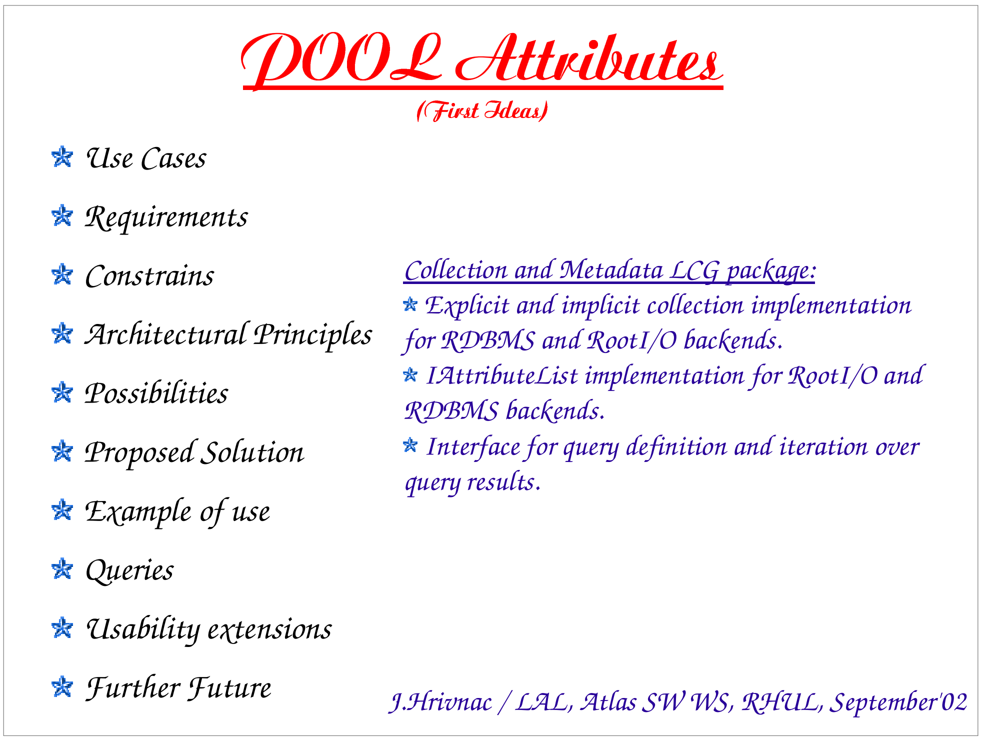


POOL Attributes

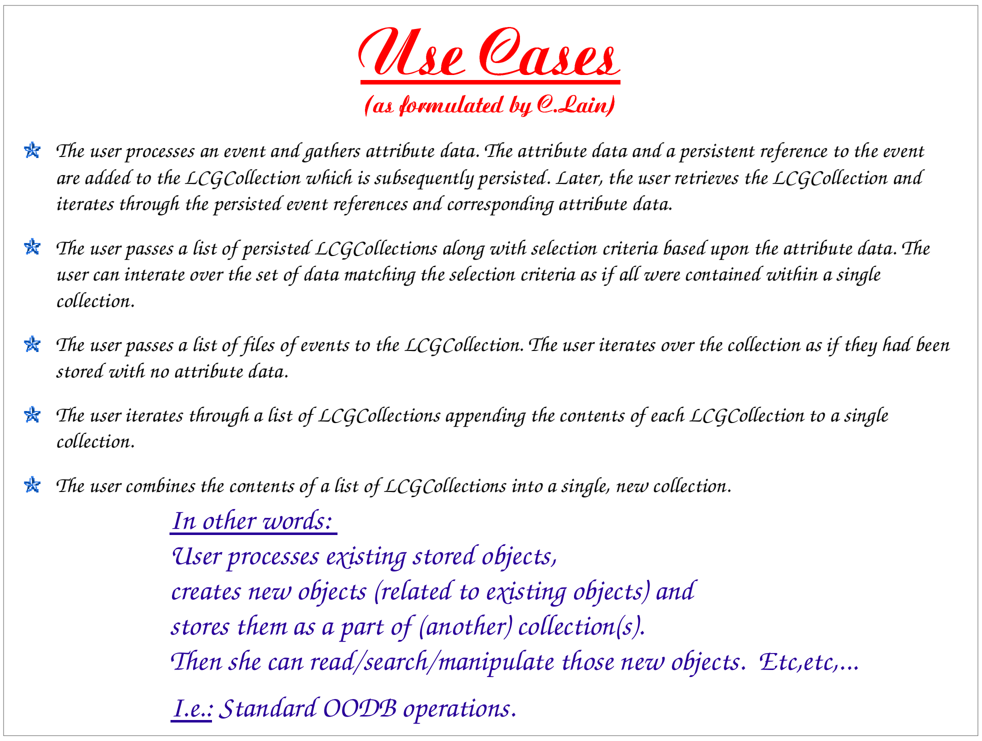
(First Ideas)

- Use Cases
- Requirements
- Constrains
- Architectural Principles
- Possibilities
- Proposed Solution
- Example of use
- Queries
- Usability extensions
- Further Future

Collection and Metadata LCG package:
- Explicit and implicit collection implementation for RDBMS and RootI/O backends.
- IAttributeList implementation for RootI/O and RDBMS backends.
- Interface for query definition and iteration over query results.

J.Hrivnac / LAL, Atlas SW WS, RHUL, September'02

- The specification doesn't define what Attributes are, it just requires them to use RootIO and RDBMS.
- Team:
  - David Malon (30%)
  - Chris Lain (50%)
  - Sasha Vaniashine (20%)
  - Julius Hrivnac (20%)



Use Cases

(as formulated by C.Lain)

- The user processes an event and gathers attribute data. The attribute data and a persistent reference to the event are added to the LCGCollection which is subsequently persisted. Later, the user retrieves the LCGCollection and iterates through the persisted event references and corresponding attribute data.
- The user passes a list of persisted LCGCollections along with selection criteria based upon the attribute data. The user can interate over the set of data matching the selection criteria as if all were contained within a single collection.
- The user passes a list of files of events to the LCGCollection. The user iterates over the collection as if they had been stored with no attribute data.
- The user iterates through a list of LCGCollections appending the contents of each LCGCollection to a single collection.
- The user combines the contents of a list of LCGCollections into a single, new collection.

In other words:
User processes existing stored objects,
creates new objects (related to existing objects) and
stores them as a part of (another) collection(s).
Then she can read/search/manipulate those new objects. Etc,etc,...

I.e.: Standard OODB operations.

It's not completely clear, which functionality should this project deliver on top of standard OODB functionality.

Requirements

(as formulated by J.Hrivnac)

- The main mission is to manipulate (create/define/add/remove/change/search/...) additional Attributes associated to already existing Objects in the DB (Events, Collection, whatever,...).
- Attributes are generally not declared per Object, but per Extent (Type).
- All Attributes have Name and Type.
- Attribute Instances have Value (or array of Values).
- Each Attribute can have DefaultValue and Comment.
- Each AttributeInstance can have Comment too.
- Each Value can be NULL.
- Attributes can have attached prescriptions (function-Classes) to create Values.
- Values can be also created by hand.
- Attributes and their AttributeInstances can't be stored with Objects they are attached (as Attributes are mutable) so they should know reference (OID) of their master Objects.

I.e.: Standard Object properties.

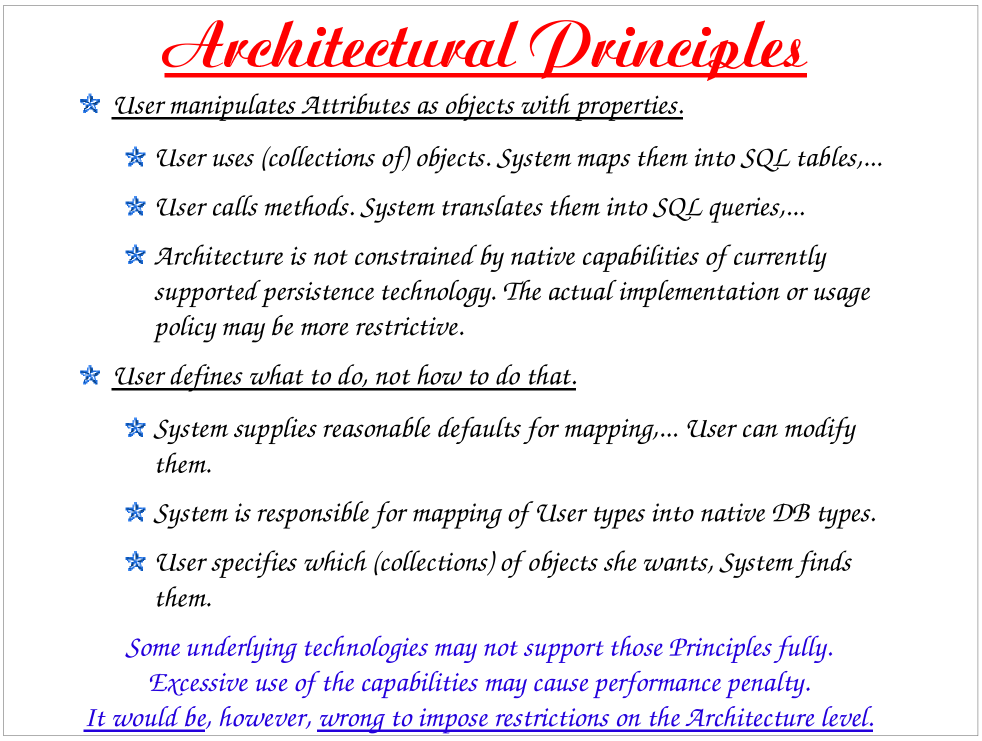
Constrains

(as formulated by LCG)

- Backends in RDBMS nad RootIO.
- Access from C++.

Other LCG components may introduce further constrains.

While requirements are well satisfied by an OODB, constrains force us to use non-OODB.

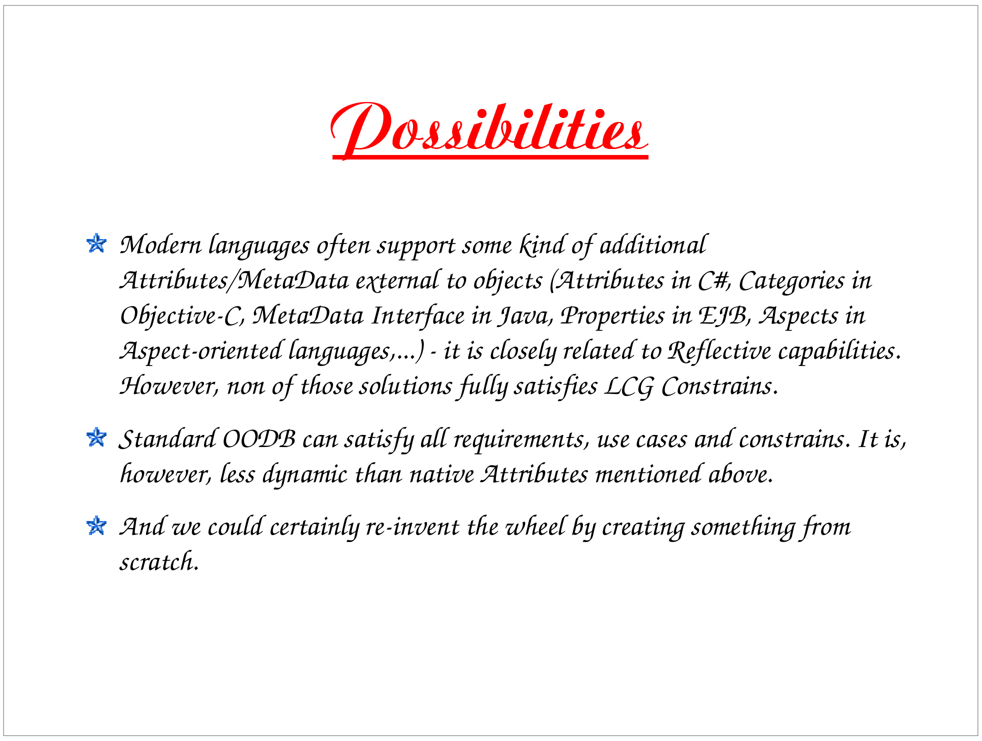


Architectural Principles

- User manipulates Attributes as objects with properties.
  - User uses (collections of) objects. System maps them into SQL tables,...
  - User calls methods. System translates them into SQL queries,...
  - Architecture is not constrained by native capabilities of currently supported persistence technology. The actual implementation or usage policy may be more restrictive.
- User defines what to do, not how to do that.
  - System supplies reasonable defaults for mapping,... User can modify them.
  - System is responsible for mapping of User types into native DB types.
  - User specifies which (collections) of objects she wants, System finds them.

Some underlying technologies may not support those Principles fully.
Excessive use of the capabilities may cause performance penalty.
It would be, however, wrong to impose restrictions on the Architecture level.

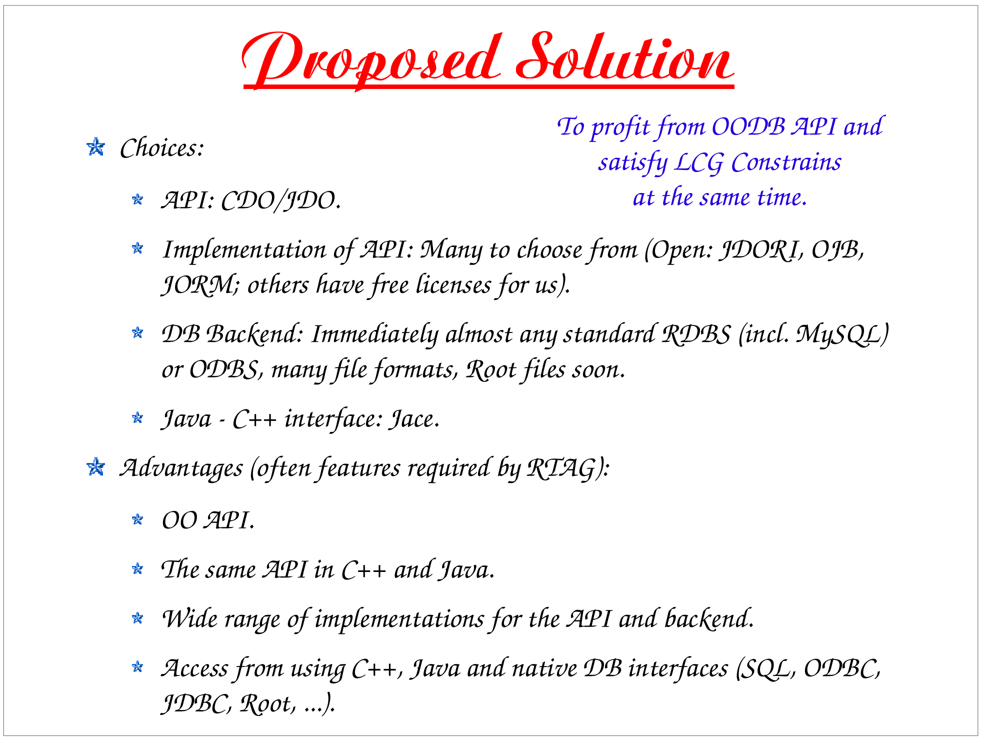
- These Principles support formulated Use Cases and Requirements.
- Similar to Transient-Persistent separation.
- Low-level access (directly to SQL) is available for performance studies,...
- Experiments may define a policy to restrict available capabilities.



Possibilities

- *Modern languages often support some kind of additional Attributes/MetaData external to objects (Attributes in C#, Categories in Objective-C, MetaData Interface in Java, Properties in EJB, Aspects in Aspect-oriented languages,...) - it is closely related to Reflective capabilities. However, non of those solutions fully satisfies LCG Constrains.*
- *Standard OODB can satisfy all requirements, use cases and constrains. It is, however, less dynamic than native Attributes mentioned above.*
- *And we could certainly re-invent the wheel by creating something from scratch.*

- 1st possibility would imply drastic shift in the current LCG policy.
- 3rd possibility would lead to substandard emulation of existing Attribute implememtation.

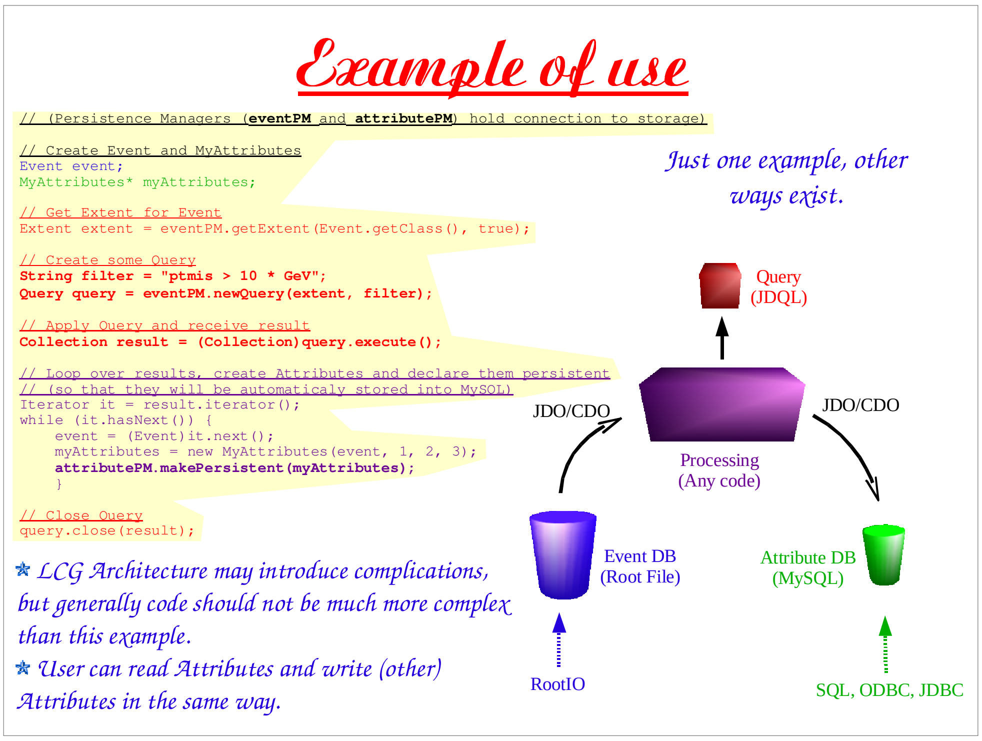


Proposed Solution

To profit from OODB API and satisfy LCG Constrains at the same time.

- *Choices:*
  - *API: CDO/JDO.*
  - *Implementation of API: Many to choose from (Open: JDORI, OJB, JORM; others have free licenses for us).*
  - *DB Backend: Immediately almost any standard RDBS (incl. MySQL) or ODBS, many file formats, Root files soon.*
  - *Java - C++ interface: Jace.*
- *Advantages (often features required by RTAG):*
  - *OO API.*
  - *The same API in C++ and Java.*
  - *Wide range of implementations for the API and backend.*
  - *Access from using C++, Java and native DB interfaces (SQL, ODBC, JDBC, Root, ...).*

- See “JDO for LCG” talk for details:
http://h.home.cern.ch/h/hrivnac/www/Activities/2002/June/JDO/
- CDO = C++ interface to (subset of) JDO.



Example of use

```
// (Persistence Managers (eventPM and attributePM) hold connection to storage)

// Create Event and MyAttributes
Event event;
MyAttributes* myAttributes;

// Get Extent for Event
Extent extent = eventPM.getExtent(Event.getClass(), true);

// Create some Query
String filter = "ptmis > 10 * GeV";
Query query = eventPM.newQuery(extent, filter);

// Apply Query and receive result
Collection result = (Collection)query.execute();

// Loop over results, create Attributes and declare them persistent
// (so that they will be automaticaly stored into MySQL)
Iterator it = result.iterator();
while (it.hasNext()) {
    event = (Event)it.next();
    myAttributes = new MyAttributes(event, 1, 2, 3);
    attributePM.makePersistent(myAttributes);
    }

// Close Query
query.close(result);
```

Just one example, other ways exist.

- *LCG Architecture may introduce complications, but generally code should not be much more complex than this example.*
- *User can read Attributes and write (other) Attributes in the same way.*

- Persistence Managers know how to find and access data using proper technology.
- Attributes are represented as Objects, internally they may be SQL tables,...
- One can read Attributes and write (possibly other) Attributes in the same way.
- This code can be either C++ or Java (after minor modifications).

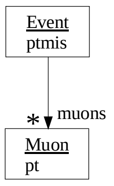
Queries

- Allows all standard mathematical operations on Attributes.
- Query = application of boolean filter on a Collection of candidate instances or an Extent. It returns all instances for which filter evaluates to true. Query is represented by a Query object.
- Internally translated to DB native query language (SQL, OQL,...) => efficient. Result of a query has the semantics of an SQL query. Query can be compiled and stored for speed.
- Native query languages still available (pm.newQuery("sql",...)).

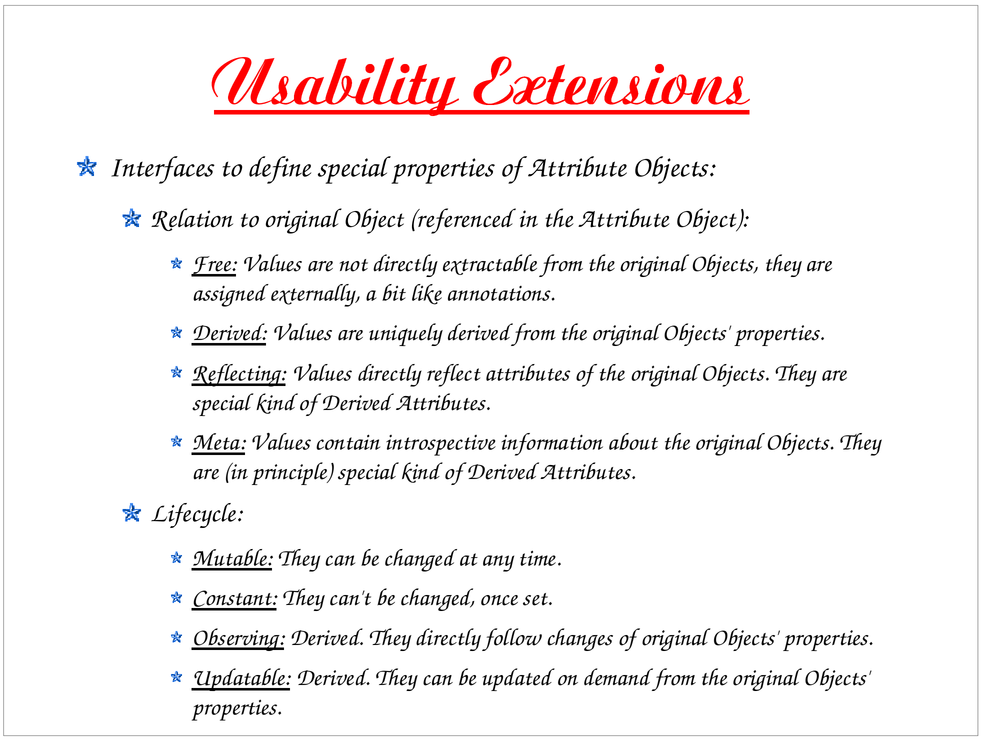
```
// Get PersistentManager
PersistenceManager pm = ...;

// Create Extent
Extent extent = pm.getExtent(Event.getClass(), true);
// Create Filter
String filter = "ptmis > ptmis_cut && " +
                "muons.contains(muon) && " +
                "muon.pt > mupt_cut";
// Create Query
Query query = persistenceManager.newQuery(extent, filter);
// Declare Variables and Parameters, set Ordering
query.declareVariables("Muon muon");
query.declareParameters("double ptmis_cut, double mupt_cut");
query.setOrdering("ptmis descending");

// Perform Query
Collection result = (Collection)query.execute(10.0, 5.0);
Iterator it = result.iterator();
...
```



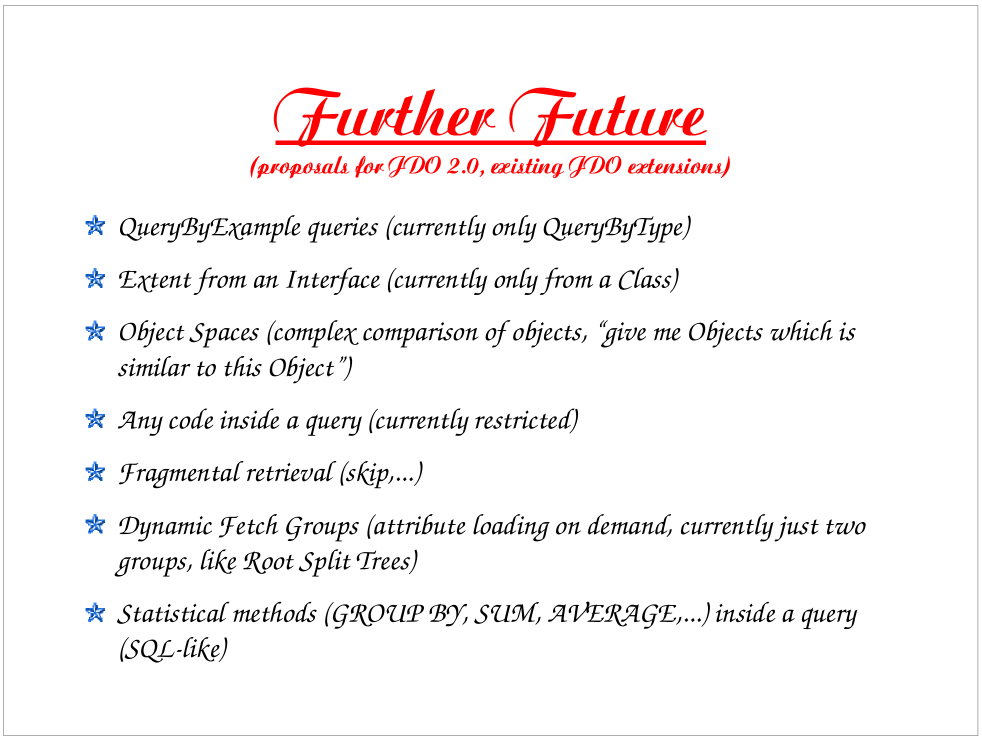
Find all Events with ptmis > 10.0 and pt of any muon > 5.0.



Usability Extensions

- Interfaces to define special properties of Attribute Objects:
  - Relation to original Object (referenced in the Attribute Object):
    - Free: Values are not directly extractable from the original Objects, they are assigned externally, a bit like annotations.
    - Derived: Values are uniquely derived from the original Objects' properties.
    - Reflecting: Values directly reflect attributes of the original Objects. They are special kind of Derived Attributes.
    - Meta: Values contain introspective information about the original Objects. They are (in principle) special kind of Derived Attributes.
  - Lifecycle:
    - Mutable: They can be changed at any time.
    - Constant: They can't be changed, once set.
    - Observing: Derived. They directly follow changes of original Objects' properties.
    - Updatable: Derived. They can be updated on demand from the original Objects' properties.

- Usefull to distinguish Attributes from ordinary Objects.
- Not essential, just interfaces to frequently used patterns to make life easier.



Further Future

(proposals for JDO 2.0, existing JDO extensions)

- QueryByExample queries (currently only QueryByType)
- Extent from an Interface (currently only from a Class)
- Object Spaces (complex comparison of objects, "give me Objects which is similar to this Object")
- Any code inside a query (currently restricted)
- Fragmental retrieval (skip,...)
- Dynamic Fetch Groups (attribute loading on demand, currently just two groups, like Root Split Trees)
- Statistical methods (GROUP BY, SUM, AVERAGE,...) inside a query (SQL-like)

- All features mentioned up-to-now are already supported by existing JDO packages.
- New features are under discussion.