Object-Oriented Programming for PHP Application Servers

Sebastian Bergmann
<http://www.sebastian-bergmann.de/>

International PHP Conference 2003 – Spring Edition
Amsterdam, May, 9th 2003

Please bear with me ...

- Maybe I should begin this presentation with: „*My written English is better than my spoken English.*“.
- This is the first time I give a presentation in English. Please bear with me ;-)
- Feel free to ask questions at any time!

This Presentation's Motto

- *„I amar prestar aen, han mathon ne nen, han mathon ne chae a han noston ned 'wilith.“*
- *„The world is changed; I can feel it in the water, I can feel it in the earth, I can smell it in the air.“*
- With the advent of PHP 5 and SRM new possibilities present themselves to PHP application developers.
- This session will give you an overview of this „changed world of PHP“.

Namespaces

```php
<?php
namespace aNamespace {
    const aConstant = 'constant value';

    class aClass {}

    function aFunction() {}
}
?>
```

Constants and Static Members

```php
<?php
class Test {
    const  constant = 'constant';
    static $static  = 1;

    public function inc() {
        return self::$static++;
    }
}

$a = new Test;
$b = new Test;

echo 'Test::constant = ' . Test::constant . "\n";
echo '$a->inc()      = ' . $a->inc()      . "\n";
echo '$b->inc()      = ' . $b->inc()      . "\n";
?>
```

```
Test::constant = constant
$a->inc()      = 1
$b->inc()      = 2
```

Static Methods

```php
<?php
class Test {
    public static function staticMethod() {
        echo "Test::staticMethod() called.\n";
    }
}

Test::staticMethod();
?>
```

```
Test::staticMethod() called.
```

References

```php
<?php
class Test {}

$a = new Test;
$b = $a;

if ($a === $b) {
    echo '$a and $b reference the same object';
}
?>
```

```
$a and $b reference the same object
```

Object Cloning

```php
<?php
class Test {}

$a = new Test;
$b = $a->__clone();

if ($a !== $b) {
    echo '$a and $b reference not the same object';
}
?>
```

```
$a and $b reference not the same object
```

Object Cloning

```php
<?php
class Connection {
    private $connection;
    private $host;
    private $database;
    private $user;
    private $pass;

    public function __construct($host, $database, $user, $pass) {
        // ...
    }

    public function __clone() {
        $this->host     = $that->host;
        $this->database = $that->database;
        $this->user     = $that->user;
        $this->pass     = $that->pass;

        $this->connect();
    }

    public function connect() {
        // ...
    }
}
?>
```

Abstract Classes

```php
<?php
abstract class AbstractClass {
    abstract public function test();
}

class ImplementedClass extends AbstractClass {
    public function test() {
        echo 'ImplementedClass::test() called.';
    }
}

$o = new ImplementedClass;
$o->test();
?>
```

```
ImplementedClass::test() called.
```

Interfaces and Class Type Hints

```php
<?php
interface Printable {
    public function print();
}

interface Serializable {
    public function readObject($file);
    public function writeObject($file);
}

class Example implements Printable, Serializable {
    // ...
}

class AnotherClass {
    public function printObject(Printable $object) {
        $object->print();
    }
}
?>
```

Exceptions

```php
<?php
class FooException extends Exception {}

class MyApplication {
    public static function main() {
        try {
          if ($foo) {
            throw new FooException;
          }

          else if ($bar) {
            throw new Exception;
          }
        }

        catch (FooException $e) {
          // ...
        }

        catch ($e) {
          // ...
        }
    }
}
?>
```

Unit Tests with PHPUnit

- „I have no time to test my software.“
- „Software Testing is boring and stupid.“
- „My code is free of bugs, at the least it is good enough.“
- „The testing department tests the software. They're better at this than I am.“
- **These often made assumptions lead to a vicious circle that is very hard to break!**

Unit Tests with PHPUnit

- The *Extreme Programming* software process breaks this vicious circle by demanding tests to be written **before** the actual code.
- These tests also come in handy when the code is **refactored**.
- Effective testing is almost synonymous with effective programming.
- PHP has available an excellent testing suite called PHPUnit.
- PHPUnit aims at having the feature set of JUnit, the standard testing suite in the Java world.

Unit Tests with PHPUnit

- **Test Case**: Tests a method of a class by comparing the **actual** result of a method call for a given set of parameters with an **expected** result.
- The comparison of actual and expected results is done using **Assertion Methods**.
- A **Test Suite** is a collection of Test Cases.

PHPUnit Assertion Methods

- assertEquals($expected, $actual, $message = '', $delta = 0)
- assertNotNull($object, $message = '')
- assertNull($object, $message = '')
- assertSame($expected, $actual, $message = '')
- assertNotSame($expected, $actual, $message = '')
- assertTrue($condition, $message = '')
- assertFalse($condition, $message = '')
- assertRegExp($expected, $actual, $message = '')
- assertType($expected, $actual, $message = '')

Unit Tests with PHPUnit

```php
<?php
require 'phpOpenTracker.php';
require 'PHPUnit.php';

class phpOpenTracker_Test extends PHPUnit_TestCase {
    function testPageImpressions() {
        $this->assertEquals(
          6,

          phpOpenTracker::get(
            array(
             'api_call' => 'page_impressions'
            )
          )
        );
    }

    // ...
}

$result = PHPUnit::run(
  new PHPUnit_TestSuite('phpOpenTracker_Test')
);

echo $result->toString();
?>
```

Design Patterns

- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" (Christopher Alexander)
- "Even though Alexander was talking about patterns in buildings and towns, what he says is true about object-oriented design patterns. Our solutions are expressed in terms of objects and interfaces instead of walls and doors, but at the core of both kinds of patterns is a solution to a problem in a context." (Gamma, Helm, Johnson, Vlissides)

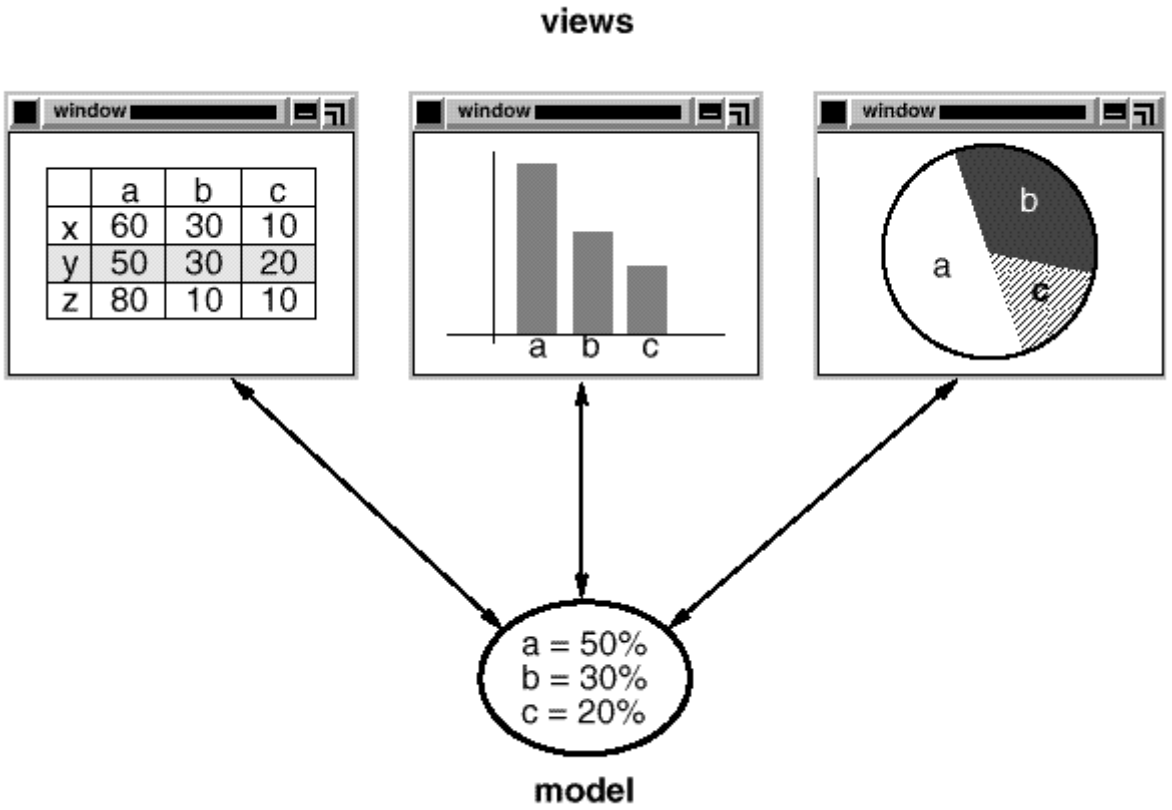
Design Patterns

- In general, a pattern has four essential elements:
- A **name** that identifies the patterns and eases its discussion with other developers.
- A description of the **problem**.
- A **solution** of the problem.
- The **consequences** (results, trade-offs, ...) of the pattern.

Design Patterns

- The software developer's job shifts from *re-inventing the wheel* to *choosing the right wheel*.
- **Patterns are no algorithms!** Algorithms solve fine-grained problems like sorting and have a lesser degree of freedom regarding their implementation than patterns.
- **Patterns are no frameworks!** Frameworks exist as ready-to-reuse code, patterns only contain code examples.

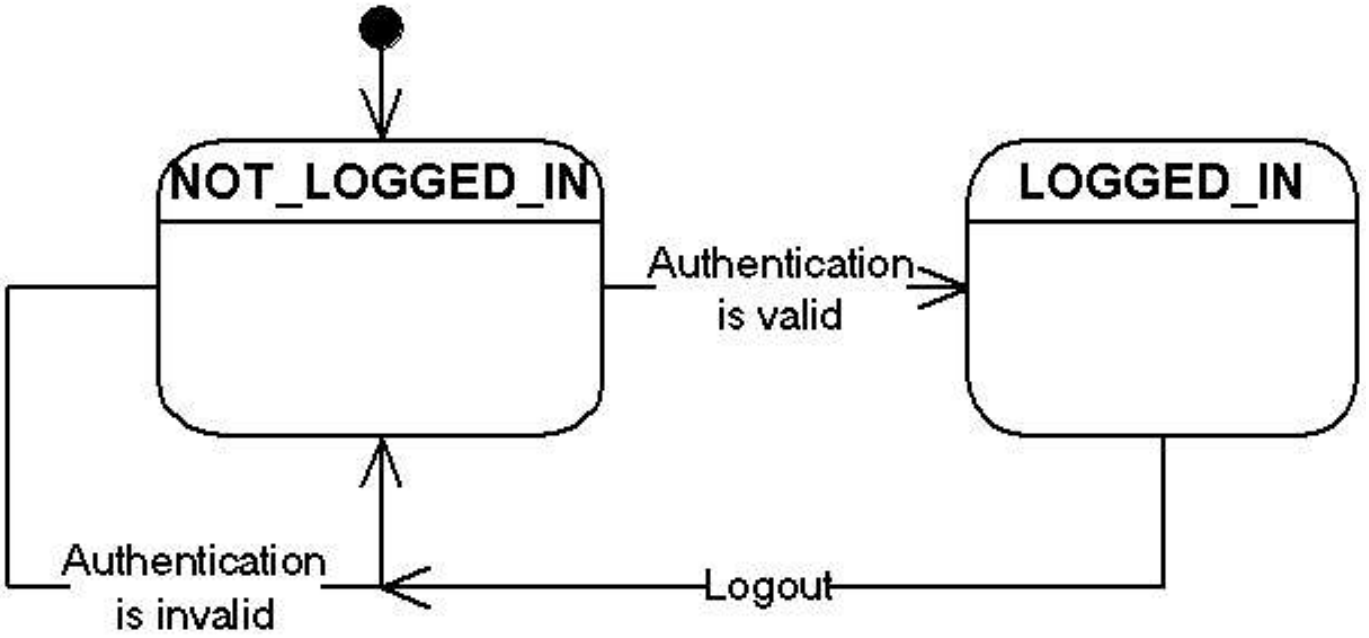
Model, View, Controller



Using a Finite State Machine For The Controller Component

- A **model of computation** consisting of a set of **states**, a **start state**, an input alphabet, and a **transition function** that maps input symbols and current states to a next state.
- The **State Charts** of the Unified Modeling Language (UML) have Finite State Machines as their underlying principle.
- The PEAR::FSM package provides a convenient implementation of Finite State Machines in PHP.

Using a Finite State Machine For The Controller Component



Using a Finite State Machine For The Controller Component

```php
<?php
require_once 'FSM.php';

class Controller extends FSM {
    private $model;
    private $stack = array();

    public function __construct($model) {
        $this->model = $model;

        parent::__construct('NOT_LOGGED_IN', $this->stack);

        $this->addTransition(
          'LOGIN',
          'NOT_LOGGED_IN',
          'NOT_LOGGED_IN',
          array($this, 'loginCheck')
        );

        $this->addTransition(
          'LOGOUT',
          'LOGGED_IN',
          'NOT_LOGGED_IN',
          array($this, 'logout')
        );
    }

    public function loginCheck() {
        $login = $this->model->checkLogin(
          $_REQUEST['username'],
          $_REQUEST['password']
        );

        if ($login) {
            return 'LOGGED_IN';
        }
    }

    public function logout() {
        $this->model->logout();
    }
}
?>
```

Using the XML Transformer For The View Component

```php
<?php
require_once 'XML/Transformer/Namespace.php';

class View extends XML_Transformer_Namespace {
    private $model;

    public function __construct($model) {
        $this->model = $model;
    }

    public function start_loginForm($attributes) {
        return '';
    }

    public function end_loginForm($cdata) {
        if (!$this->model->loggedIn) {
            return '<form action="index.php" method="post">' .
                   '<input type="hidden" name="action" value="LOGIN" />' .
                   '<input type="text" name="username" />' .
                   '<input type="password" name="password" />' .
                   '<input type="submit" value="login" />' .
                   '</form>';
        } else {
            return '<a href="index.php?action=LOGOUT">logout</a>';
        }
    }

    public function start_loggedIn($attributes) {
        return '';
    }

    public function end_loggedIn($cdata) {
        return $this->model->loggedIn ? 'yes' : 'no';
    }
}
?>
```

The Login Model

```php
<?php
class Model {
    private $loggedIn = false;

    public function checkLogin($username, $password) {
        $this->loggedIn = true;
        return true;
    }

    public function logout() {
        $this->loggedIn = false;
    }
}
?>
```

```
<?php
require_once 'Model.php';
require_once 'View.php';
require_once 'Controller.php';
require_once 'XML/Transformer/Driver/OutputBuffer.php';

session_register('model');
session_register('controller');

if (!isset($_SESSION['model'])) {
    $_SESSION['model']      = new Model;
    $_SESSION['controller'] = new Controller($_SESSION['model']);
}

$_SESSION['controller']->process(
  isset($_REQUEST['action']) ? $_REQUEST['action'] : 'DEFAULT'
);

$t = new XML_Transformer_Driver_OutputBuffer(
  array(
    'overloadedNamespaces' => array(
       '&MAIN' => new View($_SESSION['model'])
    )
  )
);
?>
<html><body><p>
Logged in: <b><loggedIn /></b></p>
<p><loginForm /></p></body></html>
```

Design Patterns

- The *Model-View-Controller* approach to application design is a **meta-pattern** as it motivates other patterns.
- For instance, the communication between the Model object and the View objects is commonly implemented using the **Observer** pattern.
- The Model, of which only one instance is needed, is commonly implemented using the **Singleton** pattern.

Singleton

- Belongs to the family of **Creational Patterns** that deal with the creation of object.
- The Singleton pattern ensures a class only has **one instance**, and provides a **global point of access** to it.
- Usually used for classes that handle / represent resources that should be globally accessable.
- Can be implemented with out-of-the-box PHP only on a per-request basis.

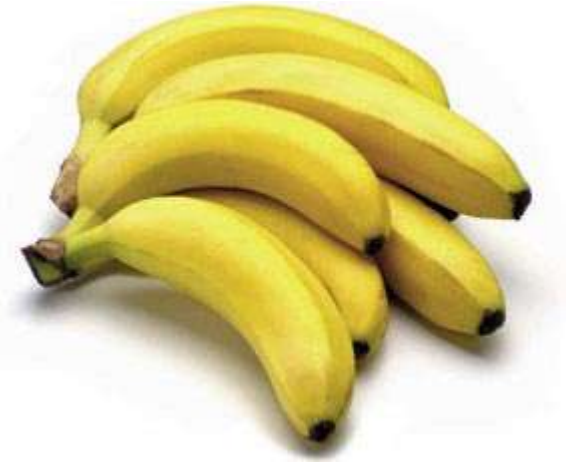
Singleton

```php
<?php
class Singleton {
    private static $instance = null;

    private function __construct() {}

    public static function getInstance() {
        if (self::$instance == null) {
            echo "Creating new object.\n";

            self::$instance = new Singleton;
        }

        return self::$instance;
    }
}

$a = Singleton::getInstance();
$b = Singleton::getInstance();

if ($a === $b) {
    echo '$a and $b reference the same object.' . "\n";
}
?>
```

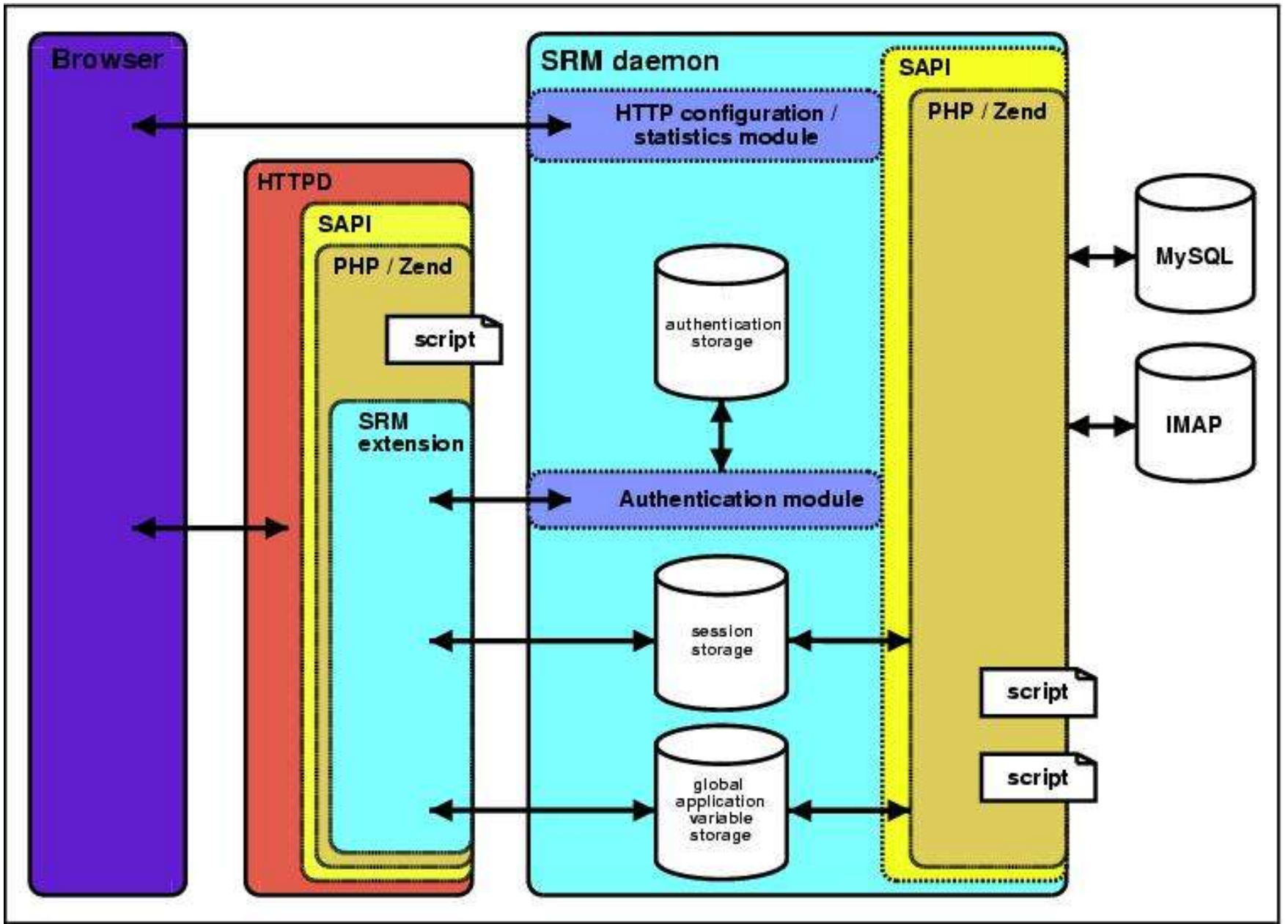
```
Creating new object.
$a and $b reference the same object.
```

Script Running Machine (SRM)

- Daemon that – running as a co-process to the Web Server – manages persistent objects and resources.
- Bridge between PHP scripts and persistent PHP objects.
- As an analogy to Java Beans these persistent PHP objects are called ...

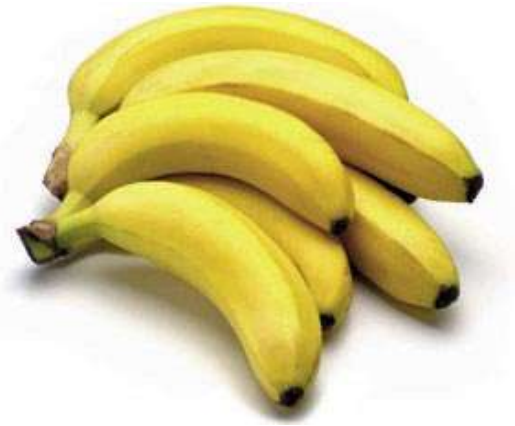


Script Running Machine (SRM)



PHP Bananas

- Persistent PHP objects.
- Compiled and initialized only once.
- Running in separate threads, managed by the SRM daemon.
- A PHP Banana is automatically a Singleton.



PHP Bananas

- A PHP Banana extends the *Banana* class provided by the SRM SAPI module.
- The source file declaring the PHP Banana class is stored in a *special folder* where the SRM daemon can access it.
- In that source file, the PHP Banana class needs to be instantiated and the *event loop* is to be started.

PHP Bananas

```php
<?php
class SampleBanana extends Banana {
    public function __construct() {
        $this->doExpensiveInitialization();
    }

    private function doExpensiveInitialization() {
        // ...
    }

    public function doSomething() {
        // ...
    }
}

$banana = new SampleBanana;
$banana->run();
?>
```

```php
<?php
$srm = new SRM('/tmp/srm.socket');
$app = new SRMApp($srm, 'SampleBanana');

$app->doSomething();
?>
```

Using Application Level Variables

```php
<?php
$srm = new SRM('/tmp/srm.socket');

$srm->globals['foo'] = 'bar';
?>
```

The End