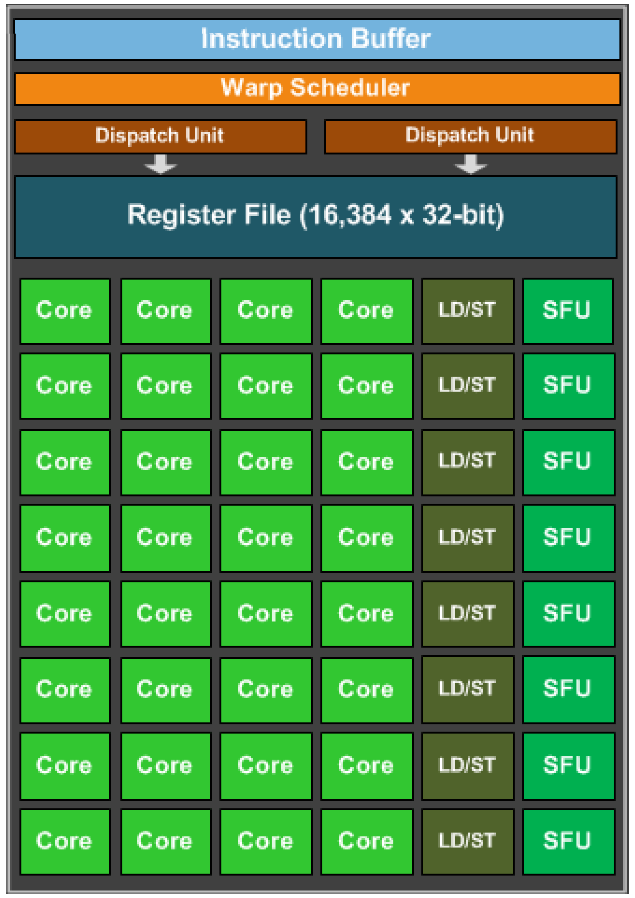
CSE113: Parallel Programming

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



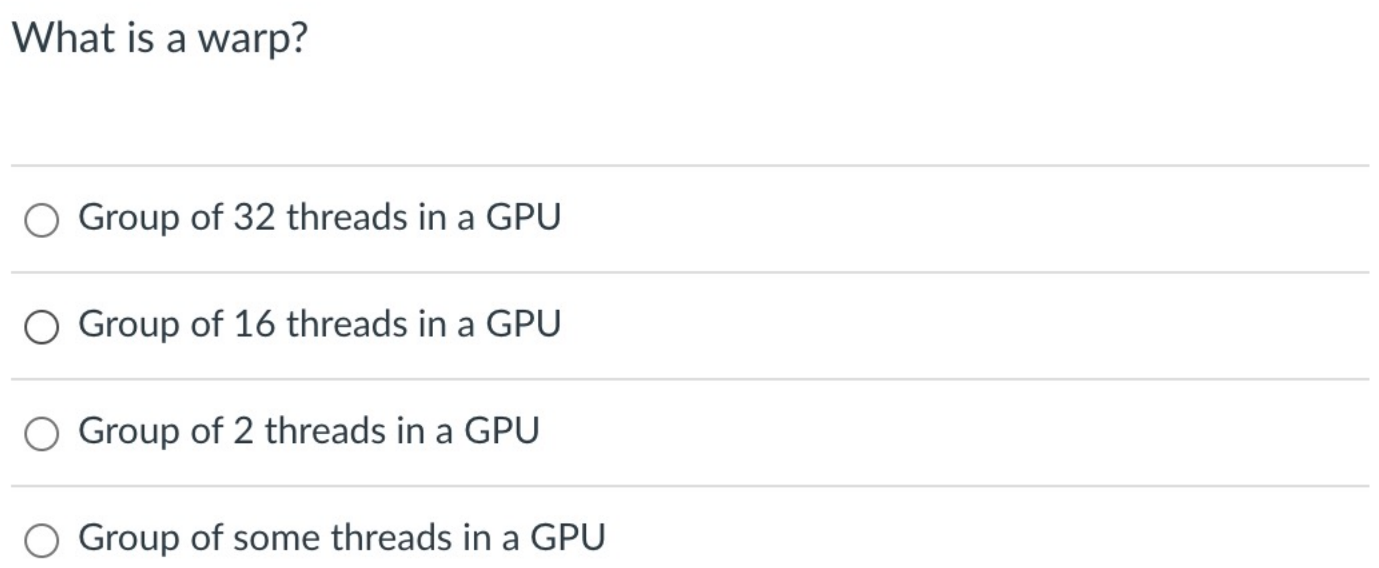
Announcements

Grading

- Grades for HW 3 should be out by the end of the week
- HW 4 is released.

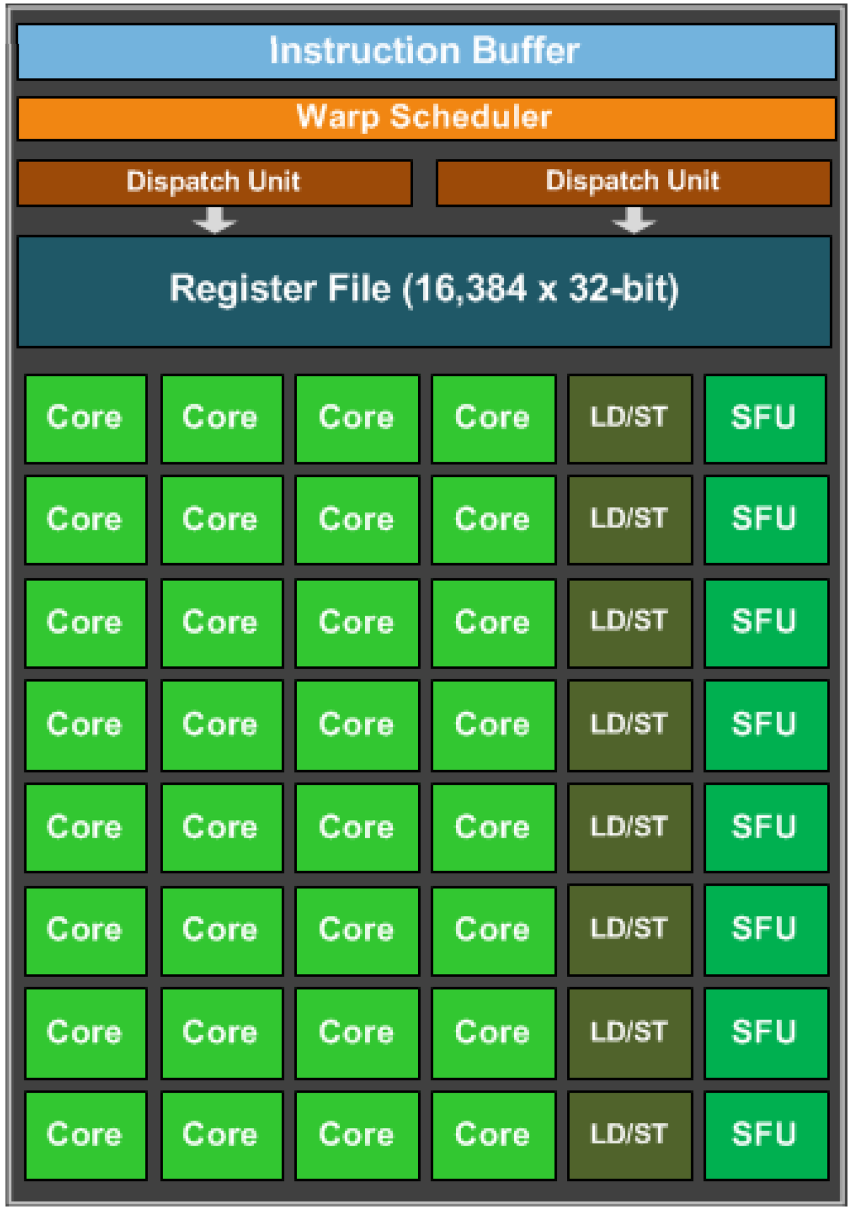
Previous quiz + Review

Previous quiz + Review

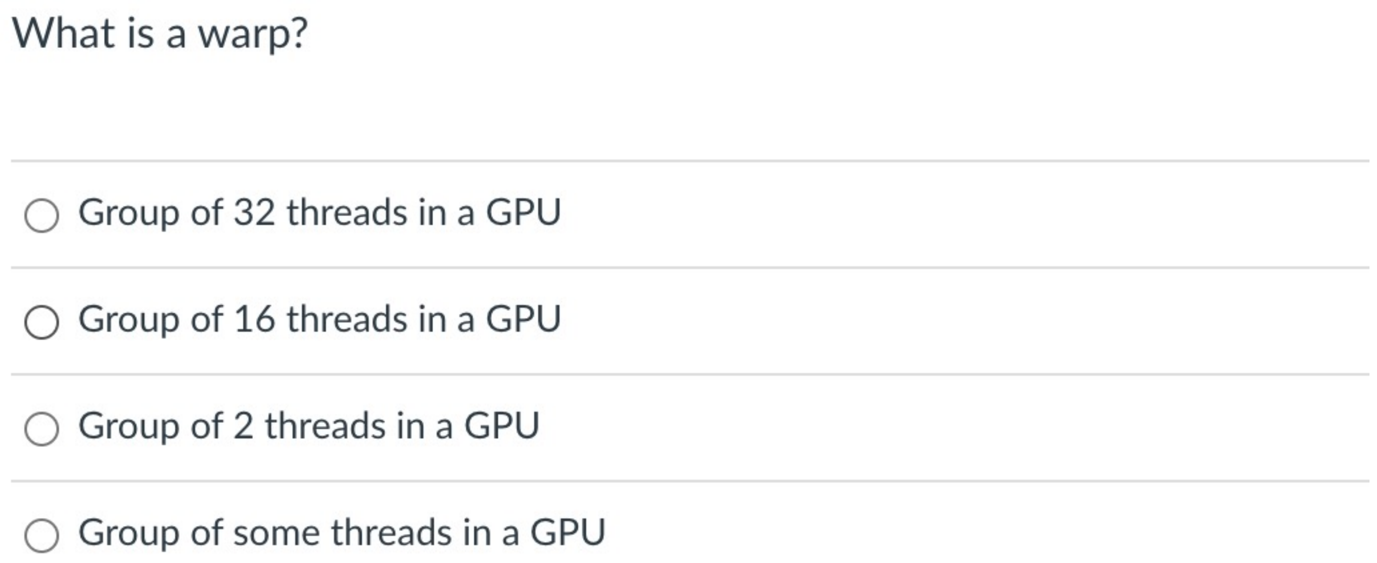


What is a warp?

- ○ 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
LD/ST
SFU

Previous quiz + Review



What is a warp?

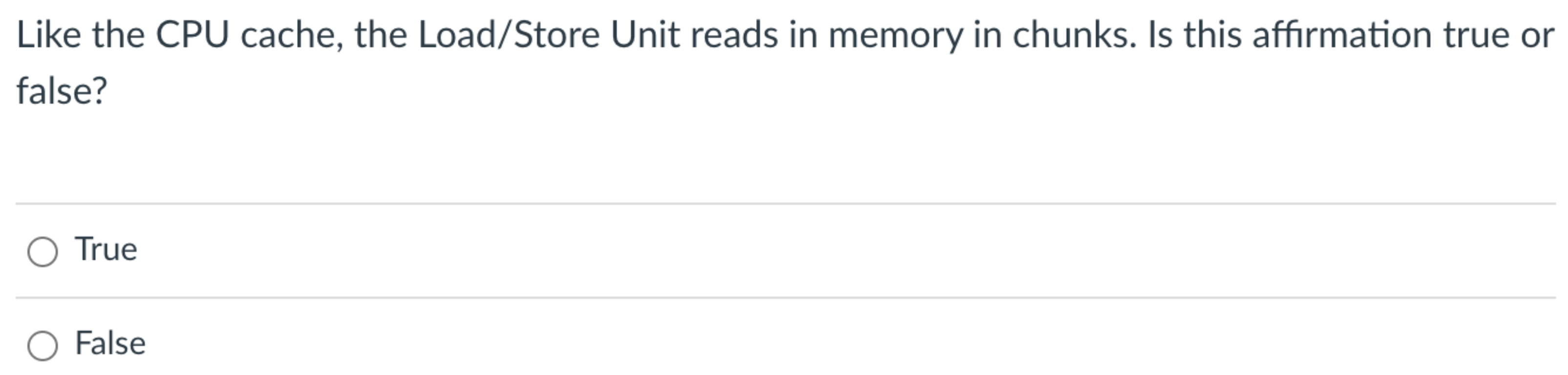
-> ○ Group of 32 threads in a GPU

○ 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

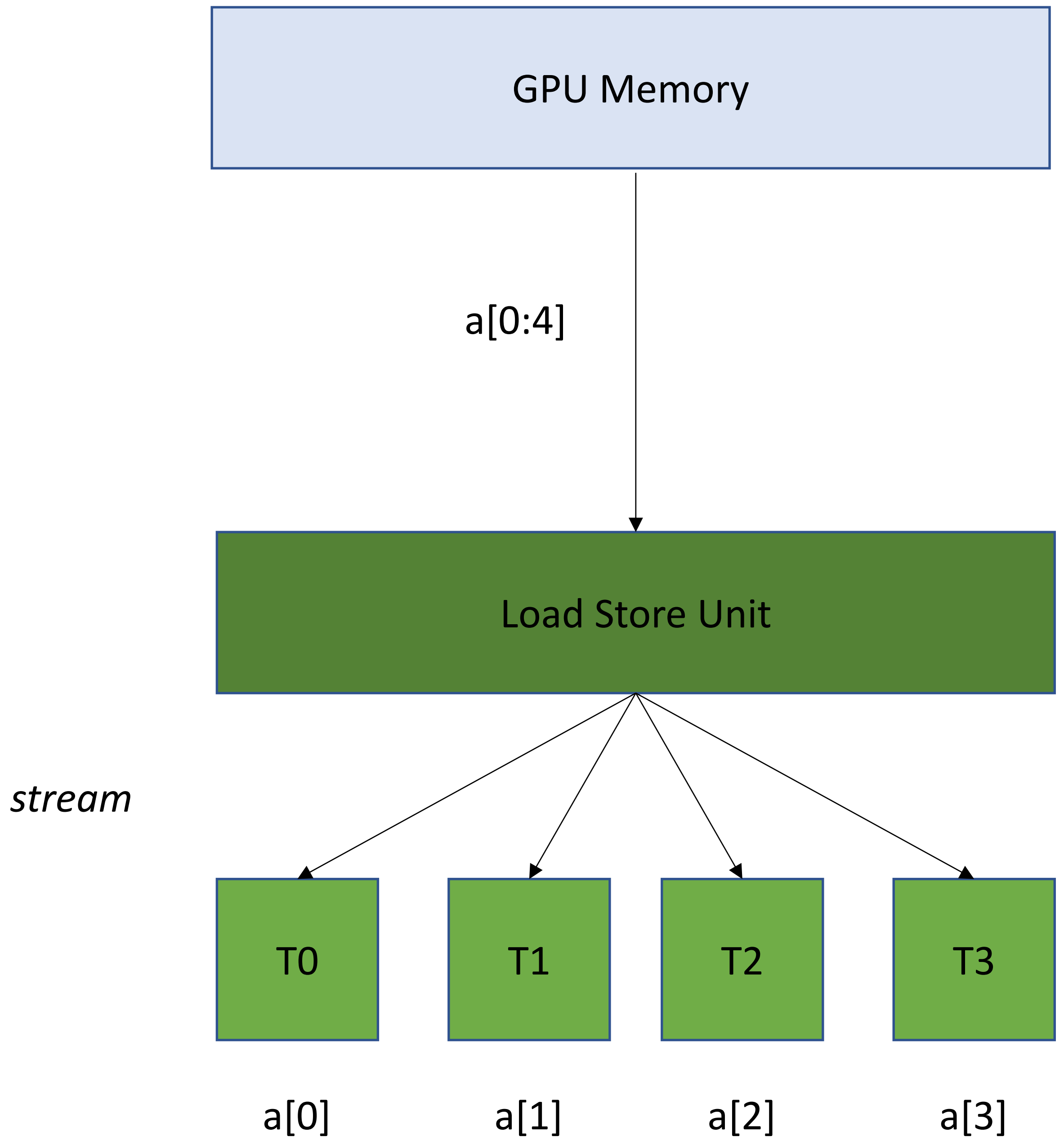
○ False

4 cores are accessing memory. What can happen

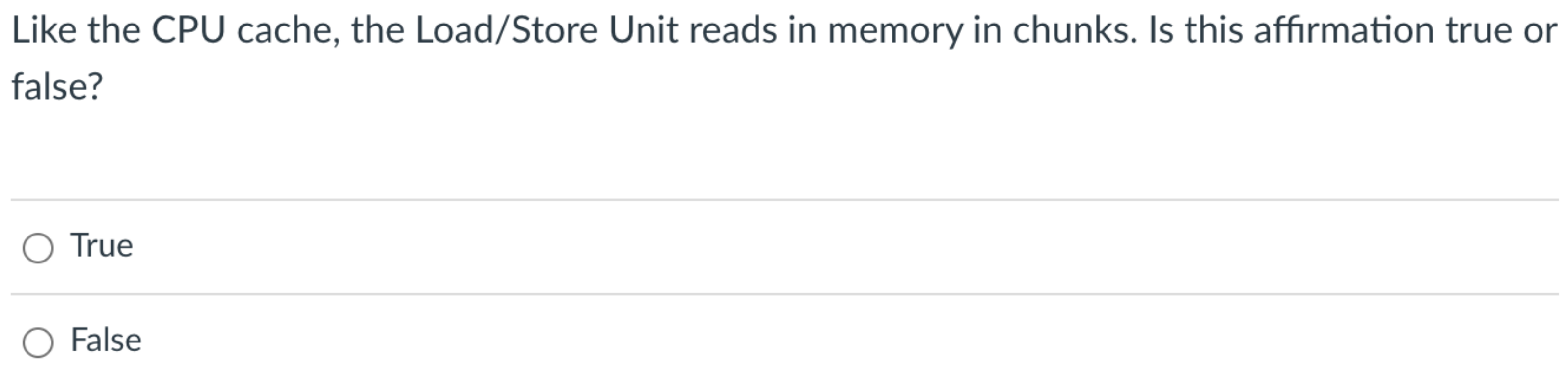
Read contiguous values
Like the CPU cache, the Load/Store Unit reads in memory in chunks. 16 bytes

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

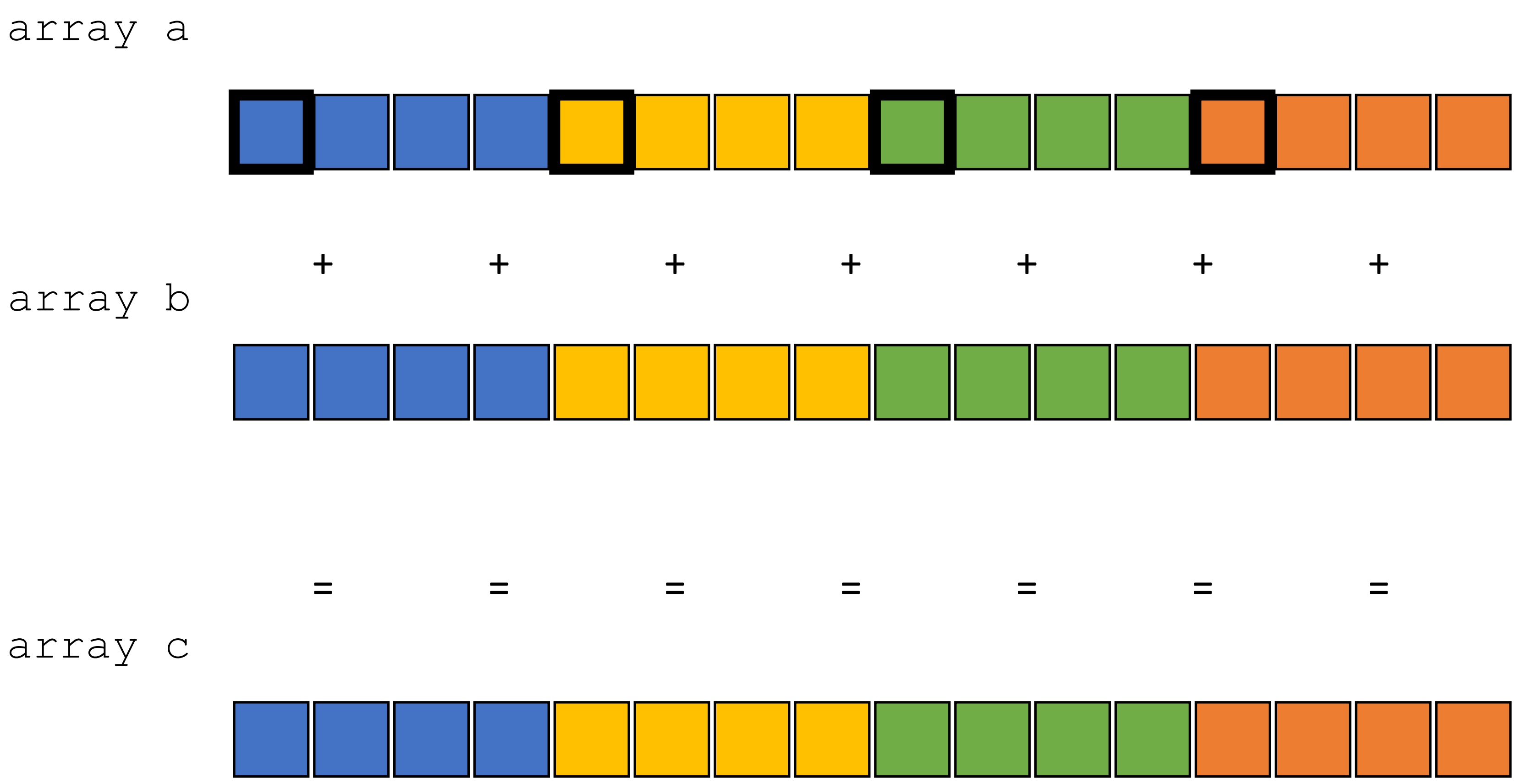
○ False

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

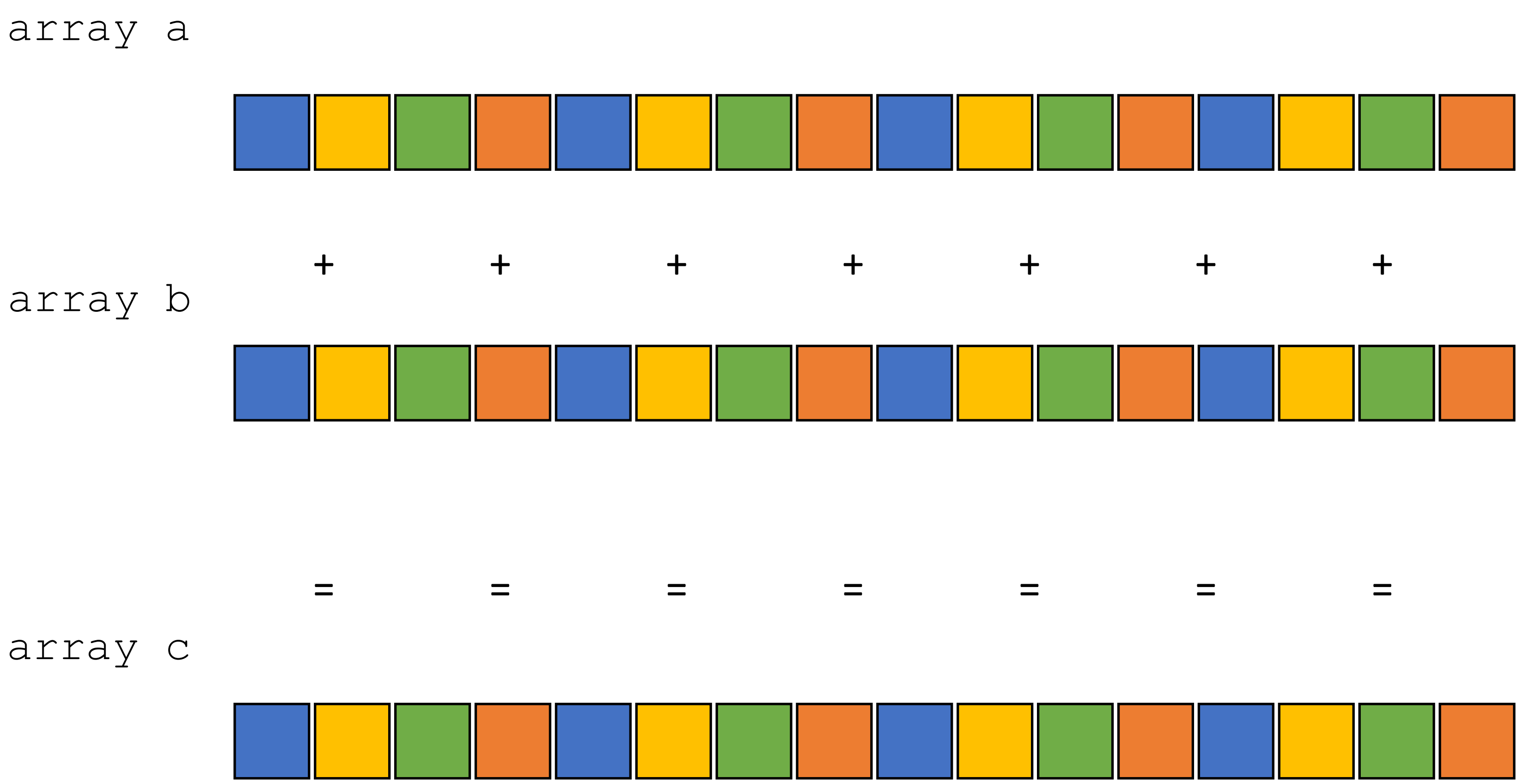
Thread 0 - Blue
Thread 1 - Yellow
Thread 2 - Green
Thread 3 - Orange

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

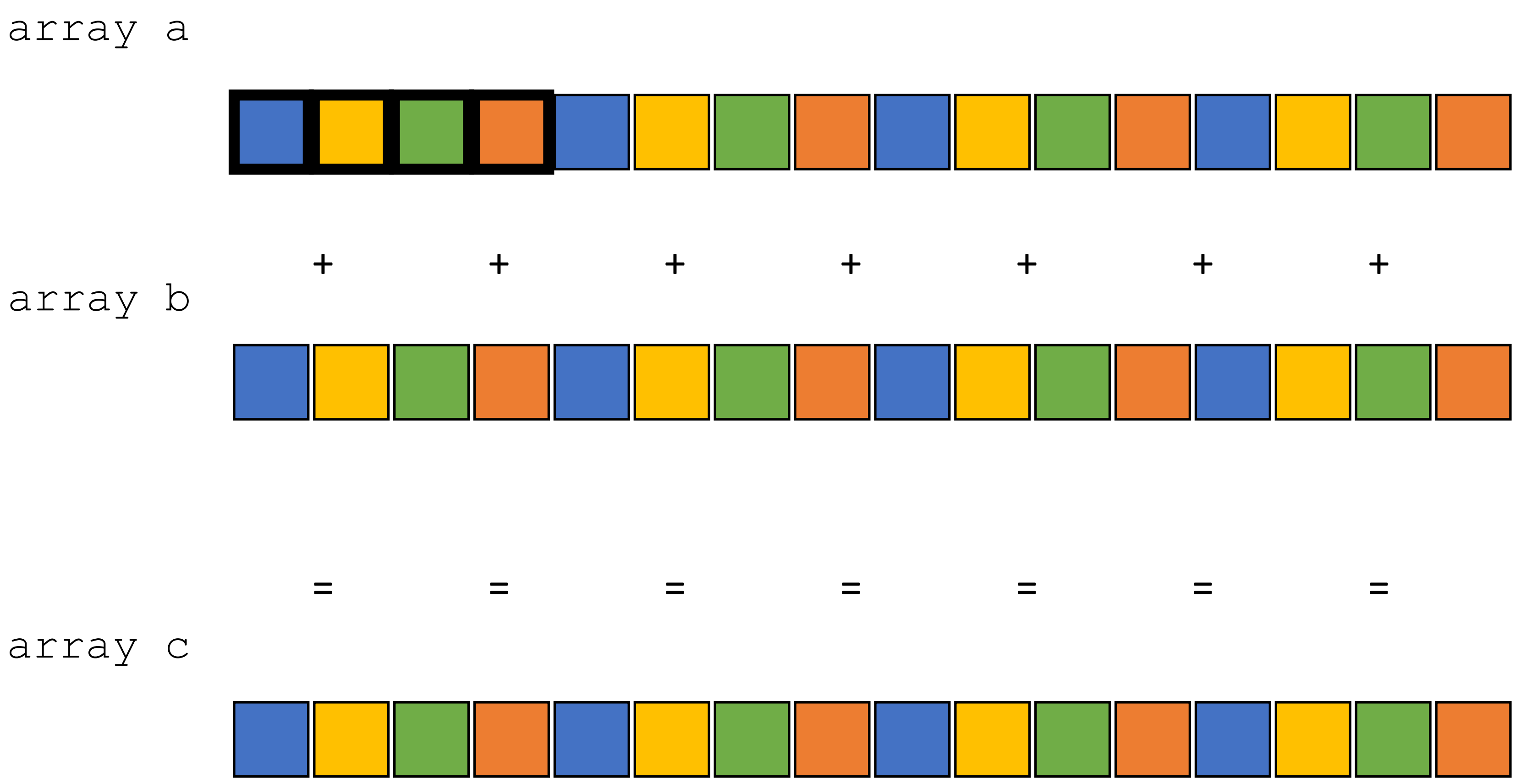


Stride Pattern

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

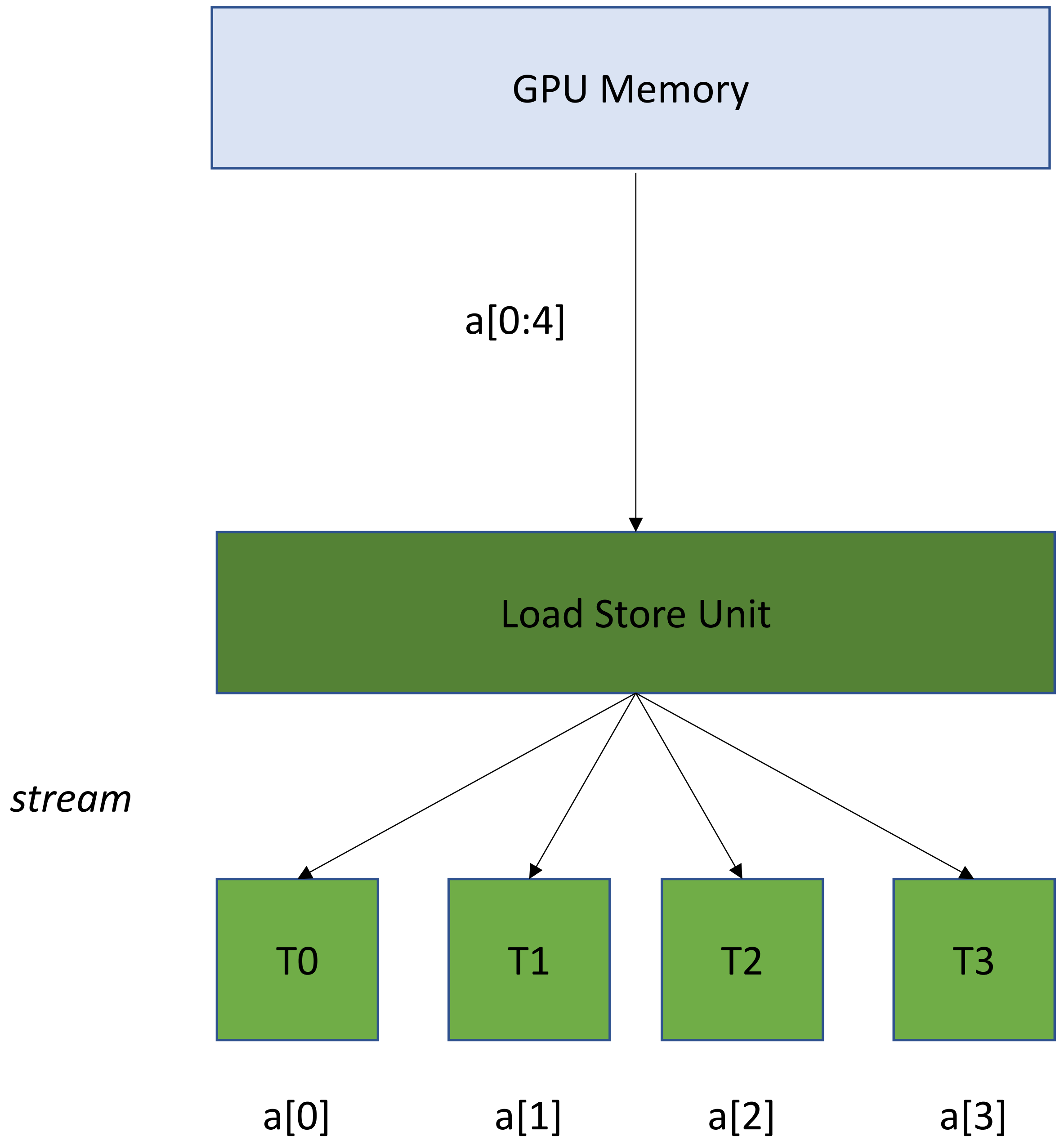


4 cores are accessing memory. What can happen

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

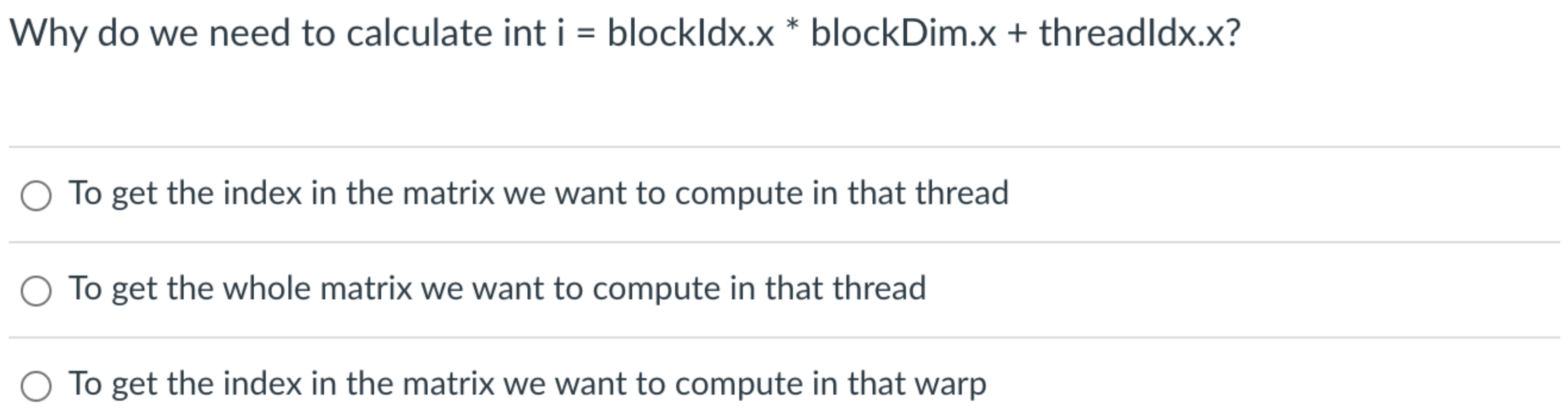


Previous quiz + Review

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

Memory coalescing accelerates execution.

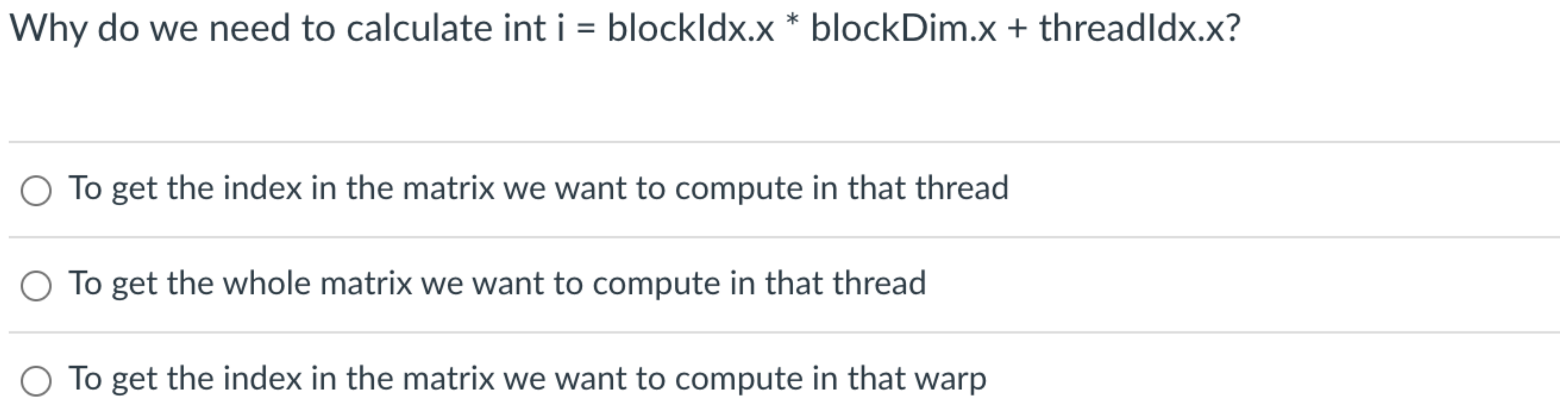
Previous quiz + Review



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);
```

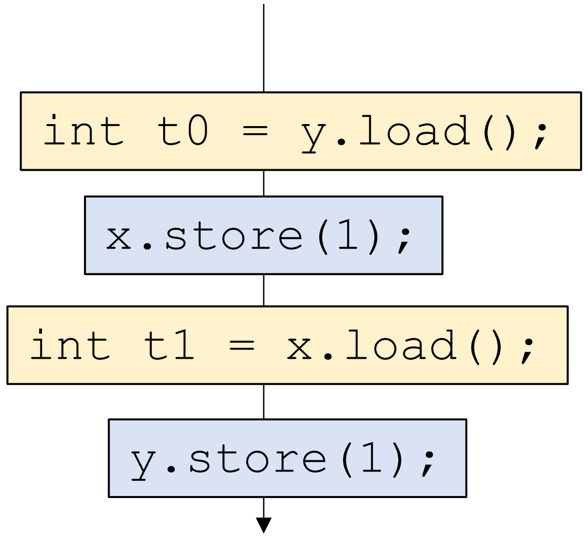
Thread 0:

```
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);
```

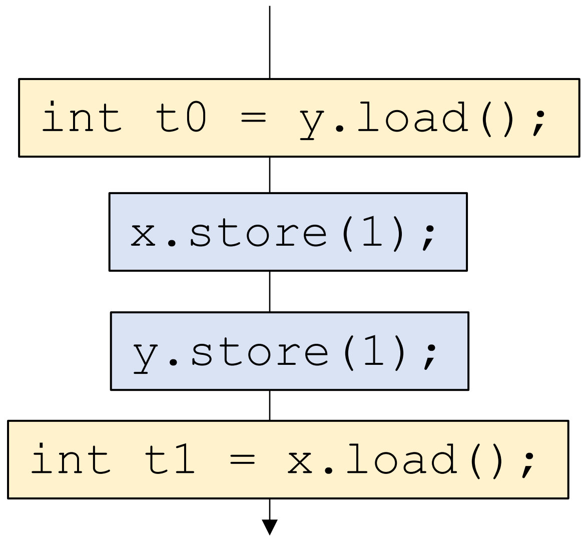
Thread 0:

```
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:

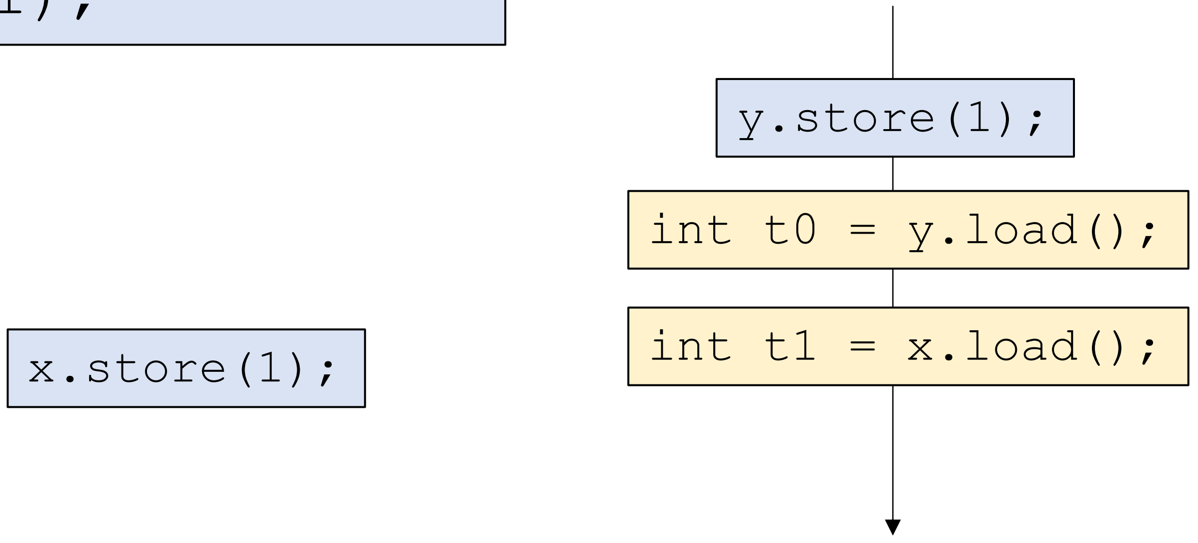
Is it possible for
`t0 == 1 and t1 ==0`

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:

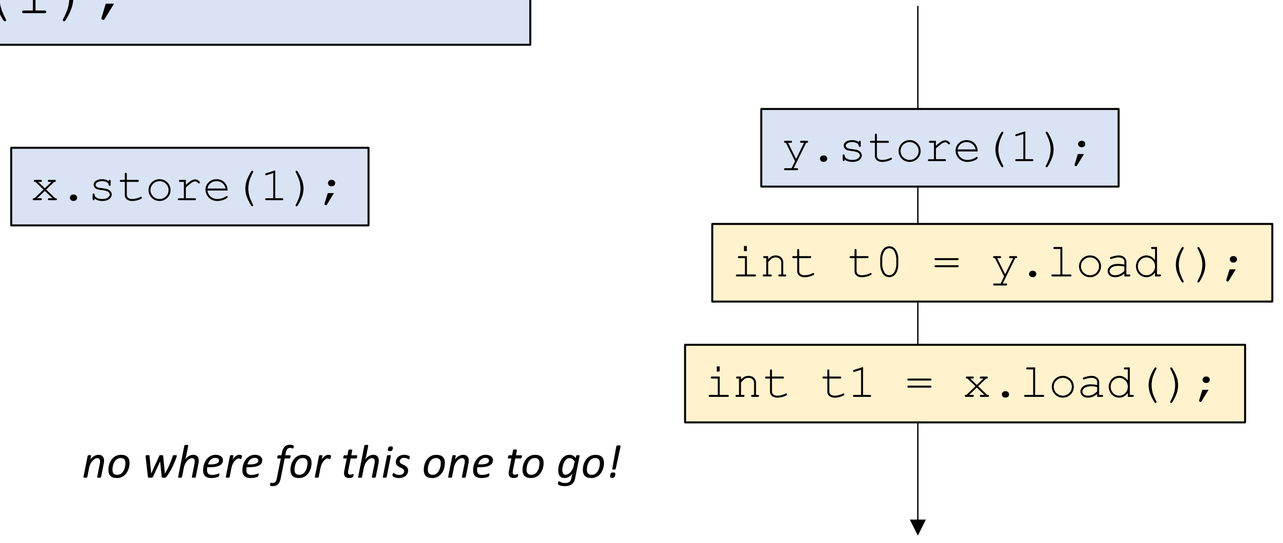
Is it possible for
`t0 == 1 and t1 ==0`

__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();
```

Global variable:

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

Dekker mutual exclusion:
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

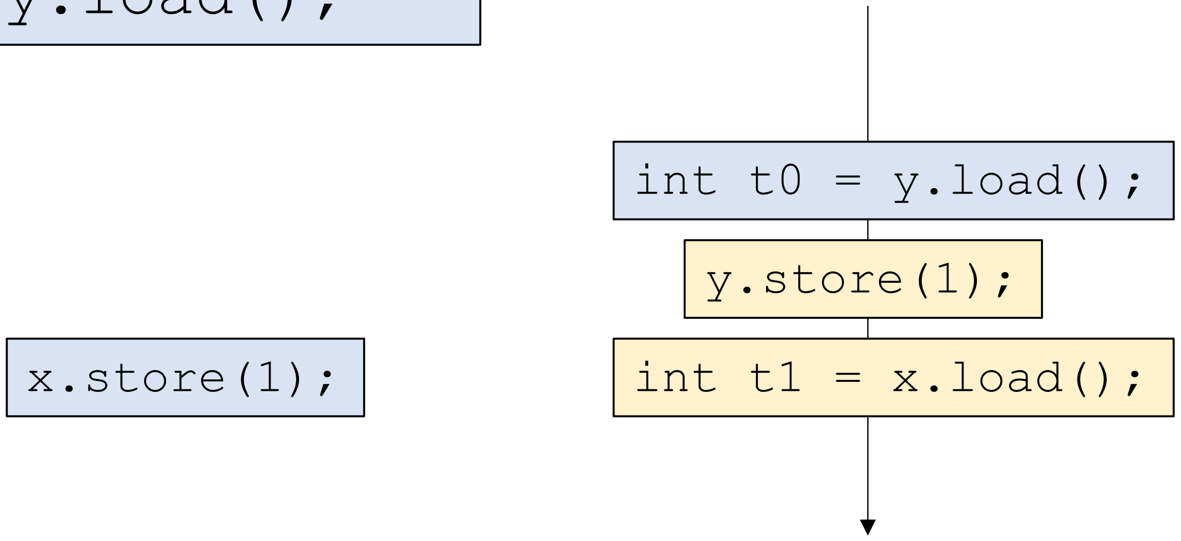
Can `t0 == t1 == 0`?

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

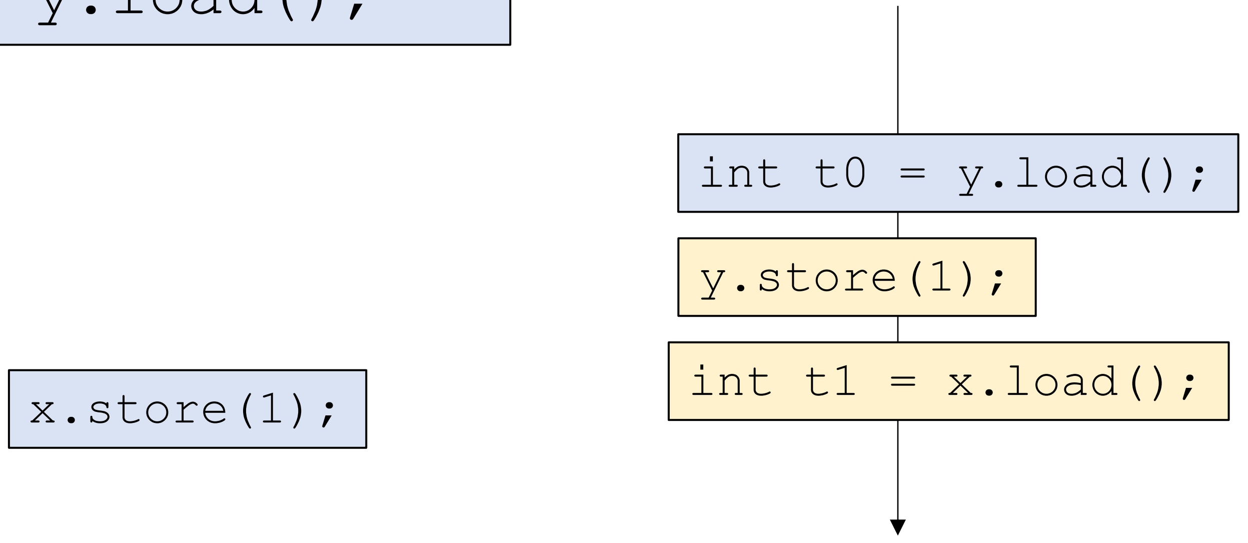
Can `t0 == t1 == 0`?

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

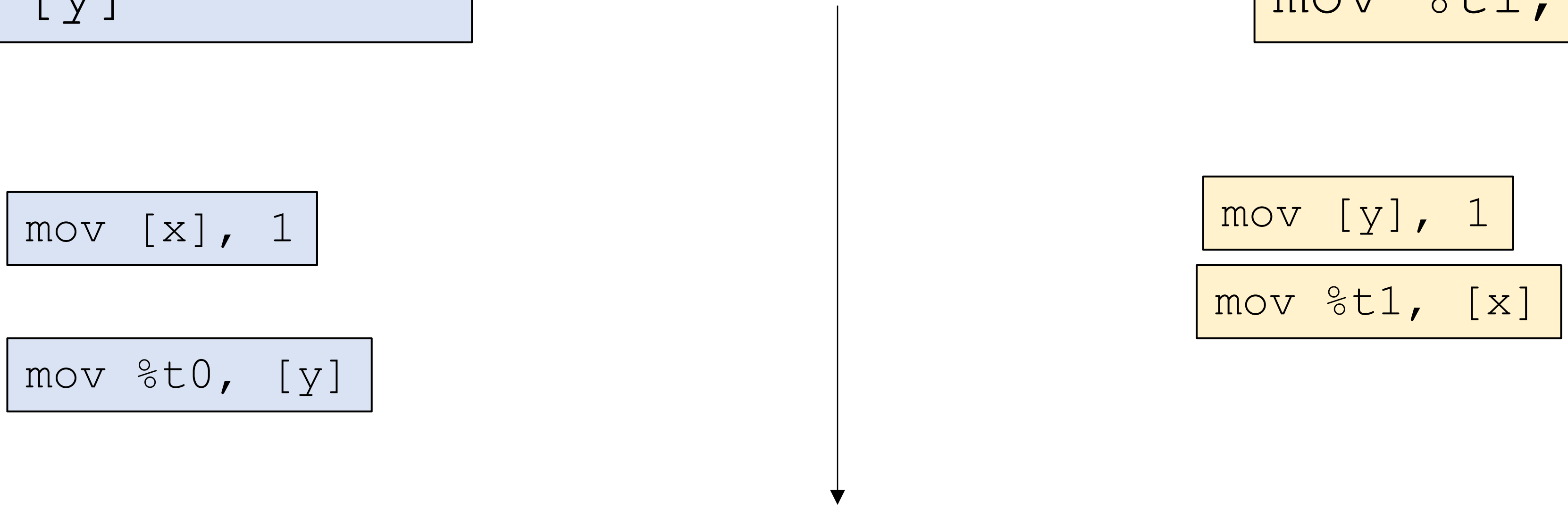
Can `t0 == t1 == 0`?

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

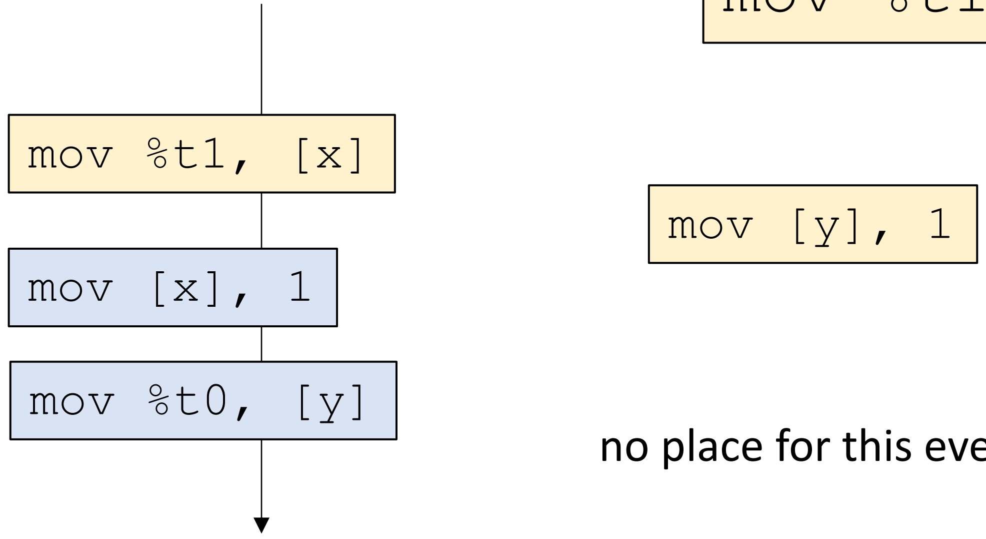
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



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:

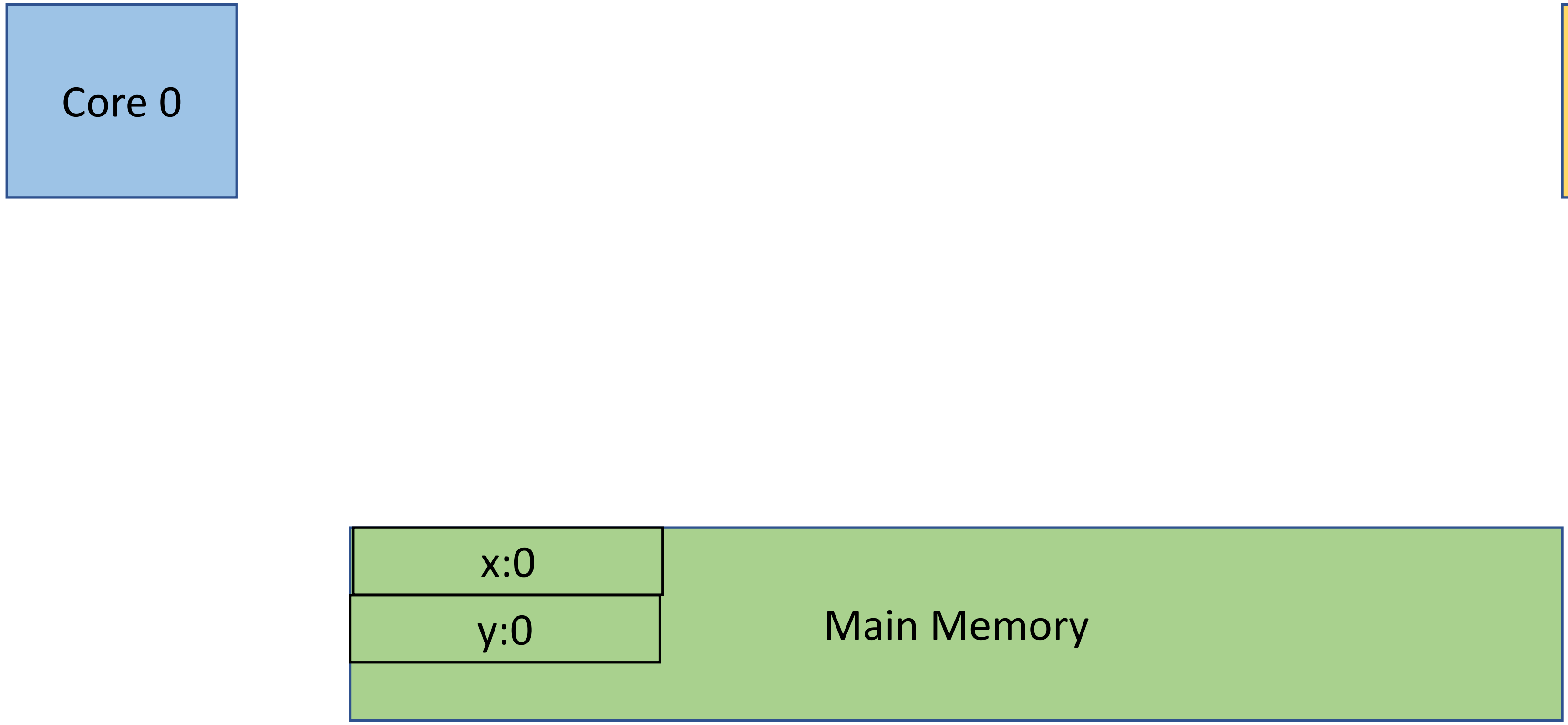
```
mov [x], 1
```

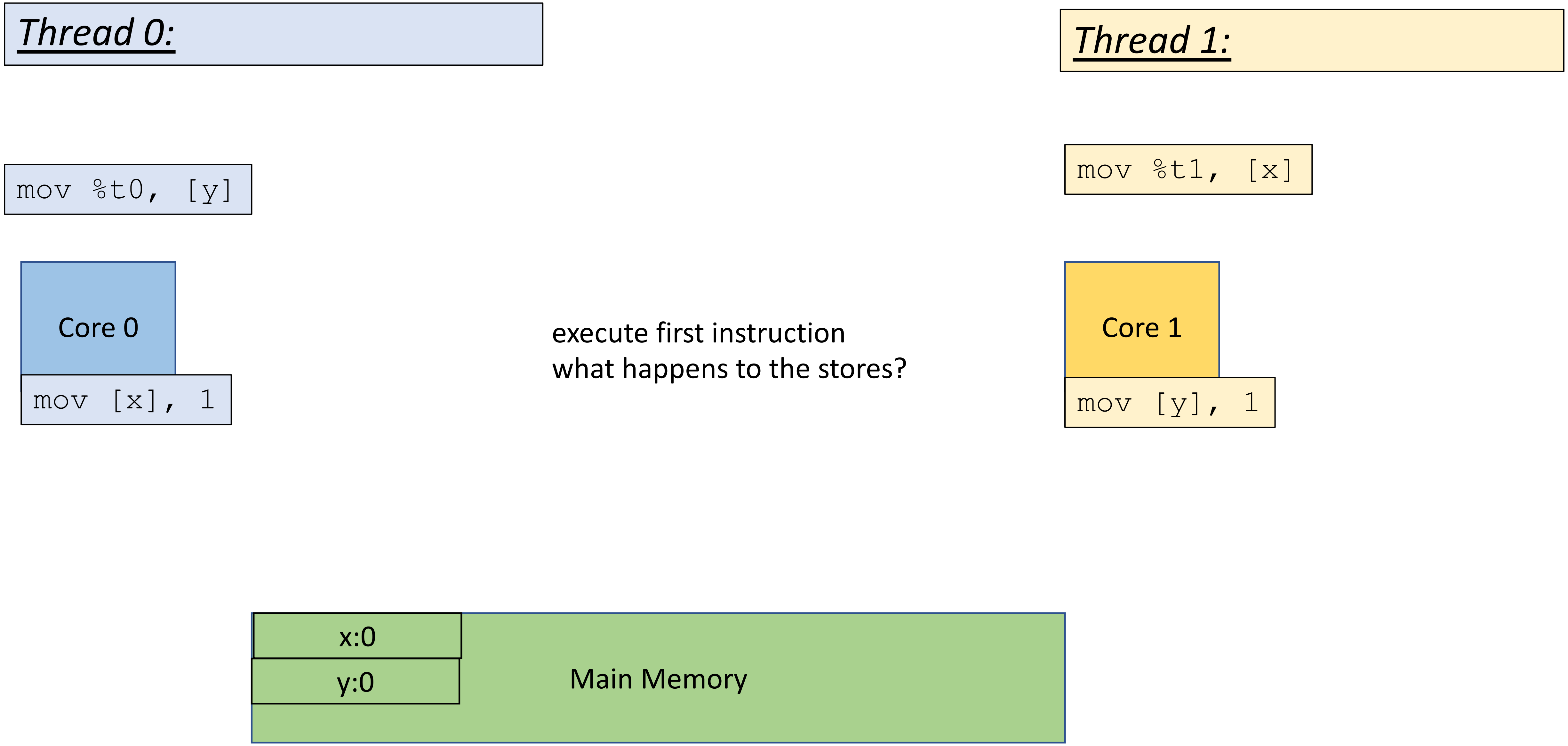
```
mov %t0, [y]
```

Thread 1:

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

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





Thread 0:

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

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

Thread 1:

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

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

execute first instruction
what happens to the stores?

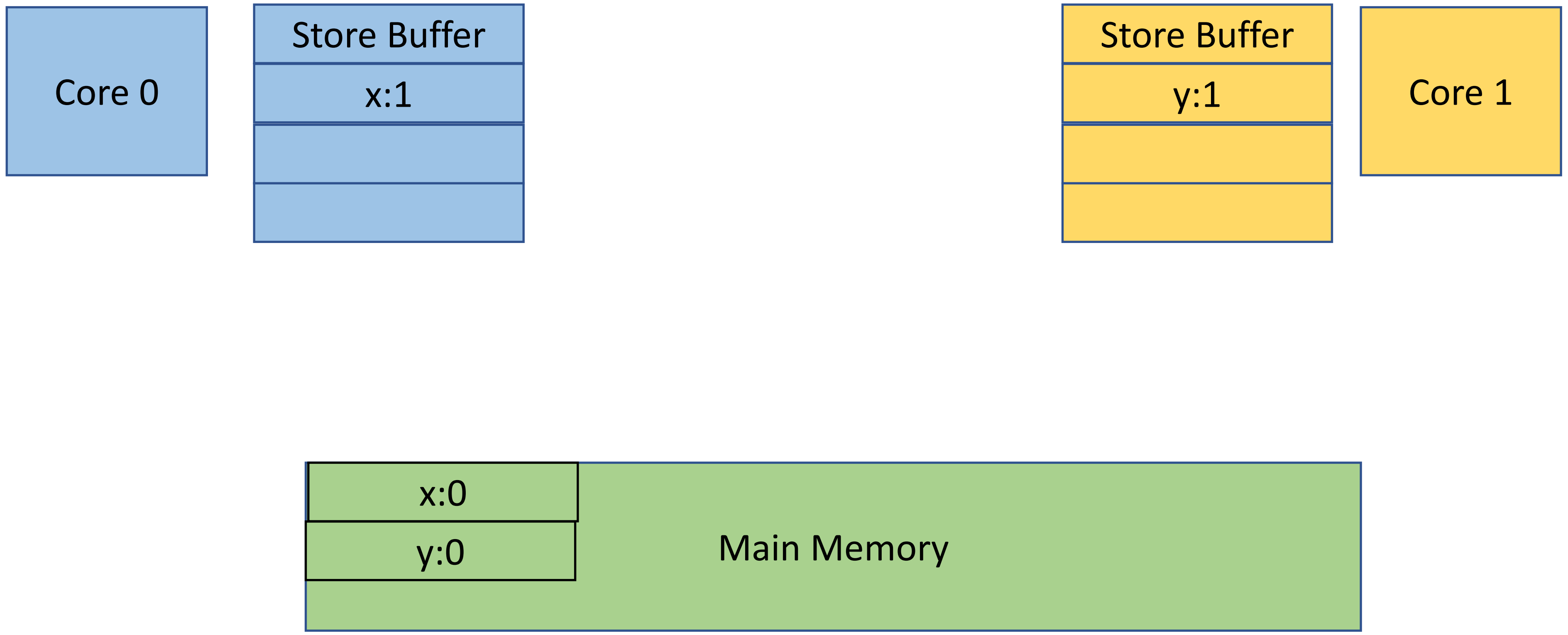
Thread 0:

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

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

Thread 1:

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



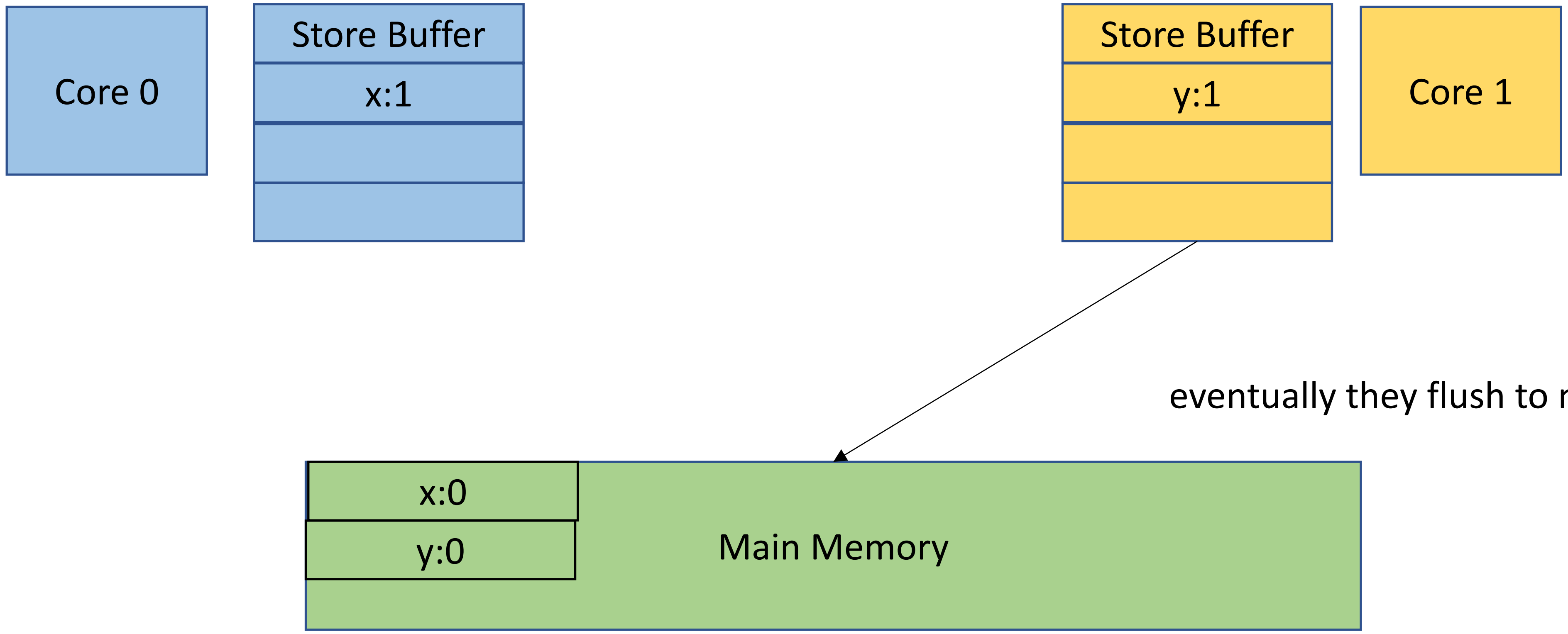
Thread 0:

Thread 1:

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

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

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



eventually they flush to main memory

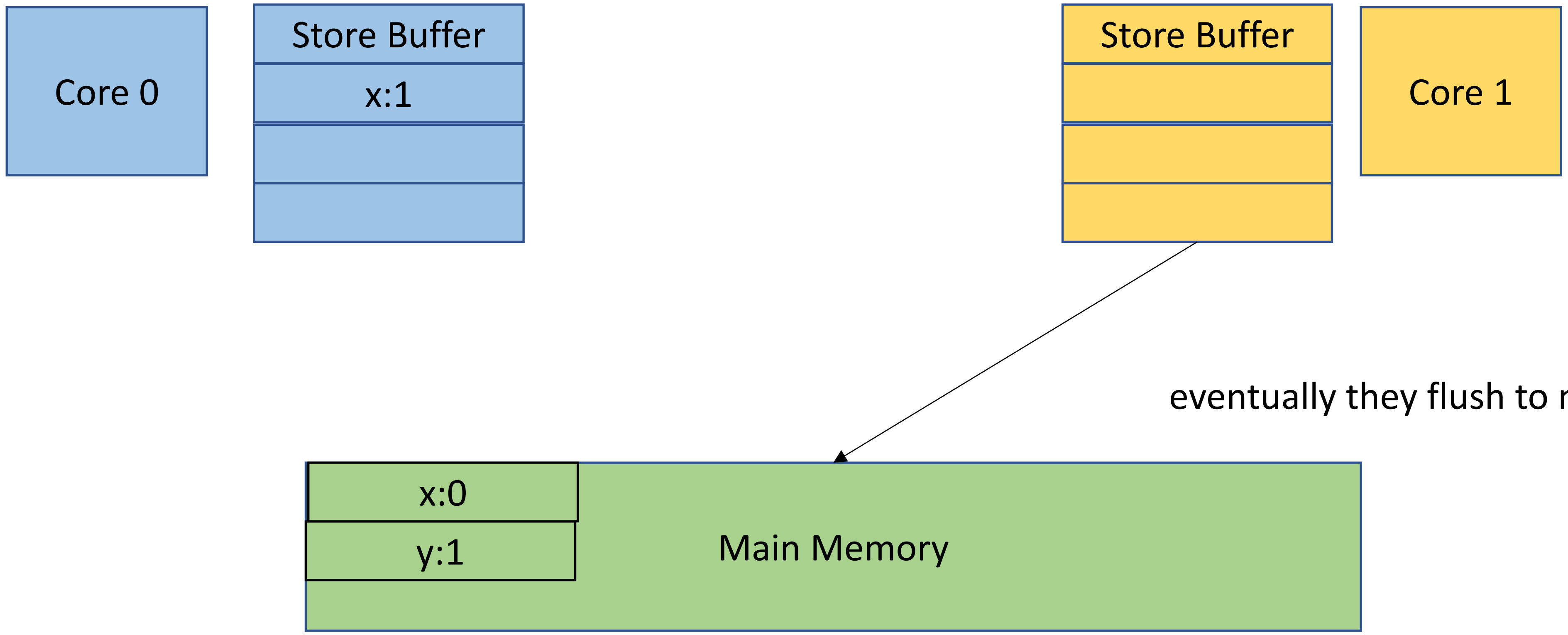
Thread 0:

Thread 1:

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

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

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



eventually they flush to main memory

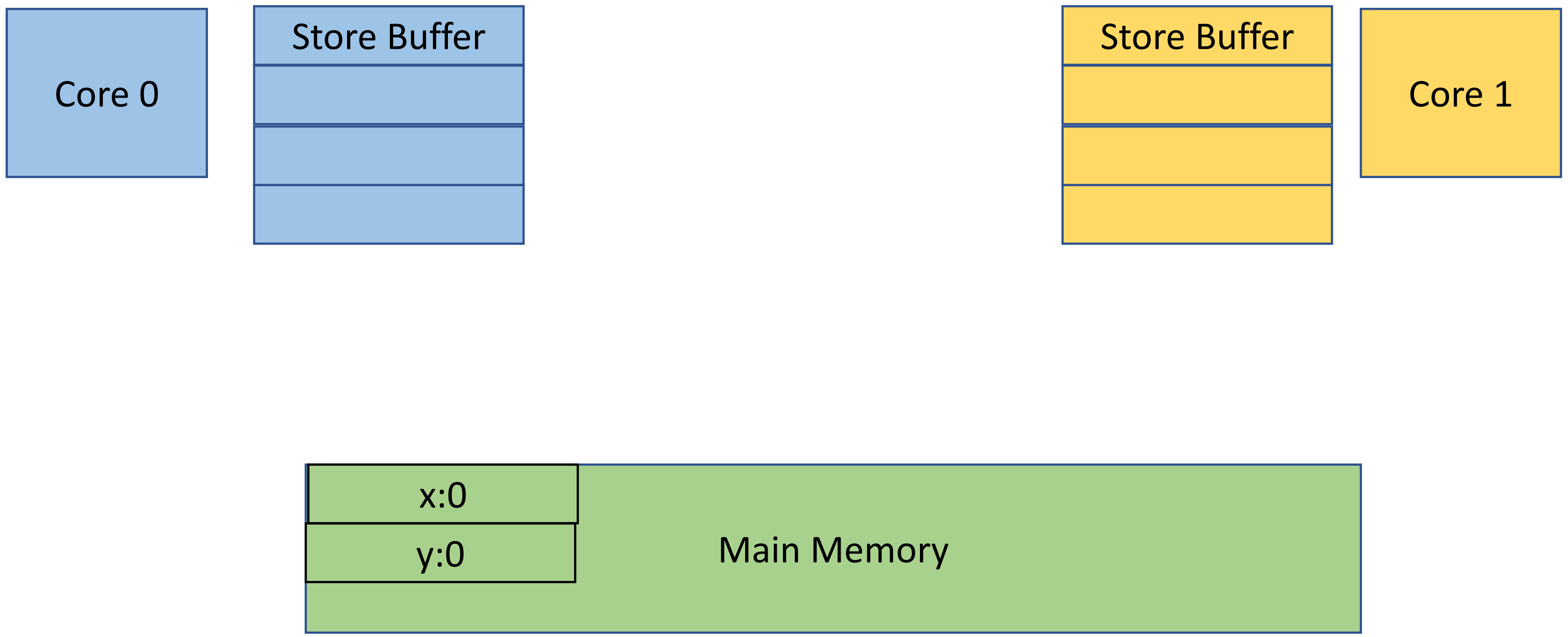
Thread 0:

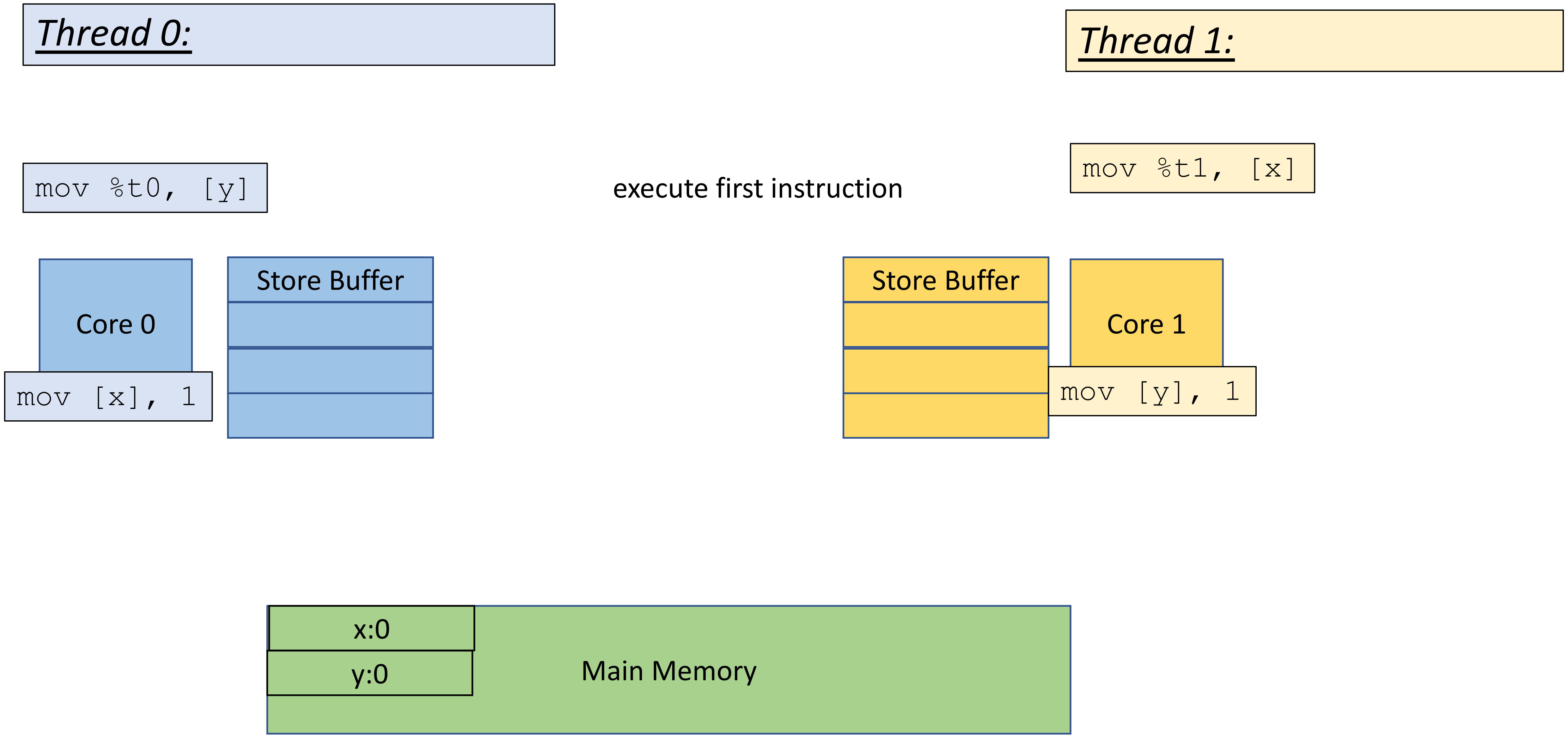
```
mov [x], 1
mov %t0, [y]
```

Thread 1:

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

rewind





```
Thread 0:
mov %t0, [y]
mov [x], 1
```

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

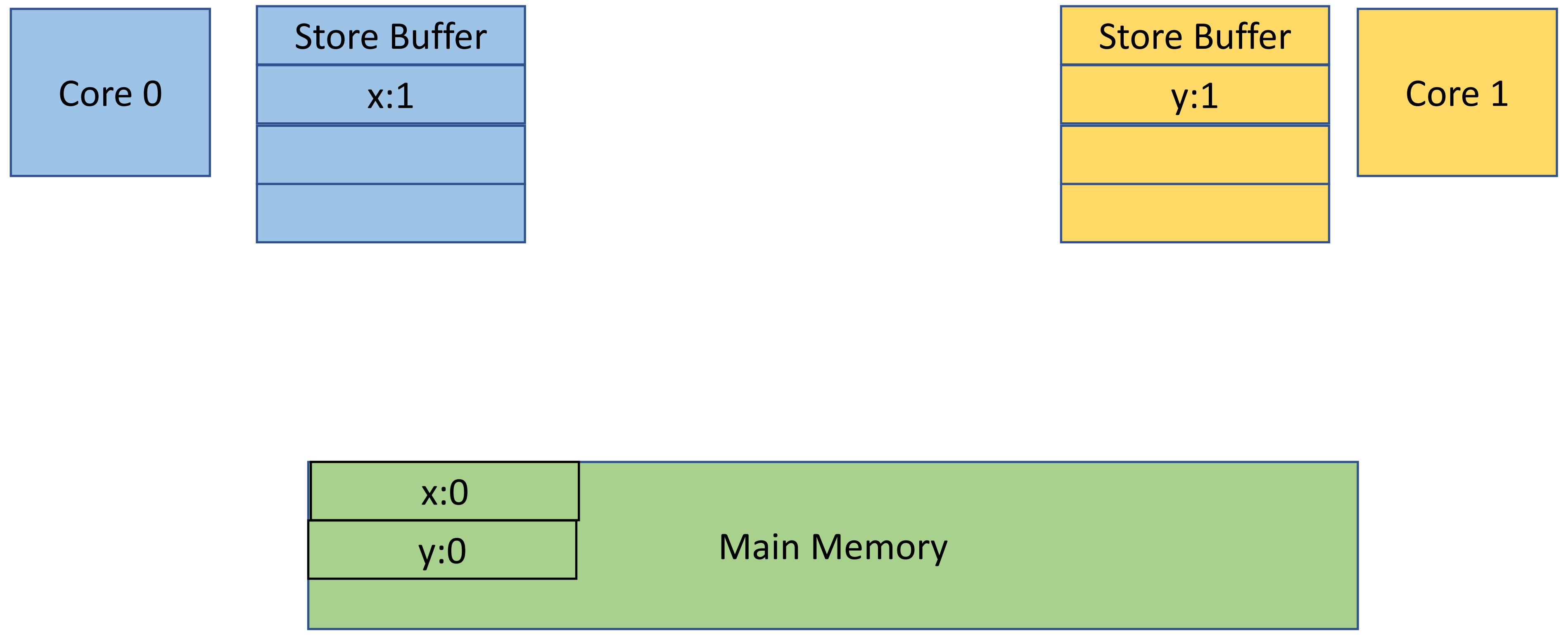
Thread 0:

Thread 1:

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

values get stored in SB

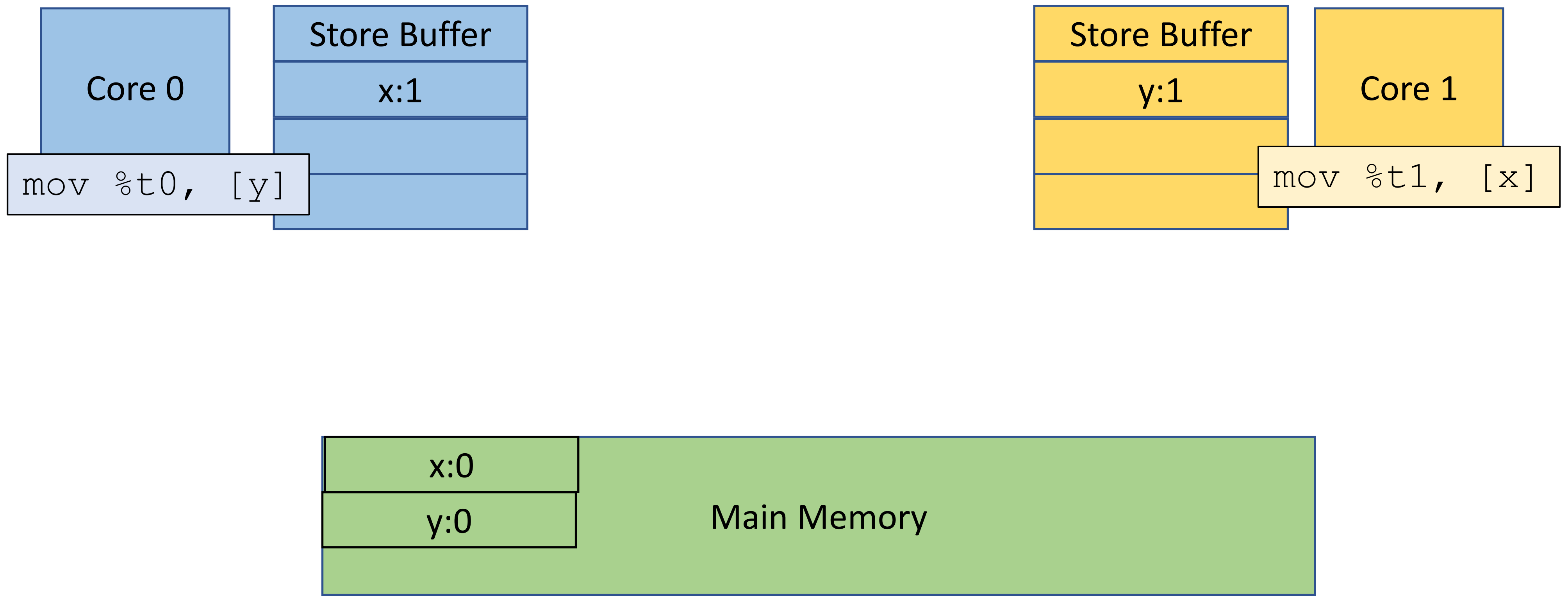
```
mov %t1, [x]
```



Thread 0:

Thread 1:

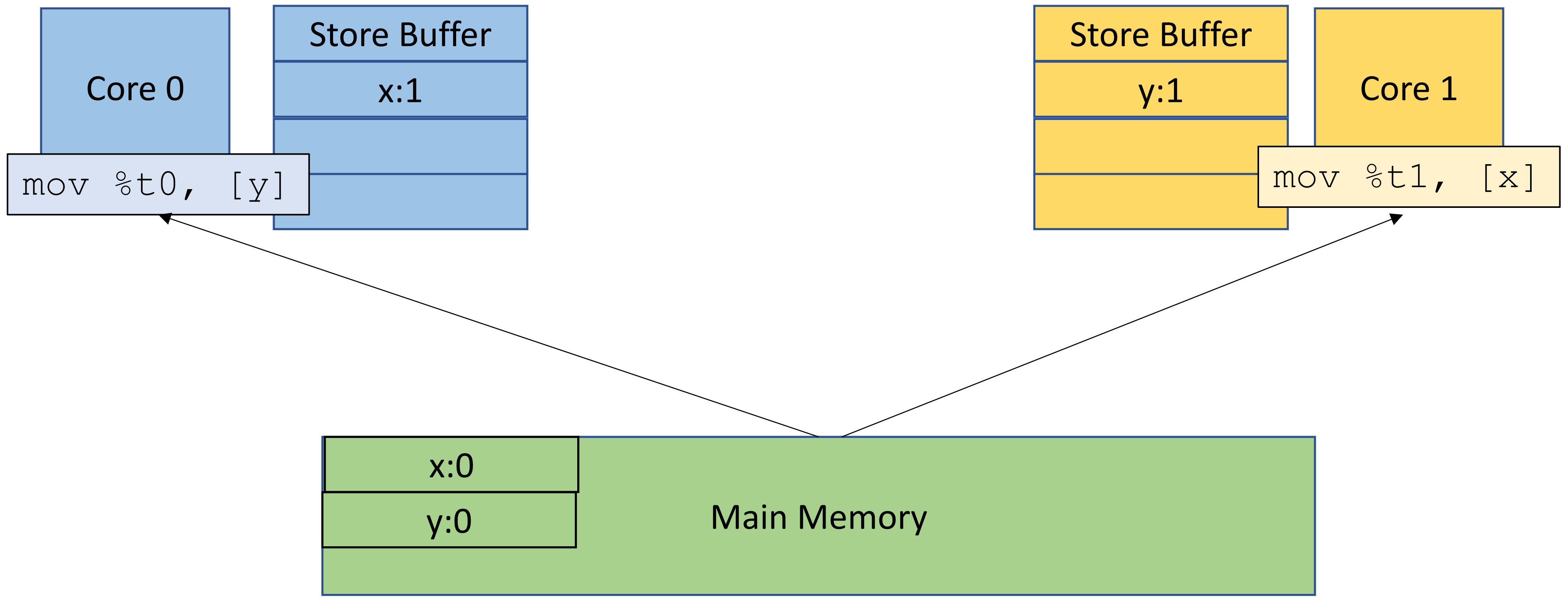
Execute next instruction



Thread 0:

Thread 1:

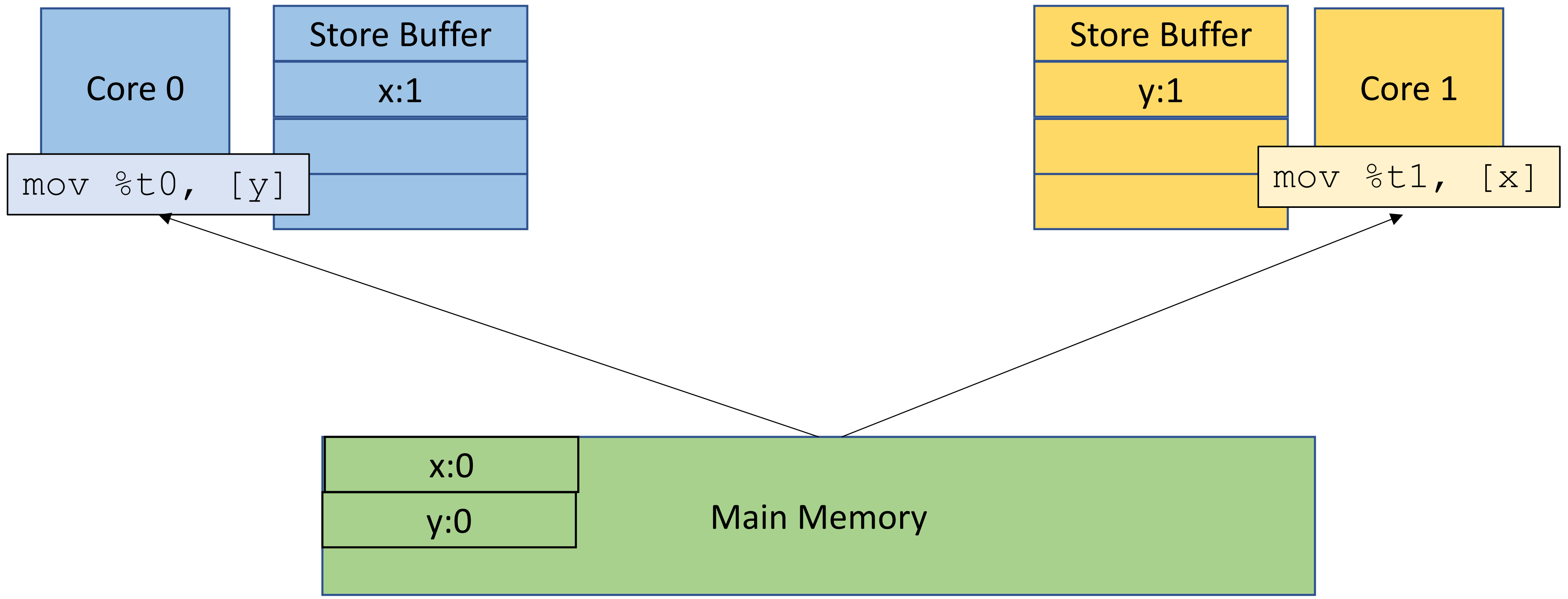
Values get loaded from memory



Thread 0:

Thread 1:

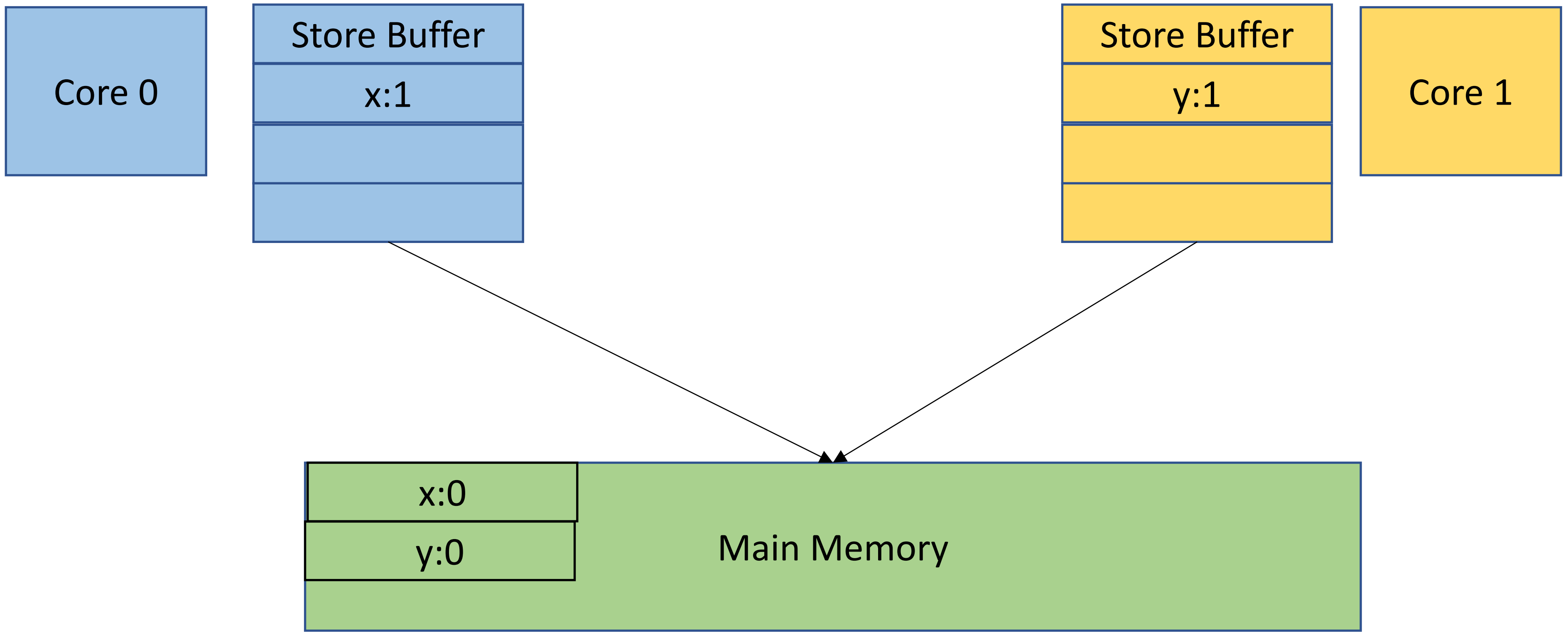
we see `t0 == t1 == 0`!



Thread 0:

Thread 1:

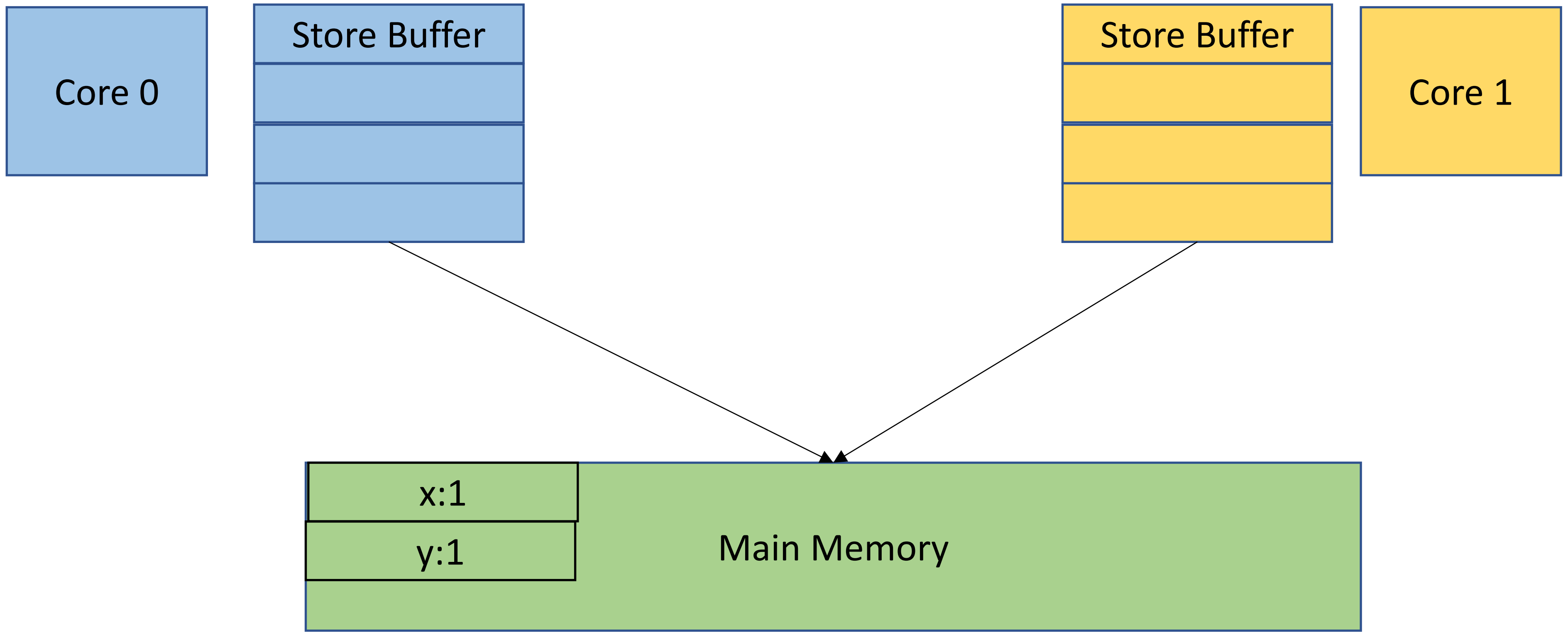
Store buffers are drained eventually



Thread 0:

Thread 1:

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



Our first relaxed memory execution!

- **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

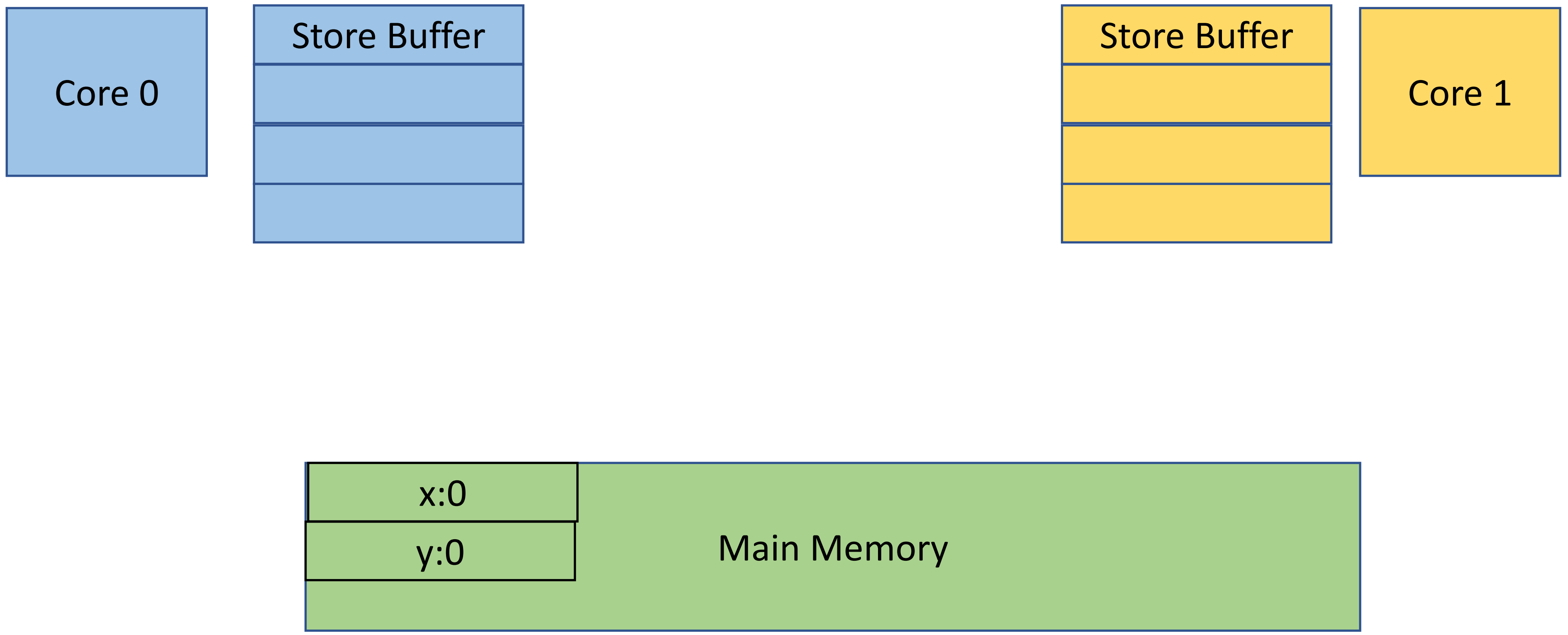
- The X86 fence that flushes the store buffer is called `mfence`.

Thread 0:

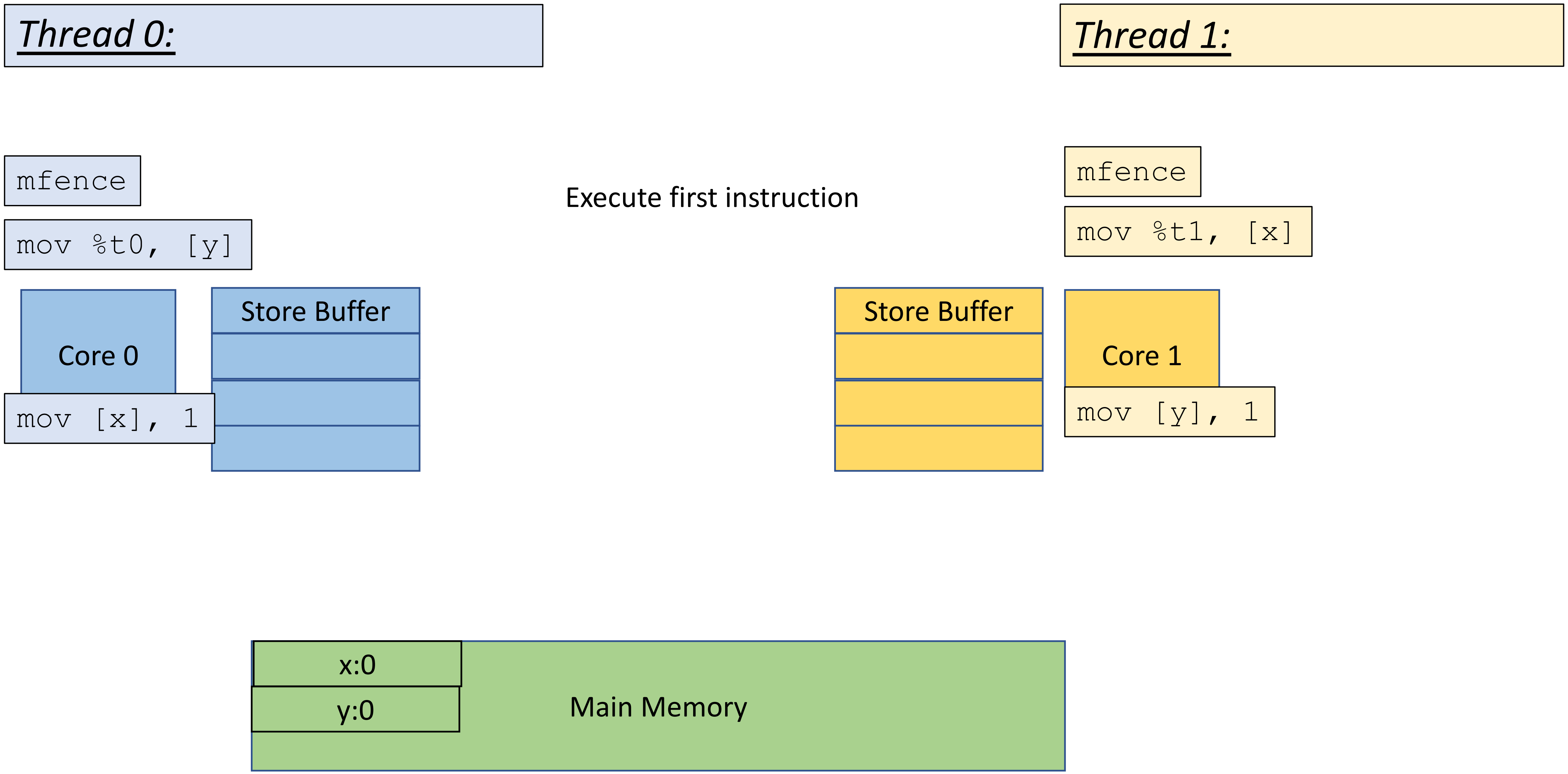
```
mov [x], 1
mfence
mov %t0, [y]
```

Thread 1:

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



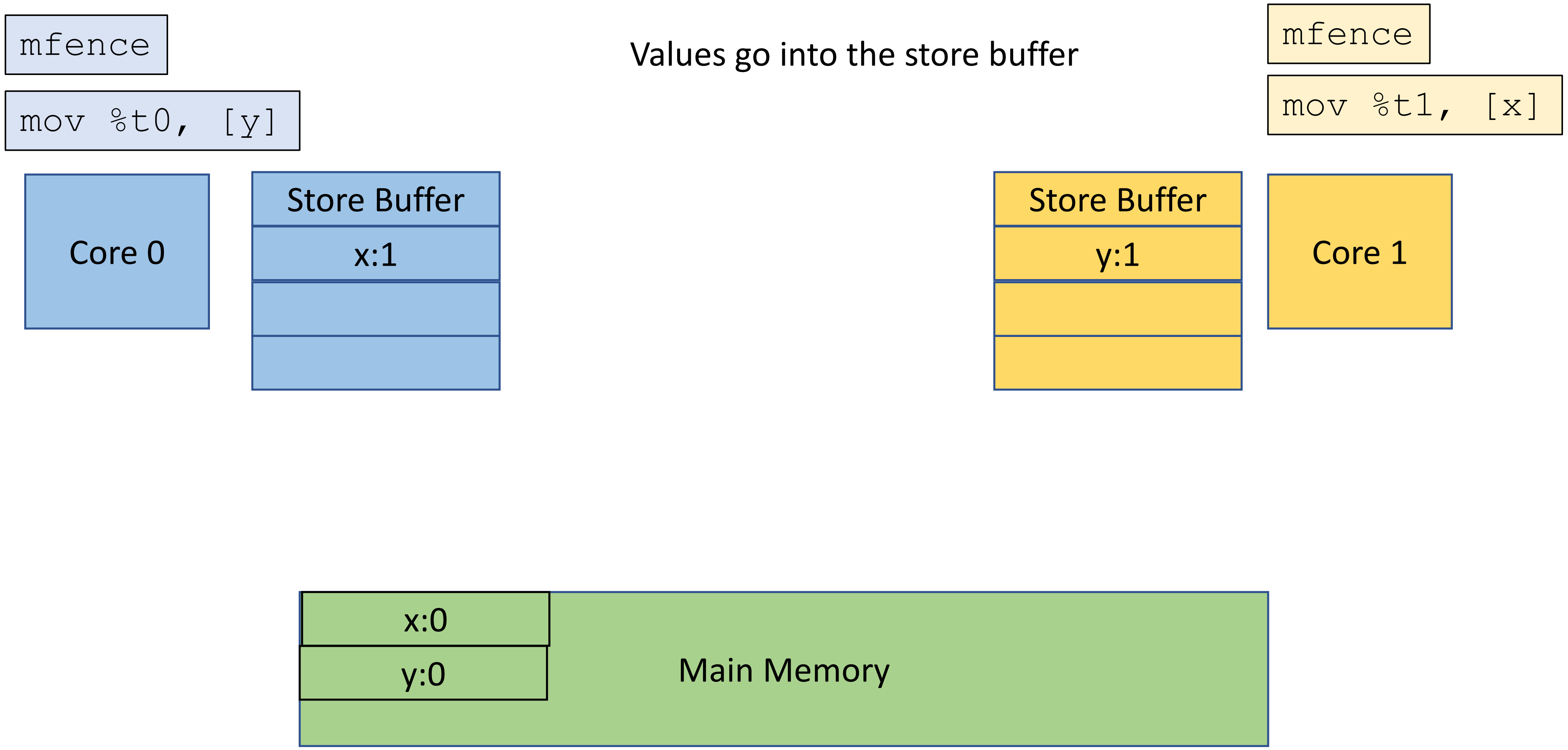
Execute first instruction



Thread 0:

Thread 1:

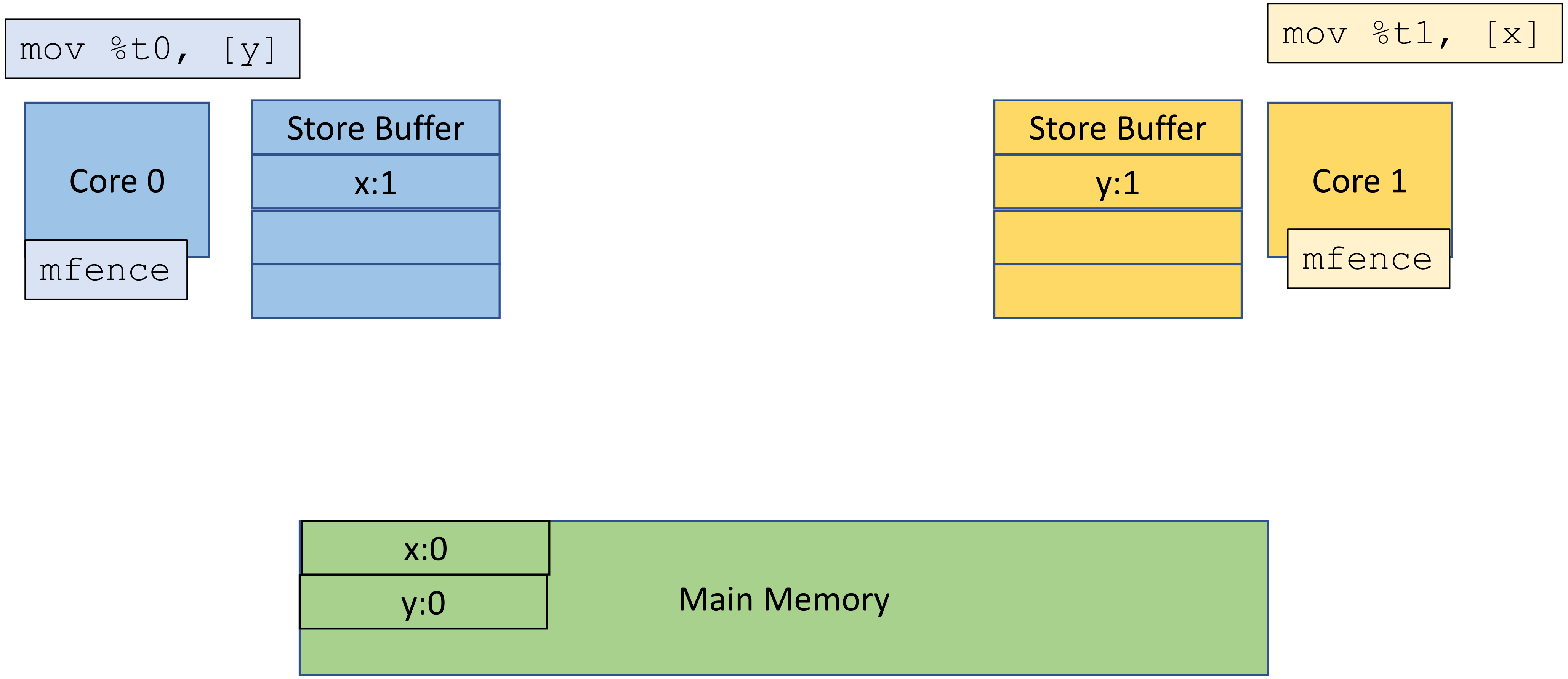
Values go into the store buffer



Thread 0:

Thread 1:

Execute next instruction



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

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

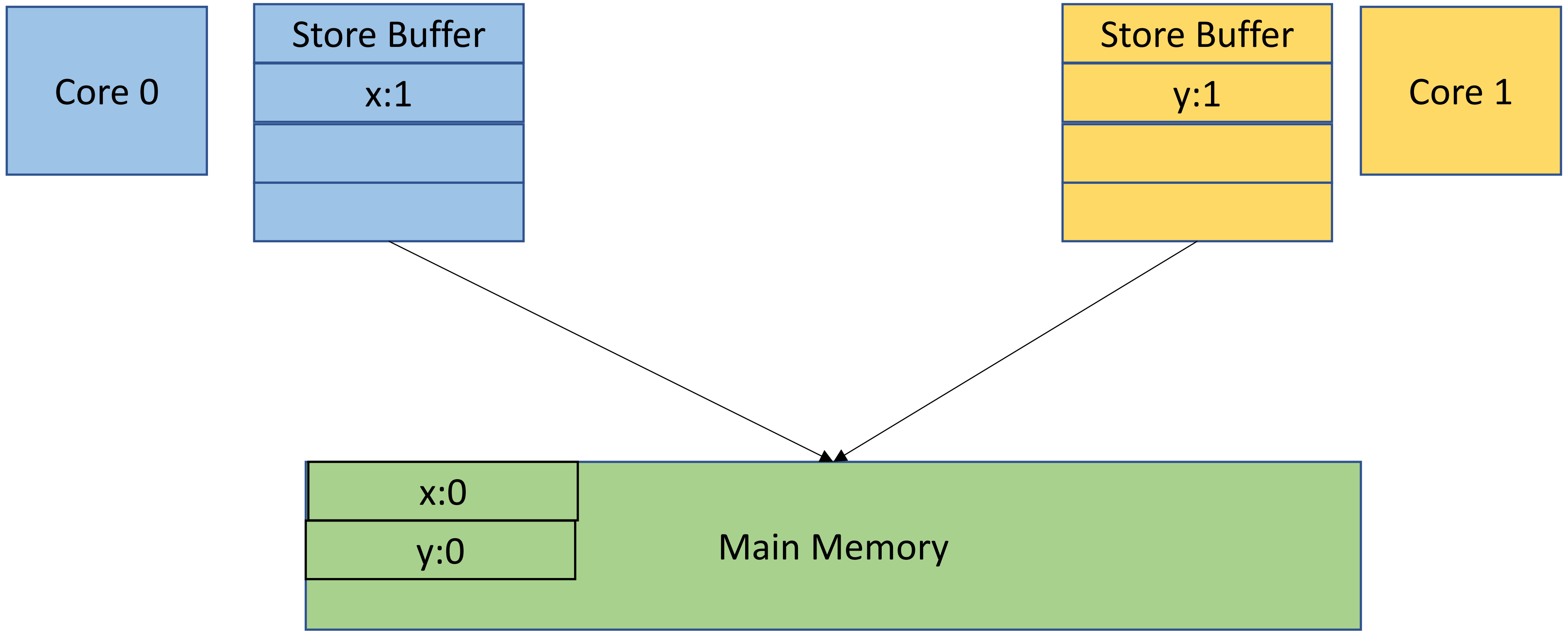
Thread 0:

Thread 1:

store buffers are flushed

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

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



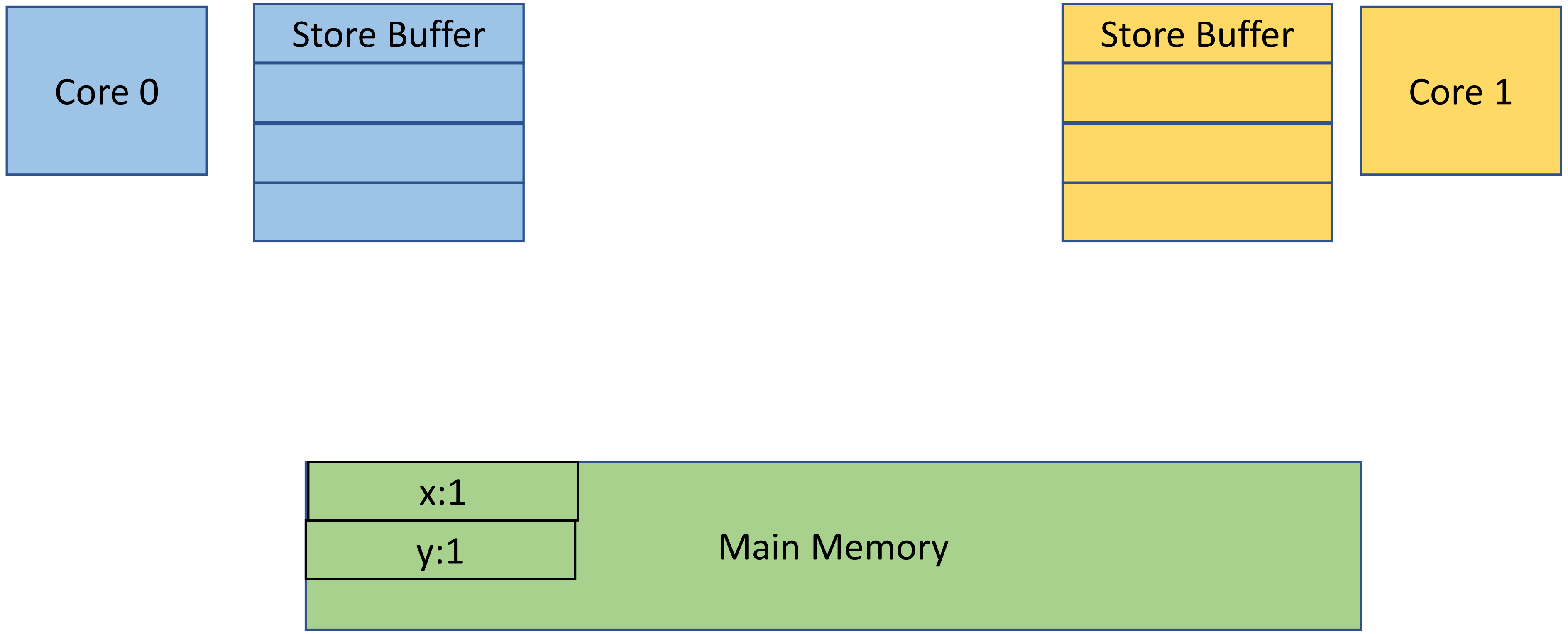
Thread 0:

Thread 1:

store buffers are flushed

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

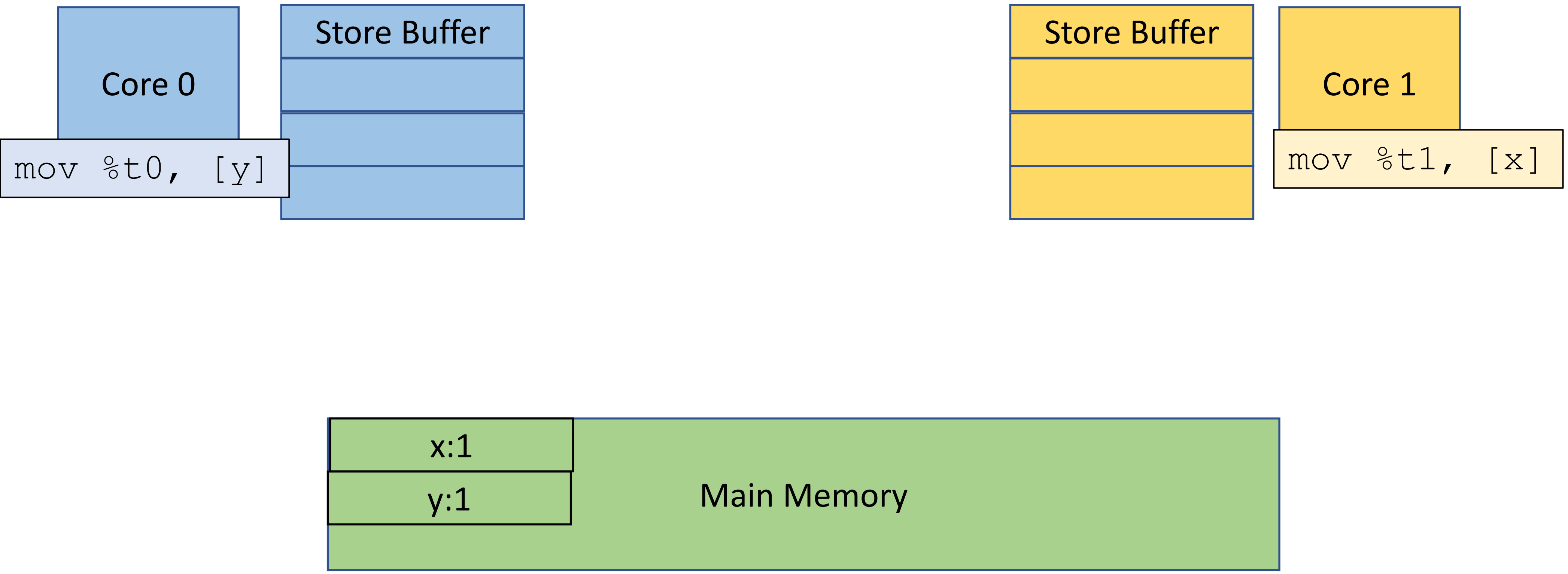
```
mov %t1, [x]
```



Thread 0:

Thread 1:

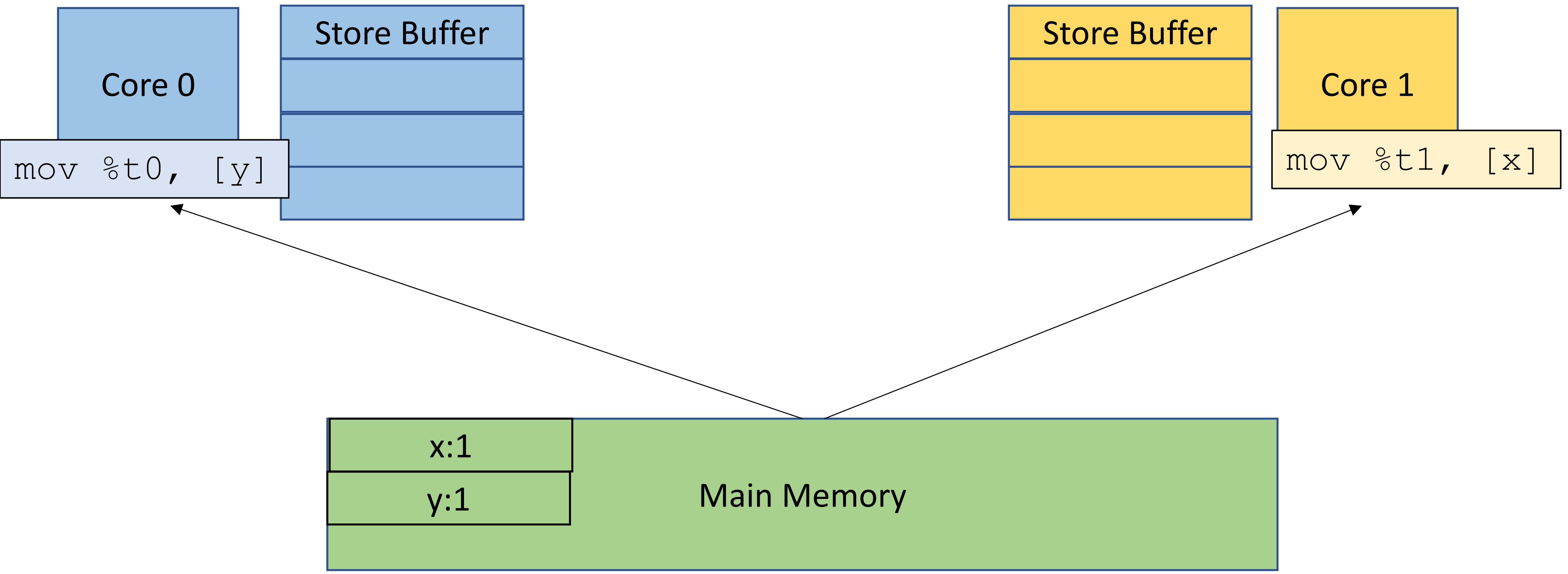
execute next instruction



Thread 0:

Thread 1:

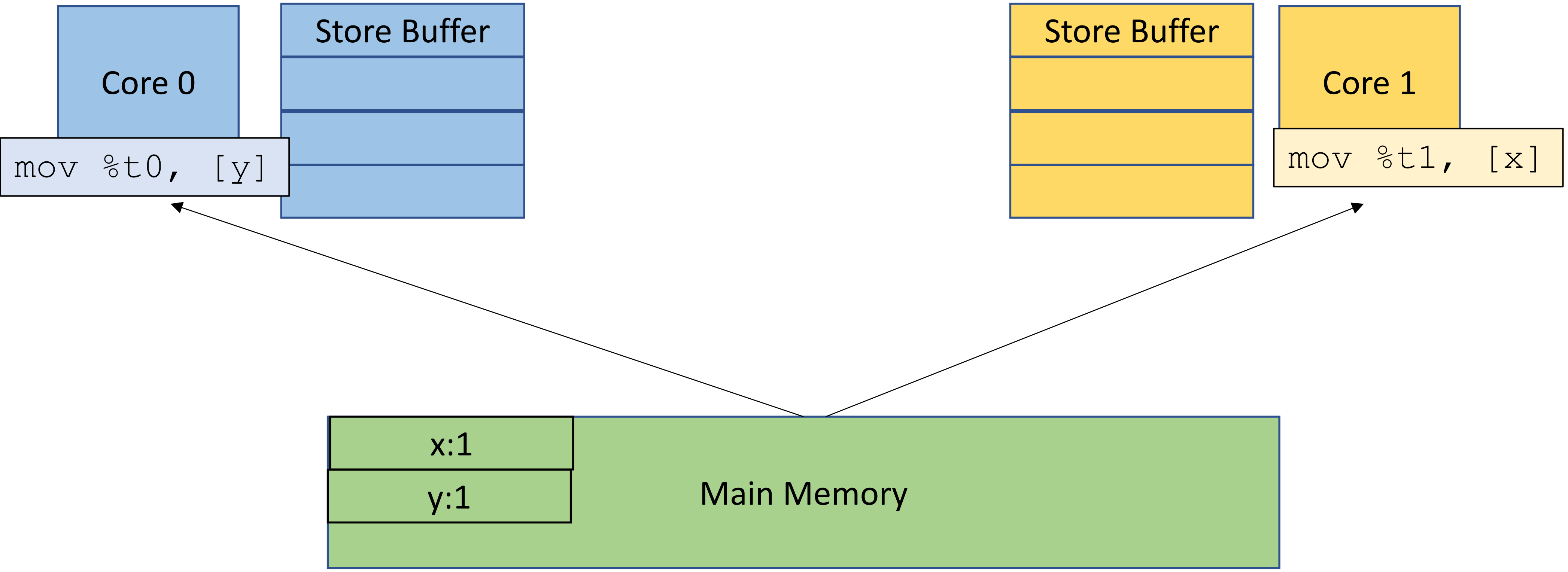
values are loaded from memory



Thread 0:

Thread 1:

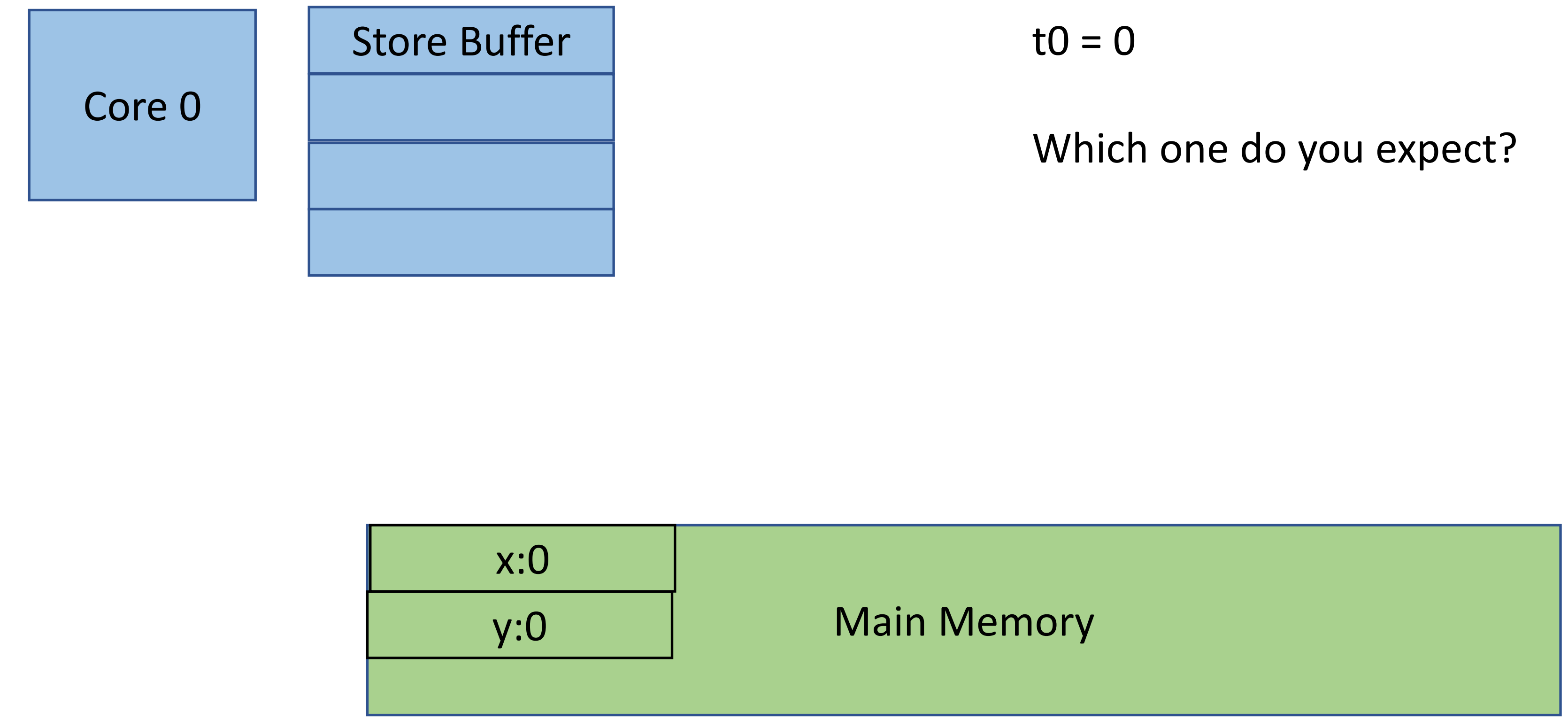
We don't get the problematic behavior: `t0 == t1 == 0`



Thread 0:

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

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



single thread
same address

possible outcomes:
t0 = 1
t0 = 0

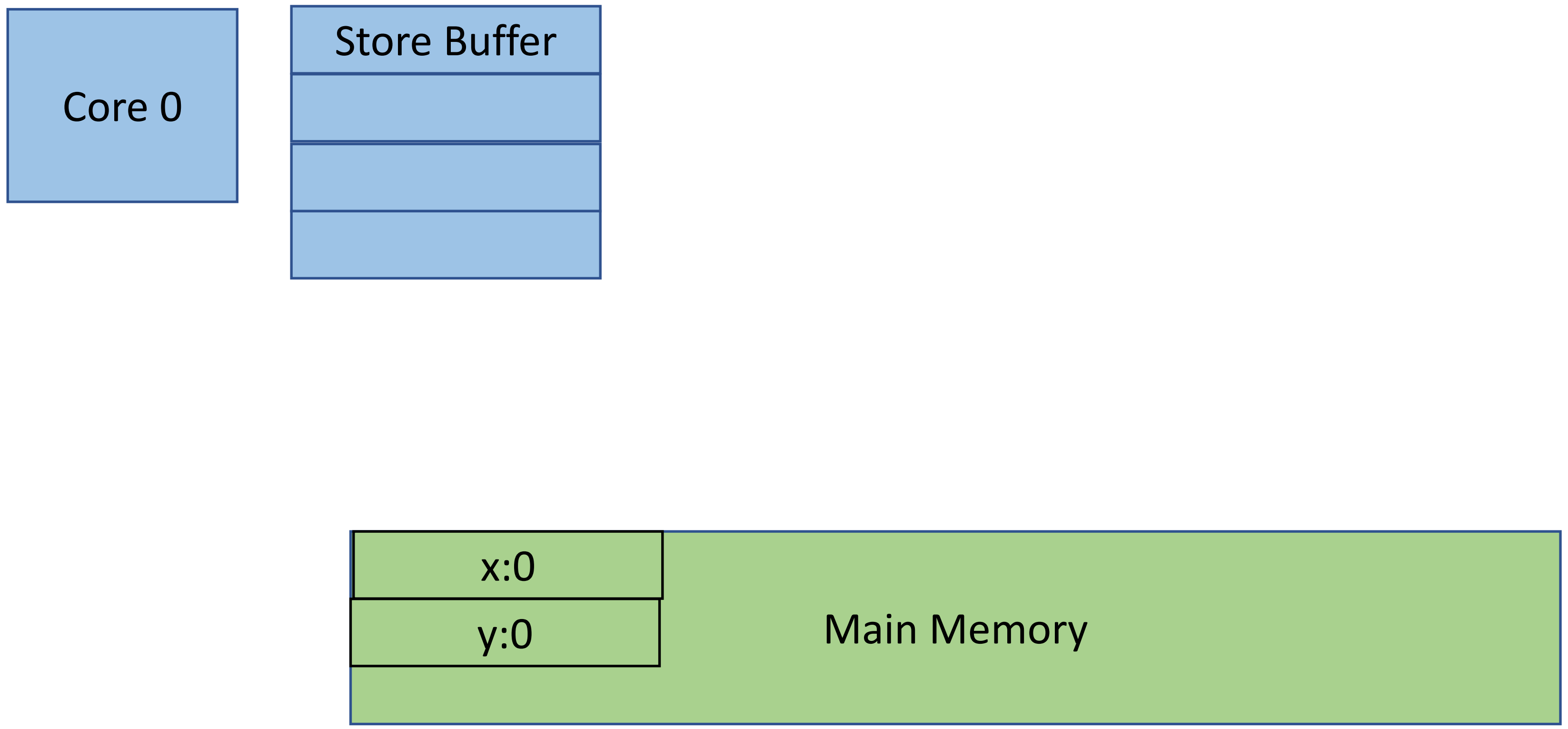
Which one do you expect?

Thread 0:

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

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

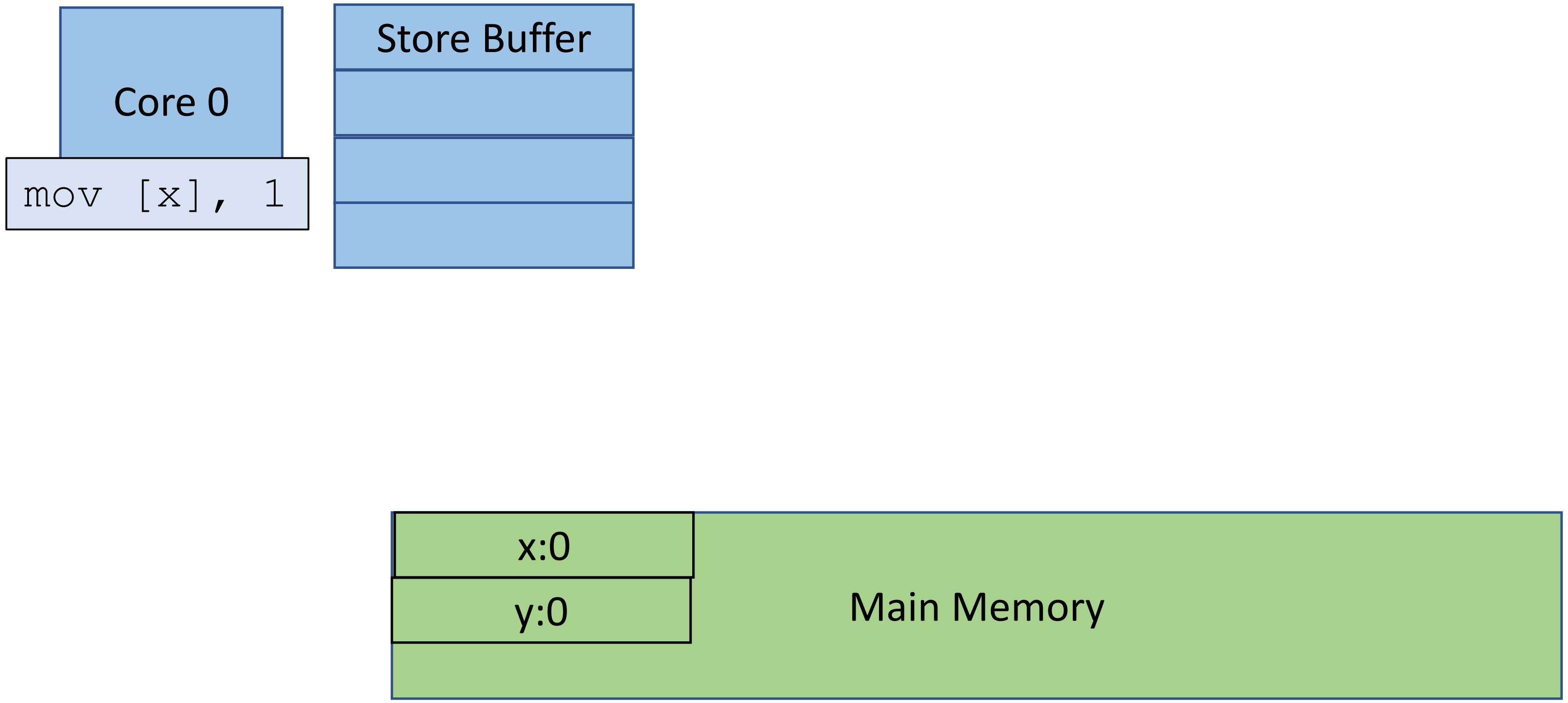
How does this execute?



Thread 0:

execute first instruction

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

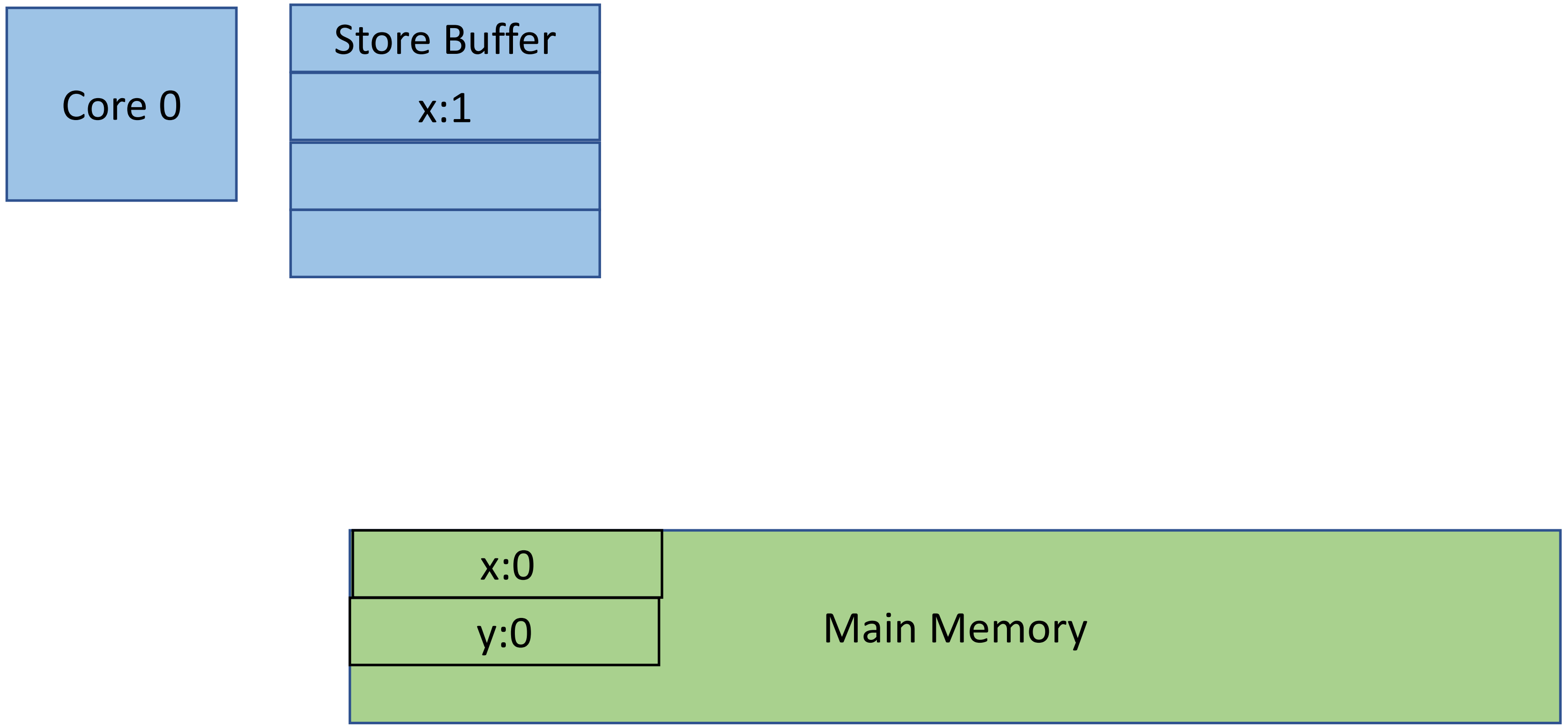


```
mov [x], 1
```

Thread 0:

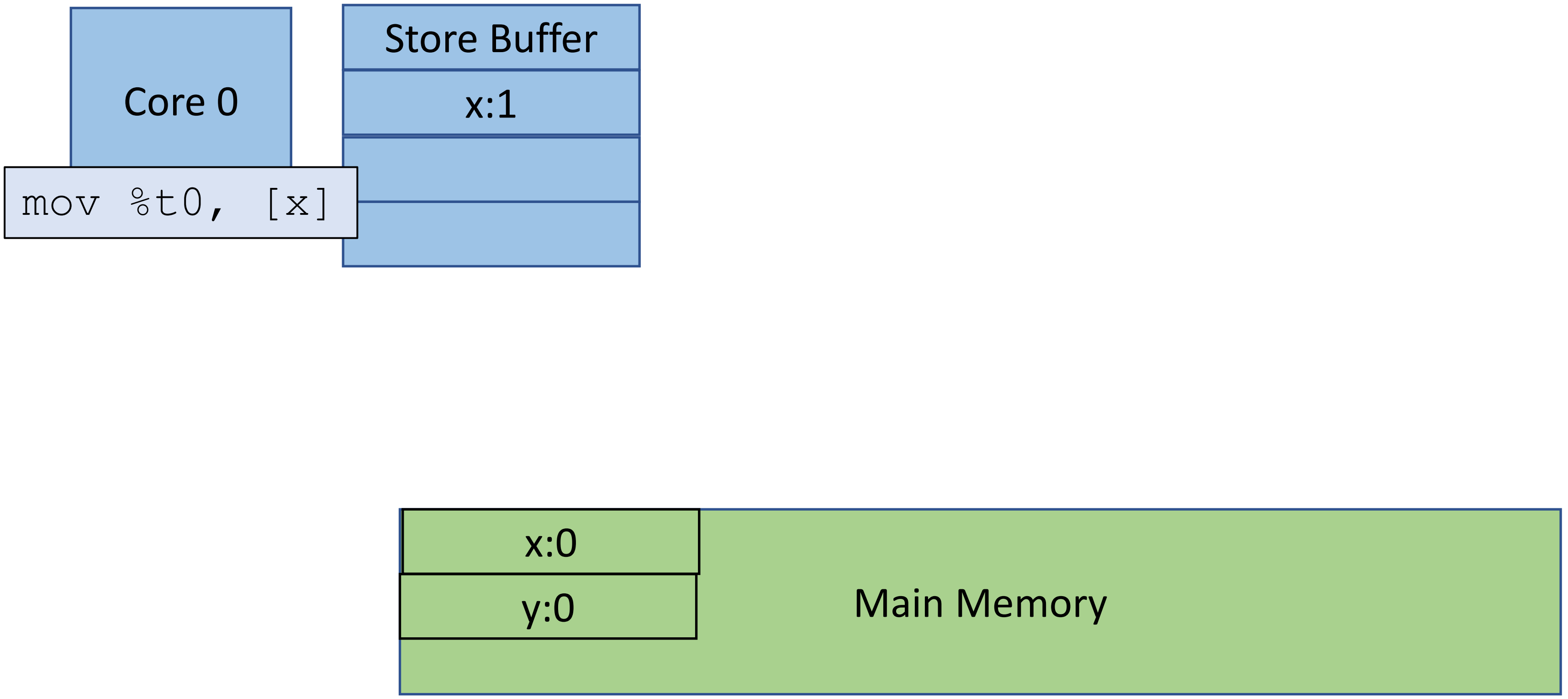
Store the value in the store buffer

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



Thread 0:

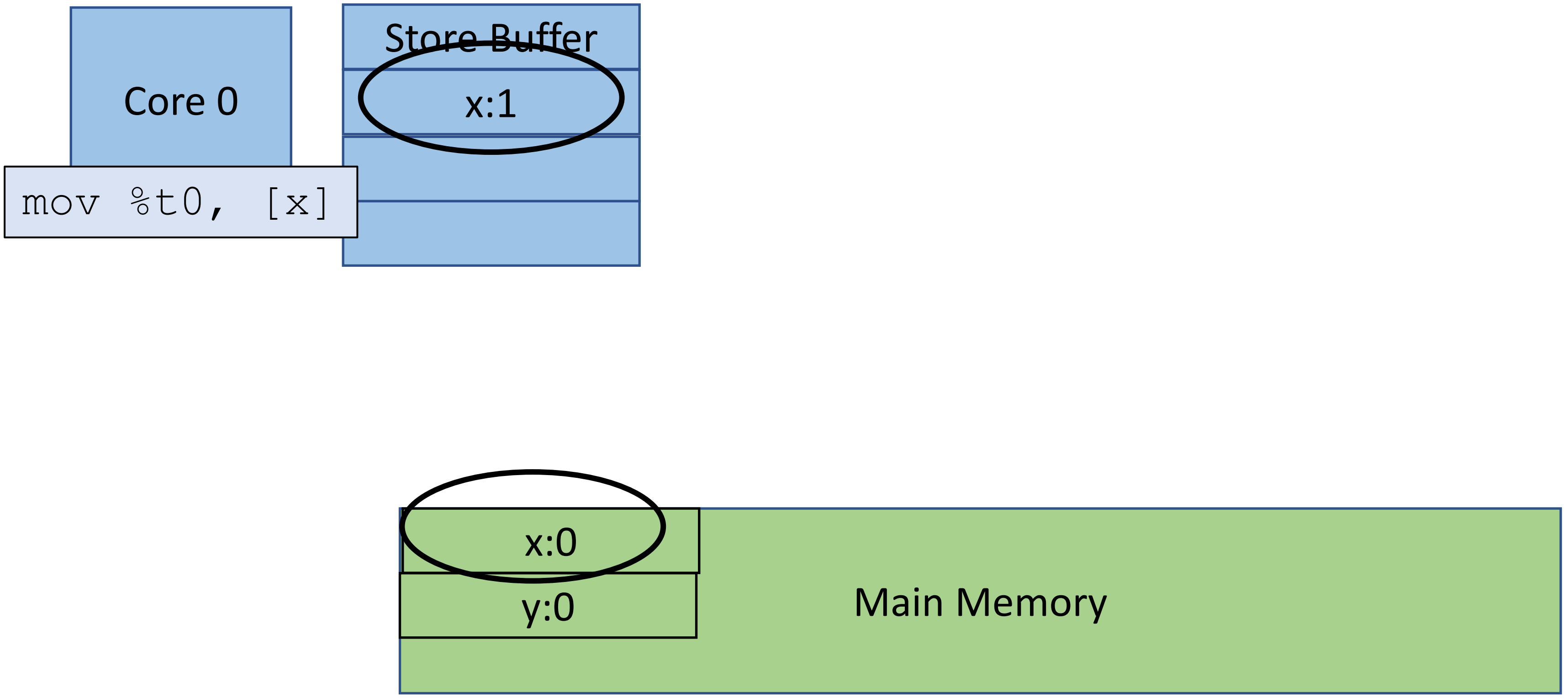
Next instruction



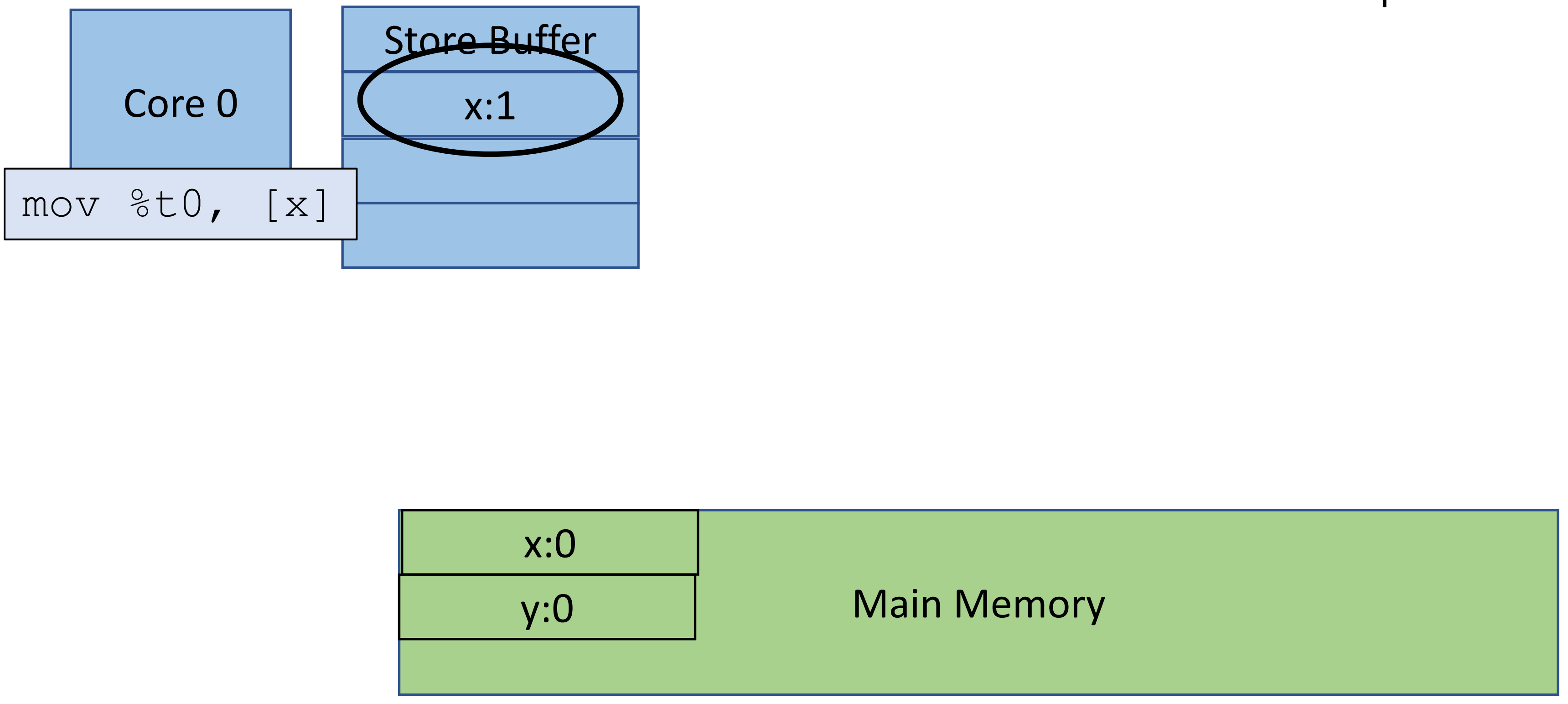
Thread 0:

Where to load??

Store buffer?
Main memory?



Thread 0:



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

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};
```

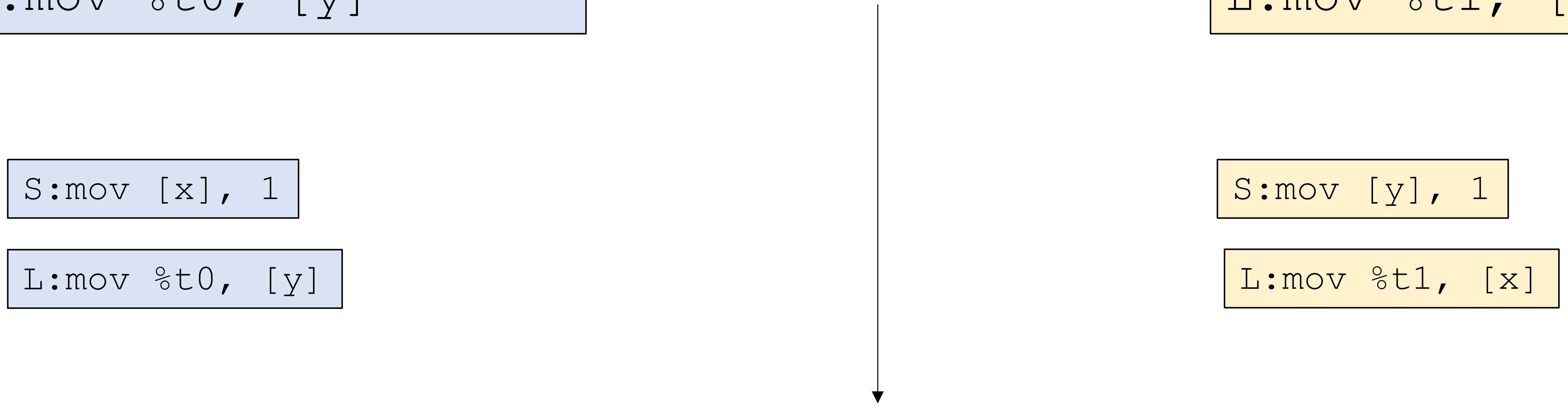
Another test
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



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

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

Global variable:

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

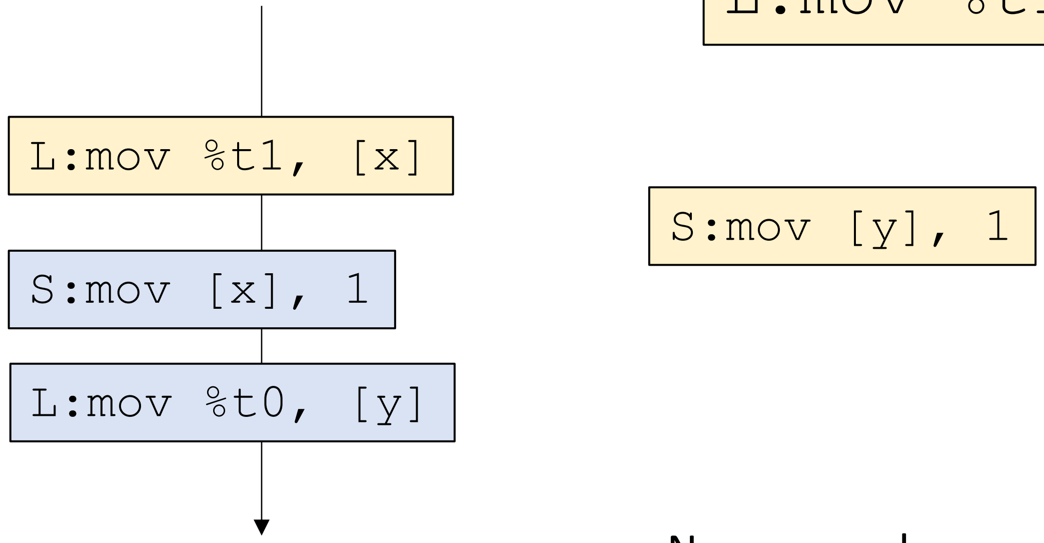
Another test
Can `t0 == t1 == 0`?

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`?

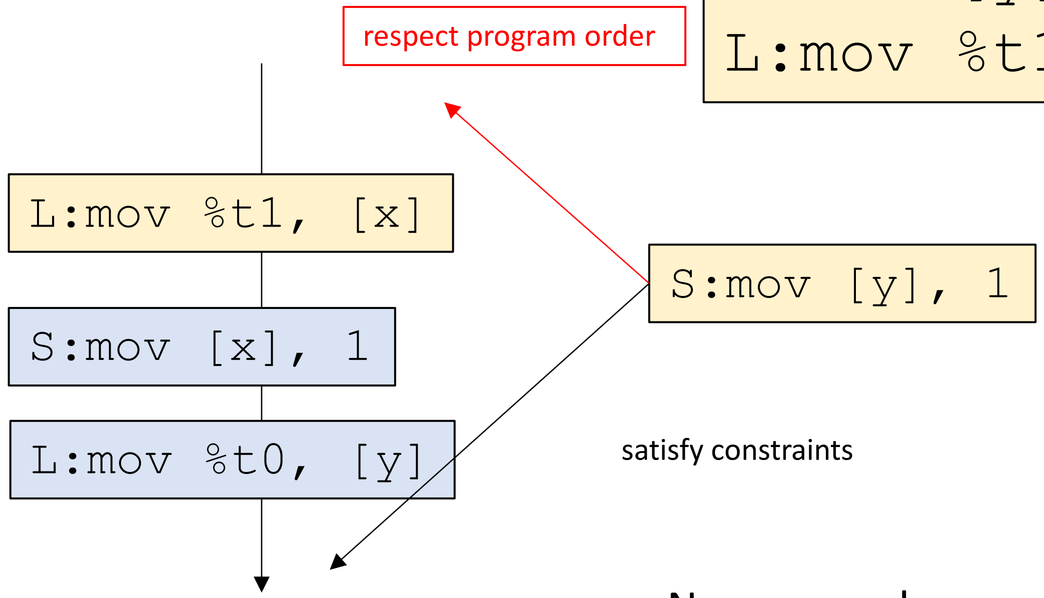
Thread 0:

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

we can ignore this condition!!

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

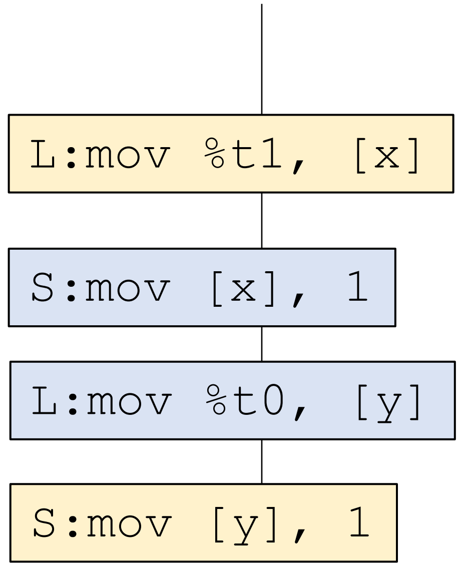
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



```
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`?

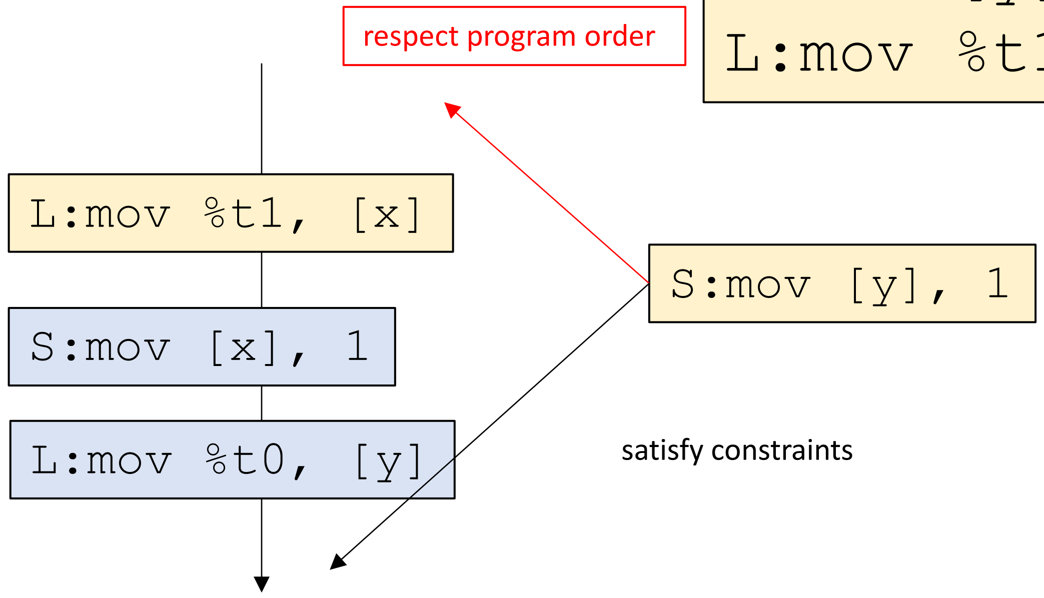
Thread 0:

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

Thread 1:

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

we can ignore this condition!!



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

Lets peak under the hood here

Global variable:

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

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

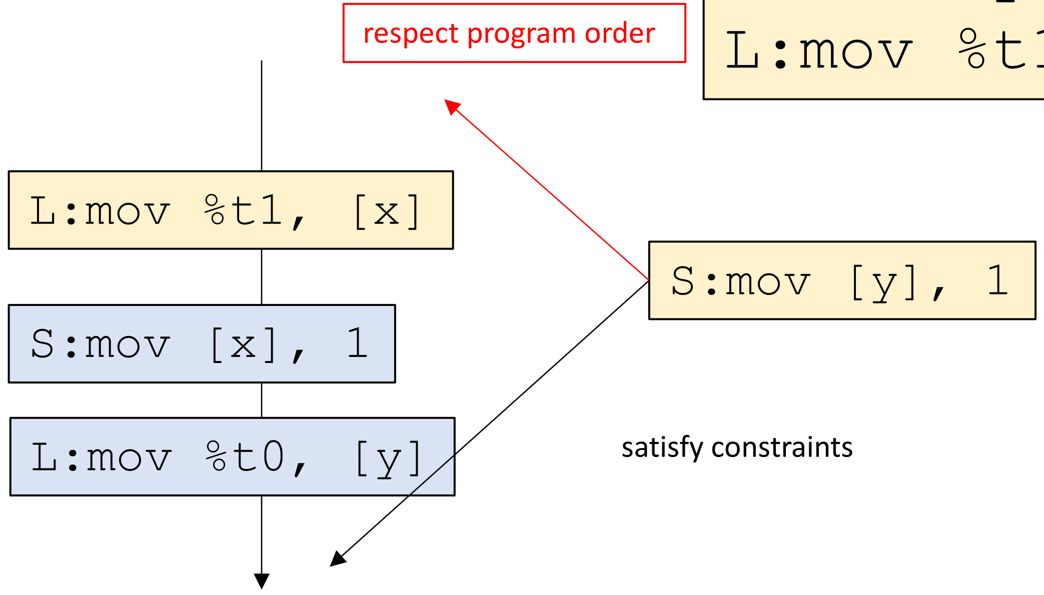
Thread 0:

```
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`?

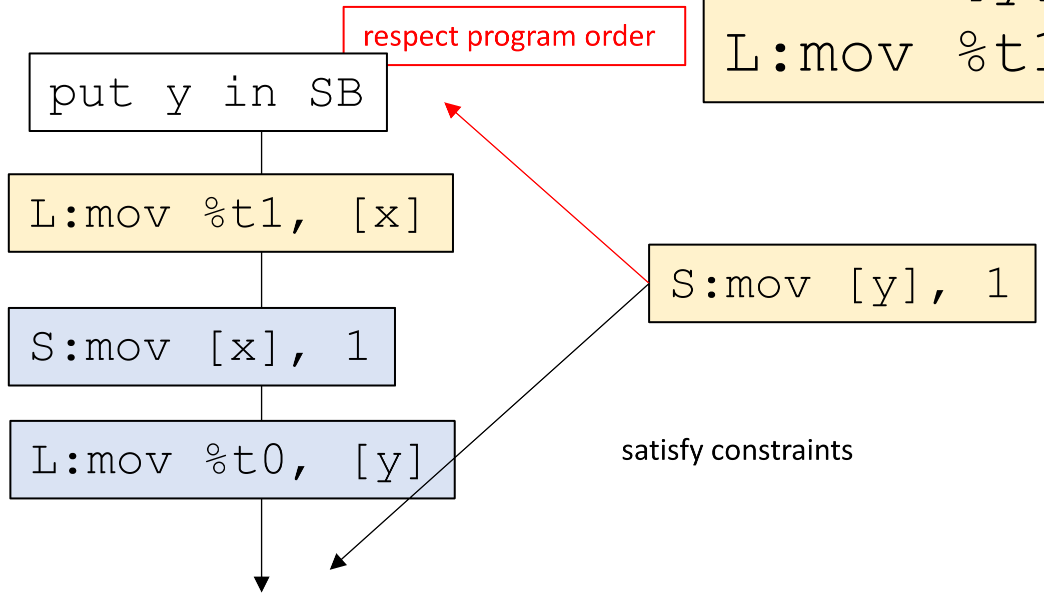
Thread 0:

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

Thread 1:

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

we can ignore this condition!!



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`?

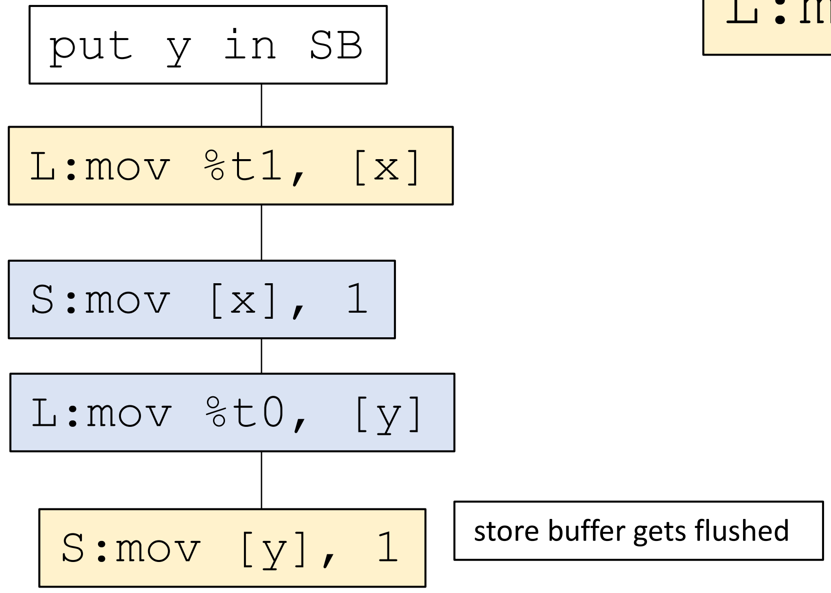
Thread 0:

```
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
```

Lets peak under the hood here

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

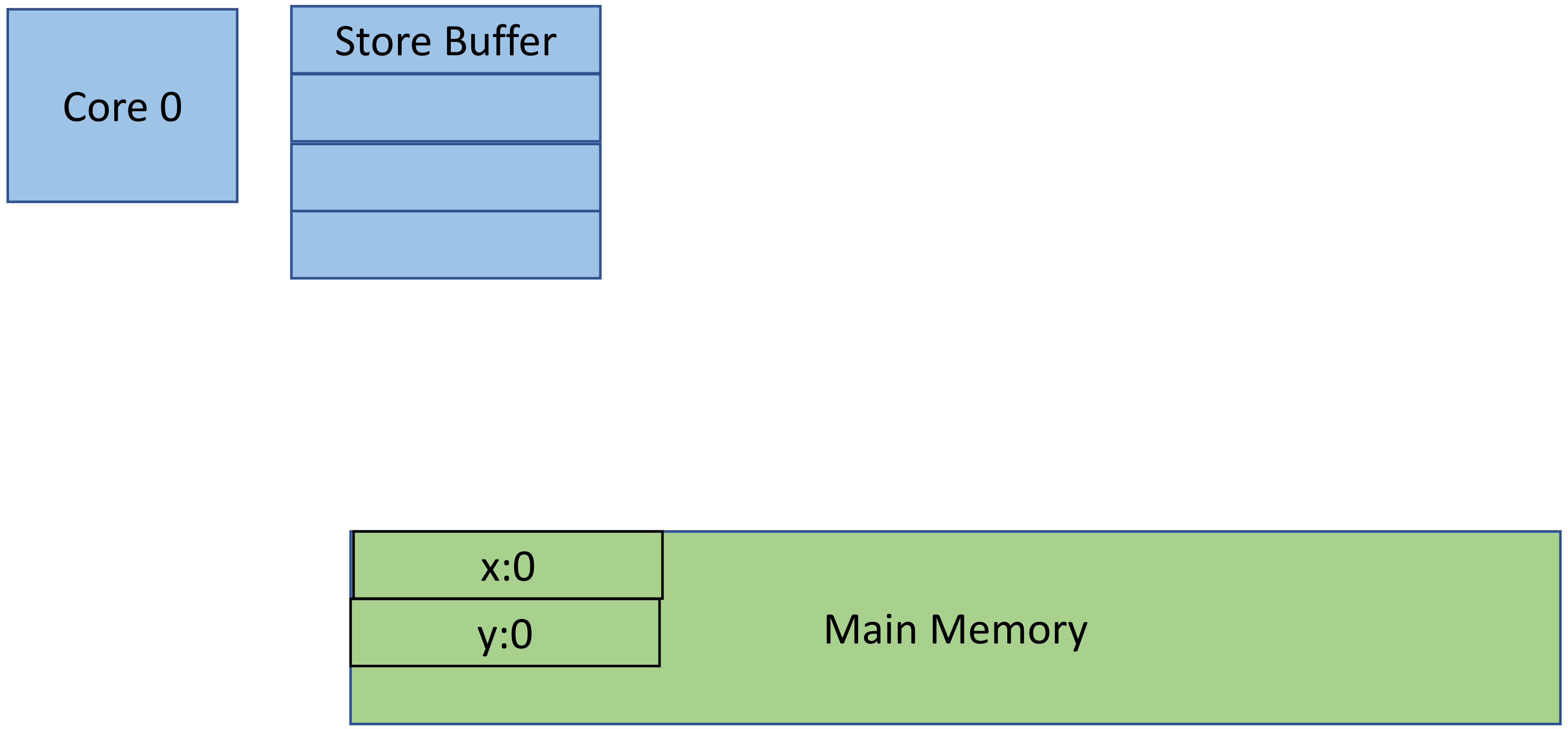
Questions

- Can stores be reordered with stores?

Thread 0:

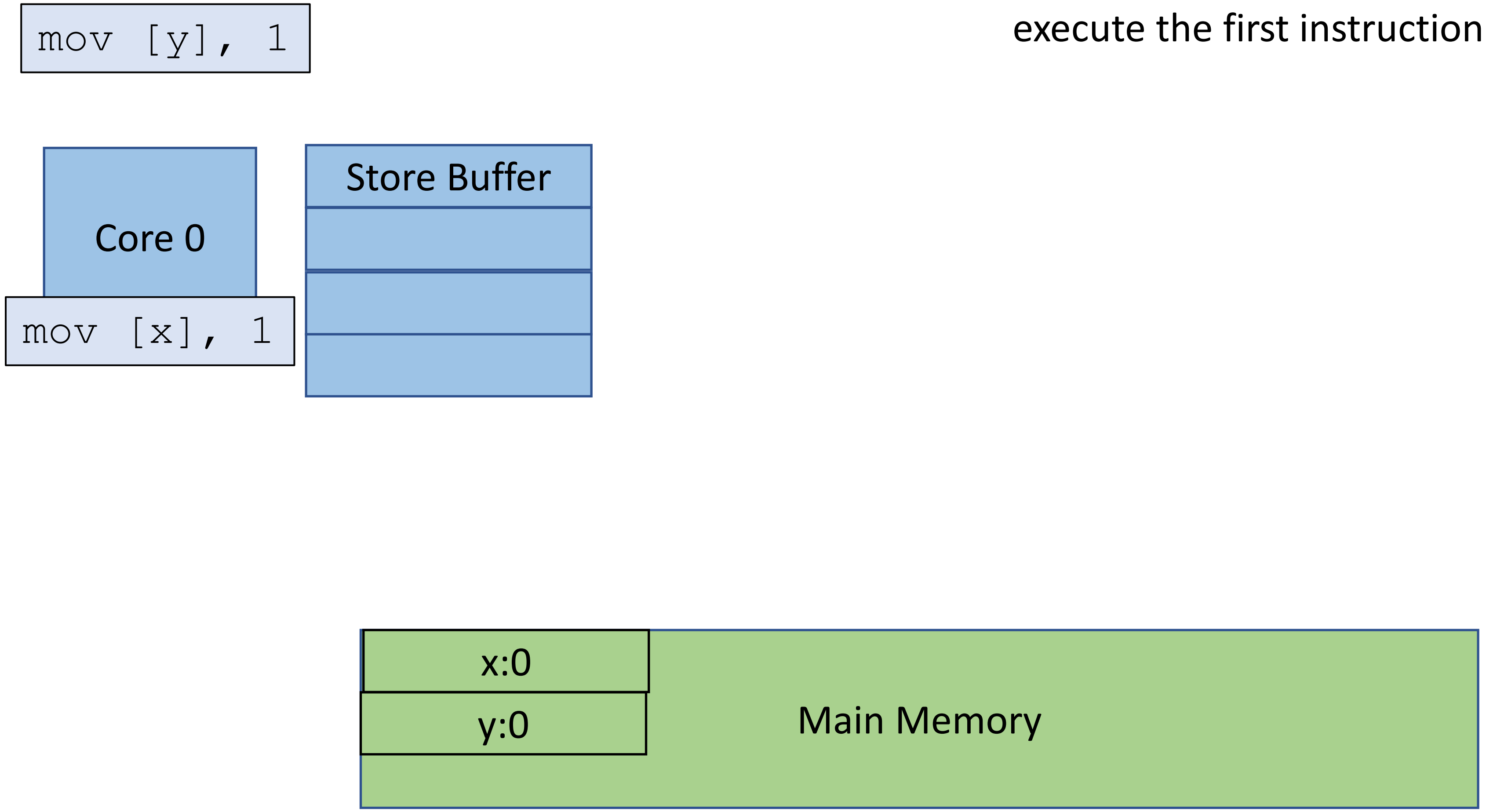
```
mov [x], 1
```

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



Thread 0:

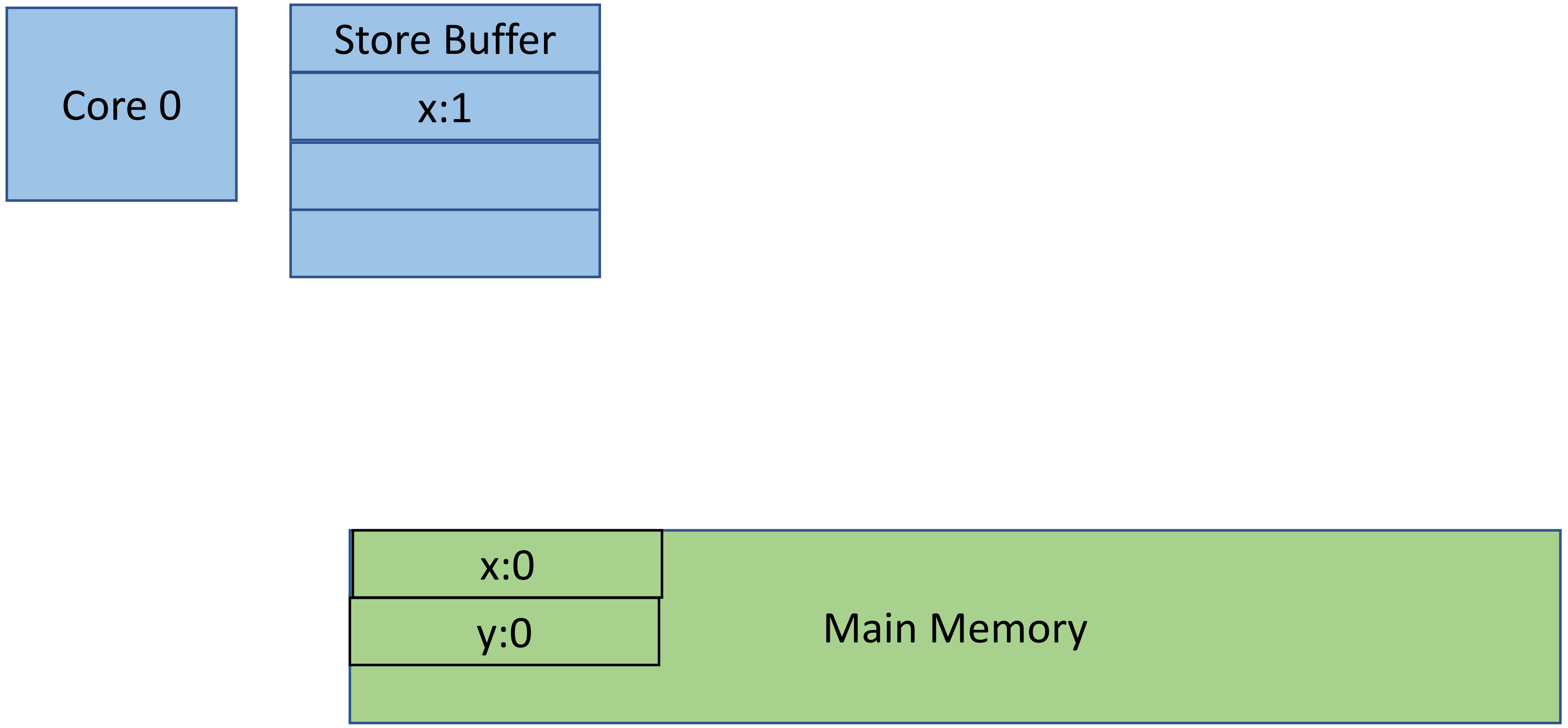
execute the first instruction



Thread 0:

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

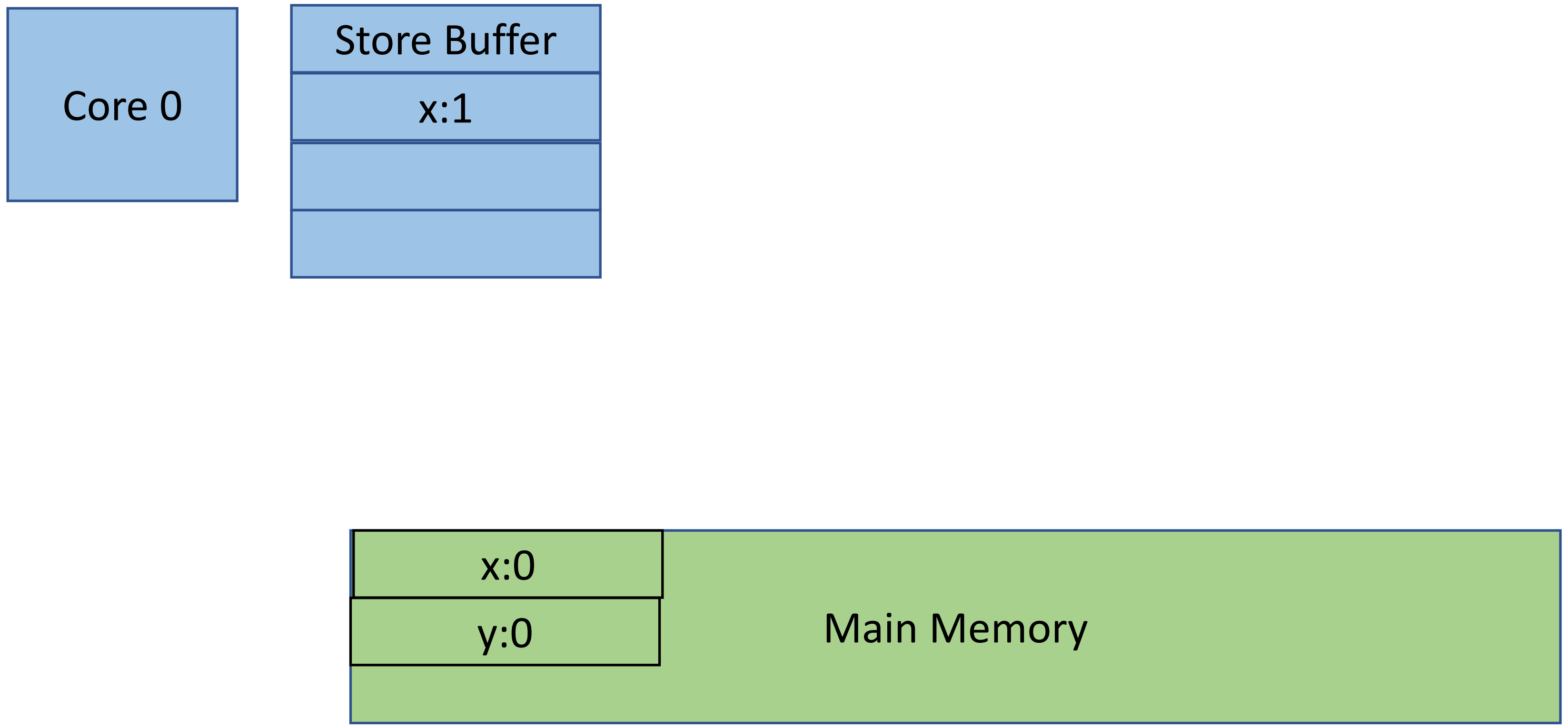
value goes into store buffer



Thread 0:

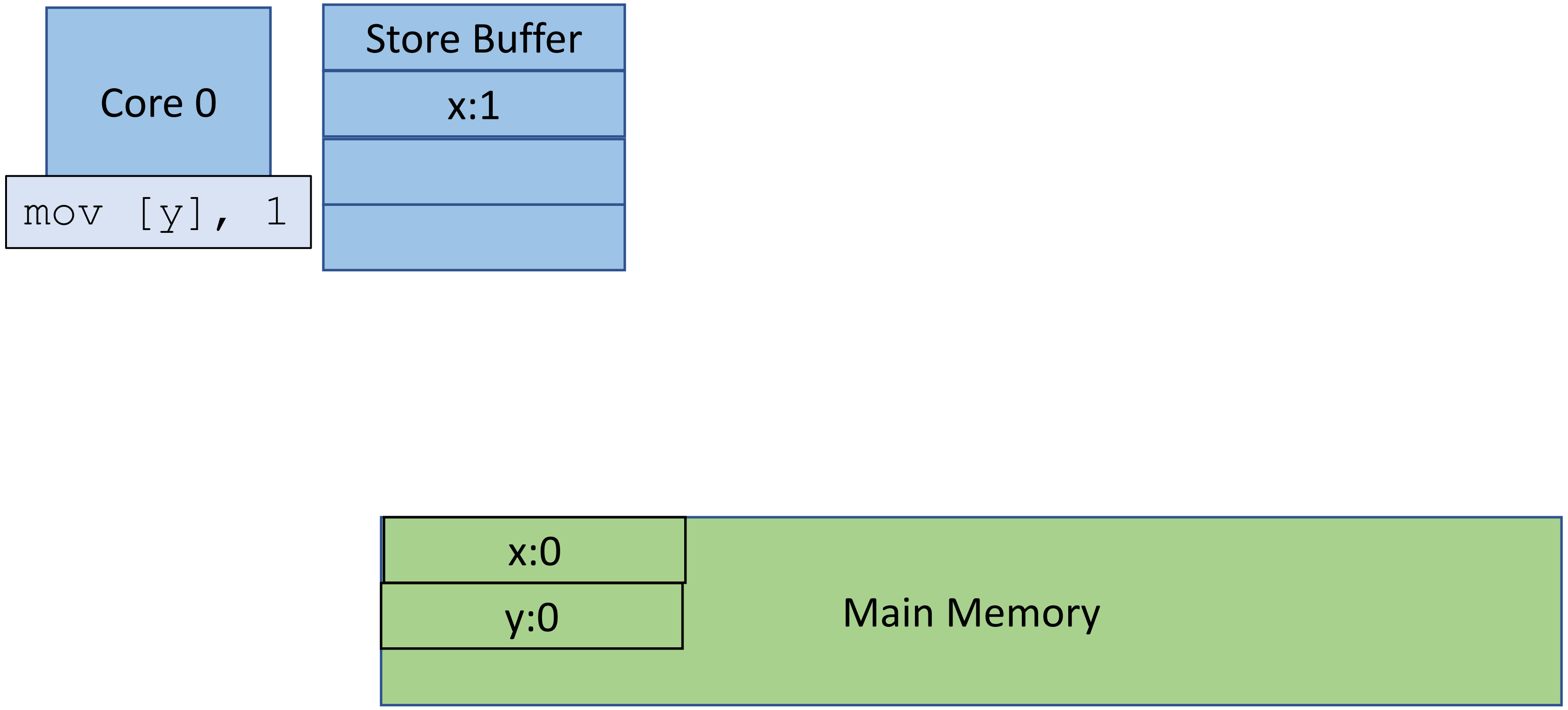
```
mov [y], 1
```

execute next instruction



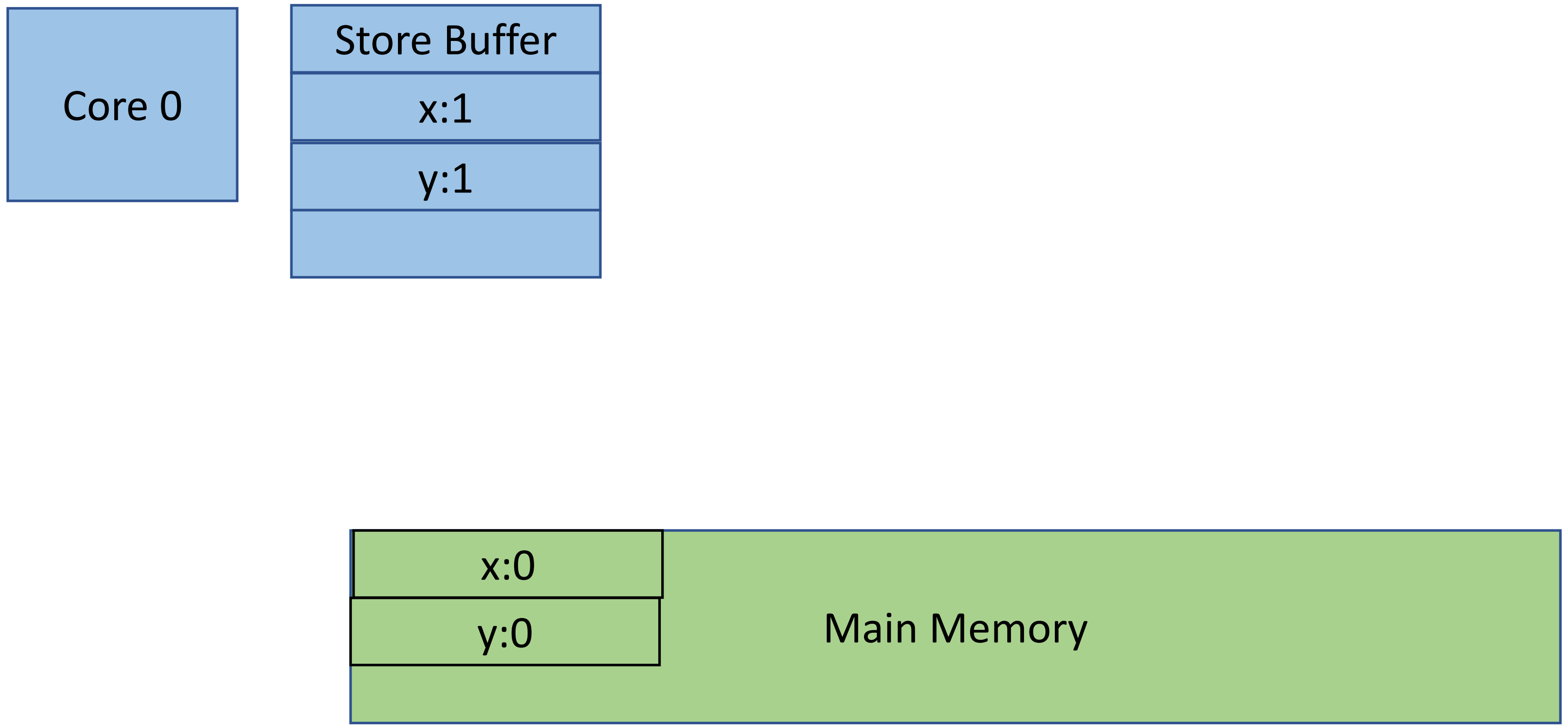
Thread 0:

execute next instruction

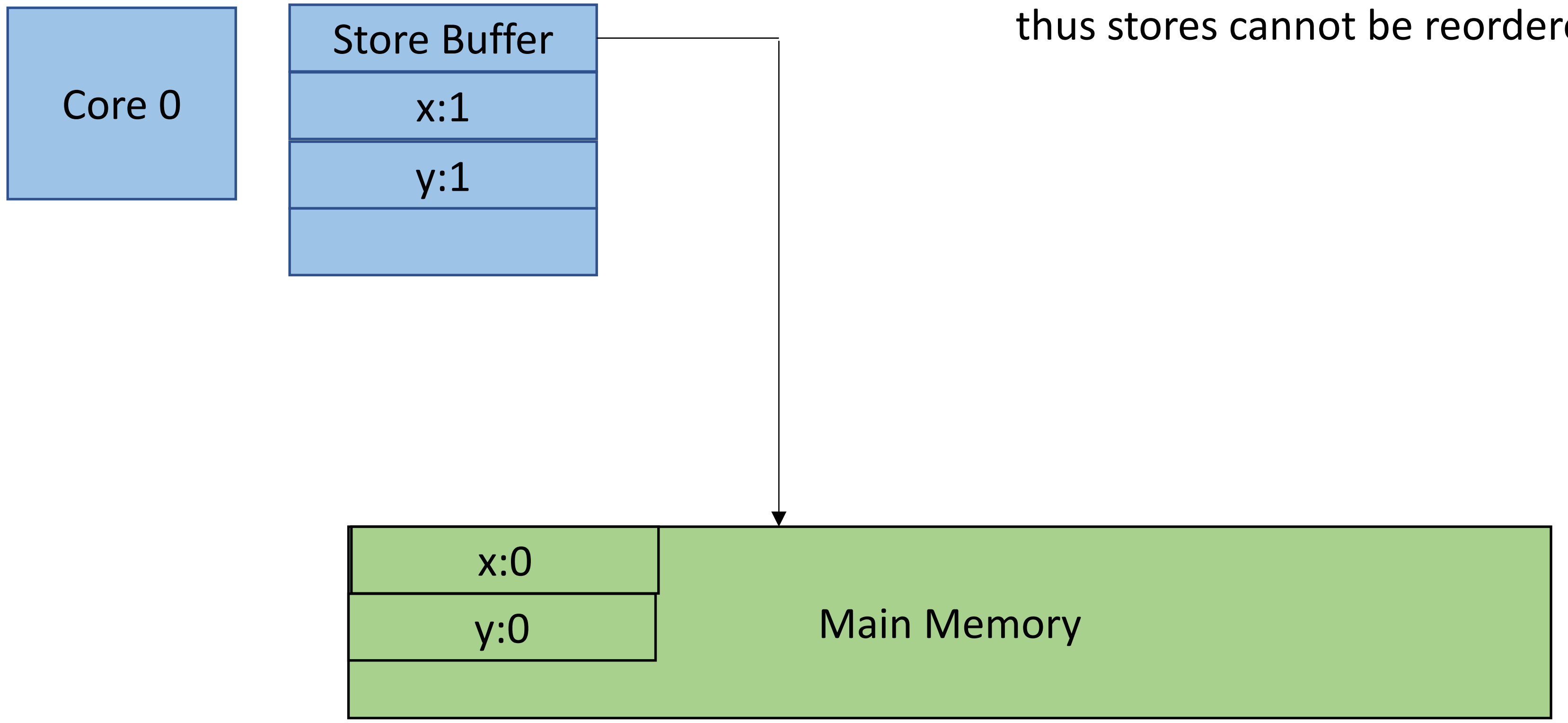


Thread 0:

value goes into the store buffer

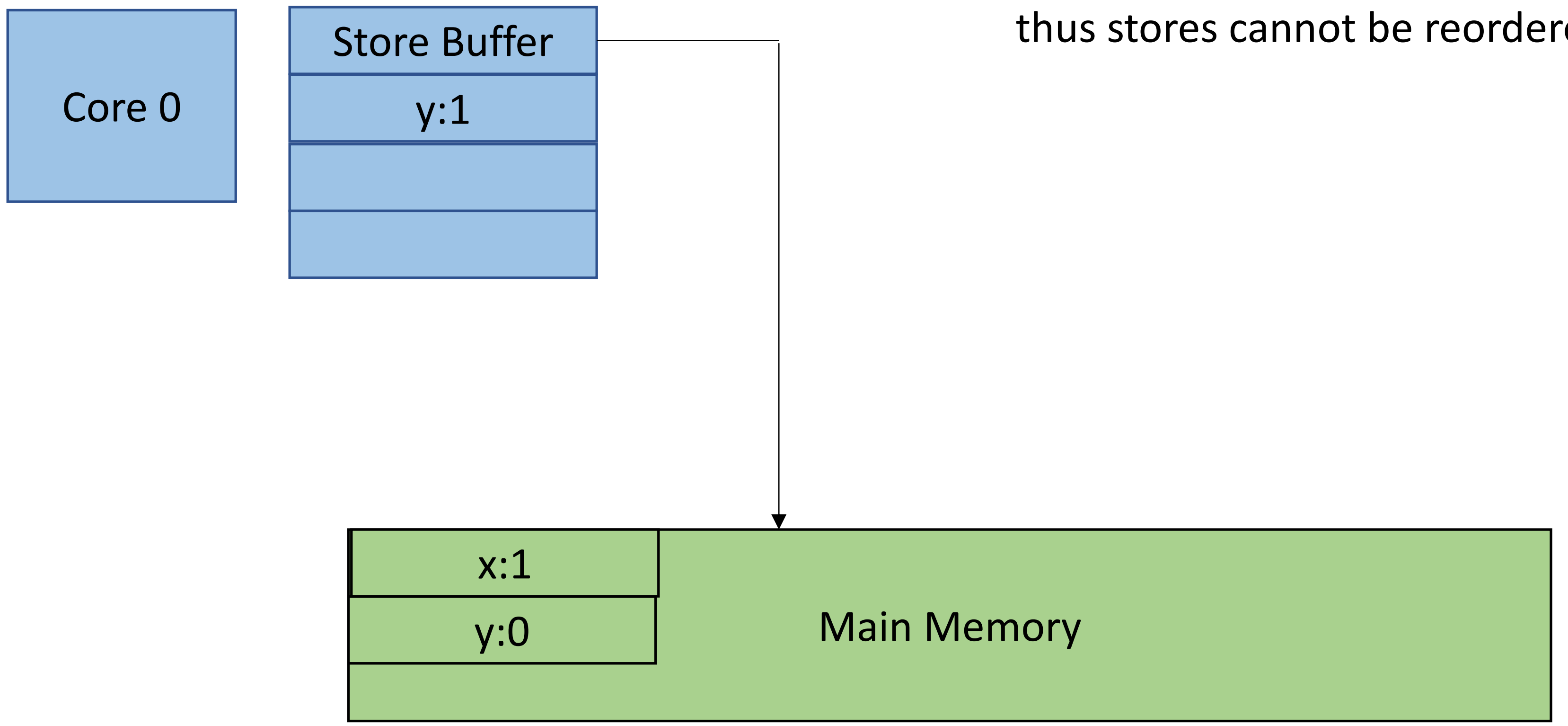


Thread 0:



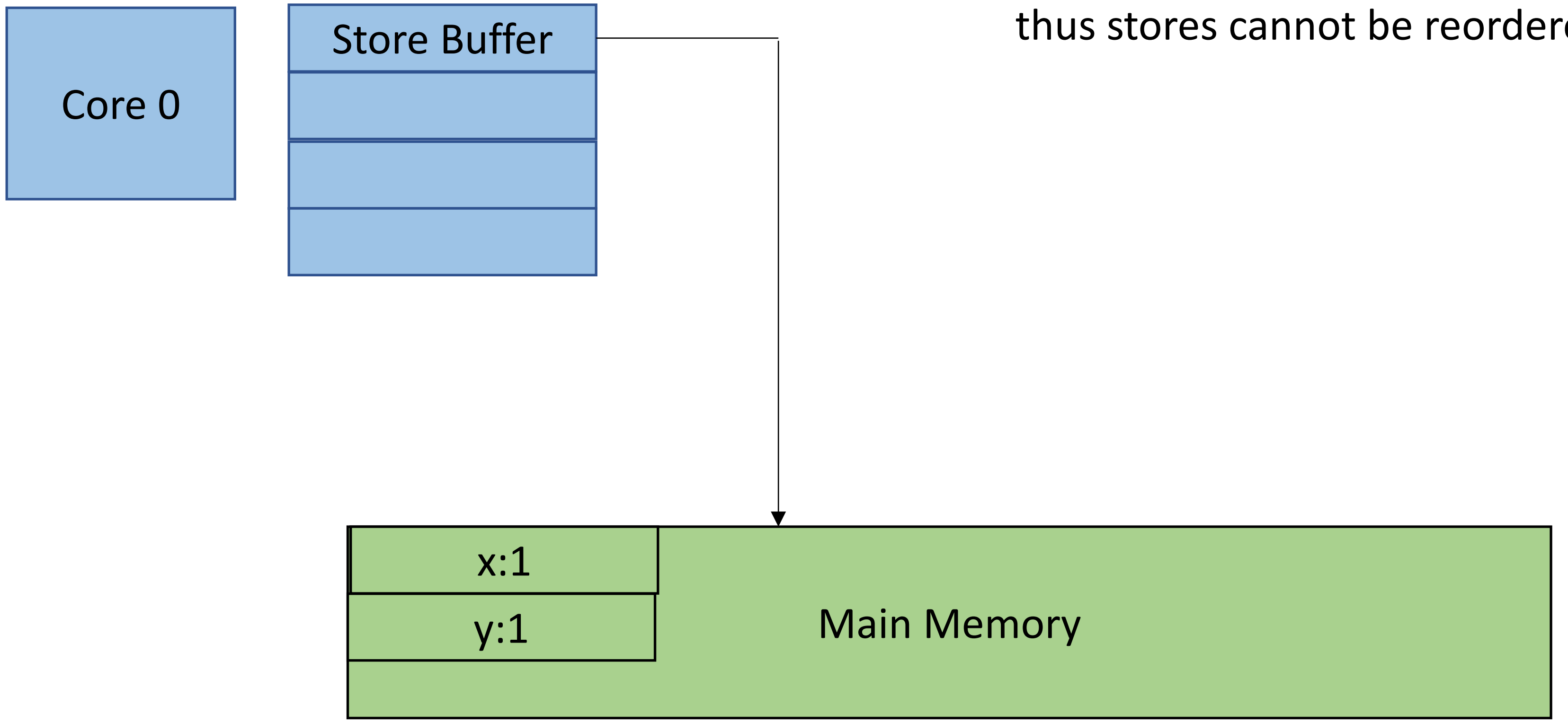
On x86, the store buffer trains in a FIFO way:
thus stores cannot be reordered

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]
```

```
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};
```

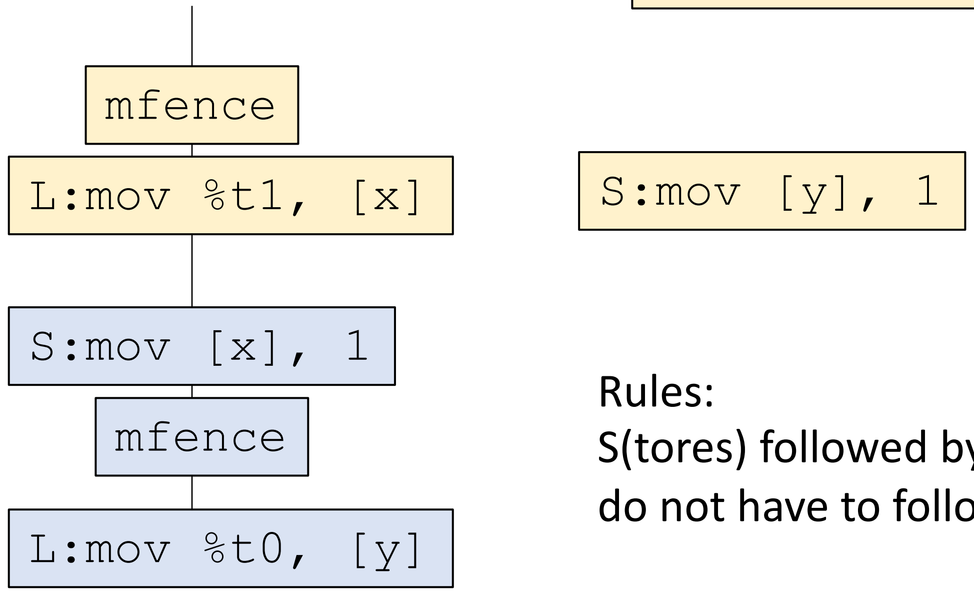
Another test
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



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

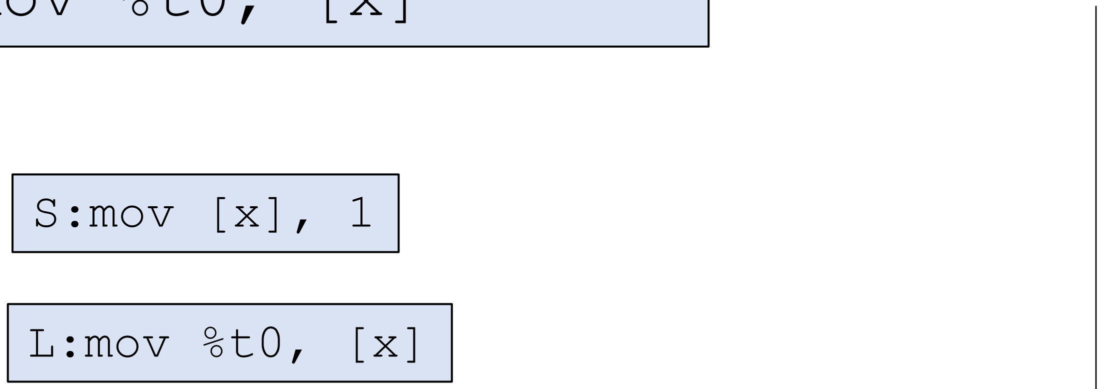
Global variable:

```
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

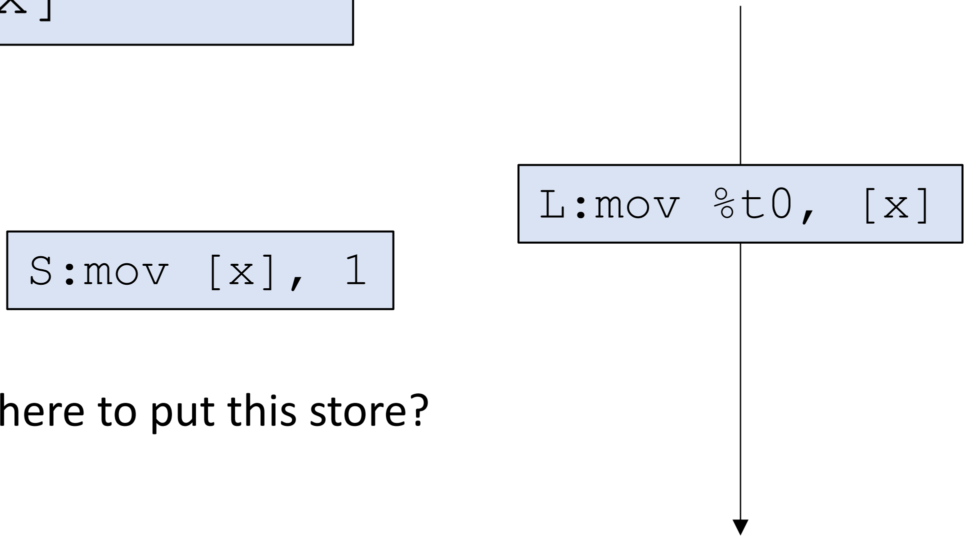
Global variable:

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

Another test
Can `t0 == 0`?

Thread 0:

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



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:

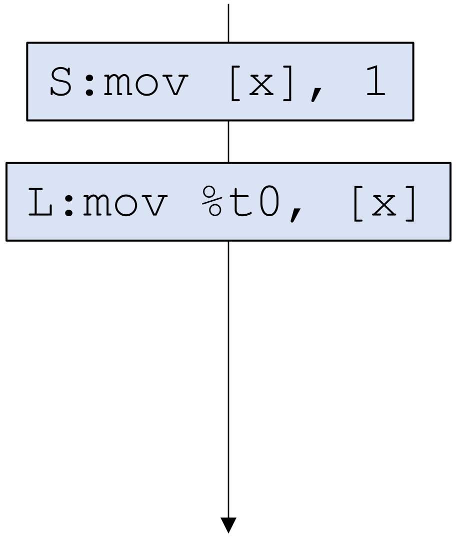
```
int x[1] = {0};
int y[1] = {0};
```

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

memory access 1

		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

memory access 1

		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

memory access 1

	L	S
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 (columns), memory access 1 (rows)

	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

memory access 0

memory access 1		L	S
	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

memory access 1		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
- **Some Examples**