Enhancing Atomic Instruction Emulation for Cross-ISA Dynamic Binary Translation

Ziyi Zhao, Zhang Jiang, Ying Chen, Xiaoli Gong*
Nankai University

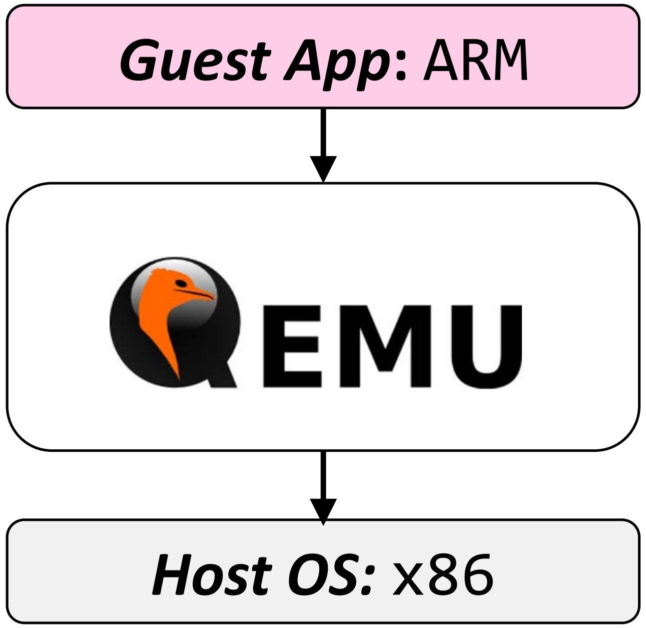
Wenwen Wang
University of Georgia

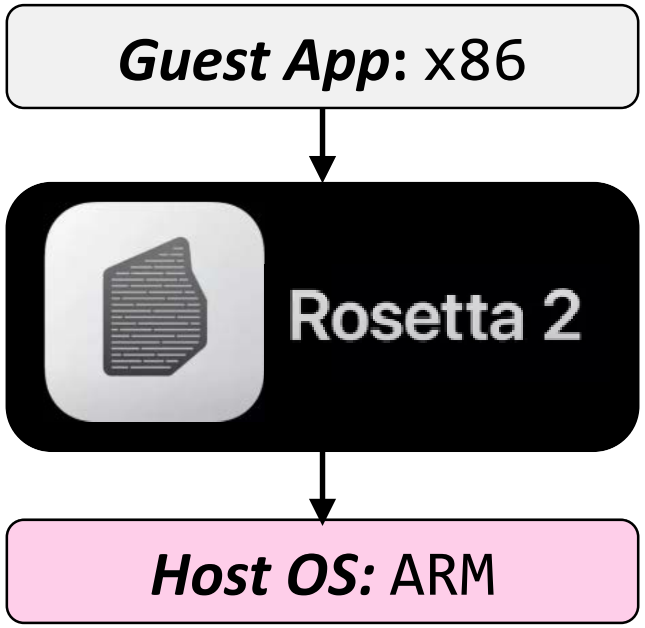
Pen-Chung Yew
University of Minnesota at Twin Cities

International Symposium on Code Generation and Optimization (CGO) 2021

Cross-ISA Dynamic Binary Translation

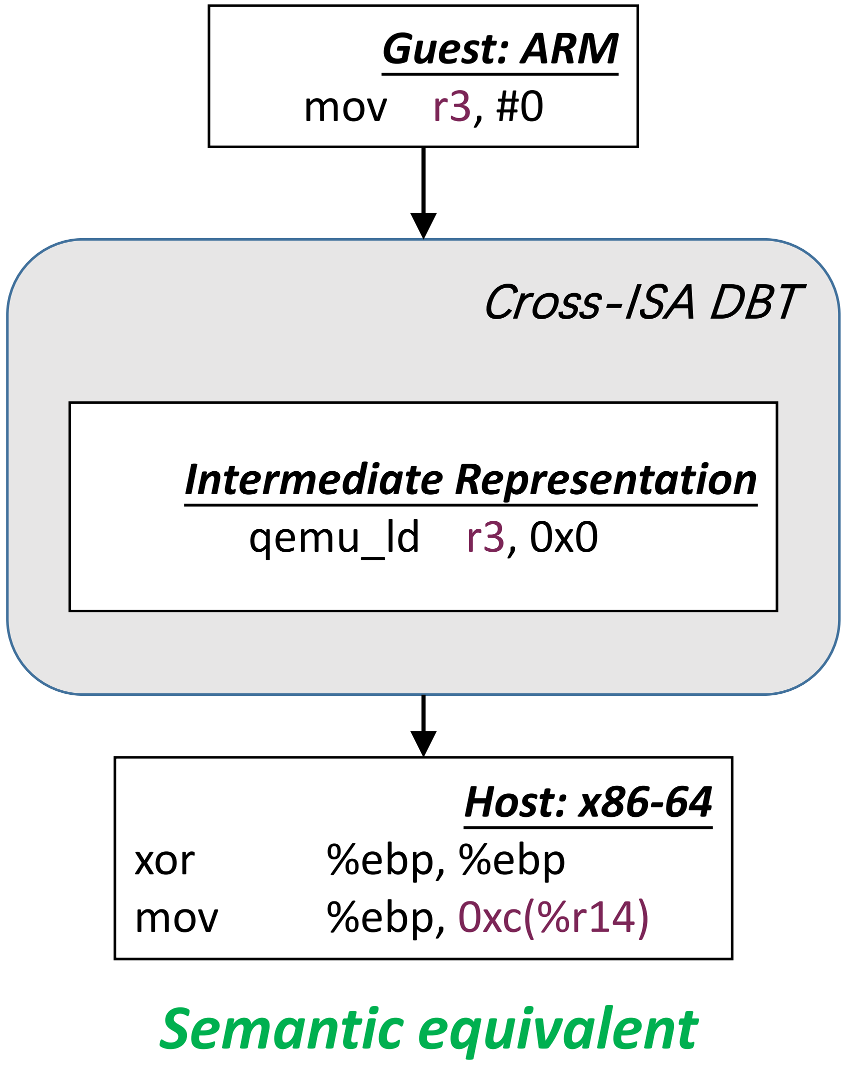
- Cross **I**nstruction **S**et **A**rchitecture **D**ynamic **B**inary **T**ranslator
- A key enabling technology
  - Developing and testing
  - Running application

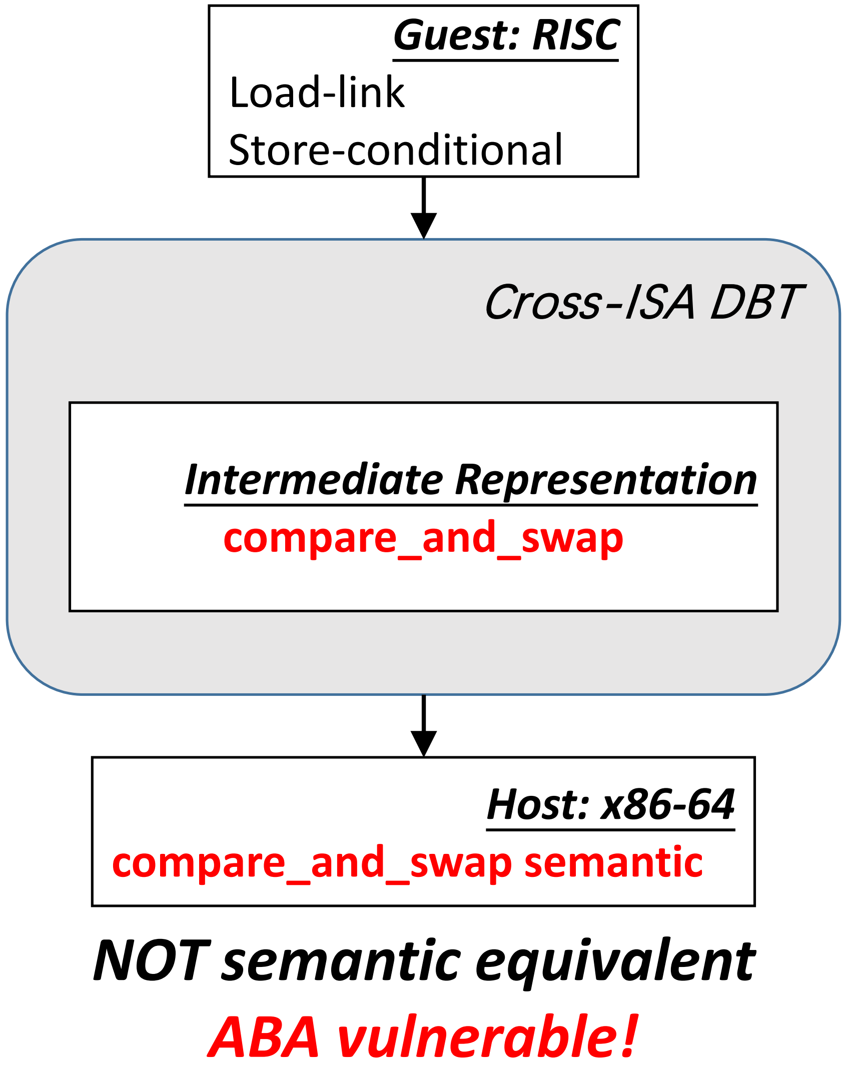




Cross-ISA Dynamic Binary Translation

- Fundamental rule: semantic equivalence





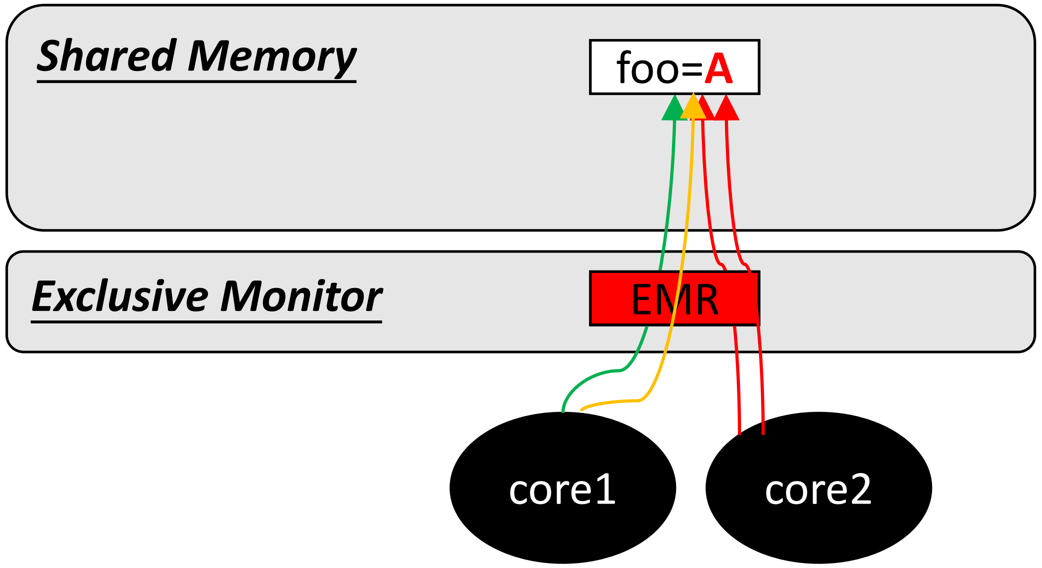
Outline

RISC Atomic Instruction: Load-link/Store-conditional

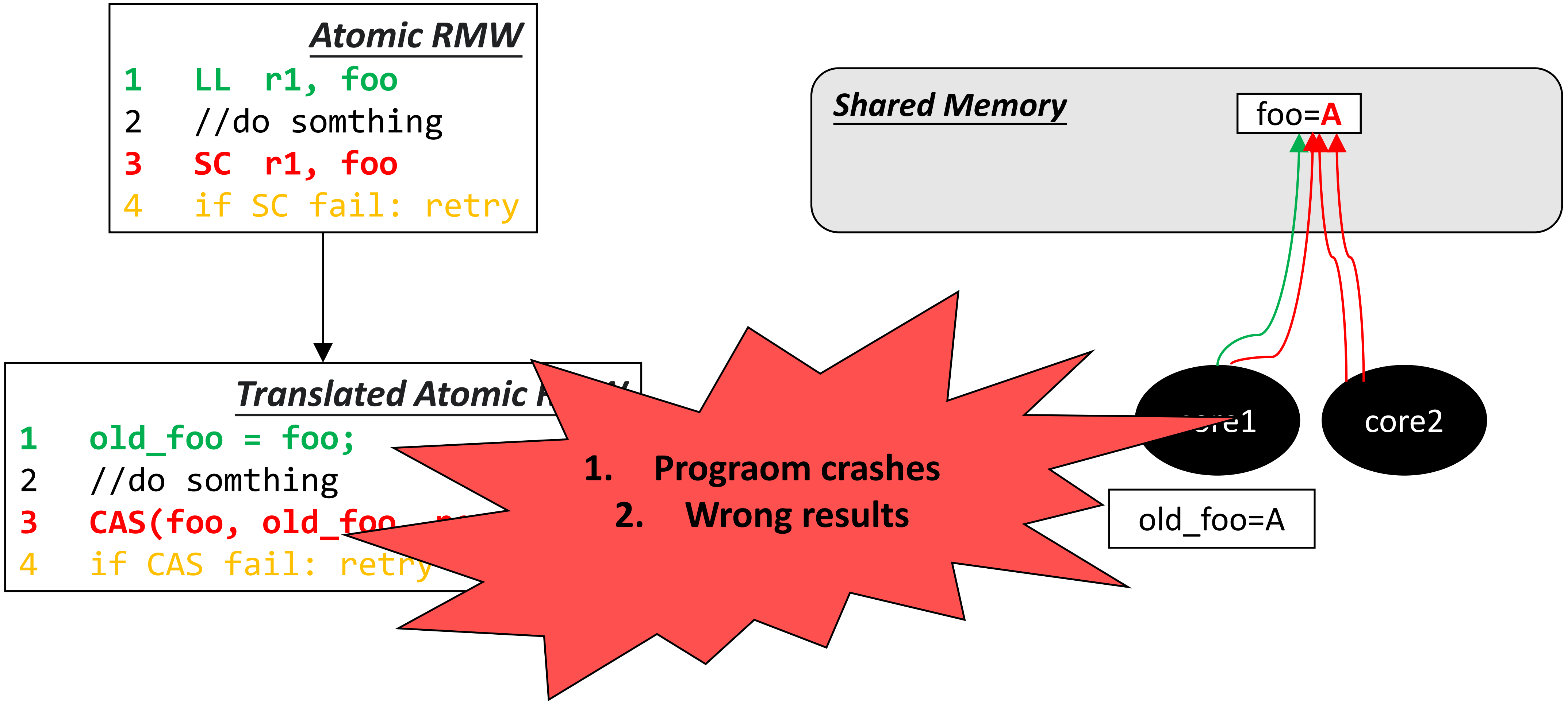
- Load-Link(LL)
  - Set Exclusive Memory Region (EMR)
  - Load data
- Store-Conditional(SC)
  - Check EMR
  - Write data

Atomic RMW

```
1   LL  r1, foo
2   //do somthing
3   SC  r1, foo
4   if SC fail: retry
```



ABA Problem



Atomic RMW

```
1   LL  r1, foo
2   //do somthing
3   SC  r1, foo
4   if SC fail: retry
```

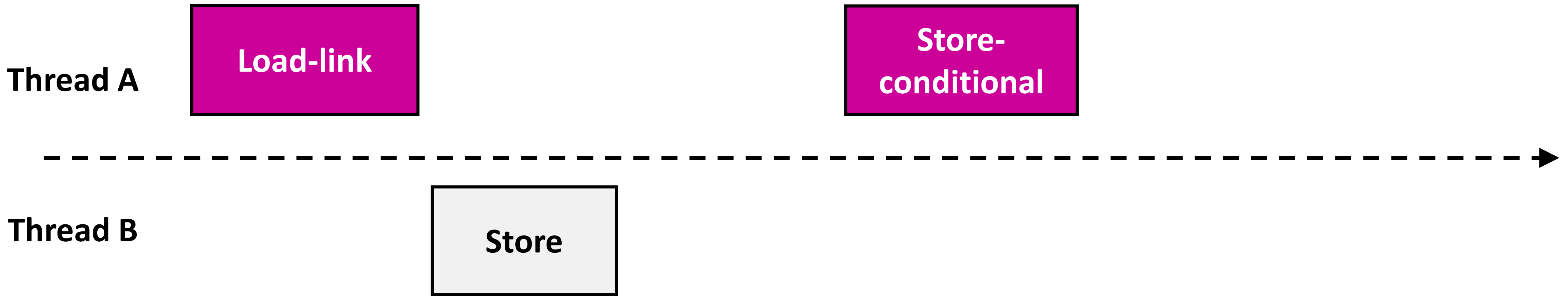
Translated Atomic R

```
1   old_foo = foo;
2   //do somthing
3   CAS(foo, old_foo, n
4   if CAS fail: retry
```

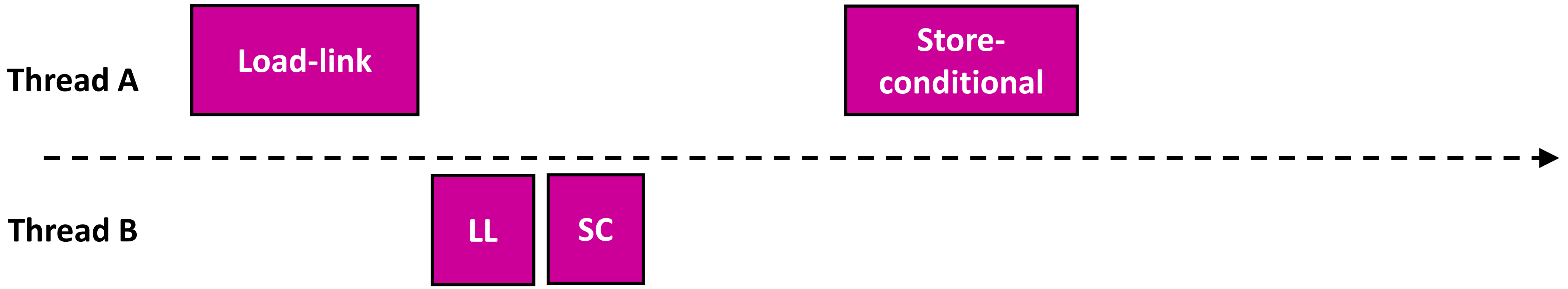
1. Prograom crashes
2. Wrong results

Design Goal: Atomicity of LL/SC

- Strong atomicity

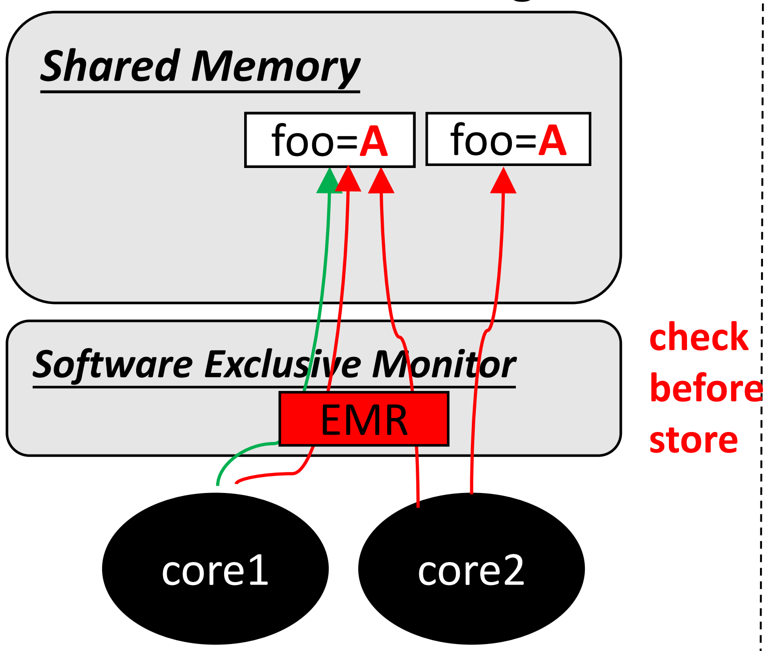


- Weak atomicity

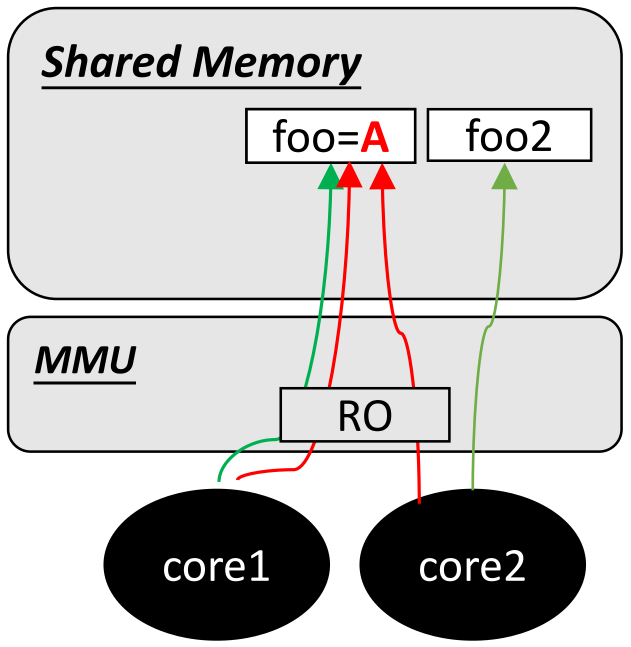


Previous Ideas to Correctly Emulate LL/SC

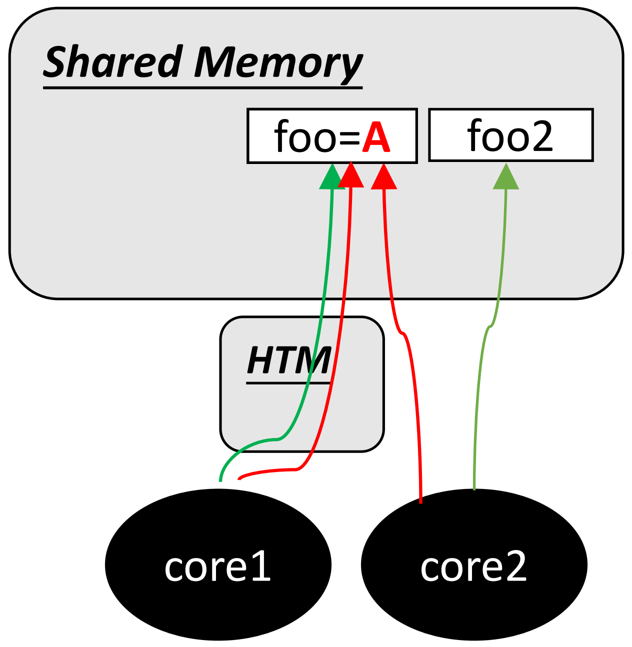
Software Checking



Let MMU to do the check



Hardware Transaction Memory



All stores are instrumented!! HUGE overhead

Only stores to the RO region are intercepted

BUT REALLY?

Fast and correct!

Can be fast with lock-free hash table

But false sharing + large syscall overhead!

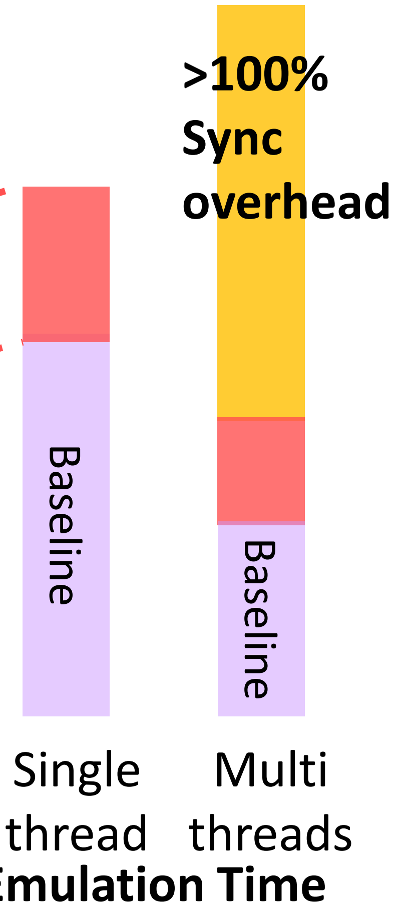
Could be challenging to implement in DBT!

A. HST: Hash-table Based Store Test

- Challenge: store instrumentation overhead
  - store instructions account for up to 20% instructions, highly concurrent
  - **Be Fast**: lightweight checking procedure
  - **Be Scalable**: no locking
- Solution: Lock-free hash table

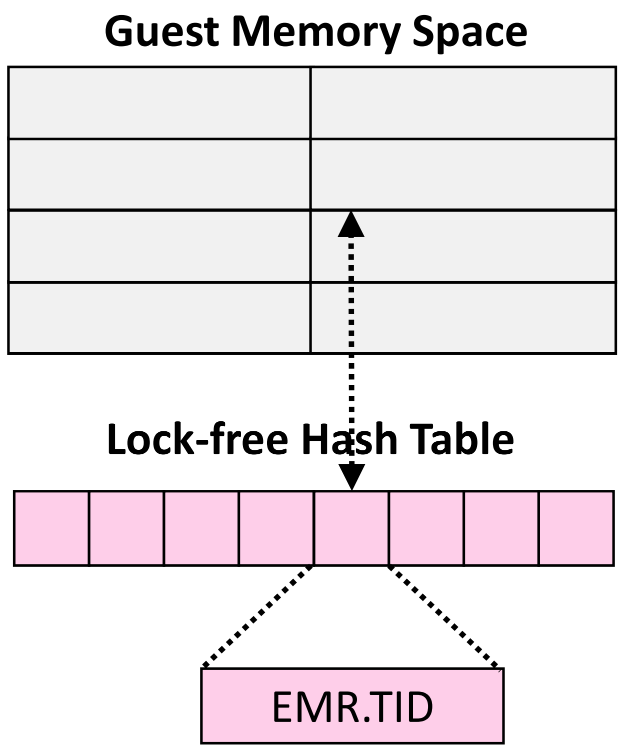
Translated Store

```
1   find_EMR(mem)
2   lock
3   update_EMR
4   unlock
5   store   reg, mem
```



A. HST: Hash-table Based Store Test

- Challenge: store instrumentation overhead
- Solution: Lock-free hash table



Hash table set

EMR.TID←TID

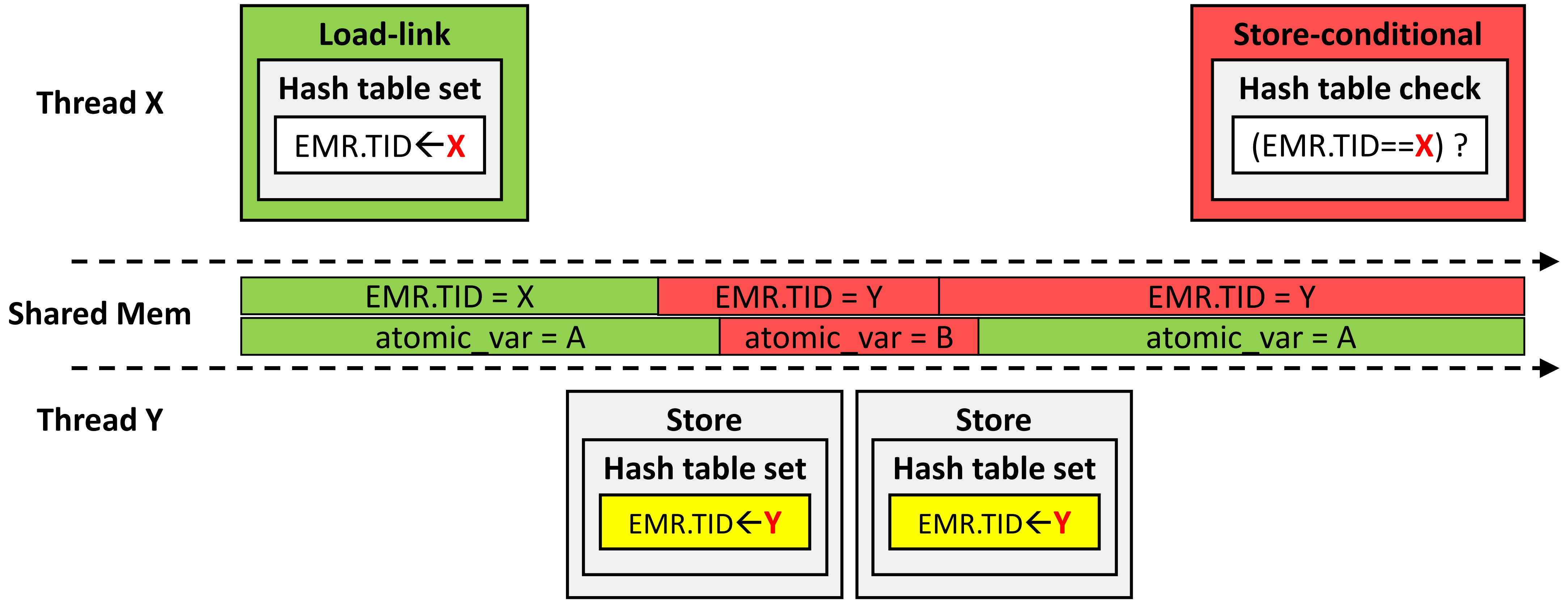
Writing to aligned 4 bytes is guaranteed atomic

Hash table check

(EMR.TID==TID) ?

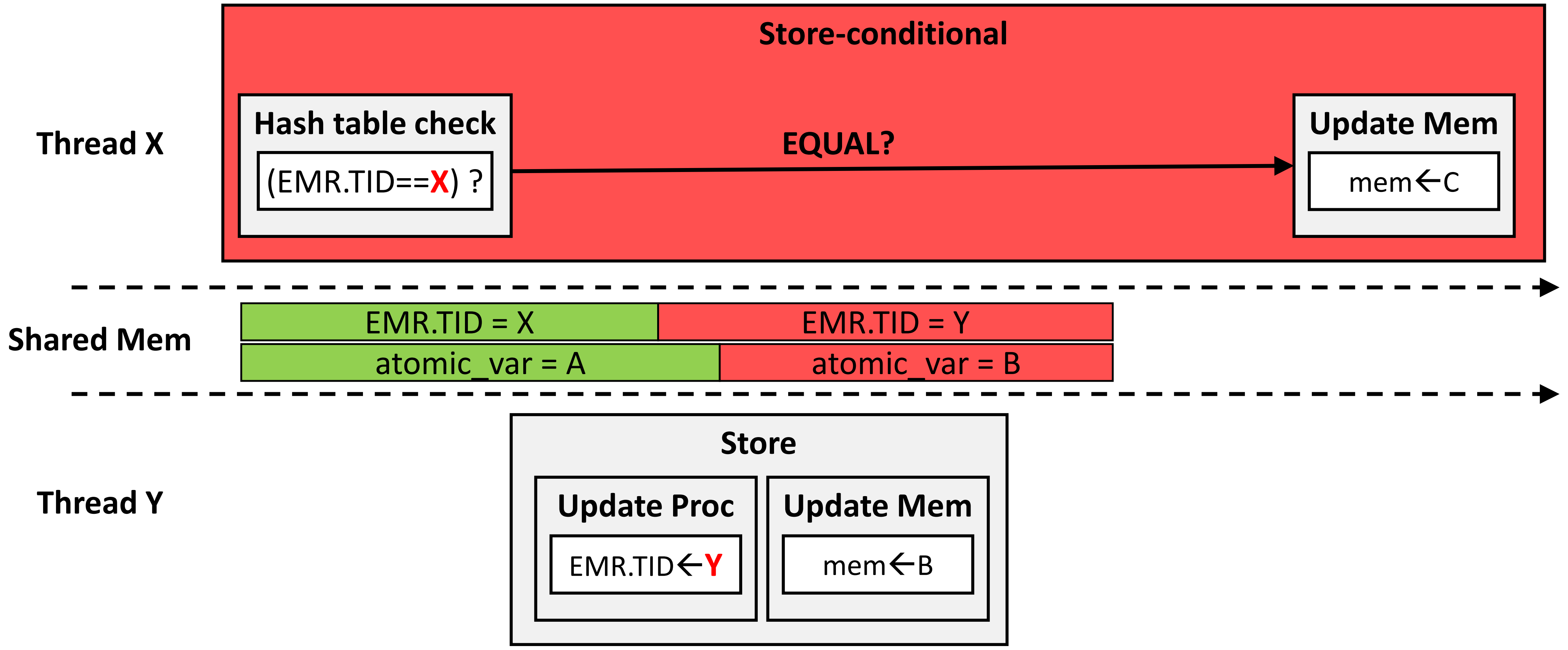
A. HST: Hash-table Based Store Test

- Challenge: store instrumentation overhead
- Solution: Lock-free hash table



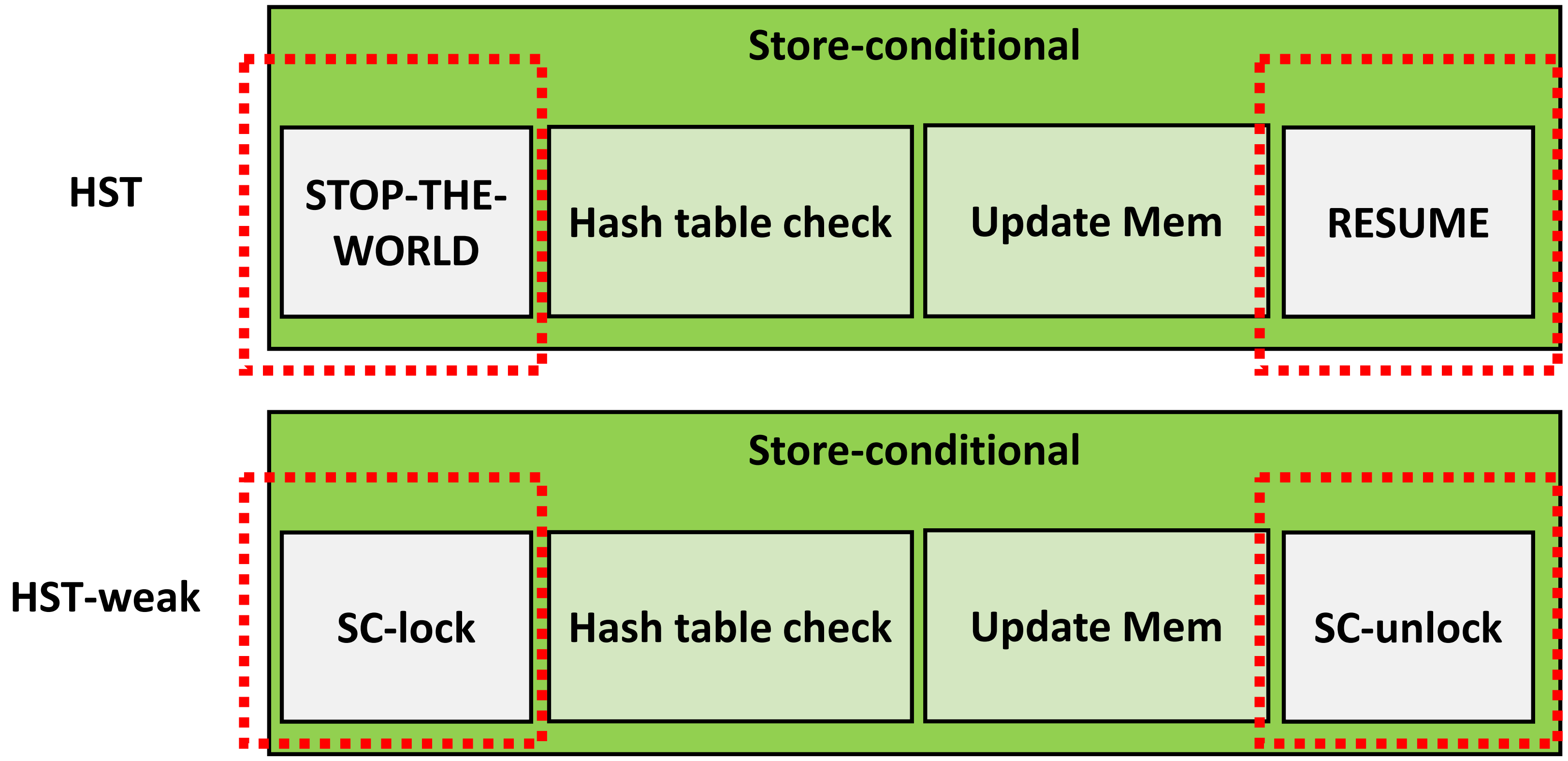
A. HST: Hash-table Based Store Test

- Challenge: race condition between SC and store



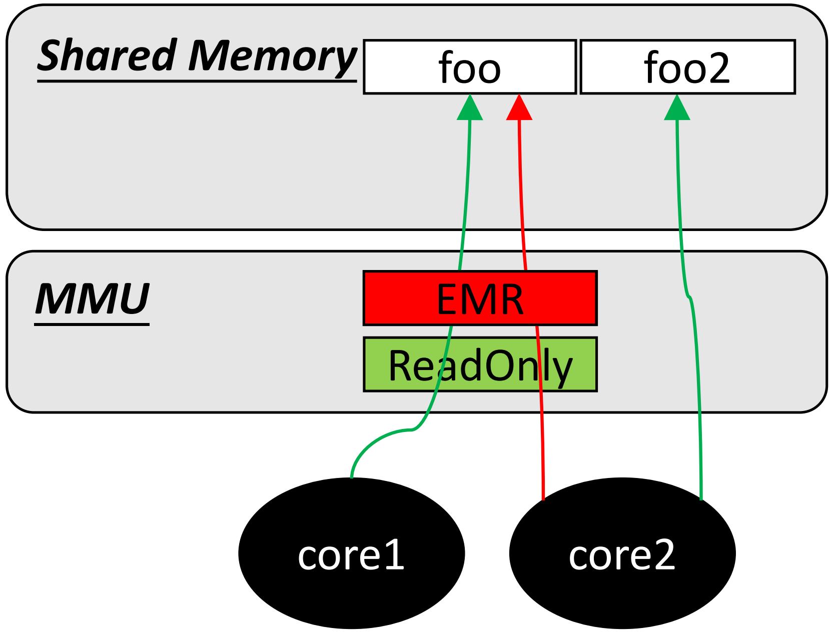
A. HST: Hash-table Based Store Test

- Solution: Guaranteeing the atomicity of SC
  - HST: stop-the-world → strong atomicity
  - HST-weak: SC-mutex-lock → weak atomicity



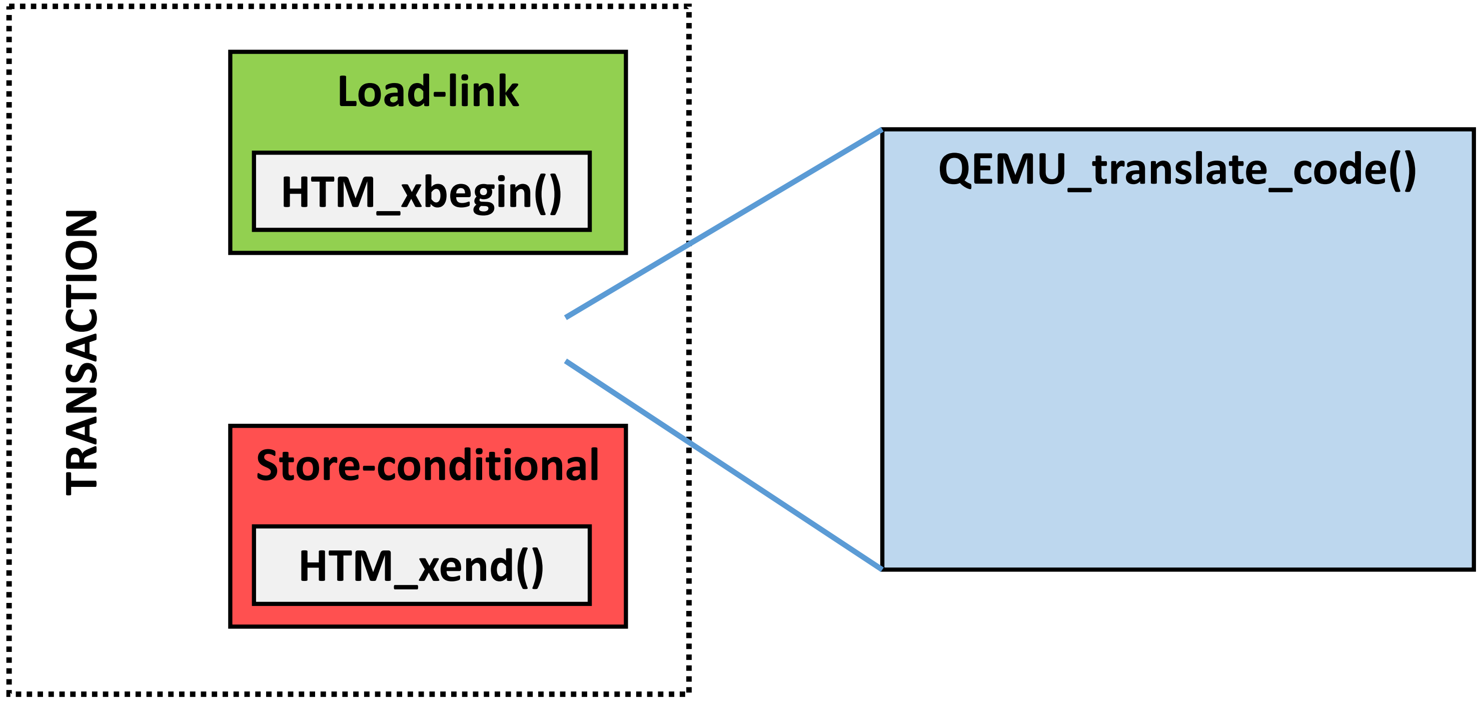
B. PST: Page Protection Based Store Test

- Key idea: Use MMU to automatically check stores
  - No store instrumentation
- Store-conditional atomicity
  - PST: stop-the-world
  - PST-remap: remapping virtual page to achieve privilege separation



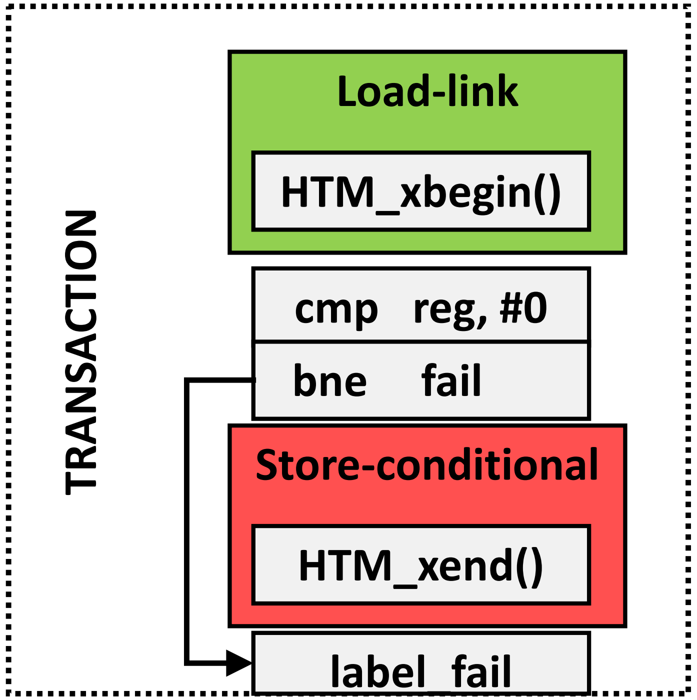
C. HTM: Hardware Transactional Memory

- Map the LL/SC to a transaction ?
  - Code translation between emulation → large footprint!



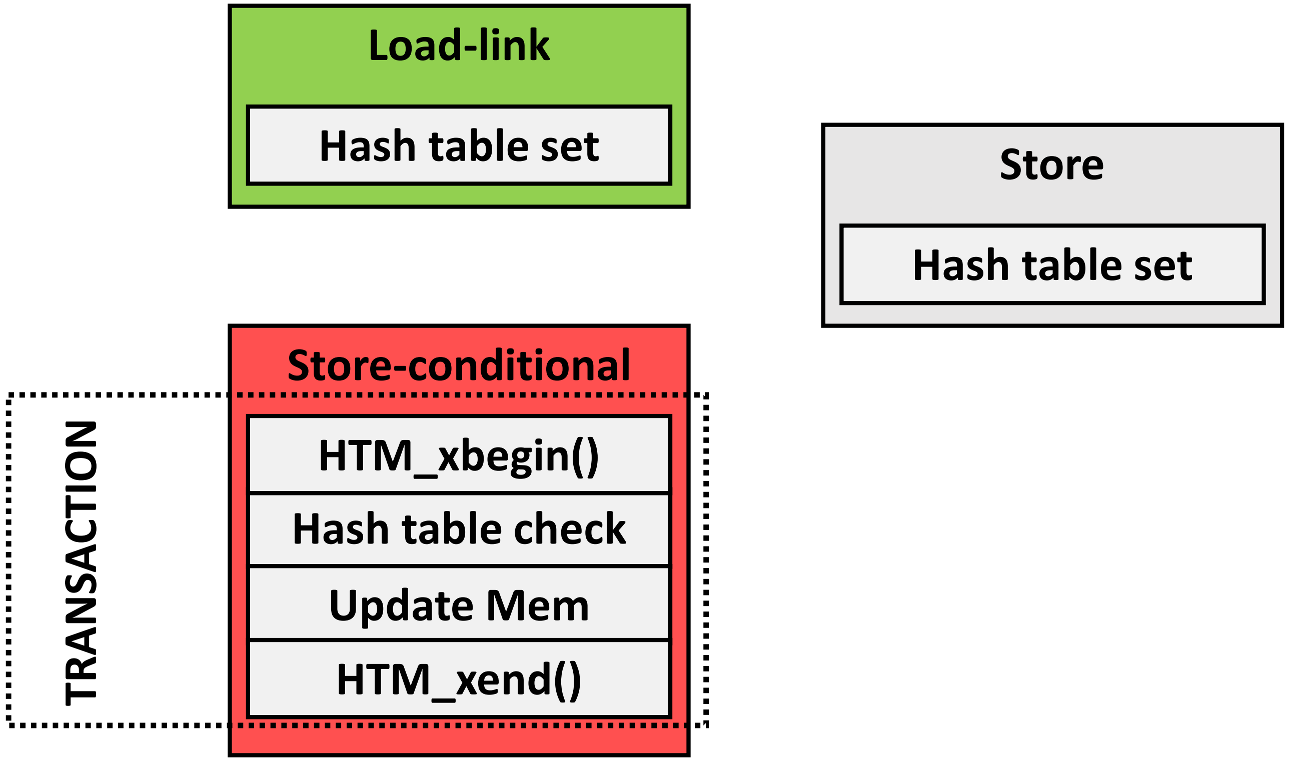
C. HTM: Hardware Transactional Memory

- Map the LL/SC to a transaction ?
  - Code translation between emulation→ large footprint!
  - SC could be jumped over→ No one ends transaction!

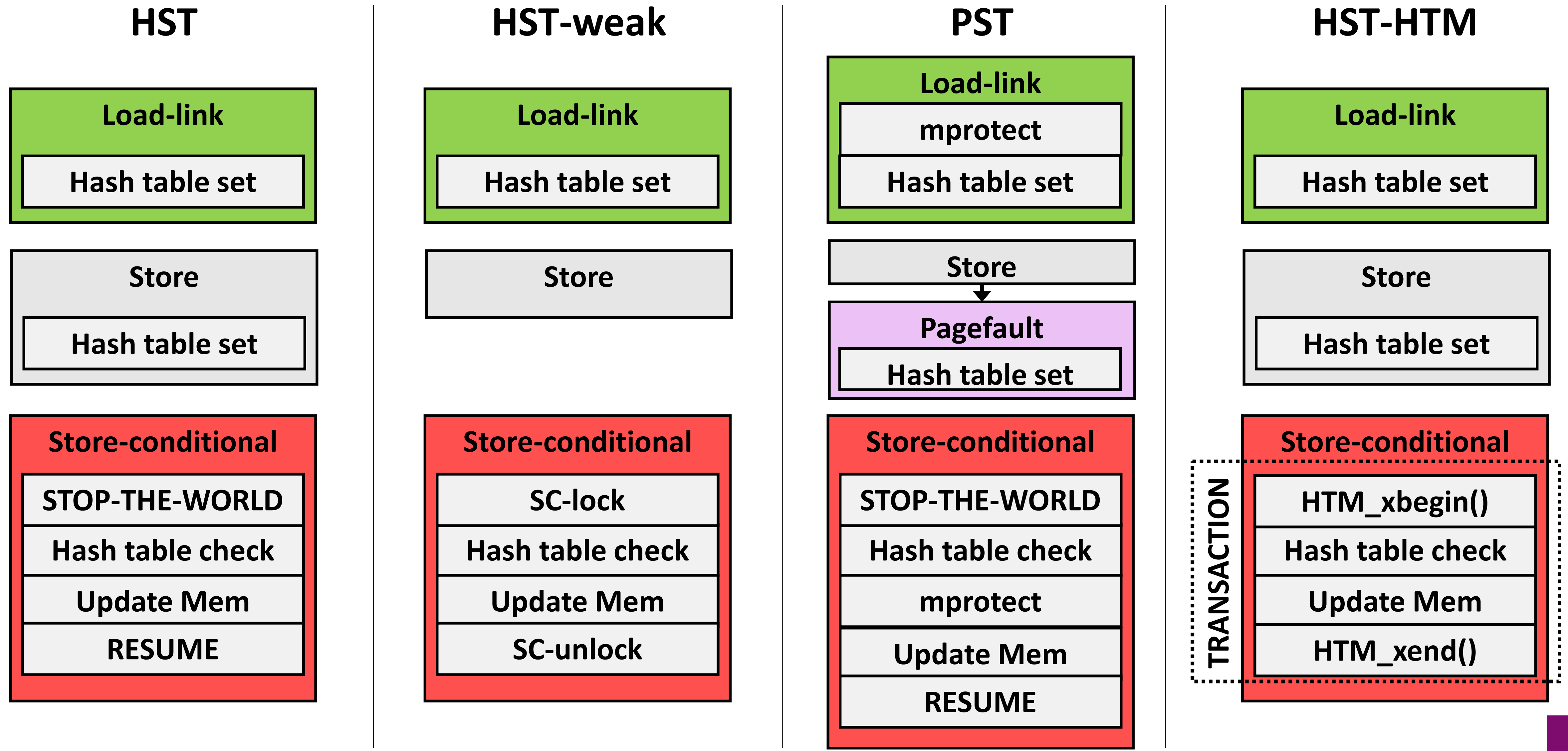


C. HST-HTM

- Solution: Combining HST and HTM together
  - Small footprint
  - HTM_xend() is guaranteed



Summary of Design



Questions to Answer for Evaluation

- Conventional assumptions
  - Save old value and check on store (QEMU) – Fast but incorrect?
  - LL/SC via MMU (PST) – Fast because not instrumenting all stores?
  - Link helpers and instrumented stores (Pico-ST, HST) - Big performance impact by having a helper run for every store?
- Are the schemes scalable in multi-threaded programs?
- What is the performance bottleneck?
- Best trade-off between correctness, speed, and portability?

Setup

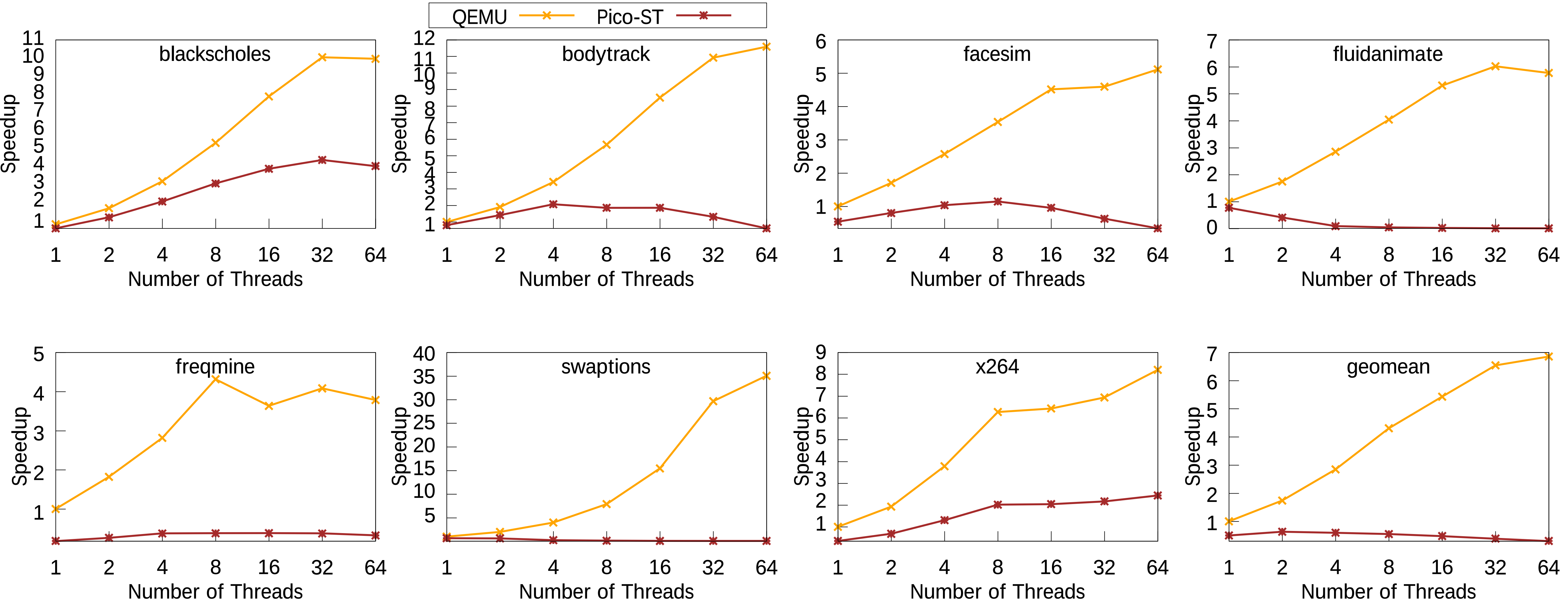
- Running *ARM* PARSEC benchmark suite on *x86* machine
  - PARSEC version 3.0
  - Input size: *simlarge*
- Machine A
  - 52 cores + 183GB Memory
- Machine B: Intel TSX support
  - 10 cores + 64GB Memory

Correctness

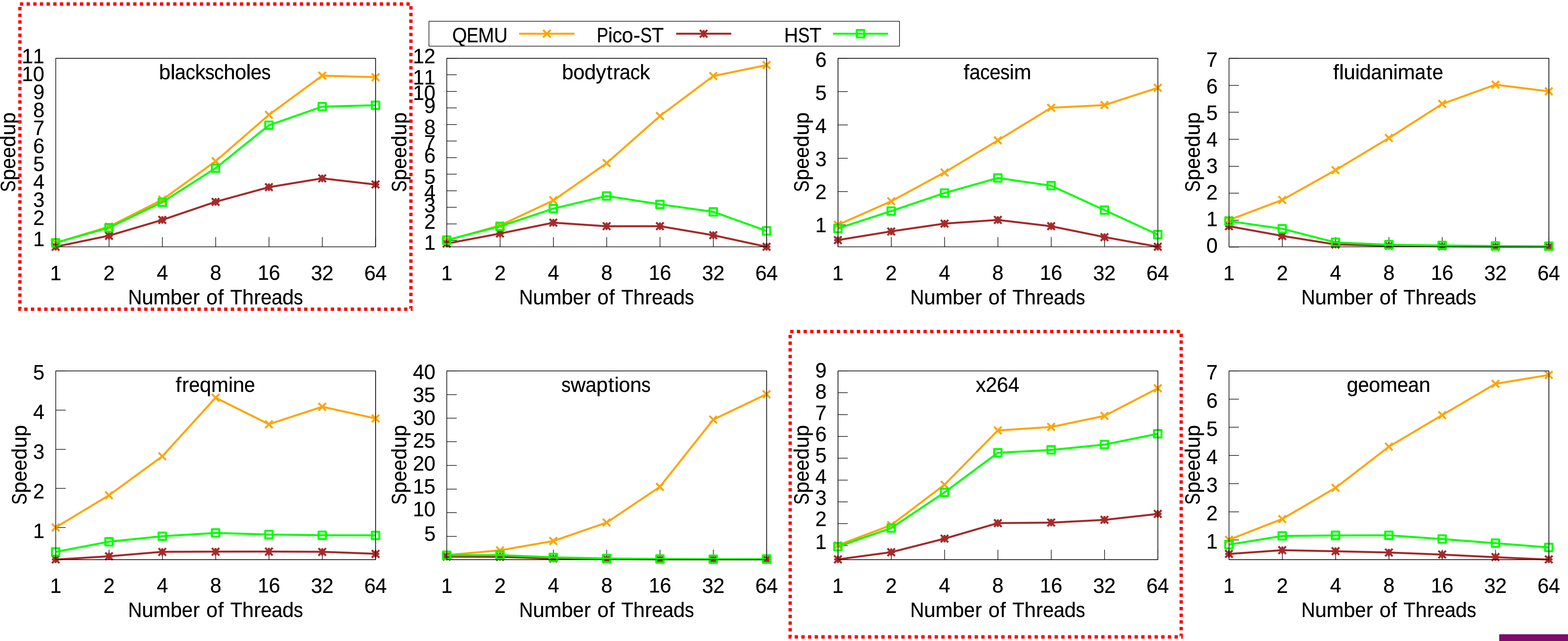
- Formula proof (details in paper)
- Experimental proof
  - A well-designed ARM lock-free stack https://github.com/NKU-EmbeddedSystem/lock-free-stack-arm-asm

Native ARM	QEMU 4.1	Pico-ST	HST	HST-weak	PST	PST-remap	HST-HTM	HTM
Pass	Crash	Pass	Pass	Pass	Pass	Pass	Pass	livelock

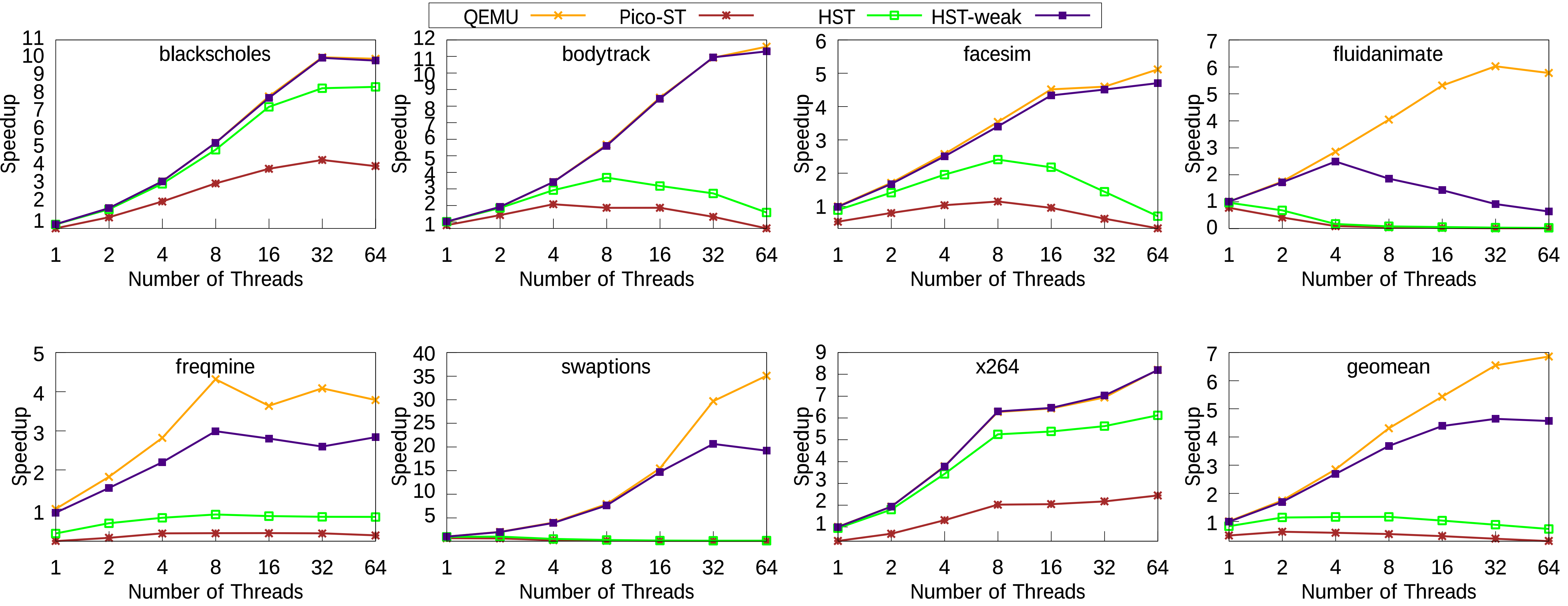
Scalability – Portable Solutions



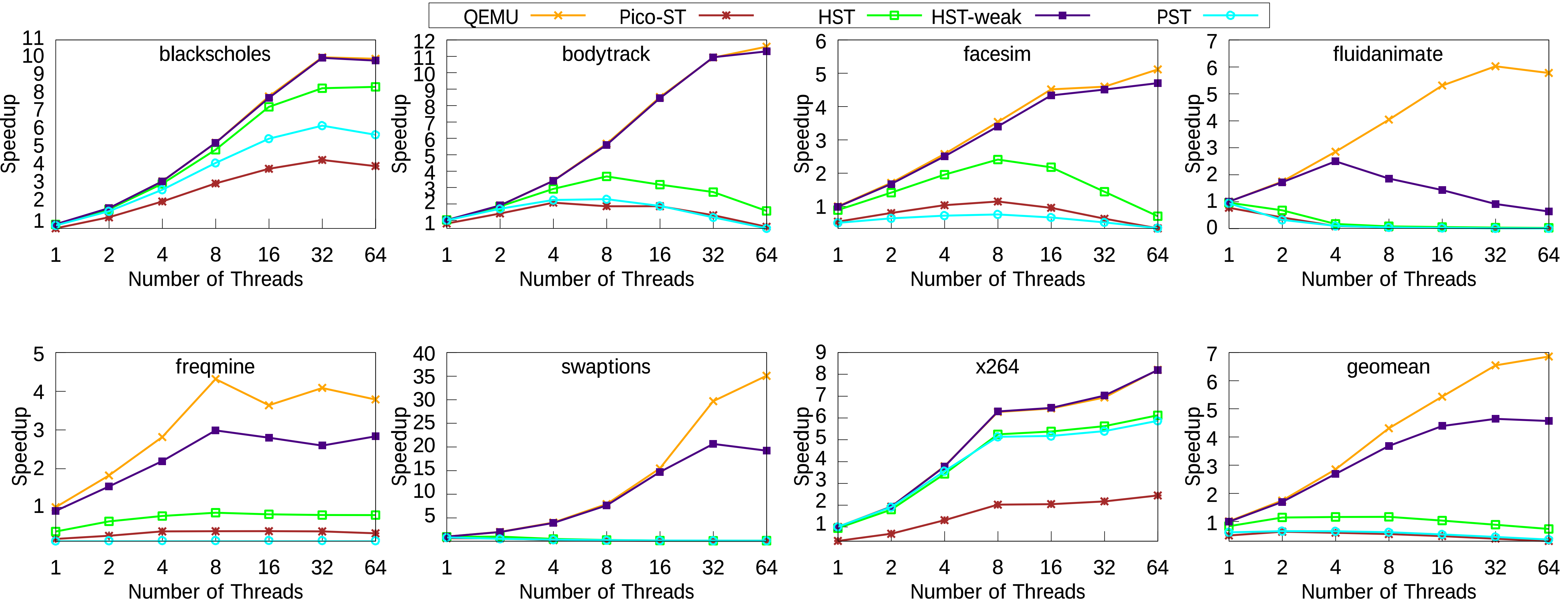
Scalability – Portable Solutions



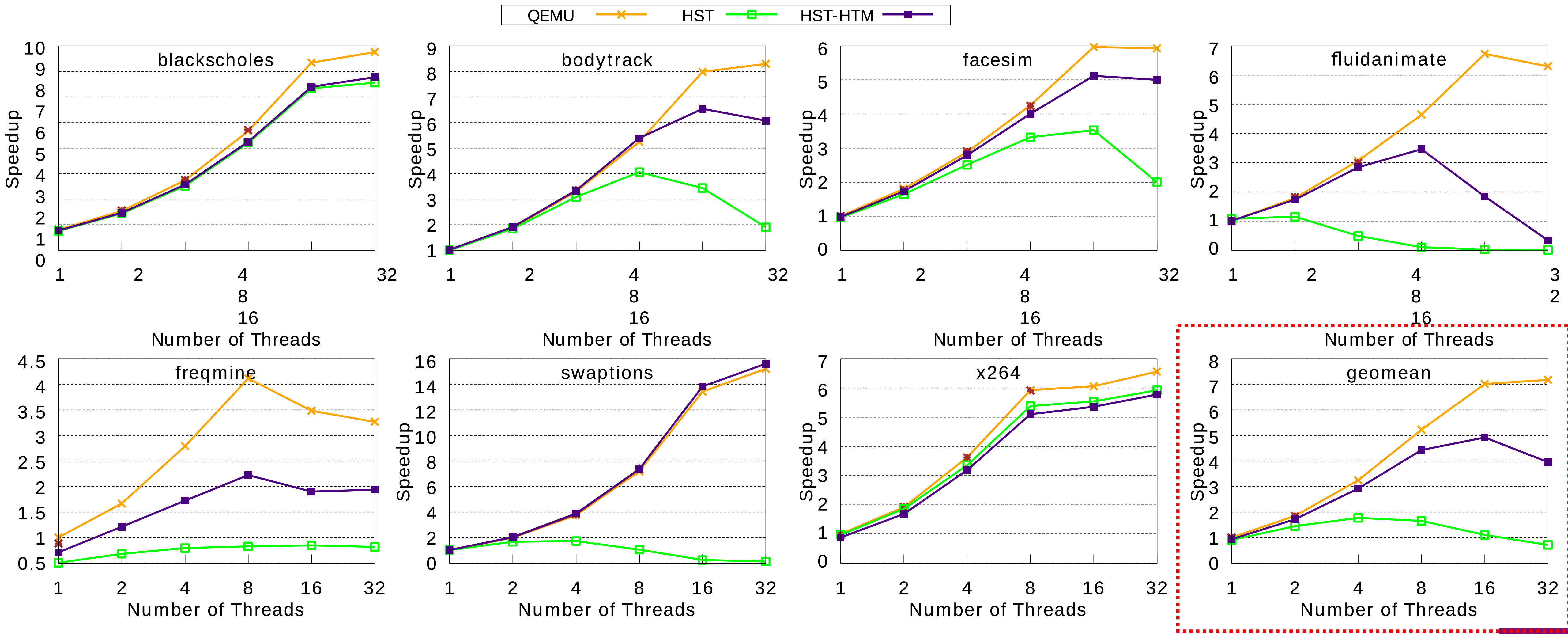
Scalability – Portable Solutions



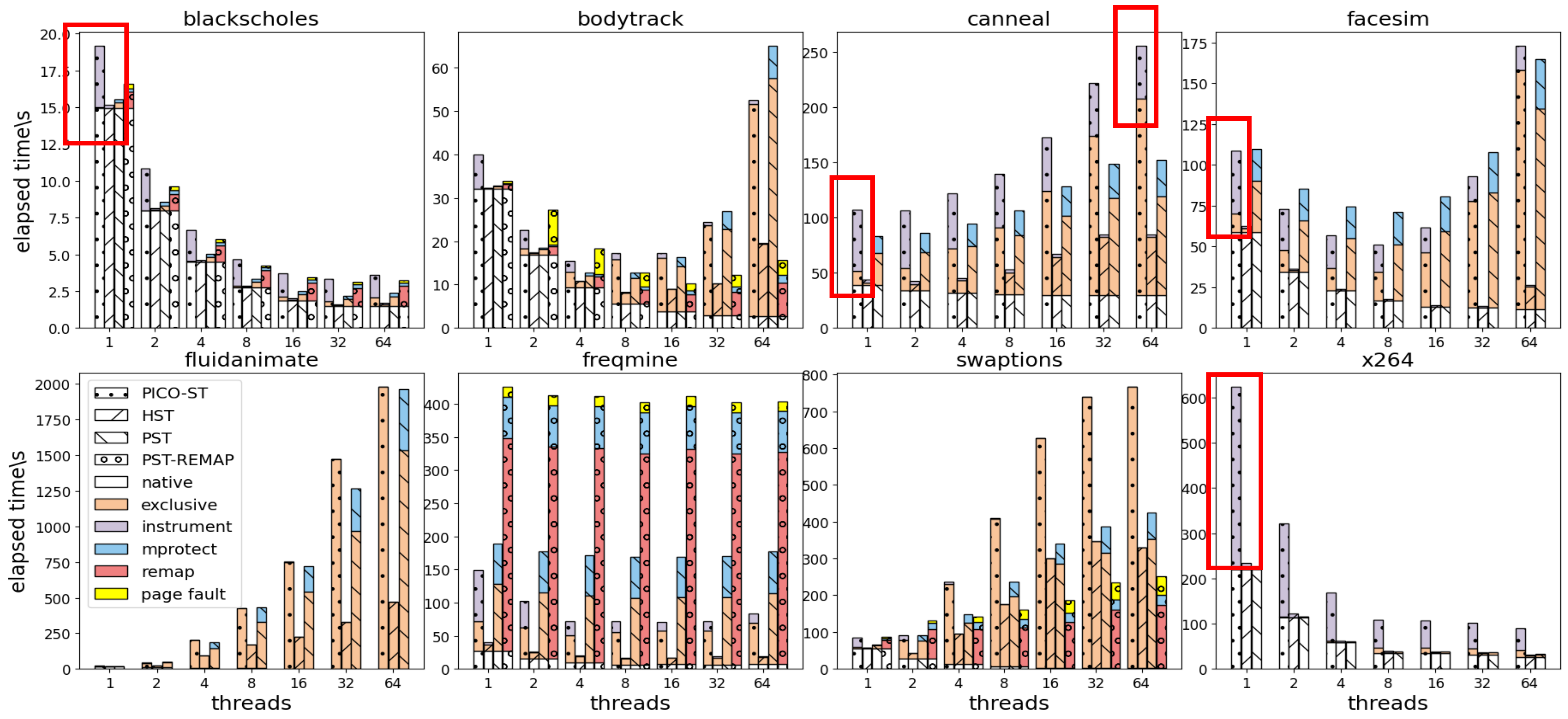
Scalability – Portable Solutions



Scalability – HTM based Solutions

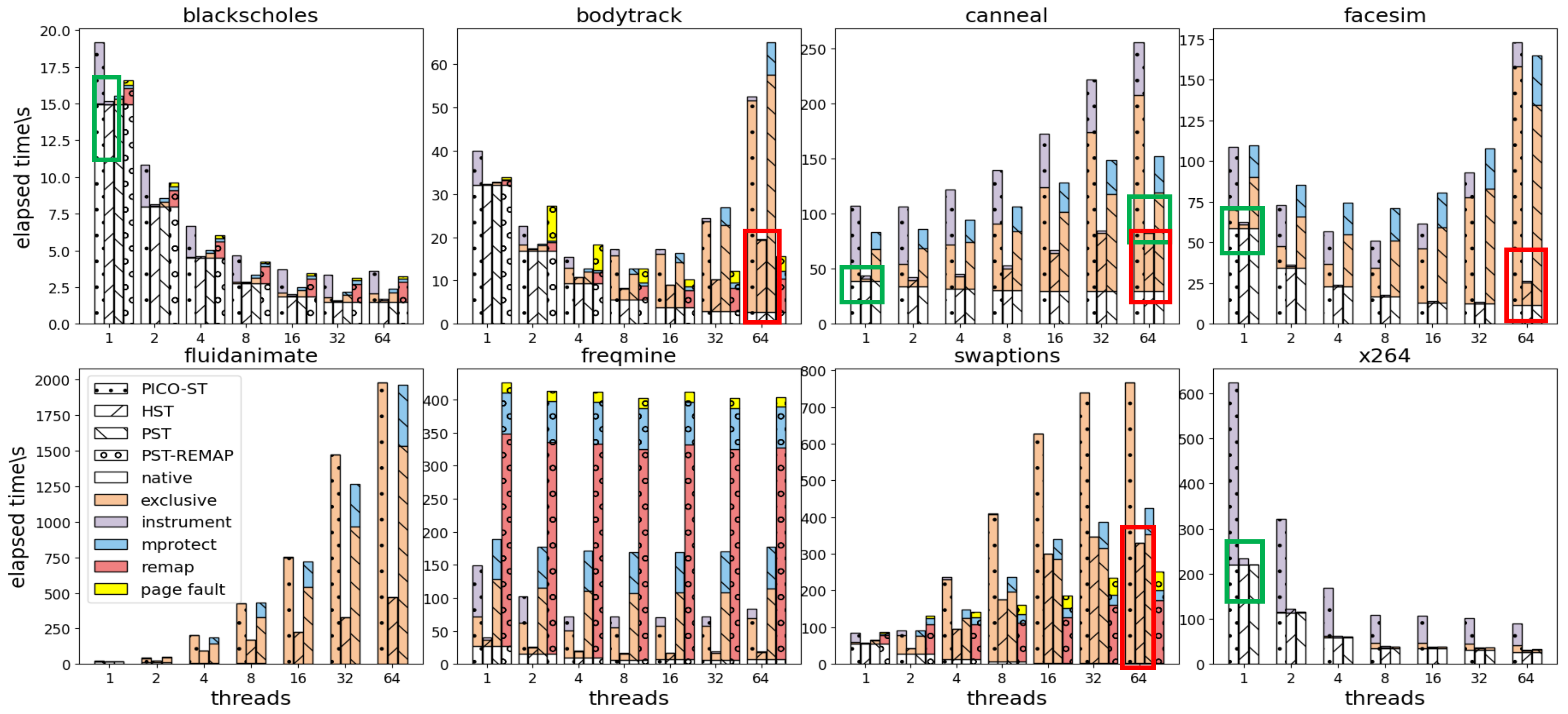


Overhead Analysis



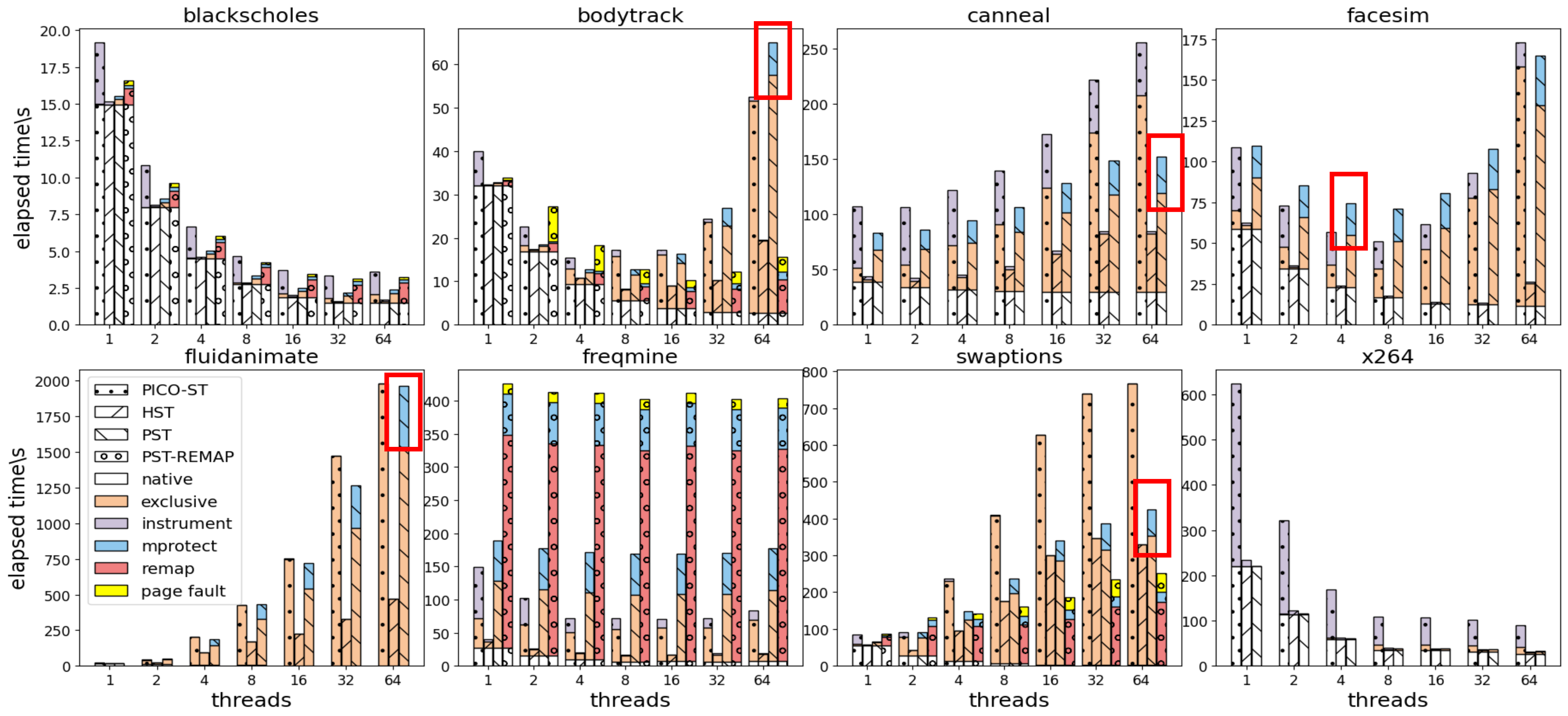
Pico-ST bottleneck: store instrumentation 20%~45%

Overhead Analysis



HST: store instrumentation 5%; major overhead: SC synchronization

Overhead Analysis



PST: mprotect system call becomes the new overhead

Summary

- Trade-off between atomicity, speed, and portability

Approaches	Speed	Atomicity	Portability
HST	fast	strong	portable
HST-weak	fast	weak	portable
HST-HTM	fast	strong	HTM
PST	slow	strong	portable
PST-remap	varies	strong	portable
Pico-ST	Baseline(slow)	strong	portable
QEMU	fast	incorrect	portable
HTM	fast	incorrect	HTM

Discussion

- Avoiding synchronization in Store-Conditional
  - Double Compare Single Swap [Timothy, 2002]
  - Intel Memory Protection Key [Park, 2019]
- Changing the translation scheme
  - Rule-based Code Translation [Jiang, 2020]
  - Adding new Intermediate Representation semantics
- Our code is available at: https://github.com/NKU-EmbeddedSystem/ABA-LLSC

Thank you!

Backup Slides

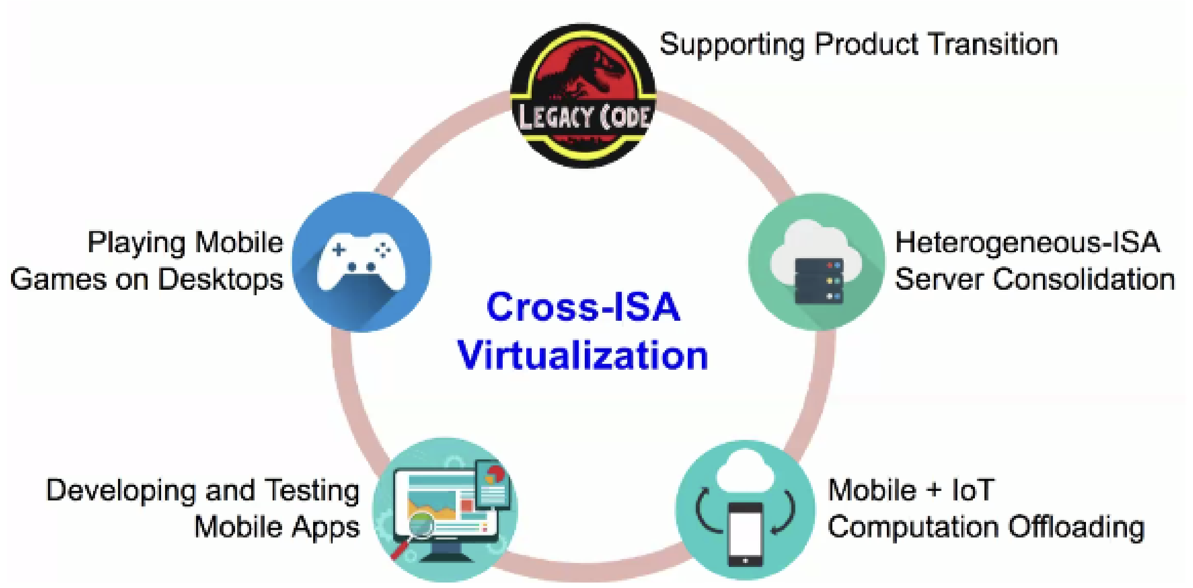
Outline

- Motivation
  - Atomic instruction in cross-ISA emulation is important
    - Cross-ISA emulation is important: key enabling tech
    - Atomic instruction translation is important: wrong translate lead to wrong results
      - RISC-->CISC r-->w -->ABA
- Design
  - Goal: keep atomicity of LL/SC
  - Prev work:
    - a. workload on threads
      - store overhead 20%~40% (really?) -- could be fast! -- with lock-free hash table
    - b. workload on MMU
      - Seems to be promising (really?) -- could be slow!! -- large syscall overhead
    - c. workload on HTM
      - Fast & correct (really?) -- could be incorrect!! -- JIT & HTM be careful!!
  - HST, HST-weak
  - PST, PST-remap, PST-MPK
  - HST-HTM
- Evaluation
- Discussion
  - PST-MPK

Cross-ISA Dynamic Binary Translation

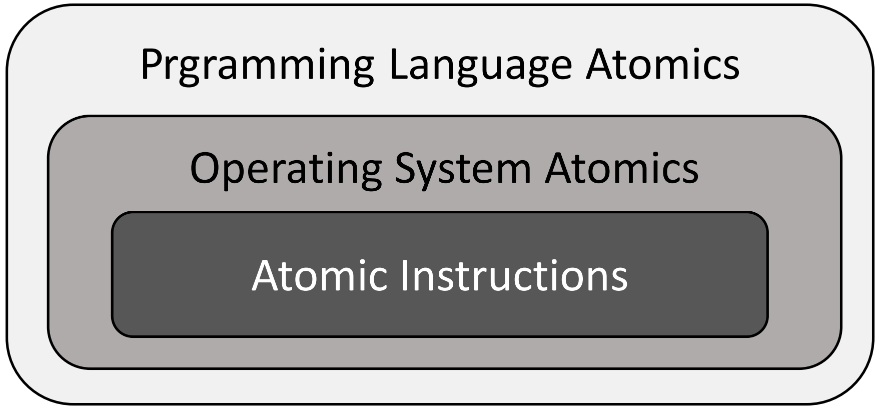
Cross-ISA Emulation

*"A Key **Enabling** Technology"*


Supporting Product Transition
LEGACY CODE
Playing Mobile Games on Desktops
Heterogeneous-ISA Server Consolidation
Cross-ISA Virtualization
Developing and Testing Mobile Apps
Mobile + IoT Computation Offloading

Atomic Instruction

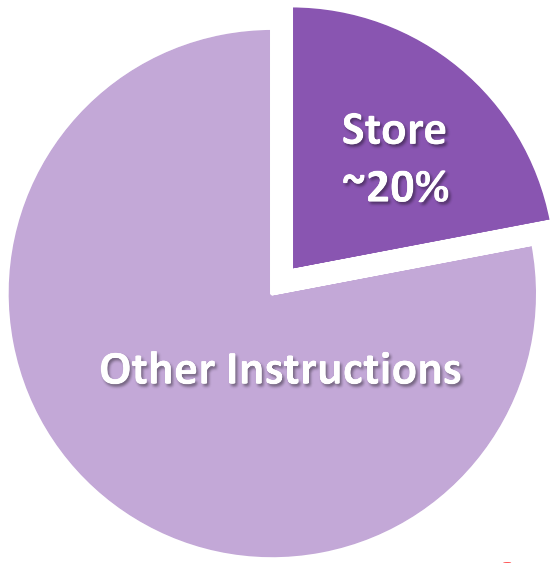
*Fundamental to **Correctness***



Atomics are not correctly translated?

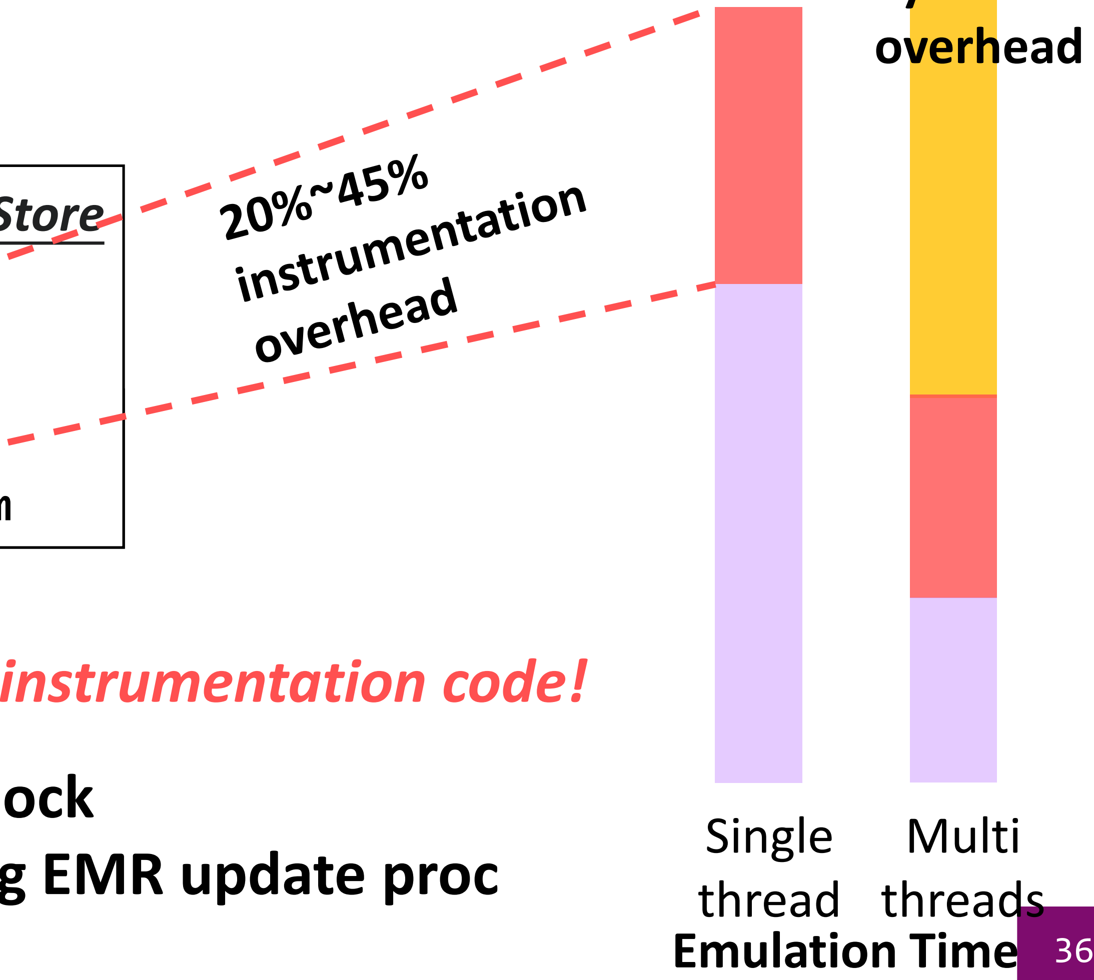
A. HST: Hash-table based STore

- Challenge: store instrumentation overhead



Translated Store

```
find_EMR(mem)
lock
update_EMR
unlock
store   reg, mem
```

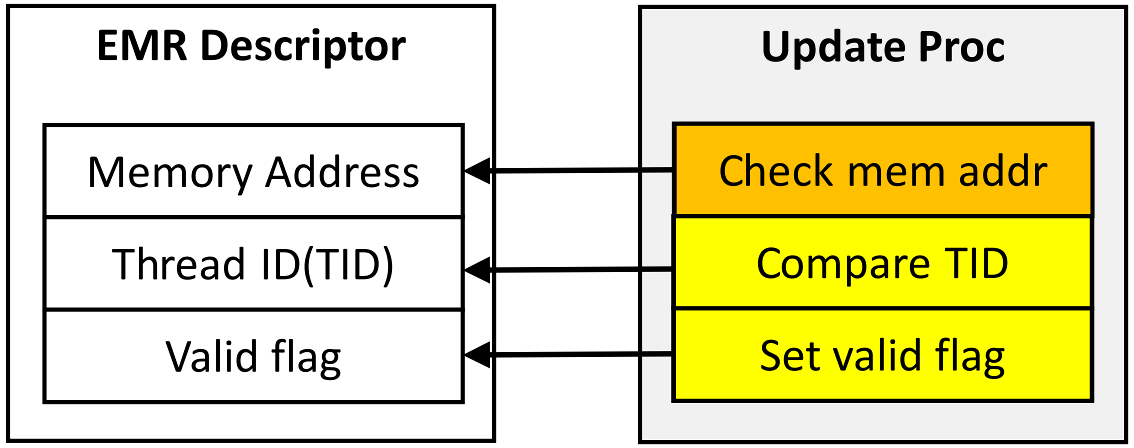


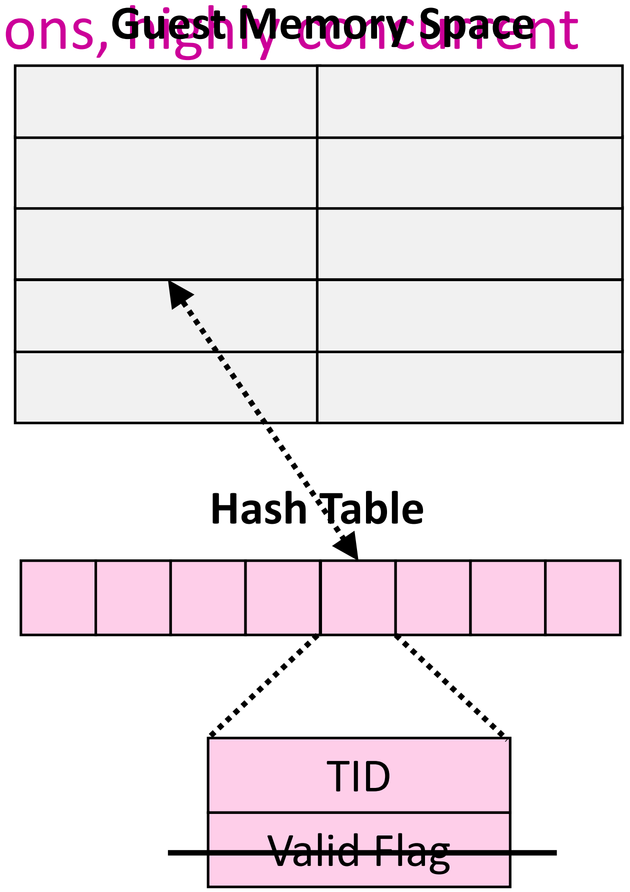
Lesson learned: Have to reduce instrumentation code!

1. **Scalability: Avoiding lock**
2. **Overhead: Simplifying EMR update proc**

A. HST: Hash-table based STore

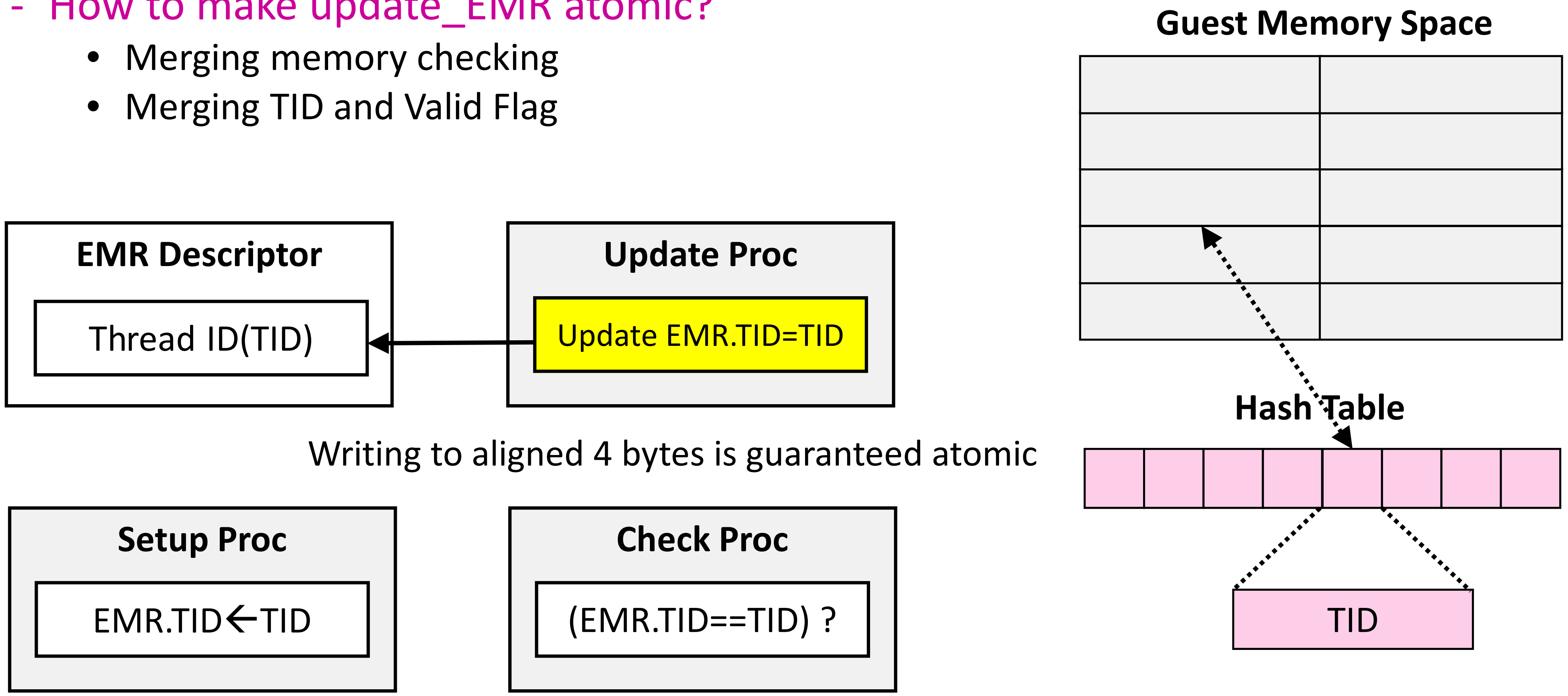
- Challenge: store instrumentation overhead
  - store instructions accounts for up to 20% instructions, highly concurrent
  - **Be Fast**: lightweight checking procedure
  - **Be Scalable**: no locking





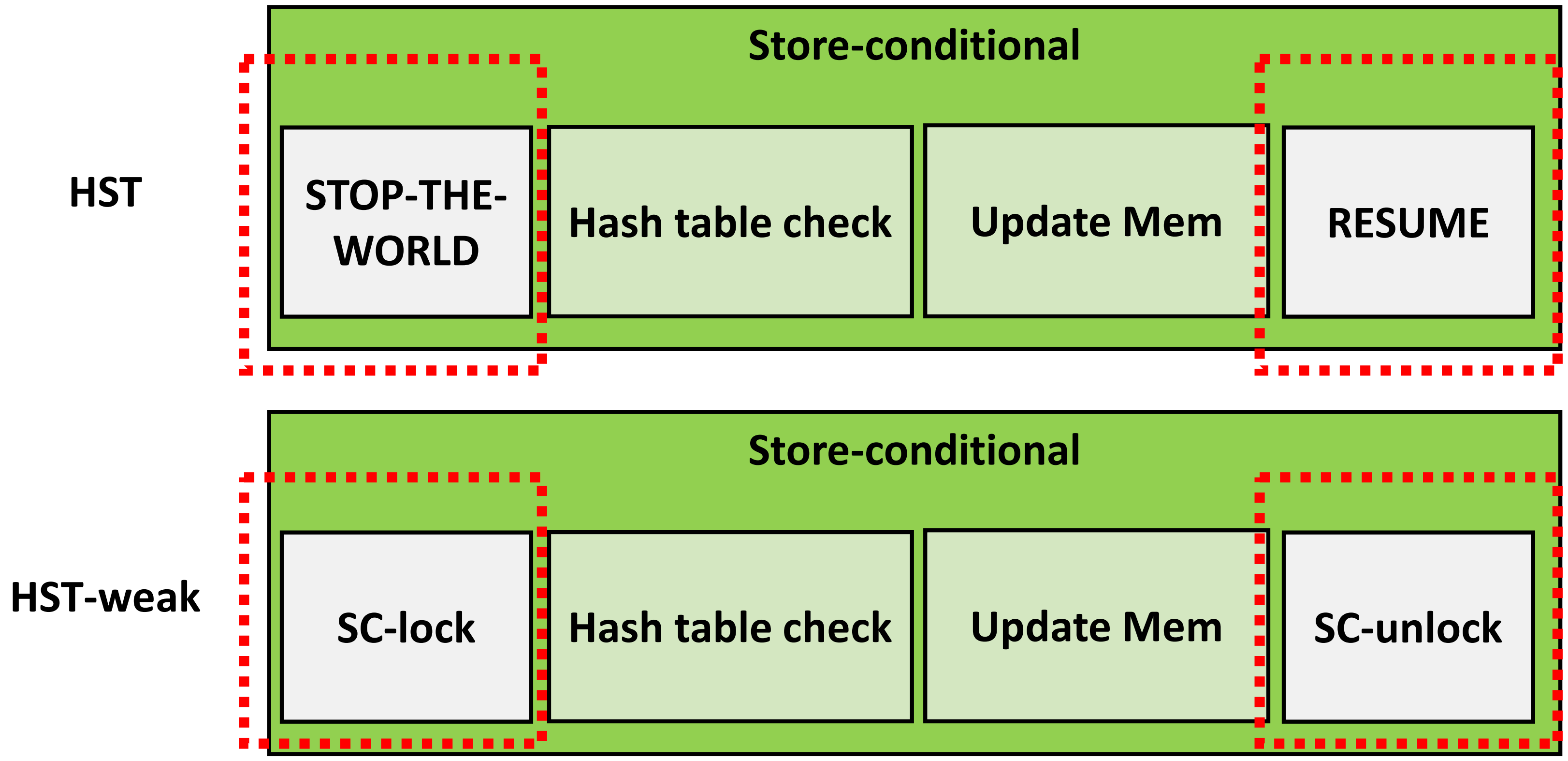
A. HST: Hash-table based STore

- Challenge: store instrumentation overhead
  - How to make update_EMR atomic?
    - Merging memory checking
    - Merging TID and Valid Flag



A. HST: Hash-table based STore

- Does it provide atomicity
- Hash confliction


HST
Store-conditional
STOP-THE-WORLD
Hash table check
Update Mem
RESUME
HST-weak
Store-conditional
SC-lock
Hash table check
Update Mem
SC-unlock