

CSE113: Parallel Programming

Feb. 18, 2022

- **Topics:**

- Memory consistency models:
 - Relaxed memory consistency
 - Examples
 - Compiling memory consistency

target machine TSO (x86)		
	L	S
L	NO	different address
S	NO	No

Announcements

- HW 4 is out
 - Due on March 4
 - Please don't share timing results until next Monday
 - You mostly had what you needed at the end of Friday
 - You will have everything you need to do it at the end of lecture today
- Grades for HW 2 are out
 - Let us know by next Monday if you have any issues
- Grades for Midterm are on their way
 - Expect them by Monday

Today's Quiz

- Due Tomorrow by midnight; please do it!

Previous quiz

A relaxed memory execution refers to:

- ☐ An execution where some stores failed to reach main memory
- ☐ Any execution which contains a data-conflict
- ☐ An execution that utilizes the processor's store buffer
- ☐ An execution that is not sequentially consistent

Previous quiz

Which of the following memory accesses pairs, when they appear in program order, does x86 allow to be re-ordered?

☐ Load followed by a Store

☐ Load followed by a Load

☐ Store followed by a Store

☐ Store followed by a Load

Previous quiz

- Didn't save the question:
 - What is the difference between a fence and a barrier?
- The naming can get confusing!
- [https://en.wikipedia.org/wiki/Barrier_\(computer_science\)](https://en.wikipedia.org/wiki/Barrier_(computer_science))
- https://en.wikipedia.org/wiki/Memory_barrier

Review

Sequential consistency and litmus tests

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);
```

```
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();
```

```
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 $t1 == 1$

int t0 = y.load();

x.store(1);

y.store(1);

int t1 = x.load();



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);
```

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

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

How about:

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



Thread 1:

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

```
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);
```

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

no where for this one to go!

How about:

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

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

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

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

Thread 1:

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

Global variable:

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

Another test

Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



Global variable:

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

Another test

Can `t0 == t1 == 0`?

Thread 0:

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

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

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

Thread 1:

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

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

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

Global variable:

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

Thread 0:

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

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

Another test

Can `t0 == t1 == 0`?

Thread 1:

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

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

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

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

no place for this one!

X86 TSO operational model

Thread 0:

mov [x], 1

mov %t0, [y]

Core 0

Thread 1:

mov [y], 1

mov %t1, [x]

Core 1



Thread 0:

mov %t0, [y]

Core 0

mov [x], 1

execute first instruction
what happens to the stores?

Thread 1:

mov %t1, [x]

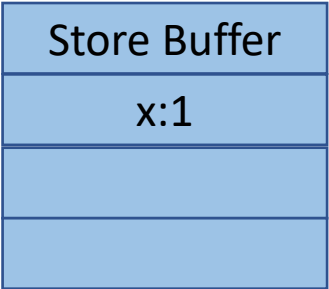
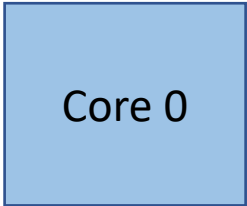
Core 1

mov [y], 1



Thread 0:

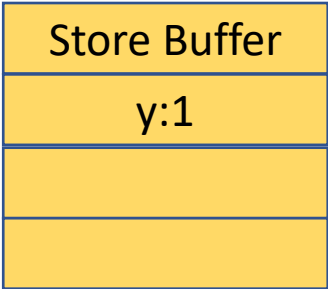
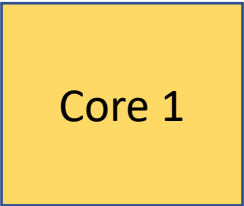
```
mov %t0, [y]
```



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

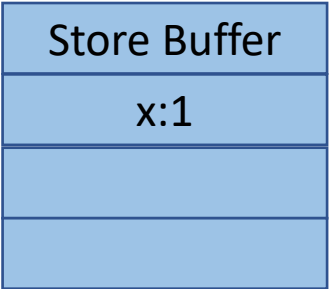
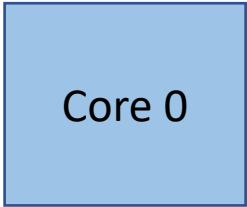
Thread 1:

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



Thread 0:

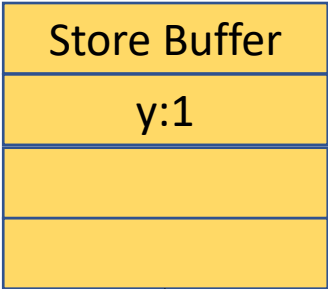
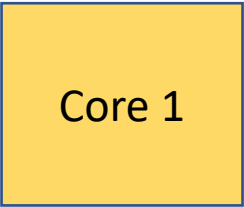
```
mov %t0, [y]
```



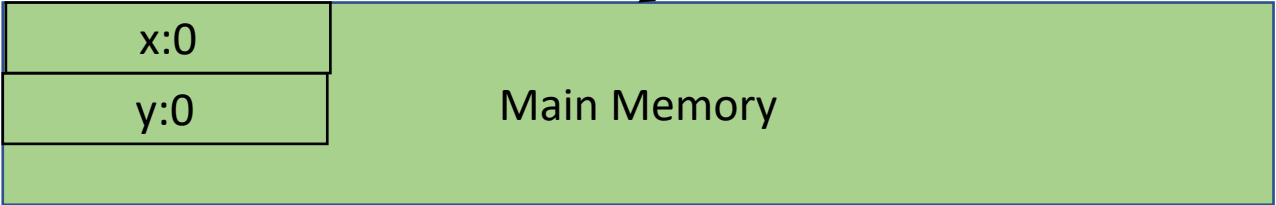
X86 cores contain a store buffer; holds stores before going to main memory

Thread 1:

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

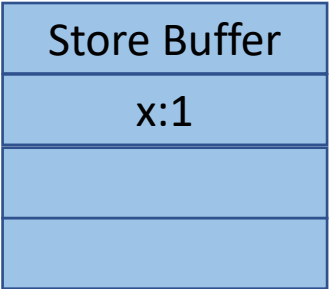
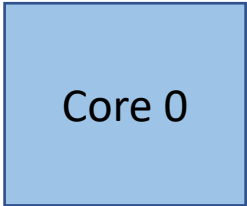


eventually they flush to main memory



Thread 0:

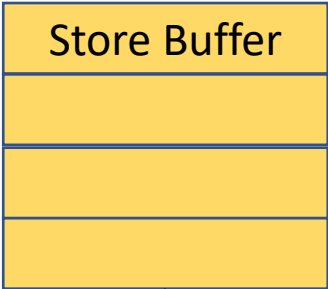
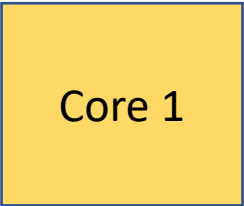
```
mov %t0, [y]
```



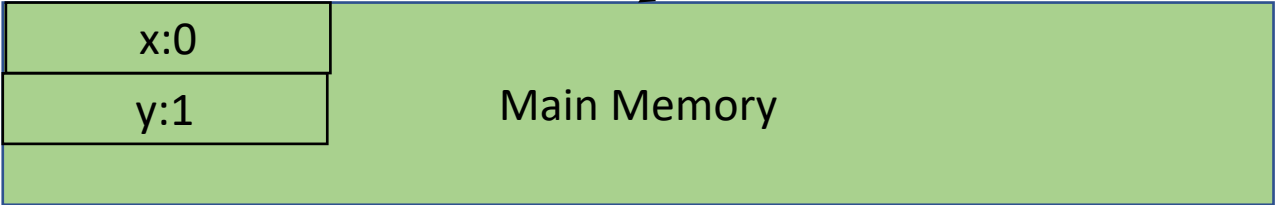
X86 cores contain a store buffer; holds stores before going to main memory

Thread 1:

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



eventually they flush to main memory



Thread 0:

mov [x], 1

mov %t0, [y]

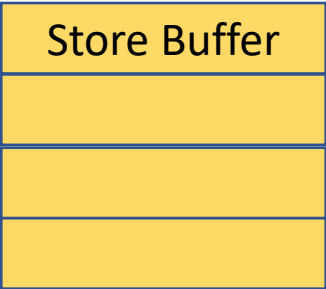
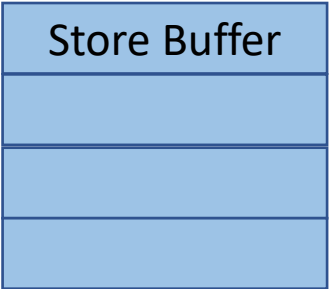
rewind

Thread 1:

mov [y], 1

mov %t1, [x]

Core 0

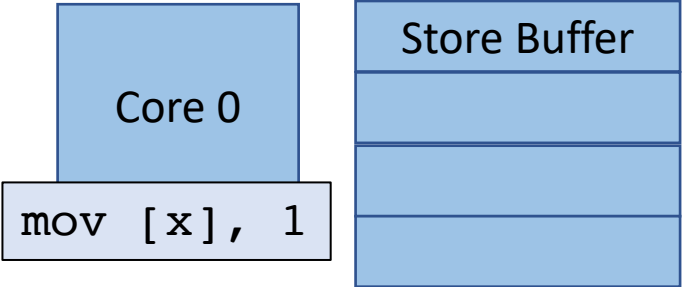


Core 1



Thread 0:

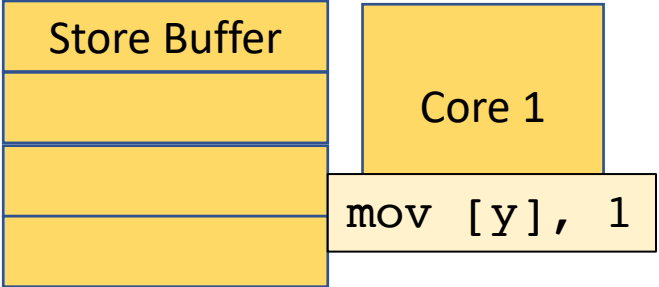
mov %t0, [y]



execute first instruction

Thread 1:

mov %t1, [x]



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

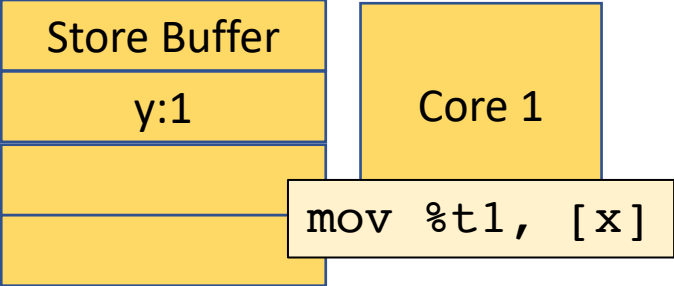
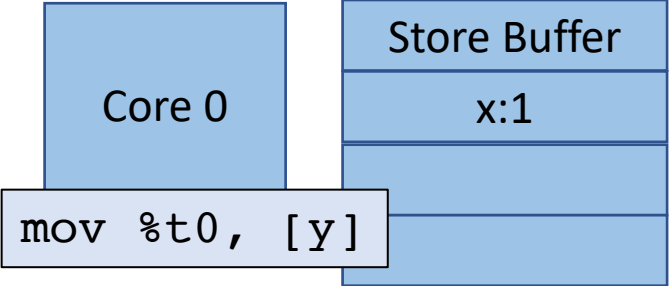
Core 1

x:0	Main Memory
y:0	

Thread 0:

Thread 1:

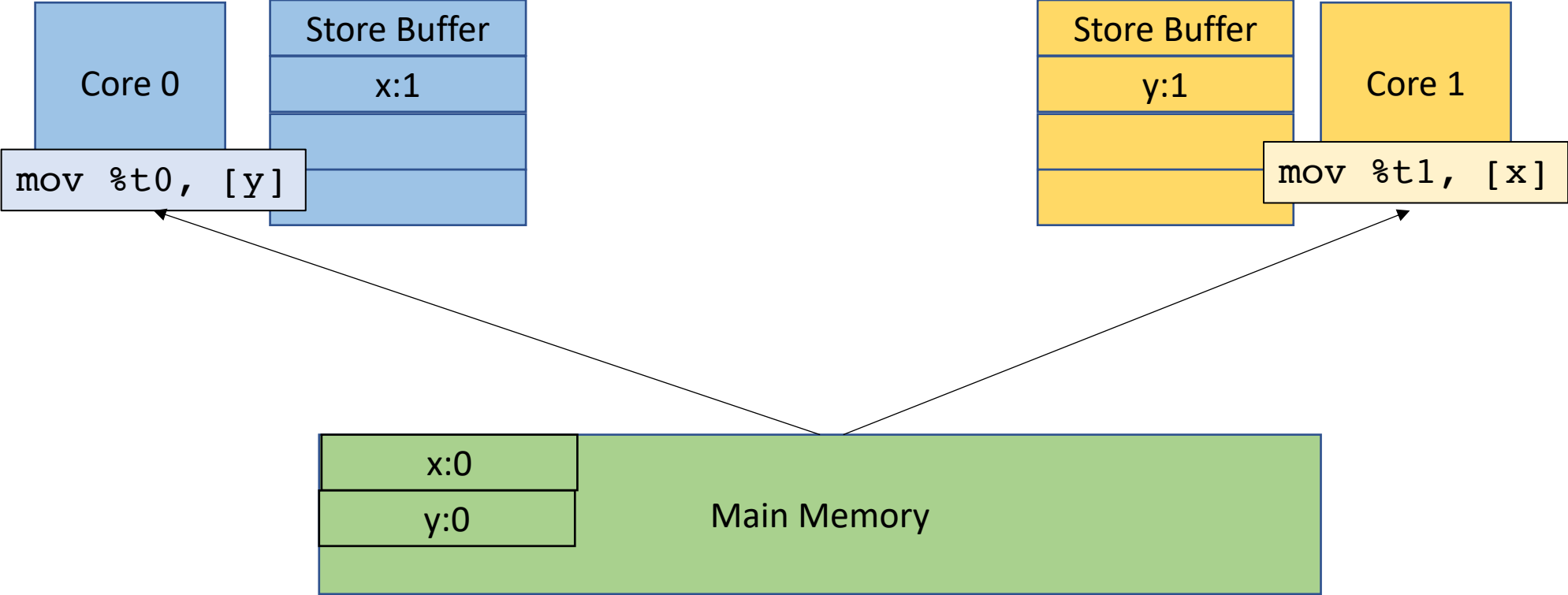
Execute next instruction



Thread 0:

Thread 1:

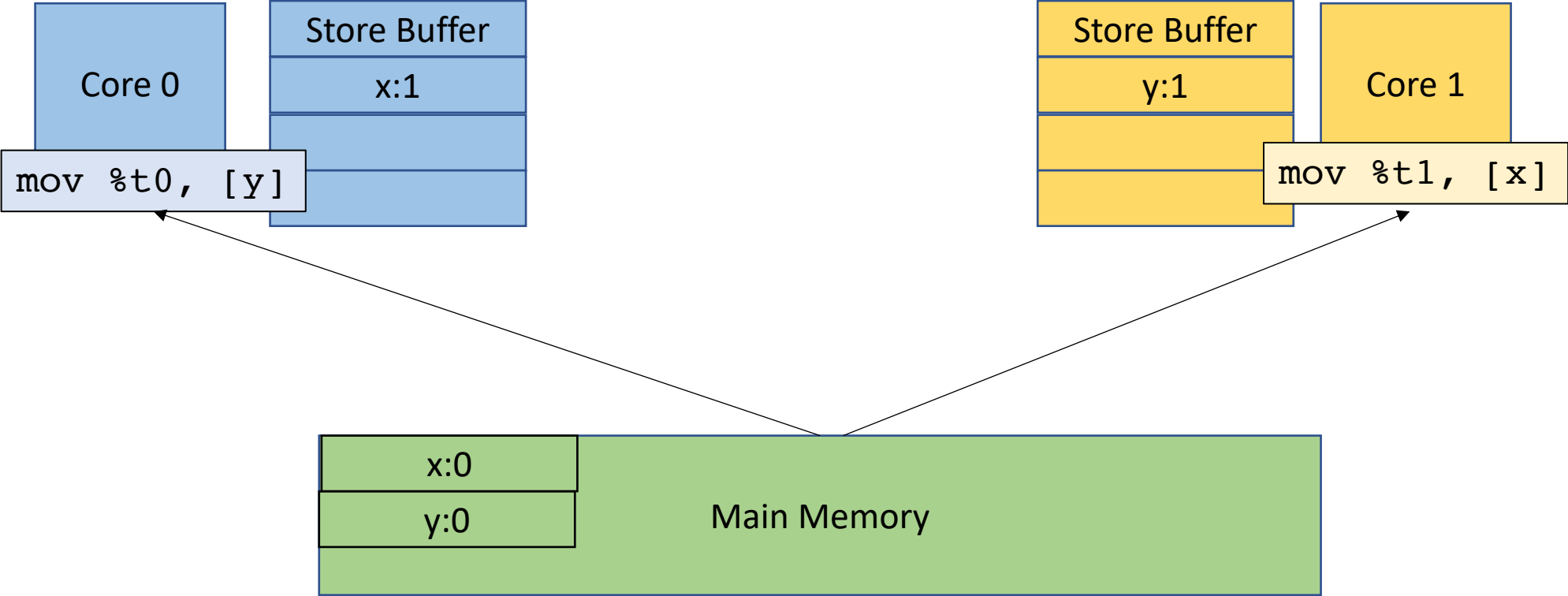
Values get loaded from memory



Thread 0:

Thread 1:

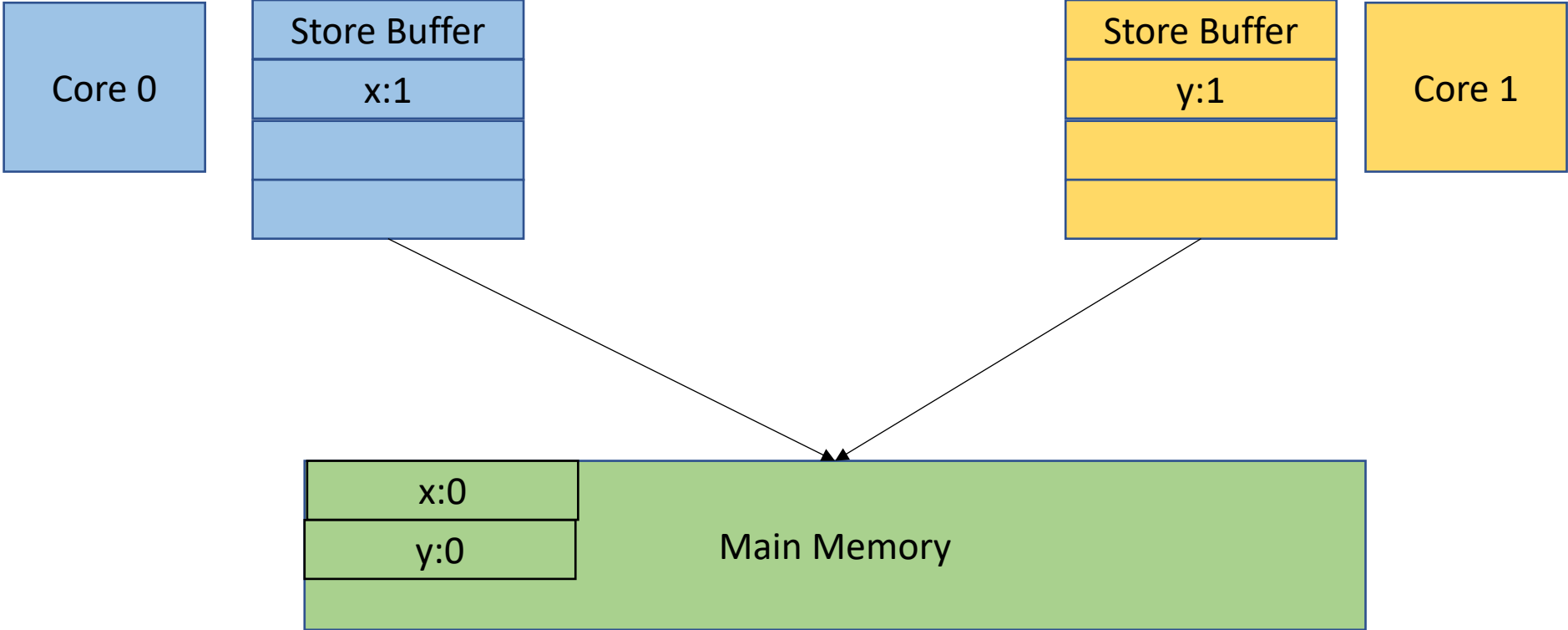
we see `t0 == t1 == 0!`



Thread 0:

Thread 1:

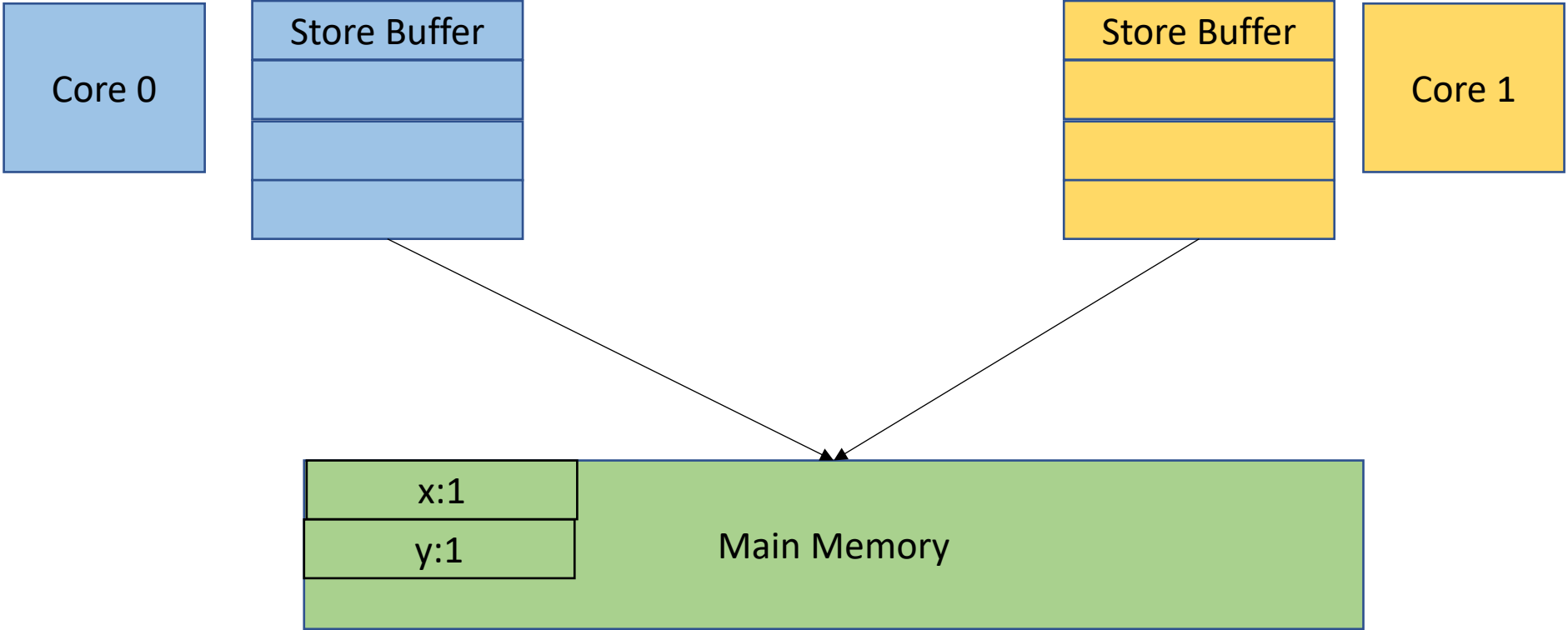
Store buffers are drained eventually



Thread 0:

Thread 1:

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



X86 TSO programming model

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]
```

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

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

Thread 1:

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

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

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



Global variable:

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

Another test

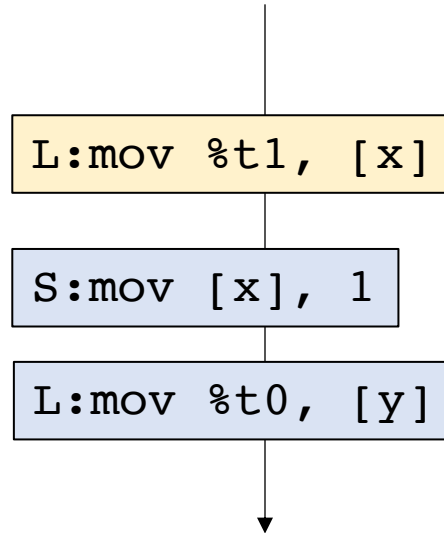
Can `t0 == t1 == 0`?

Thread 0:

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

Thread 1:

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



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

Now we make a new rule:

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

Global variable:

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

Another test

Can `t0 == t1 == 0`?

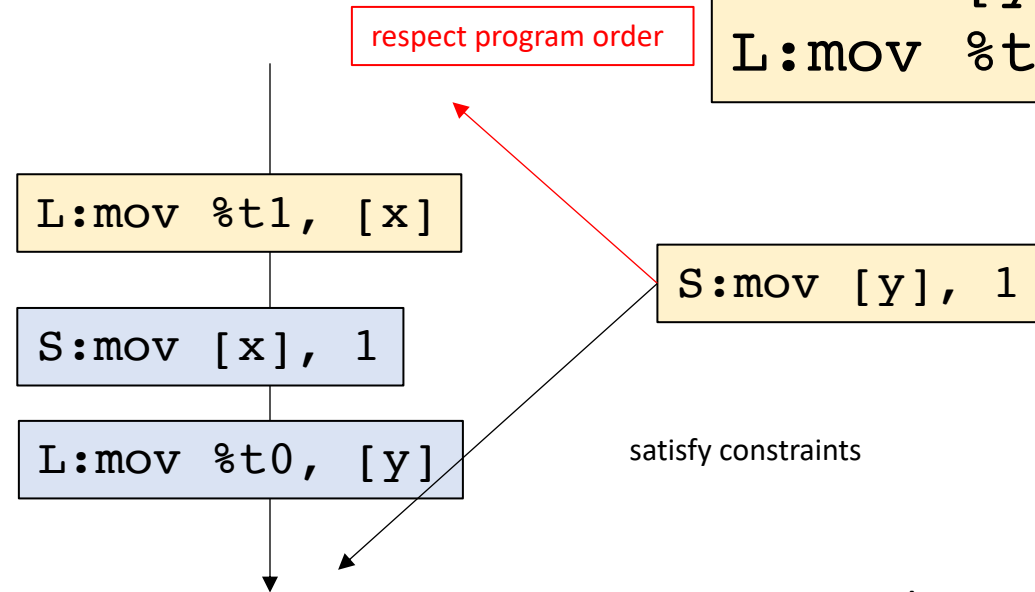
Thread 0:

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

Thread 1:

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

we can ignore this condition!!



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};
```

Thread 0:

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

Another test

Can `t0 == t1 == 0`?

we can ignore this condition!!

L:mov %t1, [x]

S:mov [x], 1

L:mov %t0, [y]

S:mov [y], 1

Thread 1:

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

Now we can satisfy the condition!

Global variable:

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

Thread 0:

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

Lets peak under the hood here

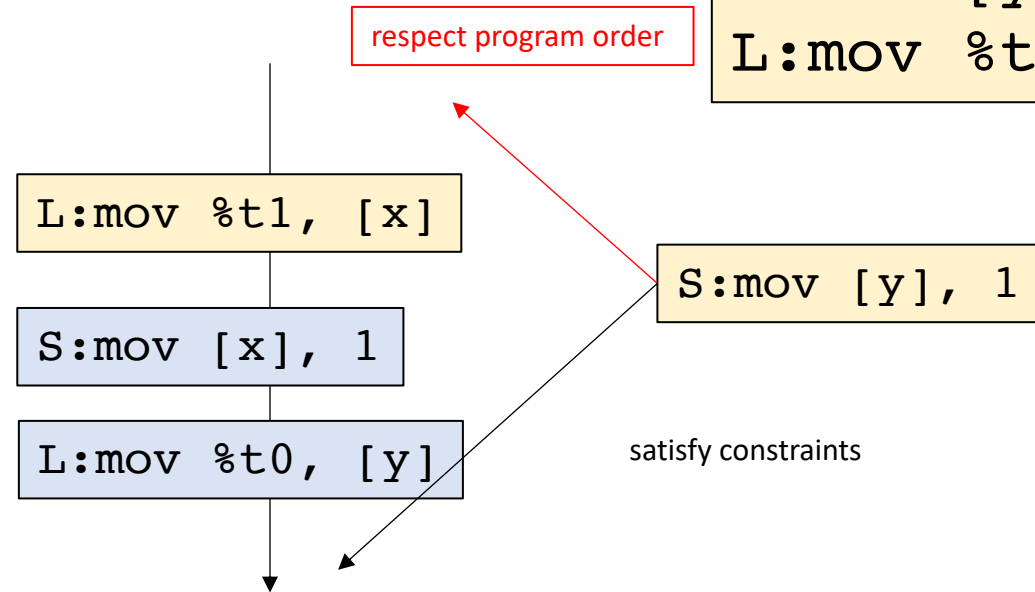
Another test

Can `t0 == t1 == 0`?

Thread 1:

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

we can ignore this 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

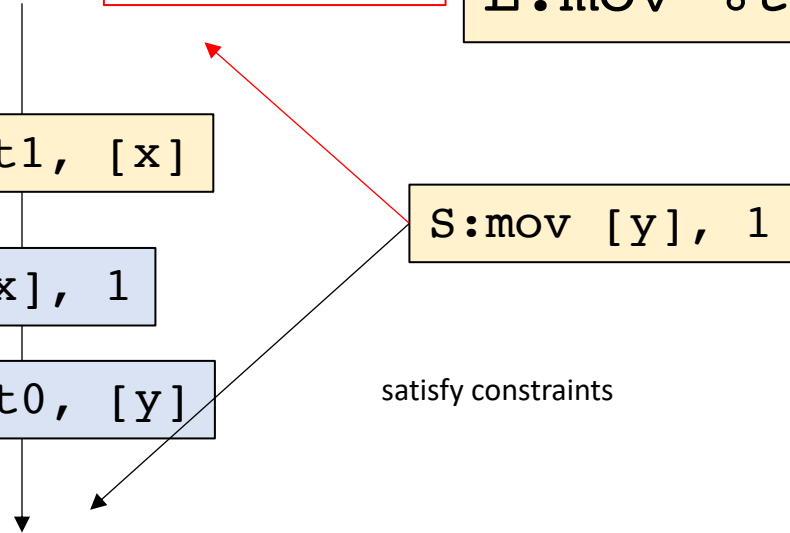
L:mov %t0, [y]

S:mov [y], 1

satisfy constraints

Lets peak under the hood here

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



Global variable:

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

Thread 0:

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

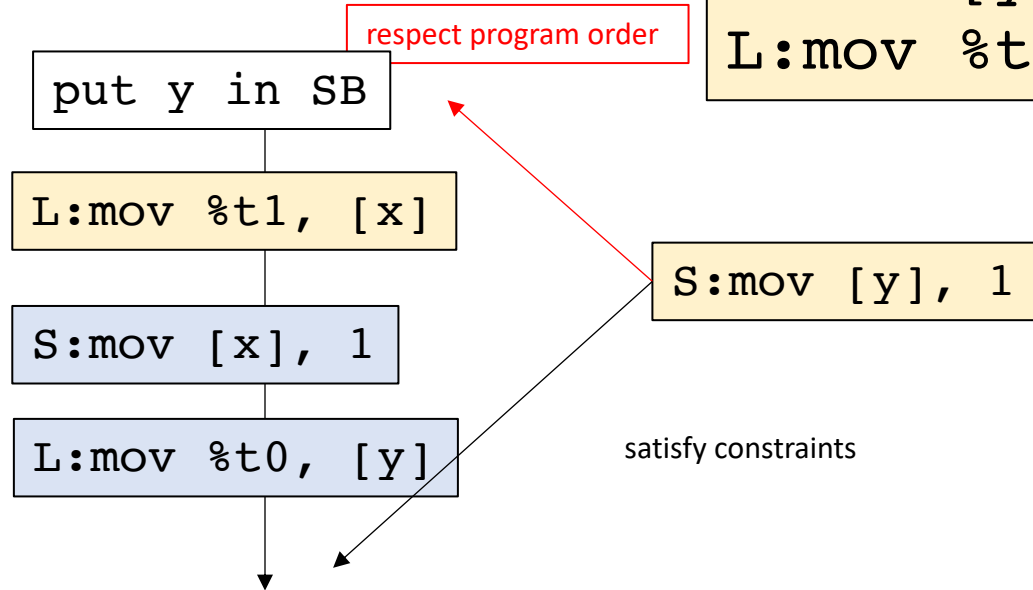
Lets peak under the hood here

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

Another test

Can `t0 == t1 == 0`?

we can ignore this condition!!



Global variable:

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

Thread 0:

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

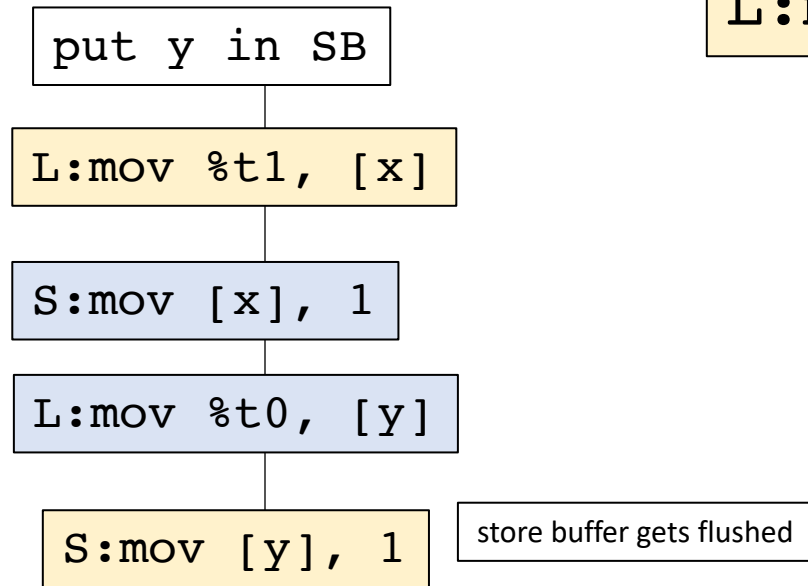
Lets peak under the hood here

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

Another test

Can `t0 == t1 == 0`?

we can ignore this condition!!



Thread 1:

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

TSO - Total Store Order

Rules:

S(tores) followed by a L(oad)

do not have to follow program order.

Global variable:

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

Thread 0:

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

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

```
mfence
```

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

Another test

Can `t0 == t1 == 0`?

Thread 1:

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

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

```
mfence
```

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

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

Global variable:

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

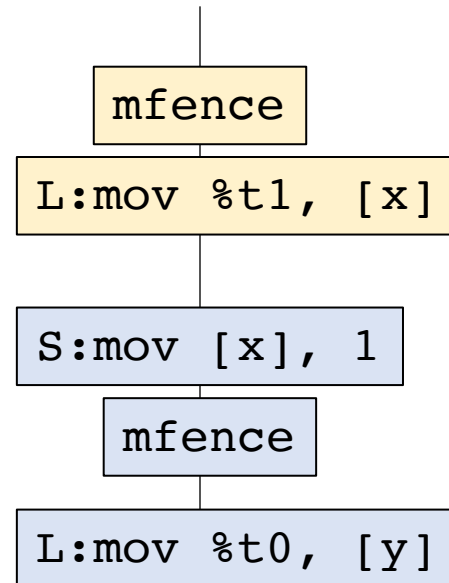
Thread 0:

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

*So we can't
reorder
this instruction
at all!*

Another test

Can `t0 == t1 == 0`?



Thread 1:

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

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

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

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

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



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

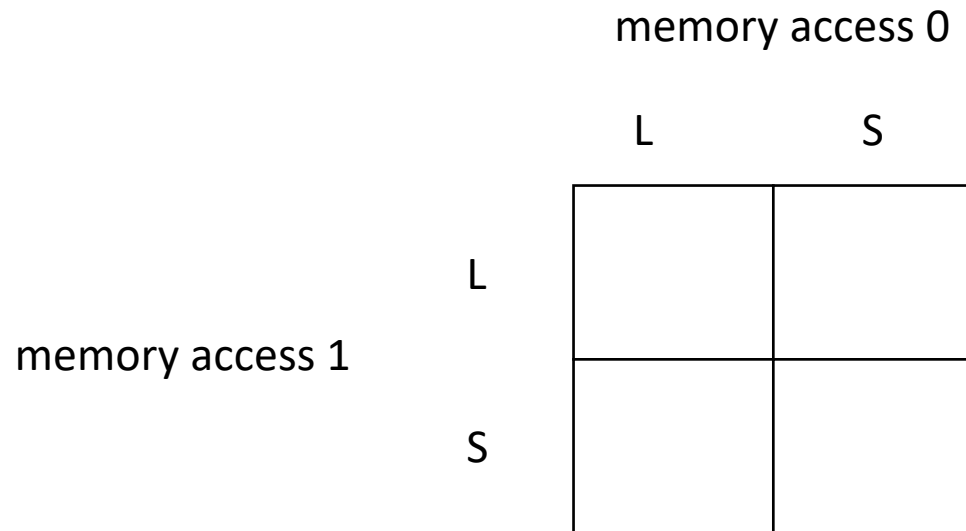
On to new material!

Schedule

- Memory consistency models
 - **More general relaxed consistency**
 - Examples
 - Compiling relaxed memory models

Other memory models?

- We can specify them in terms of what re-orderings are allowed



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

Other memory models?

- We can specify them in terms of what re-orderings 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 re-orderings 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 re-orderings 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 re-orderings 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 re-orderings 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
 - More general relaxed consistency
 - **Examples**
 - Compiling relaxed memory models

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)
```

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

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

Thread 1:

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

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

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



Global variable:

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

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

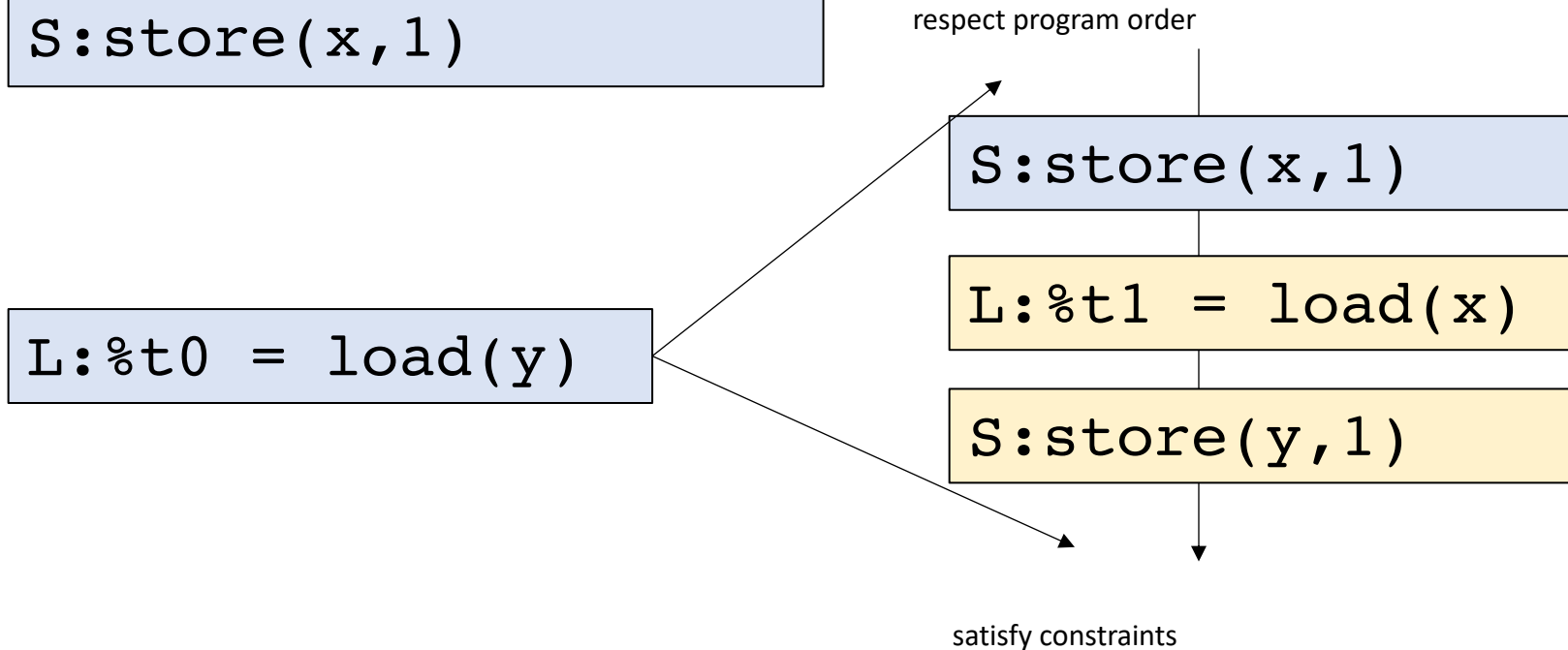
Get out our lego bricks

Thread 0:

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

Thread 1:

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



Not allowed under sequential consistency!

Global variable:

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

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

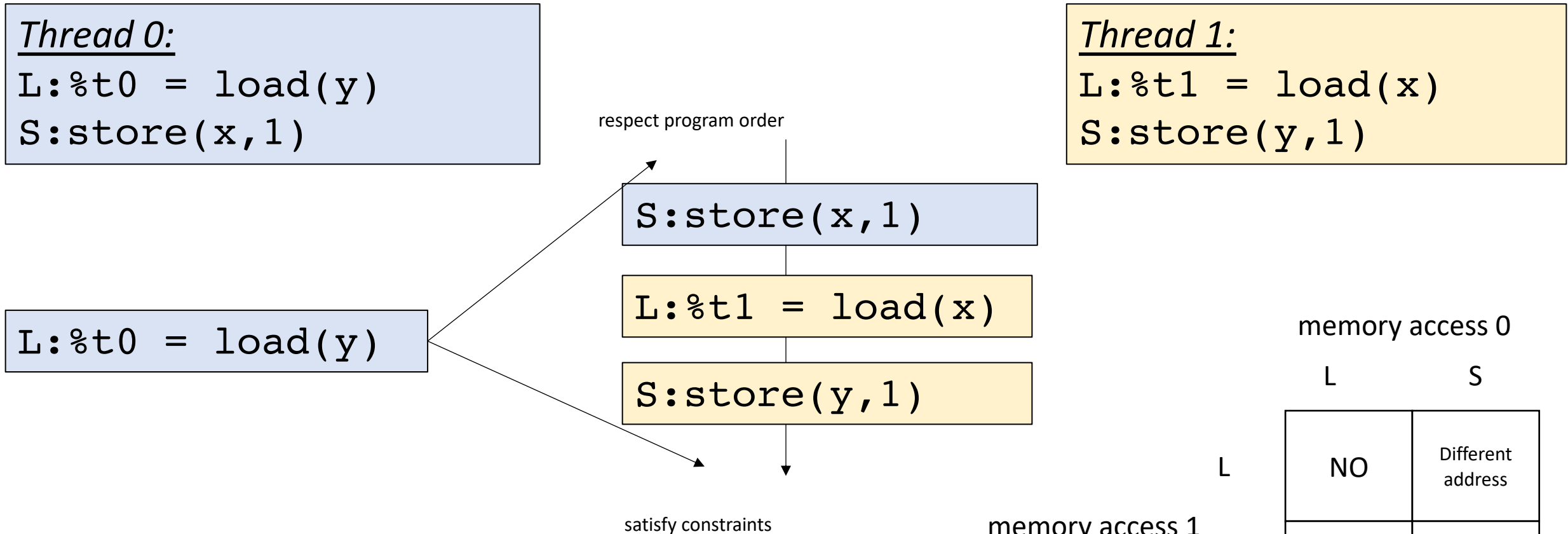
Get out our lego bricks

Thread 0:

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

Thread 1:

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



What about TSO?

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

Global variable:

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

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

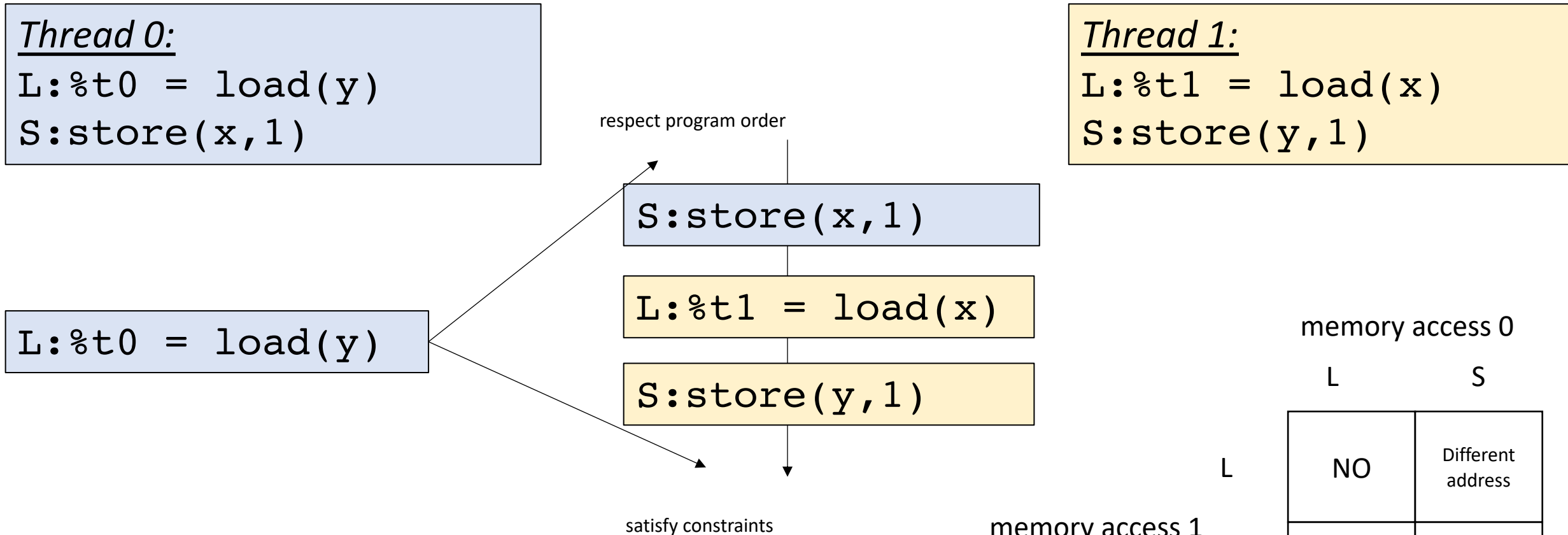
Get out our lego bricks

Thread 0:

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

Thread 1:

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



What about TSO? NOT ALLOWED!

memory access 0	
L	S
L	NO
S	NO

Global variable:

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

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

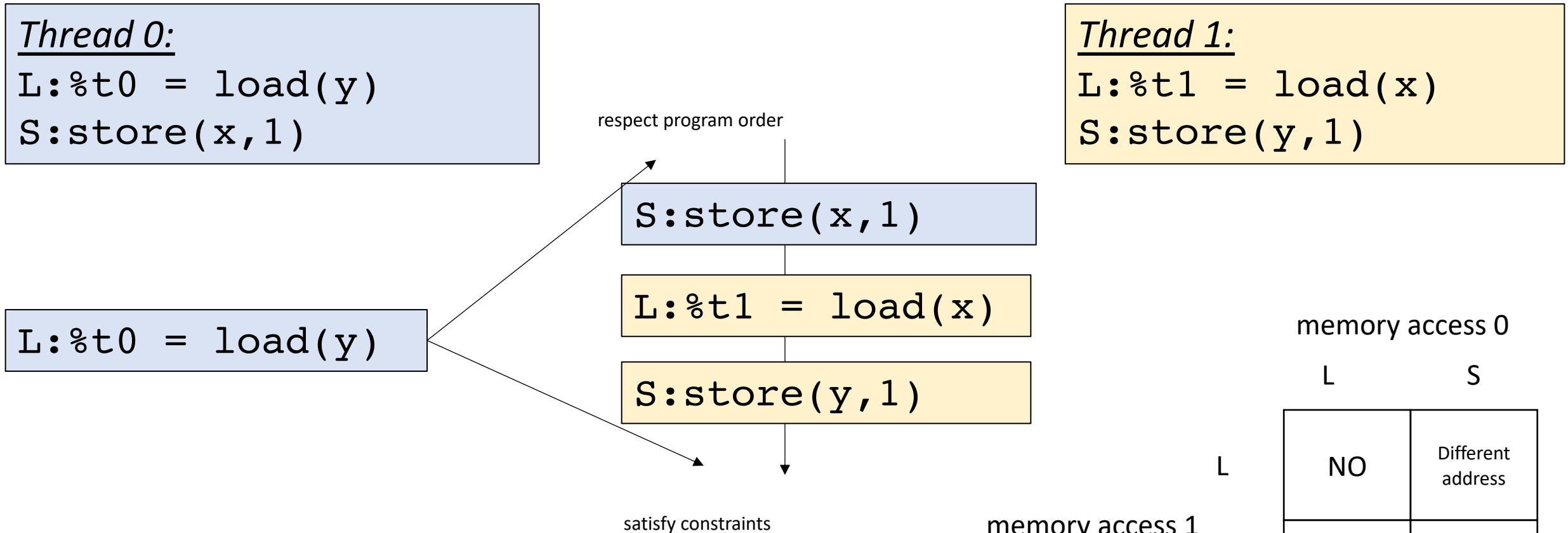
Get out our lego bricks

Thread 0:

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

Thread 1:

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



What about PSO?

memory access 0

L S

L

NO

Different
address

memory access 1

S

NO

Different
address

Global variable:

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

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

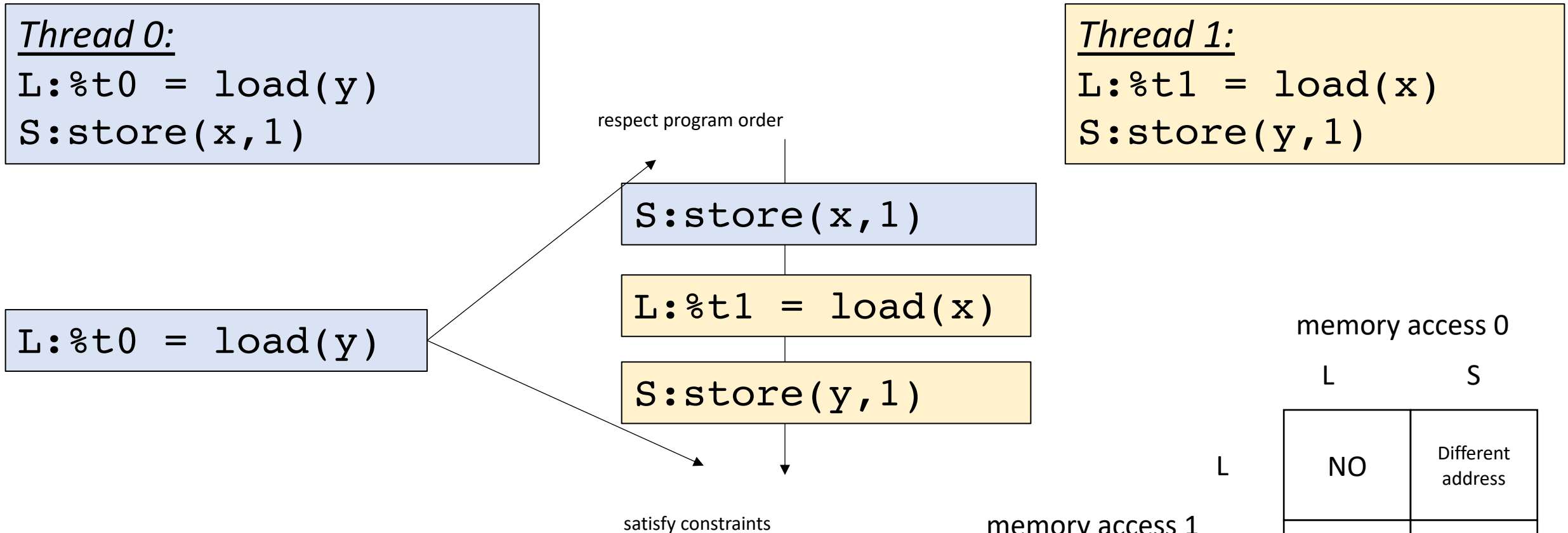
Get out our lego bricks

Thread 0:

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

Thread 1:

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



What about PSO? NO!

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

Global variable:

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

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

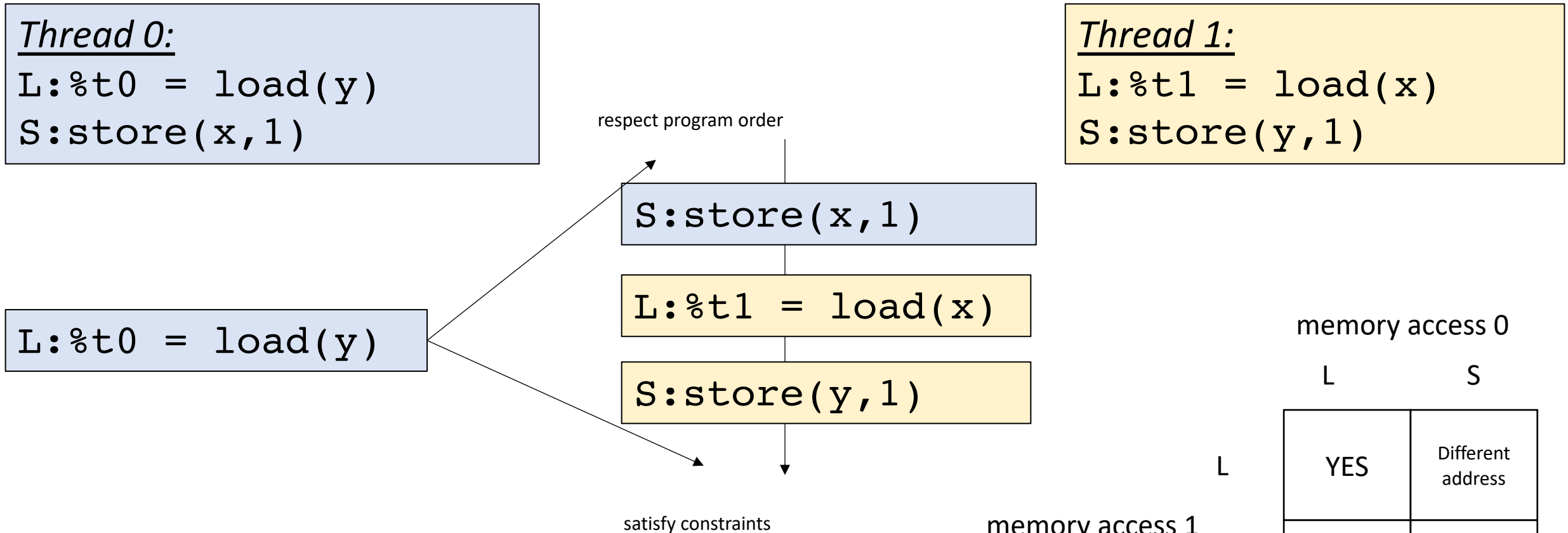
Get out our lego bricks

Thread 0:

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

Thread 1:

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



What about RMO?

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

Global variable:

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

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

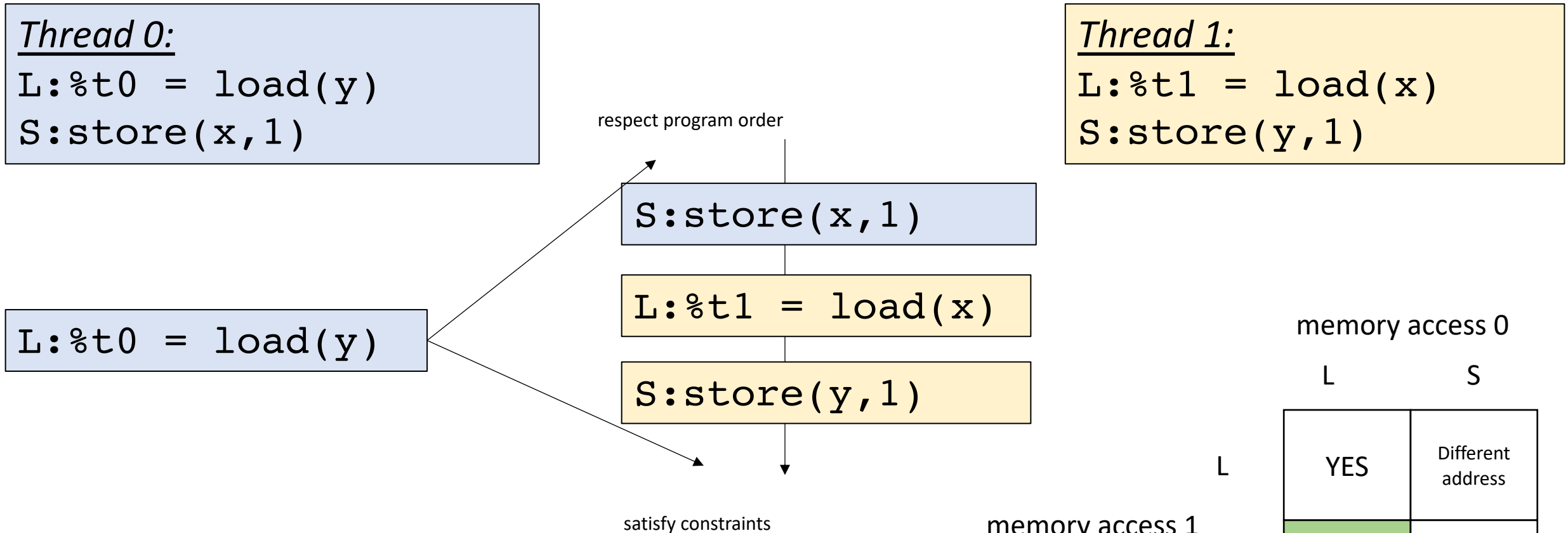
Get out our lego bricks

Thread 0:

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

Thread 1:

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



What about RMO?

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

Global variable:

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

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

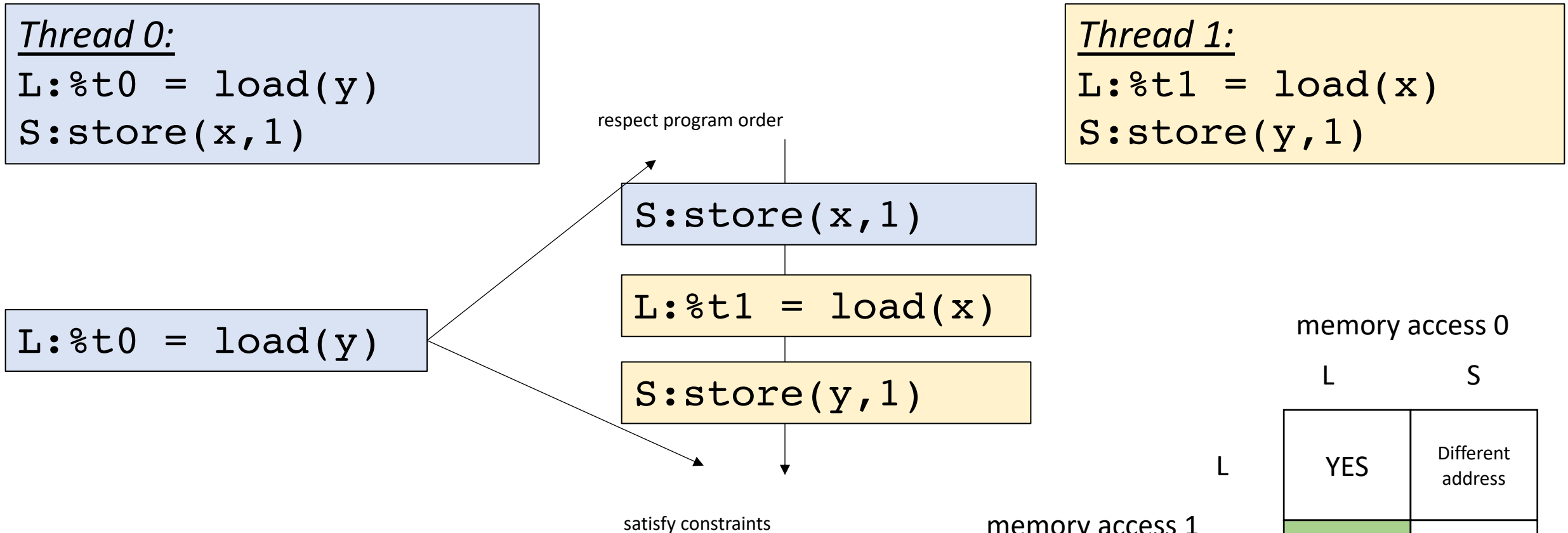
Get out our lego bricks

Thread 0:

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

Thread 1:

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



What about RMO? YES!

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

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)
```

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

respect program order

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

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

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

satisfy constraints

Thread 1:

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

memory access 0

L S

L

YES

Different
address

S

different
address

Different
address

memory access 1

How do we disallow the behavior in RMO?

Global variable:

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

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

Get out our lego bricks

Thread 0:

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

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

respect program order

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

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

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

satisfy constraints

Thread 1:

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

memory access 0

L S

L

YES

Different
address

S

different
address

Different
address

memory access 1

How do we disallow the behavior in RMO?

Global variable:

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

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

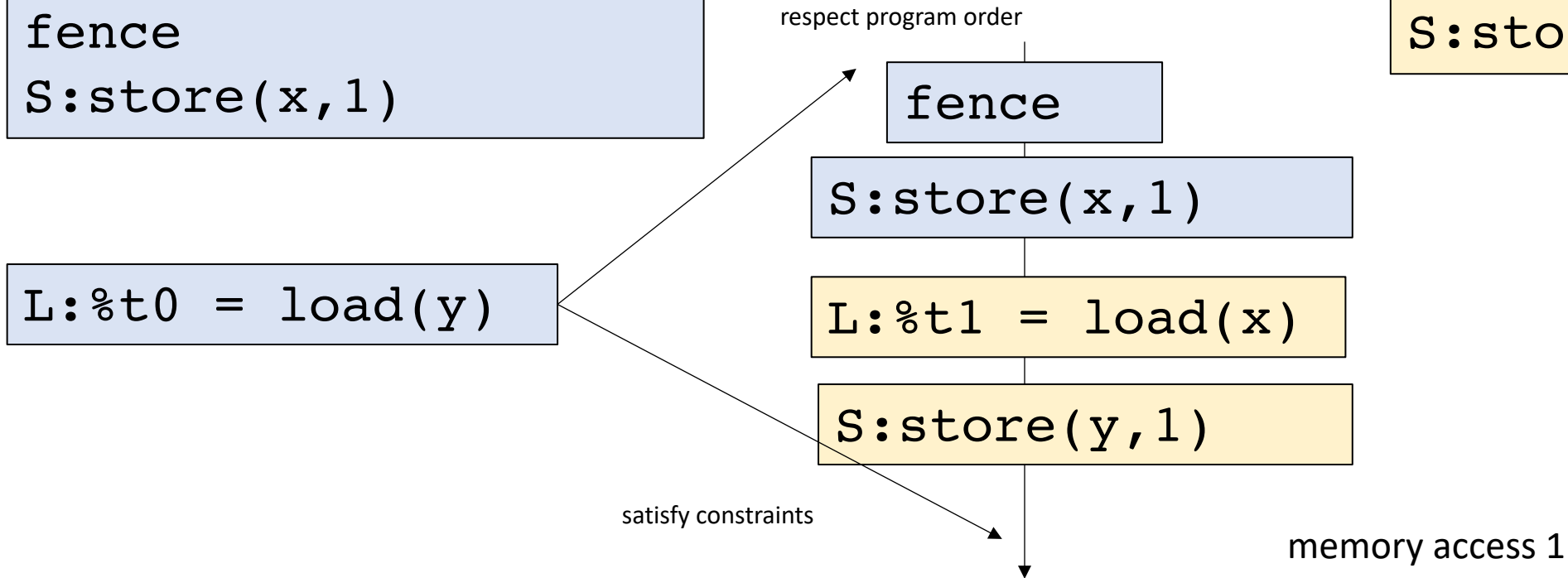
Get out our lego bricks

Thread 0:

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

Thread 1:

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



How do we disallow the behavior in RMO?

	L	S
L	YES	Different address
S	different address	Different address

Global variable:

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

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

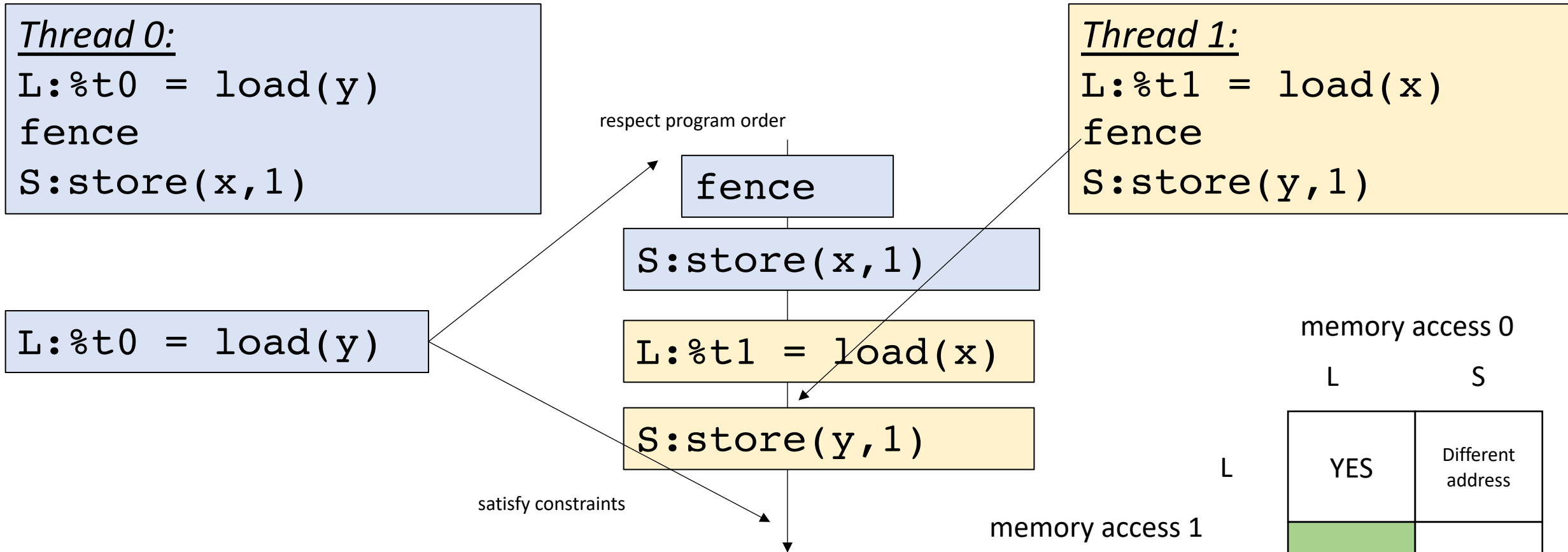
Get out our lego bricks

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

L

YES

Different
address

S

different
address

Different
address

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

Global variable:

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

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

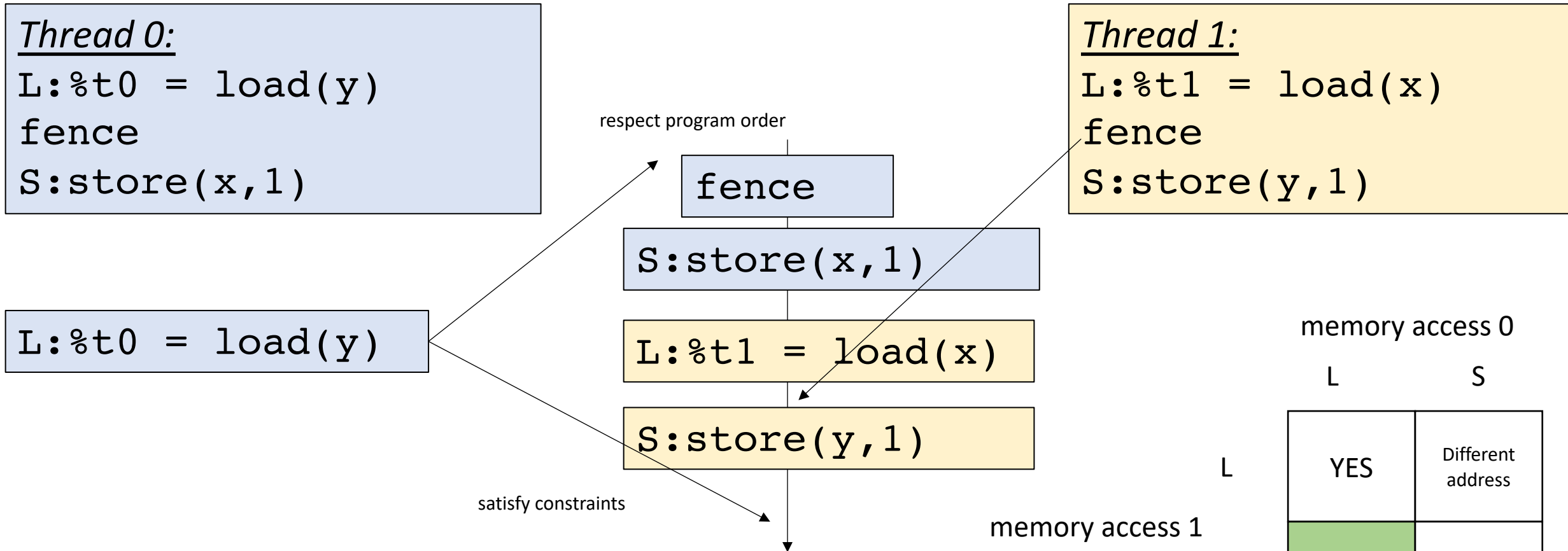
Get out our lego bricks

Thread 0:

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

Thread 1:

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



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

	L	S
L	YES	Different address
S	different address	Different address

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)
```

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

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

Thread 1:

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

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

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



Global variable:

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

Thread 0:


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

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

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

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

start off thinking
about sequential
consistency



Thread 1:

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

```
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)
```

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

respect program order

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

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

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

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

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)
```

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

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

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

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

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

satisfy constraints

respect program order

memory access 1

memory access 0

L

S

L	S
NO	Different address
NO	NO

L

S

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)
```

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

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

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

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

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

satisfy constraints

respect program order

memory access 1

memory access 0

L

S

L	S
NO	Different address
NO	NO

L

S

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)
```

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

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

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

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

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

respect program order

satisfy constraints

memory access 1

memory access 0

L

S

L	S
NO	Different address
NO	Different address

L

S

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)
```

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

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

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

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

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

respect program order

satisfy constraints

memory access 1

memory access 0

L

S

	L	S
L	NO	Different address
S	NO	Different address

What about PSO?

Global variable:

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

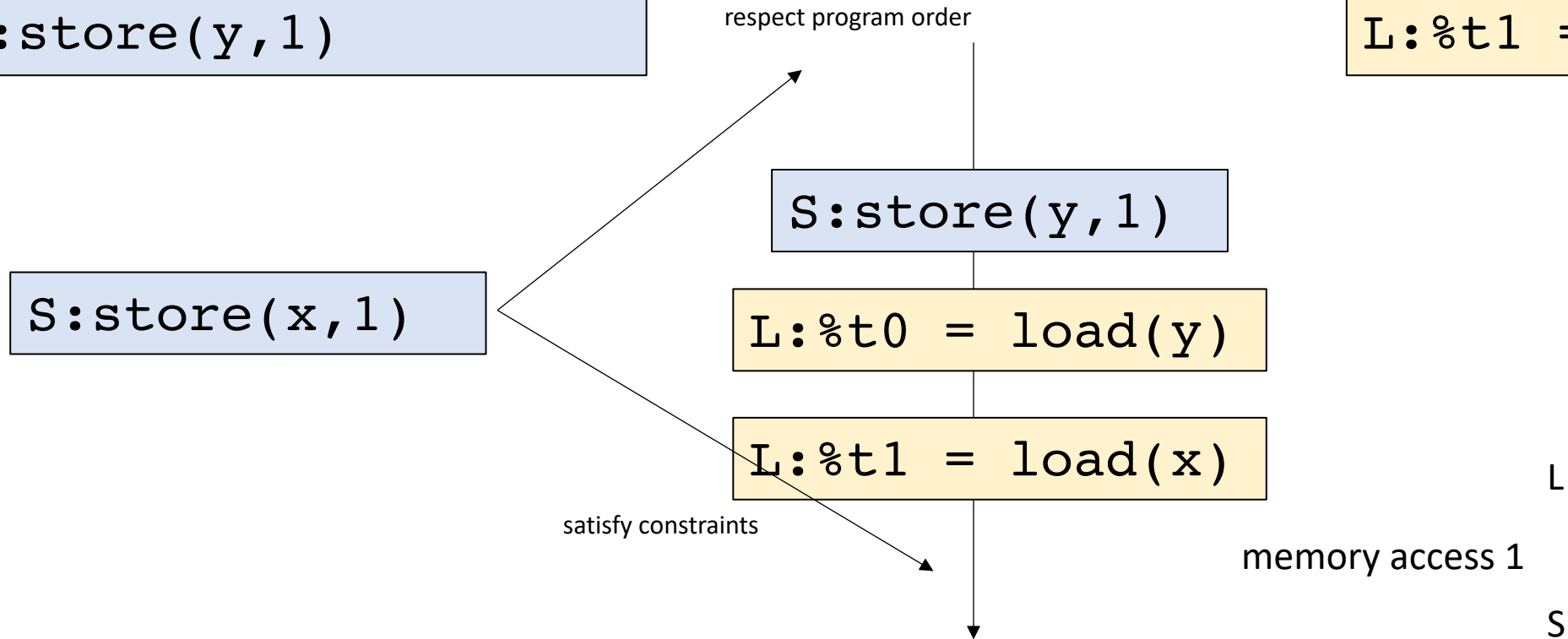
Question: can `t0 == 1` and `t1 == 0`?

Thread 0:

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

Thread 1:

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



memory access 0

L S

L

NO

Different
address

S

NO

Different
address

What about PSO? YES

Global variable:

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

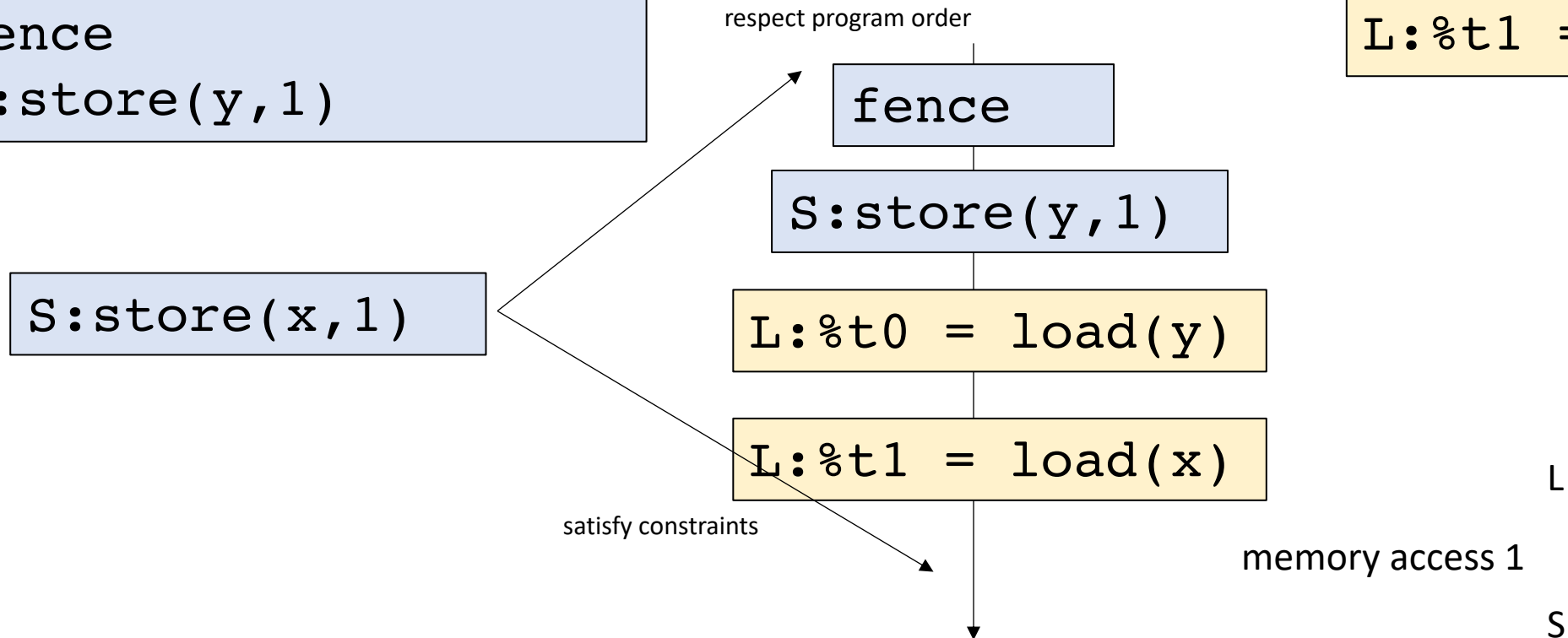
Question: can `t0 == 1` and `t1 == 0`?

Thread 0:

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

Thread 1:

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



memory access 0

L S

L

S

	L	S
L	NO	Different address
S	NO	Different address

Now it is disallowed in PSO

Global variable:

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

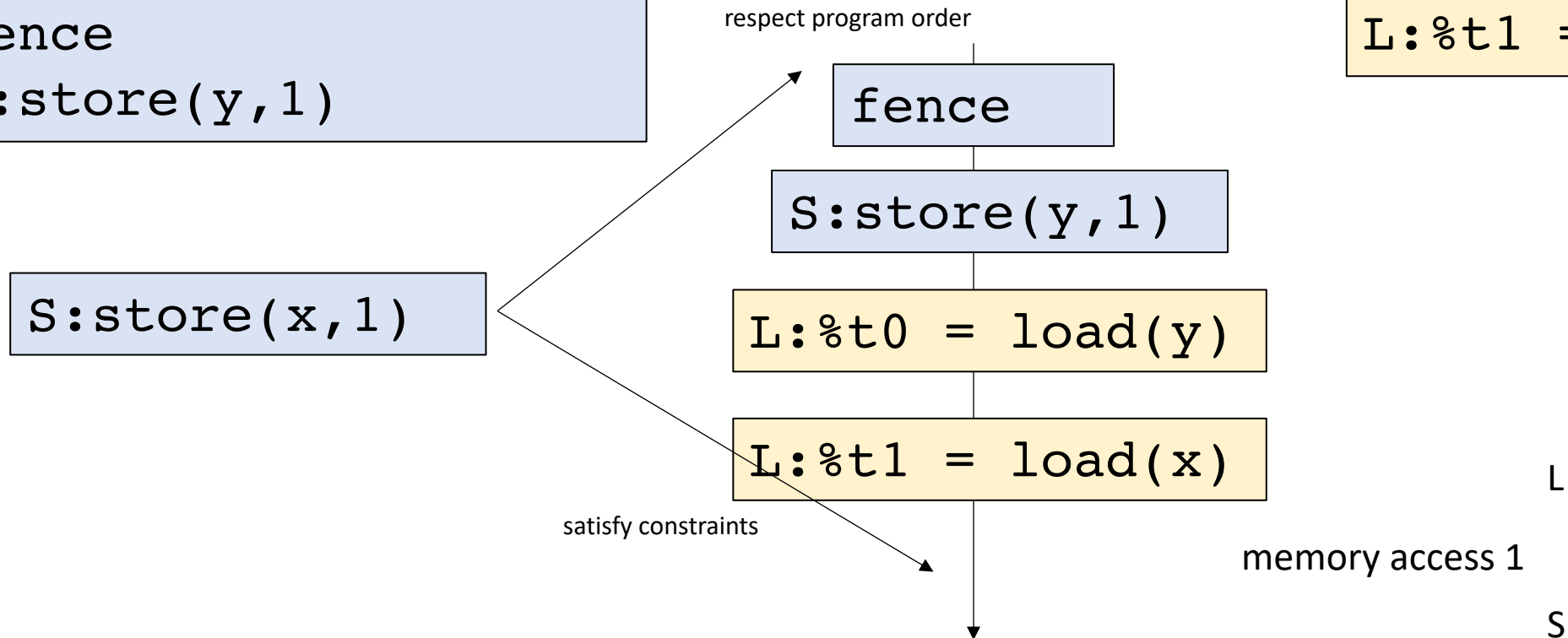
Question: can `t0 == 1` and `t1 == 0`?

Thread 0:

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

Thread 1:

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



memory access 0

L S

L

S

	L	S
L	YES	Different address
S	Different address	Different address

What about RMO?

Global variable:

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

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

Thread 0:

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

S:store(x,1)

fence

S:store(y,1)

L:%t0 = load(y)

L:%t1 = load(x)

Thread 1:

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

memory access 0

L

S

L	S
YES	Different address
Different address	Different address

L

memory access 1

S

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

memory access 0

L S

	L	S
L	YES	Different address
S	Different address	Different address

memory access 1

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`?

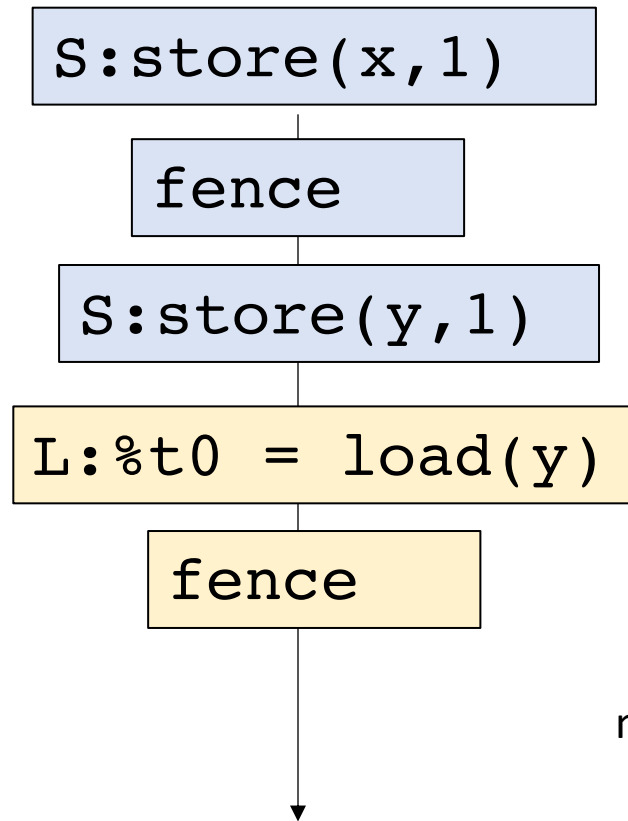
Thread 0:

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

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

Thread 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

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)
```

S:store(x,1)

fence

S:store(y,1)

L:%t0 = load(y)

fence

L:%t1 = load(x)

memory access 1

Thread 1:

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

memory access 0

L S

L

S

	L	S
L	YES	Different address
S	Different address	Different address

Now the relaxed behavior is disallowed

Memory consistency in the real world

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

Memory consistency in the real world

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

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

Getting better these days

Memory consistency in the real world

- Modern Chips:
 - RISC-V : two specs: one similar to TSO, one similar to RMO
 - Apple M1: toggles between TSO and weaker (ARM)
- Vulkan does not provide any fences that provide S - L ordering

Memory consistency in the real world

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

More about GPU memory models

- GPU Harbor
 - created by our teaching staff! Reese Levine and Tim Guo!

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]
```

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

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

Thread 1:

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

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

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



Show this example on GPU Harbor

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)
```

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

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

Thread 1:

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

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

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



Show this example on GPU Harbor

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 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

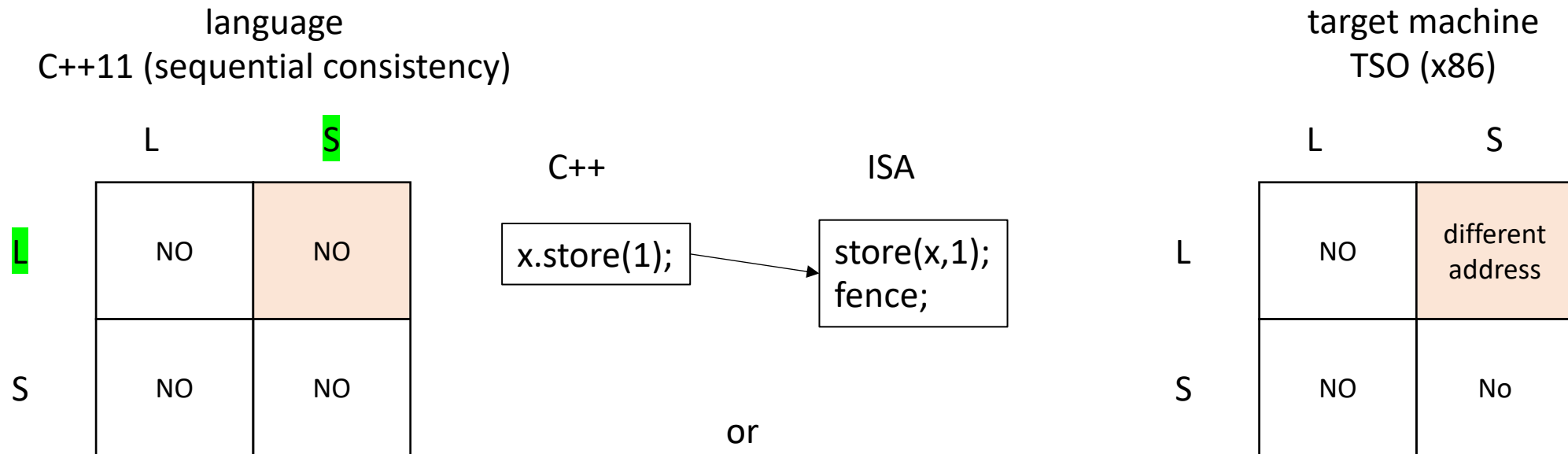
make sure loads
are not reordered
with earlier stores

target machine
TSO (x86)

	L	S
L	NO	different address
S	NO	No

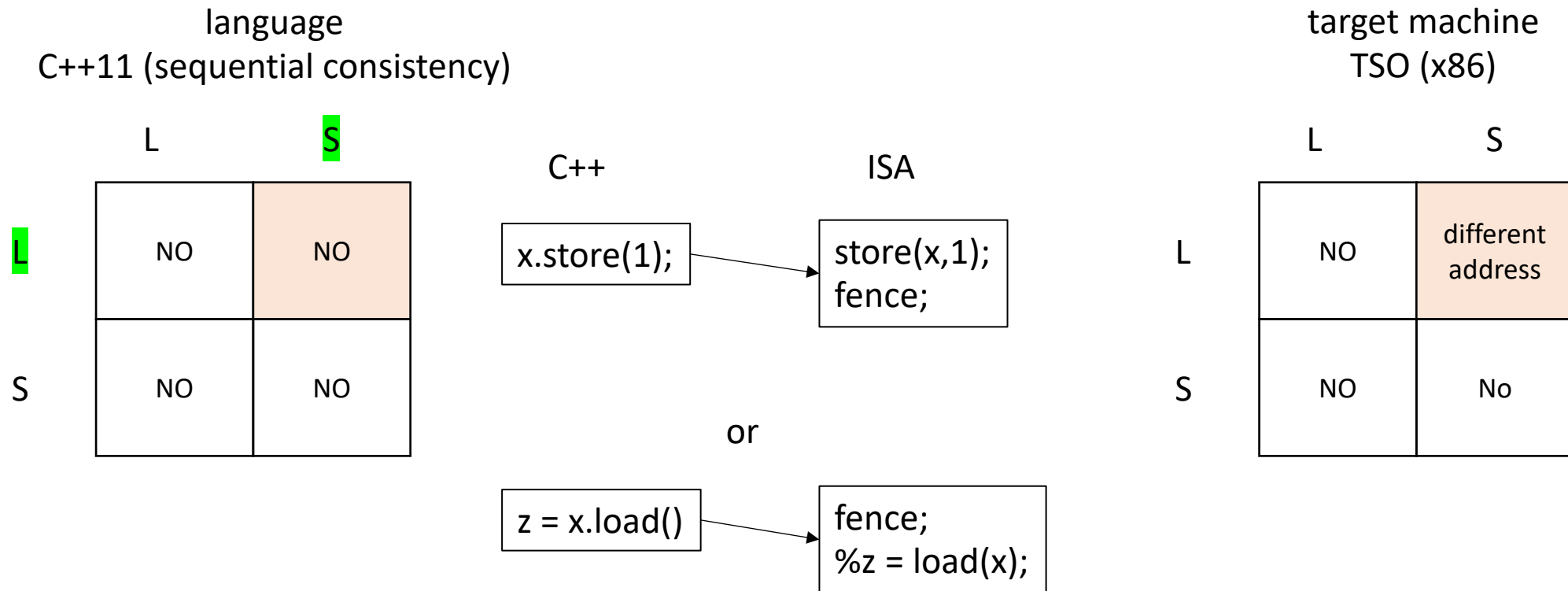
C++11 atomic operation compilation

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



C++11 atomic operation compilation

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



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

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

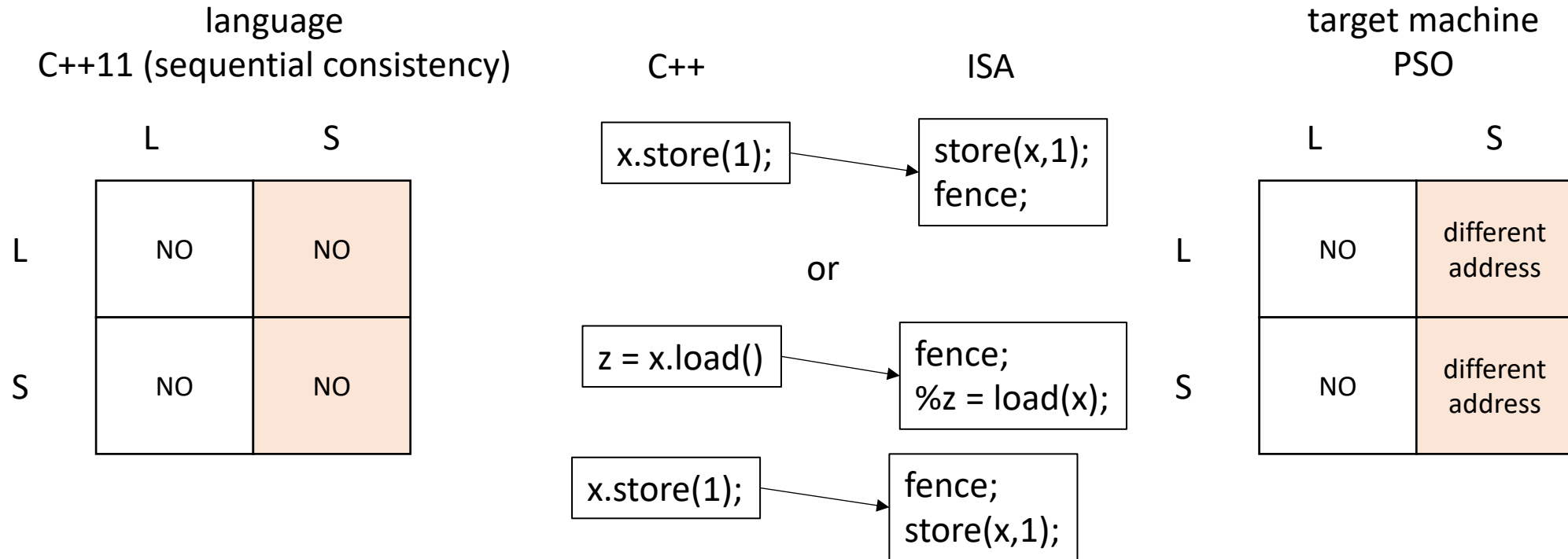
start with both both of the grids for the two different memory models

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



Memory orders

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

Where have we seen `memory_order_relaxed`?

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);
```

now yellow gets a change to go

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:

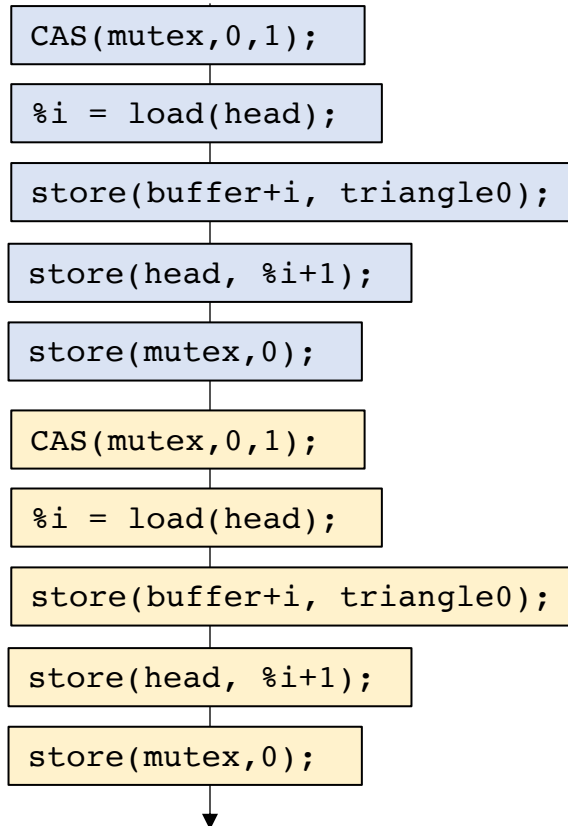
```
SPIN:CAS(mutex,0,1);  
%i = load(head);  
store(buffer+i, triangle0);  
store(head, %i+1);  
store(mutex,0);
```

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



Thread 0:

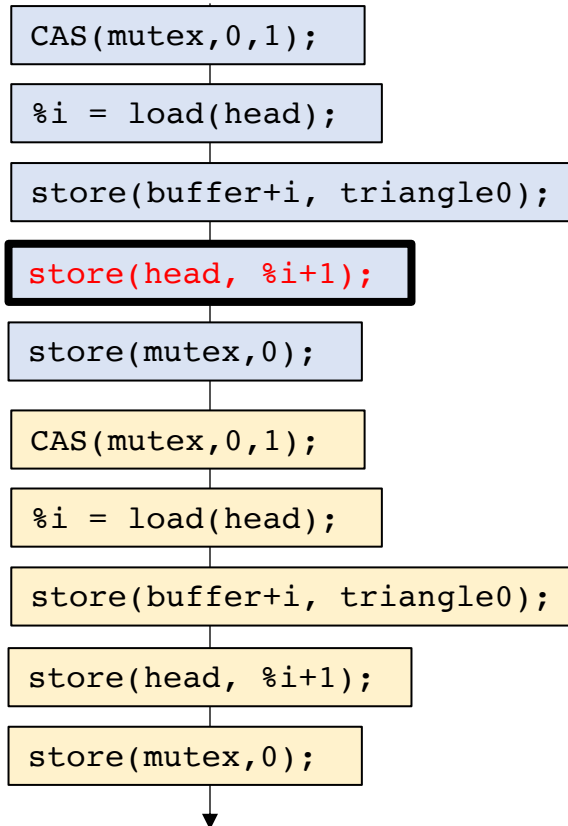
```
SPIN:CAS(mutex,0,1);  
%i = load(head);  
store(buffer+i, triangle0);  
store(head, %i+1);  
store(mutex,0);
```

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



Thread 0:

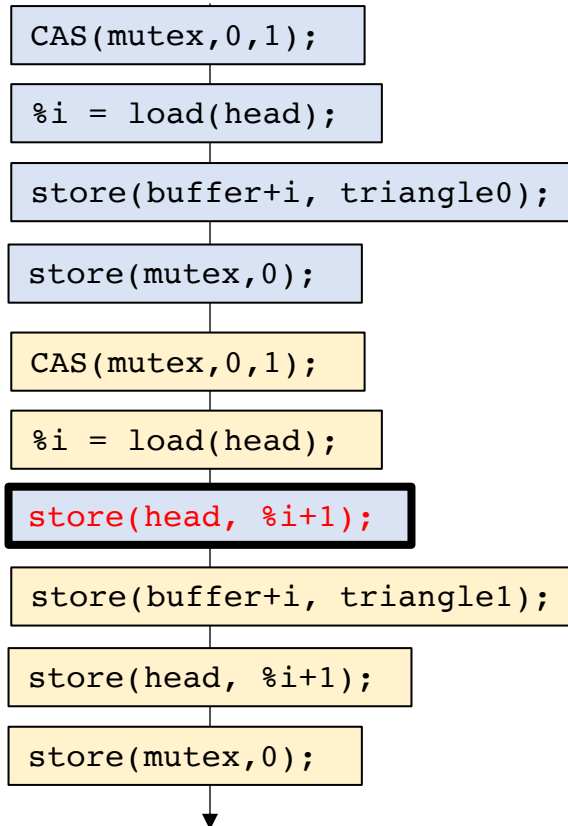
```
SPIN:CAS(mutex,0,1);  
%i = load(head);  
store(buffer+i, triangle0);  
store(head, %i+1);  
store(mutex,0);
```

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

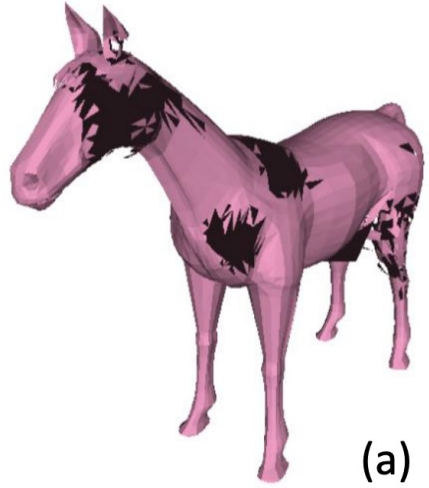


What just happened if this store moves?

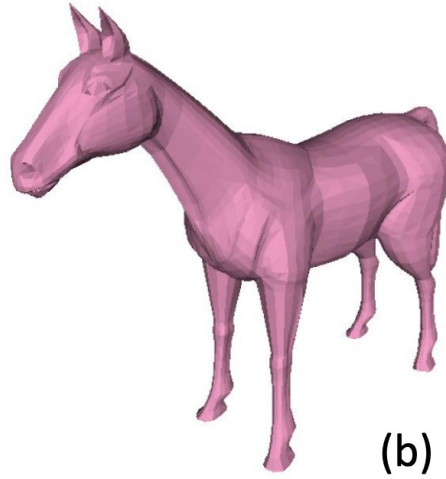
Nvidia in 2015

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

Nvidia in 2015



(a)



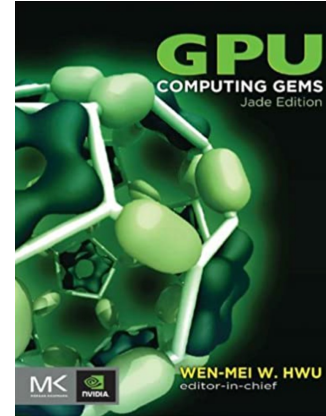
(b)



(c)



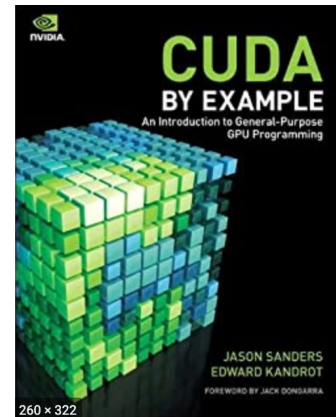
(d)



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:

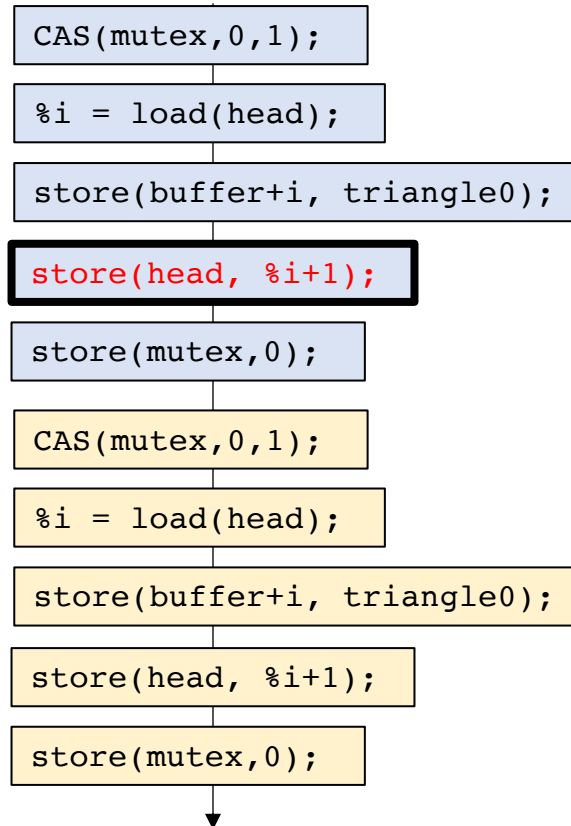
```
SPIN:CAS(mutex,0,1);  
%i = load(head);  
store(buffer+i, triangle0);  
store(head, %i+1);  
store(mutex,0);
```

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



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!*

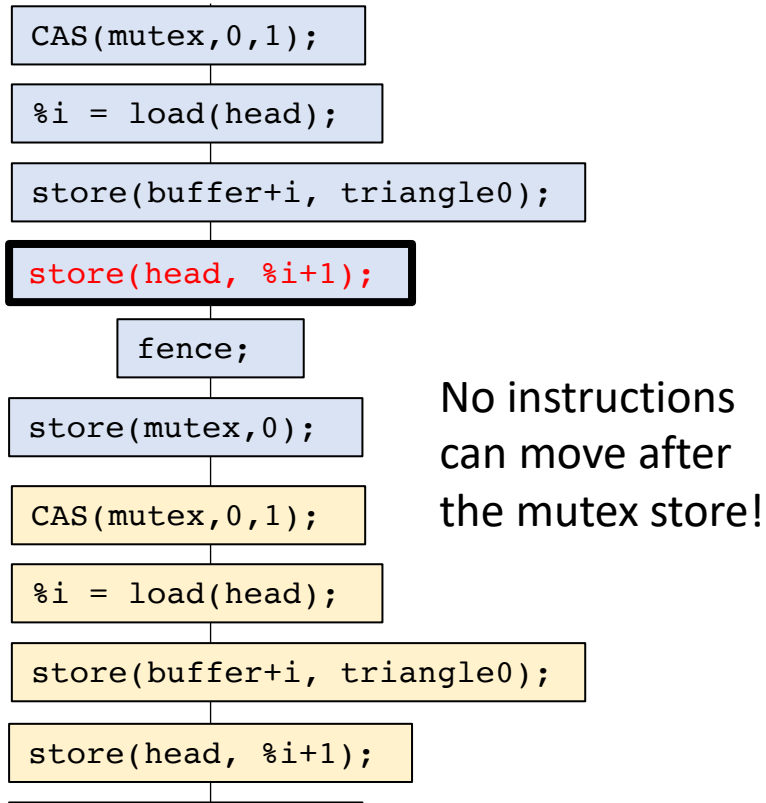
Thread 1:

```
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



How to fix the issue?

your unlock function
should contain a fence!

No instructions
can move after
the mutex store!

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:
 - Why?

	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

See you on Friday!

- Get started on HW 4
- Let us know if there are any issues with HW 2 grades
- Finishing up module 4 on Friday: forward progress models
 - Then on to GPUs!