

Modernizing with Symfony

SymfonyCon 2022

about:me

Software Developer – Freelancer

💙 php + Symfony + Doctrine 🧡🖤

💧 Open Source 🔥

🤎 Legacy Code 🤎

💕 Father of Four 💕





Legacy Software

... works.

... fails often.

... is complicated.

... implements our
business processes.

... lacks a clear
architecture.

... is hard to change.

... earns our money.

... contains important data.

... is ugly.

... is old.

... serves a purpose.

... contains known
defects.

... is bloated.

... has been developed by people
who have left the company.

Working on Legacy Software

Problems

- No or very few automated tests.
- Bad code quality.
- Missing requirements.
- Estimating the impact of a change is hard.

Consequences

- Implementing new functionality takes longer and longer.
- High rate of defects.
- Every party involved is dissatisfied with the situation.

How confident are you
when changing your code?

? old software
===
legacy ?

Your software after 20 years?

Then:

Physical servers with
classic LAMP stack



Now

Kubernetes cluster



Your software after 20 years?

Then:

50 users

Now

5000

simultaneous
users

Your software after 20 years?

Then:

Database table with
500
records

Now

Database table with
200,000
records

Software is meant to be changed.

Technical Debt

Maintenance

- Keeping dependencies up to date (including PHP!)
- Rework tests
- Refactoring
- Work on automated checks (CI)
- Remove unused functionality

Maintenance should appear
on your invoice.



Let's do a
complete rewrite

Because what could possibly go wrong?



Micro-Rewrites

Rewrite small chunks that you can reintegrate immediately.

Stay in control. Avoid redundancy.



„A8palmbach“ (CMEW, de.wikipedia.org) CC-BY-SA

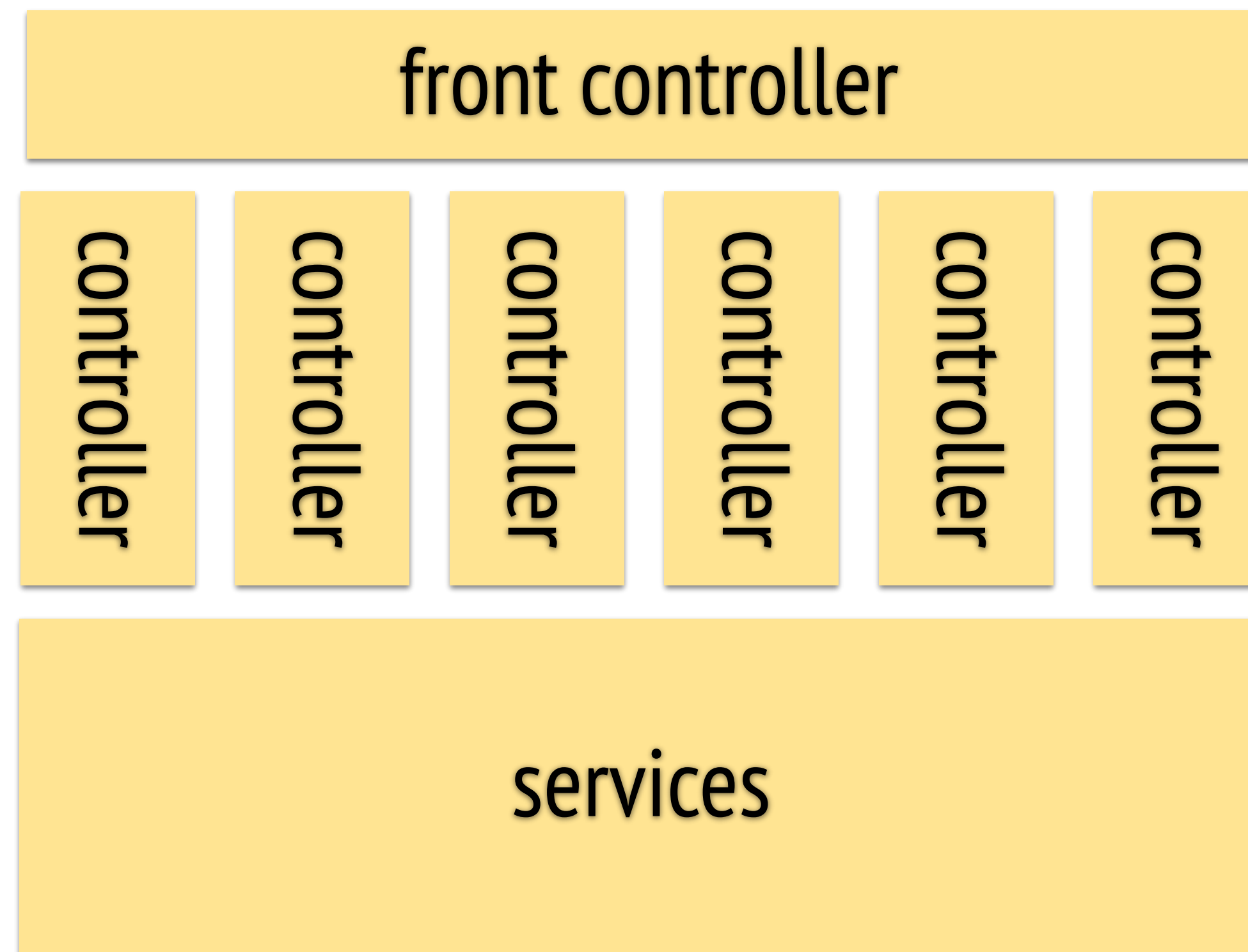
The Road to a Modernized Application

- Create a new empty application.
- Recreate a small part of the old application inside that new application.
- Let the new application take over that concern.
- Delete the corresponding code from the old app.
- Repeat.

Extension Points in Symfony Applications

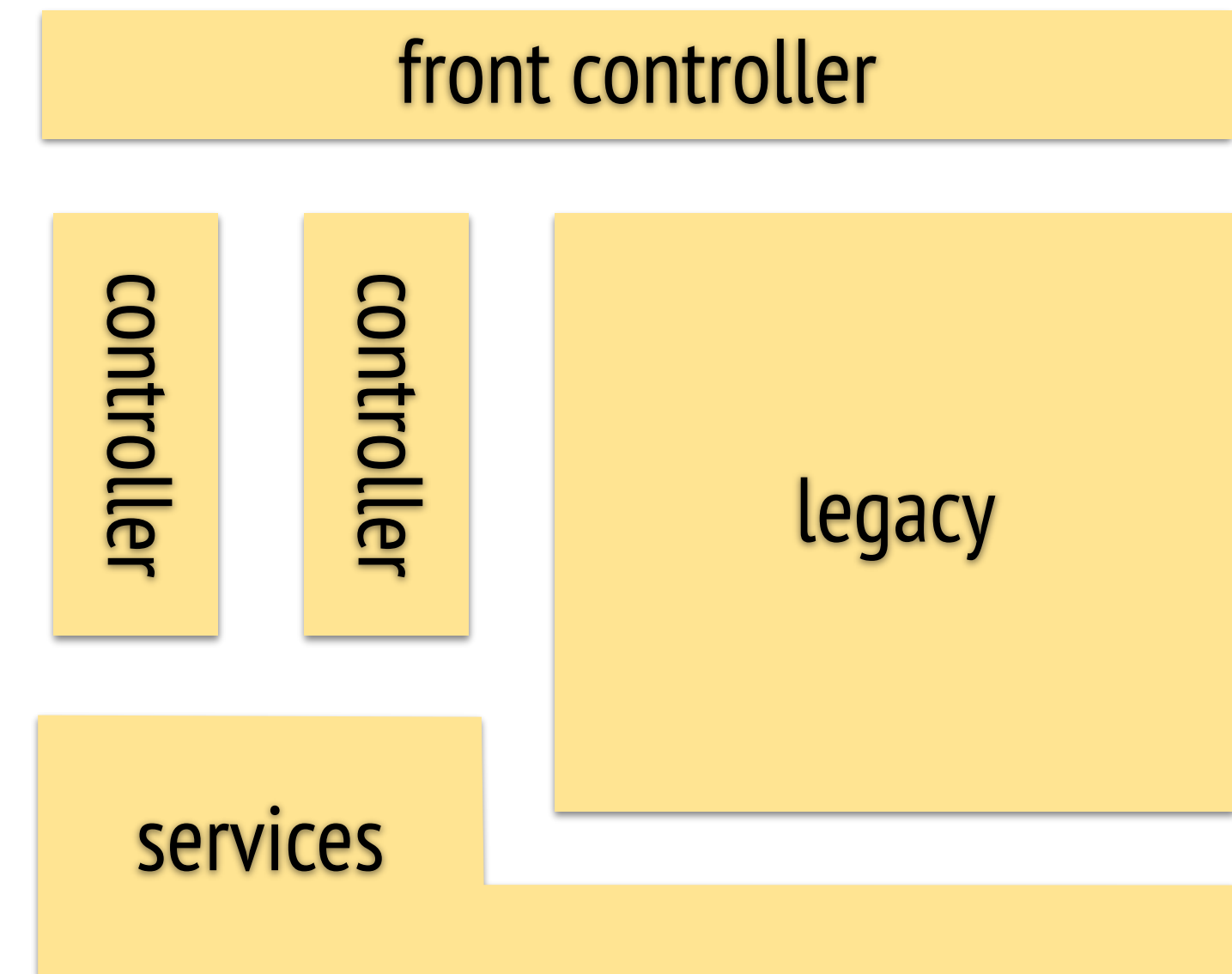
ROUTER

**SERVICE
CONTAINER**



Symfony Before Legacy

- All requests are directed to a front controller.
- A Symfony kernel is being booted.
- In case Symfony does not recognize the route, the legacy application is run.



```
$kernel = new Kernel(  
    $_SERVER['APP_ENV'],  
    (bool) $_SERVER['APP_DEBUG']  
);
```

```
$request = Request::createFromGlobals();  
$response = $kernel->handle($request);
```

```
if ($response->getStatusCode() === 404  
    || $response->getStatusCode() === 405  
) {  
    // ... Code that forwards the request to the legacy app.  
} else {  
    $response->send();  
}
```

```
$kernel->terminate($request, $response);
```



```
namespace App\Controller;

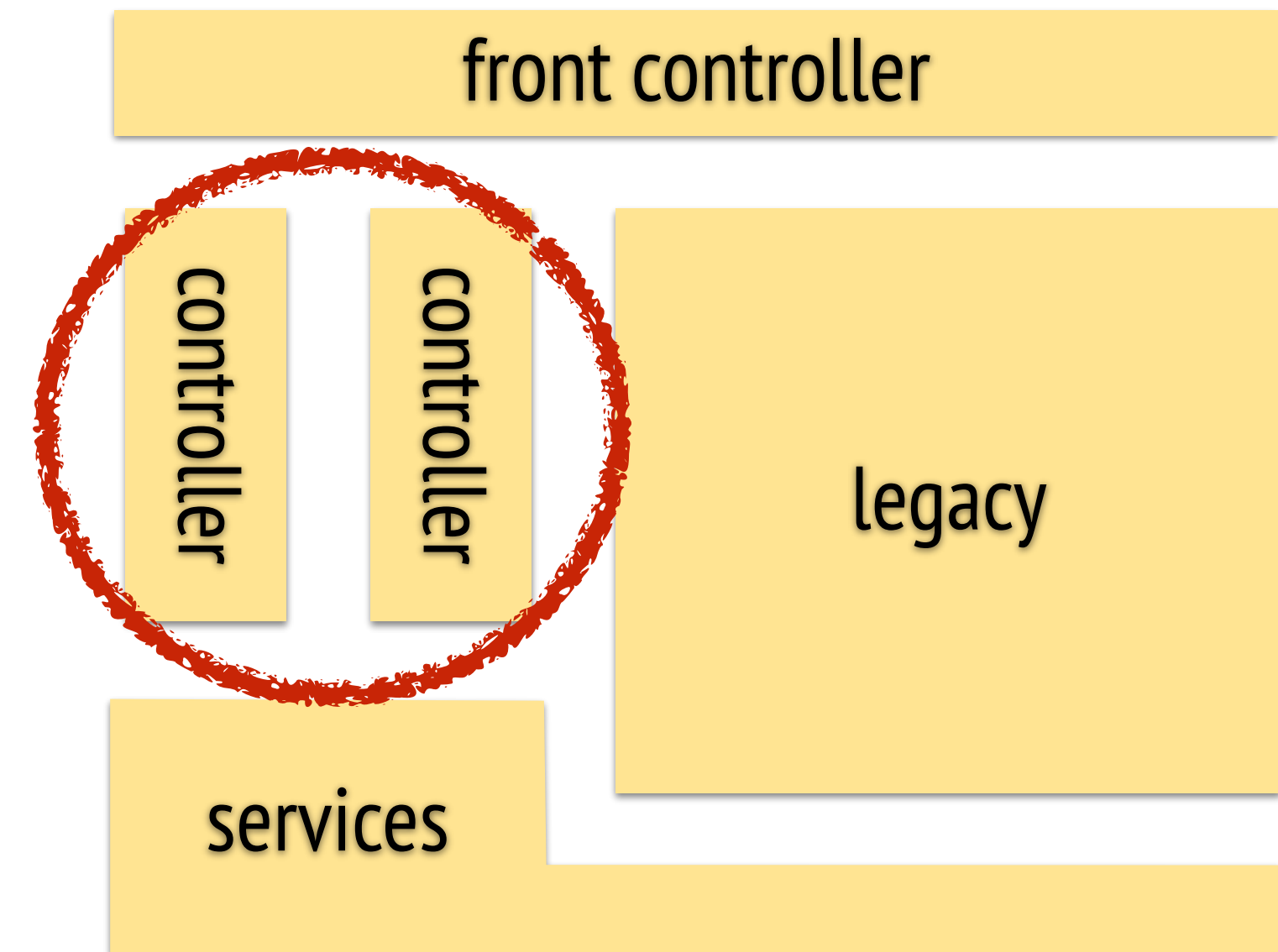
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\StreamedResponse;

#[Route('/{path}', requirements: ['path' => '.*'], priority: -500)]
class LegacyController
{
    public function __invoke(Request $request): Response
    {
        return new StreamedResponse(function () use ($request) {
            // Code that forwards the request to the legacy app.
        });
    }
}
```

Migrate Routes

Split the old application by incoming URLs.

- Pick a URL and create a Symfony controller for it.
- Delete old code.
- Repeat.



Kernel as Service Locator

```
$connection = mysql_connect($config['dbhost'], $config['dbuser'], $config['dbpassword']);  
mysql_select_db($config['database'], $connection);  
$res = mysql_query("SELECT * FROM my_table WHERE foo='$bar'", $connection);
```

The legacy application has access
to a fully booted Symfony kernel.

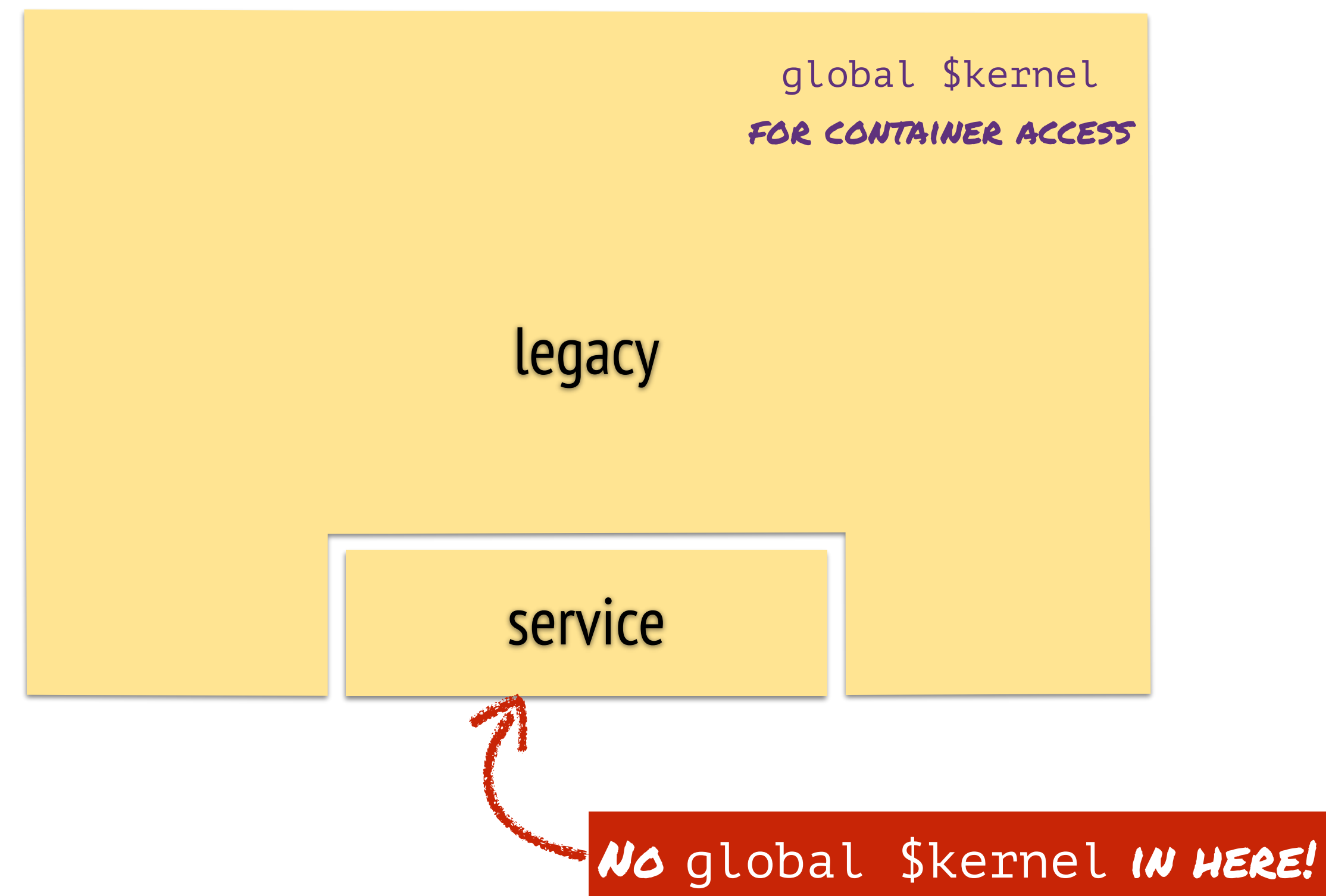


```
global $kernel;
```

```
$connection = $kernel->getContainer()->get('doctrine.dbal.default_connection');  
$res = $connection->executeQuery('SELECT * FROM my_table WHERE foo=?;', [$bar]);
```

Refactoring into Services

- Refactor a legacy component into a Symfony service.
- Change all calling code, so the new service is used.
- Delete the legacy component.
- Repeat.



Authentication

After a successful login, many legacy applications write some representation of the user into the php session.

```
if (
    isset($_POST['user'])
    && isset($_POST['password'])
    && check_login(
        $_POST['user'],
        $_POST['password']
    )
) {
    $_SESSION['username']
        = $_POST['user'];
}
```

Detect Authenticated Users

```
final class LegacySessionAuthenticator extends AbstractAuthenticator
{
    public function supports(Request $request): ?bool
    {
        return isset($_SESSION['username']);
    }

    public function authenticate(Request $request): Passport
    {
        return new SelfValidatingPassport(
            new UserBadge($_SESSION['username'])
        );
    }
}
```

Detect Authenticated Users

```
security:
  firewalls:
    dev:
      pattern: ^/(_(profiler|wdt)|css|images|js)/
      security: false

  main:
    lazy: true
    provider: main
    stateless: true

  custom_authenticators:
    - App\Security\LegacySessionAuthenticator
```


Migrate to Symfony Security

```
if (!empty($_SESSION['is_admin'])) {  
    // Display Admin stuff  
}
```



```
$authorizationChecker = $kernel->getContainer()  
    ->get('security.authorization_checker');  
if ($authorizationChecker->isGranted('ROLE_ADMIN')) {  
    // Display Admin stuff  
}
```

Modernizing with Symfony

- Symfony as the framework for new and modernized application parts.
- Refactoring in small steps.
- Refactoring before changes.
- Avoid blocking refactorings.
- No never-ending rewrite.

Thank you

spam me:

me@derrabus.de

follow me:

[@derrabus](#)

hire me:

<https://derrabus.de>