

# InvisiFence: Performance-Transparent Memory Ordering in Conventional Multiprocessors

Colin Blundell (University of Pennsylvania)

Milo M. K. Martin (University of Pennsylvania)

Thomas F. Wenisch (University of Michigan)



# This work licensed under the Creative Commons **Attribution-Share Alike 3.0 United States** License



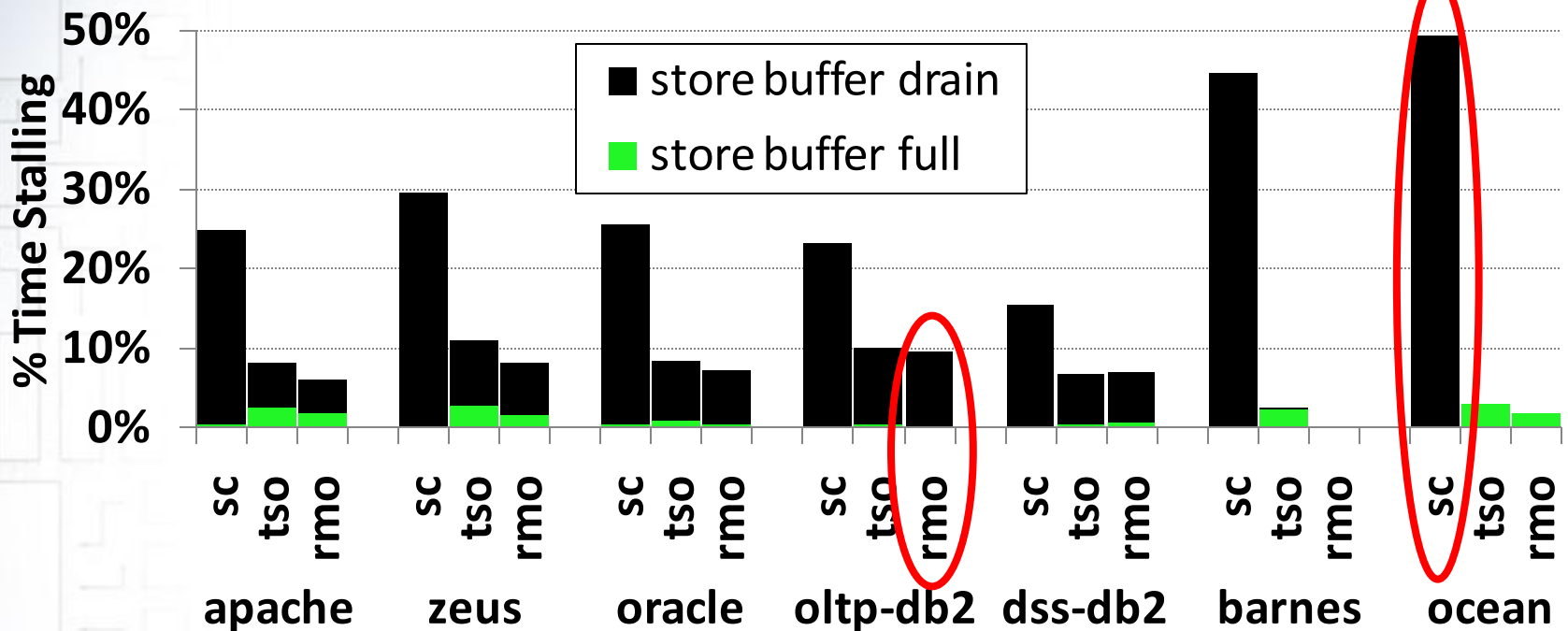
- **You are free:**
  - to **Share** — to copy, distribute, display, and perform the work
  - to **Remix** — to make derivative works
- **Under the following conditions:**
  - **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
  - **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.
- For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to:  
<http://creativecommons.org/licenses/by-sa/3.0/us/>
- Any of the above conditions can be waived if you get permission from the copyright holder.
- Apart from the remix rights granted under this license, nothing in this license impairs or restricts the author's moral rights.



# Performance Impact of Memory Ordering

- Memory ordering (still) causes stalls

# Performance Impact of Memory Ordering



- ... Even with relaxed memory models
  - Frequent memory fences & atomic RMW's (synchronization)
- ... Even with aggressive in-window speculation
  - Can't tolerate long miss latencies

# Speculative Memory Ordering

- Hide stalls with beyond-the-window speculation
    - Races rare → ordering stalls usually unnecessary [Gniady'99]
  - Past work focused on **programmability**
    - Thus, focused on strong ordering (SC/PC)
- + Effectively eliminate stalling
- Complex mechanisms:
- Fine-grained (per-store) tracking [Wenisch'07]
  - Expensive commit [Ceze'07, Wenisch'07]
  - Unconventional memory system [Ceze'07]

# Our Approach: InvisiFence

- Key departure: apply to weakly-ordered system
  - Straightforward hardware; fewest stalls to address
- Augment with familiar deep speculation mechanisms
  - Violation detection: read/write bits in cache
  - Version management: clean to L2 before 1<sup>st</sup> write
- Result: eliminate fence stalls (up to 13% speedup)
  - No fine-grained (per-store) tracking
  - Fast & simple commit and rollback
  - Conventional memory system
- For strong ordering: speculate more (“implicit fences”)
  - Bonus: can even eliminate LSQ snooping! (a la [Ceze’07])

# Roadmap

- InvisiFence for weak ordering
- Generalizing InvisiFence to stronger models
- Subsuming in-window speculation
- Conclusions



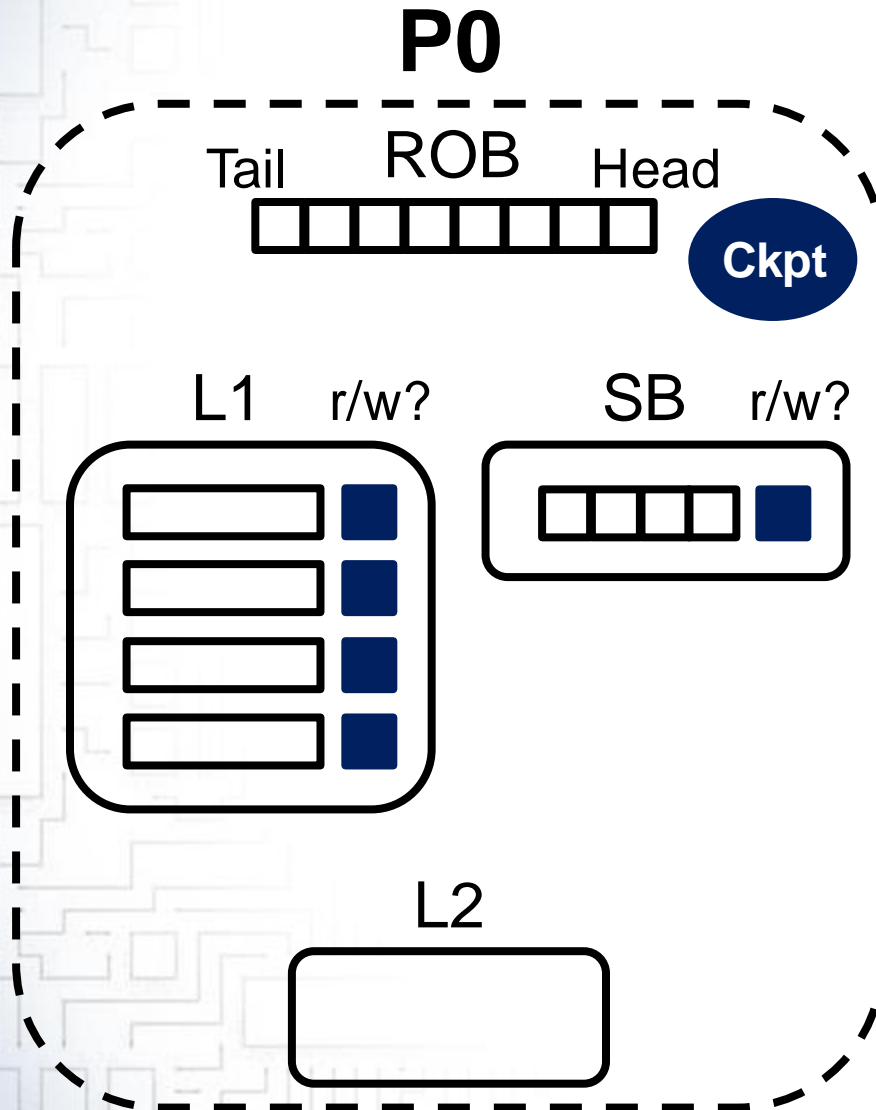
# Background: Weak Ordering

- Relaxes ordering except at programmer-inserted fences
  - Allows unordered store buffer to hide store misses
- Unordered, coalescing store buffers → simple, scalable
  - Cache-like organization
  - Store hits skip store buffer; only one entry per miss
  - Result: largely eliminate capacity stalls of FIFO store buffers
- However, still incur consistency-induced stalls
  - ...even with in-window speculation (LSQ snooping)
    - Fences: drain store buffer (stall until empty)
    - Atomic ops: stall until has write permission

# InvisiFence For Weak Ordering

- Add deep speculation to eliminate stalling on fences
- Mechanism: register ckpt + 2 bits per L1 cache line
  - Similar HW to other deep speculation (TLS, TM, Cherry...)
- Initiate speculation at fence instructions
  - Detect violations via cache coherence protocol
  - Preserve non-speculative data in L2 (facilitates rollback)
- Speculation ends when store buffer becomes empty
  - Commit by flash-clearing read/write bits

# InvisiFence Hardware



P1

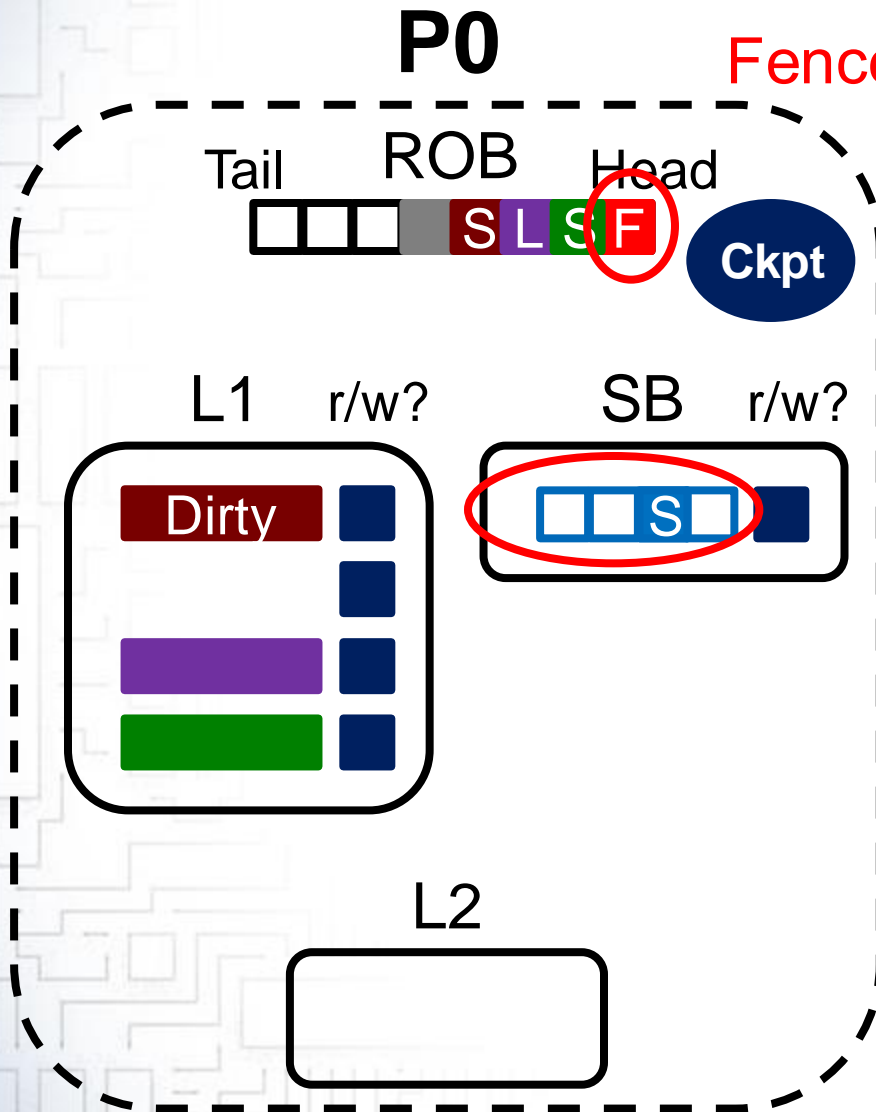
## Baseline:

- OoO pipeline
- LSQ snooping
- Writeback L1 & L2
- Invalidation-based CC
- Coalescing store buffer

## InvisiFence extensions:

- Register checkpoint
- 2 bits per L1 cache line
- 2 bits per SB entry

# InvisiFence: Example



Fence wants to retire...

P1

Initiate speculation  
Speculatively retire fence

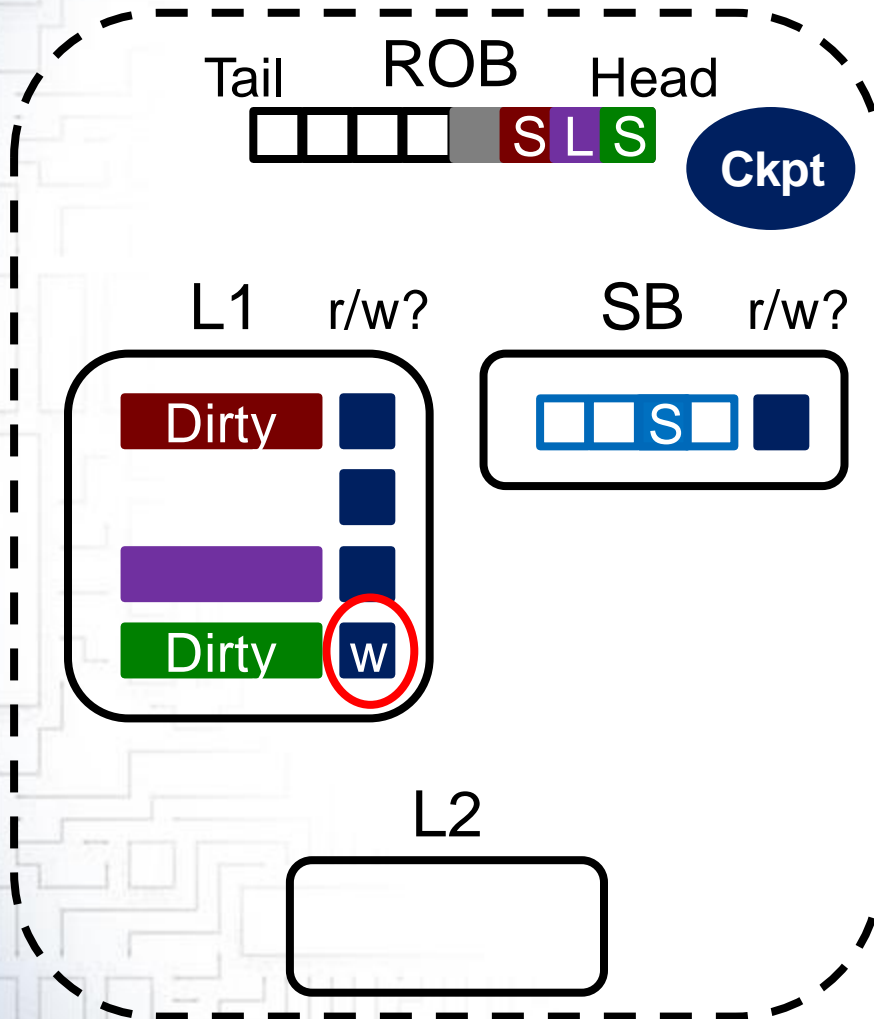
...but store miss  
outstanding

Key

<b>S</b> Store	<b>L</b> Load
<b>F</b> Fence	<b>Other Insns</b>

# InvisiFence: Violation Detection

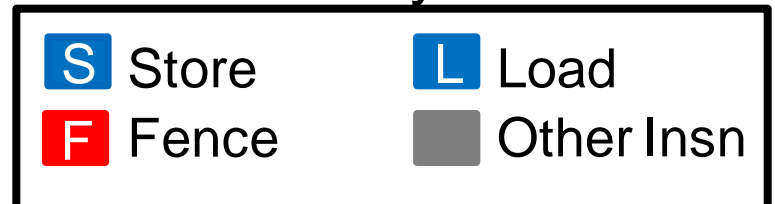
**P0**



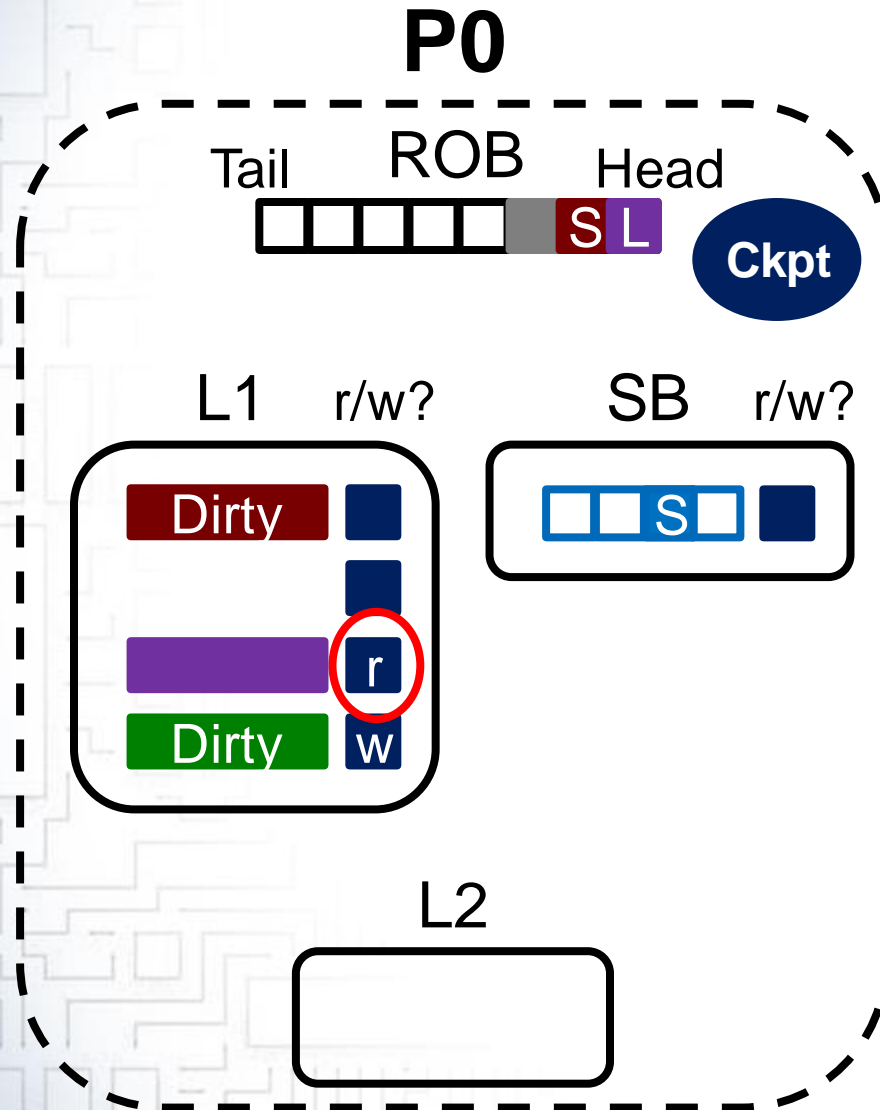
**P1**

At store retirement:  
Set write bit

Key

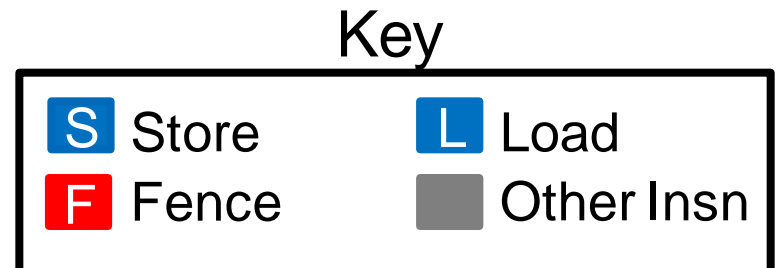


# InvisiFence: Violation Detection



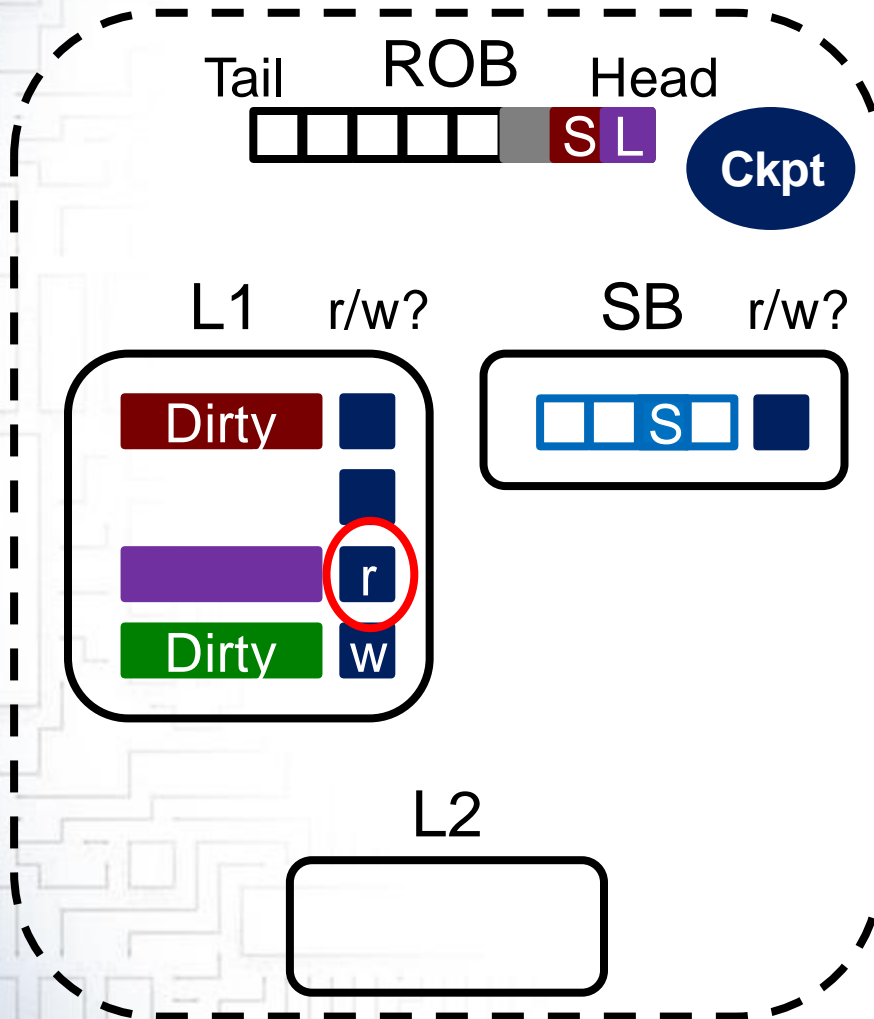
**P1**

At load retirement:  
Set read bit



# InvisiFence: Violation Detection

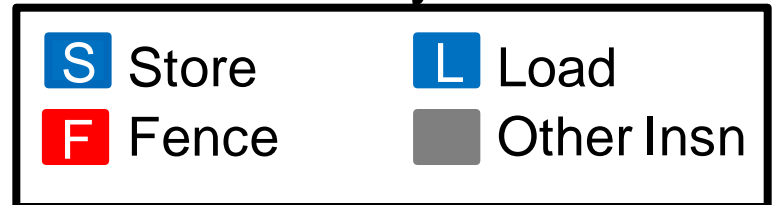
**P0**



**P1**

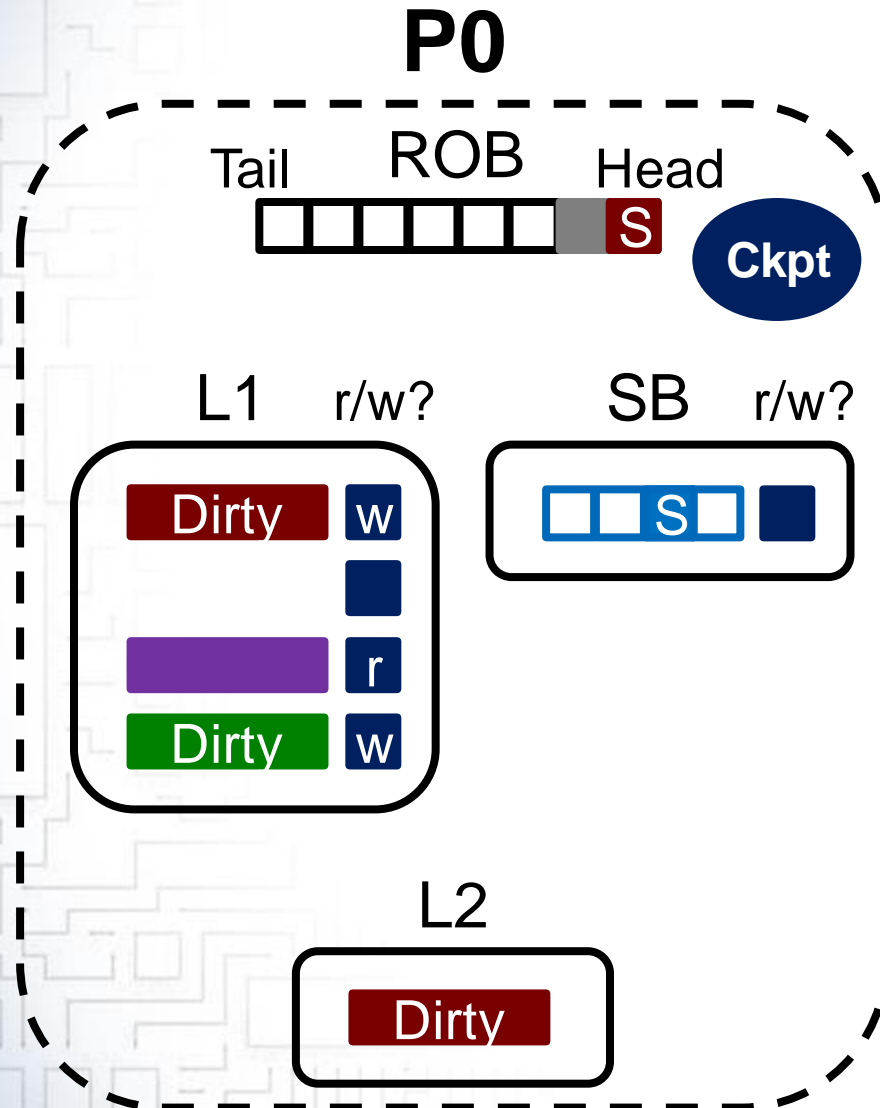
To detect violations:  
snoop bits

Key

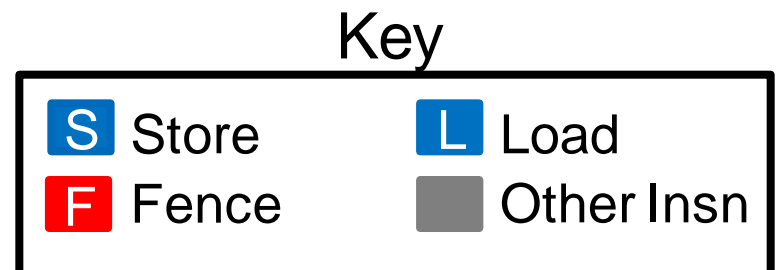




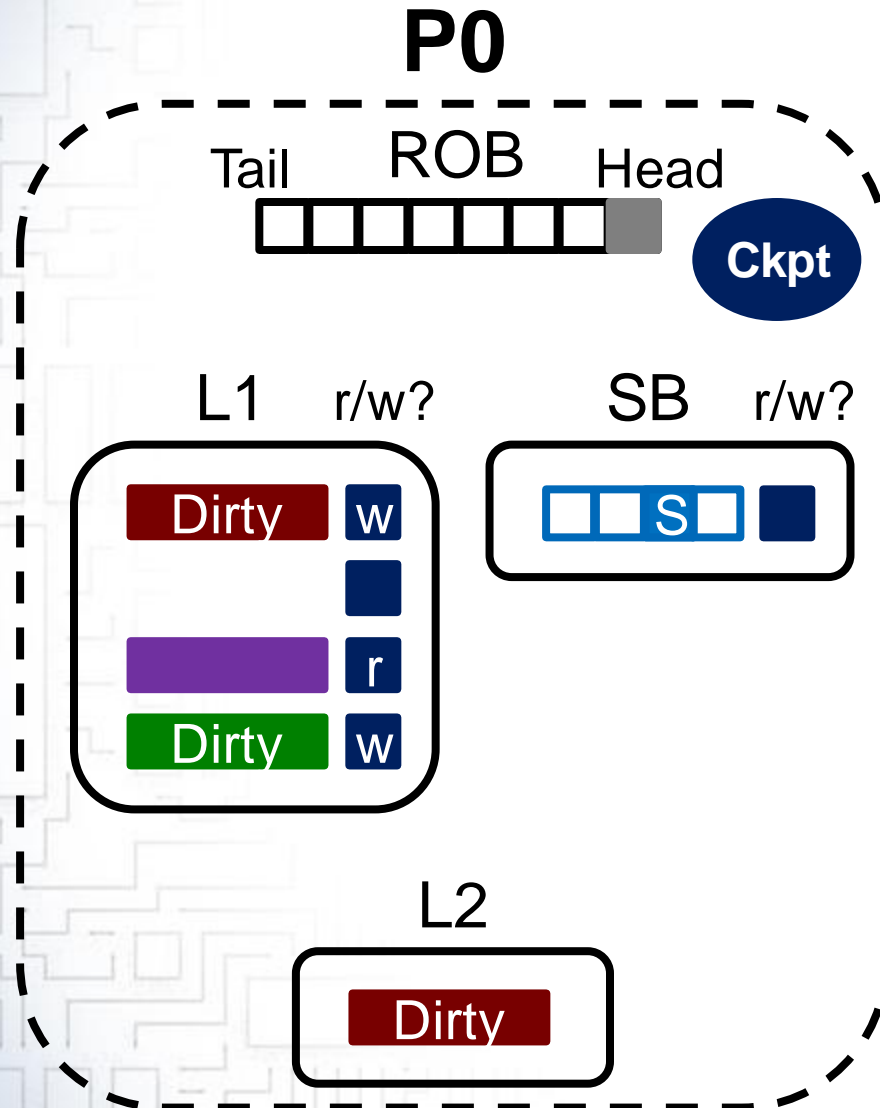
# InvisiFence: Version Management



**P1**



# InvisiFence: Version Management



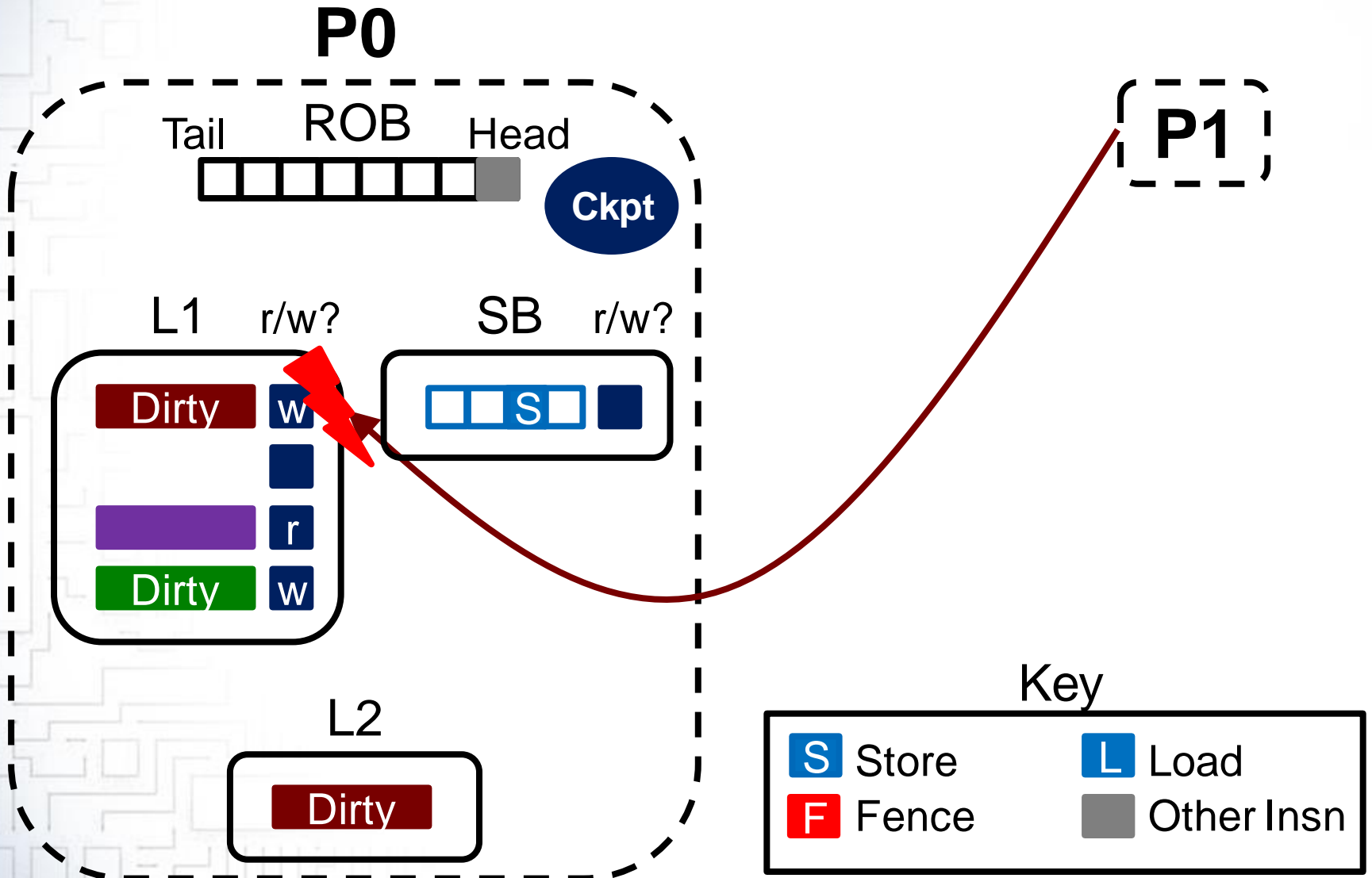
**P1**

Can always recover non-spec version from L2 (no custom storage)

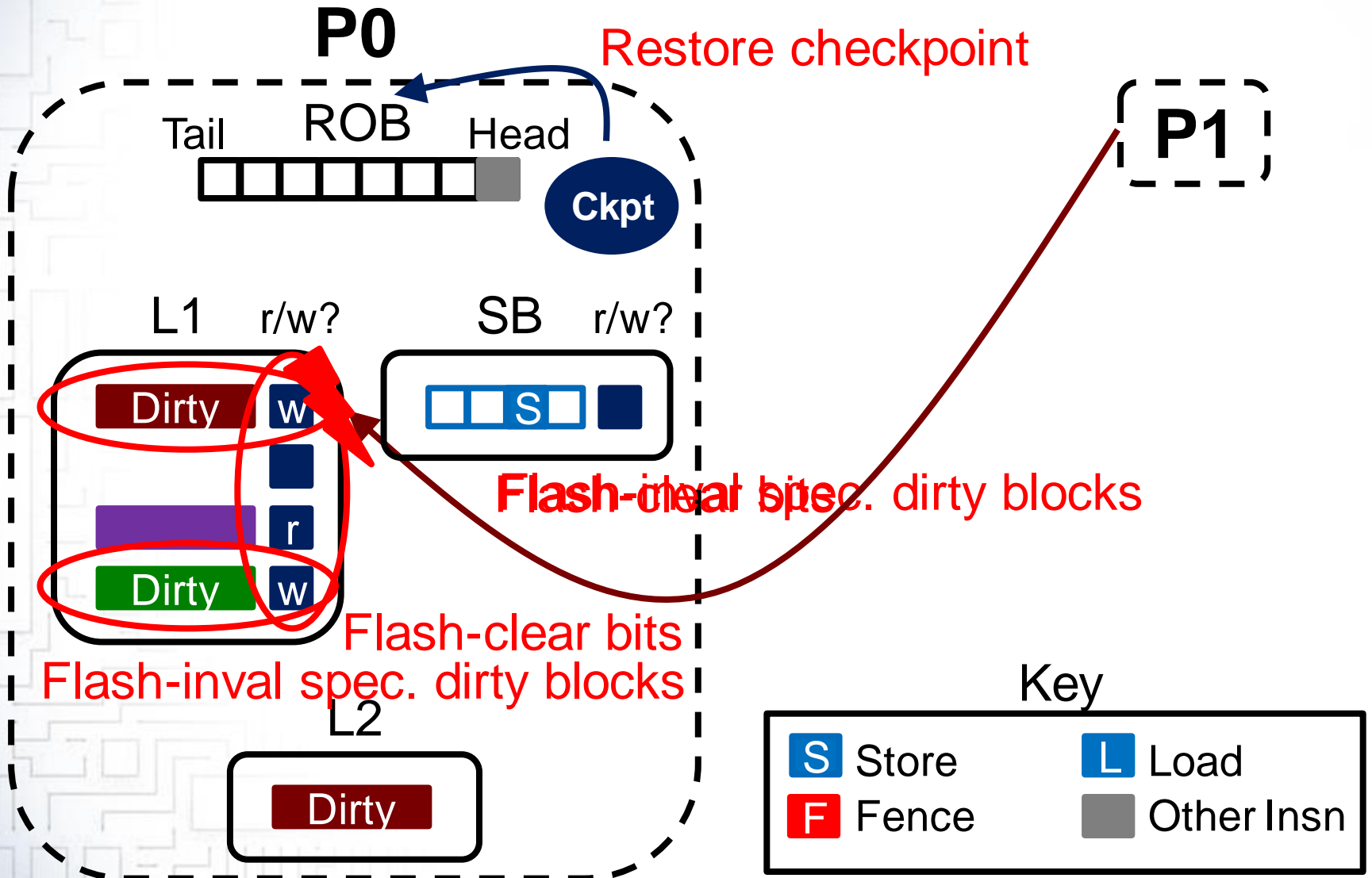
Key

<b>S</b> Store	<b>L</b> Load
<b>F</b> Fence	<b>Other Insn</b>

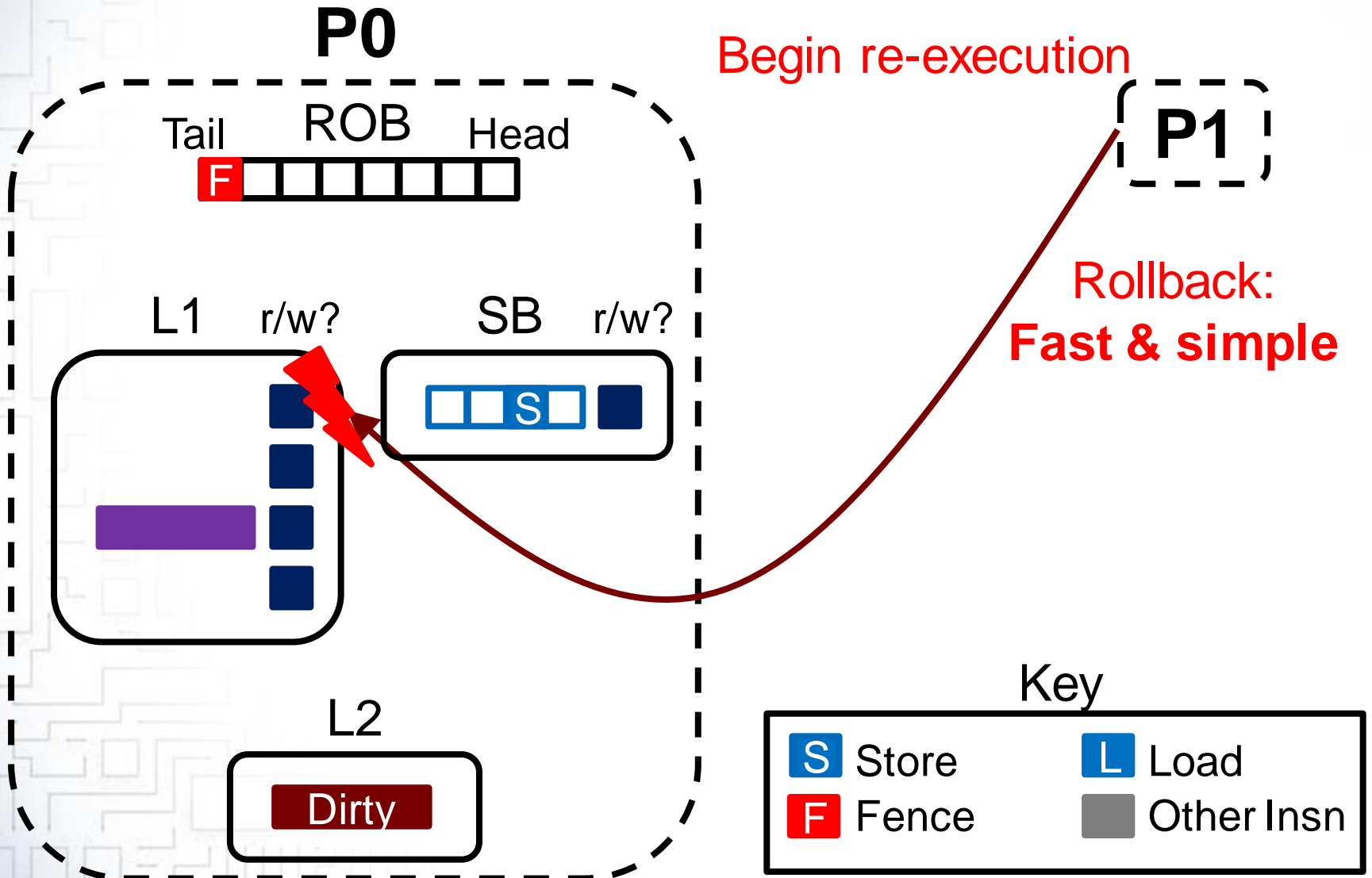
# InvisiFence: Rollback



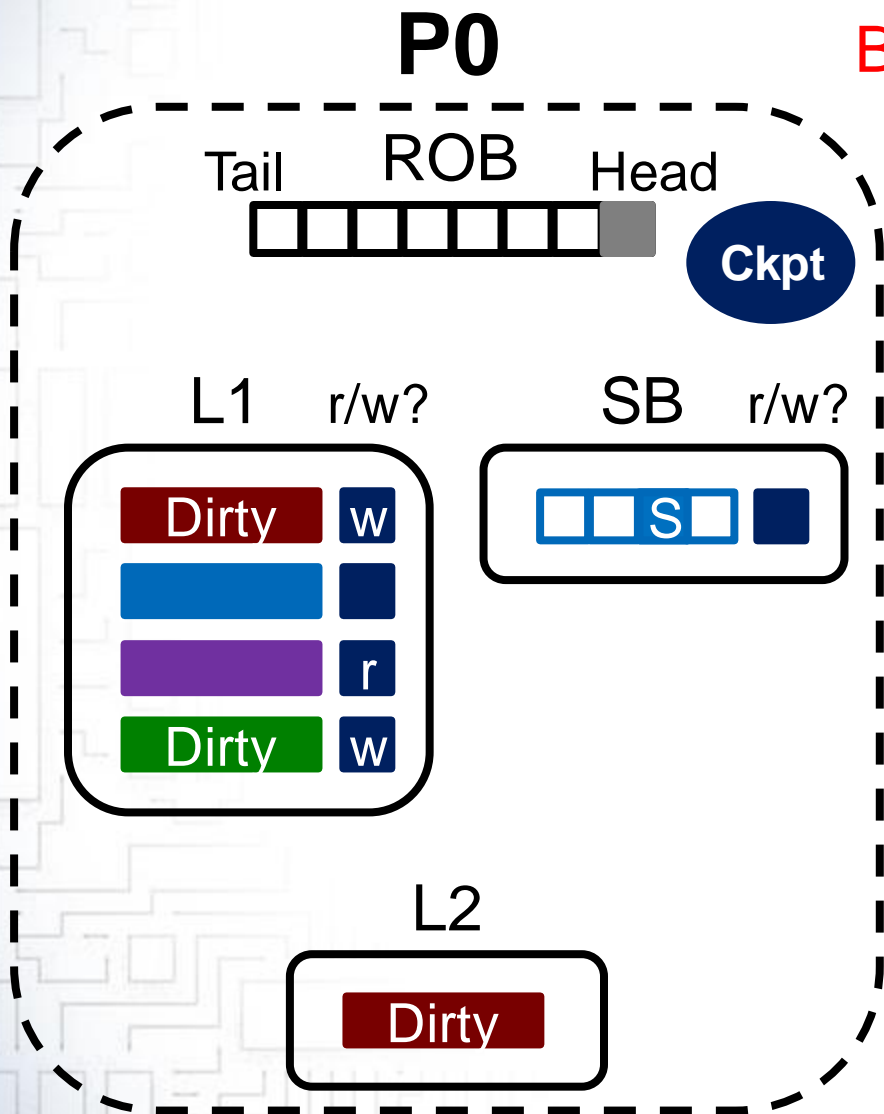
# InvisiFence: Rollback



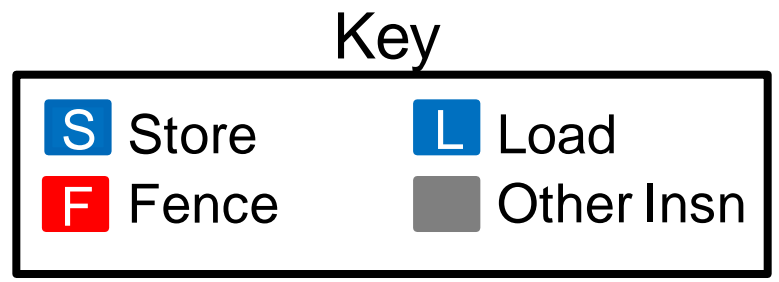
# InvisiFence: Rollback



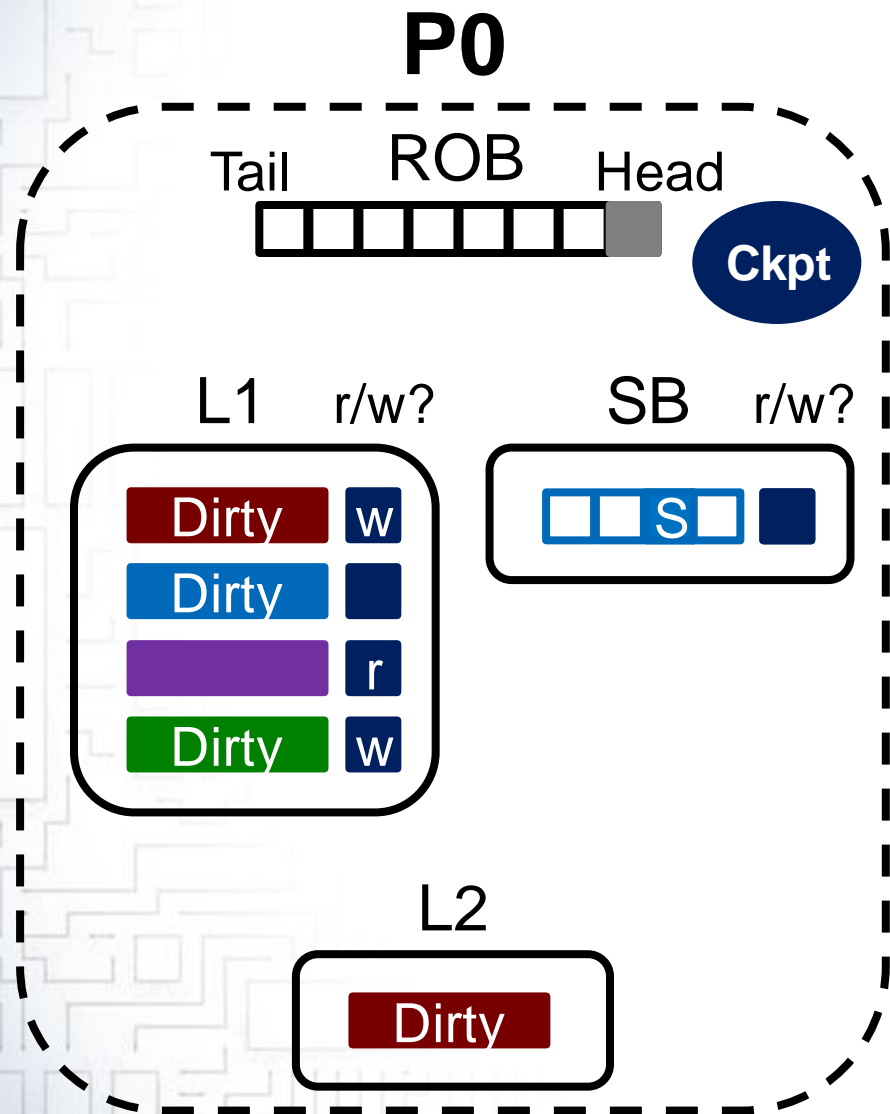
# InvisiFence: When to Commit?



Back to speculation:  
Store returns

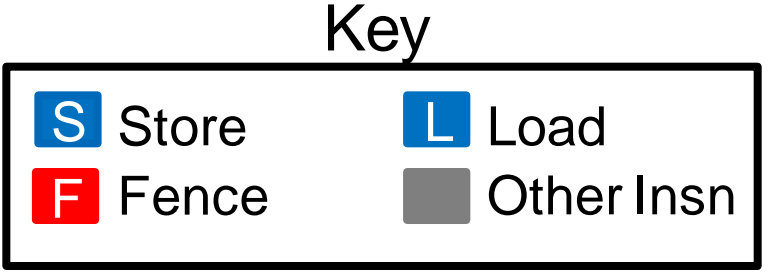


# InvisiFence: When to Commit?

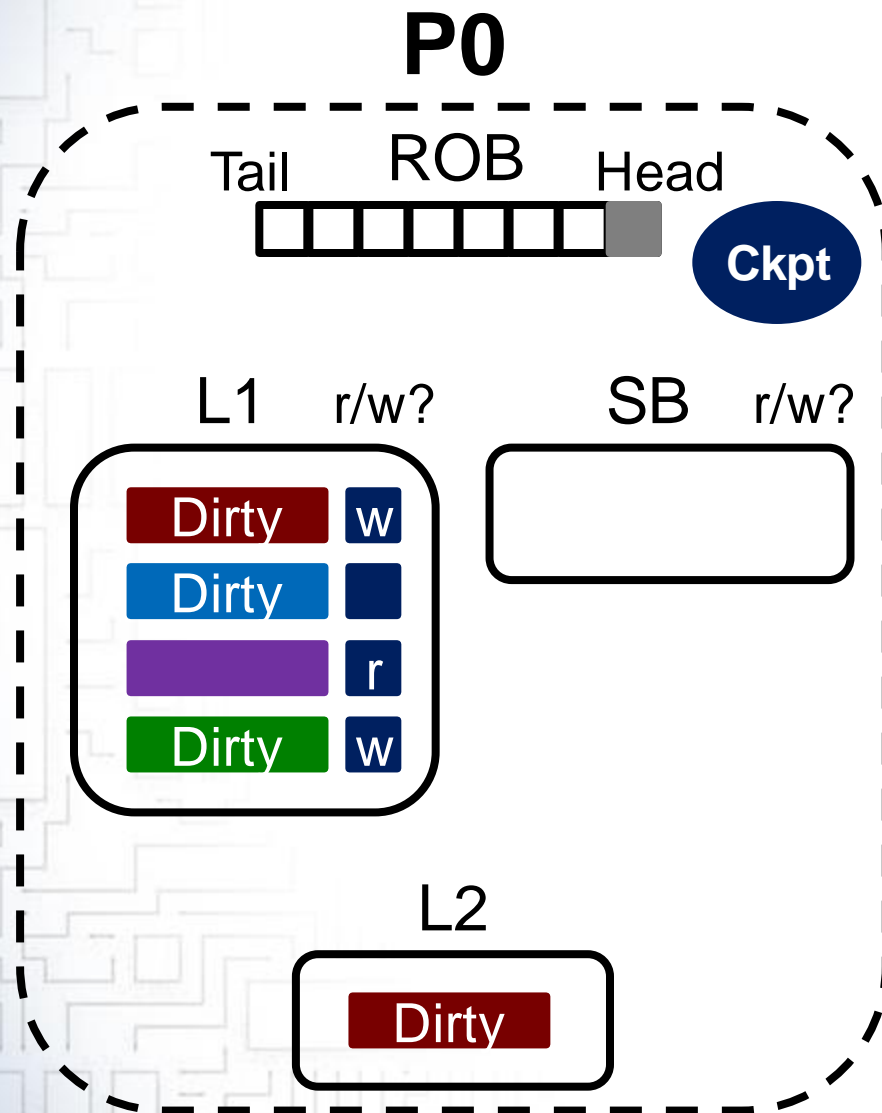


**P1**

Move store & r/w bit from SB to L1

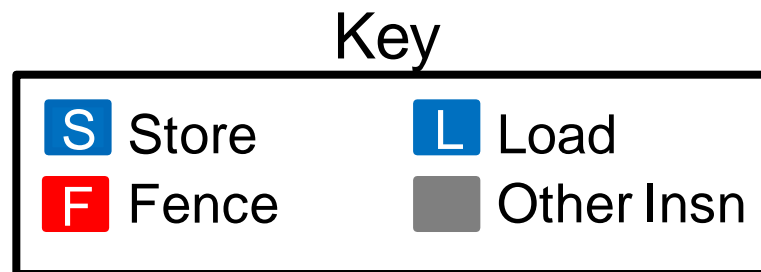


# InvisiFence: When to Commit?

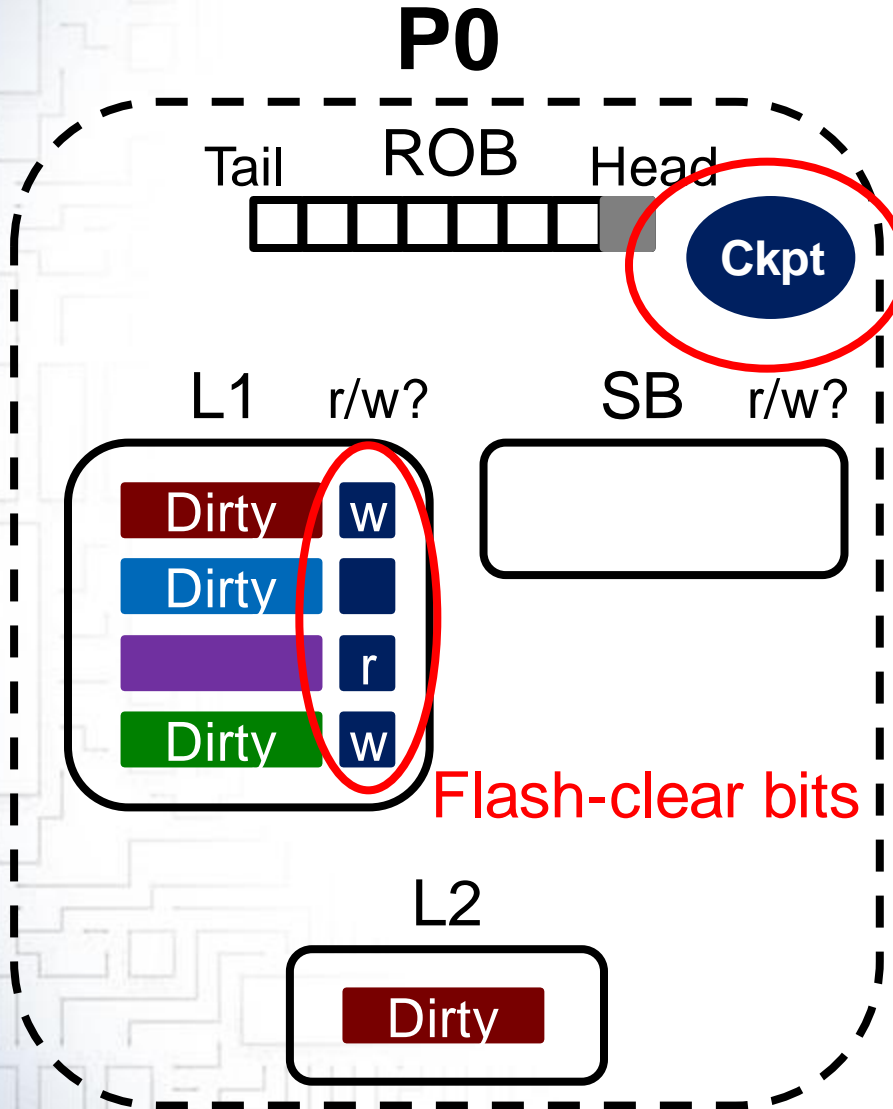


**P1**

No outstanding stores:  
Legal to commit



# InvisiFence: Commit



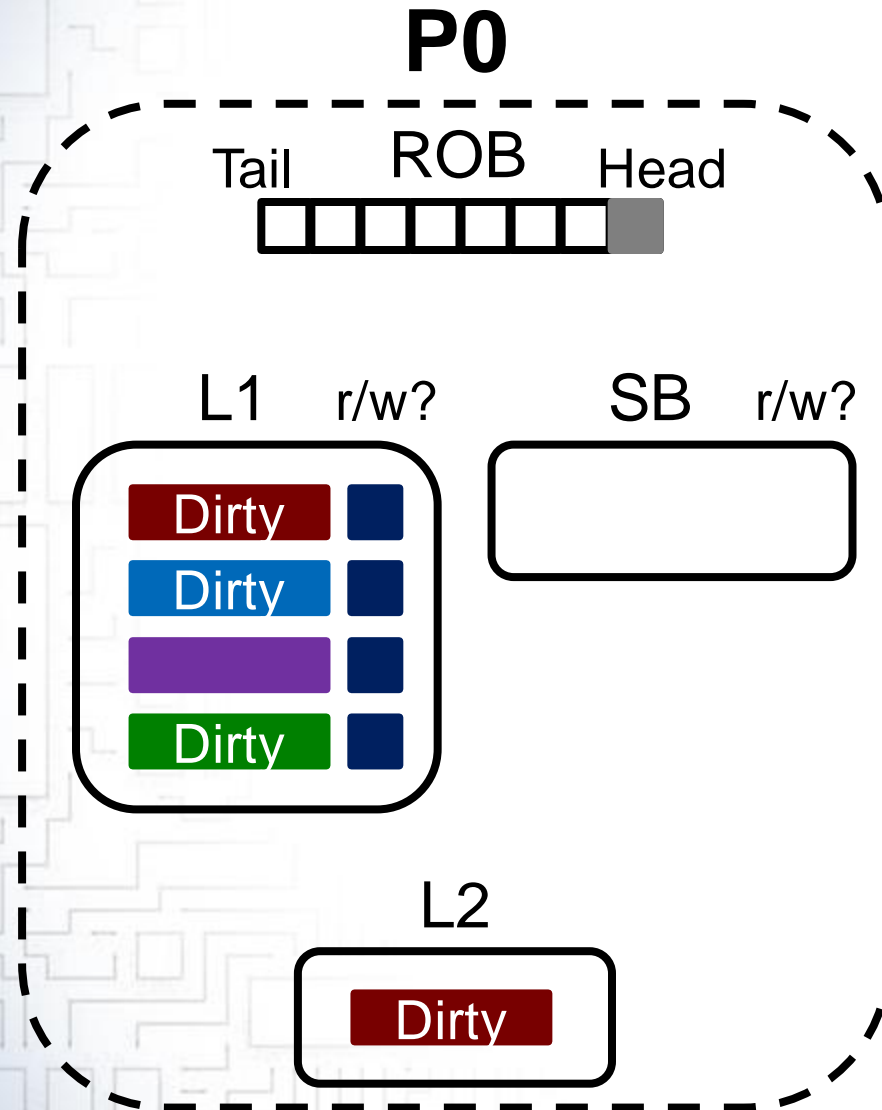
**P1**

Discard checkpoint

Key

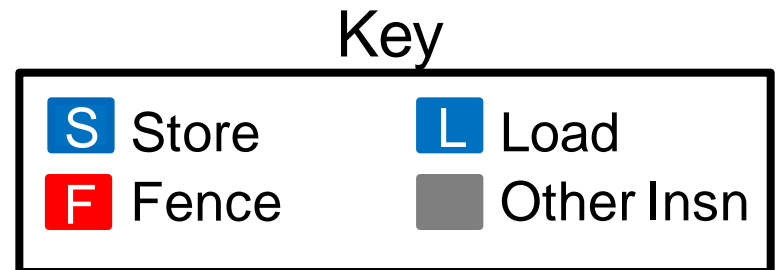
<b>S</b> Store	<b>L</b> Load
<b>F</b> Fence	<b>Other Insns</b>

# InvisiFence: Commit

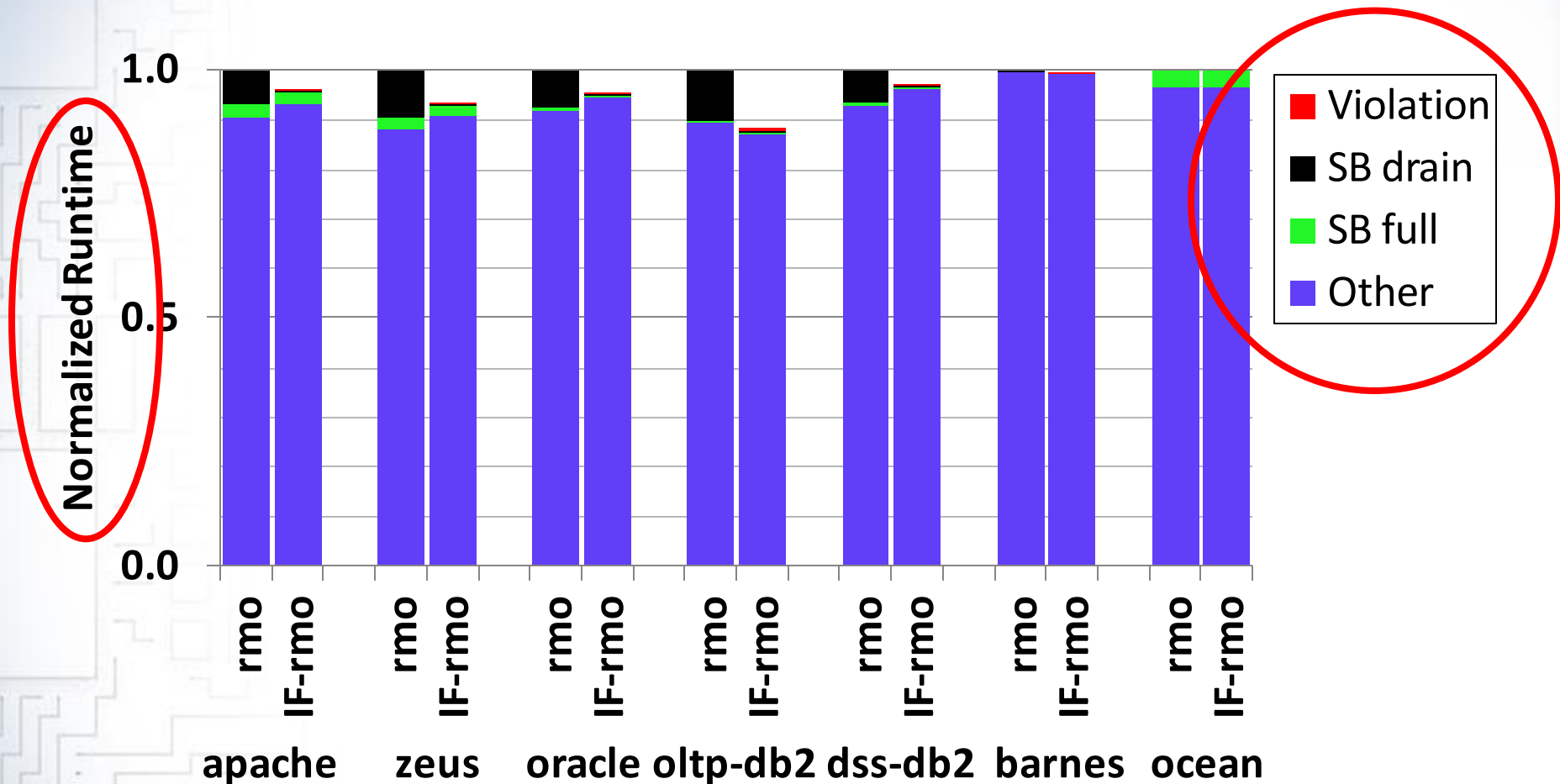


**P1**

Commit:  
**Fast & simple**



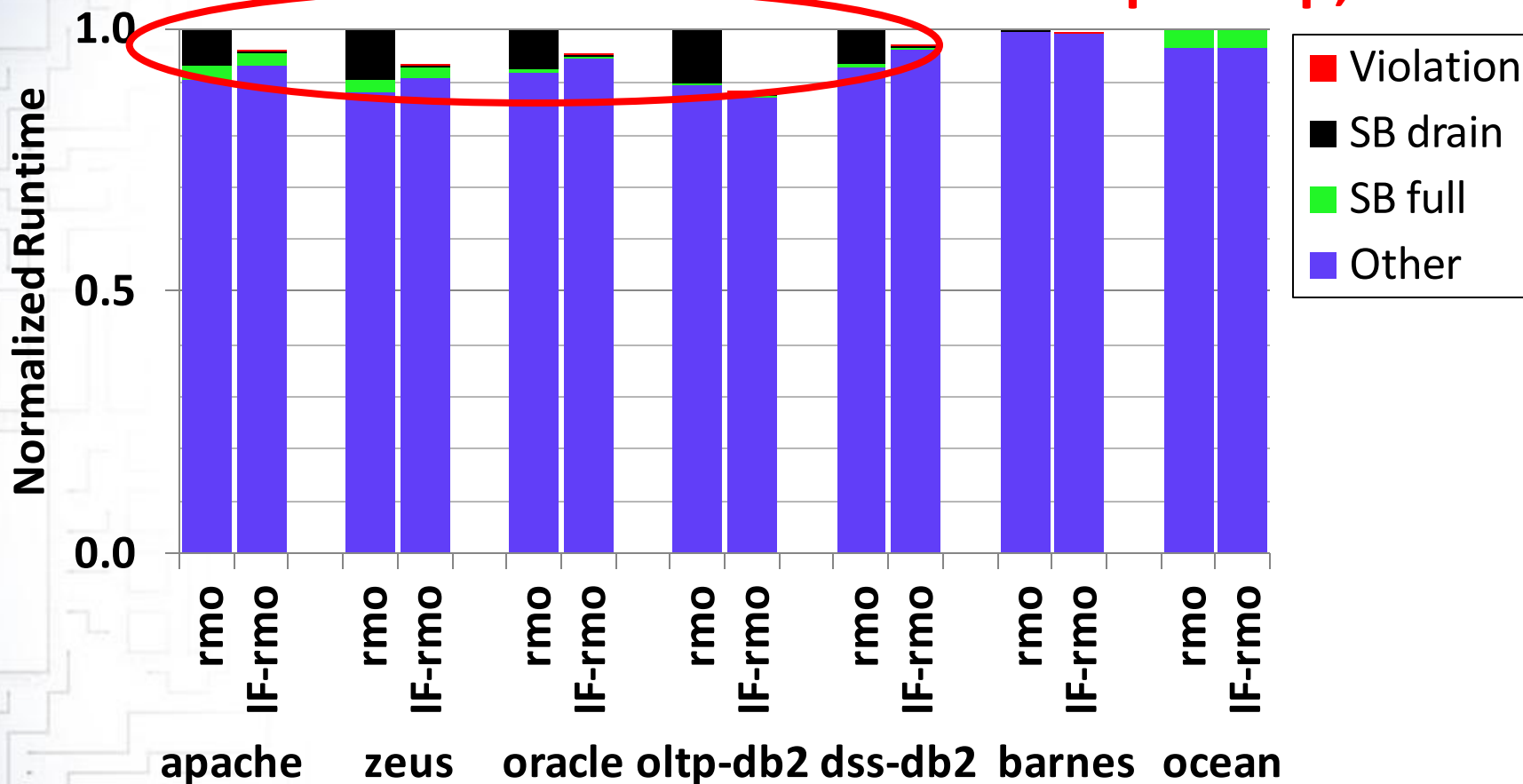
# InvisiFence Performance



**SimFlex simulation of 16-node directory-based SPARC MP  
SPARC's RMO (similar to Alpha, ARM, PowerPC)**

# InvisiFence Performance

13% max speedup; 6% avg



**InvisiFence eliminates fence stalls without violations**

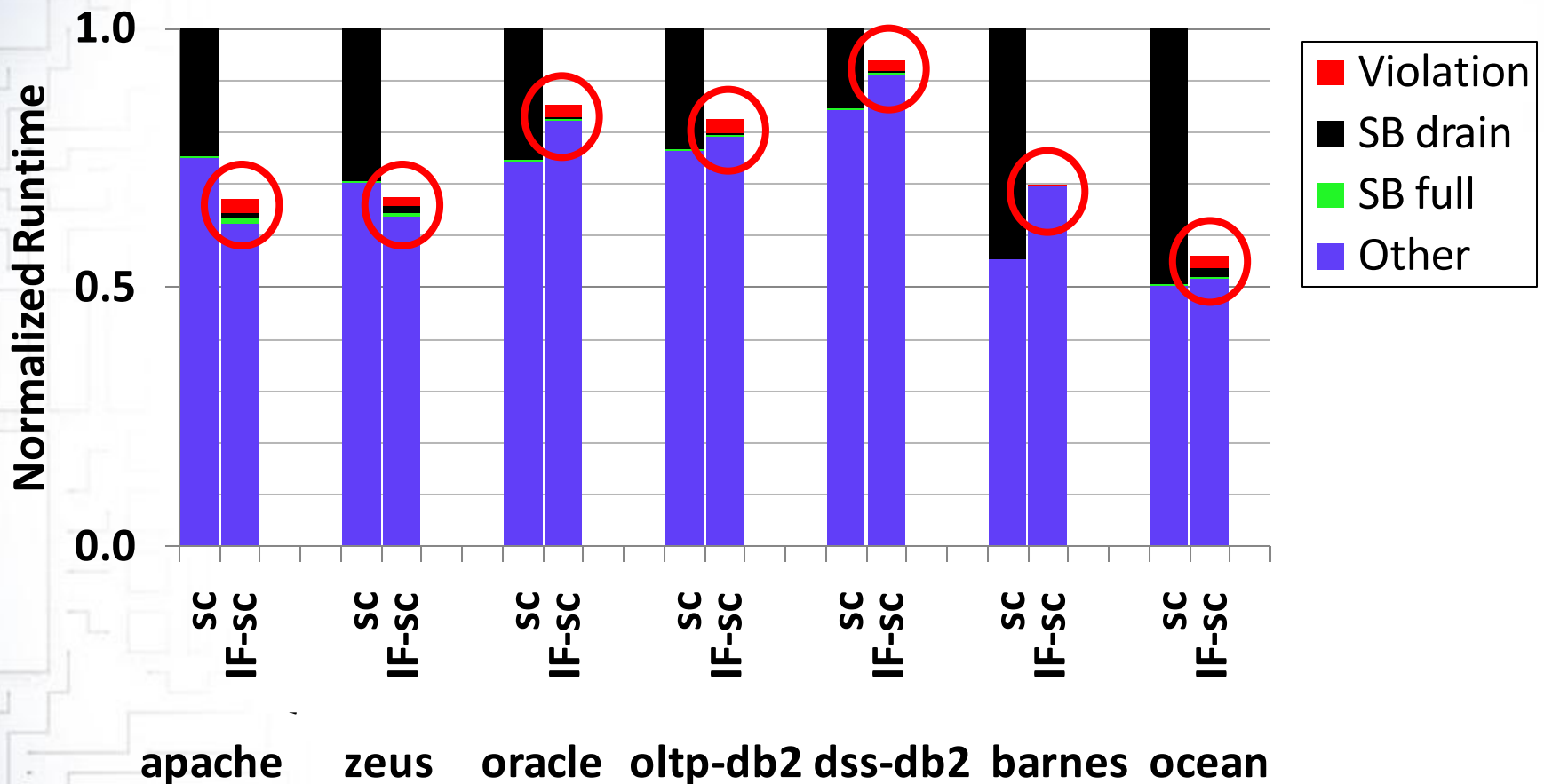
**But what about models requiring stronger ordering?**

# Generalizing InvisiFence for Strong Ordering

- Strong models impose additional ordering constraints
  - Processor Consistency (x86, TSO): ordering between stores
  - Sequential Consistency: ordering between all operations
- These constraints are conceptually “implicit fences”
  - *e.g.*, for SC: every operation is “implicit fence”
- InvisiFence can handle these just like explicit fences!
  - Increases speculation frequency...

**No other hardware changes**

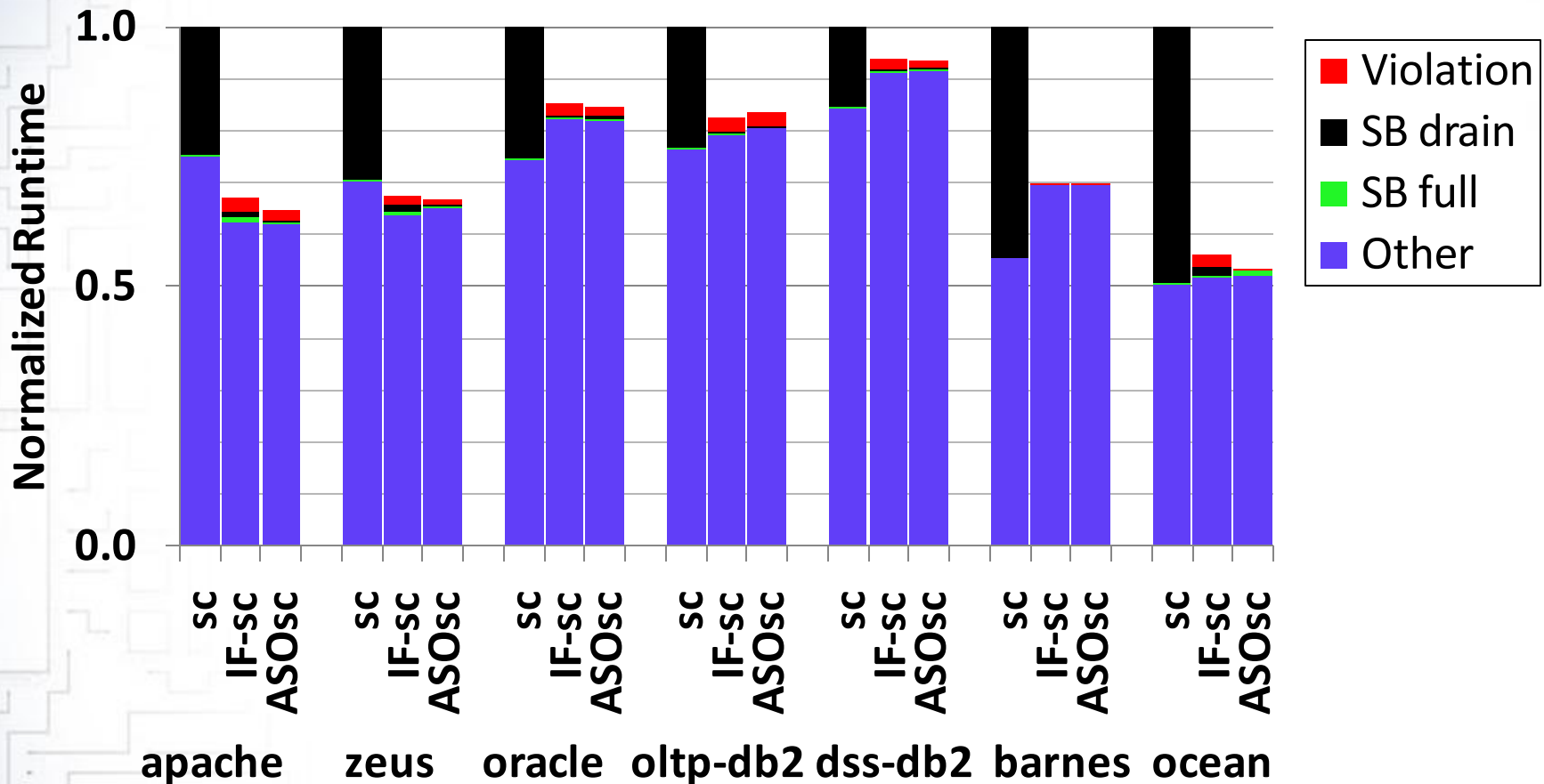
# Strong Ordering Performance (SC)



**Violations are negligible (3% slowdown from IF-RMO)**

**How does this compare to prior work?**

# Strong Ordering Performance (SC)



**Comparison to Atomic Sequence Ordering [Wenisch`07]:**  
**Both eliminate stalls**



# ASO & InvisiFence: Design Comparison

## ASO [Wenisch'07]

- Fine-grained tracking
  - 1K-entry store buffer
  - 10 KB
- Lengthy commit
  - Atomically drain SB to L2
  - Multiple checkpoints
- Changes to L1
  - Mult. per-block R/W bits
  - Write-through
  - Per-word valid bits

## InvisiFence

- Coalesced tracking
  - 8-entry store buffer
  - < 1 KB
- Constant-time commit
  - Flash-clear bits
  - Single checkpoint
- Changes to L1
  - Single per-block R/W bits
  - Clean to L2

**Both eliminate stalls, but InvisiFence hardware simpler**

# Roadmap

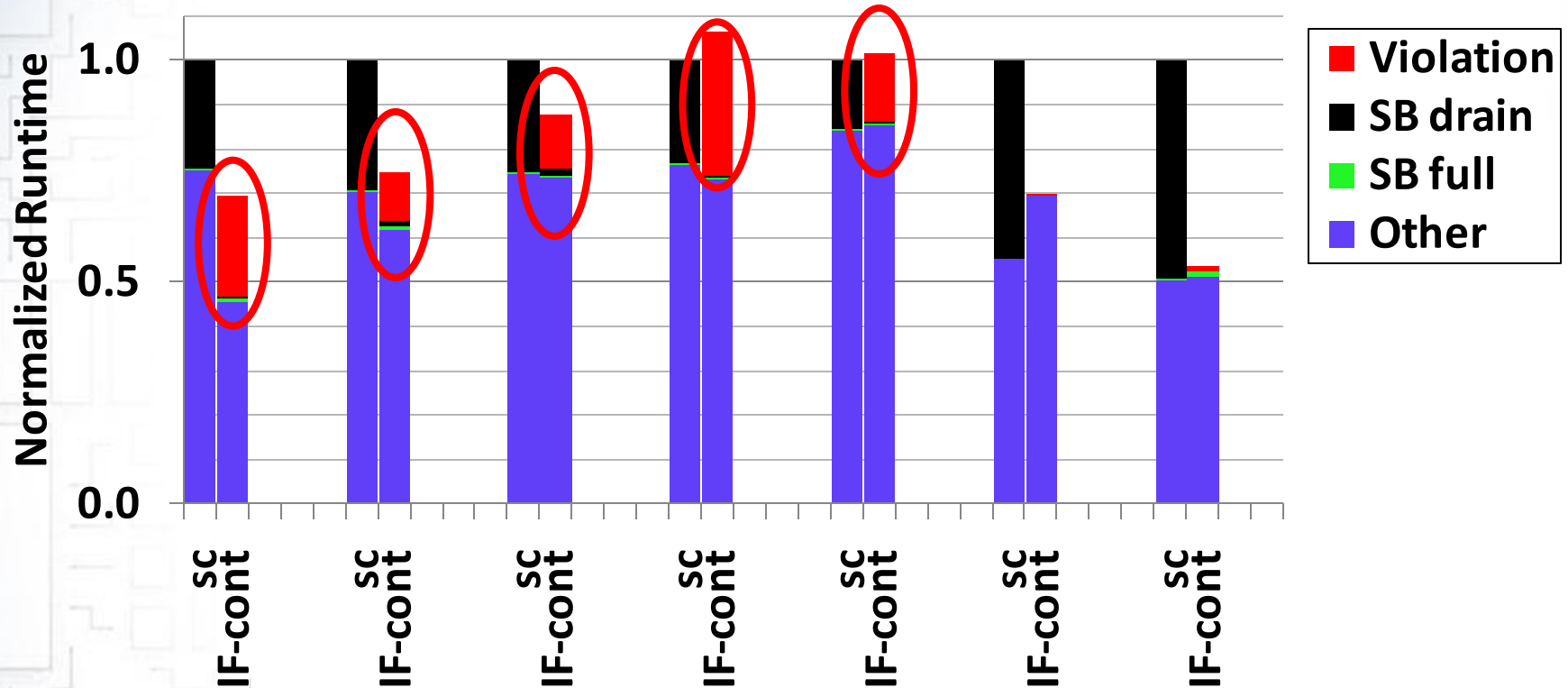
- InvisiFence for weak ordering
- Generalizing InvisiFence to stronger models
- **Subsuming in-window speculation**
- Conclusions

# Key Idea: Continuous Speculation

[Hammond'04, Ceze'07]

- Prior work: subsume LSQ snooping via **continuous spec.**
  - Execution divided into continuous speculative chunks
  - Deep spec. tracks loads from execution to chunk commit
  - Commit a chunk once all stores complete & all loads retire
- Existing designs acquire store permissions at commit
  - Lazy conflict detection (lowers vulnerability to violations)
  - Shown to be useful for other applications (TM, debugging, ...)
  - Requires extensions to conventional memory systems
- InvisiFence can also support continuous speculation
  - Eliminates LSQ snooping with local commit
  - Like prior work, pipelines commit with second checkpoint

# Continuous Speculation Performance

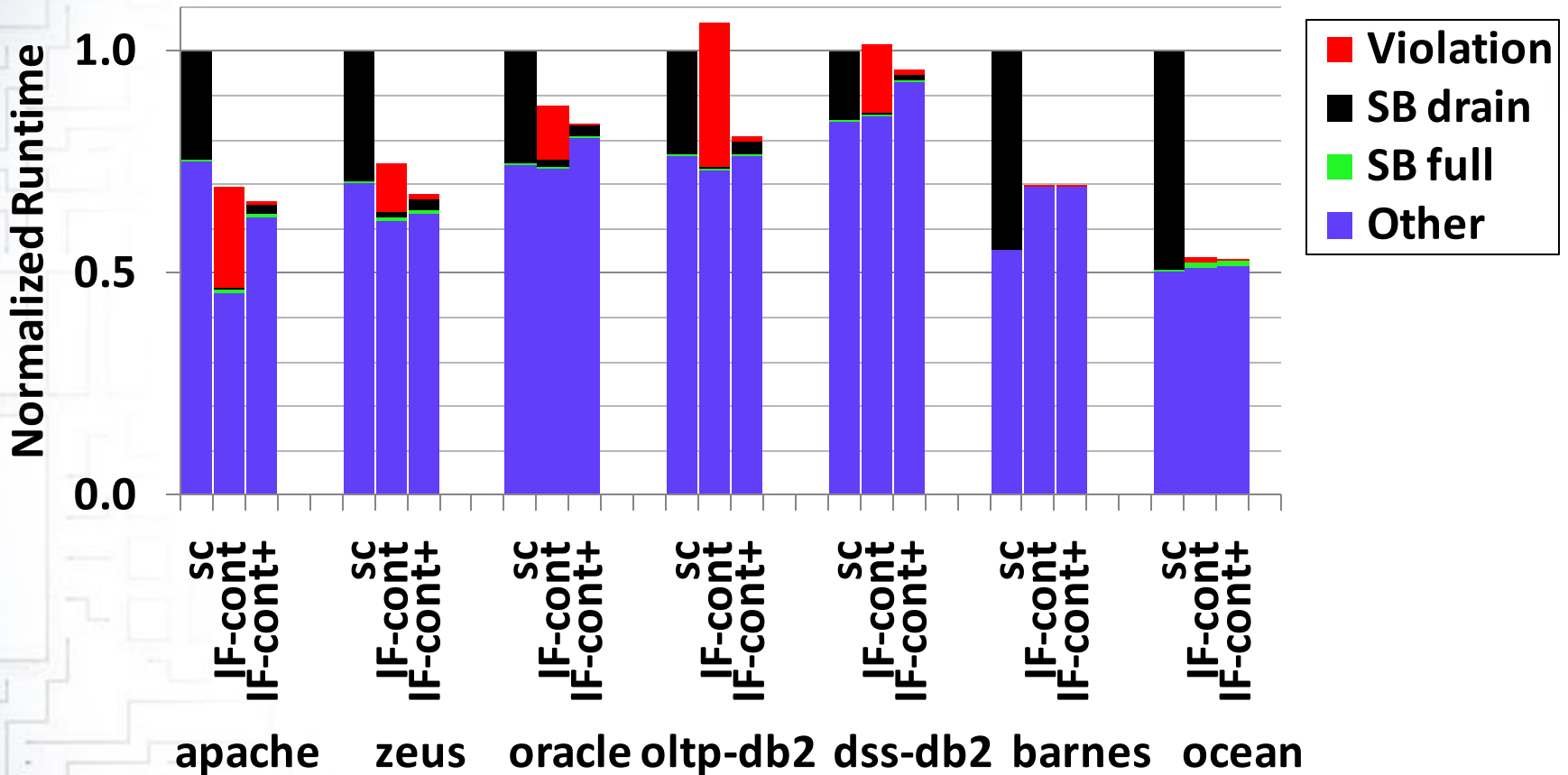


apache zeus oracle oltp-db2 dss-db2 barnes ocean

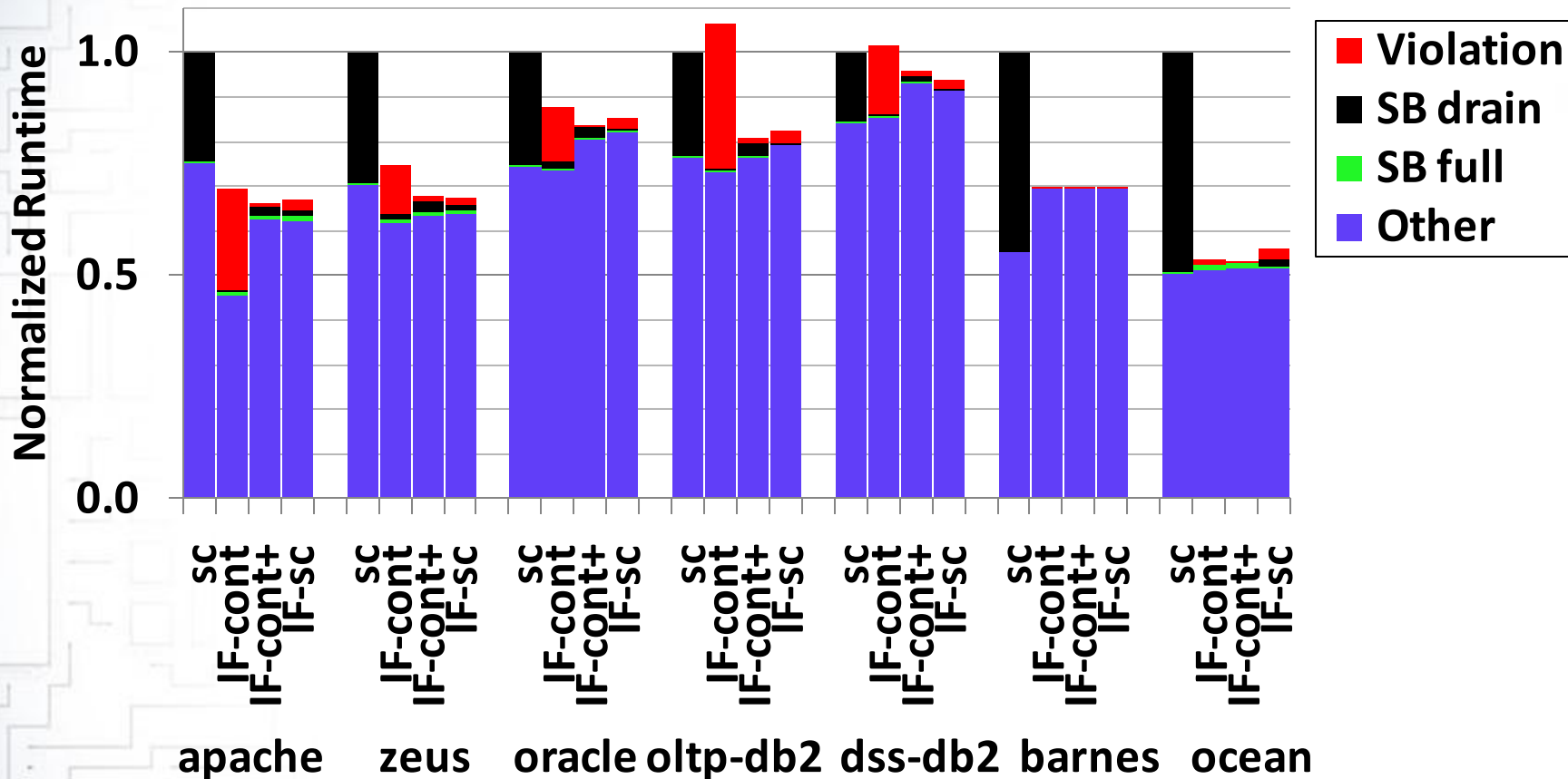
To reduce rollbacks: “Commit on Violation”

- Temporarily defer conflicting requests

# Continuous Speculation Performance



# Continuous Speculation Performance



**IF-cont+ (with commit on violation) achieves IF-sc performance without LSQ snooping**

# Conclusions

InvisiFence eliminates stalls from weak ordering

- Without per-store buffering
- With fast & simple commit and abort
- Using a conventional memory system

Same hardware can provide strong ordering

- Adjust policy to start speculation
- InvisiFence-SC: within 3% of InvisiFence-RMO

Subsume in-window speculation mechanisms

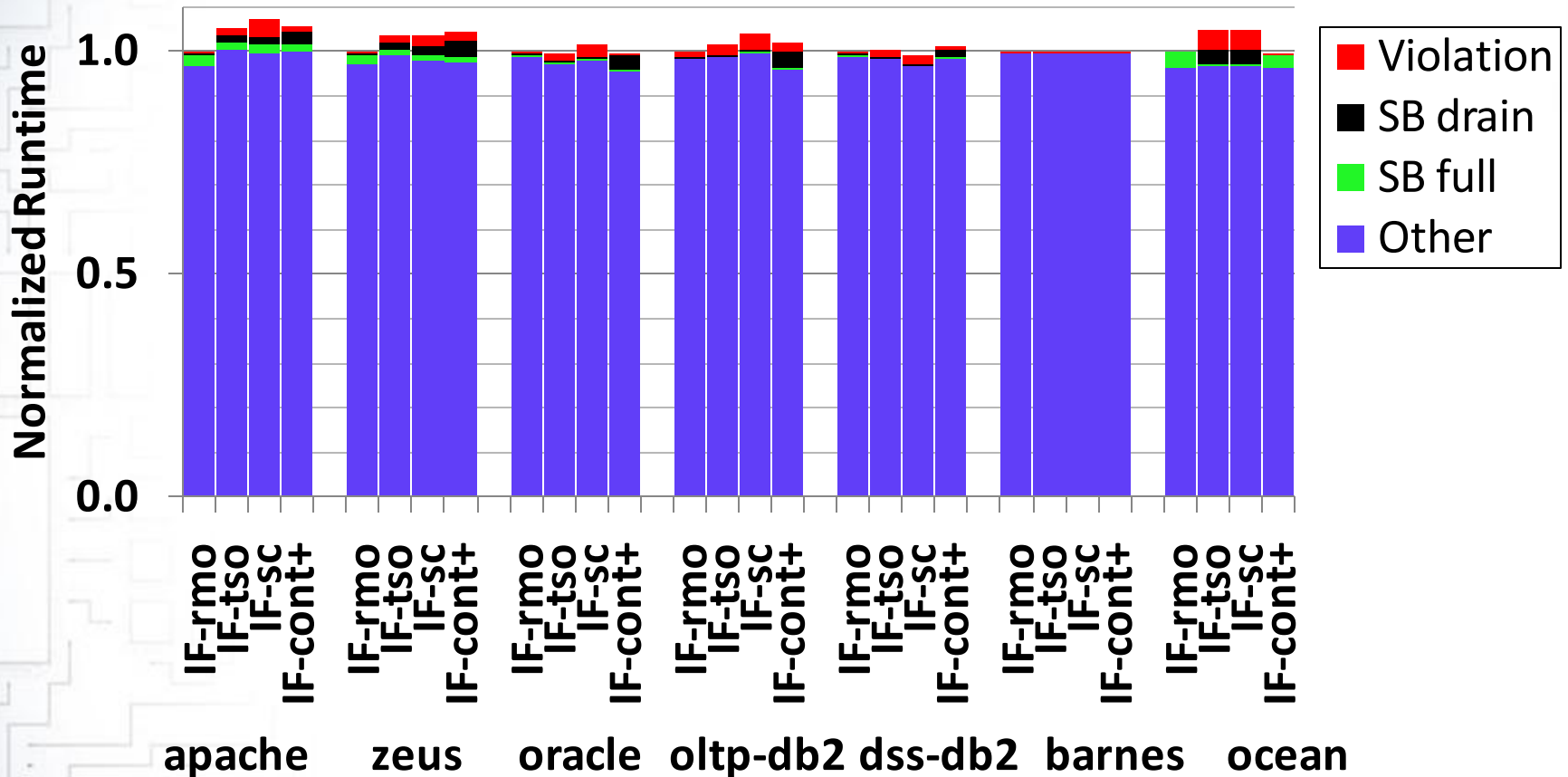
- Add continuous speculation + commit on violation
- InvisiFence-SC performance without LSQ snooping



UNIVERSITY of PENNSYLVANIA  
ARCHITECTURE + COMPILERS GROUP

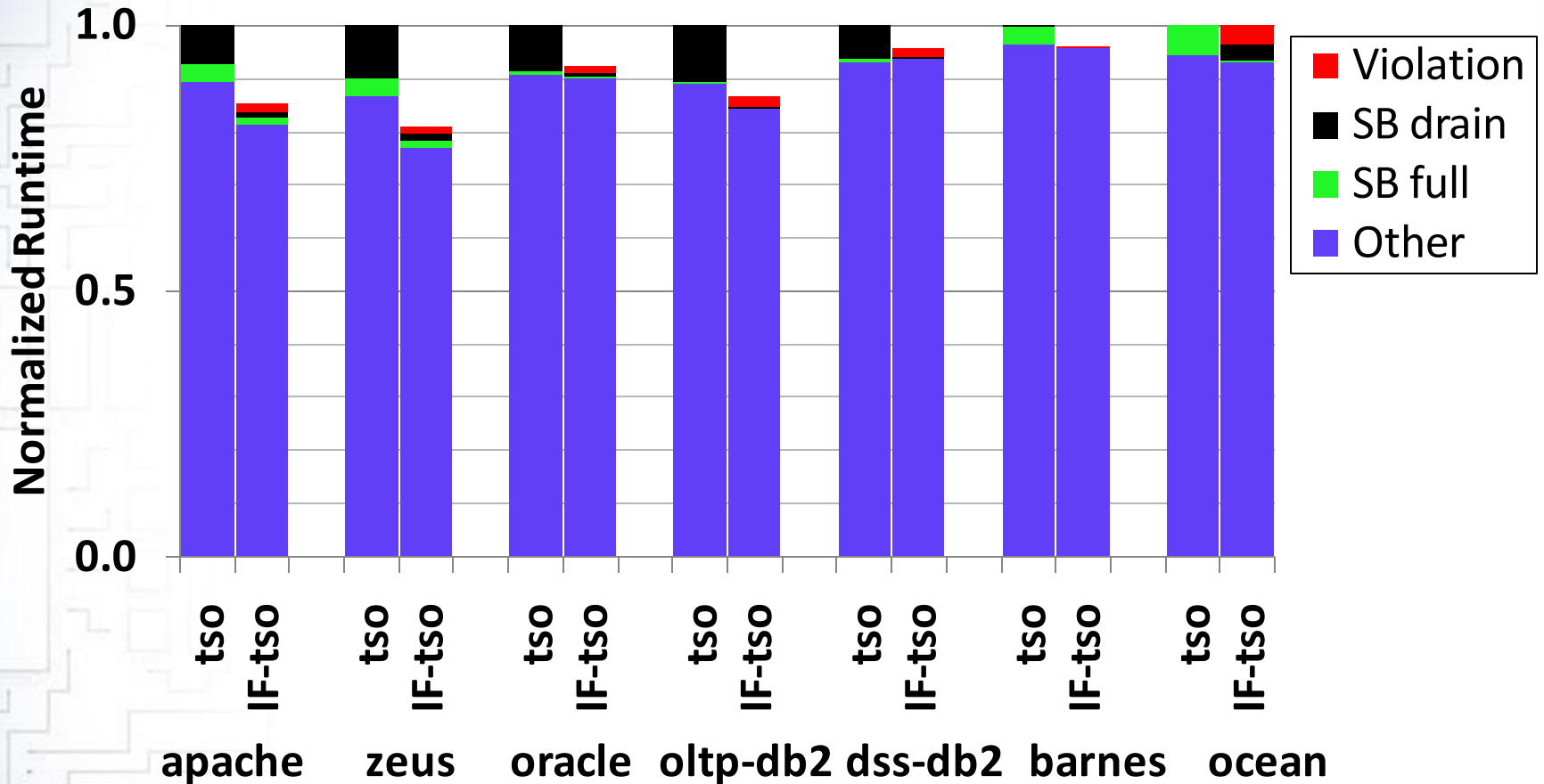


# InvisiFence Normalized Runtime Results



**Same perf. for any model; identical hardware**

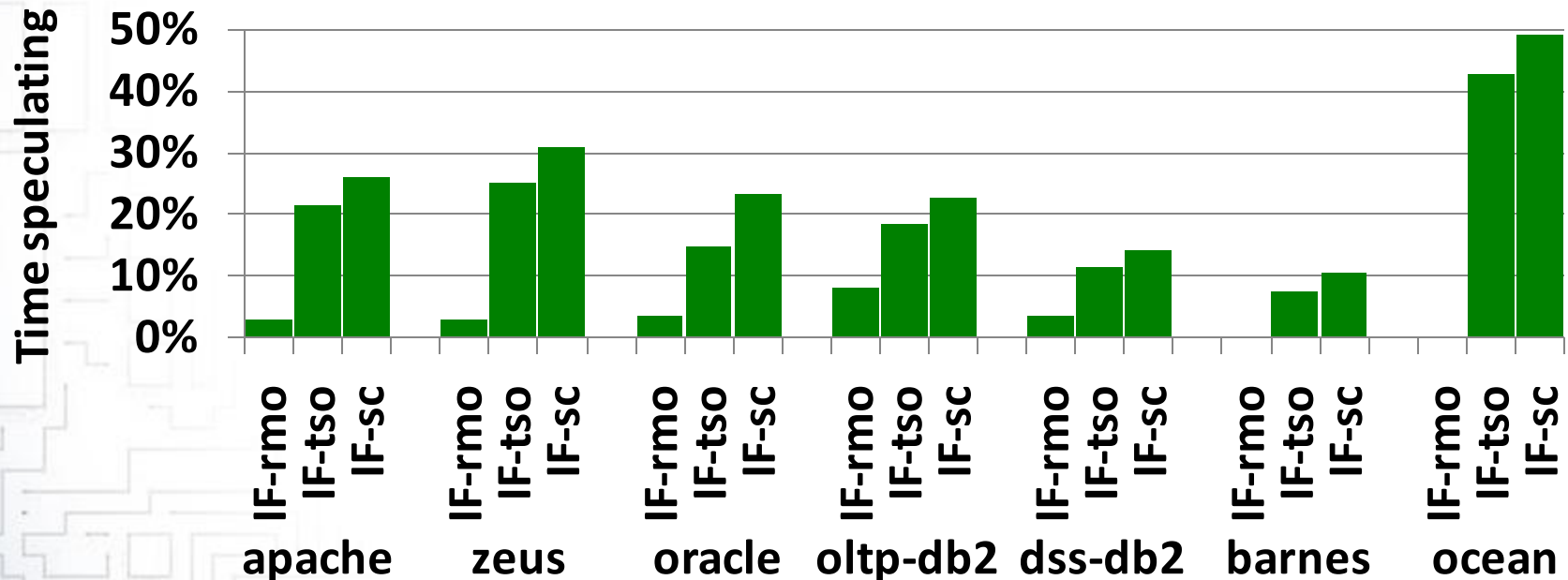
# TSO Performance



**InvisiFence eliminates ordering stalls, improves MLP**

# Generalizing InvisiFence for Strong Ordering

- Strong models impose additional ordering constraints
- InvisiFence treats these constraints as “implicit fences”
- Increases speculation frequency...



**No other hardware changes**