Design patterns… revisited for PHP 5.3

Fabien Potencier

Serial entrepreneur

Developer by passion

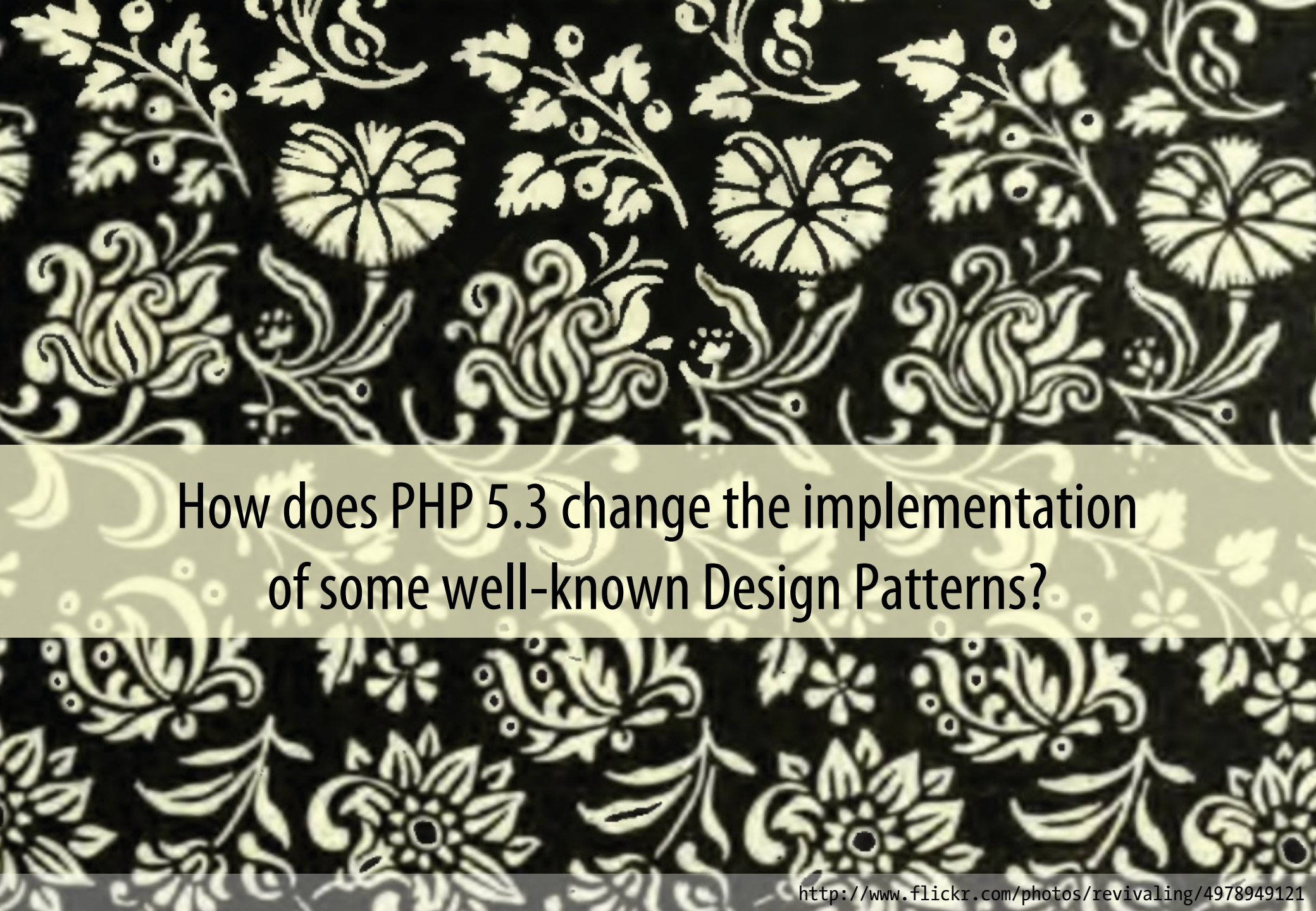
Founder of Sensio

Creator and lead developer of Symfony

http://www.twitter.com/fabpot

http://www.github.com/fabpot

http://fabien.potencier.org/



http://www.flickr.com/photos/revivaling/4978949121

« A design pattern
is a general reusable solution
to a commonly occurring problem »

http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29

« By definition, a pattern
must be programmed anew
into each application that uses it. »

http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29

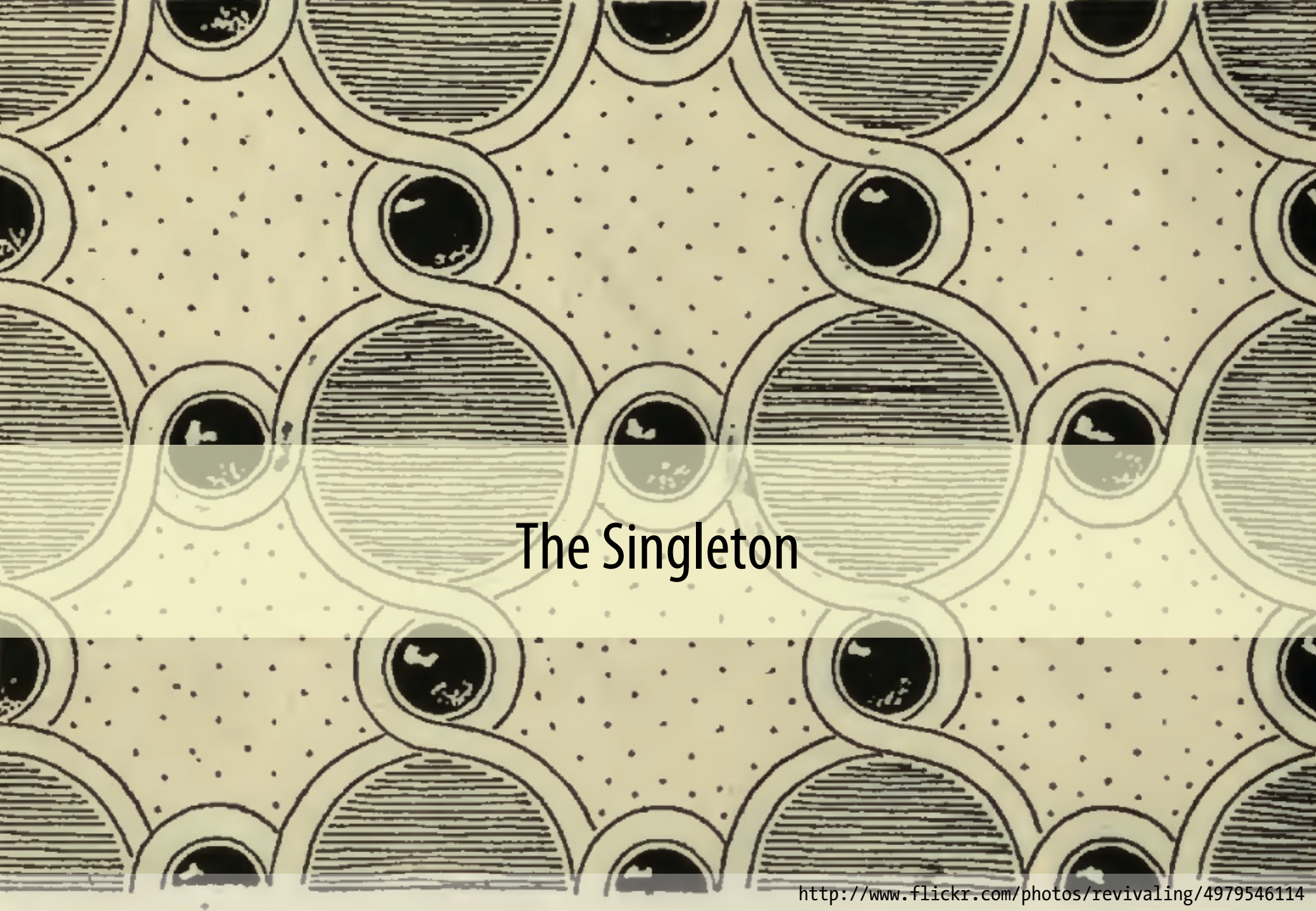
The Addison-Wesley Signature Series

A Martin Fowler Signature Book

Patterns of Enterprise Application Architecture

Martin Fowler

With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford



The Singleton

http://www.flickr.com/photos/revivaling/4979546114

The Singleton may cause serious damage to your code!

http://www.flickr.com/photos/mattbell/1696868871

The Singleton and PHP 4

```
class Singleton
{
    function &getInstance()
    {
        static $instance;

        if (!$instance) {
            $instance = new Singleton();
        }

        return $instance;
    }
}

$obj =& Singleton::getInstance();
```

You can still instantiate the class directly

The Singleton and PHP 5.0/5.1/5.2

```
class Singleton
{
    static private $instance;

    private function __construct() {}

    static public function getInstance()
    {
        if (null === self::$instance) {
            self::$instance = new self();
        }

        return self::$instance;
    }

    final private function __clone() {}
}

$obj = Singleton::getInstance();
```

do not forget to override __clone()

The Singleton and PHP 5.3

```php
abstract class Singleton
{
    private static $instances = array();

    final private function __construct()
    {
        if (isset(self::$instances[get_called_class()])) {
            throw new Exception("A ".get_called_class()." instance already exists.");
        }
        static::initialize();
    }

    protected function initialize() {}

    final public static function getInstance()
    {
        $class = get_called_class();
        if (!isset(self::$instances[$class])) {
            self::$instances[$class] = new static();
        }
        return self::$instances[$class];
    }

    final private function __clone() {}
}
```

The Singleton and PHP 5.3

```
class Foo extends Singleton {}
class Bar extends Singleton {}

$a = Foo::getInstance();
$b = Bar::getInstance();
```

Active Record
. . . Late Static Binding

http://www.flickr.com/photos/revivaling/4979548952

```php
$article = Article::findByPk(1);

$article = Article::findByTitleAndAuthorId('foo', 1);
```

```php
class Model
{
    static public function getMe()
    {
        return __CLASS__;
    }
}

class Article extends Model {}

echo Article::getMe();
```

```php
class Model
{
    static public function getMe()
    {
        return get_called_class();
    }
}

class Article extends Model {}

echo Article::getMe();
```

```php
class Model
{
    static public function findByPk($id)
    {
        $table = strtolower(get_called_class());
        $sql = "SELECT * FROM $this->table WHERE id = ?";

        return $this->db->get($sql, $id);
    }
}

class Article extends Model {}

$article = Article::findByPk(1);
```

```php
class Model
{
    static public function __callStatic($method, $arguments)
    {
        $table = strtolower(get_called_class());
        $column = strtolower(substr($method, 6));
        $value = $arguments[0];

        $sql = "SELECT * FROM $this->table WHERE $column = ?";

        return $this->db->get($sql, $value);
    }
}

class Article extends Model {}

$article = Article::findByTitle('foo');
```


This is pseudo code with no proper SQL Injection protection!
Caution
http://www.flickr.com/photos/zippy/22748510

The Observer Pattern

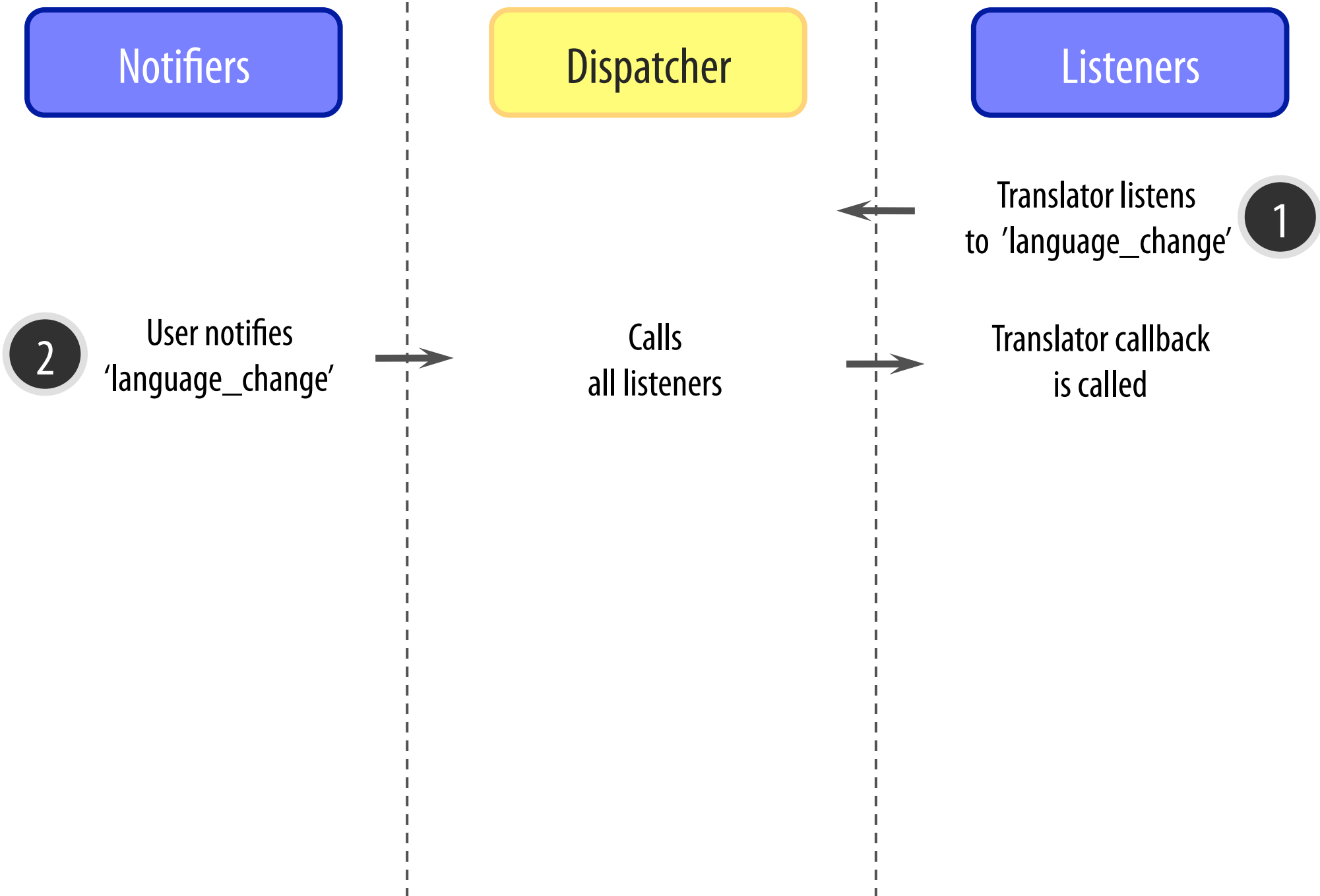
http://www.flickr.com/photos/revivaling/4978951341

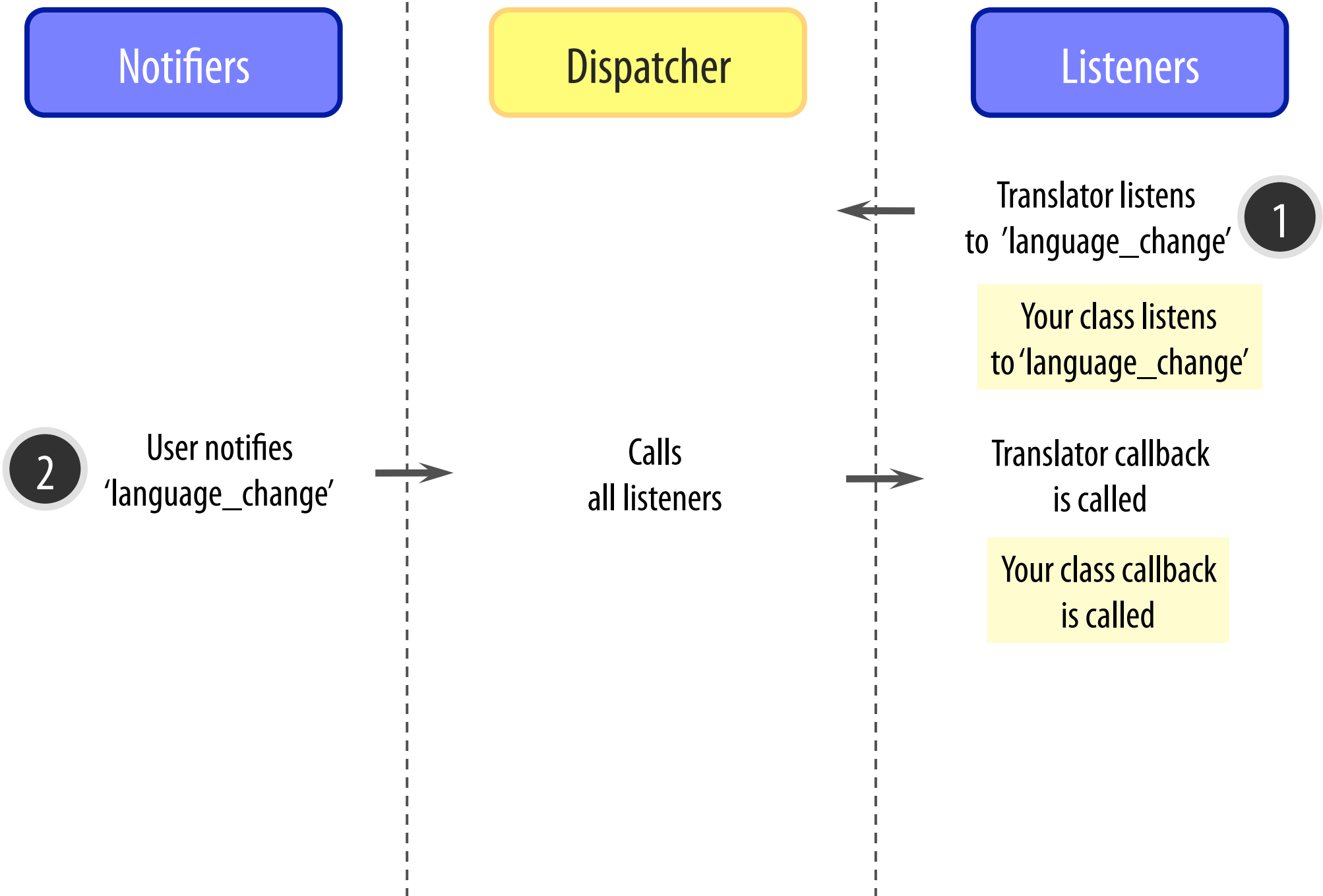
The Observer pattern allows
to extend or change the behavior of classes
without having to change them

When a method is called
on an object (subject),
it notifies other objects (listeners)
of the event

A Dispatcher is an object
that manages connections
between subjects and listeners

```php
class User
{
    function setLanguage($language)
    {
        $this->language = $language;
    }
}

class Translator
{
    function setLanguage($language)
    {
        $this->language = $language;
    }
}
```


Notifiers
Dispatcher
Listeners
Translator listens
to 'language_change'
1
2
User notifies
'language_change'
Calls
all listeners
Translator callback
is called


Notifiers
Dispatcher
Listeners
Translator listens
to 'language_change'
1
Your class listens
to 'language_change'
2
User notifies
'language_change'
Calls
all listeners
Translator callback
is called
Your class callback
is called

```php
$dispatcher = new Dispatcher();

$translator = new Translator($dispatcher);

$user = new User($dispatcher);

$user->setLanguage('fr');
```

```php
class Translator
{
    public function __construct
    {
        $listener = array($translator, 'listenLanguageChange');

        $dispatcher->connect('language_change', $listener);
    }

    public function listenLanguageChangeEvent($arguments)
    {
        $this->setLanguage($arguments['language']);
    }
}
```

A PHP callable

The Event name

```php
class User
{
    function setLanguage($language)
    {
        $this->language = $language;

        $this->dispatcher->notify('language_change',
           array('language' => $language));
    }
}
```

```php
// register the listener with the dispatcher
$dispatcher = new Dispatcher(array(
    'foo' => $listener,
));

// notify the event anywhere
$dispatcher->notify(
    'foo',
    array('name' => 'Fabien')
);
```

A listener can be any PHP callable

```
class Dispatcher
{
    function __construct($map)
    {
        $this->map = $map;
    }

    function notify($name, $args = array())
    {
        call_user_func($this->map[$name], $args);
    }
}
```

A PHP callable

```
class Dispatcher
{
    function __construct($map)
    {
        $this->map = $map;
    }

    function notify($name, $args = array())
    {
        $this->map[$name]($rags);
    }
}
```

An anonymous function

```php
// a function
function foo($args)
{
    echo "Hello {$args['name']}\n";
};

$listener = 'foo';
```

```php
// an anonymous function
$listener = function ($args)
{
    echo "Hello {$args['name']}\n";
};
```

```php
// register the listener with the dispatcher
$dispatcher = new Dispatcher(array(
    'foo' => function ($args)
    {
        echo "Hello {$args['name']}\n";
    },
));

// notify the event anywhere
$dispatcher->notify(
    'foo',
    array('name' => 'Fabien')
);
```

```php
$dispatcher = new Dispatcher(array(
    'foo' => $listener
));
```

```php
class Dispatcher
{
    function connect($name, $listener)
    {
        if (!isset($this->map[$name])) {
            $this->map[$name] = array();
        }

        $this->map[$name][] = $listener;
    }
}
```

```php
$l1 = function ($args)
      {
          echo "Hello1 {$args['name']}?\n";
      };


$l2 = function ($args)
      {
          echo "Hello2 {$args['name']}?\n";
      };
```

```php
$listener = new Listeners($l1, $l2);

$dispatcher = new Dispatcher(array(
    'foo' => $listener
));

$dispatcher = new Dispatcher(array(
    'foo' => new Listeners($l1, $l2)
));
```

```php
$listener = new Listeners($l1, $l2);

$listeners($args);
```

```php
class Listeners
{
    function __construct()
    {
        $this->listeners = func_get_args();
    }

    function __invoke($args = array())
    {
        foreach ($this->listeners as $listener)  {
            $listener($args);
        }
    }
}
```

Makes the object callable

http://www.flickr.com/photos/revivaling/4979557230

Hello {{ name }}

```php
echo render('Hello {{ name }}!', array('name' => 'Fabien'));


echo render(
    file_get_contents(__DIR__.'/template.php'),
    array('name' => 'Fabien')
);
```

```php
function render($template, $arguments)
{
    $evaluator = function ($match) use ($arguments)
    {
        if (isset($arguments[$match[1]])) {
            return $arguments[$match[1]];
        }

        return $match[1];
    };

    return preg_replace_callback('/\{\{\s*(.+?)\s*\}\}/',
        $evaluator, $template);
}
```

http://www.flickr.com/photos/revivaling/4978942907

« Dependency Injection is where components
are given their dependencies
through their constructors, methods, or directly into fields »

http://www.picoinjector.org/injection.html

In most web applications, you need to manage the user preferences

- The user language
- Whether the user is authenticated or not
- ...

This can be done with a User object

- `setLanguage(), getLanguage()`
- `setAuthenticated(), isAuthenticated()`
- ...

The User information
need to be persisted
between HTTP requests

We use the PHP session for the Storage

```php
class SessionStorage
{
    function __construct($cookieName = 'PHP_SESS_ID')
    {
        session_name($cookieName);
        session_start();
    }

    function set($key, $value)
    {
        $_SESSION[$key] = $value;
    }

    // ...
}
```

```php
class User
{
    protected $storage;

    function __construct()
    {
        $this->storage = new SessionStorage();
    }

    function setLanguage($language)
    {
        $this->storage->set('language', $language);
    }

    // ...
}

$user = new User();
```

Very hard to customize

Very easy to use

```
class User
{
    protected $storage;

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

$storage = new SessionStorage();
$user = new User($storage);
```

Very easy to customize

Slightly more difficult to use

That’s Dependency Injection

Nothing more

Instead of harcoding
the Storage dependency
inside the User class constructor

Inject the Storage dependency
in the User object

A Dependency Injector

Describes objects
and their dependencies

Instantiates and configures
objects on-demand

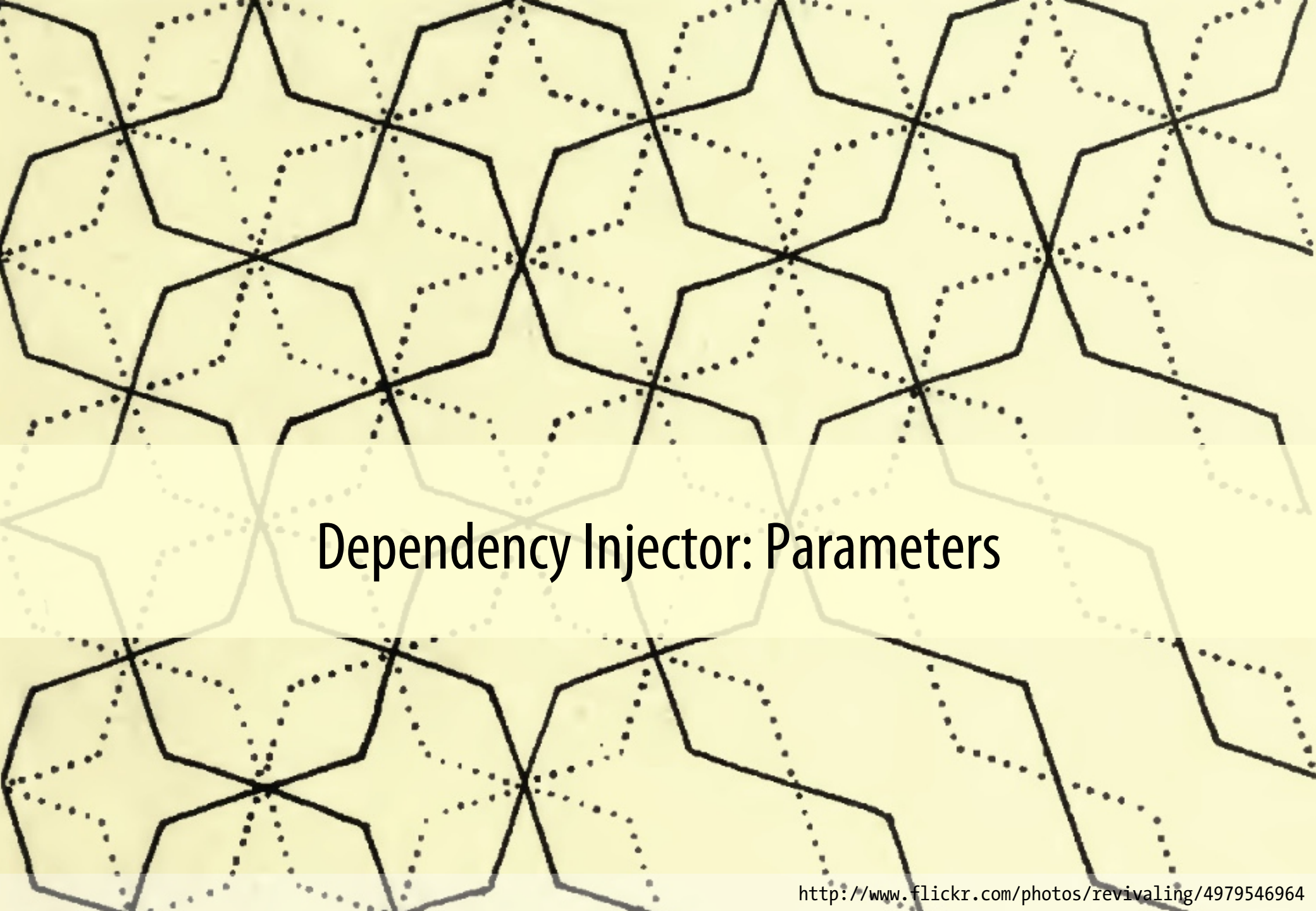
An injector
SHOULD be able to manage
ANY PHP object (POPO)

The objects MUST not know
that they are managed
by the injector

- Parameters
  - The SessionStorage implementation we want to use (the class name)
  - The session name
- Objects
  - SessionStorage
  - User
- Dependencies
  - User depends on a SessionStorage implementation

Let's build a simple injector with PHP 5.3

http://www.flickr.com/photos/22750018@N05/4531762220/



Dependency Injector: Parameters

http://www.flickr.com/photos/revivaling/4979546964

```php
class Injector
{
    protected $parameters = array();

    public function setParameter($key, $value)
    {
        $this->parameters[$key] = $value;
    }

    public function getParameter($key)
    {
        return $this->parameters[$key];
    }
}
```

Decoupling

```
$injector = new Injector();
$injector->setParameter('session_name', 'SESSION_ID');
$injector->setParameter('storage_class', 'SessionStorage');
```

Customization

```
$class = $injector->getParameter('storage_class');
$sessionStorage = new $class($injector->getParameter('session_name'));
$user = new User($sessionStorage);
```

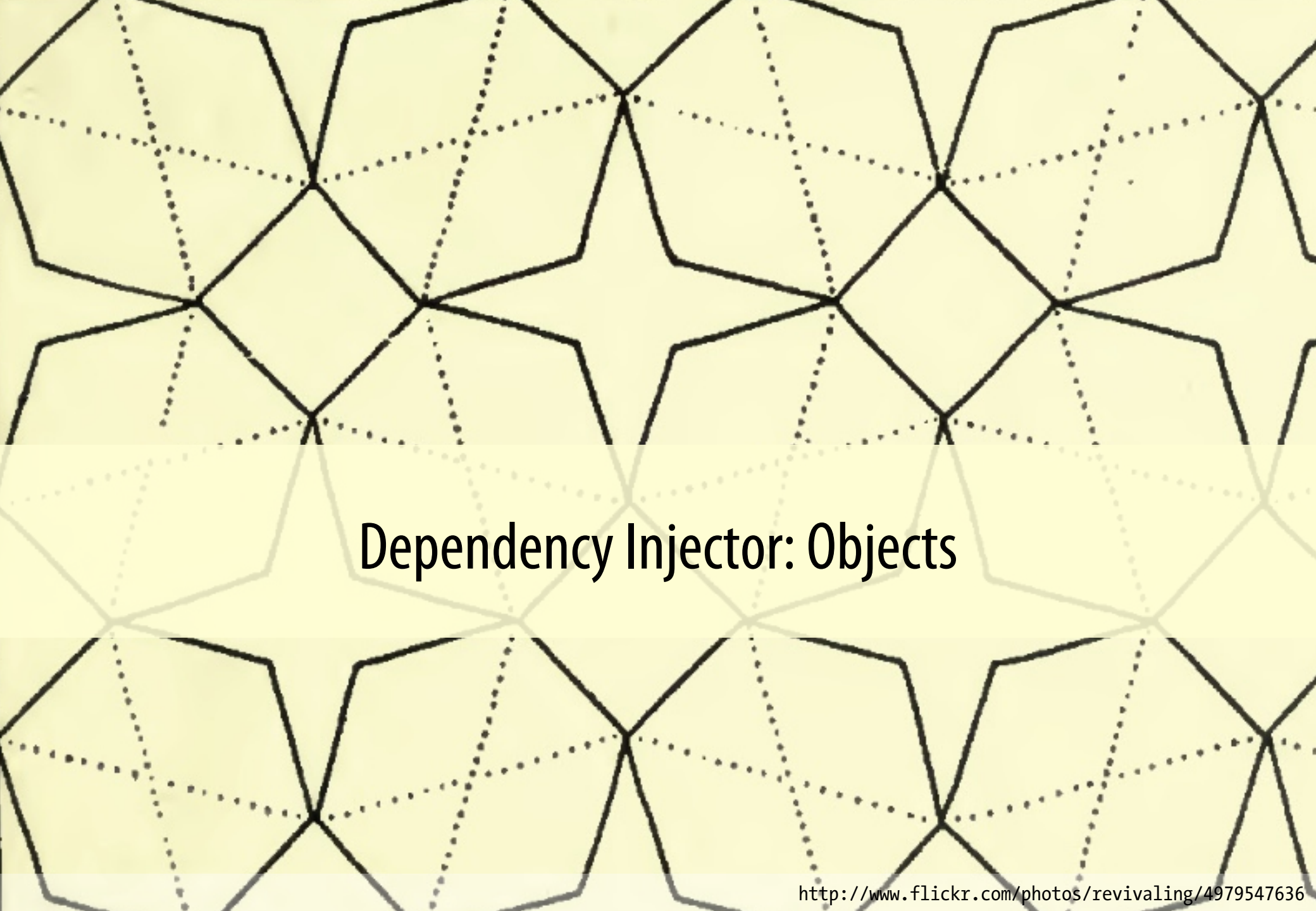
Objects creation

```php
class Injector
{
    protected $parameters = array();

    public function __set($key, $value)
    {
        $this->parameters[$key] = $value;
    }

    public function __get($key)
    {
        return $this->parameters[$key];
    }
}
```

Using PHP magic methods

Interface is cleaner

```
$injector = new Injector();
$injector->session_name = 'SESSION_ID';
$injector->storage_class = 'SessionStorage';

$sessionStorage = new $injector->storage_class($injector->session_name);
$user = new User($sessionStorage);
```



Dependency Injector: Objects

http://www.flickr.com/photos/revivaling/4979547636

We need a way to describe how to create objects, without actually instantiating anything!

Anonymous functions to the rescue!

```php
class Injector
{
    protected $parameters = array();
    protected $objects = array();

    public function __set($key, $value)
    {
        $this->parameters[$key] = $value;
    }

    public function __get($key)
    {
        return $this->parameters[$key];
    }

    public function setService($key, Closure $service)
    {
        $this->objects[$key] = $service;
    }

    public function getService($key)
    {
        return $this->objects[$key]($this);
    }
}
```

Store a lambda able to create the object on-demand

Ask the closure to create the object and pass the current injector

```php
$injector = new Injector();
$injector->session_name = 'SESSION_ID';
$injector->storage_class = 'SessionStorage';
$injector->setService('user', function ($c)
{
    return new User($c->getService('storage'));
});
$injector->setService('storage', function ($c)
{
    return new $c->storage_class($c->session_name);
});
```

Description

Creating the User
is now as easy as before

```php
$user = $injector->getService('user');
```

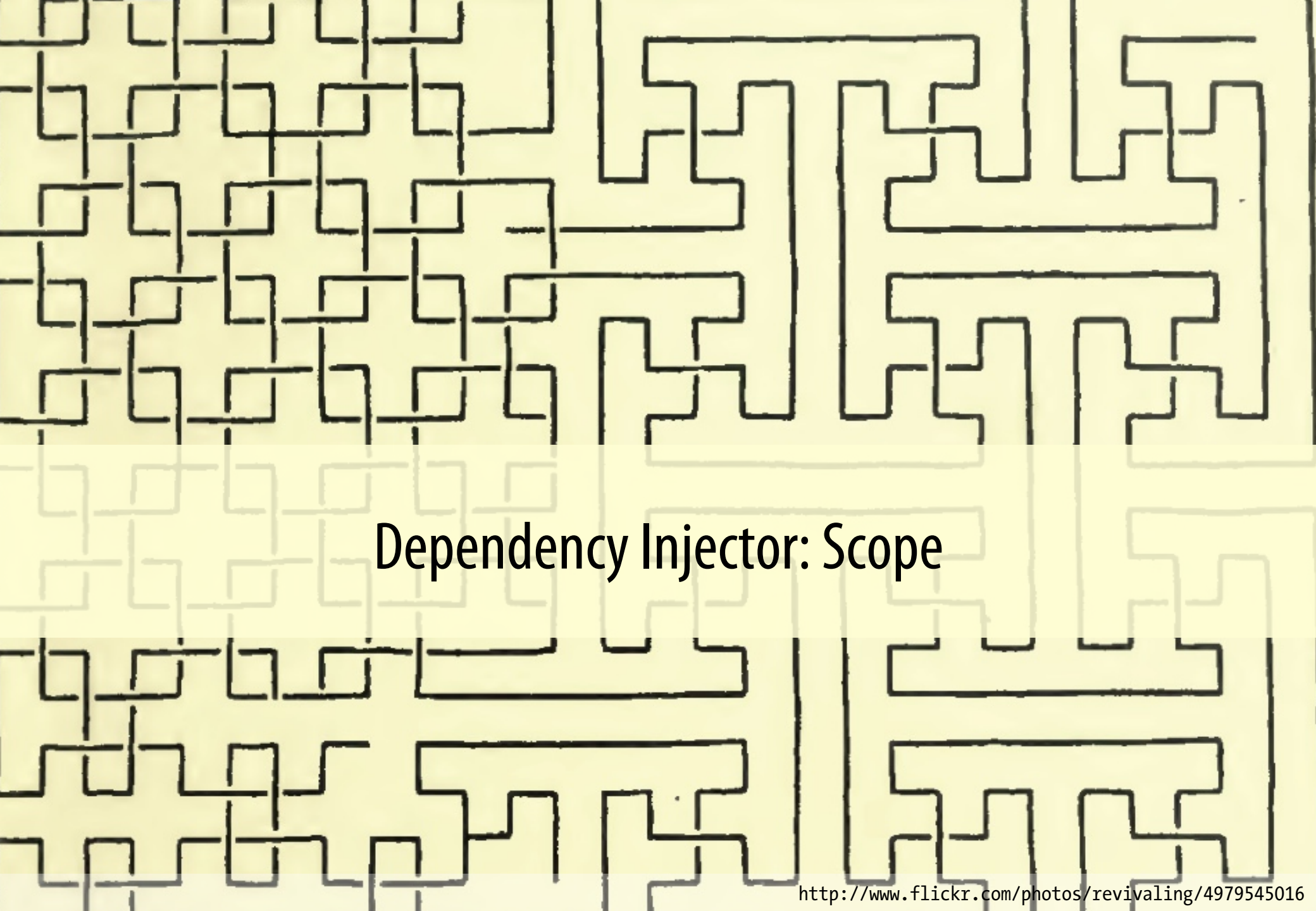
```php
class Injector
{
    protected $values = array();

    function __set($id, $value)
    {
        $this->values[$id] = $value;
    }

    function __get($id)
    {
        if (is_callable($this->values[$id])) {
            return $this->values[$id]($this);
        } else {
            return $this->values[$id];
        }
    }
}
```

Simplify the code

Unified interface

```
$injector = new Injector();
$injector->session_name = 'SESSION_ID';
$injector->storage_class = 'SessionStorage';
$injector->user = function ($c)
{
    return new User($c->storage);
};
$injector->storage = function ($c)
{
    return new $c->storage_class($c->session_name);
};
```

```
$user = $injector->user;
```



Dependency Injector: Scope

http://www.flickr.com/photos/revivaling/4979545016

For some objects, like the user,
the injector must always
return the same instance

```
spl_object_hash($injector->user)

!==

spl_object_hash($injector->user)
```

```php
$injector->user = function ($c)
{
    static $user;

    if (is_null($user)) {
        $user = new User($c->storage);
    }

    return $user;
};
```

```php
spl_object_hash($injector->user)

===

spl_object_hash($injector->user)
```

```php
$injector->user = $injector->asShared(function ($c)
{
    return new User($c->storage);
});
```

```php
function asShared(Closure $lambda)
{
    return function ($injector) use ($lambda)
    {
        static $object;

        if (is_null($object)) {
            $object = $lambda($injector);
        }

        return $object;
    };
}
```

```php
class Injector
{
    protected $values = array();

    function __set($id, $value)
    {
        $this->values[$id] = $value;
    }

    function __get($id)
    {
        if (!isset($this->values[$id])) {
            throw new InvalidArgumentException(sprintf('Value "%s" is not
defined.', $id));
        }

        if (is_callable($this->values[$id])) {
            return $this->values[$id]($this);
        } else {
          return $this->values[$id];
        }
    }
}
```

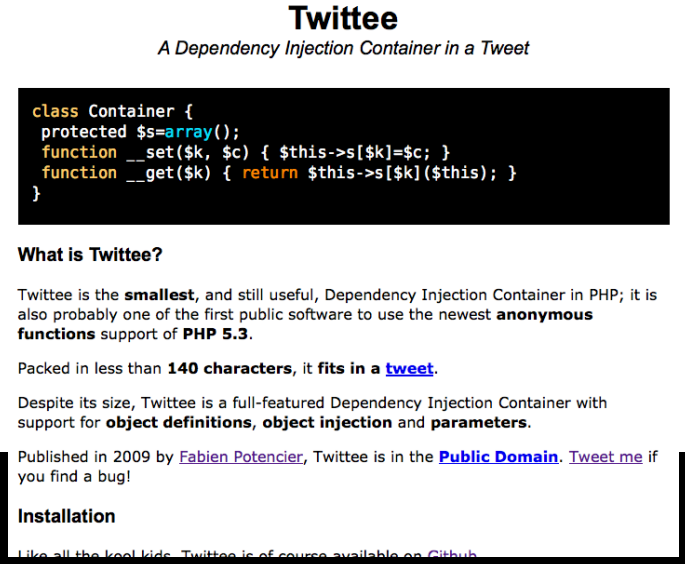
Error management

```php
class Injector
{
    protected $values = array();

    function __set($id, $value)
    {
      $this->values[$id] = $value;
    }

    function __get($id)
    {
        if (!isset($this->values[$id])) {
            throw new InvalidArgumentException(sprintf('Value "%s" is not defined.', $id));
        }

        if (is_callable($this->values[$id])) {
            return $this->values[$id]($this);
        } else {
            return $this->values[$id];
        }
    }

    function asShared($callable)
    {
        return function ($c) use ($callable)
        {
            static $object;

            if (is_null($object)) {
                $object = $callable($c);
            }
            return $object;
        };
    }
}
```

40 LOC for a fully-featured injector

Twittee: A Dependency Injection Container in a tweet

- Implementation does not use PHP 5.3
- Its usage needs PHP 5.3



Twittee

A Dependency Injection Container in a Tweet

```
class Container {
 protected $s=array();
 function __set($k, $c) { $this->s[$k]=$c; }
 function __get($k) { return $this->s[$k]($this); }
}
```

What is Twittee?

Twittee is the **smallest**, and still useful, Dependency Injection Container in PHP; it is also probably one of the first public software to use the newest **anonymous functions** support of **PHP 5.3**.

Packed in less than **140 characters**, it **fits in a tweet**.

Despite its size, Twittee is a full-featured Dependency Injection Container with support for **object definitions**, **object injection** and **parameters**.

Published in 2009 by Fabien Potencier, Twittee is in the **Public Domain**. Tweet me if you find a bug!

Installation

```
class Container {
 protected $s=array();
 function __set($k, $c) { $this->s[$k]=$c; }
 function __get($k) { return $this->s[$k]($this); }
}
```

twittee.org

Anonymous Functions
Closures
Late Static Binding
__invoke()
...

I'm NOT advocating
the usage of lambdas everywhere

This presentation was about
showing how they work
on practical examples

Questions?


symfony PRESENTS
SYMFONY
RELOADED
easy
CODE
What the hell is wrong with fr
That's what it's all about.
Pimp
My Co
Fabulous Feb's all
2FA
2CO
Don't code like sf1.
symfony PRESENTS
BACK
TO THE FRAMEWORK
POTENCIER
sf
2FAST
2CODE
Pimp
My Code
SYMFONY
RELOADED


Symfony Live 2011
PARIS
MARCH 2011
Join us
SAN FRANCISCO
FEBRUARY 2011
Join us
I ♥ Symfony
I need a geek
a SENSIOLABS event
symfony

Sensio S.A.
92-98, boulevard Victor Hugo
92 115 Clichy Cedex
FRANCE

Tél. : +33 1 40 99 80 80

Contact
Fabien Potencier
fabien.potencier at sensio.com

http://www.sensiolabs.com/

http://www.symfony-project.org/

http://fabien.potencier.org/

Interlude
. . . Anonymous Functions

http://www.flickr.com/photos/gregory_bastien/2565132371

An anonymous function
is a function
defined on the fly (no name)

```
function () { echo 'Hello world!'; };
```

Can be stored in a variable

```php
$hello = function () { echo 'Hello world!'; };
```

. . . to be used later on

```
$hello();
```

```
call_user_func($hello);
```

. . . or can be passed as a function argument

```
function foo(Closure $func)
{
  $func();
}

foo($hello);
```

Can take arguments

```php
$hello = function ($name) { echo 'Hello '.$name; };

$hello('Fabien');

call_user_func($hello, 'Fabien');
```

```php
function foo(Closure $func, $name)
{
  $func($name);
}

foo($hello, 'Fabien');
```

When is it useful?

array_*

Greatly simplify usage of some array_* functions

- `array_map()`
- `array_reduce()`
- `array_filter()`

```php
class Article
{
  public function __construct($title)
  {
    $this->title = $title;
  }

  public function getTitle()
  {
    return $this->title;
  }
}
```

How to get an array of all article titles?

```php
$articles = array(
  new Article('PHP UK - part 1'),
  new Article('PHP UK - part 2'),
  new Article('See you next year!'),
);
```

```php
$titles = array();
foreach ($articles as $article)
{
  $titles[] = $article->getTitle();
}
```

```php
$titles = array_map(
  create_function('$article', 'return $article->getTitle();'),
  $articles
);
```

```php
$titles = array_map(
  function ($article) { return $article->getTitle(); },
  $articles
);
```

```php
$titles = array();
foreach ($articles as $article)
{
  $titles[] = $article->getTitle();
}
```

100 100

```php
$titles = array_map(create_function('$article', 'return $article->getTitle();'), $articles);
```

300 1800

```php
$titles = array_map(function ($article) { return $article->getTitle(); }, $articles);
```

100 200

```php
$mapper = function ($article) { return $article->getTitle(); };
$titles = array_map($mapper, $articles);
```

100 180

memory speed

```php
$mapper = function ($article) {
  return $article->getTitle();
};

$titles = array_map($mapper, $articles);

$mapper = function ($article) {
  return $article->getAuthor();
};

$authors = array_map($mapper, $articles);
```

A closure is a lambda
that remembers the context
of its creation. . .

```php
$mapper = function ($method)
{
    return function ($article) use($method)
    {
        return $article->$method();
    };
};
```

```php
$method = 'getTitle';

$mapper = function ($article) use($method)
{
  return $article->$method();
};

$method = 'getAuthor';

$titles = array_map($mapper, $articles);
```

```php
$titleMapper = $mapper('getTitle');
$titles = array_map($titleMapper, $articles);


$authorMapper = $mapper('getAuthor');
$authors = array_map($authorMapper, $articles);
```

```php
$titles = array_map($mapper('getTitle'), $articles);

$authors = array_map($mapper('getAuthor'), $articles);
```