Concurrency models and Modern Processors

Introduction

The classical model of concurrency is the interleaving model. It corresponds to a memory model called *Sequential Consistency* (SC).

Modern processors dot not implement SC, but so-called "*relaxed memory models*", such as *Total Store Order* (TSO).

Memory models

Sequential Consistency All memory accesses are immediately visible to all processors.

Total Store Order (TSO)

- All write operations go through a buffer.
- Each processor reads the most recent value in its buffer, if there is such a value ;
- If not, the value held in memory is read.

Memory models (continued)

x86-TSO

- Intended for the programmer
- Based on TSO
- Extended with the concept of a global lock
- Compatible with the tests found in processors' documentation
- Does not aim at closely modeling the internal structure of processors

Memory models (continued 2)

Examples that can be fully executed under TSO (and X86-TSO), but not under SC

initially : x = y = 0	
Processor 1	Processor 2
$store(p_1, \mathrm{x}, 1)$ (s_1)	$store(p_2, \mathrm{y}, 1)$ (s_2)
$load(p_1, \mathrm{y}, 0)$ (l_1)	$load(p_2, \mathrm{x}, 0)$ (l_2)

$store(p, m, v)$
$p : [m] \leftarrow v$

$load(p, m, v)$
$p : ([m] == v)$

Possible interleavings :

SC		TSO		
s_1	s_1	s_1	s_2	s_1
l_1	s_2	s_2	l_2	l_1
s_2	l_1 blocks	l_1	s_1	s_2
l_2 blocks		l_2	l_1	l_2 blocks

Memory models (continued 3)

Examples that can be fully executed under TSO (and X86-TSO), but not under SC

initially : x = y = 0	
Processor 1	Processor 2
$store(p_1, \mathrm{x}, 1)$ (s_1)	$store(p_2, \mathrm{y}, 1)$ (s_2)
$load(p_1, \mathrm{x}, 1)$ (l_1)	$load(p_2, \mathrm{y}, 1)$ (l_3)
$load(p_1, \mathrm{y}, 0)$ (l_2)	$load(p_2, \mathrm{x}, 0)$ (l_4)

Possible interleavings :

SC		TSO		
s_1	s_1	s_1	s_2	s_1
l_1	s_2	s_2	l_3	l_1
l_2	l_1	l_1	s_1	l_2
s_2	l_3	l_2	l_1	s_2
l_3	l_2 blocks	l_3	l_2	l_3
l_4 blocks		l_4	l_4	l_4 blocks

SC

Formal definition

Order relations :

- program order ($<_p$) (partial, per processor order)
- memory order ($<_m$) (global)

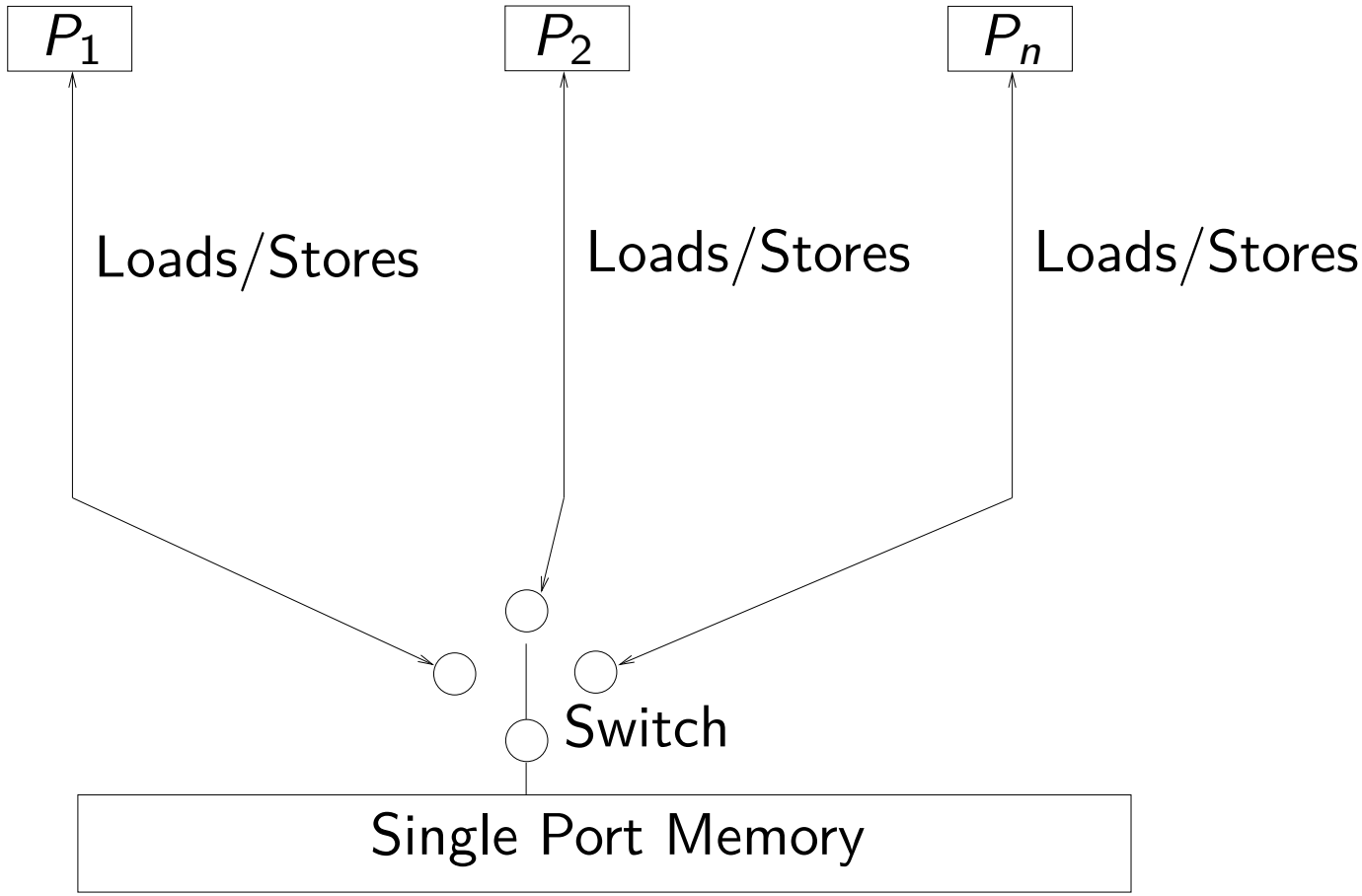
To correspond to the SC model, an execution must satisfy the following conditions :

- $\forall op_i, op_j : op_i <_p op_j \Rightarrow op_i <_m op_j$

where op_x représents a *store* or *load* operation. The result of a *load* is the one compatible with the memory order.

SC (continued)

Operational definition



SC (continued 2)

Memory orders, first example

initially : $\mathtt{x} = \mathtt{y} = 0$	
Processor 1	Processor 2
$store(p_1, \mathtt{x}, 1)$ (s_1)	$store(p_2, \mathtt{y}, 1)$ (s_2)
$load(p_1, \mathtt{y}, 0)$ (l_1)	$load(p_2, \mathtt{x}, 0)$ (l_2)

where $s_1 <_p l_1$ et $s_2 <_p l_2$.

The possible memory orders are all interleavings satisfying the conditions $s_1 <_m l_1$ et $s_2 <_m l_2$.

Example : $s_1 <_m l_1 <_m s_2 <_m l_2$

TSO

Formal definition

To correspond to the TSO model, an execution must satisfy the following conditions :

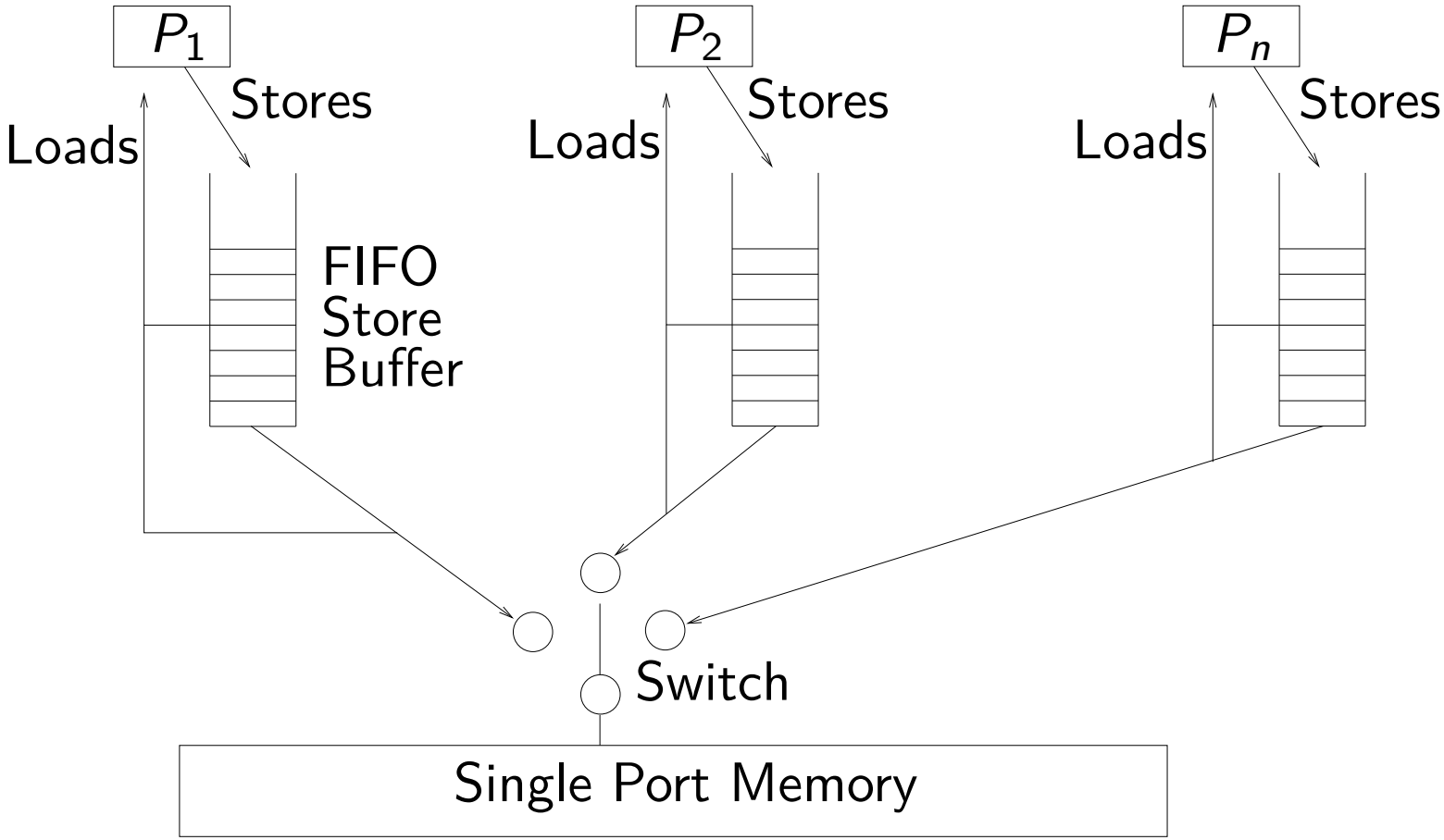
1. $\forall l_a, l_b : l_a <_p l_b \Rightarrow l_a <_m l_b$
2. $\forall s_a, s_b : s_a <_p s_b \Rightarrow s_a <_m s_b$
3. $\forall l, s : l <_p s \Rightarrow l <_m s$
4. $val(l_a) = val(\max_{<_m}\{s_a \mid s_a <_m l_a \vee s_a <_p l_a\})$.

Note that

- *stores* can be delayed,
- but *loads* have access to the latest locally written value.

TSO (continued)

Operational definition



Transferring a *store* from a buffer to main memory is called a *commit*.

TSO (continued 2)

Memory orders, first example

initially : $\mathtt{x} = \mathtt{y} = 0$	
Processor 1	Processor 2
$store(p_1, \mathtt{x}, 1)$ (s_1)	$store(p_2, \mathtt{y}, 1)$ (s_2)
$load(p_1, \mathtt{y}, 0)$ (l_1)	$load(p_2, \mathtt{x}, 0)$ (l_2)

where $s_1 <_p l_1$ et $s_2 <_p l_2$.

The compatible memory orders, are the interleavings of s_1, l_1, s_2 et l_2.

Example : $l_1 <_m s_2 <_m l_2 <_m s_1$ or $l_1 <_m l_2 <_m s_1 <_m s_2$

TSO (continued 3)

Memory orders, second example

initially : x = y = 0	
Processor 1	Processor 2
$store(p_1, \mathrm{x}, 1)$ (s_1)	$store(p_2, \mathrm{y}, 1)$ (s_2)
$load(p_1, \mathrm{x}, 1)$ (l_1)	$load(p_2, \mathrm{y}, 1)$ (l_3)
$load(p_1, \mathrm{y}, 0)$ (l_2)	$load(p_2, \mathrm{x}, 0)$ (l_4)

where $s_1 <_p l_1$, $l_1 <_p l_2$, $s_2 <_p l_3$ et $l_3 <_p l_4$.

The compatible memory orders are all interleavings of s_1, l_1, l_2, s_2, l_3 et l_4 satisfying $l_1 <_m l_2$ et $l_3 <_m l_4$

Exemple : $l_1 <_m l_2 <_m l_3 <_m l_4 <_m s_1 <_m s_2$

TSO (continued 4)

Memory orders, second example - Details

Sequence of operations	Memory order
$store(p_1, \mathtt{x}, 1)$ (s_1)	-

TSO (continued 4)

Memory orders, second example - Details

Sequence of operations	Memory order
$store(p_1, \mathtt{x}, 1)$ (s_1)	-
$load(p_1, \mathtt{x}, 1)$ (l_1)	l_1

TSO (continued 4)

Memory orders, second example - Details

Sequence of operations	Memory order
$store(p_1, \mathtt{x}, 1)\ \ (s_1)$	-
$load(p_1, \mathtt{x}, 1)\ \ (l_1)$	l_1
$load(p_1, \mathtt{y}, 0)\ \ (l_2)$	$l_1 <_m l_2$

TSO (continued 4)

Memory orders, second example - Details

Sequence of operations	Memory order
$store(p_1, \mathrm{x}, 1)\ \ (s_1)$	-
$load(p_1, \mathrm{x}, 1)\ \ (l_1)$	l_1
$load(p_1, \mathrm{y}, 0)\ \ (l_2)$	$l_1 <_m l_2$
$store(p_2, \mathrm{y}, 1)\ \ (s_2)$	$l_1 <_m l_2$

TSO (continued 4)

Memory orders, second example - Details

Sequence of operations	Memory order
$store(p_1, \mathrm{x}, 1)\ \ (s_1)$	-
$load(p_1, \mathrm{x}, 1)\ \ (l_1)$	l_1
$load(p_1, \mathrm{y}, 0)\ \ (l_2)$	$l_1 <_m l_2$
$store(p_2, \mathrm{y}, 1)\ \ (s_2)$	$l_1 <_m l_2$
$load(p_2, \mathrm{y}, 1)\ \ (l_3)$	$l_1 <_m l_2 <_m l_3$

TSO (continued 4)

Memory orders, second example - Details

Sequence of operations	Memory order
$store(p_1, \mathrm{x}, 1)\ \ (s_1)$	-
$load(p_1, \mathrm{x}, 1)\ \ (l_1)$	l_1
$load(p_1, \mathrm{y}, 0)\ \ (l_2)$	$l_1 <_m l_2$
$store(p_2, \mathrm{y}, 1)\ \ (s_2)$	$l_1 <_m l_2$
$load(p_2, \mathrm{y}, 1)\ \ (l_3)$	$l_1 <_m l_2 <_m l_3$
$load(p_2, \mathrm{x}, 0)\ \ (l_4)$	$l_1 <_m l_2 <_m l_3 <_m l_4$

TSO (continued 4)

Memory orders, second example - Details

Sequence of operations	Memory order
$store(p_1, \mathrm{x}, 1)\ \ (s_1)$	-
$load(p_1, \mathrm{x}, 1)\ \ (l_1)$	l_1
$load(p_1, \mathrm{y}, 0)\ \ (l_2)$	$l_1 <_m l_2$
$store(p_2, \mathrm{y}, 1)\ \ (s_2)$	$l_1 <_m l_2$
$load(p_2, \mathrm{y}, 1)\ \ (l_3)$	$l_1 <_m l_2 <_m l_3$
$load(p_2, \mathrm{x}, 0)\ \ (l_4)$	$l_1 <_m l_2 <_m l_3 <_m l_4$
— $(commit(s_1))$	$l_1 <_m l_2 <_m l_3 <_m l_4 <_m s_1$

TSO (continued 4)

Memory orders, second example - Details

Sequence of operations	Memory order
$store(p_1, \mathtt{x}, 1)\ \ (s_1)$	-
$load(p_1, \mathtt{x}, 1)\ \ (l_1)$	l_1
$load(p_1, \mathtt{y}, 0)\ \ (l_2)$	$l_1 <_m l_2$
$store(p_2, \mathtt{y}, 1)\ \ (s_2)$	$l_1 <_m l_2$
$load(p_2, \mathtt{y}, 1)\ \ (l_3)$	$l_1 <_m l_2 <_m l_3$
$load(p_2, \mathtt{x}, 0)\ \ (l_4)$	$l_1 <_m l_2 <_m l_3 <_m l_4$
— $(commit(s_1))$	$l_1 <_m l_2 <_m l_3 <_m l_4 <_m s_1$
— $(commit(s_2))$	$l_1 <_m l_2 <_m l_3 <_m l_4 <_m s_1 <_m s_2$

x86-TSO

Formal definition

- The order constraints on the *loads* and *stores* are the same as those for TSO.
- Extended operations :

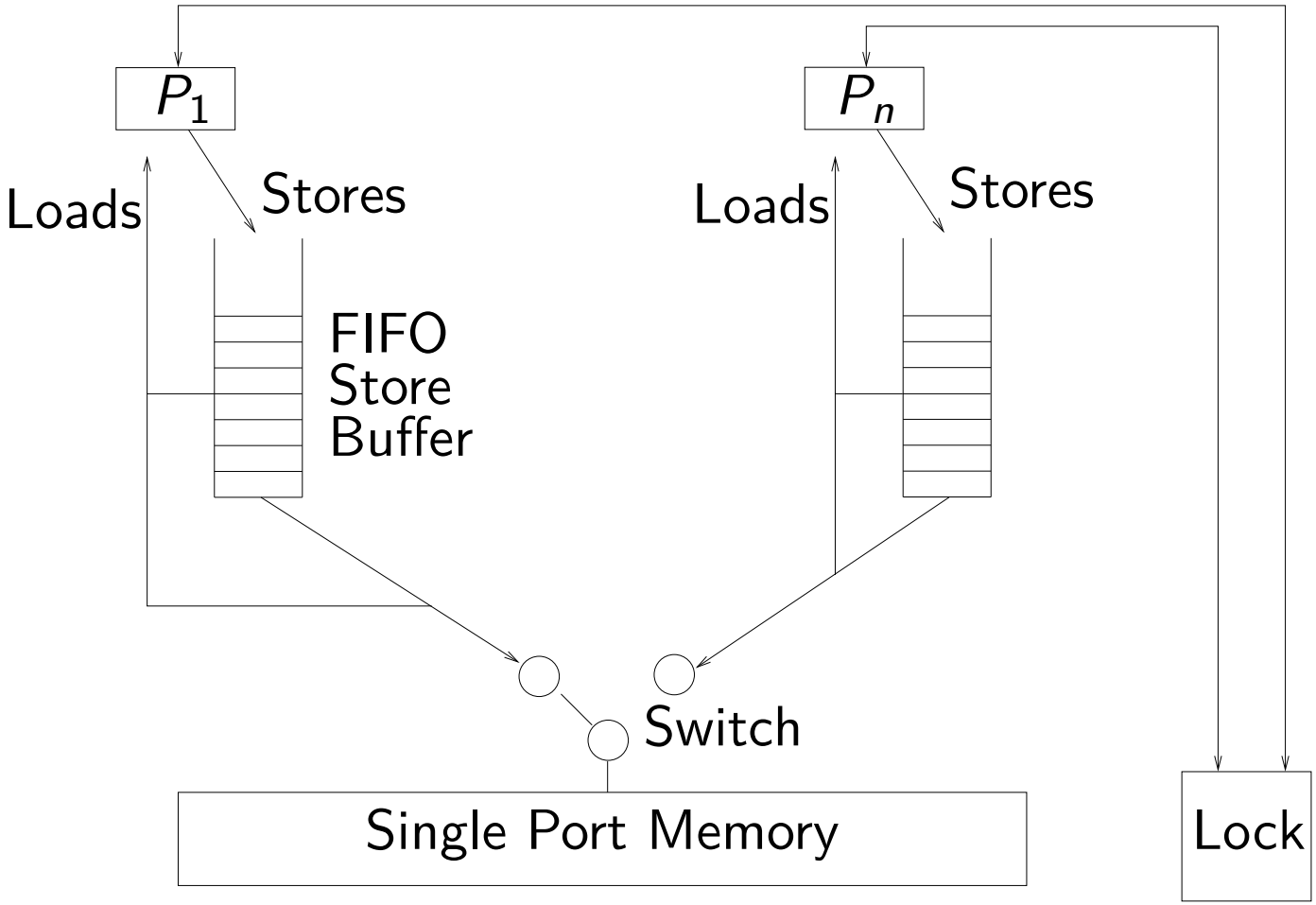
  mfence(p) blocks processor p until its buffer is empty.

  lock(p) If the lock is not already held by another processor, p takes the lock and obtains exclusive access to the global memory : the other processors are not allowed to execute the operations *commit* or *load*

  unlock(p) p flushes the buffer to global memory and releases the lock.

x86-TSO

Operational view



Other memory models

- Partial Store Order (PSO)
- Relaxed Memory Order (RMO)