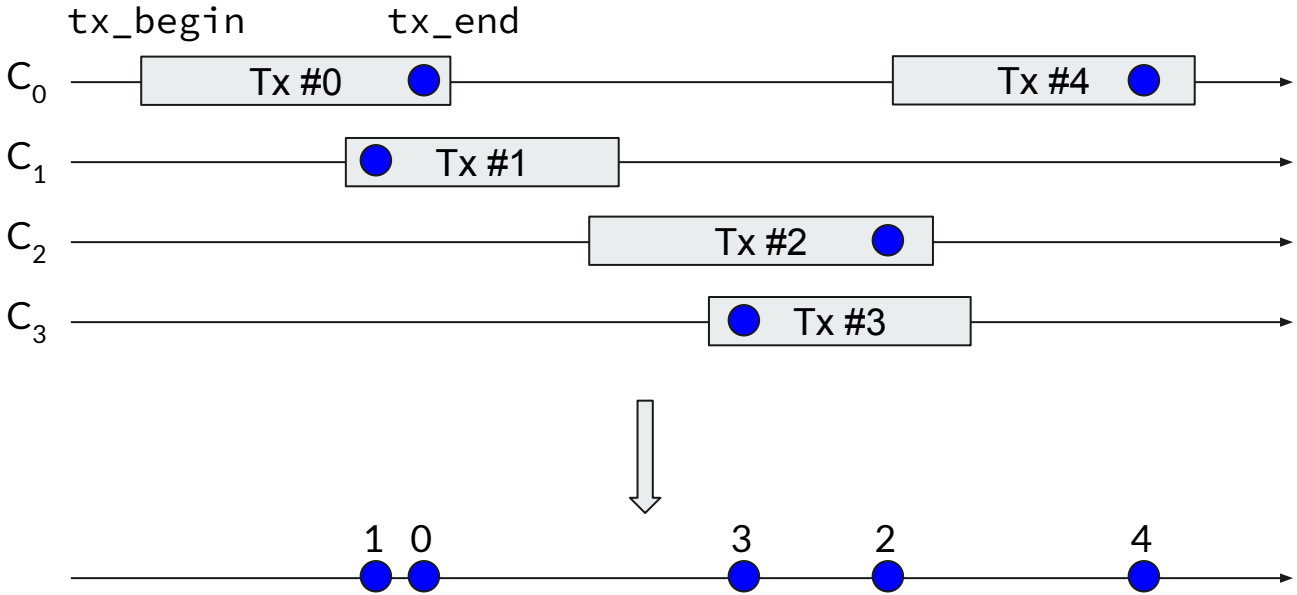
CS-453 - Project
Dual-versioning STM

Distributed Computing Laboratory

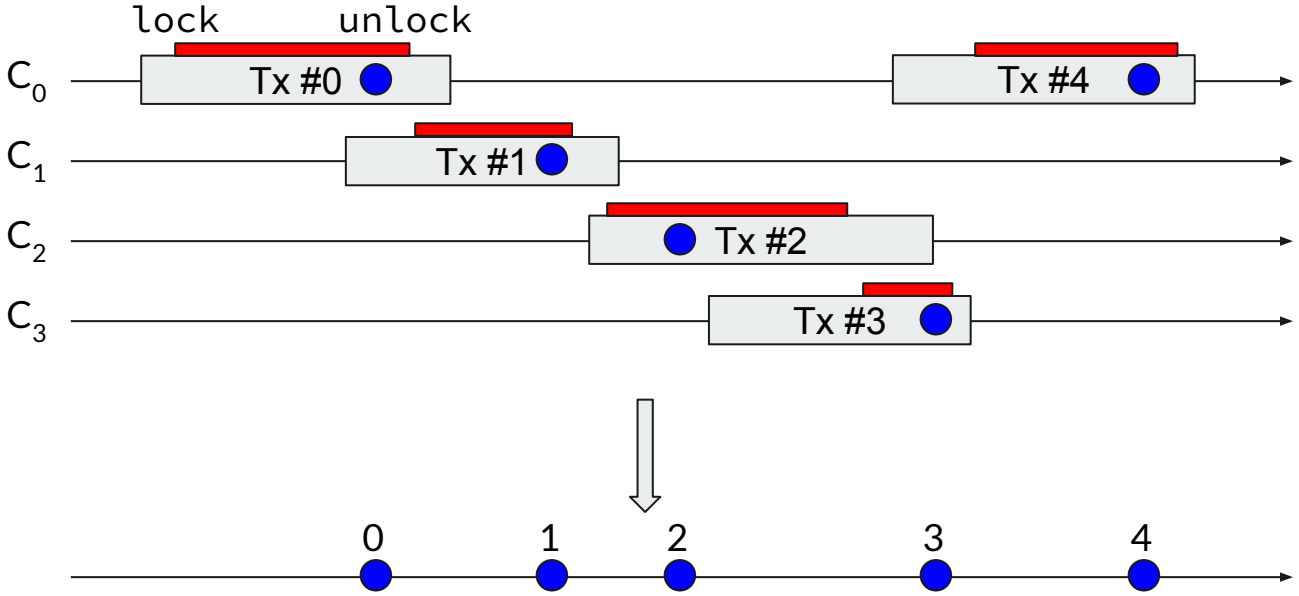
September 26 2023

STM: Serializing transactions



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

STM: With a coarse-grained lock

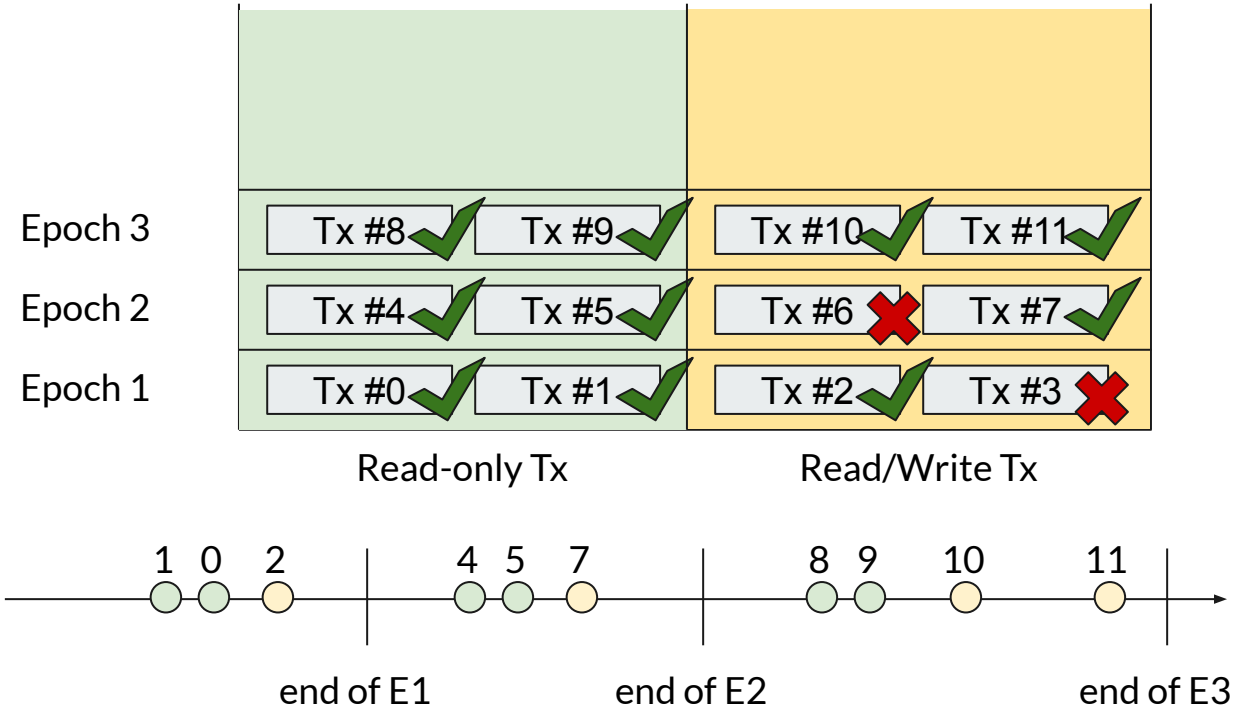


- Works but only one transaction is really running at a time.
- Transactions never fail.

Dual-versioning STM in short

- Each memory word has 2 versions:
  - One accessed by read-only (RO) transactions
  - One accessed by read-writes (R/W) transactions.
- Transactions are run (in parallel) in batches called “epochs”.
- RO transactions never fail and are *serialized before* R/W transactions.
- R/W transactions may fail if they conflict.
- Modified words see their versions swapped at the end of the epoch.

Dual-versioning STM: the batcher



- All Tx in an epoch run in parallel.
- Epoch N+1 starts after the end of Epoch N (i.e., when all Tx ended).
- Read-only Tx never fail and are *serialized* before R/W Tx of the same epoch.
- R/W transactions may fail.
- R/W Tx write to a different memory from the read one:
  R/W cannot cause RO to abort.
- Modified words see their versions swapped at the end of the epoch..
- Will get you a good grade, but not a 6, you need to play some tricks.

How to reference memory in STM?

- In *regular* programming, you reference memory via **pointers** containing (***virtual) addresses***.

- But, in the case of STM:
  - Every piece of memory exists in 2 versions.
  - Every word (~*8* bytes of memory) has metadata (e.g., who accessed it in the epoch).
  - We still want pointer arithmetic to be valid.

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