“If I can Cake it there, I’ll Cake it anywhere.”
Why a Framework?

- Technology is a commodity
- Abstract away the commodity
- Focus on the things that are of value
What about others?

- Zend Framework
- PHP on Trax
- CodeIgniter
- Symfony
Zend Framework

- A framework of exclusion - not just PHP 5, but PHP 5.1.4
- Yet Another Class Library - not enough to be called a framework
- The Windows Vista of PHP
The Windows Vista of PHP

What they said then:

- **Controller/Dispatcher:**
  
  “[...] the front controller doesn't rely on a sophisticated collection of mod_rewrite rules.”[1]

- **ActiveRecord:**

  ```php
  $people = Person::findAll(
      array('nameFirst' => 'Daniel')
  );
  ```
Controller/Dispatcher:

“you will also need the Apache web server, as some of the functionality provided by the news system I present in this article requires the use of mod_rewrite.” [2]

ActiveRecord:

```php
$sql = "INSERT INTO comments (name, comment, newsId)
VALUES ('$name', '$comment', '$newsId')";
return $this->_db->query($sql);
```
Reams of configs + Getters to the \( n \)-th degree for the simplest things

```php
propel:
  weblog_post:
    _attributes: { phpName: Post }
    id:
    title: varchar(255)
    excerpt: longvarchar
    body: longvarchar
    created_at:
  weblog_comment:
    _attributes: { phpName: Comment }
    id:
    post_id:
    author: varchar(255)
    email: varchar(255)
    body: longvarchar
    created_at:

default:
  http_metas:
    content-type: text/html; charset=utf-8

$\text{this} \rightarrow \text{post} = \text{PostPeer}::\text{retrieveByPk}(
  \$\text{this} \rightarrow \text{getRequestParameter}(\text{'id'})
);

$\text{c} = \text{new Criteria}();
$$\text{c} \rightarrow \text{add}($$
  \text{CommentPeer}::\text{POST_ID},
  \$\text{this} \rightarrow \text{getRequestParameter}(\text{'id'})
$$);$$

$$\text{c} \rightarrow \text{addAscendingOrderByColumn}($$
  \text{CommentPeer}::\text{CREATED_AT}
$$);$$

$$\text{this} \rightarrow \text{comments} = \text{CommentPeer}::\text{doSelect}($$\text{c}$$);$$

\text{Sound like another language we know of?}
$this->post = PostPeer::retrieveByPk($this->getRequestParameter('id'));

$c = new Criteria();
$c->add(CommentPeer::POST_ID, $this->getRequestParameter('id'));
$c->addAscendingOrderByColumn(CommentPeer::CREATED_AT);
$this->comments = CommentPeer::doSelect($c);

vs. “vanilla” PHP:

$post = $db->query("SELECT * From posts where id = \$_GET[‘id’]’");

$comments = $db->query(
    "SELECT * From comments where post_id = \$_GET[‘id’] ORDER BY created ASC"");
Why CakePHP?
A Revolutionary Concept:

PHP development...

In PHP!
Why CakePHP?

Playing to the strengths of the language

- **Simple**, array-based ActiveRecord
- Consistent code across PHP4 and 5: Cake brings PHP5 OO constructs to PHP4
- Structure by default
- Maintains the PHP freedom
- Deploy **anywhere**
Who’s on Cake?
I cannot put into words how much better this is than writing SQL and dealing [with] $_GET.

- Patrick Mineault, Lead Developer, AMFPHP

The ActiveRecord modeling is awesome. I can associate tables anyway I want as well as define runtime associations of tables.

- Jim Plush, Senior PHP/Ajax Developer, Panasonic

Much to my surprise, there was a page greeting me telling me what to do. [...] Who needs documentation when it tells me how to do everything?

- Jonathan Snook, snook.ca

Perl is a giant wasteland.

- John Resig, Creator, jQuery
MVC Quickie

- Primary: separation between Controller and View, to partition business logic and presentation
- Secondary: separation between data (Model) and Controller
MVC Quickie

The Dispatcher requests the appropriate Controller/action, which interacts with the Model

The Controller then sends the results of its operations to the view, where it is rendered
A simple example

/* models/post.php */
class Post extends AppModel { }

/* controllers/posts_controller.php */
class PostsController extends AppController {

    function index() {
        // Get the data from the Model
        $posts = $this->Post->findAll();

        // Send the data to the view
        $this->set('posts', $posts);
    }
}

MVC Quickie
The M: ActiveRecord

Model Definition

/* models/post.php */
class Post extends AppModel {
    var $hasMany = 'Comment';
}

/* models/comment.php */
class Comment extends AppModel {
    var $belongsTo = 'Post';
}

The Data

Array
(
    [0] => Array
    ( [Post] => Array
        ( [id] => 2
            [title] => A title once again
            [body] => And the post body fol
        )
        [Comment] => Array
            ( [0] => Array
                ( [id] => 4
                    [post_id] => 2
                    [body] => Are we there
                    [created] => 2006-09-20
                    [modified] => 2006-09-20
                )
            )
    )
    [1] => Array
        ( [Post] => Array
            ( [id] => 1
                [title] => The title
                [body] => This is the post body
            )
            [Comment] => Array
                ( [0] => Array
                    ( [id] => 3
                        [post_id] => 1
                        [body] => I'm bored
                        [created] => 2006-09-20 11:57:05
                        [modified] => 2006-09-20 11:57:05
                    )
                )
        )
)
/* models/post.php */
class Post extends AppModel {
    var $hasMany = 'Comment';
}

/* models/comment.php */
class Comment extends AppModel {
    var $belongsTo = 'Post';
}

/* controllers/posts_controller.php */
class PostsController extends AppController {
    var $scaffold;
}

/* controllers/comments_controller.php */
class CommentsController extends AppController {
    var $scaffold;
}

...and you’re done.
Building an App
Request/Response

1. Apache or IIS
2. Dispatcher
3. Controller
4. Redirect
5. Model
6. View
7. Database
8. requestAction()
Model Associations

- **hasMany:**
  
  Post **hasMany** Comment / Comment **belongsTo** Post

- **belongsTo:**
  
  Comment **belongsTo** Post / Post **hasMany** Comment

- **hasOne:**
  
  Invoice **hasOne** Payment / Payment **belongsTo** Invoice

- **hasAndBelongsToMany:**
  
  Post **hasAndBelongsToMany** Tag /
  Tag **hasAndBelongsToMany** Post
Model Associations

- **hasMany:**
  
  `Post.id = Comment.post_id`

- **belongsTo:**
  
  `Comment.post_id = Post.id`

- **hasOne:**
  
  `Payment.invoice_id = Invoice.id`

- **hasAndBelongsToToMany:**
  
  `Post.id = posts_tags.post_id && Tag.id = posts_tags.tag_id`
Simple Solutions To Common Problems
Simple Solutions

- **Validation Model:**
  ```php
class Post extends AppModel {
    var $validate = array(
      'title' => VALID_NOT_EMPTY,
      'body' => VALID_NOT_EMPTY
    );
  }
}
```

- **View:**
  ```php
echo $html->tagErrorMsg(
    'Post/title',
    'You must include a title!' );
```
Query Building - Controller::postConditions()

Array

( [Product] => Array
    ( [category_id] => 3
    [name] => V for Vendetta
    )
)

- $this->Product->findAll($this->postConditions());
  SELECT `Product`.`id`, ... FROM `products` AS `Product` WHERE `Product`.`category_id` = 3 AND `Product`.`name` = 'V for Vendetta'

- $this->postConditions(null, array('name' => 'like'), null, true);
  SELECT `Product`.`id`, ... FROM `products` AS `Product` WHERE `Product`.`name` LIKE '%V for Vendetta%'
• Pagination

Controller:

```php
var $paginate = array(
    'limit' => 20, 'order' => 'pub_date ASC'
);
function index() {
    $articles = $this->paginate();
}
```

View:

```php
$paginator->prev("<< Previous");
$paginator->next("Next >>");
```
Ajax!
Ajax

- Links

```php
$html->link('Add Post', '/posts/add');

vs.

$ajax->link('Add Post', '/posts/add', array(
    'update' => 'addPostDiv',
    'complete' => 'Effect.SlideDown("addPostDiv")');
```
• Updating Multiple DIVs

View:

```php
<?php e($ajax->div('Div1'));
   // Updated content goes here
   e(strtotime('now'));
   e($ajax->divEnd('Div1'));

   $ajax->link('Update Div', '/', array('update' => array('Div1', 'Div2')));

?>
```
Ajax

- Auto-completing Fields

  Controller:
  ```php
  var $components = array('Autocomplete');
  ```

  View:
  ```php
  <?php echo $ajax->autoComplete('User/name'); ?>
  ```

  ...and you’re done.
Security

• SQL Injection:

  Magic methods like `findById($id)`
  Building queries semantically allows Cake to escape data automatically, i.e.:

  ```php
  findAll(array('Post.id' => $id)) or
  findAll(array('Post.body' => "LIKE {$text}"));
  ```
Security

• Escaping output:

  Most methods that output HTML escape data by default:

  ```php
  $html->link("Next >" ...);
  <a href="...">Next &gt;</a>
  ```
Security

- The Sanitize Object
- Hardcore HTML escaping
- Stripping specific tags
- Replaces invalid and funny-byte characters
Security

- Cross-Site Scripting
- Output from POST data is always escaped.
Of XML, RSS and APIs

site.com/posts.rss:

- **config/routes.php:**
  
  ```php
  Router::parseExtensions();
  ```

- **views/posts/rss/index.ctp:**
  
  ```php
  echo $rss->items($data, 'transformRSS');
  function transformRSS($post) {
      return array(
          'title' => $post->title,
          'link' => array('action' => 'view', $post->id),
          'guid' => array('action' => 'view', $post->id),
          'description' => $post->body,
          'author' => $post->User->name,
          'pubDate' => $post->modified
      );
  }
  ```
What’s up with that syntax?

- The Set object

```php
$post['Post']['title'] &
$post['Comment'][0]['body']
+
Set::map($post);
=
$post->title &
$post->Comment[0]->body
```
The Future

- New Cache class
- Support for “cache engines,” including:
  - File
  - APC
  - Memcache
The Future

- New Debugging Methods
  - `Object::trace()` - Trace calls from any object, with various output options
  - New “debugging” view allows you to examine objects, along with current environment data
The Future

- Localization & Internationalization
- New String class to support Unicode in native PHP
- Already done: support for localized view templates
- In the works: localized model data
The Future

- Model Behaviors
- Allow extensions to Model similar to Components being added Controllers
- Add your own natively-callable methods directly to Model objects, plus wildcard methods like findBy*
- Callbacks for most Model methods
The Future

- Native Support for new DBs
- Firebird
- PDO
  &
- Oracle