Unit 11: Transactions

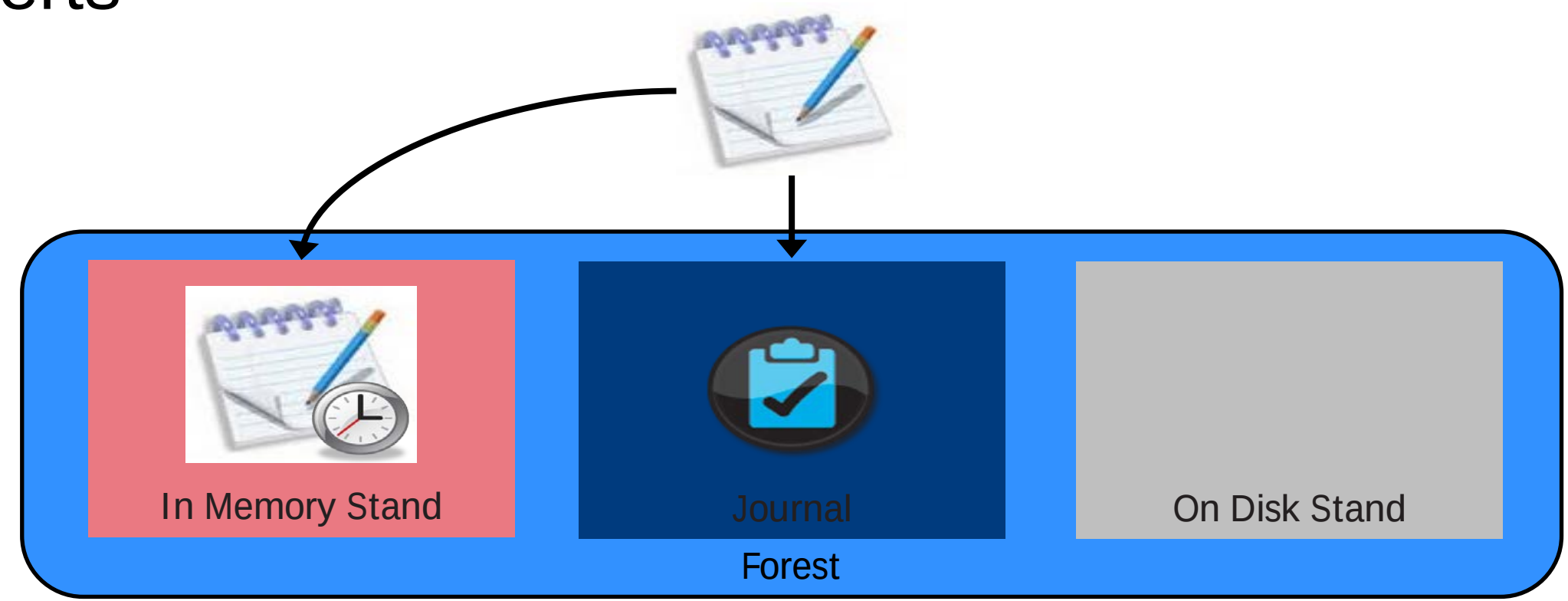
MarkLogic

Learning Objectives

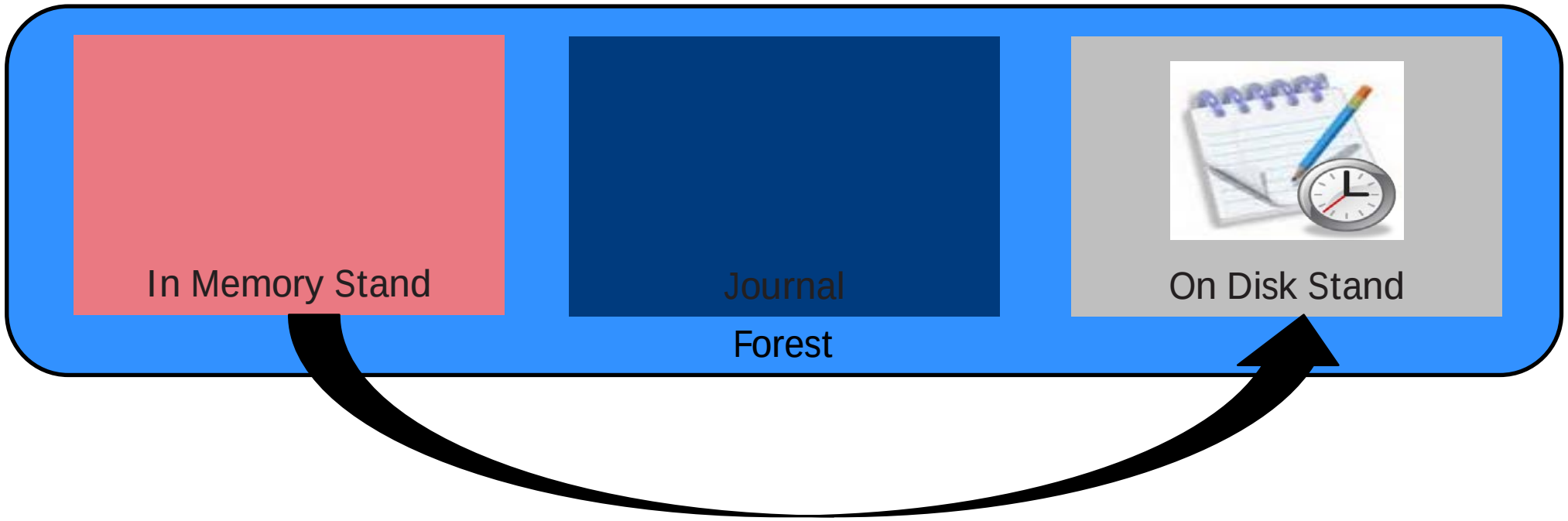
- Describe MVCC (Multi-Version Concurrency Control).
- Perform document updates.
- Perform patch (partial) updates.
- Perform multi-statement transactions.

MarkLogic

Inserts



Inserts



MarkLogic

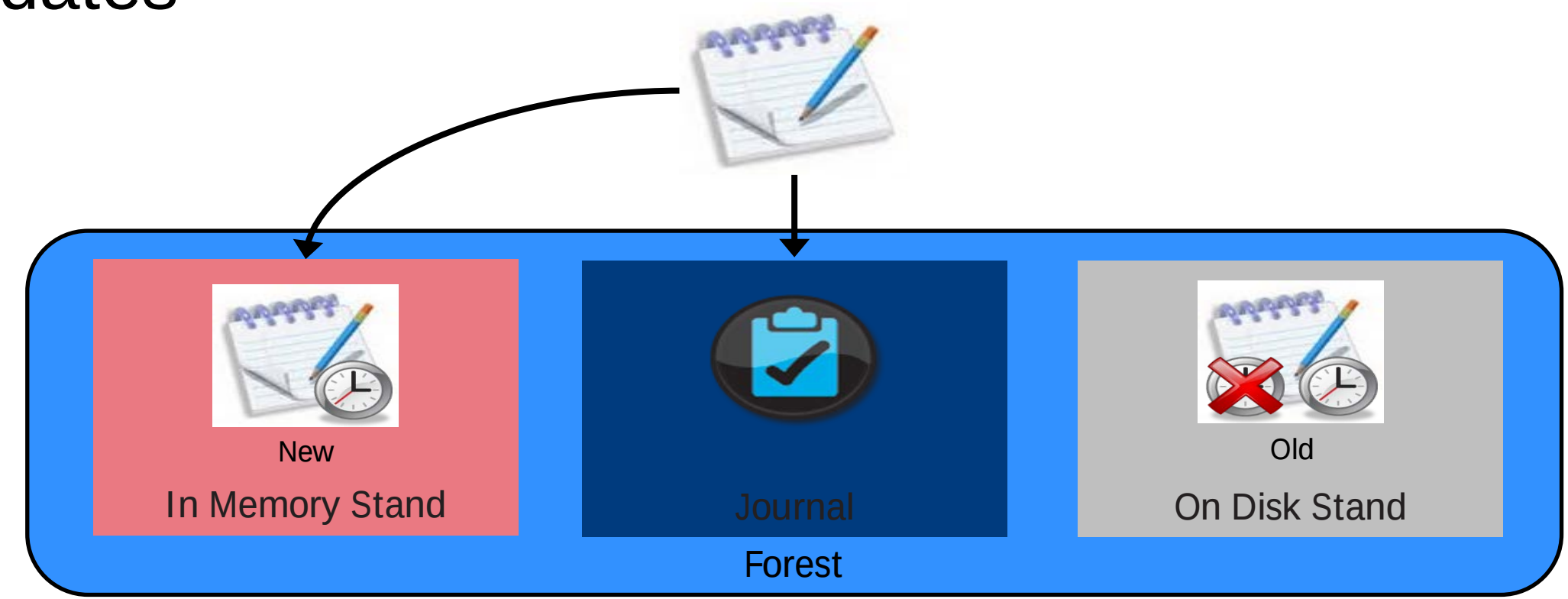
Performing an Insert

- A write where the URI **does not** currently exist in the database.

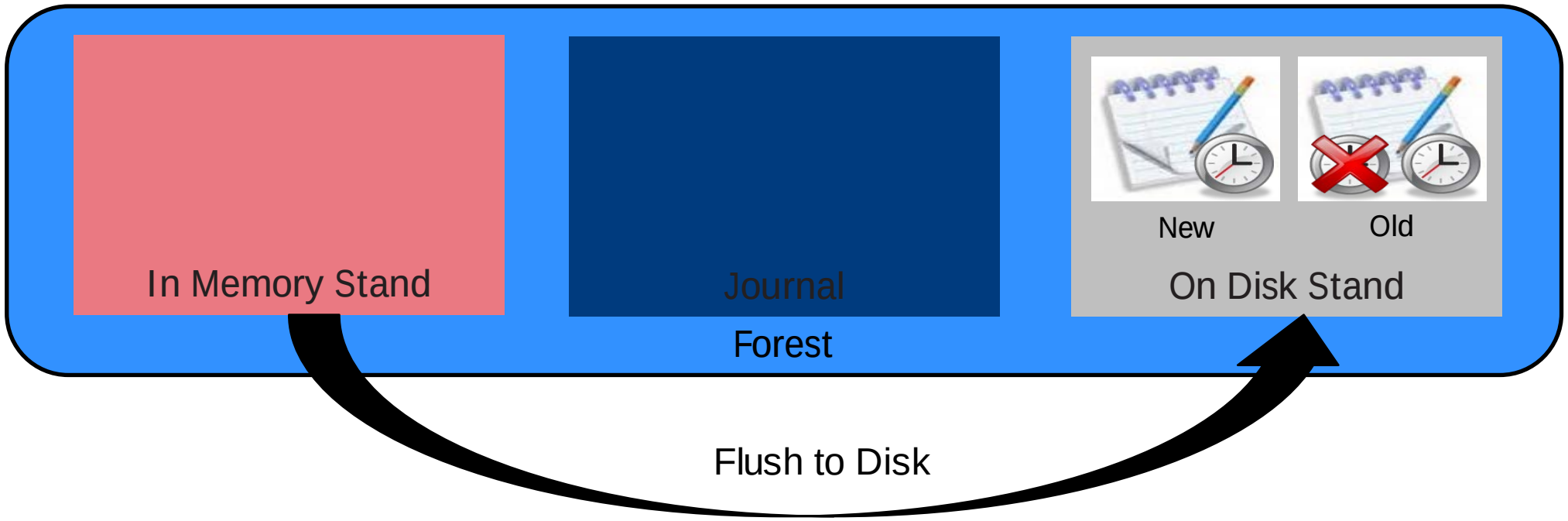
```
dbWrite.documents.write(
  [{
    "uri": "/myURI.json",
    "contentType": "application/json",
    "content":
      {
        "myDoc":
        { "myProperty": "My Property Value" }
      }
  }]).result...
```

MarkLogic

Updates



Updates



MarkLogic

Performing an Update

- Replacing a document.
  - A write where the URI **already exists** in the database.

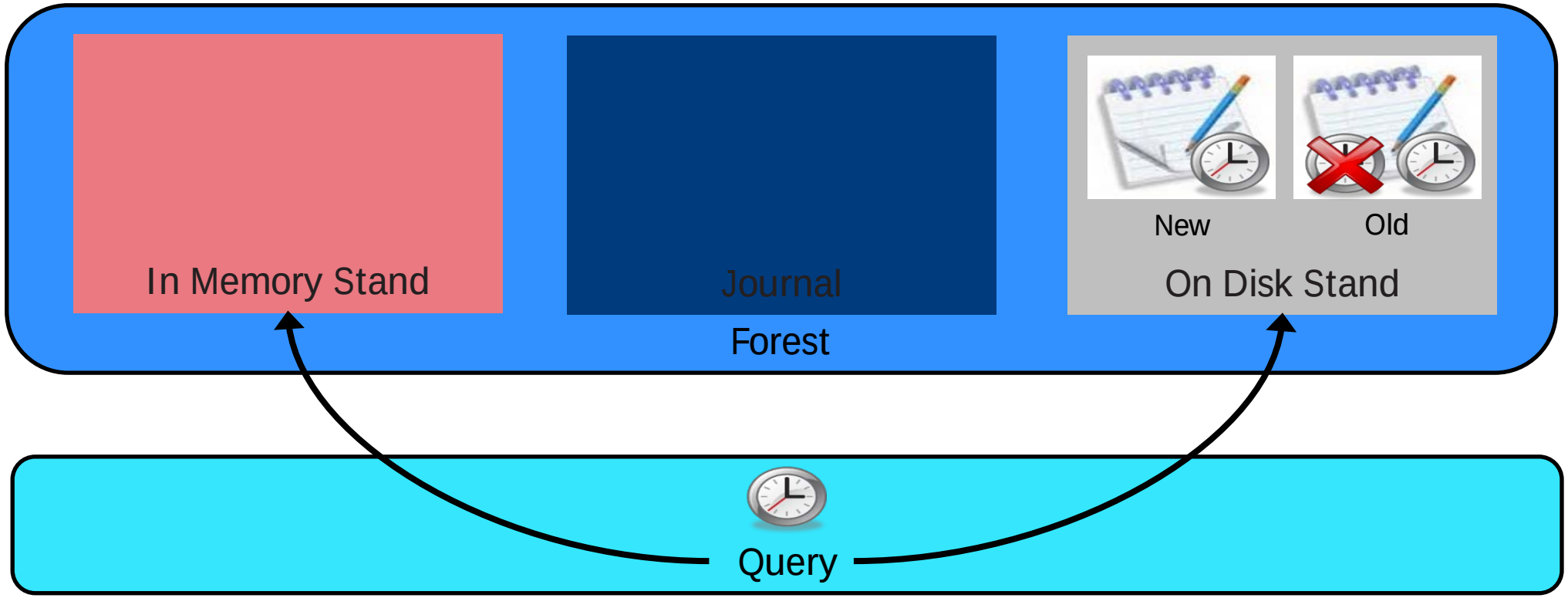
```
dbWrite.documents.write(
  [{
    "uri": "/myURI.json",
    "contentType": "application/json",
    "content":
      {
        "myDoc":
        { "myProperty": "My Property Value" }
      }
  }]).result...
```

- Patching a document (or document metadata).

```
var pb = marklogic.patchBuilder;

dbWrite.documents.patch(
  response[0].uri,
  pb.replaceInsert(
    "/top-song/artist",
    "/top-song/node('artist')",
    "last-child",
    "new property value")
).result();
```

MarkLogic

Query



MarkLogic

Performing a Query

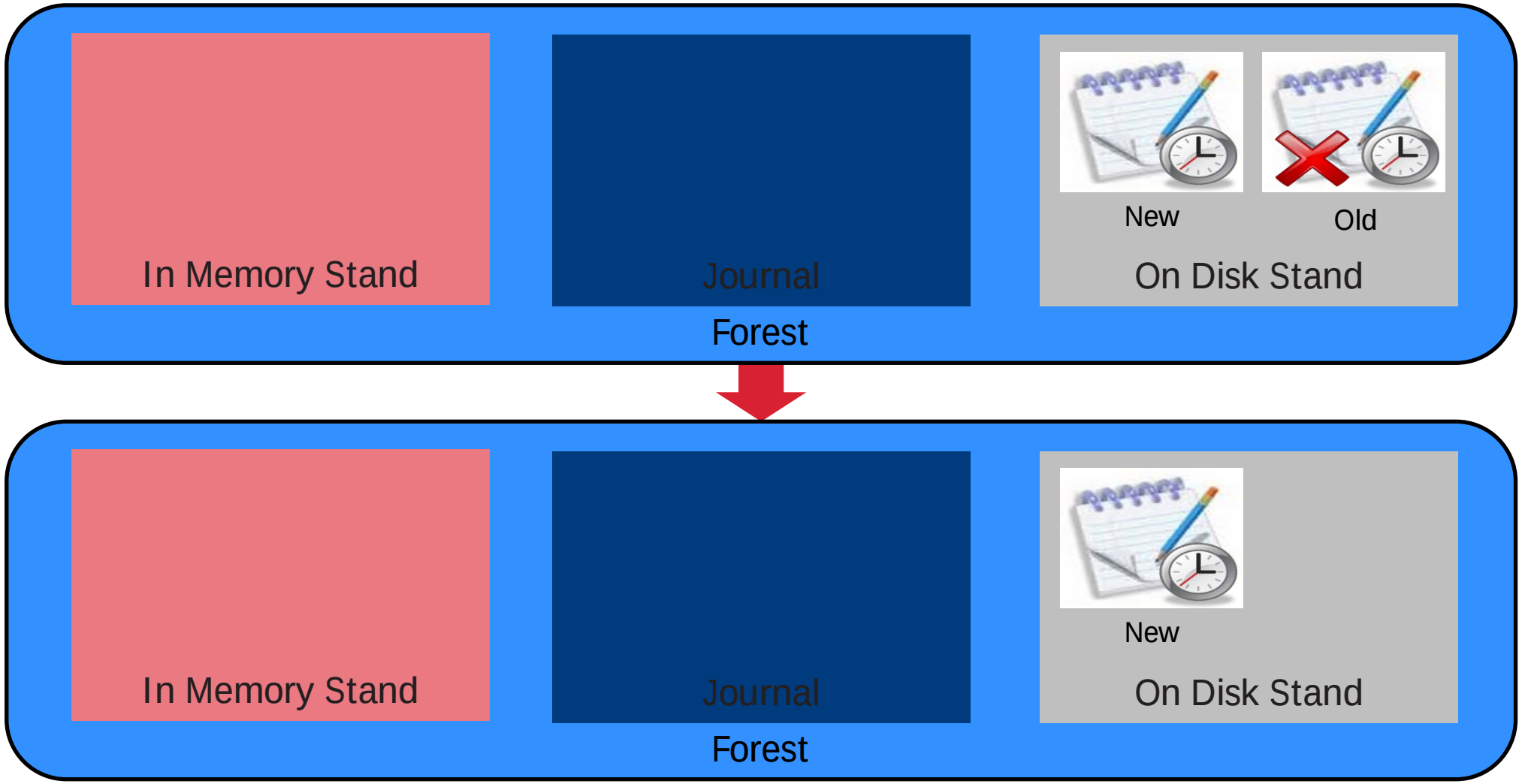
- Reading a document:

```
dbRead.documents.read("/myURI.json").result...
```

- Searching the database:

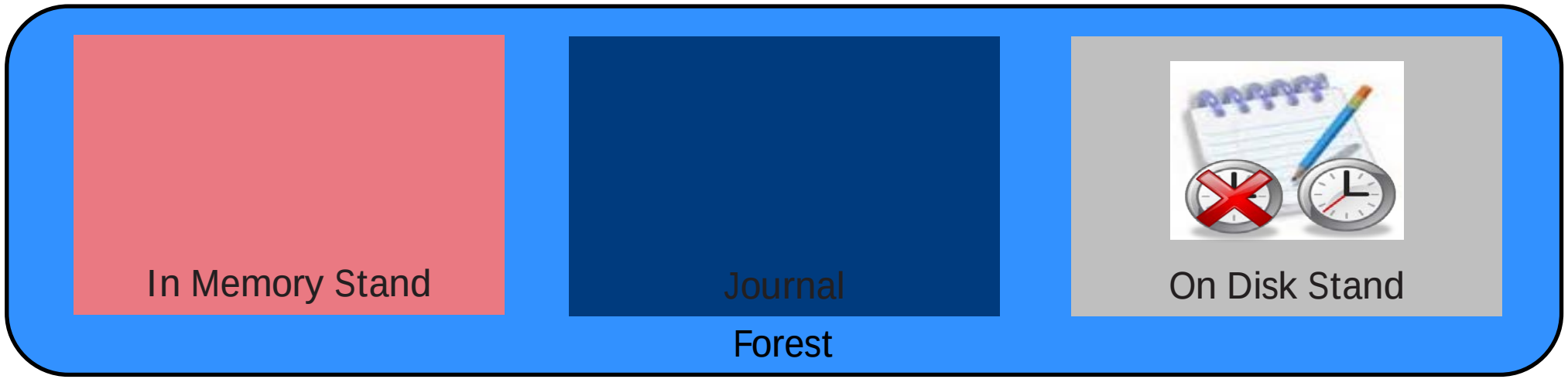
```
var qText = "coldplay";

dbRead.documents.query(
    qb.where(
      qb.parsedFrom(qText)
    ).slice(1, 5)
).result( function(results) {
  console.log(JSON.stringify(results, null, 2));
});
```

Merge



MarkLogic

Delete



Merge

MarkLogic

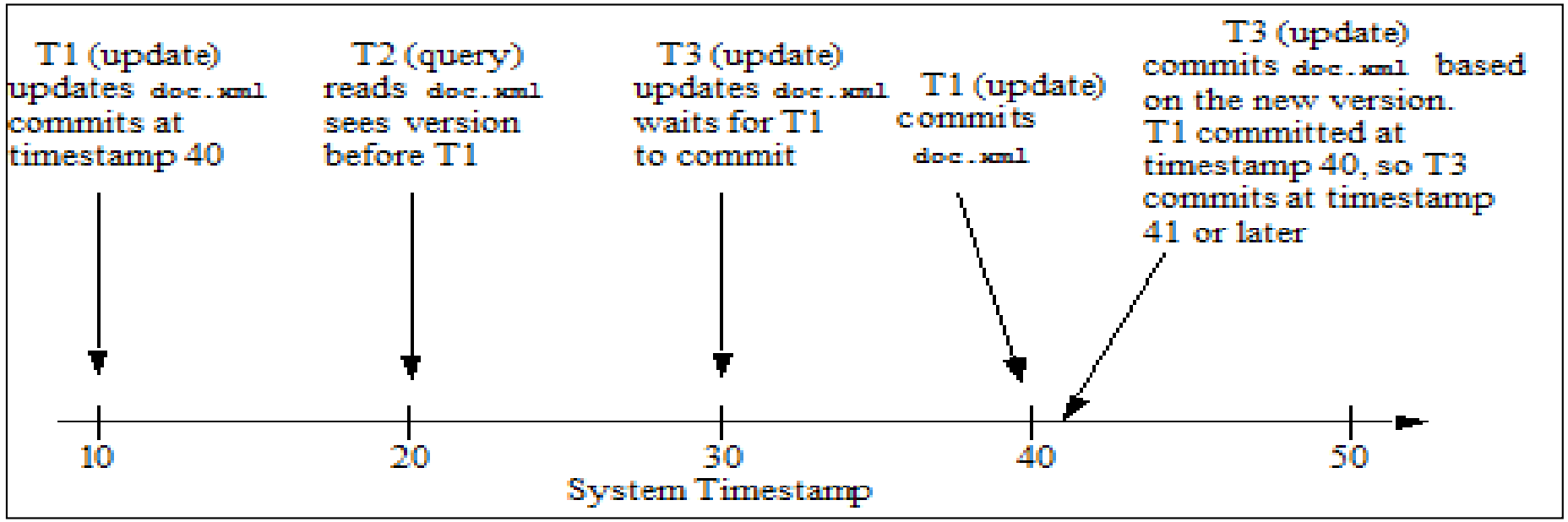
Performing a Delete

- Remove

```
dbWrite.documents.remove("myURI.json").result...
```

MarkLogic

Example Transaction Timeline



MarkLogic

Locking

- To prevent conflicts, whenever the server does something to a document while in a transaction, the database **locks the document** until that transaction either **commits or rolls back**.
- You should commit or roll back your transactions as soon as possible to avoid slowing down your applications.

MarkLogic

Single Statement Transactions

- Every request is a single transaction.
  - Automatically commits on success.
    - Update is immediately visible and available.
  - Automatically rolls back on error or timeout.

```
dbWrite.documents.write(newDocDescriptor)
```

```
dbWrite.documents.patch(
  response[0].uri,
  pb.replaceInsert(
    "/top-song/artist",
    "/top-song/node('artist')",
    "last-child",
    "new property value")
).result();
```

MarkLogic

Multi-Statement Transactions

- **Each request in the transaction sees changes made by previous requests in transaction, but requests outside the transaction do not see those changes.**
- The changes in the transaction either all succeed or all fail.
- Intermix application logic within transaction
- Explicit commit or rollback
- Implicit rollback
  - Timeout, error, session completion
- **Important:** Commit or rollback in your app; don't wait for timeout

MarkLogic

Multi-Statement Transactions

```
function transactionalMove(oldUri, newUri) {
  var transactionId = null;
  dbWrite.transactions.open().result().
  then(function(txid) {
    transactionId = txid;
    return dbWrite.documents.read({uris: oldUri, txid: transactionId.txid}).result();
  }).
  then(function(document) {
    document[0].uri = newUri;
    return dbWrite.documents.write(
      {
        documents: document,
        txid: transactionId.txid
      }).result()
  })
  .then(function(response) {
    return dbWrite.documents.remove({uri: oldUri, txid: transactionId.txid}).result();
  })
  .then(function(response) {
    return dbWrite.transactions.commit(transactionId.txid).result();
  })
  .then(function(response){
    var uriArray = [];
    uriArray.push(oldUri, newUri);
    return resultSummary(uriArray);
  })
  .catch(function(error) {
    dbWrite.transactions.rollback(transactionId.txid);
    console.log("Error: ", error);
  });
}
```

Labs: Unit 11

Exercise 1: Perform a Document Update
Exercise 2: Perform a Patch Update
Exercise 3: Perform a Multi-Statement Transaction

Unit Review Question 1:

In order to delete a document you must have a role with what permission:

1. Insert
2. Update
3. Delete
4. Execute

Unit Review Question 1:

In order to delete a document you must have a role with what permission:

1. Insert
2. **Update – remember, a delete operation is updating the document with a delete timestamp.**
3. Delete
4. Execute

Unit Review Question 2:

When searching a database:

1. No locks are acquired on documents.
2. All documents matching the search will have a read lock.
3. All documents matching the search will have a write lock.
4. All documents matching the search will have both a read and a write lock.

Unit Review Question 2:

When searching a database:

1. **No locks are acquired on documents.**
2. All documents matching the search will have a read lock.
3. All documents matching the search will have a write lock.
4. All documents matching the search will have both a read and a write lock.

Unit Review Question 3:

When a delete transaction commits, the documents involved have been purged from disk:

1. True
2. False

Unit Review Question 4:

When a delete transaction commits, the documents involved have been purged from disk:

1. True
2. **False**

Unit Review Question 4:

The forest level journal ensures a transactions:

1. Atomicity
2. Consistency
3. Isolation
4. Durability

Unit Review Question 4:

The forest level journal ensures a transactions:

1. Atomicity
2. Consistency
3. Isolation
4. **Durability**