Boosting Data Replication in Distributed Transactional Memories

Paolo Romano

inesc id lisboa

September 15 2010, Dipartimento di Informatica e Sistemistica, Sapienza Rome University

About me

- Master (2002) and PhD (2007) from Rome University “La Sapienza”
- Member of the OASIS WS-Reliability Technical Committee (2003-2004)
- Researcher & Lecturer (2007-2008) at Rome University “La Sapienza”
- Researcher at the Distributed Systems Group INESC-ID, Lisbon (since 2008)
- Coordinator of the FCT Aristos Project (Jan 2010-Jan 2012)
  - Bilateral Italian-Portuguese project
  - Autonomic Replication of Transactional Memories
- Coordinator of the FP7 Cloud-TM Project (Jun 2010-Jun2012)
  - 4 international partners from industry and academy
  - Self-tuning, Distributed Transactional Memory platform for the Cloud
- Coordinator of the Cost Action Euro-TM (fall 2010-fall 2013)
  - Pan-European Research network on Transactional Memories
  - 56 experts, 42 institutions, 12 countries

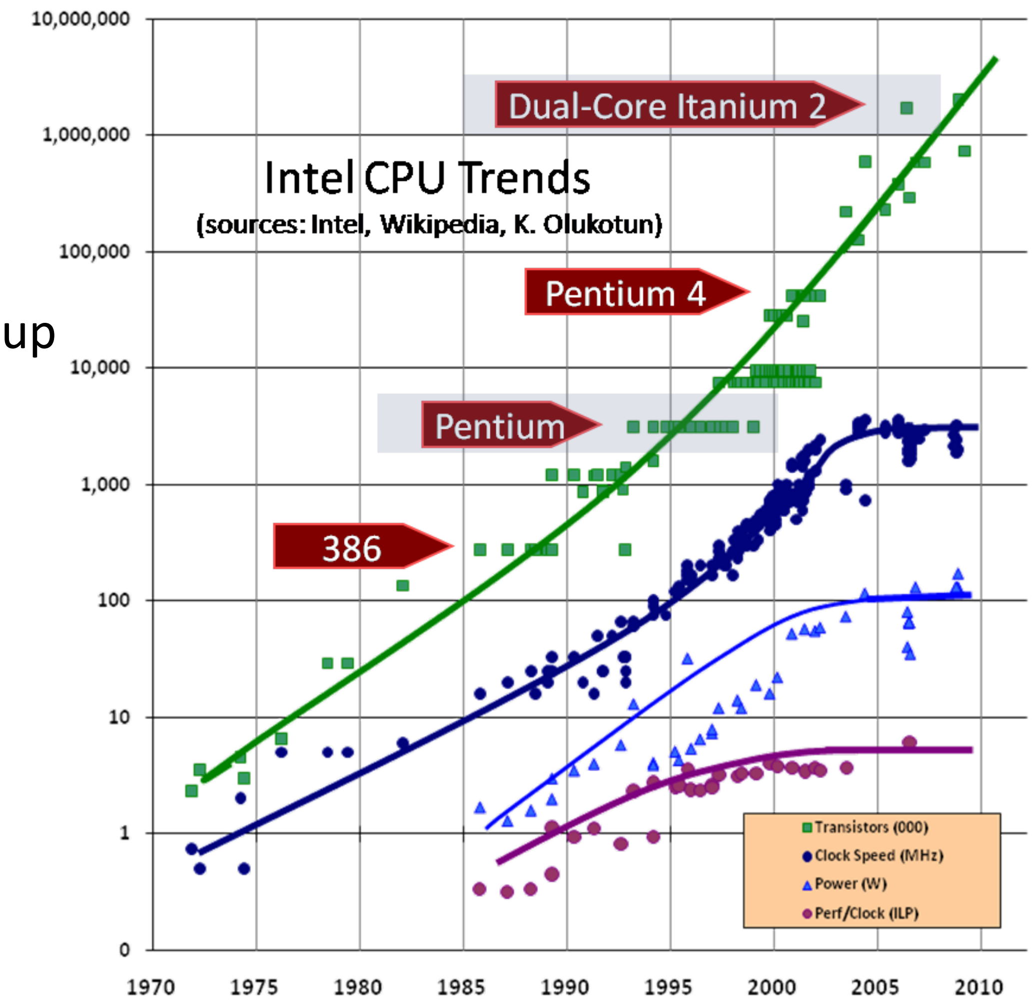
Roadmap

- Transactional Memories (TM)
- Distributed Transactional Memories (DTM)
- Data Replication in DTM
  - State of the Art of transactional replication
  - new challenges of DTMs…
  - …and two new protocols:
    - Asynchronous Lease Certification
    - Speculative Transaction Replications

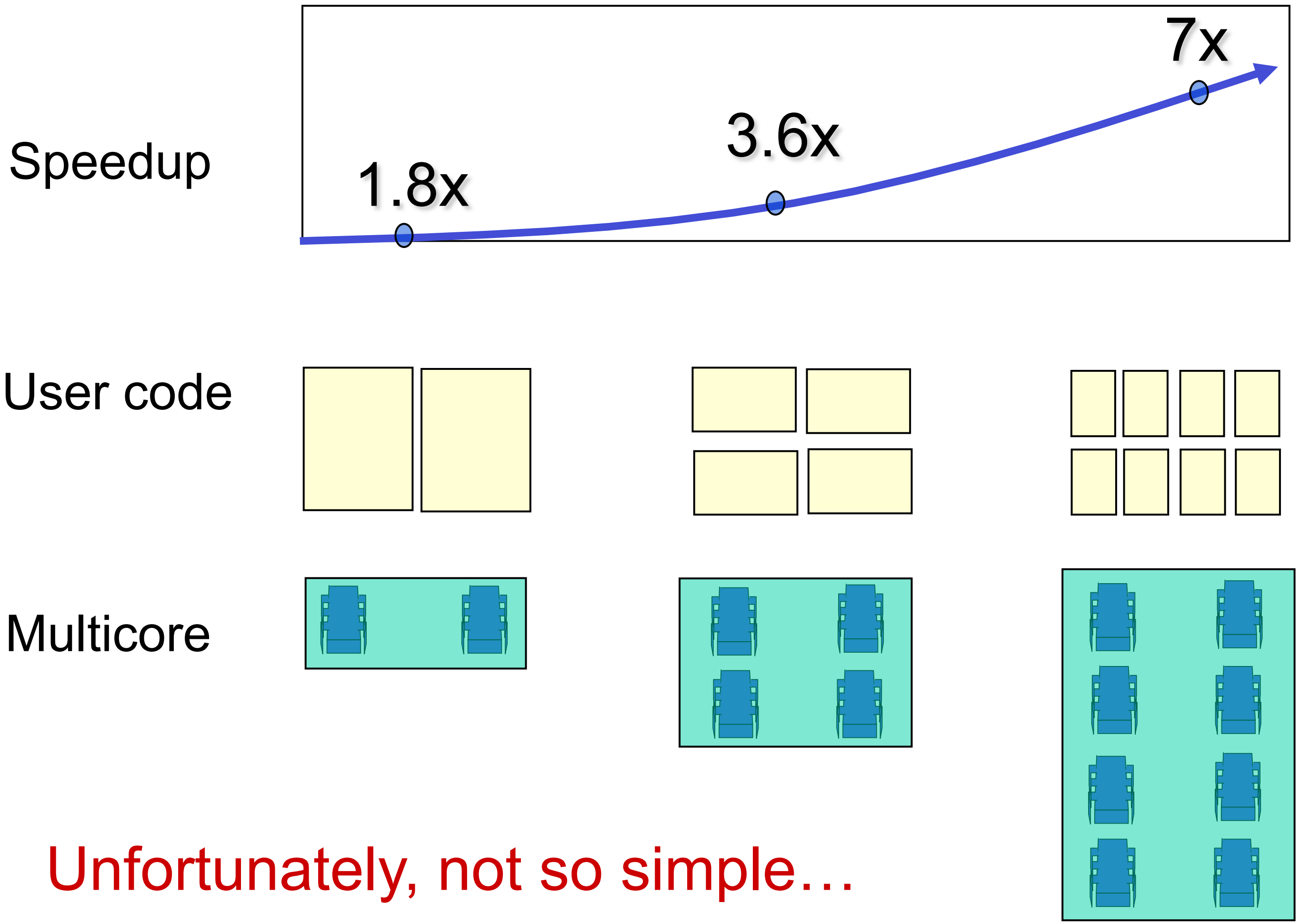
The era of free performance gains is over

- Over the last 30 years:
  - new CPU generation = free speed-up
- Since 2003:
  - CPU clock speed plateaued...
  - but Moore's law chase continues:
    - Multi-cores, Hyperthreading...

FUTURE IS PARALLEL

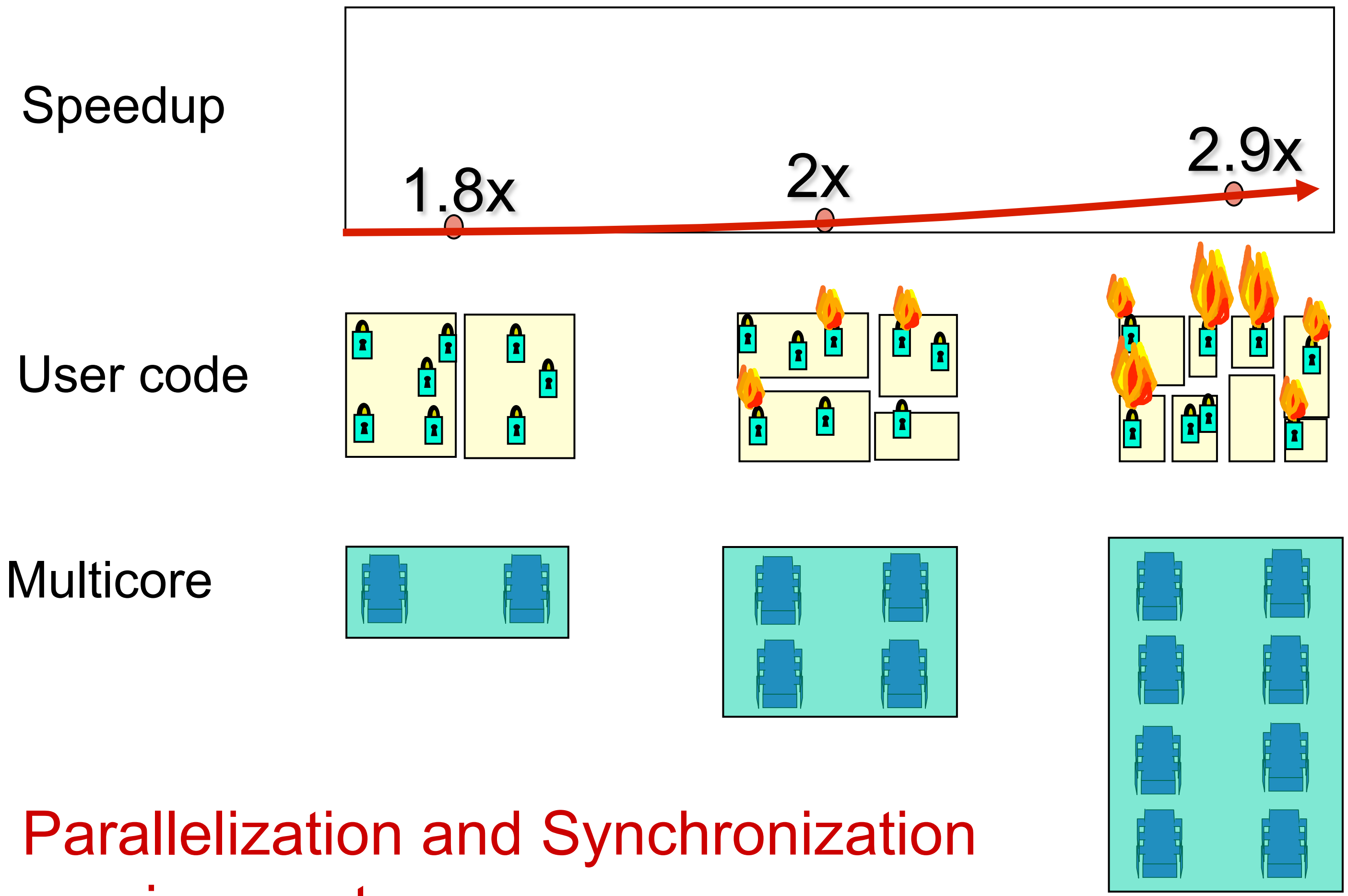


Multicore Software Scaling



Unfortunately, not so simple…

Real-World Multicore Scaling



Parallelization and Synchronization require great care…

Coarse grained parallelism?
simple but does not scale

Amdahl's Law:
Speedup = 1/(ParallelPart/N + SequentialPart)

Pay for N = 128 cores
SequentialPart = 25%

As num cores grows the effect of 25% becomes more accute
2.3/4, 2.9/8, 3.4/16, 3.7/32….

Fine grained parallelsim?
easier to say than to do

- Simple grained locking is a **conundrum:**
  - need to reason about deadlocks, livelocks, priority inversions:
    - complex/undocumented lock acquistion protocols
    - scarce composability of existing software modules

  ... and a **verification nightmare:**
    - subtle bugs that are extremely hard to reproduce

- Make parallel programming **accessible to the masses!**

Transactional memories

- Key idea:
  - hide away synchronization issues from the programmer
  - replace locks with atomic transactions:
    - avoid deadlocks, priority inversions, convoying
    - way simpler to reason about, verify, compose
    - deliver performance of hand-crafted locking via speculation (+HW support)

- Brief historic overview:
  - Original idea dating back to early 90s
  - Largely neglected until advent of multi-cores (~2003)
  - Today among the most relevant research topics in the areas of:
    - Computer architecture
    - Programming Languages
    - Operating Systems
    - Distributed Computing

} **STRONG INTERDISCIPLINARITY**

TMs: where we are, challenges, trends

- Theoretical Aspects
  - formalization of adequate consistency guarantees, performance bounds
- Software-based implementations (STM)
  - performance/scalability improving, but overhead still unsatisfactory
- Hardware support
  - very promising simulation-based results, but no support in commercial processors
- Language integration
  - advanced supports (parallel nesting, conditional synchronization) are appearing...
  - ...but lack of standard APIs & tools hampers industrial penetration
- Operating system support
  - still in its infancy, but badly needed (conflict aware scheduling, transactional I/O)
- Recent trends:
  - shift towards distributed environments to enhance scalability & dependability

Distributed
Transactional Memories

An obvious evolution

- Real, complex STM based applications are starting to appear:
  - Apache Web Server
  - FenixEDU
  - Circuit Routing
  - …
- …and are being faced with classic production environment's challenges:
  - scalability
  - high-availability
  - fault-tolerance

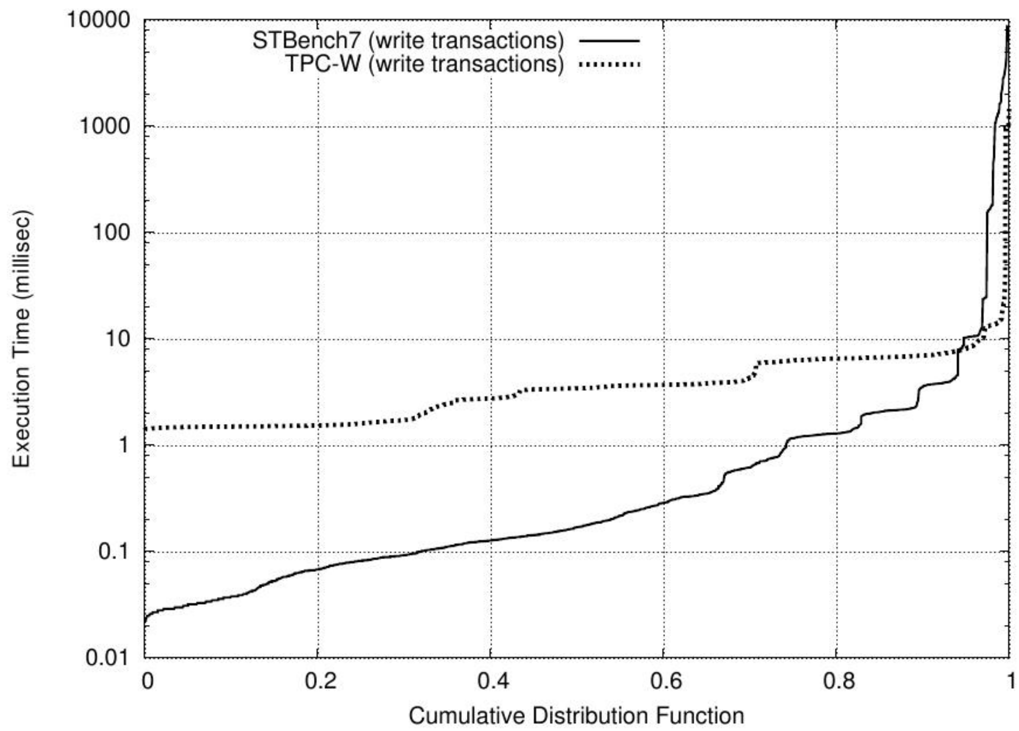
} Distributed STMs

Distributed STMs

- At the convergence of two main areas:

>70% xacts are 10-100 times shorter:

- larger impact of coordination



2. Boost performance by batching any remote synchronization during the commit phase

unique, challenging requirements!

Existing Distributed STMs

- Very recent research area....
- Only a handful of existing prototypes:
  - DMV [PPoPP,2006]
  - DiSTM [ICPP, 2008]
  - ClusterSTM [PPoPP, 2008]

DISTRIBUTION ONLY, NO REPLICATION:
NO SUPPORT FOR FAULT TOLERANCE!

Classic Synchronous Transactional Replication Schemes

Single-master schemes:

- primary runs all write xacts and propagates updates to backups
- backups exec read-only xacts

\+ *simple*

\- *scales poorly with write intensive workloads*

Multi-master schemes:

- all replicas can process both read&write xacts
- locks are acquired during xact's execution or at commit time
- 2PC ensures agreement on the outcome of conflicting transactions (and their atomicity)

\+ *better load balancing & scalability*

\- *high latency for intra-transaction lock acquisition*

\- *distributed deadlocks grow cubically with #nodes:*

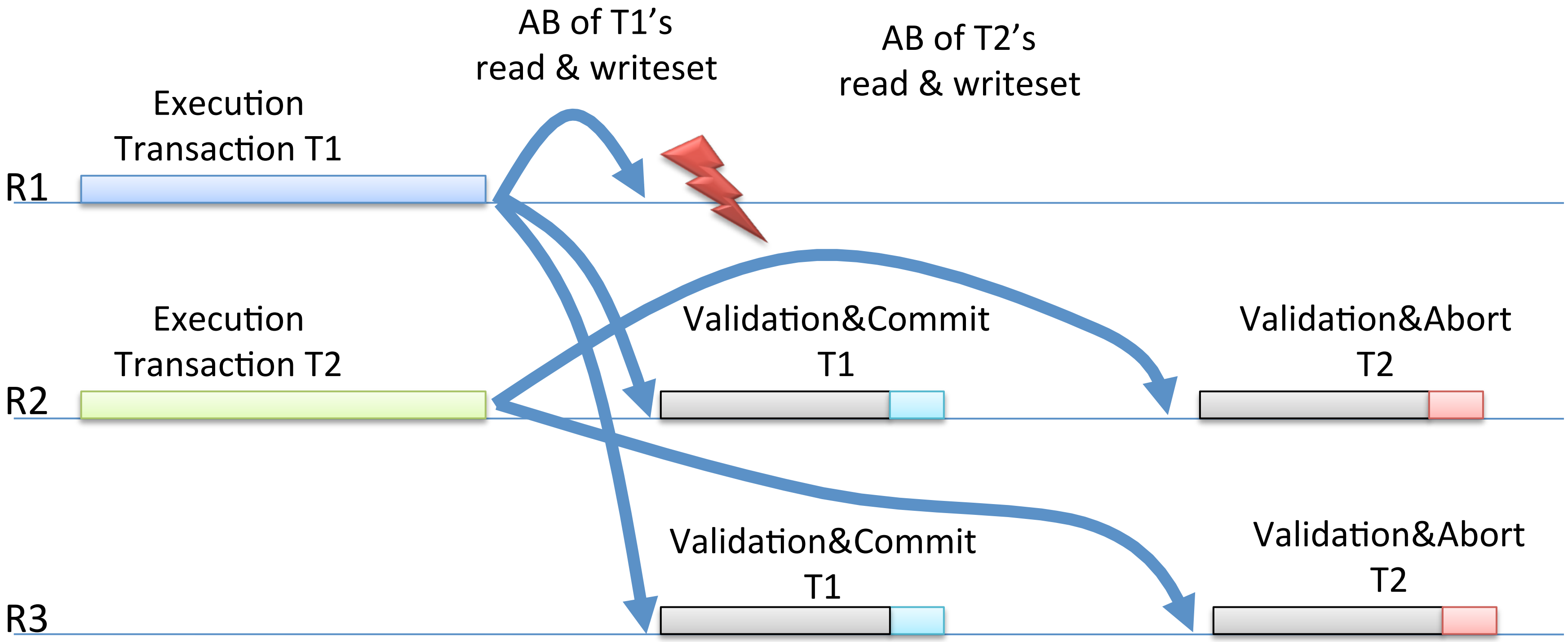
10x incr. nodes → 1000x incr.deadlocks

Atomic Broadcast-based Transactional Replication Schemes

- Multi-master schemes:
  - no intra-transaction coordination
  - rely on Atomic Broadcast (AB) rather than 2PC:
    - deadlock-freedom schemes
    - AB is (1 comm. step) faster than 2PC

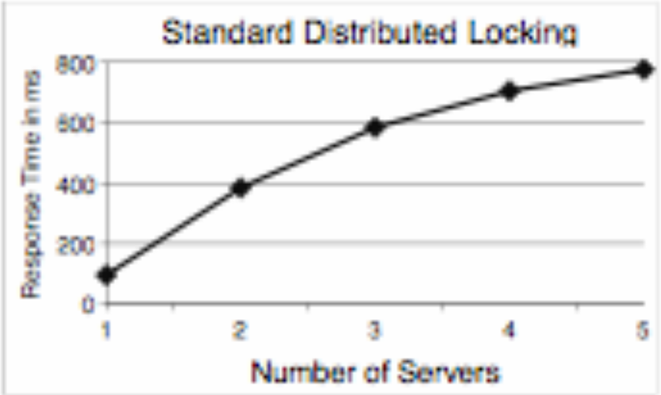
- AB ensures:
  1. agreement on set of received messages:
     - all or none (correct) processes deliver a message
  2. agreement on the order of message delivery
  3. no blocking scenarios despite process crashes

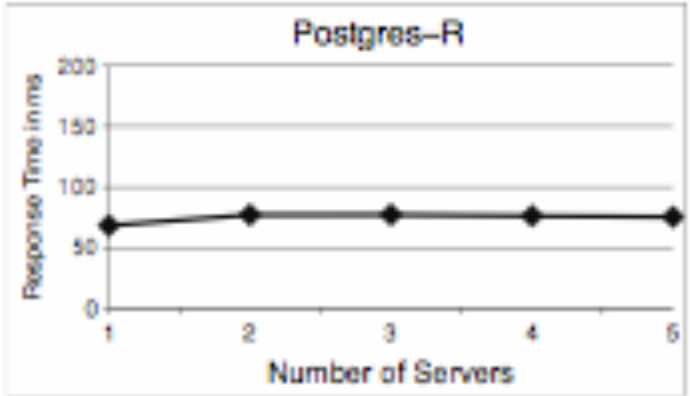
A Conventional AB-based Replication Scheme
“Non-voting Certification Protocol”



- No communication overhead during xact execution:
  - one AB per xact
- No distributed deadlocks

Perfomance of AB-based replication schemes (database world)





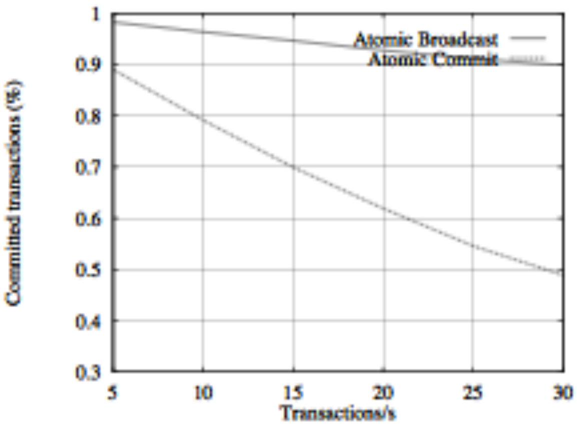


Figure 4: Equiprobable accesses

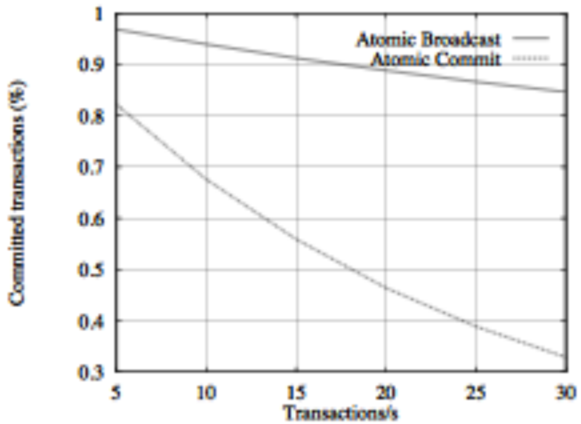
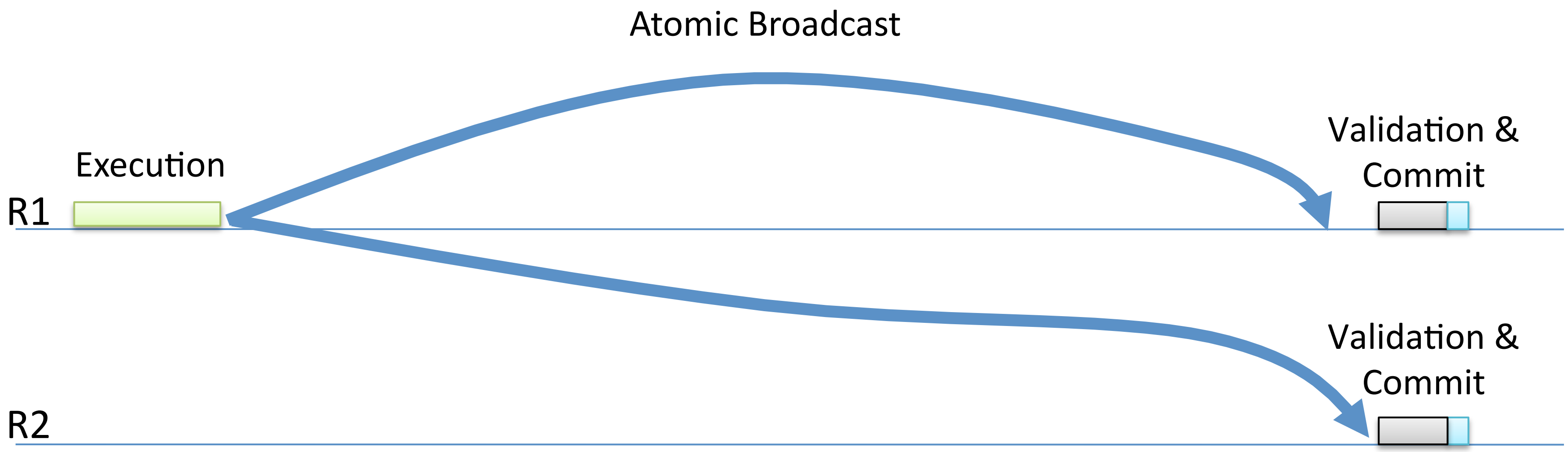


Figure 5: Hot spots

How it actually looks like in a STM context



- In STMs, transactions are often 10-100 times smaller than in DBs:
  - the cost of AB is correspondingly amplified
- Optimistic scheme subject to risk of high abort rate:
  - a posteriori certification
  - transactions might be undefinitely aborted, e.g. long xact VS stream of smaller xacts

Boosting STM's Replication

- I'll overview two recently proposed techniques:
  - Asynchronous Lease Certification (ALC)[Middleware2010]
  - Speculative Transactional Replication (STR) [SPAA2010/ISPA2010]

- ALC and STR pursue the same goal:

- …though leveraging on antithetic approaches!

ALC

joint work with Nuno Carvalho and Luís Rodrigues

Key intuition

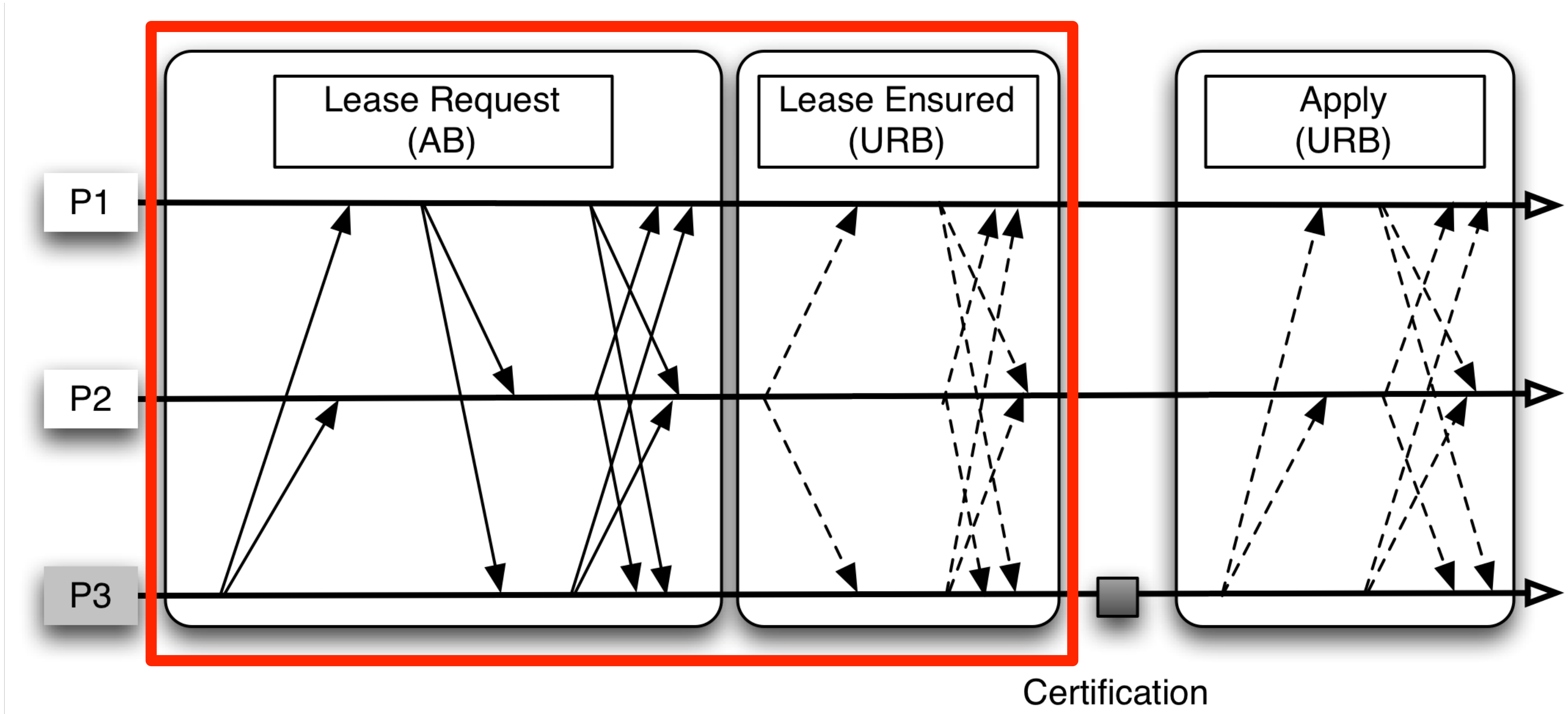
- Exploit data access locality by letting replicas dynamically establish *ownership* of memory regions:
  - replace AB with faster coordination primitives:
    - no need to establish serialization order among non-conflicting transactions
  - shelter transactions from remote conflicts

- Data ownership established by acquiring an ***Asynchronous Lease***
  - mutual exclusion abstraction, as in classic leases…
  - …but detached from the notion of time:
    - implementable in a partially synchronous system

Protocol's overview

- Transactions are locally processed
- At commit, replicas checks if a lease on the accessed data is already owned:
  - NO
    1. an Asynchronous Lease is established
    2. the transaction is locally validated
    3. if validation succeeds, its writeset is propagated using Uniform Reliable Broadcast (URB):
       - no ordering guarantee, 30-60% faster than AB
    4. if validation fails, upon re-execution the node holds the lease:
       - xact cannot be aborted due to a remote conflict!
  - YES
    - as above, but from point 2.

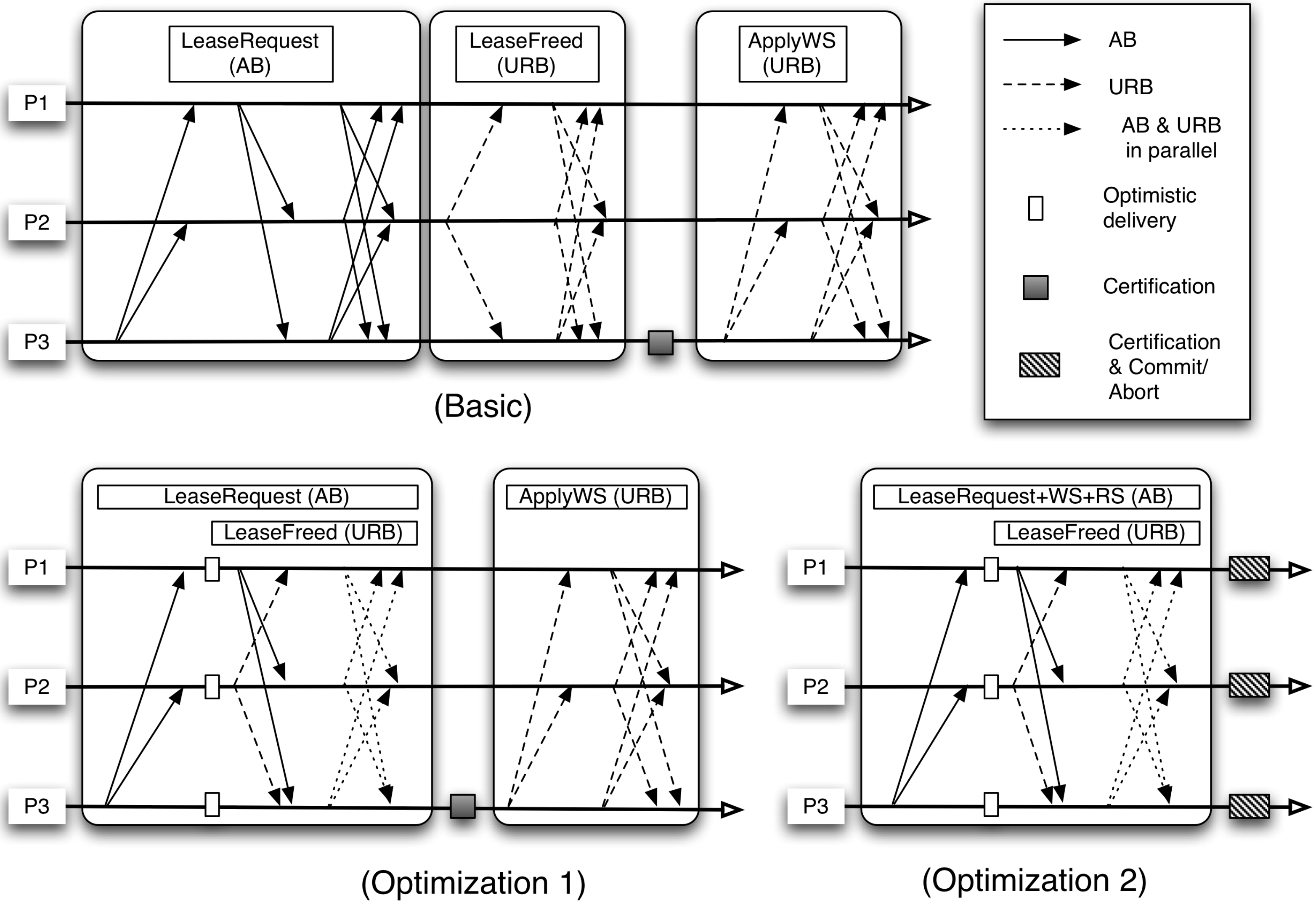
Asynchronous Lease Establishment
Basic Protocol



Simple but sloppy:

If a node doesn't own a lease, it incurs in the latency of 1 AB + 2 URB to commit a xact

Asynchronous Lease Establishment Optimized Protocol



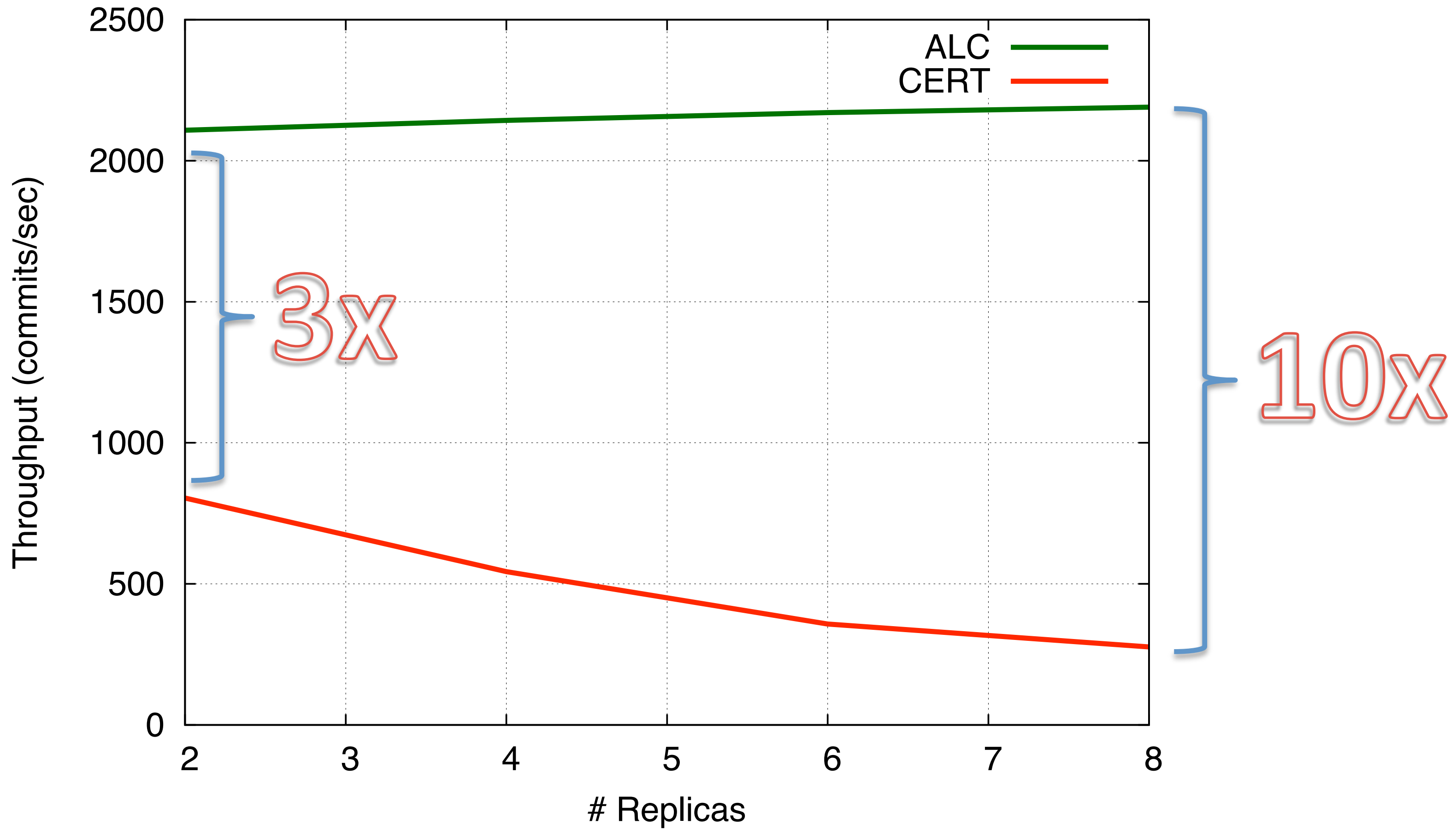
Benefits of ALC

- If applications exhibit some access locality:
  - avoid, or reduce frequency of, AB
  - locality enhanceable via conflict-aware load balancing

- Ensure transactions are aborted at most once due to remote conflicts:
  - essential to ensure liveness of long running transactions
  - benefic at high contention rate even with small running transactions

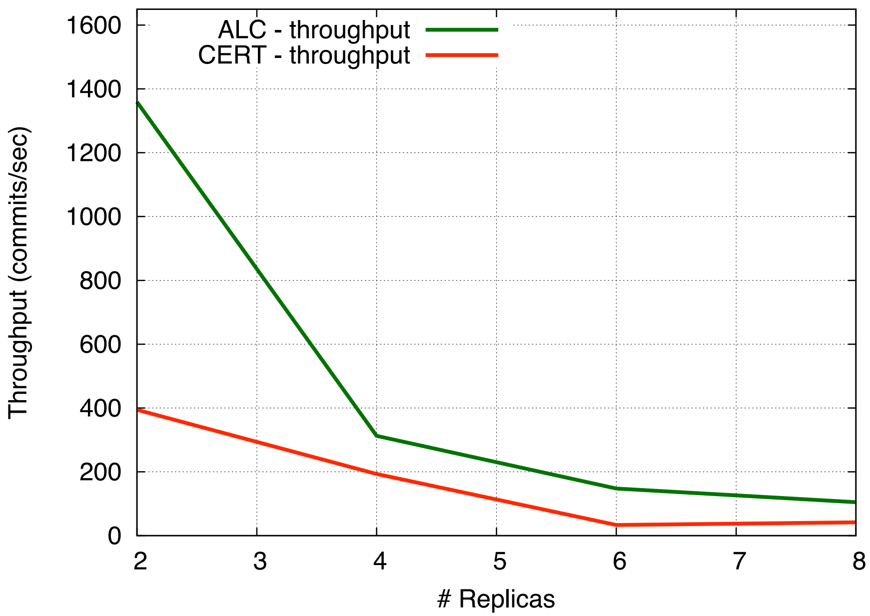
Synthetic "Best case" scenario

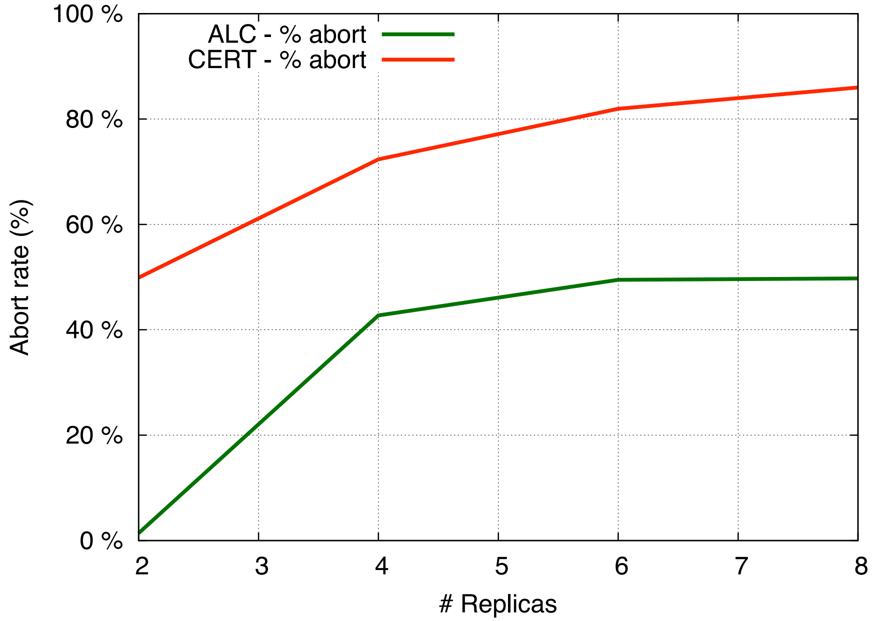
- Replicas accessing distinct memory regions



Synthetic "Worst case" scenario

- All replicas accessing the same memory region

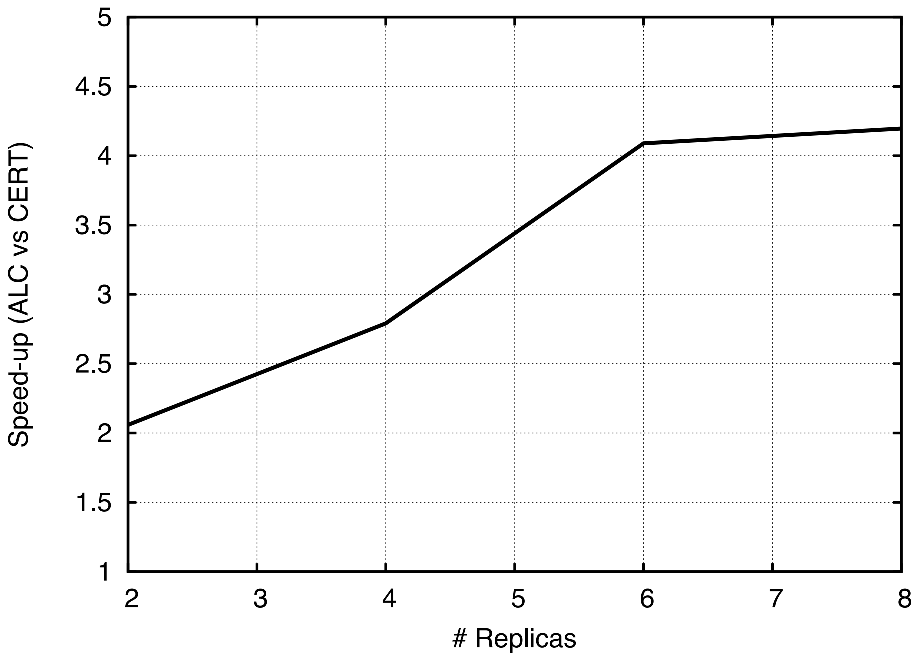


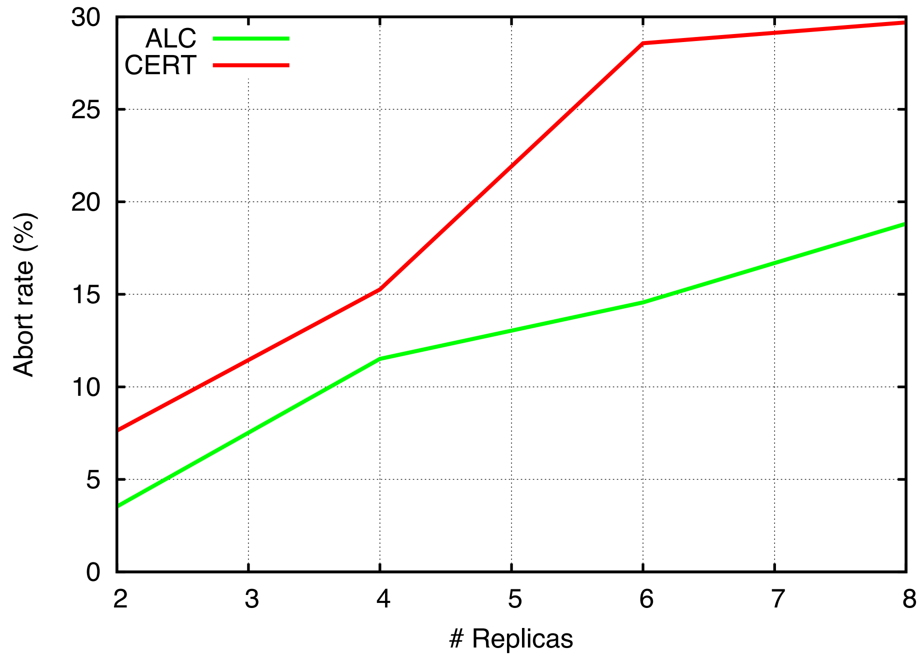


on av. ≈3x speedup due to reduced abort abort rate

Lee Benchmark

- Complex application with diverse workload:
  - both long and short running transactions





- long running transactions subject to livelock:
  - aborted up to 10 times

Speculative Transactional Replication

joint work with R. Palmieri, F. Quaglia, N. Carvalho and L. Rodrigues

Beyond certification mechanisms

- Certification schemes achieve no overlapping between transaction processing and replica coordination:
  – AB is started only after transaction ends!

- Can’t we do any better to minimize the coordination costs?

YES WE CAN!

- Using optimistic deliveries + state machine:
  – messages are received from the network long before their final order is established by the AB

1. AB incoming transactions and execute on all nodes:
   - RPC-like execution fashion of the xacts
2. start processing as soon as a xact is opt-delivered

+ overlapping between processing & communication

Easier to say than to do....

1. in STM transactions can be VERY small !!

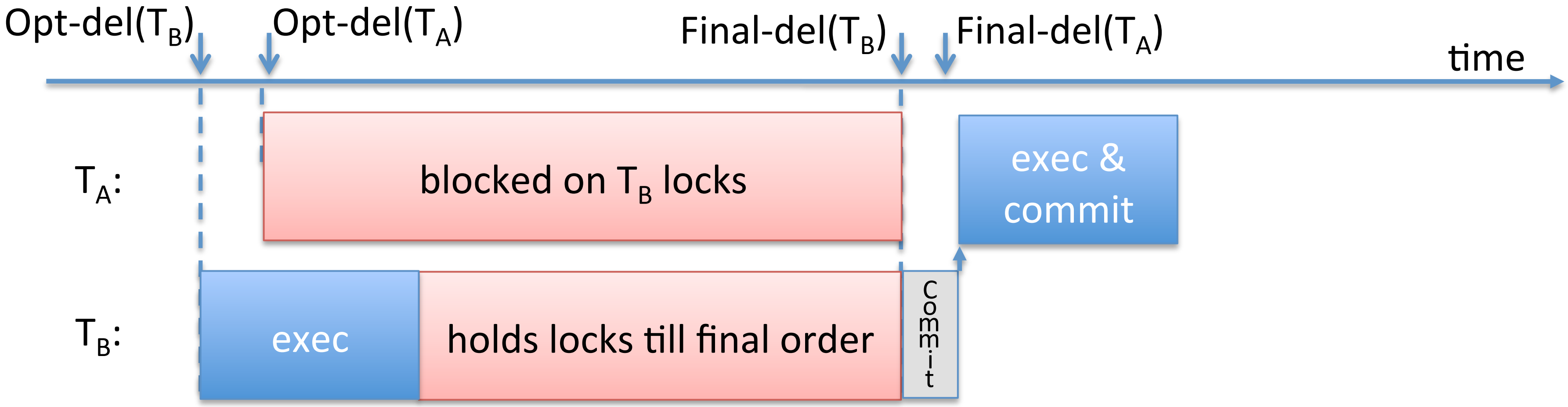
...much ado for nothing! ☹

Easier to say than to do….

2. This only works if transactions execute deterministically at all replicas

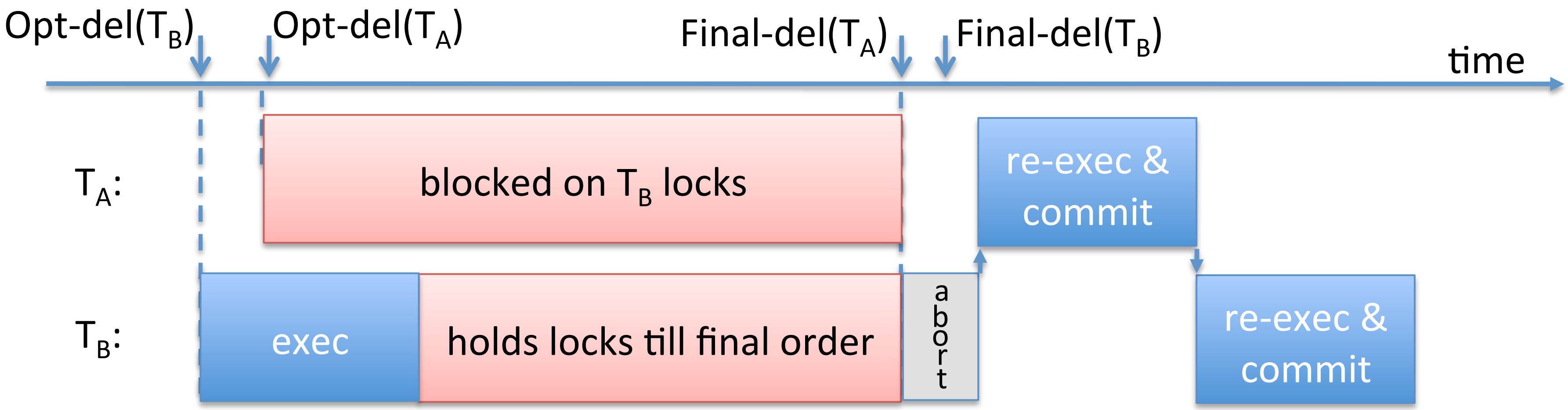
- classic concurrency controls (e.g. 2PL) are not deterministic
- existing solutions have several key limitations:
  - a-priori knowledge of readsets/writesets:
    - may force to large conflict over-estimation
  - acquire **ALL** locks as xact begins
    - way more pessimistic than classic 2PL

VERY POOR CONCURRENCY!

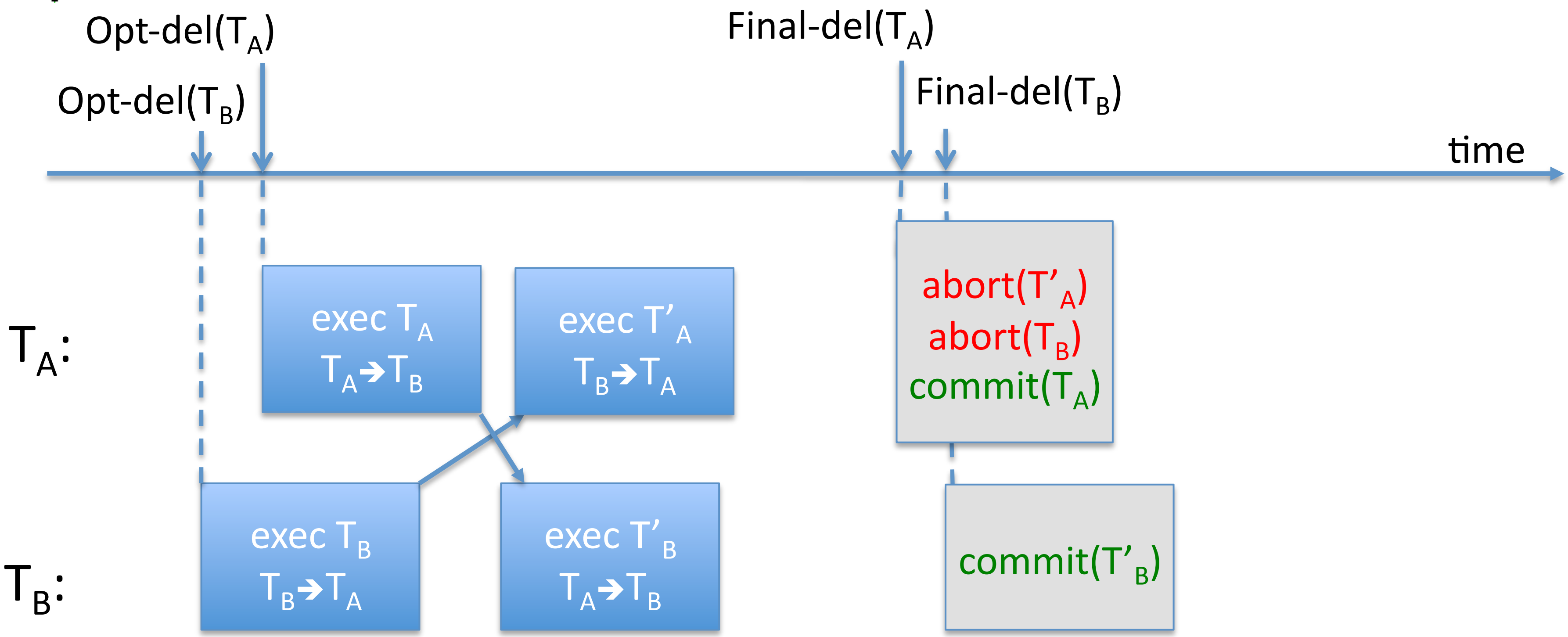


Easier to say than to do....

3. Vulnerable to mismatches between final and optimistic delivery orders!



Don't be pessimistic…be speculative!



Speculatively explore multiple Serialization Orders (SO)

+ **take max benefit from multi-core architectures**
+ **shelter from network reordering**
+ **avoid lock convoying**

- **#SOs can grow factorially with #msgs not yet finally delivered**
  - true in worst case: every xact conflicts with every other, hardly the case in practice
+ **#SOs in which a xact observes distinct snapshots depends on actual conflict graph**

Problem formalization: Optimal STR protocol

$\Sigma=\{T_1,\ldots,T_n\}$: set of Opt-delivered, but not yet TO-delivered, transactions

$\Sigma'=\{T_1^1,\ldots, T_1^k,\ldots,T_n^1,\ldots,T_n^m\}$: set of fully executed speculative transactions

An optimal STR protocol must guarantee:

Consistency: each speculative xact is view-serializable

Non-redundancy: no two speculative xacts observe the same snapshot

Completeness: if system is quiescent (stops Opt- and TO-delivering messages) then, for every permutation $\pi(\Sigma)$ of Σ and for every T_i in Σ, eventually there is a T_i^j in $\pi(\Sigma)$ that has observed the same snaphot generated by sequentially executing all the transactions preceding $T_{i.}$

→ Filters out trivial solutions that blindly enumerate all permutations of Σ

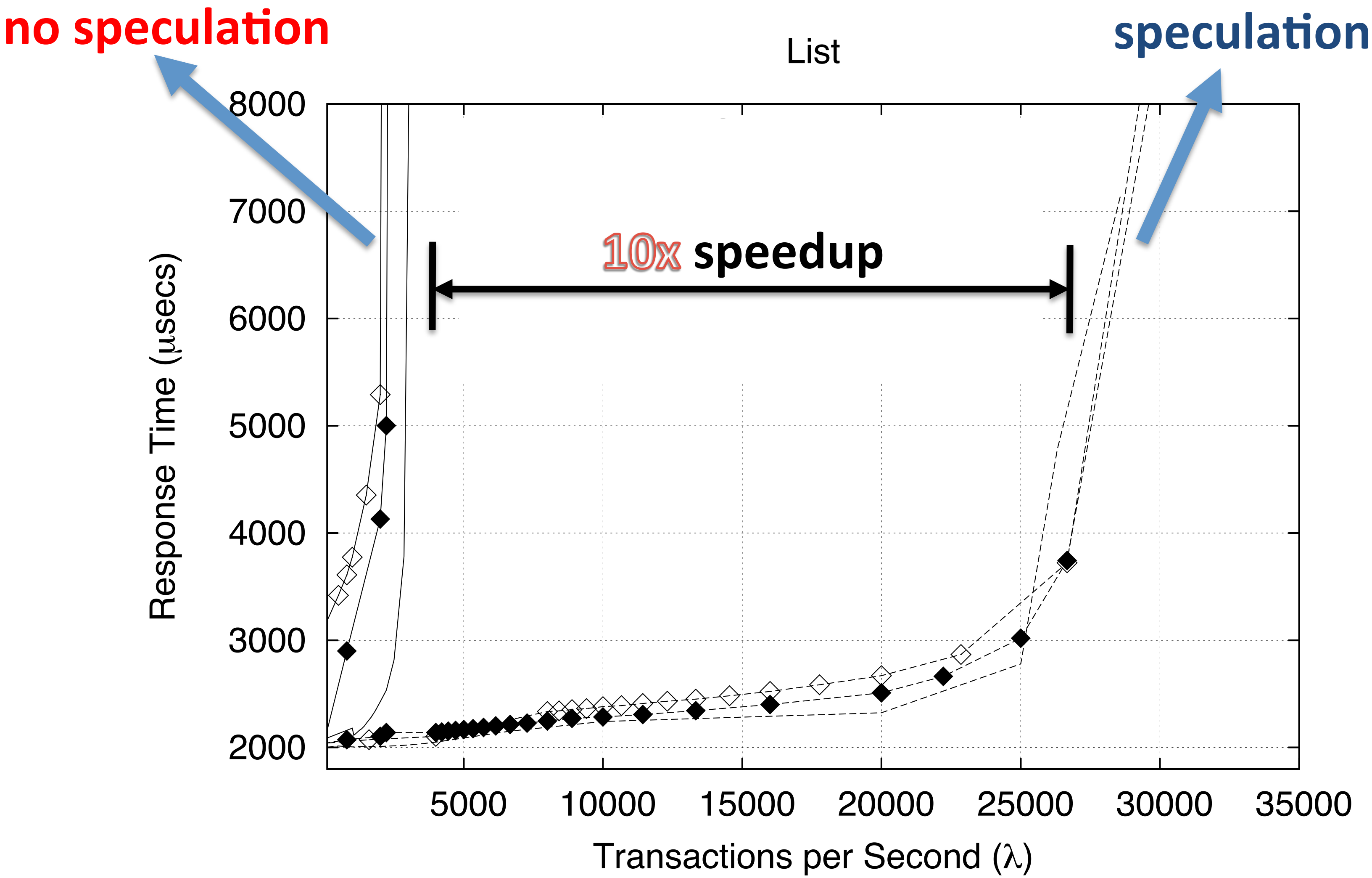
Shelters from any mismatch between optimistic and final delivery order ←

An Optimal STR Protocol
Core Technical Challenge

- Design a provably optimal speculative concurrency control:
  - online algorithm driving the dynamic generation of speculative transactions based on conflict patterns

- Key Idea:
  - each speculative xact maintains a **Speculative Polygraph (SP)**
    - keeps track of conflicts developed with other xacts
    - embeds a family of digraphs, each associated with an equivalent serialization order for the transaction
    - unlike traditional polygraphs accommodate for the coexistence of non-conciliable speculative transactions

Performance speed-up (20% reordering, only one SO explored)



ALC *vs* STR

Bridle concurrency to exploit lighter synchronization sche es & reduce conflict

+ higher scalability w … intensi… wo…
  • upda…
+ can signi…

- no o… a…
- e… locality
- can gener… …er mes… (leas… requests & writeset)

Overlap comm. & proc. via speculation, reduce abort via r dundant computation

…ping processing and …on (AB)

…ssibly large …writeset)

…by all replicas …intensive

…s' dependencies is … and ca… …pensive

- doesn… work for long ru…ning transactions

optimized for different workloads

NO ONE-SIZE-FITS-ALL SOLUTION!

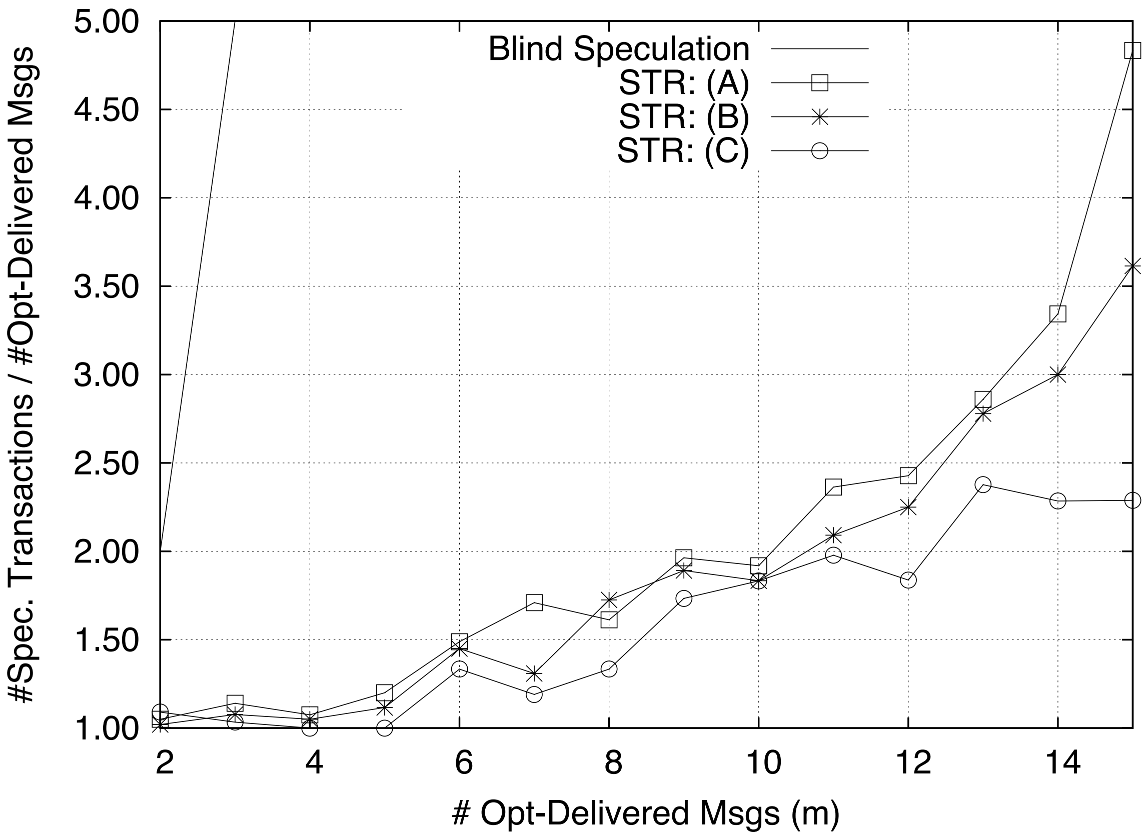
Conclusions & Future work

- Overhead of conventional transactional replication schemes is strongly amplified in STMs
- ALC & STR:
  - up to 10x performance boost via antithetic approaches
  - optimized for different workloads
- Future work:
  - Workload-driven adaptive replication
  - Partial replication
  - Deployment on elastic cloud computing platforms

Thanks for the attention

Q&A

Serialization Orders per transaction
Optimal protocol VS Blind speculation



Simulation study based on real (STM) workloads:

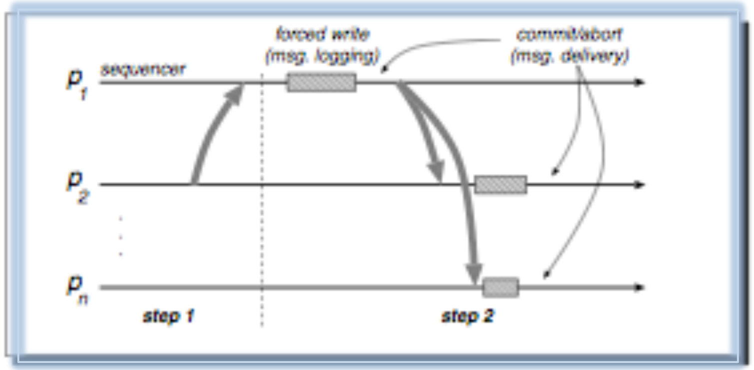
Optimal STR scheme: #SOs≈[2.5-5] with 15 opt-delivered xacts

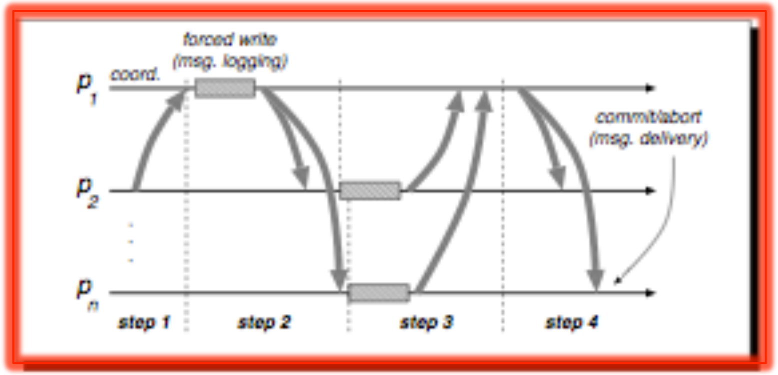
Blind enumeration: #SOs≈1,000,000 with 10 opt-delivered xacts

BACKUP SLIDES

Atomic Broadcast – how expensive?

protocol	resilience	# comm. steps	# msgs.	# forced writes
Sequencer based AB (i)	Blocking	2	n+1	n
Two Phase Commit	Blocking	3	3n	n
Sequencer based AB(ii)	Non-blocking	4	4n	n
Three phase commit	Non-blocking	5	5n	n

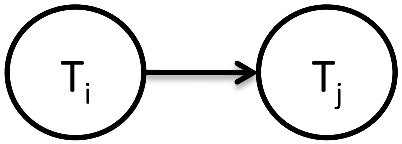




An optimal STR Protocol
Classic Polygraphs

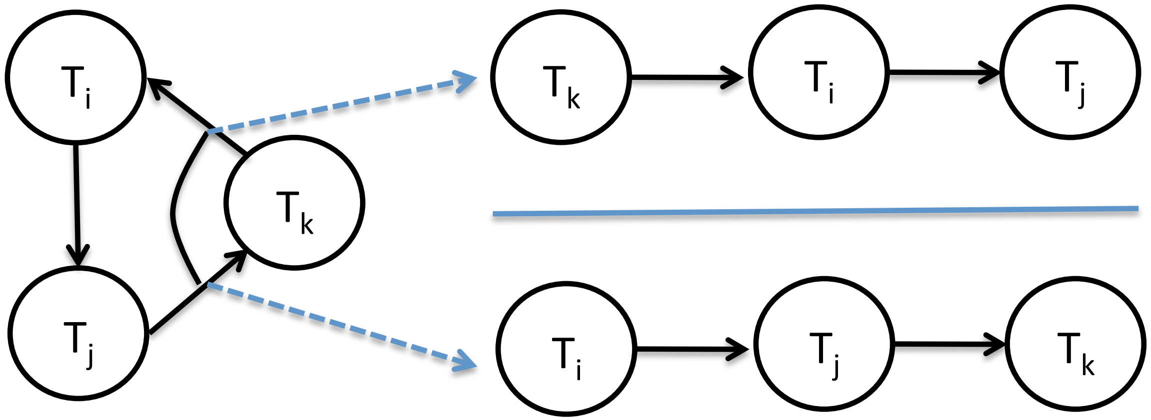
- P=(N,A,B)
  - N: set of vertexes, one per xact
  - A: set of edges (Ti➔Tj) tracking read-from relationships

T_i: w(x_i)
T_j: r(x_i)



  - B: set of bipaths <(Tk➔Ti),(Tj➔Ti)> serializing two writers with respect to a reader

T_i: w(x_i)
T_j: r(x_i)
T_k: w(x_k)



- P is associated with a family of directed graphs, called D(P)

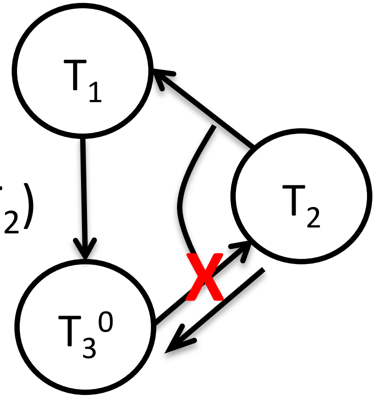
A history H is view serializable iff exists an acyclic direct graph in D(P(H))

Polygraphs don't work with speculative histories!

T_1: w(x) w(y)

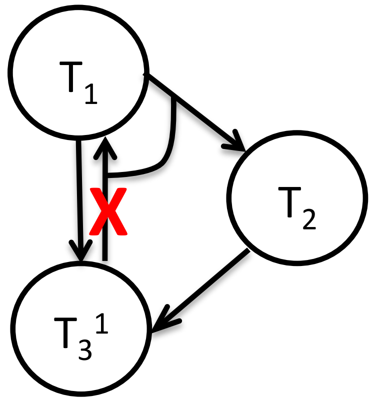
T_2: w(y) w(z)

T_3^0: r(x,T_1) r(y,T_1) r(z,T_2)



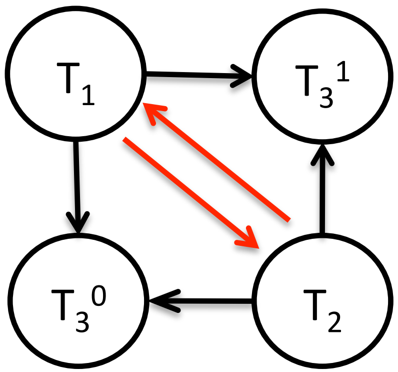
equivalent to:
T_2 T_1 T_3^0

T_3^1: r(x,T_1) r(y,T_2) r(z,T_2)



equivalent to:
T_2 T_1 T_3^0

The classic approach would merge the two above polygraphs, yielding a cycle between T1 and T2!



Speculative polygraphs (SPs)

Basic intuition:

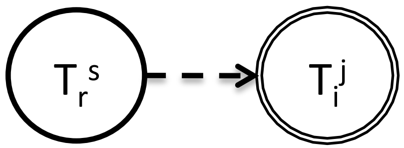
- keep into account history as perceived by each speculative transaction T_i^j
- $SP(T_i^j)$ selectively merges the polygraphs of speculative transactions T* s.t.:

1. *T* conflict, either directly or indirectly, with T_i^j*
2. *at least a serialization order exists allowing both T* and T_i^j to exist*

$SP(T_i^j)=(N,A,B)$ where:

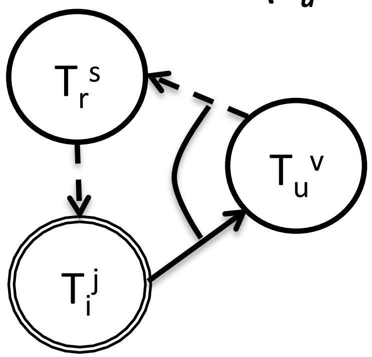
- N is a set of vertex, associated with (speculative) transactions
- A is a set of ***merging edges*** $(T_r^s \odot\!\rightarrow T_i^j)$ which merges $SP(T_r^s)$ and $SP(T_i^j)$

T_r^s: $w(x_r^s)$
T_j^j: $r(x_r^s)$



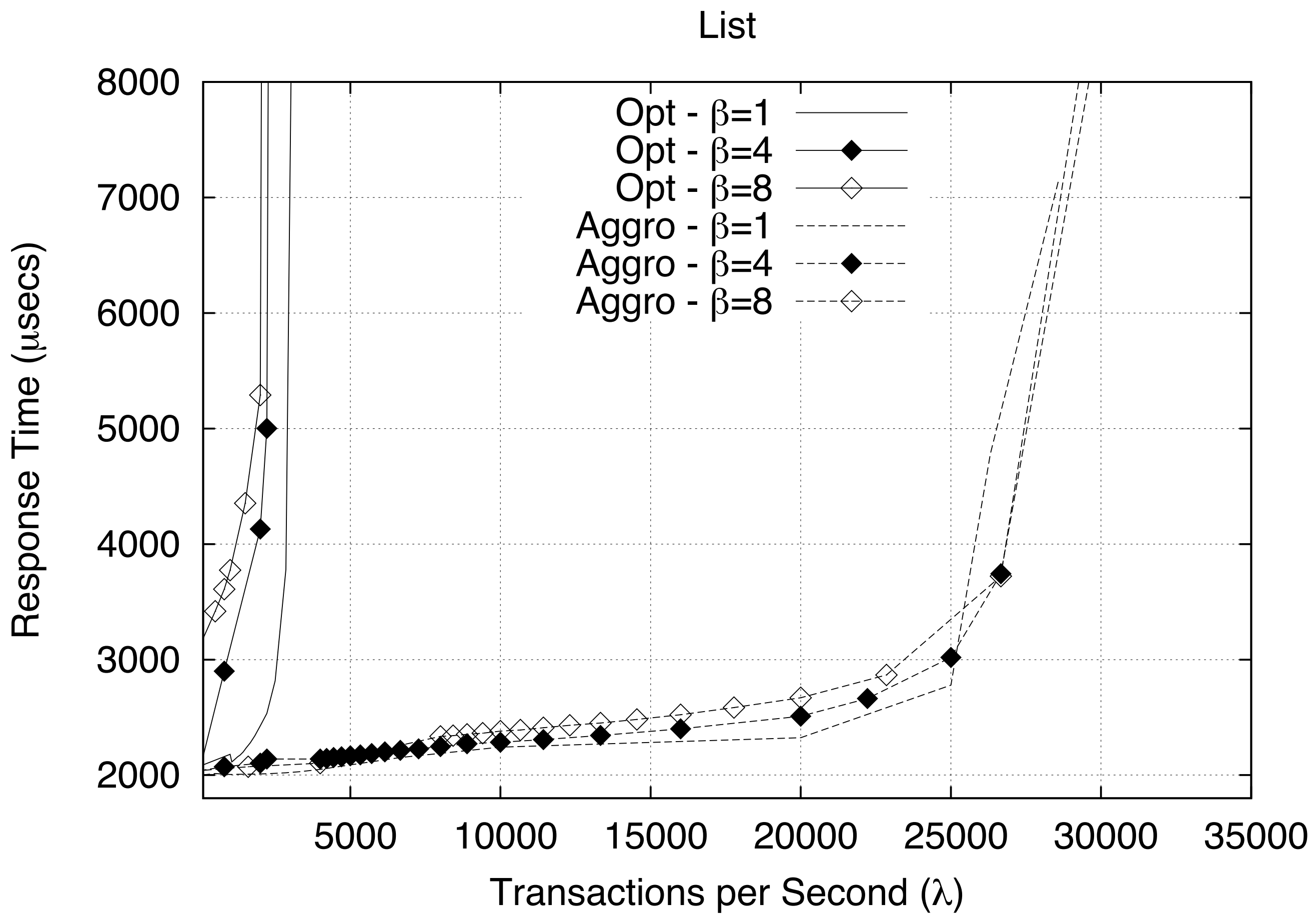
- B is a set of asymmetric bipaths denoted as $\langle (T_u^v \odot\!\rightarrow T_i^j), (T_i^j \rightarrow T_u^v) \rangle$

T_r^s: $w(x_r^s)$
T_j^j: $r(x_r^s)$
T_u^v: $w(x_u^v)$



WARNING
This is just a teaser,
details in the paper!

Performance speed-up
(20% reordering, only one SO explored)



Performance evaluation

- Based on fully fledged prototype
- Relies on a state-of-the-art multi-versioned STM for local concurrency regulation
- Permits transparent execution of legacy (distribution agnostic) STM applications