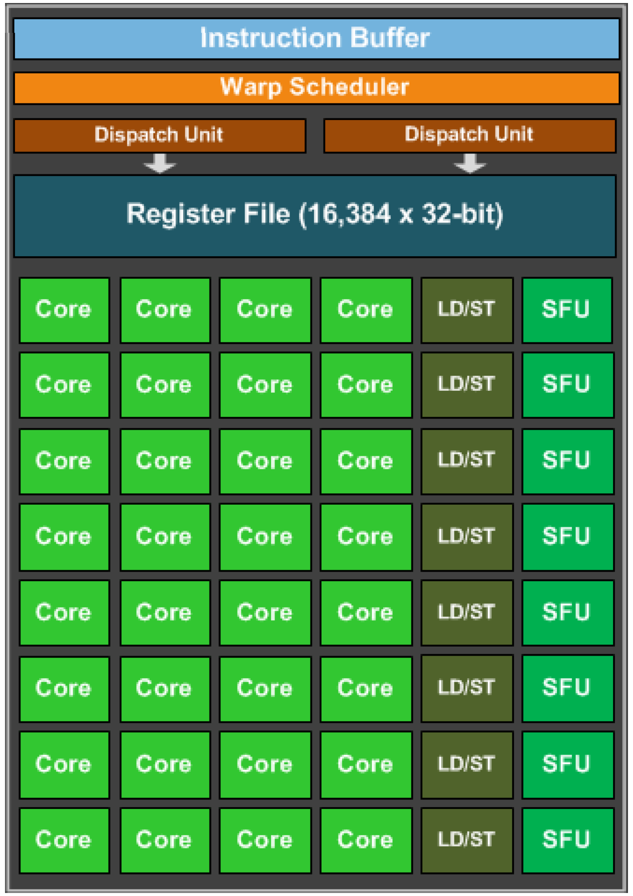
CSE113: Parallel Programming

- **Topics**:
  - Finish up GPUs
  - Homework 4
  - Start on memory models



Instruction Buffer

Warp Scheduler

Dispatch Unit | Dispatch Unit

Register File (16,384 x 32-bit)

Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU

Announcements

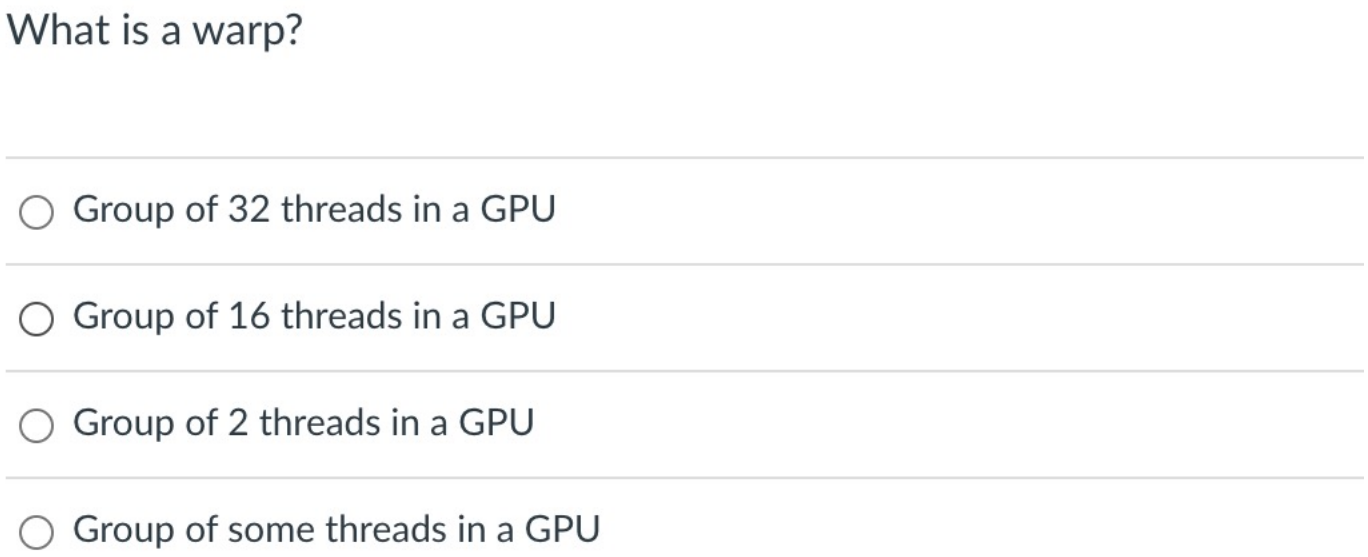
Grading

- Grades for HW 3 should be out by the end of the week

- HW 4 is released.

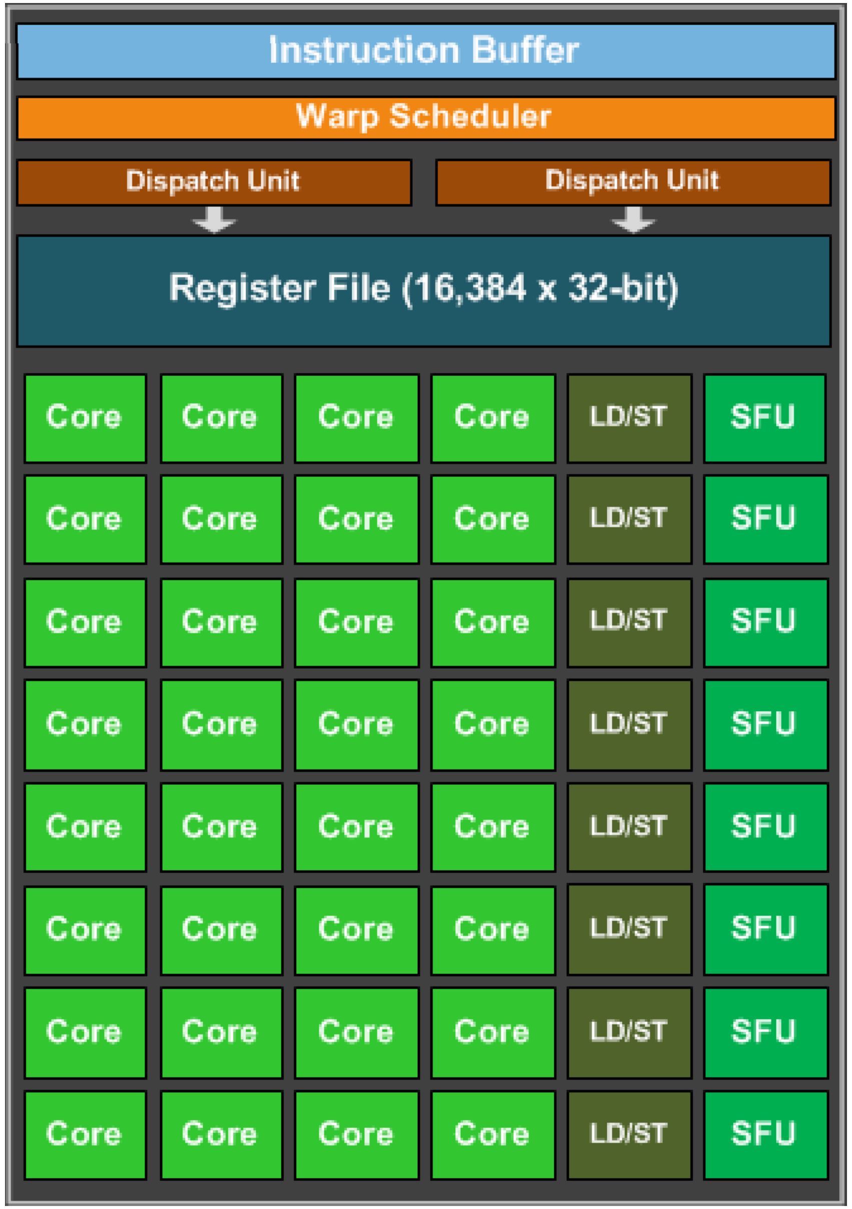
Previous quiz + Review

Previous quiz + Review



What is a warp?

- ○ Group of 32 threads in a GPU
- ○ Group of 16 threads in a GPU
- ○ Group of 2 threads in a GPU
- ○ Group of some threads in a GPU



Instruction Buffer

Warp Scheduler

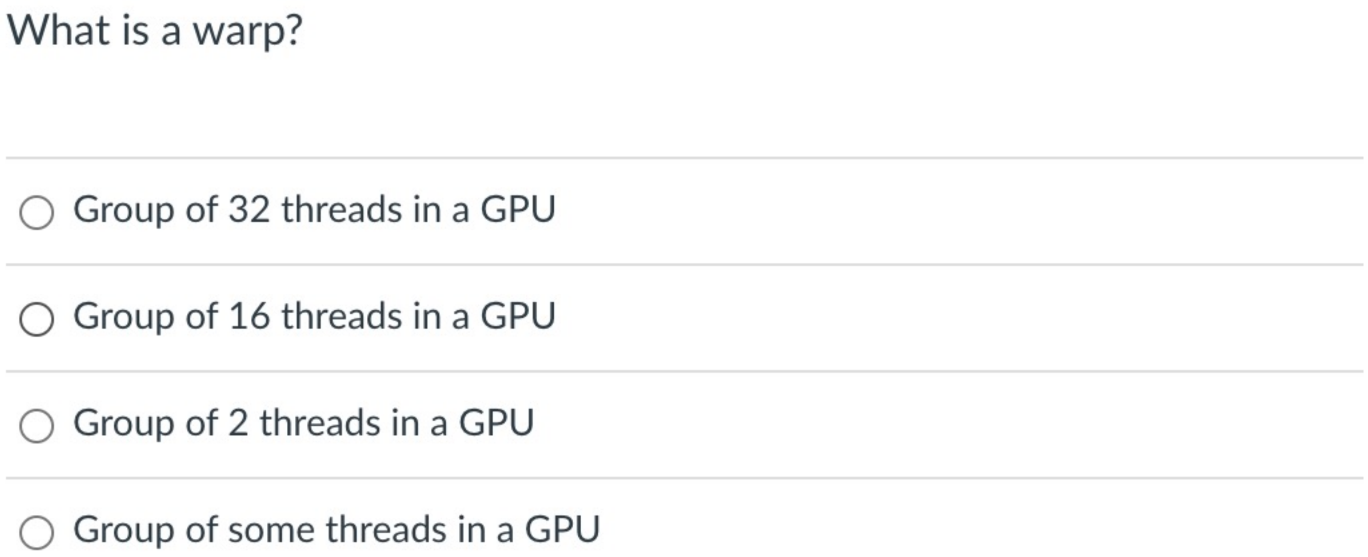
Dispatch Unit

Dispatch Unit

Register File (16,384 x 32-bit)

Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU
Core	Core	Core	Core	LD/ST	SFU

Previous quiz + Review



What is a warp?

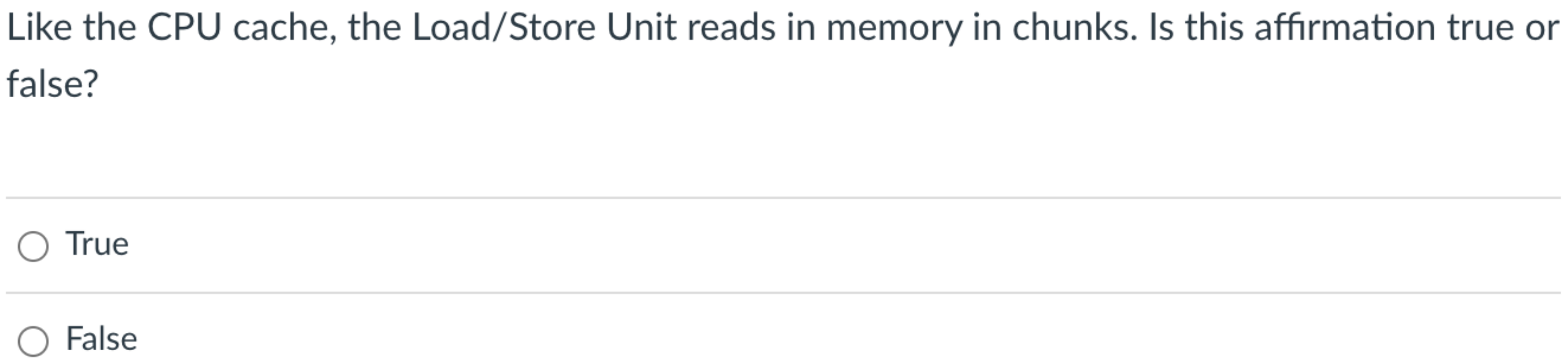
-> ○ Group of 32 threads in a GPU

○ Group of 16 threads in a GPU

○ Group of 2 threads in a GPU

-> ○ Group of some threads in a GPU

Previous quiz + Review



Like the CPU cache, the Load/Store Unit reads in memory in chunks. Is this affirmation true or false?

○ True

○ False

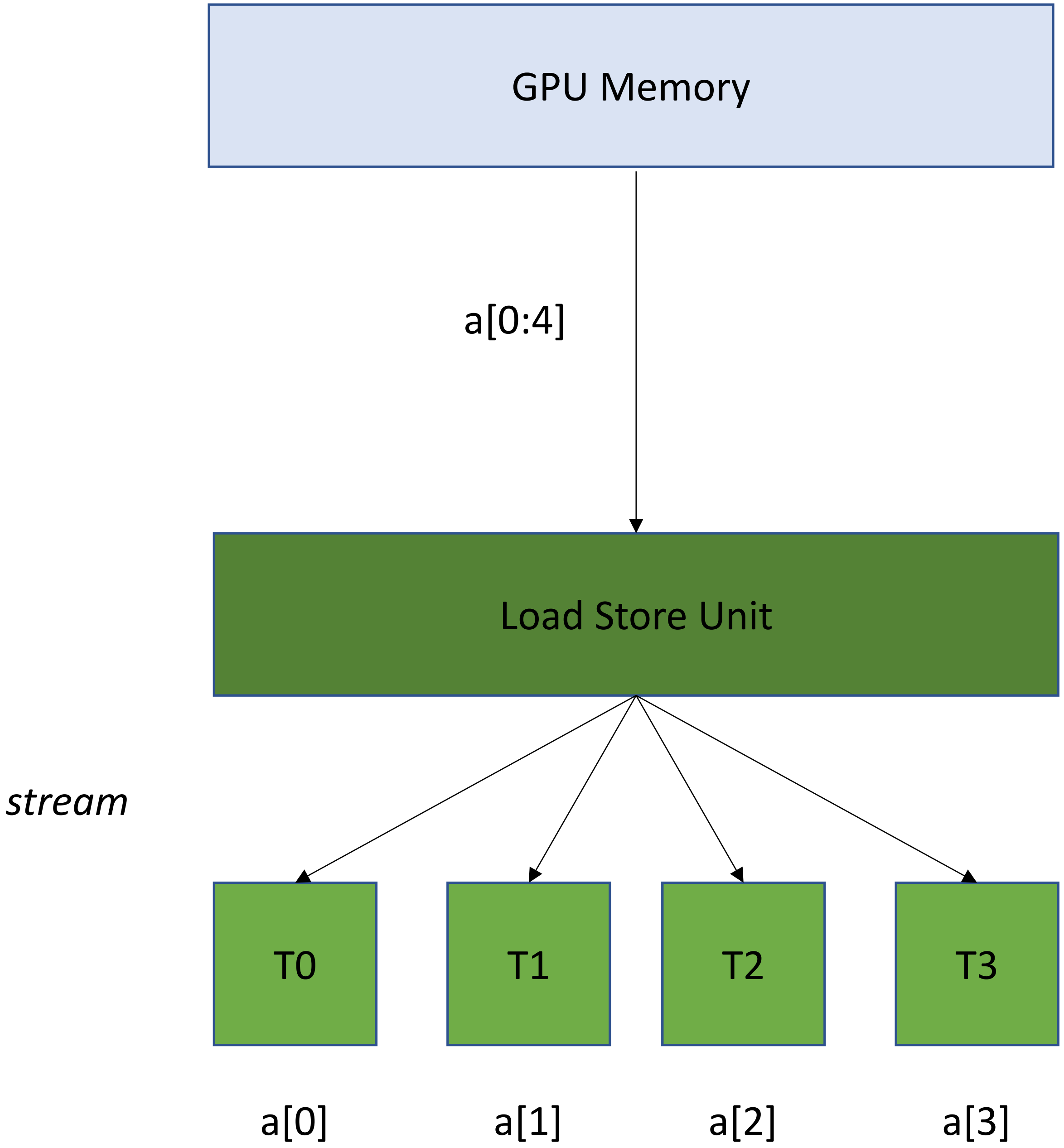
4 cores are accessing memory. What can happen

Read contiguous values

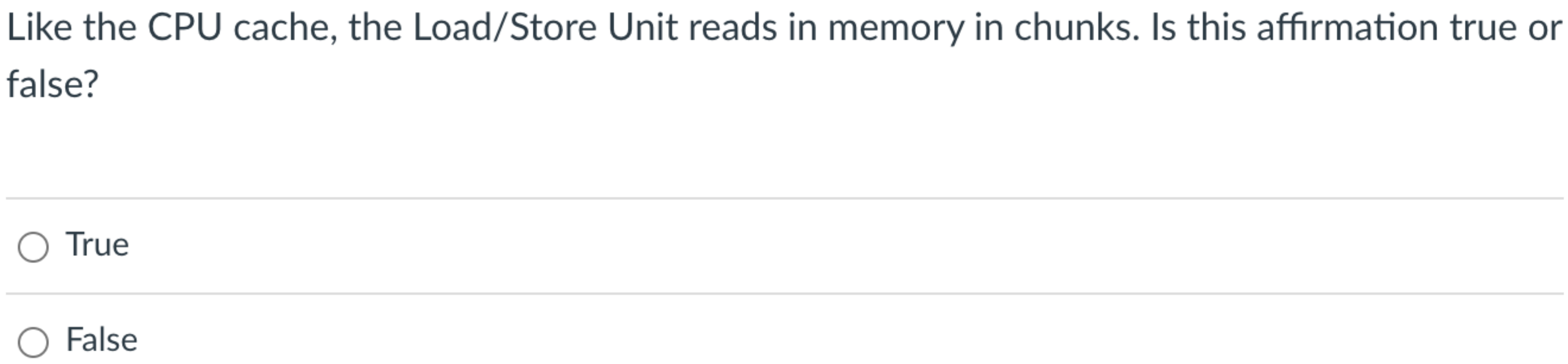
Like the CPU cache, the Load/Store Unit reads in memory in chunks. 16 bytes

Can easily distribute the values to the threads

1 request to GPU memory



Previous quiz + Review



Like the CPU cache, the Load/Store Unit reads in memory in chunks. Is this affirmation true or false?

-> ○ True

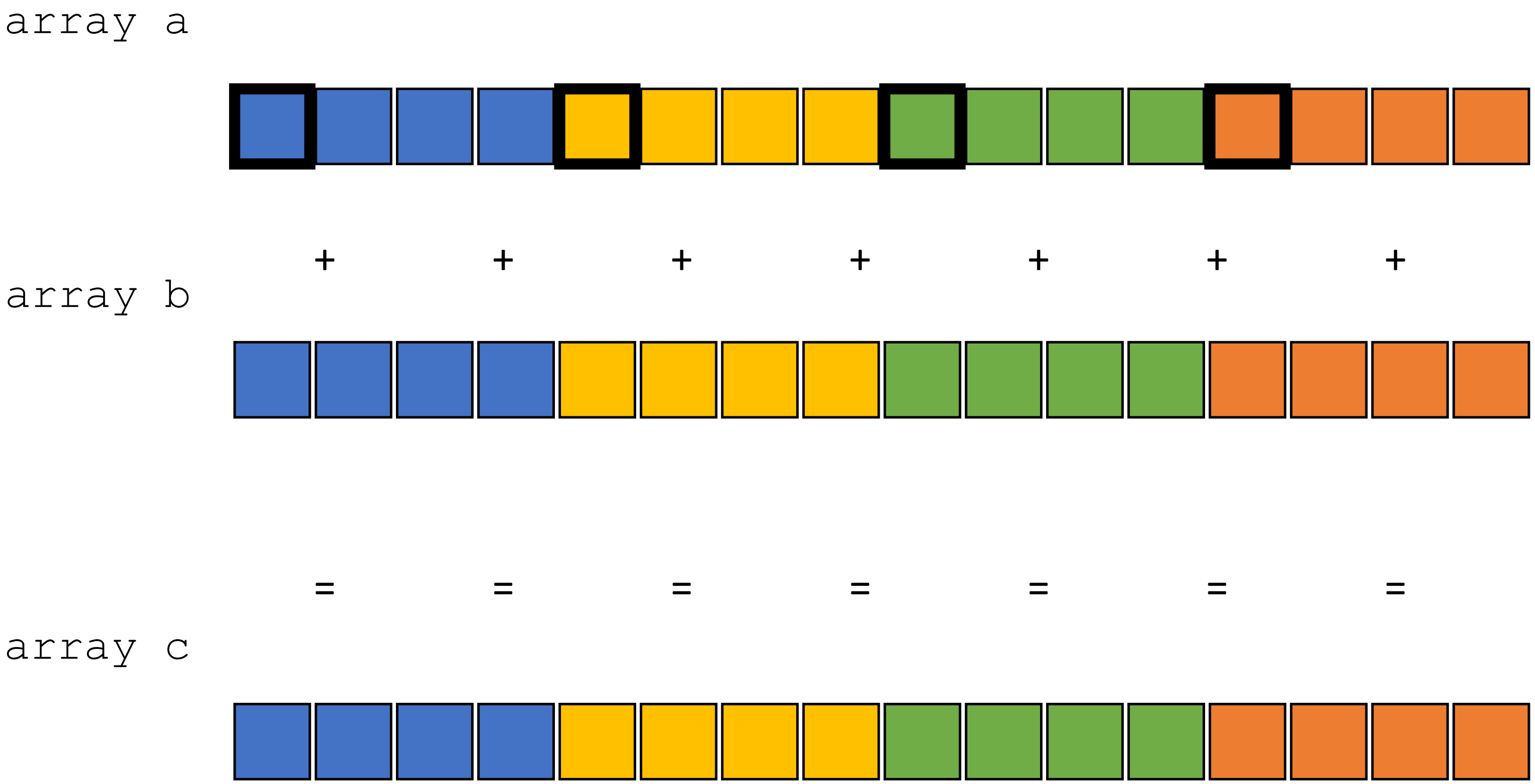
○ False

Previous quiz + Review

What could we observe with the demonstrations made in class about memory accesses on GPUs?

Chunked Pattern

the first element accessed by the 4 threads sharing a load store unit. What sort of access is this?



Computation can easily be divided into threads

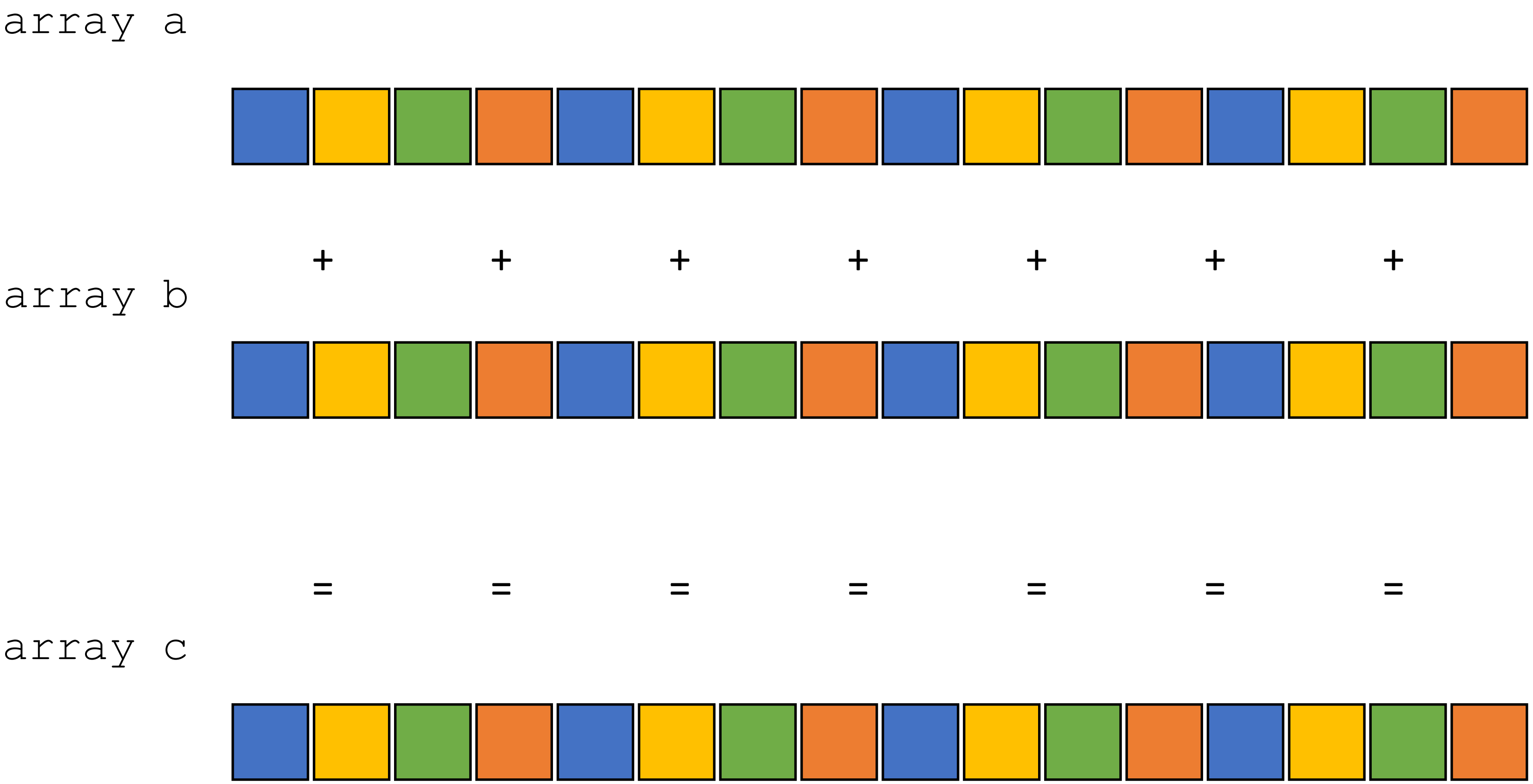
Thread 0 - Blue
Thread 1 - Yellow
Thread 2 - Green
Thread 3 - Orange

How can we fix this

Stride Pattern

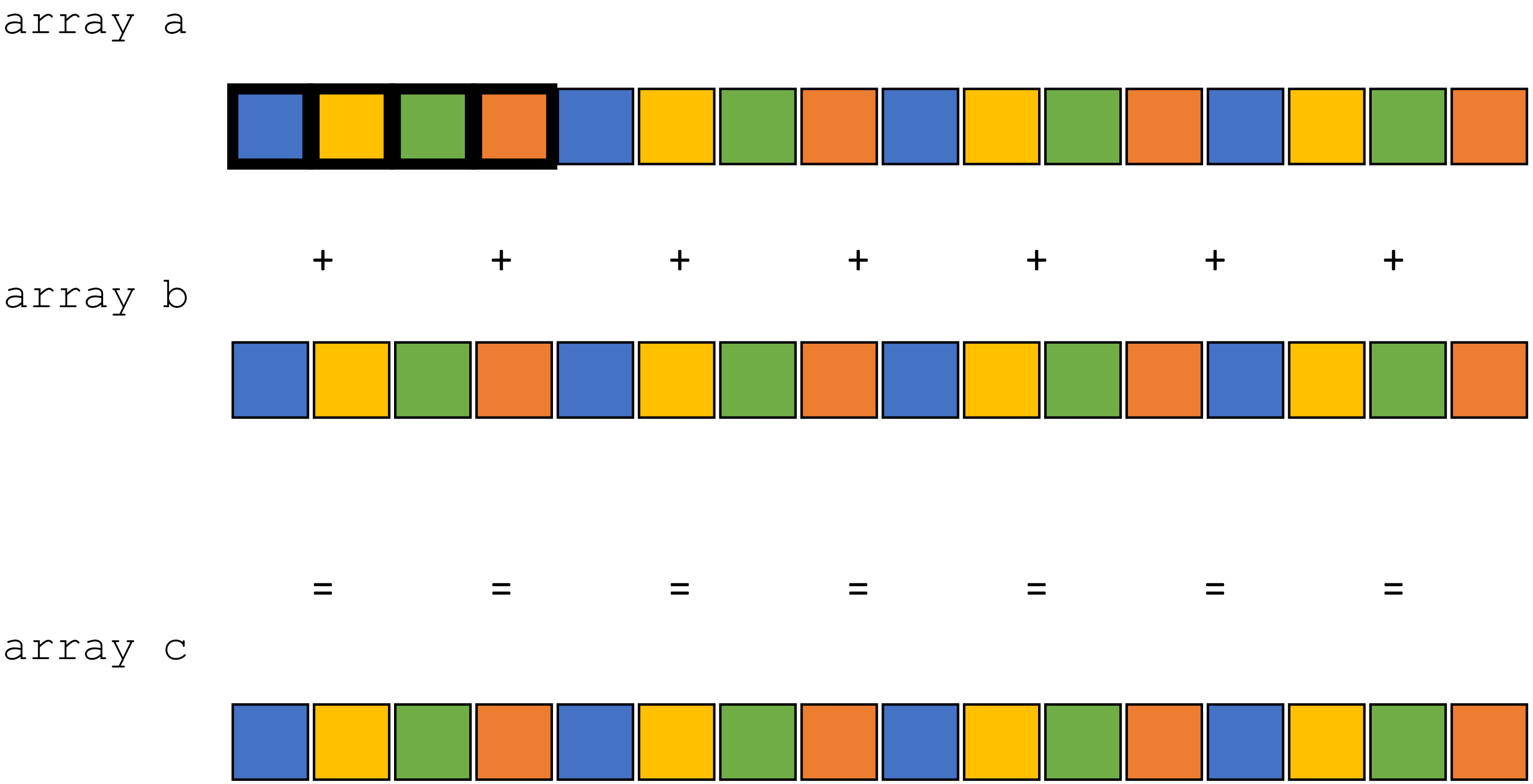
Computation can easily be divided into threads

Thread 0 - Blue
Thread 1 - Yellow
Thread 2 - Green
Thread 3 - Orange



Stride Pattern

What sort of pattern is this?



Computation can easily be divided into threads

Thread 0 - Blue
Thread 1 - Yellow
Thread 2 - Green
Thread 3 - Orange

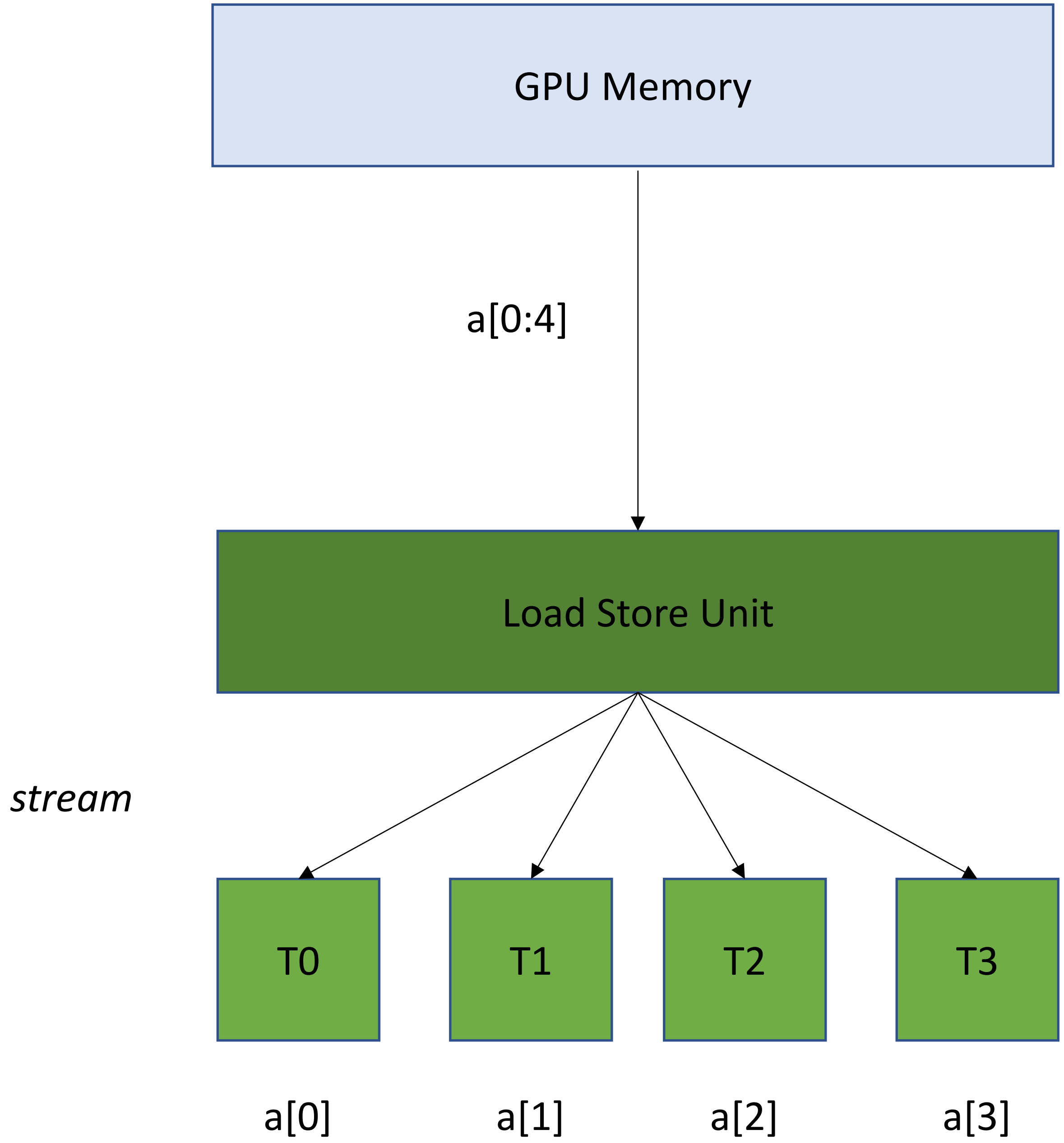
4 cores are accessing memory. What can happen

Read contiguous values

Like the CPU cache, the Load/Store Unit reads in memory in chunks. 16 bytes

Can easily distribute the values to the threads

1 request to GPU memory



Previous quiz + Review

What could we observe with the demonstrations made in class about memory accesses on GPUs?

Memory coalescing accelerates execution.

Previous quiz + Review

Why do we need to calculate int i = blockIdx.x * blockDim.x + threadIdx.x?

○ To get the index in the matrix we want to compute in that thread

○ To get the whole matrix we want to compute in that thread

○ To get the index in the matrix we want to compute in that warp

Previous quiz + Review

Why do we need to calculate int i = blockIdx.x * blockDim.x + threadIdx.x?

-> ○ To get the index in the matrix we want to compute in that thread

○ To get the whole matrix we want to compute in that thread

○ To get the index in the matrix we want to compute in that warp

Homework 4 - first look

- Prerequisits
    - Google Chrome
    - Should be stable on Linux, Windows and Mac.

- No docker container for this assignment.

Homework 4 - first look

- Javascript shared array buffer:
  - How javascript threads can actually share memory
  - Similar to memory in C++

```
var buffer = new SharedArrayBuffer(Float32Array.BYTES_PER_ELEMENT * NUM);
var array = new Float32Array(buffer);
```

Shared Array Buffer

- Like Malloc, allocates a ”pointer” to a contagious array of bytes
- Can pass the “pointer” to different threads
- Need to instantiate a typed array to access the values

Web Workers

- How to do multi-threading in javascript

- Async
  - Concurrent (executes on the same thread)
  - Good for I/O and user interactions

- Web Workers will execute on multiple cores
  - Better for compute intensive applications
  - Better performance

How to use?

- Create a new worker with a file
  - Doesn’t do anything yet
- File contains a function: “on message”
- Main file calls “post message” along with arguments to start the thread
- Worker sends a message back to the main file, it can catch the data

WebGPU

- The language is wgsl (WebGPU Shading Language)
  - It is new, there are not many examples (and the specification changes!)
  - Official specification is here: https://www.w3.org/TR/WGSL/

WebGPU

- wgsl is NOT javascript

- Javascript is interpreted: not possible on GPUs

- wgsl is compiled
  - into Vulkan on Linux
  - into Metal on Apple
  - into HLSL on Windows

- No printing (so GPU code can be difficult to debug)

WebGPU

- variables (optional types):

var <name> = <value>;

```
var cluster_dist = 3.0;
```

```
var <name> : <type> = <value>;
var cluster_dist : f32 = 3.0;
```

WebGPU

```
struct Particle {
    pos : vec2<f32>;
};
```

- types:
  - i32
  - u32
  - f32
  - vec2<f32>
  - array<*type*>

```
struct Particles {
    particles : array<Particle>;
};
```

```
var index_pos : vec2<f32> = particlesA.particles[index].pos;
```

- structures

```
var index : u32 = GlobalInvocationID.x;
```

- Built-ins (global id)

you have one thread for each particle!

WebGPU

- Built in functions:
    - arrayLength
    - sqrt
    - pow
    - distance

WebGPU

For loops:

```
for (var i : u32 = 0u; i < arrayLength(&particlesA.particles); i = i + 1u) {
...
}
```

WebGPU

- Types can be frustrating
- But compiler errors will help you, and you can do casts.

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,*
>
> Linus Torvalds

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);
```

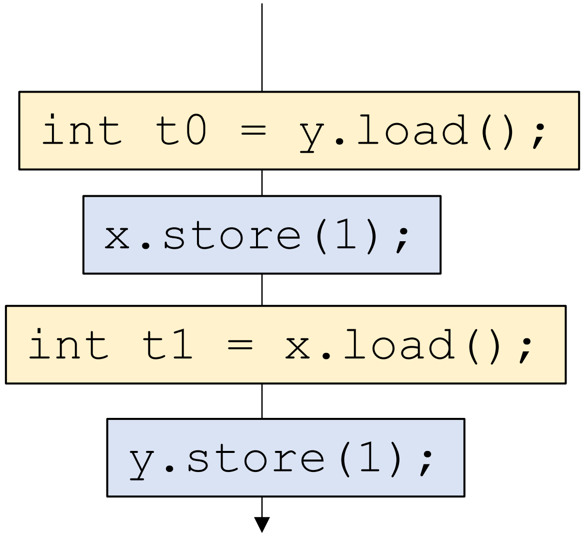
Thread 0:

```
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();
```



```
int t0 = y.load();
x.store(1);
int t1 = x.load();
y.store(1);
```

Global variable:

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

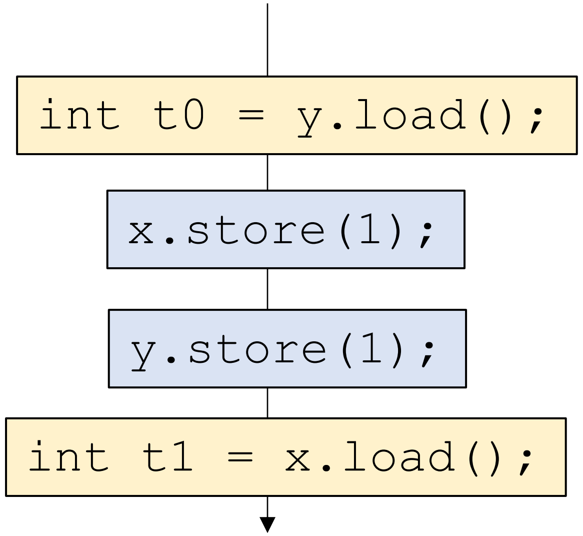
Thread 0:

```
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();
```



```
int t0 = y.load();
x.store(1);
y.store(1);
int t1 = x.load();
```

Global variable:

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

How about:

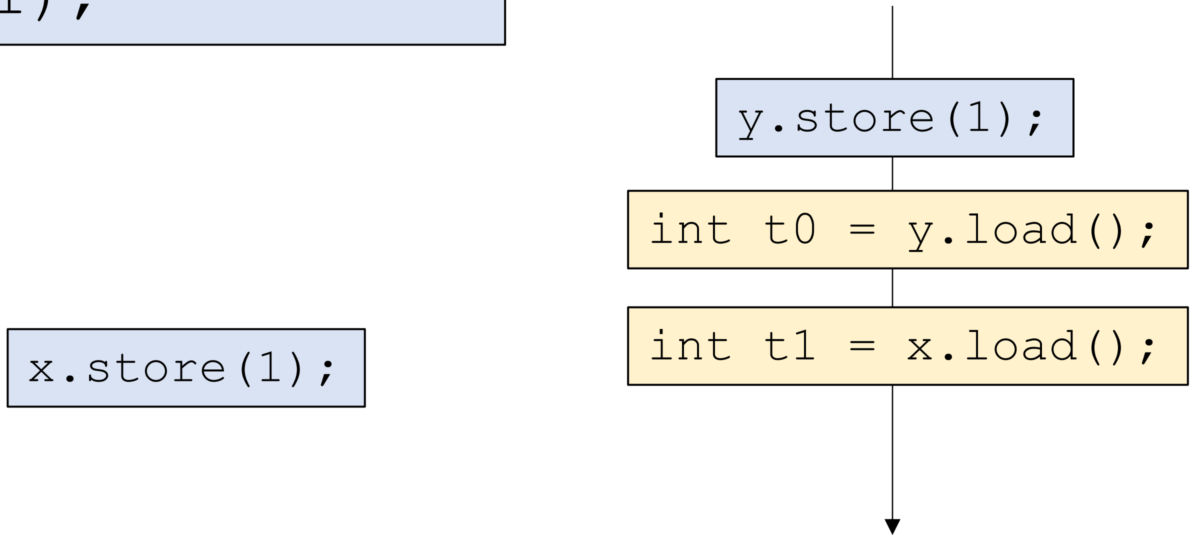
Is it possible for
`t0 == 1 and t1 ==0`

Thread 0:

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

Thread 1:

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



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

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

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

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

Global variable:

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

How about:

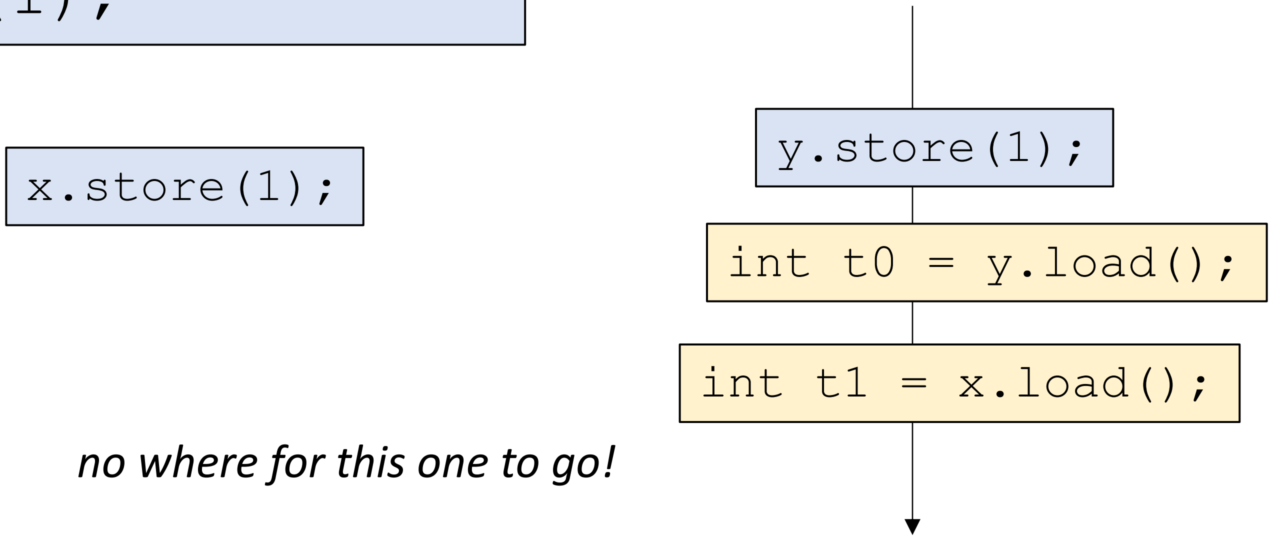
Is it possible for `t0 == 1 and t1 ==0`

Thread 0:

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

Thread 1:

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



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

```
y.store(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

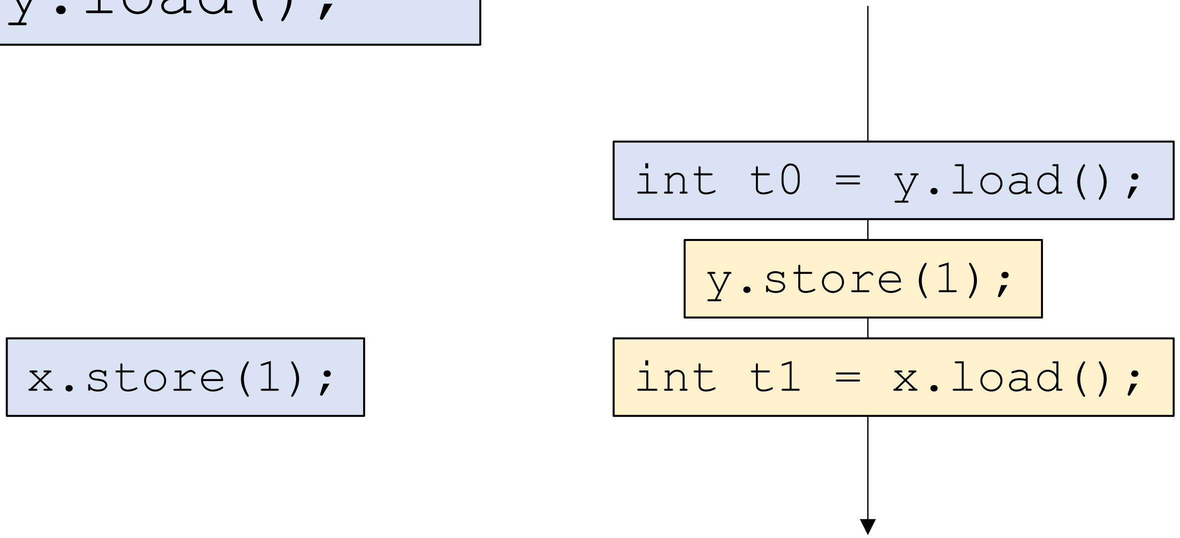
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



```
int t0 = y.load();
y.store(1);
int t1 = x.load();
```

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

Global variable:

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

Another test

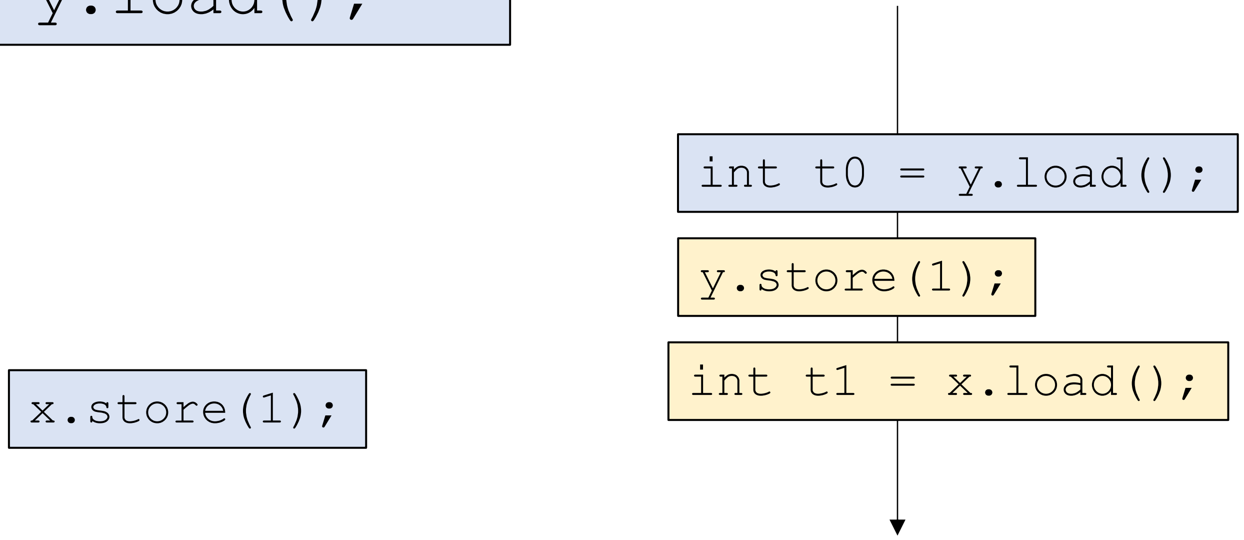
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



```
int t0 = y.load();
y.store(1);
int t1 = x.load();
```

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

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

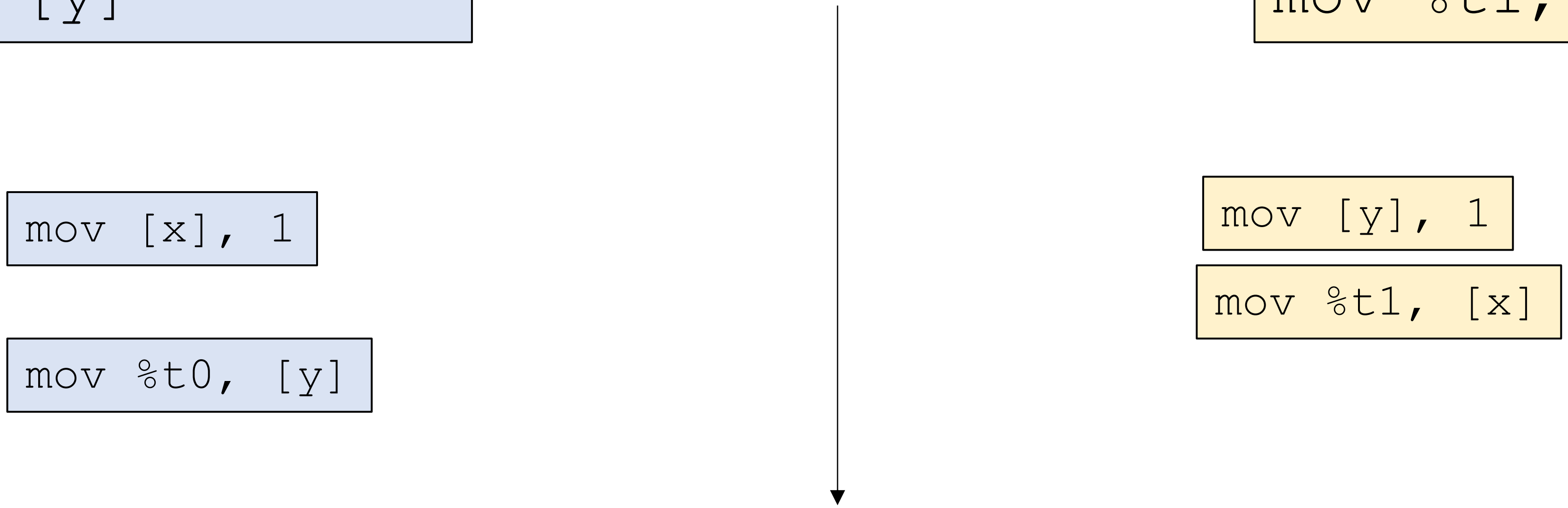
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



`mov [x], 1`

`mov %t0, [y]`

`mov [y], 1`

`mov %t1, [x]`

Global variable:

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

Another test

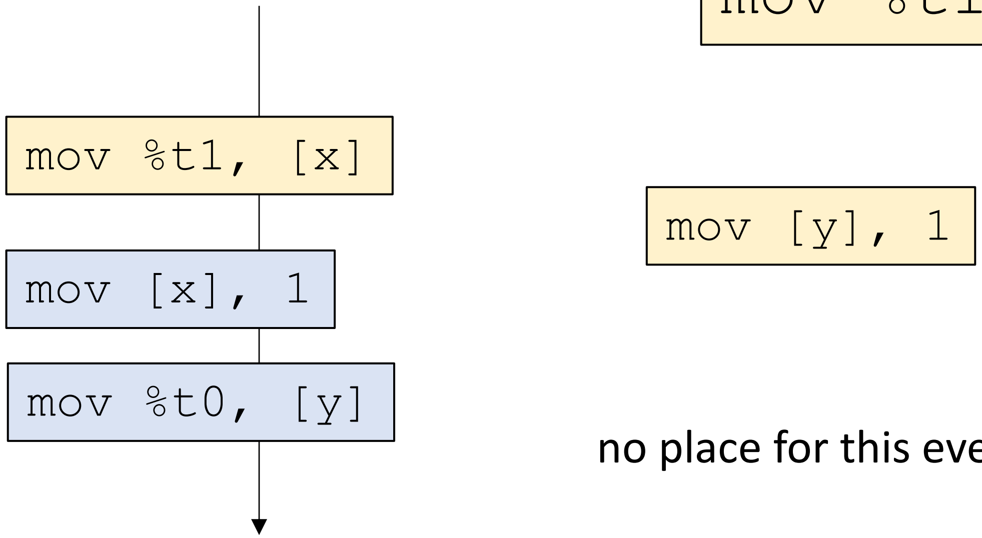
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



`mov %t1, [x]`

`mov [x], 1`

`mov %t0, [y]`

`mov [y], 1`

no place for this event!

ISA is not SC

- We’d like to be able to compile atomic instructions just to regular ISA loads and stores
- What if we actually run this code?

Schedule

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

Thread 0:

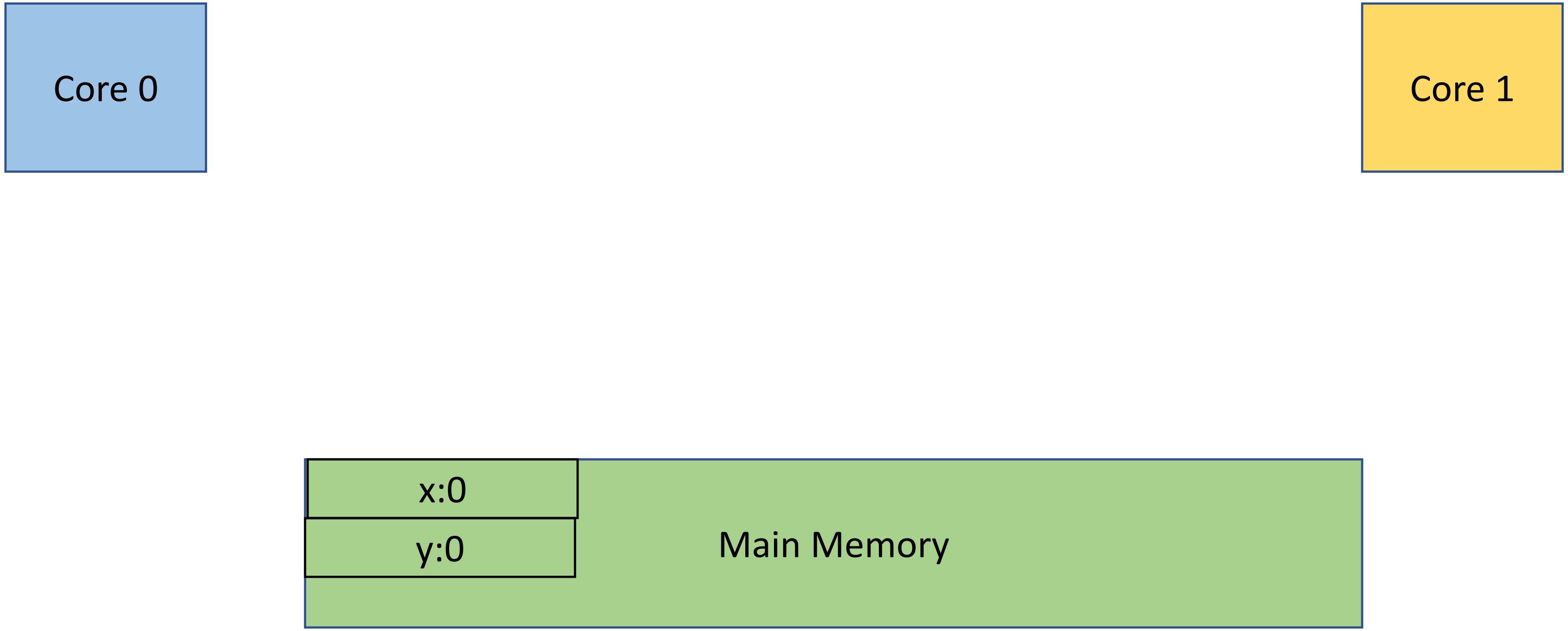
```
mov [x], 1
```

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

Thread 1:

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

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



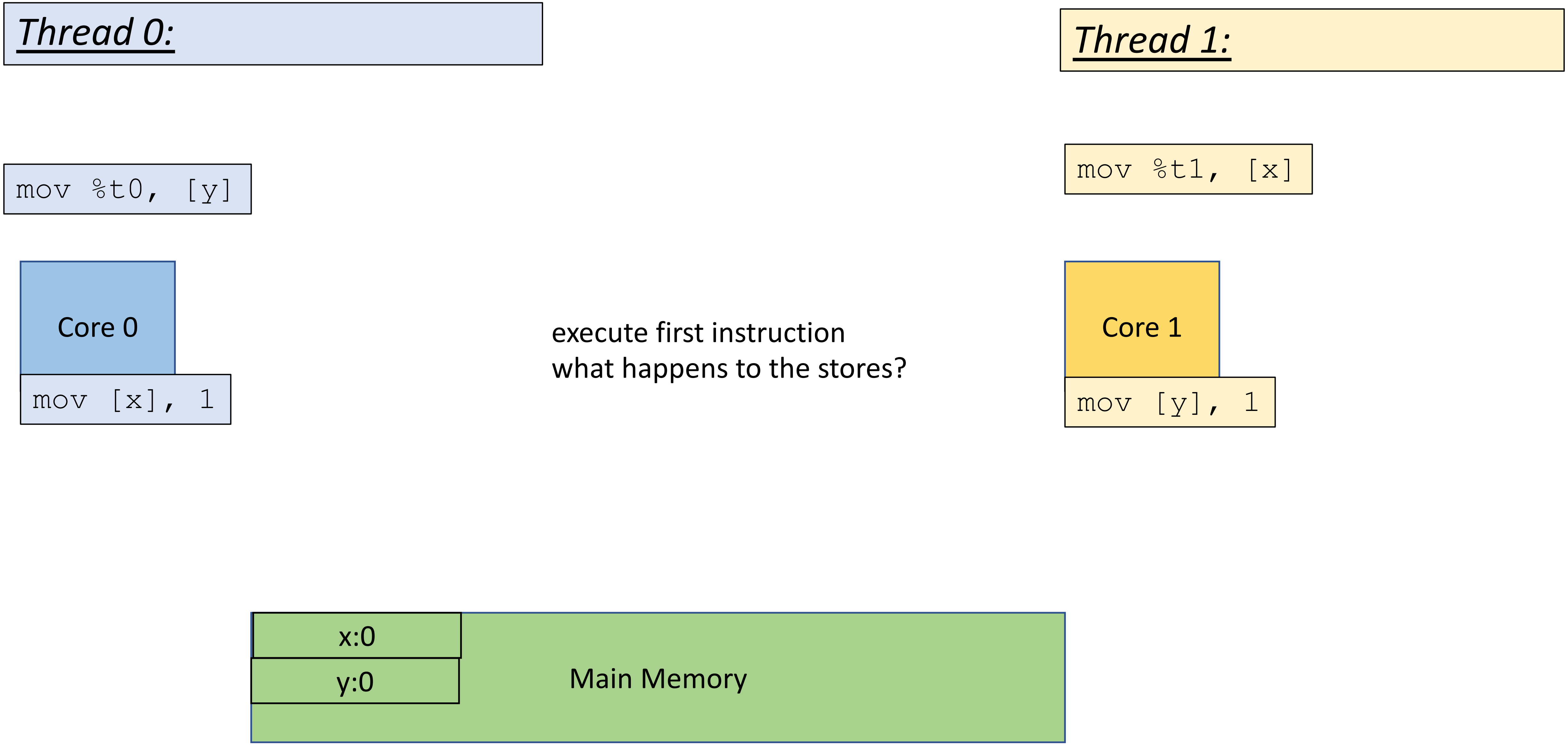
Core 0

Core 1

x:0

y:0

Main Memory



Thread 0:

`mov %t0, [y]`

Core 0

`mov [x], 1`

Thread 1:

`mov %t1, [x]`

Core 1

`mov [y], 1`

execute first instruction
what happens to the stores?

x:0
y:0

Main Memory

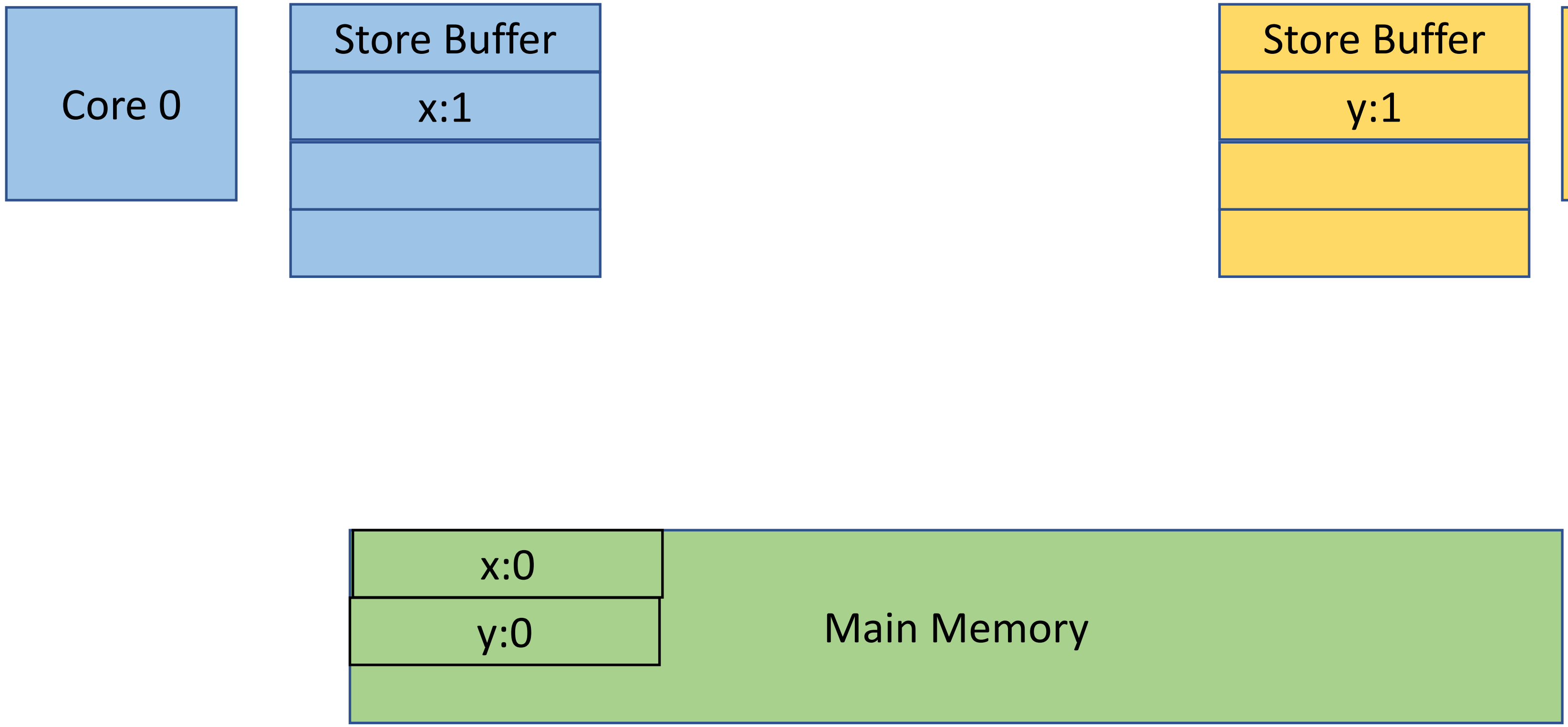
Thread 0:

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

Thread 1:

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

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



Core 0

Store Buffer

x:1

Store Buffer

y:1

Core 1

x:0

y:0

Main Memory

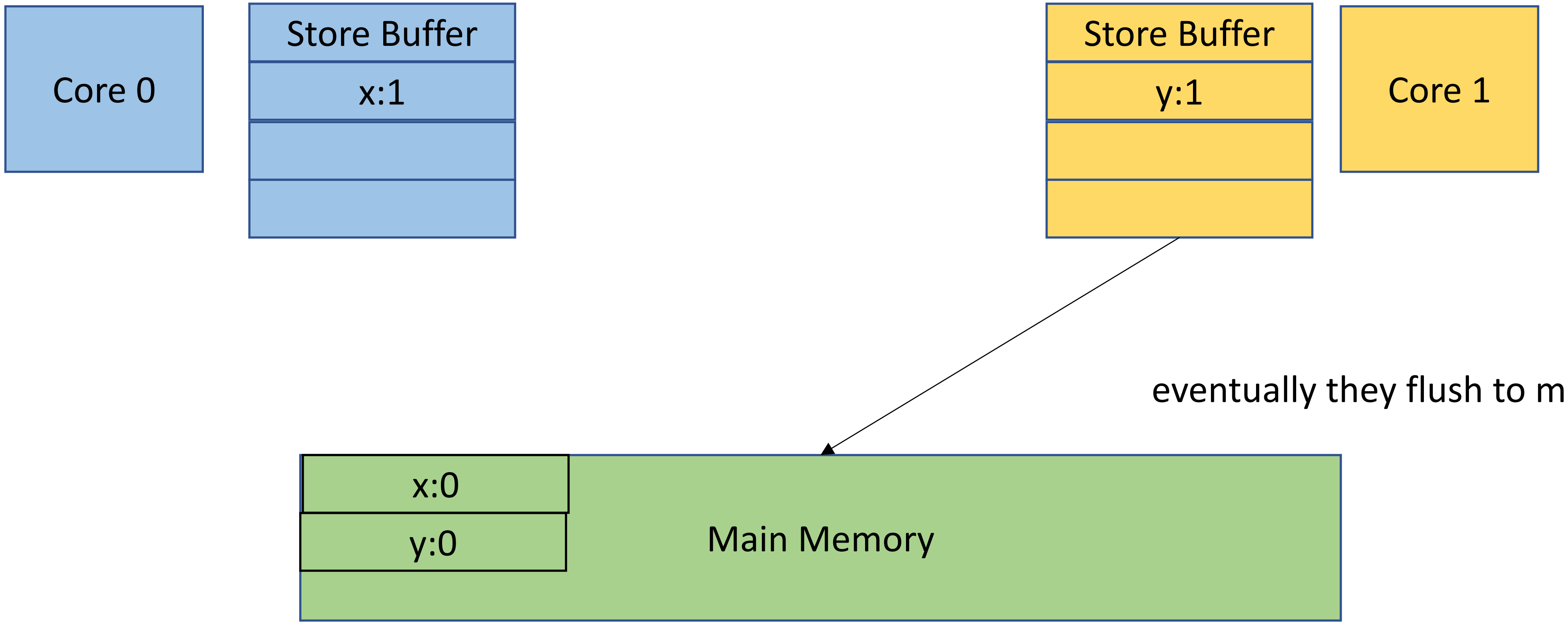
Thread 0:

`mov %t0, [y]`

Thread 1:

`mov %t1, [x]`

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



eventually they flush to main memory

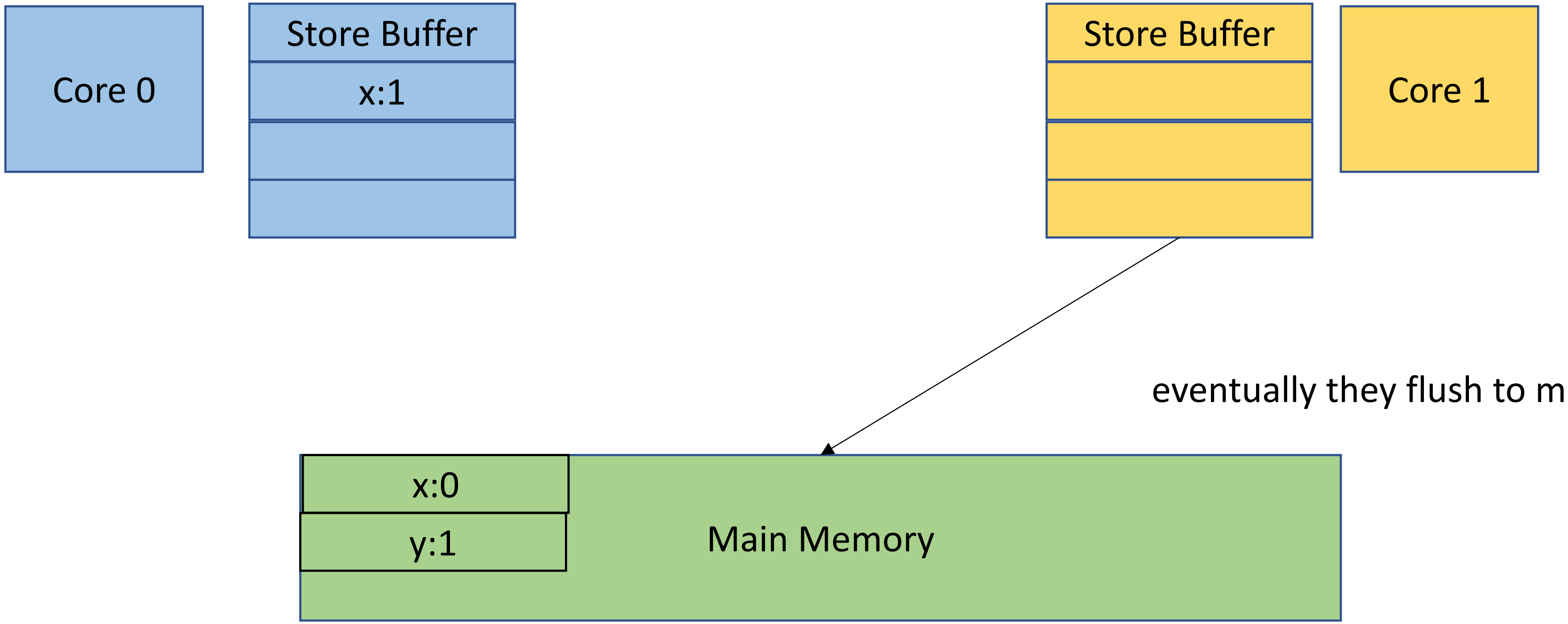
Thread 0:

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

Thread 1:

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

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



eventually they flush to main memory

Thread 0:

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

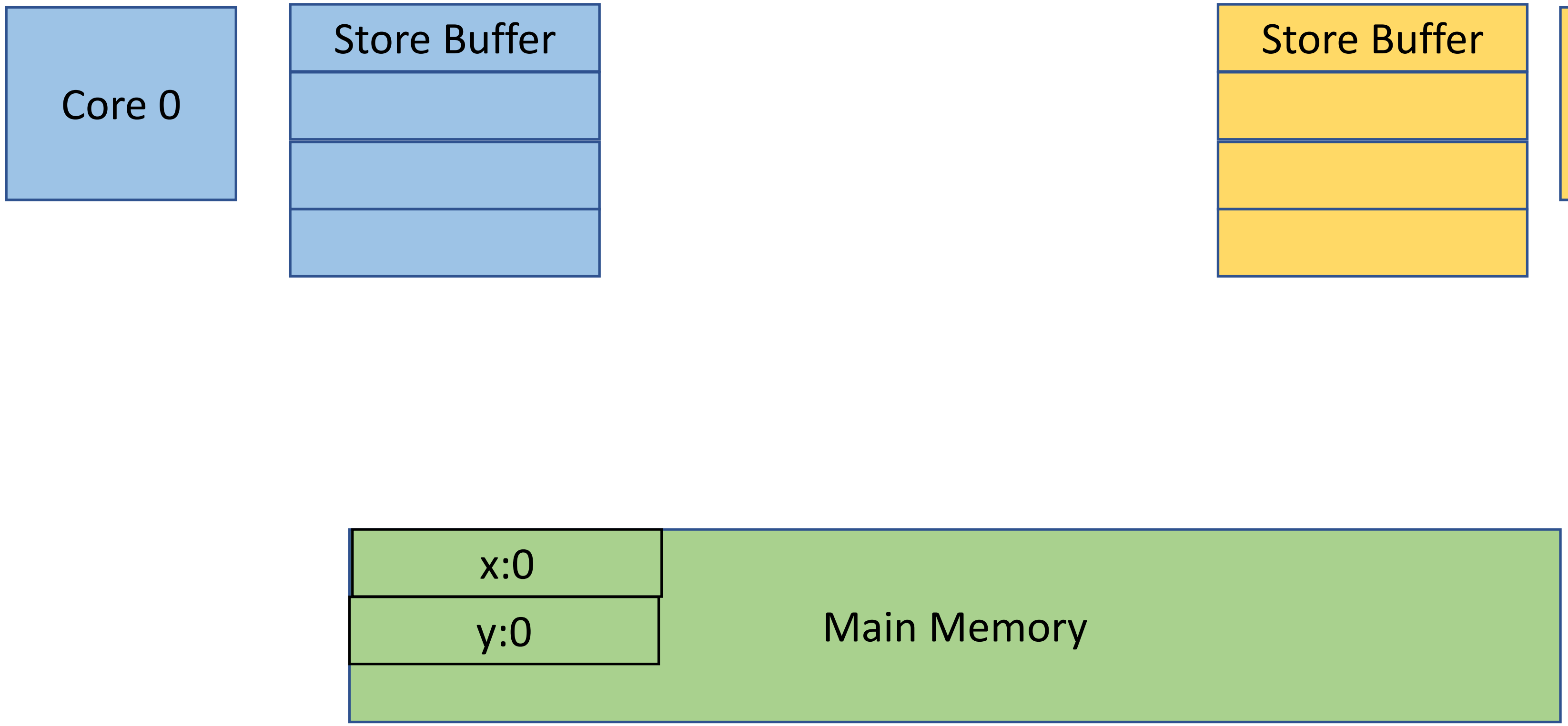
```
mov %t0, [y]
```

Thread 1:

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

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

rewind



Core 0

Store Buffer

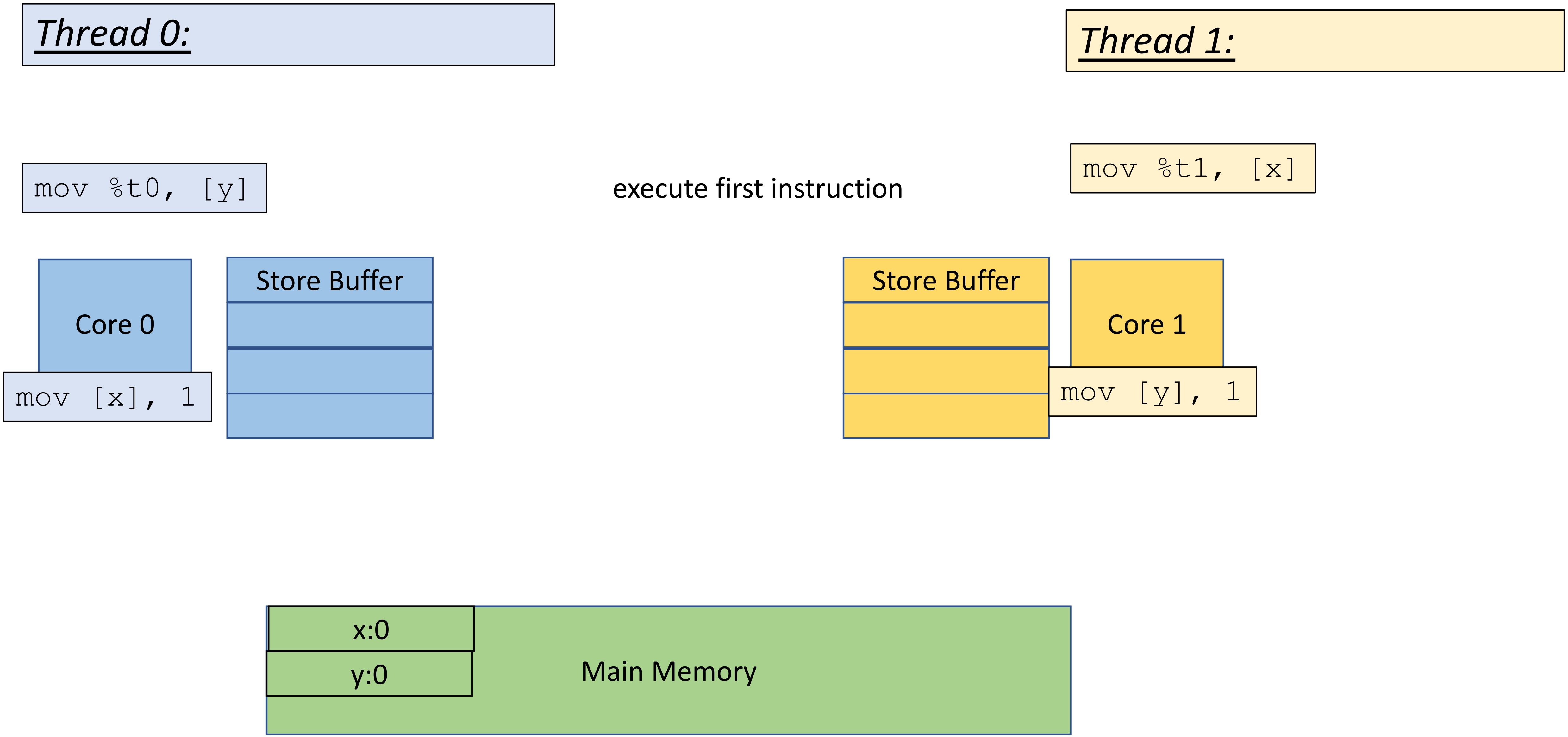
Store Buffer

Core 1

x:0

y:0

Main Memory



Thread 0:

`mov %t0, [y]`

Core 0

`mov [x], 1`

Store Buffer

execute first instruction

Thread 1:

`mov %t1, [x]`

Store Buffer

Core 1

`mov [y], 1`

x:0

y:0

Main Memory

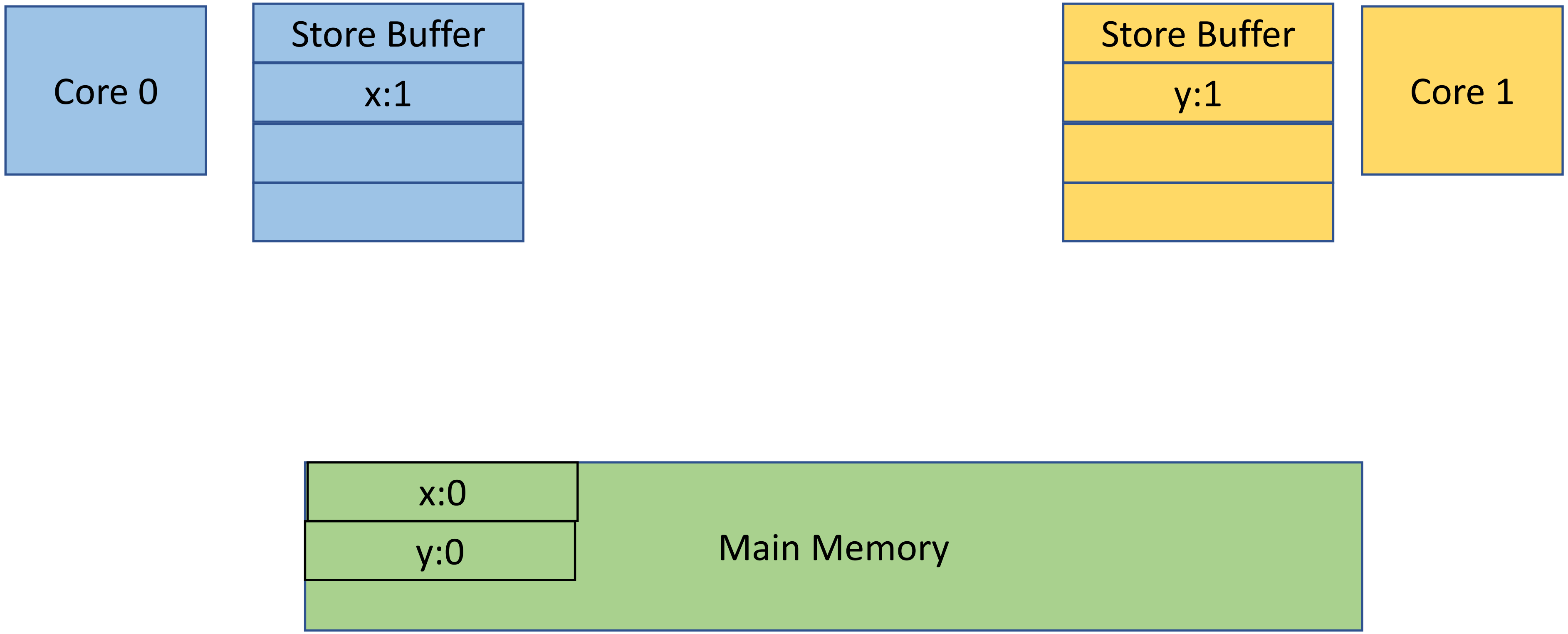
Thread 0:

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

values get stored in SB

Thread 1:

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



Core 0

Store Buffer

x:1

Store Buffer

y:1

Core 1

x:0

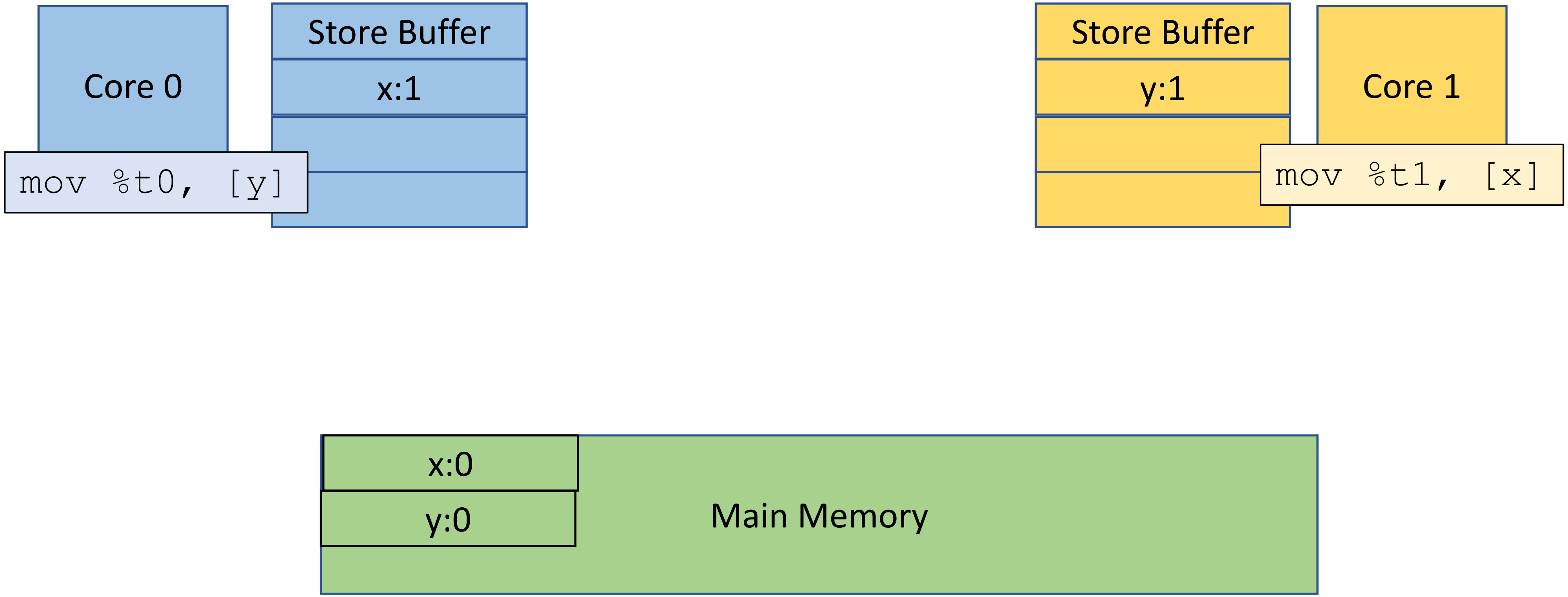
y:0

Main Memory

Thread 0:

Thread 1:

Execute next instruction



Core 0

`mov %t0, [y]`

Store Buffer

x:1

Store Buffer

y:1

Core 1

`mov %t1, [x]`

x:0

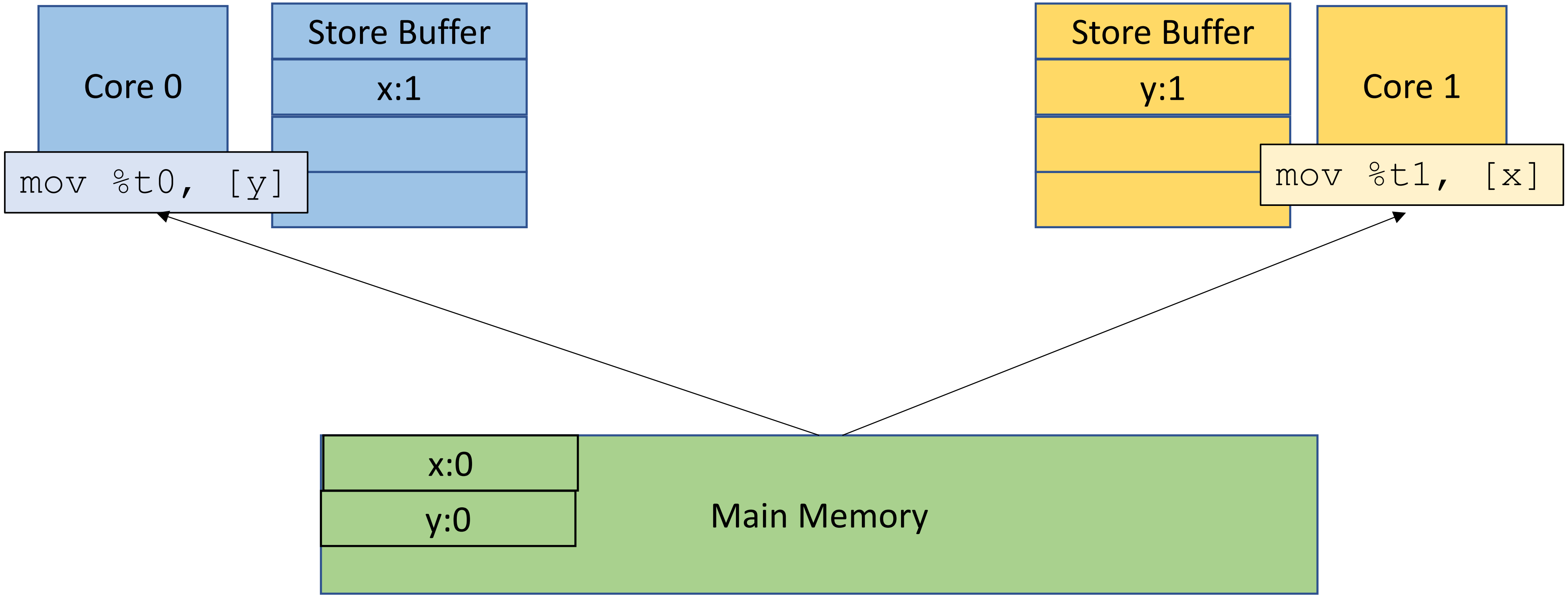
y:0

Main Memory

Thread 0:

Thread 1:

Values get loaded from memory



Core 0

Store Buffer

x:1

`mov %t0, [y]`

Store Buffer

y:1

Core 1

`mov %t1, [x]`

x:0

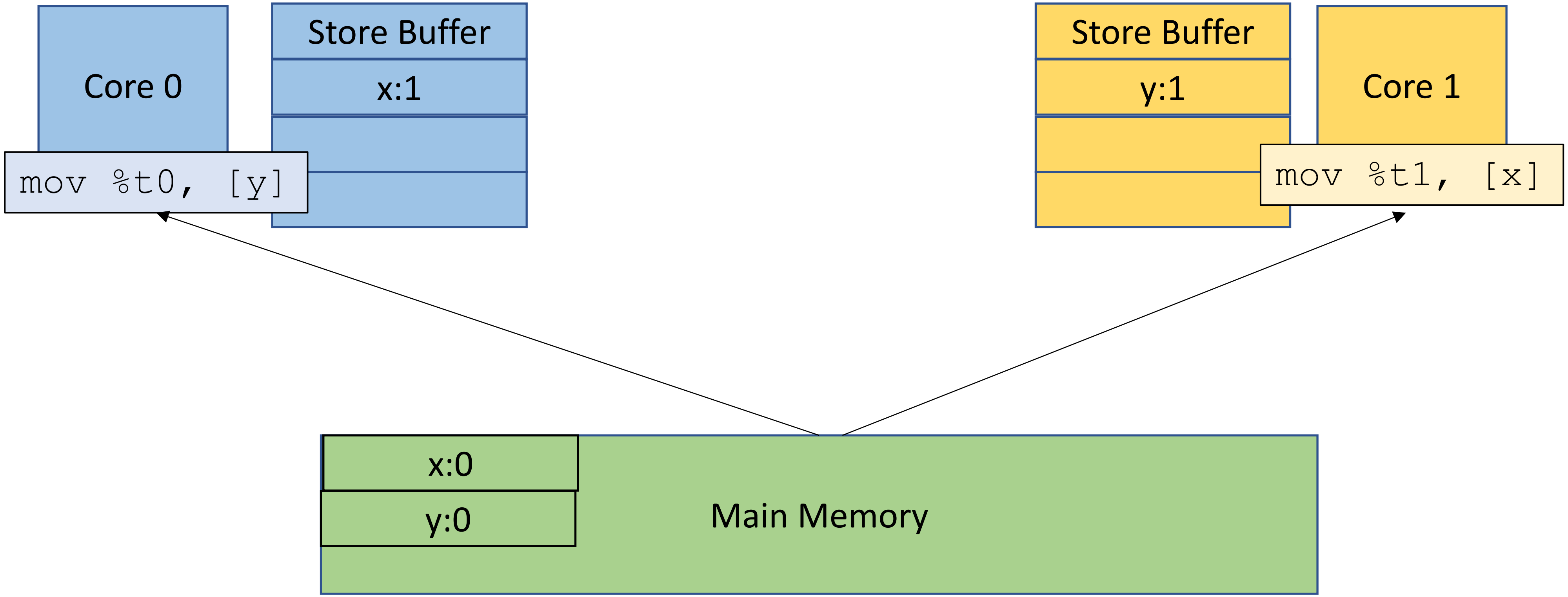
y:0

Main Memory

Thread 0:

Thread 1:

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



Core 0

Store Buffer

x:1

`mov %t0, [y]`

Store Buffer

y:1

Core 1

`mov %t1, [x]`

x:0

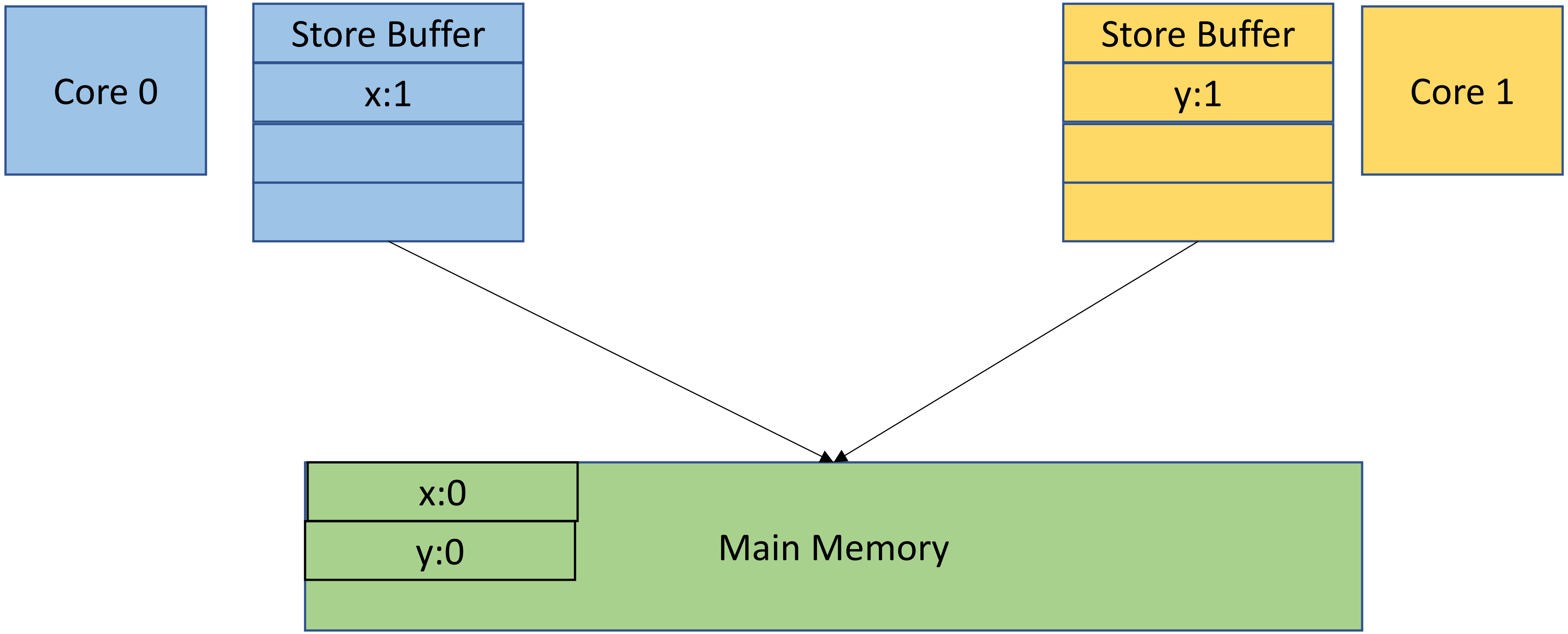
y:0

Main Memory

Thread 0:

Thread 1:

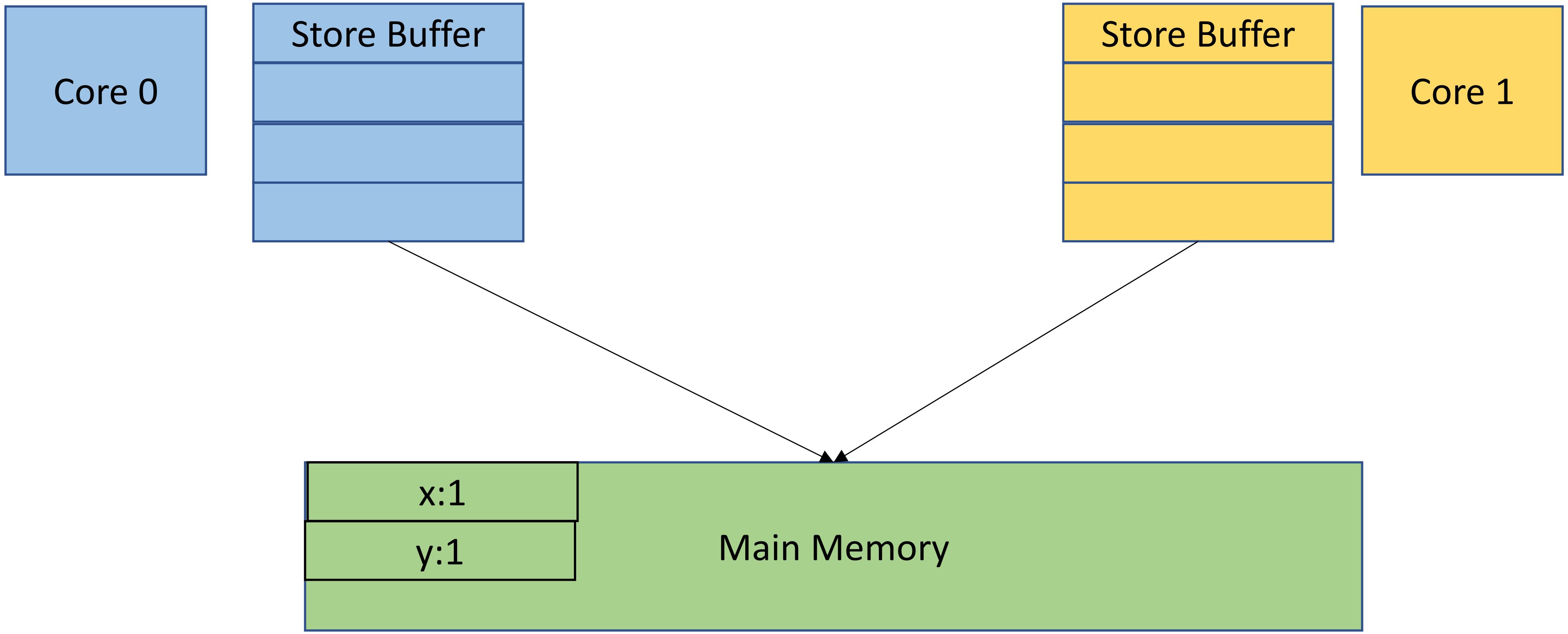
Store buffers are drained eventually



Thread 0:

Thread 1:

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



Core 0

Store Buffer

Store Buffer

Core 1

x:1

y:1

Main Memory

Our first relaxed memory execution!

- **Relaxed memory** (also known as **Weak memory)** behaviors
- An execution that is **NOT allowed by sequential consistency**
- **Relaxed memory model**: a memory model that allows relaxed memory executions
  - **X86** has a relaxed memory model due to store buffering
  - If you restrict yourself to use only default atomic operations, **C++** 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
    - X86 behaviors were documented by researchers before Intel!
    - Many vendors have unofficially endorsed academic models

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`?

Fences: restoring sequential consistency

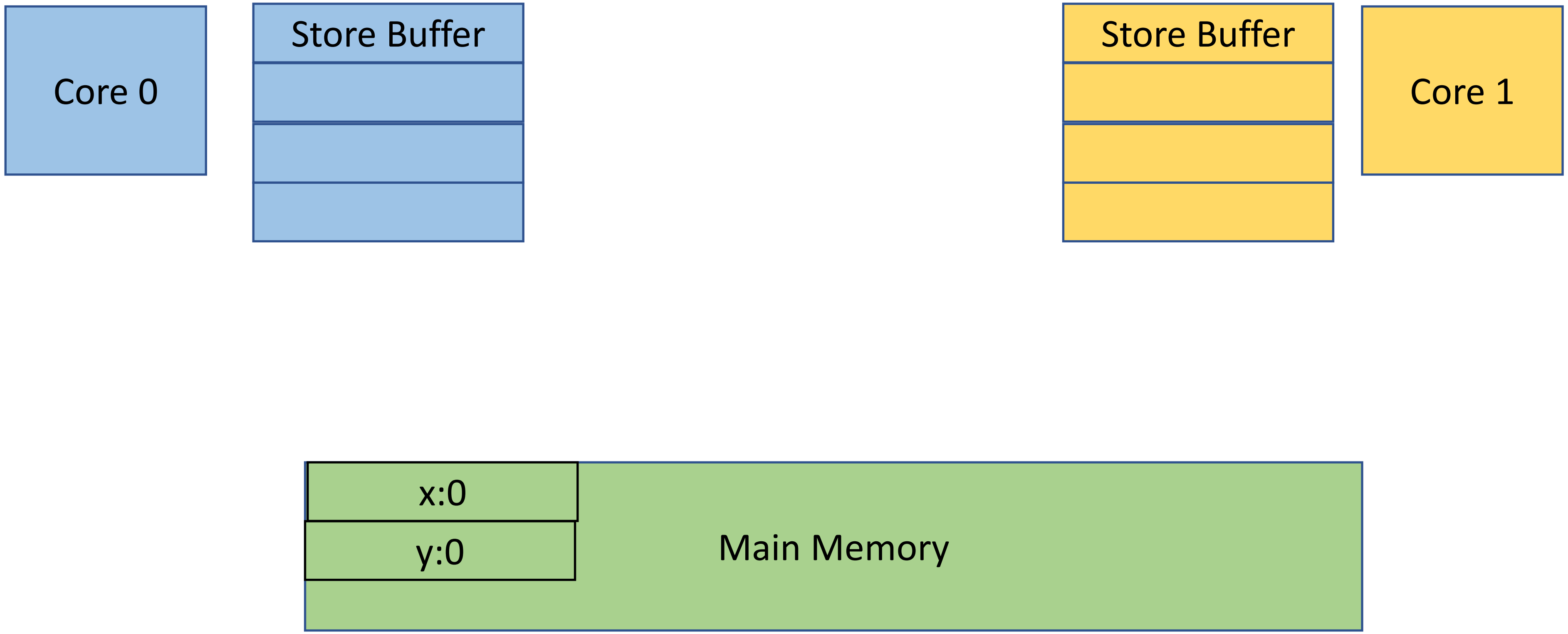
- It is typical that relaxed memory models provide special instructions which can be used to disallow weak behaviors.
- These instructions are called Fences
- The X86 fence that flushes the store buffer is called `mfence`.

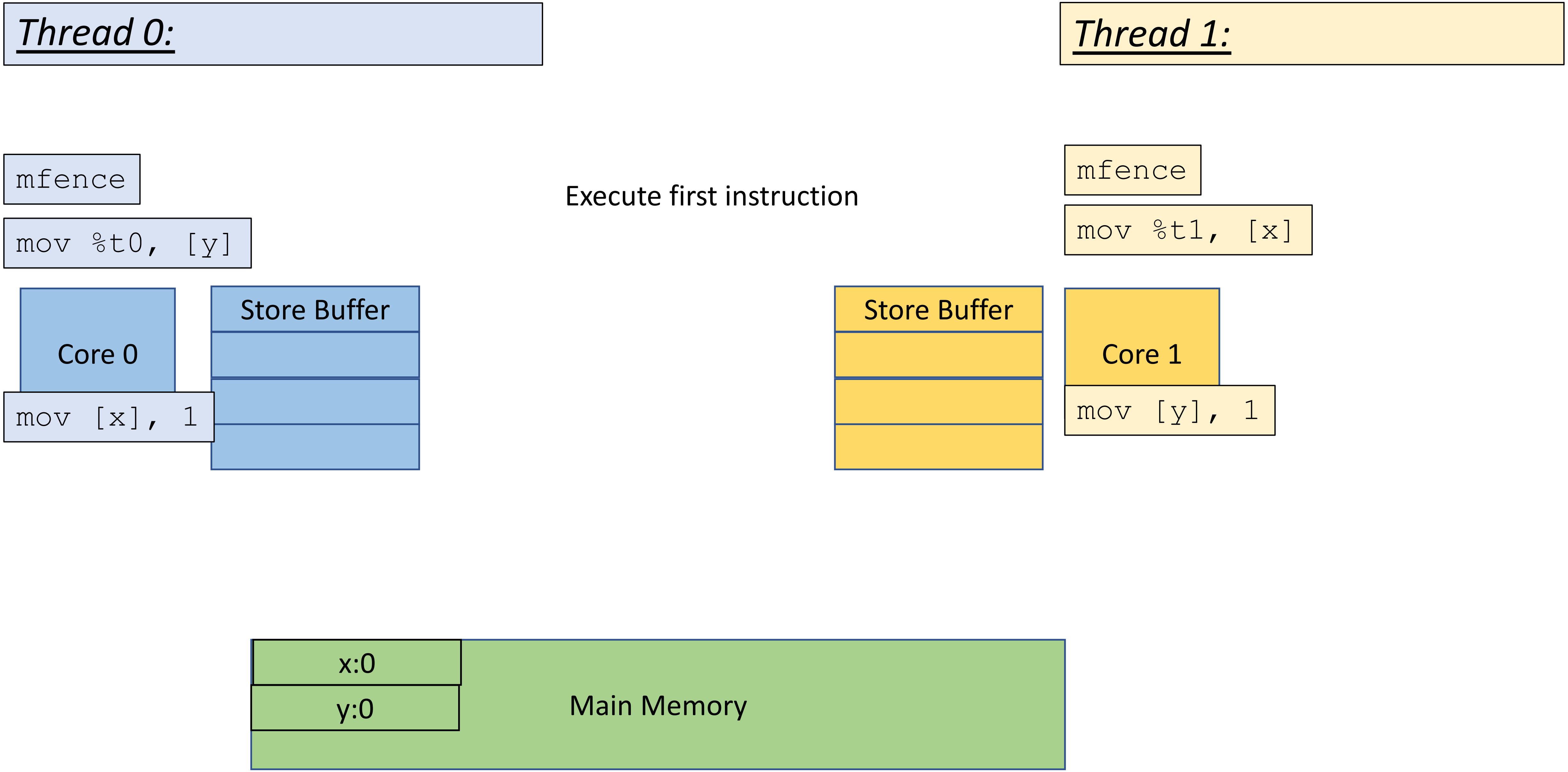
Thread 0:

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

Thread 1:

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





Thread 0:

`mfence`

`mov %t0, [y]`

Core 0

`mov [x], 1`

Store Buffer

Execute first instruction

Thread 1:

`mfence`

`mov %t1, [x]`

Store Buffer

Core 1

`mov [y], 1`

x:0

y:0

Main Memory

Thread 0:

`mfence`

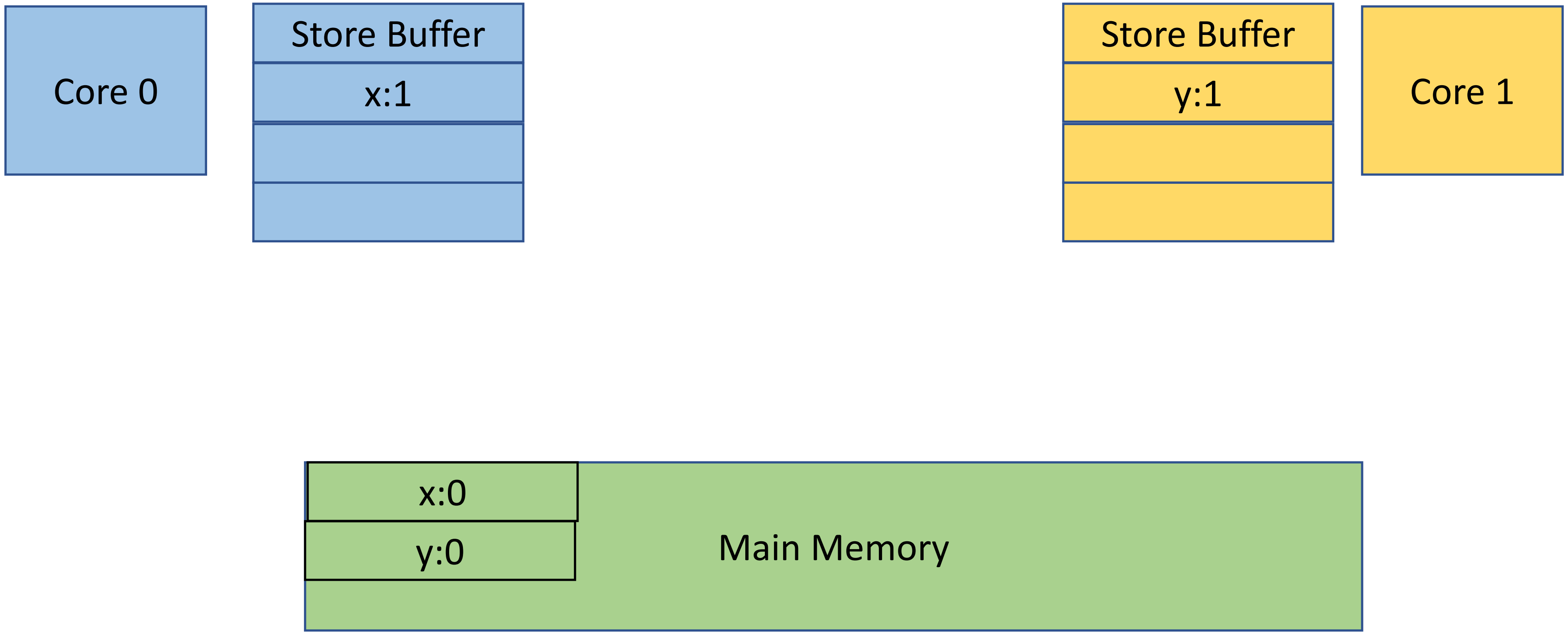
`mov %t0, [y]`

Values go into the store buffer

Thread 1:

`mfence`

`mov %t1, [x]`



Core 0

Store Buffer

x:1

Store Buffer

y:1

Core 1

x:0

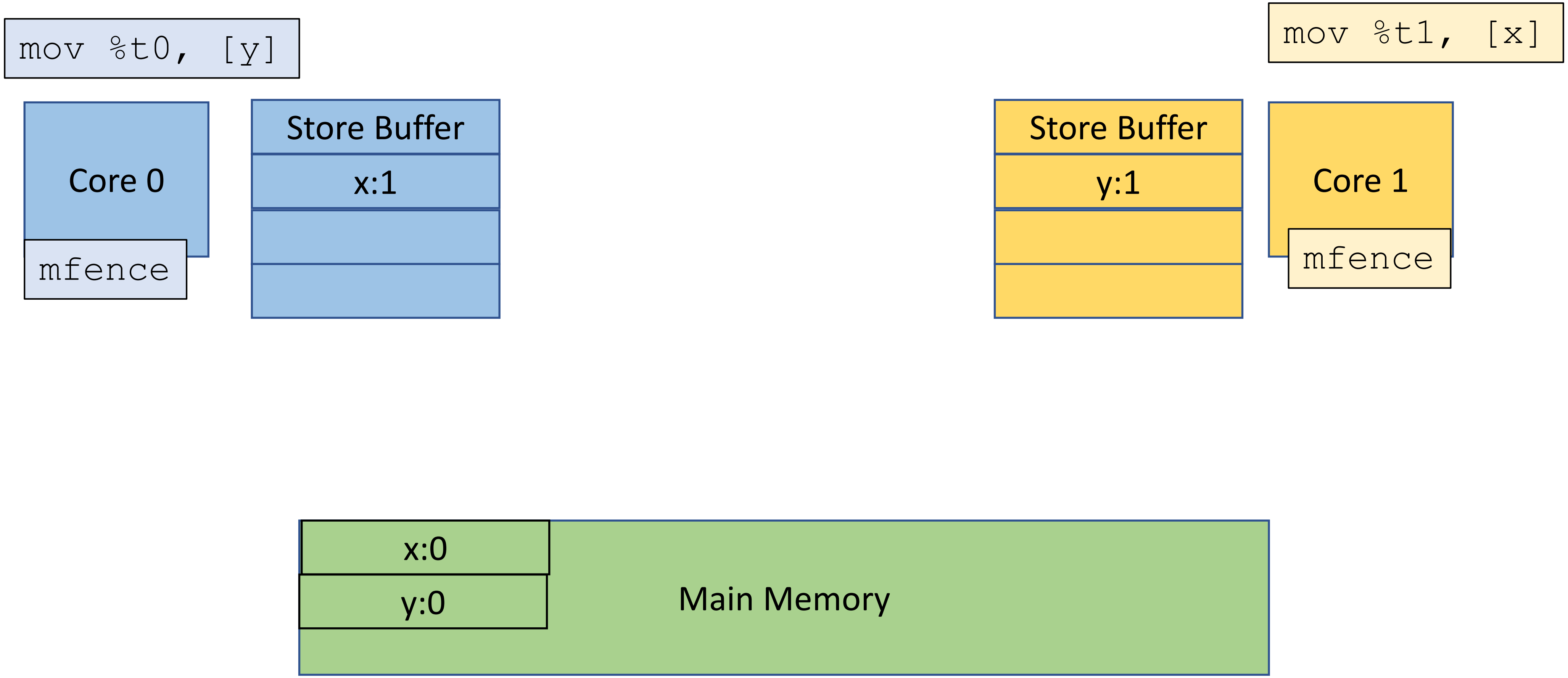
y:0

Main Memory

Thread 0:

Thread 1:

Execute next instruction



`mov %t0, [y]`

Core 0

`mfence`

Store Buffer

x:1

Store Buffer

y:1

`mov %t1, [x]`

Core 1

`mfence`

x:0

y:0

Main Memory

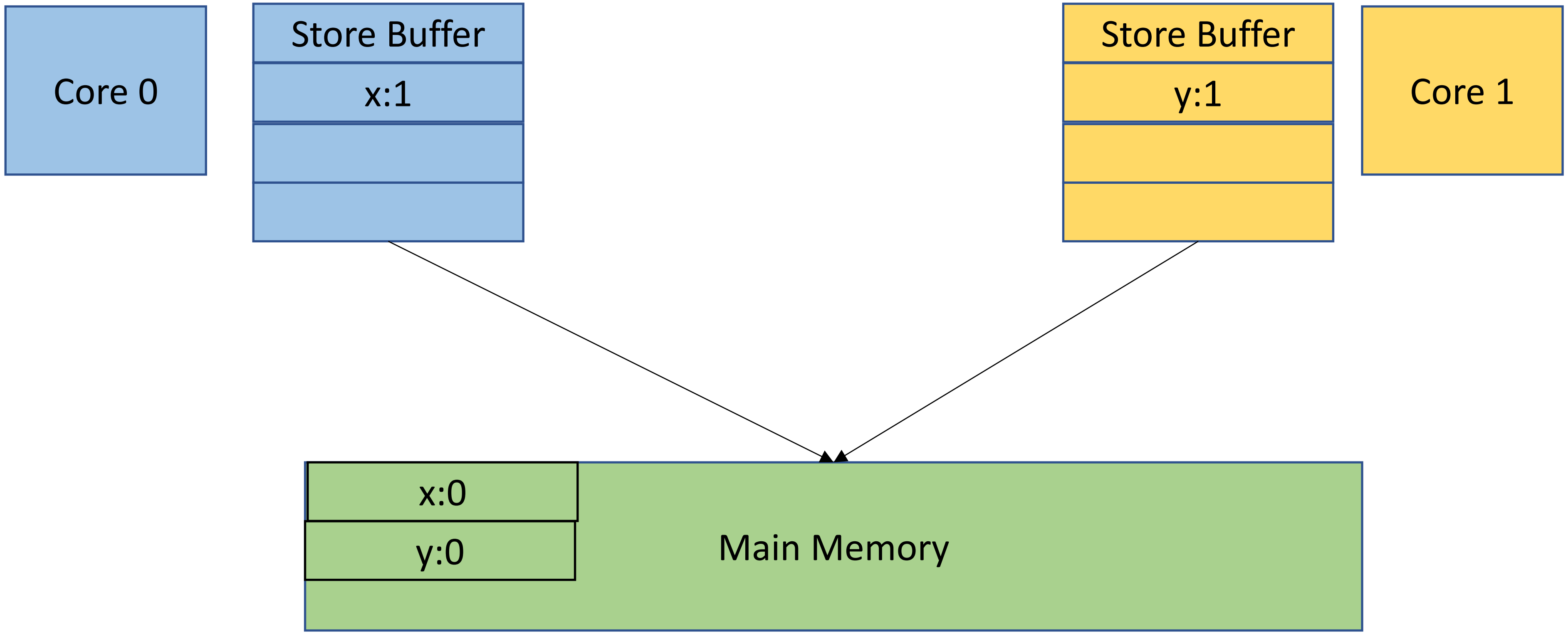
Thread 0:

Thread 1:

store buffers are flushed

`mov %t0, [y]`

`mov %t1, [x]`



Core 0

Store Buffer

x:1

Store Buffer

y:1

Core 1

x:0

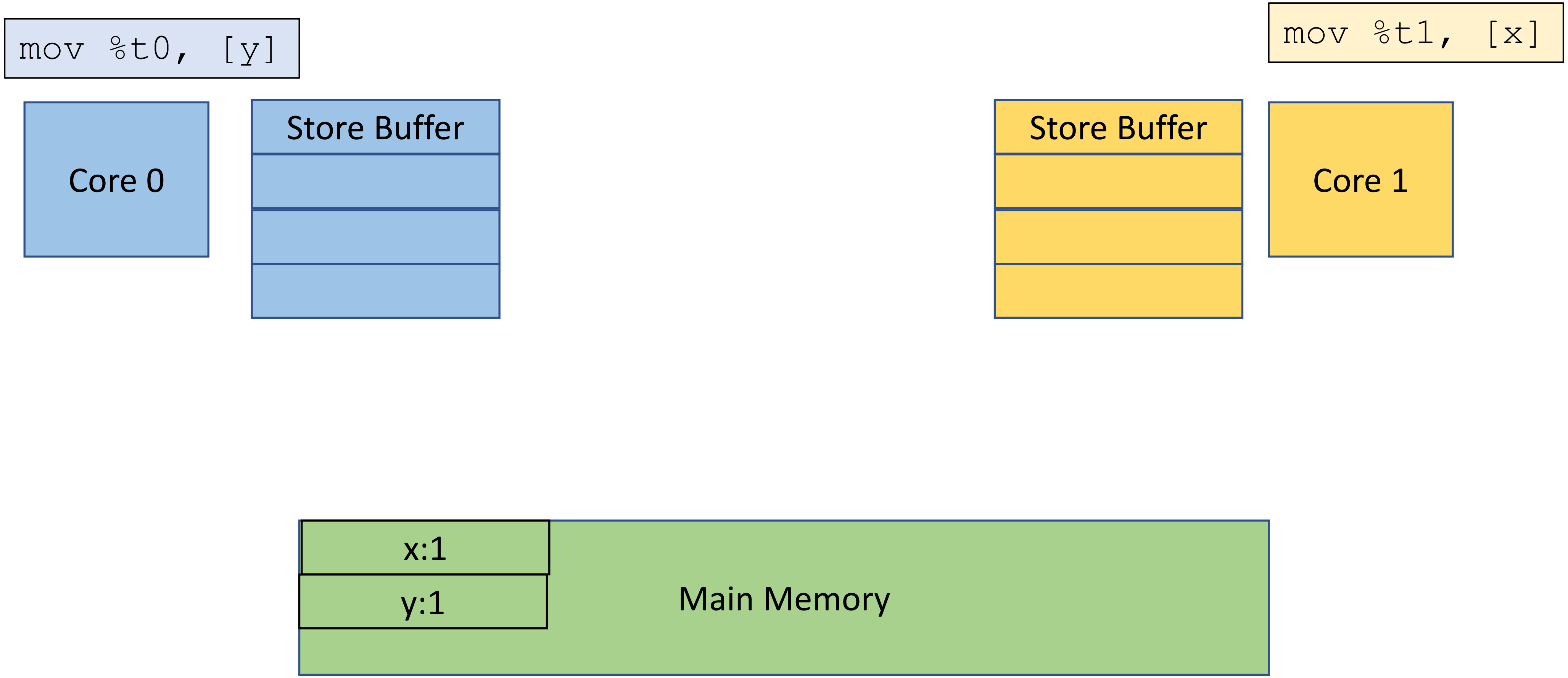
y:0

Main Memory

Thread 0:

Thread 1:

store buffers are flushed



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

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

Core 0

Store Buffer

Store Buffer

Core 1

x:1

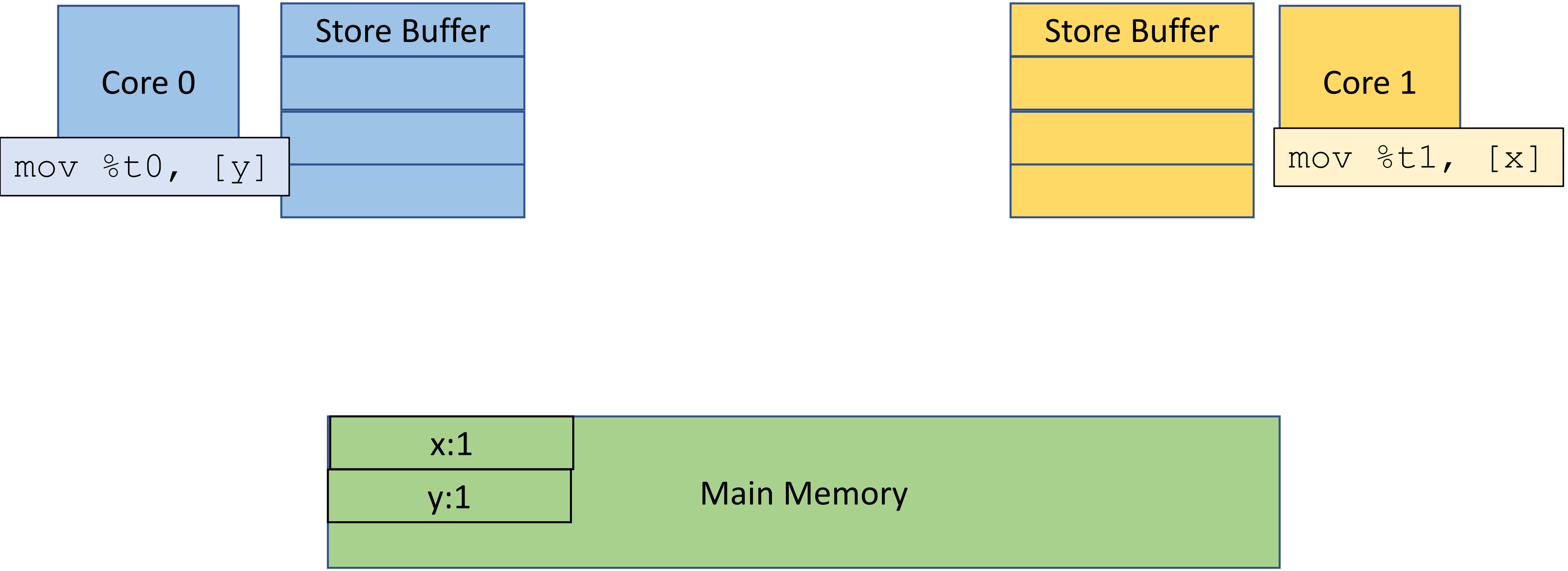
y:1

Main Memory

Thread 0:

Thread 1:

execute next instruction



Core 0

`mov %t0, [y]`

Store Buffer

Store Buffer

Core 1

`mov %t1, [x]`

x:1

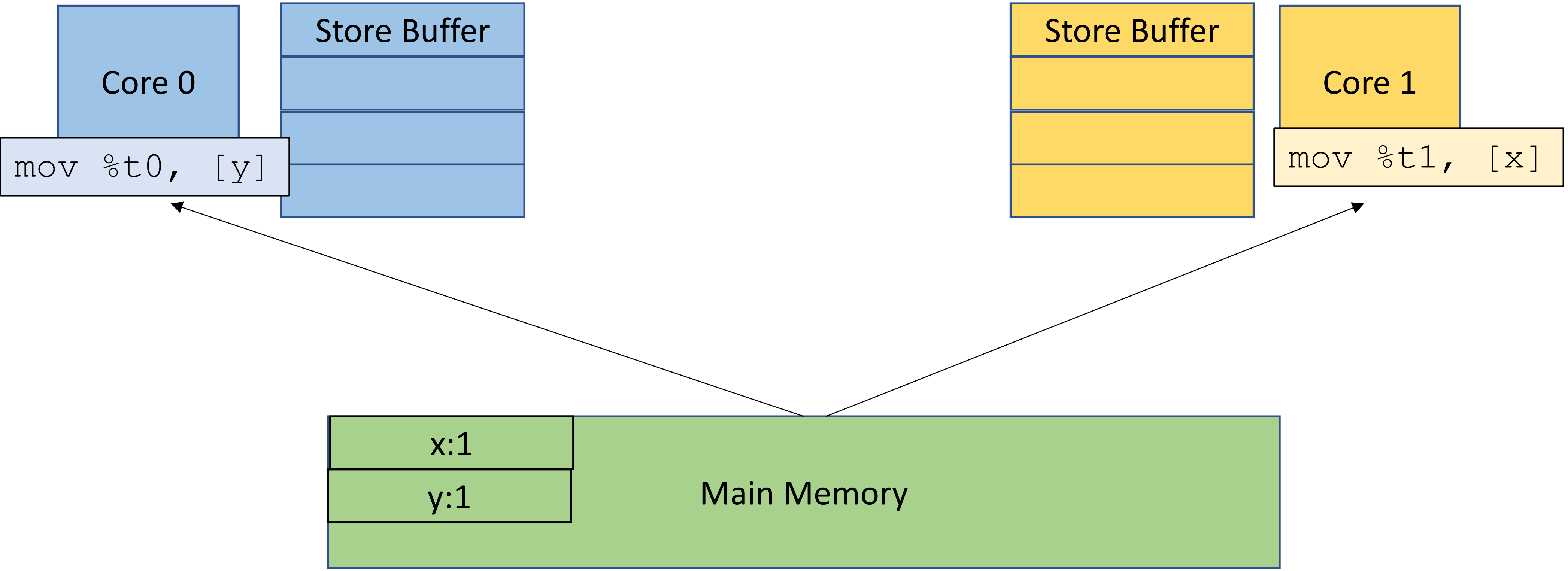
y:1

Main Memory

Thread 0:

Thread 1:

values are loaded from memory



Core 0

Store Buffer

`mov %t0, [y]`

Store Buffer

Core 1

`mov %t1, [x]`

x:1

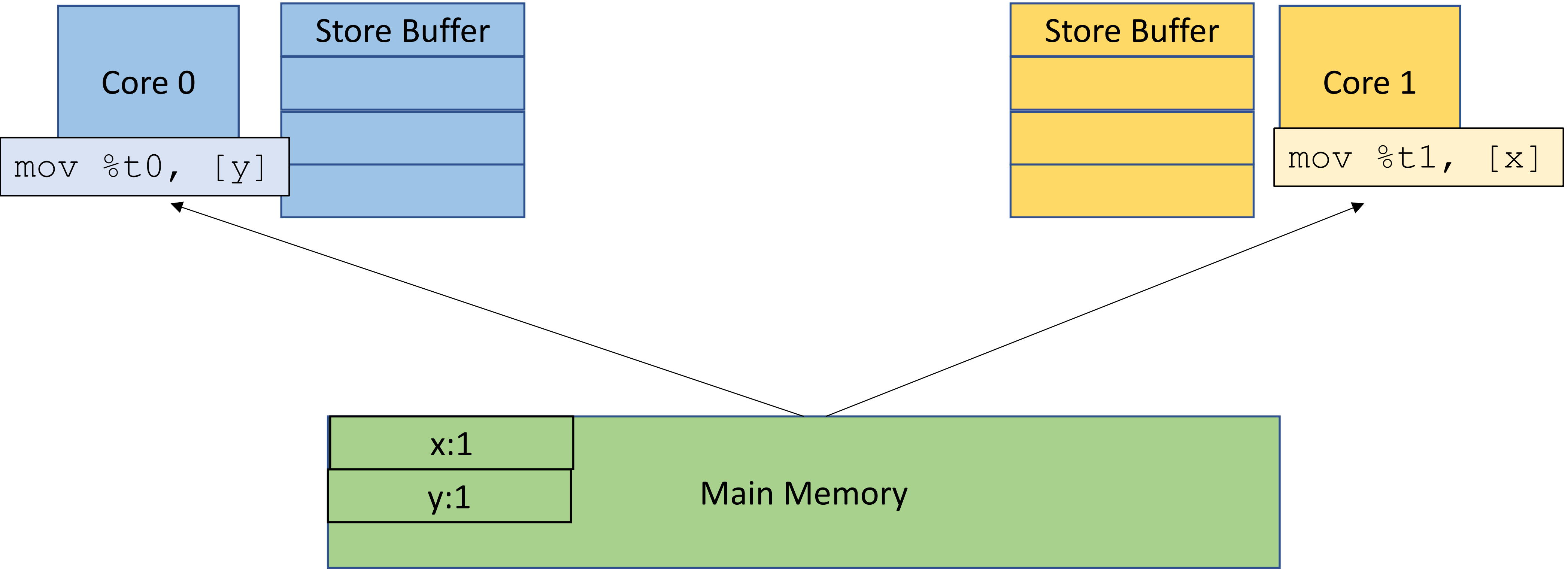
y:1

Main Memory

Thread 0:

Thread 1:

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



Core 0

`mov %t0, [y]`

Store Buffer

Store Buffer

Core 1

`mov %t1, [x]`

x:1

y:1

Main Memory

Thread 0:

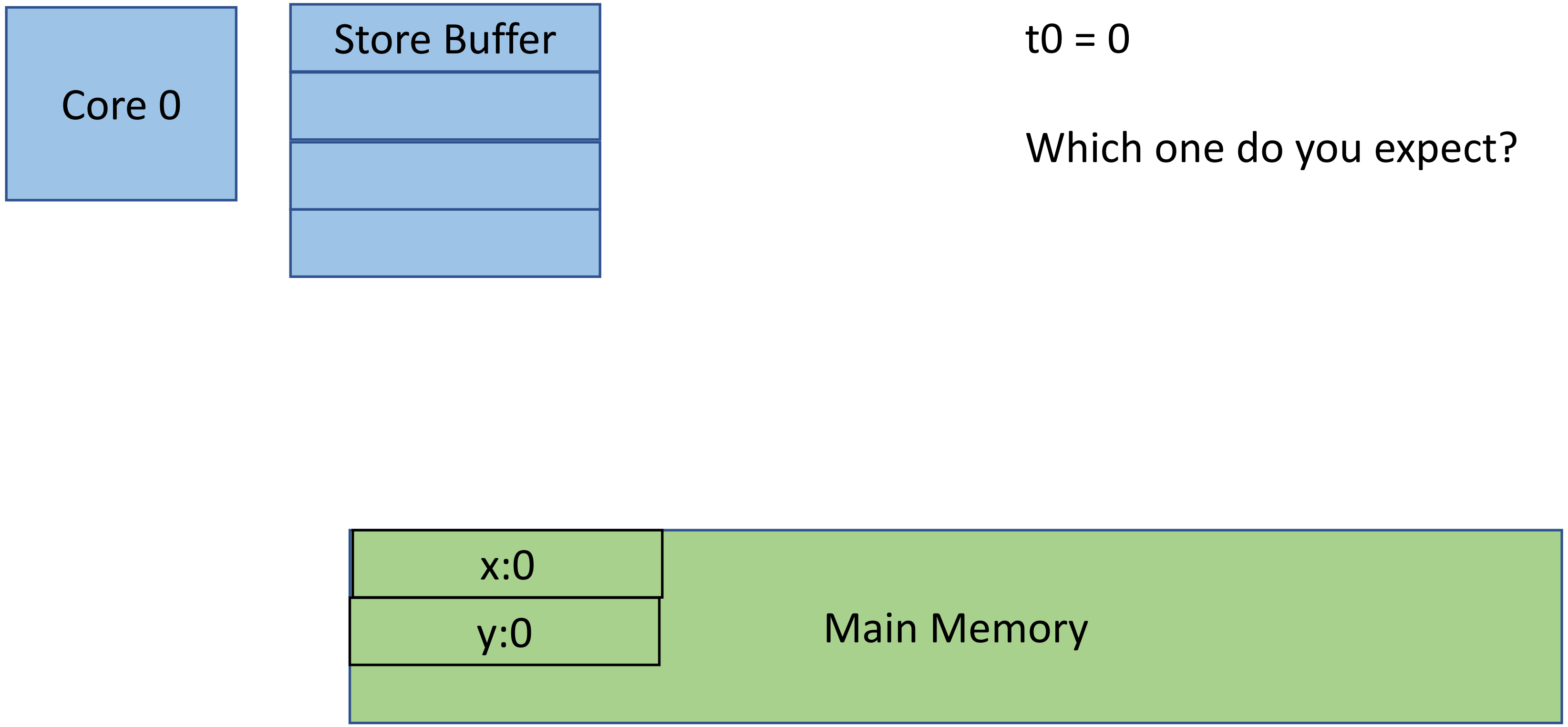
```
mov [x], 1
```

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

single thread
same address

possible outcomes:
t0 = 1
t0 = 0

Which one do you expect?

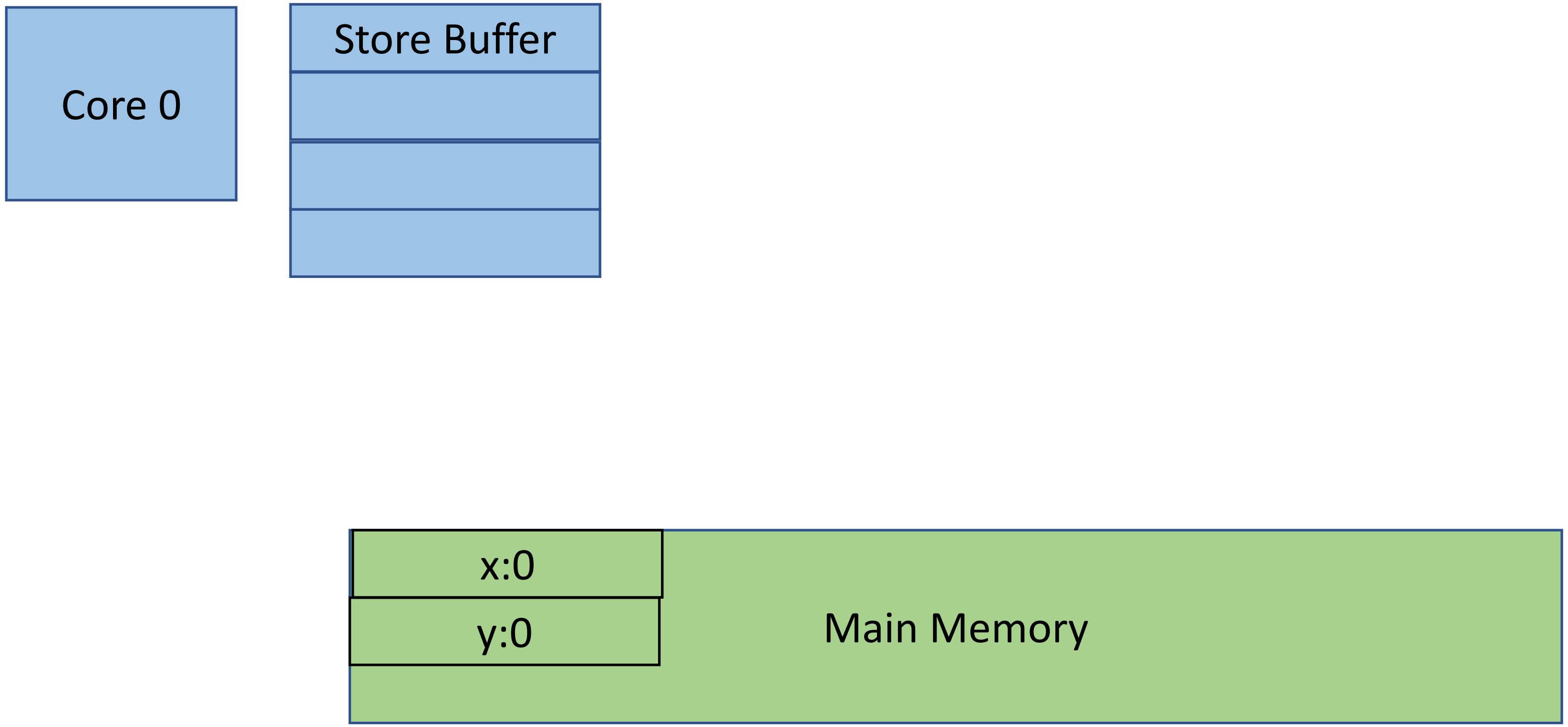


Thread 0:

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

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

How does this execute?



Core 0

Store Buffer

x:0

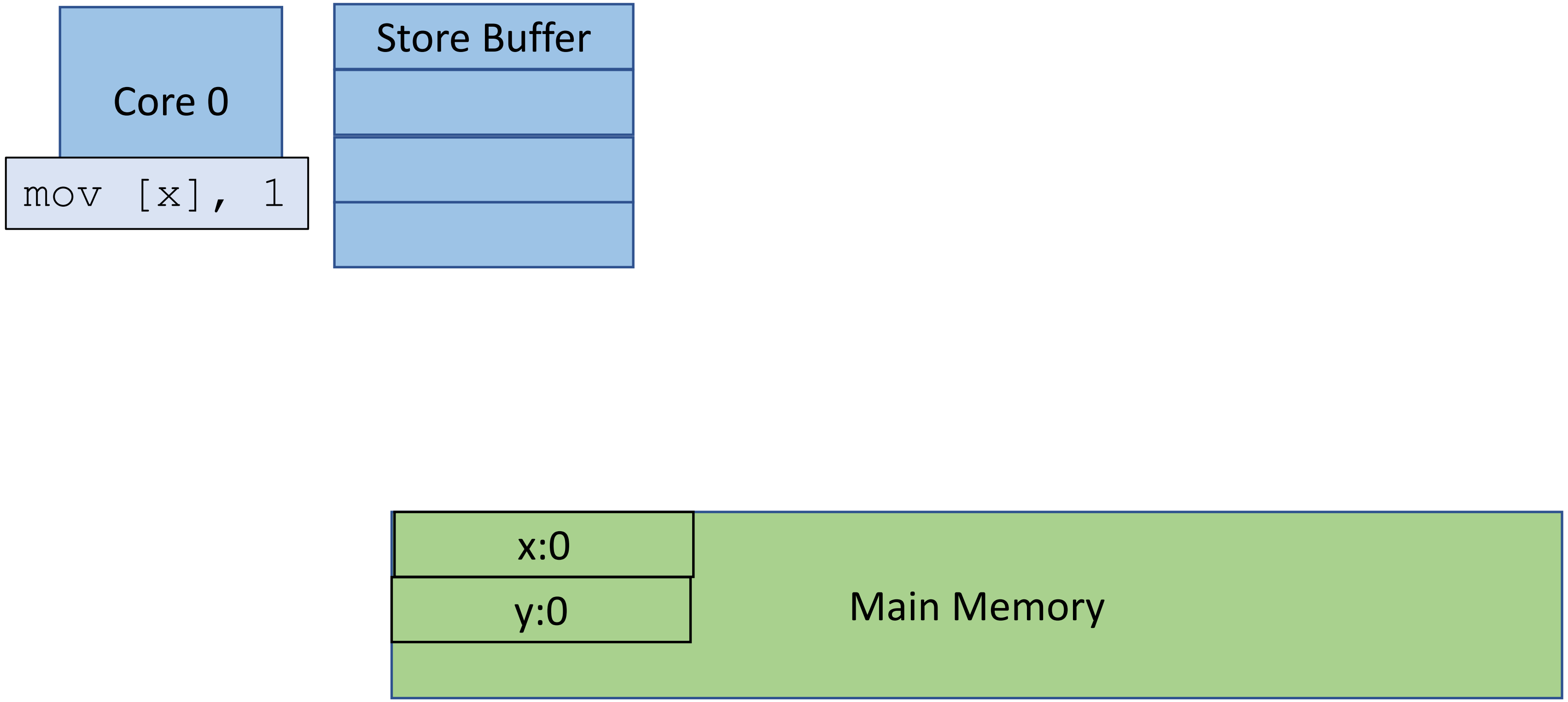
y:0

Main Memory

Thread 0:

execute first instruction

`mov %t0, [x]`



Core 0

`mov [x], 1`

Store Buffer

x:0

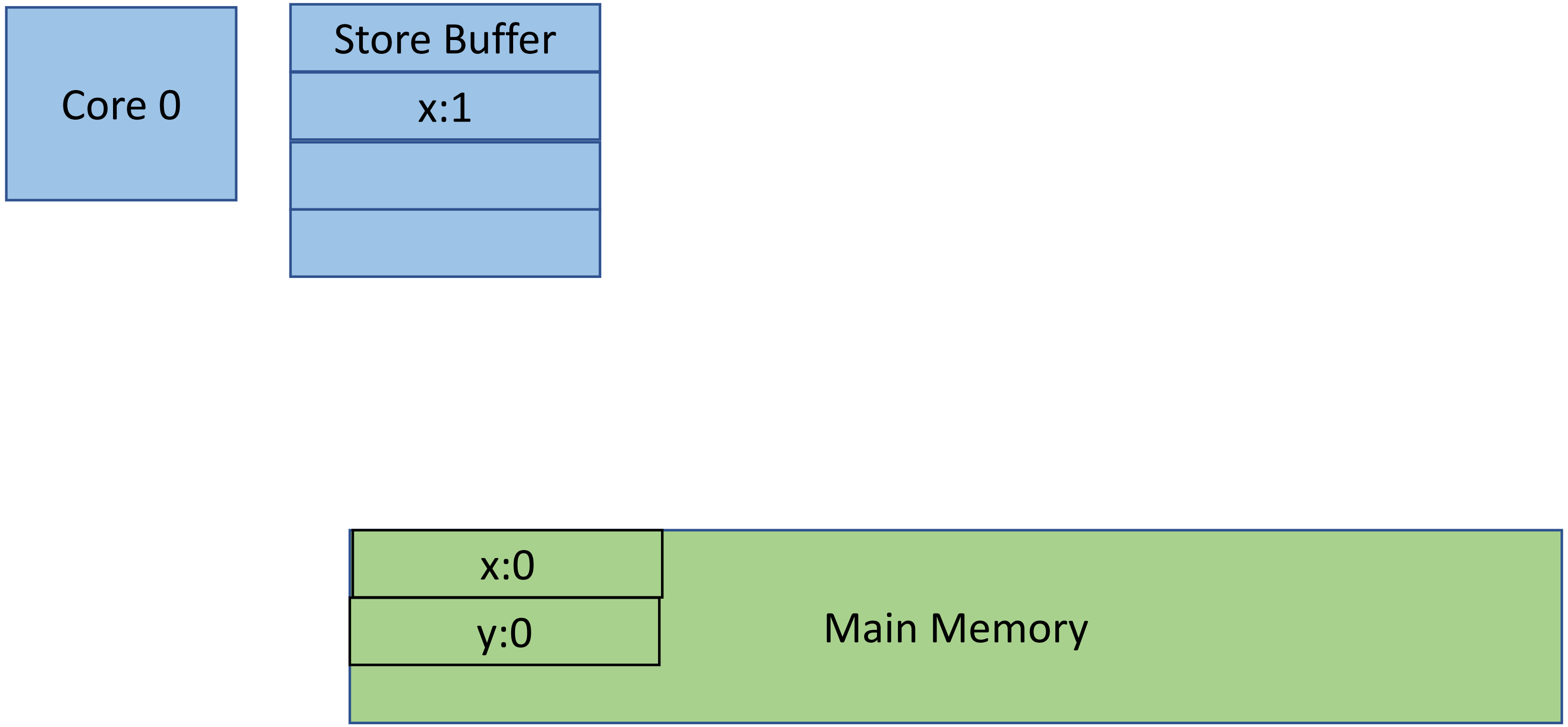
y:0

Main Memory

Thread 0:

Store the value in the store buffer

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



Core 0

Store Buffer

x:1

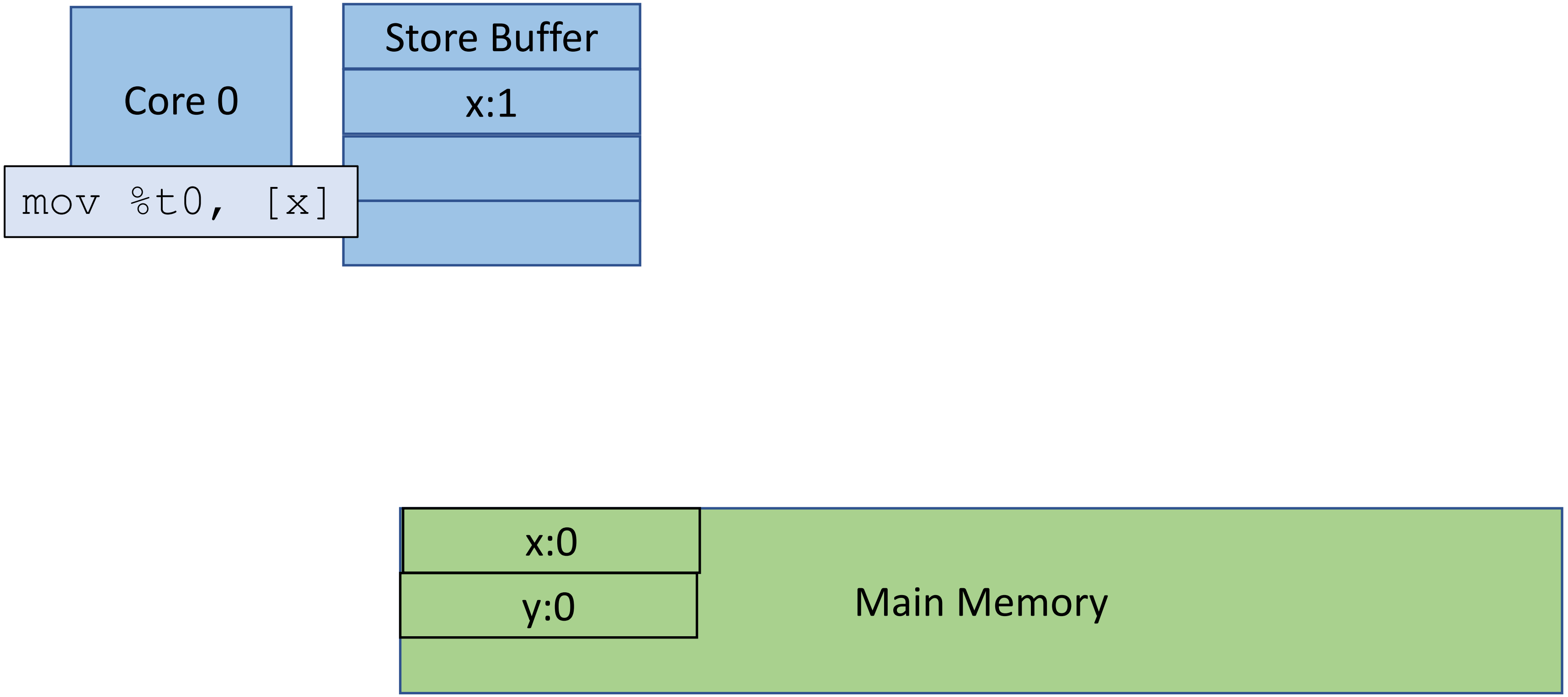
x:0

y:0

Main Memory

Thread 0:

Next instruction



Core 0

`mov %t0, [x]`

Store Buffer

x:1

x:0

y:0

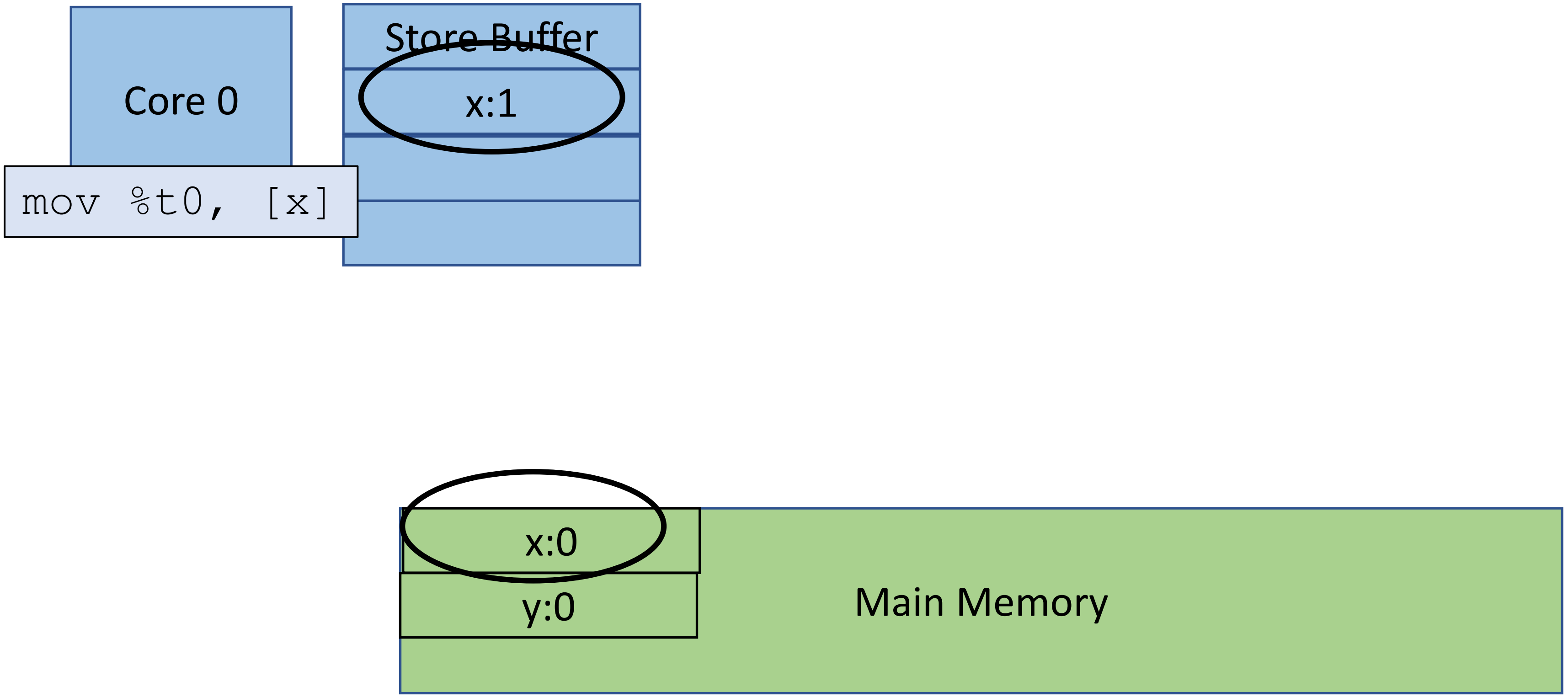
Main Memory

Thread 0:

Where to load??

Store buffer?

Main memory?



Core 0

`mov %t0, [x]`

Store Buffer

x:1

x:0

y:0

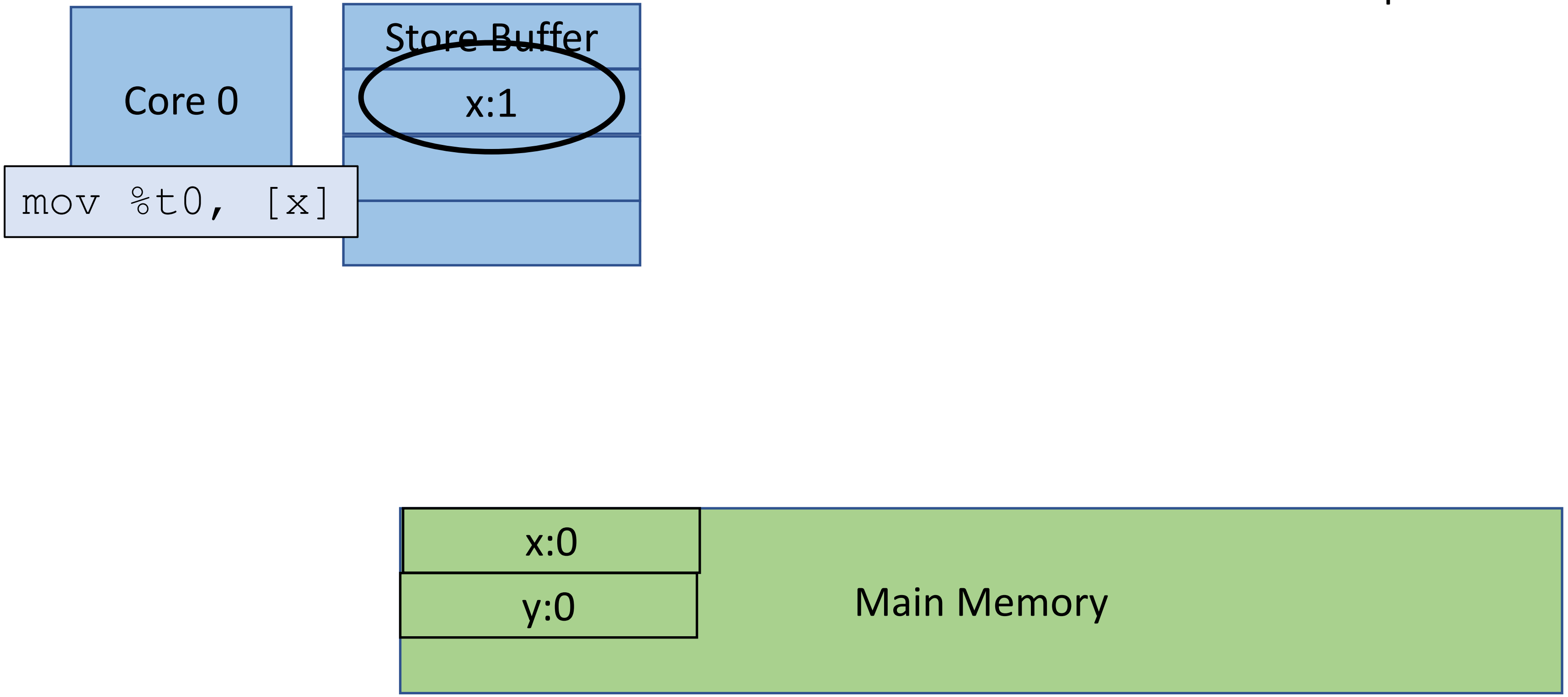
Main Memory

Thread 0:

Where to load??

Threads check store buffer before going to main memory

It is close and cheap to check.



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

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

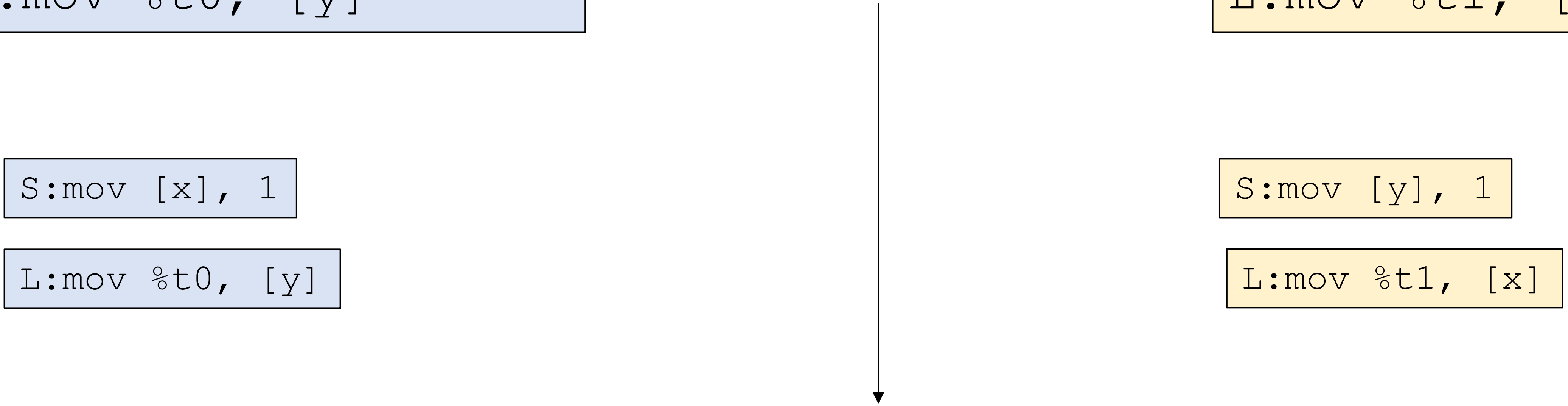
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



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

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

Global variable:

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

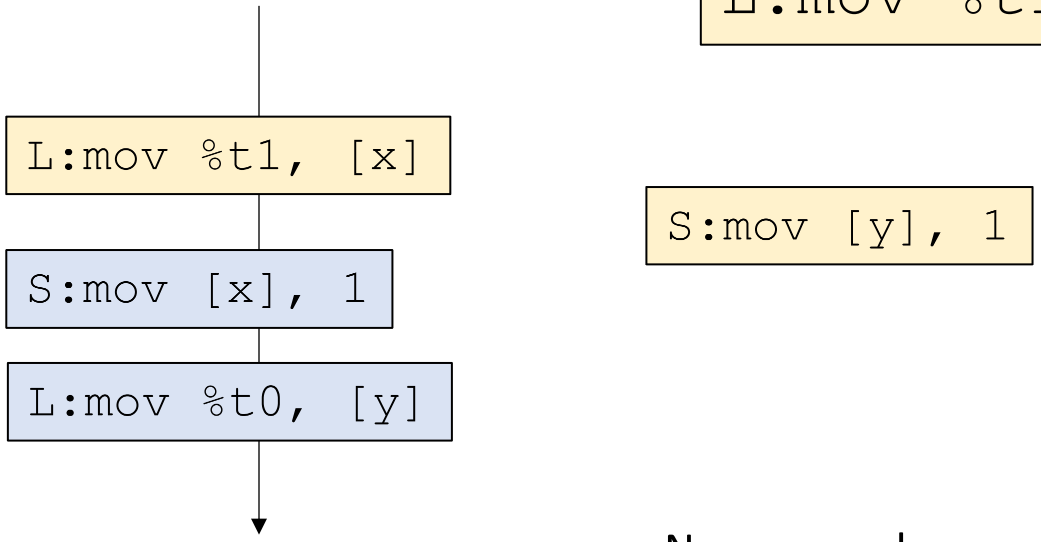
Another test
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



```
L:mov %t1, [x]
S:mov [x], 1
L:mov %t0, [y]
```

```
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`?

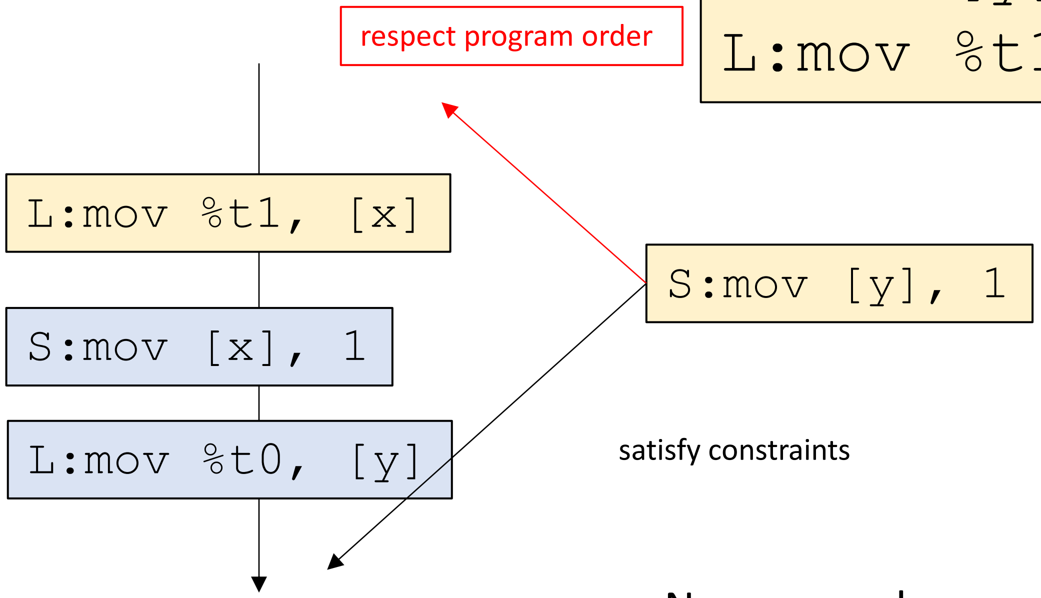
Thread 0:

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

Thread 1:

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

we can ignore this condition!!



respect program order

```
L:mov %t1, [x]
S:mov [x], 1
L:mov %t0, [y]
```

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

satisfy constraints

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

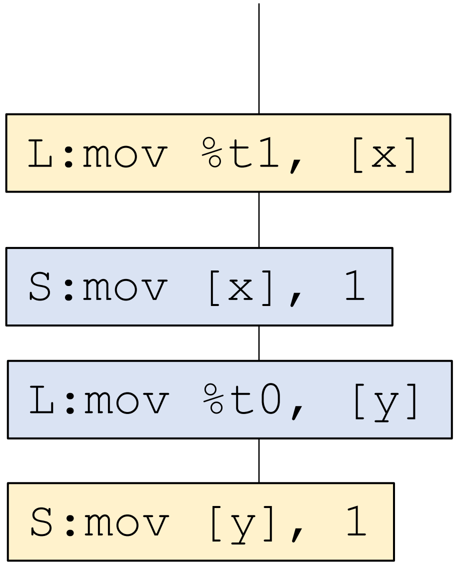
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



```
L:mov %t1, [x]
S:mov [x], 1
L:mov %t0, [y]
S:mov [y], 1
```

Now we can satisfy the condition!

Global variable:

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

Another test

Can `t0 == t1 == 0`?

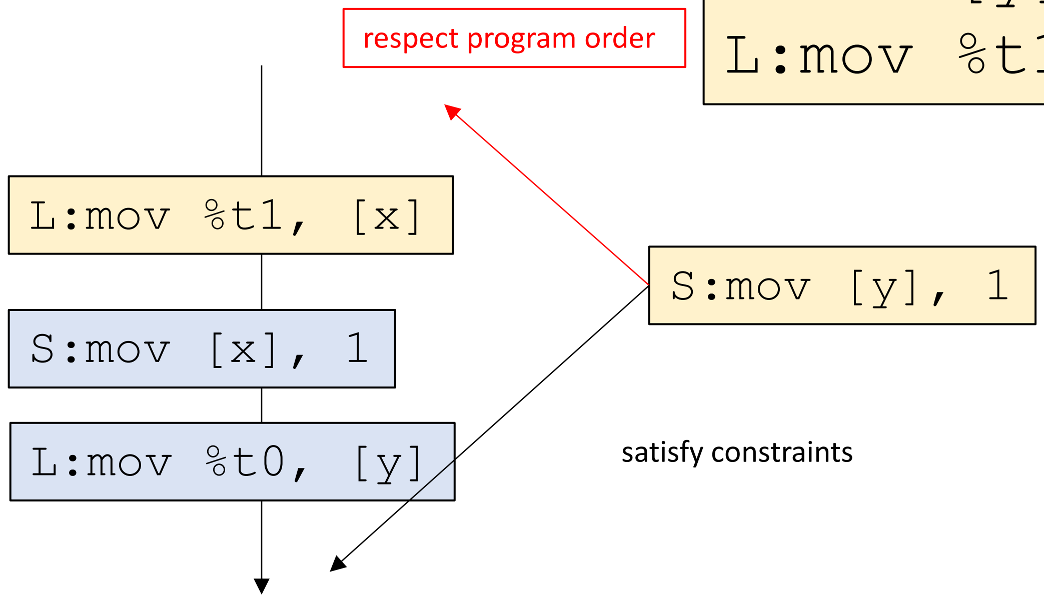
Thread 0:

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

Thread 1:

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

we can ignore this condition!!



respect program order

`L:mov %t1, [x]`

`S:mov [x], 1`

`L:mov %t0, [y]`

`S:mov [y], 1`

satisfy constraints

Lets peak under the hood here

Global variable:

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

Another test

Can `t0 == t1 == 0`?

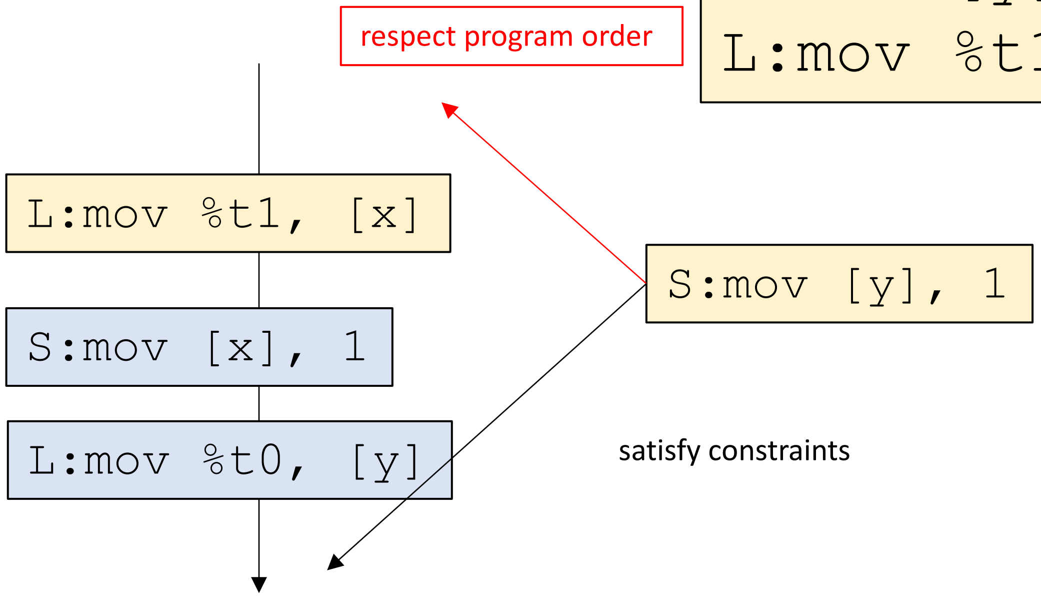
Thread 0:

```
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`?

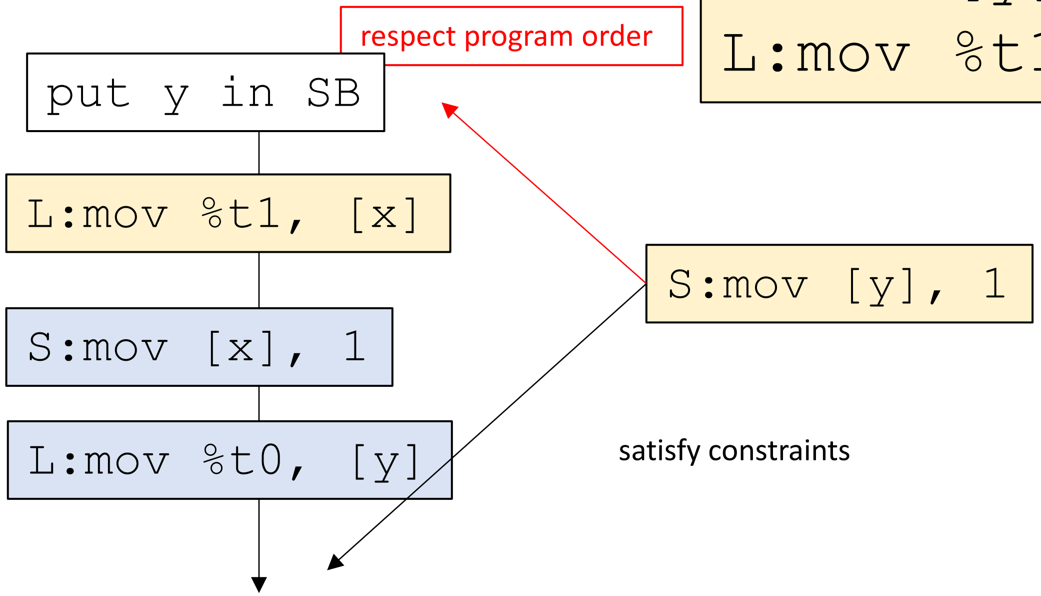
Thread 0:

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

Thread 1:

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

we can ignore this condition!!



respect program order

`put y in SB`

`L:mov %t1, [x]`

`S:mov [x], 1`

`L:mov %t0, [y]`

`S:mov [y], 1`

satisfy constraints

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]
```

```
put y in SB
L:mov %t1, [x]
S:mov [x], 1
L:mov %t0, [y]
S:mov [y], 1
```

store buffer gets flushed

Lets peak under the hood here

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

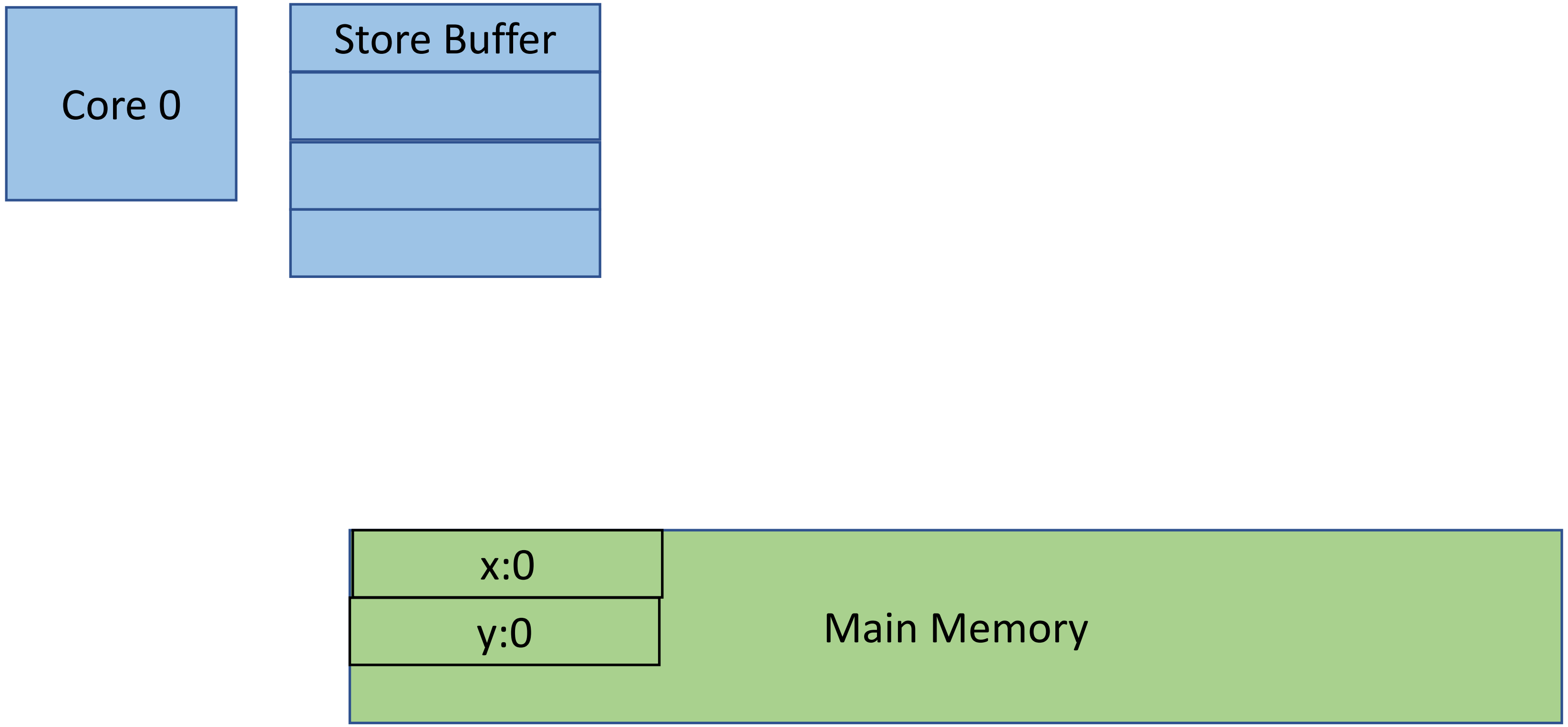
Questions

- Can stores be reordered with stores?

Thread 0:

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

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



Core 0

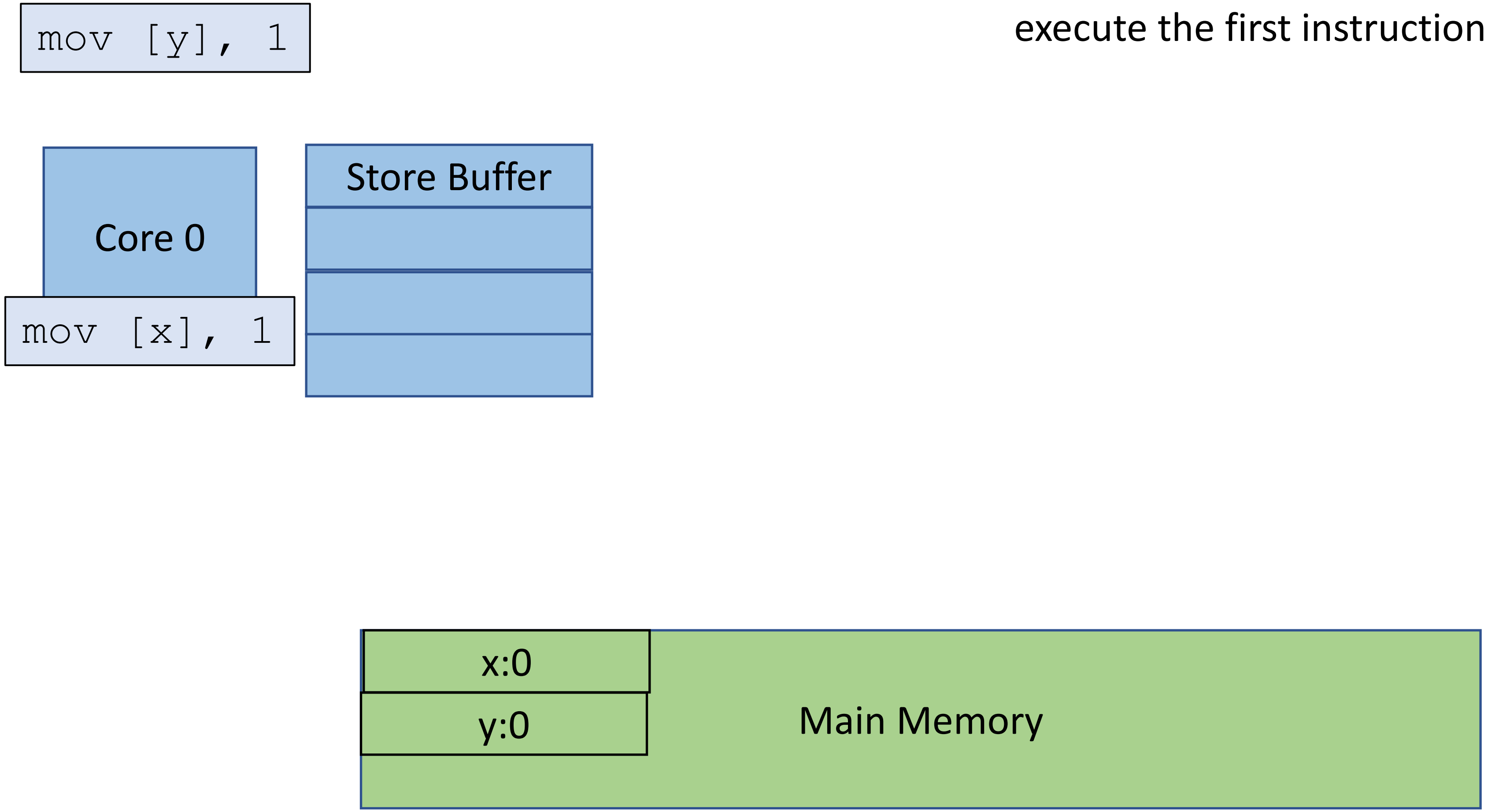
Store Buffer

x:0

y:0

Main Memory

Thread 0:



`mov [y], 1`

Core 0

`mov [x], 1`

Store Buffer

x:0

y:0

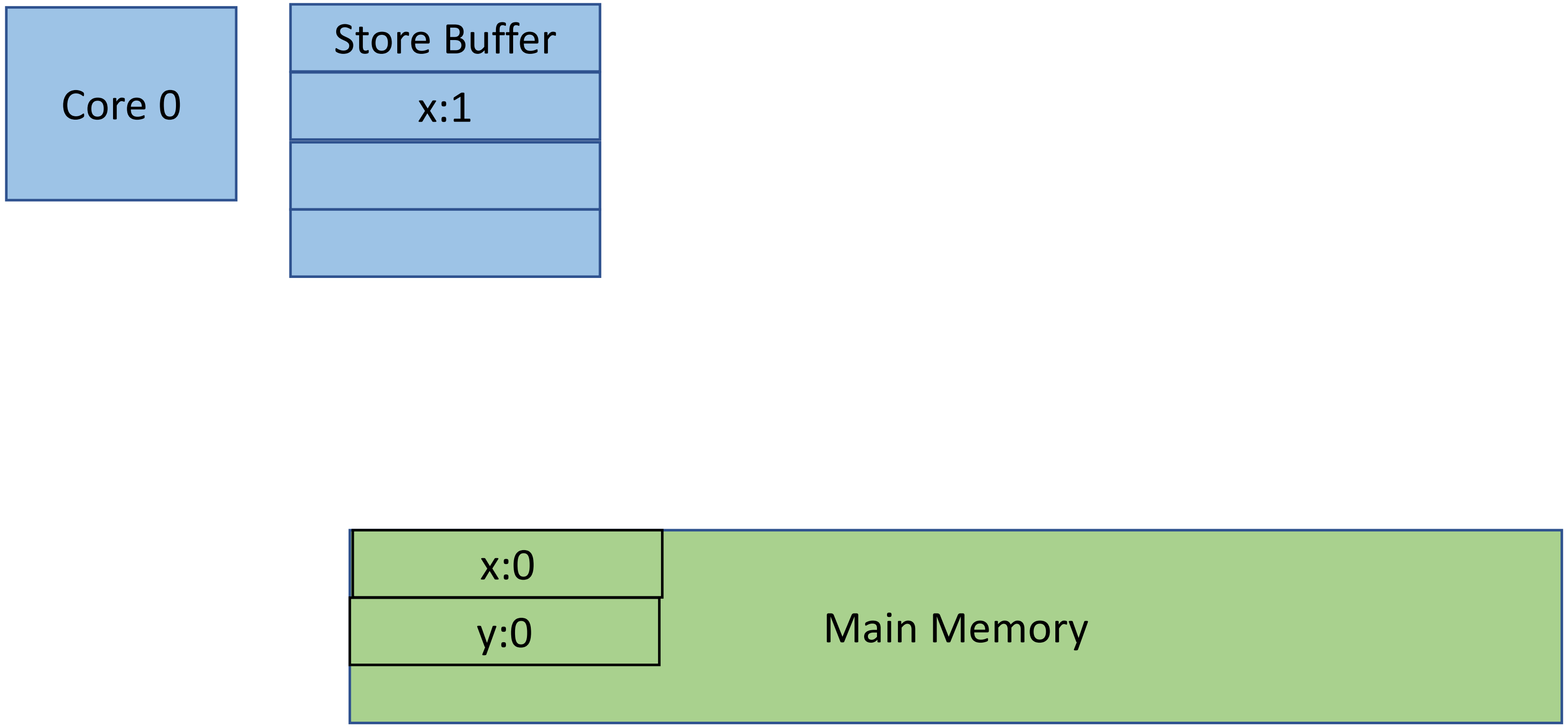
Main Memory

execute the first instruction

Thread 0:

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

value goes into store buffer



Core 0

Store Buffer

x:1

x:0

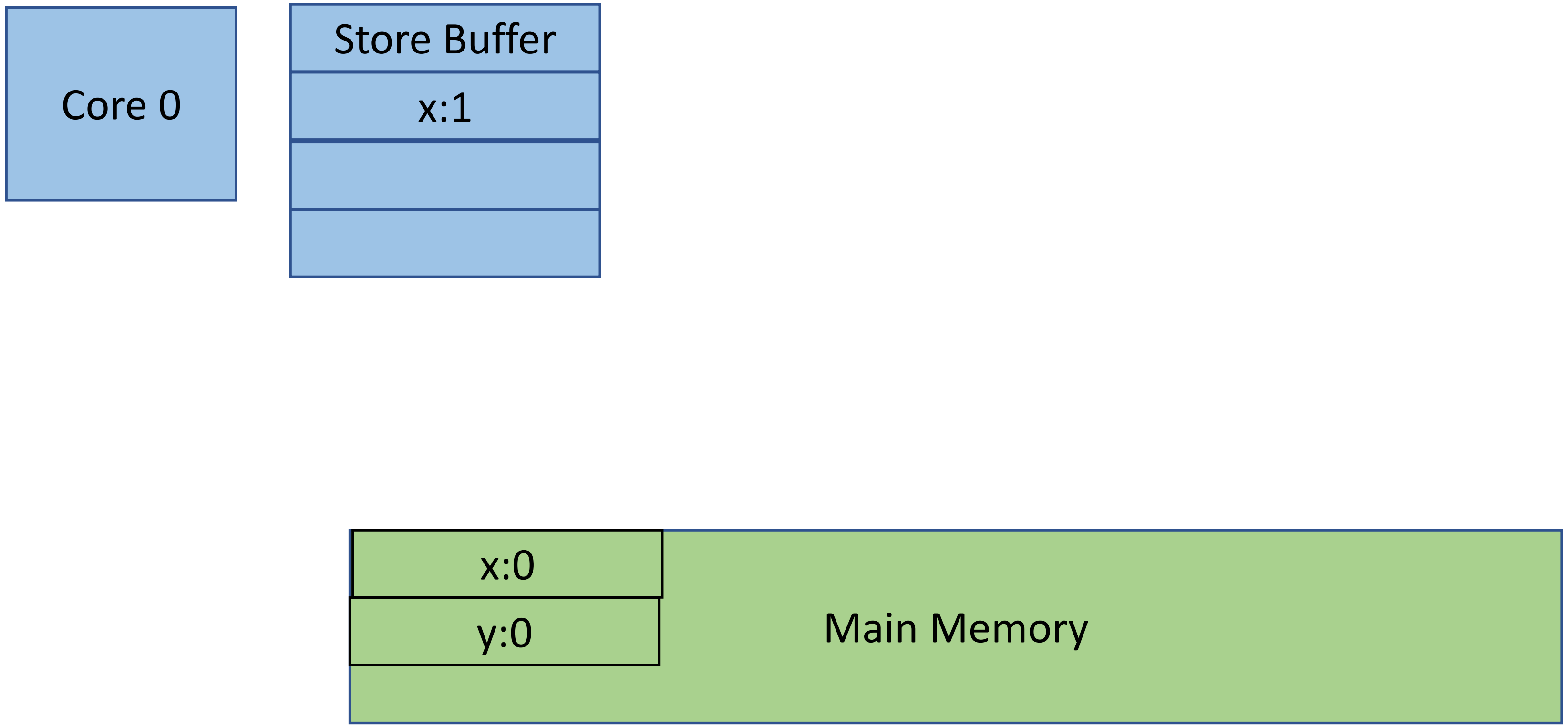
y:0

Main Memory

Thread 0:

`mov [y], 1`

execute next instruction



Core 0

Store Buffer

x:1

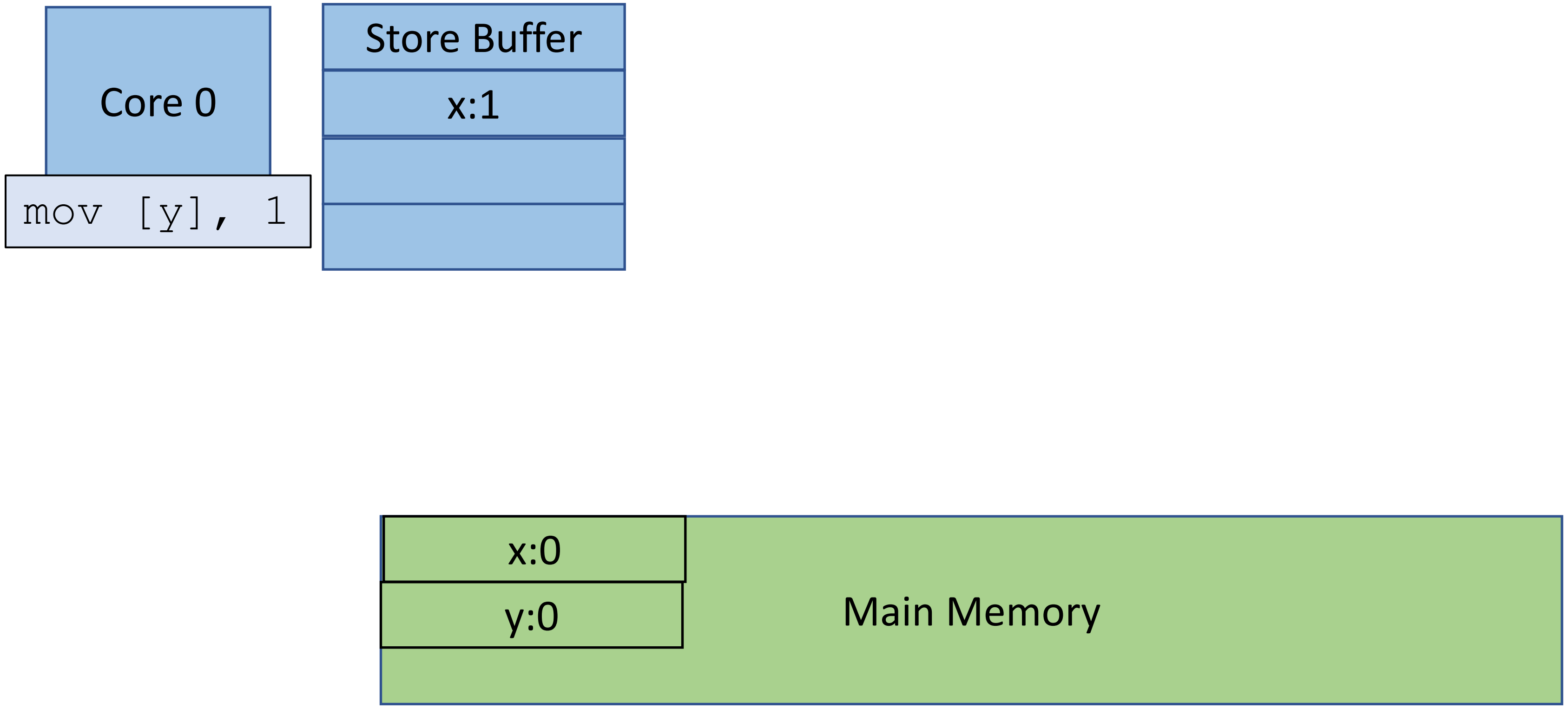
x:0

y:0

Main Memory

Thread 0:

execute next instruction



Core 0

`mov [y], 1`

Store Buffer

x:1

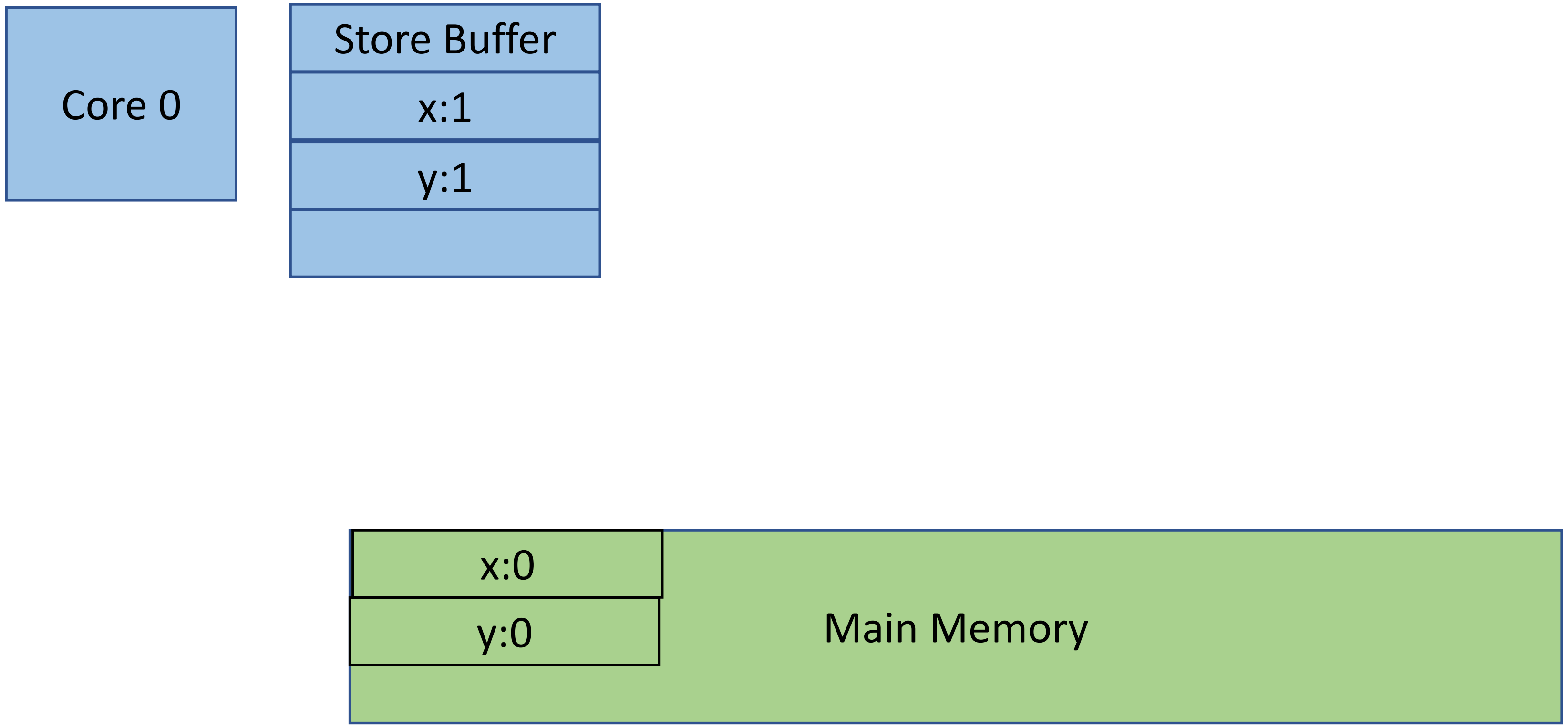
x:0

y:0

Main Memory

Thread 0:

value goes into the store buffer



Core 0

Store Buffer

x:1

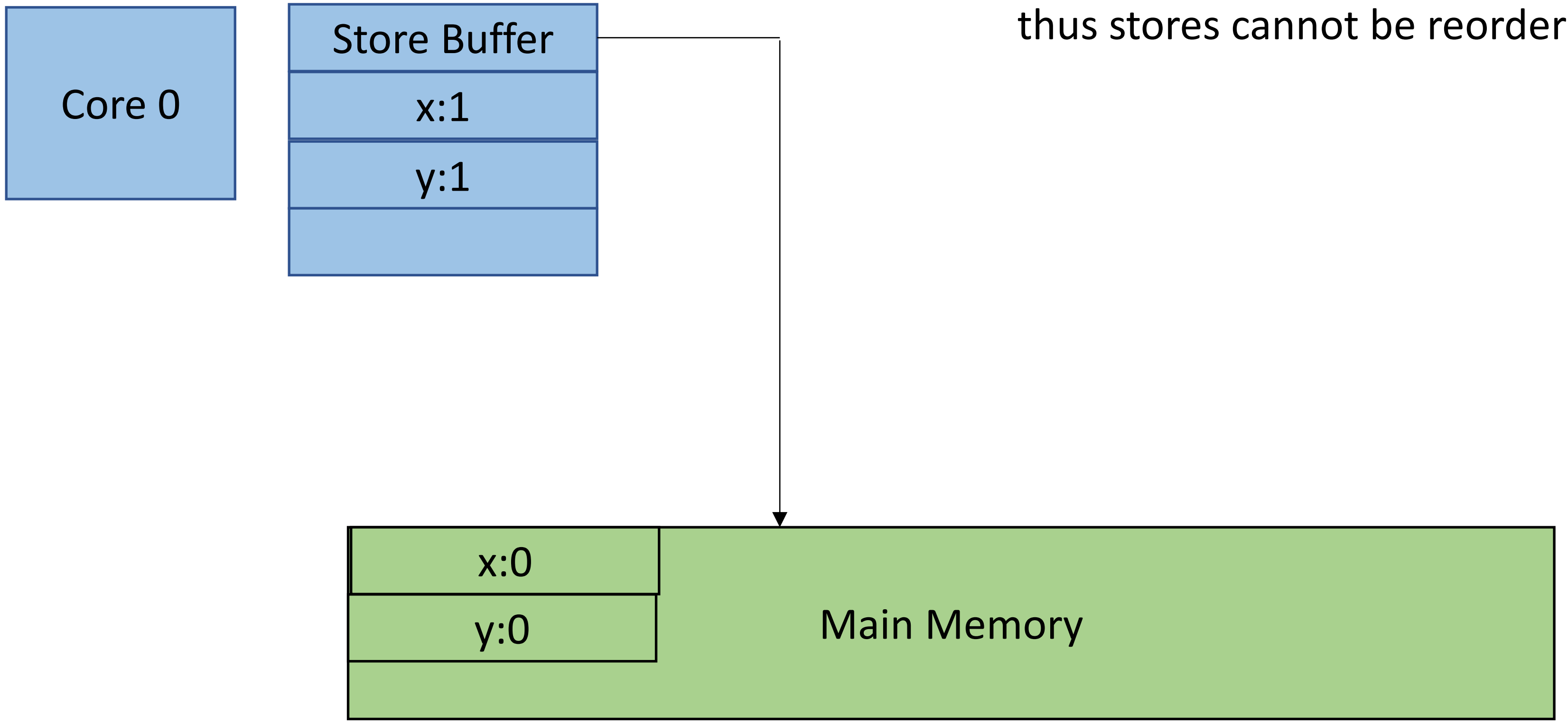
y:1

x:0

y:0

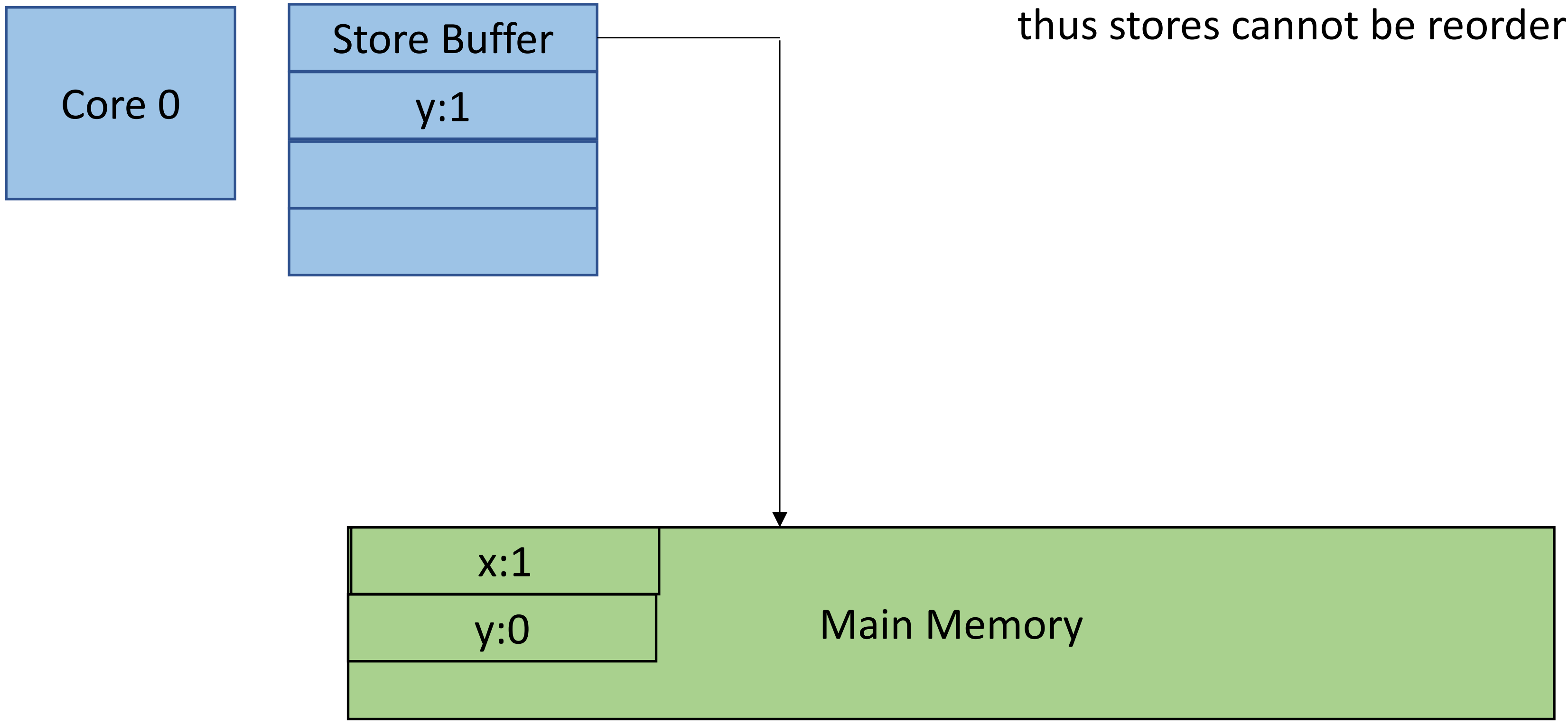
Main Memory

Thread 0:



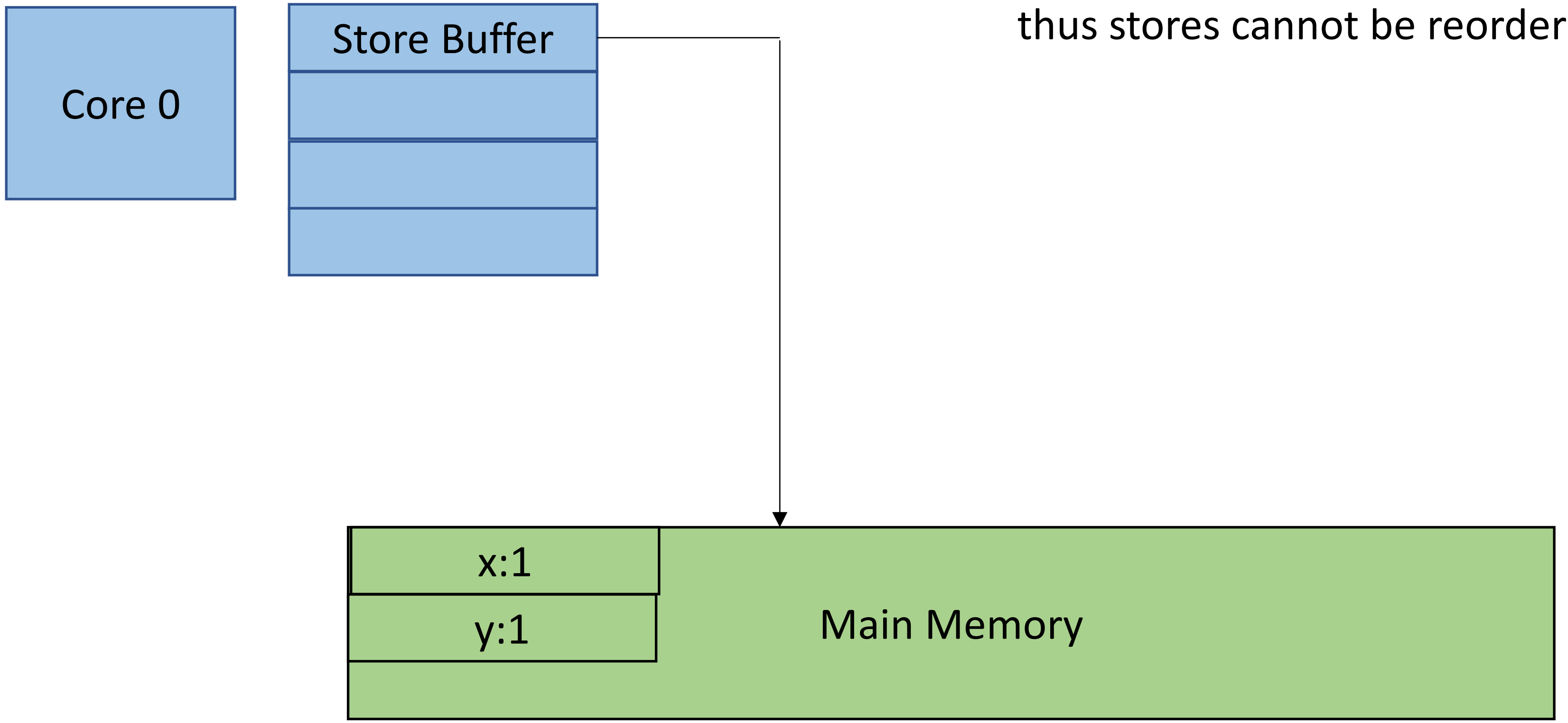
On x86, the store buffer trains in a FIFO way:
thus stores cannot be reordered

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]
```

`S:mov [x], 1`

`mfence`

`L:mov %t0, [y]`

`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};
```

Another test

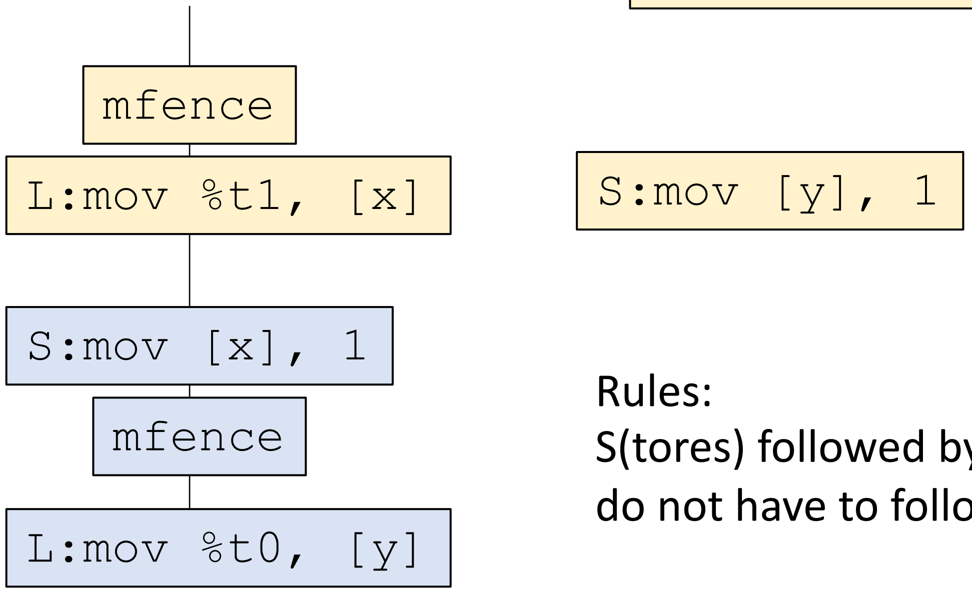
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



```
mfence
L:mov %t1, [x]
S:mov [x], 1
mfence
L:mov %t0, [y]
```

```
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

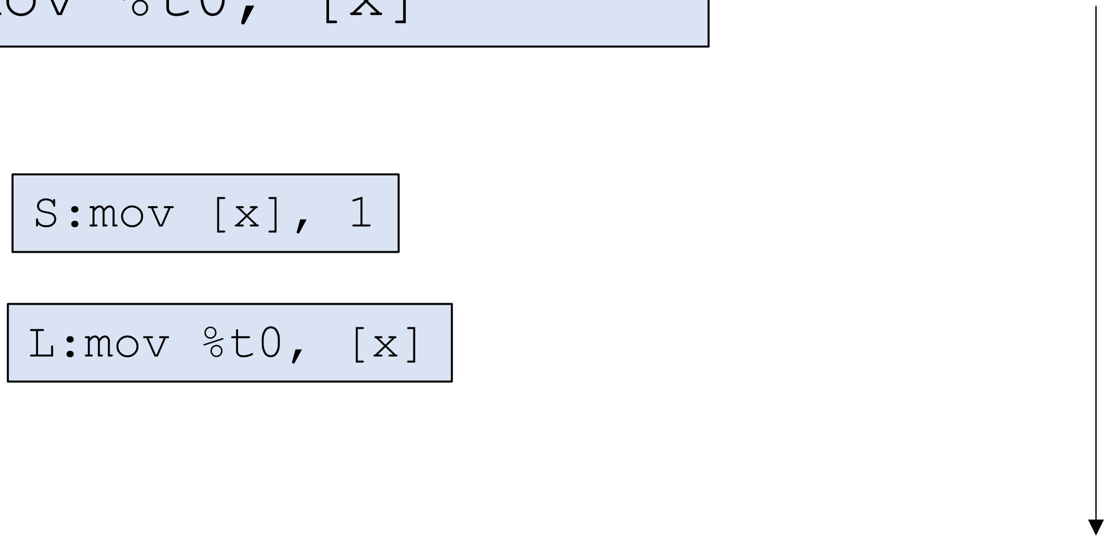
Global variable:

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

Another test
Can `t0 == 0`?

Thread 0:

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



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

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

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

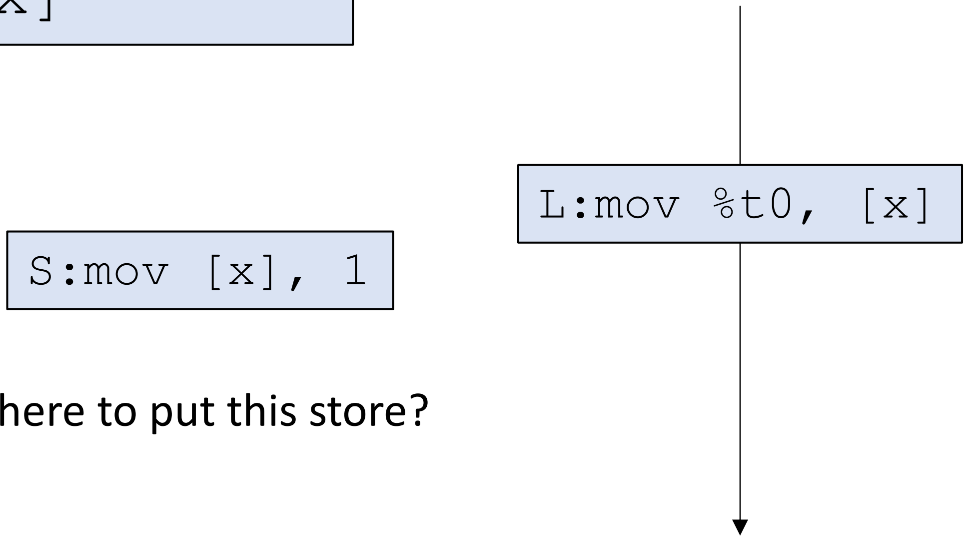
Global variable:

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

Another test
Can `t0 == 0`?

Thread 0:

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



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

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

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:

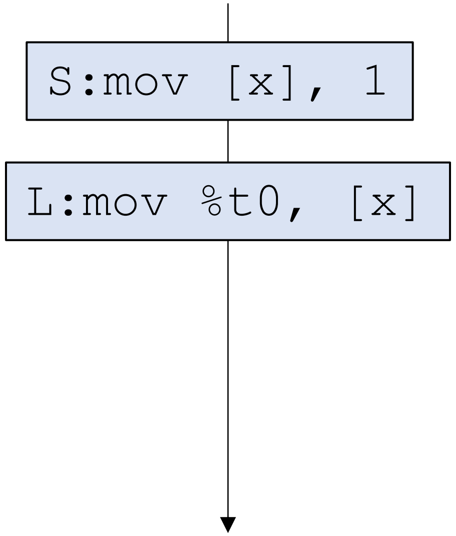
```
int x[1] = {0};
int y[1] = {0};
```

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**

Other memory models?

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

		memory access 0	
		L	S
memory access 1	L		
	S		

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 0

		L	S
memory access 1	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 0

		L	S
memory access 1	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

		memory access 0	
		L	S
memory access 1	L	?	?
	S	?	?

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 0	
		L	S
memory access 1	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

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

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 0	
		L	S
memory access 1	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
- **Some 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)
```

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

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

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

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

Global variable:

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

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

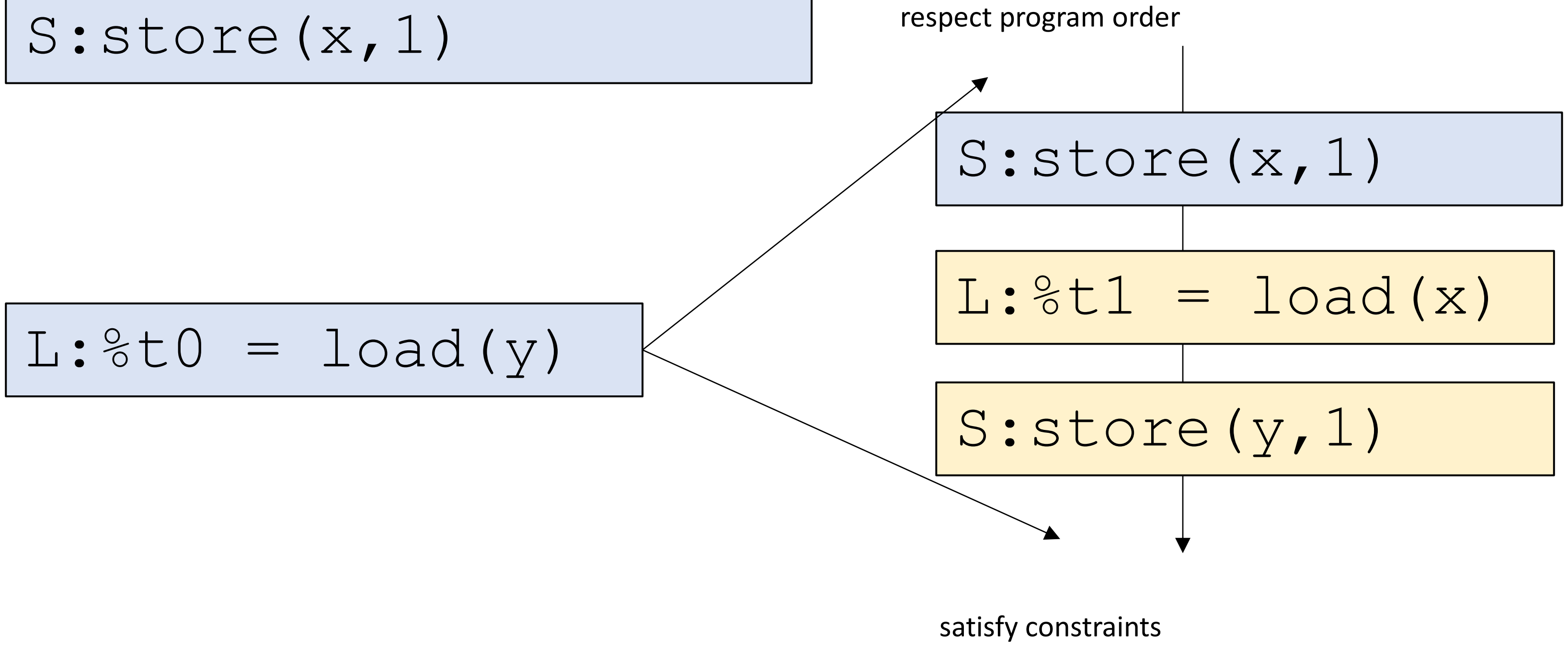
Get out our lego bricks

Thread 0:

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

Thread 1:

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



respect program order

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

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

satisfy constraints

Not allowed under sequential consistency!

Global variable:

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

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

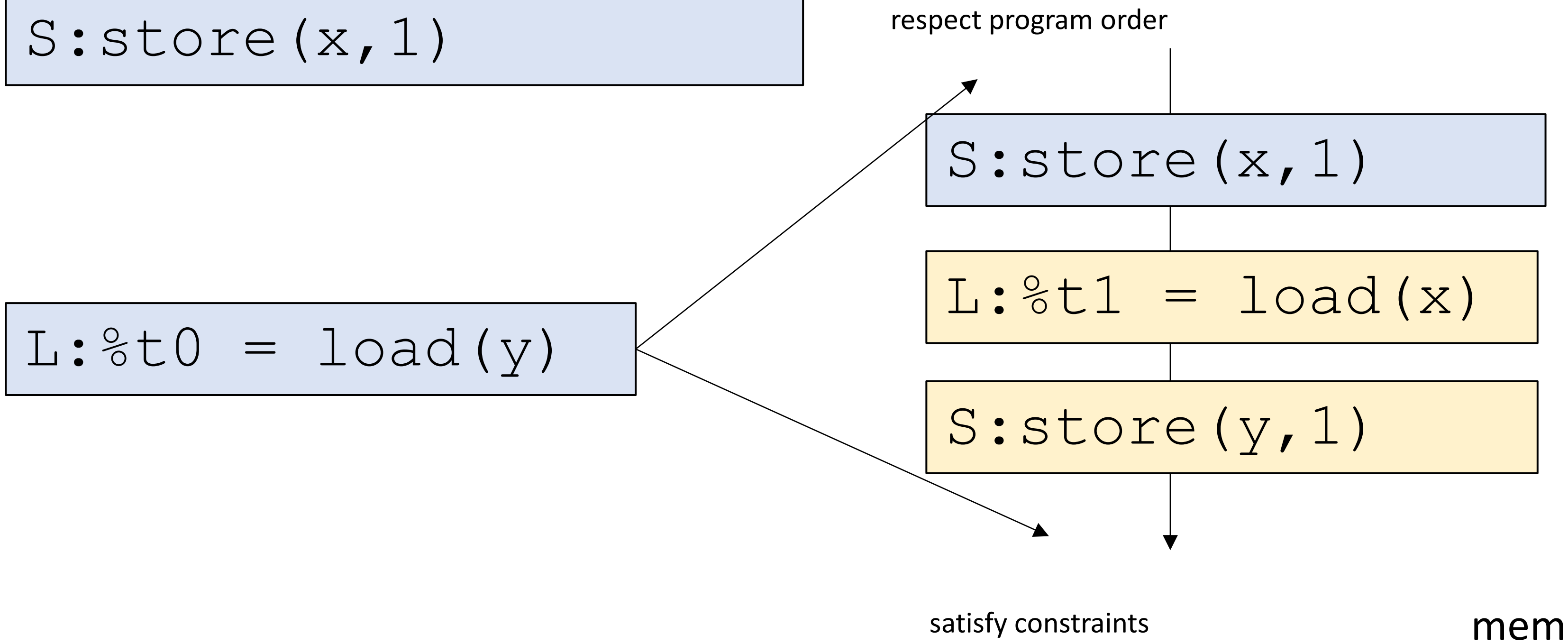
Get out our lego bricks

Thread 0:

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

Thread 1:

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



respect program order

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

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

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

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

satisfy constraints

		memory access 0	
		L	S
memory access 1	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?`

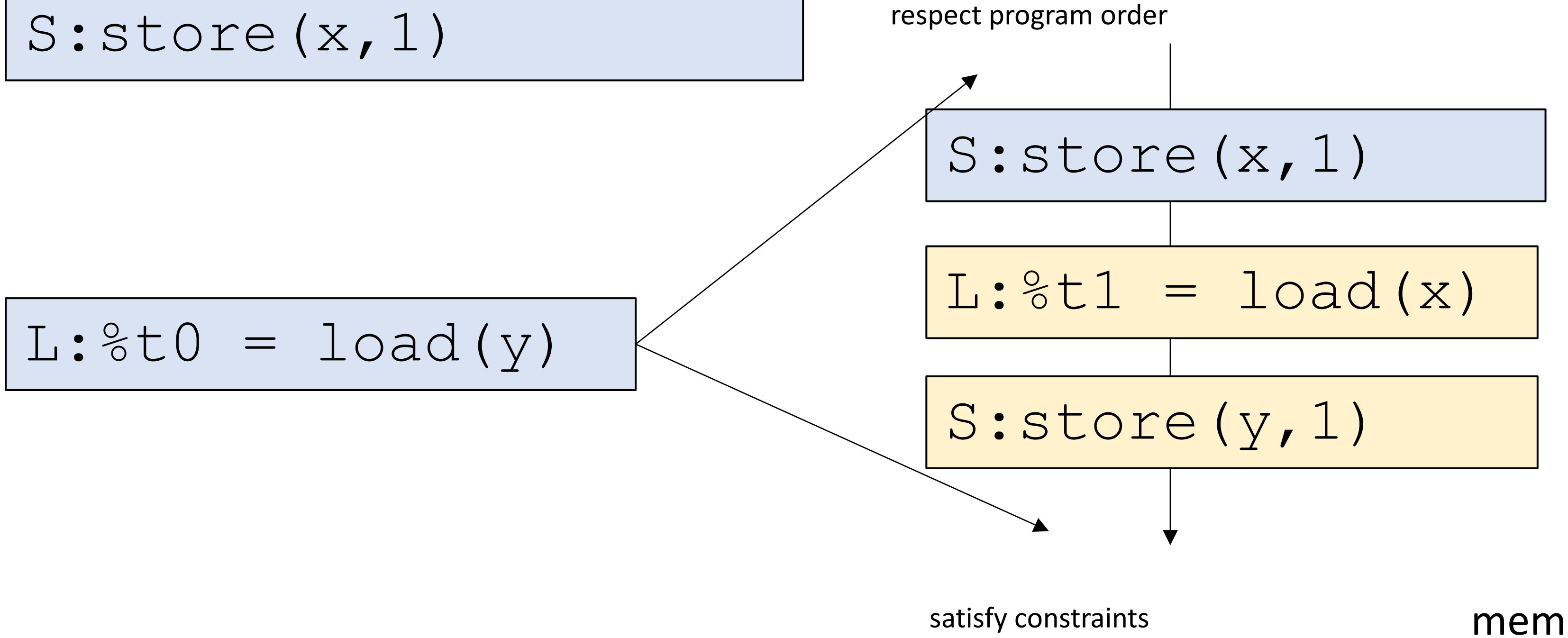
Get out our lego bricks

Thread 0:

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

Thread 1:

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



respect program order

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

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

satisfy constraints

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?`

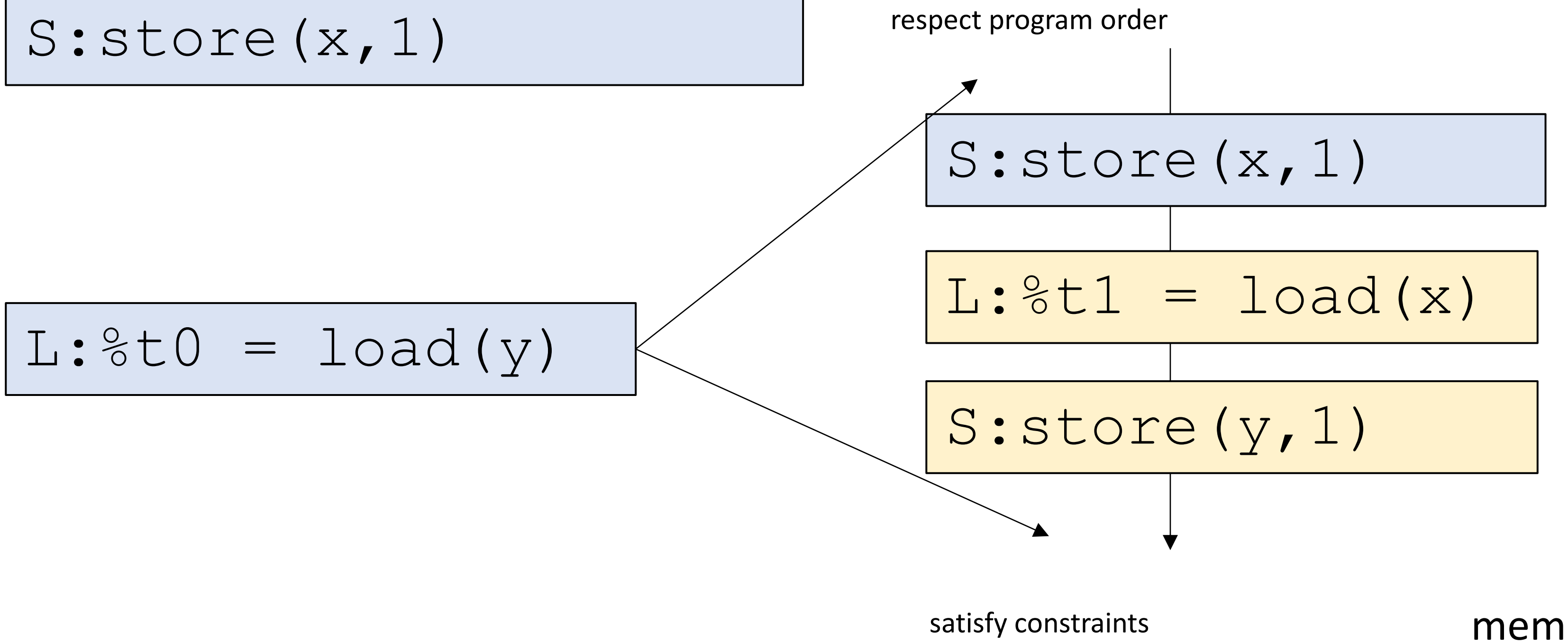
Get out our lego bricks

Thread 0:

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

Thread 1:

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



respect program order

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

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

satisfy constraints

		memory access 0	
		L	S
memory access 1	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?`

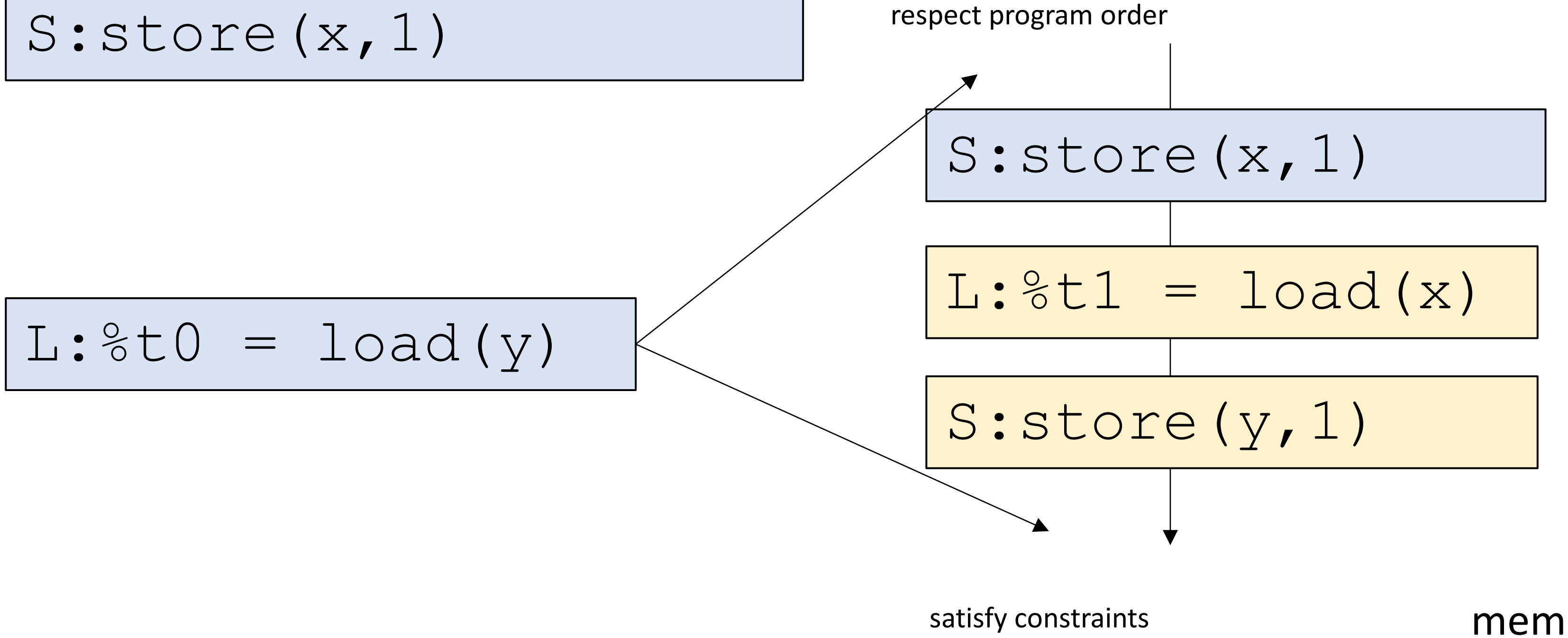
Get out our lego bricks

Thread 0:

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

Thread 1:

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



respect program order

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

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

satisfy constraints

		memory access 0	
		L	S
memory access 1	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?`

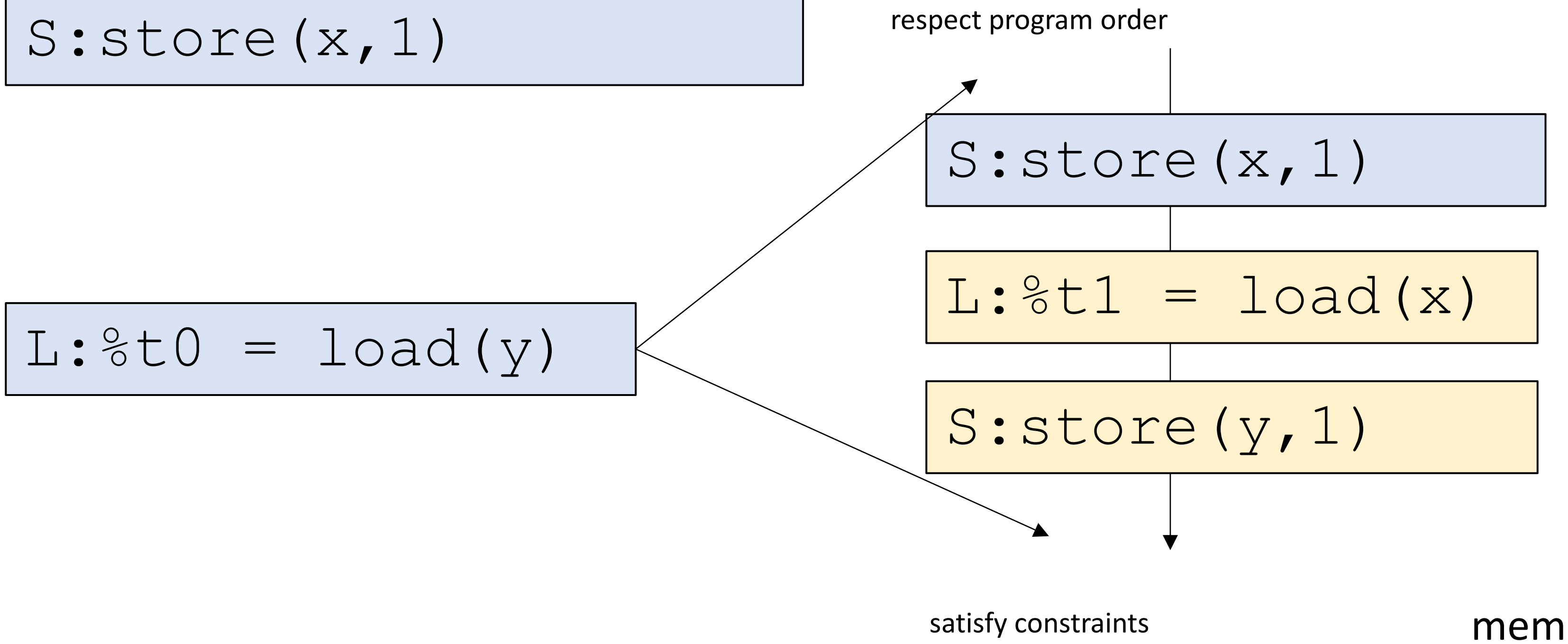
Get out our lego bricks

Thread 0:

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

Thread 1:

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



		memory access 0	
		L	S
memory access 1	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?`

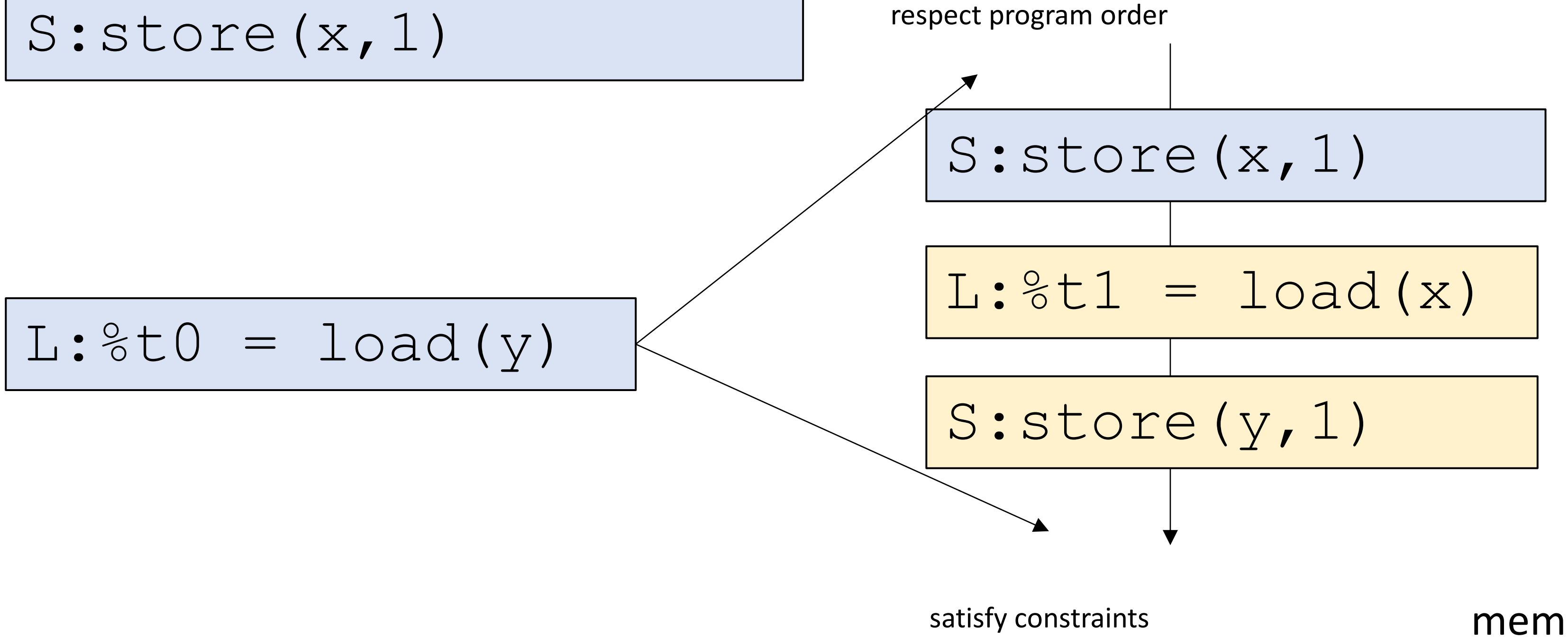
Get out our lego bricks

Thread 0:

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

Thread 1:

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



respect program order

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

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

satisfy constraints

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?`

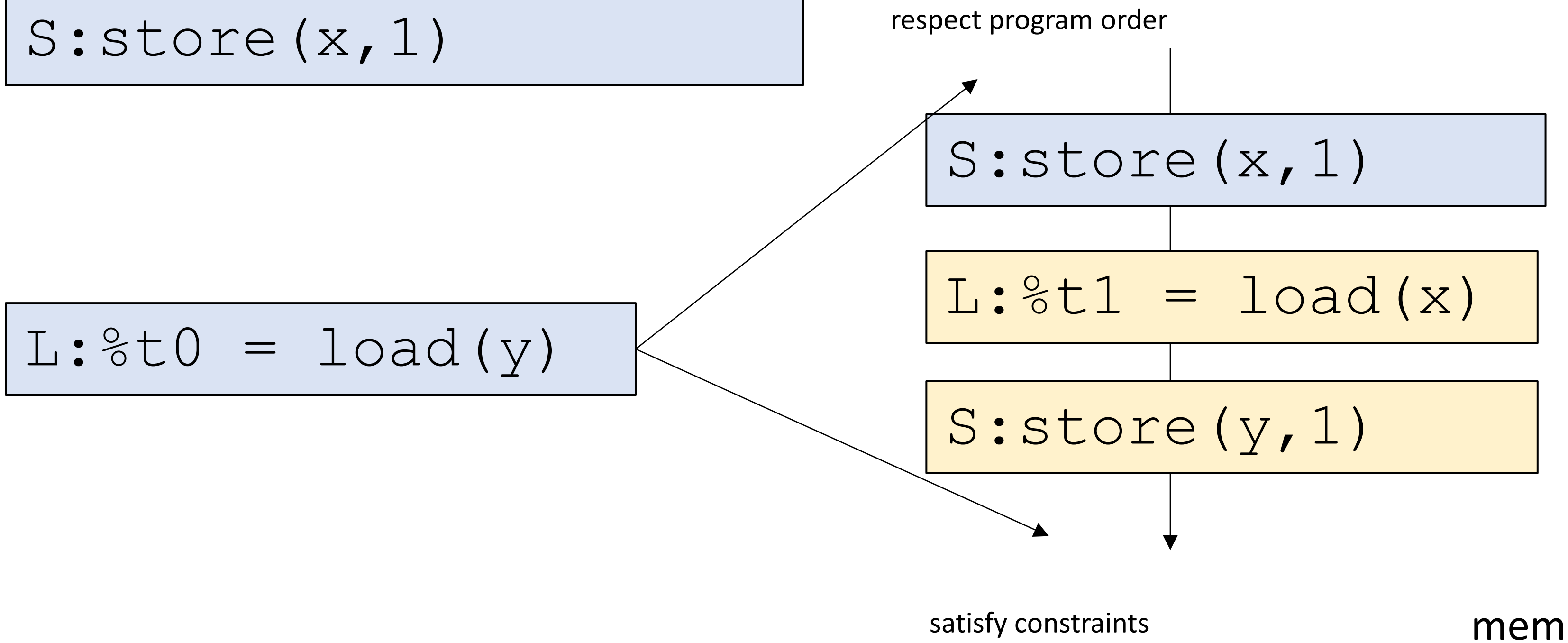
Get out our lego bricks

Thread 0:

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

Thread 1:

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



		memory access 0	
		L	S
memory access 1	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?`

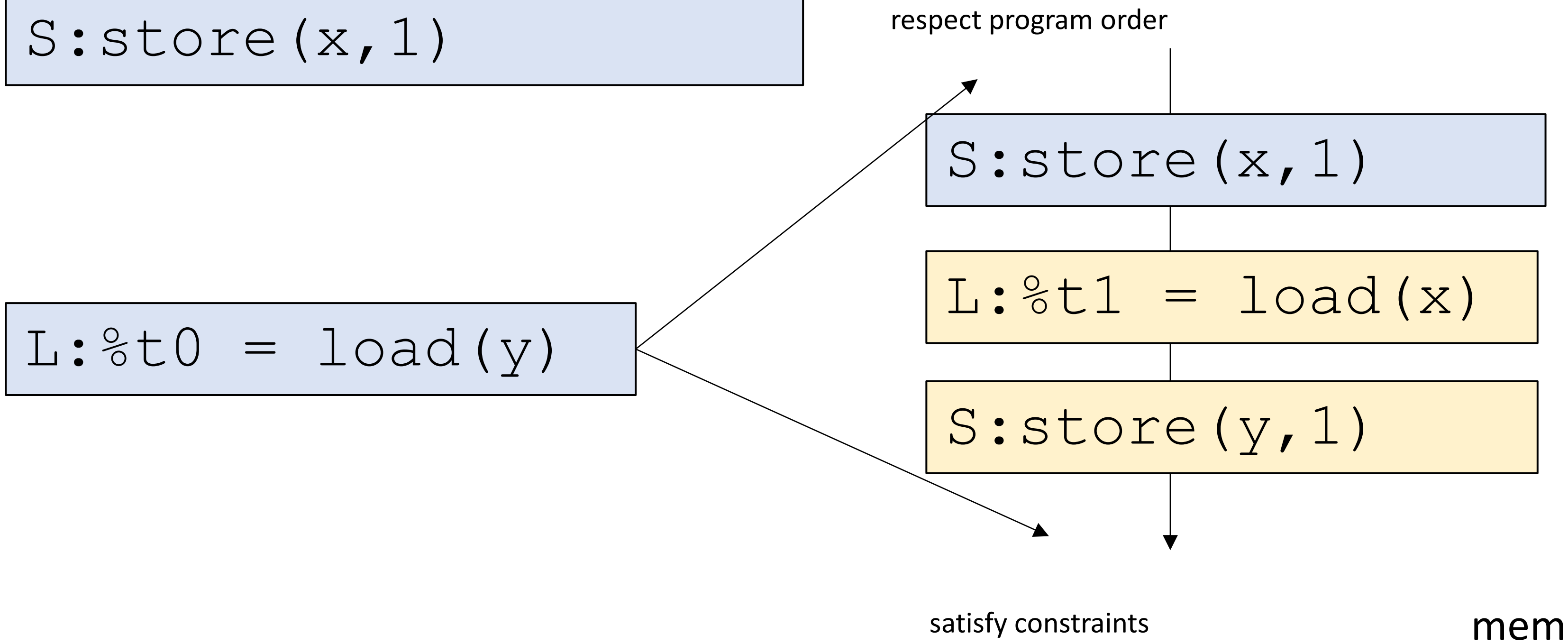
Get out our lego bricks

Thread 0:

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

Thread 1:

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



respect program order

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

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

satisfy constraints

memory access 0

		L	S
memory access 1	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?`

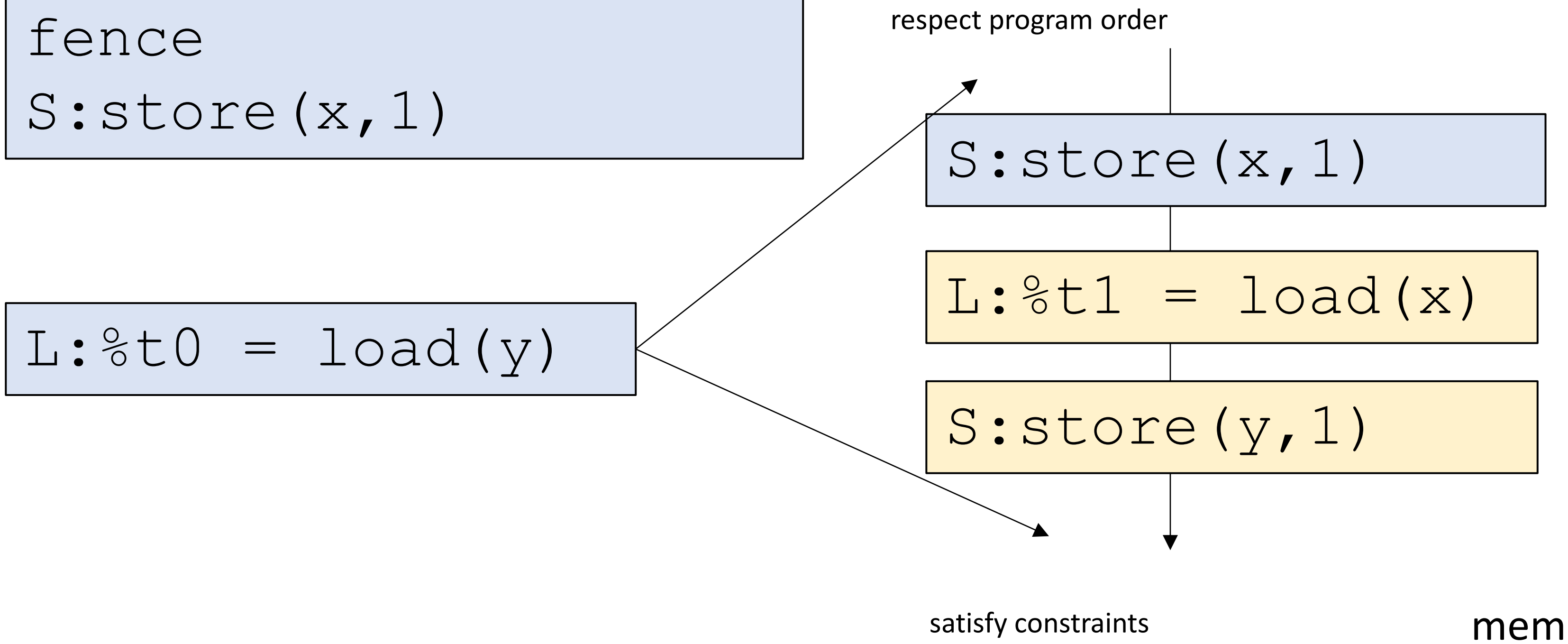
Get out our lego bricks

Thread 0:

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

Thread 1:

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



		memory access 0	
		L	S
memory access 1	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?`

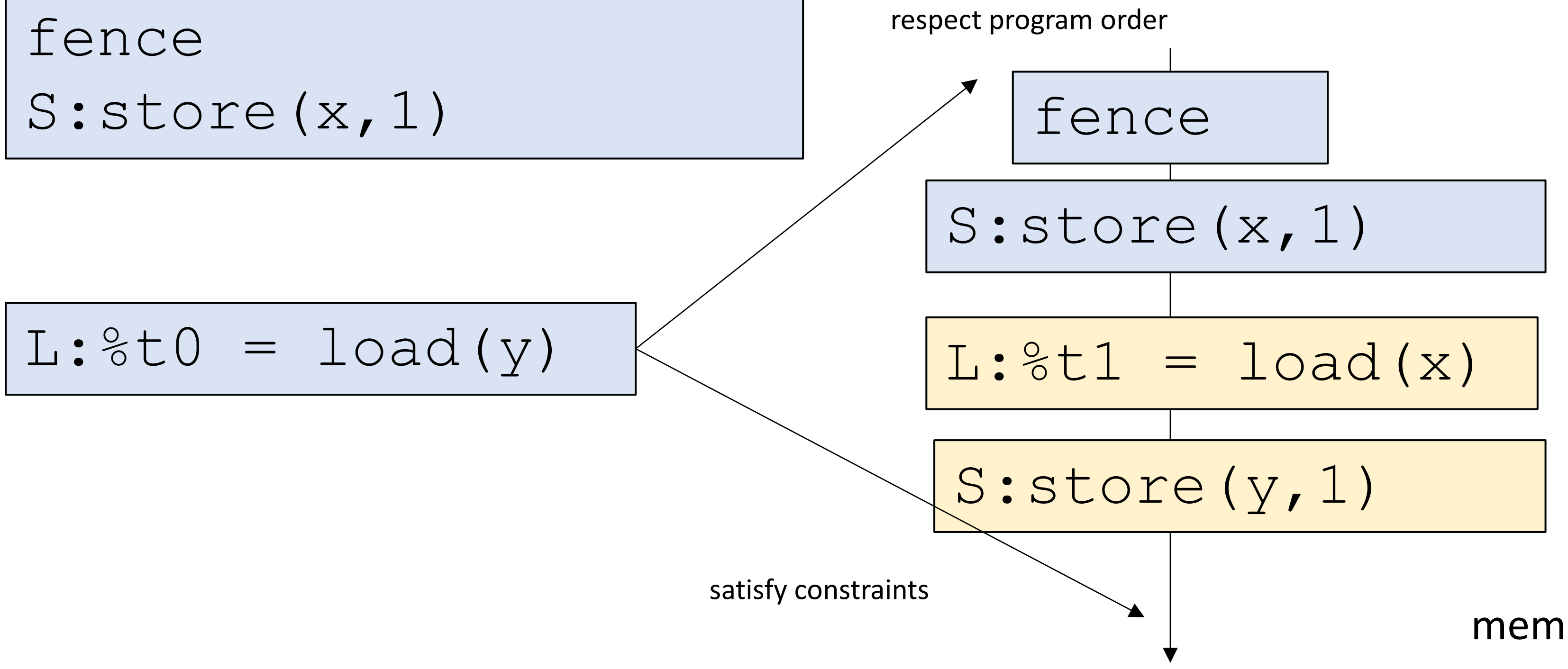
Get out our lego bricks

Thread 0:

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

Thread 1:

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



		memory access 0	
		L	S
memory access 1	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?`

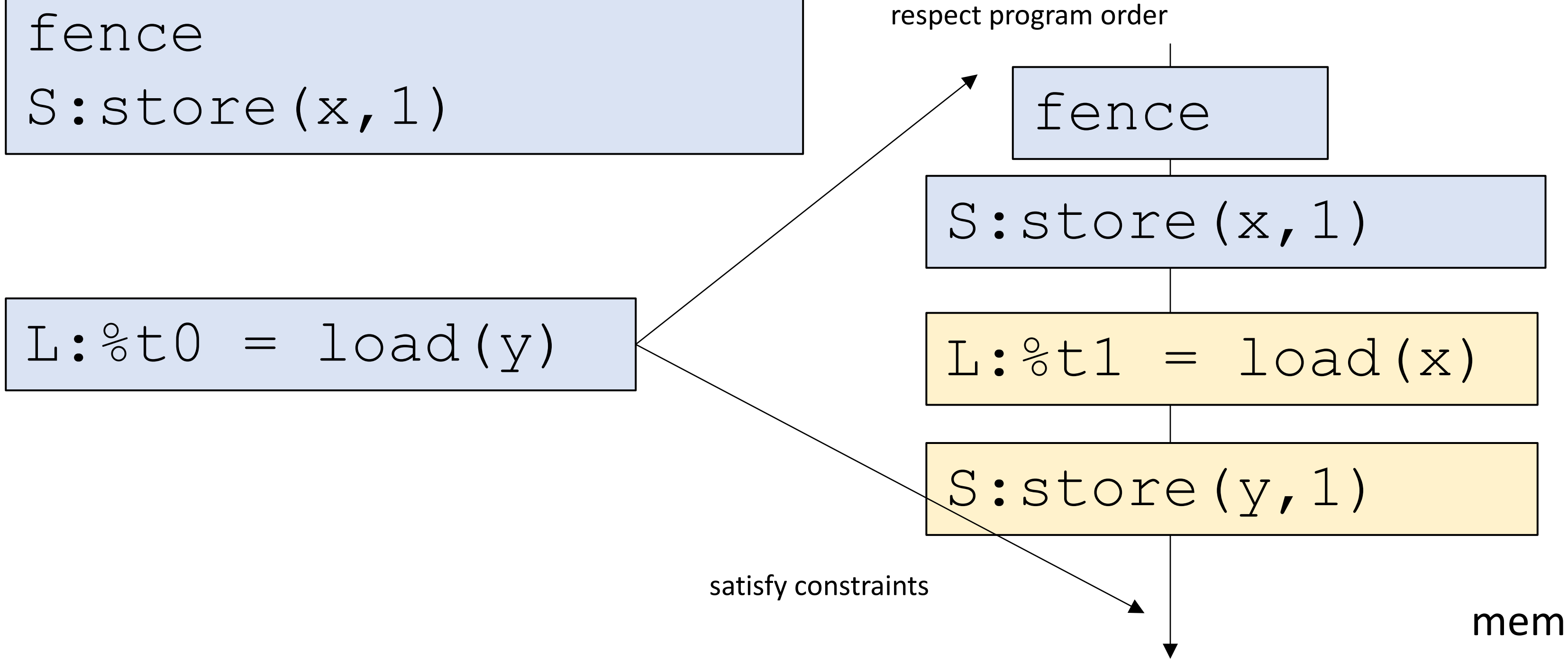
Get out our lego bricks

Thread 0:

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

Thread 1:

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



respect program order

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

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

satisfy constraints

		memory access 0	
		L	S
memory access 1	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?`

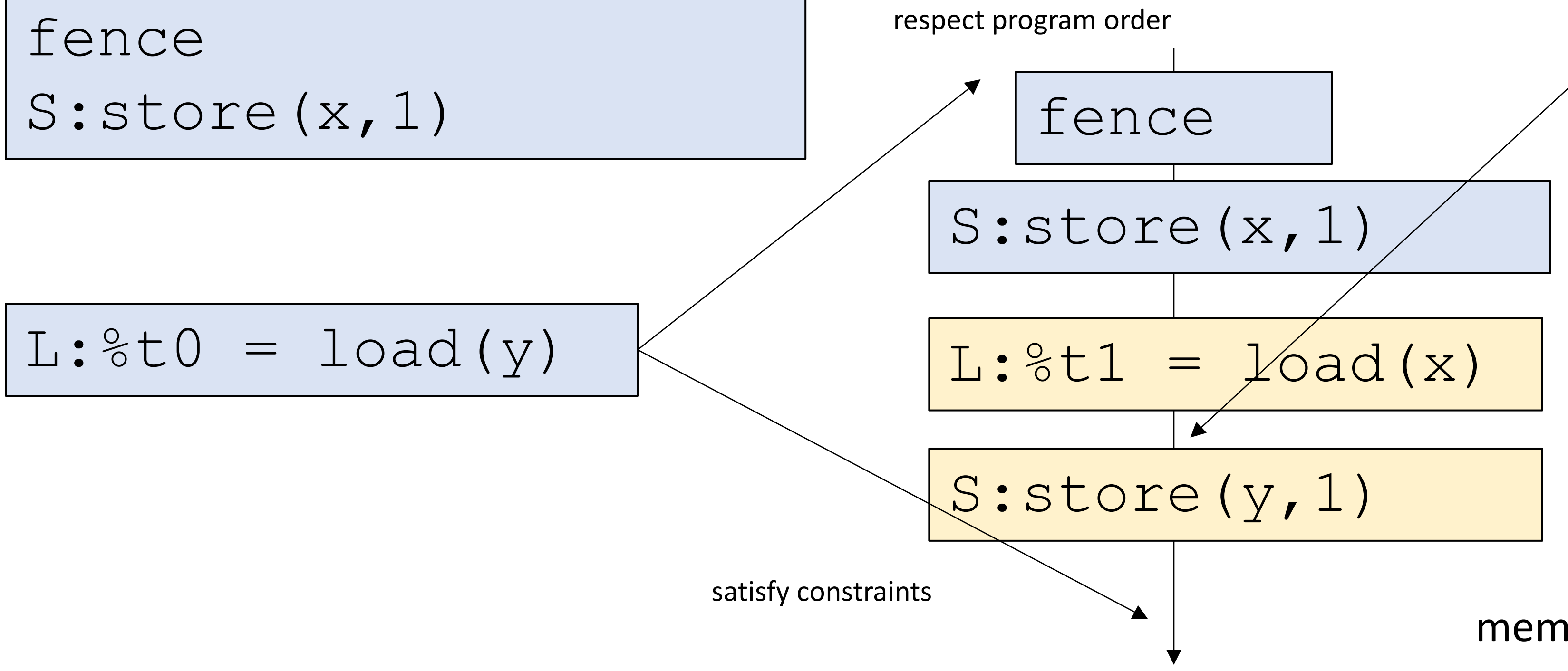
Get out our lego bricks

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	
		L	S
memory access 1	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?`

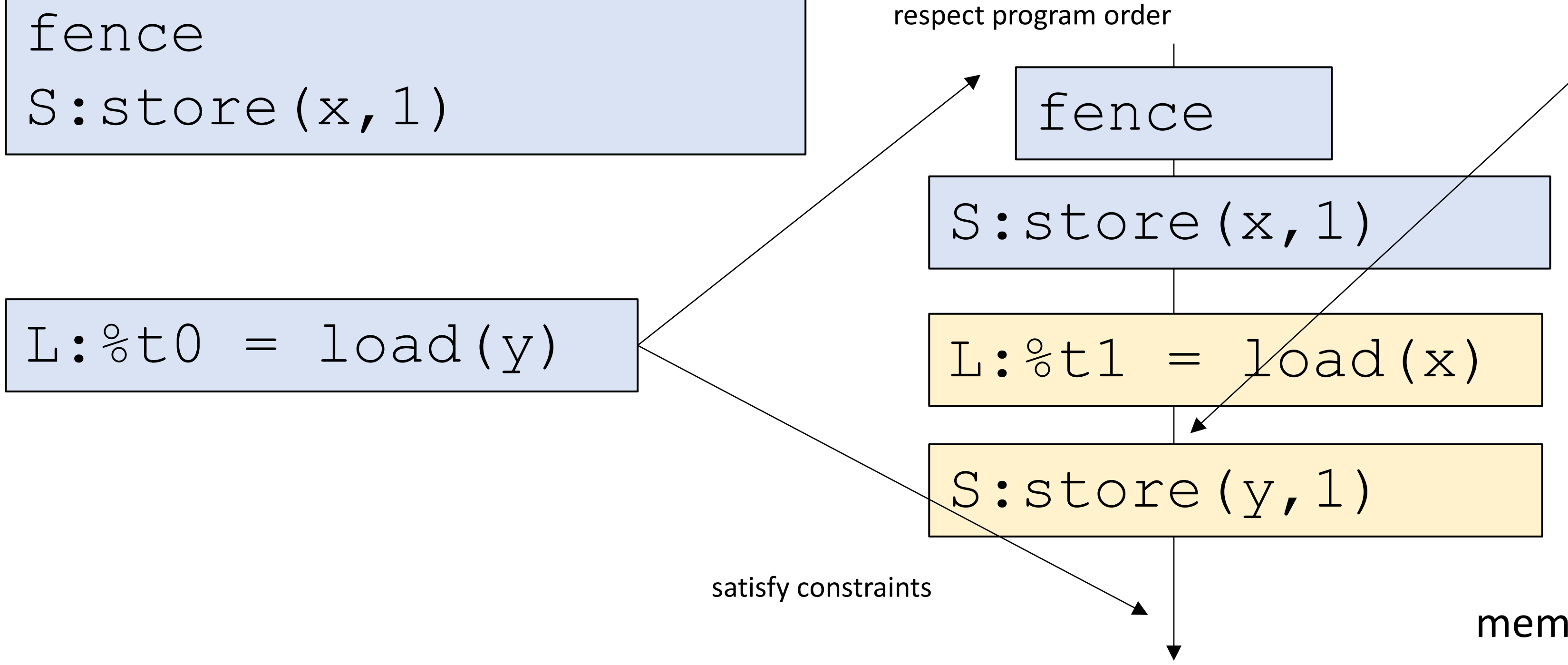
Get out our lego bricks

Thread 0:

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

Thread 1:

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



respect program order

`fence`

`S:store(x,1)`

`L:%t1 = load(x)`

`S:store(y,1)`

`L:%t0 = load(y)`

satisfy constraints

memory access 0

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

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};
```

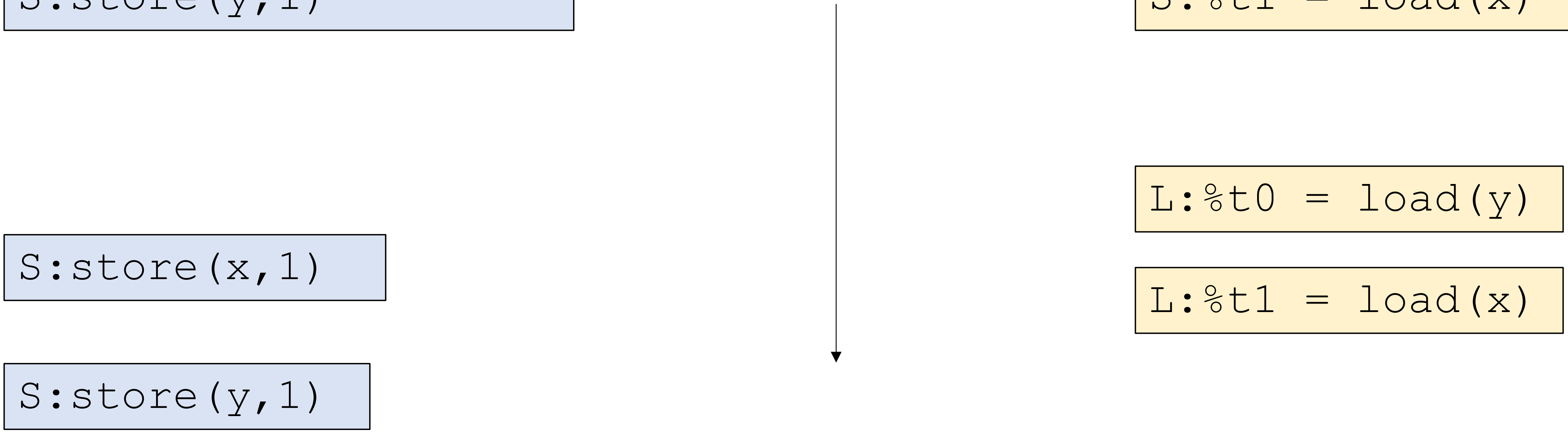
Question: can `t0 == 1 and t1 == 0`?

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`?

Thread 0:

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

start off thinking
about sequential
consistency

Thread 1:

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

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

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

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

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

Global variable:

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

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

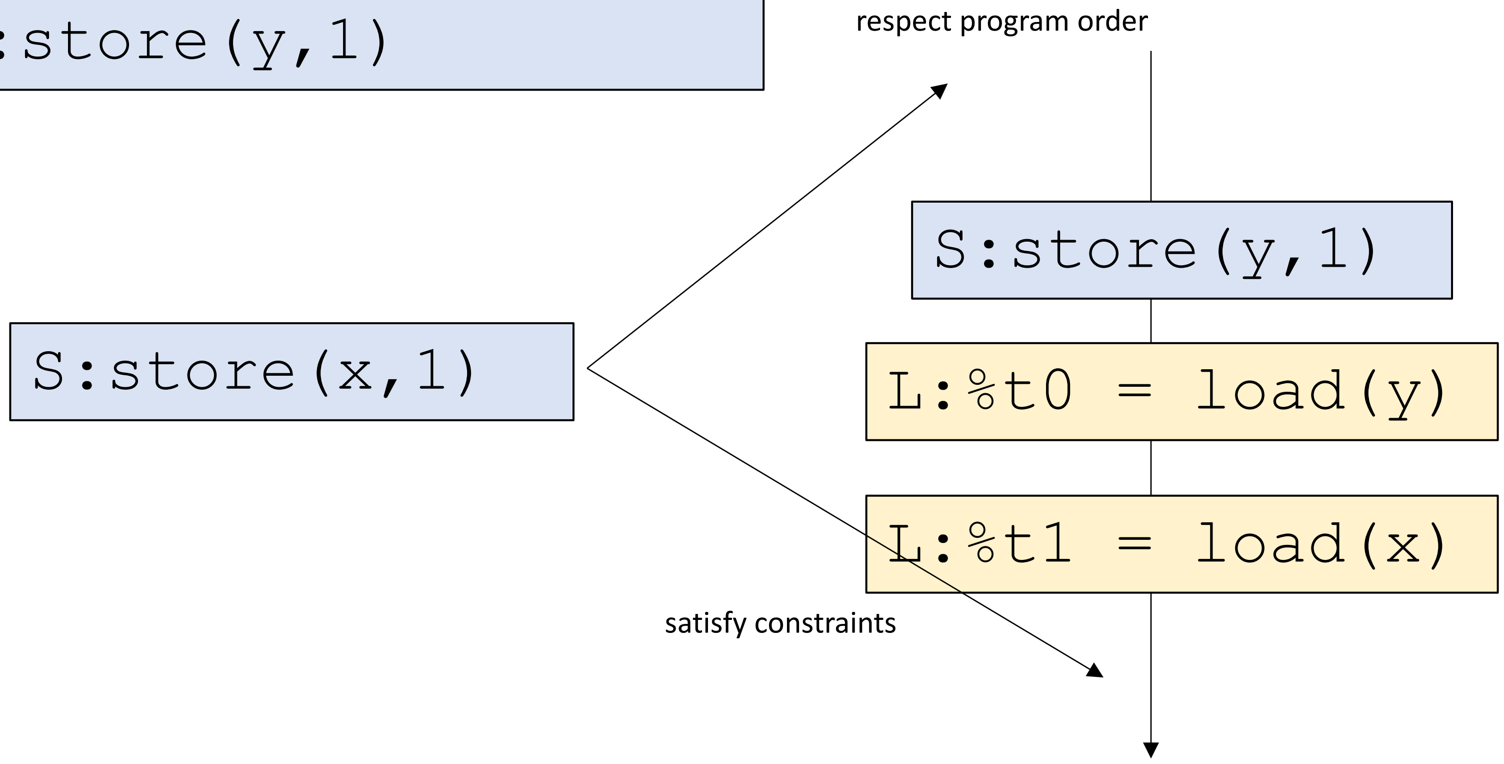
start off thinking
about sequential
consistency

Thread 0:

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

Thread 1:

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



respect program order

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

```
S:store(y,1)
L:%t0 = load(y)
L:%t1 = load(x)
```

satisfy constraints

Global variable:

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

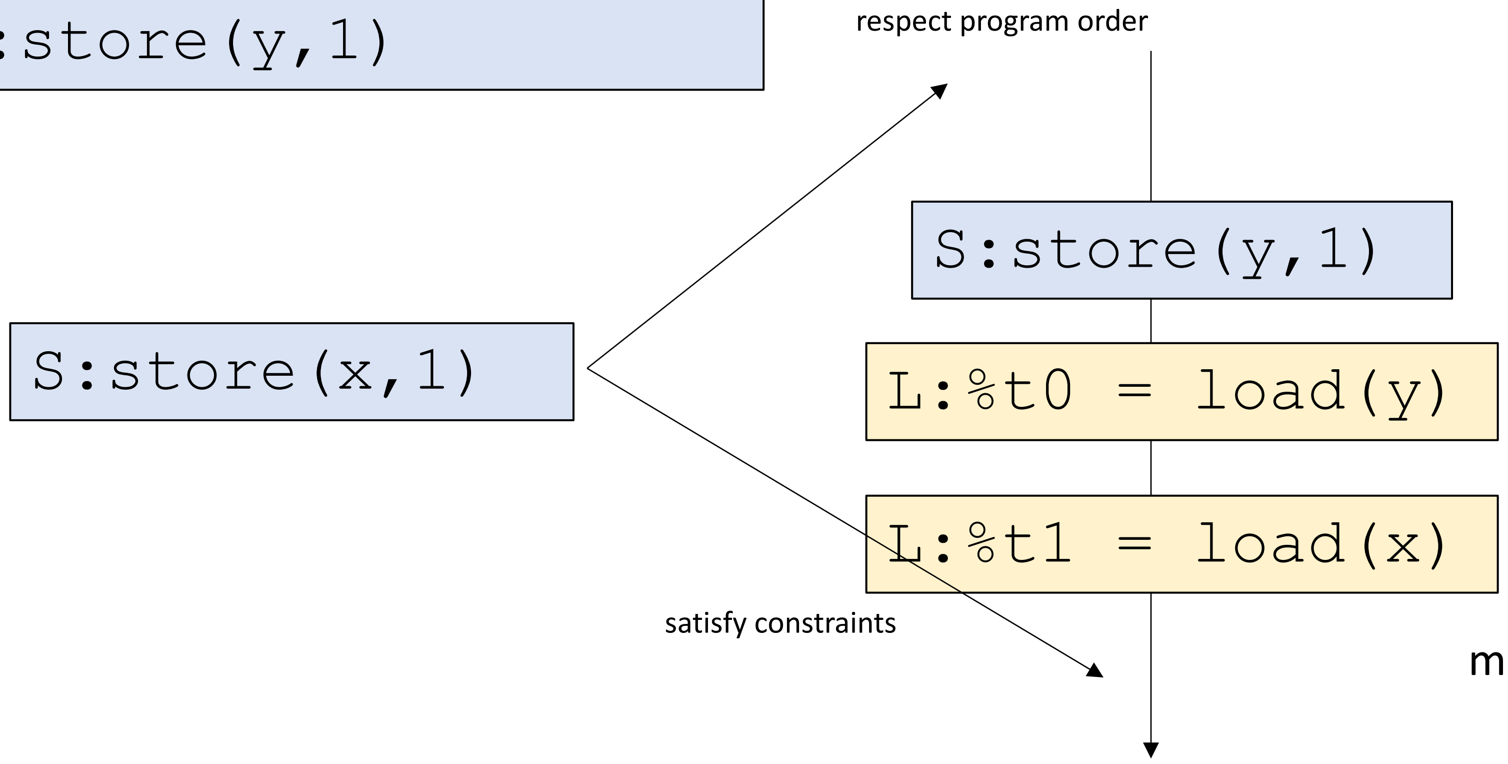
Question: can `t0 == 1 and t1 == 0?`

Thread 0:

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

Thread 1:

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



respect program order

`S:store(y,1)`

`L:%t0 = load(y)`

`L:%t1 = load(x)`

`S:store(x,1)`

satisfy constraints

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

What about TSO?

Global variable:

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

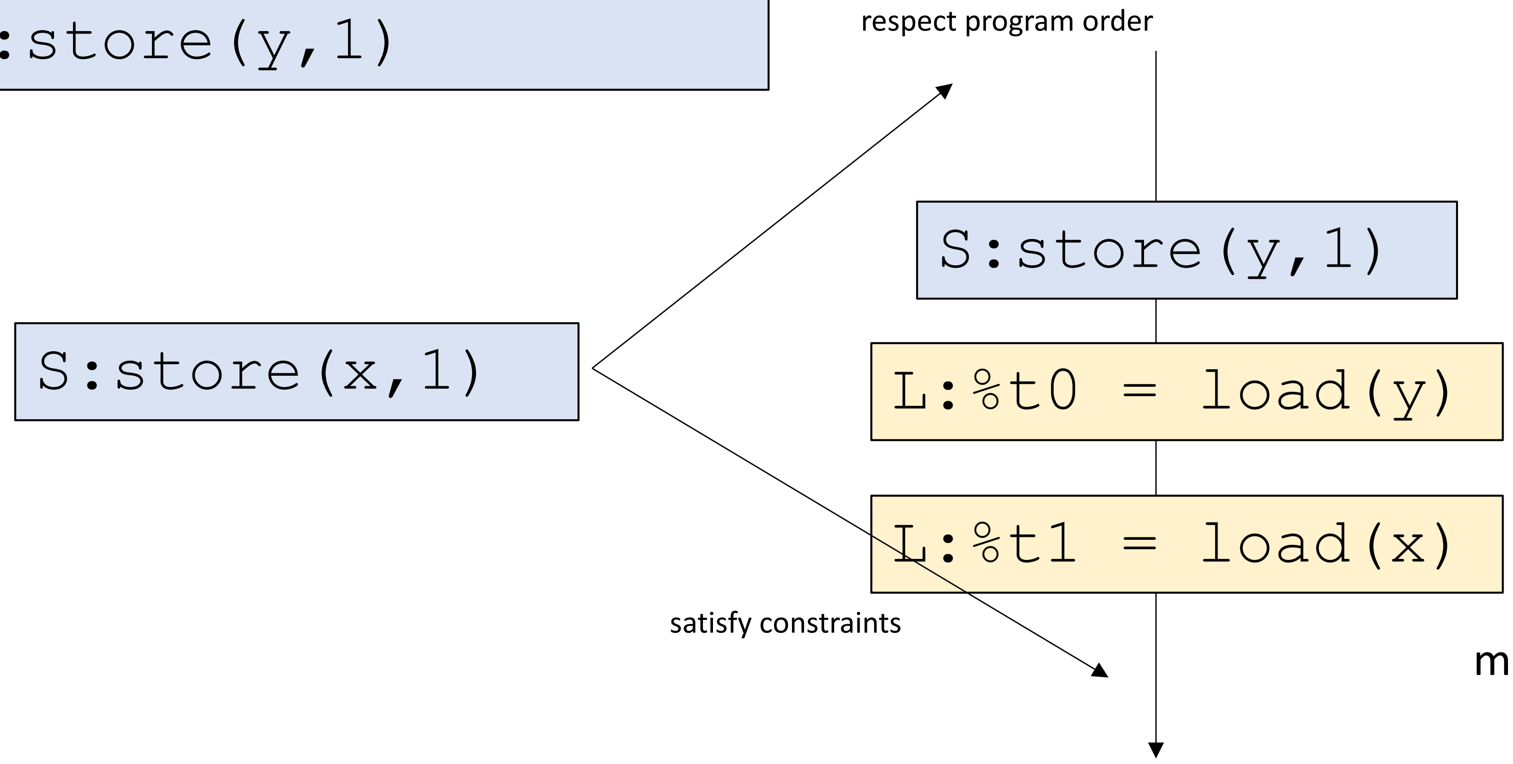
Question: can `t0 == 1 and t1 == 0?`

Thread 0:

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

Thread 1:

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



respect program order

```
S:store(y,1)
L:%t0 = load(y)
L:%t1 = load(x)
```

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

satisfy constraints

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};
```

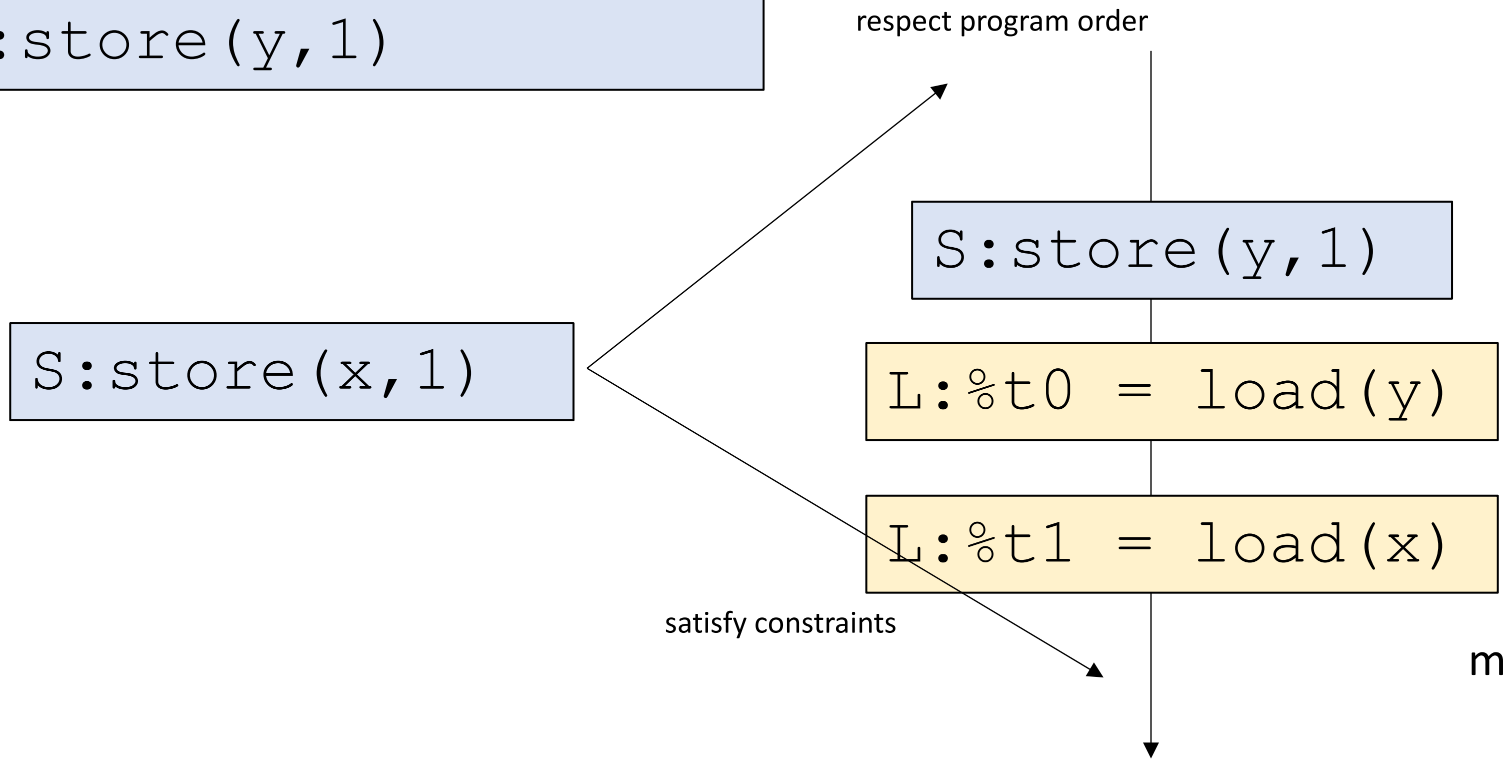
Question: can `t0 == 1 and t1 == 0`?

Thread 0:

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

Thread 1:

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



respect program order

`S:store(y,1)`

`L:%t0 = load(y)`

`L:%t1 = load(x)`

`S:store(x,1)`

satisfy constraints

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

What about PSO?

Global variable:

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

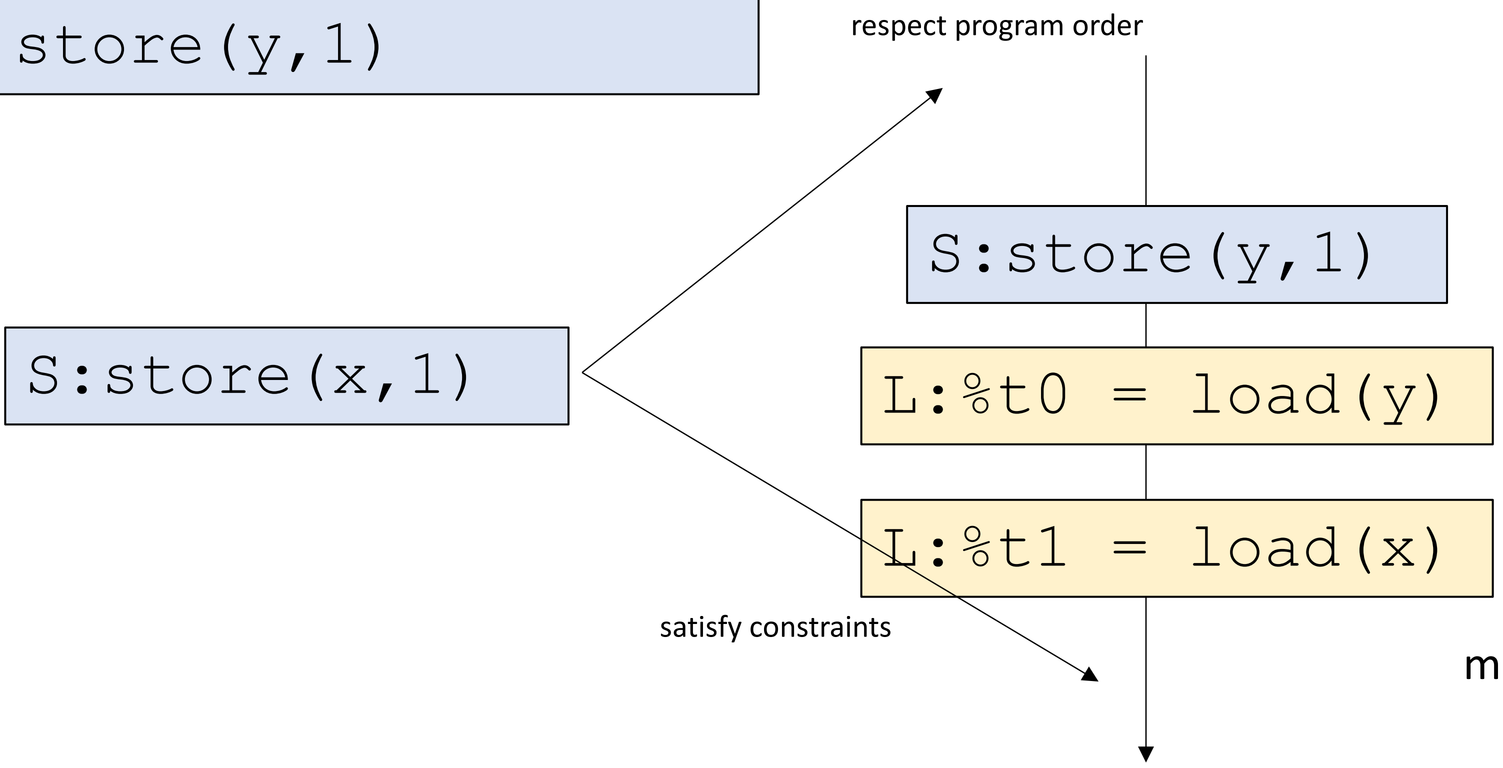
Question: can `t0 == 1 and t1 == 0?`

Thread 0:

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

Thread 1:

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



respect program order

```
S:store(y,1)
L:%t0 = load(y)
L:%t1 = load(x)
```

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

satisfy constraints

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

What about PSO?

Global variable:

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

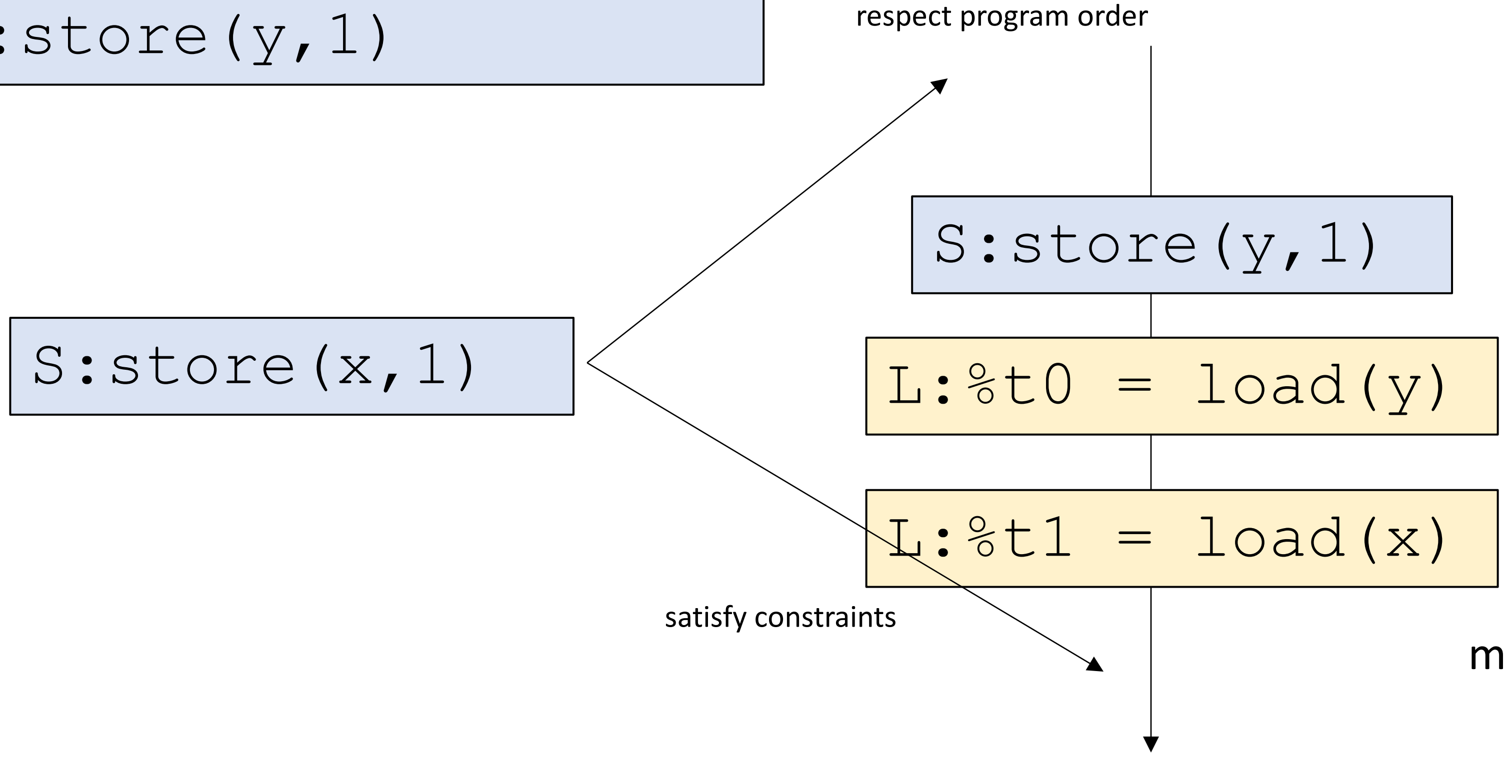
Question: can `t0 == 1 and t1 == 0?`

Thread 0:

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

Thread 1:

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



respect program order

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

```
S:store(y,1)
L:%t0 = load(y)
L:%t1 = load(x)
```

satisfy constraints

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};
```

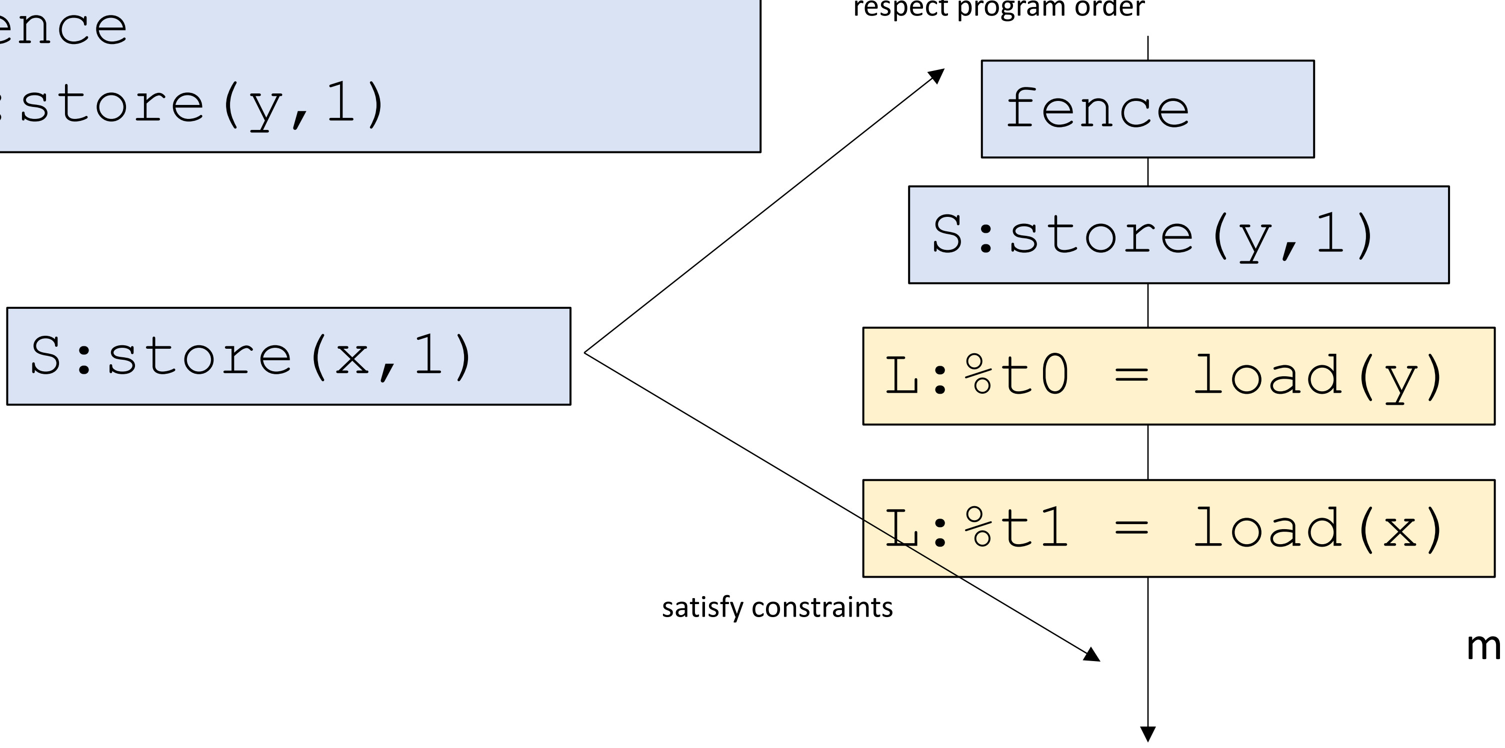
Question: can `t0 == 1 and t1 == 0?`

Thread 0:

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

Thread 1:

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



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

Now it is disallowed in PSO

Global variable:

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

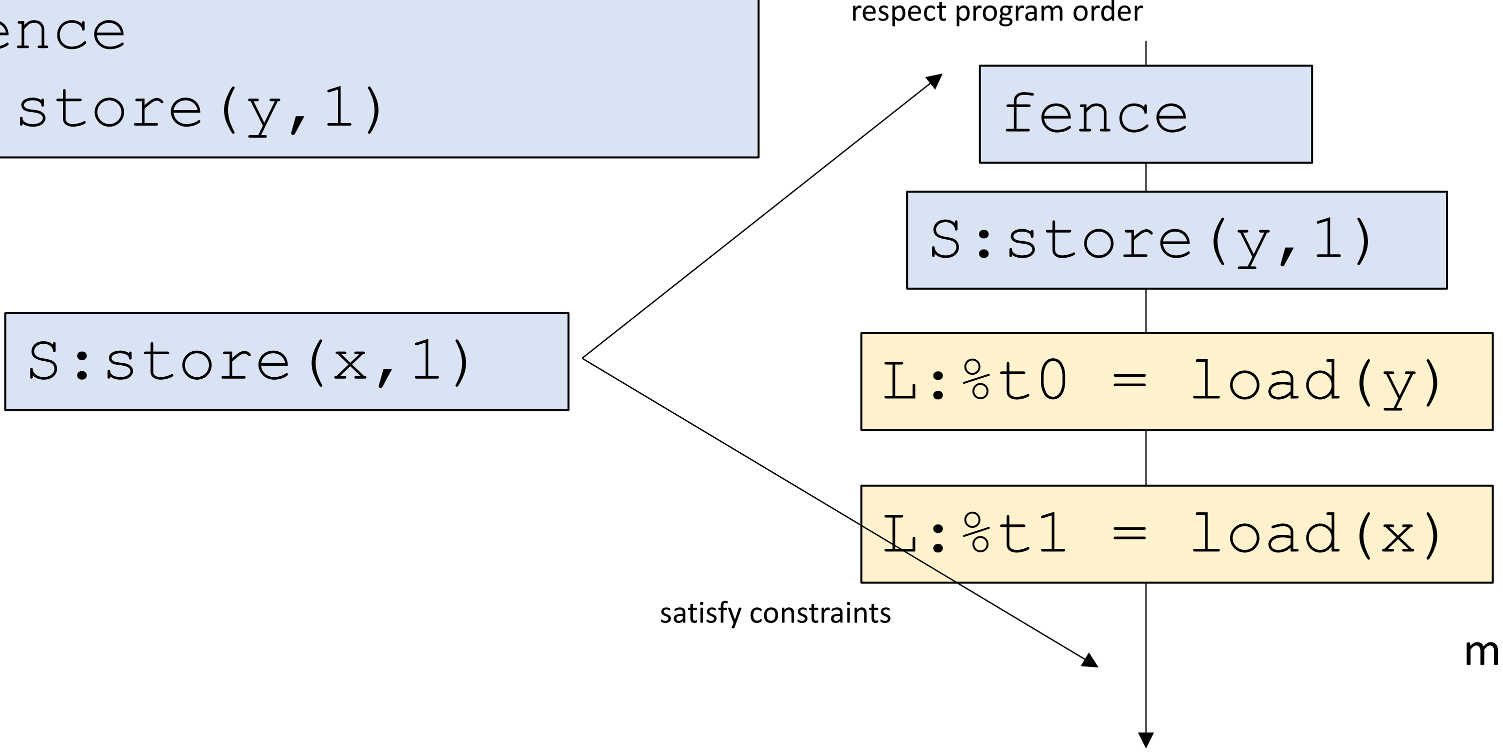
Question: can `t0 == 1 and t1 == 0?`

Thread 0:

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

Thread 1:

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



respect program order

```
fence
S:store(y,1)
L:%t0 = load(y)
L:%t1 = load(x)
```

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

satisfy constraints

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

What about RMO?

Global variable:

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

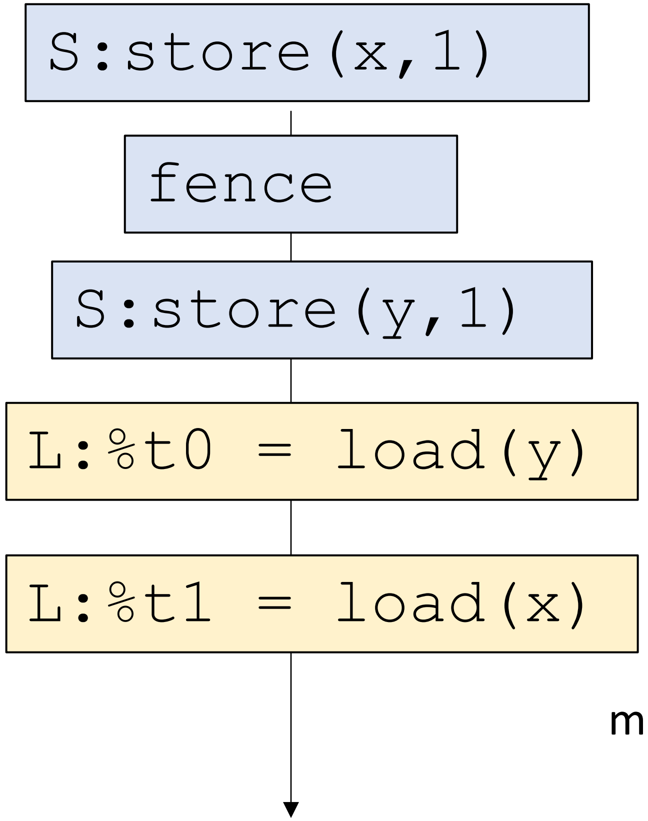
Question: can `t0 == 1 and t1 == 0?`

Thread 0:

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

Thread 1:

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



```
S:store(x,1)
fence
S:store(y,1)
L:%t0 = load(y)
L:%t1 = load(x)
```

memory access 0

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

What about RMO?

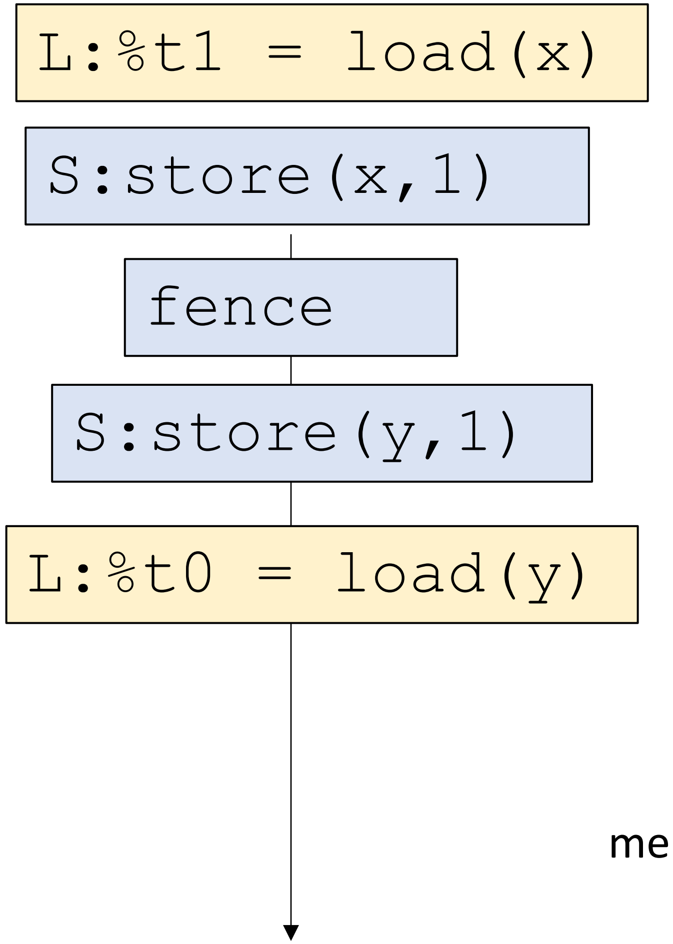
Global variable:

```
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)
```



```
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 0	
		L	S
memory access 1	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};
```

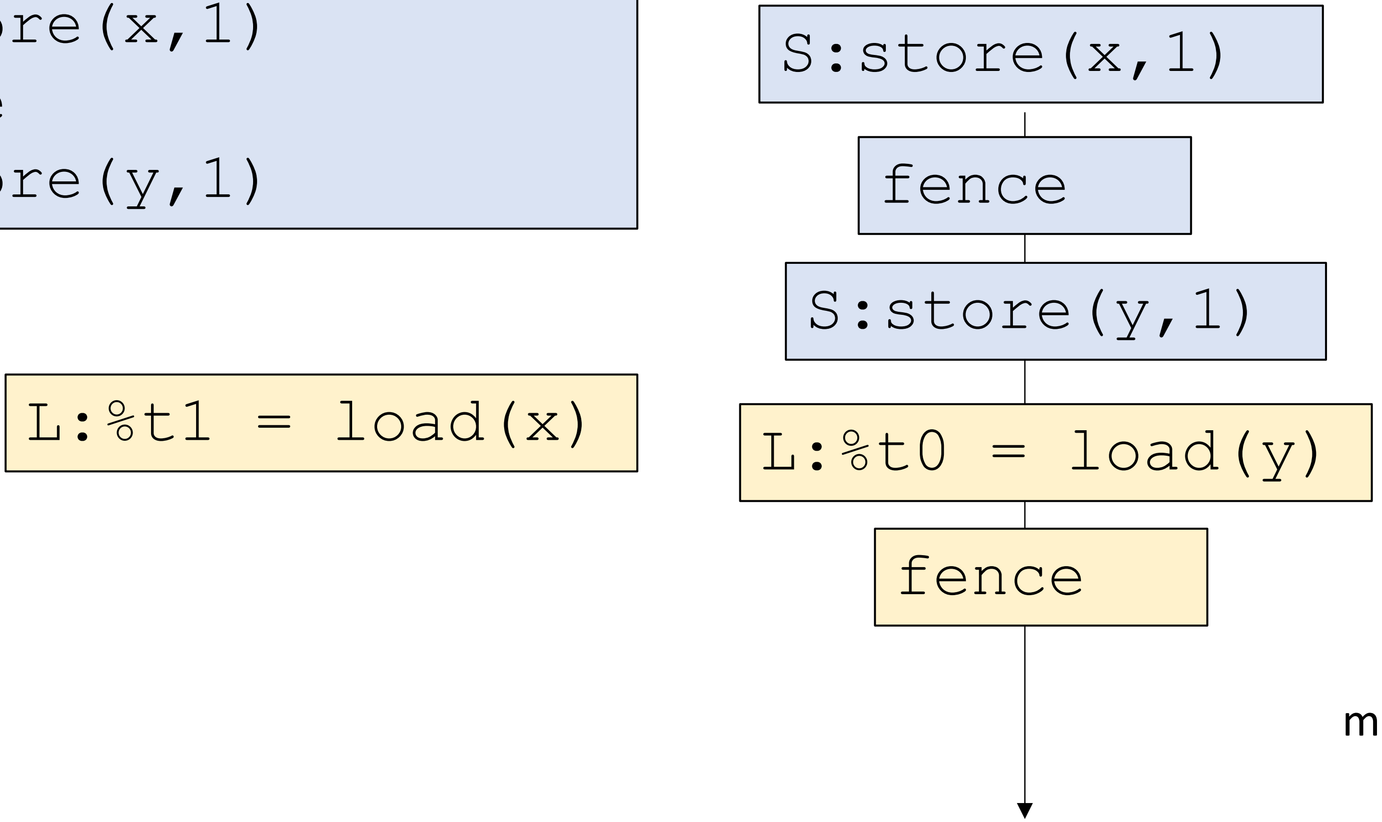
Question: can `t0 == 1 and t1 == 0?`

Thread 0:

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

Thread 1:

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



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

What about RMO? add a fence

Global variable:

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

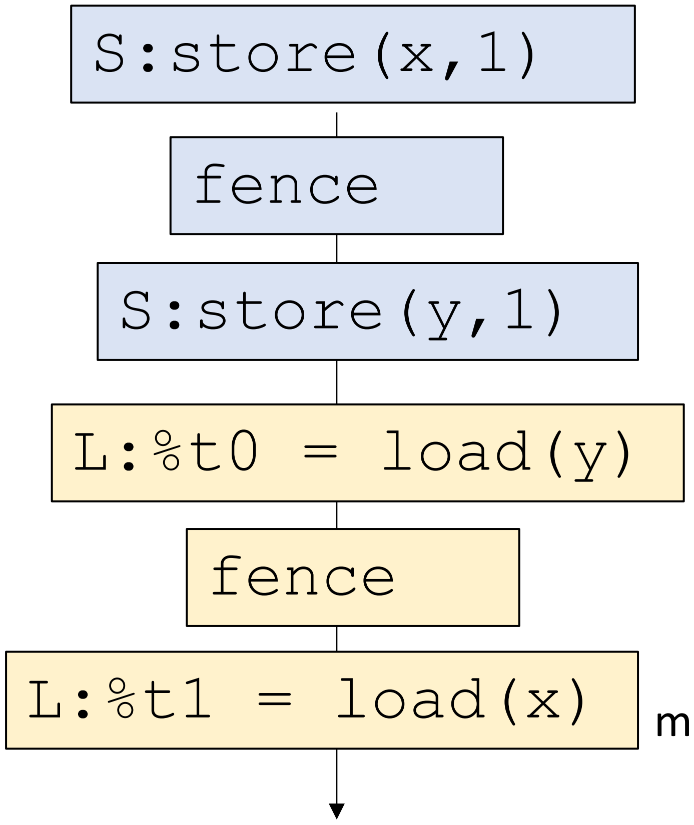
Question: can `t0 == 1 and t1 == 0?`

Thread 0:

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

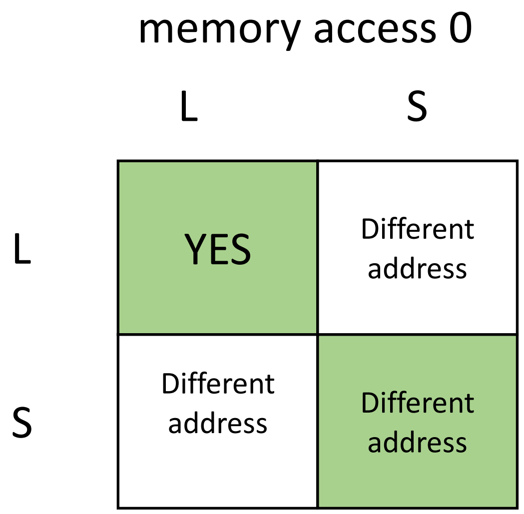
Thread 1:

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



```
S:store(x,1)
fence
S:store(y,1)
L:%t0 = load(y)
fence
L:%t1 = load(x)
```

memory access 1



memory access 0

	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
    - Surprisingly 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
    - Surprisingly 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

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

Where do programming languages fit in?

- One of the highest priorities of a programming language
    - Write once, run everywhere

C++11 atomic operation compilation

start with both of the grids for the two different memory models

language
C++11 (sequential consistency)

	L	S
L	NO	NO
S	NO	NO

target machine

	L	S
L	?	?
S	?	?

C++11 atomic operation compilation

start with both both of the grids for the two different memory models

language
C++11 (sequential consistency)

	L	S
L	NO	NO
S	NO	NO

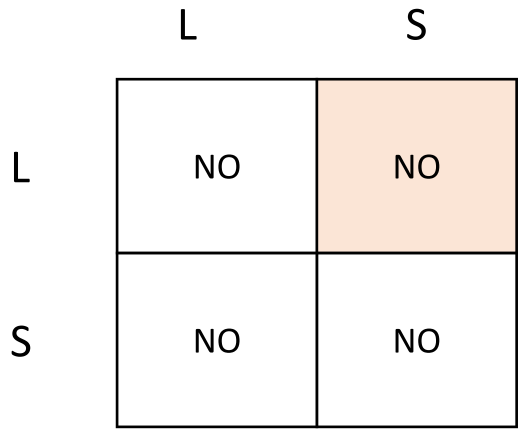
target machine
TSO (x86)

	L	S
L	NO	different address
S	NO	No

C++11 atomic operation compilation

start with both both of the grids for the two different memory models

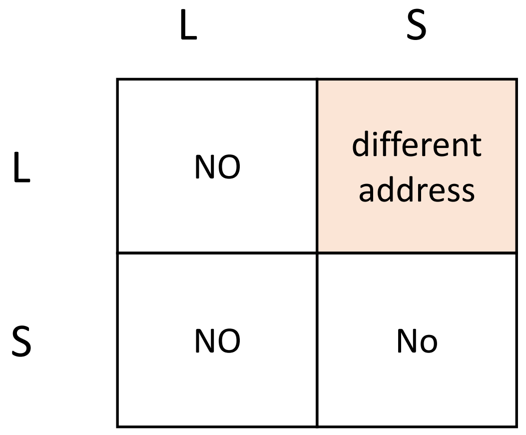
language
C++11 (sequential consistency)



	L	S
L	NO	NO
S	NO	NO

find mismatch

target machine
TSO (x86)



	L	S
L	NO	different address
S	NO	No

C++11 atomic operation compilation

start with both both of the grids for the two different memory models

language
C++11 (sequential consistency)

	L	S
L	NO	NO
S	NO	NO

find mismatch

Two options:

make sure stores
are not reordered
with later loads

make sure loads
are not reordered
with earlier stores

target machine
TSO (x86)

	L	S
L	NO	different address
S	NO	No

C++11 atomic operation compilation

start with both both of the grids for the two different memory models

language
C++11 (sequential consistency)

	L	S
L	NO	NO
S	NO	NO

C++

x.store(1);

ISA

store(x,1);
fence;

or

target machine
TSO (x86)

	L	S
L	NO	different address
S	NO	No

C++11 atomic operation compilation

start with both both of the grids for the two different memory models

language
C++11 (sequential consistency)

	L	S
L	NO	NO
S	NO	NO

C++ → ISA

x.store(1); → store(x,1); fence;

or

z = x.load() → fence; %z = load(x);

target machine
TSO (x86)

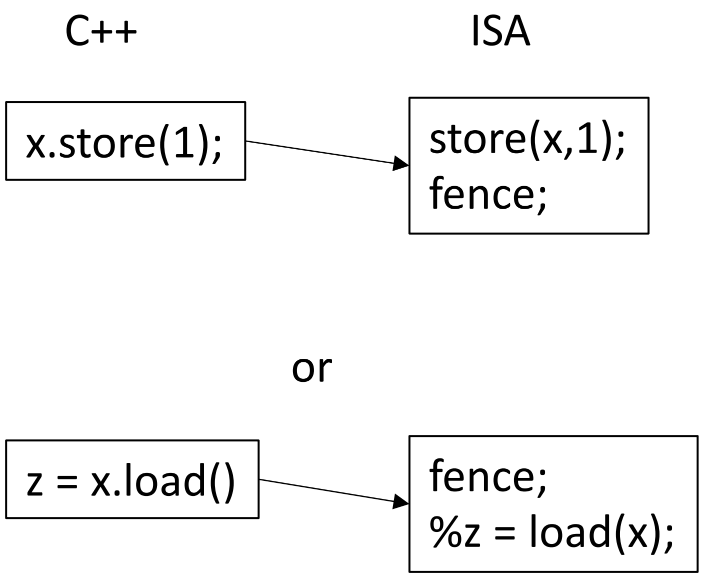
	L	S
L	NO	different address
S	NO	No

C++11 atomic operation compilation

start with both both of the grids for the two different memory models

language
C++11 (sequential consistency)

	L	S
L	NO	NO
S	NO	NO



C++

ISA

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

```
store(x,1);
fence;
```

or

```
z = x.load()
```

```
fence;
%z = load(x);
```

target machine
TSO (x86)

	L	S
L	NO	different address
S	NO	No

This should help you see why you want to reduce the number of atomic load/stores in your program

C++11 atomic operation compilation

start with both both of the grids for the two different memory models

language
C++11 (sequential consistency)

	L	S
L	NO	NO
S	NO	NO

How about this one?

target machine
PSO

	L	S
L	NO	different address
S	NO	different address

C++11 atomic operation compilation

start with both both of the grids for the two different memory models

language
C++11 (sequential consistency)

	L	S
L	NO	NO
S	NO	NO

target machine
PSO

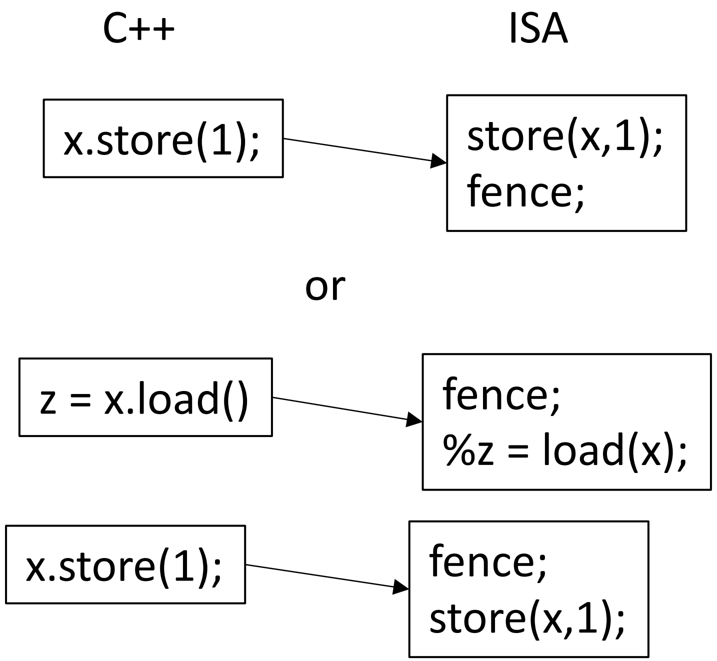
	L	S
L	NO	different address
S	NO	different address

C++11 atomic operation compilation

start with both both of the grids for the two different memory models

language
C++11 (sequential consistency)

	L	S
L	NO	NO
S	NO	NO



C++ → ISA

`x.store(1);` → `store(x,1); fence;`

or

`z = x.load()` → `fence; %z = load(x);`

`x.store(1);` → `fence; store(x,1);`

target machine
PSO

	L	S
L	NO	different address
S	NO	different address

Memory orders

- Atomic operations take an additional “memory order” argument
  - `memory_order_seq_cst` - default
  - `memory_order_relaxed` - weakest

Relaxed memory order

language
C++11 (sequential consistency)

	L	S
L	NO	NO
S	NO	NO

language
C++11 (memory_order_relaxed)

	L	S
L	different address	different address
S	different address	different address

basically no orderings except for accesses to the same address

Compiling memory order relaxed

language
C++11 (memory_order_relaxed)

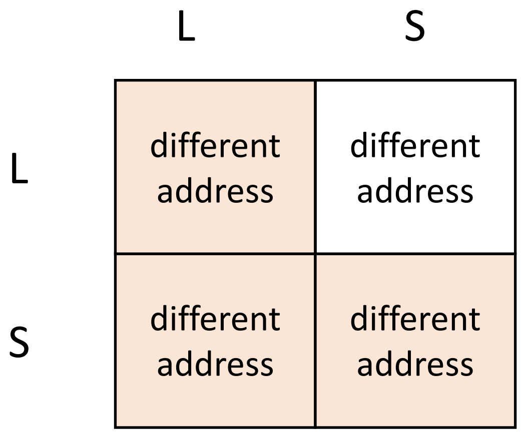
	L	S
L	different address	different address
S	different address	different address

target machine
TSO (x86)

	L	S
L	NO	different address
S	NO	No

Compiling memory order relaxed

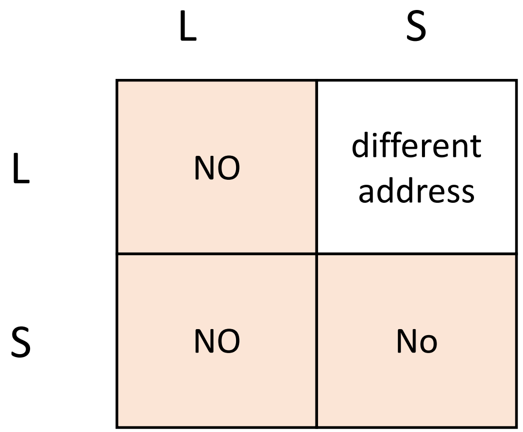
language
C++11 (memory_order_relaxed)



	L	S
L	different address	different address
S	different address	different address

lots of mismatches!

target machine
TSO (x86)



	L	S
L	NO	different address
S	NO	No

Compiling memory order relaxed

language
C++11 (memory_order_relaxed)

	L	S
L	different address	different address
S	different address	different address

lots of mismatches!

But language is more relaxed than machine

so no fences are needed

target machine
TSO (x86)

	L	S
L	NO	different address
S	NO	No

Compiling memory order relaxed

Do any of the ISA memory models need any fences for relaxed memory order?

language
C++11 (memory_order_relaxed)

	L	S
L	different address	different address
S	different address	different address

	L	S
L	NO	Different address
S	NO	NO

TSO

	L	S
L	NO	Different address
S	NO	Different address

PSO

	L	S
L	YES	Different address
S	Different address	Different address

RMO

Memory order relaxed

- Very few use-cases! Be very careful when using it
  - Peeking at values (later accessed using a heavier memory order)
  - Counting (e.g. number of finished threads in work stealing)
  - ***DO NOT USE FOR QUEUE INDEXES***

More memory orders: we will not discuss in class

- Atomic operations take an additional “memory order” argument
  - `memory_order_seq_cst` - default
  - `memory_order_relaxed` - weakest

- More memory orders (useful for mutex implementations):
  - `memory_order_acquire`
  - `memory_order_release`

- EVEN MORE memory orders (complicated: in most research it is omitted)
  - `memory_order_consume`

A cautionary tale

Consider the following example: a graphics program where each thread wants to display a triangle; the display is a queue (not thread safe)

Thread 0:

```
m.lock();
display.enq(triangle0);
m.unlock();
```

Thread 1:

```
m.lock();
display.enq(triangle1);
m.unlock();
```

Consider the following example: a graphics program where each thread wants to display a triangle; the display is a queue (not thread safe)

Thread 0:

```
m.lock();
display.enq(triangle0);
m.unlock();
```

Thread 1:

```
m.lock();
display.enq(triangle1);
m.unlock();
```

We know how lock and unlock are implemented

Consider the following example: a graphics program where each thread wants to display a triangle; the display is a queue (not thread safe)

Thread 0:

```
SPIN:CAS(mutex,0,1);
display.enq(triangle0);
store(mutex,0);
```

Thread 1:

```
SPIN:CAS(mutex,0,1);
display.enq(triangle1);
store(mutex,0);
```

We know how lock and unlock are implemented
We also know how a queue is implemented

Consider the following example: a graphics program where each thread wants to display a triangle; the display is a queue (not thread safe)

Thread 0:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 1:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle1);
store(head, %i+1);
store(mutex,0);
```

We know how lock and unlock are implemented
We also know how a queue is implemented

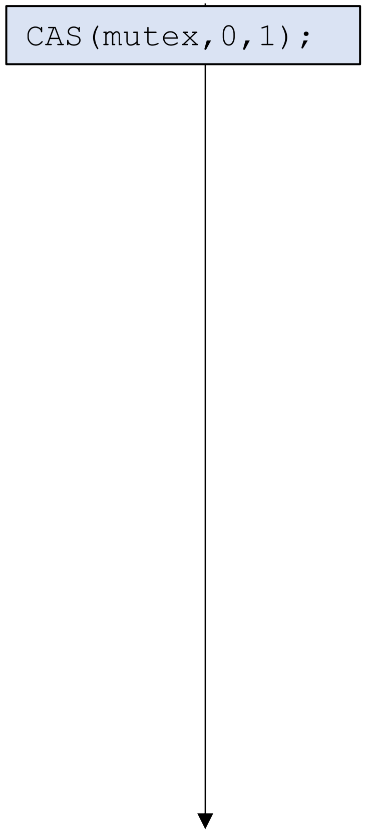
What is an execution?

Thread 0:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 1:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle1);
store(head, %i+1);
store(mutex,0);
```



`CAS(mutex,0,1);`

if blue goes first
it gets to complete
its critical section
while thread 1 is spinning

Thread 0:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 1:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle1);
store(head, %i+1);
store(mutex,0);
```

```
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 0:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 1:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle1);
store(head, %i+1);
store(mutex,0);
```

```
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

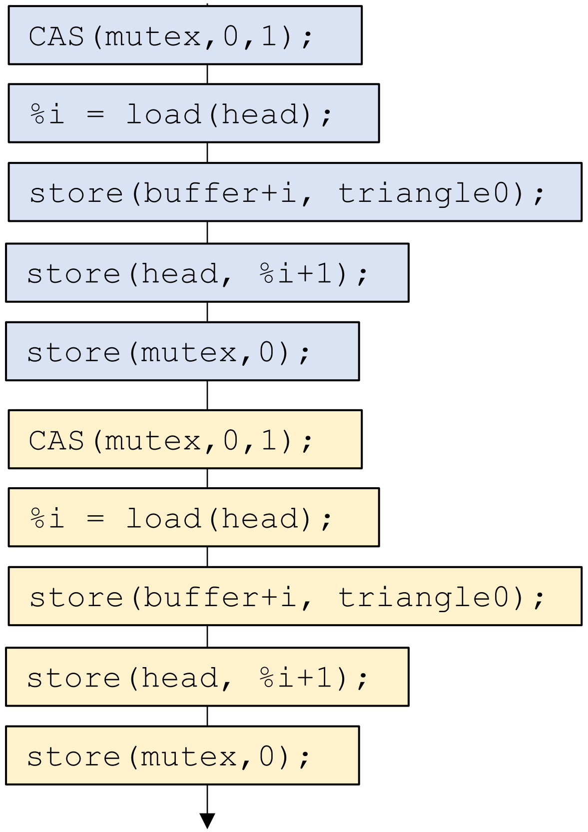
now yellow gets a change to go

Thread 0:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 1:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle1);
store(head, %i+1);
store(mutex,0);
```



```
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

now yellow gets a change to go

Thread 0:

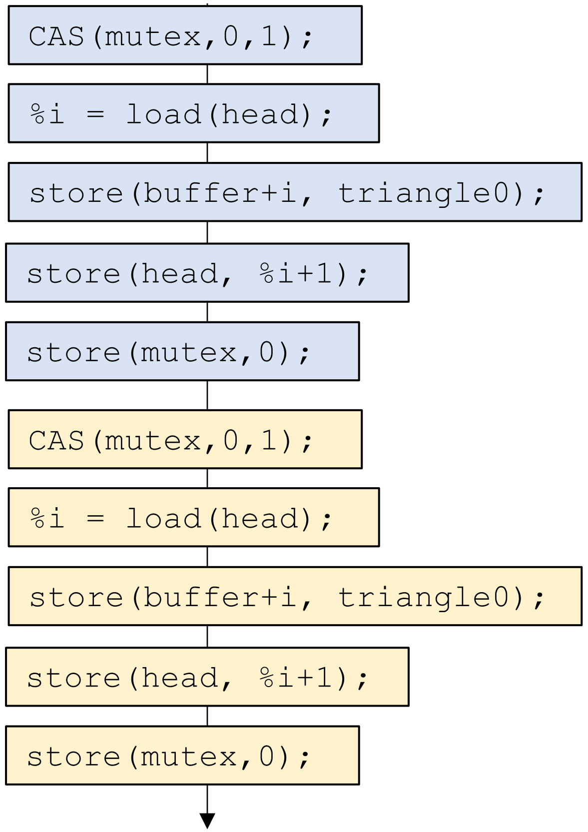
```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 1:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle1);
store(head, %i+1);
store(mutex,0);
```

what can happen in a PSO memory model?

	L	S
L	NO	Different address
S	NO	Different address



```
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 0:

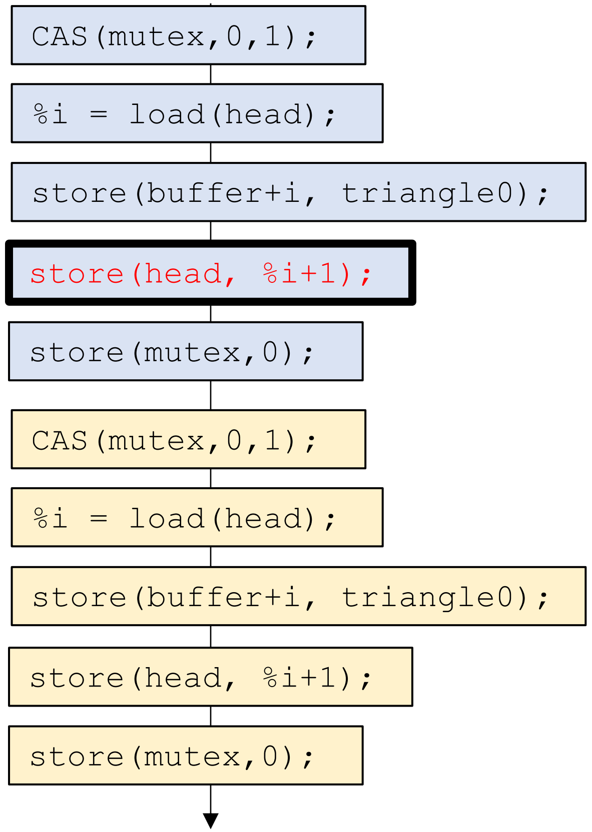
```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 1:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle1);
store(head, %i+1);
store(mutex,0);
```

what can happen in a PSO memory model?

	L	S
L	NO	Different address
S	NO	Different address



```
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 0:

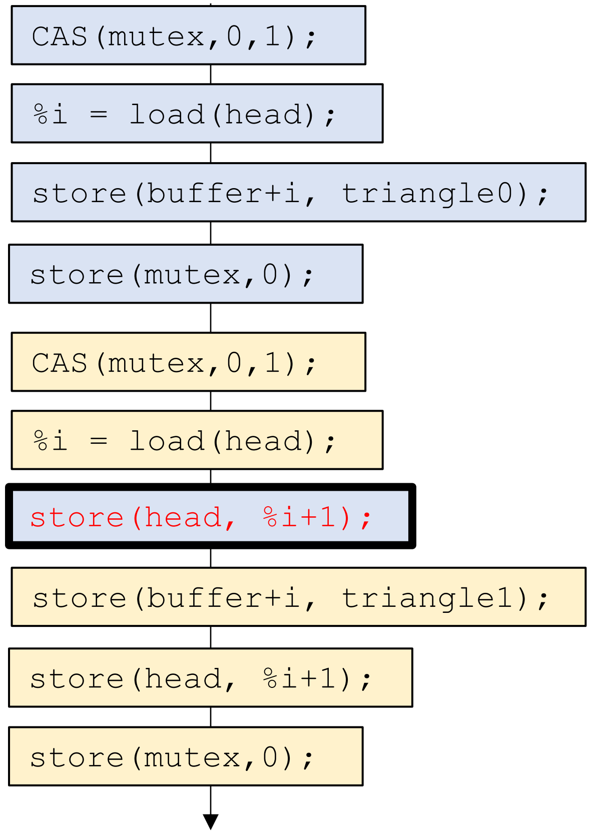
```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 1:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle1);
store(head, %i+1);
store(mutex,0);
```

what can happen in a PSO memory model?

	L	S
L	NO	Different address
S	NO	Different address



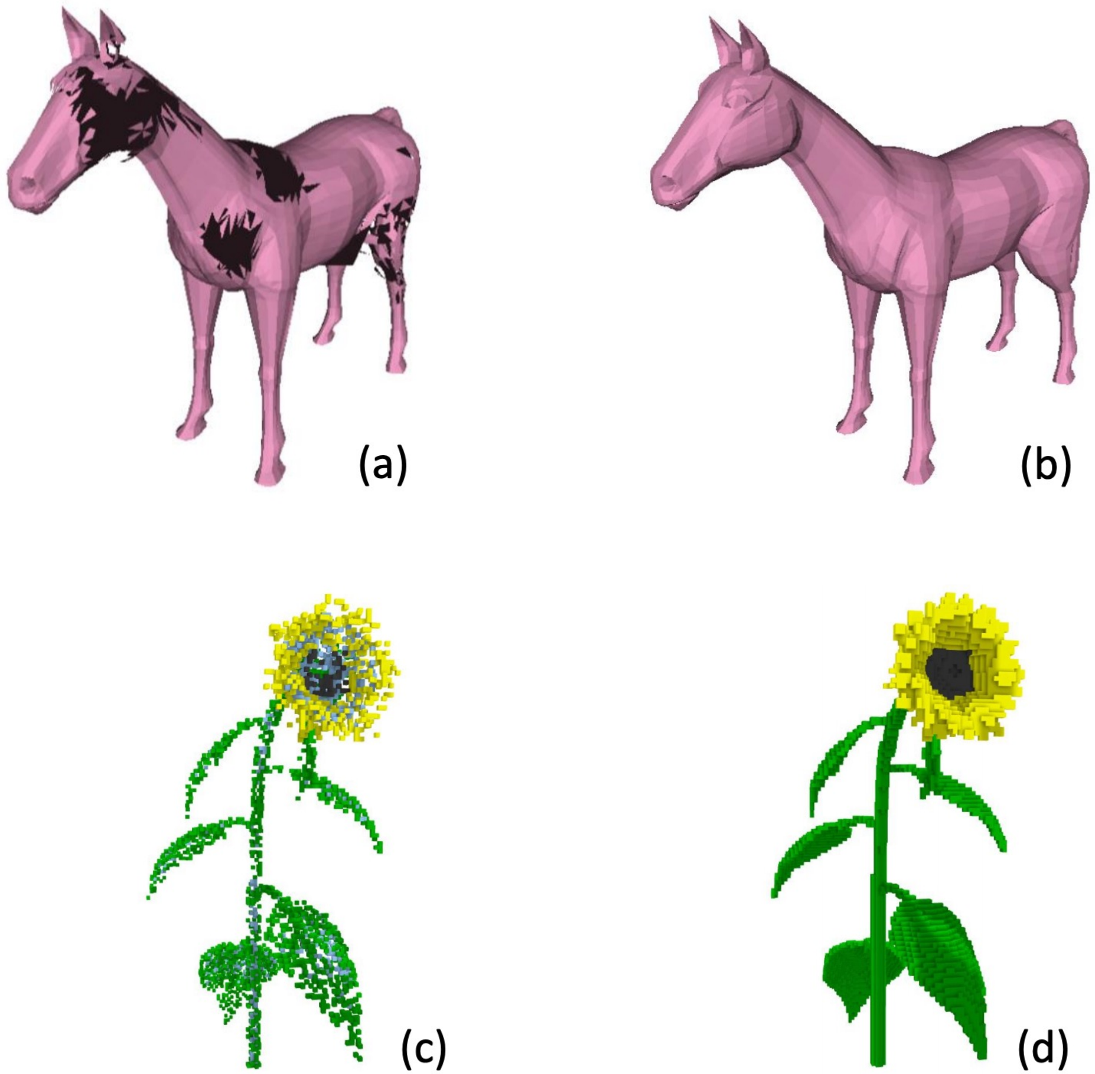
```
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(mutex,0);
CAS(mutex,0,1);
%i = load(head);
store(head, %i+1);
store(buffer+i, triangle1);
store(head, %i+1);
store(mutex,0);
```

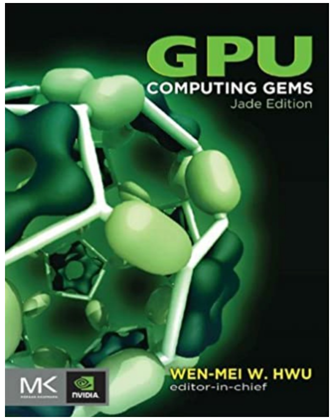
What just happened if this store moves?

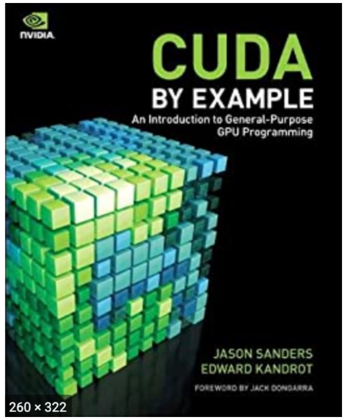
Nvidia in 2015

- Nvidia architects implemented a weak memory model
- Nvidia programmers expected a strong memory model
- Mutexes implemented without fences!

Nvidia in 2015







bug found in two
Nvidia textbooks

We implemented
a side-channel attack
that made the bugs
appear more frequently

These days Nvidia has
a very well-specified
memory model!

Thread 0:

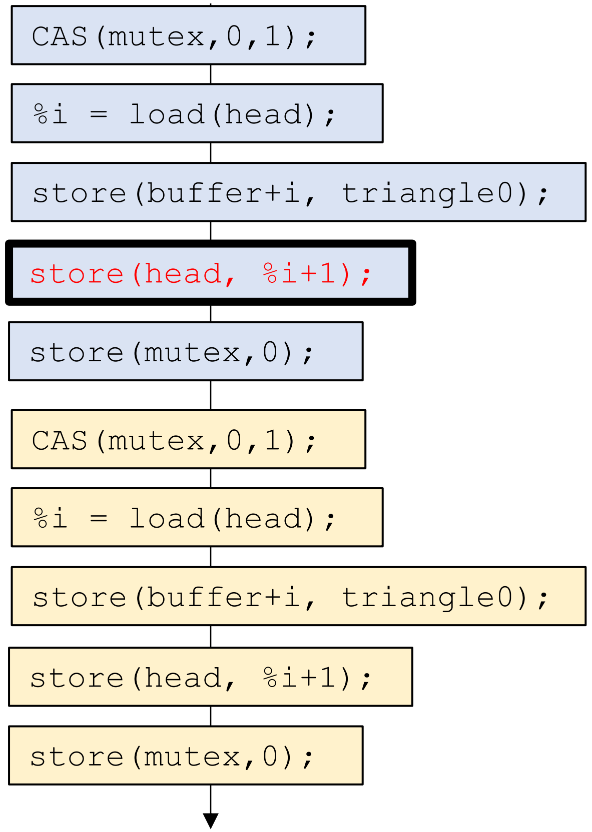
```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

Thread 1:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle1);
store(head, %i+1);
store(mutex,0);
```

what can happen in a PSO memory model?

	L	S
L	NO	Different address
S	NO	Different address



```
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

How to fix the issue?

Thread 0:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
fence;
store(mutex,0);
```

unlock contains fence before store!

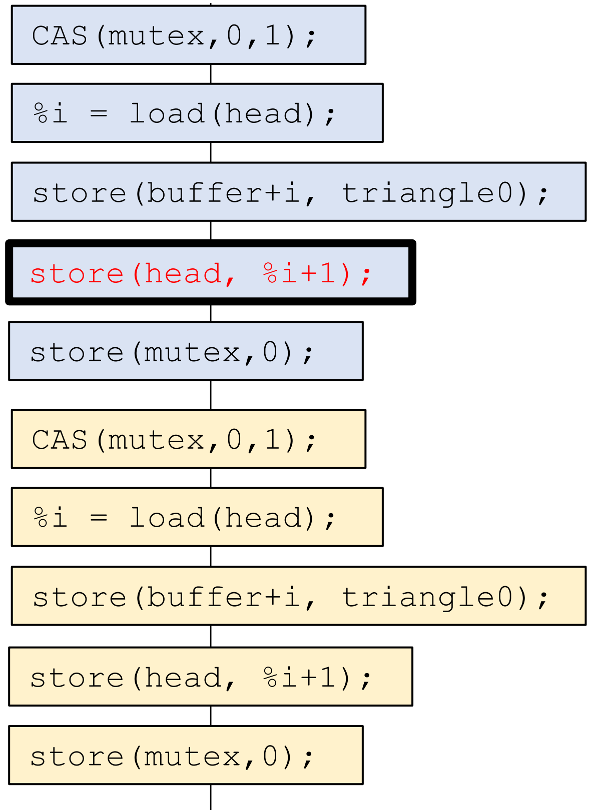
Thread 1:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle1);
store(head, %i+1);
fence;
store(mutex,0);
```

unlock contains fence before store!

what can happen in a PSO memory model?

	L	S
L	NO	Different address
S	NO	Different address



```
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
store(mutex,0);
```

How to fix the issue?

your unlock function should contain a fence!

Thread 0:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
fence;
store(mutex,0);
```

unlock contains fence before store!

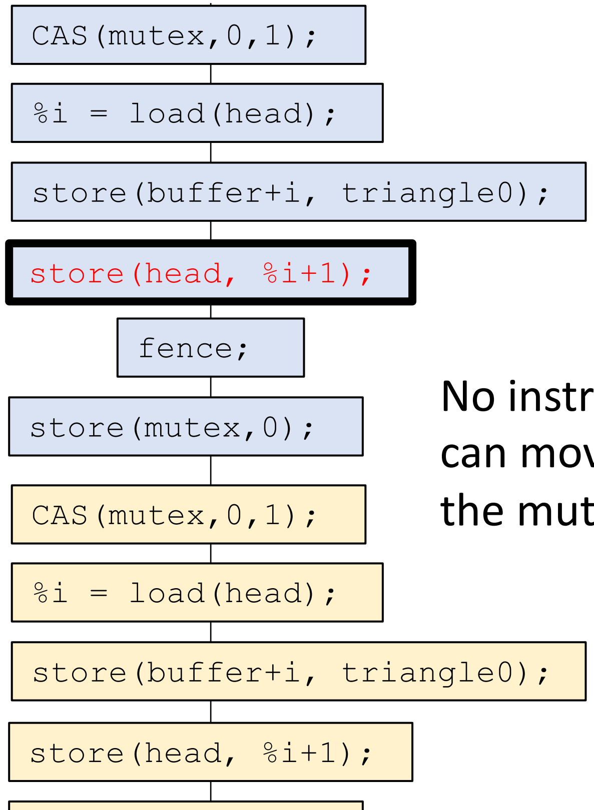
Thread 1:

```
SPIN:CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle1);
store(head, %i+1);
fence;
store(mutex,0);
```

unlock contains fence before store!

what can happen in a PSO memory model?

	L	S
L	NO	Different address
S	NO	Different address



```
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
fence;
store(mutex,0);
CAS(mutex,0,1);
%i = load(head);
store(buffer+i, triangle0);
store(head, %i+1);
```

No instructions can move after the mutex store!

How to fix the issue?

your unlock function should contain a fence!

Memory Model Strength

- If one memory model M0 allows more relaxed behaviors than another memory model M1, then M0 is more *relaxed* (or *weaker*) than M1.

- It is safe to run a program written for M0 on M1. But not vice versa

	L	S
L	NO	Different address
S	NO	NO

TSO

	L	S
L	NO	Different address
S	NO	Different address

PSO

	L	S
L	YES	Different address
S	Different address	Different address

RMO

Memory Model Strength

- Many times specifications are weaker than implementations:
  - A chip might document PSO, but implement TSO:

	L	S
L	NO	Different address
S	NO	NO

TSO

	L	S
L	NO	Different address
S	NO	Different address

PSO

	L	S
L	YES	Different address
S	Different address	Different address

RMO