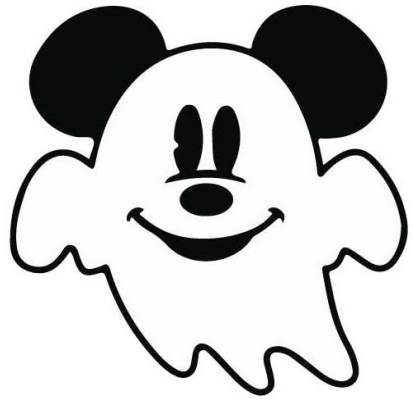
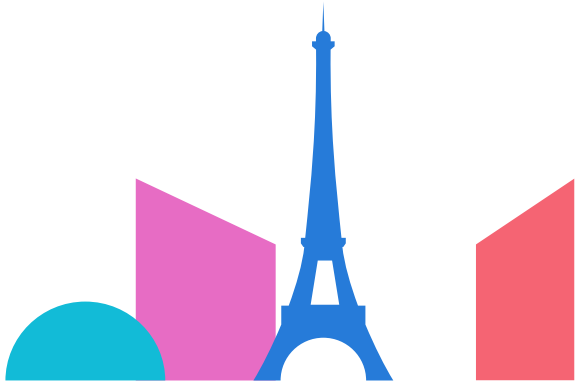
Unleashing the power of lazy objects in PHP

@nicolasgrekas



SymfonyCon
DISNEYLAND PARIS
NOV. 17-18 2022



@nicolasgrekas

- Joined in 2013 at v2.5
- SensioLabs > Blackfire.io > Symfony Corp.
- 3000+ PRs (10%)
- 4000+ commits (6%)
- 9780+ followers
- 100+ sponsors (past+present)

Don't do anything unless really needed

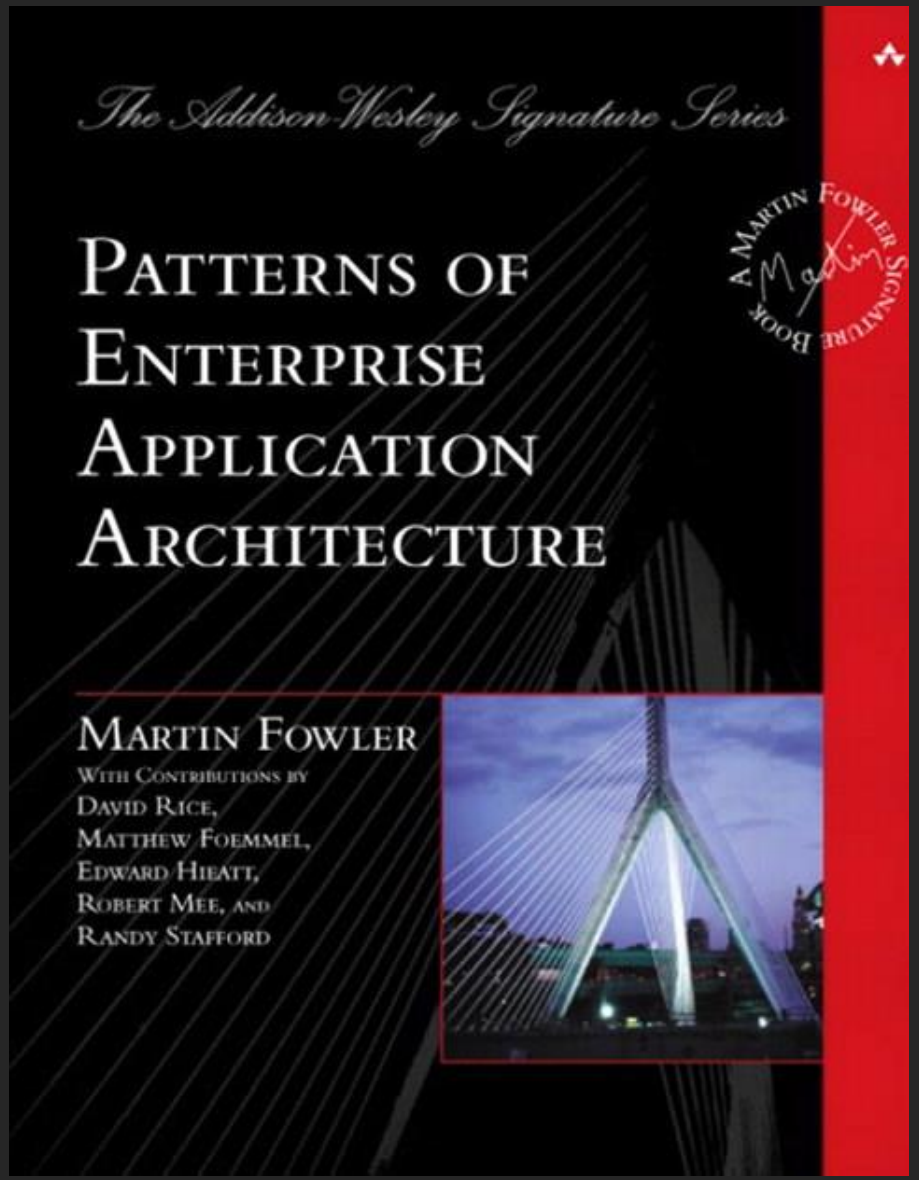
Lazy Loading

Lazy Loading

Can save time and memory

Perfect for short-lived requests

- Lazyness = autoloader
- Cache = opcache



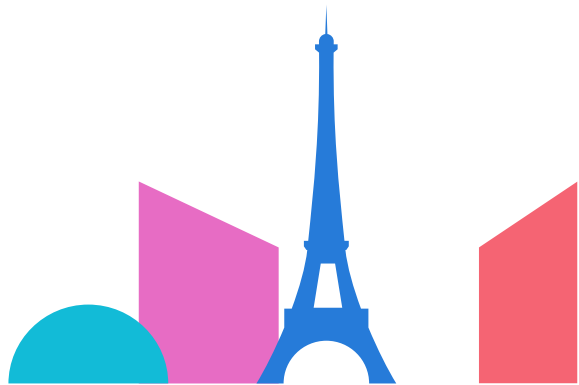
The 4 kinds of Lazy Loading

- Lazy Initialization
- Value holders
- Virtual proxies
- Ghost objects

Check properties for a marker value (usually null) and load them on demand

Lazy Initialization

```php
class LazyInitializedClass
{
    public function getData()
    {
        return $this->data ??= $this->doGetData();
    }
```

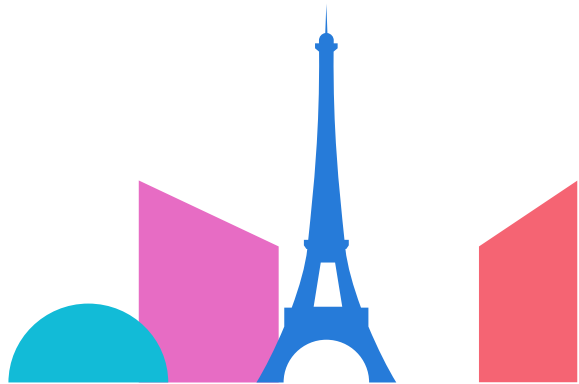
Lazy Initialization

The implementation is laziness-aware

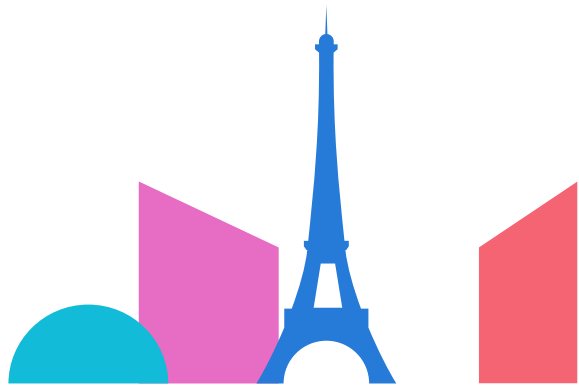
An object with a public getValue() method

Value Holders

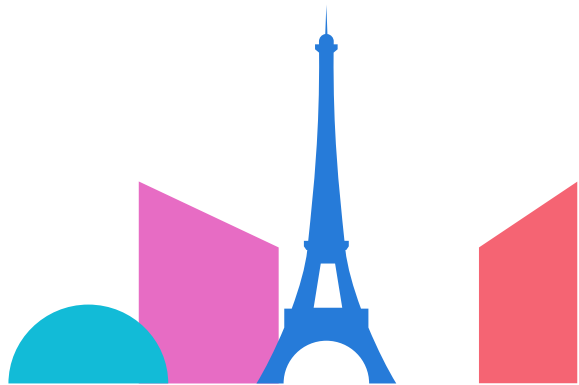
```php
class ClosureHolder
{
    public function __construct(

        private Closure|string $value
    ) {
    }

    public function getValue(): string
    {
        if ($this->value instanceof Closure) {
            $this->value = ($this->value)();
        }

        return $this->value;
    }
```

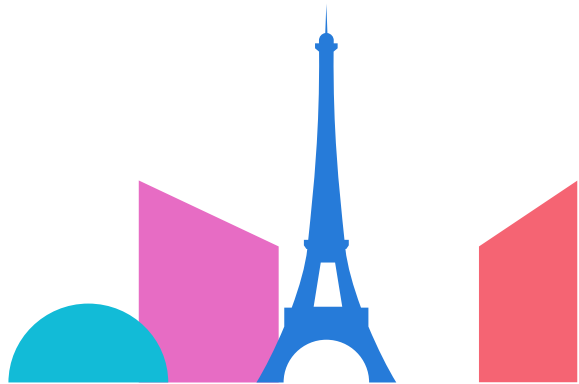
```php
class LocatorHolder
{
    public function __construct(

        private ContainerInterface $workflows
    ) {
    }

    public function getWorkflow(string $name)
    {
        return $this->workflows->get($name);
    }
```

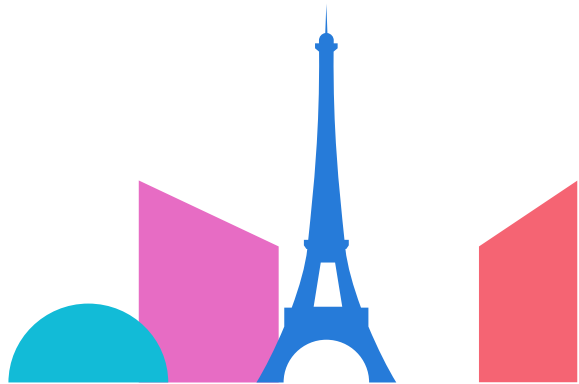
```php
class LocatorHolder
{
    public function __construct(
        #[TaggedLocator('workflow', 'name')]
        private ContainerInterface $workflows
    ) {
    }

    public function getWorkflow(string $name)
    {
        return $this->workflows->get($name);
    }
```

```php
class IterableHolder
{
    public function __construct(

        private iterable $workflows
    ) {
    }

    public function getWorkflows(): Generator
    {
        foreach ($this->workflows as $workflow) {
            yield $workflow;
        }
    }
```



```php
class IterableHolder
{
    public function __construct(
        #[TaggedIterator('workflow')]
        private iterable $workflows
    ) {
    }

    public function getWorkflows(): Generator
    {
        foreach ($this->workflows as $workflow) {
            yield $workflow;
        }
    }
```



Value Holders

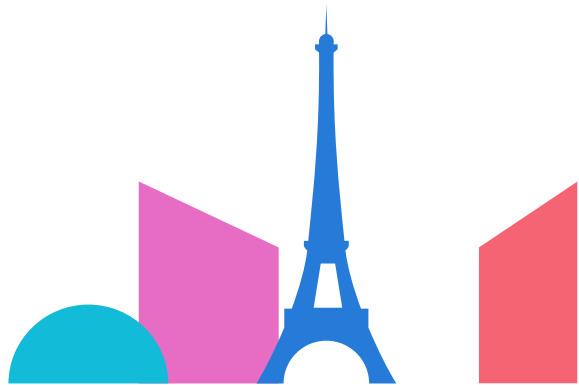
The consumers are laziness-aware

Virtual Proxies

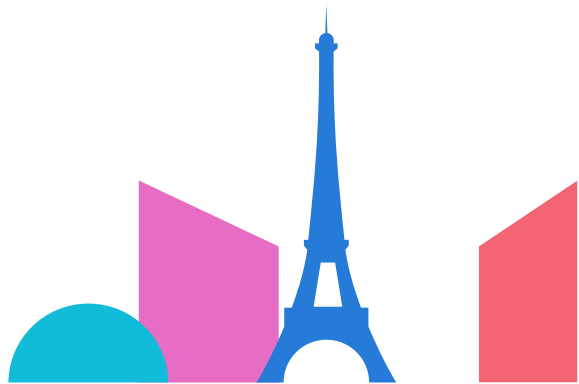
An object with the same interface as the real object

The first time any methods are called, the real object is created and called

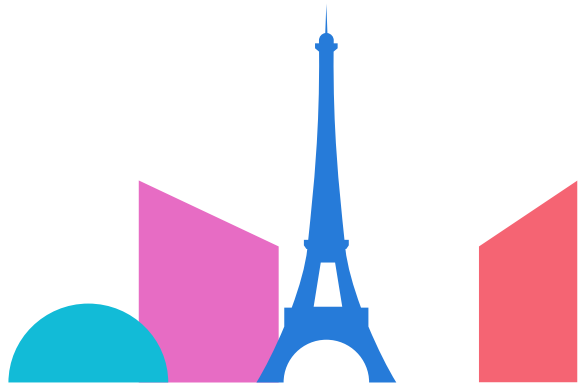
```php
class EntityManager implements EntityManagerInterface
{

    //...

    public function find(string $class, $id)
    {
        //...
    }
```

```php
class VirtualChildEntityManager extends EntityManager
{
    private parent $em;
    private bool $isInitialized = false;

    public function __construct(
        private Closure $initializer
    ) {
    }

    public function find(string $class, $id)
    {
        if (!$this->isInitialized) {
            ($this->initializer)($this);
        }

        return $this->em->find($class, $id);
    }
```

```php
class VirtualProxyEntityManager implements EntityManagerInterface
{
    private EntityManagerInterface $em;
    private bool $isInitialized = false;

    public function __construct(
        private Closure $initializer
    ) {
    }

    public function find(string $class, $id)
    {
        if (!$this->isInitialized) {
            ($this->initializer)($this);
        }

        return $this->em->find($class, $id);
    }
```

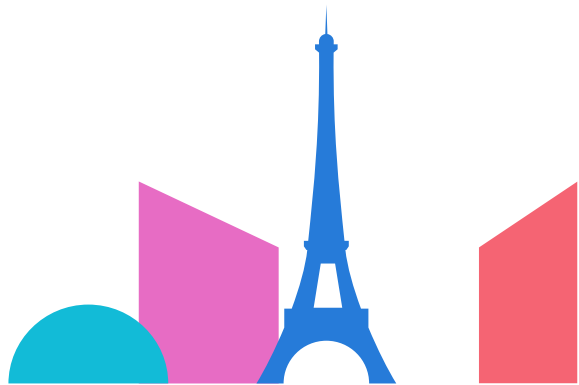
Virtual Proxies

Neither the consumers
nor the real object are laziness-aware

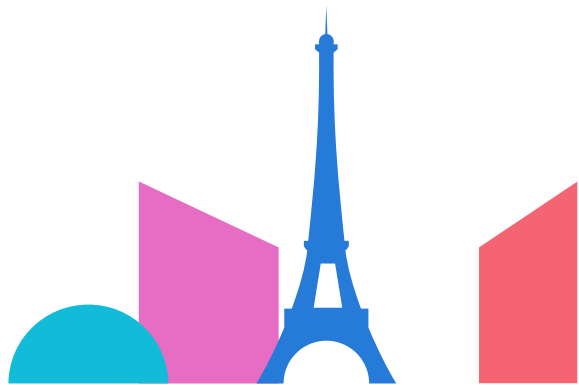
Do work with final classes

Can cause identity issues
aka break fluent/wither APIs

```php
#[Autoconfigure(lazy: true)]
class EntityManager implements EntityManagerInterface
{

    //...
```

```php
#[Autoconfigure(lazy: EntityManagerInterface::class)]
class EntityManager implements EntityManagerInterface
{

    //...
```

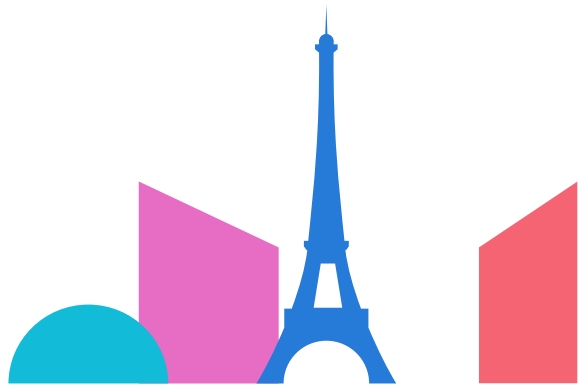



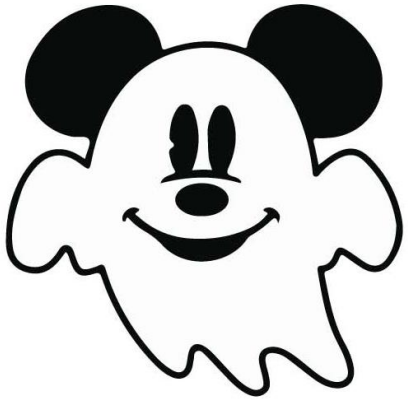
Ghost Objects

The real object without any data

The first time any methods are called, the ghost populates its properties

```php
class GhostEntityManager extends EntityManager
{
    public function __construct(
        private Closure $initializer
    ) {
        unset(/* all properties defined by the parent */);
    }

    public function __get($name)
    {
        // initialize all parent properties
    }

    // ...
```



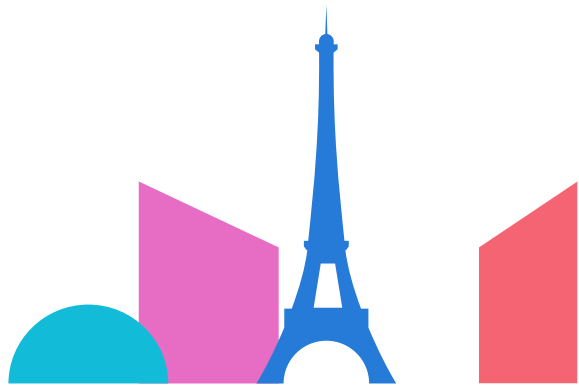
Ghost Objects

Neither the consumers
nor the real object are laziness-aware

Don't work with final classes

Don't cause identity issues
aka work with fluent/wither APIs

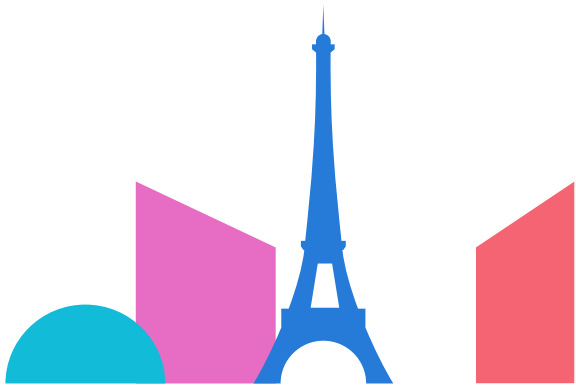
```php
namespace Proxies\__CG__\App\Entity;

use Doctrine\Persistence\Proxy;

/**
 * DO NOT EDIT THIS FILE - IT WAS CREATED BY DOCTRINE'S PROXY GENERATOR
 */
class Conference extends \App\Entity\Conference implements Proxy
{
    use \Symfony\Component\VarExporter\LazyGhostTrait
```

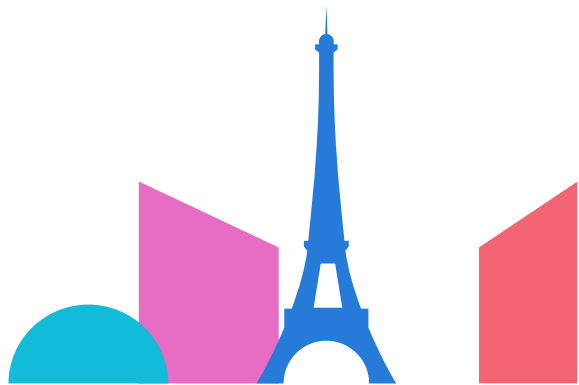
End of the month

Symfony 6.2

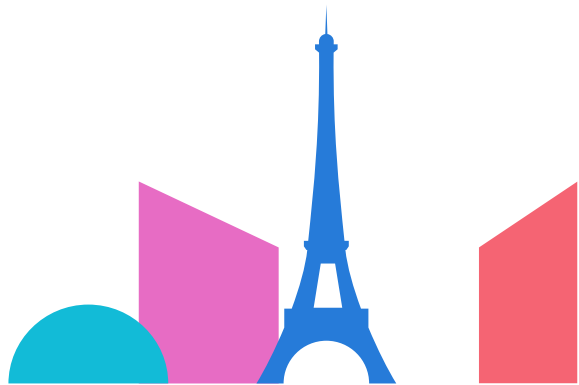
```php
class WithLazyProperty
{
    public readonly string $slowToComputeProperty;

    use LazyGhostTrait;
    private int $lazyObjectId;

    public function __construct()
    {
        self::createLazyGhost(instance: $this, initializer: [
            'slowToComputeProperty' => $this->doComputeSlowProperty(...),
        ]);
    }

    private function doComputeSlowProperty(): string
    {
        return //...
    }
```

```php
class WithLazyProperty
{
    public readonly string $slowToComputeProperty;

    use LazyGhostTrait;


    public function __construct()
    {
        $this->createLazyProperties([
            'slowToComputeProperty' => $this->doComputeSlowProperty(...),
        ]);
    }

    private function doComputeSlowProperty(): string
    {
        return //...
    }
```

```php
class Foo
{
    public function __construct(

        #[Autowire(lazy: true)]
        private BarInterface $bar,

    ) {
    }
```

TL;DR

The heavy lifting is done for you

Let me know your creative ideas

Thank you!

@nicolasgrekas

SymfonyCon
DISNEYLAND PARIS
NOV. 17-18 2022

```php
class VirtualChildEntityManager extends EntityManager
{
    private parent $em;

    public function __construct(
        private Closure $initializer
    ) {
        unset(/* all properties defined by the parent */);
    }

    public function __get($name)
    {
        $this->em ??= ($this->initializer)($this);

        return $this->em->$name;
    }
```