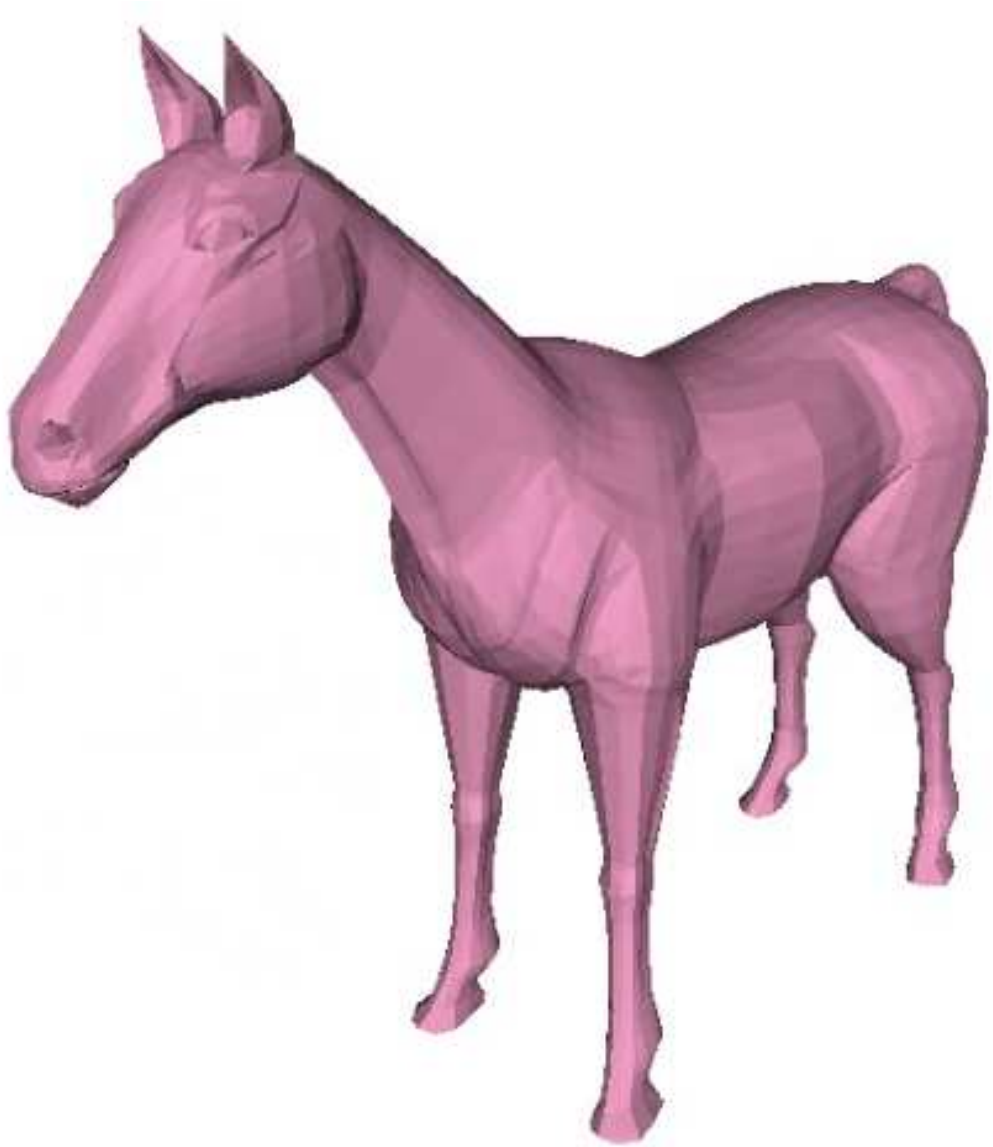
Herding cats I

Jade Alglave

Arm and University College London

March 2019

Happy pony



Sad pony

Sequential Consistency

Sequential Consistency (SC), as defined by Leslie Lamport in 1979, is a model for concurrent programming:

> *The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.*

Example

Consider the following example, where initially $x = y = 0$:

sb

P_0	P_1
$(a)\ \mathtt{x} \leftarrow 1$	$(c)\ \mathtt{y} \leftarrow 1$
$(b)\ \mathtt{r1} \leftarrow \mathtt{y}$	$(d)\ \mathtt{r2} \leftarrow \mathtt{x}$
`r1=?; r2=?;`	

Following SC, we expect three possible outcomes:

$(a)(b)(c)(d)$	$\mathsf{r1} = 0 \wedge \mathsf{r2} = 1$
$(c)(d)(a)(b)$	$\mathsf{r1} = 1 \wedge \mathsf{r2} = 0$
$(a)(c)(b)(d)$	$\mathsf{r1} = 1 \wedge \mathsf{r2} = 1$
$(a)(c)(d)(b)$	
$(c)(a)(b)(d)$	
$(c)(a)(d)(b)$	

Note that `r1=0; r2=0` is impossible on SC

Experiment

On Intel:

```
{x=0; y=0;}

 P0            | P1            ;
 MOV [y],$1    | MOV [x],$1    ;
 MOV EAX,[x]   | MOV EAX,[y]   ;

exists (0:EAX=0 /\ 1:EAX=0)
```

Instructions appear to be reordered w.r.t. the program order.
Let us check that on my machine.

Weak memory models

For performance reasons, modern architectures provide several features that are weakenings of SC:

> *For some applications, achieving sequential consistency may not be worth the price of slowing down the processors. In this case, one must be aware that conventional methods for designing multiprocess algorithms cannot be relied upon to produce correctly executing programs.*

How can we make sure that we write correct programs?

- We need to understand precisely what the memory models guarantee in order to write correct concurrent programs.
- This problem spreads to high level languages and is potentially much worse there, due to compiler optimisations.

Describing executions

Style of modelling

Memory models roughly fall into two classes:

- Operational
- Axiomatic

Building an execution

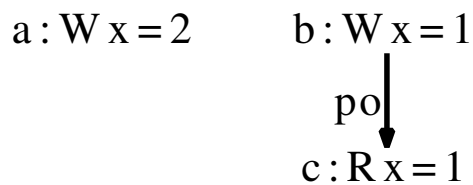
P_0	P_1
$(a)\,\mathtt{x} \leftarrow 2$	$(b)\,\mathtt{x} \leftarrow 1$
	$(c)\,\mathtt{r1} \leftarrow \mathtt{x}$

$\mathtt{1:r1=1;\ x=2;}$

Building an execution: Events $\mathbb{E}$ and program order po

P_0	P_1
$(a)\ \mathtt{x} \leftarrow 2$	$(b)\ \mathtt{x} \leftarrow 1$
	$(c)\ \mathtt{r1} \leftarrow \mathtt{x}$

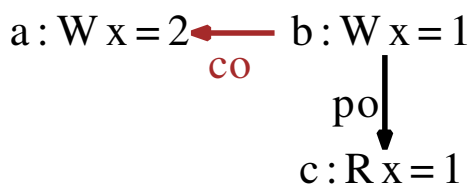
1:r1=1; x=2;



Building an execution: Coherence co

P_0	P_1
$(a)\ \mathtt{x} \leftarrow 2$	$(b)\ \mathtt{x} \leftarrow 1$
	$(c)\ \mathtt{r1} \leftarrow \mathtt{x}$

1:r1=1; x=2;

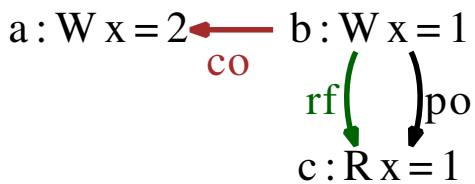


Coherence co totally orders all the write events to the same memory location.

Building an execution: Read-from rf

P_0	P_1
$(a)\ \mathtt{x} \leftarrow 2$	$(b)\ \mathtt{x} \leftarrow 1$
	$(c)\ \mathtt{r1} \leftarrow \mathtt{x}$

1:r1=1; x=2;

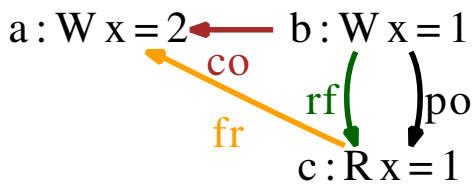


The read-from map rf links a write to reads reading from it.

Building an execution: From-read map fr

P_0	P_1
(a) x $\leftarrow$ 2	(b) x $\leftarrow$ 1
	(c) r1 $\leftarrow$ x

1:r1=1; x=2;



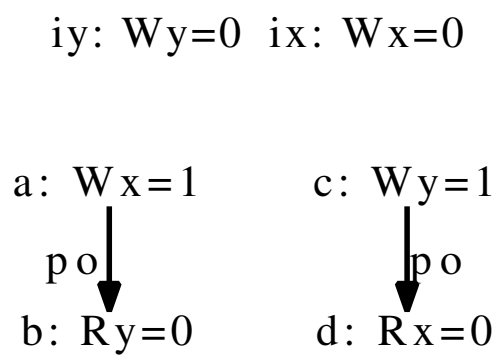
We derive the from-read map fr from co and rf.

Let's build the non-SC execution of **sb**

sb

P_0	P_1
(a) x $\leftarrow$ 1	(c) y $\leftarrow$ 1
(b) r1 $\leftarrow$ y	(d) r2 $\leftarrow$ x

r1=0; r2=0;

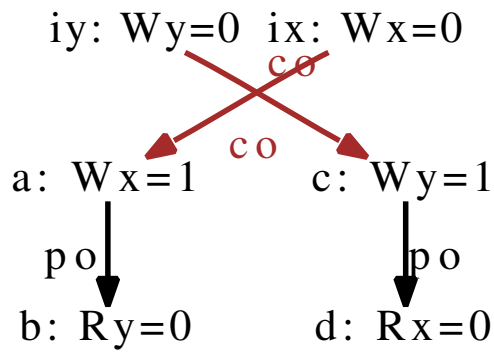


Let's build the non-SC execution of **sb**

sb

P_0	P_1
(a) x $\leftarrow$ 1	(c) y $\leftarrow$ 1
(b) r1 $\leftarrow$ y	(d) r2 $\leftarrow$ x

r1=0; r2=0;

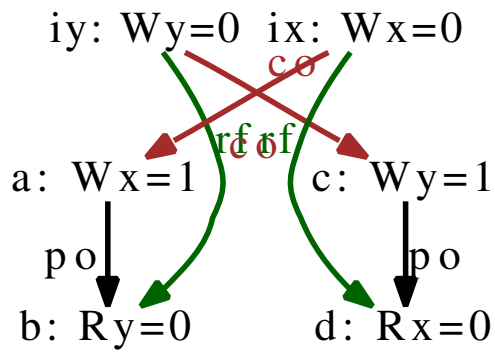


Let's build the non-SC execution of **sb**

sb

P_0	P_1
(a) x $\leftarrow$ 1	(c) y $\leftarrow$ 1
(b) r1 $\leftarrow$ y	(d) r2 $\leftarrow$ x

r1=0; r2=0;

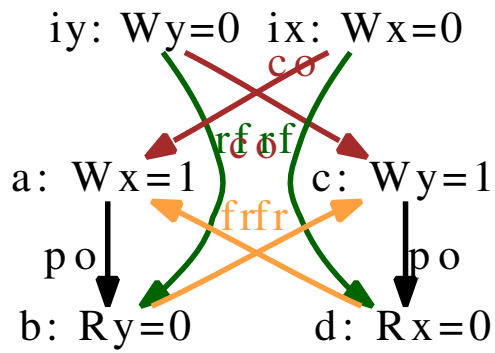


Let's build the non-SC execution of **sb**

sb

P_0	P_1
(a) x $\leftarrow$ 1	(c) y $\leftarrow$ 1
(b) r1 $\leftarrow$ y	(d) r2 $\leftarrow$ x

r1=0; r2=0;

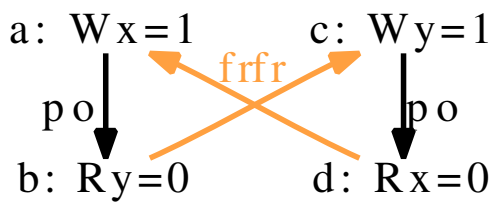


Let's build the non-SC execution of **sb**

sb

P_0	P_1
(a) x $\leftarrow$ 1	(c) y $\leftarrow$ 1
(b) r1 $\leftarrow$ y	(d) r2 $\leftarrow$ x

r1=0; r2=0;

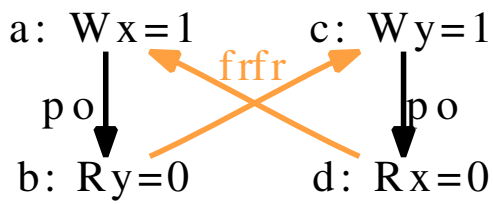


Let's build the non-SC execution of **sb**

sb

P_0	P_1
(a) x $\leftarrow$ 1	(c) y $\leftarrow$ 1
(b) r1 $\leftarrow$ y	(d) r2 $\leftarrow$ x

r1=0; r2=0;



SC corresponds to acyclic(po $\cup$ com)

SC in cat

```
SC

include "cos.cat" (*build co and fr*)

let com = rf | co | fr (*communications*)
acyclic po | com as sc
```

See also: `herdtools7/herd/libdir/sc.cat`

That's it for today!