The Test Suite
Holy Trinity

Dave Liddament

@DaveLiddament

First a sad story….

…. about a dark time

@DaveLiddament

I still have nightmares

Why this talk?

Quick introduction

@DaveLiddament

Back to the nightmare…

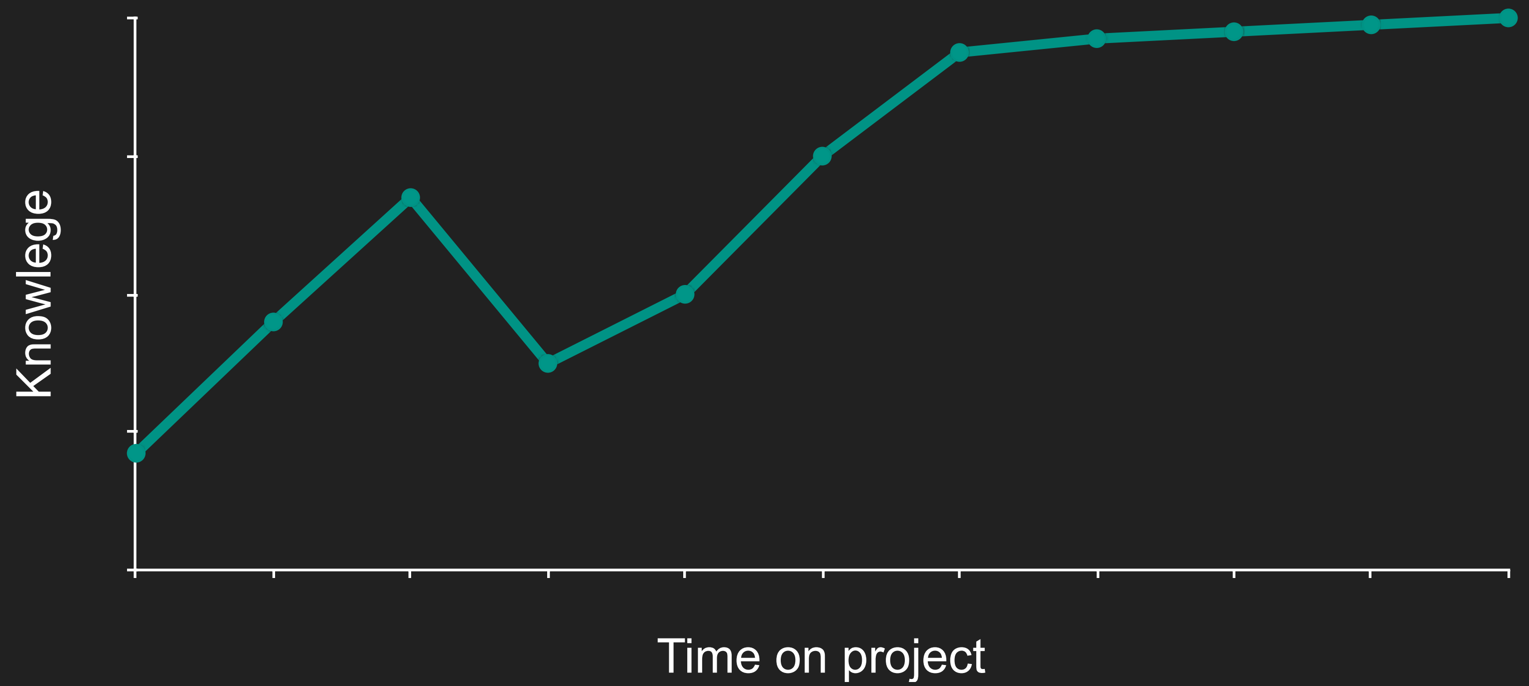
#1 I didn’t (still don’t) know much about developing high quality software

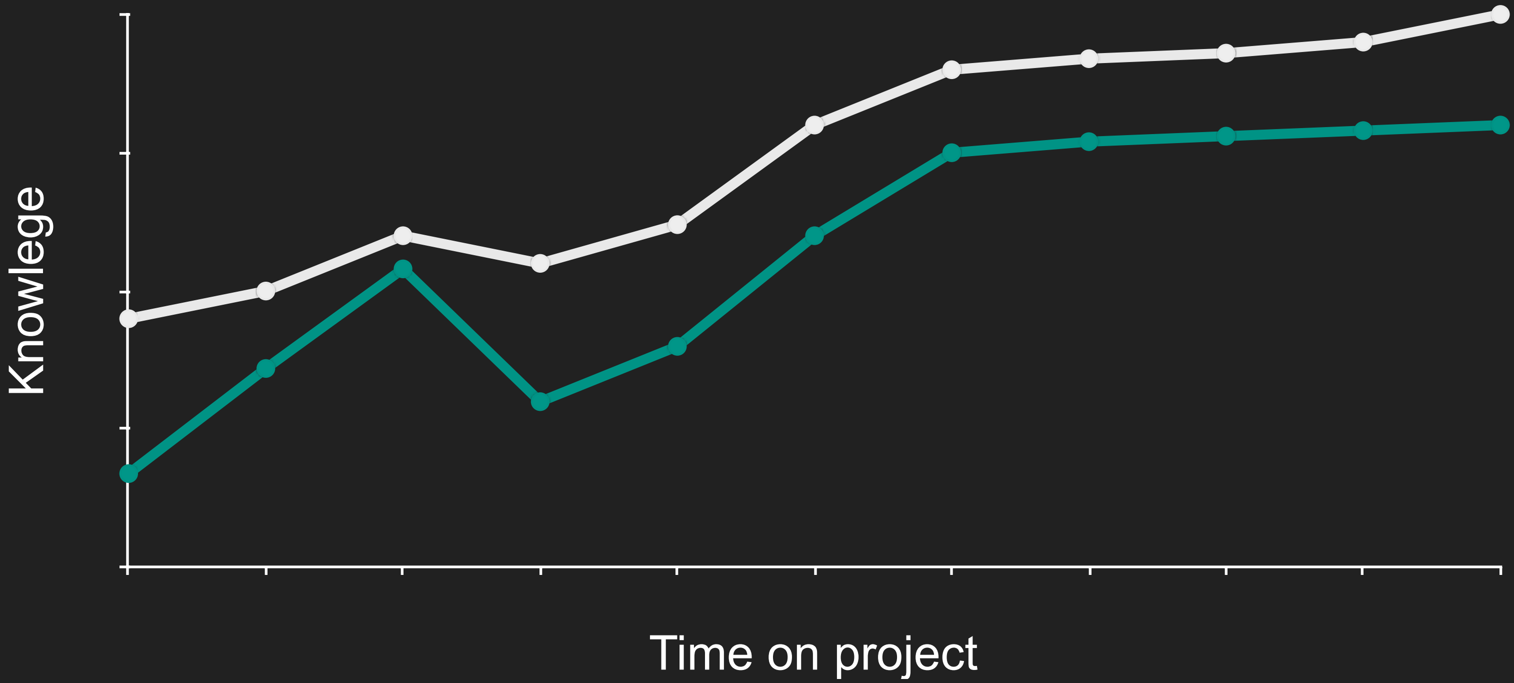
#2 Copy someone who does know about developing high quality software

We need tests

We need a test suite

Ability to refactor is important


Knowledge
Time on project


Knowledge
Time on project

A quick recap…

A test suite…

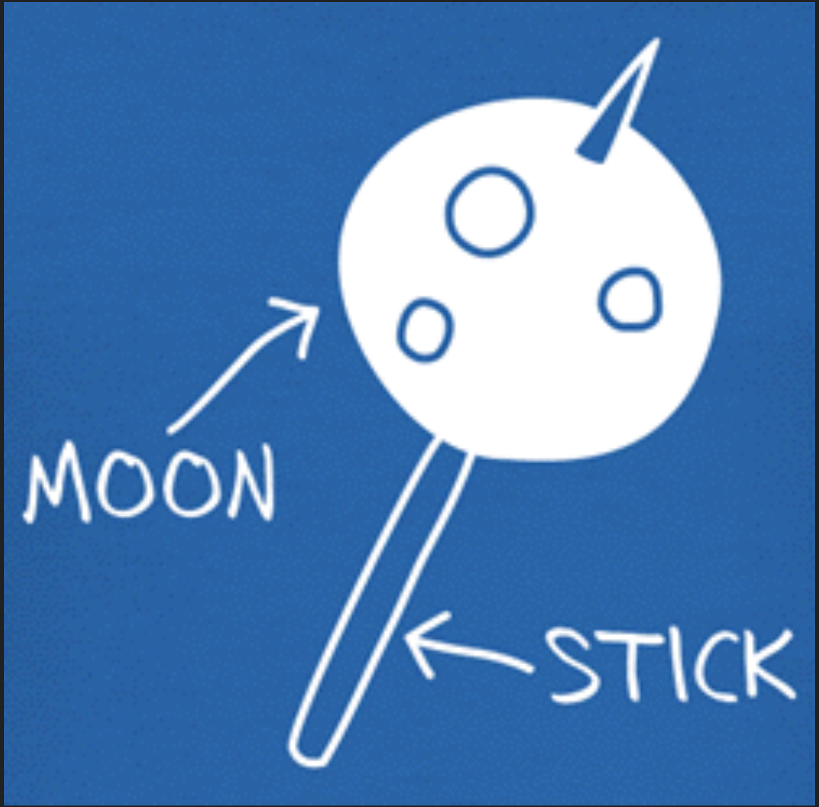
#1 Proves code works
#2 Stops regression
#3 Enables refactoring

The ideal test suite…

Fast to execute

High coverage

Low maintenance


MOON
STICK

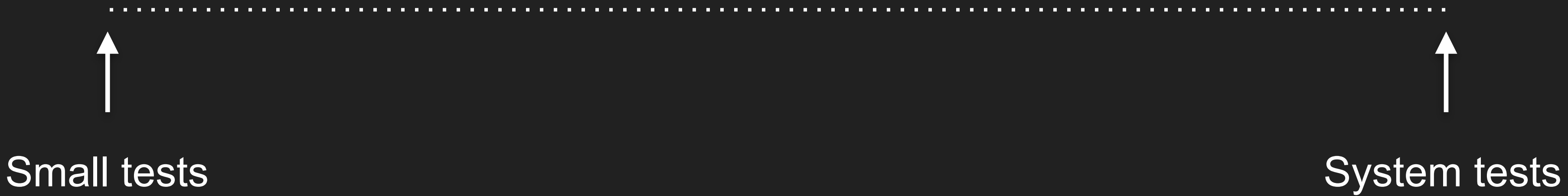
The Holy Trinity…

#1 Fast to execute

#2 High coverage

#3 Low maintenance

Testing Continuum

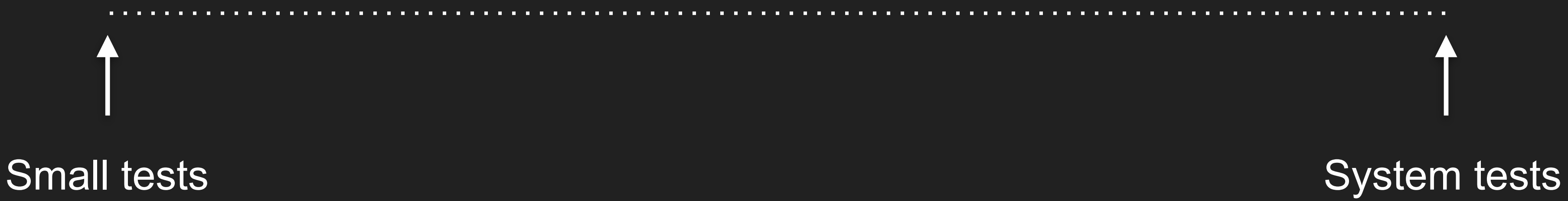


Small test example

```
class PasswordValidator
{

  /**
   * Returns true if password meets following criteria:
   *
   * - 8 or more characters
   * - at least 1 digit
   * - at least 1 upper case letter
   * - at least 1 lower case letter
   */
   public function isValid(string $password) : bool
```

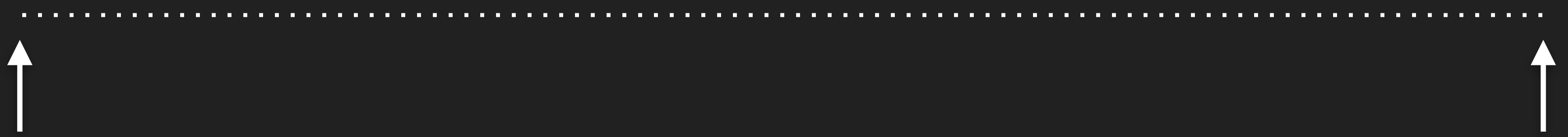
Testing continuum
#1 Fast to execute

Testing Continuum: Automation



Testing Continuum: Automation

All

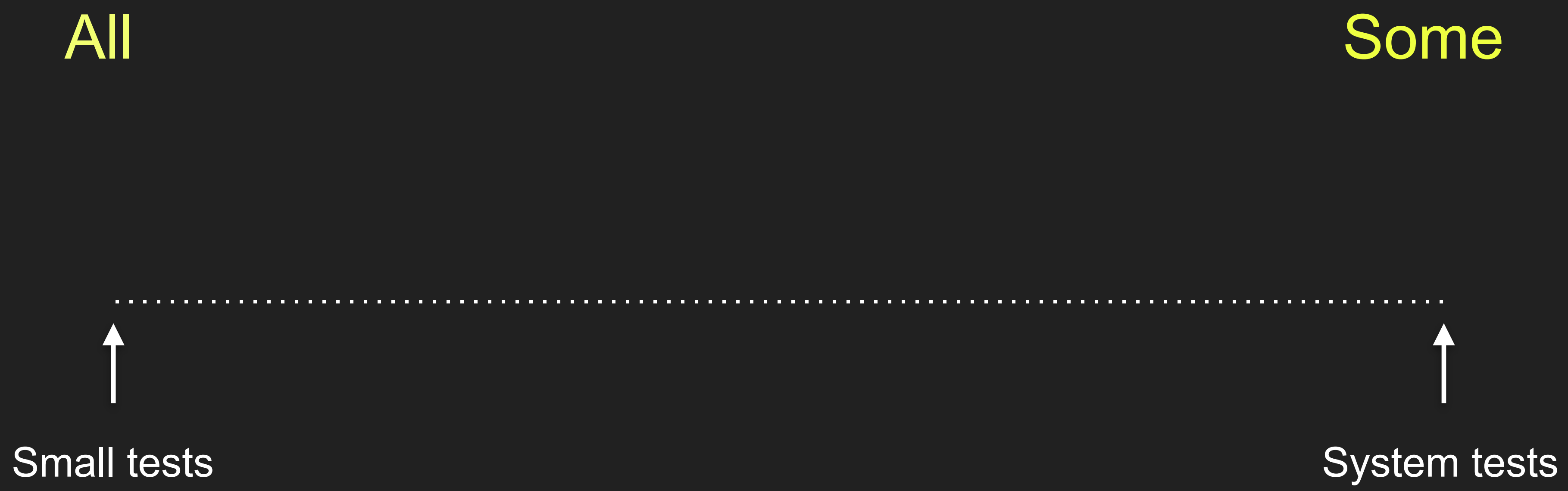


Small tests

System tests

@DaveLiddament

Testing Continuum: Automation



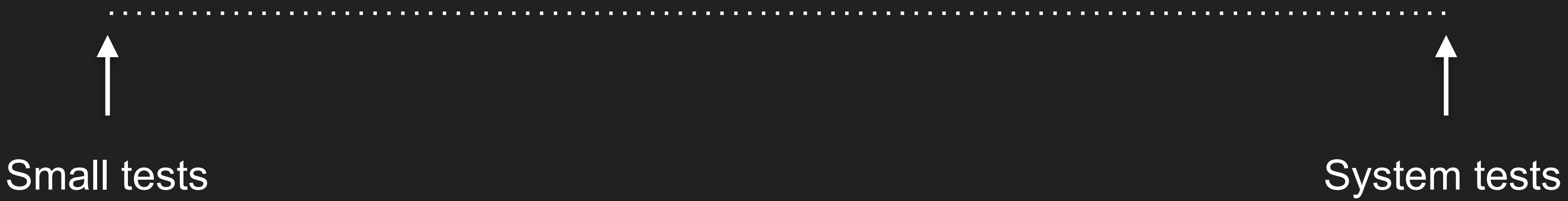
@DaveLiddament

Testing Continuum: Speed of execution

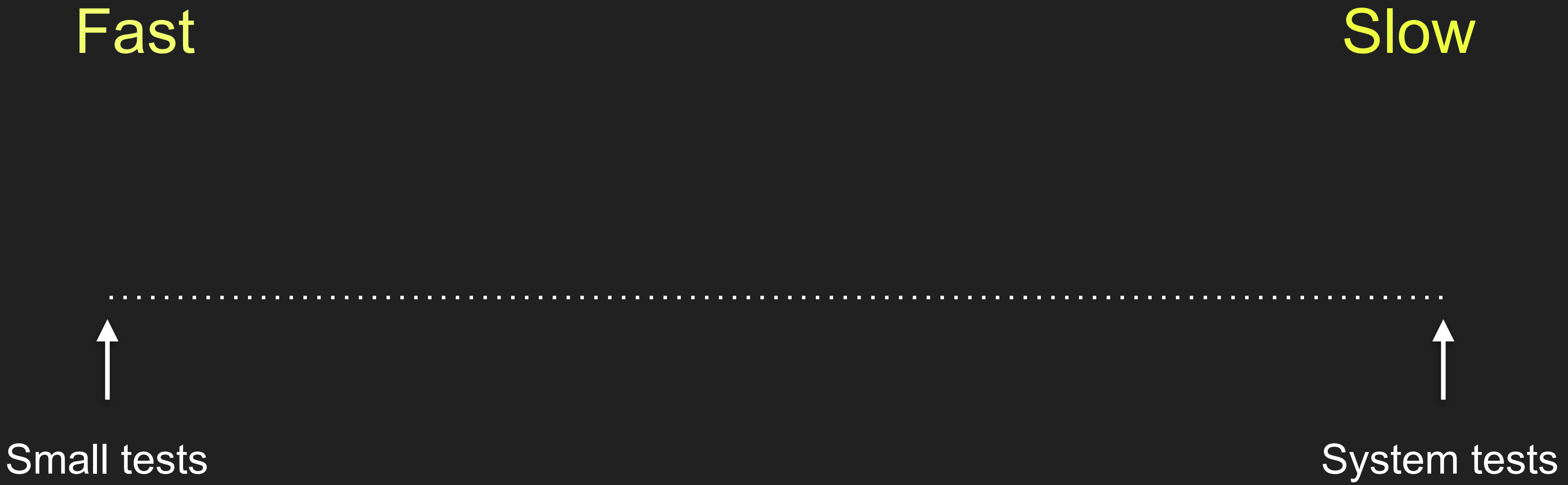
Small tests
System tests

Testing Continuum: Speed of execution

Fast

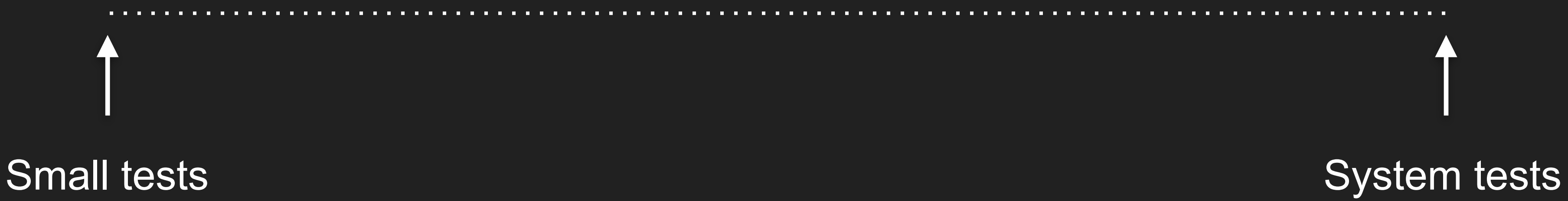


Testing Continuum: Speed of execution

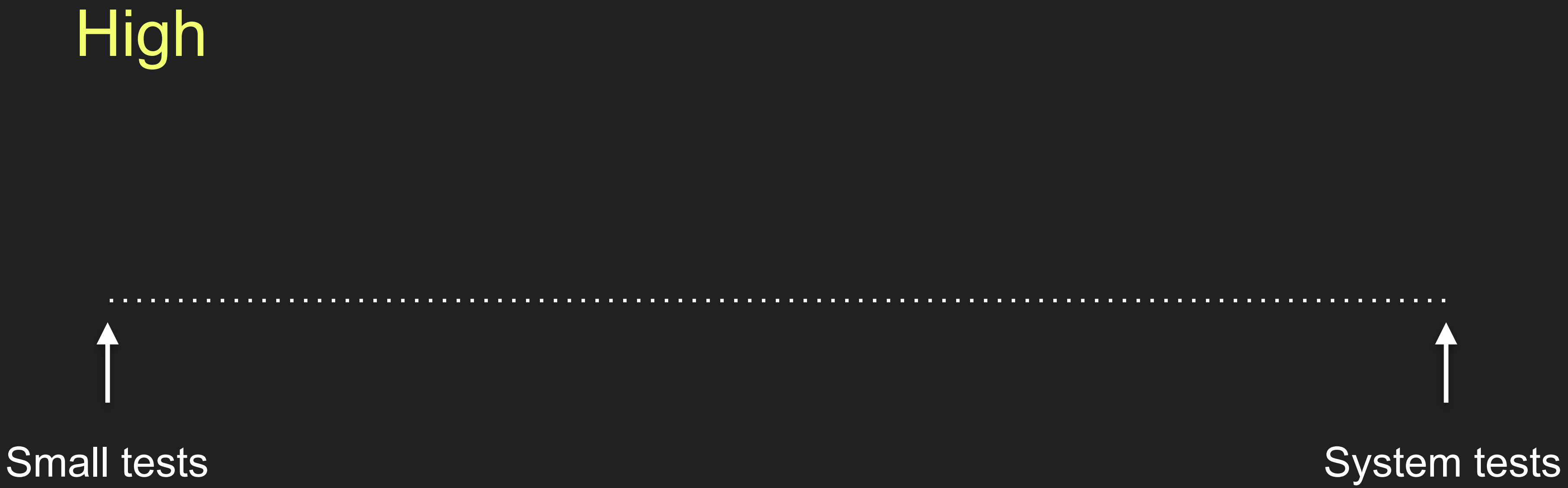


Testing continuum
#2 High coverage

Testing Continuum: Coverage

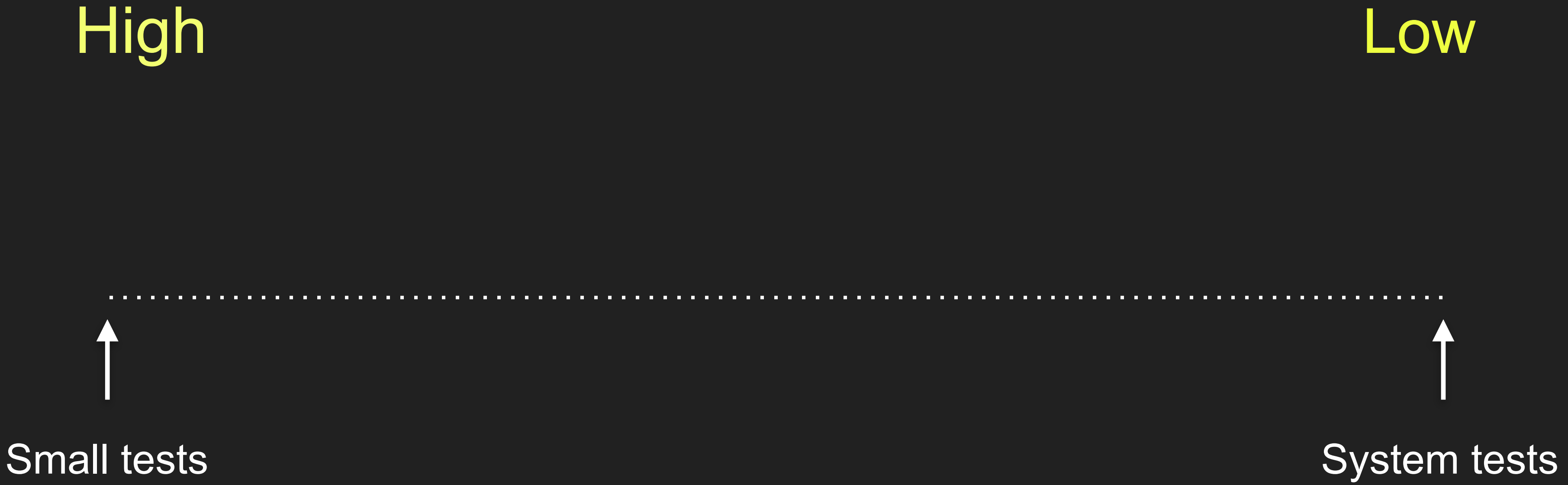


Testing Continuum: Coverage



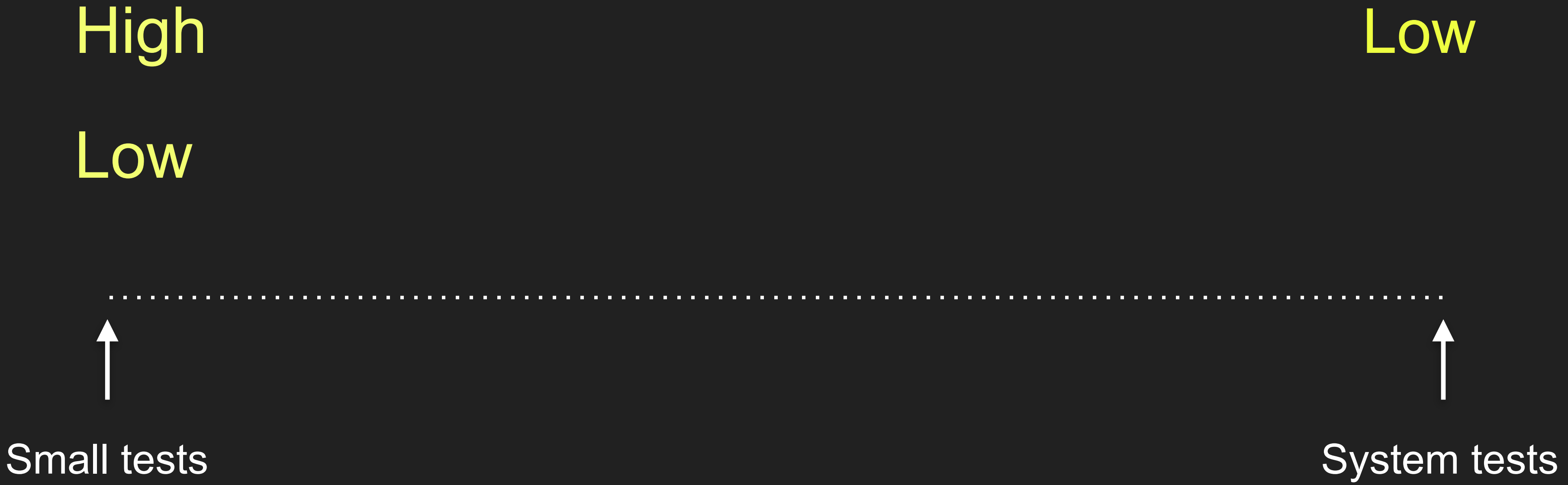
@DaveLiddament

Testing Continuum: Coverage

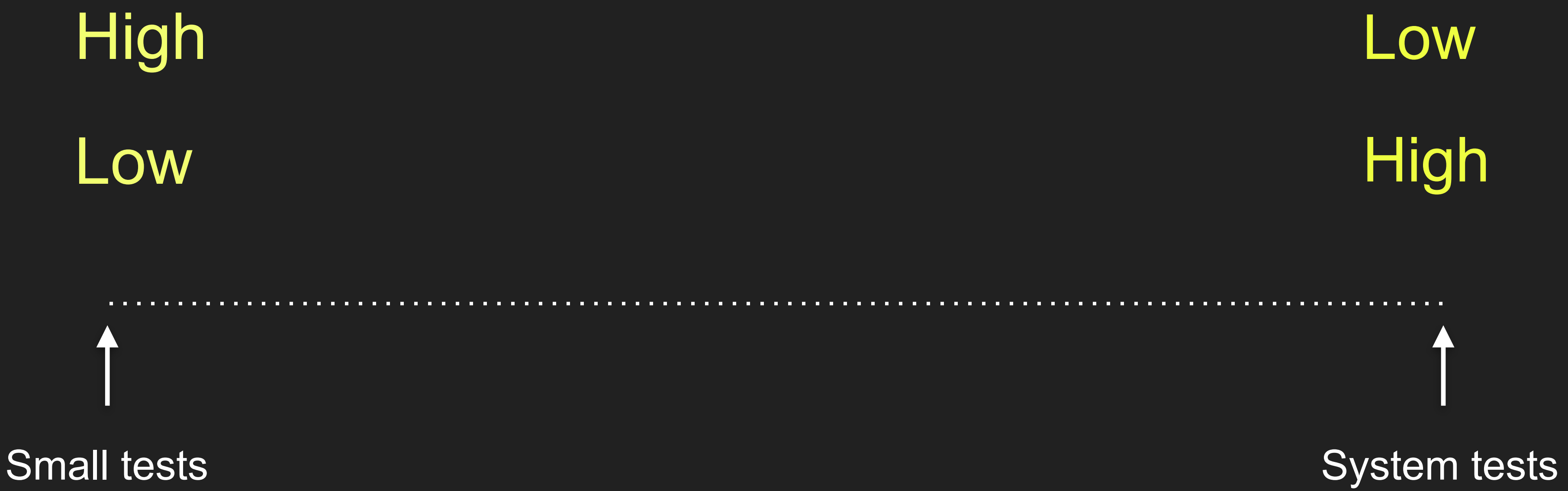


@DaveLiddament

Testing Continuum: Coverage

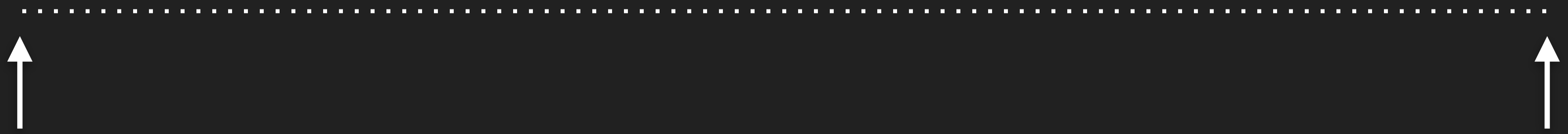


Testing Continuum: Coverage



Testing continuum
#3 Low maintenance

Testing Continuum: Speed of writing

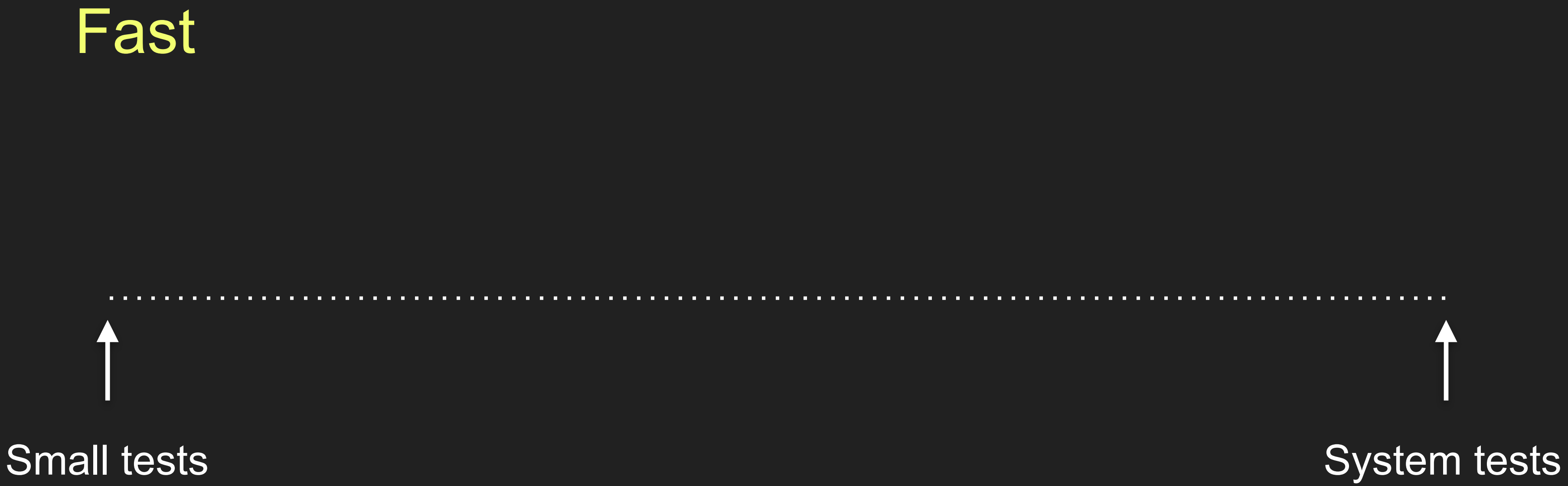


Small tests

System tests

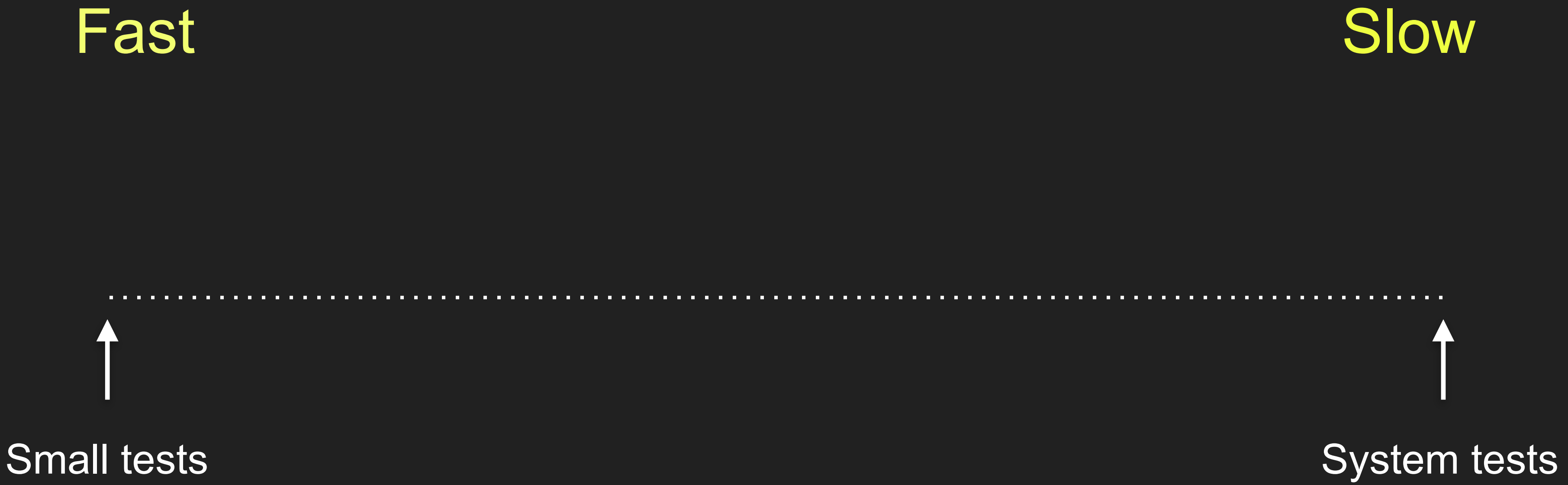
@DaveLiddament

Testing Continuum: Speed of writing



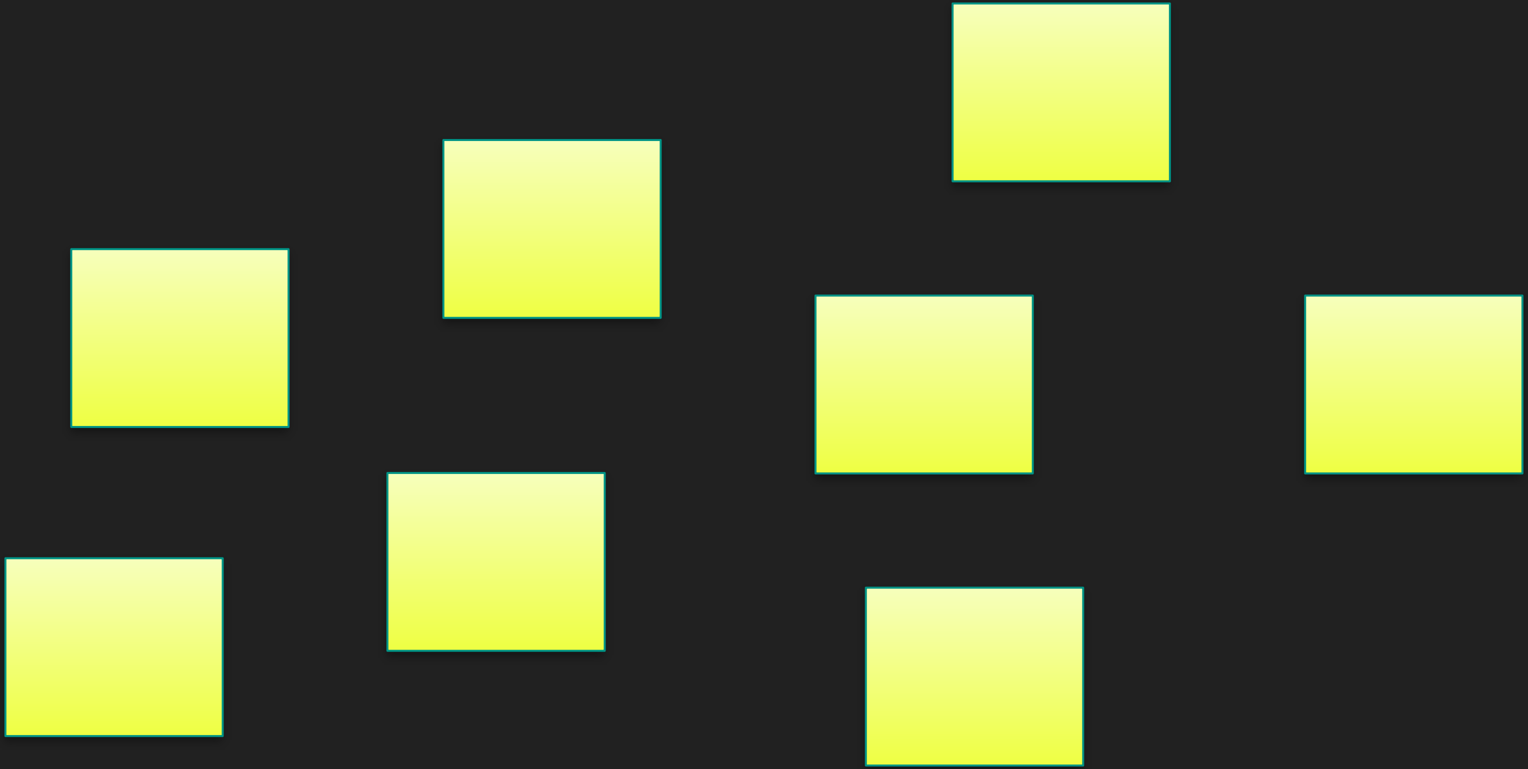
@DaveLiddament

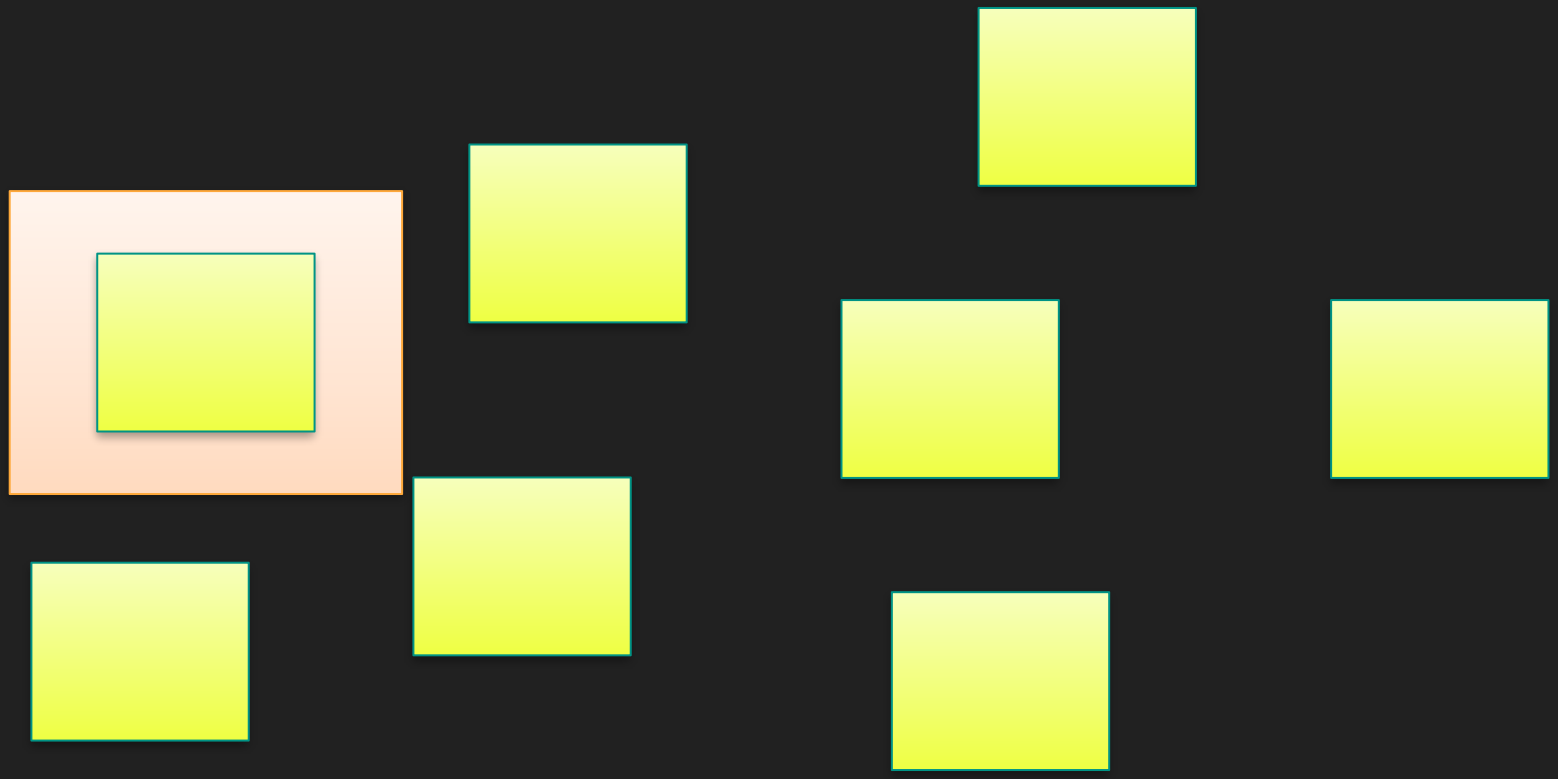
Testing Continuum: Speed of writing

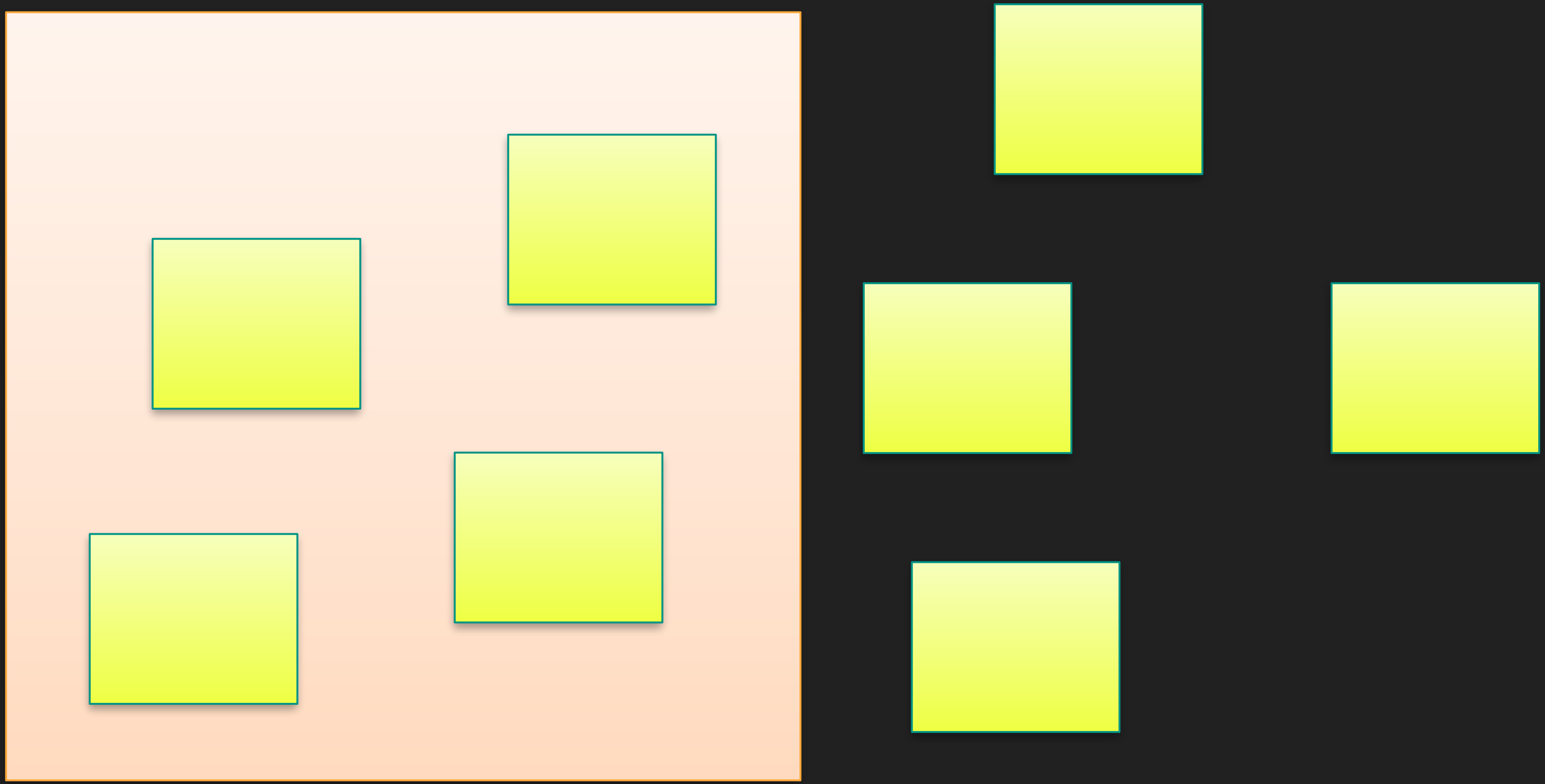


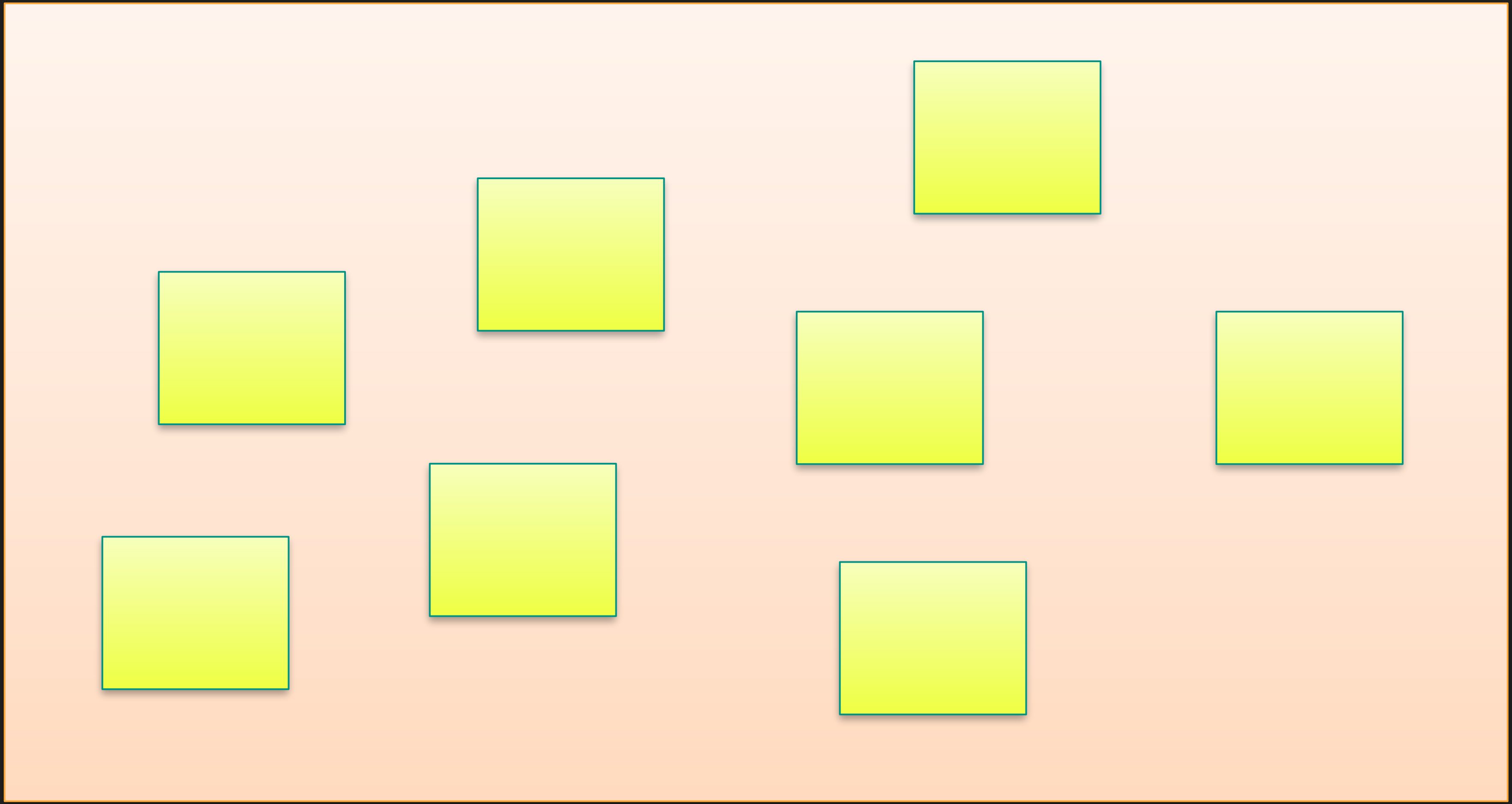
Testing Continuum: Debug speed

Small tests
System tests







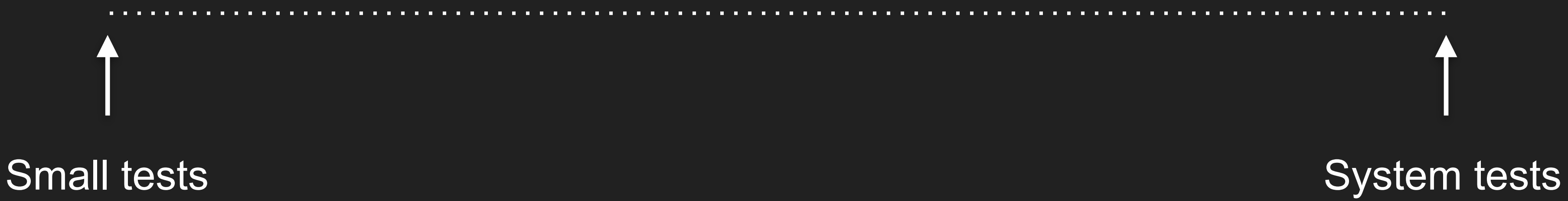


Testing Continuum: Debug speed

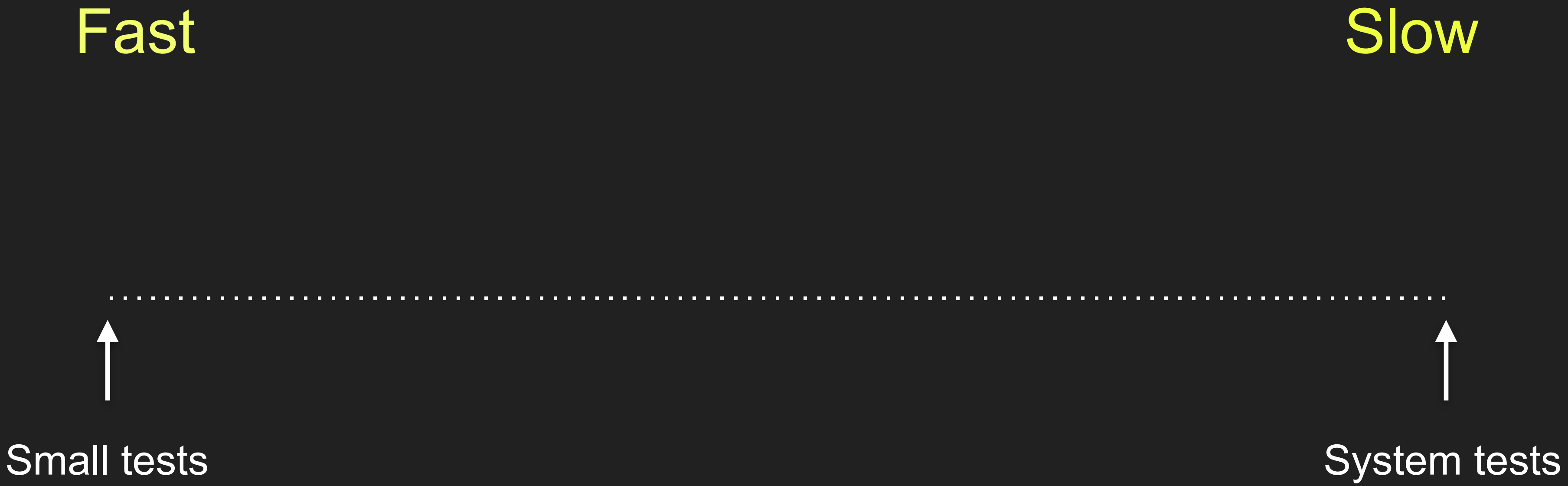
Small tests
System tests

Testing Continuum: Debug speed

Fast



Testing Continuum: Debug speed



@DaveLiddament

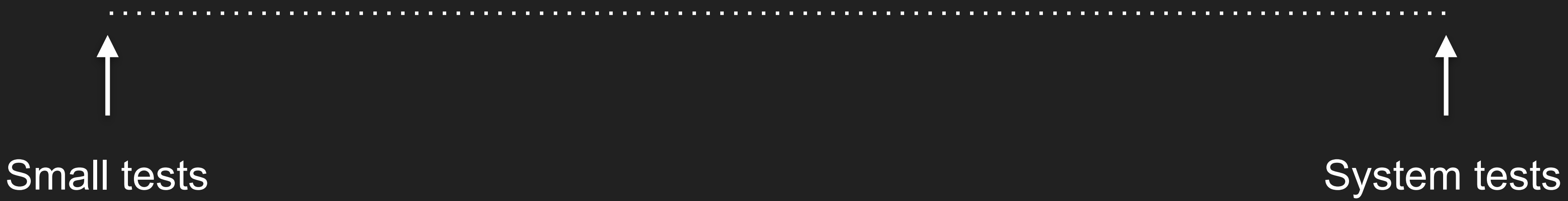
Testing Continuum: Robustness

Small tests
System tests

@DaveLiddament

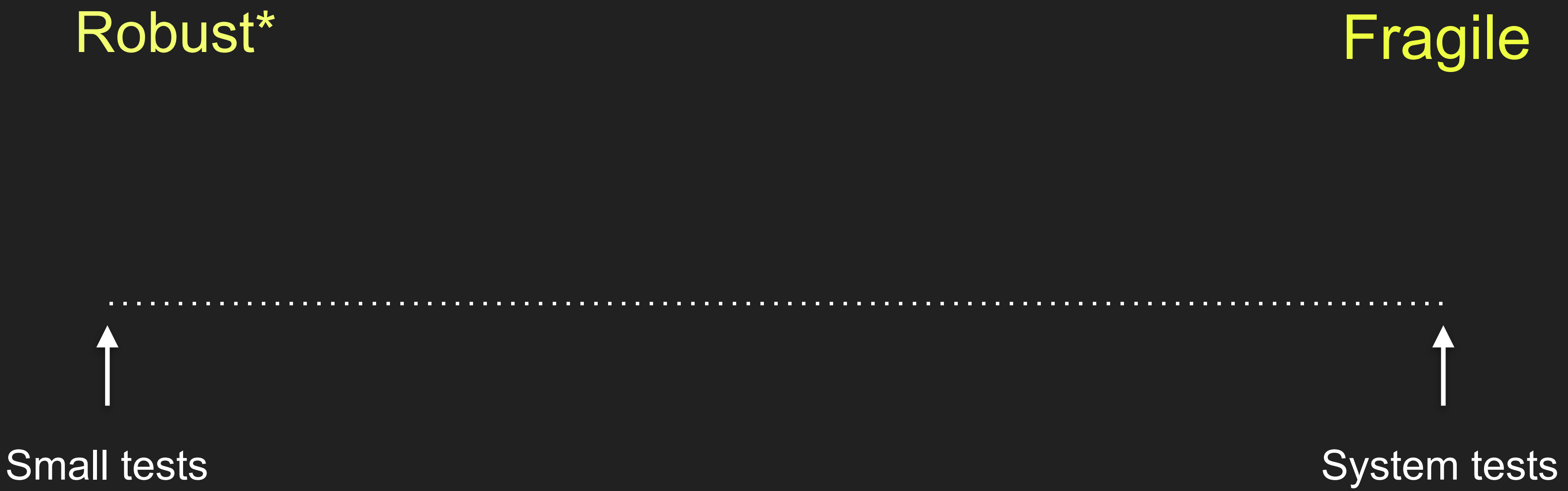
Testing Continuum: Robustness

Robust*



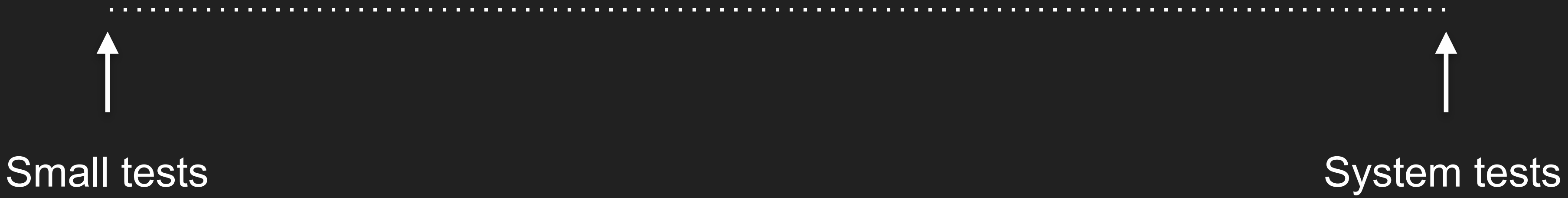
@DaveLiddament

Testing Continuum: Robustness



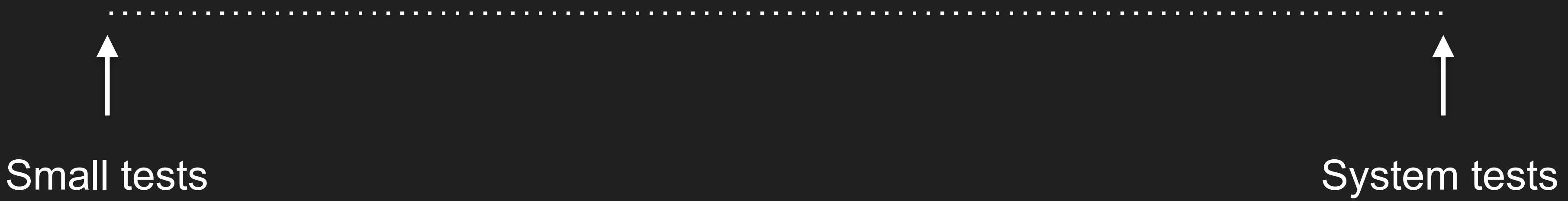
@DaveLiddament

Testing Continuum: Refactoring scope



Testing Continuum: Refactoring scope

Small



@DaveLiddament

Testing Continuum: Refactoring scope

Small

Large*

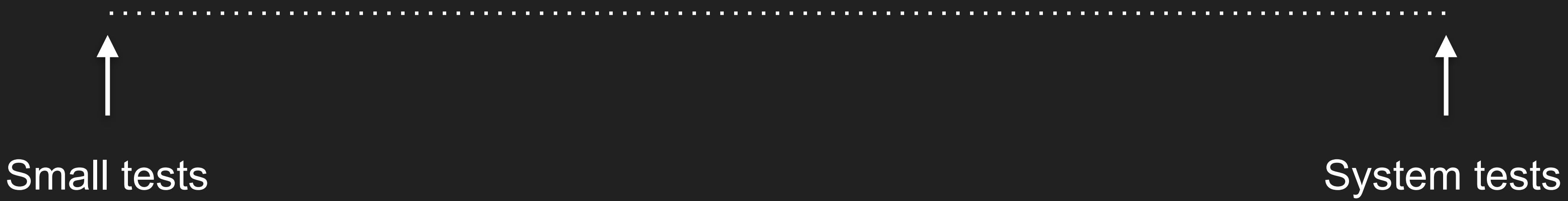
Small tests

System tests

@DaveLiddament

Other considerations

Testing Continuum: Phew factor



@DaveLiddament

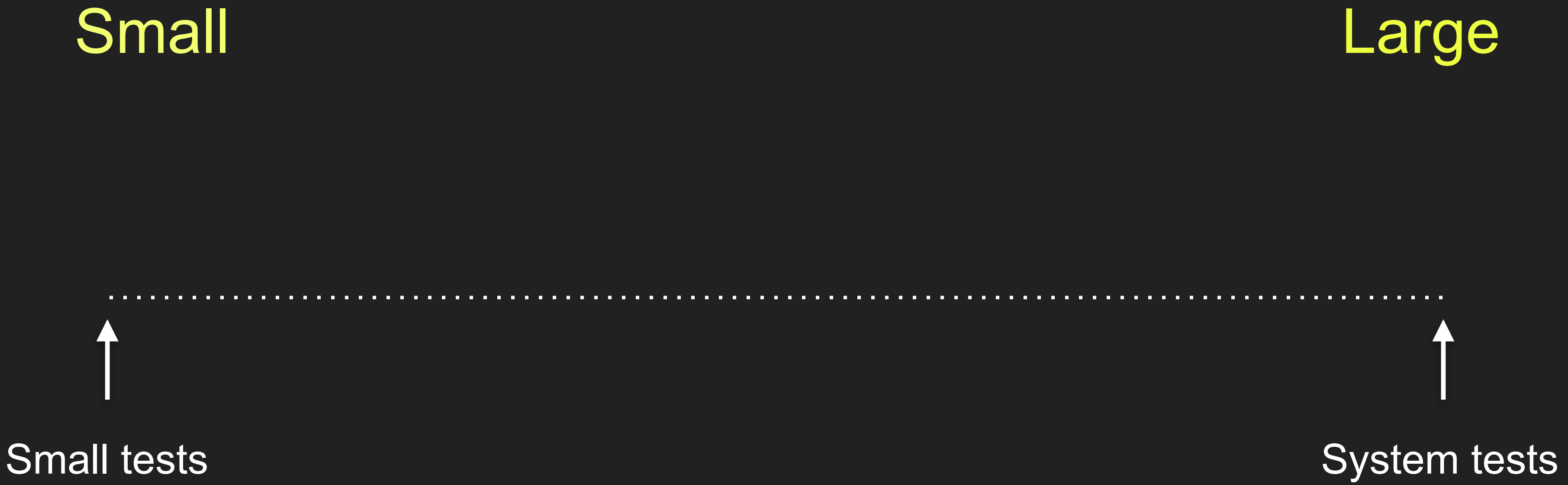
Testing Continuum: Phew factor

Small

Small tests
System tests

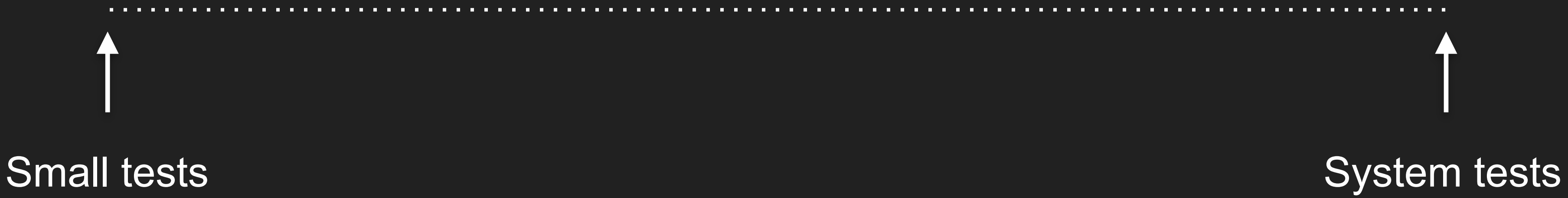
@DaveLiddament

Testing Continuum: Phew factor



@DaveLiddament

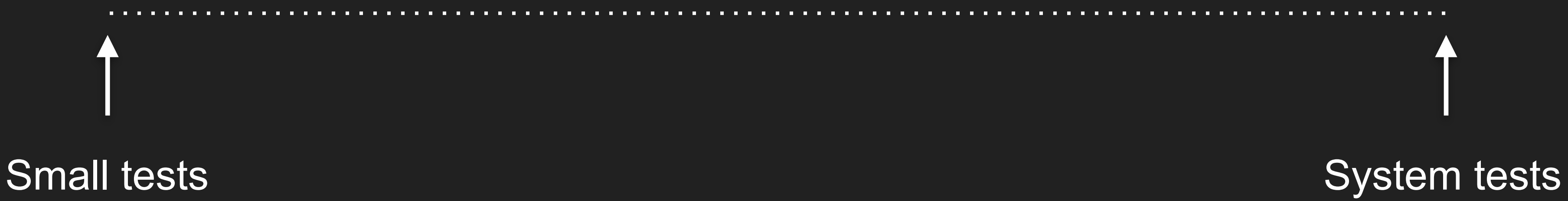
Testing Continuum: Bearing on reality



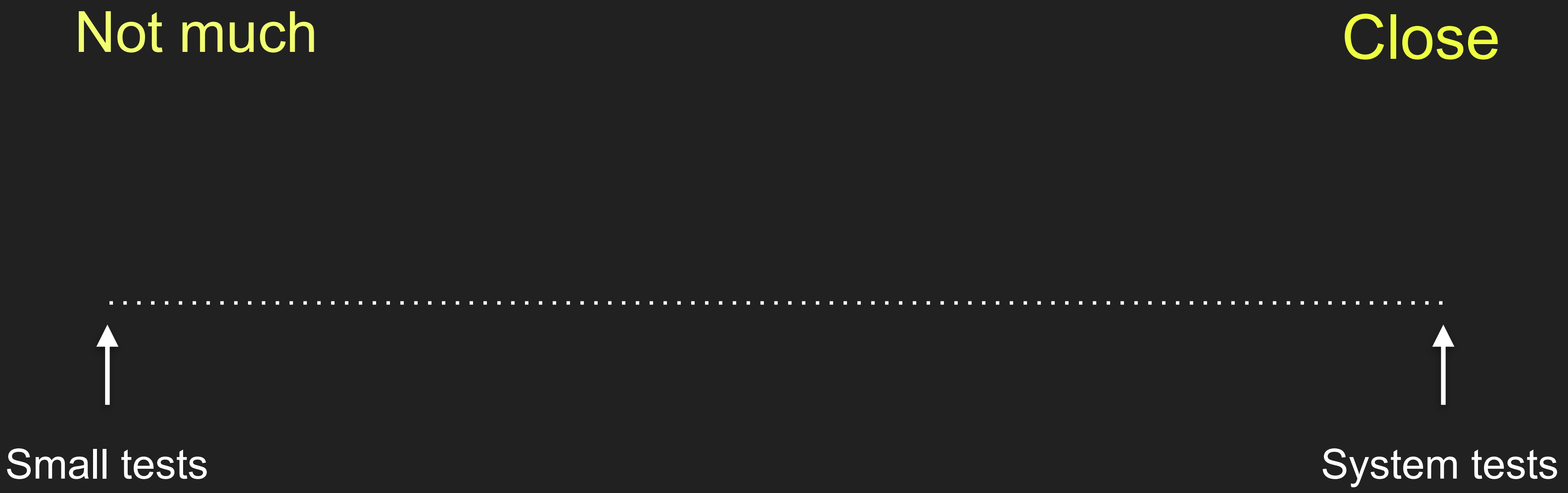
@DaveLiddament

Testing Continuum: Bearing on reality

Not much



Testing Continuum: Bearing on reality



So far nothing too controversial

Where along the testing continuum should we test?

Unit tests are dead

Integration tests are a con

Unit tests are dead

Integration tests are a con

Unit tests are dead

Mocking is dead

XXX

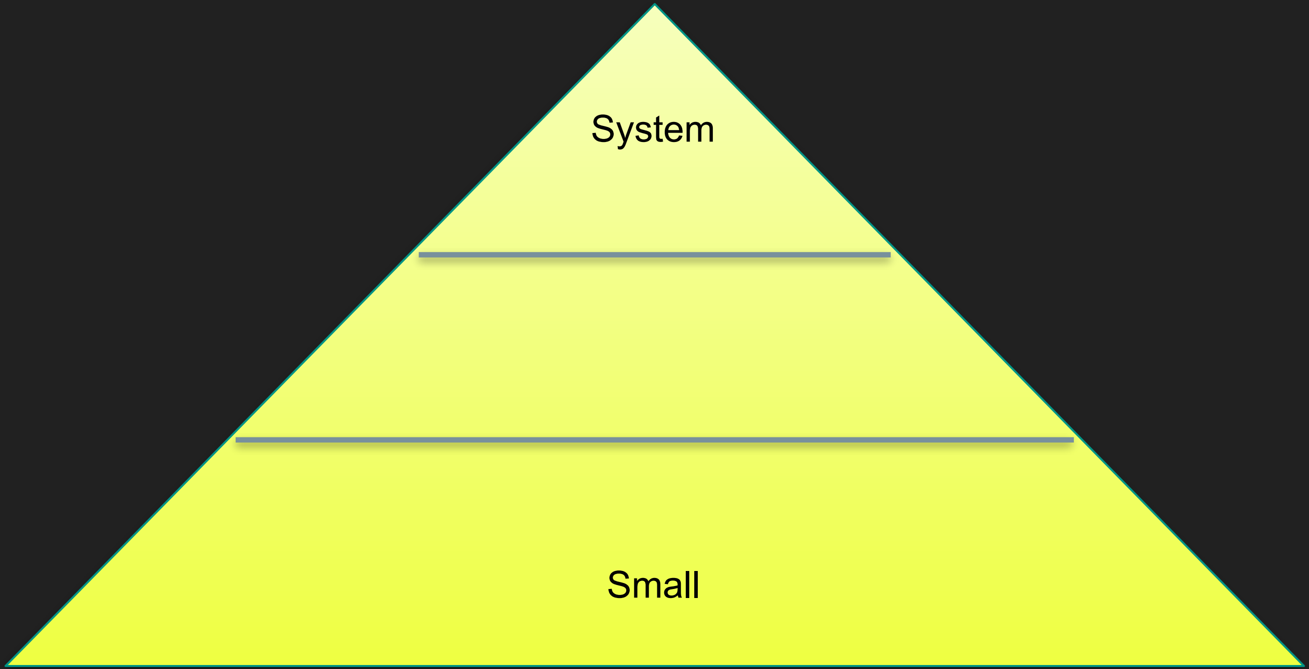
I tegra on tes are con

I it tests are ad

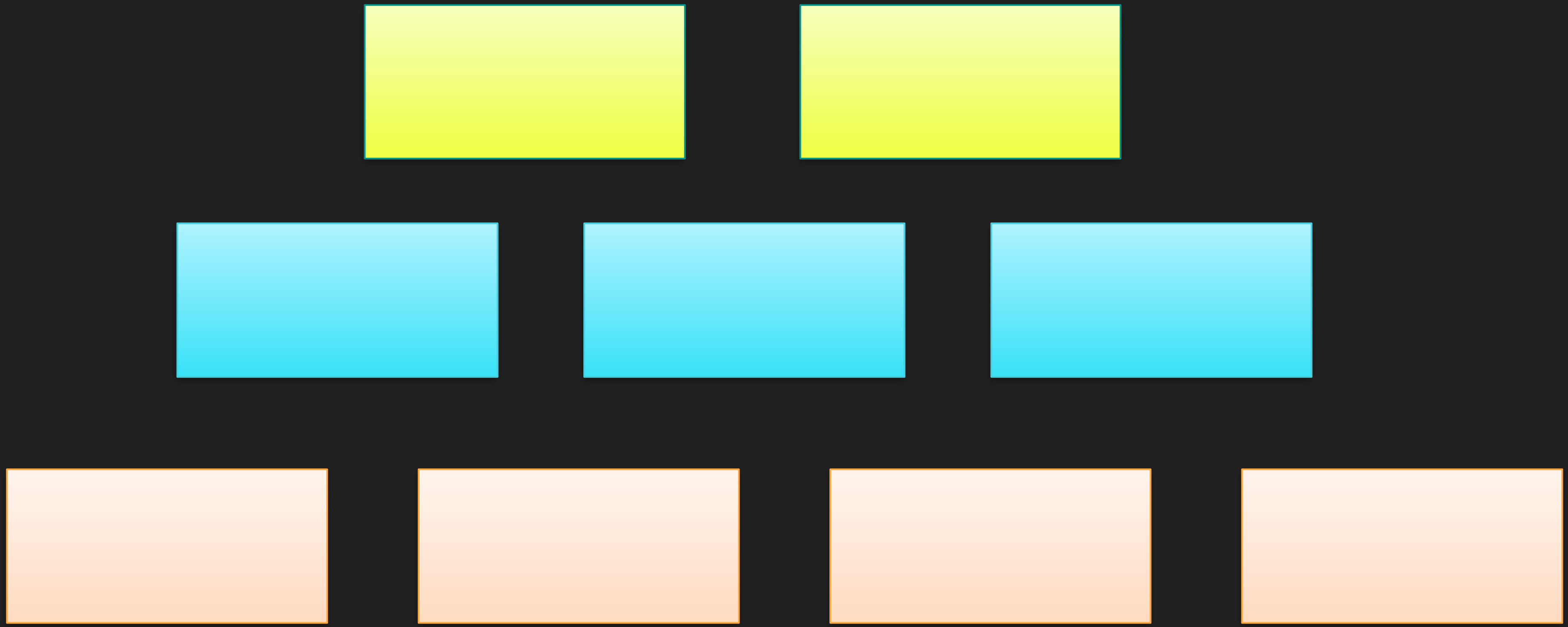
Mocki is ead

In my opinion…

Test Pyramid



Test in layers



Test in layers - we all do this

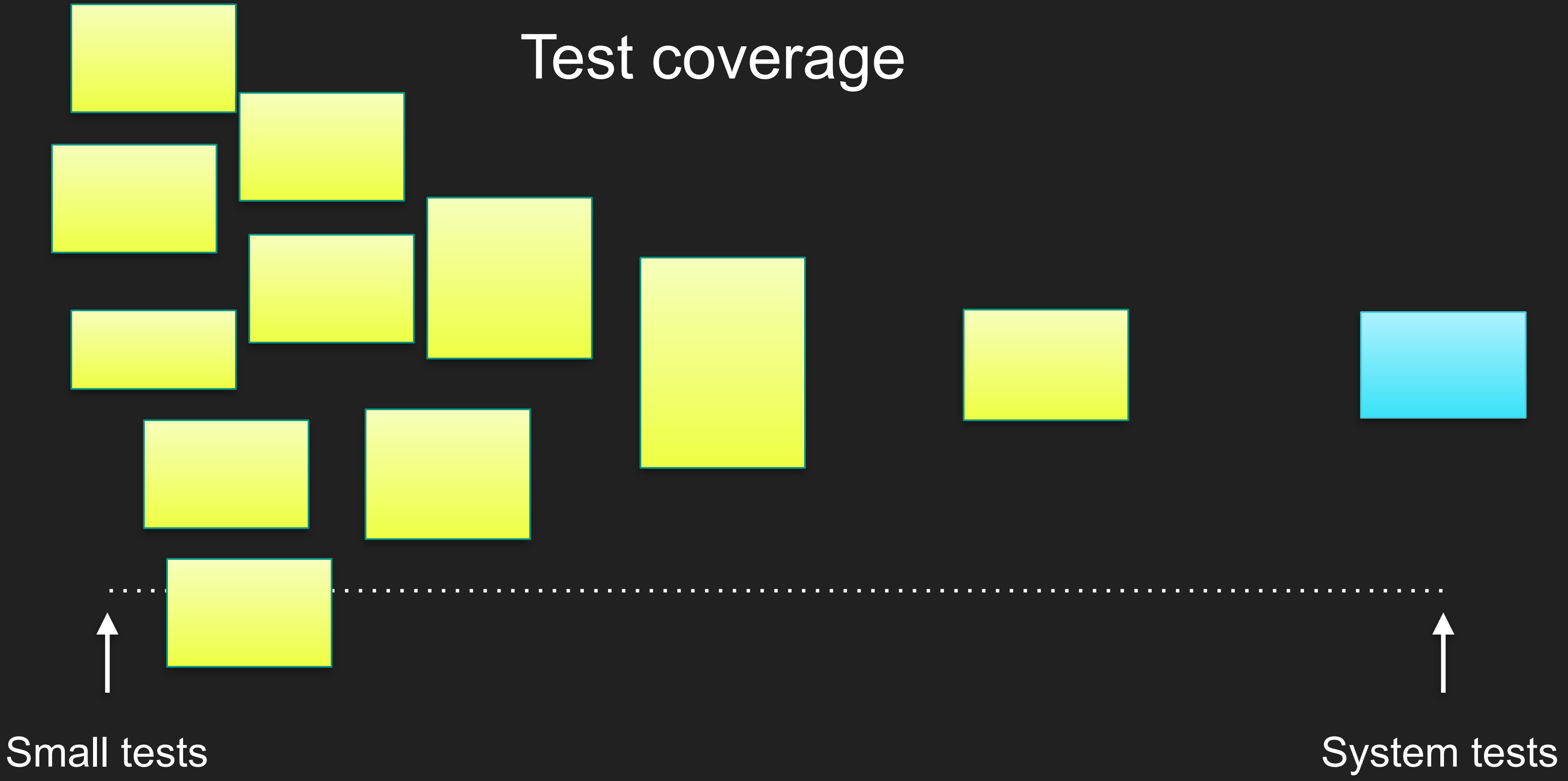
PHP application code

PHP instructions

Machine code running on computer

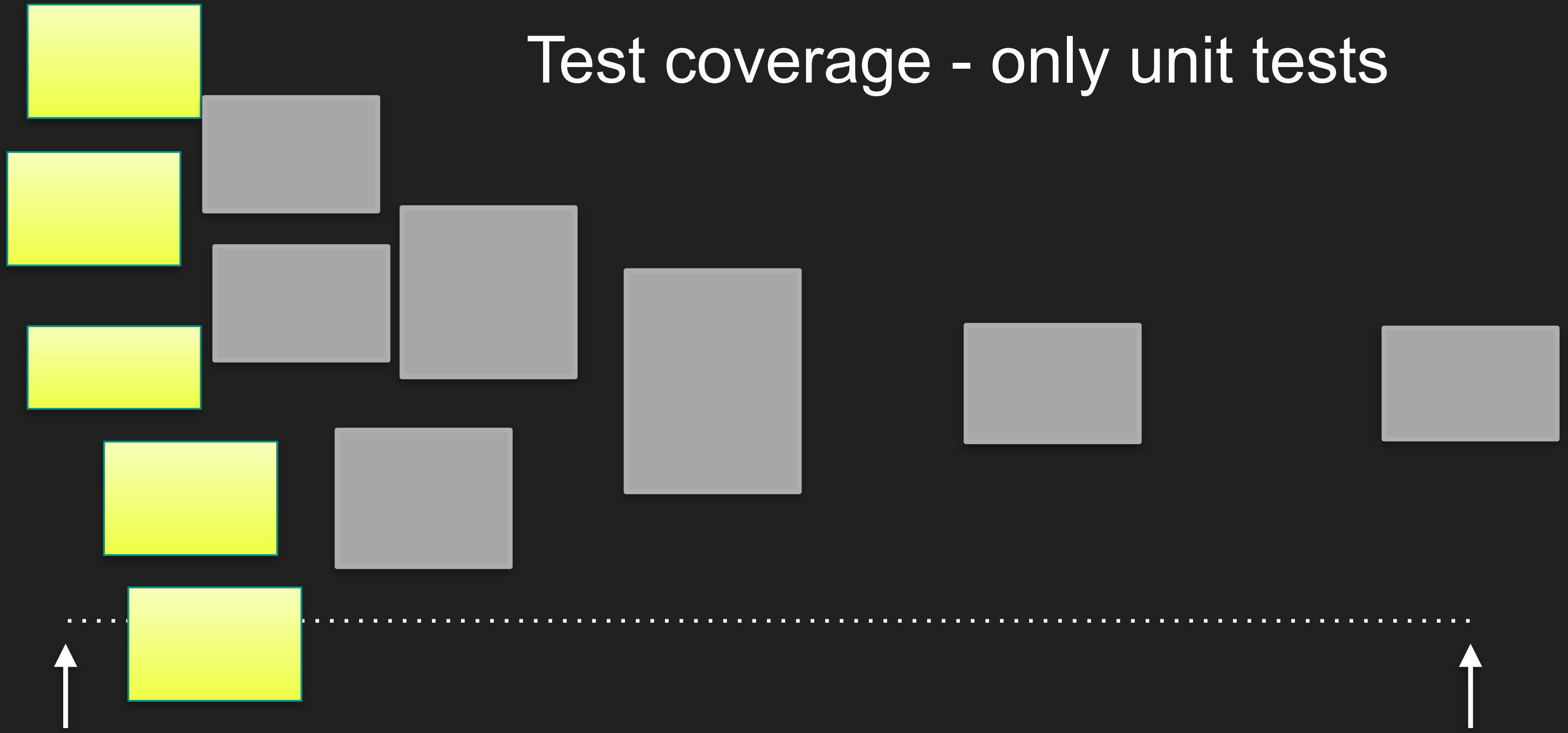
I’m going to transfer £100 to you

Test coverage



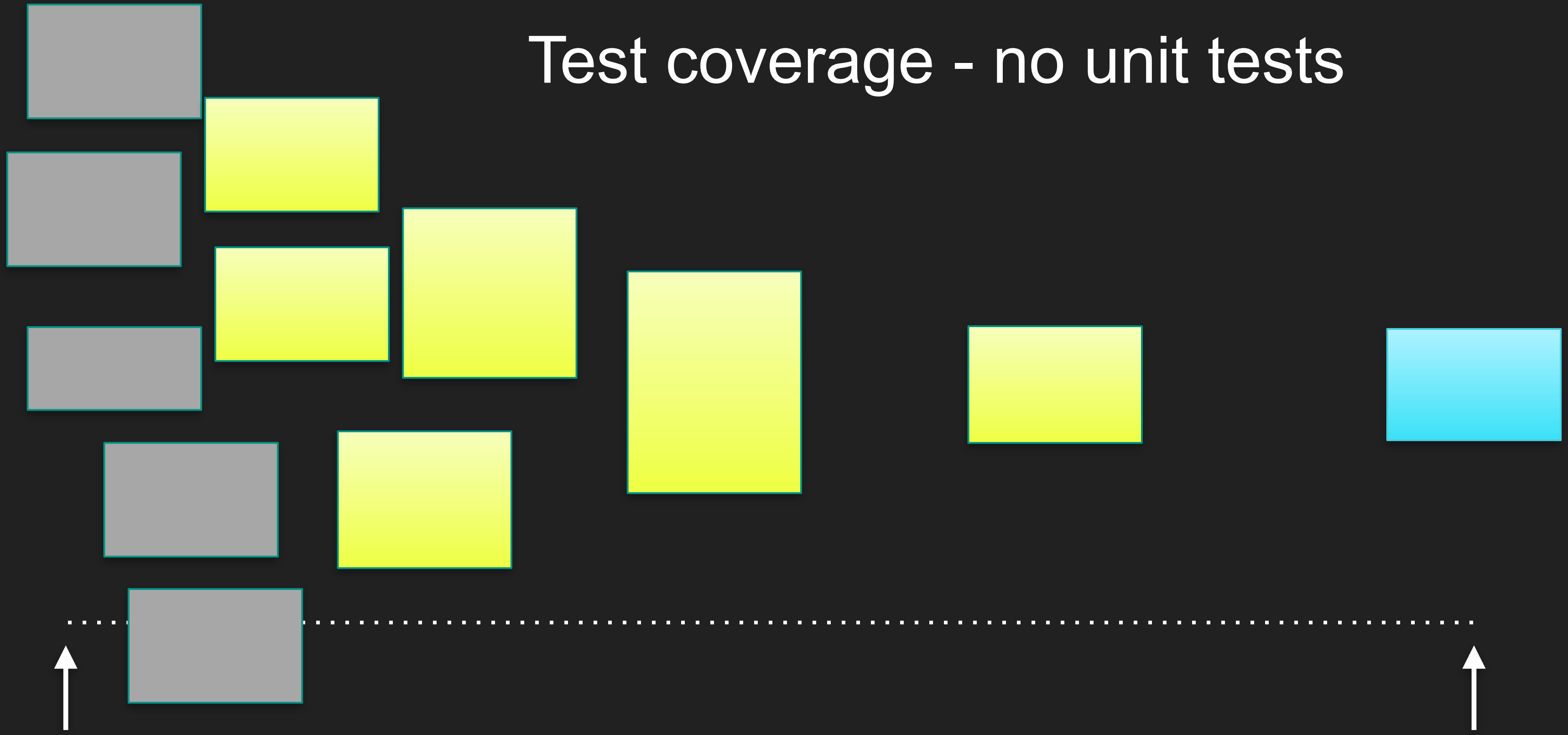
@DaveLiddament

Test coverage - only unit tests



@DaveLiddament

Test coverage - no unit tests



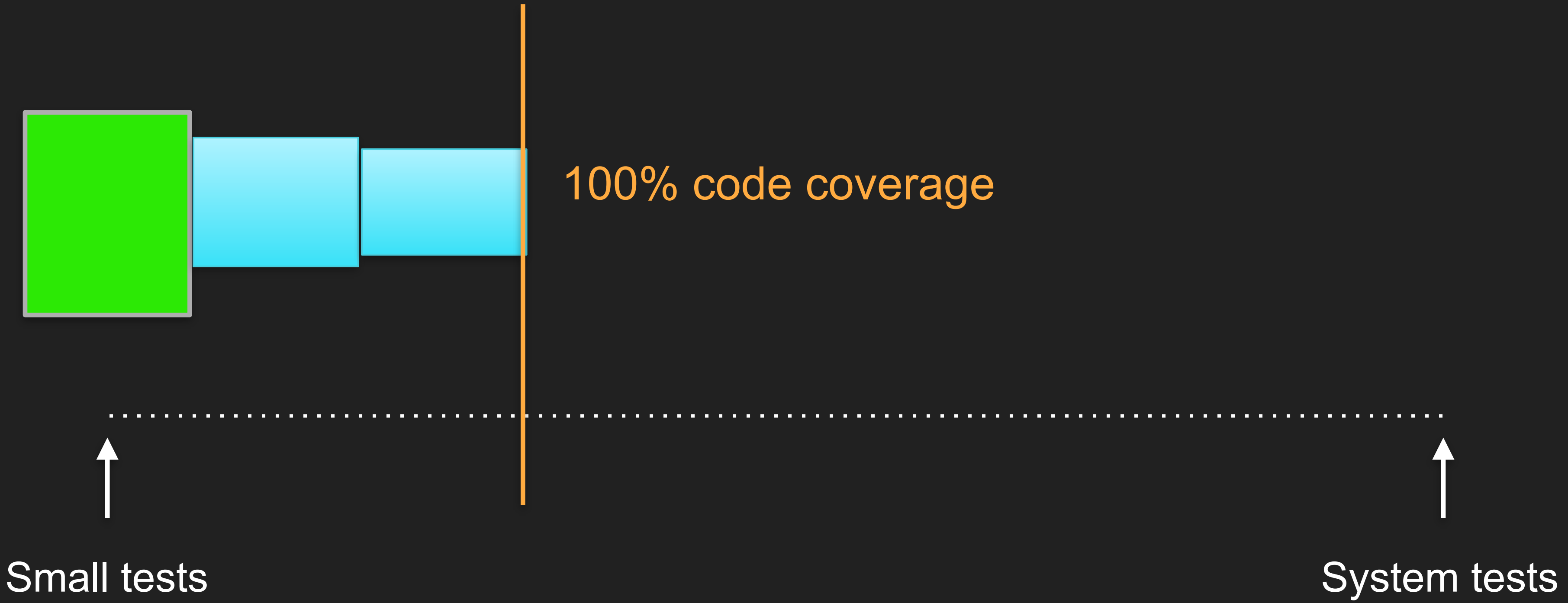
@DaveLiddament

Put the tests where there is highest value

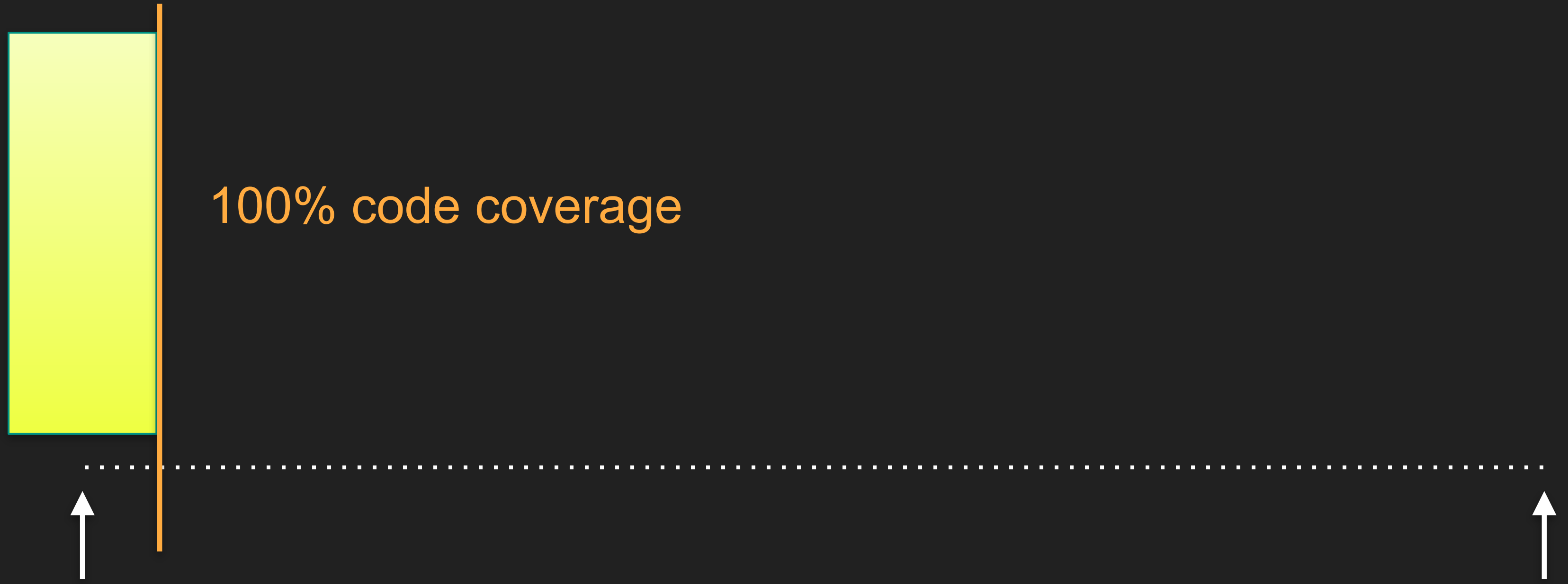
VALUE OF TEST

	LOW	HIGH
HIGH	Algorithms	Complex code
LOW	Trivial code	Coordinators

COST


100% code coverage
Small tests
System tests

Should all production code be ‘unit tested’ ?



Small tests

System tests

@DaveLiddament

A quick recap…

A test suite…

#1 Proves code works
#2 Stops regression
#3 Enables refactoring

The Holy Trinity…

#1 Fast to execute
#2 High coverage
#3 Low maintenance

Architecture

The codebase isn’t difficult to test, it’s poorly architected

Password Validator

```
class PasswordValidator
{

  /**
   * Returns true if password meets following criteria:
   *
   * - 8 or more characters
   * - at least 1 digit
   * - at least 1 upper case letter
   * - at least 1 lower case letter
   */
   public function isValid(string $password) : bool
```

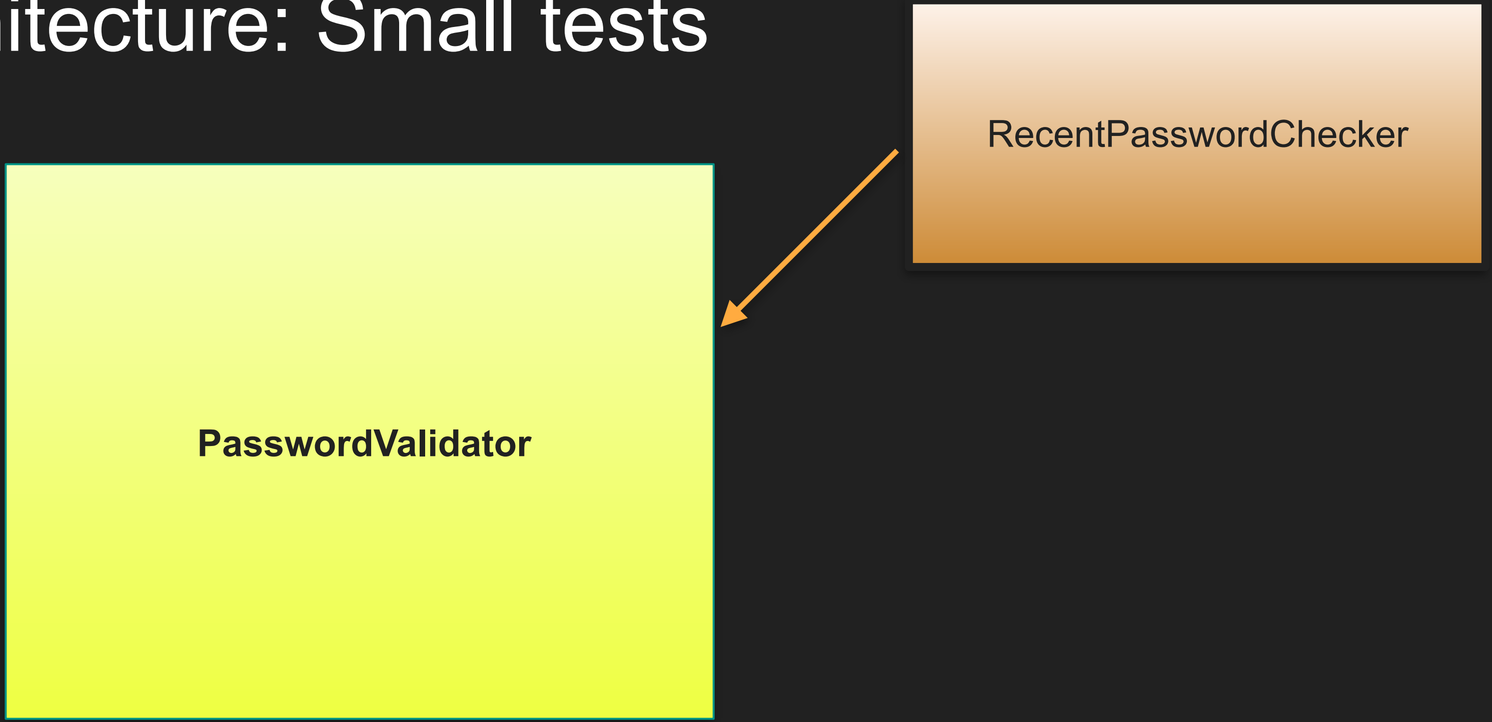
Extended Password Validator

```
class PasswordValidator
{

   /**
    * Returns true if password meets following criteria:
    *
    * - 8 or more characters
    * - at least 1 digit
    * - at least 1 upper case letter
    * - at least 1 lower case letter
    * - not one the previous user's 5 passwords
    */
    public function isValid(string $password, User $user) : bool
```

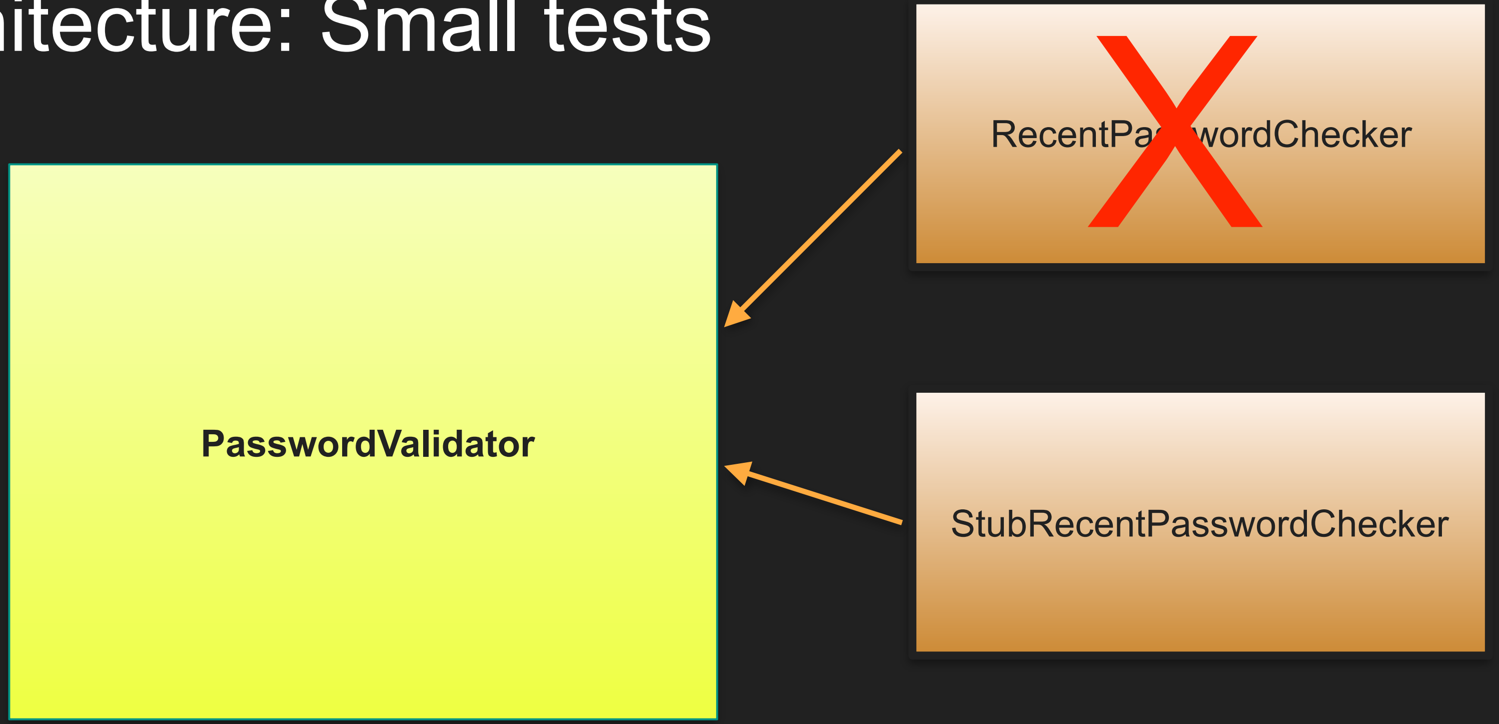
Password Validator - Checking Previous Passwords

```
interface PreviousPasswordChecker
{

  /**
   * Returns true if password has been used by user
   * in previous 5 passwords
   *
   */
  public function isRecentPassword(
   string $password, User $user) : bool
```

Architecture: Small tests



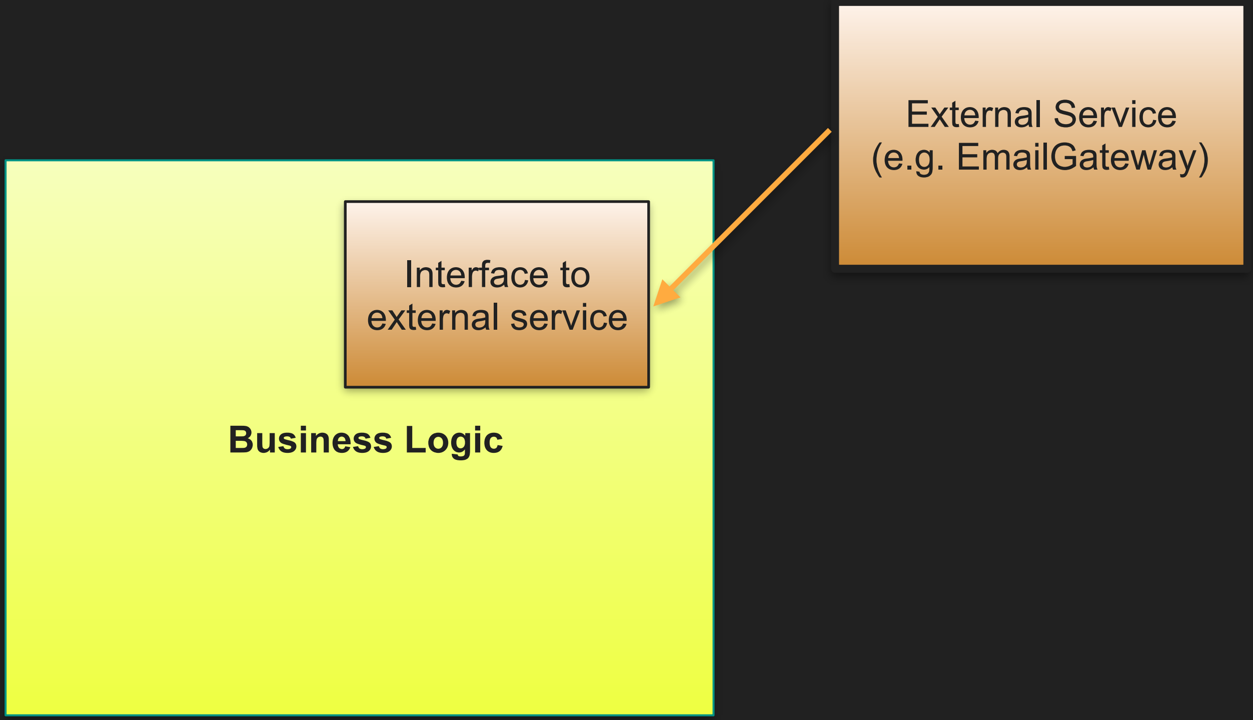
Architecture: Small tests



Architecture: Bigger tests

Business Logic

@DaveLiddament


External Service
(e.g. EmailGateway)
Interface to
external service
Business Logic

Email Gateway Interface

```
interface EmailGatewayInterface
{
    /**
     * Gateway for sending and email
     *
     * @param EmailMessage $message to send
     */
     public function sendEmail(EmailMessage $message);
}
```

@DaveLiddament

EmailMessage

To

From

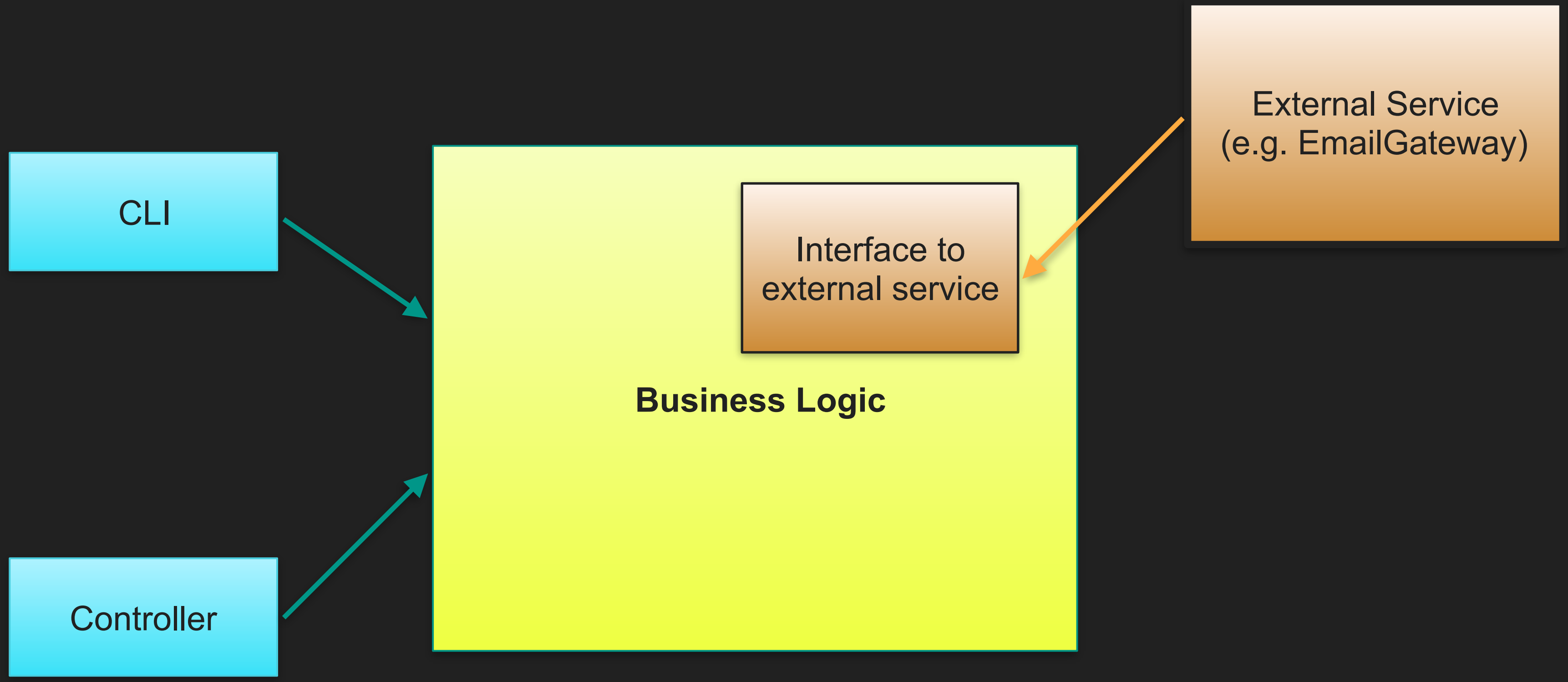
CC

Subject

Message Body

Template Name

Template Data



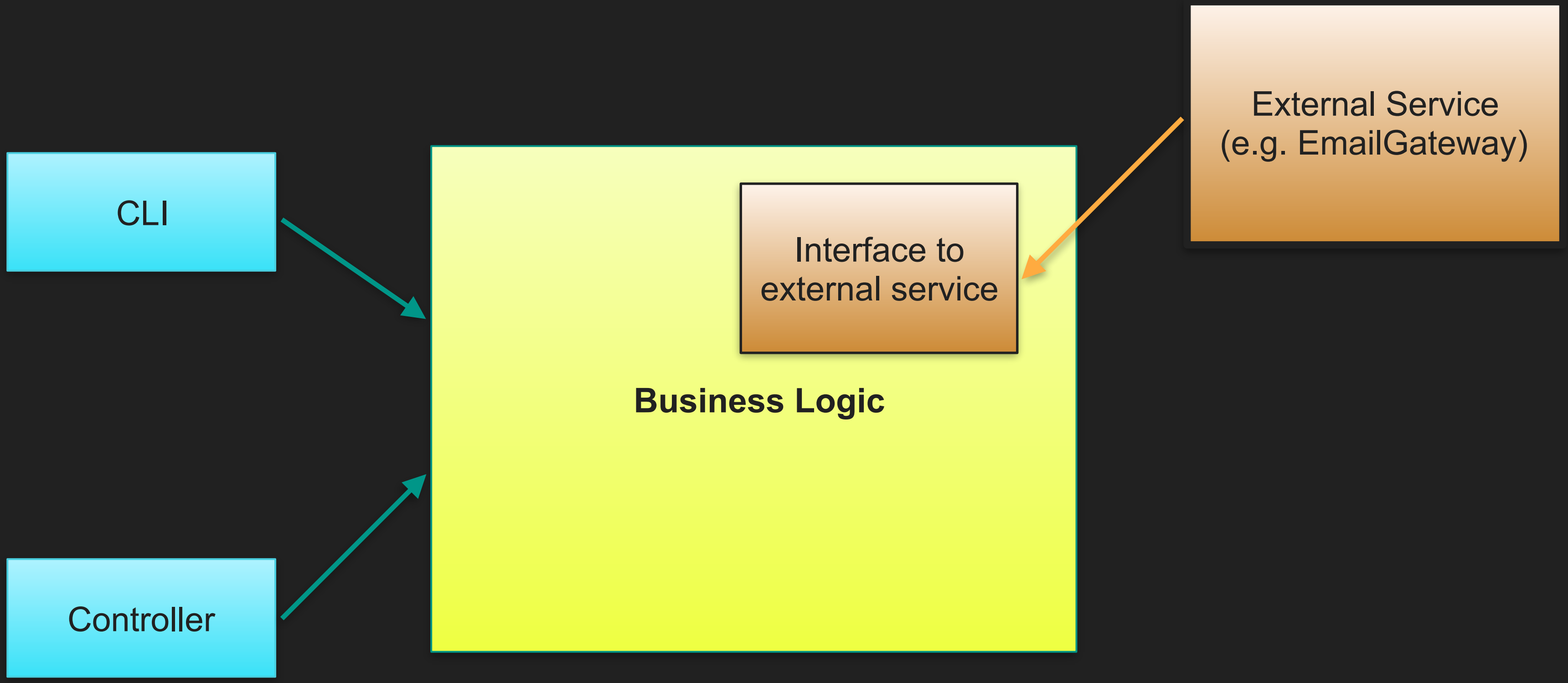
@DaveLiddament

Thin Controllers

```
class UserController
{
  public function confirmUser()
  {
    $token = Input::get("token");
    $success = $this->userService->confirmUser($token);

    if ($success) {
      // Handle success
    } else {
      // Handle failure
    }
  }
```

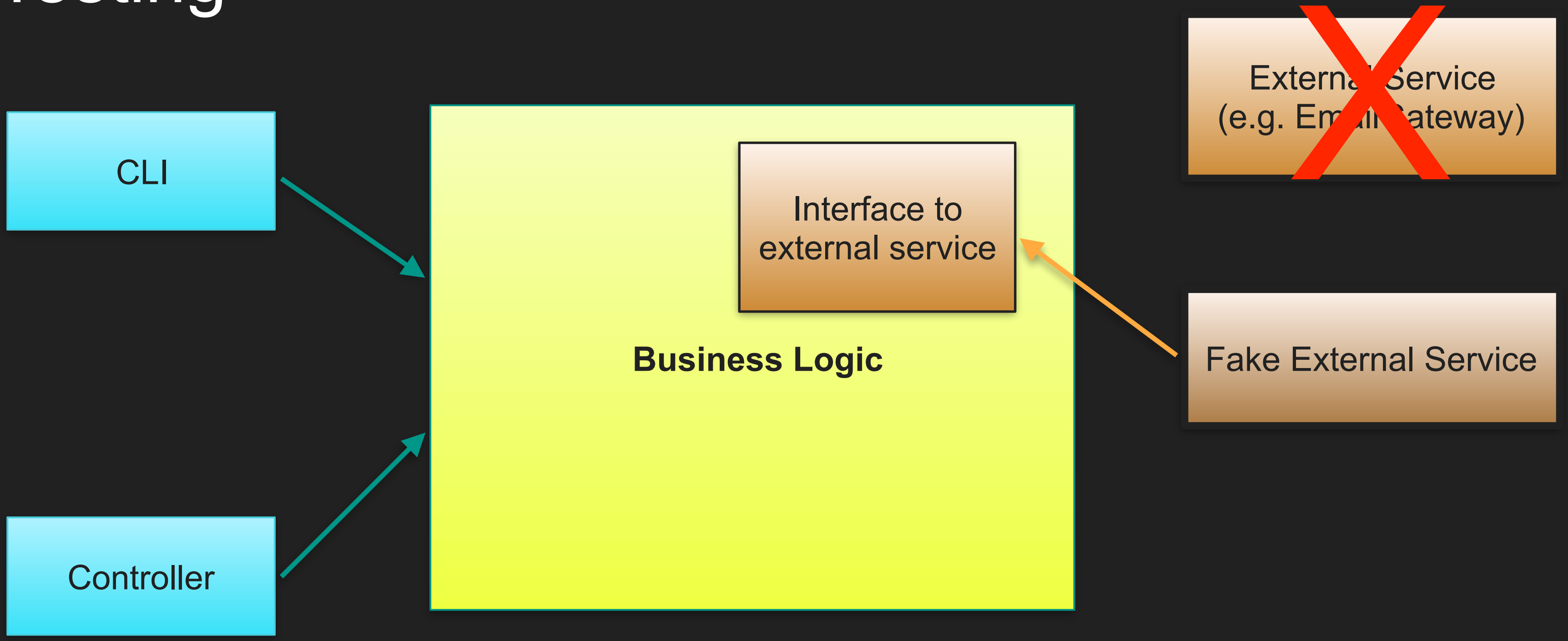
Thin Controllers

```
class UserController
{
        public function confirmUser()
        {
        $token = Input::get("token");
          $success = $this->userService->confirmUser($token);

        if ($success) {
          // Handle success
        } else {
          // Handle failure
        }
      }
```


CLI
Controller
Interface to external service
Business Logic
External Service (e.g. EmailGateway)

@DaveLiddament

Testing


CLI
Controller
Interface to external service
Business Logic
External Service (e.g. Email Gateway)
Fake External Service

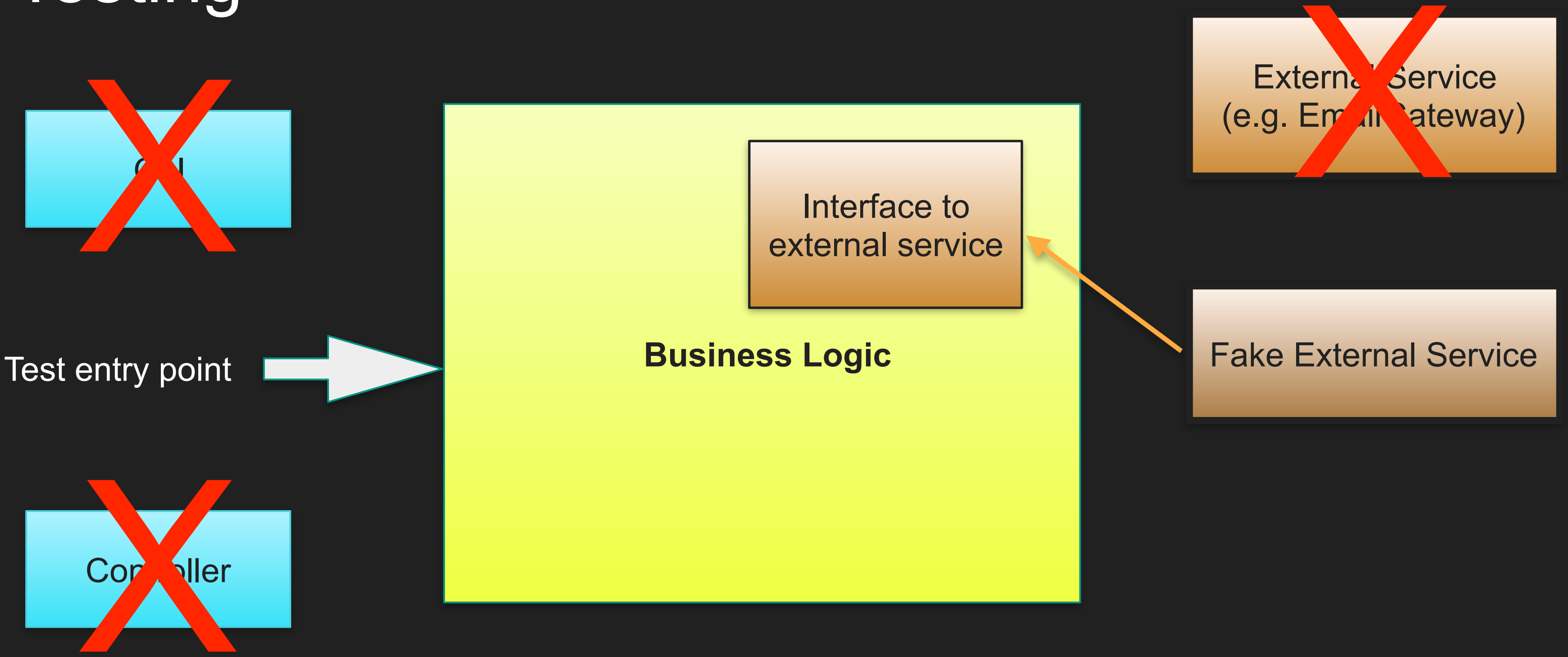
@DaveLiddament

Email Gateway Fake

```
class EmailGatewayFake implements EmailGatewayInterface
{
    public function sendEmail(EmailMessage $message)
    {
      /* implementation that stores all messages for searching */
    }

    /**
     * Find emails that would have been sent
     *
     * @param array $criteria e.g.:
     *       ['to' => 'dave@example.com', 'template' => 'RegisterUser']
     * @return EmailMessage[] messages that meet criteria
     */
     public function findEmails(array $criteria)

}
```

@DaveLiddament

Testing


GUI
Test entry point
Controller
Business Logic
Interface to external service
External Service (e.g. Email Gateway)
Fake External Service

@DaveLiddament

Testing User Registration

```
class PasswordValidatorTest extends AbstractTestCase
{
  public function testUpdatePassword()
  {
        // Get the UserService and register a new user
        $userService = $this->container->get("UserService");
        $userService->registerUser(“dave@example.com”, “1stPassword");

        // Get the EmailGatewayStub and find the registration email
        $emailGateway = $this->container->get("EmailGateway");
        $emails = $emailGateway->findEmails(
          [“to” => “dave@example.com”, “template” => “RegisterUser”]);
        $this->assertEquals(1, count($emails));

        // Get confirmation token from the registration email
        $data = $emails[0]->getData();
        $confirmationToken = $data[“confirmationToken”];

        // Complete registration
        $this->assertTrue($userService->confirmUser($confirmationToken));
```

@DaveLiddament

Testing User Registration

```
class PasswordValidatorTest extends AbstractTestCase
{
  public function testUpdatePassword()
  {
        // Get the UserService and register a new user
        $userService = $this->container->get("UserService");
        $userService->registerUser("dave@example.com", "1stPassword");

        // Get the EmailGatewayStub and find the registration email
        $emailGateway = $this->container->get("EmailGateway");
        $emails = $emailGateway->findEmails(
          ["to" => "dave@example.com", "template" => "RegisterUser"]);
        $this->assertEquals(1, count($emails));

        // Get confirmation token from the registration email
        $data = $emails[0]->getData();
        $confirmationToken = $data["confirmationToken"];

        // Complete registration
        $this->assertTrue($userService->confirmUser($confirmationToken));
```

@DaveLiddament

Testing User Registration

```
class PasswordValidatorTest extends AbstractTestCase
{
  public function testUpdatePassword()
  {
          // Get the UserService and register a new user
          $userService = $this->container->get("UserService");
          $userService->registerUser(“dave@example.com”, “1stPassword");

          // Get the EmailGatewayStub and find the registration email
          $emailGateway = $this->container->get("EmailGateway");
          $emails = $emailGateway->findEmails(
            [“to” => “dave@example.com”, “template” => “RegisterUser”]);
          $this->assertEquals(1, count($emails));

          // Get confirmation token from the registration email
          $data = $emails[0]->getData();
          $confirmationToken = $data[“confirmationToken”];

          // Complete registration
          $this->assertTrue($userService->confirmUser($confirmationToken));
```

@DaveLiddament

Testing User Registration

```
class PasswordValidatorTest extends AbstractTestCase
{
  public function testUpdatePassword()
  {
          // Get the UserService and register a new user
          $userService = $this->container->get("UserService");
          $userService->registerUser("dave@example.com", "1stPassword");

          // Get the EmailGatewayStub and find the registration email
          $emailGateway = $this->container->get("EmailGateway");
          $emails = $emailGateway->findEmails(
            ["to" => "dave@example.com", "template" => "RegisterUser"]);
          $this->assertEquals(1, count($emails));

          // Get confirmation token from the registration email
          $data = $emails[0]->getData();
          $confirmationToken = $data["confirmationToken"];

          // Complete registration
          $this->assertTrue($userService->confirmUser($confirmationToken));
```

@DaveLiddament

A codebase that’s
easy to test
is probably
well architected

3 take aways…

#1 We need a test suite

- Proves code works
- Stops regression
- Enables refactoring

#2 Ideal test suite…

- Fast to execute
- High coverage
- Low maintenance

#3 Write testable code

- Well architected
- Easy to maintain
- Happier customers

Questions

Bonus slides 1

Can we automate anything else?

Automating as much as we can:

```
php bin/console test:emailgateway --to dave@lampbristol.com

Sending email:
To      [dave@lampbristol.com]
From    [test@lampbristol.com]
CC      [dave+1@lampbristol.com]
Subject [Test email 2016-02-08 19:37]
Body    [Hi,
         This is a test email.
         Sent at 2016-02-08 19:37.
         From your tester]
```