

DAVE LIDDAMENT

TEST SUITE HOLY TRINITY

**LET'S START WITH
A STORY...**

WHY ARE WE HERE

- ▶ What went wrong
- ▶ Why testing will help
- ▶ How can we build a good test suite
 - ▶ Only talking asserting correct functionality

Dave Liddament

@daveliddament

Lamp Bristol



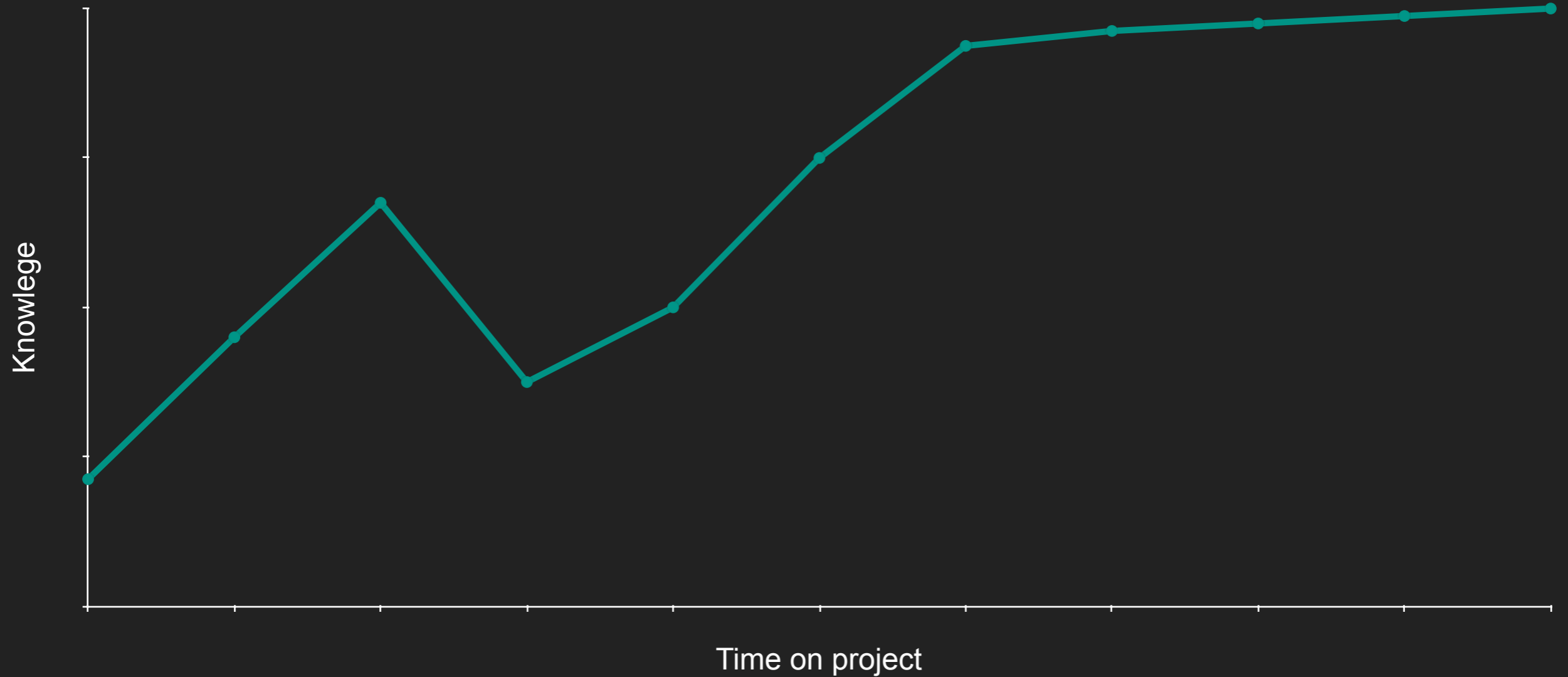
Organise PHP-SW and Bristol PHP Training

**WHAT WENT
WRONG?**

**TESTING FEEDBACK
LOOP WAS TOO SLOW**

**REFACTORING
WAS TOO RISKY**

WE WILL ALL MAKE POOR DECISIONS AT THE START OF A PROJECT



WE NEED A TEST SUITE

- ▶ Prove that code works
- ▶ Prevent regression
- ▶ Allow us to refactor

IDEAL TEST SUITE

Fast

High coverage

Low maintenance

THE IDEAL TEST SUITE

- ▶ Fast
- ▶ High coverage
- ▶ Low maintenance

TERMINOLOGY

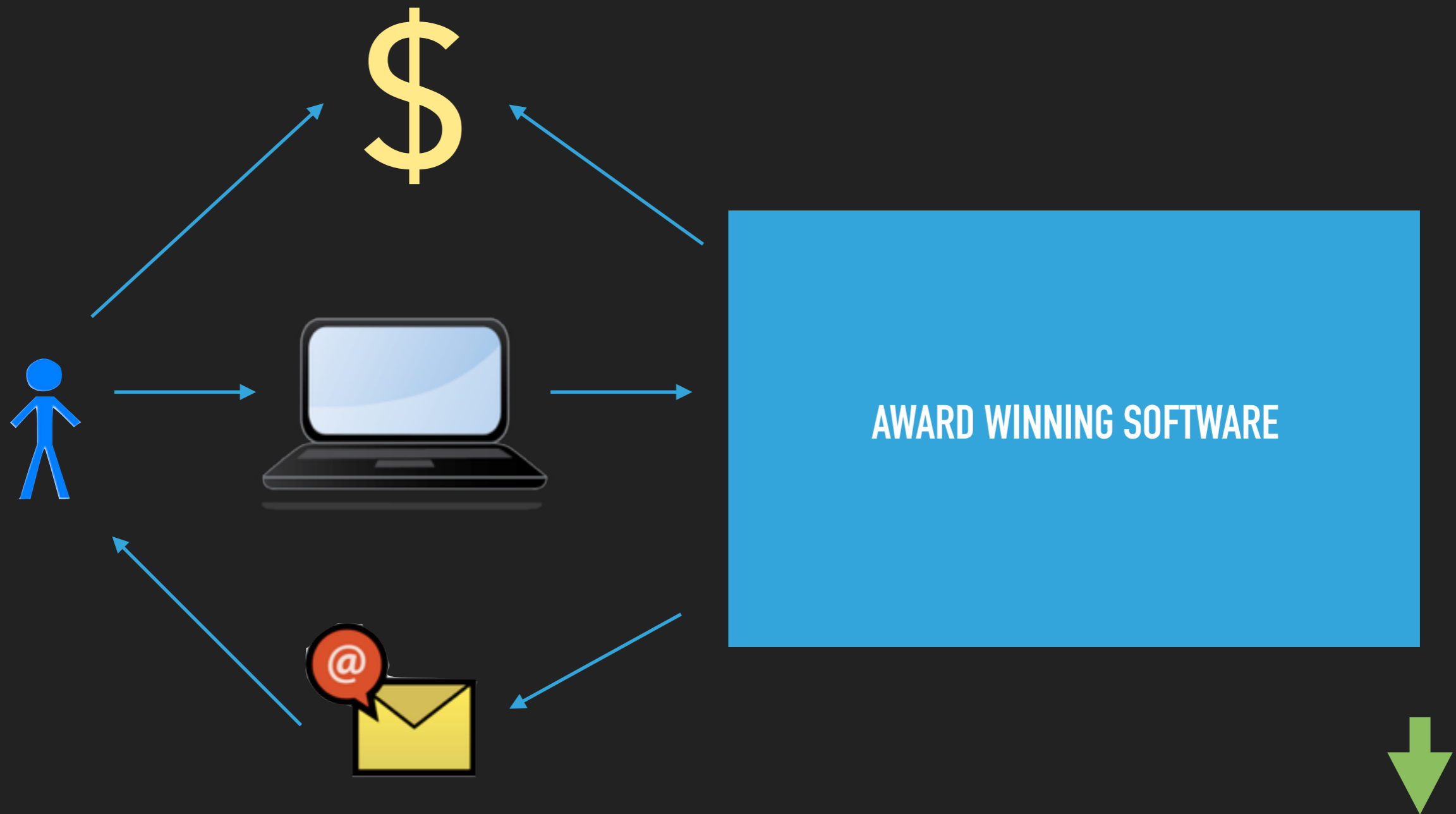
TESTING CONTINUUM

TESTING CONTINUUM

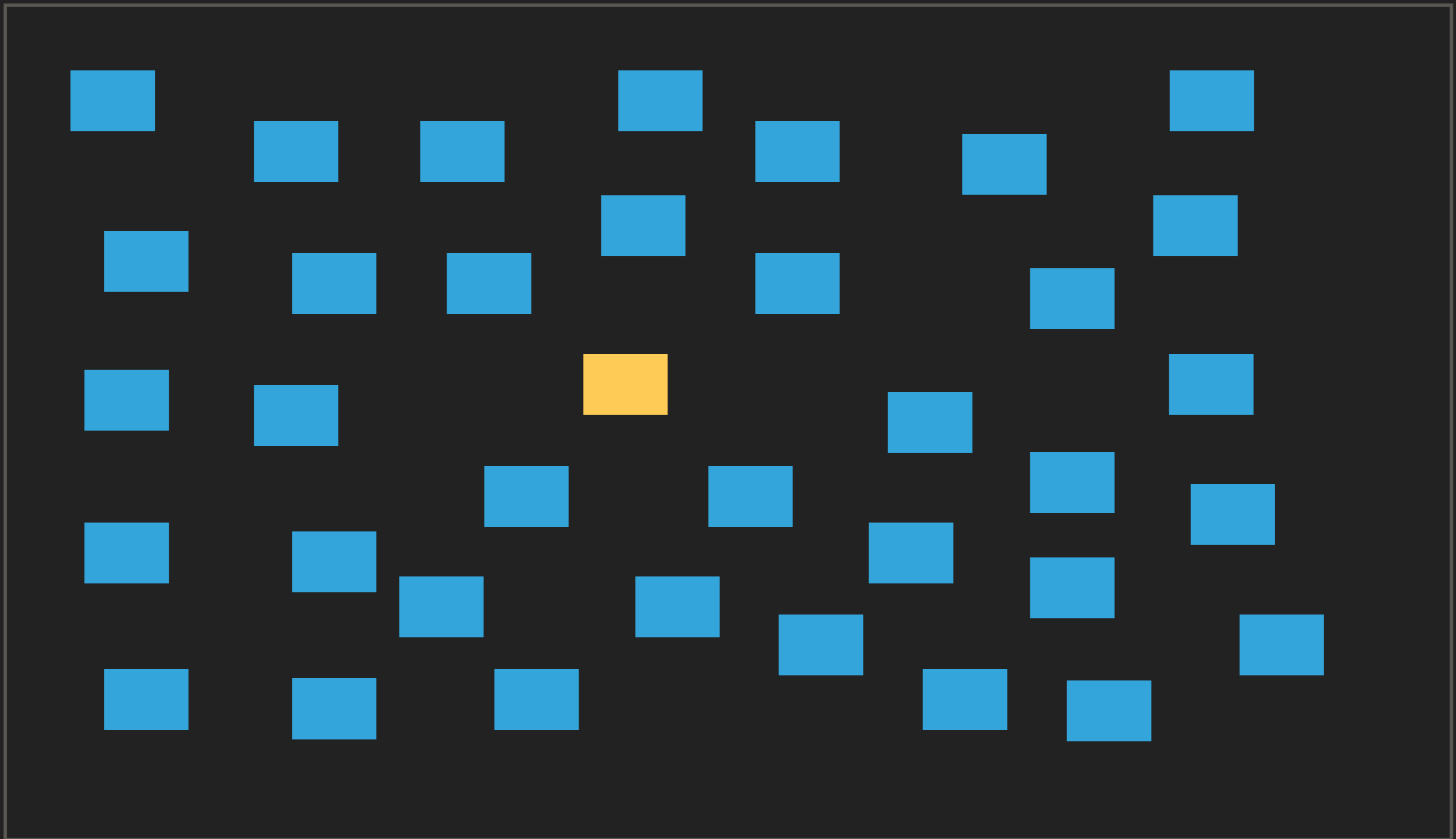
Unit tests

Systems tests

SYSTEM TEST



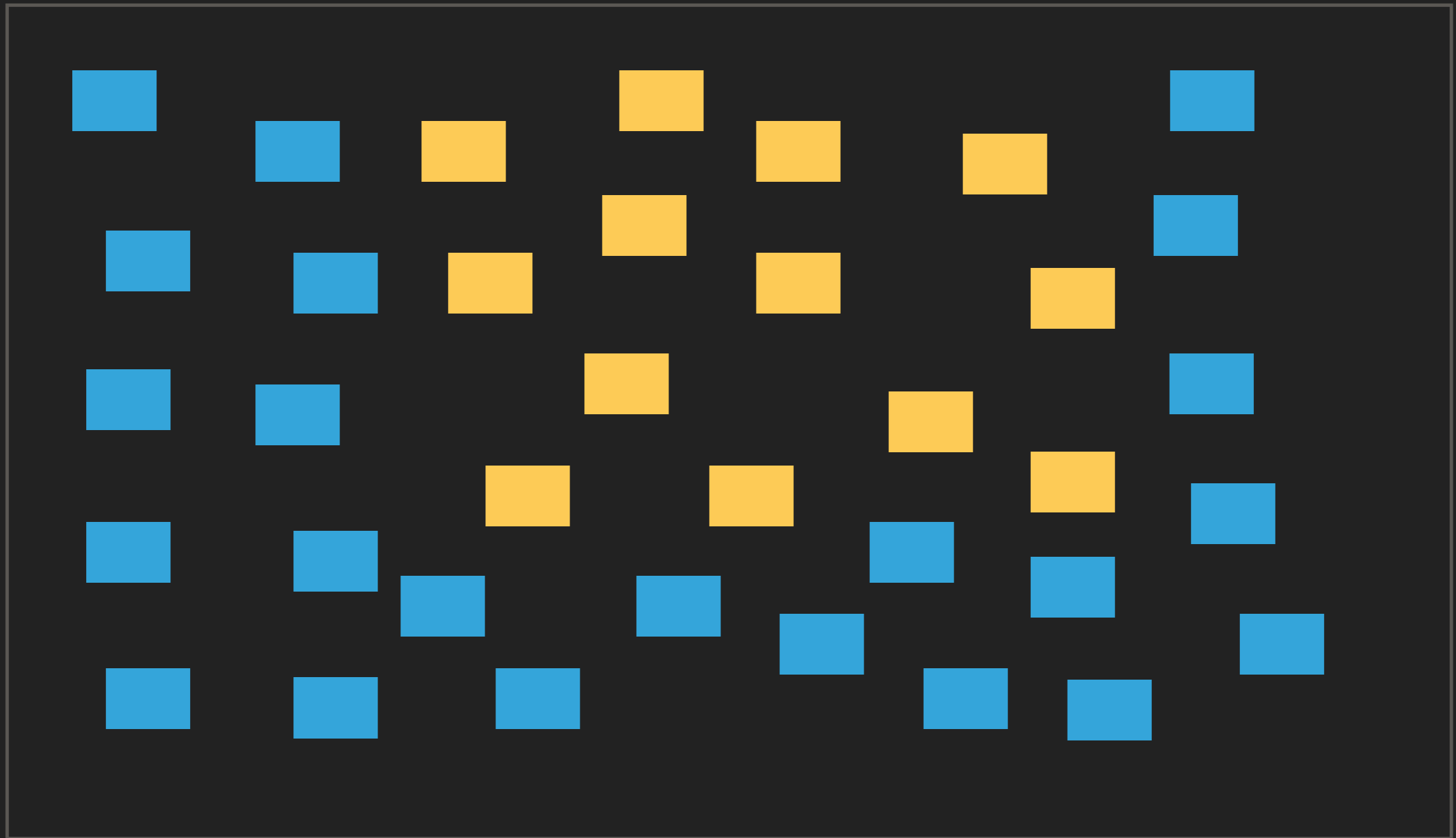
TESTING CONTINUUM



Unit tests

Systems tests

TESTING CONTINUUM



Unit tests

Systems tests

UNIT TEST EXAMPLE – SOFTWARE UNDER TEST

UNIT TEST EXAMPLE - SOFTWARE UNDER TEST

```
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
```

UNIT TEST EXAMPLE - TEST CASES REQUIRED

- ▶ Valid passwords:
 - ▶ "Passw0rd"
- ▶ Invalid passwords:
 - ▶ "Passw0r" - too short (everything else is good)
 - ▶ "Password" - no digit
 - ▶ "passwd" - no upper case letters
 - ▶ "PASSWORD" - no lower case letters

LOOK HOW EASY IT IS TO TEST...

```
class PasswordValidatorTest extends TestCase
{
    public function dataProvider() : array
    {
        return [
            "valid"          => [ true,  "Passw0rd" ],
            "tooShort"       => [ false, "Passw0r" ],
            "noDigit"        => [ false, "Password" ],
            "noUpperCase"    => [ false, "passw0rd" ],
            "noLowerCase"    => [ false, "PASSWORD" ],
        ];
    }

    /**
     * @dataProvider dataProvider
     */
    public function testValidator(bool $expectedResult, string $inputValue)
    {
        $validator = new PasswordValidator();
        $actualResult = $validator->isValid($inputValue);
        $this->assertEquals($expectedResult, $actualResult);
    }
}
```


LOOK HOW EASY IT IS TO TEST...

```
class PasswordValidatorTest extends TestCase
{
    public function dataProvider() : array
    {
        return [
            "valid"          => [ true,  "Passw0rd" ],
            "tooShort"       => [ false, "Passw0r" ],
            "noDigit"        => [ false, "Password" ],
            "noUpperCase"    => [ false, "passw0rd" ],
            "noLowerCase"    => [ false, "PASSWORD" ],
        ];
    }

    /**
     * @dataProvider dataProvider
     */
    public function testValidator(bool $expectedResult, string $inputValue)
    {
        $validator = new PasswordValidator();
        $actualResult = $validator->isValid($inputValue);
        $this->assertEquals($expectedResult, $actualResult);
    }
}
```

LOOK HOW EASY IT IS TO TEST...

```
class PasswordValidatorTest extends TestCase
{
    public function dataProvider() : array
    {
        return [
            "valid"           => [ true, "Passw0rd" ],
            "tooShort"        => [ false, "Passw0r" ],
            "noDigit"         => [ false, "Password" ],
            "noUpperCase"     => [ false, "passw0rd" ],
            "noLowerCase"     => [ false, "PASSWORD" ],
        ];
    }

    /**
     * @dataProvider dataProvider
     */
    public function testValidator(bool $expectedResult, string $inputValue)
    {
        $validator = new PasswordValidator();
        $actualResult = $validator->isValid($inputValue);
        $this->assertEquals($expectedResult, $actualResult);
    }
}
```

**THESE TESTS ARE
FAMOUS FIVE**

UNIT TEST THIS KIND OF LOGIC

- ▶ Unit test sweet spot
- ▶ Quicker to test than not test
- ▶ Learn how to use **data providers** for your test framework

THE IDEAL TEST SUITE

- ▶ Fast
- ▶ High coverage
- ▶ Low maintenance

SPEED OF EXECUTION

SPEED OF EXECUTION

Fast

SYSTEM TEST



SPEED OF EXECUTION

Fast

Slow

COVERAGE

COVERAGE

High

COVERAGE

High

Low

COVERAGE

Low

High

Low

COVERAGE

Low

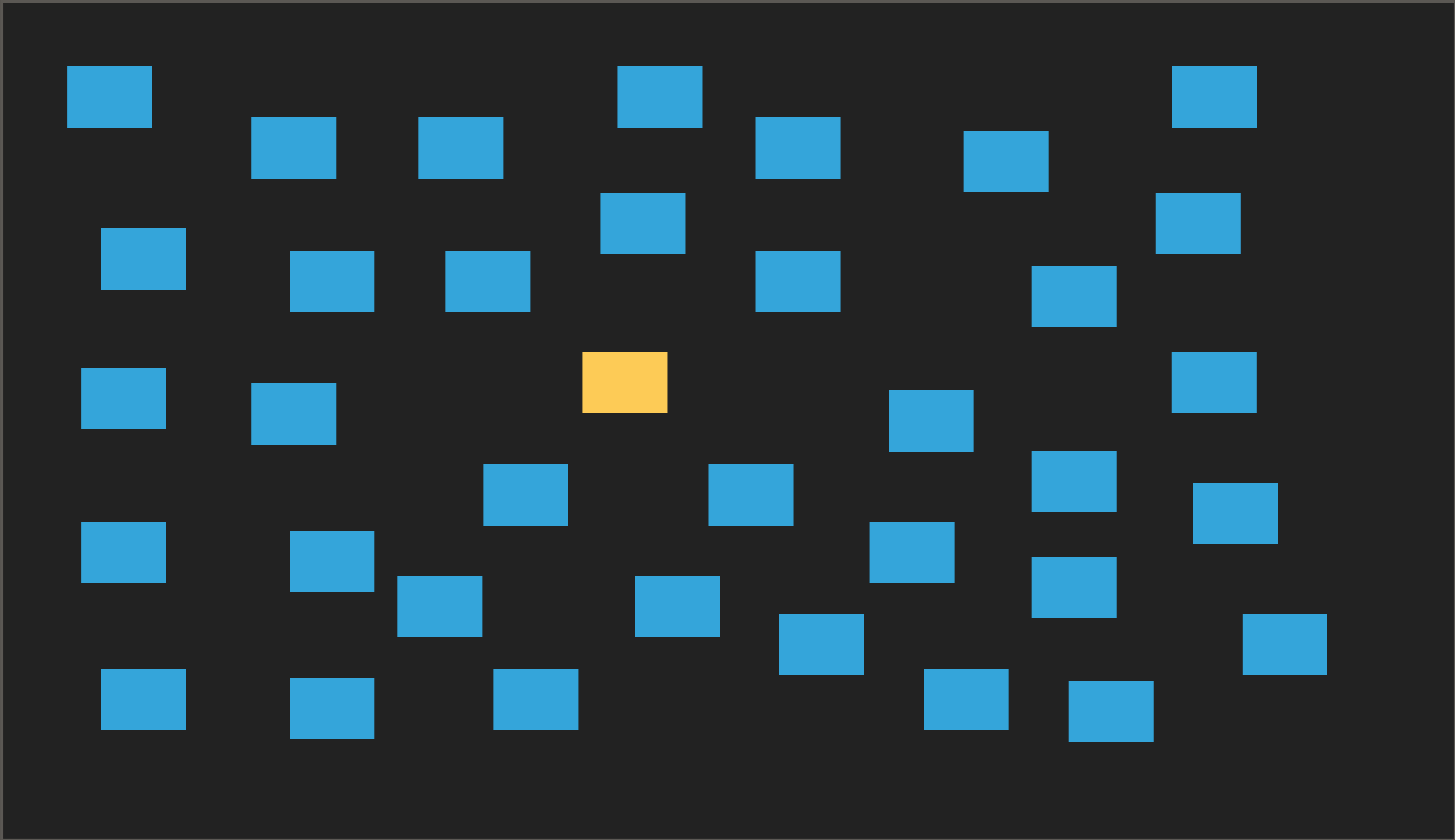
High

High

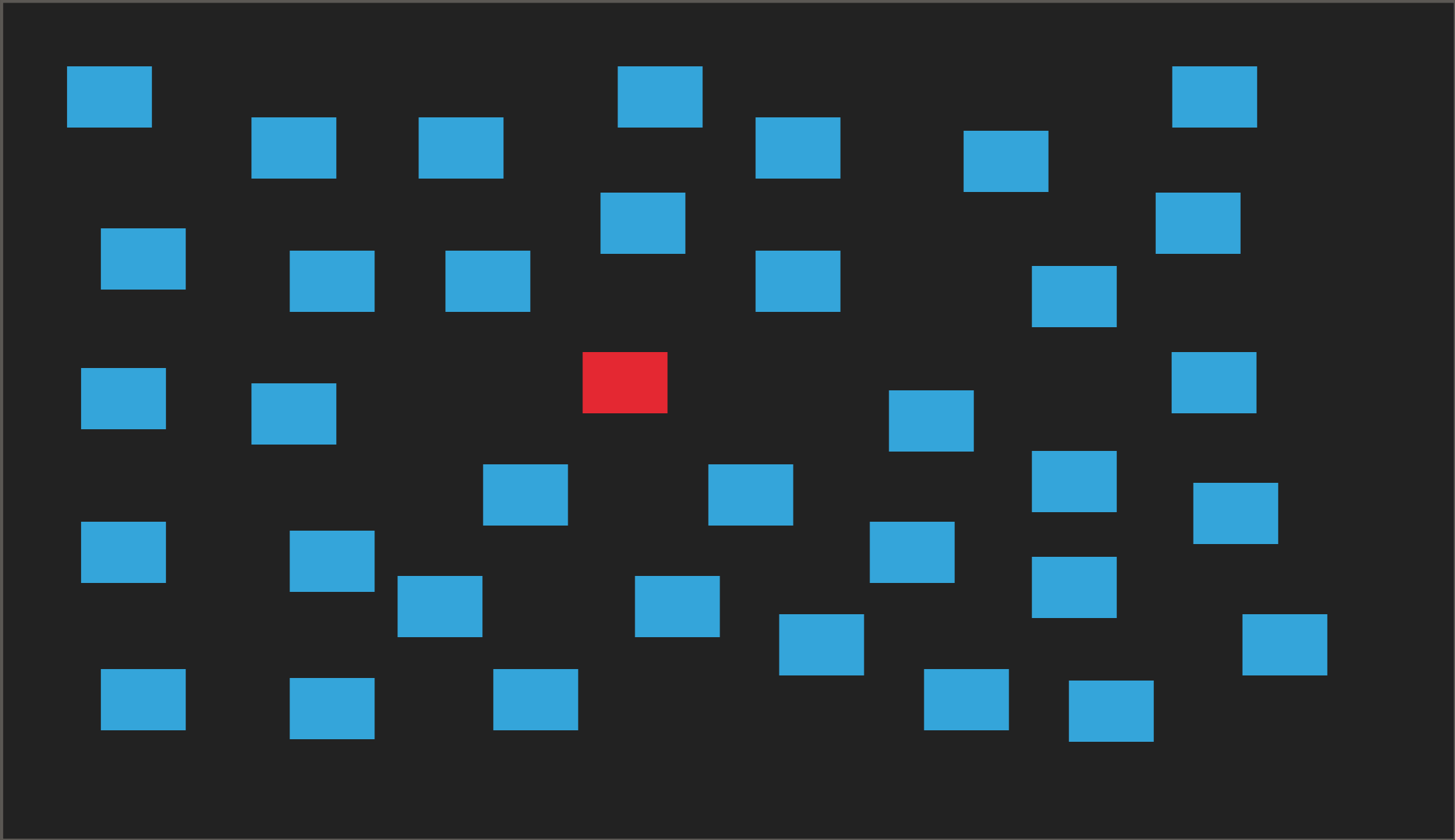
Low

MAINTENANCE COSTS

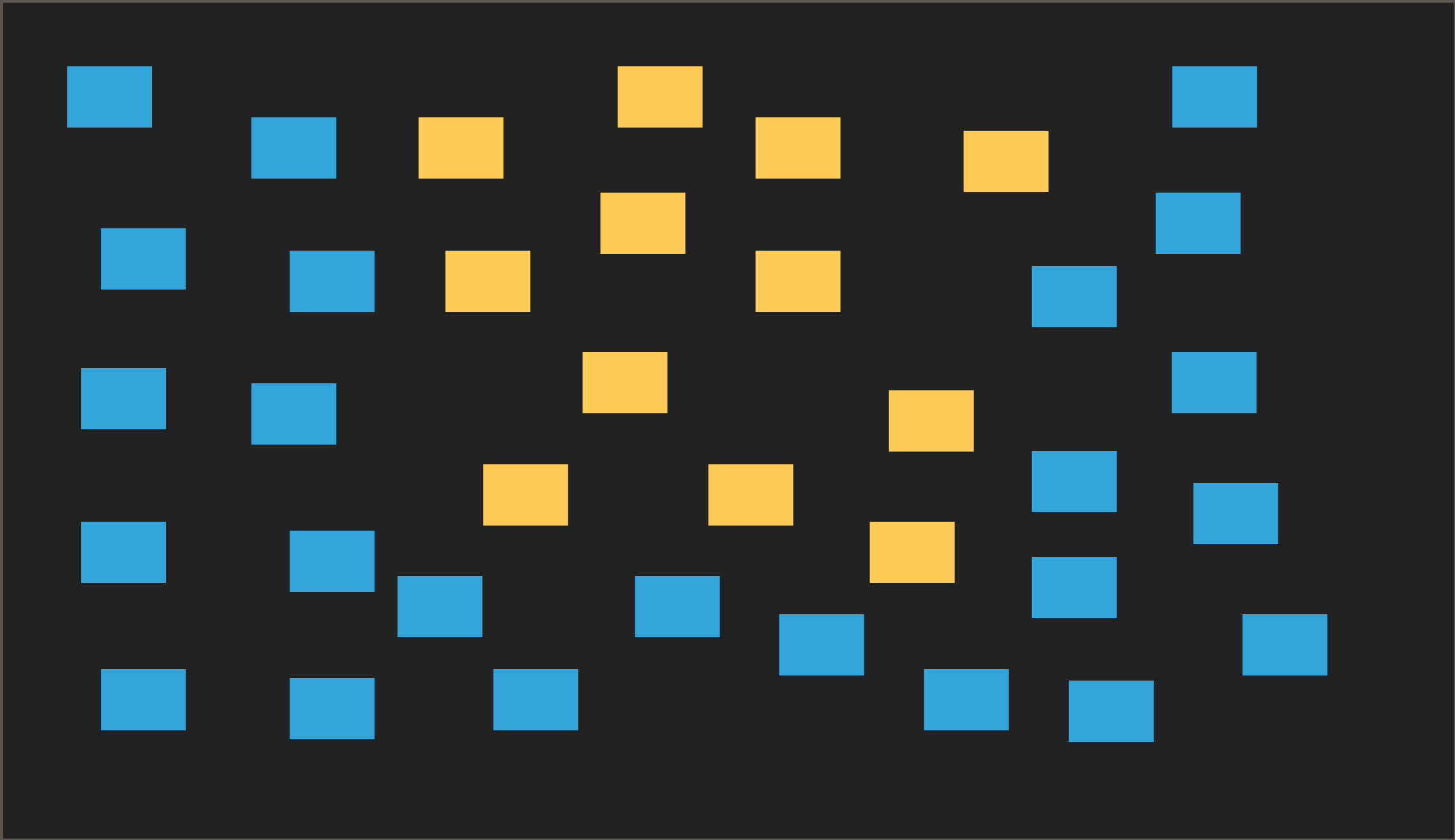
UNIT TEST



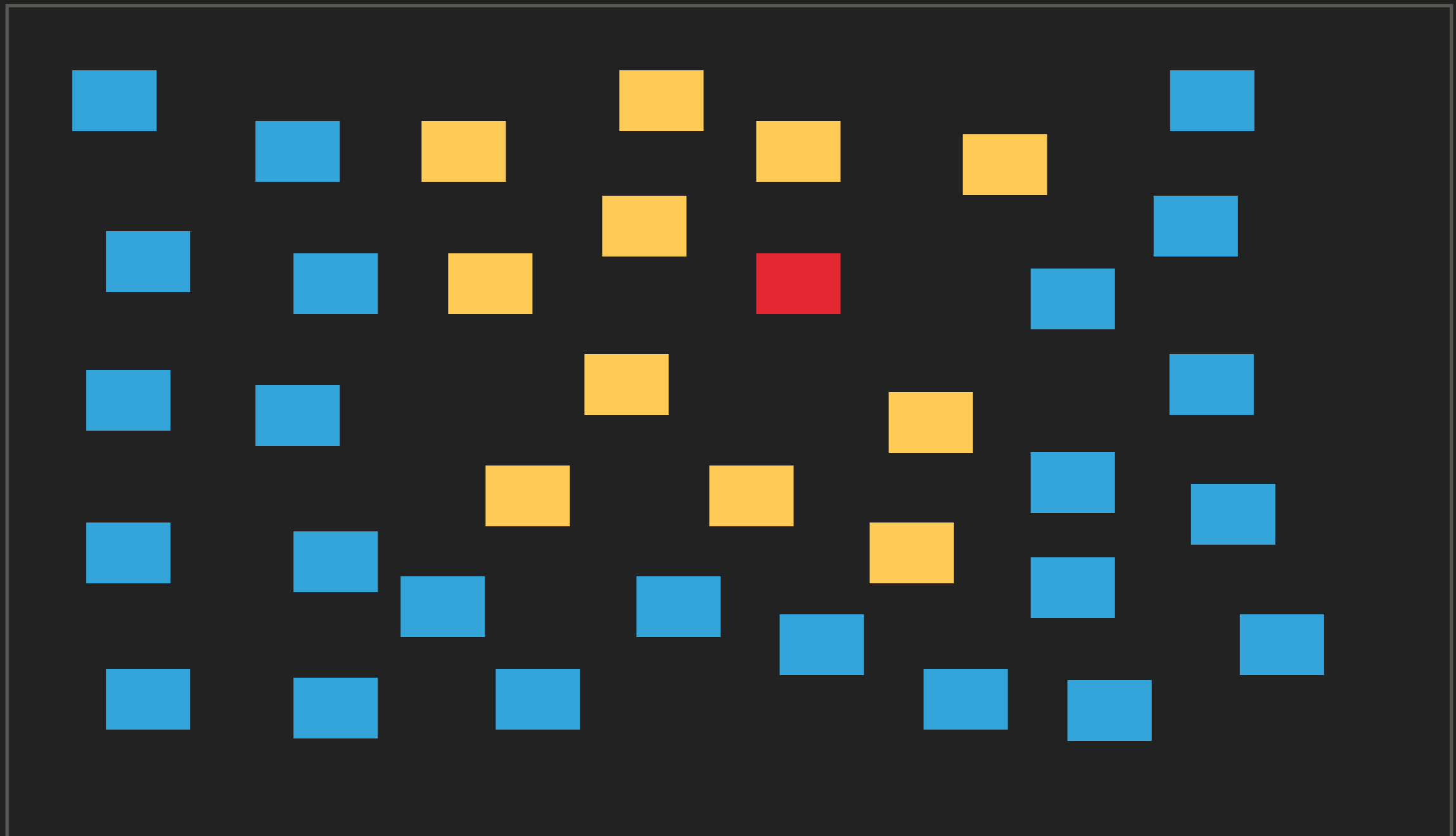
UNIT TEST



INTEGRATION TEST



INTEGRATION TEST



MAINTENANCE COSTS

MAINTENANCE COSTS

Low

MAINTENANCE COSTS

Low

High

THANK GOODNESS FOR THAT MOMENTS

THANK GOODNESS FOR THAT MOMENTS

Low

THANK GOODNESS FOR THAT MOMENTS

Low

High

**SO FAR NOTHING TOO
CONTROVERSIAL**

**NOTHING IS
BLACK AND WHITE**

**EVERYTHING IS
COMPROMISE**

**WRITING A GOOD TEST
SUITE IS A SKILL**

HOW SHOULD WE TEST

WHERE ALONG THE CONTINUUM SHOULD WE TEST?

Unit tests

Systems tests

WHERE ALONG THE CONTINUUM SHOULD WE TEST?



WHERE ALONG THE CONTINUUM SHOULD WE TEST?



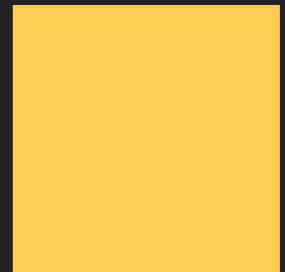
HOW SHOULD WE TEST

WHERE ALONG THE CONTINUUM SHOULD WE TEST?

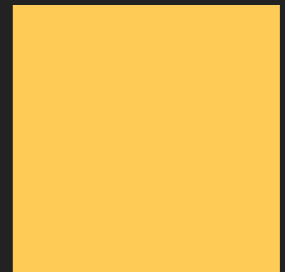
Unit tests

Systems tests

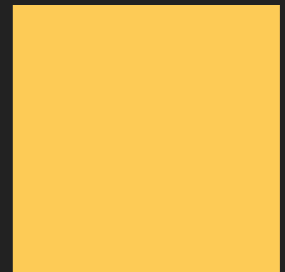
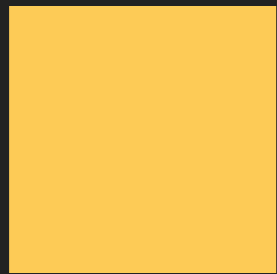
WHERE ALONG THE CONTINUUM SHOULD WE TEST?



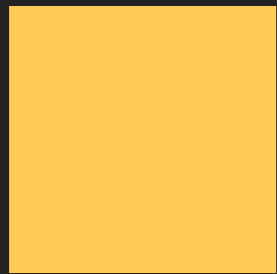
WHERE ALONG THE CONTINUUM SHOULD WE TEST?



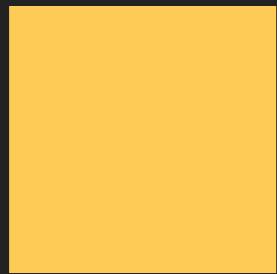
WHERE ALONG THE CONTINUUM SHOULD WE TEST?



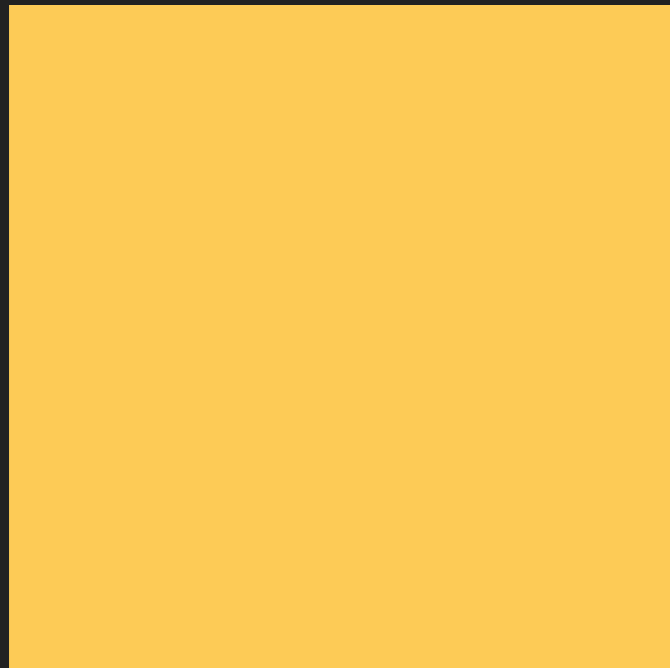
WHERE ALONG THE CONTINUUM SHOULD WE TEST?



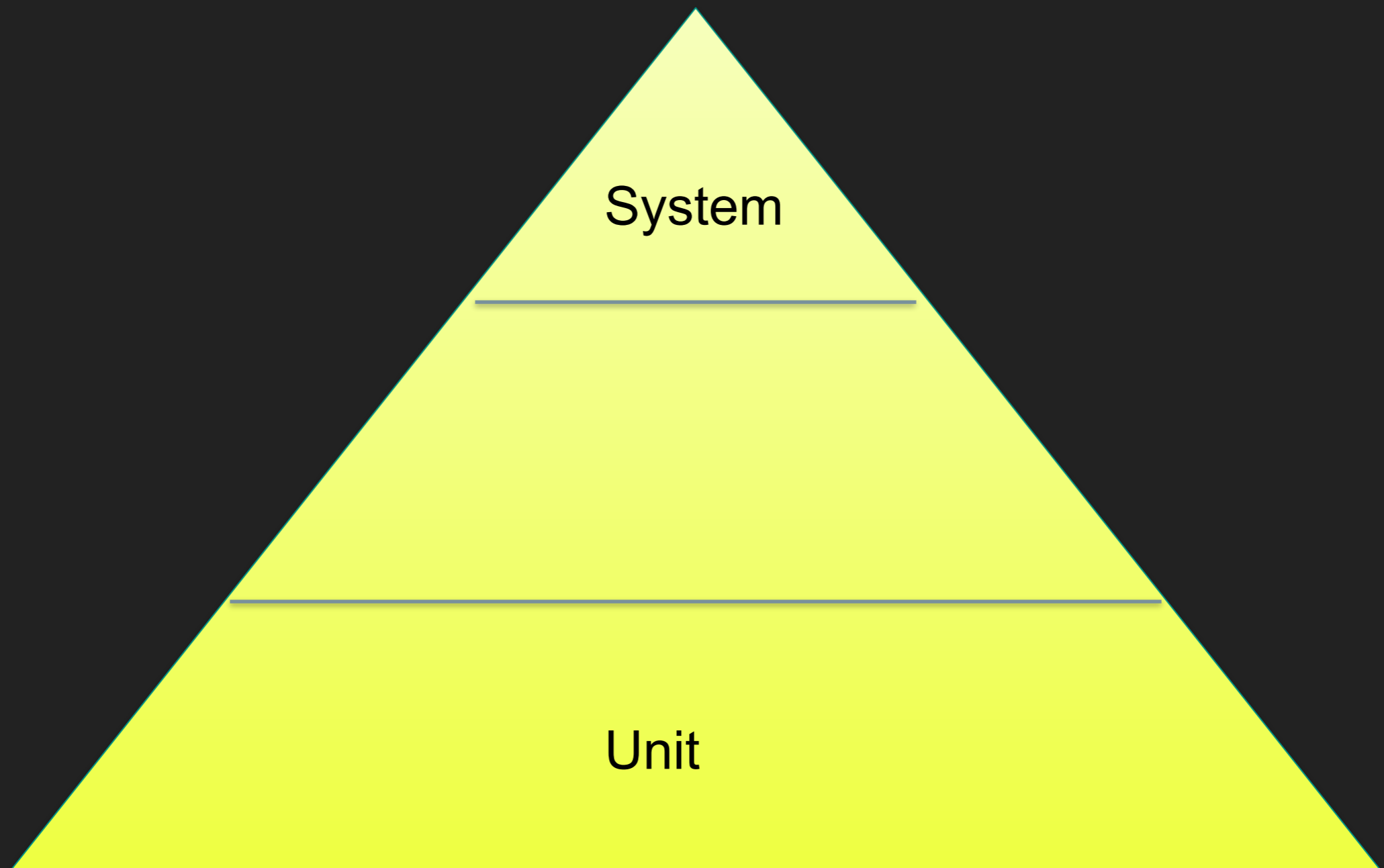
WHERE ALONG THE CONTINUUM SHOULD WE TEST?



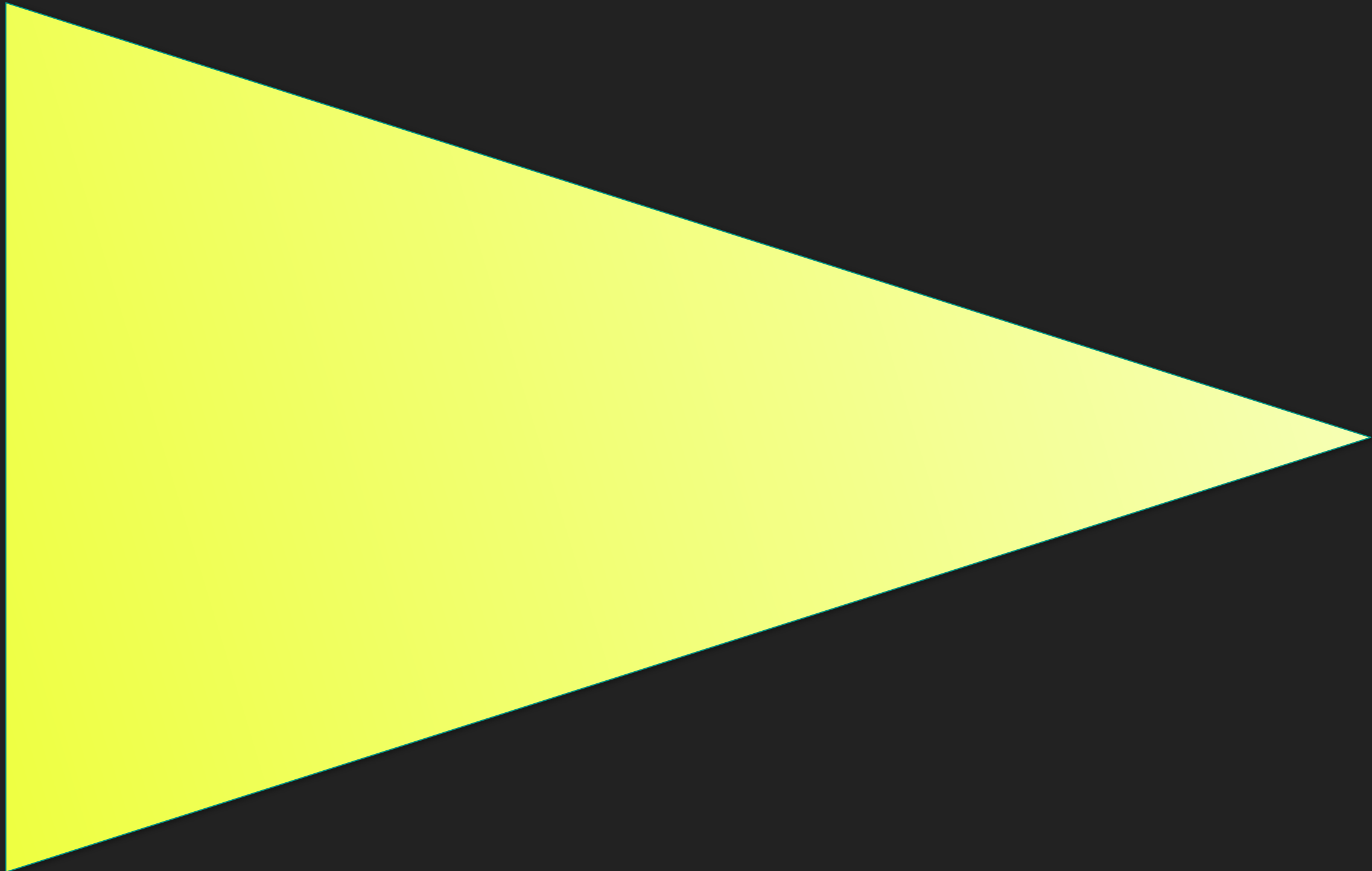
WHERE ALONG THE CONTINUUM SHOULD WE TEST?



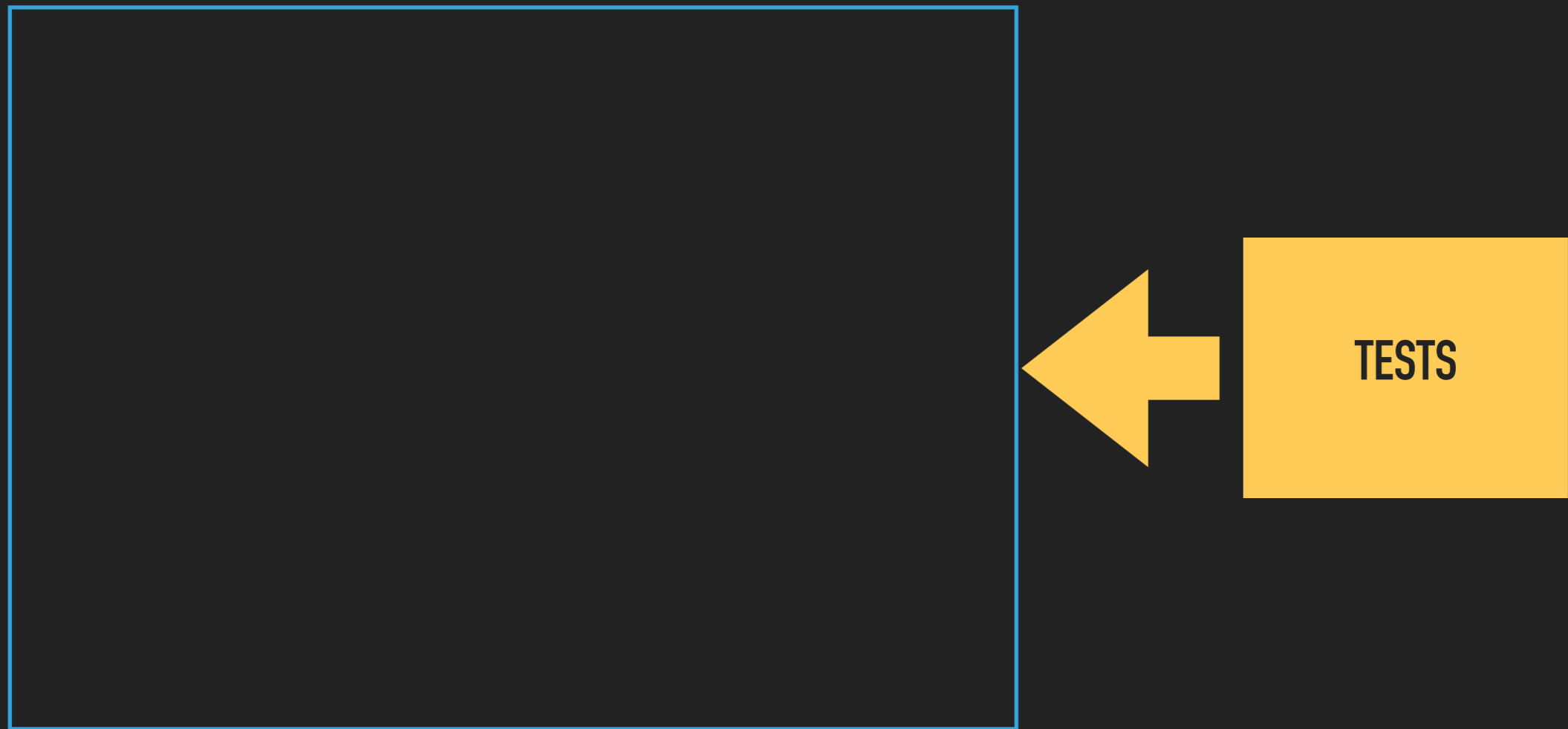
TEST PYRAMID



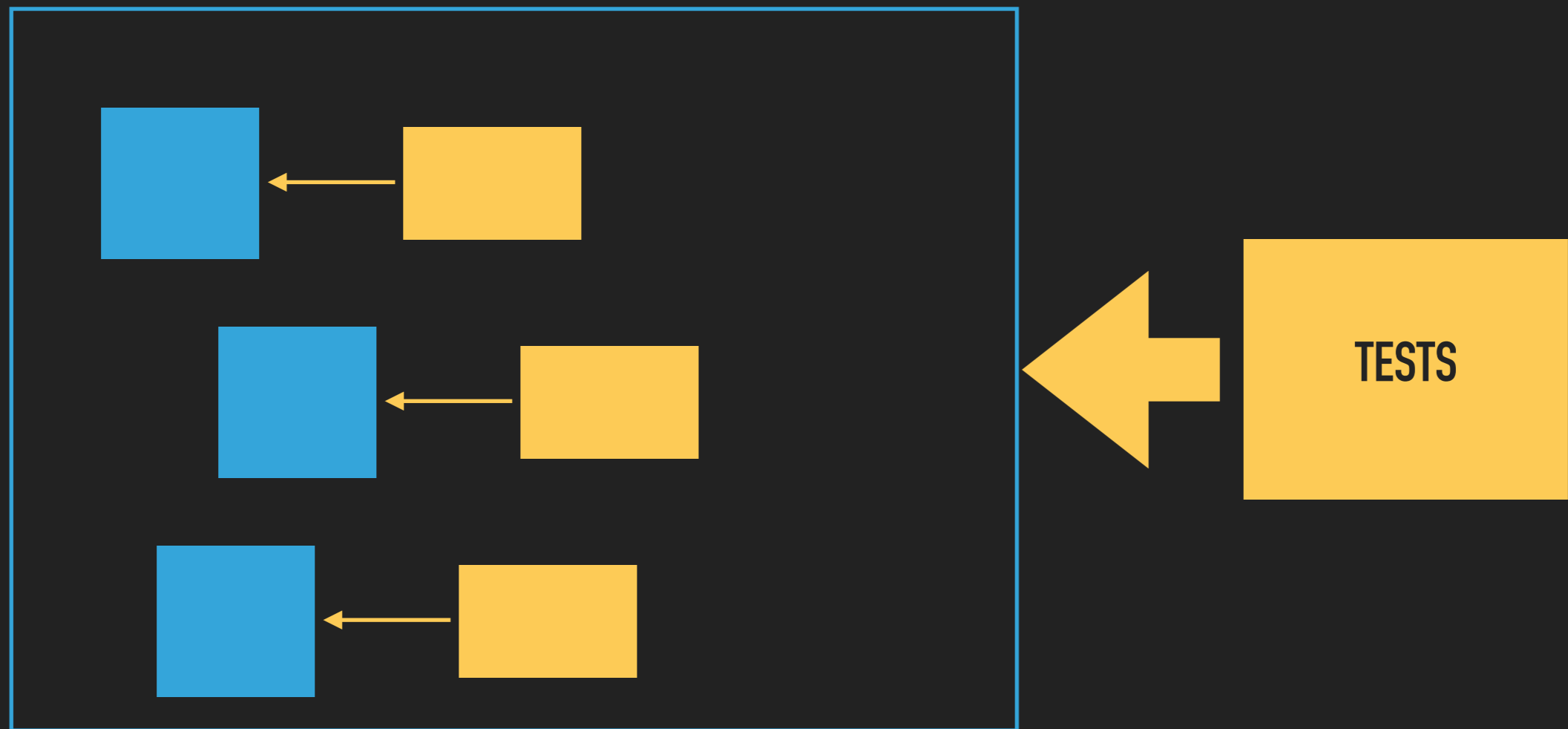
TEST PYRAMID



TEST PYRAMID IS STILL A COMPROMISE



TEST PYRAMID IS STILL A COMPROMISE



TEST PYRAMID IS STILL A COMPROMISE



WHY DO WE NEED A TEST SUITE

- ▶ Prove code works
- ▶ Prevent against regression
- ▶ Allow safe refactoring of code

OUR IDEAL TEST SUITE WOULD BE...

- ▶ Fast to execute
- ▶ High coverage
- ▶ Low maintenance

EVERY THING IS A COMPROMISE

- ▶ Not achievable
 - ▶ Our goals contradict each other
- ▶ Nothing is black and white

UNIT TESTS IN MORE DEPTH

NEW REQUIREMENT

```
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 of the user's previous 5 passwords
     */
    public function isValid(string $password, User $user) : bool
```

**EXISTING
PASSWORD
VALIDATION
RULES**

**EXISTING
PASSWORD
VALIDATION
RULES**

**CHECK IF LAST 5
PASSWORDS**

ARCHITECTURE

**EXISTING
PASSWORD
VALIDATION
RULES**

**CHECK IF LAST 5
PASSWORDS**

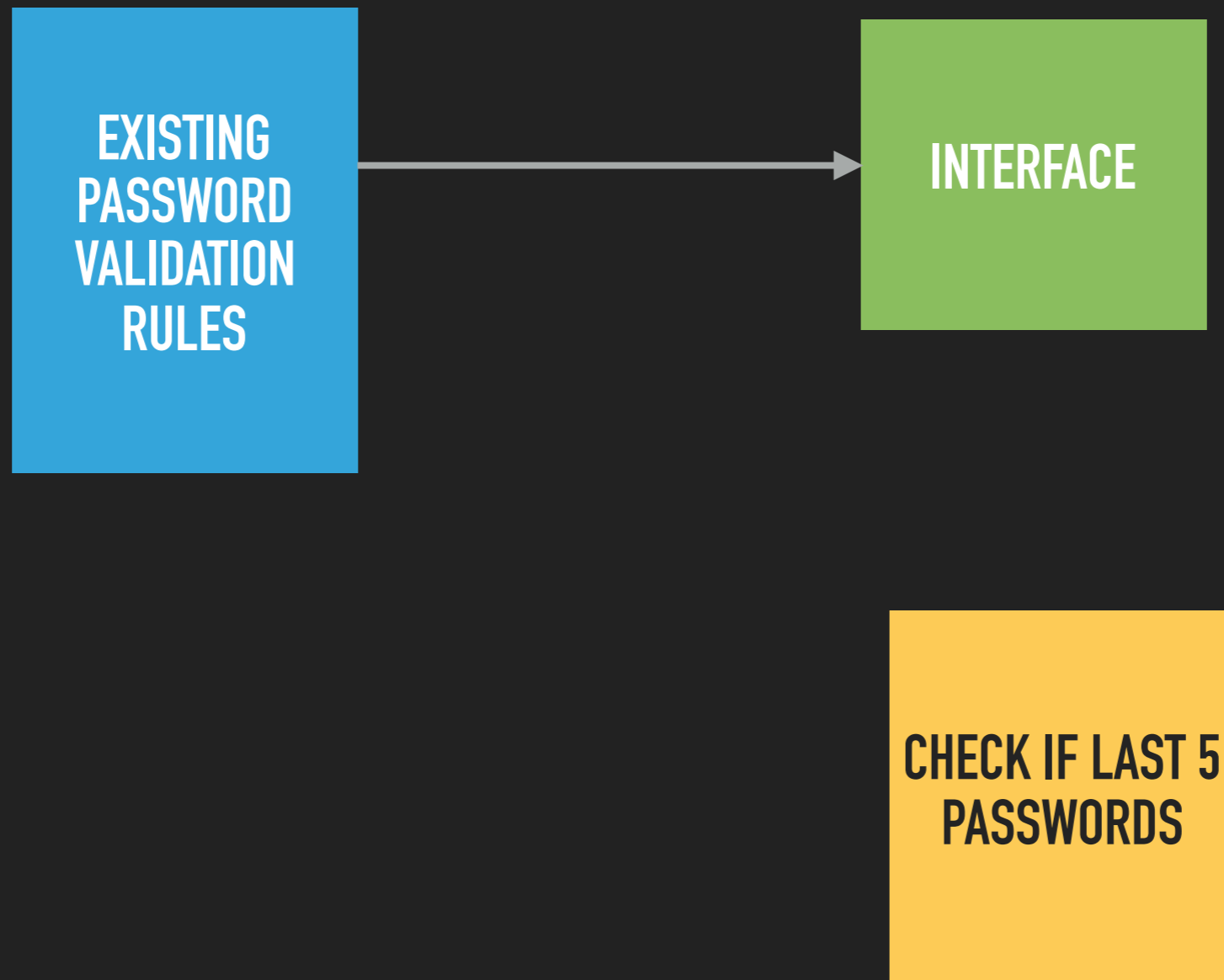
ARCHITECTURE

**EXISTING
PASSWORD
VALIDATION
RULES**

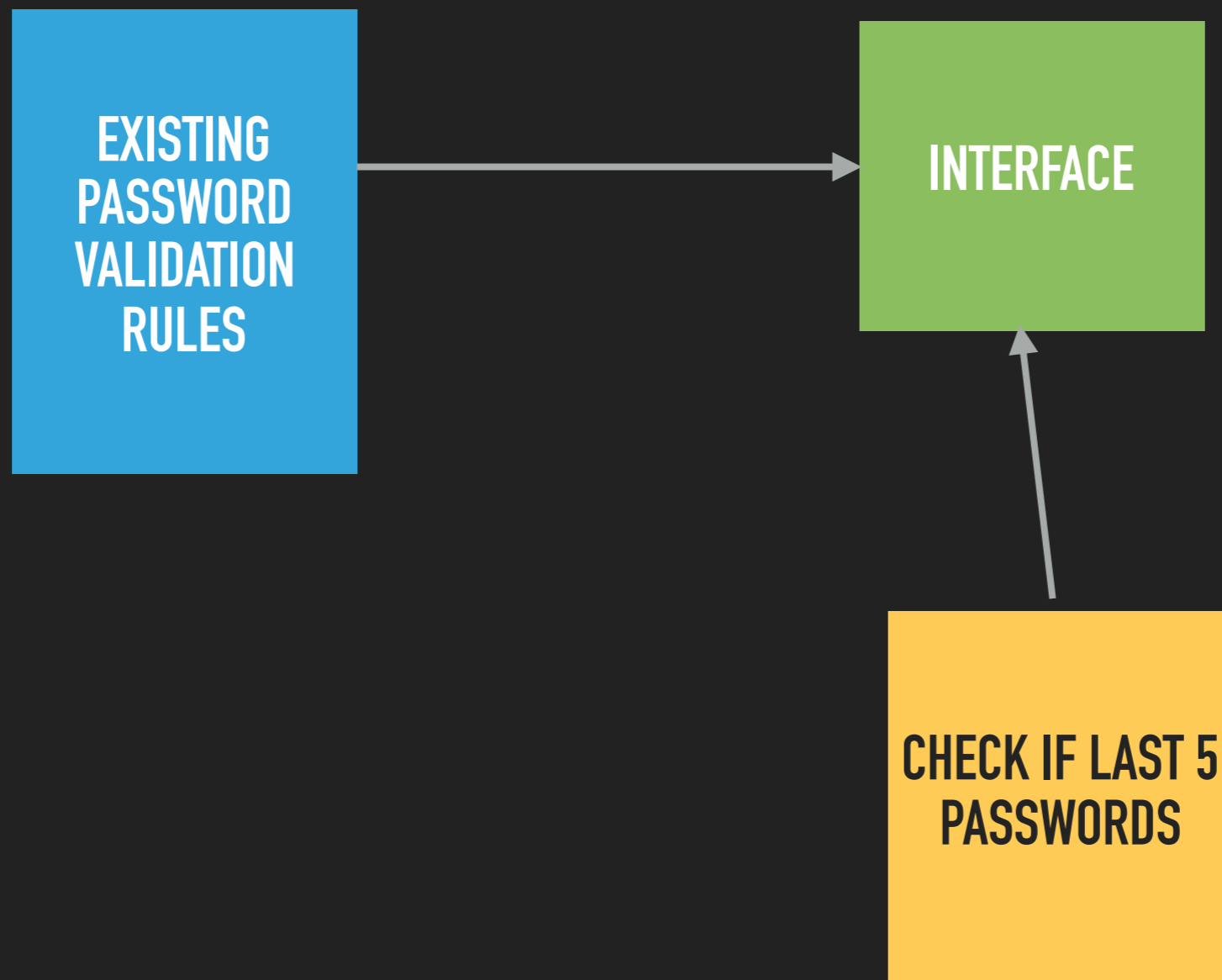
INTERFACE

**CHECK IF LAST 5
PASSWORDS**

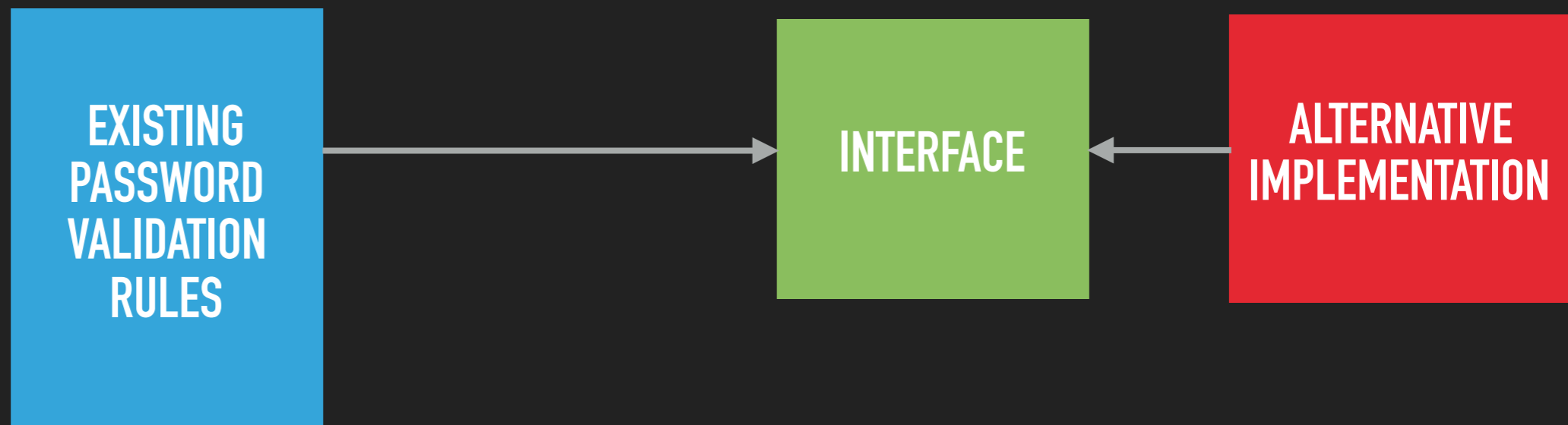
ARCHITECTURE



ARCHITECTURE



ARCHITECTURE



PREVIOUS PASSWORD CHECKER INTERFACE

```
interface PreviousPasswordChecker
{
    /**
     * Returns true if password has been used by user
     * in previous 5 passwords
     */
    public function isPreviouslyUsed($password, $user);
}
```

USE DEPENDENCY INJECTION

```
class PasswordValidator
{
    private $previousPasswordChecker

    public function __construct($previousPasswordChecker) {
        $this->previousPasswordChecker = $previousPasswordChecker;
    }

    public function isValid(string $password) : bool
    {
        ...
    }
}
```

OPTIONS WITH DEPENDENCIES

- ▶ Real thing
- ▶ Test double
 - ▶ Stub
 - ▶ Mock
 - ▶ Fake

ARCHITECTURE

PASSWORD VALIDATOR TEST REVISITED

PASSWORD VALIDATOR TEST REVISITED

- ▶ Update existing tests to account for:
 - ▶ Updated PasswordValidator constructor
 - ▶ Any calls to RecentPasswordChecker

PASSWORD VALIDATOR TEST REVISITED

- ▶ Update existing tests to account for:
 - ▶ Updated PasswordValidator constructor
 - ▶ Any calls to RecentPasswordChecker
- ▶ New tests
 - ▶ Valid password. Has been recently used
 - ▶ Valid password. Has NOT been recently used

NEW TEST: VALID PASSWORD, NOT RECENTLY USED

NEW TEST: VALID PASSWORD, NOT RECENTLY USED

TEST

NEW TEST: VALID PASSWORD, NOT RECENTLY USED

Expect: Exactly 1 call to `isPreviouslyUsed` with parameters `"Passw0rd"` and `$user`. Return false.

TEST

MOCK
RECENT
PASSWORD
CHECKER

NEW TEST: VALID PASSWORD, NOT RECENTLY USED

Expect: Exactly 1 call to `isPreviouslyUsed` with parameters `"Passw0rd"` and `$user`. Return false.

`isValid("Passw0rd", $user)`

TEST

PASSWORD
VALIDATOR

MOCK
RECENT
PASSWORD
CHECKER

NEW TEST: VALID PASSWORD, NOT RECENTLY USED

Expect: Exactly 1 call to `isPreviouslyUsed` with parameters `"Passw0rd"` and `$user`. Return false.

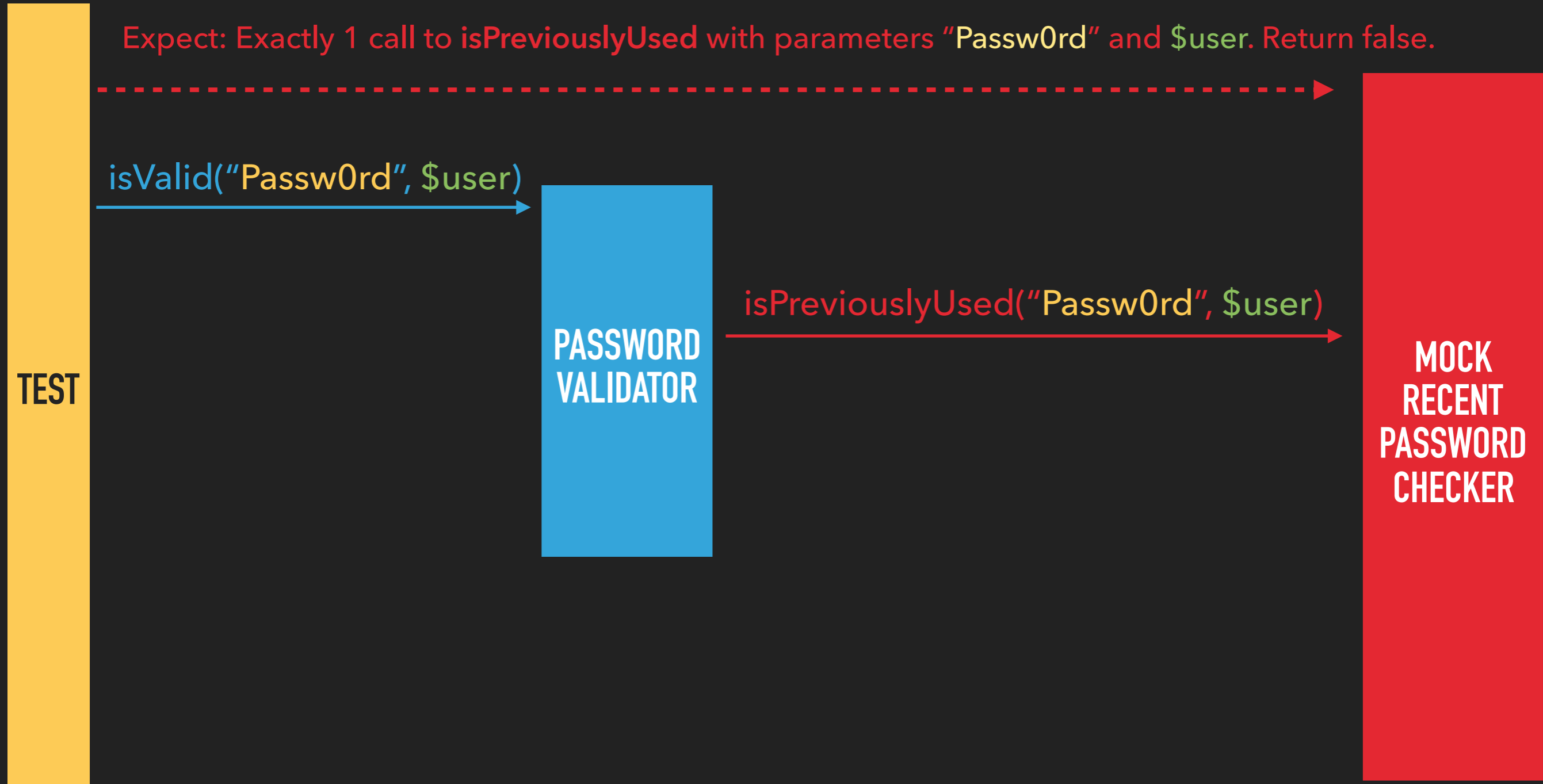
`isValid("Passw0rd", $user)`

PASSWORD
VALIDATOR

`isPreviouslyUsed("Passw0rd", $user)`

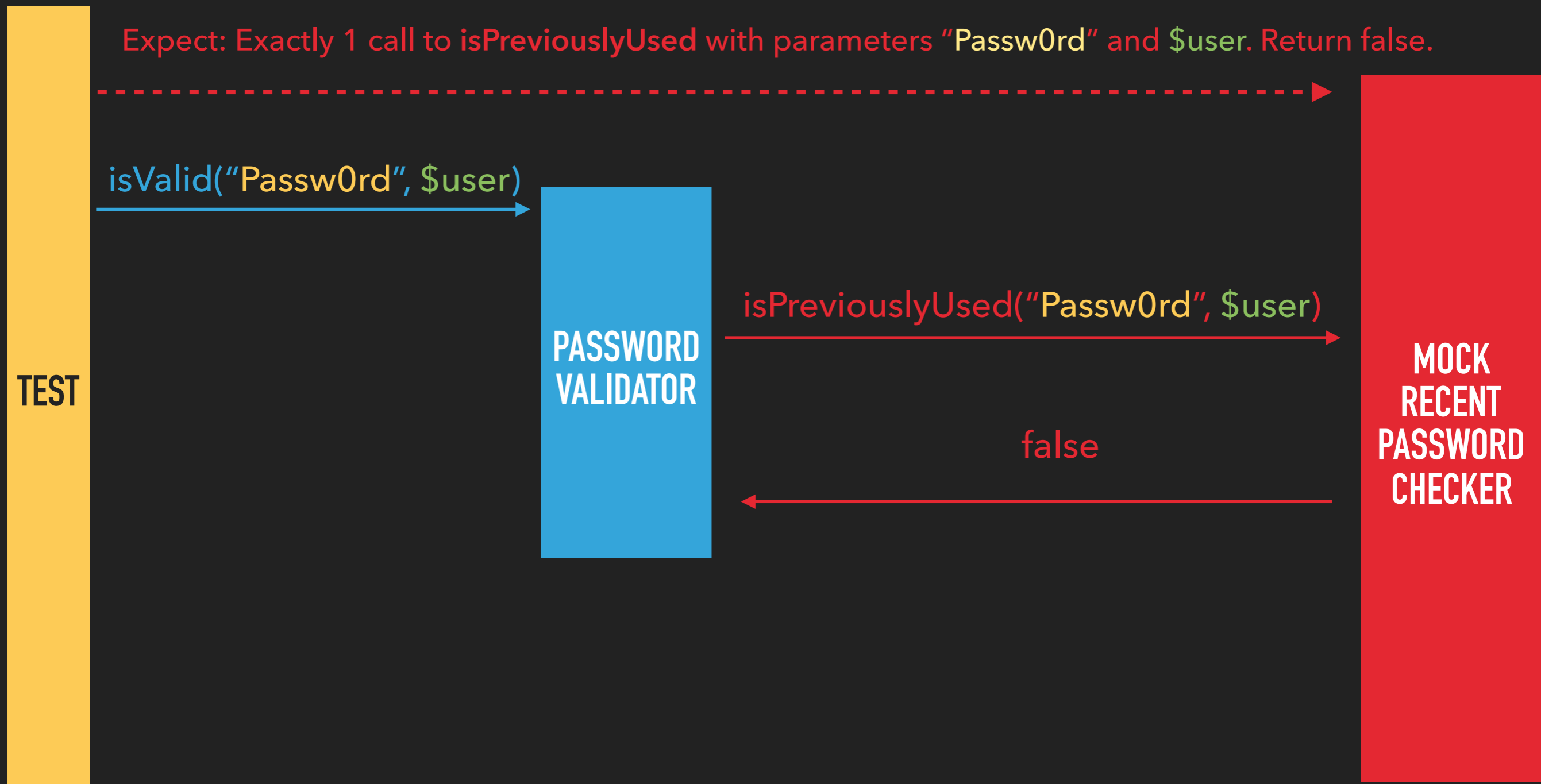
MOCK
RECENT
PASSWORD
CHECKER

TEST



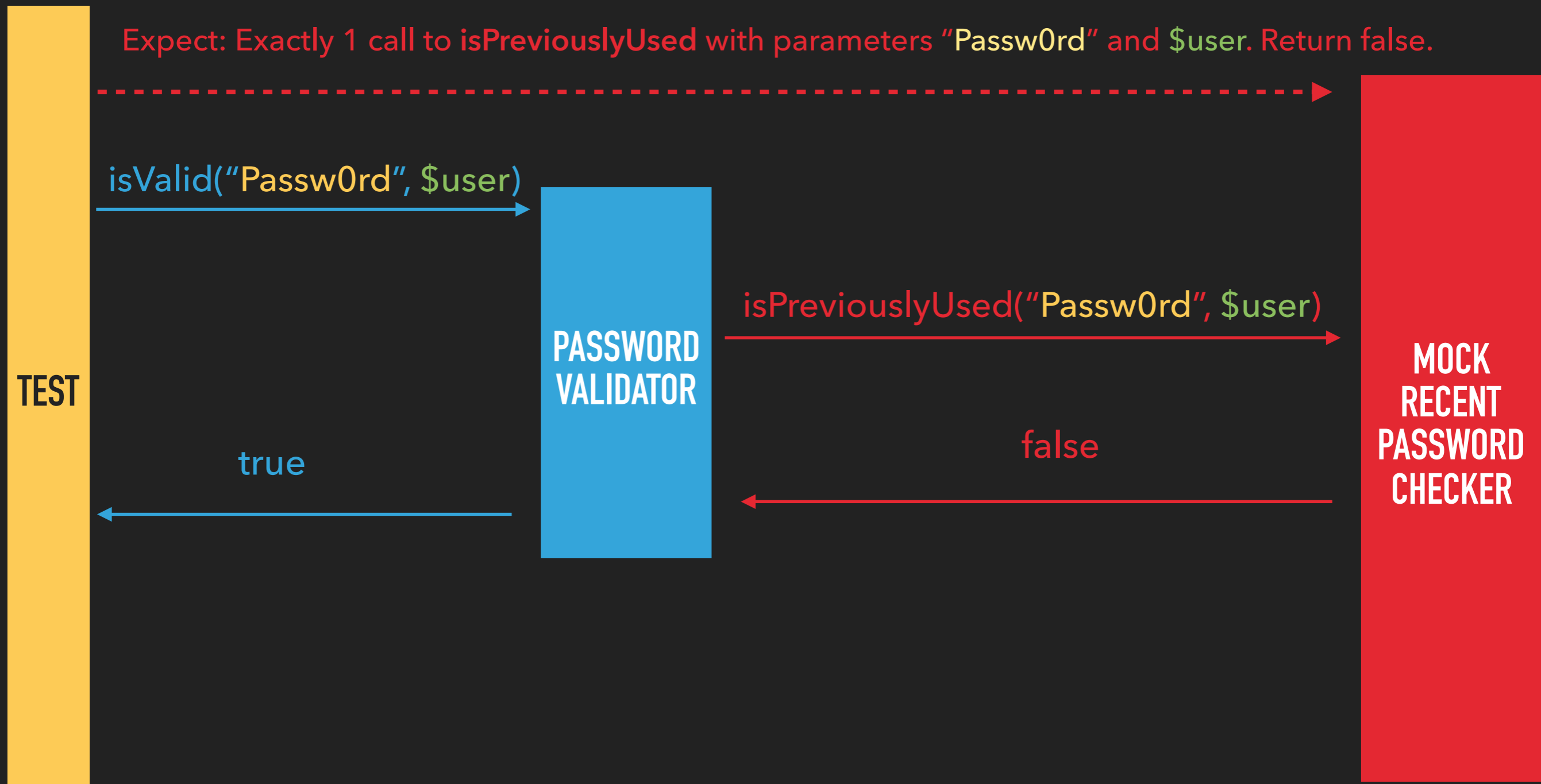
NEW TEST: VALID PASSWORD, NOT RECENTLY USED

Expect: Exactly 1 call to `isPreviouslyUsed` with parameters `"Passw0rd"` and `$user`. Return `false`.



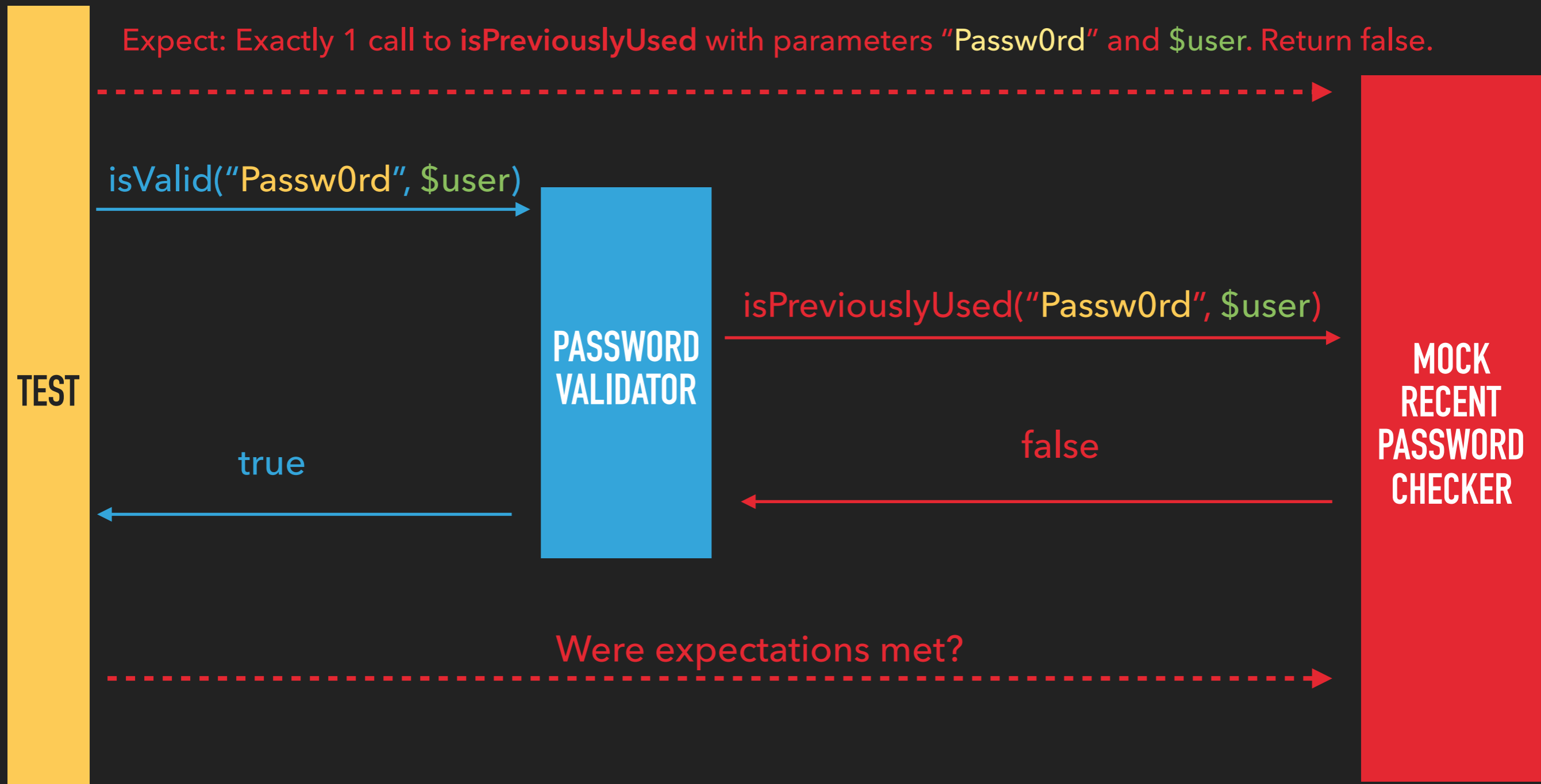
NEW TEST: VALID PASSWORD, NOT RECENTLY USED

Expect: Exactly 1 call to `isPreviouslyUsed` with parameters `"Passw0rd"` and `$user`. Return false.



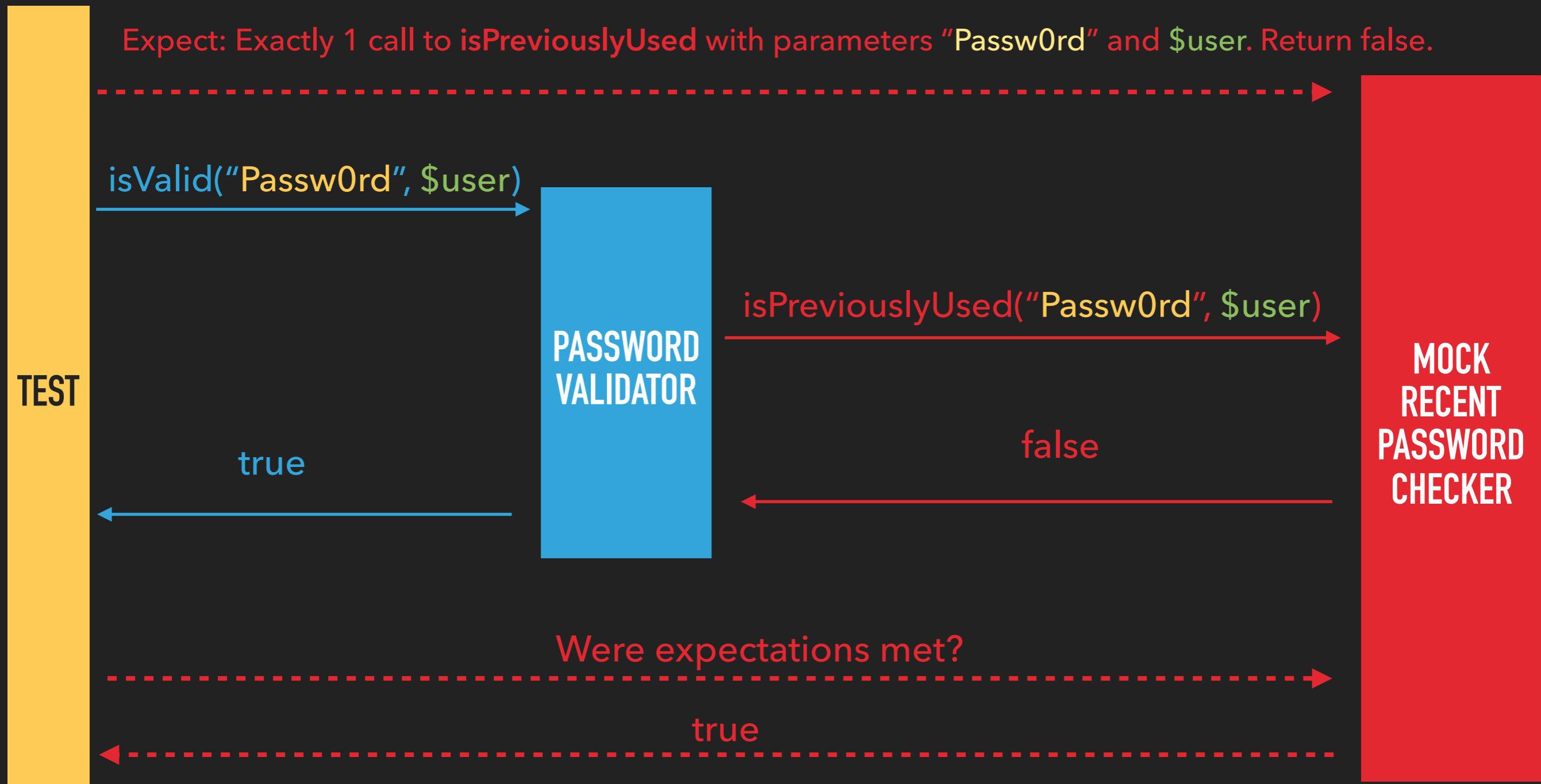
NEW TEST: VALID PASSWORD, NOT RECENTLY USED

Expect: Exactly 1 call to `isPreviouslyUsed` with parameters `"Passw0rd"` and `$user`. Return false.



NEW TEST: VALID PASSWORD, NOT RECENTLY USED

Expect: Exactly 1 call to `isPreviouslyUsed` with parameters `"Passw0rd"` and `$user`. Return `false`.



NEW TEST: VALID PASSWORD, BUT RECENTLY USED

Expect: Exactly 1 call to `isPreviouslyUsed` with parameters "Passw0rd" and \$user. Return **true**.

`isValid("Passw0rd", $user)`

`isPreviouslyUsed("Passw0rd", $user)`

false

true

Were expectations met?

true

TEST

PASSWORD
VALIDATOR

MOCK
RECENT
PASSWORD
CHECKER

**THESE EXTRA 2 TESTS
ARE THE AWKWARD DUO**

MOCKS

EXISTING TESTS (FAMOUS FIVE)

EXISTING TESTS (FAMOUS FIVE)

```
class PasswordValidator
{
    public function isValid(string $password, User $user) : bool
    {
        if ($this->recentPasswordChecker->isRecentPassword(
            $password, $user)) {
            return false;
        }

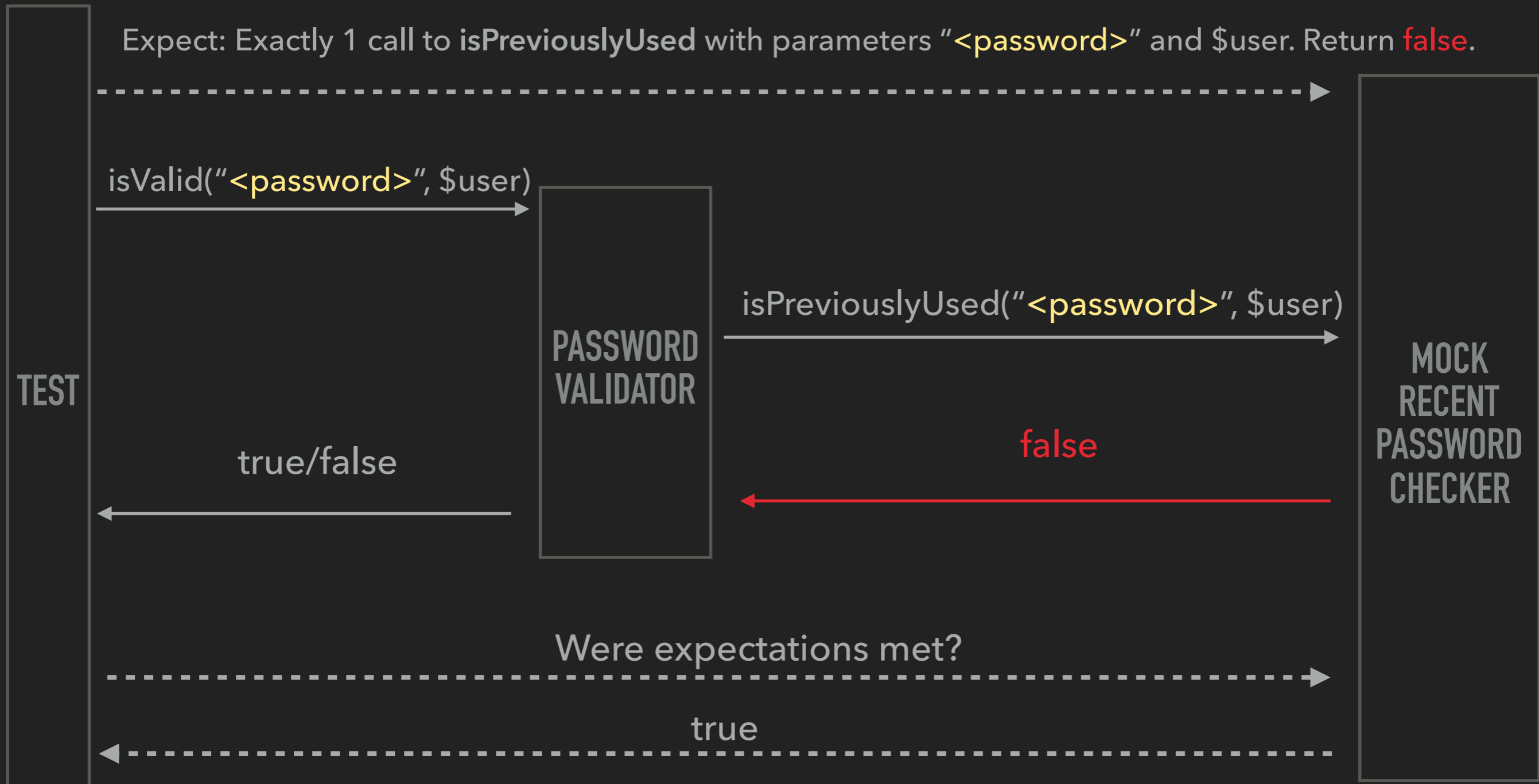
        if (... password too short ...) return false;
        if (... password has no digit ...) return false;

        ... remaining checks ...

        return true;
    }
}
```

EXISTING TESTS

Expect: Exactly 1 call to isPreviouslyUsed with parameters "<password>" and \$user. Return **false**.



EXISTING TESTS - REFACTOR CODE

```
class PasswordValidator
{
    public function isValid(string $password, User $user) : bool
    {
        if (... password too short ...) return false;
        if (... password has no digit ...) return false;

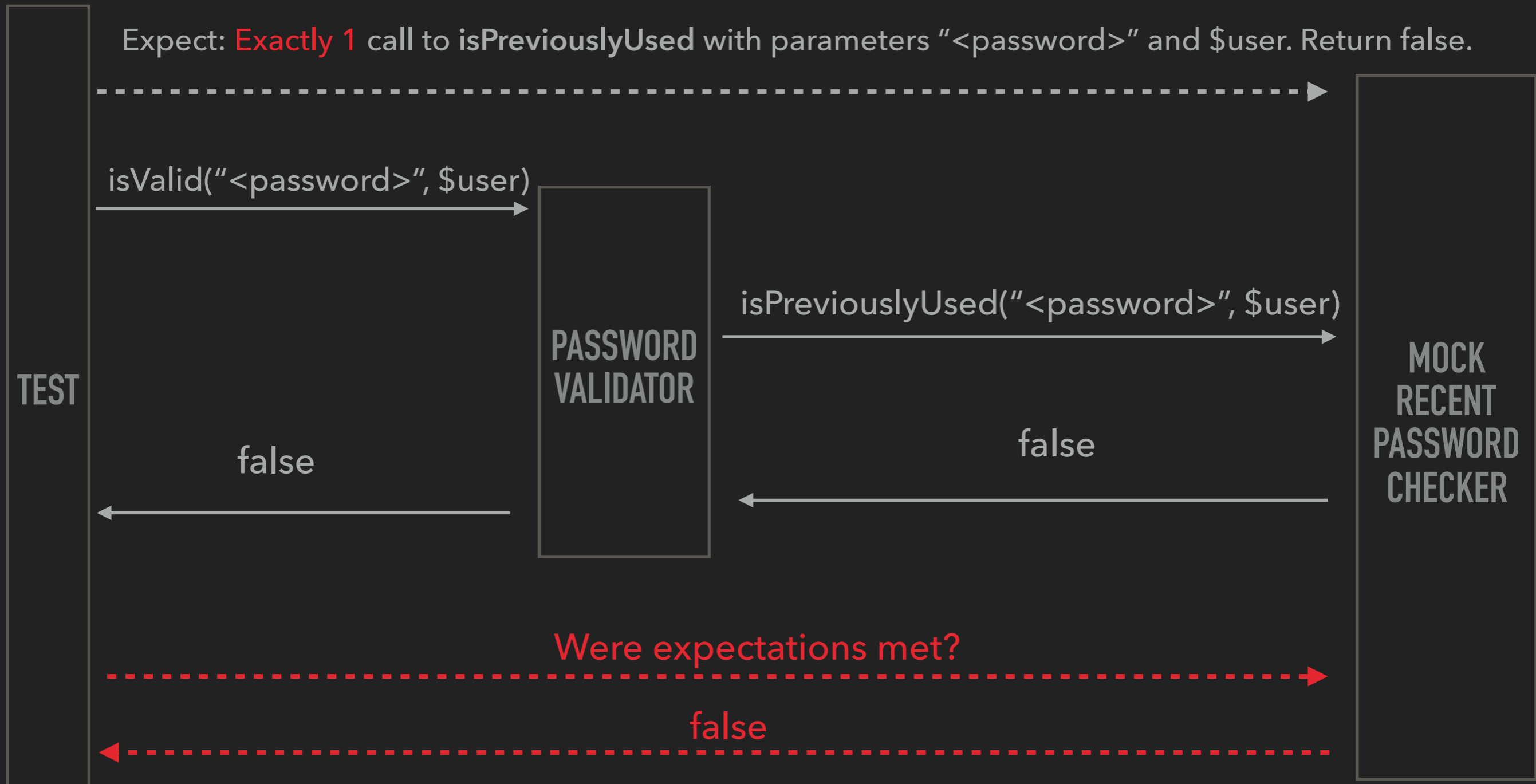
        ... remaining checks ...

        if ($this->recentPasswordChecker->isRecentPassword(
            $password, $user)) {
            return false;
        }

        return true;
    }
}
```

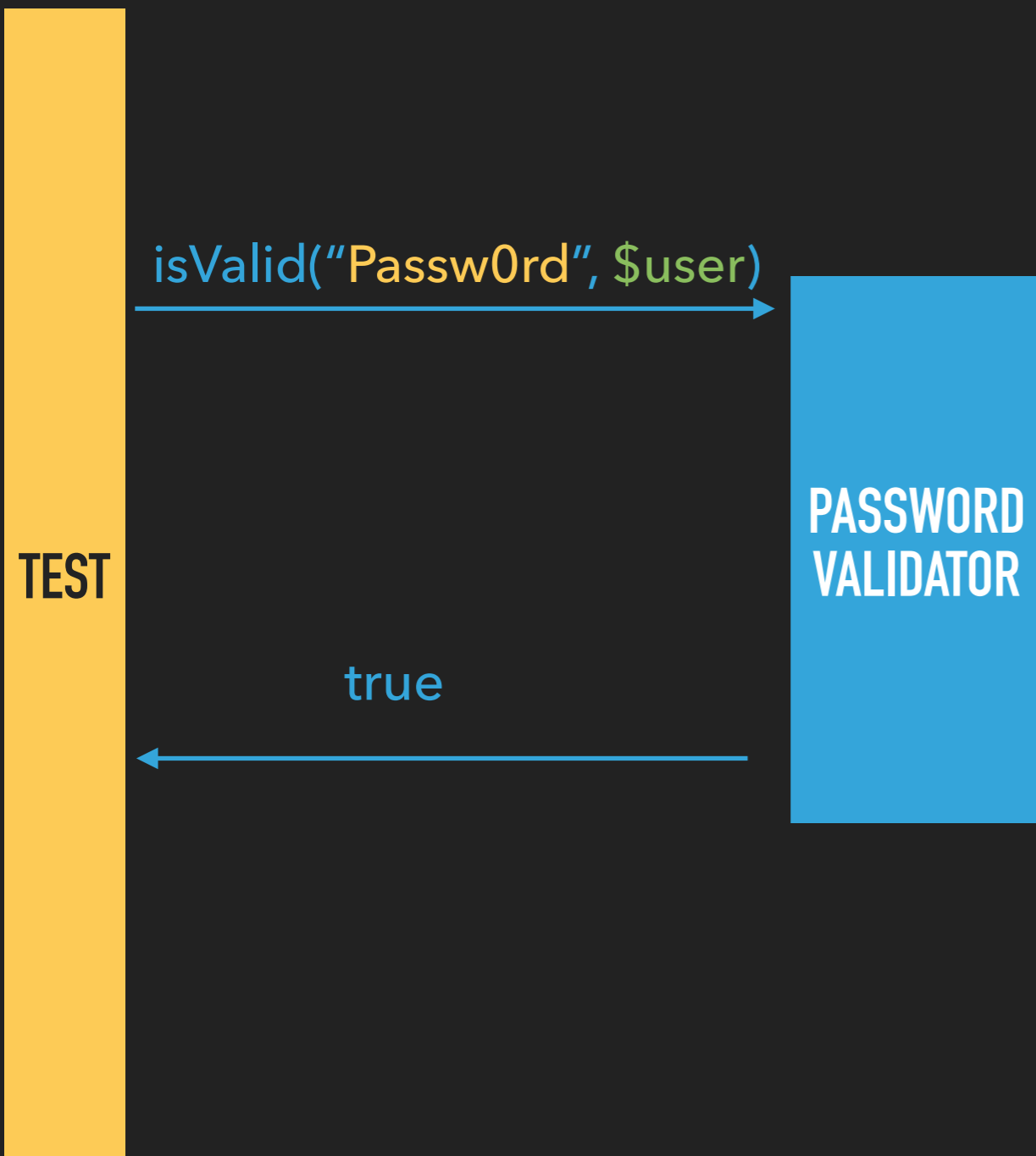
EXISTING TESTS: AFTER REFACTOR

Expect: **Exactly 1** call to `isPreviouslyUsed` with parameters "`<password>`" and `$user`. Return false.

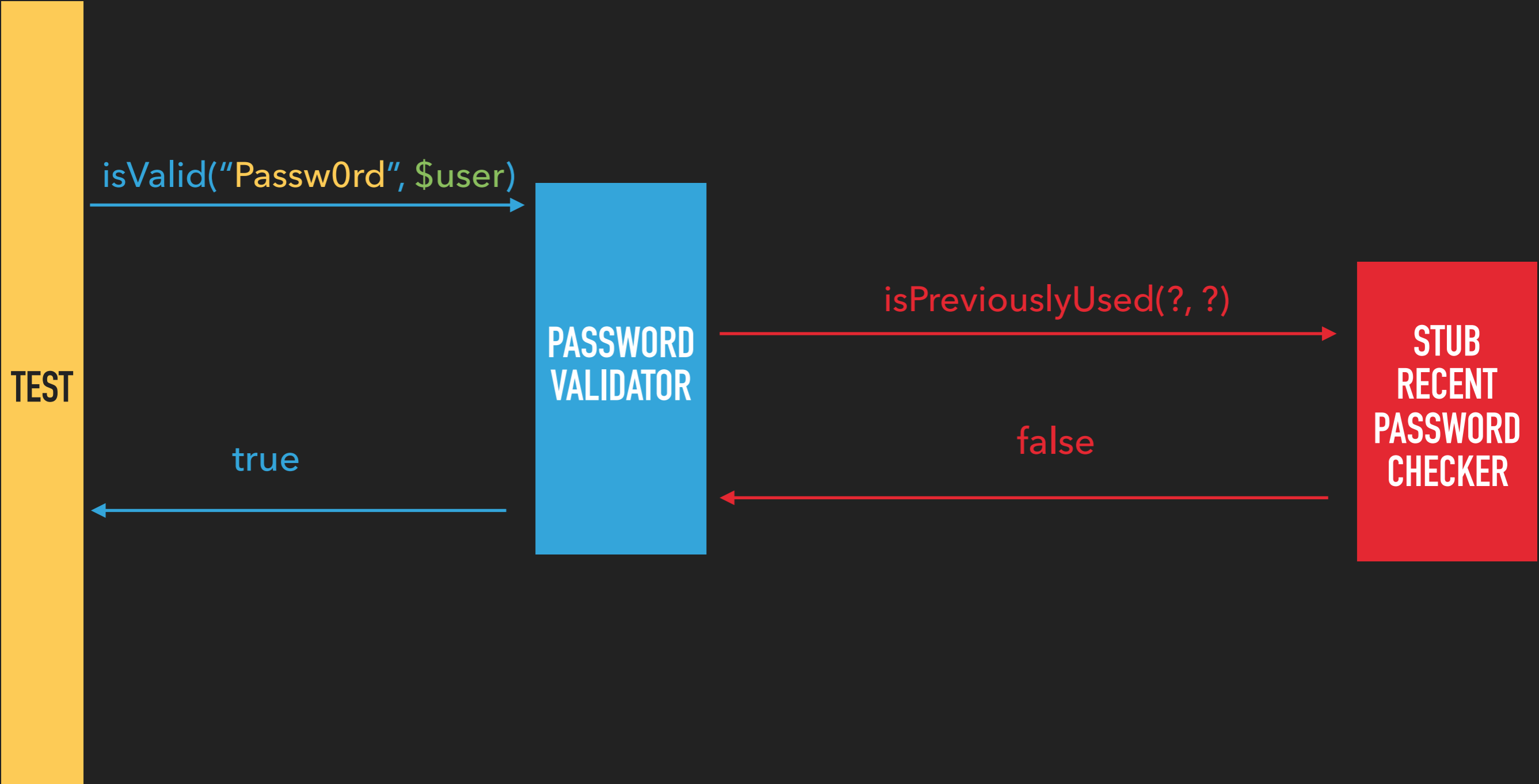


**WE'VE REFACTORED CODE
AND THE TESTS HAVE
BROKEN. NOT GOOD!**

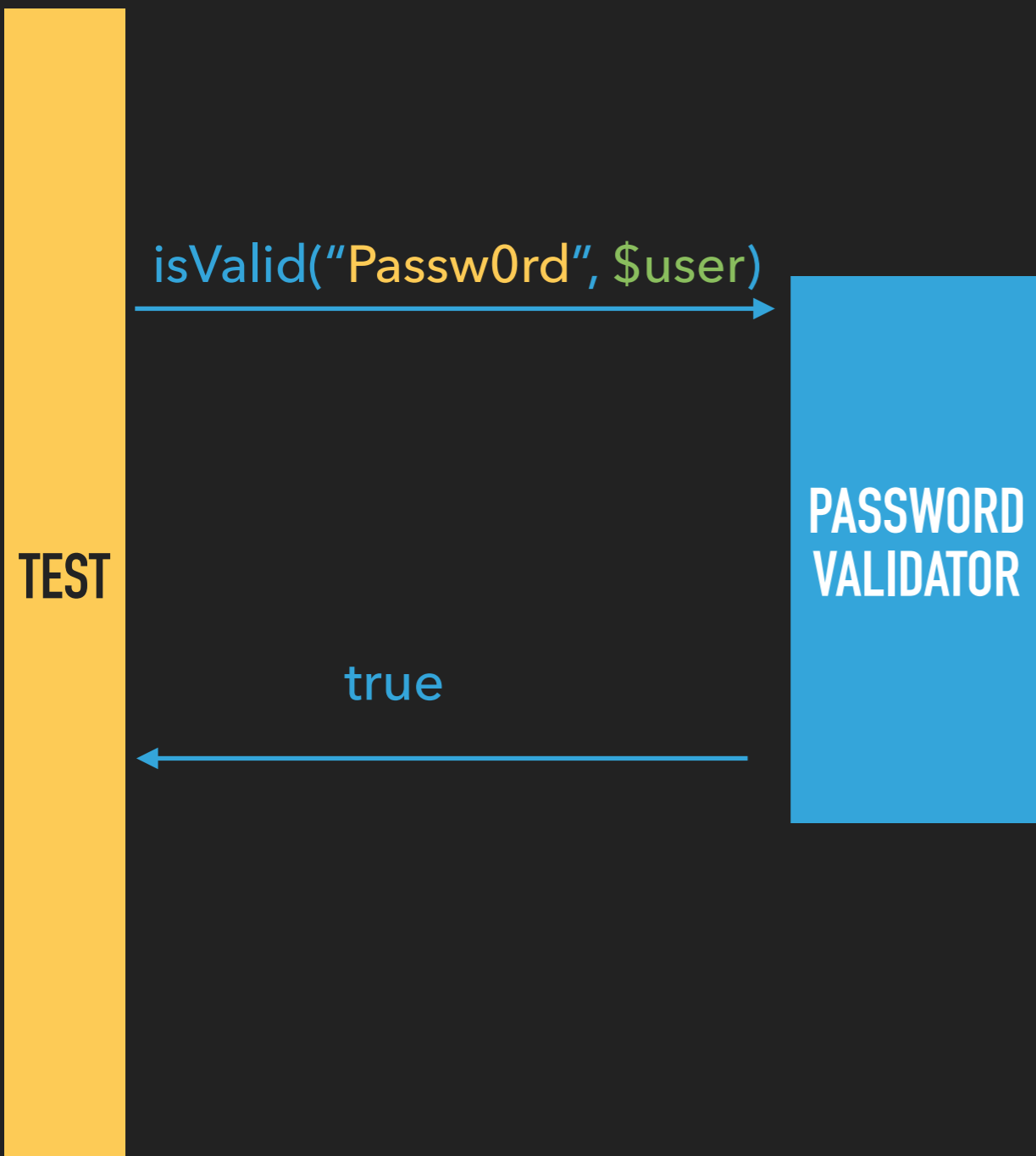
USE A STUB



USE A STUB



USE A STUB



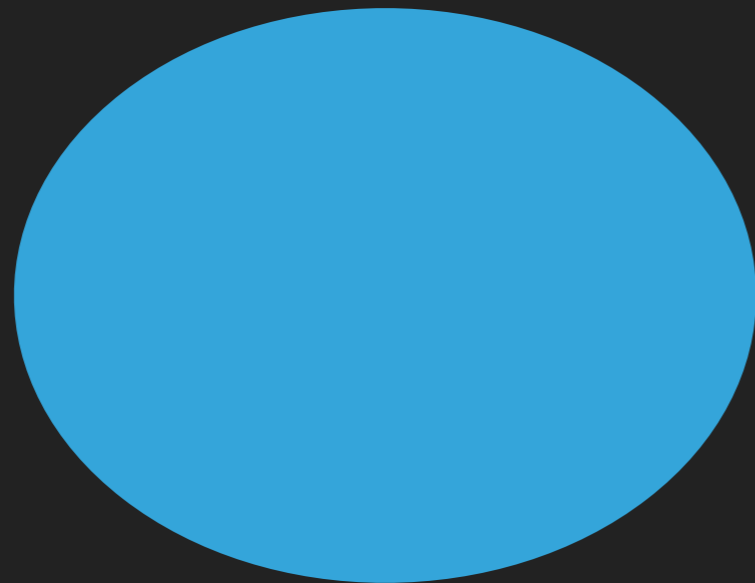
HAND CODE STUB?

```
StubPasswordChecker implements PreviousPasswordChecker
{
    public function isPreviouslyUsed(
        string $password, User $user) : bool {
        return false;
    }
}
```

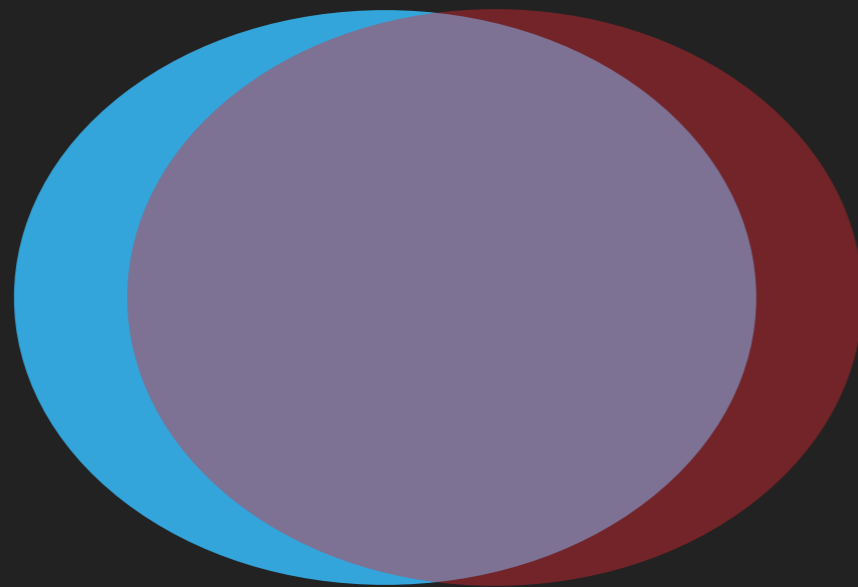
USE STUBS UNLESS YOU REALLY NEED MOCKS

- ▶ Limit the coupling between tests and the code
- ▶ Unnecessary coupling increases maintenance cost
 - ▶ tests harder to write in the first place
 - ▶ reduces ability to refactor

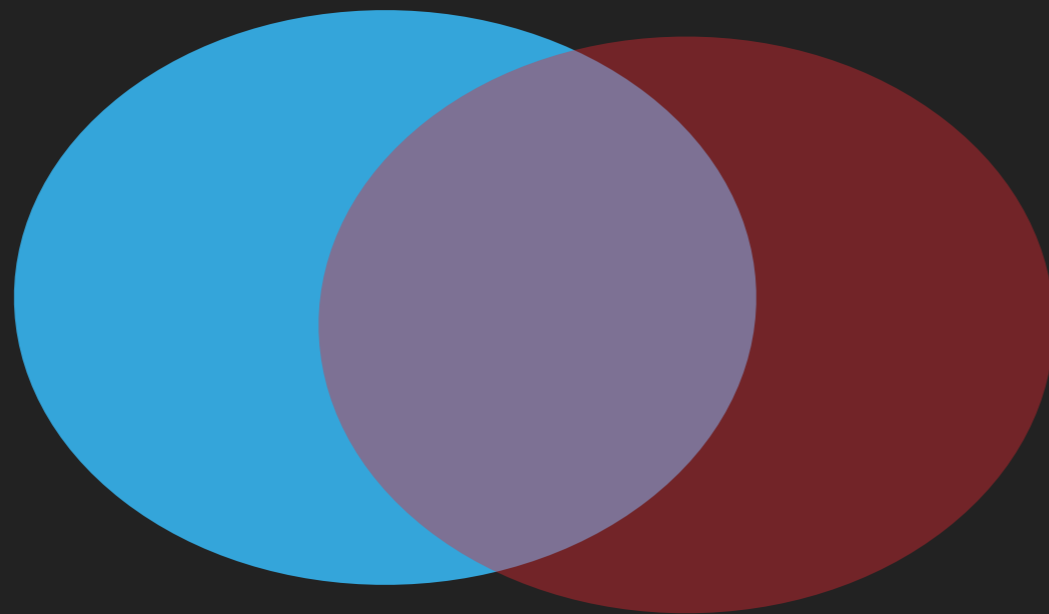
TEST DOUBLE IS AN APPROXIMATION



TEST DOUBLE IS AN APPROXIMATION



TEST DOUBLE IS AN APPROXIMATION



IMPROVE THE INTERFACE

```
interface PreviousPasswordChecker
{
    /**
     * Returns true if password has been used by user
     * in previous 5 passwords
     */
    public function isPreviouslyUsed(
        $password,
        $user
    )
}
```


IMPROVE THE INTERFACE

```
interface PreviousPasswordChecker
{
    /**
     * Returns true if password has been used by user
     * in previous 5 passwords
     */
    public function isPreviouslyUsed(
        string $password,
        $user
    )
}
```

IMPROVE THE INTERFACE

```
interface PreviousPasswordChecker
{
    /**
     * Returns true if password has been used by user
     * in previous 5 passwords
     */
    public function isPreviouslyUsed(
        string $password,
        User $user
    )
}
```

IMPROVE THE INTERFACE

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

OTHER REASONS FOR DIFFERENCES BETWEEN TEST DOUBLE

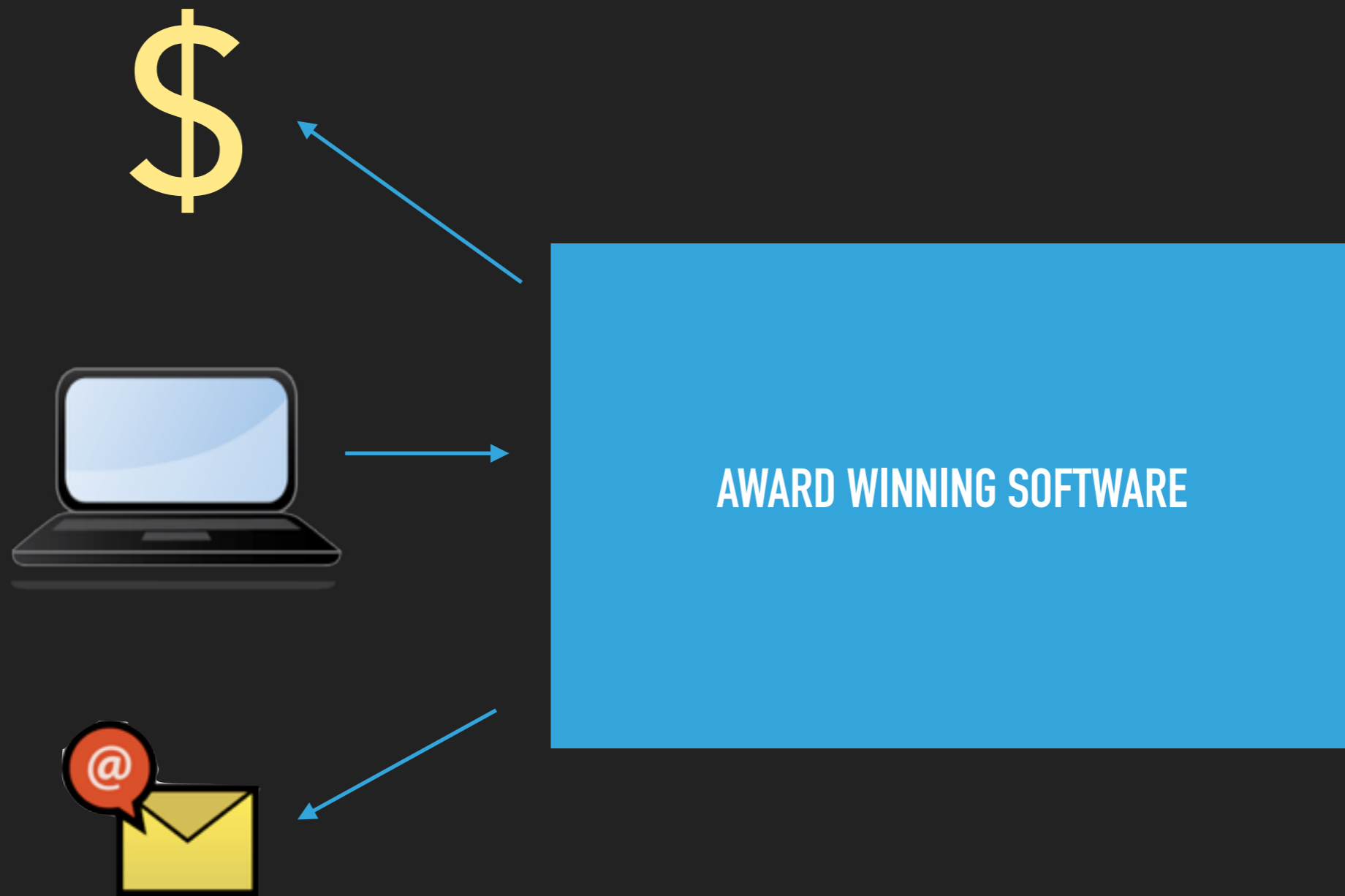
- ▶ Specification might change
- ▶ Specification might be misunderstood
- ▶ Functionality might not be implemented

UNIT TEST LEVEL

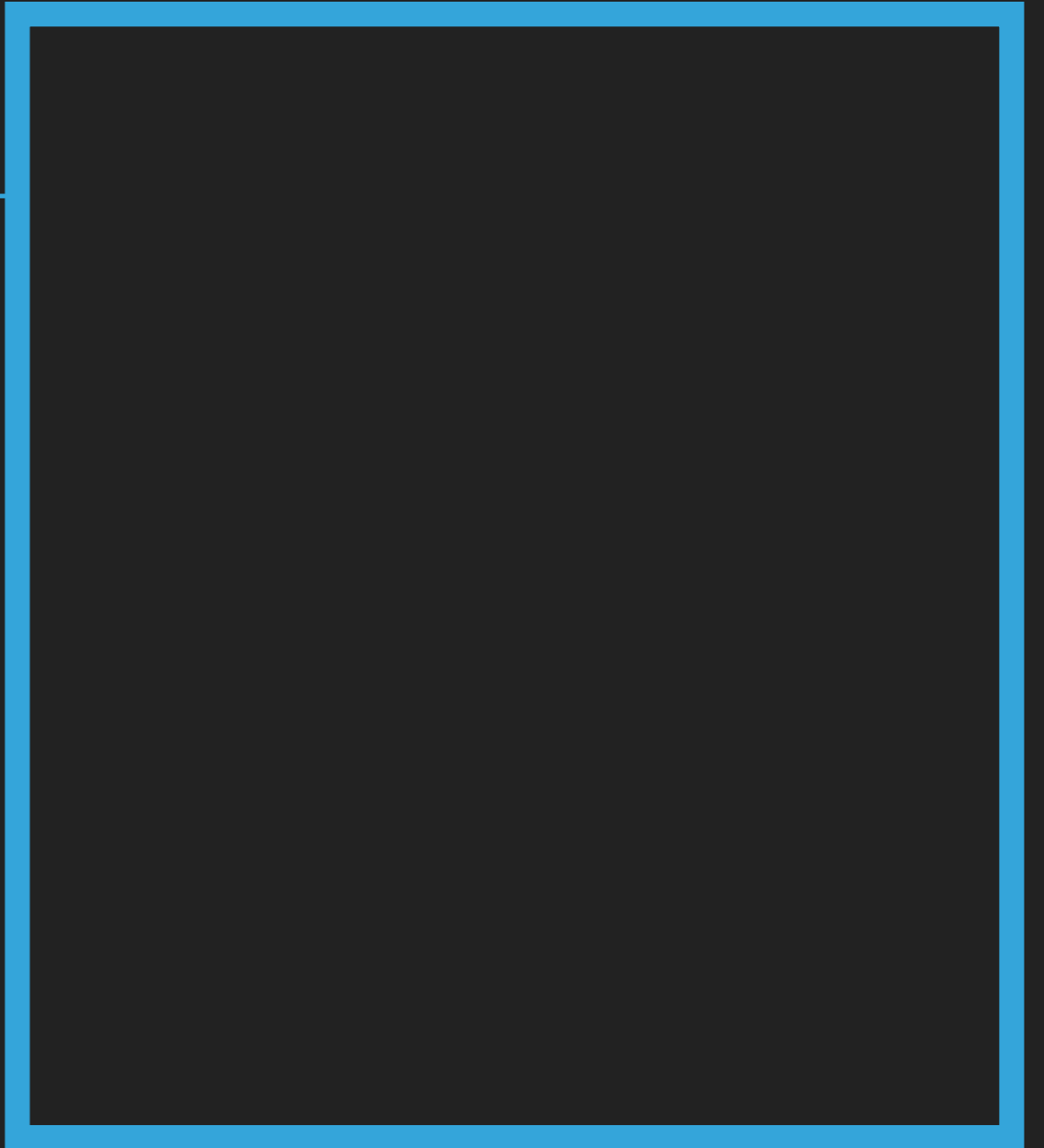
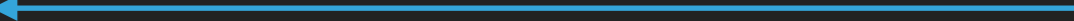
- ▶ Good architecture makes testing easier
- ▶ Decouple tests from code as much as possible
 - ▶ E.g. use stubs unless you really need a mock

BIGGER TESTS

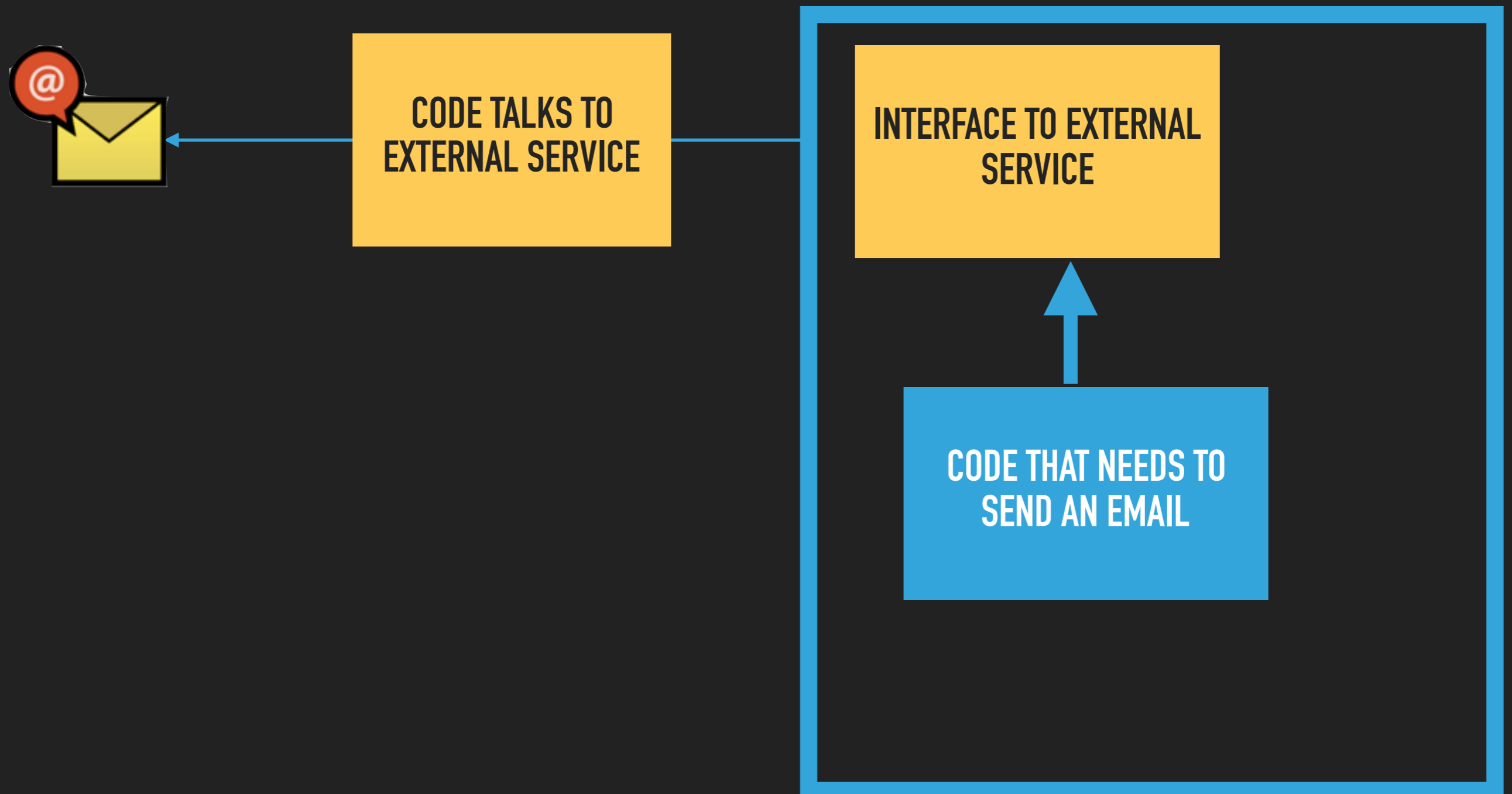
ARCHITECTURE



ARCHITECTURE



ARCHITECTURE



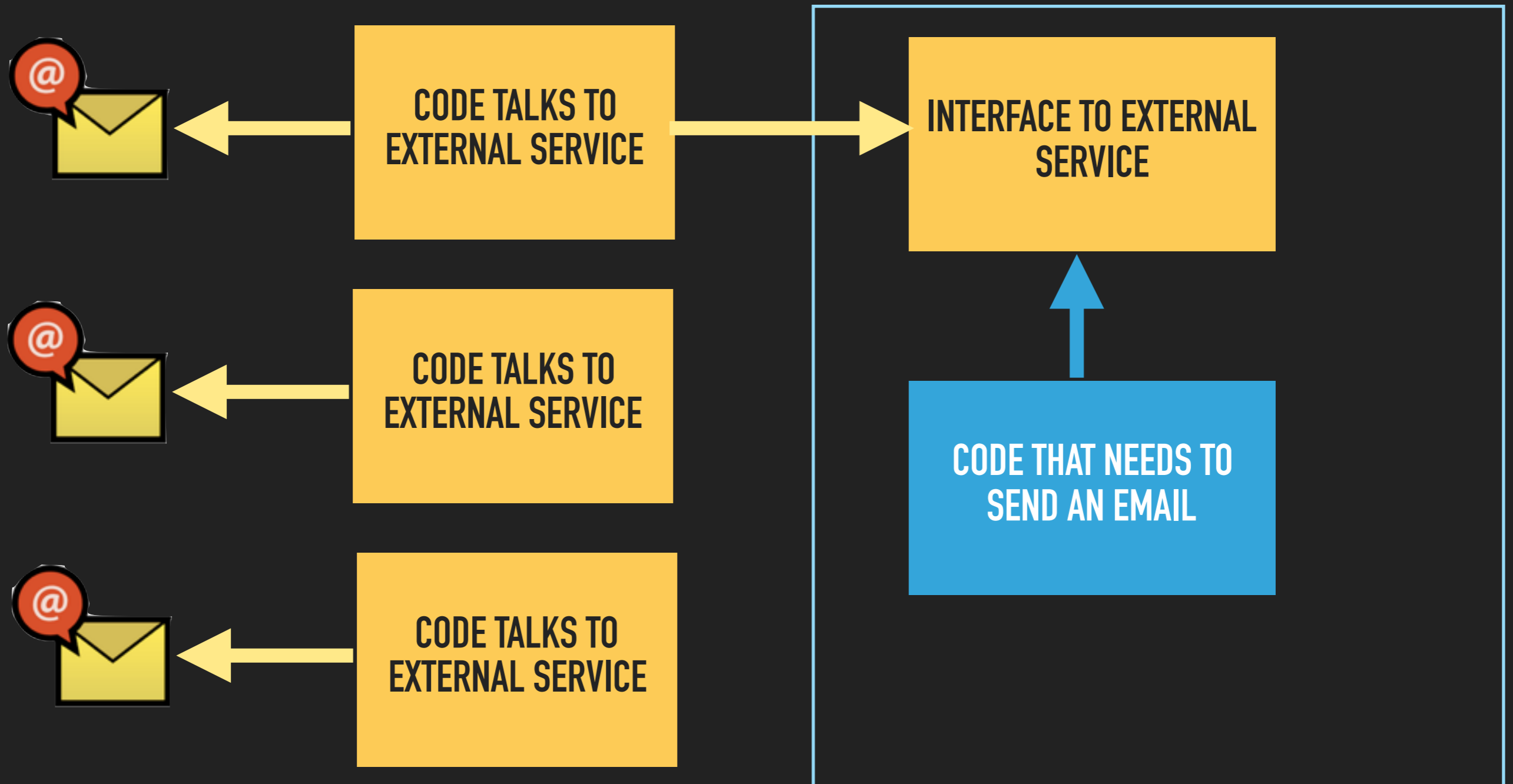
EMAIL GATEWAY INTERFACE

```
interface EmailGatewayInterface
{
    public function sendEmail(EmailMessage $message);
}
```

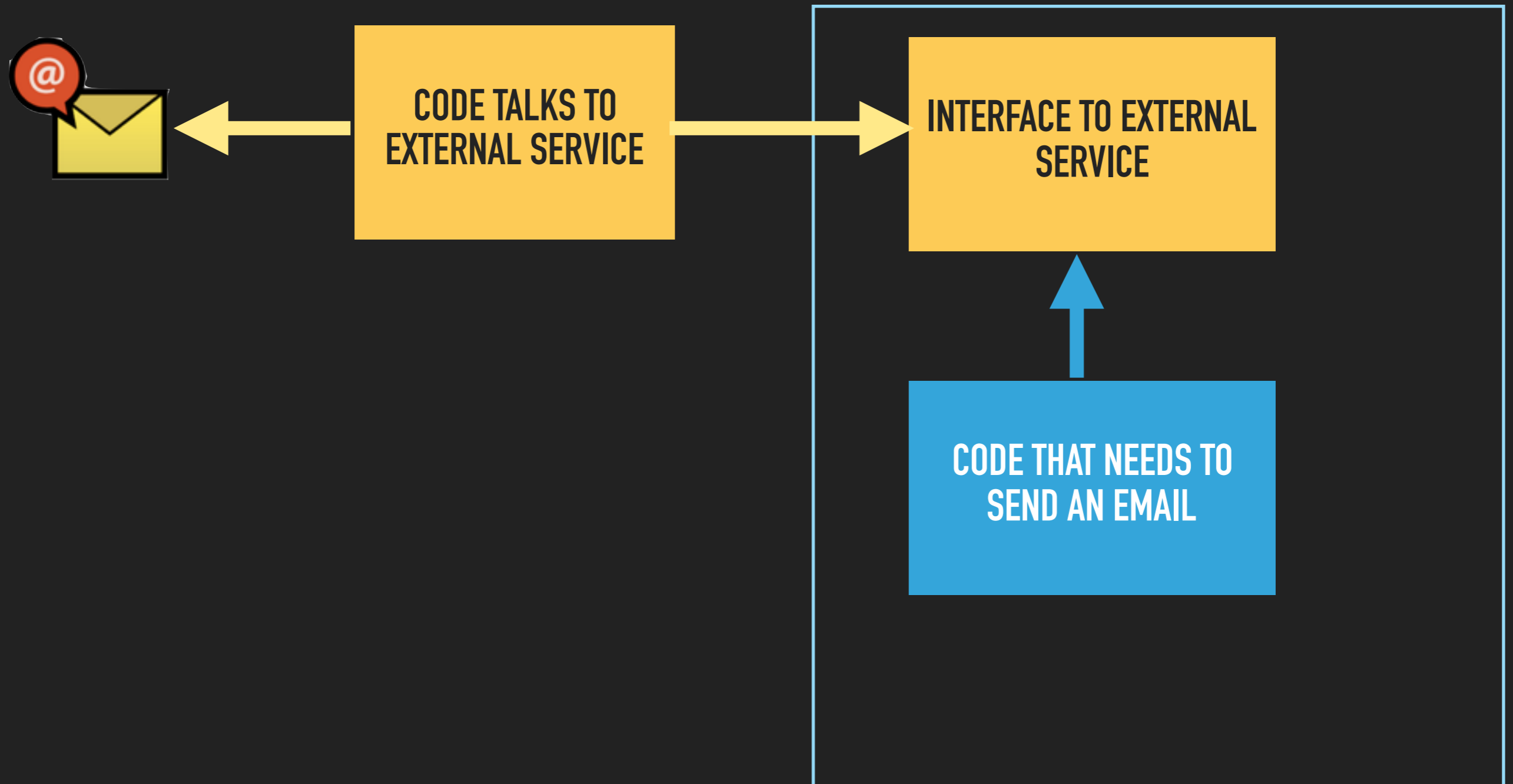
EMAIL MESSAGE OBJECT

- ▶ To
- ▶ From
- ▶ CC
- ▶ Subject
- ▶ Template
- ▶ Data

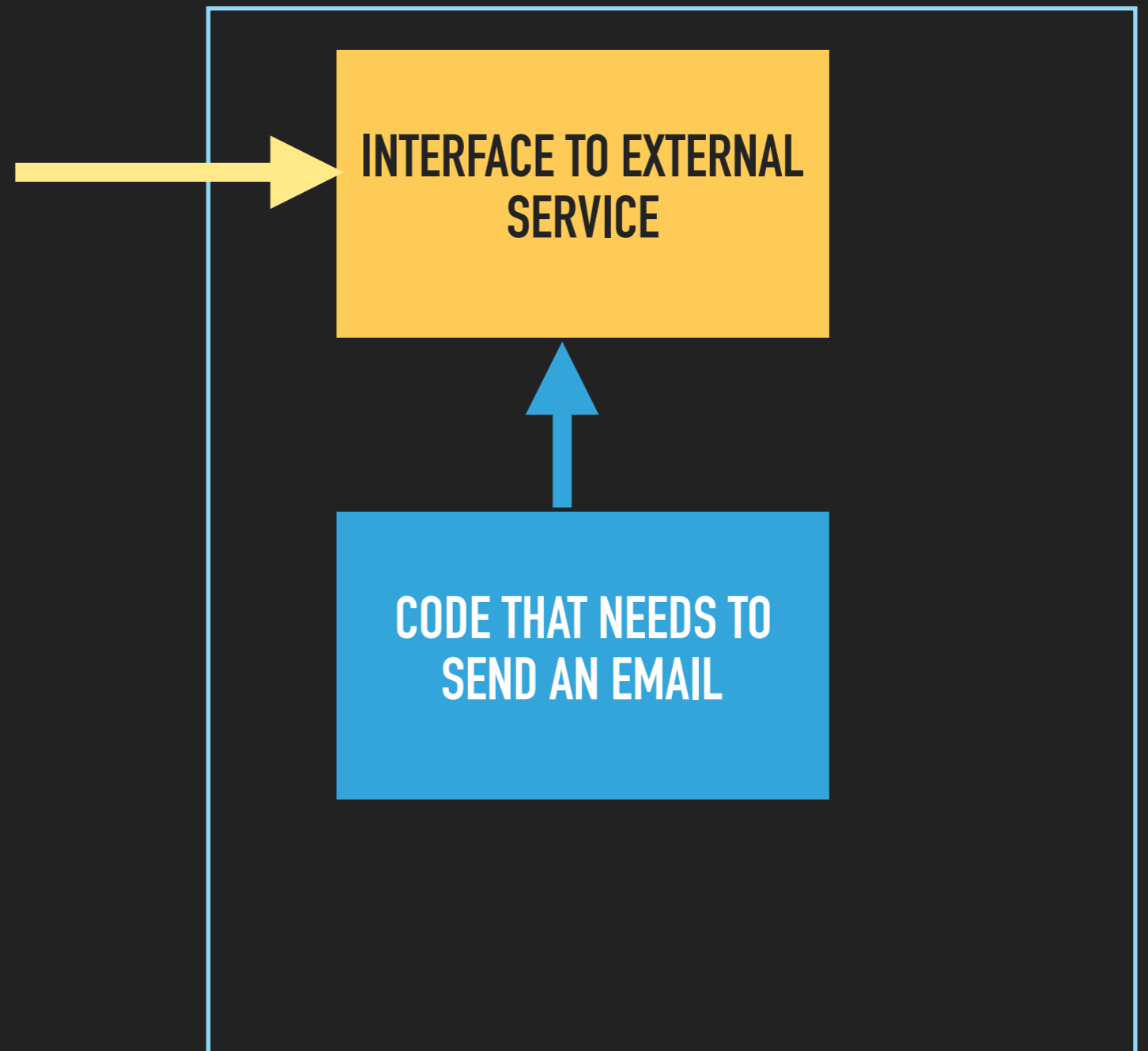
ARCHITECTURE



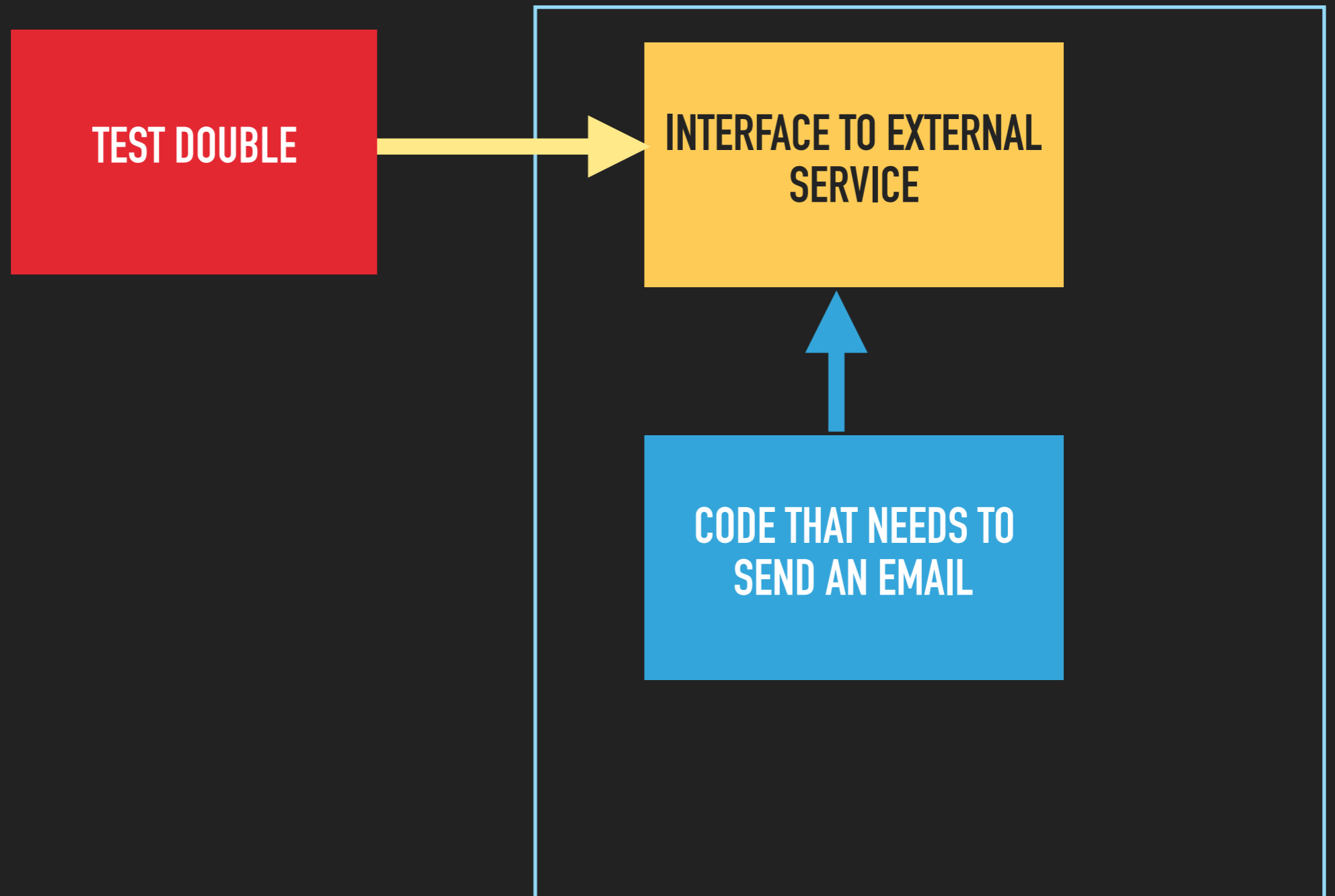
ARCHITECTURE



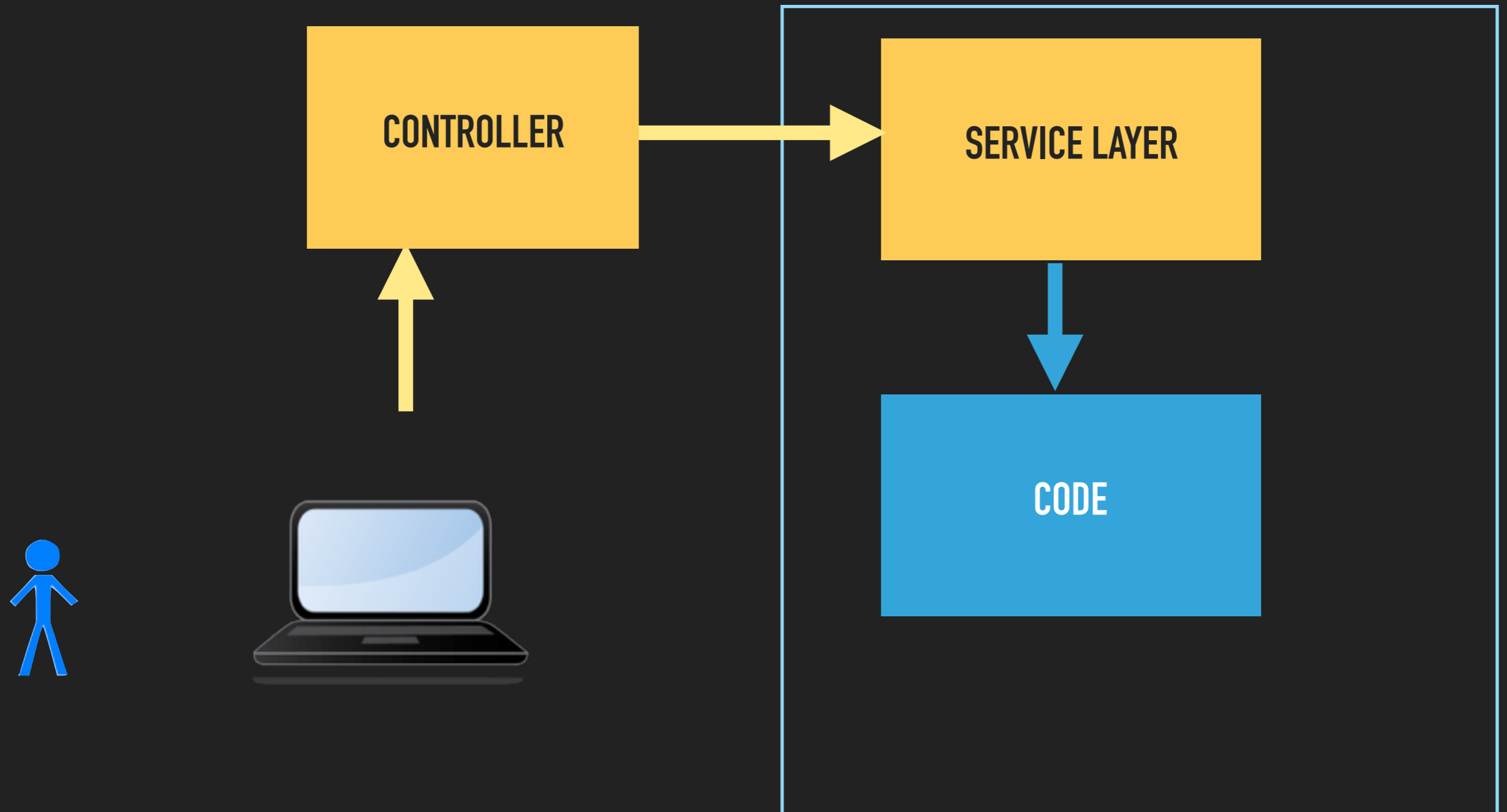
ARCHITECTURE



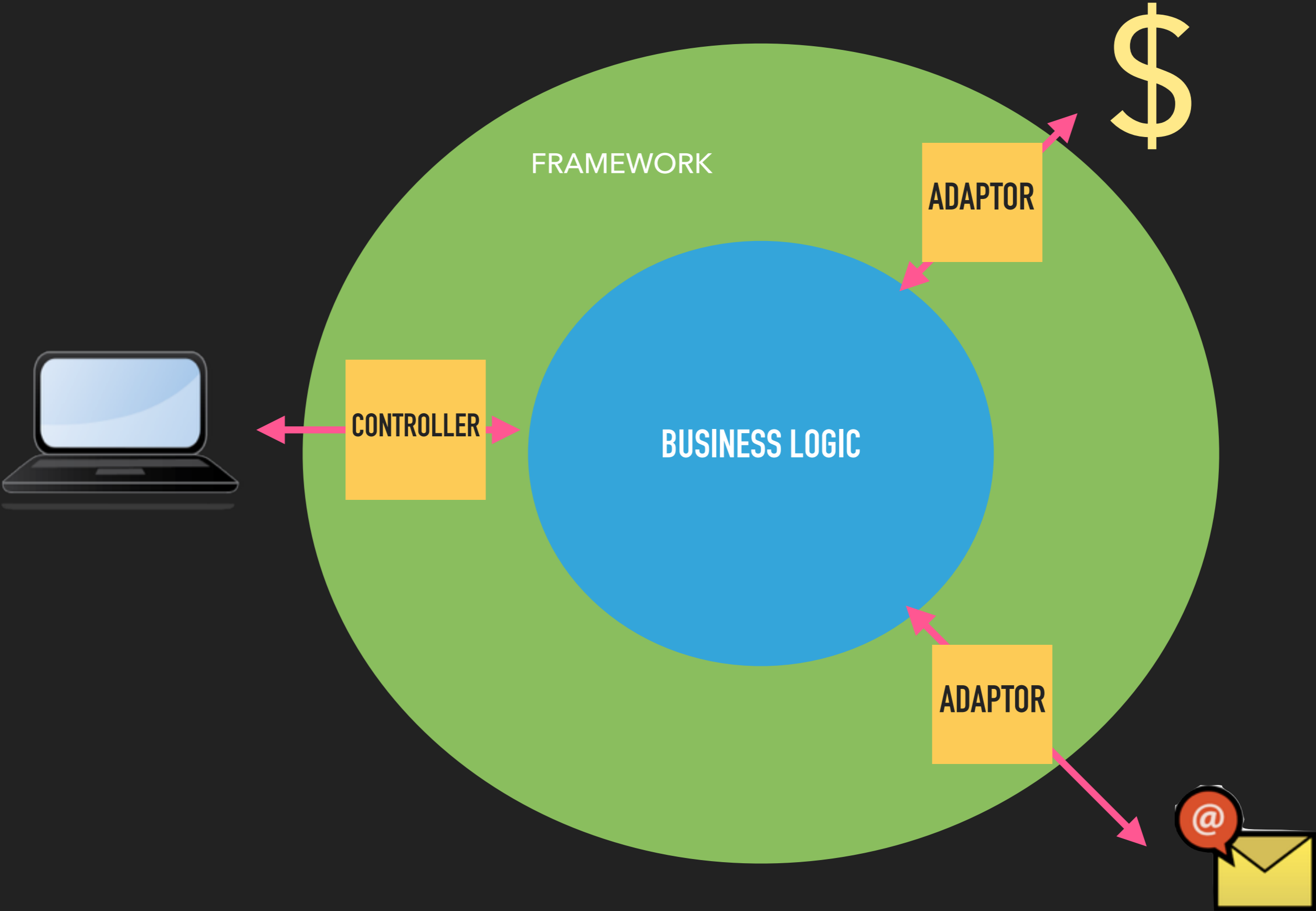
ARCHITECTURE



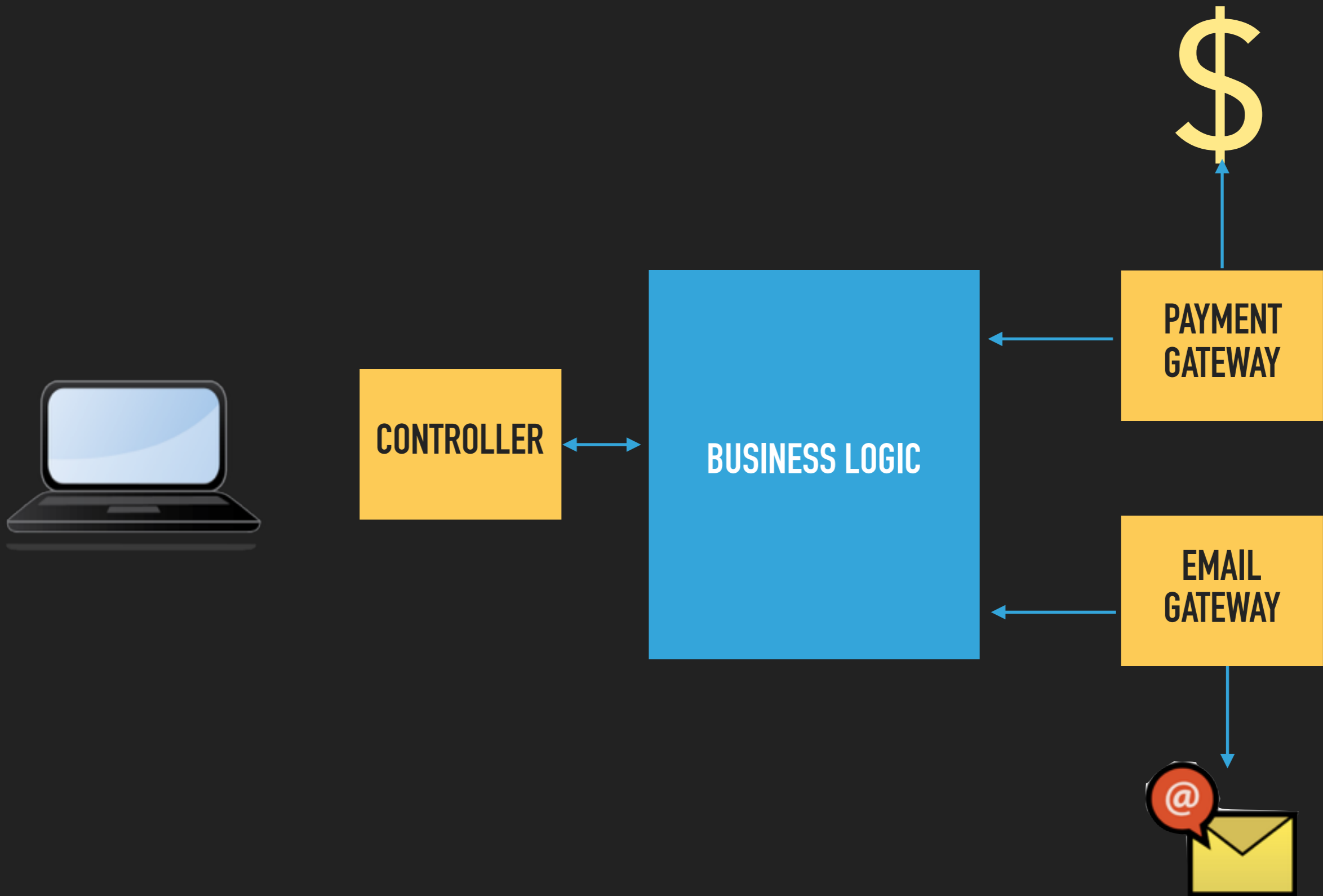
ARCHITECTURE



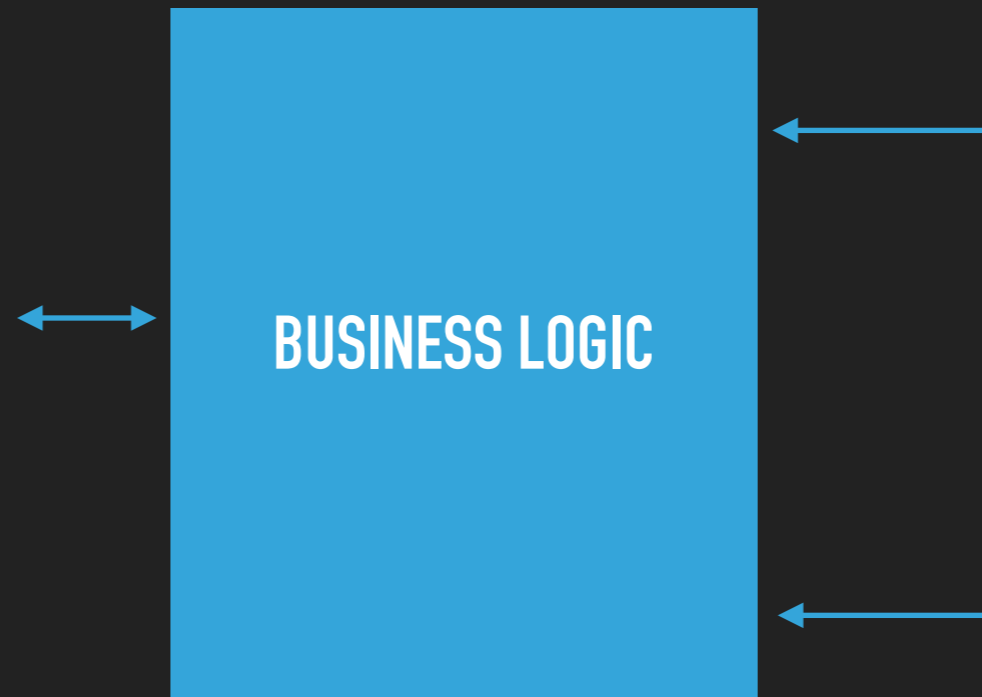
ARCHITECTURE



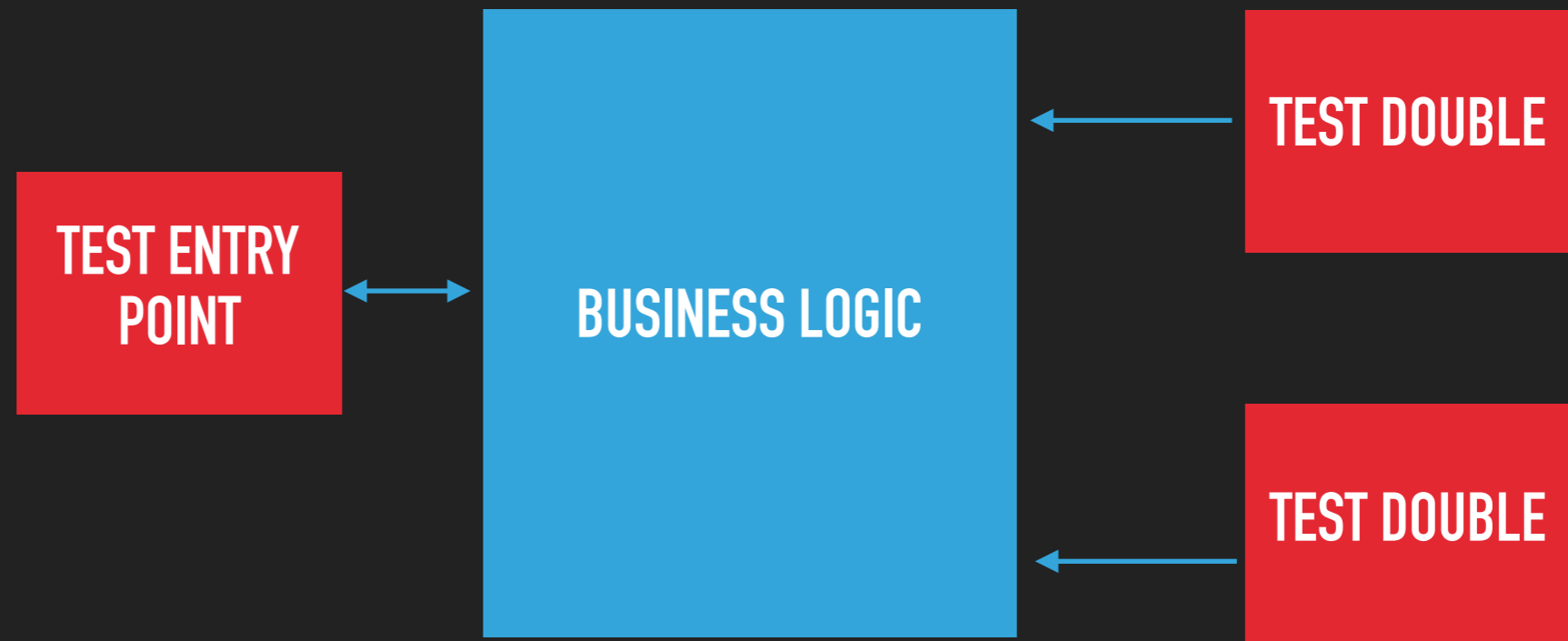
ARCHITECTURE



ARCHITECTURE



ARCHITECTURE



FAKE EMAIL GATEWAY

```
class FakeEmailGateway implements EmailGatewayInterface
{
    private $emailMessages = [];

    public function sendEmail(EmailMessage $message) {
        $this->emailMessages[] = $message;
    }

    public function findBy($to, $template): array {
        ... return EmailMessage meeting criteria ...
    }
}
```

REGISTER USER TEST 1

```
class RegisterUserTest extends TestCase
{
    public function testRegistration() {
        $userService = $this->container->getUserService();
        $success = $userService->registerUser(
            "dave@lampbristol.com", "Dave", "Passw0rd");
        $this->assertTrue($success);
    }
    ...
}
```

REGISTER USER TEST 2

...

```
$emailGateway = $this->container->getEmailGateway();

$emailMessages = $emailGateway->find(
    "dave@lampbristol.com", "REGISTRATION");

$this->assertCount(1, $emailMessages);

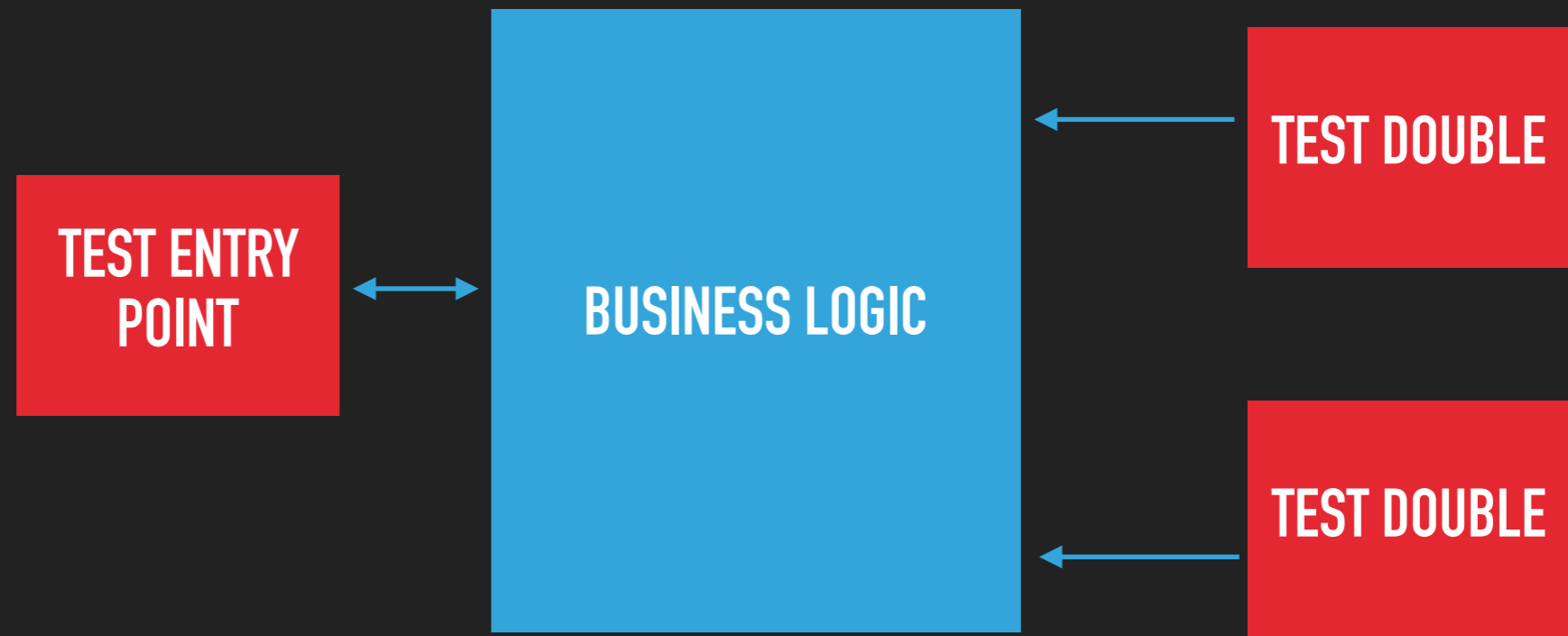
$data = $emailMessage->getData();
$confirmationToken = $data['confirmationToken'];

$success = $userService->completeRegistration(
    $confirmationToken);

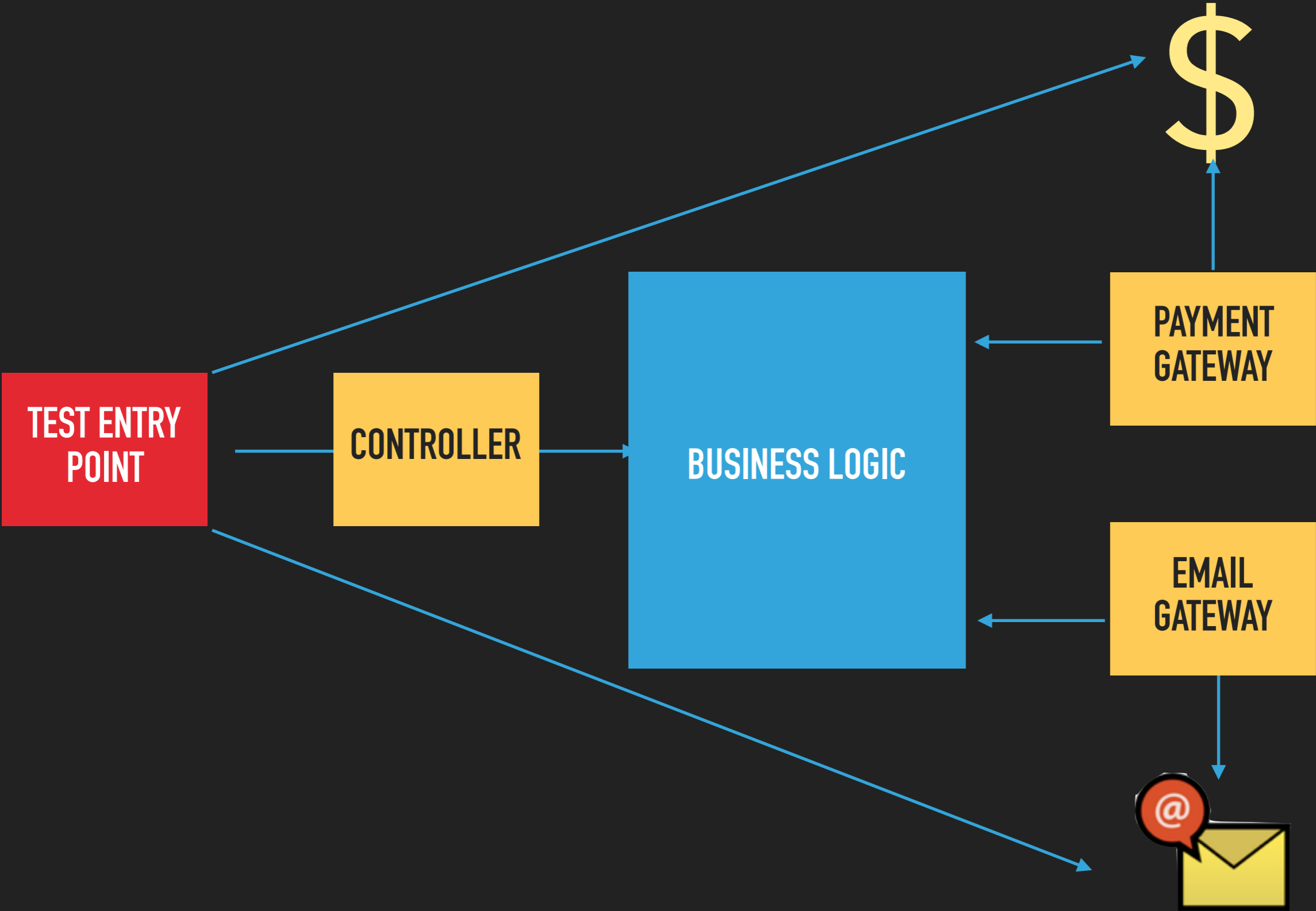
$this->assertTrue($success);
```

STOP AND ADMIRE

ARCHITECTURE



ARCHITECTURE



ARCHITECTURE IS VERY IMPORTANT

- ▶ High correlation between easy to test and good architecture.
- ▶ A code base isn't **difficult to test**, it's **poorly architected**.

RETURNING TO OUR PASSWORD VALIDATOR: 1

```
class PasswordValidatorTest extends TestCase
{
    public function testUpdatePassword() {
        ... create $user with password 'Passw1rd' ...

        $userService = $this->container->getUserService();

        $userService->updatePassword($user, 'Passw2rd');
        $userService->updatePassword($user, 'Passw3rd');
        $userService->updatePassword($user, 'Passw4rd');

        ...
    }
}
```

RETURNING TO OUR PASSWORD VALIDATOR: 2

...

```
$success = $userService->updatePassword(  
    $user, 'Passw1rd');  
$this->assertFalse($success);
```

```
$success = $userService->updatePassword(  
    $user, 'Passw5rd');  
$this->assertTrue($success);
```

```
$success = $userService->updatePassword(  
    $user, 'Passw1rd');  
$this->assertTrue($success);
```

THE BIG

INTEGRATION TEST

CONSTRUCTING OUR USER OBJECT

```
... create $user with password 'Passw1rd' ...
```

HOW DO WE BUILD THE TEST USER OBJECT?

- ▶ Hand build what is required
- ▶ Seed the database
- ▶ Object mother
- ▶ Test Builder

HAND BUILDING

```
$userService = $this->container->getUserService();
$userService->registerUser(
    "dave@lampbristol.com", "Dave", "Passw0rd");

$emailGateway = $this->container->getEmailGateway();

$emailMessages = $emailGateway->find(
    "dave@lampbristol.com", "REGISTRATION");

$data = $emailMessage->getData();
$confirmationToken = $data['confirmationToken'];

$userService->completeRegistration(
    $confirmationToken);
```

HAND BUILDING

```
$userService = $this->container->getUserService();  
$userService->registerUser(  
    "dave@lampbristol.com", "Dave", "Passw0rd");  
  
$emailGateway = $this->container->getEmailGateway();  
  
$emailMessages = $emailGateway->send(  
    "dave@lampbristol.com", "REGISTRATION");  
  
$data = $emailMessage->getData();  
$confirmationToken = $data['confirmationToken'];  
  
$userService->completeRegistration(  
    $confirmationToken);
```

SEEDING A DATABASE

users:

- name: Dave
email: dave@lampbristol.com
password: Passw1rd

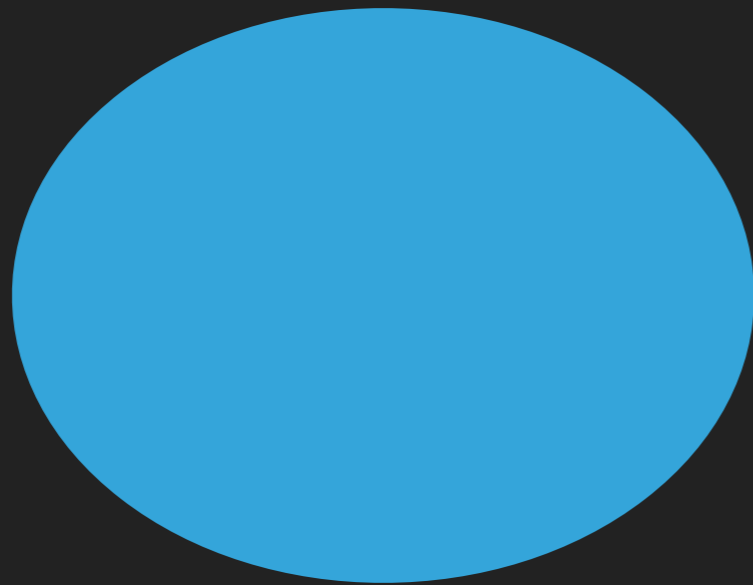
- name: Sarah
email: sarah@example.com
password: Passw5rd

SEEDING A DATABASE

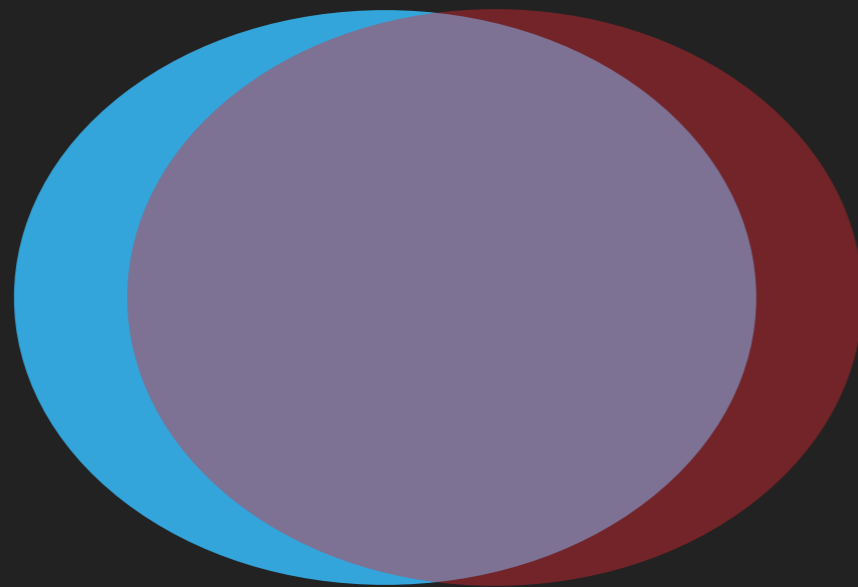
users:

- name: Dave
email: dave@lampbristol.com
password: Password
 - name: Sarah
email: sarah@example.com
password: Passw5rd
- 

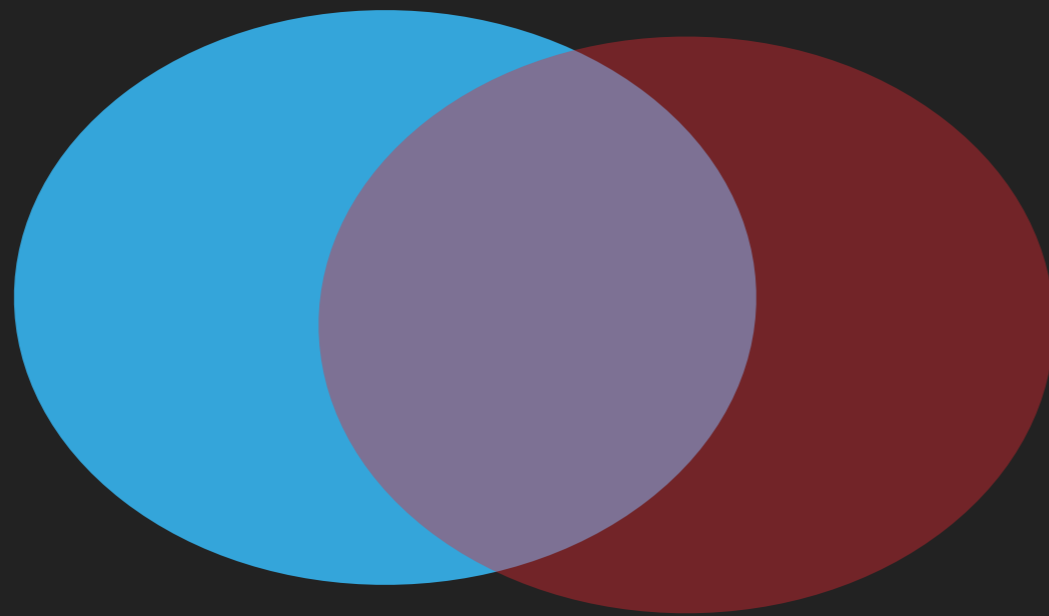
SEEDING A DATABASE IS AN APPROXIMATION



SEEDING A DATABASE IS AN APPROXIMATION



SEEDING A DATABASE IS AN APPROXIMATION



OBJECT MOTHER

```
$userObjectMother = $this->getUserObjectMother();
```

```
$user = $userObjectMother->getDave();
```

```
// User will have default values for name, email, etc
```


TEST BUILDER: 1

```
$userBuilder = $this->getUserBuilder();  
$user = $userBuilder->build();
```

```
// User will have default values for name, email, etc
```

USING A TEST BUILDER (2)

```
$userBuilder = $this->getUserBuilder();  
$user = $userBuilder  
    ->name("David")  
    ->password("Passw4rd")  
    ->previousPasswords([  
        "Passw1rd",  
        "Passw2rd",  
        "Passw3rd",  
    ])  
    ->build();
```

OBJECT MOTHER AND TEST BUILDER BENEFITS

- ▶ Single place where test business object built
 - ▶ Easy to find
 - ▶ Easy to update
- ▶ Defer to other Object Mothers / Test Builders

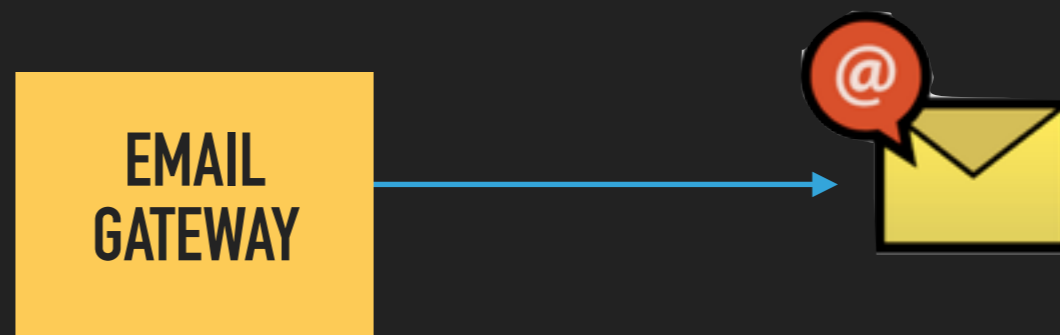
USE OBJECT MOTHER AND TEST BUILDER PATTERNS

- ▶ Reduce coupling between test and production code
- ▶ Help make your tests more resilient to change
 - ▶ Lowers maintenance cost
 - ▶ Increases our coverage

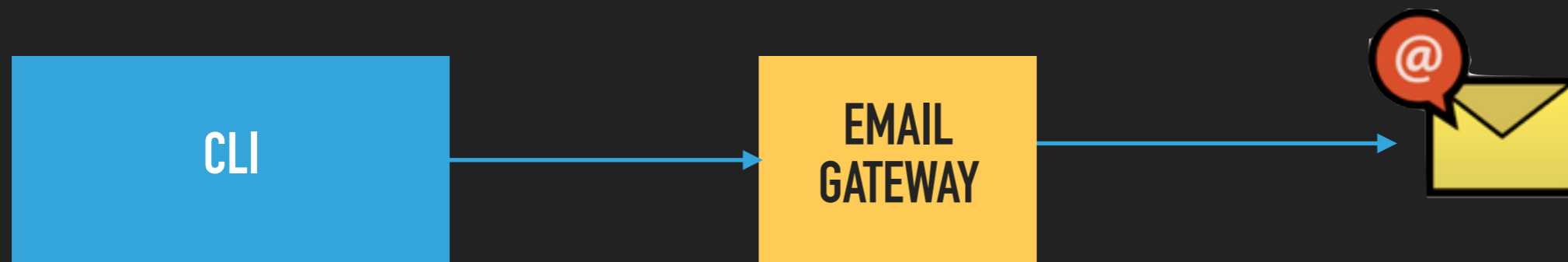
**FAMOUS 5 VS
AWKWARD DUO VS
THE BIG INTEGRATION TEST?**

**ASSESS VALUE OF TESTS.
REMOVE ONES THAT ARE
DUPLICATED (AND OFFER NO
BENEFIT)**

CAN WE AUTOMATE ANYTHING ELSE?



CAN WE AUTOMATE ANYTHING ELSE?



CAN WE AUTOMATE ANYTHING ELSE?

```
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]
```

WHY DO WE NEED A TEST SUITE

- ▶ Prove code works
- ▶ Prevent against regression
- ▶ Allow safe refactoring of code

OUR IDEAL TEST SUITE WOULD BE...

- ▶ Fast to execute
- ▶ High coverage
- ▶ Low maintenance

EVERY THING IS A COMPROMISE

- ▶ Nothing is black and white

TO MAKE A GOOD TEST SUITE

- ▶ Requires skill
- ▶ Good code architecture
- ▶ Reduce coupling between tests and code under test:
 - ▶ Use mocks only when needed
 - ▶ Use patterns like Object Mother and Test Builder

QUESTIONS

ARCHITECTURE

