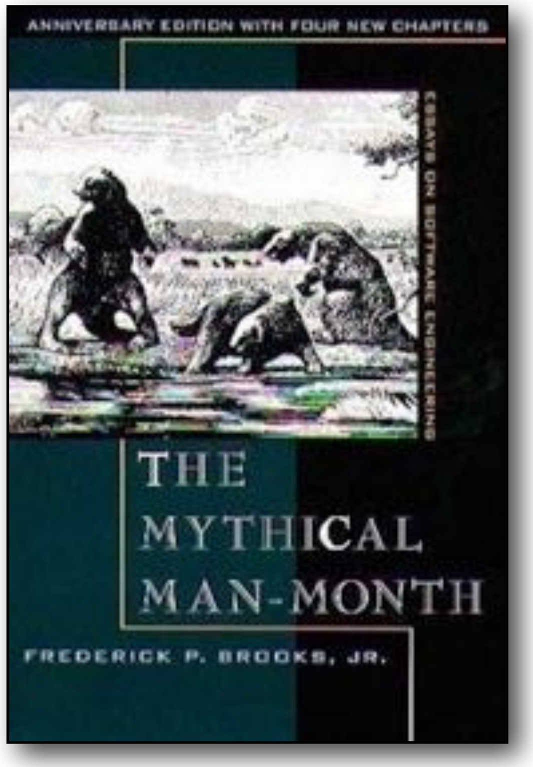
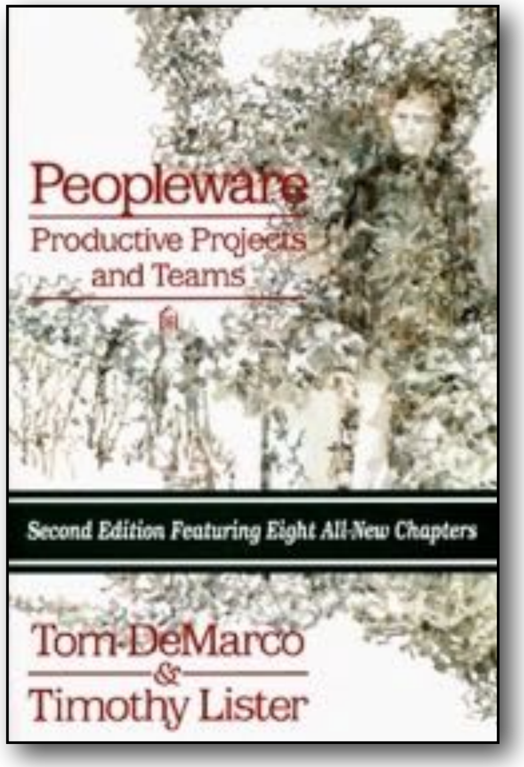
Organizing Your PHP Projects

Paul M. Jones

Read These

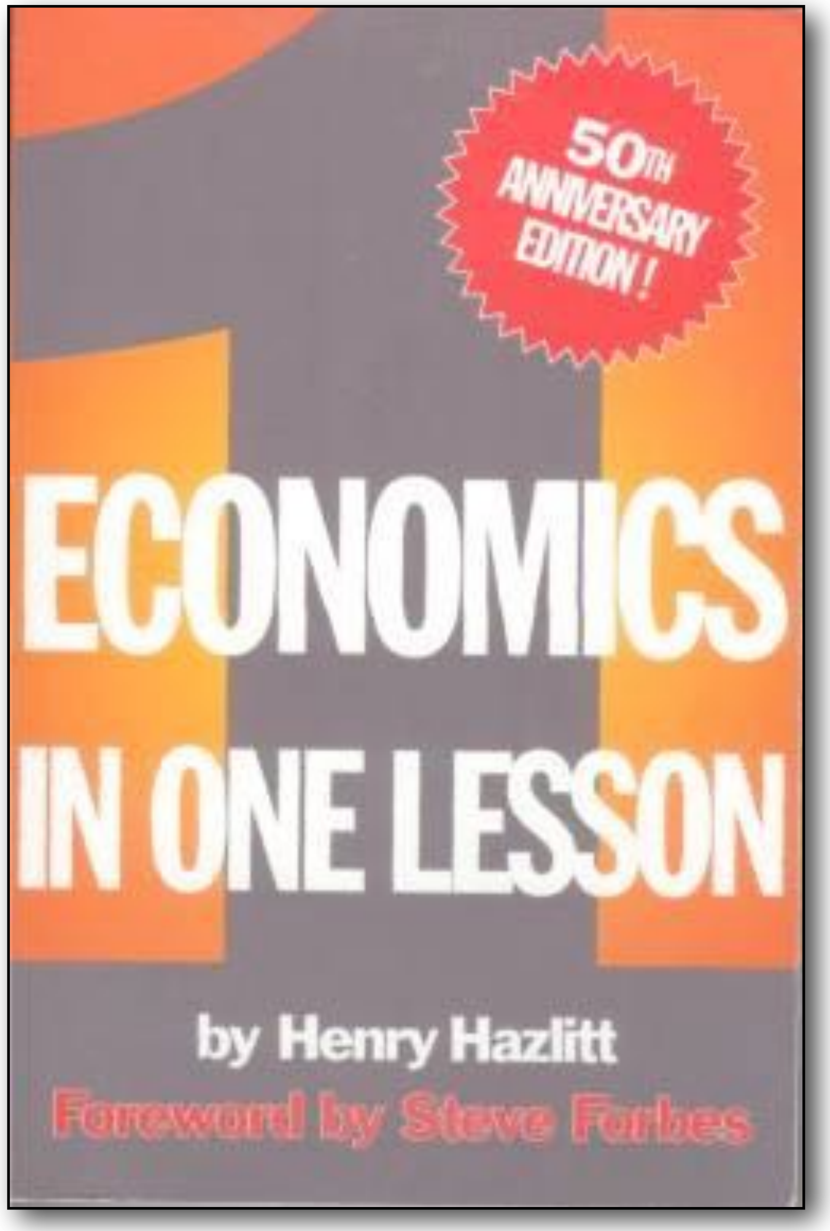
- "Mythical Man-Month", Brooks
- "Art of Project Management", Berkun
- "Peopleware", DeMarco and Lister


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.


Peopleware
Productive Projects and Teams
Second Edition Featuring Eight All-New Chapters
Tom DeMarco & Timothy Lister

Project Planning in One Lesson

- Examine real-world projects
- The One Lesson for organizing your project
- Elements of The One Lesson
- The One Lesson in practice



About Me

- Web Architect
- PHP since 1999 (PHP 3)
- Solar Framework (lead)
- Savant Template System (lead)
- Zend Framework (found. contrib.)
- PEAR Group (2007-2008)

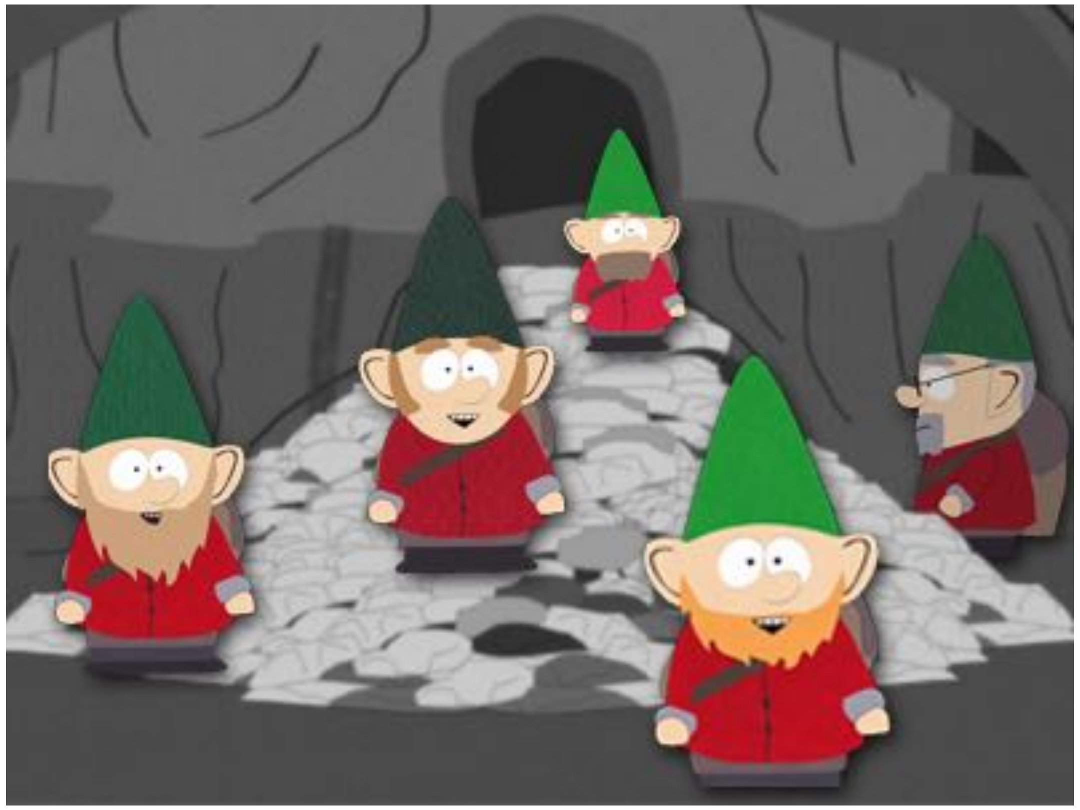
About You

- Project lead/manager?
- Improve team consistency?
- Want to share your code with others?
- Want to use code from others?
- Want to reduce

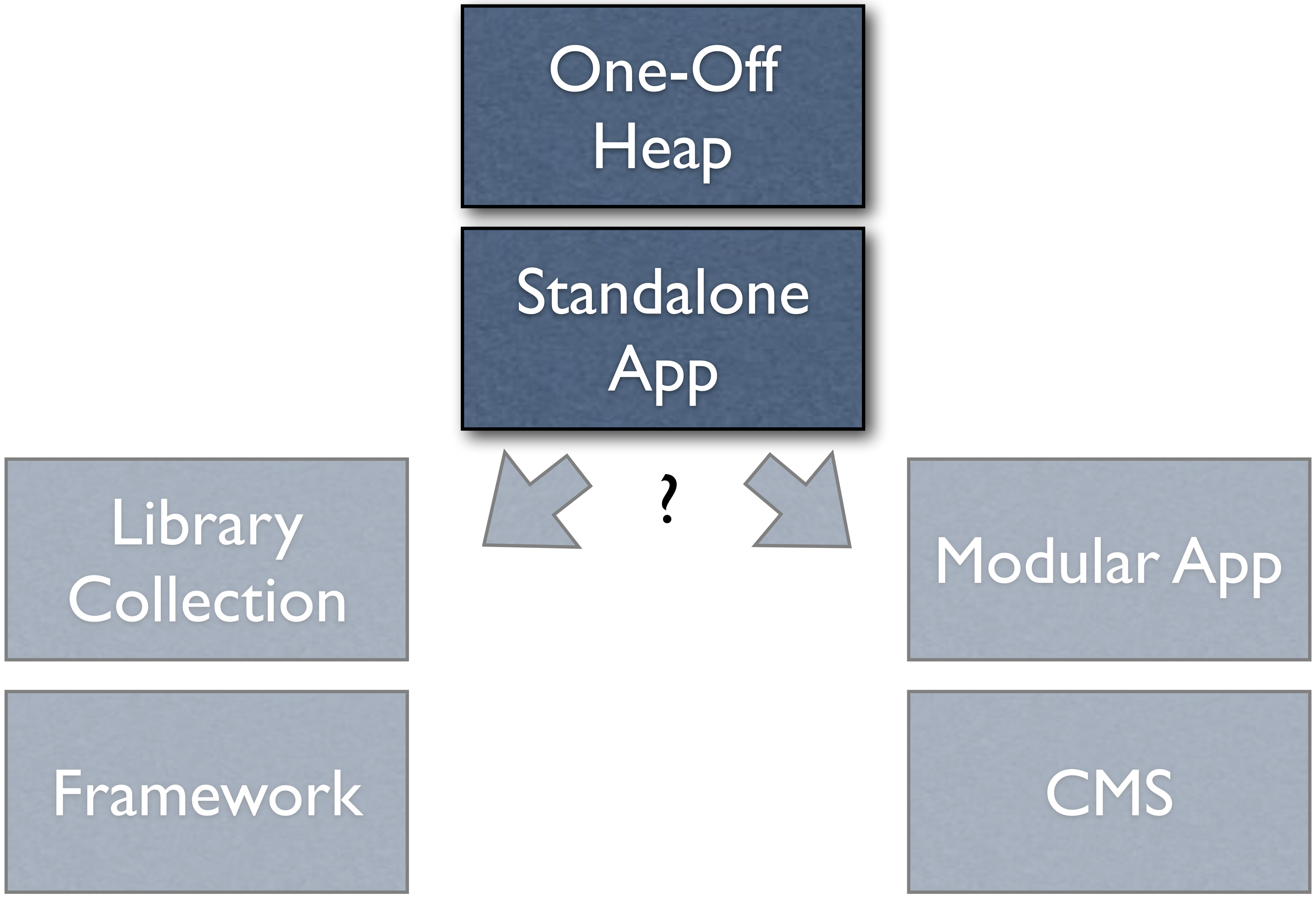
Goals for Organizing

- Security
- Integration and extension
- Adaptable to change
- Predictable and maintainable
- Teamwork consistency
- Re-use rules on multiple projects

Project Research;
or,
"Step 1: Study Underpinnings"



Project Evolution Tracks



One-Off Heap

- No discernible architecture
- Browse directly to the scripts
- Add to it piece by piece
- Little to no separation of concerns
- All variables are global
- Unmanageable, difficult to extend

Standalone Application

- One-off heap ++
- Series of separate page scripts and common includes
- Installed in web root
- Each responsible for global execution environment
- Script variables still global

Standalone Application: Typical Main Script

```php
// Setup or bootstrapping
define('INCLUDE_PATH', dirname(__FILE__) . '/');
include_once INCLUDE_PATH . 'inc/prepend.inc.php';
include_once INCLUDE_PATH . 'lib/foo.class.php';
include_once INCLUDE_PATH . 'lib/View.class.php';

// Actions (if we're lucky)
$foo = new Foo();
$data = $foo->getData();

// Display (if we're lucky)
$view = new View(INCLUDE_PATH . 'tpl/');
$view->assign($data);
echo $view->fetch('template.tpl');

// Teardown
include_once INCLUDE_PATH . "inc/append.inc.php";
```

Standalone Application: Typical Include File

```
// expects certain globals
if (! defined('APP_CONSTANT')) {
    die('Direct access not allowed.');
}
```

Standalone Application: Typical File Structure

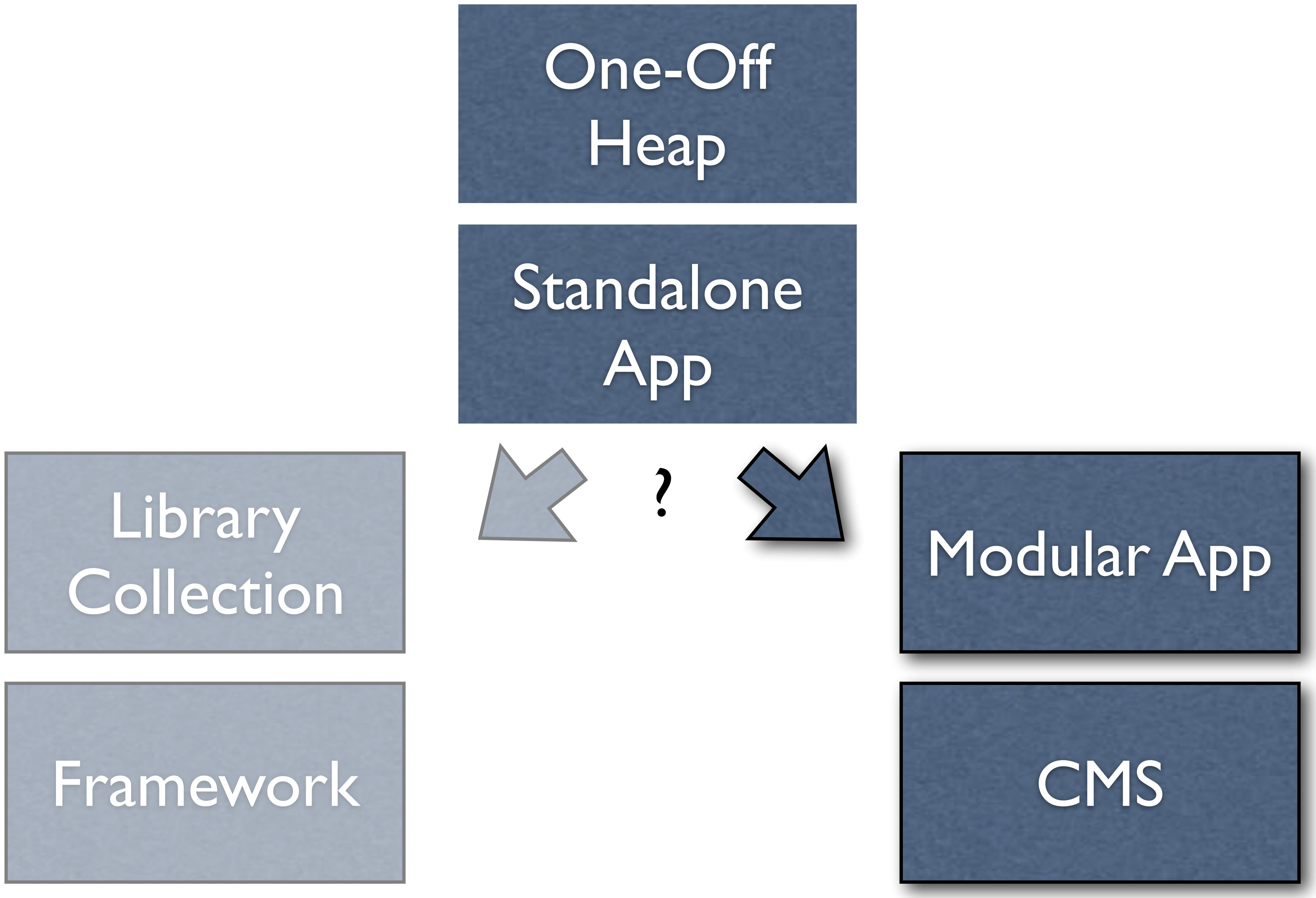
```
index.php                   # main pages
page1.php                   #
page2.php                   #
page3.php                   #
sub/                        # sub-section
    index.php               #
    zim.php                 #
    gir.php                 #
inc/                        # includes
    config.inc.php          #
    prepend.inc.php         #
lib/                        # libraries
    foo.class.php           #
    Bundle1/                #
    Bundle2/                #
```

Standalone Application: Support Structure

```
bin/                # command-line tools
cache/              # cache files
css/                # stylesheets
docs/               # documentation
img/                # images
install/            # installation scripts
js/                 # javascript
log/                # log files
sql/                # schema migrations
theme/              # themes or skins
tpl/                # templates

-- no standard naming or structure
-- index.html file in each directory
```

Project Evolution Tracks



Modular Application

- Standalone application ++
- Same file structure and script style
- One additional directory: "modules", "plugins", etc
- Hooks in the "main" scripts for additional behaviors
- Use global variables to coordinate between modules

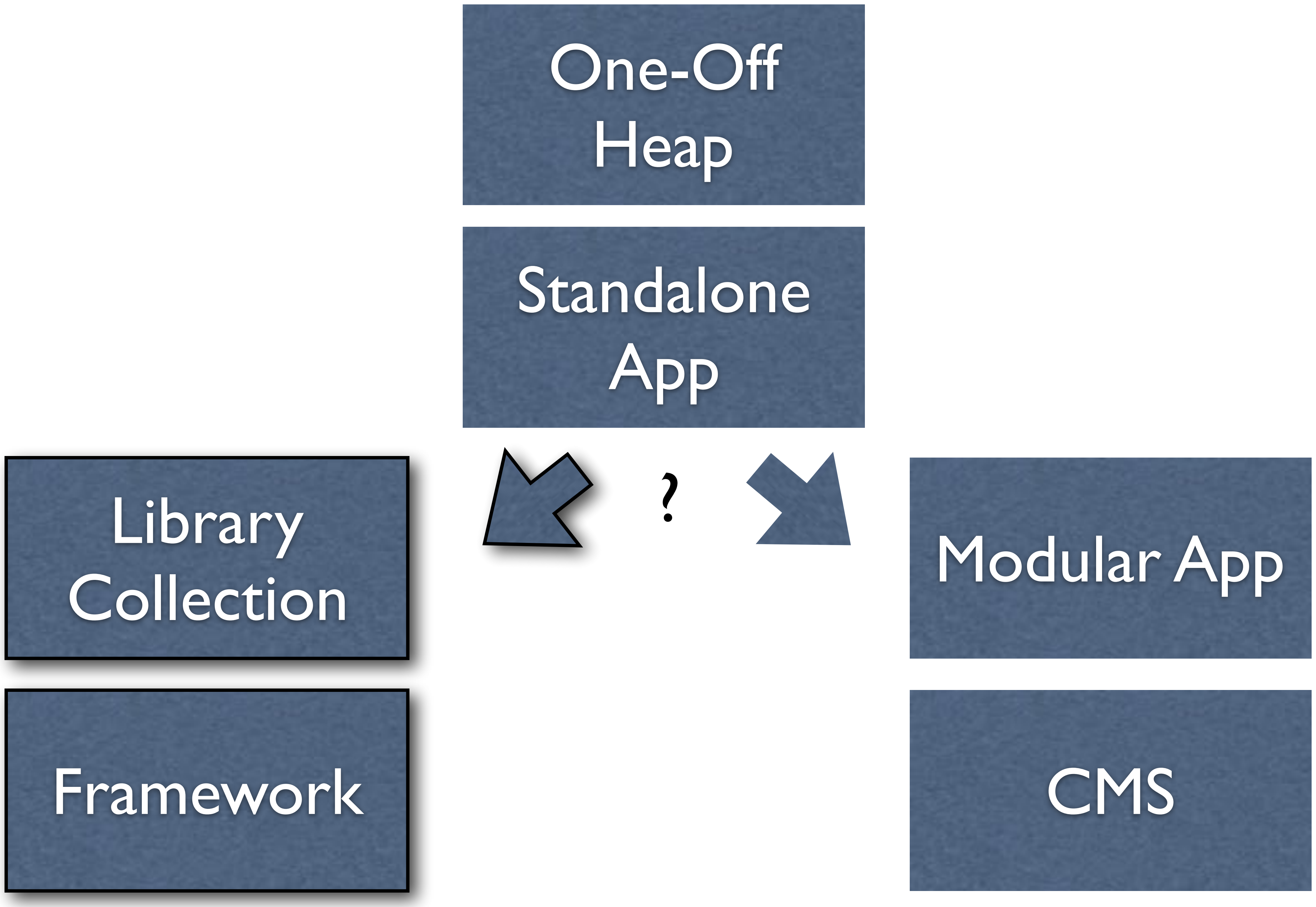
CMS

- Modular application ++
- General-purpose application broker
- All "main" scripts become sub-applications
- Still in the web root, still using globals to coordinate

Application/CMS Projects

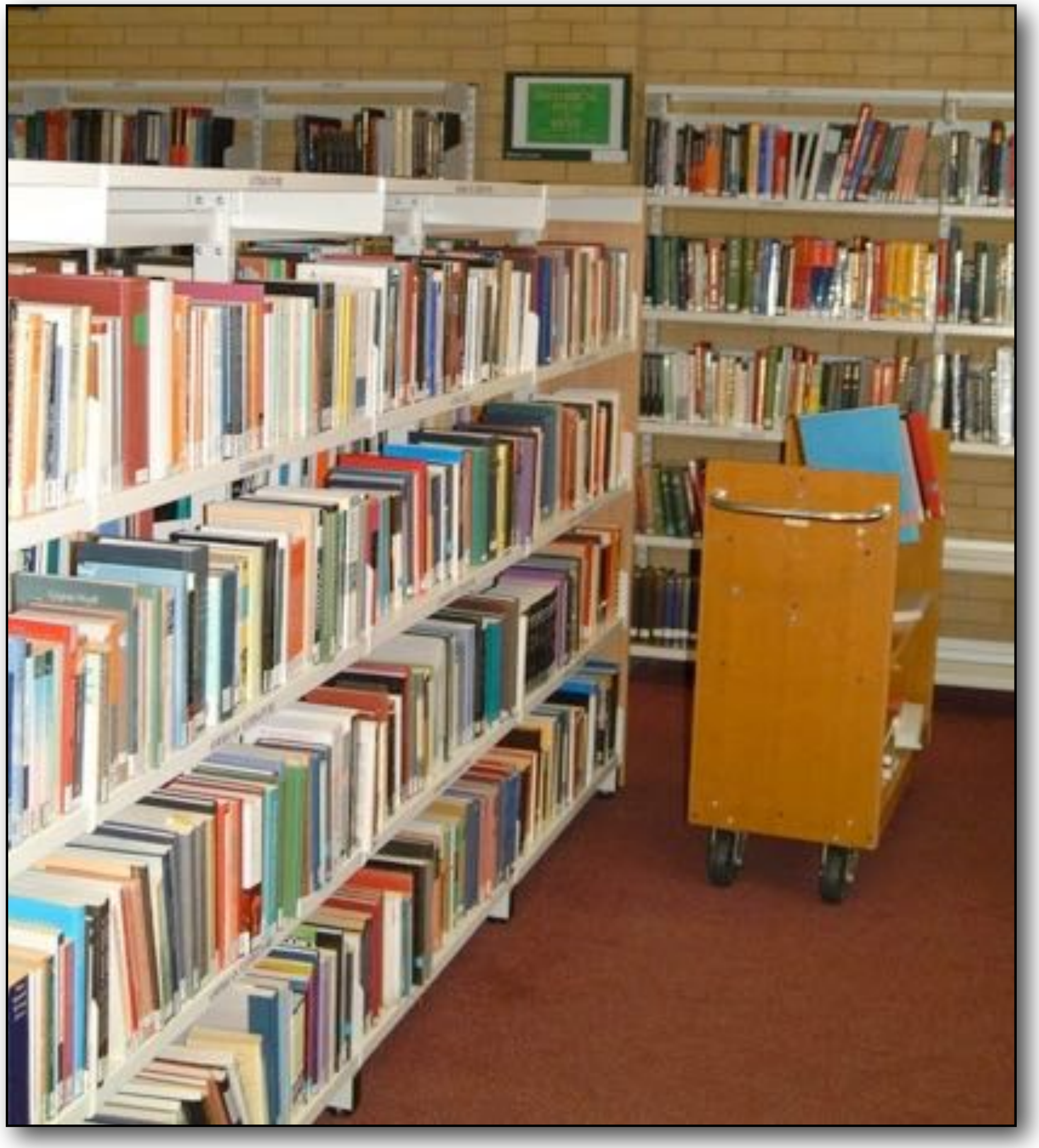
- Achievo
- Code Igniter*
- Coppermine
- DokuWiki
- Drupal
- Eventum
- Gallery
- Joomla/Mambo
- MediaWiki
- PhpMyAdmin
- Seagull*
- SugarCRM

Project Evolution Tracks



Library Collection

- Specific, limited logic extracted from an app
- Re-used directly in unrelated applications and other libraries
- No global variables
- Class-oriented
- Can exist anywhere in the file system

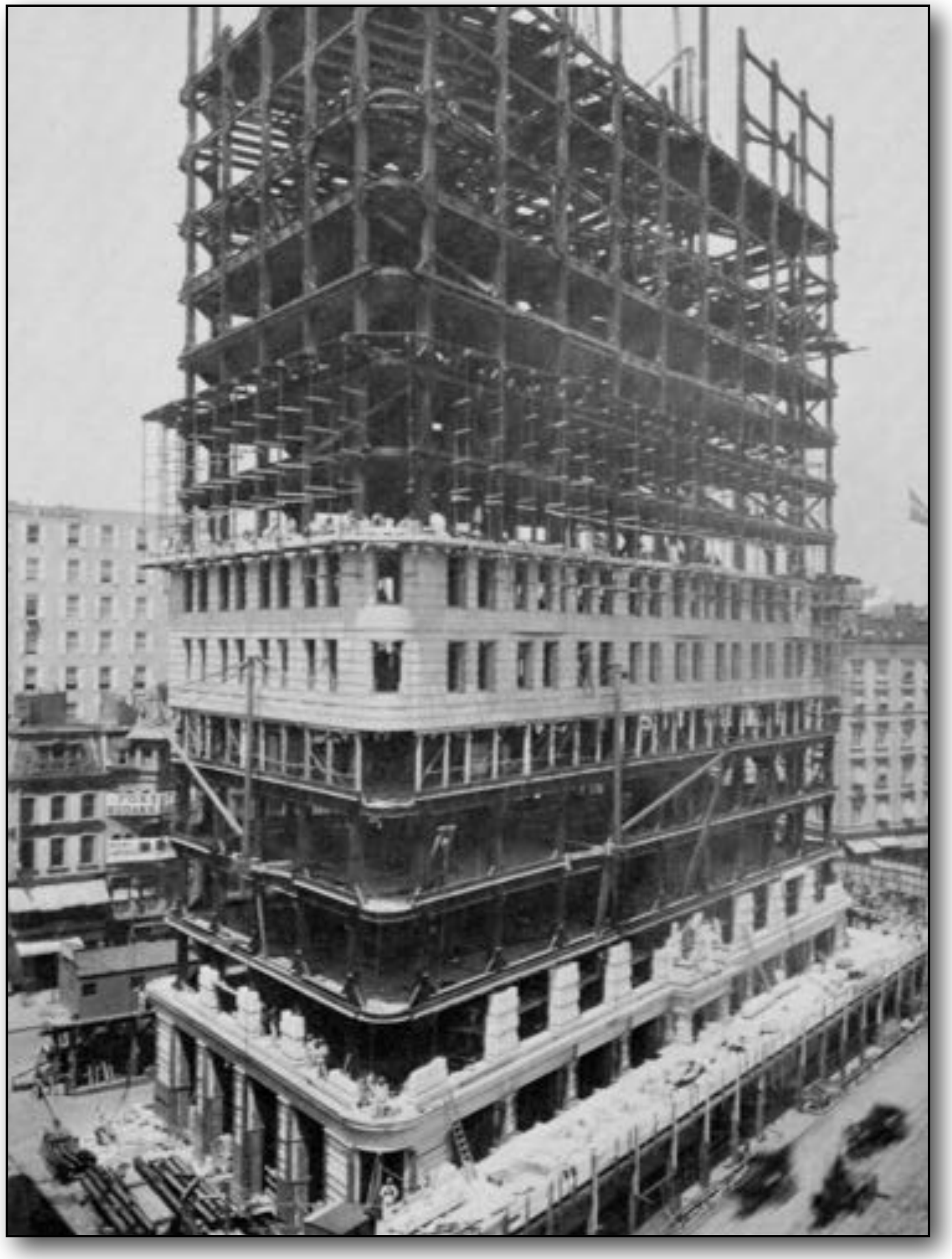


Library Project: Typical File Structure

```
Foo.php                       # Foo
Foo/                          #
    Component.php             # Foo_Component
    Component/                #
        Element1.php          # Foo_Component_Element1
        Element2.php          # Foo_Component_Element2
Bar.php                       # Bar
Bar/                          #
    Task.php                  # Bar_Task
    Task/                     #
        Part1.php             # Bar_Task_Part1
        Part2.php             # Bar_Task_Part2
```

Framework

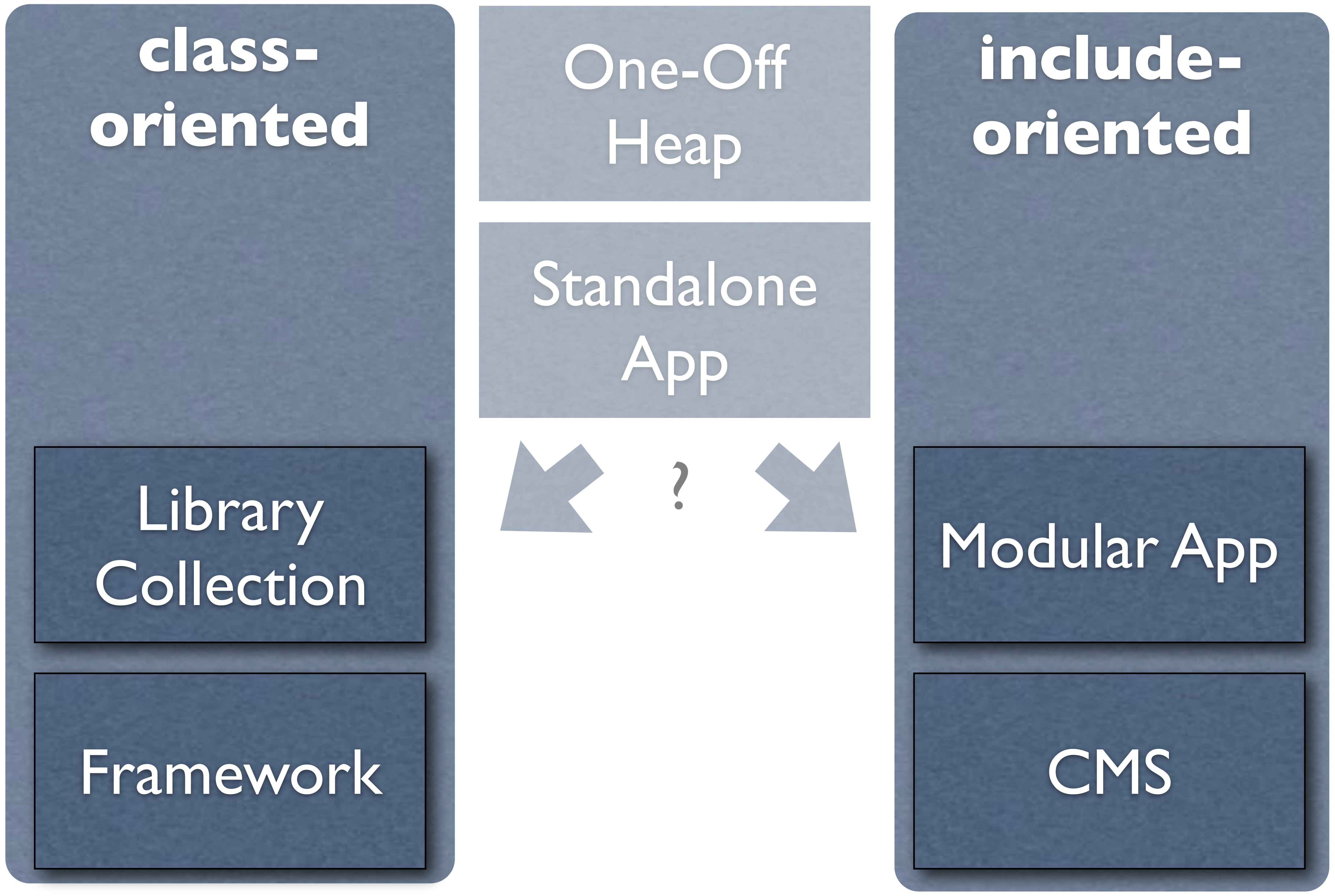
- Codebase
  - Library collection
  - Apps extend from it
- Support structure
  - Bootstrap file
  - Public assets
  - Protected assets



Library/Framework Projects

- AdoDB
- Cake
- **CgiApp**
- Code Igniter *
- **Doctrine**
- EZ Components
- **HtmlPurifier**
- **Horde**
- Lithium
- Mojavi/Agavi
- **PAT**
- **PEAR**
- **PHP Unit**
- Phing
- **Phly**
- Prado
- **Savant**
- **Seagull** *
- **Smarty**
- **Solar**
- **SwiftMailer**
- Symfony
- WACT
- **Zend**

Project Evolution Tracks



The One Lesson;
or,
“Step 2: … ?”

Organize your project

as if

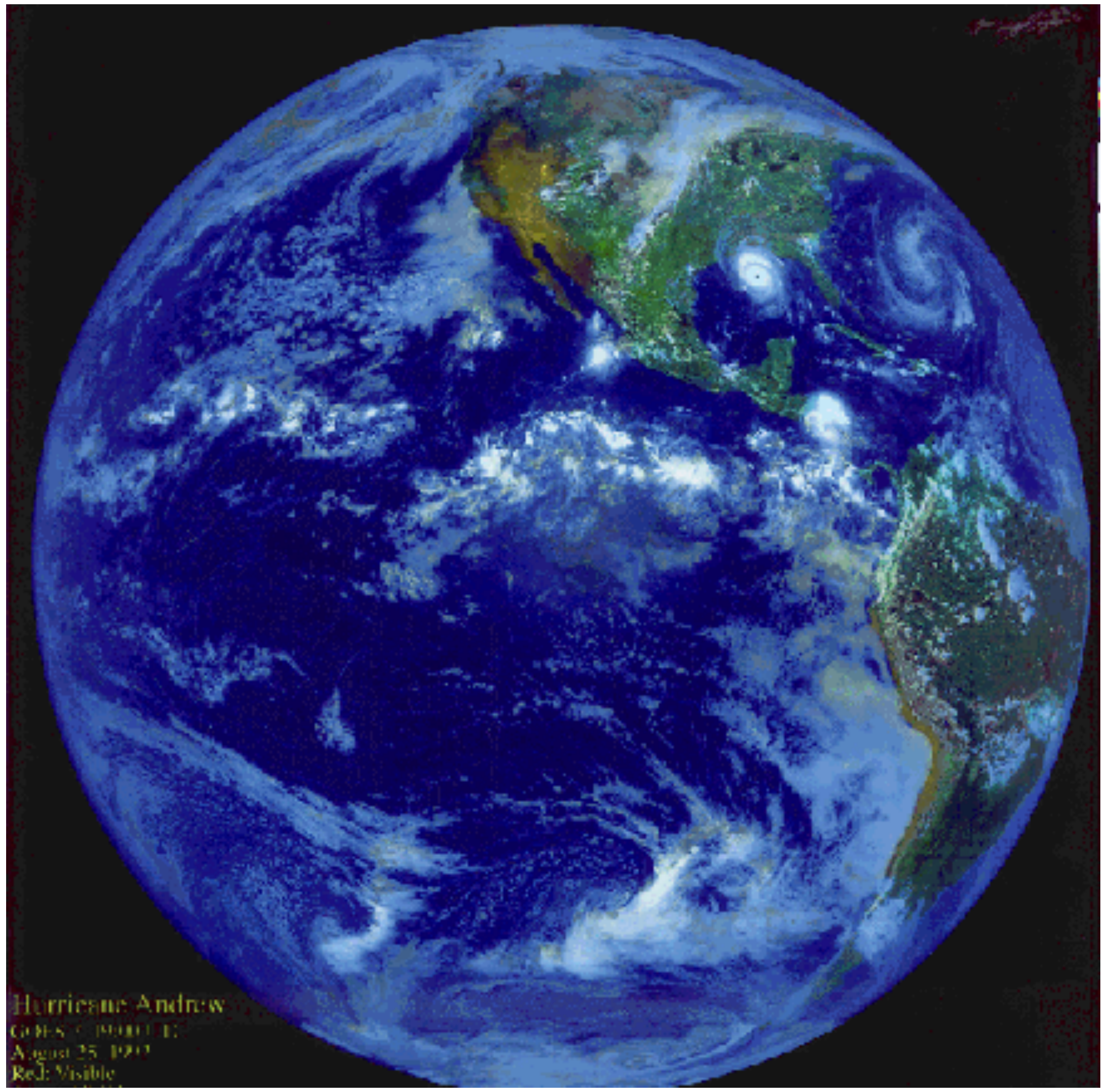
it is a library collection.

Elements of The One Lesson

- Stop using globals
- Namespace everything
- Class-to-file naming

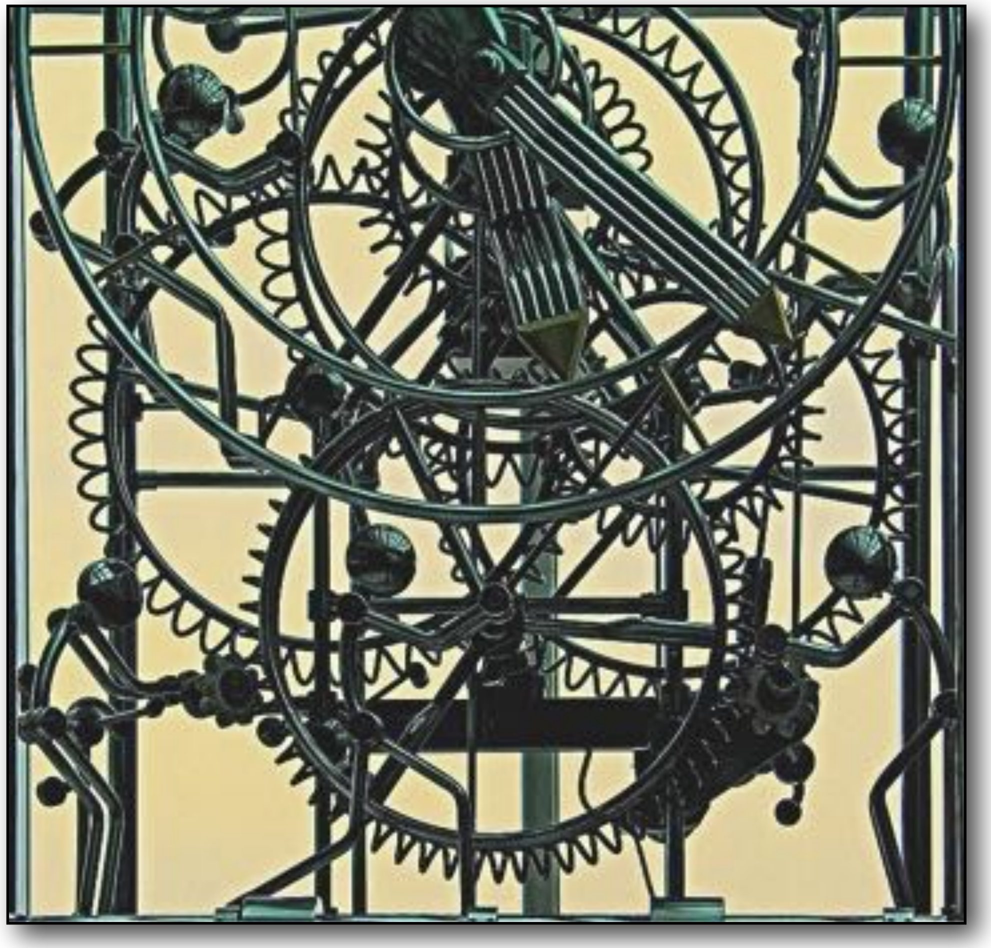
1. Stop Using Globals

- Stop using **`register_globals`**
- Stop using **`$GLOBALS`**
- Stop using **`global`**



2. Namespace Everything

- Automatic deconfliction of identifiers
    - Classes (“vendor”)
    - Functions, variables, constants
    - Use with `$_SESSION`, `$_COOKIE`, etc. keys



Choosing a Namespace

- Project, client, brand, channel
- A short word or acronym, not a letter (“Z”)
- A unique name, not a generic name related to a task (Date, HTML, RSS, Table, User)

PHP 5.2 “Namespaces”

```
// class User {}
class Vendor_User {}
$user = new Vendor_User();

// function get_info() {}
function vendor_get_info()

// $_SESSION["user_prefs"]
$_SESSION["Vendor_User"]["prefs"];
```

PHP 5.3 Namespaces

```php
namespace vendor;
class User {}

// relative namespace
namespace vendor;
$user = new User();

// absolute namespace
namespace other;
$user = new \vendor\User();
```

3. Class-To-File Naming

- Class name maps directly to file name
  - `Vendor_User => Vendor/User.php`
- Horde, PEAR, Solar, Zend, others
- Highly predictable file locations
- Lends itself to autoloading

Class-to-File Naming (PHP 5.2, Horde/PEAR)

```
// studly-caps needs preg_replace(), but:
VendorAuthOpenId => ...
    Vendor/Auth/Open/Id.php?
    Vendor/Auth/OpenId.php?

// underscores just need str_replace()
Vendor_Auth_OpenId => Vendor/Auth/OpenId.php
```

Class-to-File (PHP 5.3, PSR-0)

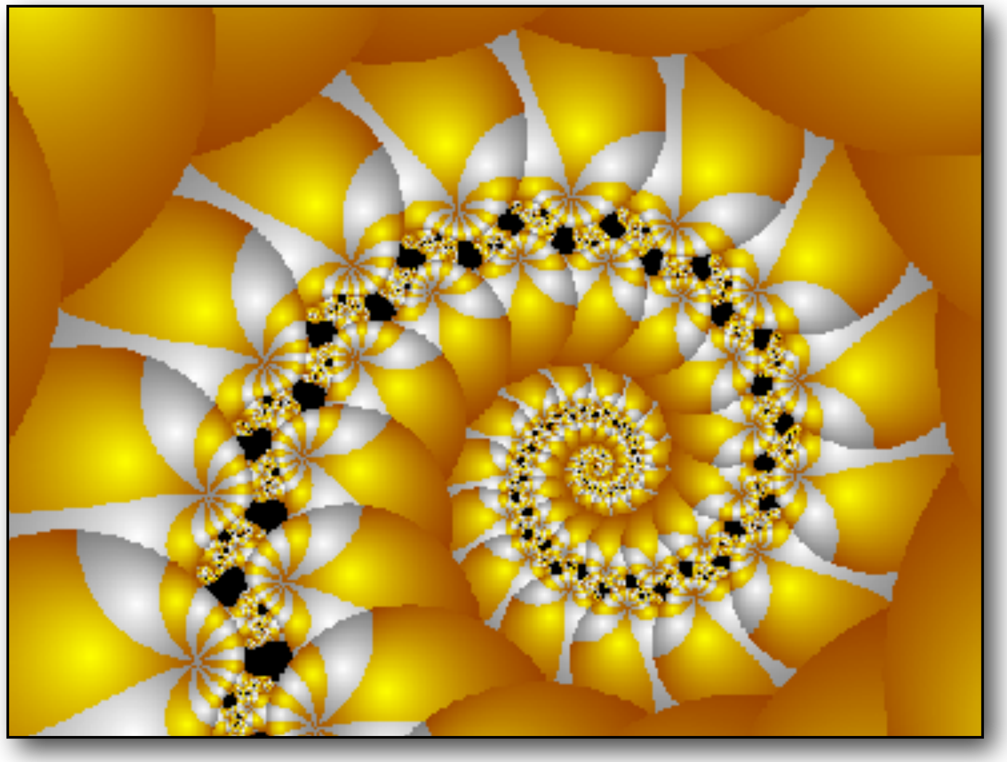
```
\foo_bar\pkg\Main     => /foo_bar/pkg/Main.php
\foo_bar\pkg\Main_Sub => /foo_bar/pkg/Main/Sub.php
```

- PEAR, Solar, Zend, Doctrine, Lithium, Symfony

The One Lesson In Practice; *or,* *"Step 3: Profit!"*

Extended Effects of The One Lesson

- Can be used anywhere (app, module, lib, CMS, framework)
- Structure for refactoring and additions
- Testing, profiling, and public files can parallel the same structure
- Intuitive for new developers
- No more include-path woes



Mambo CMS

```
administrator/
components/
editor/
files/
help/
images/
includes/
    Vendor/
index.php
installation/
language/
mainbody.php
mambots/
media/
modules/
templates/
```

Zend Framework

```
project/
    application/
        bootstrap.php
        configs/
        controllers/
        models/
        views/
            helpers/
            scripts/
    library/
        Zend/
        Vendor/
    public
        index.php
```

Lithium

```
app/
    config/
    controllers/
    extensions/
    index.php
    libraries/
    models/
    resources/
    tests/
    views/
    webroot/
libraries/
    lithium/
    vendor/
```

Symfony 2

```
hello/
    config/
    console/
    HelloKernel.php
src/
    Application/
        HelloBundle/
            Bundle.php
            Controller/
            Resources/
    autoload.php
    vendor/
        symfony/
        zend/
        vendor/
web/
```

Solar

```
system/
    config/
    config.php
    docroot/
        index.php
        public/
    include/
        Solar.php
        Solar/
        Vendor/
    script/
    source/
    sqlite/
    tmp/
```

Solar Apps Are Libraries Too

```
include/
  Solar/
  Vendor/
    App/
      Page.php
      Page/
        Layout/
        Locale/
        Public/
        View/
    Model/
      Gir.php
      Gir/
    Zim.php
    Zim/
```

Refactoring

- Move from existing include-based architecture
  to class-based architecture ...
  - ... by functionality
  - ... by script
- Then build scripts out of classes, not includes
- Then build apps out of classes, not scripts
- Leads to MVC / MVP / PAC architecture

Summary

The One Lesson

- Organize your project as if it will be part of a library collection
  - Avoid globals
  - Use namespaces for deconfliction
  - Use class-to-file naming convention

Goals for Organizing

- Security
- Integration and extension
- Adaptable to change
- Predictable and maintainable
- Teamwork consistency
- Re-use rules on multiple projects

- Questions?
- Comments?
- Criticism?

Thanks!

- <http://paul-m-jones.com>
- <http://solarphp.com>
- Google for “web framework benchmarks”