software pilots

TRIFORK.

Versioning your object model

Ole Friis Østergaard

RubyFools 2008

Agenda

- Why do versioning?
- Walk-through of Rails plug-ins
  - What do they do?
  - How do you use them?
  - Problems and advantages.
  - Examples.
- What not to version?

Who am I?

- Ole Friis Østergaard
- Software Pilot, Trifork A/S
- Not a native English-speaker
- Professional work: .NET, Java
- Semi-professional work, spare time: Ruby
- TDD enthusiast
  - ...though we'll do slide-driven development the next hour...

Which problems do we solve?

- The user regrets pressing that "Delete project" button.

- You want to track the changes to certain objects.

- You need to know who changed an object, in order to ask him why.

Our demo application

- Really simple and ugly.

- Create/read/update/delete projects and tasks.

- Let's have a quick look at it.

Walk-through of plug-ins

- Rails makes it easy to be creative when doing plug-ins!
- Really simple pieces of code that do exciting stuff.
- Very non-intrusive.
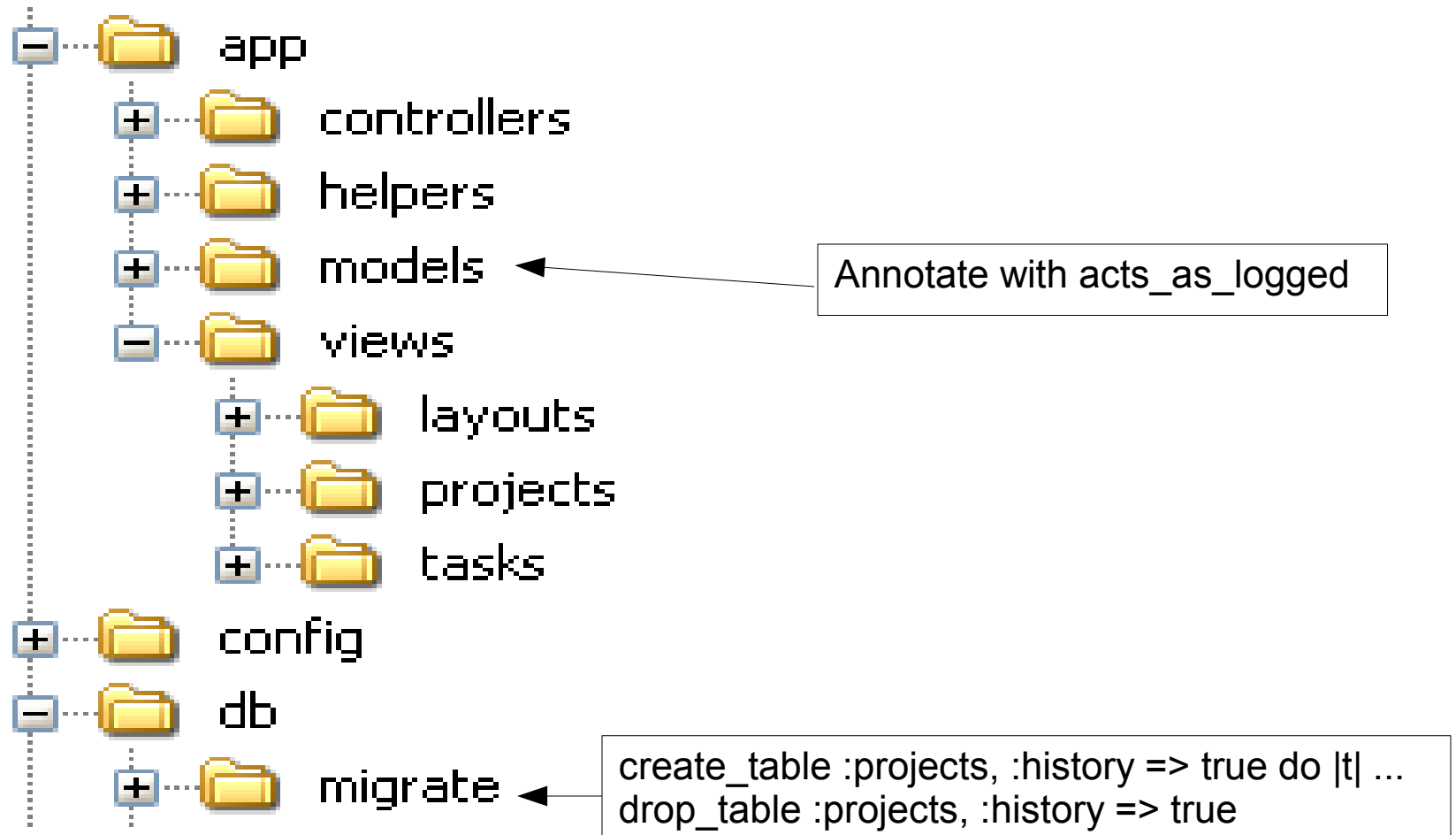- Easy to use.
- Different angles on the versioning problems.

acts_as_logged

acts_as_logged

- Makes a "shadow copy" of your normal database entries.
- Makes it easy to comply with laws concerning logging.
- Cannot show the history.
- Created by some of my colleages at Trifork.
- No official homepage (yet).
- Copy plug-in manually.

What to do?

```
app
    controllers
    helpers
    models          ← Annotate with acts_as_logged
    views
        layouts
        projects
        tasks
config
db
    migrate         ← create_table :projects, :history => true do |t| ...
                      drop_table :projects, :history => true
```
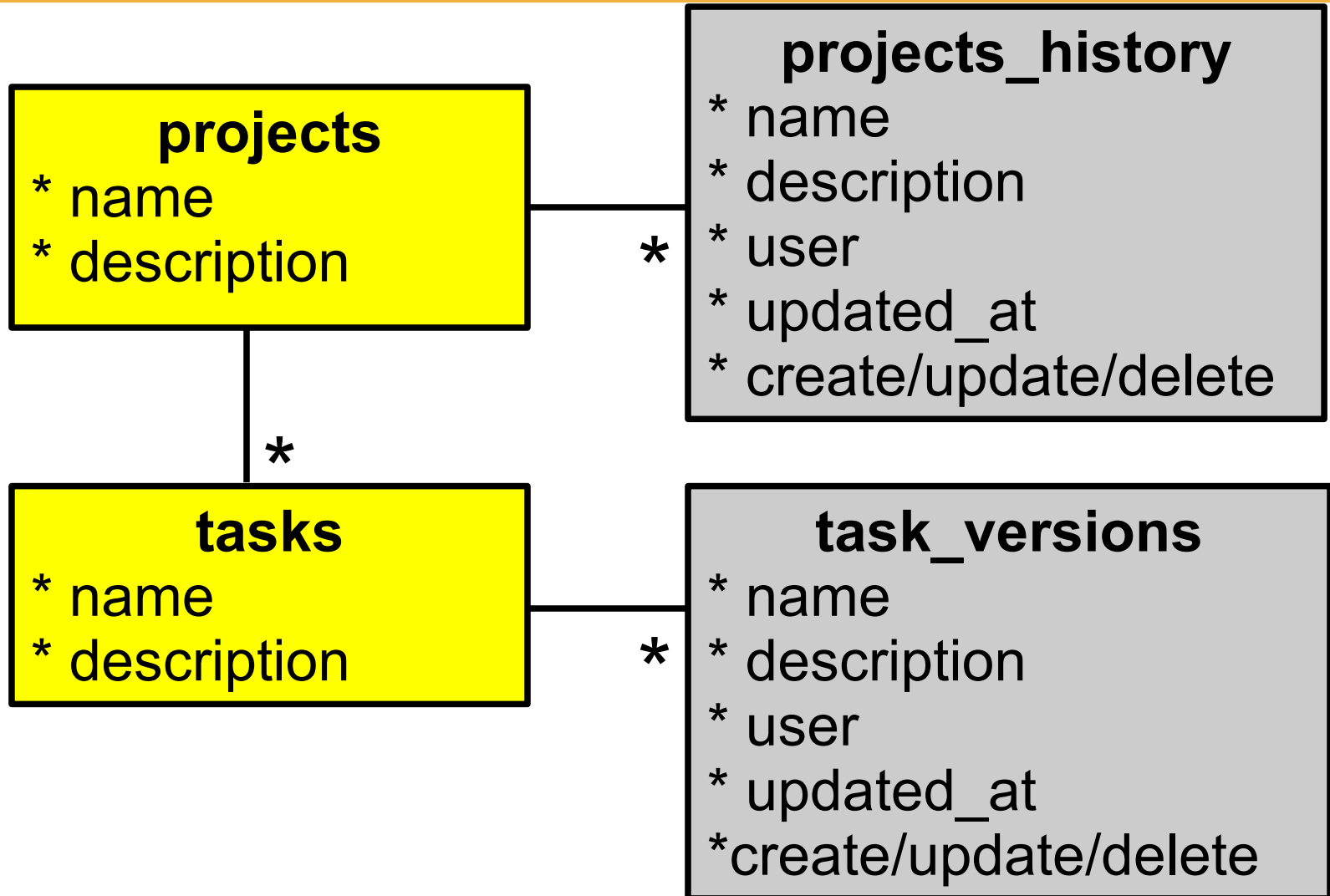
How does it work?

- Whenever a model object is saved, updated, or deleted, creates new row in the corresponding logging table.

TRIFORK.

Data model

projects
* name
* description

projects_history
* name
* description
* user
* updated_at
* create/update/delete

*

tasks
* name
* description

*

task_versions
* name
* description
* user
* updated_at
*create/update/delete

*

Problems?

- No built-in functionality to read the history items.
- New history table for each logged model class.

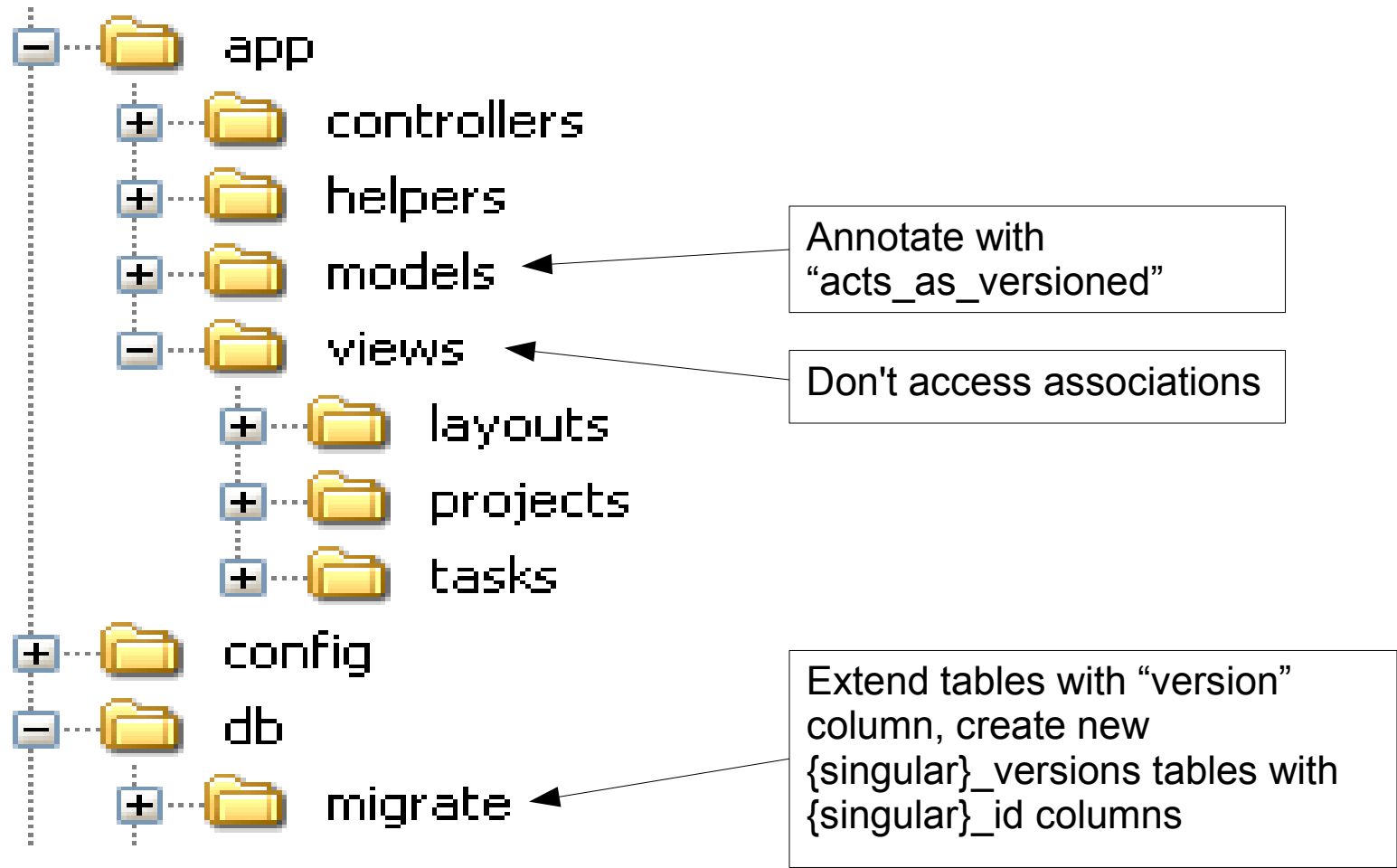# However...

- Very simple solution to simple problem.

acts_as_versioned

acts_as_versioned

- Makes "shadow copies" of rows in the database.

- Makes it possible to view these old versions.

- http://ar-versioned.rubyforge.org/

- ruby script/plugin install acts_as_versioned

TRIFORK.

What to do?

```
└─ 📁 app
    ├─ 📁 controllers
    ├─ 📁 helpers
    ├─ 📁 models
    └─ 📁 views
        ├─ 📁 layouts
        ├─ 📁 projects
        └─ 📁 tasks
└─ 📁 config
└─ 📁 db
    └─ 📁 migrate
```

Annotate with "acts_as_versioned"

Don't access associations

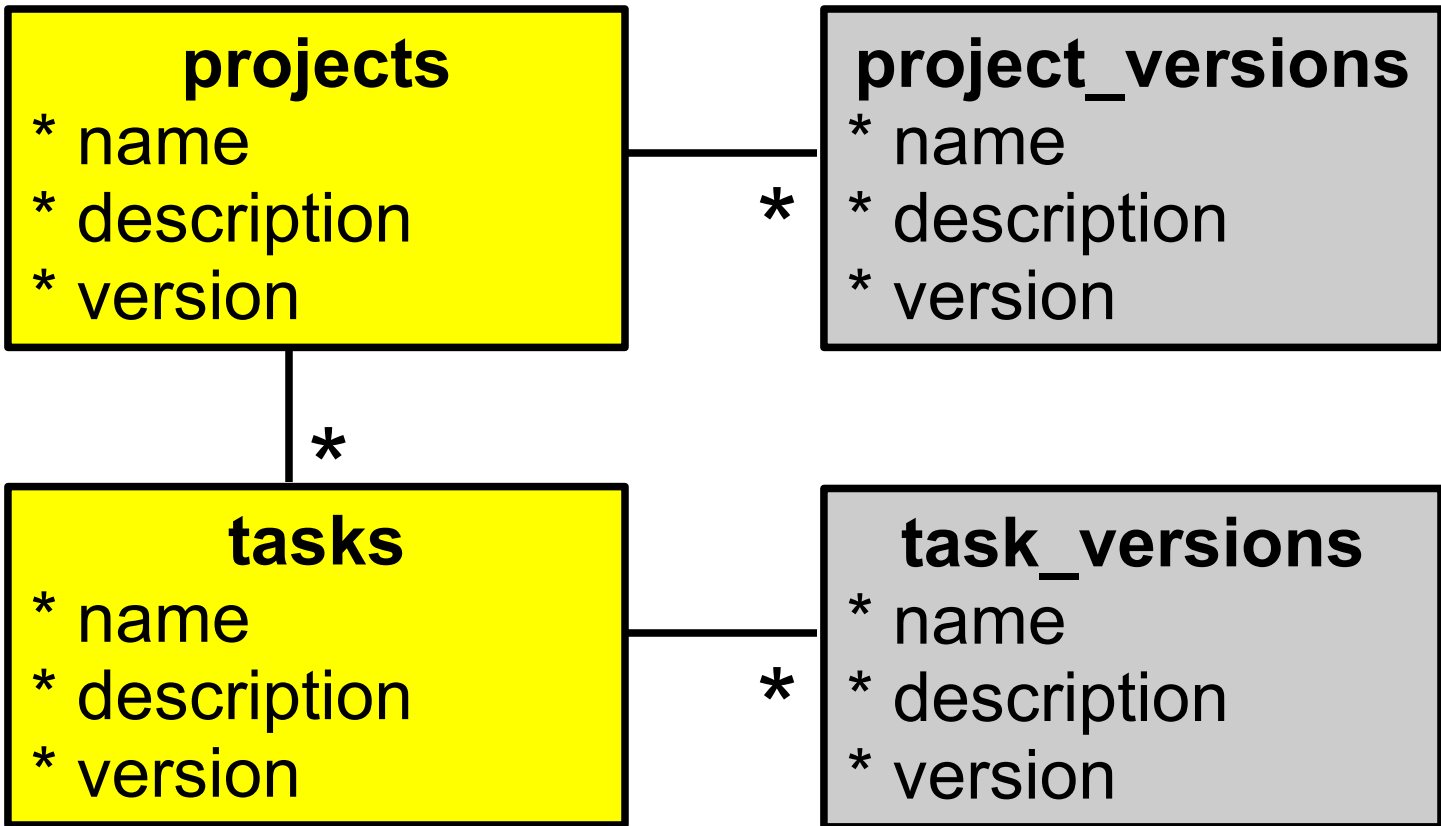Extend tables with "version" column, create new {singular}_versions tables with {singular}_id columns

How does it work?

- Whenever a model object is saved or updated, creates new row in the corresponding version table.

- When finding previous versions, instances of OriginalClass::Version are returned.

Data model

Problems?

- Versioning local to each object, i.e. associations do not work on old versions.

- Adds an extra table to each of your original tables.

- Since the old versions are given as another class, your own logic methods ("short_description", "sum_of_expenses") will not work.

TRIFORK.

However...

- Mature plug-in, has been used "in the wild".
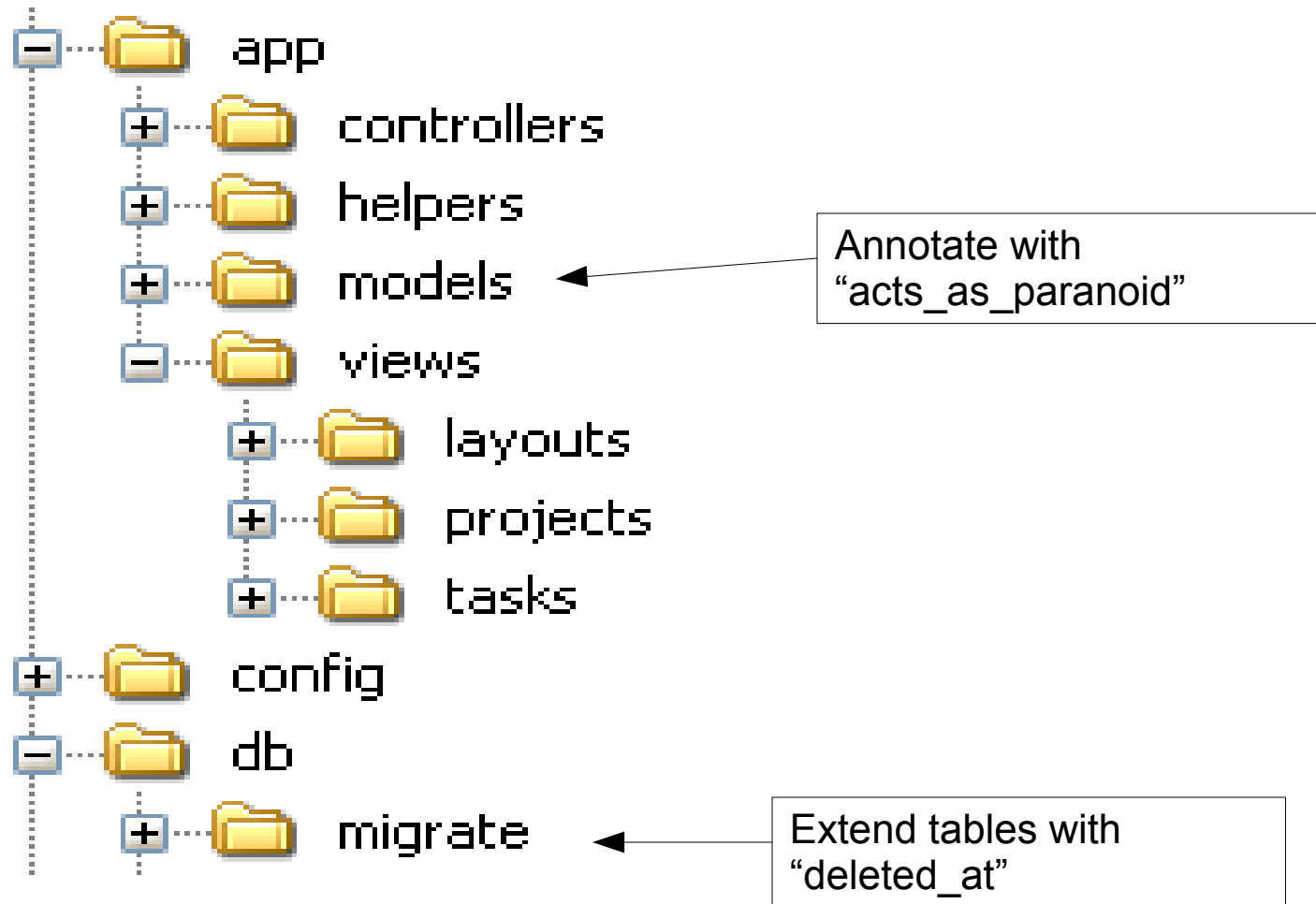- Easy to set up.
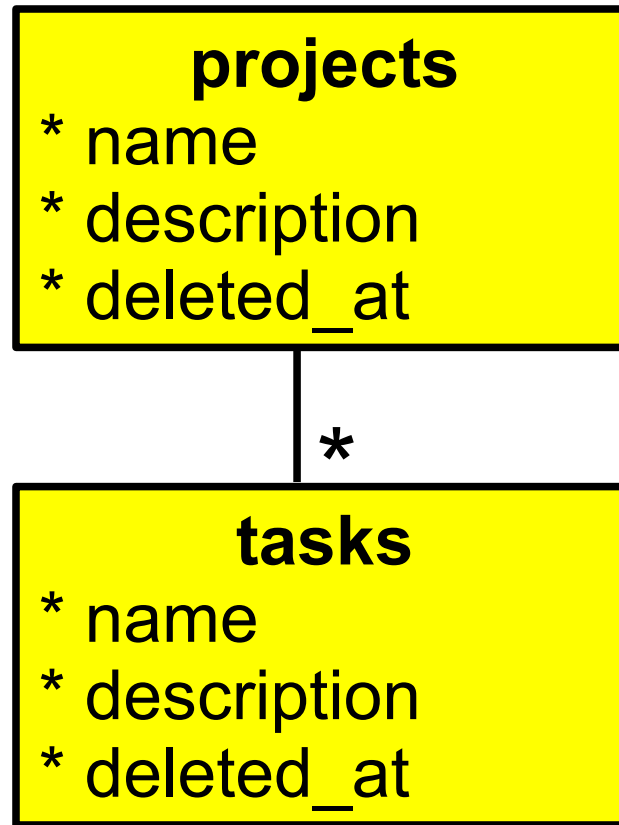- Simple concept.
- Perfect for wikis.

acts_as_paranoid

acts_as_paranoid

- Calling destroy simply sets a "deleted_at" flag.
- Therefore, deleted objects can still be found.
- Not really a versioning plug-in, but solves part of the problem.
- http://ar-paranoid.rubyforge.org/
- ruby script/plugin install acts_as_paranoid

What to do?

```
- app
  + controllers
  + helpers
  + models                ◄──── Annotate with
  - views                       "acts_as_paranoid"
    + layouts
    + projects
    + tasks
+ config
- db
  + migrate               ◄──── Extend tables with
                                "deleted_at"
```

Data model

```
┌─────────────────────┐
│      projects       │
│  * name             │
│  * description      │
│  * deleted_at       │
└──────────┬──────────┘
           │ *
┌──────────┴──────────┐
│       tasks         │
│  * name             │
│  * description      │
│  * deleted_at       │
└─────────────────────┘
```

Problems?

- Not protected against updates, only deletes.
- Queries are altered by the plug-in in order to filter result.
- Not really a versioning plug-in.

However...

- Extremely simple.
- No new tables.
- Can be used in conjunction with acts_as_versioned.

acts_as_versioned_association

acts_as_versioned_association

- Extends the functionality of acts_as_versioned.
- Annotate model objects.
- http://livsey.org/2006/8/13/major-update-to-acts_as_versioned_association/
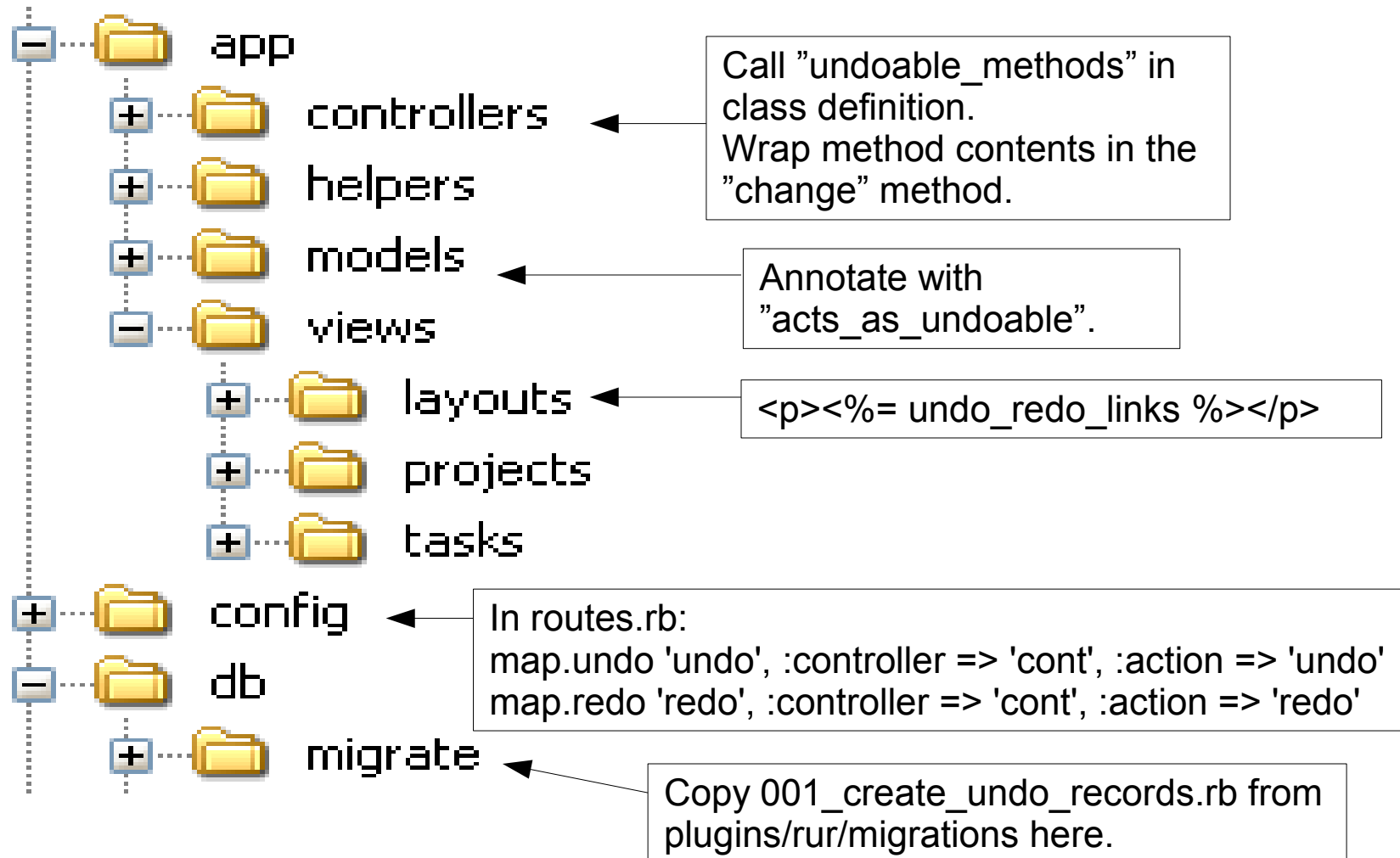- Might be cool.
- No real documentation.
- So, no details here!

Rails Undo Redo

Rails Undo Redo (rur)

- Undo/redo user actions, as known from e.g. paint programs.

- Annotate model classes and controllers.

- http://blog.nanorails.com/rails-undo-redo

- Ruby script/plugin install http://svn.nanorails.com/plugins/rur

What to do?

```
app
    controllers
    helpers
    models
    views
        layouts
        projects
        tasks
config
db
    migrate
```

Call "undoable_methods" in class definition.
Wrap method contents in the "change" method.

Annotate with "acts_as_undoable".

`<p><%= undo_redo_links %></p>`

In routes.rb:
map.undo 'undo', :controller => 'cont', :action => 'undo'
map.redo 'redo', :controller => 'cont', :action => 'redo'

Copy 001_create_undo_records.rb from plugins/rur/migrations here.
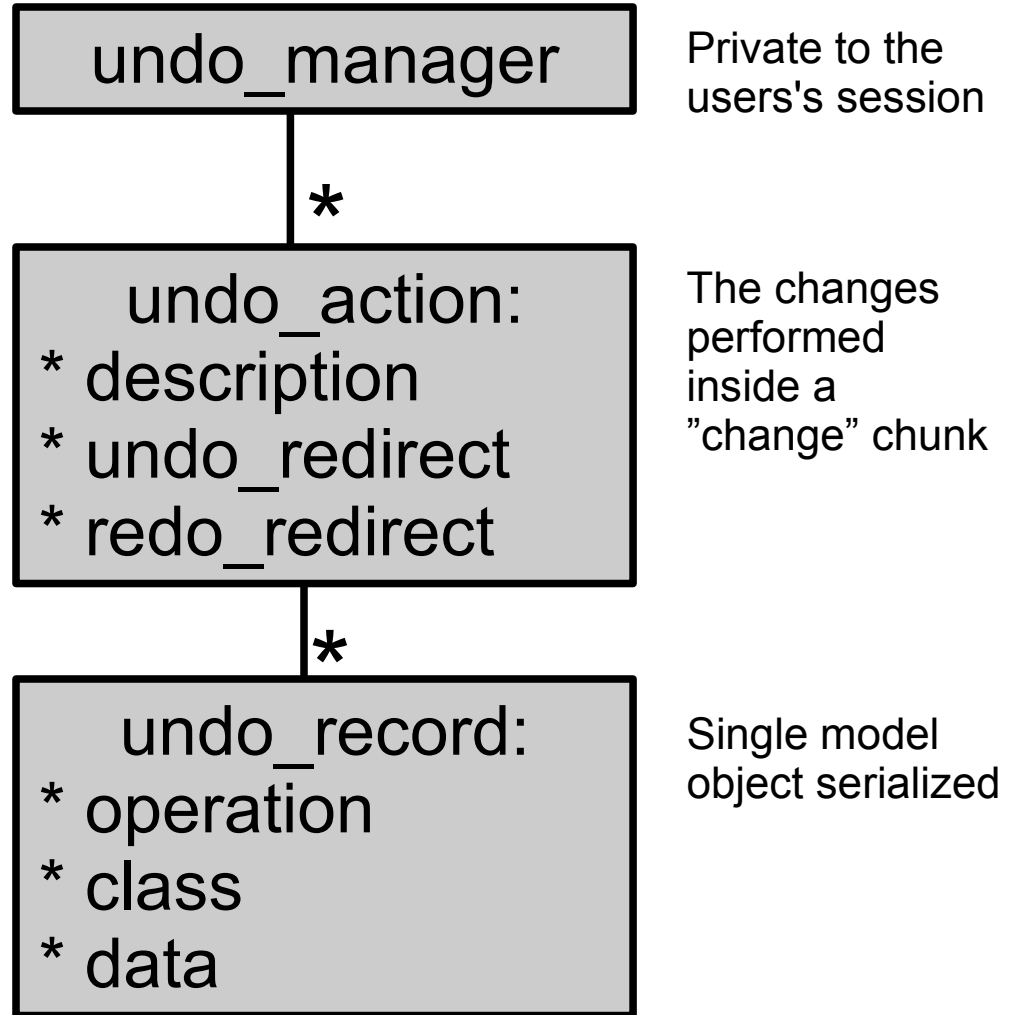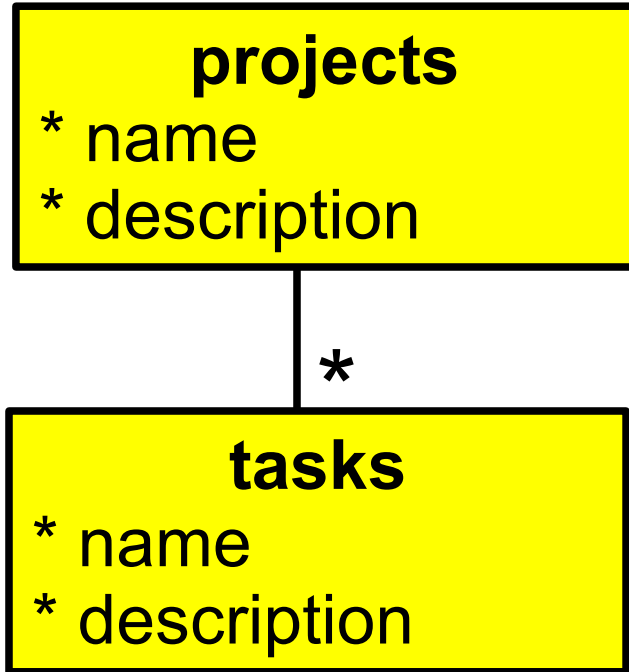
How does it work?

- Keeps track of the model object changes wrapped in the "change" method.

- Serialized model objects are stored in a separate table.

- When undoing/redoing, the serialized objects are restored.

- Each session has its own "undo manager" which tracks the current user's undoable/redoable actions.

TRIFORK.

Data model

projects
* name
* description

*

tasks
* name
* description

undo_manager

Private to the
users's session

*

undo_action:
* description
* undo_redirect
* redo_redirect

The changes
performed
inside a
"change" chunk

*

undo_record:
* operation
* class
* data

Single model
object serialized

TRIFORK.

Problems?

- The undo/redo trail is local to the users, so a user can undo the creation of an object that another user has later been dependent on.

- System-wide maximum size of single serialized model objects (default 2MB).

- What happens when developing the database schema?

- Serializes objects in the session cookie.

However...

- Easy to set up.
- Only 3 new database tables.
- User scope of undo/redo trail is not a problem in many scenarios.
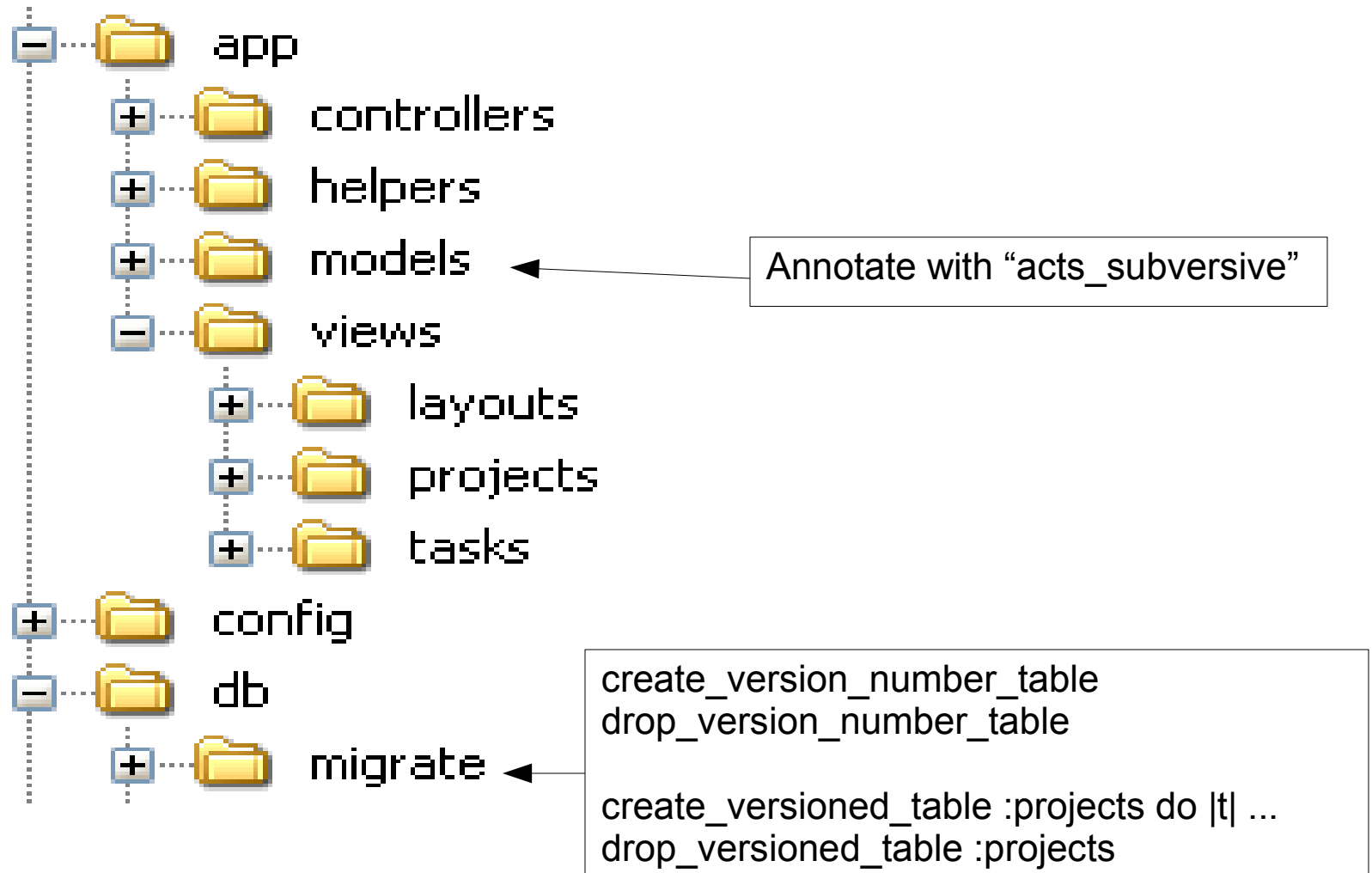- Incredibly cool!

acts_subversive

acts_subversive

- Like acts_as_logged, but can fetch old versions.

- Like acts_as_versioned, but works with associations and your own logic methods.

- http://acts-subversive.rubyforge.org/

- ruby script/plugin install svn://rubyforge.org/var/svn/acts-subversive/tags/0.1/acts_subversive

What to do?

```
[-] 📁 app
     [+] 📁 controllers
     [+] 📁 helpers
     [+] 📁 models          ◄──── Annotate with "acts_subversive"
     [-] 📁 views
          [+] 📁 layouts
          [+] 📁 projects
          [+] 📁 tasks
[+] 📁 config
[-] 📁 db
     [+] 📁 migrate   ◄──
```

create_version_number_table
drop_version_number_table

create_versioned_table :projects do |t| ...
drop_versioned_table :projects
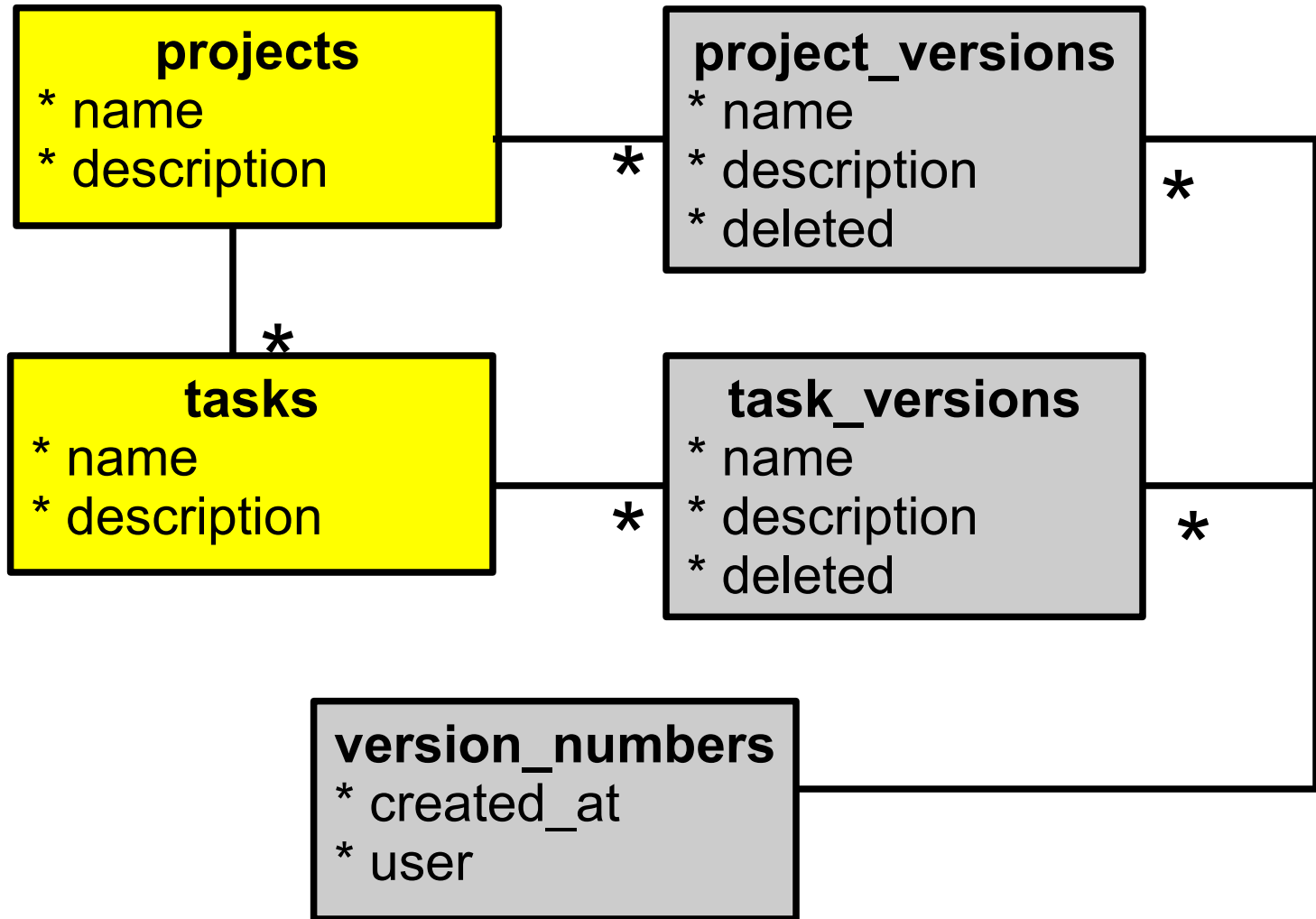
How does it work?

- Keeps a global, always increasing version number that gets stamped on the shadow copies.

- Only makes shadow copies of the updated objects, not the objects they reference (unless, of course, they have been updated too).

- When traversing associations, keeps the version number from the first find_version call.

- Complex SQL for finding associated objects.

Data model

Problems?

- has_and_belongs_to_many unsupported.
- Inheritance unsupported.
- (n+1) new tables when versioning n classes.

However...

- Traversing associations back in time actually does work.

- Your own instance methods actually work.

Recap

Recap

- 6 different, very elegant solutions to parts of our versioning problems.

- Neat code – check it out yourself!

- No distributed versioning, no branching.

What not to version?

- It is tempting to just annotate every model class with "acts_subversive", but...

- User rights?
  - Should the user be able to "turn back time" and read what he once could read?

- Big blobs?
  - Spacewise, it all adds up...
  - ...but are updated rarely?

By the way...

- ...I'm doing a "showdiff" plugin:
- http://show-diffs.rubyforge.org/
- Not there yet, but use it in the future :-)

Thanks!

- Check out the existing plug-ins.
- Do your own plug-ins – it's really simple.
- If you need a new feature, request it or do it yourself.
- Participate!
- Be creative!