```php
<?php

namespace {

    use SilverStripe\CMS\Model\SiteTree;
    use SilverStripe\ORM\HasManyList;

    /**
     * @method PageLike[]|HasManyList Likes
     */
    class Page extends SiteTree
    {
        private static array $has_many = [
            'Likes' => PageLike::class,
        ];
    }
}
```

```php
<?php

namespace {

    use App\Models\PageLike;
    use SilverStripe\Control\HTTPResponse;
    use SilverStripe\Control\HTTPRequest;
    use SilverStripe\CMS\Controllers\ContentController;

    class PageController extends ContentController
    {
        private static array $allowed_actions = [
            'likethispage',
        ];

        public function likethispage(HTTPRequest $request): HT
        {
            $ip = $request->getIP();

            if (!$this->Likes()->filter('IP', $ip)->exists()) {
                $pageLike = PageLike::create();
                $pageLike->IP = $ip;
                $this->Likes()->add($pageLike);
            }

            $this->redirectBack();
        }
    }
}
```

SILVERSTRIPE

THE GOOD THE BAD THE UGLY

and the awesome parts

Stripecon Europe 2024

RELAXT

```php
<?php

namespace {

    use SilverStripe\CMS\Model\SiteTree;
    use SilverStripe\ORM\HasManyList;

    /**
     * @method PageLike[]|HasManyList Likes
     */
    class Page extends SiteTree
    {
        private static array $has_many = [
            'Likes' => PageLike::class,
        ];
    }
}
```

Is it good or even awesome? It shows a strength of Silverstripe - getting custom functionality at a low cost (in terms of lines of code). But as always there are some tradeoffs.

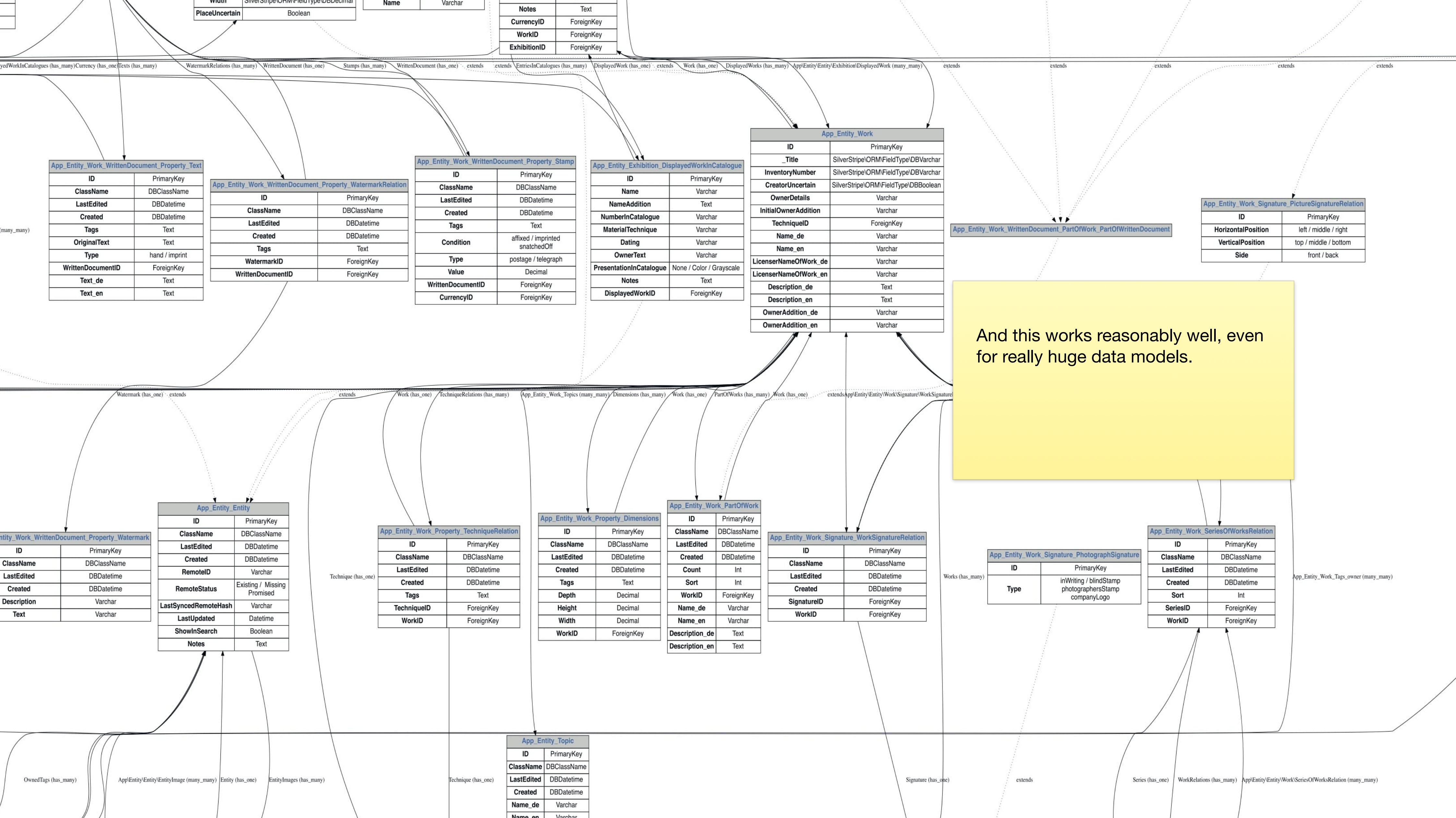
```php
<?php

namespace {

    use App\Models\PageLike;
    use SilverStripe\Control\HTTPResponse;
    use SilverStripe\Control\HTTPRequest;
    use SilverStripe\CMS\Controllers\ContentController;

    class PageController extends ContentController
    {
        private static array $allowed_actions = [
            'likethispage',
        ];

        public function likethispage(HTTPRequest $request): HTTPResponse
        {
            $ip = $request->getIP();

            if (!$this->Likes()->filter('IP', $ip)->exists()) {
                $pageLike = PageLike::create();
                $pageLike->IP = $ip;
                $this->Likes()->add($pageLike);
            }

            $this->redirectBack();
        }
    }
}
```

```php
<?php

namespace App\Models;

use Page;
use SilverStripe\ORM\DataObject;

/**
 * @property string IP
 * @method Page Page
 */
class PageLike extends DataObject
{
    private static array $db = [
        'IP' => 'Varchar(45)',
    ];

    private static array $belongs_to = [
        'Page' => Page::class,
    ];
}
```

Stephan Bauer

That's me. I'm part of relaxt (https://www.relaxt.at).

ENTITIES
ORM



And this works reasonably well, even for really huge data models.
App_Entity_Work_WrittenDocument_Property_Text
ID PrimaryKey
ClassName DBClassName
LastEdited DBDatetime
Created DBDatetime
Tags Text
OriginalText Text
Type hand / imprint
WrittenDocumentID ForeignKey
Text_de Text
Text_en Text
App_Entity_Work_WrittenDocument_Property_WatermarkRelation
ID PrimaryKey
ClassName DBClassName
LastEdited DBDatetime
Created DBDatetime
Tags Text
WatermarkID ForeignKey
WrittenDocumentID ForeignKey
App_Entity_Work_WrittenDocument_Property_Stamp
ID PrimaryKey
ClassName DBClassName
LastEdited DBDatetime
Created DBDatetime
Tags Text
Condition affixed / imprinted snatchedOff
Type postage / telegraph
Value Decimal
WrittenDocumentID ForeignKey
CurrencyID ForeignKey
App_Entity_Exhibition_DisplayedWorkInCatalogue
ID PrimaryKey
Name Varchar
NameAddition Text
NumberInCatalogue Varchar
MaterialTechnique Varchar
Dating Varchar
OwnerText Varchar
PresentationInCatalogue None / Color / Grayscale
Notes Text
DisplayedWorkID ForeignKey
App_Entity_Work
ID PrimaryKey
_Title SilverStripe\ORM\FieldType\DBVarchar
InventoryNumber SilverStripe\ORM\FieldType\DBVarchar
CreatorUncertain SilverStripe\ORM\FieldType\DBBoolean
OwnerDetails Varchar
InitialOwnerAddition Varchar
TechniqueID ForeignKey
Name_de Varchar
Name_en Varchar
LicenserNameOfWork_de Varchar
LicenserNameOfWork_en Varchar
Description_de Text
Description_en Text
OwnerAddition_de Varchar
OwnerAddition_en Varchar
App_Entity_Work_WrittenDocument_PartOfWork_PartOfWrittenDocument
App_Entity_Work_Signature_PictureSignatureRelation
ID PrimaryKey
HorizontalPosition left / middle / right
VerticalPosition top / middle / bottom
Side front / back
App_Entity_Entity
ID PrimaryKey
ClassName DBClassName
LastEdited DBDatetime
Created DBDatetime
RemoteID Varchar
RemoteStatus Existing / Missing Promised
LastSyncedRemoteHash Varchar
LastUpdated Datetime
ShowInSearch Boolean
Notes Text
App_Entity_Work_Property_TechniqueRelation
ID PrimaryKey
ClassName DBClassName
LastEdited DBDatetime
Created DBDatetime
Tags Text
TechniqueID ForeignKey
WorkID ForeignKey
App_Entity_Work_Property_Dimensions
ID PrimaryKey
ClassName DBClassName
LastEdited DBDatetime
Created DBDatetime
Tags Text
Depth Decimal
Height Decimal
Width Decimal
WorkID ForeignKey
App_Entity_Work_PartOfWork
ID PrimaryKey
ClassName DBClassName
LastEdited DBDatetime
Created DBDatetime
Count Int
Sort Int
WorkID ForeignKey
Name_de Varchar
Name_en Varchar
Description_de Text
Description_en Text
App_Entity_Work_Signature_WorkSignatureRelation
ID PrimaryKey
ClassName DBClassName
LastEdited DBDatetime
Created DBDatetime
SignatureID ForeignKey
WorkID ForeignKey
App_Entity_Work_Signature_PhotographSignature
ID PrimaryKey
Type inWriting / blindStamp photographersStamp companyLogo
App_Entity_Work_SeriesOfWorksRelation
ID PrimaryKey
ClassName DBClassName
LastEdited DBDatetime
Created DBDatetime
Sort Int
SeriesID ForeignKey
WorkID ForeignKey
App_Entity_Topic
ID PrimaryKey
ClassName DBClassName
LastEdited DBDatetime
Created DBDatetime
Name_de Varchar

```
$project->???
```

There is a certain price to pay, as Silverstripe has a very dynamic approach there is no static type safety, and no auto-completion out of the box.

```
$project->getDescription():string
          getLogo():Image
          getName():string
          getProperties():array
```

I sometimes miss the comfort of a Symfony/Doctrine project. But nevertheless this comes a certain cost (in terms of code required for Entities and so on).

```php
6   class Project extends \SilverStripe\ORM\DataObject {
7
8       public const DB__NAME = 'Name';
        1 usage
9       public const DB__DESCRIPTION = 'Description';
10
        2 usages
11      public const HAS_MANY__PROPERTIES = 'Properties';
12
        2 usages
13      public const HAS_ONE__LOGO = 'Logo';
14
        no usages
15      private static $has_many = [
16          self::HAS_MANY__PROPERTIES => Property::class,
17      ];
18
```

An approach that we are using as a compromise: Define some public constants and use them to access everything. Might be ugly, but it helps with refactoring.

RELAXT

```
    no usages
15  private static $has_many = [
16      self::HAS_MANY__PROPERTIES => Property::class,
17  ];
18
    no usages
19  private static $db = [
20      self::DB__NAME => 'Varchar',
21      self::DB__DESCRIPTION => 'HTMLText',
22  ];
23
    no usages
24  private static $has_one = [
25      self::HAS_ONE__LOGO => Image::class,
26  ];
27
    no usages
28  private static $owns = [
29      self::HAS_ONE__LOGO,
30  ];
31
```

Use them in the static config…

RELAXT

```php
19      private static $db = [
20          self::DB__NAME => 'Varchar',
21          self::DB__DESCRIPTION => 'HTMLText',
22      ];
23
        no usages
24      private static $has_one = [
25          self::HAS_ONE__LOGO => Image::class,
26      ];
27
        no usages
28      private static $owns = [
29          self::HAS_ONE__LOGO,
30      ];
31
        no usages
32      public function getCountOfProperties(): int{
33          return $this->{self::HAS_MANY__PROPERTIES}()->count();
34      }
35
36  }
```

… and in methods (in the same or other classes as well).

RELAXT

What else are we missing?

Identity Map

Foreign Key Constraints

Unit of Work

Proper use of Transactions

Transactions are used but sparingly throughout the core…

Versioning

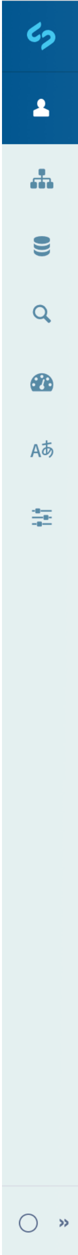
Don't get me wrong, Versioning is awesome and it is integrated into the core, but there are some problems:
* Cleanup of old versions
* Versioning of relations (there ist the experimental https://github.com/silverstripe/silverstripe-versioned-snapshots)

But those are inherently complex problems, in any framework…

Is it a Page?

Something that bothered me a lot when starting with Silverstripe. Is an Event a subclass of Page, a RealestateObject a subclass of Page, …?

My lesson learned: Use Composition, create an EventPage that references an Event, create/delete the EventPage on the fly when creating an Event. That seems to be the best of both worlds.



BACKEND

Seiten

Hinzufügen

Massenaktionen

- Gustav Klimt-Datenbank
  - Startseite DE EN
  - Klimt Werk DE EN
    - 1862–1882 (Entdeckung und…
    - 1883–1888 (Theatermaler in …
    - 1889–1894 (Ausstattungskü…
    - 1895–1897 (Symbolismus im…
    - 1898–1900 (Wiege der Mode…
    - 1901–1903 (Goldener Ritter …
    - 1904–1906 (Klimt-Affäre um …
    - 1907–1909 (Gustav Klimt am…
    - 1910–1913 (Expressive Farb…
    - 1914–1918 (Letzte Schaffen…
    - 1919–1945 (Nachruhm und …
    - 1946–2023 (Klimt heute) DE EN
  - Netzwerk Wien 1900 DE EN
  - Forschung DE
  - Die Biografie DE EN
  - Über das Projekt DE EN
  - Projektteam DE EN
  - Projektpartner:innen DE EN
  - Registrierung DE EN
  - Datenschutzerklärung DE EN
  - Nutzungsbedingungen DE EN
  - Kontakt & Impressum DE EN
  - Zugriff verweigert DE EN
  - Seite nicht gefunden DE EN
  - Seite nicht mehr verfügbar DE EN
  - Interner Serverfehler DE EN
  - Aktuelles ENTWURF DE EN
  - Aktuelles DE EN

Klimt Werk — Inhalt | Einstellungen | Verlauf

Sprache: Alle Sprachen

Haupt-Inhalt | Abhängige Seiten (1) | Feature

Feld	Sprache	Wert
Available-InLocale	de	☑
	en	☑
Seitenname	de	Klimt Werk
	en	Klimt's Artworks
URL-Segment	de	https://www.klimt-database.com/
	en	https://www.klimt-database.com/
Navigationsbezeichnung	de	
	en	
Inhalt	de	Absatz

Gespeichert | Veröffentlicht

GKDB KLIMT WERK

Im Fokus von Klimt-Werk stehen sämtliche Aspekte des Œuvres des Jugendstilmeisters. Visualisiert durch eine Timeline, werden hier Klimts Schaffensperioden aufgerollt, beginnend von seiner Ausbildung, über seine Zusammenarbeit mit Franz Matsch und seinem Bruder Ernst in der »Künstler-Compagnie«, die Affäre um die Fakultätsbilder bis hin zu seinem Nachruhm und Mythos, der diesen Ausnahmekünstler noch heute …

NUTZUNGSBEDINGUNGEN
DATENSCHUTZERKLÄRUNG

Everything

Use (*) in your query as a placeholder for parts of a word.

Sulu CMS

This is a Symfony based CMS, they have a really nice backend: https://sulu.io

Say Hello to Gutenberg, the WordPress E... ⌘K

Publish

Say Hello to Gutenberg, the WordPress Editor

Experience the flexibility that blocks allow, whether you're building yo... code for a living.

Try Gutenberg today in WordPress

This page was built with blocks — pieces of content that you can move around. ***Click around to explore them.***

Document | Block

Say Hello to Gutenberg, the WordPress Editor

380 words, 2 minutes read time.

Status Pending

Document

WordPress

Gutenberg Editor

Love it or hate it, from my point of view, inline editing fails for more complex scenarios.

Customizing

React

Webpack

Redux

Entwine

Babel

GraphQL

Schema

I am not really into React, so this is very opinionated, but a lot of developers agreed - it is hard work to extend the frontend of the Silverstripe backend (much more difficult than extending the serverside parts :-P)

Silverstripe CMS Demo

My "favorite" bug in the interface…

RELAXT

TEMPLATE

```twig
<!DOCTYPE html>
<html lang="de">
<head>

    <title>{% block title %}Welcome!{% endblock %}</title>

    {% block stylesheets %}{% endblock %}

    {% block javascripts %}{% endblock %}

</head>

<body class="{% block body_class %}{% endblock %}">

{% block body %}

    <nav>…</nav>

    <aside>
        {% block sidebar %}{% endblock %}
    </aside>

    <main>
        <h1>{% block body_header %}…{% endblock %}</h1>

        {% block body_content %}{% endblock %}
    </main>

{% endblock %}

</body>
</html>
```

twig

```
<!DOCTYPE html>
<html lang="de">
<head>

    <title>{% block title %}Welcome!{% endblock %}</title>

    {% block stylesheets %}{% endblock %}

    {% block javascripts %}{% endblock %}

</head>

<body class="{% block body_class %}{% endblock %}">

{% block body %}

    <nav>…</nav>

    <aside>
        {% block sidebar %}{% endblock %}
    </aside>

    <main>
        <h1>{% block body_header %}…{% endblock %}</h1>

        {% block body_content %}{% endblock %}
    </main>

{% endblock %}

</body>
</html>
```

```
{% extends 'base.html.twig' %}

{% block title %}Homepage{% endblock %}

{% block sidebar %}
    …
{% endblock %}

{% block body_header -%}
    Hello World
{%- endblock %}


{% block body_content %}
    …
{% endblock %}
```

I really like to have multiple blocks that can be overwritten in subtemplates, makes it very flexible :-)

Maybe I'm too optimistic, but it should be possible with Silverstripe 6.0. Fingers crossed

```
1  <!DOCTYPE html>
2  <html lang="de">
3  <head>
4
5      <title>{% block title %}Welcome!{% endblock %}</title>
6
7      {% block stylesheets %}{% endblock %}
8
9      {% block javascripts %}{% endblock %}
10
11 </head>
12
13 <body class="{% block body_class %}{% endblock %}">
```

```
9   {% block body_header -%}
10      Hello World
11  {%- endblock %}
```

```
21     </aside>
22
23     <main>
24         <h1>{% block body_hea
25
26         {% block body_content
27     </main>
28
29 {% endblock %}
30
31 </body>
32 </html>
33
```

Whitespace control allows you to fix unnecessary spaces and line breaks:

https://twig.symfony.com/doc/2.x/templates.html#whitespace-control

```
1  {% extends 'base.html.twig' %}
2
3  {% block title %}Homepage{% endblock %}
4
5  {% block sidebar %}
6      …
7  {% end
8
9  {% blo
10     He
11 {%- en
12
13
   {% block body_content %}
       …
   {% endblock %}
```

```
<h1>      Hello World
</h1>
```

```
<h1>Hello World</h1>
```

MISC

Forms

Forms are difficult to style and customize, in any framework I have tried yet. So I guess there isn't much to improve here.

Asset-Management

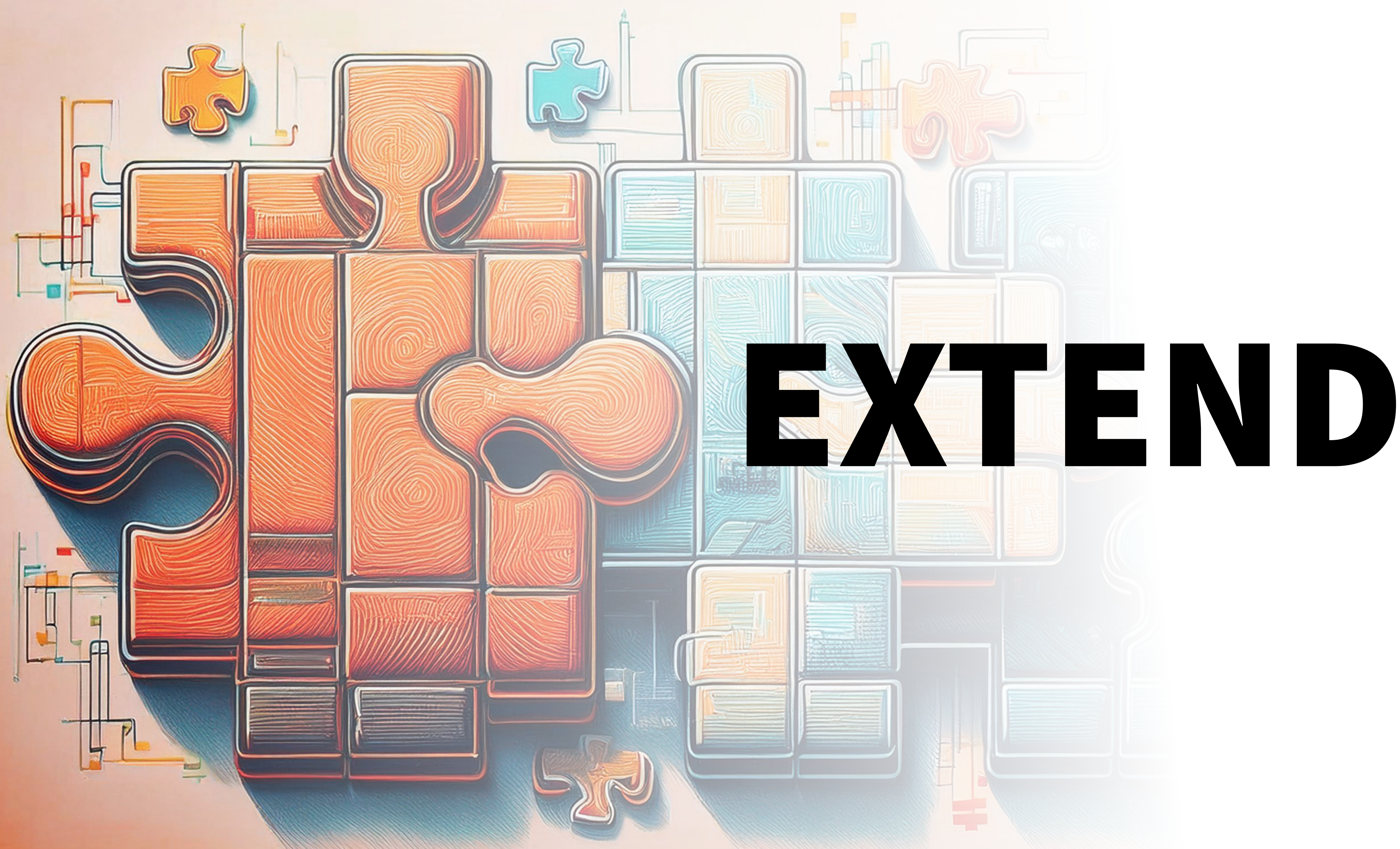
I think it could work better, especially if you have lots of assets and want to handle storage locations for your users. There are also cases where out customers don't want to expose the original files, just smaller versions - but publishing always happens for original and variants together.

Performance

From our experience this can be a major issue on more complex websites. There is a lot you can do, caching in the template, but there is a certain overhead that is difficult to tackle - the template rendering is not the fastest around, ORM and DataExtensions can cause a certain overhead and so on…

Community

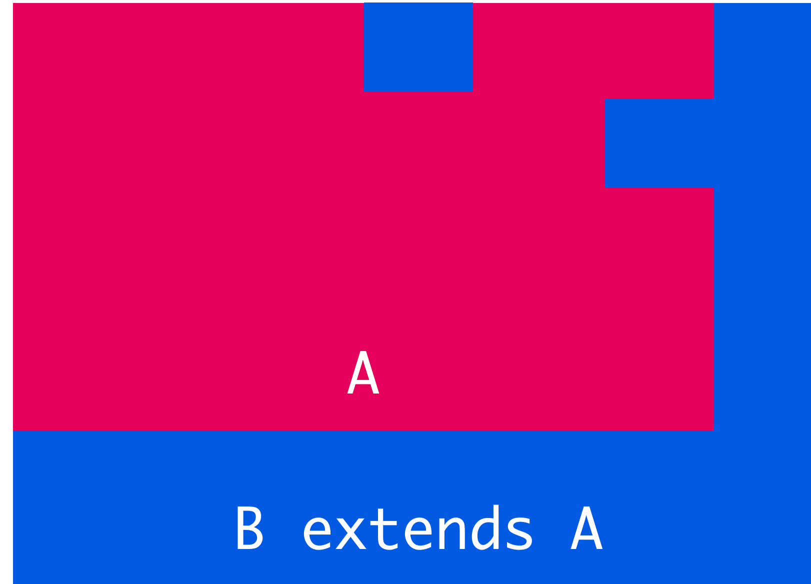
That's a good or even awesome part. I believe what really helps is, that you need to have a certain technological know-how in order to use the CMS as there is no way to do something useful without coding.


EXTEND

Okay, bear with me. I try to point out why extending Silverstripe works the way it works and why this is well thought (aka awesome).

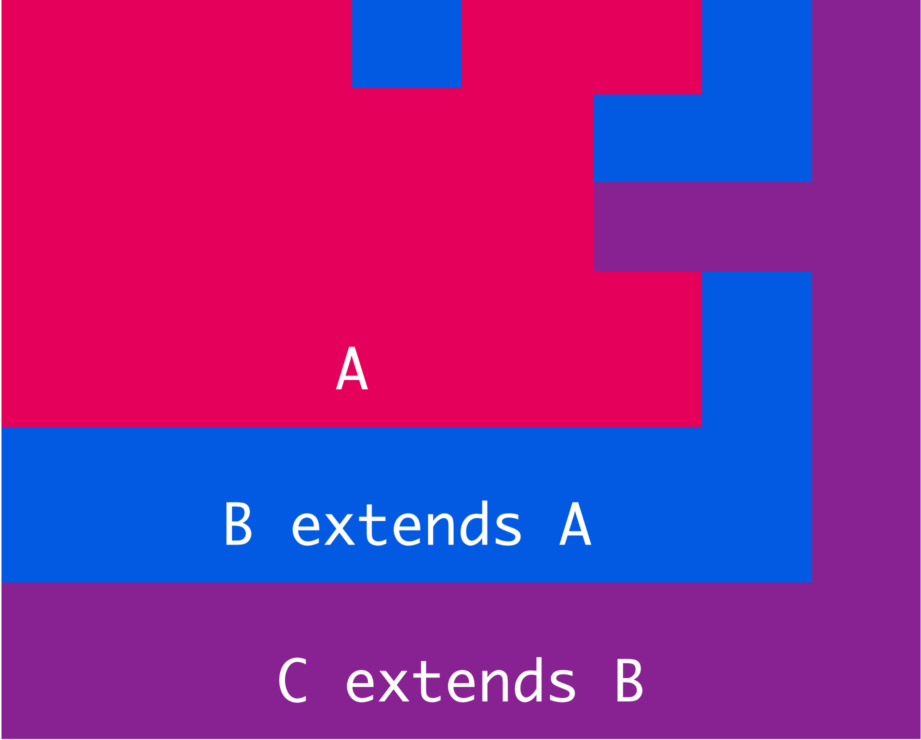
A

So this is a class A



The typical way to add new and modify existing behavior is by subtyping the class.

Inheritance

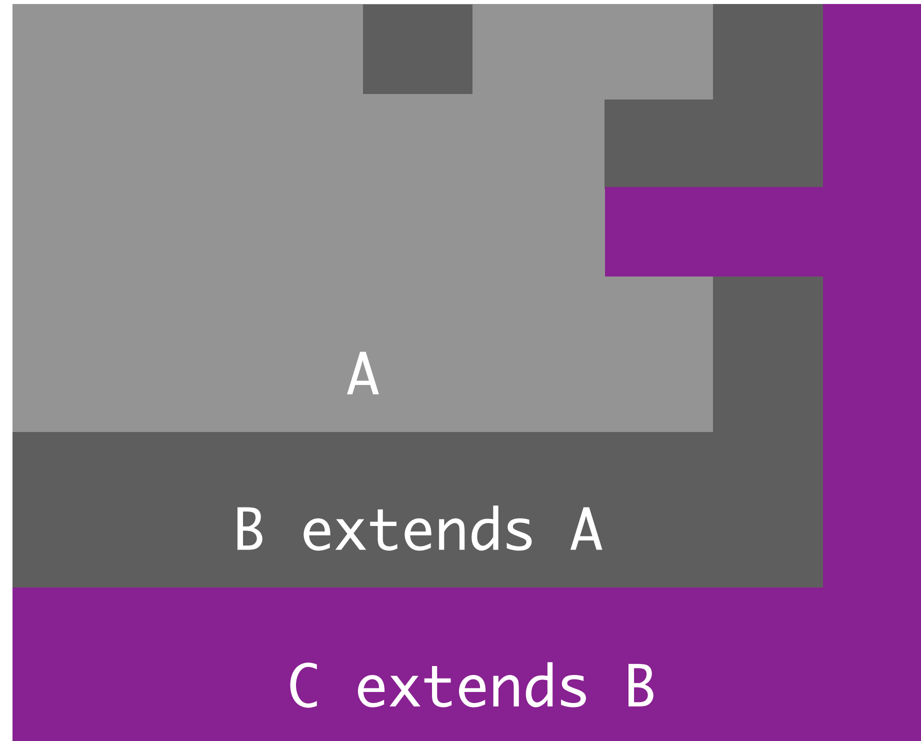


This works reasonably well, even on multiple levels.

Stripecon Europe 2024

Inheritance

RELAXT

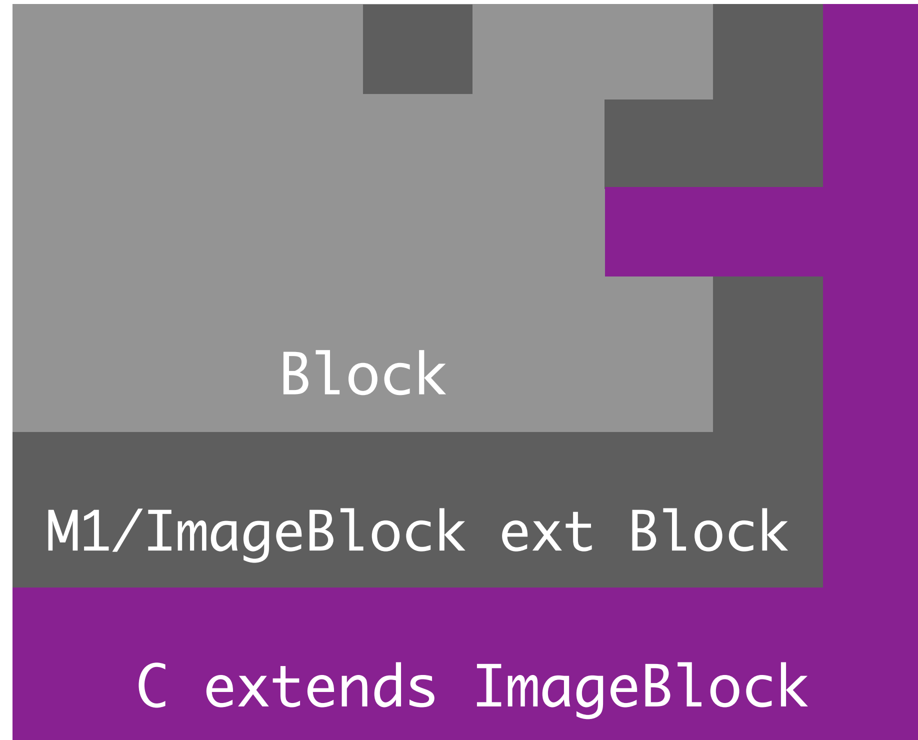


And it even works if the base classes (A, B) are not modifyable by us, but are part of a framework.

Stripecon Europe 2024

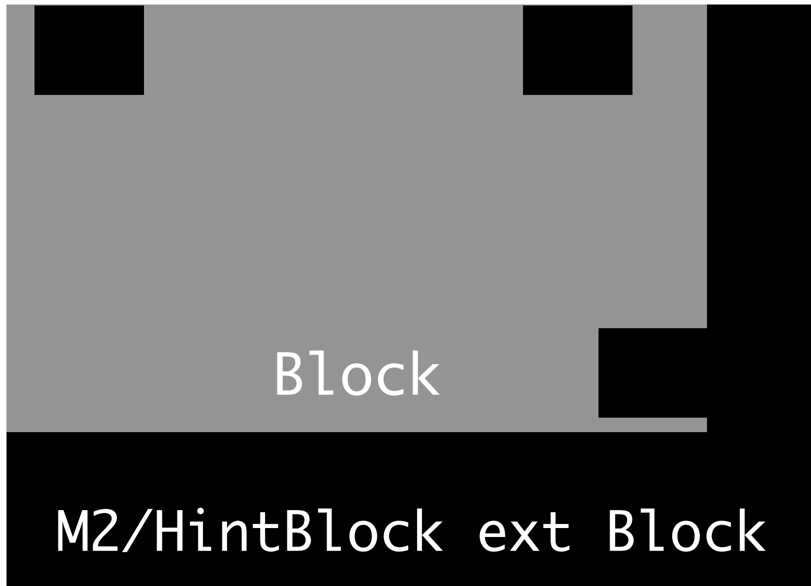
Inheritance

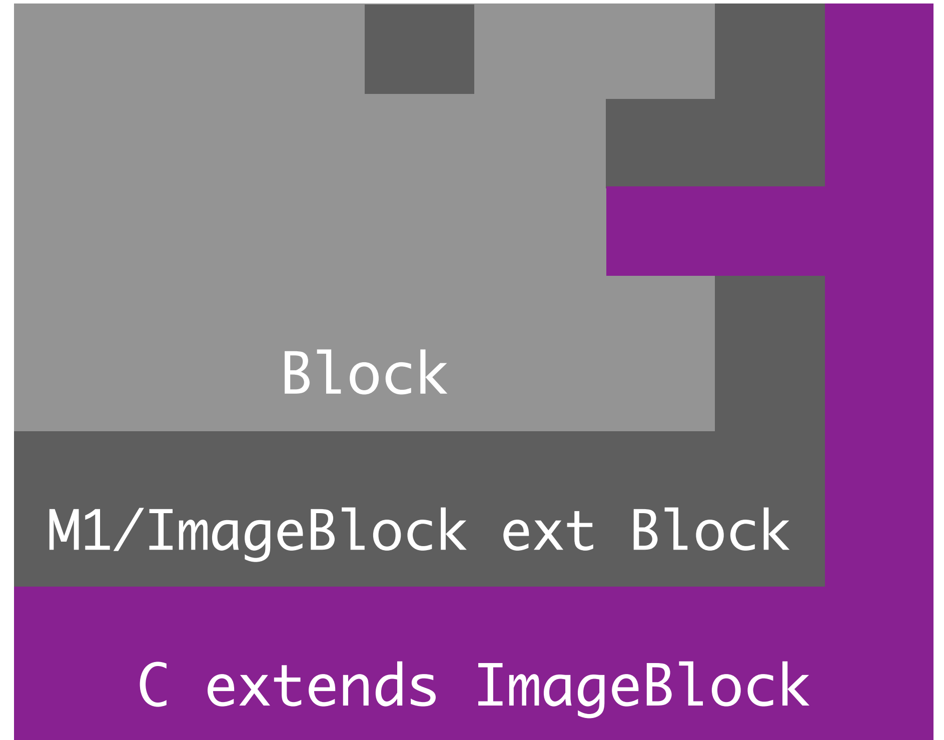
RELAXT



Let's make this example a little bit more real world. Let's assume we have some content Block (inside some core package) and a derived ImageBlock (inside some other package M1) and some custom block C in our app that wants to modify some implementation details.

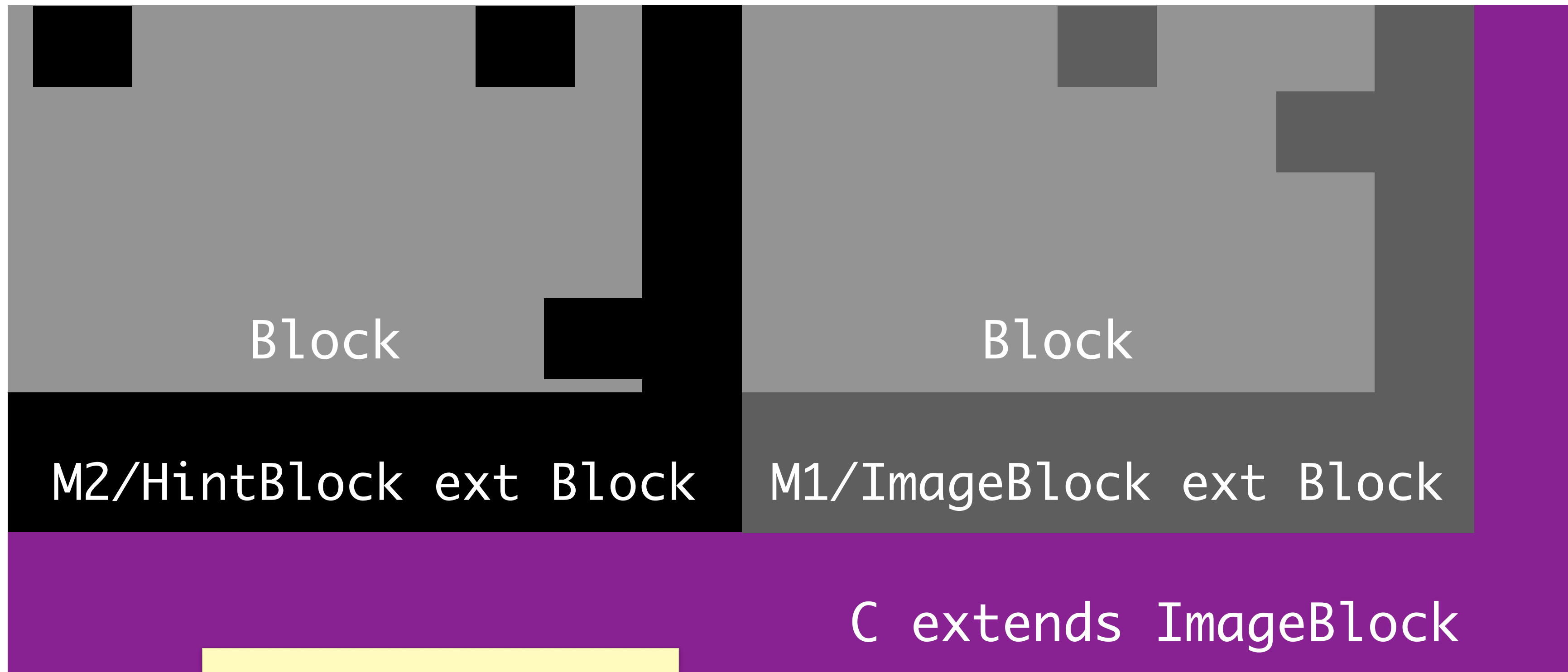
Inheritance





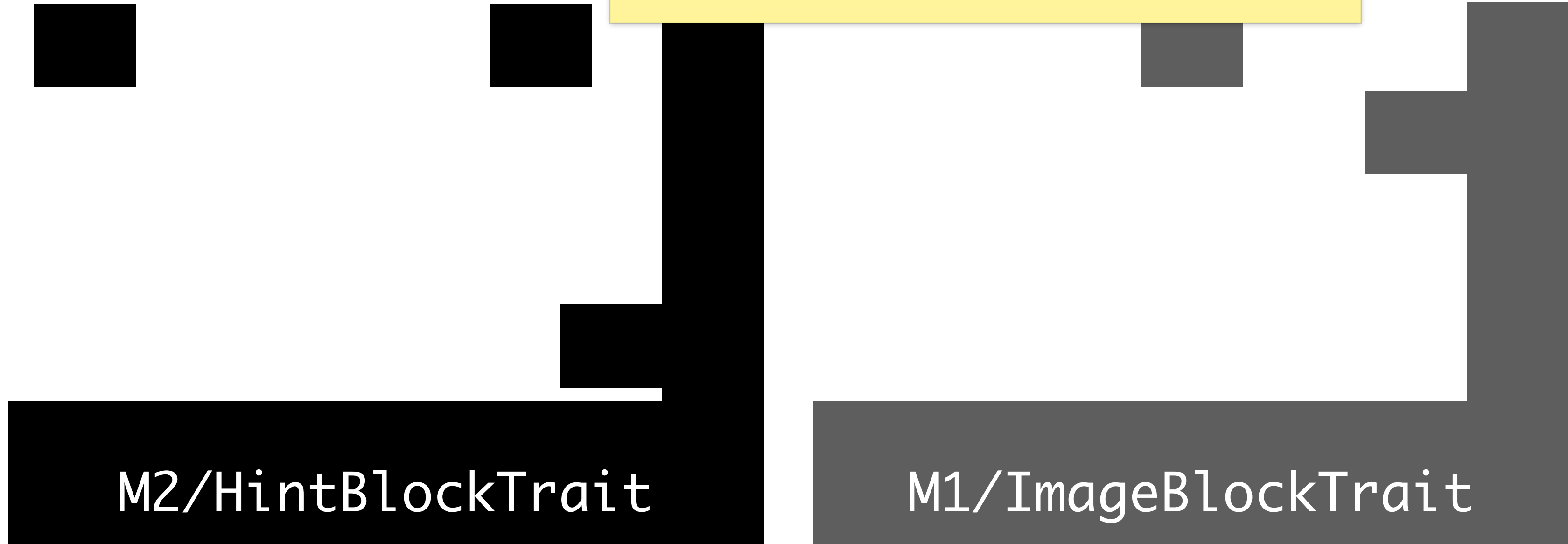
Let's assume that we have another module M2 that proves a HintBlock. And our custom block C wants to be a HintBlock and an ImageBlock…

Inheritance

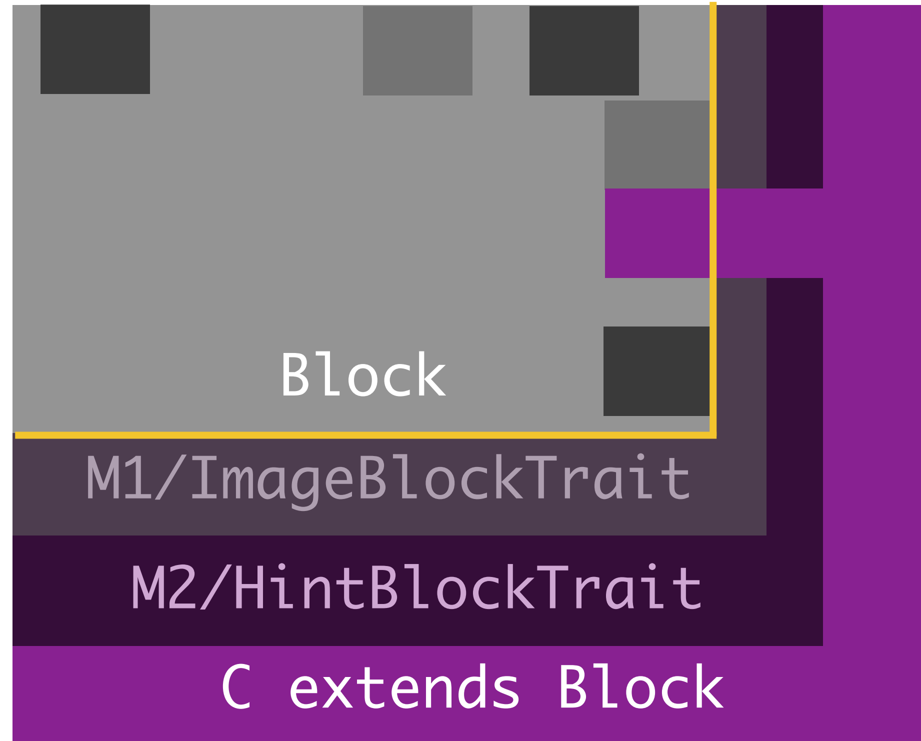


No way, there is no multiple inheritance in PHP, but…

Too much Inheritance



Mixin

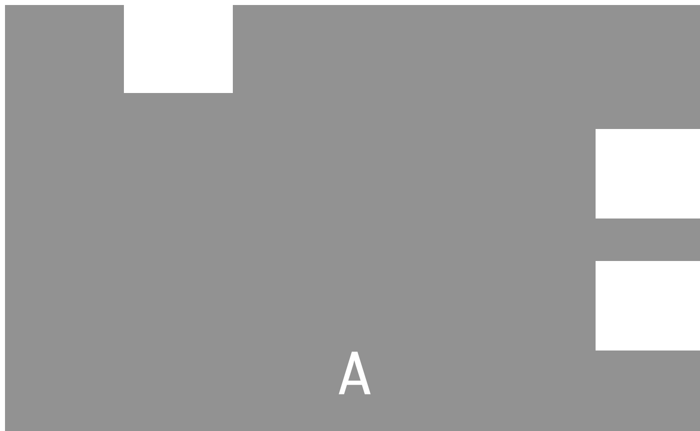


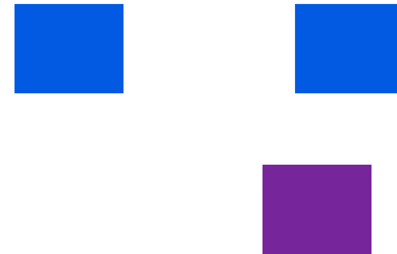
Also keep in mind that the traits are part of C, they can not be used to add behavior to the Block (as we need to modify a class in order to add a Trait)

Typically this approach is called a "mixin" allowing to flexible mix functionalities. Think "composition over inheritance"

Mixin

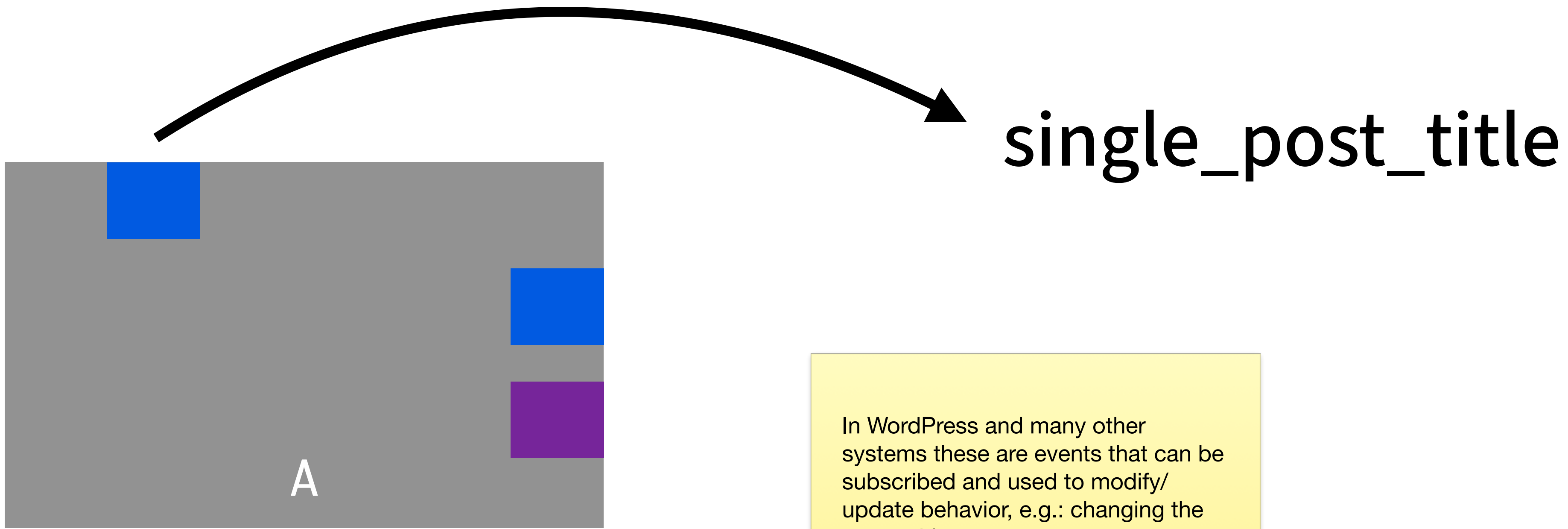
Too much Mixin



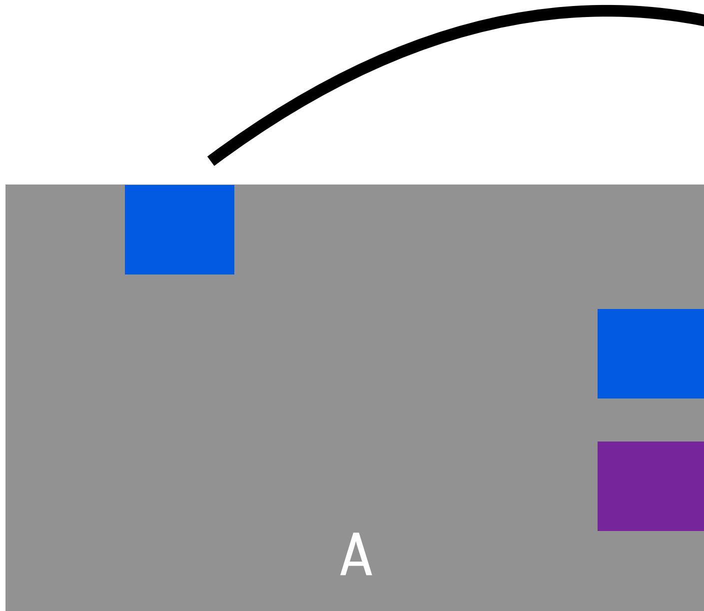


In order to add functionality to a base class, this class needs to be designed in a way that allows us to use composition/configuration to modify it's behavior without actually changing the base class. Graphically speaking, there are some holes in implementation that can be filled by custom implementations.

Composition



Composition



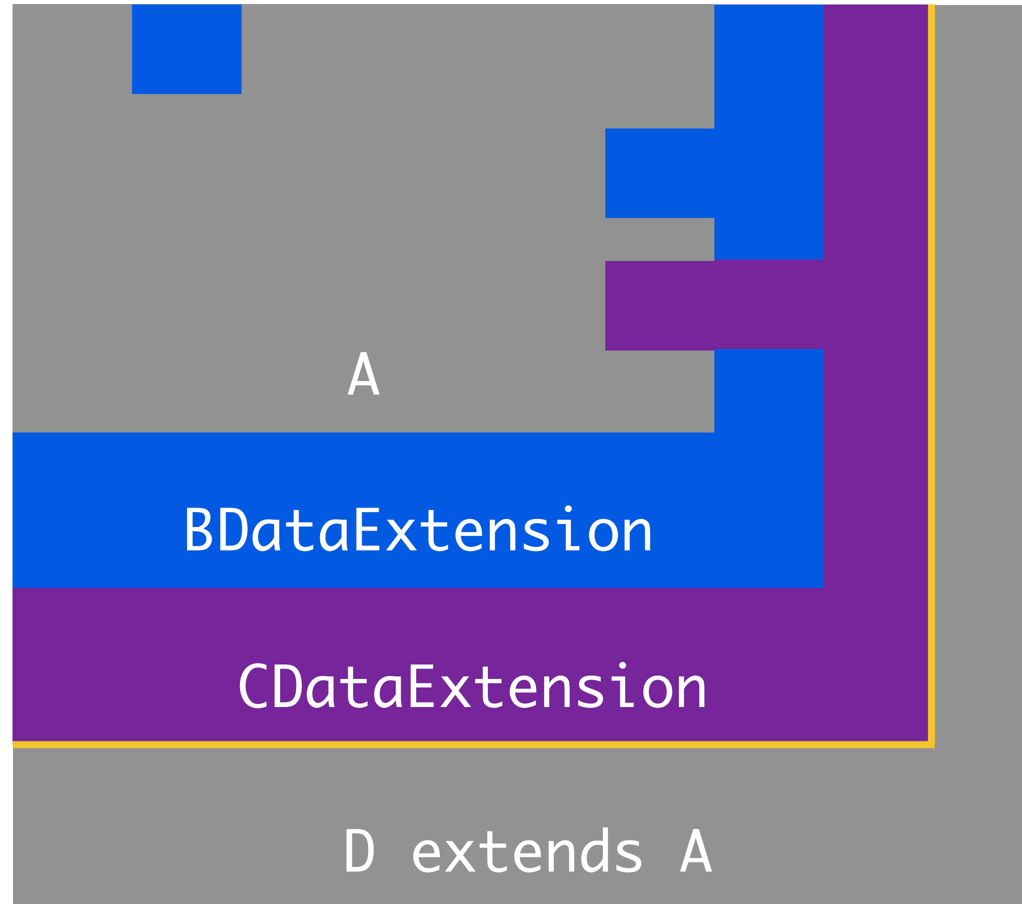
PostTitleDriverInterface#generate

DefaultTitleDriver#generate

In Symfony we could use drivers that implement a certain functionality and are configured via dependency injection.

This is utilized by Silverstripe as well, the extension points are just a variant of this composition pattern. This allows us to add behavior to a class A even if we don't control the base class (or any subclass).

Where Silverstripe really shines is, that these Extensions also allow to add public methods to the base class, combining advantages of composition and inheritance.



Stripecon Europe 2024

Composition

RELAXT

State of Silverstripe Survey

Some random idea. You might know "State of HTML" or "State of CSS" -
I think it would be beneficial to get a feeling for how people use
Silverstripe and what are the parts where improvements should be made.

```php
<?php

namespace {

    use SilverStripe\CMS\Model\SiteTree;
    use SilverStripe\ORM\HasManyList;

    /**
     * @method PageLike[]|HasManyList Likes
     */
    class Page extends SiteTree
    {
        private static array $has_many = [
            'Likes' => PageLike::class,
        ];
    }
}
```

```php
<?php

namespace {

    use App\Models\PageLike;
    use SilverStripe\Control\HTTPResponse;
    use SilverStripe\Control\HTTPRequest;
    use SilverStripe\CMS\Controllers\ContentController;

    class PageController extends ContentController
    {
        private static array $allowed_actions = [
            'likethispage',
        ];

        public function likethispage(HTTPRequest $request):
        {
            $ip = $request->getIP();

            if (!$this->Likes()->filter('IP', $ip)->exists())
                $pageLike = PageLike::create();
                $pageLike->IP = $ip;
                $this->Likes()->add($pageLike);
            }

            $this->redirectBack();
        }
    }
}
```

SILVERSTRIPE

THE GOOD THE BAD THE UGLY

and the awesome parts

Stripecon Europe 2024

RELAXT