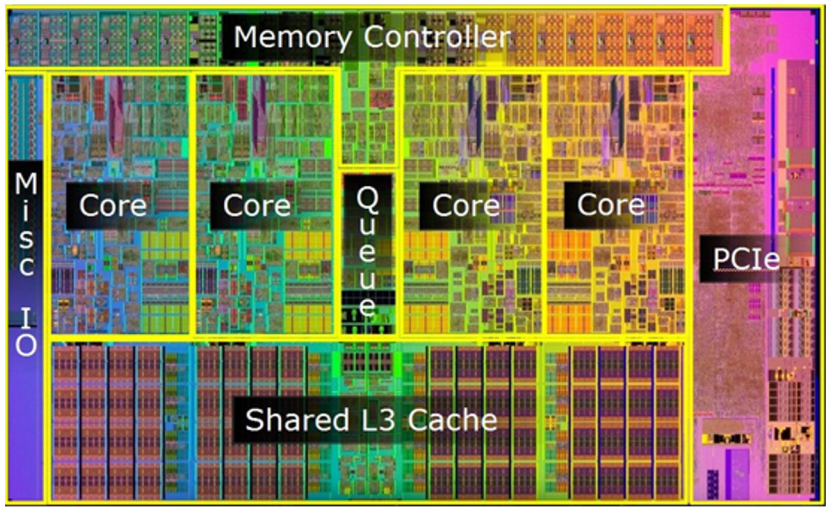
Persistent Snapshot Isolation with Unlimited Reads on Hardware Transactional Memory

Alexandro Baldassin, João Barreto, Daniel Castro,
Miguel Figueiredo, Paolo Romano

SC23
Denver, CO | i am hpc.
HMEM workshop

After the multi-core revolution…
… the persistent memory (PM) revolution



Multi-core CPUs

require

concurrency control

Persistent objects on PM

require

failure atomicity

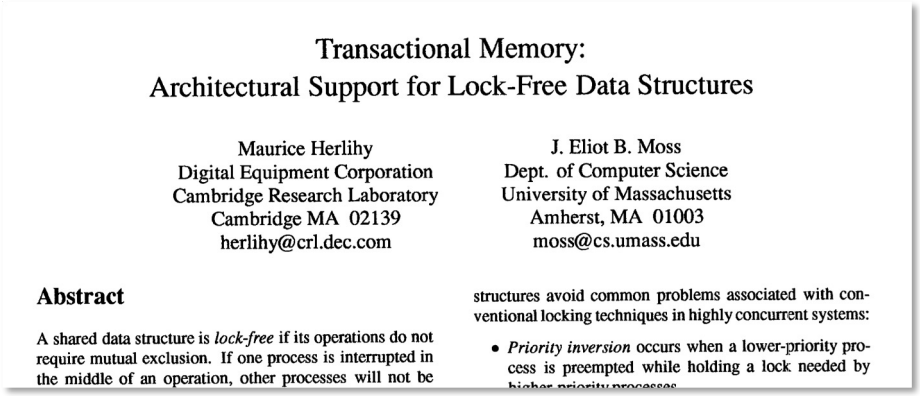
The **transactional memory** abstraction

- Concurrency control: important advantages over fine-grained locks
- Failure atomicity: popular abstraction for failure-atomic sections in PM literature

```
1  TX_BEGIN(pool) {
2      TX_ADD_DIRECT(&D_RW(FOO));
3      TX_ADD_DIRECT(&D_RW(BAR));
4      D_RW(FOO) = D_RO(FOO)-100;
5      D_RW(BAR) = D_RO(BAR)+100;
6  } TX_END
```

A persistent memory transaction with PMDK

The promise of hardware transactional memory (HTM)



Transactional Memory:
Architectural Support for Lock-Free Data Structures

Maurice Herlihy
Digital Equipment Corporation
Cambridge Research Laboratory
Cambridge MA 02139
herlihy@crl.dec.com

J. Eliot B. Moss
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
moss@cs.umass.edu

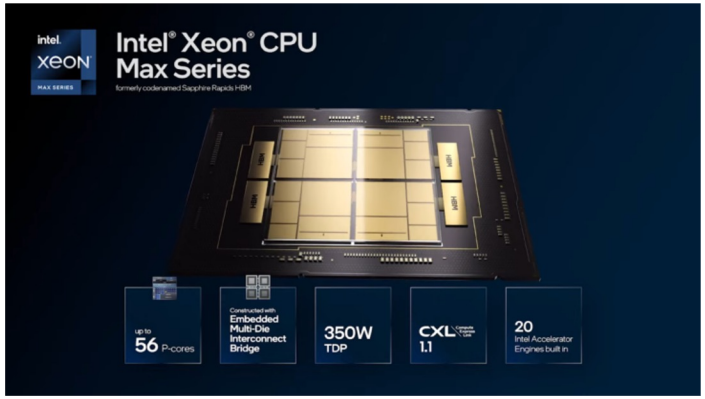
Abstract

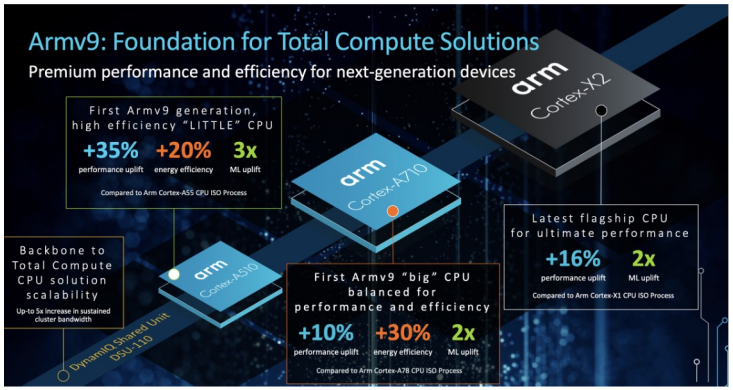
A shared data structure is *lock-free* if its operations do not require mutual exclusion. If one process is interrupted in the middle of an operation, other processes will not be

structures avoid common problems associated with conventional locking techniques in highly concurrent systems:

- *Priority inversion* occurs when a lower-priority process is preempted while holding a lock needed by

ACM SIGARCH Computer Architecture News, 1993





Two fundamental obstacles of HTM

- Limited read/write-set capacity
  - Transactions that exceed such capacity are aborted
  - Single global lock fallback
- Not failure-atomic
  - HTM atomically writes to volatile CPU caches, not PM
  - Use persistent software TM alternatives instead (e.g. PMDK)

Two successful research avenues

- In recent years, substantial achievements in mitigating each issue
- Insight: we can work around each limitation of an **unmodified** HTM implementation by means of **software-based extensions**

Talk outline

- State-of-the-art on each research avenue
- How can we unify both avenues?

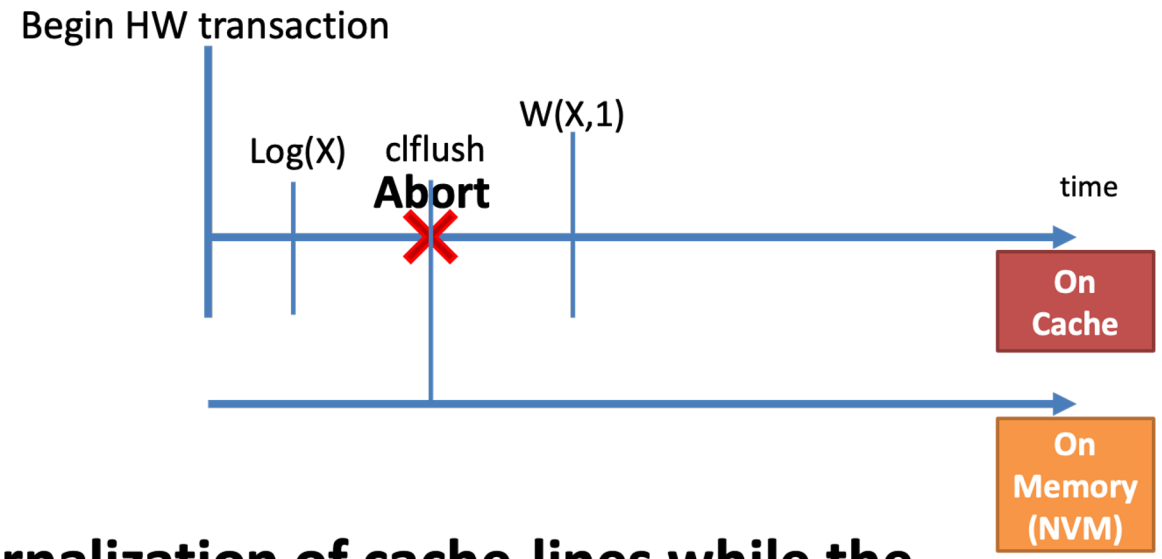
State of the art on each research avenue

Durable HTM transactions

HTM transactions with unlimited reads

Durable hardware transactions

- Writes to PM also added to a durable redo log (in PM)
- However, we cannot flush redo log entries to PM before the HTM commits the transaction

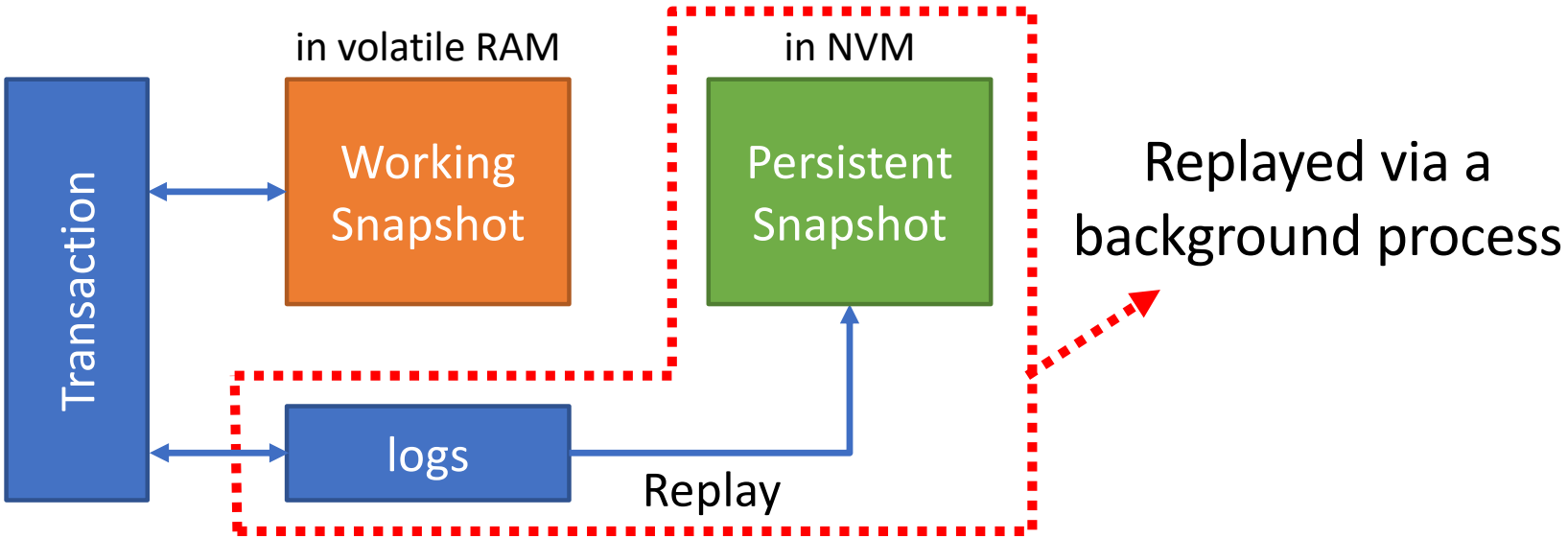


Externalization of cache-lines while the transactions is running causes it to abort!

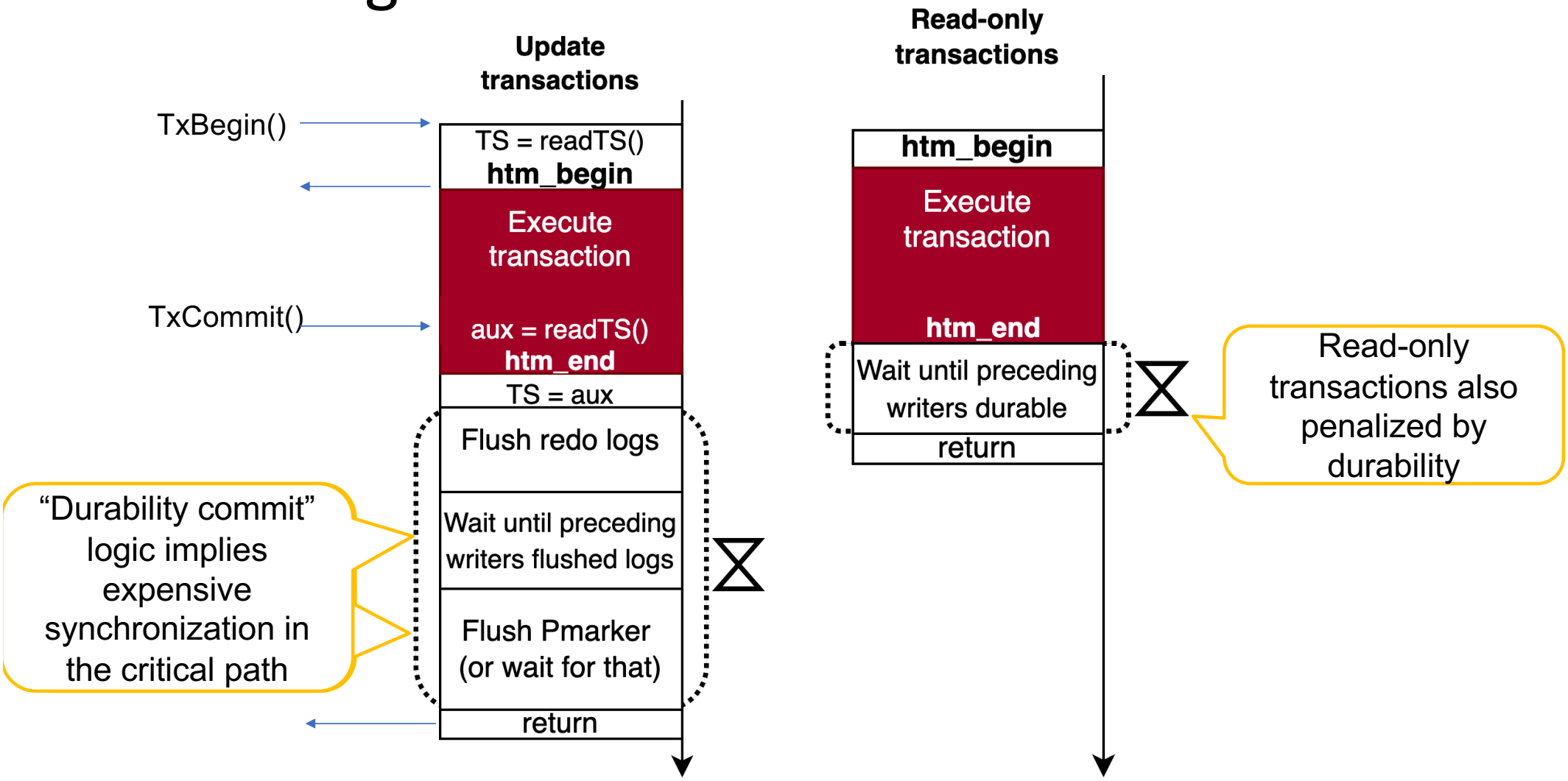
Durable hardware transactions with SPHT

[Castro et al., FAST'21]

- Application writes in a **volatile working snapshot**
- Logged writes are replayed asynchronously to produce a consistent **persistent snapshot** on PM



SPHT: algorithm



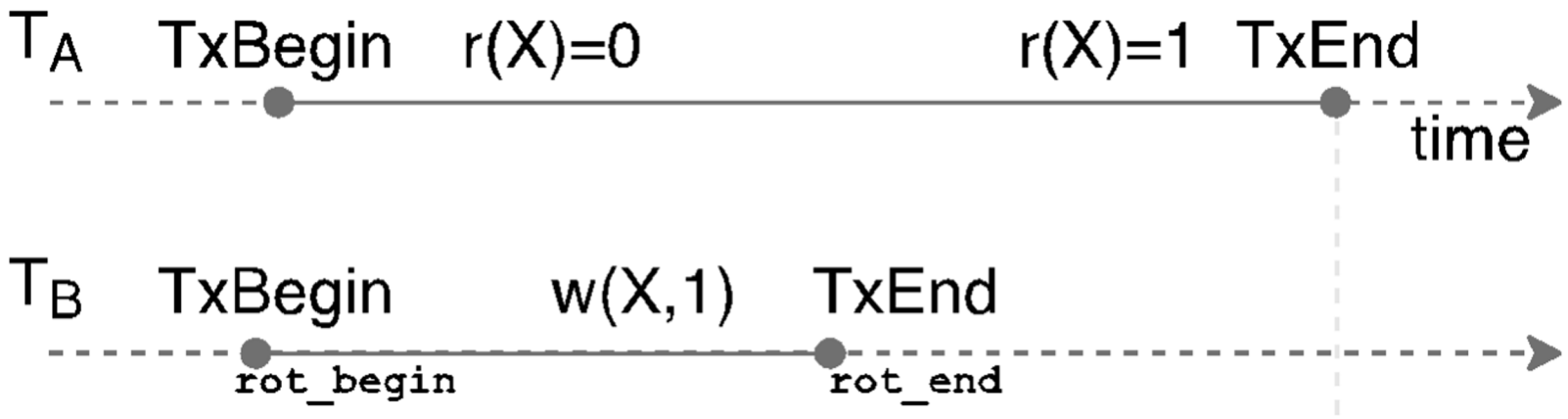
State of the art on each research avenue

Durable HTM transactions

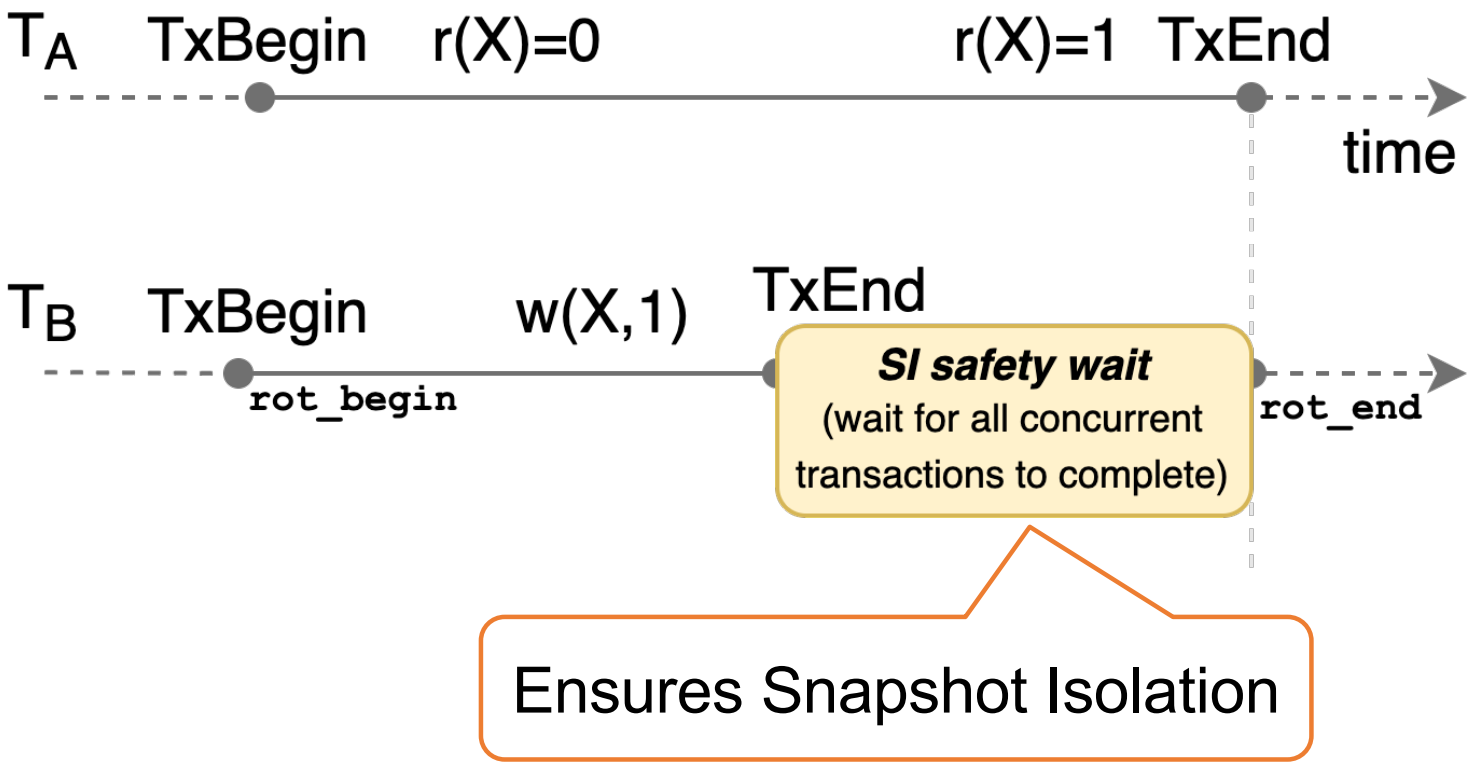
HTM transactions with unlimited reads

Hardware transactions with unlimited reads with SI-HTM [Filipe et al., PPoPP'19]

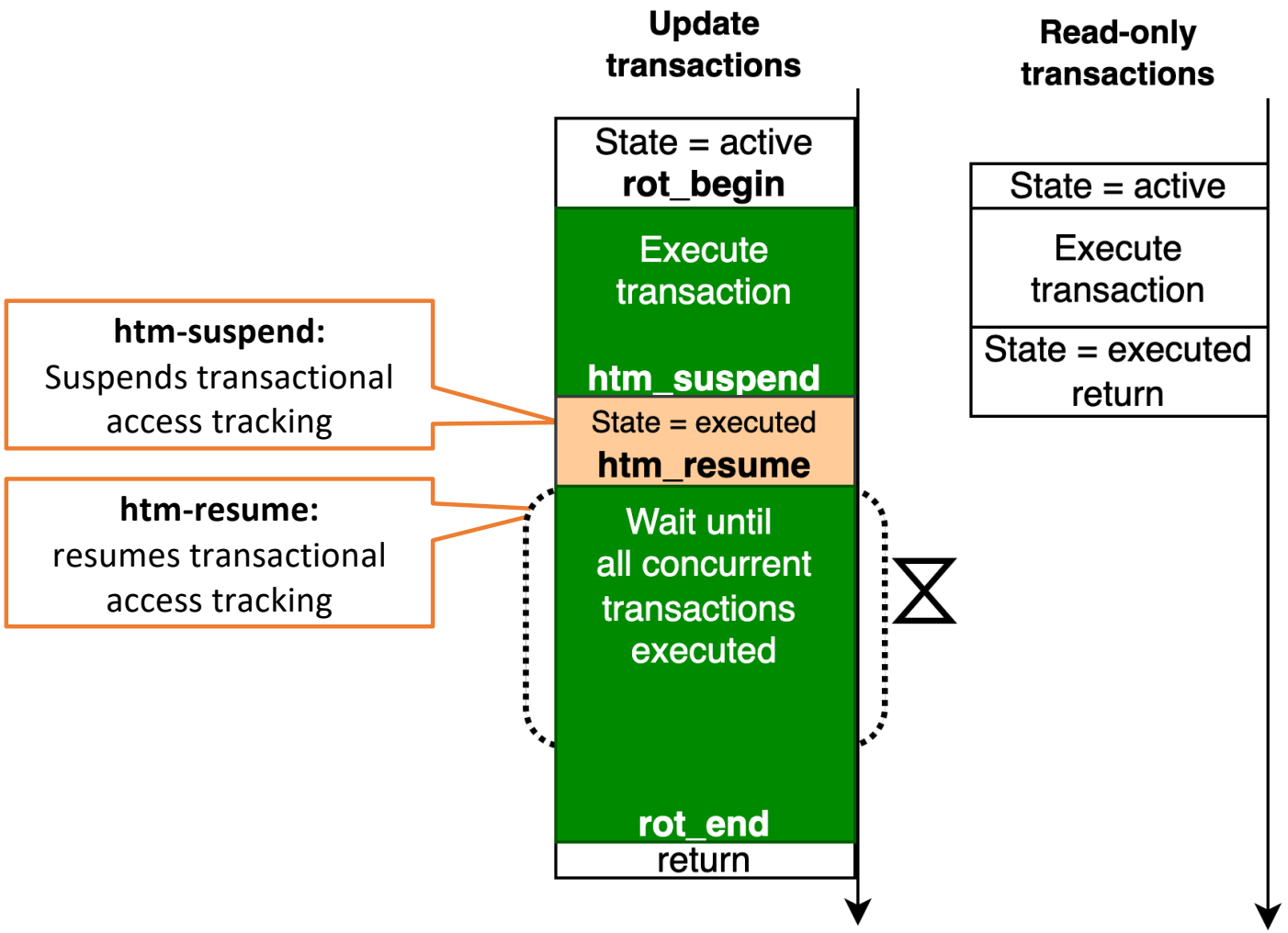
- Explores advanced primitives available IBM POWER's HTM
- **Update transactions** run in **roll-back only transactions** (ROT), which **don't track loads**
- **Read-only transactions** run in **no hw transaction**
- **Unlimited reads** …but could lead to **consistency anomalies**



Waiting to prevent consistency anomalies



SI-HTM: algorithm



Can we combine the state-of-the-art from both avenues?

Durable hardware transactions (SPHT)

Unlimited Reads (SI-HTM)

Can we combine the state-of-the-art from both avenues?

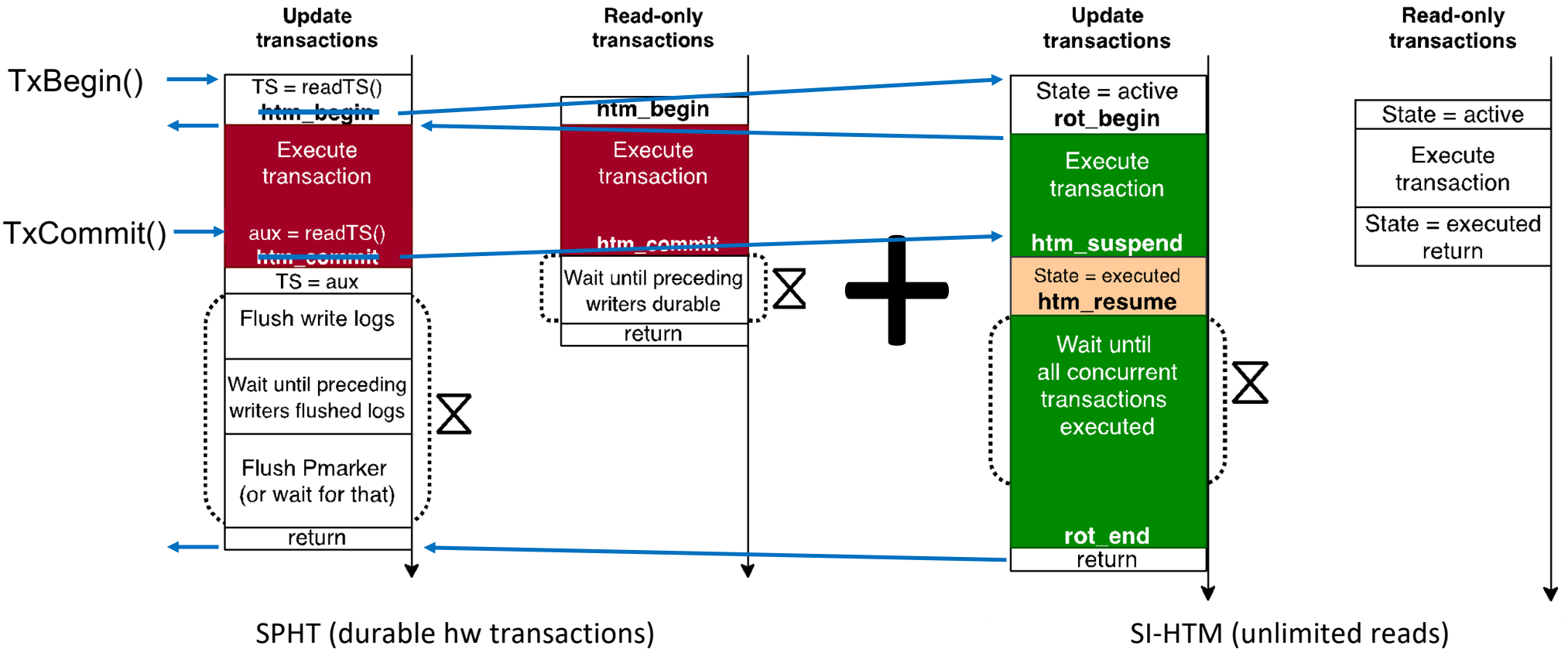
Durable hardware transactions (SPHT)

Unlimited Reads (SI-HTM)

Rest of my talk

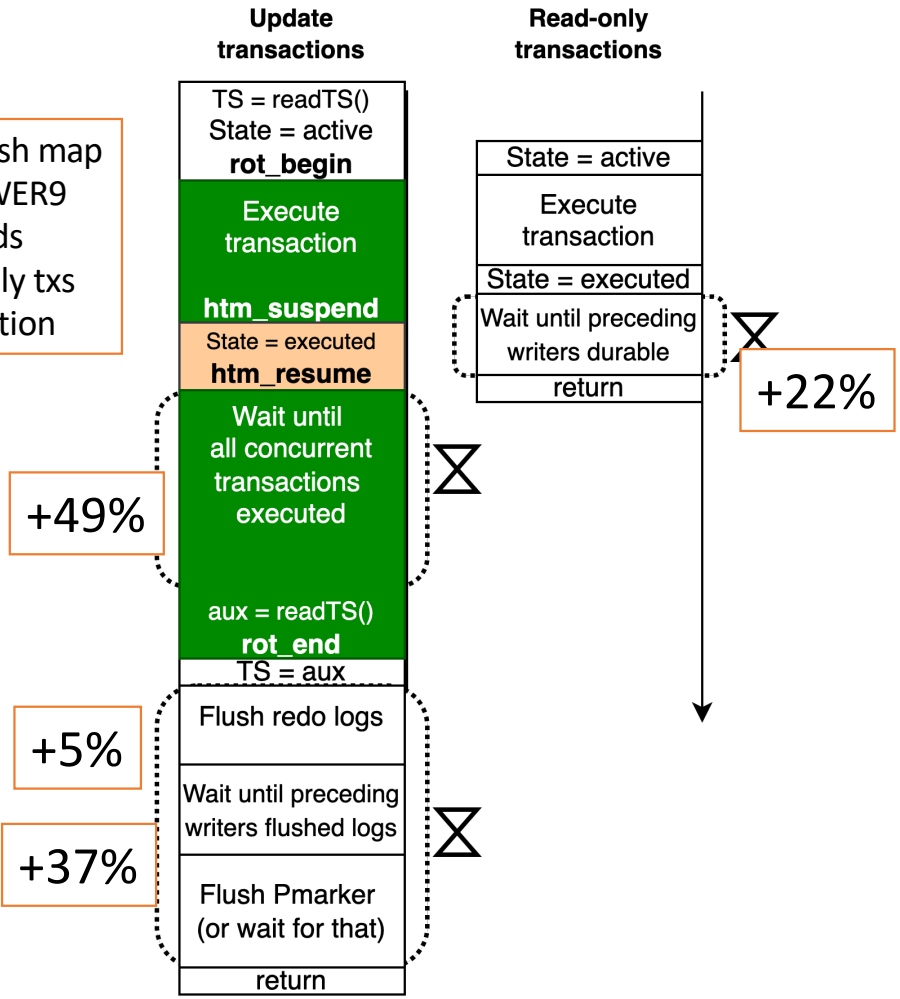
1. Yes, we can combine both solutions
2. Yes, we can **efficiently** combine both solutions by **deeply rethinking** them

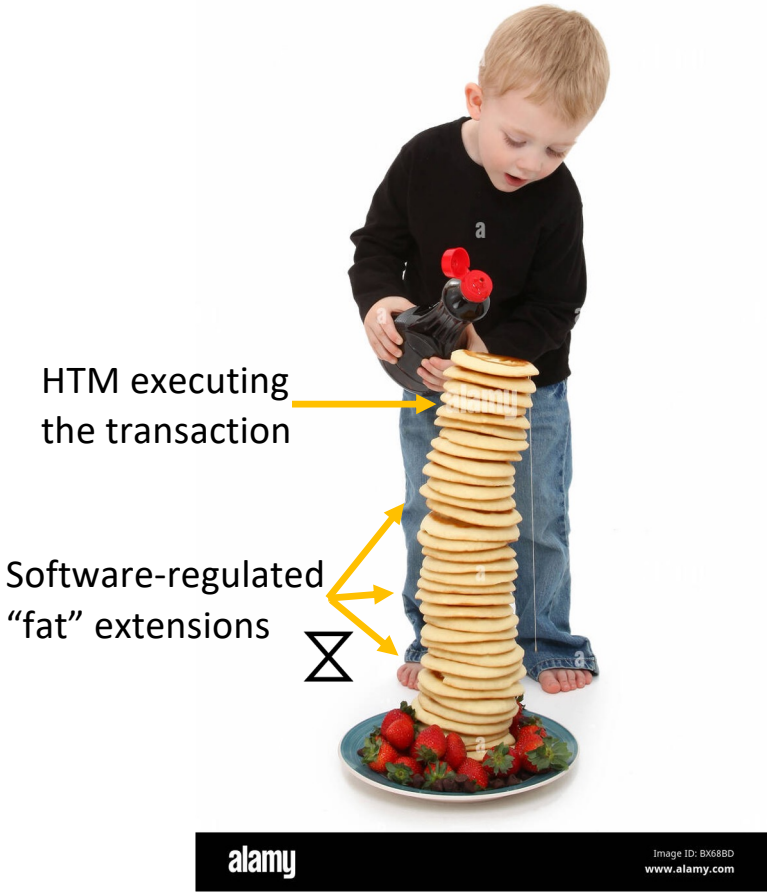
Directly stacking SPHT on top of SI-HTM



Directly stacking SPHT on top of SI-HTM

Concurrent hash map
on IBM POWER9
16 threads
90% read-only txs
low contention





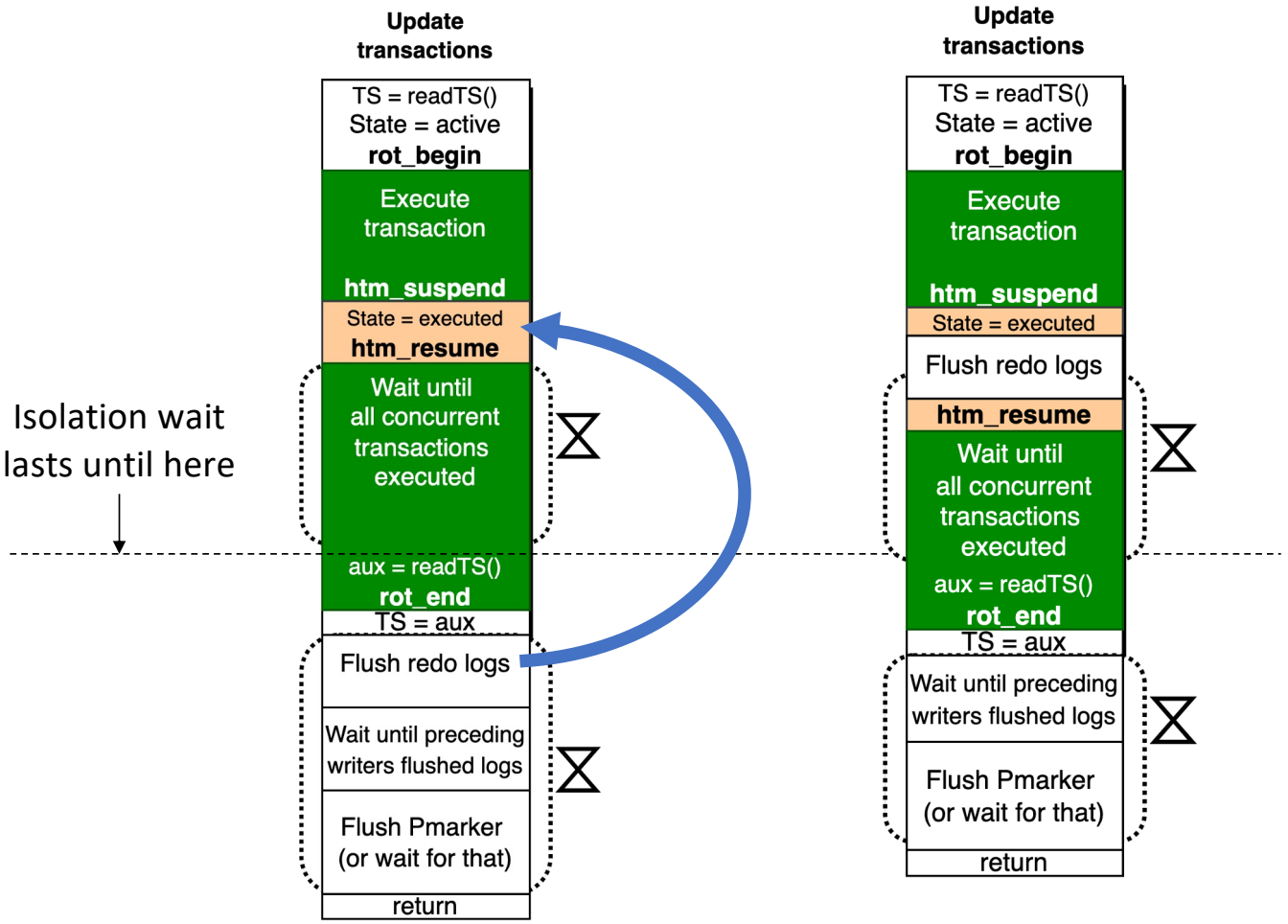
Persistent Snapshot Isolation (PSI)

PSI: an **efficient** combination of durability and unlimited reads on HTM

3 novel optimizations to reduce the ”software fat” of SPHT+SI-HTM:

1. Opportunistic log flushing
2. Reduced durability wait for read-only transactions
3. Non-transactional logical durability timestamps

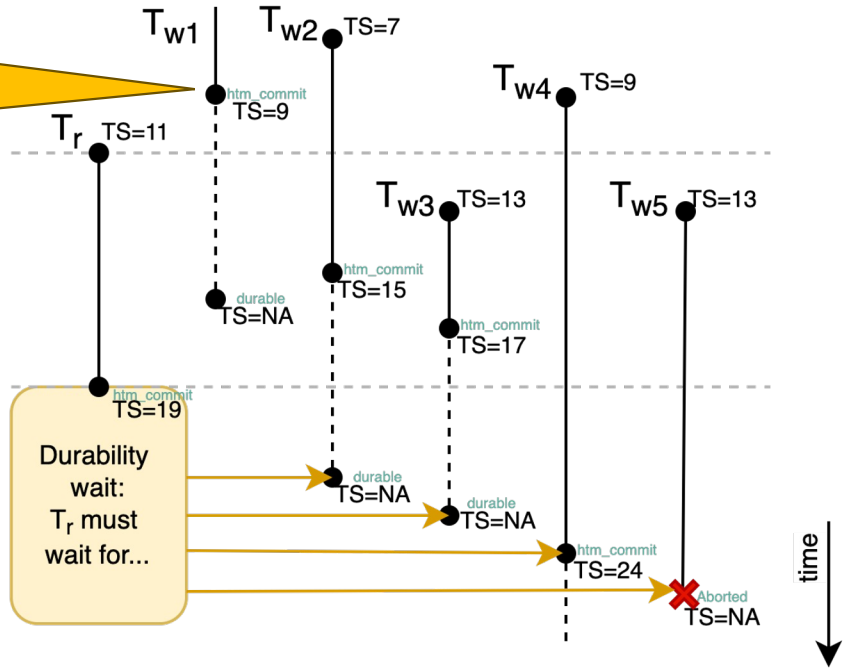
Technique #1: Opportunistic log flushing



- Hides the log flush latency
- No harm as long as waiting window is longer

Technique #2: Reduced durability wait for RO txs

In PSI, T_r only needs to wait for T_{w1} durable



Recall: In SI-HTM (and in PSI) any concurrent update transaction T_w postpones its commit (in HTM) until T_r has completed

Insight: T_r will never observe the writes of T_w, so T_r doesn't need to wait for T_w to become durable

Technique #3:
Optimized durability commit

- More scalable than SPHT's durability commit logic
- Based on a logical timestamp obtained by an atomic_inc instruction executed in **suspended mode**

Two main advantages:

- **Minimal synchronization in the critical path** (after HTM commit)
- **Hides latency of atomic instruction** inside the SI safety wait

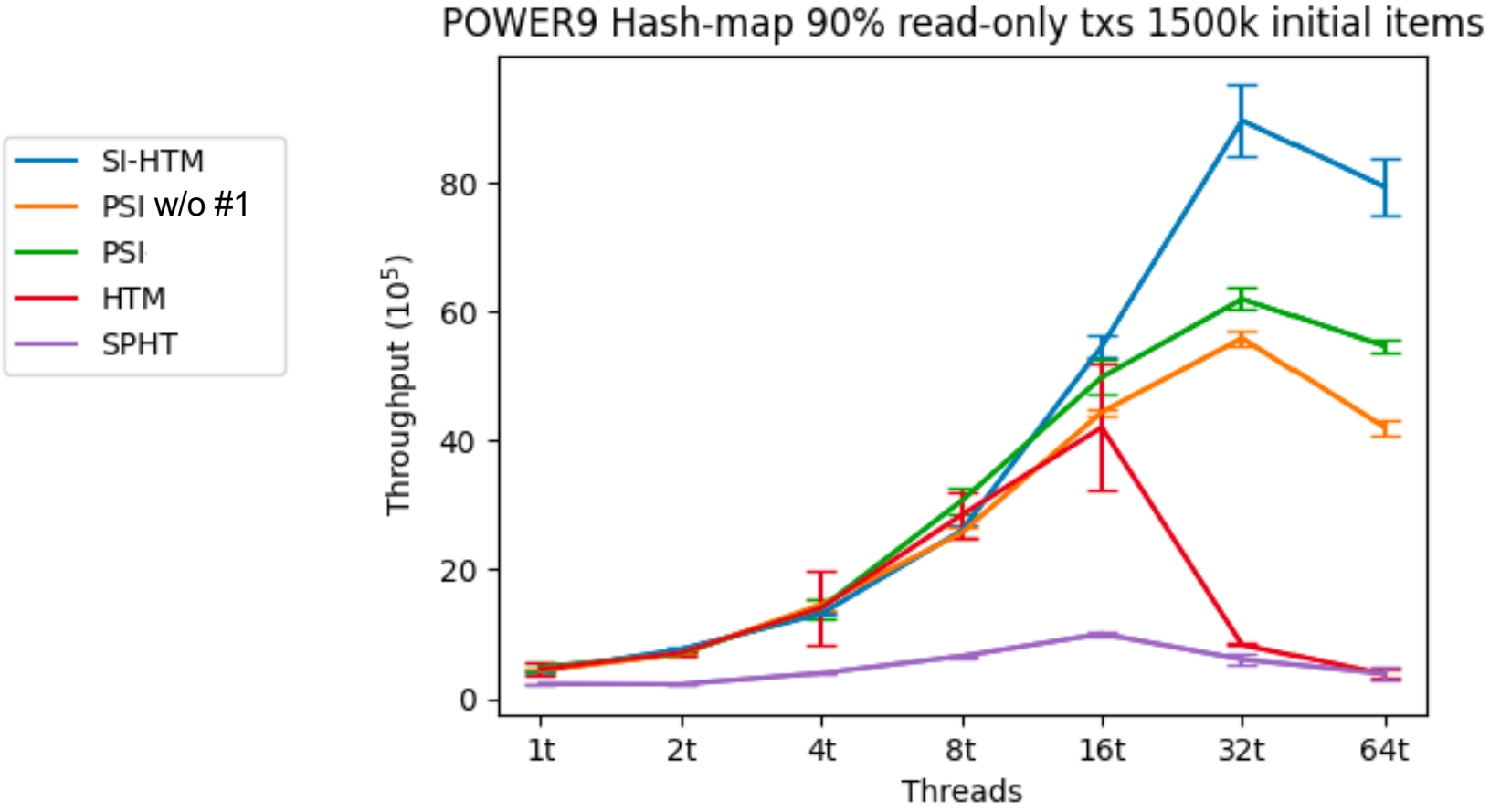
Evaluation

Experimental setup

- Dual-socket IBM POWER9
  - 2.3GHz, 1023GB RAM, 16x4 hw threads per socket
  - Running in a qemu VM (required for *suspend-resume*)

- Hash map benchmark
  - 10% insert/remove, 90% lookup
  - Low contention (512 buckets)
  - Large transactions, prone to capacity aborts (1.5M initial elements)

Results


POWER9 Hash-map 90% read-only txs 1500k initial items
SI-HTM
PSI w/o #1
PSI
HTM
SPHT
Throughput (10^5)
0
20
40
60
80
1t
2t
4t
8t
16t
32t
64t
Threads

Take-aways

- HTM is a powerful mechanism in multicore+PM architectures
- But hindered by **limited capacity** and no **failure atomicity**
- Recent advances work around **each limitation** but have **never been unified**

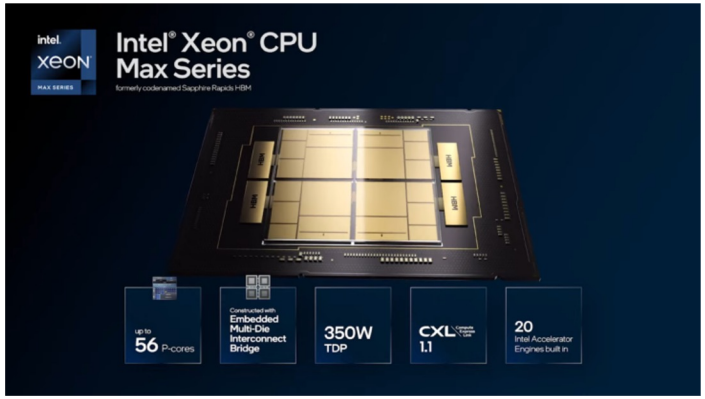
- **PSI:** is the first solution enabling **durable hardware transactions** with **unlimited reads**, with promising scalability
- The secret sauce: **3 novel techniques** that explore **the synergies between the building blocks** of PSI
- Future work: adapting PSI to the new Intel Sapphire Rapids HTM

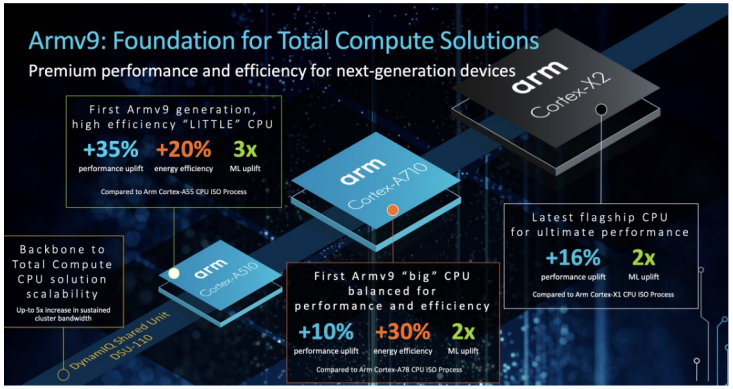
João Barreto | joao.barreto@tecnico.ulisboa.pt | https://www.dpss.inesc-id.pt/~jpbarreto/

Backup slides

What about other HTM implementations?

Existing HTMs	HTM can suspend access tracking?	PSI's benefits are possible?	
		On read-only txs	On update txs
ARM TME	no		
Intel TSX	loads	Yes	
PowerHTM	loads or full	Yes	Yes





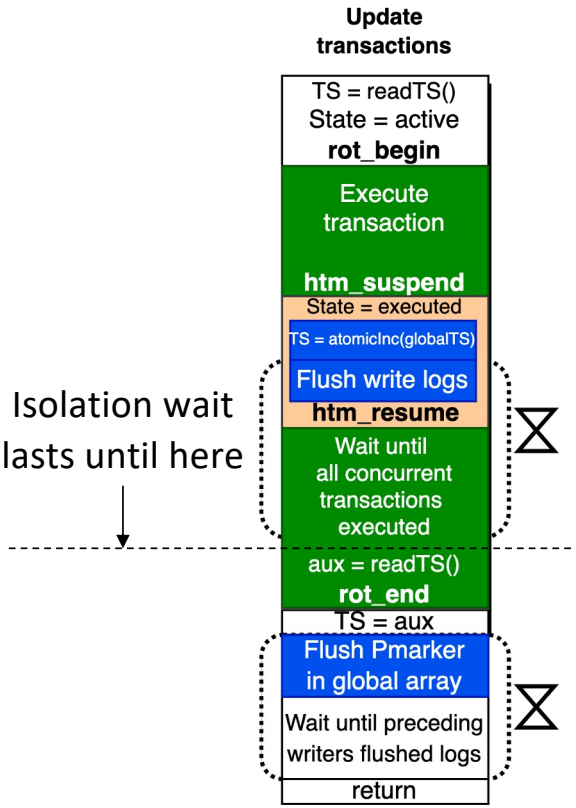
Technique #3: Optimized durability commit

- The challenge: how can T obtain its logical durability TS?
- Strawman solution:

```
htm_begin;
[...]
myTS=globalClock++;
tEnd;
```

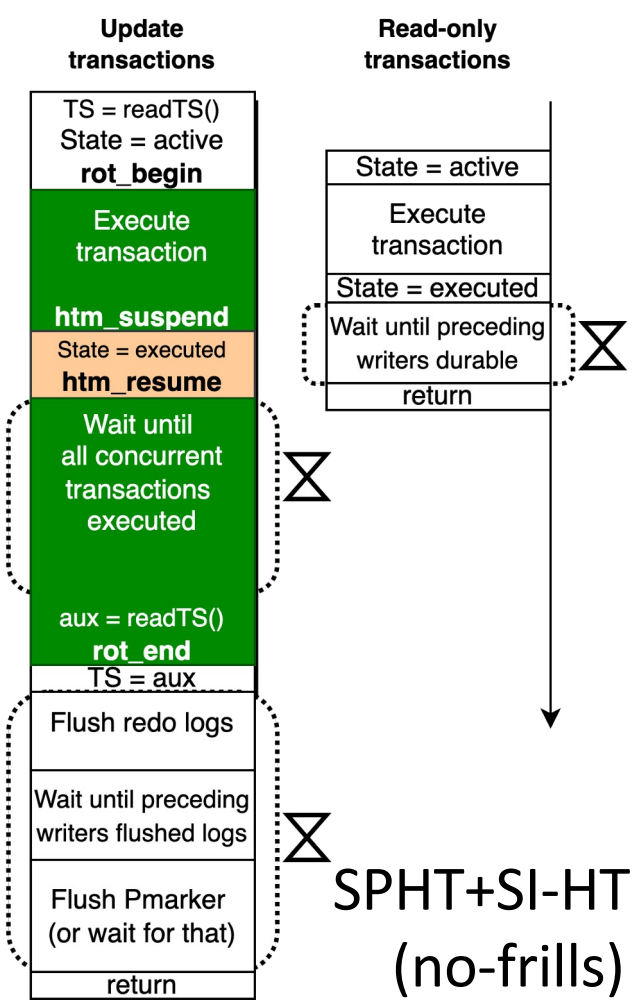
- Increases transactional conflicts, sacrifices scalability

- Our **synergistic** solution:

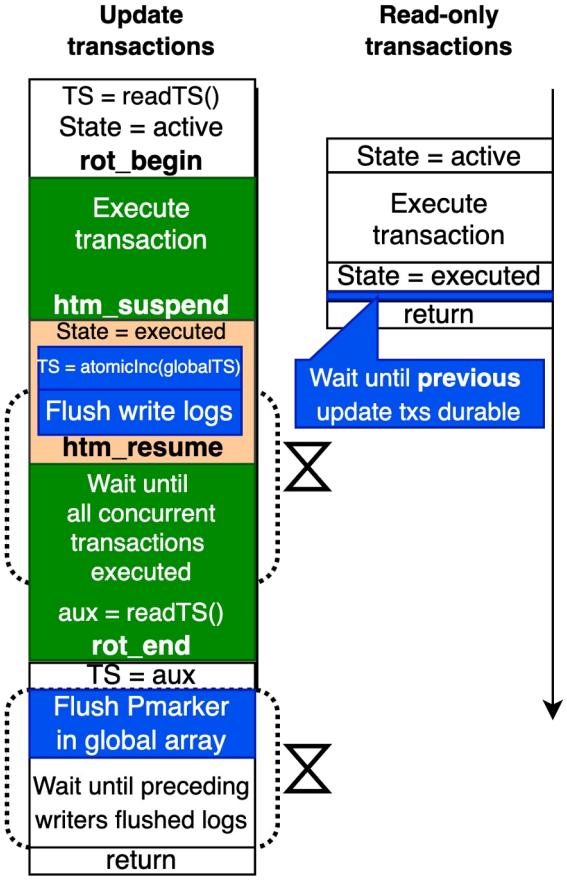


- No additional transactional conflicts
- Latency of atomic instruction hidden inside the waiting window
- No harm as long as waiting window is longer

Putting it all together



SPHT+SI-HTM
(no-frills)



PSI