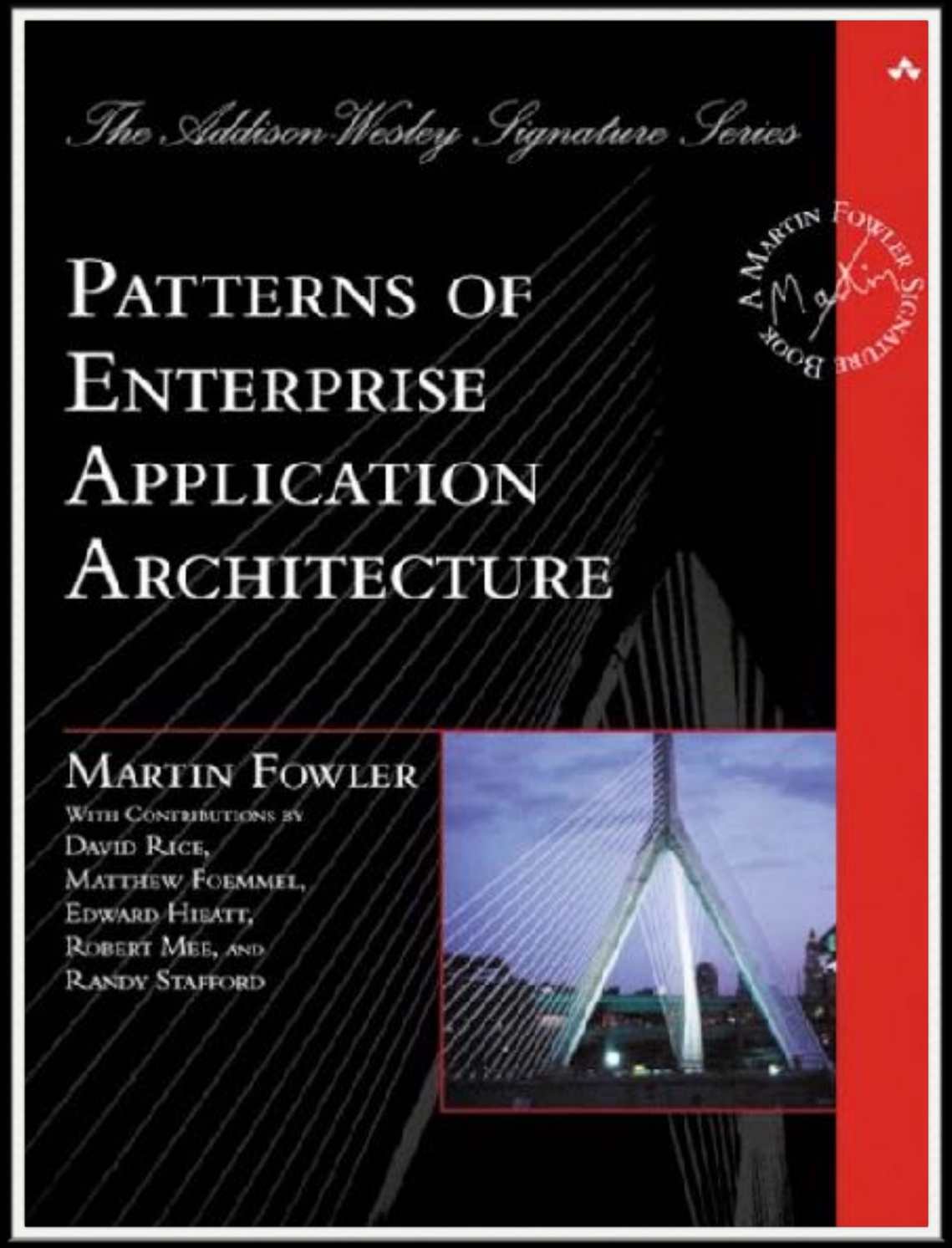
Solar for PHP 5

Paul M. Jones

http://joind.in/2814

Read This


The Addison Wesley Signature Series
PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE
A MARTIN FOWLER SIGNATURE BOOK
MARTIN FOWLER
WITH CONTRIBUTIONS BY
DAVID RICE,
MATTHEW FOEMMEL,
EDWARD HIEATT,
ROBERT MEE, AND
RANDY STAFFORD

About Me

- Developer to VP Engineering
- PHP since 1999 (PHP 3)
- Savant Template System (phpsavant.com)
- PEAR Group (2007-2008)
- Zend Framework
- ZCE Advisory Board
- Web framework benchmarks

“Another Framework?”

- http://solarphp.com
- The first E_STRICT framework for PHP 5
- Pre-dates Zend Framework
- Portions of ZF based on early Solar work (DB, DB_Select, DB_Table/Row/Rowset, View, View_Helper, and others)

Overview

- Foundational concepts and techniques
- Dispatch cycle
- Package highlights
- Project architecture
- Performance indicators

Foundations

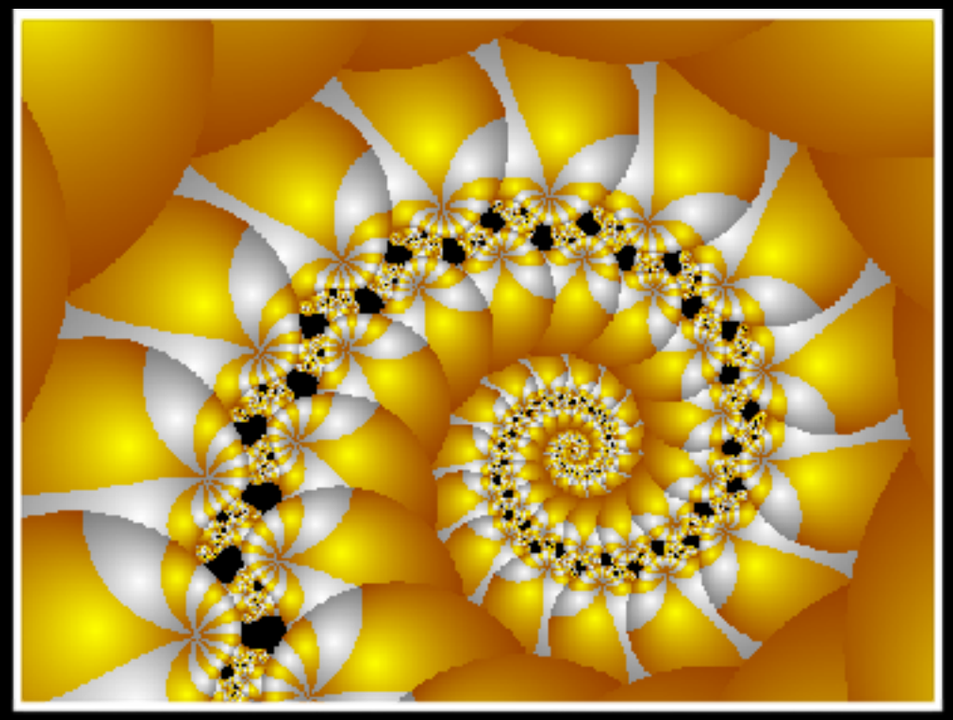
PEAR Standards

- pear.php.net
- 10+ years of public usage and vetting
- Coding style guide
- Class-to-file naming convention (PSR-0)

```php
<?php class Vendor_Foo_Bar {
   // Vendor/Foo/Bar.php
}
```

Class-to-File Effects

- Consistent and predictable (autoloader)
- Low cognitive friction on structure and organization
- Adaptable to change and integration
- Support files in parallel



Techniques

- Unified constructor
- Unified configuration
- Unified factory
- Lazy-loading registry
- Dependency management

Typical Constructor

```
class Foo {
  protected $_foo;
  protected $_bar;
  protected $_baz;
  public function __construct(
    $foo = null,
    $bar = "dib",
    $baz = "gir"
  ) {
    $this->_foo = $foo;
    $this->_bar = $bar;
    $this->_baz = $baz;
  }
}
```

Unified Constructor

```
class Vendor_Foo {
  protected $_Vendor_Foo = array(
      "foo" => null,
      "bar" => "baz",
      "dib" => "gir",
  );
  protected $_foo;
  protected $_bar;
  protected $_baz;
  public function __construct($config = array())
  {
    $config = array_merge($this->_Vendor_Foo, $config);
    $this->_foo = $config["foo"];
    $this->_bar = $config["bar"];
    $this->_baz = $config["baz"];
  }
}
```

Typical Config File

```
webhost                = www.example.com
database.adapter       = pdo_mysql
database.params.host   = db.example.com
database.params.username = dbuser
database.params.password = secret
database.params.dbname = dbname

# what class uses them?
# what if different classes use "database" as their key?
```

Solar Config File

```
<!-- /path/to/config.php -->
<?php
$values = array();
$values["Solar_Sql"] = array(
  "adapter" => "Solar_Sql_Adapter_Mysql",
  "host"    => "db.example.com",
  "user"    => "dbuser",
  "pass"    => "secret",
  "name"    => "dbname",
);

return $values;
?>

<!-- capture return value from include -->
<?php $config = include "/path/to/config.php"; ?>
```

Unified Configuration

- Given unique class names, and unified constructor ...
- ... configuration keys map directly to class names ...
- ... unified configuration mechanism for all classes.

```php
$config = array(
  "Vendor_Foo" => array(
    "foo" => "zim",
  ),
);

// default values from config file
$foo = new Vendor_Foo($config['Vendor_Foo']);
```

Unified Factory

- Solar::factory() instead of `new` keyword

```
// non-adapter
$foo = Solar::factory("Vendor_Foo"); // Vendor_Foo with config

// adapter: assume that the config file has set the value ...
// $config["Solar_Sql"]["host"] = "db1.example.com";
$sql = Solar::factory("Solar_Sql");

// factory a new instance with instance-time config
$sql_other = Solar::factory("Solar_Sql", array(
  "host" => "db2.example.com",
));
```

Typical Registry

```
// populate registry entries
$db = new DB::factory('mysql');
Registry::set('db', $db);

// later, elsewhere
$db = Registry::get('db');
```

Lazy-Loading Registry

```
// lazy, using default adapter and configs.
Solar_Registry::set("sql", "Solar_Sql");

// lazy, via config file
$config["Solar"]["registry_set"]["sql"] = "Solar_Sql";

// instantiation
$sql = Solar_Registry::get("sql");
```

Dependency Management

- How to maintain dependencies on other objects?
- How to configure, replace, or test the other object?
- "Dependency injection" or "service locator"

```
// typical dependency creation
public function __construct() {
  $this->db = new DB();
}
```

Dependency Injection

```
// constructor-based dependency injection.
// $db = DB::getAdapter();
// $foo = new Foo($db);
public function __construct($db) {
   $this->db = $db;
}

// setter-based dependency injection.
// $foo = new Foo();
// $db = DB::getAdapter();
// $foo->setDb($db);
public function setDb($db) {
   $this->db = $db;
}
```

Service Locator

```
// static locator.
// $db = DB::getAdapter();
// Locator::set("db", $db);
public function __construct() {
   $this->db = Locator::get("db");
}

// instance locator as dependency injection.
// $locator = new Locator();
// $db = DB::getAdapter();
// $locator->set("db", $db);
// $foo = new Foo($locator);
public function __construct($locator) {
   $this->db = $locator->get("db");
}
```

Solar::dependency()

- Combined service locator and dependency injector
- Uses a registry key (which may be lazy-loaded) ...
- ... or a directly-passed dependency object.

```
Solar::dependency($class_hint, $specification);
```

Solar::dependency()

```php
class Vendor_Foo extends Solar_Base {
  public function _postConstruct() {
    parent::_postConstruct();
    $this->_sql = Solar::dependency(
      "Solar_Sql",
      $this->_config["sql"]
    );
  }
}

// inject an object
$sql = Solar::factory("Solar_Sql");
$foo = Solar::factory("Vendor_Foo", array("sql" => $sql));

// locate via registry key "sql"
Solar_Registry::set("sql", "Solar_Sql");
$foo = Solar::factory("Vendor_Foo", array("sql" => "sql"));
```

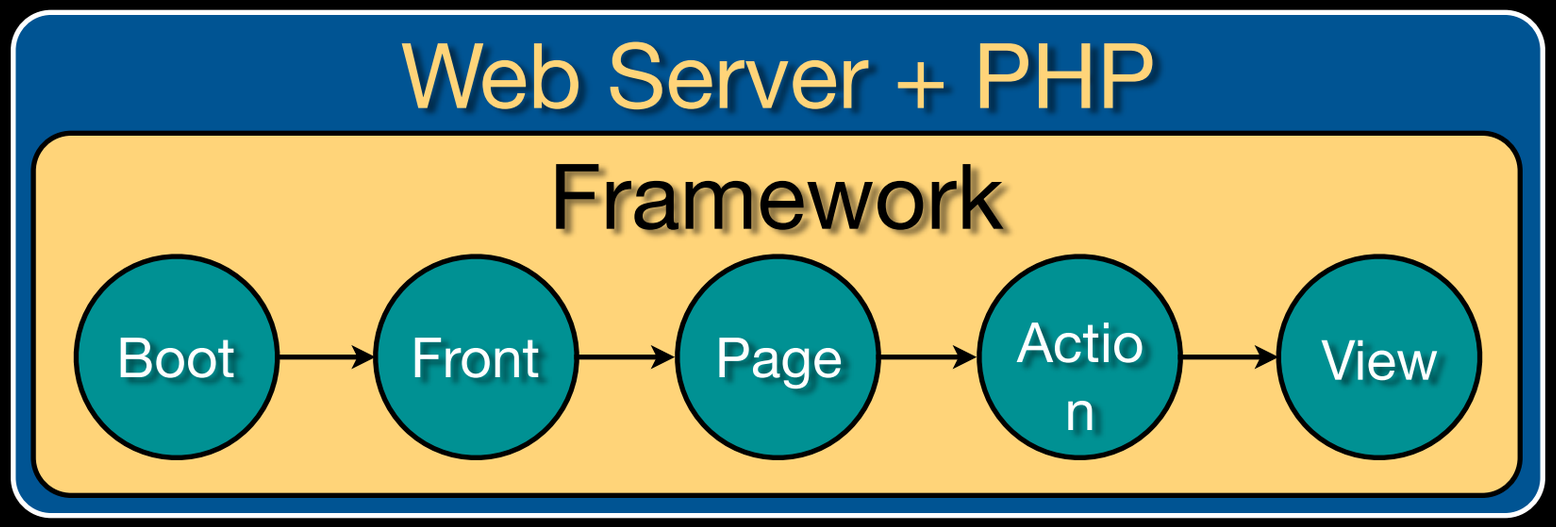
Dependency Config

```
<!-- /path/to/config.php -->
<?php
// lazy-load Solar_Sql instance as "sql"
$config["Solar"]["registry_set"]["sql"] = "Solar_Sql";

// use "sql" registry key for Vendor_Foo dependency
$config["Vendor_Foo"]["sql"] => "sql";
?>

<!-- script file -->
<?php
// now Vendor_Foo locates the "sql" entry automatically
$foo = Solar::factory("Vendor_Foo");
```

Dynamic Dispatch Cycle


Web Server + PHP
Framework
Boot
Front
Page
Actio
n
View

Bootstrap

```php
$system = dirname(dirname(__FILE__));
set_include_path("$system/include");

require "Solar.php";

$config = "$system/config.php";
Solar::start($config);

$front = Solar_Registry::get("controller_front");
$front->display();

Solar::stop();
```

Web App Controllers

- Independent front and page controllers
- URI default format of
  `/controller/action/param/param/param`
- Front controller determines **only** the page controller class, **not** the action or params

Front Controller

- Stack of class prefixes, e.g. `Vendor_App`
- `/foo/bar/baz => Vendor_App_Foo`

Front Controller

- Dynamic rewriter
- Static routing

```
$config["Solar_Controller_Front"]["replace"] = array(
  "{:alnum}" => "([a-zA-Z0-9]+)",
);

$config["Solar_Controller_Front"]["rewrite"] = array(
  "page/{:alnum}/edit" => "page/edit/$1",
);

$config["Solar_Controller_Front"]["routing"] = array(
  "page" => "Other_App_Zim",
);
```

Front Controller

- Named actions

```php
<?php
$config["Solar_Controller_Front"]["rewrite"]["edit-page"] = array(
    'pattern' => 'page/{:id}/edit',
    'rewrite' => 'page/edit/$1',
    'replace' => array(
        '{:id}' => '(\d+)',
    ),
    'default' => array(
        '{:id}' => 88,
    );
);

// then in a view:
echo $this->namedAction('edit-page', array('id' => 70));
```

Page Controller

- Looks at remaining portion of URI path and picks action
  - /foo/bar/baz => Vendor_App_Foo
  - public function actionBar($param)
  - preRun(), preAction(), postAction(), postRun(), preRender(), postRender()

Package Highlights

- SQL adapters
- ORM system
- Form generation
- CLI tooling
- Auth, Role, Access

SQL Adapters

- Adapters for Mysql, Pgsql, Sqlite, Sqlite2, Oracle
- PDO-based, multiple `fetch*()` styles
- Standardized `insert()`, `update()`, `delete()`
- Abstracted LIMIT, column types, table creation, index creation, sequences, auto-increment, column description, index description

Prepared Statements

- Named placeholders and bound values

```
$sql = Solar::factory("Solar_Sql");

$stmt = "SELECT * FROM students WHERE name = :name";

$bind = array(
  "name"  => "Bob'; DROP TABLE students;--",
);

$list = $sql->fetchAll($stmt, $bind);
```

Query Builder

- Programmatically build complex SELECT statements

```
$select = Solar::factory("Solar_Sql_Select");
$select->from("table", array("col", "col", ...))
       ->join()       // leftJoin(), innerJoin()
       ->where("col = ?", $value)
       ->group()
       ->having()
       ->union()      // unionAll()
       ->order()
       ->limit()      // page(), paging()
       ->fetchAll(); // fetchOne(), fetchPairs(), etc
```

MysqlReplicated

- Adapter picks master or slave
- Switch to/from non-replicated with no code changes

```
$config["Solar_Sql_Adapter_MysqlReplicated"] = array(
  "host" => "master.db.example.com",
  // ...
  "slaves" => array(
    0 => array("host" => "slave1.db.example.com"),
    1 => array("host" => "slave2.db.example.com"),
  ),
);
```

ORM System

- TableDataGateway + DataMapper
  Model objects
- Record objects (with relateds)
- Collection of Record objects
- Relationship definition objects
- Catalog for model objects
- Uses underlying SQL layers

Model Setup

```
class Vendor_Model_Invaders extends Solar_Sql_Model {
  protected function _setup() {
    $this->_table_name = "invaders"; // default

    // reads columns from table, or can be stored in:
    // $this->_table_cols = array(...);

    // each record has these relateds:
    $this->_belongsTo("irk");
    $this->_hasOne("robot");
    $this->_hasMany("weapons");
    $this->_hasMany("invasions");
    $this->_hasManyThrough("planets", "invasions");
  }
}
```

Relationship Definitions

```
'merge'          => merge data at 'server' (DB) or 'client'  (PHP)
'native_by'      => 'wherein' or 'select' when matching natives
'native_col'     => native col to match against foreign col
'foreign_class'  => class name of the foreign model
'foreign_alias'  => alias of the foreign table name
'foreign_col'    => foreign col to match against the native col
'cols'           => fetch these foreign cols
'conditions'     => where or join-on conditions
'foreign_key'    => magic shorthand for the foreign key col
'order'          => how to order related rows

... plus fetch-time params and eager-fetch params.
```

Fetching

```php
$model = Solar_Registry::get("model_catalog");

$collection = $model->invaders->fetchAll(array(
  "where"    => array("type = ?" => "short"),
  "group"    => ...
  "order"    => ...
  "page"     => ...
  "paging"   => ...
  "eager"    => array("weapons", "robot", "planets"),
  ...
));

$array  = $model->invaders->fetchAllAsArray(array(...));

$record = $model->invaders->fetchOne(array(...));
```

Records

```php
$zim = $model->invaders->fetchOne(array(
  "where" = array("name = ?" => "Zim")
));

echo $zim->doom;
$zim->enemy = "Dib";

foreach ($zim->weapons as $weapon) {
  echo $weapon->name;
}

$zim->weapons->appendNew(array("name" => "raygun"));

$zim->save();
```

Record Filters

- Solar_Filter, Solar_Filter_Validate*, Solar_Filter_Sanitize* as generally-available classes, not regular expressions

- Vendor_Model_* will use Vendor_Filter_* automatically

```php
<?php
class Vendor_Model_Invaders extends Solar_Sql_Model_Record
{
  protected function _setup() {
    // ...
    $this->_addFilter("bar", "validateAlnum");
    $this->_addFilter("foo", "validateInList", array(
        "one", "two", "three",
    ));
    $this->_addFilter("baz", "validateCustomThing");
  }
}
```

Form Generation

```
$zim  = $model->invaders->fetch($id);
$form = $zim->newForm(); // Solar_Form object

$view = Solar::factory("Solar_View");
$html = $view->form()
             ->auto($form)
             ->addProcessGroup("save", "cancel");

echo $html;

// automatic CSRF protection
```

Much More Model

- Single-table inheritance
- Calculated and virtual columns
- Auto serializing of columns (arrays)
- Auto XML structs (EAV)
- Auto creation of tables and indexes
- Auto count-pages at fetch time

CLI Tooling

```
$ ./script/solar make-vendor Vendor

$ ./script/solar make-model Vendor_Model_Invaders

$ ./script/solar make-app Vendor_App_Zim --model-name=invaders

# make-docs, make-tests, run-tests, link-public, ...
```

CLI Controllers

- Independent "console" (front) and "command" (page) controllers
- Direct output with `_out()`, `_outln()`, `_err()`, `_errln()`
- VT100 escape codes built in
- Vendor-specific invocation
  - `./script/vendor command-name`

Auth, Role, Access

- Auth adapters (SQL, LDAP, email, etc.)
- Role adapters (File, SQL)
- Access adapters (File, SQL)
- User class is composed of these

Project Architecture

System Directories

```
system/
  config.php     # config file
  config/        # config support
  docroot/       # vhost document root
    index.php    # bootstrap
    public/      # copies or symlinks to public assets
  include/       # symlinks to PHP files
  script/        # command-line scripts
  source/        # actual PHP and support files
  sqlite/        # sqlite databases
  tmp/           # sessions, logs, caches, etc
```

Source vs. Include

```
include/          # single include-path for PHP files
  Solar/          # symlink to source/solar/Solar
  Solar.php       # symlink to source/solar/Solar.php
  Vendor/         # symlink to source/vendor/Vendor
  some.php        # symlink to source/other/some.php
  else.php        # symlink to source/other/else.php

source/           # all "real" files for a vendor
  solar/          # solar files
    config/
    Solar.php
    Solar/
      ...
  vendor/         # vendor files
    Vendor/
  other/          # random assortments of 3rd-party libs
    some.php
    else.php
```

Typical Page and View

```
framework/                  # libraries
application/                # separate include path
  controllers/
    FooController.php       # actions "bar" and "dib"
    ZimController.php       # extends Foo
  views/
    foo/
      bar.php
      dib.php
    zim/
      bar.php
      dib.php
  layouts/
    default.php
  public/
```

Solar Page And View

```
Solar/                  # libraries
Vendor/                 # same include path
  App/
    Foo.php             # actions "bar" and "dib"
    Foo/
      Layout/
        default.php
    Public/
    View/
      bar.php
      dib.php
  Zim.php               # extends Foo
  Zim/
    View/               # inherits Foo views
      dib.php           # override view
```

Performance Indicators

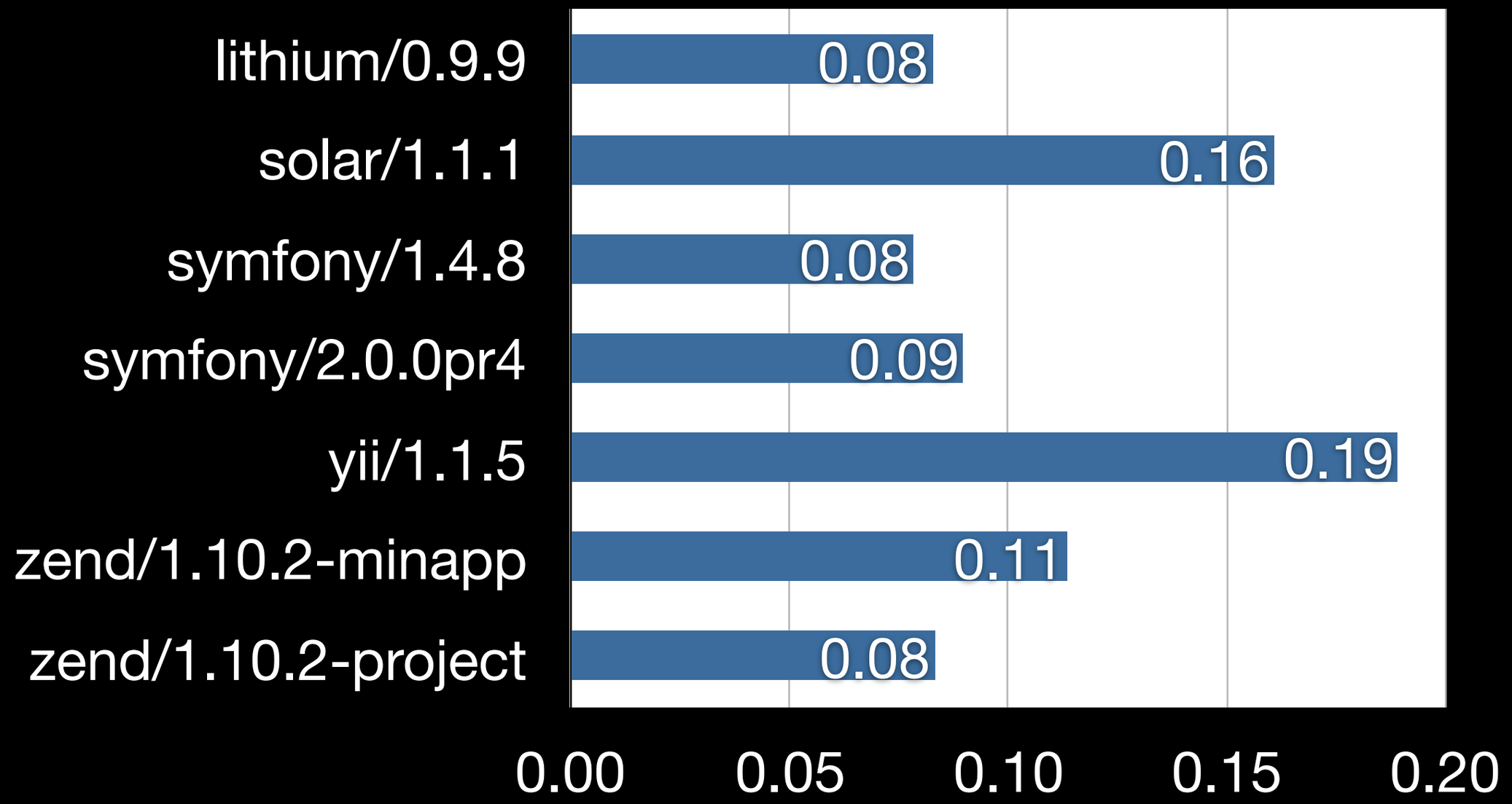
Benchmarks

- Google "web framework benchmarks"
- Amazon EC2 "Large" instance
- Apache, mod_php 5.3.2, APC
- 10 concurrent users, no sessions
- 5 runs of 1 minute each, pre-warmed cache

Results Table

Framework	relative	average
baseline/html	1.2823	3220.56
baseline/php	1.0000	2511.54
lithium/0.9.9	0.0827	207.62
solar/1.1.1	**0.1609**	**404.17**
symfony/1.4.8	0.0784	196.91
symfony/2.0.0pr4	0.0897	225.22
yii/1.1.5	0.1891	474.82
zend/1.10.2-minapp	0.1136	285.28
zend/1.10.2-project	0.0832	208.84

Results Graph



- Be suspicious of benchmarks
- Only one data-point in decision making
- Measures only what you test
- https://github.com/pmjones/php-framework-benchmarks

Conclusion

- Foundational concepts and techniques
- Dispatch cycle
- Package highlights
- Project architecture
- Performance indicators
- Stable, mature, proven