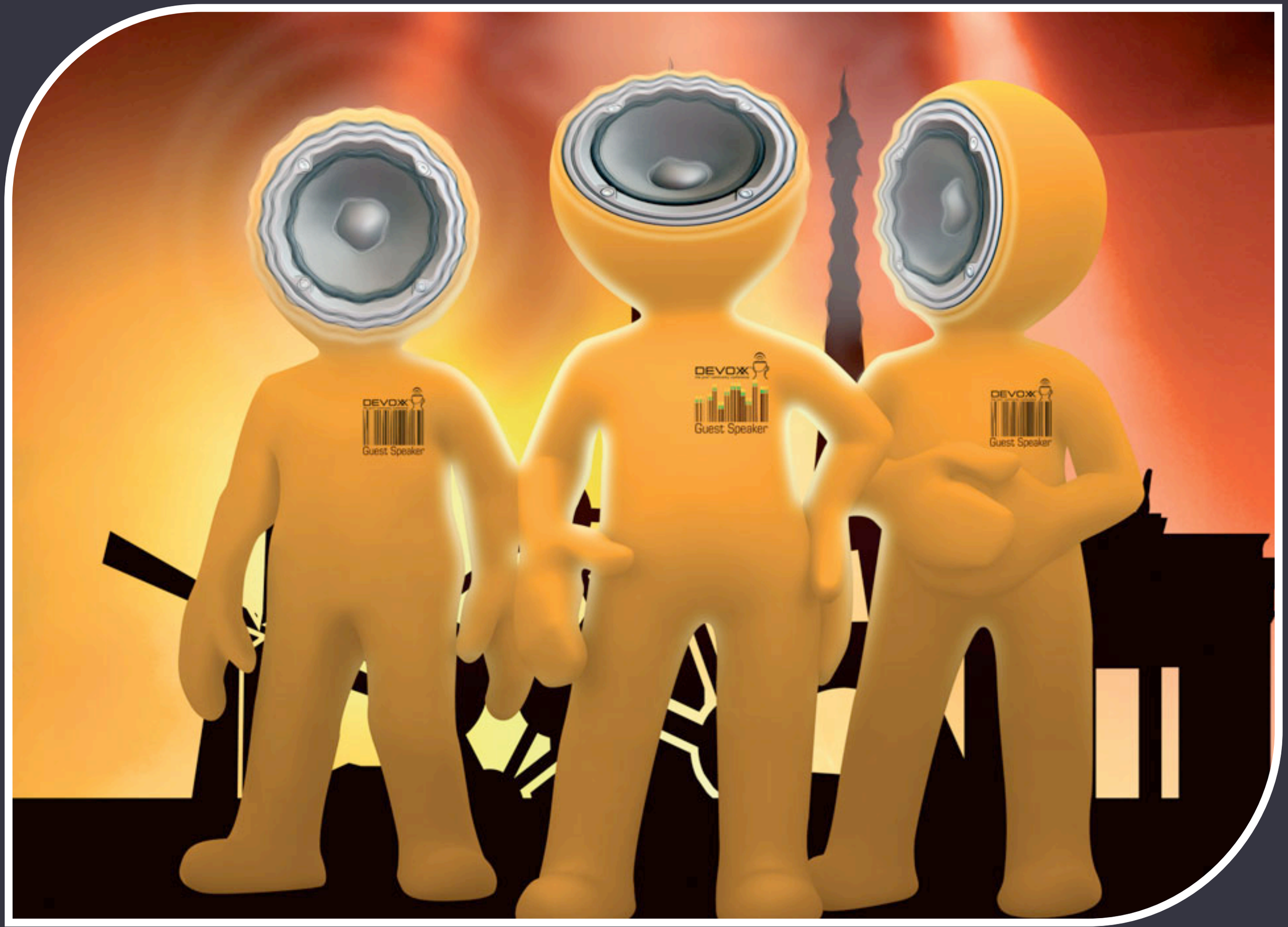

DEVOXX
Guest Speaker
DEVOXX
Guest Speaker
DEVOXX
Guest Speaker

Envers
Easy Entity ~~Versioning~~ Auditing

Adam Warski
Software Engineer
Level N Consulting

HIBERNATE

jboss.org

Who am I?

- Adam Warski :)
- Formerly at JBoss
- Senior Software Engineer at Level N Consulting
- Creator of Envers
- http://www.warski.org/blog (Envers, Seam, Typestate, ...)

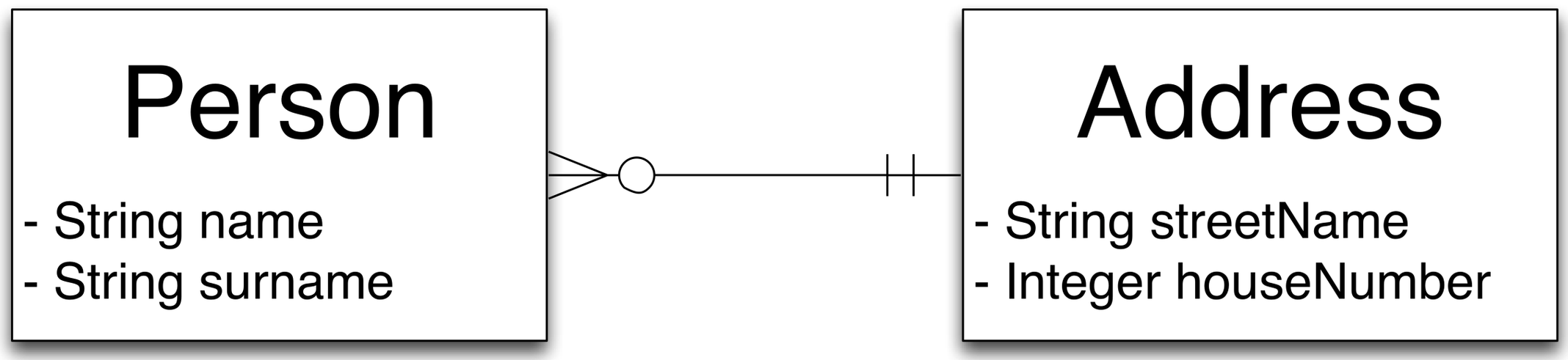
Agenda

- Overview of patterns for auditing
- What is Envers?
- How does it work?
- Configuration
- An example
- Queries
- Use cases
- Using Envers

Overview of patterns for auditing

What’s the problem and how to approach it?

Example: Person & Address



- We want to store the history of a Person's addresses

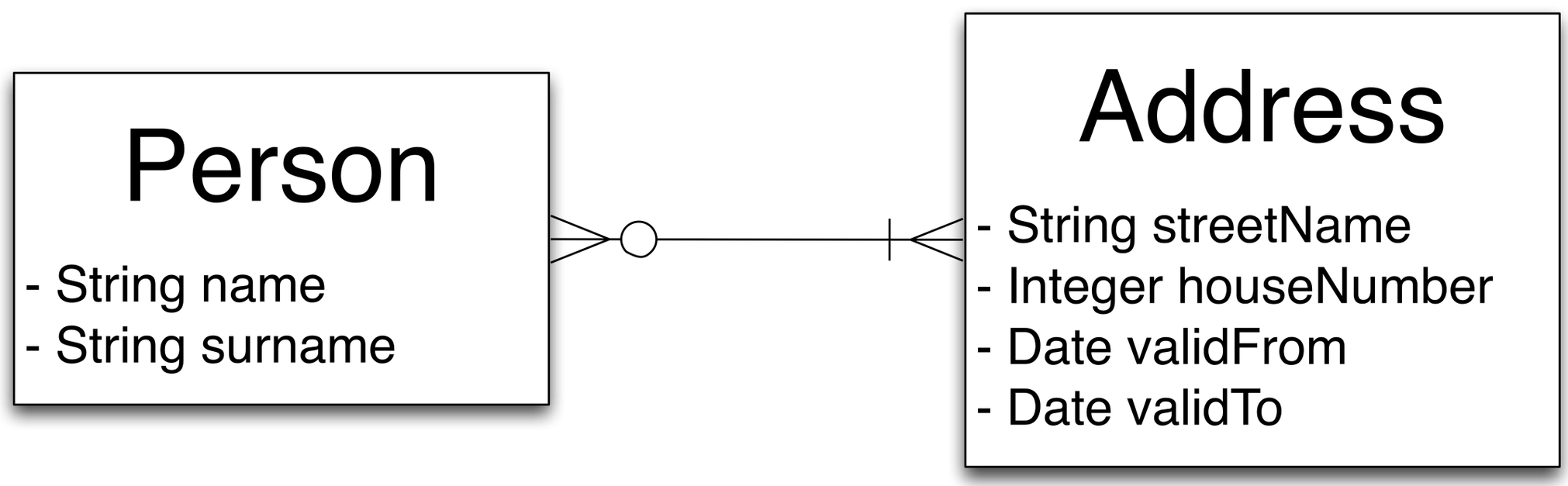
Patterns: **Audit Log**

- Logging all activity to a file or database
- Entity state: String or CSV
- Very simple
- Hard to read historic data

```
2008-08-10 06:24:19,761 [org.jboss.envers.example.Person] add,id=1,name=John,surname=Doe,address=10
2008-08-25 18:49:18,420 [org.jboss.envers.example.Person] mod,id=1,name=John,surname=Doe,address=24
2008-09-10 13:44:42,120 [org.jboss.envers.example.Address] add,id=29,streetName=East st.,houseNumber=53
2008-09-16 01:12:33,590 [org.jboss.envers.example.Person] mod,name=John,surname=Doe,address=29
~
```

Patterns: **Effectivity**

- Explicit start and end date (validity)
- Verbose
- “Manual”
- Complicates mapping



Patterns: **Temporal property**

- Getter with a Date argument
- Complicates mapping; facades needed
- Verbose
- When more properties are temporal: temporal object / snapshot

Person

- String name
- String surname
- Address getAddress(Date validOn)

Address

- String streetName
- Integer houseNumber

Patterns overall

- In most use cases and most of the time:
  - we are only interested in “current” data

- Historical data:
  - much less often
  - in different places - uniform access not a necessity

Envers

What is it and how does it work?

What is Envers?

- An entity auditing (versioning) library
- Part of Hibernate
- Simplifies storing and retrieving historical data

Assumptions

- Transparent: data can be used as always (queried, persisted, etc.)

- Not intrusive:
  - The database schema isn't changed (some tables can be added)
  - Minimal code changes

- Slowly changing data

How does Envers work?

- The programmer specifies which entities should be audited
- For each audited entity, an audit entity is (dynamically) created
- e.g. entity “Address” has an “Address_AUD” companion
- The companion stores historical data

How does Envers work?

- On an update/insert/delete: data inserted to audit tables
- All changes in a transaction: **1 revision**
- Revisions capture consistent state
- Revisions are global
- Similar to SVN

Envers Configuration

- To make an entity audited: annotate with **`@Audited`**
- Add event listeners to `persistence.xml`

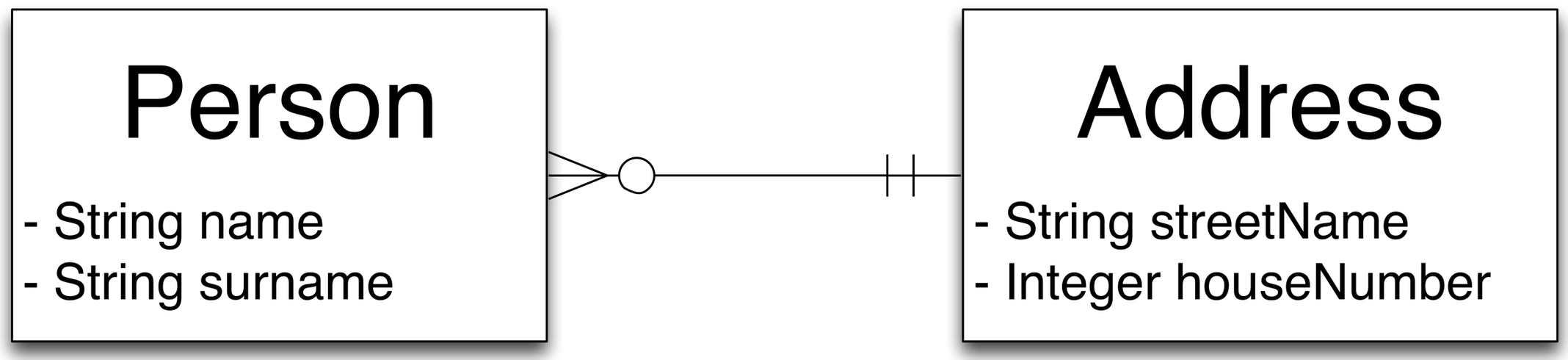
```
<property name="hibernate.ejb.event.post-insert"
          value="org.hibernate.envers.event.AuditEventListener" />
<property name="hibernate.ejb.event.post-update"
          value="org.hibernate.envers.event.AuditEventListener" />
<property name="hibernate.ejb.event.post-delete"
          value="org.hibernate.envers.event.AuditEventListener" />
<property name="hibernate.ejb.event.pre-collection-update"
          value="org.hibernate.envers.event.AuditEventListener" />
<property name="hibernate.ejb.event.pre-collection-remove"
          value="org.hibernate.envers.event.AuditEventListener" />
<property name="hibernate.ejb.event.post-collection-recreate"
          value="org.hibernate.envers.event.AuditEventListener" />
```

```
@Entity
public class Person {
    @Id
    @GeneratedValue
    private int id;

    @Audited
    private String name;

    @Audited
    private String surname;

    @Audited
    @ManyToOne
    private Address address;
```

What can be audited?

- Mappings defined by JPA:
  - Simple properties: `String`s, `Integer`s, `Date`s, ...
  - Components
  - Relations
- Some Hibernate extensions
  - Custom types
  - Collections

Person & Address example



- Now let's assume that both entities are annotated with `@Audited`

```
// Revision 1
em.getTransaction().begin();
Address a1 = new Address("West st.", 10);
Address a2 = new Address("East st.", 15);
Person p = new Person("John", "Doe");
p.setAddress(a1);
entityManager.persist(a1);
entityManager.persist(a2);
entityManager.persist(p);
em.getTransaction().commit();
```

```
// Revision 2
em.getTransaction().begin();
p = entityManager.find(Person.class, id);
p.setName("Paul");
p.setAddress(a2);
em.getTransaction().commit();
```

- Old person data is stored
- No code changes - completely transparent to the user

```
AuditReader ar = AuditReaderFactory.get(em);
// Reading the person at revision 1
old_p = ar.find(Person.class, id, 1);
assert "John".equals(old_p.getName());
assert a1.equals(old_p.getAddress());
```

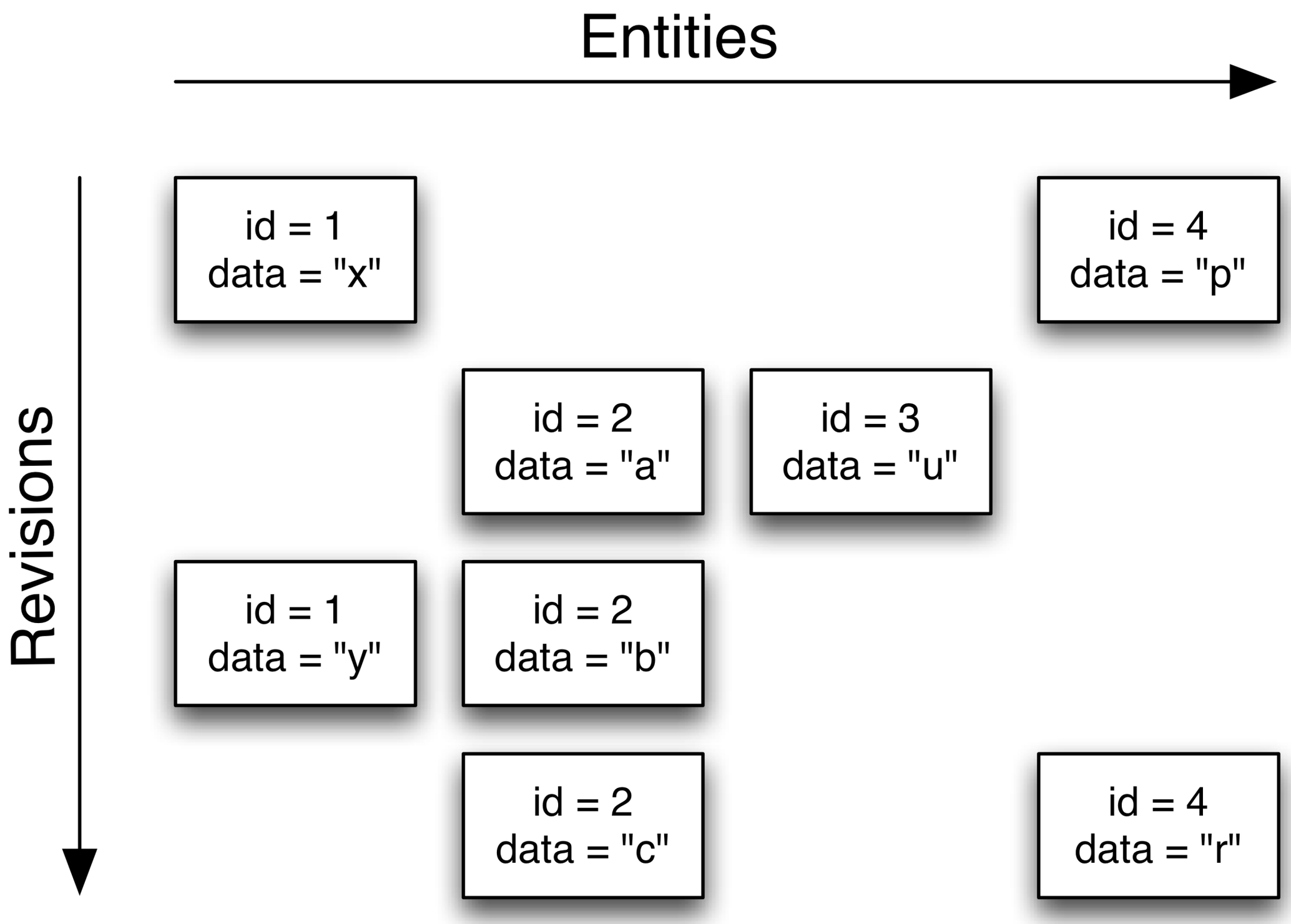
- Transparent traversing of relations

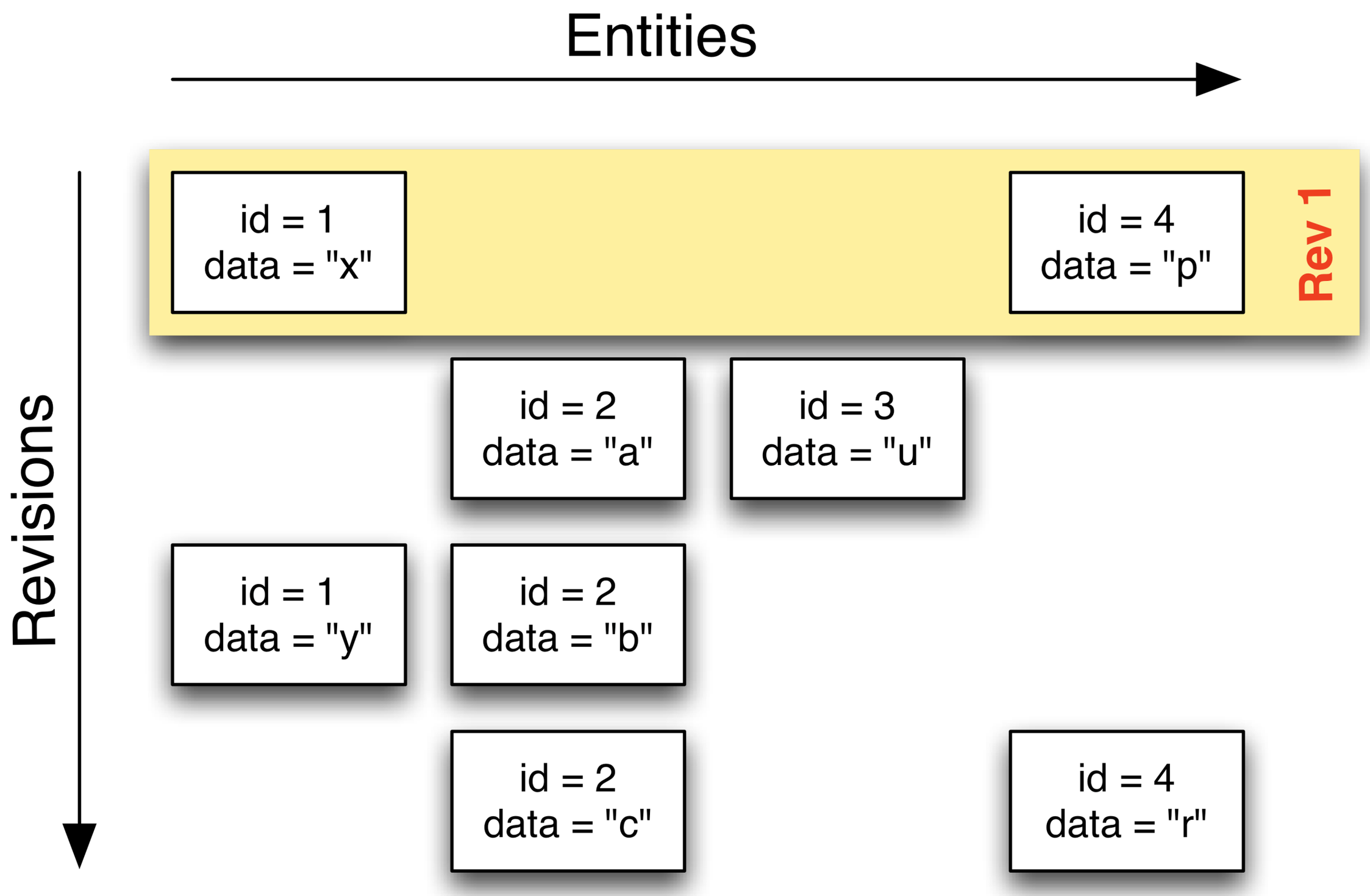
```
// Reading the addresses at revision 1
old_a1 = ar.find(Address.class, a1_id, 1);
assert old_a1.getPersons().size() == 1;
assert old_a1.getPersons().contains(p);

old_a2 = ar.find(Address.class, a2_id, 1);
assert old_a2.getPersons().size() == 0;
```

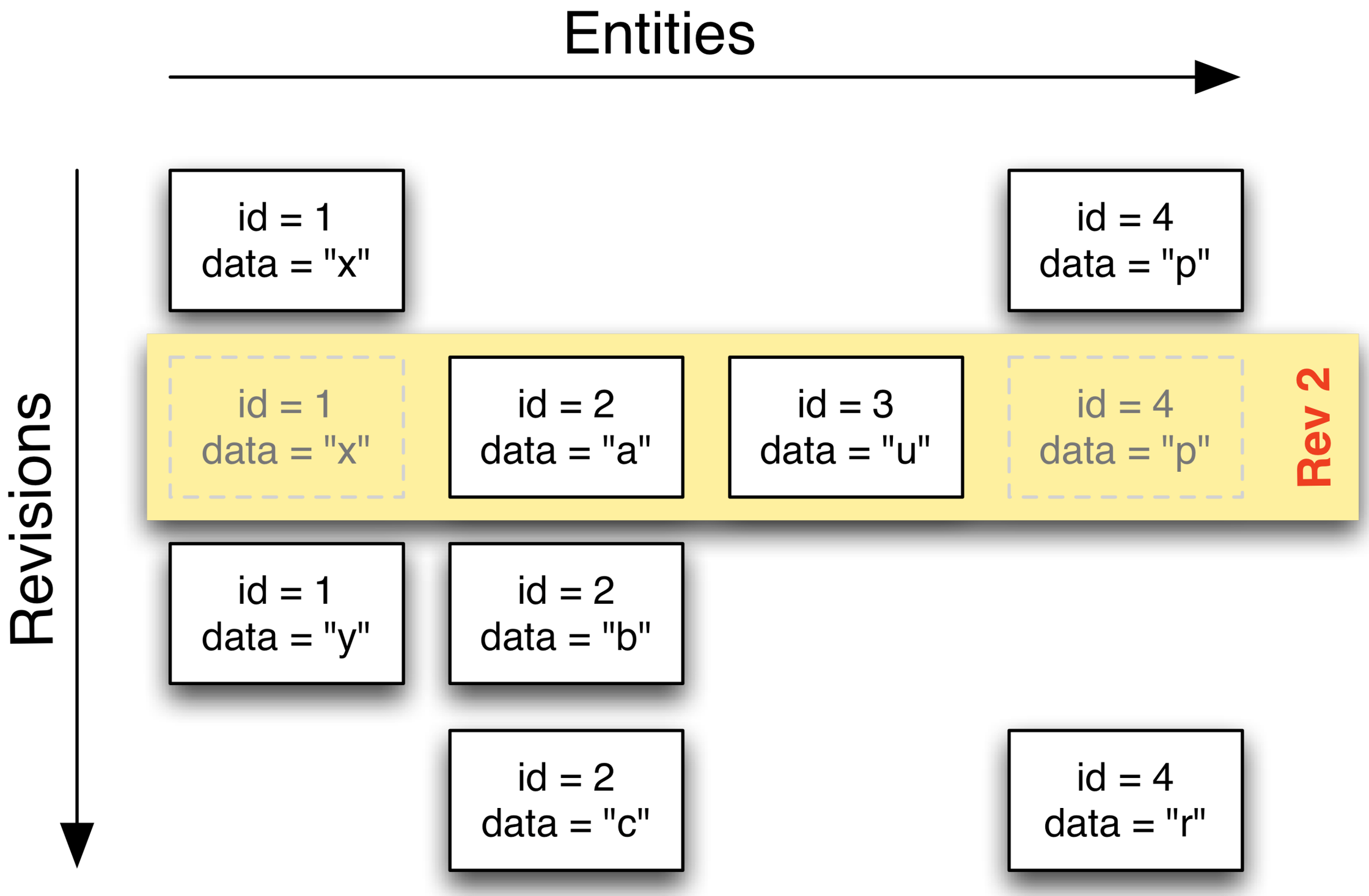
- Transparent traversing of relations: also in case of collections

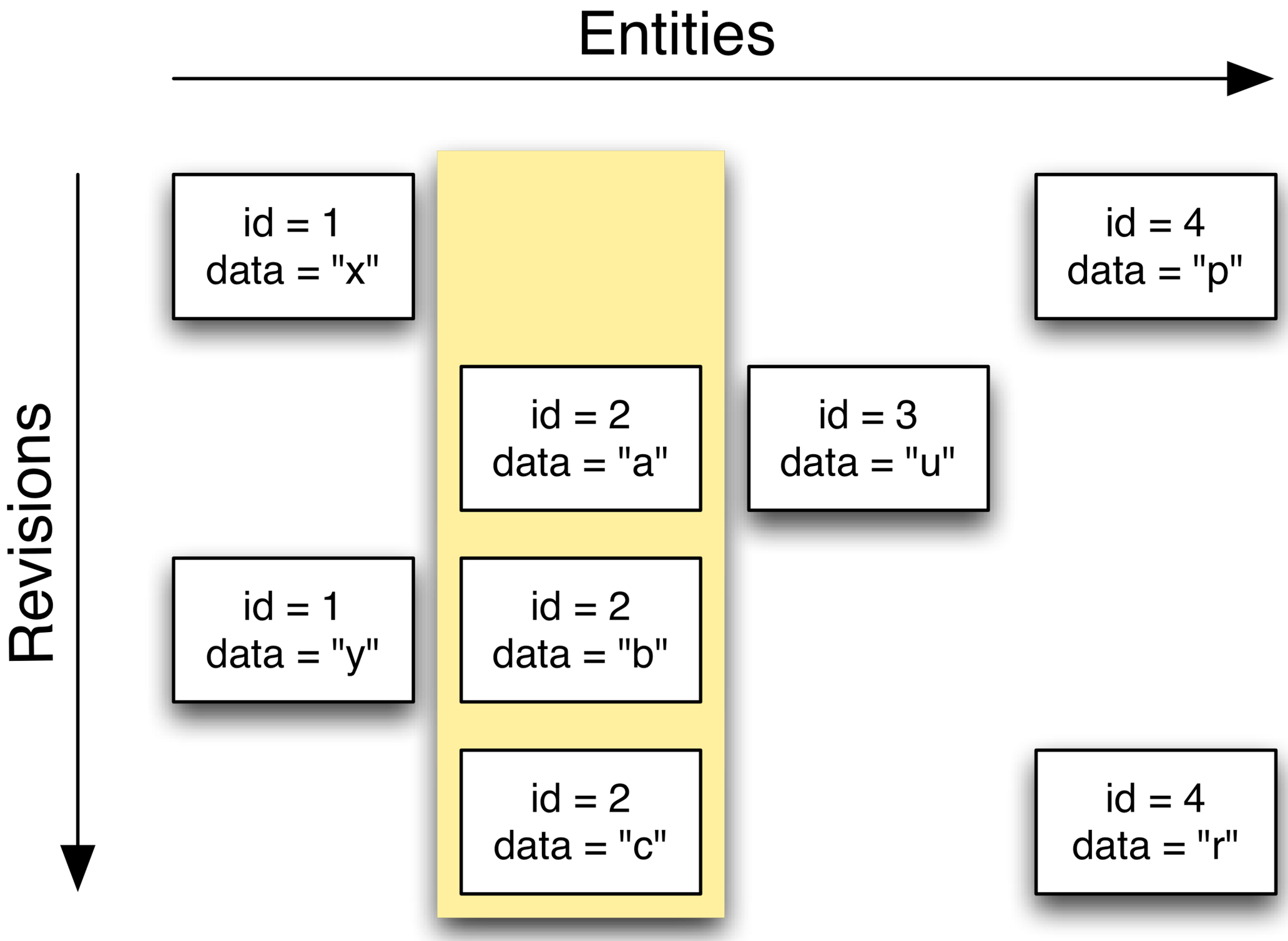
Querying

How to query historical data?


Entities
Revisions
id = 1
data = "x"
id = 4
data = "p"
id = 2
data = "a"
id = 3
data = "u"
id = 1
data = "y"
id = 2
data = "b"
id = 2
data = "c"
id = 4
data = "r"


Entities
Revisions
id = 1
data = "x"
id = 4
data = "p"
Rev 1
id = 2
data = "a"
id = 3
data = "u"
id = 1
data = "y"
id = 2
data = "b"
id = 2
data = "c"
id = 4
data = "r"


Entities
Revisions
id = 1
data = "x"
id = 4
data = "p"
id = 1
data = "x"
id = 2
data = "a"
id = 3
data = "u"
id = 4
data = "p"
Rev 2
id = 1
data = "y"
id = 2
data = "b"
id = 2
data = "c"
id = 4
data = "r"


Entities
Revisions
id = 1
data = "x"
id = 2
data = "a"
id = 3
data = "u"
id = 4
data = "p"
id = 1
data = "y"
id = 2
data = "b"
id = 2
data = "c"
id = 4
data = "r"

Querying

- Entities-at-revision
- Revisions-of-entity
- Inspired by criteria queries

```
auditReader.createQuery()
  .forRevisionsOfEntity(Person.class, false, true)
  .addProjection(AuditEntity.revisionNumber().count())
  .add(AuditEntity.id().eq(person.getId()))
  .getSingleResult()
```

Logging data for revisions

- With each revision, arbitrary data can be bound (metadata)
- For example: user making the changes
- Special entity (`@RevisionEntity`), storing the metadata
- Listener, invoked when a new revision is created

Use-cases

Envers in practice

Use case: Structured Wiki

- The power of wikis: everybody can edit
- It works well, because history is stored and viewable by everyone
- Plus, you know who made the changes

- What if the editable part of your website is more than just one textbox?
  - e.g. a set of links, images

Step 1: main entity

```
@Entity
@Audited
public class WikiPage {
   @Id @GeneratedValue private Long id;
   private String title;
   private String content;
   @CollectionOfElements private Set<String>
     links;
   @OneToMany private Set<Image> images;
}
```

Step 2: revision entity

```
@Entity
@RevisionEntity(WikiListener.class)
public class WikiRevision {
  @Id @GeneratedValue @RevisionNumber
  private Long id;
  @RevisionTimestamp private Long
   timestamp;
  @ManyToOne private User modifiedBy;
  // (...)
}
```

Step 3: revision listener

```
public class WikiListener implements
  RevisionListener {

    public void newRevision(Object revEntity) {

        WikiRevision wikiRev = (WikiRevision)
  revEntity;

        User currentUser = (User)
  Component.getInstance("currentUser");

        wikiRev.setModifiedBy(currentUser);

    }

}
```

Step 4: paginating history

```
public List getHistory(int from, int count,
  Long pageId) {

   return auditReader.createQuery()

     .forRevisionsOfEntity(WikiPage.class,
     false, true)

     .addOrder(revisionNumber().desc())

     .add(id().eq(pageId))

     .setFirstResult(from)

     .setMaxResults(count)

     .getResultList();

}
```

Changes by user

```
public List getChangesByUser(User user) {
  return auditReader.createQuery()
    .forRevisionsOfEntity(WikiPage.class, false,
    true)
    .addOrder(revisionNumber().desc())
    .add(revisionProperty("modifiedBy").eq(user))
    .getResultList();
}
```

Simple tagging

- We add a field to **`WikiPage`**: **`verified`**
- We want to find the latest verified version of a page

```
auditReader.createQuery()
.forRevisionsOfEntity(WikiPage.class, false,
true)
.add(revisionNumber().maximize()
    .add(property("verified").eq(true))
    .add(id().eq(pageId)))
.getResultList();
```

Practical

How to use Envers?

- Works everywhere where Hibernate works
  - Standalone, Webapps, Seam, Spring, ...
- Formerly a stand-alone project
  - 1.1.0.GA released in October
- Now part of Hibernate
  - Next release in 1-2 months

How to use Envers?

- Download from:

http://www.jboss.org/envers

- Also in JBoss Snapshot repository:

http://www.jboss.org/community/docs/DOC-11381

Your data is safe!

Person:

Id	Name	Surname
123	John	Smith
857	Brad	Pitt
698	Mary	Doe
	...	

Person_AUD:

Rev number	Id	Name	Surname	Rev type
64	123	James	Smith	**ADD**
64	857	Brad	Pitt	**ADD**
85	123	Peter	Smith	**MOD**
90	501	Arnold	Schwarzenegger	**DEL**
90	123	John	Smith	**MOD**

Performance

- Must be slower than without auditing:
  - 1 insert for each modified entity
  - 1 insert for each transaction

MySQL 5.1.30 (InnoDB)	5000 inserts	1000 complex inserts	5000 updates
Not audited	6.307s	6.622s	8.487s
Audited	9.807s	12.758s	11.444s
Difference	**x 1.55**	**x 1.92**	**x 1.34**

Envers overall

- Unchanged mapping (not intrusive)
- Unchanged code (transparent)
- Straightforward history reading
- Deleted entities aren't gone
- Easier to use (`@Audited`)
- Data for revisions
- Queries

Future

- Support for other Hibernate-specific mappings
  - relations in components
  - collections of components

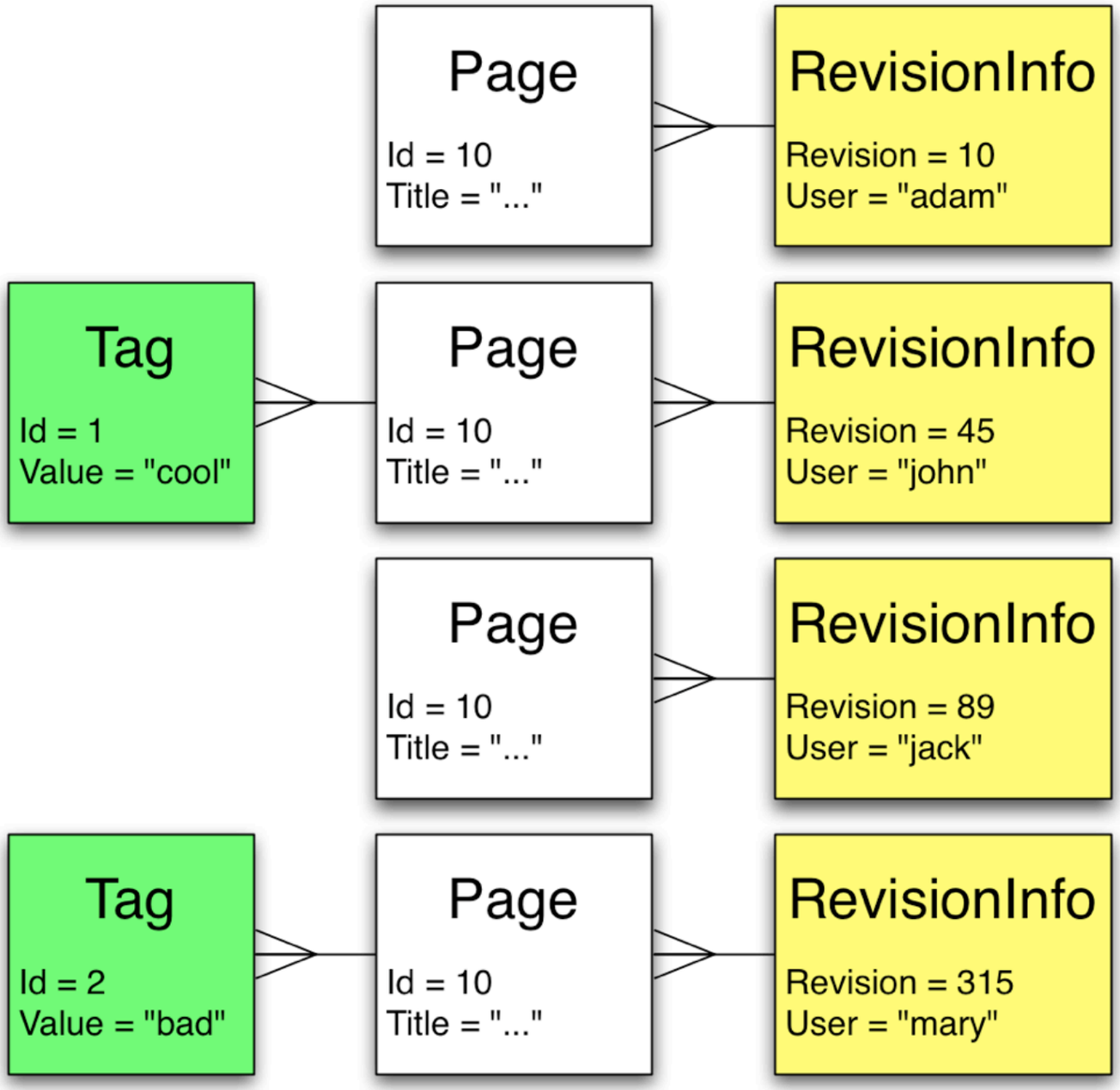
- JPA2

Future

- Tools
  - import
  - revert
  - branch
- DIFF
- Different auditing strategies
  - storing only fields

Future

- Tagging
  - a tag can be anything
  - only one revision can be tagged (many?)
- find entity by tag



Thank you for your attention!

http://www.jboss.org/envers/
adam@warski.org