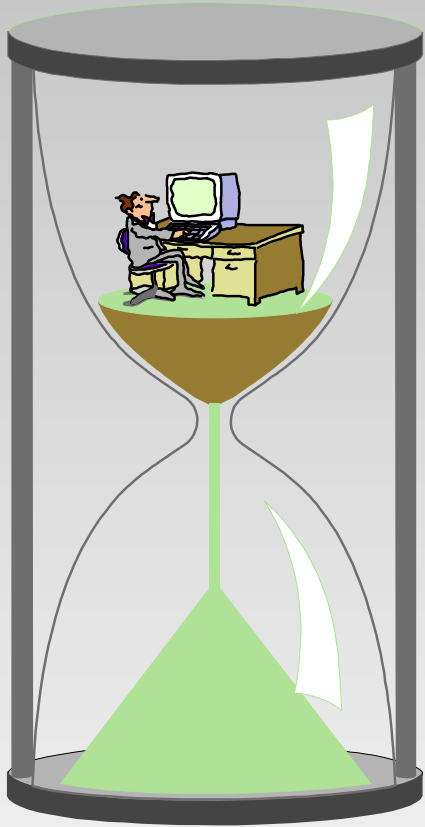
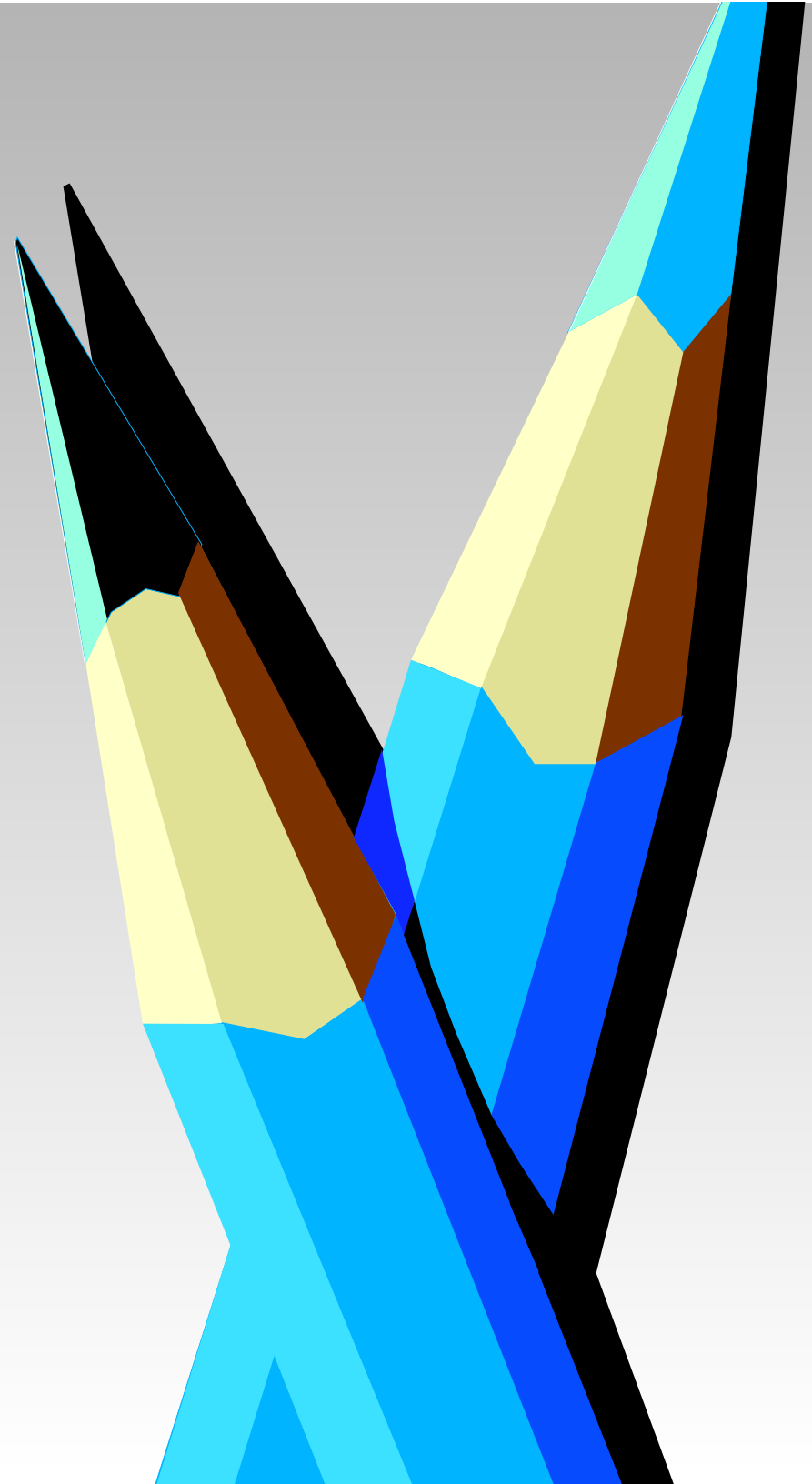


LRUOW-HLS



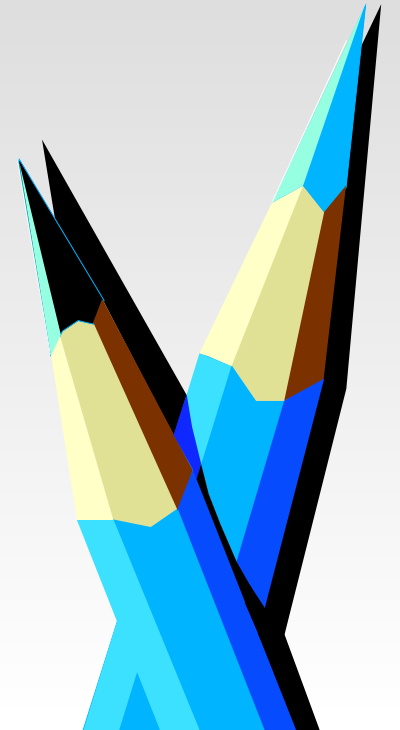
A High-level
Service of the
J2EE Activity
Service

Thomas Mikalsen, Isabelle Rouvellou, Stefan Tai
Advanced Enterprise Middleware
IBM T.J. Watson Research Center
Hawthorne, New York
September 14, 2001



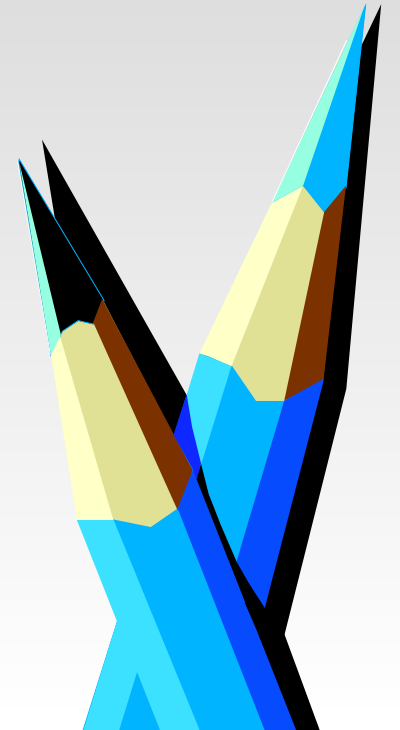
Overview

- ▲ The Problem
- ▲ The LRUOW Approach
- ▲ The Activity Service in a Nutshell
- ▲ LRUOW High-level Service



The Problem

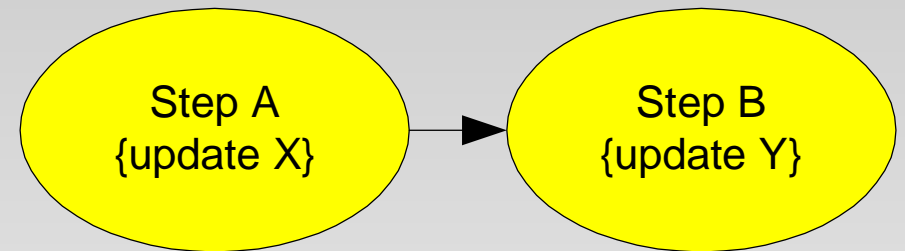
Providing **transactional properties** for **long-running** business processes



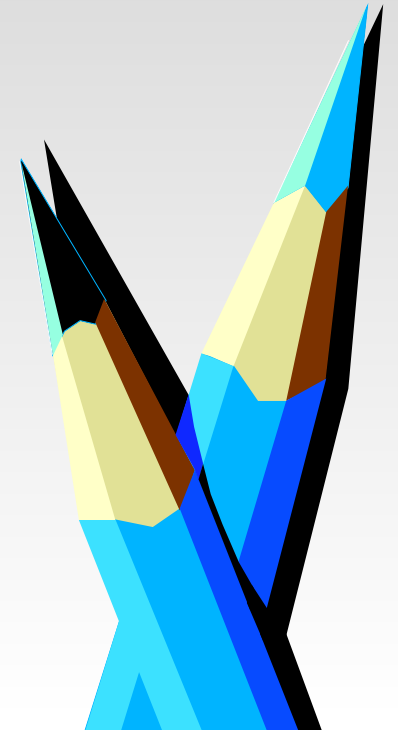
The Problem (cont.)

▲ A simple business process:

- *Step A* manipulates some object *X*
- *Step B* manipulates some object *Y*

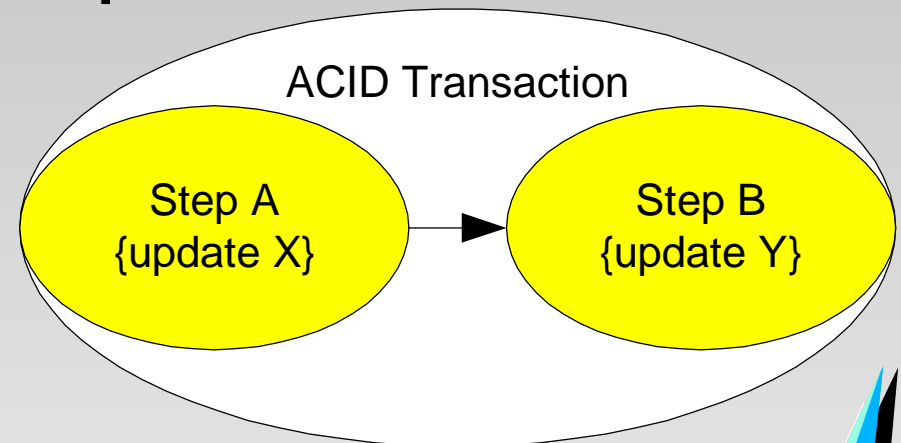


▲ We would like to execute the entire process as an atomic, isolated action



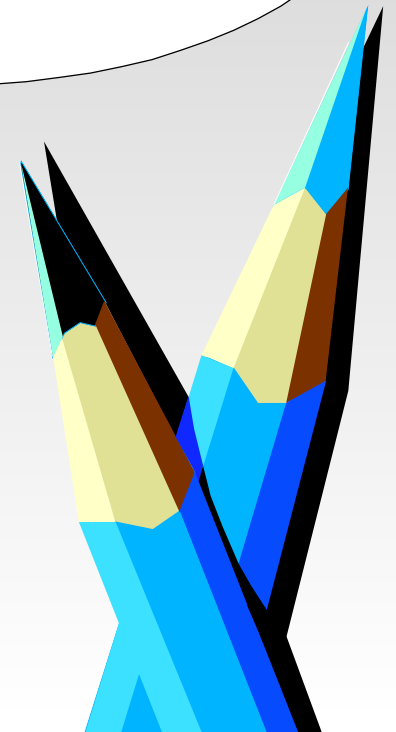
The Problem (cont.)

▲ We could treat the entire process as an ACID transaction:



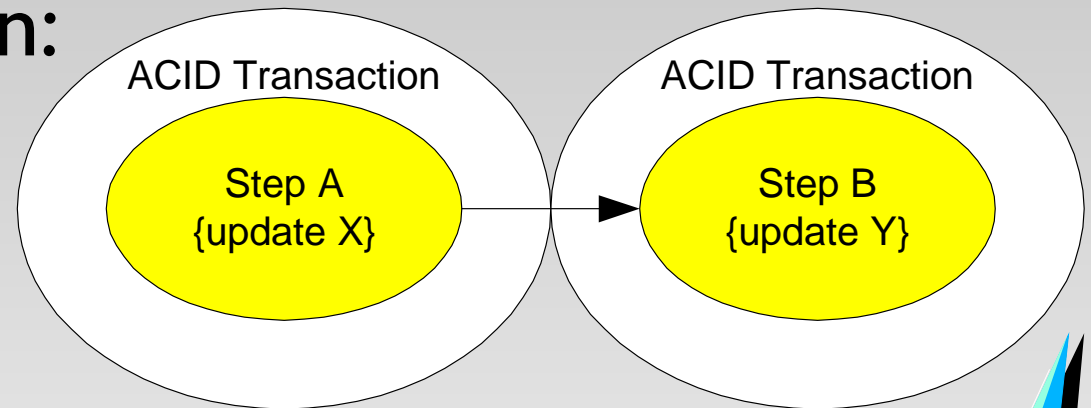
● The problem:

- ▶ introduces **temporal dependencies** between X and Y
- ◆ both X and Y have to be **available at the same time**
- ◆ X **holds locks** until the entire process completes



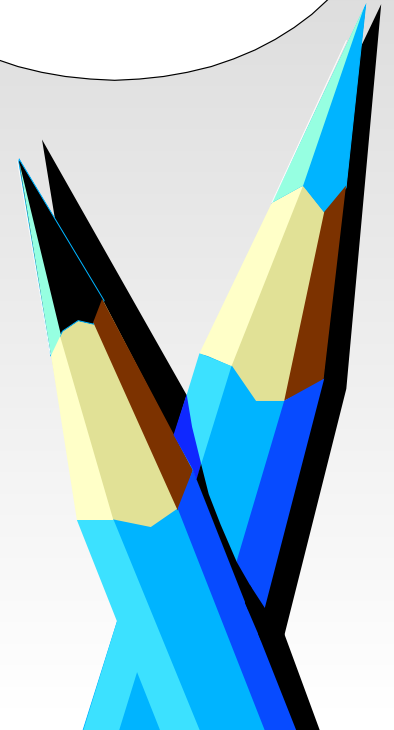
The Problem (cont.)

▲ We could execute each step as a separate transaction:



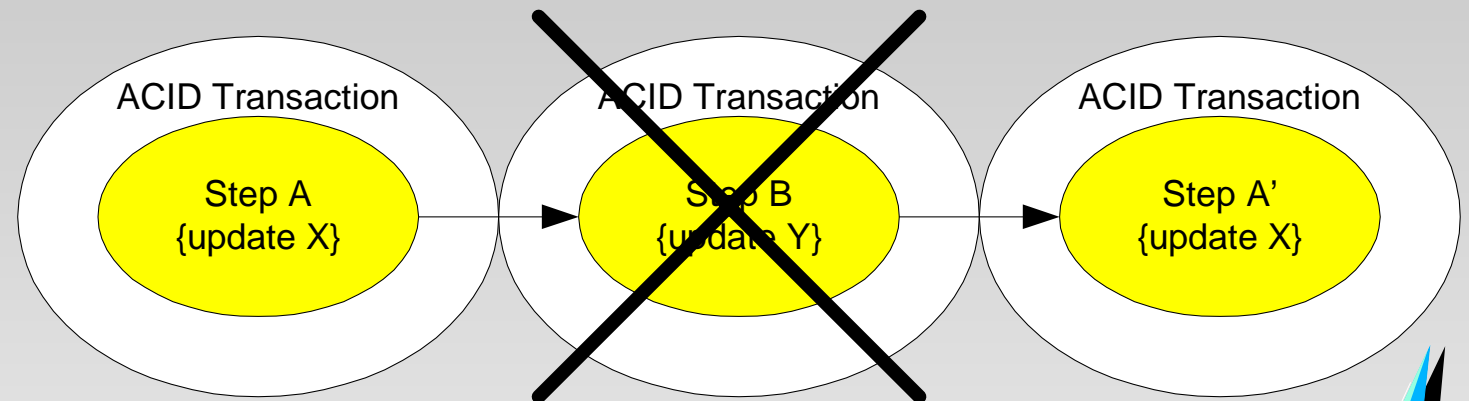
● The Problem:

- ▶ the process is **no longer atomic**
- ▶ the process is **no longer isolated**

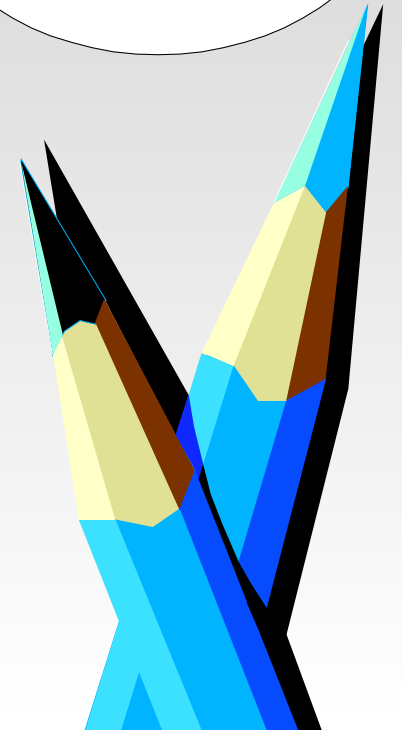


The Problem (cont.)

- ▲ We could use **compensation steps**:

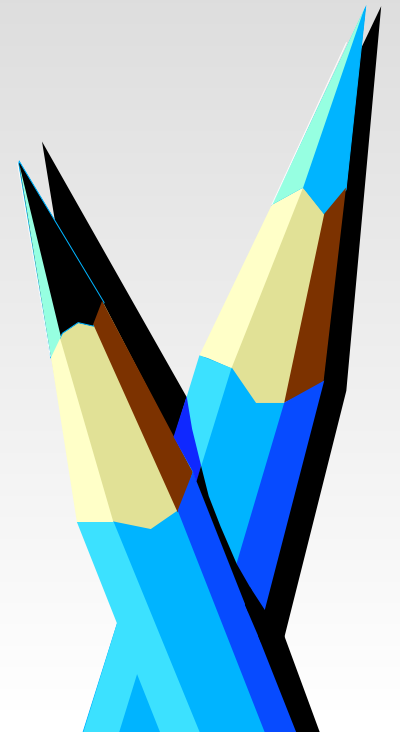


- The problem:
 - ▶ compensation steps are **difficult to get right**
 - ▶ **limited support** for compensation in existing systems



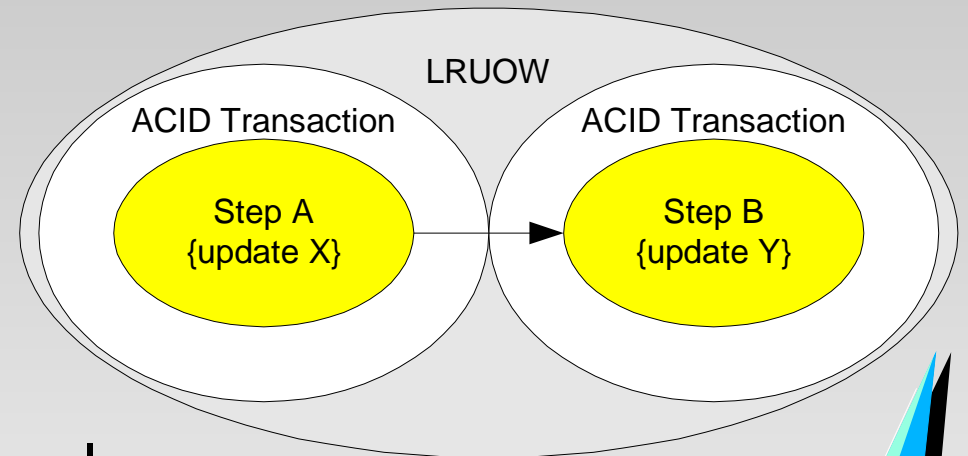
The LRUOW Approach

The **Long-running Unit of Work (LRUOW)** approach addresses the problem by allowing a **long-running** business process to execute as **multiple, transactional (ACID) steps**, while providing **isolation** and **atomicity** for the process as a whole.

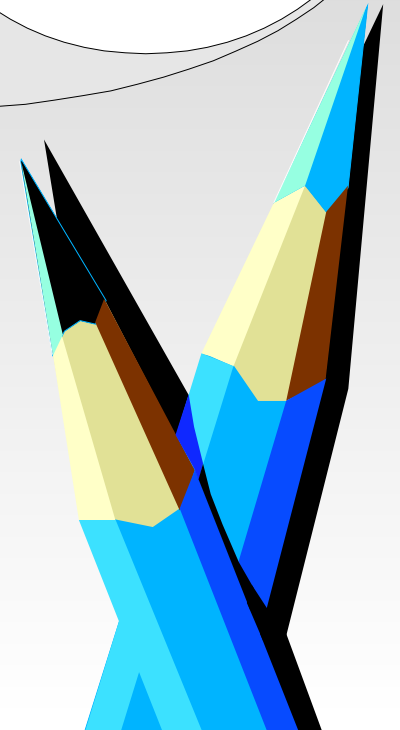


LRUOW Approach (cont.)

▲ Each step executes as an ACID transaction within the context of some LRUOW



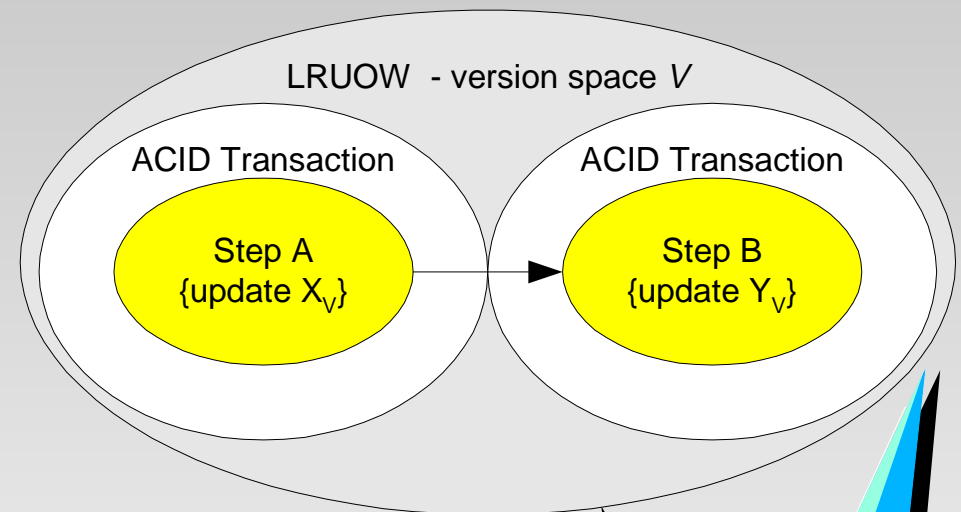
- the effect of each step is visible only to other steps executing in the same LRUOW
- concurrently executing steps within the same LRUOW behave as concurrently executing ACID transactions



LRUOW Approach (cont.)

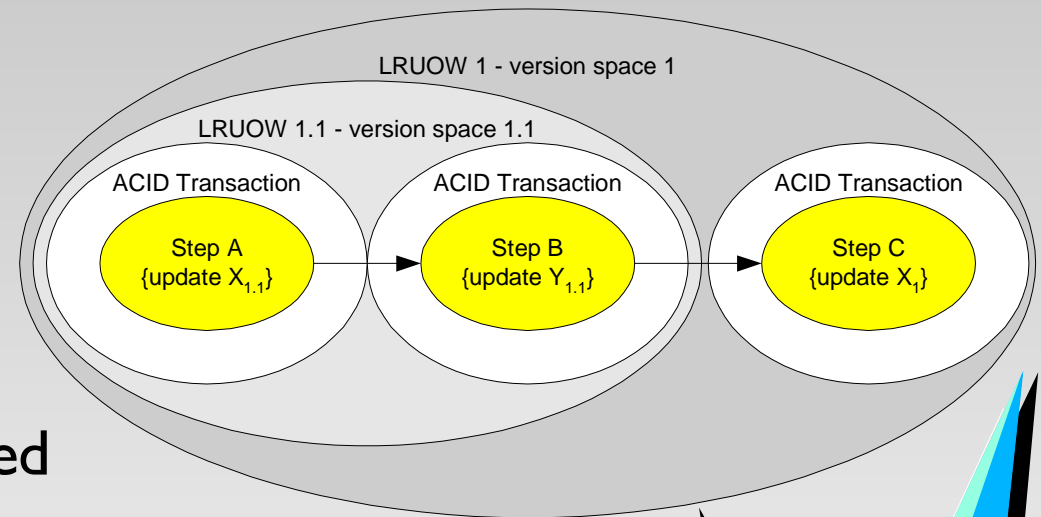
▲ Each LRUOW context creates a **version space**

- steps manipulate **versioned** objects
- the initial state of an object is the version state associated with the **global version space**
- when a LRUOW completes, its version space is **reconciled** with the global version space



LRUOW Approach (cont.)

▲ LRUOW contexts, and associated version spaces, can be **nested**

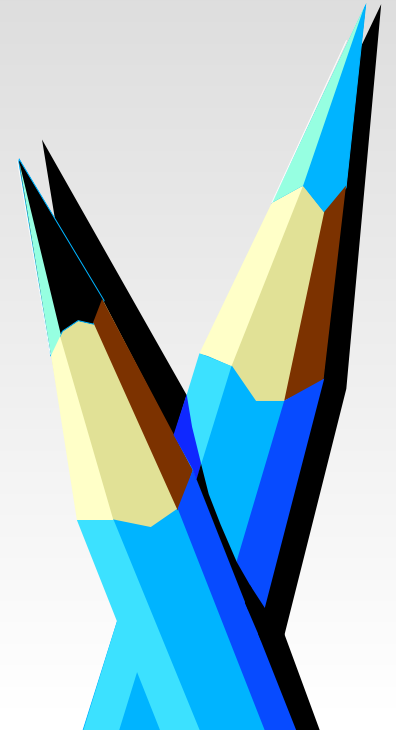


- the initial state of an object is the version state associated with the parent's version space
- when an nested LRUOW completes, its version space is reconciled with its parent's version space

LRUOW Approach (cont.)

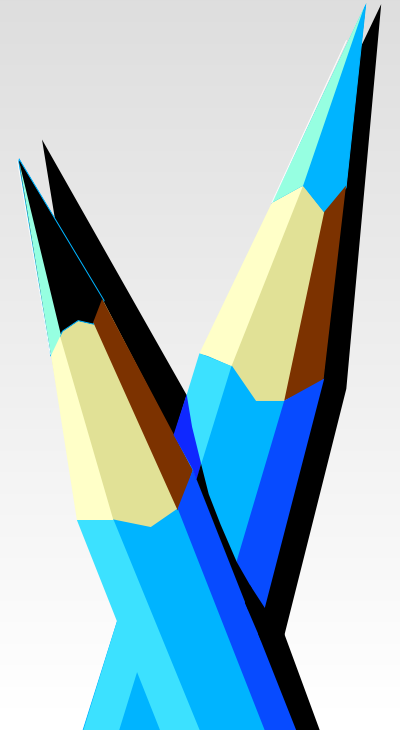
▲ The trick is Version Space Reconciliation

- Two general approaches
 - ▶ **Replay**
 - ◆ log operations performed on objects in child version space
 - ◆ replay logged operations in parent version space
 - ▶ **Conflict Detection and Resolution (CD/R)**
 - ◆ detect object version conflicts
 - ◆ apply algorithms to resolve conflicts



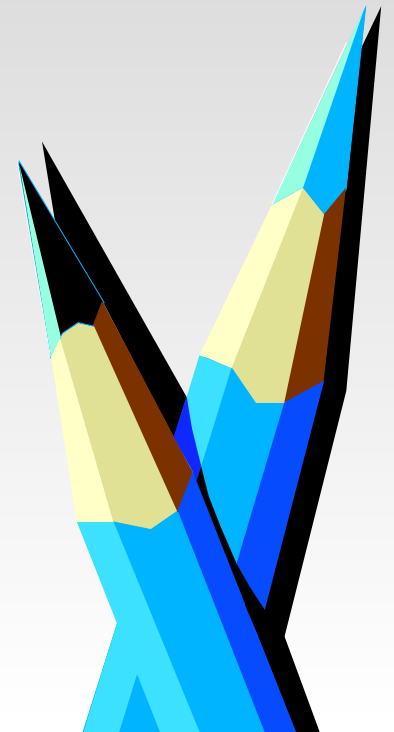
The J2EE Activity Service in a Nutshell

The **J2EE Activity Service (AS)** provides a framework for constructing interoperable, distributed activity coordination services



J2EE Activity Service (cont.)

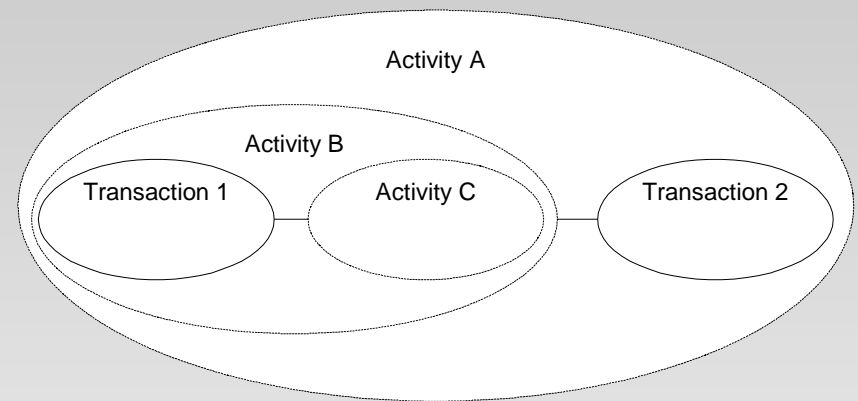
- ▲ A particular activity coordination service is called a **High-level Service (HLS)**
 - Examples of an HLS include
 - ▶ long-running / extended transaction services
 - ▶ workflows engines



J2EE Activity Service (cont.)

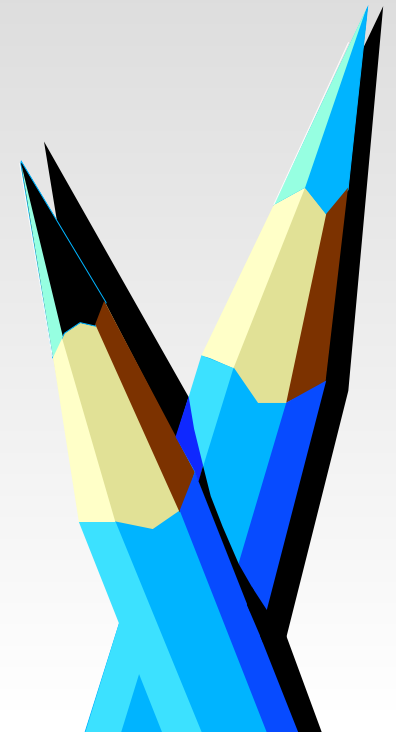
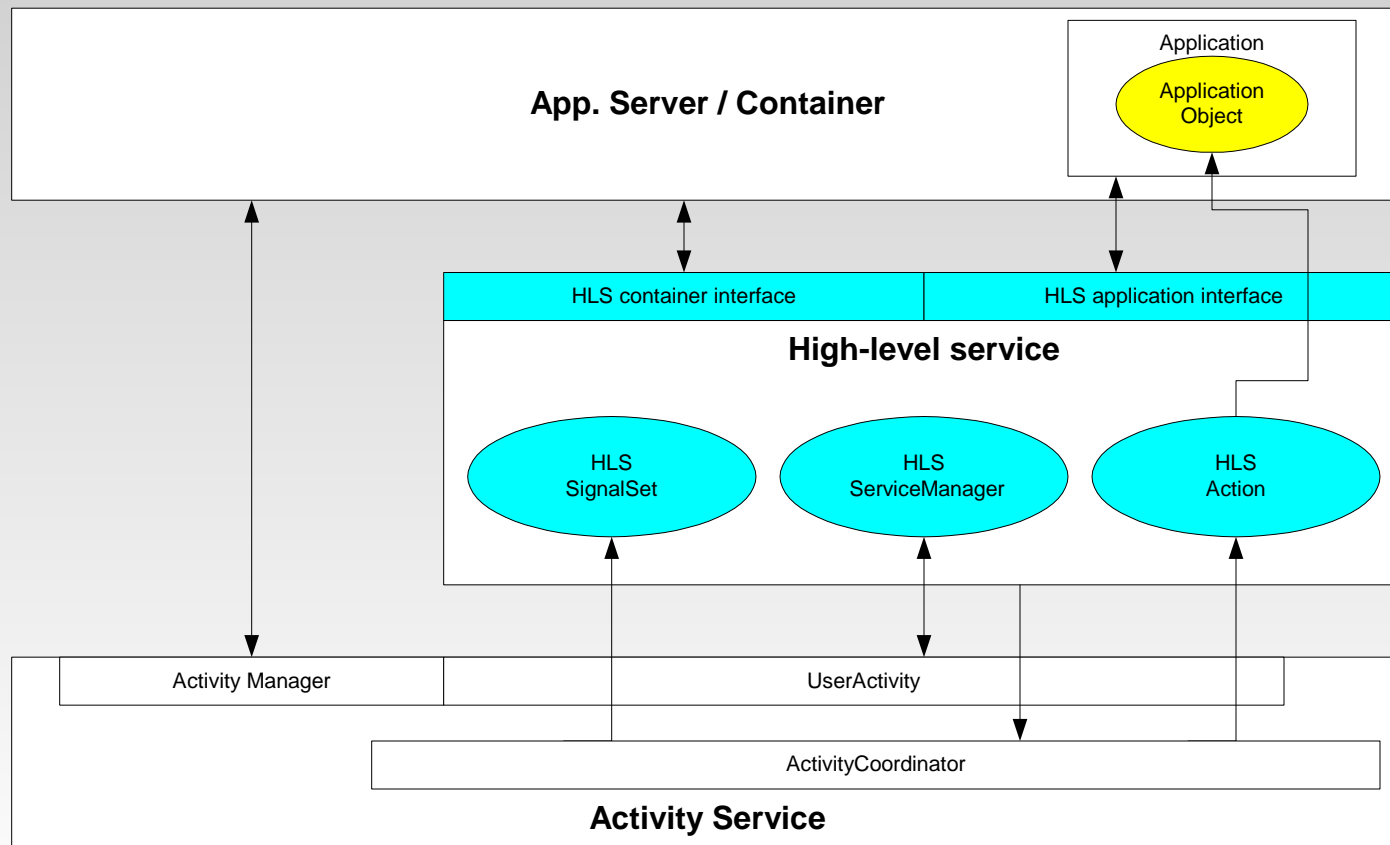
▲ An application process is split into **multiple steps**

- each step is either a **transaction** or an **activity**
 - ▶ JTA coordinates transactions
 - ▶ the HLS provides the logic to coordinate its activities
- the overall application process (e.g., Activity A) can become "transactional" according to the semantics of the particular HLS



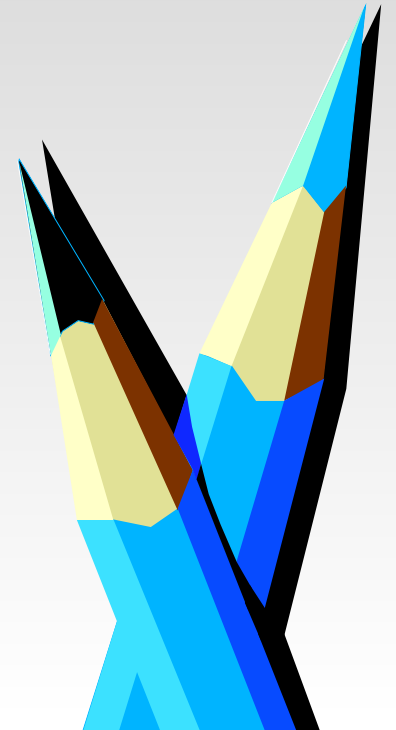
J2EE Activity Service (cont.)

- ▶ An HLS *plugs-into* the AS and Container



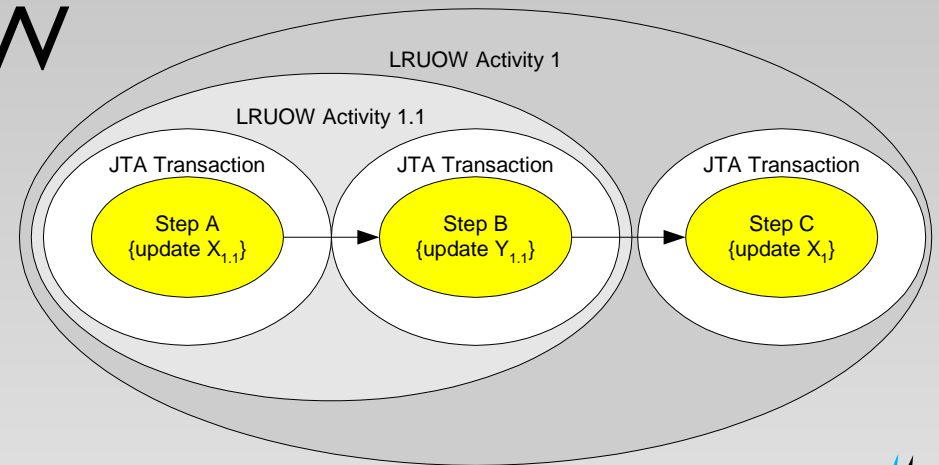
LRUOW High-level Service

The **LRUOW** approach can be implemented **as a High-level Service** of the J2EE Activity Service



LRUOW HLS (cont.)

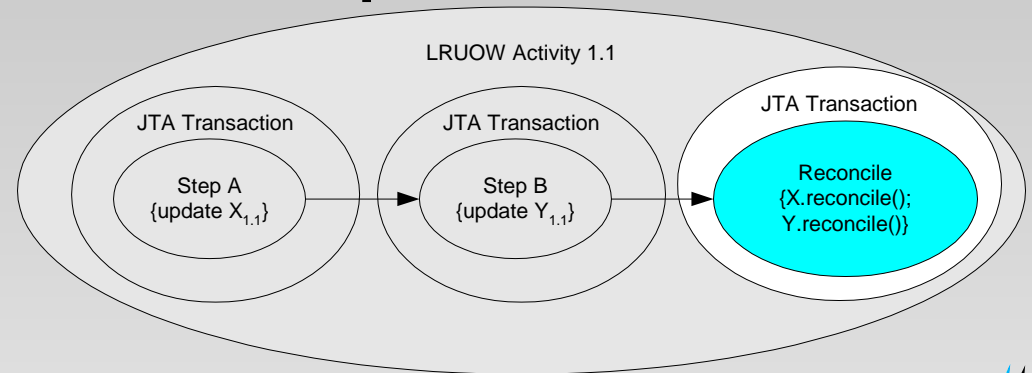
- ▶ We model each LRUOW context as an activity and execute each business process step within a transaction



LRUOW HLS (cont.)

▲ An LRUOW activity has an implicit **reconcile step**

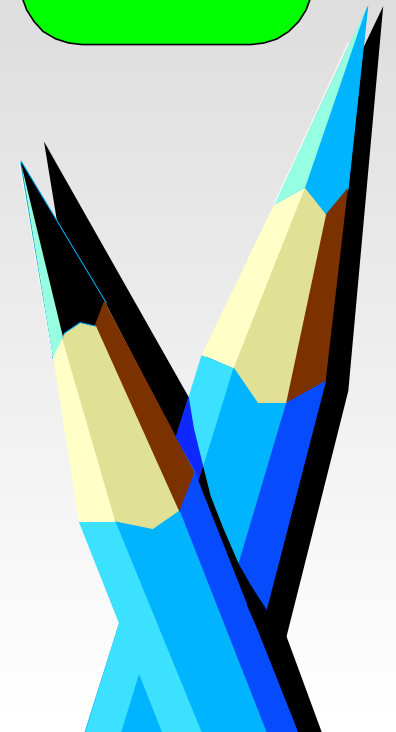
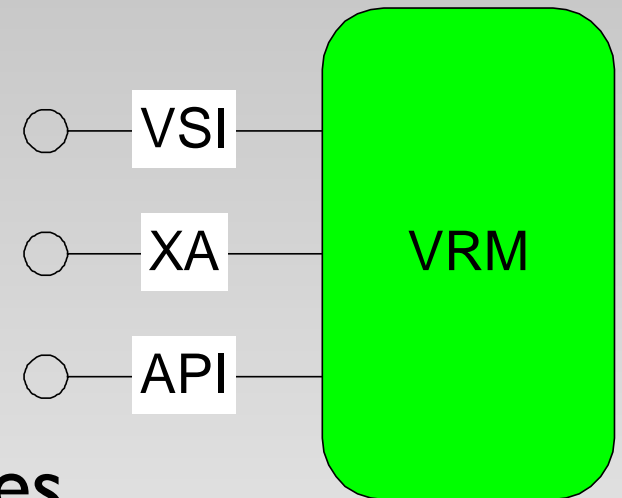
- the reconcile step executes to reconcile the LRUOW's version space
- the reconcile step executes as a transaction
- the reconcile step can *fail*
 - ▶ the application can initiate additional steps to address/correct the failure



LRUOW HLS (cont.)

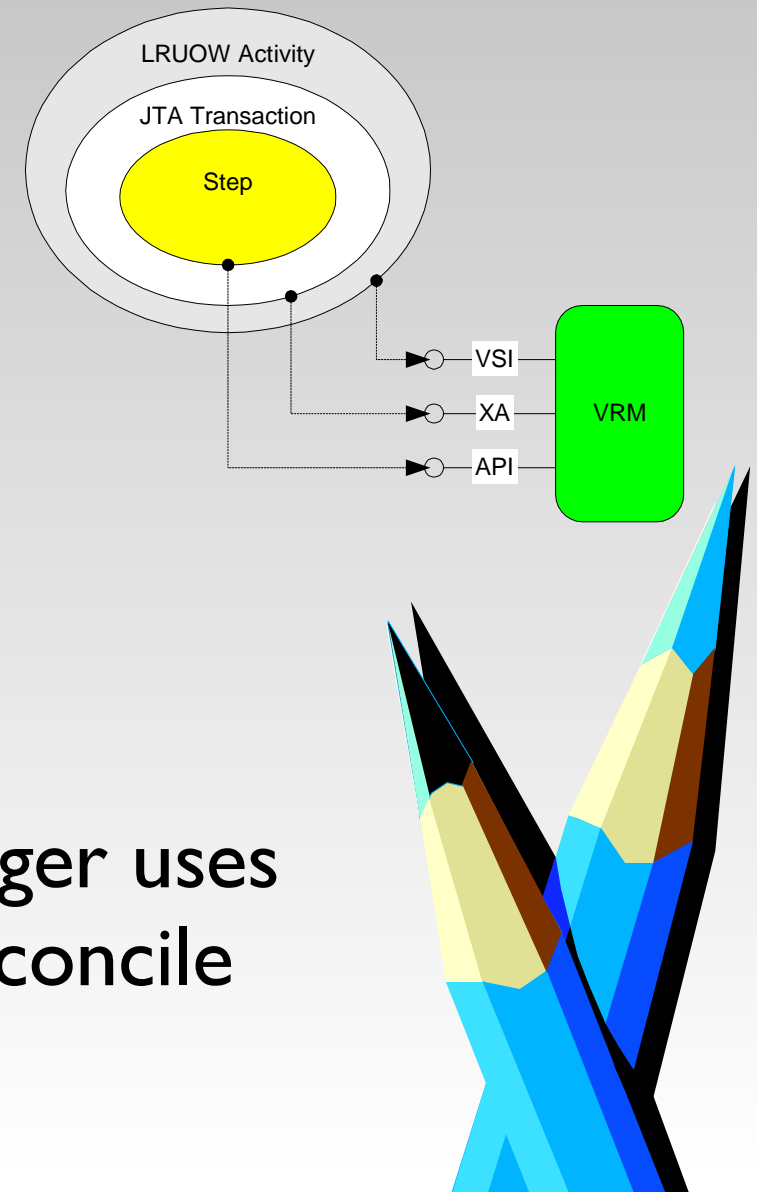
▲ Versioned objects are modeled as **versioned resource managers (VRMs)**

- a VRM supports three interfaces
 - ▶ the **version space interface (VSI)**
 - ▶ the JTA XAResource interface
 - ▶ an application program interface (e.g., SQL)



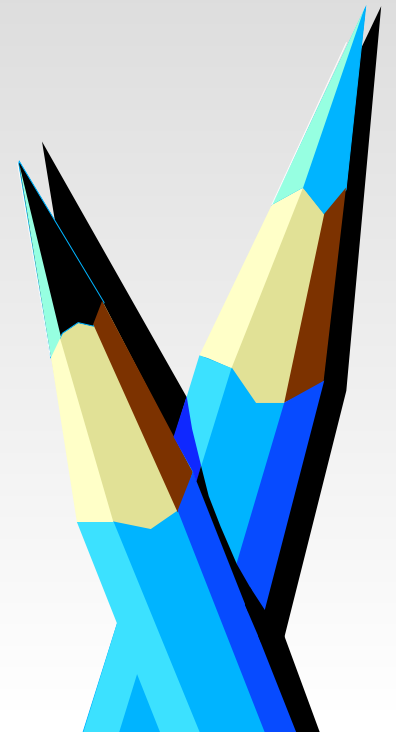
LRUOW HLS (cont.)

- ▶ application steps use the API to manipulate versioned objects
- ▶ a transaction manager uses the JTA/XA interface to coordinate transactions
- ▶ the LRUOW service manager uses the VSI to establish and reconcile version spaces



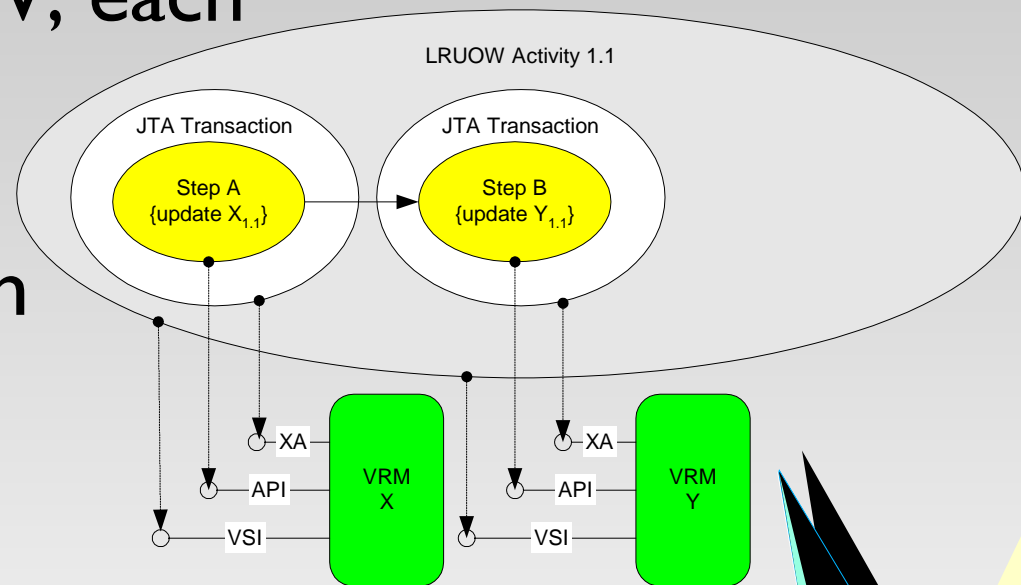
LRUOW HLS (cont.)

- ▶ VSI verbs
 - ◆ `vsi_start(vsid)` - start (or resume) work within a version space
 - ◆ `vsi_end(vsid)` - end (or suspend) work within a version space
 - ◆ `vsi_reconcile(vsid)` - reconcile work performed within a version space
 - ◆ `vsi_abandon(vsid)` - abandon work performed within a version space



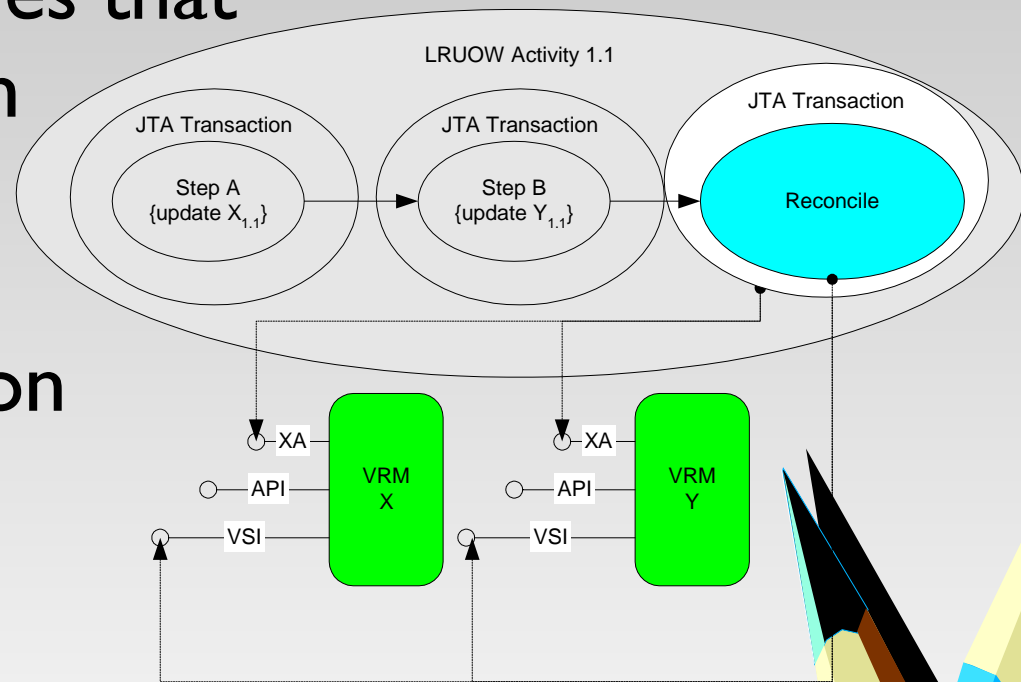
LRUOW HLS (cont.)

- when multiple VRMs are accessed within an LRUOW, each VRM represents a subset of the associated version space



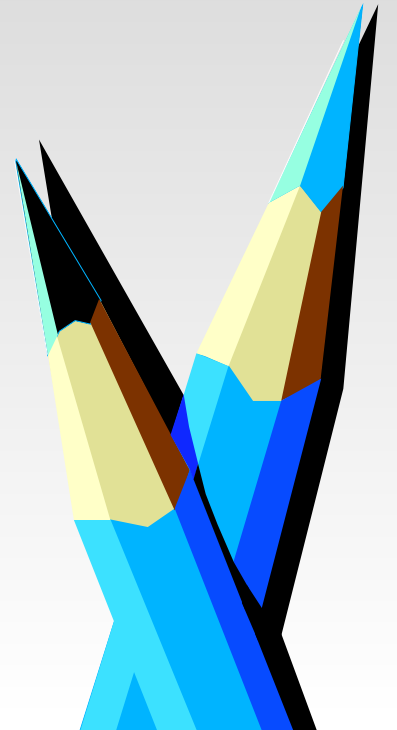
LRUOW HLS (cont.)

- executing the reconcile step as a transaction ensures that the entire version space is reconciled as an atomic action



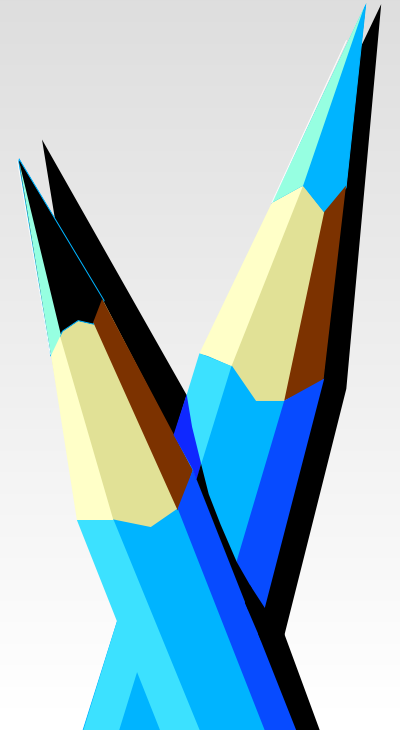
LRUOW HLS (cont.)

- ▲ The LRUOW HLS uses the **signal broadcast** feature of the AS to implement reconcile steps

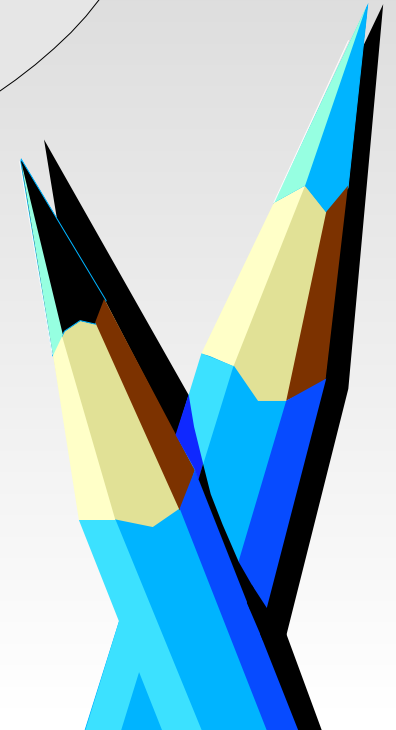
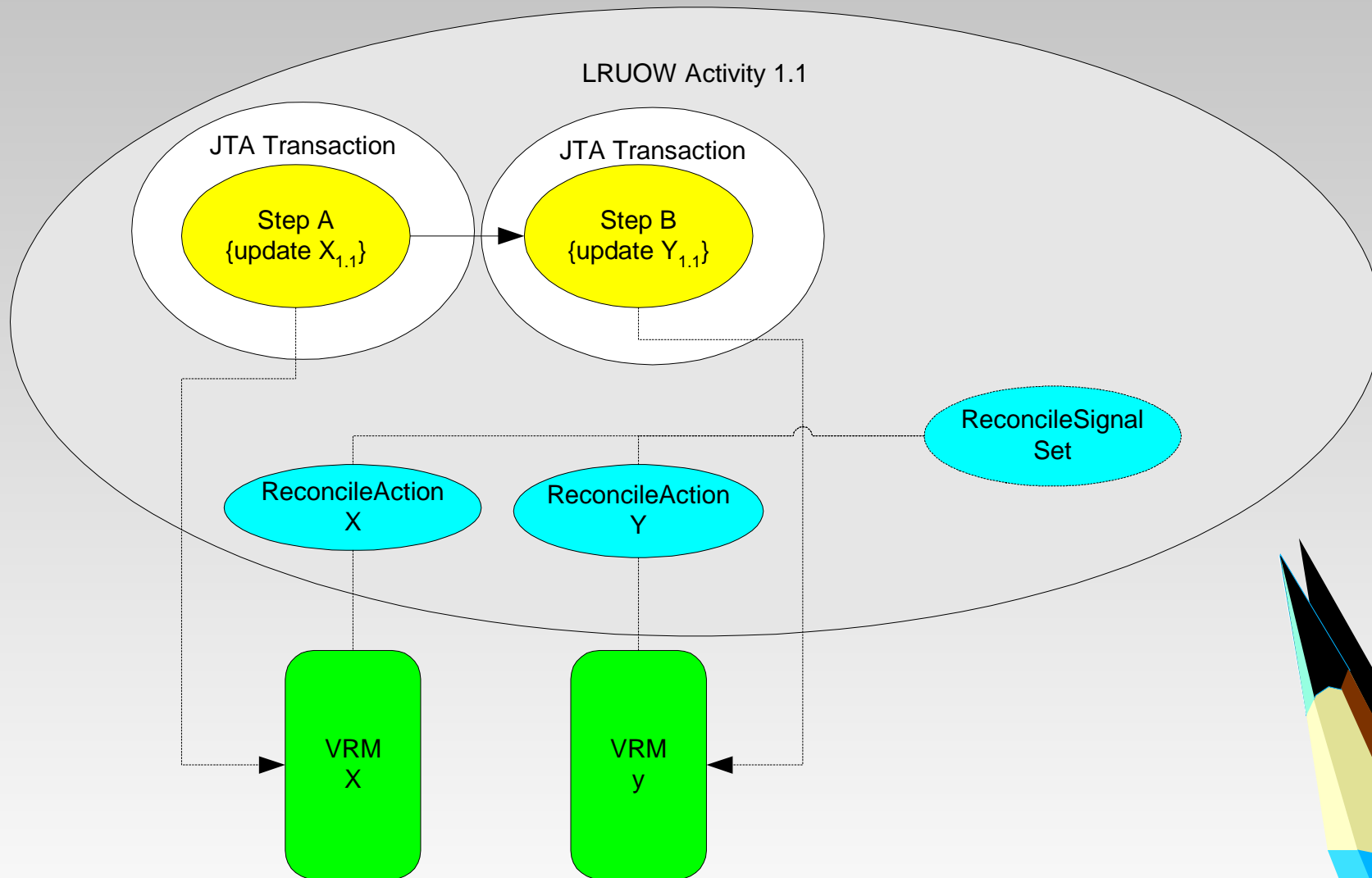


LRUOW HLS (cont.)

- when first accessed, a VRM becomes a participant in the LRUOW activity by adding a `ReconcileAction` to the `ReconcileSignalSet` associated with the LRUOW's activity ...

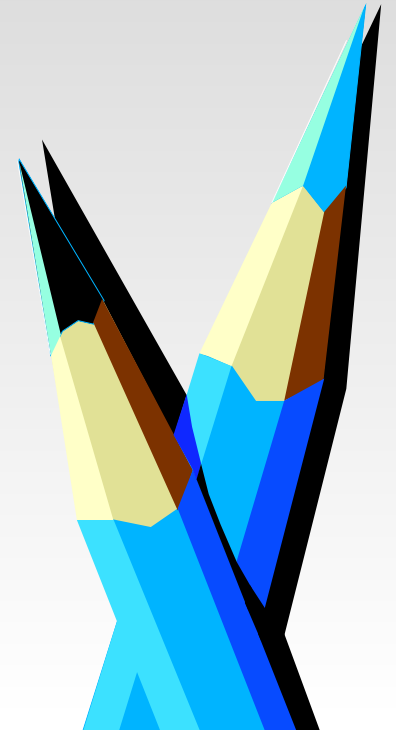


LRUOW HLS (cont.)

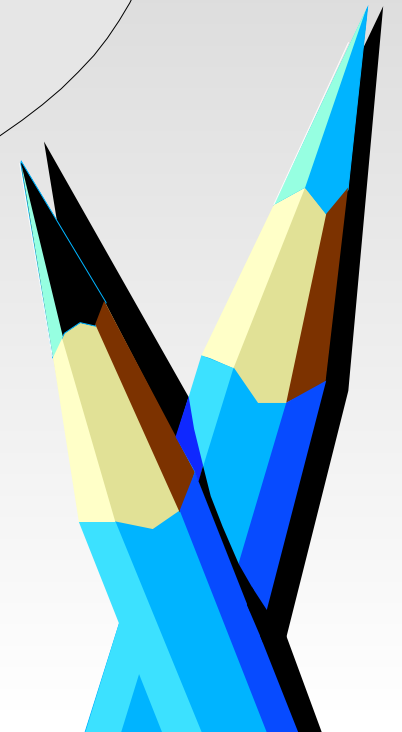
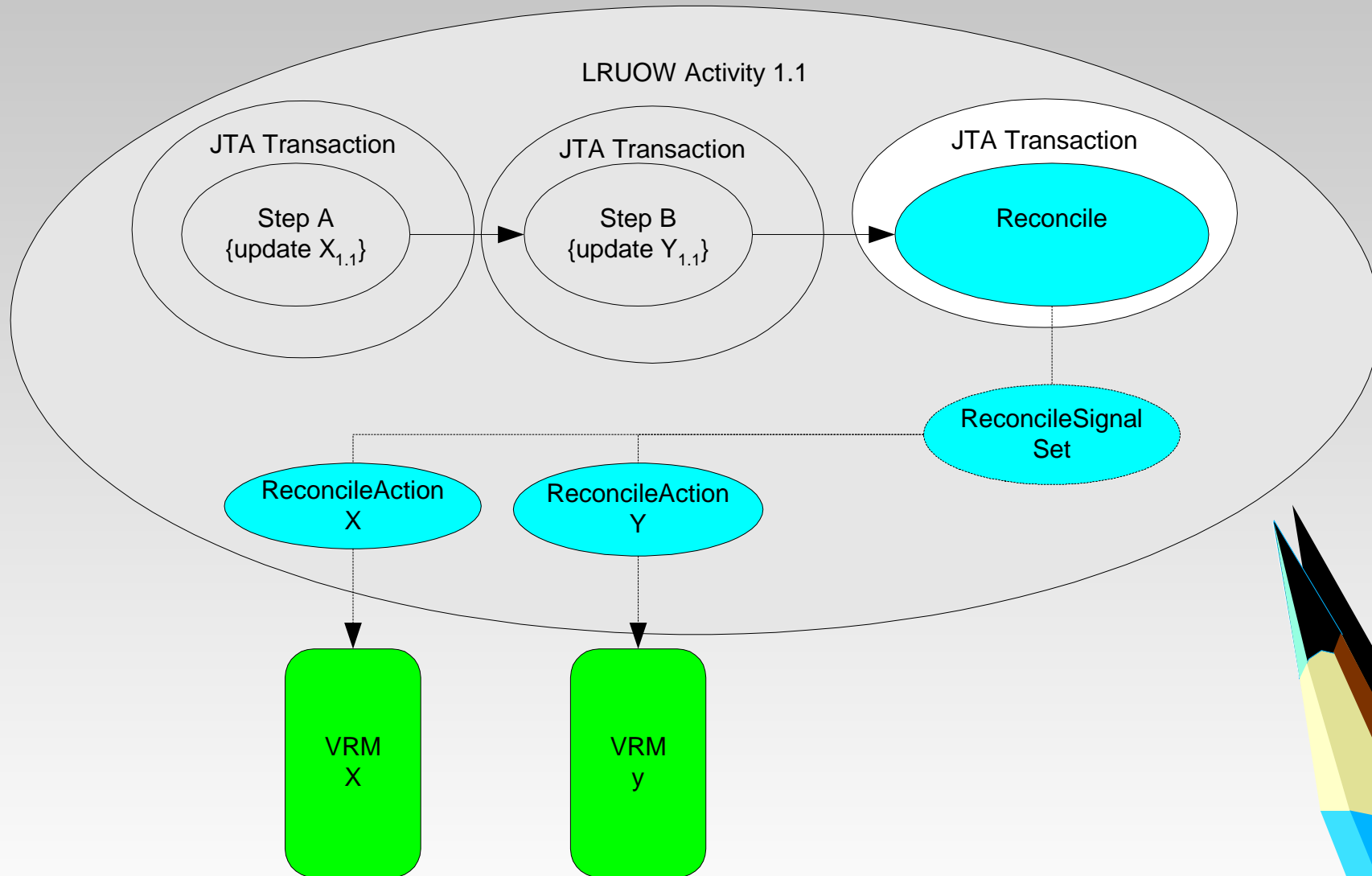


LRUOW HLS (cont.)

- to reconcile the version space associated with an LRUOW, the reconcile step broadcasts **ReconcileSignals** to all registered **ReconcileActions** ...

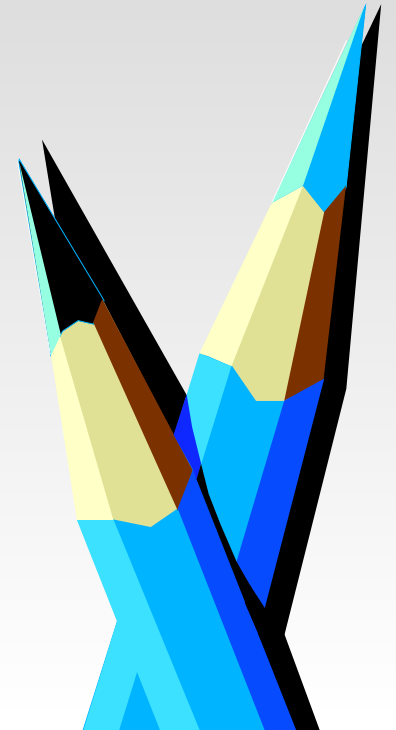


LRUOW HLS (cont.)



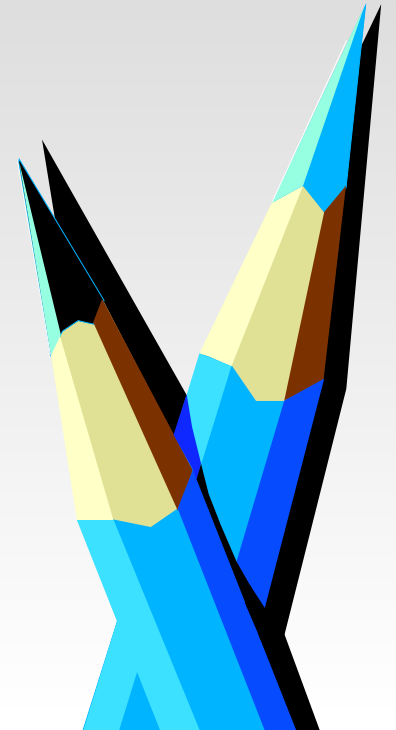
LRUOW HLS (cont.)

- a ReconcileAction produces an **outcome** to the ReconcileSignal
 - ▶ **ReconciledOutcome**
 - ▶ **FailedToReconcileOutcome**
- if any ReconcileAction produces the FailedToReconcileOutcome, the reconcile step is rolled-back

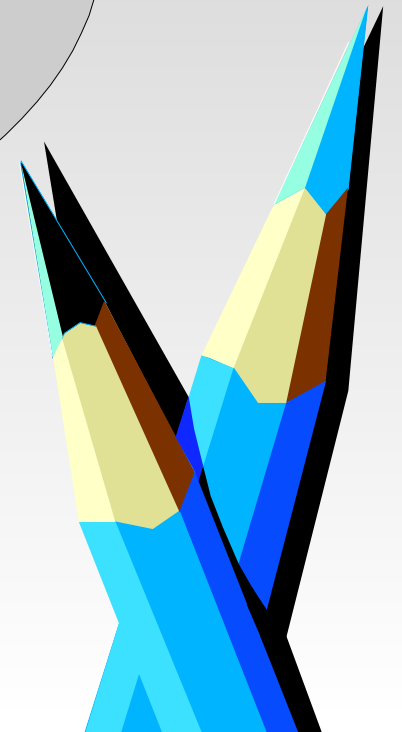
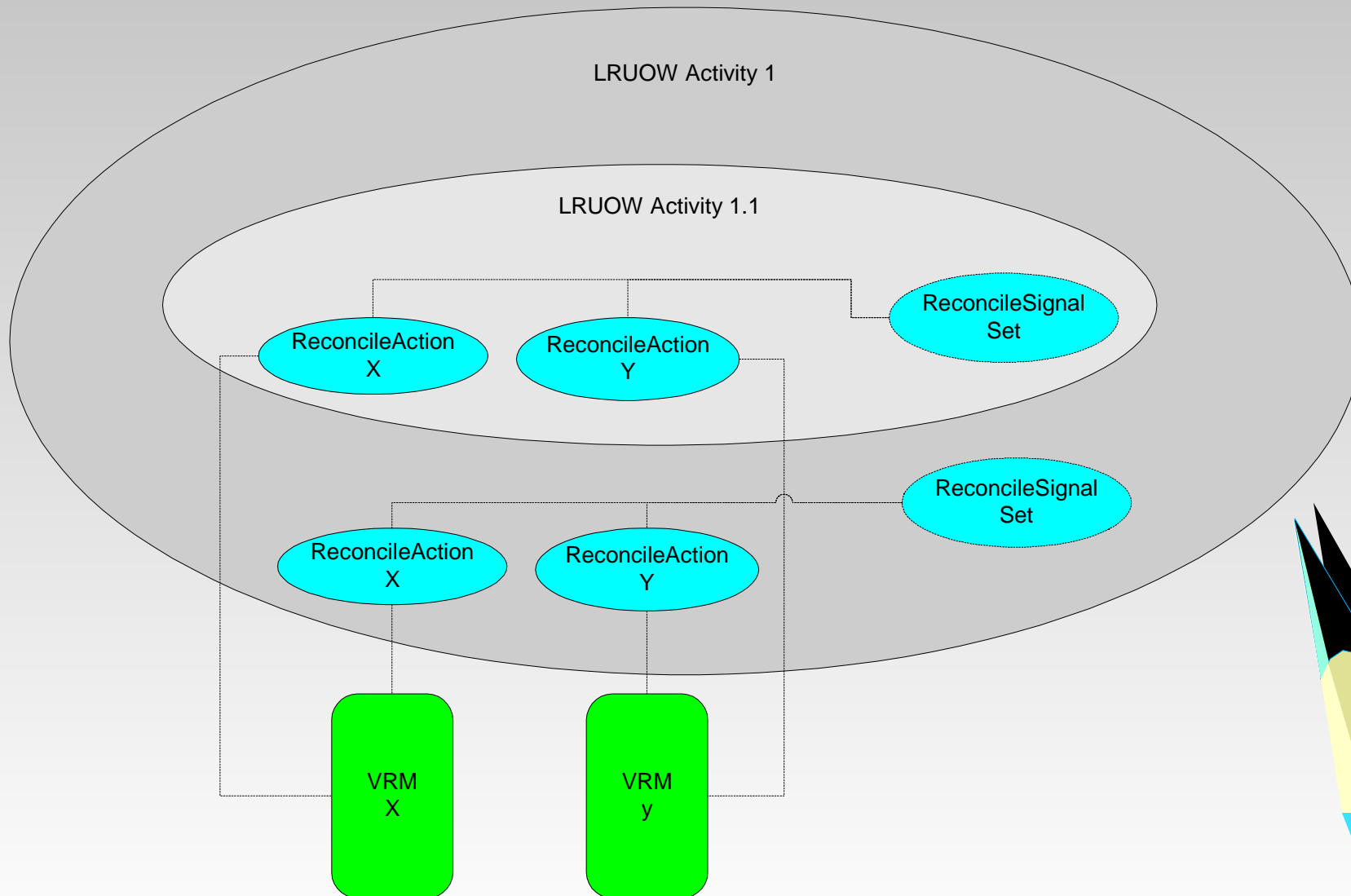


LRUOW HLS (cont.)

- to reconcile a *nested* LRUOW activity, corresponding `ReconcileActions` are added to the `ReconcileSignalSet` associated with the parent LRUOW's activity ...

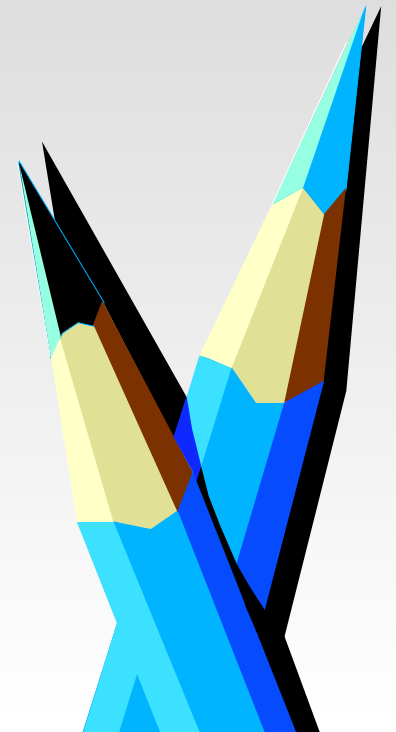


LRUOW HLS (cont.)



LRUOW HLS (cont.)

- ▲ Applications access VRMs indirectly through **VRMAdapters**
 - adapters are typically bound to JNDI names
 - adapters enlist/delist the calling thread in the appropriate transaction and LRUOW context



LRUOW HLS (cont.)

- for database VRMs, the adapter is modeled after JDBC DataSource
 - ▶ e.g., VersionedDataSource
- for other VRMs, the adapter is modeled using JCA (J2EE Connector Arch.) Resource Adapters
 - ▶ e.g., Versioned XML Collection Adapter

