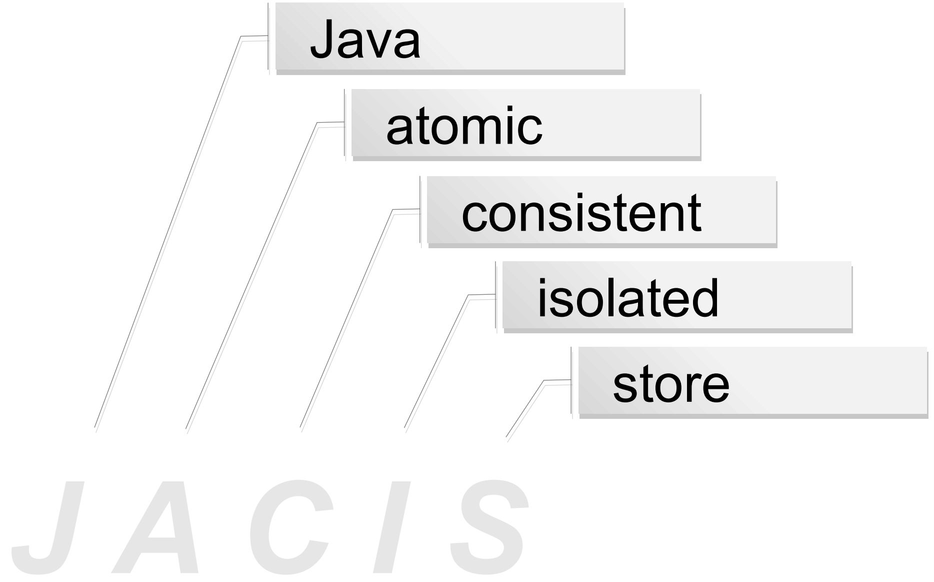
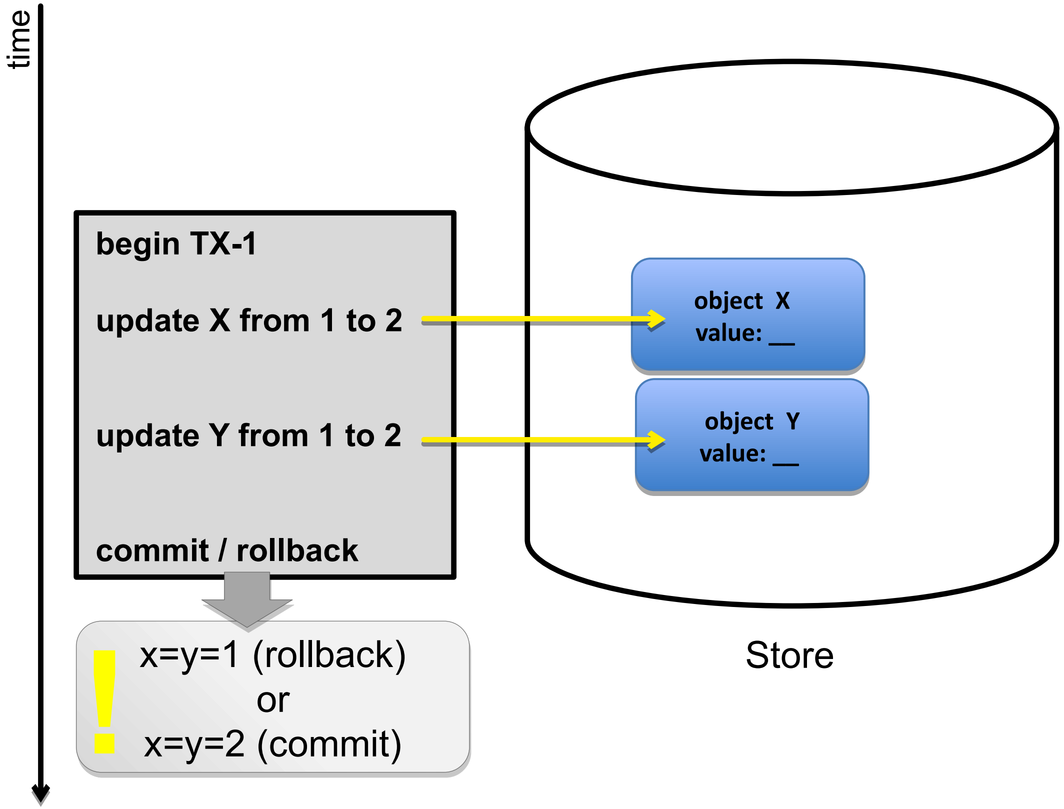
JACIS



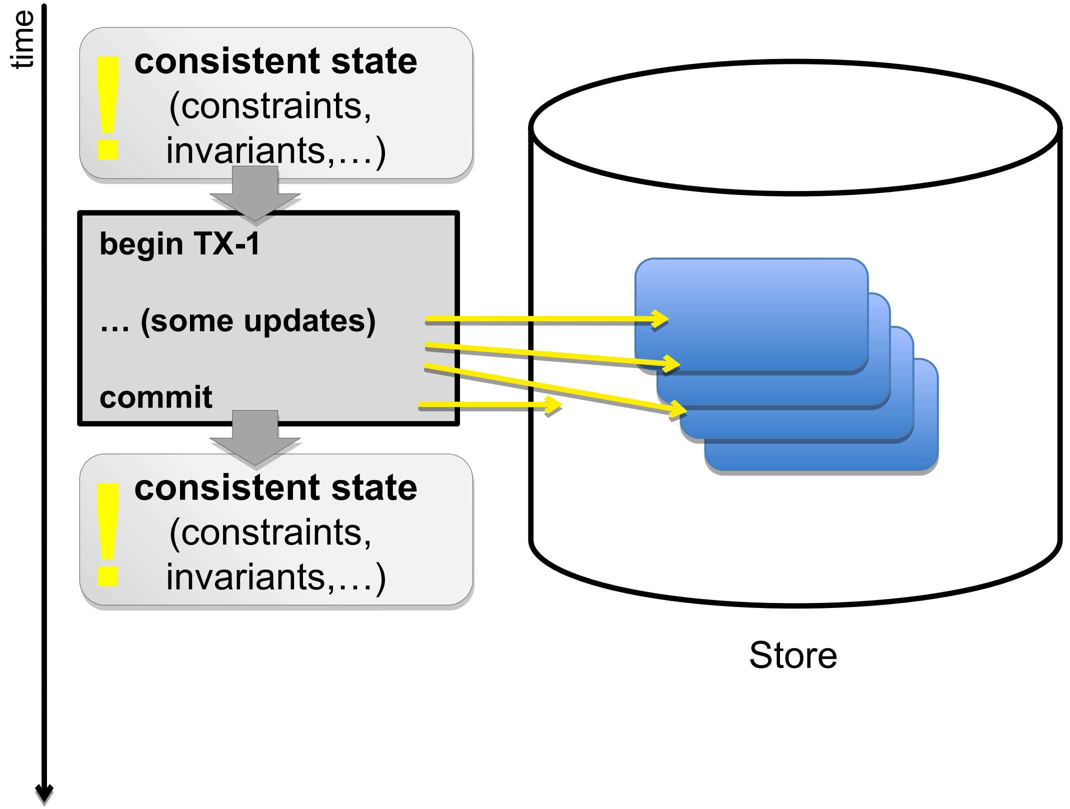
a transactional in-memory store in Java

https://github.com/JanWiemer/jacis

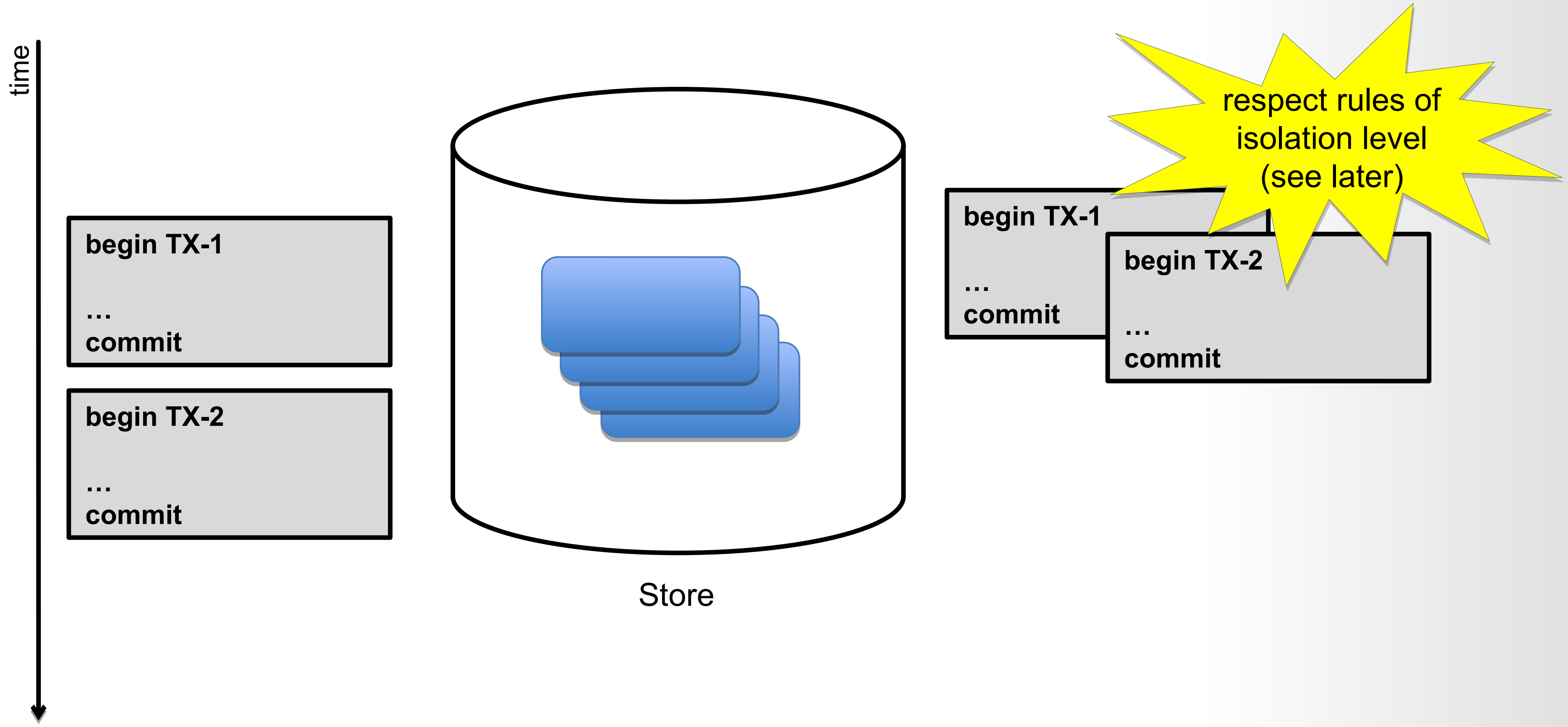
ACID - ATOMICITY



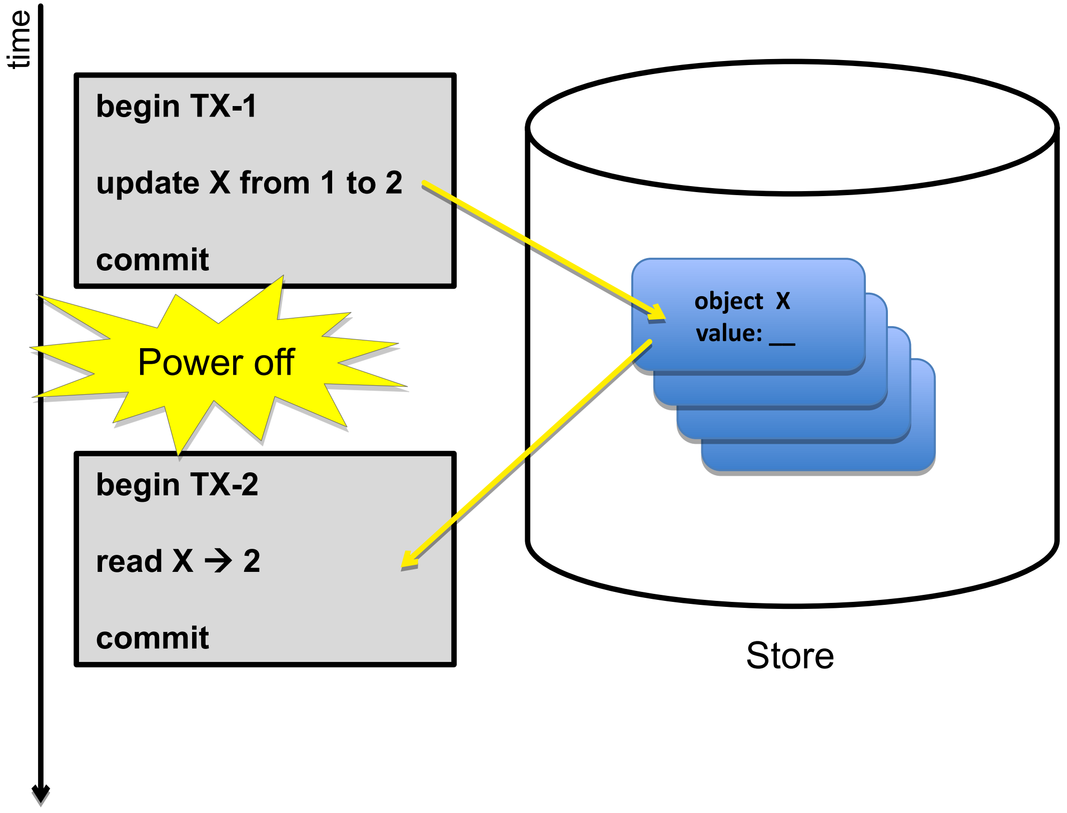
ACID - CONSISTENCY



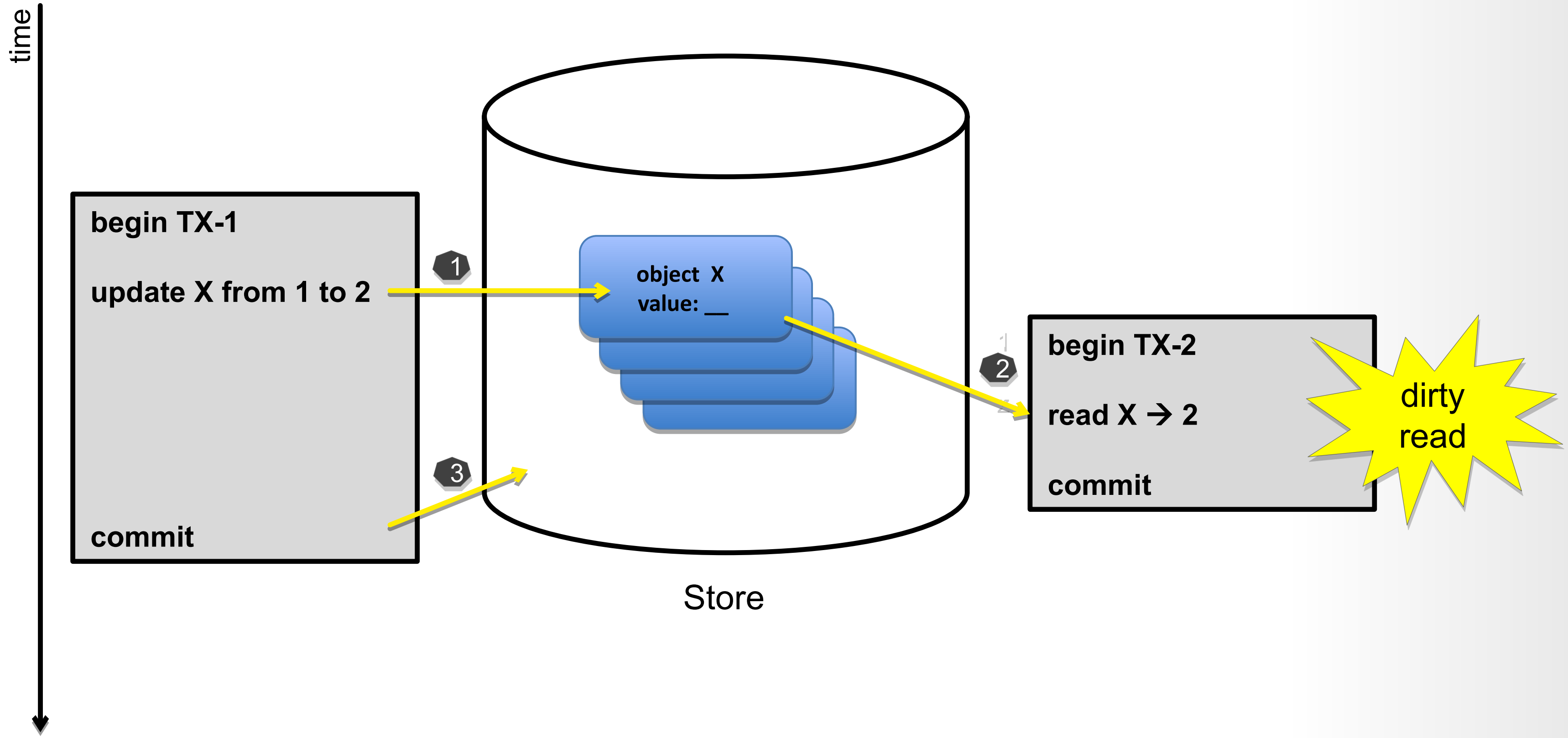
ACID - ISOLATION



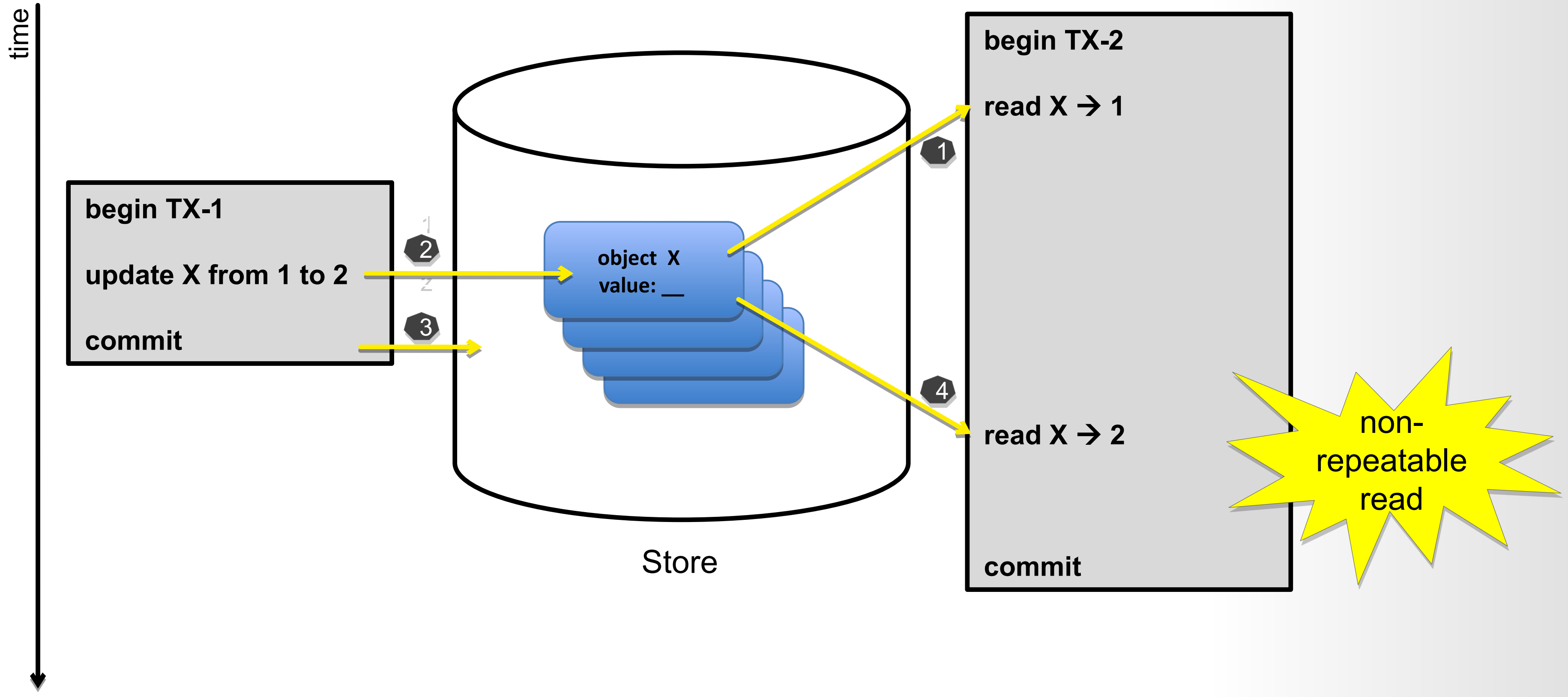
ACID - DURABILITY



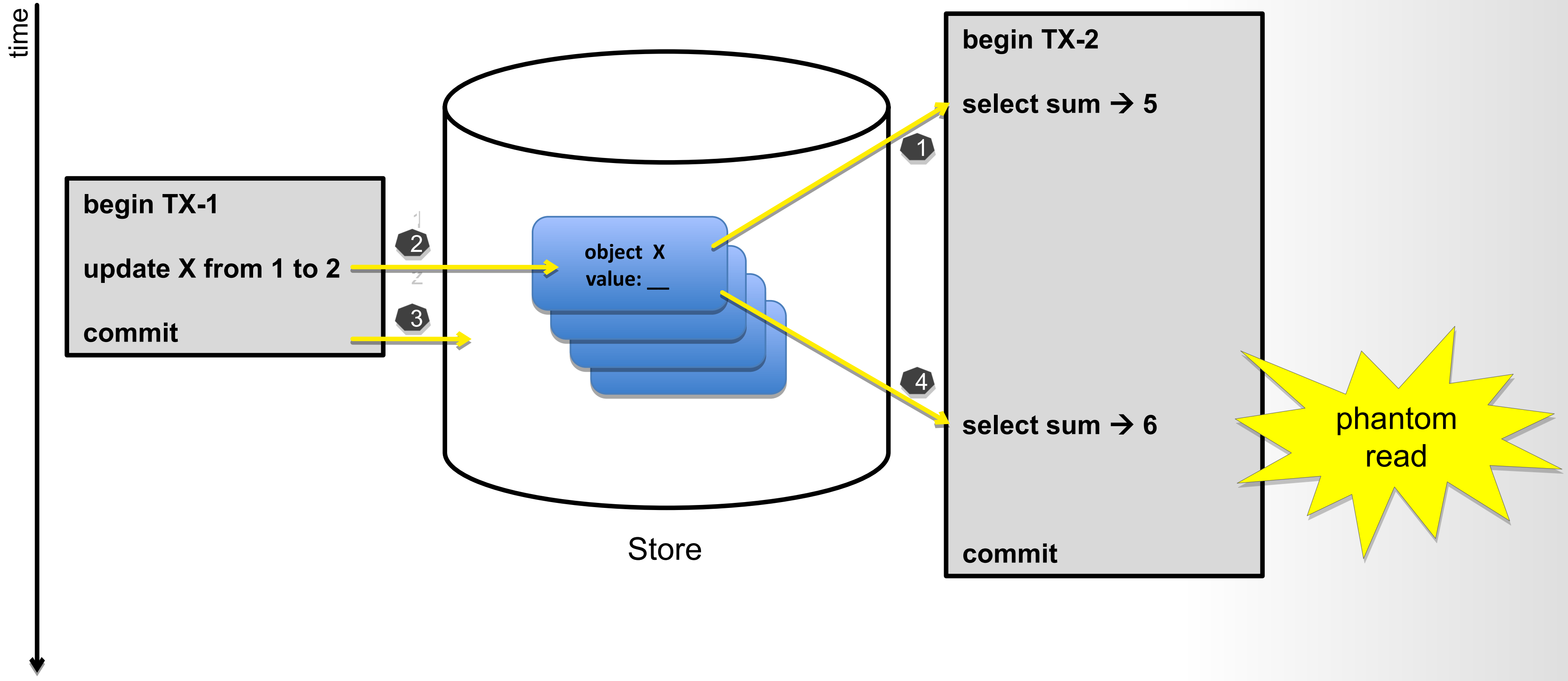
ISOLATION - DIRTY READ



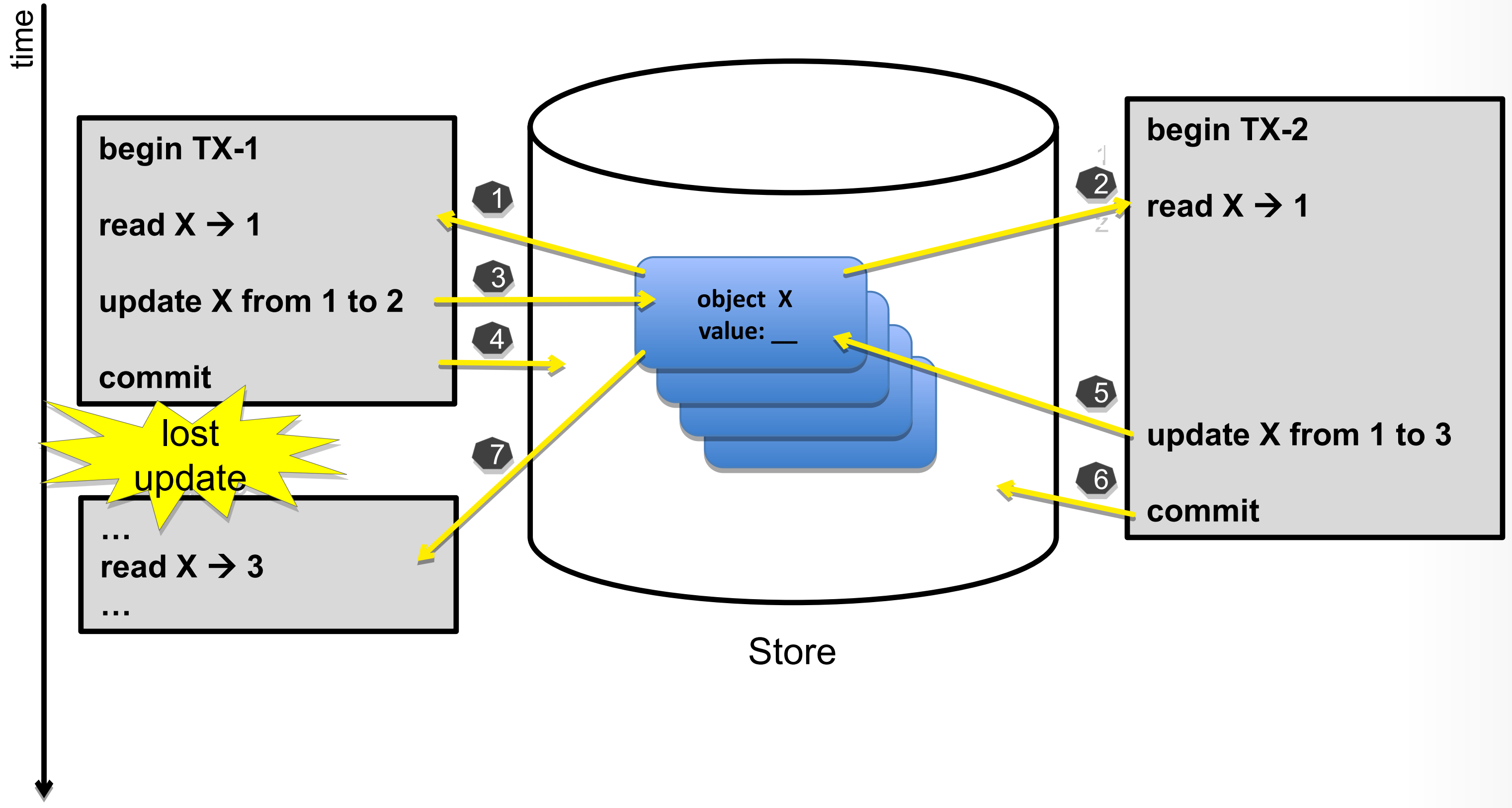
ISOLATION - NON-REPEATABLE READ



ISOLATION - PHANTOM READ



ISOLATION - LOST UPDATE



ISOLATION LEVELS

Isolation Level	Lost Updates	Dirty Read	Non-repeatable Read	Phantom Read
READ UNCOMMITTED		permitted	permitted	permitted
READ COMMITTED			permitted	permitted
REPEATABLE READ				permitted
SNAPSHOT ISOLATION				*permitted*
SERIALIZABLE				

ANSI/ISO SQL-Standard (SQL-92)

CONCURRENCY CONTROL MECHANISMS

Categories:

• **Pessimistic**	(prevent rule violations by blocking operations with **locks**)
• **optimistic**	(detect rule violations later and retry execution of the TX)
• semi optimistic	(sometimes pessimistic, sometimes optimistic)

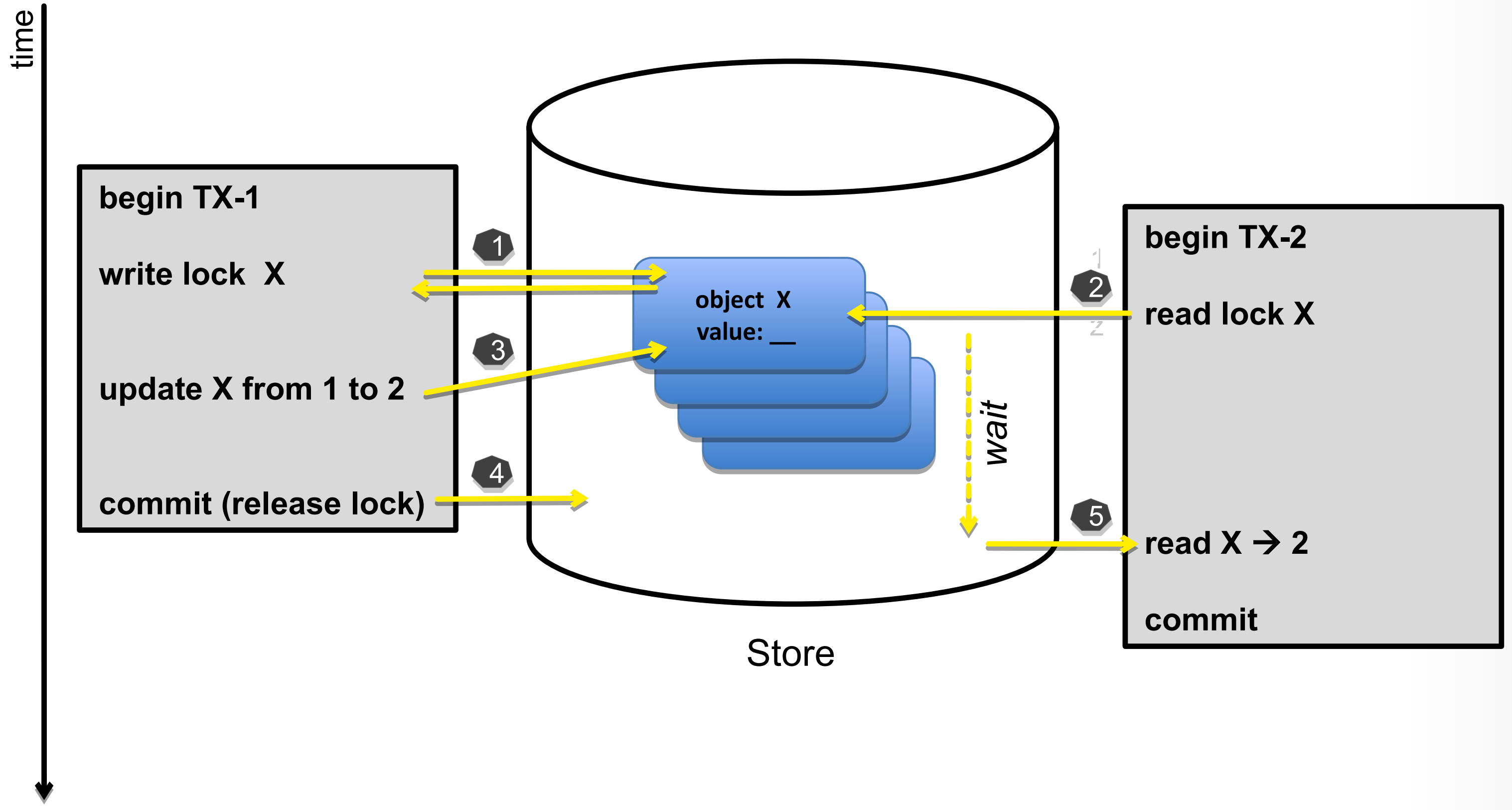
Methods:

- locking
- serialization graph checking
- timestamp ordering
- commitment ordering

Methods:

- multi-version concurrency control (MVCC)
- index concurrency control (index locking)
- private workspace model (deferred update)

CONCEPT - PESSIMISTIC LOCKING



CONCEPT - PESSIMISTIC LOCKING

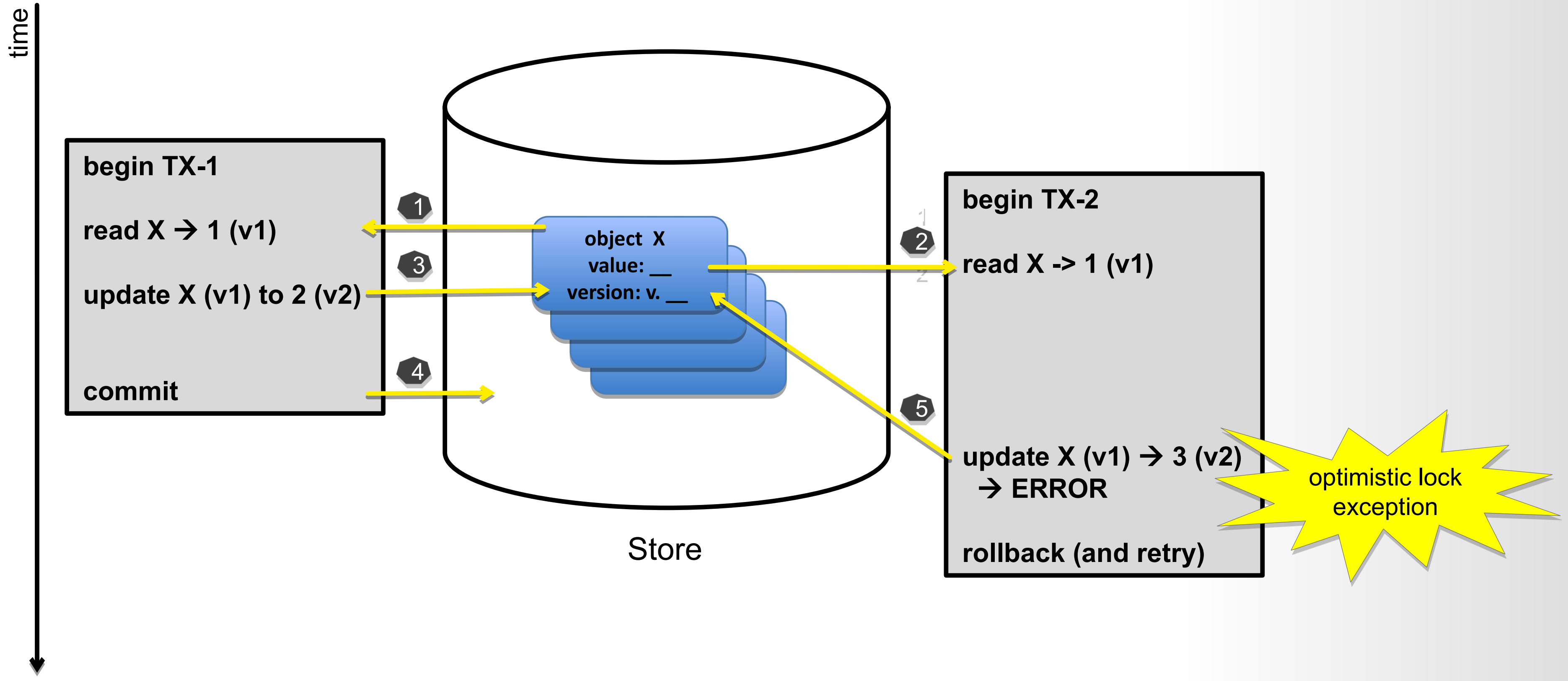
Lock Type	read-lock (shared lock)	write-lock (exclusive lock)
read-lock (shared lock)	allowed	incompatible
write-lock (exclusive lock)	incompatible	incompatible

CONCEPT - PESSIMISTIC LOCKING

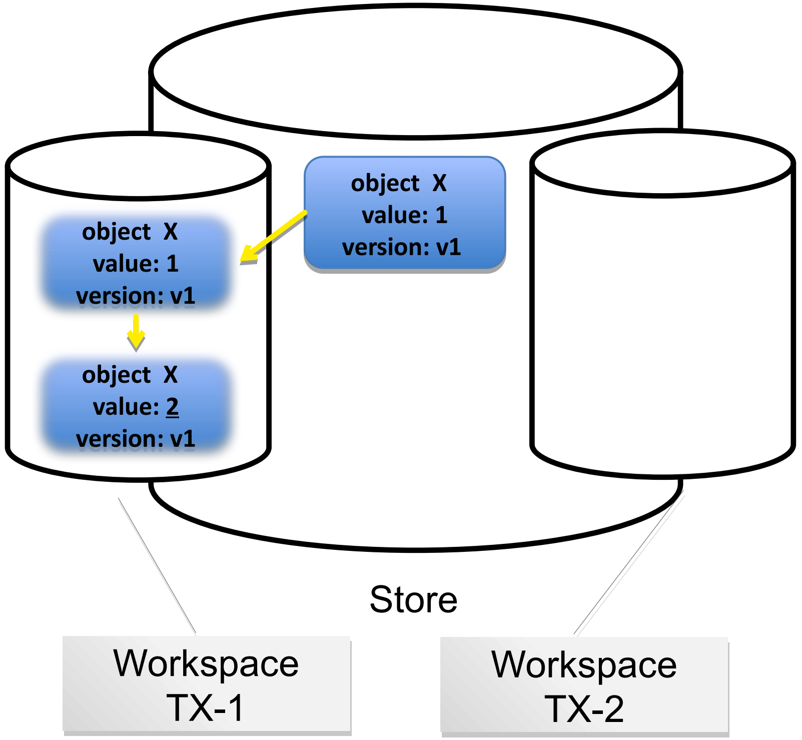
Lock Type	read-lock (shared lock)	write-lock (exclusive lock)
read-lock (shared lock)	allowed	incompatible
write-lock (exclusive lock)	incompatible	incompatible

- **Two-Phase Locking (2PL)**
  - **Expanding phase** (/ Growing phase): locks are acquired
  - **Shrinking phase**: locks are released
- Conservative 2PL *prevents* deadlocks
- **Strong strict two-phase locking** or **Rigorousness**, or **Rigorous scheduling**, or **Rigorous two-phase locking**

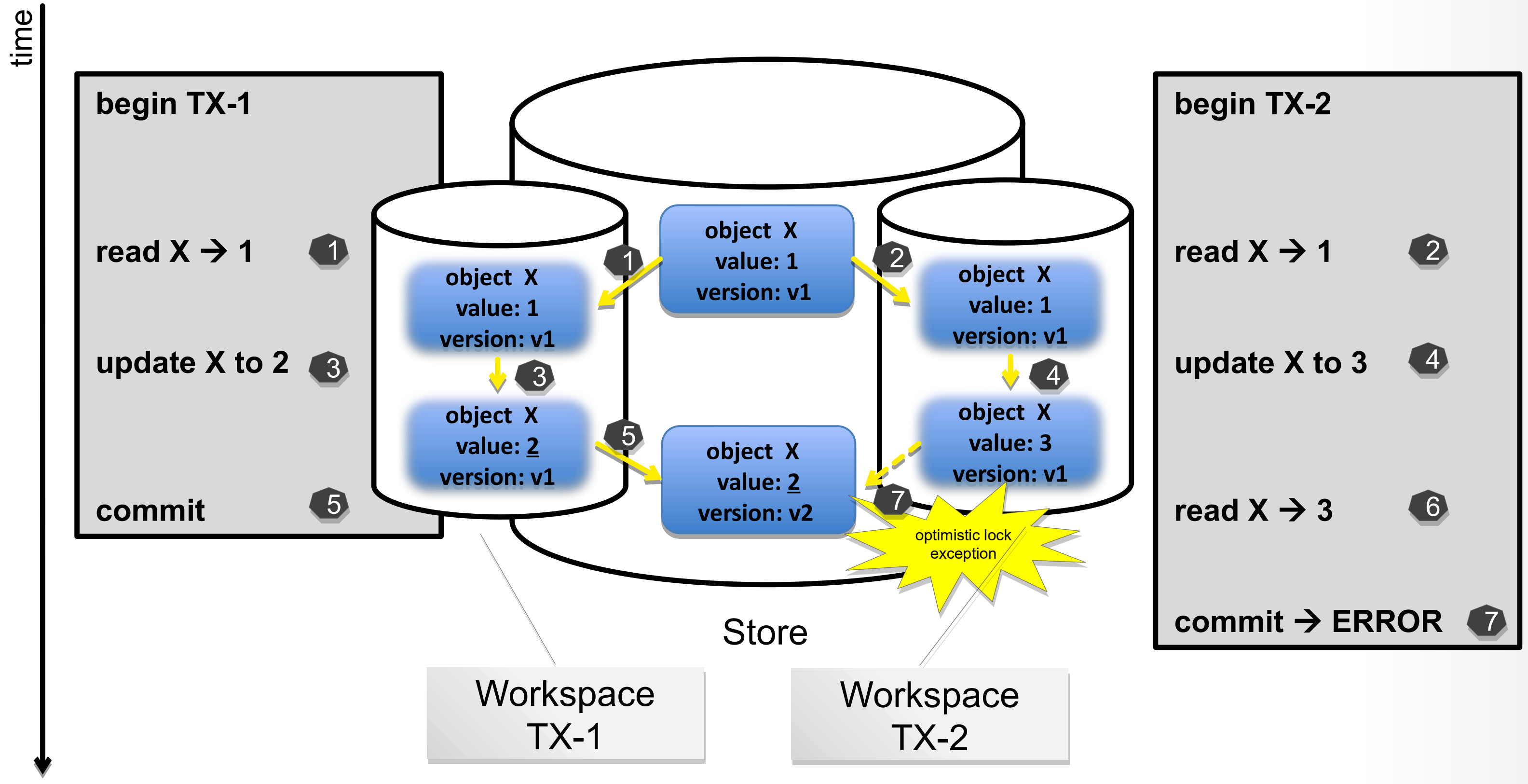
CONCEPT - OPTIMISTIC LOCKING



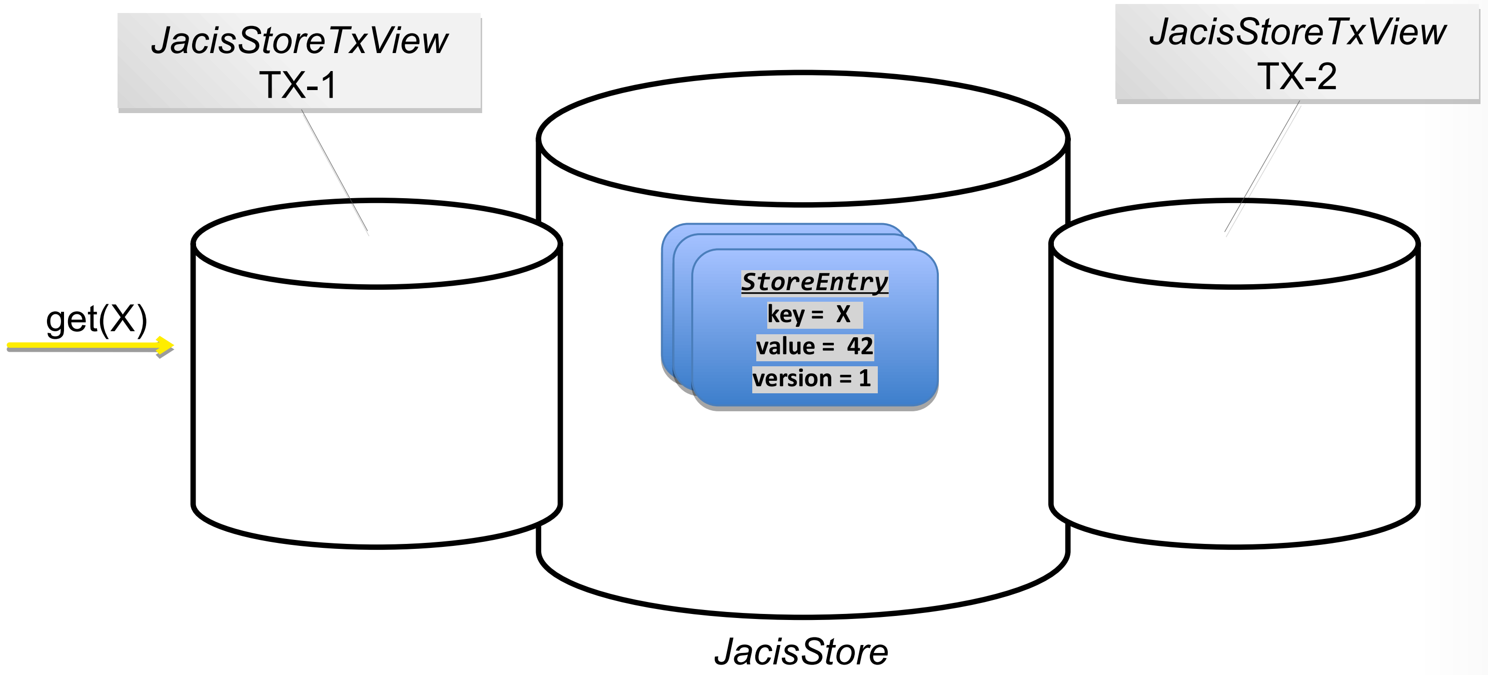
CONCEPT – PRIVATE WORKSPACE MODEL



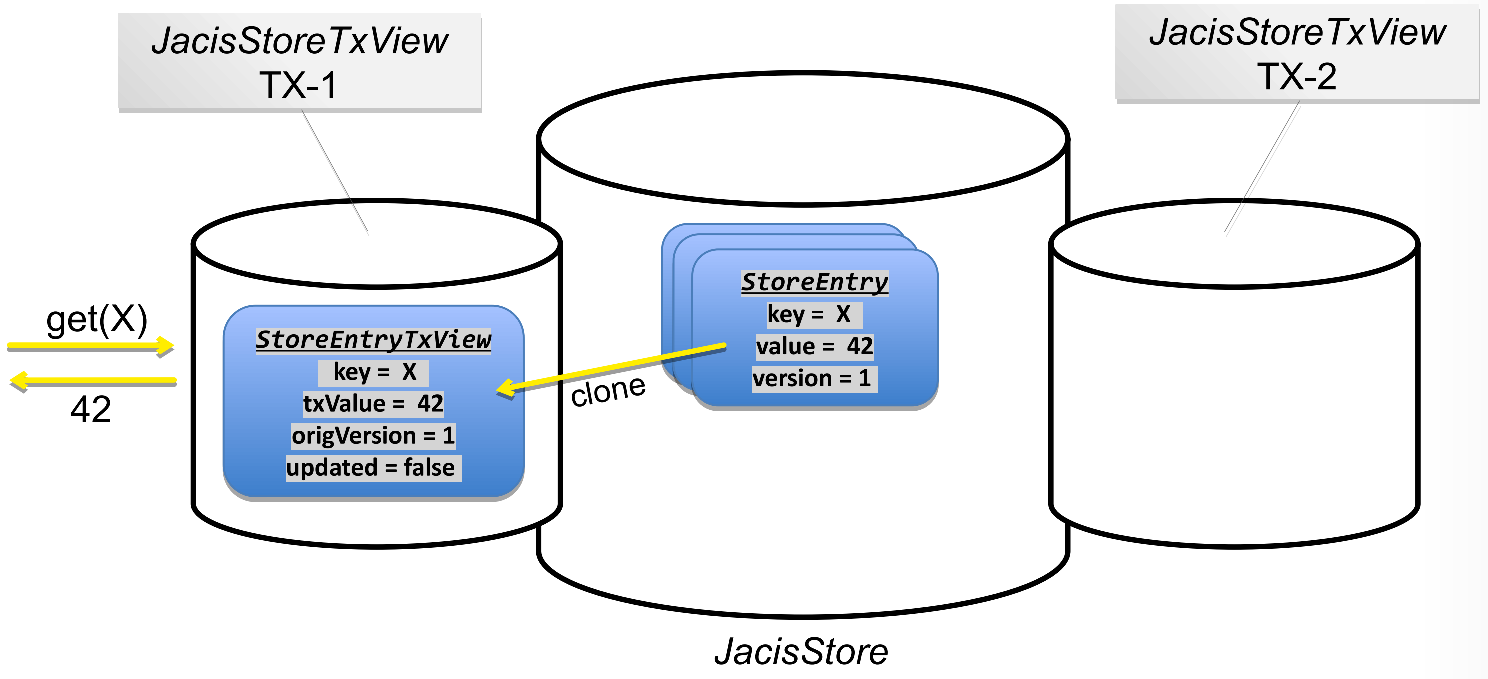
CONCEPT – PRIVATE WORKSPACE MODEL



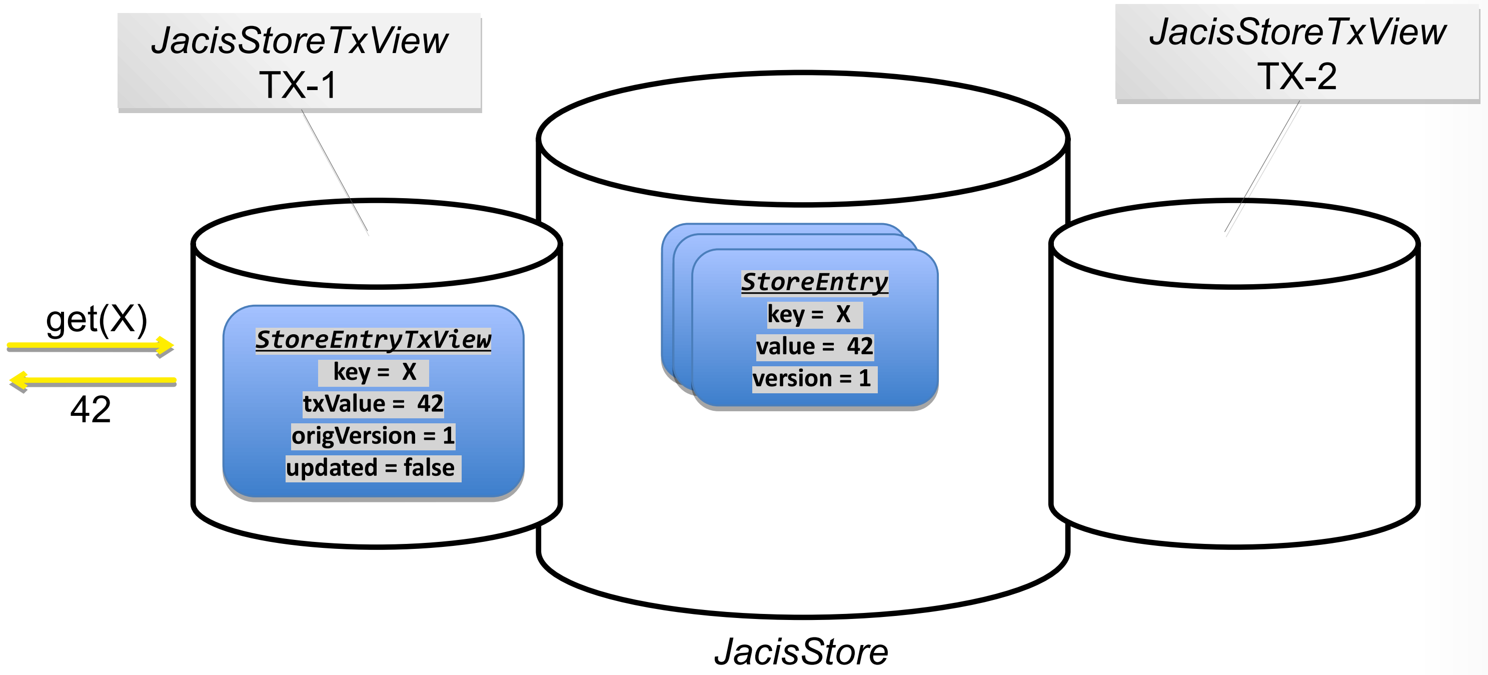
JACIS


JacisStoreTxView
TX-1
JacisStoreTxView
TX-2
get(X)
StoreEntry
key = X
value = 42
version = 1
JacisStore

JACIS

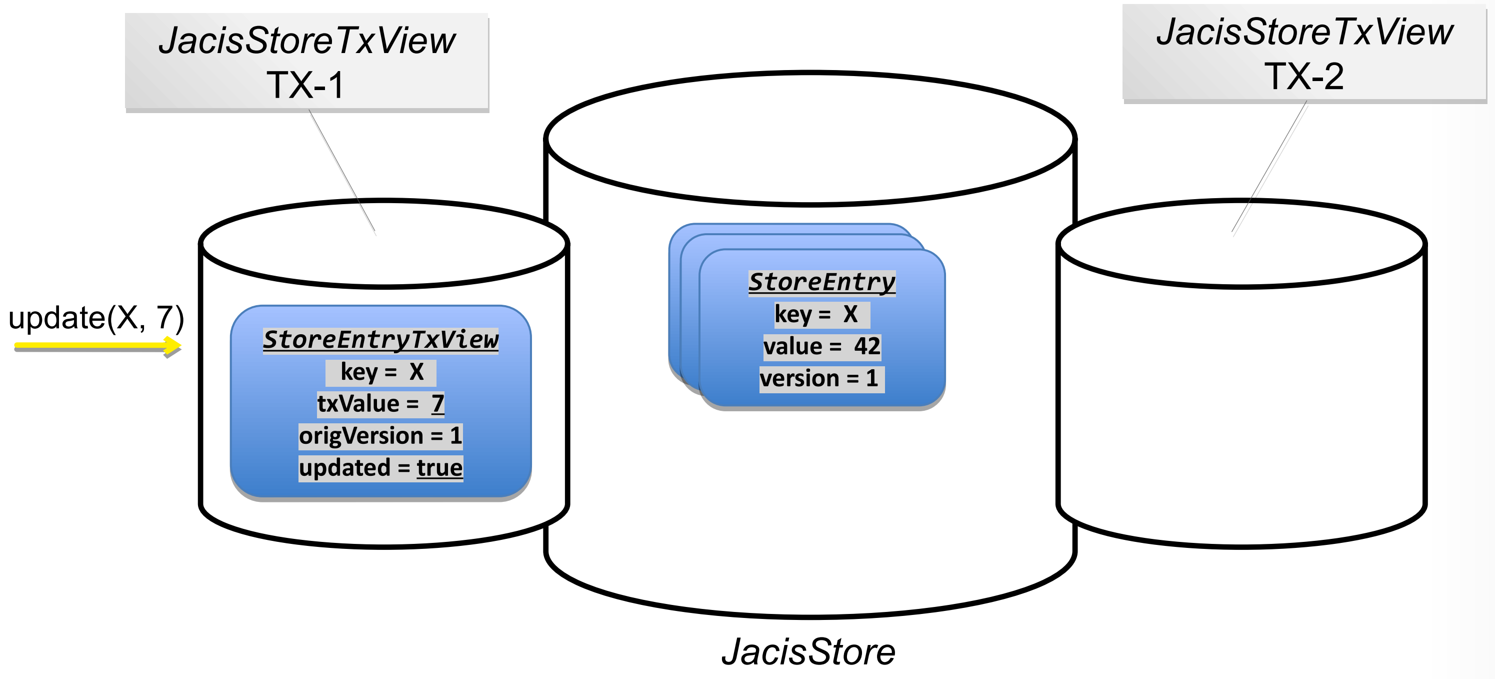

JacisStoreTxView
TX-1
JacisStoreTxView
TX-2
get(X)
42
StoreEntryTxView
key = X
txValue = 42
origVersion = 1
updated = false
clone
StoreEntry
key = X
value = 42
version = 1
JacisStore

JACIS



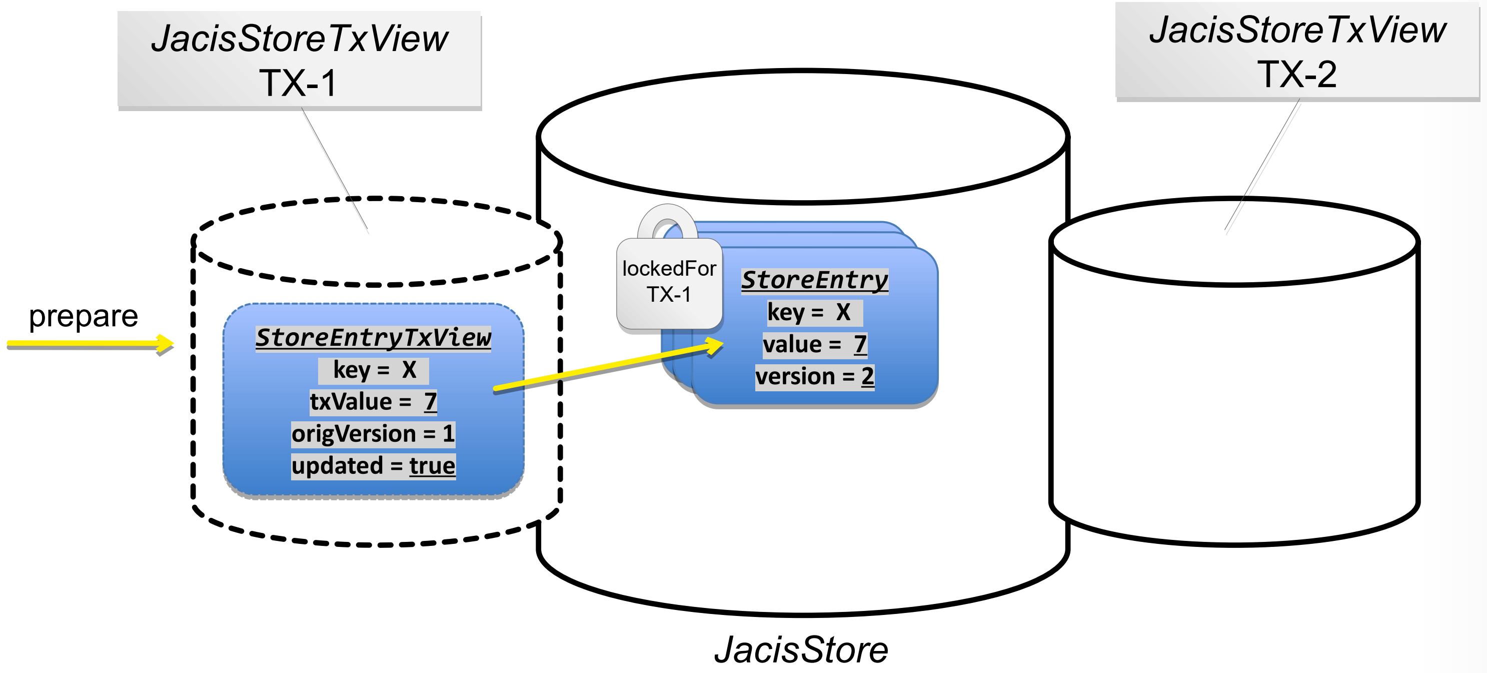
JacisStoreTxView
TX-1
get(X)
42
StoreEntryTxView
key = X
txValue = 42
origVersion = 1
updated = false
StoreEntry
key = X
value = 42
version = 1
JacisStoreTxView
TX-2
JacisStore

JACIS

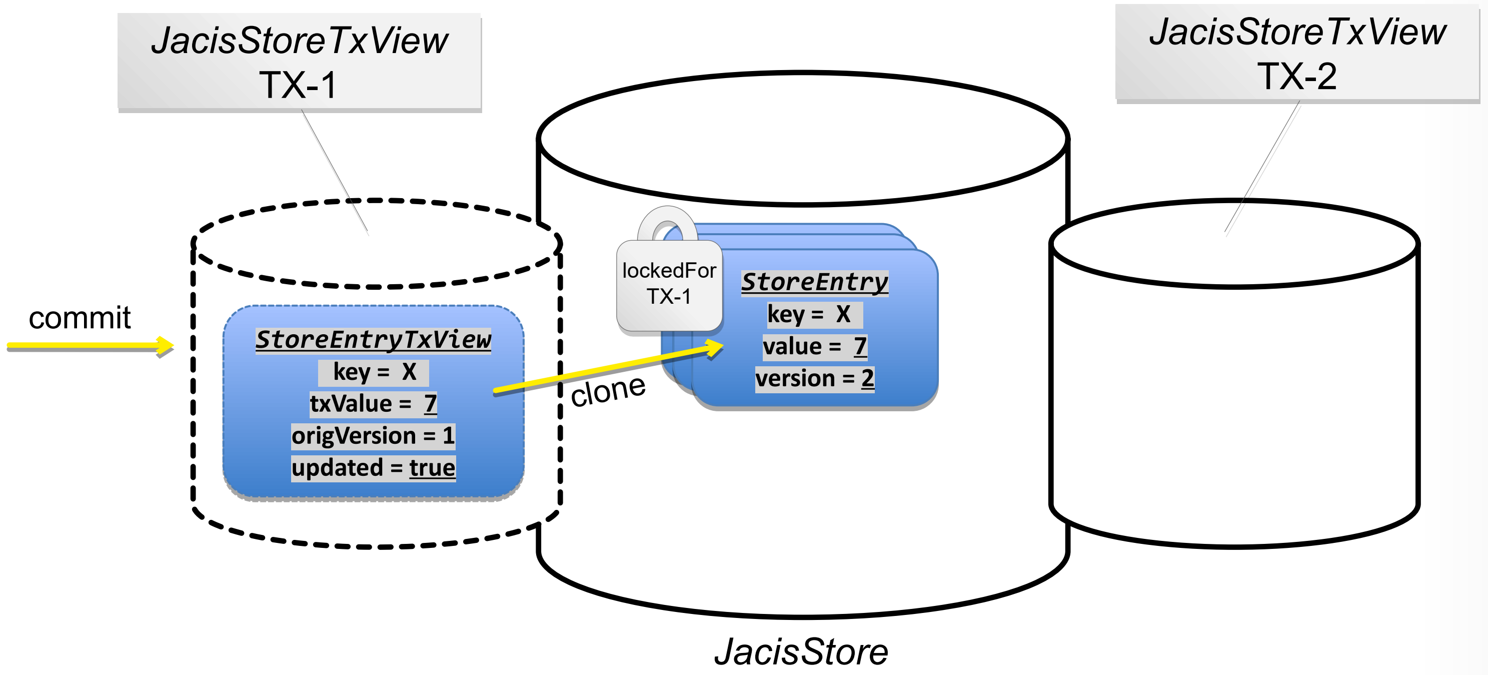


JacisStoreTxView
TX-1
JacisStoreTxView
TX-2
update(X, 7)
StoreEntryTxView
key = X
txValue = 7
origVersion = 1
updated = true
StoreEntry
key = X
value = 42
version = 1
JacisStore

JACIS

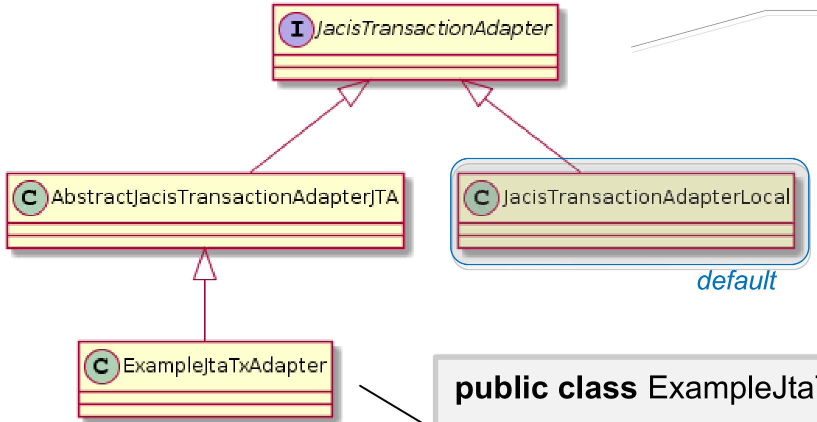

JacisStoreTxView
TX-1
JacisStoreTxView
TX-2
prepare
StoreEntryTxView
key = X
txValue = 7
origVersion = 1
updated = true
lockedFor
TX-1
StoreEntry
key = X
value = 7
version = 2
JacisStore

JACIS



JacisStoreTxView
TX-1
JacisStoreTxView
TX-2
commit
StoreEntryTxView
key = X
txValue = 7
origVersion = 1
updated = true
clone
lockedFor
TX-1
StoreEntry
key = X
value = 7
version = 2
JacisStore

JACIS – TRANSACTION ADAPTER

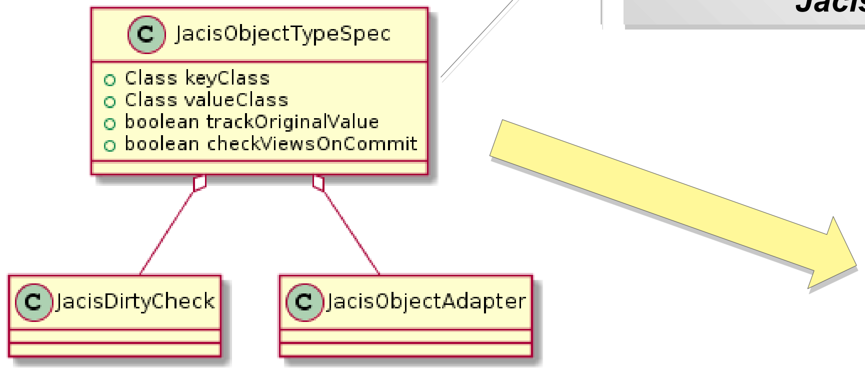

JacisTransactionAdapter
AbstractJacisTransactionAdapterJTA
JacisTransactionAdapterLocal
default
ExampleJtaTxAdapter

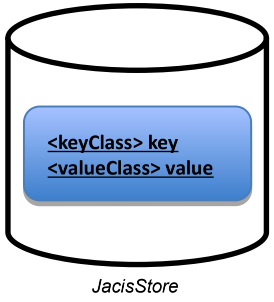
pass transaction adapter to the constructor of the *JacisContainer*

```
public class ExampleJtaTxAdapter extends AbstractJacisTransactionAdapterJTA {

  @Override
  protected javax.transaction.TransactionManager getTransactionManager() {
    return JtaHelper.getTransactionManager(); // provide access to the tx manager
  }

}
```

JACIS – OBJECT TYPE SPECIFICATION

specify as parameter when creating a *JacisStore:*
JacisContainer.createStore

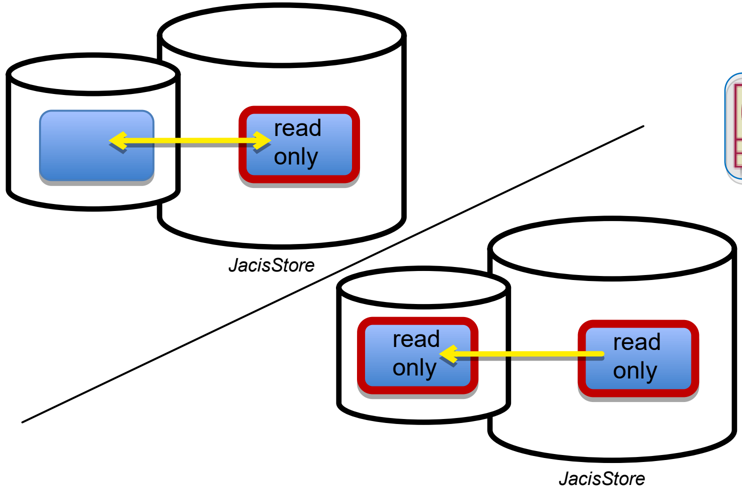




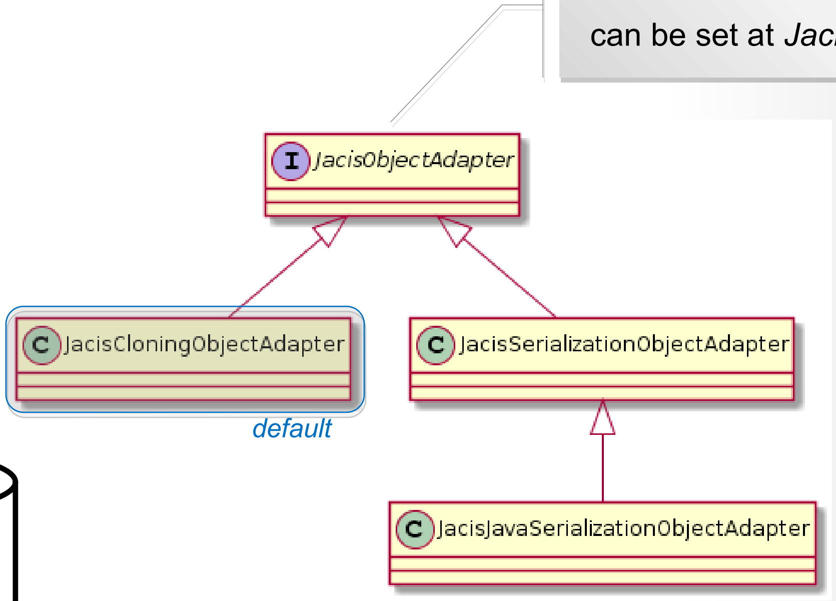
JACIS – OBJECT ADAPTER

Tell JACIS how to clone objects between store and TX view

- cloneCommitted2WritableTxView
- cloneTxView2Committed
- cloneCommitted2ReadOnlyTxView
- cloneTxView2ReadOnlyTxView

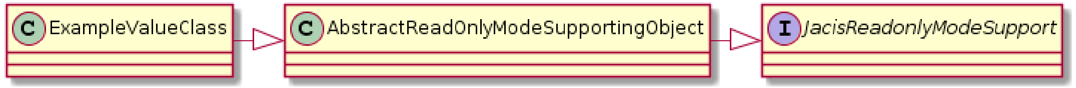


can be set at *JacisObjectTypeSpec*



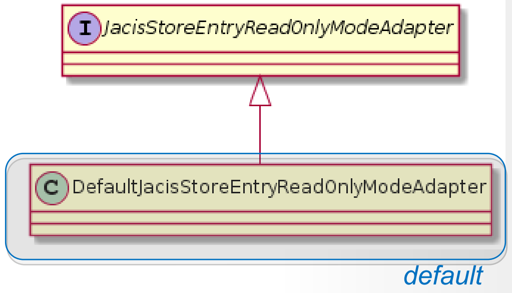
I JacisObjectAdapter
C JacisCloningObjectAdapter
default
C JacisSerializationObjectAdapter
C JacisJavaSerializationObjectAdapter

JACIS – READ ONLY MODE

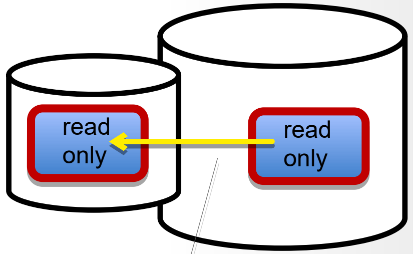


```
class ExampleValueClass extends AbstractReadOnlyModeSupportingObject  {

  private final String name;

  public String getName() {
    return name;
  }

  public ExampleValueClass setName(String name) {
    checkWritable();
    this.name = name;
    return this;
  }
}
```

passed to the constructor of the *JacisCloningObjectAdapter*



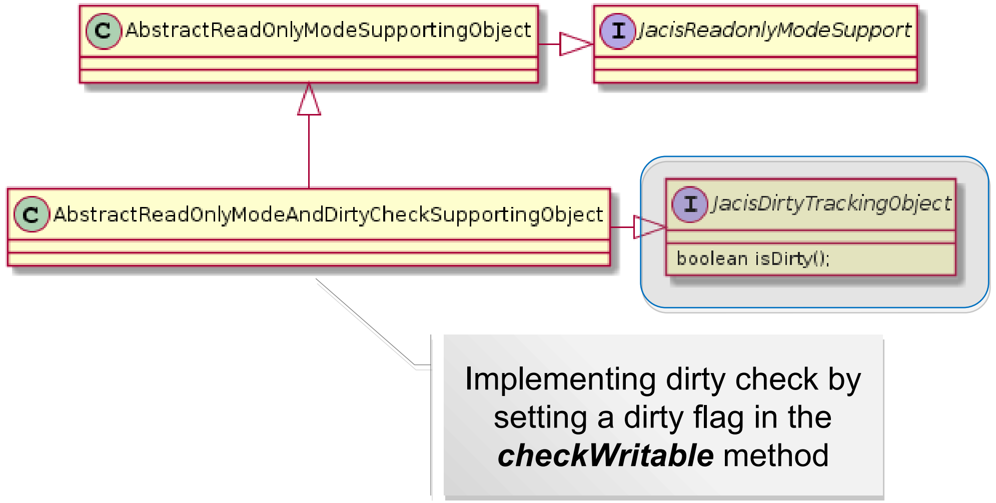
JacisStoreEntryReadOnlyModeAdapter
DefaultJacisStoreEntryReadOnlyModeAdapter
default



avoid cloning

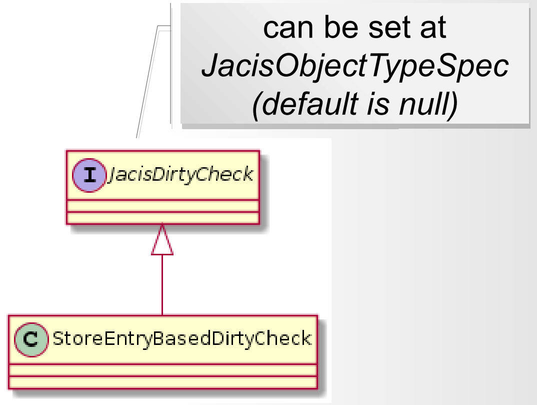
JACIS – DIRTY CHECK

! By **default** there is **no dirty check**!
All changed have to be explicitly notified by:
JacisStore.update



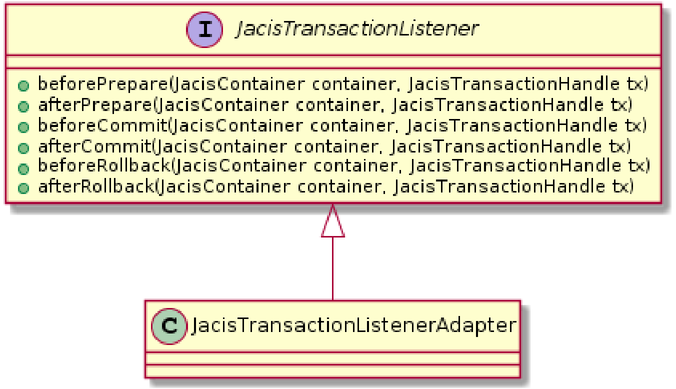
C AbstractReadOnlyModeSupportingObject
I JacisReadonlyModeSupport
C AbstractReadOnlyModeAndDirtyCheckSupportingObject
I JacisDirtyTrackingObject
boolean isDirty();

Implementing dirty check by setting a dirty flag in the ***checkWritable*** method



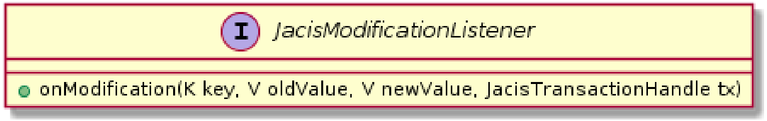
can be set at *JacisObjectTypeSpec (default is null)*

JACIS – TRANSACTION LISTENER


I JacisTransactionListener
beforePrepare(JacisContainer container, JacisTransactionHandle tx)
afterPrepare(JacisContainer container, JacisTransactionHandle tx)
beforeCommit(JacisContainer container, JacisTransactionHandle tx)
afterCommit(JacisContainer container, JacisTransactionHandle tx)
beforeRollback(JacisContainer container, JacisTransactionHandle tx)
afterRollback(JacisContainer container, JacisTransactionHandle tx)
C JacisTransactionListenerAdapter

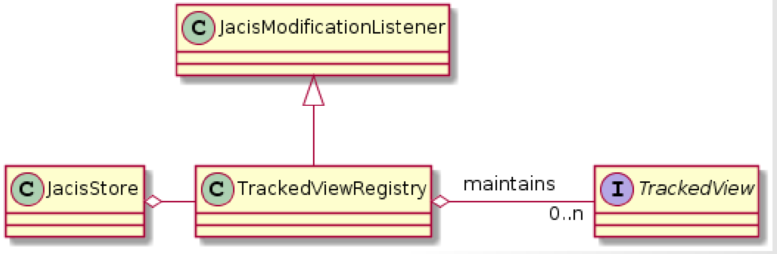
- provides possibility to execute code on transaction demarcation events
- register by calling ***JacisContainer. registerTransactionListener***

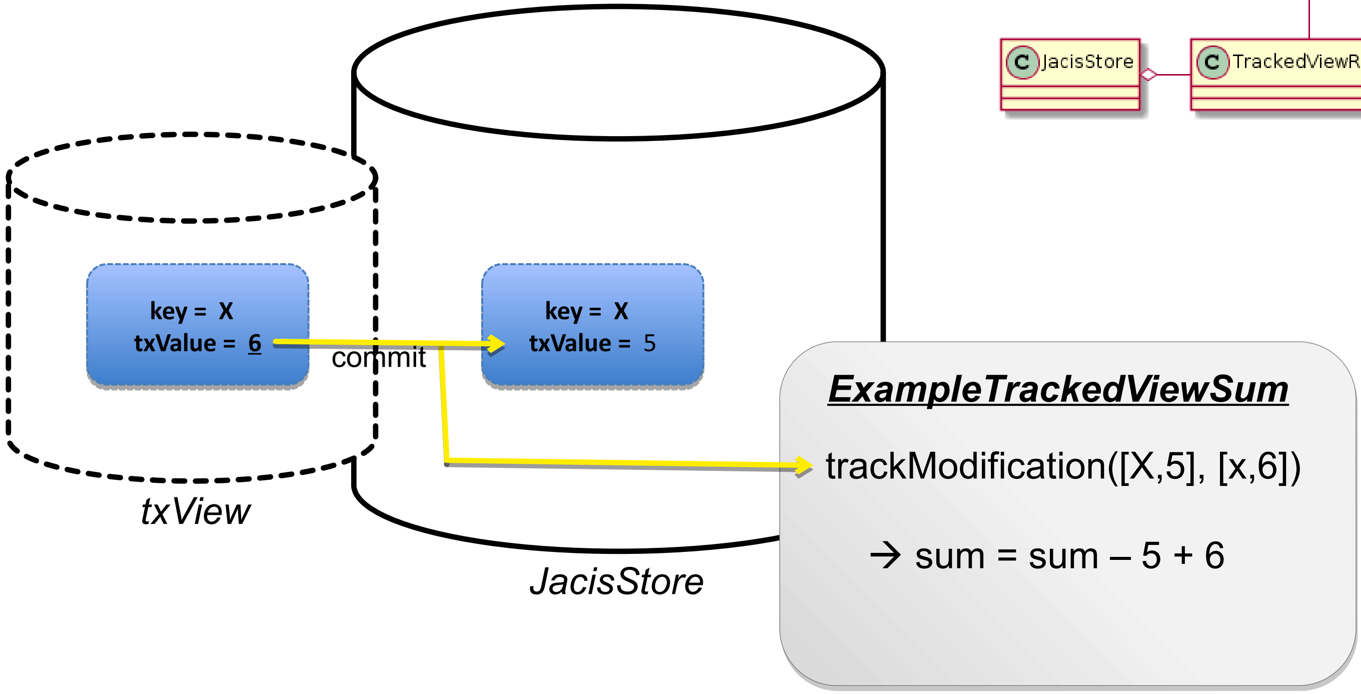
JACIS – MODIFICATION LISTENER


I
JacisModificationListener
onModification(K key, V oldValue, V newValue, JacisTransactionHandle tx)

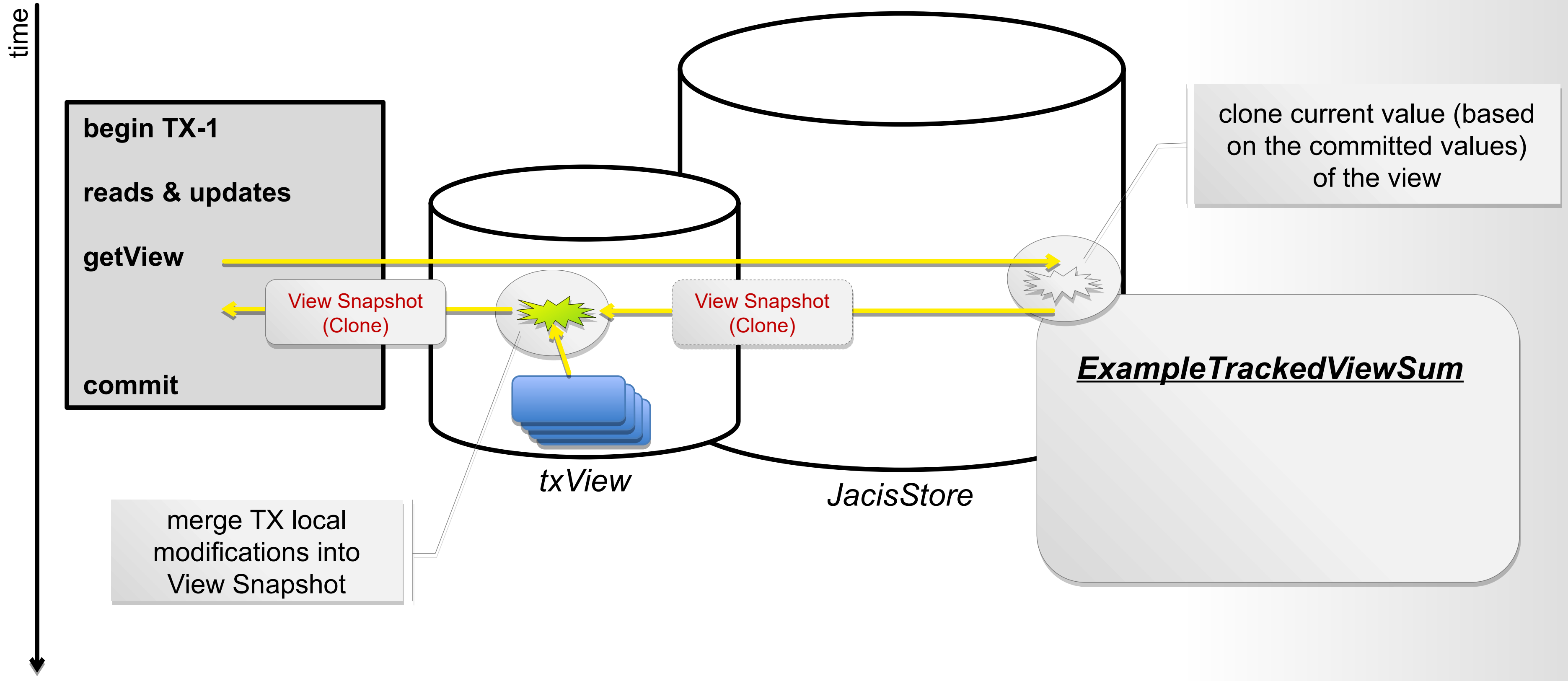
- provides possibility to execute code on each modification
- executed when the modification is done in the store during commit
- register by calling ***JacisStore. registerModificationListener***

JACIS – TRACKED VIEWS

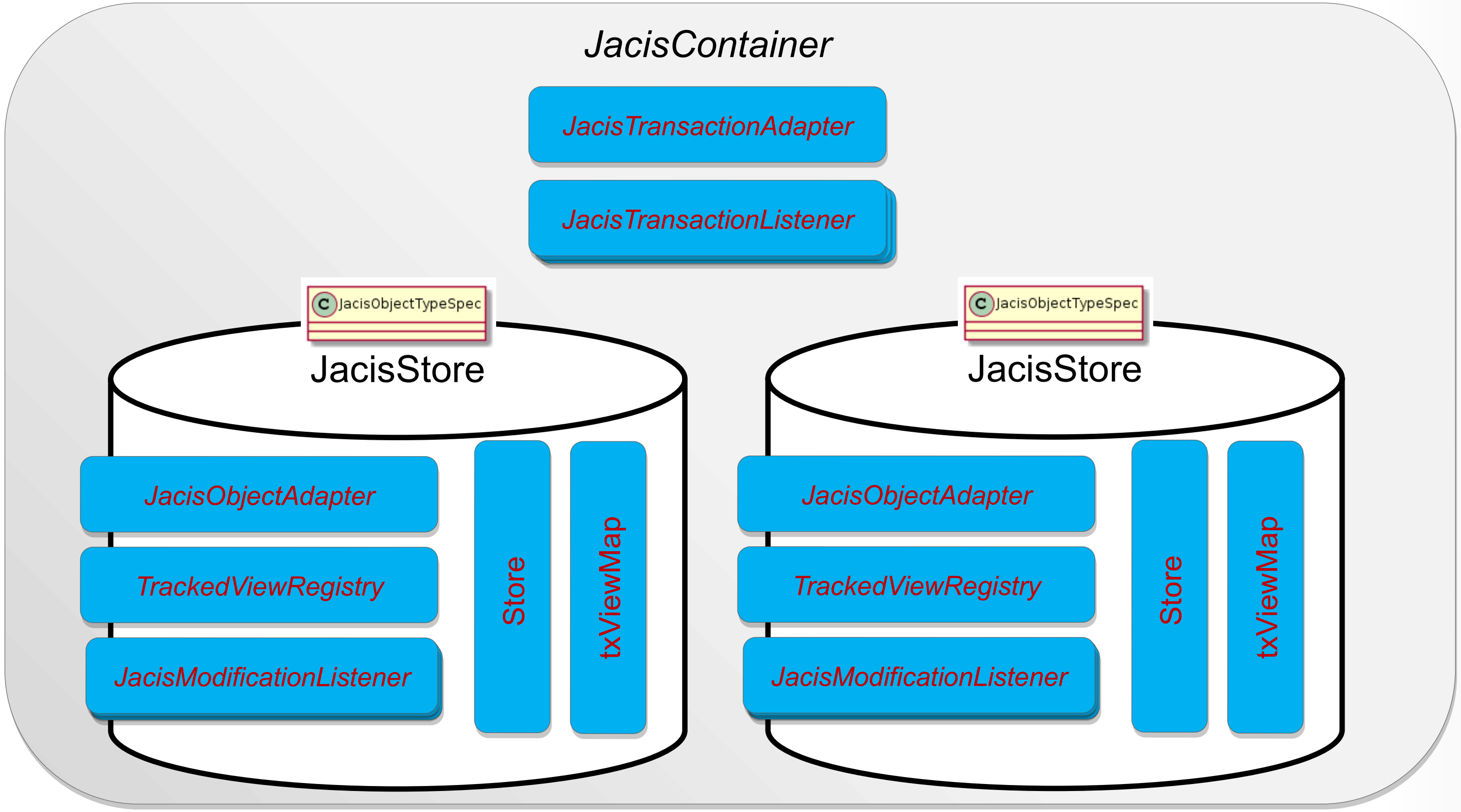




JACIS – TRACKED VIEWS



JACIS – STRUCTURE



JACIS API – EXAMPLE VALUE

```
static class Account extends AbstractReadOnlyModeSupportingObject
                     implements JacisCloneable<Account> {

  private final String name;
  private long balance;

  public Account(String name) {
    this.name = name;
  }

  @Override
  public Account clone() {
    return (Account) super.clone();
  }

  public Account deposit(long amount) {
    checkWritable();
    balance += amount;
    return this;
  }

  public Account withdraw(long amount) {
    checkWritable();
    balance -= amount;
    return this;
  }
…
```

```
…

  public Account withdraw(long amount) {
    checkWritable();
    balance -= amount;
    return this;
  }

  public String getName() {
    return name;
  }

  public long getBalance() {
    return balance;
  }

 }
}
```

JACIS API – CREATE STORE

the root class, containing all stores

```
JacisContainer container = new JacisContainer();
```

key type

value type

```
JacisObjectTypeSpec<String, Account> objectTypeSpec
            = new JacisObjectTypeSpec<>(String.class, Account.class);
```

store containing the values

```
JacisStore<String, Account> store = container.createStore(objectTypeSpec);
```

JACIS API – CREATE OBJECT

```
JacisLocalTransaction tx = container.beginLocalTransaction();

Account account1 = new Account("account1");

store.update(account1.getName(), account1);

tx.commit();
```

key

value

JACIS API – UPDATE OBJECT

executes the lambda
inside a local transaction

```
container.withLocalTx(() -> {

      Account acc = store.get("account1");

      acc.deposit(100);

      store.update("account1", acc);

   });
```

explicitly call the update method!
No dirty check!

JACIS API – GET OBJECT

returns a **writable** instance
(changes are written back to the store
during commit if update was called)

```
container.withLocalTx(() -> {

    Account acc = store.get("account1");

    System.out.println("balance of " + acc.getName() + ": " + acc.getBalance());

  });
```

JACIS API – GET READ-ONLY OBJECT

returns a **read-only** instance
(calls to modifying methods would throw a *org.jacis.exception.ReadOnlyException*)

```
container.withLocalTx(() -> {
  Account acc = store.getReadOnly("account1");
  System.out.println("balance of " + acc.getName() + ": " + acc.getBalance());
});
```

JACIS API – STREAM API

```
  // To cumulate values usually read only access is enough (this is possible without a transaction)
  System.out.println("sum=" + store.streamReadOnly().mapToLong(acc -> acc.getBalance()).sum());

  // streaming the objects starting with a filter
  System.out.println("#>500=" + store.streamReadOnly(acc -> acc.getBalance() > 500).count());

  // as an example to modify some objects add 10% interest to each account with a positive balance
  container.withLocalTx(() -> {
    store.stream(acc -> acc.getBalance() > 0).forEach(acc -> {
      store.update(acc.getName(), acc.deposit(acc.getBalance() / 10));
    });
  });

  // finally output all accounts
  String str = store.streamReadOnly().//
      sorted(Comparator.comparing(acc -> acc.getName())). //
      map(acc -> acc.getName() + ":" + acc.getBalance()).//
      collect(Collectors.joining(", "));
  System.out.println("Accounts: " + str);
}
```

JACIS API – TRACKED VIEW

```
public static class TotalBalanceView implements TrackedView<Account> {

 private long totalBalance = 0;

  @Override
  public void trackModification(Account oldValue, Account newValue) {
    totalBalance += newValue == null ? 0 : newValue.getBalance();
    totalBalance -= oldValue == null ? 0 : oldValue.getBalance();
  }

  public long getTotalBalance() {
    return totalBalance;
  }

 @Override
  public void clear() {
    totalBalance = 0;
  }

 @Override
  public TrackedView<Account> clone() {
    try {
      return (TrackedView<Account>) super.clone();
    } catch (CloneNotSupportedException e) {
      throw new RuntimeException("clone failed");
    }
  }
…
```

```
…

  @Override
  public void checkView(List<Account> values) { // check method for testing
    long checkValue = values.stream().mapToLong(Account::getBalance).sum();
    if (totalBalance != checkValue) {
      throw new IllegalStateException(
"Corrupt view! Tracked value=" + totalBalance + " computed value=" + checkValue);
    }
  }

}
```

```
// Register View:
store.getTrackedViewRegistry().registerTrackedView(new TotalBalanceView());
```