JBoss Certified Technology Partner

Israel JBoss User Group

Session 10 / 11.12.2008

Hibernate Search in Action

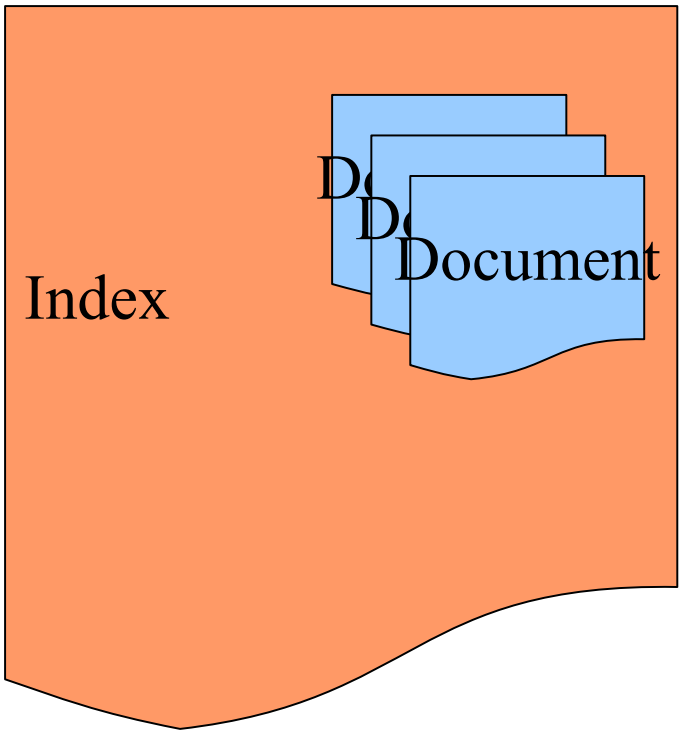
By : Yanai Franchi, Chief Architect , Tikal

Agenda

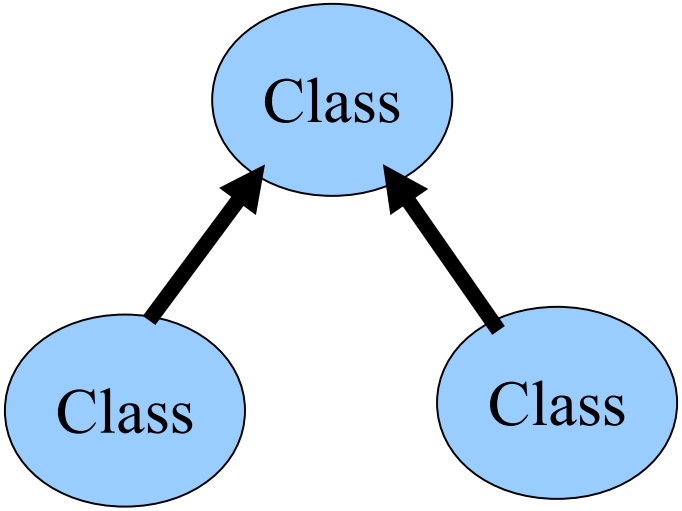
- The mismatch problems
- Hibernate Search in a nutshell
- Mapping – Solving the structural mismatch
- Indexing – Solving the synchronization mismatch
- Querying – Solving the retrieval mismatch
- Demo
- Scale Hibernate Search

The Mismatch Problems

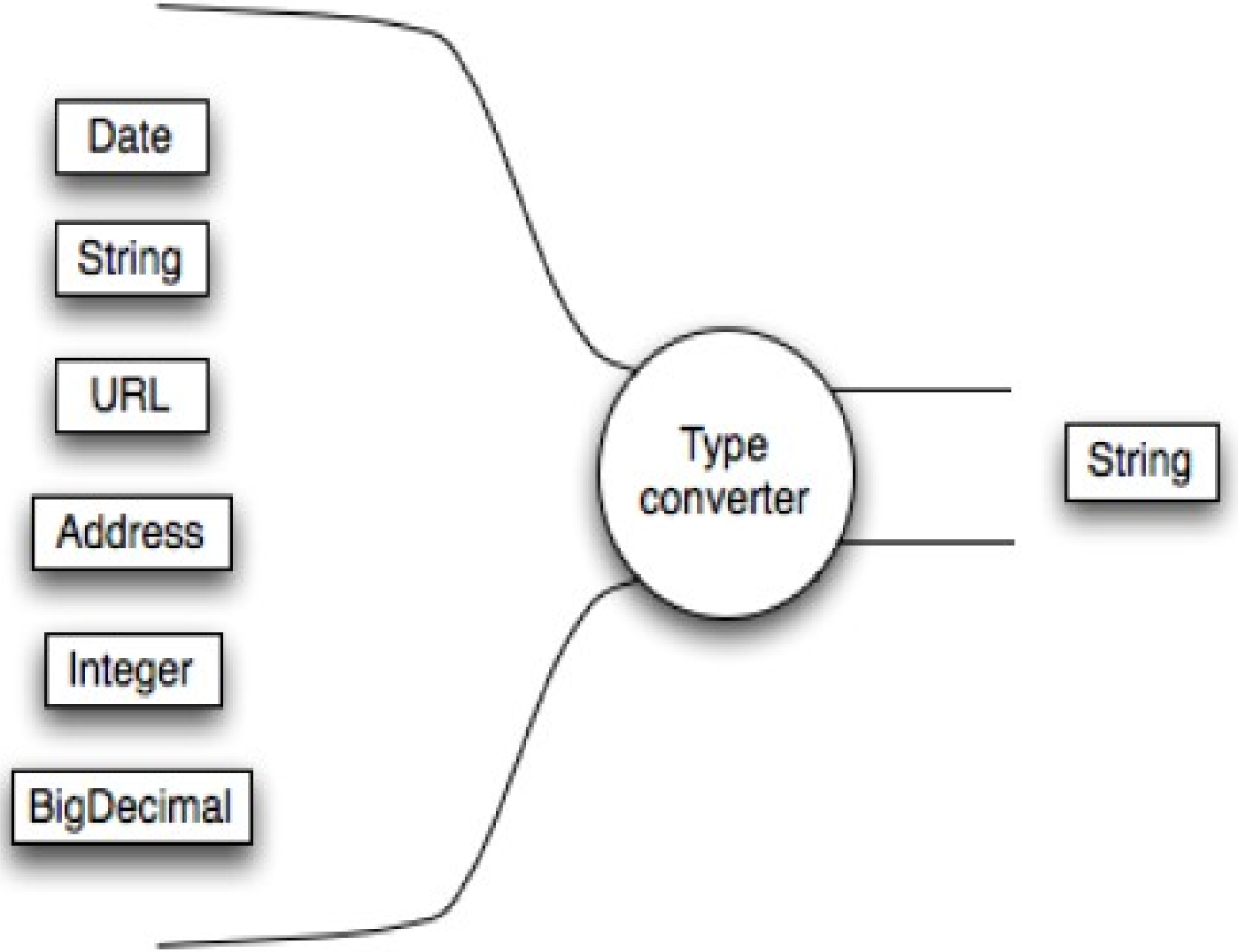
Impedance Mismatch Between Object And Index Models



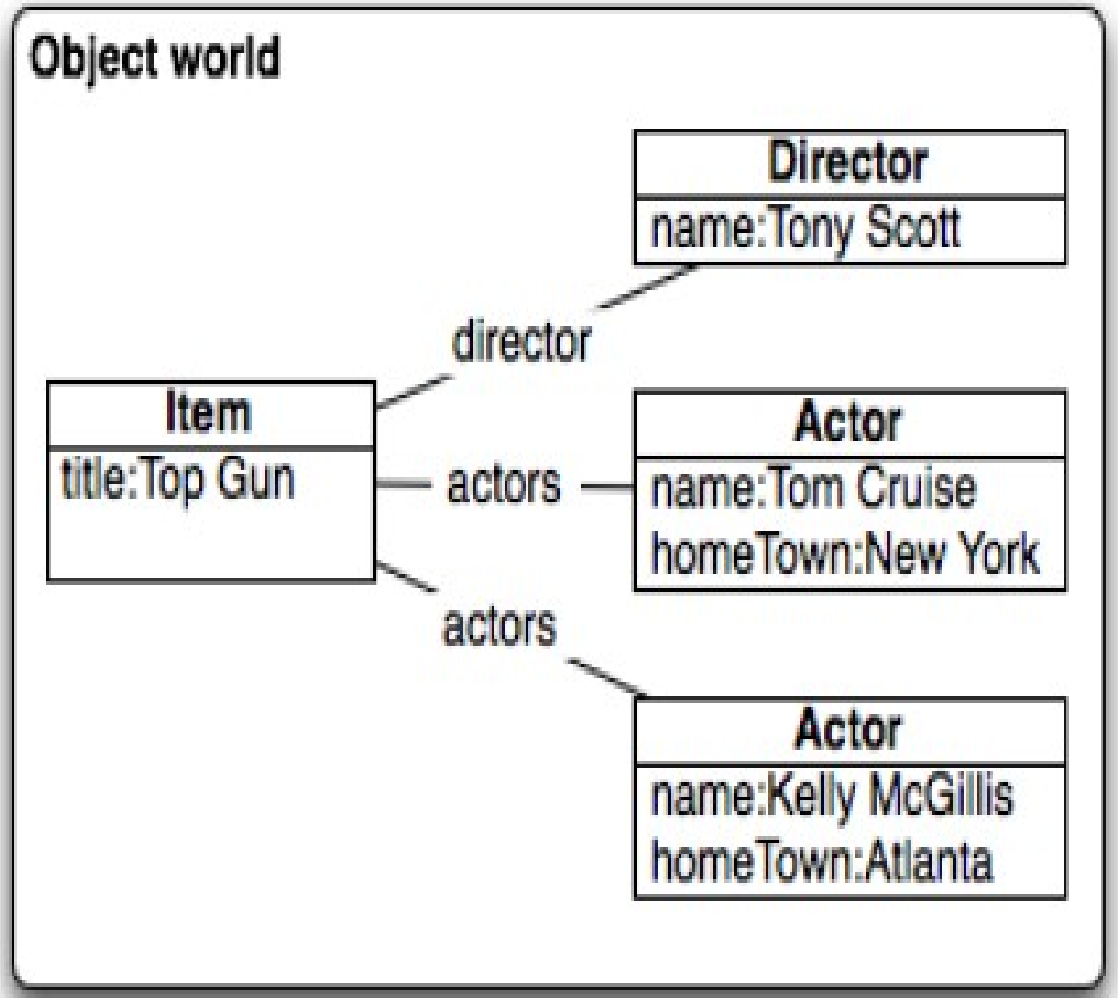
!=

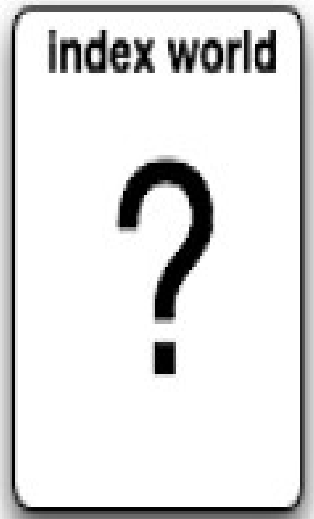


Mismatch With Types

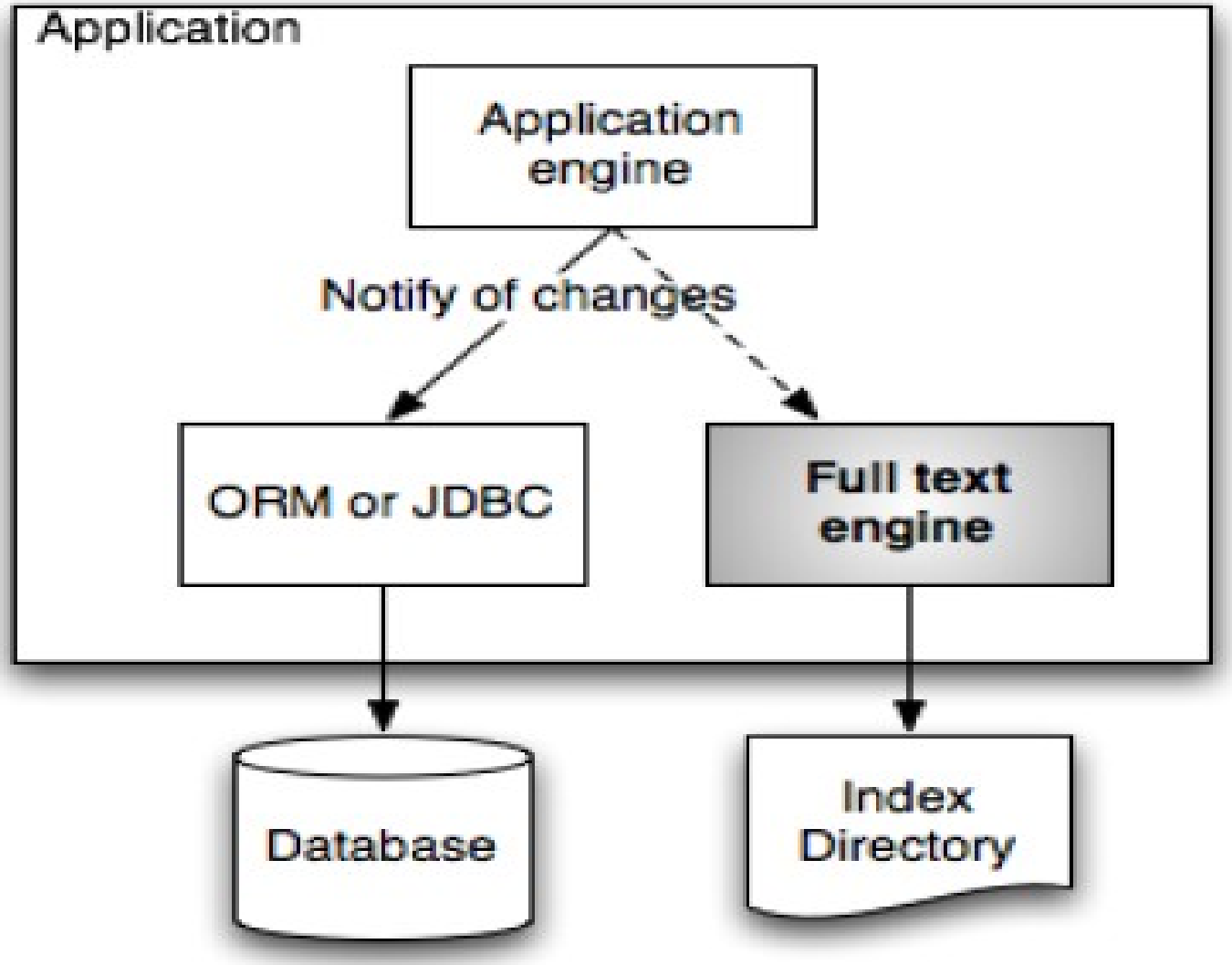


Mismatch With Associations





Synchronization Mismatch



Retrieval Mismatch

- NO Conversation – You don't want to go there...
  - » Loose domain driven, and OO paradigm
  - » No type safety and strong type
- Conversion
  - » “rehydrate” Document from field values stored in index.
    - No lazy loading and transparent access
    - No automatic synchronous against the DB (and index)
  - » Retrieve Hibernate managed objects.
    - Loading one-by-one is NOT efficient...

Hibernate Search in a Nutshell

Hibernate Search Goal

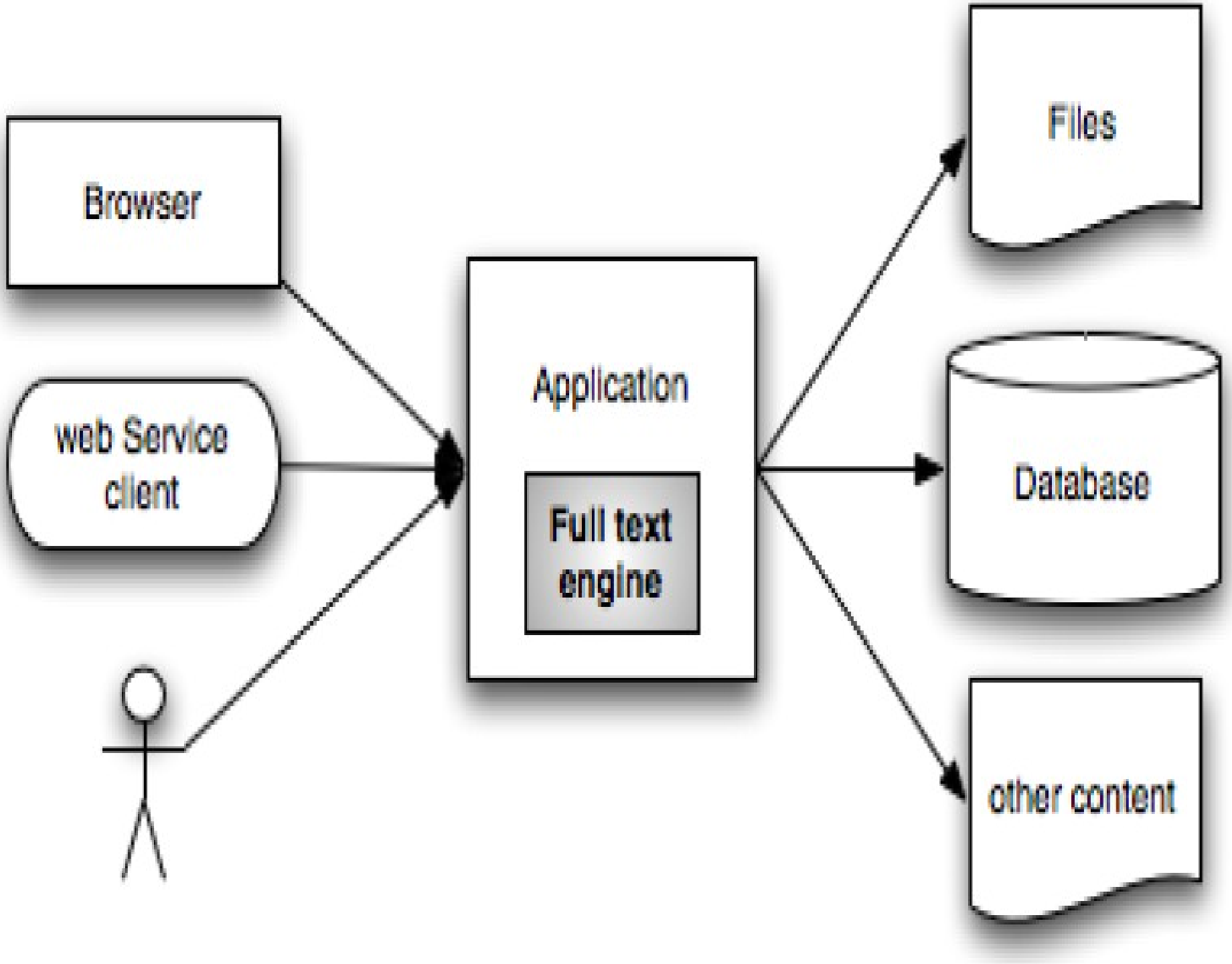
Leverage Hibernate (ORM)

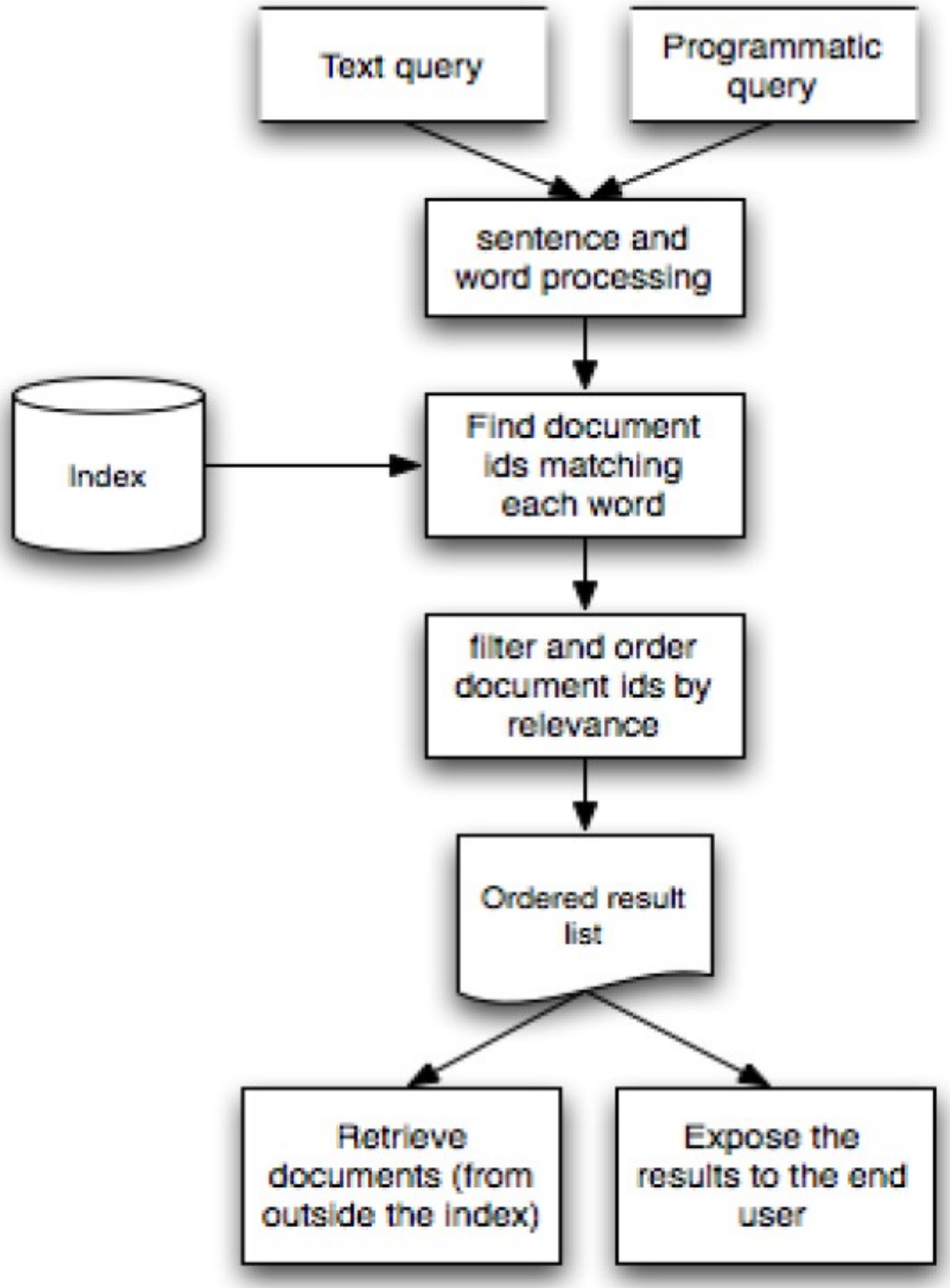
and Apache Lucene (full-text search engine),

while address the mismatch problems.

Hibernate Search Features

- Under the Hibernate platform
  - » LGPL
- Built on top of Hibernate Core
- Use Apache Lucene(tm) under the hood
  - » Hides the low level and complex Lucene API usage
- Solve the mismatches


Browser
web Service client
Application
Full text engine
Files
Database
other content


Text query
Programmatic query
sentence and word processing
Index
Find document ids matching each word
filter and order document ids by relevance
Ordered result list
Retrieve documents (from outside the index)
Expose the results to the end user

Project Set-up

- Set your classpath
  - » hibernate-search.jar: the core API and engine of Hibernate Search
  - » lucene-core.jar: Apache Lucene engine
  - » hibernate-commons-annotations.jar: some common utilities for the Hibernate project

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-search</artifactId>
    <version>3.1.0.GA</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-annotations</artifactId>
    <version>3.4.0.GA</version>
</dependency>
```

(Optional) Project Configuration

- Configure hibernate search
  - » No need for event listeners.
    - When using JPA/Hibernate Annotations
- hibernate-cfg.xml or META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?> META-INF/persistence.xml
<persistence>
  <persistence-unit name="dvdstore-catalog">
    <jta-data-source>jdbc/test</jta-data-source>
      <properties>
        ...
        <property
           name="hibernate.search.default.indexBase"
           value="/users/application/indexes"/>
        ..
      </properties>
  </persistence-unit>
</persistence>
```

Map Your Domain Model

```
@Entity
@Indexed
public class Book {
    @Id // → Automatically mapped to @DocumentId
    private Integer id;

    @Field
    private String title;

    ...
}
```

How Is The Index Look Like?

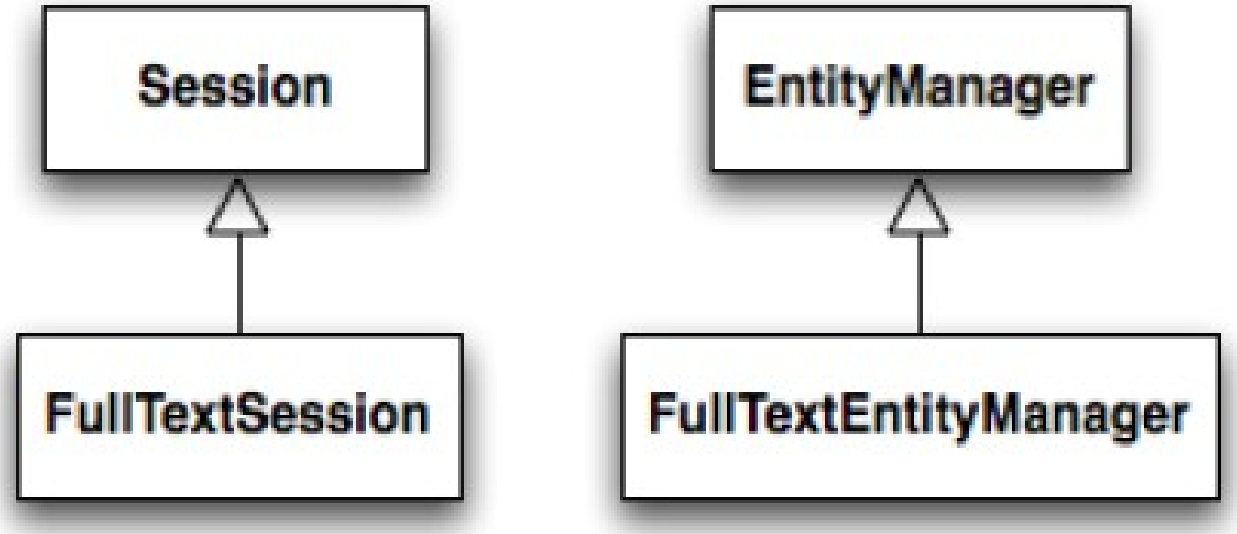
Book table

id	title
1	The Da Vinci Code
2	Alice's adventures in Wonderland
...	...
234	Economics for Dummies
235	The real history behind the Da Vinci Code

Full text index

Word	id	freq.	position
adventures	2	1/3	2
alice	2	1/3	1
code	1	1/3	3
	235	1/6	5
da	1	1/3	1
	235	1/6	4
dummies	234	1/2	2
economics	234	1/2	1
vinci	1	1/3	2
	235	1/6	5
wonderland	2	1/3	3

Hibernate Search Managers




```
Session session = sessionFactory.getCurrentSession();

FullTextSession fts =
          org.hibernate.search.Search.getFullTextSession(session);
```

```
@PersisntenceContext EntityManager em;
...

FullTextEntityManager ftem =
      org.hibernate.search.jpa.Search.getFullTextEntityManager(em);
```

Query in Action

```
String searchStr = "title:hypernate~ OR description:persistence";

org.apache.lucene.search.Query luceneQuery =
                              buildLuceneQuery(searchStr);

javax.persistence.Query jpaQuery =
  ftEm.createFullTextQuery(luceneQuery,Book.class, Course.class);


List booksAndCourses = query.getResultList();


applySomeChanges(booksAndCourses);
```

Can accept more than one class

Books and Courses get into JPA ***"persistent context"*** and changes will be automatically applied to ***DB*** (and the ***Lucene Index)***

Mapping – Solve The Structural Mismatch

Object world

Item
id:Long title:String description:String price:double releaseDate:Date

DVD
ean:String zone:Zone

Additional types

- Long
- String
- double
- Date
- Zone <<Enum>>

Conversion to (and sometimes from)

Lucene index world

Item document
id field:String title field:String description field:String releaseDate field:String

DVD document
id field:String title field:String description field:String releaseDate field:String ean:String zone:String

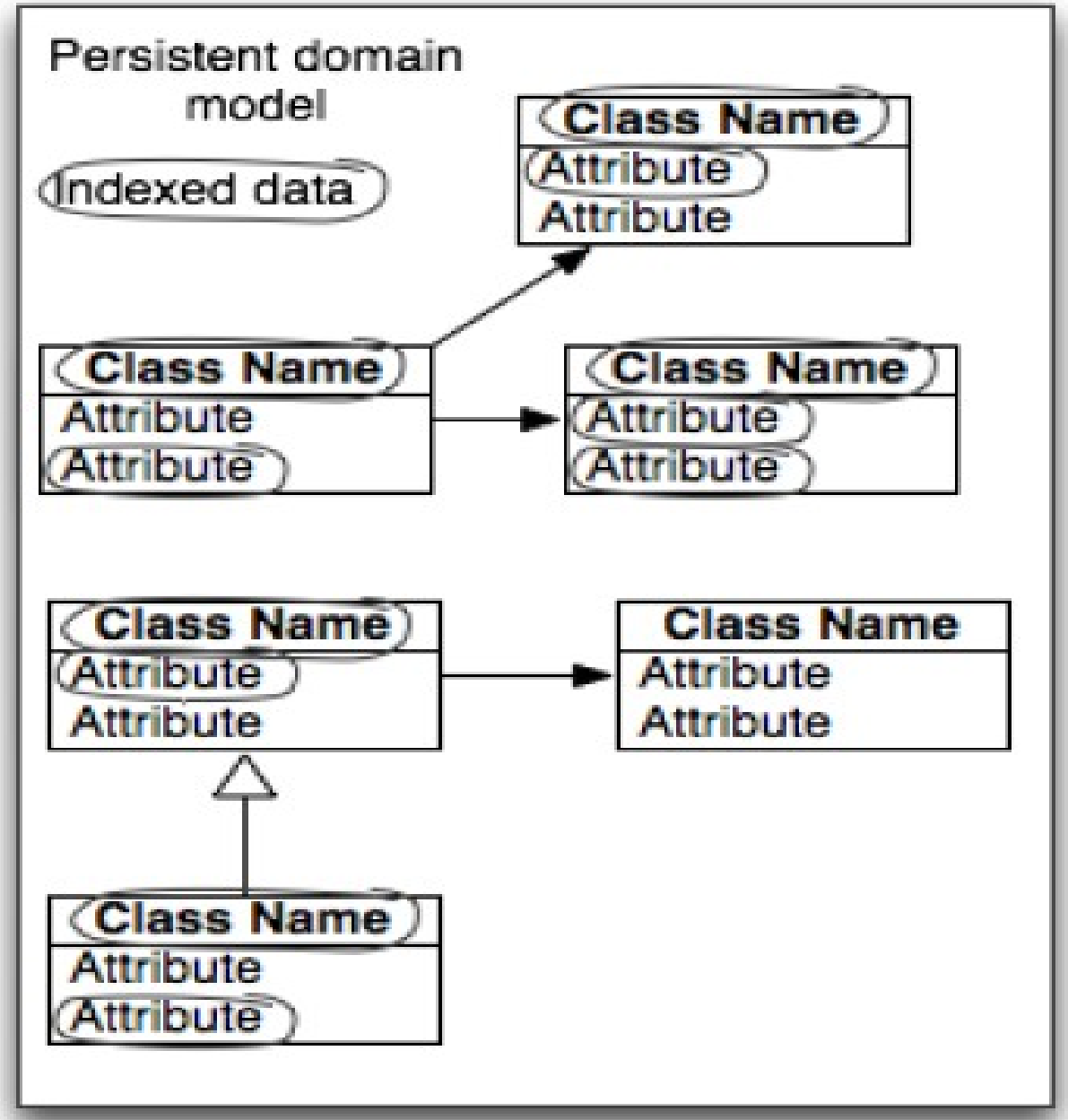
Additional types

- String

Mapping Entity & Primary Key

```
@Entity
@Indexed
public class Item {
    @Id // → Automatically mapped to @DocumentId
    @GeneratedValue
    private Integer id;
    ...
}
```

Marking Properties As Indexed



Mapping Properties

```
@Entity
@Indexed
public class Item {
    @Id @GeneratedValue
    private Integer id;
    @Field(index=Index.UN_TOKENIZED)
    private String ean;

    @Field(store=Store.YES)
    private String title;

    //Will not be indexed while still being stored into DB
    private String imageURL;

    private String description;
    ...
    @Field //Annotation on the getter
    public String getDescription() {
        return this.description;
    }
}
```

Multiple Indexed Property

```
@Entity
@Indexed
public class Item {
    ...
  @Fields({
    @Field(index=Index.TOKENIZED)
    @Field(name="title_sort", index=Index.UN_TOKENIZED)
  })
private String title;
```

- Properties that will be used to sort query results (rather than by relevance) must not be tokenized but must be indexed.
  - » Use UN_TOKENIZED indexing strategy

Mapping Inheritance

```
@Entity //Superclasses do not have to be marked @Indexed
public abstract class Item {
    @Id // used as @DocumentId
    @GeneratedValue
    private Integer id

    @Field //Superclasses can contain indexed properties
    private String title;
    ...
}
```

```
@Entity
@Indexed //Concrete subclasses are marked @Indexed
public class Dvd extends Item {

    @Field(index=Index.UN_TOKENIZED)
    private String ean;
    ...
}
```

Built-In Bridges

- Bridges convert a Java object type into a string.
- Some field bridges also convert back the string into the original object structure
  - » Identity and projected fields
- Hibernate Search comes with many out-of-the-box field bridges. But you can write (or reuse) you own...
  - » PDF, Microsoft-Word and other document types
  - » Index Year, Month, Day on separate fields
  - » Make numbers comparerable

Bridge Issues

- Dates
  - » [20080112 TO 20080201] – field is between 12 January 2008 and 1 February 2008.
  - » Hibernate Search lets you pick the date precision you wish from year to millisecond:

```
@DateBridge( resolution = Resolution.DAY )
private Date birthdate;

@DateBridge( resolution = Resolution.MINUTE )
private Date flightArrival;
```

- Numbers
  - » “2 > “12”
  - » [6 TO 9] => 6 OR 7 OR 8 OR 9

Custom Bridge in Action

```
@Entity @Indexed
public class Item {
  @Field
  @FieldBridge(
    impl=PaddedRoundedPriceBridge.class, // So 2 becomes "002"
    params= { @Parameter(name="pad", value="3") }
  )
  private double price;
  ...
}
```

- Mapping a property to split the information to multiple fields in the index.

ClassBridge

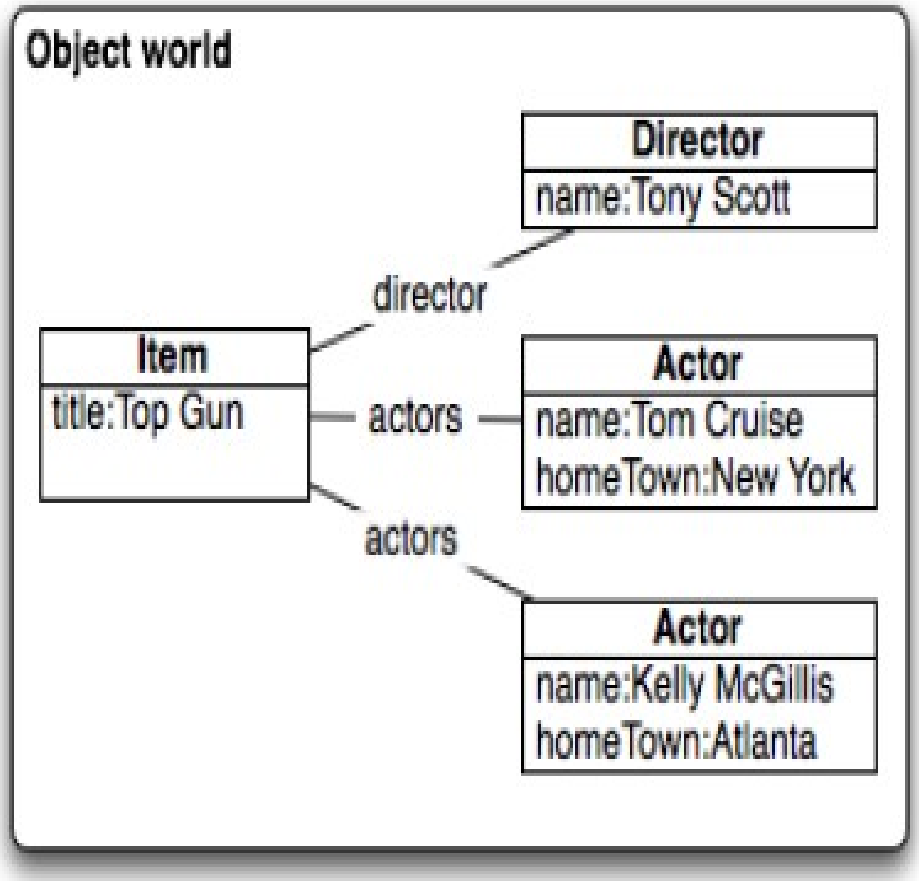
```
@Entity @Indexed
@ClassBridge(name="childrenOnly",
             impl=ChildrenFlagBridge.class,index=Index.UN_TOKENIZED)
public class Item {
...
}
```

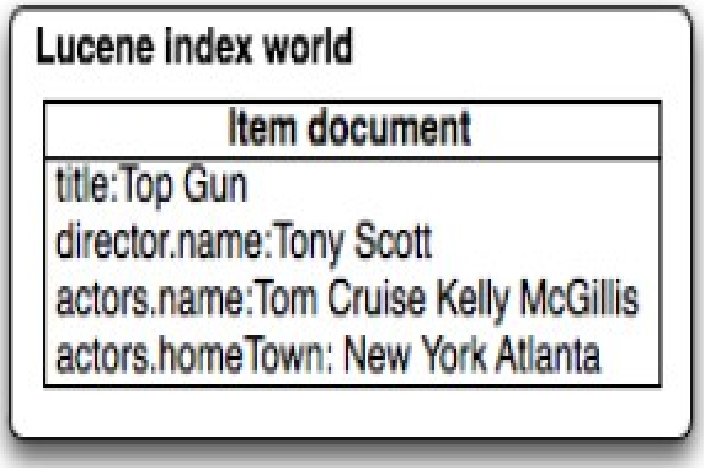
```
public class ChildrenFlagBridge implements StringBridge {
  public String objectToString(Object object) {
    Item item = (Item) object;
    Category childrenCategory = new Category("Children");

    boolean hasChildrenCategory  =
                    item.getCategories().contains(childrenCategory);

    return hasChildrenCategory ? "yes" : "no";
}
```

How to Map Associations ?

Entity	Relation	Entity	Query
Class Name Association Attribute	1 — *	**Class Name** Attribute Attribute	Navigation and filter
Table Foreign key column column		**Association table** Foreign key column Foreign key column	**Table** Column Column	SQL join and where clause
Lucene Document field field	?	**Lucene Document** field field	?

De-Normalize Associations





De-Normalization Implication

- Can return items that :
  - » One of the actor is "**Cruise**" and another one is "**McGillis**"
  - » One of the actor is either "**Cruise**" or "**McGillis**"
  - » **"Cruise"** plays but not "**McGillis**"
- Can **NOT** do:
  - » Return items where one of the actor is "**Tom**" and his home town is "**Atlanta**".
    - Turn the query upside down by targeting actor as the root entity and then collect the matching items
    - Use a query filter to refine an initial query
- Sometime you may end up in a dead end...
  - » Apply part of the query (the discriminant part) in Lucene,
  - » Collect the matching identifiers
  - » Run a HQL query restricting by these identifiers.

Indexing Embeddables

```
@Embeddable
public class Rating {
   @Field(index=Index.UN_TOKENIZED) private Integer overall;
   @Field(index=Index.UN_TOKENIZED) private Integer scenario;
   @Field(index=Index.UN_TOKENIZED) private Integer soundtrack;
   @Field(index=Index.UN_TOKENIZED) private Integer picture;
   ...
}
```

```
@Entity @Indexed
public class Item {
   @IndexedEmbedded private Rating rating;
...
}
```

"find items with overall rating equals to 9"

rating.overall : 9

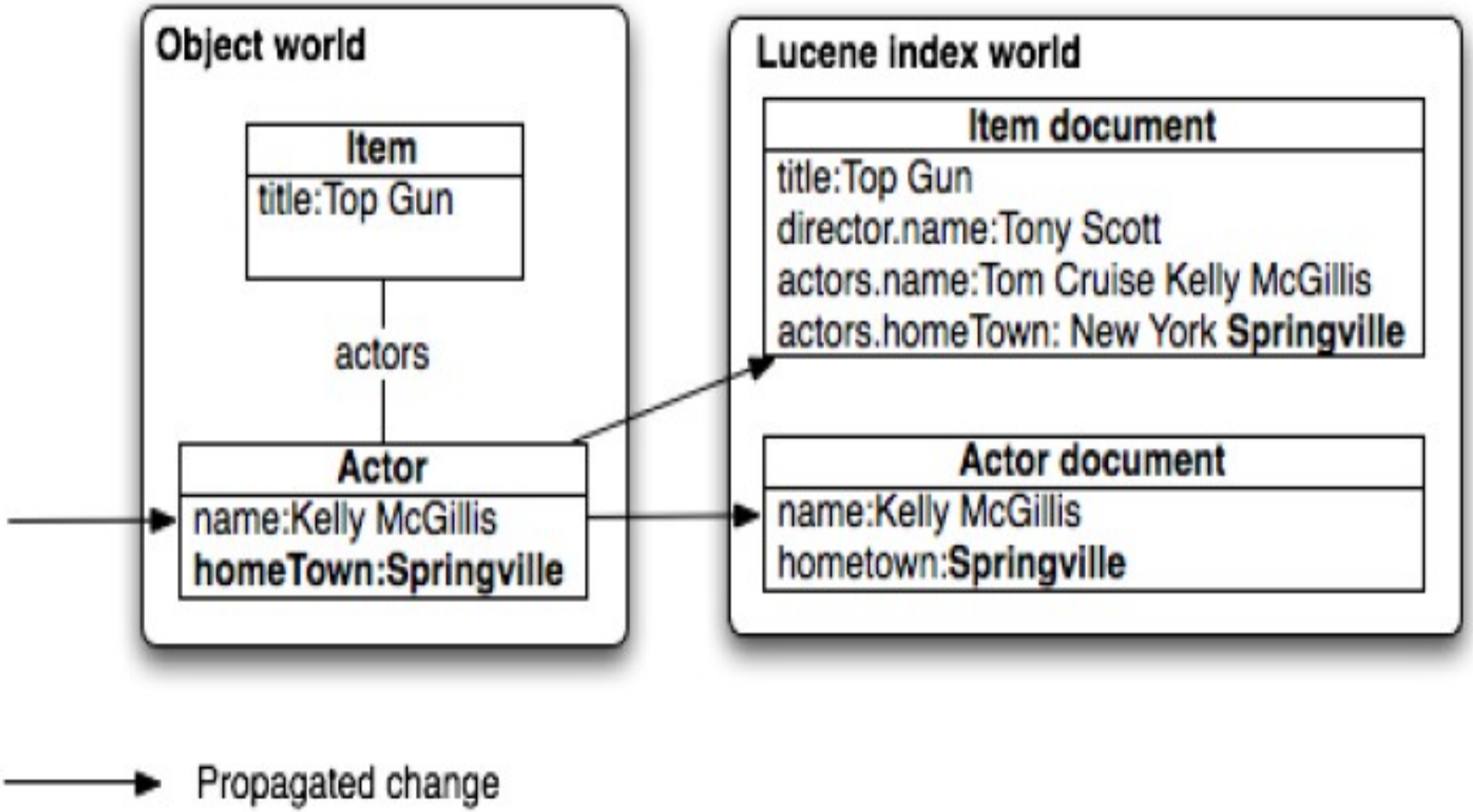
...And Embeddables Collection

```
@Embeddable
public class Country {
   @Field private String name;
   ...
}
```

```
@Entity @Indexed
public class Item {
   @CollectionOfElements @IndexedEmbedded
   private Collection<Country> distributedIn;
...
}
```

- Don't abuse IndexedEmbedded. Be careful on collection indexing...

Indexing Associated Entities



When a change is done on an associated entity, Hibernate Search must update all the documents where the entity is embedded in

Indexing Associated Entities

```
@Entity @Indexed
public class Item {
  @ManyToMany
  @IndexedEmbedded
  private Set<Actor> actors; //embed actors when indexing
  ...
}
```

```
@Entity @Indexed
public class Actor {
  @Field private String name;

  @ManyToMany(mappedBy="actors")
  @ContainedIn // We may use (depth=4) to limit depth
  private Set<Item> items;
  ...
}
```

- Relations between entities become bi-directional in case the Actor is not immutable, or do manual index

Indexing Your Data - Solve The Synchronization Mismatch

Defining a DirectoryProvider

```
# Production configuration
hibernate.search.default.directory_provider
                    org.hibernate.search.store.FSDirectoryProvider
hibernate.search.default.indexBase /User/production/indexes
```

```
# File directory structure
/Users
  /Production
    /indexes
      /com.manning.hsia.dvdstore.model.Item
      /com.manning.hsia.dvdstore.model.Actor
```

```
# Test Configuration
hibernate.search.default.directory_provider
                    org.hibernate.search.store.RAMDirectoryProvider
```

Analyzers – Lucene Brain

- The key feature of the full text search
- Taking text as an input, chunking it into individual words (by a tokenizer) and optionally applying some operations (by filters) on the tokens.
- Applied: globally, per entity, or per property

Tokenizers & Filters

- StandardTokenizer –Splits words at punctuation characters and removing punctuation signs with a couple of exception rules.

- Filters alter the stem of tokens (remove/change/add)
  - » StandardFilter – Removes apostrophes and acronyms dots
  - » LowerCaseFilter
  - » StopFilter – Eliminates “noise” words.

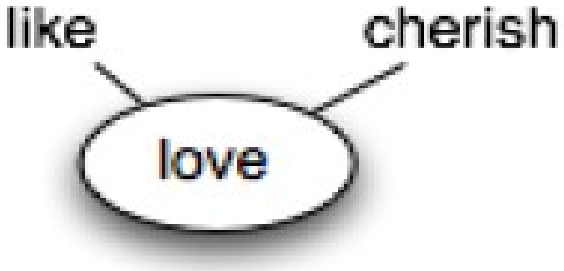
StandardAnalyzer in Action

```
@AnalyzerDef( //This is the default → no need to write it
  name="applicationAnalyzer",
  tokenizer =@TokenizerDef(factory=StandardTokenizerFactory.class),
  filters = {
    @TokenFilterDef(factory = StandardFilterFactory.class),
    @TokenFilterDef(factory = LowerCaseFilterFactory.class),
    @TokenFilterDef(factory = StopFilterFactory.class)
  }
)
```

```
@Entity @Indexed
@Analyzer(definition="applicatioAanalyzer")
public class Item {
...
}
```

More Available Filters

Synonym



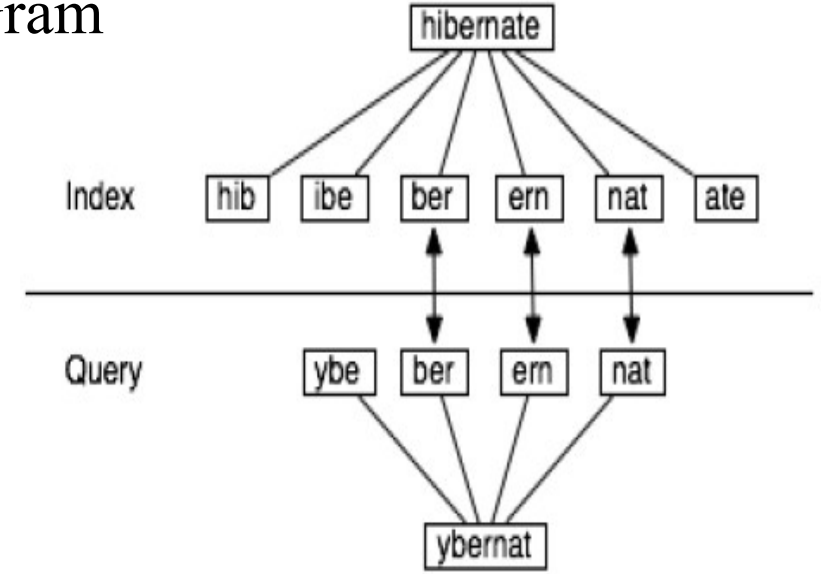
Stem

Rule			Example		
SSES	->	SS	Caresses	->	Caress
IES	->	I	Ponies	->	Poni
SS	->	SS	Caress	->	Caress
S	->		Cats	->	Cat

Phonetic

Les Misérable => LS MSRP

N-Gram



N-Gram Analyzer Example

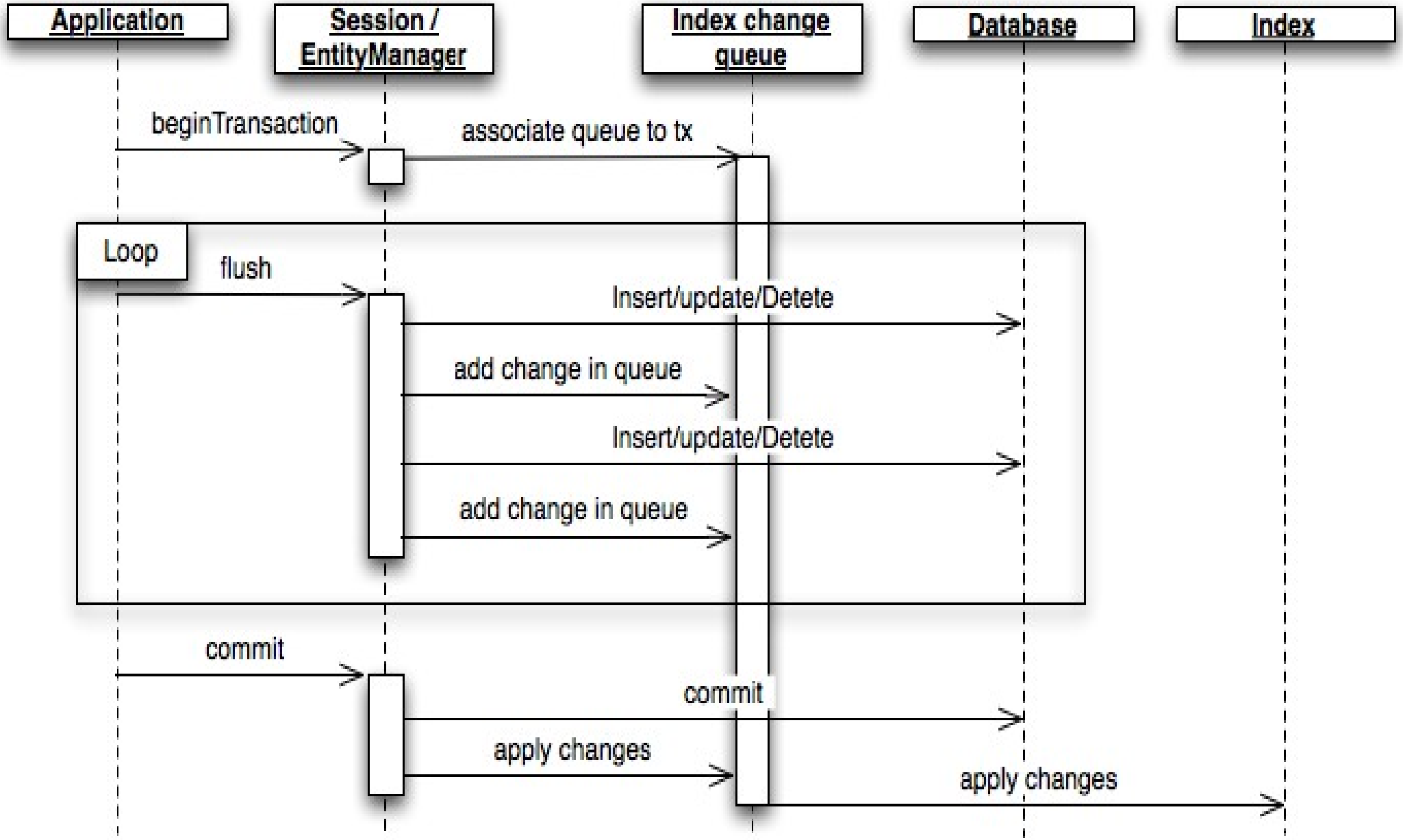
```
@AnalyzerDef(
  name="ngramAnalyzer",
  tokenizer =@TokenizerDef(factory=StandardTokenizerFactory.class),
  filters = {
    //Standard, LowerCase and Stop filters goes here
    @TokenFilterDef(factory = NGramTokenFilterFactory.class,
      params = {
        @Parameter(name="minGramSize", value="3"),
        @Parameter(name="maxGramSize.", value="3")
     })
  }
)
```

```
@Entity @Indexed
// The default StandardAnalyzer will be used
public class Item{
  @Fields({
    @Field(index=Index.TOKENIZED),
    @Field(name="title_ngram",index=Index.UN_TOKENIZED,
      analyzer=@Analyzer(definition="ngramAnalyzer")
  })
  private String title;
}
```

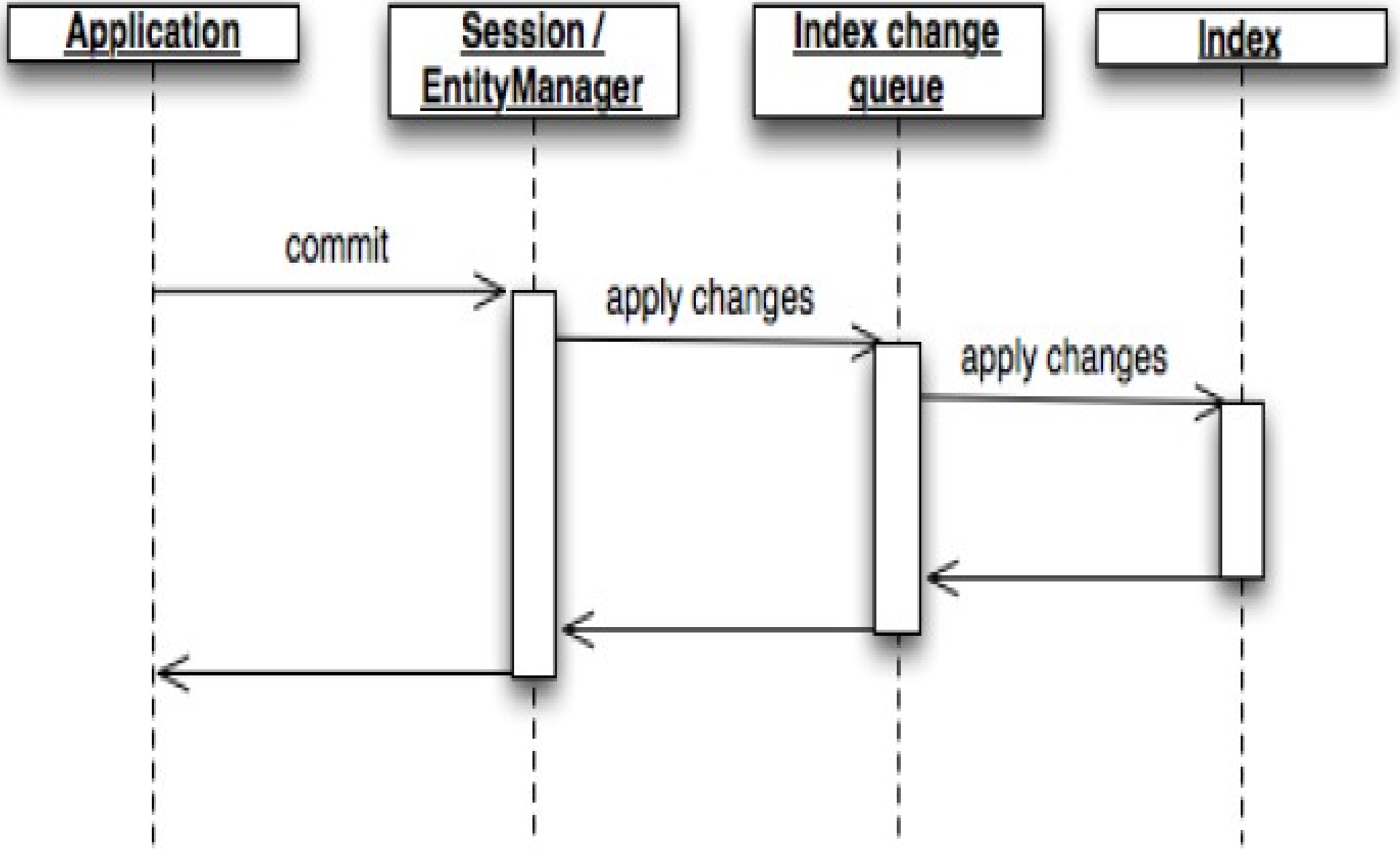
Which Technique to Choose?

- Use approximation analyzers on dedicated fields.
- Search in layers – Expand the approximation level.
  - The search engine can execute the strict query first
  - If more data is required a second query using approximation techniques can be used and so on.
  - Once the search engine has retrieved enough information, it bypasses the next layers.
- Remember that a Lucene query is quite cheap. Running several Lucene queries per user query is perfectly acceptable.

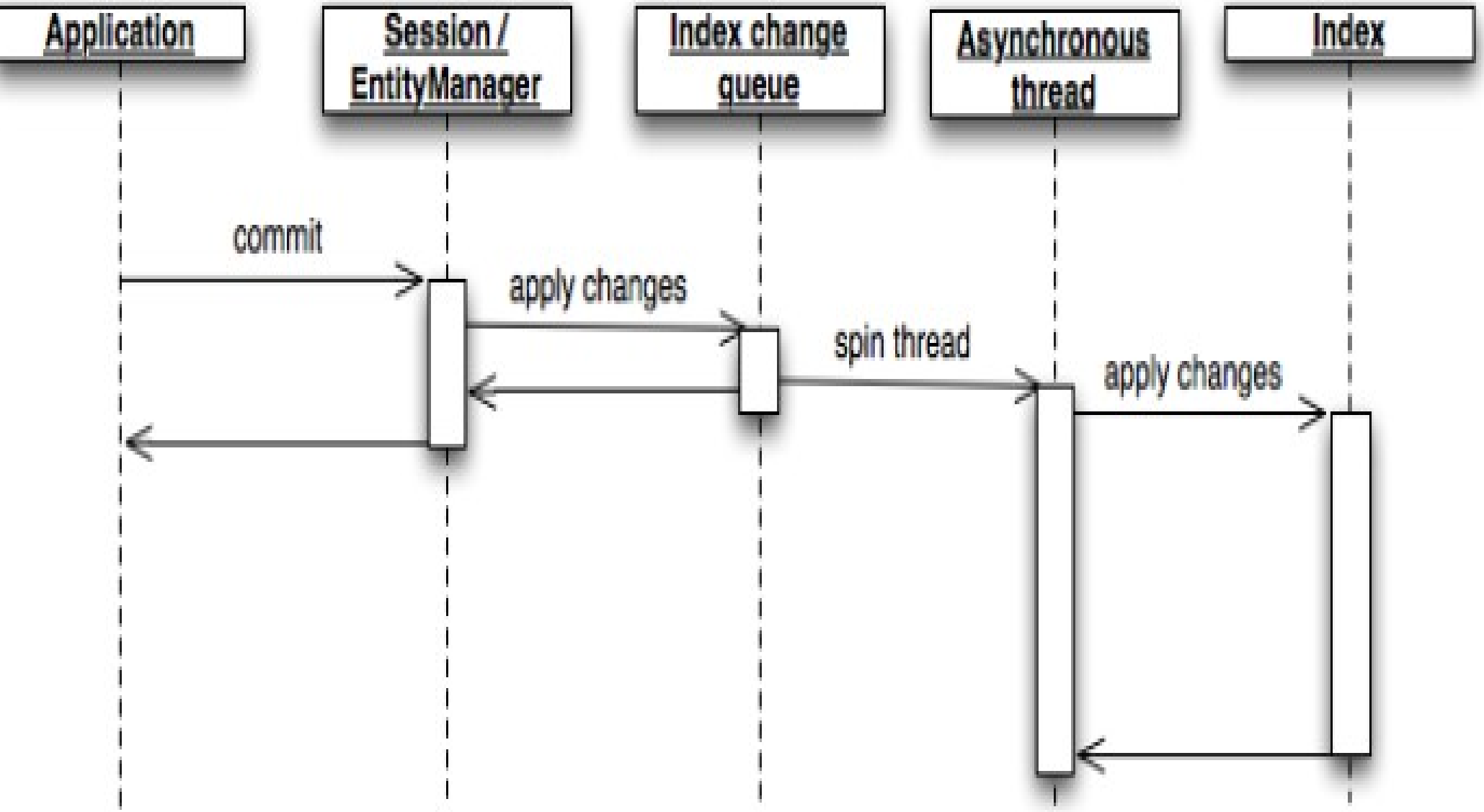
Indexing Flow Diagram



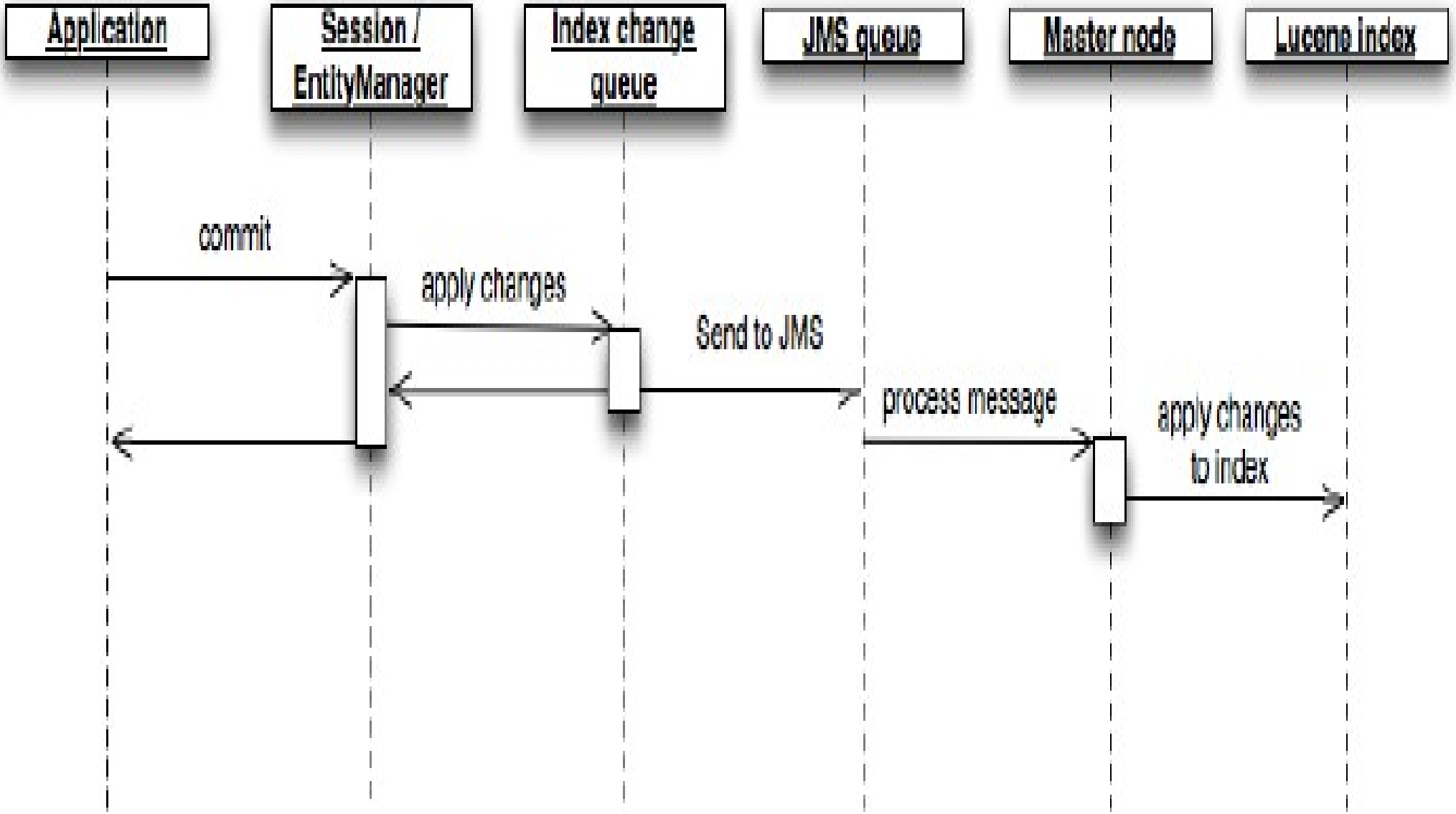
Synchronous Flow



Asynchronous Flow



JMS Flow


Application
Session / EntityManager
Index change queue
JMS queue
Master node
Lucene index
commit
apply changes
Send to JMS
process message
apply changes to index

Manual Index – Naïve Approach

```
Transaction tx = session.beginTransaction();

//read the data from the database
Query query = ftSession.createCriteria(Item.class);
List<Item> items = query.list();

//index the data
for (Item item : items) {
   ftSession.index(item);
}


tx.commit();
```

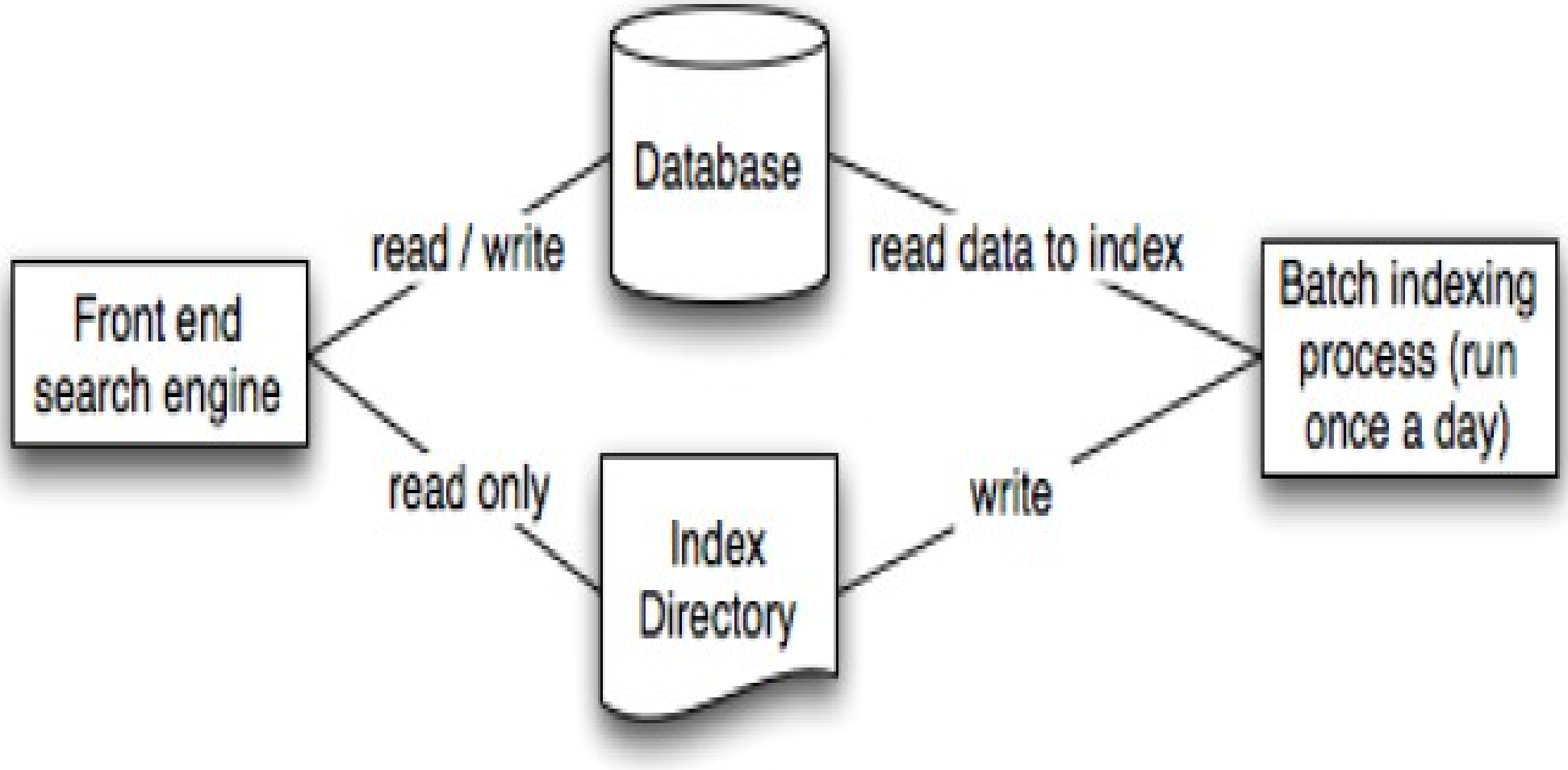
OutOfMemoryError

Load “distributor” for each item

Manual Index – The Right Way

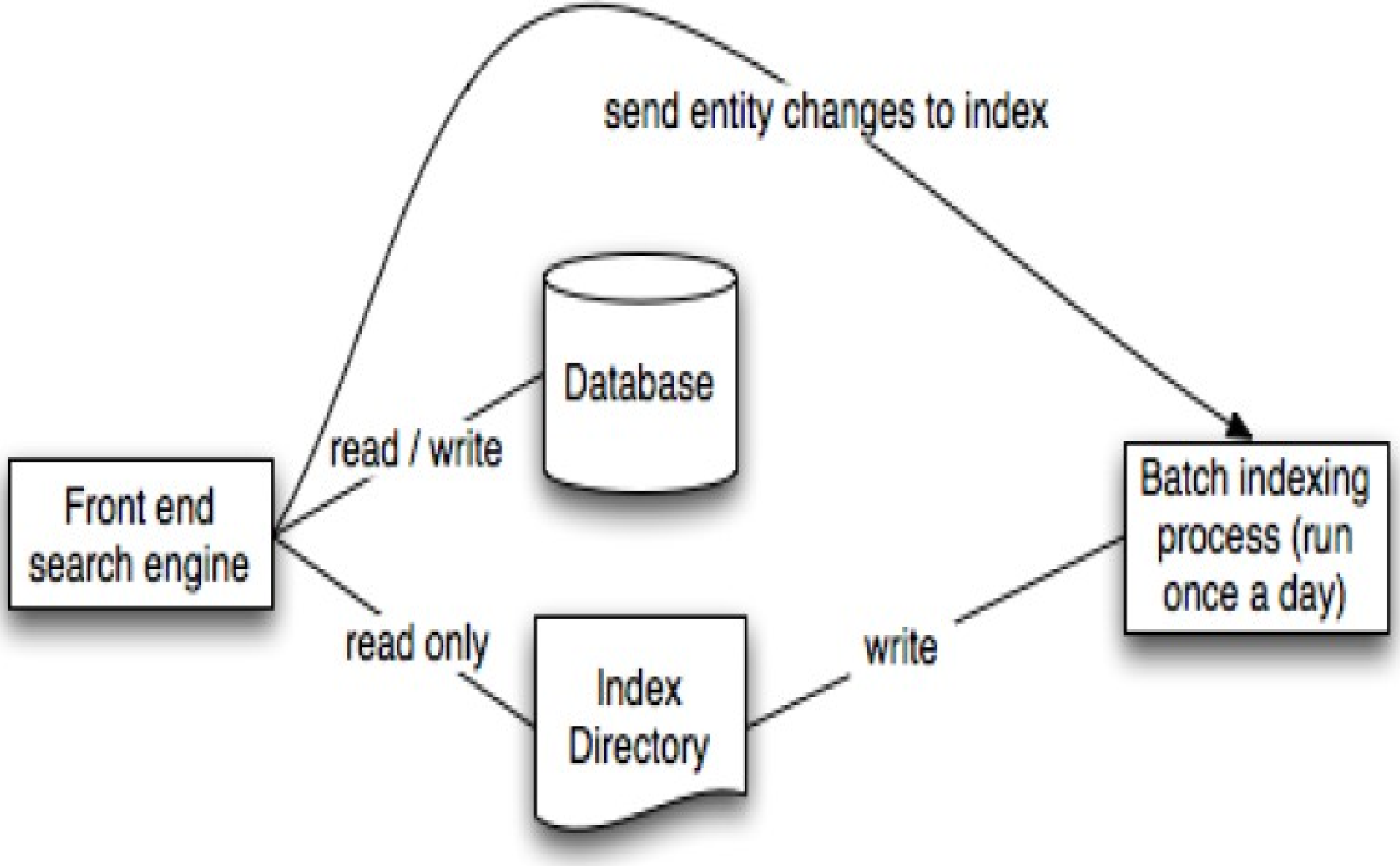
```
Transaction tx = ftSession.beginTransaction();

ftSession.setFlushMode(FlushMode.MANUAL);//disable flush
ftSession.setCacheMode(CacheMode.IGNORE);//disable 2nd level cache

ScrollableResults results = ftSession.createCriteria( Item.class )
    .setFetchMode("distributor", FetchMode.JOIN)
    .setResultTransformer(CriteriaSpecification.DISTINCT_ROOT_ENTITY)
    .setFetchSize(BATCH_SIZE);
    .scroll( ScrollMode.FORWARD_ONLY );


for(int i=1; results.next() ; i++) {
     ftSession.index( results.get(0) );
     if (i % BATCH_SIZE == 0) {
         ftSession.flushToIndexes(); //apply changes to the index
         ftSession.clear(); //clear the session releasing memory
   }
}
tx.commit(); //apply the remaining index changes
```

Index With Batch Approach

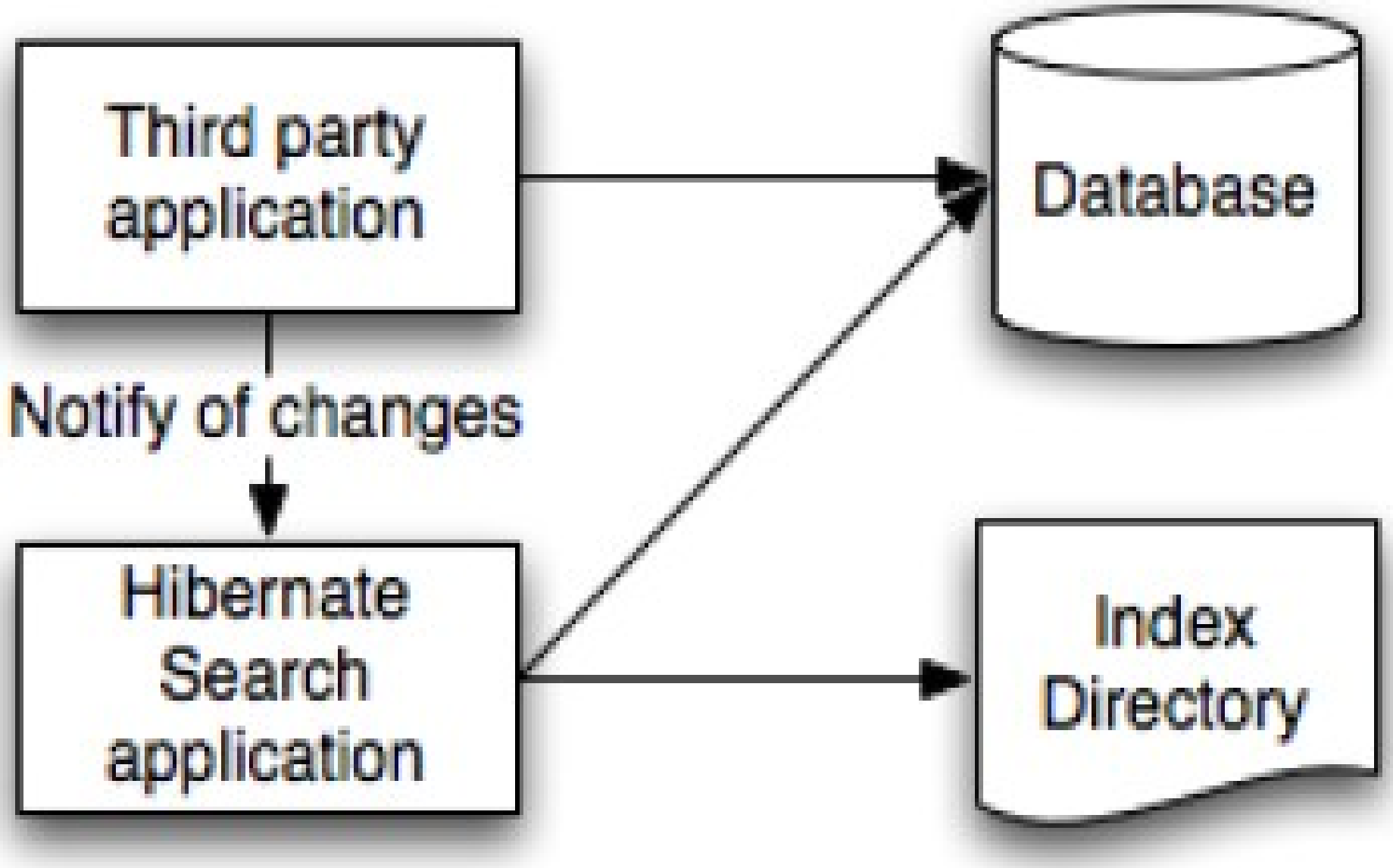



```
hibernate.search.indexing_strategy = manual
```

Mix Batch And Event Approach



Third Party Updates Your DB



What Influences Indexing Time

- Number of properties indexed
- Type of analyzer used
- Properties stored
- Properties embedded

- On Mass Indexing
  - » Index asynchronously
  - » Index on a different machine
  - » Use our previous manual sample as a template
  - » `session.getSearchFactory().optimize();`

Query – Solving The Retrieval Mismatch

Full-Text Search Query

- Running Hibernate-Search Query:
  - **Building a Lucene query** to express the full text search (either through the query parser or the programmatic API)
  - **Building an Hibernate Search query** wrapping the Lucene query
  - **Execute Hibernate Search Query.**

- But why do we need this wrapper around Lucene ?
  - Build the Lucene Query is easy :
    - title:Always description:some desc actors.name:Tom Cruise

Executing Lucene Query Is Low Level API

- Open the Lucene directory(ies)
- Build one or several IndexReaders, and an IndexSearcher on top of them
- Call the appropriate execution method from IndexSearcher.
- Resource management for Lucene API
- Convert Documents into objects of your domain model.
  - » "rehydrate" values from Lucene index
    - No lazy loading, No transparent access, No change propagation
  - » Load entities using ORM
    - Loading one by one will work inefficiently

Hibernate Search Query

- Return managed Hibernate entities.
- Query API is similar. Use the same Query API as JPA or Hibernate-Query API.
- Query semantic is also similar.
  - » Lazy loading mechanism.
  - » Transparent propagation to DB and Index

Build Lucene Query With QueryParser

```
private org.apache.lucene.search.Query buildLuceneQuery
                              (String words, Class<?> searchedEntity) {

  Analyzer analyzer = getFTEntityManager().getSearchFactory()
                                          .getAnalyzer(searchedEntity);

  QueryParser parser = new QueryParser( "title", analyzer );
  org.apache.lucene.search.Query luceneQuery = parser.parse(words);
  return luceneQuery;
}
```

Build Lucene Query With MutilFieldParser

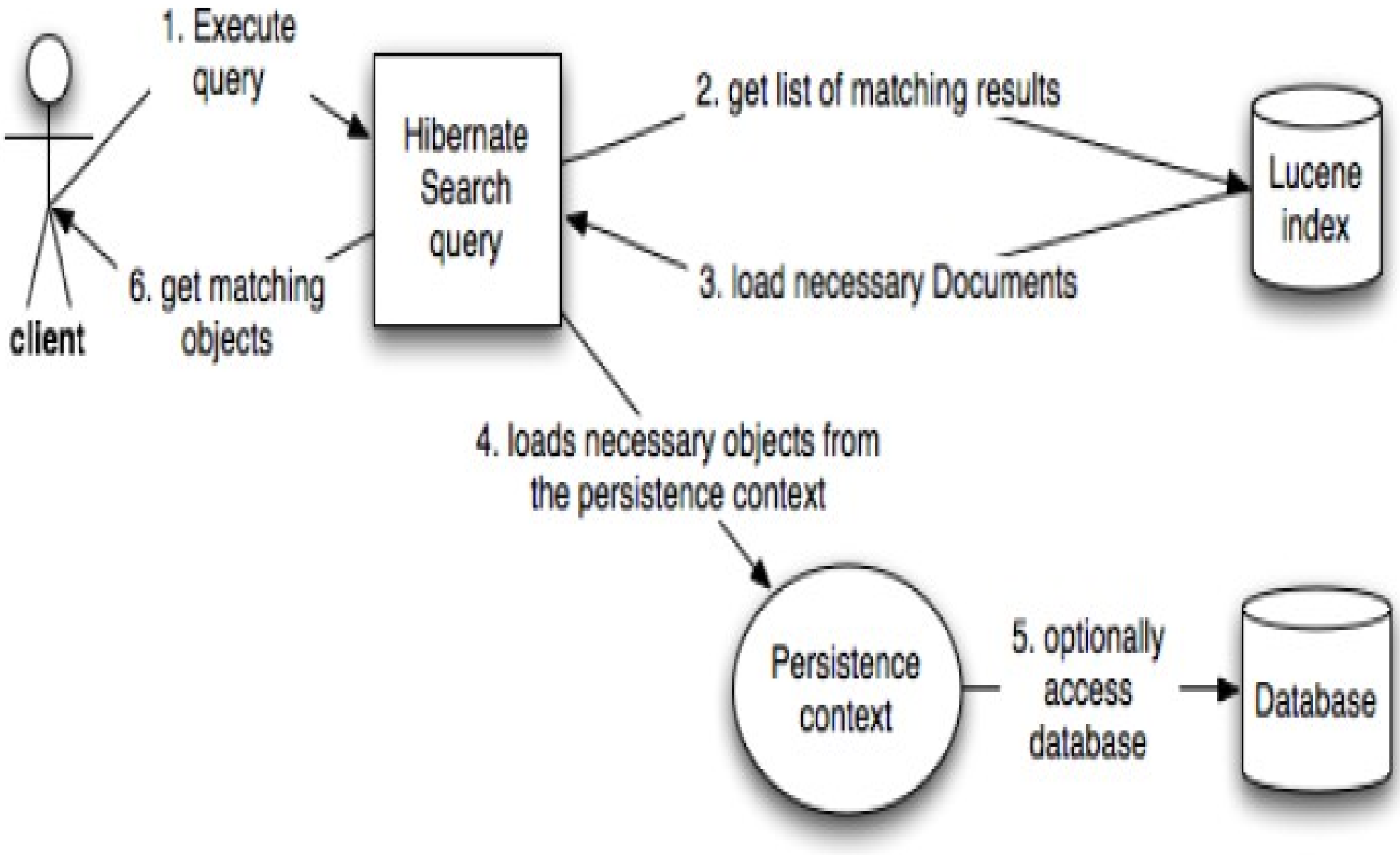
```
private org.apache.lucene.search.Query buildLuceneQuery
                              (String words, Class<?> searchedEntity) {

  Analyzer analyzer=getFTEntityManager().getSearchFactory()
                                        .getAnalyzer(searchedEntity);


  String[] productFields = {"title", "description"};
  Map<String,Float> boostPerField = new HashMap<String,Float>;
  boostPerField.put( "title", 4f);
  boostPerField.put( "description", 1f);

  QueryParser parser = new MultiFieldQueryParser(
                                  productFields,analyzer,boostPerField);

  org.apache.lucene.search.Query luceneQuery = parser.parse(words);
  return luceneQuery;
}
```

Build & Execute The FullTextQuery

```
@PersisstenceContext private EntityManager em;

private FullTextEntityManager getFTEntityManager() {
  return Search.getFullTextEntityManager(em);
}
...
```

```
public List<Item> findByTitle(String words) {
  org.apache.lucene.search.Query luceneQuery =
                                  buildLuceneQuery(words,Item.class);
  javax.persistence.Query query =
    getFTEntityManager().createFullTextQuery(luceneQuery,Item.class);

 return query.getResultList();
}
```

Execute FullTextQuery



Pagination & Result Size

```
public Page<Item> search(String words,int pageNumber,int window) {
  org.apache.lucene.search.Query luceneQuery =
                                    buildLuceneQuery(words,Item.class);
  FullTextQuery query =
     getFTEntityManager().createFullTextQuery(luceneQuery,Item.class);

  List<Item> results = query
                         .setFirstResult( (pageNumber - 1) * window )
                         .setMaxResults(window)
                         .getResultList();

  int resultSize = query.getResultSize();

  Page<Item> page = new Page<Item>(resultSize, results);
  return page;
}
```

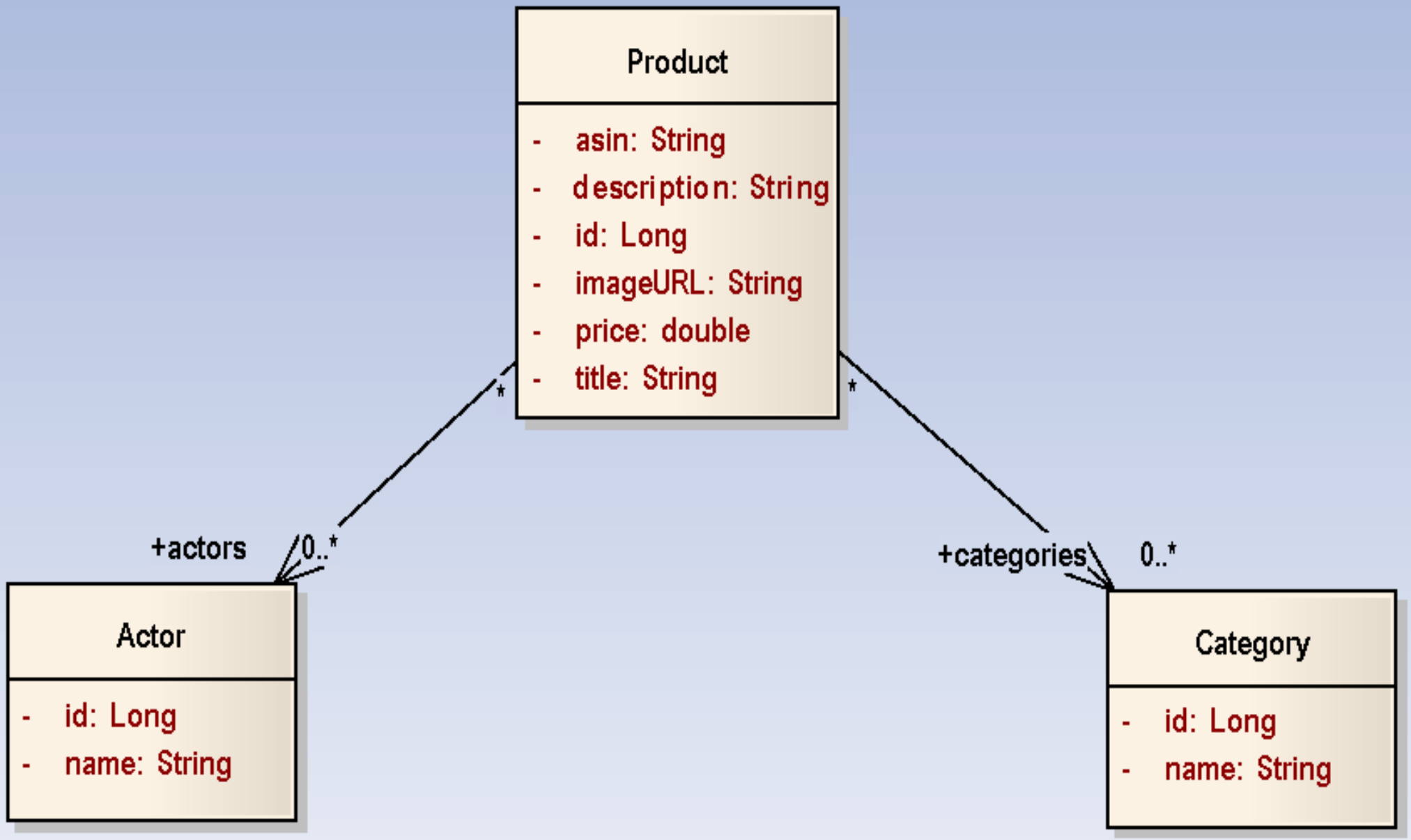
Override Fetch Strategy

```
public List<Item> findByTitle(String words) {
  org.apache.lucene.search.Query luceneQuery =
                                    buildLuceneQuery(words,Item.class);
  FullTextQuery query =
         getFTSession().createFullTextQuery(luceneQuery,Item.class);

  Criteria fetchingStrategy =
            getFTSession().createCriteria(Item.class)
            .setFetchMode("actors", FetchMode.JOIN);

  query.setCriteriaQuery(fetchingStrategy);

  return query.list();
}
```

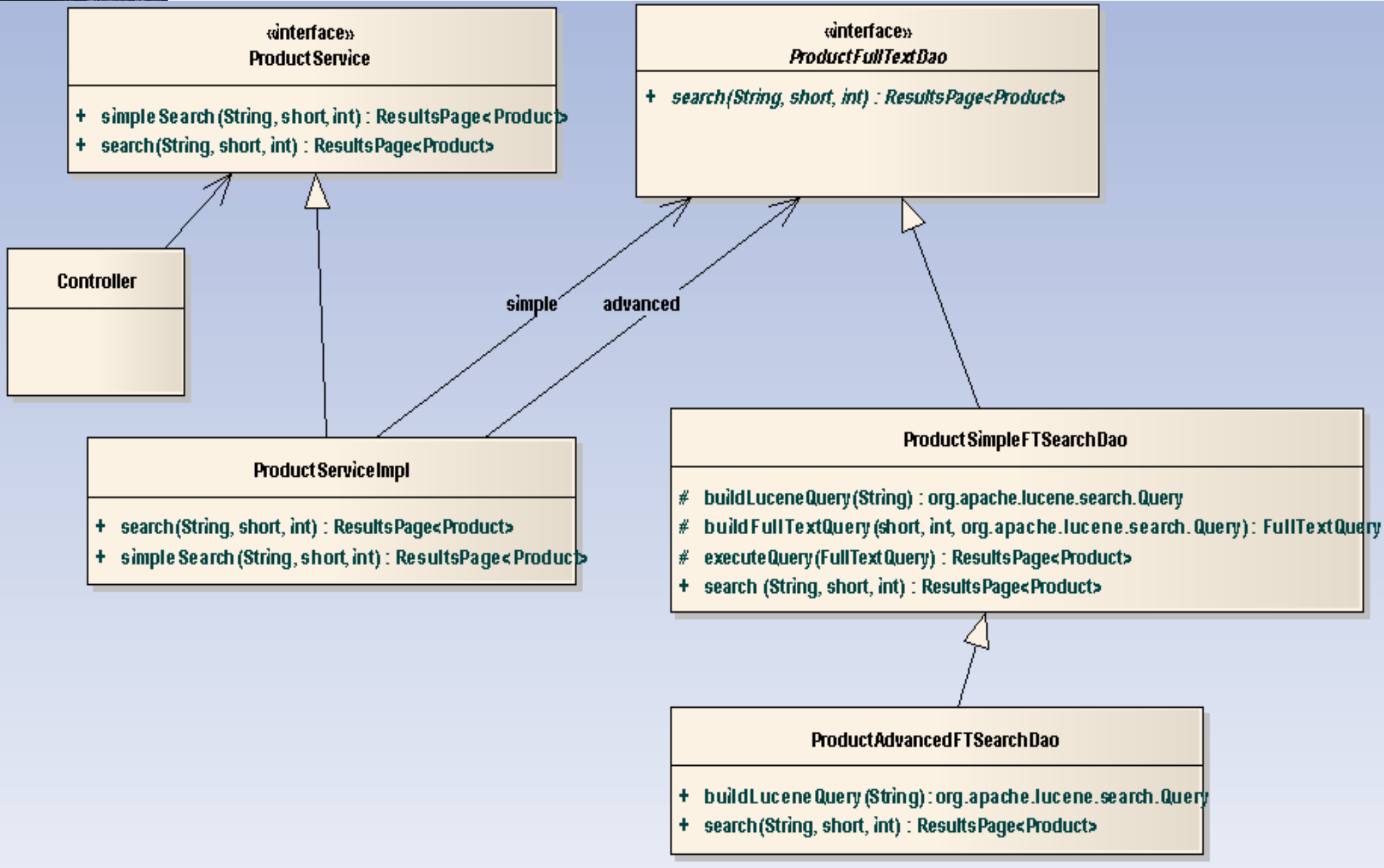
- For JPA use getDelegate()
- Don't use Criteria restrictions
  - » Will hurt pagination and will provide wrong resultSize

Demo

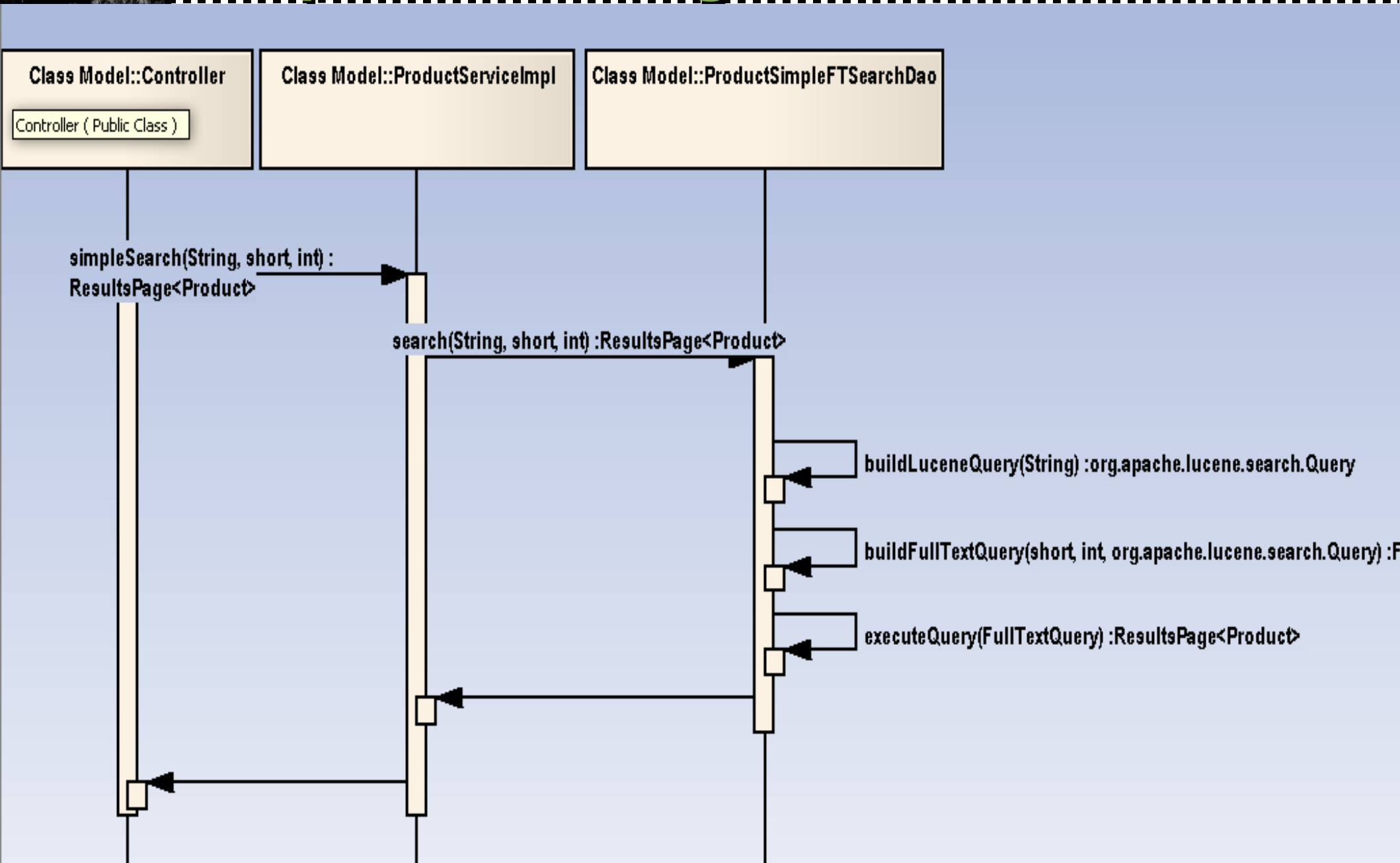
Product Domain Model

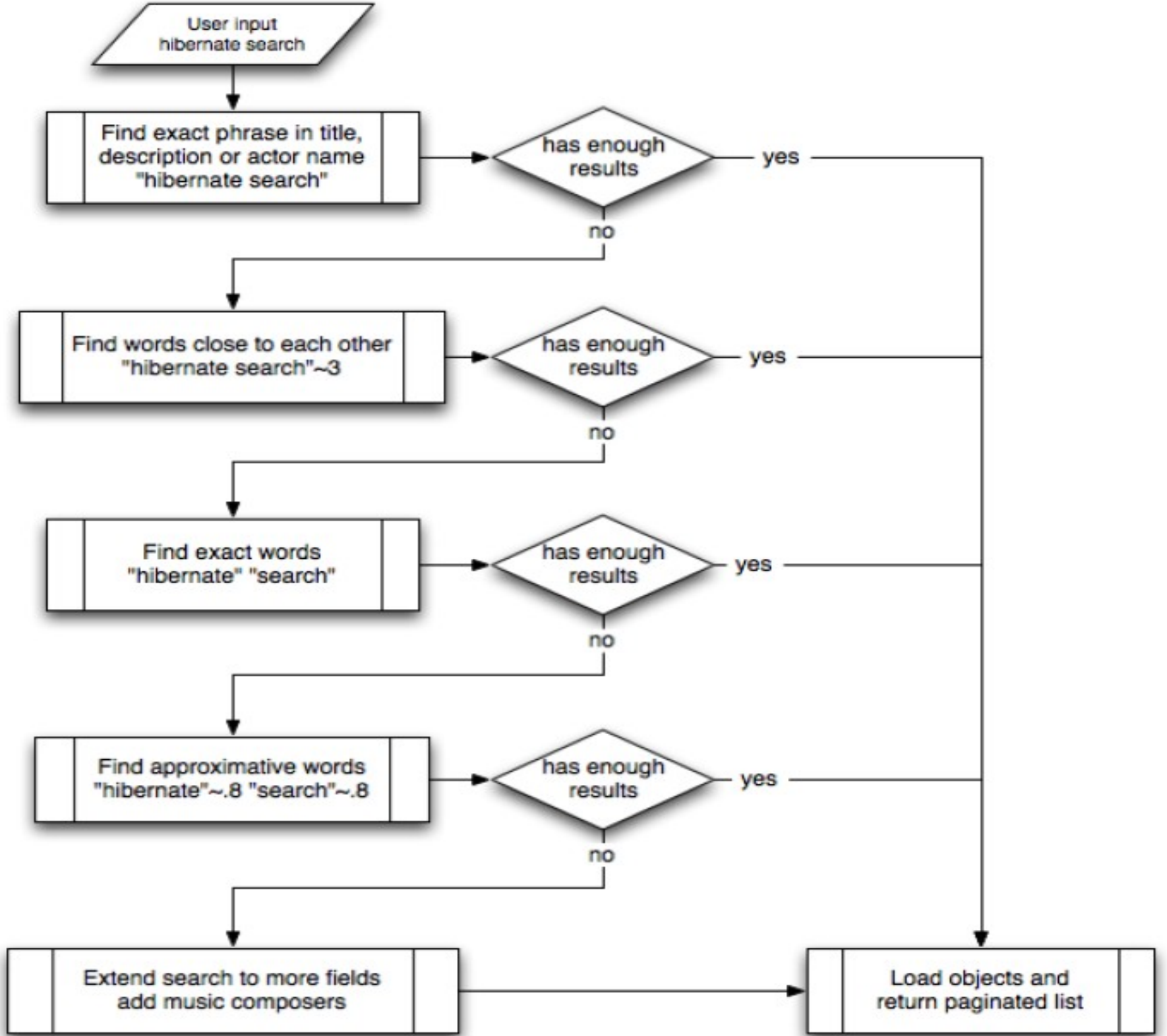


Service & DAO Layers



Simple Search Sequence Diagram




User input
hibernate search
Find exact phrase in title, description or actor name "hibernate search"
has enough results
yes
no
Find words close to each other "hibernate search"~3
has enough results
yes
no
Find exact words "hibernate" "search"
has enough results
yes
no
Find approximative words "hibernate"~.8 "search"~.8
has enough results
yes
no
Extend search to more fields add music composers
Load objects and return paginated list

Projection

```
public class ItemView {// A view Object NOT necessary an entity...
  private String ean;
  private String title;
  public String getEan() {
    return ean;
  }
  public String getTitle() {
    return title;
  }
}
```

```
public List<ItemView> search(String words) {
  org.apache.lucene.search.Query luceneQuery =
                                  buildLuceneQuery(words,Item.class);
  FullTextQuery query =
    getFTSession().createFullTextQuery(luceneQuery,Item.class);

  query.setProjection("ean", "title");

  List<ItemView> results = query.setResultTransformer(
            new AliasToBeanResultTransformer(ItemView.class)).list();
  return results;
}
```

No hit on the DB

Store Properties In The Index For Projection

```
@Entity @Indexed
public class Item {
    @Id @GeneratedValue
    private Integer id;

    @Field(store=Store.YES)
    private String title;

    @Field
    private String description;

    @Field(index=Index.UN_TOKENIZED, store=Store.YES)
    private String ean;
    ...
}
```

Sorting By Field

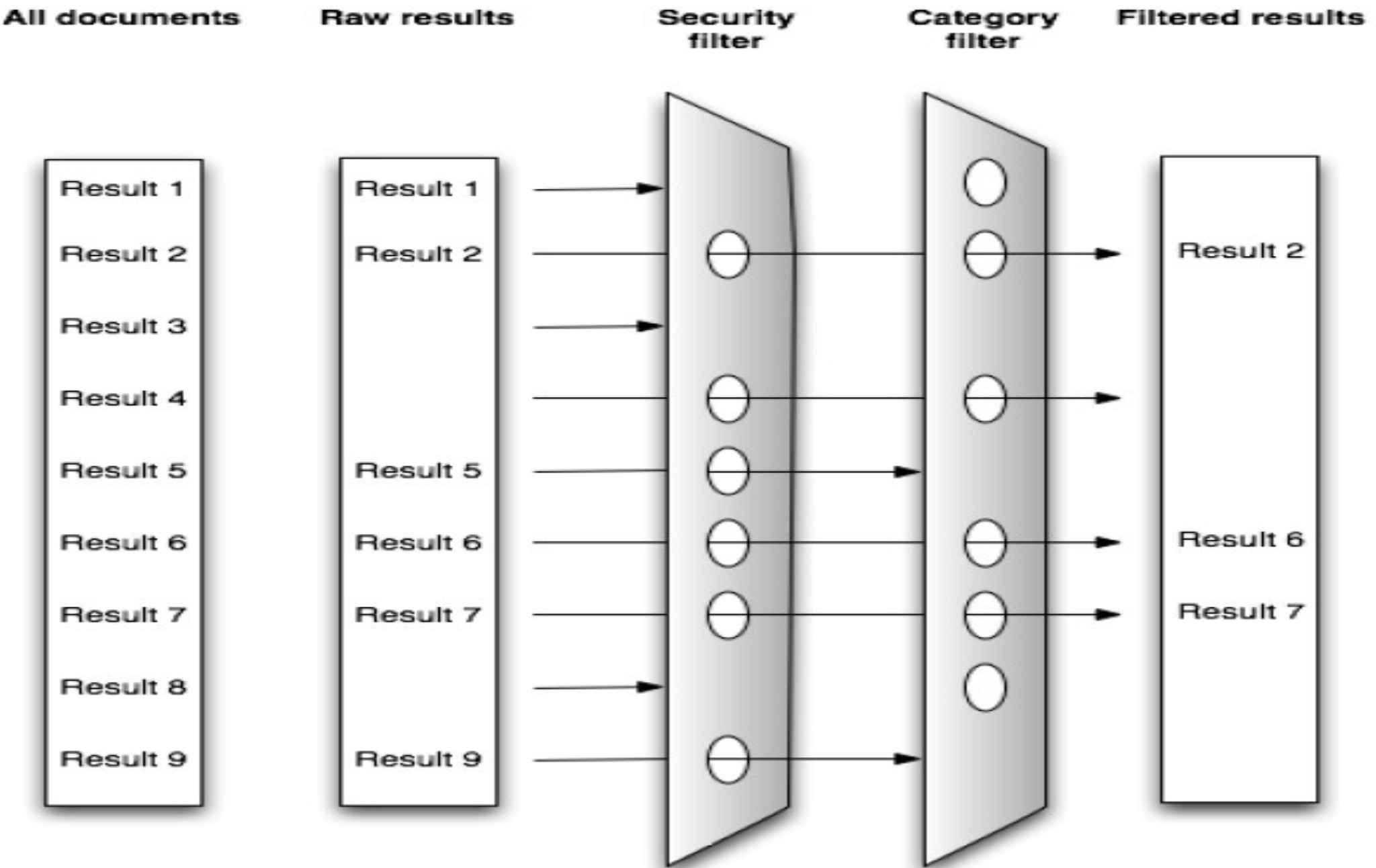
```
public List<Item> findByTitle(String words) {
  org.apache.lucene.search.Query luceneQuery =
                                    buildLuceneQuery(words,Item.class);
  FullTextQuery query =
         getFTSession().createFullTextQuery(luceneQuery,Item.class);

  Sort sort = new Sort(new SortField("title_sort",SortField.STRING));

  query.setSort(sort);

  return query.list();
}
```

```
@Entity @Indexed public class Item {
 ...
  @Fields({
    @Field(index=Index.TOKENIZED)
    @Field(name="title_sort", index=Index.UN_TOKENIZED)
  })
  private String title;

}
```

Dynamic Data Filtering

- Restrict results of a query after the Lucene query has been executed
  - » Rules that are not directly related to the query.
  - » Cross-cutting restrictions
    - category, availability , security.
- The ordering defined by the original query is respected.

Filters


All documents
Raw results
Security filter
Category filter
Filtered results
Result 1
Result 2
Result 3
Result 4
Result 5
Result 6
Result 7
Result 8
Result 9
Result 1
Result 2
Result 5
Result 6
Result 7
Result 9
Result 2
Result 6
Result 7

Filter Example

```
//service implementation
public List<Item> searchItems(String search, boolean isChild) {
  org.apache.lucene.search.Query luceneQuery =
                                                buildLuceneQuery(search);
  FullTextQuery query =
        getFTSession().createFullTextQuery(luceneQuery, Item.class);

  if (isChild)
    query.enableFullTextFilter("chldFilter");

  List<Item> results = query.list();
  return results;
}
```

Filter Example Cont.

```
@Entity @Indexed
@FullTextFilterDef(name="childFilter",
                   impl=ChildFilterFactory.class)
@ClassBridge(name="childrenOnly",
             impl=ChildrenFlagBridge.class,index=Index.UN_TOKENIZED)
public class Item {
...
}
```

```
public class ChildFilterFactory {
  @Factory
  public Filter getChildrenFilter() {
    Query query = new TermQuery( new Term("childrenOnly", "yes") );
    return new QueryWrapperFilter( query );
  }
}
```

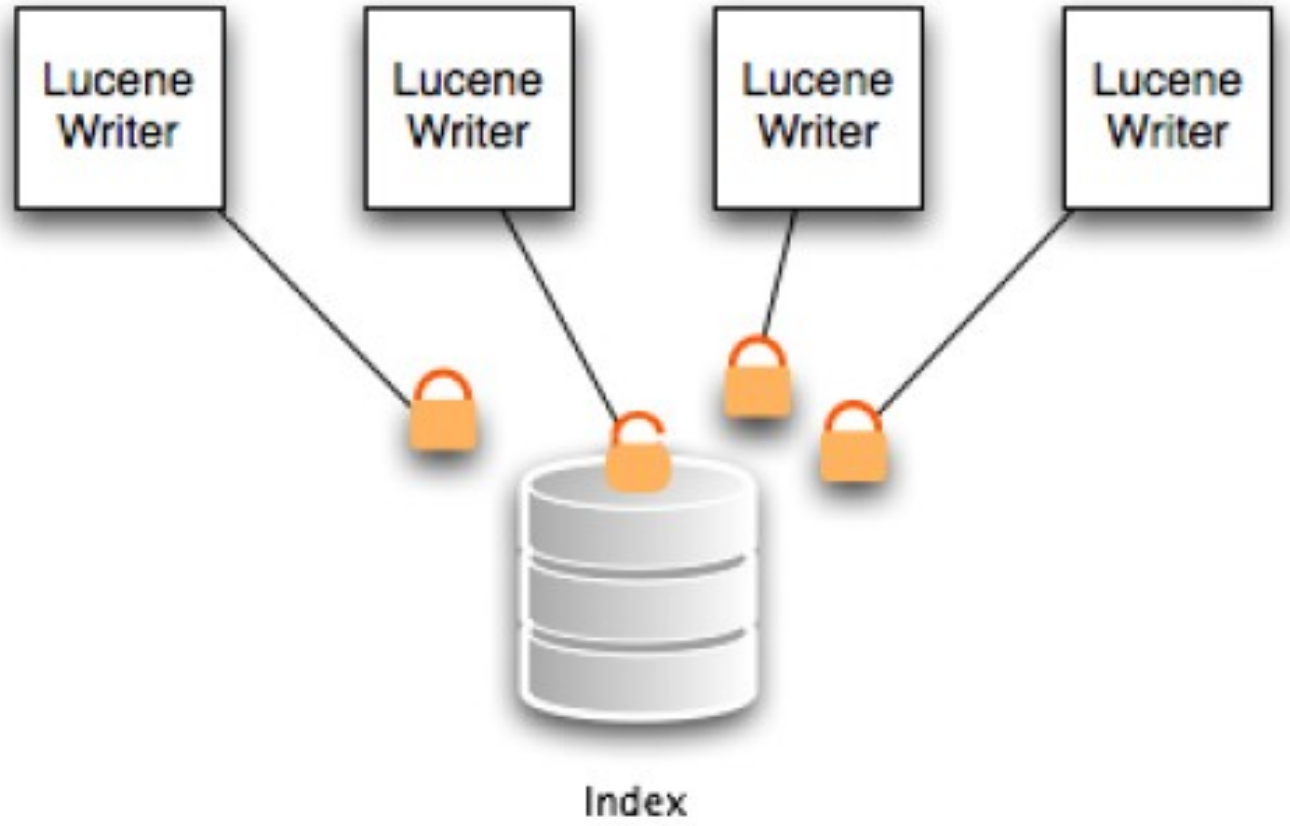
Optimizing Search

- Limit targeted classes (one class is the best)
  - » ftSession.createFullTextQuery(luceneQuery, Item.class);
- Use pagination
- Avoid the n+1 by using setCriteria()
- Use projection carefully

Scale Hibernate Search

Synchronous Clustering

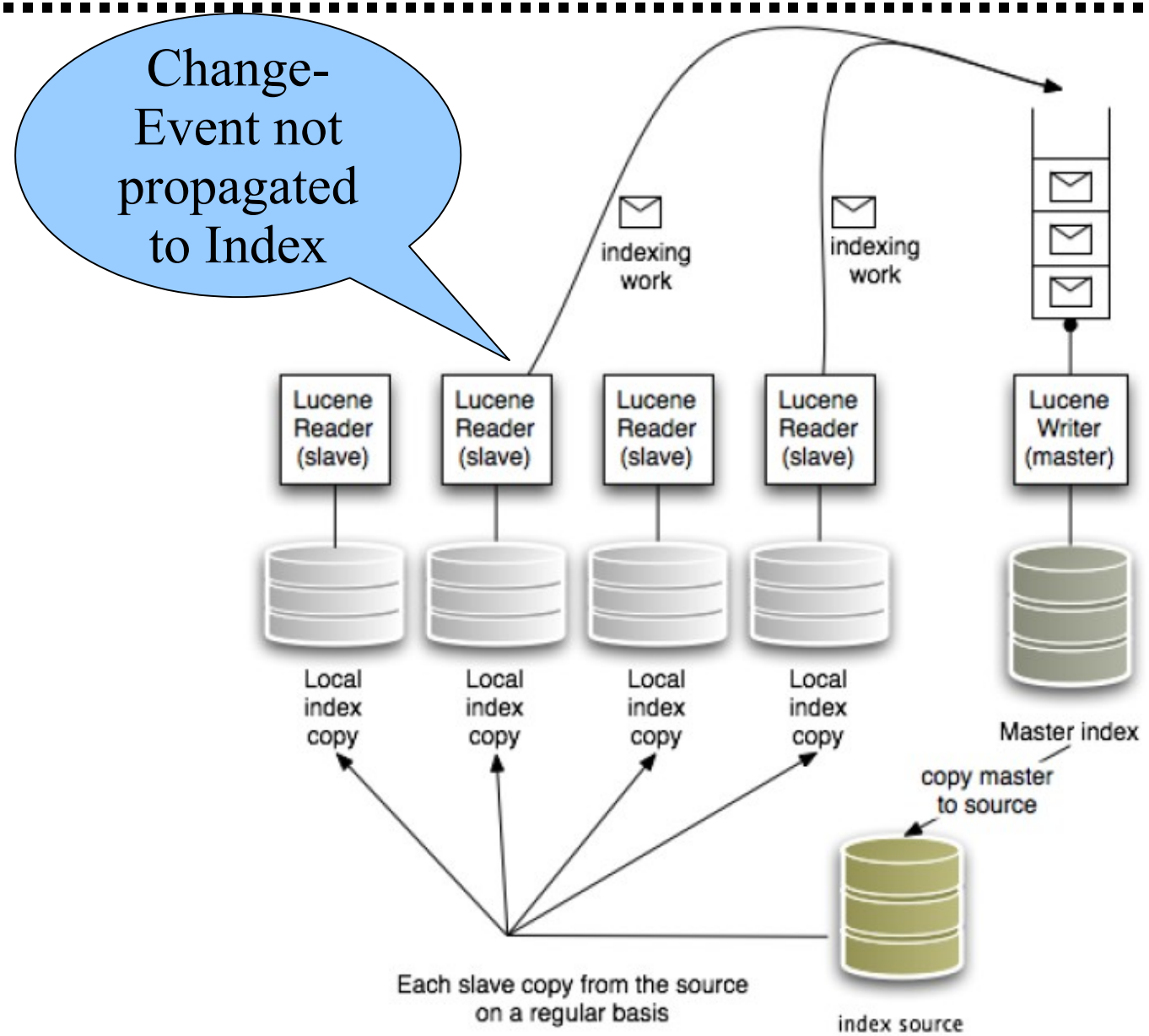
- Who can use it?
  - » Applications with medium-size indexes
    - Network traffic will be needed to retrieve the index.
  - » Applications with low to moderate write intensive .



Synchronous Clustering Problems

- Some NFS cache the directory contents
  - » No immediate visibility for the directory content
    - Lucene relies (partially) on an accurate listing of files.
  - » “delete on last close” semantic NOT always implemented .
- Database Directory issues
  - » Segments are represented as blobs
  - » A pessimistic lock hurts concurrency on massive updates.
- In-memory distributed Directory
  - » GigaSpace, JBoss Cache and Terracotta

Asynchronous Clustering



Slave Configuration

```
<persistence-unit name="dvdstore-catalog">
  <jta-data-source>java:/DefaultDS</jta-data-source>
  <properties>
    <!-- regular Hibernate Core configuration -->
    <property name="hibernate.dialect"
                          value="org.hibernate.dialect.H2Dialect"/>


    <!-- JMS backend →
    <property name="hibernate.search.worker.backend"
                                        value="jms"/>
    <property name="hibernate.search.worker.jms.connection_factory"
                                        value="/ConnectionFactory"/>
    <property name="hibernate.search.worker.jndi.url"
                                        value="jnp://master:1099"/>
    <property name="hibernate.search.worker.jms.queue"
                                        value="queue/hibernatesearch"/>
  ...
```

Slave Configuration Cont.

```
...

    <!-- DirectoryProvider configuration -->
    <property name="hibernate.search.default.directory_provider"
       value="org.hibernate.search.store.FSSlaveDirectoryProvider"/>
     <property name="hibernate.search.default.refresh"
                                                  value="1800"/>
     <property name="hibernate.search.default.indexBase"
                                          value="/Users/prod/lucenedirs"/>
     <property name="hibernate.search.default.sourceBase"
                                                 value="/mnt/share"/>
  </properties>
</persistence-unit>
```

Master Configuration

```
<persistence-unit name="dvdstore-catalog">
  <jta-data-source>java:/DefaultDS</jta-data-source>
  <properties>
    <!-- regular Hibernate Core configuration -->
    <property name="hibernate.dialect"
                          value="org.hibernate.dialect.H2Dialect"/>

    <!-- Hibernate Search configuration -->
    <!-- no backend configuration necessary -->
    <property name="hibernate.search.default.directory_provider"
      value="org.hibernate.search.store.FSMasterDirectoryProvider"/>

    <property name="hibernate.search.default.refresh"
      value="1800"/>

    <property name="hibernate.search.default.indexBase"
      value="/Users/prod/lucenedirs"/>

    <property name="hibernate.search.default.sourceBase"
      value="/mnt/share"/>

  </properties>
</persistence-unit>
```

Building The Master MDB

```
@MessageDriven(activationConfig = {
  ActivationConfigProperty(propertyName="destinationType",
                                    propertyValue="javax.jms.Queue"),
  @ActivationConfigProperty(propertyName="destination",
                                  propertyValue="queue/hibernatesearch")
} )
public class MDBSearchController extends
    AbstractJMSHibernateSearchController implements MessageListener{

  @PersistenceContext private EntityManager em;

  @Override
  protected void cleanSessionIfNeeded(Session session) {
    //clean the session if needednothing to do container managed
  }

  @Override
  protected Session getSession() {
    return (Session) em.getDelegate();
  }
}
```

What Happens On Master Failure ?

- Slave
  - » Continue to serve full-text queries
  - » Continue push changes that need indexing.
- Master
  - » Messages on the master are roll-backed to queue.
  - » Optional – Prepare a standby for the master
- On corrupted Index…
  - » Re-index manually from DB
  - » Optional – Use Storage Area Network (SAN)

Summary

Full-Text Search Without The Hassle

- Solves The 3 mismatch problems
  - » Automatic structural conversion through Mapping
  - » Transparent index synchronization
  - » Retrieved data from index become “persistent” entities.
- Easier / Transparent optimized Lucene use
- Scalability capabilities out of the box

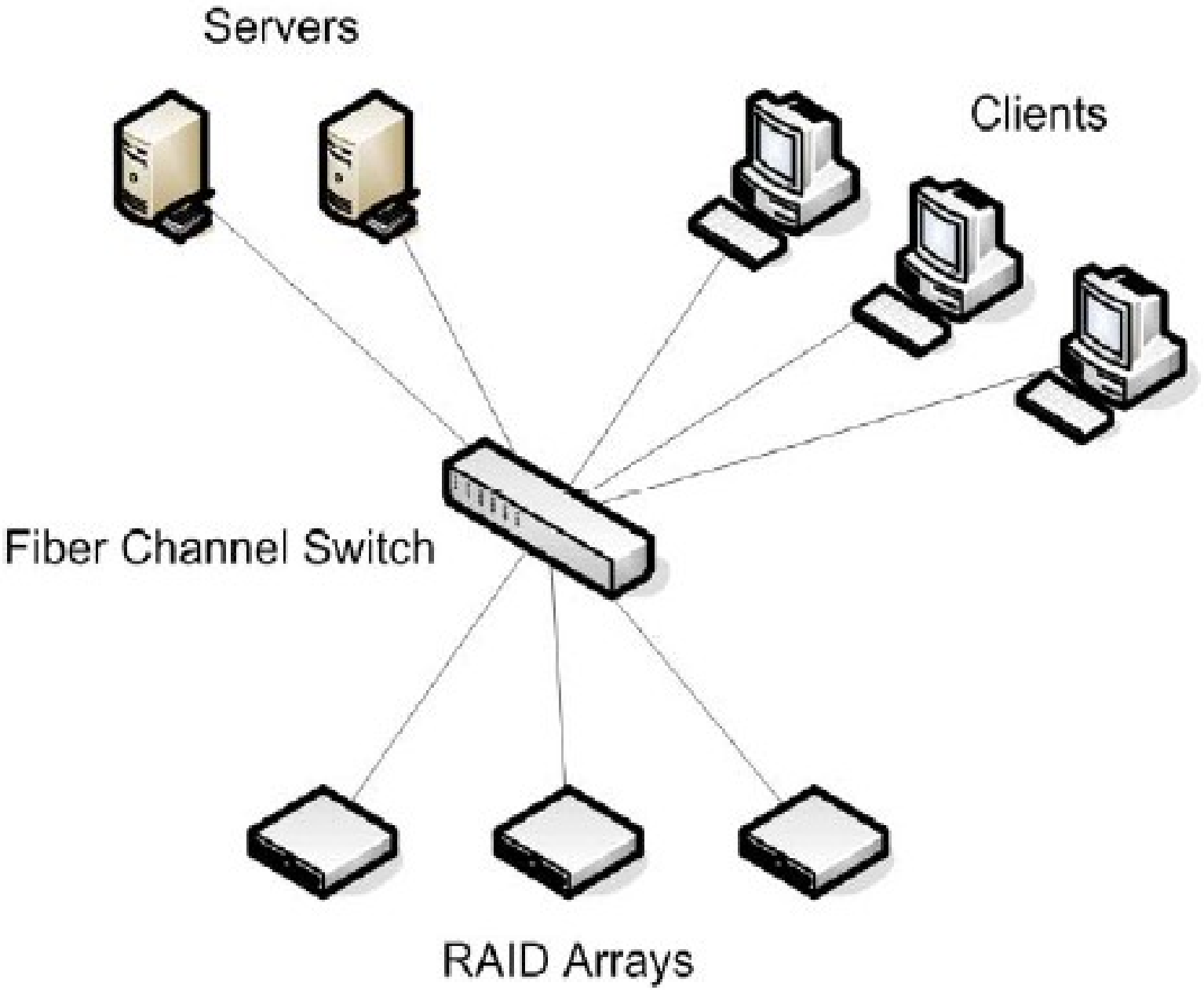
Q & A

Thank You

yanai@tikalk.com

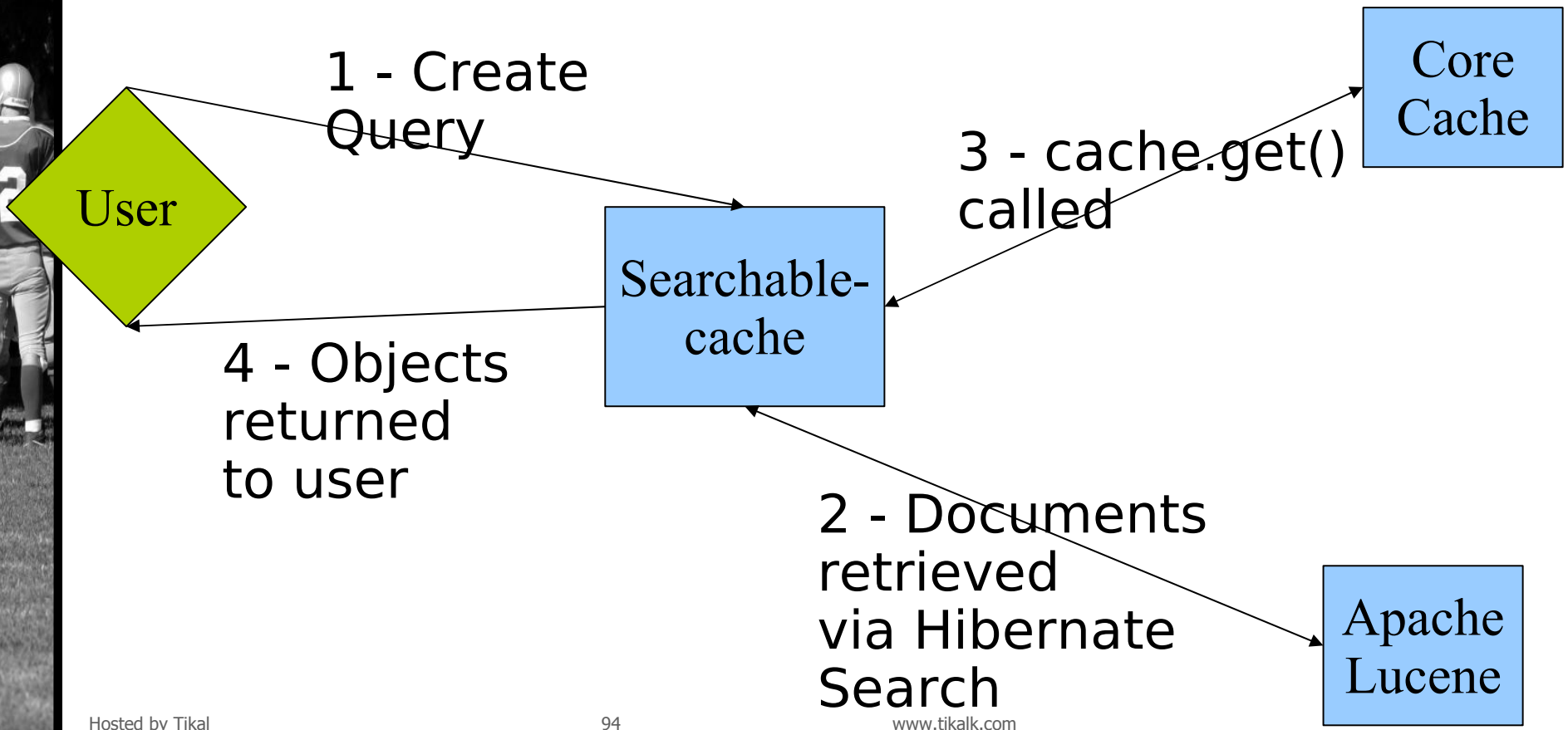
Appendixes

Use SAN For Lucene Directory



JBoss Cache Searchable

- Integration package between JBoss Cache and Hibernate Search.
- Provides full text search capabilities to the cache.



Annotated Pojo

```
@Indexed
@ProvidedId
public class Person { //Not necessary a Hibernate Entity

  @Field
  private String name;

  @Field
  private Date dateOfBirth;

  //Not Indexed
  private String massiveString;

  //Standard getters, setters etc follow.

}
```

FullText Search on Cache

```
Cache<String, Person> cache =
            new DefaultCacheFactory<String, Person>().createCache();
SearchableCache searchableCache = new SearchableCacheFactory();
                          createSearchableCache(cache, Person.class);
...
```

```
public void putStuffIn(Person p){
  searchableCache.put(Fqn.fromString("/a/b/c"), p.getName(), p);
}
```

```
public List findStuff(String searchStr){
  Query luceneQuery = buildLuceneQuery(String searchStr)

  CacheQuery cacheQuery =
              searchableCache.createQuery(luceneQuery,Person.class);

  return cacheQuery.list();
}
```