# SPRYKER

No innovation through configuration

# Our story

# Our story

## DEVELOPMENT

- Ambitious commerce projects — zalando
- Appropriate software not available
- Internal framework solution
- 100+ online business models launched
- November 2014: Spryker is founded

## Spryker roots

dafiti .com.br
Category Leader for Fashion in Brazil

Lamudi
Real Estate Marketplace 33 countries in 15 months

WESTWING HOME AND LIVING
Largest Furniture Shopping Club in Europe

Delivery Hero
Meal Delivery in EU and SE Asia, $740 Mio. GMV

HelloFresh
Largest Meal Ingredients delivery in EU, 380% YTY growth

CONTORION ALLES FÜR DEN PROFI.
B2B Webshop with complex business logics

# Spryker is based on a proven architecture

## Spryker 0.5 (A&B)

dafiti

WESTWING
HOME AND LIVING

HelloFresh

## Spryker 1.0 (Y&Z)

CONTORION

TIRENDO

natue

## Spryker 2.0

PROVEN ARCHITECTURE

# Outline

- Established commerce software developed within real-world projects
- Agile framework relevant for ambitious players &non-standard models
- Best practices of 100+ implementations went into Spryker with a focus on
  - agility
  - speed
  - innovation
  - business intelligence

# Spryker solves a lot of common issues

| Common issued of shop software | | Solutions integrated in Spryker | |
|---|---|---|---|
| Full-page cache | > | Separation of frontend and backend |  |
| Website draws from database | > | Frontend draws from Key Value Storage |  |
| Unnecessary code components & features | > | Lean code base and modular features |  |
| Entity Attribute Value (EAV) model | > | Extend DB Schema + JSON values |  |
| Monolithic approach | > | Modular - monolith |  |

# The market

# New market rules in eCommerce

- New players with strong online skills outperform the market
- Technology has become a key differentiator

**Speed**
constant change requires a new level of speed and flexibility

**Innovation**
today, users are at the center of entire business strategies

**Business intelligence**
real knowledge of the customer is key to offering attractive products

**Ability to execute**
companies may have good ideas, but their IT is often a bottleneck

# Successful (new) groups

| Online Pure Players | Category Leaders | Unique Business Models |
| --- | --- | --- |
| Zappos.com | wayfair | UBER |
| blue nile | INDOCHINO | airbnb |
| FARFETCH | Toys"R"Us | MUNCHERY |
| GILT | zalando | WARBY PARKER |

Digital transformation + non-standard models like banks, insurances, B2B, IoT, …

# Successful (new) groups

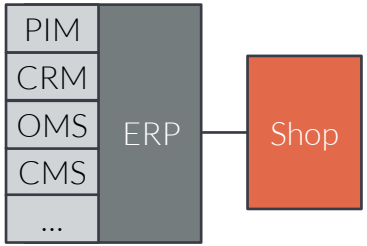
| Online Pure Players | Category Leaders | Unique Business Models |
| --- | --- | --- |
| ao.com | zooplus | airbnb |
| ABOUT YOU | eventim | vente-privee |
| home 24 | WESTWING HOME AND LIVING | Delivery Hero |
| CONTORION ALLES FÜR DEN PROFI. | zalando | TIRENDO |

Digital transformation + non-standard models like banks, insurances, B2B, IoT, …

# eCommerce Technology matures

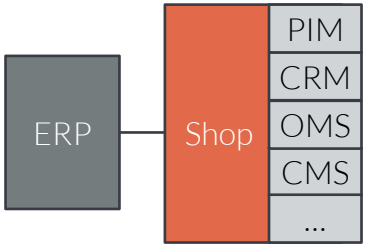
## 1st Generation
## ERP-focused setup

*eCommerce =*
*additional sales channel*



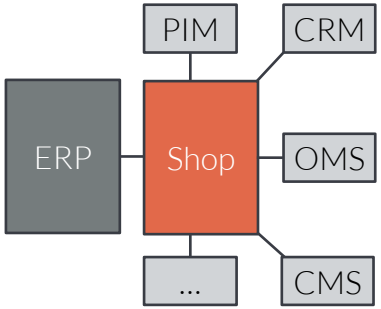

## 2nd Generation
## Features move into Shop

*eCommerce =*
*new business model*




## 3rd Generation
## Features move out in Ecosystem

*eCommerce =*
*part of an ecosystem*




+ channels
+ devices
+ complexity
+ content
+ uncertainty

1995 - 2002

2003 - 2013

2014 +

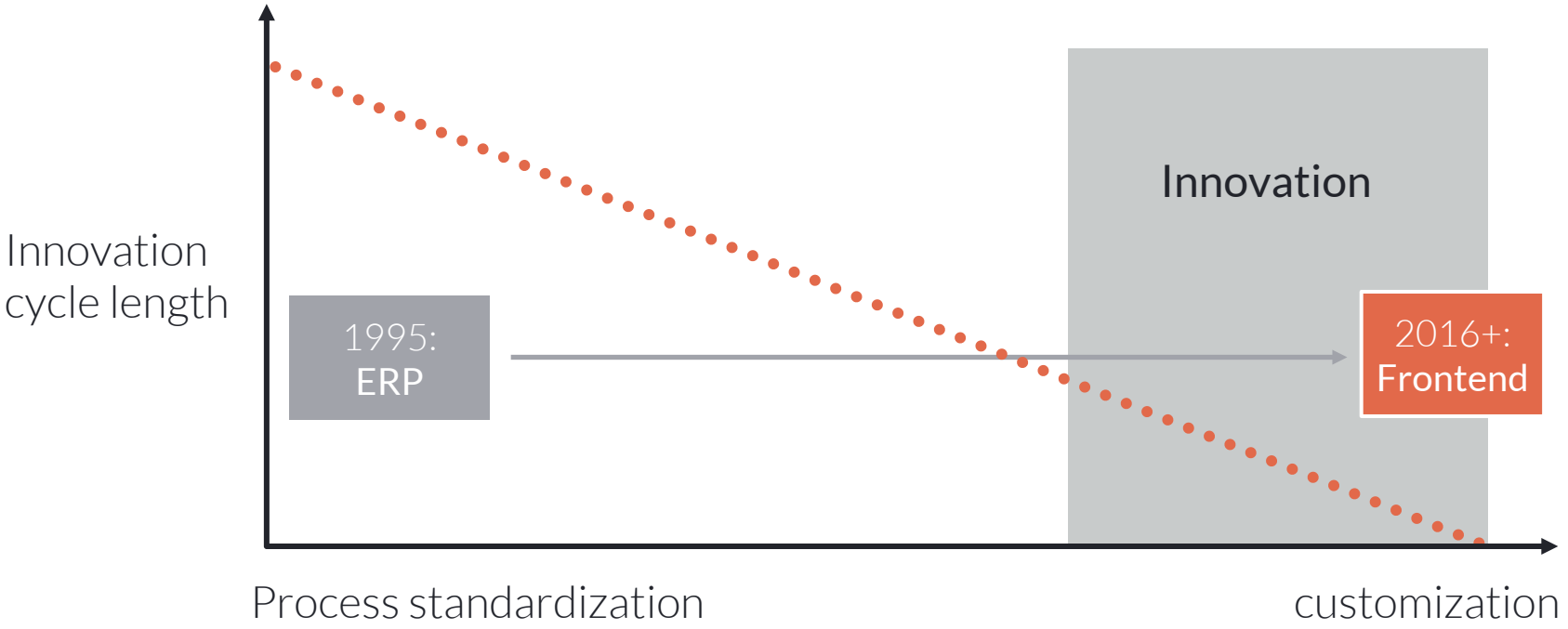
atg
hybris
WebSphere.
INTERSHOP

demandware
Magento
Open Source eCommerce

commercetools

SPRYKER

# Customer centric time creates uncertainty



# Sophisticated leaders own their technology

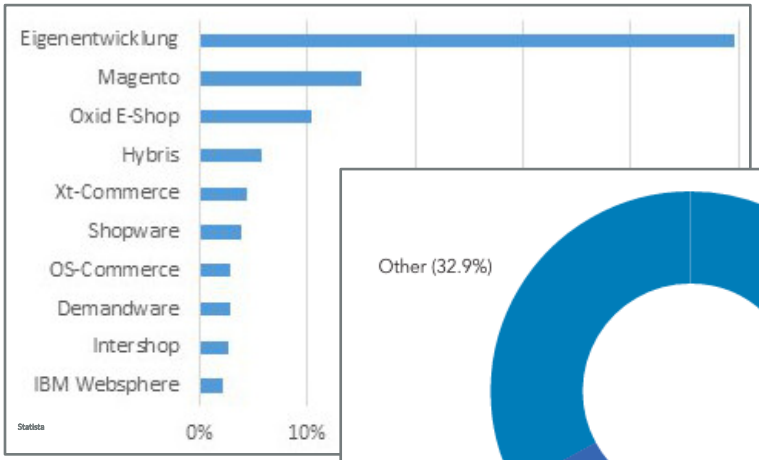
| E-Commerce Value Chain: | amazon | Alibaba.com | wayfair | zalando | Etsy | home 24 | ABOUT YOU |
|---|---|---|---|---|---|---|---|
| | $ 89 bn* | $ 12 bn | $ 1.3 bn | € 2.2 bn | $ 1.9 bn | € 160 m | € 100 m |
| Shop | ⌂ | ⌂ | ⌂ | ⌂ | ⌂ | ⌂ | ⌂ |
| OMS | ⌂ | ⌂ | ⌂ | ⌂ | ⌂ | ⌂ | ⌂ |
| Warehousing | ⌂ | | ⌂ | ⌂ | | ⌂ | |
| Payment | ⌂ | ⌂ | | | | | |
| Logistics | ⌂ | | | | | | |
| Hosting | ⌂ | ⌂ | | ⌂ | ⌂ | | |

*online revenue in 2014

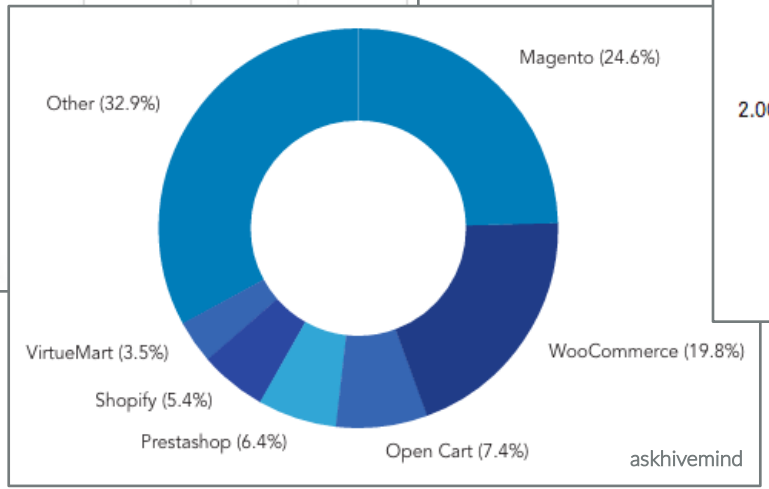
# Shop system usage

Although there are lots of shop systems, many merchants decide to build their own? **Why?**

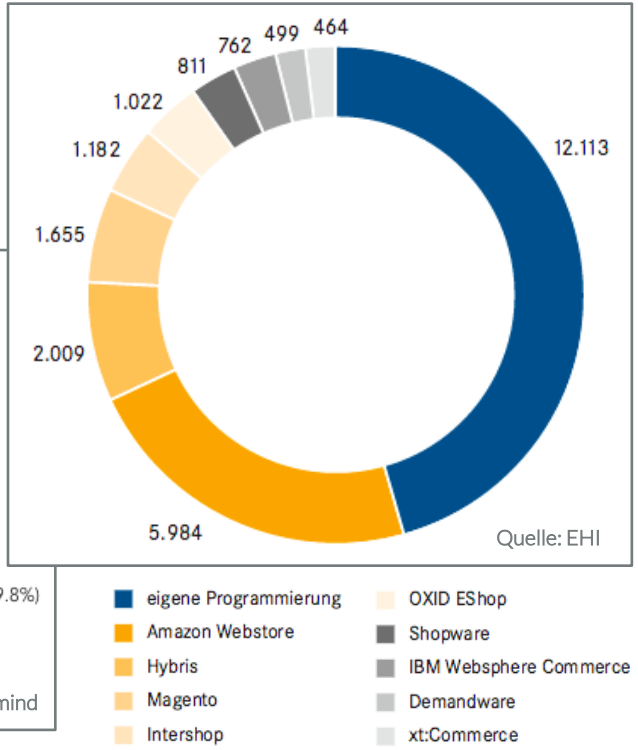
2014 – Shopsystems Germany




Top 50K
Top 1M
Worldwide




Total revenue, in Mio. EUR



# Standard Shop-software

Standardized shop software offer many features that can be configured and some hooks for extensions.

➔ Enough for the long tail of merchants.

But …

# Framework vs. Shop-software

Standardized shop software offer many features that can be configured and some hooks for extensions.

➔ Enough for the long tail of merchants.

But that doesn't fit for ambitious projects!

If your business model is not standardized, a standard shop software does not help you!

# Why standard software is often no option

There is no standard software for a non-standard business.

- Hard to **remove components** from off-the-shelf solutions
  e.g. monolithic Magento system
- Hard to **integrate into not supported environments**
  e.g. closed sourced environments & Hybris
- Hard to create **new features**
  e.g. can't run full stack on developer machine at Demandware and other SaaS solutions

> *Retailers want control over their destiny. They need to be able to make changes quicker and innovate faster. By **bringing the e-Commerce technology in-house** they will be better able to do this — versus being beholden to a legacy platform.*
>
> *Internet Retailer, July 2015*

# The framework approach

Instead of an out-of-the-box shop software, ambitious projects need a **framework** that optimizes developers' productivity.

---

How?

# Increased developer's productivity

- ✔ Clean and SOLID code
- ✔ Consistent software design
- ✔ Strict modularization
- ✔ Generic set of features (like state machines)
- ✔ High performance and scalability
- ✔ Deterministic behavior
- ✔ Tested and measured code
- ✔ Avoidance of bad practices

Like full page cache, EAV, event-dispatcher, AOP, code configuration in database, dependency-container magic, …

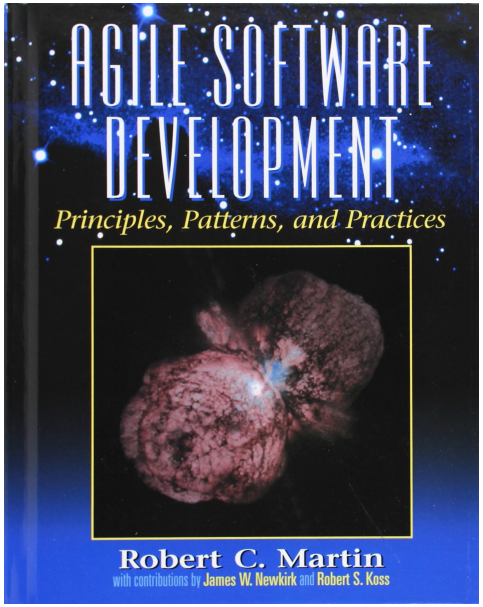
SPRYKER

# Good software matters

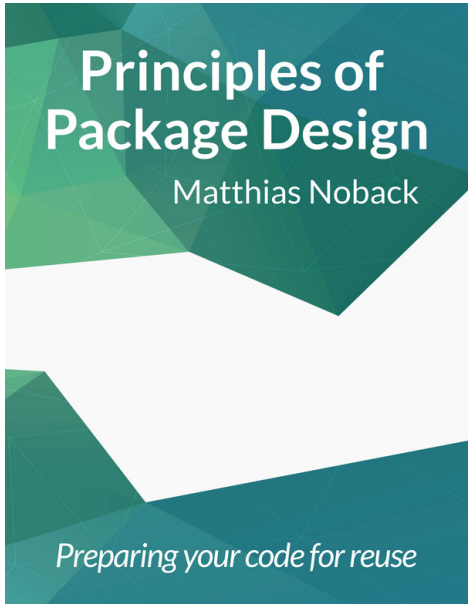
# Good software principles

SPRYKER

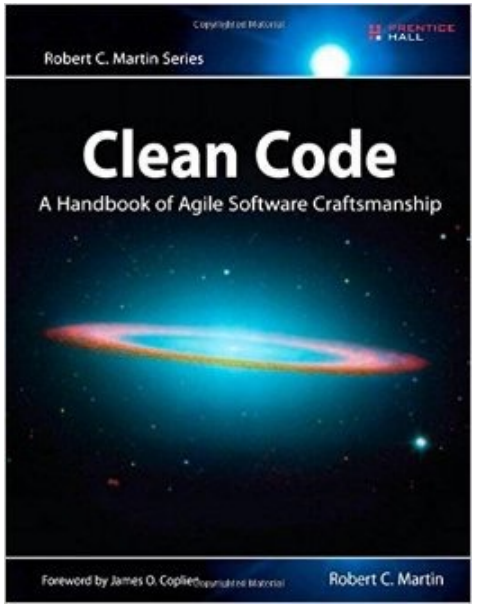
## SOLID




## Package design




## Clean code

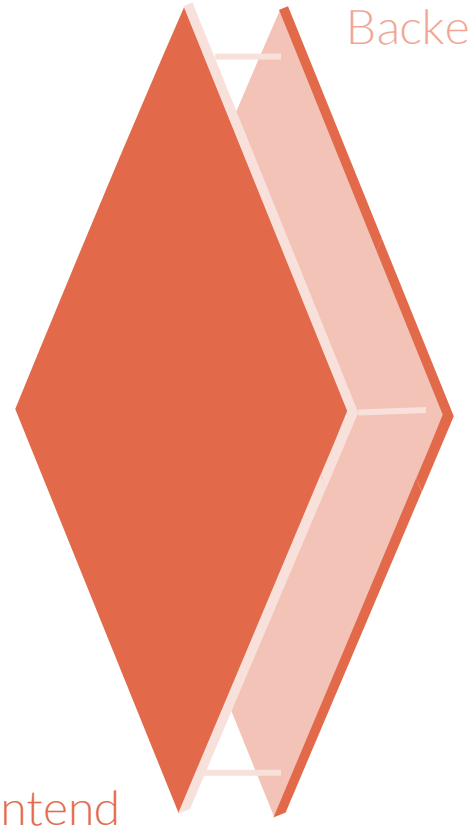


It is not enough for code to work.

*Robert C. Martin (uncle Bob)*

# Spryker architecture

# First sketch – Frontend & Backend



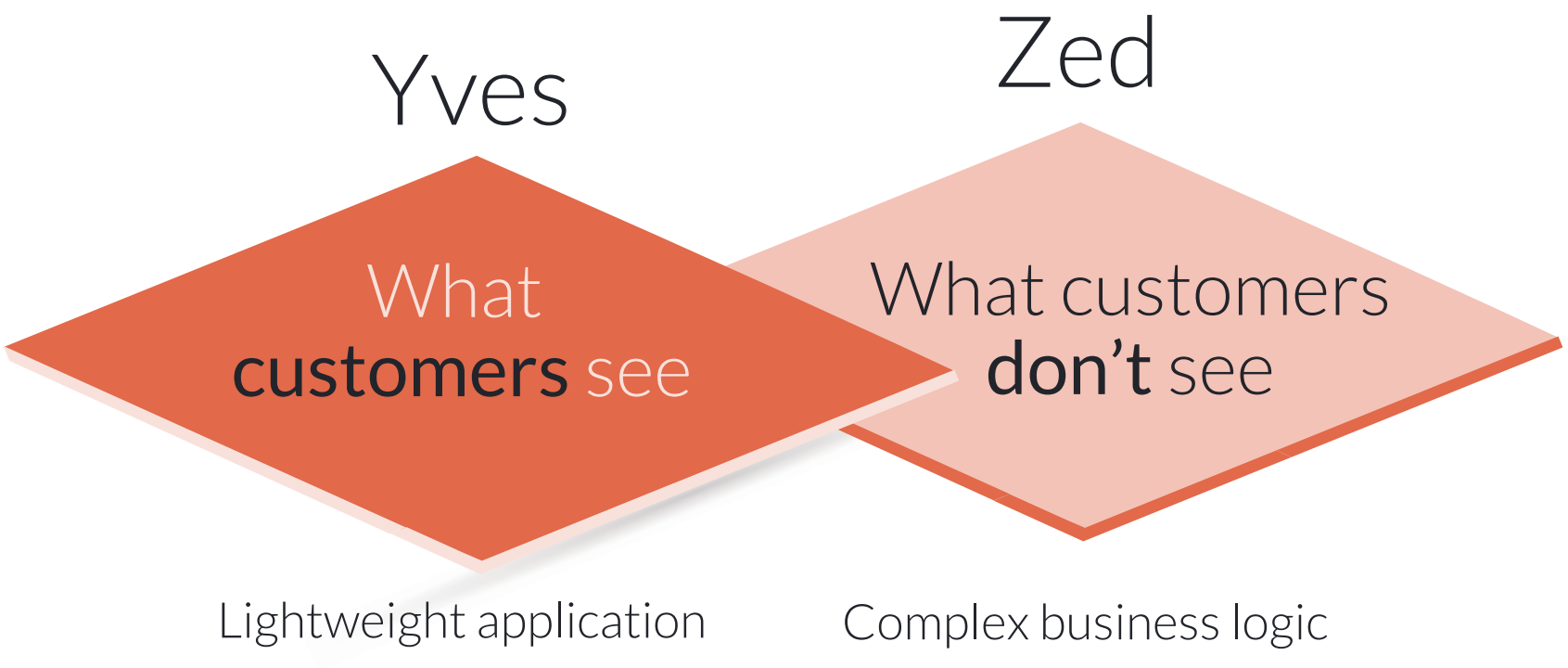

High performance

Scalability

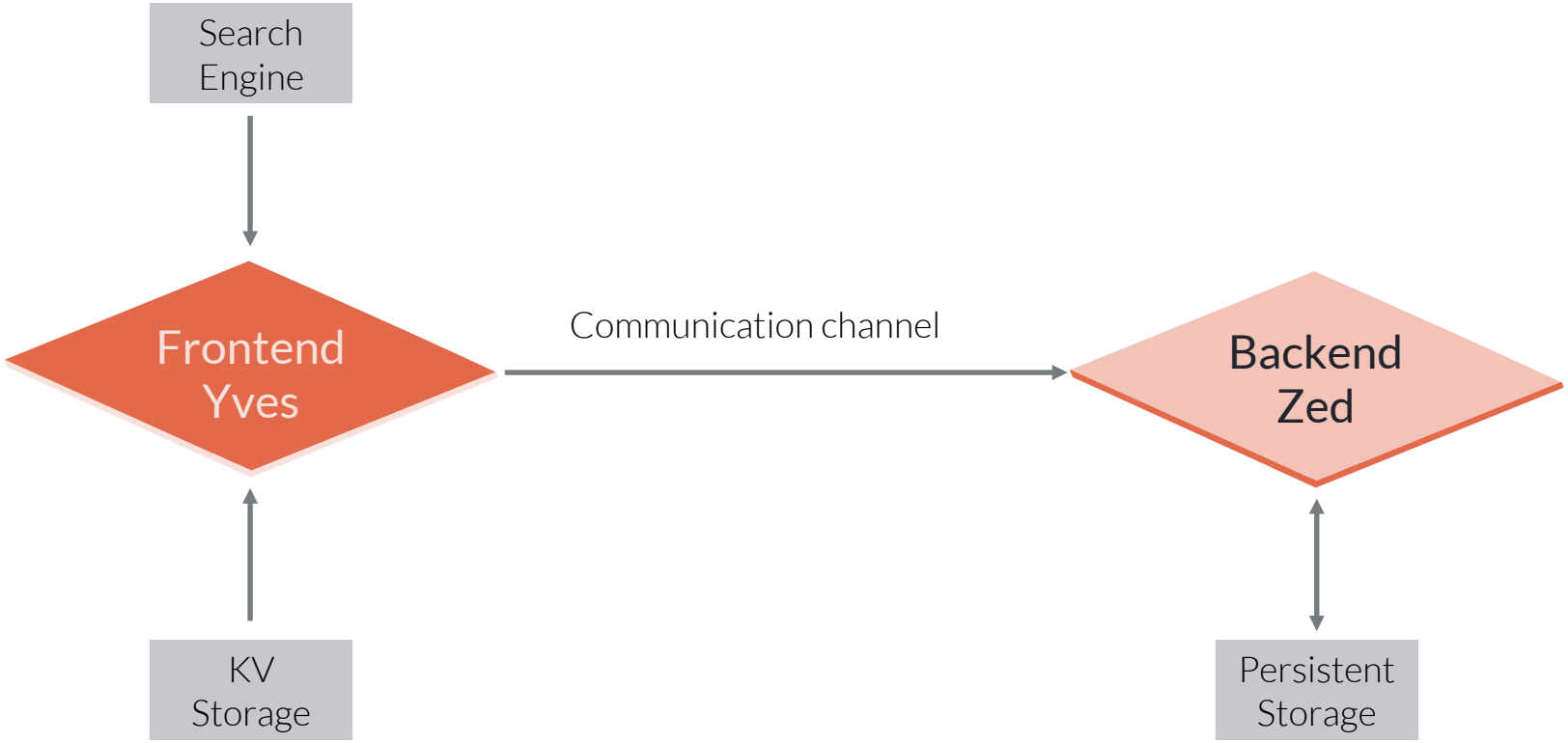
Complex logic

Lot of data to persist

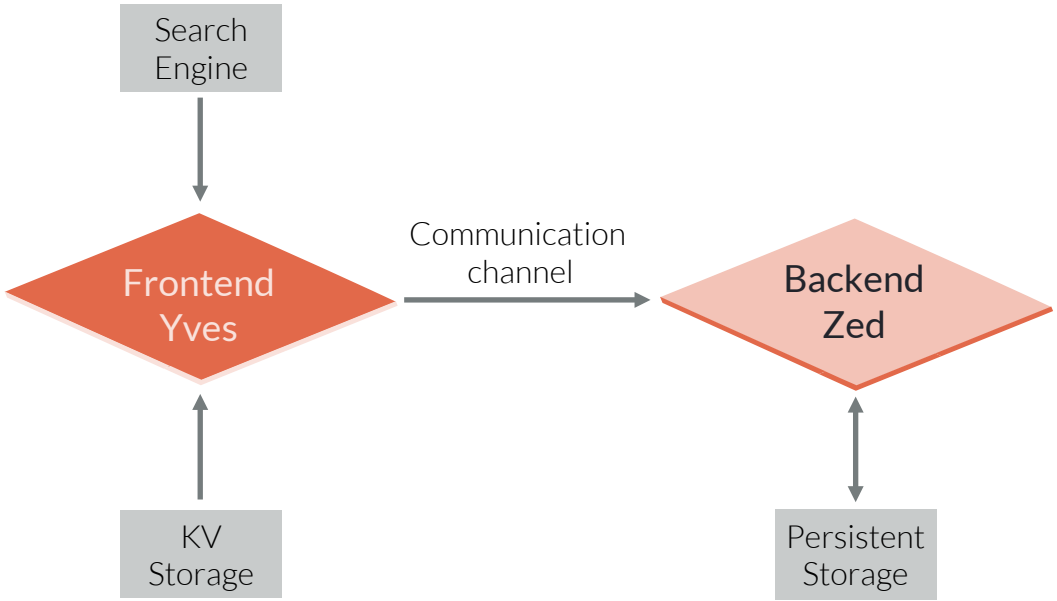
# First sketch – Frontend & Backend



# Frontend & Backend

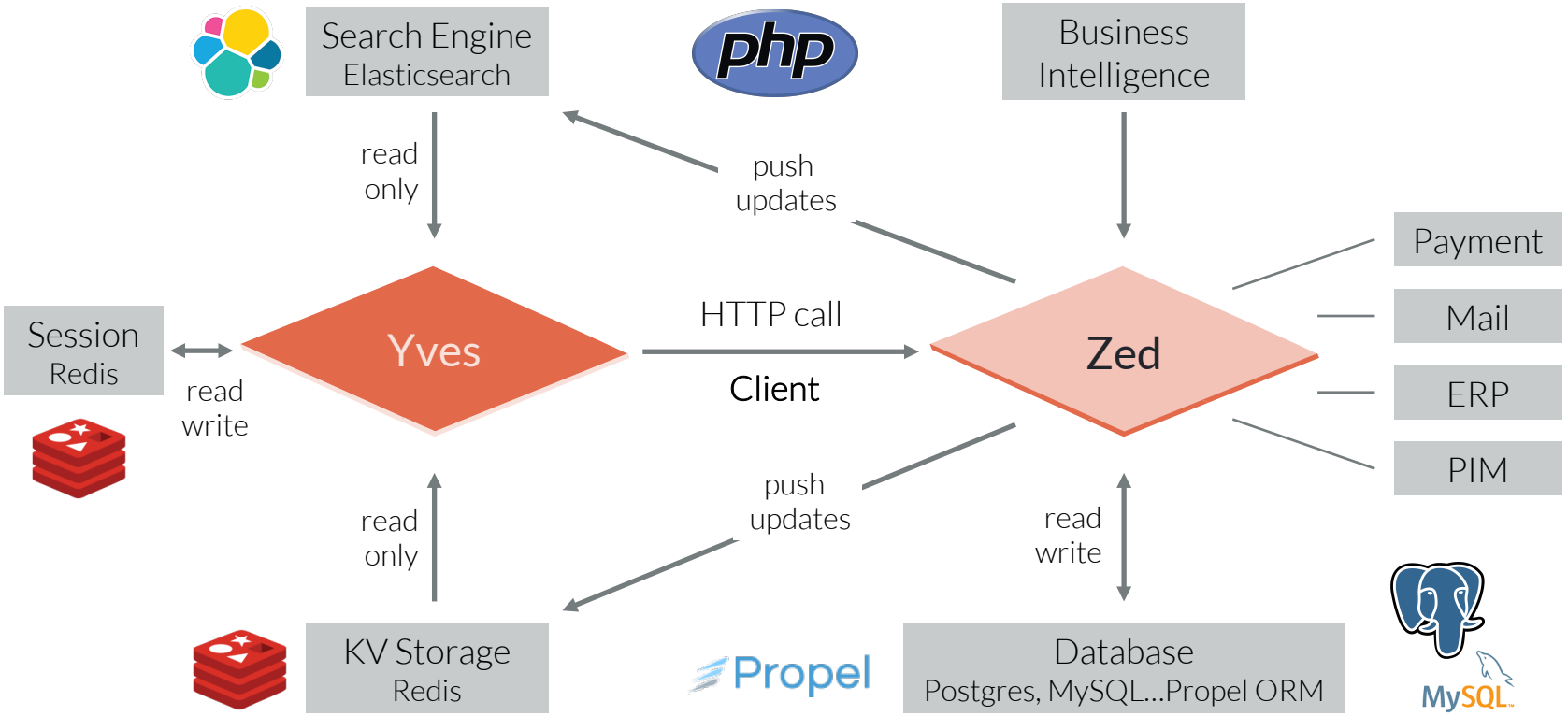


# Frontend & Backend

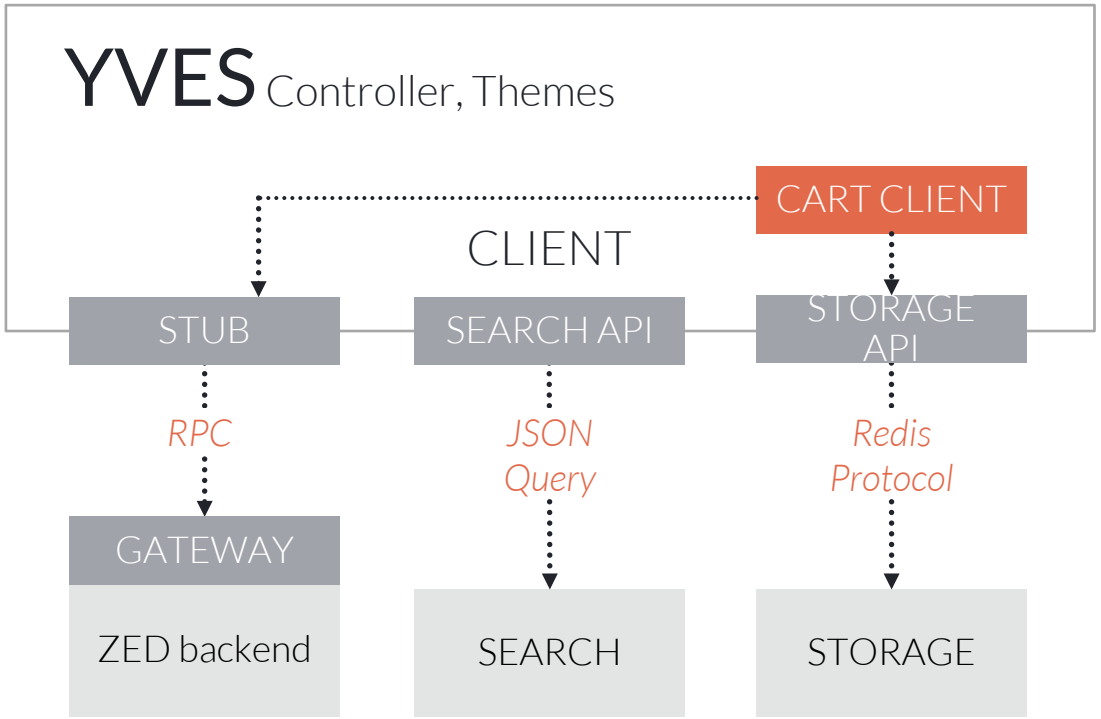



- Clear responsibilities
- No dynamic-page-caching
- No direct access to per-sistence from frontend
- KV-storage is easily scalable
- Boost performance
- < 50ms

# Yves & Zed

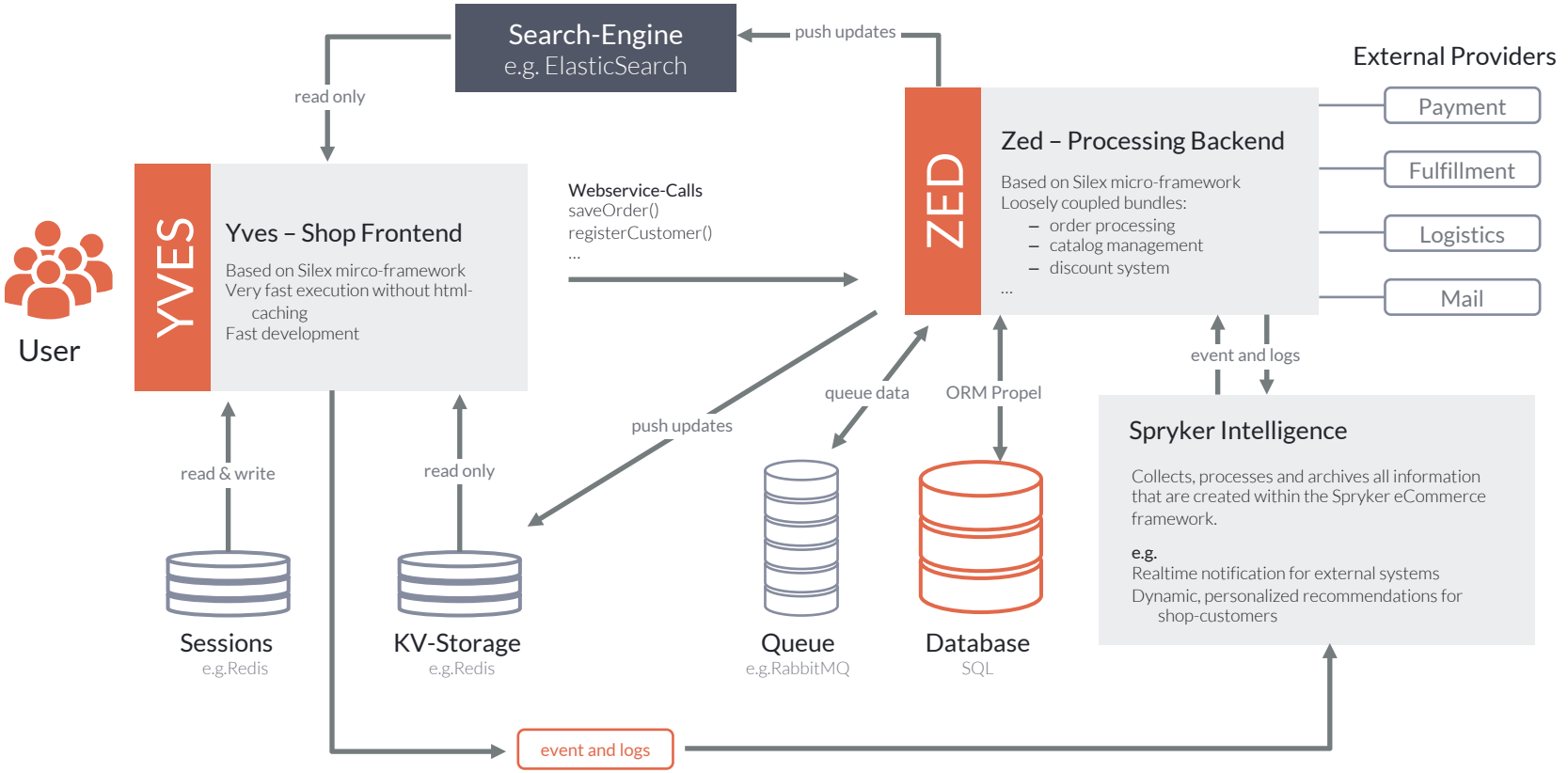

Search Engine
Elasticsearch
php
Business
Intelligence
read
only
push
updates
Payment
Mail
ERP
PIM
Session
Redis
read
write
Yves
HTTP call
Client
Zed
push
updates
read
write
read
only
KV Storage
Redis
Propel
Database
Postgres, MySQL...Propel ORM
MySQL

# Yves Client

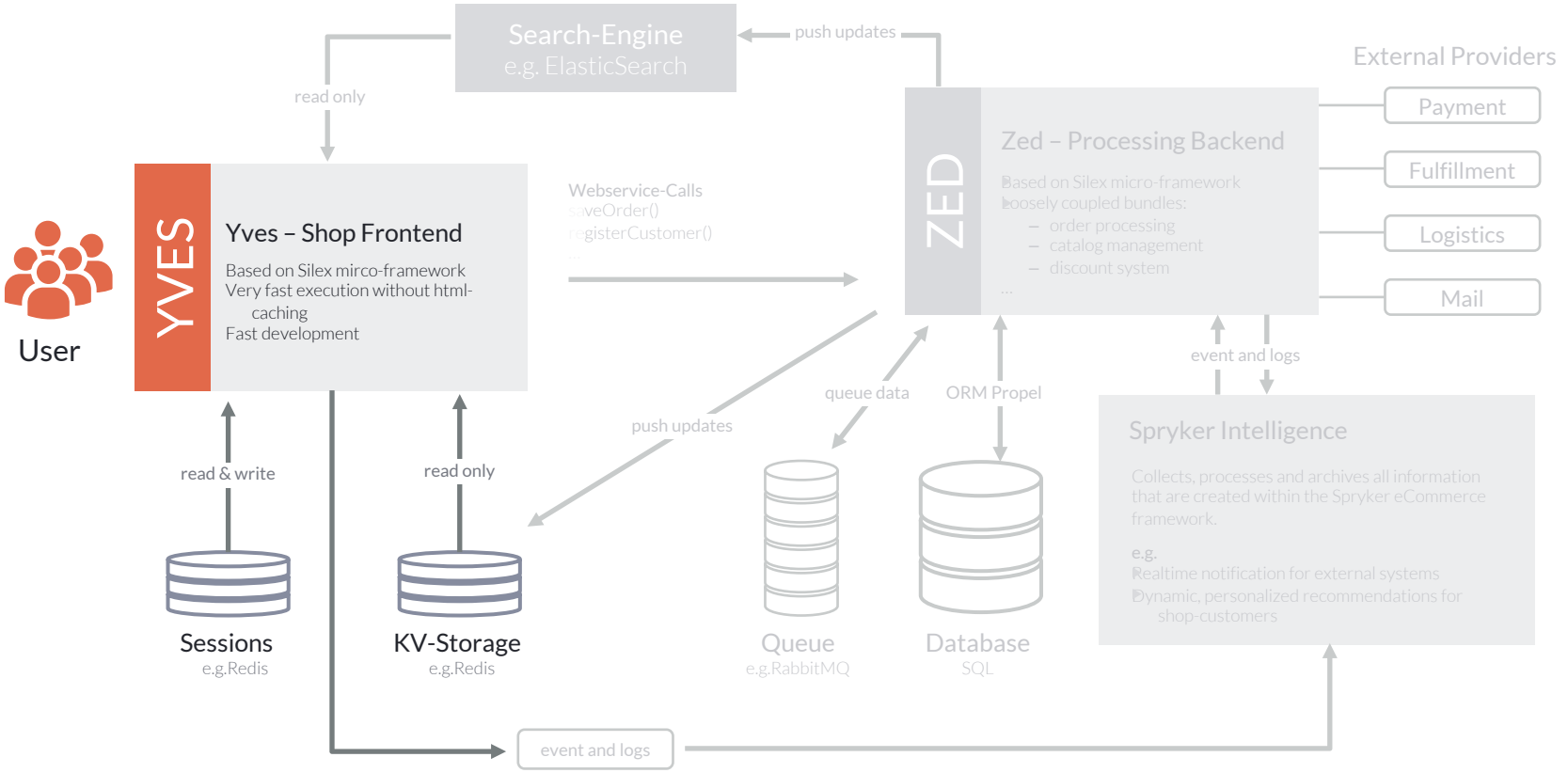



- Yves only consists of templates, controllers, service providers
- Possible to build another Yves with another framework and still use Spryker
- Client provides core functionnality for the shop frontend
- Client acts as a facade & delegates calls to Zed, Search & Storage

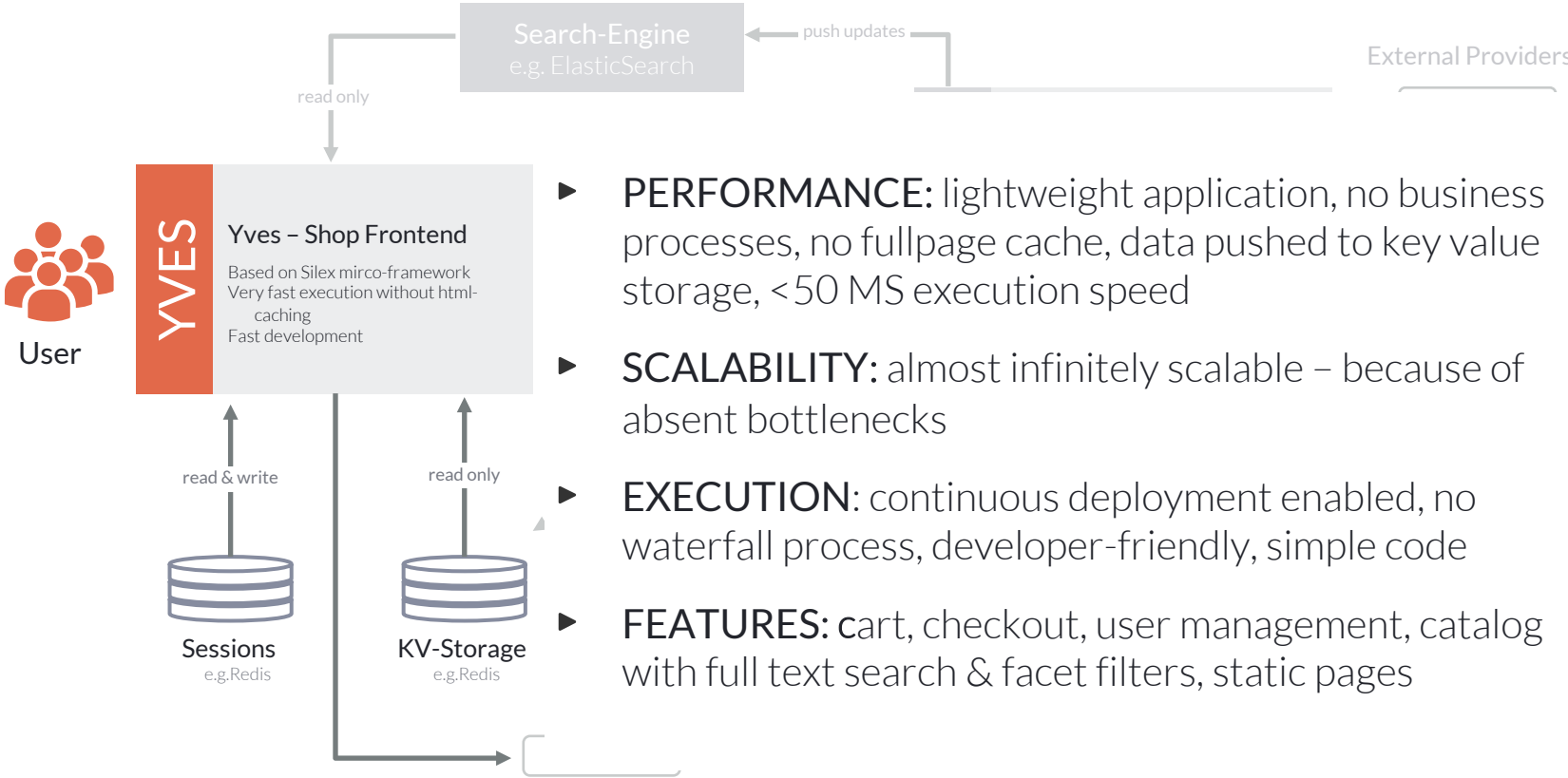
# Technology Architecture



# Technology Architecture - Frontend

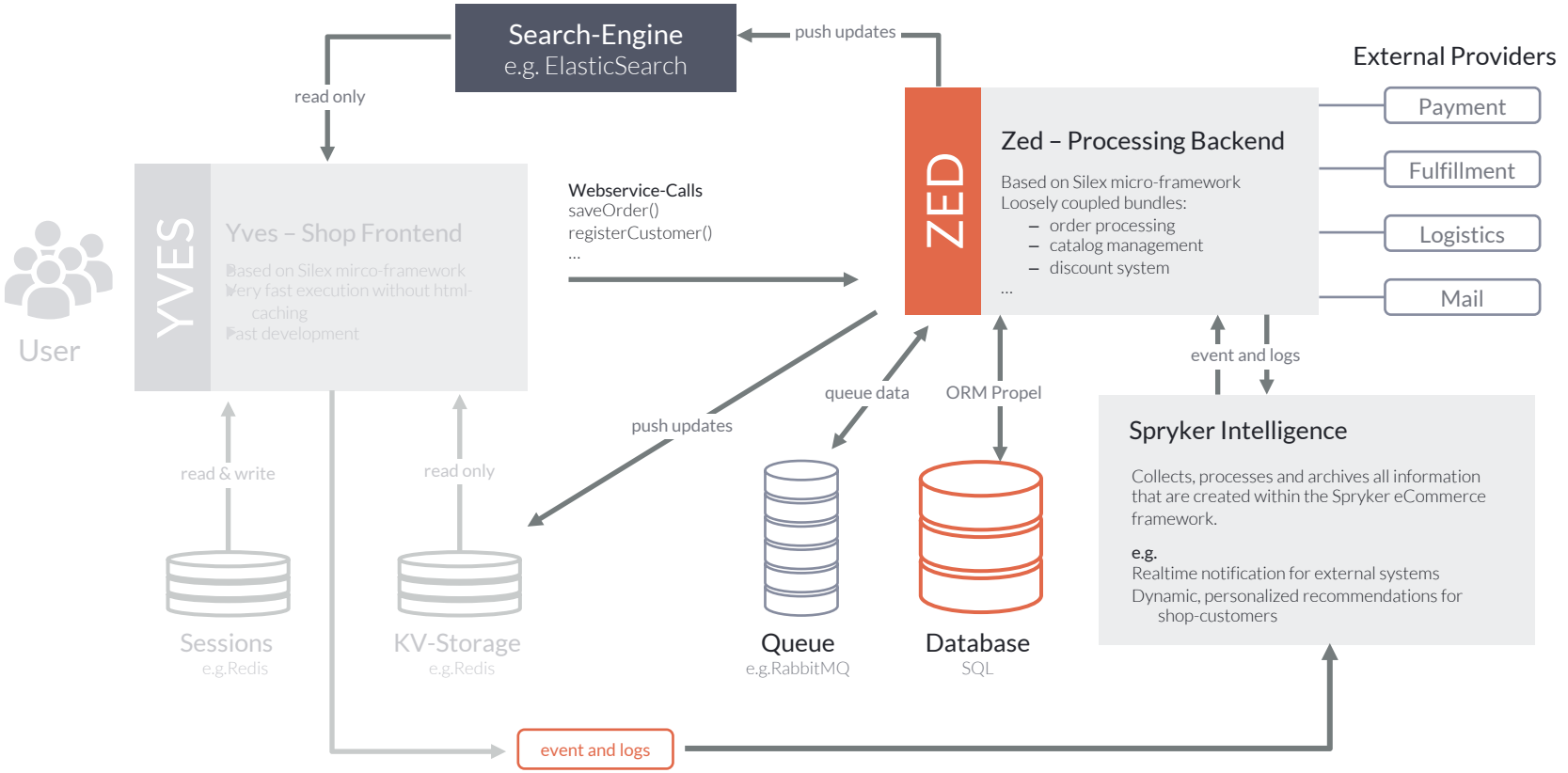


# Technology Architecture - Frontend



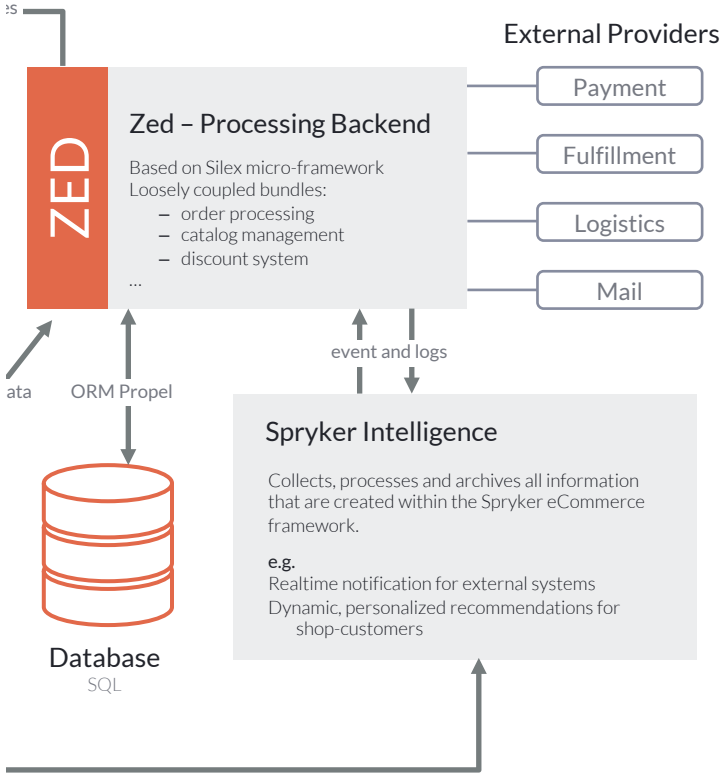

- **PERFORMANCE:** lightweight application, no business processes, no fullpage cache, data pushed to key value storage, <50 MS execution speed
- **SCALABILITY:** almost infinitely scalable – because of absent bottlenecks
- **EXECUTION**: continuous deployment enabled, no waterfall process, developer-friendly, simple code
- **FEATURES:** cart, checkout, user management, catalog with full text search & facet filters, static pages

# Technology Architecture - Backend

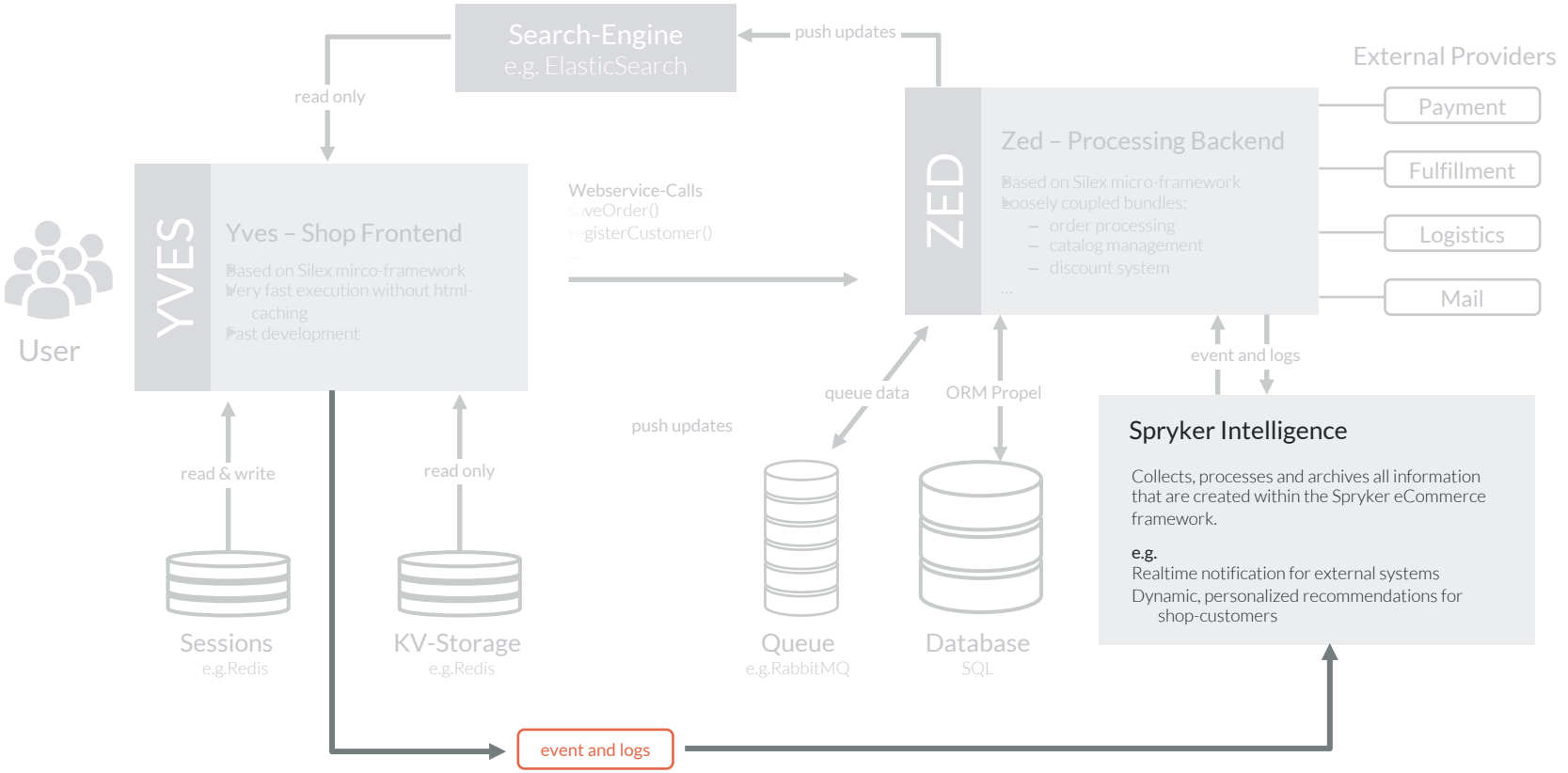


# Technology Architecture - Backend

- **BUSINESS LOGIC:** handles complex business processes, infinitely scalable
- **DATA MGMT:** communication hub
- **INTEGRATE:** external providers like Payment, Logistic, PIM, …
- **FEATURES:** e.g. admin GUI, product management, content management, discounts, order management, ...
- **ORDER MGMT**: as part of the shop, separation of concerns possible, state machine

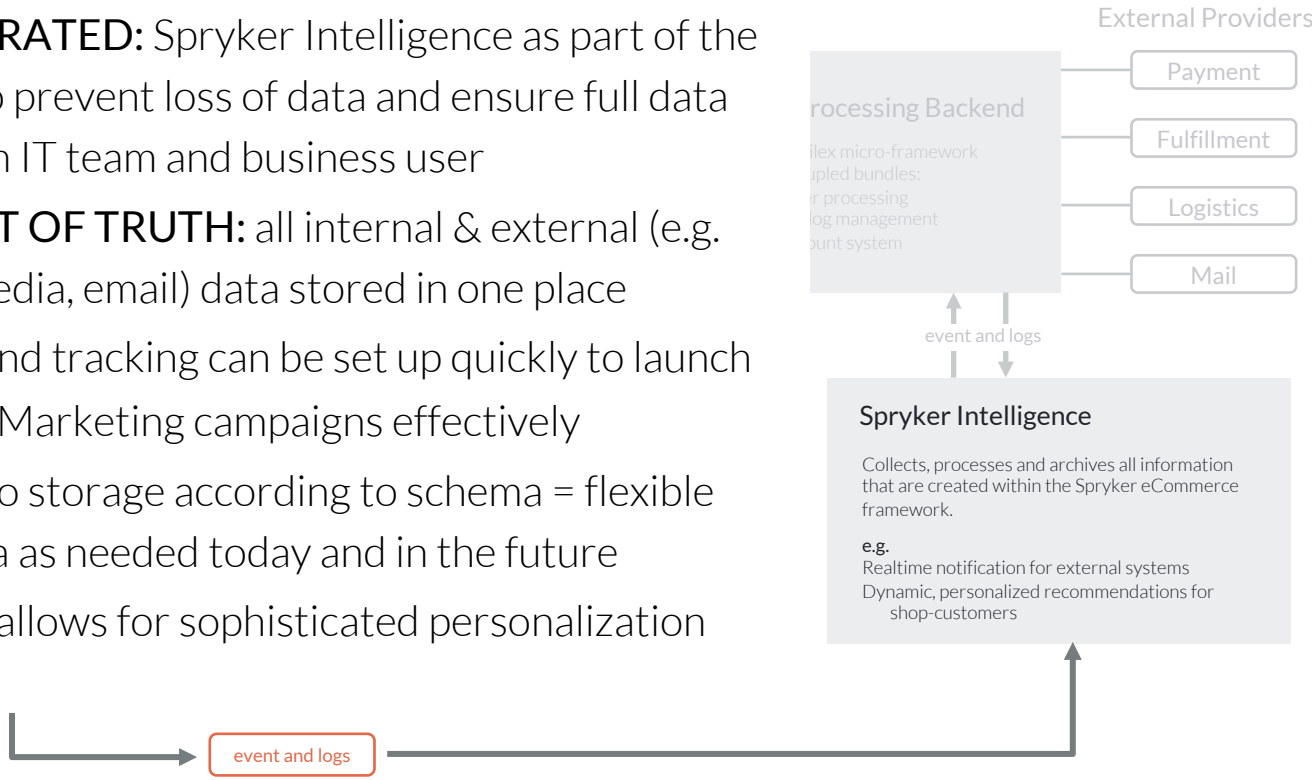


# Technology Architecture – BI (Draft)



# Technology Architecture – BI (Draft)

- **FULLY INTEGRATED:** Spryker Intelligence as part of the shop system to prevent loss of data and ensure full data insight for both IT team and business user
- **SINGLE POINT OF TRUTH:** all internal & external (e.g. CRM, social media, email) data stored in one place
- **SPEED:** KPIs and tracking can be set up quickly to launch and track new Marketing campaigns effectively
- **DATALAKE:** no storage according to schema = flexible analysis of data as needed today and in the future
- **EXECUTION:** allows for sophisticated personalization



# Building Blocks - Technologies

Spryker uses conservative state of the art technologies









# Architectural Thinking

## Single application approach

Runs on every laptop


“Just a PHP application”

Quick and simple deployment

## Service approach

High productivity with large teams with big projects

Update, replace or extend parts of the application

# Architectural thinking

## Single application approach

Runs on every laptop


"Just a PHP application"

Quick and simple deployment

## Service approach

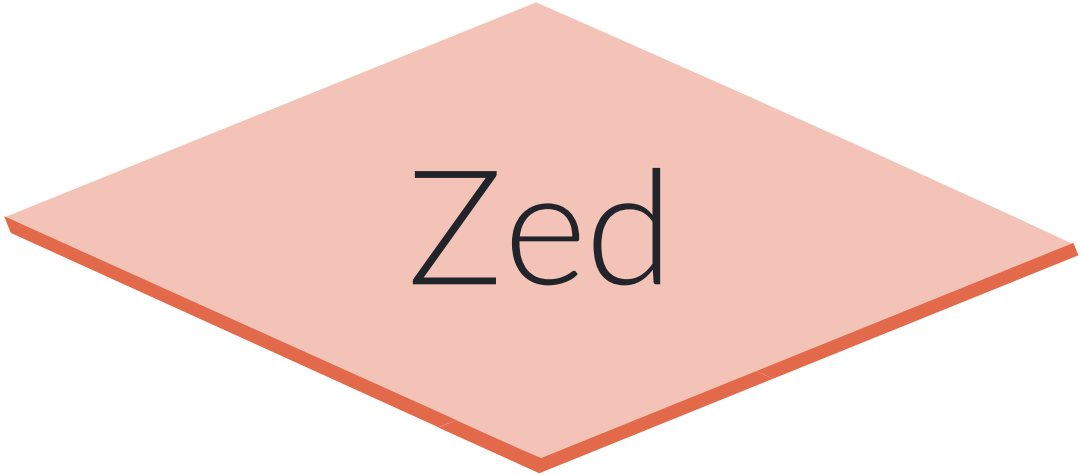
High productivity with large teams with big projects

Update, replace or extend parts of the application

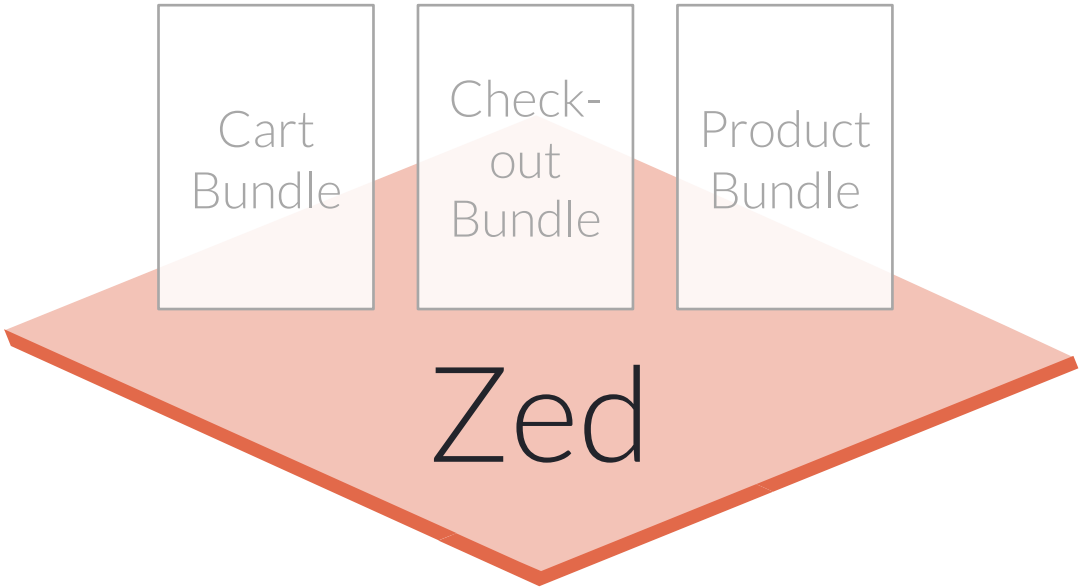
## Not wanted:

- Technology zoo
- Overhead of distribution and asynchronous communication
- Multiple database without transactions and consistency

# Monolith?

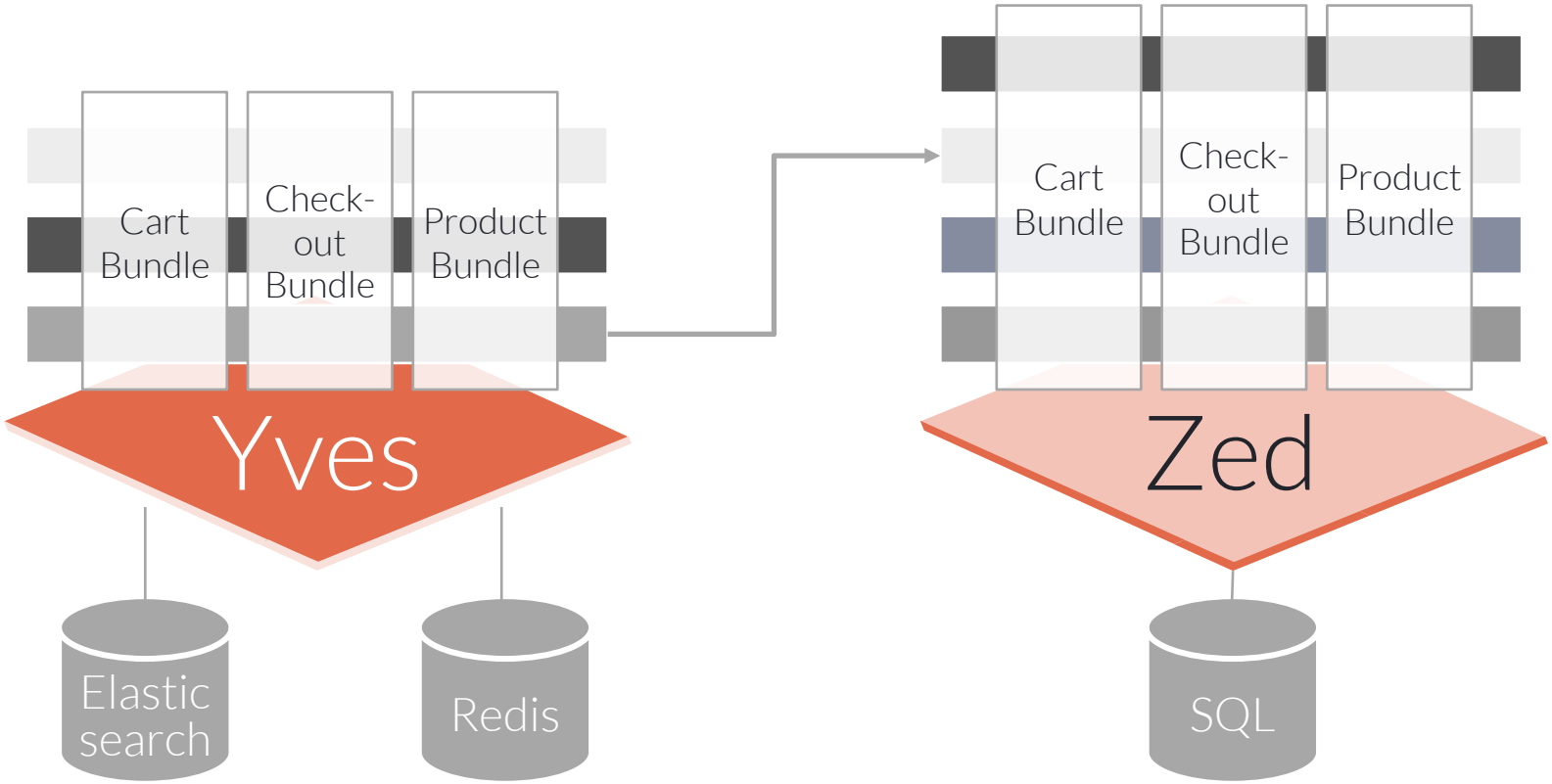


# Monolith?




A bundle is a functional unit which has a single responsibility and well-defined relationships.

# Monolith vs. Modular



# Monolith vs. Modular


| Monolith approach | | Modular approach |
|---|---|---|
| ✗ | Release / reuse | ✓ |
| ✓ | Simple to deploy | ✗ |
| ✗ | High productivity for teams | ✓ |
| ✓ | Scaling | ✓ |
| ✓ | Performance | ✓ |

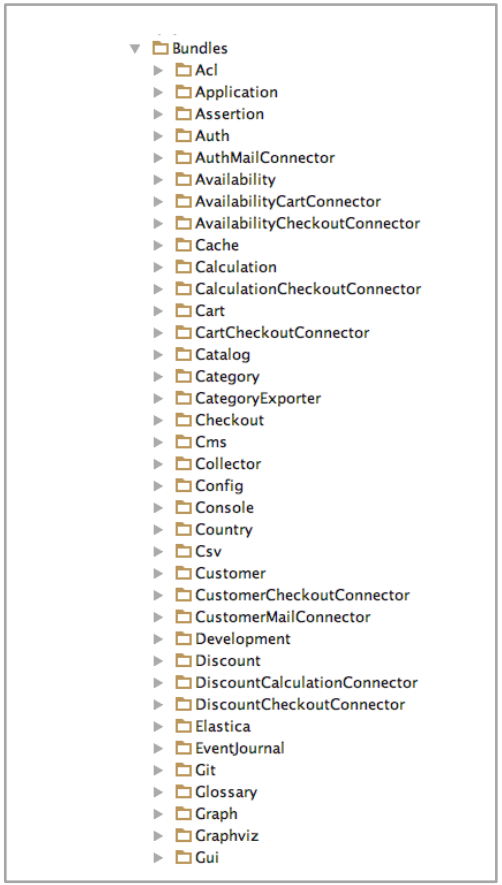
# Spryker Bundles

| | | | |
|---|---|---|---|
| Acl | Country | Oms | Search |
| Auth | Discount | Payment | Session |
| Availability | Distributor | Price | Shipment |
| Cart | Glossary | Product | Stock |
| Catalog | Invoice | Product Category | Storage |
| Checkout | Kernel | Product Option | Url |
| CMS | Locale | Queue | User |
| Customer | Mail | Sales | Wishlist |

- **All important features**: Spryker comes with 100+ bundles
- **Easy and fast updates**: Each bundle has its own version
- **Consistent**: Each bundle has the same basic architecture
- **Flexible & adaptable**: Bundle dependencies are reduced to a minimum
- **Splitted:** Every bundle is a functional unit with explicit dependencies
- **Coherent:** Bundles are decoupled & single respon-sibility

# Strict modularization

- A bundle is a “functional unit”.
- Bundles are loose coupled, coherent and have explicit dependencies.
- They follow the packaging principles!
- Currently there are > 100 bundles.
- Main benefit: Code keeps well-structured even in big projects with large development teams.



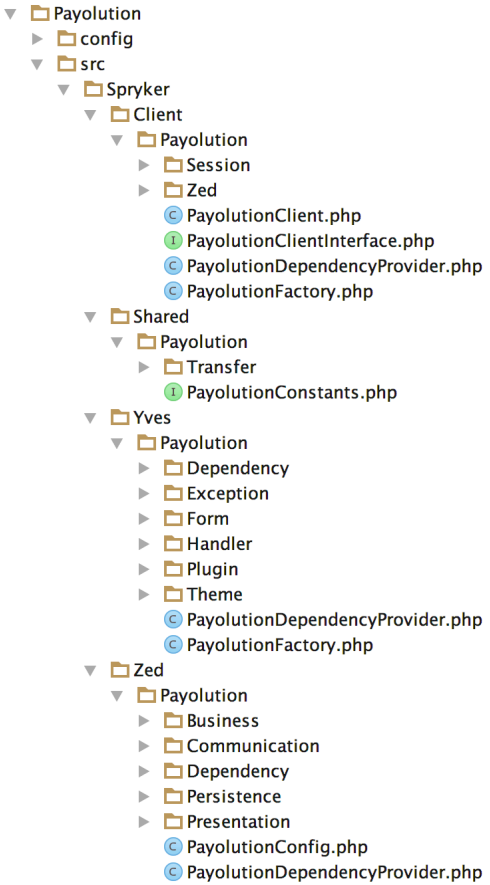
# Bundle Structure

**Client**
Yves to Zed communication
Managing sessions

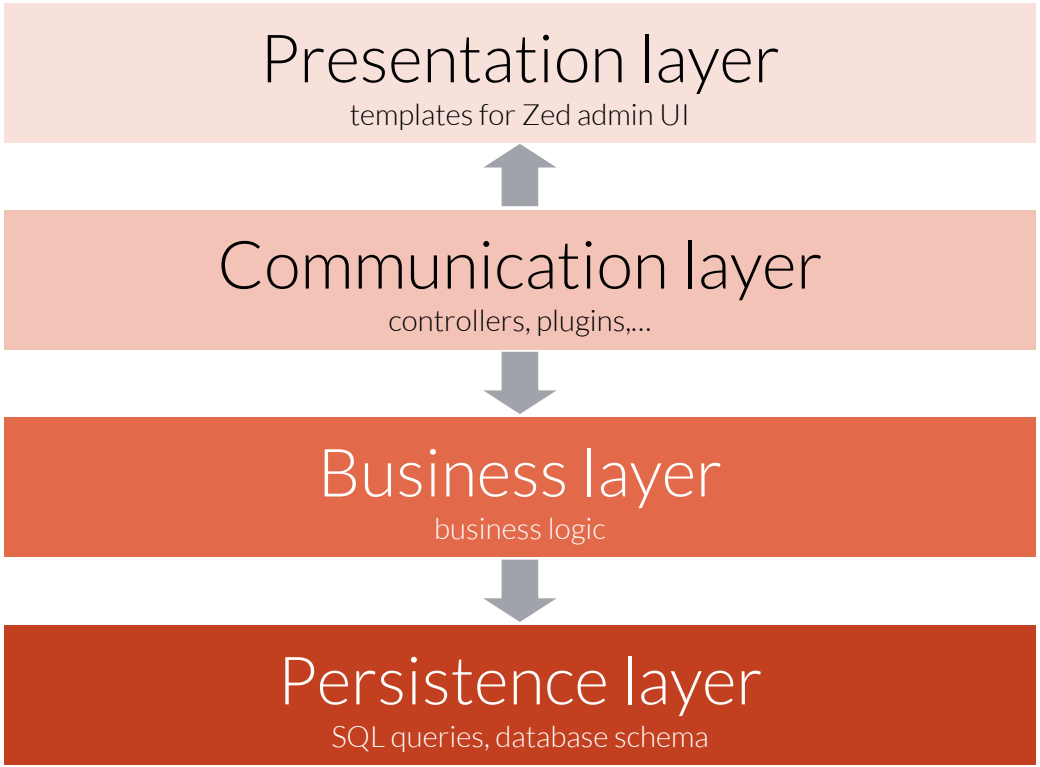
**Shared**
Shared between
Yves and Zed

**Yves**
The frontend application

**Zed**
Bundle layers, business logic
and implementation



# Bundle layers



# Persistence Layer





## Here there are all entities, queries and the schema

| Elements | Parts and purpose | Locatable |
|---|---|---|
| Query Container | The QueryContainer contains of all queries and returns them unterminated. This class' responsibility is to hide the structure of the schema and to enable reuse of queries | yes |
| Entities | An entity represents a domain object, like a 'customer' or a 'cms-page'<br>This is an implementation of the active record pattern | yes |
| Queries | The query-classes contain queries which are often used, which is useful to avoid QueryContainer to get too big | no |
| Schema | Every bundle ships with a piece of the big schema which is defined in XML | no |
| Factory | Here all objects are instantiated and classes are wired up | yes |

# Business Layer


Presentation layer
Communication layer
**Business layer**
Persistence layer

## This layer holds all business logic

| Elements | Parts and purpose | Locatable |
|---|---|---|
| **Facade** | A facade represents the entry point into the business layer. We could also see it as internal API<br>*i* This is a implementation of the facade desing pattern | yes |
| **Models** | The models hold the business logic | no |
| **Factory** | Here all objects are instantiated and classes are wired up | yes |

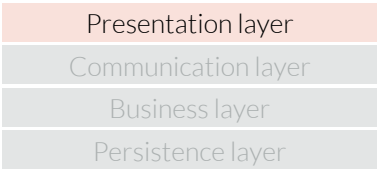
# Communication Layer


Presentation layer
Communication layer
Business layer
Persistence layer

This layer is a bridge between the presentation layer and the external world *(HTTP-request, CLI-call, cronjob, other bundles, etc.)*

| Elements | Parts and purpose | Locatable |
|---|---|---|
| Controllers | Normal controller with actions. Responsible to retrieve request-data, delegate to business layer and provide content to the presentation layer | yes |
| Forms | A class holds the constraints and provides the data. The layout and field-types are defined in twig-template | no |
| Plugins | Plugins are classes which are used to loosly couple bundles (see dependency system) | no |
| Factory | Here all objects are instantiated and classes are wired up | yes |

# Presentation Layer



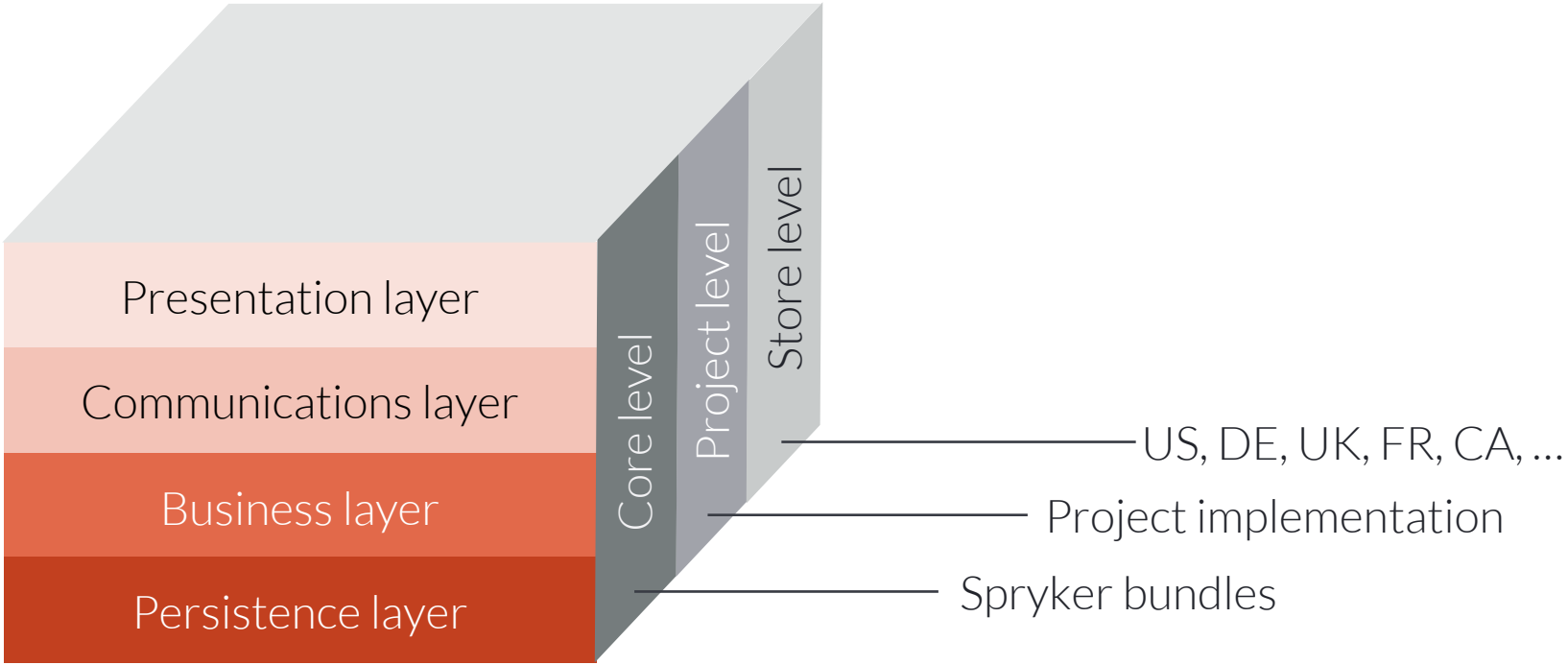

**This layer holds all templates including all HTML, CSS and Javascript content**

We separate presentation & logic

We use semantic CSS to get a uniform look and feel

| Technology | Purpose |
| --- | --- |
| **Twig** | Templating Engine |
| **Forms** | Frontend Development System |

# Separated Core, Project & Localized code



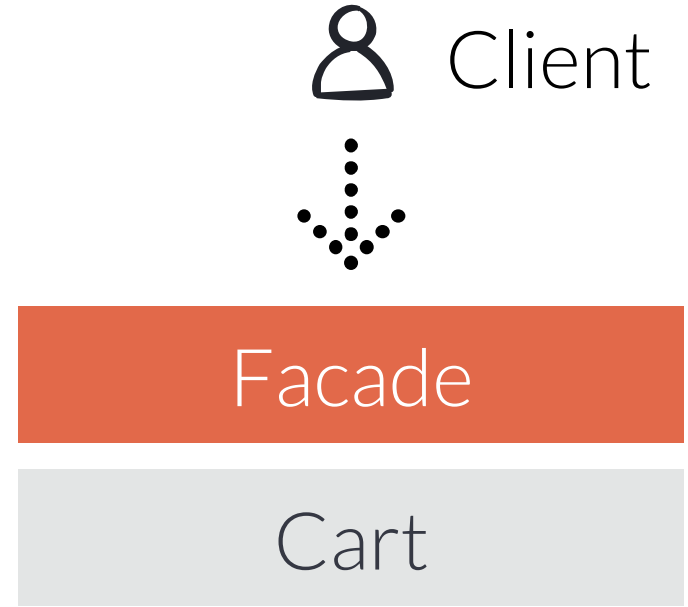
# Main software concept

# Facade

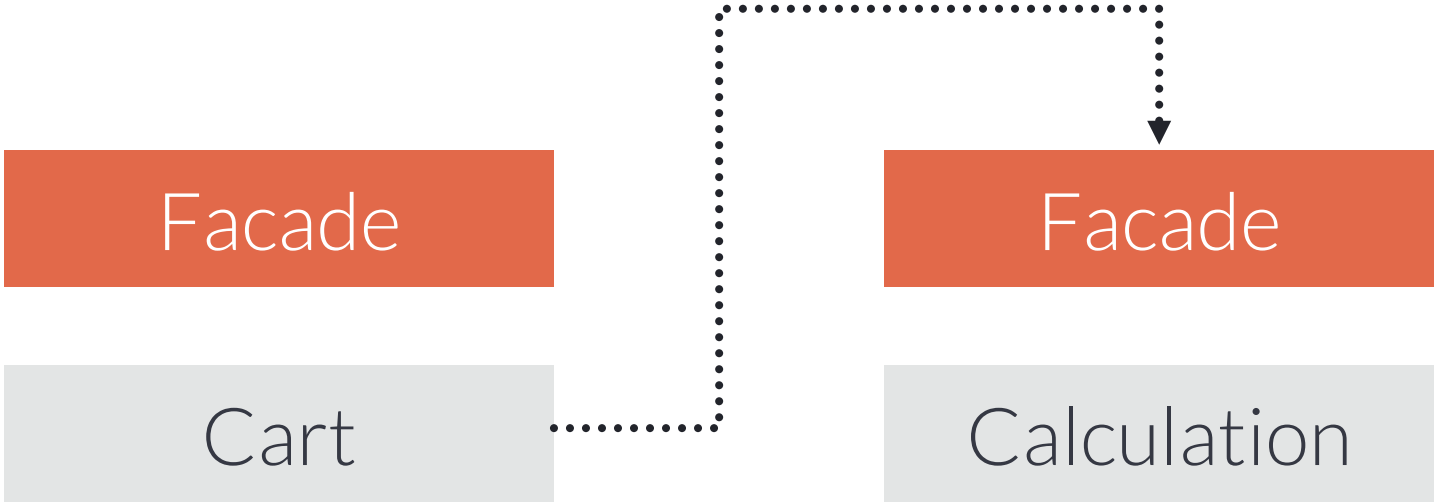
## The main API Located in the business layer

```
/**
 * Specification:
 * - Hydrate discount entity from DiscountConfiguratorTransfer and persist it.
 * - If discount type is voucher create voucher pool without voucher Codes
 * - Return id of discount entity in persistence.
 *
 * @api
 *
 * @param \Generated\Shared\Transfer\DiscountConfiguratorTransfer $discountConfigurator
 *
 * @return int
 */
public function saveDiscount(DiscountConfiguratorTransfer $discountConfigurator)
{
    return $this ->getFactory()
        ->createDiscountPersist()
        ->save($discountConfigurator);
}
```

# Bundle API



# Bundle to Bundle



# QueryContainer

## Builds and execute bundle database queries

```
/**
 * @api
 *
 * @return \Orm\Zed\Discount\Persistence\SpyDiscountQuery
 */
public function queryDiscount()
{
    return $this ->getFactory()
        ->createDiscountQuery();
}
```

# DependencyProvider

Provides bundle-to-bundle dependency Factories uses get prefix instead of create

```
/**
 * @param \Spryker\Zed\Kernel\Container $container
 * @return \Spryker\Zed\Kernel\Container
 */
public function provideBusinessLayerDependencies(Container $container)
{
    $container [self: :FACADE_MESSENGER] = function (Container $container) {
        return new DiscountToMessengerBrige ($container->get Locator()->messenger()->facade());
    };

    return $container;
}
```

# Factory

## Create objects Every layer has its own factory

```
/**
 *@return \Spryker\Zed\Discount\Business\Persistence\DiscountPersist
 */
public function createDiscountPersist()
{
    return new DiscountPersist($this->createVoucherEngine(), $this->getQueryContainer());
}
```

# Client

## Facilitate Yves to Zed communication Manage bundle session

```
/**
 * @param \Generated\Shared\Transfer\AddressesTransfer $addressesTransfer
 *
 * @return \Generated\Shared\Transfer\CustomerTransfer
 */
public function createAddressAndUpdateCustomerDefaultAddresses(AddressesTransfer $addressesTransfer)
{
    $customerTransfer = parent : : createAddressAndUpdateCustomerDefaultAddresses ($addressesTransfer);
    $this->setCustomer($customerTransfer);

    return $customerTransfer;
}
```

# Dependency Injection with Factories

Explicit injection with background magic Separated Factories per bundle/layer, no big DI-container

```
/**
 * @return \Spryker\Zed\Discount\Business\Calculator\Discount
 */
public function createDiscount()
{
    return new Discount(
        $this->getQueryContainer(),
        $this->createCalculator(),
        $this->createDecisionRuleBuilder(),
        $this->createVoucherValidator(),
    );
}
```

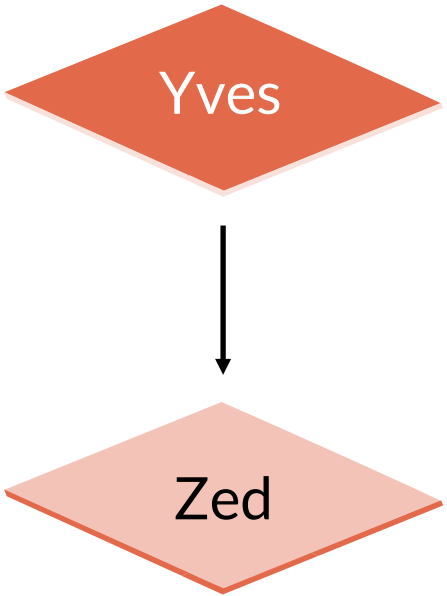
# Extension with Plugins

## Easy to extend Simple to use Help for generic implementation Plugin stacks

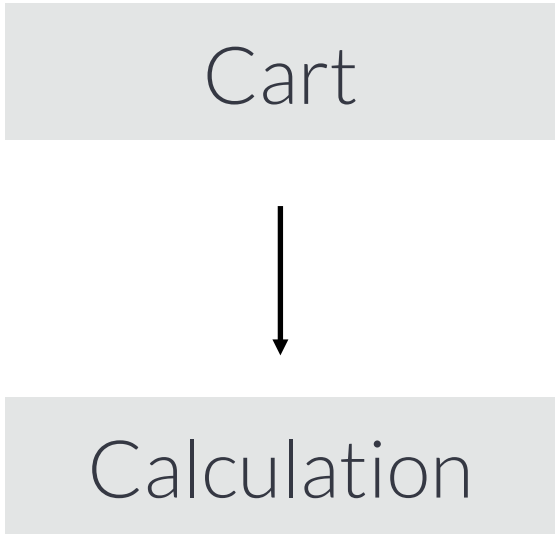
```
/**
 * @param \Spryker\Zed\Kernel\Container $container
 *
 * @param \Spryker\Zed\Calculation\Dependency\Plugin\CalculatorPluginInterface[]
 */
protected function getCalculatorStack(Container $container)
{
    return [
        //Remove calculated values, start with clean state.
        new RemoveTotalsCalculatorPlugin(),
        new RemoveAllCalculatedDiscountsCalculatorPlugin(),
        //Item calculator
        new ProductOptionGrossSumCalculatorPlugin(),
        new ItemGrossAmountsCalculatorPlugin(),
        //SubTotal
        new SubtotalTotalsCalculatorPlugin(),
        //Expenses (e.g. shipping)
        new ExpensesGrossSumAmountsCalculatorPlugin(),
        new ExpenseTotalsCalculatorPlugin(),
        //GrandTotal
        new GrandTaoralTotalsCalculatorPlugin(),
        new GrandTotalWithDiscountsCalculatorPlugin()
    ];
}
```

# Data Transfer Objects

Transfer data from Yves to Zed




Transfer data between bundles



# Transfer Objects

```
<transfer name="Discount">
        <property name="idDiscount" type="int"/>
        <property name="fkDiscountVoucherPool" type="int"/>
        <property name="idSalesDiscount" type="int"/>
        <property name="displayName" type="string"/>
        <property name="description" type="string"/>
        <property name="amount" type="int"/>
        <property name="voucherCode" type="string"/>
</transfer>

<transfer name="CollectedDiscount">
  <property name="discountableItems" type="DiscountableItem[] "/>
  <property name="discount" type="Discount"/>
</transfer>
```

# Dependencies

One bundle depends on another bundle

# Dependencies

The two bundles are independent

# Dependencies

The two bundles depend on one another

# Software Principles & Patterns

## SOLID: Single Responsibility Principle

```
namespace SprykerFeature\Zed\Glossary\Business;
use ...
/**
 *@method GlossaryDependencyContainer getDependencyContainer()
 */
class GlossaryFacade extends AbstractFacade
{
    /**@param string $keyName ...*/
    public function createKey ($keyName)
    {
        $keyManager = $this->getDependencyContainer()->createKeyManager();
        return $keyManager->createKey($keyName);
    }
    /**@param string $keyName ...*/
    public function hasKey ($keyName)
    {
        $keyManager = $this->getDependencyContainer()->createKeyManager();
        return $keyManager->hasKey($keyName);
    }
```

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

- Default principle for each class is defined in Spryker
- Single responsibility principle is applied on bundle, class, and method level
- Example : facade classes represent the internal API of a bundle and the only point of access to a bundle
- Facades are flat and stateless classes - their only responsibility is to orchestrate the incoming requests to the business logic implemented in the bundle

# Software Principles & Patterns

## SOLID: Open Closed Principle

```
namespace SprykerFeature\Shipment\Communication\Plugin;
use Generated\Shared\Shipment\ShipmentMethodAvailabilityInterface;
interface ShipmentMethodAvailabilityPluginInterface
{
    /**
     *@param ShipmentMethodAvailabilityInterface $shipmentMethodAvailability
     *
     *@return bool
     */
    public function isAvailable (ShipmentMethodAvailabilityInterface $shipmentMethodAvailability);
}
namespace Pyz\Zed\Shipment\Communication\Plugin\Availability;
use ...
class DHLExpressPlugin extends AbstractAvailabilityPlugin
{
    const VALID_COUNTRIES = [...];
    /**
     *@param ShipmentMethodAvailabilityInterface $shipmentMethodAvailability
     *
     * @return bool
     */
    public function isAvailable (ShipmentMethodAvailabilityInterface $shipmentMethodAvailability);
{
        return $this-> isCountryAvailable(self:: VALID_COUNTRIES, $shipmentMethodAvailability->
                getAddress());
    }
}
```

A module should be open for exten-sion but closed for modification

- Modules are written so that they can be extended without being modified
- Implemented through plugins in Spryker
- Example: deliver customization & add complex configuration in the sales flow
- Example of possible implement-tations for *ShipmentMethodAvailabi-lity PluginInterface* : DHLExpress Plugin / DHLStandardPlugin
- Each of these plugins must implement ShipmentMethod-AvailabilityPluginInterface

# Software Principles & Patterns

## SOLID: Interface Segregation Principle

```
namespace SprykerFeature\Zed\Glossary\Dependency\Facade;
use ...
interface GlossaryToLocaleInterface
{
        /**
         *@param string $localeName
         *
         *@return LocaleTransfer
         *@throws MissingLocaleException
        */
    public function getLocale($localeName);
        /**
        * @return LocaleTransfer
         */
    public function getCurrentLocale();
        /**
         * @return array
         */
    public function getAvailableLocales();
```

Many client specific interfaces are better than one general interface

- In Spryker this is used for facades
- In order to substitute a bundle with a new one, its facade must contain the same operations so that it doesn't break the contract established with other bundles
- Each bundle ships with a set of interfaces for facades of the required bundles

# Software Principles & Patterns

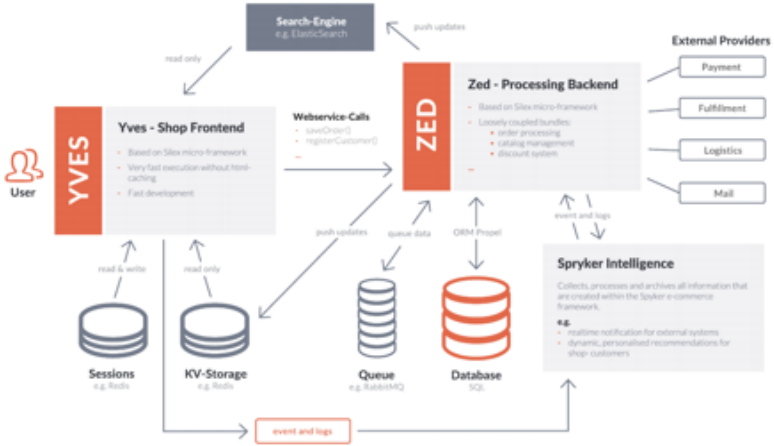
## SOLID: Dependency Inversion Principle

```
class GlossaryDependencyContainer extends AbstractBusinessDependencyContainer
{
    /**
     *@return TranslationManagerInterface
     */
    public function createTranslationManager();
{
     return $this->getFactory ( ) ->createTranslationTranslationManager(
            $this->getQueryContainer ( ),
            $this->getTouchFacade ( ),
            $this->getLocaleFacade ( ),
            $this->createManager ( ),
            $this->getFlashMessagesFacade ( )
     ) ;
}
class GlossaryDependencyProvider extends AbstractBundleDependencyProvider
{
    /** @param Container $container ...*/
    public function provideCommunicationLayerDependencies (Container $container);
{
     $container [GlossaryDependencyProvider : : FAÇADE_LOCALE] = function (Container $container) {
            return $this->getLocator ( ) ->application( ) -> pluginPimple ( ) -> getApplication( ) [´validator´];
     } ;
       $container [GlossaryDependencyProvider : : PLUGIN_VALIDATOR] = function (Container $container) {
            return $this->getLocator ( ) ->application( ) -> pluginPimple ( ) -> getApplication( ) [´validator´];
     } ;
     return $container ;
}
```

Depend upon Abstractions. Do not depend upon concretions

- High level modules shouldn't know implementation details about other modules, they should depend only on abstractions
- Dependency container: one for each layer in a bundle
- The dependency container is IDE friendly php code, no hassle with foreign structures like yaml or xml
- Dependency provider: one for each bundle

# Does it Really Work?




1

Incredible scores for code quality & speed (Img: Scrutinizer CI code quality rating)


2

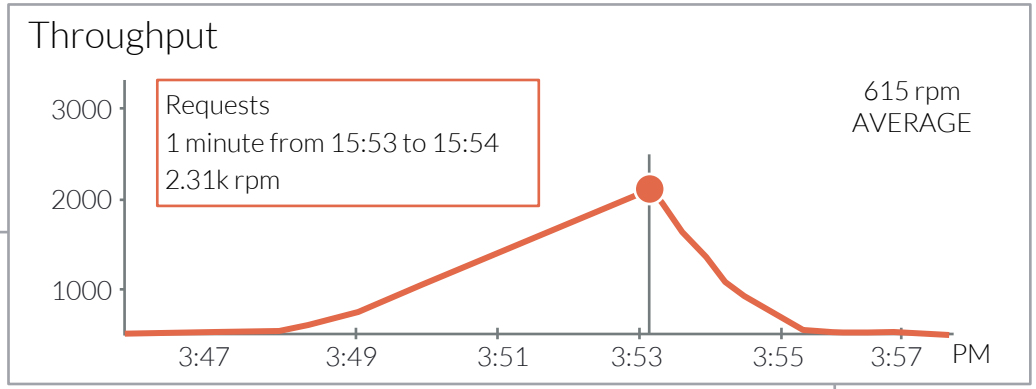
Survey with active Spryker developers (clients & agencies) How much faster are you now compared to Magento & Co.?

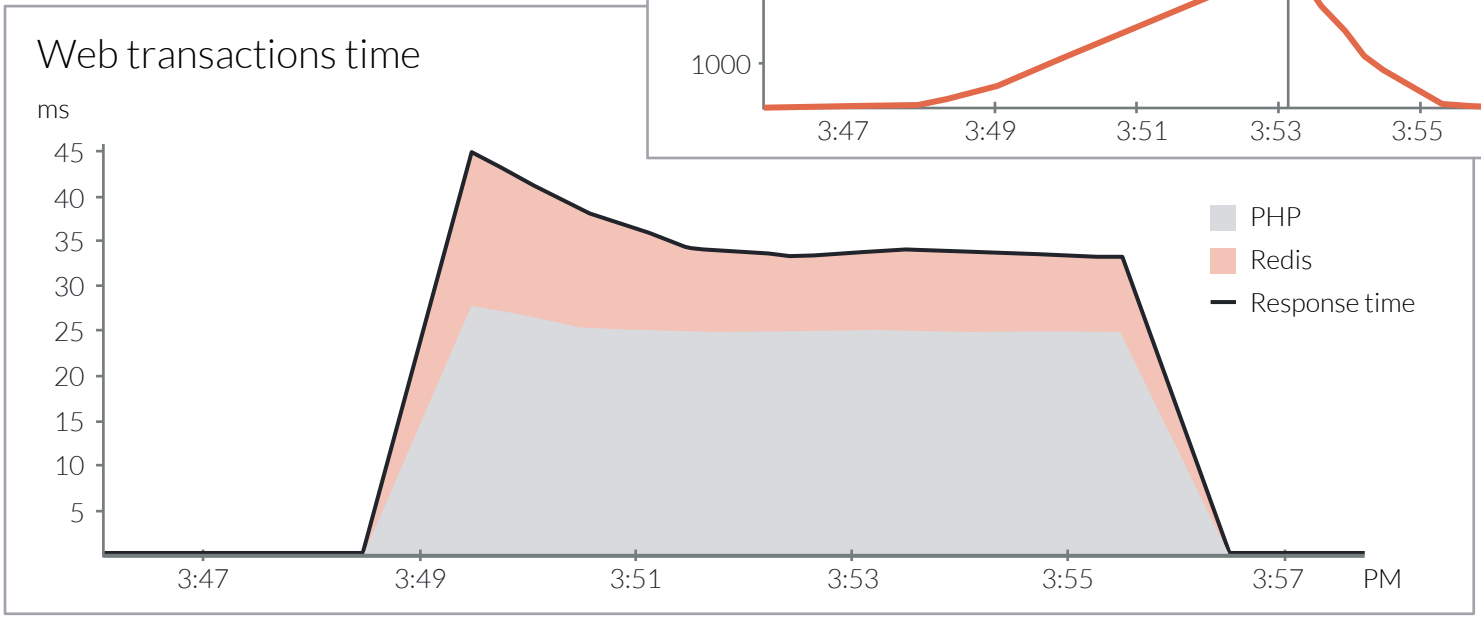
**2x –20x**

> Truth can only be found in one place: the code.

# Speed & Security

# Spryker Load Test

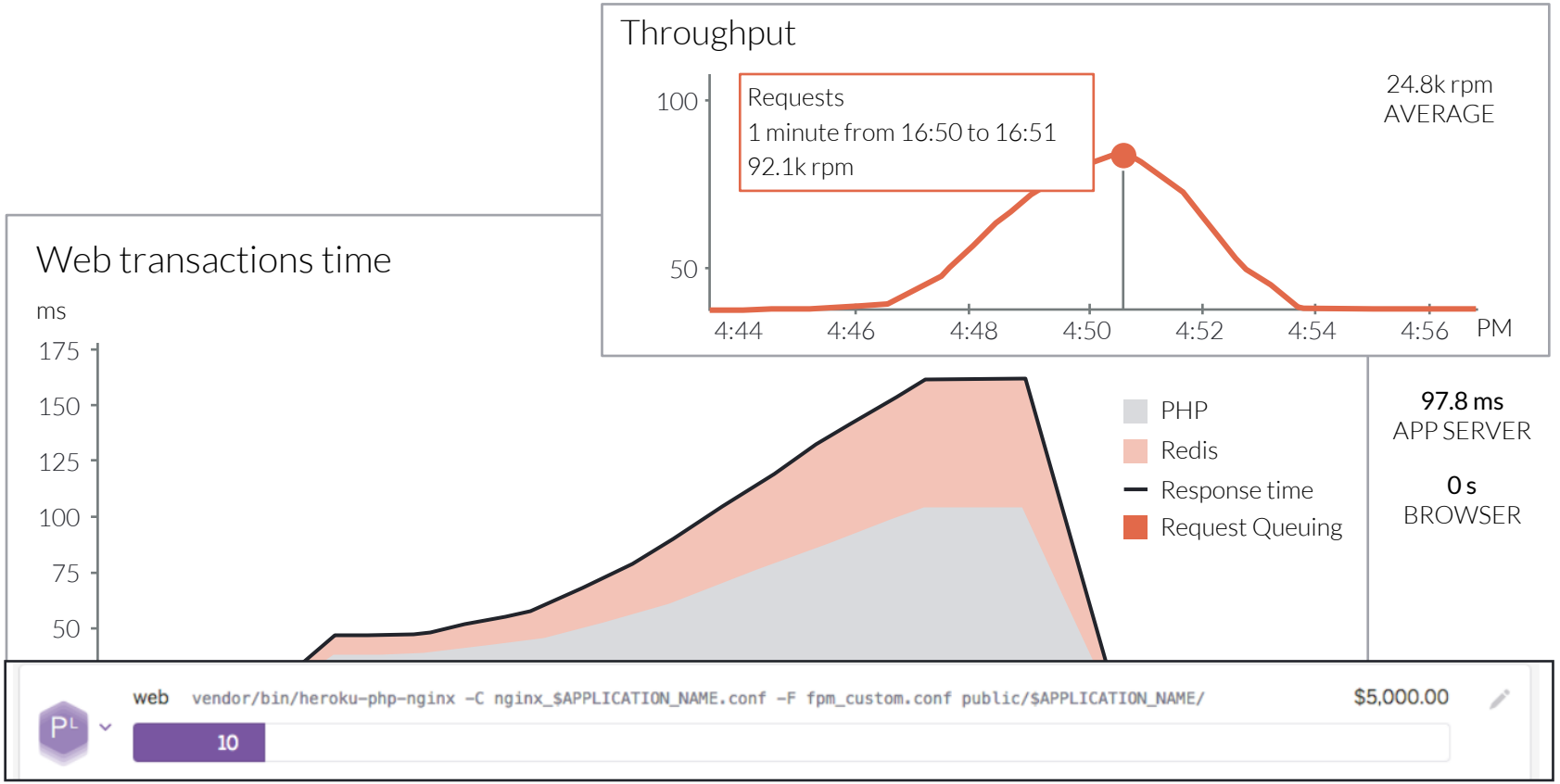



**34.5 ms**
APP SERVER

**0 s**
BROWSER

# Spryker Load Test



# Improved performance & code quality

**Yves**
shop frontend


*Demoshop execution time in DevVM*

# Security

- Faster security patches through individual bundle upgrades
- Spryker is always compatible with latest 3rd party tools & libraries (e.g. Symfony)
- Latest PHP Version is always supported
- Frontend / Backend separation provides extra security
- Platform agnostic (Webserver, DB, KV-Storage, …)

# Security


- Security audit by SektionEins
- No major issues found
- Fixed all minor flaws

# State Machine(s)

# OMS

if … else

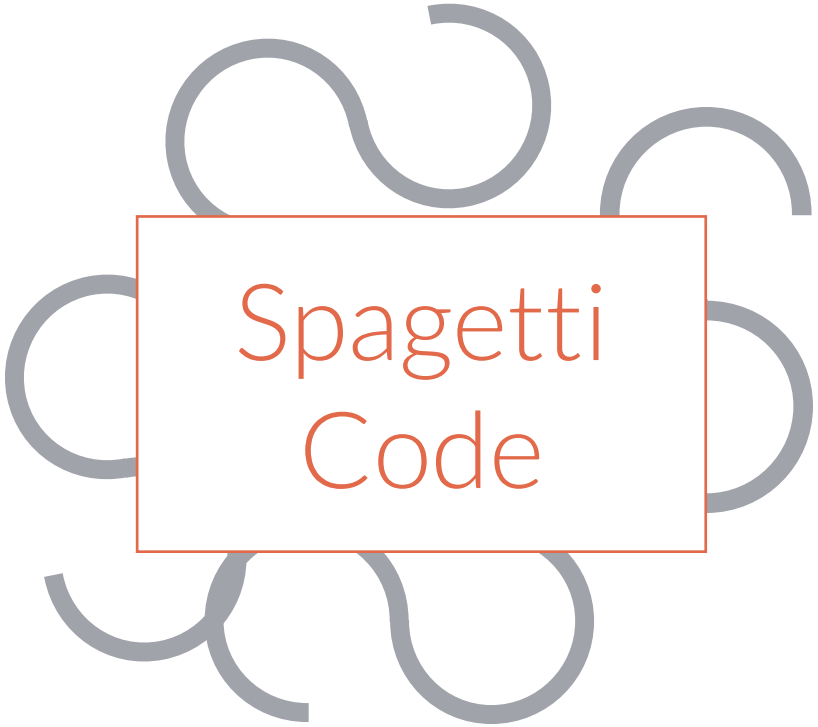
if … else if …

else …

if …

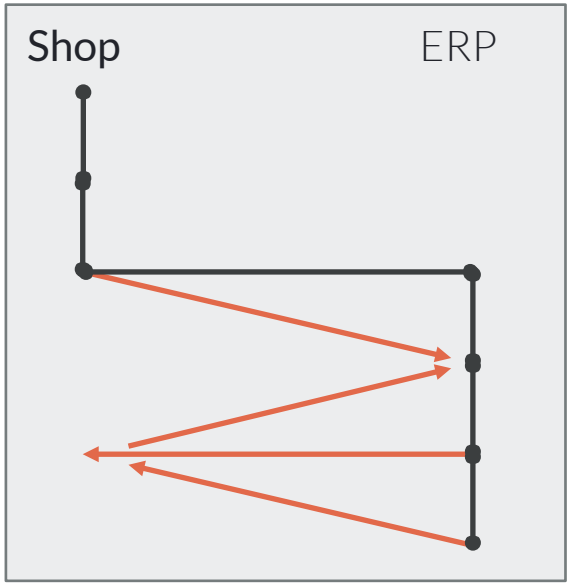
else if …

else …



# Order Management in Spryker

## Other Systems (optional w/Spryker)




reality Shop and ERP: black box with high potential for errors

*order*

*payment*

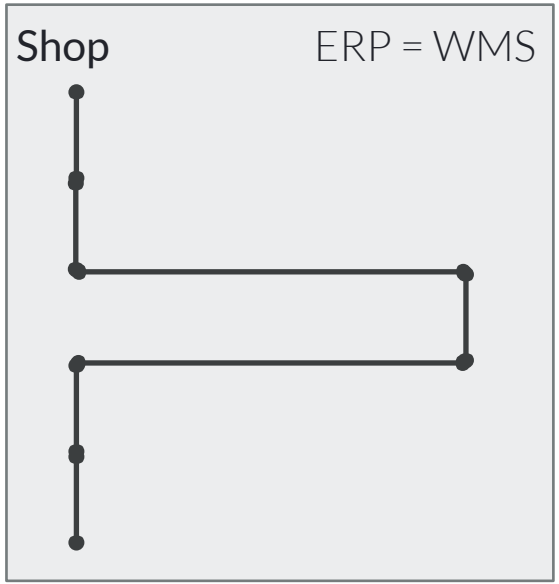
*pick & pack*

*ship*

*return*

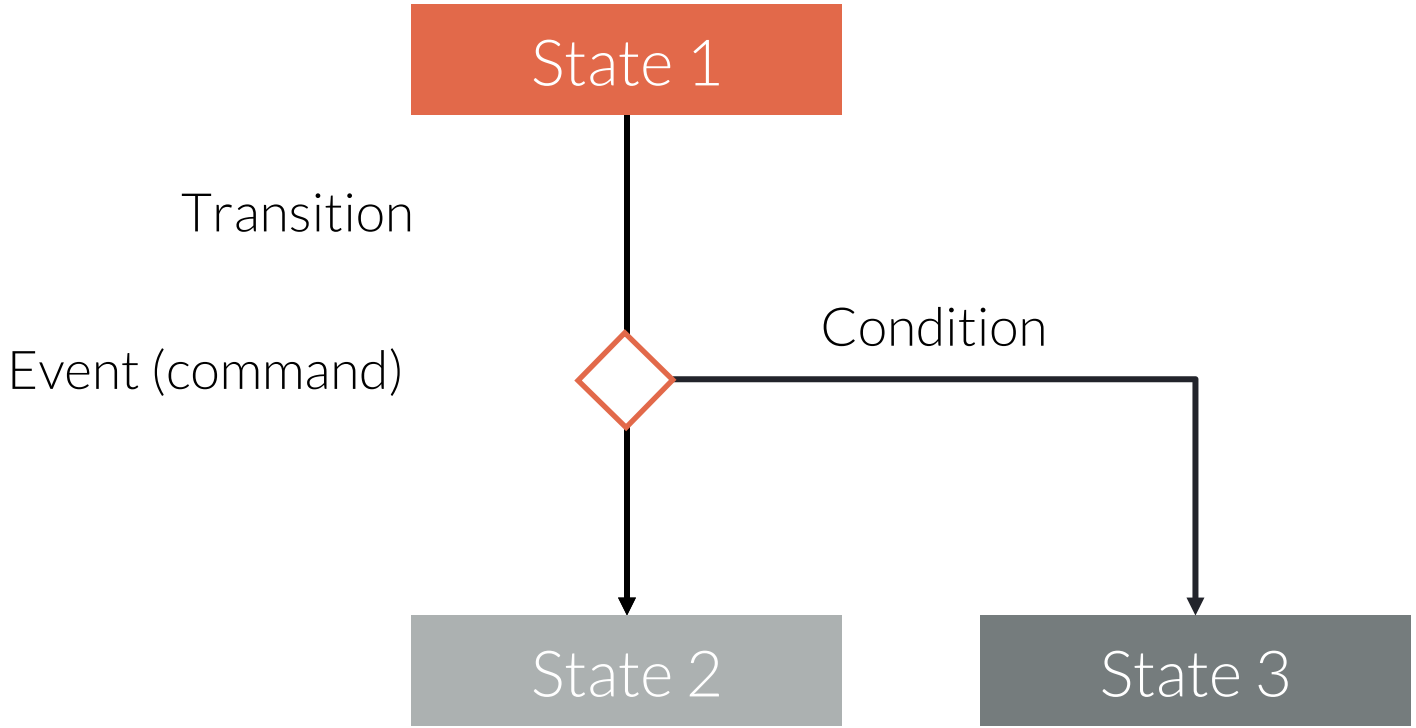
*refund*

## Spryker (recommended w/Spryker)



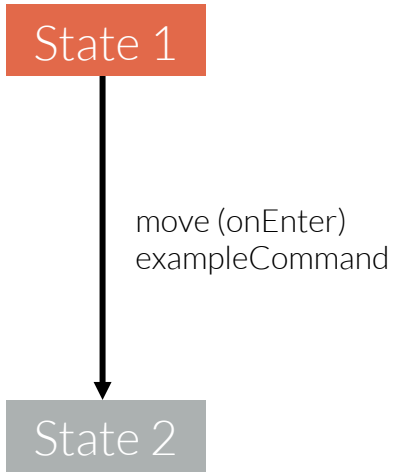

Spryker allows for separation of concerns, order processed in shop

# State Machine



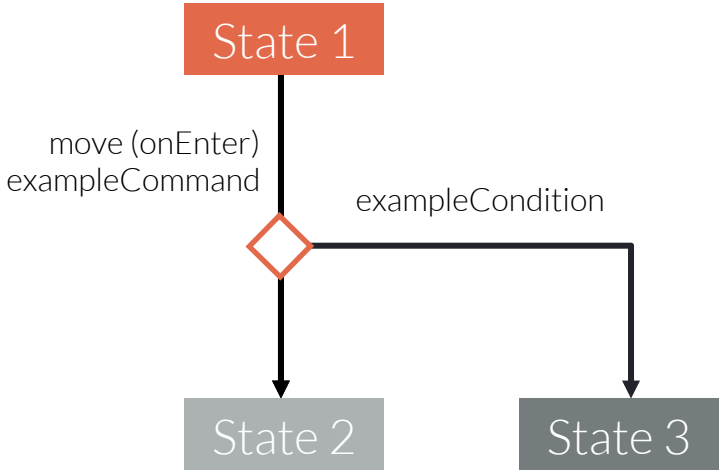
# State Machine

```
<?xml version="1.0" encoding="utf-8"?>
<statemachine
                    xmlns="https://static.spryker.com"
                    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
                    xsi:schemaLocation="https://static.spryker.com https://static.spryker.com/oms-01.xsd">
        <process name="Example" main="true">
      <states>
                    <state name="State 1" reserved="true"/>
                    <state name="State 2" reserved="true"/>
      </states>
          <transitions>
            <transition>
                    <source>State 1</source>
                    <target>State 2</target>
                    <event>move</event>
             </transition>
          </transitions>
          <events>
            <event name="move" onEnter="true" command="ExampleCommand"/>
          </events>
        </process>
</statemachine>
```



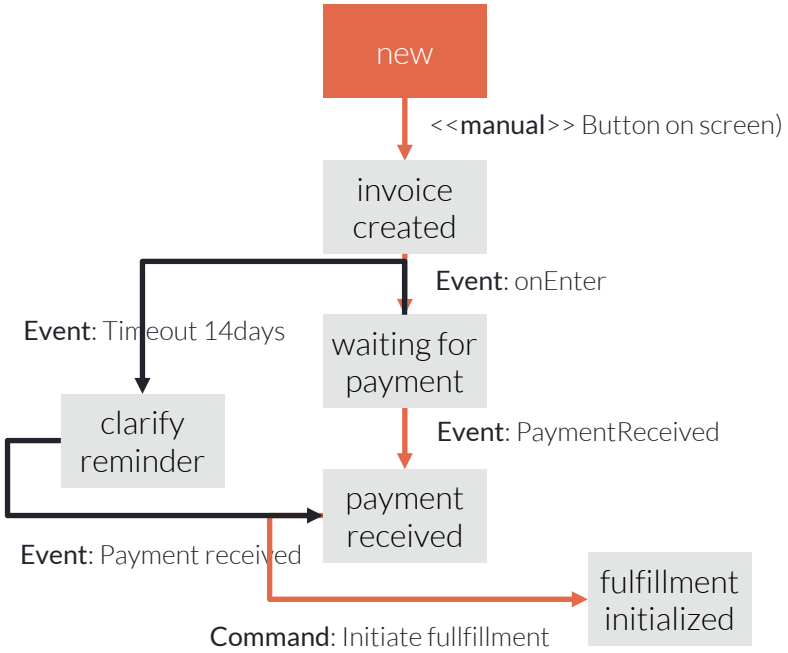
# State Machine

```
<?xml version="1.0" encoding="utf-8"?>
<statemachine
                    xmlns="https://static.spryker.com"
                    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
                    xsi:schemaLocation="https://static.spryker.com https://static.spryker.com/oms-01.xsd">
  <process name="Example" main="true">
    <states>
                    <state name="State 1" reserved="true"/>
                    <state name="State 2" reserved="true"/>
                    <state name="State 3" reserved="true"/>
    </states>
      <transitions>
        <transition condition="ExampleCondition">
                    <source>State 1</source>
                    <target>State 2</target>
                    <event>move</event>
        </transition>
        <transition>
                    <source>State 1</source>
                    <target>State 2</target>
                    <event>move</event>
        </transition>
      <transitions>
        <events>
          <event name="move" onEnter="true" command="ExampleCommand"/>
        </events>
    </process>
</statemachine>
```

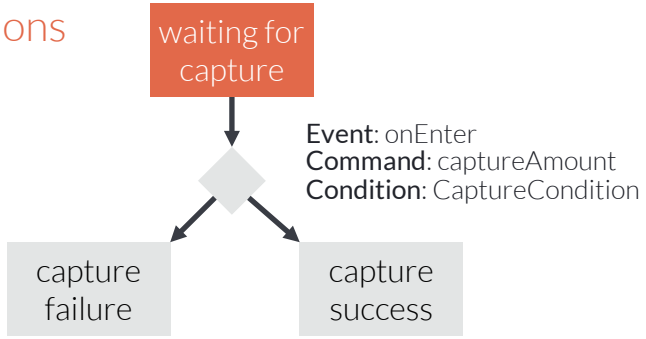


# Order Management in Spryker

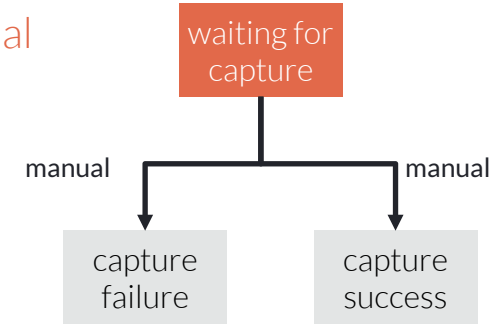
## Process model design / State machines




## Conditions

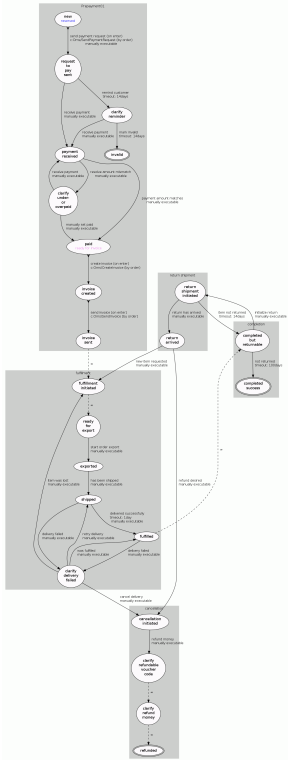


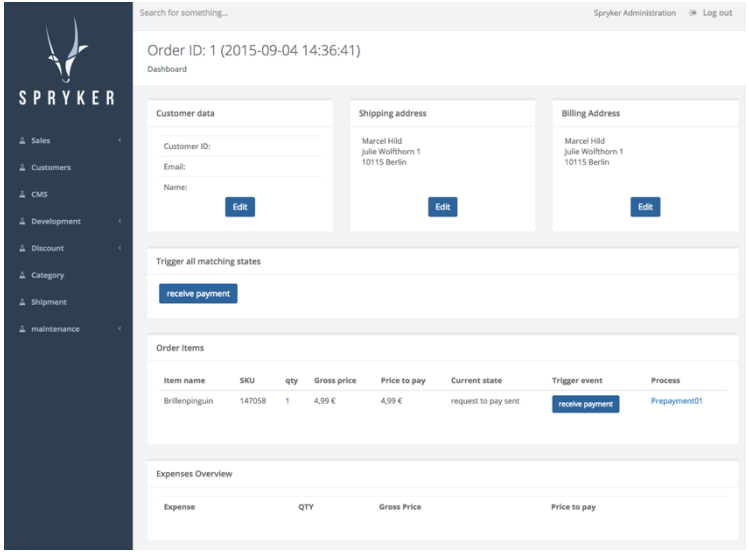

## Manual



# Order Management in Spryker

```xml
<?xml version="1.0" encoding="utf-8"?>
<statemachine
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="process.xsd">
    <process name="Prepayment01" main="true">
        <subprocesses>
            <process>fulfillment</process>
            <process>completion</process>
            <process>return shipment</process>
            <process>cancellation</process>
        </subprocesses>
        <states>
            <state name="new" reserved="true"/>
            <state name="request to pay sent"/>
            <state name="payment received"/>
            <state name="clarify reminder"/>
            <state name="invalid" reserved="false"/>
            <state name="clarify under- or overpaid"/>
            <state name="paid">
                <flag>ready for invoice</flag>
            </state>
            <state name="invoice sent"/>
            <state name="invoice created"/>
        </states>
        <transitions>

            <transition>
                <source>new</source>
                <target>request to pay sent</target>
                <event>send payment request</event>
            </transition>

            <transition>
                <source>request to pay sent</source>
                <target>clarify reminder</target>
                <event>remind customer</event>
            </transition>

            <transition>
                <source>clarify reminder</source>
                <target>payment received</target>
                <event>receive payment</event>
            </transition>

            <transition>
                <source>request to pay sent</source>
                <target>payment received</target>
                <event>receive payment</event>
            </transition>

            <transition>
                <source>clarify under- or overpaid</source>
                <target>payment received</target>
                <event>receive payment</event>
            </transition>
```






Screenshots / Images available separately as well as state machine demo and source code

# Atomic release

# Why doing it this way?

## Developers

- no update hell
- freedom to update specific bundles

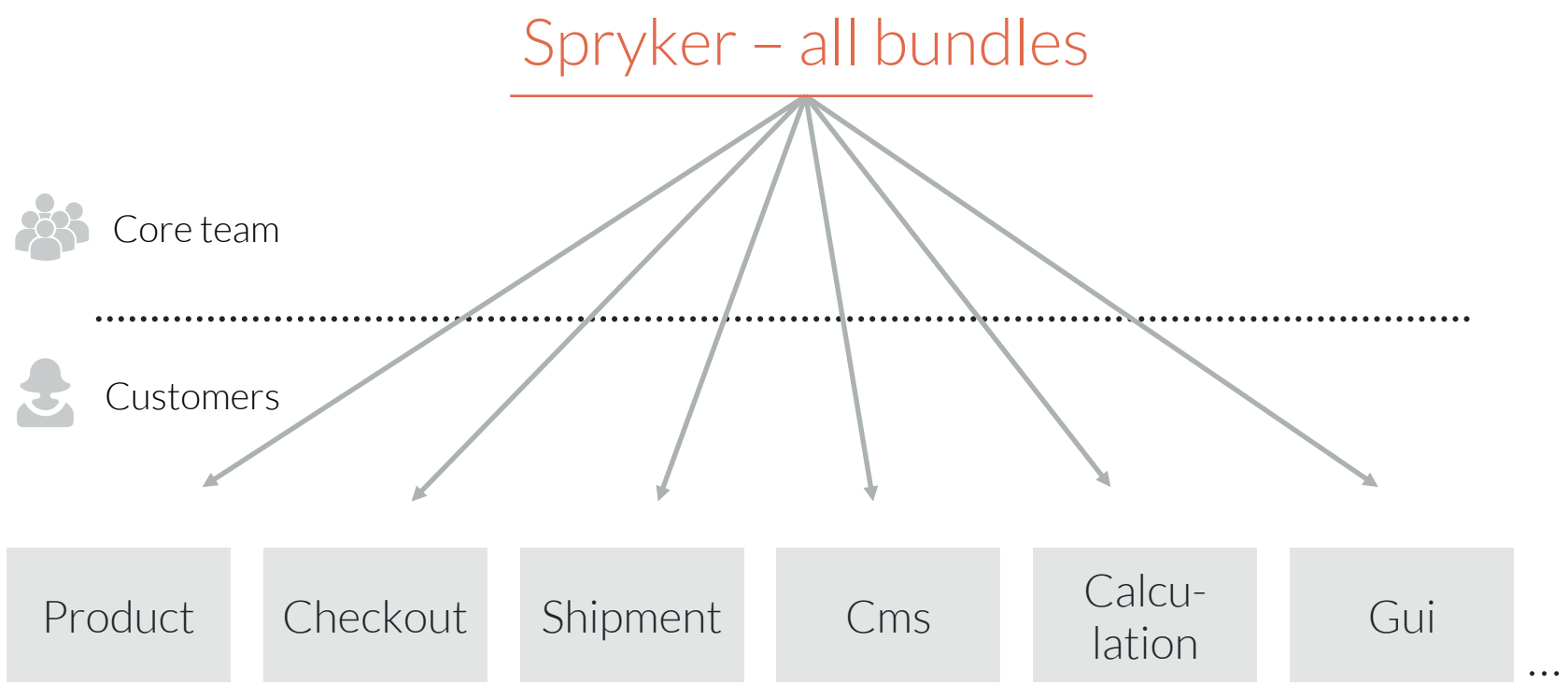
## Core team

- develop as we used to

## Release

- easy planning
- easy deploying and maintaining

# Subtree Split



# Versioning

1 . 2 . 3

# Versioning


1 . 2 . 3

BC Break

# Versioning


1 . 2 . 3

BC Break Feature

# Versioning

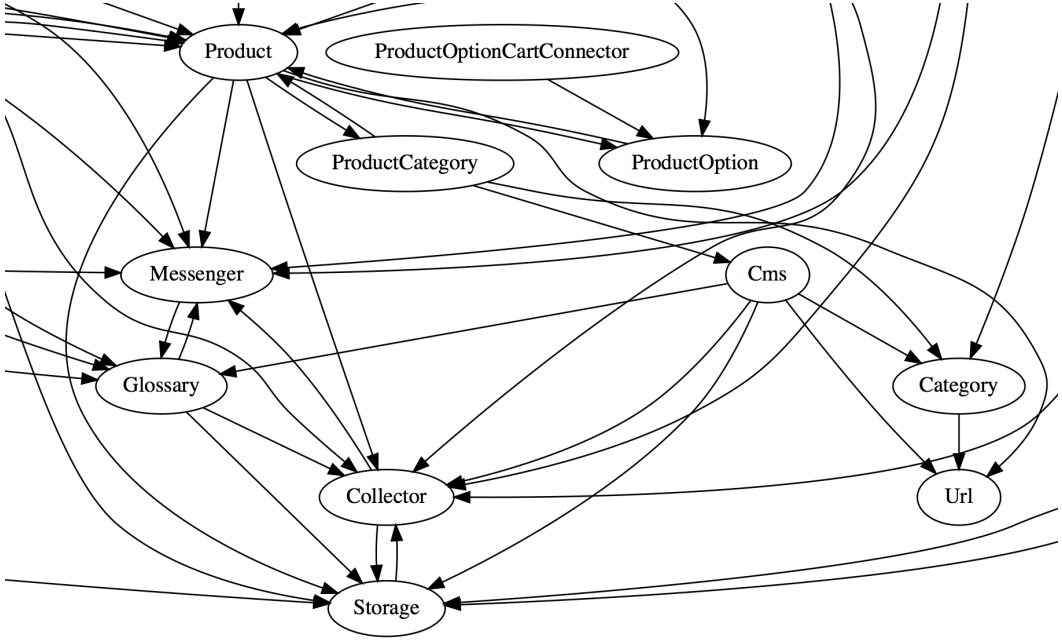

1 . 2 . 3

BC Break Feature Patch

# Releases & Versions

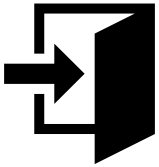
| | | | | |
|---|---|---|---|---|
| Product 2.0.0 | Checkout 2.3.0 | Shipment 2.0.2 | Catalog 2.0.1 | Customer 2.5.1 |
| Payment 2.7.0 | Cms 2.0.0 | Cart 3.1.0 | Category 2.1.0 | Oms 3.1.0 |
| Auth 3.0.0 | Tax 2.2.2 | Discount 2.5.0 | Search 3.0.0 | Calculation 2.1.1 |

# Solving Dependencies




```
{
"name": "spryker/product",
"autoload": {
    "psr-0": {
                "Spryker": "src/",
                "Acceptance": "tests/",
                "Functional":  "tests/",
                "Integration":  "tests/",
                "Unit":  "tests/",
                "YvesUnit": "tests/"
    }
},
"require": {
    "spryker/application": "^2.0.0",
    "spryker/collector": "^2.0.0 || ^3.0.0",
    "spryker/console": "^2.0.0",
    "spryker/gui": "^2.0.0",
    "spryker/installer": "^2.0.0 || ^3.0.0",
    "spryker/kernel": "^2.0.0",
    "spryker/library": "^2.0.0",
    "spryker/locale": "^2.0.0",
    "spryker/messenger": "^2.0.0",
    "spryker/product-category": "^2.0.0",
    "spryker/product-image": "^1.0.0",
    "spryker/product-option": "^2.0.0",
    "spryker/url": "^2.0.0",
},
"require-dev": {
    "spryker/tax": "^2.0.0"
},
"description": "Product bundle",
"license": "proprietary",
"minimum-stability": "dev",
"prefer-stable": true,
"extra": {
    "branch-alias": {
        "dev-master": "2.0.x-dev"
    }
},
"config": {
    "sort-packages": true
}
}
```

# Solving Dependencies



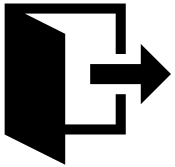
## Internal dependencies

^1.2.3

>=1.2.3

<2.0.0

All compatible versions



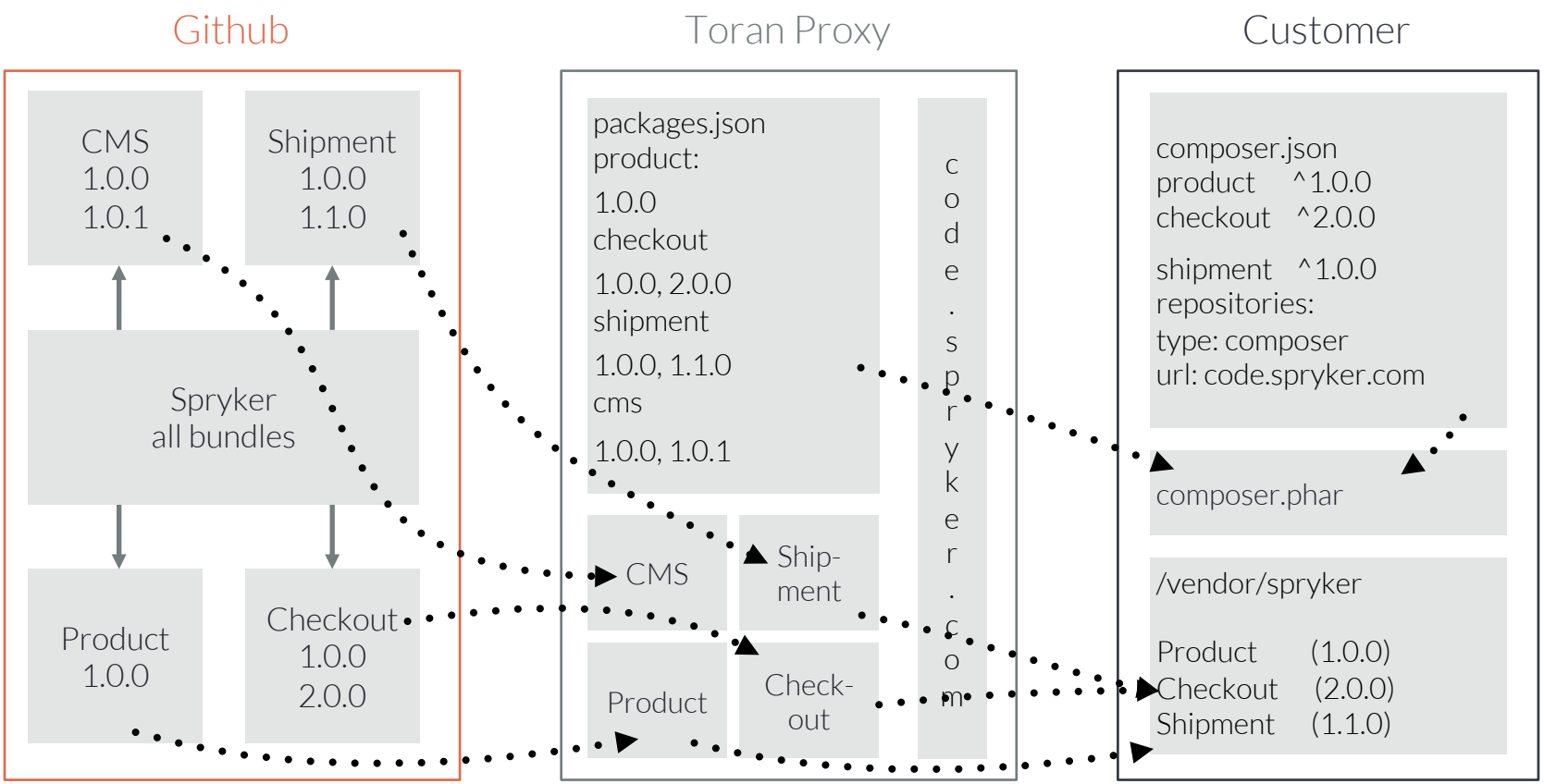
## External dependencies

~1.2.3

>=1.2.3

<1.3.0

Bug fixes only


Github
Toran Proxy
Customer
CMS
1.0.0
1.0.1
Shipment
1.0.0
1.1.0
Spryker
all bundles
Product
1.0.0
Checkout
1.0.0
2.0.0
packages.json
product:
1.0.0
checkout
1.0.0, 2.0.0
shipment
1.0.0, 1.1.0
cms
1.0.0, 1.0.1
CMS
Ship-
ment
Product
Check-
out
code.spryker.com
composer.json
product ^1.0.0
checkout ^2.0.0
shipment ^1.0.0
repositories:
type: composer
url: code.spryker.com
composer.phar
/vendor/spryker
Product (1.0.0)
Checkout (2.0.0)
Shipment (1.1.0)

# Why doing it this way?

## Developers

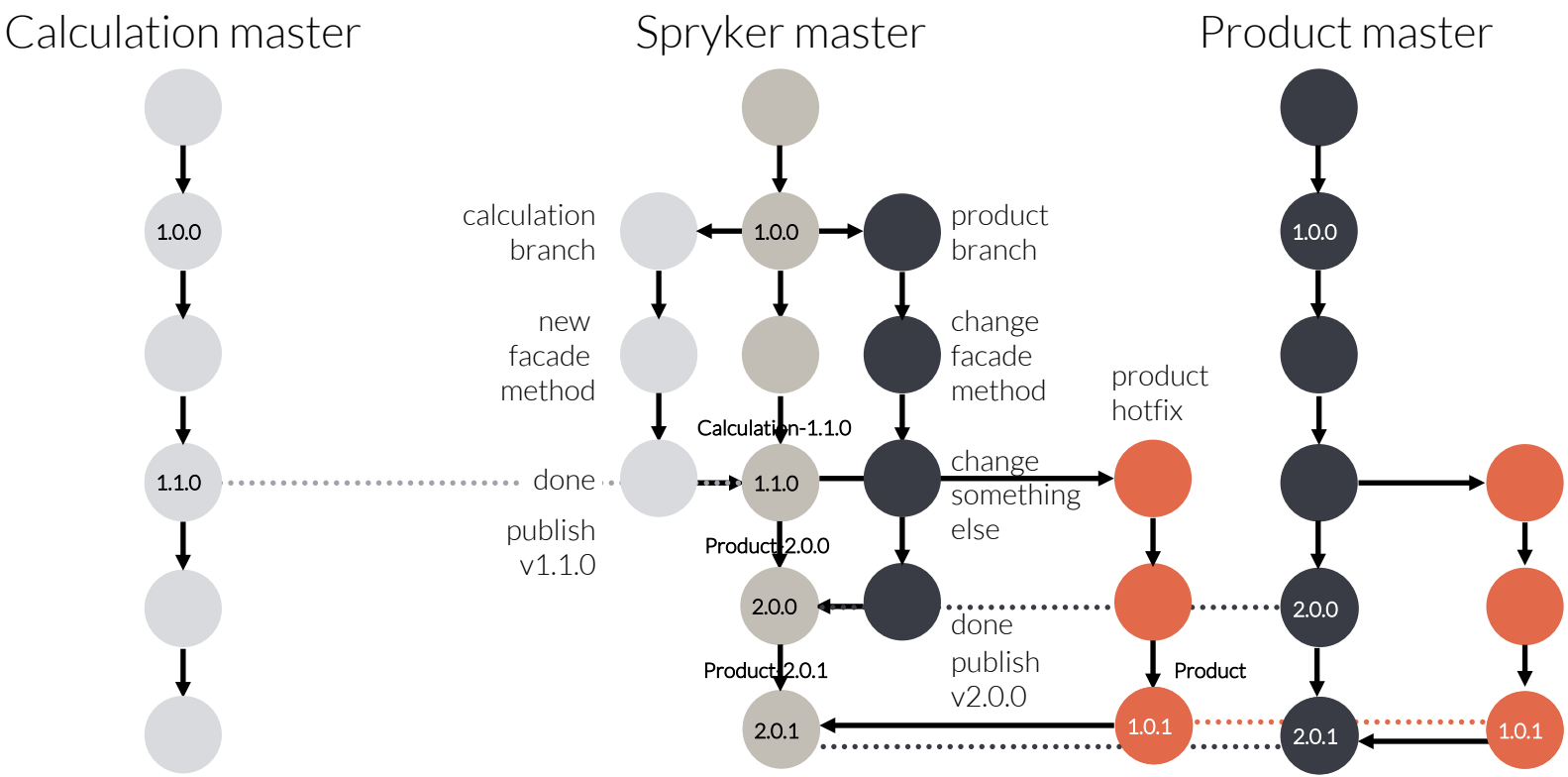
- each bundle has its own version
- semantic versio-ning

## Core team

- subtree split

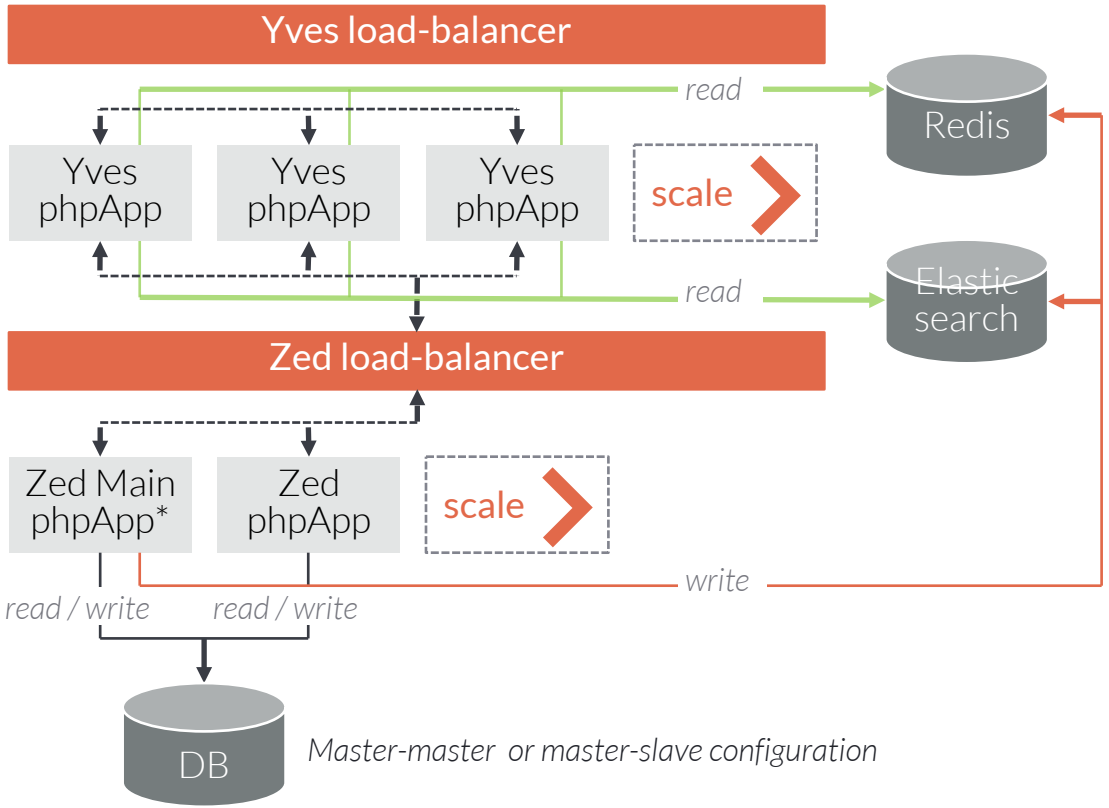
## Release

- plan only one feature for a release
- use only master
- automation


Calculation master
Spryker master
Product master
1.0.0
1.1.0
calculation branch
new facade method
done
publish v1.1.0
1.0.0
product branch
change facade method
Calculation-1.1.0
1.1.0
change something else
Product-2.0.0
2.0.0
done
publish v2.0.0
Product-2.0.1
2.0.1
product hotfix
1.0.1
Product
1.0.0
2.0.0
2.0.1
1.0.1

# Operations concepts

# Scaling (implementation example)




- Horizontal + vertical scaling enabled
- Allows for rolling deployment + Zero Downtime
- DB/Redis/Elasticsearch as high available services
- Sample infrastructure provided via saltstack templates

*Zed main server only: run cronjobs and post deployment hooks

# Store Concept

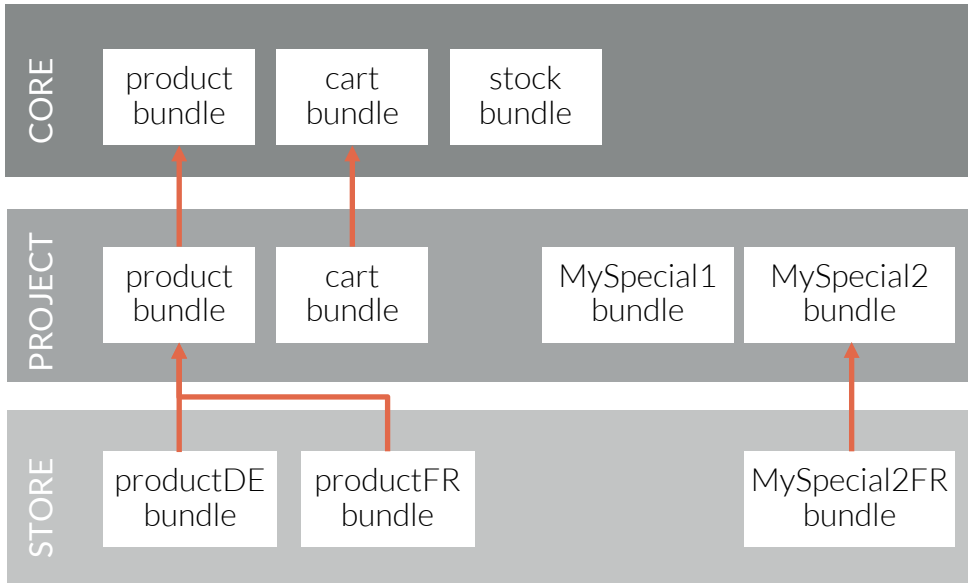
Stores are separated business units – e.g. by country

- can serve multiple languages / locales per store
- only one currency per store
- separate product and order data per store
- separate logic e.g. tax-calculator for Switzerland
- replication / distribution of data across stores possible

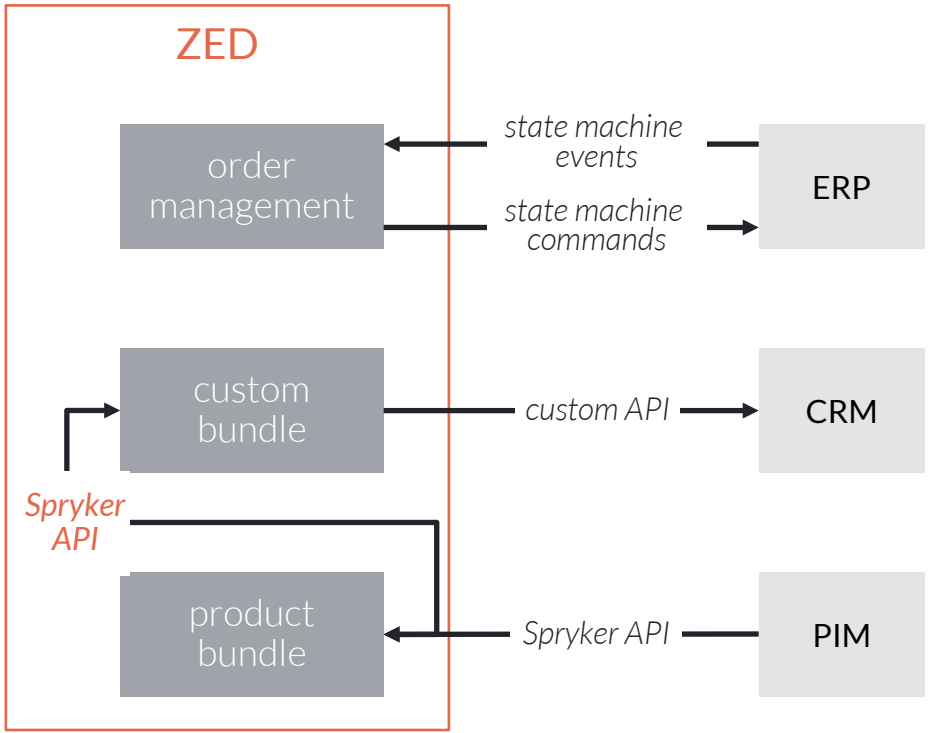
# Separated core, project & localized code

**Implementation example of the three levels**

- Extract of implemented core bundles
- Custom projects added to adapted core bundles
- Custom bundles for specific stores / localized shops
- Factories allow to extend all classes.
- Extension of the dependency container allows to replace any class.



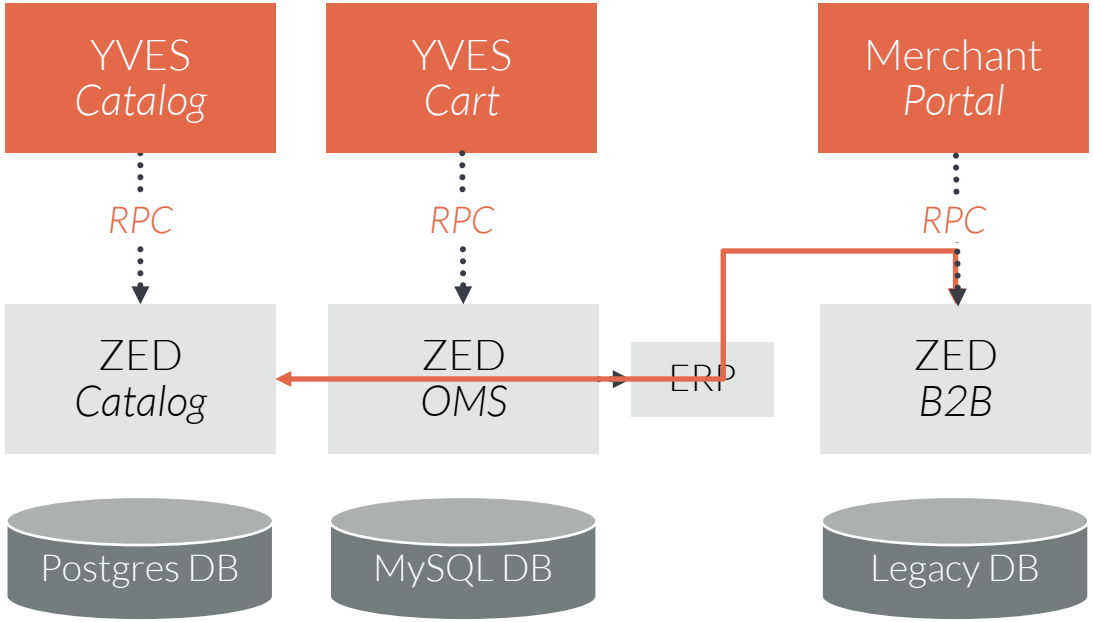
# How to extend / integrate Spryker




Spryker in use:

- Integrate with external systems via state machine commands
- Introduce new bundles with provided architectural layout
- Use and extend API of existing bundles on project / store level
- Everything is replaceable by consequent use of dependency containers and Interfaces
- External services use REST API

# Spryker enables micro-services




- Framework approach allows for independent services
- Decoupled bundles and defined APIs make integration and separation into services easy

# System Requirements

Spryker makes no assumptions about external systems ...

... but comes ready with the right implementations

| Item | Requirements | Recommended |
|---|---|---|
| **Webserver** | Any PHP enabled Webserver | Php-fpm with nginx |
| **KV Storage** | Any KV-Storage with Set, Get, Multiget | Redis |
| **Search** | Search Engine with faceting | Elasticsearch |
| **DB** | DB supported by Propel (MySQL, PostgreSQL, Oracle, MSSQL) | PostgresSQL or MySQL |
| **OS** | Any OS running a Webserver | Linux |

# Delivery

# Spryker delivery contents

GIT repositories: all bundles as separate components

Infrastructure: deployment templates via saltstack

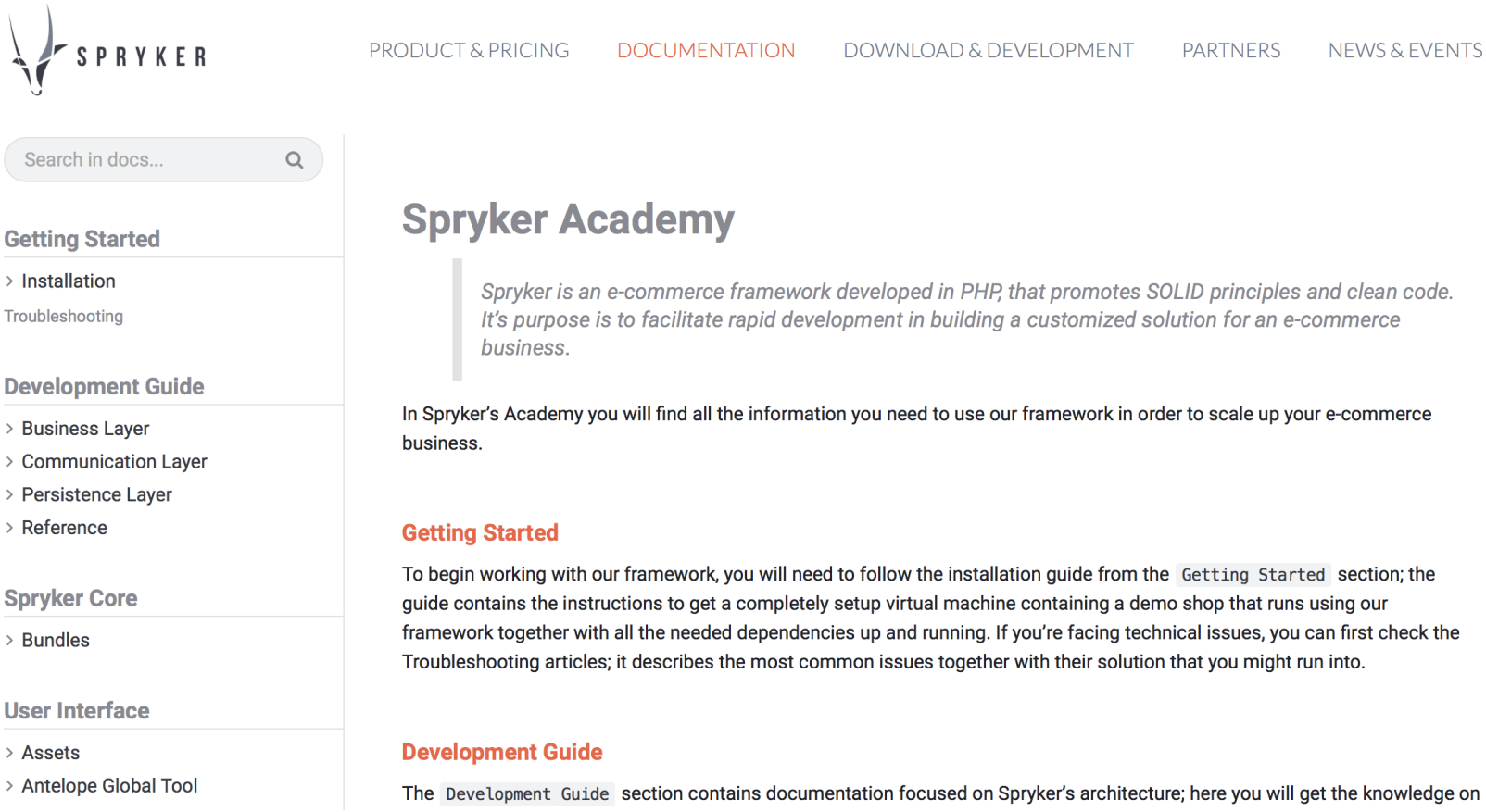
Demo-shop via Developer-VM: full stack development shop template with all components

Spryker Academy: access to documentation and project controlling templates

# What makes development with Spryker so fast?

- Architectural decisions are provided by the framework
- Best practices from 100+ implementations
- Consequent de-coupling of bundles supports fast iterations
- Atomic release process
- “No magic methods” enable easy debugging
- Best in class code quality
- Coherent architecture (Frontend, Backend, Extensions)

# And yes, we have documentation☺



# Spryker`s key promises

Productivity

Performance

Flexibility

# What makes Spryker development so fast?

- Architectural decisions are provided by the framework
- Best practices from 100+ implementations
- Consequent de-coupling of bundles supports fast iterations
- Atomic release process
- “No magic methods” enable easy debugging
- Best in class code quality
- Coherent architecture (Frontend, Backend, Extensions)

# Spryker innovations

- Architecture and approach
- State machines (OMS), scalability, deployment
- Release model
- Pricing
- Quality and speed
- Development productivity
- Contract terms
- And much more….

SPRYKER

# Technology Summary

Framework approach

Modular monolith

Frontend / Backend separation

Symfony / Silex based PHP Application

Developer friendly

SPRYKER

You cannot solve 21st century e-commerce problems with 20th century technology

www.spryker.com

@sprysys