and t1 == 0`?

Thread 0:

```
S:store(x,1)
fence
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```



memory access 0

memory access 1	L	S
L	YES	Different address
S	Different address	Different address

What about RMO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

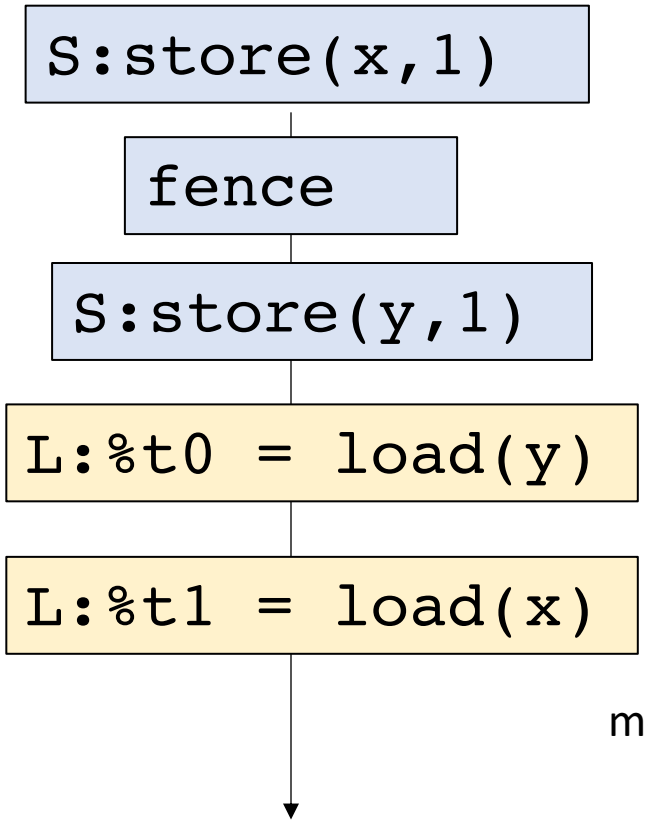
Question: can `t0 == 1 and t1 == 0`?

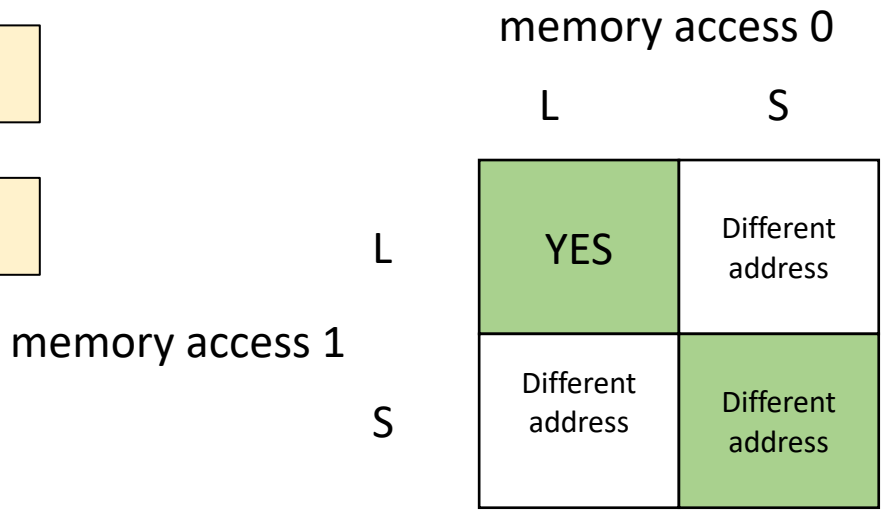
Thread 0:

```
S:store(x,1)
fence
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```





What about RMO?

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == 1 and t1 == 0`?

Thread 0:

```
S:store(x,1)
fence
S:store(y,1)
```

```
L:%t1 = load(x)
```

```
S:store(x,1)
```

```
fence
```

```
S:store(y,1)
```

```
L:%t0 = load(y)
```

Thread 1:

```
L:%t0 = load(y)
S:%t1 = load(x)
```

memory access 1 \ memory access 0	L	S
L	YES	Different address
S	Different address	Different address

What about RMO? The loads can be reordered also!

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

Question: can `t0 == 1 and t1 == 0`?

Thread 0:

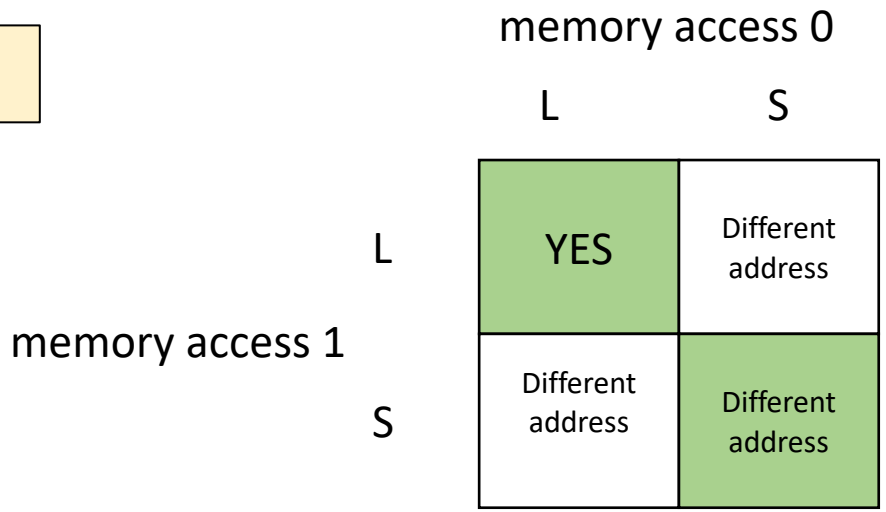
```
S:store(x,1)
fence
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
fence
S:%t1 = load(x)
```

```
L:%t1 = load(x)
```

```
S:store(x,1)
fence
S:store(y,1)
L:%t0 = load(y)
fence
```



What about RMO? add a fence

Global variable:

```
int x[1] = {0};
int y[1] = {0};
```

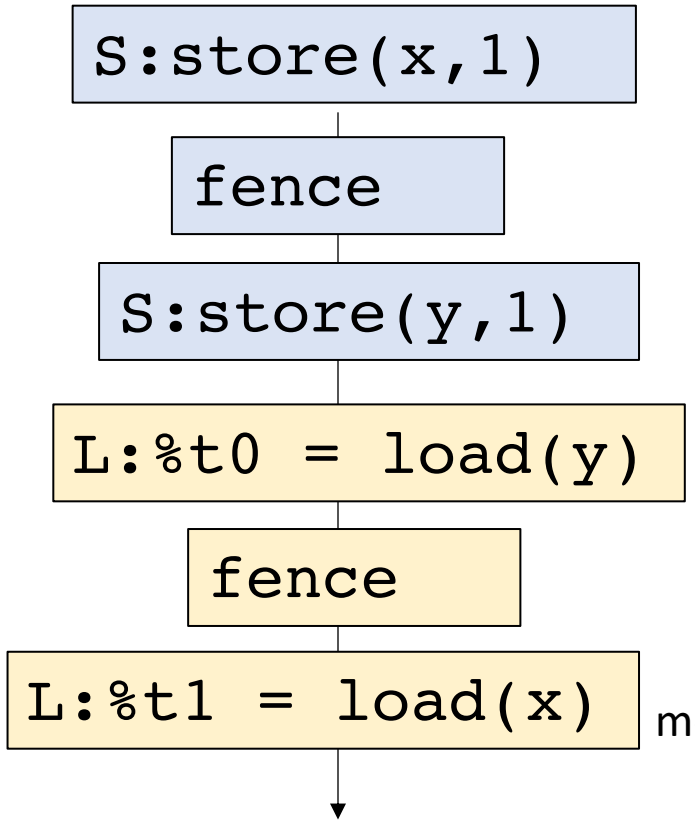
Question: can `t0 == 1 and t1 == 0`?

Thread 0:

```
S:store(x,1)
fence
S:store(y,1)
```

Thread 1:

```
L:%t0 = load(y)
fence
S:%t1 = load(x)
```



memory access 0

memory access 1	L	S
L	YES	Different address
S	Different address	Different address

Now the relaxed behavior is disallowed

Memory consistency in the real world

- Historic Chips:
  - X86: TSO
    - Surprising robust
    - mutexes and concurrent data structures generally seem to work
    - watch out for store buffering
  - IBM Power and ARM
    - Very relaxed. Similar to RMO with even more rules
    - Mutexes and data structures must be written with care
    - ARM recently strengthened theirs

Memory consistency in the real world

- Historic Chips:
  - X86: TSO
    - Surprising robust
    - mutexes and concurrent data structures generally seem to work
    - watch out for store buffering
  - IBM Power and ARM
    - Very relaxed. Similar to RMO with even more rules
    - Mutexes and data structures must be written with care
    - ARM recently strengthened theirs

Companies have a history of providing insufficient documentation about their rules: academics have then gone and figured it out!

Getting better these days

Memory consistency in the real world

- Modern Chips:
  - RISC-V : two specs: one similar to TSO, one similar to RMO
  - Apple M1: toggles between TSO and weaker

  - Vulkan does not provide any fences that provide S - L ordering

Memory consistency in the real world

- PSO and RMO were never implemented widely
  - I have not met anyone who knows of any RMO taped out chip
  - They are part of SPARC ISAs (i.e. RISC-V before it was cool)
  - These memory models might have been part of specialized chips
- Interestingly:
  - Early Nvidia GPUs appeared to informally implement RMO
- Other chips have very strange memory models:
  - Alpha DEC - basically no rules

See you on Wednesday!

- Get HW 3 in!
- Watch out for HW2 grades
- Finishing up memory models on Wednesday