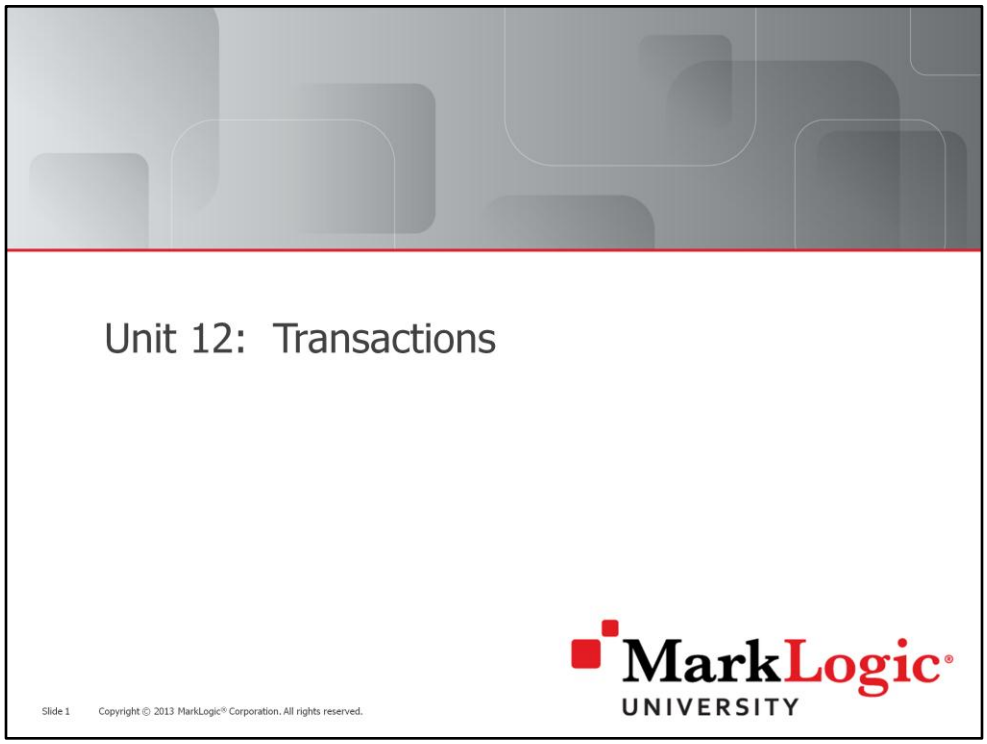


Learning Objectives

- Describe MVCC (Multi-Version Concurrency Control)
- Perform insert, update, delete and query transactions
- Describe merges
- Mange merge policy

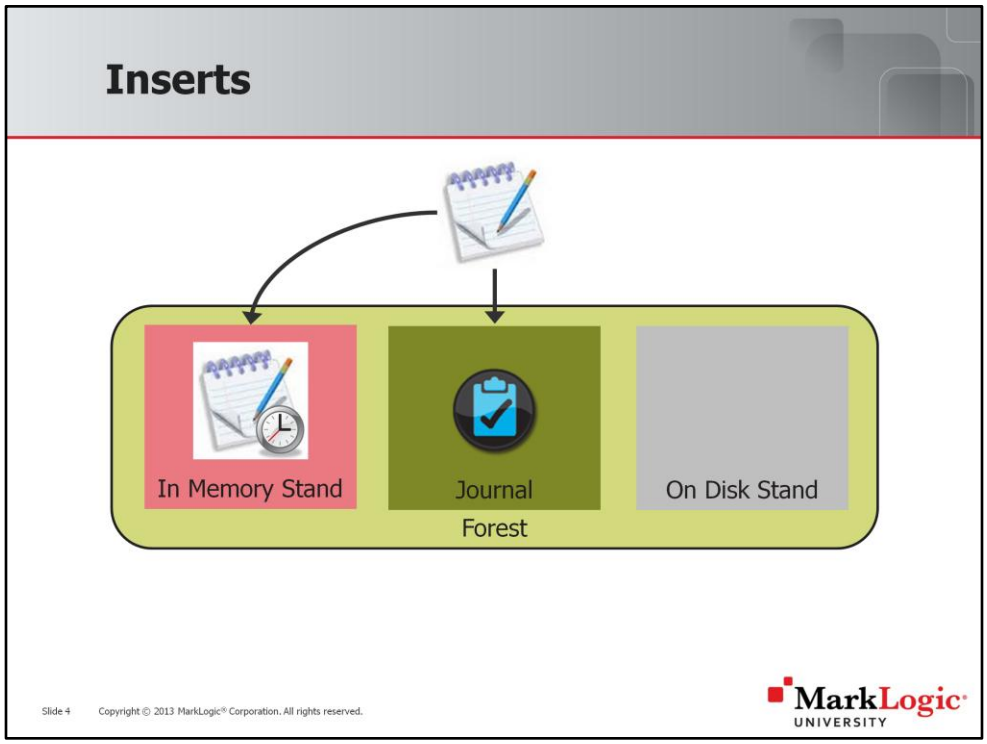
MarkLogic UNIVERSITY

ACID

- Atomicity
  - Either all operations of the transaction are correctly executed or none are executed
- Consistency
  - Database will remain in a consistent state after the transaction commits
- Isolation
  - In a concurrent transactional system transactions are unaware of each other
- Durability
  - After a transaction completes, changes persist even if the system fails

Slide 3

MarkLogic UNIVERSITY

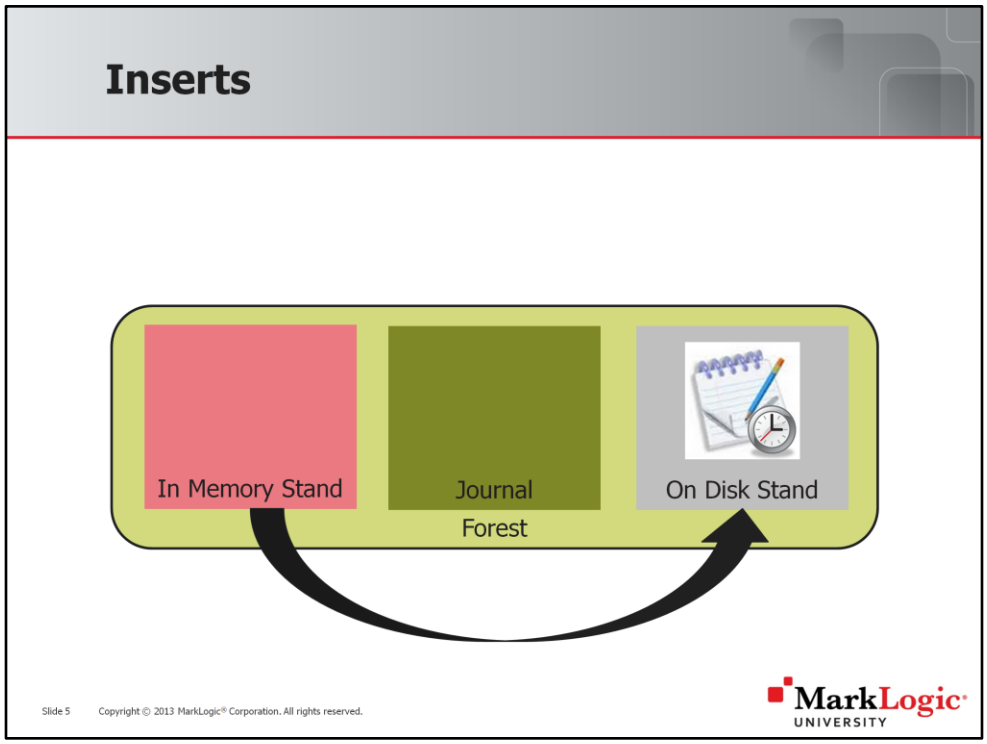


When a document is inserted, or ingested, it goes into the database as part of the in memory stand. The document gets created and populated with a “Create” timestamp. Conceptually it will also have a “delete” timestamp, although this will be empty until the doc gets updated or deleted.

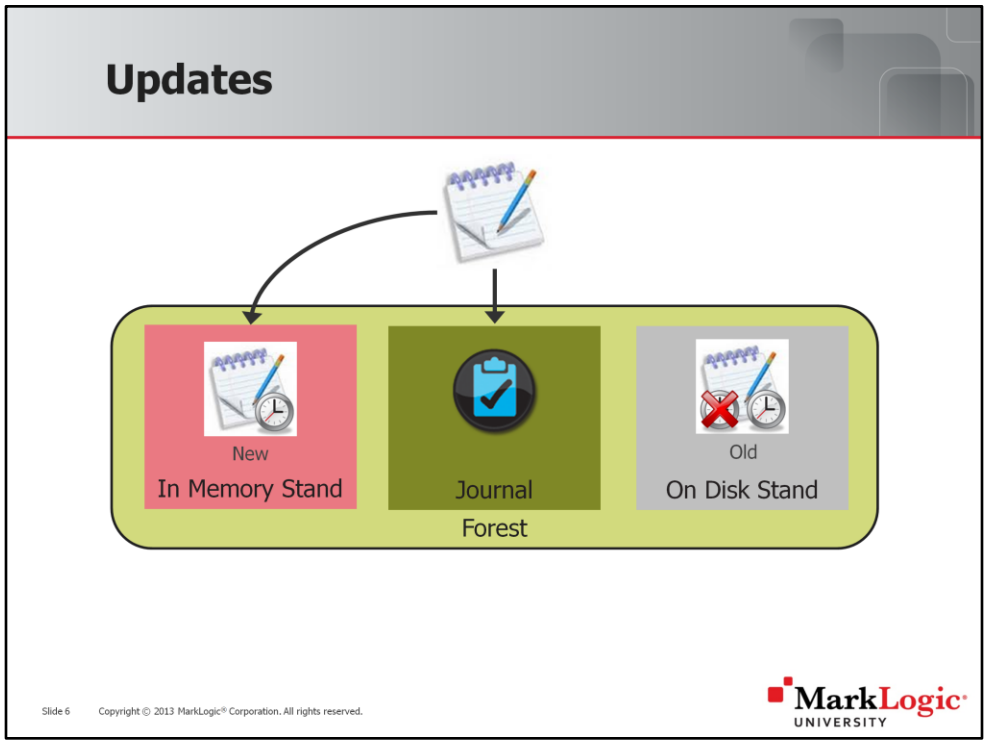
At this same time an entry is made in the journal, which would allow for the in memory stand to be rebuilt in the case of a power failure or other catastrophic event.

If a query is run it will have access to the data in both the in memory stand and the on disk stand.

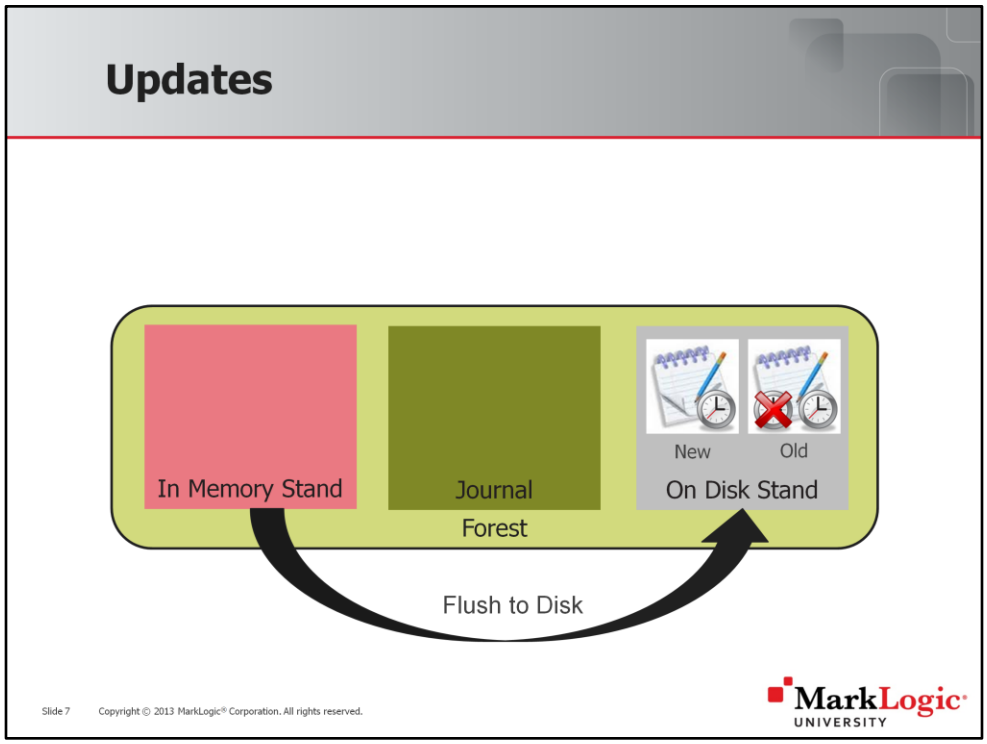
When the in memory stand gets full, or the server finds an opportune moment (like a period of low disk I/O), the in memory stand will get flushed to the on disk stand.



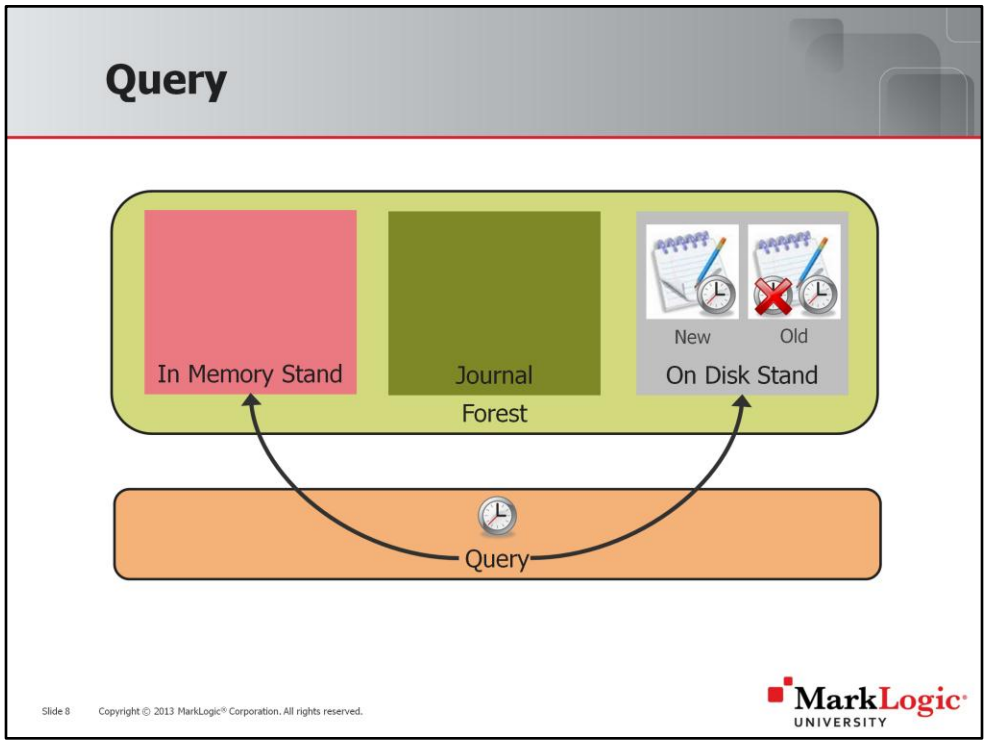
When the in memory stand gets full, or the server finds an opportune moment (like a period of low disk I/O), the in memory stand will get flushed to the on disk stand.



When we do an update, a new version of the document gets created, with a new "create" timestamp. So an update is actually an "up-sert" in that we are inserting a new, updated version of the document. The old version still exists on disk, but gets populated with a "delete" timestamp. The old version is no longer available to queries.

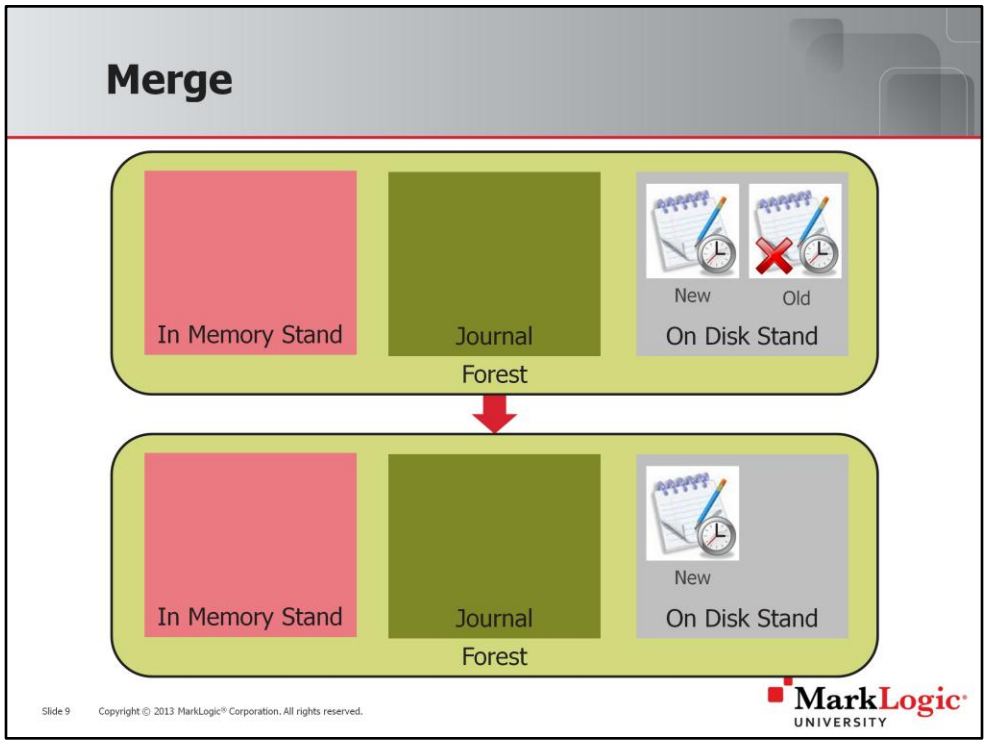


When the in memory stand gets flushed to disk, both versions of the document will exist in the on disk stands (the reality is they will probably be on different on disk stands, but conceptually this is what happens).



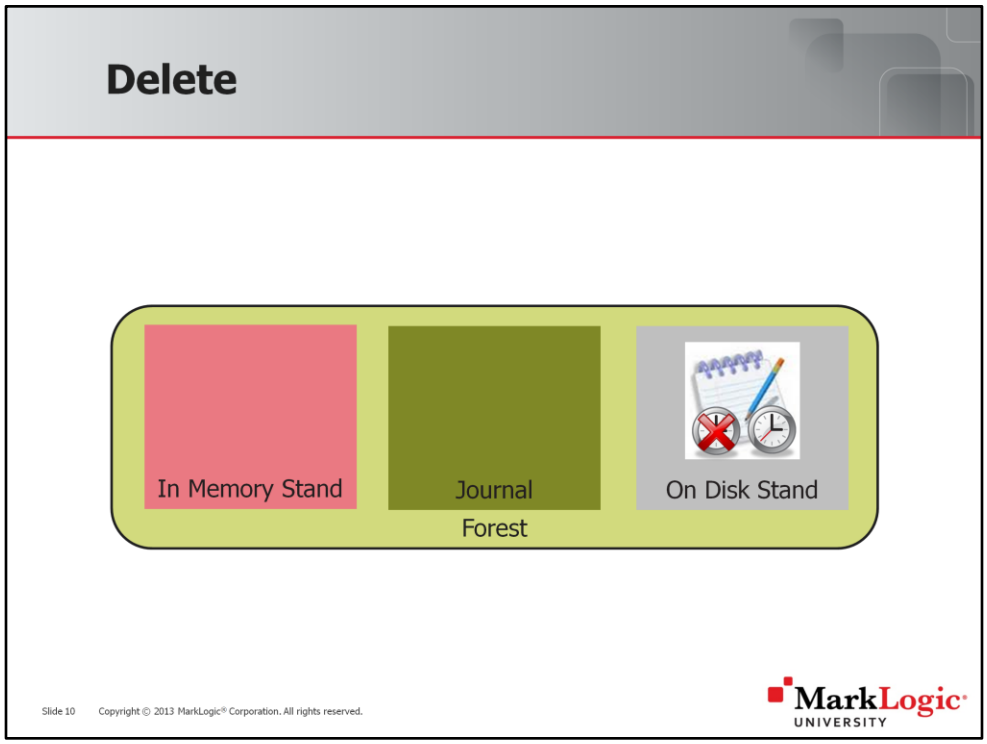
When a query runs, it also gets a timestamp and runs against a view of the database at that time. This prevents MarkLogic from having to lock documents, making it perform much better.

The query will not use the old version of the document because it has been marked for deletion. If an updated version of a document exists, either in memory or on disk, the query will use the appropriate version.

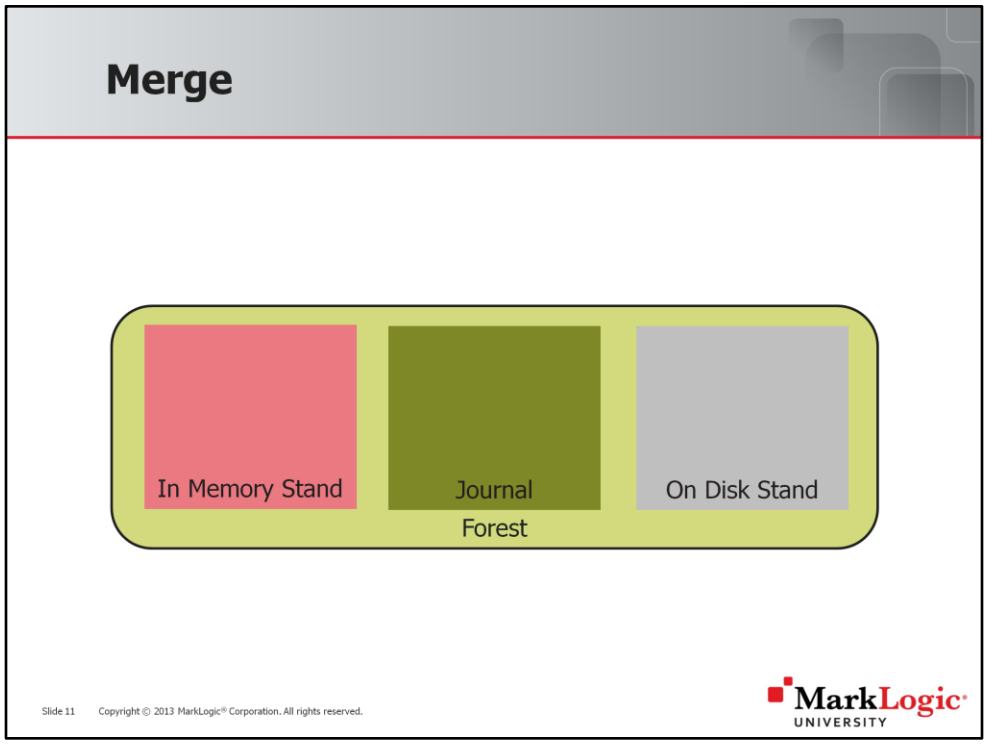


When a merge happens, the versions of the documents with the delete timestamps are purged.

Merging happens by default. The server makes the decision and optimizes the operation. Merge blackout periods can be implemented by the user (for example, don't merge during the daytime core business hours). This could cause more problems, and experienced MarkLogic engineers tend to prefer letting the server handle merges as it sees fit.

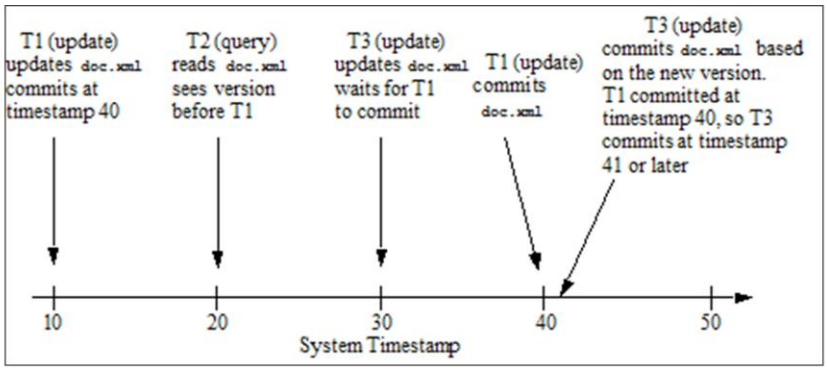


When a document is deleted, it is marked as such but not permanently removed yet. When marked as deleted it will not be available to any queries that run against the database.



When a merge happens, the versions of the documents marked for delete are purged.

Example Transaction Timeline



MarkLogic UNIVERSITY

Assume T1 is a long-running update transaction which starts when the system is at timestamp 10 and ends up committing at timestamp 40 (meaning there were 30 updates or other changes to the system while this update statement runs).
When T2 reads the document being updated by T1 (doc.xml), it sees the latest version that has a system timestamp of 20 or less, which turns out to be the same version T1 uses before its update.
When T3 tries to update the document, it finds that T1 has readers/writers locks on it, so it waits for them to be released. After T1 commits and releases the locks, then T3 sees the newly updated version of the document, and performs its update which is committed at a new timestamp of 41.

In Memory Stand Tuning

- In-Memory Stand limits
  - In Memory Limit (**fragments**)
  - In Memory List Size (**indexes**)
  - In Memory Tree Size (**xml**)
  - In Memory **Range Index** Size
  - In Memory **Reverse Index** Size
- Mark Logic adjusts defaults to system memory at installation
- Changes apply to all forests within the database
- Indexes + xml+ range <= journal

Unit 12: Applying the Learning Objectives

- Describe MVCC (Multi-Version Concurrency Control)
- Perform insert, update, delete and query transactions
  - Exercise 1
  - Exercise 2
  - Exercise 3
  - Exercise 4
  - Exercise 5
  - Exercise 6

MarkLogic UNIVERSITY