Modular Application Architecture

DPC 2011

Kore Nordmann (@koredn)
Tobias Schlitt (@tobySen)

May 21, 2011

About us

- Degree in computer sience in 2010
- Each more than 10 years of professional PHP
- Open source enthusiasts
- Contributing to various FLOSS projects
- 2 of 3 founders of **Qafoo GmbH**, which provides **Services all around high quality PHP**

Outline

Motivation

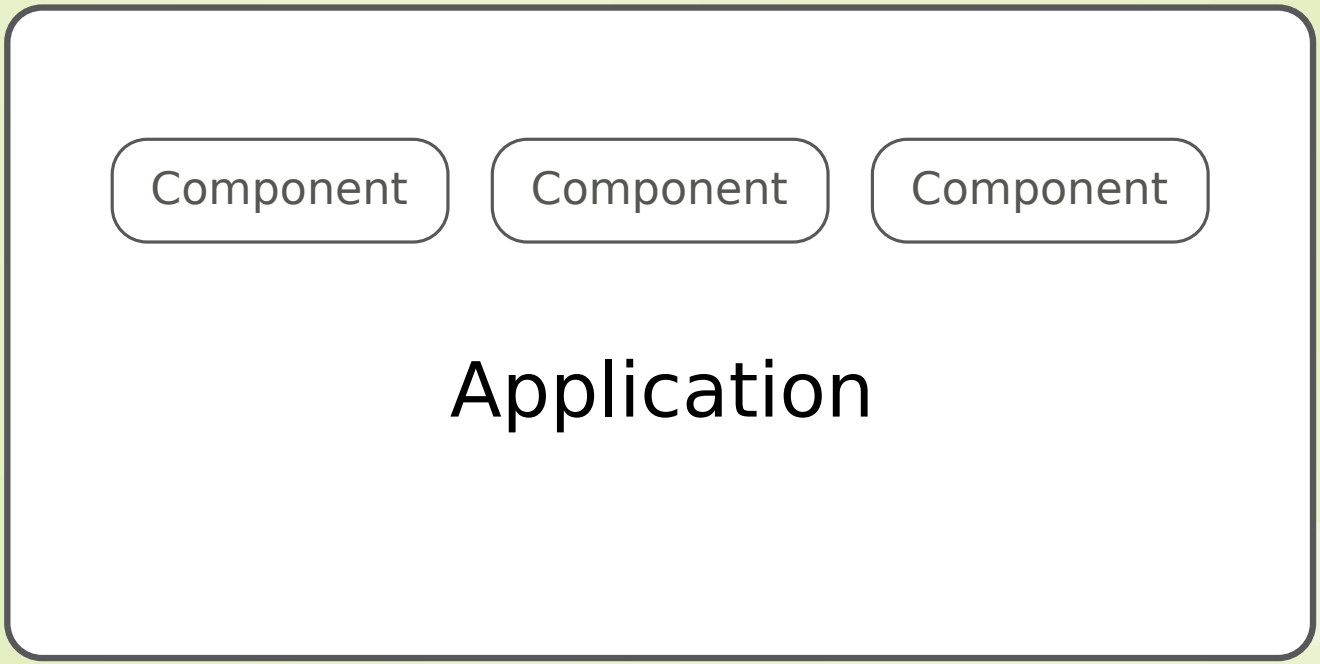
Resources

Approaches

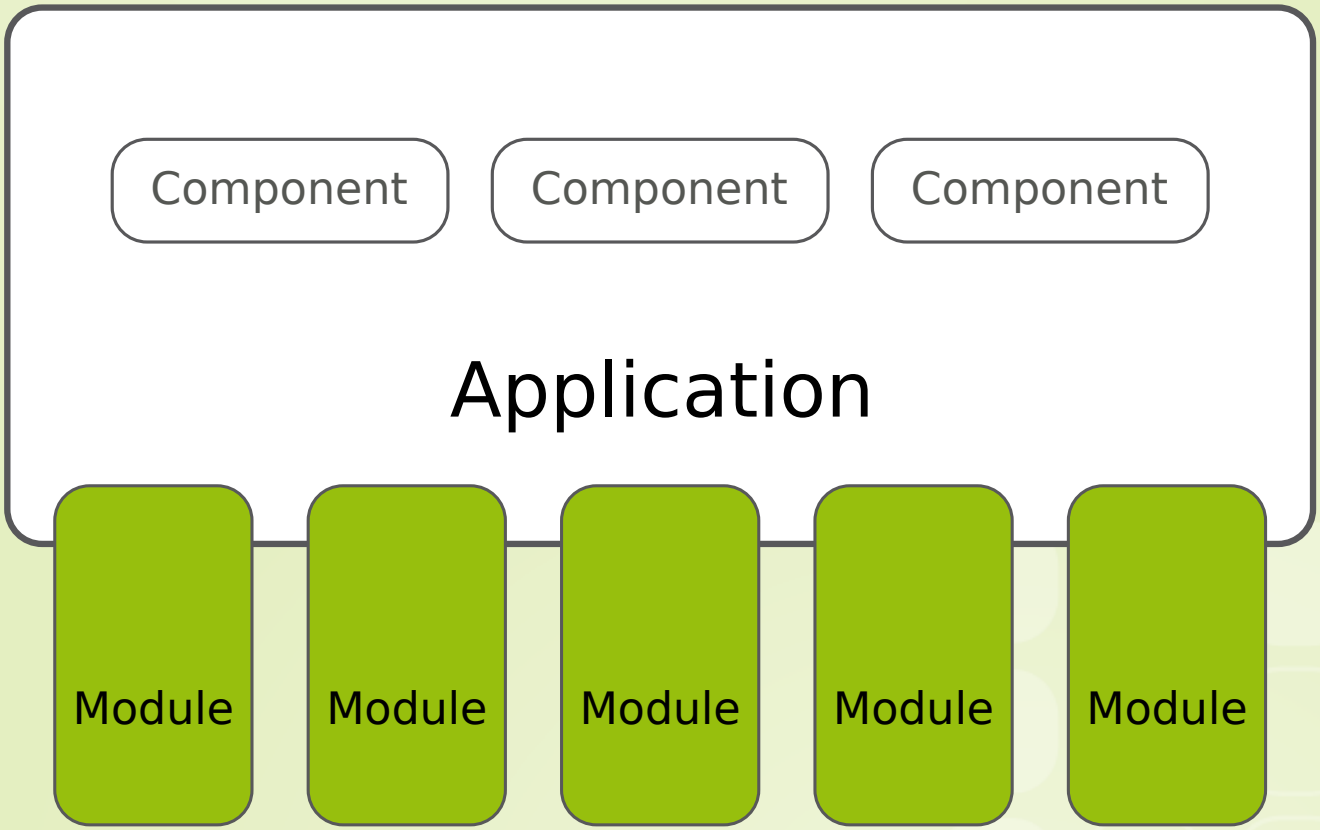
Real world

Summary

Application



Modules


Component
Component
Component
Application
Module
Module
Module
Module
Module

Why modules?

- Need for customization
  - Custom setup for customers
  - 3rd party extensions

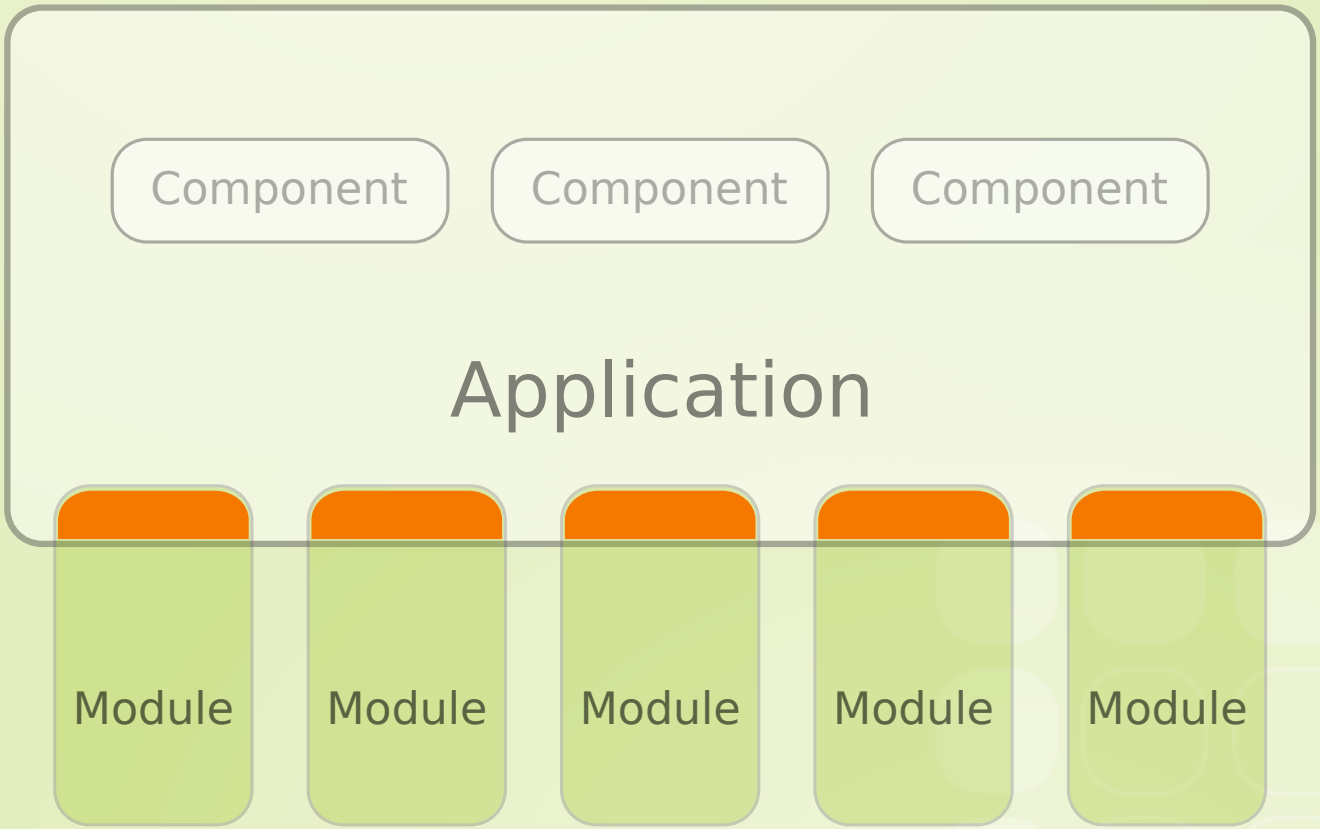
Why modules?

- Need for customization
  - Custom setup for customers
  - 3rd party extensions
- Develop modules seperately from main application
  - External developers
  - Seperate release cycles

Why modules?

- Need for customization
  - Custom setup for customers
  - 3rd party extensions
- Develop modules seperately from main application
  - External developers
  - Seperate release cycles
- Slag the main application
  - Raise maintainability

Essential


Component
Component
Component
Application
Module
Module
Module
Module
Module

Challenges

- Module structure
- Registration / configuration
- Handling ressources
- Interaction with core

Challenges

- Module structure ✓
- Registration / configuration
- Handling ressources
- Interaction with core

Challenges

- Module structure ✓
- Registration / configuration ✓
- Handling ressources
- Interaction with core

Challenges

- Module structure ✓
- Registration / configuration ✓
- Handling ressources $\longrightarrow$
- Interaction with core

Challenges

- Module structure ✓
- Registration / configuration ✓
- Handling ressources $\longrightarrow$
- Interaction with core $\Longrightarrow$

Outline

Motivation

Resources

Approaches

Real world

Summary

Dealing with resources

- Typical module resources
  - Templates
  - Translations
  - Images
  - CSS

Dealing with resources

- Typical module resources
  - Templates
  - Translations
  - Images
  - CSS
- Resources handled by code are “easy”
  - Register “overrides“

Dealing with resources

- Typical module resources
  - Templates
  - Translations
  - Images
  - CSS
- Resources handled by code are “easy”
  - Register “overrides“
- Static file resources are not
  - Put modules in a web accessible path?
  - Copy / link static files to `htdocs/` ?
  - Pipe static files through PHP?
  - Webserver configuration?

Outline

Motivation

Resources

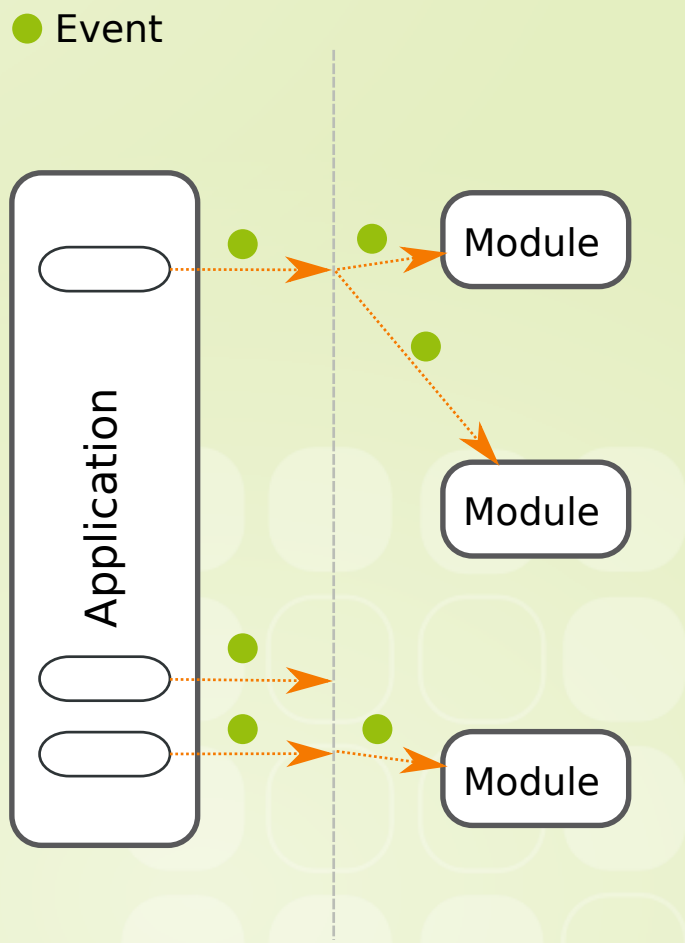
Approaches

Real world

Summary

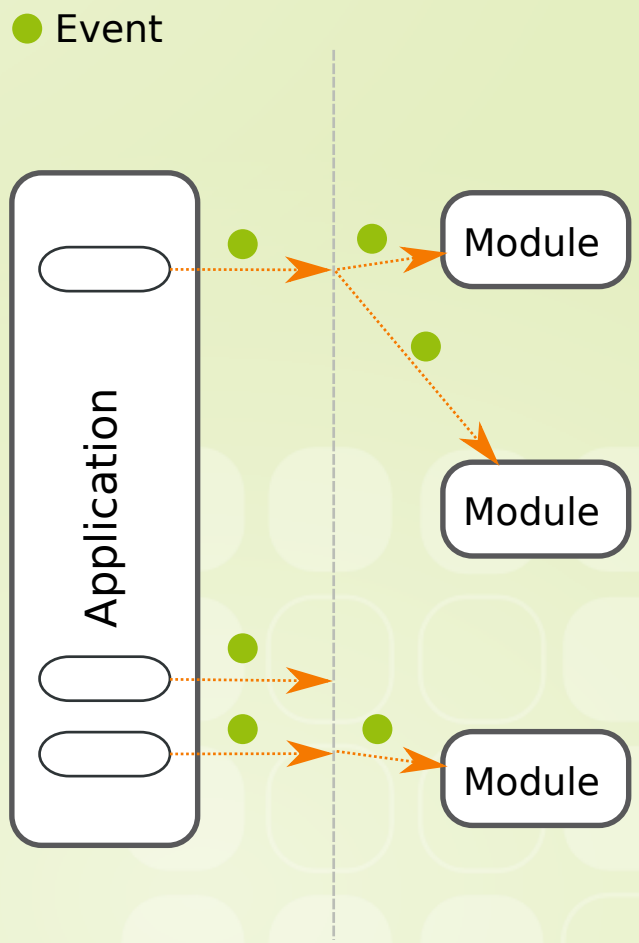
Event handling

- Interaction
  - Modules register for event types
  - Events "thrown" (by application or module)
  - Application routes event (including data) to modules transparently



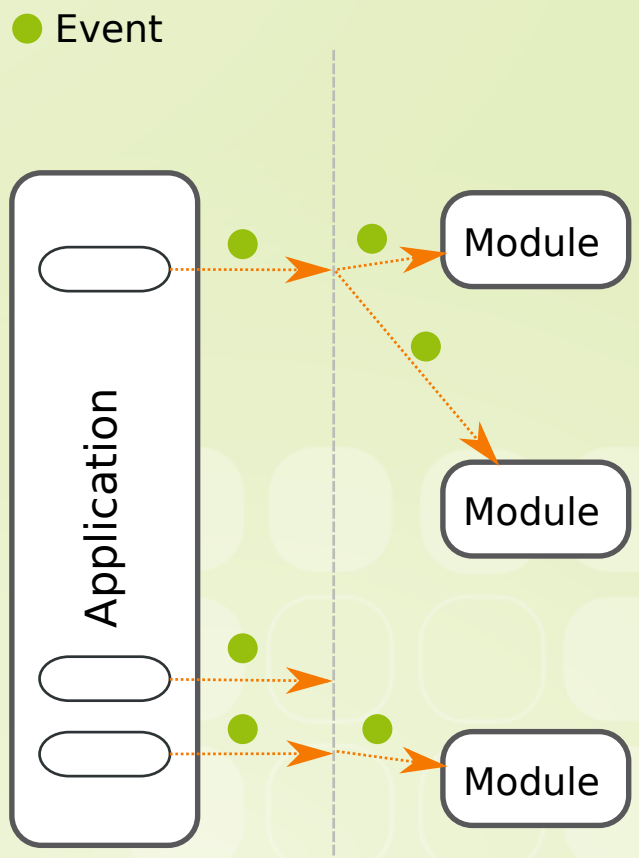
Event handling

- Interaction
- Modules register for event types



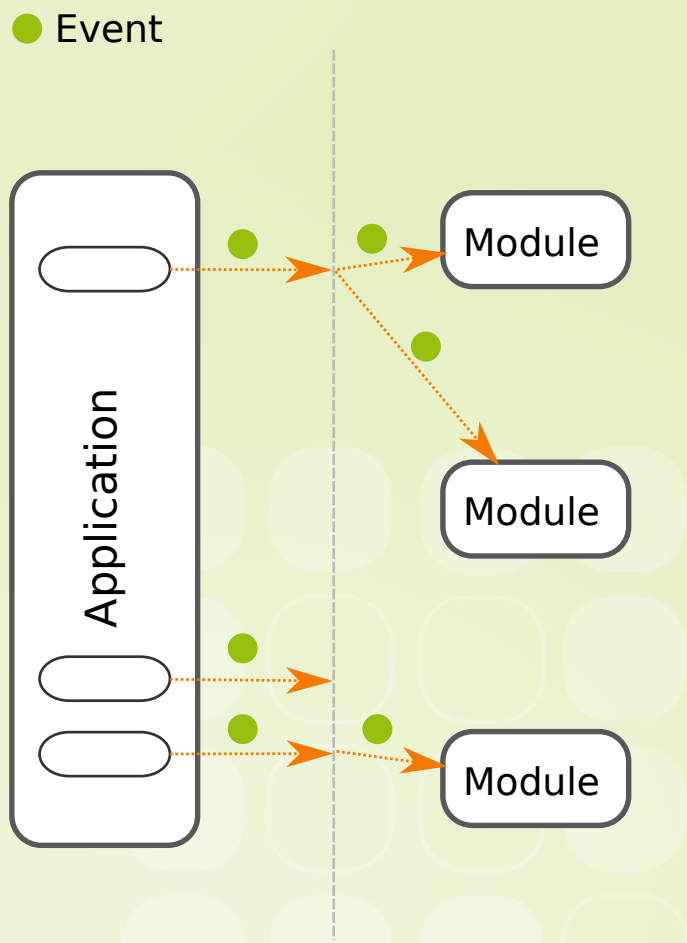
Event handling

- Interaction
- Modules register for event types
- Events “thrown” (by core or module)



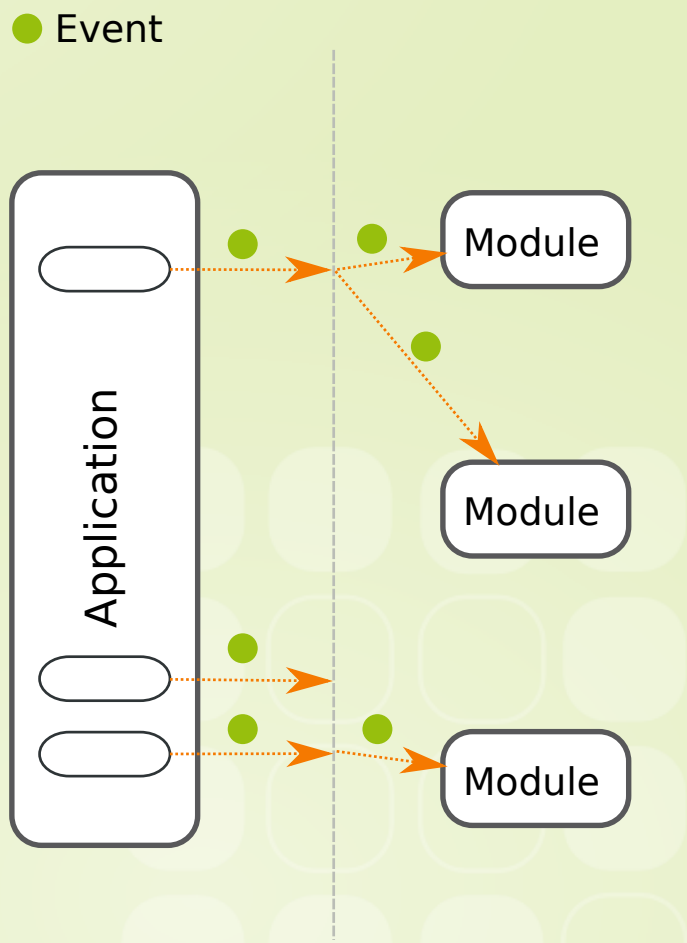
Event handling

- Interaction
- Modules register for event types
- Events "thrown" (by core or module)
- Registered modules informed about event (maybe including data)



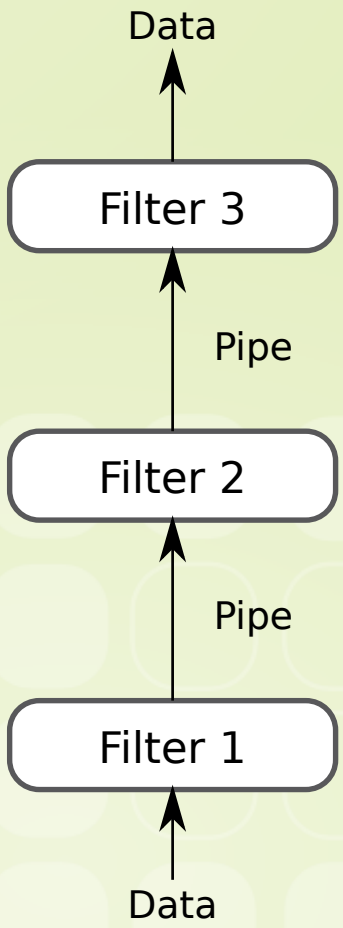
Event handling

- Interaction
- Modules register for event types
- Events “thrown” (by core or module)
- Registered modules informed about event (maybe including data)
- Optionally transparent



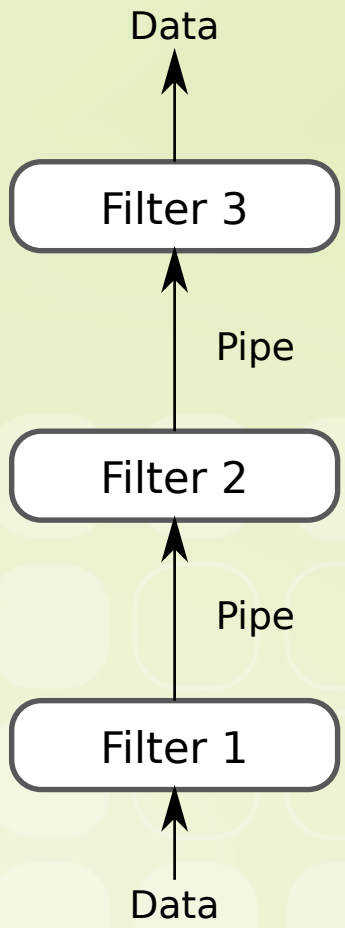
Data handling

- Data processing
  - Pipes
    - Transport data
  - Filters
    - Manipulate data



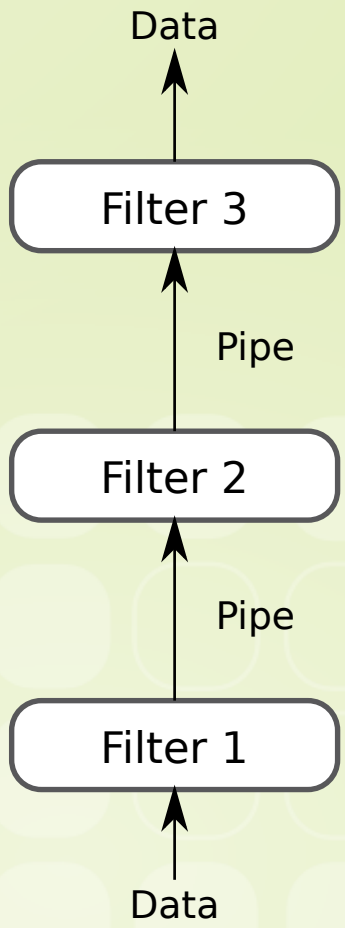
Data handling

- Data processing
- Pipes
  - Transport data
- Filters
  - Manipulate data



Data handling

- Data processing
- Pipes
  - Transport data
- Filters
  - Manipulate data



Outline

Motivation

Resources

Approaches

Real world

Summary

Outline

Real world
- Pipes & Filters
- Events
- Others

Popoon

```
1  <?xml version="1.0"?>
2
3  <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
4  <map:pipelines>
5    <map:pipeline>
6      <map:match type="uri" pattern="examples.tgz">
7        <map:read type="tgz" src="." name="examples.tgz"/>
8      </map:match>
9    </map:pipeline>
10
11   <map:pipeline >
12     <map:generate type="xmlfile" src="examples.xml"/>
13     <map:transform type="libxslt" src="examples.xsl"/>
14     <map:serialize type="html"/>
15   </map:pipeline>
16 </map:pipelines>
17 </map:sitemap>
```

Popoon

```
1  <?xml version="1.0"?>
2
3  <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
4  <map:pipelines>
5    <map:pipeline>
6      <map:match type="uri" pattern="examples.tgz">
7        <map:read type="tgz" src="." name="examples.tgz" />
8      </map:match>
9    </map:pipeline>
10
11   <map:pipeline >
12     <map:generate type="xmlfile" src="examples.xml" />
13     <map:transform type="libxslt" src="examples.xsl" />
14     <map:serialize type="html" />
15   </map:pipeline>
16 </map:pipelines>
17 </map:sitemap>
```

Popoon

```
1  <?xml version="1.0"?>
2
3  <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
4  <map:pipelines>
5    <map:pipeline>
6      <map:match type="uri" pattern="examples.tgz">
7        <map:read type="tgz" src="." name="examples.tgz"/>
8      </map:match>
9    </map:pipeline>
10
11   <map:pipeline >
12     <map:generate type="xmlfile" src="examples.xml"/>
13     <map:transform type="libxslt" src="examples.xsl"/>
14     <map:serialize type="html"/>
15   </map:pipeline>
16 </map:pipelines>
17 </map:sitemap>
```

Popoon

```
1  <?xml version="1.0"?>
2
3  <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
4  <map:pipelines>
5    <map:pipeline>
6      <map:match type="uri" pattern="examples.tgz">
7        <map:read type="tgz" src="." name="examples.tgz"/>
8      </map:match>
9    </map:pipeline>
10
11   <map:pipeline >
12     <map:generate type="xmlfile" src="examples.xml"/>
13     <map:transform type="libxslt" src="examples.xsl"/>
14     <map:serialize type="html"/>
15   </map:pipeline>
16 </map:pipelines>
17 </map:sitemap>
```

Popoon

```
1  <?xml version="1.0"?>
2
3  <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
4  <map:pipelines>
5    <map:pipeline>
6      <map:match type="uri" pattern="examples.tgz">
7        <map:read type="tgz" src="." name="examples.tgz" />
8      </map:match>
9    </map:pipeline>
10
11   <map:pipeline >
12     <map:generate type="xmlfile" src="examples.xml" />
13     <map:transform type="libxslt" src="examples.xsl" />
14     <map:serialize type="html" />
15   </map:pipeline>
16 </map:pipelines>
17 </map:sitemap>
```

Pro & Contra

- Benefits:
  - Clean architectural approach
  - Might gain high re-usability
- Drawbacks:
  - Might break data easily
  - What forces linear code flow

Pro & Contra

- Benefits:
  - Clean architectural approach
  - Might gain high re-usability
- Drawbacks:
  - Might break data easily
  - What forces linear code flow

Pro & Contra

- Benefits:
  - Clean architectural approach
  - Might gain high re-usability
- Drawbacks:
  - Might break data easily
  - What forces linear code flow

Pro & Contra

- Benefits:
  - Clean architectural approach
  - Might gain high re-usability
- Drawbacks:
  - Filters might break data easily

Pro & Contra

- Benefits:
  - Clean architectural approach
  - Might gain high re-usability
- Drawbacks:
  - Filters might break data easily
  - Somewhat forces linear code flow

Outline

Real world
- Pipes & Filters
- Events
- Others

Subject-Observer

```php
<?php

class Subject
{
    public function doSomething()
    {
        $this->notify( 'doSomethingStart' );
        // ...
        $this->notify( 'doSomethingEnd' );
    }
}
```

Subject-Observer

```php
<?php

class Subject
{
    protected $observers = array();

    public function addObserver( Observer $observer )
    {
        $this->observers[] = $observer;
    }

    public function notify( $event, $data = null )
    {
        foreach ( $this->observers as $observer )
        {
            $observer->$event( $data );
        }
    }

    public function doSomething()
    {
        $this->notify( 'doSomethingStart' );
        // ...
        $this->notify( 'doSomethingEnd' );
    }
}
```

Subject-Observer

```php
<?php


class Observer
{
    public function doSomethingStart()
    {
        // ...
    }

    public function doSomethingEnd()
    {
        // ...
    }
}
```

Pro & Contra

- Benefits:
- Drawbacks:

Pro & Contra

- Benefits:
  - Transparent – any number of observers can register
  - (Documented) clearly defined extension API
    - optionally with clearly defined transmitted data structs
- Drawbacks:
  - Intransparent – you have no idea how long a singal will take to process
  - Limited to defined extension points
  - Requires **implementation in each subject**

Pro & Contra

- Benefits:
  - Transparent – any number of observers can register
  - (Documented) clearly defined extension API
- Drawbacks:

Pro & Contra

- Benefits:
  - Transparent – any number of observers can register
  - (Documented) clearly defined extension API
  - . . . optionally with clearly defined transmitted data structs
- Drawbacks:

Pro & Contra

- Benefits:
  - Transparent – any number of observers can register
  - (Documented) clearly defined extension API
  - . . . optionally with clearly defined transmitted data structs
- Drawbacks:
  - Fully transparent – you have no idea how long a singal will take to process

Pro & Contra

- Benefits:
  - Transparent – any number of observers can register
  - (Documented) clearly defined extension API
  - . . . optionally with clearly defined transmitted data structs
- Drawbacks:
  - Fully transparent – you have no idea how long a singal will take to process
  - Limited to defined extension points
  - requires implementation in each subject

Pro & Contra

- Benefits:
  - Transparent – any number of observers can register
  - (Documented) clearly defined extension API
  - ... optionally with clearly defined transmitted data structs
- Drawbacks:
  - Fully transparent – you have no idea how long a singal will take to process
  - Limited to defined extension points
  - **Requires implementation in each subject**

Signal slot

```php
<?php

$handler = new arbitSignalSlot();

$handler->register( 'signalA', array( new myModule(), 'handleSignalA' ) );
$handler->register( 'signalA', array( new yourModule(), 'handleSignalA' ) );

// In module c
$handler->emit( 'signalA', new signalADataStruct( /* ... */ ) );

// Now all modules registerd for this signal are called with the provided data
class myModule
{
    public function handleSignalA( $name, signalADataStruct $data )
    {
        // ...
    }
}
```

Pro & Contra

- Benefits:
  - Fully transparent – nobody needs to know who
  - (Documented) clearly defiend extension API
  - optionally with clearly defined transmitted data structs
  - can easily be made asynchronous
- Drawbacks:
  - Not transparent – you have no idea how long a singal will take to process
  - Hard to defiend extension points

Pro & Contra

- Benefits:
  - Fully transparent – nobody needs to know who is called
  - (Documented) clearly defiend extension API
  - optionally with clearly defined transmitted data structs
  - Can easily be made asynchronous
- Drawbacks:
  - Not transparent – you have no idea how long a singal will take to process
  - Hard to defiend extension points

Pro & Contra

- Benefits:
  - Fully transparent – nobody needs to know who is called
  - (Documented) clearly defiend extension API
- Drawbacks:

Pro & Contra

- Benefits:
  - Fully transparent – nobody needs to know who is called
  - (Documented) clearly defiend extension API
  - . . . optionally with clearly defined transmitted data structs
- Drawbacks:

Pro & Contra

- Benefits:
  - Fully transparent – nobody needs to know who is called
  - (Documented) clearly defiend extension API
  - ... optionally with clearly defined transmitted data structs
  - Can easily be made asynchronous
- Drawbacks:
  - Fully transparent – you have no idea how long a singal will take to process
  - Hard to defiend extension points

Pro & Contra

- Benefits:
  - Fully transparent – nobody needs to know who is called
  - (Documented) clearly defiend extension API
  - . . . optionally with clearly defined transmitted data structs
  - Can easily be made asynchronous
- Drawbacks:
  - Fully transparent – you have no idea how long a singal will take to process

Pro & Contra

- Benefits:
  - Fully transparent – nobody needs to know who is called
  - (Documented) clearly defiend extension API
  - ... optionally with clearly defined transmitted data structs
  - Can easily be made asynchronous
- Drawbacks:
  - Fully transparent – you have no idea how long a singal will take to process
  - Limited to defiend extension points

Outline

Real world
- Pipes & Filters
- Events
- Others

Serendipity hook announcement

```php
<?php

// ... in CSS code ...

// $out is CSS string
serendipity_plugin_api::hook_event($css_hook, $out);

echo $out;

// ... in entry display code ...

// $entry is blog entry
// $addData is meta data
serendipity_plugin_api::hook_event('frontend_display', $entry, $addData);
```

Serendipity hook reaction

```
18  function event_hook($event, &$bag, &$eventData) {
19      global $serendipity;
20
21      $hooks = &$bag->get('event_hooks');
22
23      if (isset($hooks[$event])) {
24          switch($event) {
25              case 'frontend_display':
26                  if ( $condition /* ... */) {
27                      $element = $temp['element'];
28                      $eventData[$element] = $this->bbcode(
29                          $eventData[$element]
30                      );
31                  }
32                  return true;
33                  break;
```

Serendipity hook reaction

```
18      function event_hook($event, &$bag, &$eventData) {
19          global $serendipity;
20
21          $hooks = &$bag->get('event_hooks');
22
23          if (isset($hooks[$event])) {
24              switch($event) {
35                  case 'css':
36                      if (strpos($eventData, '.bb-code') !== false) {
37                          // class exists in CSS, so a user has customized it and
                                we don't need default
38                          return true;
39                      }
40  ?>
41  .bb-quote, .bb-code, .bb-php, .bb-code-title, .bb-php-title {
42      margin-left: 20px;
43      margin-right: 20px;
44      /* ... */
45  }
46  /* ... */
47  <?php
48                  return true;
49                  break;
```

Pro & Contra

- Benefits:
  - High flexibility
  - Low coding efforts
- Drawbacks:
  - easily break hook data
  - data formats
  - substitution principle" limits what you are allowed to

Pro & Contra

- Benefits:
  - High flexibility
  - Low coding efforts
- Drawbacks:
  - easily break hook data
  - ed data formats
  - substitution principle" limits what you are allowed to

Pro & Contra

- Benefits:
  - High flexibility
  - Low coding efforts
- Drawbacks:

Pro & Contra

- Benefits:
  - High flexibility
  - Low coding efforts
- Drawbacks:
  - Plugin can easily break hook data

Pro & Contra

- Benefits:
  - High flexibility
  - Low coding efforts
- Drawbacks:
  - Plugin can easily break hook data
  - No defined data formats

Pro & Contra

- Benefits:
  - High flexibility
  - Low coding efforts
- Drawbacks:
  - Plugin can easily break hook data
  - No defined data formats
  - “Liskov substitution principle” limits what you are allowed to do

Patching the source

- The naive approach

Patching the source

- The naive approach
  - Works suprisingly well for some of the largest module ecosystems: phpBB

phpBB MODx format

```
1  <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
2  <?xml-stylesheet type="text/xsl" href="1.2.0/modx.prosilver.en.xsl"?>
3  <mod xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.
       phpbb.com/mods/xml/modx-1.2.0.xsd">
4      <header>
42     </header>
43         <open src="index.php">
44             <edit>
45                 <comment lang="en">Here is a comment</comment>
46                 <comment lang="nl">Hier is een stukje commentaar</comment>
47                 <find>text to find</find>
48                 <action type="replace-with">text to be replaced with</action>
49             </edit>
50             <edit>
51                 <find>text to find</find>
52                 <action type="after-add">text to be added on the line after</
                       action>
53             </edit>
54             <edit>
55                 <find>text to find</find>
56                 <action type="before-add">text to be added on the line before</
                       action>
57             </edit>
```

Pro & Contra

- Benefits:
  - Trivial to get started with (high "hackability")
  - You can change anything
- Drawbacks:
  - ...itely break
  - ...d to unparsable code
  - ...lex modules require deep knowledge

Pro & Contra

- Benefits:
  - Trivial to get started with (high “hackability”)
  - You can change anything
- Drawbacks:
  - You can trivially break
  - Leads to unparsable code
  - Complex modules require deep knowledge

Pro & Contra

- Benefits:
  - Trivial to get started with (high "hackability")
  - You can change anything
- Drawbacks:

Pro & Contra

- Benefits:
  - Trivial to get started with (high “hackability”)
  - You can change anything
- Drawbacks:
  - Will definitely break
  - Leads to unparsable code
  - Complex modules require deep knowledge

Pro & Contra

- Benefits:
  - Trivial to get started with (high “hackability”)
  - You can change anything
- Drawbacks:
  - Will definitely break
  - Can lead to unparsable code

Pro & Contra

- Benefits:
  - Trivial to get started with (high “hackability”)
  - You can change anything
- Drawbacks:
  - Will definitely break
  - Can lead to unparsable code
  - Complex modules require deep knowledge

Inheritance

- Generally: Use Aggregation instead of inheritance for code re-use.

Inheritance

- Generally: Use Aggregation instead of inheritance for code re-use.
- Oxid eSales (OS shop software) has an interesting extension model build entirely on inheritance
  - Any number of modules can inherit from “any” class . . .

Inheritance

- Generally: Use Aggregation instead of inheritance for code re-use.
- Oxid eSales (OS shop software) has an interesting extension model build entirely on inheritance
  - Any number of modules can inherit from “any” class . . .
  - . . . and each inheriting class will be used anywhere the original object would be used.

. . . can that be possible?

Inheritance

- Generally: Use Aggregation instead of inheritance for code re-use.
- Oxid eSales (OS shop software) has an interesting extension model build entirely on inheritance
  - Any number of modules can inherit from “any” class . . .
  - . . . and each inheriting class will be used anywhere the original object would be used.
  - *How can that be possible?*

Modular inheritance

- Objects are instantiated with a special function instead of the `new` operator.
- Inheritance graph is created on-the-fly by generating intemediate classes

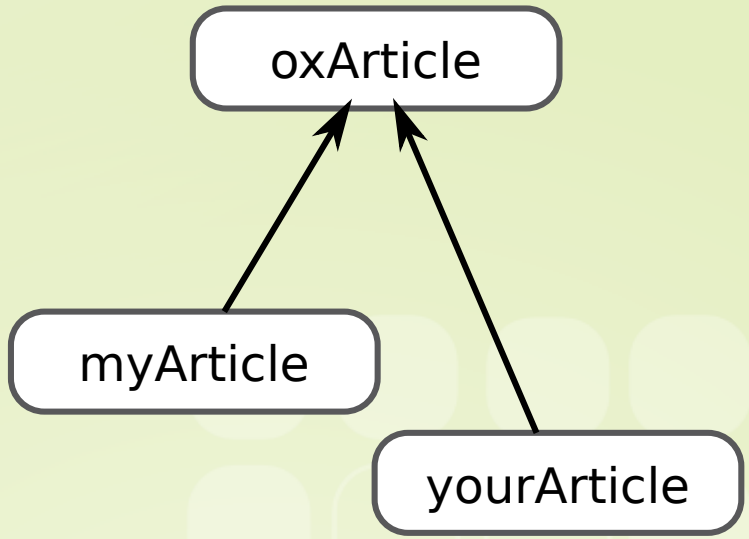
Example

```php
<?php

class oxArticle
{
    public function calculatePrice()
    {
        // ...
    }
}

class myArticle
    extends oxArticle
{
    // ...
}

class yourArticle
    extends oxArticle
{
    // ...
}
```



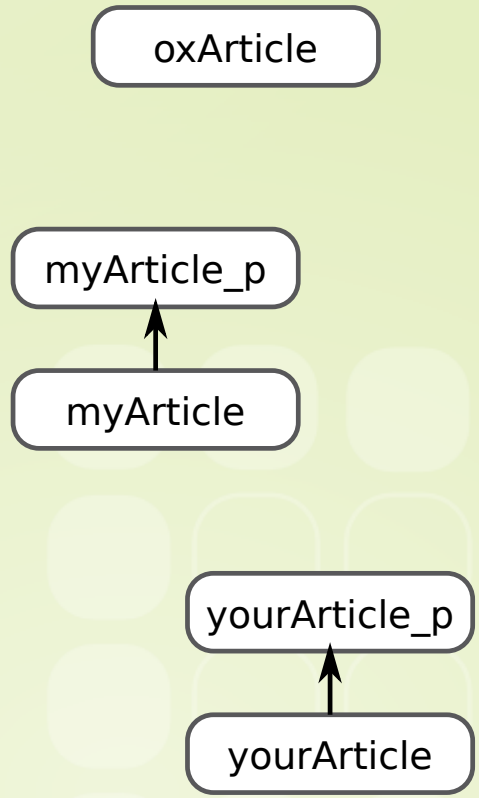
Example

```php
<?php

class oxArticle
{
    public function calculatePrice()
    {
        // ...
    }
}

class myArticle
    extends myArticle_parent
{
    // ...
}

class yourArticle
    extends yourArticle_parent
{
    // ...
}
```



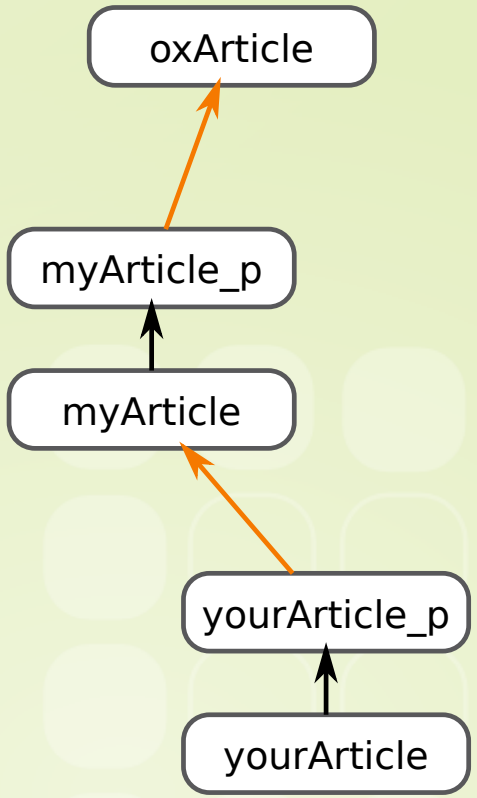
Example

```php
<?php

class oxArticle
{
    public function calculatePrice()
    {
        // ...
    }
}

class myArticle
    extends myArticle_parent
{
    // ...
}

class yourArticle
    extends yourArticle_parent
{
    // ...
}
```



Pro & Contra

- Benefits:
- Drawbacks:

Pro & Contra

- Benefits:
  - You can extend about everything...
- Drawbacks:

Pro & Contra

- Benefits:
  - You can extend about everything...
- Drawbacks:
  - About everything will be extended...

Pro & Contra

- Benefits:
  - You can extend about everything...
- Drawbacks:
  - About everything will be extended...
  - You may not use the `new` operator – but use something like: `oxNew( "oxArticle" )`

Pro & Contra

- Benefits:
  - You can extend about everything...
- Drawbacks:
  - About everything will be extended...
  - You may not use the `new` operator – but use something like: `oxNew( "oxArticle" )`
  - Violates object-oriented design principles

Pro & Contra

- Benefits:
  - You can extend about everything...
- Drawbacks:
  - About everything will be extended...
  - You may not use the `new` operator – but use something like: `oxNew( "oxArticle" )`
  - Violates object-oriented design principles
  - Non-enforcable constraints (`parent::method()`)
  - Liskov substitution principle limits what you are allowed to do

Pro & Contra

- Benefits:
  - You can extend about everything...
- Drawbacks:
  - About everything will be extended...
  - You may not use the `new` operator – but use something like: `oxNew( "oxArticle" )`
  - Violates object-oriented design principles
  - Non-enforcable constraints (`parent::method()`)
  - Liskov substitution principle limits what you are allowed to do

Outline

Motivation

Resources

Approaches

Real world

Summary

Summary

- Patching
- Hooks
- Pipes & Filters
- Inheritance
- Subject-Observer
- Signal-Slot

Thanks for listening

Please rate this talk at
`http://joind.in/3249`

Slides will be online (soon)
`http://talks.qafoo.com`

Thanks for listening

Please rate this talk at
`http://joind.in/3249`

Stay in touch

- Kore Nordmann
- kore@qafoo.com
- @koredn / @qafoo

- Tobias Schlitt
- toby@qafoo.com
- @tobySen / @qafoo

Rent a PHP quality expert:
`http://qafoo.com`