*Introducing*

# Symfony

*The most revolutionary thing in France since 1789*

*Presented by Claudio Beatrice*

# Allow me to introduce myself

**Claudio Beatrice**
**@omissis**

6+ years of
experience on PHP

Organizing Drupal
events since 2009

drupal italia
ASSOCIAZIONE

PHP, Drupal &
Symfony consulting

agavee
DEVELOPMENT

GOOM
RADIO

Web Radio

Skylogic
a eutelsat company

Telecommunications

# What's Symfony2

A reusable set of standalone, decoupled,
and cohesive PHP 5.3 components

A full-stack web framework 

A Request/Response framework built
around the HTTP specification

A promoter of best practices,
standardization and interoperability 

An awesome community!

# Leave The STUPID Alone...

As time went by, habits and practices that once seemed acceptable have proven that they were making our code harder to understand and maintain. In other words, STUPID.

But what makes code such a thing?

- **S**ingleton
- **T**ight coupling
- **U**ntestability
- **P**remature optimization
- **I**ndescriptive naming
- **D**uplication

# ...BE SOLID!

What alternatives to write STUPID code do we have?

Another acronym to the rescue: SOLID!

It encloses five **class design principles**:

- **S**ingle responsibility principle
- **O**pen/closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

# Single Responsibility

*There should never be more than one reason for a class to change.*

Every class should have a single responsibility and fully encapsulate it.

If change becomes localized, complexity and cost of change are reduced, moreover there's less risk of ripple effects.

# Single Responsibility

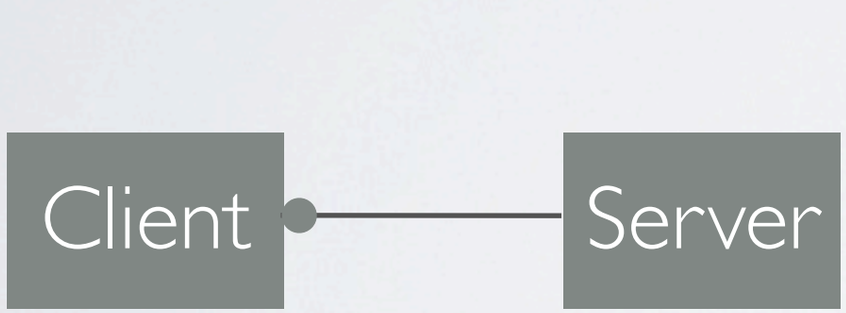
```
interface Modem
{
    function dial($phoneNumber);
    function hangup();
    function send($message);
    function receive();
}
```

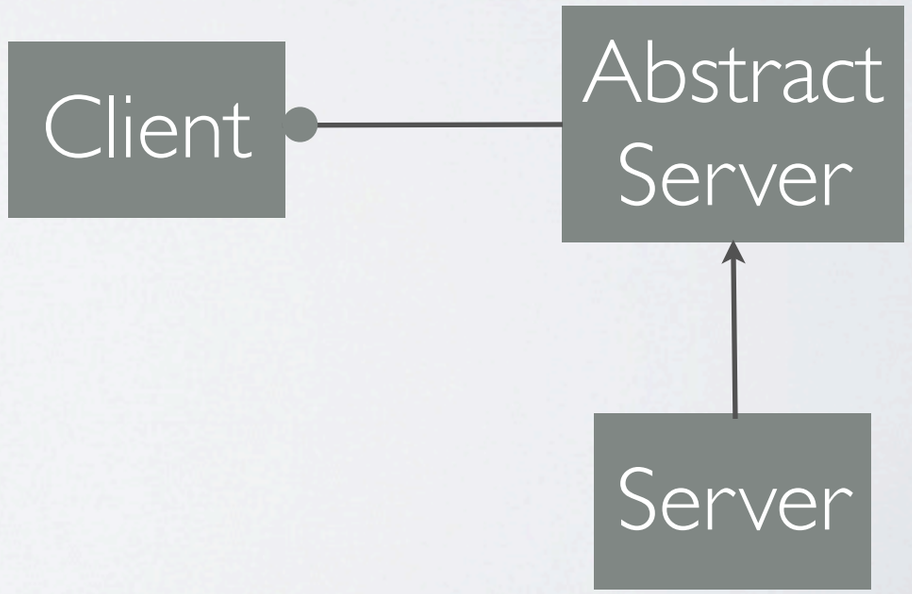
The above interface shows two responsibilities: connection management and data communication, making them good candidates for two separate interfaces/implementations.

# Open/Closed

*Software entities (classes, functions, etc) should be open for extension, but closed for modification.*

This principle states that the source code of software entities shouldn't ever be changed: those entities must be derived in order to add the wanted behaviors.



# Liskov Substitution

Objects in a program should be replaceable with instances of their subtypes without altering any of the desirable properties of that program, such as correctness and performed task.

It intends to guarantee semantic interoperability of object types in a hierarchy.

# Liskov Substitution

```php
class Rectangle {
  protected $width;
  protected $height;

  function setWidth($width) {...}
  function getWidth() {...}
  function setHeight($height) {...}
  function getHeight() {...}
}

class Square extends Rectangle {
  function setWidth($width) {
    $this->width = $width;
    $this->height = $width;
  }
  function setHeight($height) {
    $this->width = $height;
    $this->height = $height;
  }
}
```

```php
function draw(Rectangle $r) {
  $r->setWidth(5);
  $r->setHeight(4);

  // is it correct to assume that
changing the width of a Rectangle
leaves is height unchanged?
  assertEquals(
    20,
    $r->setWidth() * $r->setHeight()
  );
}
```

# Liskov Substitution

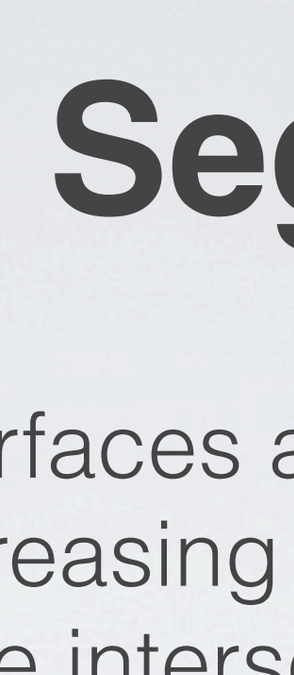
The flaw in the Rectangle-Square design shows that even if conceptually a square is a rectangle, a *Square* object is not a *Rectangle* object, since **a *Square* does not behave as a *Rectangle***.

As a result, the public behavior the clients expect for the base class must be preserved in order to conform to the LSP.

# Interface Segregation

Many client-specific interfaces are better than one big one. This principle helps decreasing the coupling between objects by minimizing the intersecting surface.

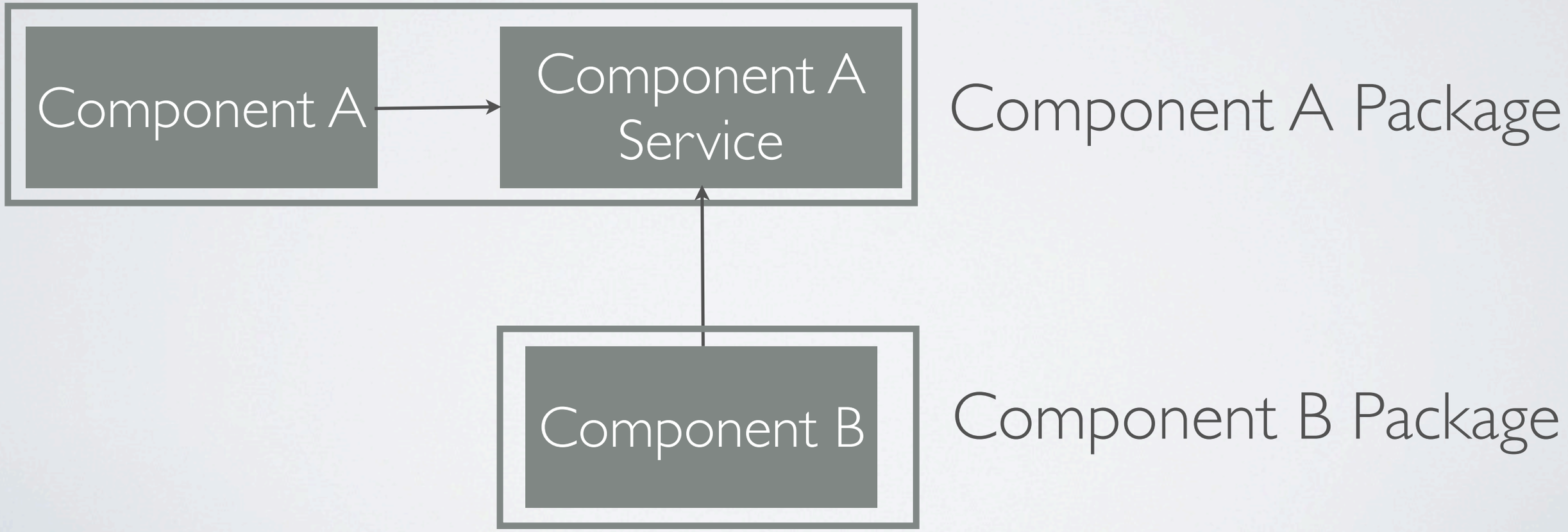
```
interface MultiFunctionPrinter
{
    function print(...);
    function scan(...);
    function fax(...);
}
```



```
interface Printer
{
    function print(...);
}

interface Scanner
{
    function scan(...);
}

interface Fax
{
    function fax(...);
}
```

# Dependency Inversion

- High-level entities should not depend on low-level entities and both should depend on abstractions.
- Abstractions should not depend upon details: details should depend upon abstractions.



# Dependency Inversion

```
class Vehicle {
    protected $tyres;

    public function __construct() {
        $this->tyres = array_fill(0, 4, new Tyre(50));
    }
}
class Tyre {
    private $diameter;

    public function __construct($diameter) {
        $this->setDiameter($diameter);
    }

    public function setDiameter($diameter) {
        $this->diameter = $diameter;
    }

    public function getDiameter() {
        return $this->diameter;
    }
}
```

# Dependency Inversion

```php
namespace Vehicle;

class Vehicle {
    protected $tyres;

    function addTyre(AbstractTyre $tyre) {
        $this->tyres[] = $tyre;
    }
}
```

```php
namespace Vehicle;

abstract class AbstractTyre {
    private $diameter;

    function __construct($diameter) {...}

    function setDiameter($diameter) {...}

    function getDiameter() {...}
}
```

---

```php
namespace Tyre;

use Vehicle\AbstractTyre;

class RaceTyre extends AbstractTyre {
    private $compound;

    function setCompound($compound) {...}

    function getCompound() {...}
}
```

# How About Symfony Now?

Being aware of the principles of software development mentioned earlier allow us to better understand some of the choices that have been made for the framework as well as some of the tools that have been made available, such as:

- Class Loader
- Service Container
- Event Dispatcher
- HTTP Foundation
- HTTP Kernel
- Twig

# Class Loader

It loads your project's classes automatically if they're following a standard PHP convention aka **PSR-0**.

- **Doctrine\Common\IsolatedClassLoader**
  => /path/to/project/lib/vendor/**Doctrine/Common/IsolatedClassLoader.php**
- **Symfony\Core\Request**
  => /path/to/project/lib/vendor/**Symfony/Core/Request.php**
- **Twig_Node_Expression_Array**
  => /path/to/project/lib/vendor/**Twig/Node/Expression/Array.php**

It's a great way to get out of the require_once hell while gaining better interoperability and lazy loading at the same time.

# Service Container

*aka Dependency Injection Container*

A **Service** is any PHP object that performs a “global” task: think of a Mailer class.

A **Service Container** is a special object (think of it as an **Array of Objects on Steroids**) that centralizes and standardizes the way objects are constructed inside an application: instead of directly creating Services, the developer configures the Container to take care of the task.

# Service Container

```php
// Instantiate the container builder
$container = new ContainerBuilder();

// Create a config.storage.options parameter
$container->setParameter('config.storage.options', array(
  'connection' => 'default',
  'target' => 'default',
));

// Register a config.storage service, passing the options
to its constructor
// and invoking a mutator to set a third parameter
$container->register('config.storage', 'Drupal\Core\Config
\DatabaseStorage')
  ->addArgument('%config.storage.options%')
  ->addMethodCall('setPersistent', (boolean)
$isPersistent);

// Retrieve the storage service
$storage = $container->get('config.storage');
```

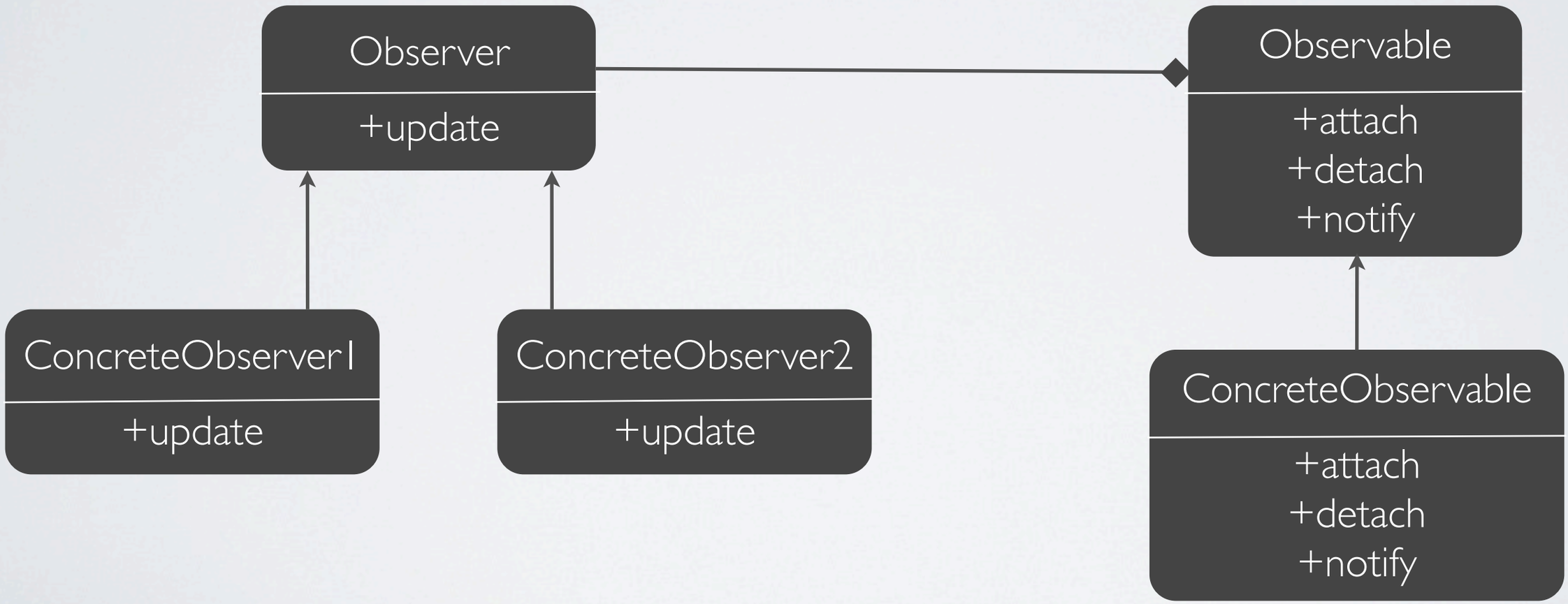
# Service Container

```
function drupal_container(Container $reset = NULL) {
  static $container = NULL;
  if (isset($reset)) {
    $container = $reset;
  }
  elseif (!isset($container)) {
    // build container
  }
  return $container;
}
```

Whilst it's possible to use the container to fetch the services, it's best not to do so. Injecting only the needed dependencies will free our code from depending on the specific instance of the container we're using.

# Event Dispatcher

A lightweight implementation of the **Observer Pattern**,
it provides a powerful and easy way to extend objects.



# Event Dispatcher

```php
use Symfony\Component\EventDispatcher\EventDispatcher;

$dispatcher = new EventDispatcher();

$callable = function (Event $event) use ($log) {
    $log->addWarning('th3 n1nj4 d1sp4tch3r 1s 4ft3r y0u');
}
$dispatcher->addListener('foo.bar', $callable);

$dispatcher->dispatch('foo.bar', new Event());
```

# HTTP Foundation

It replaces the PHP's global variables and functions that represent either requests or responses with a full-featured object-oriented layer for the HTTP messages.

```php
use Symfony\Component\HttpFoundation\Request;
// http://example.com/?foo=bar
$request = Request::createFromGlobals();
$request->query->get('foo'); // returns bar
// simulate a request
$request = Request::create('/foo', 'GET', array('name' => 'Bar'));


use Symfony\Component\HttpFoundation\Response;

$response = new Response('Content', 200, array(
    'content-type' => 'text/html'
));
// check the response is HTTP compliant and send it
$response->prepare($request);
$response->send();
```

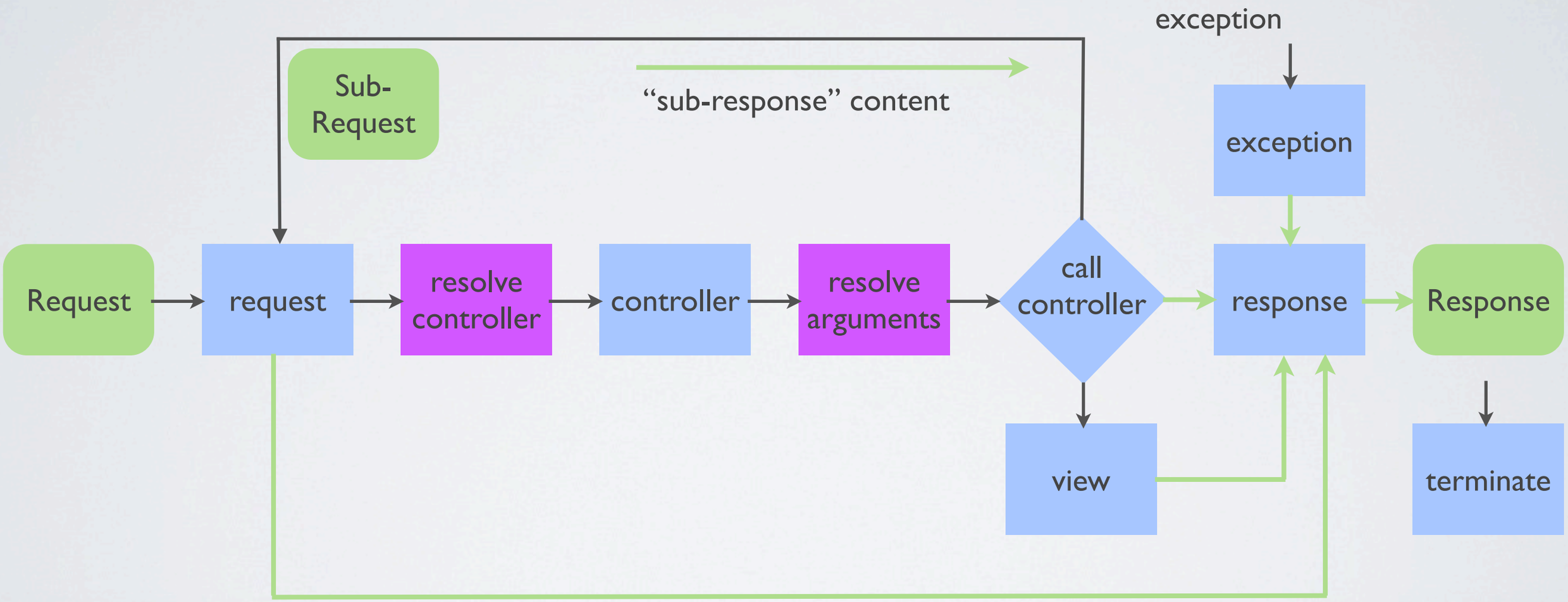
# HTTP Kernel

The Kernel is the core of Symfony2: it is built on top of the HttpFoundation and its main goal is to “convert” a Request object into a Response object using a Controller, which in turn can be any kind of PHP callable.

```
interface HttpKernelInterface
{
    const MASTER_REQUEST = 1;
    const SUB_REQUEST = 2;

    /**
     * ...
     * @return Response A Response instance
     * ...
     * @api
     */
    function handle(Request $request, $type = self::MASTER_REQUEST,
$catch = true);
}
```

# HTTP Kernel

## Workflow



# Twig

A flexible, fast and secure template engine for PHP.

It offers a great set of features, a concise syntax and good performances: it compiles to PHP and has an optional C extension; moreover it's super easy to extend and it's thoughtfully documented.

It gives the presentation layer a huge boost in terms of expressiveness, making development easier, faster and more structured.

# Twig

All this power is enclosed by just three types of tags...

- **comment tag**, they surprisingly *do nothing*:

```
{# foo comment #}
```

- **block tag**, they *do something*, like setting a variable, or invoking a control statement:

```
{% set bar = 'baz' %}
```

- **print tag**, they *say something*, like:

```
{{ "bar contains: " ~ bar }}
```

# ...and a ton of built-in goodness!

## Tags

autoescape
block
do
embed
extends
filter
flush
for
from
if
import
include
macro
raw
sandbox
set
spaceless
use

## Filters

abs
capitalize
convert_encoding
date
default
escape
format
join
json_encode
keys
length
lower
merge
nl2br
number_format
raw
replace
reverse
slice
sort
striptags
title
trim
upper
url_encode

## Functions

attribute
block
constant
cycle
date
dump
parent
random
range

## Tests

constant
defined
divisibleby
empty
even
iterable
null
odd
sameas

## Operators

in
is
Math (+, -, /, %, /, *, **)
Logic (and, or, not, (), b-and, b-xor, b-or)
Comparisons (==, !=, <, >, >=, <=, ===)
Others (.., |, ~, ., [], ?:)

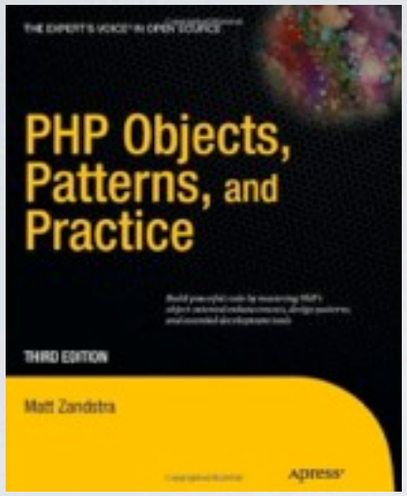
# Twig

```
{# Node list page #}
{% extends 'layout.html.twig' %}

{% macro node_render(node) %}
    <div id="node-{{ node.id }}">
        <h2>{{ node.title|title }}</h2>
        <div>{{ node.creationDate|date('d/m/Y') }}</div>
        <div>{{ node.body }}</div>
        <div>{{ node.tags|join(', ')|lower }}</div>
        <a href="{{ slug(node.path) }}">Permalink</a>
    </div>
{% endmacro %}

{% block body %}
    {% for node in nodes %}
        node_render(node);
    {% else %}
        {{ 'We did not find any node.'|trans }}
    {% endfor %}
{% endblock body %}
```

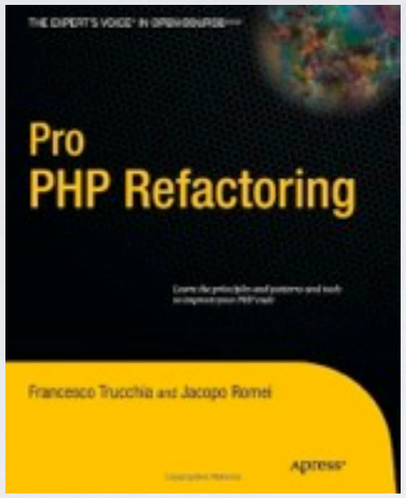
# Twig

```
class UrlExtension extends \Twig_Extension
{
    public function getFunctions()
    {
        return array(
            'slug' => new \Twig_Function_Method($this, 'slug'),
        );
    }

    public function slug($str) {
        return strtolower(preg_replace(array('/[^a-zA-Z0-9 -]/', '/[ -]
+/', '/^-|-$/'),
        array('', '-', ''), transliterate($str)));
    }

    public function transliterate($str) {
        // convert $str to ASCII characters
    }
}
```

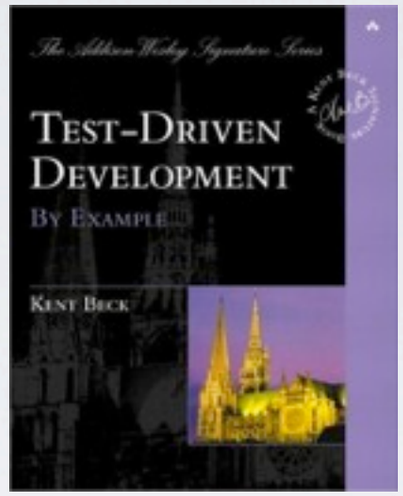
# Giving The Devil His Due



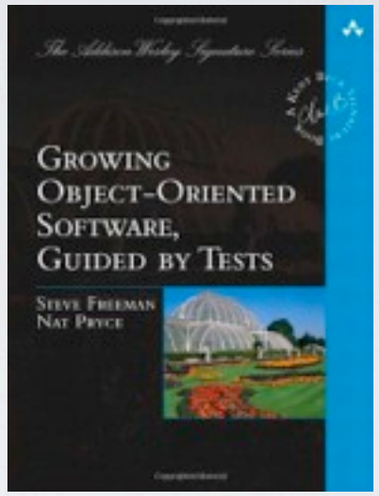

PHP Objects, Patterns and Practice



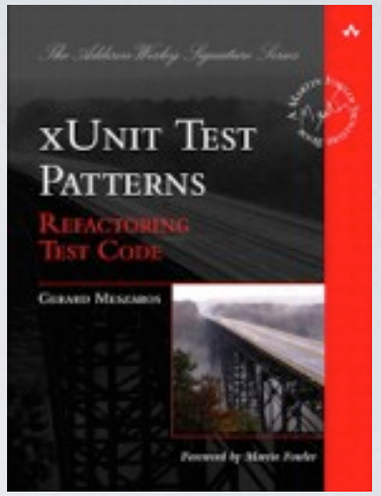

Pro PHP Refactoring




Test-Driven Development By Example




Growing Object-Oriented Software, Guided by Tests




xUnit Test Patterns

http://www.slideshare.net/weaverryan/being-dangerous-with-twig-symfony-live-paris
http://www.slideshare.net/weaverryan/symony2-a-next-generation-php-framework
http://www.slideshare.net/weaverryan/handson-with-the-symfony2-framework
http://www.butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
http://www.slideshare.net/jwage/symfony2-from-the-trenches
http://martinfowler.com/articles/injection.html
http://fabien.potencier.org/
http://nikic.github.com/
http://symfony.com/

# Thank You!

Please take a second to rate this session here:

http://munich2012.drupal.org/program/sessions/introducing-symfony2

**Claudio Beatrice**
twitter: @omissis
blog: silent-voice.org