STM as a building block for parallel embedded real-time systems

Submitted to Euromicro Conference on Software Engineering and Advanced Applications 2011

António Barros

CISTER Spring Seminar Series 2011

15th of April, 2011

Research Centre in
Real-Time Computing Systems

INSTITUTO SUPERIOR
DE ENGENHARIA DO PORTO
POLITÉCNICO DO PORTO

Presentation outline

- Motivation
- Reducing contention with multi-version STM
- Providing RT guarantees with our contention management algorithm
- Conclusions

Motivation

The problem

- Traditional lock-based concurrency control (semaphores, mutexes) hinders performance of current and future foreseen parallel systems.
  - Coarse-grained locking serialises non-conflicting operations.
  - Fine-grained locking becomes too complex and error prone.

CISTER
Research Centre in Real-Time Computing Systems

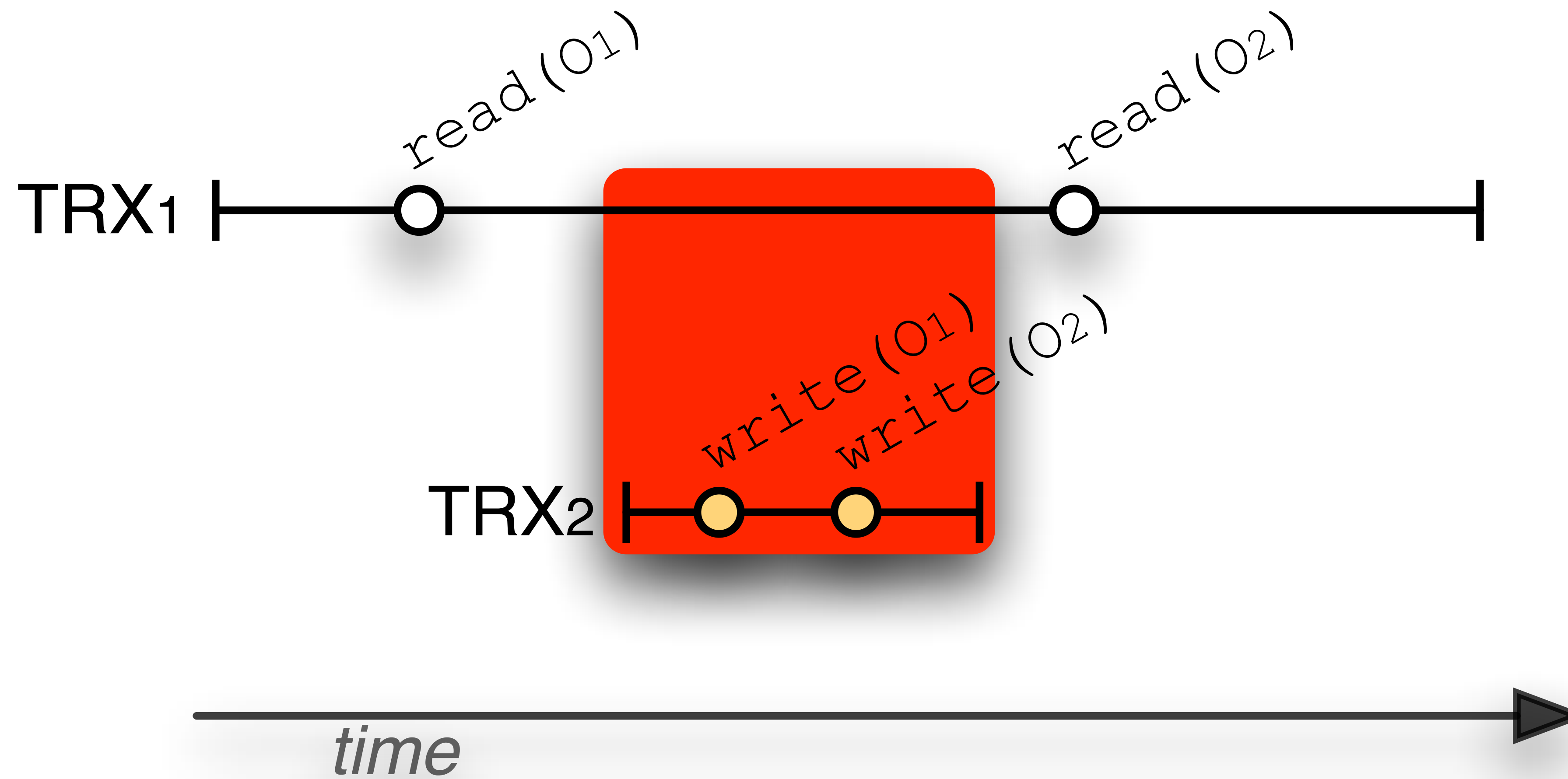
ISEP
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
POLITÉCNICO DO PORTO

Possible alternatives to lock-based synchronisation

- Transactional Memory
  - *"Transaction is a sequence of instructions, including reads and writes to memory, that either executes completely (commits) or has no effect (aborts)."*
    Tim Harris et al., "Transactional Memory: An Overview", 2007
  - Provides a higher-level abstraction for writing concurrent programs:
    - The programmer focuses on the algorithms.
    - The underlying TM mechanism deals with concurrency.

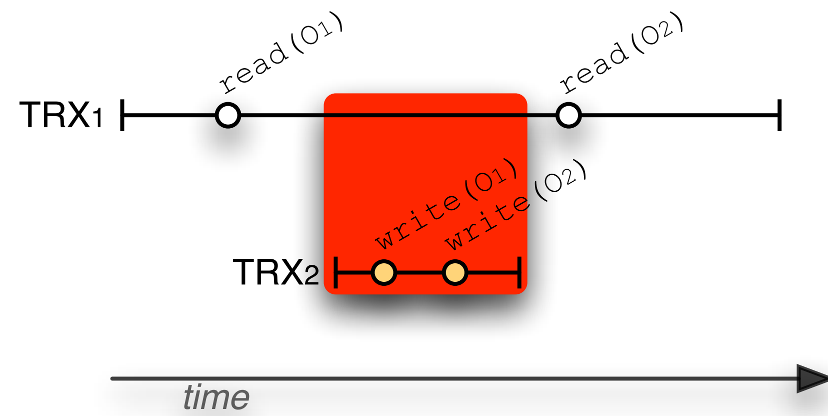
How does it work?

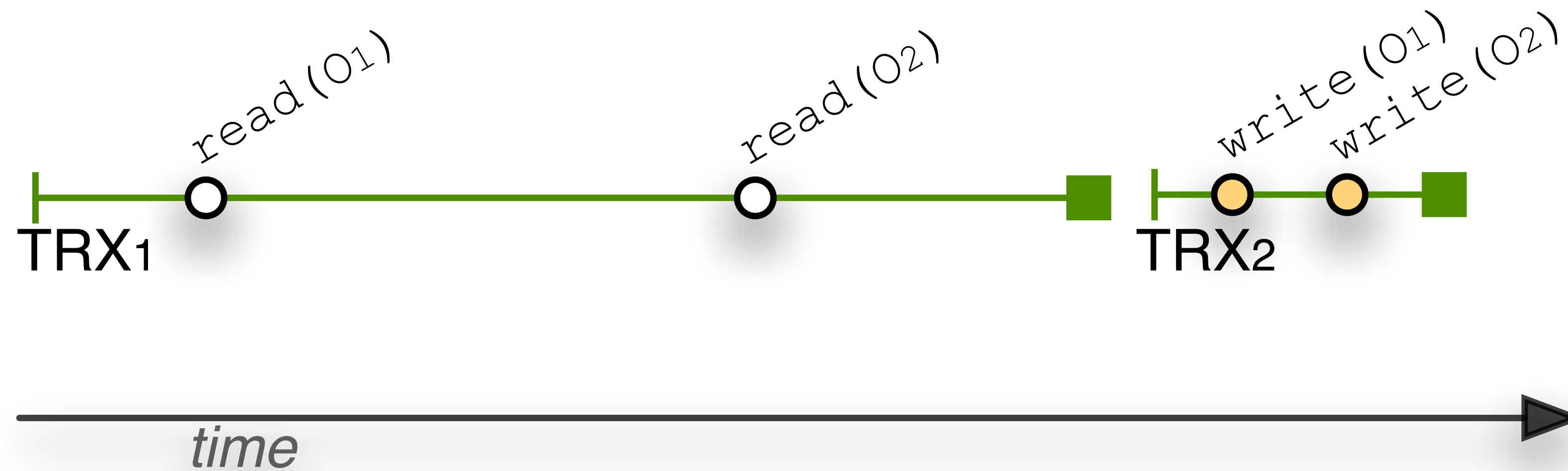
- Transactions are executed **concurrent** and **speculatively**, in **isolation**.
- The outcome of transactions must be serialisable.
  - If no conflicts occur, transaction commits.
  - A conflict dictates: **some one has to die**!!!



Conflicts between transactions

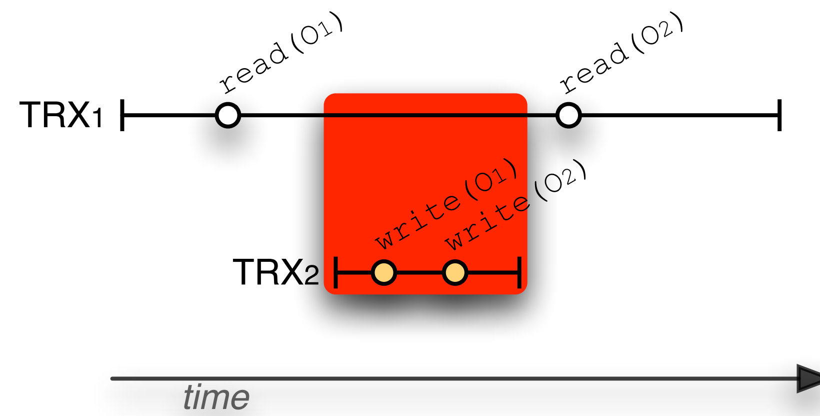
A simple example...

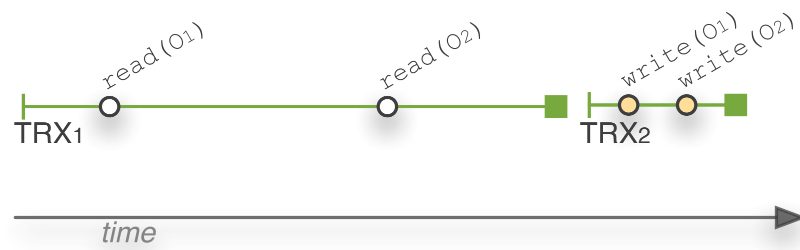


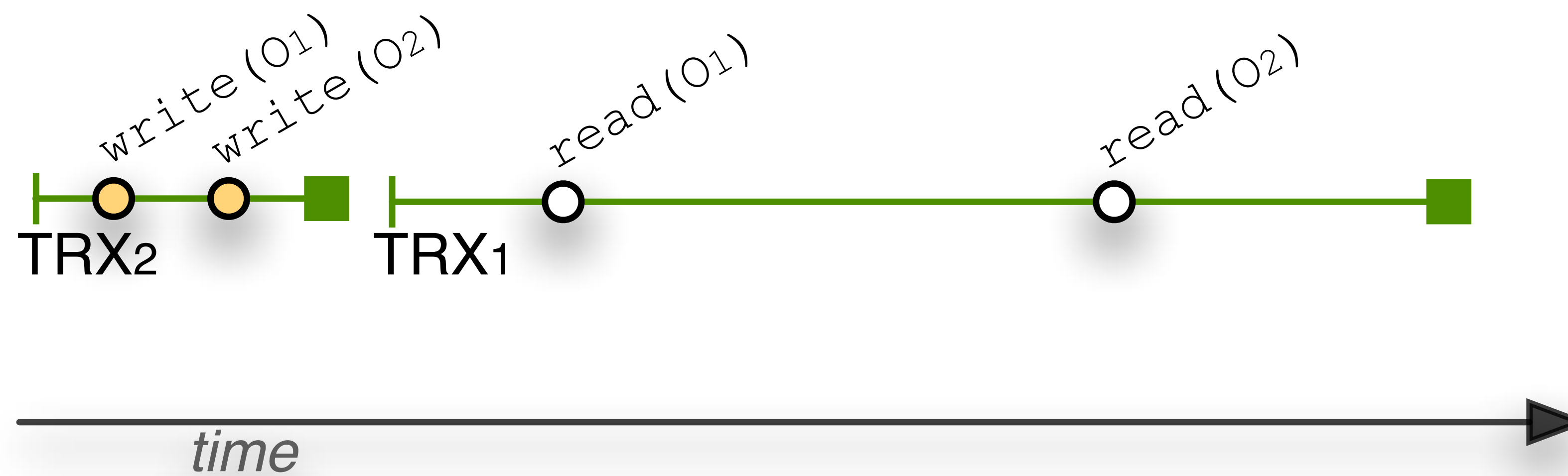


Conflicts between transactions

One possible solution







Conflicts between transactions

Another possible solution

CISTER Research Centre in Real-Time Computing Systems

ISEP Instituto Superior de Engenharia do Porto Politécnico do Porto

Role of a contention management policy

- Contention manager should:
  - Avoid live-lock.
  - Prevent starvation.

STM conflict detection/resolution

- STM comes in many "flavours"...
  - Word-based *vs.* object-based.
  - Write-through *vs.* write-back.
  - Early conflict detection *vs.* late conflict detection.
  - Contention mgmt policy...

Factors affecting the performance of STM

- STM is keen of low contention:
  - Predominance of read-only transactions.
  - Short-running transactions.
  - Low ratio of context switching during the execution of a transaction.

Using STM on real-time systems...

- Number of aborts suffered by a transaction affects:
  - *execution time* and the *processor utilisation* ratio of the host job.
- Number of aborts for each transaction must be **minimised**!
  - Intended to minimise wasted processor time.
- Number of aborts for each transaction must be **limited**!
  - Allows to calculate the WCET of a task.

Reducing contention with multi-version STM

Multi-version STM

- STM stores multiple versions of each shared data object.
- Read-only transactions read values from a consistent state.
  - Read-only transactions execute in a **wait-free** manner!
- Adversities:
  - Requires additional memory.
  - How many versions should be stored???

Multi-version STM in real-time systems

- Timing characteristics of a task set are known.
- Data access pattern for each task is known.
- It is possible to determine the **exact** number of versions for each data object!
  - Guarantees that read-only transactions will never abort!

Determining the number of versions required for a data object

- Determine the maximum period of read-only transactions that access the data object.

$$T_k^{store} = \max\{T_i : O_k \in RS_i \wedge TRX_i \text{ is RO}\}$$

- Determine the maximum possible committed number of writes on the data object, during the time interval determined before.

$$N_k^{versions} = \sum_i a_i \times \left\lceil \frac{T_k^{store}}{T_i} \right\rceil$$

$$a_i = \begin{cases} 1 & \text{if } O_k \in WS_i, \\ 0 & \text{otherwise.} \end{cases}$$

CISTER Research Centre in Real-Time Computing Systems

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO POLITÉCNICO DO PORTO

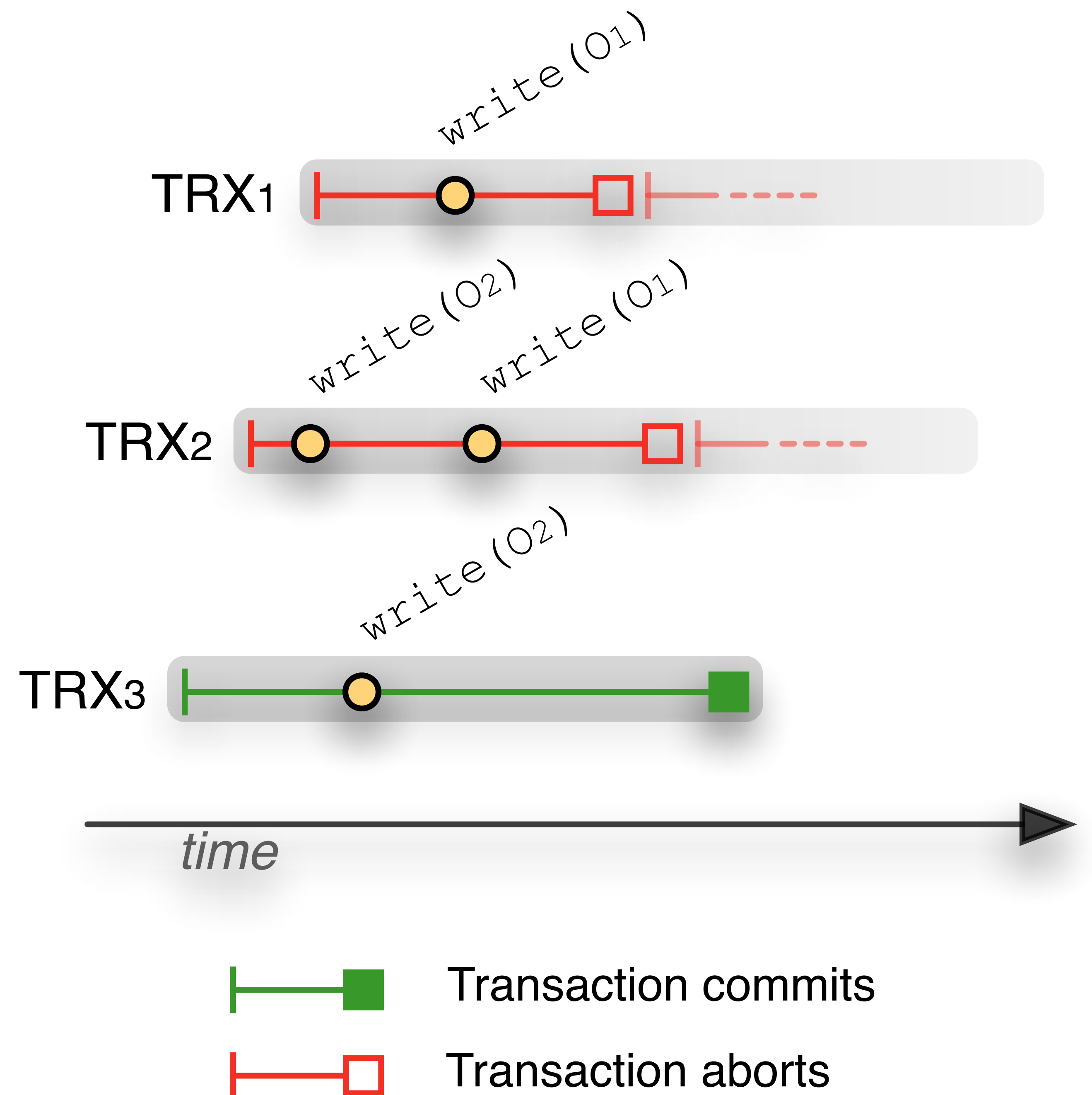
Providing RT guarantees with our contention management algorithm

Proposed contention management policy

- Conflicting transactions commit by order of arrival.
  - **Liveliness**: balances aborts between transactions.
  - **Predictability**: abort overhead depends on the set of active transactions at the moment the transaction arrives.
  - **Distributed**: all transactions reach a consensus on which transaction should commit.
- Ties are solved by slack and, ultimately, by processor ID.
- **Exception**: transactions executing on the same processor.

Parallel conflicts

- Transaction aborts due to direct contenders that **arrived earlier**, and **are running**.
  - However may have to wait for “indirect” contenders...



CISTER Research Centre in Real-Time Computing Systems

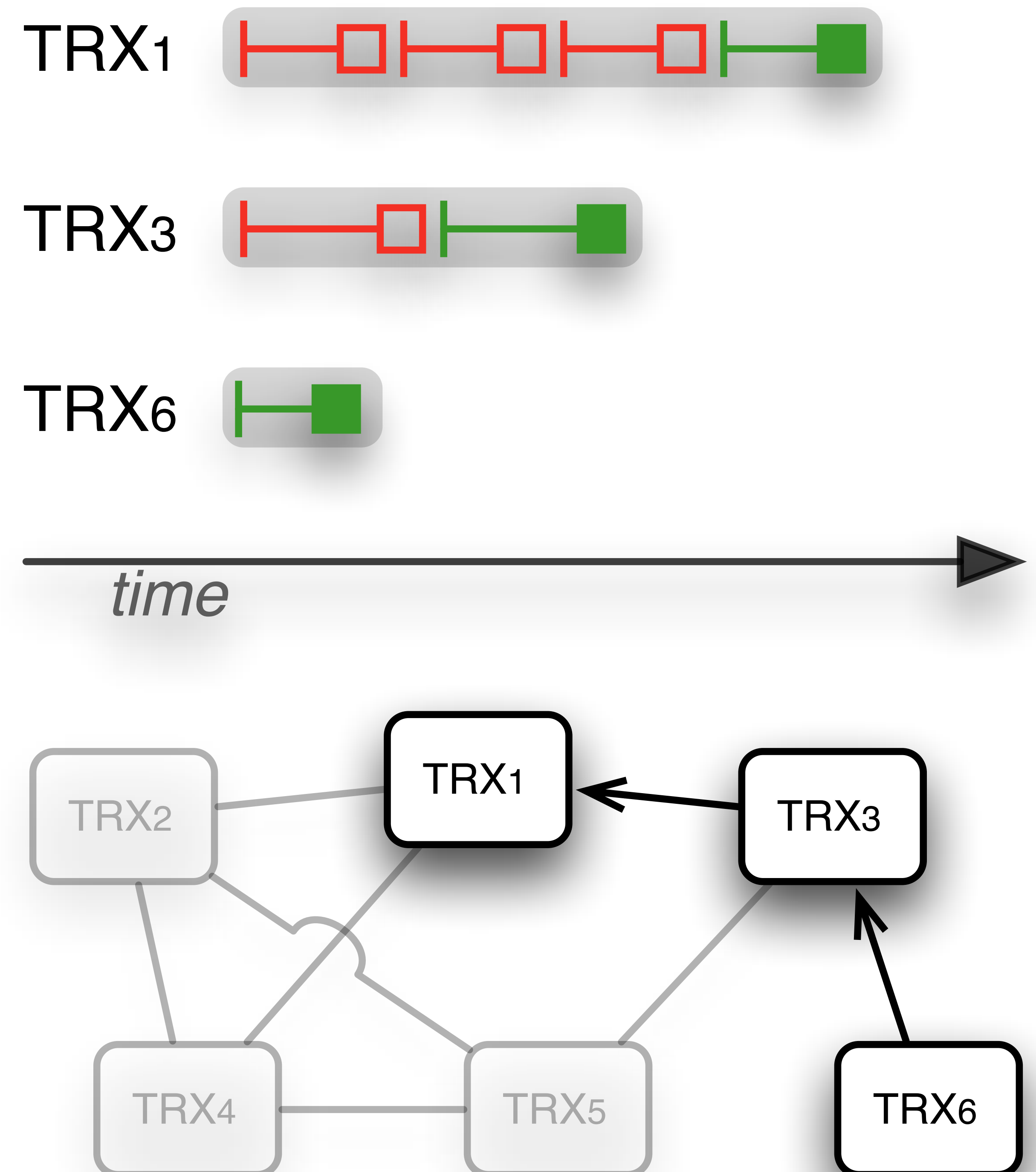
ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO POLITÉCNICO DO PORTO

Parallel conflicts

- Establish a graph connecting direct contenders.
- For each transaction, determine the maximum amount of waiting time, from all the *simple paths* diverging from the transaction vertex.

$$tc_n = \begin{cases} W_i & \text{if } n = 1, \\ \left(\left\lceil \frac{tc_{n-1}}{W_i} \right\rceil + 1 \right) * W_i & \text{if } n > 1. \end{cases}$$

$$W_i^{overhead_p} = max\{tc_n : vertex_{n,i}\} - W_i$$



CISTER
Research Centre in Real-Time Computing Systems

ISEP
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
POLITÉCNICO DO PORTO

Intra-processor conflicts

- Transaction will abort, at most, once per pre-emption.
- Maximum number of possible pre-emptions..

$$n_i^{preempt} = \sum_j \left\lfloor \frac{T_i}{T_j} \right\rfloor \quad \forall j \neq i \wedge P(\tau_i) = P(\tau_j)$$

- ... leads to maximum overhead due to intra-processor conflicts.

$$W_i^{overhead_c} = n_i^{preempt} \times W_i$$

$$C_i = C_i' + W_i^{overhead_p} + W_i^{overhead_c}$$

And the WCET of a task can be upper bounded!

Conclusions

Conclusions

- Multi-version STM
  - Timing characteristics of RTS allows to determine the exact number of versions for each data object.
- Proposed contention management algorithm.
  - Provides liveliness, predictability and distributed consensus.
  - Allows to establish an upper bound of the WCET of each task.

Thank you, for your attention!

Questions?

Remember: there is pizza after this seminar!!!