

STM for Speculative Out-of-order Event Processing

Pascal Felber
University of Neuchâtel
Pascal.Felber@unine.ch

Joint work with:
Christof Fetzer, Andrey Brito, Heiko Sturzrehm



Motivations

- Tens of different STM designs
 - Optimized for different **environments** (e.g., word- vs. object-based, OS and H/W) and **workloads** (e.g., read- vs. write-dominated, long vs. short transactions, high vs. low contention)

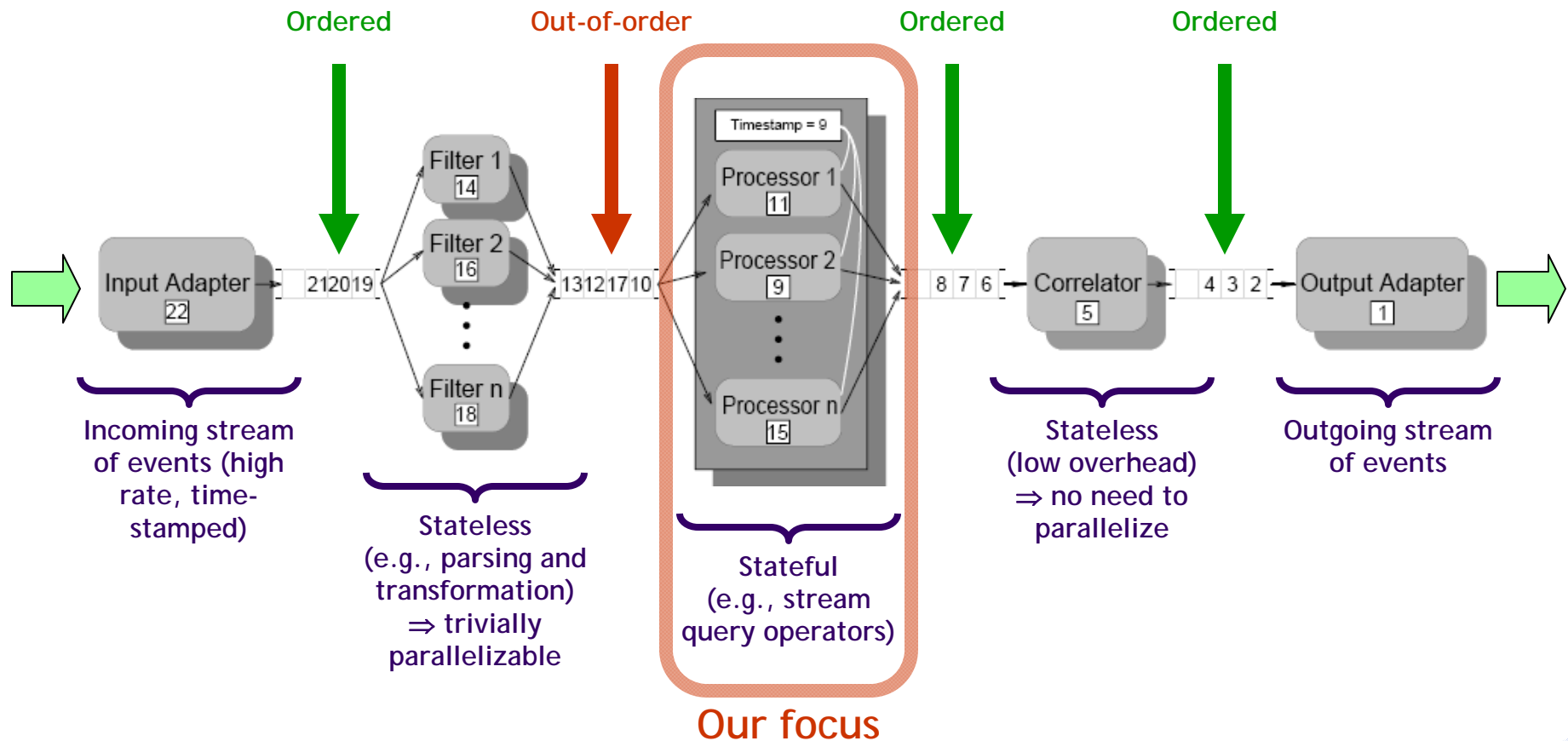
[Lesson] **There is no “one-size-fits-all” STM**

- We need **workloads** and **applications** to drive and evaluate the STM designs
 - Might lead to specialized designs

Target application

- Event stream applications
 - One-way flow of events
 - Chained event processors
 - Some event processors do not maintain state...
 - ⇒ trivially **parallelizable**
 - ...while others don't
 - ⇒ need to process events **sequentially**
 - There exist dependencies between some events
 - ⇒ must be processed **in order**
 - Some events are processed faster than others
 - ⇒ parallel processing may **disorder** events

Event stream applications



Stateful event processing

2 major limitations

1. Events typically need to be processed in order (when they have dependencies)
 - Must **wait** (idle) until the next event arrives, even if following events are available
2. Events cannot be processed in parallel (when they have dependencies)
 - Must process events **sequentially** (cannot use multiple cores)

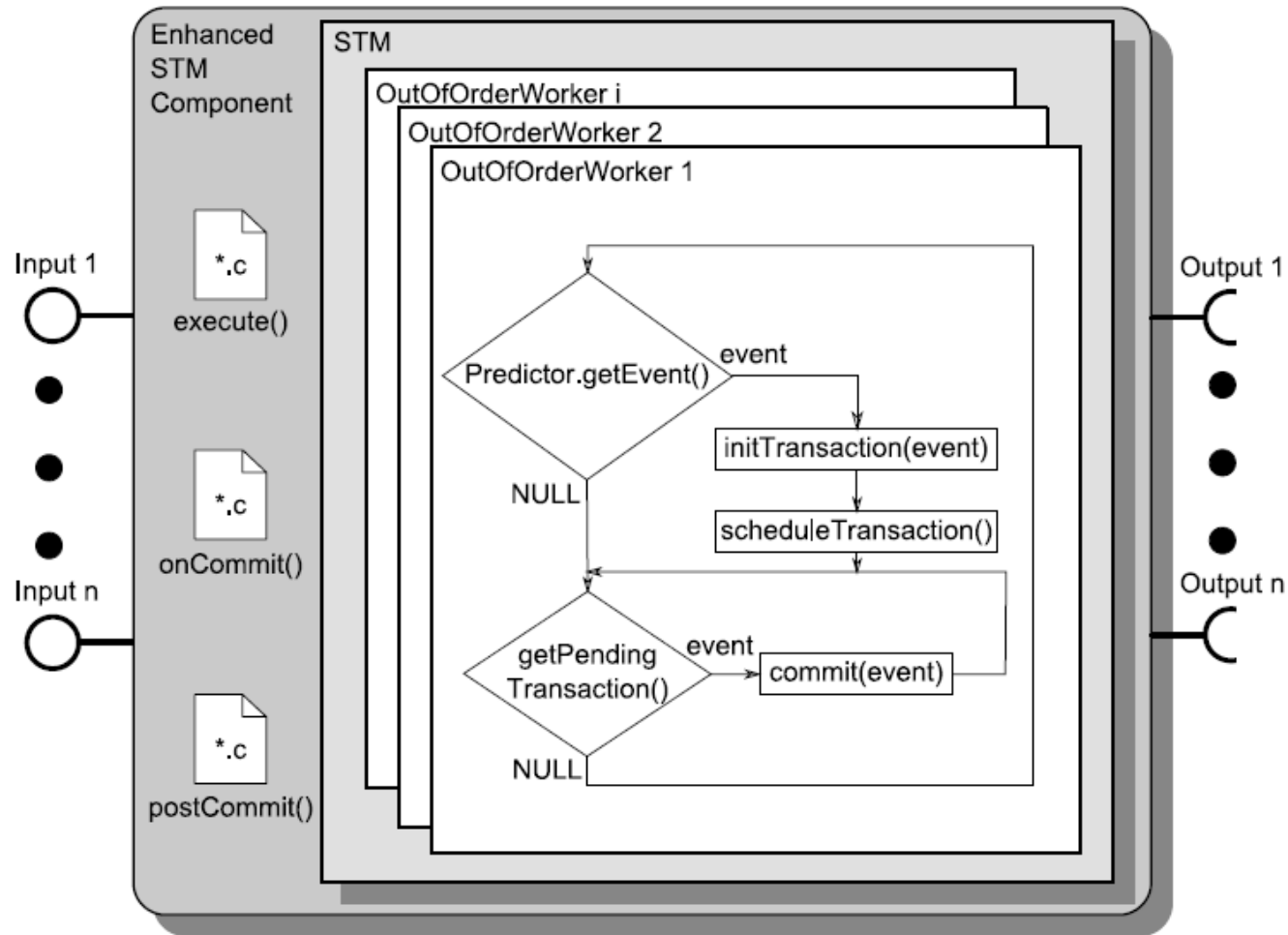
Speculative event processing

- Process events in transactions as they arrive
 - Transactions have a **pre-assigned** commit time (based on timestamps)
 - Upon completion
 - If event is in-order (next to be processed): commit
 - If event is out-of-order: move to “ready queue”
 - Conflicts/dependencies detected dynamically
 - Upon conflict, lowest commit time wins
 - Transactions in “ready queue” may be aborted
 - One-way stream \Rightarrow delayed commit does not block sender

How much optimism do we need?

- Too optimistic may be counterproductive
 - Do not process events too much out-of-order (might be worse than sequential!)
 - When given a choice, pick events that are likely to have few dependencies with missing events
- ⇒ Use **“conflict predictor”** to pick next event
 - **Boolean** predictors: indicate whether a given event should be processed speculatively
 - **Predicate-based** predictors: indicate condition to be met for processing the event (e.g., $e_{i-2} \rightarrow e_i$)
 - Provided by user, or from static/dynamic analysis

Enhanced ESP component



Prototype

- Extended version of tinySTM
- Word-based lock-based STM implementation
 - Written in portable C, 32/64-bit
 - Small code base (<1000 LOC), GPL
- Time-based algorithm
 - Versioned locks used to build consistent snapshot
- “Classical” word-based STM design
 - Per-stripe locks, ETL and CTL
 - Write-through and write-back versions

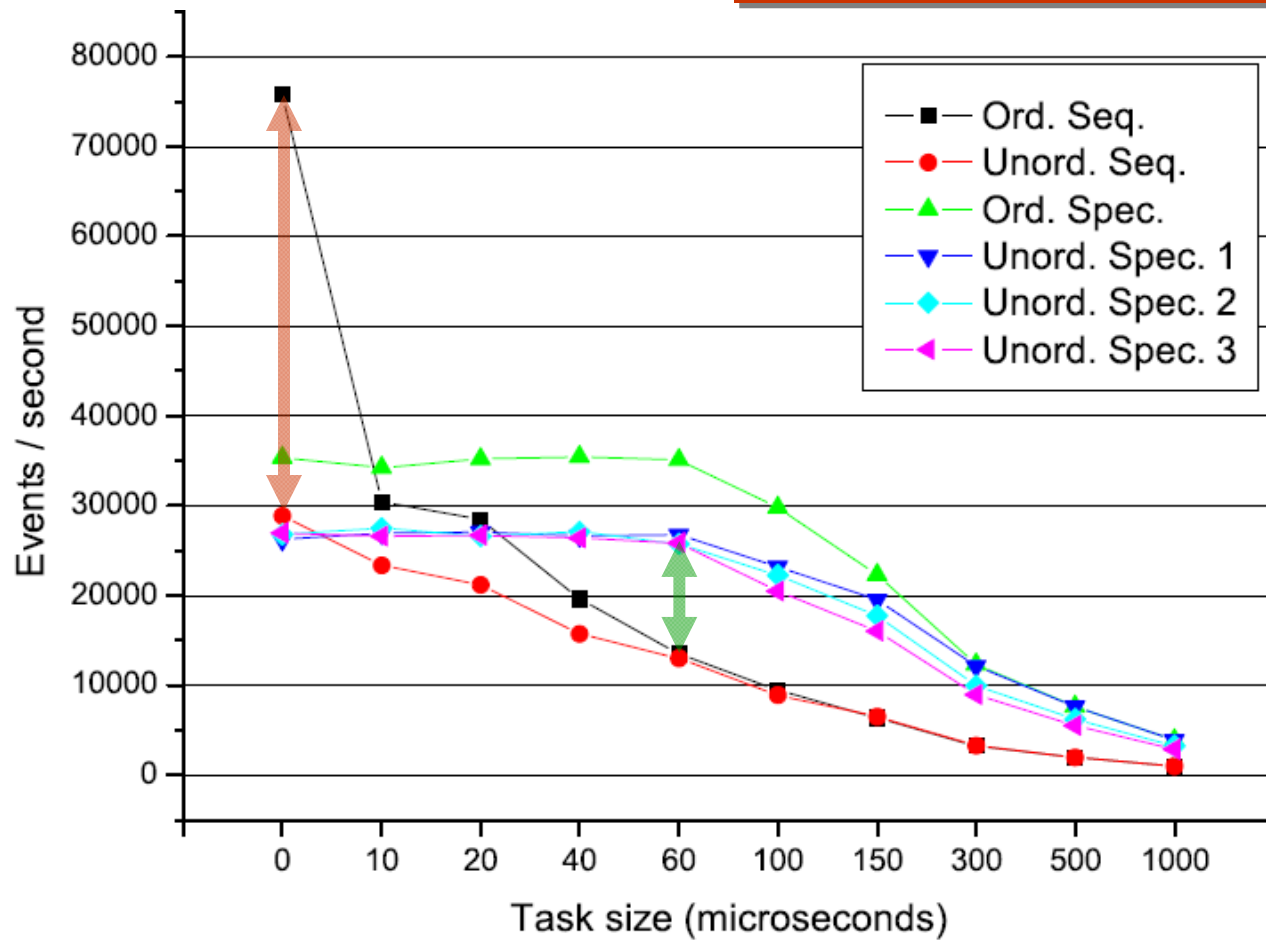
<http://www.tinystm.org>

Scenarios

1. Ord. Seq.: **ordered input, sequential processing**
 2. Unord. Seq.: **unordered input, sequential processing**
 3. Ord. Spec.: **ordered, speculation (exploit parallelism)**
 4. Unord. Spec. 1: **unordered, speculation, 0% conflicts (exploit parallelism and out-of-order processing)**
 5. Unord. Spec. 2: **unordered, speculation, 10% conflicts**
 6. Unord. Spec. 3: **unordered, speculation, 20% conflicts**
- 8-core Xeon, 2 GHz, Linux 2.6.18-4 (64bit)

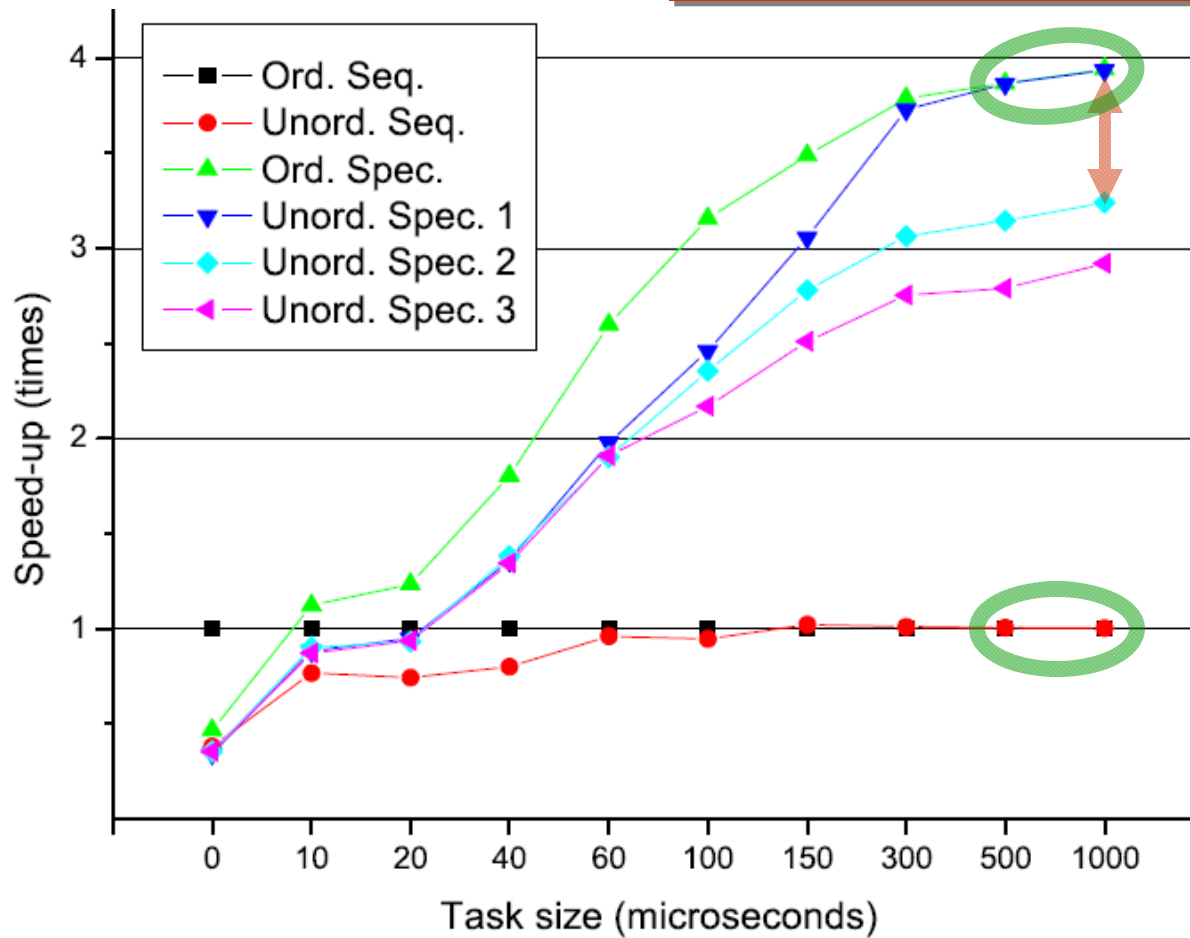
Throughput

Overhead of STM becomes negligible with tasks that take some time. Speculation is beneficial from 20us.



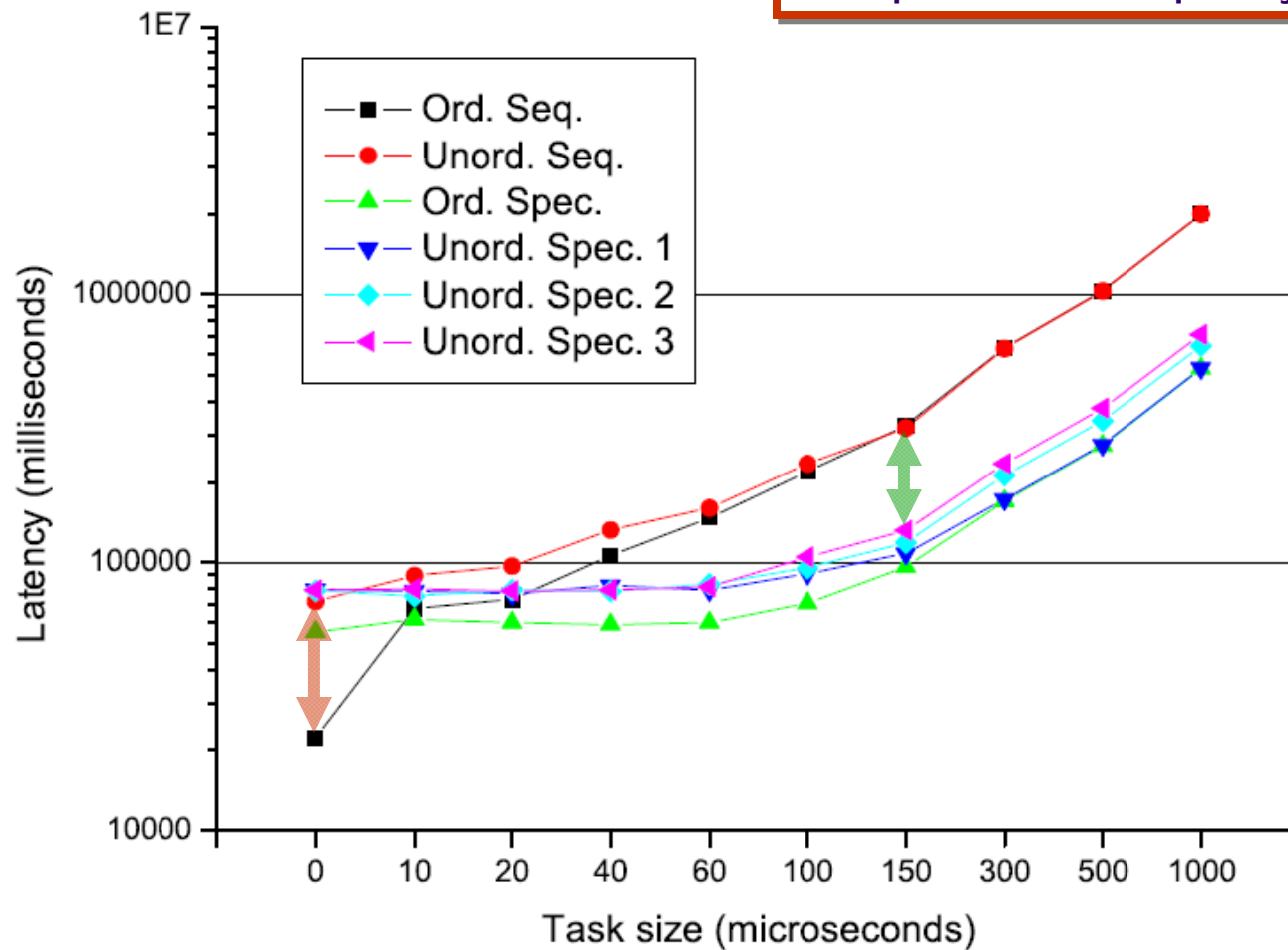
Speedup

Speedup from 20us, reaches 4x.
Even with conflicts, speedup is high.
Longer tasks are more in-order.



End-to-end latency

Latency increases with task size and initial overhead becomes negligible. Speculation is quickly beneficial.

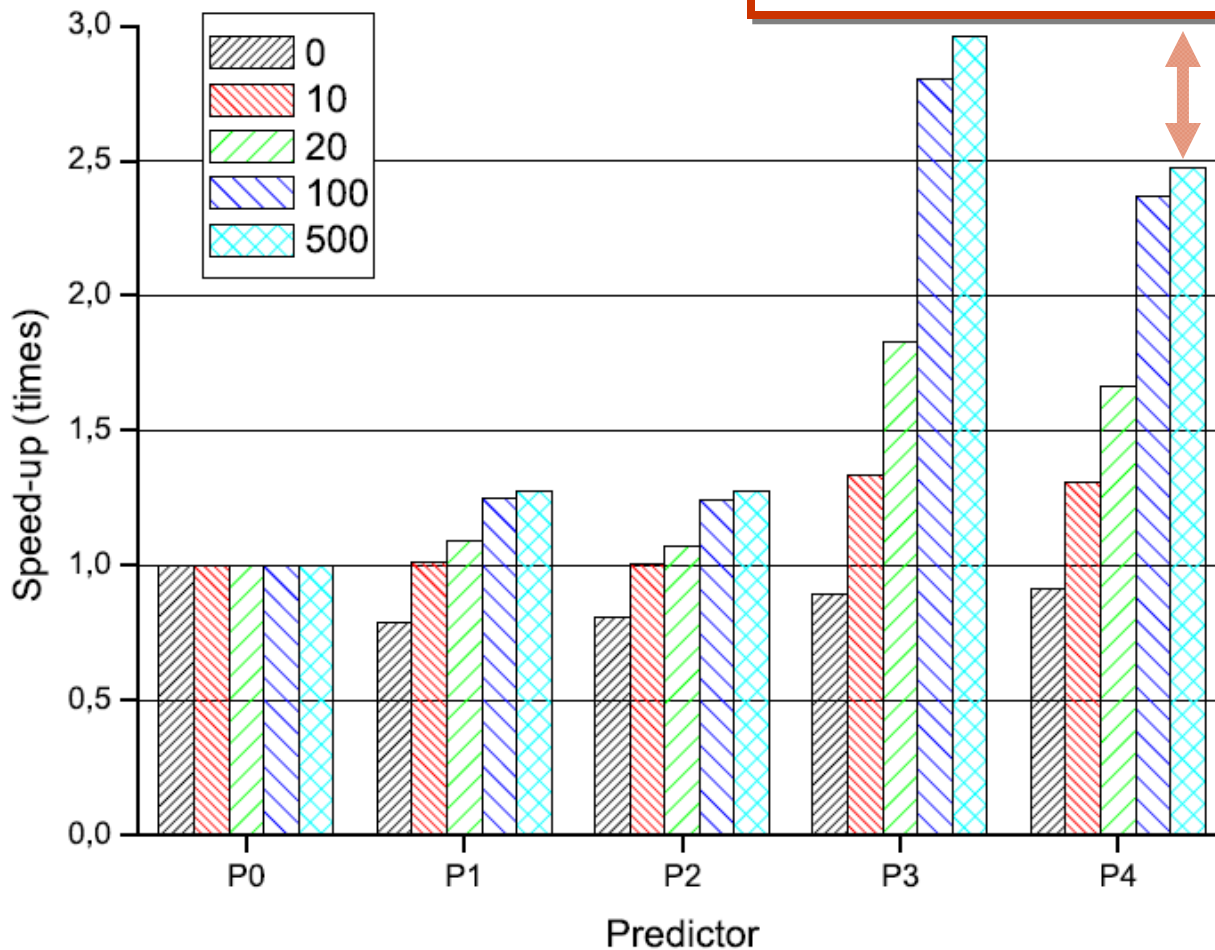


Evaluating predictors

- Experiment with Unord. Spec. 1 and 3
 1. Pred. 0: **always return true**
 2. Pred. 1: **returns true only if the event is no more than 20 time units from the last committed event**
 3. Pred. 2: **perfectly predicts conflicts of Unord. Spec. 3**
 4. Pred. 3: **as Pred. 1 but returns the conflicting event(s) that should be processed before the current one**
 5. Pred. 4: **as Pred. 2 but returns the conflicting event(s) that should be processed before the current one**

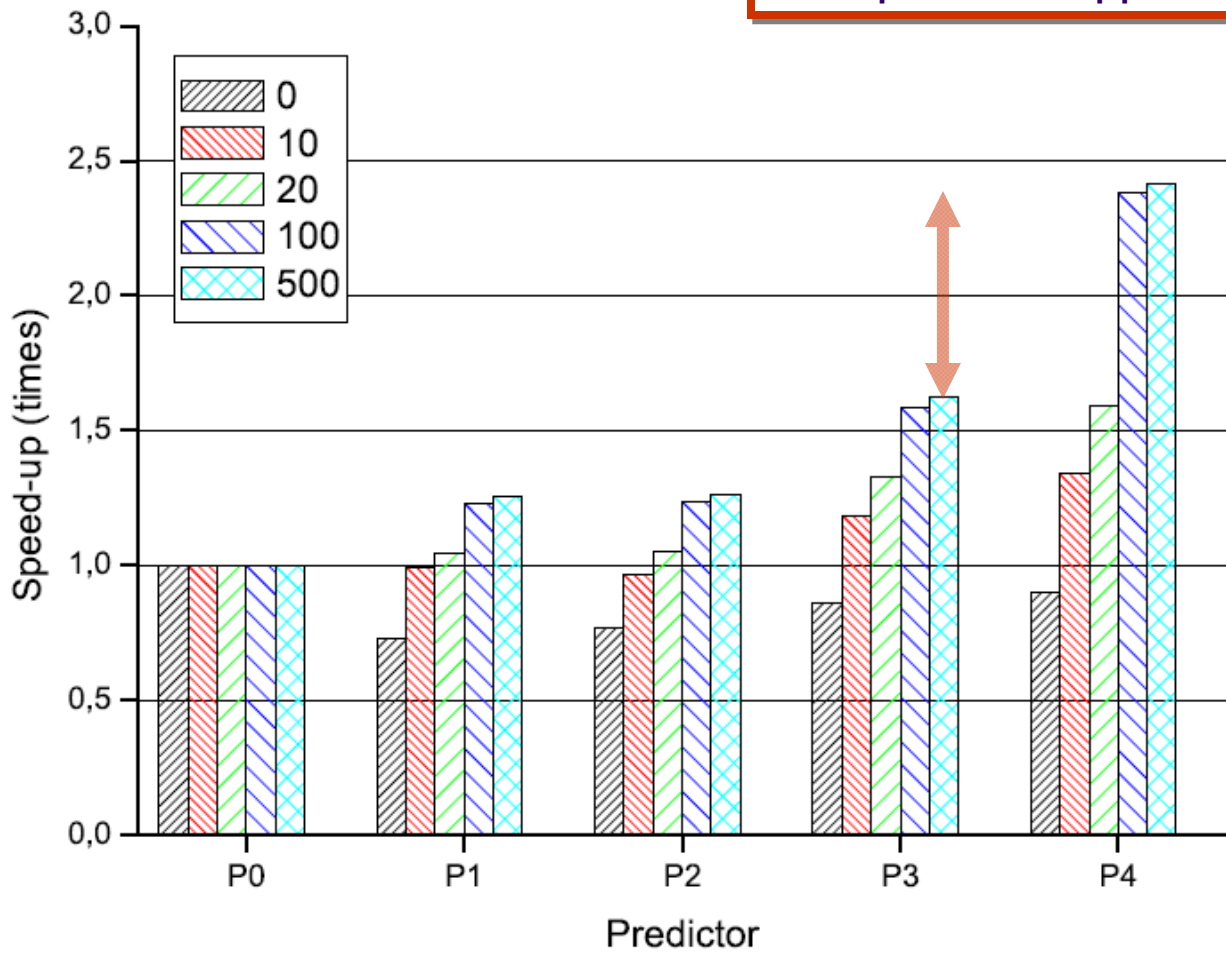
Predictor speedup (Unord. Spec. 1)

No gain with short tasks.
Sophisticated predictors are better.
P4 too conservative (false conflicts).



Predictor speedup (Unord. Spec. 3)

No gain with short tasks.
 P4 is accurate (no aborts) while P3 produces approx. 40% aborts.



Conclusions

- We have **many great STM designs...**
... but we still lack **application domains** and **real workloads** for STM
- Our focus: STM for **speculative out-of-order event processing**
 - Tailored STM for ESP (based on tinySTM)
 - Pre-ordered transactions, delayed commit
 - Predictors drive speculation, improve efficiency
 - Can be easily generalized to support distributed speculation