Consumer Driven Contracts and Your Microservice Architecture

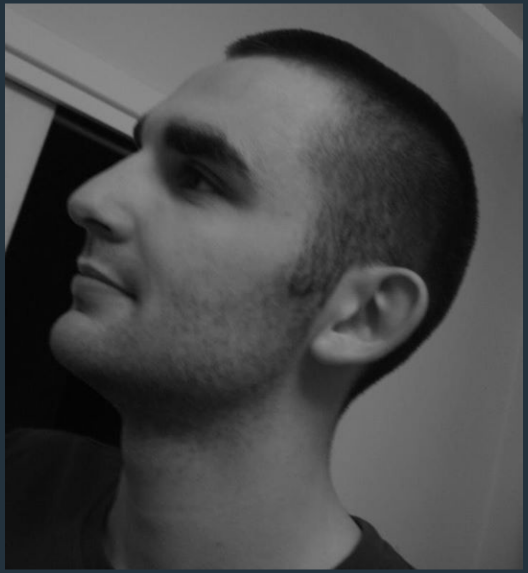
Marcin Grzejszczak, @mgrzejszczak

About me

Spring Cloud developer at Pivotal

Working mostly on

- Spring Cloud Sleuth
- Spring Cloud Contract
- Spring Cloud Pipelines



Twitter: @mgrzejszczak

Blog: http://toomuchcoding.com

Pivotal

Agenda

Introduction

Demo

Summary

What we will NOT be talking about?

spring

What we will NOT be talking about?

- Schema
- WSDL
- ESB
- XSD
- XSLT

THE IDEA OF SPRING CLOUD CONTRACT IS NOT TO INTRODUCE UNNECESSARY COUPLING OR REPLICATE OLD MISTAKES

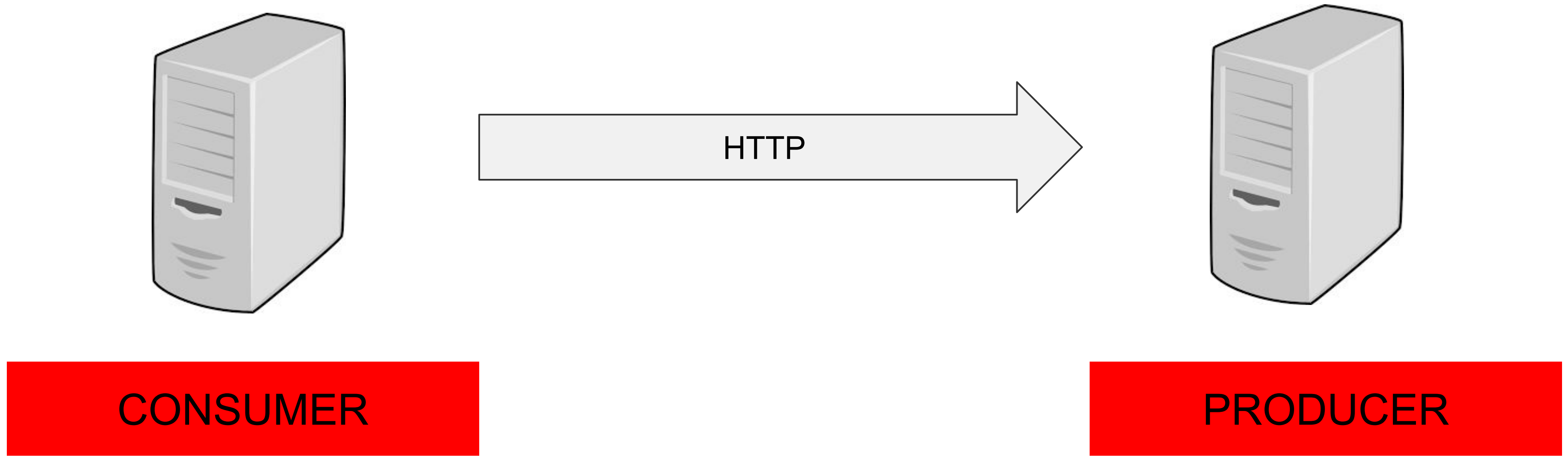
Introduction

- Producer
  - service that exposes an API
- Consumer
  - service that consumes the API of the producer
- Contract
  - agreement between producer and consumer how the API will look like
- Consumer Driven Contracts
  - approach where the consumer drives the changes of the API of the producer

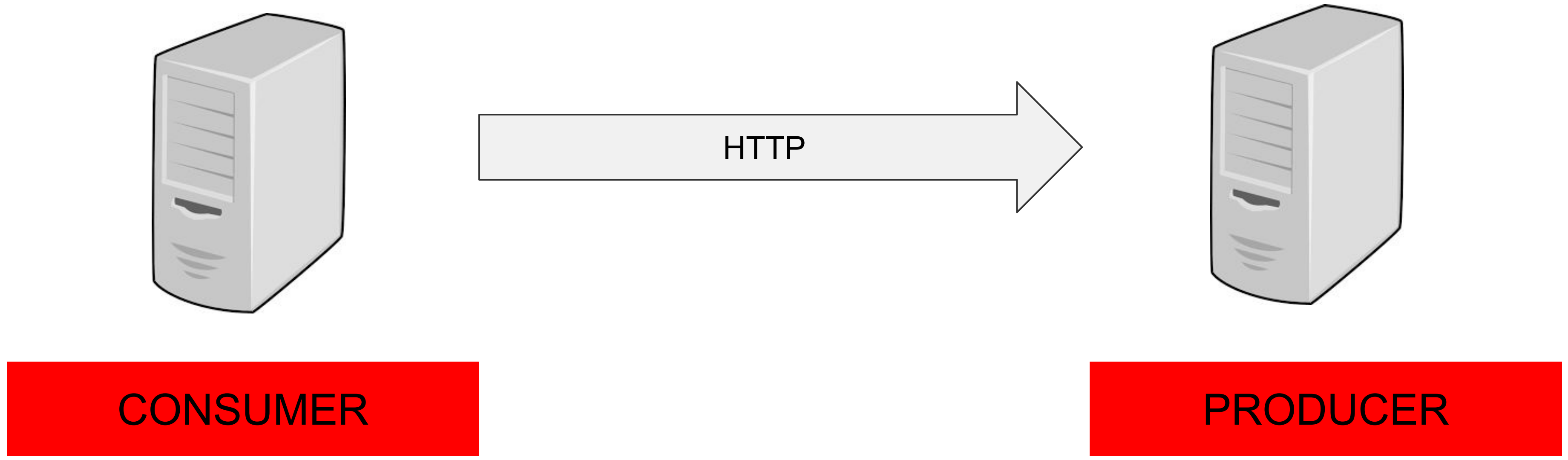
What problems are we trying to solve?

- Stub validity & reusability in the integration tests
- Nice API creation

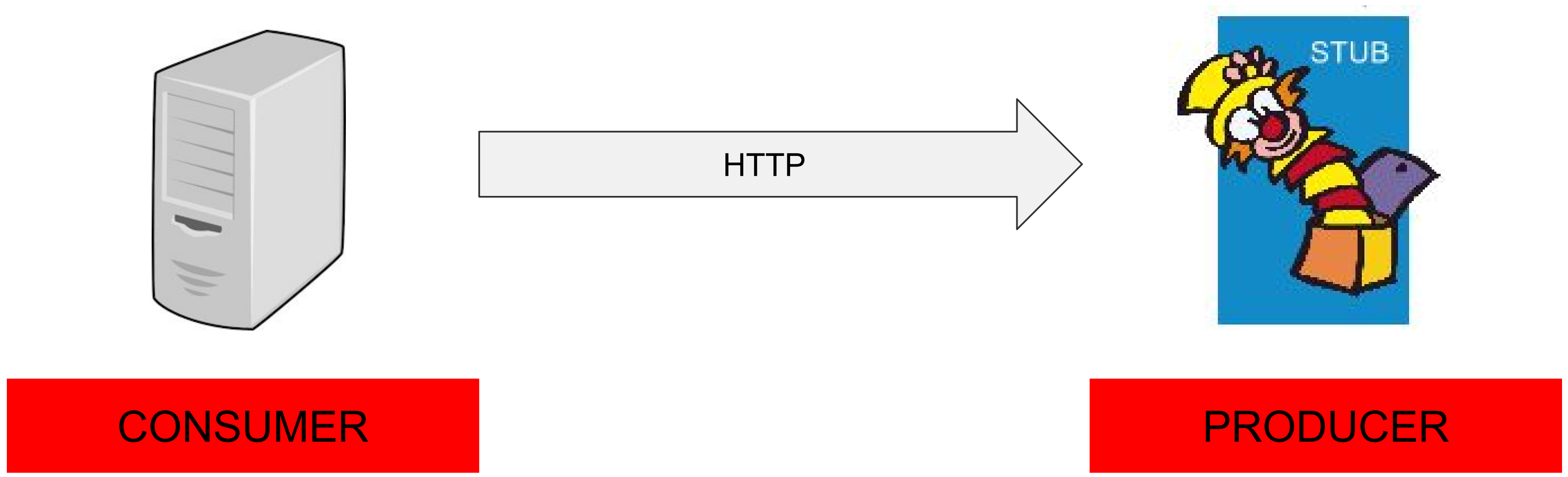
Typical situation



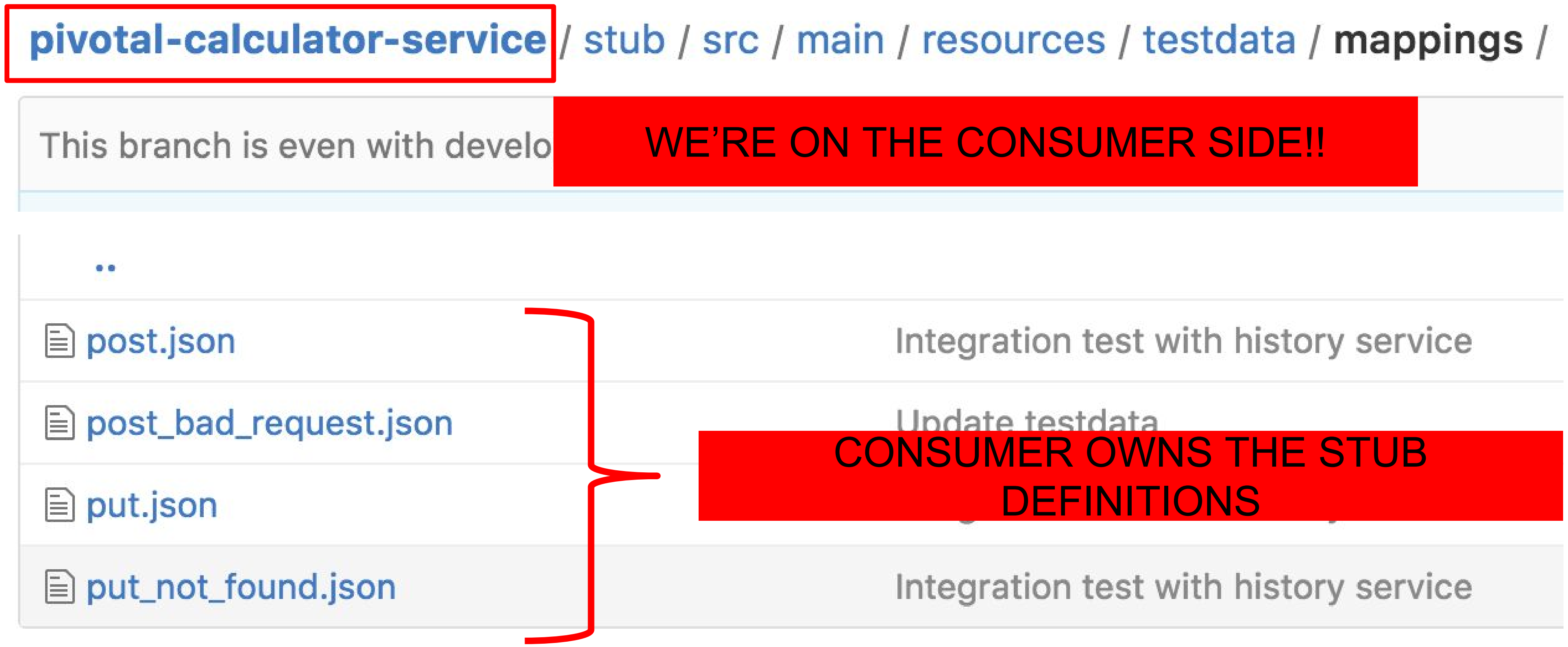
How to write a test for it?



How to write a test for it?



Stub validity & reusability



pivotal-calculator-service / stub / src / main / resources / testdata / **mappings** /

This branch is even with develo

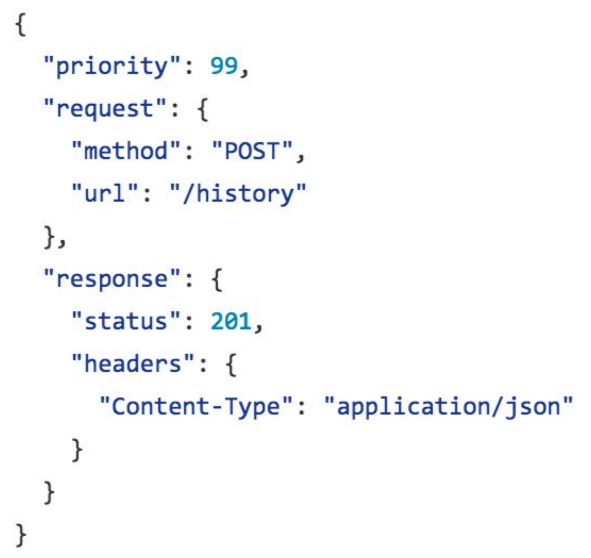
WE'RE ON THE CONSUMER SIDE!!

..

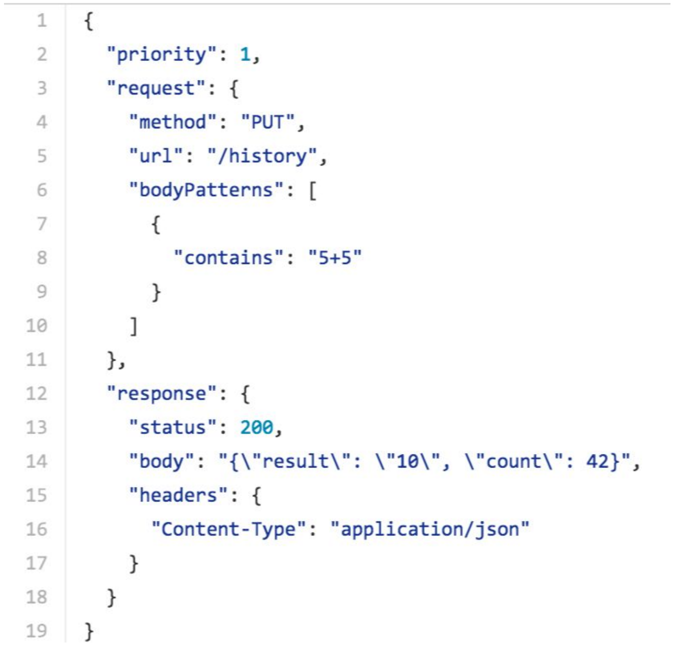
post.json	Integration test with history service
post_bad_request.json	Update testdata
put.json	
put_not_found.json	Integration test with history service

CONSUMER OWNS THE STUB DEFINITIONS

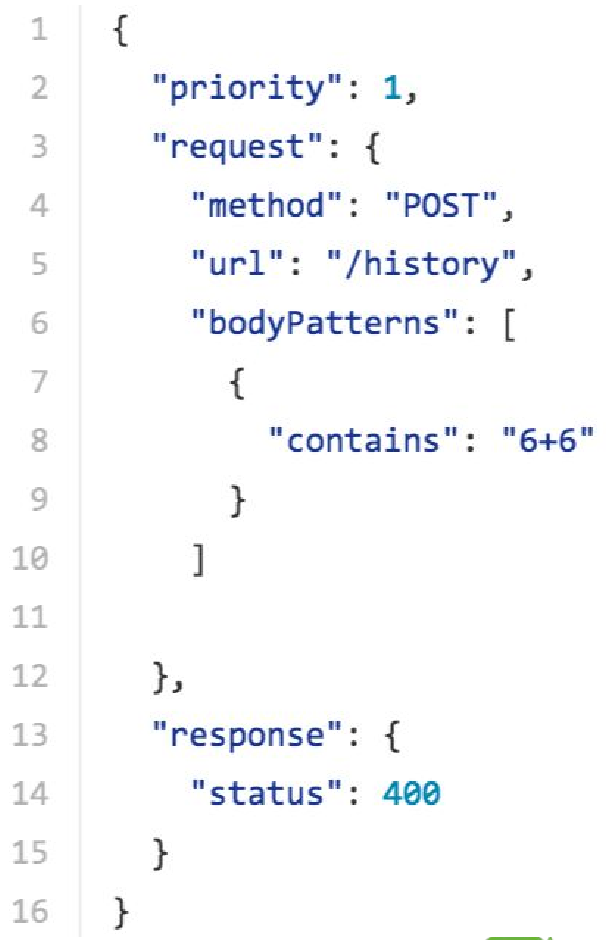
Stub validity & reusability



```
{
  "priority": 99,
  "request": {
    "method": "POST",
    "url": "/history"
  },
  "response": {
    "status": 201,
    "headers": {
      "Content-Type": "application/json"
    }
  }
}
```



```
{
  "priority": 1,
  "request": {
    "method": "PUT",
    "url": "/history",
    "bodyPatterns": [
      {
        "contains": "5+5"
      }
    ]
  },
  "response": {
    "status": 200,
    "body": "{\"result\": \"10\", \"count\": 42}",
    "headers": {
      "Content-Type": "application/json"
    }
  }
}
```

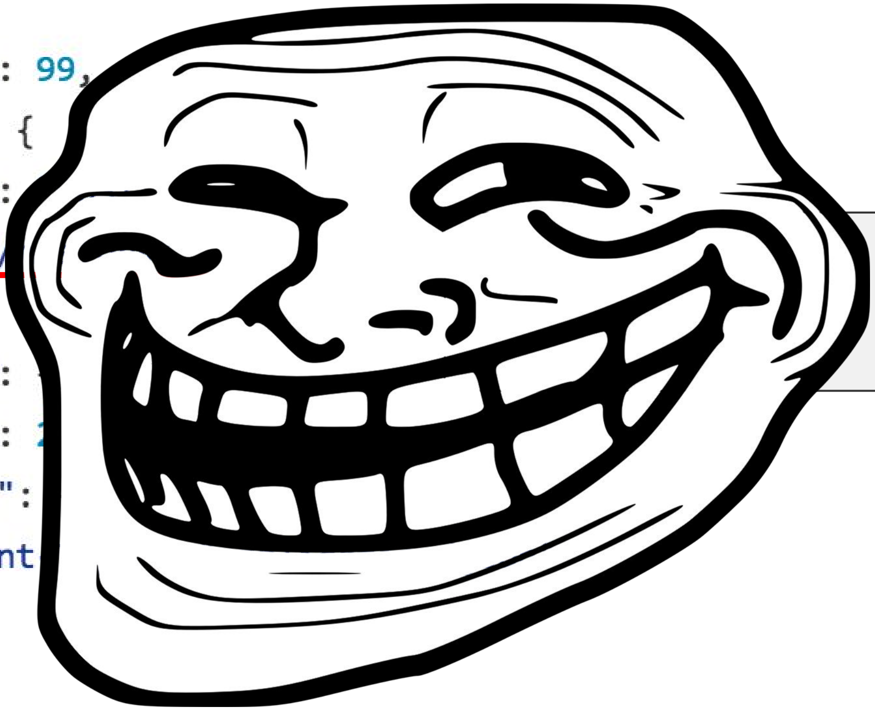


```
{
  "priority": 1,
  "request": {
    "method": "POST",
    "url": "/history",
    "bodyPatterns": [
      {
        "contains": "6+6"
      }
    ]

  },
  "response": {
    "status": 400
  }
}
```

Stub validity & reusability

```
{
  "priority": 99,
  "request": {
    "method":
    "url": "/
  },
  "response":
    "status":
    "headers":
      "Content
    }
  }
}
```



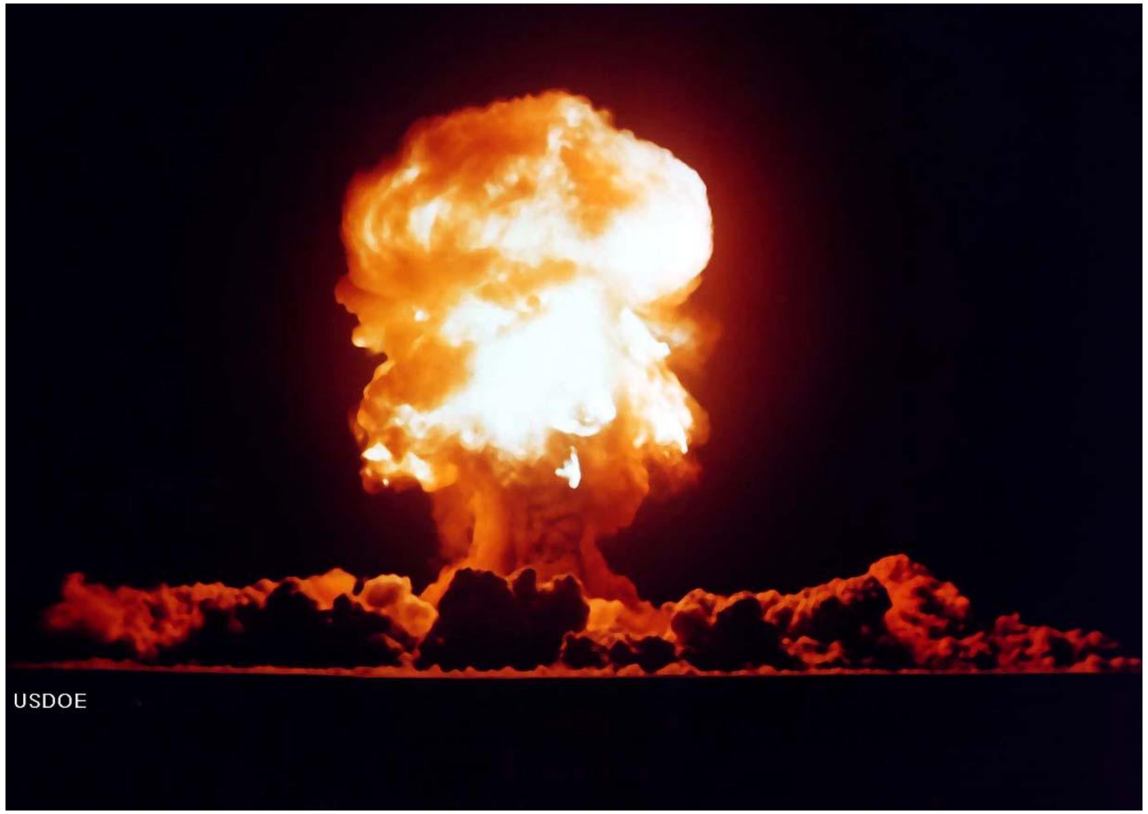
```
{
  "priority": 99,
  "request": {
    "method": "POST",
    "url": "/someNonExistantUrl"
  },
  "response": {
    "status": 201,
    "headers": {
      "Content-Type": "application/json"
    }
  }
}
```

So now what?

- My tests fail cause I’m shooting a request at a non existant URL
- I rewrote all my code to work with the new URL
- The unit and integration tests pass
- Now we deploy to an environment

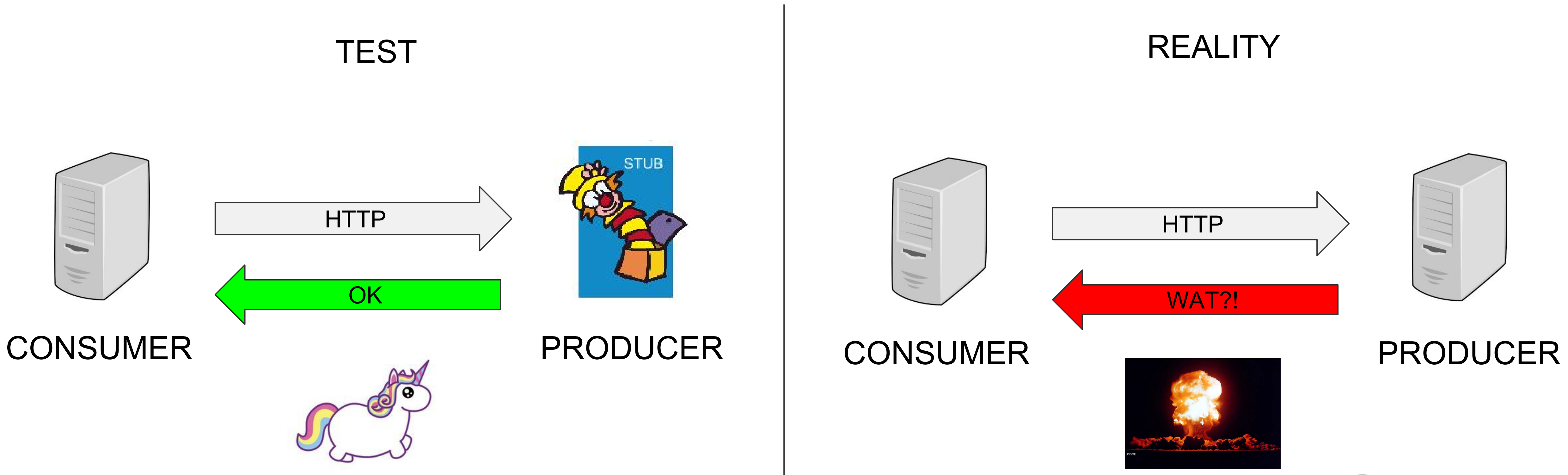
  where real integrations will take place...

Ooops...



What went wrong?

Stubs that were used in the build phase have nothing to do with the real API!!



Stub validity & reusability

- Stubs reside with the consumer
- Consumer controls the stubbing process
- How are you sure that the stubs are valid?
- What if other teams want to reuse those stubs?

Nice API creation

- It’s the consumer that uses the API
- Consumers should take part in the creation of the API of the producer
- The producer’s API change should be driven by consumers

Nice API creation - no cooperation results

Nice API creation - no cooperation results

Demo

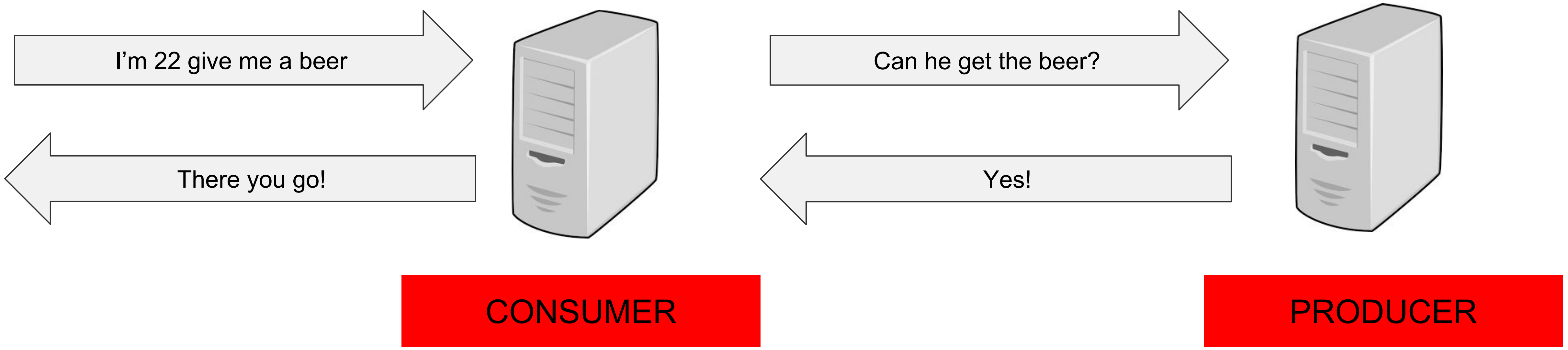
Potential answer

Spring Cloud Contract

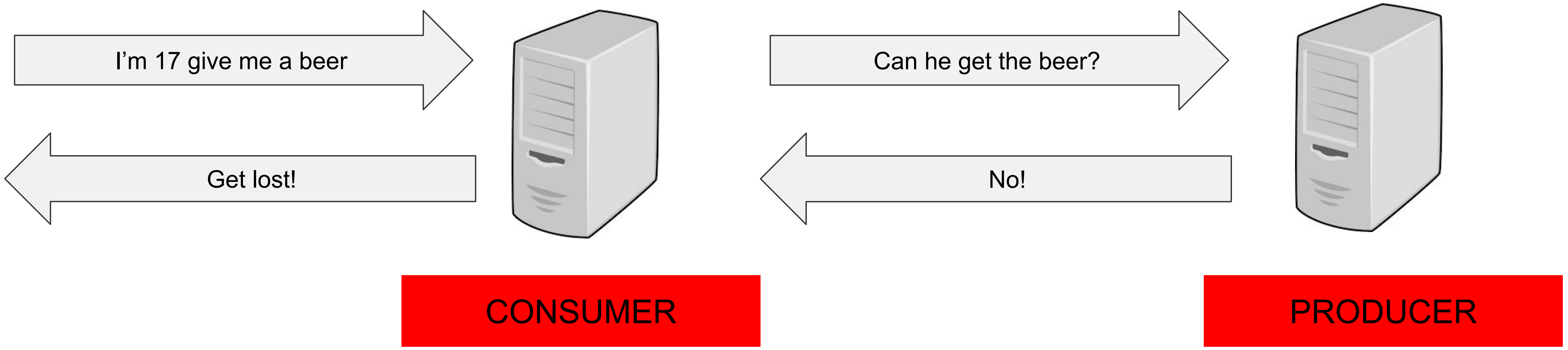
What are we going to code?

- Consumer
  - service that gets beer requests
  - has to ask another service if the client can get the beer
- Producer
  - service that checks if the client is old enough to buy beer
- Feature
  - if the user is too young - the beer will not be sold
  - otherwise the beer will be granted

What are we going to code?



What are we going to code?



What are we going to code?

CONSUMER

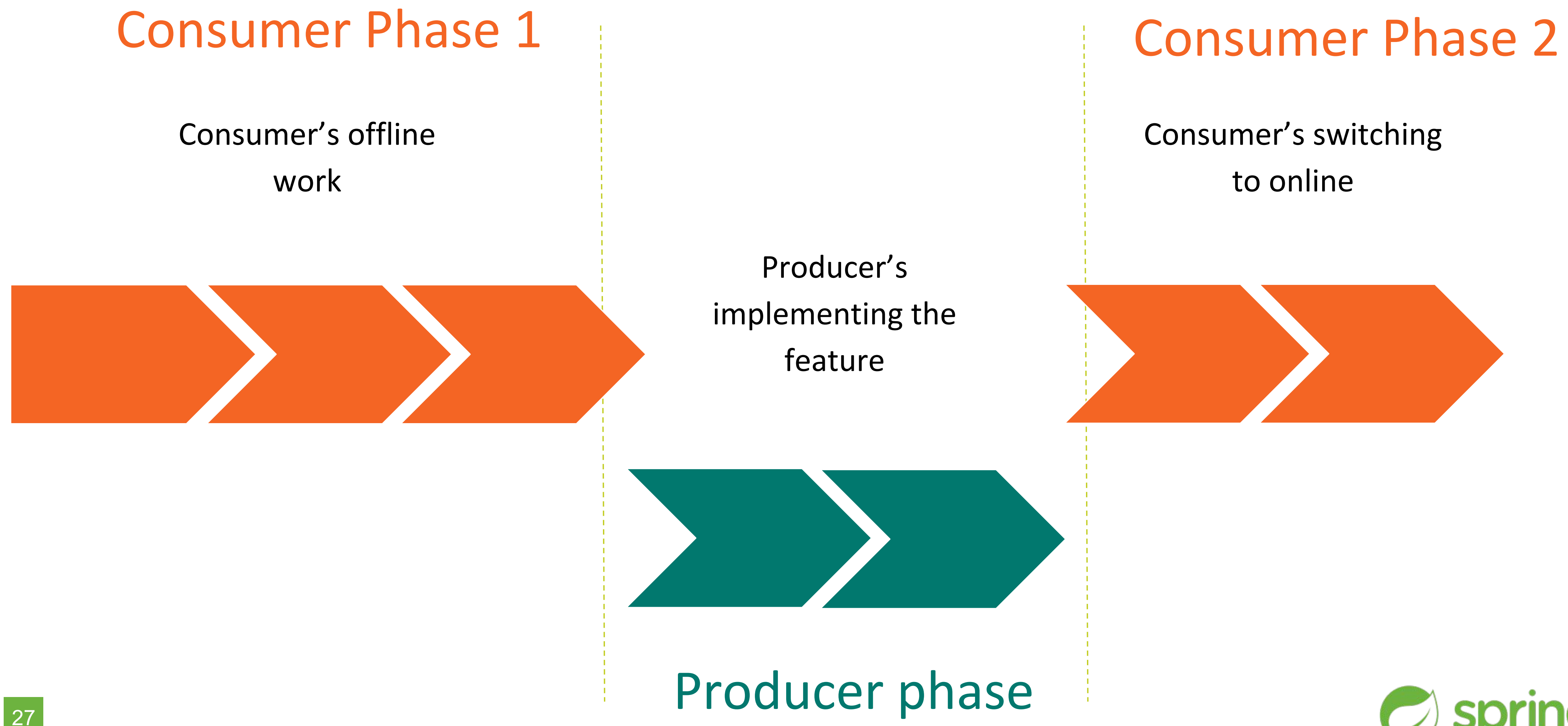
BLACK TERMINAL
BLACK IDE

PRODUCER

WHITE TERMINAL
WHITE IDE

Demo

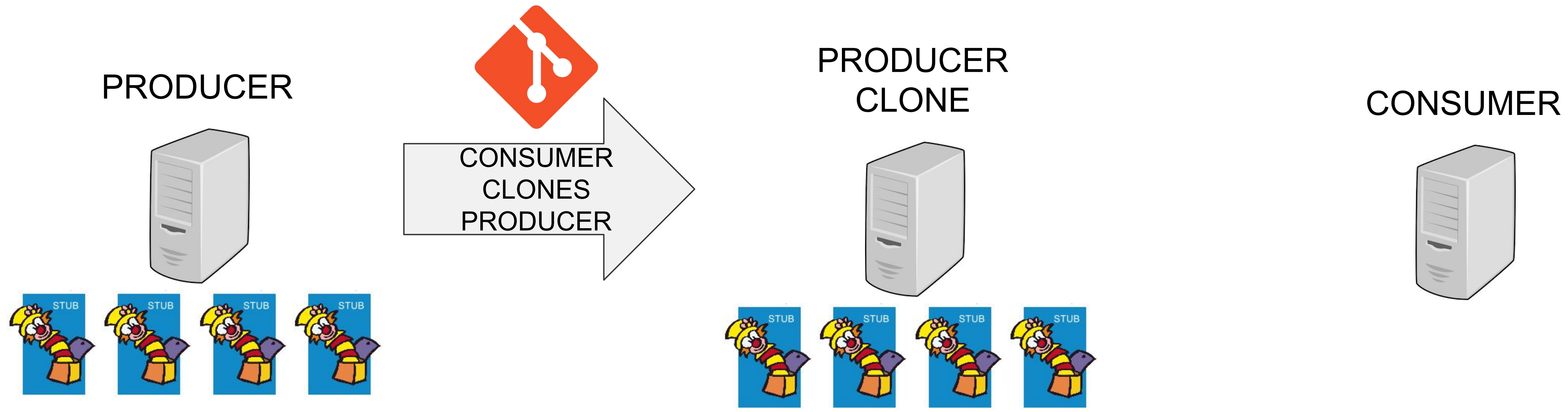
Summary


Consumer Phase 1
Consumer's offline work
Producer's implementing the feature
Consumer Phase 2
Consumer's switching to online
Producer phase

Consumer flow 1

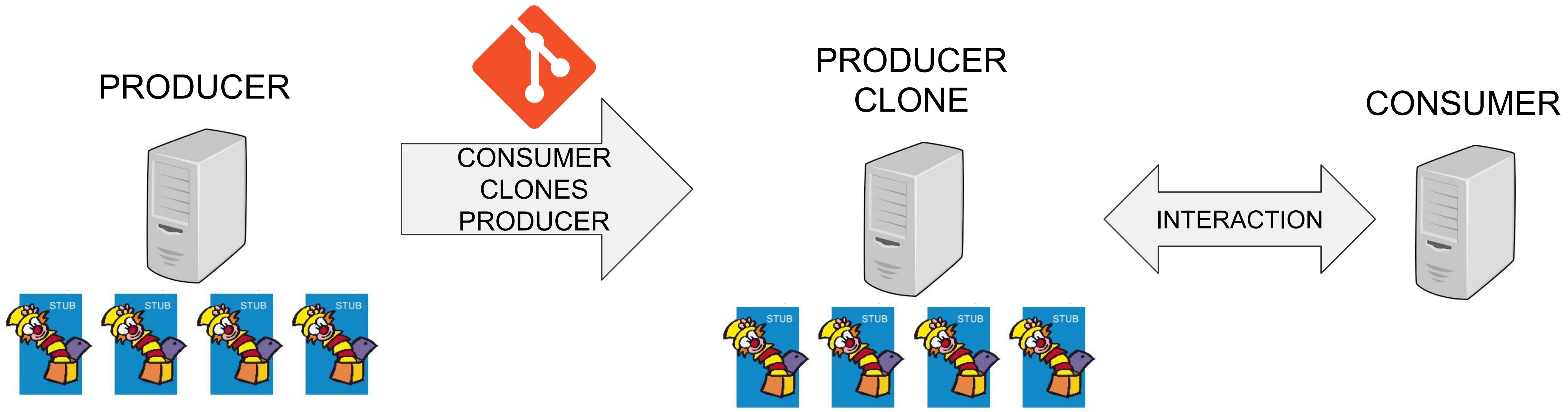
PRODUCER
CONSUMER
STUB
STUB
STUB
STUB

Consumer flow 1

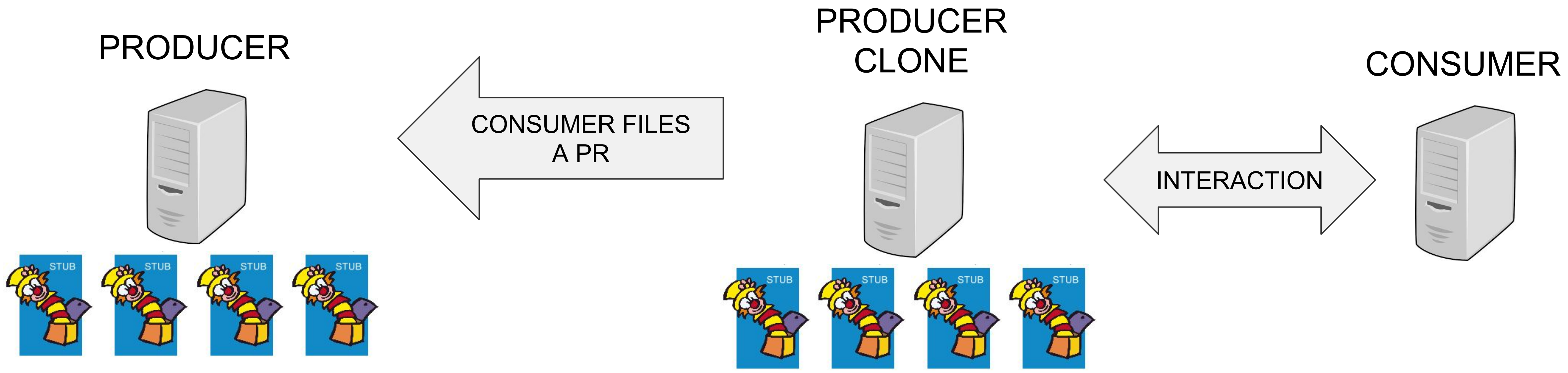


Summary

Consumer flow 1



Consumer flow 1



Consumer flow 1

starts TDD - writes the test for the feature

clones producer code to change the API locally

in the cloned producer code converts contracts into stubs and installs them locally

in the consumer code turns Stub Runner to offline mode

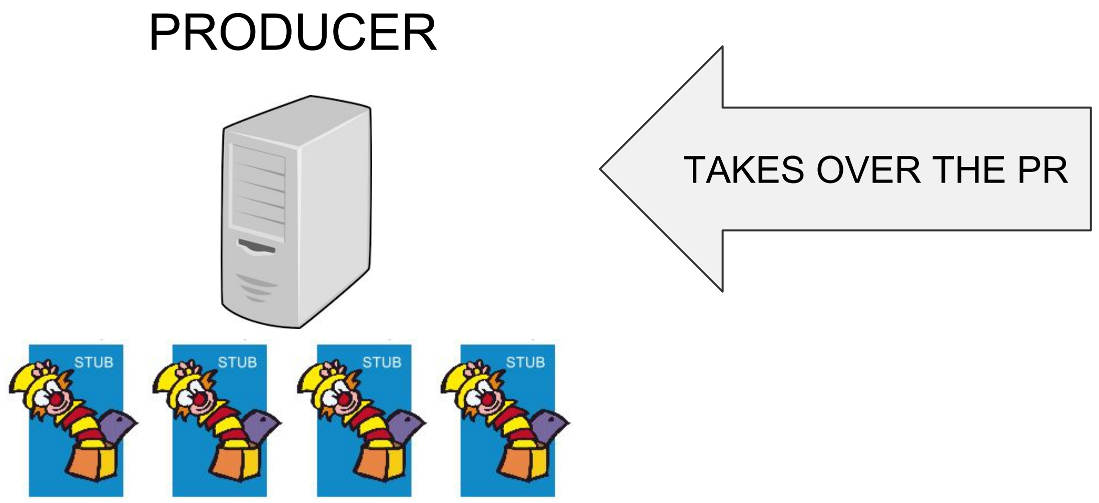
configures Stub Runner to download stubs of the producer

red - green - refactor on the API and tests

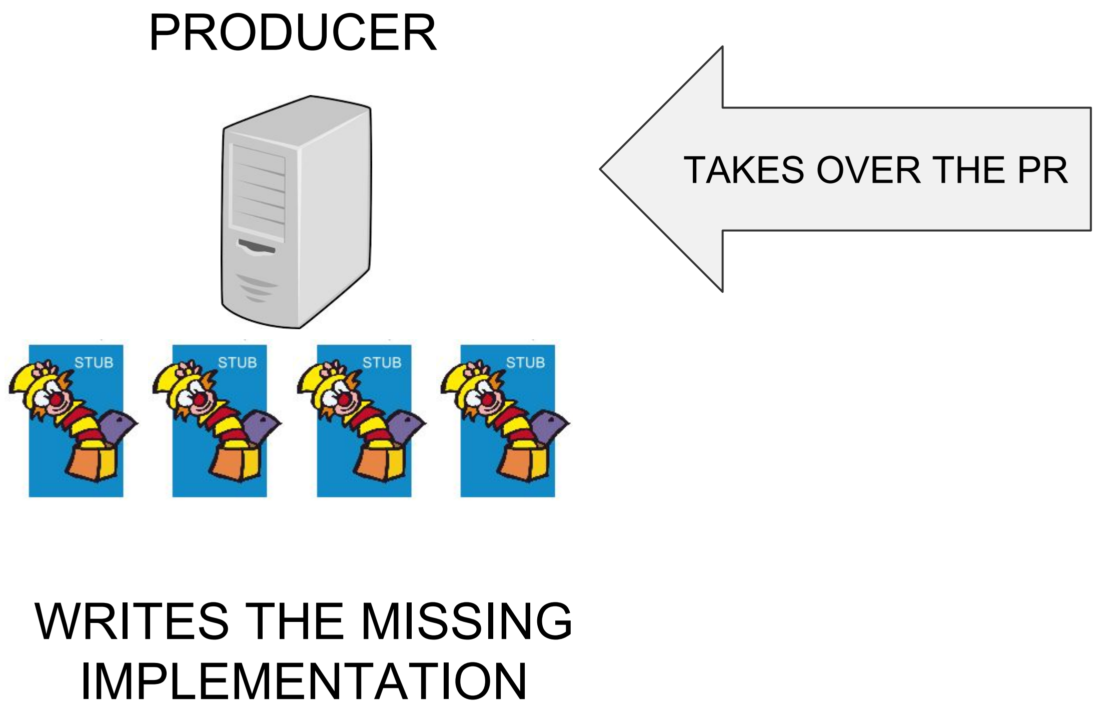
repeats the process until the tests are green and API acceptable

files a PR to the producer with the contract proposal

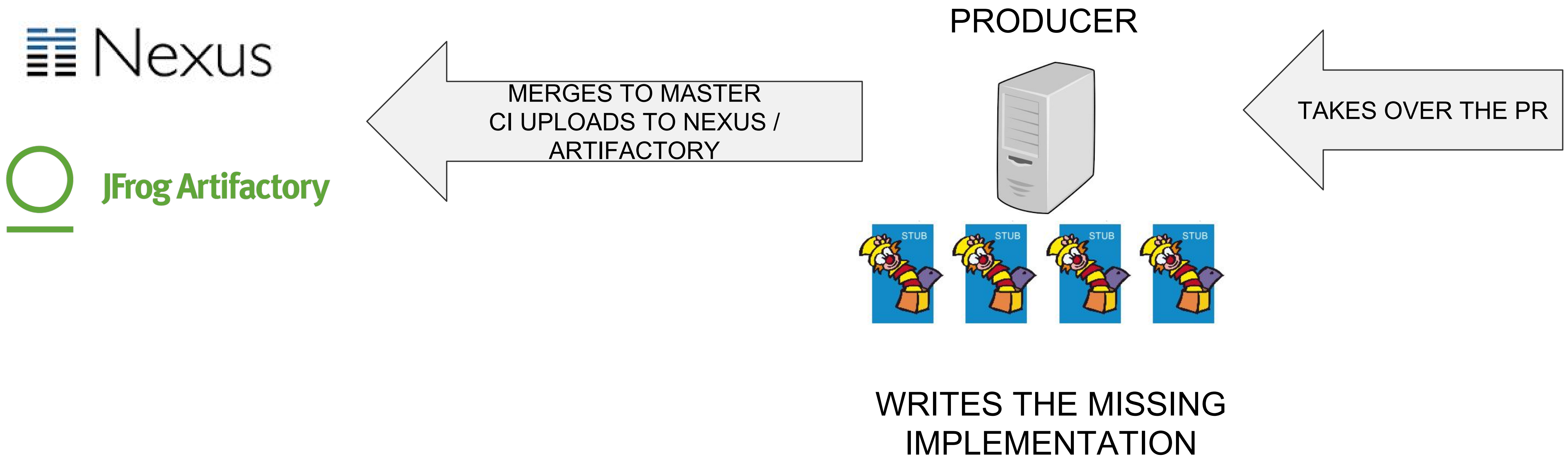
Producer flow



Producer flow



Producer flow



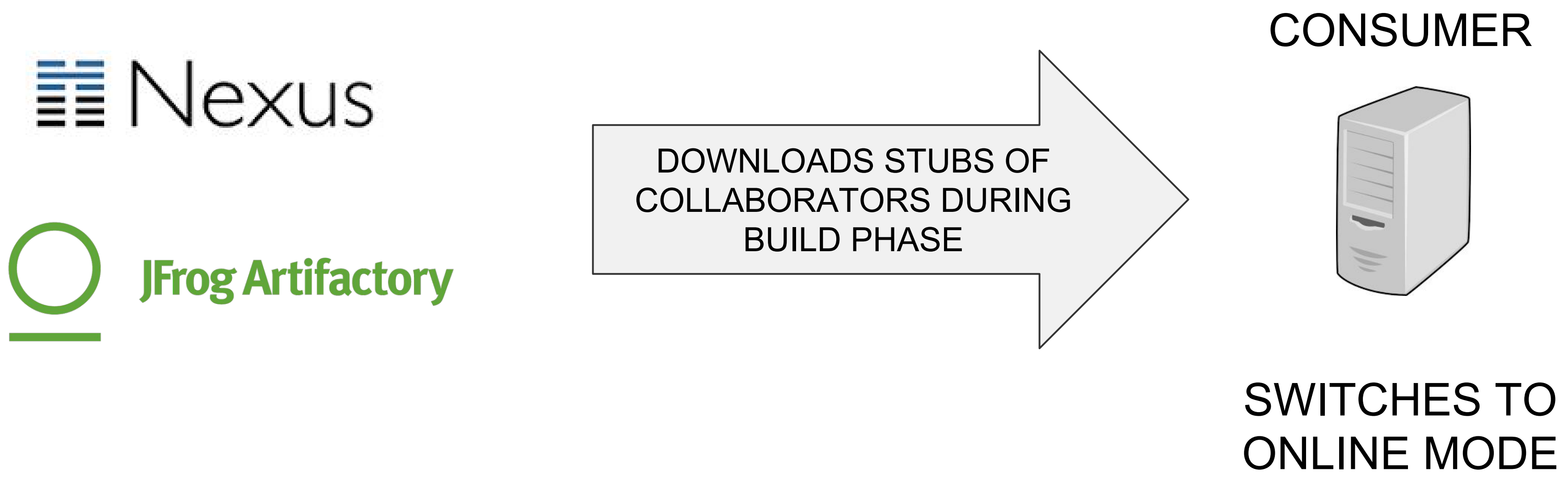
Producer flow

takes over the PR

writes the missing implementation that will make the autogenerated tests pass

merges PR and deploys the JARs with the app and the stubs

Consumer flow 2



Consumer flow 2

switches off the Stub Runner's offline mode once the producer uploads the stubs

configure Stub Runner by providing the URL to a repo with stubs

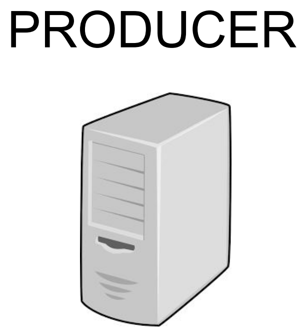
will have its test broken if the producer makes any breaking changes of the API

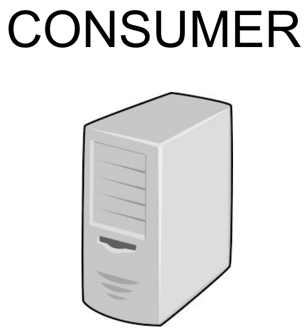
I don’t want to keep my contracts with the producer

Common repo with contracts - consumer

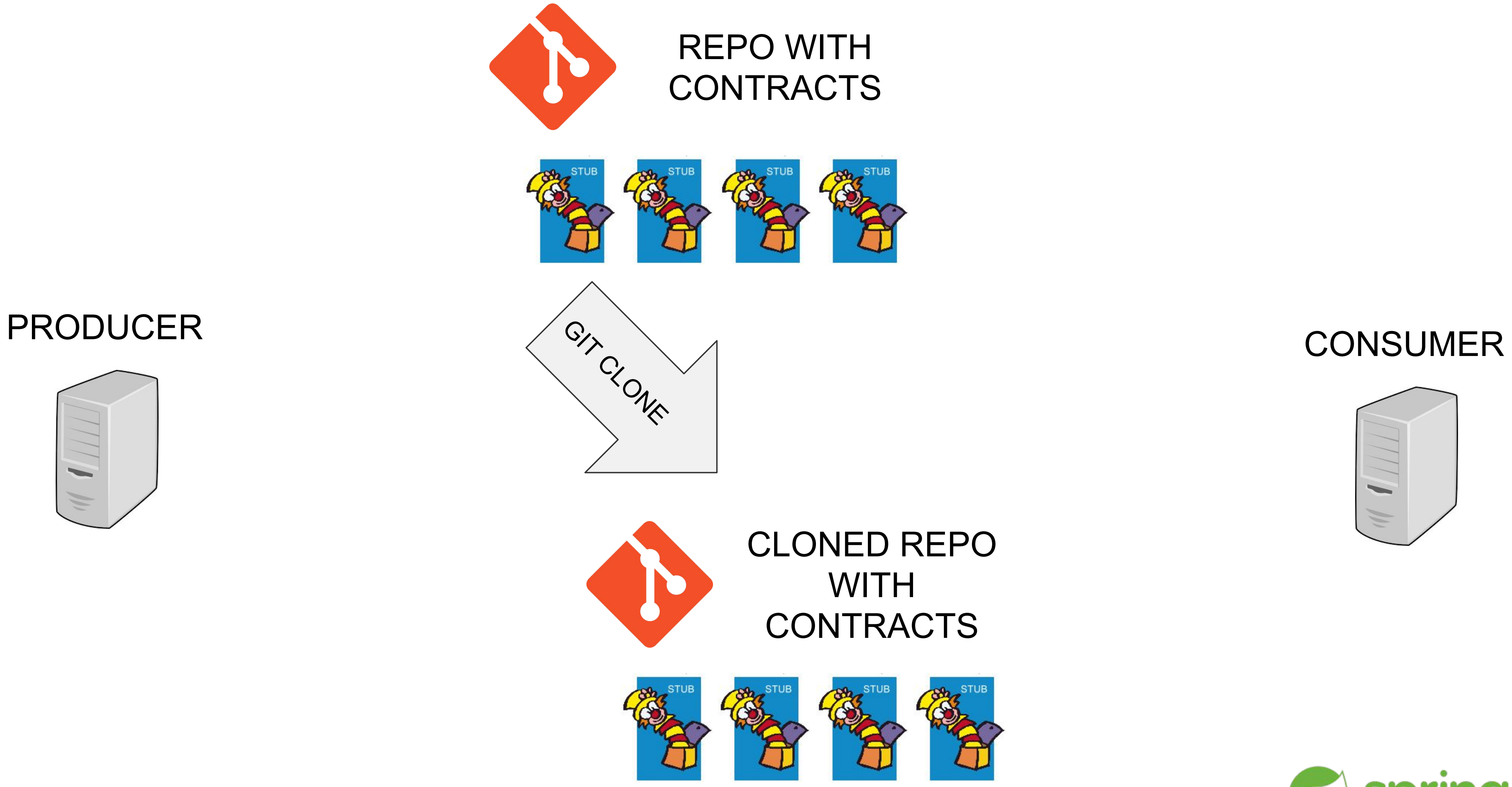
Common repo with contracts - consumer

REPO WITH
CONTRACTS
STUB
STUB
STUB
STUB

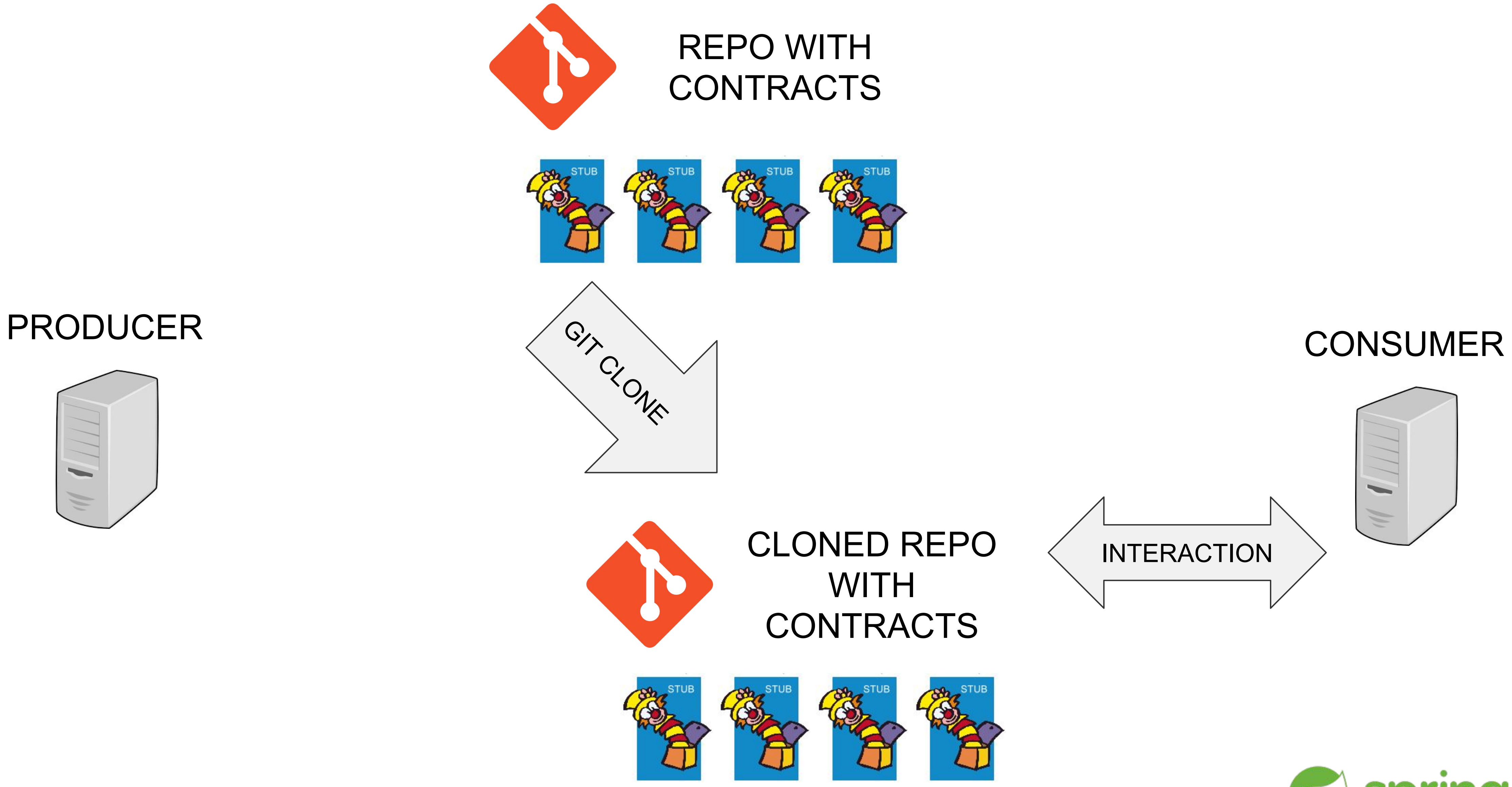




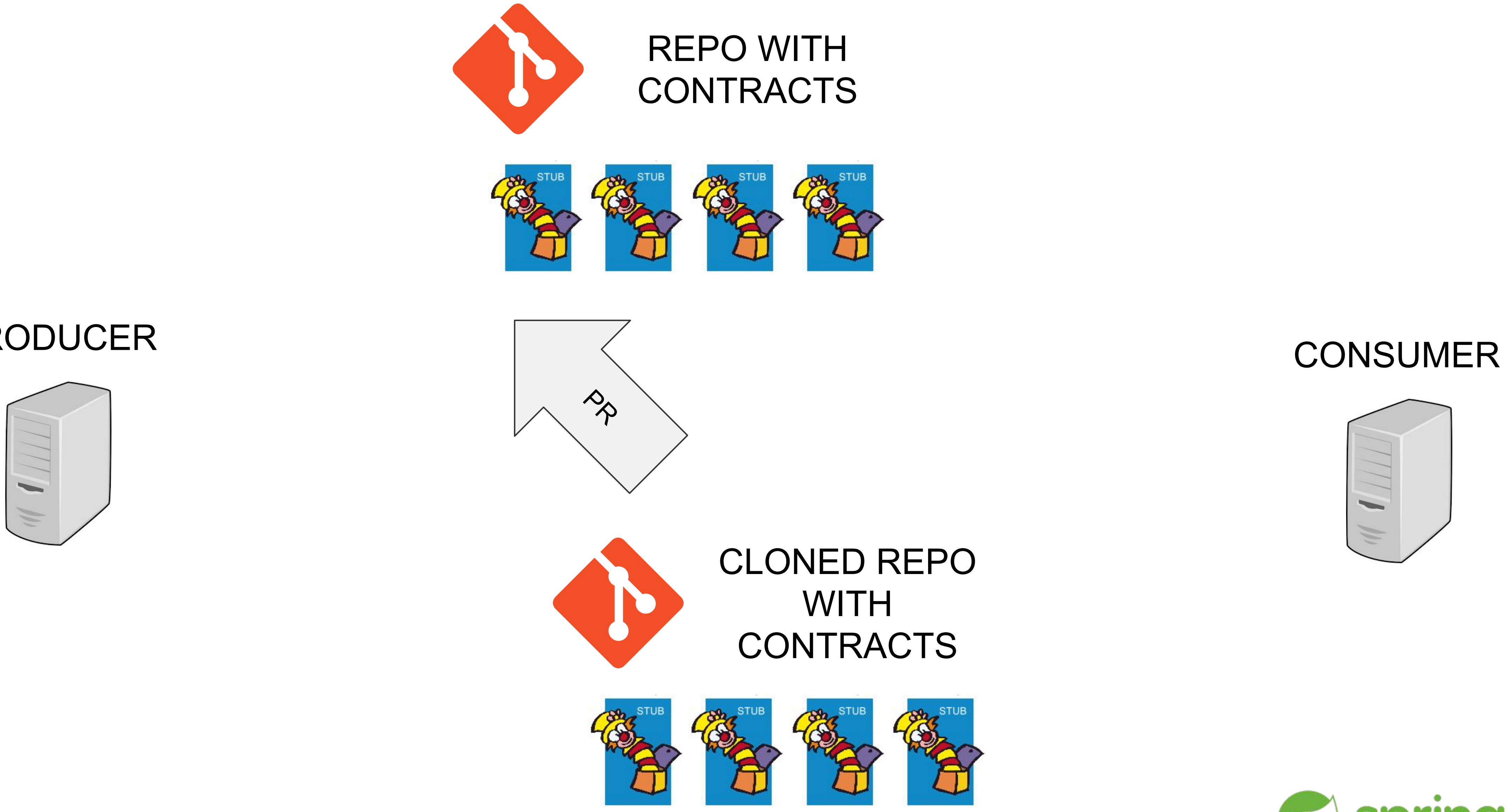
Common repo with contracts - consumer



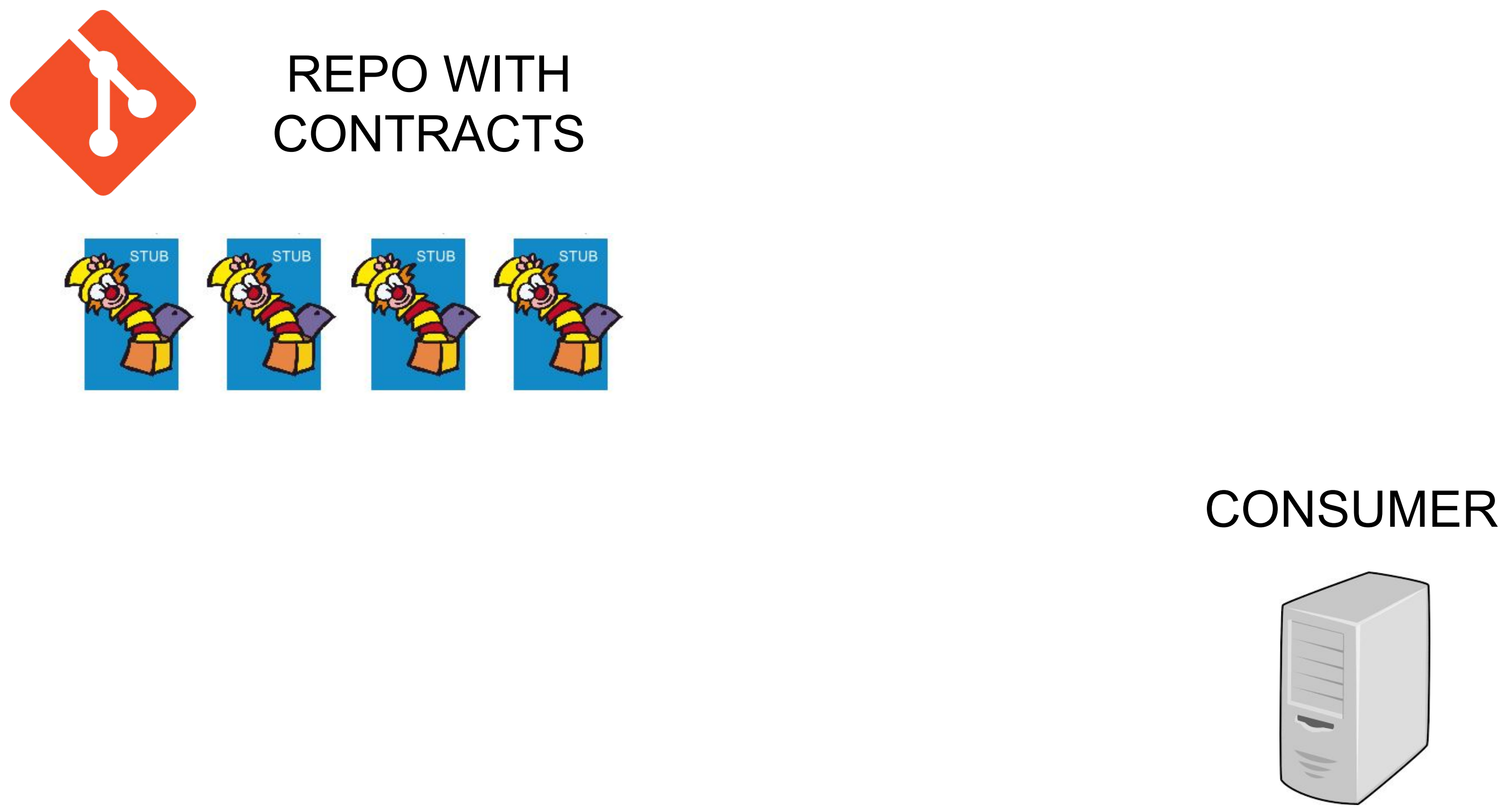
Common repo with contracts - consumer



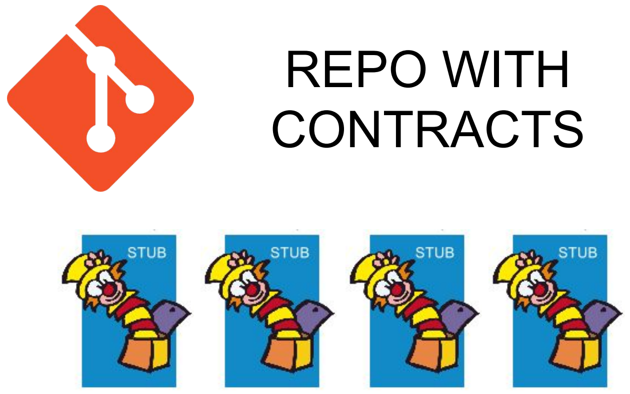
Common repo with contracts - consumer

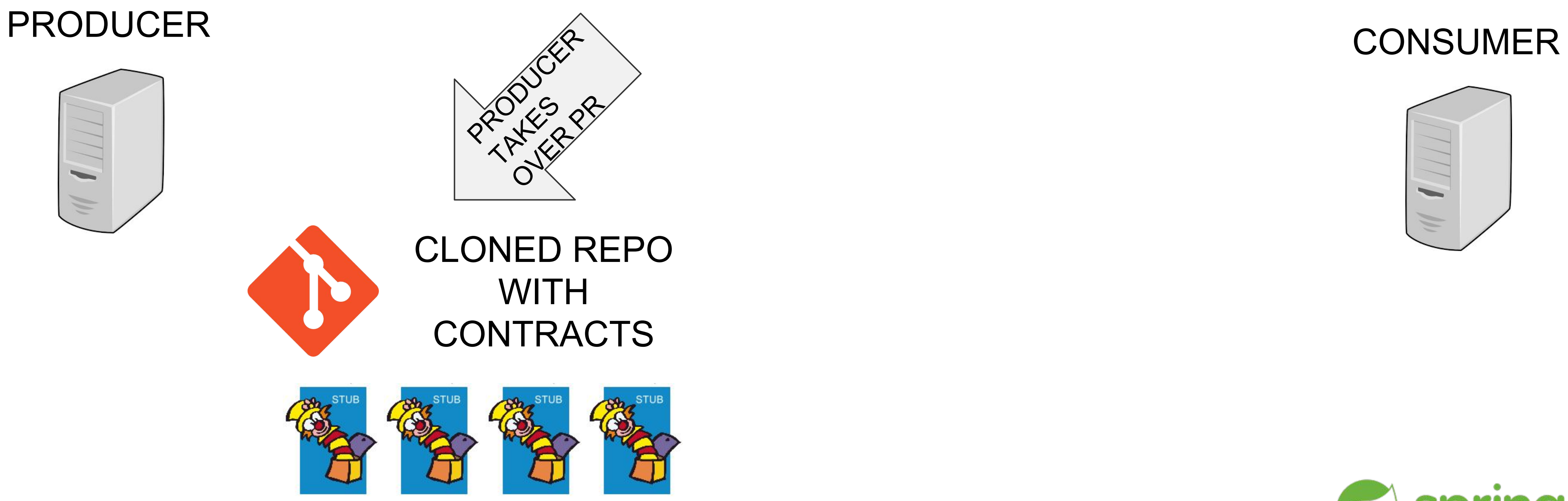


Common repo with contracts - producer

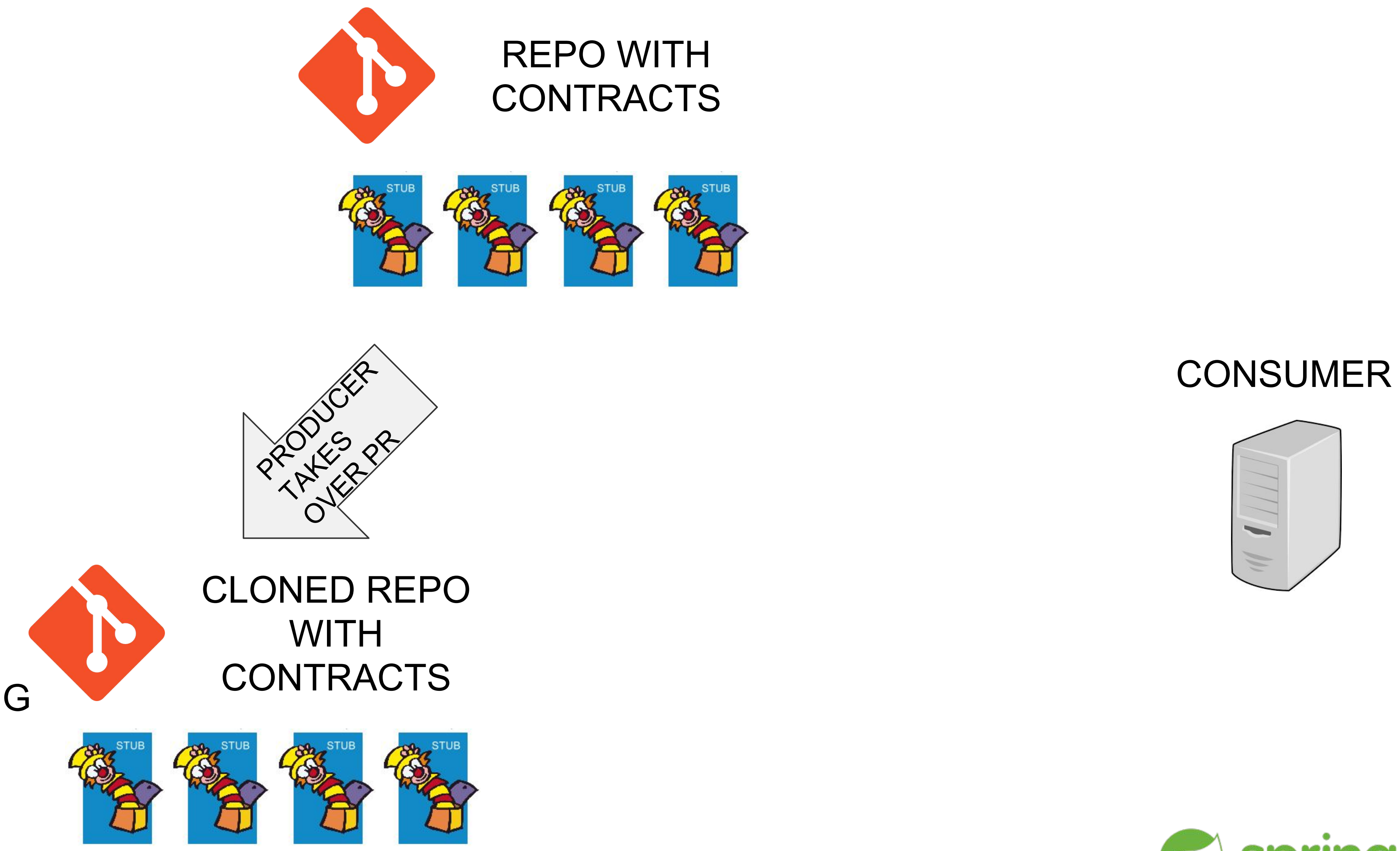


Common repo with contracts - producer

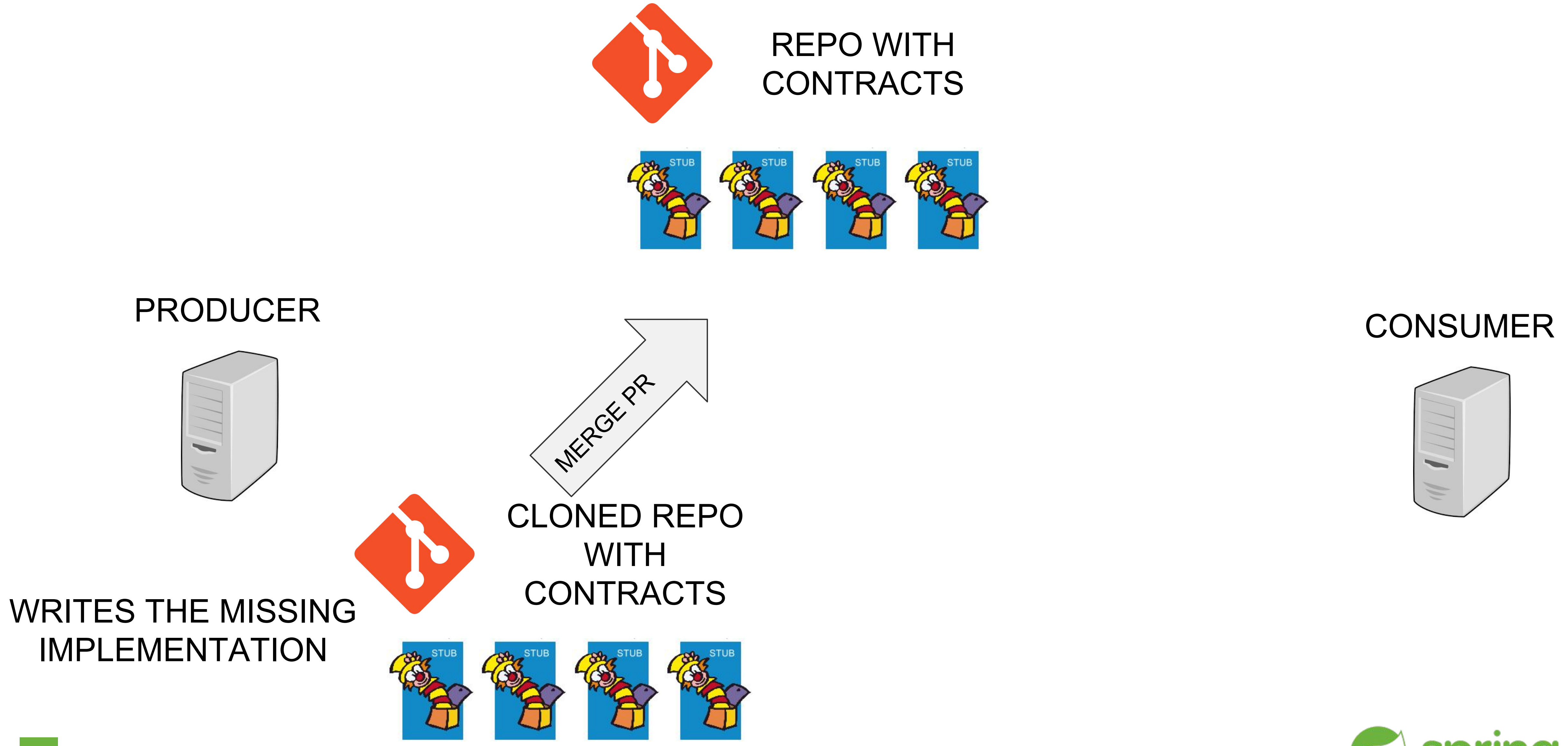




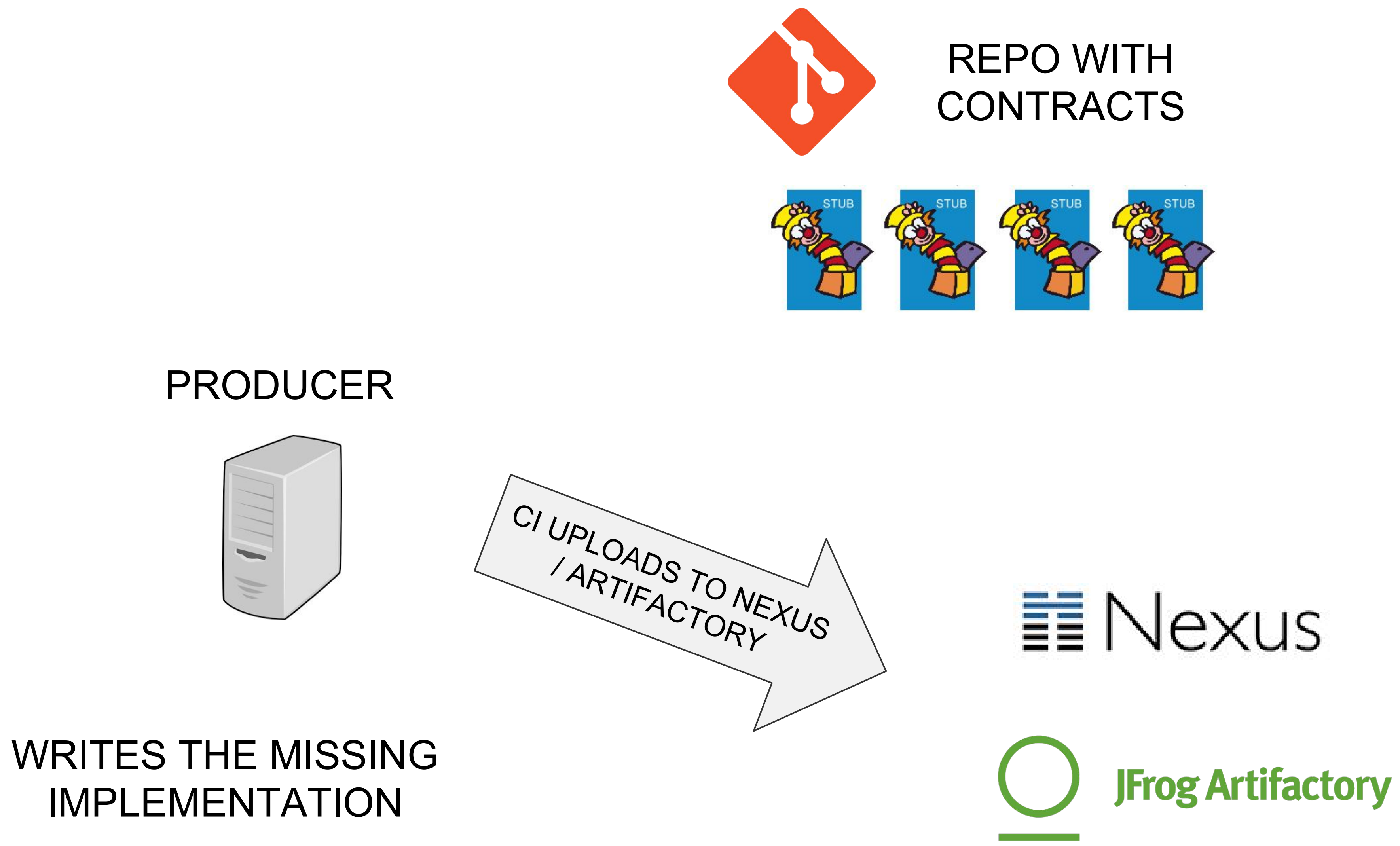
Common repo with contracts - producer



Common repo with contracts - producer



Common repo with contracts - producer



I don’t want to keep my contracts with the producer

Sometimes, due to e.g. security reasons, you don’t have access to the producer’s code.

Then you can use a shared repo with contracts:

- that repo is built by your CI and produces a JAR with contract defintions
- the producer downloads the JAR with contracts and finds his contract definitions either via convention or by provided property
- the producer creates tests and stubs from aforementioned contracts
- the producer deploys a fat JAR and JAR with stubs once the autogenerated tests pass
- the consumer via Stub Runner downloads the stubs from Nexus or can manually create stubs from the shared repo for offline work

Summary

With Spring Cloud Contract and Consumer Driven Contracts:

- we’ve created an API that suits the consumer and the producer
- expectations were defined by readable contracts
- expectations were tested against the producer
- producer stubs can be reused by consumers
- starting and setting stubs is fully automated

Summary

Why use Spring Cloud Contract Verifier?

- Possibility to do CDC with messaging
- Clear and easy to use, statically typed DSL
- Automatic generation of tests from the defined Contract
- Stub Runner functionality - the stubs are automatically downloaded at runtime from Nexus / Artifactory
- Spring Cloud integration - no discovery service is needed for integration tests
- Integration with Cloud Foundry - Stub Runner Boot

QUESTIONS

Learn More. Stay Connected.

- Read the docs
- Check the samples
- Talk to us on Gitter

Twitter: twitter.com/springcentral

YouTube: spring.io/video

LinkedIn: spring.io/linkedin

Google Plus: spring.io/gplus

Pivotal®

Transforming How The World Builds Software

mgrzejszczak