

# No API? No problem!

API mocking with WireMock

An open source workshop by ...

# What are we going to do?

- \_Stubbing, mocking and service virtualization

- \_WireMock

- \_Exercises, examples, ...

# Preparation

\_Install JDK (Java 17 or newer)

\_Install IntelliJ IDEA (or any other IDE)

\_Download or clone project

\_Import Maven project in IDE

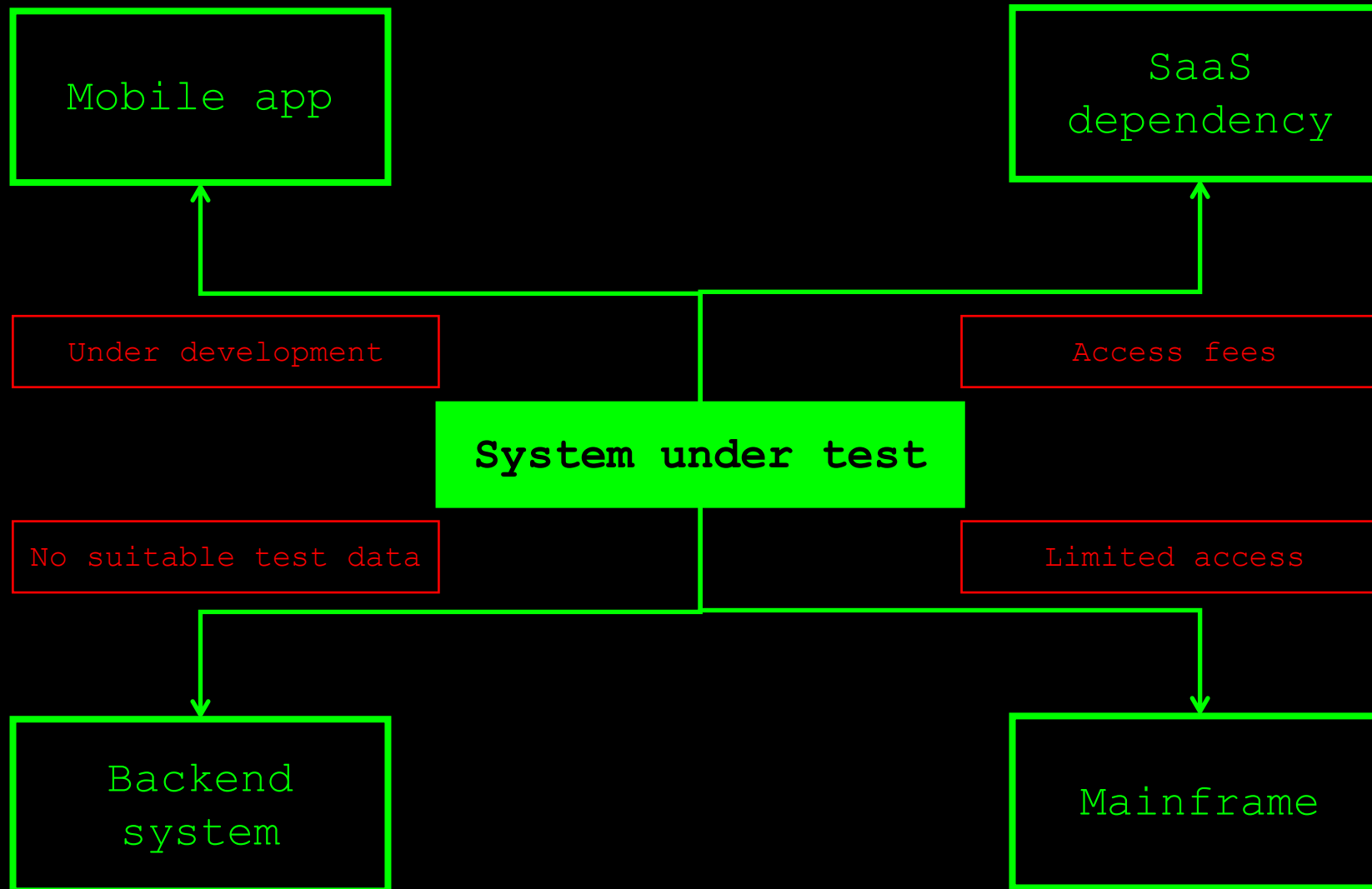
Section 0:

An introduction to  
service virtualization

# Problems in test environments

- \_ Systems are constructed out of many different components
- \_ Not all of these components are always available for testing
  - \_ Parallel development
  - \_ No control over test data
  - \_ Fees required for using third party components
  - \_ ...

# Problems in test environments



# Simulation during test execution

- \_Simulate dependency **behaviour**

- \_Regain full control over test environment

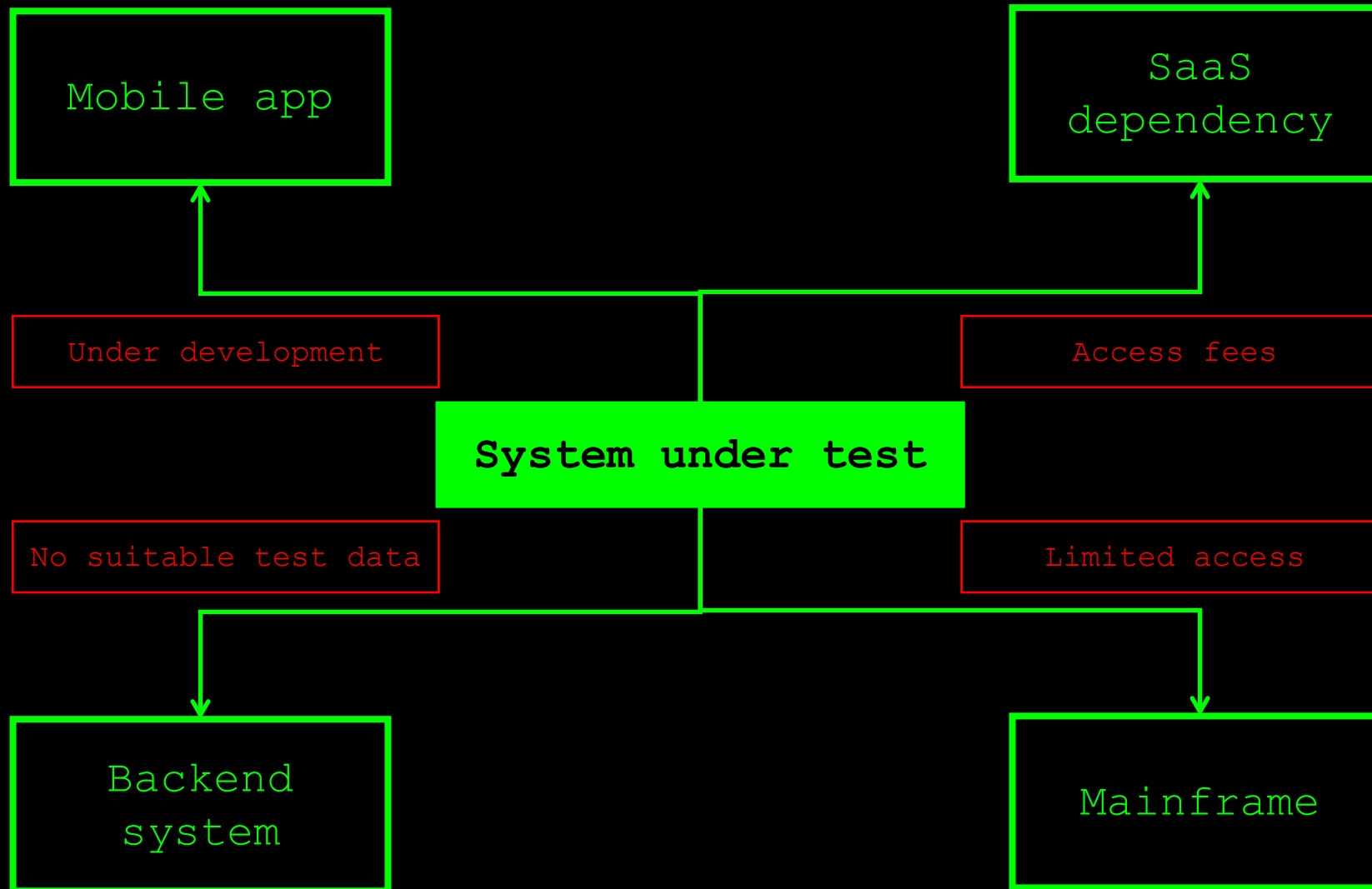
  - \_Available on demand

  - \_Full control over test data (edge cases!)

  - \_No third party component usage fees

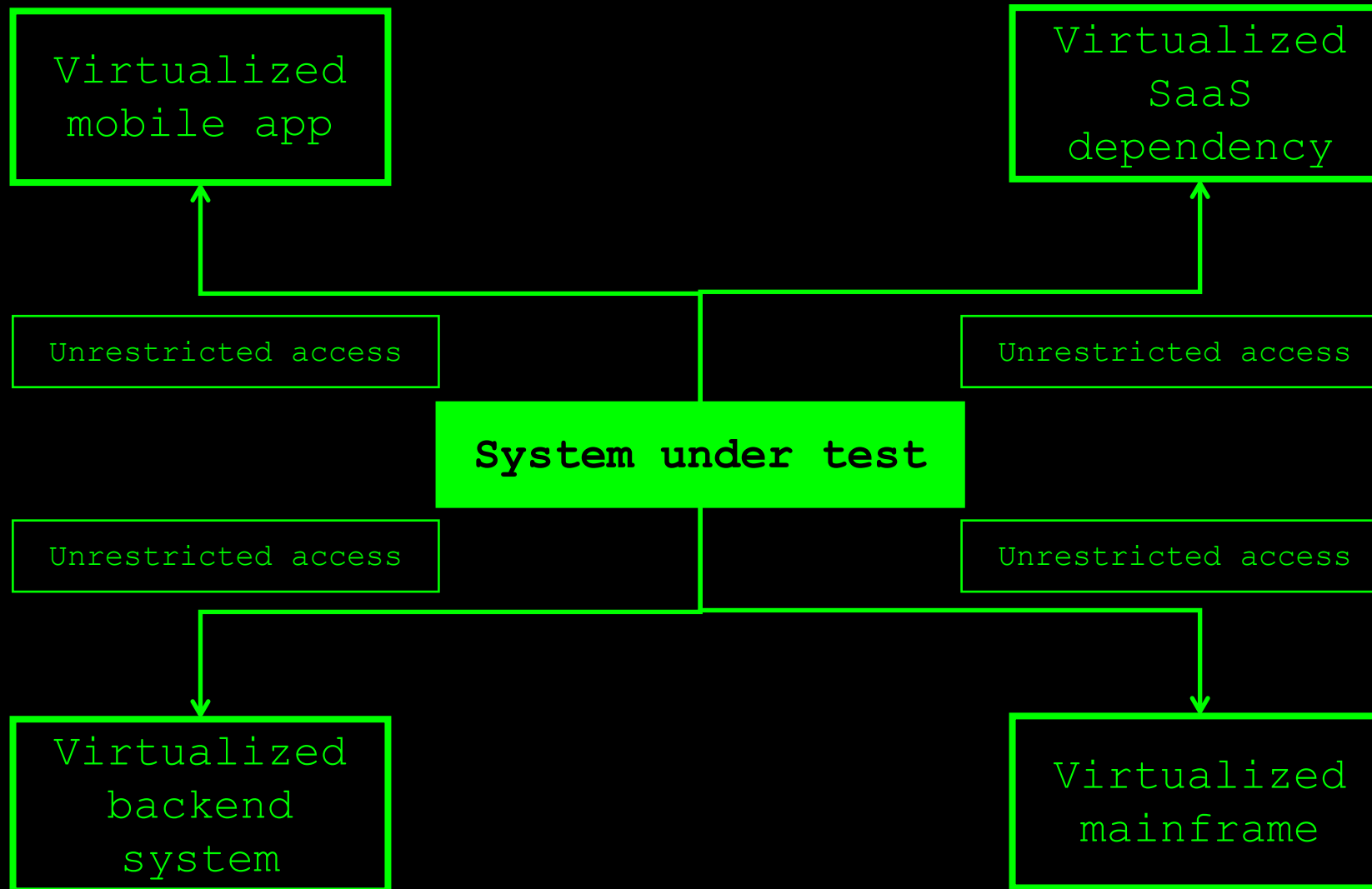
  - \_...

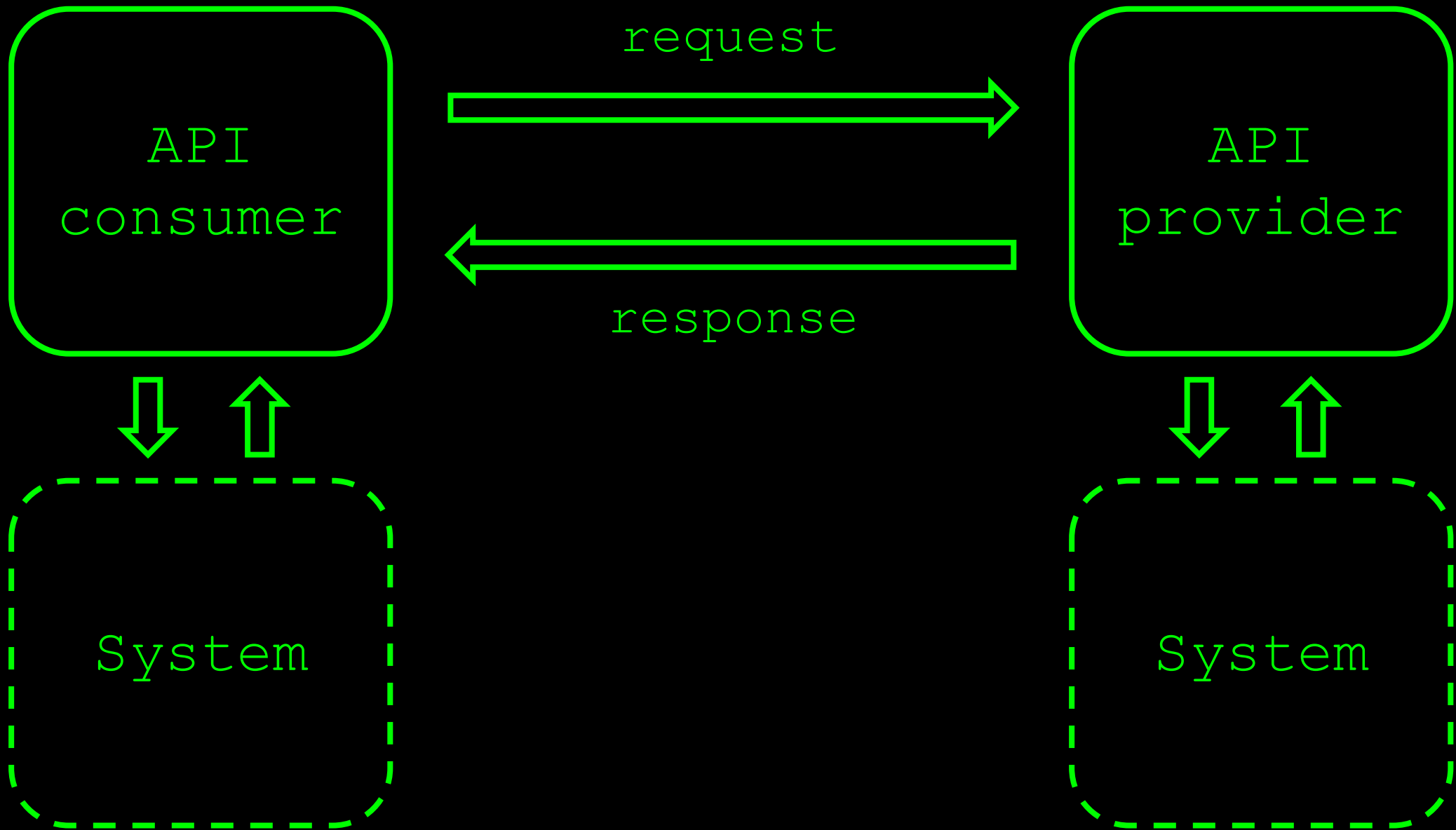
# Problems in test environments

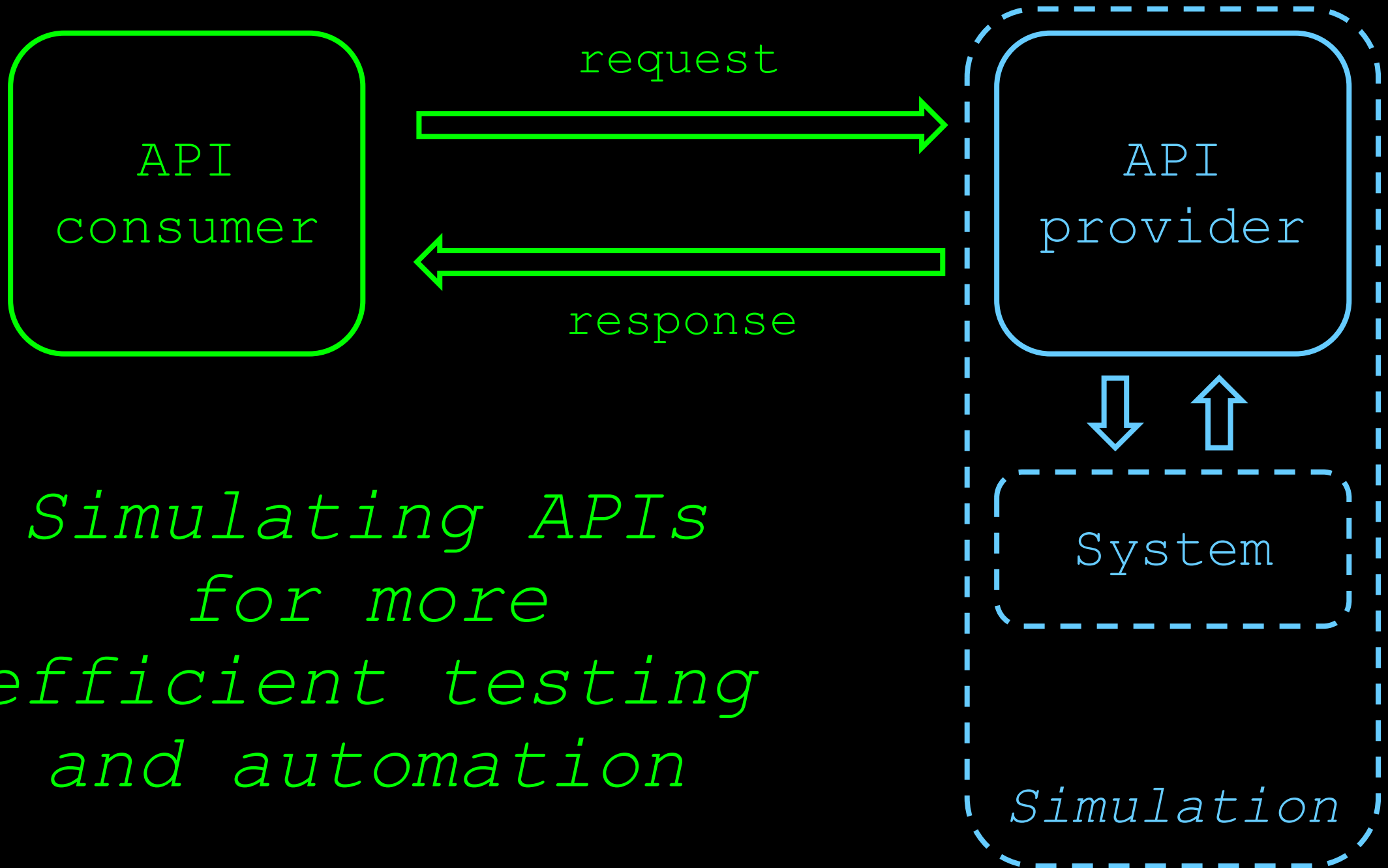




# Simulation in test environments







*Simulating APIs  
for more  
efficient testing  
and automation*

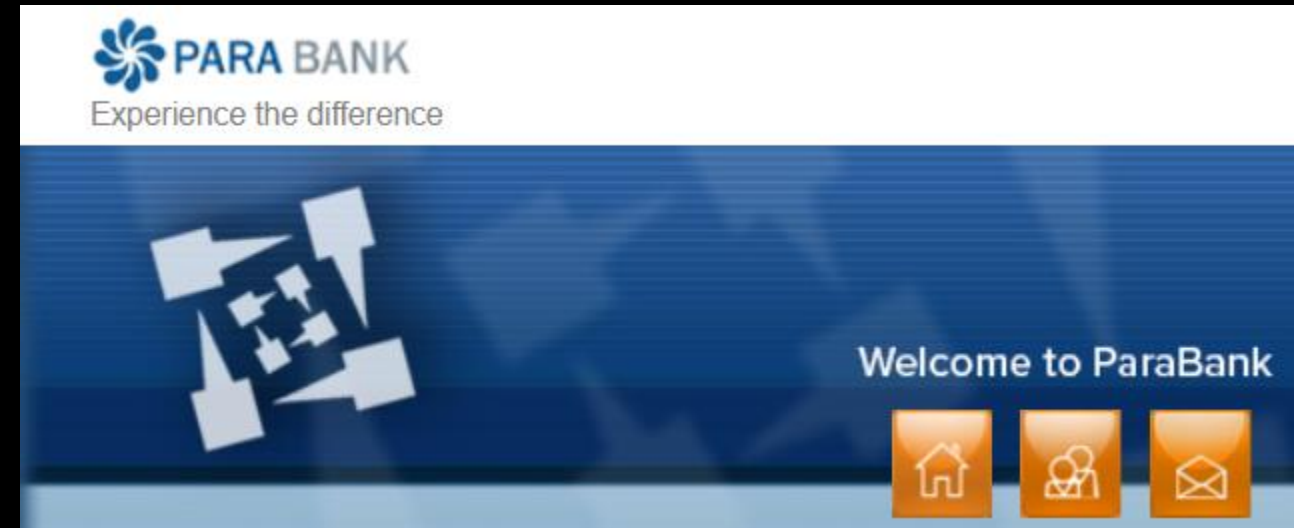
# Our system under test

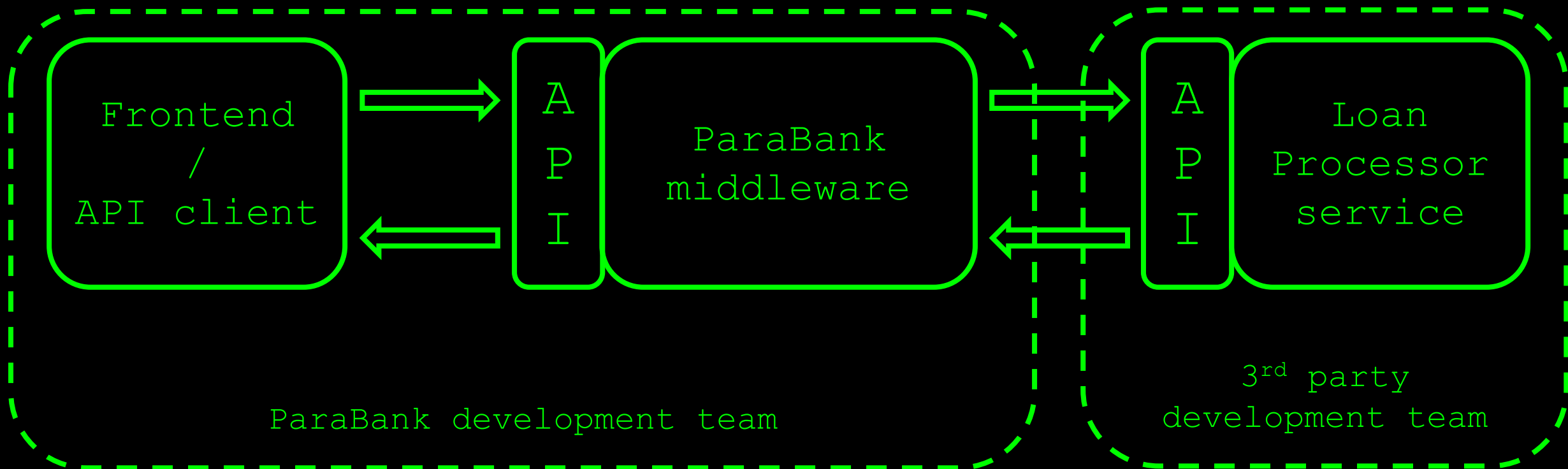
\_ParaBank

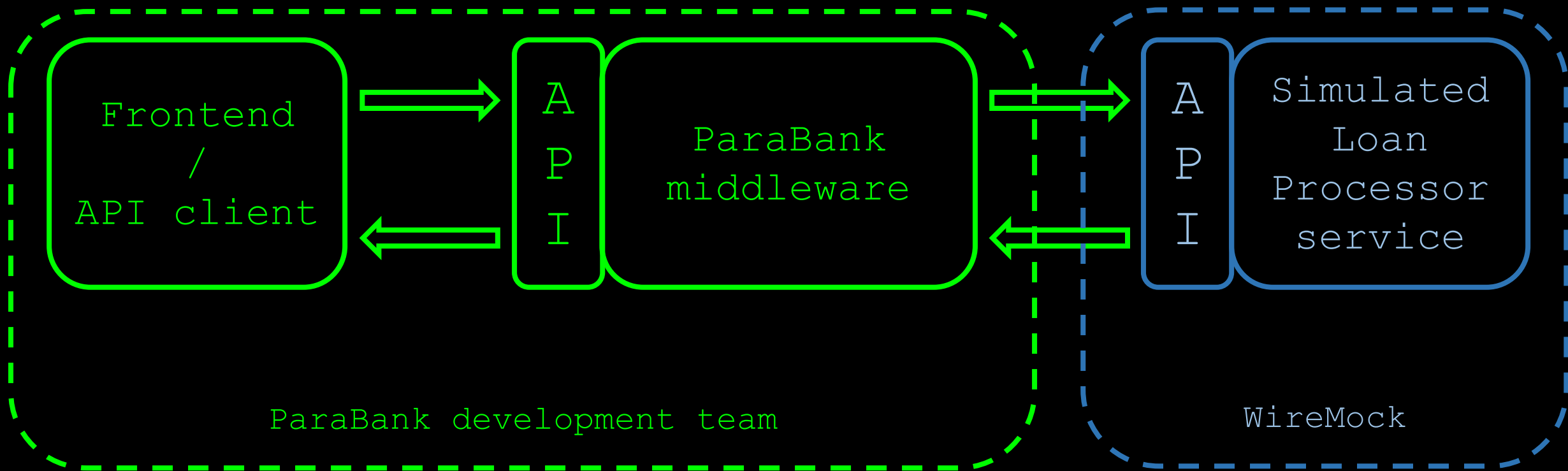
\_The world's least safe  
online bank

\_Request Loan process

\_Loan application is processed by 3rd party loan  
provider component







Start testing against features under development

Easy setup of state for edge cases

What might we  
want to simulate?

Delays, fault status codes, malformed responses, ...

...

Section 1:

Getting started with  
WireMock



# WireMock

- \_ <https://wiremock.org>

- \_ Java

- \_ ports and adapters are available for many other languages

- \_ HTTP mock server

- \_ only supports HTTP(S)

- \_ Open source

# Install WireMock

\_Maven

```
<dependency>  
  <groupId>org.wiremock</groupId>  
  <artifactId>wiremock</artifactId>  
  <version>3.3.1</version>  
  <scope>test</test>  
</dependency>
```

# Starting WireMock (JUnit 4)

## \_Via JUnit 4 @Rule

```
@Rule
public WireMockRule wireMockRule = new WireMockRule( port: 9876 );
```

## \_Without using JUnit 4 @Rule

```
WireMockServer wireMockServer =
    new WireMockServer(new WireMockConfiguration().port(9876));

wireMockServer.start();
```

# Starting WireMock (JUnit 5)

\_ Uses the JUnit 5 Jupiter extension mechanism

\_ Via @WireMockTest class annotation (basic configuration)

```
@WireMockTest(httpPort = 9876)
public class WireMockAnswers1Test {
```

\_ Programmatically using @RegisterExtension (full control)

```
@RegisterExtension
static WireMockExtension wiremock = WireMockExtension.newInstance()
    .options(wireMockConfig().port(9876).globalTemplating(true))
    .build();
```

# Starting WireMock (standalone)

\_Useful for exploratory testing purposes

\_Allows you to share WireMock instances between teams

\_Long-running instances

\_Download the .jar first

```
java -jar wiremock-standalone-3.3.1.jar --port 9876
```

# Configure responses

\_In (Java) code

\_Using JSON mapping files

# An example mock defined in Java

```
public void helloWorld() {  
  
    stubFor(  
        get(  
            urlEqualTo(testUrl: "/helloworld")  
        )  
        .willReturn(  
            aResponse()  
                .withHeader(key: "Content-Type", ...values: "text/plain")  
                .withStatus(200)  
                .withBody("Hello world!")));  
    }  
}
```

# Some useful WireMock features

## \_ Verification

- \_ Verify that certain requests are sent by application under test

## \_ Record and playback

- \_ Generate mocks based on request-response pairs (traffic)

## \_ Fault simulation

\_ ...

\_ Full documentation at <https://wiremock.org/docs/>



# Now it's your turn!

\_exercises > WireMockExercises1Test.java

\_Create a couple of basic mocks

\_Implement the responses as described in the comments

\_Verify your solution by running the tests in the same file

\_Answers are in answers > WireMockAnswers1Test.java

\_Examples are in examples > WireMockExamplesTest.java

## Section 2:

Request matching  
strategies and fault  
simulation

# Request matching

\_ Send a response only when certain properties in the request are matched

\_ Options for request matching:

\_ URL

\_ HTTP method

\_ Query parameters

\_ Headers

\_ Request body elements

\_ ...

# Example: URL matching

```
public void setupStubURLMatching() {  
    stubFor(get(urlEqualTo("/urlmatching"))  
        .willReturn(aResponse()  
            .withBody("URL matching")  
        ));  
}
```

## \_Other URL options:

- \_urlPathEqualTo (matches only path, no query parameters)
- \_urlMatching (using regular expressions)
- \_urlPathMatching (using regular expressions)

# Example: header matching

```
public void setupStubHeaderMatching() {  
  
    stubFor(get(urlEqualTo( testUrl: "/headermatching"))  
        .withHeader(s: "Content-Type", containing(value: "application/json"))  
        .withHeader(s: "DoesntExist", absent())  
        .willReturn(aResponse()  
            .withBody("Header matching")  
        ));  
}
```

`_absent()`: check that header is **not** in request

## Example: using logical AND and OR

```
public void setupStubLogicalAndHeaderMatching() {  
  
    stubFor(get(urlEqualTo( testUrl: "logical-or-matching"))  
        .withHeader( s: "my-header",  
            matching( regex: "[a-z]+" ) .and( containing( value: "somevalue" ) )  
        )  
        .willReturn(aResponse()  
            .withBody("Logical AND matching"))  
    );  
}
```

- \_ 'somevalue' is matched
- \_ 'bananasomevaluebanana' is matched
- \_ 'banana' is not matched (does not contain 'somevalue')
- \_ '123somevalue' is not matched (contains numeric characters)

# Some more examples...

```
public void setupStubLogicalAndHeaderMatchingMoreVerbose() {  
  
    stubFor(get(urlEqualTo( testUrl: "logical-or-matching"))  
        .withHeader( s: "my-header", and(  
            matching( regex: "[a-z]+" ),  
            containing( value: "somevalue" ) )  
        )  
        .willReturn(aResponse()  
            .withBody("Logical AND matching, a little more verbose"))  
    );  
}
```

Same behaviour as the previous example,  
using a slightly different syntax

```
public void setupStubLogicalOrHeaderMatching() {  
  
    stubFor(get(urlEqualTo( testUrl: "logical-or-matching"))  
        .withHeader( s: "Content-Type",  
            equalTo( value: "application/json" ) .or( absent() ) )  
        )  
        .willReturn(aResponse()  
            .withBody("Logical OR matching"))  
    );  
}
```

# Matching on request body elements

```
public void setupStubRequestBodyValueMatching() {  
    stubFor(post(urlEqualTo( testUrl: "/request-body-matching"))  
        .withRequestBody(  
            matchingJsonPath( value: "$.fruits[?(@.banana == '2')]" )  
        )  
        .willReturn(aResponse()  
            .withBody("Request body matched successfully"))  
        );  
}
```

Matching only those request bodies that have a root level element *fruits* with a child element *banana* with value 2

`{"fruits": {"banana": "2", "apple": "5"} }` ➔ **MATCH**

`{"fruits": {"apple": "5"} }` ➔ **NO MATCH**

`{"fruits": {"banana": "3", "apple": "5"} }` ➔ **NO MATCH**



# Matching using date/time properties

```
public void setupStubAfterSpecificDateMatching() {  
  
    stubFor(get(urlEqualTo( testUrl: "date-is-after"))  
        .withHeader( s: "my-date",  
            after(dateTimeSpec: "2021-07-01T00:00:00Z")  
        )  
        .willReturn(aResponse()  
            .withBody("Date is after midnight, July 1, 2021"))  
    );  
}
```

Matching all dates after  
midnight of July 1, 2021

```
public void setupStubRelativeToCurrentDateMatching() {  
  
    stubFor(get(urlEqualTo( testUrl: "date-is-relative-to-now"))  
        .withHeader( s: "my-date",  
            beforeNow().expectedOffset( amount: 1, DateTimeUnit.MONTHS)  
        )  
        .willReturn(aResponse()  
            .withBody("Date is at least 1 month before current date"))  
    );  
}
```

Matching all dates at least 1  
month before the current date

# Other matching strategies

`_Authentication (Basic, OAuth(2))`

`_Query parameters`

`_Multipart/form-data`

`_You can write your own matching logic, too`

# Fault simulation

- \_Extend test coverage by simulating faults

- \_Often hard to do in real systems

- \_Easy to do using stubs or mocks

- \_Used to test the exception handling of your application under test

# Example: HTTP status code

```
public void setupStubReturningErrorCode() {  
  
    stubFor(get(urlEqualTo( testUrl: "/errorcode"))  
        .willReturn(aResponse()  
            .withStatus(500)  
            .withStatusMessage("Status message goes here")  
        ));  
}
```

Some often used HTTP status codes:

## Consumer error

403 (Forbidden)

404 (Not found)

## Provider error

500 (Internal server error)

503 (Service unavailable)

# Example: timeout

```
public void setupStubFixedDelay() {  
    stubFor(get(urlEqualTo( testUrl: "/fixeddelay"))  
        .willReturn(aResponse()  
            .withFixedDelay(2000)  
        ));  
}
```

— Random delay can also be used

— Uniform, lognormal distribution

— Can be configured on a per-stub basis as well as globally

# Example: bad response

```
public void setupStubBadResponse() {  
  
    stubFor(get(urlEqualTo( testUrl: "/badresponse" ))  
        .willReturn(aResponse()  
            .withFault(Fault.MALFORMED_RESPONSE_CHUNK)  
        ));  
}
```

\_HTTP status code 200, but garbage in response body

\_Other options:

\_RANDOM\_DATA\_THEN\_CLOSE (as above, without HTTP 200)

\_EMPTY\_RESPONSE (does what it says on the tin)

\_CONNECTION\_RESET\_BY\_PEER (close connection, no response)

# Now it's your turn!

\_exercises > WireMockExercises2Test.java

\_Practice fault simulation and different request matching strategies

\_Implement the responses as described in the comments

\_Verify your solution by running the tests in the same file

\_Answers are in answers > WireMockAnswers2Test.java

\_Examples are in examples > WireMockExamplesTest.java

Section 3:

Creating stateful mocks



# Statefulness

\_ Sometimes, you want to simulate stateful behaviour

\_ Shopping cart (empty / containing items)

\_ Database (data present / not present)

\_ Order in which requests arrive is significant

# Stateful mocks in WireMock

- \_Supported through the concept of a Scenario

- \_Essentially a finite state machine (FSM)

  - \_States and state transitions

- \_Combination of current state and incoming request determines the response being sent

  - \_Before now, it was only the incoming request

# Stateful mocks: an example

```
public void setupStubStateful() {  
  
    stubFor(get(urlEqualTo(testUrl: "/order")).inScenario(s: "Order processing")  
        .whenScenarioStateIs(Scenario.STARTED)  
        .willReturn(aResponse()  
            .withBody("Your shopping cart is empty")  
        )  
    );  
  
    stubFor(post(urlEqualTo(testUrl: "/order")).inScenario(s: "Order processing")  
        .whenScenarioStateIs(Scenario.STARTED)  
        .withRequestBody(equalTo(value: "Ordering 1 item"))  
        .willReturn(aResponse()  
            .withBody("Item placed in shopping cart")  
        )  
        .willSetStateTo("ORDER_PLACED")  
    );  
  
    stubFor(get(urlEqualTo(testUrl: "/order")).inScenario(s: "Order processing")  
        .whenScenarioStateIs("ORDER_PLACED")  
        .willReturn(aResponse()  
            .withBody("There is 1 item in your shopping cart")  
        )  
    );  
}
```

Responses are grouped by scenario name

Response depends on both the incoming request as well as the current state

The initial state should always be `Scenario.STARTED`

Incoming requests can trigger state transitions

State names other than `Scenario.STARTED` are yours to define

# Now it's your turn!

\_exercises > WireMockExercises3Test.java

\_Create a stateful mock that exerts the described behaviour

\_Implement the responses as described in the comments

\_Verify your solution by running the tests in the same file

\_Answers are in answers > WireMockAnswers3Test.java

\_Examples are in examples > WireMockExamplesTest.java

Section 4:

Response templating

# Response templating

\_Often, you want to reuse elements from the request in the response

\_Request ID header

\_Unique body elements (client ID, etc.)

\_Cookie values

\_WireMock supports this through response templating

## Setup response templating (JUnit 4)

\_In code: through the JUnit @Rule

```
@Rule
public WireMockRule wireMockRule =
    new WireMockRule(wireMockConfig()
        .port(9876)
        .extensions(new ResponseTemplateTransformer( global: true))
    );
```

\_Global == false: response templating transformer  
has to be enabled for individual stubs

# Setup response templating (JUnit 5)

\_In code: through the JUnit @RegisterExtension

```
@RegisterExtension
static WireMockExtension wiremock = WireMockExtension.newInstance()
    .options(wireMockConfig().port(9876).globalTemplating(true))
    .build();
```

\_Argument == false: response templating has to be enabled for individual stubs



# Enable/apply response templating

— This template reads the HTTP request method (GET/POST/PUT/...) using `{{request.method}}` and returns it as the response body

```
public void setupStubResponseTemplatingHttpMethod() {  
    wiremock.stubFor(any(urlEqualTo( testUrl: "/template-http-method" ))  
        .willReturn(aResponse()  
            .withBody("You used an HTTP {{request.method}}")  
            .withTransformers("response-template")  
        ));  
}
```

This call to `withTransformers()` is only necessary when response templating isn't activated globally

# One thing to keep in mind...

```
@RegisterExtension
static WireMockExtension wiremock = WireMockExtension.newInstance()
    .options(wireMockConfig().port(9876).globalTemplating(true))
    .build();
```

Because we're explicitly initializing  
a WireMock instance here...

```
public void setupStubResponseTemplatingHttpMethod() {
    wiremock.stubFor(any(urlEqualTo( testUrl: "/template-http-method")))
        .willReturn(aResponse()
            .withBody("You used an HTTP {{request.method}}")
            .withTransformers("response-template")
        )); ... we need to explicitly assign our stub definition to that instance
    } here, or else the stub definition will not be picked up!
```

# Request attributes

Many different request attributes available for use

|  |                                |
|--|--------------------------------|
| <code>_request.method</code>                 | : HTTP method (example)        |
| <code>_request.pathSegments.&lt;n&gt;</code> | : $n^{\text{th}}$ path segment |
| <code>_request.headers.&lt;key&gt;</code>    | : header with name <i>key</i>  |
| <code>_...</code>                            |                                |

All available attributes listed at

[\*https://wiremock.org/docs/response-templating/\*](https://wiremock.org/docs/response-templating/)

# Request attributes (cont'd)

\_Extracting and reusing body elements

\_In case of a JSON request body:

```
{{jsonPath request.body '$.path.to.element'}}
```

\_In case of an XML request body:

```
{{XPath request.body '/path/to/element/text()'}}
```

# JSON extraction example

\_When sent this JSON request body:

```
{
  "book": {
    "author": "Ken Follett",
    "title": "Pillars of the Earth",
    "published": 2002
  }
}
```

\_This stub returns a response with body "Pillars of the Earth":

```
public void setupStubResponseTemplatingJsonBody() {
    stubFor(post(urlEqualTo( testUrl: "/template-json-body"))
        .willReturn(aResponse()
            .withBody("{\"jsonPath request.body '$.book.title'}")
            .withTransformers("response-template")
        ));
}
```

Again, this call to `withTransformers()` is only necessary when response templating isn't activated globally

# Now it's your turn!

\_exercises > WireMockExercises4Test.java

\_Create mocks that use response templating

\_Implement the responses as described in the comments

\_Verify your solution by running the tests in the same file

\_Answers are in answers > WireMockAnswers4Test.java

\_Examples are in examples > WireMockExamplesTest.java

Section 5:

Verification

# Verifying incoming requests

- Apart from returning responses, you might also want to verify that incoming requests have certain properties

- Fail a test if these verifications aren't met

- You can do this with WireMock in a way very similar to mocking frameworks for unit tests (e.g., Mockito for Java)



```
public void setupHelloWorldStub() {
```

Given this simple  
'hello world' stub

```
    stubFor(  
        get(  
            urlEqualTo( testUrl: "/hello-world")  
        )  
        .willReturn(  
            aResponse()  
                .withHeader( key: "Content-Type", ...values: "text/plain")  
                .withStatus(200)  
                .withBody("Hello world!")  
        )  
    );  
}
```

# Verifying incoming requests

Then this verification can be added to the test to ensure that indeed, an HTTP GET to '/hello-world' has been made exactly once

```
verify(exactly( expected: 1), getRequestedFor(urlEqualTo( testUrl: "/hello-world")));
```

```
@Test
```

```
public void helloWorldVerificationTest() {
```

```
    setupHelloWorldStub();
```

When we have this  
test that should  
invoke that stub  
exactly once

```
    given().
```

```
        spec(requestSpec).
```

```
    when().
```

```
        get( s: "/hello-world").
```

```
    then().
```

```
    and().
```

```
        body(org.hamcrest.Matchers.equalTo( operand: "Hello world!"));
```

```
verify(exactly( expected: 1), getRequestedFor(urlEqualTo( testUrl: "/hello-world")));
```

## Some more verification examples

```
verify(getRequestedFor(urlEqualTo( testUrl: "/hello-world")));
```

The same as the above, but less verbose

```
verify(lessThan( expected: 5), postRequestedFor(urlEqualTo( testUrl: "/requestLoan")));
```

Verify that less than 5 HTTP POSTs were made to /requestLoan

```
verify(  
    moreThanOrExactly( expected: 10),  
    postRequestedFor(urlEqualTo( testUrl: "/requestLoan"))  
        .withHeader( key: "Content-Type", containing( value: "application/json"))  
);
```

Verify that 10 or more HTTP POSTs with a 'Content-Type' header value containing 'application/json' were made to /requestLoan

# Now it's your turn!

\_exercises > WireMockExercises5Test.java

\_Add WireMock verifications to the tests

\_Verify request properties as described in the comments

\_Verify your solution by running the tests

\_Answers are in answers > WireMockAnswers5Test.java

\_Examples are in examples > WireMockExamplesTest.java

Section 6:

Extending WireMock

# Extending WireMock

- \_ In some cases, the default WireMock feature set might not fit your needs
- \_ WireMock is open to extensions
- \_ Allows you to create even more powerful stubs
- \_ Several options available

Section 6.1:

Filtering incoming  
requests

# Request filtering

- \_ Modify incoming requests (or halt processing)

- \_ This has a variety of use cases:

  - \_ Checking authentication details

  - \_ Request header injection

  - \_ URL rewriting

- \_ Created by implementing the *StubRequestFilterV2* interface

# Request filtering - build

```
public class HttpDeleteFilter implements StubRequestFilterV2 {  
  
    @Override  
    public RequestFilterAction filter(Request request, ServeEvent serveEvent) {  
        If the HTTP verb used equals DELETE...  
        if (request.getMethod().equals(RequestMethod.DELETE)) {  
            return RequestFilterAction.stopWith(ResponseDefinition.notPermitted("HTTP DELETE is not allowed!"));  
        }  
        Return an HTTP 403 and stop  
        processing the request  
        return RequestFilterAction.continueWith(request);  
    }  
  
    @Override  
    Else continue processing the request  
    public String getName() { return "http-delete-filter"; }  
}
```



# Request filtering - use

```
@RegisterExtension
static WireMockExtension wiremock = WireMockExtension.newInstance().
    options(wireMockConfig().
        port(9876).
        extensions(new HttpDeleteFilter())
    ).build();
```

An extension can be registered using:

- its class name ("com.example.HttpDeleteFilter")
- the class (HttpDeleteFilter.class)
- an instance (new HttpDeleteFilter())

# Now it's your turn!

\_exercises > extensions > BasicAuthFilter.java

\_Implement a custom request filter that filters out  
all requests that do not have the proper basic  
authentication credentials

\_Verify your solution by running the tests in  
\_exercises > WireMockExercises6dot1Test.java

\_Answers are in answers > extensions >  
\_BasicAuthFilter.java

\_Examples are in examples > extensions >  
\_HttpDeleteFilter.java

Section 6.2:

Building a custom  
request matcher

# Custom request matchers

- \_ Add custom request matching logic to WireMock

- \_ Can be combined with existing standard matchers

- \_ Done by extending RequestMatcherExtension class

# Custom request matcher - build

```
public class BodyLengthMatcher extends RequestMatcherExtension {  
  
    @Override  
    public String getName() {  
        return "body-too-long";  
    }  
  
    @Override           Get the value of the maxLength matcher parameter  
    public MatchResult match(Request request, Parameters parameters) {  
        int maxLength = parameters.getInt( key: "maxLength");  
        return MatchResult.of(request.getBody().length > maxLength);  
    }  
}           Compare the request body length to the maxLength  
           parameter value and return the result as a MatchResult
```

# Custom request matcher – use

```
@RegisterExtension
static WireMockExtension wiremock = WireMockExtension.newInstance().
    options(wireMockConfig().
        port(9876).
        extensions(new BodyLengthMatcher())
    ).build();
```

Register the extension

Use custom matcher in a  
stub definition using its  
name (can be combined  
with existing matchers)

Specify desired parameter value

```
stubFor(get(urlEqualTo( testUrl: "/custom-matching"))).
    andMatching( "body-too-long", Parameters.one( name: "maxLength", value: 20))
    willReturn(aResponse().withStatus(400))
);
```

# Now it's your turn!

```
exercises > extensions >  
-RejectedHttpVerbsMatcher.java
```

```
Implement a custom matcher that reads a list of  
-rejected HTTP verbs and matches the HTTP verb used in  
the incoming request against it
```

```
Verify your solution by running the tests in  
-exercises > WireMockExercises6dot2Test.java
```

```
Answers are in answers > extensions >  
-RejectedHttpVerbsMatcher.java
```

```
Examples are in examples > extensions >  
-BodyLengthMatcher.java
```

Section 6.3 is waiting on

<https://github.com/wiremock/wiremock/issues/2525>

to be resolved



[https://wiremock.org/docs  
/extending-wiremock/](https://wiremock.org/docs/extending-wiremock/)

# Appendix A:

JSON equivalents for  
the Java examples

# Our Hello world! mock

```
{
  "request": {
    "method": "GET",
    "url": "/helloworld"
  },
  "response": {
    "status": 200,
    "body": "Hello world!",
    "headers": {
      "Content-Type": "text/plain"
    }
  }
}
```

# URL matching

```
{
  "request": {
    "method": "GET",
    "url": "/urlmatching"
  },
  "response": {
    "status": 200,
    "body": "URL matching"
  }
}
```

# Request header matching

```
{
  "request": {
    "method": "GET",
    "headers": {
      "headerName": {
        "equalTo": "headerValue"
      }
    }
  },
  "response": {
    "status": 200,
    "body": "Header matching"
  }
}
```

# Simulating a delay

```
{
  "request": {
    "method": "GET",
    "url": "/fixeddelay"
  },
  "response": {
    "status": 200,
    "fixedDelayMilliseconds": 2000
  }
}
```

# Returning a fault response

```
{
  "request": {
    "method": "GET",
    "url": "/badresponse"
  },
  "response": {
    "fault": "MALFORMED_RESPONSE_CHUNK"
  }
}
```

```

{
  "mappings": [
    {
      "scenarioName": "Order processing",
      "requiredScenarioState": "Started",
      "request": {
        "method": "GET",
        "url": "/order"
      },
      "response": {
        "status": 200,
        "body": "Your shopping cart is empty"
      }
    },
    {
      "scenarioName": "Order processing",
      "requiredScenarioState": "Started",
      "newScenarioState": "ORDER_PLACED",
      "request": {
        "method": "POST",
        "url": "/order",
        "bodyPatterns": [
          { "equalTo": "Ordering 1 item" }
        ]
      },
      "response": {
        "status": 200,
        "body": "There is 1 item in your shopping cart"
      }
    }
  ]
}

```

# Creating a stateful mock

```

    "response": {
      "status": 200,
      "body": "Item placed in shopping cart"
    }
  },
  {
    "scenarioName": "Order processing",
    "requiredScenarioState": "ORDER_PLACED",
    "request": {
      "method": "GET",
      "url": "/order"
    },
    "response": {
      "status": 200,
      "body": "There is 1 item in your shopping cart"
    }
  }
]
}

```



# Use response templating

```
{
  "request": {
    "url": "/template-http-method"
  },
  "response": {
    "status": 200,
    "body": "You used an HTTP {{request.method}}",
    "transformers": ["response-template"]
  }
}
```

# Use response templating

\_When sent this JSON  
request body:

```
{
  "book": {
    "author": "Ken Follett",
    "title": "Pillars of the Earth",
    "published": 2002
  }
}
```

\_This stub returns a response with body "Pillars of the Earth":

```
{
  "request": {
    "method": "POST",
    "urlPath": "/template-json-body"
  },
  "response": {
    "body": "{{jsonPath request.body '$.book.title'}}",
    "transformers": ["response-template"]
  }
}
```

# Using WireMock extensions

```
{
  "request" : {
    "customMatcher" : {
      "name" : "body-too-long",
      "parameters" : {
        "maxLength" : 2048
      }
    }
  },
  "response" : {
    "status" : 422
  }
}
```

Using a custom matcher

Specifying transformer parameters

```
{
  "request": {
    "method": "GET",
    "url": "/local-transform"
  },
  "response": {
    "status": 200,
    "body": "Original body",
    "transformers": ["my-transformer", "other-transformer"]
  }
}
```

Registering a local transformer

```
{
  "request" : {
    "url" : "/transform",
    "method" : "GET"
  },
  "response" : {
    "status" : 200,
    "transformerParameters" : {
      "paramName" : "value"
    }
  }
}
```

