ITSLF: Inter-Thread Store-to-Load Forwarding in Simultaneous Multithreading

Josué Feliu [1], Alberto Ros [1], Manuel E. Acacio [1], and Stefanos Kaxiras [2]

[1] Computer Engineering Department
University of Murcia

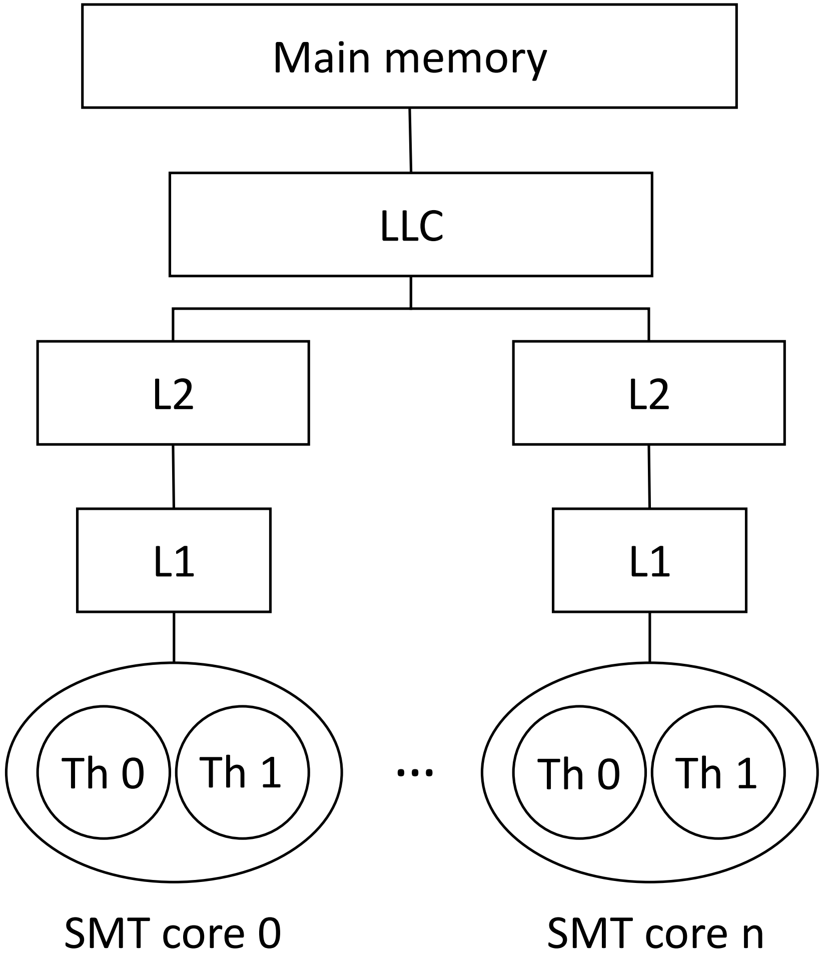
[2] Department of Information Technology
Uppsala University

Introduction

- Fine-grain, synchronization-intensive workloads scale poorly.
  - The farthest the synchronization, the more expensive.

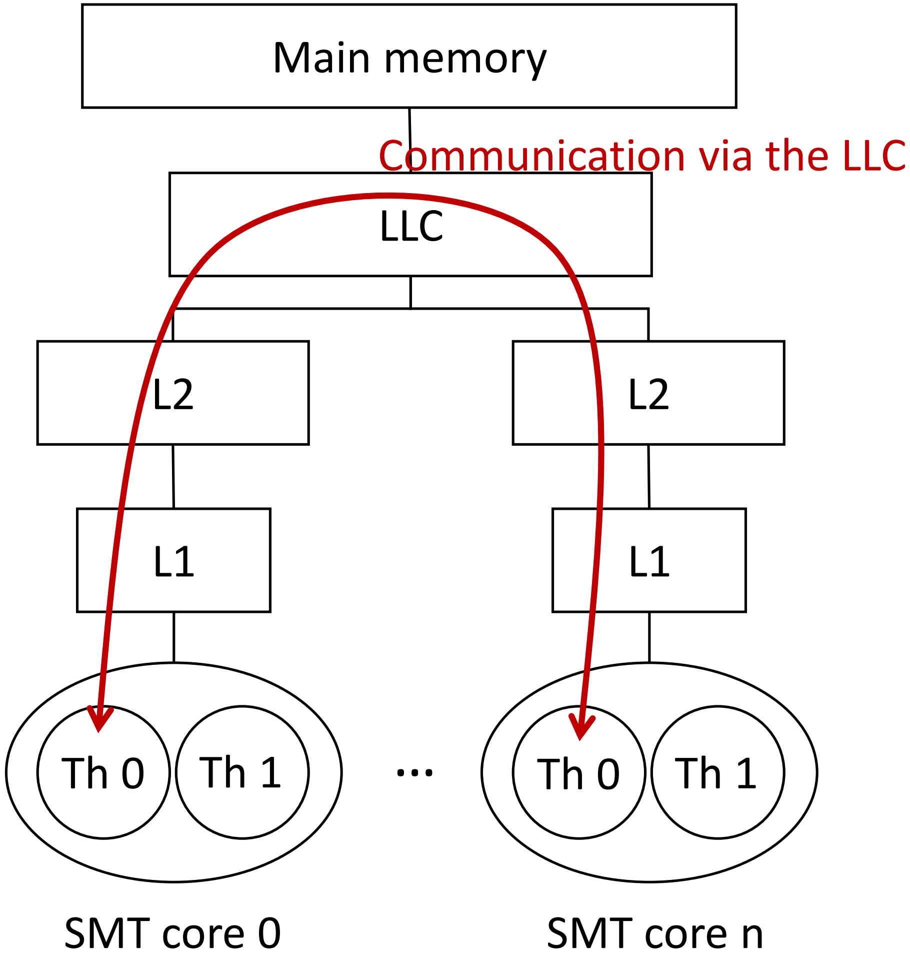
Introduction

- Fine-grain, synchronization-intensive workloads scale poorly.
  - The farthest the synchronization, the more expensive.



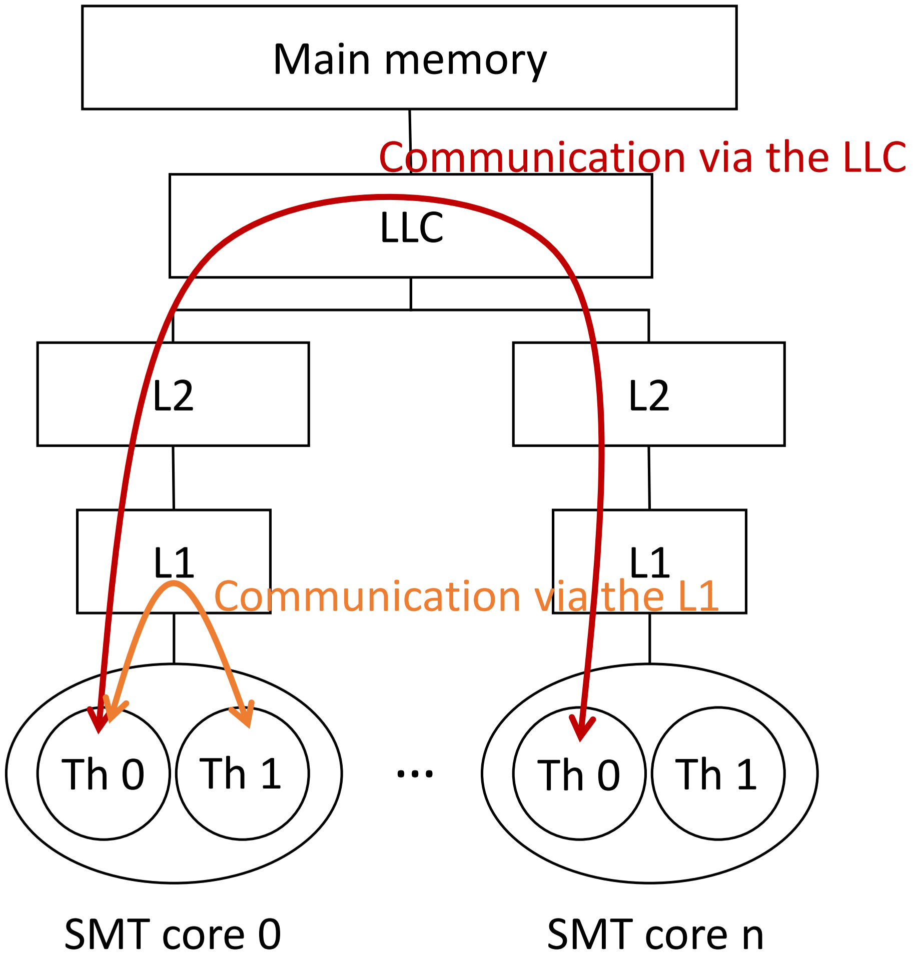
Introduction

- Fine-grain, synchronization-intensive workloads scale poorly.
  - The farthest the synchronization, the more expensive.



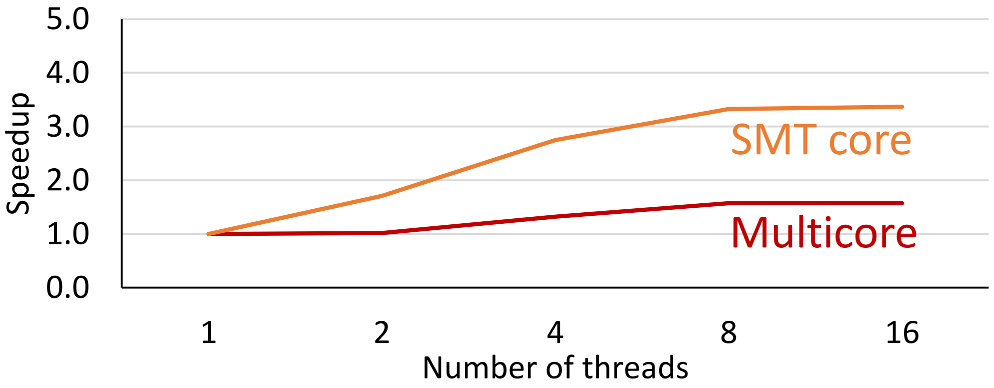
Introduction

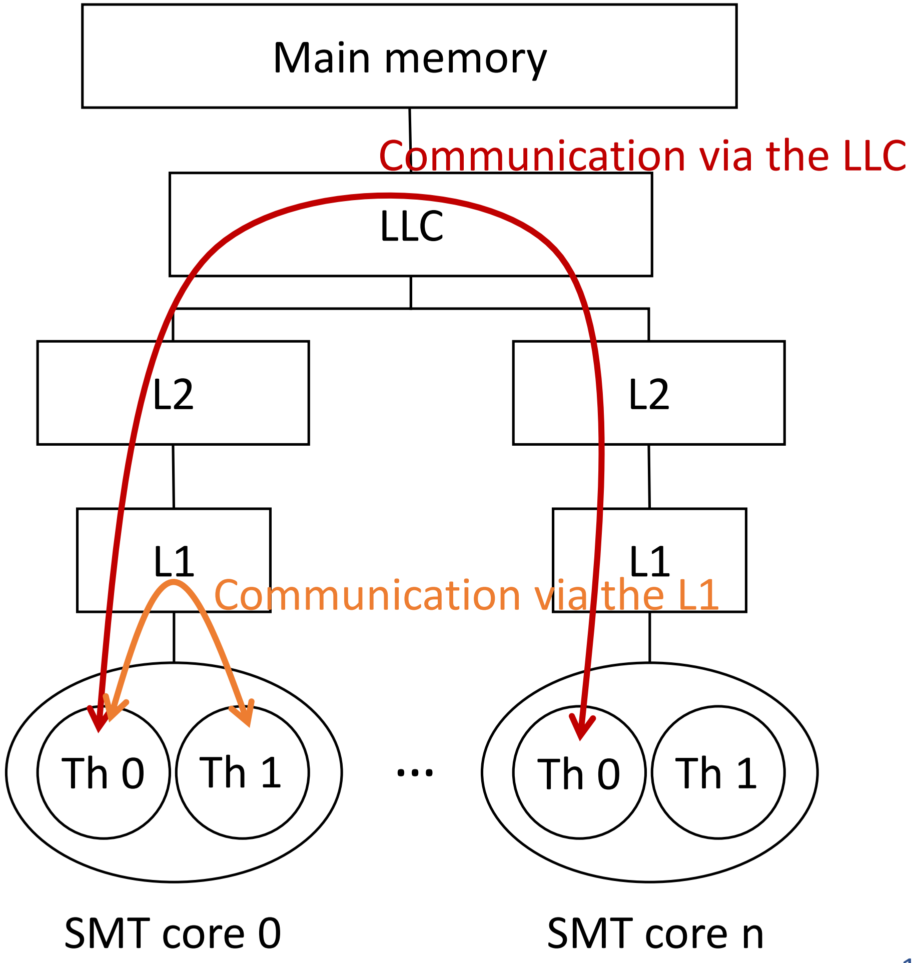
- Fine-grain, synchronization-intensive workloads scale poorly.
    - The farthest the synchronization, the more expensive.



Introduction

- Fine-grain, synchronization-intensive workloads scale poorly.
  - The farthest the synchronization, the more expensive.

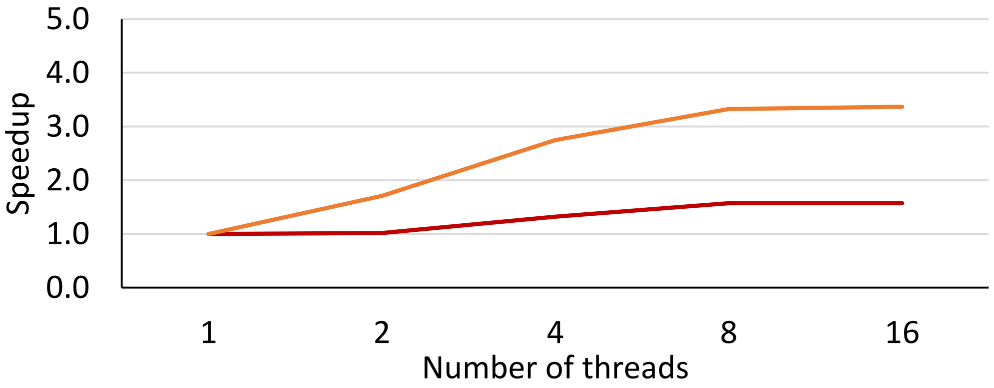


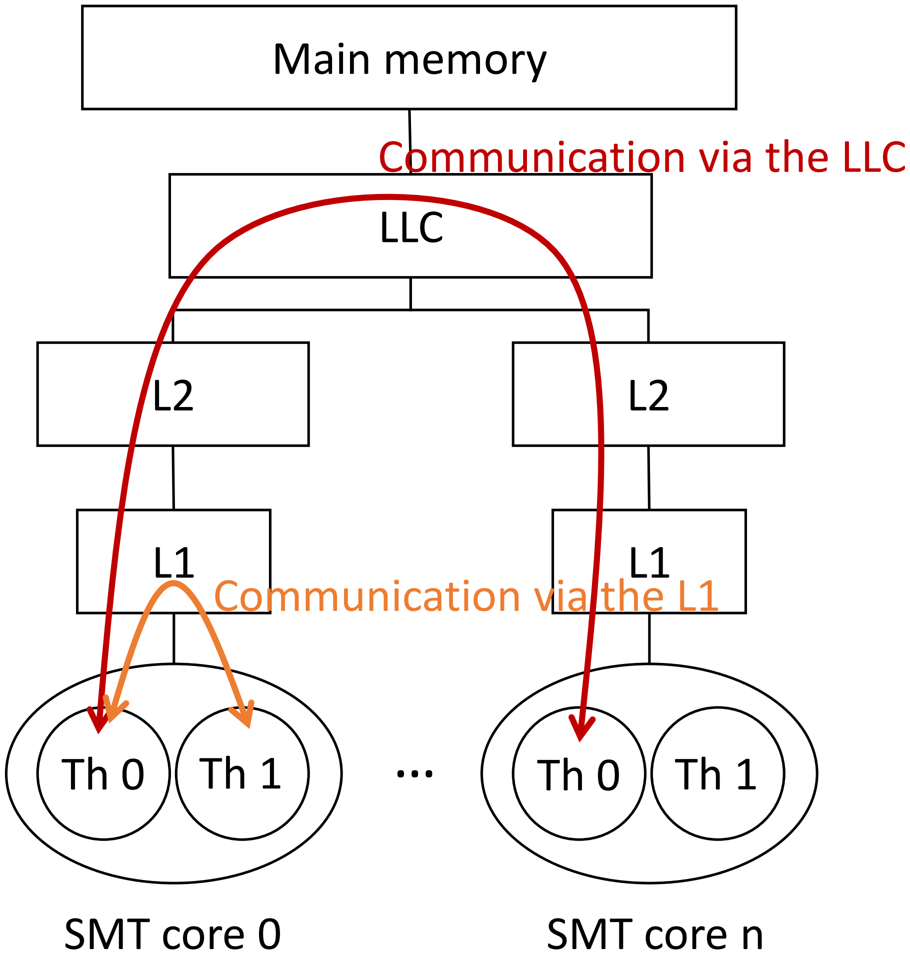


Introduction

- Fine-grain, synchronization-intensive workloads scale poorly.
  - The farthest the synchronization, the more expensive.

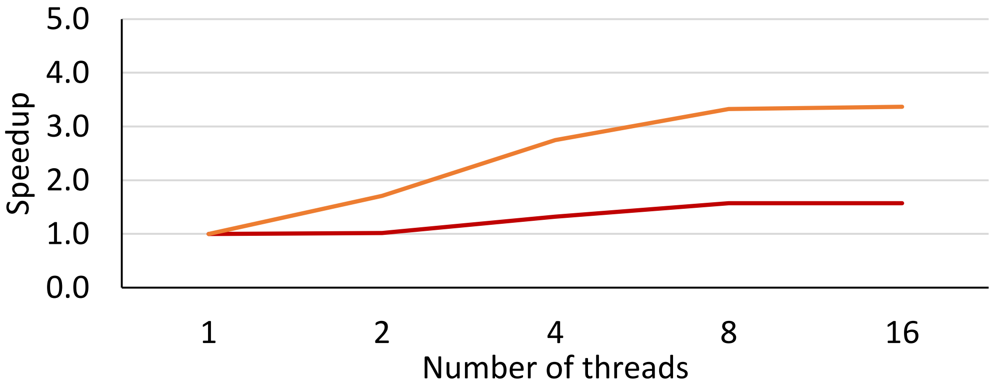
- Can we bring communication closer to the threads?

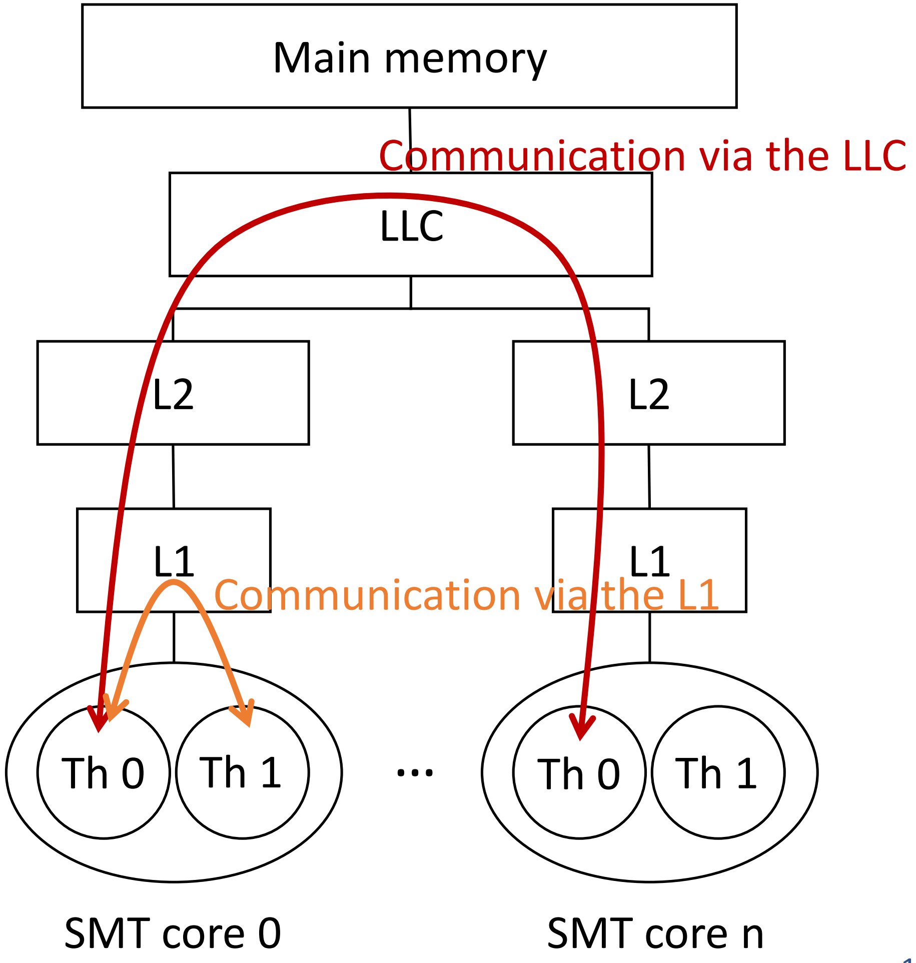




Introduction

- Fine-grain, synchronization-intensive workloads scale poorly.
  - The farthest the synchronization, the more expensive.
- Can we bring communication closer to the threads?
  - The first shared level between threads in an SMT is not the L1 cache but the SQ/SB.

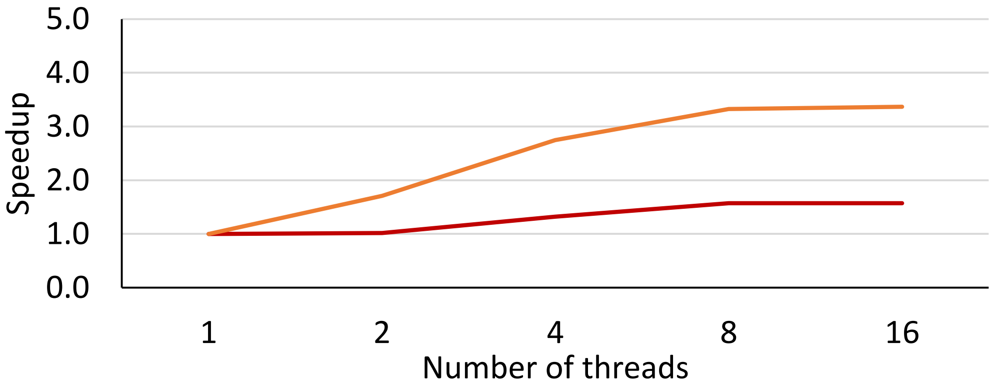


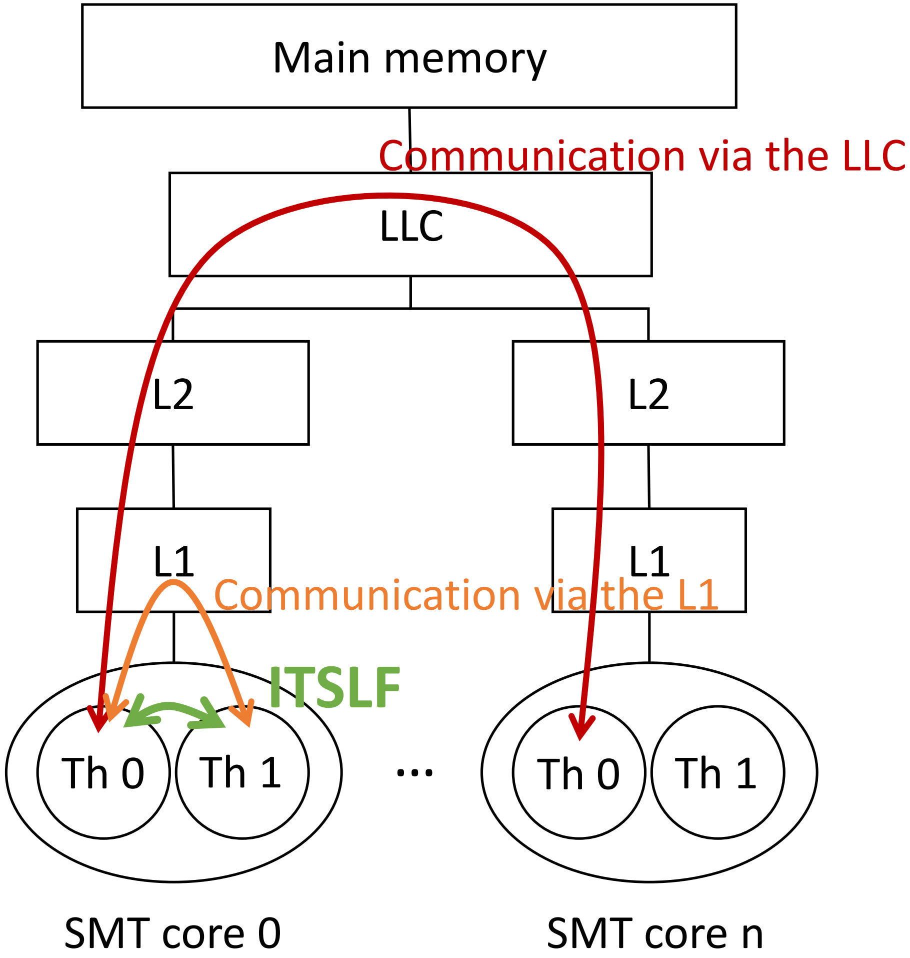


Introduction

- Fine-grain, synchronization-intensive workloads scale poorly.
  - The farthest the synchronization, the more expensive.

- Can we bring communication closer to the threads?
  - The first shared level between threads in an SMT is not the L1 cache but the SQ/SB.

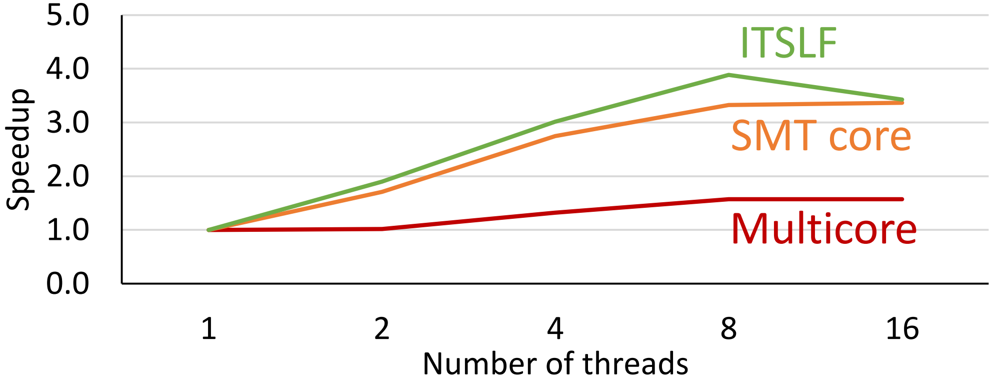


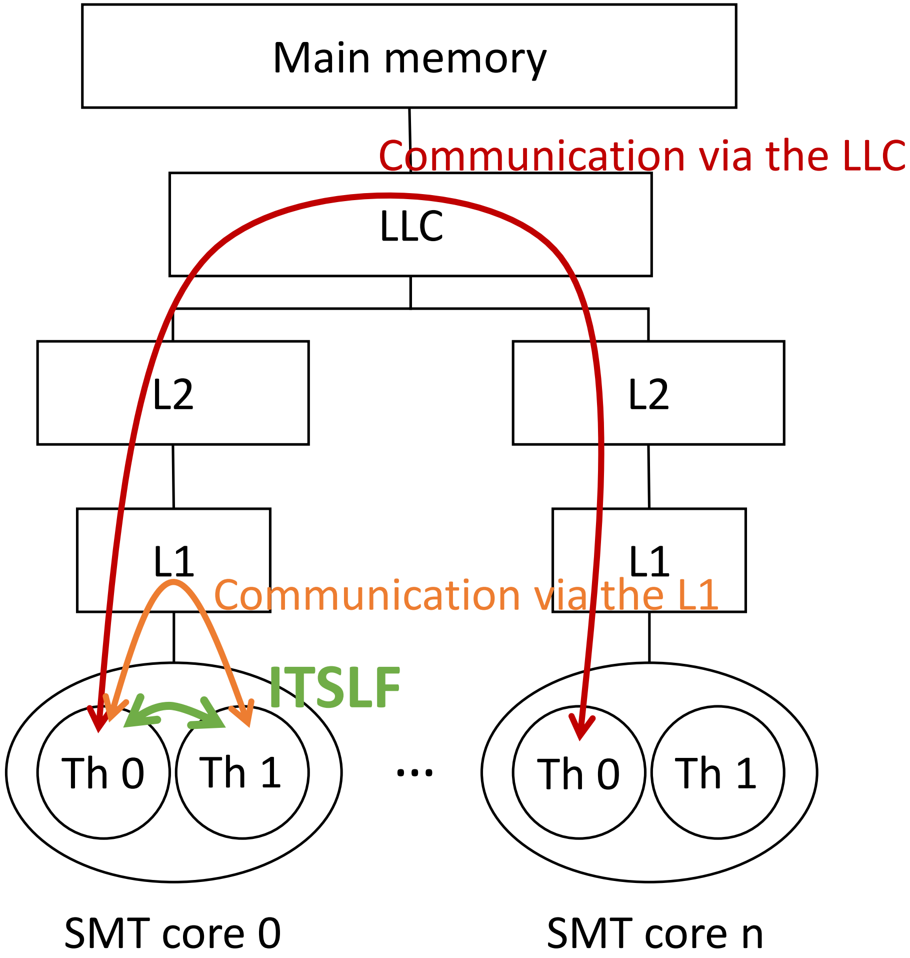


Introduction

- Fine-grain, synchronization-intensive workloads scale poorly.
  - The farthest the synchronization, the more expensive.

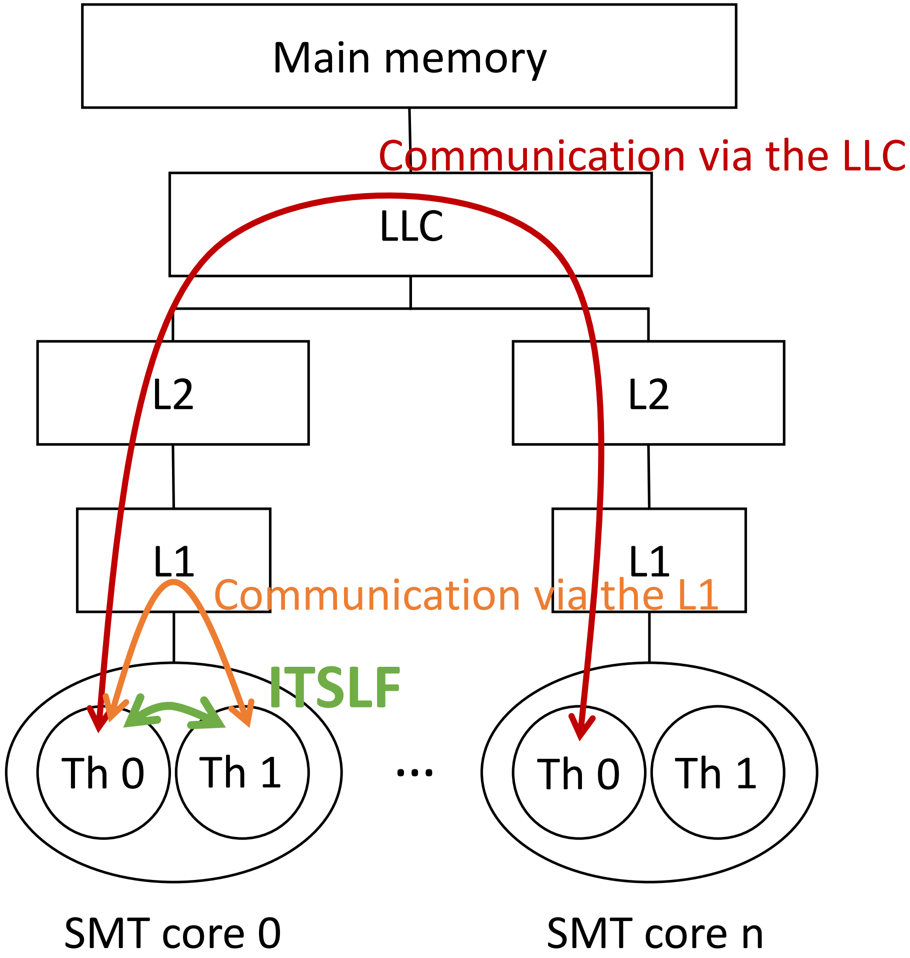
- Can we bring communication closer to the threads?
  - The first shared level between threads in an SMT is not the L1 cache but the SQ/SB.





Introduction

- Fine-grain, synchronization-intensive workloads scale poorly
  - The farthest the synchronization, the more expensive
- Can we bring communication closer to the threads?
  - The first shared level between threads in an SMT is not the L1 cache but the SQ/SB
- Implications for the memory models!
  - Violates coherence and consistency.



Introduction
What are our main contributions?

- **We propose Inter-Thread Store-to-Load Forwarding (ITSLF) for SMT architectures** and solve the problems that arise related to the memory model.

1. Determine the point when a store becomes locally visible to SMT threads.
2. Safeguard write serialization for same-address stores.
3. Efficiently maintain multi-copy atomicity (MCA).

Outline

- Introduction
- Background
- Issues and Solutions with ITSLF
- Experimental Evaluation
- Conclusion

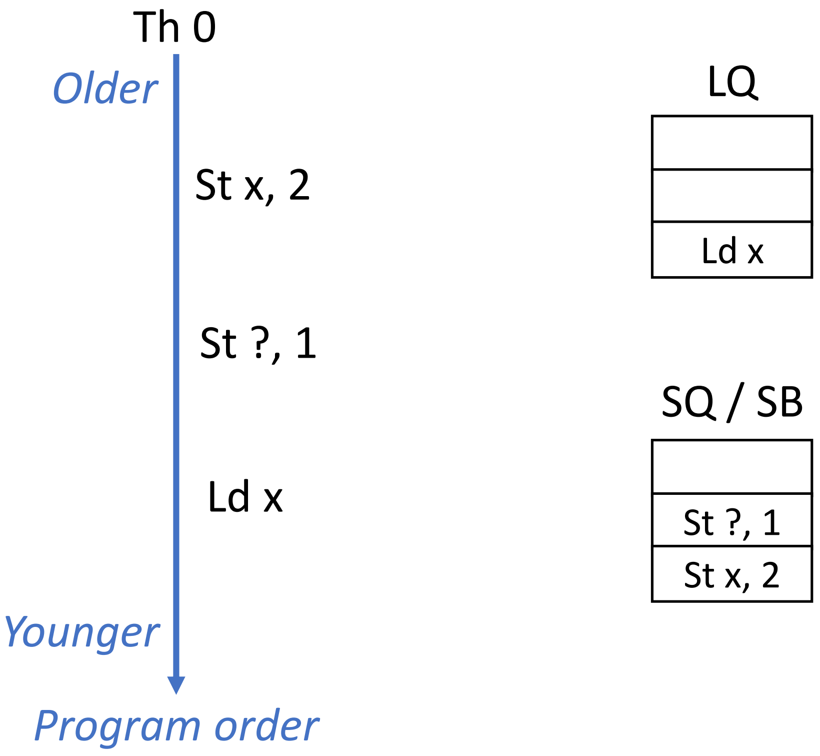
Background: Speculative support for memory ordering

- Memory operations are speculatively issued out-of-order.
- A correctness execution must respect:
  - Memory dependencies.
  - Load → Load ordering.

Background:
Speculative support for memory ordering

- Memory operations are speculatively issued out-of-order.

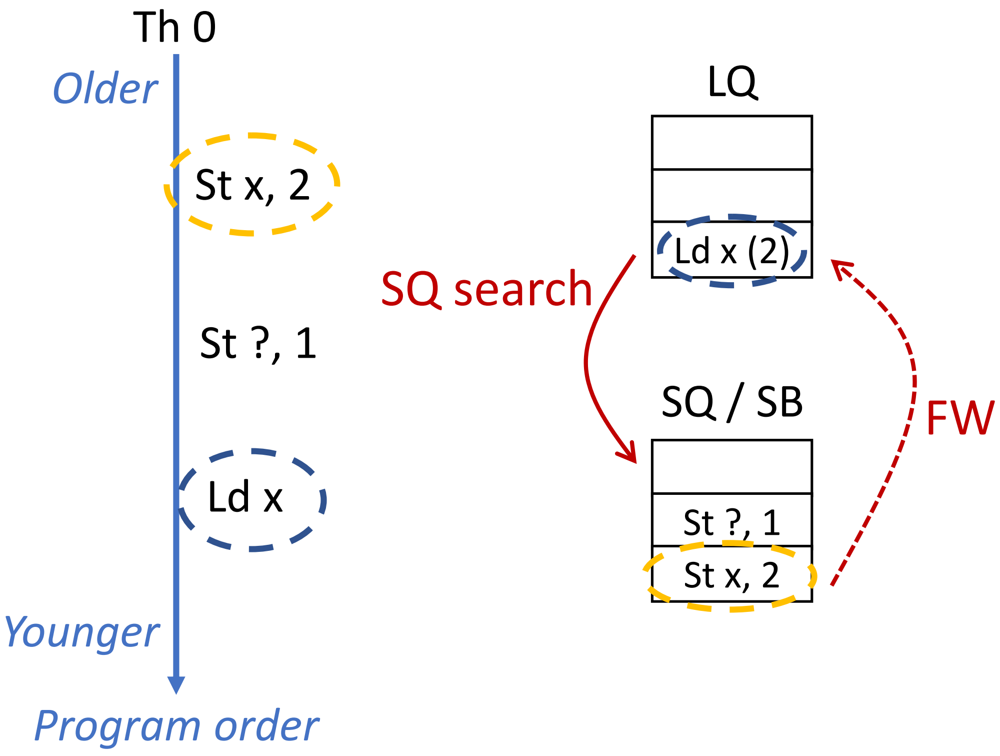
- A correctness execution must respect:
  - Memory dependencies.
  - Load → Load ordering.



Background:
Speculative support for memory ordering

- Memory operations are speculatively issued out-of-order.

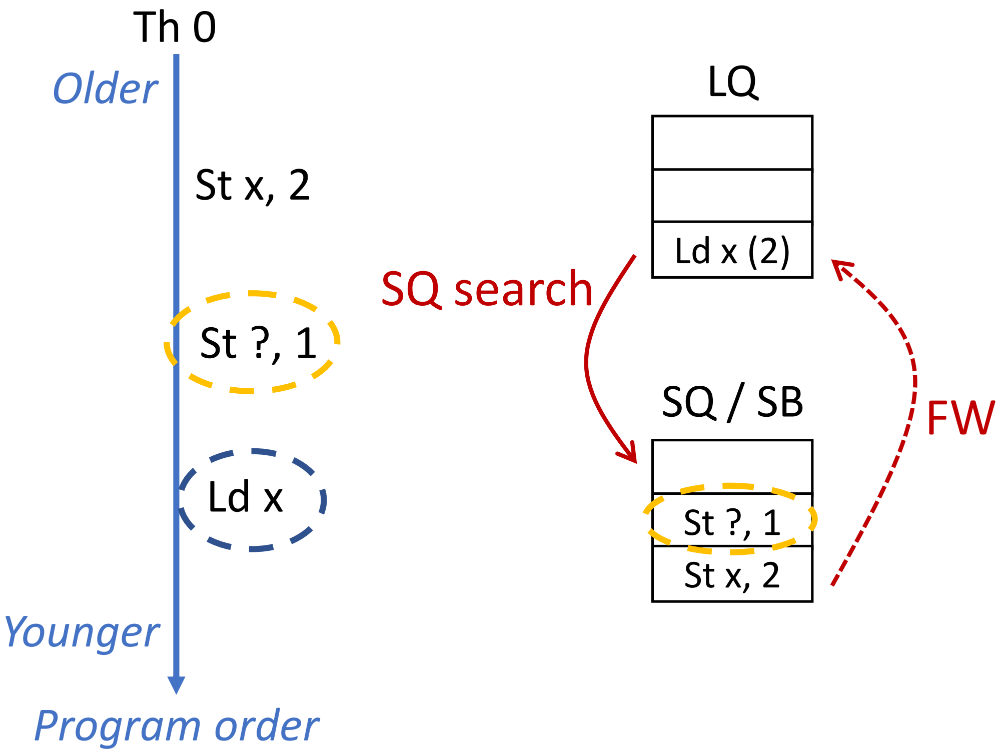
- A correctness execution must respect:
  - Memory dependencies.
    - Loads search the SB.
  - Load → Load ordering.



Background:
Speculative support for memory ordering

- Memory operations are speculatively issued out-of-order.

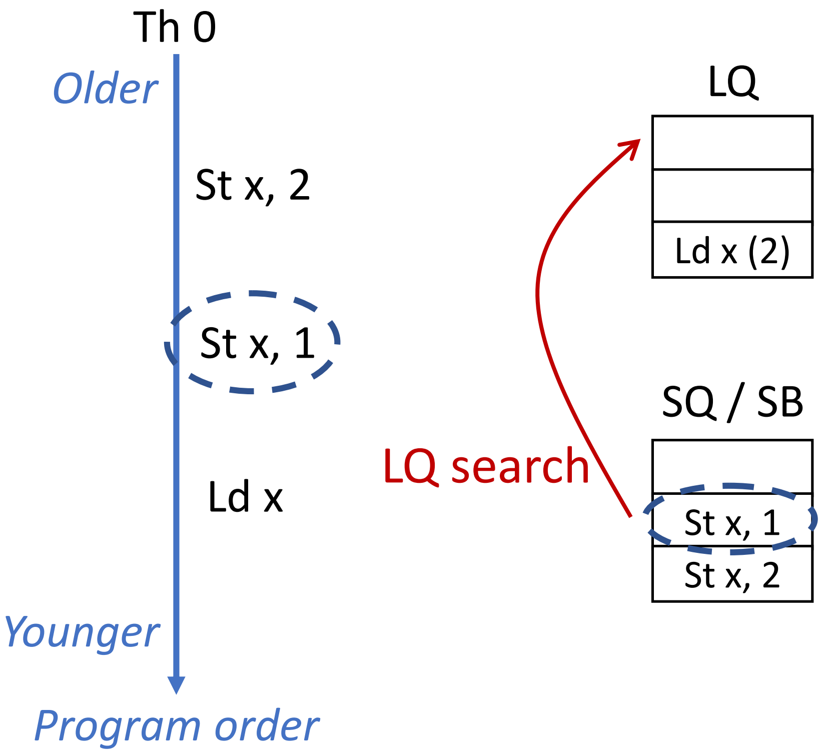
- A correctness execution must respect:
  - Memory dependencies.
    - Loads search the SB.
    - Stores search the LQ.
  - Load → Load ordering.



Background:
Speculative support for memory ordering

- Memory operations are speculatively issued out-of-order.

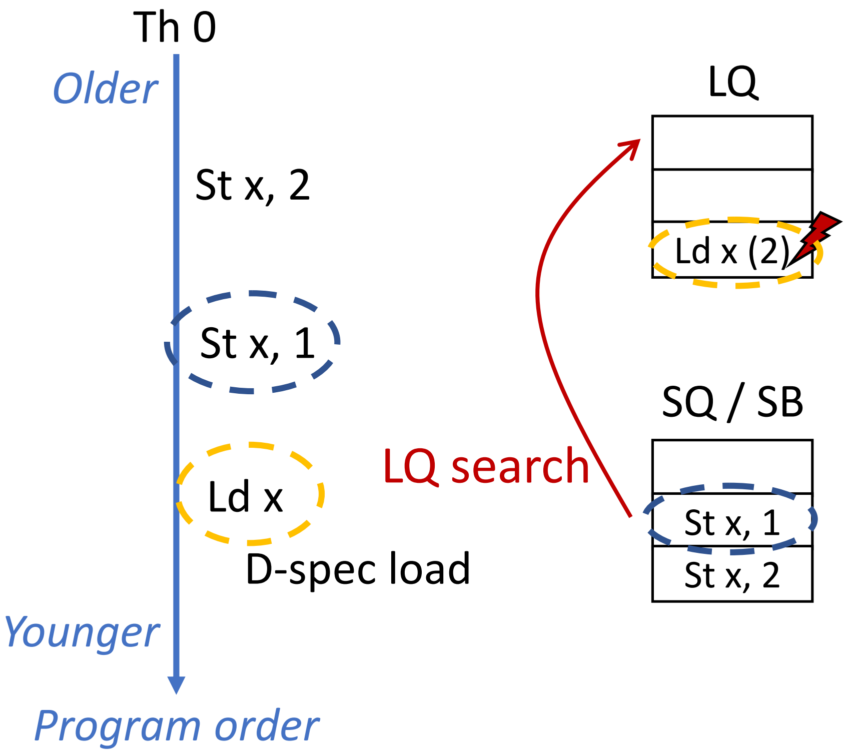
- A correctness execution must respect:
  - Memory dependencies.
    - Loads search the SB.
    - Stores search the LQ.
  - Load → Load ordering.



Background:
Speculative support for memory ordering

- Memory operations are speculatively issued out-of-order.

- A correctness execution must respect:
  - Memory dependencies.
    - Loads search the SB.
    - Stores search the LQ.
  - Load → Load ordering.



Background:
Speculative support for memory ordering

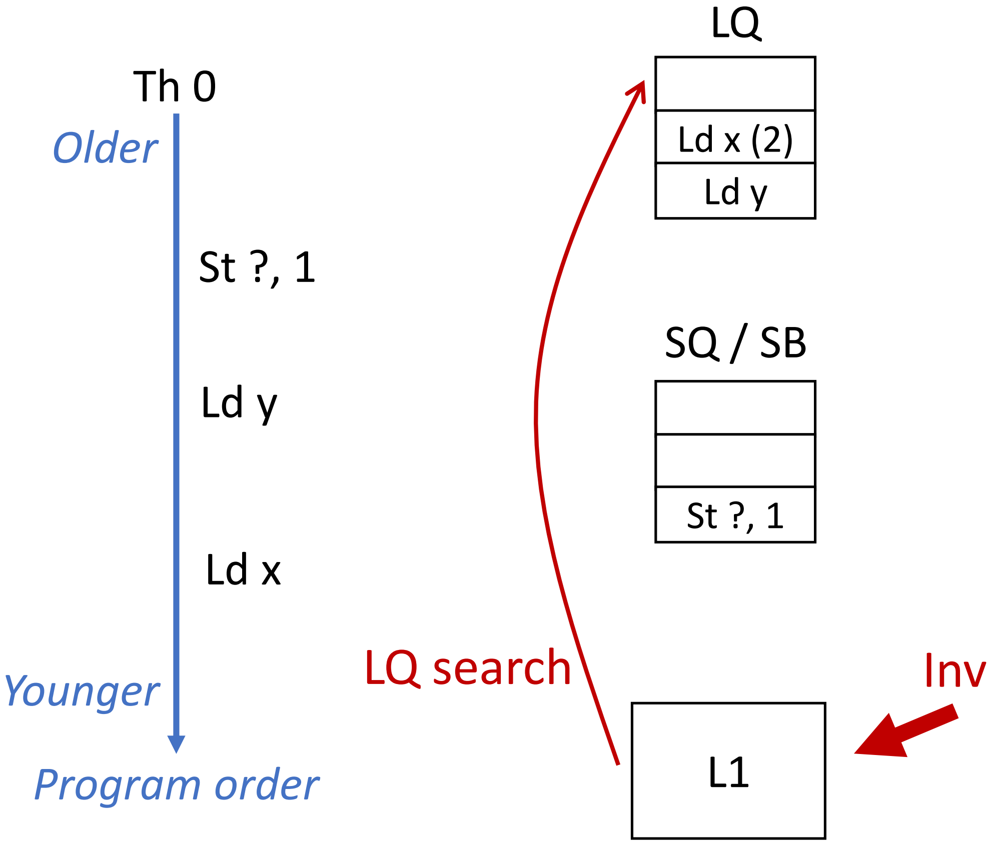
- Memory operations are speculatively issued out-of-order.

- A correctness execution must respect:
  - Memory dependencies.
    - Loads search the SB.
    - Stores search the LQ.
  - Load → Load ordering.

Background:
Speculative support for memory ordering

- Memory operations are speculatively issued out-of-order.

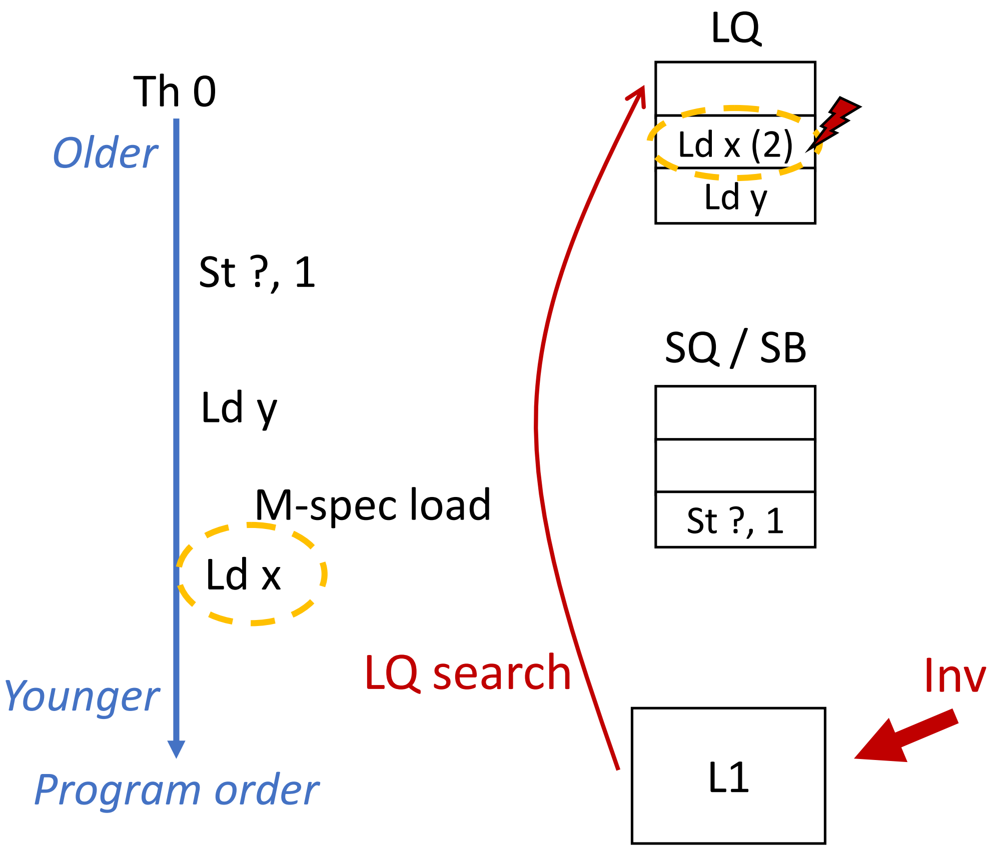
- A correctness execution must respect:
  - Memory dependencies.
    - Loads search the SB.
    - Stores search the LQ.
  - Load → Load ordering.
    - Invalidations search the LQ.



Background:
Speculative support for memory ordering

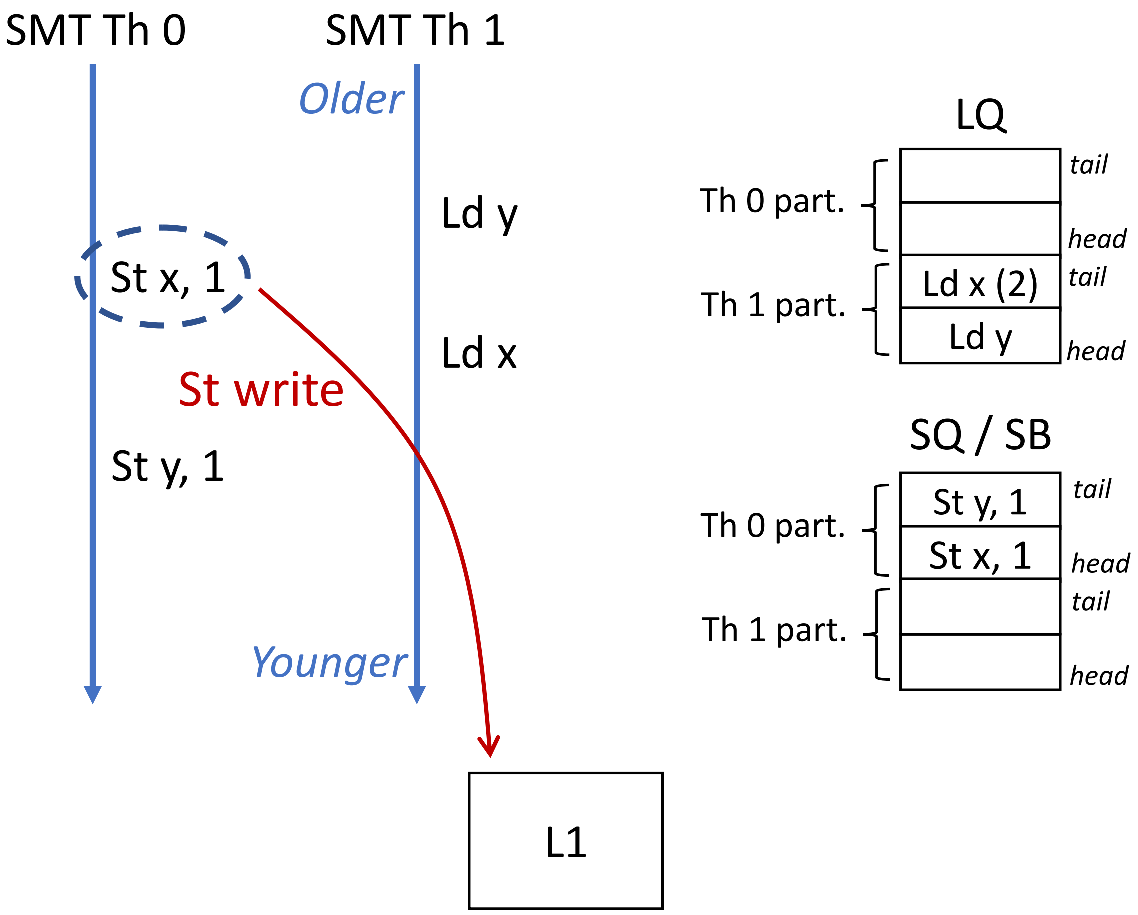
- Memory operations are speculatively issued out-of-order.

- A correctness execution must respect:
  - Memory dependencies.
    - Loads search the SB.
    - Stores search the LQ.
  - Load → Load ordering.
    - Invalidations search the LQ.



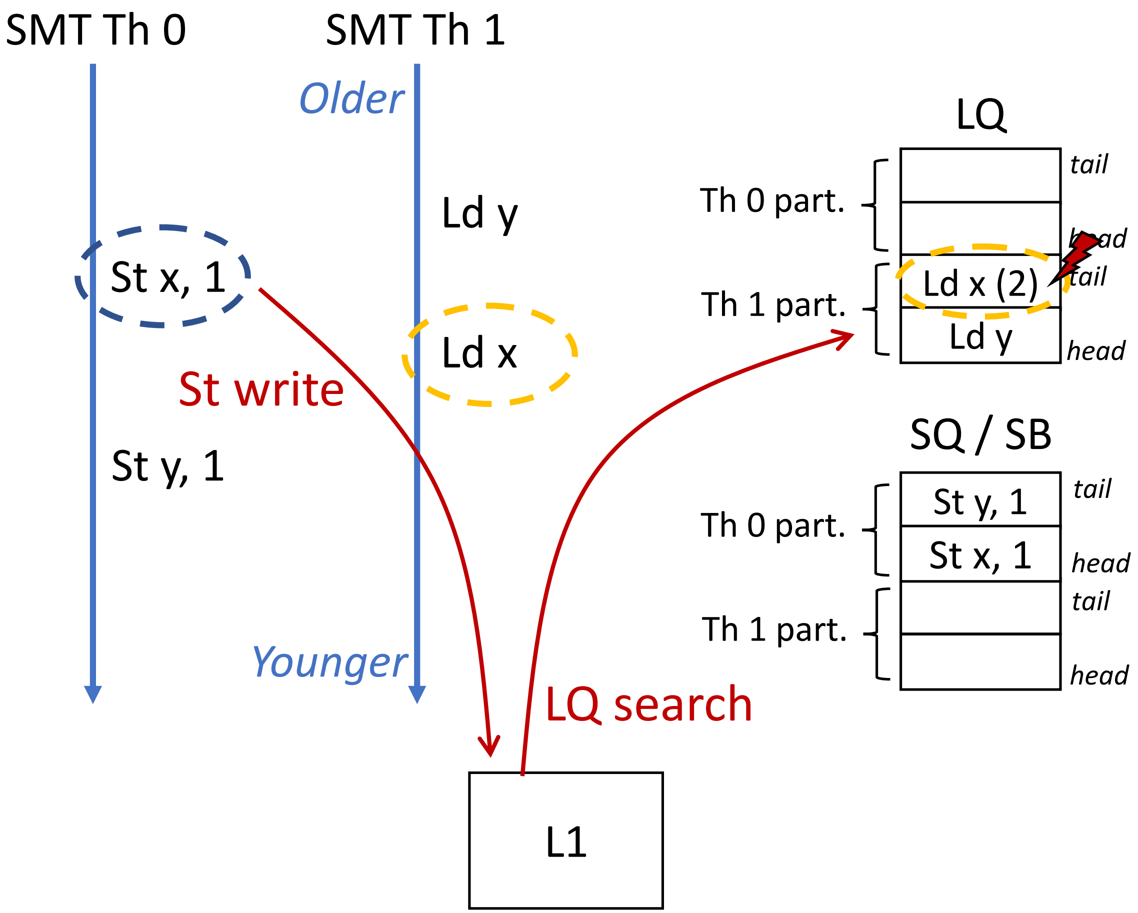
Background:
Speculative support for memory ordering in SMT

- Load → Load ordering violations could be exposed by stores from a thread running in the same SMT core.
  - Same-core threads share the state of cachelines in the L1.
  - No invalidation arrives to threads in a SMT core due to a store from a same-core thread.



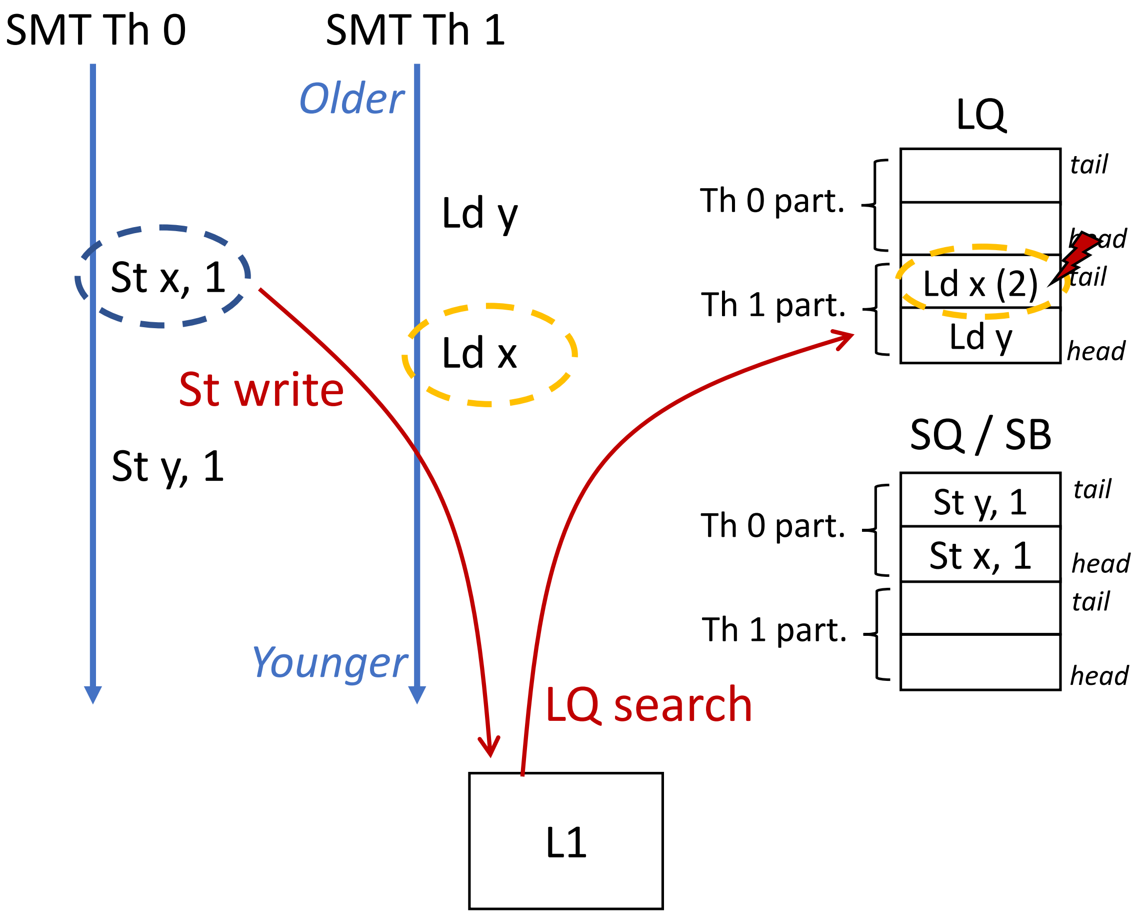
Background:
Speculative support for memory ordering in SMT

- Load → Load ordering violations could be exposed by stores from a thread running in the same SMT core.
  - Same-core threads share the state of cachelines in the L1.
  - No invalidation arrives to threads in a SMT core due to a store from a same-core thread.
  - Store search the LQs of the other threads in the same core when they write to memory.



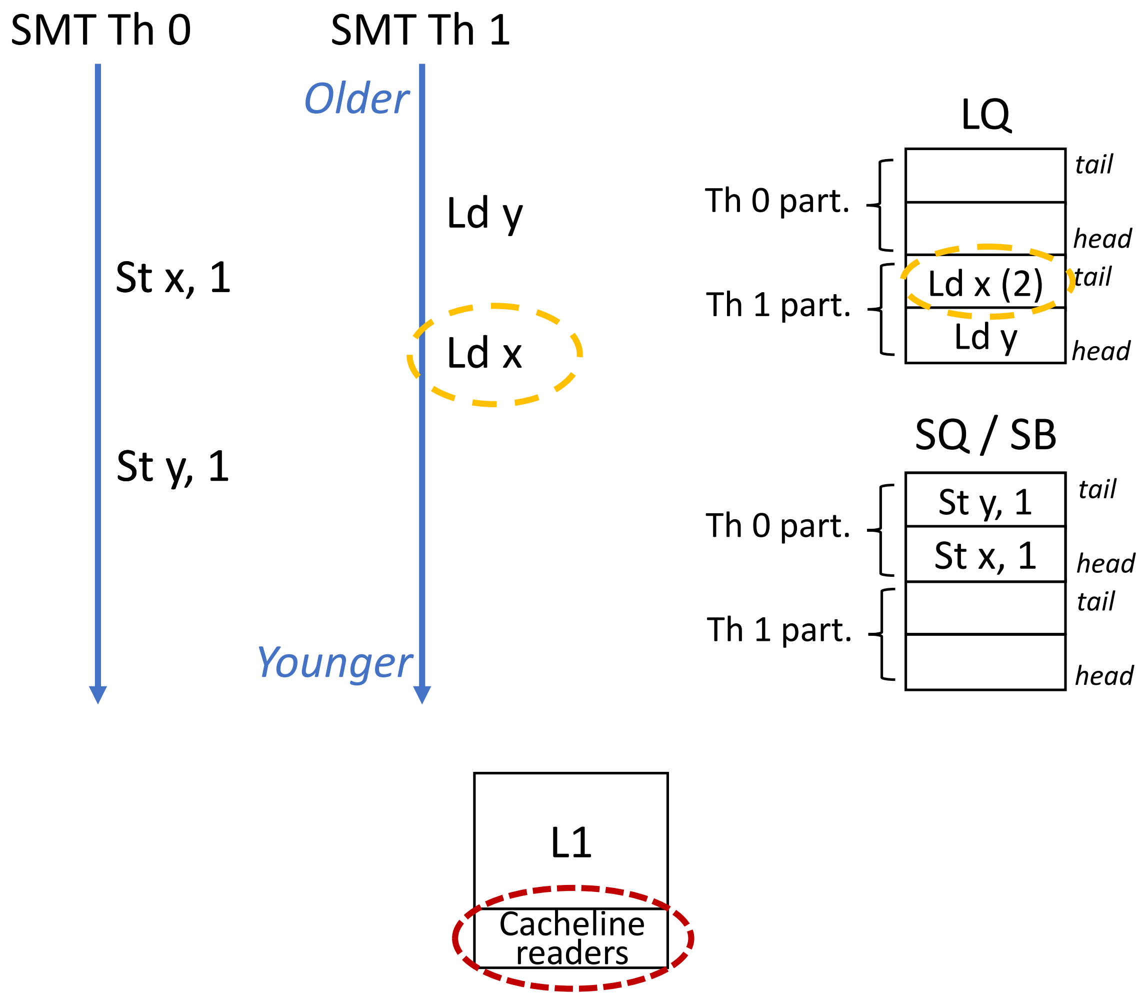
Background:
Speculative support for memory ordering in SMT

- Load → Load ordering violations could be exposed by stores from a thread running in the same SMT core.
  - Same-core threads share the state of cachelines in the L1.
  - No invalidation arrives to threads in a SMT core due to a store from a same-core thread.
  - Store search the LQs of the other threads in the same core when they write to memory.
  - Increases LQ snoop port contention.



Background:
Speculative support for memory ordering in SMT

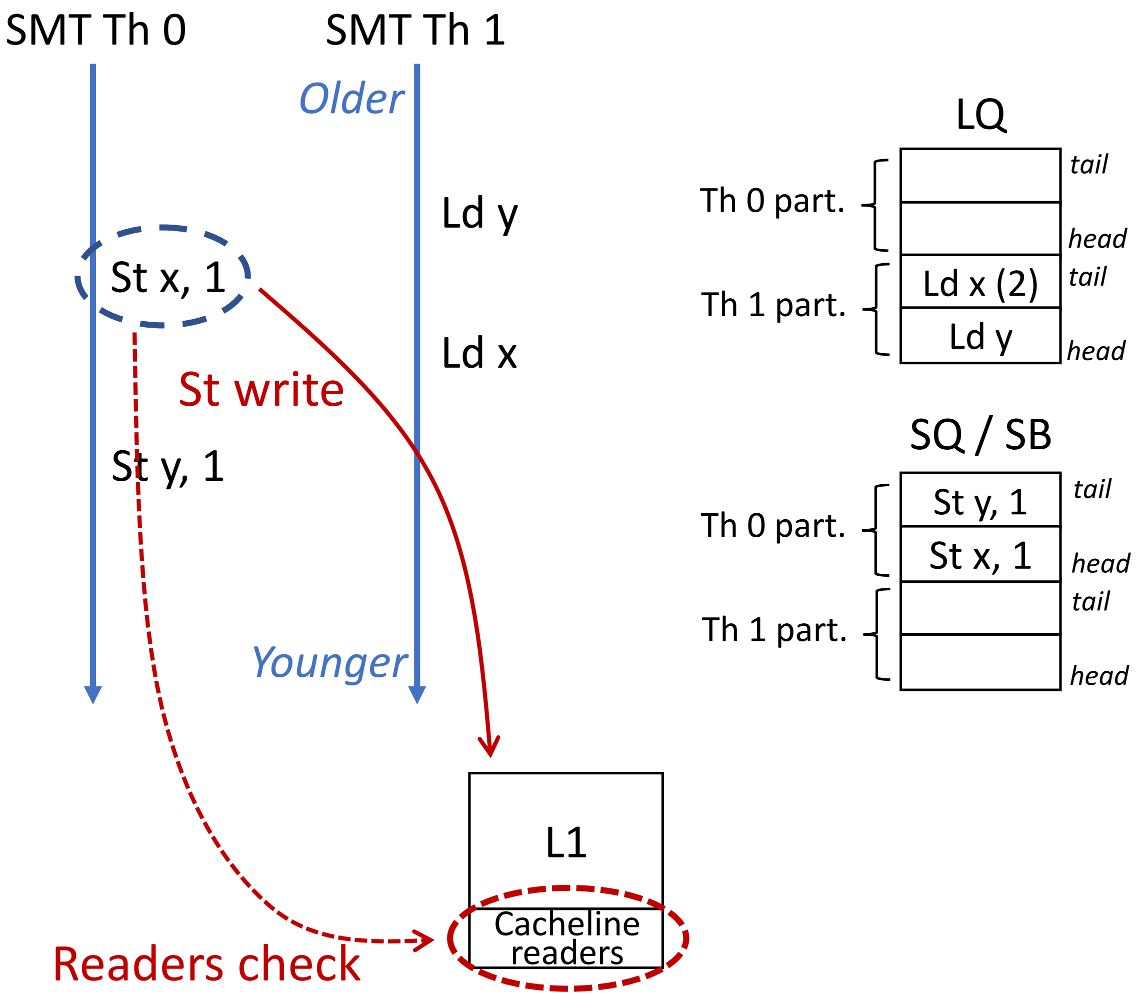
- LQ-search filtering optimization [1]: only the LQs of threads that read the cacheline need to be snooped.
  - Store cacheline readers in the L1.
  - Squashing is rare and thus, it reduces LQ snoop contention.



[1] Hilton and Roth at CAL’10

Background:
Speculative support for memory ordering in SMT

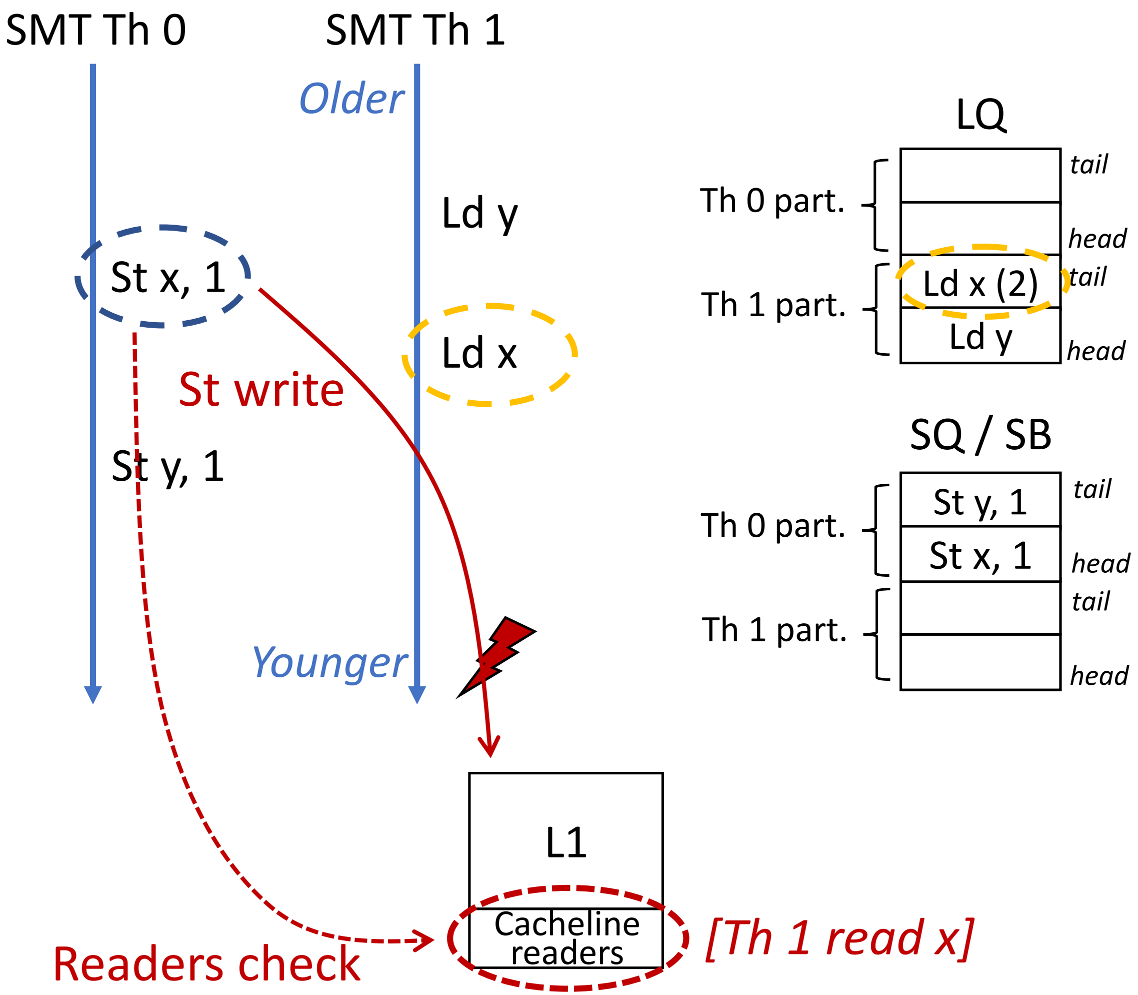
- LQ-search filtering optimization [1]: only the LQs of threads that read the cacheline need to be snooped.
  - Store cacheline readers in the L1.
  - Squashing is rare and thus, it reduces LQ snoop contention.



[1] Hilton and Roth at CAL'10

Background:
Speculative support for memory ordering in SMT

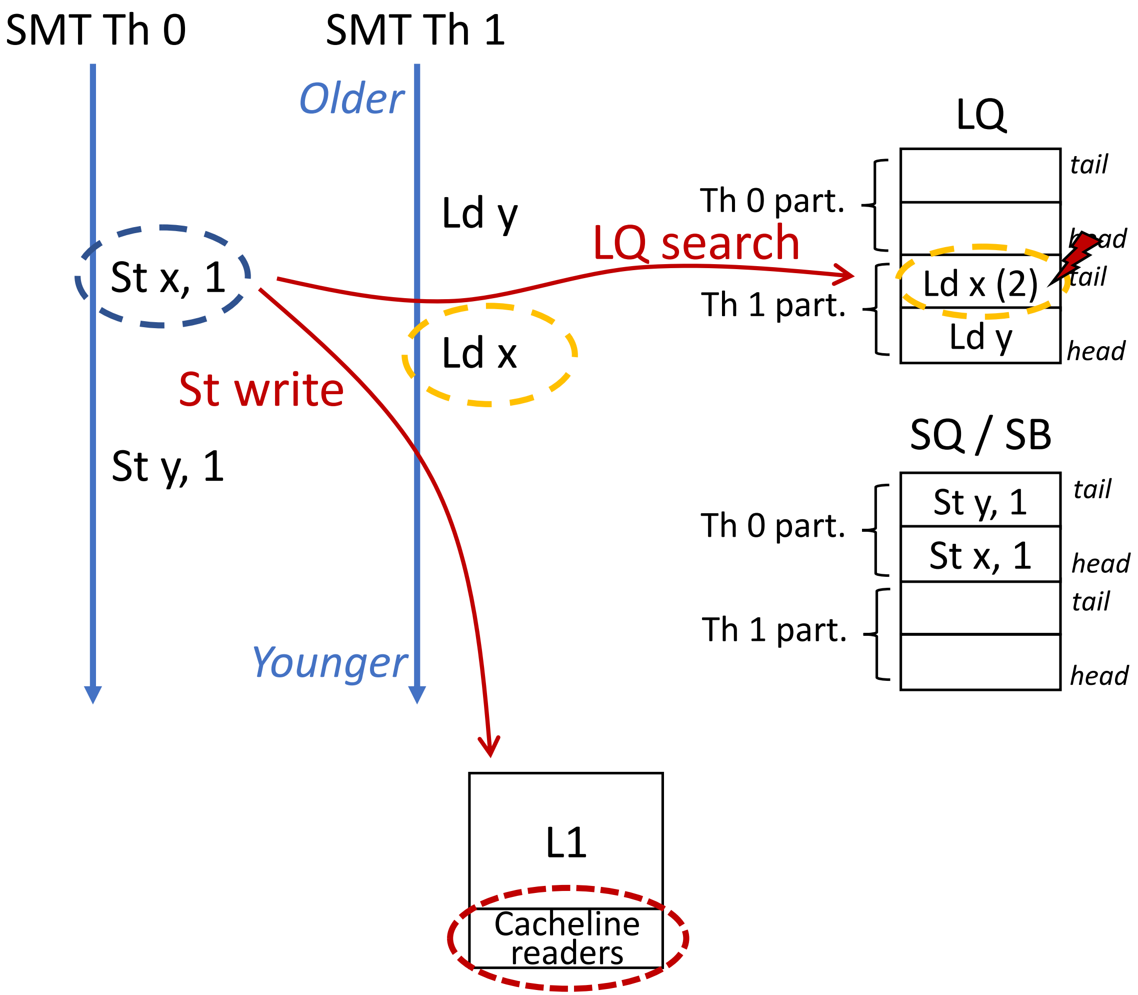
- LQ-search filtering optimization [1]: only the LQs of threads that read the cacheline need to be snooped.
  - Store cacheline readers in the L1.
  - Squashing is rare and thus, it reduces LQ snoop contention.



[1] Hilton and Roth at CAL'10

Background:
Speculative support for memory ordering in SMT

- LQ-search filtering optimization [1]: only the LQs of threads that read the cacheline need to be snooped.
  - Store cacheline readers in the L1.
  - Squashing is rare and thus, it reduces LQ snoop contention.
  - Doubles the write latency when the snoop if required.



[1] Hilton and Roth at CAL’10

Outline

- Introduction
- Background
- Issues and Solutions with ITSLF
- Experimental Evaluation
- Conclusion

Issues and Solutions with ITSLF

- *Inter-thread store-to-load-forwarding* could be enabled by not restricting the SQ/SB search to the same thread.
- Exposes store values to some threads before they are inserted in global order and breaks:

1. Coherence and TSO
2. Write serialization
3. Multi-Copy Atomicity

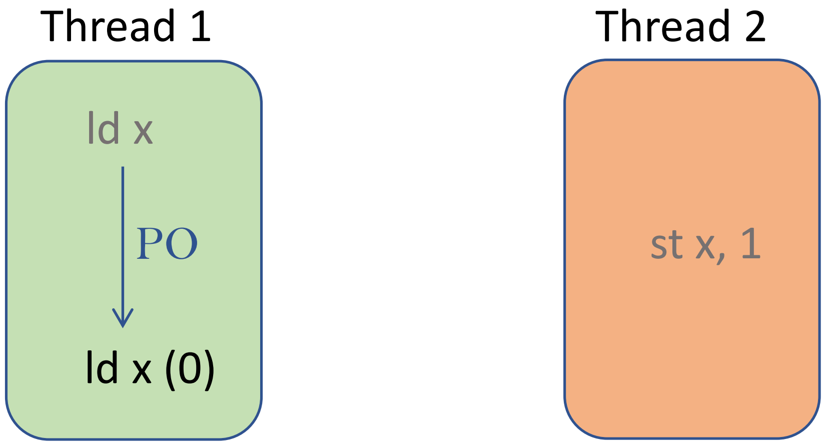
Issues and Solutions with ITSLF

- *Inter-thread store-to-load-forwarding* could be enabled by not restricting the SQ/SB search to the same thread.
- Exposes store values to some threads before they are inserted in global order and breaks:

1. **Coherence and TSO → Point of Local Visibility**
2. Write serialization
3. Multi-Copy Atomicity

ITSLF: Point of local visibility

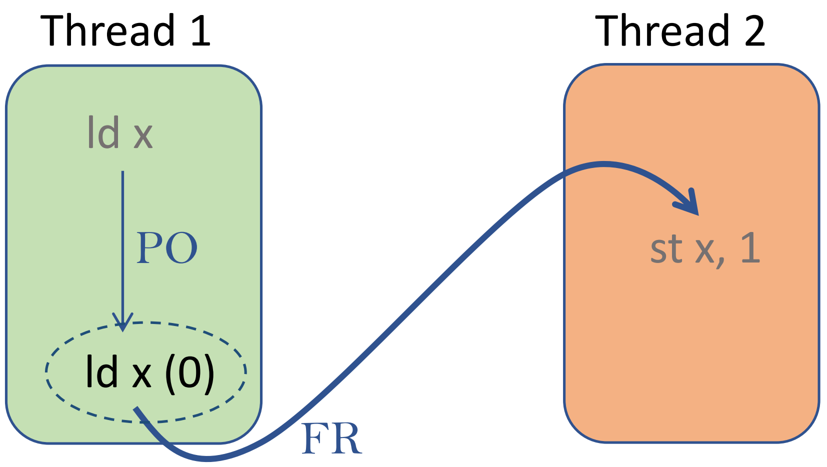
Initially: x = 0



PO: program order

ITSLF: Point of local visibility

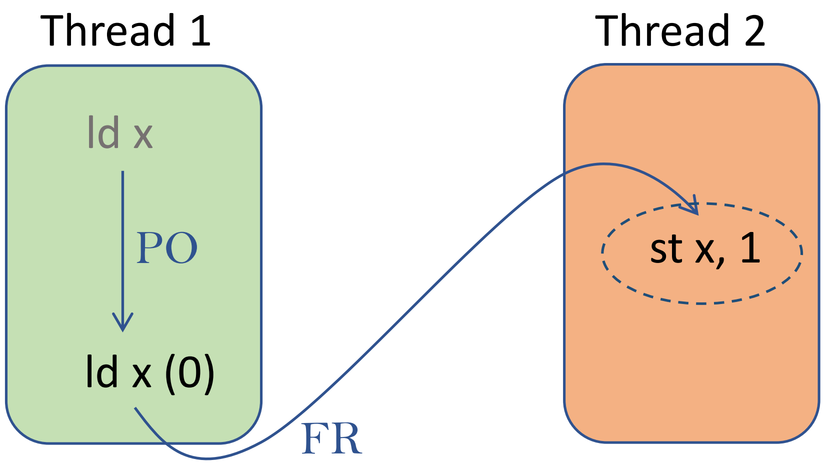
Initially: x = 0



PO: program order
FR: from-read

ITSLF: Point of local visibility

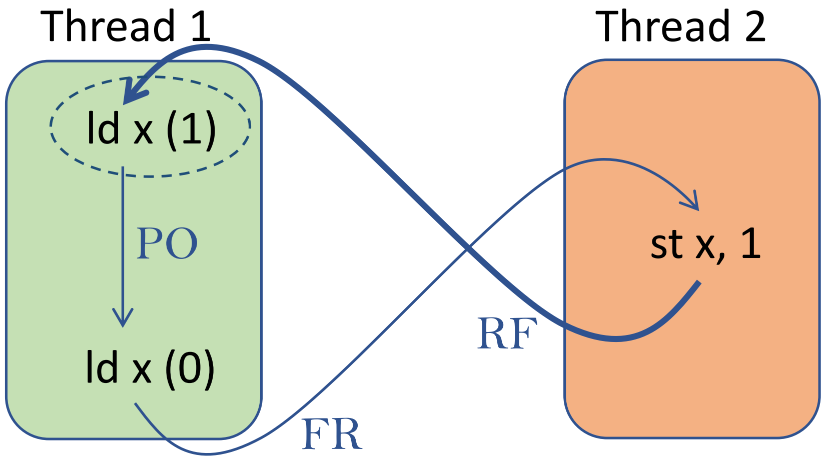
Initially: x = 0



PO: program order
FR: from-read

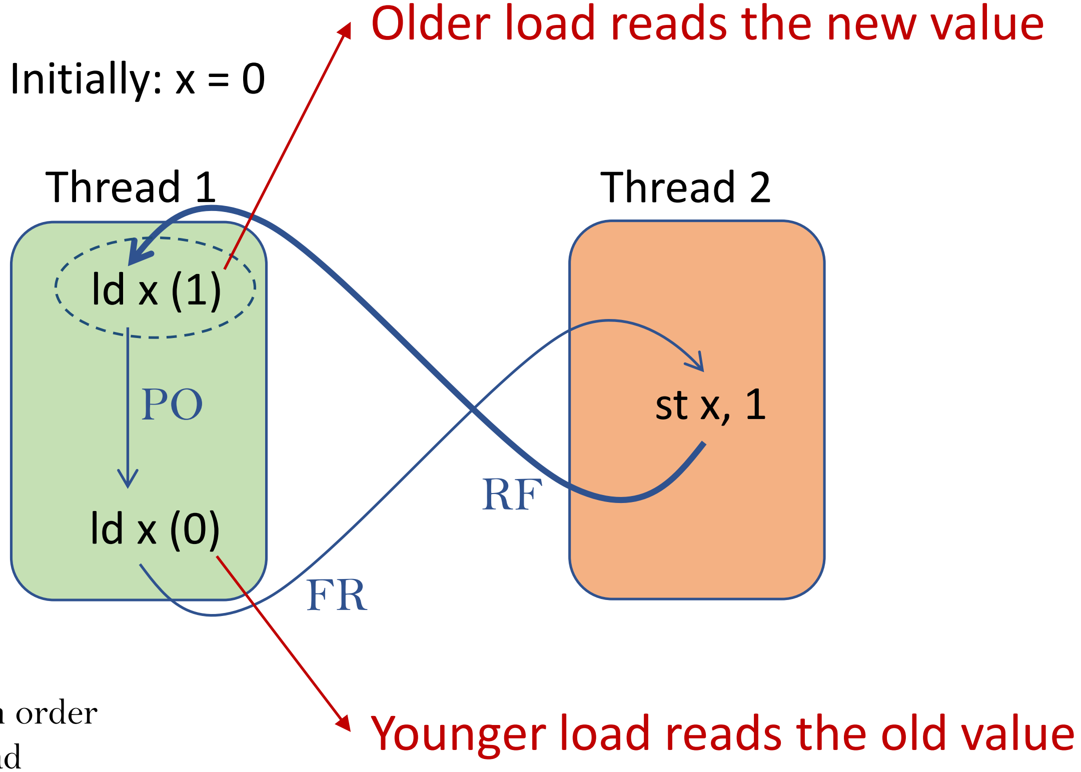
ITSLF: Point of local visibility

Initially: x = 0



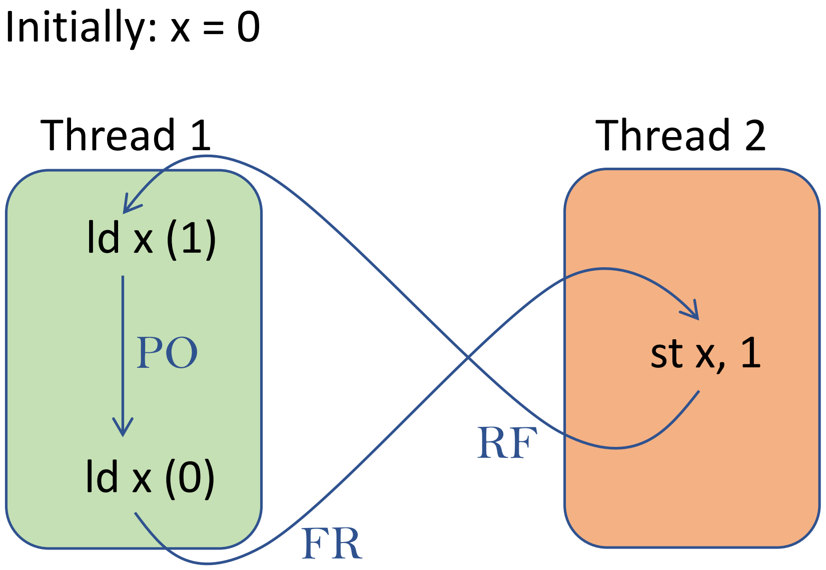
PO: program order
FR: from-read
RF: read-from

ITSLF: Point of local visibility


Older load reads the new value
Initially: x = 0
Thread 1
Thread 2
ld x (1)
PO
ld x (0)
st x, 1
RF
FR
Younger load reads the old value

PO: program order
FR: from-read
RF: read-from

ITSLF: Point of local visibility



PO: program order
FR: from-read
RF: read-from

ITSLF solution

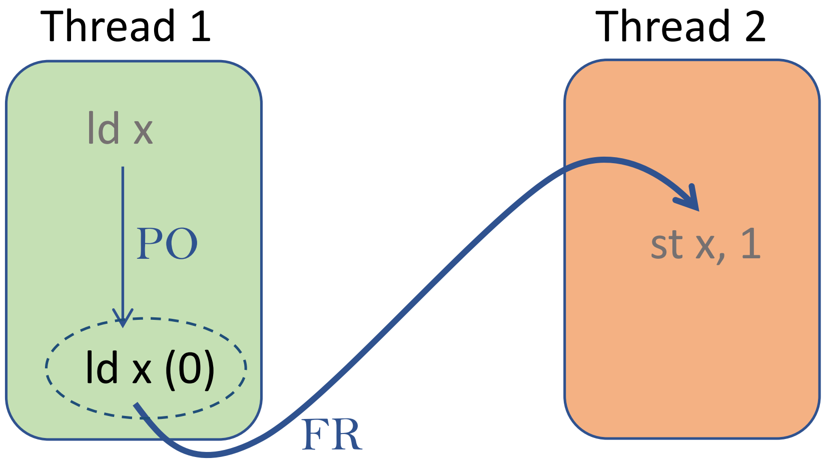
Stores become visible when they become non-speculative. At that point, they:

i) squash any matching M-speculative load in all other SMT threads.

ii) can forward its data to loads of other SMT threads.

ITSLF: Point of local visibility

Initially: x = 0



PO: program order
FR: from-read
RF: read-from

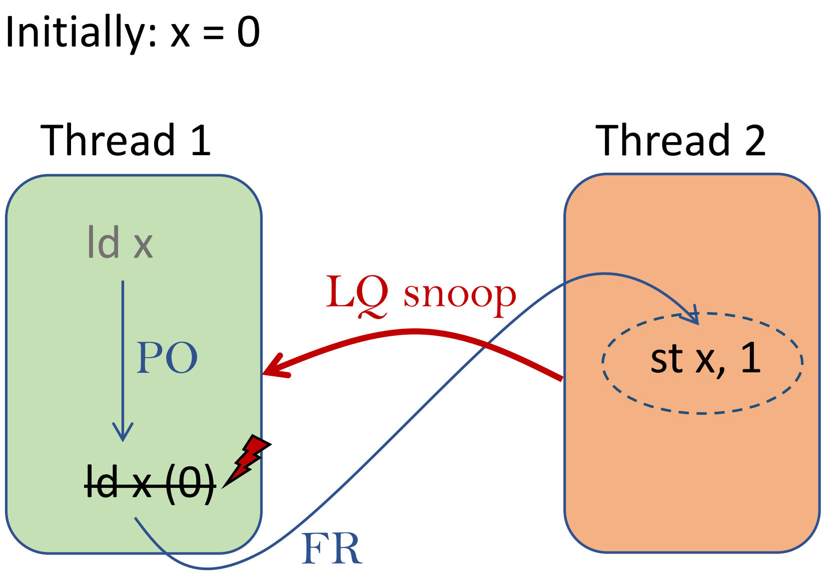
ITSLF solution

Stores become visible when they become non-speculative. At that point, they:

i) squash any matching M-speculative load in all other SMT threads.

ii) can forward its data to loads of other SMT threads.

ITSLF: Point of local visibility



PO: program order
FR: from-read
RF: read-from

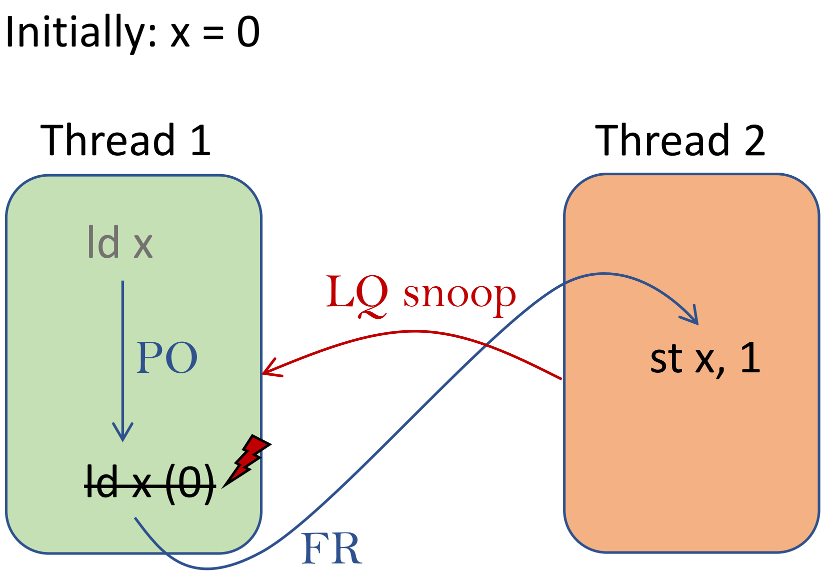
ITSLF solution

Stores become visible when they become non-speculative. At that point, they:

i) squash any matching M-speculative load in all other SMT threads.

ii) can forward its data to loads of other SMT threads.

ITSLF: Point of local visibility



PO: program order
FR: from-read
RF: read-from

ITSLF solution

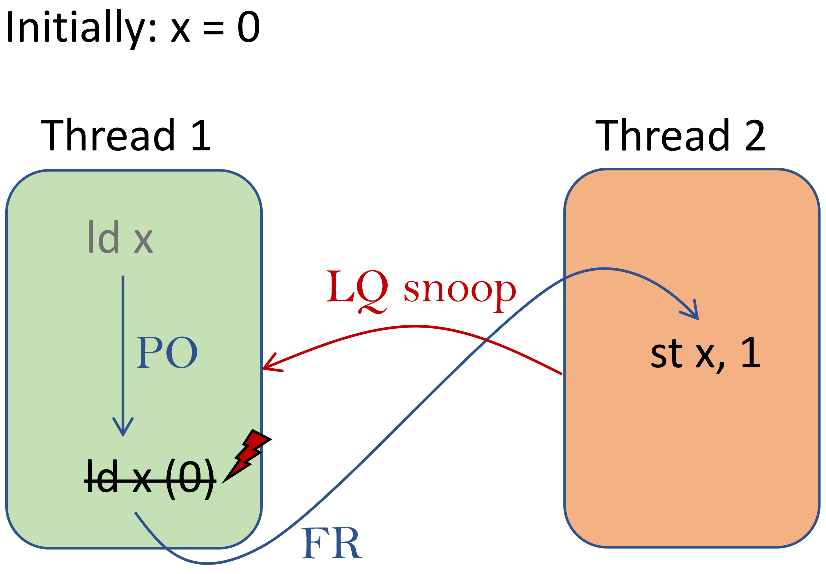
Stores become visible when they become non-speculative. At that point, they:

i) squash any matching M-speculative load in all other SMT threads.

ii) can forward its data to loads of other SMT threads.

ITSLF combines the same-thread LQ search and other-threads LQ search into a single LQ snoop.

ITSLF: Point of local visibility



PO: program order
FR: from-read
RF: read-from

ITSLF solution

Stores become visible when they become non-speculative. At that point, they:

i) squash any matching M-speculative load in all other SMT threads.

ii) can forward its data to loads of other SMT threads.

ITSLF combines the same-thread LQ search and other-threads LQ search into a single LQ snoop.

Cost

Requires support to determine when stores become non-speculative (et al. at . ISCA’19)

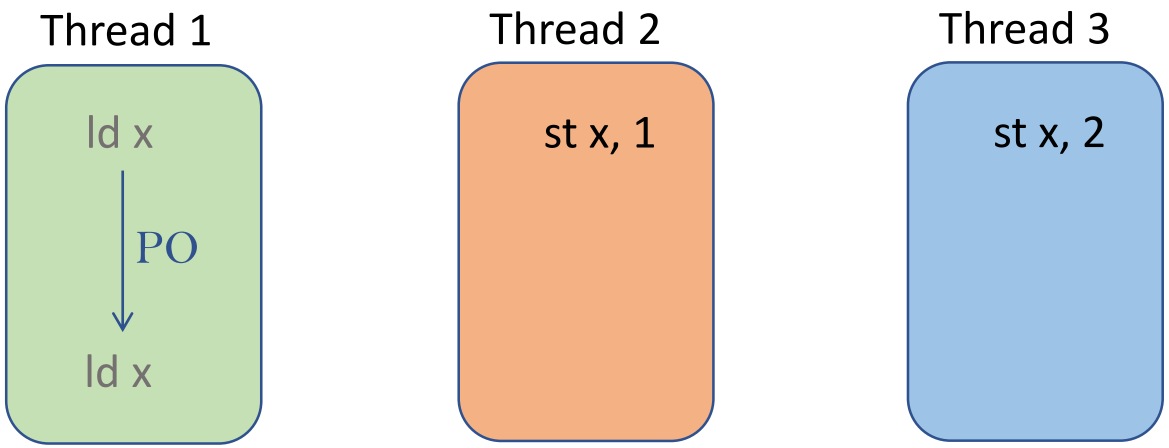
Issues and Solutions with ITSLF

- *Inter-thread store-to-load-forwarding* could be enabled by not restricting the SQ/SB search to the same thread.
- Exposes store values to some threads before they are inserted in global order and breaks:

1. Coherence and TSO → Point of Local Visibility
2. **Write serialization → Local Store Order**
3. Multi-Copy Atomicity

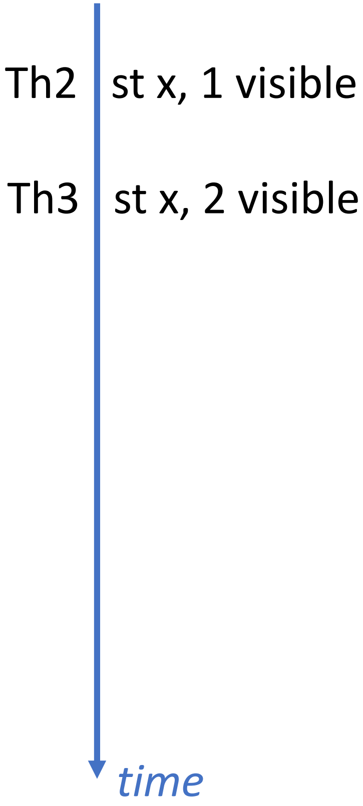
ITSLF: Local Store Order

Initially: x = 0



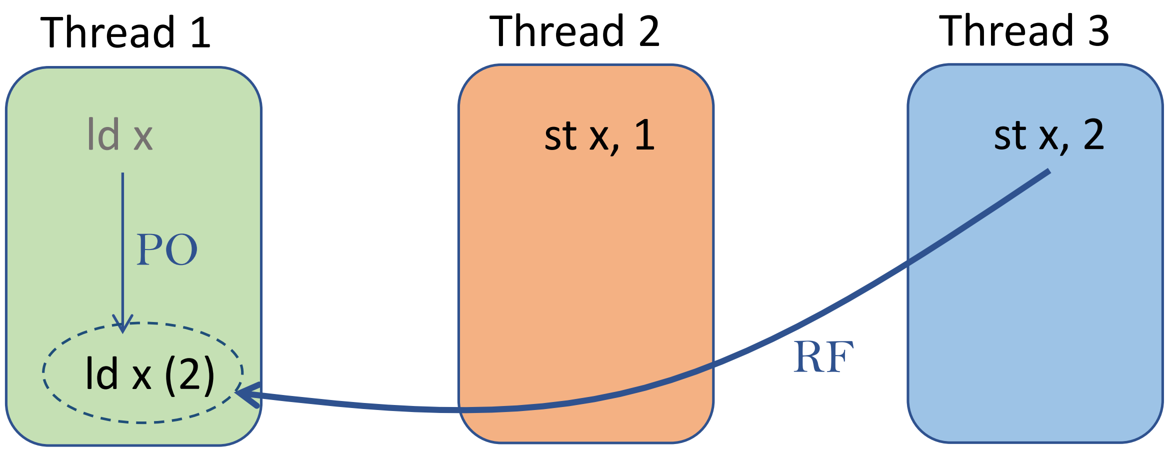
PO: program order
FR: from-read
RF: read-from

Memory
x = 0



ITSLF: Local Store Order

Initially: x = 0



Thread 1

ld x

PO

ld x (2)

Thread 2

st x, 1

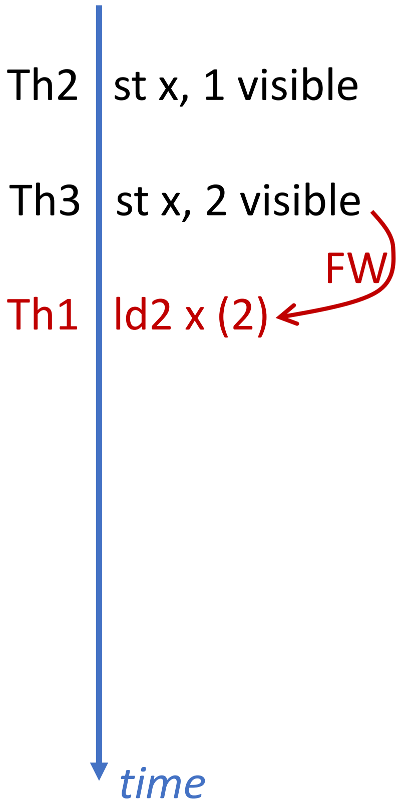
Thread 3

st x, 2

RF

PO: program order
FR: from-read
RF: read-from

Memory
x = 0



Th2 | st x, 1 visible

Th3 | st x, 2 visible

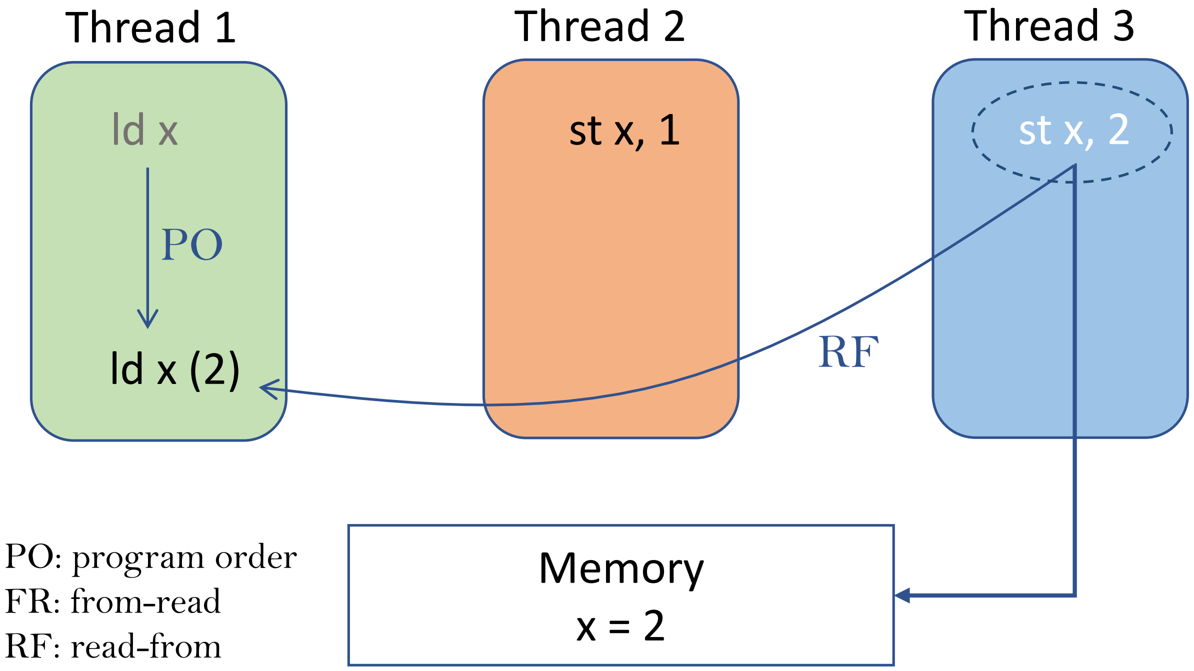
FW

Th1 | ld2 x (2)

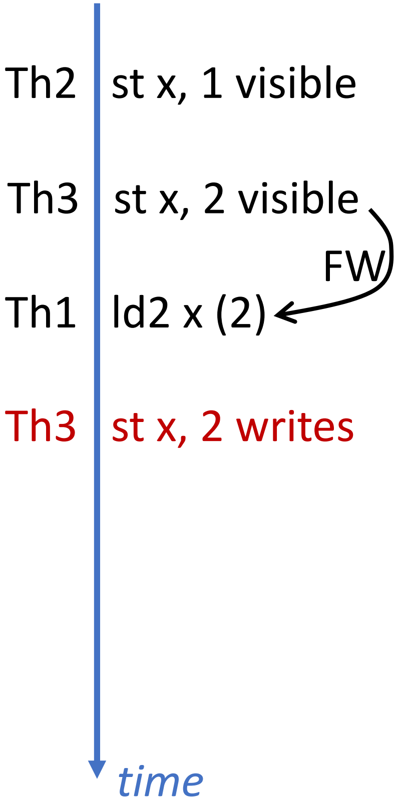
time

ITSLF: Local Store Order

Initially: x = 0

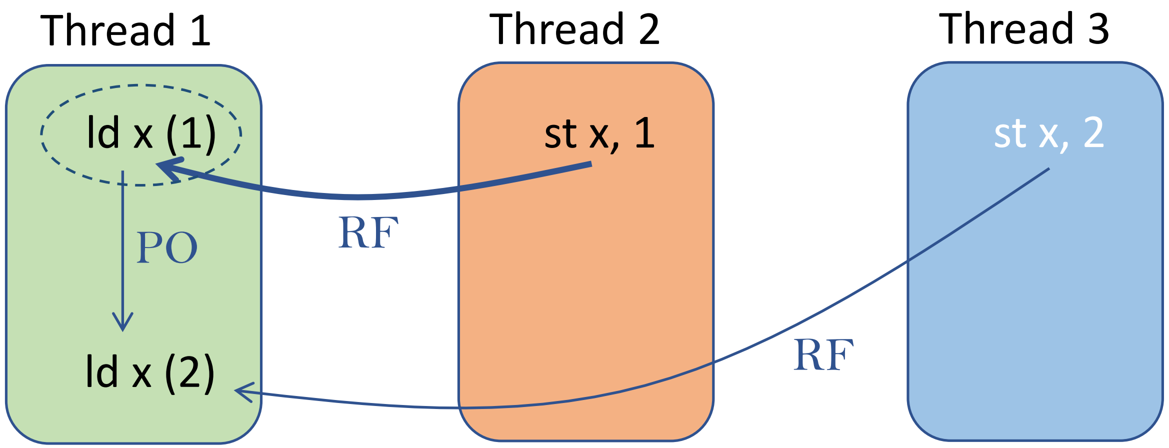


PO: program order
FR: from-read
RF: read-from



ITSLF: Local Store Order

Initially: x = 0


Thread 1

ld x (1)

PO

ld x (2)

Thread 2

st x, 1

Thread 3

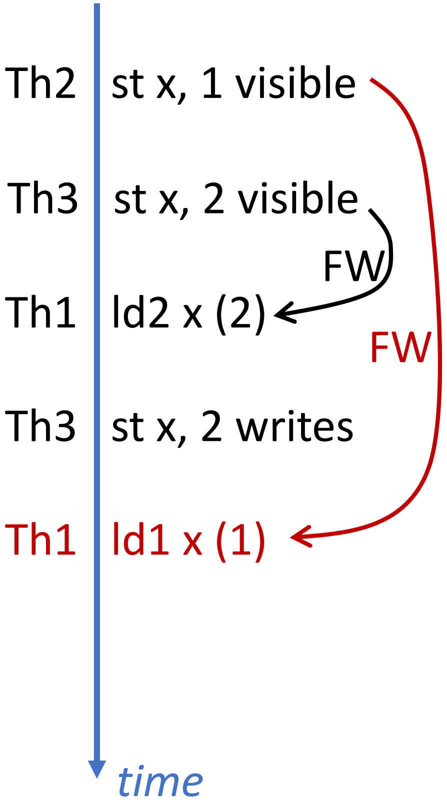
st x, 2

RF

RF

PO: program order
FR: from-read
RF: read-from

Memory
x = 2


Th2 st x, 1 visible

Th3 st x, 2 visible

FW

Th1 ld2 x (2)

FW

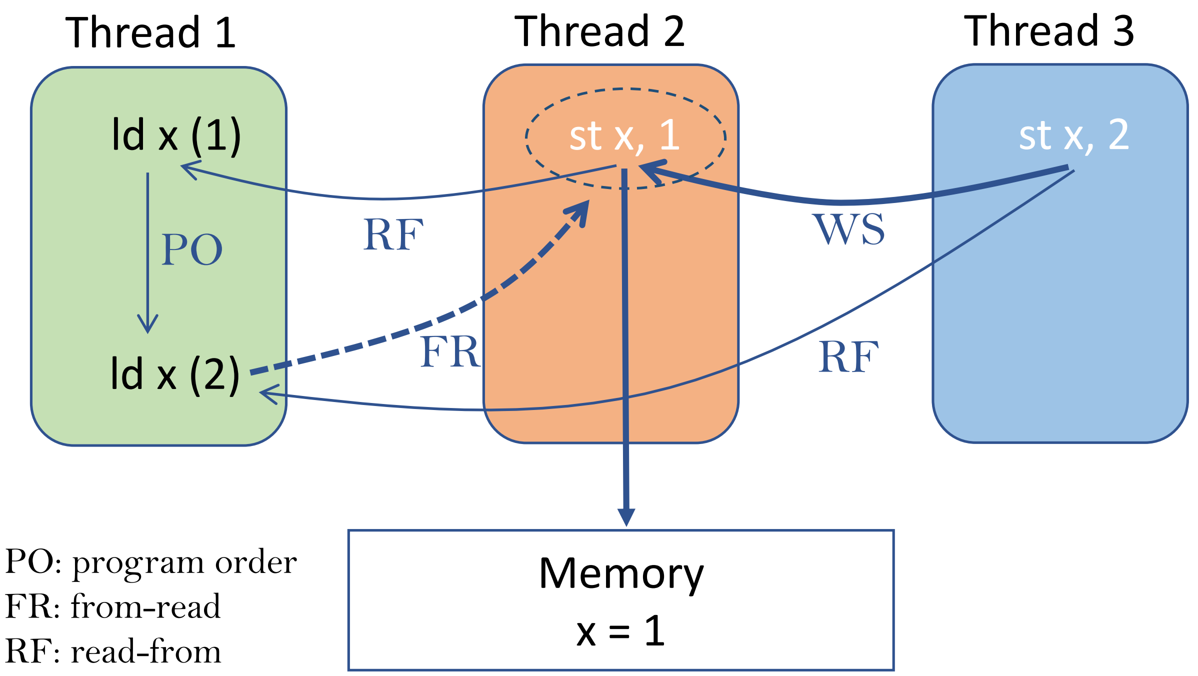
Th3 st x, 2 writes

Th1 ld1 x (1)

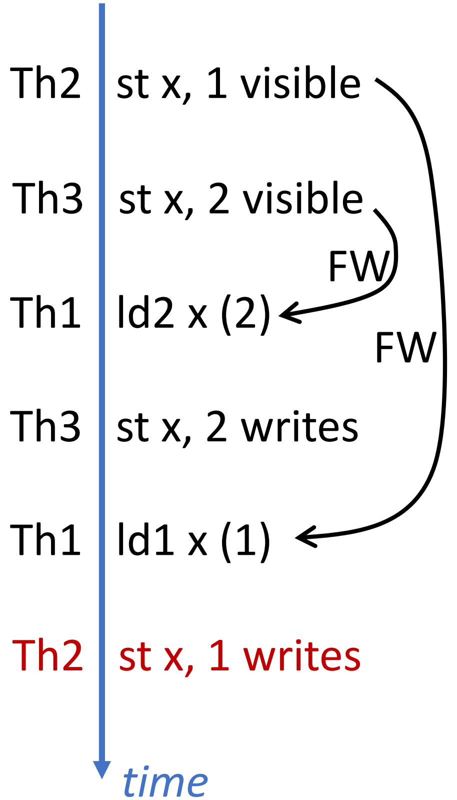
time

ITSLF: Local Store Order

Initially: x = 0

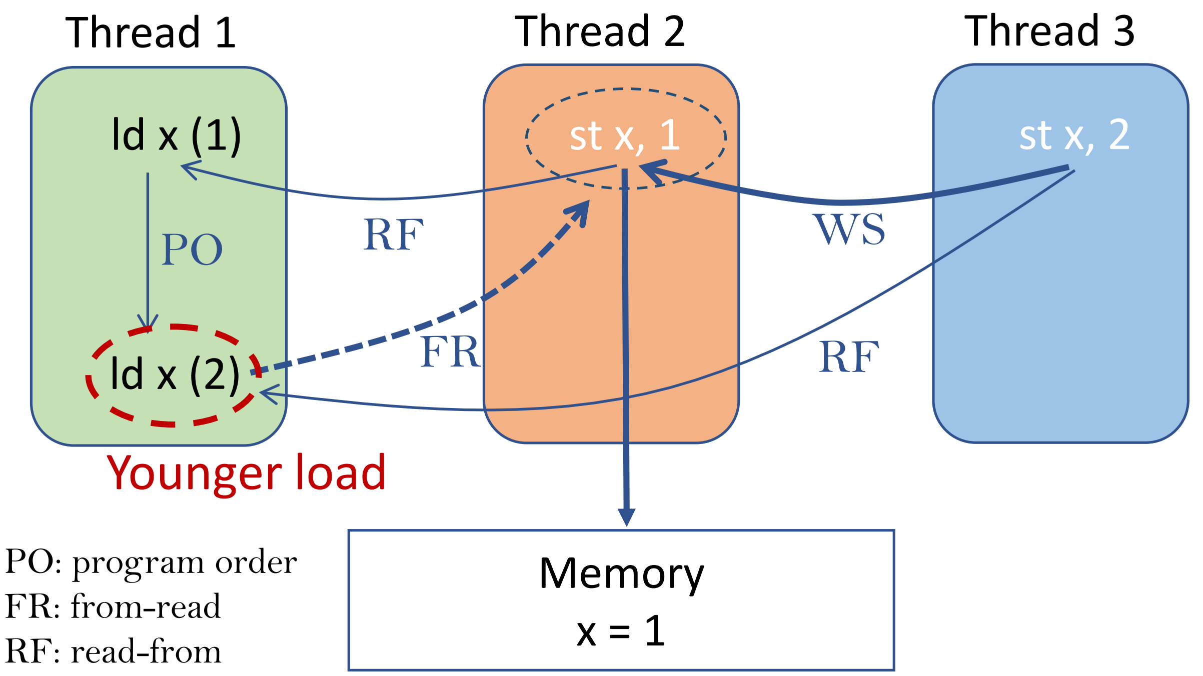


PO: program order
FR: from-read
RF: read-from
WS: write serialization

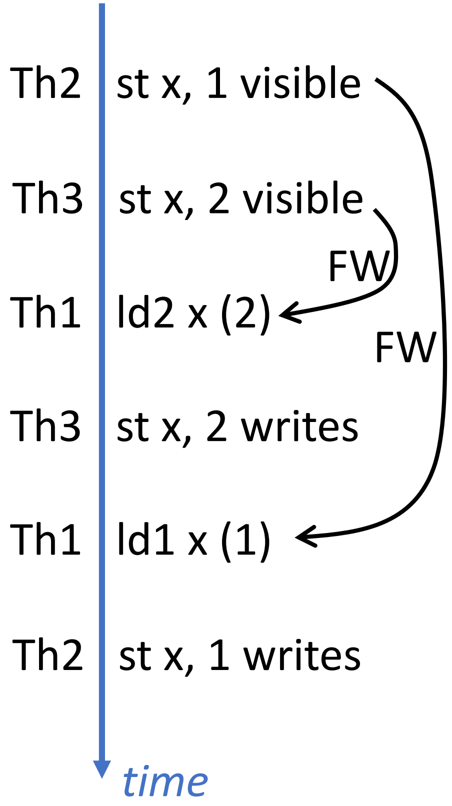


ITSLF: Local Store Order

Initially: x = 0

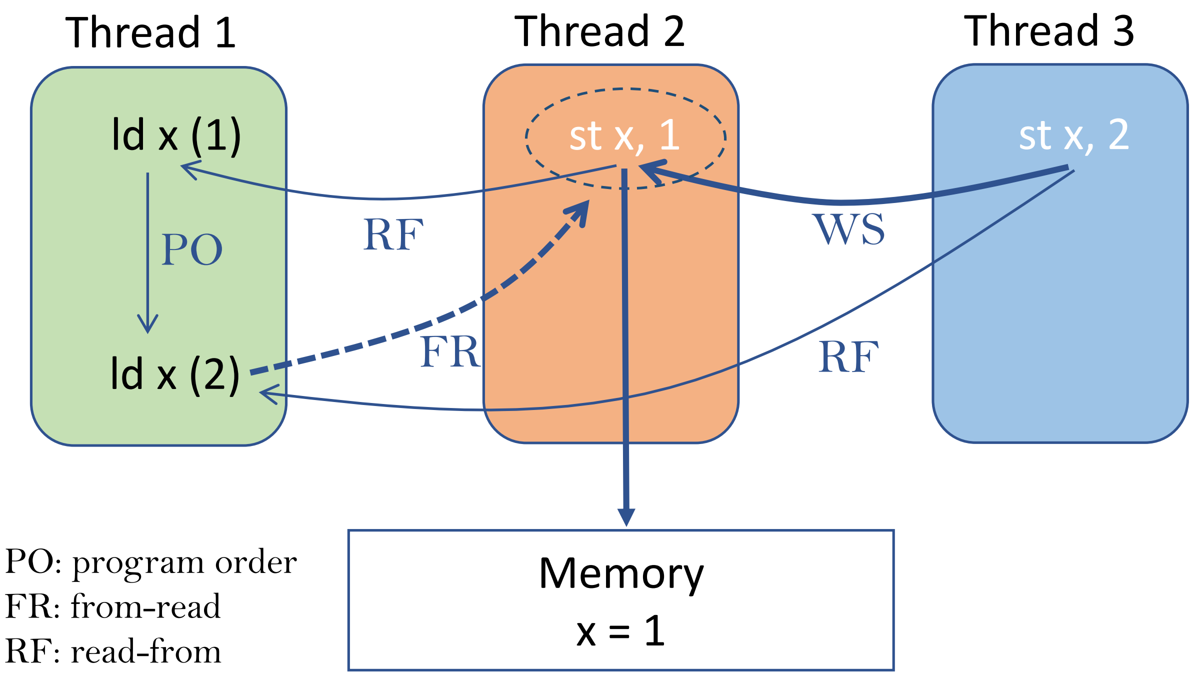


PO: program order
FR: from-read
RF: read-from
WS: write serialization

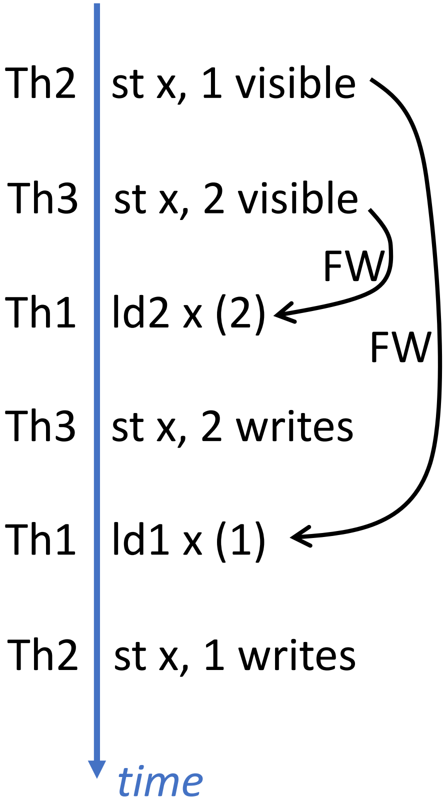


ITSLF: Local Store Order

Initially: x = 0



PO: program order
FR: from-read
RF: read-from
WS: write serialization

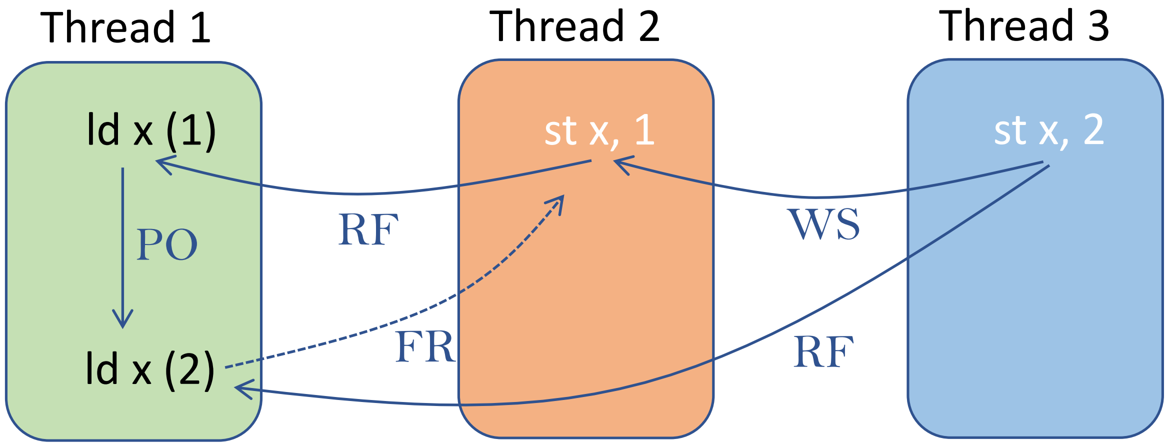


ITSLF: Local Store Order

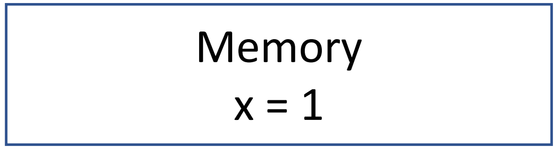
ITSLF solution

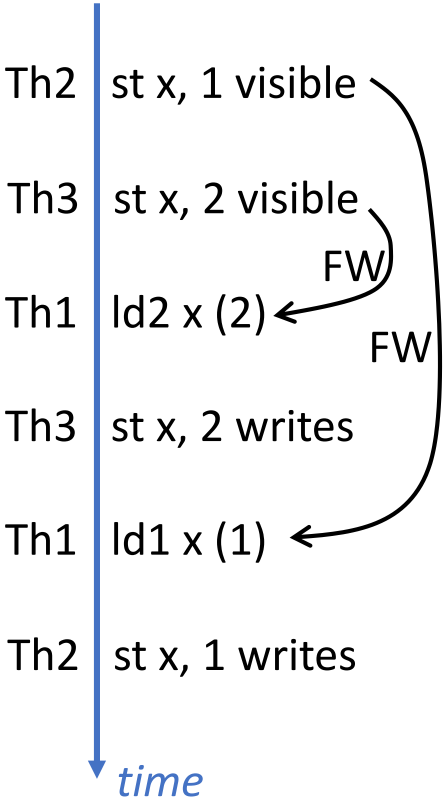
Only a single store on a particular address, the youngest based on local visibility order (youngest to become non-speculative), can forward to loads.

Initially: x = 0



PO: program order
FR: from-read
RF: read-from
WS: write serialization



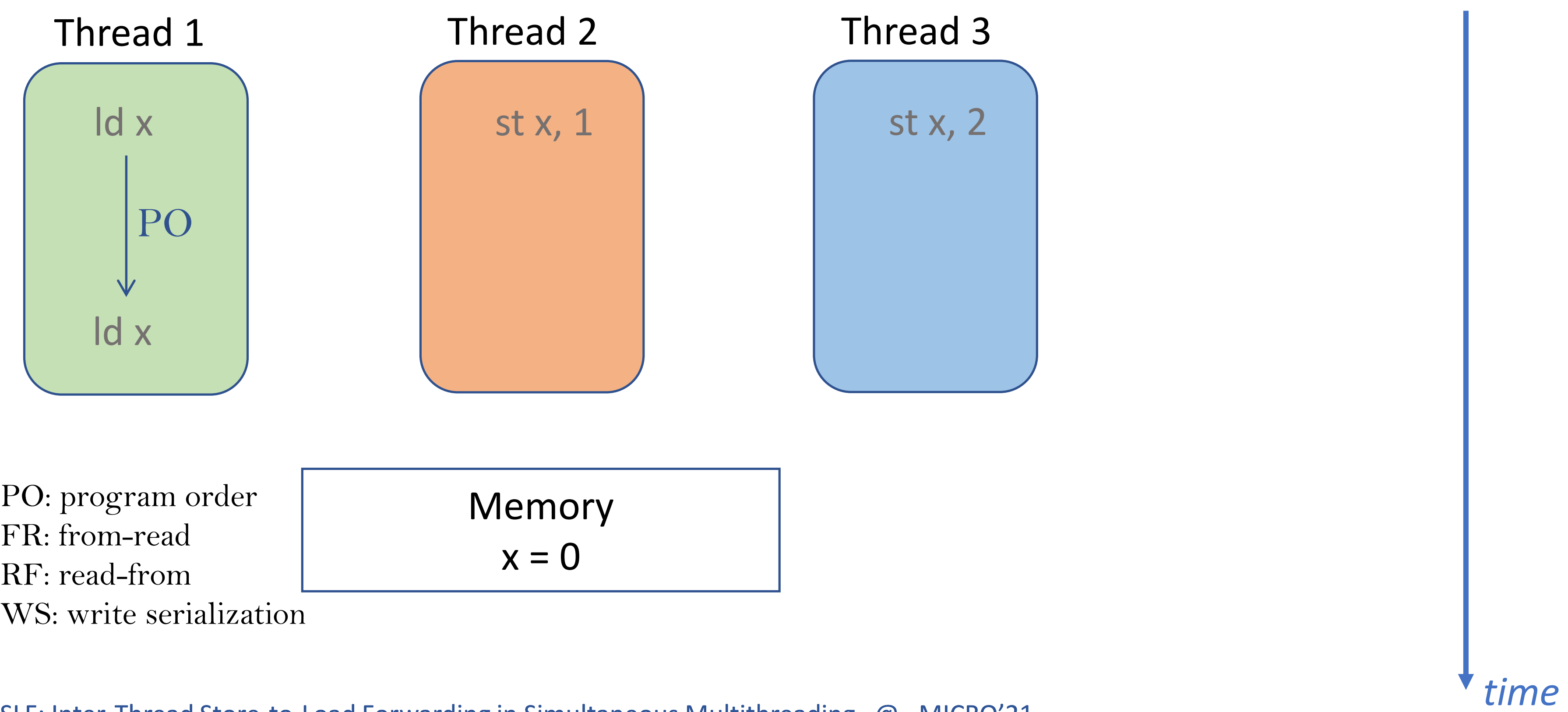


ITSLF: Local Store Order

ITSLF solution

Only a single store on a particular address, the youngest based on local visibility order (youngest to become non-speculative), can forward to loads.

Initially: x = 0



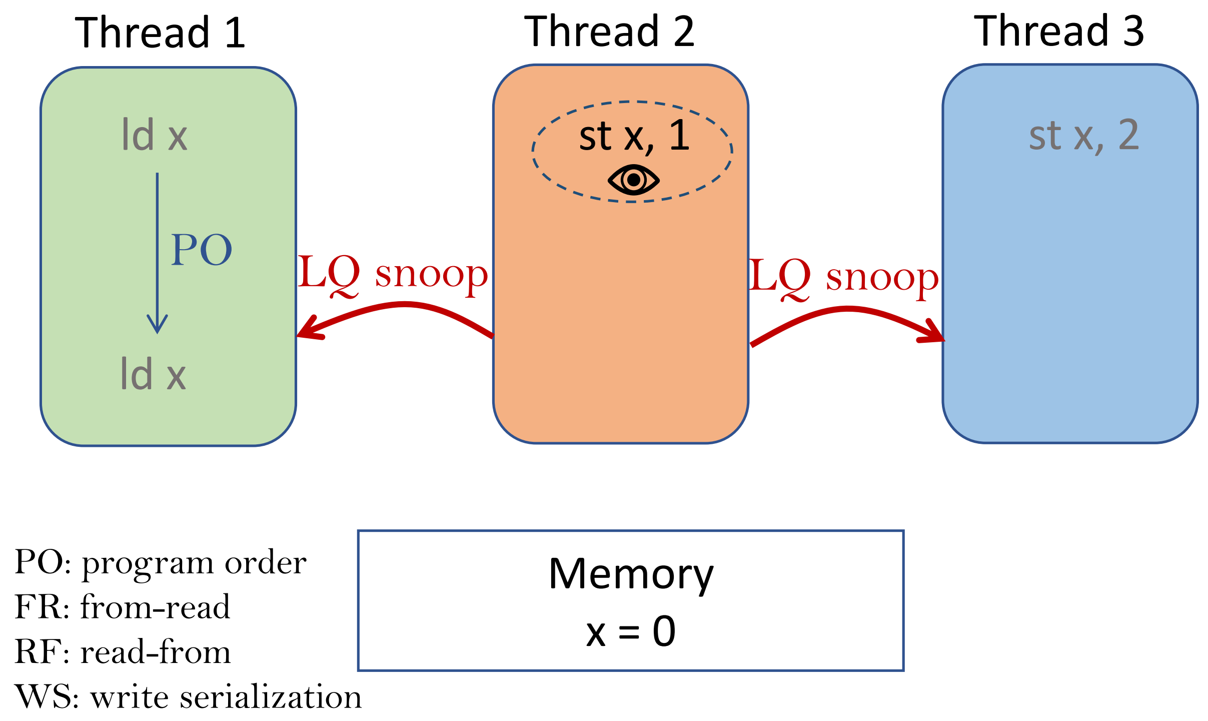
PO: program order
FR: from-read
RF: read-from
WS: write serialization

ITSLF: Local Store Order

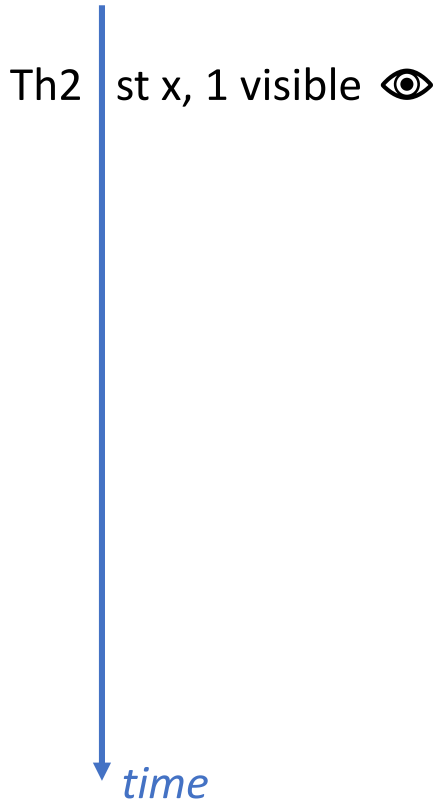
ITSLF solution

Only a single store on a particular address, the youngest based on local visibility order (youngest to become non-speculative), can forward to loads.

Initially: x = 0



PO: program order
FR: from-read
RF: read-from
WS: write serialization

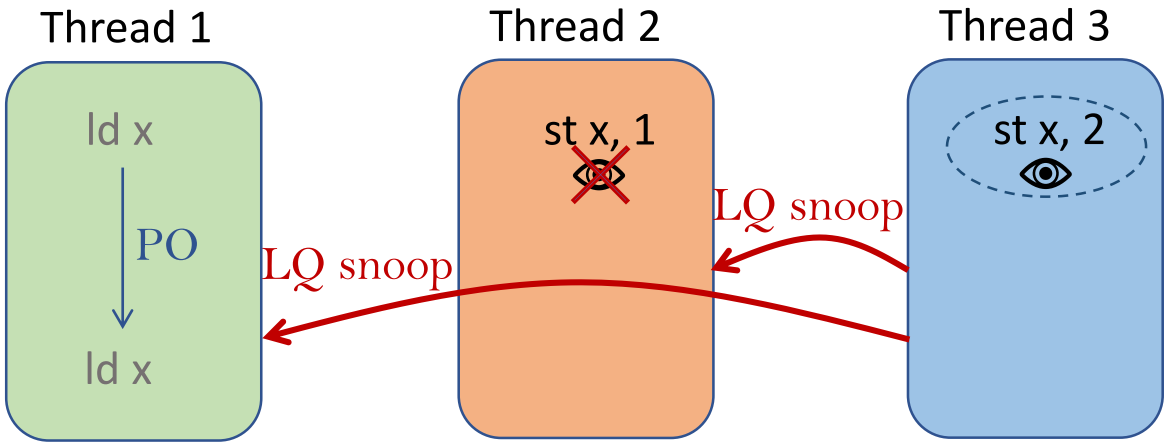


ITSLF: Local Store Order

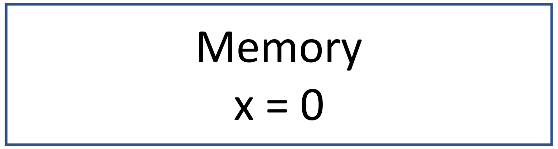
ITSLF solution

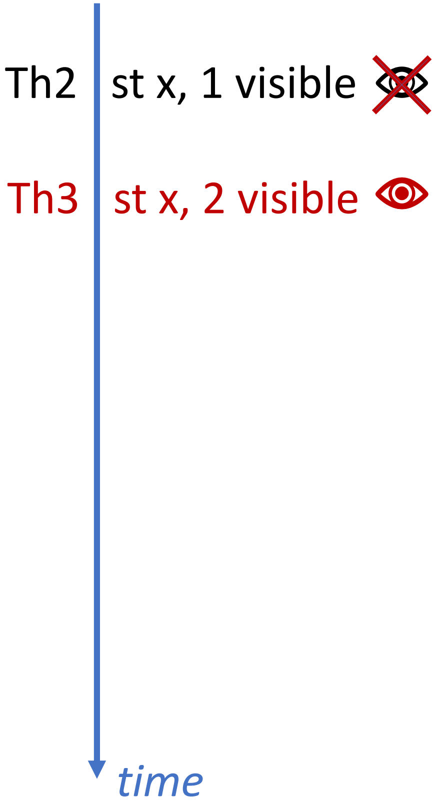
Only a single store on a particular address, the youngest based on local visibility order (youngest to become non-speculative), can forward to loads.

Initially: x = 0



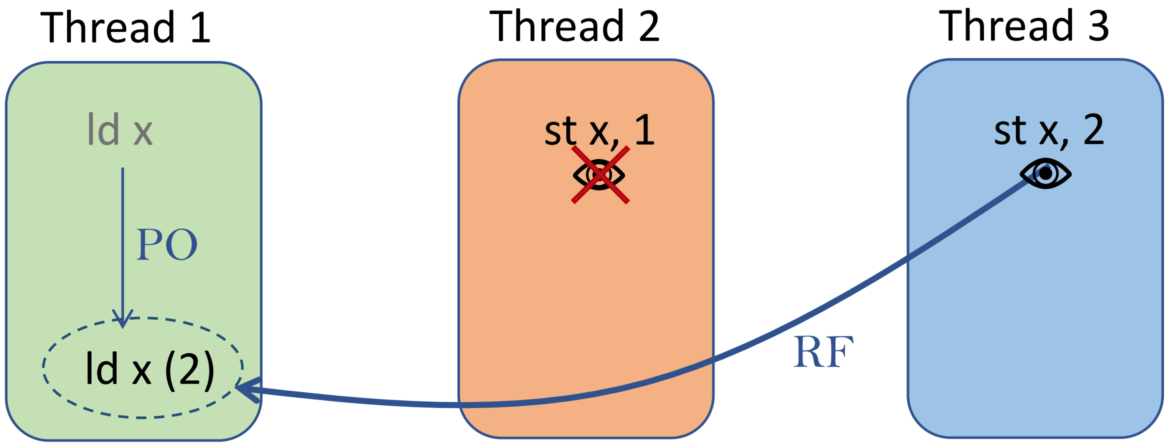
PO: program order
FR: from-read
RF: read-from
WS: write serialization

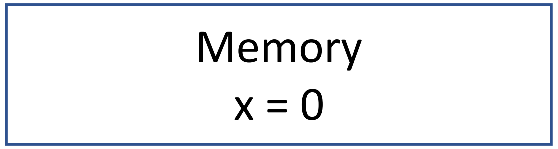




ITSLF: Local Store Order

Initially: x = 0

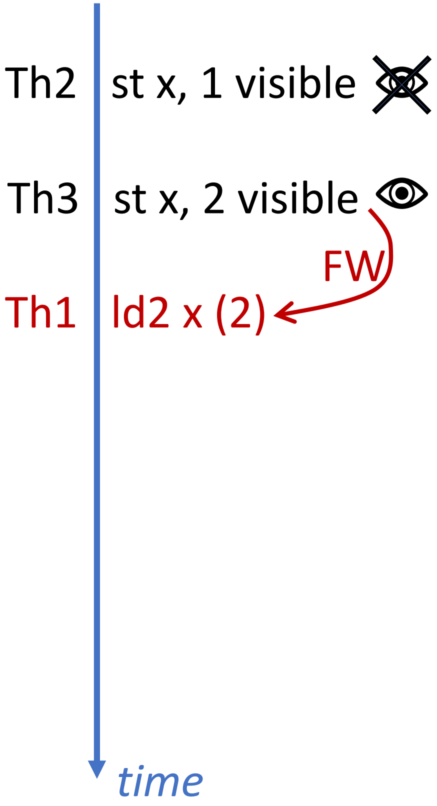




PO: program order
FR: from-read
RF: read-from
WS: write serialization

ITSLF solution

Only a single store on a particular address, the youngest based on local visibility order (youngest to become non-speculative), can forward to loads.

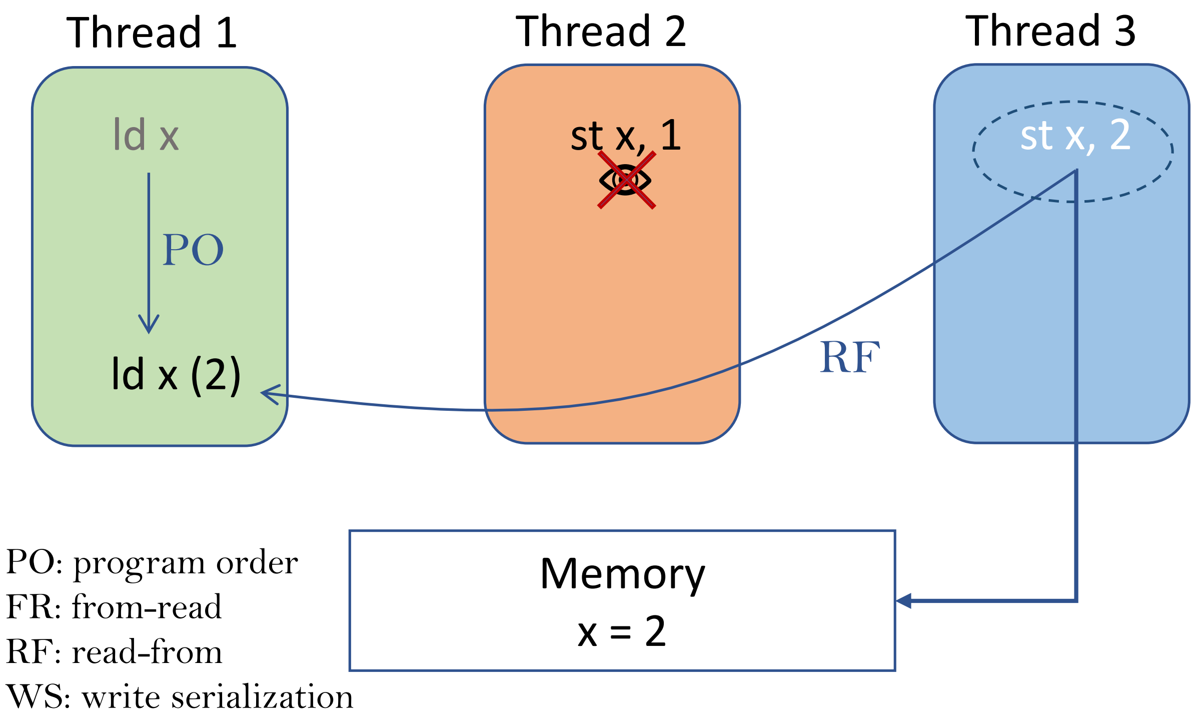


ITSLF: Local Store Order

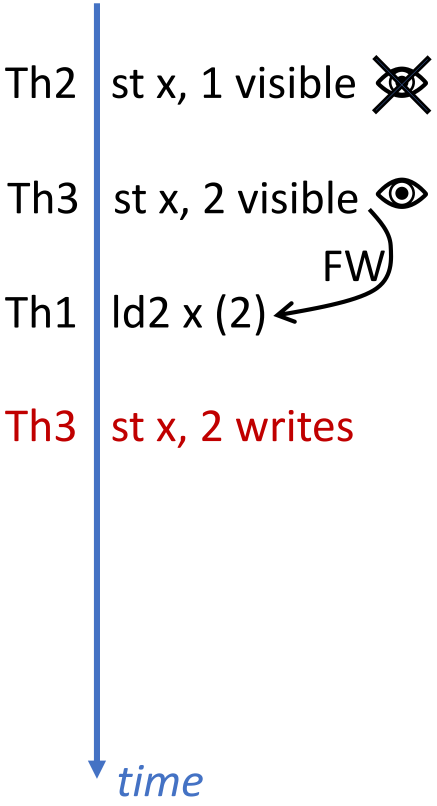
ITSLF solution

Only a single store on a particular address, the youngest based on local visibility order (youngest to become non-speculative), can forward to loads.

Initially: x = 0

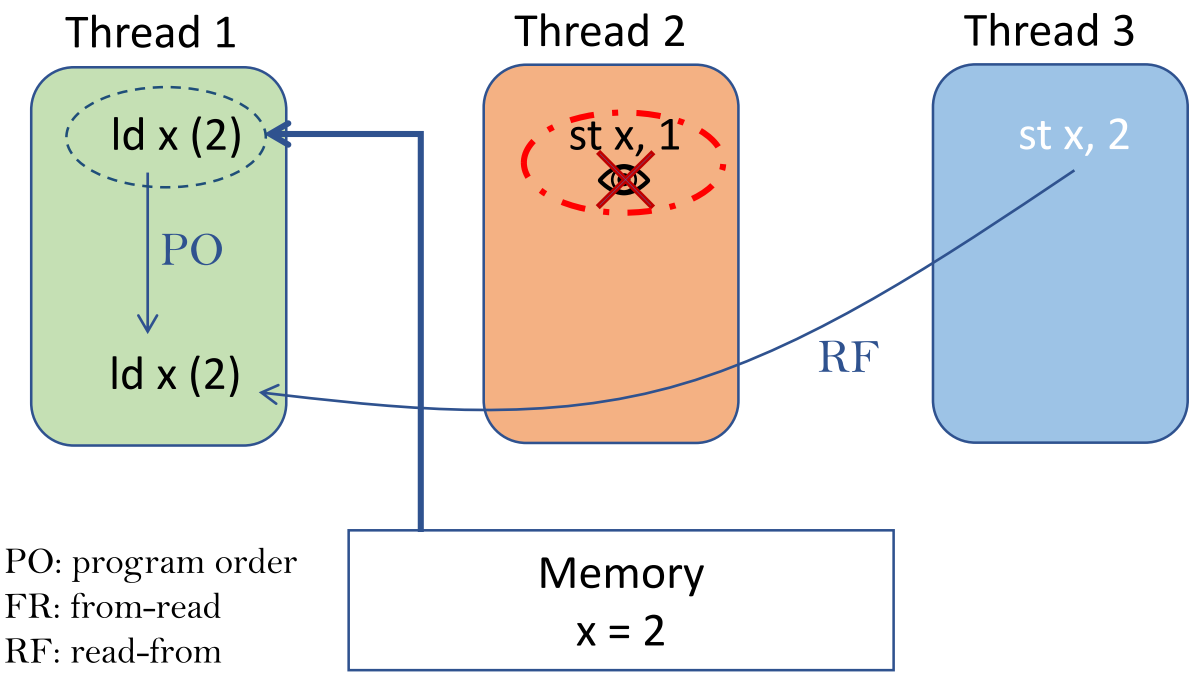


PO: program order
FR: from-read
RF: read-from
WS: write serialization



ITSLF: Local Store Order

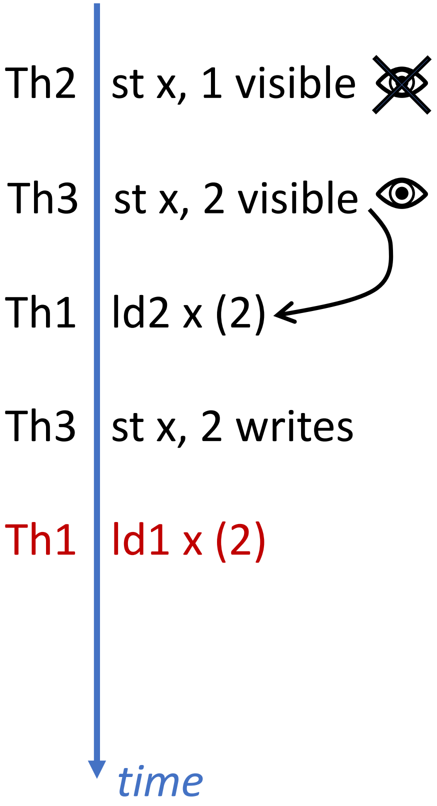
Initially: x = 0



PO: program order
FR: from-read
RF: read-from
WS: write serialization

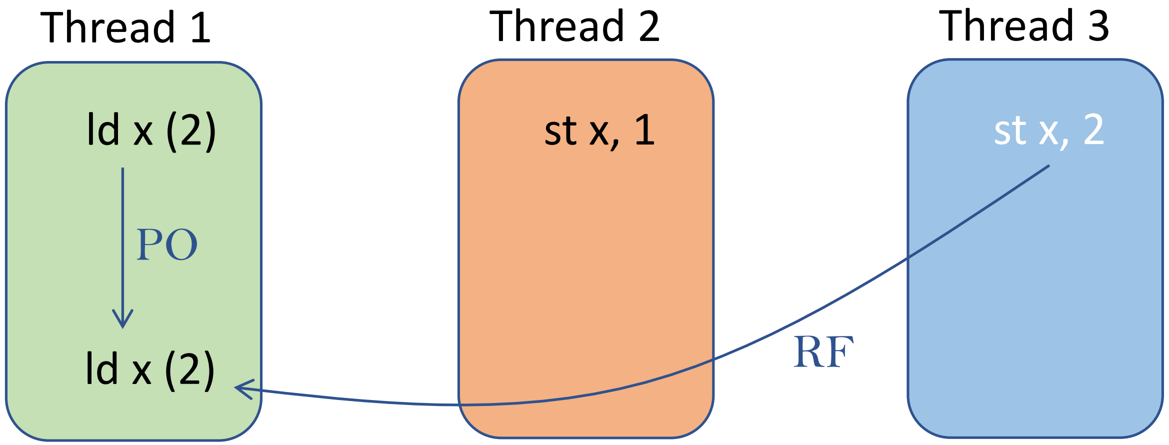
ITSLF solution

Only a single store on a particular address, the youngest based on local visibility order (youngest to become non-speculative), can forward to loads.



ITSLF: Local Store Order

Initially: x = 0



Memory
x = 2

PO: program order
FR: from-read
RF: read-from
WS: write serialization

ITSLF solution

Only a single store on a particular address, the youngest based on local visibility order (youngest to become non-speculative), can forward to loads.

Cost

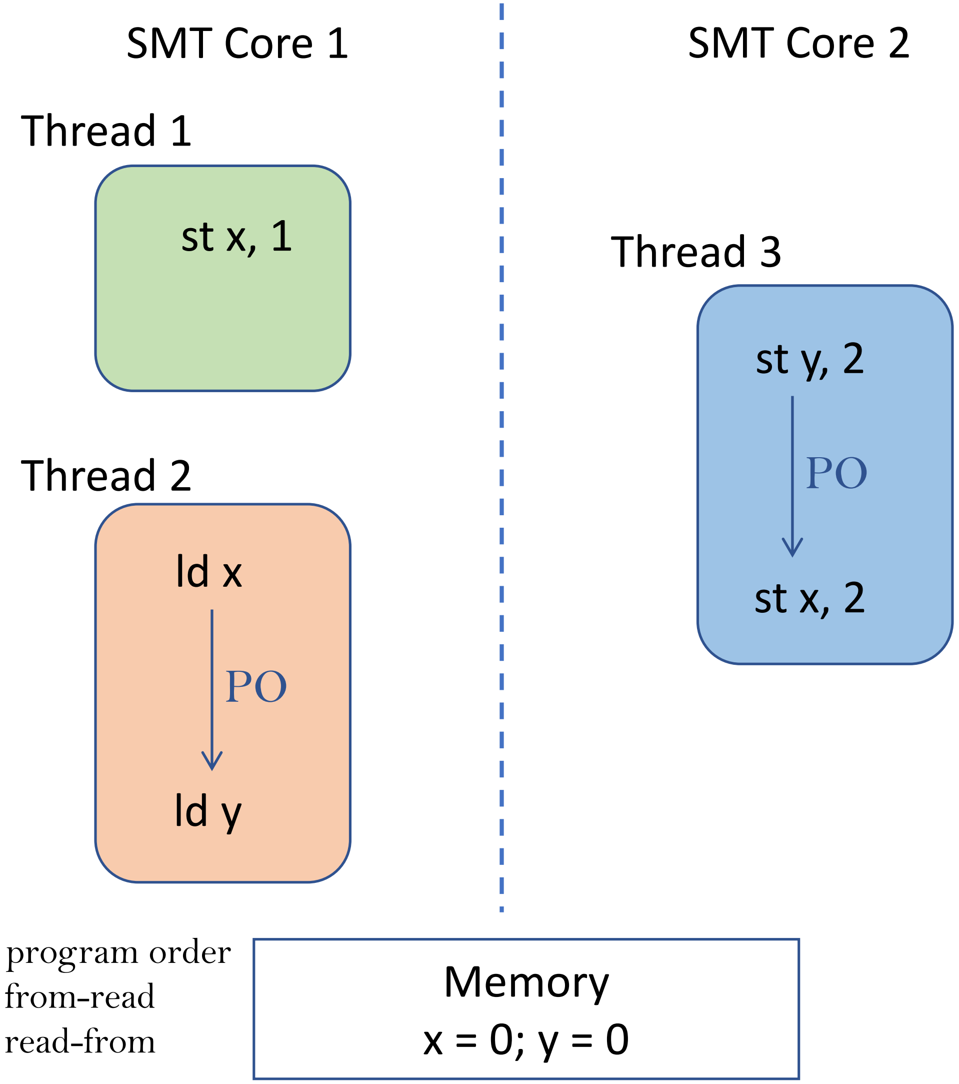
ITSLF only requires extending the SQ entries with a field to store their LV order ($\lceil log_2(SB\ entries) + 1 \rceil$ bits per SB entry).

Issues and Solutions with ITSLF

- *Inter-thread store-to-load-forwarding* could be enabled by not restricting the SQ/SB search to the same thread.
- Exposes store values to some threads before they are inserted in global order and breaks:

1. Coherence and TSO → Point of Local Visibility
2. Write serialization → Local Store Order
3. **Multi-Copy Atomicity**

ITSLF: Multi-Copy Atomicity

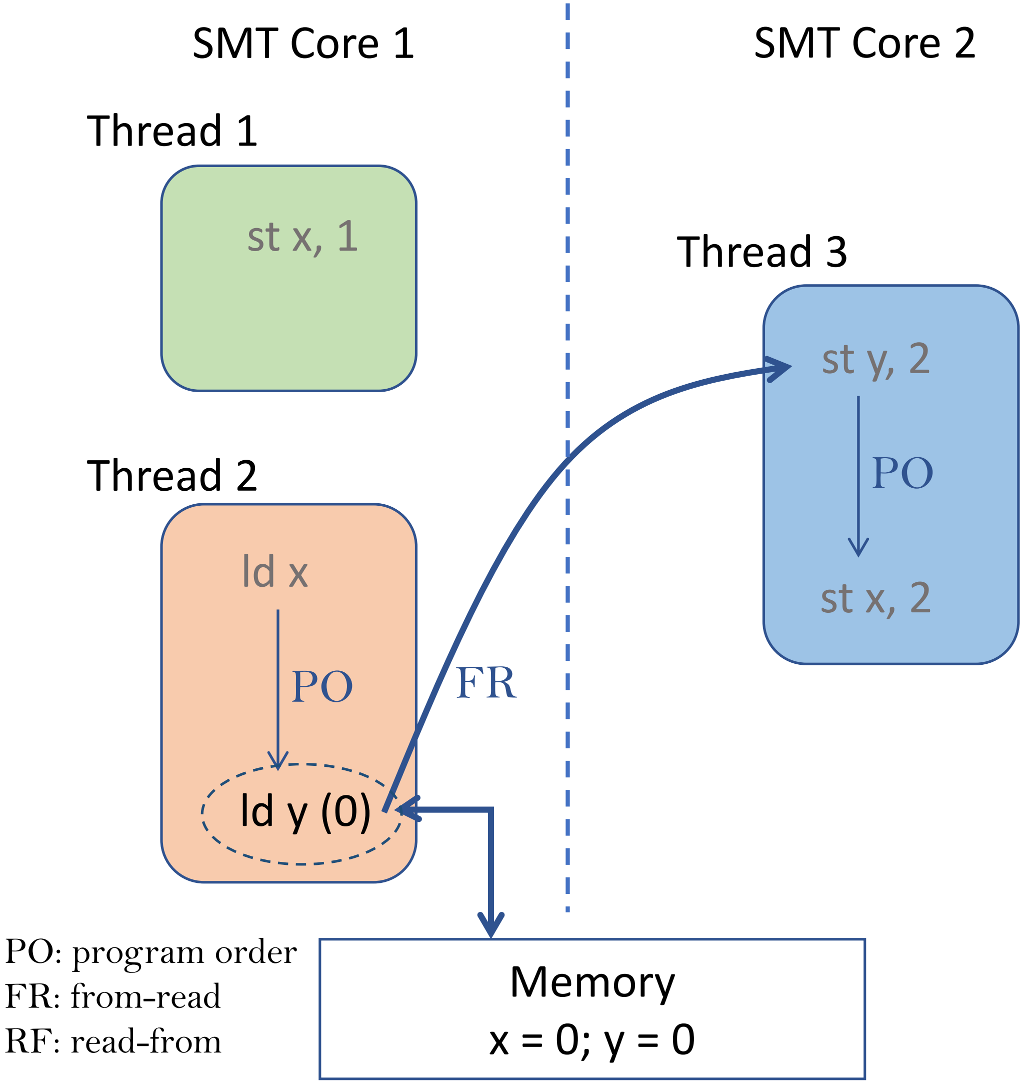


PO: program order
FR: from-read
RF: read-from

Invalid outcome with x86-TSO:

- Memory: [x] = 1; [y] = 2;
- Thread 2: x = 1; y = 0;

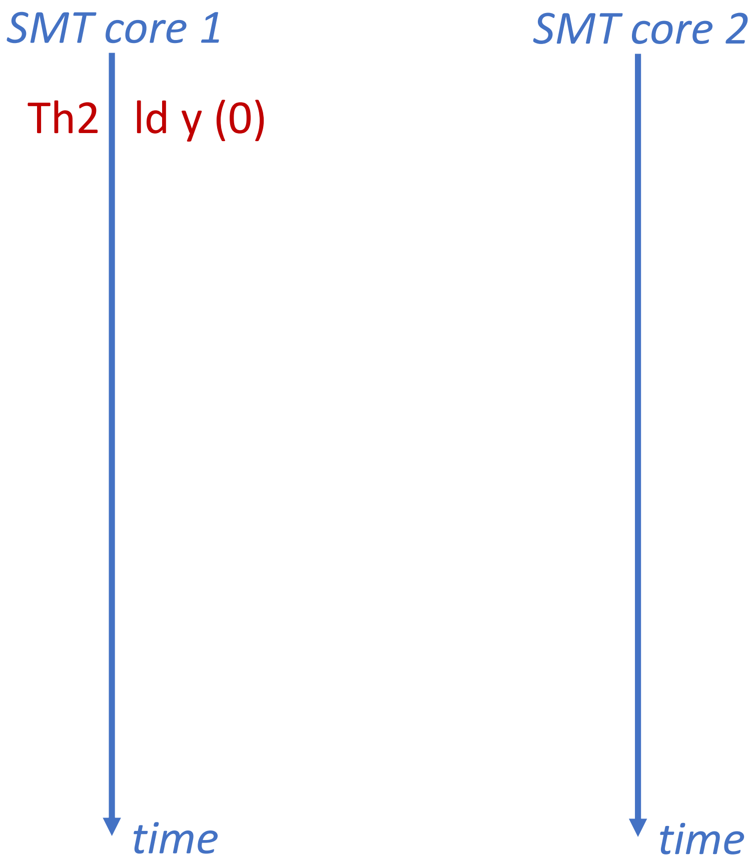
ITSLF: Multi-Copy Atomicity



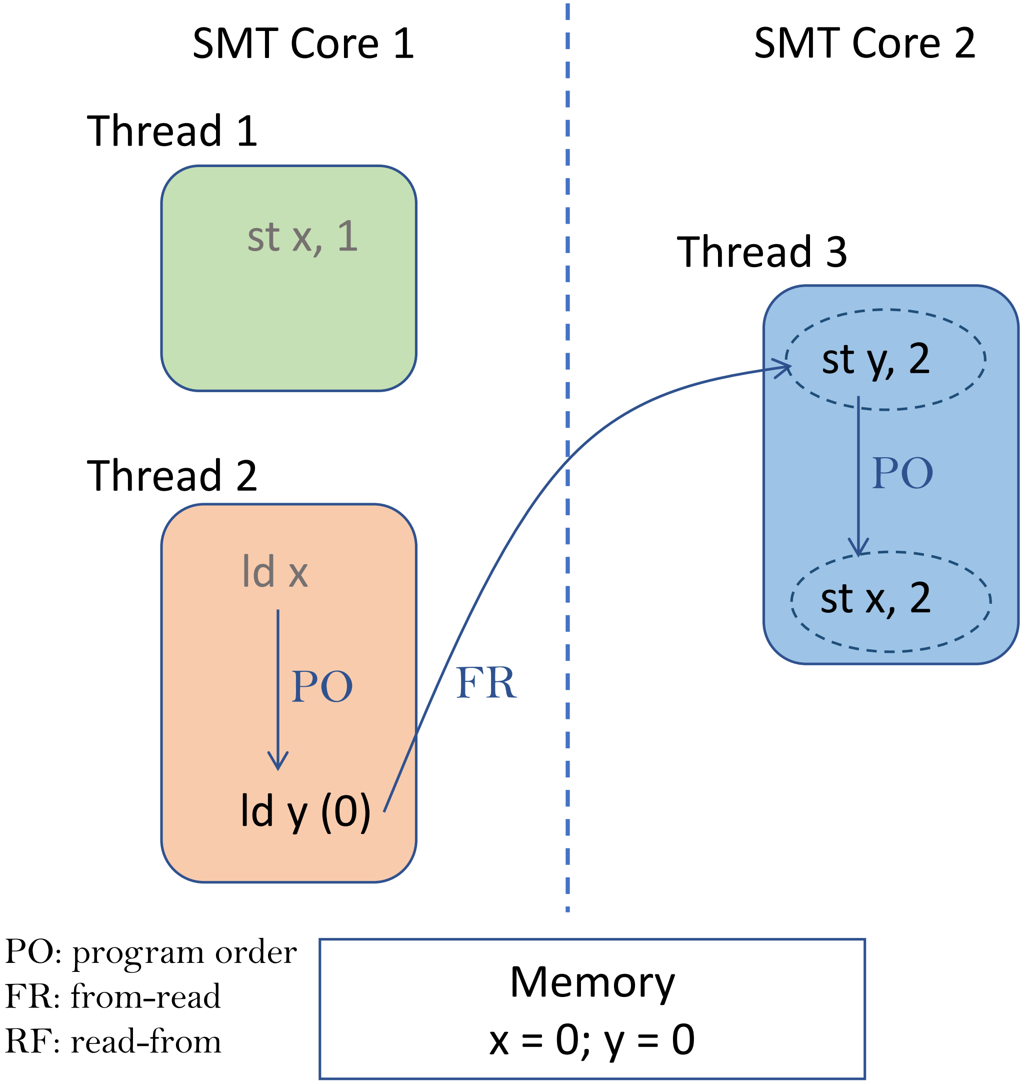
PO: program order
FR: from-read
RF: read-from

Invalid outcome with x86-TSO:

- Memory: [x] = 1; [y] = 2;
- Thread 2: x = 1; y = 0;

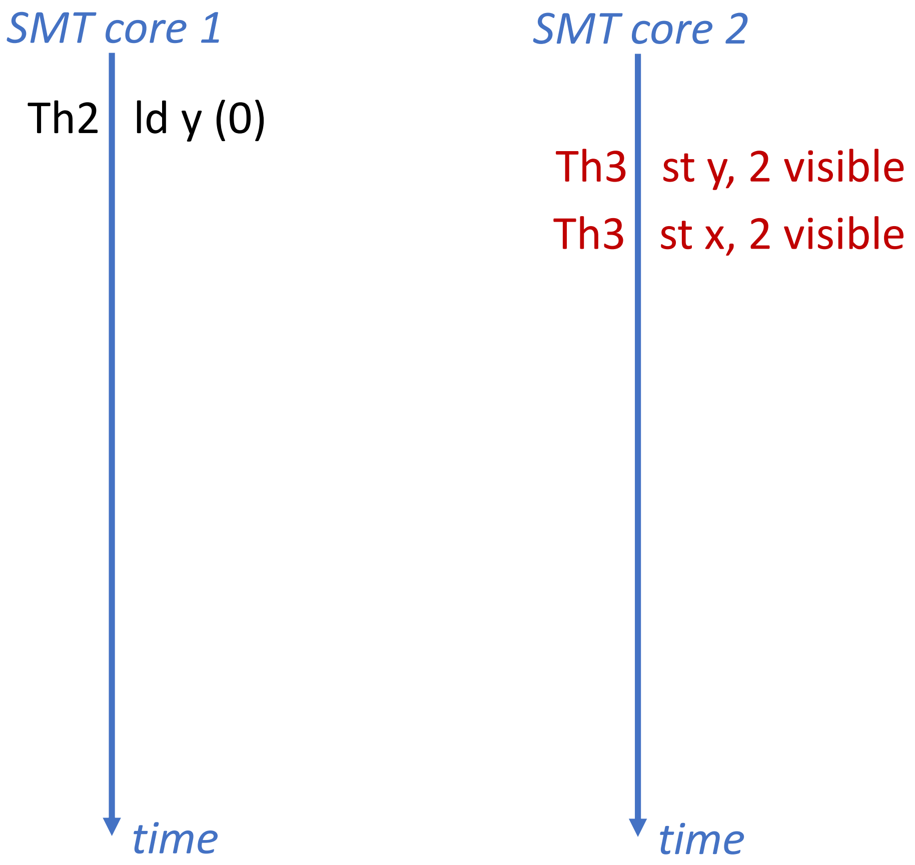


ITSLF: Multi-Copy Atomicity

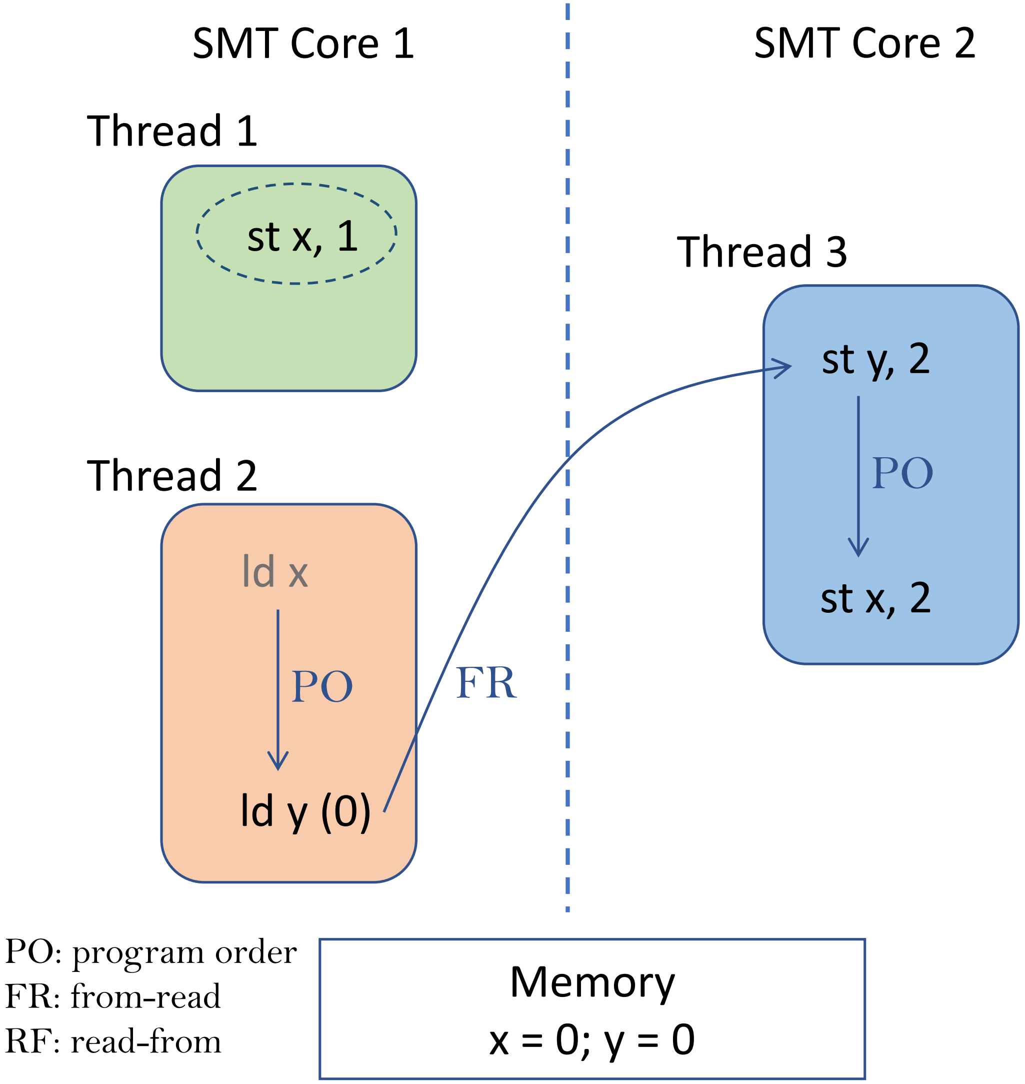


Invalid outcome with x86-TSO:

- Memory: [x] = 1; [y] = 2;
- Thread 2: x = 1; y = 0;



ITSLF: Multi-Copy Atomicity

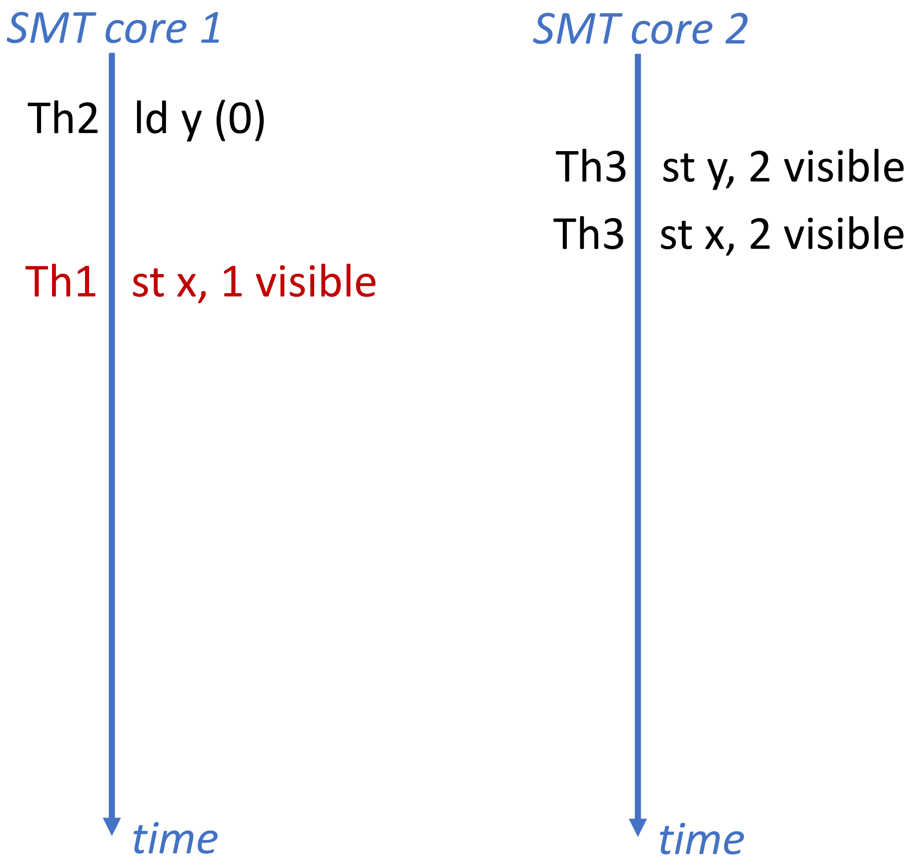


PO: program order
FR: from-read
RF: read-from

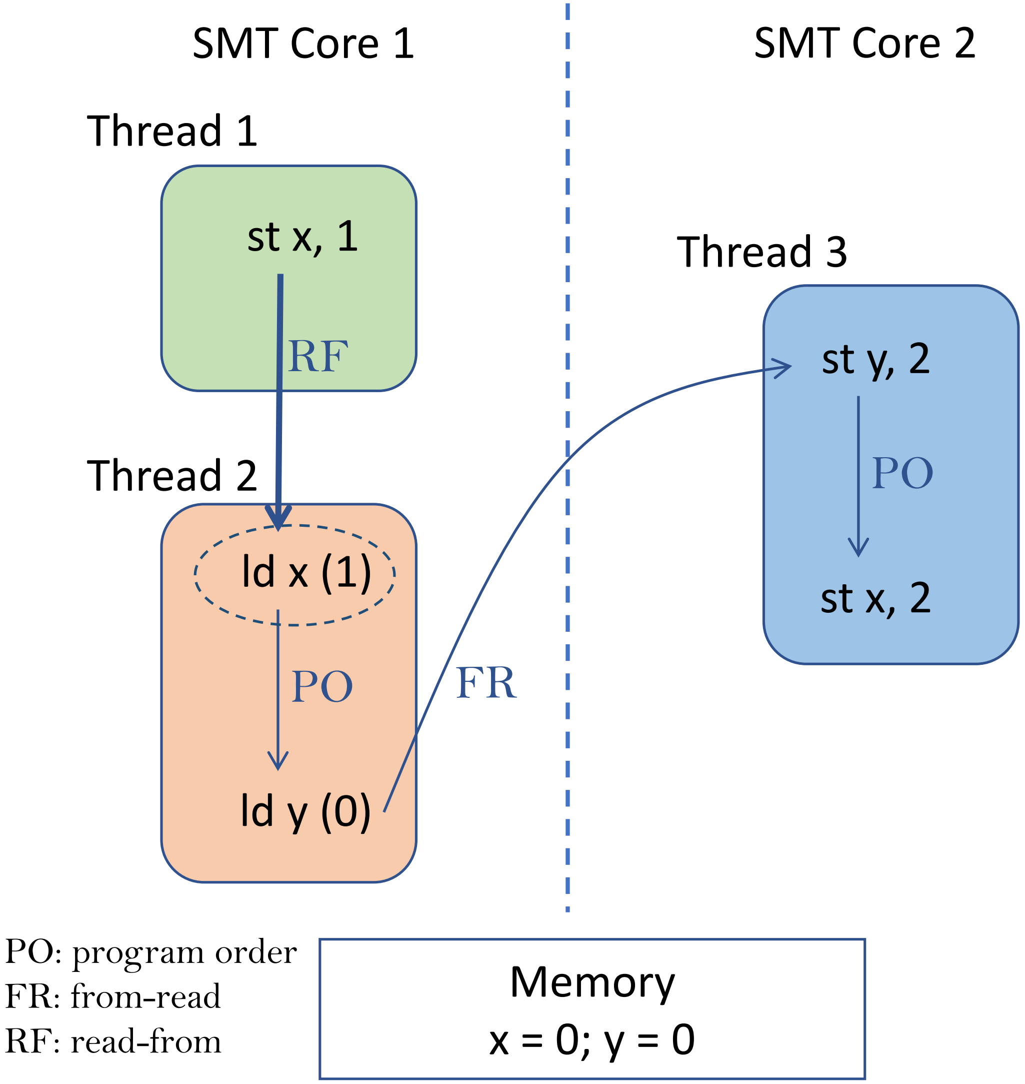
Memory
x = 0; y = 0

Invalid outcome with x86-TSO:

- Memory: [x] = 1; [y] = 2;
- Thread 2: x = 1; y = 0;

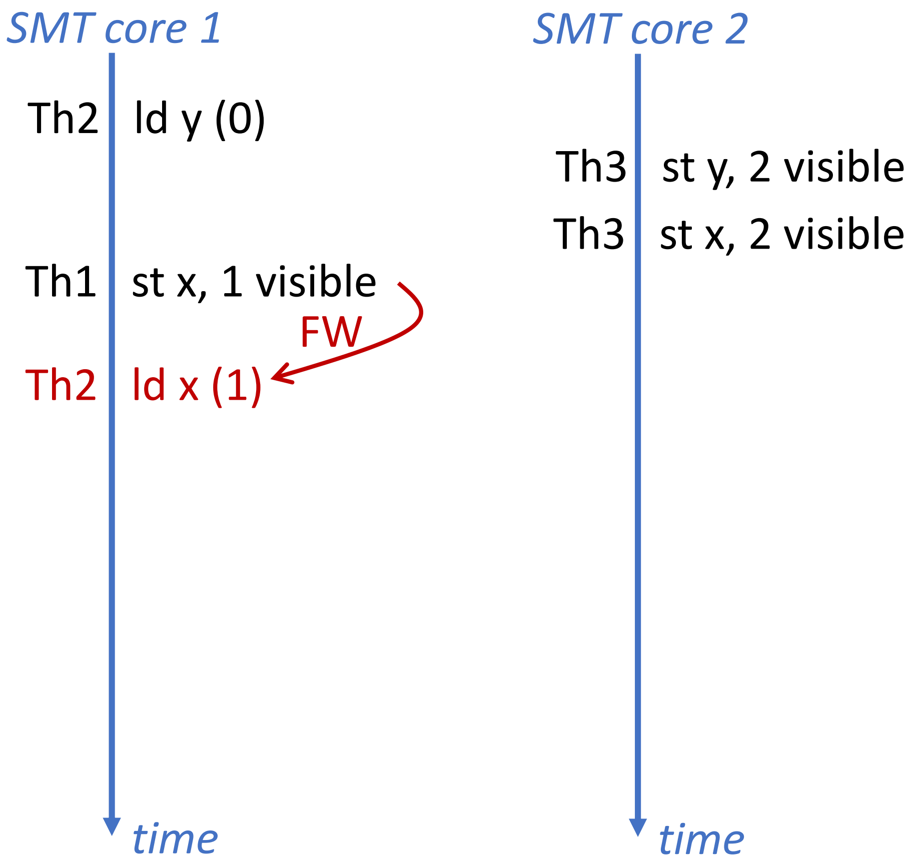


ITSLF: Multi-Copy Atomicity

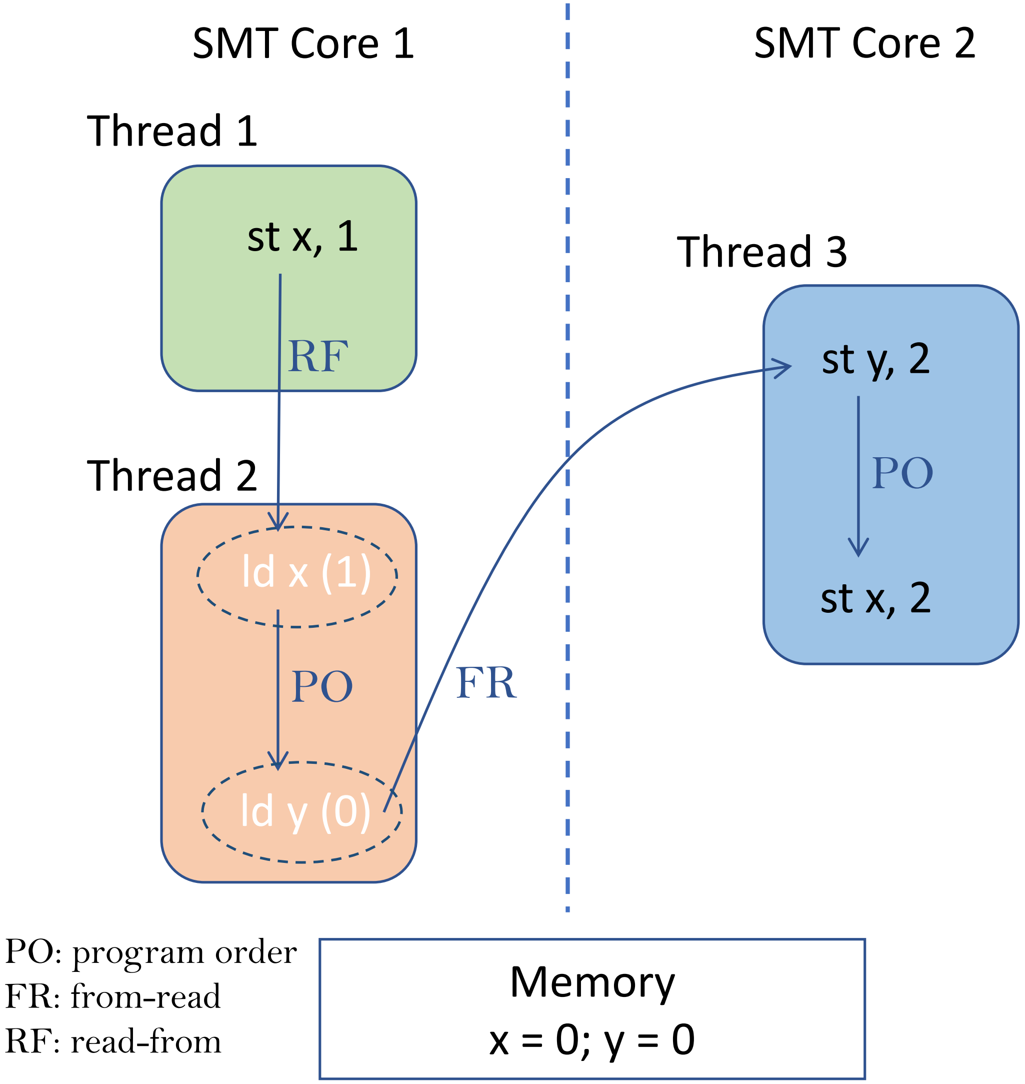


Invalid outcome with x86-TSO:

- Memory: [x] = 1; [y] = 2;
- Thread 2: x = 1; y = 0;

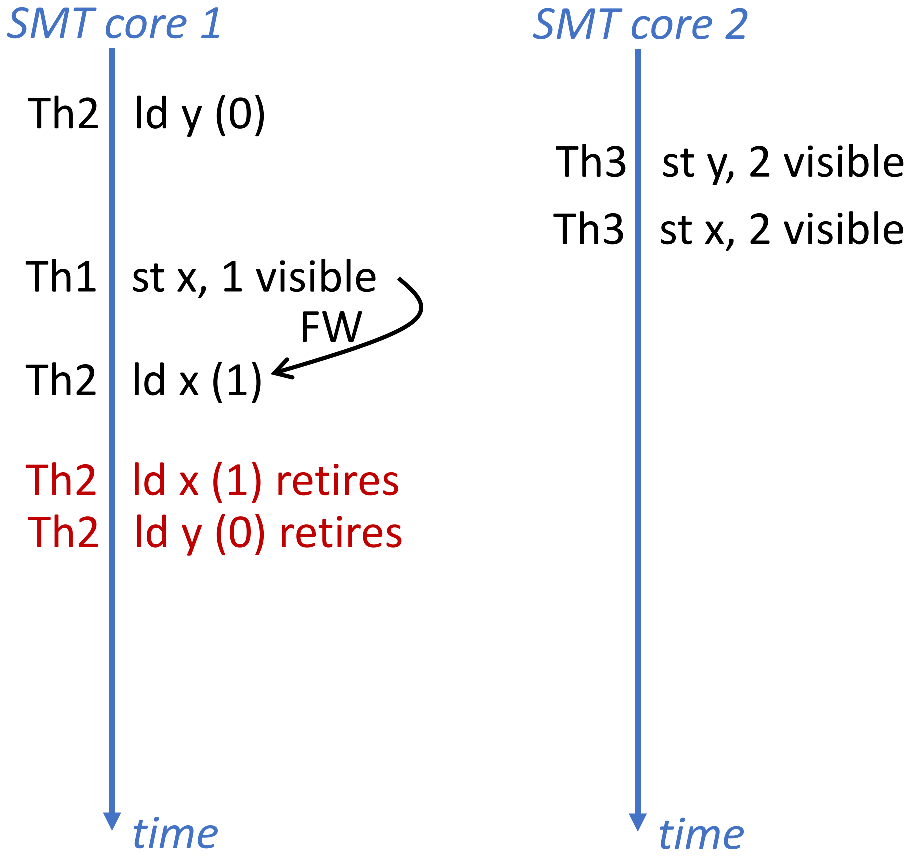


ITSLF: Multi-Copy Atomicity

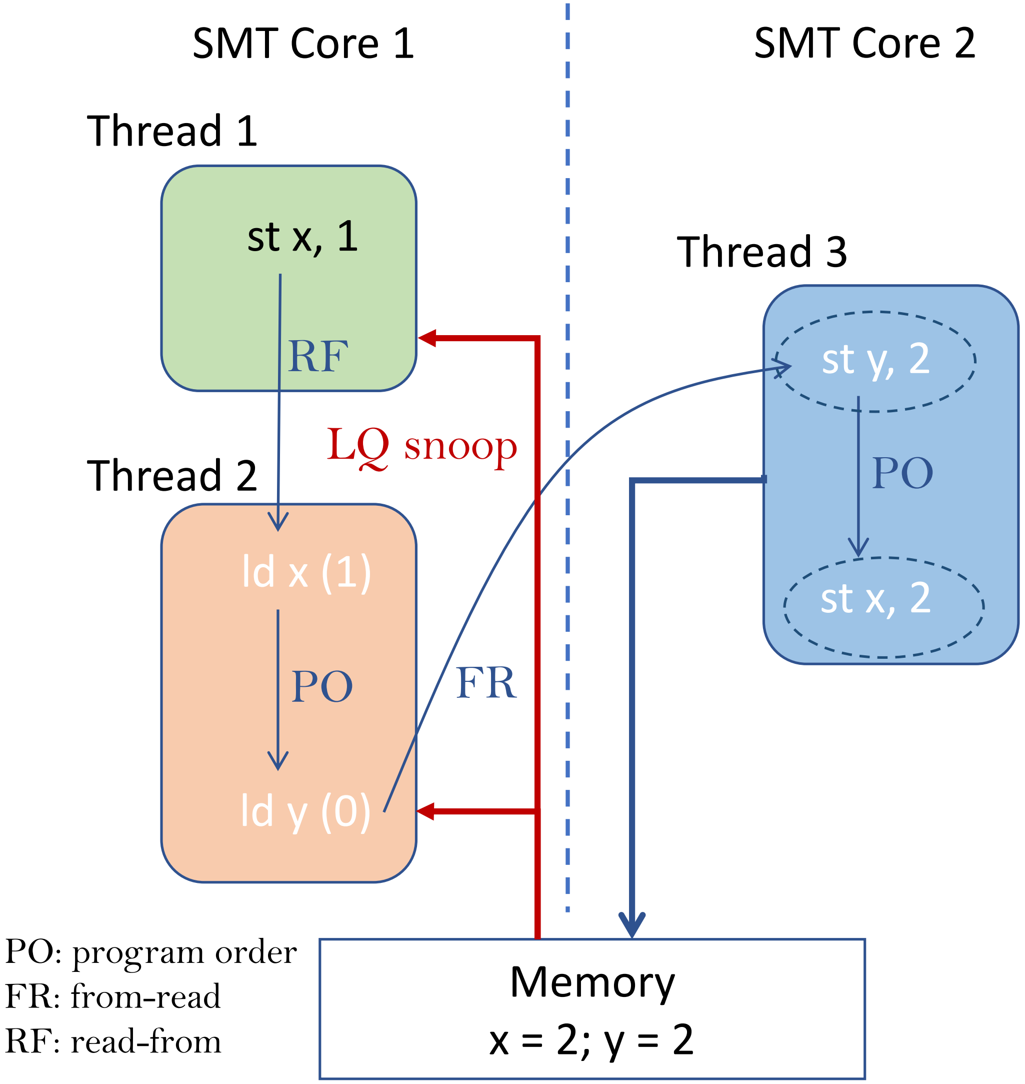


Invalid outcome with x86-TSO:

- Memory: [x] = 1; [y] = 2;
- Thread 2: x = 1; y = 0;



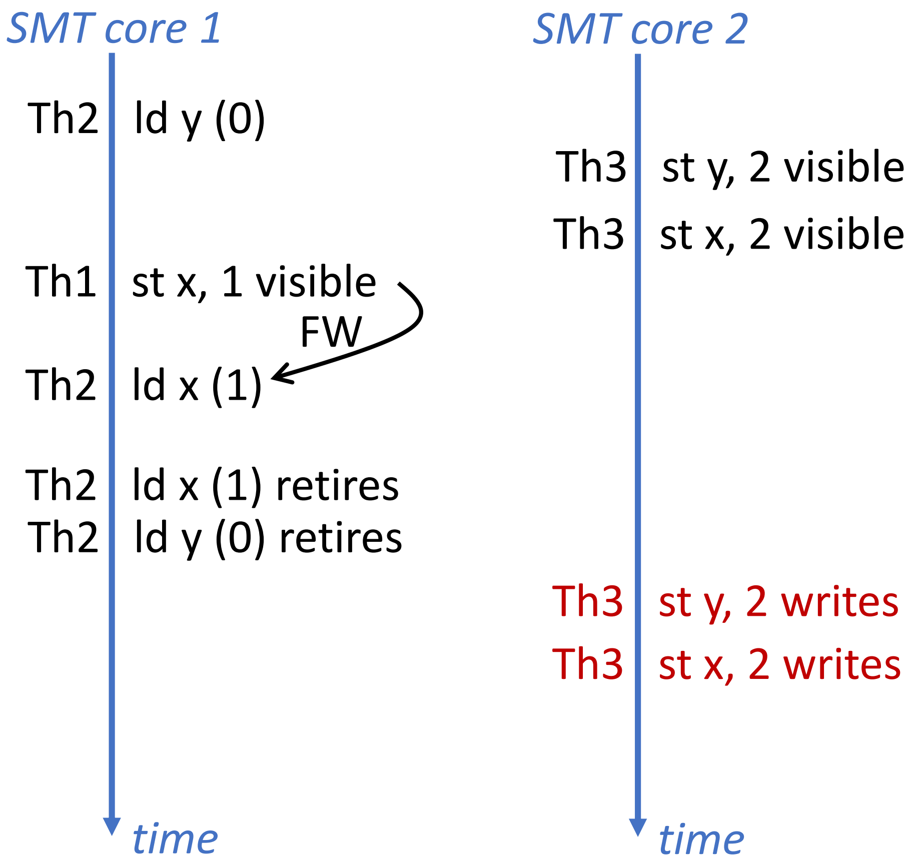
ITSLF: Multi-Copy Atomicity



PO: program order
FR: from-read
RF: read-from

Invalid outcome with x86-TSO:

- Memory: [x] = 1; [y] = 2;
- Thread 2: x = 1; y = 0;



SMT core 1

Th2 ld y (0)

Th1 st x, 1 visible

FW

Th2 ld x (1)

Th2 ld x (1) retires

Th2 ld y (0) retires

time

SMT core 2

Th3 st y, 2 visible

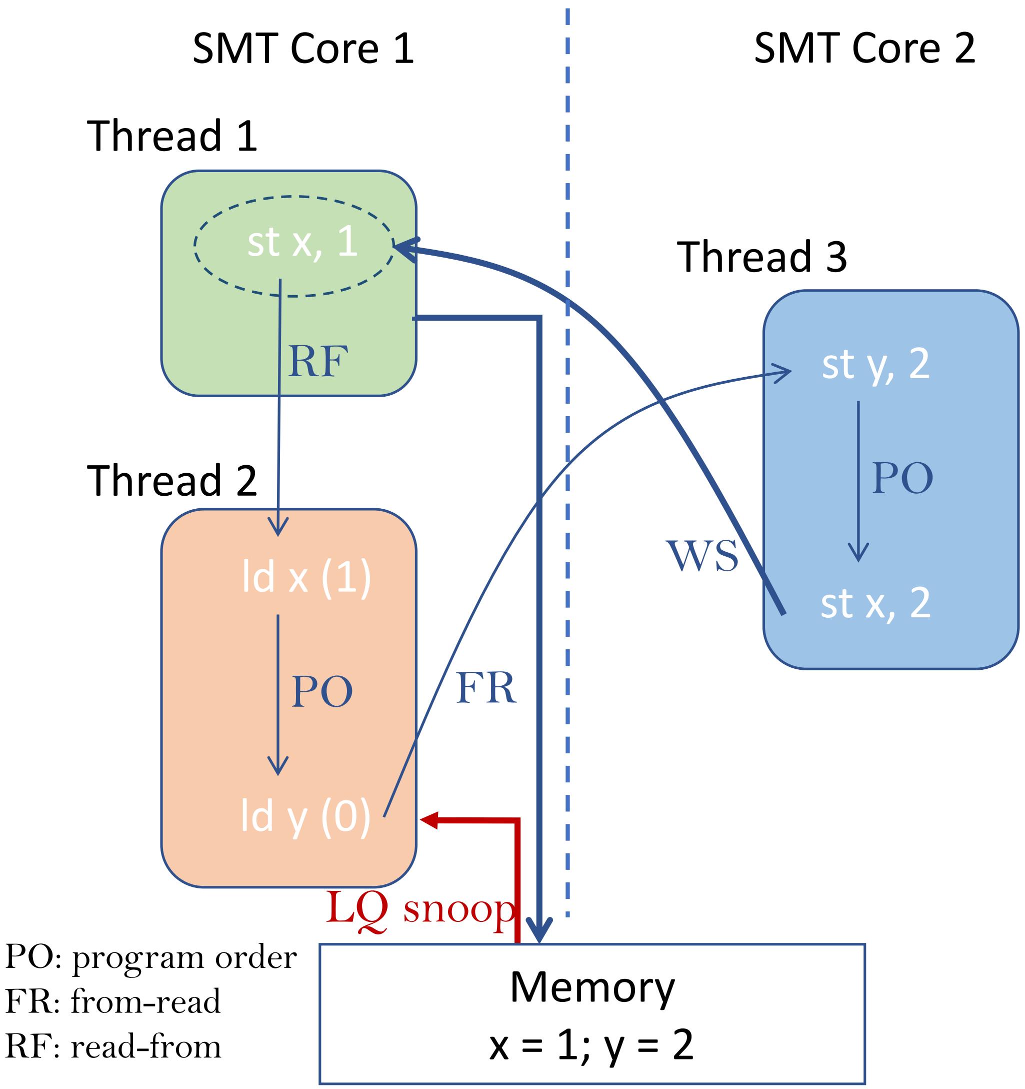
Th3 st x, 2 visible

Th3 st y, 2 writes

Th3 st x, 2 writes

time

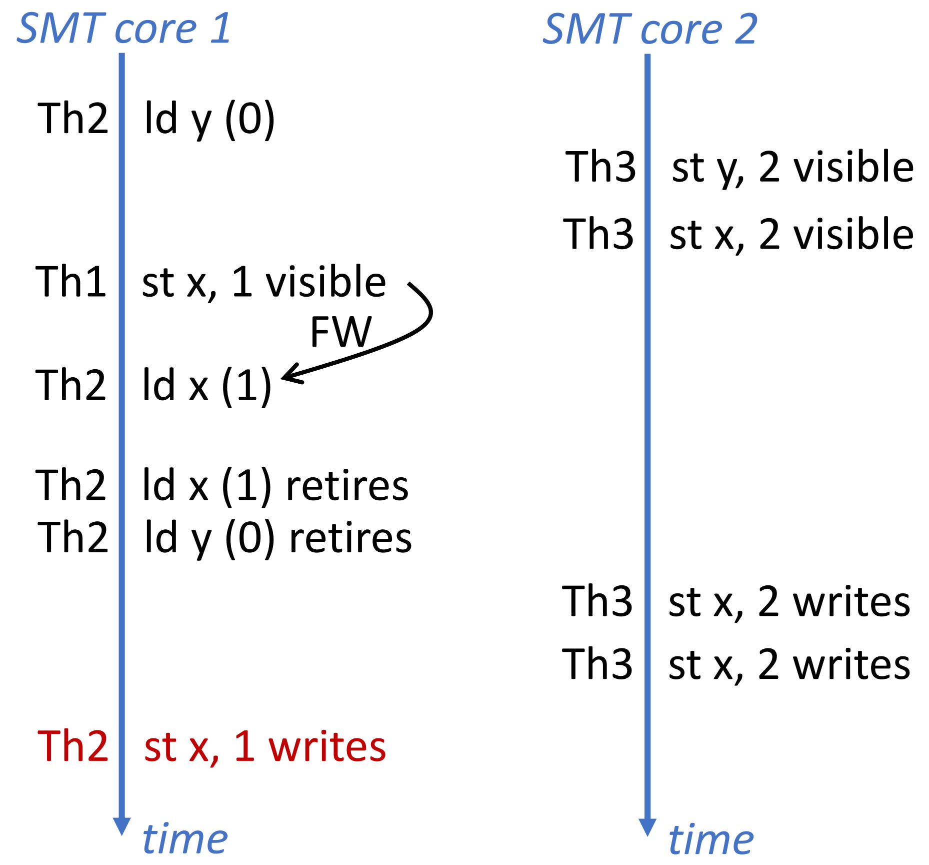
ITSLF: Multi-Copy Atomicity



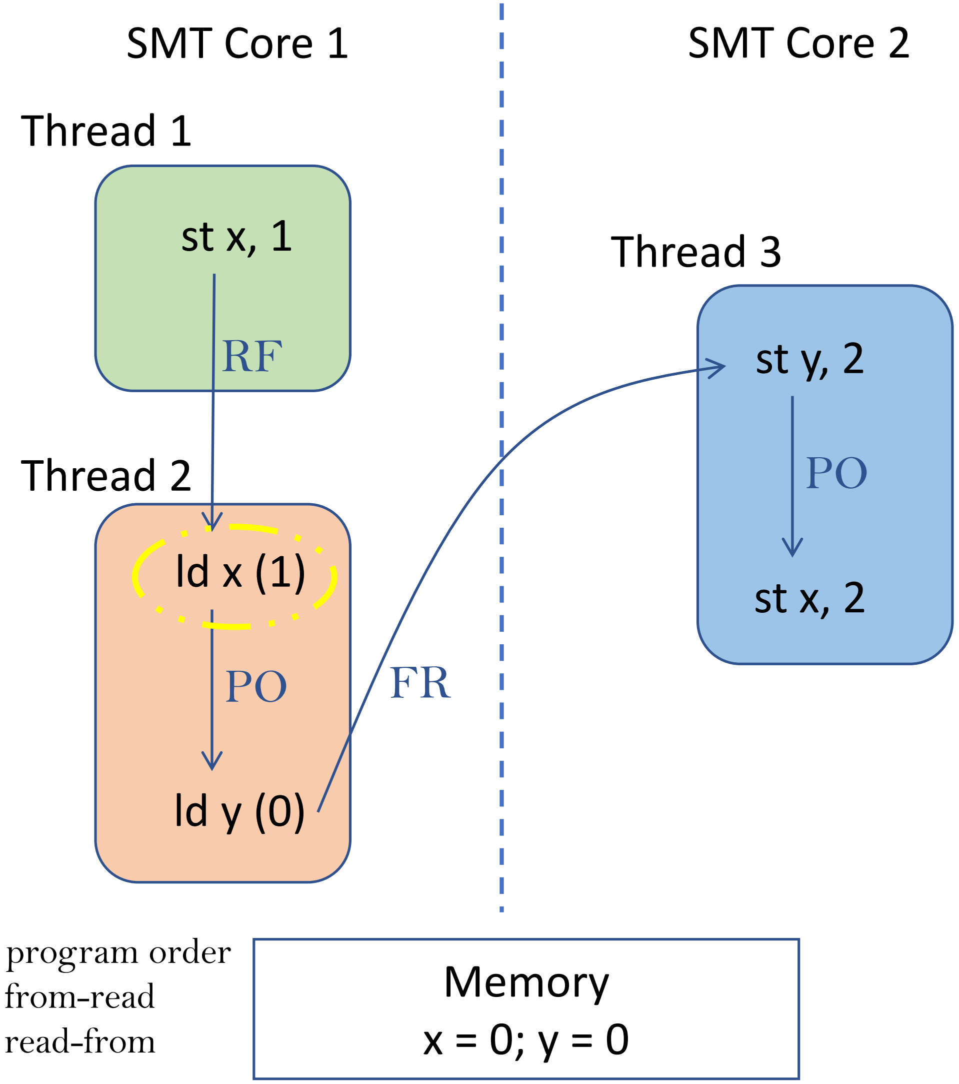
PO: program order
FR: from-read
RF: read-from

Invalid outcome with x86-TSO:

- Memory: [x] = 1; [y] = 2;
- Thread 2: x = 1; y = 0;



ITSLF: Multi-Copy Atomicity



PO: program order
FR: from-read
RF: read-from

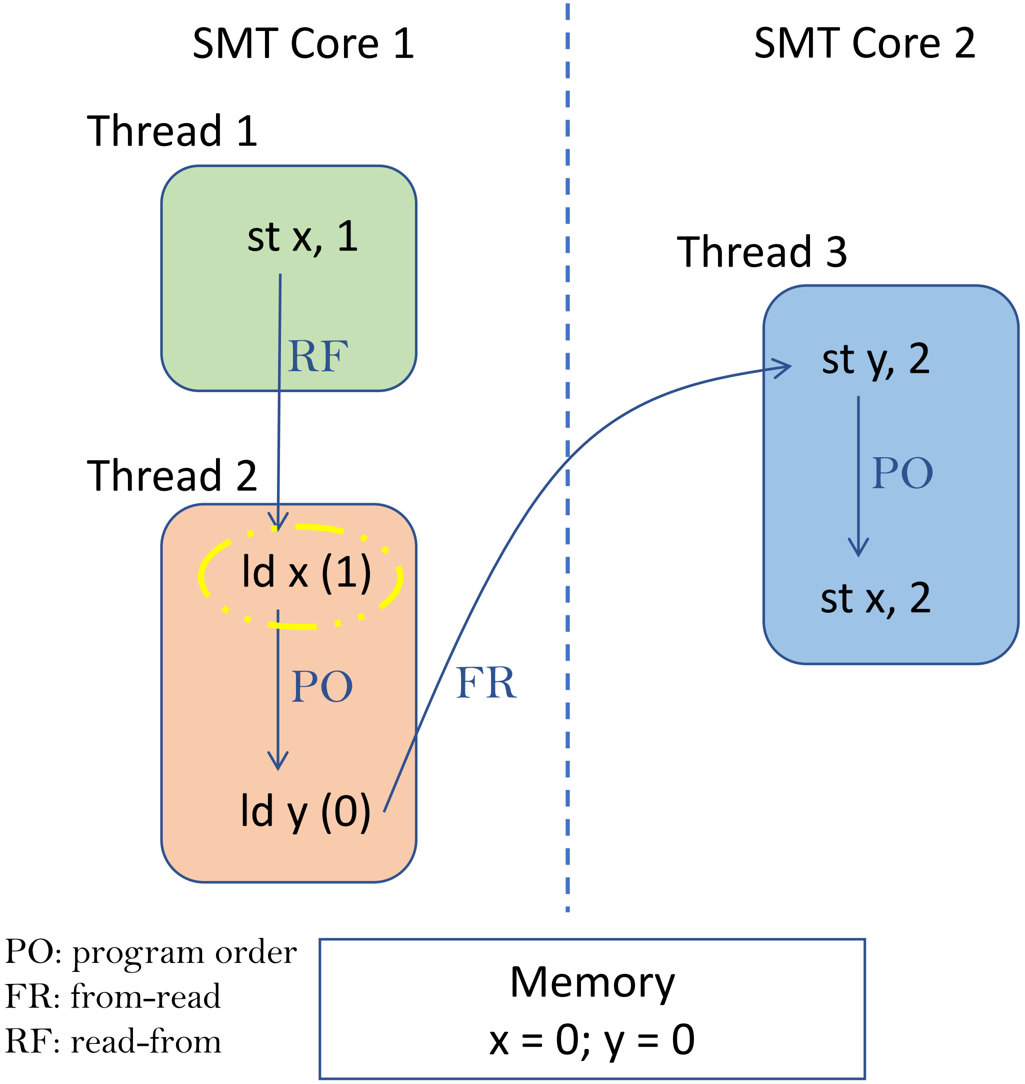
ITSLF solution

A load receiving forwarded data from a different thread:

i) cannot retire until the forwarding store becomes globally visible

ii) until it retires, it makes all younger loads in its thread speculative and subject to squashing from conflicting stores.

ITSLF: Multi-Copy Atomicity



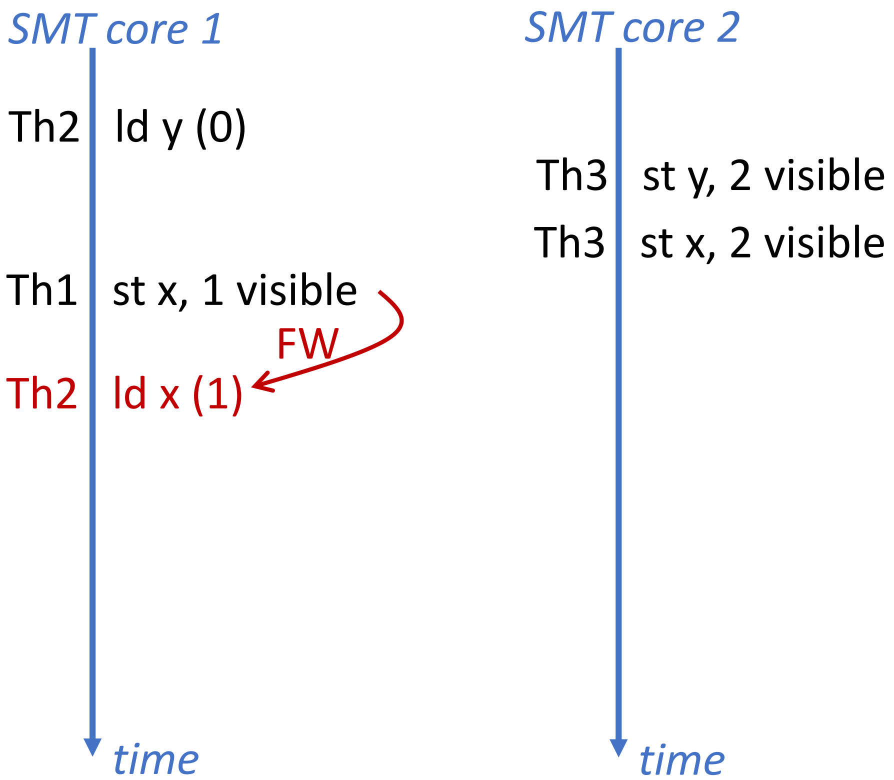
PO: program order
FR: from-read
RF: read-from

ITSLF solution

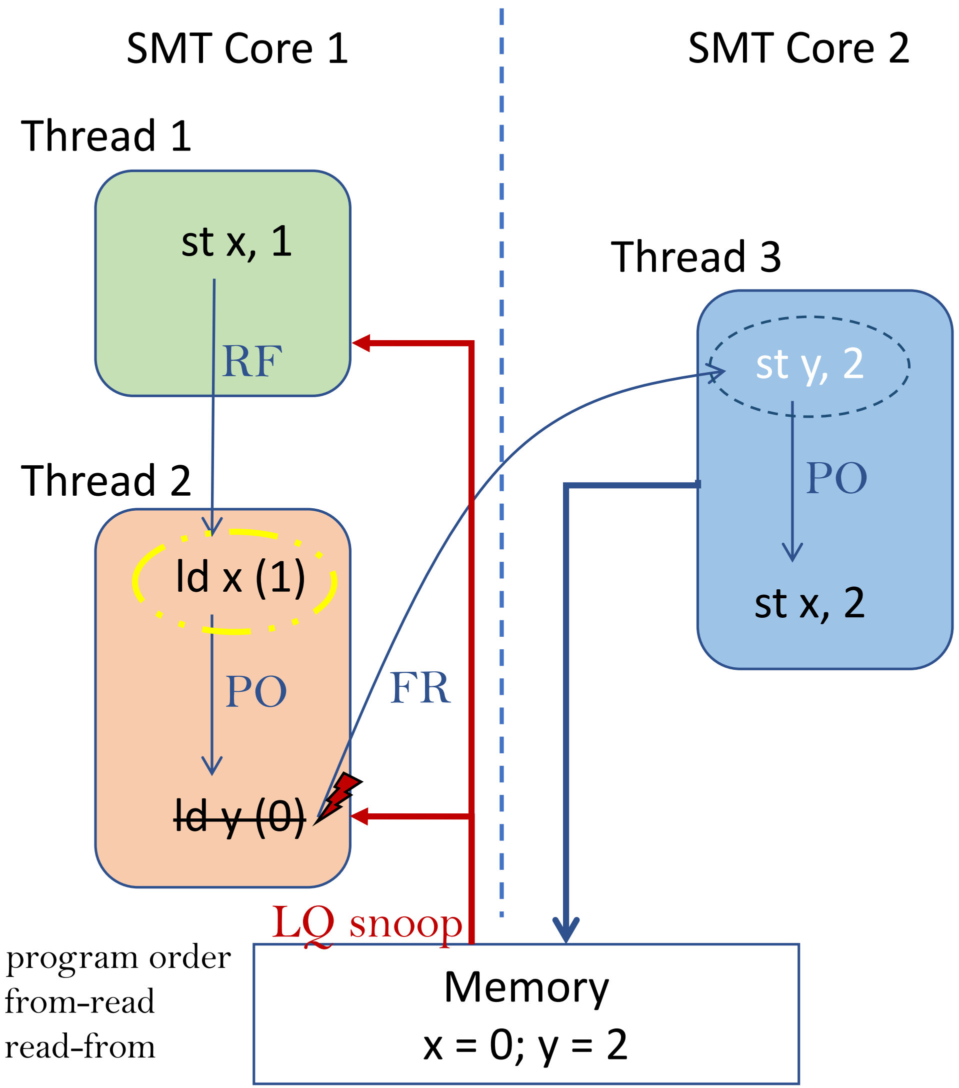
A load receiving forwarded data from a different thread:

i) cannot retire until the forwarding store becomes globally visible

ii) until it retires, it makes all younger loads in its thread speculative and subject to squashing from conflicting stores.



ITSLF: Multi-Copy Atomicity


SMT Core 1

SMT Core 2

Thread 1

st x, 1

RF

Thread 2

ld x (1)

PO

~~ld y (0)~~

FR

Thread 3

st y, 2

PO

st x, 2

LQ snoop

Memory

x = 0; y = 2

PO: program order

FR: from-read

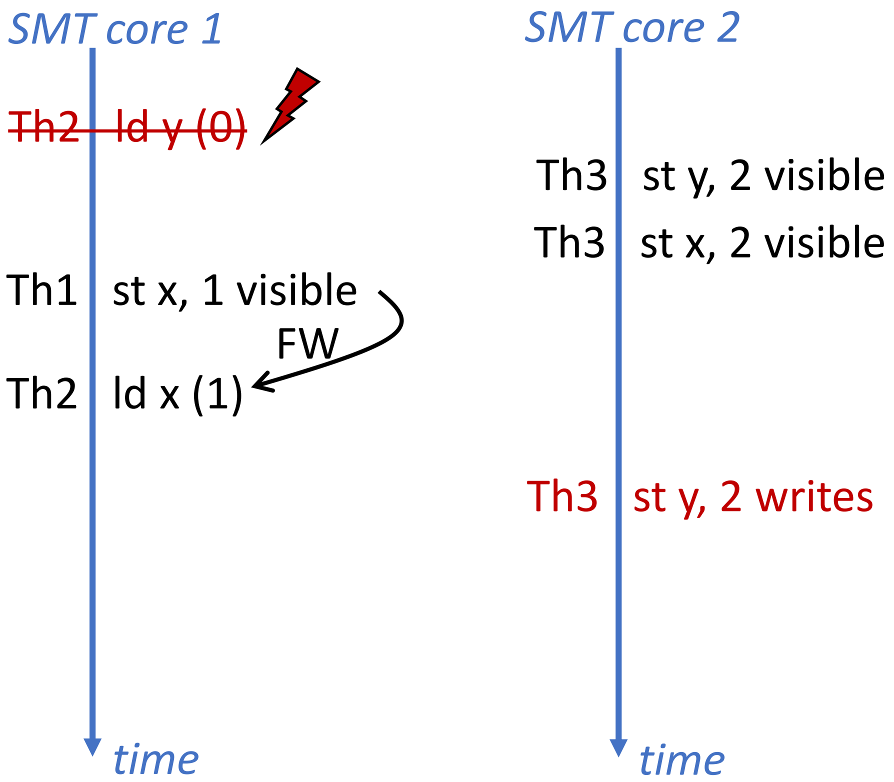
RF: read-from

ITSLF solution

A load receiving forwarded data from a different thread:

i) cannot retire until the forwarding store becomes globally visible

ii) until it retires, it makes all younger loads in its thread speculative and subject to squashing from conflicting stores.


SMT core 1

SMT core 2

~~Th2 ld y (0)~~

Th3 st y, 2 visible

Th3 st x, 2 visible

Th1 st x, 1 visible

FW

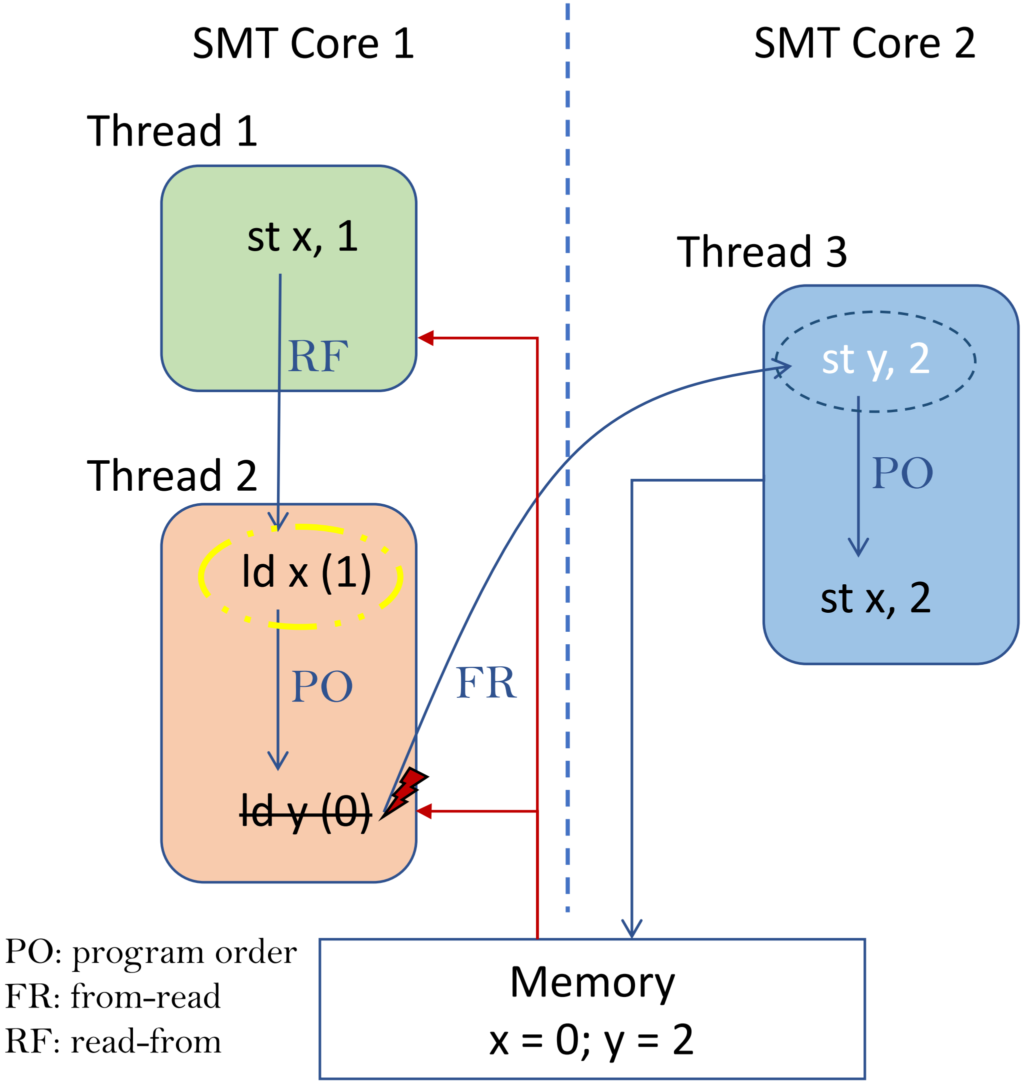
Th2 ld x (1)

Th3 st y, 2 writes

time

time

ITSLF: Multi-Copy Atomicity



ITSLF solution

A load receiving forwarded data from a different thread:

i) cannot retire until the forwarding store becomes globally visible

ii) until it retires, it makes all younger loads in its thread speculative and subject to squashing from conflicting stores.

Cost

ITSLF requires extending each LQ entry with two fields:

i) a single-bit field to indicate if the load was forwarded from a different thread.

ii) a field to store the augmented position of the forwarding store order ($\lceil log_2(SB\ entries) + 1 \rceil$ *bits).*

Outline

- Introduction
- Background
- Issues and Solutions with ITSLF
- Experimental Evaluation
- Conclusion

Experimental evaluation
Setup

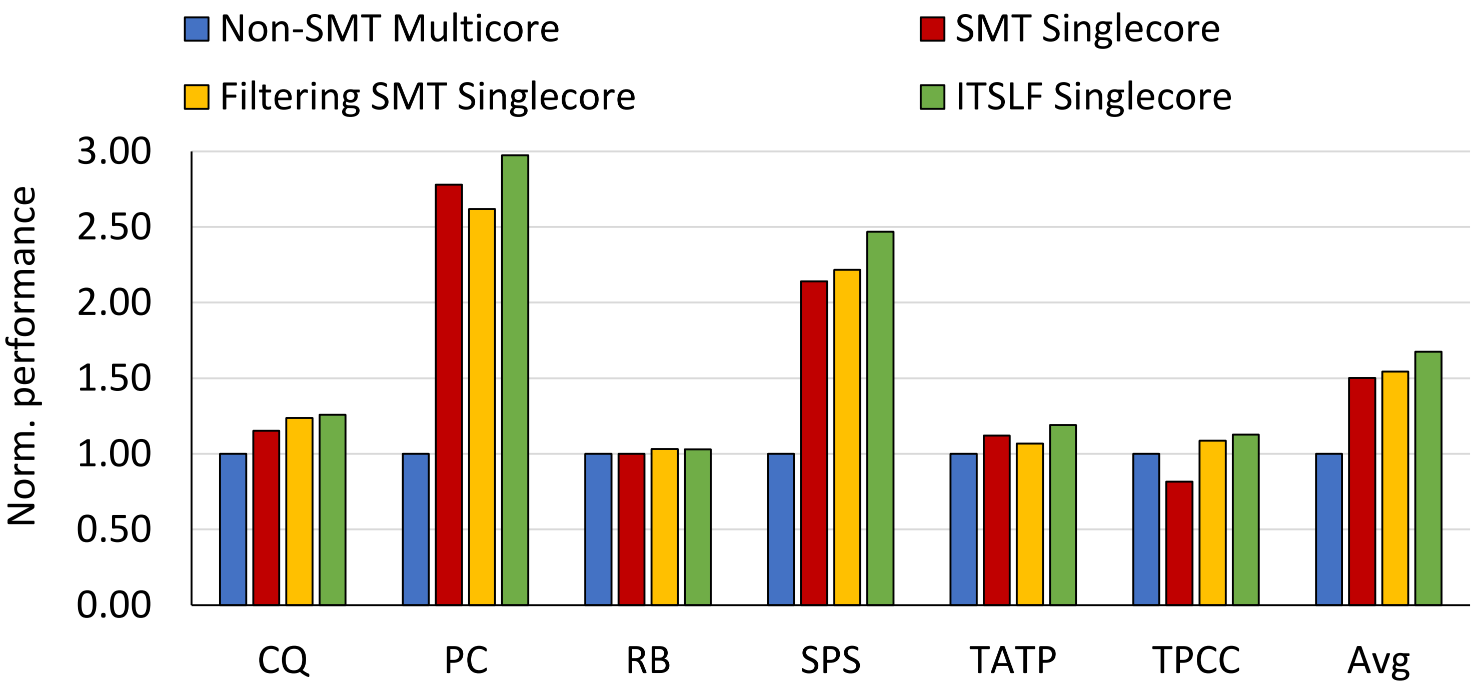
- Ice Lake like SMT multicore.
  - Up to 16 SMT threads with resources statically partitioned among threads.

- Fine-grain, synchronization-intensive, parallel benchmarks:
  - CQ, PC, RB, SPS, TATP, TPCC.

- Synchronization-poor workloads:
  - SPLASH-3 and PARSEC 3.0.

Experimental evaluation

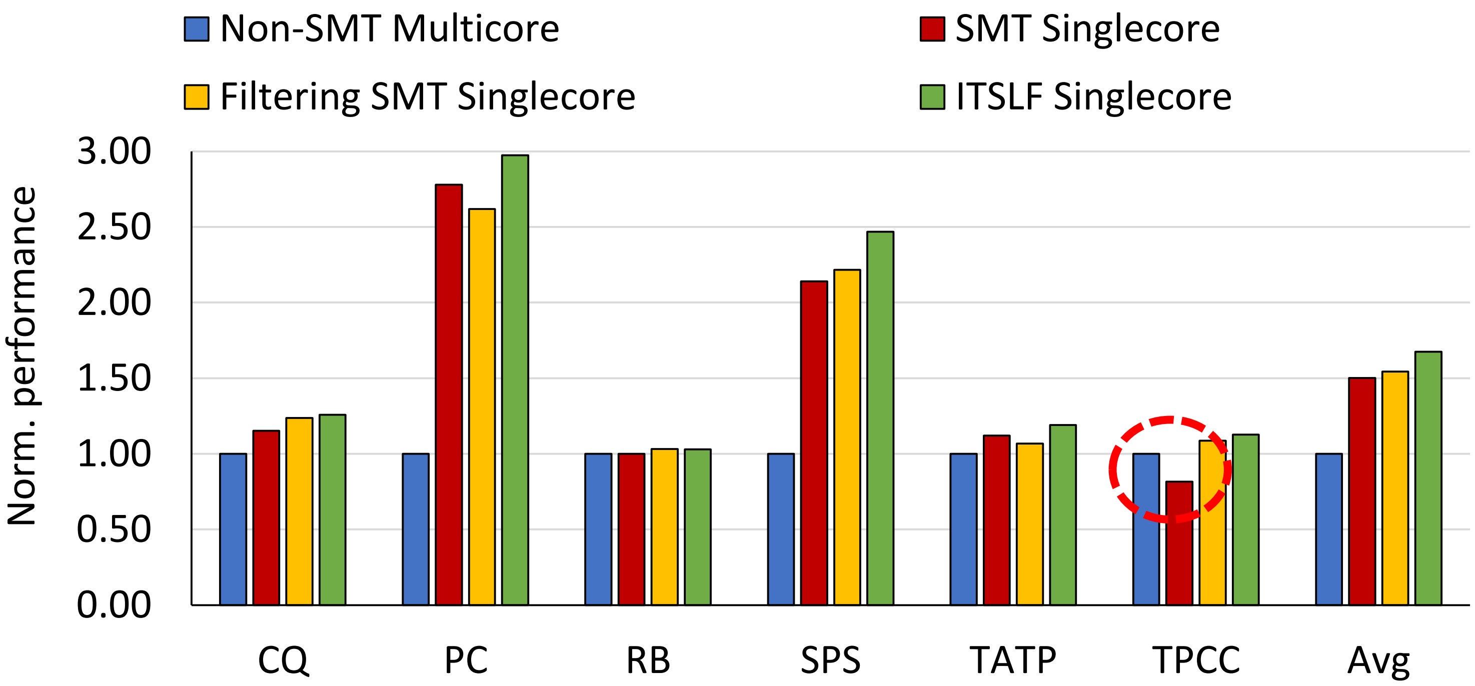
Performance impact of ITSLF in synchronization-intensive workloads



Performance benefit with optimal number of threads for synchronization-intensive workloads compared to the baseline SMT.

Experimental evaluation

Performance impact of ITSLF in synchronization-intensive workloads

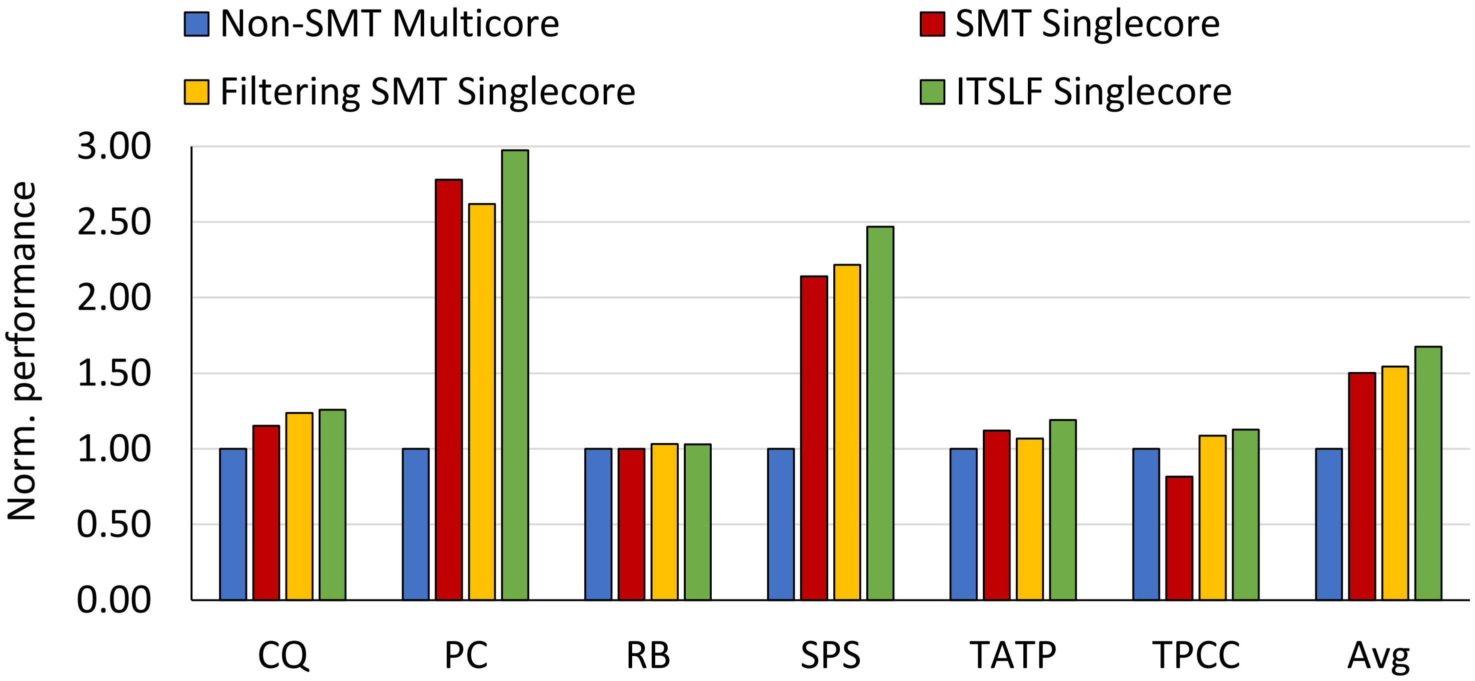


SMT singlecore
not consistently better
than non-SMT multicore

Performance benefit with optimal number of threads for synchronization-intensive workloads compared to the baseline SMT.

Experimental evaluation

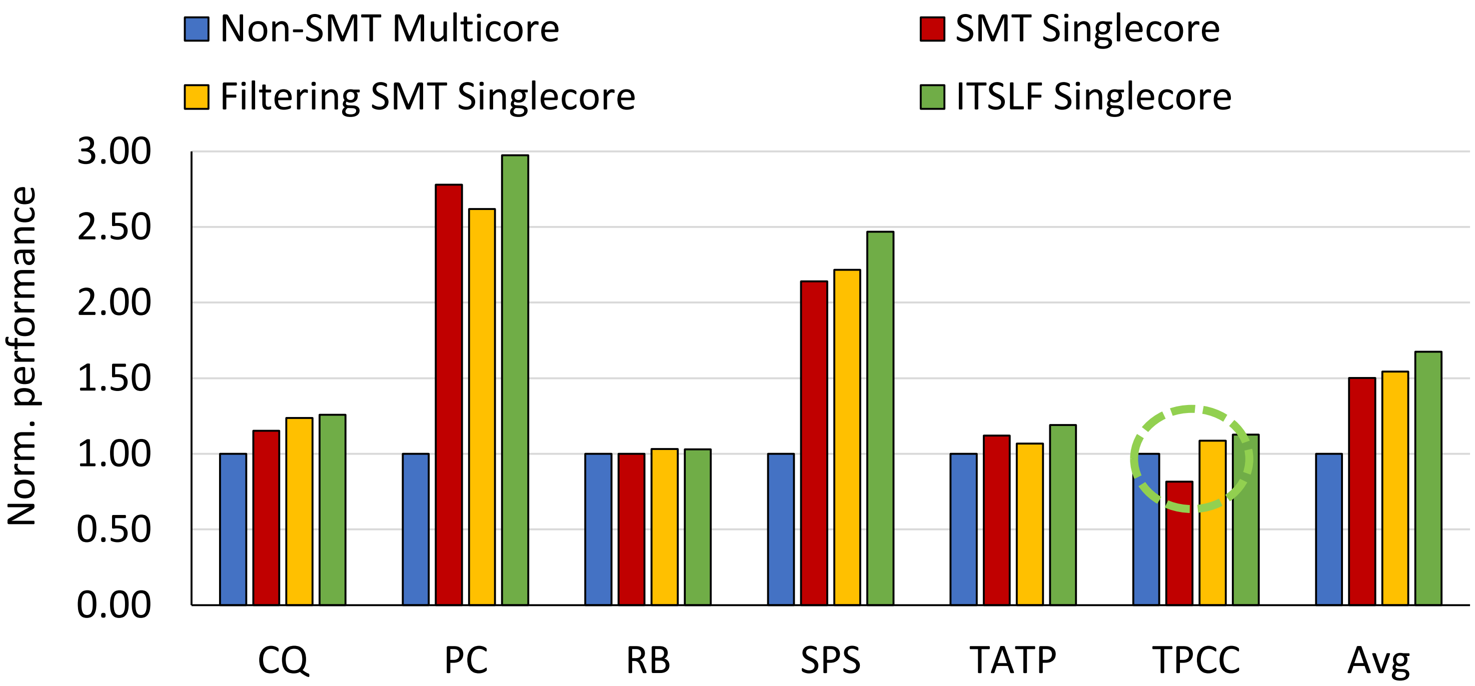
Performance impact of ITSLF in synchronization-intensive workloads



Performance benefit with optimal number of threads for synchronization-intensive workloads compared to the baseline SMT.

Experimental evaluation

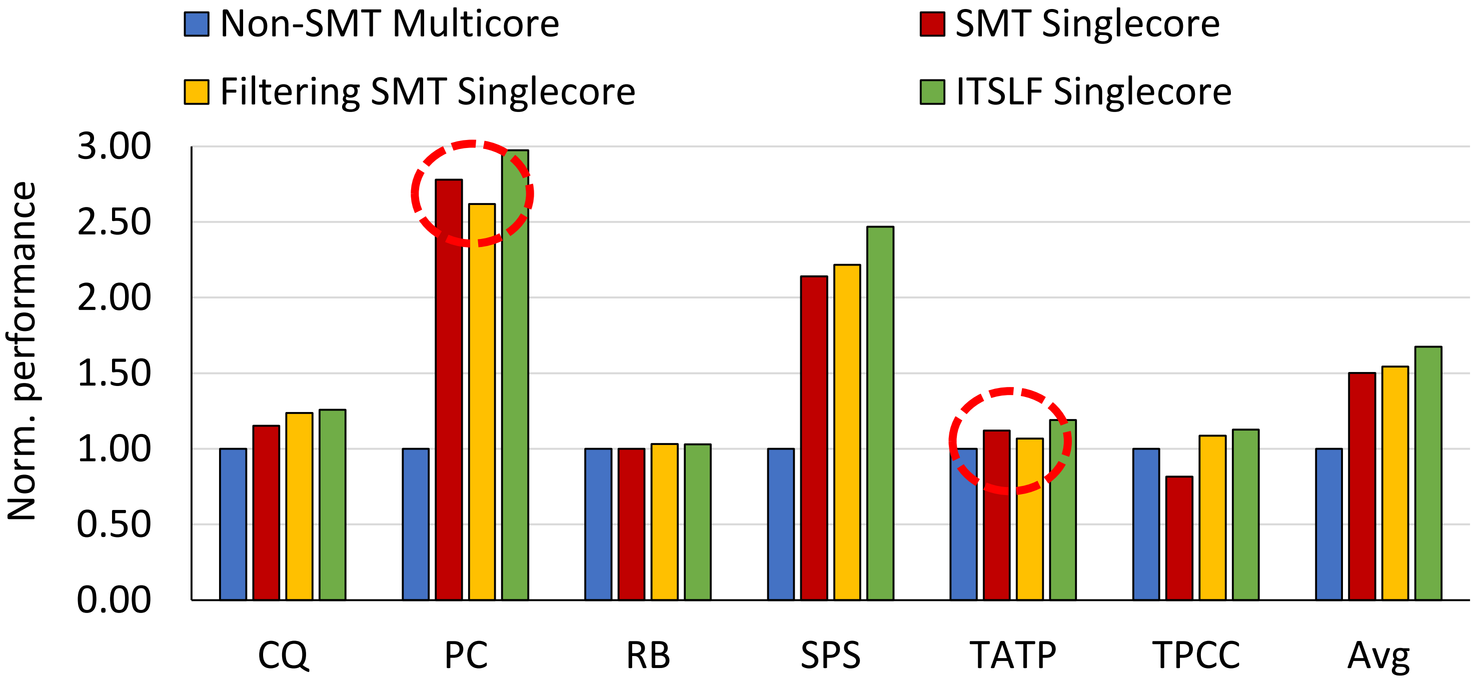
Performance impact of ITSLF in synchronization-intensive workloads



Performance benefit with optimal number of threads for synchronization-intensive workloads compared to the baseline SMT.

Experimental evaluation

Performance impact of ITSLF in synchronization-intensive workloads

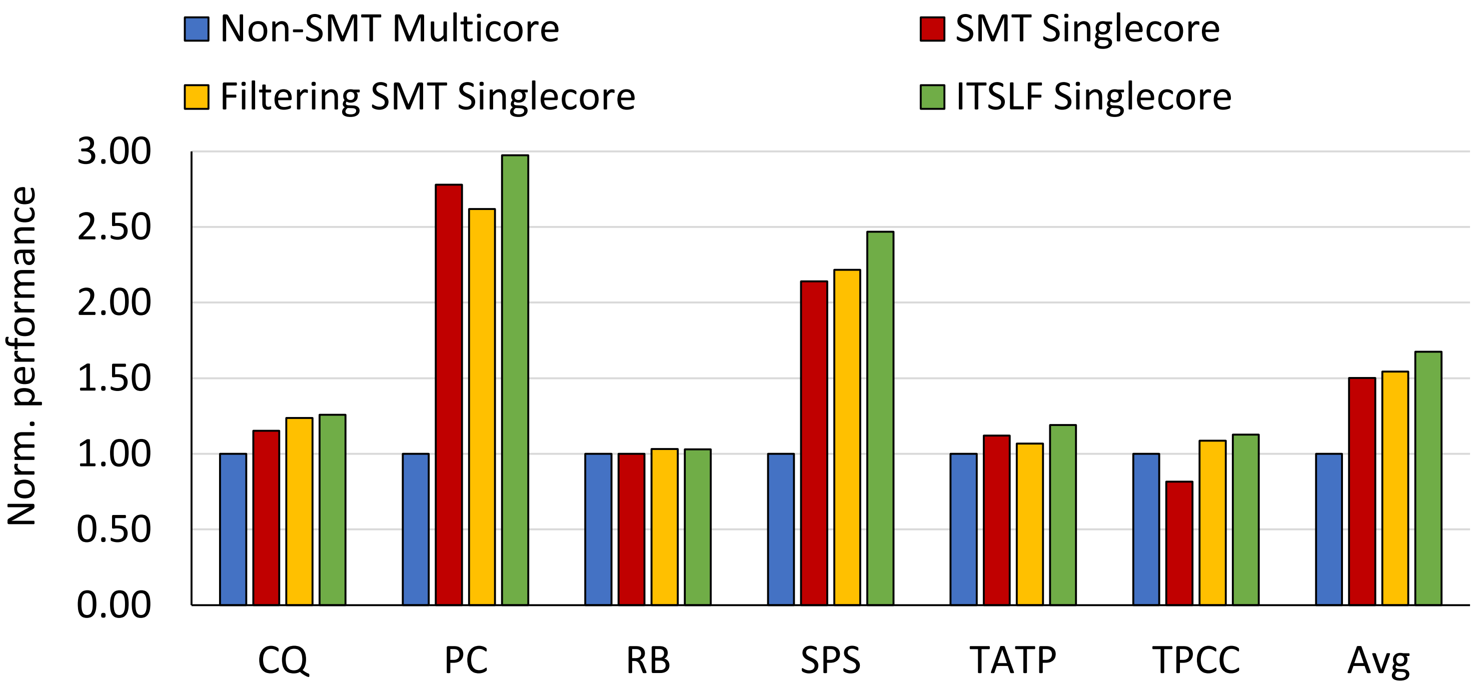


Filtering SMT
not consistently better
than baseline SMT

Performance benefit with optimal number of threads for synchronization-intensive workloads compared to the baseline SMT.

Experimental evaluation

Performance impact of ITSLF in synchronization-intensive workloads

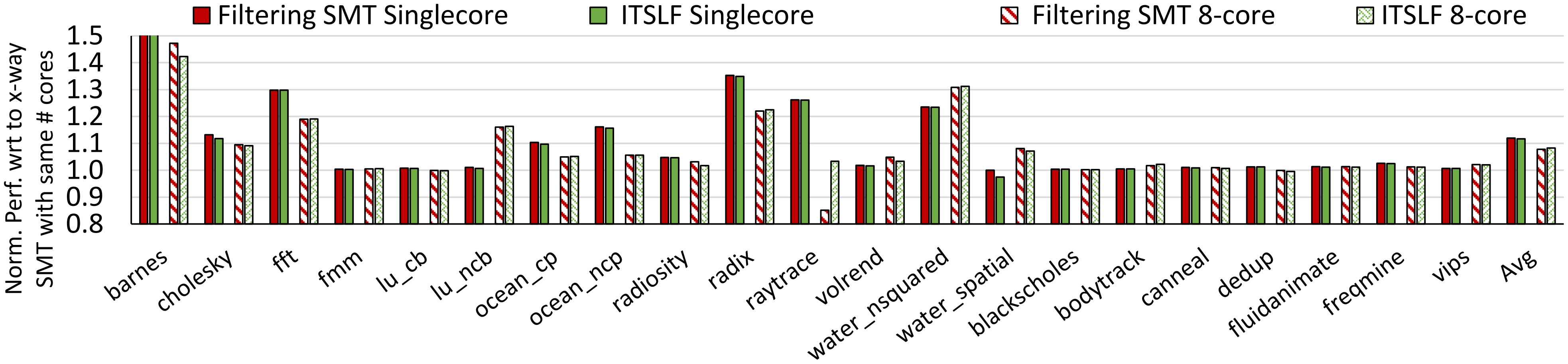


ITSLF outperforms all other setups in all workloads

Performance benefit with optimal number of threads for synchronization-intensive workloads compared to the baseline SMT.

Experimental evaluation

Performance impact of ITSLF in synchronization-poor workloads



Normalized performance compared to the baseline SMT across SPLASH-3 and PARSEC 3.0 workloads.

Conclusion

- We demonstrate that store-to-load forwarding from the SQ/SB of SMT threads is possible without violating MCA.
- We show that synchronization-intensive workloads consistently benefit from ITSLF (13% speedup).
- We show that ITSLF reduces the number of expensive CAM searches to the LQ.

ITSLF: Inter-Thread Store-to-Load Forwarding in Simultaneous Multithreading

Josué Feliu [1], Alberto Ros [1], Manuel E. Acacio [1], and Stefanos Kaxiras [2]

[1] Computer Engineering Department
University of Murcia

[2] Department of Information Technology
Uppsala University

josue.f.p@um.es

MICRO-54 – Session 10B: Microarchitecture II

Thanks for your attention!

This work was supported by the Spanish MCIU and AEI, as well as European Commission FEDER funds, under grant RTI2018-098156-B-C53, the European Research Council (ERC) under the Horizon 2020 research and innovation program (grant agreement No 819134), the Vetenskapsradet project 2018-05254, and the European joint Effort toward a Highly Productive Programming Environment for Heterogeneous Exascale Computing (EPEEC) (grant No 801051). Josué Feliu is supported by a Juan de la Cierva Formación Contract (FJC2018-036021-I).