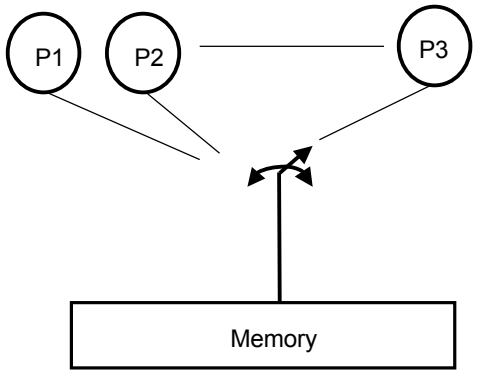
18-742

Lecture 16

Memory Ordering Revisited



Spring 2005
Prof. Babak Falsafi
http://www.ece.cmu.edu/~ece742

Slides developed in part by Profs. Adve, Falsafi, Hill, Lebeck, Reinhardt, Smith, and Singh of University of Illinois, Carnegie Mellon University, University of Wisconsin, Duke University, University of Michigan, and Princeton University.

Readings

Chapter 9 of Culler & Singh

Reader 5:

- **Sarita Adve and Kourosh Gharachorloo, *Shared Memory Consistency Models: A Tutorial*, IEEE Computer, pp 66-76, Dec. 1996.**
- ***The SPARC Architecture Manual*, v9 (Memory Model chapter).**
- **C. Gniady, B. Falsafi, and T. N. Vijaykumar, *Is SC + ILP = RC?*, ISCA 1999.**

Review: Coherence vs. Consistency

- Intuition says loads should return latest value
  - what is latest?
- Coherence concerns only one memory location
- Consistency concerns apparent ordering for all locations
- A Memory System is Coherent if
  - can serialize all operations to that location such that,
  - operations performed by any processor appear in program order
    - » program order = order defined program text or assembly code
  - value returned by a read is value written by last store to that location

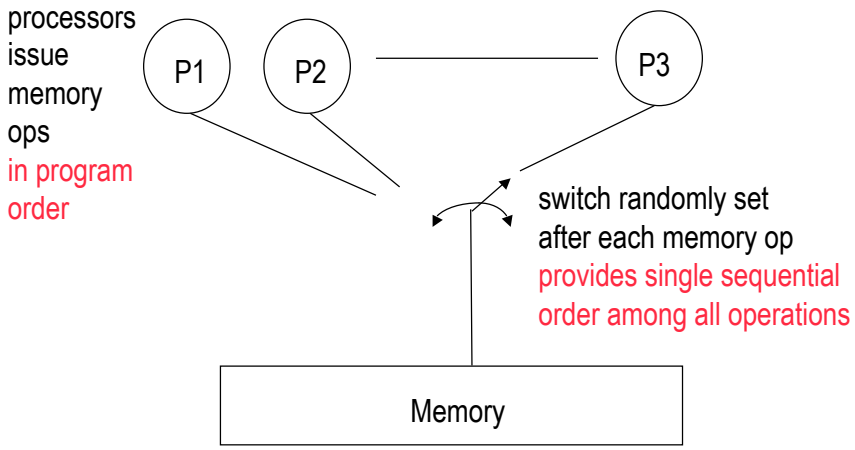
Review: Why Coherence != Consistency

```
/* initial A = B = flag = 0 */
 P1                P2
A = 1;             while (flag == 0); /* spin */
B = 1;             print A;
flag = 1;          print B;
```

Intuition says printed A = B = 1

Coherence doesn't say anything, why?

Review: Sequential Consistency (SC)



Review: Sufficient Conditions for SC

- **Every proc. issues memory ops in program order**
- **Memory ops happen (start and end) atomically**
  - **must wait for store to complete before issuing next memory op**
  - **after load, issuing proc waits for load to complete, before issuing next op**
- **Easily implemented with a shared bus**

Relaxed Memory Models

- **Motivation with Directory Protocols**
  - Misses have longer latency (do cache hits to get to next miss)
  - Collecting acknowledgements can take even longer
- **Recall SC has**
  - Each processor generates at total order of its reads and writes (R-->R, R-->W, W-->W, & W-->R)
  - That are interleaved into a global total order

Example Relaxed Models

- **PC: Relax ordering from writes to (other proc's) reads**
- **RC: Relax all read/write orderings (but add "fences")**

Relax Write to Read Order

```
/* initial A = B = 0 */
  P1              P2
A = 1;          B = 1
r1 = B;         r2 = A;
```

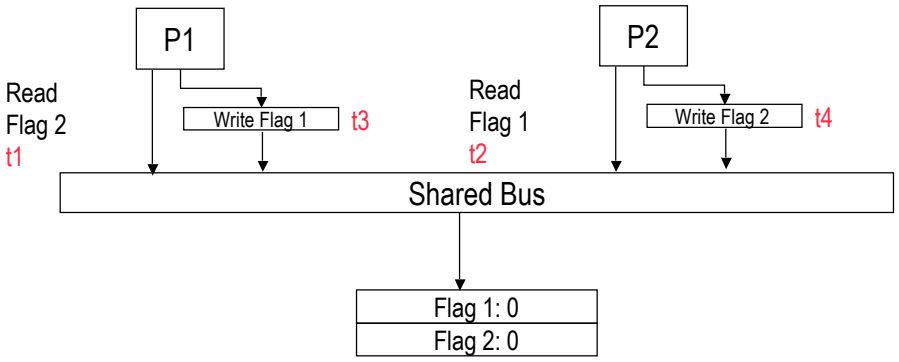
Processor Consistent (PC) Models

- **Allow r1==r2==0 (precluded by SC)**
  - Examples: IBM 370, Sun TSO, & Intel IA-32

Why do this?

- **Allows FIFO write buffers**
- **Does not astonish programmers (too much)**

Write Buffers w/ Read Bypass




```
P1                          P2
Flag 1 = 1                  Flag 2 = 1
if (Flag 2 == 0)            if (Flag 1 == 0)
   critical section            critical section
```


Also Want "Causality"

/* initially all 0 */

P1	**P2**	**P3**
A = 1;	**while (flag1==0) {};**	**while (flag2==0) {};**
flag1 = 1;	**flag2 = 1;**	**r3 = A;**

All commercial models guarantee causality

Why Not Relax All Order?

```
/* initially all 0 */
 P1                  P2
A = 1;               while (flag == 0); /* spin */
B = 1;               r1 = A;
flag = 1;            r2 = B;
```

Reorder of "A = 1"/"B = 1" or "r1 = A"/"r2 = B"

- Via OOO proc., non-FIFO write buffers, delayed directory acks., etc.

But Must Order

- "A = 1"/"B = 1" before "flag =1"
- "flag != 0" before "r1 = A"/"r2 = B"

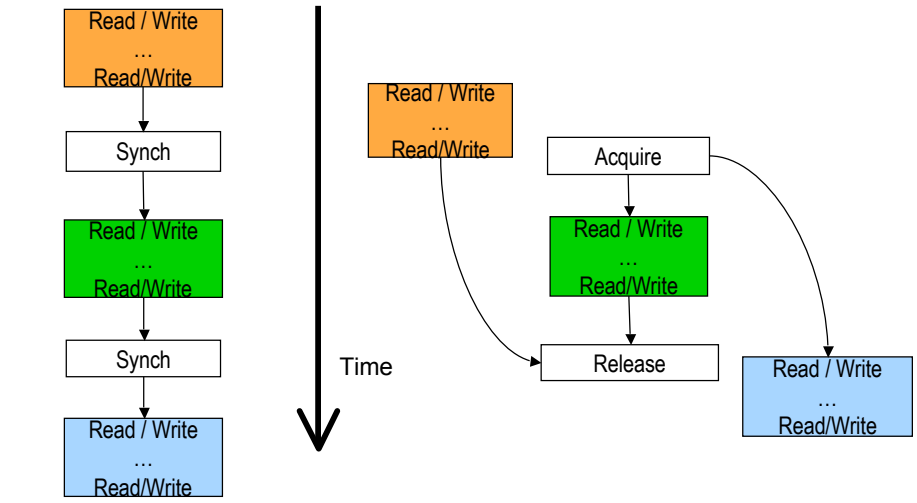
Order with "Synch" Operations

```
/* initially all 0 */
 P1                  P2
A = 1;               while (SYNCH  flag == 0);
B = 1;               r1 = A;
SYNCH flag = 1;      r2 = B;
```

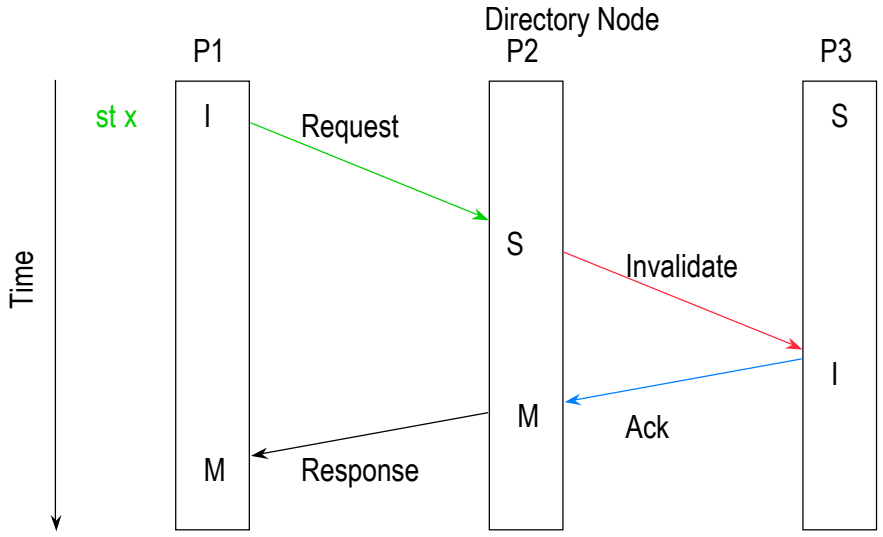
1. Called "weak ordering" of "weak consistency" (WC)
2. Alternatively, relaxed consistency (RC) specializes

- Acquires: force subsequent reads/writes after
- Releases: force previous reads/writes before

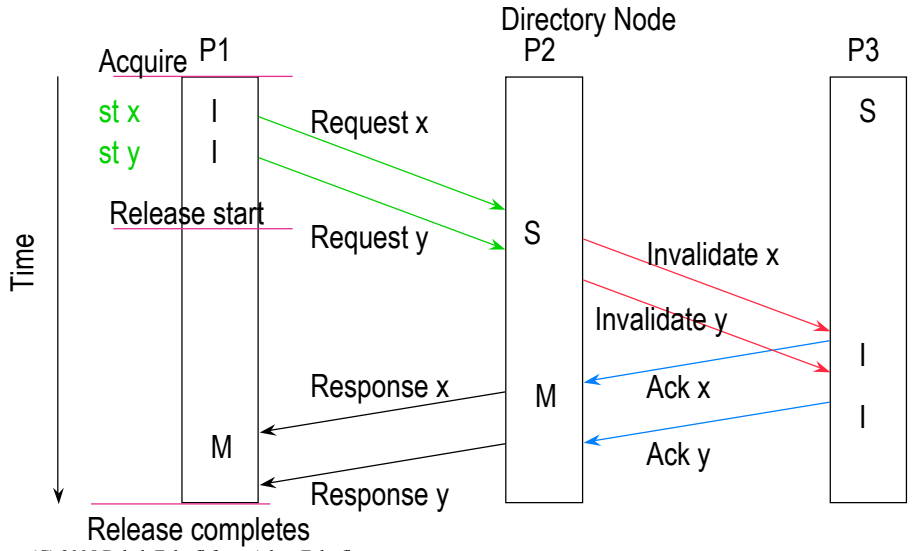
WC (Left) & RC (Right) Examples



Directory Example Sequential Consistency



Directory Example Release Consistency



Commercial Models use "Fences"

/* initially all 0 */

P1	**P2**
A = 1;	**while (flag == 0);**
B = 1;	**FENCE;**
FENCE;	**r1 = A;**
flag = 1;	**r2 = B;**

Examples: Compaq Alpha, IBM PowerPC,& Sun RMO

- **Can specialize fences (e.g., RMO)**

The Programming Interface

- WC & RC require synchronized programs
- All synchronization operations must be labeled and visible to the hardware
  - easy if synchronization library used
  - must provide language support for arbitrary ld/st synchronization (event notification, e.g., flag)
- Program that is correct for TSO portable to WO & RC

The Big Debate

When shall hardware enforce order?

Always order:

- Always (SC)
  - » Software wants a "multiprogrammed CPU" behavior
- Easy to program & understand
- Assumed to exhibit inferior performance

Relax order:

- Order enforced through software annotation
- Let the hardware overlap write latency

The Big Misconception: Large Performance Gap

But, strong ordering thought to hurt performance
- One reason for a variety of memory models and flavors

Not true!

Memory only has to appear to be ordered
- Hardware can relax order speculatively
- Save state while speculating
- Roll back if relaxed order observed by others
- E.g. result, SC + Speculation ≥ RC!

This is the Bart Simpson's approach to relaxing order:
"I didn't do it. Noone saw me doing it!"

Evolution of SC Systems

Naiive SC: every access is ordered

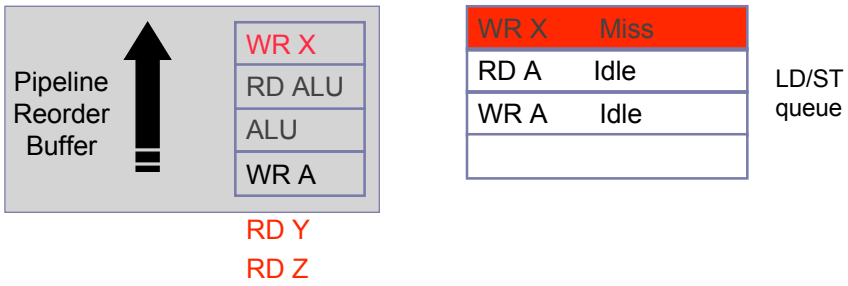
Optimized SC:
- Three simple optimizations
- Existing pipeline HW
- E.g., MIPS R10K

Wait-free SC

Enhancing SC's Performance

1. **Store buffering**
   - Store misses (either missing data or permission) can be removed from pipeline in program order
   - Place them in a store buffer
2. **All miss "fetches" can be overlapped**
   - L1 cache is a point of serialization (think "switch")
   - Requests for cache blocks can be sent out in parallel
   - All accesses to L1 must be atomic and in order
   - Therefore:
     - Order in which blocks are fetched/filled has no bearing on actual observed program order
     - Not until L1 cache access for the block is performed
3. **Load hits can "speculate" past store misses**

Memory Ordering in Naiive SC

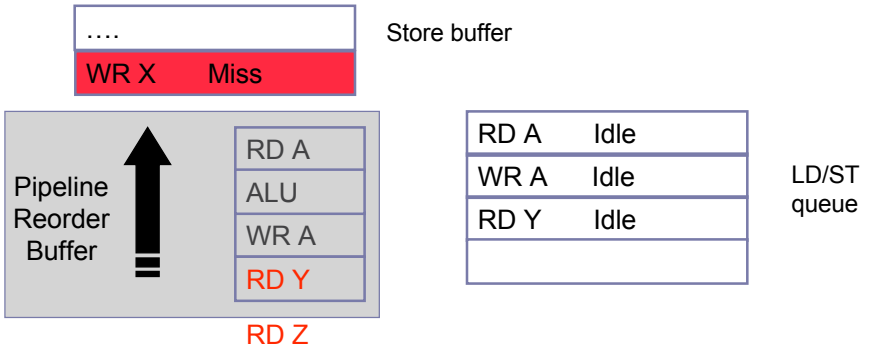


- **WR X, RD Y, RD Z access remote memory**
- **X, Y, Z, A are unrelated → need not be ordered**

①WR X blocks pipeline hundreds of cycles

②Can not overlap RD Y & RD Z with WR X

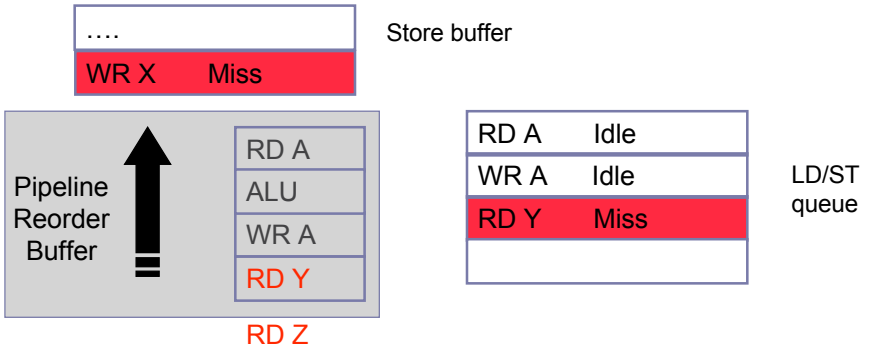
SC + Store Buffer



Store buffer

+ Removes pending stores from the head of ROB

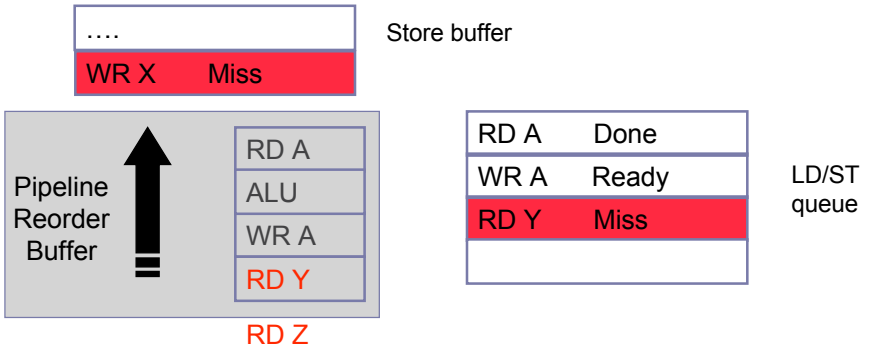
SC + Store Buffer + Multiple Pending Misses



Pre-fetch cache blocks

+ Overlaps multiple pending latencies

MIPS R10000:
SC + Store Buffer + Multiple Misses + Load Speculation



Load speculation

+ Allows load to bypass pending stores speculatively

Not speculative enough! Why?

Alleviating Reorder Buffer Stalls

Speculative Retirement [Ranganathan, et al.]

- **Small extension to the reorder buffer**
- **Speculatively graduate reorder buffer instructions**
- **Loads/Stores remain ordered in LD/ST queue**
- **Significant gap remains**

Our observation:

Small ILP buffering not enough for long DSM latencies

Wait-Free Sequential Consistency

Reminder:
- SC must only appear in program order
- need order only when others race to access

SC hardware can emulate RC iff
- overlap accesses speculatively
- keep a log of computation in program order
- roll back in case of a race

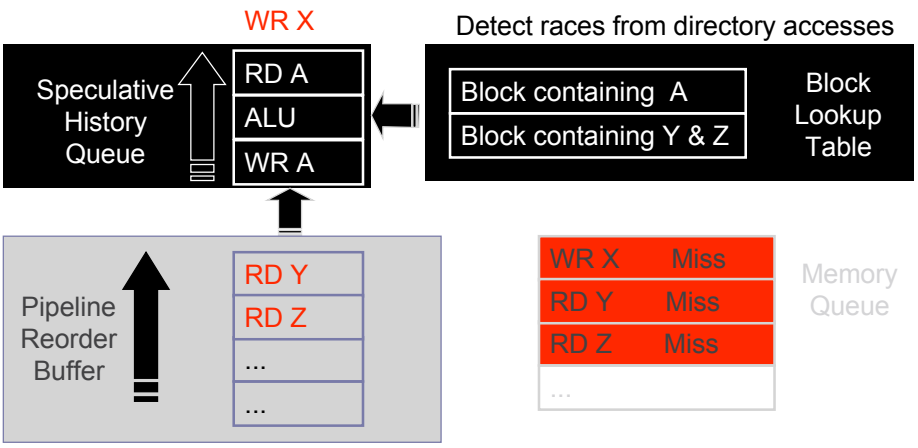
+ no help from software → SC programming
+ infrequent rollback → better than RC performance (why?)

Requirements for Wait-Free SC

[Gniady, et al.]:

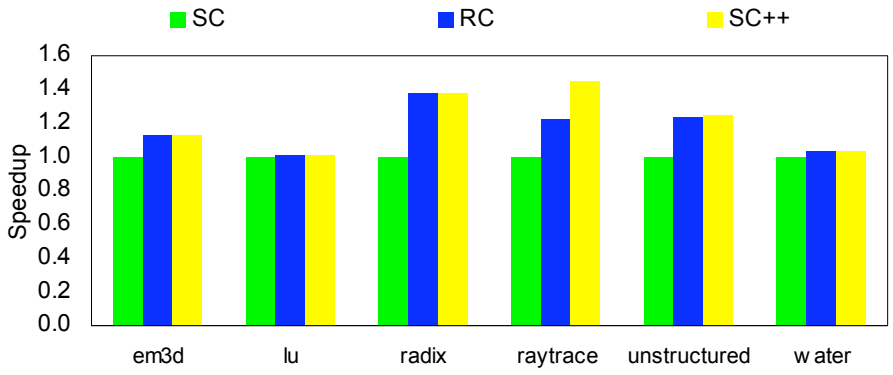
- ❑ Full custom H/W support for speculation
- ❑ Large speculative state to tolerate long latencies
  - ❑ Maintains old processor state
  - ❑ Maintains old memory state
- ❑ Fast lookup to detect possible order violation
  - ❑ upon cache invalidations and replacements
- ❑ Infrequent rollbacks
  - – Rollbacks are due to false sharing or data races
  - – Typical of well-behaved applications

SC++: A Design to Emulate RC



- SHiQ: Back up computation in a queue
- BLT: Quick lookup to detect races

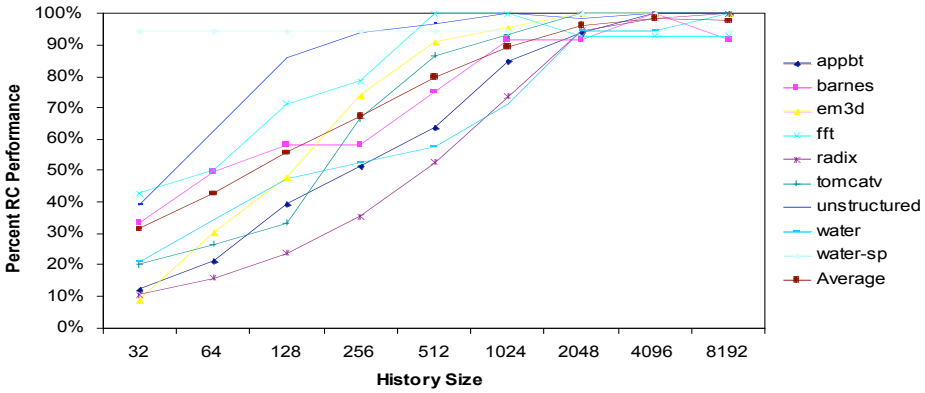
Base SC, RC, SC++ Comparison



- Data from RSIM DSM simulator
- 8, 300 MHz MIPS R10000 processors
- Up to 40% gap between SC & RC

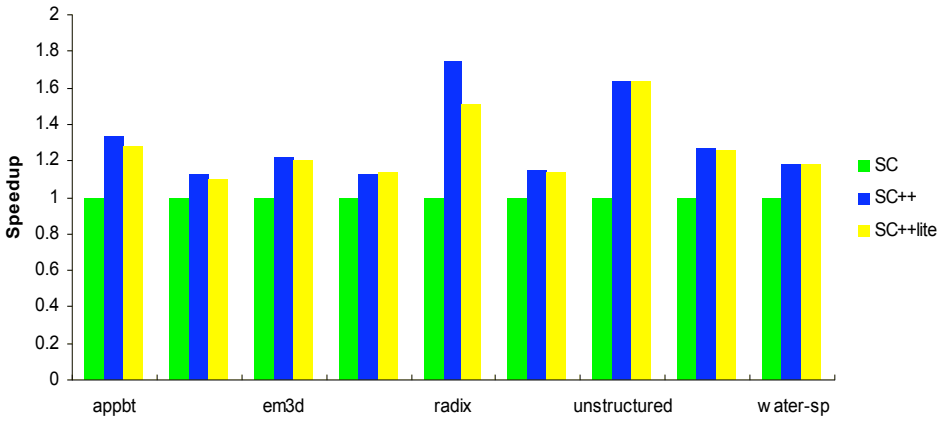
SC++ can fully emulate RC

Required History Size



- History size varies across applications (& system size)
- Worst-case history => >64K
- History is bursty => on average 70%-90% empty SHiQ

SC++lite [PACT paper]: Buffering History in L2



- Can get effect of infinite buffering
- Minimal history storage overhead

Can We Build a Wait-Free Relaxed System?

This helps get dusty-deck SW to run fast

- Solaris is TSO-compatible
- Can implement a wait-free TSO

Relaxed systems always enforce order at fence ops

Speculation semantics:

- Start with a pending fence op
- Commit with the acks to accesses prior to fence
- Roll back in case of race to accesses prior to fence