An overview of my research

Paolo Romano

Lisbon University & INESC-ID

Roadmap

- About me
- About IST & INESC-ID
- An overview of my past research activities
- Current research lines:
  - Transactional Memory & emerging HW technologies:
    - Persistent Memory
    - GPUs
  - Leveraging Symbolic Execution for Distributed Transactional Systems
  - Parallel/distributed platforms for Machine Learning

About me: scientific career

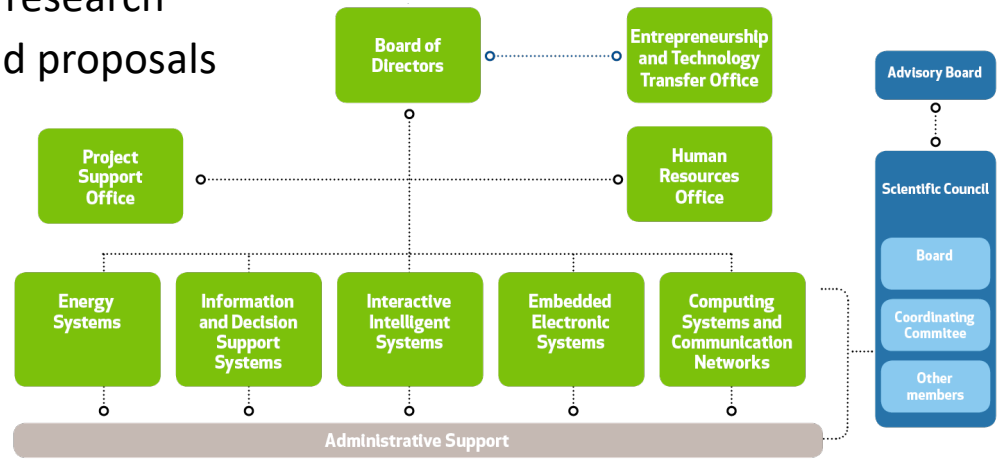
- MSc at Tor Vergata (2002)
  - Thesis on Formal Verification of the HTTP**R** protocol (Adv. Prof. B. Ciciani)
- PhD at Sapienza (2004-2007)
  - Protocols for End-to-End Reliability in Multi-tier systems (Adv. Prof. F. Quaglia)
- PostDoc at Rome University (2007)
- Senior Researcher at INESC-ID, Lisbon, Portugal (2008-today)
- Assistant Professor, Comp. Engineering, U. Lisbon (2011-2015)
- Associate Professor, Comp. Engineering, U. Lisbon (2015-today)

About IST

- IST, Lisbon University:
  - Top engineering school of Portugal
  - Two sites: Lisbon center & Tagus Park
- Computer Engineering Department:
  - 91 Faculty members, 5 scientific areas
  - Pioneering open search process for faculty positions
- Courses I've been teaching so far:
  - BSc: Operating Systems, Computer Architectures
  - MSc: Highly Dependable Systems, Distributed Systems
  - PhD: Advance Topics in Parallel & Distributed Systems

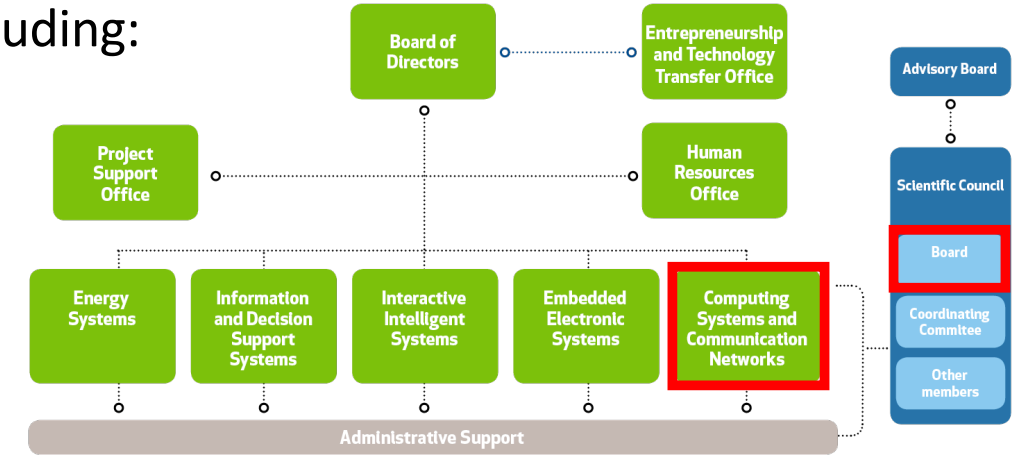
About INESC-ID

- Research center affiliated with IST
  - Partly owned by IST
    - No-profit & private nature enables agile processes (e.g., hiring, purchases)
  - Hosts researchers (mostly IST faculty members) with diverse background
    - Strong impulse to pursue **interdisciplinary** research
    - Support for both project administration and proposals
      - Recently opened new office in Brussels to support EU project proposal preparation
  - 20th anniversary in 2019!



About INESC-ID

- I am a member of the Distributed Systems Group
  - 15 faculty members from IST
    - 2 full professors, 5 associate professors, 8 assistant professors
  - Expertise in a broad range of areas, including:
    - Autonomic computing
    - Fault tolerance
    - Mobile computing
    - Parallel programming
    - Theory of distributed computing
    - Transaction processing
    - Security



- Member of the Scientific Board of the INESC-ID in 2018

Roadmap

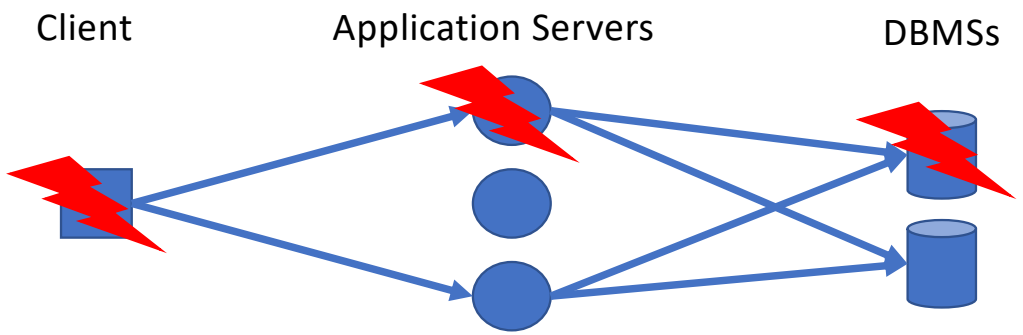
- About me
- About IST & INESC-ID
- An overview of my past research activities
- Current research lines:
  - Transactional Memory & emerging HW technologies:
    - Persistent Memory
    - GPUs
  - Leveraging Symbolic Execution for Distributed Transactional Systems
  - Parallel/distributed platforms for Machine Learning

Past research activities: MsC Thesis (2002)

- Formal Verification of HTTP**R**
  - Extension of HTTP to ensure exactly-once semantics
    - Goal: enhance reliability of Web Services
      - very hot topic back in the days!
  - Model checking of HTTPR specification (PROMELA & SPIN)

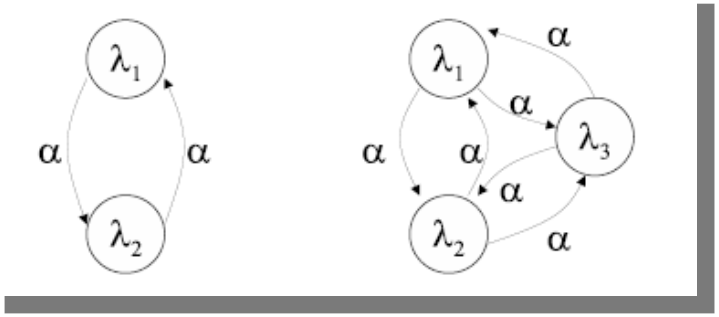
Past research activities: PhD thesis (2003-2006)

- Jointly address **reliability** and **performance** issues in multi-tier systems
- Mix of theory and systems:
  - Theory: minimum synchrony requirements for solving the e-Transaction problem
    - End-to-end reliability guarantees in three-tier system
    - In a nutshell: exactly-once semantics despite failures of clients, mid-tier, back-end DBMS(s)
  - Practice: multi-path/parallel invocation schemes in multi-tiered applications
    - Goal: reduce client-perceived latency in geo-distributed systems



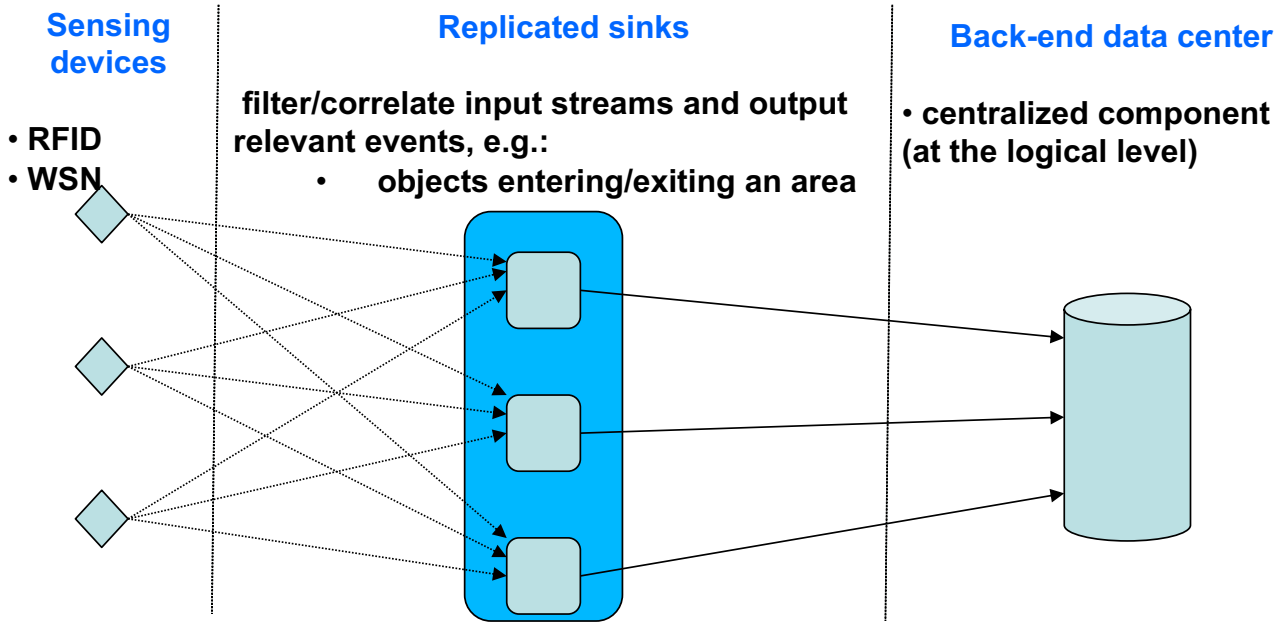
Past research activities: PostDoc@Sapienza(2007) (1/3)

- Approximate solution of MMPP/MMPP/1 queues
  - Markov Modulated Poisson Processes:
    - Poisson processes whose means change according to a Markov Chain
    - Useful to capture burstiness, self-similarity, failure/recovery of servers



Past research activities: PostDoc@Sapienza(2007) (2/3)

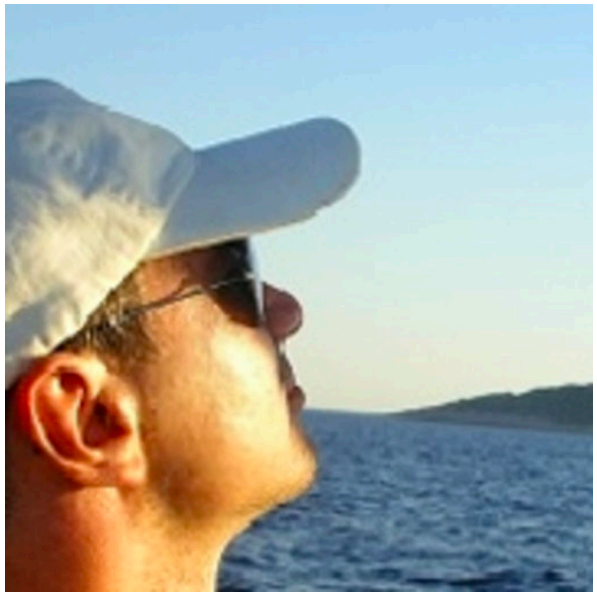
- Efficient replication schemes for data streaming applications



Past research activities: PostDoc@Sapienza(2007) (3/3)

- Performance modelling of Multi-Version Concurrency Control
  - Analytical model of Oracle's MVCC scheme

- Main publication: MASCOTS'08



Past research activities: PostDoc@INESC-ID (2008-2010) (1/2)

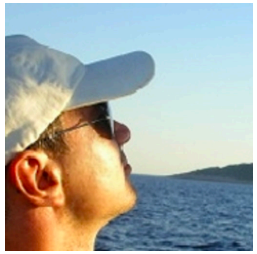
- **Distributed Software Transactional Memory**
- My group at INESC-ID pioneered this research area
  - Hot topic at the intersection between STM and distributed databases
  - Advantage position thanks to FénixEDU
    - Management system of IST's teaching activities (Moodle-like)
    - One of the first systems to adopt STM in production….
    - …and faced with real reliability and scalability challenges!
- Research funded by 2 Portuguese research projects :
  - PASTRAMY, coordinated by Prof. Luís Rodrigues
  - ARISTOS, my first project as coordinator

Past research activities: PostDoc@INESC-ID (2008-2010) (2/2)

- Investigation of a number of research lines:
  - Design of novel replication protocols for STM
    - PhD thesis of Nuno Carvalho (IST)

  - Speculative transaction processing techniques
    - PhD thesis of Roberto Palmieri (Sapienza)

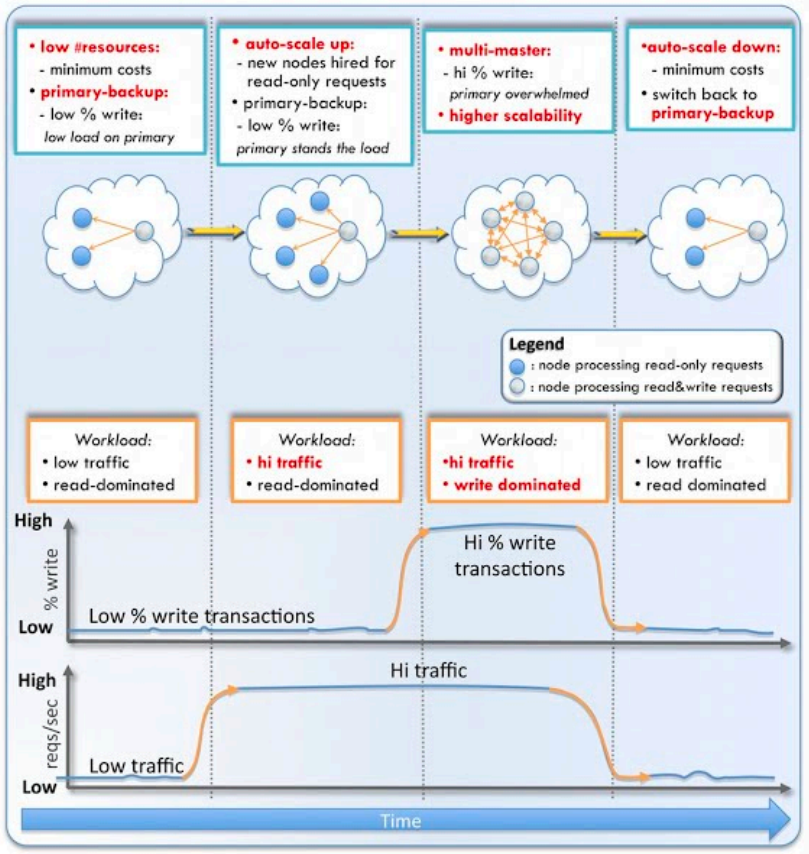
  - Autonomic replicated STM (start of research on ML for system optimization)
    - PhD thesis of Maria Couceiro (IST)
  - Performance modelling of STM concurrency control schemes
    - PhD thesis of Pierangelo Di Sanzo (Sapienza)

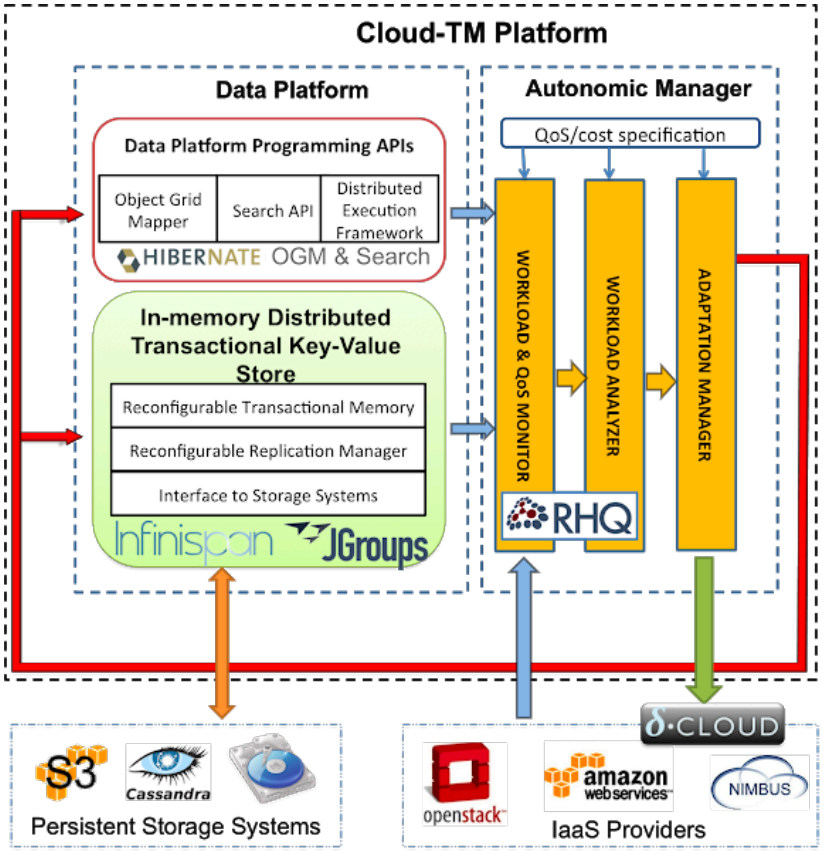


Past research activities: Assistant Professor@IST (2011-2015)

- Research propelled by 3 EU projects:
  - Cloud-TM (serving as coordinator)
    - Distributed TM platform for the Cloud
    - Natural evolution of previous research on DTM, with emphasis on:
      - Scalability
      - Elasticity
      - Self-tuning
  - FastFix (participant)
    - Reducing cost and latency of software maintenance
    - INESC-ID focus: deterministic fault replication
      - in multi-threaded applications (non deterministic scheduling)
      - anonymizing sensible application data
  - Euro-TM (serving as chair)
    - Pan-european research network on Transactional Memory

Past research activities: Cloud-TM (2011-2013) (1/2)

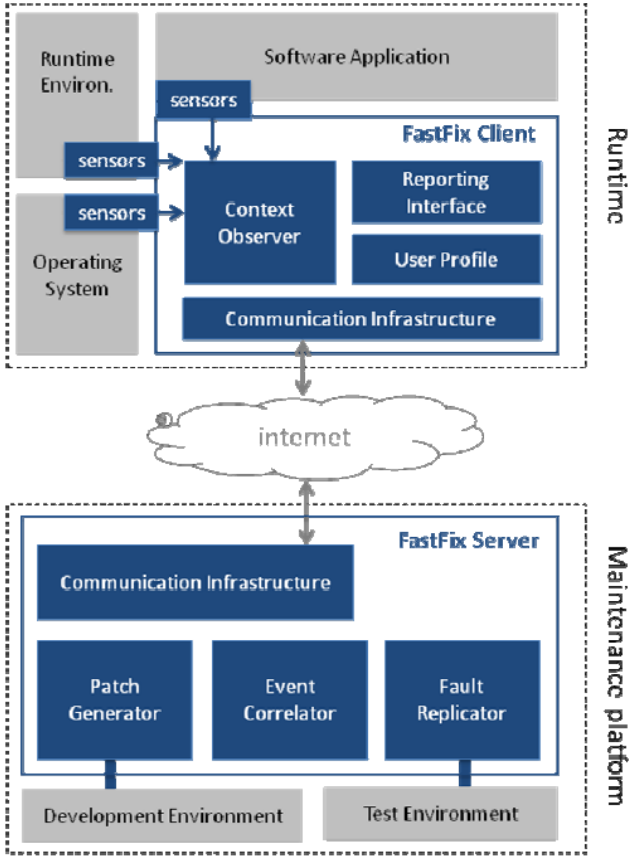


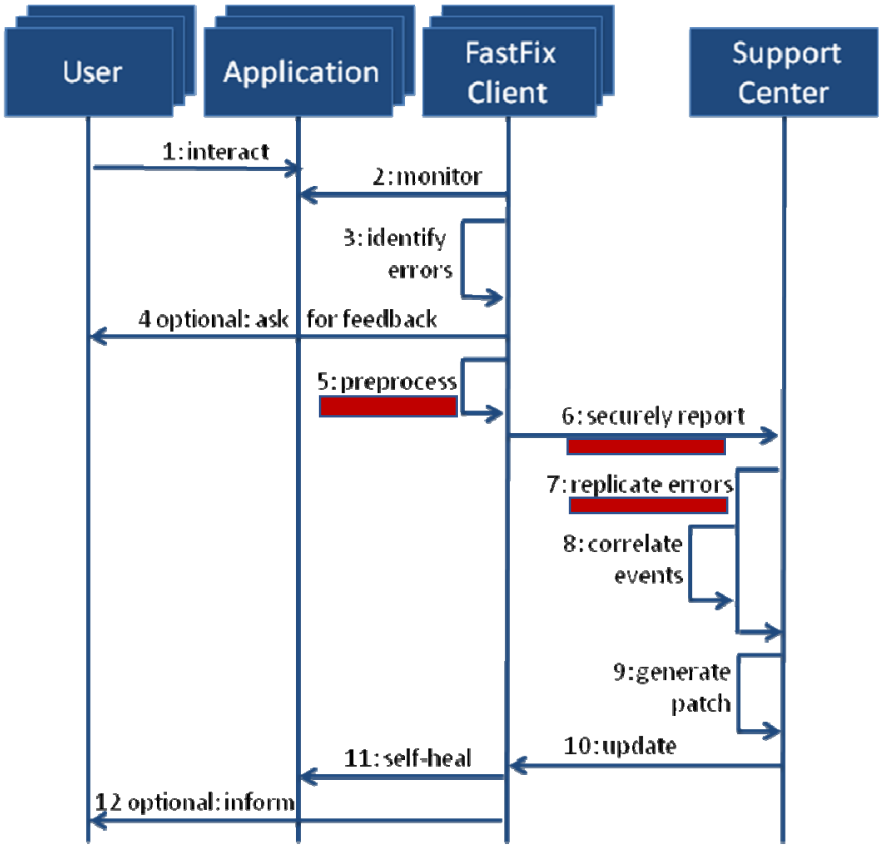


Past research activities: Cloud-TM (2011-2013) (2/2)

- Main research lines:
  - Scalable protocols for distributed transactions
    - PhD thesis of Sebastiano Peluso (Sapienza & IST)
    - IEEE/IFIP William C. Carter PhD Dissertation Award in Dependability 2016
  - Enhancing the efficiency of (non-distributed) TM, both hw and sw
    - PhD thesis of Nuno Diegues (IST)
  - Joint usage of analytical methods and machine learning for modelling and optimization of complex systems
    - PhD thesis of Diego Di Dona (IST)

Past research activities: FastFix (2011-2013) (1/2)

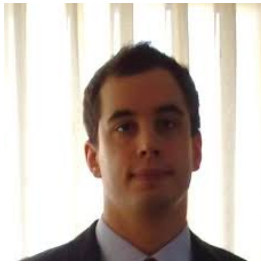




Past research activities: FastFix (2011-2013) (2/2)

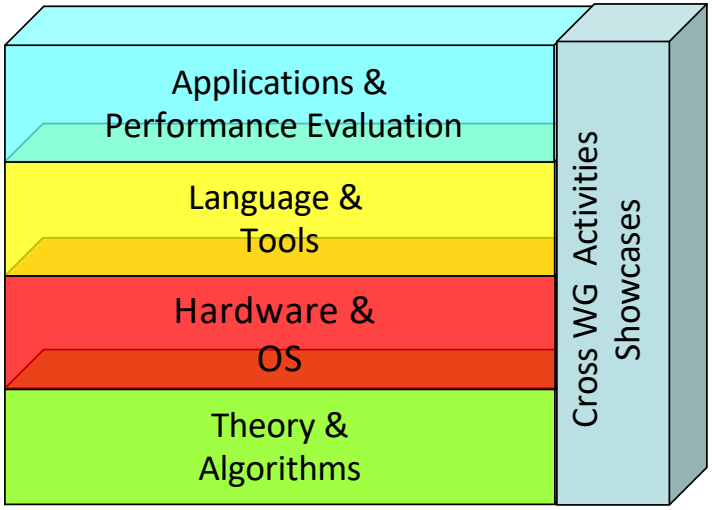
- 2 main research lines:
  - Reducing cost of deterministic bug replay in multi-threaded programs
    - How? By combining the partial traces of multiple clients
      - Reduce logging cost at each client, leveraging large client populations
      - Recombine traces of independent executions using lightweight similarity metrics
    - PhD thesis of Nuno Machado (IST)

  - Anonymization of information included in bug reports
    - Leverage symbolic execution to identify alternative user inputs that lead to the same bug
      - First contact with symbolic execution toolkits
    - PhD thesis of João Matos (IST)



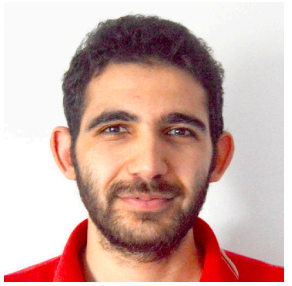
Past research activities: Euro-TM (2011-2015)

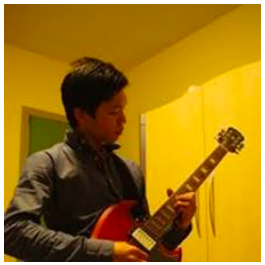
- Research network bridging >200 researchers, 50 institutions, 17 EU countries active in the área of Transactional Memory
- Interdisciplinary research across the entire stack
- Support for mobility of researchers
- Organization of 10 scientific meetings
- Organization of 2 PhD schools
- Dissemination of results in industrial conferences
- 20 joint project proposals
- Final book coauthored by 60 autors from 13 countires



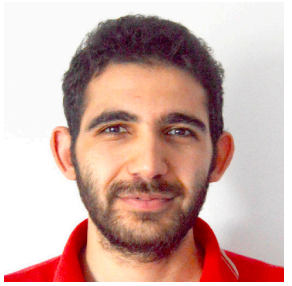
Past research activities: 2015-2018

- 4 main research lines:
  - Energy efficiency for TM systems
    - PhD Thesis of Shady Issa (IST & KTH)
  - Extending capacity of Hardware TM (HTM) via software mechanisms
    - PhD Thesis of Shady Issa (IST & KTH)
  - Integrating Futures and (S)TM
    - Phd Thesis of Jingna Zeng (IST & KTH, planned for. Jan. 2020)
  - Speculative processing in partially replicated transactional systems
    - PhD Thesis of Zhongmiao Li (IST & UCL, planned for Jan. 2020)



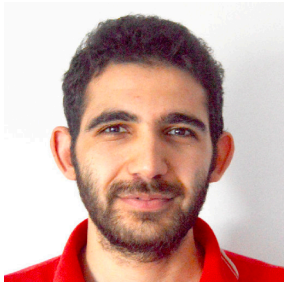


Past research activities: Energy efficiency of TM systems



- Due to their speculative nature, TM systems are prone to waste work/energy when conflicts do arise.
- Contention Management (CM) policies have long been studied to enhance TM efficiency in unfavorable workloads
- **Green-CM:**
  - First CM designed to maximize energy efficiency
  - 2 main ideas:
    - Adaptive implementation of “wait” mechanism (spin vs sleep)
    - Leverage Dynamic Voltage and Frequency Scaling via Asymmetric CM
      - Diversify duration of waiting phases among threads (linear vs exponential back-off)
        - Threads using EBO likely to release processor for long time, lowering thermal envelope
        - Threads using LBO likely to be boosted by DVFS

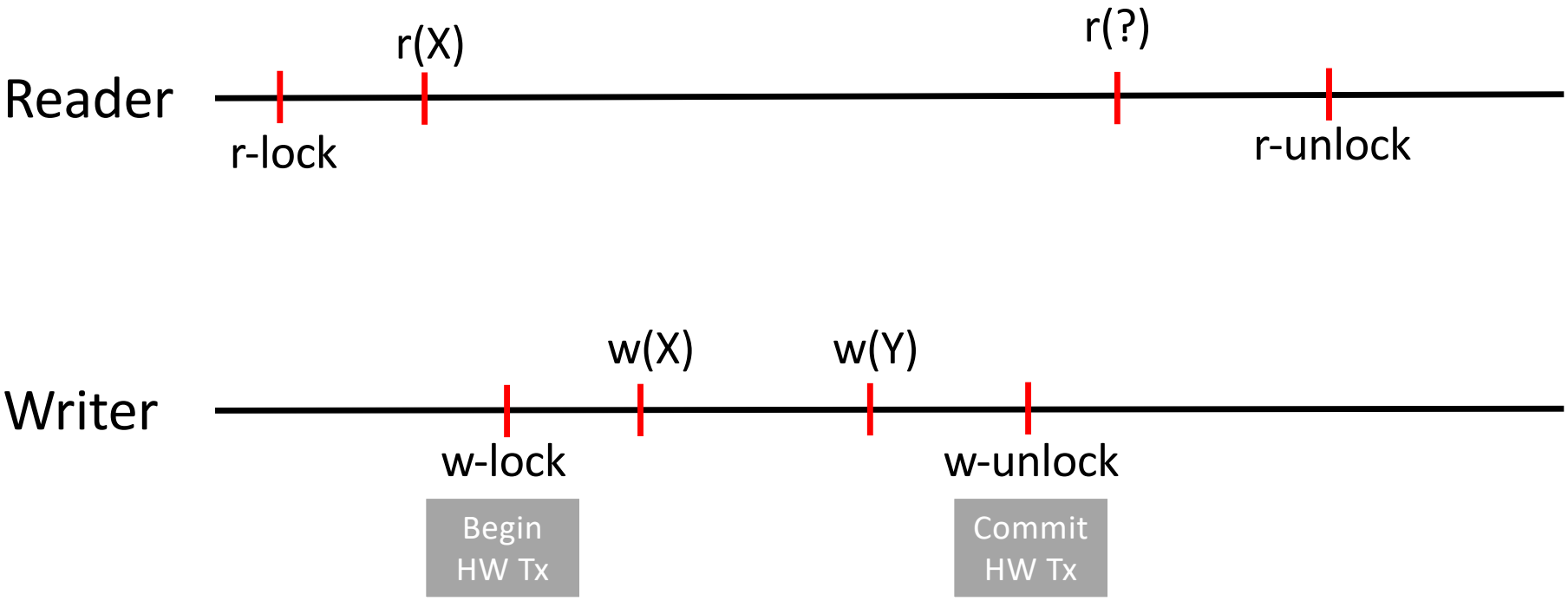
Past research activities:
Stretching HTM capacity via software techniques



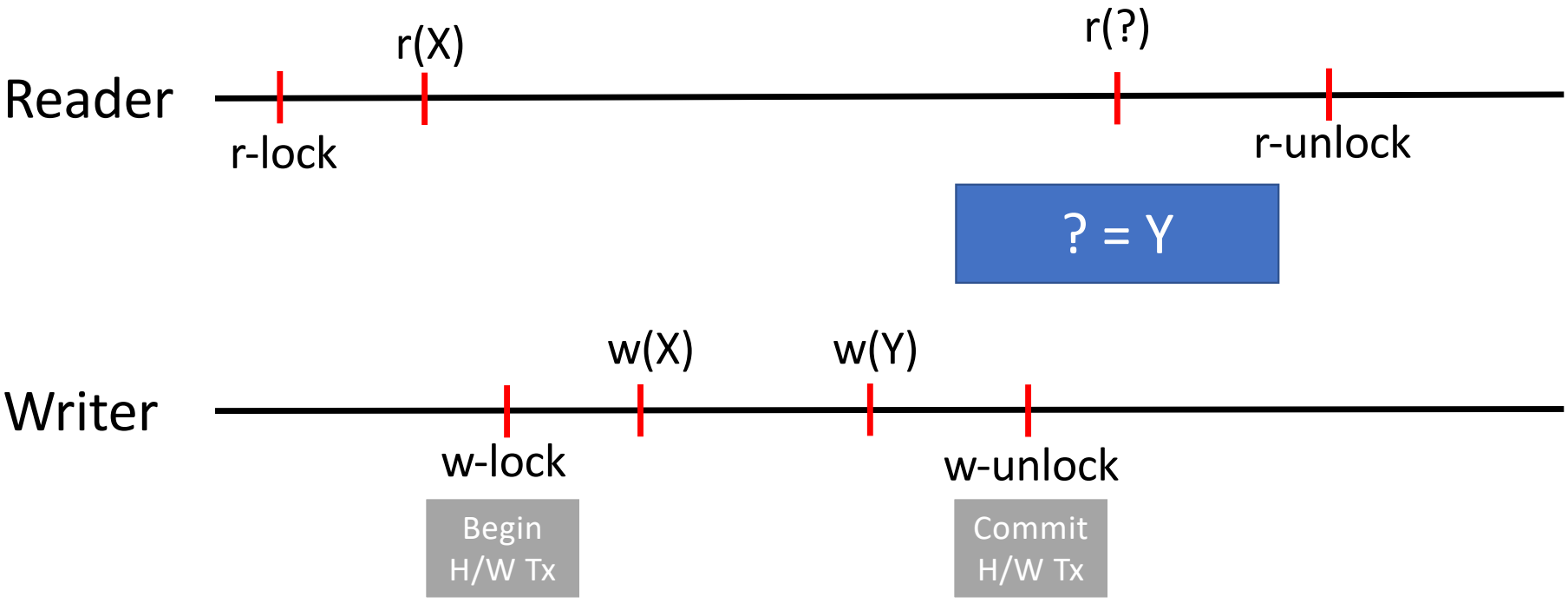
- Base idea:
  - Run read-only transactions without any HW instrumentation
    - Infinite capacity
  - Allow update transactions to commit only in absence of concurrent readers
    - Exploit IBM Power8/9 tx suspend/resume to let writers monitor state of concurrent readers

- Applied to elide Read Write Lock
  - Hardware Elided Read Write Lock (HERWL) [EuroSys'16]

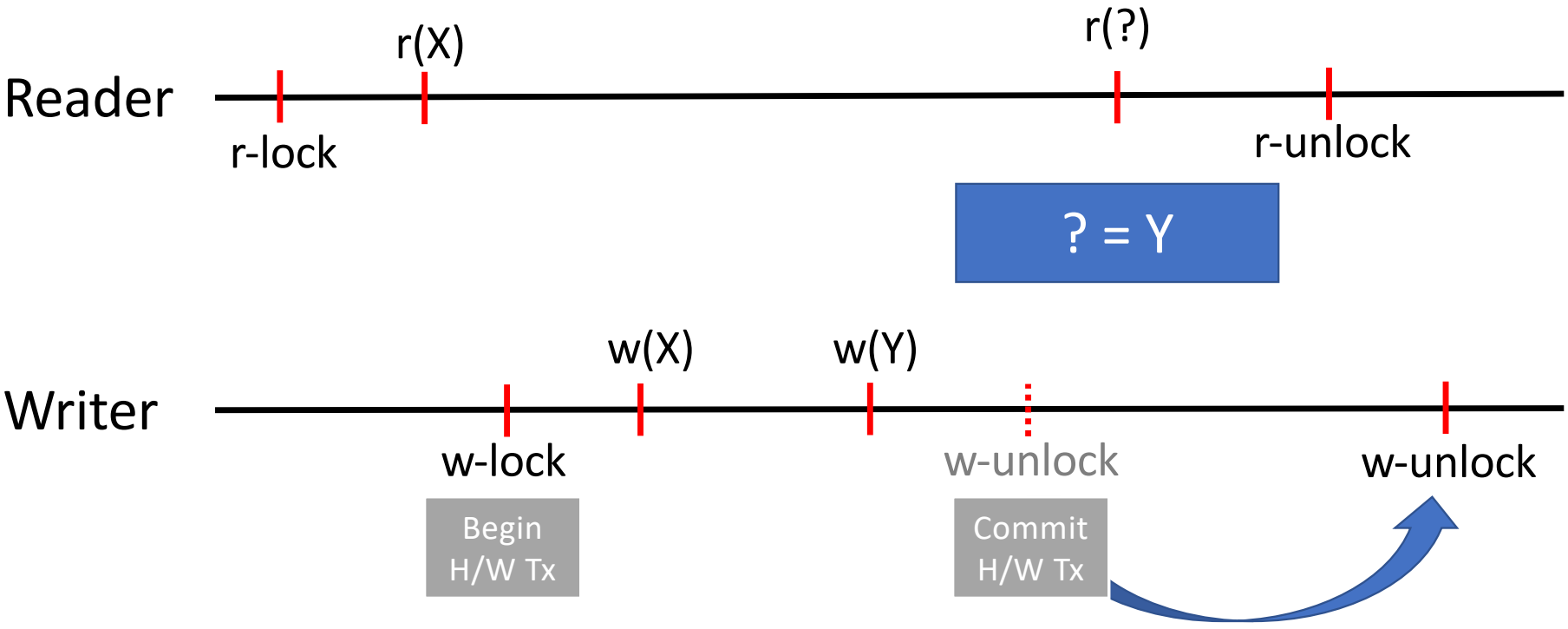
Past research activities:
Stretching HTM capacity via software techniques



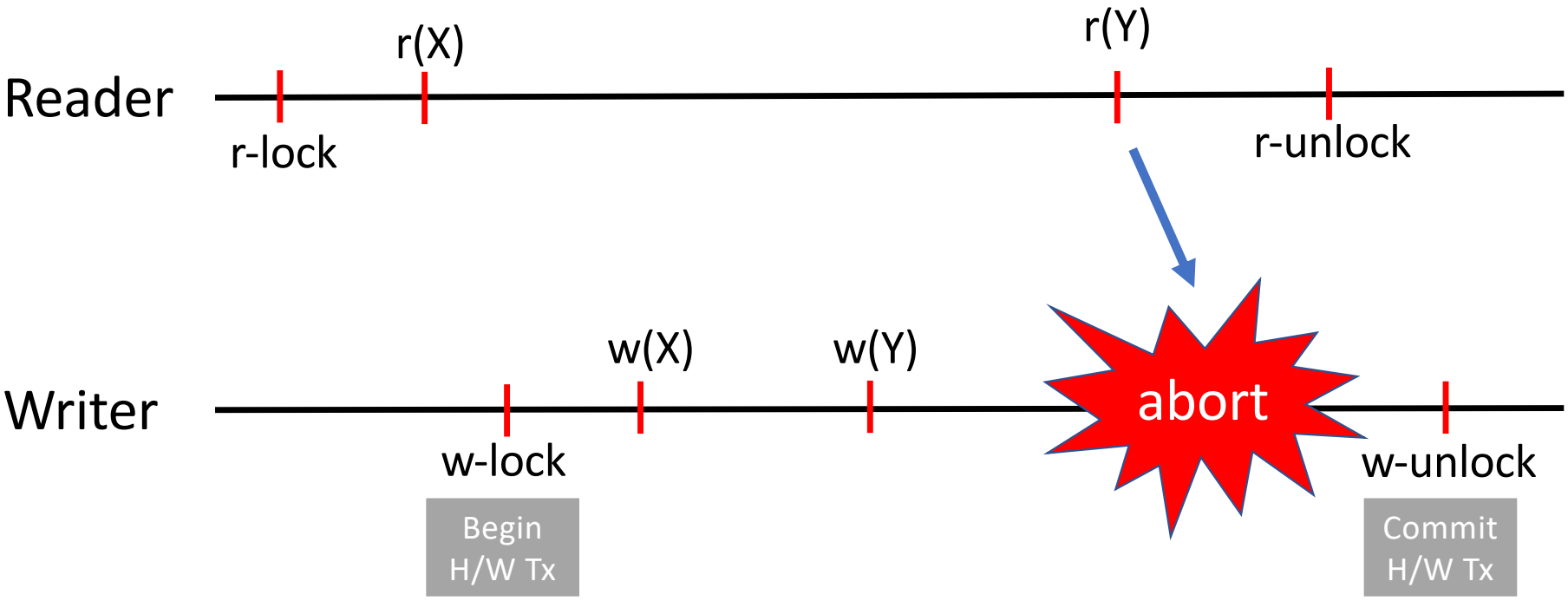
Past research activities:
Stretching HTM capacity via software techniques



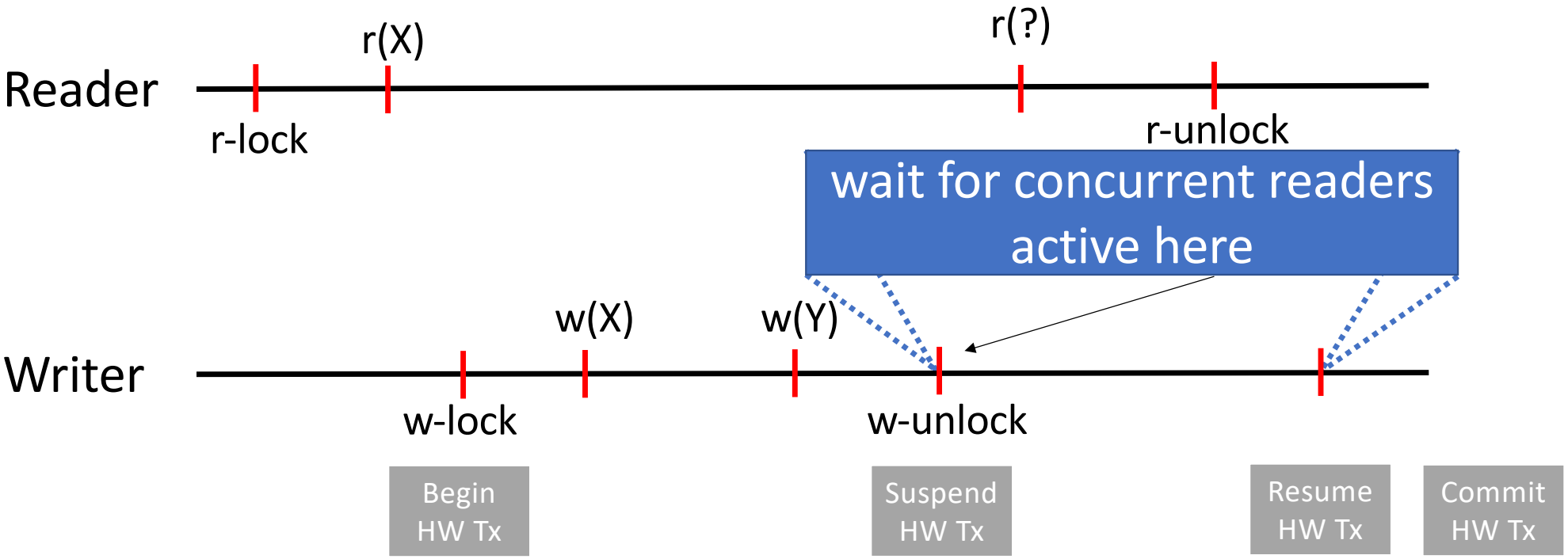
Past research activities:
Stretching HTM capacity via software techniques



Past research activities:
Stretching HTM capacity via software techniques



Past research activities:
Stretching HTM capacity via software techniques

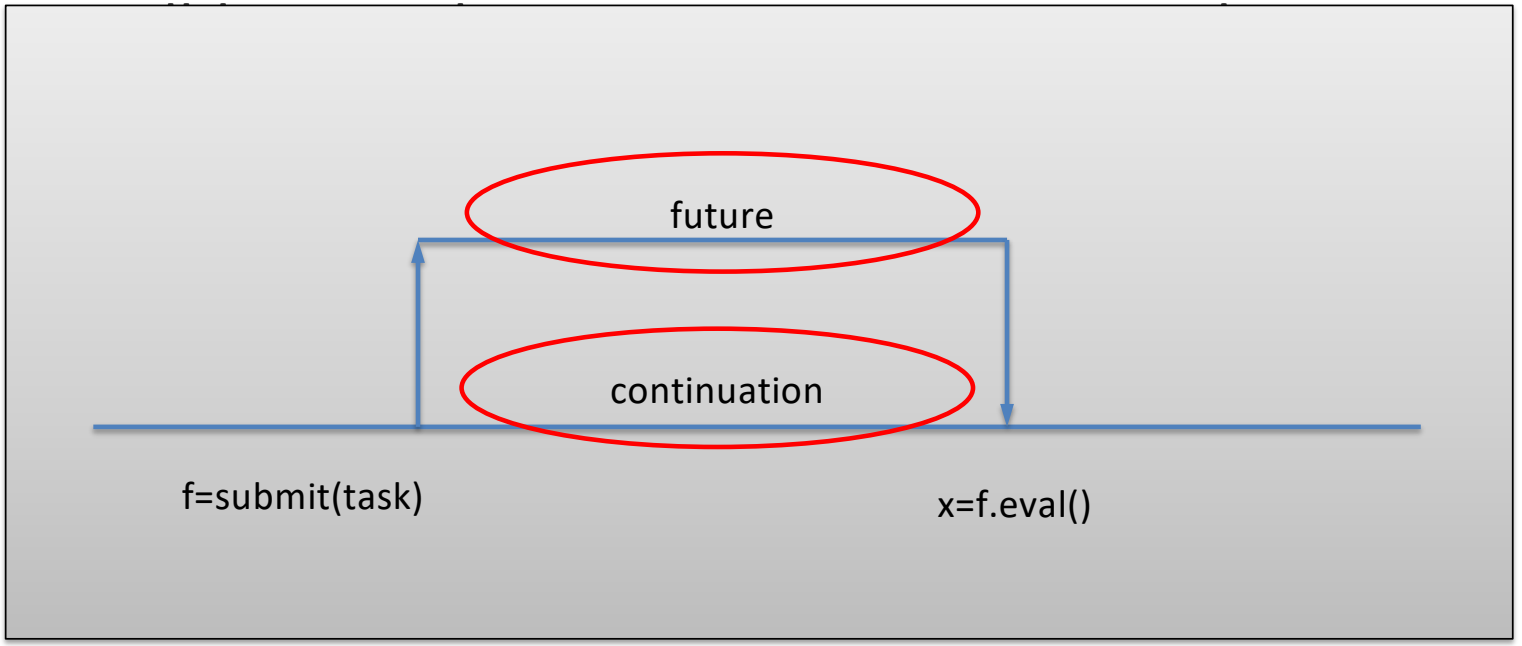


Past research activities: Stretching HTM capacity via software techniques

- Enhancements:
  - Increase capacity of update transactions by exploiting another unique feature of IBM Power processors:
    - Rollback Only Transactions (ROTs):
      - Atomic but not isolated HW transaction
      - ROTs do not track readsets of transactions
      - ROTs have infinite read capacity
      - Unsafe to run concurrently!
- Follow ups:
  - Enable concurrent execution of ROTs [DISC'17]
  - Avoid reliance on IBM-unique HTM features (Suspend/Resume + ROTs) [MW'18]
  - Adaptation of the mechanism to ensure Snapshot Isolation [PPoPP'19]

Past research activities: Integrating futures and (S)TM

```
Future<T> f = submit(task); // submit an asynchronous task
 ...                        //do something else
T x = f.eval();             //pick up task's result
```

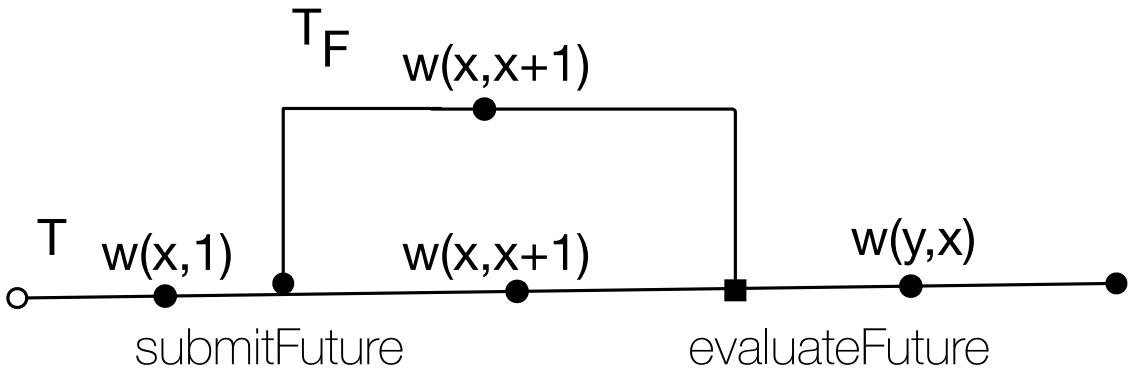


How to support Futures in TM?

- Basic idea – *Transactional Future*:
  - allow transactions to submit/evaluate futures
  - futures run as transactions that:
    - can access shared variables
    - can return some result value
  - a future and its continuation appear as atomic units

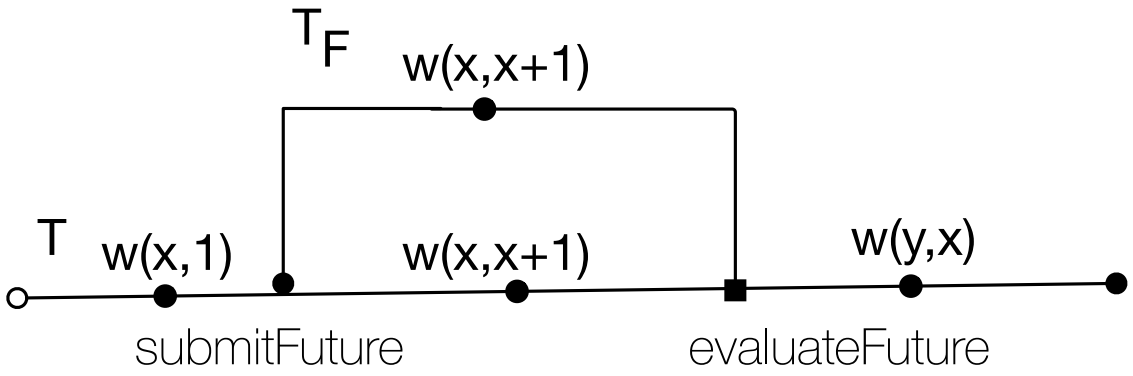
- 2 key issues:
  - which serialization orders should be allowed for futures and continuations?
  - how to define the boundaries of a continuation?

Transactional Futures Semantics: a basic example



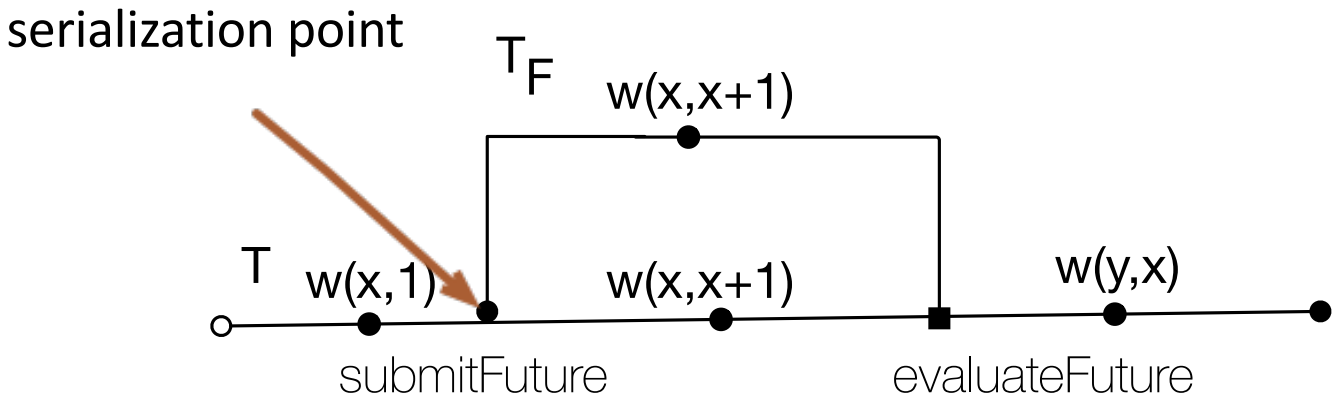
- Intuitively we want to guarantee atomicity between T_F and its continuation…

Transactional Futures Semantics: a basic example



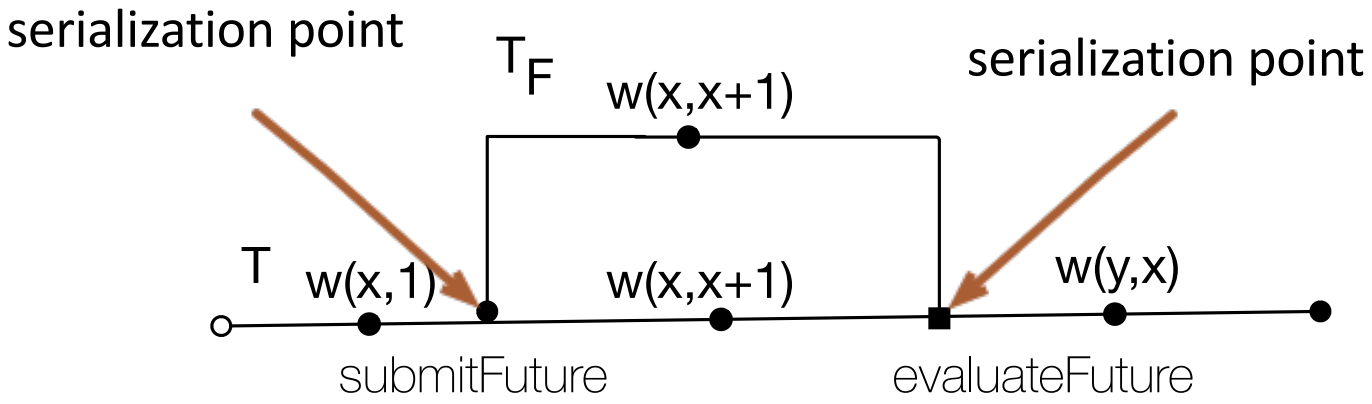
- …but what are the expected serialization orders between T_F and its continuation?

Transactional Futures Semantics: a basic example



- …but what are the expected serialization orders between T_F and its continuation?
  - before T_F's continuation: strongly ordered

Transactional Futures Semantics: a basic example



- …but what are the expected serialization orders between T_F and its continuation?
  - before T_F's continuation: strongly ordered
  - either before or after T_F's continuation: weakly ordered

How to support Futures in TM?

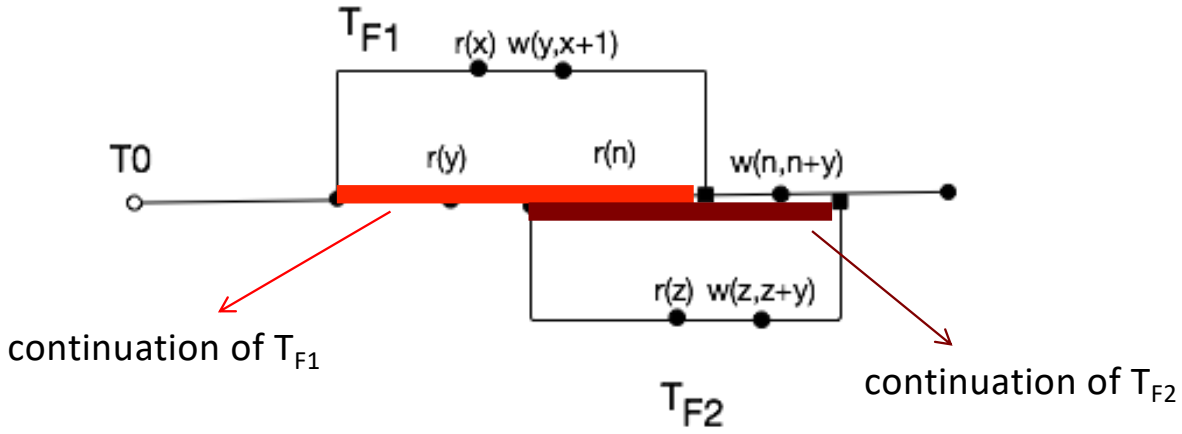
- Basic idea – *Transactional Future*:
  - allow transactions to submit/evaluate futures
  - futures run as transactions that:
    - can access shared variables
    - can return some result value
  - a future and its continuation appear as atomic units

- 2 key issues:
  - which serialization orders should be allowed for futures and continuations?
  - how to define the boundaries of a continuation?

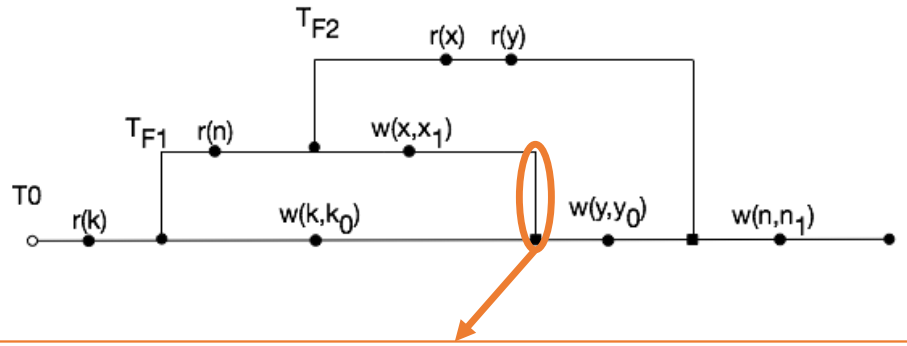
How to define continuations?

- The Future abstraction enables parallel computations with complex dependency graphs, e.g.:
  - submitting futures from within continuations
  - escaping transactional futures
    - within the same top-level transaction, or
    - submitted and evaluated in different top-level transact.
- **Pro**: great flexibility for expert programmers
- **Con:** non-trivial to define continuations

Submission of a future by a continuation



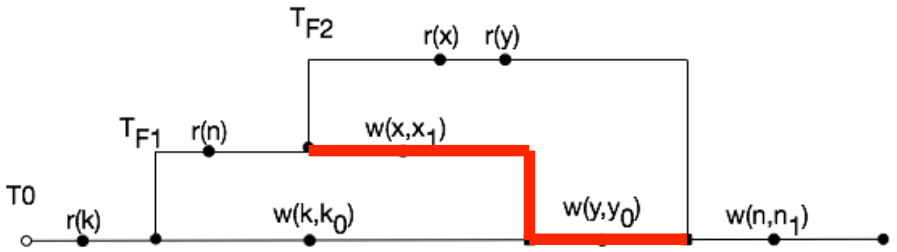
Escaping transactional future



Here T_{F1} returns the reference of T_{F2} to T0,
in order to allow T0 to evaluate T_{F2}

Escaping transactional future

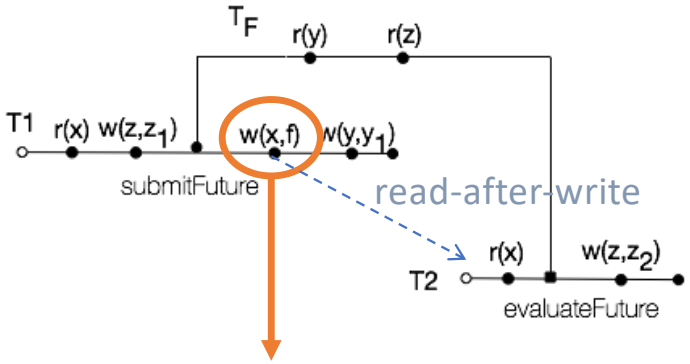
Logic underlying definition of T_{F2} continuation:
Sequence of causally-related operations that leads from T_{F2}'s submission to its evaluation



- Continuation of T_{F2} spans two transactional futures!
- T_{F2} should observe both writes on x and y or none!

Transactional future escaping from its top-level transaction

T_F is used as a communication means between T1 and T2.

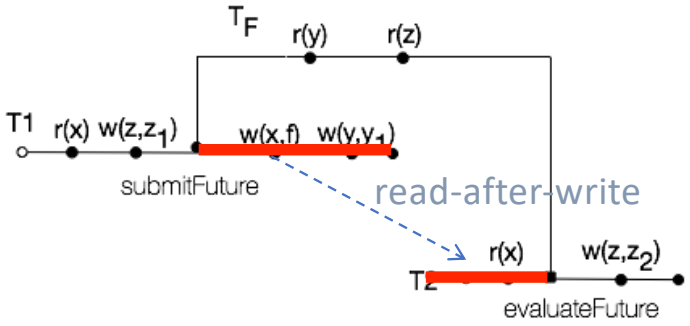


T1 writes T_F's reference in variable x and commits.
This allows a different top-level transaction, e.g. T2, to evaluate T_F.

Transactional future escaping from its top-level transaction

Logic underlying definition of T_F continuation:

Sequence of causally-related operations that leads from T_F's submission to its evaluation



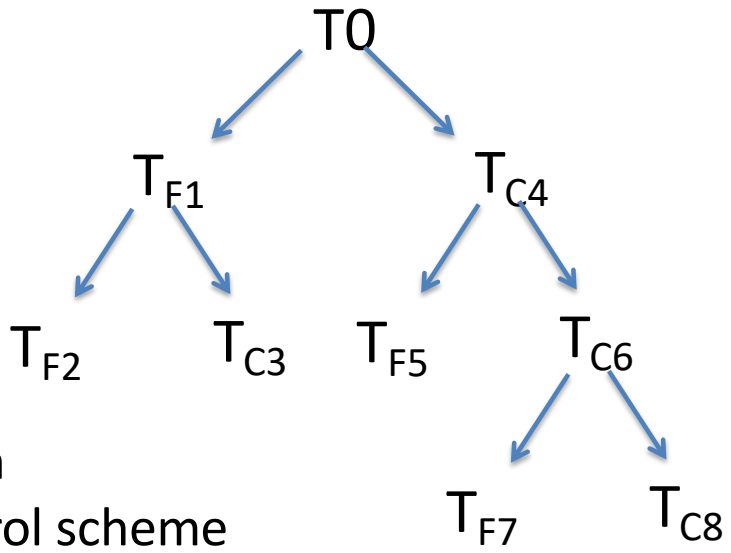
- Using the above rationale, a continuation can span two or more top-level transactions ➔ *strongly atomic continuation*
- Constrain T_F's continuation within the top-level tx that submitted T_F ➔ *weakly atomic continuation*

How to formalize these concepts?

- Via a Future Serialization Graph:
  - similar in spirit to transaction serialization graph
  - but aimed to:
  1. allow for rigorous definition of futures and their continuations
  2. capture ordering relations between futures and continuations

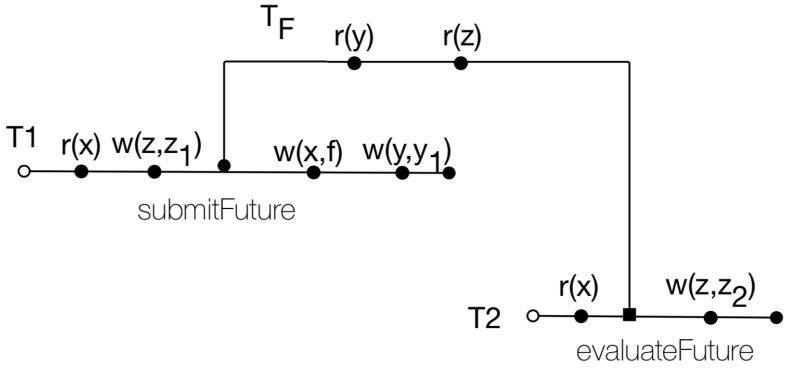
How to implement the abstraction of Transactional Futures

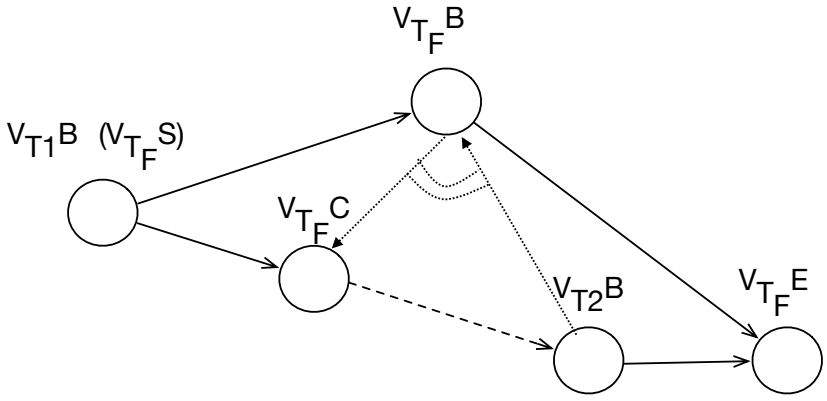
- First implementation proposed in [ICPP'16]
  - Support only for strongly ordered futures
  - Transactional futures serialized solely upon submission:
    - No escaping futures
      ➔ FSG encoded via a tree
    - Versions produced by futures managed via an innovative multi-versioned concurrency control scheme



How to implement the abstraction of Transactional Futures

- Second implementation (under submission)
  - Support for weakly ordered futures
    - 2 serialization points for futures
    - Possibility of escaping futures
  - Novel concurrency control based on explicit management of the FSG



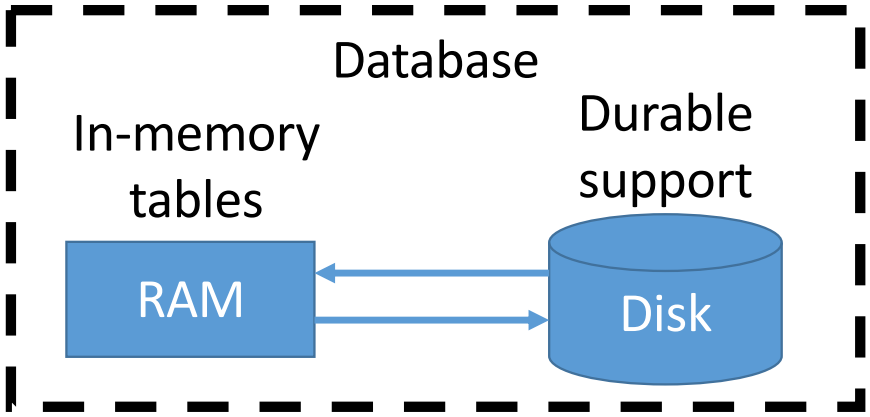


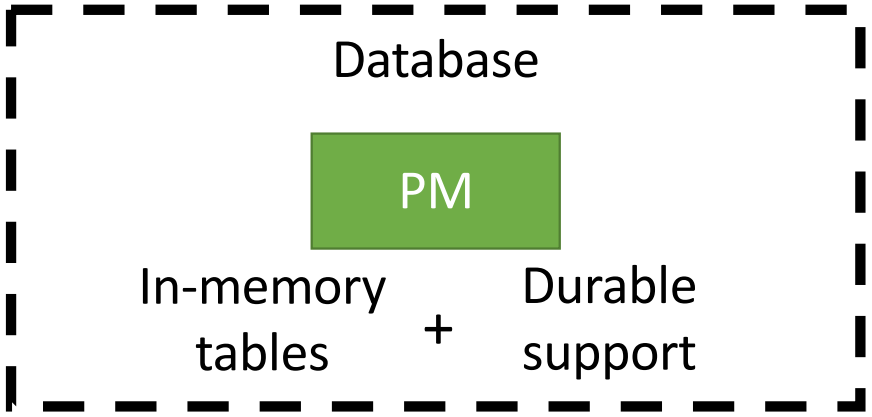
Roadmap

- About me
- About IST & INESC-ID
- An overview of my past research activities
- Current research lines:
  - Transactional Memory & emerging HW technologies:
    - Persistent Memory
    - GPUs
  - Leveraging Symbolic Execution for Distributed Transactional Systems
  - Parallel/distributed platforms for Machine Learning

Persistent Memory (PM)

- Fast byte-addressable storage
- Higher density when compared with volatile RAM
- Expect writes to be slower than RAM (2x-5x):
- Subject to wear off upon write (technology dependent)

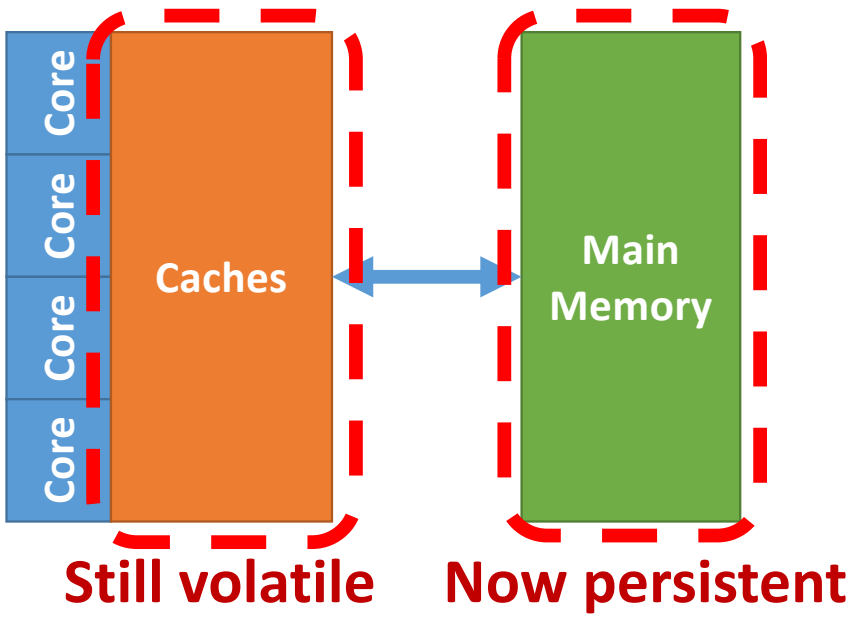




Free durability?

Persistent Memory (PM)

- CPU Caches (most likely) will continue being volatile:
  - What is effectively written into memory?

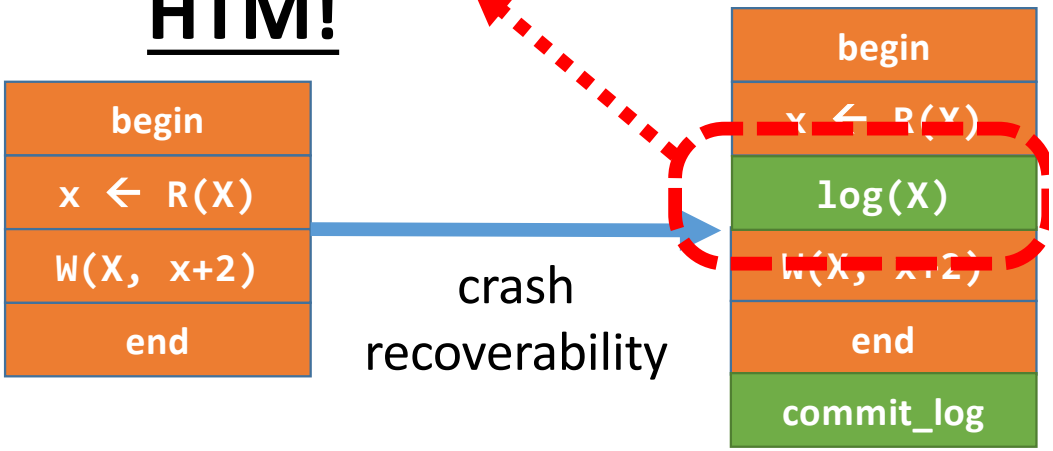


- Applications must explicitly bypass caches:
  - clflush, clflushopt, clwb
  - Else:
    - writes are not guaranteed to enter PM
    - writes may be reordered
- What about applications that require atomic access/transactions to memory regions?

Integrating PM and Software-based TM

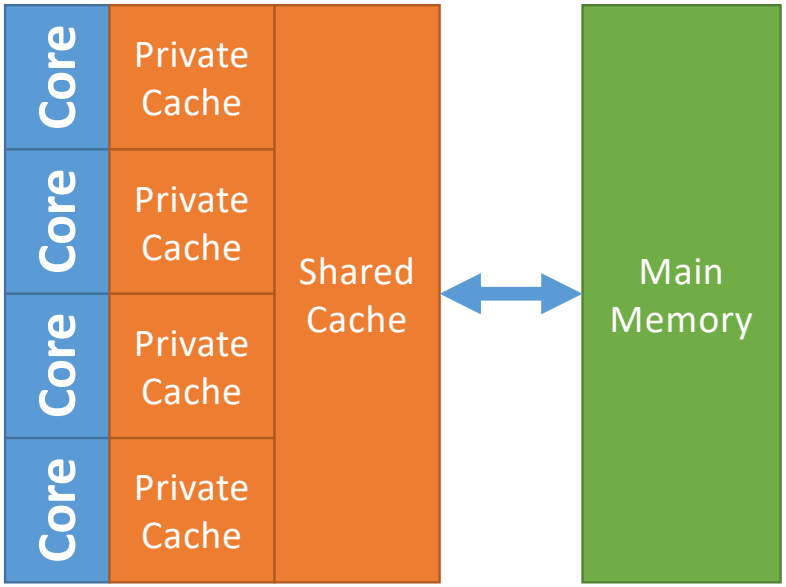
- Durability of transactions regulated via software concurrency is well-understood: decades of literature in DBMS area!
- Example based on a recent PM-oriented software-based approach [**ASPLOS'16**]:
  - Upon write
    1. Lock the value
    2. Log (flush) the old value
    3. Do the write
  - Upon commit
    1. Flush write-set
    2. Add commit marker
    3. Unlock values
    4. Destroy log

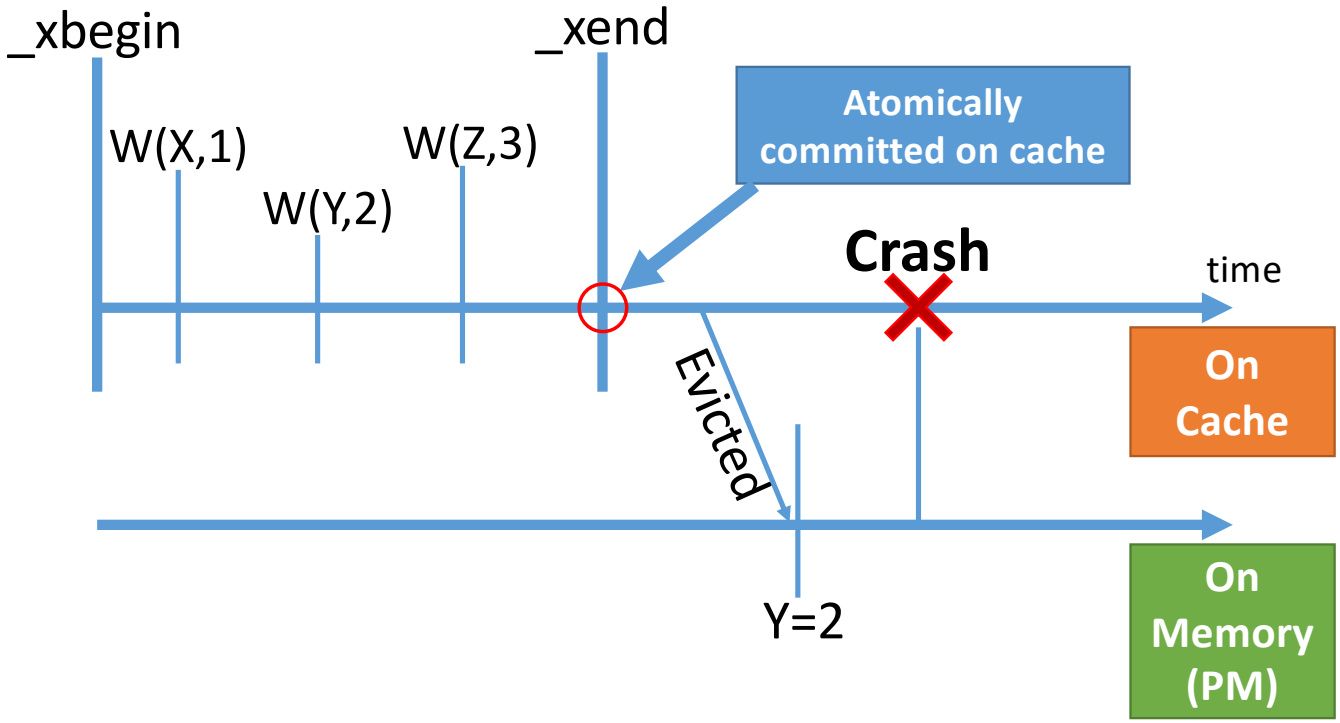
Unfortunately not possible with HTM!



Hardware Transactional Memory (HTM)

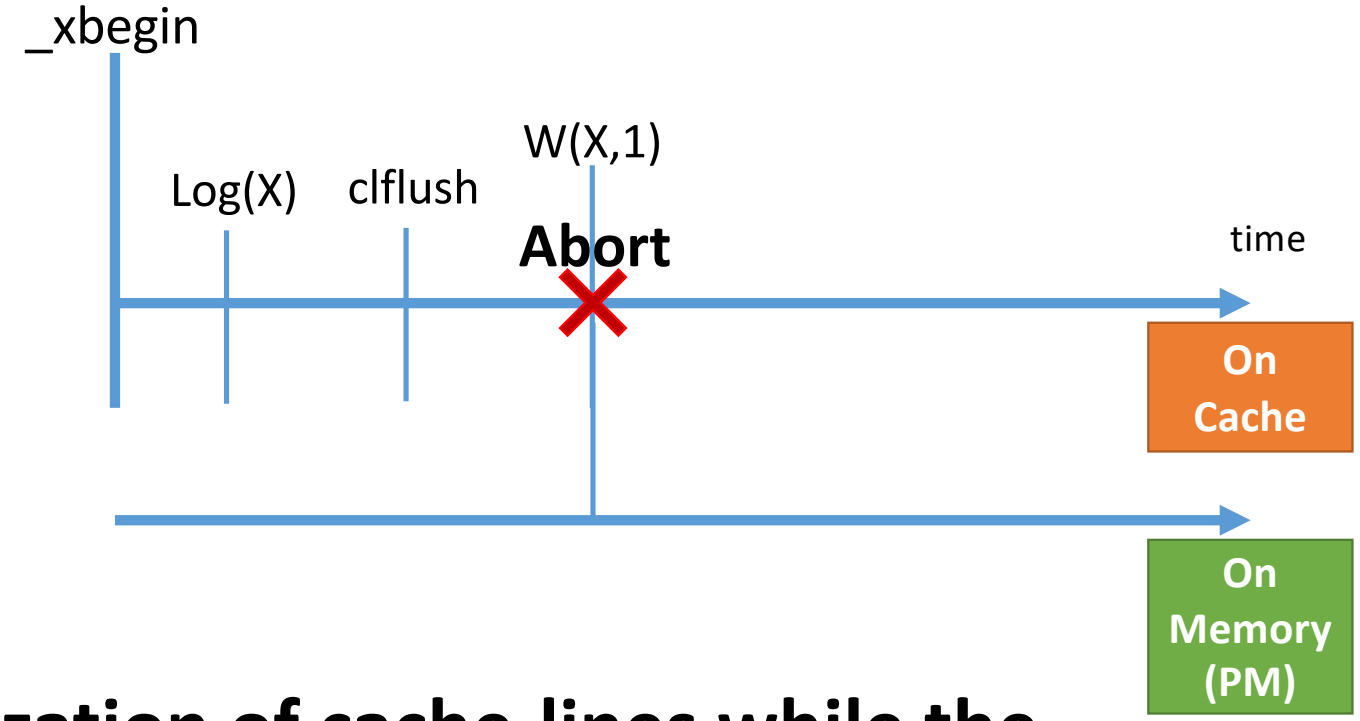
Concurrency is built on on cache coherency protocols [**ISCA'93**]





Example of a story of a non-durable (and non-atomic after recovery) transaction!

Hardware Transactional Memory (HTM)



Externalization of cache-lines while the transactions is running is not allowed!

Related Work

STM-based solutions[**ASPLOS'11**, **ASPLOS'16**]

- build on DBMS literature on logging schemes:
  - adapted & optimized for PM
  - flexible design
  - boilerplate on each load and store

Drawbacks:

- **STM incurs much larger overhead than HTM!**
- **Do not work with HTM**

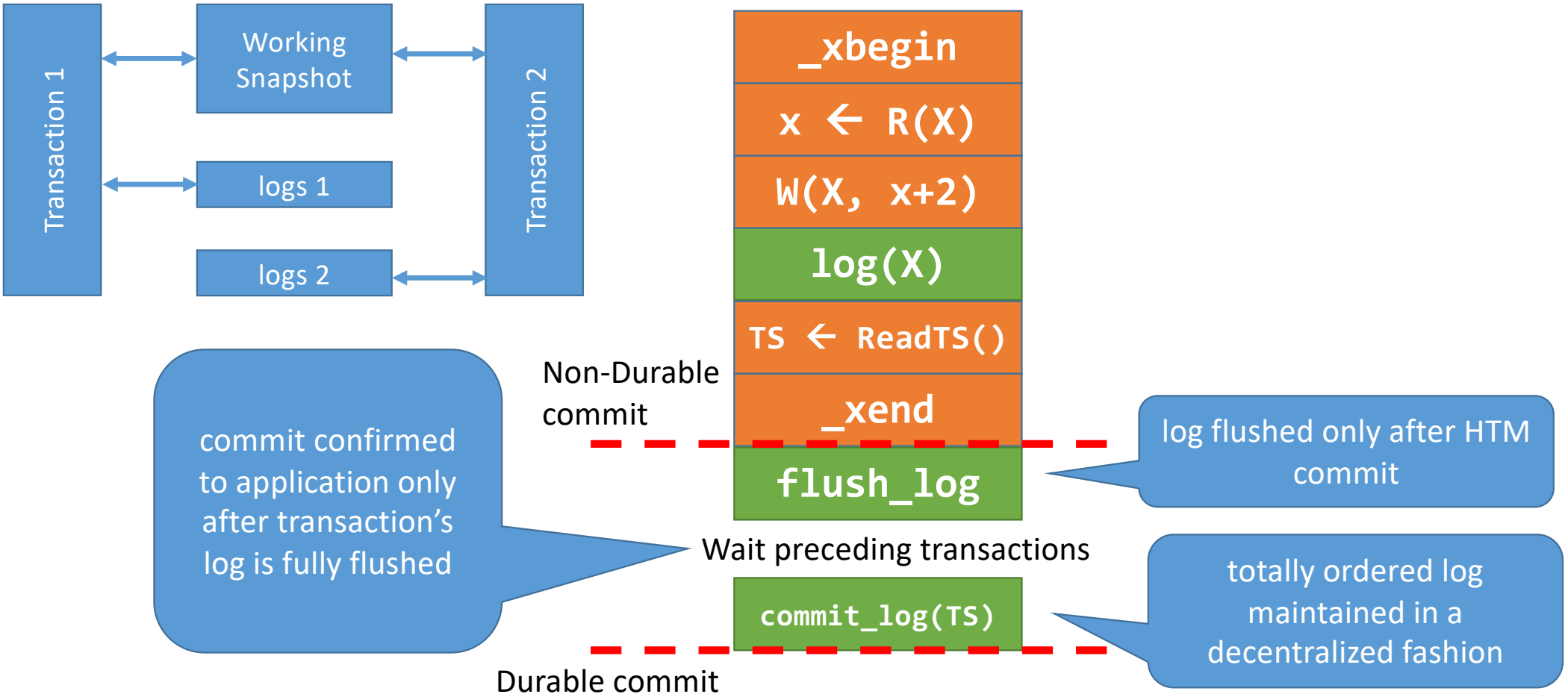
HTM-based solutions [**DISC'15**, **CAL'15**]

- Rely on modified HTM implementation
- PHTM [**DISC'15**]:
  - Flush cache-lines within transaction
  - Order writes to logs via additional locks
  - Commit flushes a commit marker

Drawbacks:

- **Incompatible with commodity HTM**
- **Additional locks reduce concurrency and available capacity**

NV-HTM: Transaction logging – 1/3



NV-HTM: Transaction logging – 1/3

Pros:

✓ Ensure interoperability with existing HTM systems!

✓ Avoid contention hot-spots to maximize scalability

Challenge:

- If a transaction is durable, all transactions it depends upon also are:
  - novel synchronization scheme based on physical clock
- Upon crash:
  - no guarantee that updates of non-durably committed transaction hit PM
  - possible corrupted snapshot upon failure!

NV-HTM: Transaction logging – 1/3

Pros:

✓ Ensure interoperability with existing HTM systems!

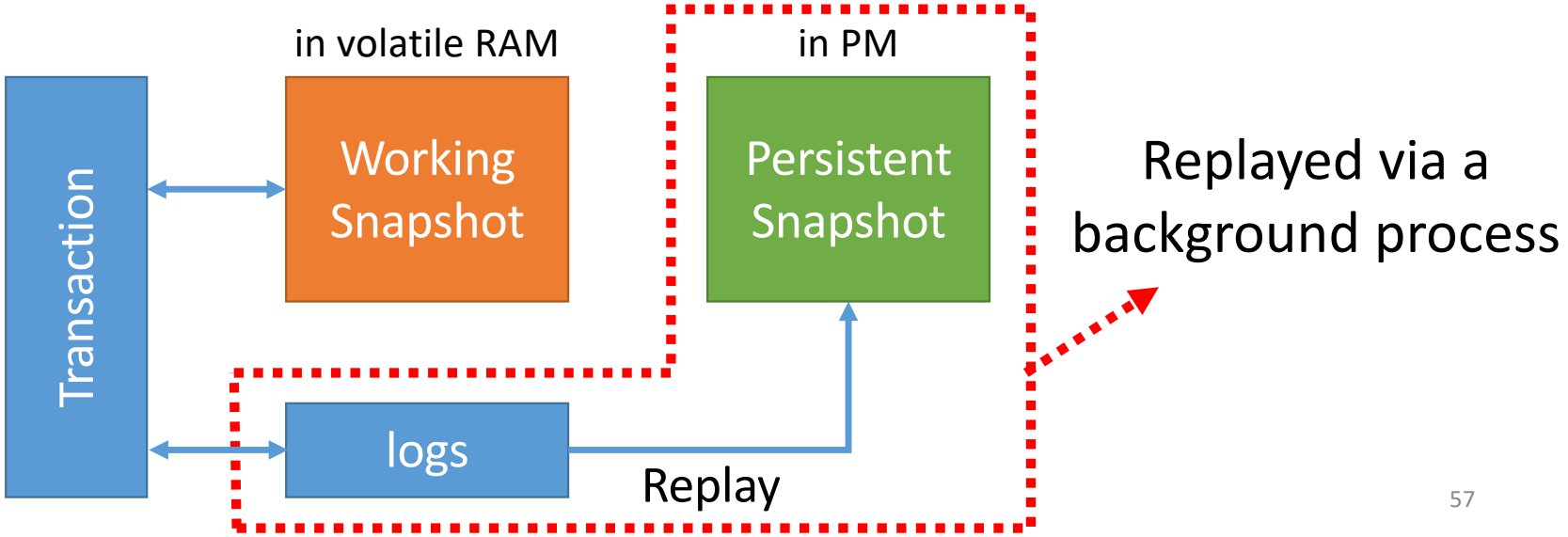
✓ Avoid contention hot-spots to maximize scalability

Challenge:

- If a transaction is durable, all transactions it depends upon also are:
  - novel synchronization scheme based on physical clock
- Upon crash:
  - no guarantee that updates of non-durably committed transaction hit PM
  - possible corrupted snapshot upon failure!

NV-HTM: Working and Persistent Snapshots – 2/3

- Application writes in a (volatile) **working snapshot**
- Logged writes are replayed asynchronously to produce a consistent **persistent snapshot** on PM
  - via background checkpoint process



NV-HTM: Working and Persistent Snapshots – 2/3

Pros:

✓ Writes to PM are 2x-5x slower than on volatile RAM!

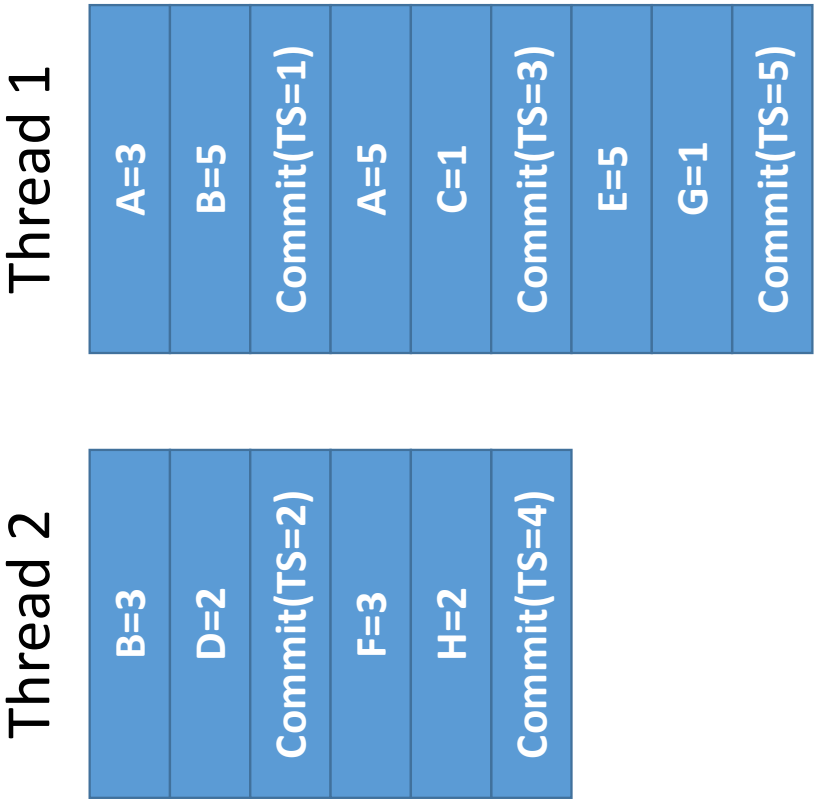
✓ Provides opportunity to filter redundant (duplicate) writes in the log

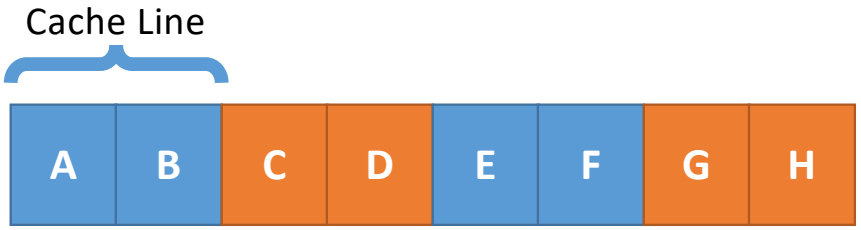
- less writes/flushes === longer life for PM!

Challenge:

- Memory efficiency: avoid maintaining 2 full copies of application's memory

Log filtering





The Checkpoint Process may follow different policies to flush the logs:

- Naïve approach: flush every log entry:
  - **Forward No Filtering (FNF)**
- Replay all writes but flush each updated cache line only once:
  - **Forward Flush Filtering (FFF)**
- Scan logs backwards and write/flush only most recent update:
  - **Backward Filtering Checkpointing (BFC)**

NV-HTM: Working and Persistent Snapshots – 2/3

Pros:

✓ Writes to PM are 2x-5x slower than on volatile RAM!

✓ Provides opportunity to filter redundant (duplicate) writes in the log

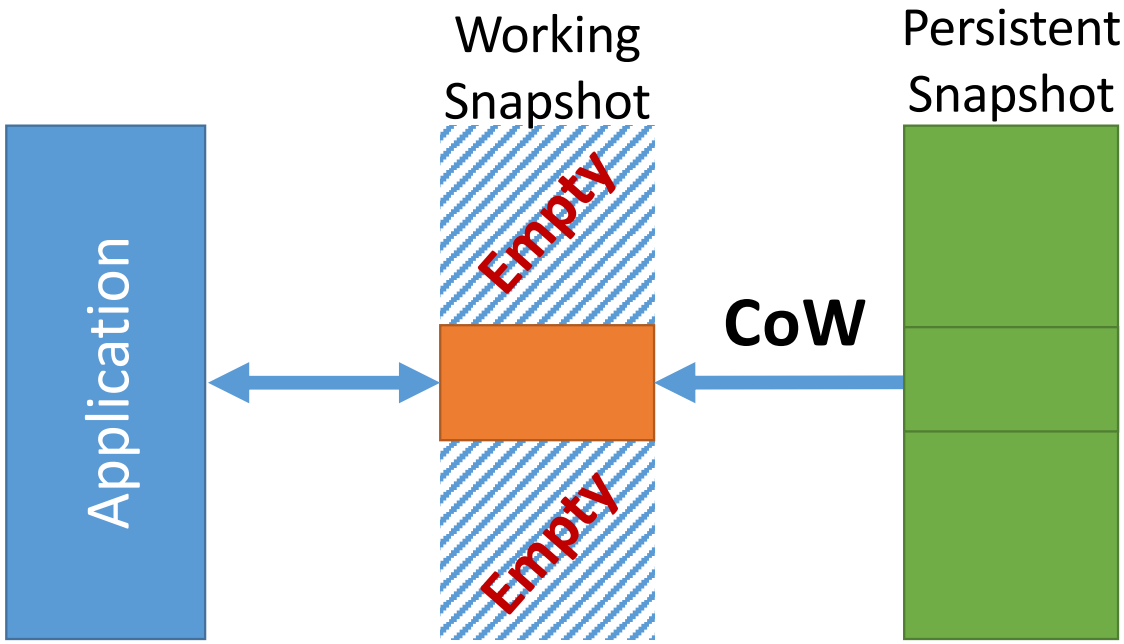
- less writes/flushes === longer life for PM!

Challenge:

- Memory efficiency: avoid maintaining 2 full copies of application's memory

Memory efficiency via CoW – 3/3

- Efficient management of working and persistent snapshot via OS/HW-assisted Copy-on-Write mechanism:
  - duplicate on volatile memory only regions actually modified by application



Recovering from a crash

1. Checkpoint Process replays any pending logged transaction
   - Updated persistent snapshot

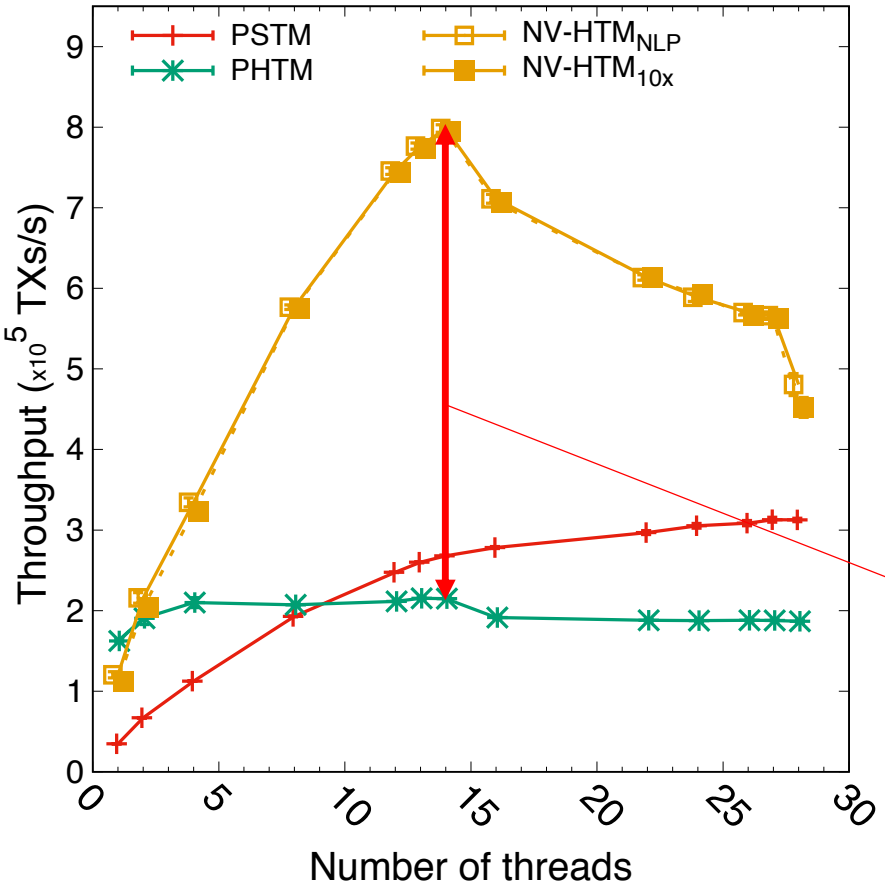
2. Fork the Checkpoint Process:
   - Checkpoint Process mmaps the Persistent Snapshot in shared mode

3. Worker Process mmaps the Persistent Snapshot in private mode
   - Obtains a volatile copy of the Persistent Snapshot (the Working Snapshot)
   - OS ensures Copy-on-Write

Experimental evaluation

- System configuration:
  - 14C/28T TSX enabled Intel Xeon Processor (E5-2648L v4), 22MB L3 cache
  - 32 GB RAM
  - Emulate write to PM latency by spinning 500ns
- Synthetic Benchmark: Bank
- STAMP Benchmark Suit [**IISWC'08**]
- Baselines:
  - PHTM [**DISC'15**]
  - PSTM [**ASPLOS'11**]

STAMP benchmarks



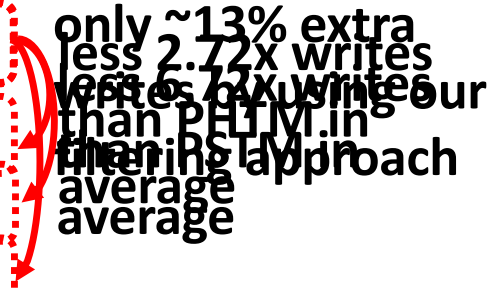
- Comparison for Kmeans (High contention)
- NV-HTM$_{NLP}$: enough capacity for all writes
- NV-HTM$_{10x}$: logs are 1/10 of all writes
  - Checkpoint Manager has minimal impact in throughout

Up to ~4x greater throughput than PHTM

STAMP benchmarks

	Vacation (high)		Kmeans (high)		Yada	
	writes	flushes	writes	flushes	writes	flushes
NV-HTM$_{NLP}$	12.62	4.601	27.00	8.249	18.36	6.016
NV-HTM$_{10\times}$	13.06	4.798	27.03	8.253	25.55	10.10
PHTM	55.01	10.84	45.99	5.000	77.44	14.91
PSTM	174.4	98.08	198.1	125.0	68.69	45.61

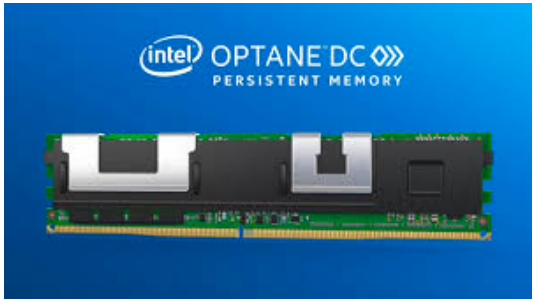
Average Writes and Flushes per transaction



- In average, NV-HTM$_{x10}$ produces 2.72x less writes than PHTM and 6.72x less than PSTM, while only producing 13% more writes than NV-HTM$_{NLP}$

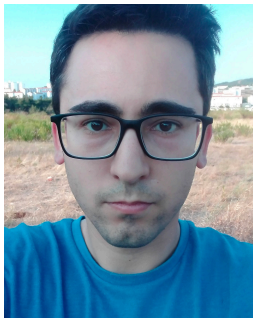
Ongoing work/opportunities of collaboration

- NV-HTM introduces a serial step in commit phase:
  - Waiting for previous transactions to be durably committed, before a new transaction can be durably committed
    - Latency for flushing commit marker is on critical path of execution
    - Can limit throughput especially if NVM latency is high
  - Ongoing work on how to bypass this limitation



- Intel has finally made NVM commercially available
  - Every previous work was based on simulation…
  - Need to reassess actual performance on realistic system

Roadmap

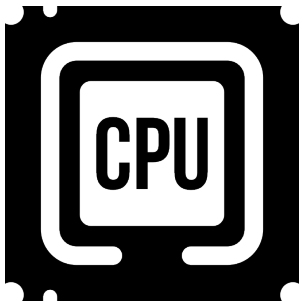


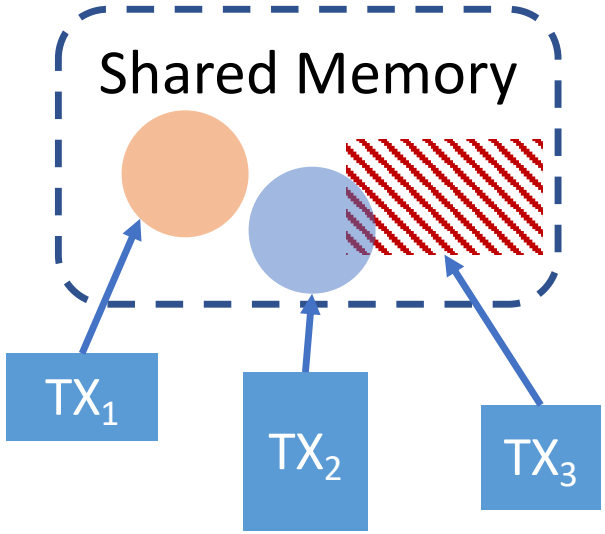
- About me
- About IST & INESC-ID
- An overview of my past research activities
- Current research lines:
  - Transactional Memory & emerging HW technologies:
    - Persistent Memory
    - GPUs
  - Leveraging Symbolic Execution for Distributed Transactional Systems
  - Parallel/distributed platforms for Machine Learning

Transactional Memory

CPU TM

- Mature research
- Widely available in:
  - **Software**
  - **Hardware**
  - **combinations thereof**



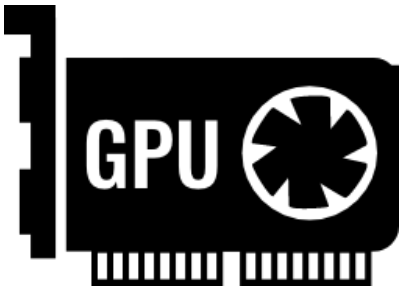


HeTM

Transactional Memory
for CPU+GPU systems

GPU TM

- More recent
- Adapted for GPUs
  - Highly parallel architecture
  - Threads execute lockstep



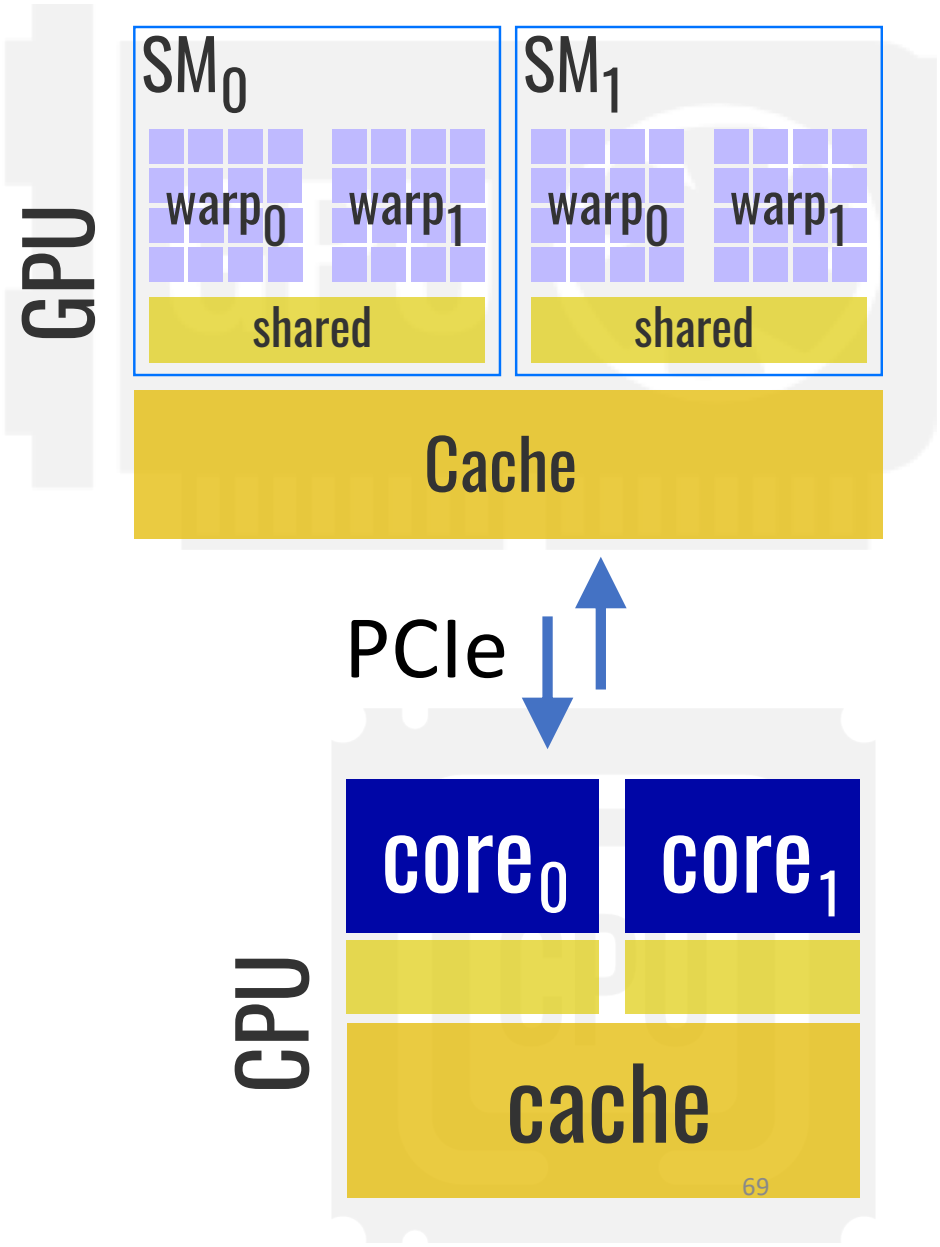
Challenges

Existing TM implementations rely on fast intra-device communication

Serial inter-device communication makes fine-grained synchronization difficult

Need to revisit the TM abstraction and consistency criteria

Build a system upon this new abstraction



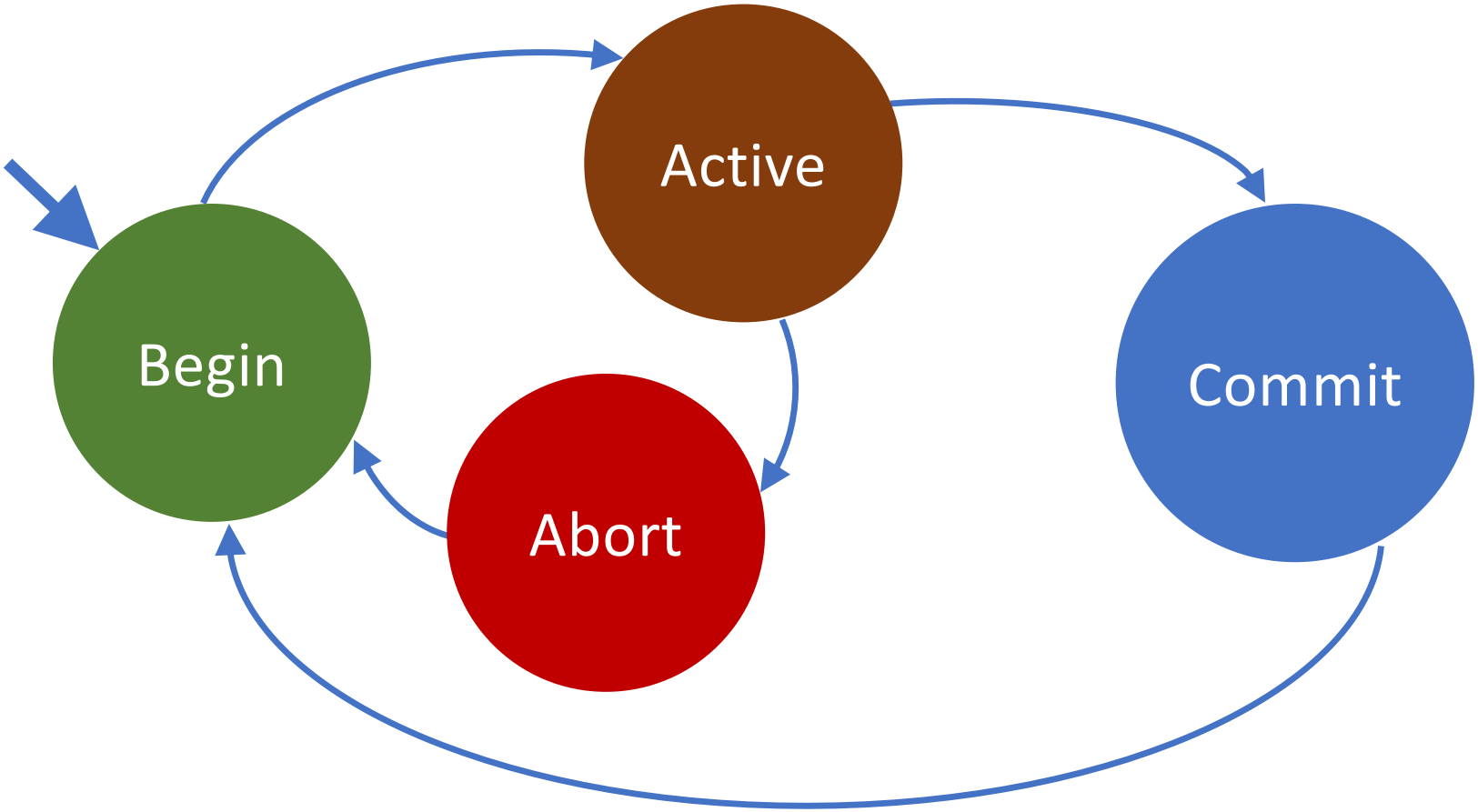
Correctness guarantee for traditional TM

P1. *The behavior of every **committed transaction** has to be justifiable by the same sequential execution containing only committed transactions, without contradicting real-time order.*

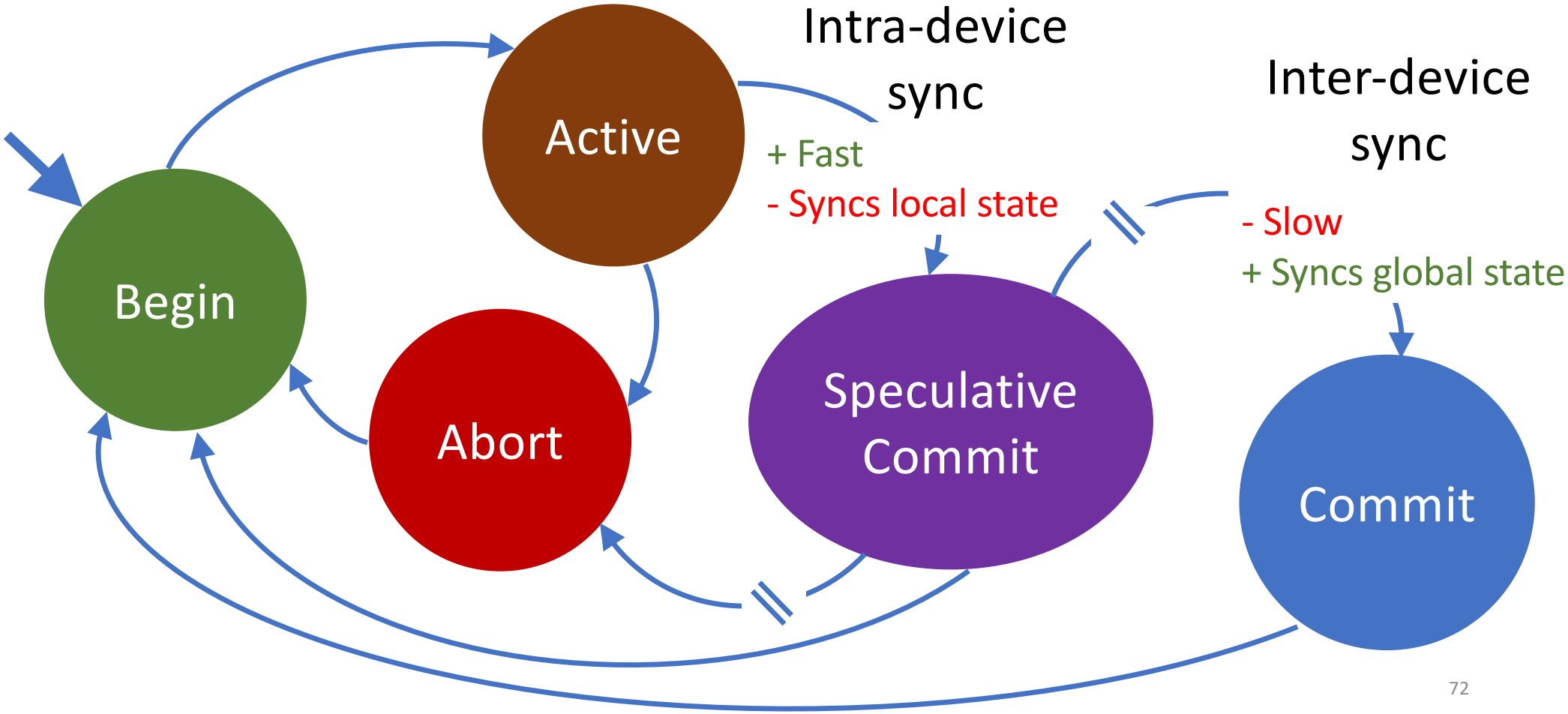
P2. *The behavior of any **active transaction**, even if it eventually aborts, has to be justifiable by some sequential execution (possibly different) containing only committed transactions.*

Hard notion of **committed transaction**:
need to transfer single transaction metadata over PCIe

Correctness guarantee for traditional TM



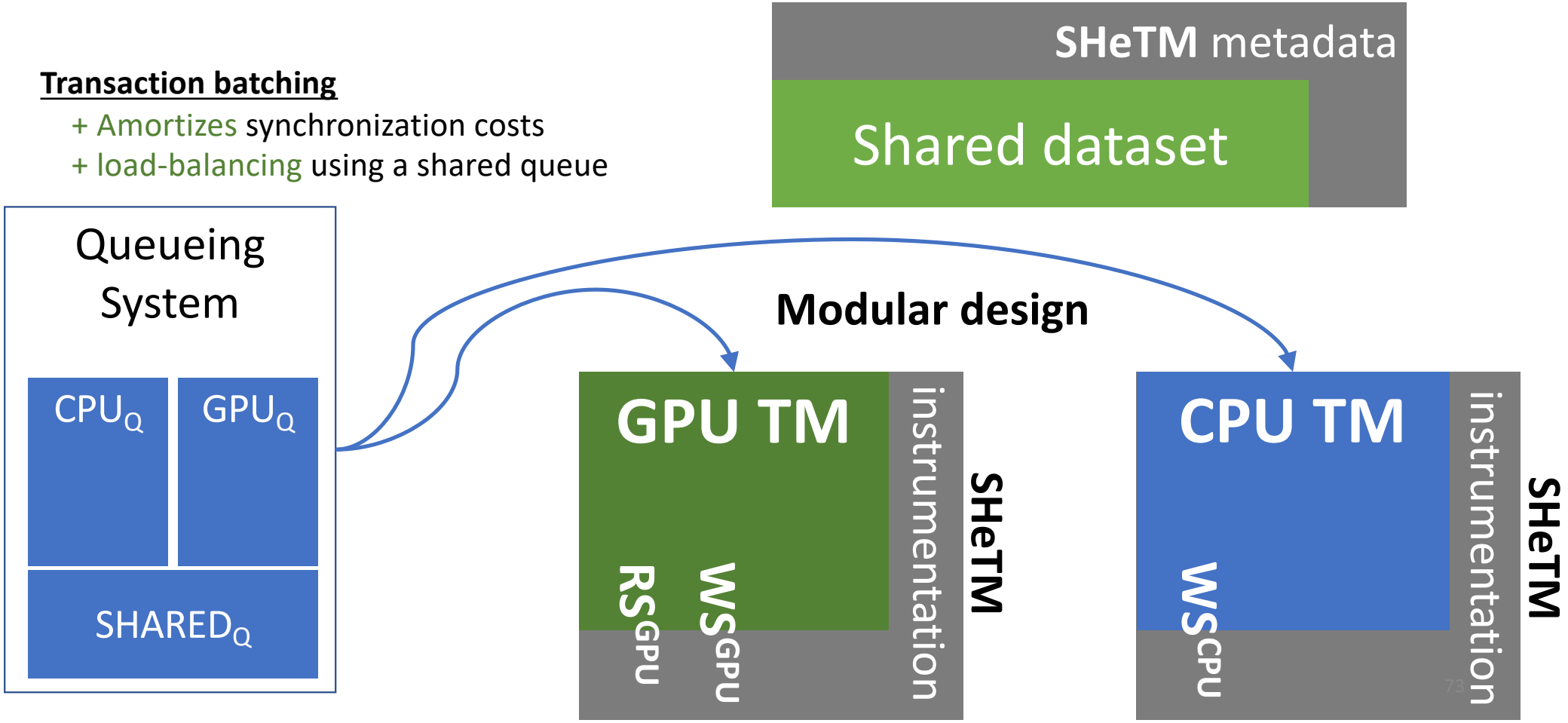
Correctness guarantee for HeTM



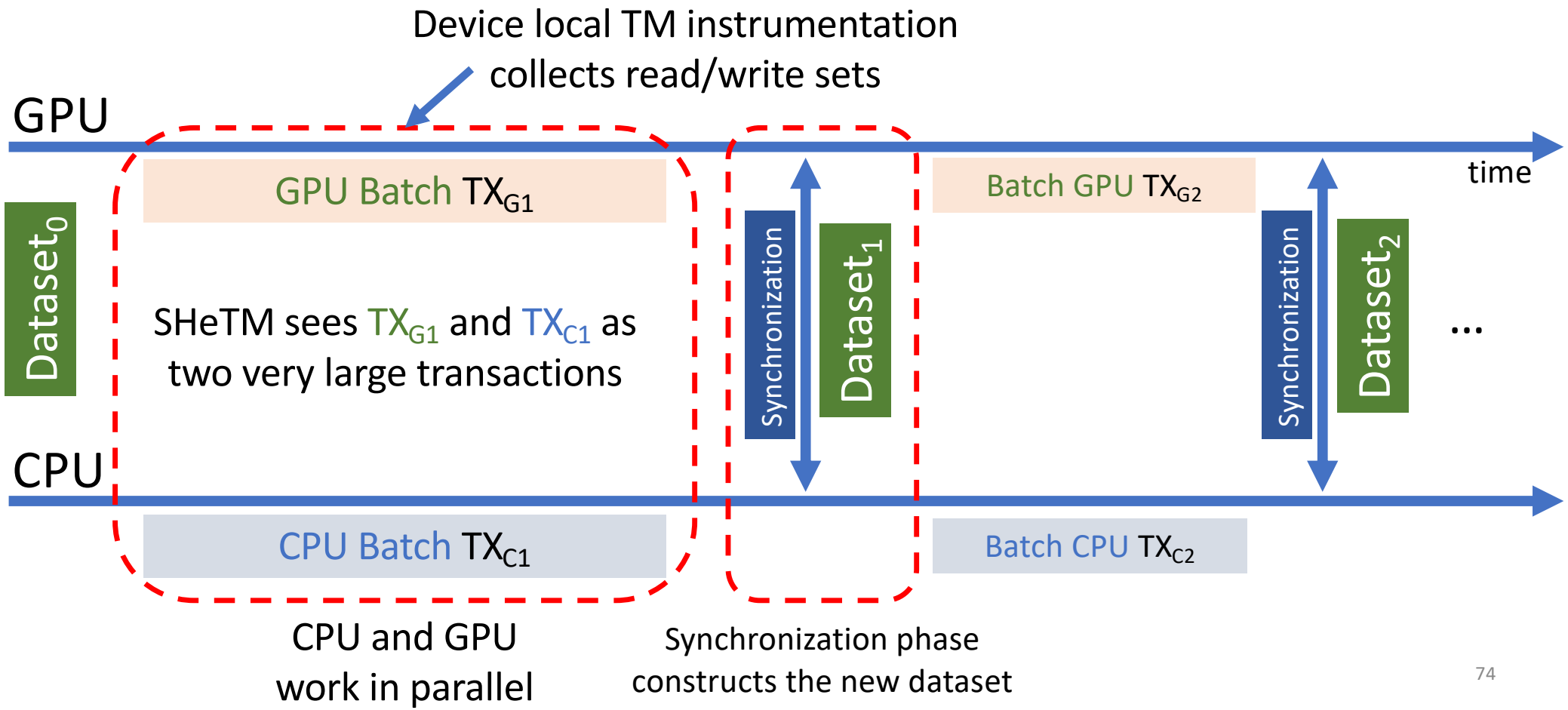
Speculative HeTM (SHeTM): architecture

Transaction batching

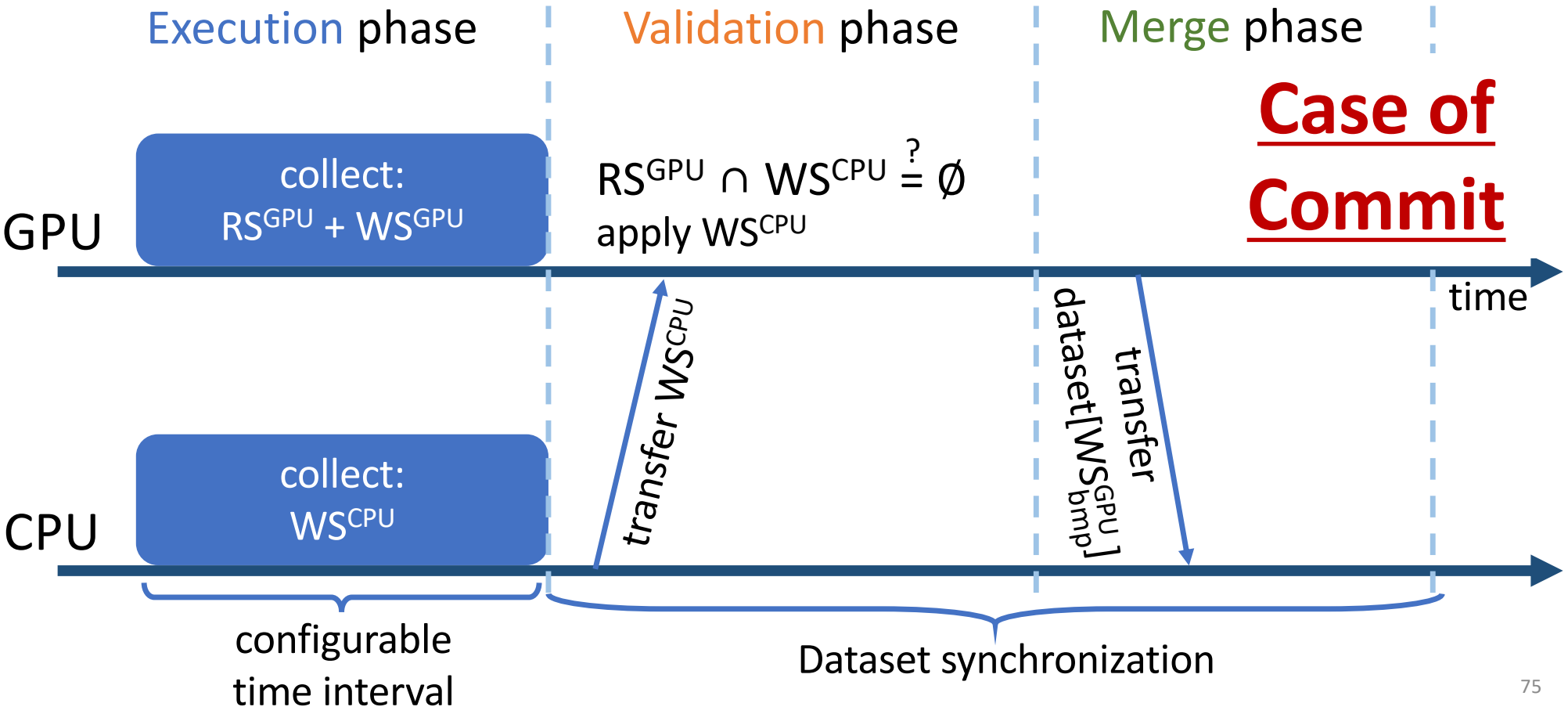
+ Amortizes synchronization costs
+ load-balancing using a shared queue



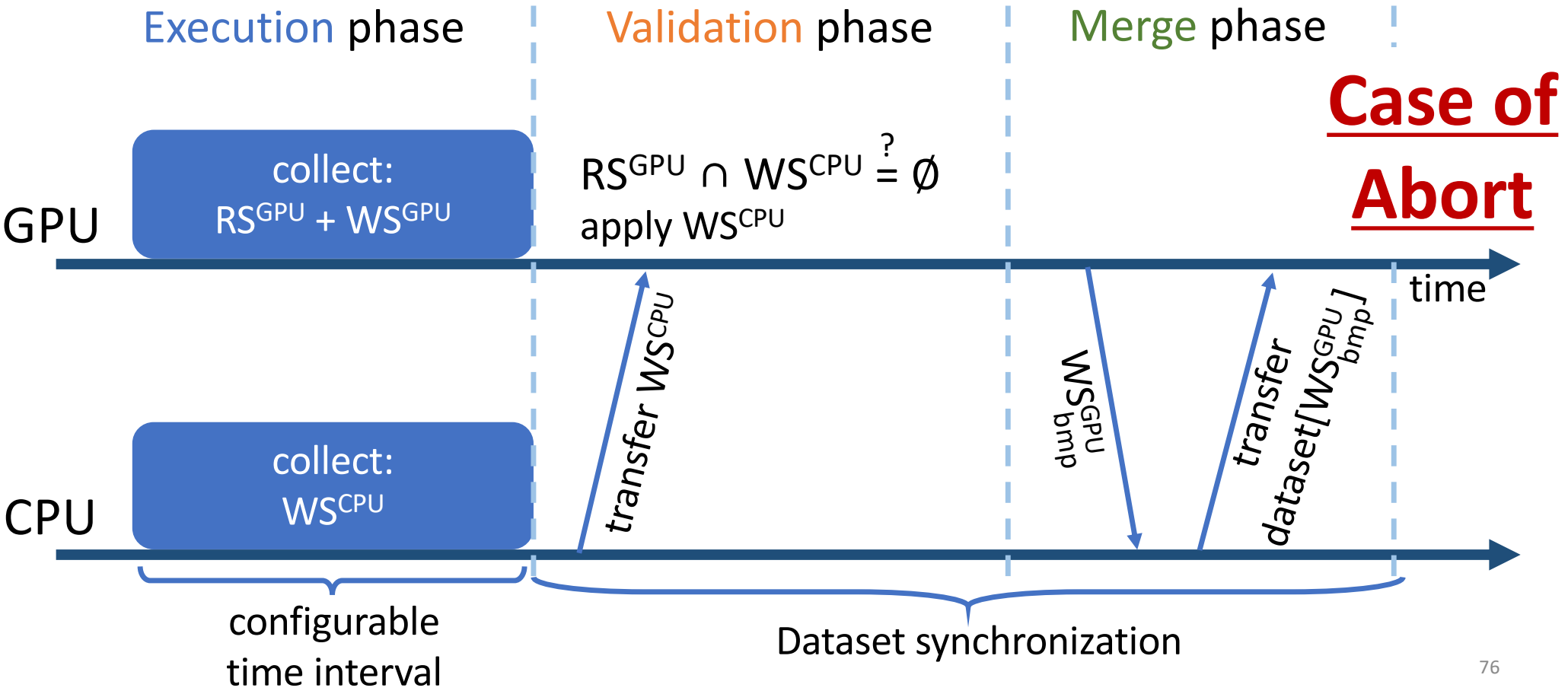
Speculative HeTM (SHeTM): overview



Base (unoptimized) idea

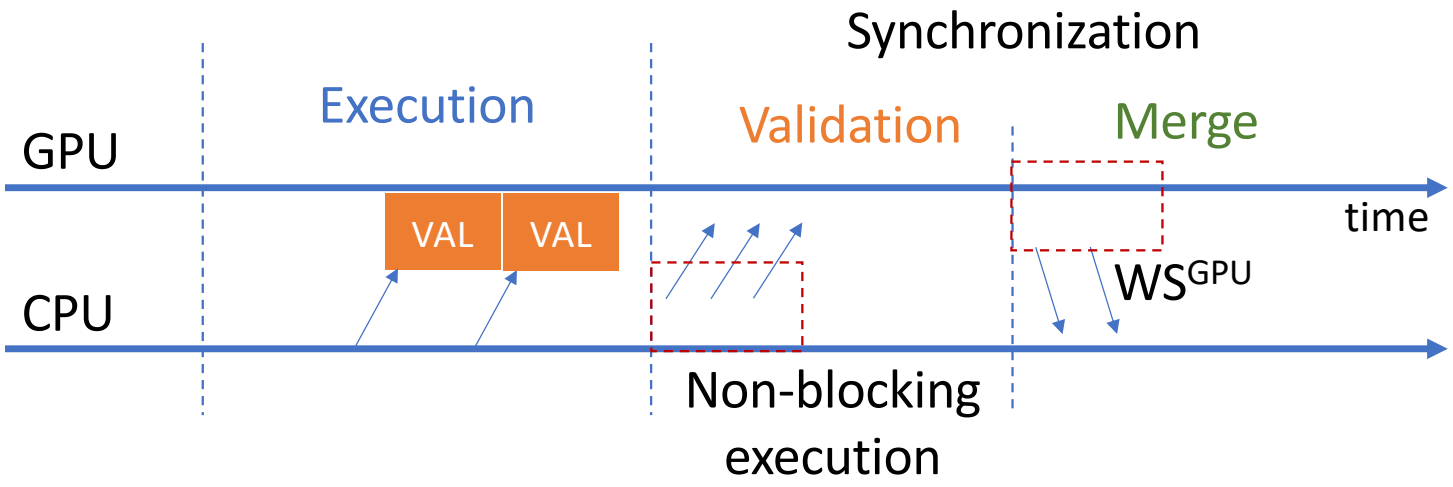


Base (unoptimized) idea



Optimizations

- Synchronization imposes significative overheads!
- Some optimizations:
  - Early validation kernels may reduce wasted work
  - Execution of transactions can be overlapped with synchronization stages



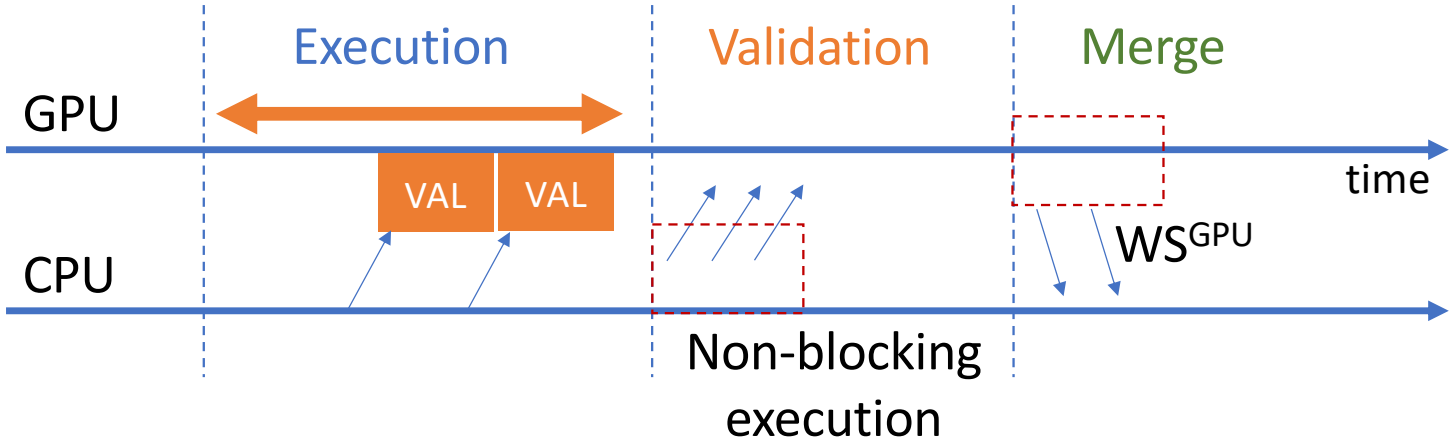
Details in the paper

Evaluation

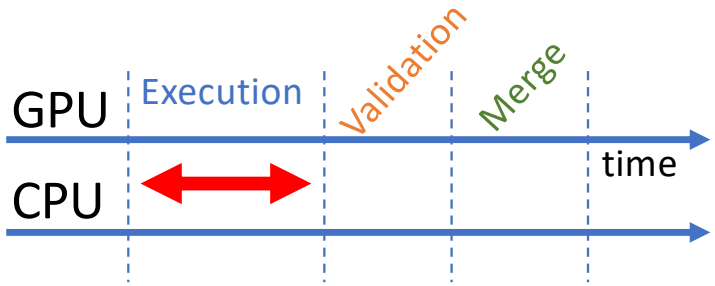
- Intel Xeon E5-2648L v4 (14C/28T, HTM, 32GB DRAM)
- Nvidia GTX 1080 (8GB XDDR5, driver 387.34, CUDA 9.1)
- CPU TM:
  - Intel's hardware TM implementation (TSX)
  - TinySTM in the paper
- GPU TM:
  - PR-STM [**EuroPar'15**]
- Synthetic benchmark
  - Random memory accesses on array of integers
- MemcachedGPU-TM
  - Popular web caching application

Synthetic benchmark

- Evaluate the impact of the duration of the Execution phase
  - Overhead of synchronization
- Benefits of two main optimizations
  1. Early validation
  2. Overlapping execution and synchronization

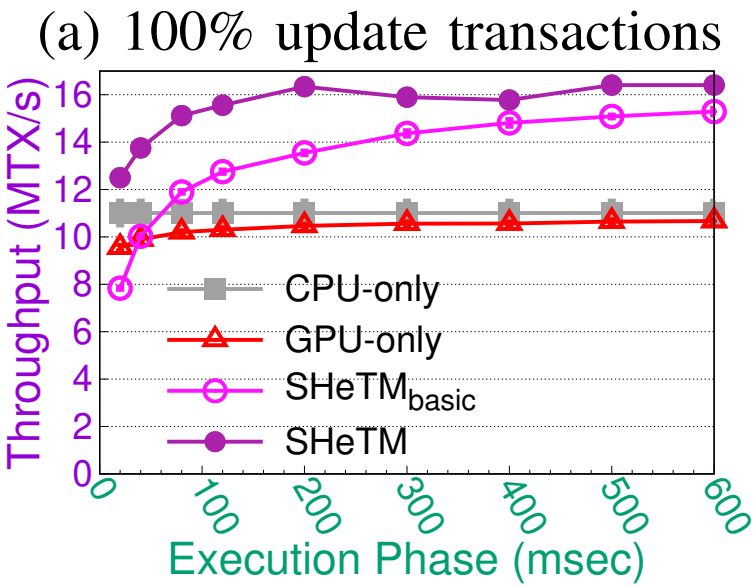


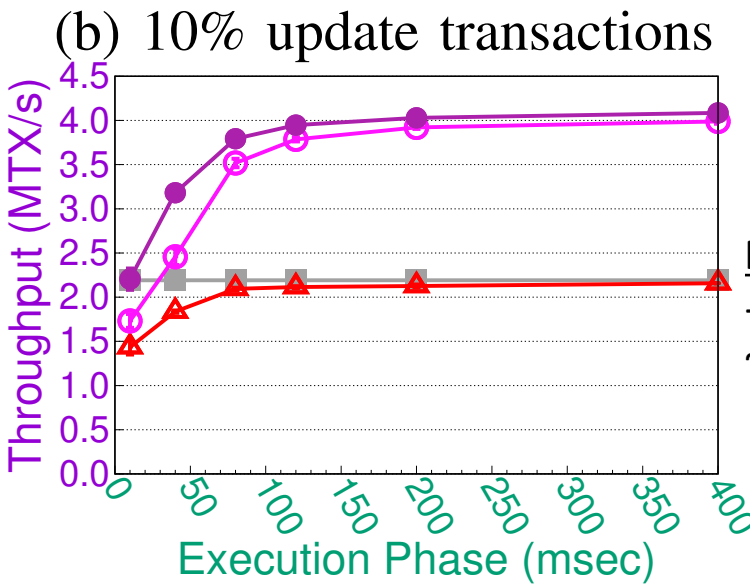
Synthetic benchmark – Execution time



In this experiment:

- no inter-devices conflicts (stresses the overheads of commit batches)

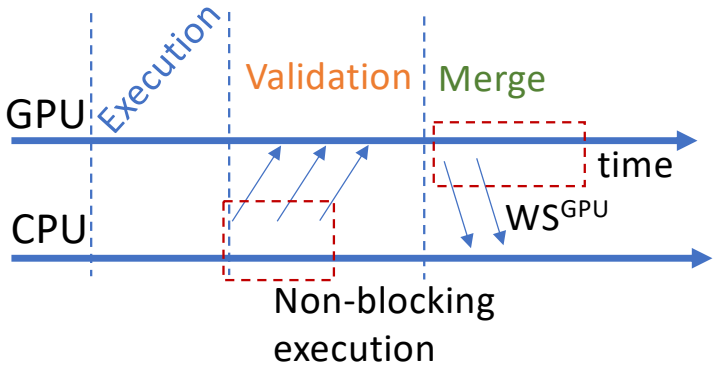




Write intensive workloads:
- stress more SHeTM
still only ~25% below sum CPU+GPU performance

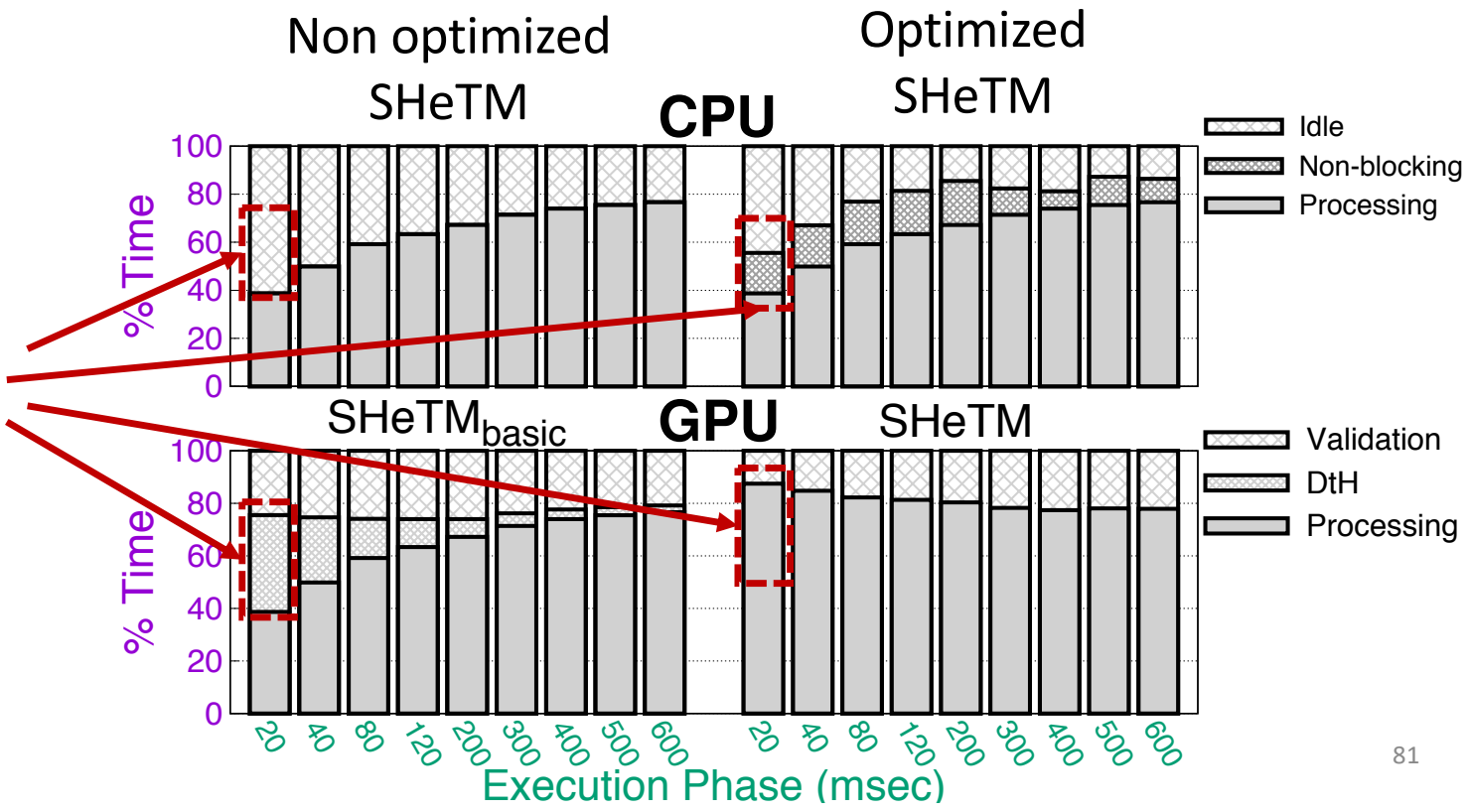
Read intensive workloads:
+ SHeTM throughput is ~95% the sum CPU+GPU

Synchronization overlapping



Significative reduction on CPU and GPU idle time:

- CPU: 60% ➔ 45%
- GPU: 60% ➔ 20%

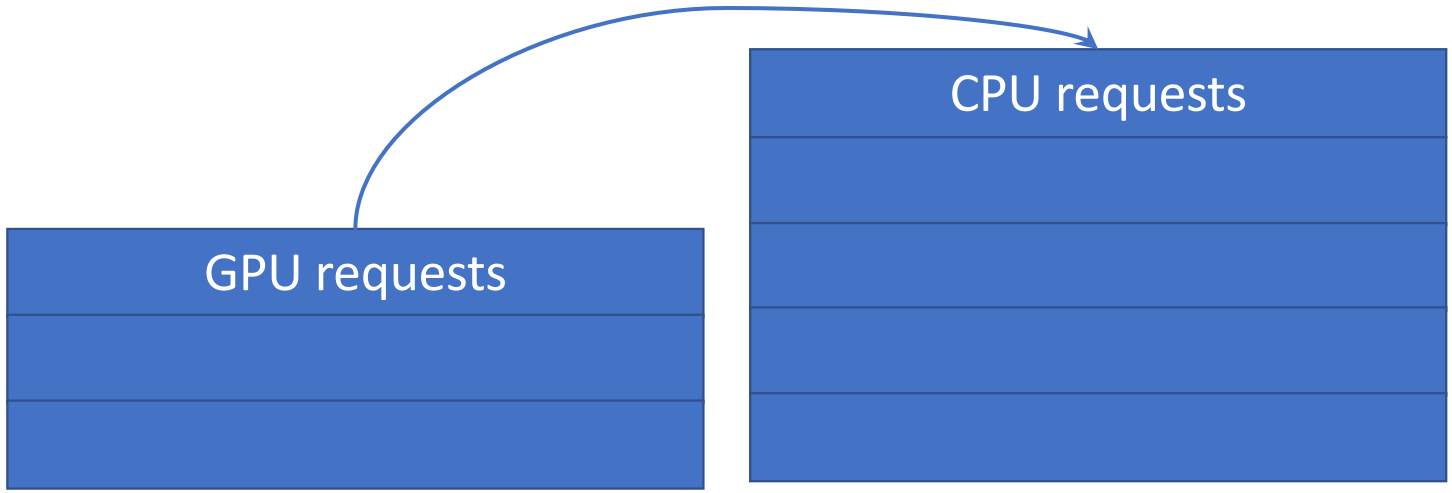


MemcachedGPU-TM

- Popular object caching system built by Facebook
- [**SoCC'15**]: port of Memcached to GPU
  - Complex lock-based scheme that unnecessarily restricts concurrency
- Workload:
  - 99.9% of GETs and key frequency follow a Zipfian distribution ($\alpha = 0.5$)
  - Keys partitioned based on last bit:
    - Odd keys ➔ GPU; Even keys ➔ CPU
  - Emulate load unbalances:
    - vary the popularity of keys maintained by GPU and CPU
    - GPU steals CPU requests (non-zero probability of conflicting in a key)

MemcachedGPU-TM

- Emulate load unbalances:
  - vary the popularity of keys maintained by GPU and CPU
  - GPU steals CPU requests (non-zero probability of conflicting in a key)



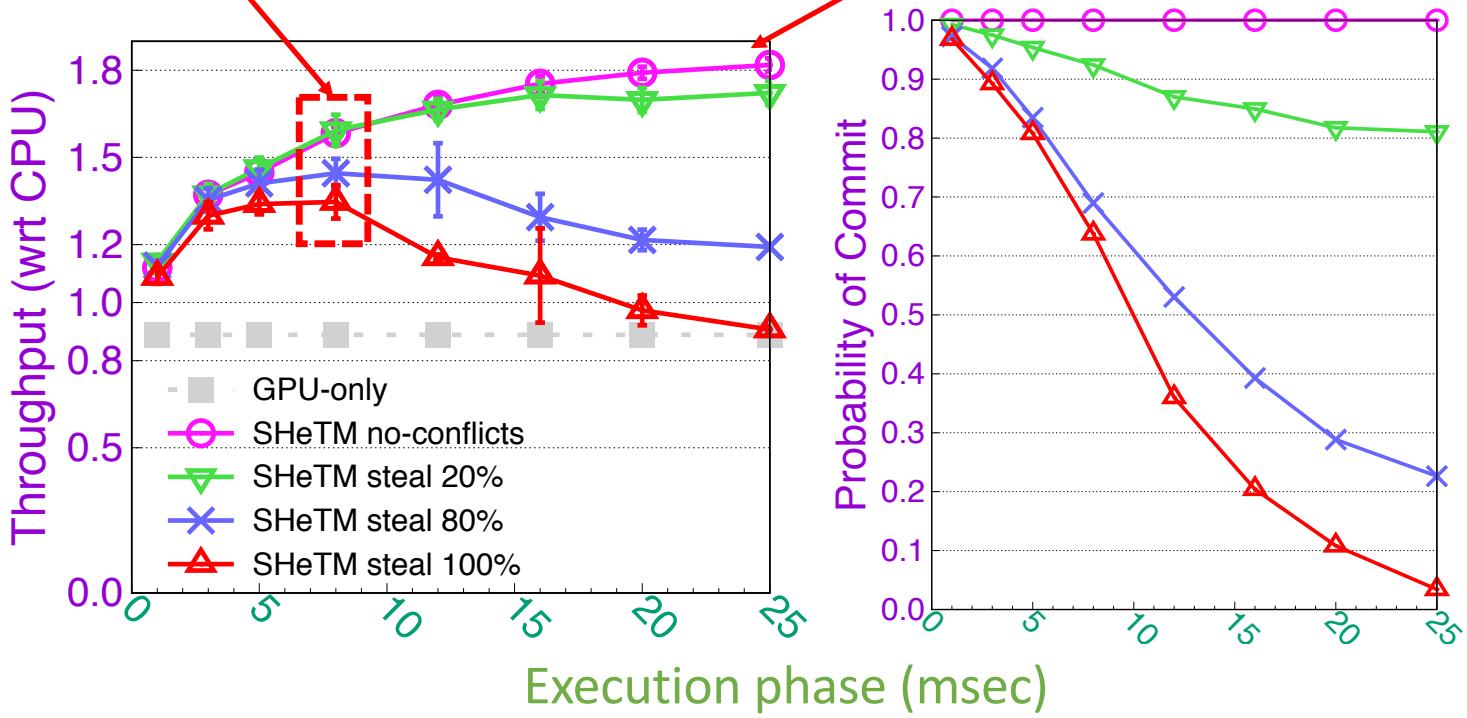
GPU Steal with probability X% (X=100% means that GPU operates only on the keys assigned to CPU)

The higher the “steal” probability, the higher the inter-device contention probability

MemcachedGPU-TM

Tuning the durations allows high contention workloads to still benefit from CPU+GPU

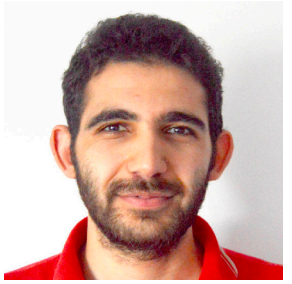
overhead is ~10% in absence of contention



Ongoing work/opportunities of collaboration

- Extend SHeTM to support multiple GPUs
- Exploit integrated GPUs to accelerate STMs
- Design of STMs for GPUs

Roadmap

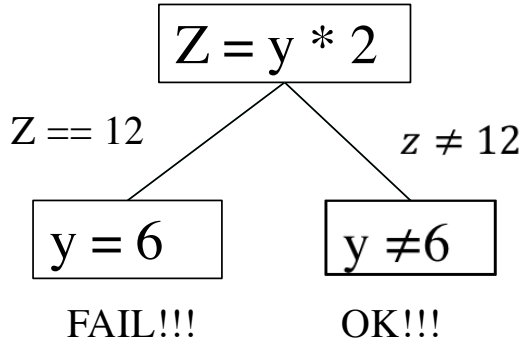


- About me
- About IST & INESC-ID
- An overview of my past research activities
- Current research lines:
  - Transactional Memory & emerging HW technologies:
    - Persistent Memory
    - GPUs
  - Leveraging Symbolic Execution for Distributed Transactional Systems
  - Parallel/distributed platforms for Machine Learning

Symbolic Execution

Typical usage: testing/verification

```
 1 int f() {
 2   ...
 3   y = read();
 4   z = y * 2;
 5   if (z == 12) {
 6     fail();
 7   } else {
 8     printf("OK");
 9   }
10 }
```



Symbolic execution of transactional programs

Data access prediction

```
public void buy_umbrella(int client_id, int input){
        if(input>=0 && input <=2){
                int umbrella_id = input*2;
                int price = kv.get(umbrella_id);
                kv.put(umbrella_id,price*2);
        }else if(input <= NUM_RECORDS){
               int umbrella_id = input*5;
                int price = kv.get(umbrella_id);
                kv.put(umbrella_id,price*5);
        }
```

→ Accesses *umbrella_id* 0, 2 and 4

→ Accesses *umbrella_id* 15, 20,25…NUM_RECORDS *5

Possible applications & collaboration opportunities

- A priori-knowledge of Read&Write-set of txs opens a number of interesting opportunities
  - Scheduling
  - Deterministic concurrency control (State Machine Replication)
  - Automatic data partitioning schemes
  - …

Challenges

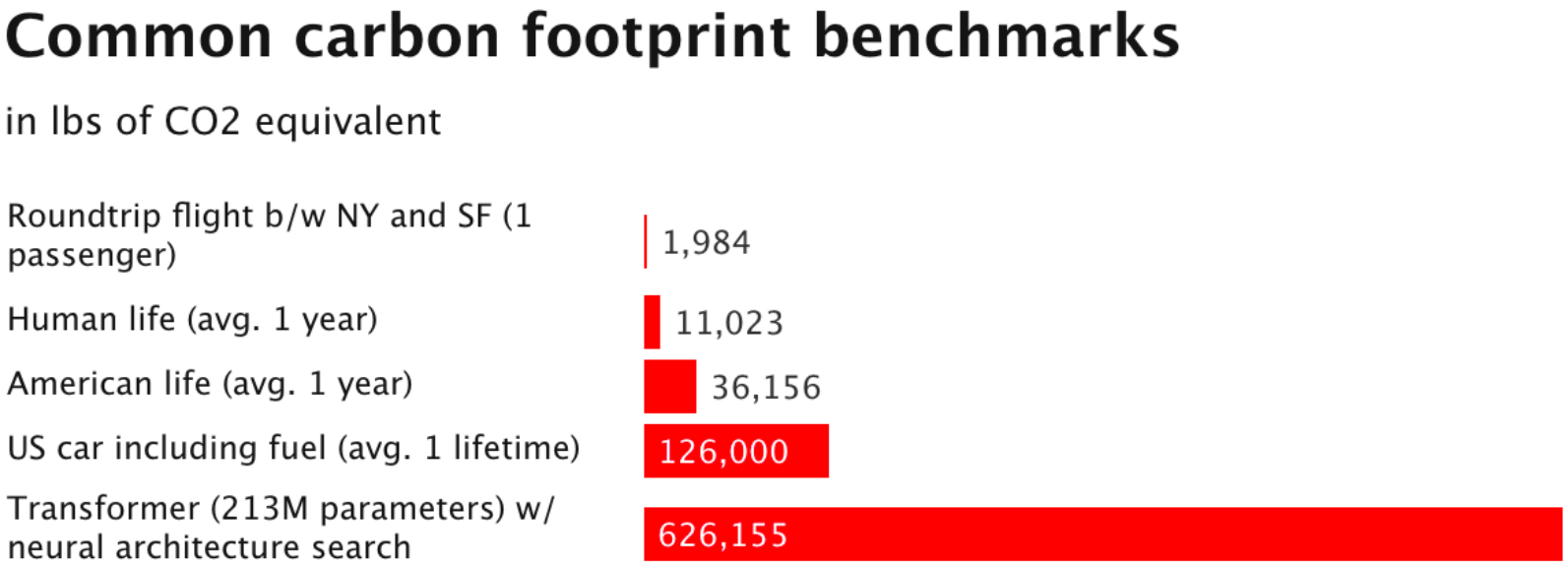
- State explosion:
  - SE is sound but not complete (halting problem)

- If used prior to program execution, SE suffers of limitations of static analysis techniques
  - What if program behavior depends on the DB's state?
    - Over-approximation
    - Combine SE && run-time execution

Roadmap

- About me
- About IST & INESC-ID
- An overview of my past research activities
- Current research lines:
  - Transactional Memory & emerging HW technologies:
    - Persistent Memory
    - GPUs
  - Leveraging Symbolic Execution for Distributed Transactional Systems
  - Parallel/distributed platforms for Machine Learning

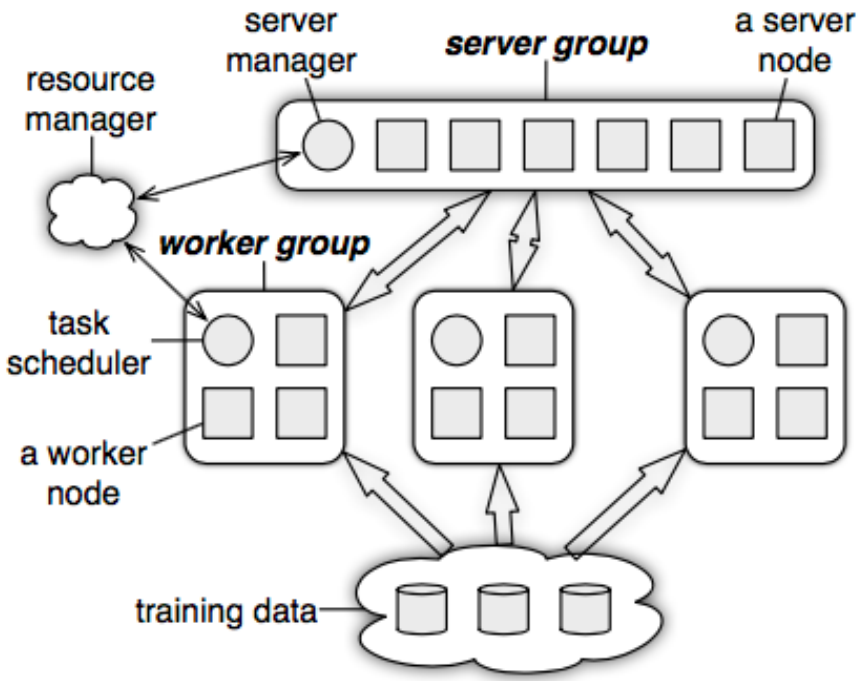
"Training a single AI model can emit as much carbon as five cars in their lifetimes (and that includes manufacture of the car itself)" [ACL'19]



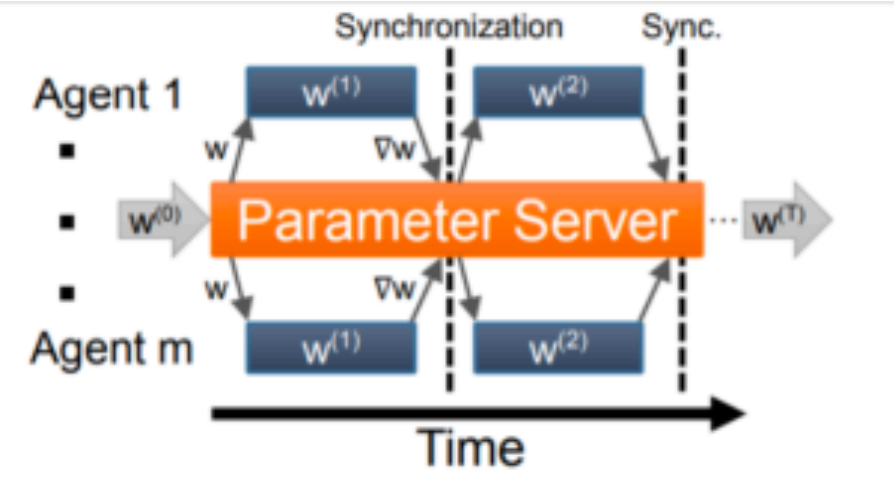
The estimated costs of training a model

	Date of original paper	Energy consumption (kWh)	Carbon footprint (lbs of CO2e)	Cloud compute cost (USD)
Transformer (65M parameters)	Jun, 2017	27	26	$41–$140
Transformer (213M parameters)	Jun, 2017	201	192	$289–$981
ELMo	Feb, 2018	275	262	$433–$1,472
BERT (110M parameters)	Oct, 2018	1,507	1,438	$3,751–$12,571
Transformer (213M parameters) w/ neural architecture search	Jan, 2019	656,347	626,155	$942,973–$3,201,722
GPT-2	Feb, 2019	-	-	$12,902–$43,008

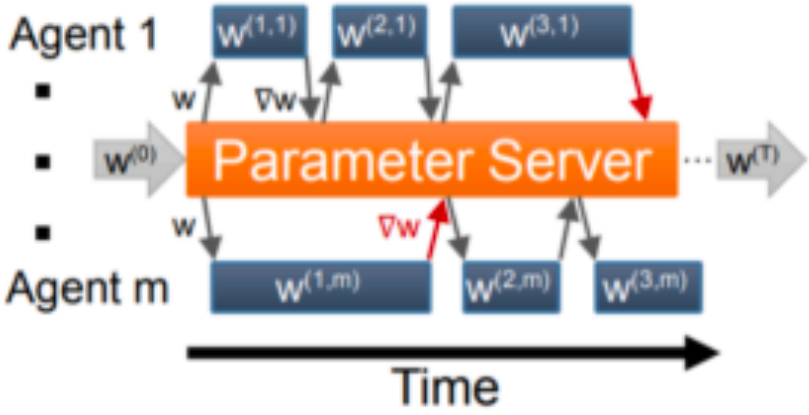
Typical architecture of ML Platforms a.k.a. Parameter Server



To synchronize or not to synchronize?



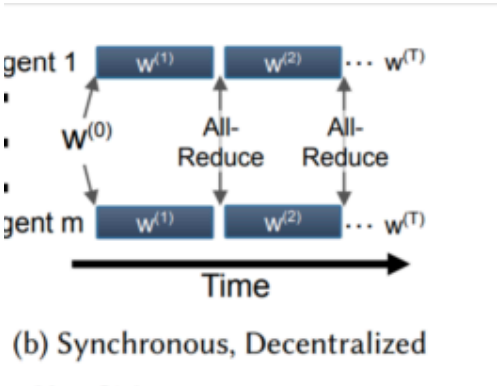
(a) Synchronous, Parameter Server



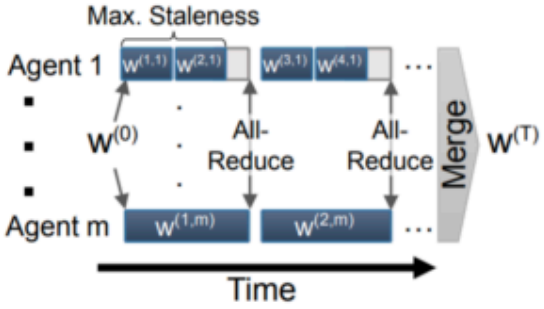
(c) Asynchronous, Parameter Server

Other training related design choices/parameters

- How many parameter servers/worker nodes?
  - Extreme settings: fully decentralized (1 to 1)



(b) Synchronous, Decentralized



(d) Stale-Synchronous, Decentralized

- Size of the batch processed by each worker
- Learning rate
- …

Ongoing work & collaboration opportunities

- Understand the system-related trade-offs associated with these design choices
  - …and propose novel approaches to enhance efficiency of state of the art approaches

Ongoing work & collaboration opportunities

- Automate the identification of the “optimal” configuration:
  - Challenges/opportunities:
    - Building black box models of these platforms can be prohibitively expensive
    - Configuration space is huge:
      - Cartesian product of model related and cloud related parameters
    - Techniques to minimize the cost of “testing” configurations
      - Bayesian optimization
      - Sub-sampling
      - Aborting testing of “bad” connfigurations ASAP