Symfony 2

Fabien Potencier

Who am I?

- Founder of Sensio
  - Web Agency
  - Since 1998
  - 70 people
  - Open-Source Specialists
  - Big corporate customers

- Creator and lead developer of symfony

How many of you have used symfony?

1.0? 1.1? 1.2?

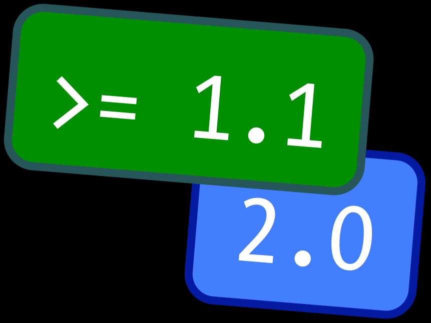
symfony 1.0

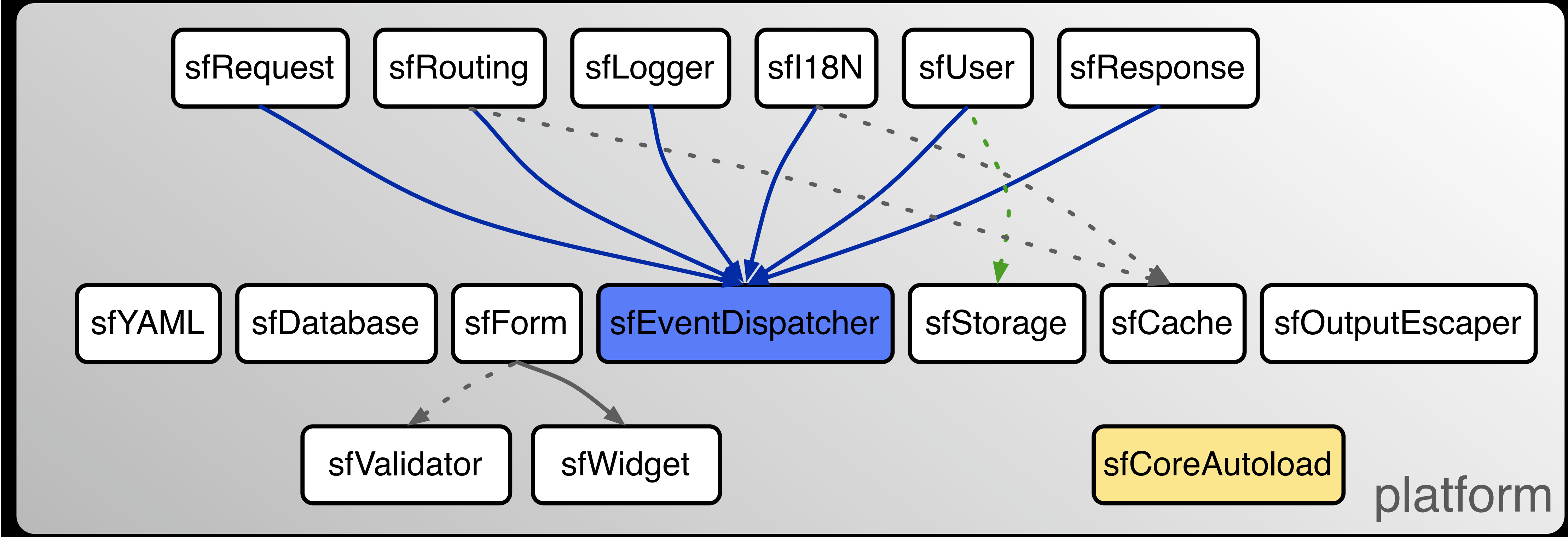
- Started as a glue between existing Open-Source libraries:
  - – Mojavi (heavily modified), Propel, Prado i18n, . . .
- Borrowed concepts from other languages and frameworks:
  - – Routing, CLI, functional tests, YAML, Rails helpers. . .
- Added new concepts to the mix
  - – Web Debug Toolbar, admin generator, configuration cascade, . . .

symfony 1.2

- Decoupled but cohesive components: the symfony platform
  - Forms, Routing, Cache, YAML, ORMs, . . .
- Controller still based on Mojavi
  - View, Filter Chain, . . .
- Could have been named 2.0 ;)

symfony platform





symfony platform

```
require_once '/path/to/sfCoreAutoload.class.php';
sfCoreAutoload::register();
```

```php
$config = sfYaml::load(<<<EOF
config:
  key: value
  foo: [bar, foobar]
  bar: { bar: foo }
EOF
);

print_r($config);
echo sfYaml::dump($config);
```

```php
$cache = new sfSQLiteCache(array(
  'database' => dirname(__FILE__).'/cache.db'
));
$cache->set('foo', 'bar');
echo $cache->get('foo');
```

Symfony 2 is an evolution of symfony 1

- Same symfony platform
- Different controller implementation
- Oh! Symfony now takes a capital S!!!

Symfony 2 main goals

Flexibility

Fast

Smart

Symfony 2: New components

Dependency Injection Container

Templating Framework

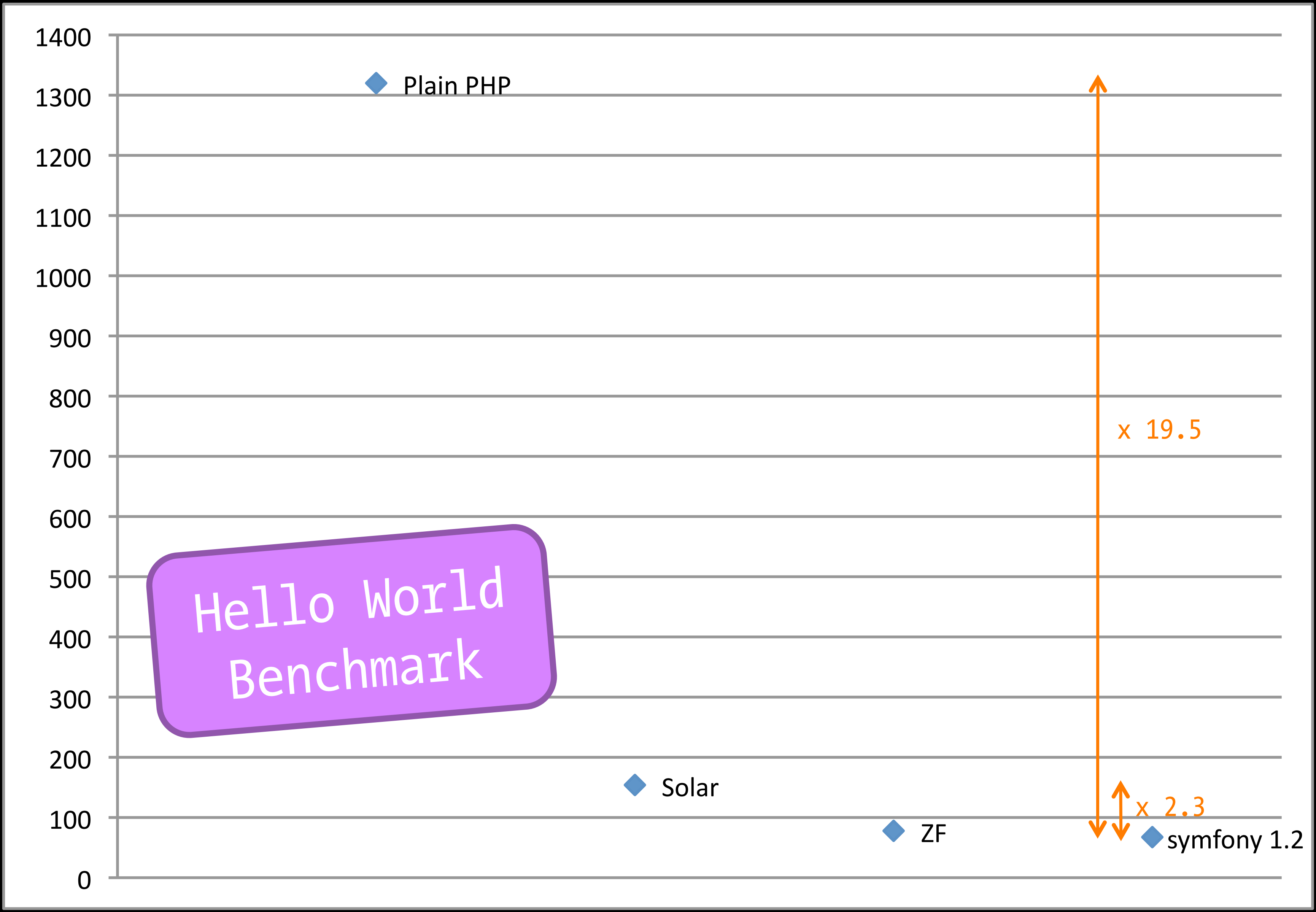
Controller Handling

Symfony 2

- Not yet available as a full-stack MVC framework
- Some components have already been merged into Symfony 1
  - Event Dispatcher
  - Form Framework
- Other new components will soon be released as standalone components:
  - Controller Handling
  - Templating Framework
  - Dependency Injection Container

symfony 1: Not fast enough?

symfony 1 is
one of the slowest framework
when you test it against
a simple Hello World application



based on numbers from `http://paul-m-jones.com/?p=315`

Conclusion?

Don’t use symfony
for your next « Hello World » website

Use PHP ;)

By the way,

the fastest implemention
of a Hello World application with PHP:

```php
die('Hello World');
```

But symfony 1 is probably fast enough for your next website

… anyway, it is fast enough for Yahoo!

Yahoo! Bookmarks	`sf-to.org/bookmarks`
Yahoo! Answers	`sf-to.org/answers`
delicious.com	`sf-to.org/delicious`

… and recently dailymotion.com announced its migration to Symfony

sf-to.org/dailymotion

Secondmost popular video sharing website

One of the *top 50* websites in the world

42 million unique users in December

…and of course
many other smaller websites…

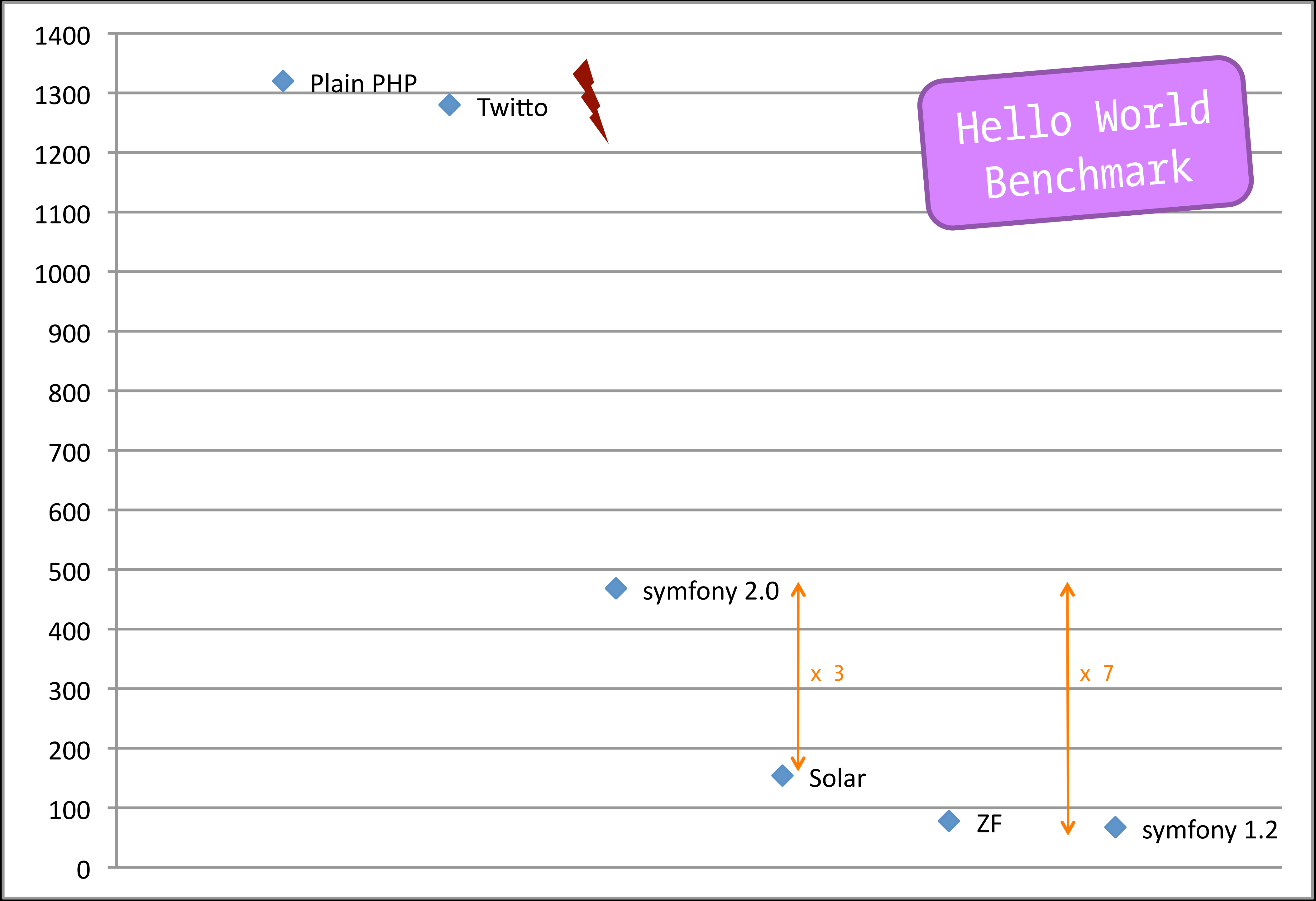
Symfony 2: Faster?

~~Symfony 2 core is so light and flexible that you can easily customize it to have outstanding performance for a Hello World application~~

Symfony 2 core is so light and flexible that its raw performance is outstanding

Hello World with Symfony 2.0

```php
require_once dirname(__FILE__).'/sf20/autoload2/sfCore2Autoload.class.php';
sfCore2Autoload::register();

$app = new HelloApplication();
$app->run()->send();

class HelloApplication
{
  public function __construct()
  {
    $this->dispatcher = new sfEventDispatcher();
    $this->dispatcher->connect('application.load_controller', array($this, 'loadController'));
  }

  public function run()
  {
    $request = new sfWebRequest($this->dispatcher);
    $handler = new sfRequestHandler($this->dispatcher);
    $response = $handler->handle($request);

    return $response;
  }

  public function loadController(sfEvent $event)
  {
    $event->setReturnValue(array(array($this, 'hello'), array($this->dispatcher, $event['request'])));

    return true;
  }

  public function hello($dispatcher, $request)
  {
    $response = new sfWebResponse($dispatcher);
    $response->setContent('Hello World');

    return $response;
  }
}
```

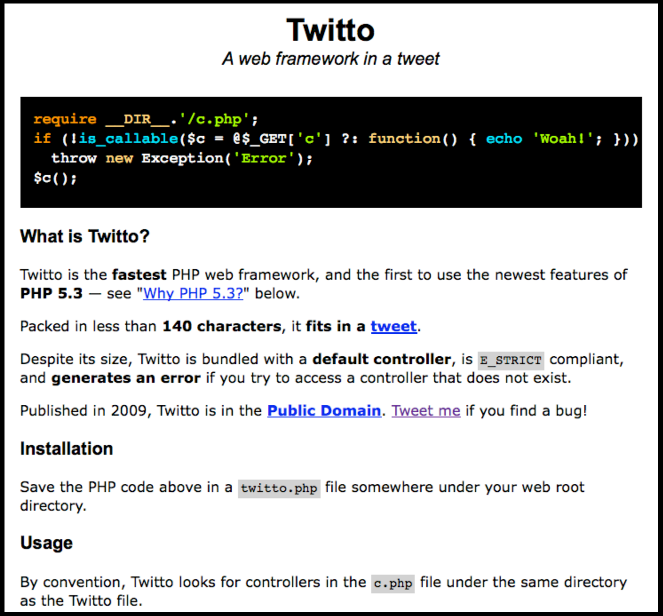


based on numbers from `http://paul-m-jones.com/?p=315`

Twitto ?!

Twitto: The PHP framework that fits in a tweet

- The fastest framework around?
- Uses some PHP 5.3 new features
- It also fits in a slide…



Twitto

A web framework in a tweet

```
require __DIR__.'/c.php';
if (!is_callable($c = @$_GET['c'] ?: function() { echo 'Woah!'; }))
  throw new Exception('Error');
$c();
```

What is Twitto?

Twitto is the **fastest** PHP web framework, and the first to use the newest features of **PHP 5.3** — see "Why PHP 5.3?" below.

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

Despite its size, Twitto is bundled with a **default controller**, is `E_STRICT` compliant, and **generates an error** if you try to access a controller that does not exist.

Published in 2009, Twitto is in the **Public Domain**. Tweet me if you find a bug!

Installation

Save the PHP code above in a `twitto.php` file somewhere under your web root directory.

Usage

By convention, Twitto looks for controllers in the `c.php` file under the same directory as the Twitto file.

```
require __DIR__.'/c.php';
if (!is_callable($c = @$_GET['c'] ?:
function() { echo 'Woah!'; }))
  throw new Exception('Error');
$c();
```

twitto.org

symfony

SENSIOLABS

Don’t use Twitto for your next website

It is a joke ;)

7 times faster ?!

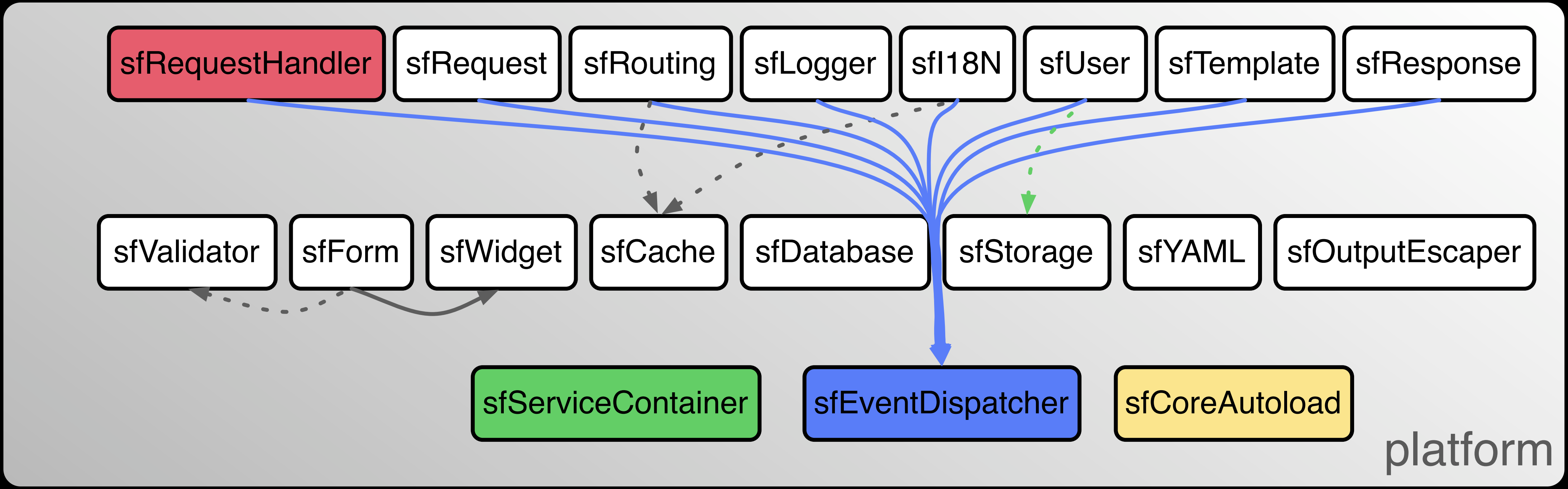
You won’t have such a difference for real applications
as most of the time, the limiting factor
is not the framework itself

7 times faster ?!

- But raw speed matters because
  - It demonstrates that the core « kernel » is very light
  - It allows you to use several Symfony frameworks within a single application with the same behavior but different optimizations:
    - One full-stack framework optimized for ease of use (think symfony 1)
    - One light framework optimized for speed (think Rails Metal ;))

symfony platform

2.0



Symfony 2 ~~kernel:~~

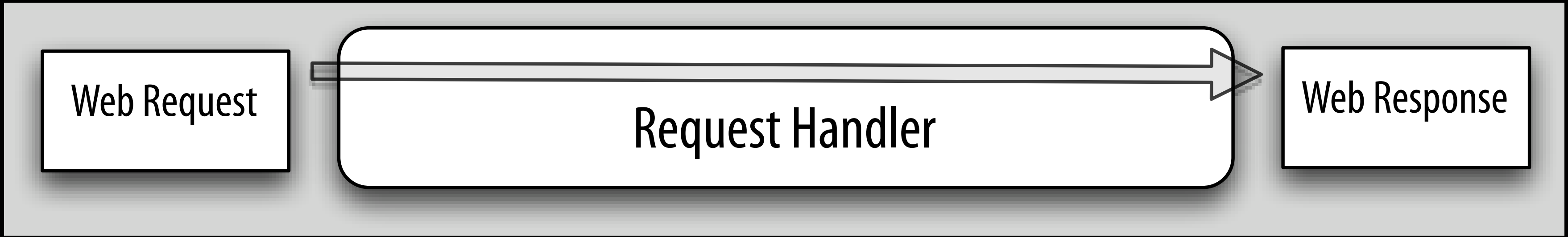
The Request Handler

Symfony 2 secret weapon:

The Request Handler

The Request Handler

- The backbone of Symfony 2 controller implementation
- Class to build web frameworks, not only MVC ones
- Based on a simple assumption:
  - The input is a request object
  - The output is a response object
- The request object can be anything you want
- The response object must implement a `send()` method



The Request Handler

```php
$handler = new sfRequestHandler($dispatcher);

$request = new sfWebRequest($dispatcher);
$response = $handler->handle($request);

$response->send();
```

The Request Handler

- The sfRequestHandler does several things:
  - Notify events
  - Execute a callable (the controller)
  - Ensure that the Request is converted to a Response object
- The framework is responsible for choosing the controller
- The controller is responsible for the conversion of the Request to a Response

```php
class sfRequestHandler
{
  protected $dispatcher = null;

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

  public function handle($request)
  {
    try
    {
      return $this->handleRaw($request);
    }
    catch (Exception $e)
    {
      $event = $this->dispatcher->notifyUntil(new sfEvent($this, 'application.exception', array('request' => $request, 'exception' => $e)));
      if ($event->isProcessed())
      {
        return $this->filterResponse($event->getReturnValue(), 'An "application.exception" listener returned a non response object.');
      }

      throw $e;
    }
  }

  public function handleRaw($request)
  {
    $event = $this->dispatcher->notifyUntil(new sfEvent($this, 'application.request', array('request' => $request)));
    if ($event->isProcessed())
    {
      return $this->filterResponse($event->getReturnValue(), 'An "application.request" listener returned a non response object.');
    }

    $event = $this->dispatcher->notifyUntil(new sfEvent($this, 'application.load_controller', array('request' => $request)));
    if (!$event->isProcessed())
    {
      throw new Exception('Unable to load the controller.');
    }

    list($controller, $arguments) = $event->getReturnValue();

    if (!is_callable($controller))
    {
      throw new Exception(sprintf('The controller must be a callable (%s).', var_export($controller, true)));
    }

    $event = $this->dispatcher->notifyUntil(new sfEvent($this, 'application.controller', array('request' => $request, 'controller' => &$controller, 'arguments' => &$arguments)));
    if ($event->isProcessed())
    {
      try
      {
        return $this->filterResponse($event->getReturnValue(), 'An "application.controller" listener returned a non response object.');
      }
      catch (Exception $e)
      {
        $retval = $event->getReturnValue();
      }
    }
    else
    {
      $retval = call_user_func_array($controller, $arguments);
    }

    $event = $this->dispatcher->filter(new sfEvent($this, 'application.view'), $retval);

    return $this->filterResponse($event->getReturnValue(), sprintf('The controller must return a response (instead of %s).', is_object($event->getReturnValue()) ? 'an object of class '.get_class($event->getReturnValue()) : (string) $event->getReturnValue()));
  }

  protected function filterResponse($response, $message)
  {
    if (!is_object($response) || !method_exists($response, 'send'))
    {
      throw new RuntimeException($message);
    }

    $event = $this->dispatcher->filter(new sfEvent($this, 'application.response'), $response);
    $response = $event->getReturnValue();

    if (!is_object($response) || !method_exists($response, 'send'))
    {
      throw new RuntimeException('An "application.response" listener returned a non response object.');
    }

    return $response;
  }
}
```

sfRequestHandler is less than 100 lines of PHP code!

symfony

SENSIOLABS

Request Handler Events

```
application.request

application.load_controller

application.controller

application.view

application.response

application.exception
```

`application.response`

As the very last event notified, a listener can modify the Response object just before it is returned to the user

application.request

- The very first event notified
- It can act as a short-circuit event
- If one listener returns a Response object, it stops the processing

application.load_controller

- Only event for which at least one listener must be connected to
- A listener must return
  - A PHP callable (the controller)
  - The arguments to pass to the callable

application.view

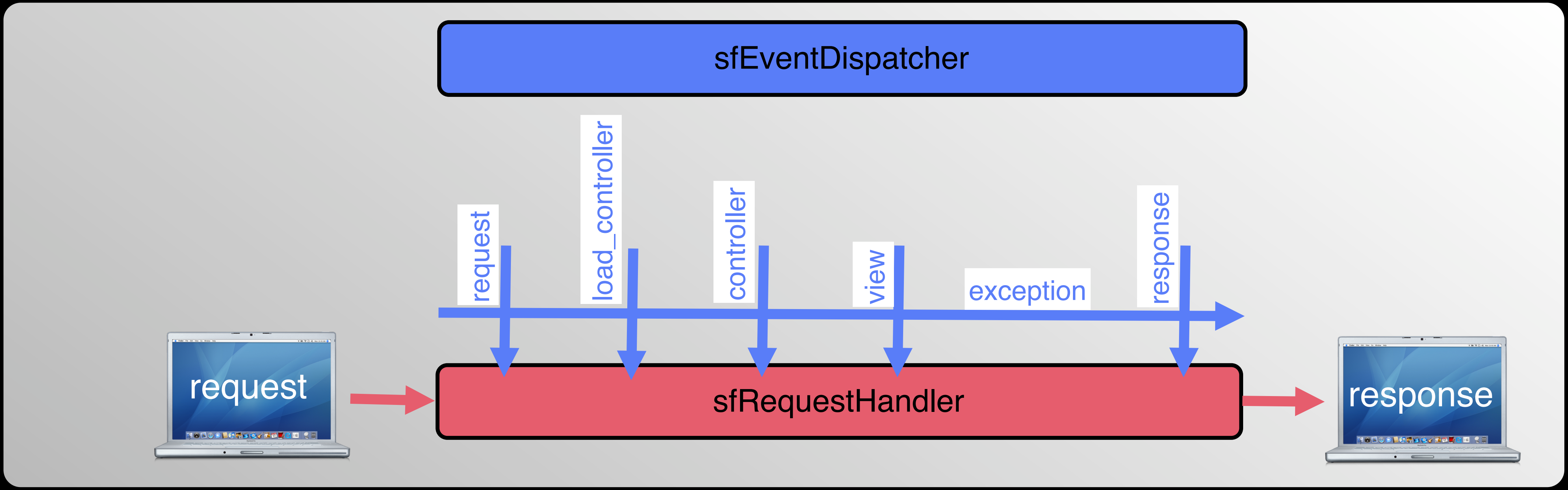
The controller must return a Response object
except if a listener can convert
the controller return value to a Response

application.exception

The request handler catches all exceptions
and give a chance to listeners
to return a Response object

Request Handler

- Several listeners can be attached to a single event
- Listeners are called in turn



Hello World with Symfony 2.0

```php
require_once dirname(__FILE__).'/sf20/autoload2/sfCore2Autoload.class.php';
sfCore2Autoload::register();

$app = new HelloApplication();
$app->run()->send();

class HelloApplication
{
  public function __construct()
  {
    $this->dispatcher = new sfEventDispatcher();
    $this->dispatcher->connect('application.load_controller', array($this, 'loadController'));
  }

  public function run()
  {
    $request = new sfWebRequest($this->dispatcher);
    $handler = new sfRequestHandler($this->dispatcher);
    $response = $handler->handle($request);

    return $response;
  }

  public function loadController(sfEvent $event)
  {
    $event->setReturnValue(array(array($this, 'hello'), array($this->dispatcher, $event['request'])));

    return true;
  }

  public function hello($dispatcher, $request)
  {
    $response = new sfWebResponse($dispatcher);
    $response->setContent('Hello World');

    return $response;
  }
}
```

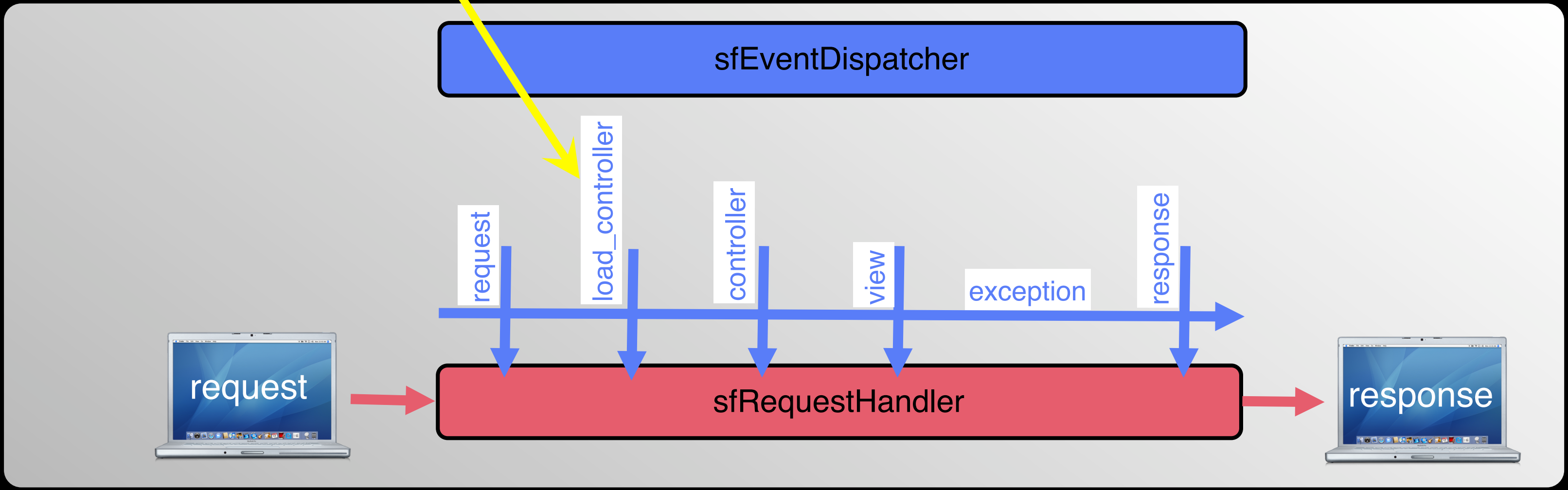
symfony

SENSIOLABS

```php
require_once '/path/to/sfCore2Autoload.class.php';
sfCore2Autoload::register();
```

```php
$app = new HelloApplication();
$app->run()->send();
```

```php
public function __construct()
 {
   $this->dispatcher = new sfEventDispatcher();
   $this->dispatcher->connect(
     'application.load_controller',
     array($this, 'loadController')
   );
 }
```



```php
public function loadController(sfEvent $event)
 {
   $event->setReturnValue(array(
    array($this, 'hello'),
    array($this->dispatcher, $event['request'])
   ));

   return true;
 }
```

```php
public function hello($dispatcher, $request)
 {
   $response = new sfWebResponse($dispatcher);
   $response->setContent('Hello World');

   return $response;
 }
```

```php
public function run()
 {
   $request = new sfWebRequest($this->dispatcher);
   $handler = new sfRequestHandler($this->dispatcher);
   $response = $handler->handle($request);

   return $response;
 }
```

Case study: dailymotion.com

- The problem: the Dailymotion developers add new features on a nearly everyday basis
- The challenge: Migrate by introducing small doses of Symfony goodness
- The process
  - Wrap everything with sfRequestHandler by implementing an application.load_controller listener that calls the old code, based on the request
  - Migrate the mod_rewrite rules to the symfony routing
  - Add unit and functional tests

Symfony 2: The Templating Framework

New Templating Framework

- 4 components
  - Template Engine
  - Template Renderers
  - Template Loaders
  - Template Storages

- Independant library

```php
require_once '/path/to/sfCore2Autoload.class.php';
sfCore2Autoload::register();

$dispatcher = new sfEventDispatcher();

$loader = new sfTemplateLoaderFilesystem($dispatcher,
  '/path/to/templates/%s.php');

$t = new sfTemplateEngine($dispatcher, $loader);

echo $t->render('index', array('name' => 'Fabien'));
```

Template Loaders

- No assumption about where and how templates are to be found
  - Filesystem
  - Database
  - Memory
  - …
- Template names are « logical » names:

```
$loader = new sfTemplateLoaderFilesystem($dispatcher,
              '/path/to/templates/%s.php');
```

Template Renderers

- No assumption about the format of the templates
- Template names are prefixed with the renderer name:
  - `index == php:index`
  - `user:index`

```php
$t = new sfTemplateEngine($dispatcher, $loader, array(
  'user' => new ProjectTemplateRenderer($dispatcher),
  'php'  => new sfTemplateRendererPhp($dispatcher),
));
```

Template Embedding

```
Hello <?php echo $name ?>

<?php $this->render('embedded', array('name' => $name)) ?>

<?php $this->render('smarty:embedded') ?>
```

Template Inheritance

```
<?php $this->decorator('layout') ?>

Hello <?php echo $name ?>
```

```
<html>
  <head>
  </head>
  <body>
    <?php $this->output('content') ?>
  </body>
</html>
```

Template Slots

```
<html>
  <head>
    <title><?php $this->output('title') ?></title>
  </head>
  <body>
    <?php $this->output('content') ?>
  </body>
</html>
```

```
<?php $this->set('title', 'Hello World! ') ?>
```

```
<?php $this->start('title') ?>
  Hello World!
<?php $this->stop() ?>
```

Template Multiple Inheritance

A layout can be decorated by another layout

Each layout can override slots

Templating: An example

CMS Templating

- Imagine a CMS with the following features:
  - The CMS comes bundled with default templates
  - The developer can override default templates for a specific project
  - The webmaster can override some templates
- The CMS and developer templates are stored on the filesystem and are written with pure PHP code
- The webmaster templates are stored in a database and are written in a simple templating language: `Hello {{ name }}`

CMS Templating

- The CMS has several built-in sections and pages
  - Each page is decorated by a layout, depending on the section
  - Each section layout is decorated by a base layout

```
cms/templates/              project/templates/
  base.php                    base.php
  articles/                   articles/
    layout.php                  layout.php
    article.php                 article.php
                                content.php
```

articles/content.php

```
<h1>{{ title }}</h1>

<p>
  {{ content }}
</p>
```

articles/article.php

```php
<?php $this->decorator('articles/layout') ?>

<?php $this->set('title', $title) ?>

<?php echo $this->render(
  'user:articles/content',
  array('title' => $title, 'content' => $content)
) ?>
```

articles/layout.php

```
<?php $this->decorator('base') ?>

<?php $this->set('title', 'Articles | '.$this->get('title')) ?>

<?php $this->start('head') ?>
  <?php $this->output('head') ?>
  <link rel="stylesheet" type="text/css" media="all" href="/css/
articles.css" />
<?php $this->stop() ?>

<?php $this->output('content') ?>
```

base.php

```
<html>
  <head>
    <title>
      <?php $this->output('title') ?>
    </title>
    <?php $this->output('head') ?>
  </head>
  <body>
    <?php $this->output('content') ?>
  </body>
</html>
```

Template Renderer

```php
$t = new sfTemplateEngine($dispatcher, $loader, array(
  'user' => new ProjectTemplateRenderer($dispatcher),
  'php'  => new sfTemplateRendererPhp($dispatcher),
));
```

Template Renderer

```
class ProjectTemplateRenderer extends sfTemplateRenderer
{
  public function evaluate($template, array $parameters = array())
  {
    if ($template instanceof sfTemplateStorageFile)
    {
      $template = file_get_contents($template);
    }

    $this->parameters = $parameters;

    return preg_replace_callback('/{{\s*(.+?)\s*}}/', array($this,
'replaceParameters'), $template);
  }

  public function replaceParameters($matches)
  {
    return isset($this->parameters[$matches[1]]) ? $this->parameters[$matches[1]] :
null;
  }
}
```

Template Loaders

```php
$loader = new sfTemplateLoaderFilesystem($dispatcher,
  array(
    '/path/to/project/templates/%s.php',
    '/path/to/cms/templates/%s.php'
  ));
```

Template Loader Chain

```php
$loader = new sfTemplateLoaderChain($dispatcher, array(
  new ProjectTemplateLoader(
    $dispatcher, array('pdo' => $pdo)),
  new sfTemplateLoaderFilesystem($dispatcher, array(
   '/path/to/project/templates/%s.php',
   '/path/to/cms/templates/%s.php'
  )),
));
```

Database Template Loader

```php
class ProjectTemplateLoader extends sfTemplateLoader
{
  public function load($template)
  {
    $stmt = $this->options['pdo']->prepare('SELECT tpl FROM tpl WHERE name = :name');
    try
    {
      $stmt->execute(array('name' => $template));
      if (count($rows = $stmt->fetchAll(PDO::FETCH_NUM)))
      {
        return $rows[0][0];
      }
    }
    catch (PDOException $e)
    {
    }

    return false;
  }
}
```

Database Template Loader

```
$pdo = new PDO('sqlite::memory:');
$pdo->exec('CREATE TABLE tpl (name, tpl)');
$pdo->exec('INSERT INTO tpl (name, tpl) VALUES
("articles/content", "{{ title }} {{ name }}")');
```

Template Loader Cache

```php
$loader = new sfTemplateLoaderCache(
  $dispatcher,
  $loader,
  new sfFileCache(array('dir' => 'path/to/cache'))
);
```

```php
$pdo = new PDO('sqlite::memory:');
$pdo->exec('CREATE TABLE tpl (name, tpl)');
$pdo->exec('INSERT INTO tpl (name, tpl) VALUES ("articles/content", "{{ title }}
{{ name }}")');


$loader = new sfTemplateLoaderCache(
  $dispatcher,
  new sfTemplateLoaderChain($dispatcher, array(
    new ProjectTemplateLoader($dispatcher, array('pdo' => $pdo)),
    new sfTemplateLoaderFilesystem($dispatcher, array(
     '/path/to/project/templates/%s.php',
     '/path/to/cms/templates/%s.php'
    )),
  )),
  new sfFileCache(array('dir' => 'path/to/cache'))
);


$t = new sfTemplateEngine($dispatcher, $loader, array(
  'user' => new ProjectTemplateRenderer($dispatcher)
));


$t->render('articles/article', array('title' => 'Title', 'content' => 'Lorem...'));
```

Symfony 2: Dependency Injection Container

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

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

The Symfony 2 dependency injection container replaces several symfony 1 concepts into one integrated system:

- sfContext
- sfConfiguration
- sfConfig
- factories.yml
- settings.yml / logging.yml / i18n.yml

DI Hello World example

```php
class Message
{
  public function __construct(OutputInterface $output, array $options)
  {
    $this->output = $output;
    $this->options = array_merge(array('with_newline' => false), $options);
  }

  public function say($msg)
  {
     $this->output->render($msg.($this->options['with_newline'] ? "\n" : ''));
  }
}
```

DI Hello World example

```php
interface OutputInterface
{
  public function render($msg);
}

class Output implements OutputInterface
{
  public function render($msg)
  {
    echo $msg;
  }
}

class FancyOutput implements OutputInterface
{
  public function render($msg)
  {
    echo sprintf("\033[33m%s\033[0m", $msg);
  }
}
```

DI Hello World example

```php
$output = new FancyOutput();
$message = new Message($output, array('with_newline' => true));
$message->say('Hello World');
```

A DI container facilitates
objects description and object relationships,
configures and instantiates objects

DI Container Hello World example

```
$container = new sfServiceContainer();

$outputDef = new sfServiceDefinition('FancyOutput');
$container->setServiceDefinition('output', $outputDef);

$msgDef = new sfServiceDefinition(
  'Message',
  array(new sfServiceReference('output'), array('with_newline' => true))
);
$container->setServiceDefinition('message', $msgDef);

$container->message->say('Hello World!');
```

```
$message = $container->message;
```

Get the configuration for the message service

The Message constructor must be given an output service

Get the output object from the container

Create a Message object by passing the constructor arguments

```php
$message = $container->message;
```

is roughly equivalent to

```php
          $output = new FancyOutput();
$message = new Message($output, array('with_newline' => true));
```

```php
$container = new sfServiceContainer();

$msgDef = new sfServiceDefinition(
  'Message',
  array(new sfServiceReference('output'), array('with_newline' => true))
);
$outputDef = new sfServiceDefinition('FancyOutput');

$container->setServiceDefinition('message', $msgDef);
$container->setServiceDefinition('output', $outputDef);
```

PHP

```xml
<services>
  <service id="output" class="FancyOutput" />

  <service id="message" class="Message">
    <argument type="service" id="output" />
    <argument type="collection">
      <argument key="with_newline">true</argument>
    </argument>
  </service>
</services>
```

XML

```php
$container = new sfServiceContainer(new sfServiceLoaderXml());
$container->load('services.xml');
```

```xml
<services>
  <parameters>
    <parameter key="output.class">FancyOutput</parameter>
    <parameter key="message.options" type="collection">
      <parameter key="with_newline">true</parameter>
    </parameter>
  </parameters>

  <service id="output" class="%output.class%" />

  <service id="message" class="Message">
    <argument type="service" id="output" />
    <argument>%message.options%</argument>
  </service>
</services>
```

```php
$container = new sfServiceContainer(new sfServiceLoaderXml());
$container->load('services.xml');
```

```xml
<services>
 <import resource="config.xml" />

  <service id="output" class="%output.class%" />
  <service id="message" class="Message">
    <argument type="service" id="output" />
    <argument>%message.options%</argument>
  </service>
</services>
```

```xml
<services>
  <parameters>
    <parameter key="output.class">FancyOutput</parameter>
    <parameter key="message.options" type="collection">
      <parameter key="with_newline">true</parameter>
    </parameter>
  </parameters>
</services>
```

```php
$container = new sfServiceContainer(new sfServiceLoaderXml());
$container->load('services.xml');
```

```xml
<services>
 <import resource="config.yml" class="sfServiceLoaderYamlParameters" />

  <service id="output" class="%output.class%" />
  <service id="message" class="Message">
    <argument type="service" id="output" />
    <argument>%message.options%</argument>
  </service>
</services>
```

```yaml
output.class: FancyOutput

message.options:
  with_newline: true
```

```php
$container = new sfServiceContainer(new sfServiceLoaderXml());
$container->load('services.xml');
```

```php
$pdo = new PDO('sqlite::memory:');
$pdo->exec('CREATE TABLE tpl (name, tpl)');
$pdo->exec('INSERT INTO tpl (name, tpl) VALUES ("articles/content", "{{ title }}
{{ name }}")');

$loader = new sfTemplateLoaderCache(
  $dispatcher,
  new sfTemplateLoaderChain($dispatcher, array(
    new ProjectTemplateLoader($dispatcher, array('pdo' => $pdo)),
    new sfTemplateLoaderFilesystem($dispatcher, array(
     '/path/to/project/templates/%s.php',
     '/path/to/cms/templates/%s.php'
    )),
  )),
  new sfFileCache(array('dir' => 'path/to/cache'))
);

$t = new sfTemplateEngine($dispatcher, $loader, array(
  'user' => new ProjectTemplateRenderer($dispatcher)
));

$t->render('articles/article', array('title' => 'Title', 'content' => 'Lorem...'));
```

```php
$pdo = new PDO('sqlite::memory:');
```

```xml
<service id="pdo" class="PDO">
  <argument>sqlite::memory:</argument>
</service>
```

```php
$container = new sfServiceContainer(new sfServiceLoaderXml());
$container->load(dirname(__FILE__).'/cms.xml');

$pdo->exec('CREATE TABLE tpl (name, tpl)');
$pdo->exec('INSERT INTO tpl (name, tpl) VALUES ("articles/content",
"{{ title }} {{ name }}")');

echo $container->template->render('articles/article', array('title' => 'Title',
'content' => 'Lorem...'));
```

```xml
<services>
  <import resource="config.yml" class="sfServiceLoaderYamlParameters" />
  <import resource="template_loader.xml" />

  <service id="event_dispatcher" class="sfEventDispatcher" />

  <service id="pdo" class="PDO">
    <argument>sqlite::memory:</argument>
  </service>

  <service id="template_renderer" class="ProjectTemplateRenderer" lazy="true">
    <argument type="service" id="event_dispatcher" />
  </service>

  <service id="template" class="sfTemplateEngine" lazy="true">
    <argument type="service" id="event_dispatcher" />
    <argument type="service" id="template_loader" />
    <argument type="collection">
      <argument type="service" key="user" id="template_renderer" />
    </argument>
  </service>
</services>
```

```xml
<services>
  <service id="template_loader_project" class="ProjectTemplateLoader">
    <argument type="service" id="event_dispatcher" />
    <argument type="collection"><argument type="service" key="pdo" id="pdo" /></argument>
  </service>

  <service id="template_loader_filesystem" class="sfTemplateLoaderFilesystem">
    <argument type="service" id="event_dispatcher" />
    <argument>%template.filesystem_pattern%</argument>
  </service>

  <service id="template_loader_chain" class="sfTemplateLoaderChain">
    <argument type="service" id="event_dispatcher" />
    <argument type="collection">
      <argument type="service" id="template_loader_project" />
      <argument type="service" id="template_loader_filesystem" />
    </argument>
  </service>

  <service id="template_loader_cache" class="sfFileCache">
    <argument type="collection"><argument key="cache_dir">%application.dir%/cache</argument></argument>
  </service>

  <service id="template_loader" class="sfTemplateLoaderCache" lazy="true">
    <argument type="service" id="event_dispatcher" />
    <argument type="service" id="template_loader_chain" />
    <argument type="service" id="template_cache" />
  </service>
</services>
```

```
<services>
  <service id="template_loader" class="sfTemplateLoaderCache" lazy="true">
    <argument type="service" id="event_dispatcher" />
    <argument type="service">
      <service class="sfTemplateLoaderChain">
        <argument type="service" id="event_dispatcher" />
        <argument type="collection">
          <argument type="service">
            <service class="ProjectTemplateLoader">
              <argument type="service" id="event_dispatcher" />
              <argument type="collection"><argument type="service" key="pdo" id="pdo" /></argument>
            </service>
          </argument>
          <argument type="service">
            <service class="sfTemplateLoaderFilesystem">
              <argument type="service" id="event_dispatcher" />
              <argument>%template.filesystem_patterns%</argument>
            </service>
          </argument>
        </argument>
      </service>
    </argument>
    <argument type="service">
      <service class="sfFileCache">
        <argument type="collection"><argument key="cache_dir">%application.dir%/cache</argument></
argument>
      </service>
    </argument>
  </service>
</services>
```

Questions?

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/