# Dependency Injection

# DI

Timon Schroeter

@PHP_Entwickler

*PHP & Symfony Schulungen in Berlin, Nürnberg, München*

Berlin · http://www.php-entwickler-berlin.de/

61 TWEETS

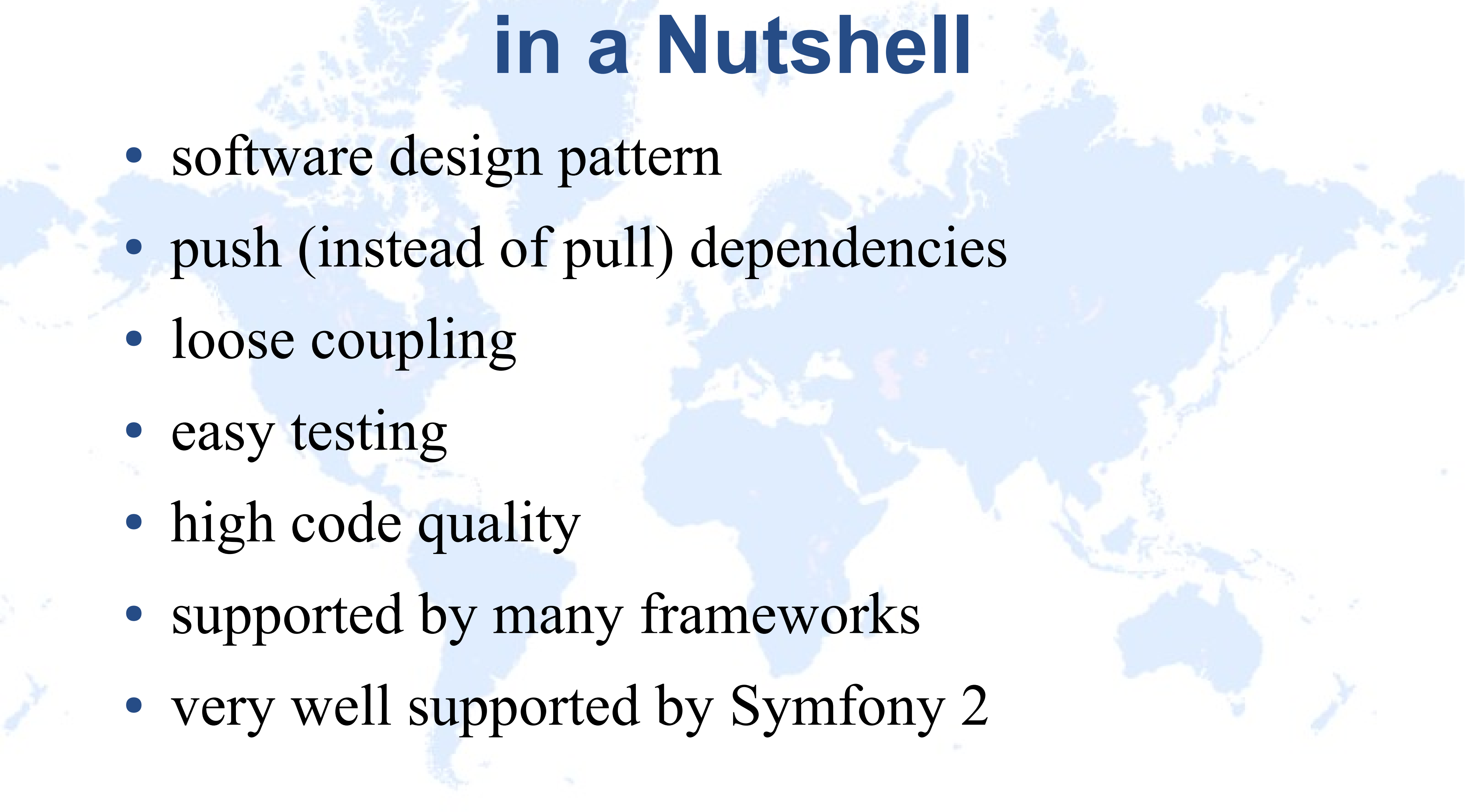
133 FOLGT

74 FOLLOWER

- Developer & Consultant: PHP, Symfony 2 etc.
  - www.php-entwickler-berlin.de
- Trainer & Coach: Symfony 2 workshops 1-5 days
  - www.php-schulung.de
- Available for Your project
  - timon.schroeter@gmail.com

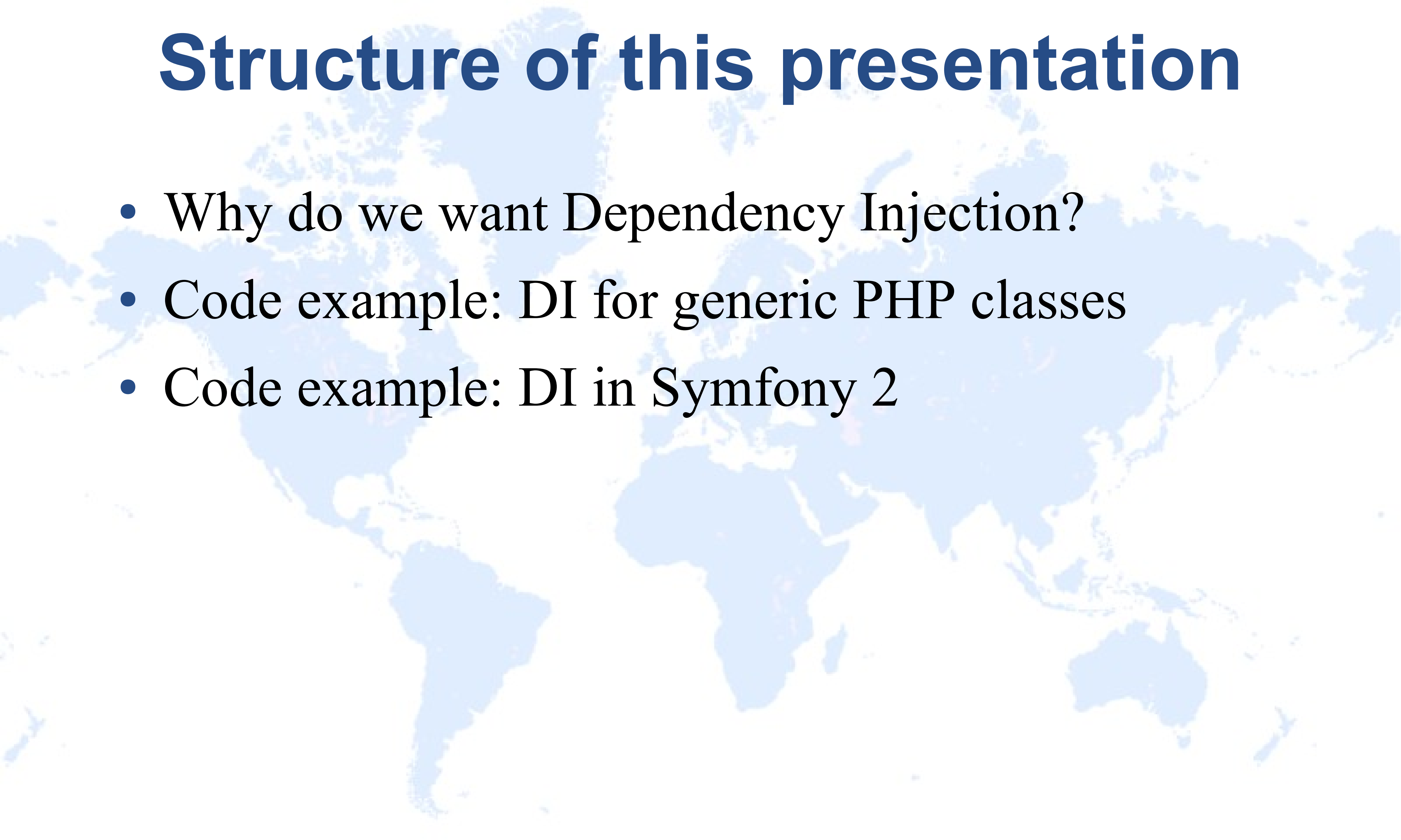
## What is Dependency Injection?

- Your answer?



## Dependency Injection in a Nutshell

- software design pattern
- push (instead of pull) dependencies
- loose coupling
- easy testing
- high code quality
- supported by many frameworks
- very well supported by Symfony 2



## Structure of this presentation

- Why do we want Dependency Injection?
- Code example: DI for generic PHP classes
- Code example: DI in Symfony 2

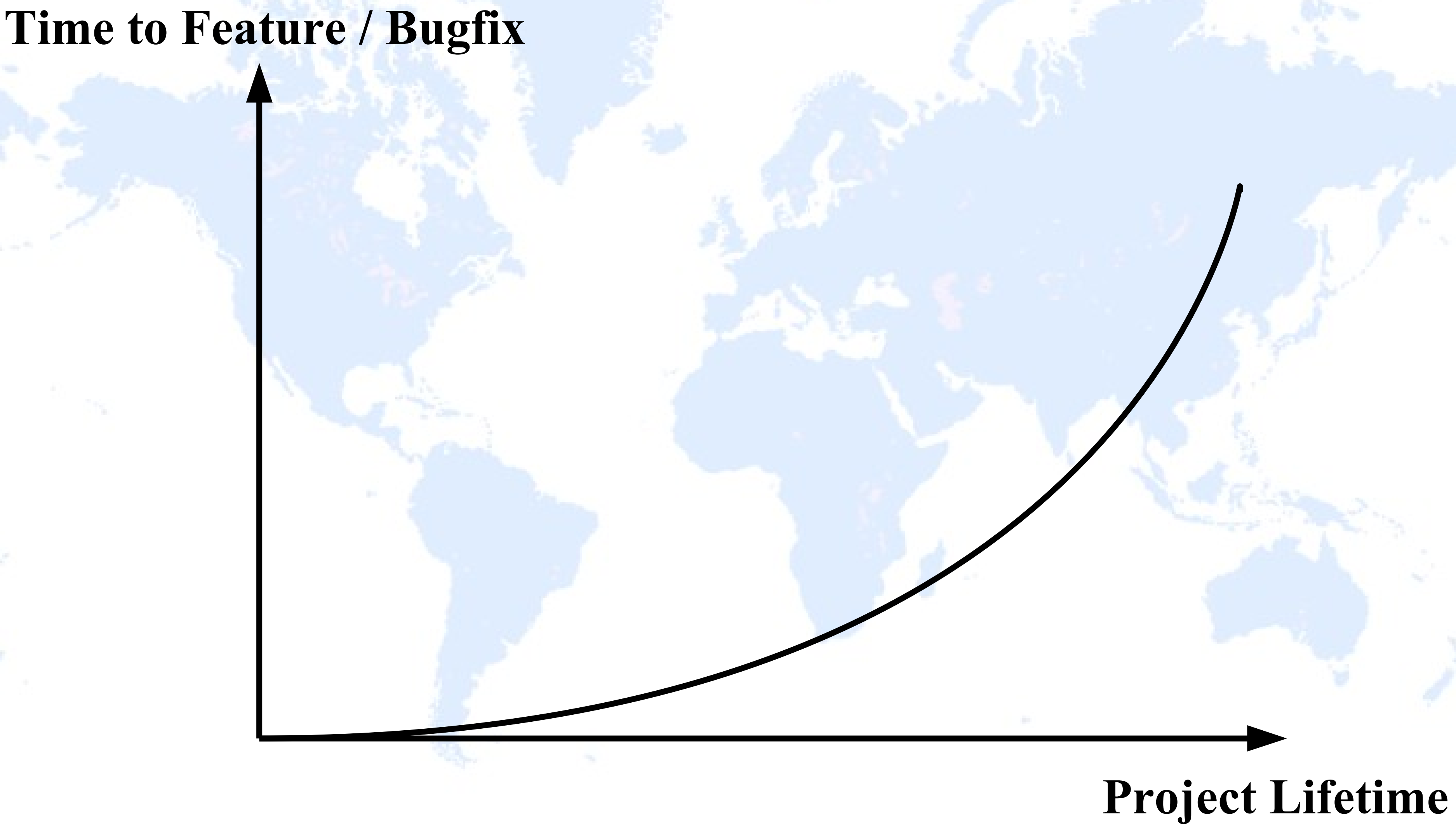
## Structure of this presentation

- Why do we want Dependency Injection? ⬅ You are here
- Code example: DI for generic PHP classes
- Code example: DI in Symfony 2

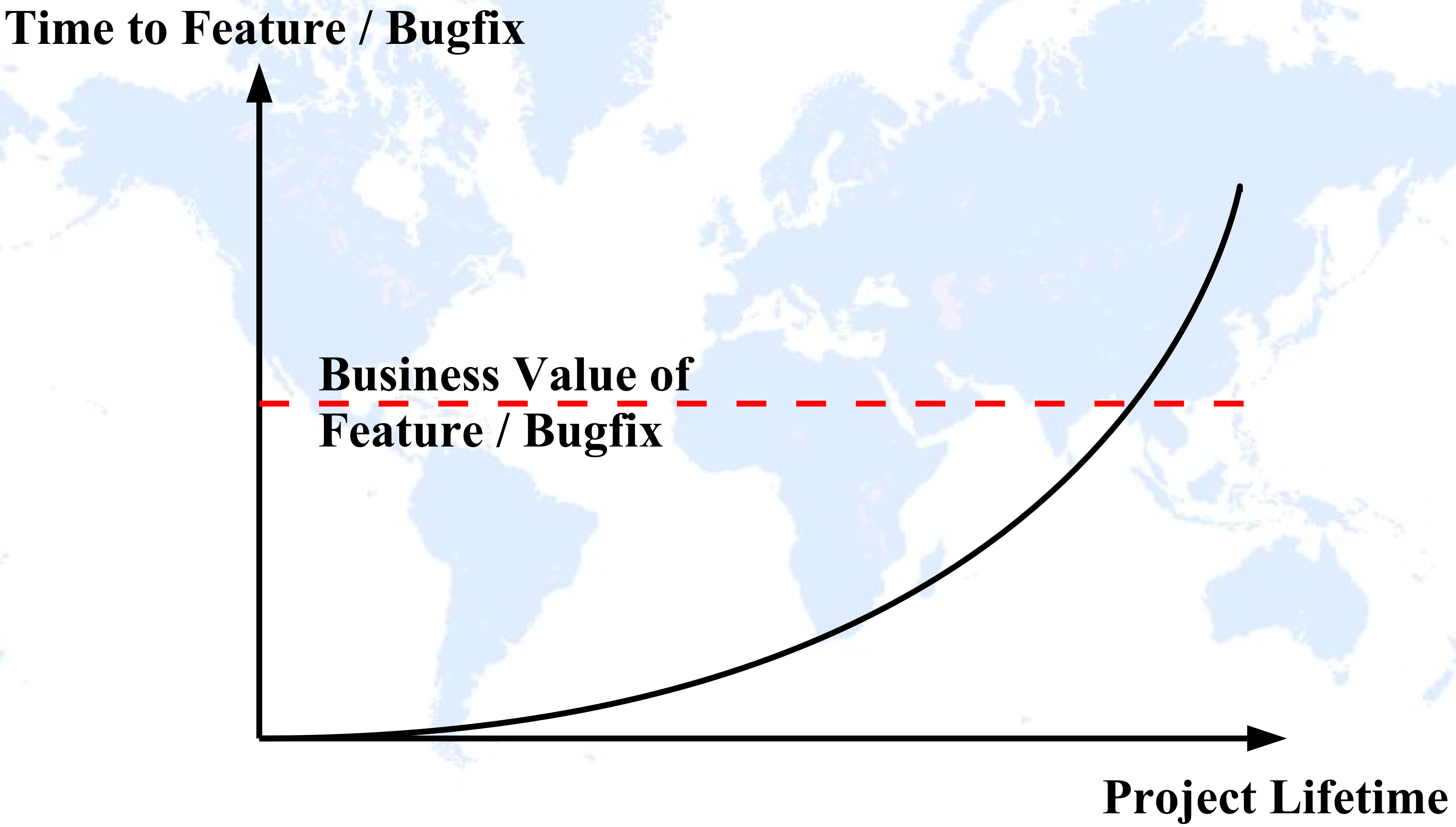
## Why do we want to use Dependency Injection?

- Who has ever worked on a project that was more than 2 years old?

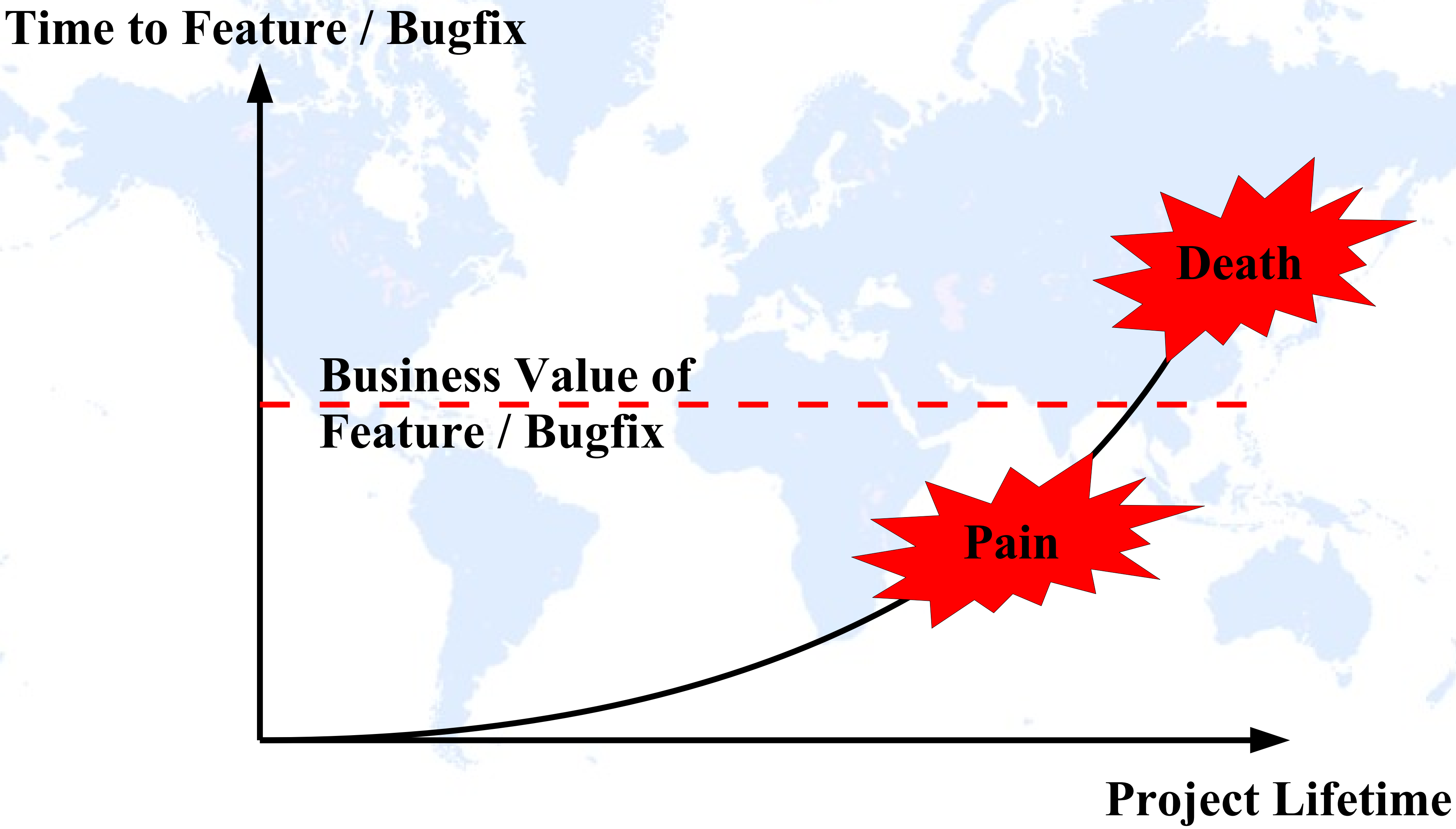
## Project without a really good QA and testing strategy



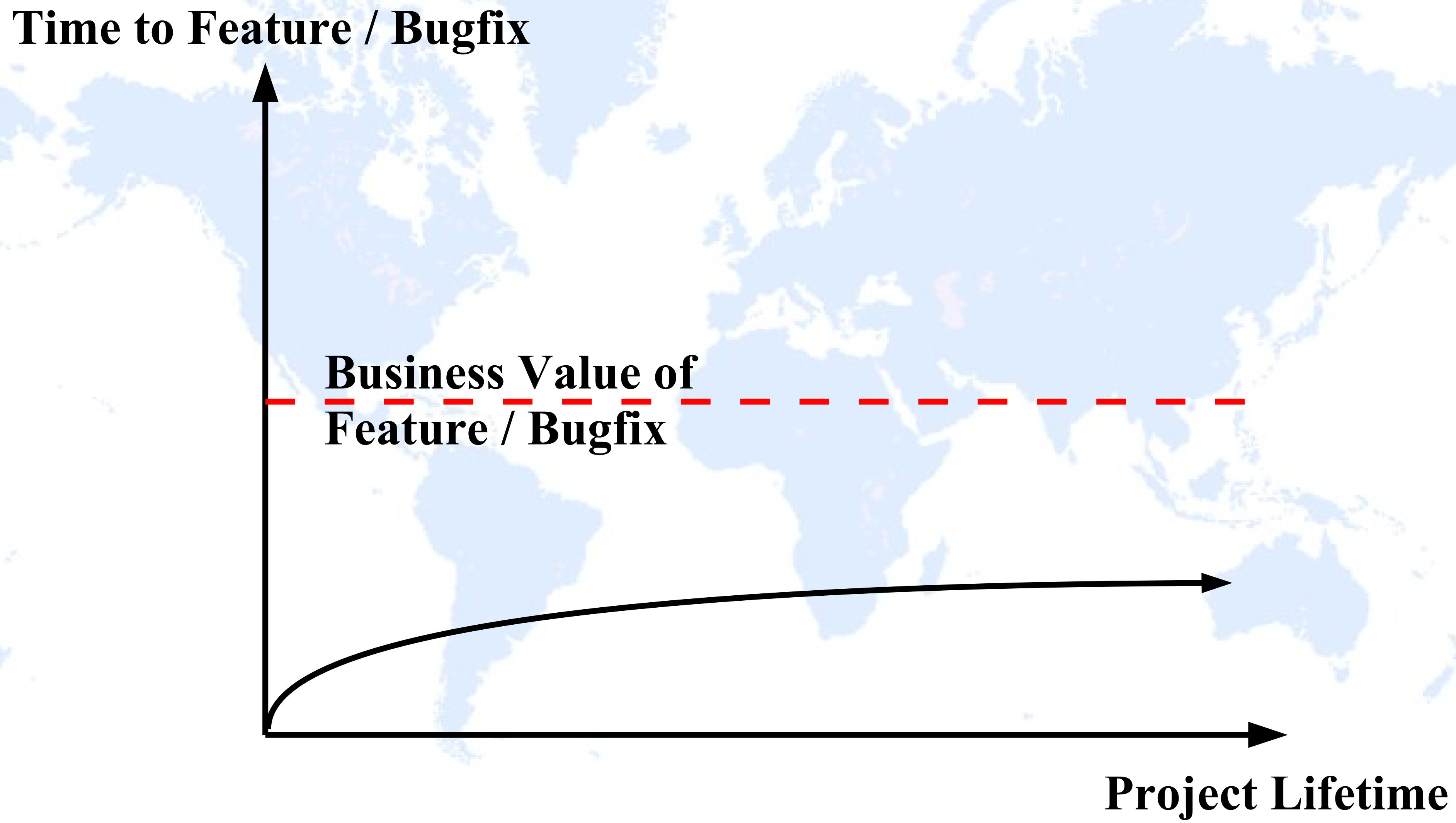
# Project without a really good QA and testing strategy



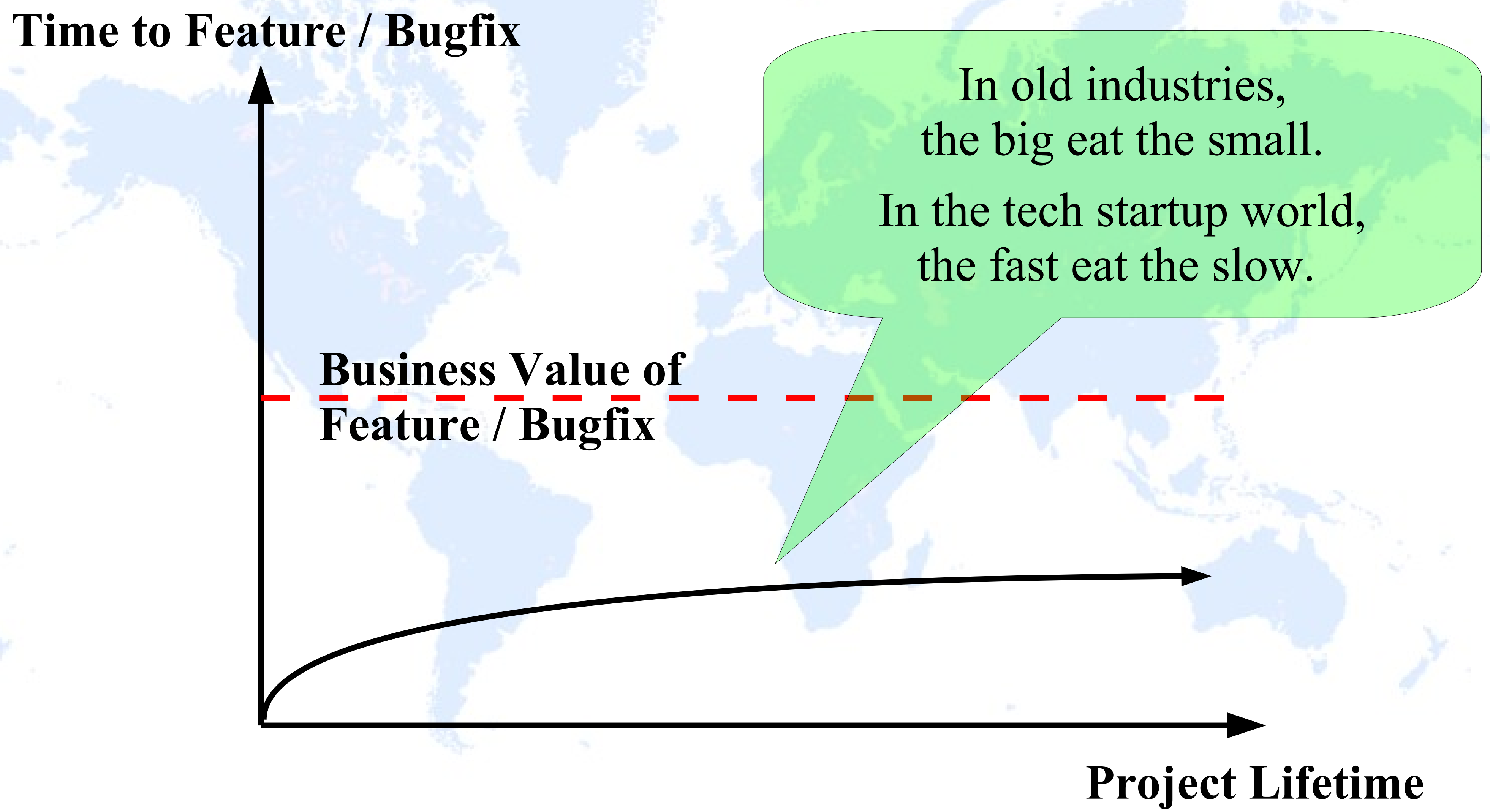
# Project without a really good QA and testing strategy



# Project with a really good QA and testing strategy



# Project with a really good QA and testing strategy



## Good QA and Testing Strategy includes a mix of:

- Acceptance Tests
  - Manual testing by real users
- Functional Tests
  - Automated backbox test (Selenium etc.)
- Integration Tests
  - Tests two or more classes together
- (real) Unit Tests
  - Tests one(!) class

# Good QA and Testing Strategy includes a mix of:

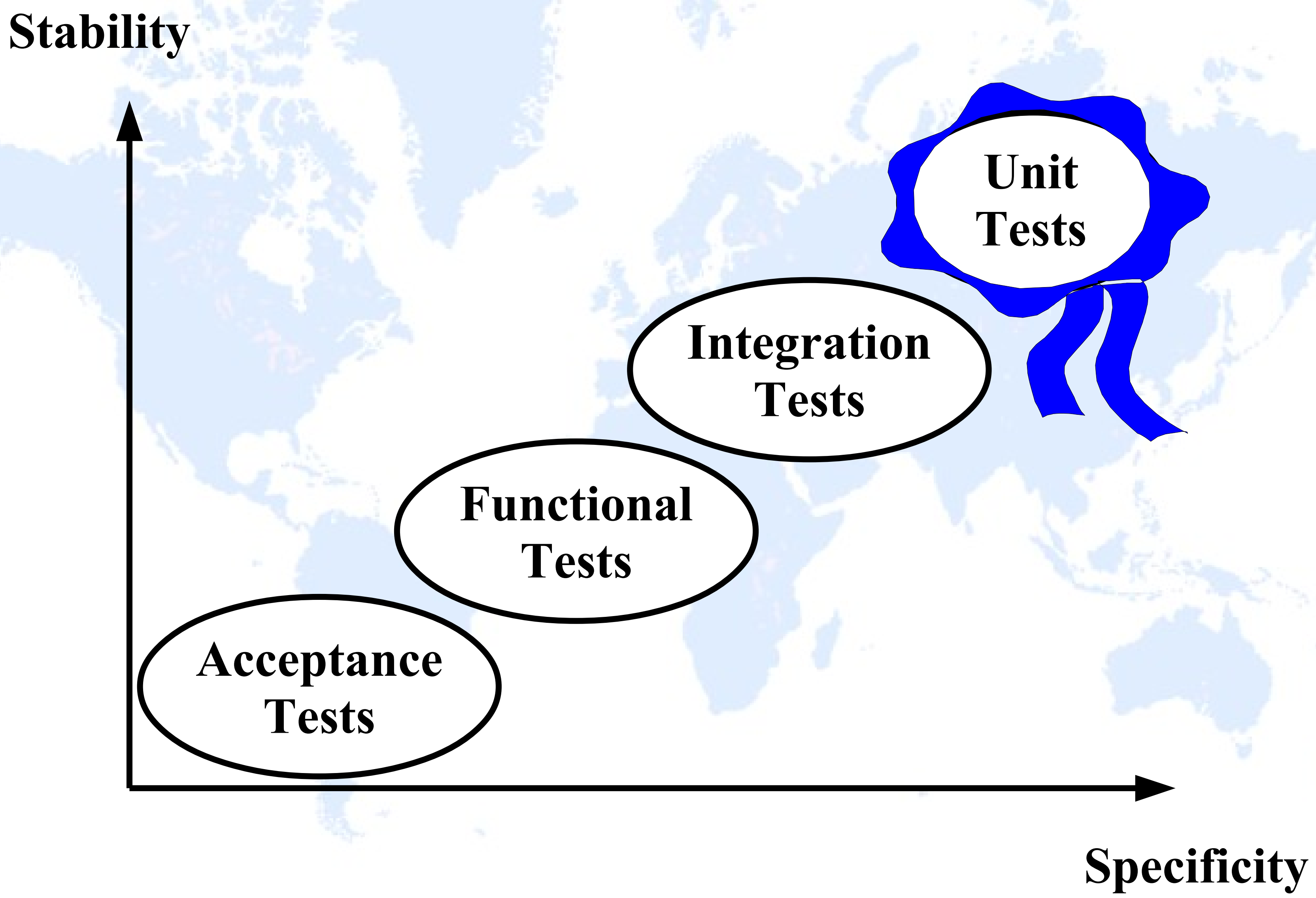
- Acceptance Tests
  - Manual testing by real users
- Functional Tests
  - Automated backbox test (
- Integration Tests
  - Tests two or more classes together
- (real) Unit Tests
  - Tests one(!) class

**Testing can be hard.**

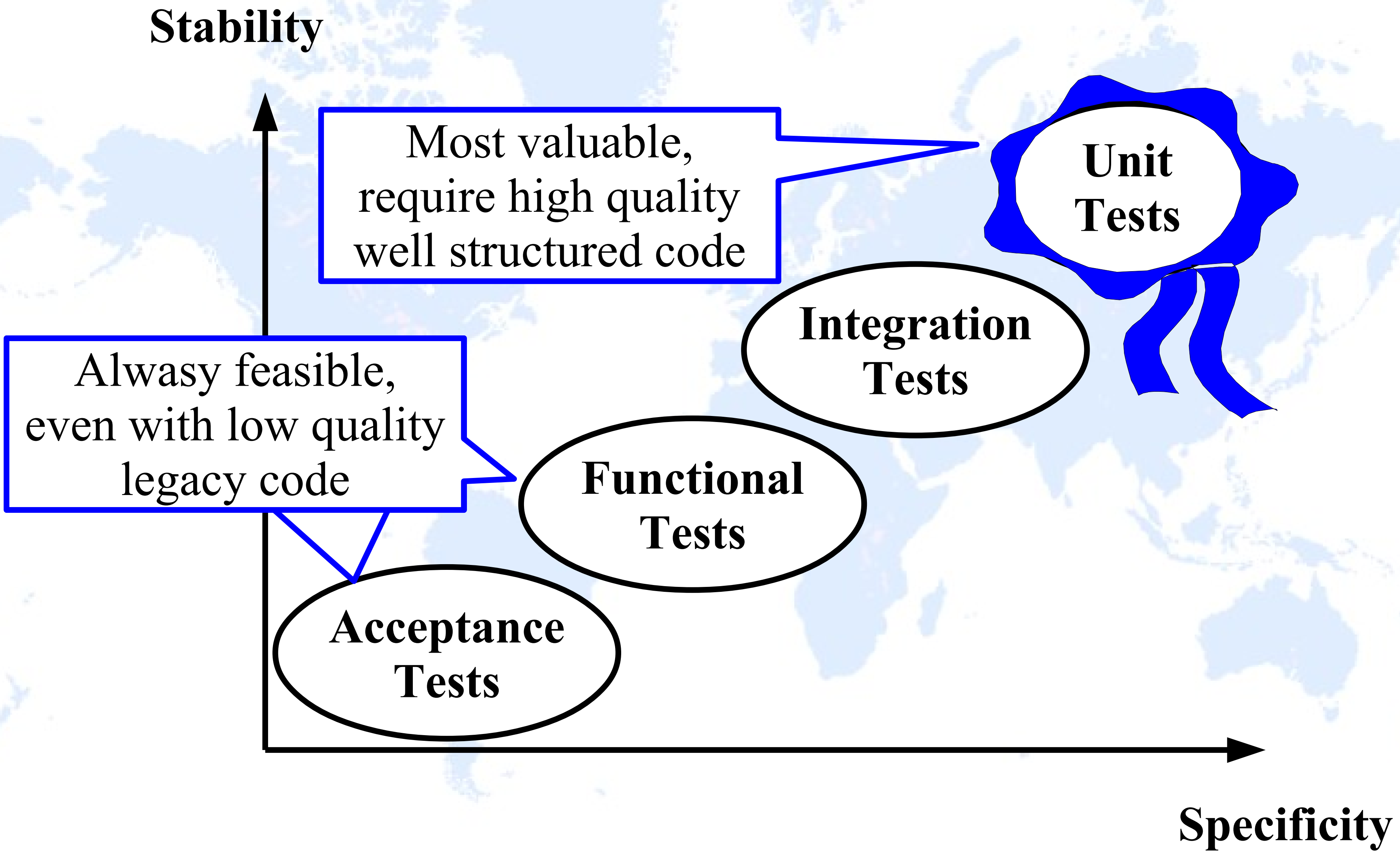
**Reality check:**

**How do You test?**

## Not all tests are created equal



## Not all tests are created equal


Stability
Most valuable, require high quality well structured code
Unit Tests
Integration Tests
Alwasy feasible, even with low quality legacy code
Functional Tests
Acceptance Tests
Specificity

## (Real) Unit Tests

- Test a single unit of code, i.e. a **single class**
- Validate correct functionality and API of each class
- Help avoid regressions
- Facilitate migrations (server, PHP version etc.)
- Ensure backwards compatability of new code

small dedicated classes & methods

## (Real) Unit Tests

- Test a single unit of code, i.e. a **single class**
- Validate correct functionality and API of each class
- H

stable well designed API

- Facilitate migratio (server, PHP version etc.)
- Ensure backwards compatability of new code

small dedicated classes & methods

# Real Unit Tests

- Test a single unit of code, i.e. a **single class**
- Validate correct functionality and AP f each class
- H

stable well designed API

- Facilitate migratio (server, PH
- Ensure backwards compatability of new code

We need to be able to **replace** (mock/stub) **dependencies dynamically**

## Structure of this presentation

- Why do we want Dependency Injection?
- Code example: DI for generic PHP classes ← You are here
- Code example: DI in Symfony 2

Why is this class difficult to unit test?

```
<?php
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

private $client;
private $logger;

class FeedAggregator {
   __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()) {
         $this->logger->log('Could not get: '.$host.$path);
         return null;
      }

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

```php
<?php
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

private $client;
private $logger;

class FeedAggregator {
    __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()) {
            $this->logger->log('Could not get: '.$host.$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
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

private $client;
private $logger;

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

    public function retrieveFee
        $request = $this->client                              h);
        $response = $request->send()
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$host.$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?

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

```php
<?php
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

private $client;
private $logger;

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

    public function retrieveFee
        $request = $this->client                                   h);
        $response = $request->send()
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$host.$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?

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

```php
<?php
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

            nt;
            er;

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

    public function retrieveFee
        $request = $this->client                               h);
        $response = $request->send(
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$host.$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?

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

```
<?php
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

            ent
            er

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

    public function retrieveFee
        $request = $this->client                                        h);
        $response = $request->send()
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$host.$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
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;
             nt
             er
             r
             t
      $this->c
      $this->l
   }

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

What if we want unit tests to run fast
aiting for the network?

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

nit tests to run fast without logging?

Dependencies are **pulled**.

```php
<?php
use Guzzle\Http\Client;
use Acme\Logger\XmlLogger;

private $client;
private $logger;

class FeedAggregator {
   __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()) {
         $this->logger->log('Could not get: '.$host.$path);
         return null;
      }

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

Dependencies are **pushed**.

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

private $client;
private $logger;

class FeedAggregator {
   __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();
   }
   // ...
}
```

Dependencies are **pushed**.

Class only depends on **interfaces**

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

private $client;
private $logger;

class FeedAggregator {
   __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 Acme\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

private $client;
private $logger;

class FeedAggregator {
    __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();
    }
    // ...
}
```

Dependencies are **pushed**.

Class only depends on **interfaces**

Implementations are **injected** at runtime

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

private $client;
private $logger;

class FeedAggregator {
   __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();
   }
   // ...
}
```

Dependencies are **pushed**.

Class only depends on **interfaces**

Implementations are **injected** at runtime

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

private $client;
private $logger;

class FeedAggregator {
    __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();
    }
    // ...
}
```

Dependencies are **pushed**.

Class only depends on **interfaces**

Implementations are **injected** at runtime

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

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

private $client;
private $logger;

class FeedAggregator {
    __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                                         {
            $this->logg                                      '.$host.$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
use Acme\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

private $client;
private $logger;

class FeedAggregator {
   __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                                       {
         $this->logg                                       '.$host.$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
use Acme\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

private $client;
private $logger;

class FeedAggregator {
   __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                                  {
         $this->logg                                  '.$host.$path);
         return nul
      }

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

Dependencies are **pushed**.

Class only depends on **interfaces**

Implementations are **injected** at runtime

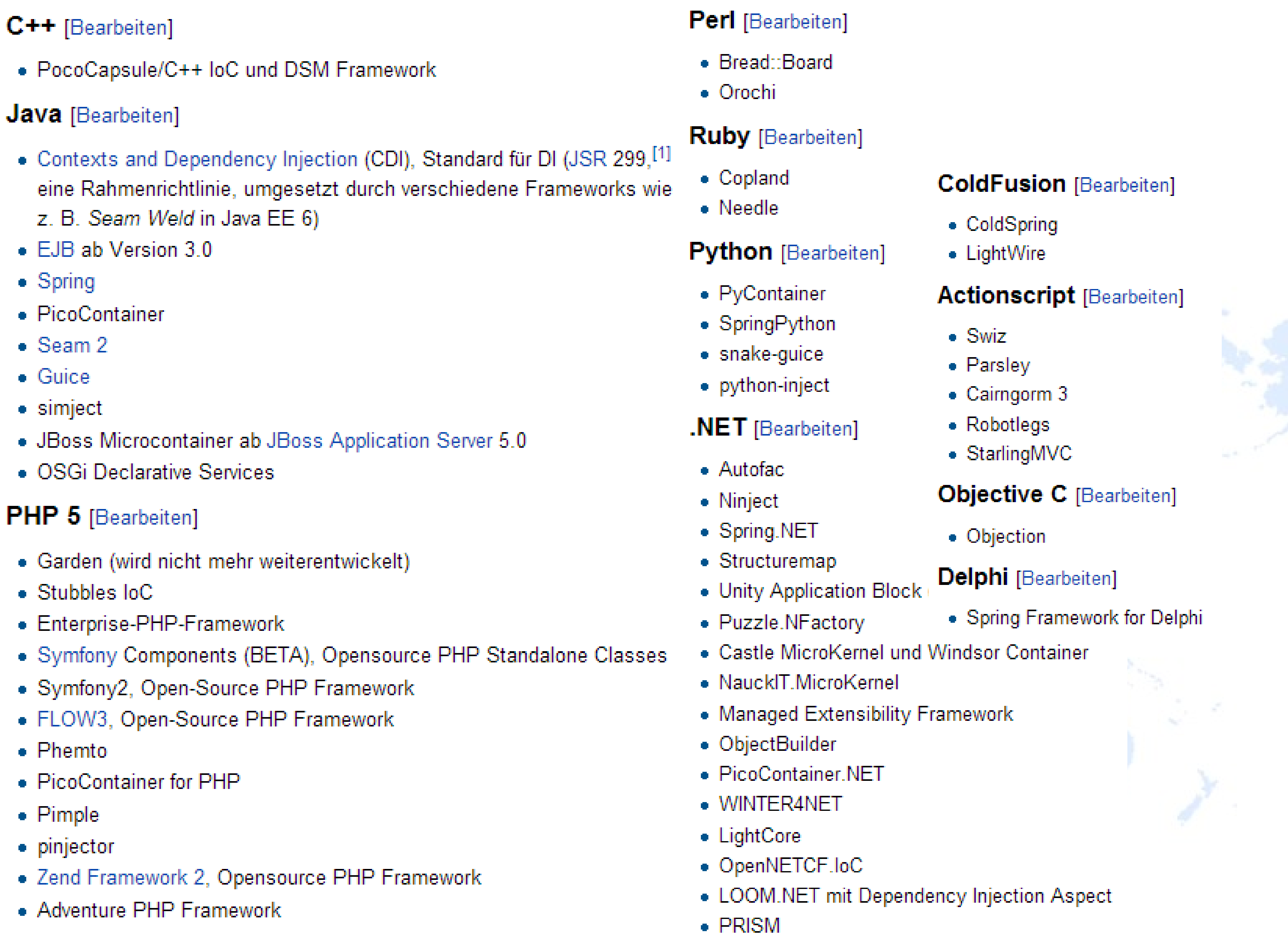
Easy to **replace**, even **dynamically** (for testing)

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

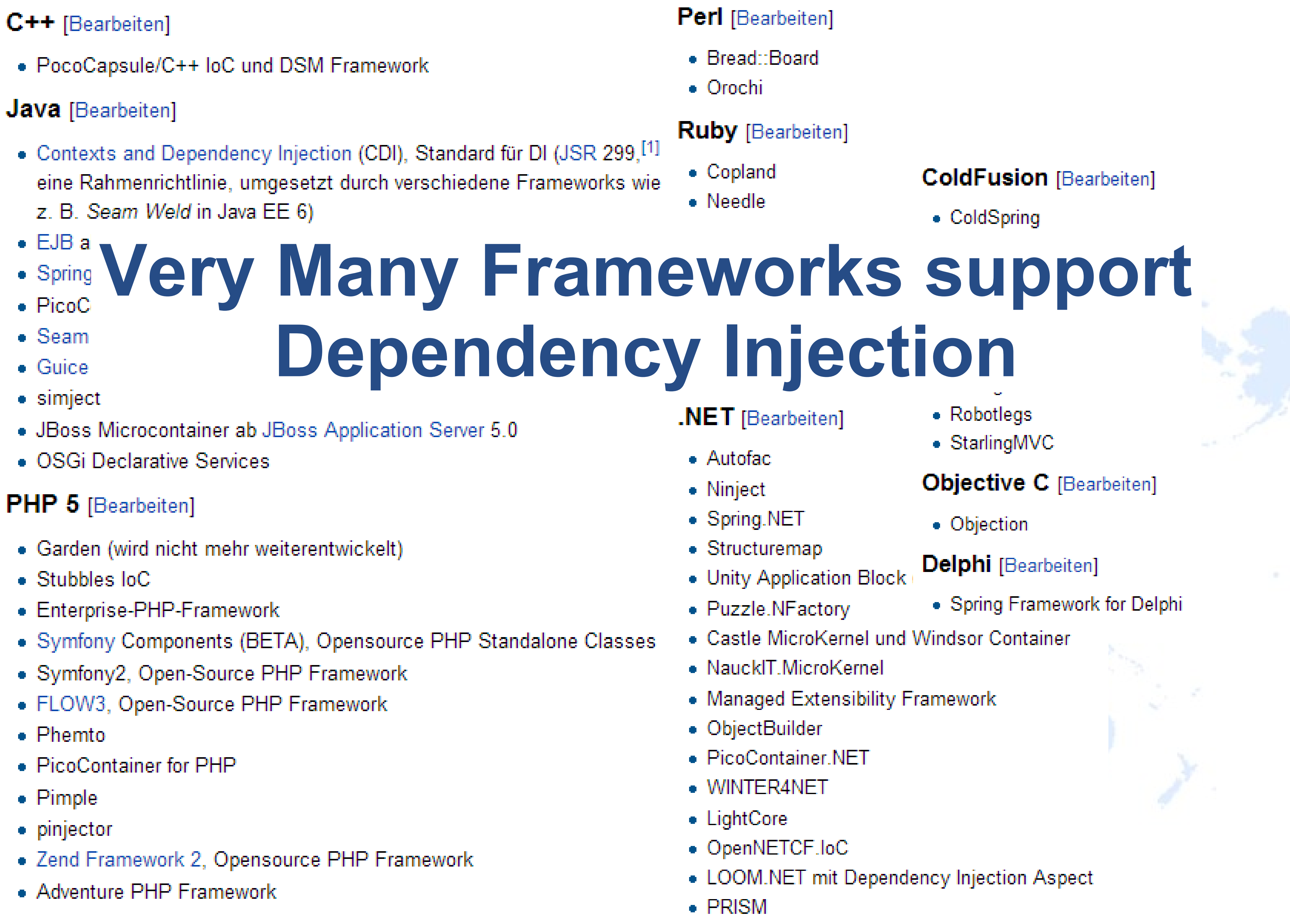
## Objective C [Bearbeiten]

- 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

ColdFusion [Bearbeiten]

- ColdSpring

- Robotlegs
- StarlingMVC

.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

Objective C [Bearbeiten]

- Objection

Delphi [Bearbeiten]

- Spring Framework for Delphi

## Structure of this presentation

- Why do we want Dependency Injection?
- Code example: DI for generic PHP classes
- Code example: DI in Symfony 2 ← You are here

## DI ist very easy in Symfony 2

Symfony

## DI ist 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 ist 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 ist very easy in Symfony 2

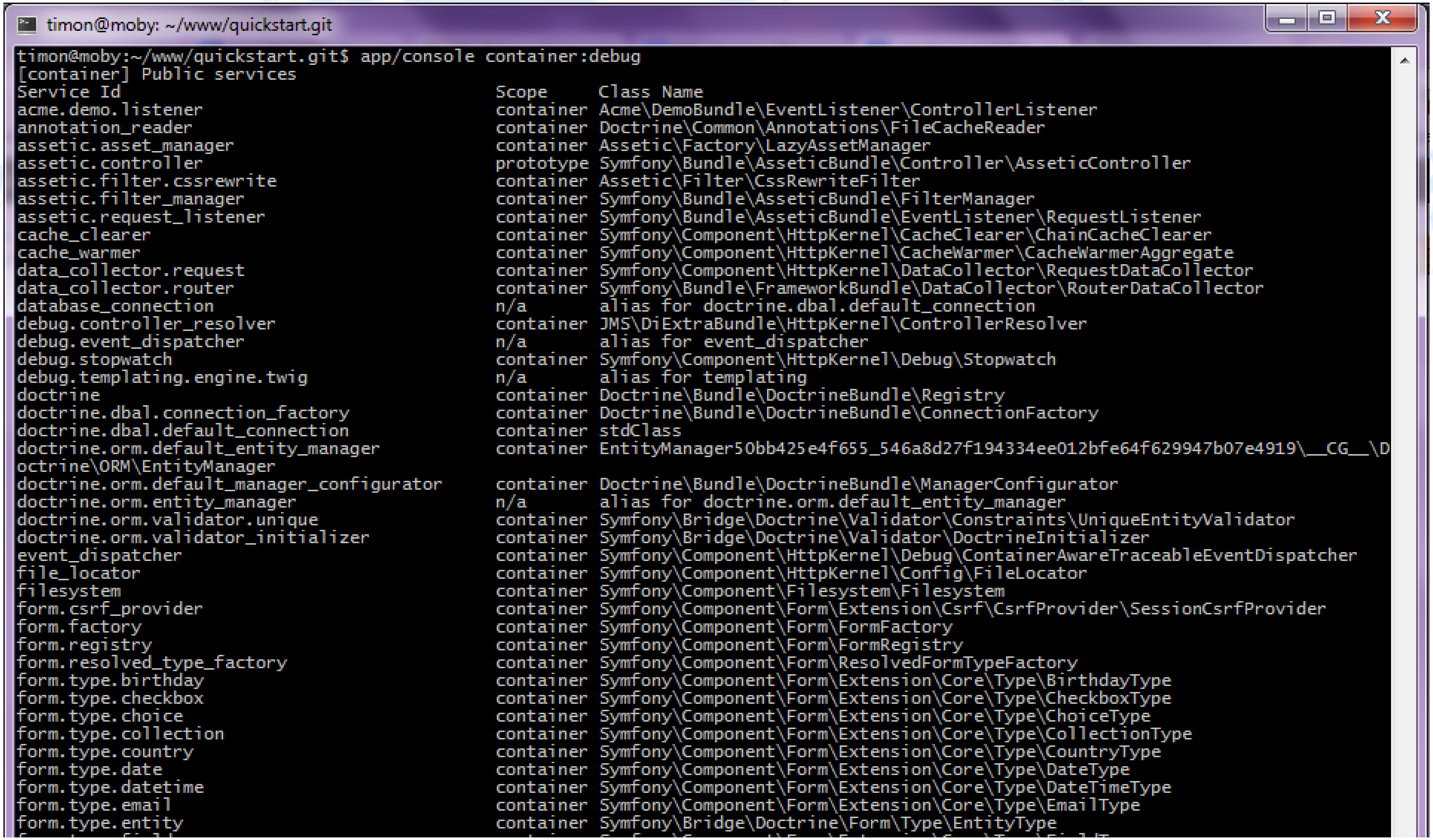
- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

```
# 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

```
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 ist 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 ist 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:
                - arry_member
                - arry_member
            even_more:   @another_service
```

# DI ist 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:     Acm                         neClass
        arguments:
            some_arg:    “string”
            another:
                - arry_member
                - arry_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
// src/Acme/FeedBundle/Service/FeedAggregator.php

use Acme\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

private $client;
private $logger;

class FeedAggregator {
   __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
// src/Acme/FeedBundle/Service/FeedAggregator.php

use Acme\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

private $client;
private $logger;

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

   public function retrieveFeed ($baseurl, $path) {
      $request = $this->client->setBaseUrl($baseurl)->get($path);
```

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

```
<?php
// src/Acme/FeedBundle/Service/FeedAggregator.php

use Acme\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

private $client;
private $logger;

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

   public function retrieveFeed ($baseurl, $path) {
      $request = $this->client->setBaseUrl($baseurl)->get($path);
```

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

```
<?php
// src/Acme/FeedBundle/Service/FeedAggregator.php

use Acme\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

private $client;
private $logger;

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

   public function retrieveFeed ($baseurl, $path) {
      $request = $this->client->setBaseUrl($baseurl)->get($path);
```

## Different Config for Testing?

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

Put it here here

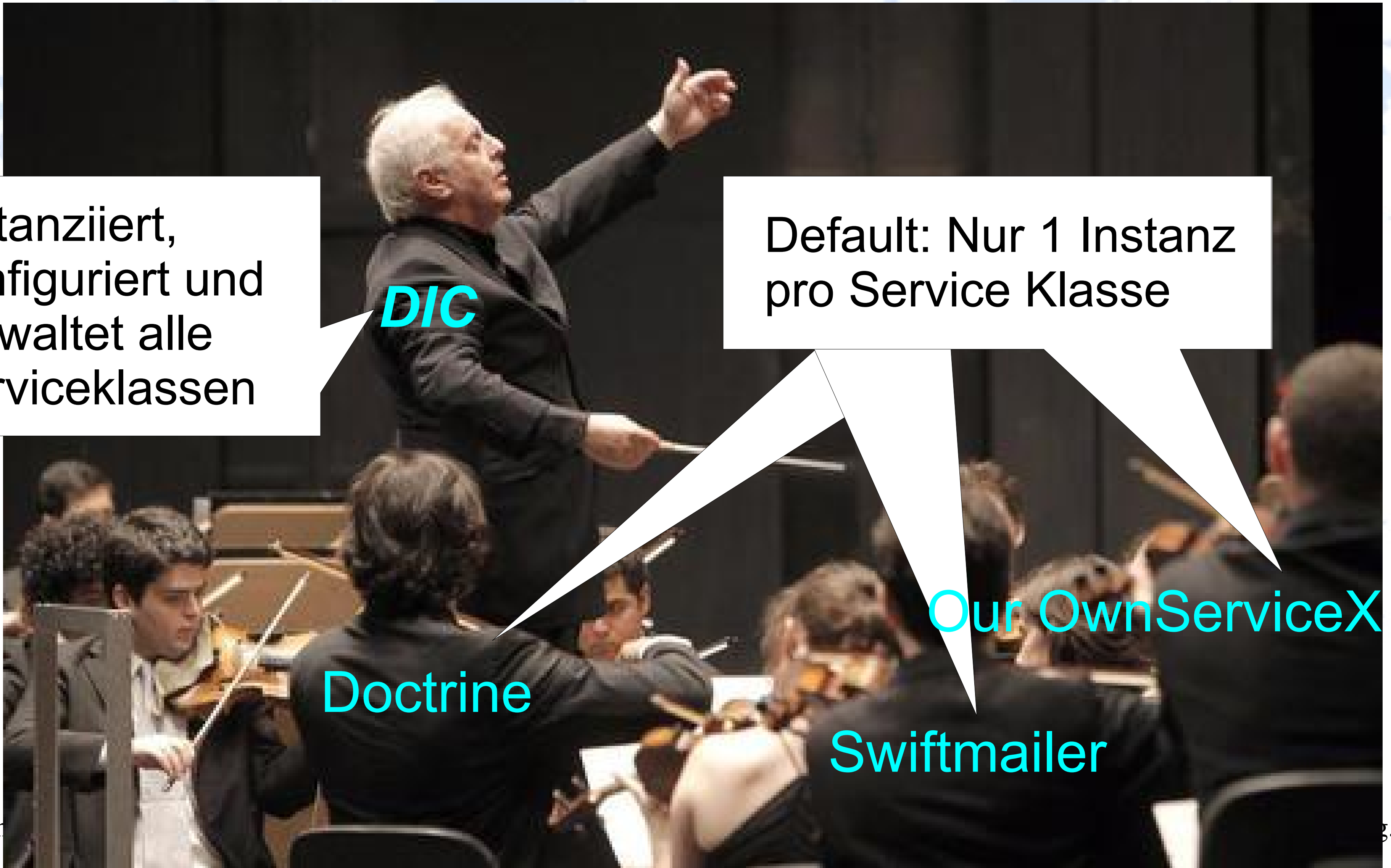
## Dependency Injection Container

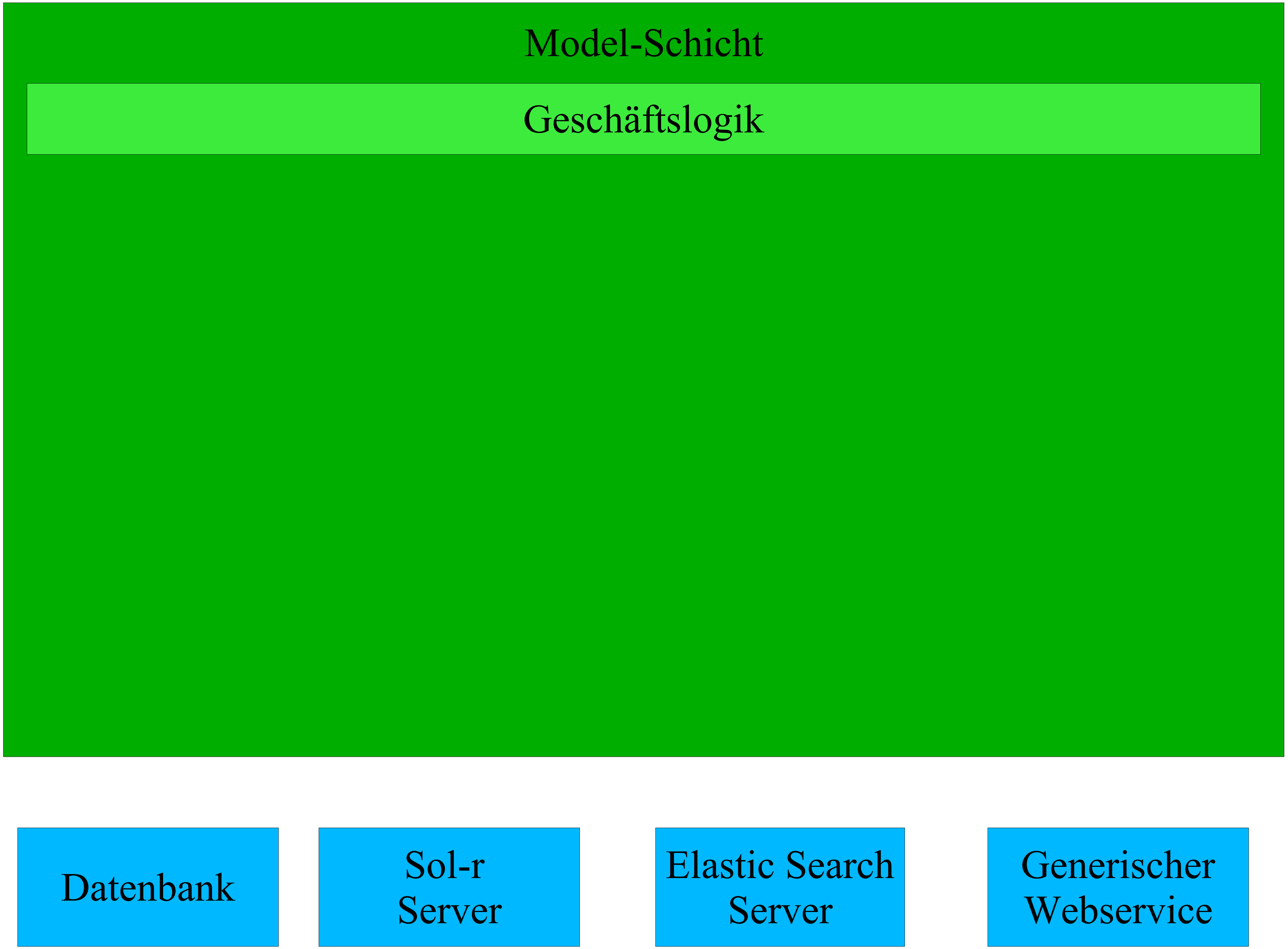
Instanziiert, konfiguriert und verwaltet alle Serviceklassen

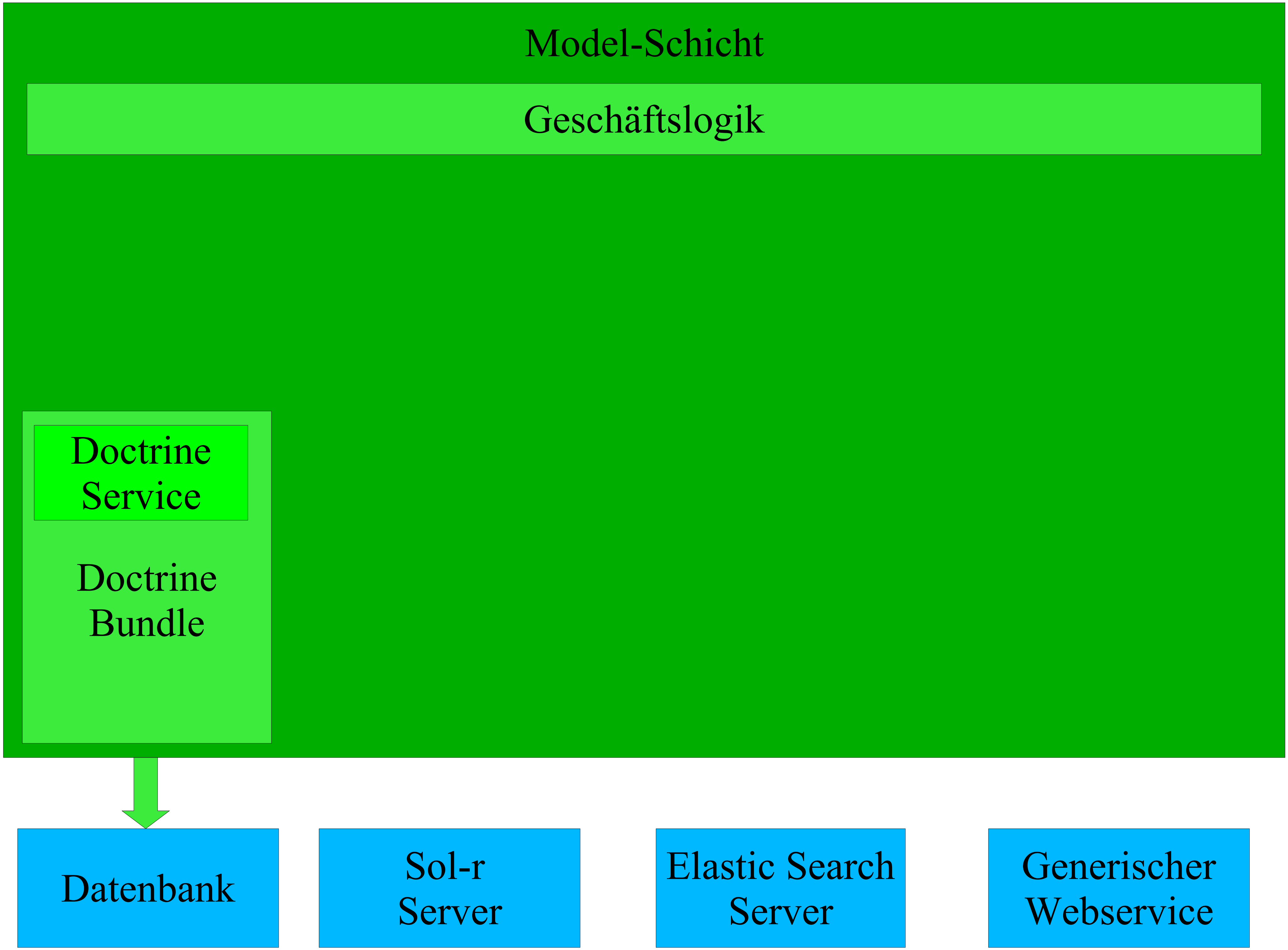
# Dependency Injection Container

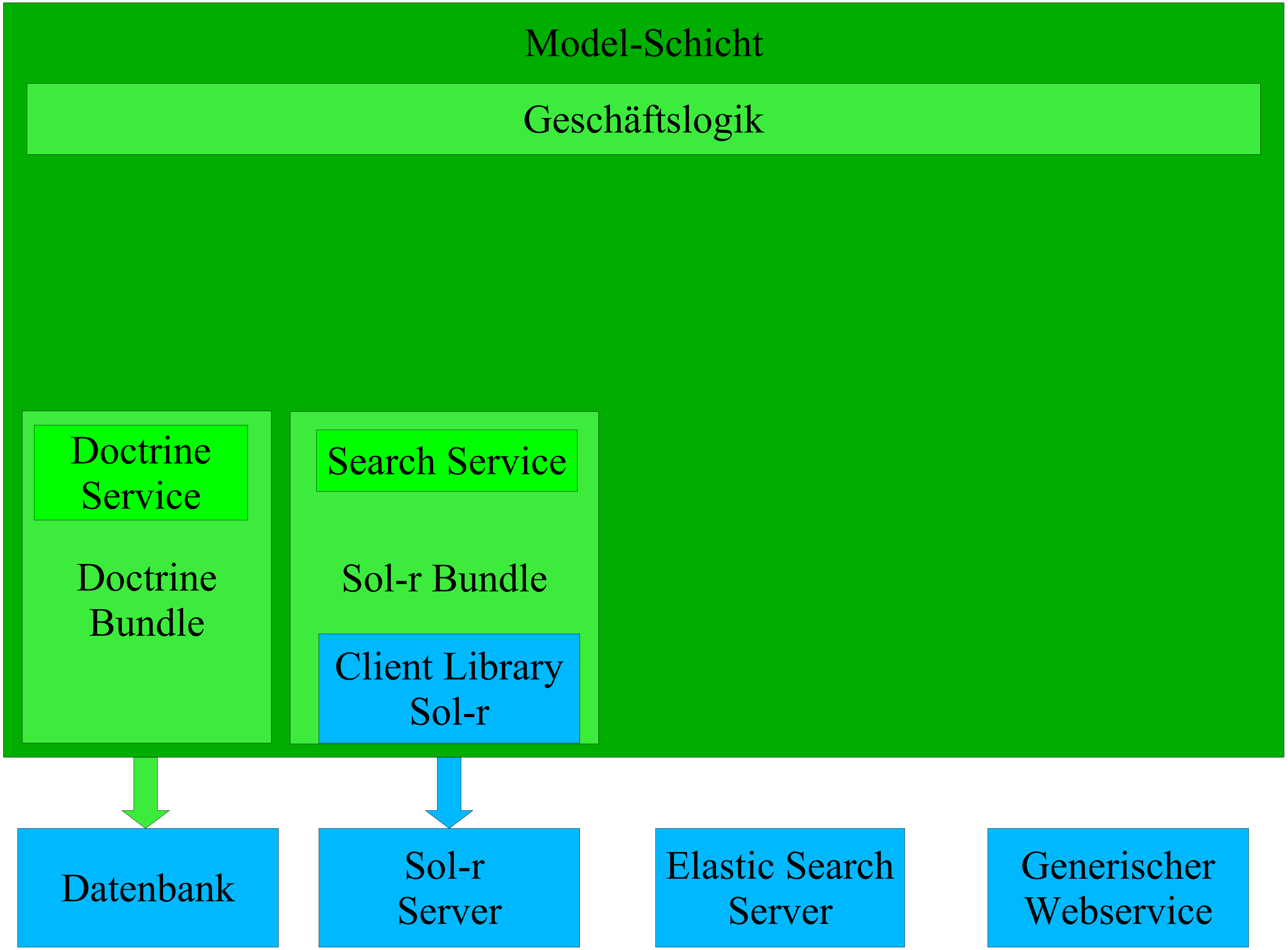
Instanziiert, konfiguriert und verwaltet alle Serviceklassen

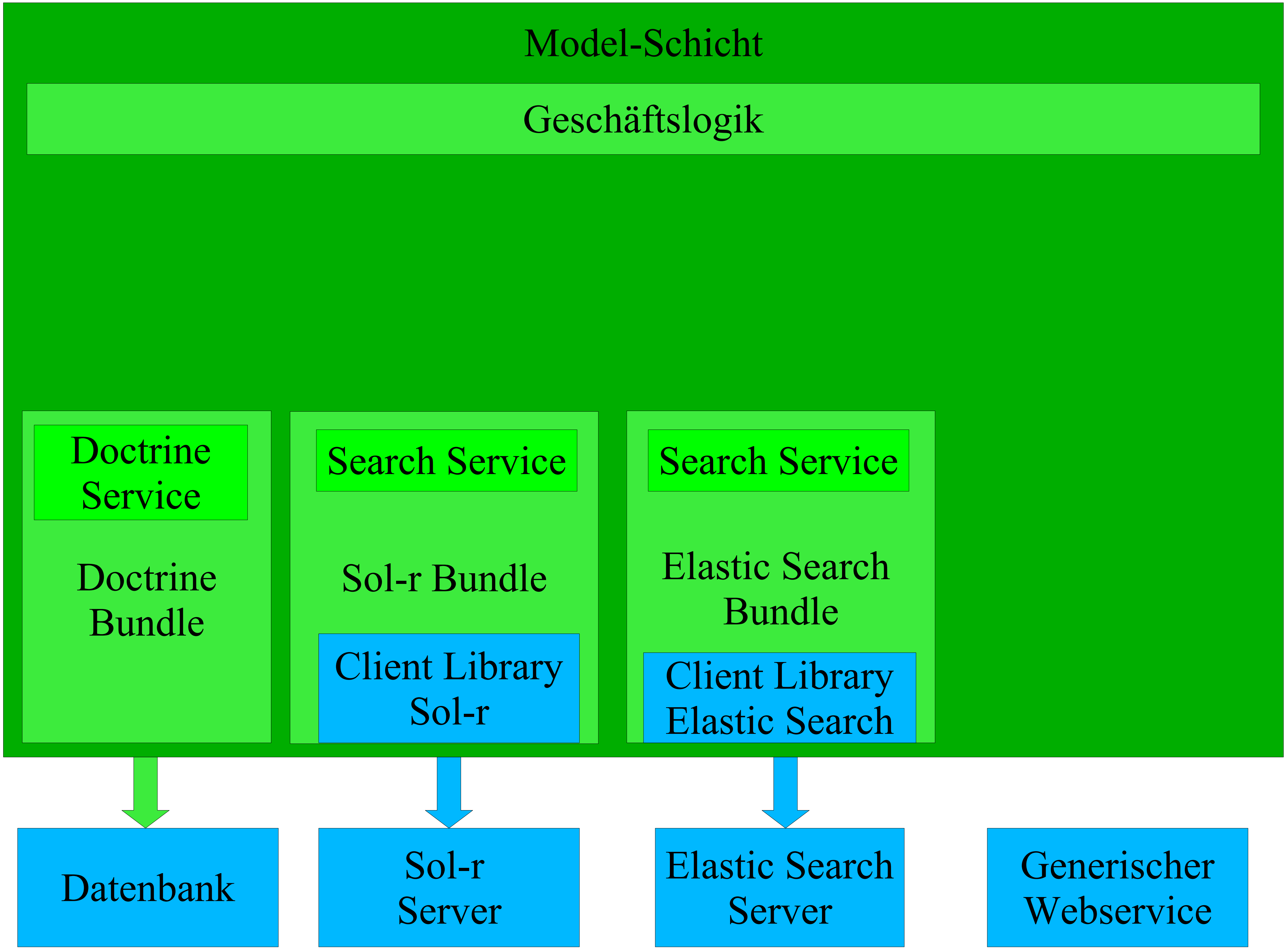
# Dependency Injection Container

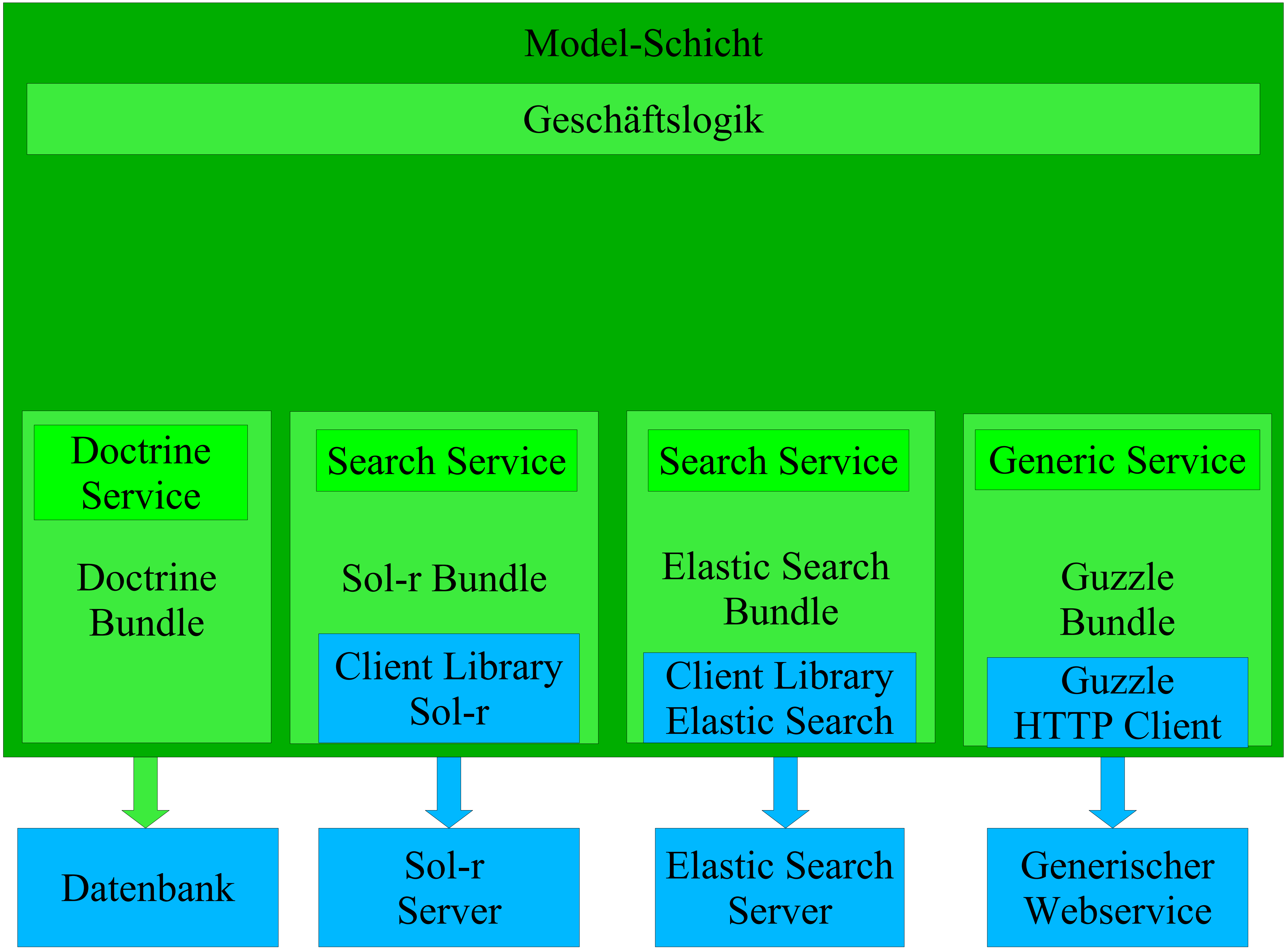

Instanziiert, konfiguriert und verwaltet alle Serviceklassen
DIC
Default: Nur 1 Instanz pro Service Klasse
Doctrine
Swiftmailer
Our OwnServiceX

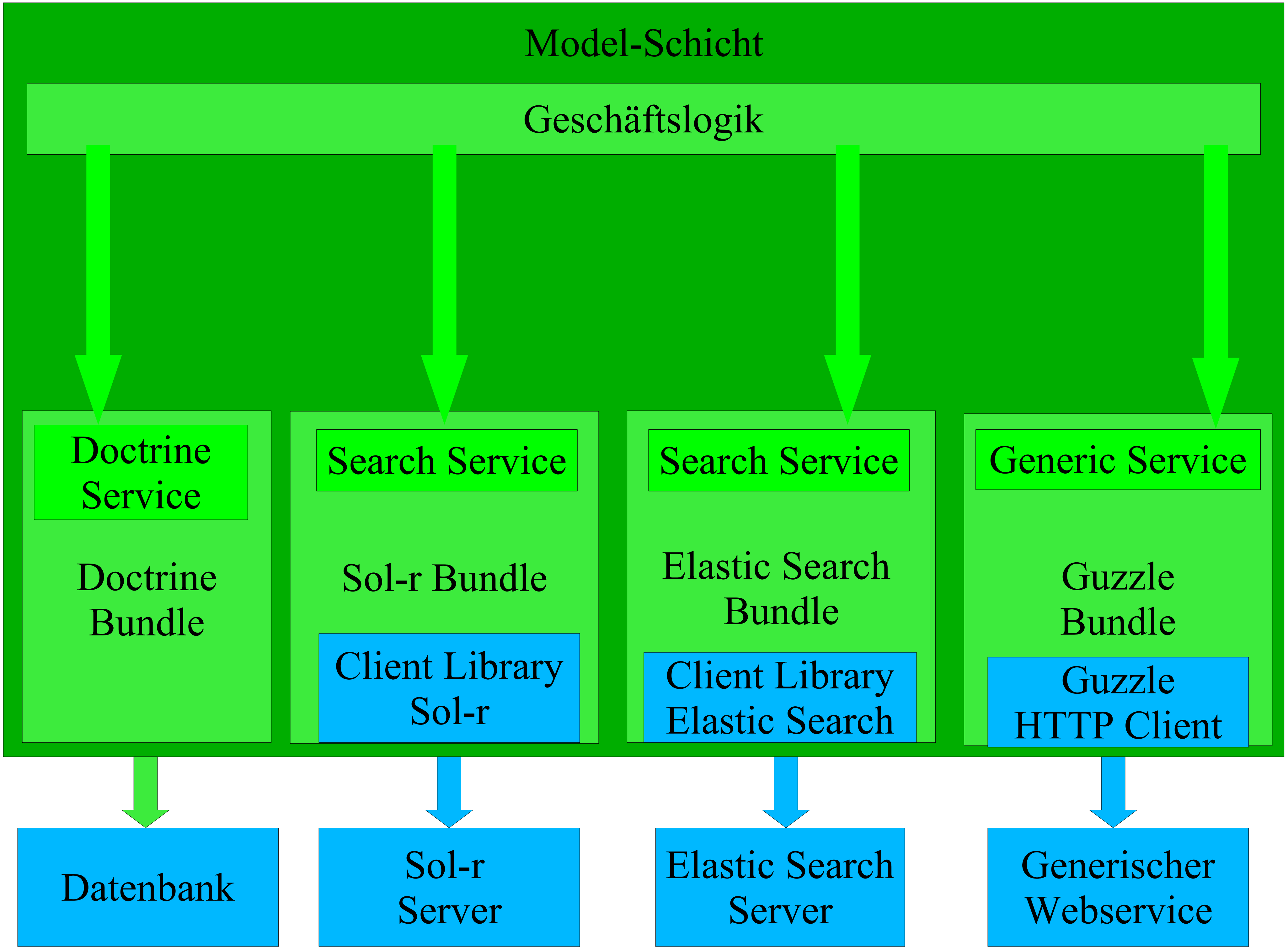

Model-Schicht
Geschäftslogik
Datenbank
Sol-r Server
Elastic Search Server
Generischer Webservice

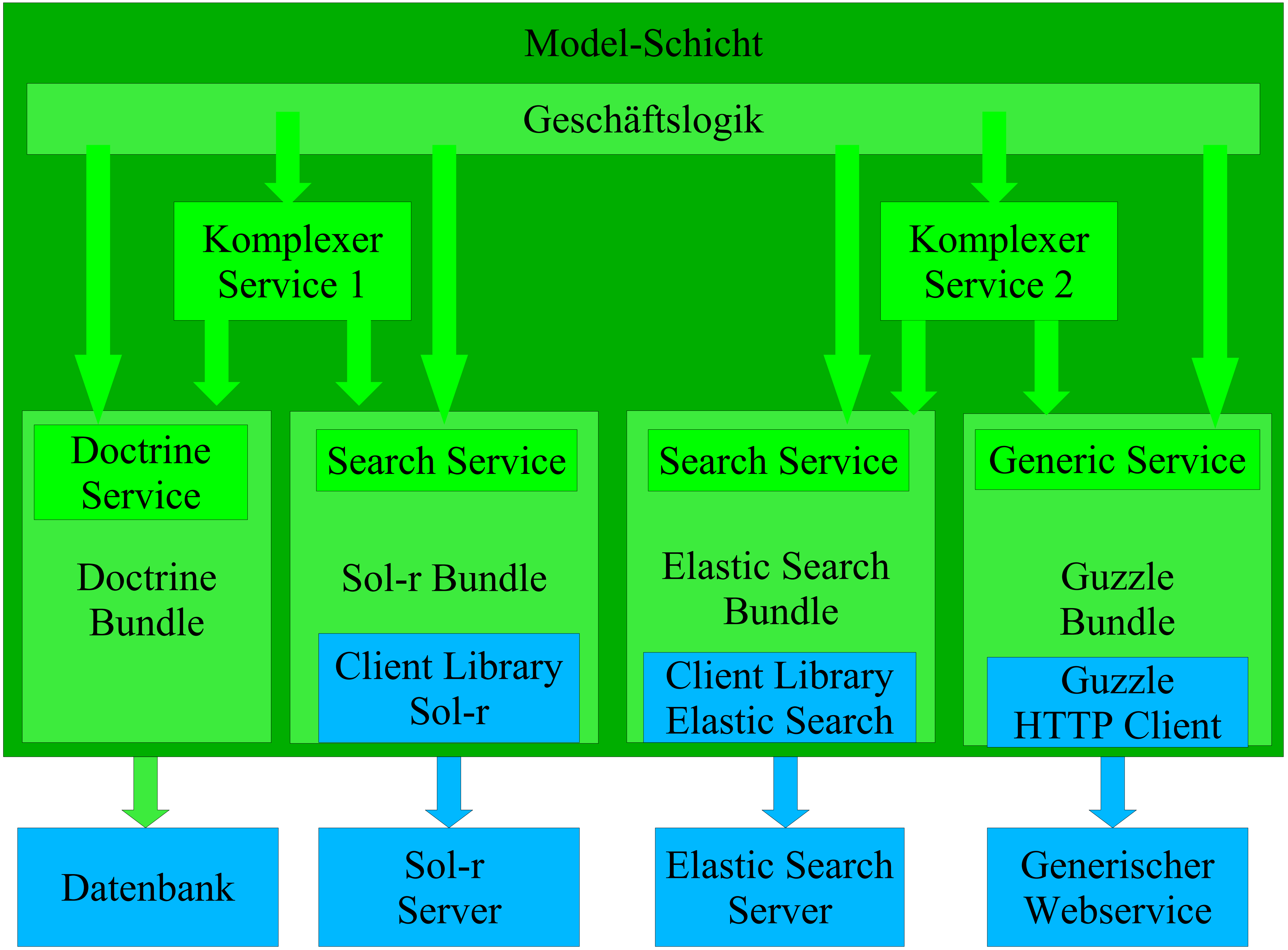

Model-Schicht
Geschäftslogik
Doctrine Service
Doctrine Bundle
Datenbank
Sol-r Server
Elastic Search Server
Generischer Webservice

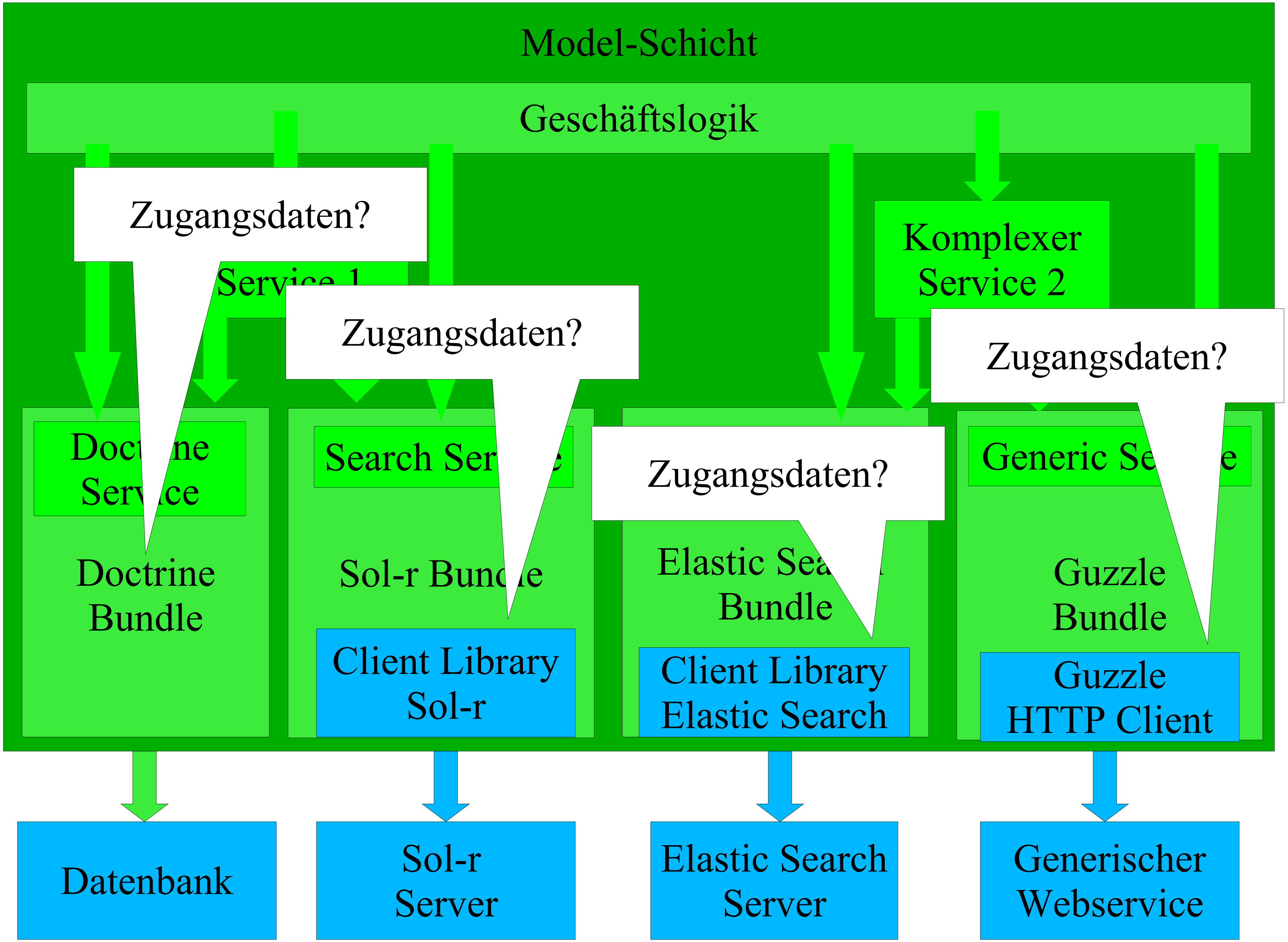

Model-Schicht
Geschäftslogik
Doctrine Service
Doctrine Bundle
Search Service
Sol-r Bundle
Client Library Sol-r
Datenbank
Sol-r Server
Elastic Search Server
Generischer Webservice

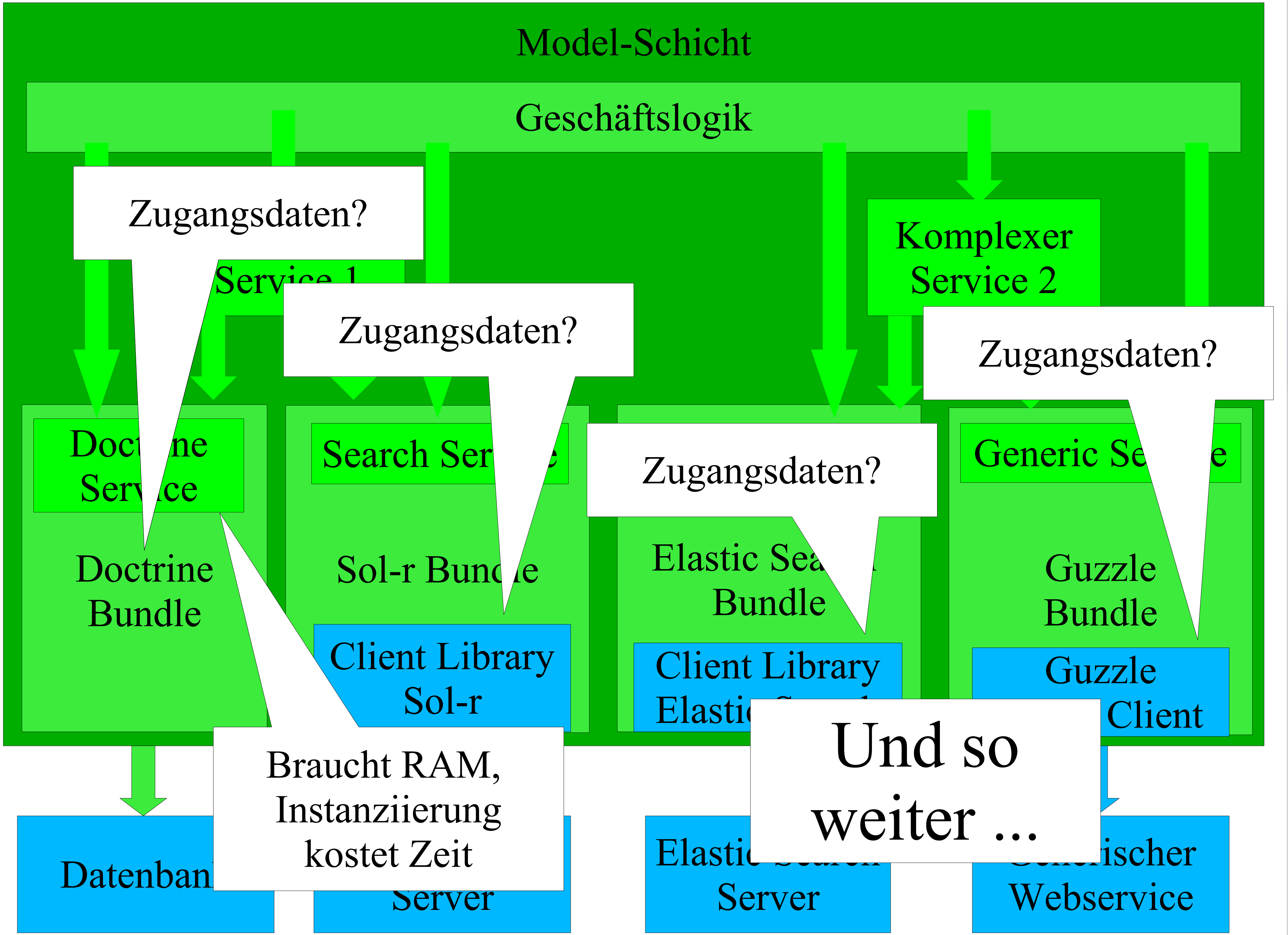

Model-Schicht
Geschäftslogik
Doctrine Service
Doctrine Bundle
Search Service
Sol-r Bundle
Client Library Sol-r
Search Service
Elastic Search Bundle
Client Library Elastic Search
Datenbank
Sol-r Server
Elastic Search Server
Generischer Webservice

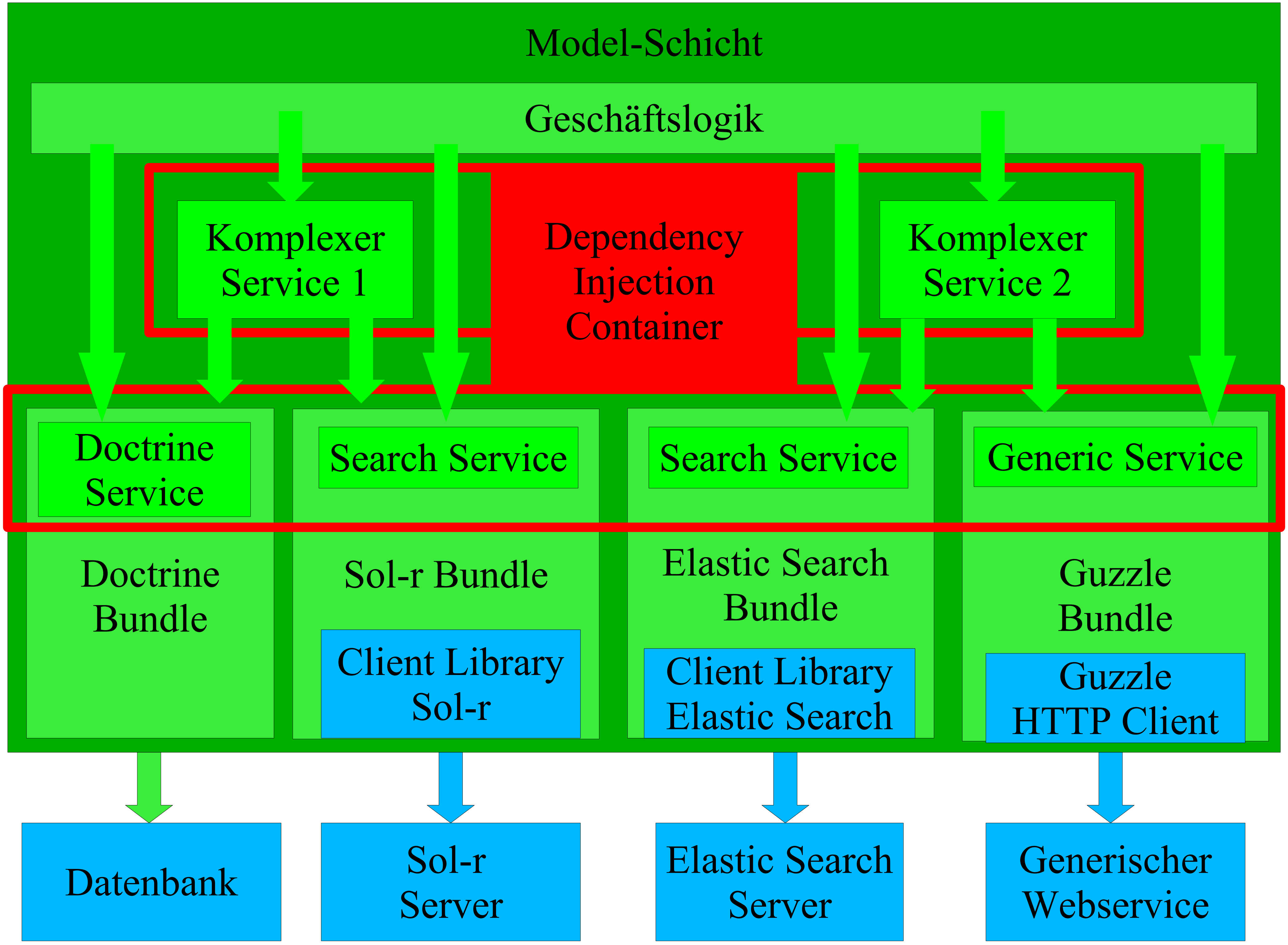

Model-Schicht
Geschäftslogik
Doctrine Service
Doctrine Bundle
Search Service
Sol-r Bundle
Client Library Sol-r
Search Service
Elastic Search Bundle
Client Library Elastic Search
Generic Service
Guzzle Bundle
Guzzle HTTP Client
Datenbank
Sol-r Server
Elastic Search Server
Generischer Webservice


Model-Schicht
Geschäftslogik
Doctrine Service
Doctrine Bundle
Search Service
Sol-r Bundle
Client Library Sol-r
Search Service
Elastic Search Bundle
Client Library Elastic Search
Generic Service
Guzzle Bundle
Guzzle HTTP Client
Datenbank
Sol-r Server
Elastic Search Server
Generischer Webservice


Model-Schicht
Geschäftslogik
Komplexer Service 1
Komplexer Service 2
Doctrine Service
Doctrine Bundle
Search Service
Sol-r Bundle
Client Library Sol-r
Search Service
Elastic Search Bundle
Client Library Elastic Search
Generic Service
Guzzle Bundle
Guzzle HTTP Client
Datenbank
Sol-r Server
Elastic Search Server
Generischer Webservice


Model-Schicht
Geschäftslogik
Zugangsdaten?
Service 1
Zugangsdaten?
Komplexer Service 2
Zugangsdaten?
Doct ine Serv ce
Search Ser e
Zugangsdaten?
Generic Se e
Doctrine Bundle
Sol-r Bund e
Elastic Sea Bundle
Guzzle Bundle
Client Library Sol-r
Client Library Elastic Search
Guzzle HTTP Client
Datenbank
Sol-r Server
Elastic Search Server
Generischer Webservice

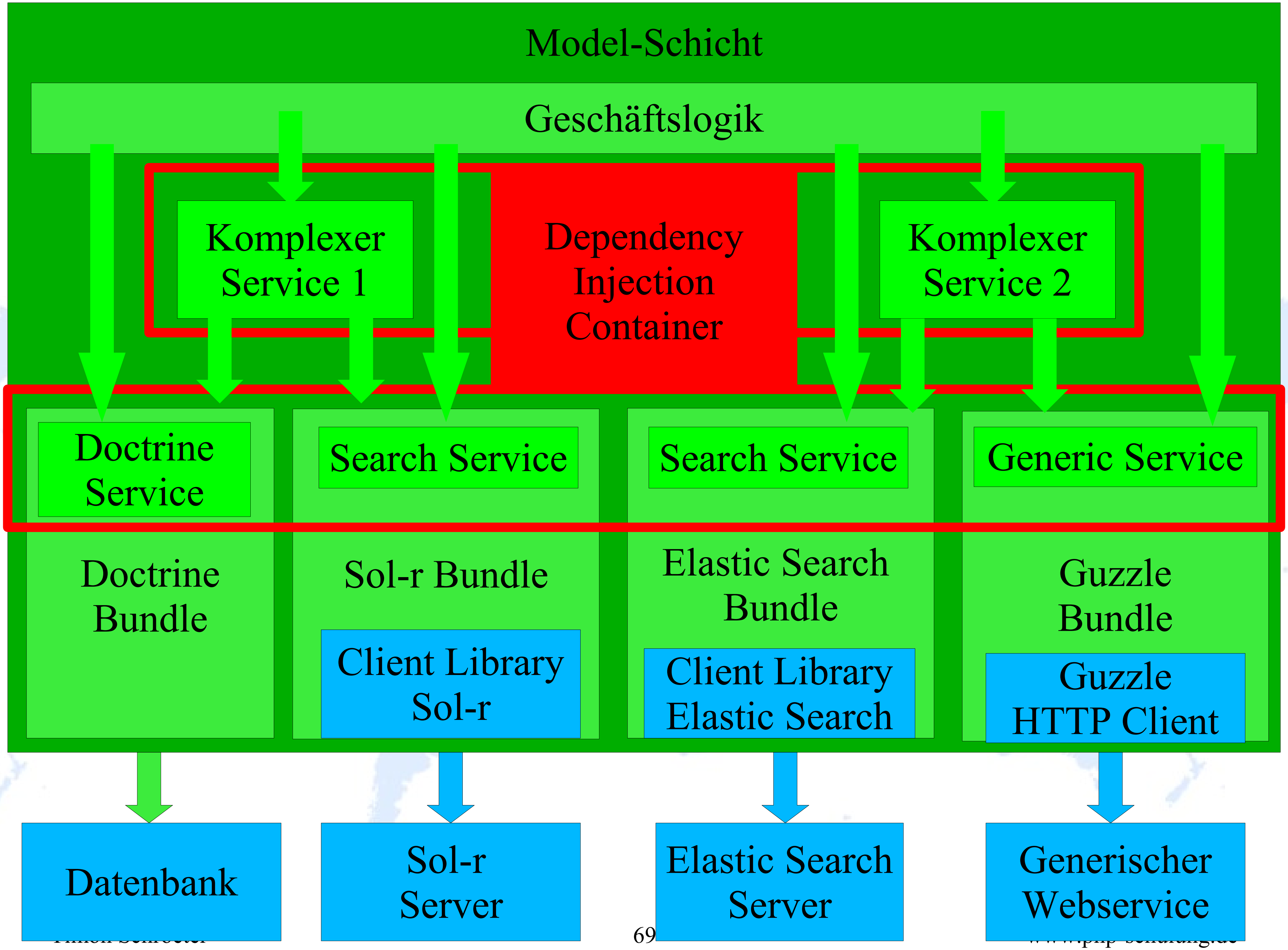

Model-Schicht
Geschäftslogik
Zugangsdaten?
Service 1
Zugangsdaten?
Komplexer Service 2
Zugangsdaten?
Doctrine Service
Search Service
Zugangsdaten?
Generic Service
Doctrine Bundle
Sol-r Bundle
Elastic Search Bundle
Guzzle Bundle
Client Library Sol-r
Client Library Elastic Search
Guzzle Client
Und so weiter ...
Braucht RAM, Instanziierung kostet Zeit
Datenbank
Server
Elastic Search Server
Generischer Webservice


Model-Schicht
Geschäftslogik
Komplexer Service 1
Dependency Injection Container
Komplexer Service 2
Doctrine Service
Search Service
Search Service
Generic Service
Doctrine Bundle
Sol-r Bundle
Client Library Sol-r
Elastic Search Bundle
Client Library Elastic Search
Guzzle Bundle
Guzzle HTTP Client
Datenbank
Sol-r Server
Elastic Search Server
Generischer Webservice

## DIC und Performance

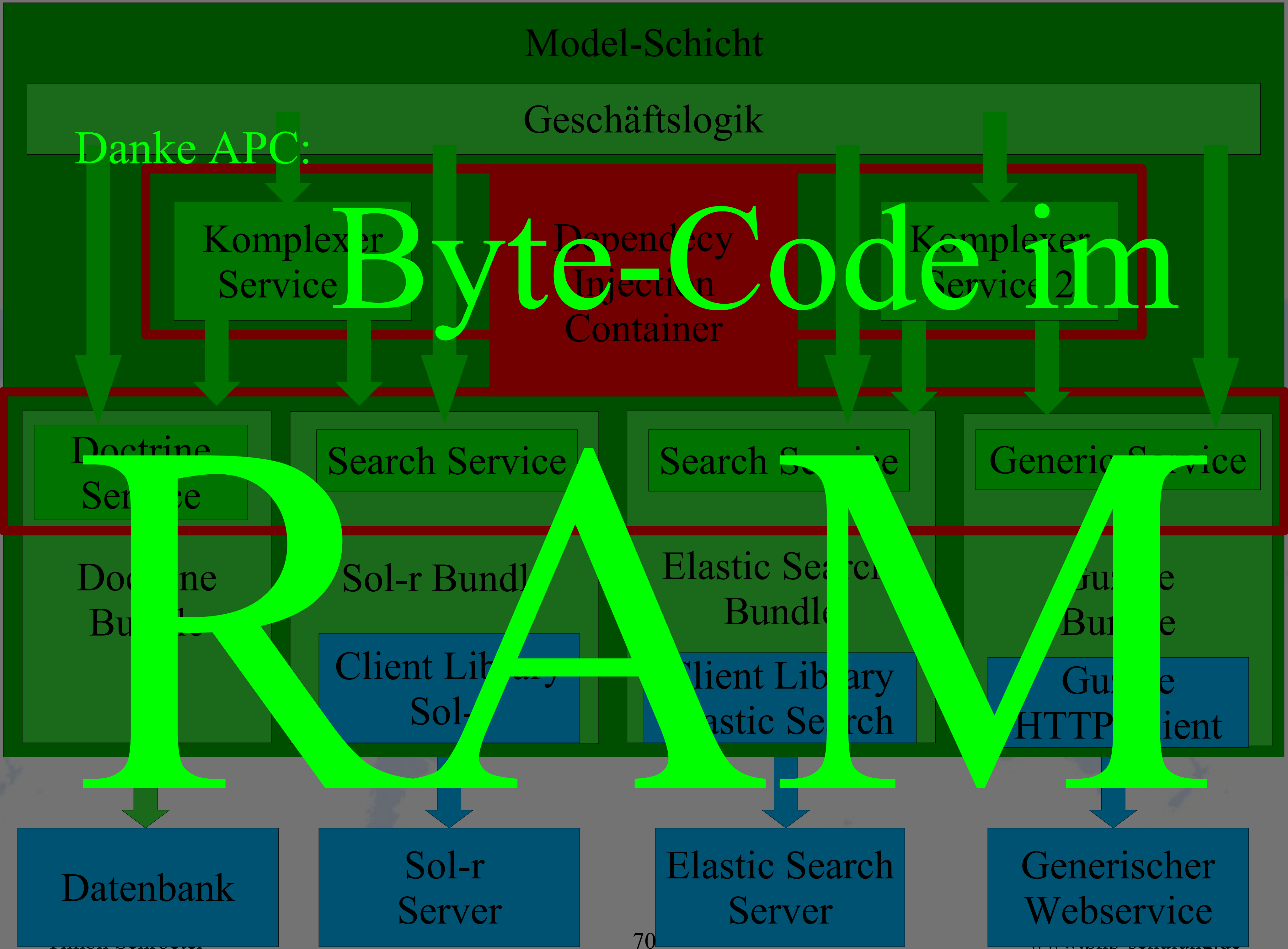
- Kompilierter Container:

```
app/cache/dev/appDevDebugProjectContainer.php
app/cache/prod/appProdProjectContainer.php
```


Model-Schicht
Geschäftslogik
Komplexer Service 1
Dependency Injection Container
Komplexer Service 2
Doctrine Service
Search Service
Search Service
Generic Service
Doctrine Bundle
Sol-r Bundle
Client Library Sol-r
Elastic Search Bundle
Client Library Elastic Search
Guzzle Bundle
Guzzle HTTP Client
Datenbank
Sol-r Server
Elastic Search Server
Generischer Webservice

Danke APC:

Byte-Code im RAM



## Summary

- Our classes only depend on interfaces
- All implementation classes are instanciated and provided (injected) by the DIC
- Our classes create only value objects and exceptions
- The DIC is not passed to any model / value class
- Controllers can access the DIC to obtain services

## 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 ...

# Thank you very much

## for your attention!

# Questions?

## Ideas, wishes, suggestions?

## I'm ready to support Your Project!

- Developer & Consultant: PHP, Symfony 2 etc.
  - www.php-entwickler-berlin.de
- Trainer & Coach: Symfony 2 workshops 1-5 days
  - www.php-schulung.de

## SOLID

- S Single Responsibility Principle
- O Open / Close Principle
- L Liskov Substitution Principle
- I Interface Segregation Principle
- D Dependency Inversion Principle