Enhancing Efficiency of Hybrid Transactional Memory via Dynamic Data Partitioning Schemes

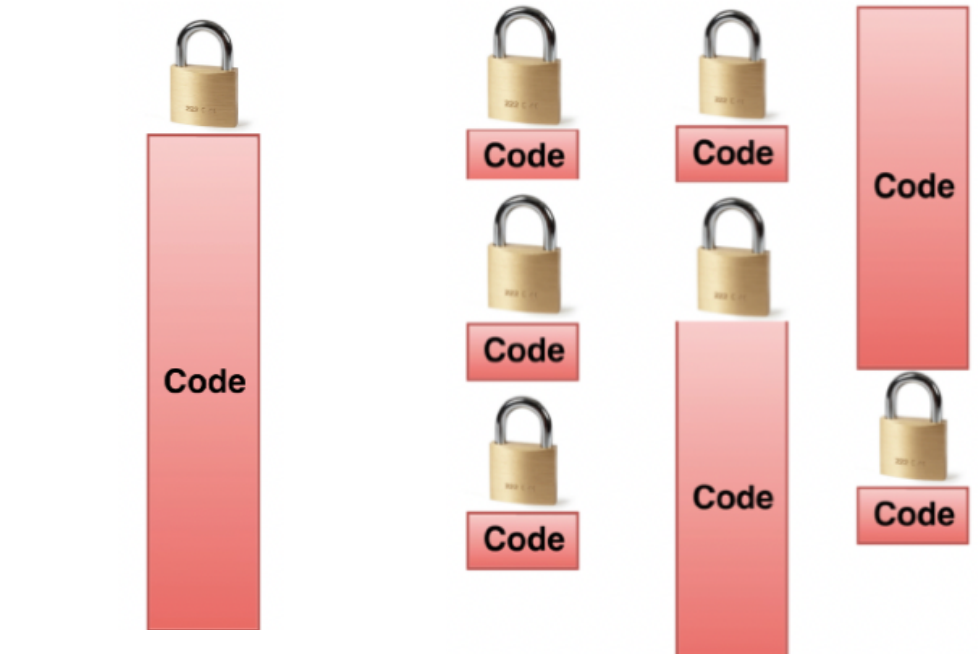
Pedro Raminhas, Shady Issa, Paolo Romano

pedro.raminhas@tecnico.ulisboa.pt

INESC-ID/Instituto Superior Técnico, University of Lisbon

Motivation

- Multicore processors are a mainstream technology
  - HPC
  - Laptops
  - Smartphones
- Hard to develop code that takes advantage of multicore processors

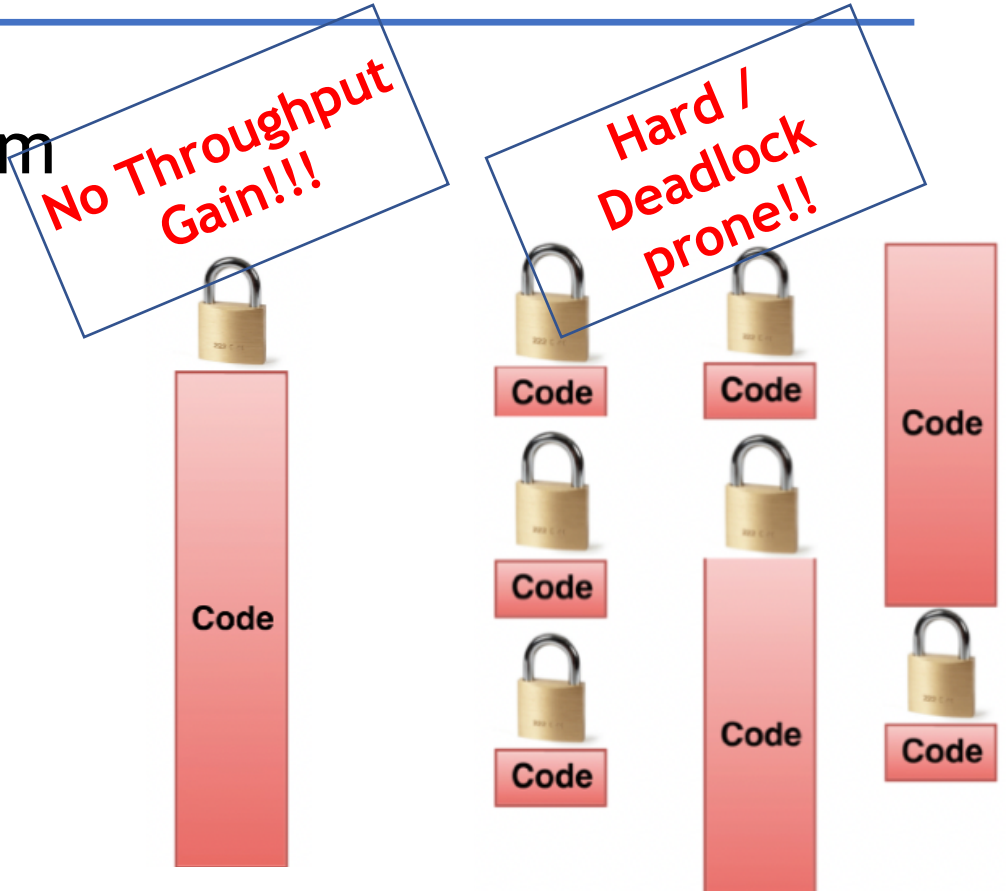


Coarse-Grained Locking

Fine-Grained Locking

Motivation

- Multicore processors are a mainstream technology
  - HPC
  - Laptops
  - Smartphones
- Hard to develop code that takes advantage of multicore processors



Background of TM

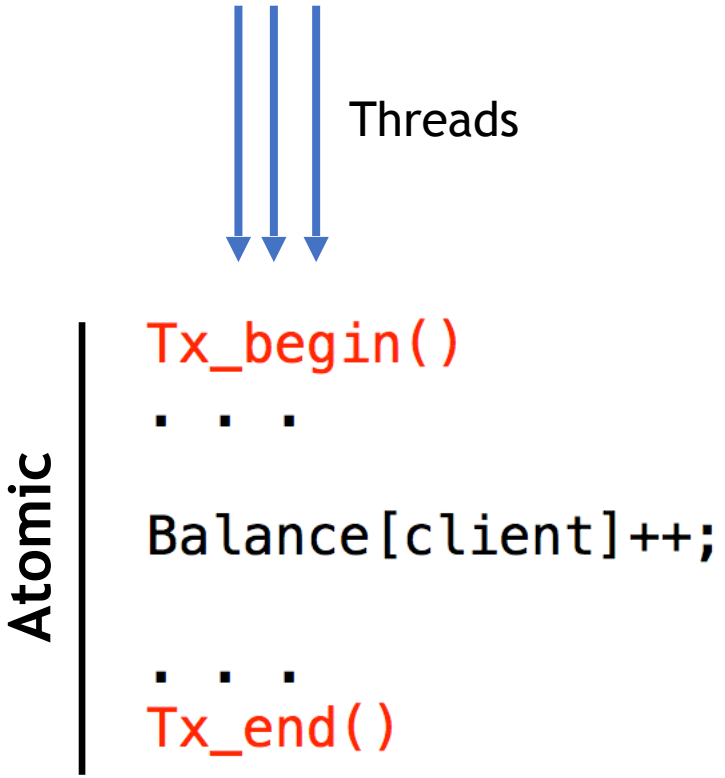
- Transactional Memory [ISCA' 93] promises:
  - Ease of use
  - performance equal or better than fine-grained locking

```
. . .

Balance[client]++;

. . .
```

- Extension of multi-processors' cache coherence protocols
  - Speculative run => apply the concept of **transactions** from databases to parallel code
  - The underlying system guarantees isolation and atomicity

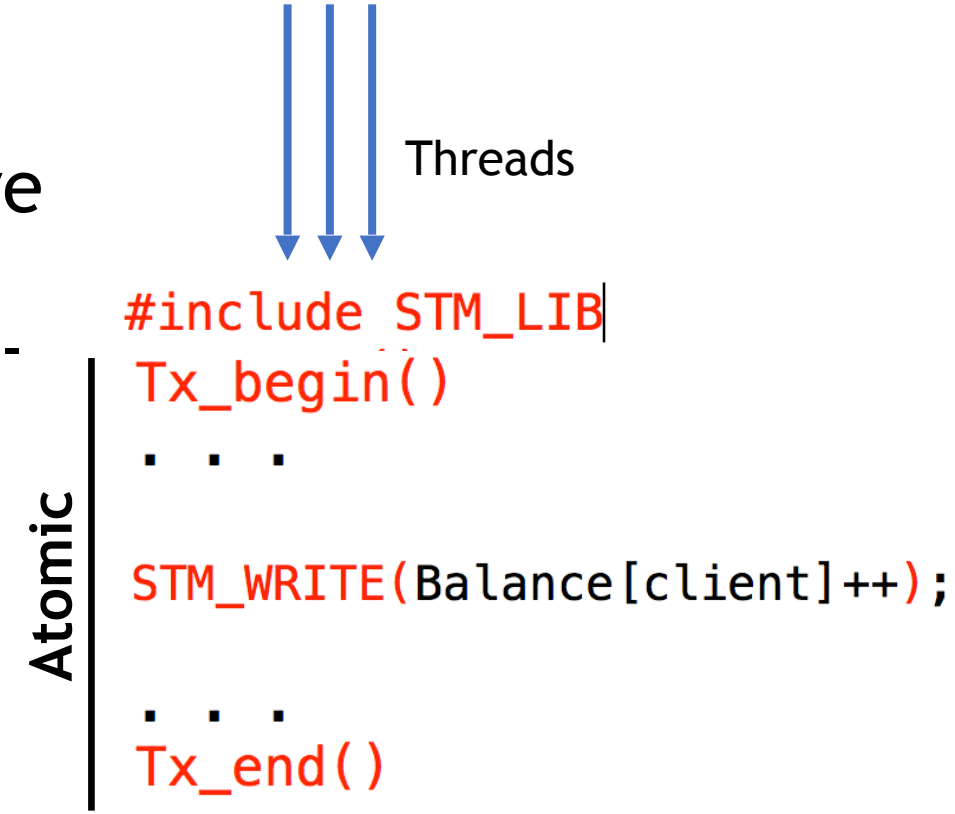
Background of TM

- Transactional Memory [ISCA' 93] promises:
  - Ease of use
  - performance equal or better than fine-grained locking
- Extension of multi-processors' cache coherence protocols
  - Speculative run => apply the concept of **transactions** from databases to parallel code
  - The underlying system guarantees isolation and atomicity



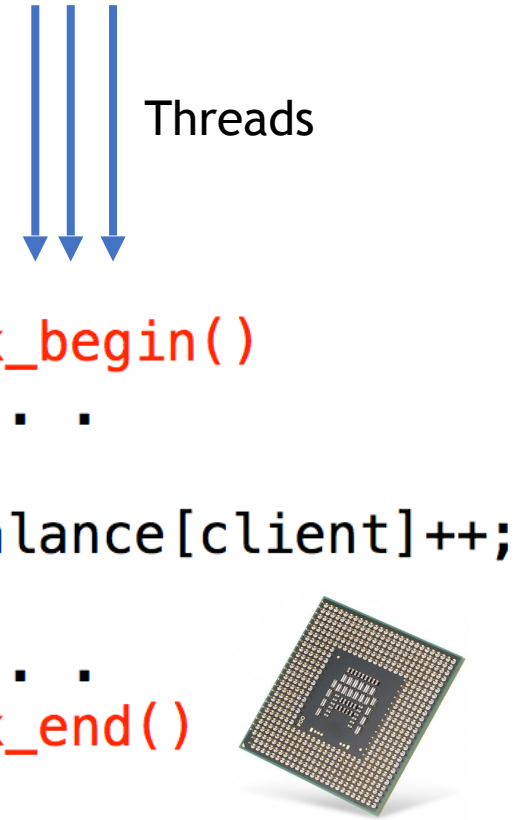
Software Transactional Memory (STM)

- Implements TM abstraction via a software library
  - ✓ No restriction on transactions' read and write-set sizes
  - ✓ Good Scalability at high thread counts
  - ✗ High instrumentation costs due to software-based tracking of reads/writes



Hardware Transactional Memory

- Best effort nature:
  - Based on extension of the cache coherence mechanisms
  - Recently introduced both in high-end (e.g. IBM Power8) and commodity (Intel Haswell) CPUs



✓ No software instrumentation over reads and writes

✓ Faster than STM for read and write-sets that fit in hardware

✗ No progress guarantees:

-Transactions can **abort** even when running solo due to, e.g., exceeding cache capacities, context switches, timer interrupts...

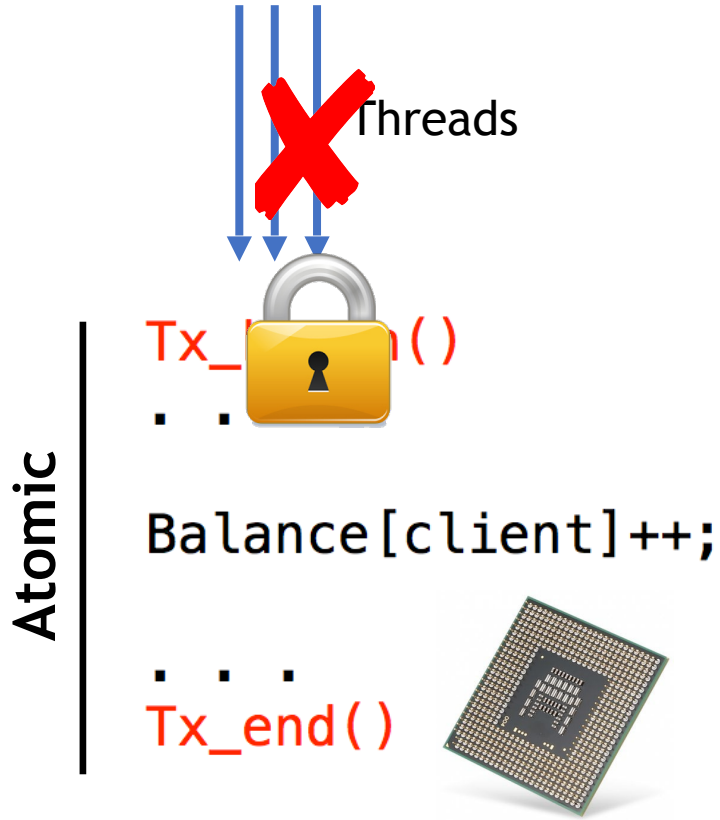
Fallback mechanism

- After a few attempts using HTM, the transaction is executed using a software synchronization mechanism:

✓ Single global lock

✓ Progress guarantee

✗ No parallelism (abort concurrent HW transactions)

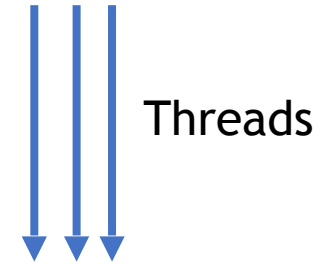


Hybrid Transactional Memory

- Fallback to STM => Hybrid Transactional Memory (Hybrid TM)

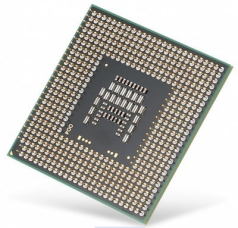
✓ preserve concurrency when activating the fallback path

✗ **Coordinating HTM and STM is not trivial**



Atomic

```
Tx_begin()
. . .

Balance[client]++;

. . .
Tx_end()
```



#include STM LIB

Problem

- State of the art Hybrid TM is **not competitive** with HTM and STM [PACT' 14]

Exceed HTM capacity by requiring HTM to store the information of addresses accessed by STM [SPAA'11]

Spurious aborts of non-conflicting HTM transactions whenever an update STM transaction commits [ASPLOS'11]

DMP in a nutshell

- Dynamic Memory Partitioning (DMP) relies on memory protection OS mechanism to enforce correctness between HTM and STM

Conventional HyTM design

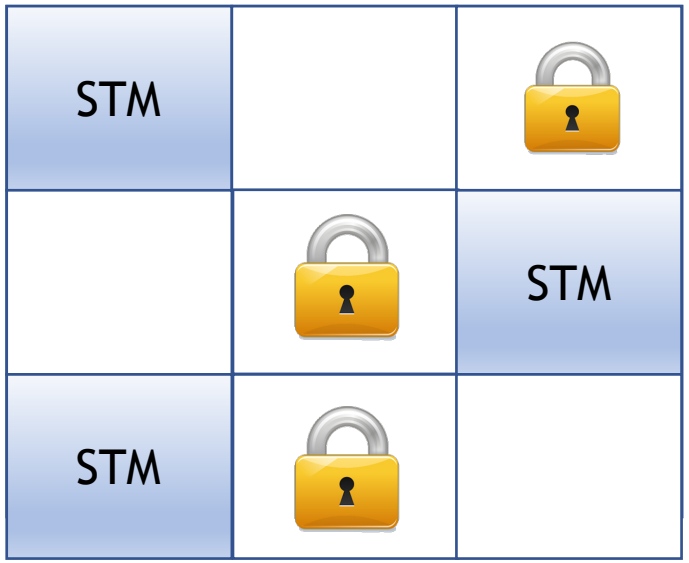
- **High** overhead in absence of conflict:
  1. Expensive instrumentation of HTM path, or
  2. Reliance on non-scalable STM
- Relatively **low** cost incurred upon contention

DMP

- Assume **low** frequency of contention between HTM and STM
- High efficiency in absence of contention between HTM & STM:
  1. STM agnostic: can use scalable OREC-based schemes
  2. No instrumentation of HTM
- Relatively **higher** overhead in absence of conflict

Key-property

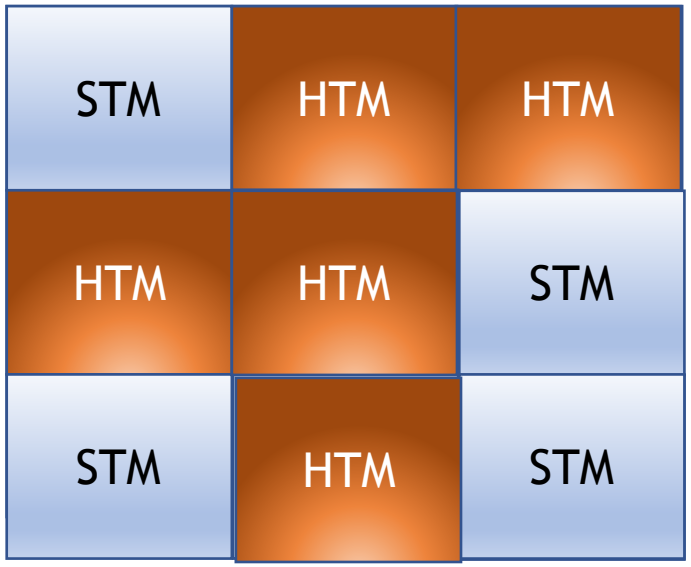
- Several reference benchmarks present ***partitionability*** in their memory accesses [*SPAA' 08]*, i.e. transactions that access one partition are unlikely to access other partitions.



Use multiple concurrency control mechanisms/STMs

Key-property

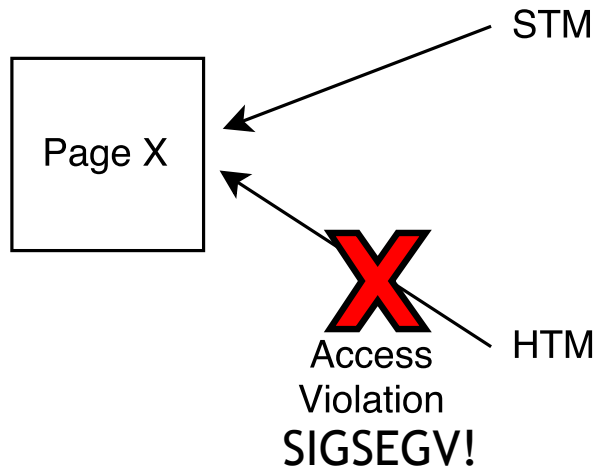
- DMP-TM is designed to take advantage of this property and minimize contention between HTM and STM
- Each partition fits the characteristics of HTM and STM, respectively



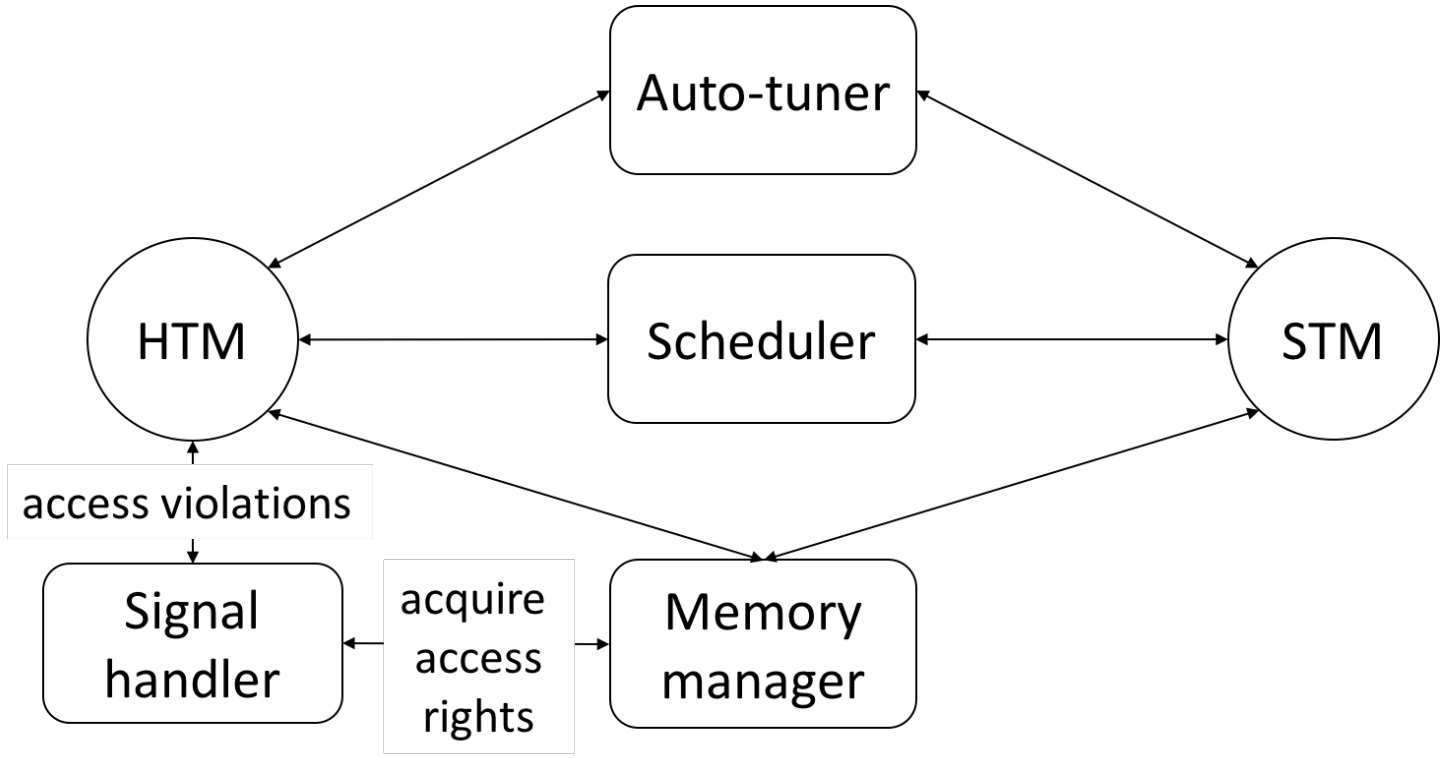
How??

DMP in a nutshell

- Using Operating system's memory protection mechanism
  - Is possible to revoke the access to certain pages to either HTM or STM
  - Depending on the access performed (read or write)
  - Conflicts are detected at page level by catching a Segmentation Fault (SIGSEGV) signal

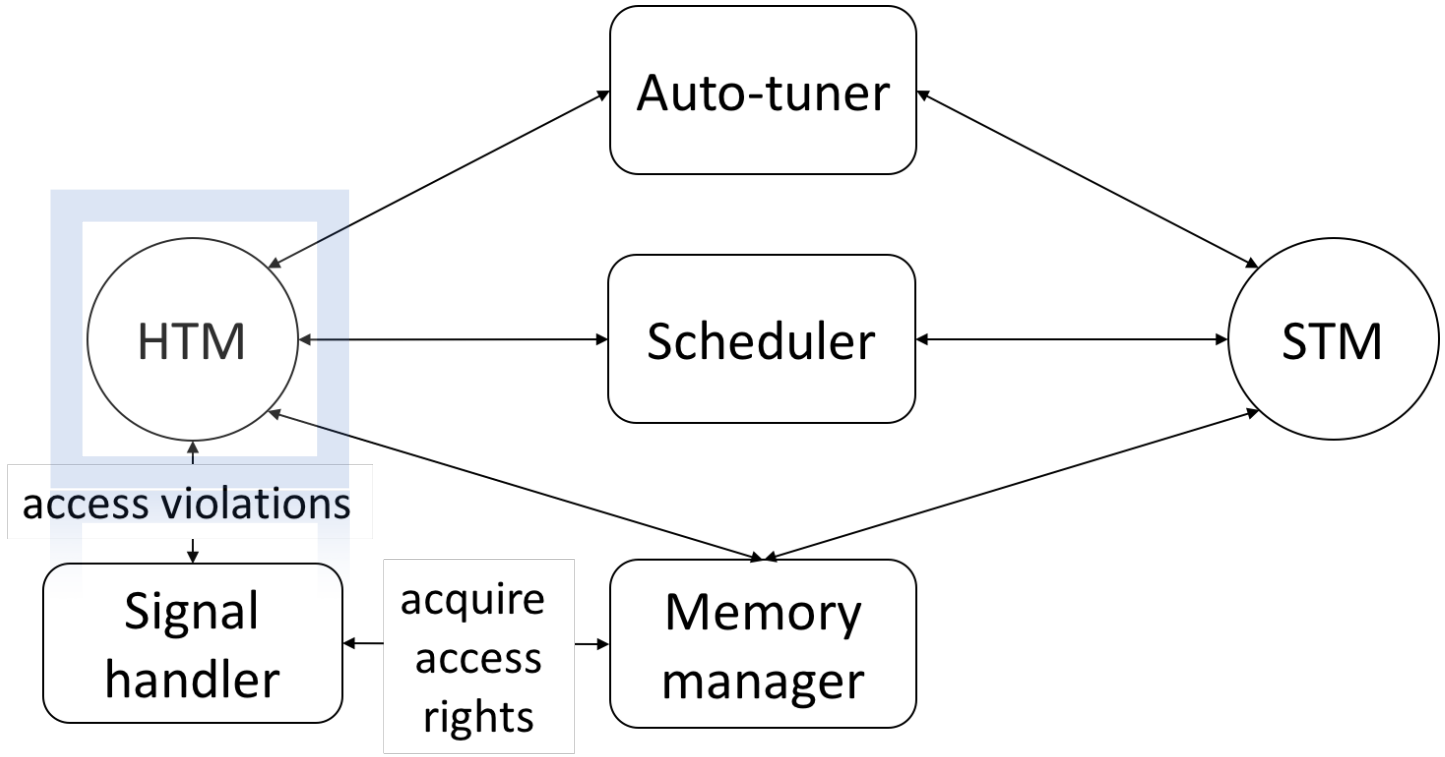


DMP - Architecture



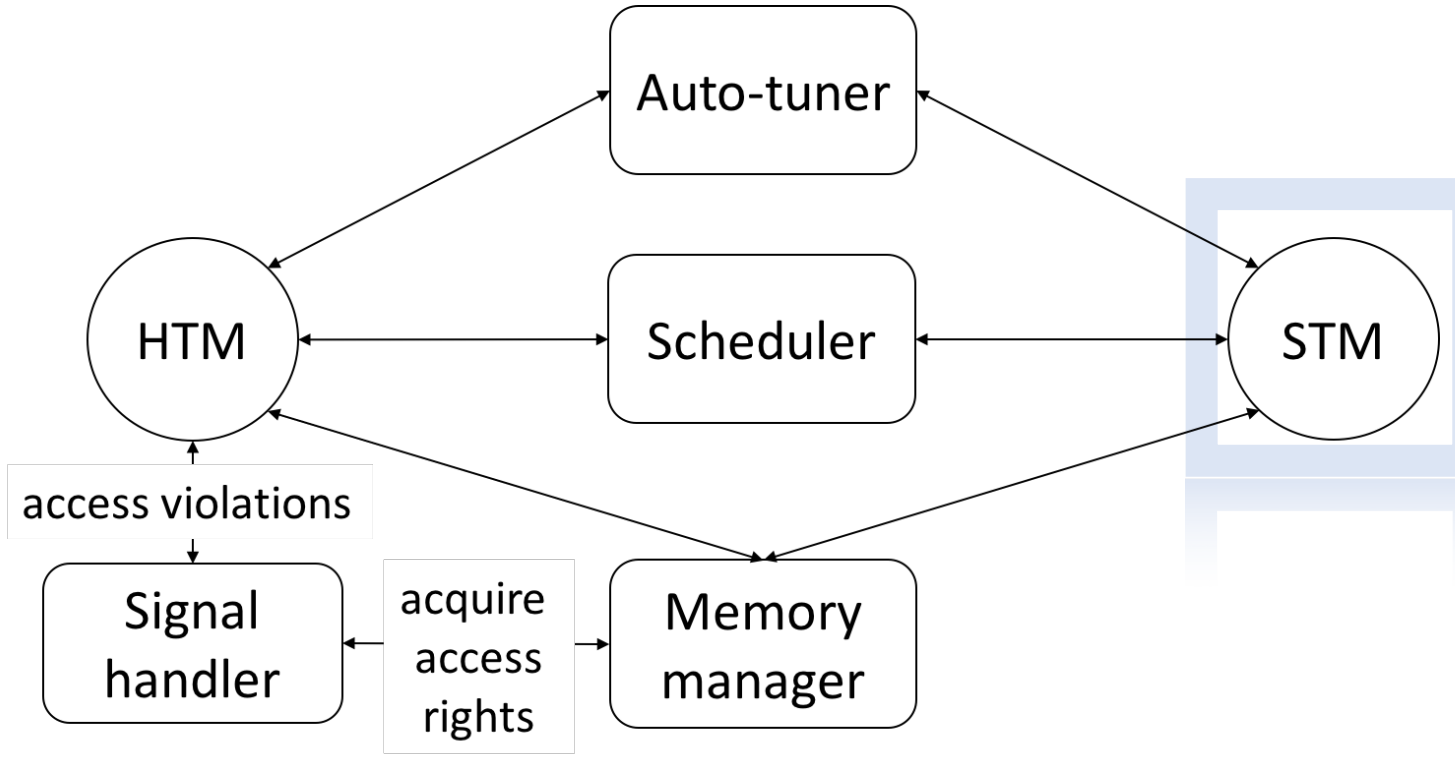
DMP - HTM

- No instrumentation (reduces probability of exceeding cache capacity)

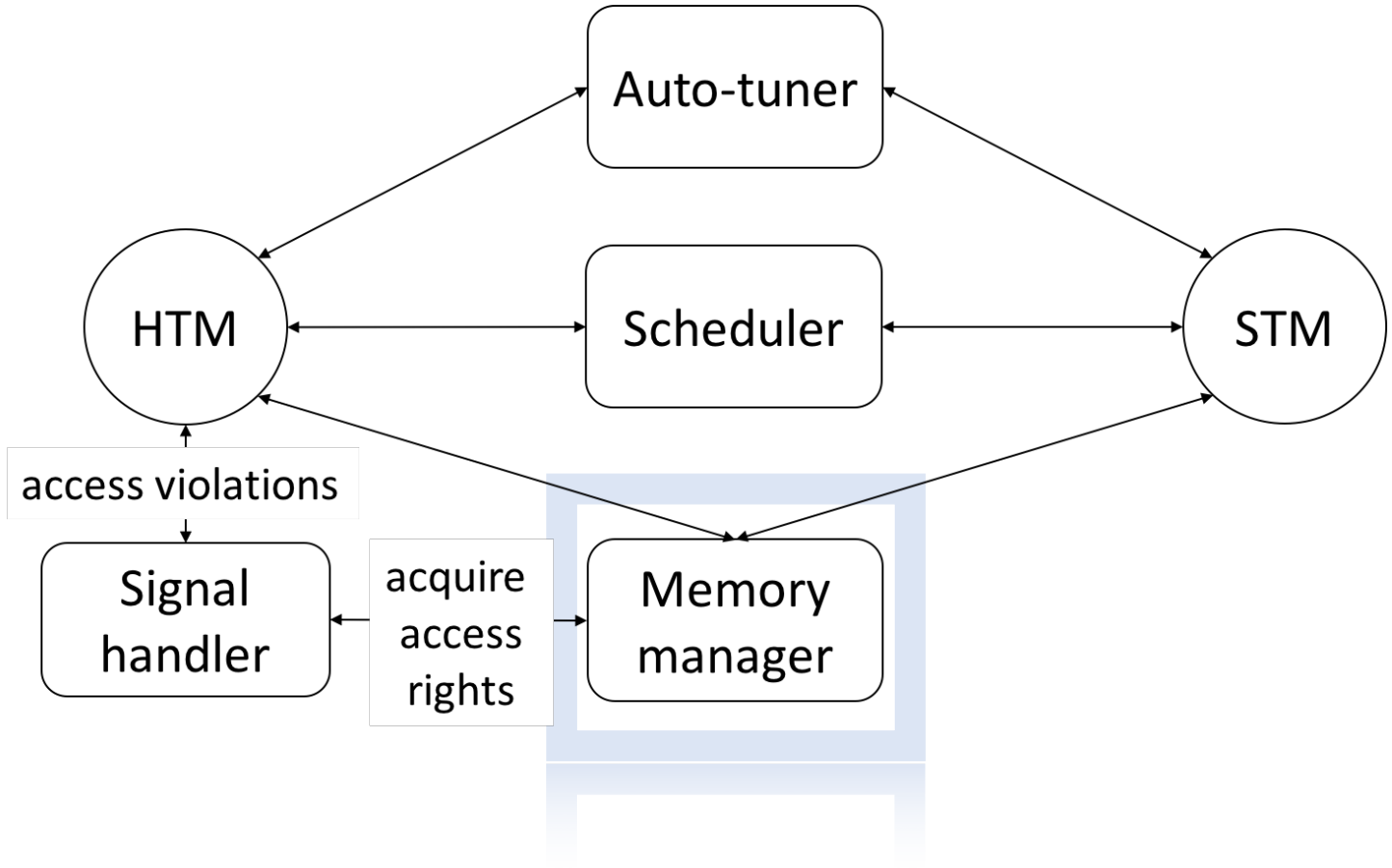


DMP - STM

- **TinySTM** [PPoPP '08]
- *Very robust across heterogeneous workloads*
- Scalable *Orec-based* STM
- Added instrumentation to synchronise with HTM

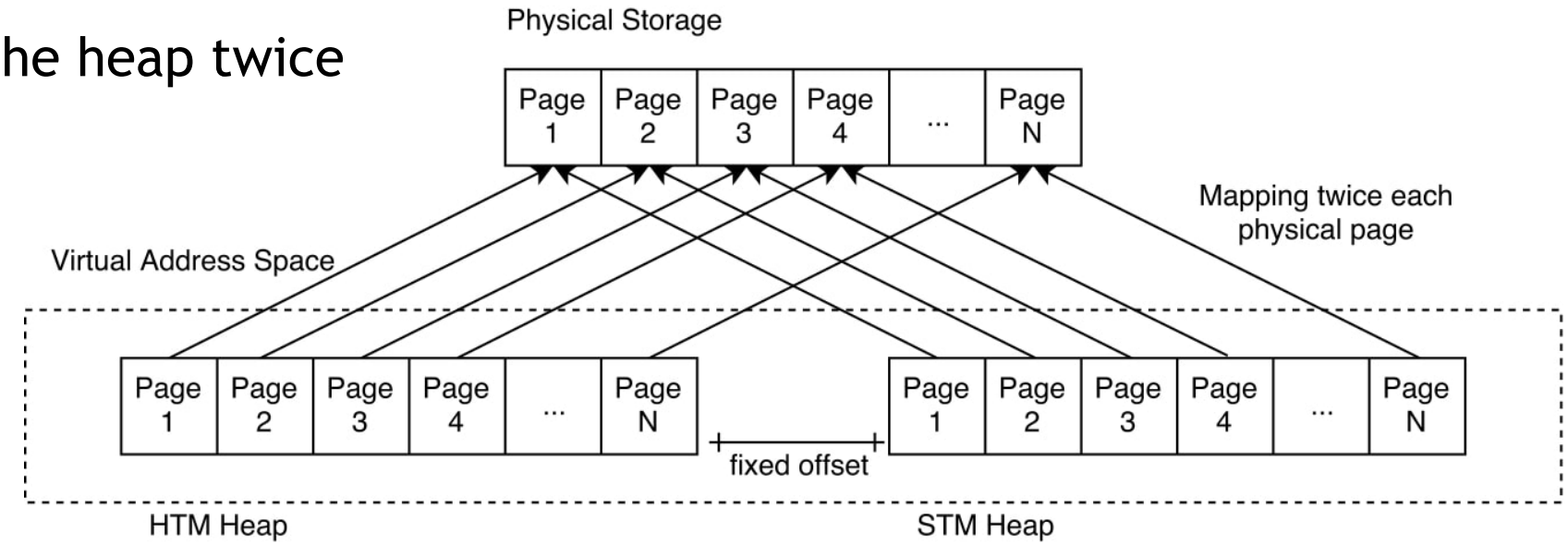


DMP - Memory Manager

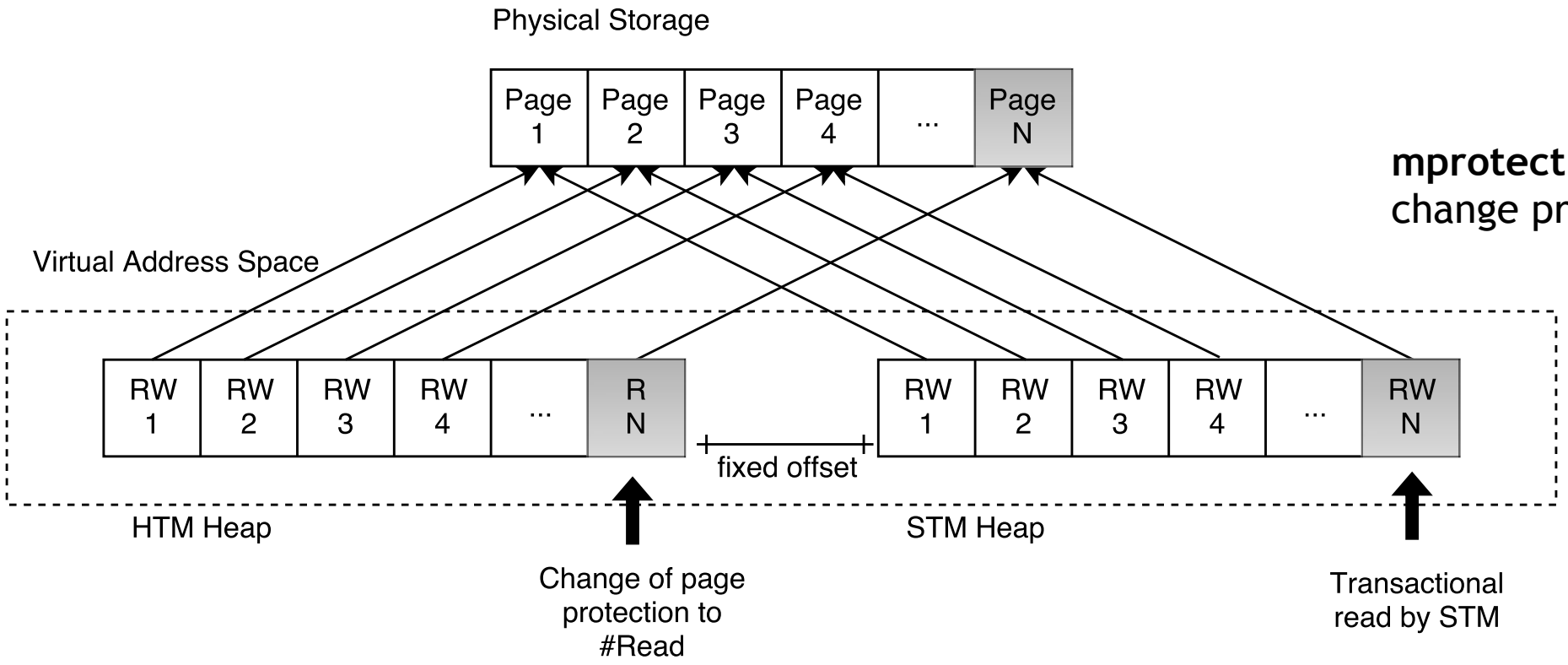


DMP - Memory Manager

Responsible for mapping the heap twice
mmap() system call

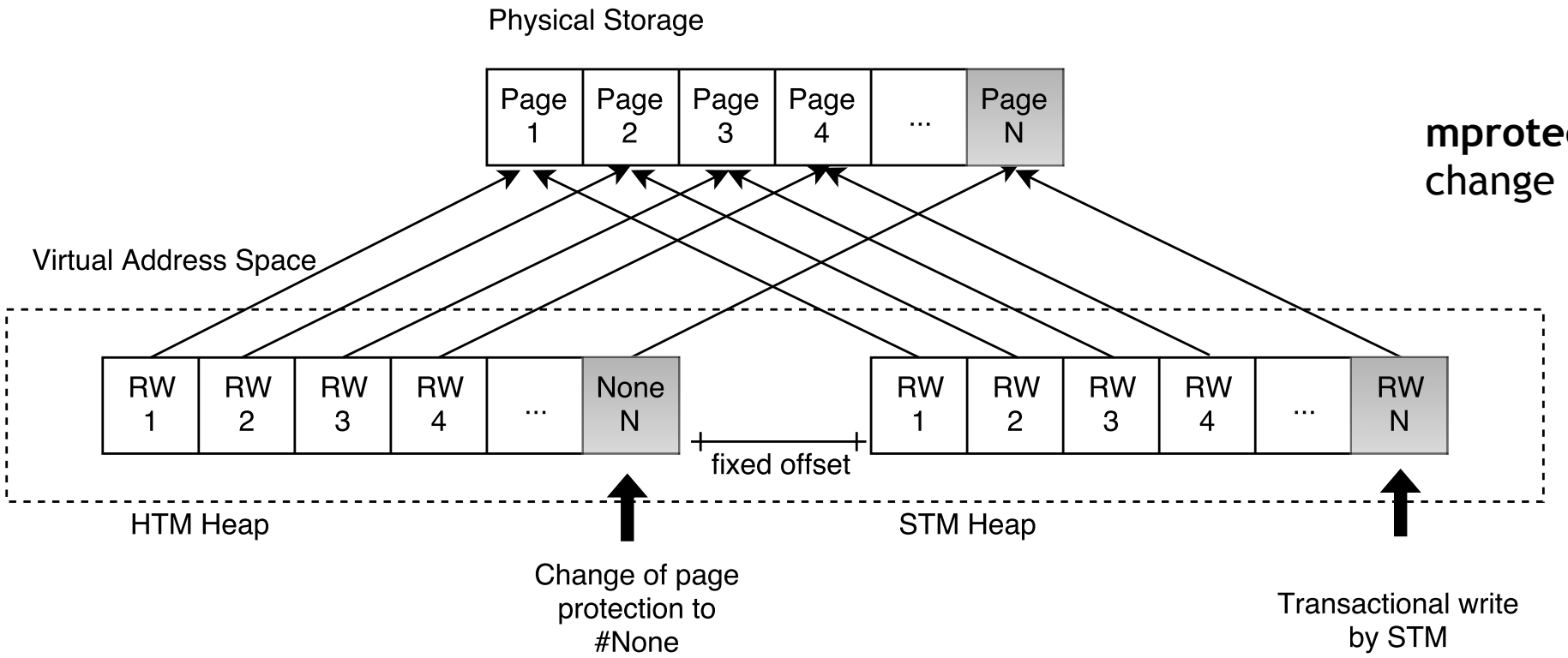


DMP - Read operations



mprotect() system call to change protection of page

DMP - Write Operations

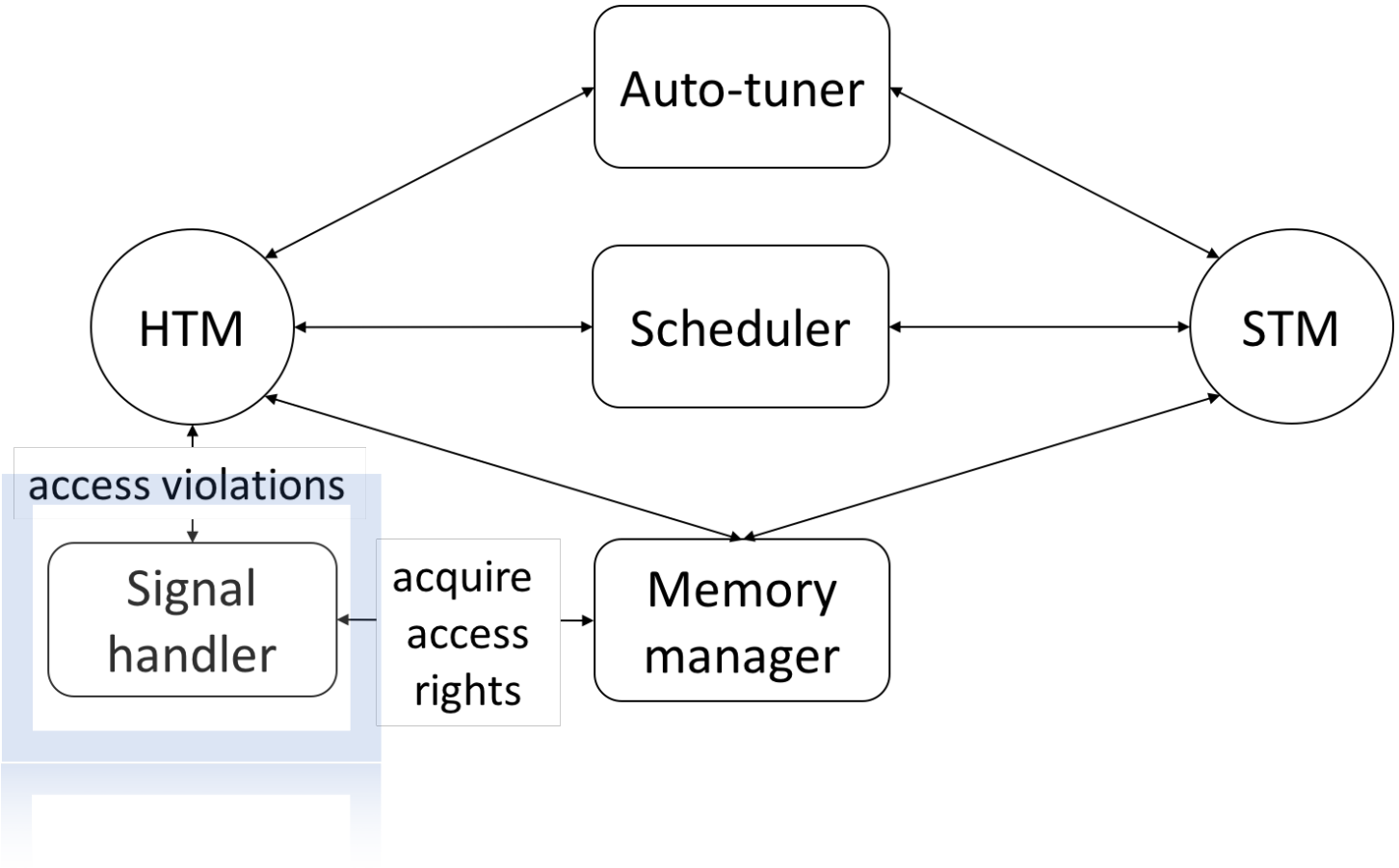


mprotect() system call to change protection of page

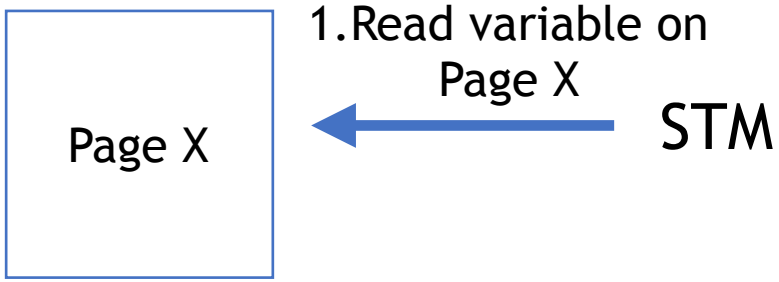
DMP - Unix Handler

Signal Handler

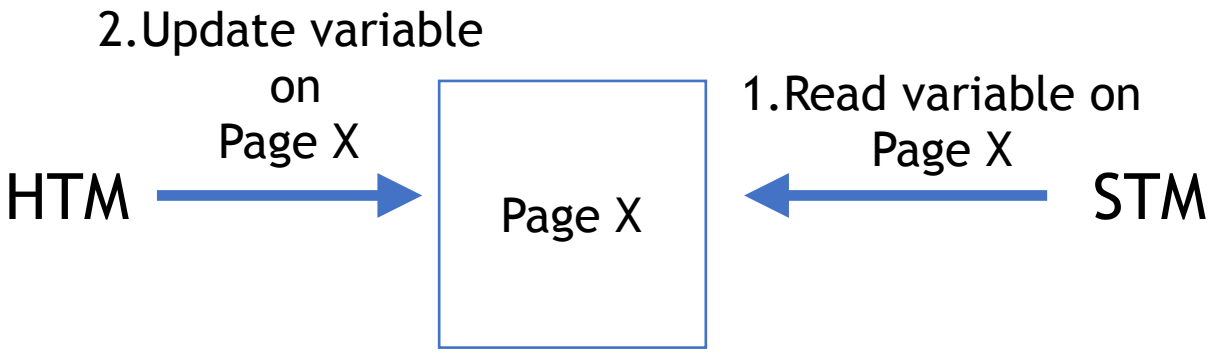
- HTM receives a SIGSEGV signal whenever it tries to access a page which will cause a conflict
- Handler is responsible for either wait until the completion of STM or change page protection



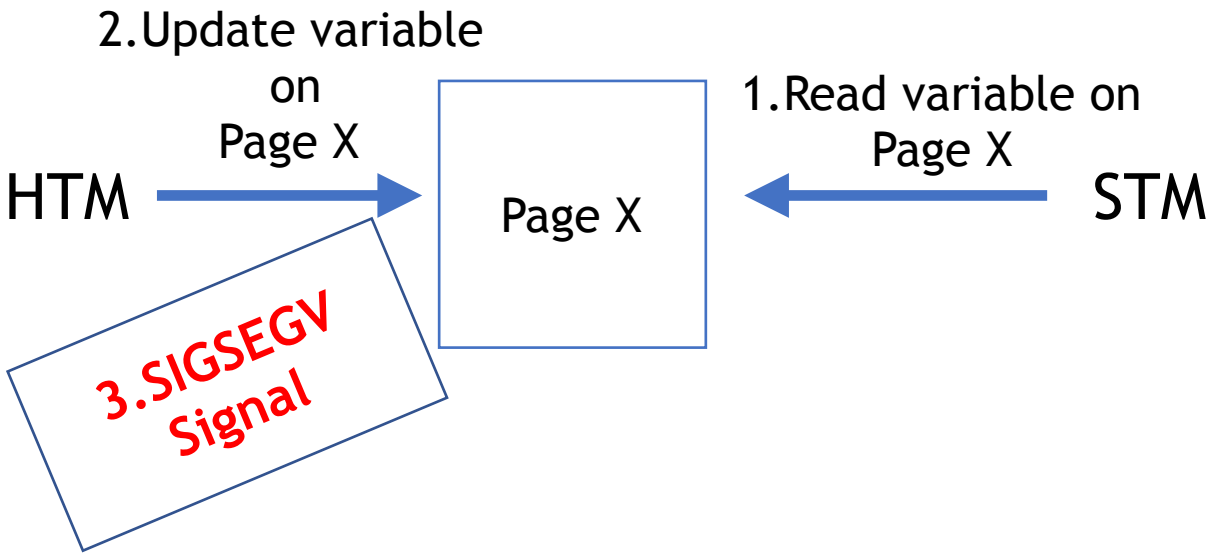
Problems - STM reads inconsistent values



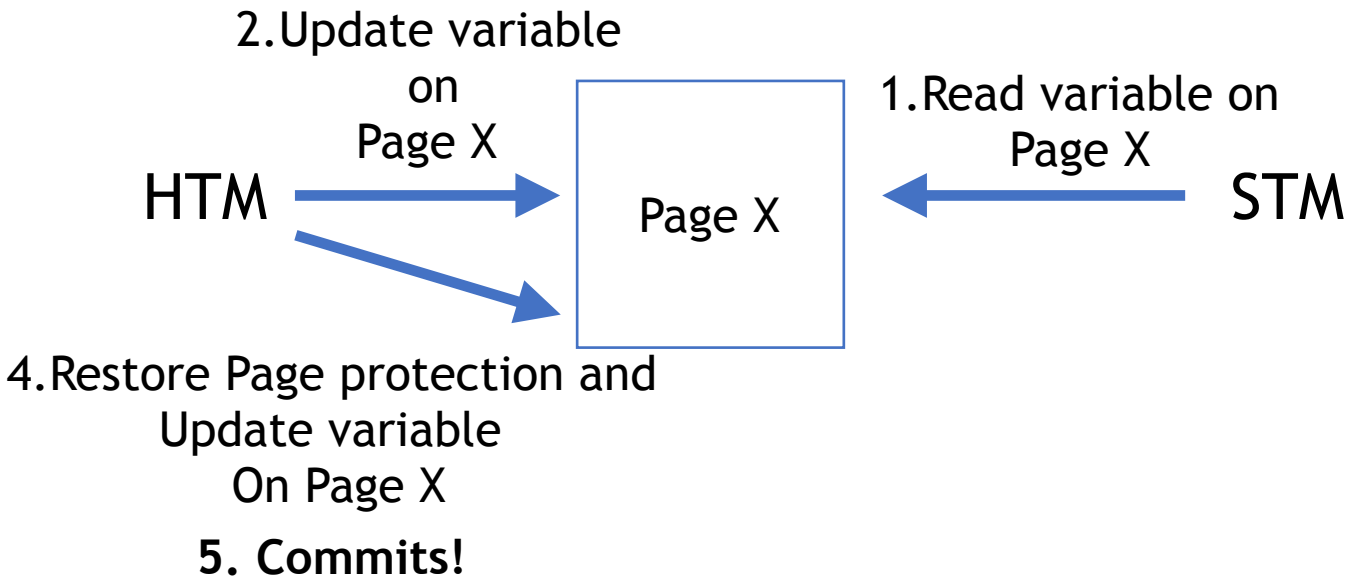
Problems - STM reads inconsistent values



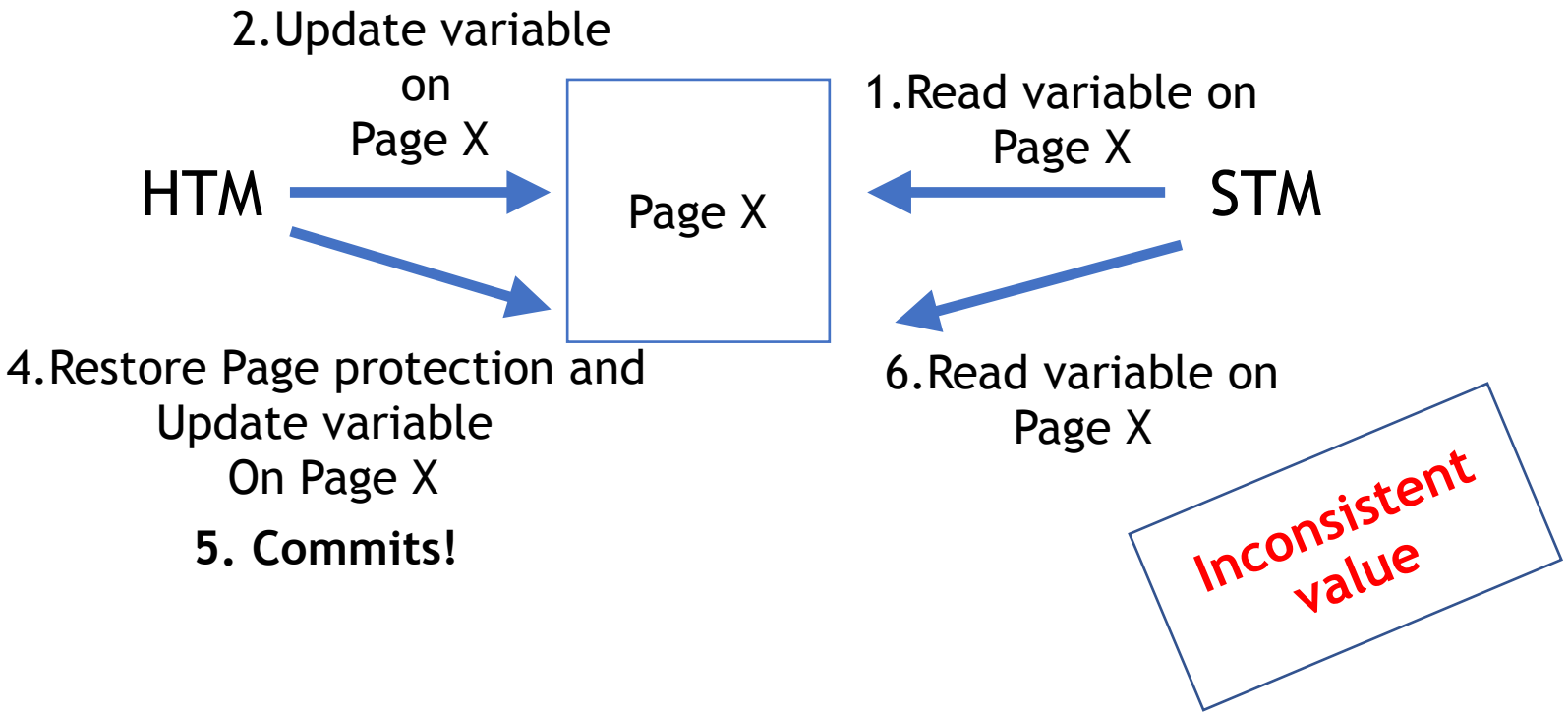
Problems - STM reads inconsistent values



Problems - STM reads inconsistent values

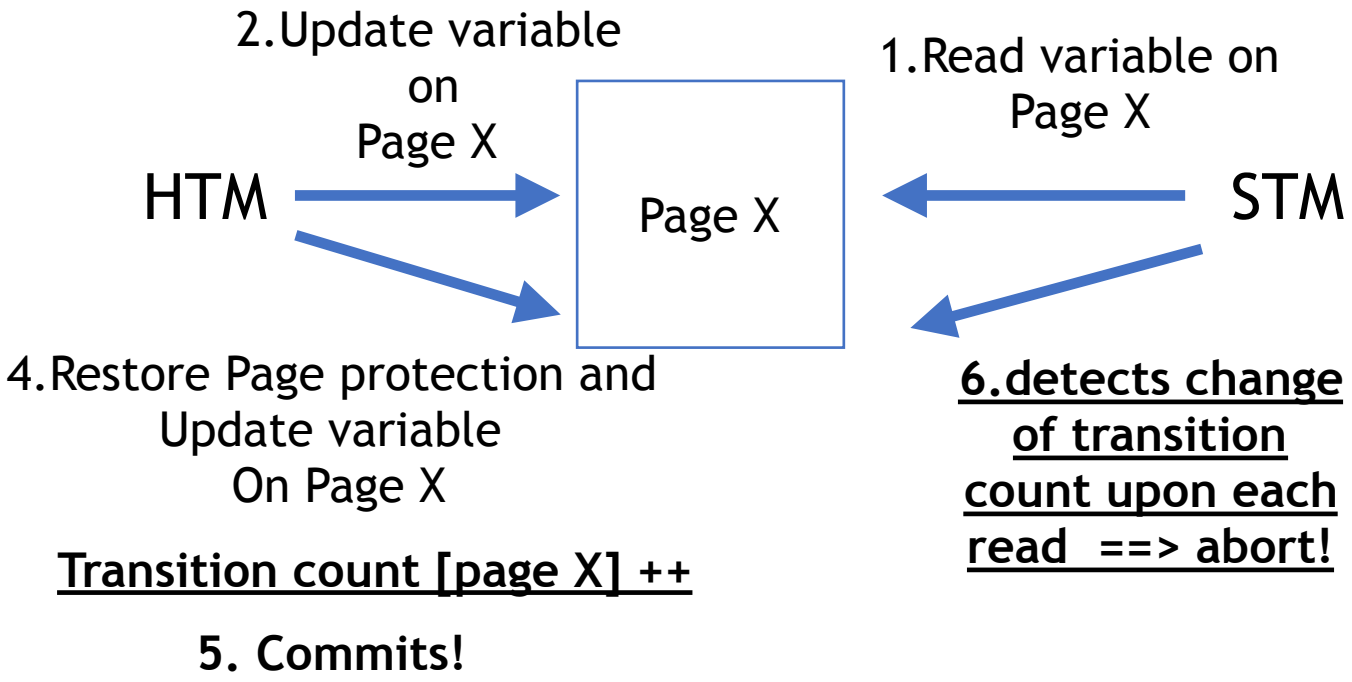


Problems - STM reads inconsistent values

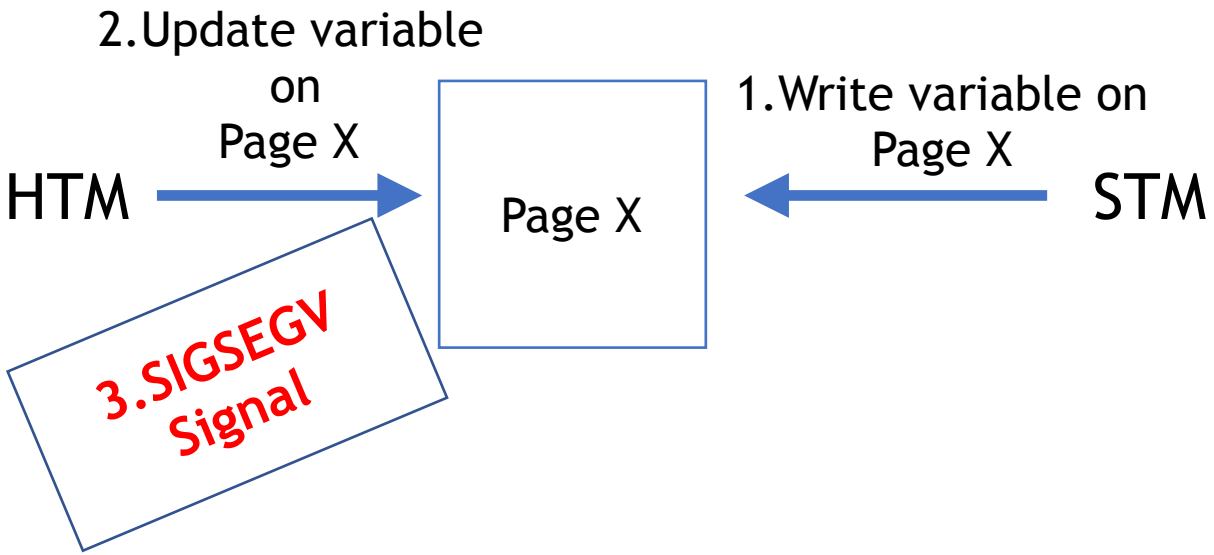


Problems - STM reads inconsistent values

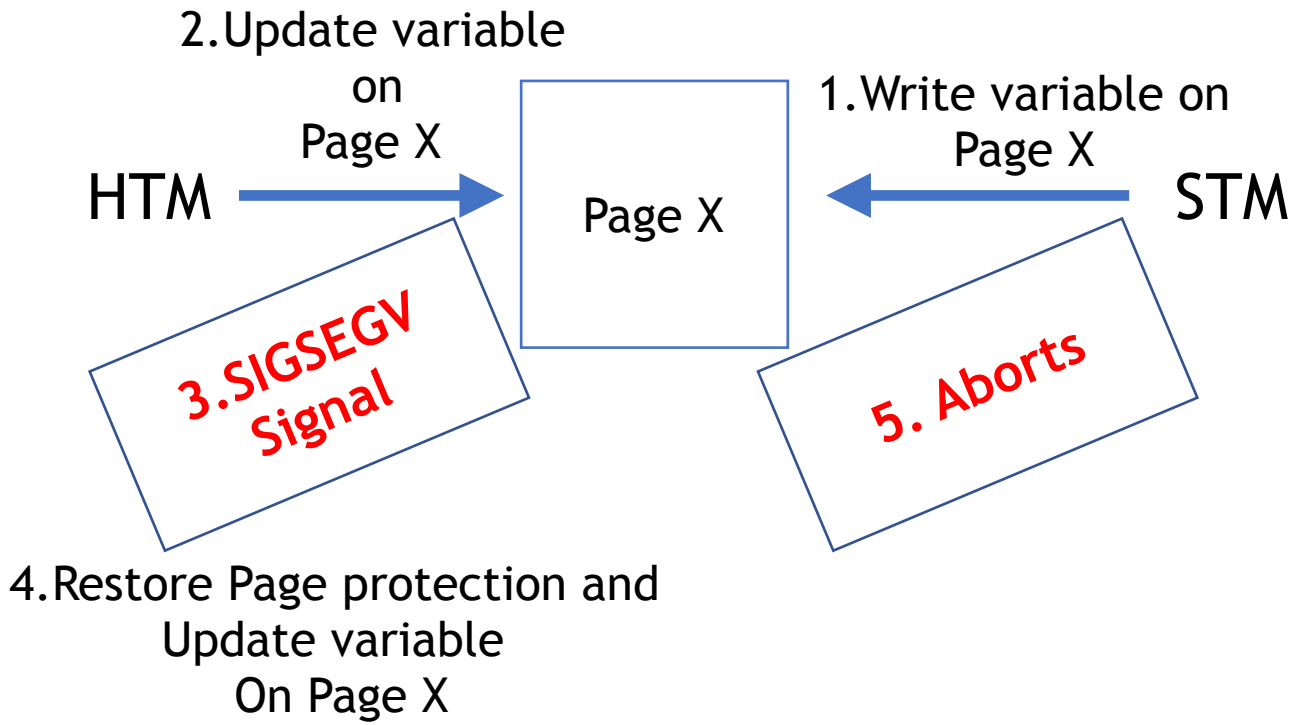
- **Solution:** Use per-page variable ***transition count***:



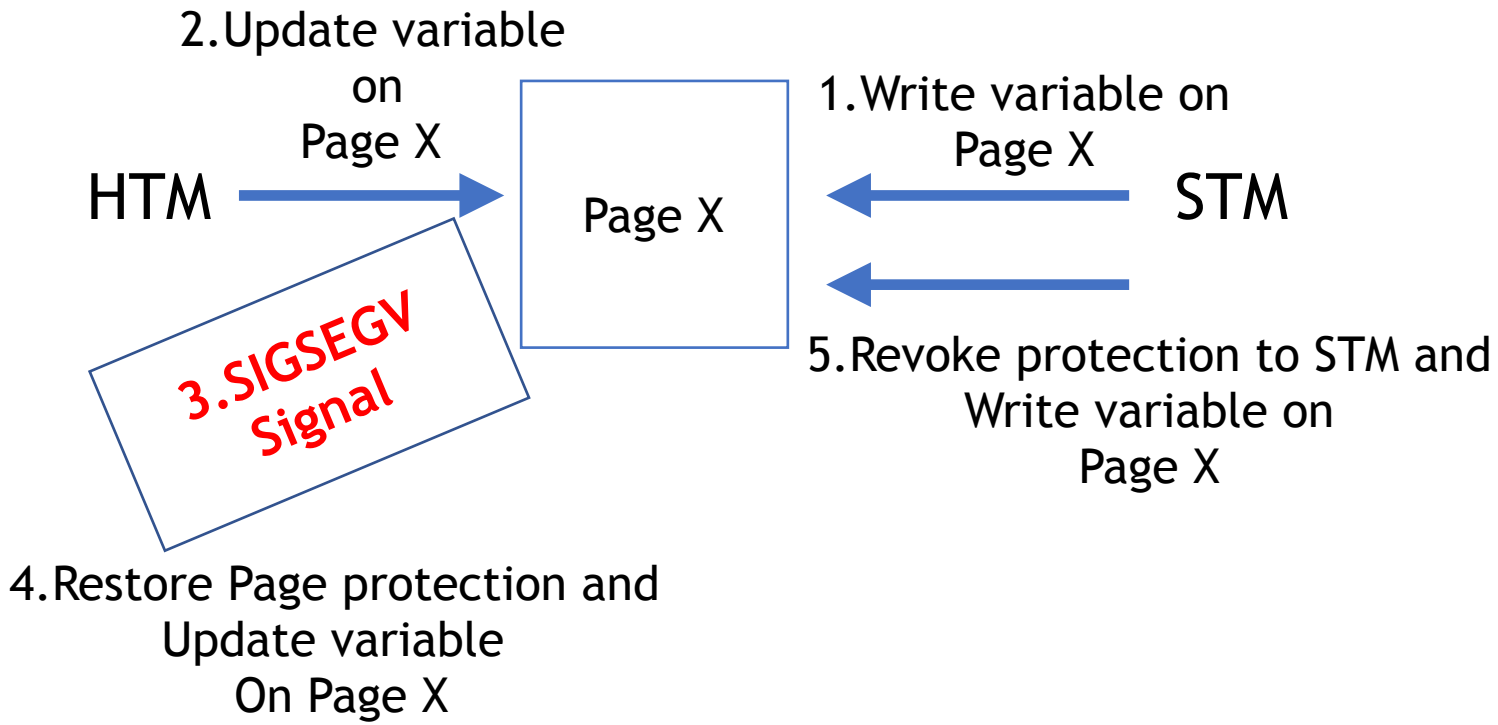
Problems - Unnecessary ping-pongs



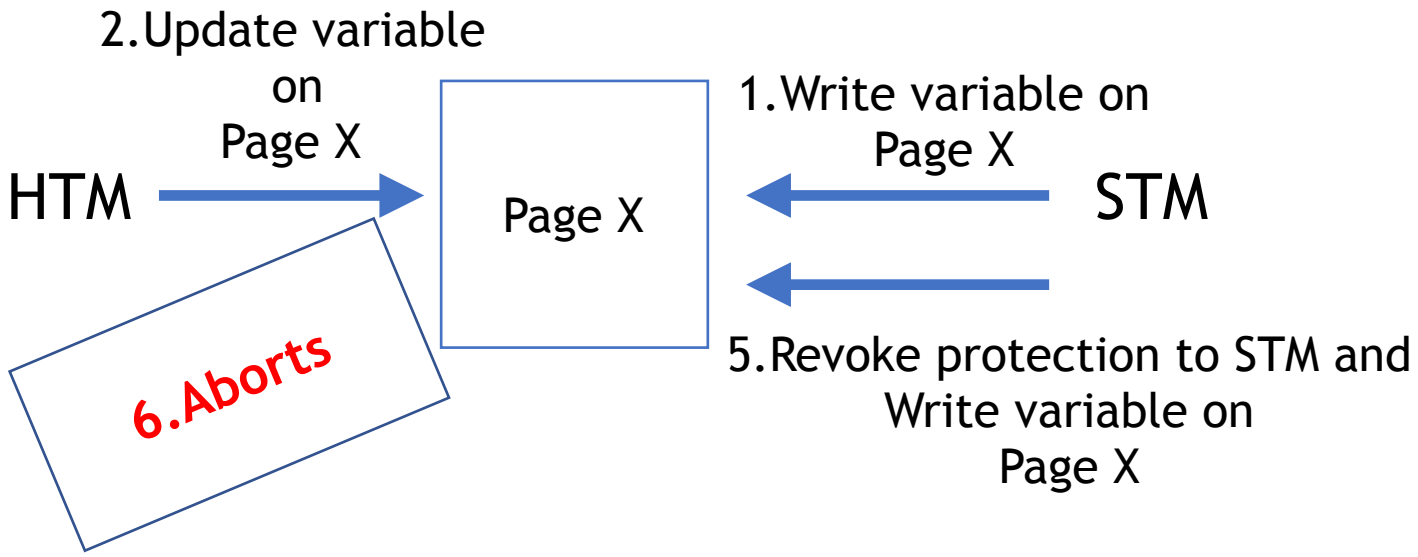
Problems - Unnecessary ping-pongs



Problems - Unnecessary ping-pongs

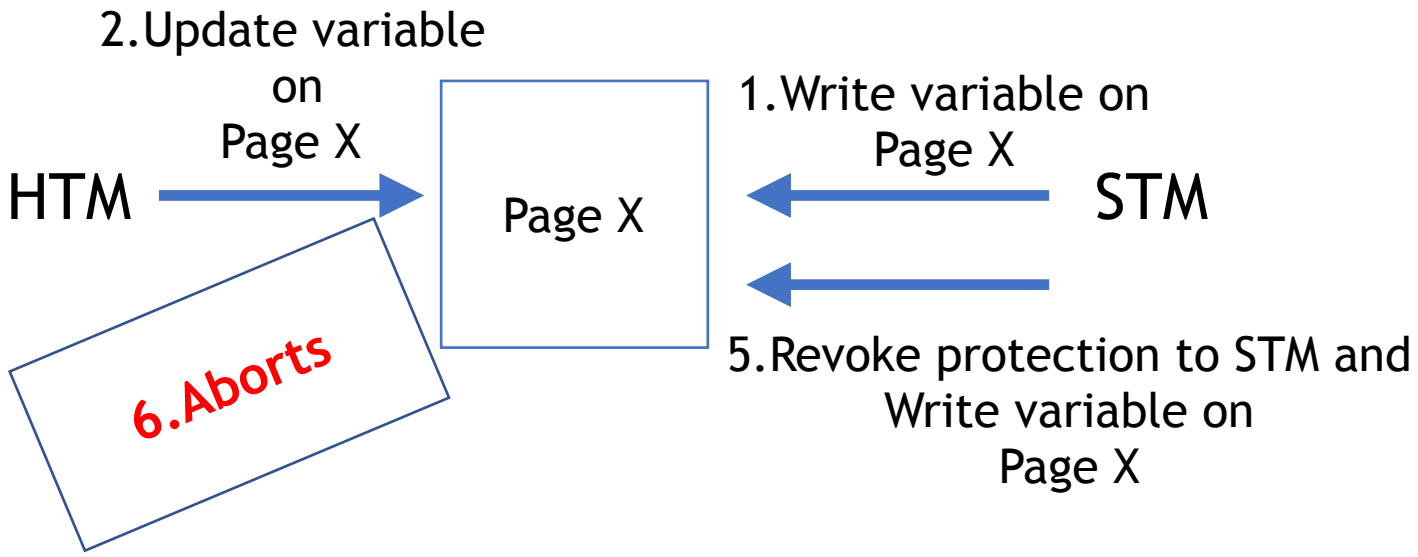


Problems - Unnecessary ping-pongs

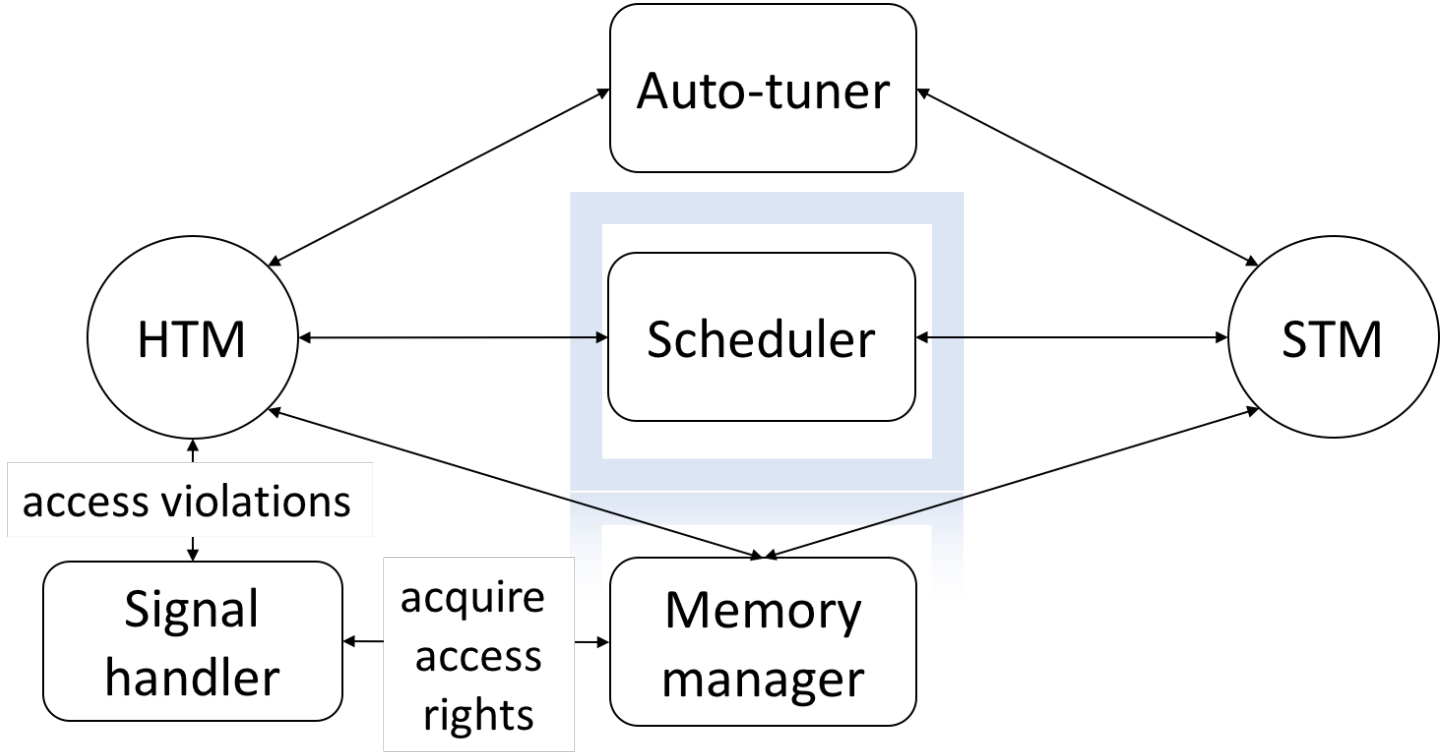


Problems - Unnecessary ping-pongs

- **Solution:** Use per-page variable ***writer count***:
  Handler only restores when no STM writers active on the page



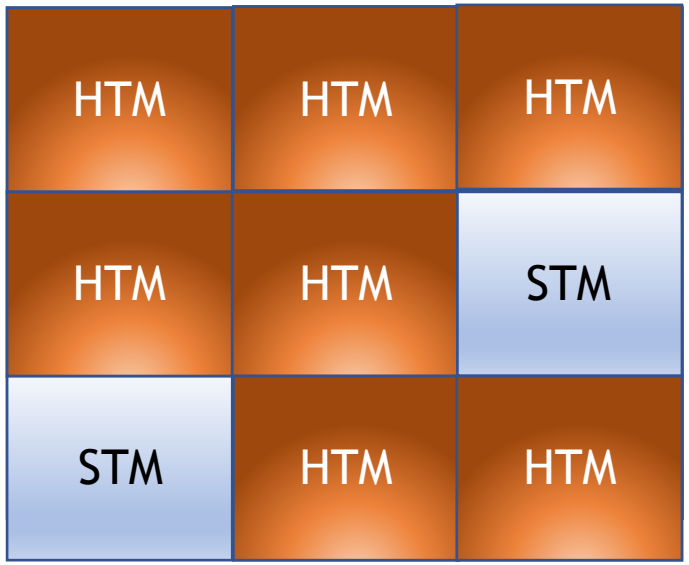
DMP - Scheduler



DMP - Scheduler

Scheduler automatically classifies transactions:

- **HTM friendly** (< 5% of Capacity aborts)
- **HTM non-friendly** (Capacity aborts)

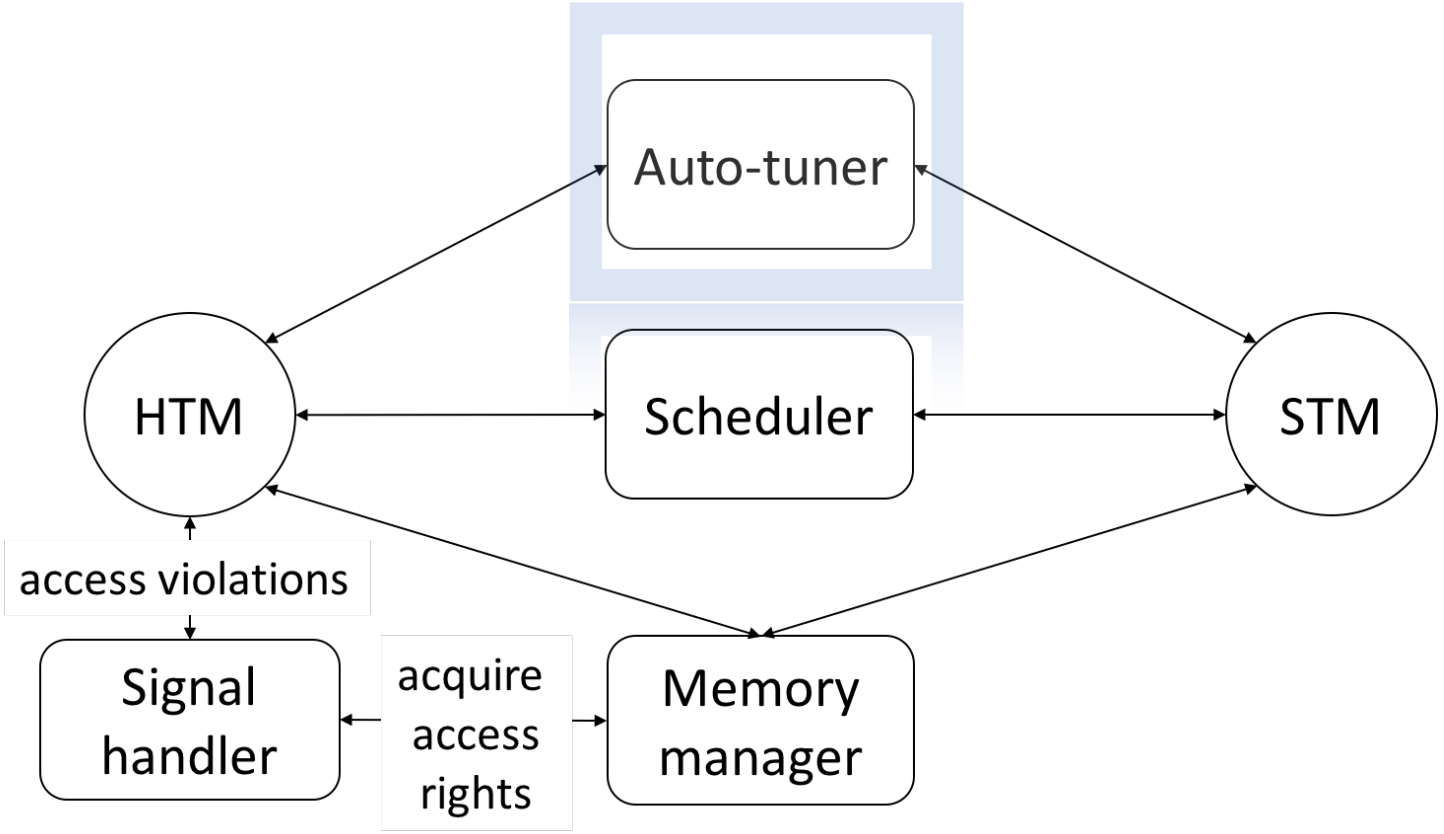


DMP - Auto-tuner

Automatically detect non-partitionable workloads:

- If we spend more than 20% of the time issuing sys calls:

- Fallback to one of the backends



Evaluation

- Microbenchmarks
  - **Best-case Scenario:** Completely disjoint set of pages, both incurring heterogenous workloads
  - **Worst-case Scenario:** Only one page
- STAMP
  - Genome
  - Intruder
- TPC-C

Evaluation

- Baselines
  - DMP
  - DMP w/Auto-Tune module
  - HyNOrec
  - HyTinySTM
  - HyTL2
  - HTM-SGL
  - NOrec
  - TinySTM

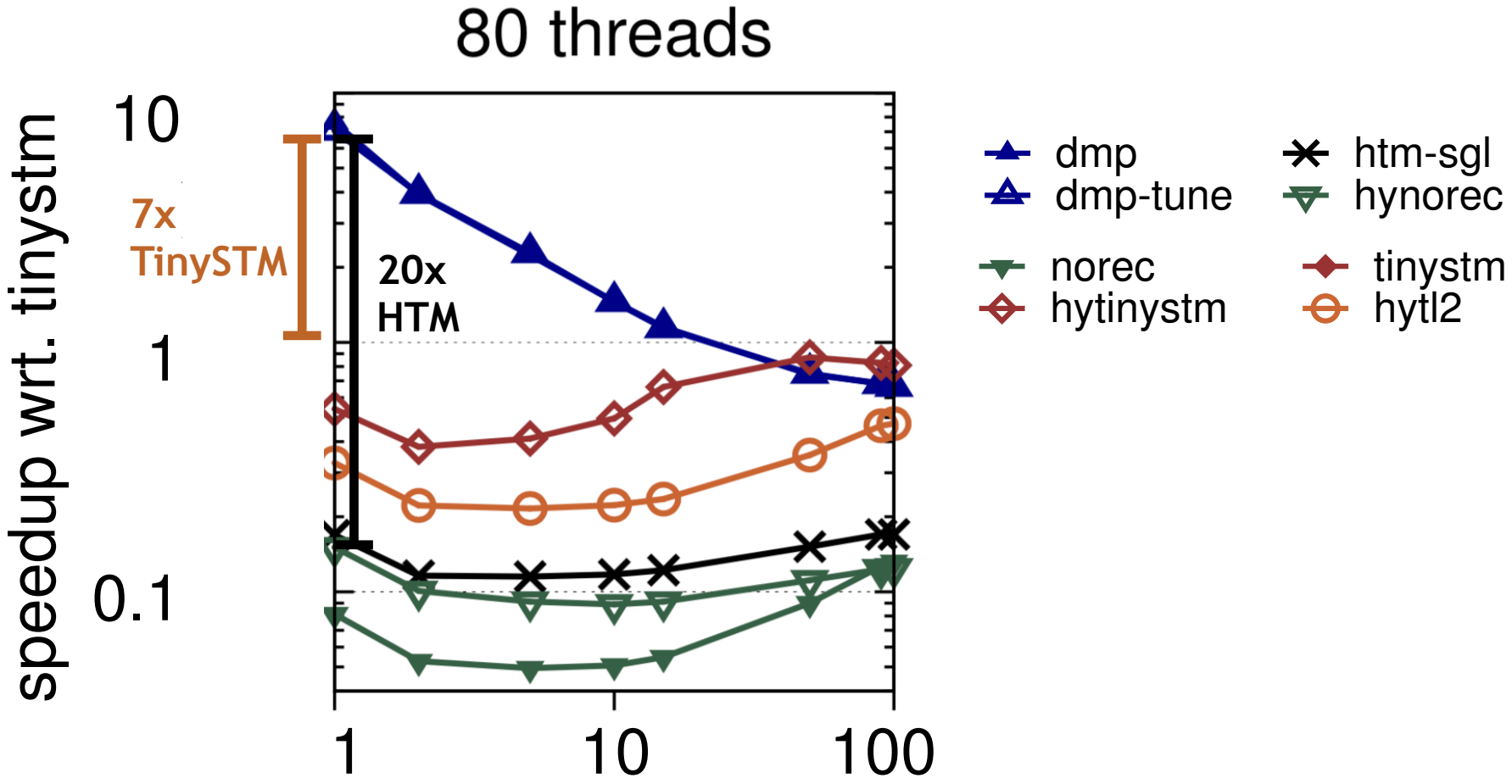
} HyTM

} STM

Evaluation

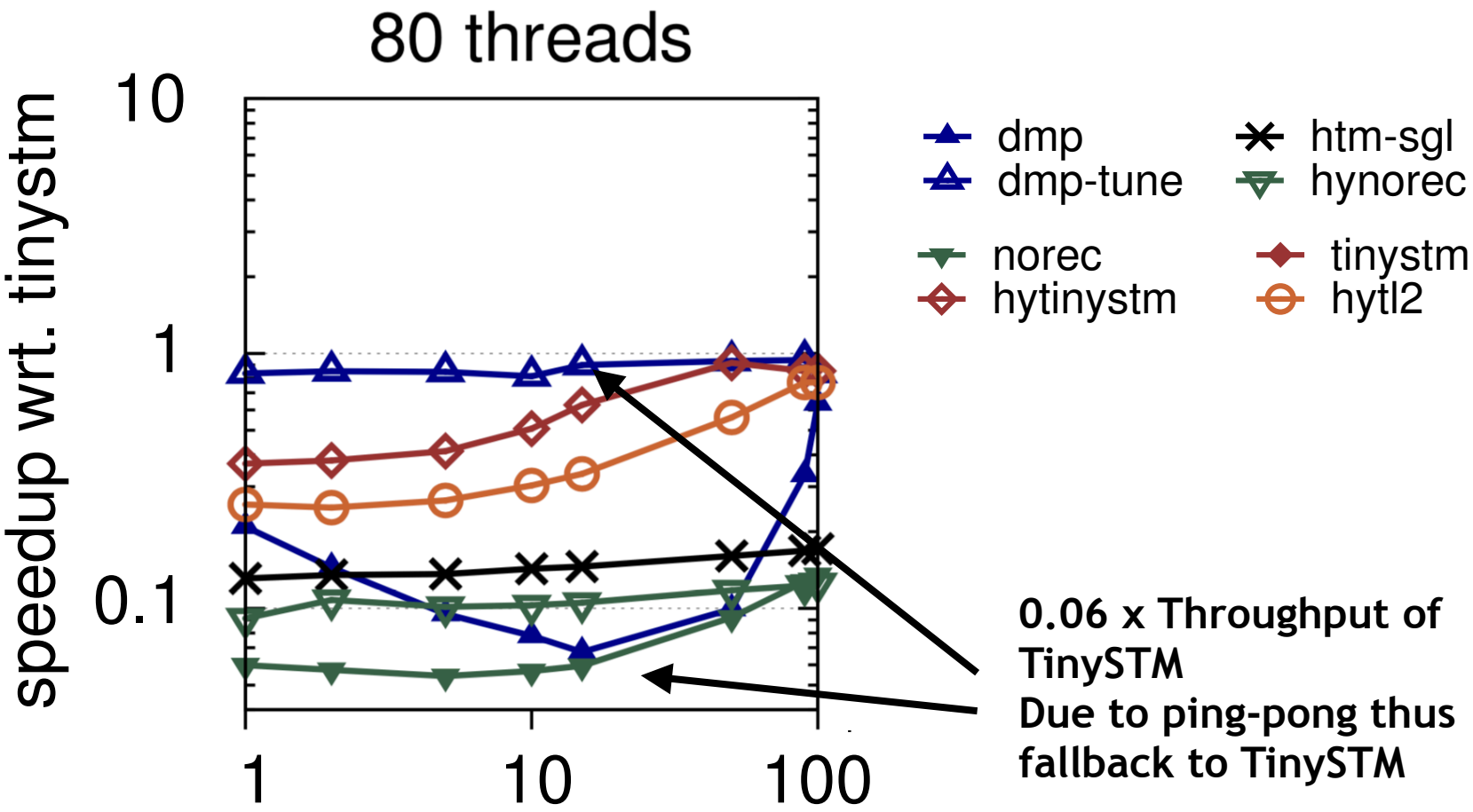
- IBM Power8
- 10 cores
- 8 Threads per core

Evaluation - Disjoint Microbenchmark

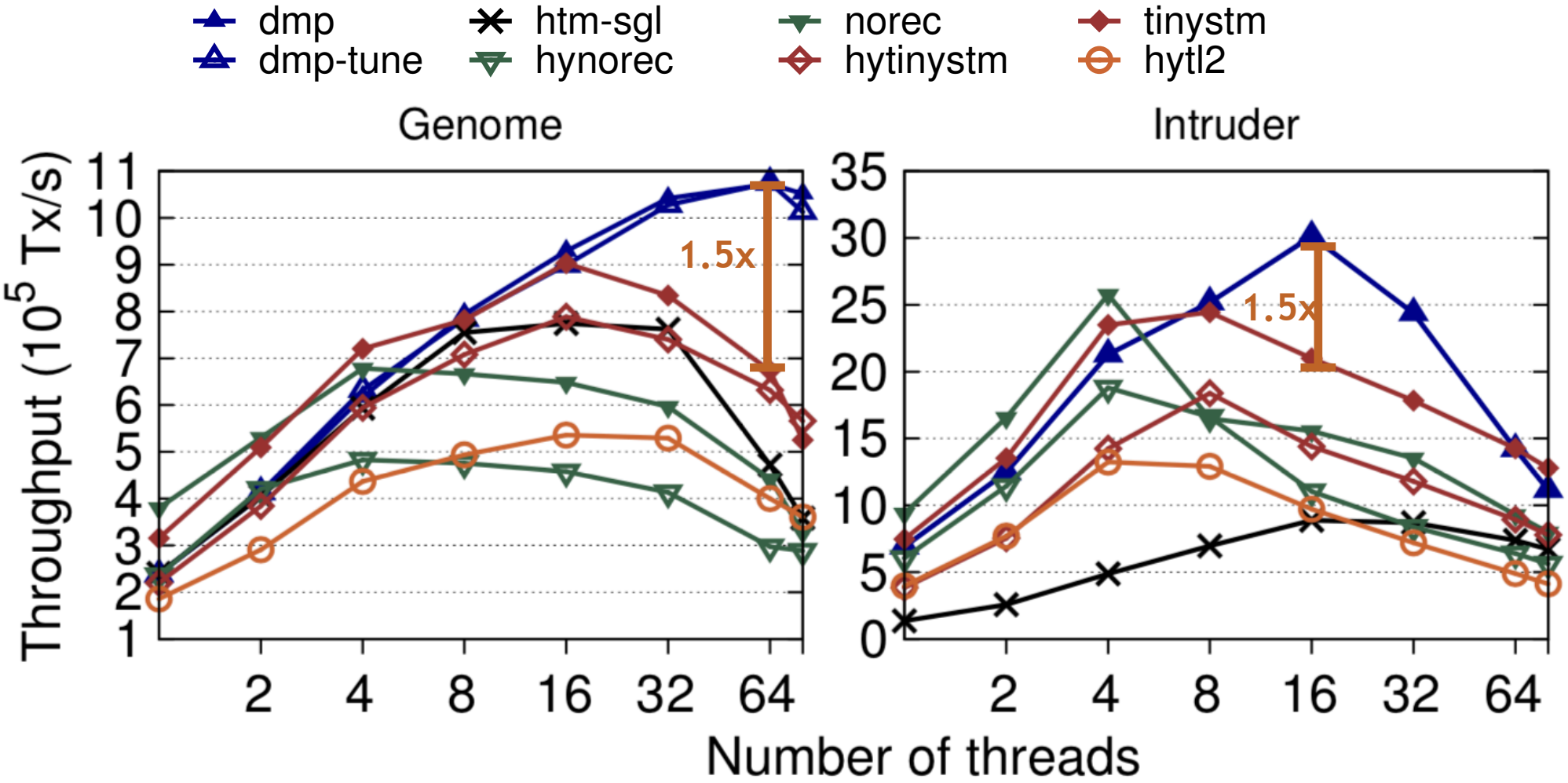


Evaluation - Non-disjoint Microbenchmark

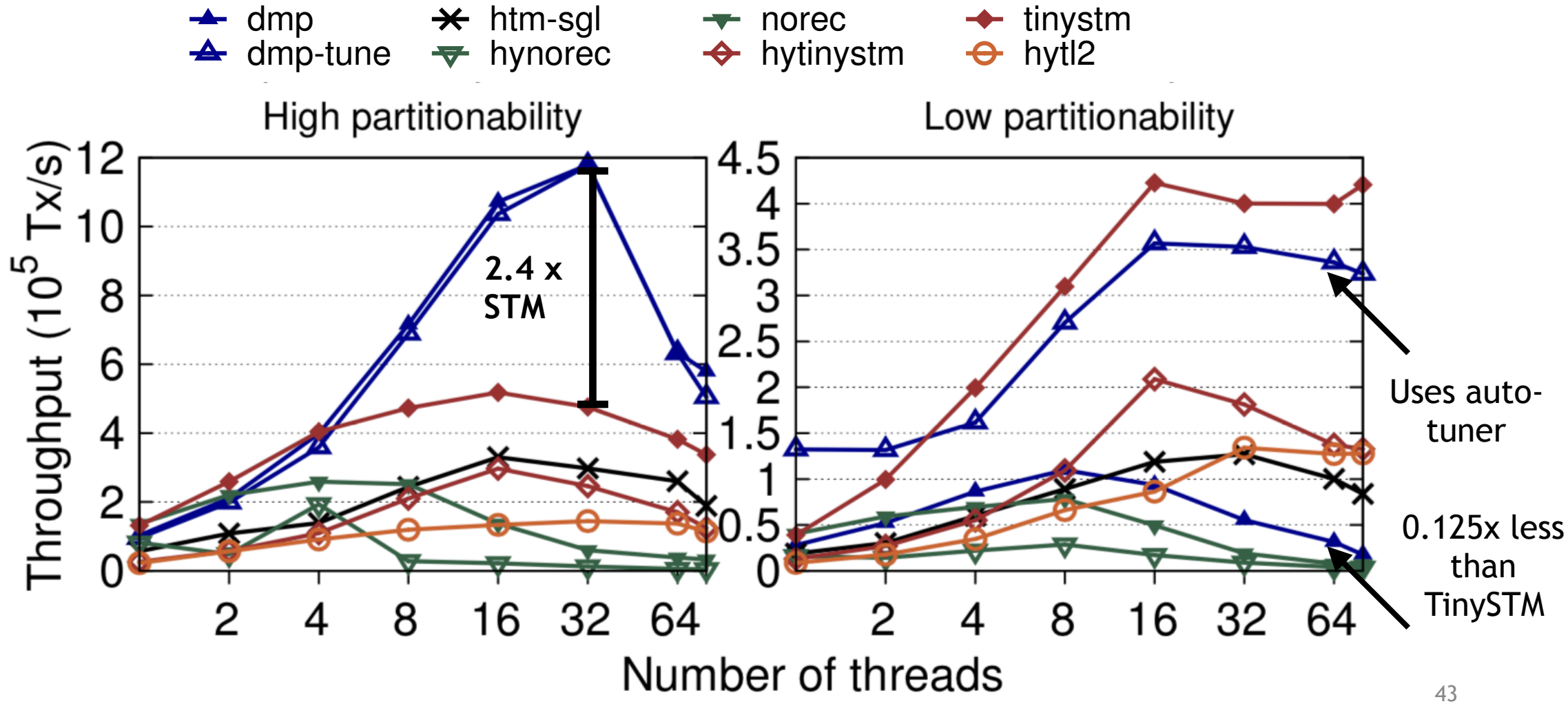
Auto-Tune allows to avoid overheads in unfavourable workloads



Evaluation - STAMP



Evaluation - TPC-C



Summary

- DMP design optimized for workloads with infrequent conflicts between transactions executing in HW and SW
- DMP uses OS memory protection mechanism to devise partitions where back-ends can execute without interference.
- Up to ~20x speedups compared to state of the art HTM, STM and HyTM

Thank you


https://github.com/pedroraminhas/DMP-TM

http://www.gsd.inesc-id.pt/~praminhas/


pedro.raminhas@tecnico.ulisboa.pt