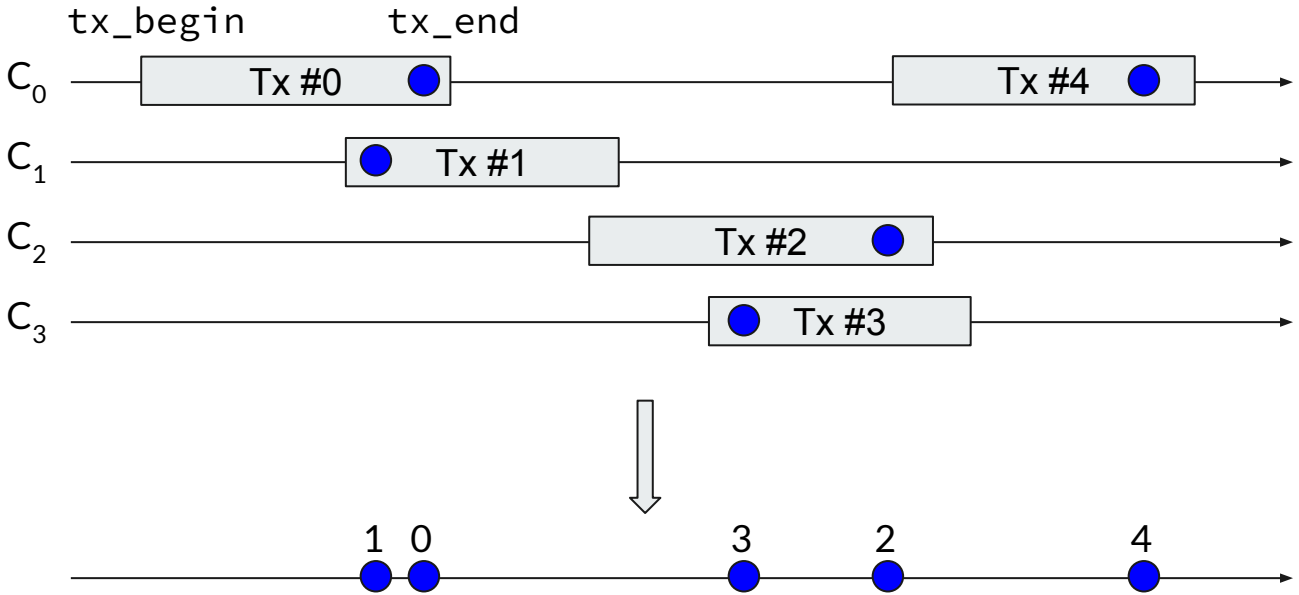
CS-453 - Project
Dual-versioning STM

Distributed Computing Laboratory

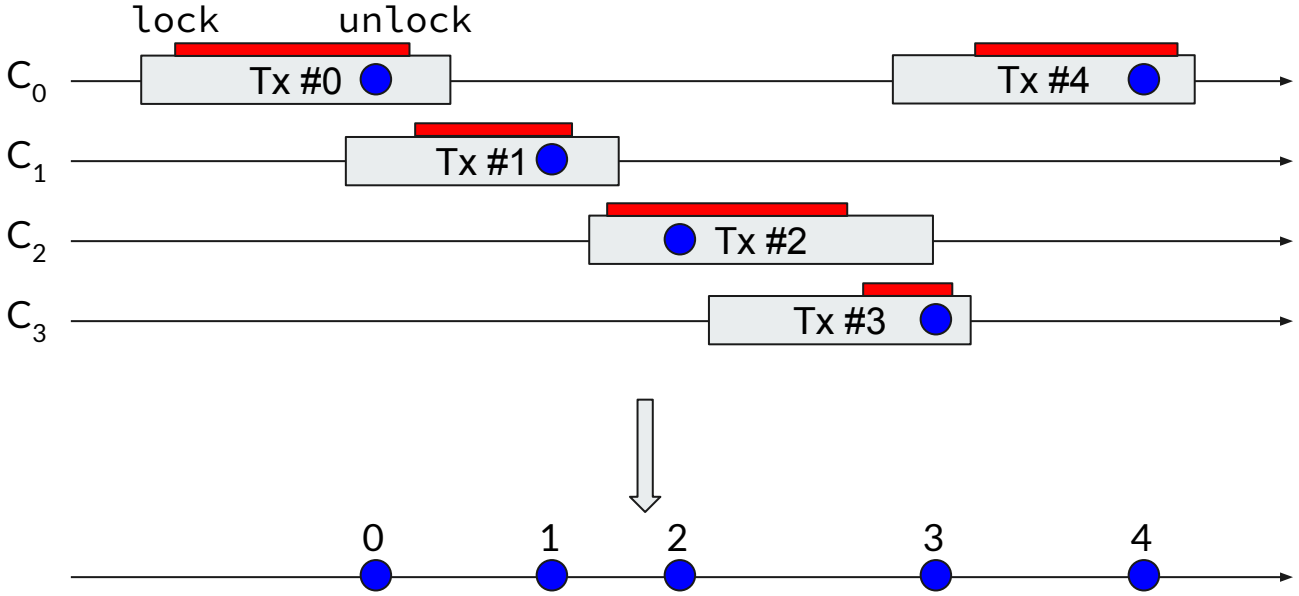
September 24, 2024

STM: Serializing transactions



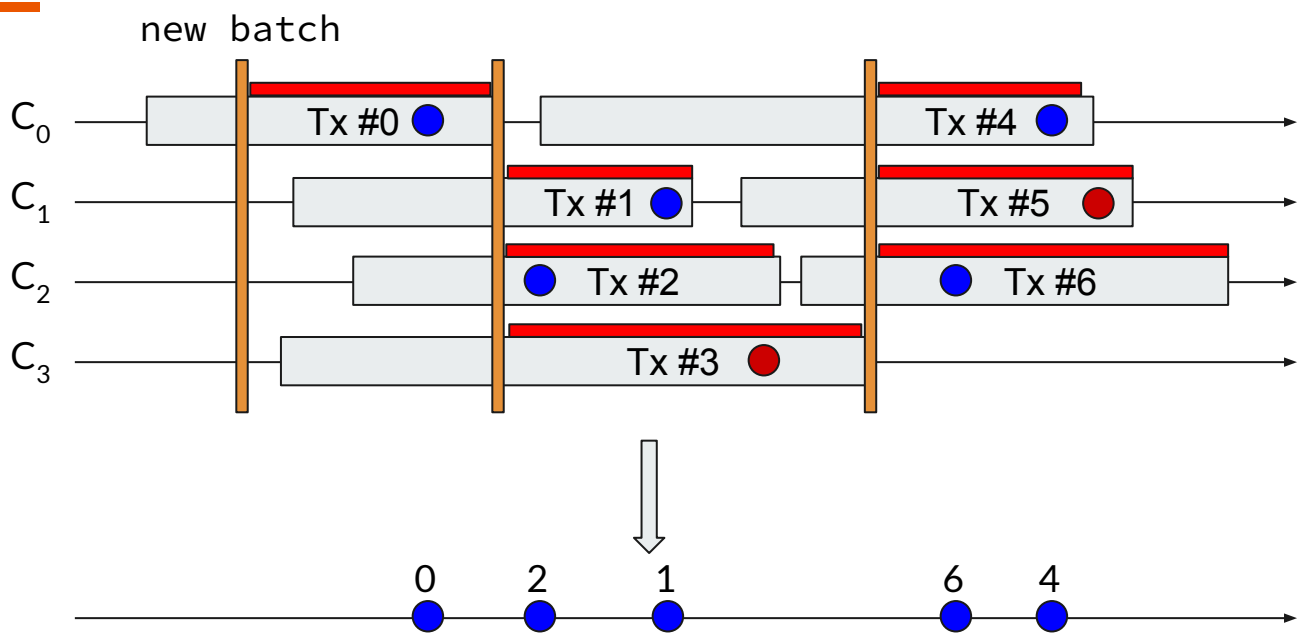
- The execution on multiple cores is equivalent to a *serial* execution on a single core.
- The (atomic) execution points are between the start and the end of each transaction: *strict serialization*.

STM: With a coarse-grained lock



- Works, but only one transaction executes at a time.
- Transactions never fail.

STM: Batch & detect conflicts

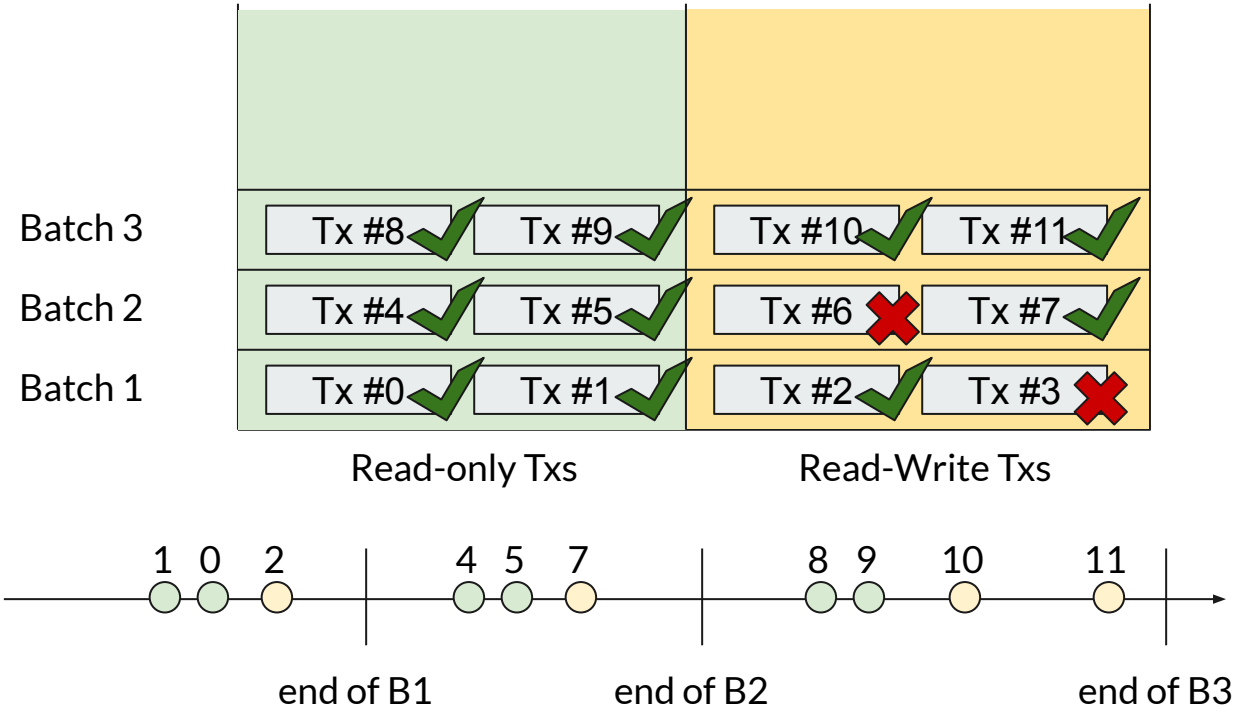


- A tad slow because of the batching, but allows concurrent executions.
- Transactions can fail if they conflict.

STM: Dual-versioning, briefly

- **Observation**: If all Txs were read-only (RO), no need to check conflicts! ⚡⚡⚡
- **Problem:** We also need to run read-write (RW) Txs :(
- **Solution:** Run RO Txs on a RO copy of memory, while RW Txs fight to update a RW copy.
  1. Batch Txs for a little while,
  2. Run all RO Txs on the RO copy.
  3. In parallel, synchronize all RW Txs on the RW copy.
  4. Once every Tx (tentatively) executed, update the RO copy to match the RW.
- **Notes**:
  - RO Txs never fail and are *serialized before* RW Txs.
  - RW Txs can fail if they conflict.

Dual-versioning batching visualized



- All Txs in a batch run in parallel.
- RO Txs never fail and are *serialized* before RW Txs of the same batch.
- RW Txs can fail.
- RW Txs write to their own copy of memory (not to disturb RO Txs).
- The RO memory copy is updated at the end of a batch.
- Will get you a good grade, but you need to play some tricks to get a 6.

How to reference memory in the dual-versioned STM?

- Usually, memory references are **pointers** containing (***virtual) addresses***.

- But, in the case of the dual-versioned STM:
  - Every piece of memory exists in 2 versions.
  - We still want pointer arithmetic to be valid.

- ***What should my STM pointers contain?***

- Good news:
  - The pointers are ***created by your STM*** (i.e., when allocating memory).
  - The pointers are only ever ***used by your STM*** (via `tm_read`/`tm_write`/etc.).
  - They ***don't*** need to ***hold*** *valid (virtual)* ***addresses*** (we never dereference them).
  - You just need ***your STM*** to understand them.