



The PHP Company

ZEND FRAMEWORK

¡Toma el control!


Karén Nalbandian

knalbandian@alfa9.com

Mira el webinar grabado : <http://bit.ly/rp5iK1>


Original author and presenter: Ryan Maugier, ZF Contributor
& ZF CR Team Member.

¿Quién es Karén Nalbandian?

- Project Manager, Líder tecnológico y Desarrollador Senior en Alfa9 Servicios Web S.L.
- 12 años de experiencia en diseño y programación de aplicaciones Web en PHP y ASP .NET
- Zend Certified PHP5 Engineer 
- Alfa9 Servicios Web S.L. es Socio de Negocios de Zend Technologies en España.



¿Quién es Ryan Mauger?

- Original author and presenter of this webinar
- Zend Framework Contributor
- Zend Framework CR Team Member
- Co-Author of *Zend Framework 2 in Action* (With Rob Allen)
- Technical Editor for *Zend Framework: A Beginners Guide*
- Community Supporter
- Zend Certified PHP5 Engineer 
- Lead Developer at Lupimedia

¿Qué aprenderás?

Conceptos fundamentales que te
ayudarán a empezar a resolver las cosas
por ti mismo

¿Dónde empezar?

- Tutoriales
 - Akrobat's (Rob Allen): <http://akrobat.com/zft>
 - Quickstart Oficial: <http://bit.ly/zf-quickstart>
- Crea tu propio sandbox
 - TENLO SIEMPRE A MANO!
 - Actualízalo, experimenta y guarda los añadidos para usarlos mas adelante

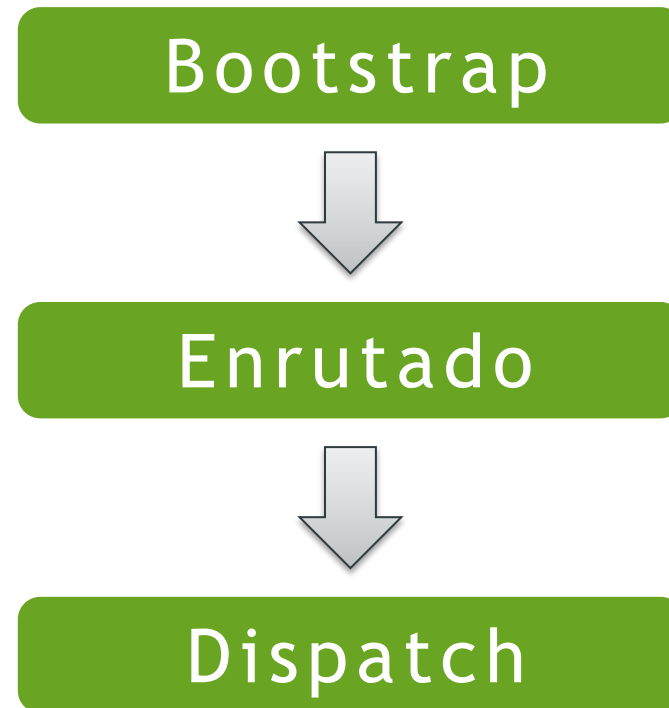
¿Qué sigue?

- Ciclo de dispatching
- Autoloaders, Plugin Loaders y Resource Loaders
- Plugins
- Helpers
- Models
- Forms, Decorators, Validators y Filters

Pero ¿y ahora que hago?

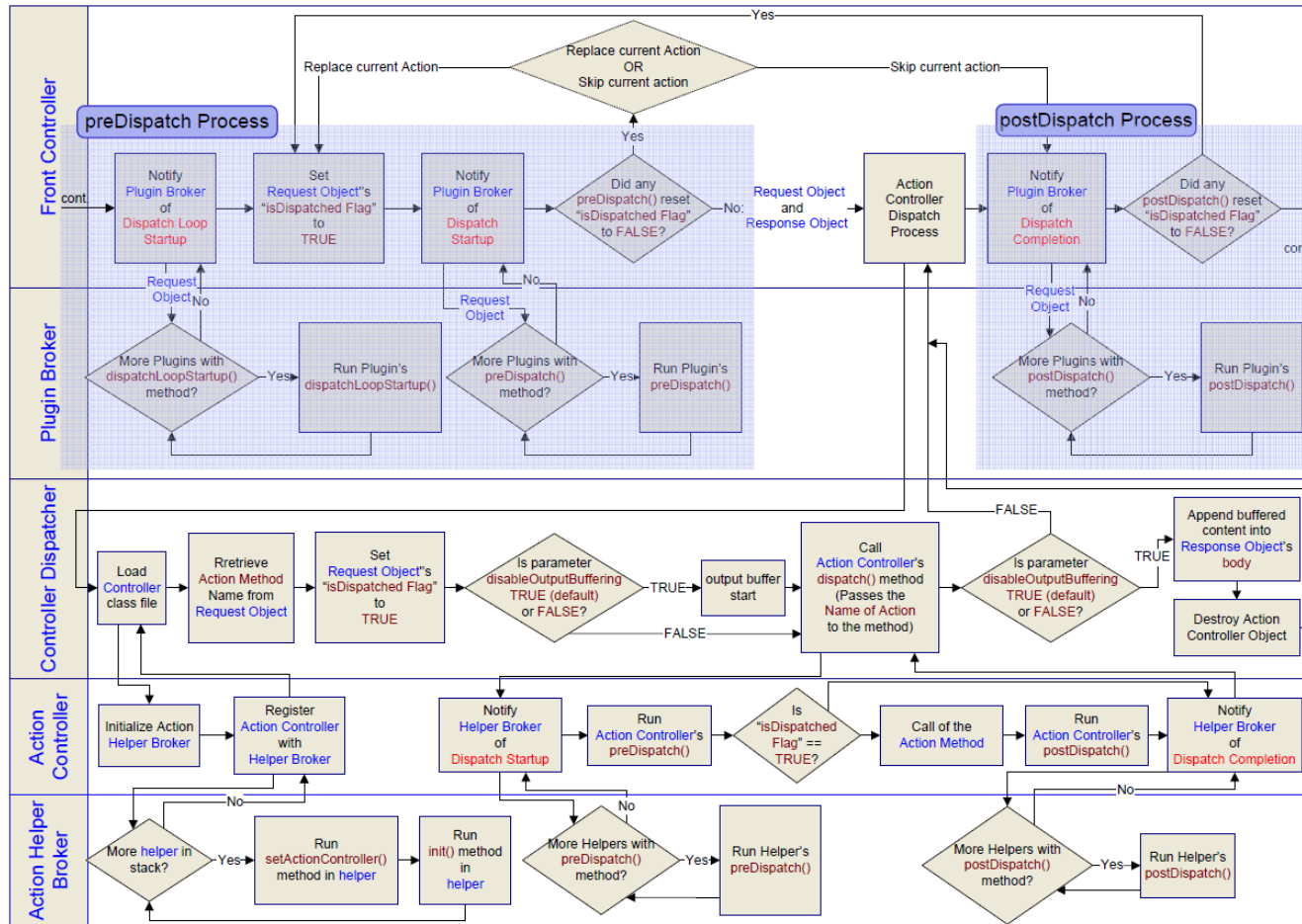
- Entender el ciclo de una petición en ZF
- Entender el ciclo de una petición en ZF
- Entender el ciclo de una petición en ZF
- Entender el ciclo de una petición en ZF

Ciclo de vida de una petición



Ah... ¿y nada más?

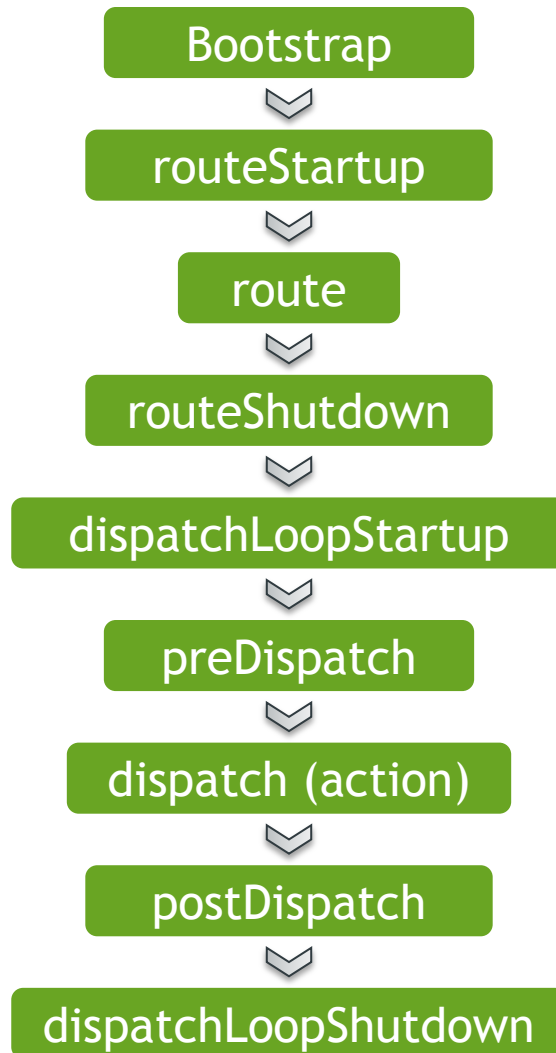
¿Más? ¿Qué tal el diagrama completo?



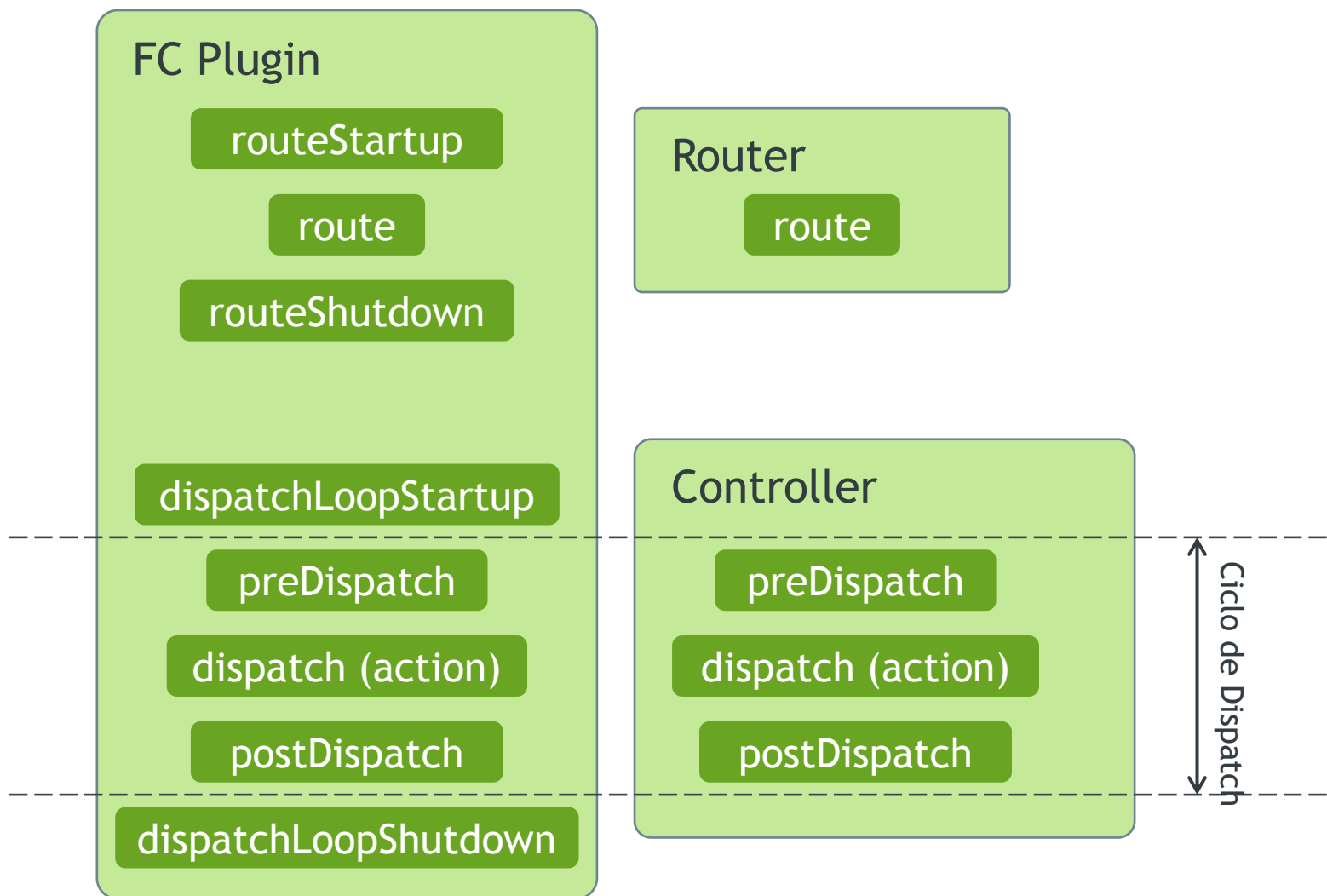
Fuente: <http://www.slideshare.net/polleywong/zend-framework-dispatch-workflow>

iOUCH!

Poniéndolo un poco mas simple



Poniéndolo un poco mas simple



BOOTSTRAPPING

Bootstrapping

- Inicializa los recursos que puedas necesitar.
- Prepara todo para que la petición pueda ser despachada.
- No hace nada específico a los módulos.
- Recuerda los Bootstrap de los módulos, incluso si estos están vacíos.

Bootstrap de Módulo

```
resources.frontController.moduleDirectory = APPLICATION_PATH "/modules"  
resources.frontController.controllerDirectory.default = APPLICATION_PATH "/controllers"  
resources.modules[] = ""
```

```
<?php  
  
class Abcd_Bootstrap extends Zend_Application_Module_Bootstrap  
{  
    protected function _initRest()  
    {  
        $fc = Zend_Controller_Front::getInstance();  
        $restRoute = new Zend_Rest_Route($fc, array(), array(  
            'abcd' => array('contacts'),  
        ));  
        $fc->getRouter()->addRoute('contacts', $restRoute);  
    }  
}
```

PLUGINS Y ACTION HELPERS DE FRONT CONTROLLER

Plugins de Front Controller

- Proveen hooks (ganchos) a diferentes puntos del ciclo de vida de una petición.
- Se ejecutan de forma automática.
- No deben depender de Action Controller para ser ejecutados.
- Las excepciones provocadas en preDispatch **no** impedirán la ejecución de preDispatch en el resto de los Plugins.
- Son más fáciles de usar si no tienen parámetros en el constructor.

Añadiendo Plugin a Front Controller

- Desde el archivo de configuración

```
autoloaderNamespaces[] = "My_"  
resources.frontController.plugins[] = "My_Controller_Plugin"
```

- Desde el Bootstrap (útil para los módulos)

```
<?php  
  
class Abcd_Bootstrap extends Zend_Application_Module_Bootstrap  
{  
    protected function _initPlugins()  
    {  
        $this->getApplication()  
            ->getResourcePlugin('frontController')  
            ->registerPlugin(new ModuleName_Plugin_Acl());  
    }  
}
```

Action Helpers

- Proveen hooks (ganchos) a varios puntos del ciclo de vida de una petición.
- Se ejecutan de forma automática o a petición.
- Están destinados a eliminar el código repetido en las acciones del Controller (principio DRY) o extender la funcionalidad de Action Controllers.
- Las excepciones lanzadas en pre/postDispatch impiden la ejecución del resto de Action Helpers.

Añadiendo Action Helper

- Desde el archivo de configuración

```
resources.frontController.actionHelperPaths.My_Action_Helper = "My/Action/Helper"
```

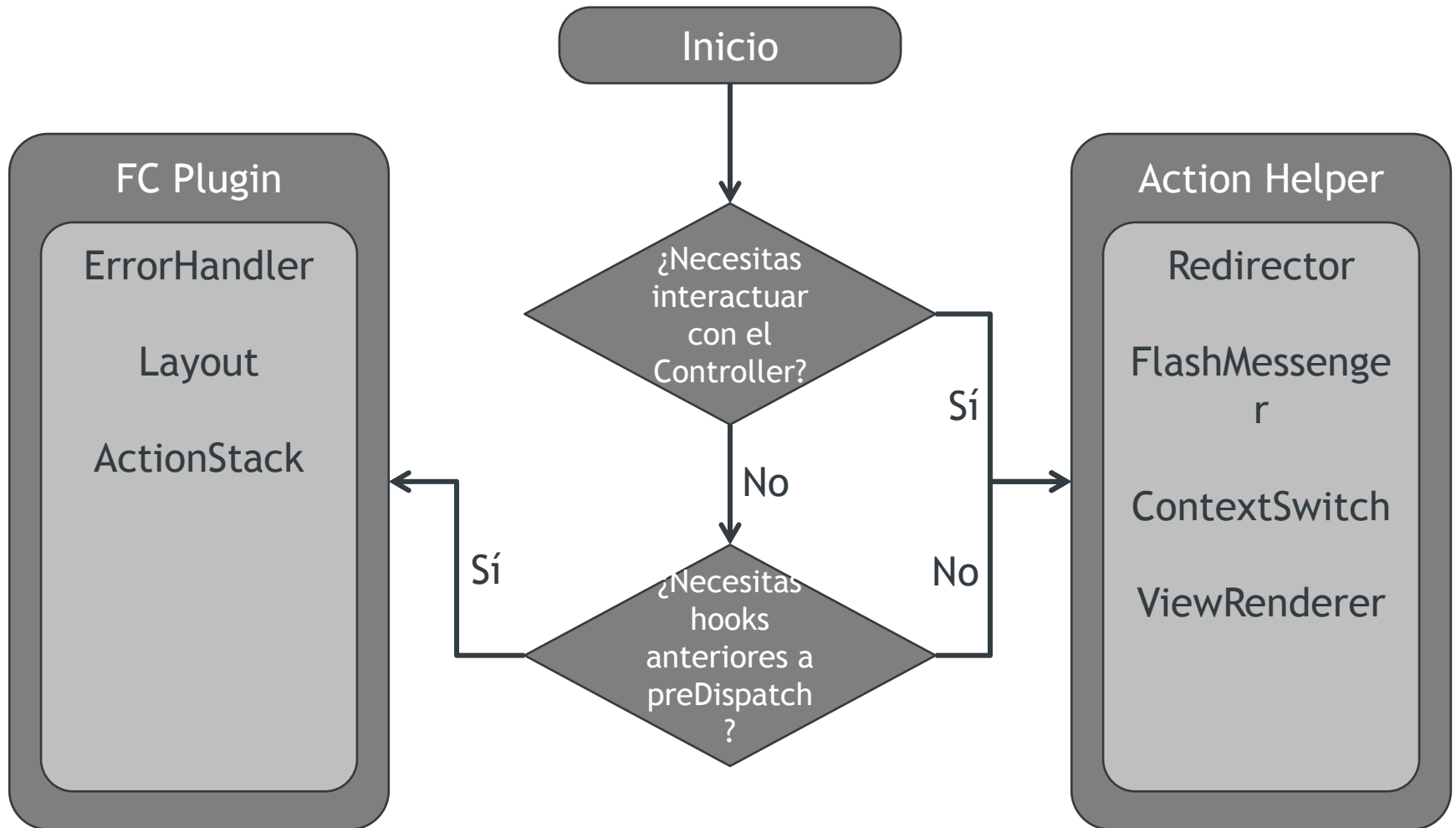
- Desde el Bootstrap (útil para los módulos)

```
<?php

class Abcd_Bootstrap extends Zend_Application_Module_Bootstrap
{
    protected function _initActionHelpers()
    {
        Zend_Controller_Action_HelperBroker::addPath(
            'My/Action/Helper',
            'My_Action_Helper'
        );

        Zend_Controller_Action_HelperBroker::addHelper(
            new My_Action_Helper_Thingy()
        );
    }
}
```

¿Action Helper o FC Plugin?



Autoloading

Autoloading

Carga de Plugins

Carga de Resources

Autoloading

- Componentes de librería
- Sigue el formato de nombres de PEAR
- Utilizado en todas partes del framework:
 - ▶ `new Zend_Form()`
 - ▶ `new Zend_Db()`
 - ▶ `new Zend_Service()...`

Carga de Plugins

- Utiliza prefijos para resolver nombres de clases y rutas para la carga.
- Funcionar fuera del contexto de `include_path`.
- Se utiliza siempre que una clase se crea sólo con un sufijo.
- Form Elements, View Helpers, Action Helpers, Resource Plugins

Carga de Resources

- Resuelve nombres de clases que no tienen correspondencia 1:1 con el sistema de archivos:
 - Application_Model_Page
 - application/models/Page.php
- Es usado por componentes de un Módulo (modelos, formularios, servicios)
- Provee espacio de nombres para el Módulo
- Se crea de forma automática para cada Módulo durante el proceso de Bootstrap
- Hace que las carpetas de los Módulos estén mas ordenados (controllers, models, views, forms, etc.)

Loaders en acción

Carga de Plugins

```
<?php

class My_Form extends Zend_Form
{
    public function init()
    {
        $this->addElement('Text', 'aTextBox', array(
            'label' => 'A text Box'));
    }
}
```

“Text” se resuelve a `Zend_Form_Element_Text` mediante la iteración a través de todos los prefijos/rutas configurados hasta que se encuentre una coincidencia.

Loaders en acción

Añadiendo tus propios prefijos

```
<?php

class My_Form extends Zend_Form
{
    public function init()
    {
        $this->addElementPrefixPath(
            'My_Form_Element_', 'My/Form/Elements/');
        $this->addElement('Text', 'aTextBox', array(
            'label' => 'A text Box'));
    }
}
```

`Zend_Form::addElementPrefixPath` es un ejemplo de cómo se accede al Cargador de Resources para añadir un nuevo prefijo/ruta. También acepta un tercer argumento opcional para aplicar la configuración únicamente a elementos, decoradores, filtros o validadores.

Loaders en acción

Autoloading

```
[production]
phpSettings.display_startup_errors = 0
phpSettings.display_errors = 0

autoloaderNamespaces[] = "My_"
includePaths.library = APPLICATION_PATH "../library"
bootstrap.path = APPLICATION_PATH "/Bootstrap.php"
bootstrap.class = "Bootstrap"
```

```
<?php

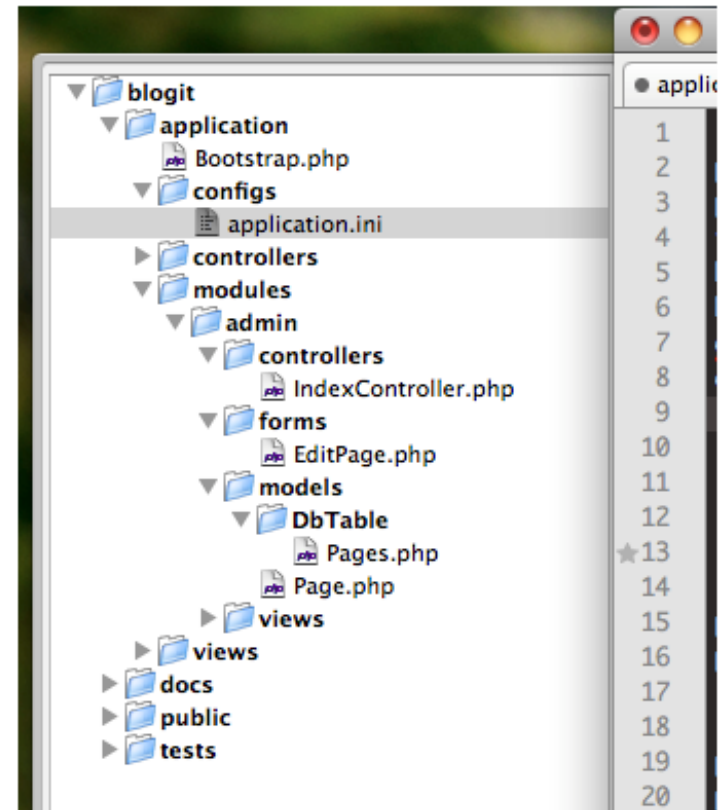
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        $myComponent = new My_Cool_Component();
    }
}
```

Loaders en acción

Carga de Resources

```
<?php

class Admin_IndexController
    extends Zend_Controller_Action
{
    public function indexAction()
    {
        $form = new Admin_Form_EditPage();
        $this->view->form = $form;
    }
}
```



Opciones del constructor

¿Qué puedo poner en \$options?

¿Qué puedo poner en \$options?

- \$options es siempre un array asociativo
- Cada clave es normalizada y convertida en nombre de método setter. Así, “foo” se convierte en “setFoo” y “bar”, en “setBar”.
- Si existe tal setter, es invocado pasándole el valor correspondiente de \$options como el único argumento.
- Por regla general no se lanzan excepciones para setters que no existen.
- Algunos componentes guardarán cualquier opción que no tenga setter para otros propósitos. Por ejemplo, en Zend_Form estos se convierten en atributos.

¿Dónde puedo encontrar los setters disponibles?

- En la documentación API (<http://framework.zend.com/apidoc/core>)
- En su IDE (Entorno de Desarrollo) si dispone de autocomplete
- En el Manual

Ventajas de usar array de opciones

- Flexibilidad de configuración
- Fácil de extender
- Autodocumentado

Aprovechando el array de opciones en Zend_Form

```
<?php

class Admin_Form_EditPage
    extends Zend_Form
{
    protected $_page;

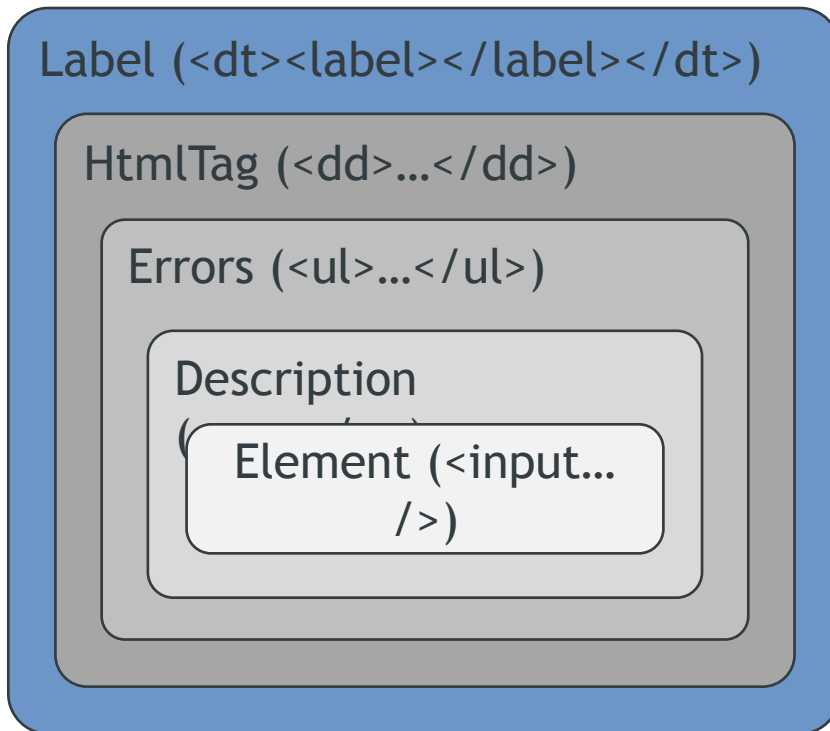
    public function init()
    {
        $this->addElements(array(
            // ...
            new Zend_Form_Element_Select('multi'),
            // ...
        ));
        $this->getElement('multi')
            ->setMultiOptions($this->_page);
    }

    public function setPage($page)
    {
        $this->_page = $page;
        return $this;
    }
}
```

El método `setPage()` es invocado antes del `init()` permitiendo pasar un objeto, array o un valor escalar a través del constructor y utilizarlo para cualquier propósito durante la inicialización del formulario.

DECORADORES

¿Cómo funcionan los decoradores?



Se renderizan desde dentro hacia fuera. El nuevo contenido generado envuelve, se añade antes o se añade después del contenido obtenido desde el decorador anterior.

Definiendo el stack de decoradores

```
$this->getElement('multi')
->setMultiOptions($this->_page)
->setDecorators(array(
    'ViewHelper',
    'Description',
    'Errors',
    array('HtmlTag', array(
        'tag' => 'dd',
    )),
    array('Label', array(
        'tag' => 'dt',
    )),
));
```

- ViewHelper – renderiza el elemento.
- Description – añade el párrafo con descripción (si existe) debajo del elemento.
- Errors – añade la lista de errores debajo del elemento.
- HtmlTag – envuelve el contenido con etiquetas *dd*.
- Label – añade la etiqueta *label* envuelta en *dt* antes del elemento.

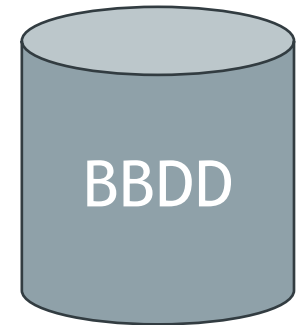
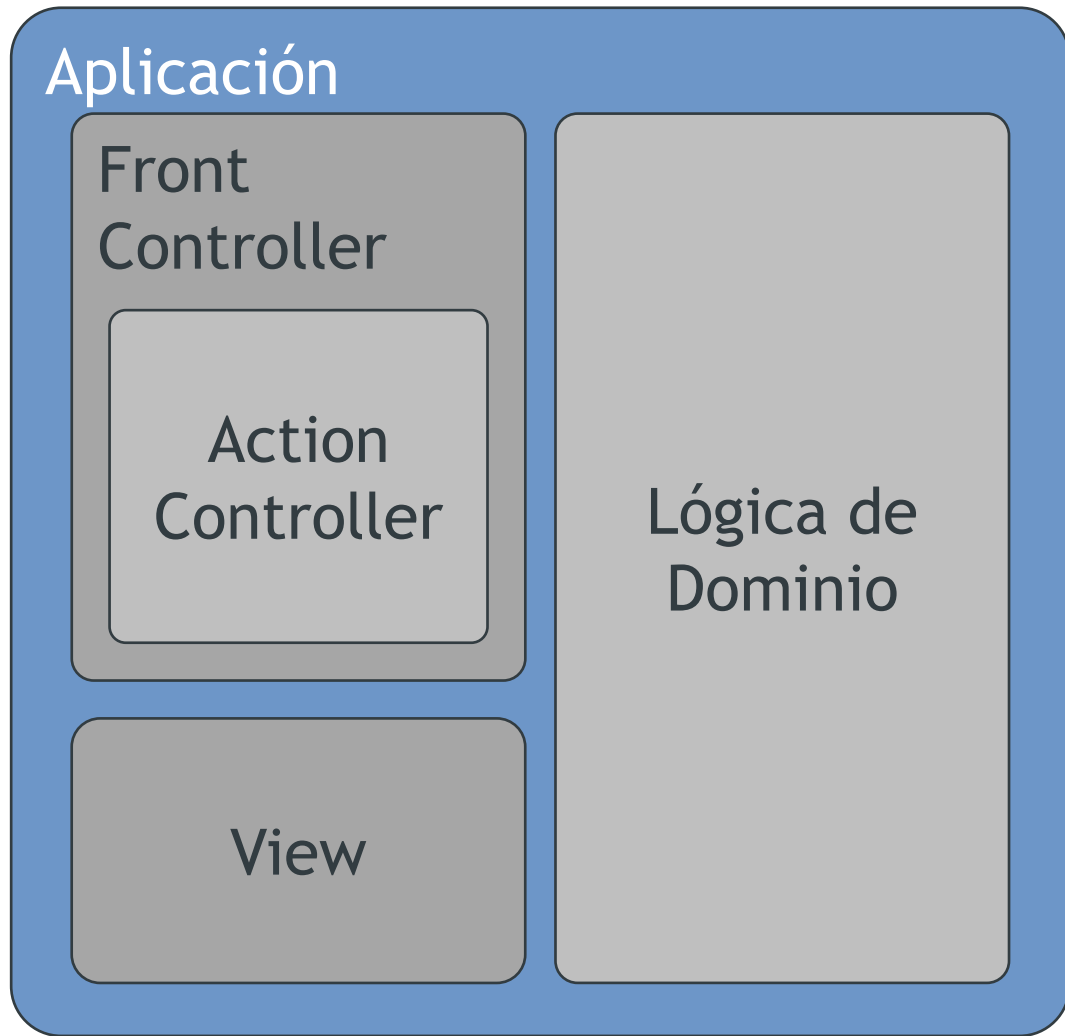
Definiendo el stack de decoradores

La M del MVC

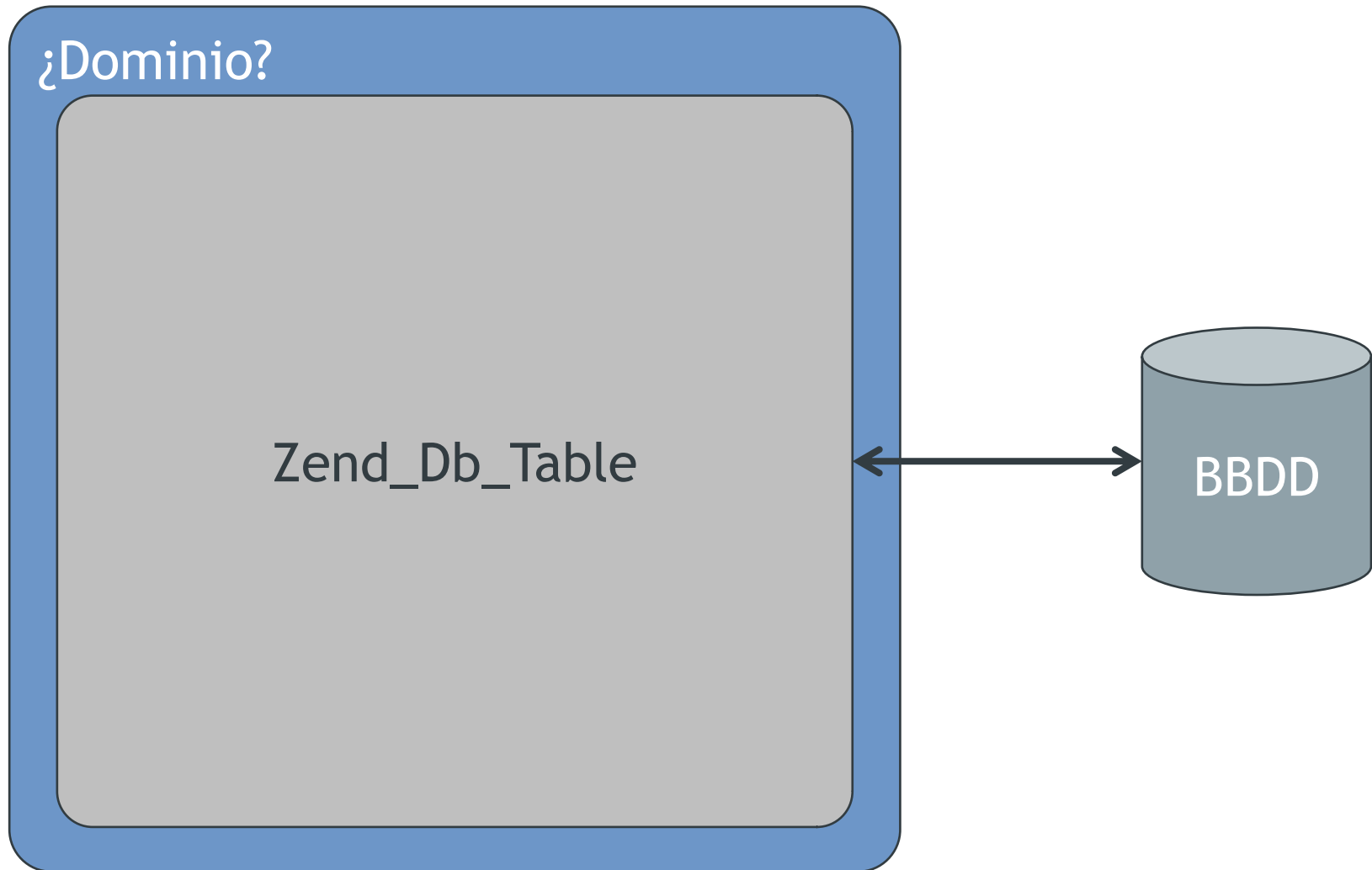
Modelando los datos

- Lógica de negocios
- Lógica de dominio
- Servicios
- Mappers
- Entidades
- Modelos

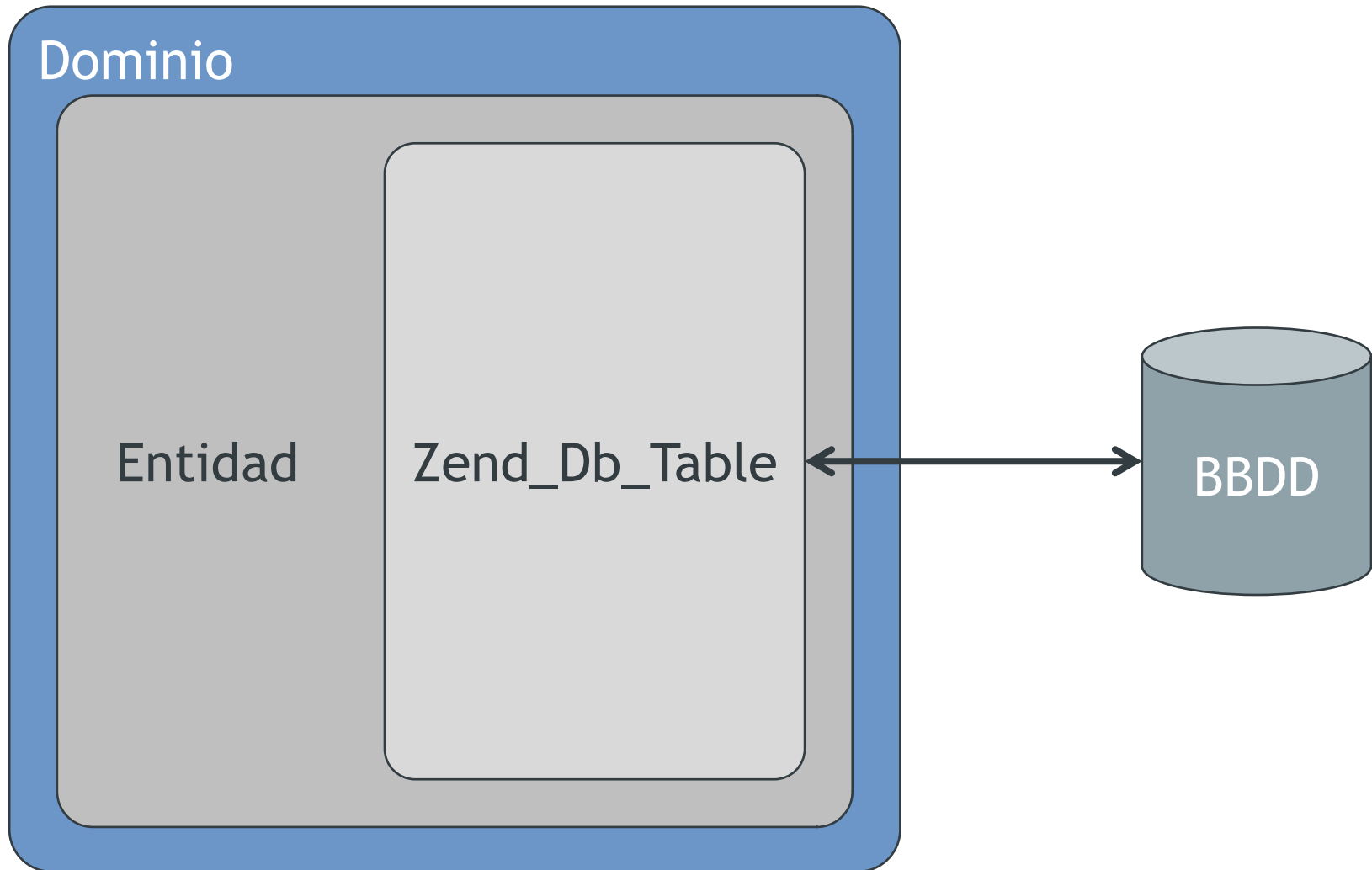
Un poco de MVC



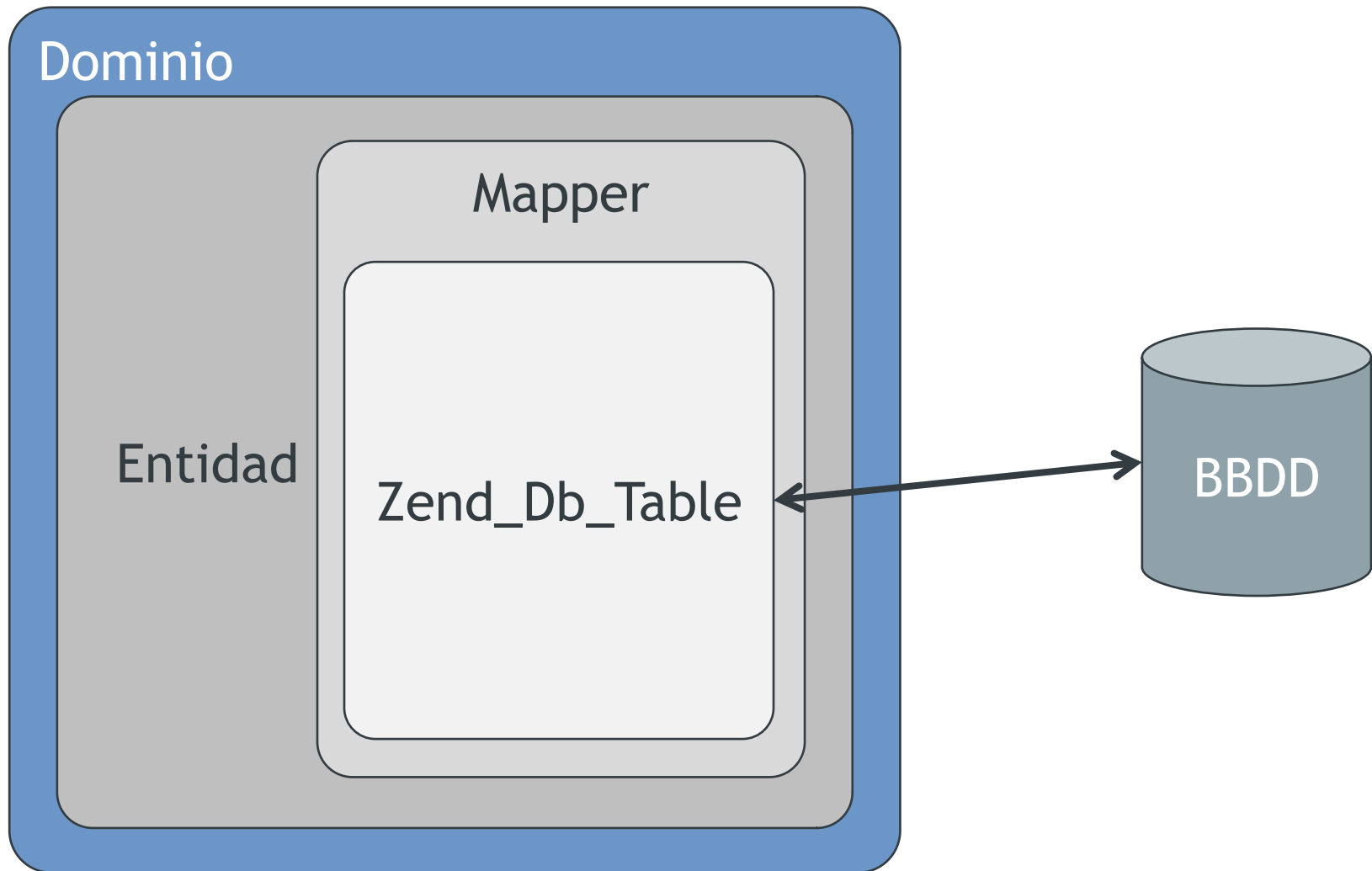
Modelos basados en Zend_Db_Table



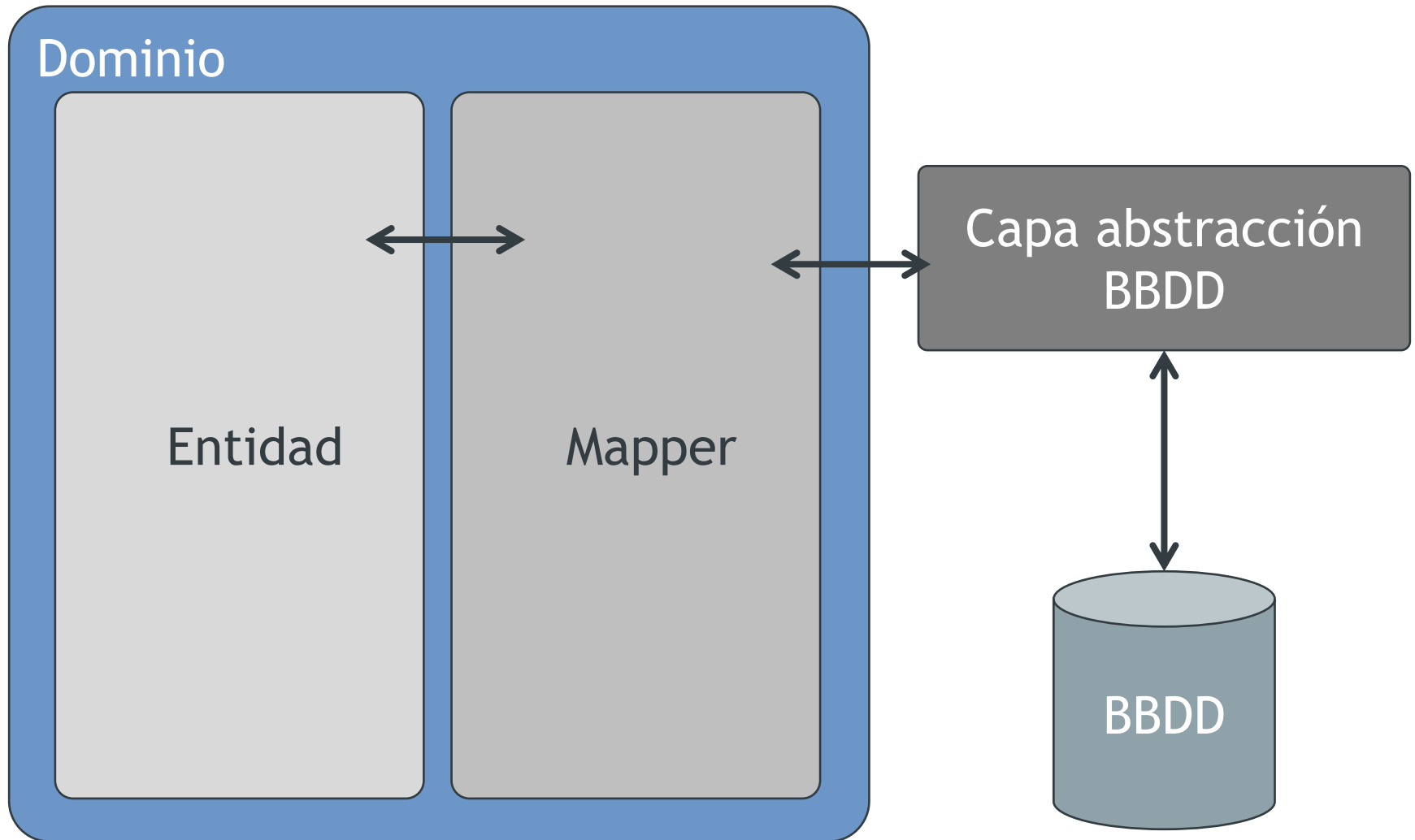
Modelos basados en Zend_Db_Table



Modelos basados en DataMapper



Modelos basados en DataMapper



¿Qué patrón utilizar?

- Ciclo de mantenimiento
- Complejidad de la Aplicación
- Estado actual del Proyecto
- Soluciones disponibles (Doctrine, Propel, phpDataMapper) y su conveniencia de uso.

¿Qué soporta ZF?

Abstracción de Base de Datos

¿Qué soporta ZF?

Abstracción de Base de Datos

- Suponemos algún tipo de integración de Doctrine2 cuando salga ZF2.
- Doctrine2 está ganando popularidad como el sistema ORM para ZF.
- Doctrine 1.x sigue siendo una elección de ORM muy extendida, a pesar de implementar el patrón Active Record.



¡Te agradecemos la participacion!
¿Preguntas?

Karén Nalbandian

knalbandian@alfa9.com

Mira el webinar grabado: <http://bit.ly/rp5iK1>

Si quieres contactar **Zend Technologies** o enviarnos tus
comentarios:

Ana Maria Valarezo - ana.m@zend.com