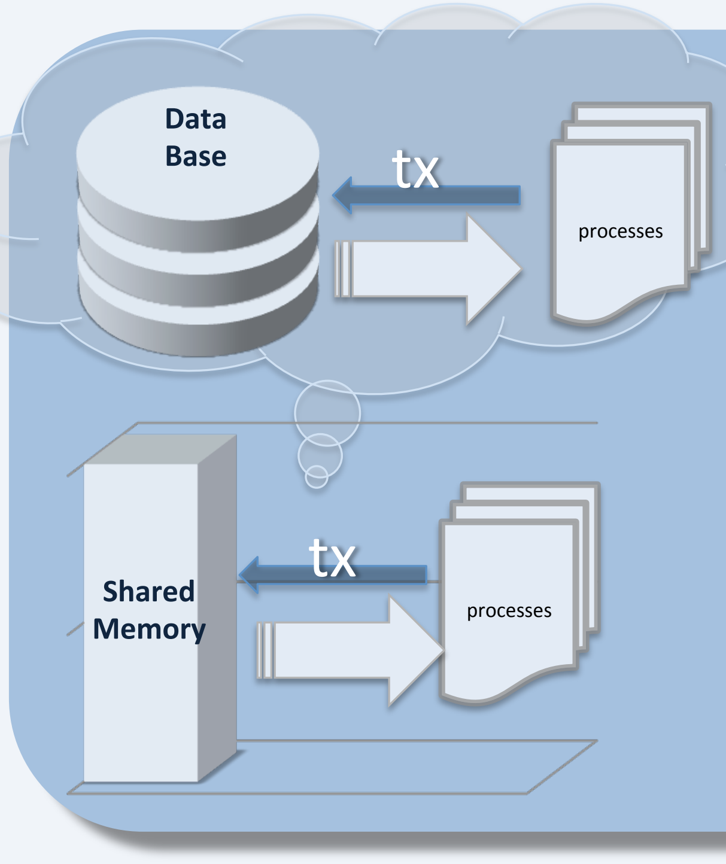
How to Commit More Transactions ?

Eleni Kanellou - Université de Rennes 1

Transactional Memory



Transactional Memory (TM) adapts the concept of atomic accesses to multiple locations – a concept expressed through the *transaction* in databases – for use in a multi-process, shared memory system.

The aim…

Relieve the programmer of the need to take care of synchronization.

- ❑ Hide the synchronization details in the transaction abstraction.
- ❑ Provide an implementation for the transaction.

➢ The programmer encapsulates those memory accesses that have to happen atomically, *inside a transaction.*

STM:
Software Transactional Memory

HTM:
Hardware Transactional Memory

HyTM:
Hybrid Transactional Memory

The memory transaction:
An atomic procedure

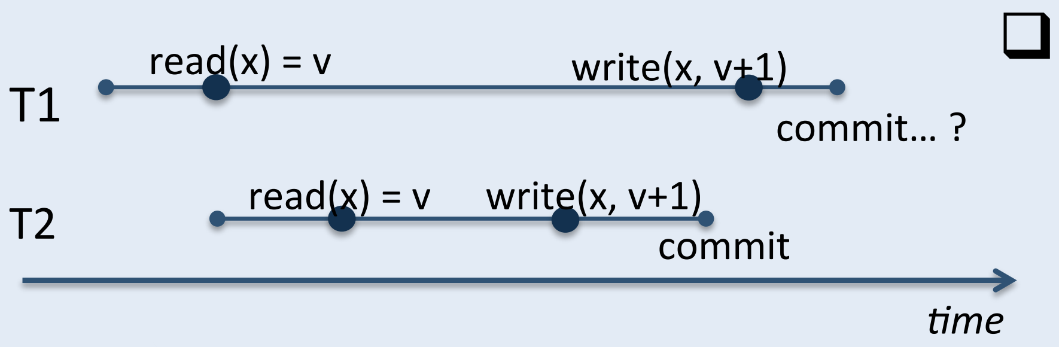
- ❑ Commonly, *reads* and *writes* shared memory locations.
- ❑ Those reads and writes appear to have happened *all, instantaneously* or *not at all*.
  - ○ i.e., the transaction *commits* or *aborts*.

When *should* transactions abort?

- ❑ Roughly speaking, a concurrent execution of transactions is considered correct when it is equivalent to a correct sequential execution.
- ❑ When this cannot be guaranteed, a transaction has to be aborted.

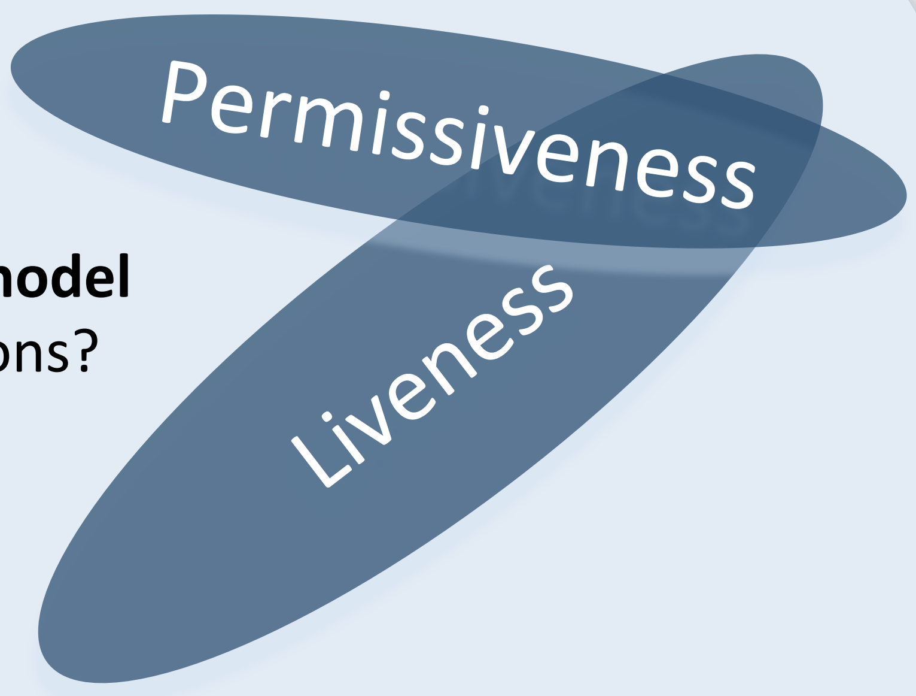
Why does an STM system abort transactions?

- ❑ Correctness is violated by the current execution.
- ❑ It is uncertain if correctness can be guaranteed for the future of the execution.
- ❑ There is presence of failures in the system.



- ❑ «Better safe than sorry» : The implementation is more efficient if it preemptively aborts transactions in case of doubt.

Our Focus



- ❑ What **characteristics** of an **STM model** can satisfy good progress conditions?
  - ❑ Improving **liveness**.
- ❑ What **characteristics** of an **STM implementation** can avoid unnecessary aborts?
  - ❑ Improving **permissiveness**.

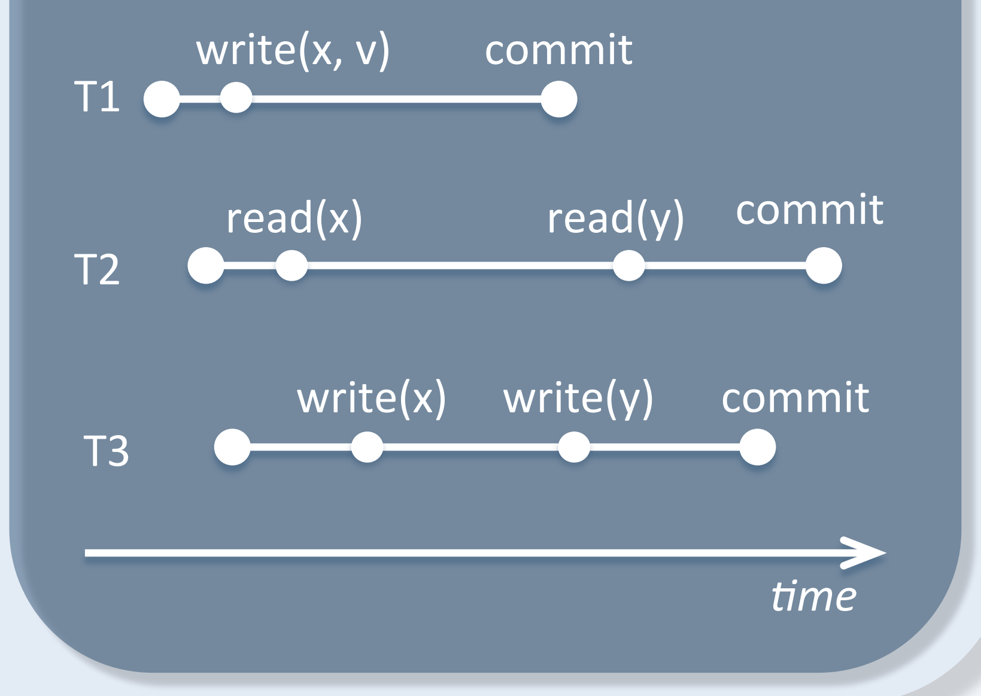
The Intended Outcome

- ❑ How to **hide** the *abort-retry* mechanism from the programmer?
- ❑ Is it possible to **avoid** the *abort-retry* mechanism all together?
- ❑ i.e., how to make transactions execute **exactly once** and terminate?

Existing solutions in this direction

- ❑ « Helping Mechanisms » : In case of conflict with transaction T_x, transaction T_y helps it complete its operations and commit.
- ❑ « Pessimistic Execution »: The system imposes sequential execution on conflicting transactions.
- ❑ « Probabilistic permissiveness » : Transactions negotiate their commit point, in order to avoid unnecessary preemptive aborts.

Also: Multiple Versions



What to do next?

- ❑ What restrictions are imposed by the limitation of transactions to read and write operations?
- ❑ What makes an operation suitable for the use inside transactions?
- ❑ Can more complex operations be « transactionalized »?

Research funded by

TRANSFORM
Theoretical Foundations of Transactional Memory