Kernel-Assisted Scheduling and Deadline Support for Software Transactional Memory

Walther Maldonado, Patrick Marlier, Pascal Felber, Etienne Rivière

University of Neuchâtel, Switzerland

Julia Lawall *(DIKU, Denmark)*

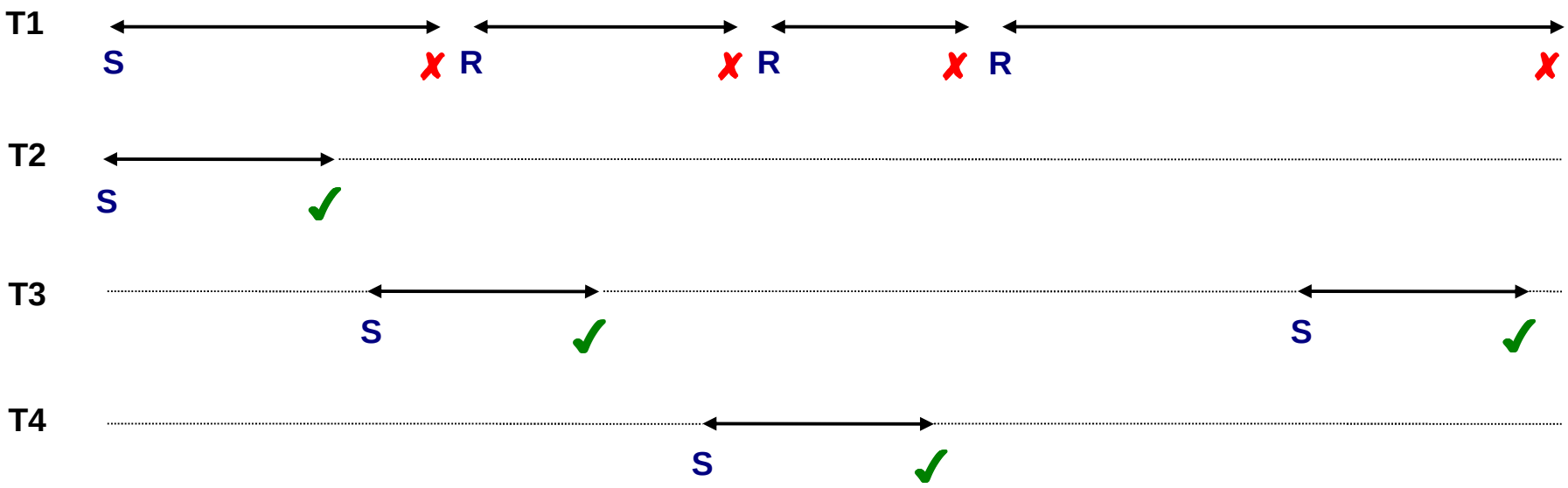
Gilles Muller *(INRIA, France)*

Problems

- Target : situations where conflicts degrade performance because of how transactions are *scheduled*
- Two scenarios considered :
  - Performance on high-contention settings (i.e.: large number of threads)
  - Responsiveness for reactive applications

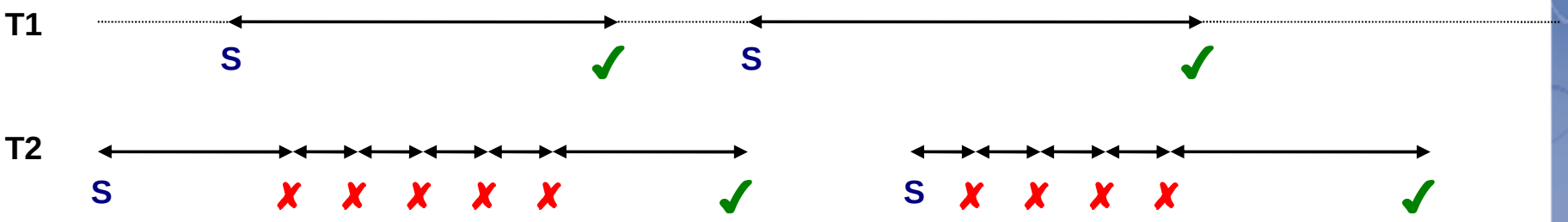
High contention scenario

- Long transactions abort frequently

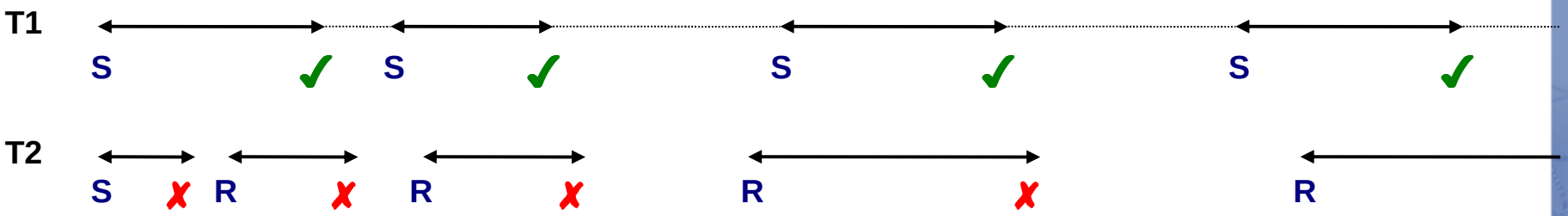


High contention scenario

- Transactions which re-execute immediately are likely to abort again...



- ...but exponential back-offs are imprecise



Deadlines & Real Time

- Reactive (soft real time) applications have time constraints
- *Quality of Service* : acceptable miss rate
- Example : Rendering
  - Target : 60 fps , deadline : 16ms / frame
- We need an acceptable QoS for the reactive transaction (s)
  - *QoS* : 95% (57 fps)
  - *QoS* : 80% (48 fps)
- Minimize penalty on other (computing) transactions

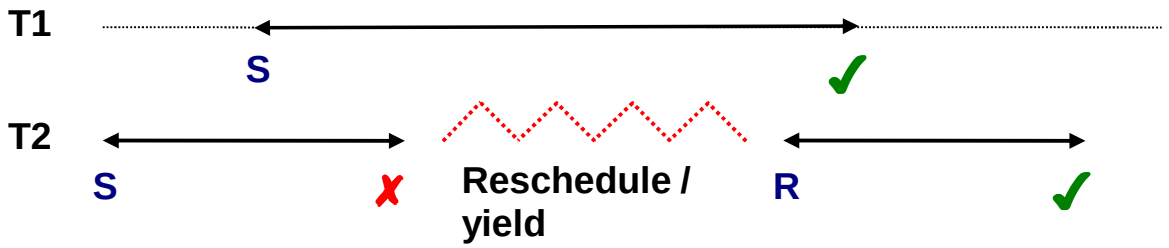
Scheduling Approach

- Add *Serializing* Contention Managers to TinySTM
  - Hard Serializing
  - Soft Serializing
- Add Linux kernel specific support
  - Time Slice extensions

Serializing

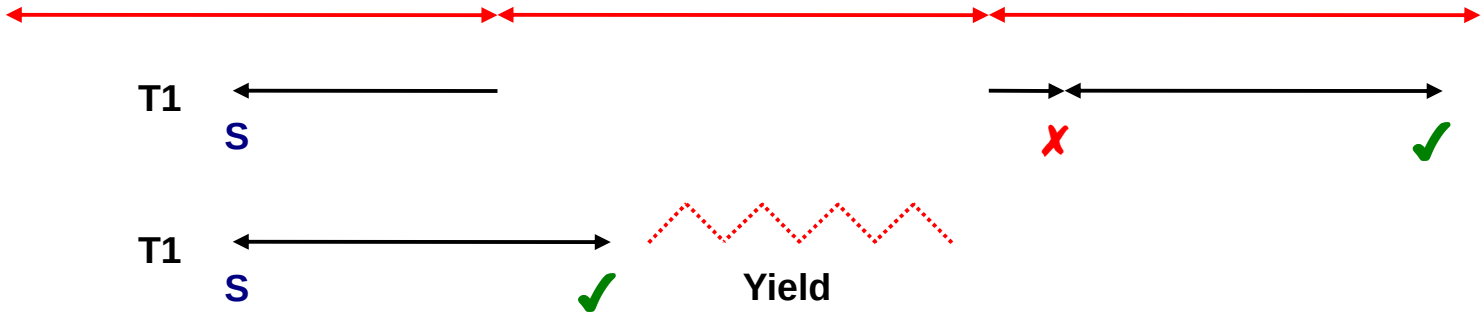
- Conflict bookkeeping
- Hard Serializing (kernel queue)

- Soft Serializing (yield + priority)



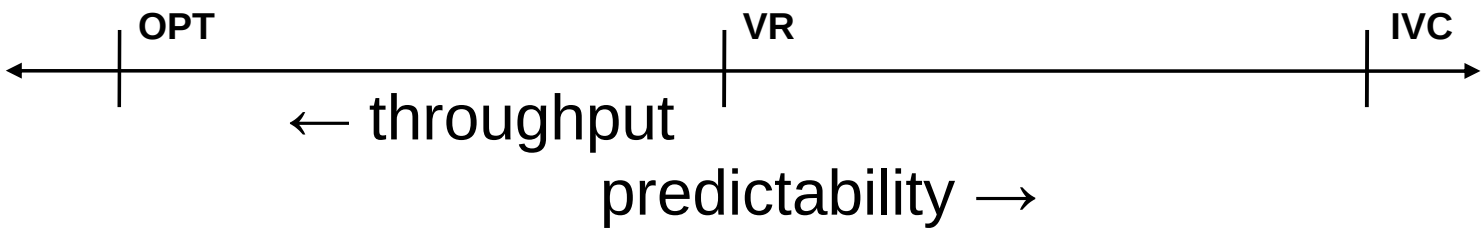
Time Slice Extensions

- Time-Slice Extension



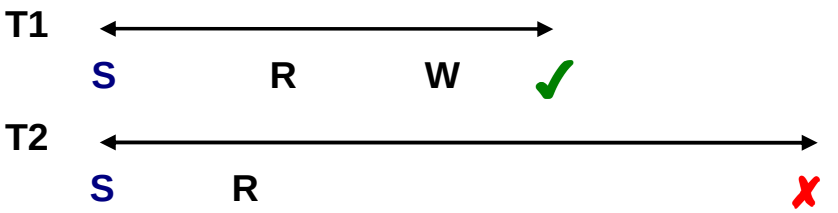
Considerations for Deadlines

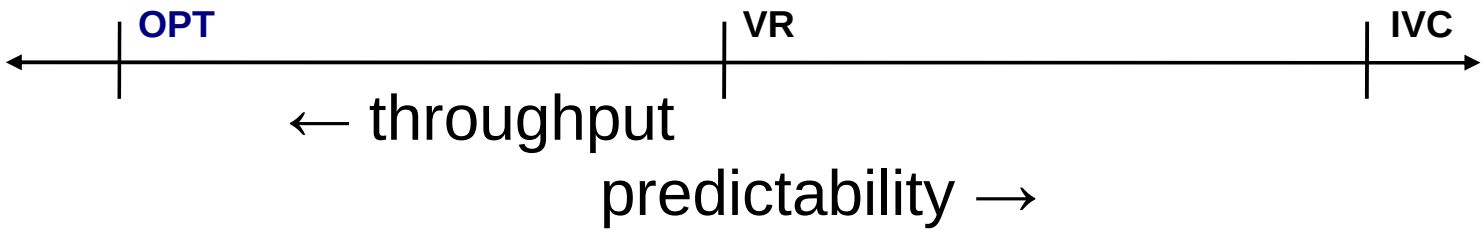
- Hypothesis:
  - *Stable* transaction length
  - Only one thread with deadlines
- Approach based on changing execution modes on abort depending on remaining time before the deadline



Optimistic (OPT)

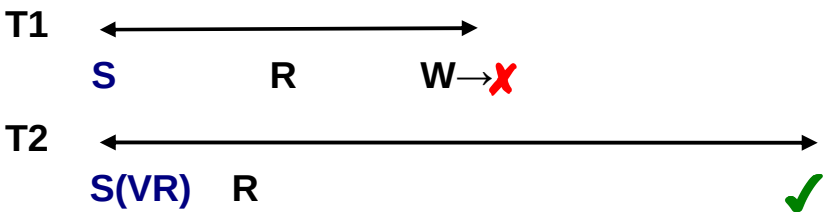
- ✓ Highest throughput
- ✓ Low overhead
- ✓ Fast reads
- ✗ Does not detect R/W conflicts until commit

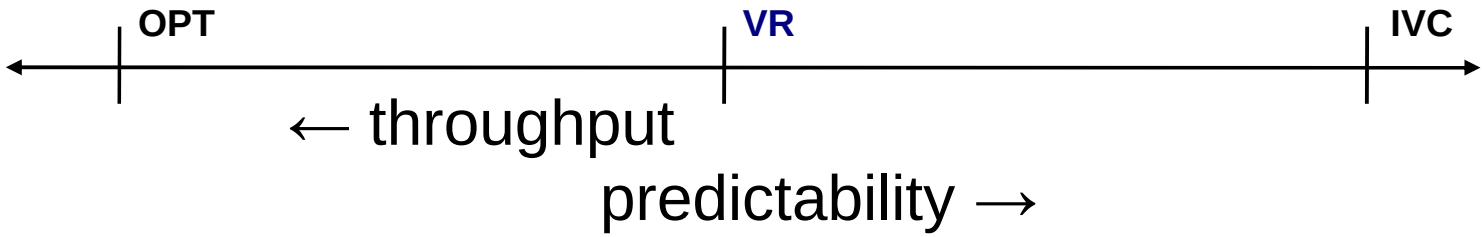




Visible Reads (VR)

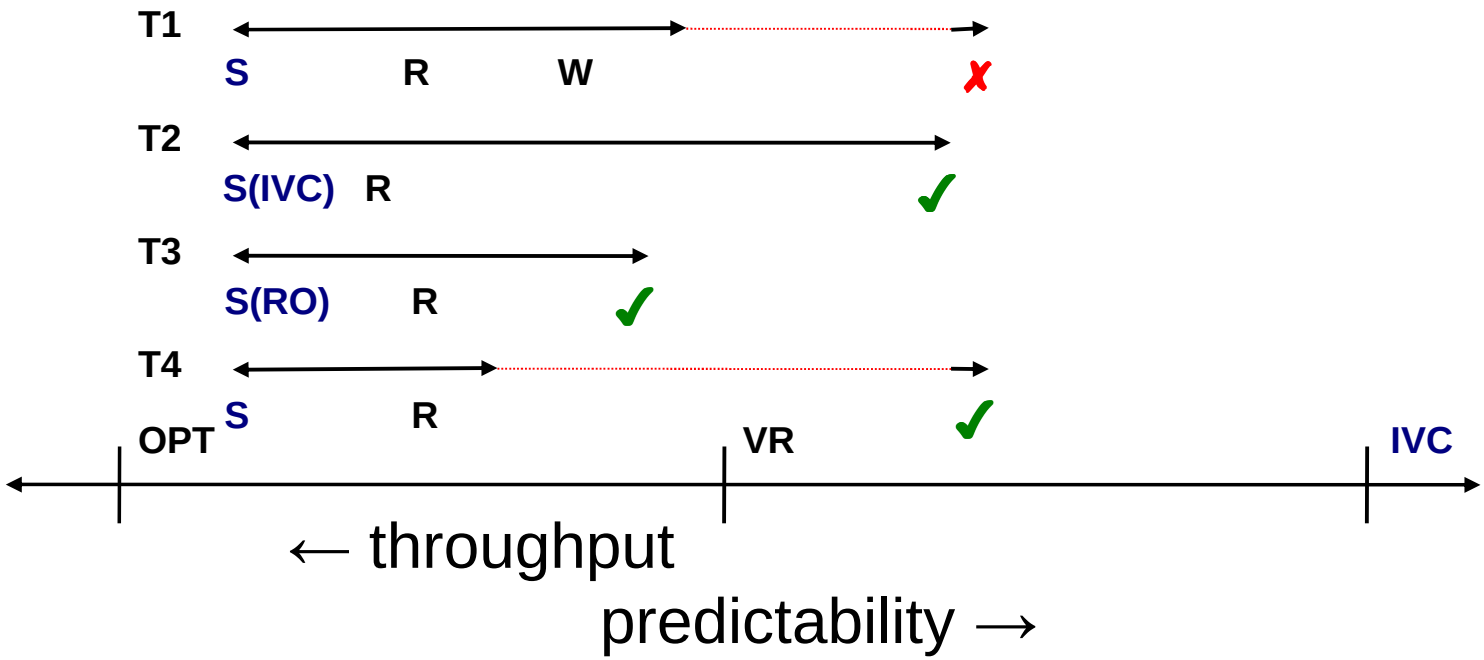
- ✓Reads can be seen by other transactions
- ✓Enables detecting R/W conflicts
- ✗High overhead for Reads





Irrevocable (IVC)

- ✓Ensures commit
- ✓Lowest overhead
- ✓Enables RO transactions to commit
- ✘Hinders parallelism



Implementation

- Monitoring of transaction execution time
  - Keep previous 10 000 executions
- Time slice extension at the kernel level
  - Disable time sampling in case of preemption
- Adaptive run-time system implemented in TinySTM

Results at a glance

- *Scheduling: less or no degradation when the number of thread increases*
  - Performance varies widely based on workload
  - Future work → a workload adapting contention manager
- *Soft-realtime: less degradation of non-real time transactions while maintaining 99% success rate for the real-time one*
  - Future work → support of multiple threads with deadlines

Detailed results

PP0PP 2010

Scheduling support for transactional memory contention management

DSN 2011

Deadline-Aware Scheduling for Software Transactional Memory