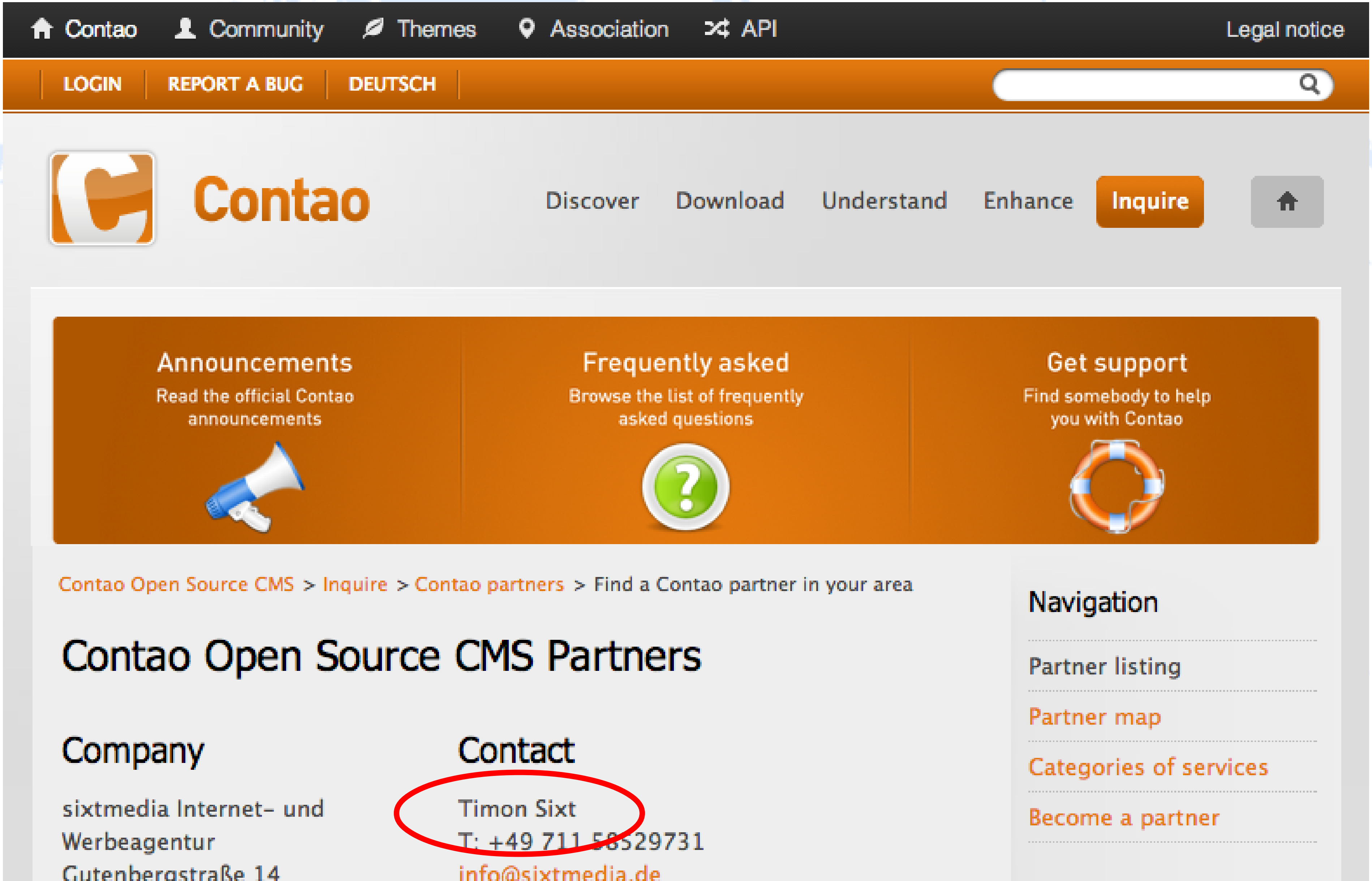
(Meine) Wahrheit über

Symfony

Timon Schroeter www.php-schulung.de

Timon != Timon



Announcements
Read the official Contao announcements

Frequently asked
Browse the list of frequently asked questions

Get support
Find somebody to help you with Contao

Contao Open Source CMS > Inquire > Contao partners > Find a Contao partner in your area

Contao Open Source CMS Partners

Company

sixtmedia Internet- und Werbeagentur
Gutenbergstraße 14

Contact

Timon Sixt
T: +49 711 58529731
info@sixtmedia.de

Timon Schroeter

- www.php-schulung.de
- Schulung, Coaching, Beratung

Version 4.2 (2002)

~~OOP~~ ?

register_globals ?

http://doitandhow.com/2012/07/16/fun-colored-spaghetti/

OOP

C++

C++

HPC

C++

fortran

HPC

C++

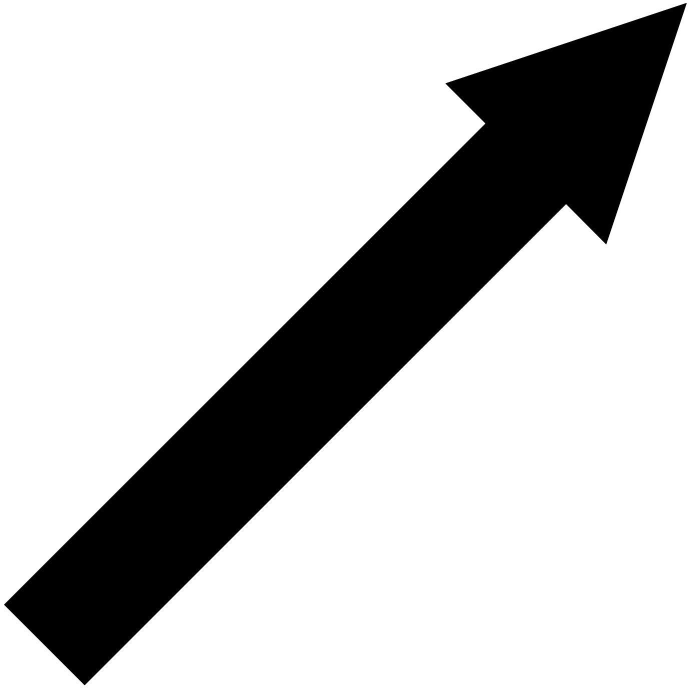
HPC

fortran

Matlab

php

professionell entwickeln



rumbasteln

TDD

TDD

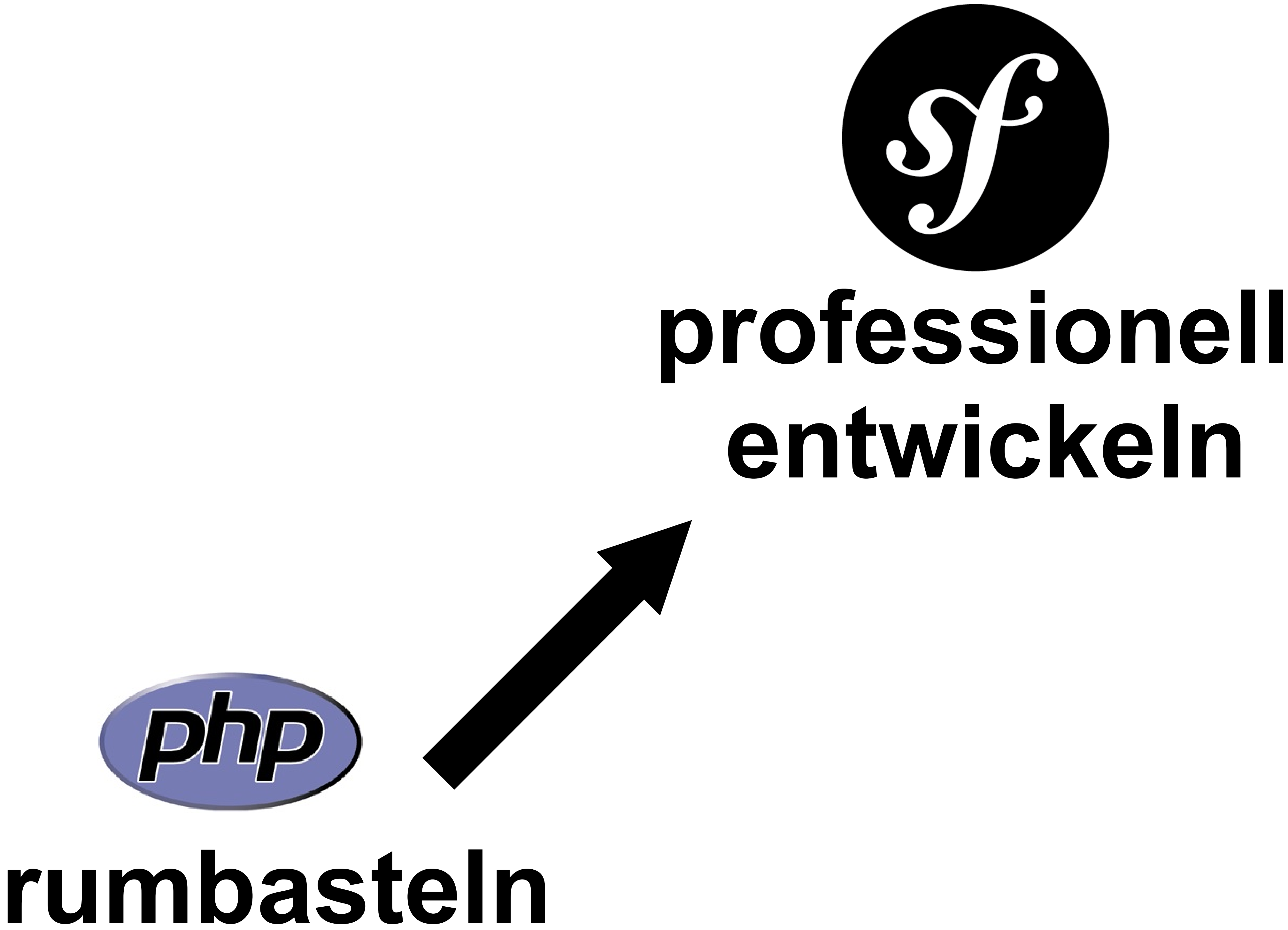
~~unit~~

TDD

~~unit~~

OOP ~~+ DI~~

Version 2.0 (2011)



professionell entwickeln

rumbasteln

Forms

Validation

TWIG

uvm.

Forms

Validation

TWIG

uvm.

Dependency Injection
Container

Event
Dispatcher

php

Was sind die Kennzeichen dieser Paradigmen?

- Imperativ
- Prozedural
- Objektorientiert

Was sind die Kennzeichen dieser Paradigmen?

- Imperativ
  - Sequenz, Schleife, Verzweigung

Was sind die Kennzeichen dieser Paradigmen?

- Imperativ
  - Sequenz, Schleife, Verzweigung
- Prozedural
  - Funktionen die alle global verfügbar sind

http://doitandhow.com/2012/07/16/fun-colored-spaghetti/

Was sind die Kennzeichen dieser Paradigmen?

- Imperativ
  - Sequenz, Schleife, Verzweigung
- Prozedural
  - Funktionen die alle global verfügbar sind
- Objektorientiert
  - Objekte, Eigenschaften, Methoden usw.

Was sind die Kennzeichen dieser Paradigmen?

- Imperativ
  - Sequenz, Schleife, Verzweigung
- Prozedural
  - Funktionen die alle global verfügbar sind
- Objektorientiert
  - Objekte, Eigenschaften, Methoden usw.
  - Funktionen in abgegrenzten Kontexten verfügbar

http://www.spieltischshop.de/lego-soft.html

Was sind die Kennzeichen dieser Paradigmen?

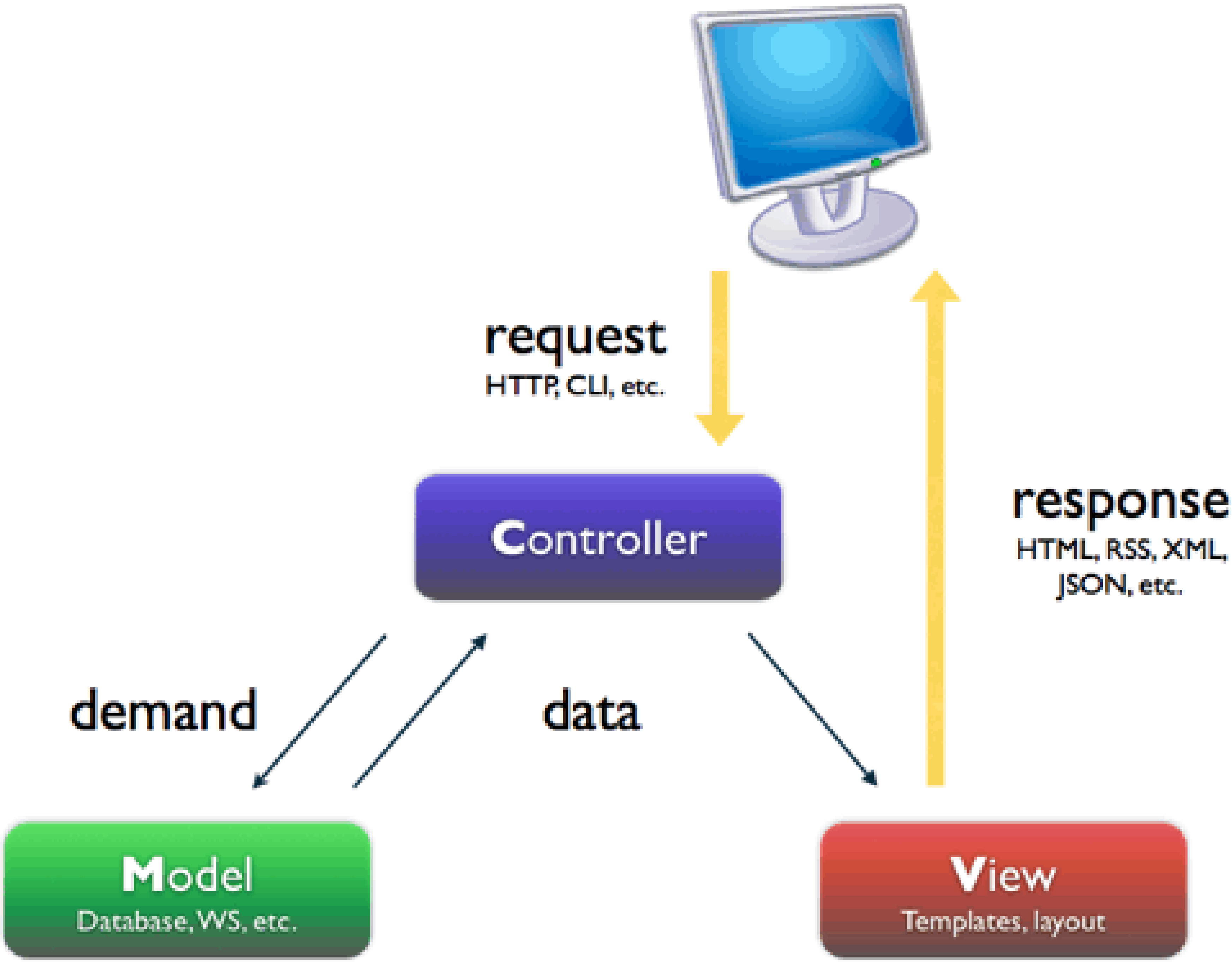
- Imperativ
  - Sequenz, Schleife, Verzweigung
- Prozedural
  - Funktionen die alle global verfügbar sind
- Objektorientiert
  - Objekte, Eigenschaften, Methoden usw.
  - Funktionen in abgegrenzten Kontexten verfügbar

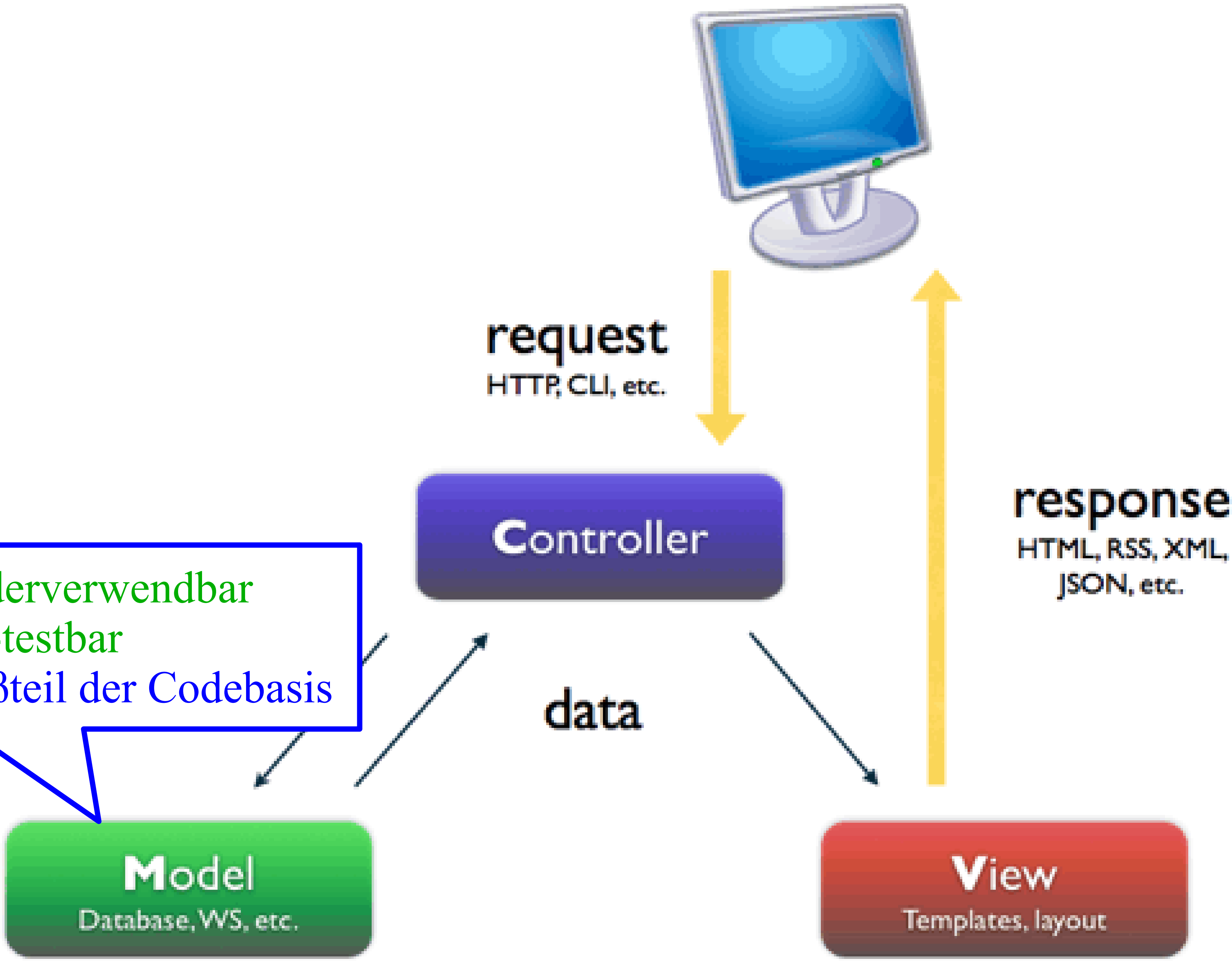
Was sind die Kennzeichen dieser Paradigmen?

- Imperativ
  - Sequenz, Schleife, Verzweigung
- Prozedural
  - Funktionen die alle global verfügbar sind
- Objektorientiert
  - Objekte, Eigenschaften, Methoden usw.
  - Funktionen in abgegrenzten Kontexten verfügbar
  - feste Abhängigkeiten zwischen Klassen

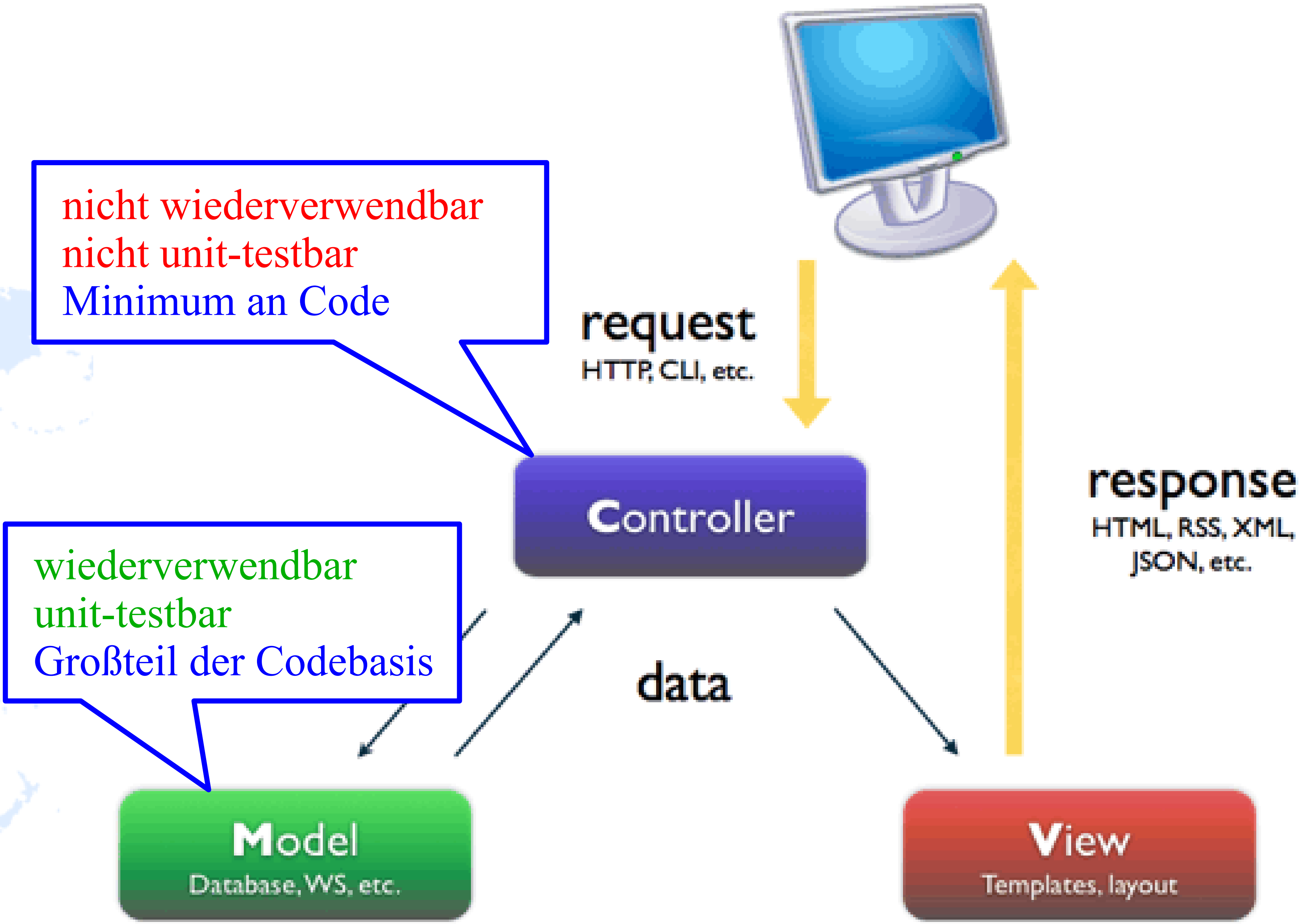
Was sind die Kennzeichen dieser Paradigmen?

- Imperativ
  - Sequenz, Schleife, Verzweigung
- Prozedural
  - Funktionen die alle global verfügbar sind
- Objektorientiert
  - Objekte, Eigenschaften, Methoden usw.
  - Funktionen in abgegrenzten Kontexten verfügbar
  - feste Abhängigkeiten zwischen Klassen
- OOP mit Dependency Injection
  - Instanzen variabel kombinierbar (Schraubendreher)


request
HTTP, CLI, etc.
Controller
response
HTML, RSS, XML,
JSON, etc.
demand
data
Model
Database, WS, etc.
View
Templates, layout

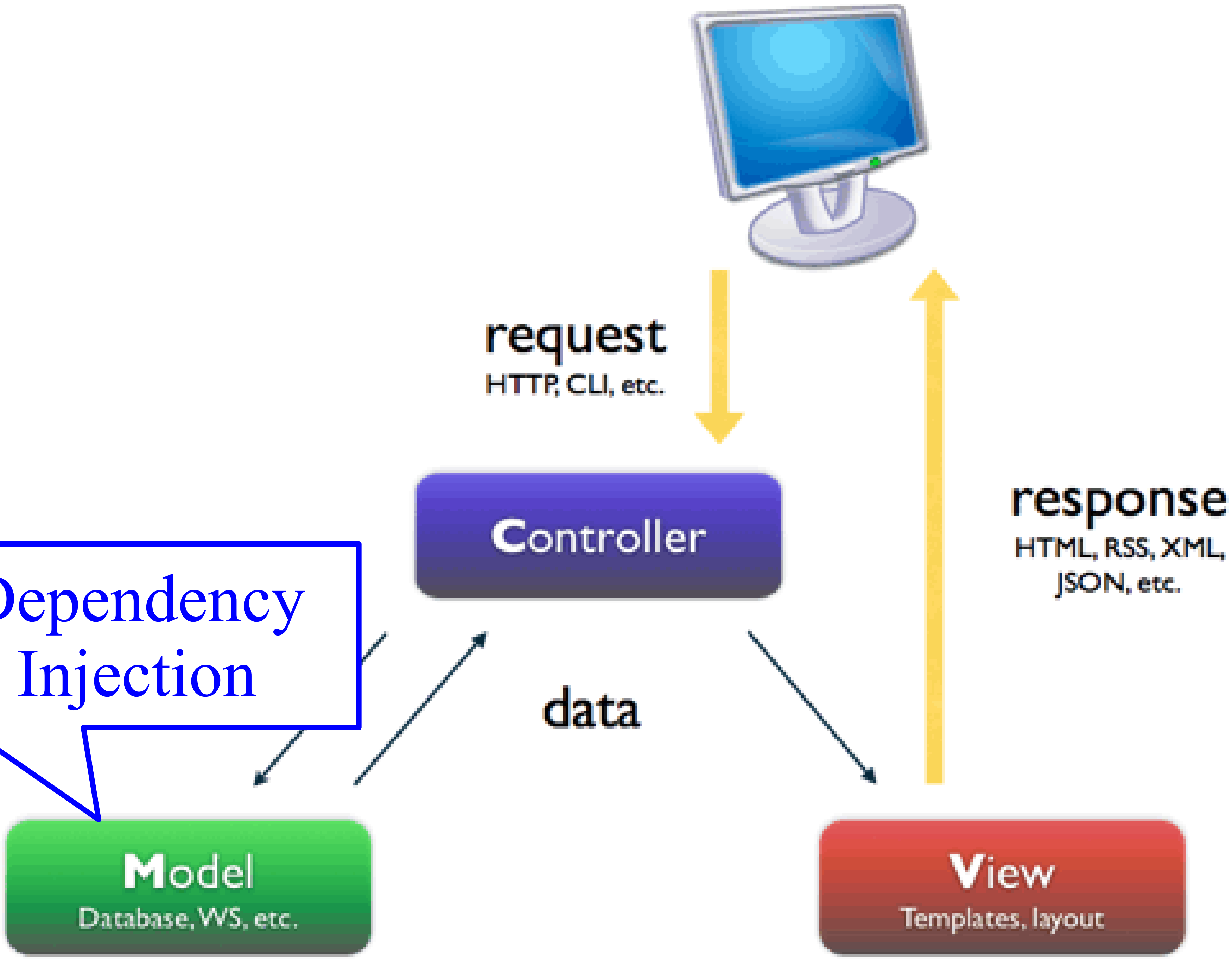


wiederverwendbar
unit-testbar
Großteil der Codebasis



nicht wiederverwendbar
nicht unit-testbar
Minimum an Code

wiederverwendbar
unit-testbar
Großteil der Codebasis



Dependency Injection

Dependency Injection

DI

Structure of the next slides

- Code example: DI for generic PHP classes
- Code example: DI in Symfony 2

You are here

```
<?php
namespace Acme\FeedBundle\Service\FeedAggrega
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

class FeedAggregator {
   private $client;
   private $logger;

   public function __construct () {

      $this->client = new Client();
      $this->logger = new XmlLogger();
   }

   public function retrieveFeed ($baseurl, $path) {
      $request = $this->client->setBaseUrl($baseurl)->get($path);
      $response = $request->send();
      if (200 != $response->getStatusCode()) {
         $this->logger->log('Could not get: '.$baseurl.$path);
         return null;
      }

      return $response->getBody();
   }
   // ...
}
```

Why is this class difficult to unit test?

```php
<?php
namespace Acme\FeedBundle\Service\FeedAggrega
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

class FeedAggregator {
   private $client;
   private $logger;

   public function __construct () {

      $this->client = new Client();
      $this->logger = new XmlLogger();
   }

   public function retrieveFeed ($baseurl, $path) {
      $request = $this->client->setBaseUrl($baseurl)->get($path);
      $response = $request->send();
      if (200 != $response->getStatusCode()) {
         $this->logger->log('Could not get: '.$baseurl.$path);
         return null;
      }

      return $response->getBody();
   }
   // ...
}
```

Why is this class difficult to unit test?

What if we want unit tests to run fast without waiting for the network?

```php
<?php
namespace Acme\FeedBundle\Ser                ga
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

class FeedAggregator {
    private $client;
    private $logger;

    public function __construct () {

        $this->client = new Client();
        $this->logger = new XmlLogger();
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $request->send();
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$baseurl.$path);
            return null;
        }

        return $response->getBody();
    }
    // ...
}
```

What if we ever want to use a different HTTP client?

Why is this class difficult to unit test?

What if we want unit tests to run fast without waiting for the network?

```
<?php
namespace Acme\FeedBundle\Serv            ga
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

class F     gregator {
             lient;
             ogger;

             ction __construct () {

        $this->client = new Client();
        $this->logger = new XmlLogger();
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $request->send();
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$baseurl.$path);
            return null;
        }

        return $response->getBody();
    }
    // ...
}
```

What if we ever want to use a different HTTP client?

Why is this class difficult to unit test?

What if we ever want to use a different logger class?

What if we want unit tests to run fast without waiting for the network?

```
<?php
namespace Acme\FeedBundle\Service\FeedAggregator;
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

class FeedAggregator {
    private $client;
    private $logger;

    public function __construct () {
        $this->client = new Client();
        $this->logger = new XmlLogger();
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $request->send();
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$baseurl.$path);
            return null;
        }

        return $response->getBody();
    }
    // ...
}
```

Why is this class difficult to unit test?

What if we ever want to use a different HTTP client?

What if we ever want to use a different logger class?

What if we ever want to use a different log format?

What if we want unit tests to run fast without waiting for the network?

```
<?php
namespace Acme\FeedBundle\Serv                      ga
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

          re

         cli
         og

        ct                   uct () {
        $this->client = new Client();
        $this->logger = new XmlLogger();
    }

    public function retrieveFee
        $request = $this->client                                    );
        $response = $request->send();
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$baseurl.$path);
            return null;
        }

        return $response->getBody();
    }
    // ...
}
```

What if we ever want to use a different HTTP client?

Why is this class difficult to unit test?

What if we ever want to use a different logger class?

What if we ever want to use a different log format?

What if we want unit tests to run fast without waiting for the network?

What if we want unit tests to run fast without logging?

```
<?php
namespace Acme\FeedBundle\Ser                     ga
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

            re              {
          $cli
          $lo

              c

        $this->c
        $this->l
    }

    public function retrieveFee
        $request = $this->client                             );
        $response = $request->send();
        if (200 != $response->getS    usCode()) {
            $this->logger->log('Could not get: '.$baseurl.$path);
            return null;
        }

        return $response->getBody();
    }
    // ...
}
```

What if we ever want to use a different HTTP client?

Why is this class difficult to unit test?

What if we ever want to use a different logger class?

What if we ever want to

Dependencies are **pulled**.
=> Replacing requires refactoring
=> Dynamic replacing (e.g. for testing) is **impossible**

vant unit tests to run fast
iting for the network?

What if we want unit tests to run fast without logging?

Dependencies are **pulled**.

```
<?php
namespace Acme\FeedBundle\Service\FeedAg
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

class FeedAggregator {
   private $client;
   private $logger;

   public function __construct () {

      $this->client = new Client();
      $this->logger = new XmlLogger();
   }

   public function retrieveFeed ($baseurl, $path) {
      $request = $this->client->setBaseUrl($baseurl)->get($path);
      $response = $request->send();
      if (200 != $response->getStatusCode()) {
         $this->logger->log('Could not get: '.$baseurl.$path);
         return null;
      }

      return $response->getBody();
   }
   // ...
}
```

Dependencies are **pushed**.

```php
<?php
namespace Acme\FeedBundle\Service\FeedAg
use Guzzle\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

class FeedAggregator {
   private $client;
   private $logger;

   public function __construct(ClientInterface $client,
                               LoggerInterface $logger) {
      $this->client = $client;
      $this->logger = $logger;
   }

   public function retrieveFeed ($baseurl, $path) {
      $request = $this->client->setBaseUrl($baseurl)->get($path);
      $response = $request->send();
      if (200 != $response->getStatusCode()) {
         $this->logger->log('Could not get: '.$baseurl.$path);
         return null;
      }

      return $response->getBody();
   }
   // ...
}
```

```php
<?php
namespace Acme\FeedBundle\Service\FeedAg
use Guzzle\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

class FeedAggregator
    private $client
    private $logger

    public function __construct(ClientInterface $client,
                                LoggerInterface $logger) {
        $this->client = $client;
        $this->logger = $logger;
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $request->send();
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$baseurl.$path);
            return null;
        }

        return $response->getBody();
    }
    // ...
}
```

Dependencies are **pushed**.

Class only depends on **interfaces**

```
<?php
namespace Acme\FeedBundle\Service\FeedAg
use Guzzle\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

class FeedAggregator
    private $client
    private $logger

    public function __construct(ClientInterface $client,
                                LoggerInterface $logger) {
        $this->client = $client;
        $this->logger = $logger;
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $request->send();
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$baseurl.$path);
            return null;
        }

        return $response->getBody();
    }
    // ...
}
```

Dependencies are **pushed**.

Class only depends on **interfaces**

Implementations are **injected** at runtime

```php
<?php
namespace Acme\FeedBundle\Service\FeedAg
use Guzzle\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

class FeedAggregator
    private $client
    private $logger

    public function __construct(ClientInterface $client,
                                LoggerInterface $logger) {
        $this->client = $client;
        $this->logger = $logger;
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $request->send();
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$baseurl.$path);
            return null;
        }

        return $response->getBody();
    }
    // ...
}
```

Dependencies are **pushed**.

Class only depends on **interfaces**

Implementations are **injected** at runtime

Easy to **replace**, even **dynamically** (for testing)

```
<?php
namespace Acme\FeedBundle\Service\FeedAg
use Guzzle\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

class FeedAggregat
    private $client
    private $logger

    public function __construct(ClientInterface $client,
                                LoggerInterface $logger) {
        $this->client = $client;
        $this->logger = $logger;
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $r
        if (200 != $r                                   {
            $this->log                              '.$baseurl.$path);
            return nul
        }

        return $response->getBody();
    }
    // ...
}
```

Dependencies are **pushed**.

Class only depends on **interfaces**

Implementations are **injected** at runtime

Easy to **replace**, even **dynamically** (for testing)

On the level of the class, You are now experts for **Dependency Injection**.

```
<?php
namespace Acme\FeedBundle\Service\FeedAg
use Guzzle\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

class FeedAggregat
    private $client
    private $logger

    public function __construct(ClientInterface $client,
                                LoggerInterface $logger) {
        $this->client = $client;
        $this->logger = $logger;
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $
        if (200 != $re                             {
            $this->log                             '.$baseurl.$path);
            return nul
        }

        return $response->getBody();
    }
    // ...
}
```

Dependencies are **pushed**.

Class only depends on **interfaces**

Implementations are **injected** at runtime

Easy to **replace**, even **dynamically** (for testing)

On the level of the class, You are now experts for **Dependency Injection**.

Any questions?

```php
<?php
namespace Acme\FeedBundle\Service\FeedAg
use Guzzle\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

class FeedAggregator
    private $client
    private $logger

    public function __construct(ClientInterface $client,
                                LoggerInterface $logger) {
        $this->client = $client;
        $this->logger = $logger;
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $r
        if (200 != $re                               {
            $this->log                             '.$baseurl.$path);
            return nul
        }

        return $response->getBody();
    }
    // ...
}
```

Dependencies are **pushed**.

Class only depends on **interfaces**

Implementations are **injected** at runtime

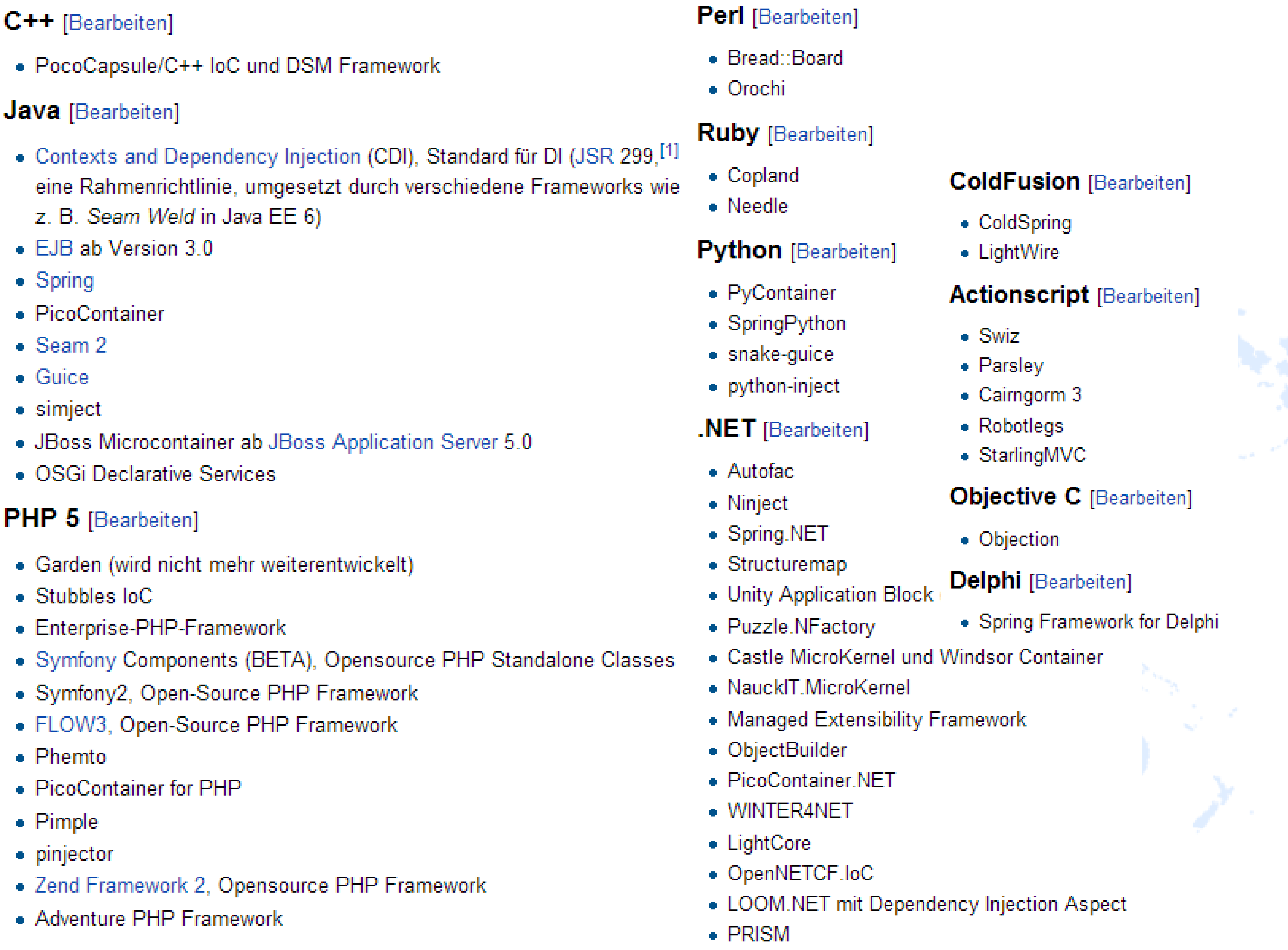
Easy to **replace**, even **dynamically** (for testing)

On the level of the class, You are now experts for **Dependency Injection**.

Who constructs and pushes all the dependencies?

Dependency Injection Container

“DI Container”, “DIC”, “Service Container”, “the Container”



C++ [Bearbeiten]

- PocoCapsule/C++ IoC und DSM Framework

Java [Bearbeiten]

- Contexts and Dependency Injection (CDI), Standard für DI (JSR 299,[1] eine Rahmenrichtlinie, umgesetzt durch verschiedene Frameworks wie z. B. *Seam Weld* in Java EE 6)
- EJB ab Version 3.0
- Spring
- PicoContainer
- Seam 2
- Guice
- simject
- JBoss Microcontainer ab JBoss Application Server 5.0
- OSGi Declarative Services

PHP 5 [Bearbeiten]

- Garden (wird nicht mehr weiterentwickelt)
- Stubbles IoC
- Enterprise-PHP-Framework
- Symfony Components (BETA), Opensource PHP Standalone Classes
- Symfony2, Open-Source PHP Framework
- FLOW3, Open-Source PHP Framework
- Phemto
- PicoContainer for PHP
- Pimple
- pinjector
- Zend Framework 2, Opensource PHP Framework
- Adventure PHP Framework

Perl [Bearbeiten]

- Bread::Board
- Orochi

Ruby [Bearbeiten]

- Copland
- Needle

Python [Bearbeiten]

- PyContainer
- SpringPython
- snake-guice
- python-inject

.NET [Bearbeiten]

- Autofac
- Ninject
- Spring.NET
- Structuremap
- Unity Application Block
- Puzzle.NFactory
- Castle MicroKernel und Windsor Container
- NauckIT.MicroKernel
- Managed Extensibility Framework
- ObjectBuilder
- PicoContainer.NET
- WINTER4NET
- LightCore
- OpenNETCF.IoC
- LOOM.NET mit Dependency Injection Aspect
- PRISM

ColdFusion [Bearbeiten]

- ColdSpring
- LightWire

Actionscript [Bearbeiten]

- Swiz
- Parsley
- Cairngorm 3
- Robotlegs
- StarlingMVC

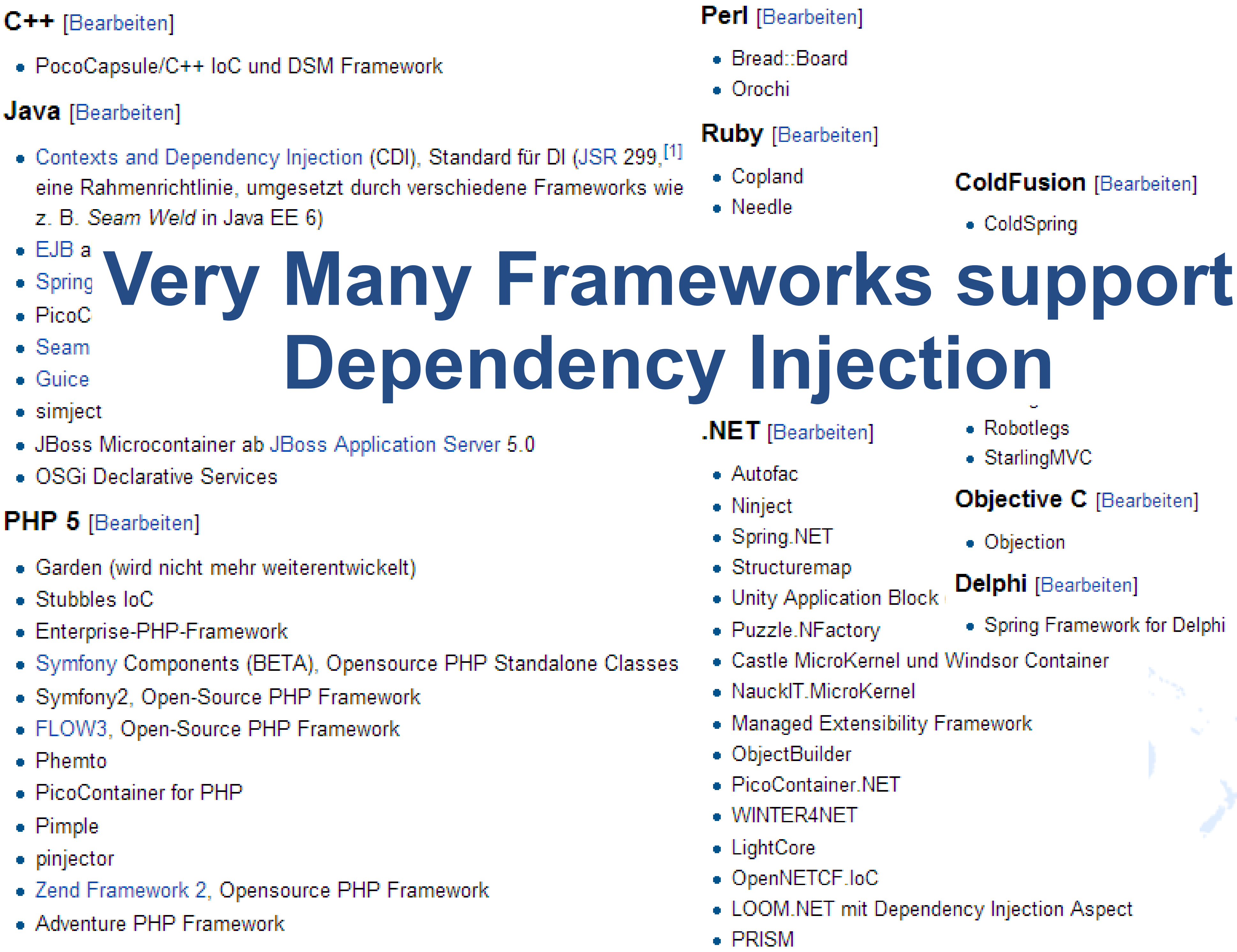
Objective C [Bearbeiten]

- Objection

Delphi [Bearbeiten]

- Spring Framework for Delphi

Very Many Frameworks support Dependency Injection



C++ [Bearbeiten]

- PocoCapsule/C++ IoC und DSM Framework

Java [Bearbeiten]

- Contexts and Dependency Injection (CDI), Standard für DI (JSR 299,[1] eine Rahmenrichtlinie, umgesetzt durch verschiedene Frameworks wie z. B. *Seam Weld* in Java EE 6)
- EJB a
- Spring
- PicoC
- Seam
- Guice
- simject
- JBoss Microcontainer ab JBoss Application Server 5.0
- OSGi Declarative Services

PHP 5 [Bearbeiten]

- Garden (wird nicht mehr weiterentwickelt)
- Stubbles IoC
- Enterprise-PHP-Framework
- Symfony Components (BETA), Opensource PHP Standalone Classes
- Symfony2, Open-Source PHP Framework
- FLOW3, Open-Source PHP Framework
- Phemto
- PicoContainer for PHP
- Pimple
- pinjector
- Zend Framework 2, Opensource PHP Framework
- Adventure PHP Framework

Perl [Bearbeiten]

- Bread::Board
- Orochi

Ruby [Bearbeiten]

- Copland
- Needle

.NET [Bearbeiten]

- Autofac
- Ninject
- Spring.NET
- Structuremap
- Unity Application Block (
- Puzzle.NFactory
- Castle MicroKernel und Windsor Container
- NauckIT.MicroKernel
- Managed Extensibility Framework
- ObjectBuilder
- PicoContainer.NET
- WINTER4NET
- LightCore
- OpenNETCF.IoC
- LOOM.NET mit Dependency Injection Aspect
- PRISM

ColdFusion [Bearbeiten]

- ColdSpring

- Robotlegs
- StarlingMVC

Objective C [Bearbeiten]

- Objection

Delphi [Bearbeiten]

- Spring Framework for Delphi

DI is very easy in Symfony 2

DI is very easy in Symfony 2

- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

DI is very easy in Symfony 2

- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

```
# app/config/config.yml
# ...
services:
    my_service:
        class:    Acme\MyBundle\Service\AwesomeClass
```

DI is very easy in Symfony 2

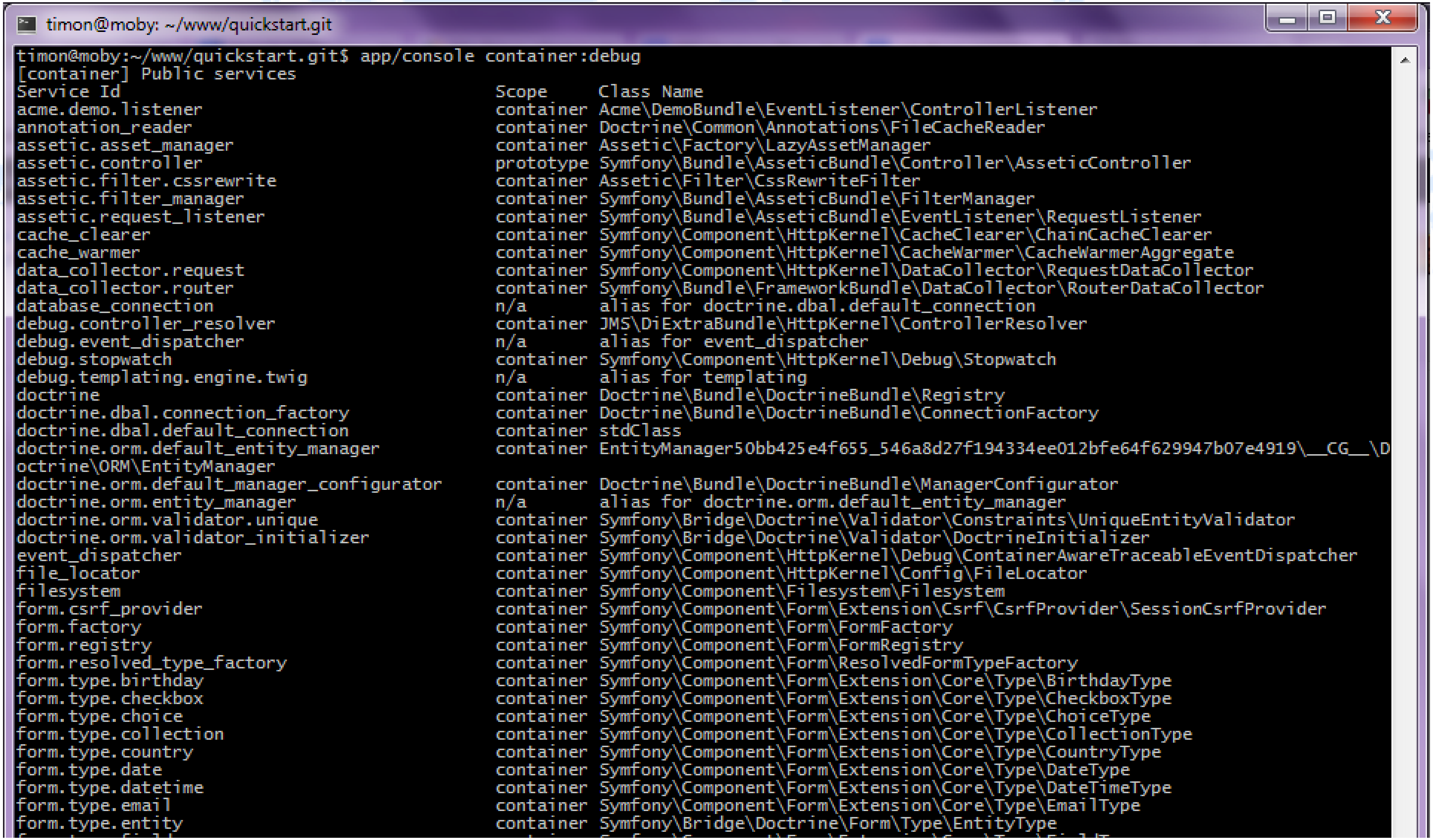
- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

```
# app/config/config.yml
# ...
services:
    my_service:
        class:    Acme\MyBundle\Service\AwesomeClass
```

Class to manage

Name of the new service

# php app/console container:debug



```
timon@moby:~/www/quickstart.git$ app/console container:debug
[container] Public services
Service Id                                     Scope     Class Name
acme.demo.listener                             container Acme\DemoBundle\EventListener\ControllerListener
annotation_reader                              container Doctrine\Common\Annotations\FileCacheReader
assetic.asset_manager                          container Assetic\Factory\LazyAssetManager
assetic.controller                             prototype Symfony\Bundle\AsseticBundle\Controller\AsseticController
assetic.filter.cssrewrite                      container Assetic\Filter\CssRewriteFilter
assetic.filter_manager                         container Symfony\Bundle\AsseticBundle\FilterManager
assetic.request_listener                       container Symfony\Bundle\AsseticBundle\EventListener\RequestListener
cache_clearer                                  container Symfony\Component\HttpKernel\CacheClearer\ChainCacheClearer
cache_warmer                                   container Symfony\Component\HttpKernel\CacheWarmer\CacheWarmerAggregate
data_collector.request                         container Symfony\Component\HttpKernel\DataCollector\RequestDataCollector
data_collector.router                          container Symfony\Bundle\FrameworkBundle\DataCollector\RouterDataCollector
database_connection                            n/a       alias for doctrine.dbal.default_connection
debug.controller_resolver                      container JMS\DiExtraBundle\HttpKernel\ControllerResolver
debug.event_dispatcher                         n/a       alias for event_dispatcher
debug.stopwatch                                container Symfony\Component\HttpKernel\Debug\Stopwatch
debug.templating.engine.twig                   n/a       alias for templating
doctrine                                       container Doctrine\Bundle\DoctrineBundle\Registry
doctrine.dbal.connection_factory               container Doctrine\Bundle\DoctrineBundle\ConnectionFactory
doctrine.dbal.default_connection               container stdClass
doctrine.orm.default_entity_manager            container EntityManager50bb425e4f655_546a8d27f194334ee012bfe64f629947b07e4919\__CG__\D
octrine\ORM\EntityManager
doctrine.orm.default_manager_configurator      container Doctrine\Bundle\DoctrineBundle\ManagerConfigurator
doctrine.orm.entity_manager                    n/a       alias for doctrine.orm.default_entity_manager
doctrine.orm.validator.unique                  container Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntityValidator
doctrine.orm.validator_initializer             container Symfony\Bridge\Doctrine\Validator\DoctrineInitializer
event_dispatcher                               container Symfony\Component\HttpKernel\Debug\ContainerAwareTraceableEventDispatcher
file_locator                                   container Symfony\Component\HttpKernel\Config\FileLocator
filesystem                                     container Symfony\Component\Filesystem\Filesystem
form.csrf_provider                             container Symfony\Component\Form\Extension\Csrf\CsrfProvider\SessionCsrfProvider
form.factory                                   container Symfony\Component\Form\FormFactory
form.registry                                  container Symfony\Component\Form\FormRegistry
form.resolved_type_factory                     container Symfony\Component\Form\ResolvedFormTypeFactory
form.type.birthday                             container Symfony\Component\Form\Extension\Core\Type\BirthdayType
form.type.checkbox                             container Symfony\Component\Form\Extension\Core\Type\CheckboxType
form.type.choice                               container Symfony\Component\Form\Extension\Core\Type\ChoiceType
form.type.collection                           container Symfony\Component\Form\Extension\Core\Type\CollectionType
form.type.country                              container Symfony\Component\Form\Extension\Core\Type\CountryType
form.type.date                                 container Symfony\Component\Form\Extension\Core\Type\DateType
form.type.datetime                             container Symfony\Component\Form\Extension\Core\Type\DateTimeType
form.type.email                                container Symfony\Component\Form\Extension\Core\Type\EmailType
form.type.entity                               container Symfony\Bridge\Doctrine\Form\Type\EntityType
```

DI is very easy in Symfony 2

- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

```
# app/config/config.yml
# ...
services:
    my_service:
        class:    Acme\MyBundle\Service\AwesomeClass
```

DI is very easy in Symfony 2

- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

```
# app/config/config.yml
# ...
services:
    my_service:
        class:    Acme\MyBundle\Service\AwesomeClass
        arguments:
            some_arg:    “string”
            another:
                - array_member
                - array_member
            even_more:   @another_service
```

DI is very easy in Symfony 2

- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

```
# app/config/config.yml
# ...
services:
    my_service:
        class:     Acme\                        meClass
        arguments:
            some_arg:      "string"
            another:
                - array_member
                - array_member
            even_more:     @another_service
```

Arguments can be strings, numbers, arrays, placeholders, and many more ...

Any other service can be injected as as argument

```php
<?php
use   Guzzle\Http\ClientInterface;
use   Acme\Logger\LoggerInterface;

class FeedAggregator {
   private $client;
   private $logger;

   public function __construct(ClientInterface $client,
                               LoggerInterface $logger) {
      $this->client = $client;
      $this->logger = $logger;
   }

   public function retrieveFeed ($baseurl, $path) {
      $request = $this->client->setBaseUrl($baseurl)->get($path);
      $response = $request->send();
      if (200 != $response->getStatusCode()) {
         $this->logger->log('Could not get: '.$host.$path);
         return null;
      }

      return $response->getBody();
   }
   // ...
}
```

```php
<?php
use   Guzzle\Http\ClientInterface;
use   Acme\Logger\LoggerInterface;

class FeedAggregator {
   private $client;
   private $logger;

   public function __construct(ClientInterface $client,
                               LoggerInterface $logger) {
      $this->client = $client;
      $this->logger = $logger;
   }

   public function retrieveFeed ($baseurl, $path) {
      $request = $this->client->setBaseUrl($baseurl)->get($path);
      $response = $request->send();
      if (200 != $response->getStatusCode()) {
         $this->logger->log('Could not get: '.$host.$path);
```

```
# app/config/config.yml
# ...
services:
    feed_aggregator:
        class:    Acme\FeedBundle\Service\FeedAggregator
        arguments:
            client:        @http_client
            logger:        @logger
```

```
<?php
use   Guzzle\Http\ClientInterface;
use   Acme\Logger\LoggerInterface;

class FeedAggregator {
   private $client;
   private $logger;

   public function __construct(ClientInterface $client,
                               LoggerInterface $logger) {
      $this->client = $client;
      $this->logger = $logger;
   }

   public function retrieveFeed ($baseurl, $path) {
      $request = $this->client->setBaseUrl($baseurl)->get($path);
      $response = $request->send();
      if (200 != $response->getStatusCode()) {
         $this->logger->log('Could not get: '.$host.$path);
```

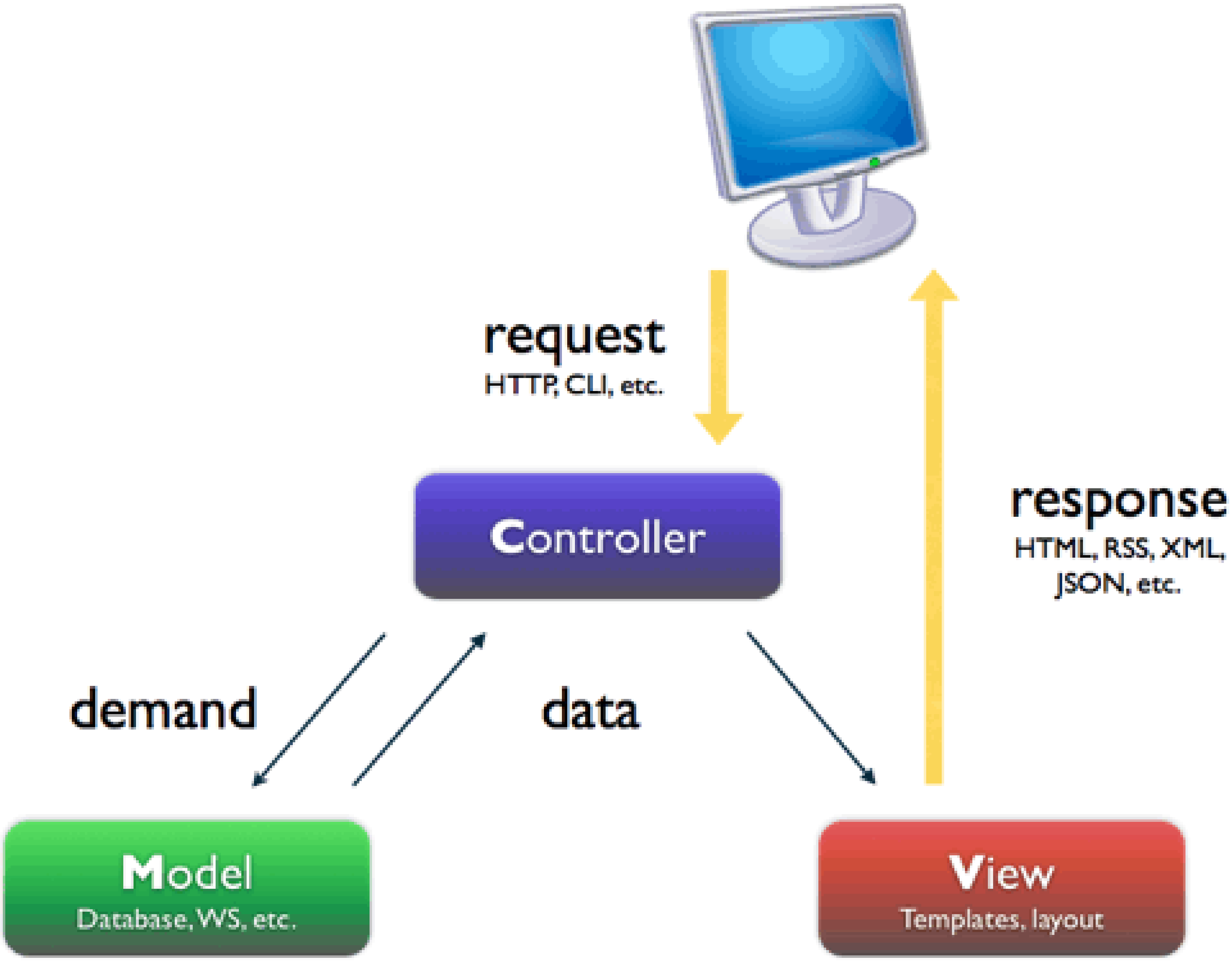
```
# app/config/config.yml
# ...
services:
    feed_aggregator:
        class:    Acme\FeedBundle\Service\FeedAggregator
        arguments:
            client:        @http_client
            logger:        @logger
```

```
<?php
use   Guzzle\Http\ClientInterface;
use   Acme\Logger\LoggerInterface;

class FeedAggregator {
   private $client;
   private $logger;

   public function __construct(ClientInterface $client,
                               LoggerInterface $logger) {
      $this->client = $client;
      $this->logger = $logger;
   }

   public function retrieveFeed ($baseurl, $path) {
      $request = $this->client->setBaseUrl($baseurl)->get($path);
      $response = $request->send();
      if (200 != $response->getStatusCode()) {
         $this->logger->log('Could not get: ' . $host . $path);
```

Different Config needed for Functional Testing?

- app/config/config.yml
- app/config/config_dev.yml
- app/config/config_test.yml

Put it here here



request

HTTP, CLI, etc.

response

HTML, RSS, XML, JSON, etc.

Controller

demand

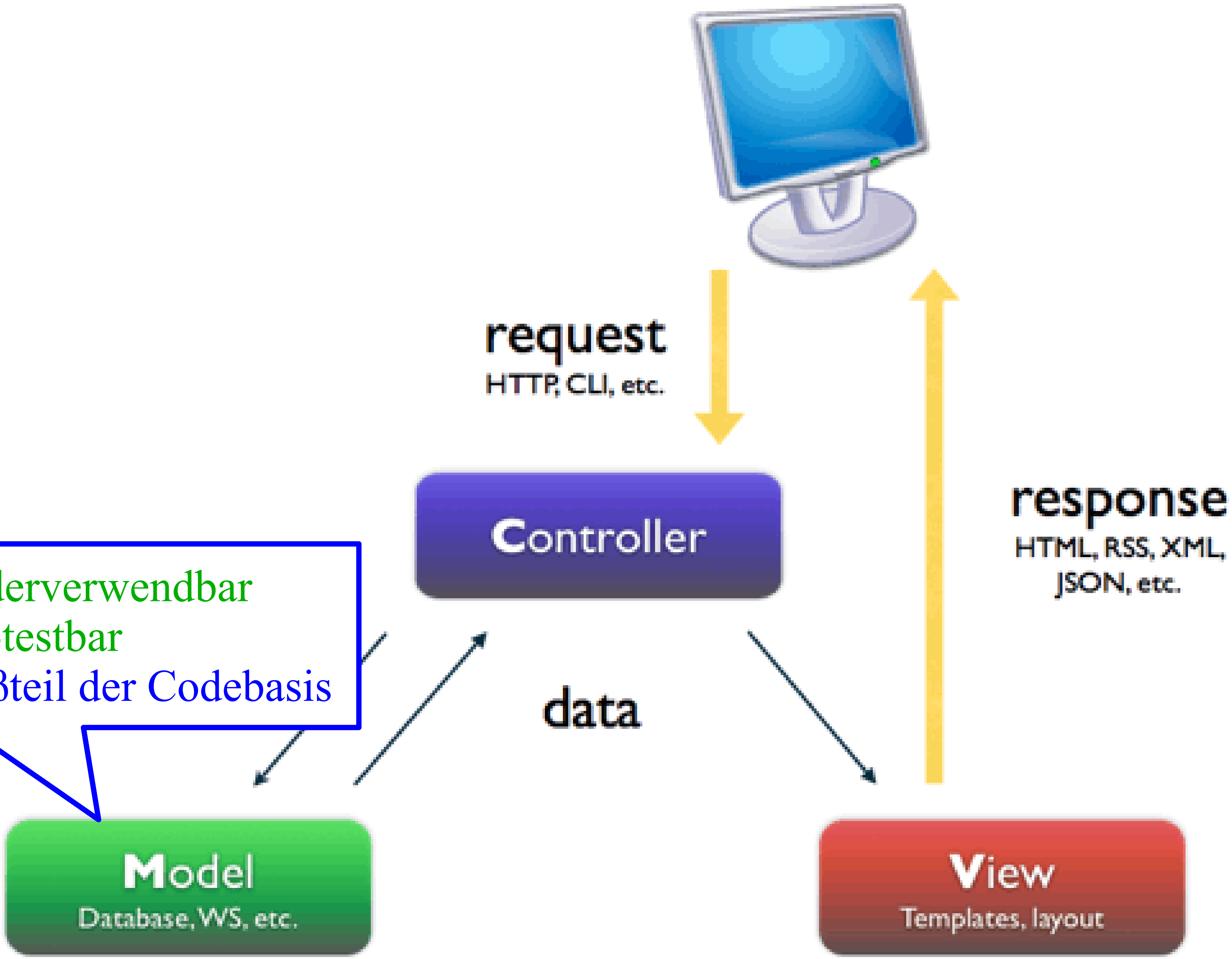
data

Model

Database, WS, etc.

View

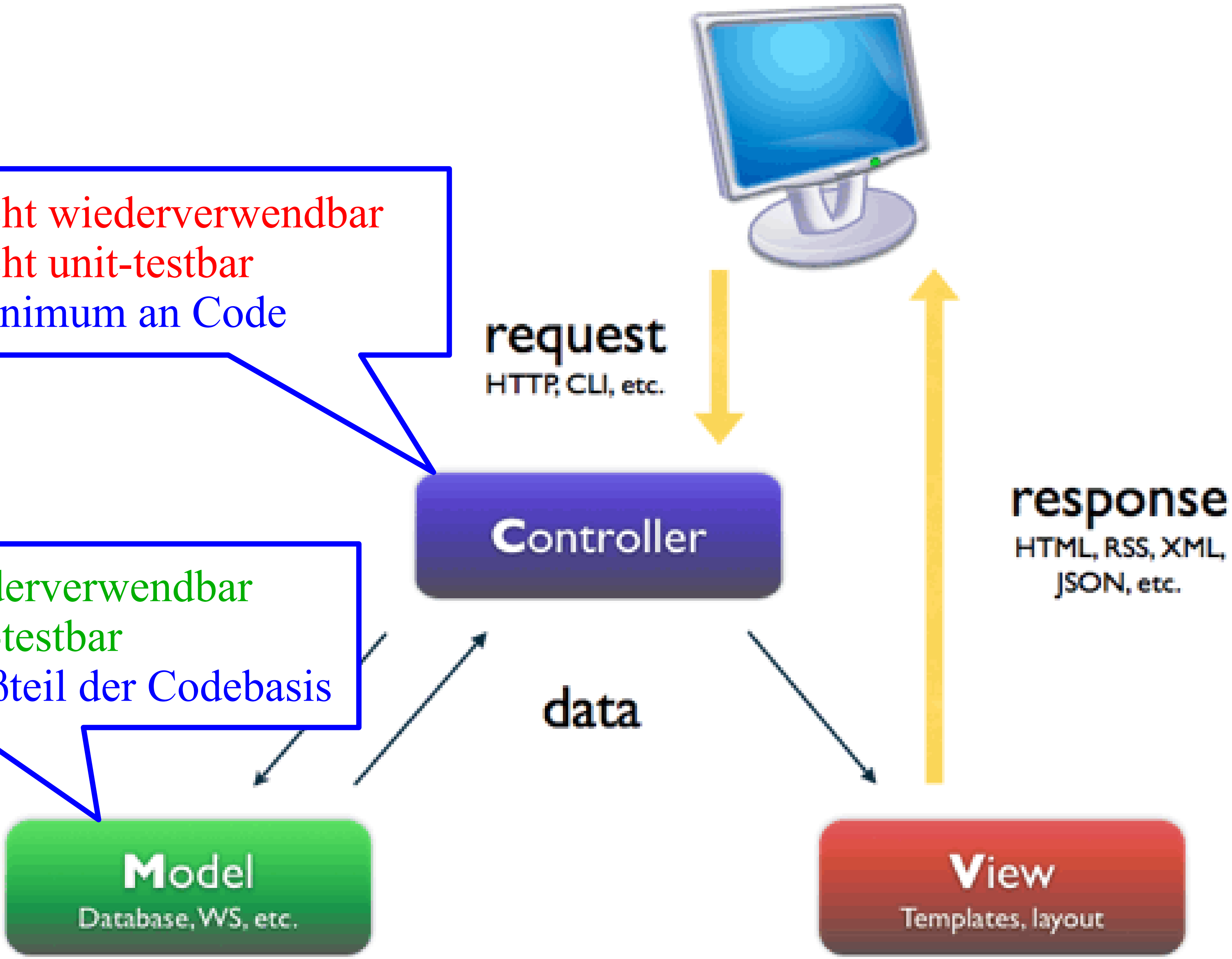
Templates, layout

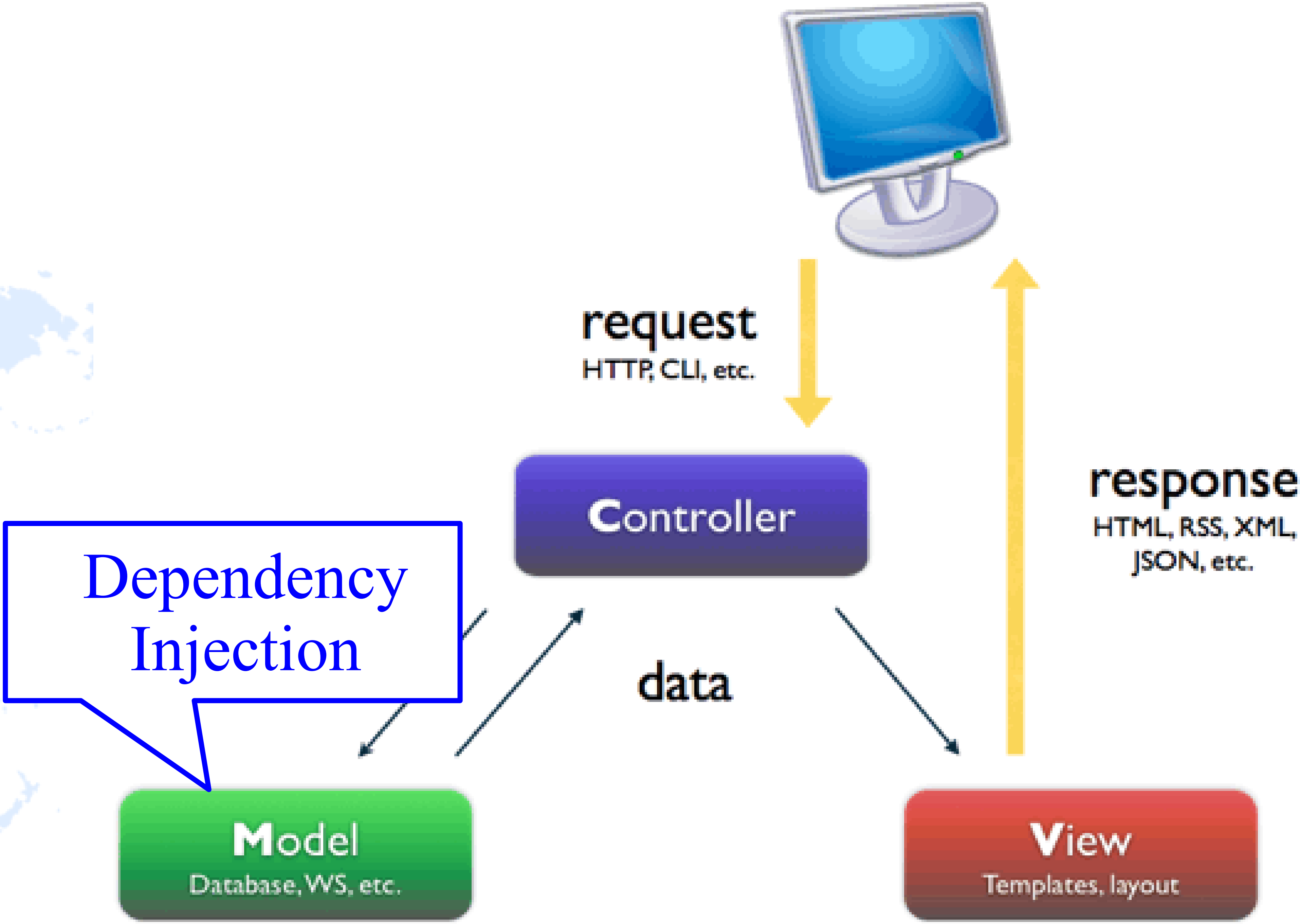


wiederverwendbar
unit-testbar
Großteil der Codebasis

nicht wiederverwendbar
nicht unit-testbar
Minimum an Code

wiederverwendbar
unit-testbar
Großteil der Codebasis





request
HTTP, CLI, etc.

Controller

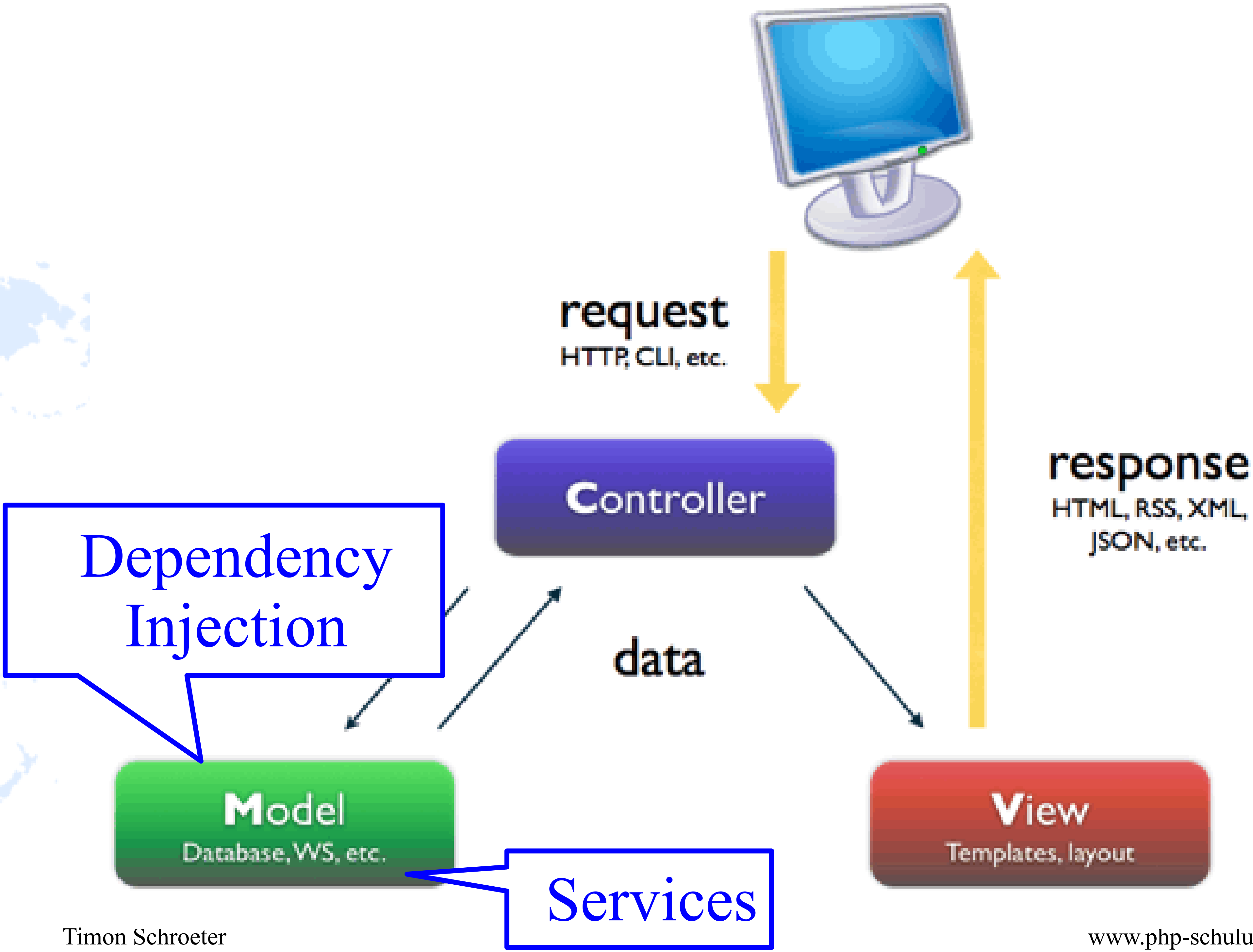
response
HTML, RSS, XML, JSON, etc.

Dependency Injection

data

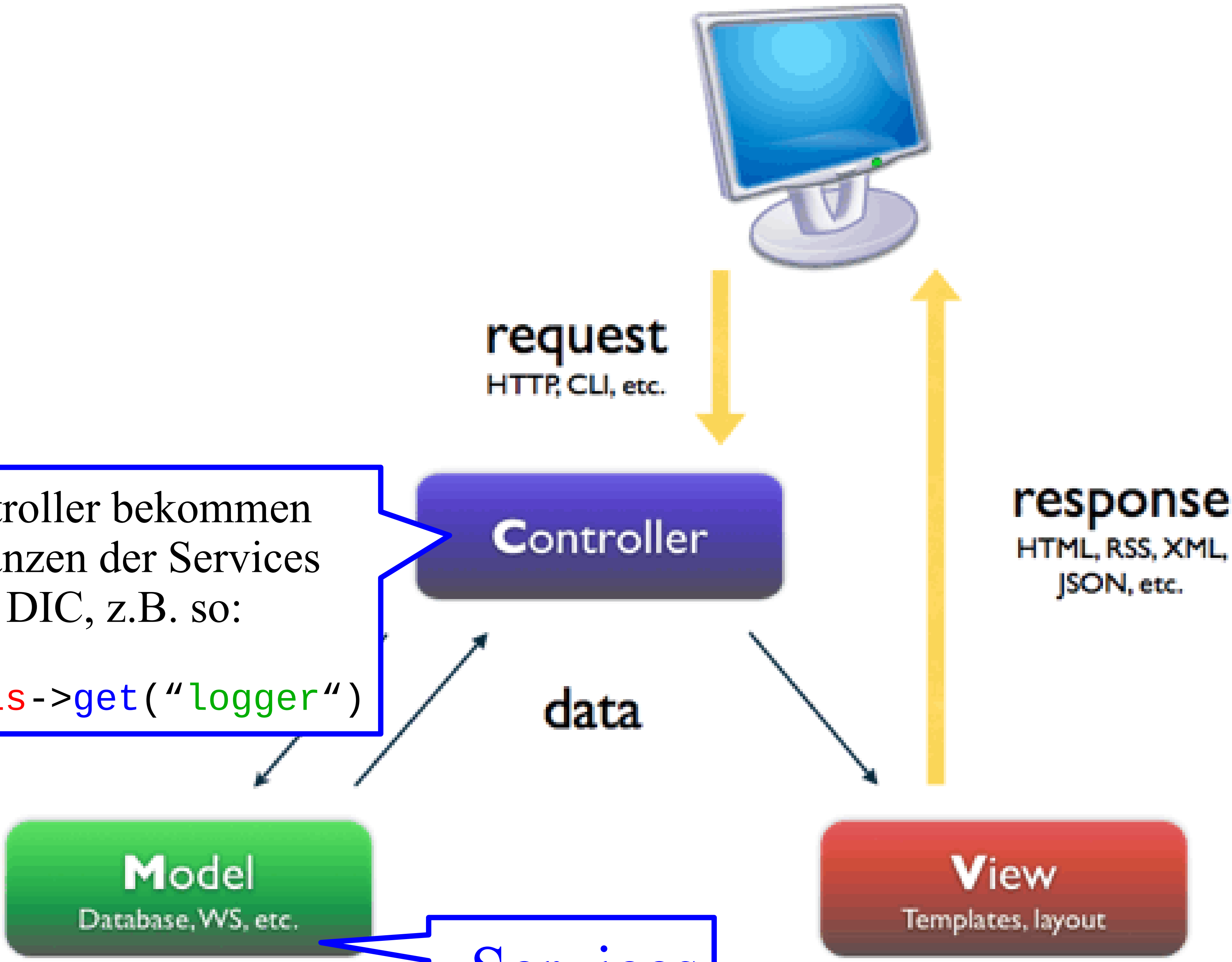
Model
Database, WS, etc.

View
Templates, layout



Dependency Injection

Services



Controller bekommen Instanzen der Services vom DIC, z.B. so:

```
$this->get(“logger“)
```

Services

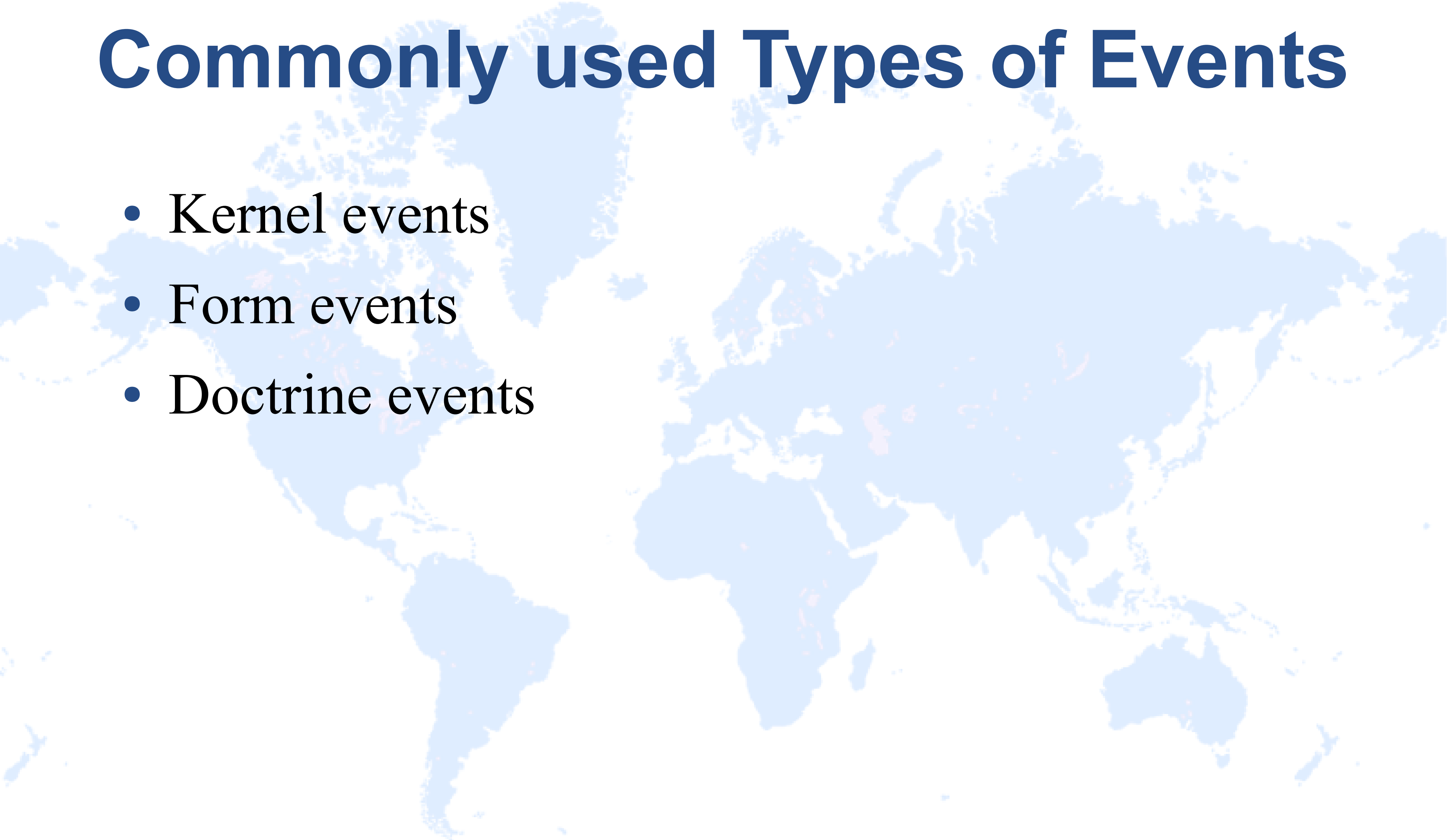
Nachhaltige Architektur für große Web-Projekte

- ★ Modularität
- ★ Erweiterbarkeit
- ★ Wartbarkeit
- ★ Testbarkeit
- Performance

Further Reading

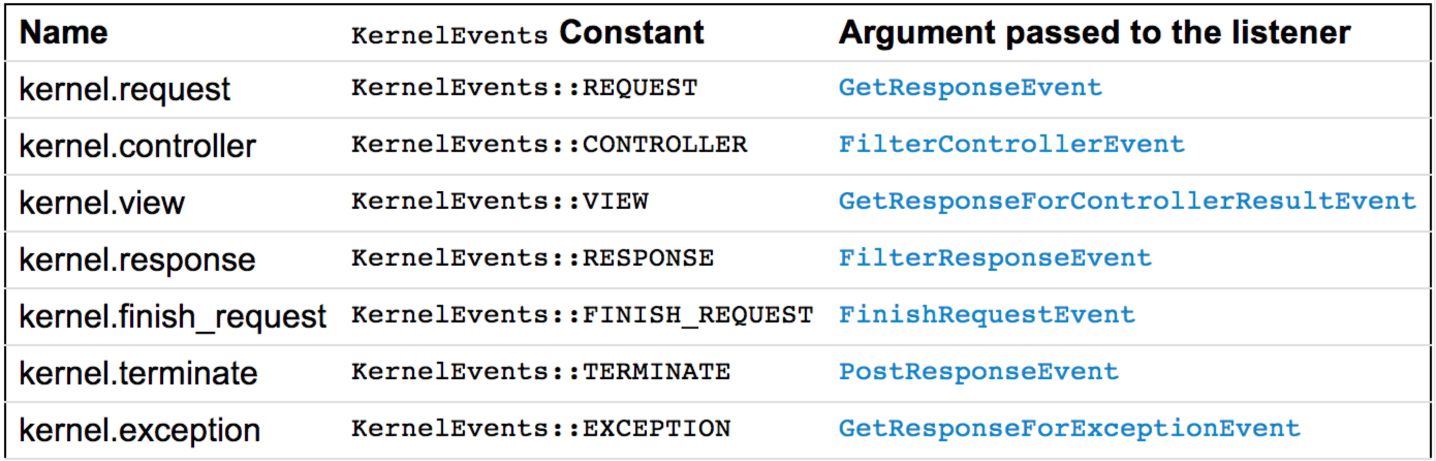
- http://fabien.potencier.org/article/11/what-is-dependency-injection
- http://symfony.com/doc/current/components/dependency_injection/compilation.html
- http://symfony.com/doc/current/cookbook/service_container/compiler_passes.html
- Use the source ...

Commonly used Types of Events



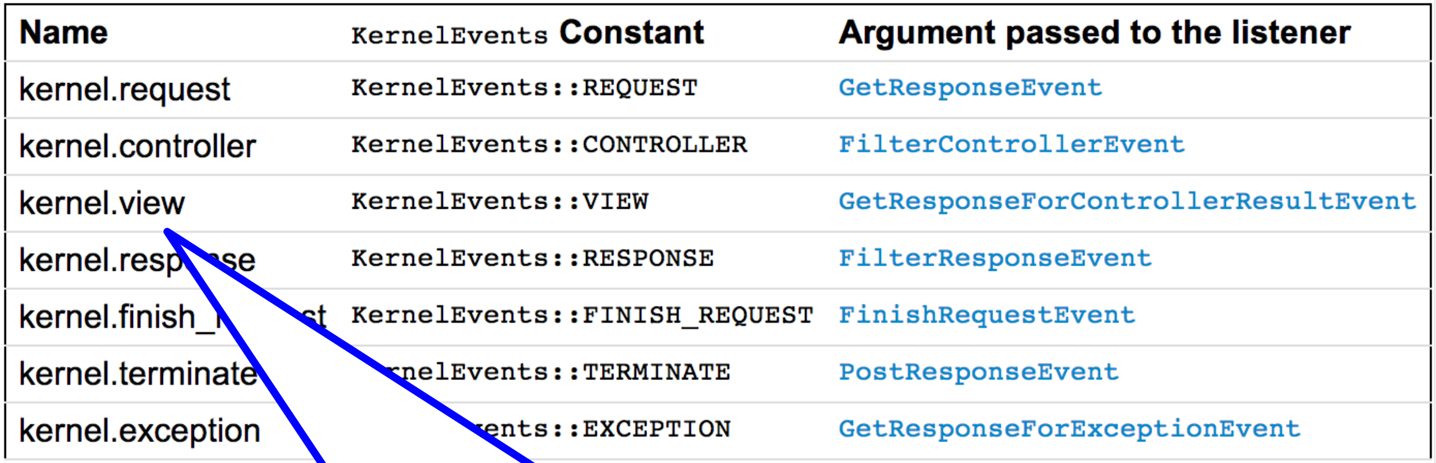
- Kernel events
- Form events
- Doctrine events

Kernel Events



Name	KernelEvents Constant	Argument passed to the listener
kernel.request	KernelEvents::REQUEST	GetResponseEvent
kernel.controller	KernelEvents::CONTROLLER	FilterControllerEvent
kernel.view	KernelEvents::VIEW	GetResponseForControllerResultEvent
kernel.response	KernelEvents::RESPONSE	FilterResponseEvent
kernel.finish_request	KernelEvents::FINISH_REQUEST	FinishRequestEvent
kernel.terminate	KernelEvents::TERMINATE	PostResponseEvent
kernel.exception	KernelEvents::EXCEPTION	GetResponseForExceptionEvent

Kernel Events



Name	`KernelEvents` Constant	Argument passed to the listener
kernel.request	`KernelEvents::REQUEST`	`GetResponseEvent`
kernel.controller	`KernelEvents::CONTROLLER`	`FilterControllerEvent`
kernel.view	`KernelEvents::VIEW`	`GetResponseForControllerResultEvent`
kernel.response	`KernelEvents::RESPONSE`	`FilterResponseEvent`
kernel.finish_request	`KernelEvents::FINISH_REQUEST`	`FinishRequestEvent`
kernel.terminate	`KernelEvents::TERMINATE`	`PostResponseEvent`
kernel.exception	`KernelEvents::EXCEPTION`	`GetResponseForExceptionEvent`

Used in SensioFrameworkExtraBundle to act on the @Template annotation.

To act on a Kernel Event

- Create a Listener Class
- Let the DIC do all remaining work for you

To act on a Kernel Event

- Create a Listener Class
- Let the DIC do all remaining work for you
  - Create a service definition

To act on a Kernel Event

```
# app/config/config.yml
services:
    your_listener_name:
        class: Acme\DemoBundle\EventListener\AcmeExceptionListener
```

- Create a Listener Class
- Let the DIC do all remaining work for you
  - Create a service definition

To act on a Kernel Event

```
# app/config/config.yml
services:
    your_listener_name:
        class: Acme\DemoBundle\EventListener\AcmeExceptionListener
```

- Create a Listener Class
- Let the DIC do all remaining work for you
  - Create a service definition
  - Add a tag labeling the service as a listener

To act on a Kernel Event

```
# app/config/config.yml
services:
   your_listener_name:
      class: Acme\DemoBundle\EventListener\AcmeExceptionListener
      tags:
         - { name: kernel.event_listener, event: kernel.exception, method: onKernelException }
```

- Create a Listener Class
- Let the DIC do all remaining work for you
  - Create a service definition
  - Add a tag labeling the service as a listener

How to listen to

- Kernel events
  - listener class + service tag
- Form events
- Doctrine events

How to listen to

- Kernel events
  - listener class + service tag
- Form events
  - listener class + service tag
  - closure inside form type class
- Doctrine events

How to listen to

- Kernel events
  - listener class + service tag
- Form events
  - listener class + service tag
  - closure inside form type class
- Doctrine events
  - listener class + service tag
  - callback method inside entity class

Time Checkpoint

- Skip?

Which listeners are active?

- Your answer?

Which listeners are active?

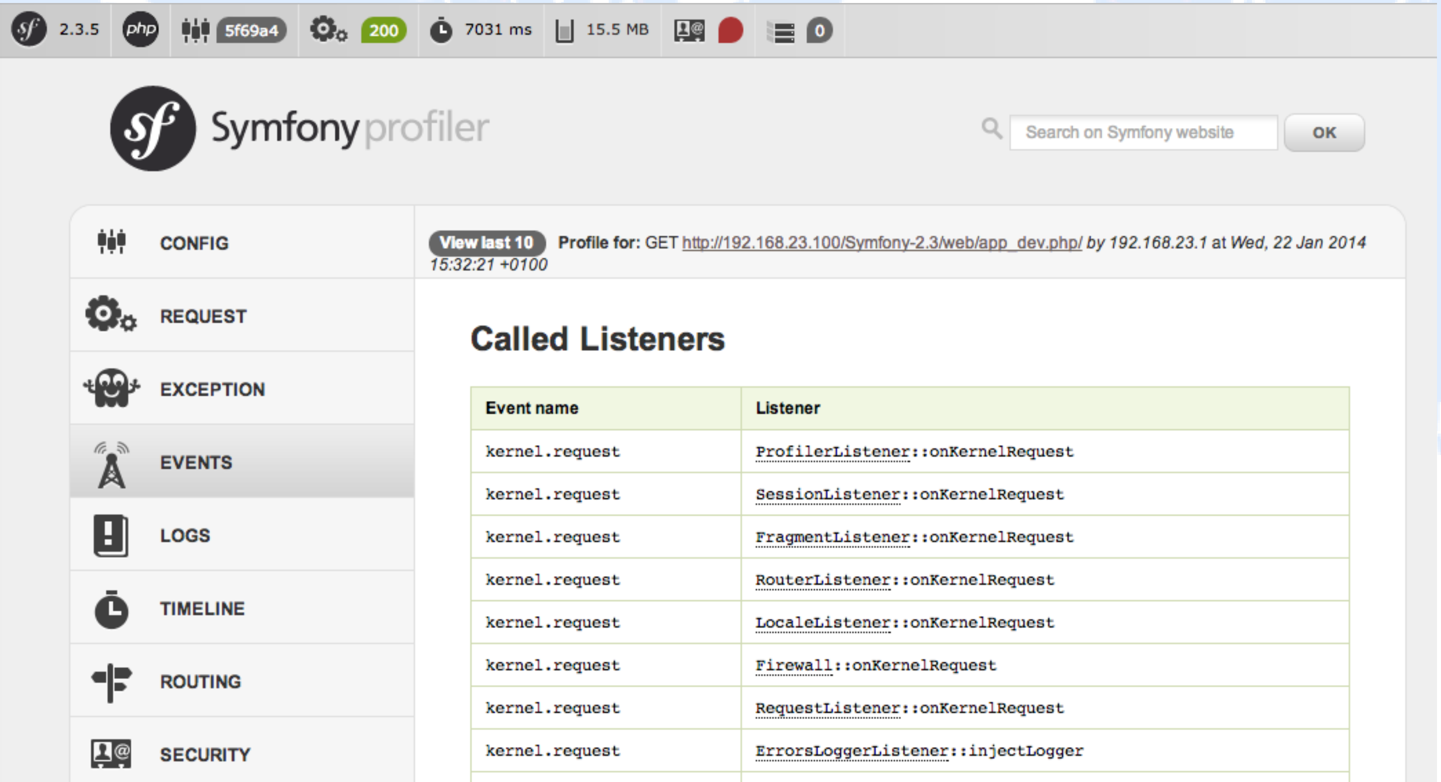
- Kernel events
  - listener class + service tag
- Form events
  - listener class + service tag
  - closure inside form type class
- Doctrine events
  - listener class + service tag
  - callback method inside entity class
- Your own custom events
  - listener class + service tag
  - closure etc.

Which listeners are active?

- Symfony profiler > Events

Which listeners are active?

- Symfony profiler > Events



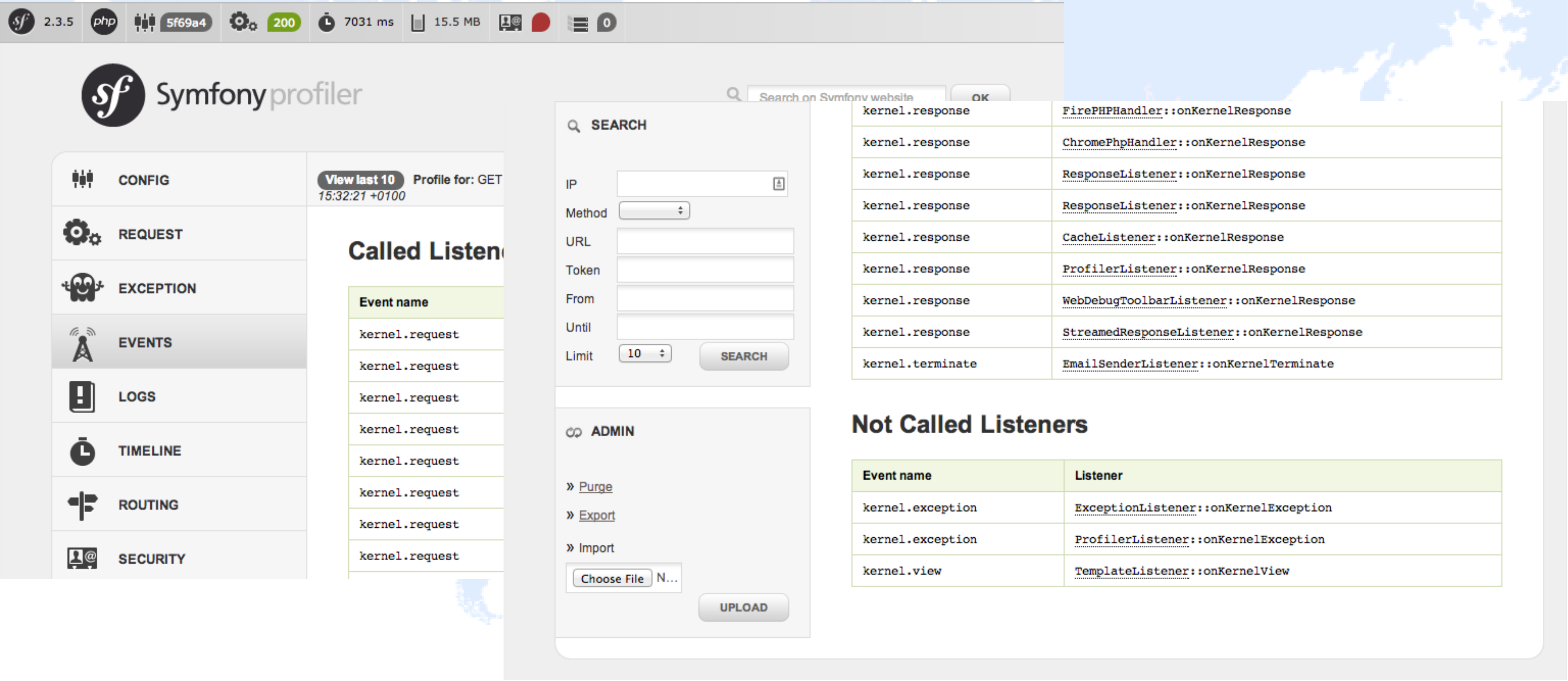
Profile for: GET http://192.168.23.100/Symfony-2.3/web/app_dev.php/ *by 192.168.23.1* at *Wed, 22 Jan 2014 15:32:21 +0100*

Called Listeners

Event name	Listener
kernel.request	ProfilerListener::onKernelRequest
kernel.request	SessionListener::onKernelRequest
kernel.request	FragmentListener::onKernelRequest
kernel.request	RouterListener::onKernelRequest
kernel.request	LocaleListener::onKernelRequest
kernel.request	Firewall::onKernelRequest
kernel.request	RequestListener::onKernelRequest
kernel.request	ErrorsLoggerListener::injectLogger

Which listeners are active?

- Symfony profiler > Events



Profile for: GET
15:32:21 +0100

Called Listen

Event name
kernel.request
kernel.request
kernel.request
kernel.request
kernel.request
kernel.request
kernel.request
kernel.request

kernel.response	FirePHPHandler::onKernelResponse
kernel.response	ChromePhpHandler::onKernelResponse
kernel.response	ResponseListener::onKernelResponse
kernel.response	ResponseListener::onKernelResponse
kernel.response	CacheListener::onKernelResponse
kernel.response	ProfilerListener::onKernelResponse
kernel.response	WebDebugToolbarListener::onKernelResponse
kernel.response	StreamedResponseListener::onKernelResponse
kernel.terminate	EmailSenderListener::onKernelTerminate

Not Called Listeners

Event name	Listener
kernel.exception	ExceptionListener::onKernelException
kernel.exception	ProfilerListener::onKernelException
kernel.view	TemplateListener::onKernelView

Which listeners are active?

- Symfony profiler > Events
- Check in your own Command oder Controller

Which listeners are active?

- Symfony profiler > Events
- Check in your own Command oder Controller

```
$listeners = $this->getContainer()->get('event_dispatcher')->getListeners();
```

Which listeners are active?

- Symfony profiler > Events
- Check in your own Command oder Controller

```
$listeners = $this->getContainer()->get('event_dispatcher')->getListeners();

print_r (array_keys($listeners));
print_r (array_keys($listeners['kernel.view']));
```

Which listeners are active?

- Symfony profiler > Events
- Check in your own Command oder Controller

```
$listeners = $this->getContainer()->get('event_dispatcher')->getListeners();

print_r (array_keys($listeners));
print_r (array_keys($listeners['kernel.view']));

$entityManager = $this->getContainer()->get('doctrine.orm.entity_manager');
```

Doctrine Entity Manager uses an instance of the Doctrine Event Manager (injected by the DIC).

Which listeners are active?

- Symfony profiler > Events
- Check in your own Command oder Controller

```
$listeners = $this->getContainer()->get('event_dispatcher')->getListeners();

print_r (array_keys($listeners));
print_r (array_keys($listeners['kernel.view']));

$entityManager = $this->getContainer()->get('doctrine.orm.entity_manager');
$listeners = $entityManager->getEventManager()->getListeners();

print_r (array_keys($listeners));
// ...
```

Which listeners are active?

- Symfony profiler > Events
- Check in your own Command oder Controller
- ListenersDebugCommandBundle

ListenersDebugCommandBundle

- https://github.com/egulias/ListenersDebugCommandBundle
- app/console container:debug:listeners

  --event=event.name: if issued will filter to show only the listeners listening to the given name (ordered by descending priority, unless you use: --order-asc)

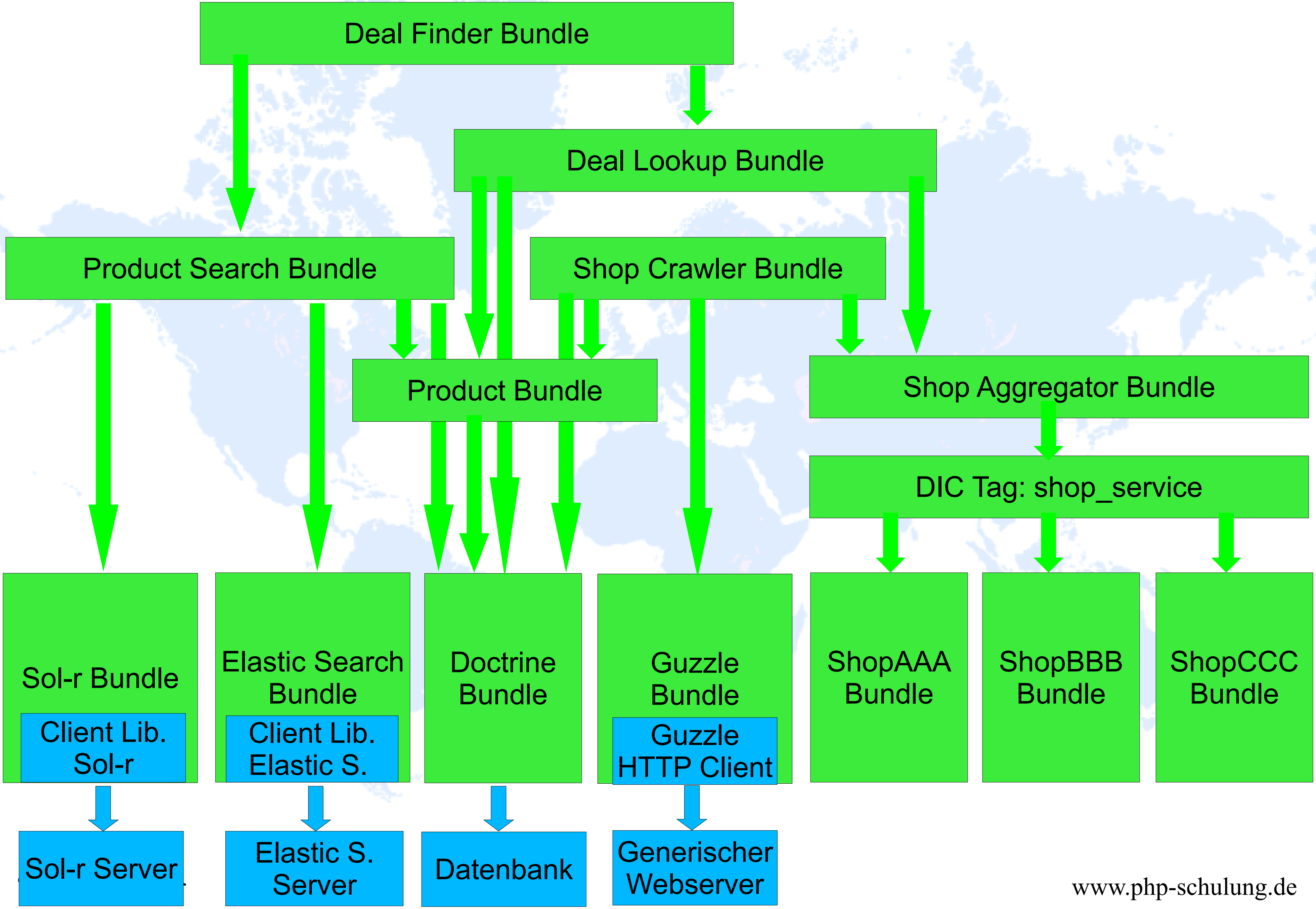
  --show-private : if issued will show also private services

  ...

Nachhaltige Architektur für große Web-Projekte

- ★ Modularität
- ★ Erweiterbarkeit
- ★ Wartbarkeit
- ★ Testbarkeit
- ➡ Performance

Gute Performance machbar?



Deal Finder Bundle

Deal Lookup Bundle

Product Search Bundle

Shop Crawler Bundle

Product Bundle

Shop Aggregator Bundle

DIC Tag: shop_service

Sol-r Bundle

Client Lib. Sol-r

Elastic Search Bundle

Client Lib. Elastic S.

Doctrine Bundle

Guzzle Bundle

Guzzle HTTP Client

ShopAAA Bundle

ShopBBB Bundle

ShopCCC Bundle

Sol-r Server

Elastic S. Server

Datenbank

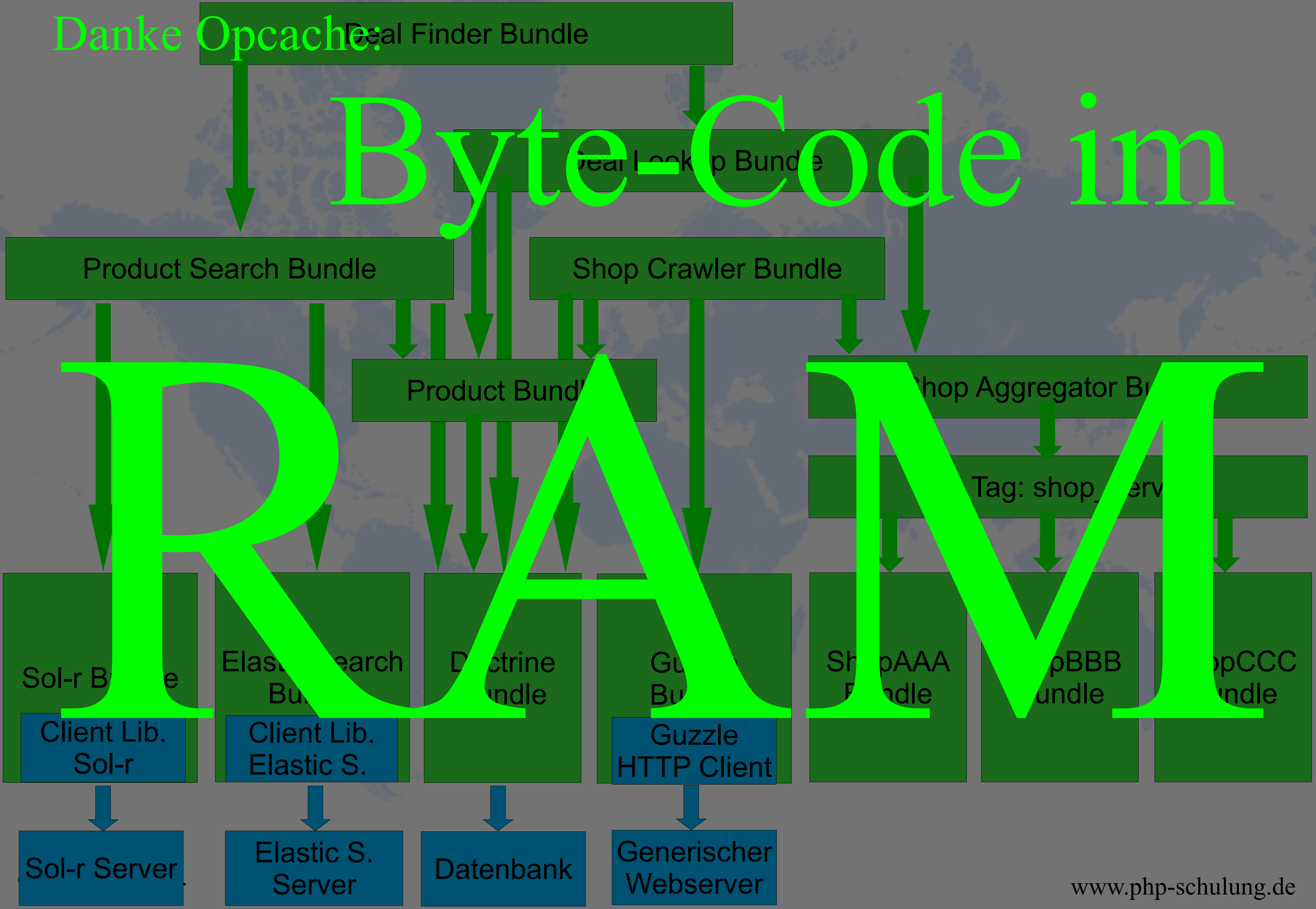
Generischer Webserver

Gute Performance machbar !

Danke Opcache:

Byte-Code im

RAM



www.php-schulung.de

Nachhaltige Architektur für große Web-Projekte

- ★ Modularität
- ★ Erweiterbarkeit
- ★ Wartbarkeit
- ★ Testbarkeit
- ★ Performance

Further Reading: Dependency Injection

- http://fabien.potencier.org/article/11/what-is-dependency-injection
- http://symfony.com/doc/current/components/dependency_injection/compilation.html
- http://symfony.com/doc/current/cookbook/service_container/compiler_passes.html
- Use the source ...

Further Reading: Event Dispatcher

- Symfony Event Dispatcher
  - http://symfony.com/doc/current/components/event_dispatcher/
  - http://symfony.com/doc/current/cookbook/event_dispatcher/
  - http://symfony.com/doc/current/cookbook/doctrine/event_listeners_subscribers.html
  - http://api.symfony.com/2.4/Symfony/Component/EventDispatcher/EventDispatcher.html

- Doctrine Event Manager
  - http://docs.doctrine-project.org/projects/doctrine1/en/latest/en/manual/event-listeners.html
  - http://www.whitewashing.de/2013/07/24/doctrine_and_domainevents.html
  - http://www.doctrine-project.org/api/orm/2.4/class-Doctrine.ORM.EntityManager.html
  - http://www.doctrine-project.org/api/common/2.4/class-Doctrine.Common.EventManager.h

Vielen Dank

für Eure Aufmerksamkeit!

Fragen?

Ideen, Wünsche, Anmerkungen?