

Transparent Persistence with Java Data Objects

- What is JDO:
 - *Requirements on Transparent Persistence*
 - *Architecture of Java Data Objects*
 - *Available Implementations*
- Applications using JDO:
 - *Trivial*
 - Indicium: *AttributeList/Metadata for LCG*
 - AIDA *Persistence*
 - Minerva: *Lightweight Application Framework*
- Prototypes using JDO:
 - *Object Evolution*
 - *References*



*Objects can be made persistent
without heavy complex systems
polluting user applications.*

What is

Transparent (Orthogonal) Persistence (1)

- Object model independent on persistence
 - Automatic mapping of Java types to native storage types
 - Persistence for 3rd party objects (even when source isn't available)
 - Persistent class source = Transient class source
 - All classes can be persistent
- Illusion of in-memory access to data
 - Implicit update of dirty instances
 - Automatic caching, synchronisation, retrieval, lazy loading
 - Persistence by reachability (all referenced objects are automatically persistent)

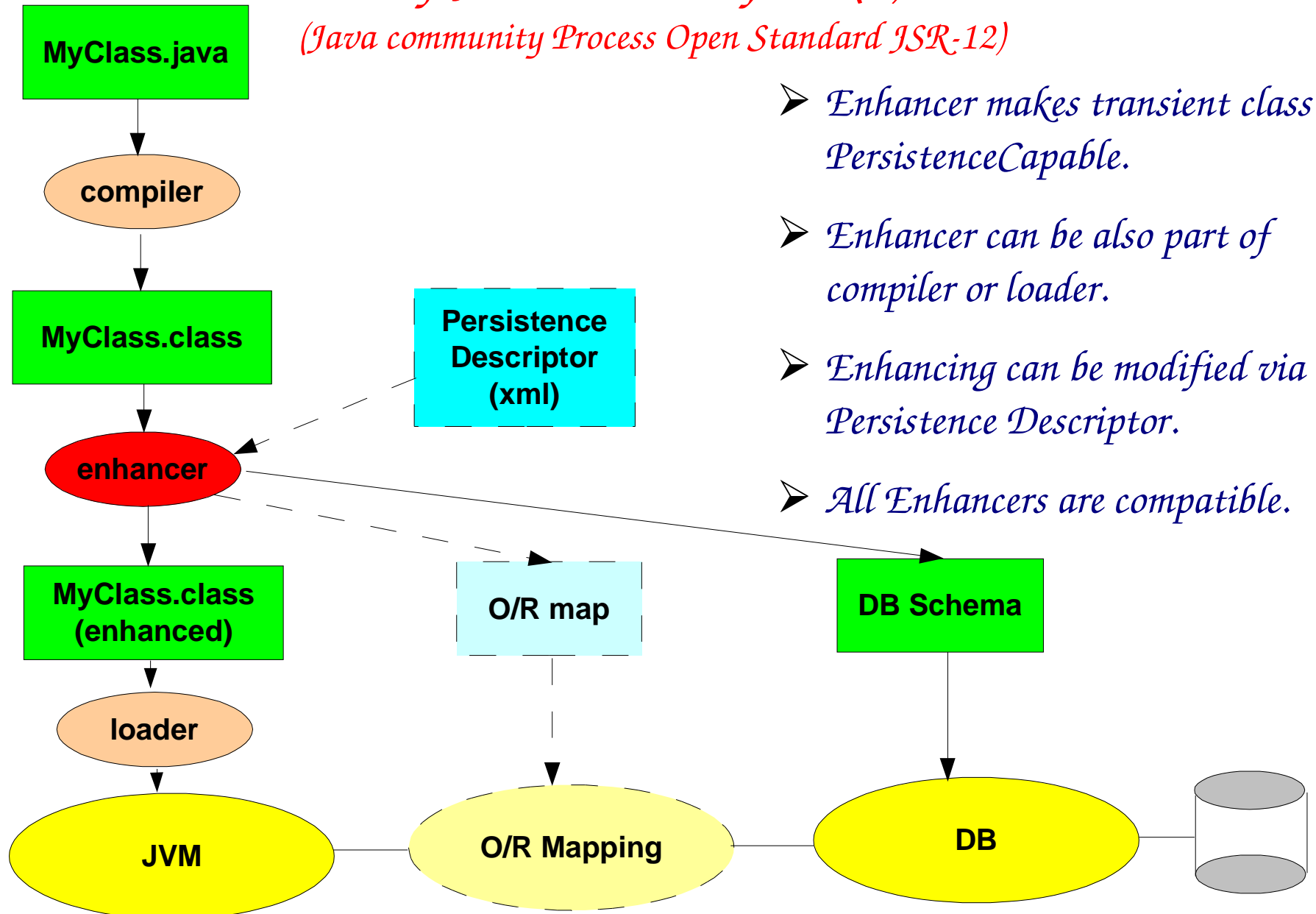
What is

Transparent (Orthogonal) Persistence (2)

- Portability across technologies
 - Data Storage: *RDBS, ODBS, Files,...*
 - API implementations
- Portability across platforms
 - Automatic in Java
- No need for different language (*DDL, SQL, ...*) to handle persistence (incl. queries)
- Interoperability with Application Servers (*EJB,...*)

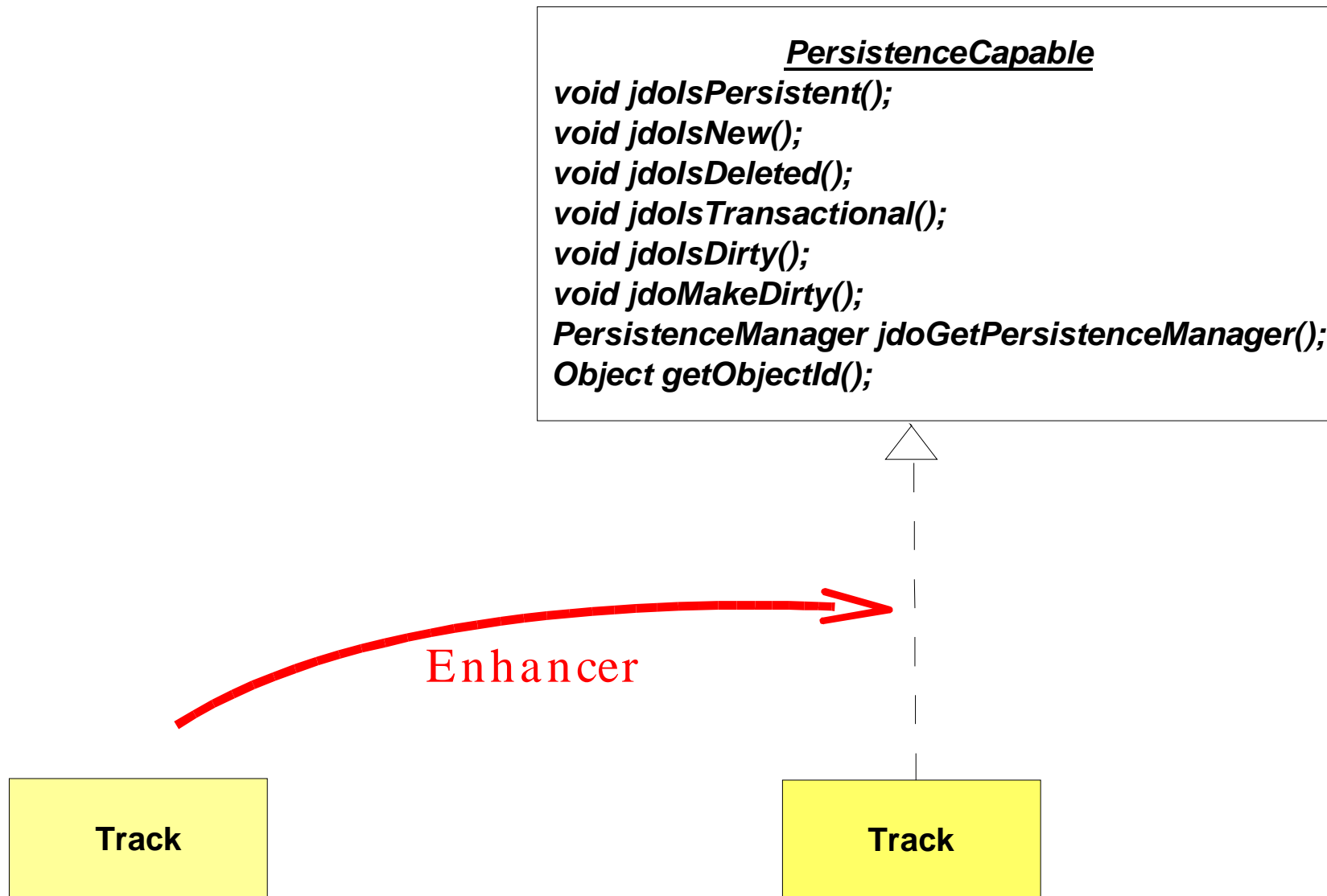
Architecture of Java Data Objects (1)

(Java community Process Open Standard JSR-12)

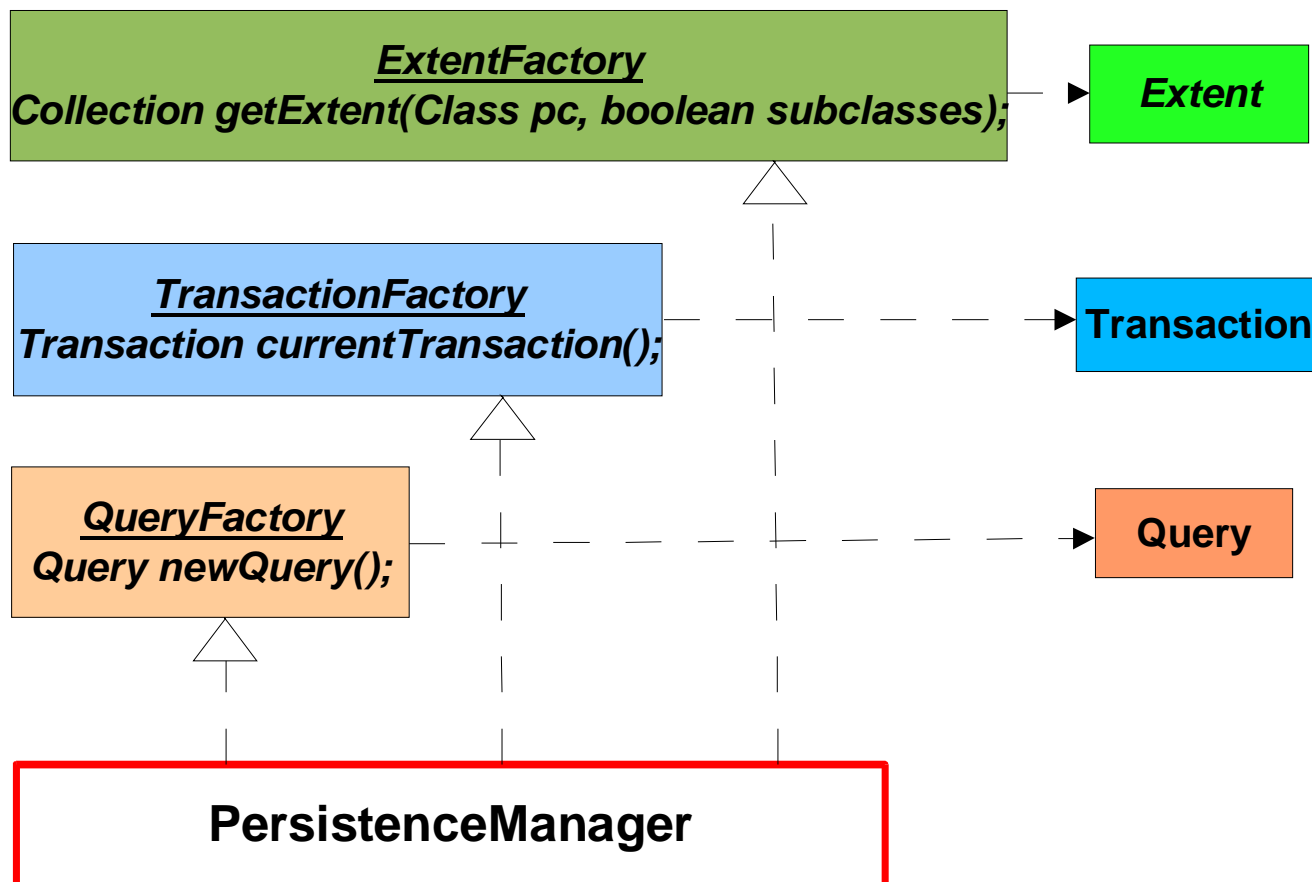


- *Enhancer makes transient class PersistenceCapable.*
- *Enhancer can be also part of compiler or loader.*
- *Enhancing can be modified via Persistence Descriptor.*
- *All Enhancers are compatible.*

Architecture of Java Data Objects (2)



Architecture of Java Data Objects (3)



All interactions with Persistence is mediated by PersistenceManager:

- *Manages instances lifecycle*
- *Factory for Transactions*
- *Factory for Queries*
- *Factory for Extents*

Available Implementations

- Commercial (often with free community license):
 - *enJin(Versant), FastObjects(Poet), FrontierSuit(ObjectFrontier), IntelliBO (Signsoft), JDOGenie(Hemisphere), JRelay(Object Industries), KODO(SolarMetric), LiDO(LIBeLIS), OpenFusion(Prism), Orient(Orient), PE:J(HYWY),...*
- Open:
 - JDORI(Sun): + reference/standard; - only with files
 - TJDO(SourceForge): + high quality, all RDBS, automatically generated schema, full JDO; - inflexible mapping
 - XORM(SourceForge): + reuse of existing schema; - no schema generation, not full JDO
 - JORM(JOnAS/ObjectWeb)
 - OJB(Apache) : + mature mapping engine; - not full JDO

Supported Databases

➤ *RDBS and ODBS:*

➤ *Oracle, MS SQL Server, DB2, PointBase, Cloudscape, MSAccess, JDBC/ODBC Bridge, Sybase, Interbase, InstantDB, Informix, SAPDB, Postgress, MySQL, Hypersonic SQL, Versant,...*

➤ *Files:*

➤ *XML, FOSTORE, flat, C-ISAM, ...*

➤ *JDO performance = DB performance, JDO itself introduces very small overhead.*

Trivial Example

// Initialisation

```
PersistenceManagerFactory pmf = JDOHelper.getPersistenceManagerFactory(properties);
PersistenceManager pm = pmf.getPersistenceManager();
Transaction tx = pm.currentTransaction();
```

// Writing

```
tx.begin();
...
Event event = ...;
pm.makePersistent(event);
...
tx.commit();
```

// Searching using Java-like query language translated internally to DB native query language

// (SQL available too for RDBS)

```
tx.begin();
Extent extent = pm.getExtent(Track.class, true);
String filter = "pt > 20.0";
Query query = pm.newQuery(extent, filter);
Collection results = query.execute();
...
tx.commit();
```

Indicium (1)



- *Mission (as defined by LCG): To define, accumulate, store, search, filter and manage Attributes (metadata) external/additional to existing (Event) data.*
In other words: Better ntuples. Used in the first analysis phase.
- *Related to Collections (of Events).*
- *Satisfied by Java + JDO:*
 - *AttributeSet = Object with Attributes and reference to another (Event) Object*
 - *Explicit Collection = Standard Java Collection*
 - *Implicit Collection (all objects of type T within DB) = Extent*
- *Indicium works with any JDO/DB, the only DB-specific part is DB-management (creation, opening,...).*
- *JDO/DB implementation can be switched via properties file, no re-building is needed. Configuration for JDORI + FOSTORE and TJDO + Cloudscape/MySQL bundled, others are simple to add.*
- *Data stored by Indicium are accessible also via native database protocols (JDBC, SQL) and tools using them.*

Indicium (2)

- *Convenience AttributeSet interface introduced to capture standard AttributeSet Usage Patterns.*
- *Four ways of creating AttributeSet:*
 - Assembled: AttributeSet constructed at run-time; similar to classical n-tuples.
 - Generated: AttributeSet class generated from XML specification.
 - Implementing: AttributeSet interface implemented by hand.
 - FreeStyle: Any class can serve as AttributeSet (some convenience functionality of Indicium is lost here).

Cindicum

- *Indicum C++ interface via automatically created JACE proxies.*
- *AttributeList interface, implementable even in C++, proposed.*

// Construct Signature

```
Signature signature("AssembledClass");
signature.add("j", "int", "Some Integer Number");
signature.add("y", "double", "Some Double Number");
signature.add("s", "String", "Some String");
```

// Obtain Accessor to database

```
Accessor accessor = AccessorFactory::createAccessor("MyDB.properties");
```

// Create Collection

```
accessor.createCollection("MyCollection", signature, true);
```

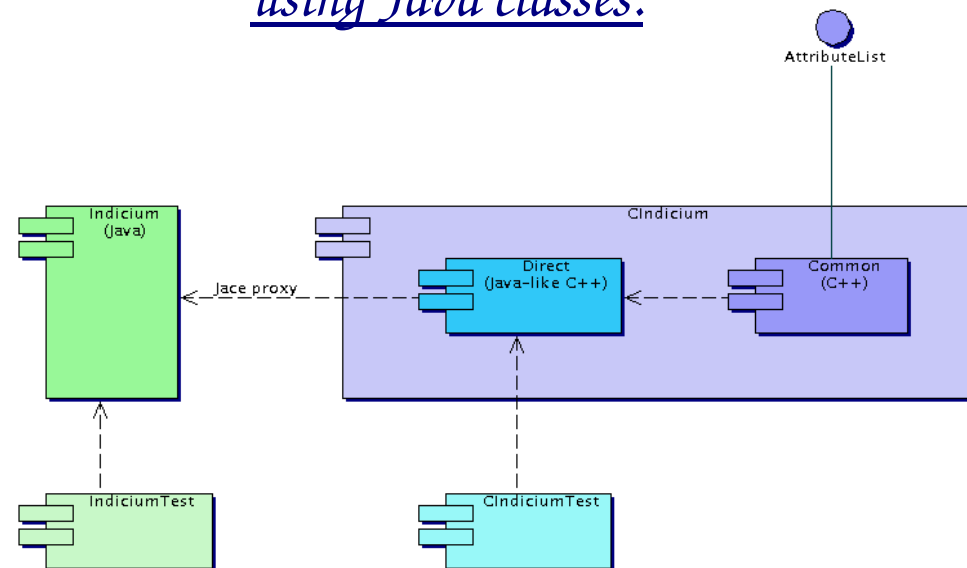
// Write AttributeSets into database

```
AssembledAttributeSet* as;
for (int i = 0; i < 100; i++) {
    as = new AssembledAttributeSet(signature);
    as->set("j", ...);
    as->set("y", ...);
    as->set("s", ...);
    accessor.write(*as);
}
```

// Search database

```
std::string filter = "y > 0.5";
Query query = accessor.newQuery(filter);
Collection collection = query.execute();
std::cout << "First: " << collection.toArray()[0].toString() << std::endl;
```

It is C++ application
using Java classes.



(FreeHEP) AIDA Persistency

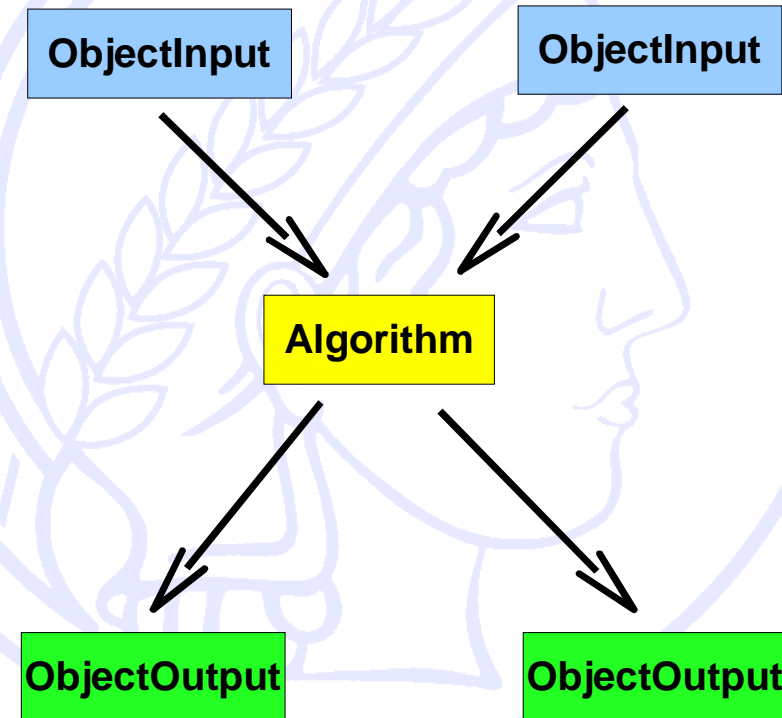
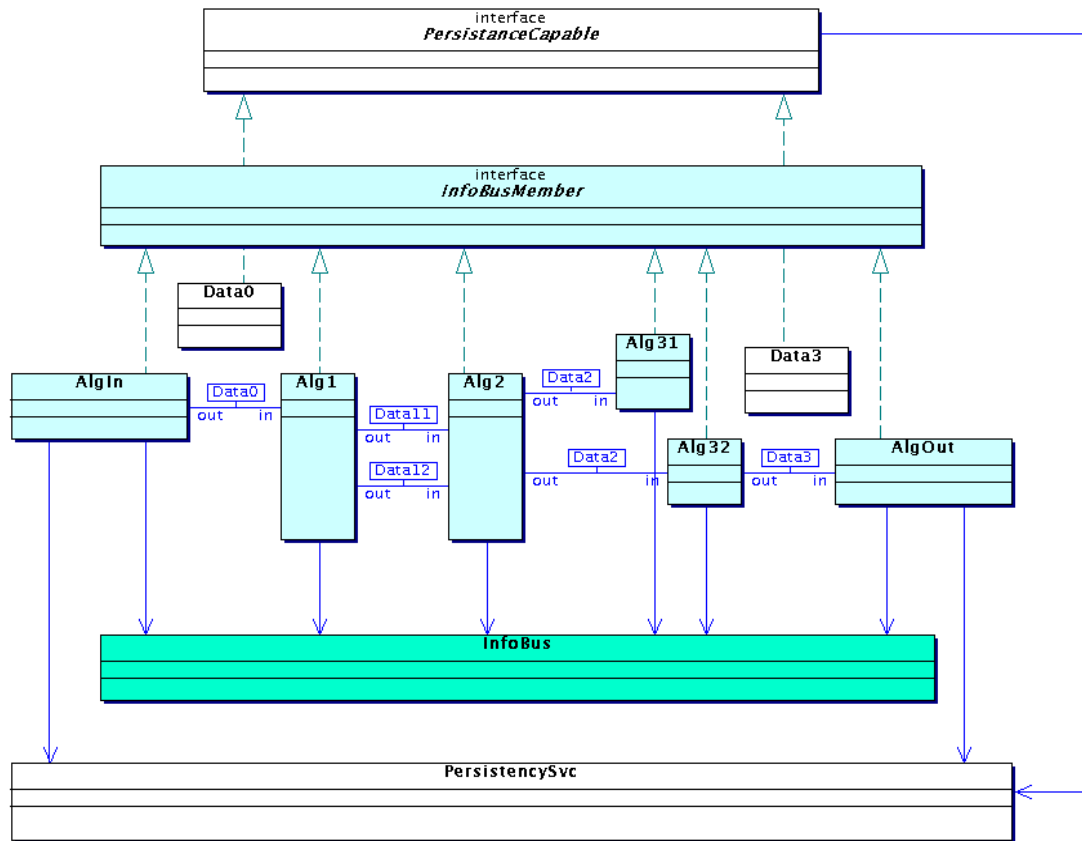
- *AidaJDOStore implements Istore.*
- *The only necessary change to AIDA: Each managed class should have its XML description.*
- *Extensions to existing API:*
 - *Istore should have more control over persistent objects.*
 - *Richer Query API should be introduced.*
- *JDO bug (4779785):*
 - *Persistent subclasses wrongly enhanced in Java 1.4.x.*
 - *Already fixed in JDO CVS, soon in release.*
- *Ituple is also a candidate for another LCG/Pool/AttributeSet API.*

```
<jdo>
<package name="hep.aida.ref.histogram">
  <class name="Histogram2D"
    persistence-capable-superclass="hep.aida.ref.histogram.Histogram">
  </class>
</package>
</jdo>
```

Minerva (1)

- Lightweight Java Framework implementing main Architecture principles of Athena/Gaudi:
 - Algorithm - Data Separation
 - Persistent - Transient Separation
 - Implementation independence
 - Modularity
- Based on InfoBus:
 - Data Producers + Data Consumers
 - Declared I/O types of Algorithms
 - Implicit scheduling
 - Algorithms and Services as Servers
- Multithreaded

Minerva (2)



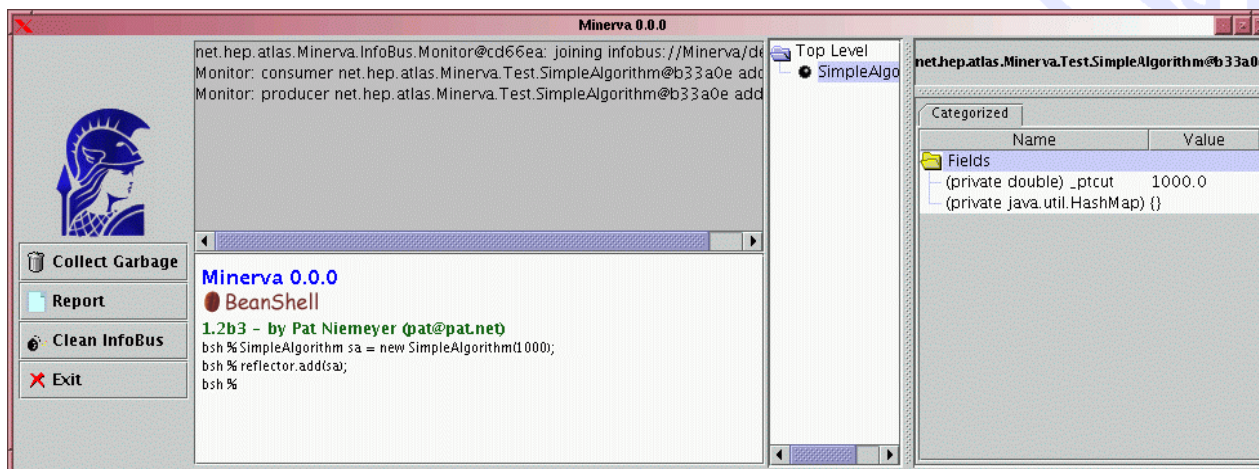
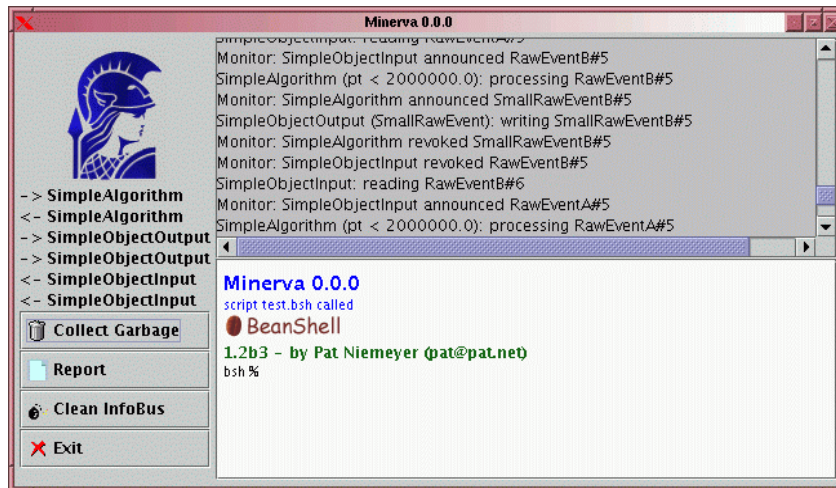
Script

```
new Algorithm(<Algorithm properties>);
new ObjectOutput(<dbO1>, <Event characteristics1>);
new ObjectOutput(<dbO2>, <Event characteristics2>);
new ObjectInput(<db1>);
new ObjectInput(<db2>);
```

*5 Servers are running in parallel
in this example.
They read data from two databases,
process them and
write to other two databases.*

Minerva (3)

*Running set of Producers/Consumers
created from the script.*



*Using ObjectBrowser
to inspect Algorithm.*

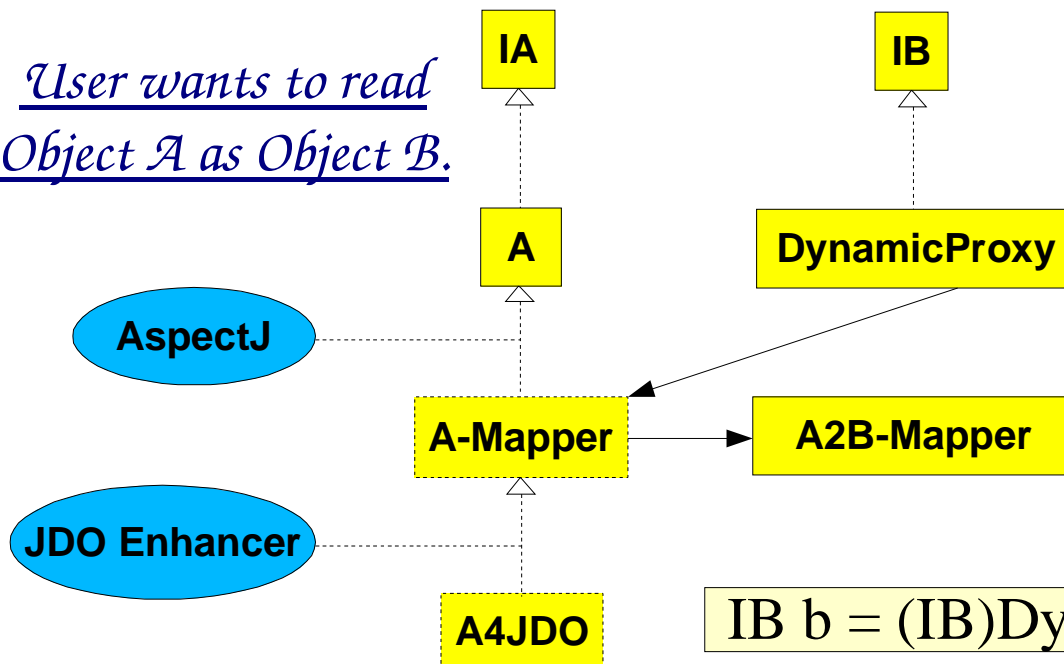
Object Evolution (1)

- Ability to change Object shape while keeping its content and identity.
- Two flavors:
 - Schema evolution (Versioning)
 - Object Mapping (DB Projection): Retrieving an Object of type *A* dressed as an Object of another type *B*.
- Not addressed by JDO.

Object Evolution (2)

- JDO Enhances class *A* so it is *PersistenceCapable*.
- AspectJ adds read callback with mapping *IA*->*IB*. It is called when JDO reads.
- *DynamicProxy* delivers content of *A* with interface of *IB*.
- DB of Mappers needed.
- Three concepts are used:

User wants to read
Object *A* as Object *B*.



- JDO enhancement
- Aspect extensions
- Dynamic Proxy

All this manipulation
is of course hidden.

```
IB b = (IB)DynamicProxy.newInstance(A, IB);
```

References (1)

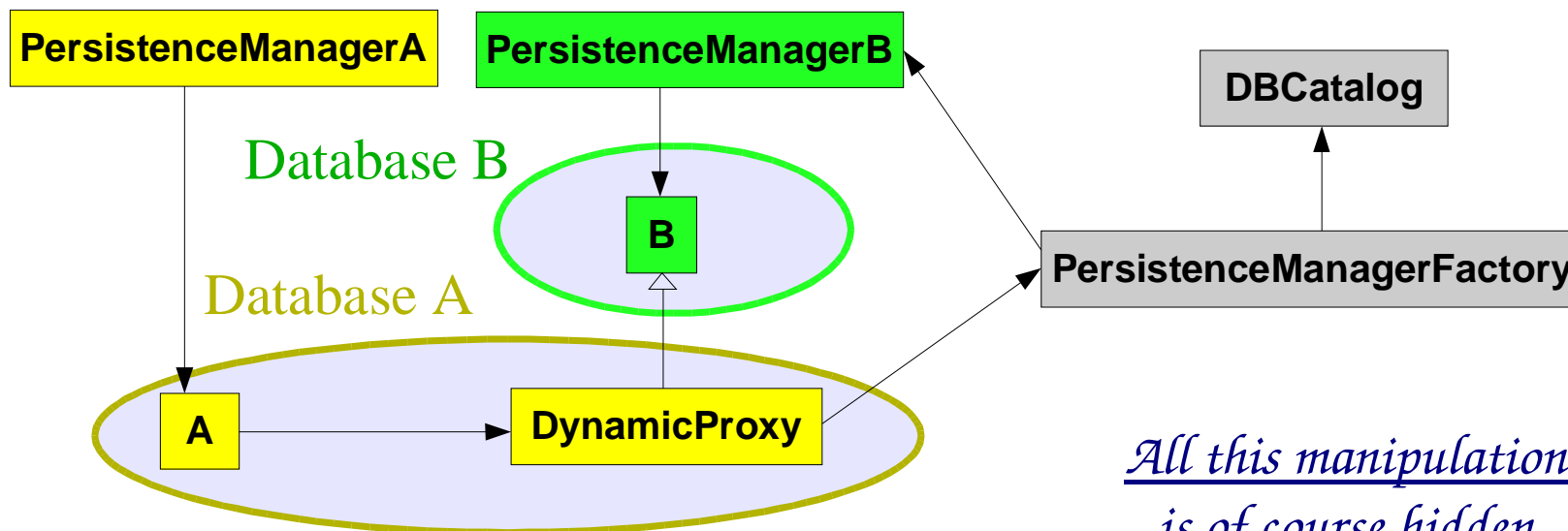
- Home references (within the same DB) automatically resolved by JDO (Persistence by Reachability).
- Foreign references (between different databases, possibly over the network) not resolved by JDO, but:
 - by underlying DB,
 - by Application Framework (EJB,...),
 - By PersistenceManagerFactory and Dynamic Proxy.

References (2)

- *DynamicProxy is stored when foreign reference is needed.*
- *When read, DynamicProxy calls its callback to:

 - *request PersistenceManager handling foreign Object,*
 - *receive that foreign Object,*
 - *cast itself into it.**

Object A references Object B,
which resides in different database.



All this manipulation
is of course hidden.

Summary

- *JDO standard provides suitable foundation of the persistence service for HEP applications.*
- *Two major characteristics of persistence solutions based on JDO are:*
 - *Not intrusiveness.*
 - *Wide range of available JDO implementation, both commercial and free, giving access to all major databases.*
- *JDO profits from the native databases functionality and performance (SQL queries,...), but presents it to users in a native Java API.*

Links

- JDO:
 - Standard: <http://java.sun.com/products/jdo>
 - Portal: <http://www.jdocentral.com>
 - TJDO: <http://tjdo.sourceforge.net>
 - More details talks:
 - <http://hrivnac.home.cern.ch/hrivnac/Activities/2002/June/JDO>
 - <http://hrivnac.home.cern.ch/hrivnac/Activities/2002/November/Indicium>
- Indicium: <http://hrivnac.home.cern.ch/hrivnac/Activities/Packages/Indicium>
- AIDA: <http://aida.freehep.org>
- Minerva: <http://hrivnac.home.cern.ch/hrivnac/Activities/Packages/Minerva>
- JACE: <http://sourceforge.net/projects/jace>
- Author: <http://hrivnac.home.cern.ch/hrivnac>