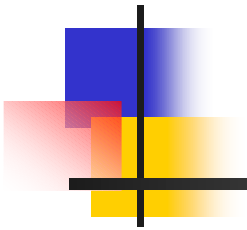


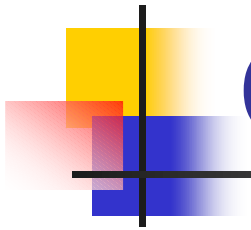
Memory Ordering Mechanisms for ARM?

Tao C. Lee, Marc-Alexandre Boéchat

CS, EPFL

Forecast

- This research studies the performance of memory ordering mechanisms on Chip Multi-Processors (CMPs) for modern workloads.
- Results using both aggressive cores and simple cores will be presented.
- Results for server workloads and cloud workloads will be presented.
- Reflections on ARM's memory model will be addressed.



Outline

- Motivation & problem statement
- Background & related work
- Methodology
- Results
- Summary
- Future work

Motivation & problem statement

- Memory ordering was considered a significant performance bottleneck for Distributed-Shared Memory (DSM) server systems.
- Chip Multi-Processors (CMPs) and energy-efficient computing are emerging and making disruptive impacts.
- Is memory ordering still a serious problem for CMPs and future energy-efficient processors (e.g. ARM Cortex-A15)?

Background & related work

ST X
ST B
LD A
ALU
ST A
LD Y
ST Z

- Memory ordering
  - Should ordering of load/store instructions be preserved?
  - Can we or should we relax this restriction to improve performance?
    - SC: preserve orders sequentially
    - TSO: relax ordering of loads, but keep ordering of stores
    - RMO: relax all, only keep ordering if specified
- Memory ordering research in the past
  - Is SC + ILP = RC? (Chris Gniady et al., 1999)
  - Mechanisms for Store-wait–free Multiprocessors (Thomas F. Wenisch et al., 2007)

Methodology (1/2)

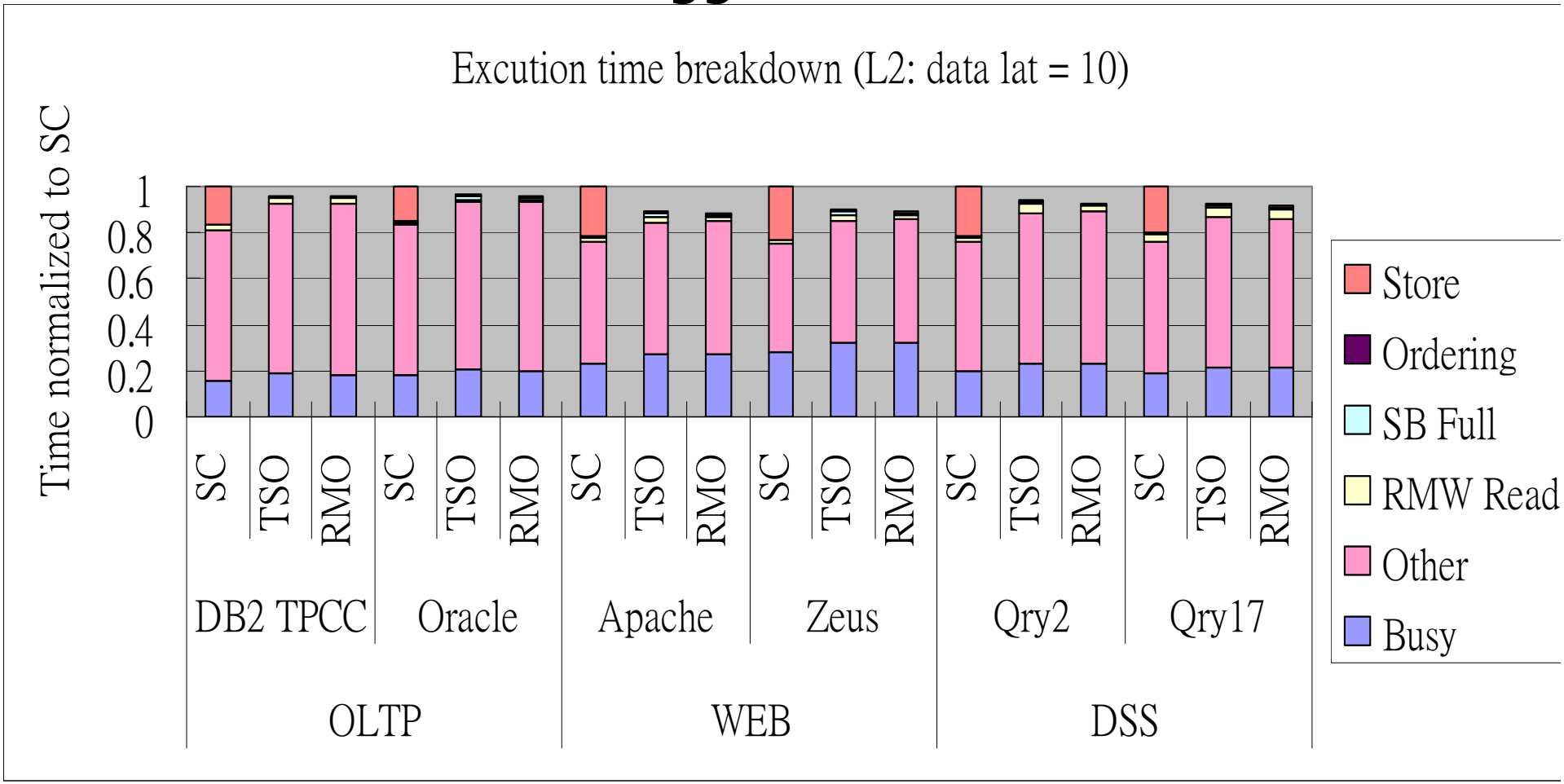
- Full-system Chip Multi-Processors (CMPs) simulation platform (PARSA, EPFL)
  - Flexus 4.0: current version of CMPs simulator – simulate both system and user behavior for un-modified commercial software
  - Statistical sampling: sample system states with flexpoints to reduce simulation time with sufficient confidence interval
  - Support SC, TSO, RMO memory ordering mechanisms
- Modern workloads
  - Server workloads – databases, web servers, and decision systems
    - OLTP: DB2 TPCC, Oracle
    - Web: Apache, Zeus
    - DSS: Qry2, Qry17
  - Cloud workloads – emerging cloud workloads for data centers
    - Web09_bank, cloud9, nutch, word_count, cassandra_test
- Modern processors -- Aggressive cores and simple cores

Methodology (2/2)

- Architecture parameters
  - Aggressive cores
    - Processing nodes: 16 cores, SUN SPARC III V9 ISA, 3GHz, 8-stage pipeline, 4-issue, 96-entry ROB, 32-entry SB
    - L1 split I/D caches: 64KB, 2-way
    - L2 caches: 8MB, 16-way
    - Micro-architecture parameters:
      - memory consistency models (SC, TSO, RMO)
      - L2:data latency = 10, 50, 100 cycles
    - Interconnect: tiled mesh
  - Simple cores
    - Processing nodes: 16 cores, ARM Cortex-A15, 2GHz, 60-entry ROB, 16-entry SB
    - L1 split I/D caches: 32KB, 2-way
    - L2 caches: 4MB, 16-way
    - Micro-architecture parameters:
      - memory consistency models (SC, TSO, RMO)
      - L2: data latency = 4 cycles
    - Interconnect: tiled mesh

Results (1/5)

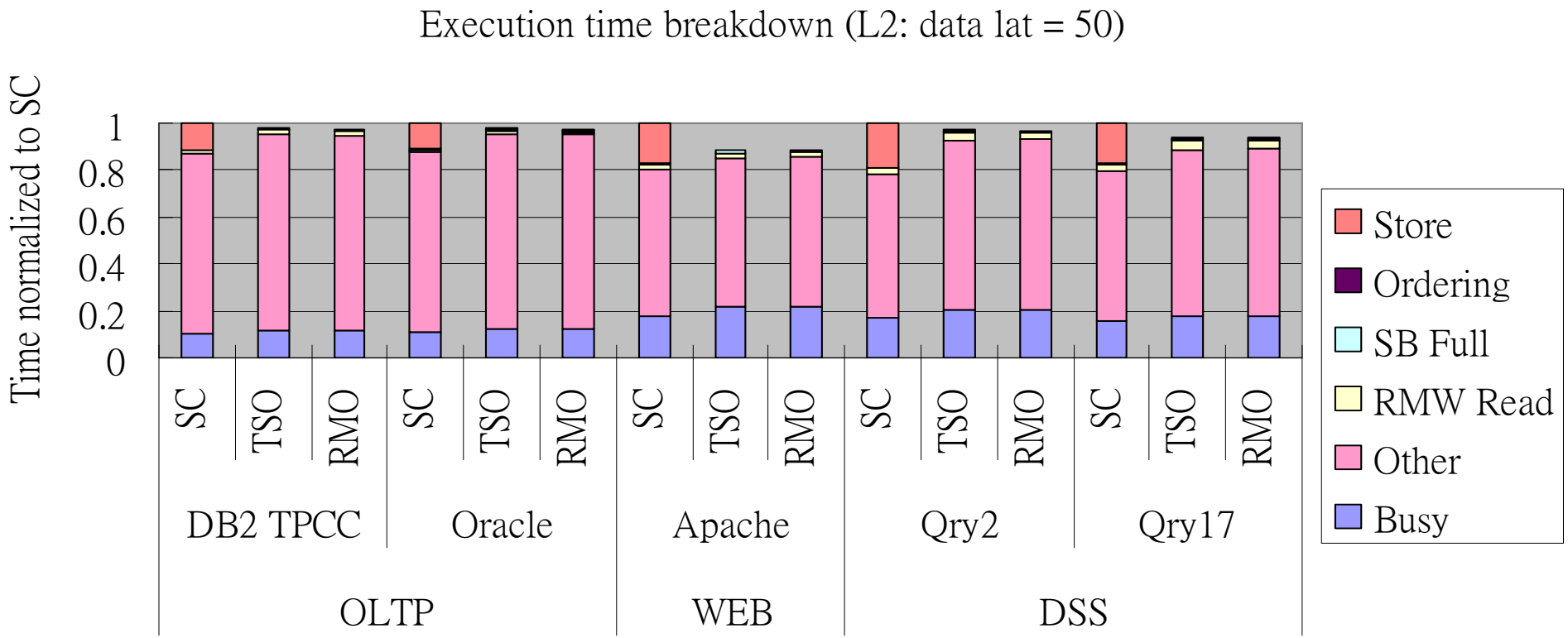
- Server workloads on aggressive cores



- 7% ~ 13% speedup from SC to TSO
- 1% ~ 3% speedup from TSO to RMO
- Store latencies significant (18% ~ 22%) for SC
- 20% busy time on average

Results (2/5)

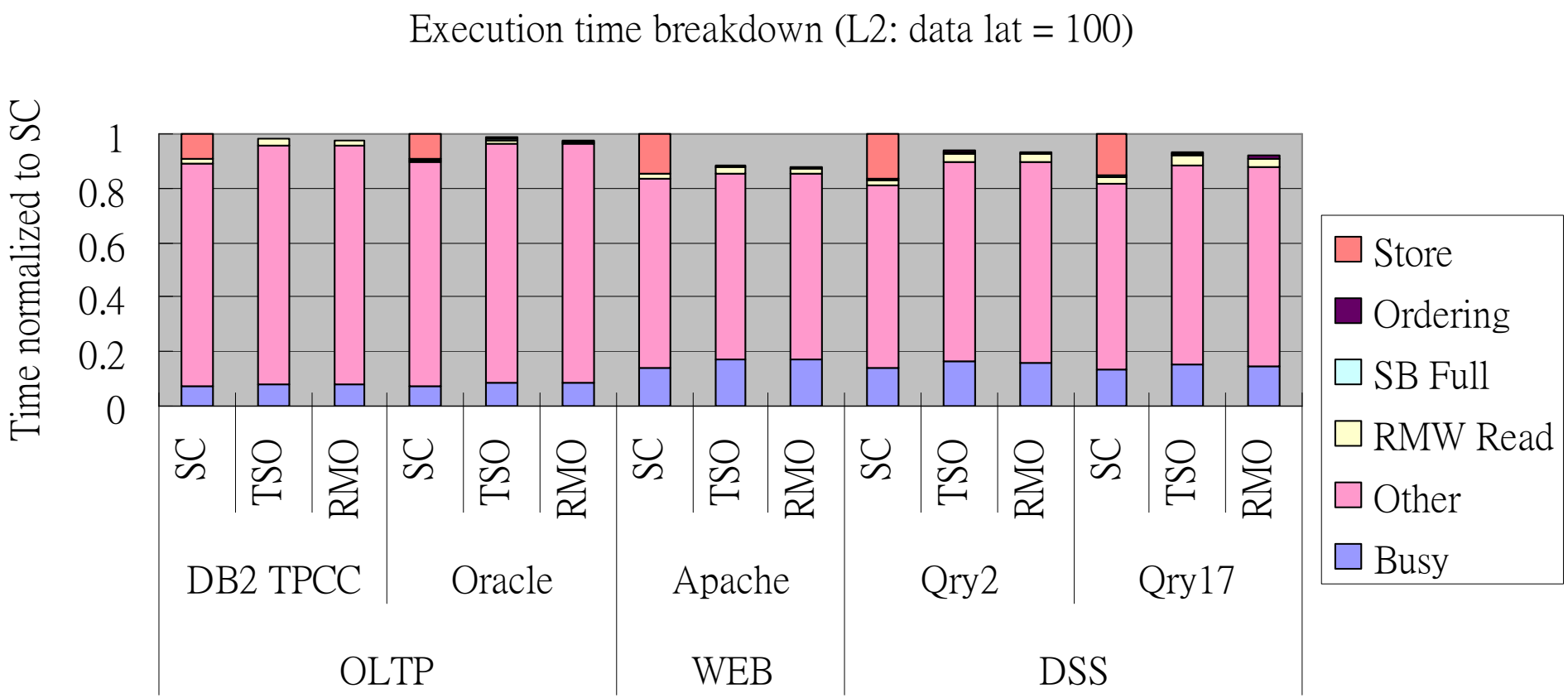
- Server workloads on aggressive cores



- 3% ~ 12% speedup from SC to TSO
- 1% ~ 2% speedup from TSO to RMO
- Other latencies (EmptyROB, L2:Load) begin to dominate
- 15% busy time on average

Results (3/5)

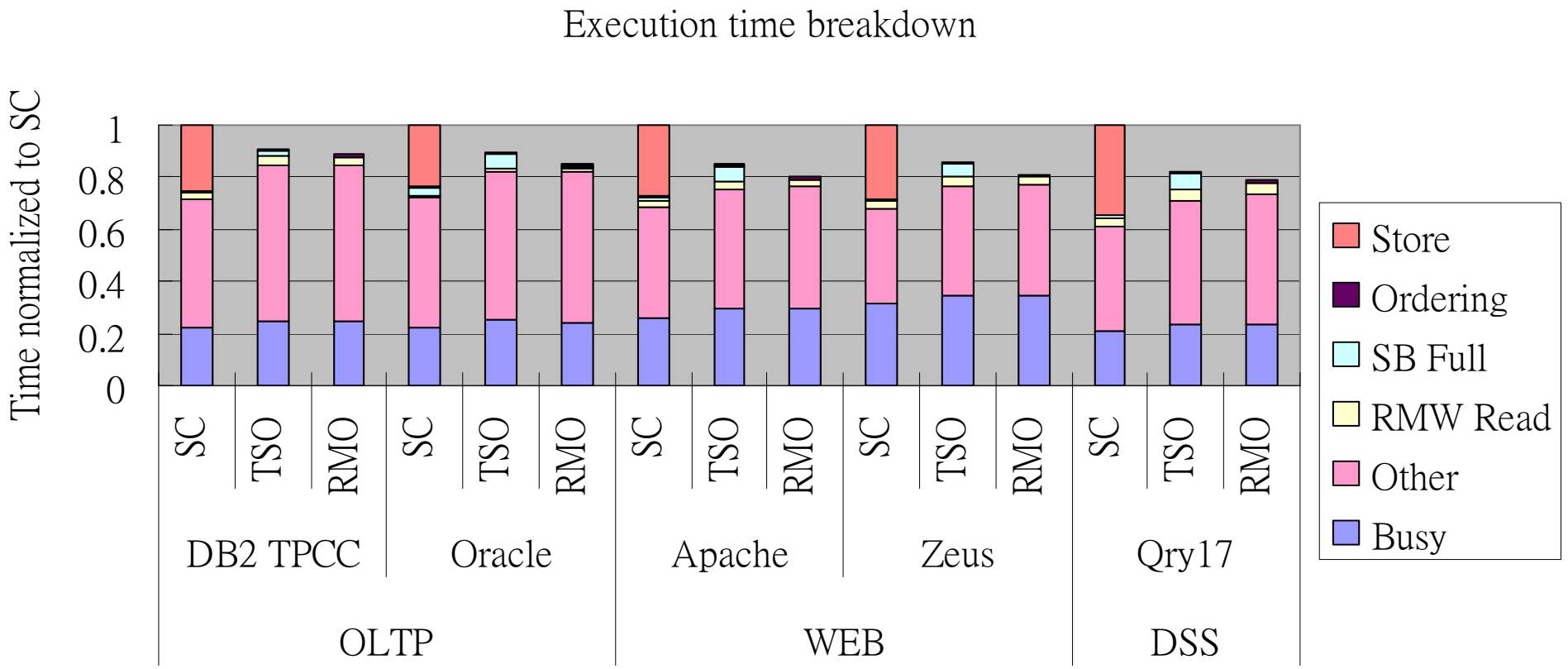
- Server workloads on aggressive cores



- 1% ~ 11% speedup from SC to TSO
- 1% ~ 2% speedup from TSO to RMO
- Other latencies (EmptyROB, L2:Load) dominate
- 10% busy time on average

Results (4/5)

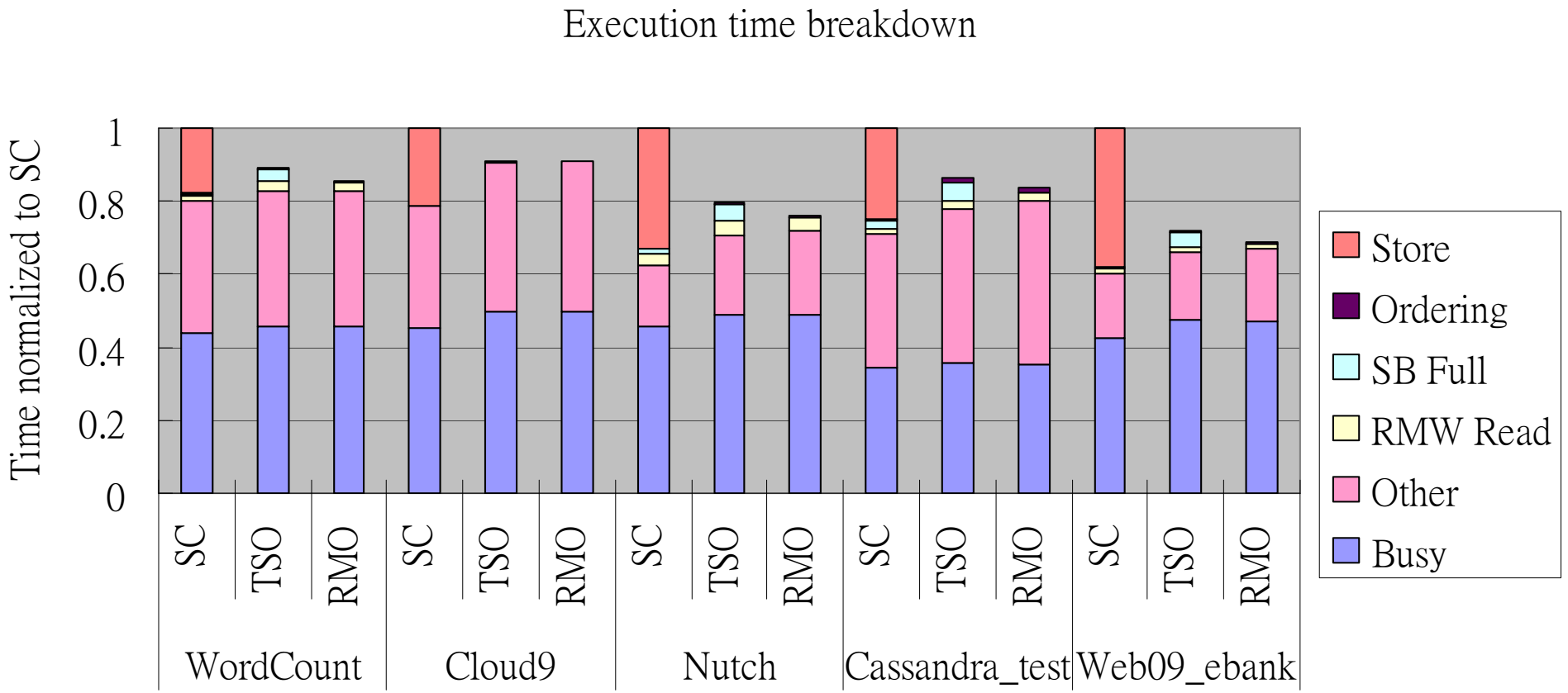
- Server workloads on simple cores



- 10% ~ 19% speedup from SC to TSO
- 3% ~ 6% speedup from TSO to RMO
- Store latencies significant (25% ~ 35%) for SC
- 25% busy time on average

Results (5/5)

- Cloud workloads on simple cores



- 11% ~ 28% speedup from SC to TSO
- 3% ~ 6% speedup from TSO to RMO
- Store latencies significant (19% ~ 39%) for SC
- 42% busy time on average

Reflections on ARM

- ARM's weakly consistent memory model
  - Model description
    - Similar to RMO – relax ordering constraints unless specified with barrier instructions.
  - Barrier instructions
    - DMB (Data Memory Barrier)
      - Instructions before the barrier are observed by all processors before the instructions after the barrier.
    - DSB (Data Synchronization Barrier)
      - Instructions before the barrier complete before the instructions after the barrier (global synchronization w/ acknowledgement).
- ARM cores for cloud & server workloads?
  - Evaluation on memory ordering
    - RMO is sufficient to provide performance guarantee for CMPs w/ aggressive cores or simple cores.
    - If performance suffering of 3% ~ 6% is acceptable, TSO could be considered as an alternative for its simpler programming model.

Summary

- Memory ordering penalties for CMPs
  - are not as severe as DSM server systems.
- TSO and RMO
  - almost provide the same performance with aggressive cores
  - slightly larger gap with simple cores
- Cloud workloads
  - are more computation-intensive than server workloads (high busy time),
  - but some of them suffer from more memory ordering stalls
- ARM's weak memory model
  - could provide good performance.
  - But TSO could provide simpler programming model with 3% ~ 6% performance penalties.

Future work

- Speculation
  - For simple cores, there is a performance gap between TSO and RMO, it would be interesting to see how to bridge this gap with speculation.
  - The energy efficiency and area overheads of using speculation for simple cores could be another interesting topic.
- Cloud workloads on aggressive cores
  - It would be interesting to see the memory ordering behavior of cloud workloads on aggressive cores (compared to the behavior of server workloads).

Thank you for your attention

Merry X’mas

Questions?

Appendix

- Mechanisms for Store-wait–free Multiprocessors (Thomas F. Wenisch et al., 2007)

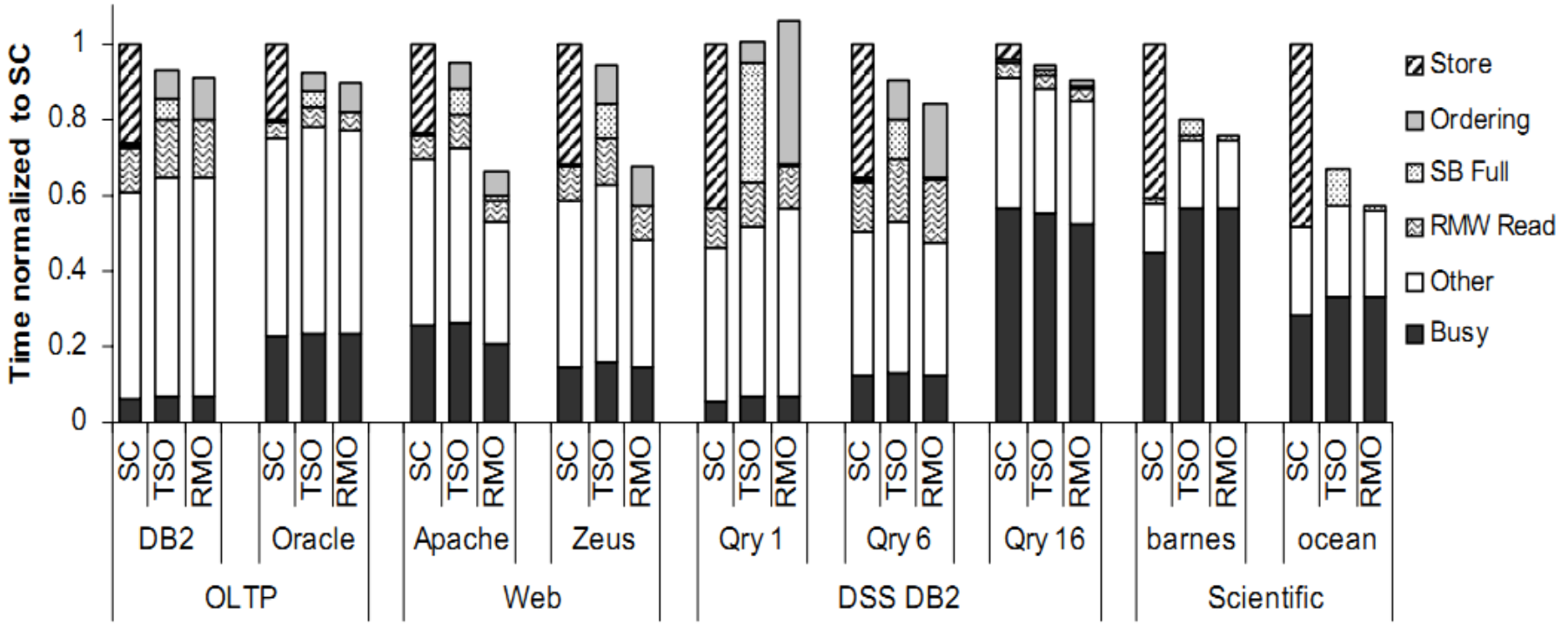
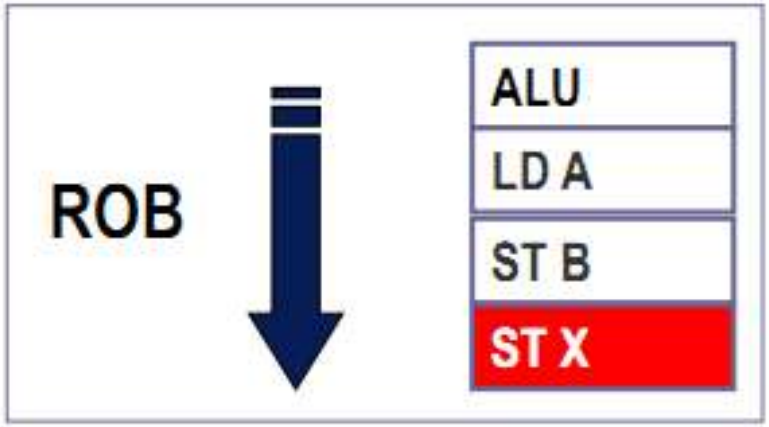


FIGURE 4. Execution time breakdown in conventional systems. Each bar shows an execution time breakdown under conventional implementations of SC, TSO, RMO, normalized to SC performance.

The following slides are from Prof. Falsafi's course slides

Memory Ordering in Naiive SC



ST A	Idle
LD A	Idle
ST B	Idle
ST X	Miss

LD/ST queue

....
....

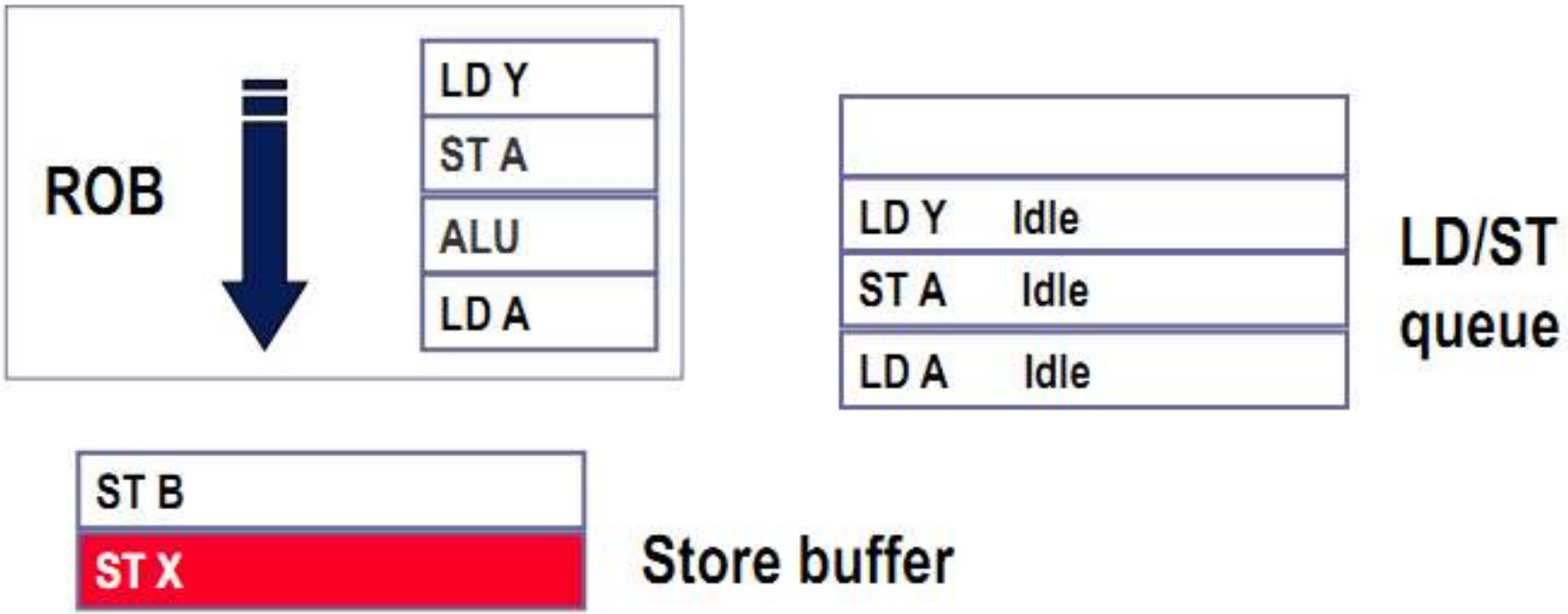
Store buffer

- ST X, LD Y, LD Z are long-latency misses
- X, Y, Z, A, B are unrelated → need not be ordered
  - CPU does not know they are unrelated

① ST X blocks CPU for hundreds of cycles

② Can not overlap LD Y & LD Z with ST X

SC + Store Buffer



Store buffer

+ Removes one pending store from the head of ROB

+ Remove back-to-back stores w/o coalescing (must preserve order)

Processor Consistency

- PC
  - Was built in systems in 1970's before it was defined
- Relax the loads with respect to stores
  - Let load go if there are only pending stores ahead
- Enforce order upon an atomic instruction
  - E.g., XCHG instruction in x86 or ldstub or swap instructions in SPARC
  - Wait for store buffer to drain before proceeding
- Examples: IBM 370, Sun TSO, & Intel x86
- Relatively simple model to understand (for programmers)

PC

LD/ST queue

Store buffer

LD A, ALU (unrelated) retire

LD Y and ST Z overlapped with ST X

- LD Y is allowed to proceed (load)

But, ST A blocks because Store buffer is full

Weak Ordering or Weak Consistency

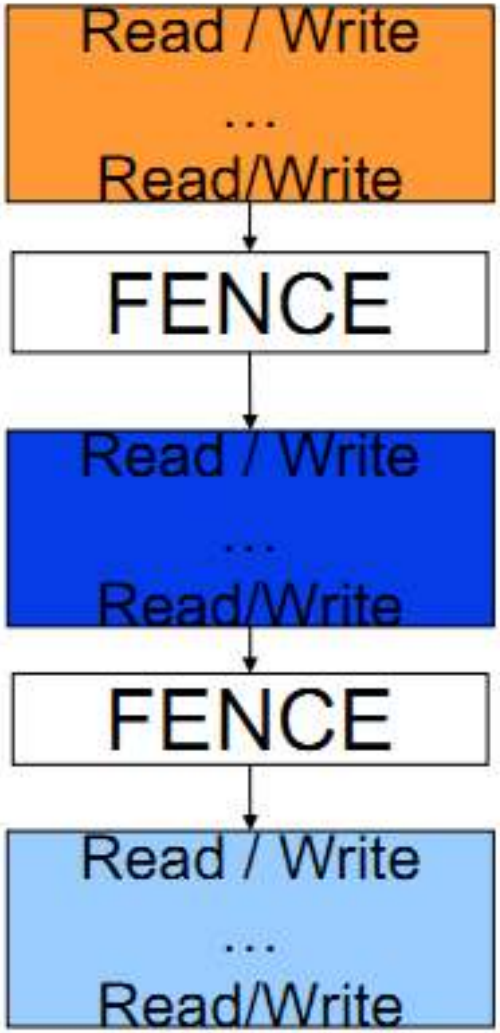
```
/* initially all 0 */
 P1                    P2
A = 1;                 while (FENCE flag == 0);
B = 1;                 r1 = A;
FENCE flag = 1;        r2 = B;
```

A special "FENCE" instruction in the code

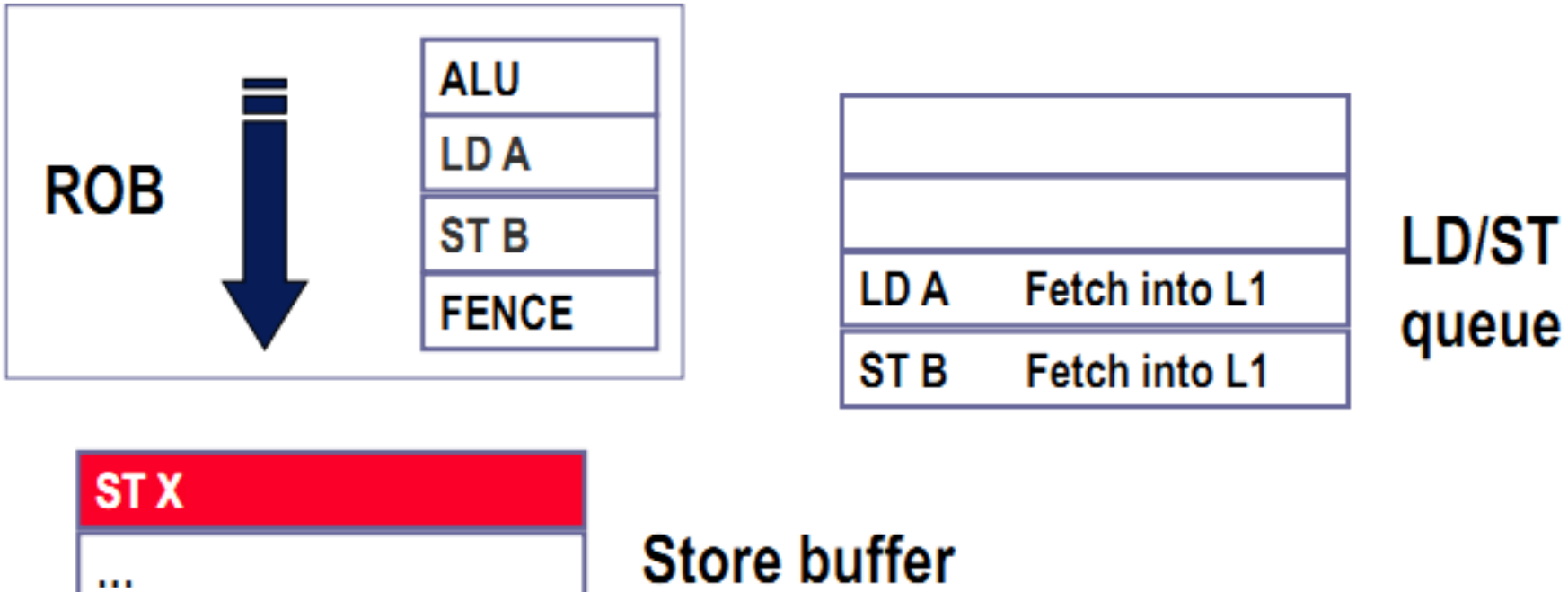
No order (but uniprocessor order) is preserved between two FENCE instructions

All order is preserved across a FENCE instruction

Weak Ordering Example



WC with FENCE



When FENCE reaches head of ROB, pipeline blocks!

ST A, LD Y and ST Z can not enter pipeline

- Enter after ST X is drained