

# Transparent Object Persistence (with FLOW3)

Karsten Dambekalns <[karsten@typo3.org](mailto:karsten@typo3.org)>

Old school persistence

DDD persistence

FLOW3 persistence

# An example to use...

- ☞ Let's use a Blog
- ☞ A Blog projects has
  - a Blog
  - some Posts,
  - Comments and
  - Tags
- ☞ Easy enough, eh?

# Setting the stage

- ☞ A blog is requested
- ☞ We want to show posts
- ☞ We have data somewhere
- ☞ We use MVC and have a view waiting
- ☞ We only need to hand over the posts

# Blog tables^Wrelations

blogs

uid	name
1	FLOW3

posts

uid	blog_id	title	author	text	date
1	1	FLOW3	R. Lemke	Mēs esam vissforšākie, tāpēc ka...	2008-10-07

comments

uid	post_id	author	text
1	1	Kasper Skårhøj	Nice writeup, but...

posts\_tags

post_id	tag_id
1	1
1	3

tags

uid	name
1	PHP
2	DDD
3	FLOW3

# Straight DB usage

```
$row = mysql_fetch_assoc(
    mysql_query('SELECT uid FROM blogs WHERE name = \'' .
                mysql_real_escape_string($blog) .
                '\''')
);
$blogId = $row['uid'];

$posts = array();
$res = mysql_query('SELECT * FROM posts WHERE blog_id = ' . $blogId);

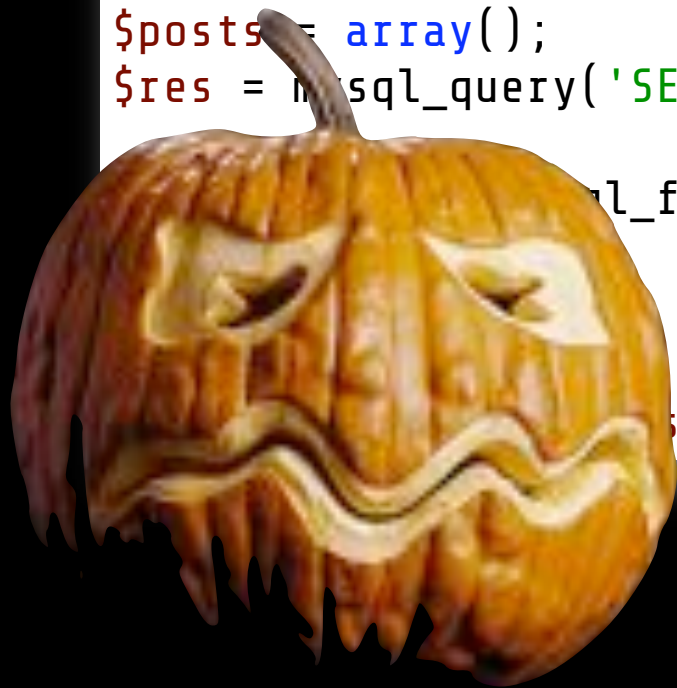
while (($row = mysql_fetch_assoc($res)) !== FALSE) {
    $posts[] = $row;
}

$view->setPosts($posts);
```

# Straight DB usage

```
$row = mysql_fetch_assoc(  
    mysql_query('SELECT uid FROM blogs WHERE name = \'' .  
                mysql_real_escape_string($blog) .  
                '\'  
    );  
$blogId = $row['uid'];
```

```
$posts = array();  
$res = mysql_query('SELECT * FROM posts  
mysql_fetch_assoc($res))
```



- SQL injection!
- clumsy code
- other RDBMS
- field changes?

# DBAL in TYPO3v4

```
$rows = $GLOBALS['TYPO3_DB']->exec_SELECTgetRows('uid', 'blogs', 'name = ' .  
        $GLOBALS['TYPO3_DB']->fullQuoteStr($blog, 'blogs'));  
$blogId = $rows[0]['uid'];  
  
$posts = $GLOBALS['TYPO3_DB']->exec_SELECTgetRows('*', 'posts', 'blog_id = ' .  
        $blogId);  
  
$view->setPosts($posts);
```



# DBAL in TYPO3v4

```
$rows = $GLOBALS['TYPO3_DB']->exec_SELECTgetRows('uid', 'blogs', 'name = ' .  
        $GLOBALS['TYPO3_DB']->fullQuoteStr($blog, 'blogs'));  
$blogId = $rows[0]['uid'];  
  
$posts = $GLOBALS['TYPO3_DB']->exec_SELECTgetRows('*', 'posts', 'blog_id = ' .  
        $blogId);  
  
$view->setPosts($posts);
```



- SQL injection!
- still clumsy...
- field changes?

# ORM tools

- ☛ ORM means Object-Relational Mapping
- ☛ Objects represent a database row
- ☛ Most tools are rather ROM – still focus on relational thinking
- ☛ Implementations differ
  - ORM base classes need to be extended
  - Schema is used to generate code

# Active Record

```
$blog = new Blog($blog);  
$posts = $blog->getPosts();  
$view->setPosts($posts);
```

- very nice
- dependency on ORM tool?
- save()/delete()

# Active Record

```
$blog = new Blog($blog);  
$posts = $blog->getPosts();  
$view->setPosts($posts);
```

• very nice  
• independence

```
$post->save();  
$post->delete();
```

save() / delete()

# “DDD persistence”

- ❖ Domain Models encapsulate behavior and data
- ❖ Concerned about the problem – only
- ❖ Infrastructure is of no relevance

# Entities

- ☛ Identity is important
- ☛ Are not defined by their attributes
- ☛ Examples could be...
  - People
  - Blog posts



# Value objects

- ☛ Have no identity
- ☛ Their value is of importance
- ☛ Are interchangeable
- ☛ Examples could be...
  - Colors
  - Numbers
  - Tags in a blog





# Aggregates

- ❖ Cluster associated objects
- ❖ Have a boundary and a root
- ❖ The root is a specific entity
- ❖ References from outside point to the root



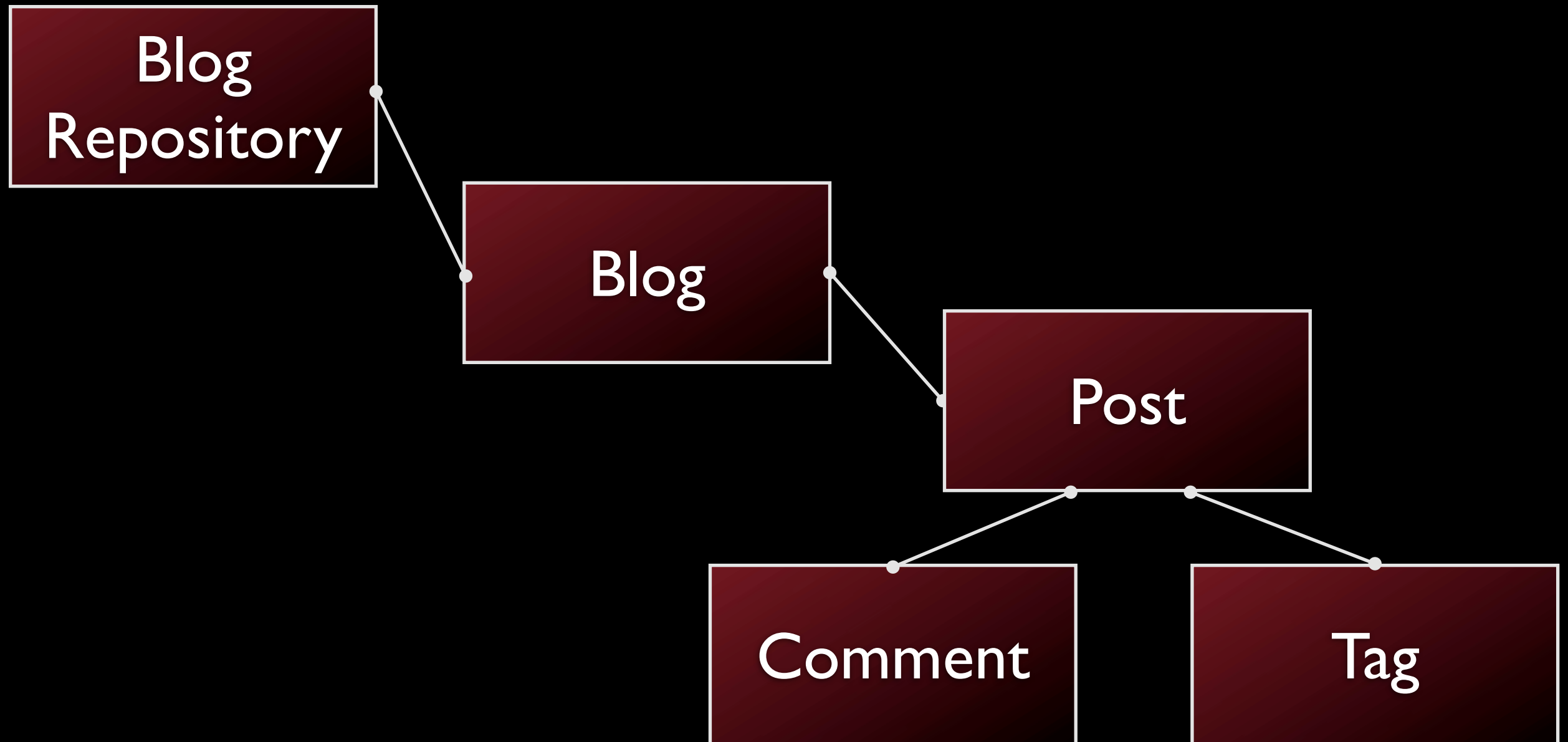


# Repositories

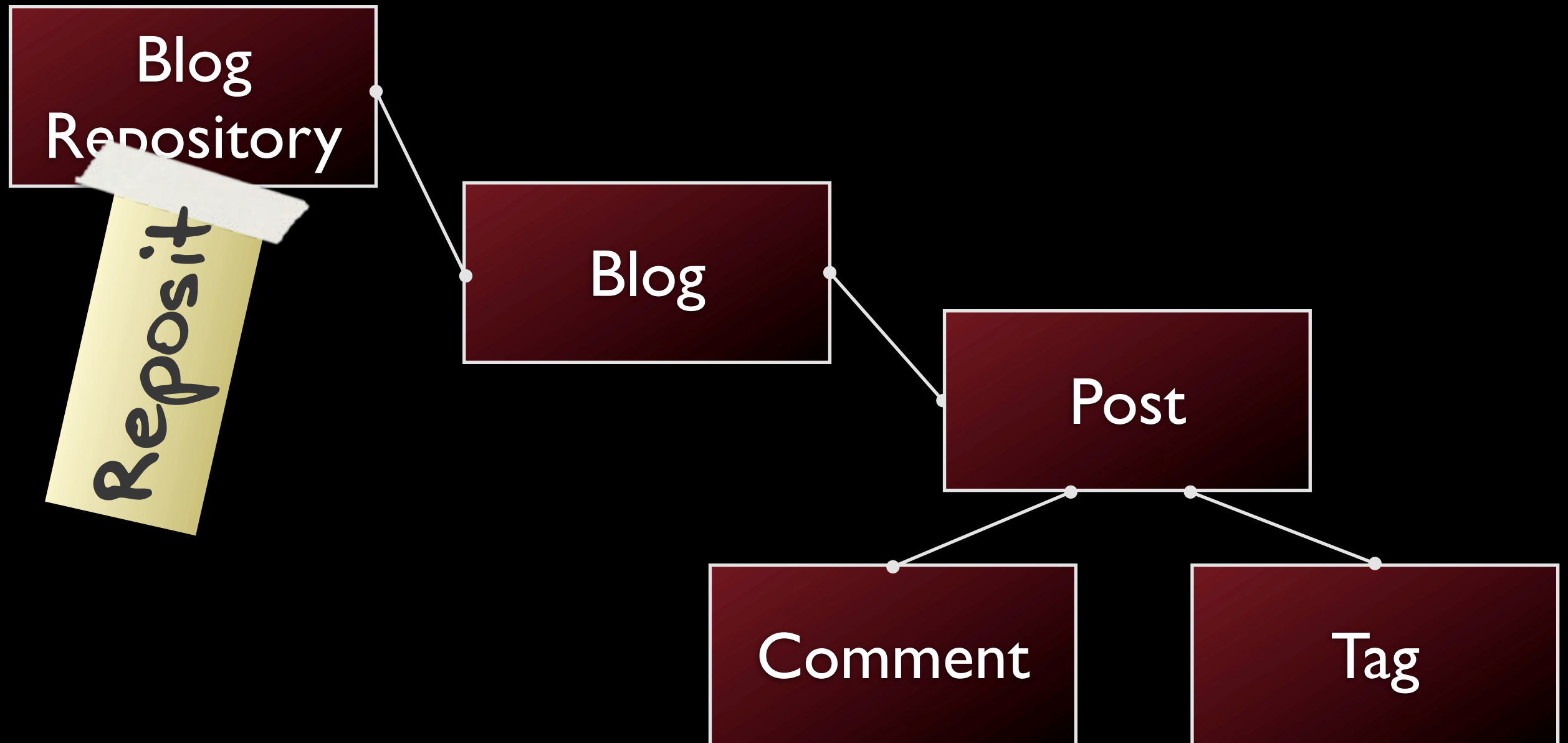
- ❖ Provide access to aggregates and entities
- ❖ Allow to find a starting point for traversal
- ❖ Persistent objects can be searched for
- ❖ Queries can be built in various ways
- ❖ Handle storage of additions and updates



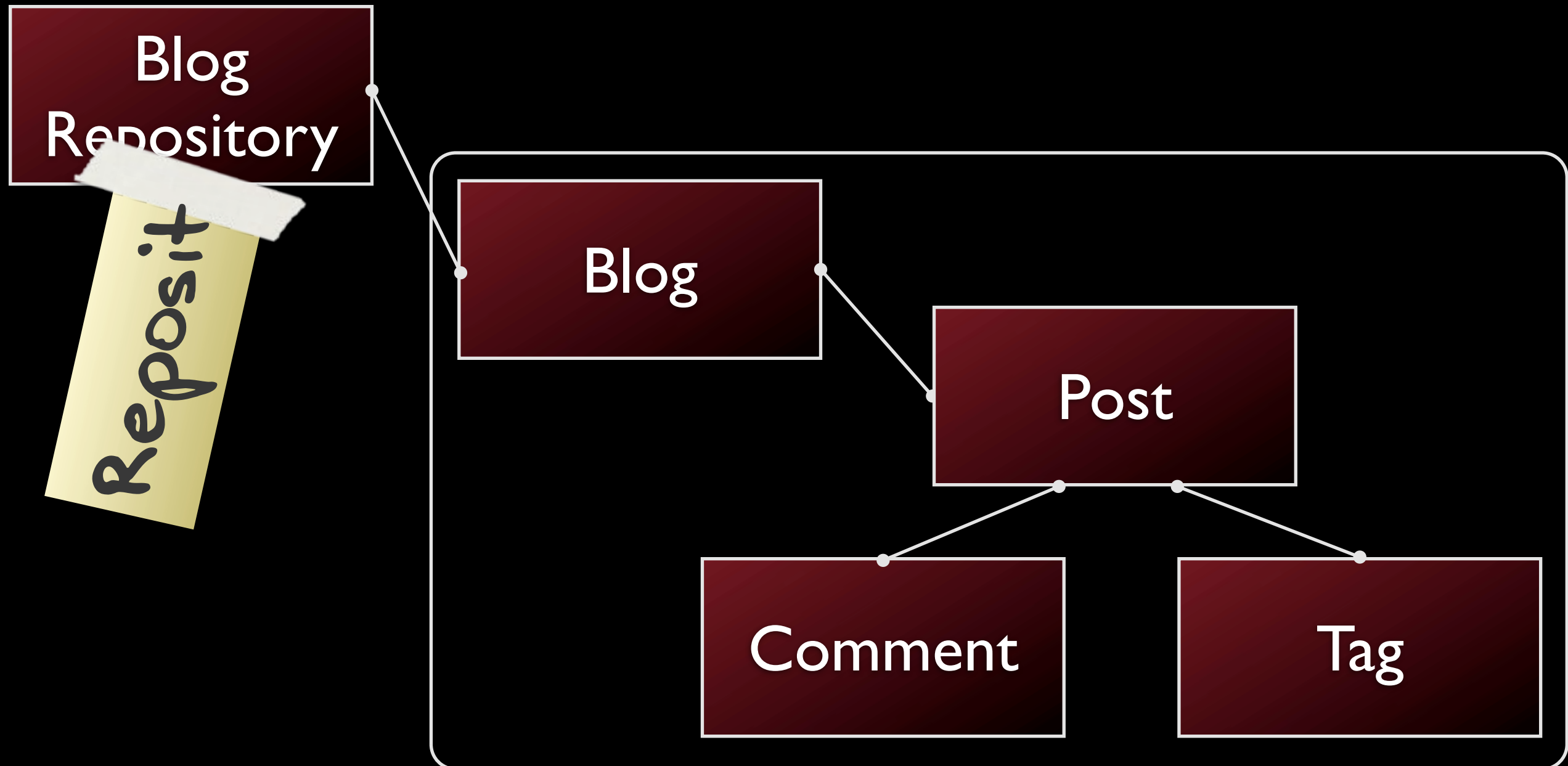
# A Domain Model



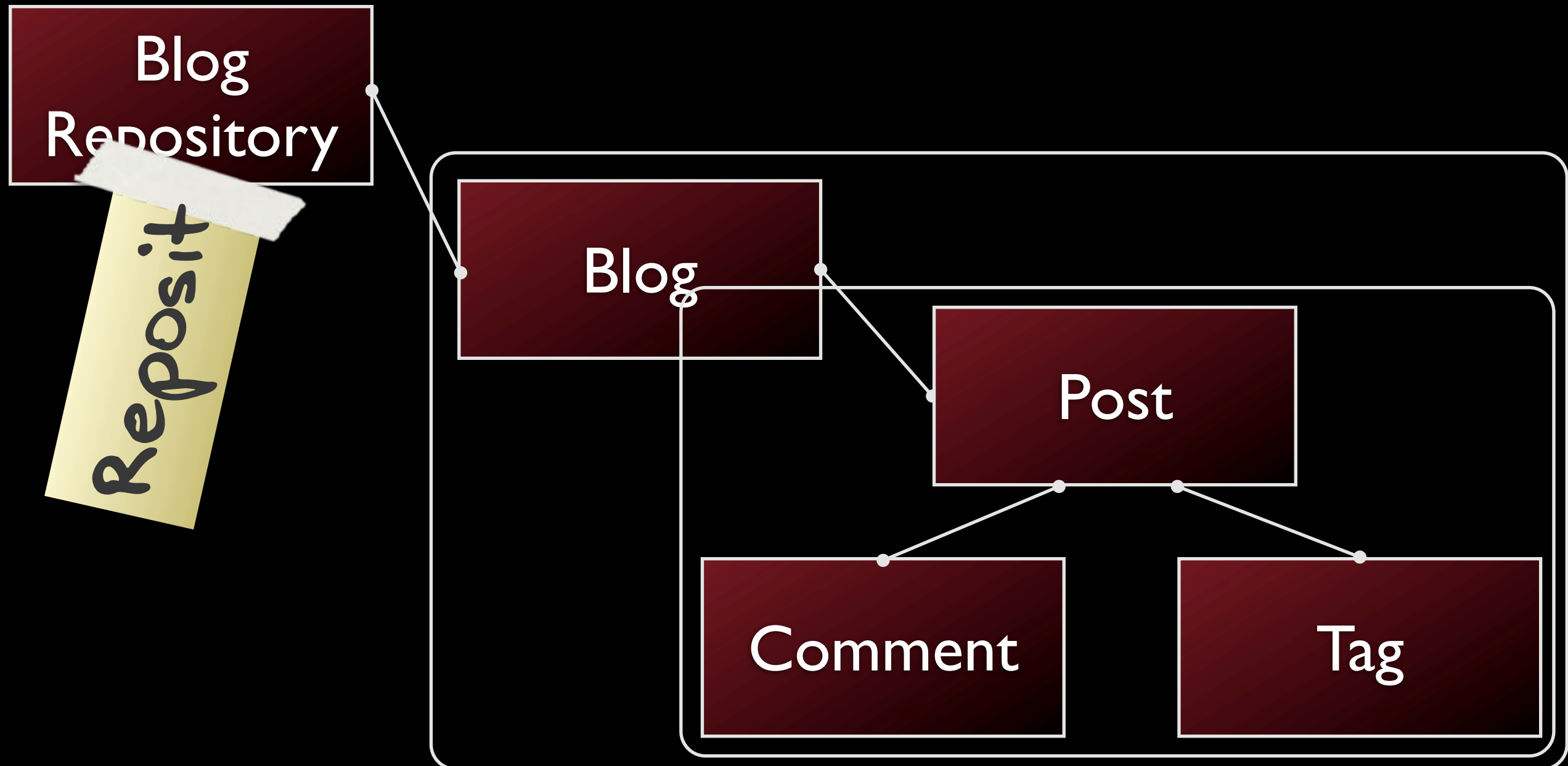
# A Domain Model



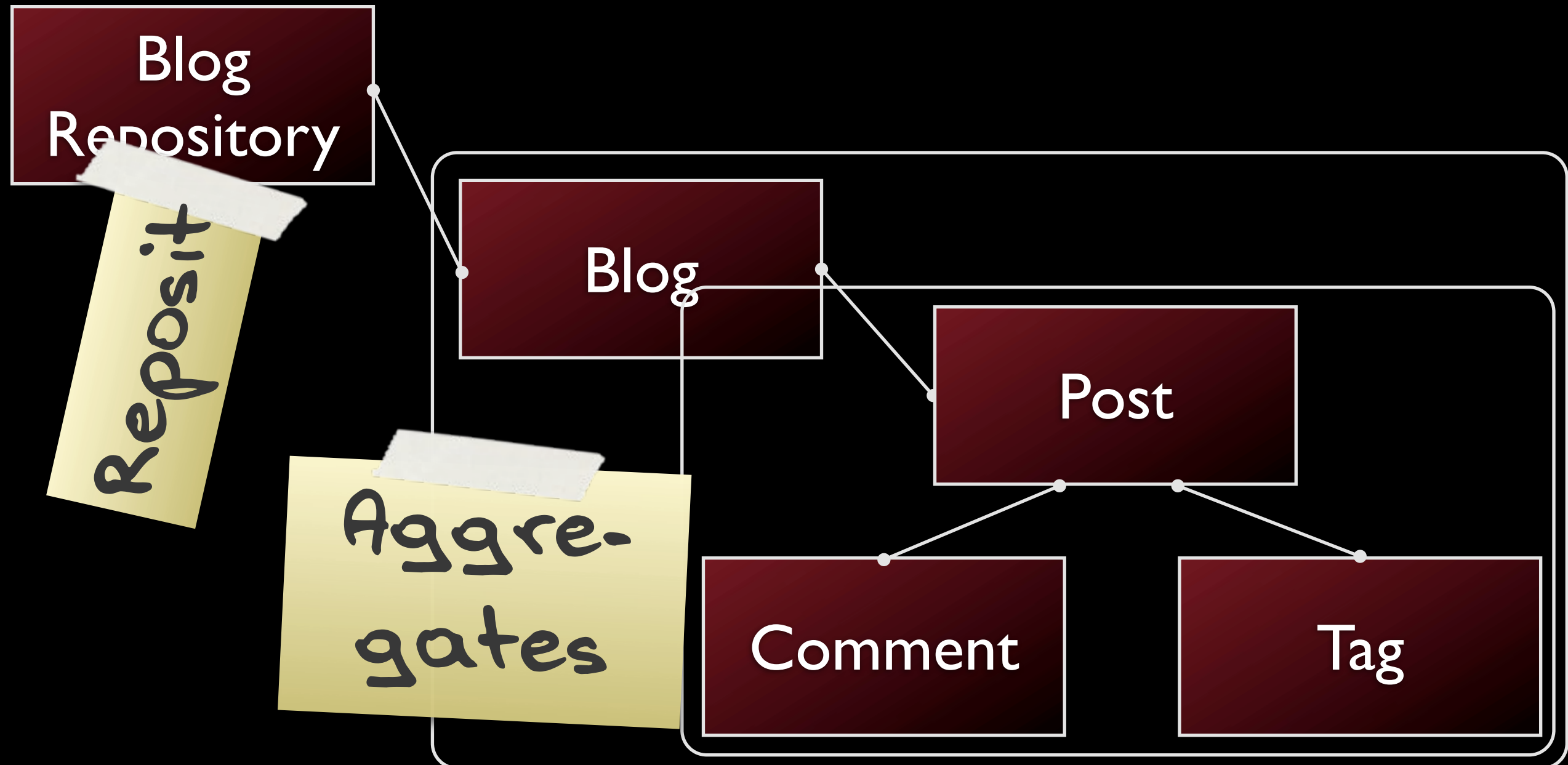
# A Domain Model



# A Domain Model

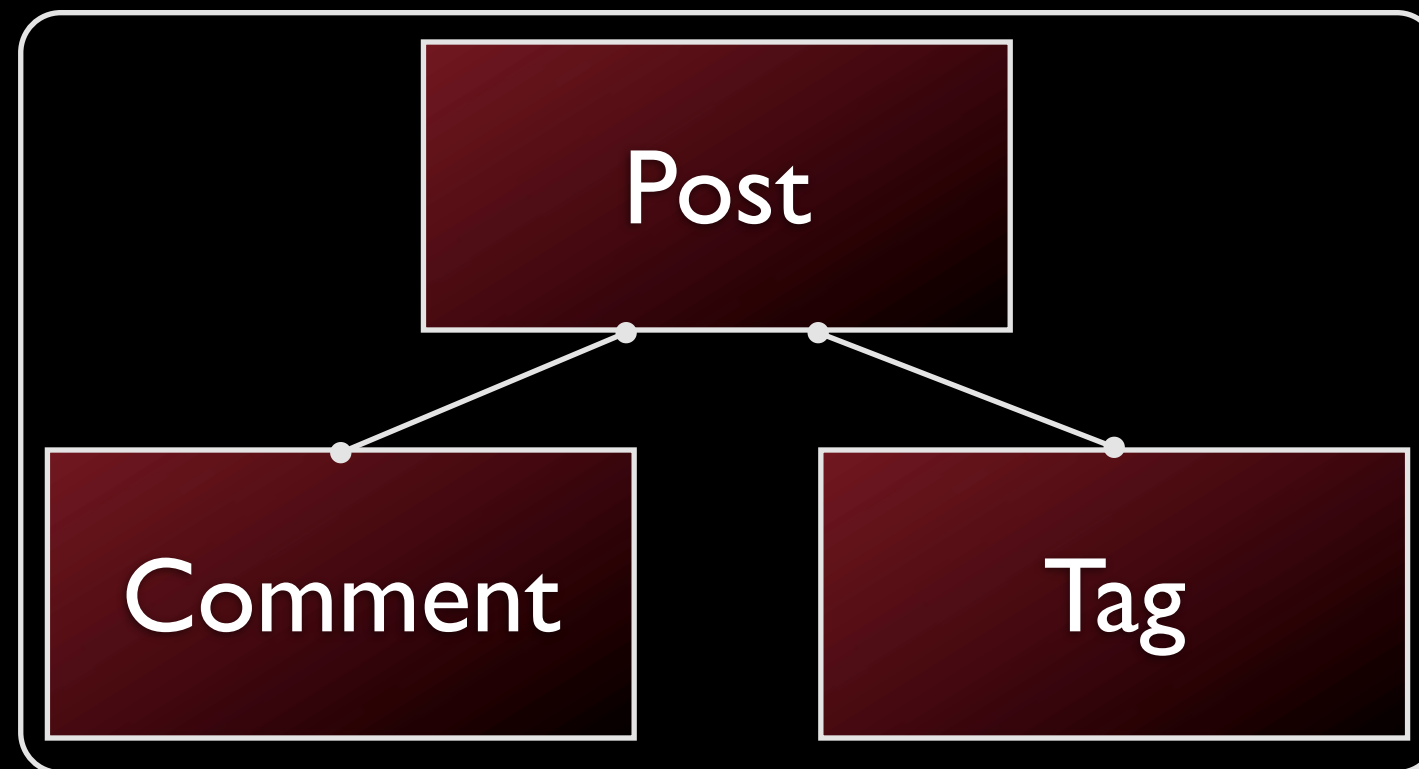


# A Domain Model

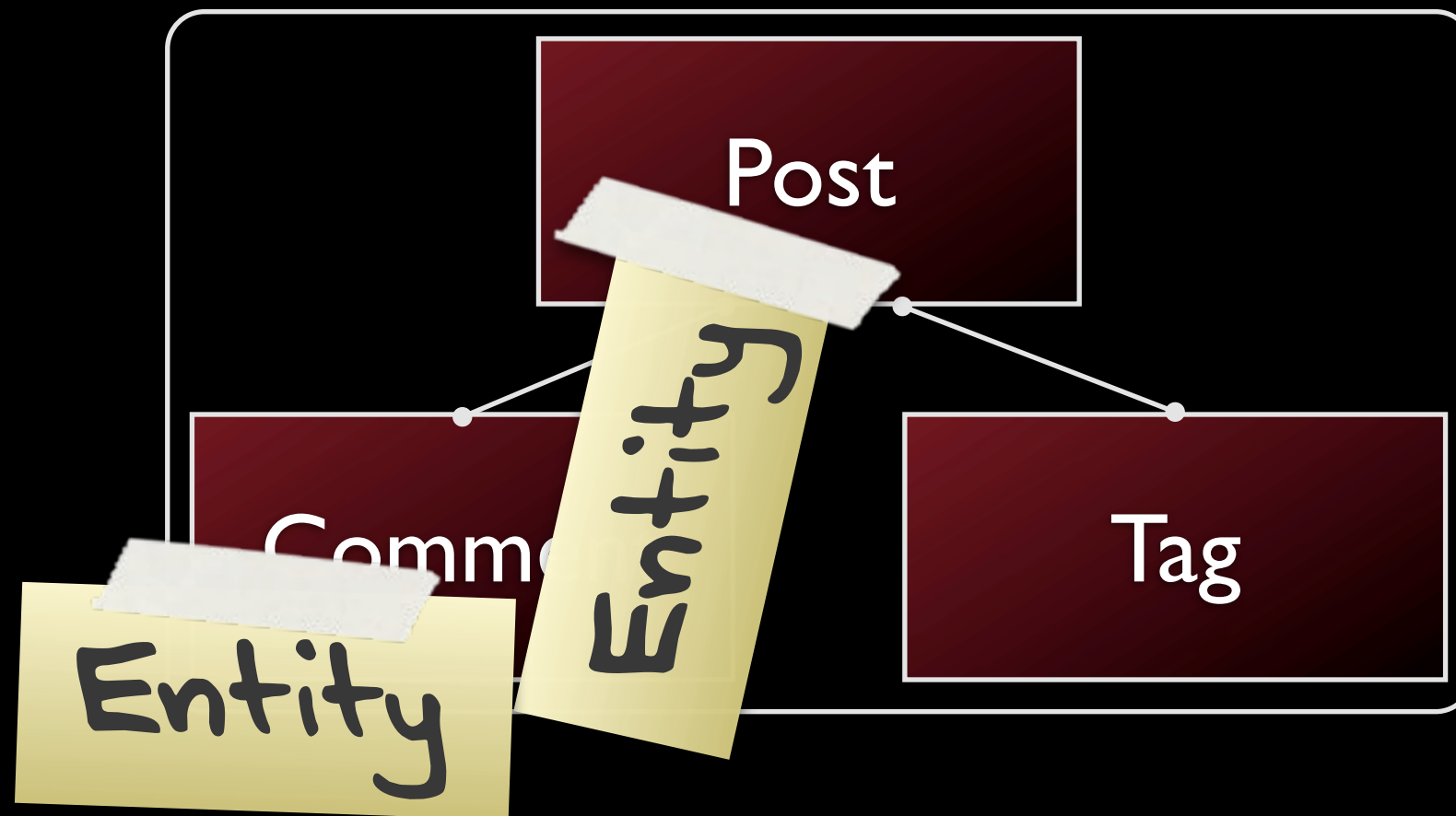




# A Domain Model

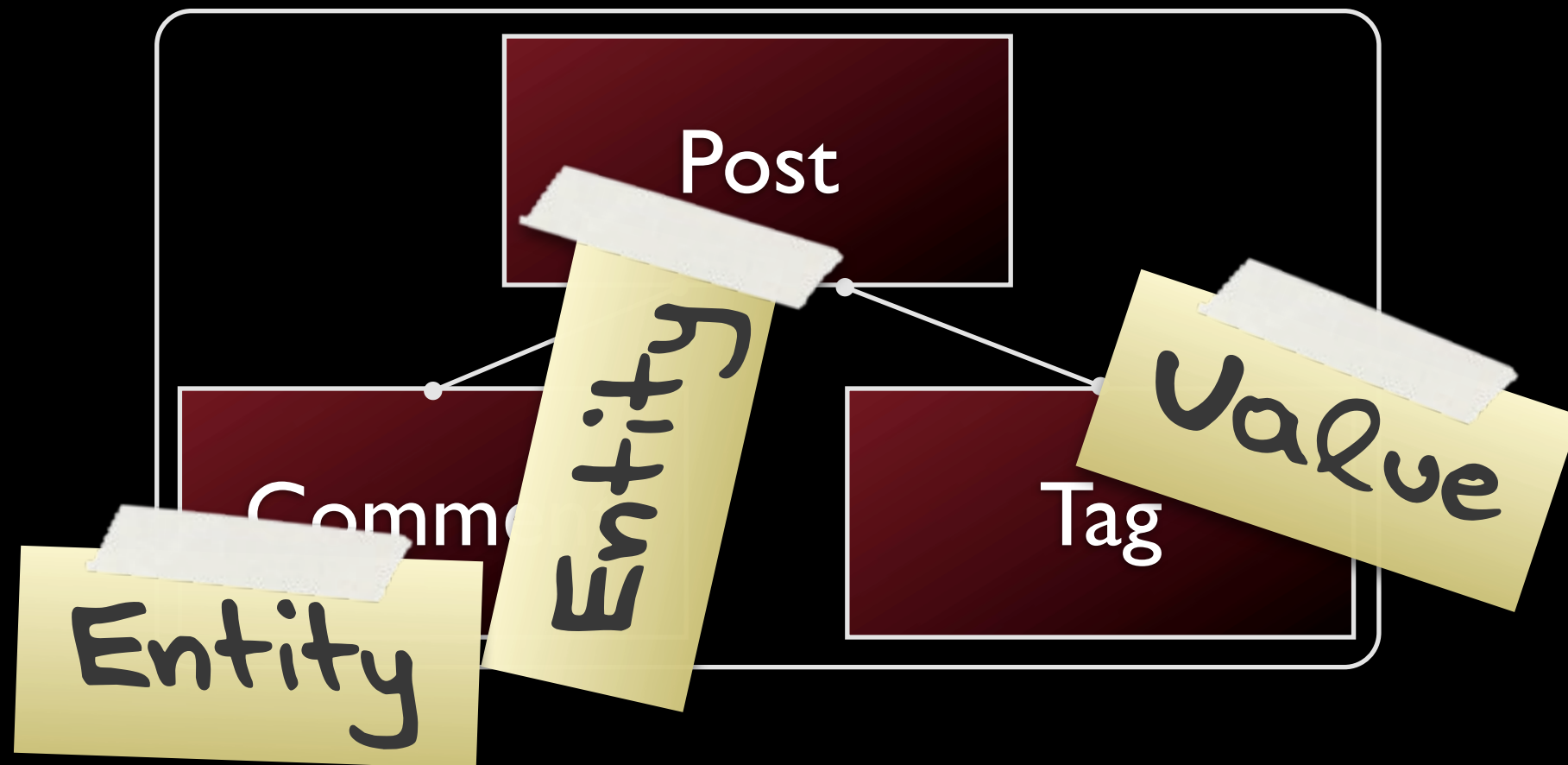


# A Domain Model





# A Domain Model



# Implementing this...

# must be a lot of work!

# Using FLOW3?

# Using FLOW3?



# Using FLOW3...

- ☛ You just implement the model
- ☛ No need to care about persisting your model
- ☛ FLOW3 handles this for you – transparently
- ☛ Even a Repository only needs little work
- ☛ Define relevant metadata in the source file

# The Blog class

```
class Blog {  
  
    /**  
     * @var string  
     */  
    protected $name;  
  
    /**  
     * @var array  
     * @reference  
     */  
    protected $posts = array();  
  
    /**  
     * Constructs this blog  
     *  
     * @param string $name Name of this blog  
     * @return  
     */  
    public function __construct($name) {  
        $this->name = $name;  
    }  
}
```

# The Blog class

```
/**
 * Adds a post to this blog
 *
 * @param F3::Blog::Domain::Post $post
 * @return void
 * @author Karsten Dambekalns <karsten@typo3.org>
 */
public function addPost(F3::Blog::Domain::Post $post) {
    $this->posts[] = $post;
}

/**
 * Returns all posts in this blog
 *
 * @return array of F3::Blog::Domain::Post
 * @author Karsten Dambekalns <karsten@typo3.org>
 */
public function getPosts() {
    return $this->posts;
}
```

# The Blog class

```
/**
 * Returns the latest $count posts from the blog
 *
 * @param integer $count
 * @return array of F3::Blog::Domain::Post
 * @author Karsten Dambekalns <karsten@typo3.org>
 */
public function getLatestPosts($count = 5) {
    return array_slice($this->posts, -$count, $count, TRUE);
}

/**
 * Returns posts posts by tag
 *
 * @param string $tag
 * @return array of F3::Blog::Domain::Post
 * @author Bastian Waidelich <bastian@typo3.org>
 */
public function findPostsByTag($tag) {
    ...
}
```



# The Post class

```
class Post {  
  
    /**  
     * @var string UUID  
     * @identifier  
     */  
    protected $identifier;  
  
    /**  
     * @var string  
     */  
    protected $title;  
  
    /**  
     * @var array  
     * @reference  
     */  
    protected $tags = array();  
}
```

# The Post class

```
/**
 * Constructs this post
 *
 * @author Robert Lemke <robert@typo3.org>
 * @author Bastian Waidelich <bastian@typo3.org>
 */
public function __construct() {
    $this->date = new DateTime();
    $this->identifier = F3::FLOW3::Utility::Algorithms::generateUUID();
}

/**
 * Adds a comment to this post
 *
 * @param F3::Blog::Domain::Comment $comment
 * @return void
 * @author Robert Lemke <robert@typo3.org>
 */
public function addComment(F3::Blog::Domain::Comment $comment) {
    $this->comments[] = $comment;
}
```

# The Comment class

```
class Comment {  
  
    /**  
     * @var string  
     */  
    protected $author;  
  
    /**  
     * @var string  
     */  
    protected $content;  
  
    /**  
     * Constructs this comment  
     *  
     * @author Karsten Dambekalns <karsten@typo3.org>  
     */  
    public function __construct() {  
        $this->date = new DateTime();  
    }  
}
```

# The Tag class

```
class Tag {  
  
    /**  
     * @var string  
     */  
    protected $name;  
  
    /**  
     * Setter for name  
     *  
     * @param string $name  
     * @return void  
     * @author Karsten Dambekalns <karsten@typo3.org>  
     */  
    public function setName($name) {  
        $this->name = $name;  
    }  
}
```

# The BlogRepository

- ☞ Now this really must be a complex piece of code, no?
- ☞ One word: **No**


# The BlogRepository

- ☞ Now this really must be a complex piece of code, no?
- ☞ One word: **No**

```
class BlogRepository extends F3::FLOW3::Persistence::Repository {  
  
    /**  
     * Returns one or more Blogs with a matching name if found.  
     *  
     * @param string $name The name to match against  
     * @return array  
     */  
    public function findByName($name) {  
        $query = $this->createQuery();  
        $blogs = $query->matching($query->equals('name', $name))->execute();  
  
        return $blogs;  
    }  
}
```

# The BlogRepository

- ☞ Now this really must be a complex piece of code, no?
- ☞ One word: **No**



```
class BlogRepository extends F3::FLOW3::Persistence::Repository {  
  
    /**  
     * Returns one or more Blogs with a matching name if found.  
     *  
     * @param string $name The name to match against  
     * @return array  
     */  
    function findByName($name) {  
        $query = $this->createQuery();  
        $blogs = $query->matching($query->equals('name', $name))->execute();  
        return $blogs;  
    }  
}
```

# @notations used

 @repository

 @entity

 @valueobject

 @var

 @transient

 @reference

 @identifier



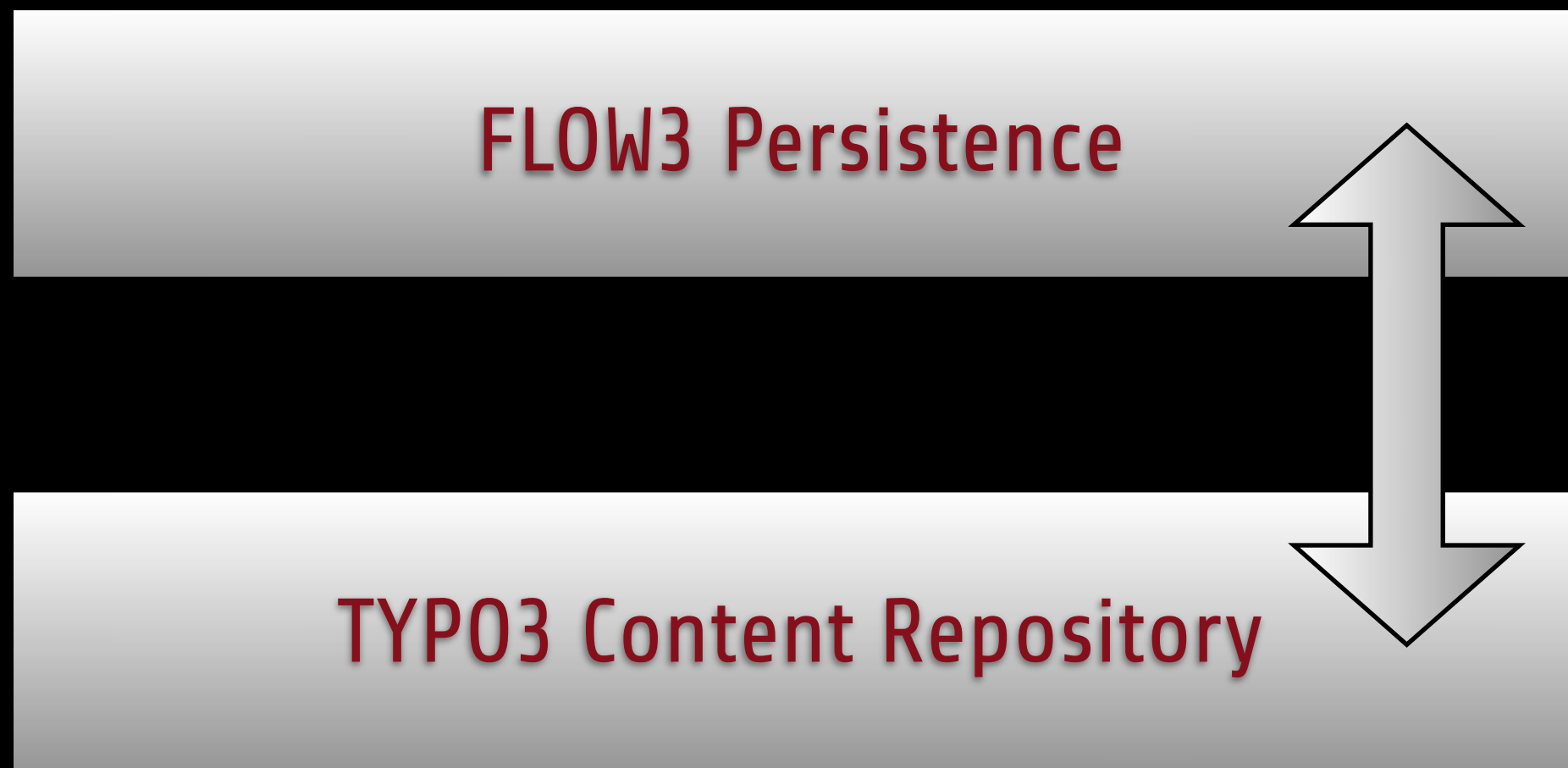
# Persistence Manager

- ☛ Mostly invisible to developers
- ☛ Manages additions of and updates to objects
- ☛ Concerned only about objects in repositories
- ☛ Collects objects and hands over to backend
- ☛ Allows for objects being persisted at any time
- ☛ Automatically called to persist at end of script run

# Simplified stack



# Simplified stack



# Transparent persistence

- ☛ Explicit support for Domain-Driven Design
- ☛ Class Schemata are defined by the Domain Model class
  - No need to write an XML or YAML schema definition
  - No need to define the database model and object model multiple times at different places
- ☛ Automatic persistence in the JSR-283 based Content Repository
- ☛ Legacy data sources can be mounted

# JSR-283 Repository

- ❖ Defines a uniform API for accessing content repositories
- ❖ A Content Repository
  - is a kind of object database for storage, search and retrieval of hierarchical data
  - provides methods for versioning, transactions and monitoring
- ❖ TYP03CR is the first working port of JSR-170 / JSR-283
- ❖ Karsten Dambekalns is member of the JSR-283 expert group

# Legacy databases

- ☛ Often you...
  - still need to access some existing RDBMS
  - need to put data somewhere for other systems to access
- ☛ The Content Repository will allow “mounting” of RDBMS tables
- ☛ Through this you can use the same persistence flow

# Legacy databases

- ☛ Often you...
  - still need to access some existing RDF
  - need to put data somewhere for other
- ☛ The Content Repository will allow “mounting”
- ☛ Through this you can use the same persistence flow



future  
fun!

# Query Factory

- ☞ Creates a query for you
- ☞ Decouples persistence layer from backend
- ☞ Must be implemented by backend
- ☞ API with one method:
  - `public function create($className);`



# Query objects

- ☛ Represent a query for an object type
- ☛ Allow criteria to be attached
- ☛ Must be implemented by backend
- ☛ Simple API
  - `execute()`
  - `matching()`
  - `equals()`, `lessThan()`, ...
  - ...

**Client code ignores  
Repository  
implementation;  
Developers do not!**

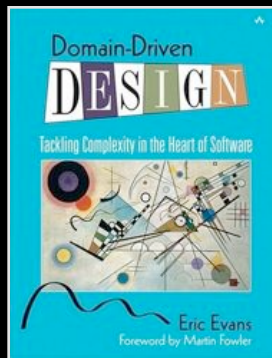
*Eric Evans*

# Usability

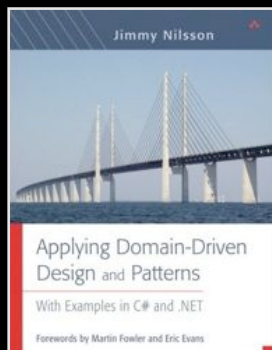
- ☛ Repositories extending FLOW3's base Repository
  - have basic methods already
  - only need to implement custom find methods
  - already have a query object set up for "their" class
- ☛ No tables, no schema, no XML, no hassle
- ☛ No save calls, no fiddling with hierarchies
- ☛ Ready-to-use objects returned

# Questions!

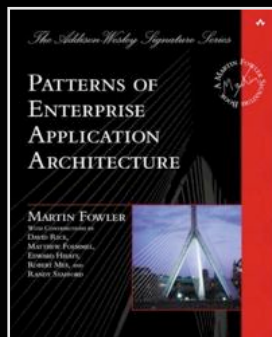
# Literature



Domain-Driven Design  
*Eric Evans, Addison-Wesley*

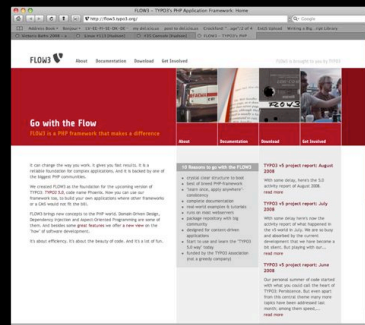


Applying Domain-Driven Design and Patterns  
*Jimmy Nilsson, Addison-Wesley*

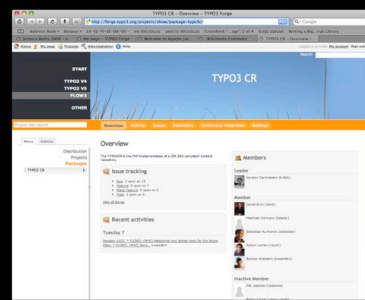


Patterns of Enterprise Application Architecture  
*Martin Fowler, Addison-Wesley*

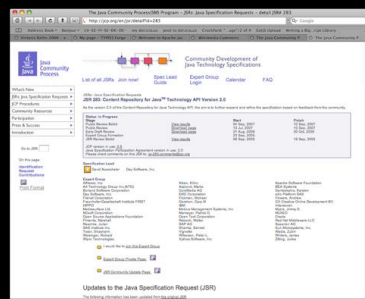
# Links



FLOW3  
<http://flow3.typo3.org/>



TYPO3CR  
<http://forge.typo3.org/projects/show/package-typo3cr>



JSR-283  
<http://jcp.org/en/jsr/detail?id=283>

TYP03



Flickr photo credits:

megapixel13 (barcode), rmrayner (ring), Ducatirider (grapes), Here's Kate (book shelf)

Pumpkin pictures:

<http://ptnoticias.com/pumpkinway/>