

Software testing in CI/CD pipelines

Sven Hettwer
ConSol Software GmbH



Buzzword Bingo - Rules

1

Take a Bingo card

2

Every time you recognize
a **highlighted buzzword**,
mark it on your card

3

If you completed a row,
column or diagonal,
come over and choose a
Giveaway.

Agenda

- Concepts - Software testing in [CI/CD pipelines](#)
- Our goals for today
- Sample app
- Container platform
- CI/CD pipelines
- Integration testing
- End-2-End testing
- What's next?
- Recap



Concepts – Software testing

- Software tests influencing your company
- Software testing has various aspects
 - Technical aspect
 - Business aspect



Concepts – Software testing - Technical perspective

- Ensures that your software works as expected
- Provides information about the quality of the software
- Helps identifying bugs
- Eases software changes

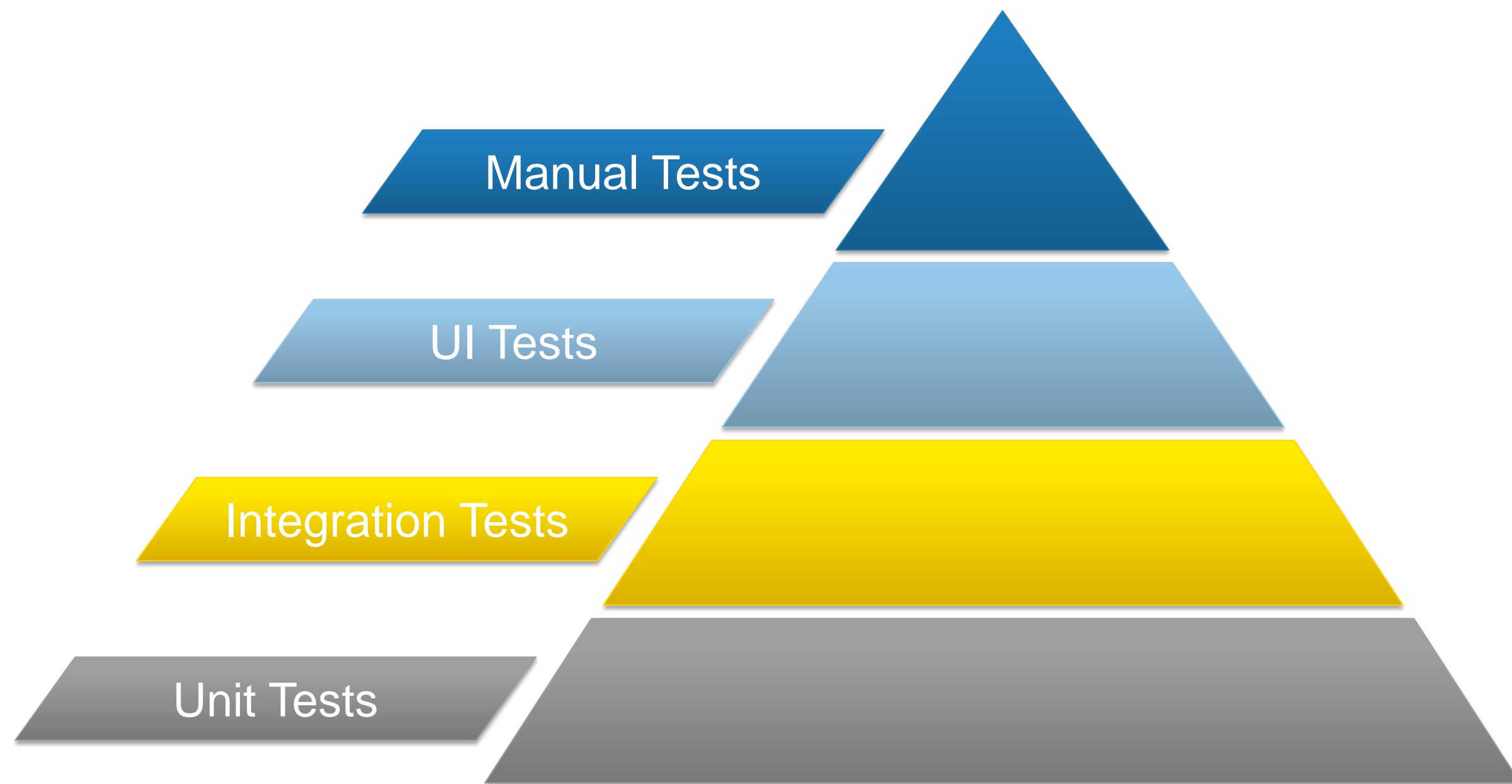


Concepts – Software testing – Business perspective

- Ensures stable revenue streams
- Increases customer satisfaction
- Lowers costs for development

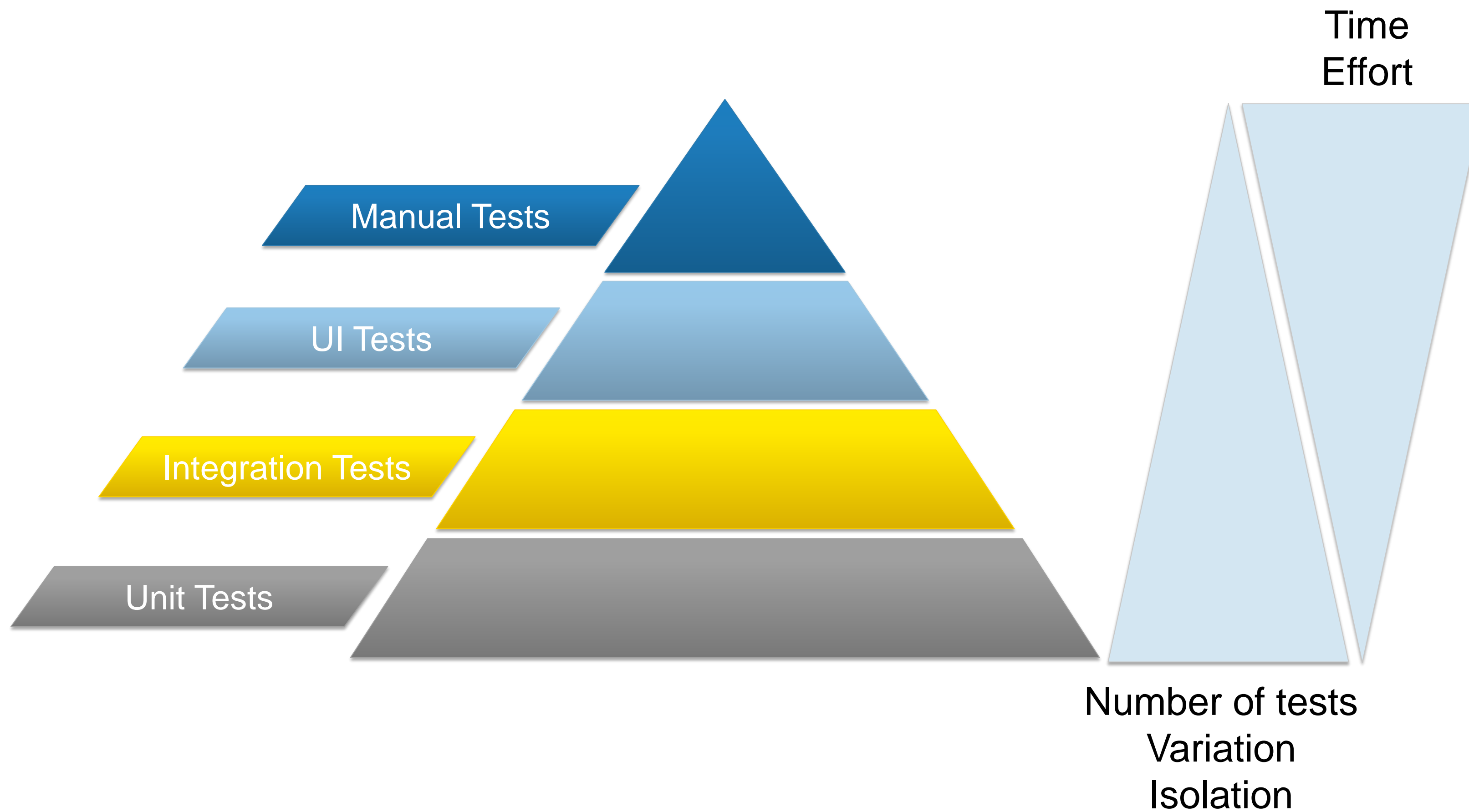


Concepts – Software testing

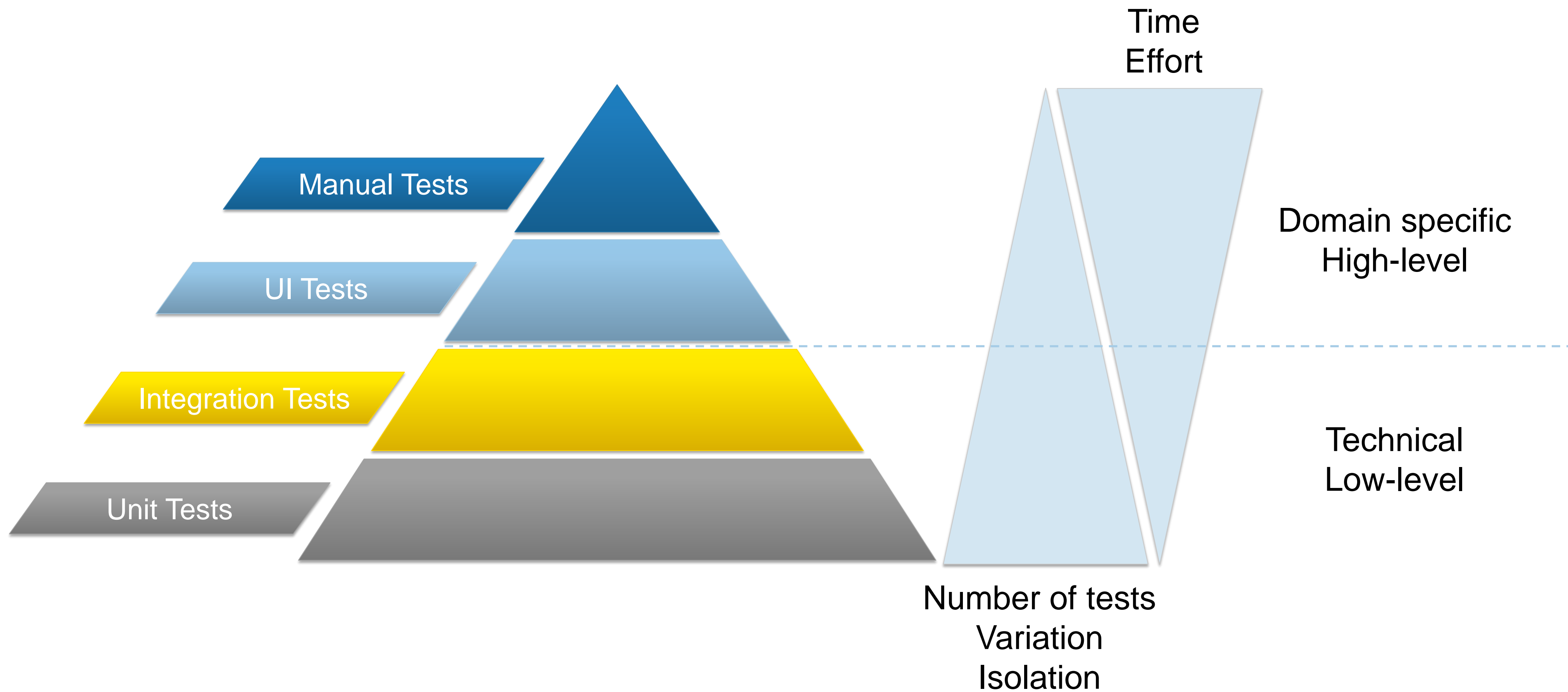


Test Pyramid

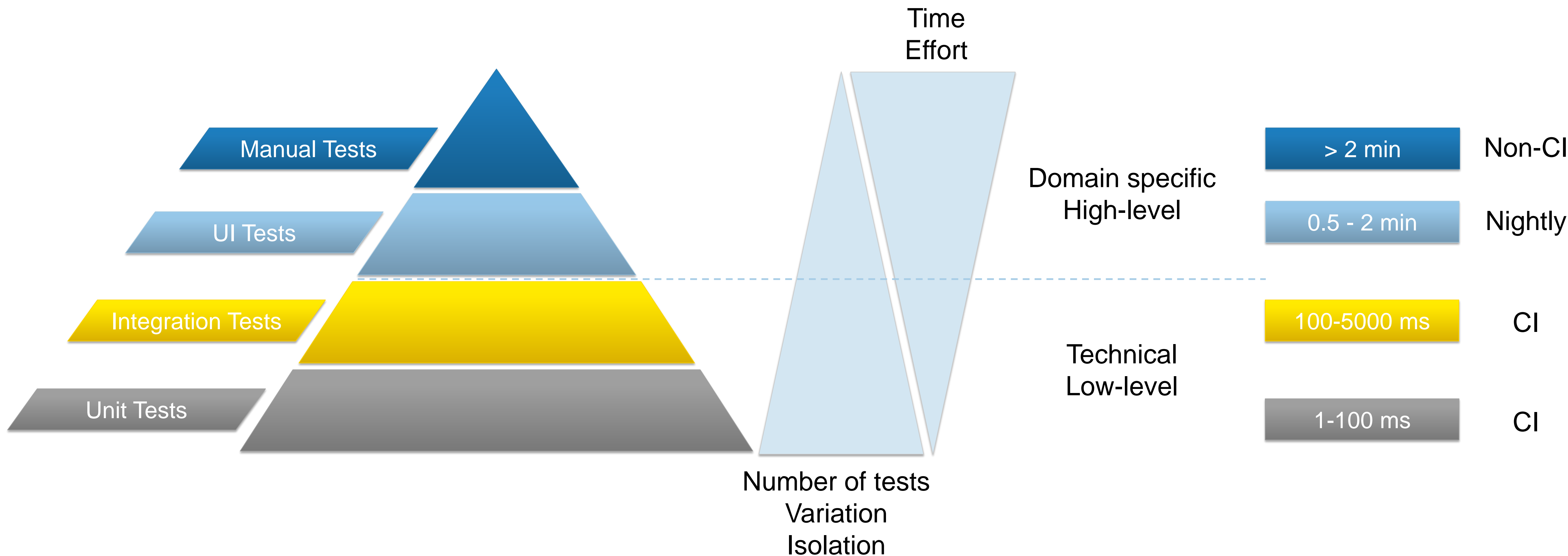
Concepts – Software testing



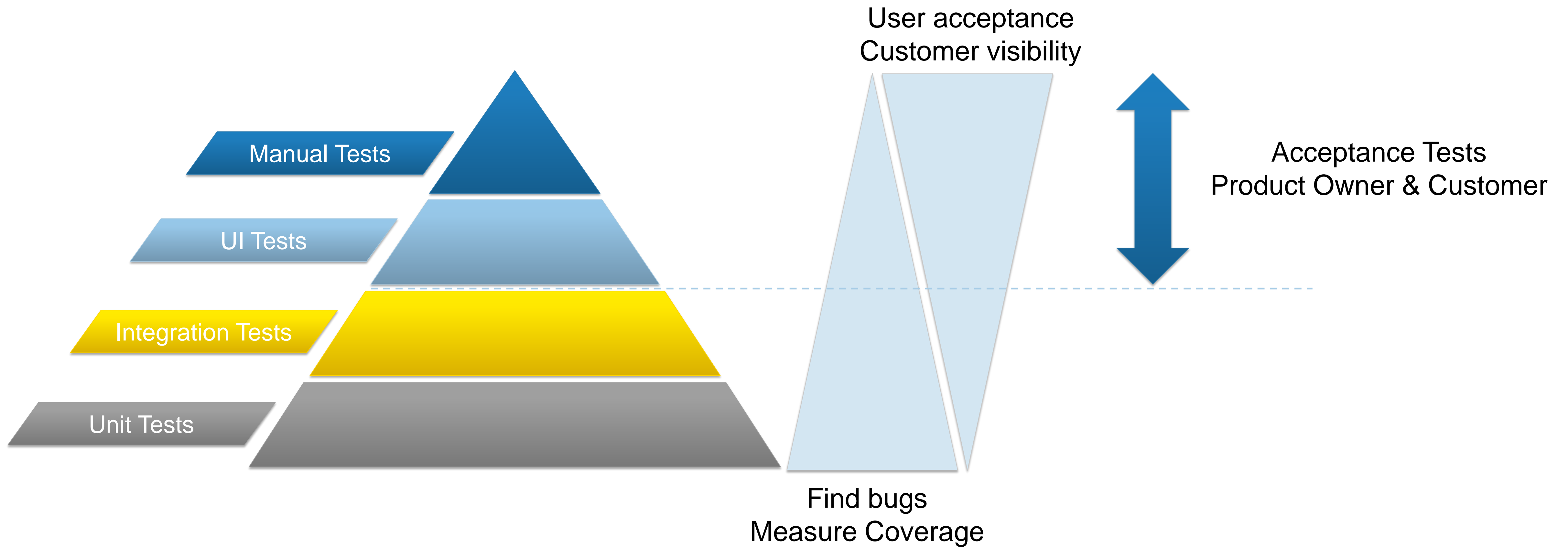
Concepts – Software testing



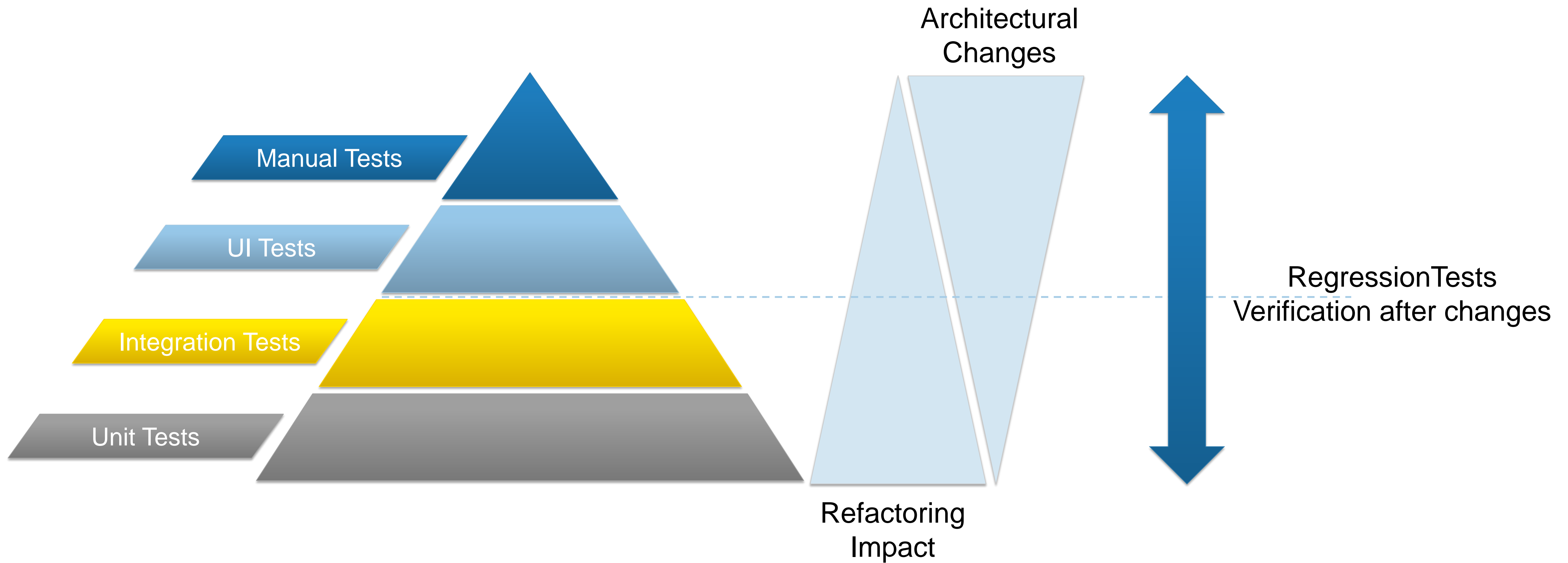
Concepts – Software testing



Concepts – Software testing

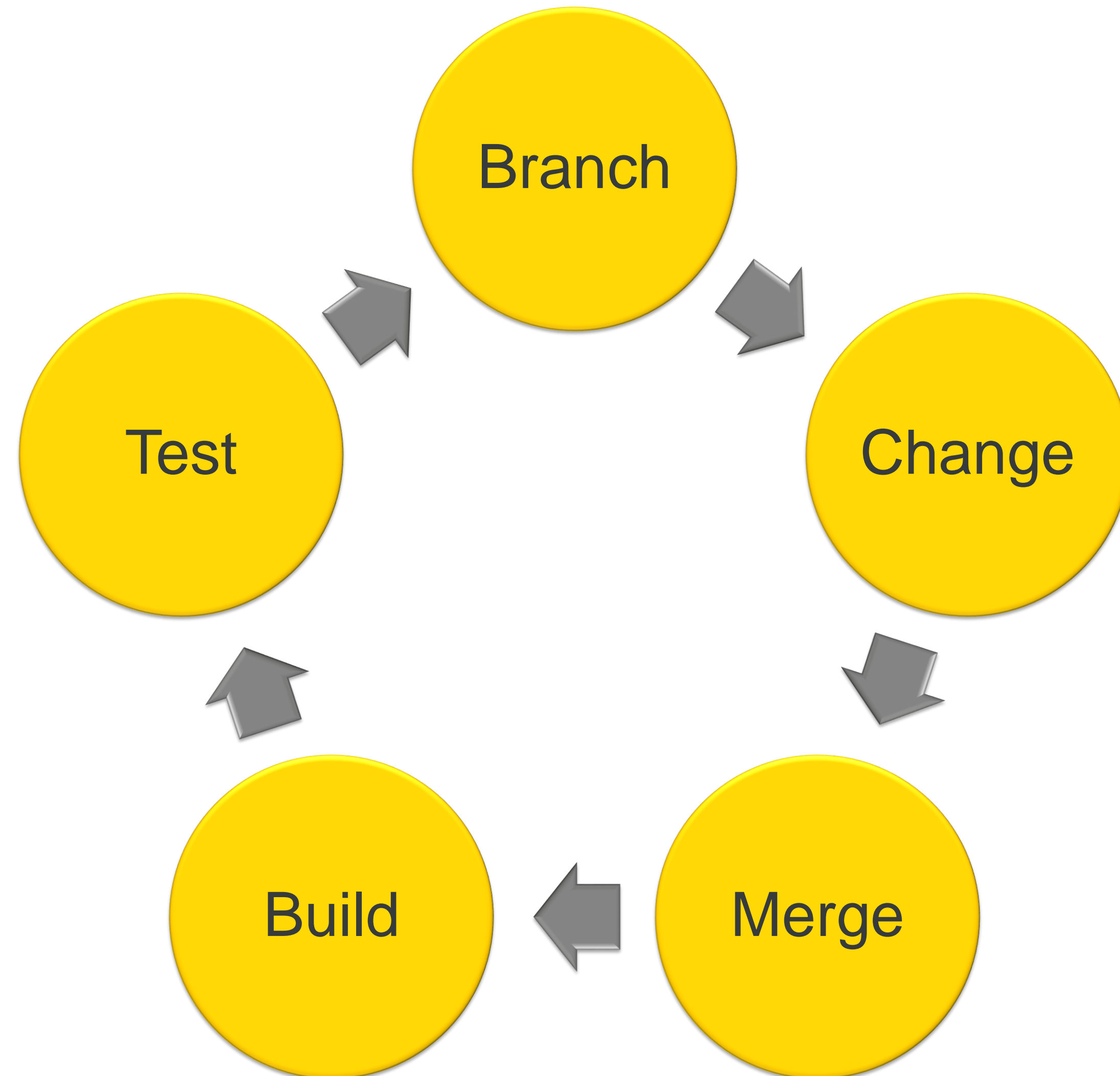


Concepts – Software testing



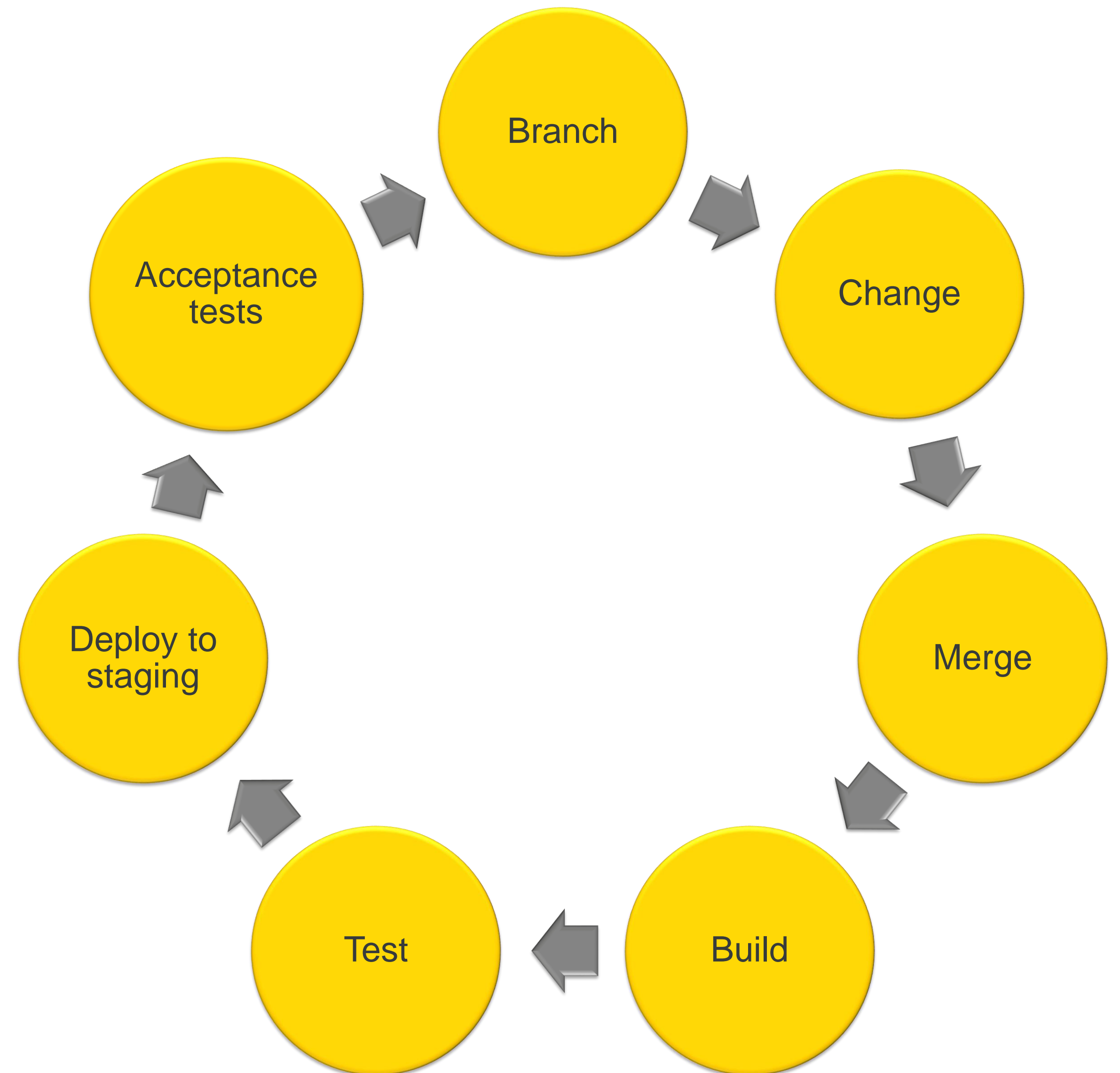
Concepts – CI – Continuous integration

- Born from [extreme programming](#)
- Merge your code early and often
- Build your code automated after change
- Test your code automated after change
- Goal
 - Early feedback on changes
 - Reduce merge conflicts



Concepts – CD – Continuous delivery

- Release your software whenever it's required
- Extends [continuous integration](#) by
 - Deployment into a staging environment
 - Acceptance tests



Concepts – CI/CD pipelines

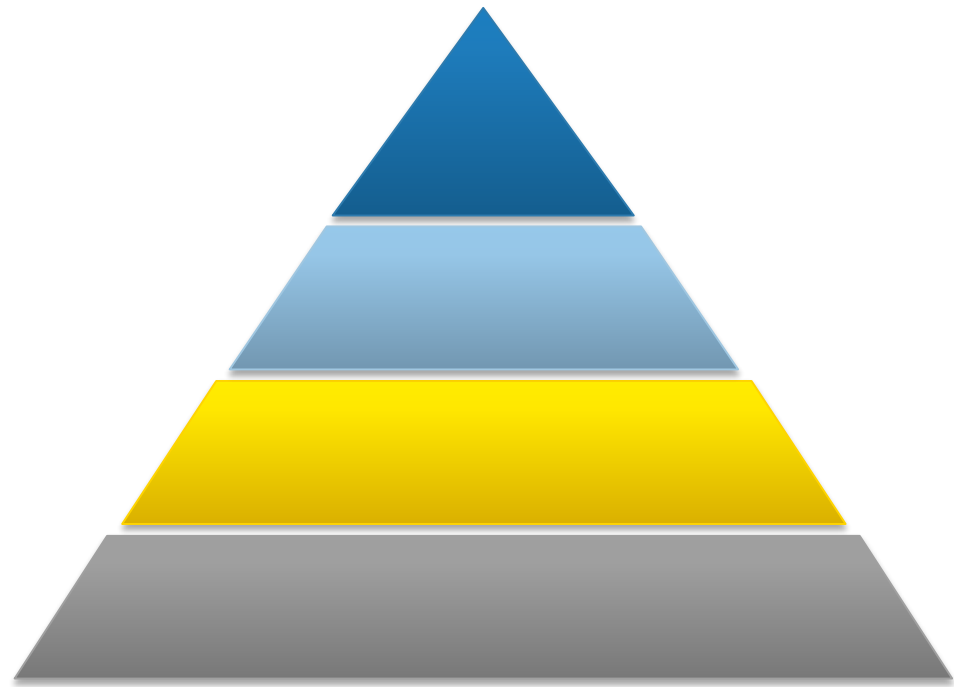
- Automate everything from code change to pre production
- Increase visibility of issues while building, testing, deploying your product
- Allows fast feedback loops
- Empowers you to deliver continuously
- Enables continuous deployment

Continuous delivery

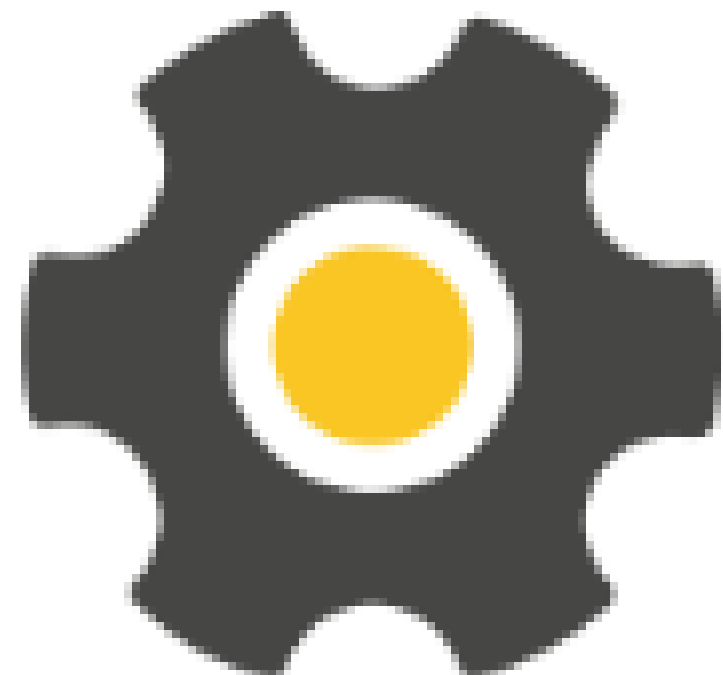
Continuous integration



Concepts



Software testing in CI/CD pipelines



Our goals for today

1. Take a sample app
2. Create a multi stage environment on a container platform
3. Build the app
4. Create a pipeline to deploy the app
5. Add integration tests to the pipeline
6. Add End-2-End tests to the pipeline



Sample app

- Todo list app
- Located in a [git](#) service (GOGS)
- Small GUI
- Small REST API

TODO list

No todos found

New TODO entry

Title

Short title of the task

Description

Full description of the task

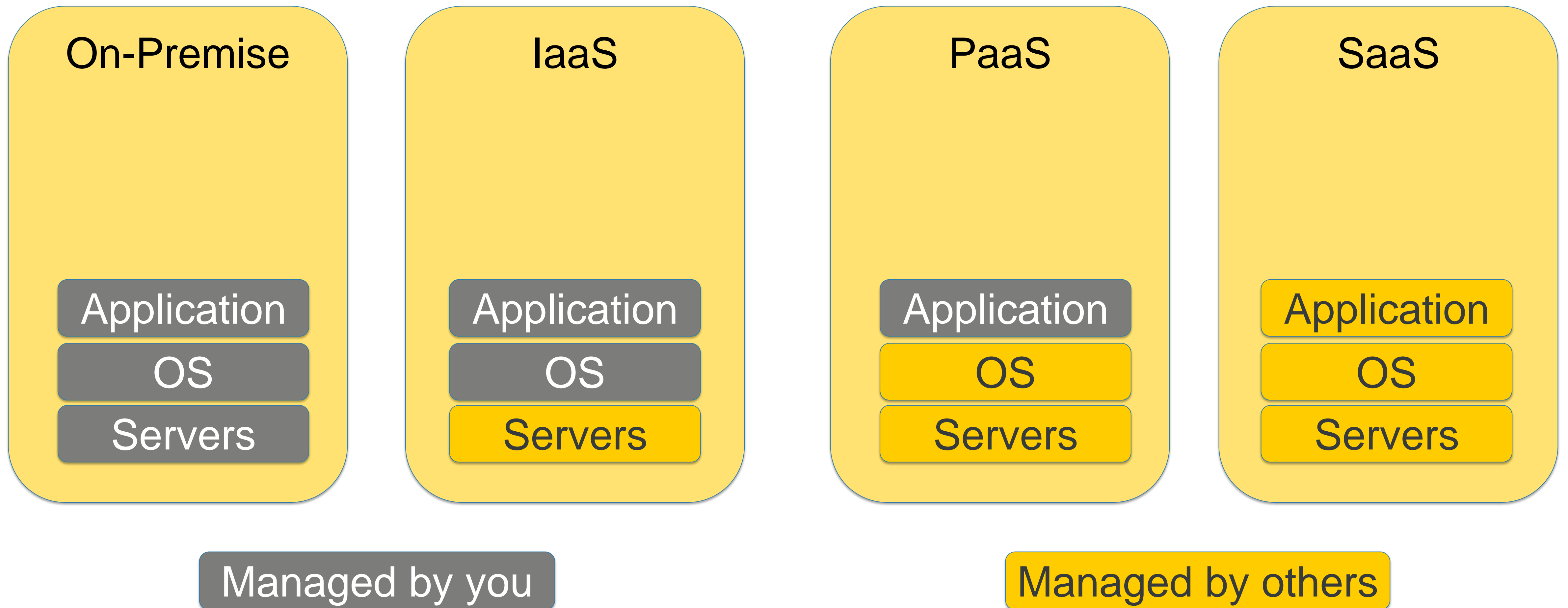
Add

Our goals for today

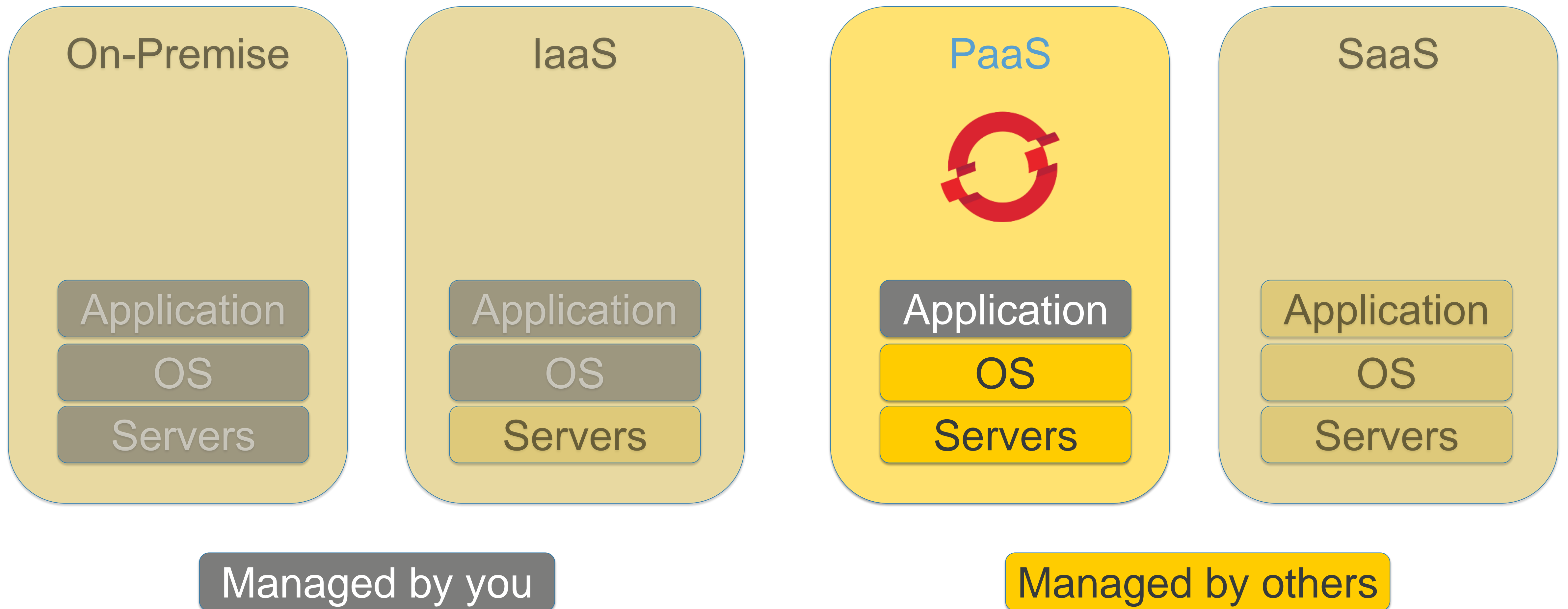
1. Take a sample app ✓
2. Create a multi stage environment on a container platform
3. Build the app
4. Create a pipeline to deploy the app
5. Add integration tests to the pipeline
6. Add End-2-End tests to the pipeline



Container platform

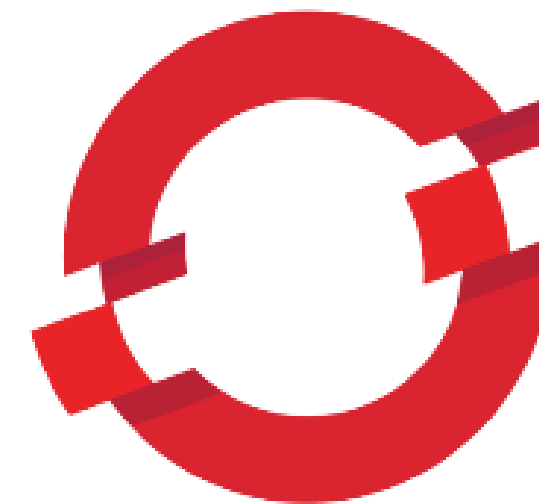
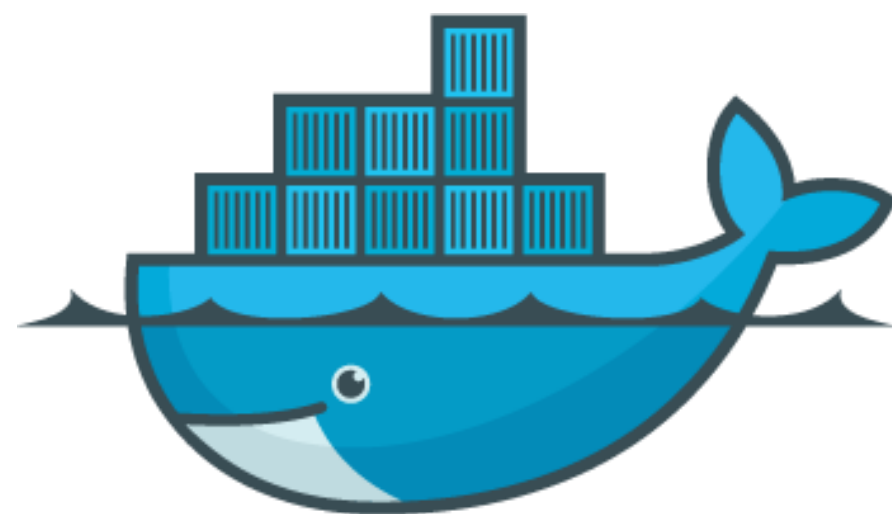


Container platform

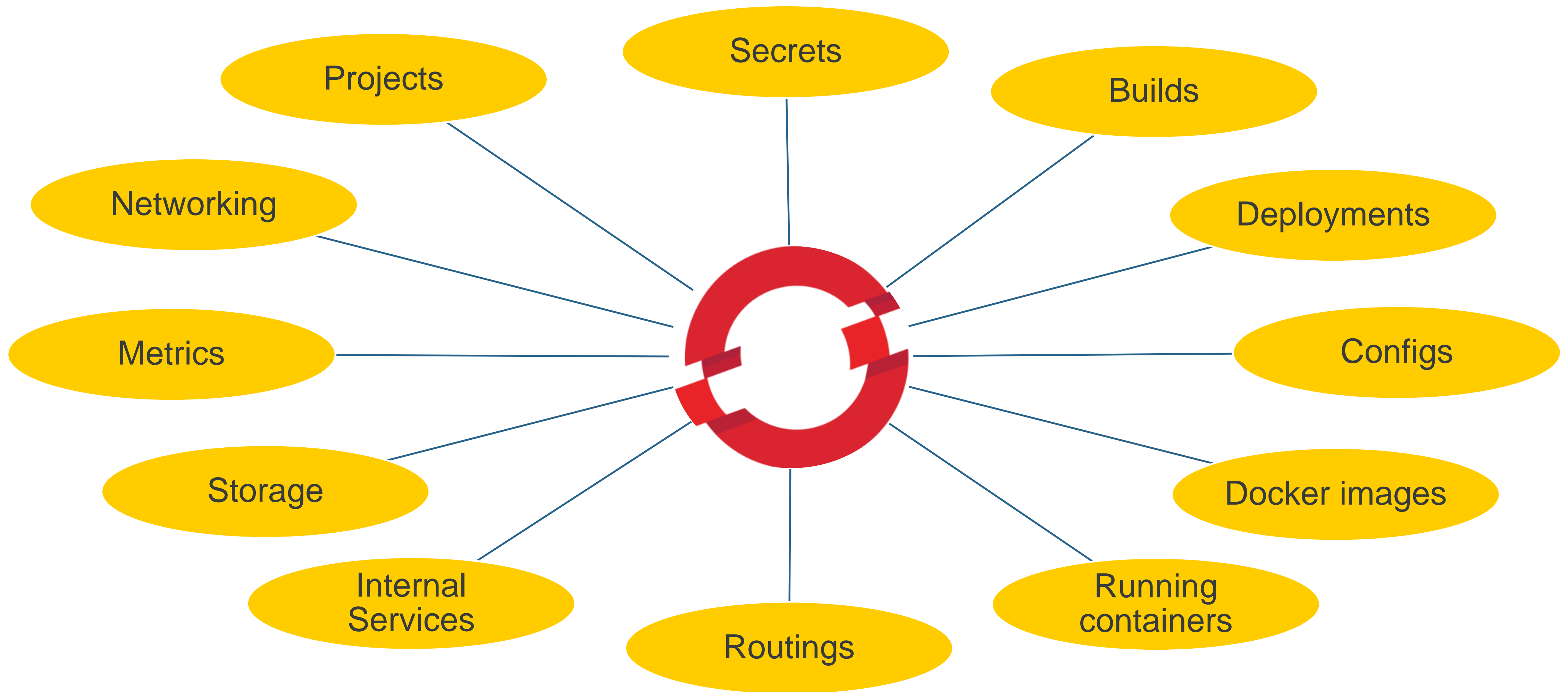


Container platform - OpenShift

- Developed and maintained by RedHat since 2011
- Open source container platform *OKD* (Apache Licence 2.0)
- Professional support options
- Public OpenShift cloud available
- Based on Kubernetes
- Uses *Docker* containers
- Additional platform specific features



Container platform - OpenShift



Mission

Create a multi stage environment on a container platform

Let's code!

Our goals for today

1. Take a sample app ✓
2. Create a multi stage environment on a container platform ✓
3. Build the app
4. Create a pipeline to deploy the app
5. Add integration tests to the pipeline
6. Add End-2-End tests to the pipeline



Classic vs. Modern

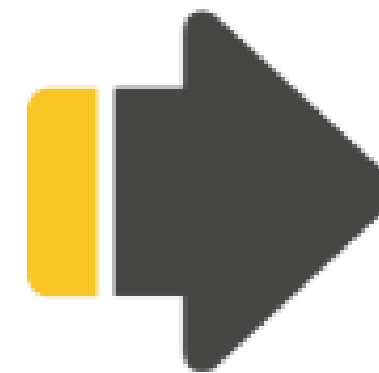
CI/CD pipelines – Classic

- Jenkins installed on a VM or bare metal
- Create a job
 - By yourself
 - Via admin
 - Via a business process
- Maybe adapt to a predefined pipeline
- Shared resources / workers



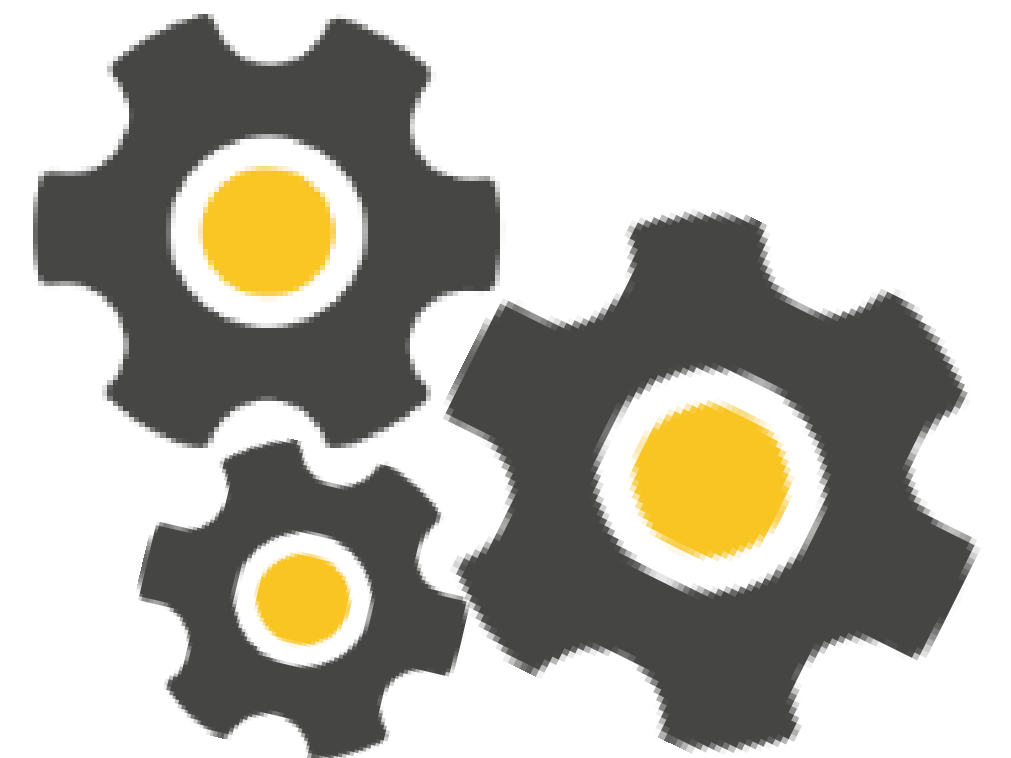
CI/CD pipelines – Modern

- Define your individual pipeline
- Bind it to your source control
- Deploy the pipeline to your Jenkins
- Get some coffee

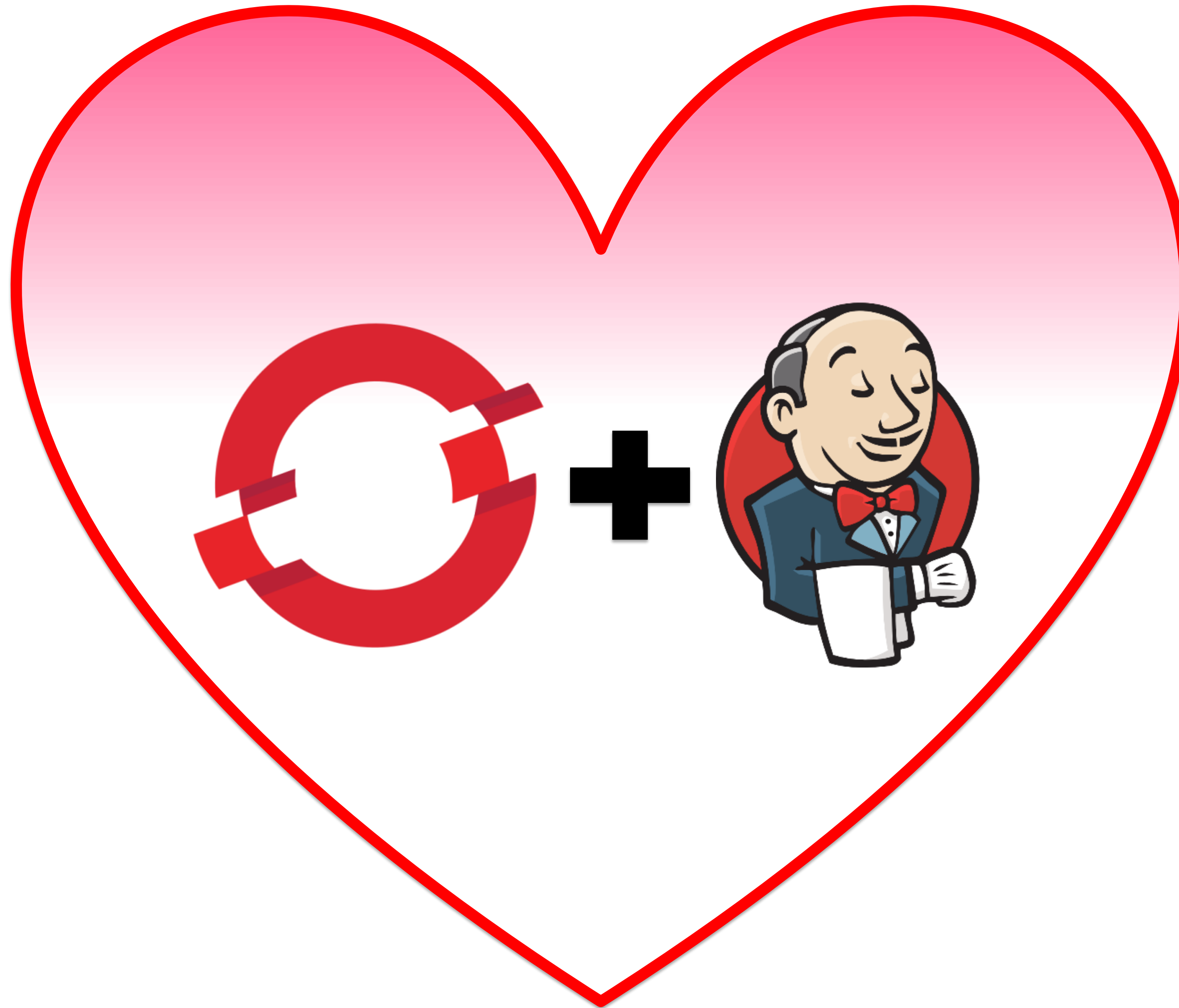


CI/CD pipelines – Modern

- The modern life cycle has to be enabled by software
 - [Configuration as code](#)
- Various software solutions on the market
 - Travis CI
 - GitLab CI
 - OpenShift with Jenkins
 - etc.
- Provide a configurable platform to specify build pipelines

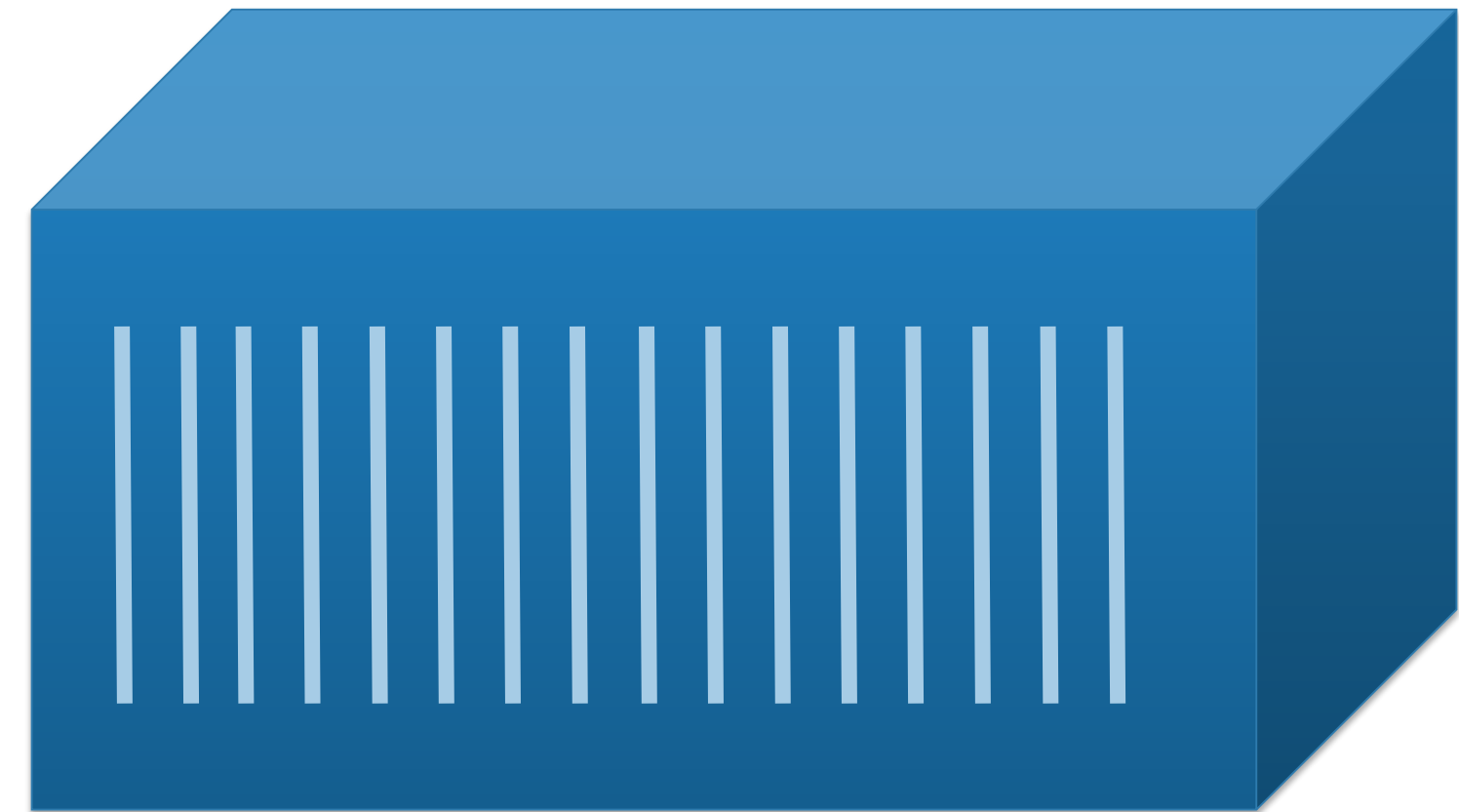
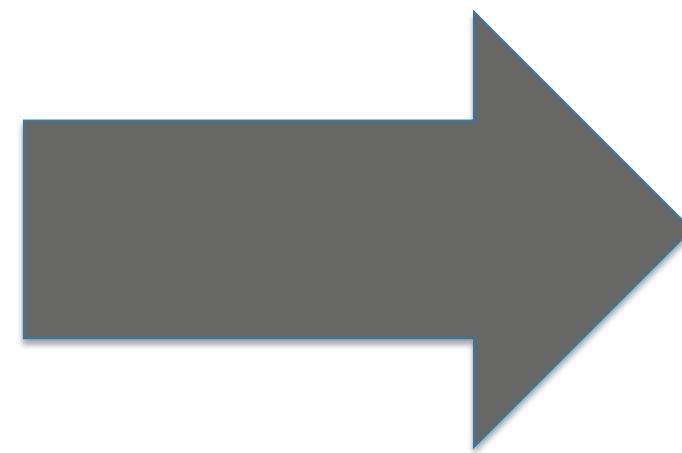


CI/CD pipelines – Jenkins and OpenShift



CI/CD pipelines – Build the software

- What we have: Source code
- What we want: A deployable artefact
- How do we come from Source code to a deployable artefact?



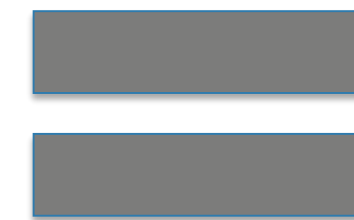
CI/CD pipelines – Build the software – Jenkins

- We need a Jenkins file to define the pipeline
- Build a [Jenkins](#) containing the Jenkins file
- To build software, you need a build config
- Jenkins and build configuration stored in the git repository

```
node{  
    checkout scm  
}
```

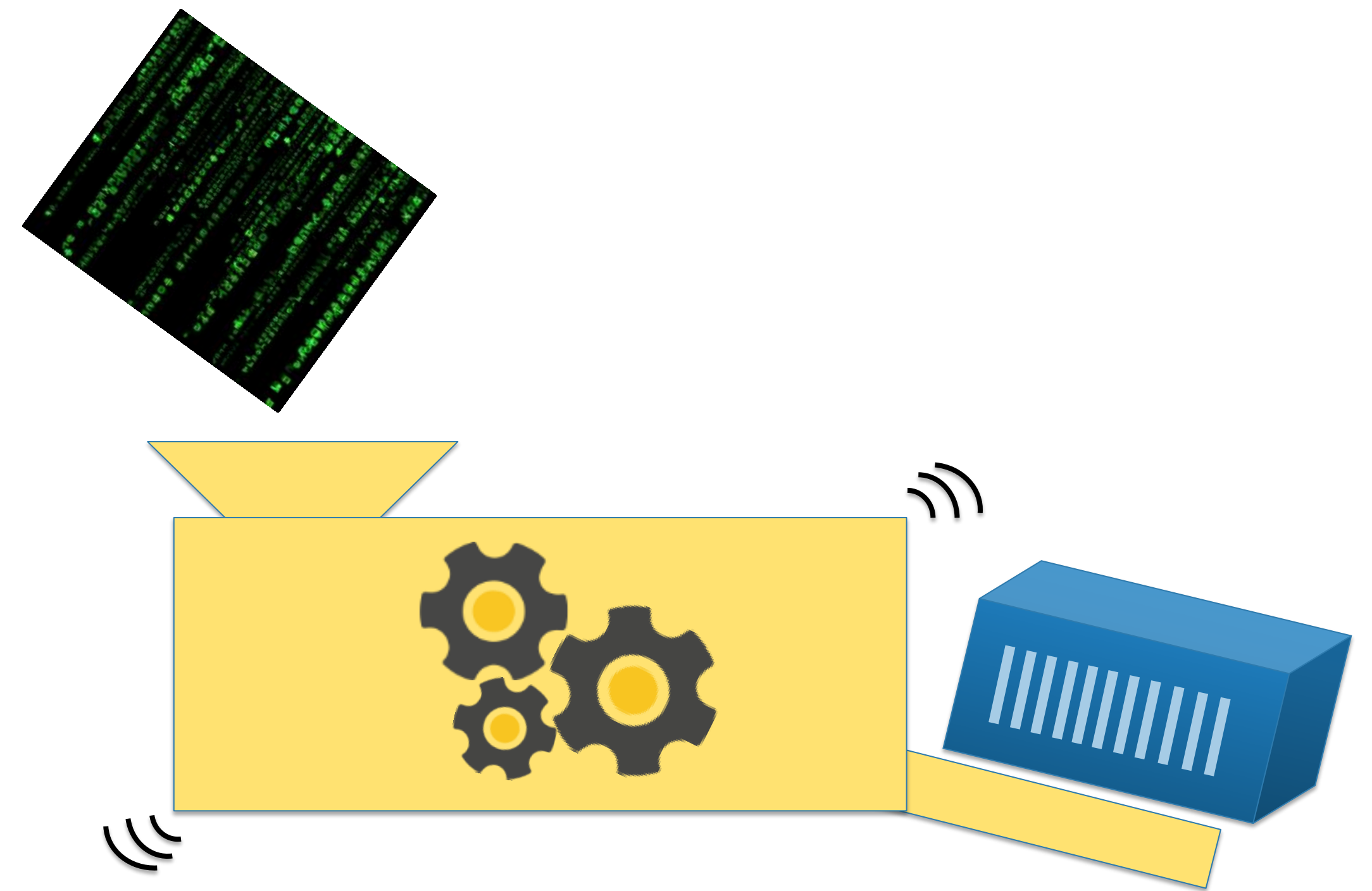


```
- apiVersion: v1  
kind: BuildConfig  
metadata:  
  labels:  
    build: ${APPLICATION_NAME}-pipeline  
    component: backend  
    app: ${APPLICATION_NAME}  
    name: ${APPLICATION_NAME}-pipeline  
spec:  
  failedBuildsHistoryLimit: 5  
  runPolicy: Serial  
  source:  
    git:  
      uri: ${INFRASTRUCTURE_REPOSITORY_URL}  
      sourceSecret:  
        name: gitlab-ssh  
      type: Git  
  strategy:  
    jenkinsPipelineStrategy:  
      jenkinsfilePath: infra/jenkins/Jenkinsfile  
    type: Source
```



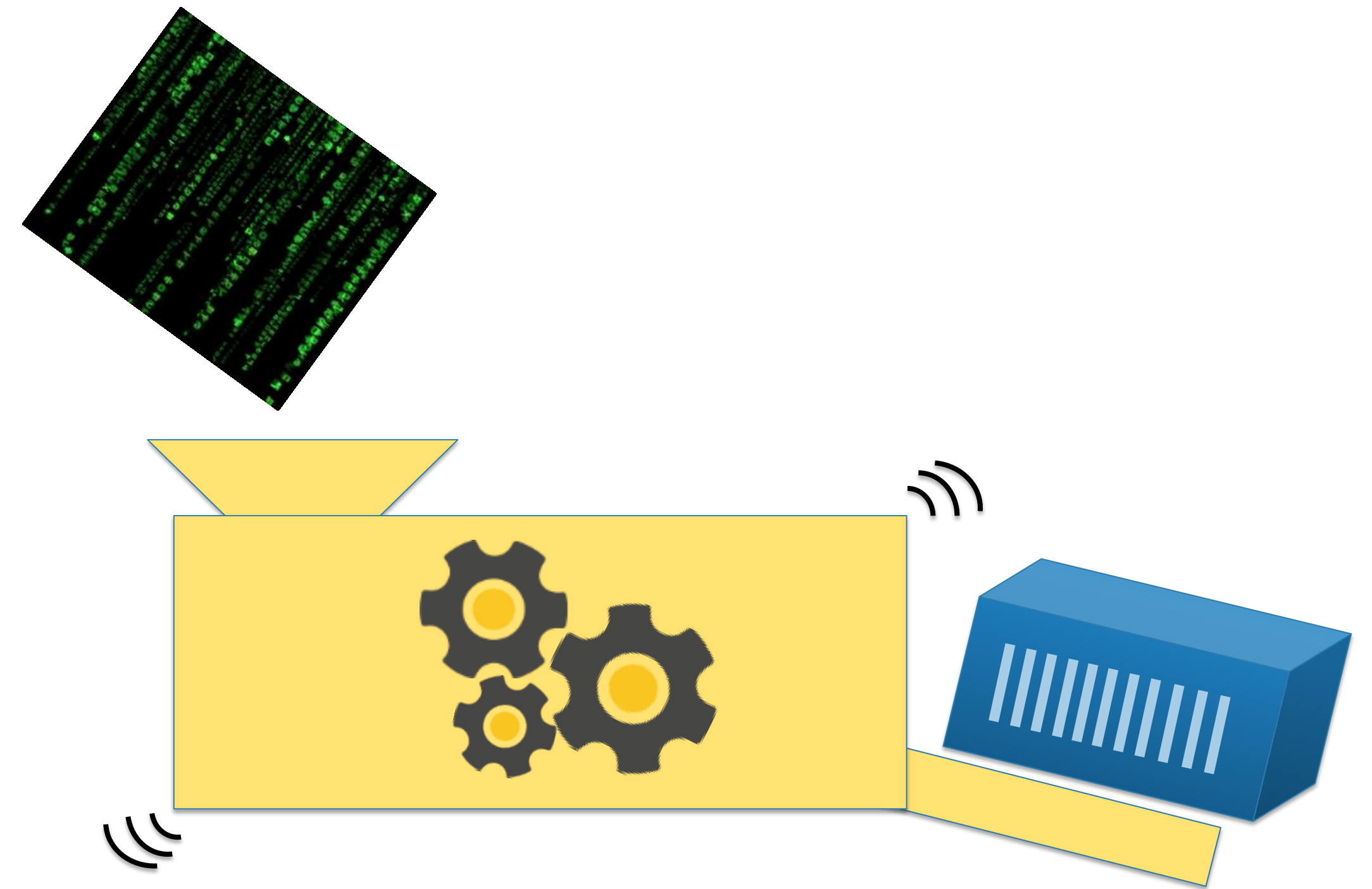
CI/CD pipelines – Build the software – S2I

- We're going to use [Source to Image](#) (S2I)
- OpenShift feature
- S2I defines an interface to build software
- Used via build config



CI/CD pipelines – Build the software – S2I

- S2I images contain a build environment
- S2I creates a deployable Docker image containing the app
- Image is pushed to the registry into an image stream
- Available for many technologies and ready to use



CI/CD pipelines – Build the software – S2I



CI/CD pipelines – Build the software – S2I

Mission

Build the app

Jenkins! Please serve the app!

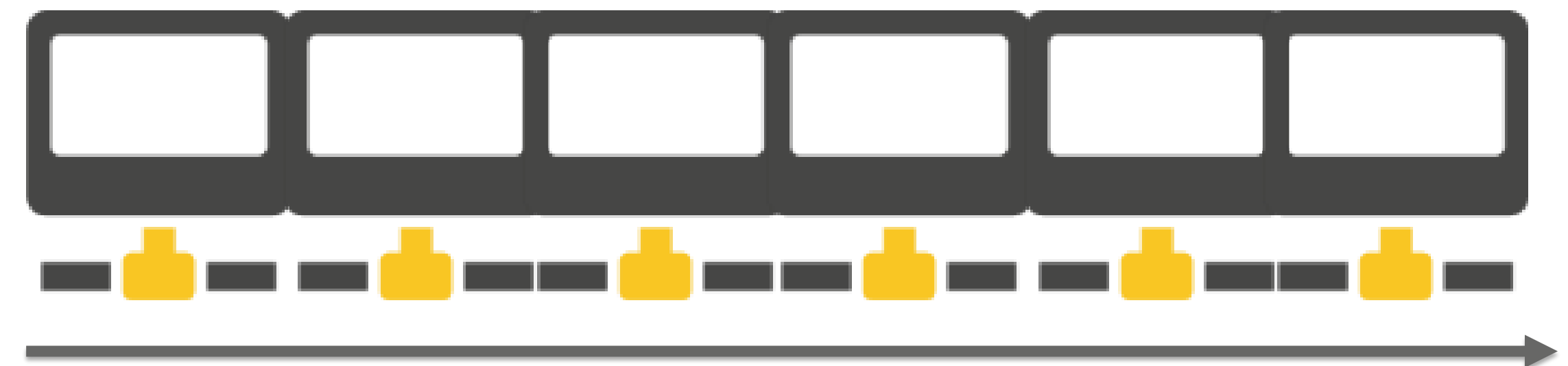
Our goals for today

1. Take a sample app ✓
2. Create a multi stage environment on a container platform ✓
3. Build the app ✓
4. Create a pipeline to deploy the app
5. Add integration tests to the pipeline
6. Add End-2-End tests to the pipeline



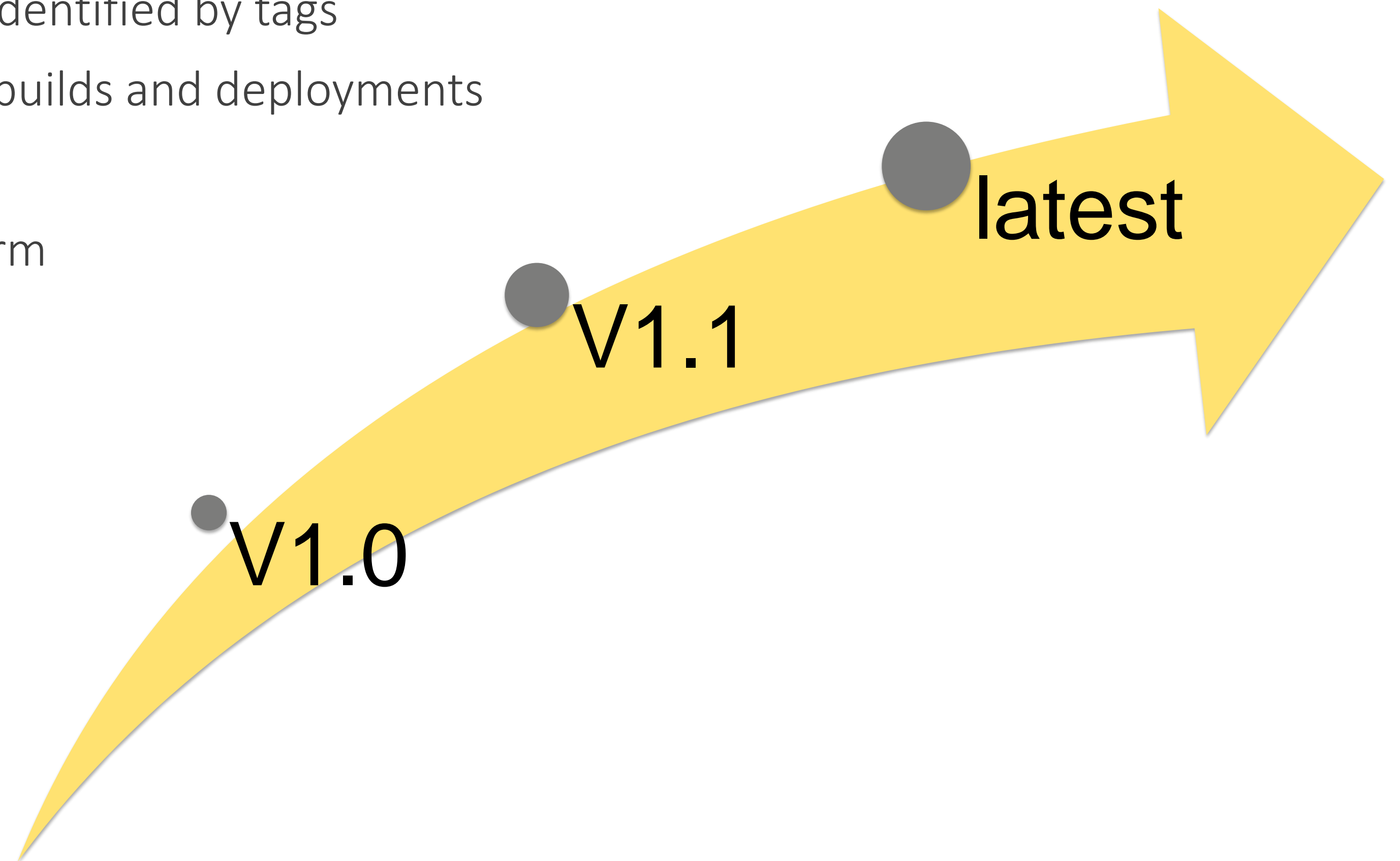
CI/CD pipelines – The pipeline

- We have
 - A Jenkins file
 - A deployable immutable artefact
- We need
 - To deploy the artefact - [Deployment config](#)
 - To transport the artefact between stages - Image streams



CI/CD pipelines – The pipeline – Image streams

- Same idea as for a Docker registry
- ... with some additions
- Image streams contain Docker images identified by tags
- Changes on [image streams](#) can trigger builds and deployments
- Image streams can refer to
 - The registry of the container platform
 - Other image streams
 - External registries



CI/CD pipelines – The pipeline

Mission

Create a pipeline to deploy the app

The deployment is strong with this one.

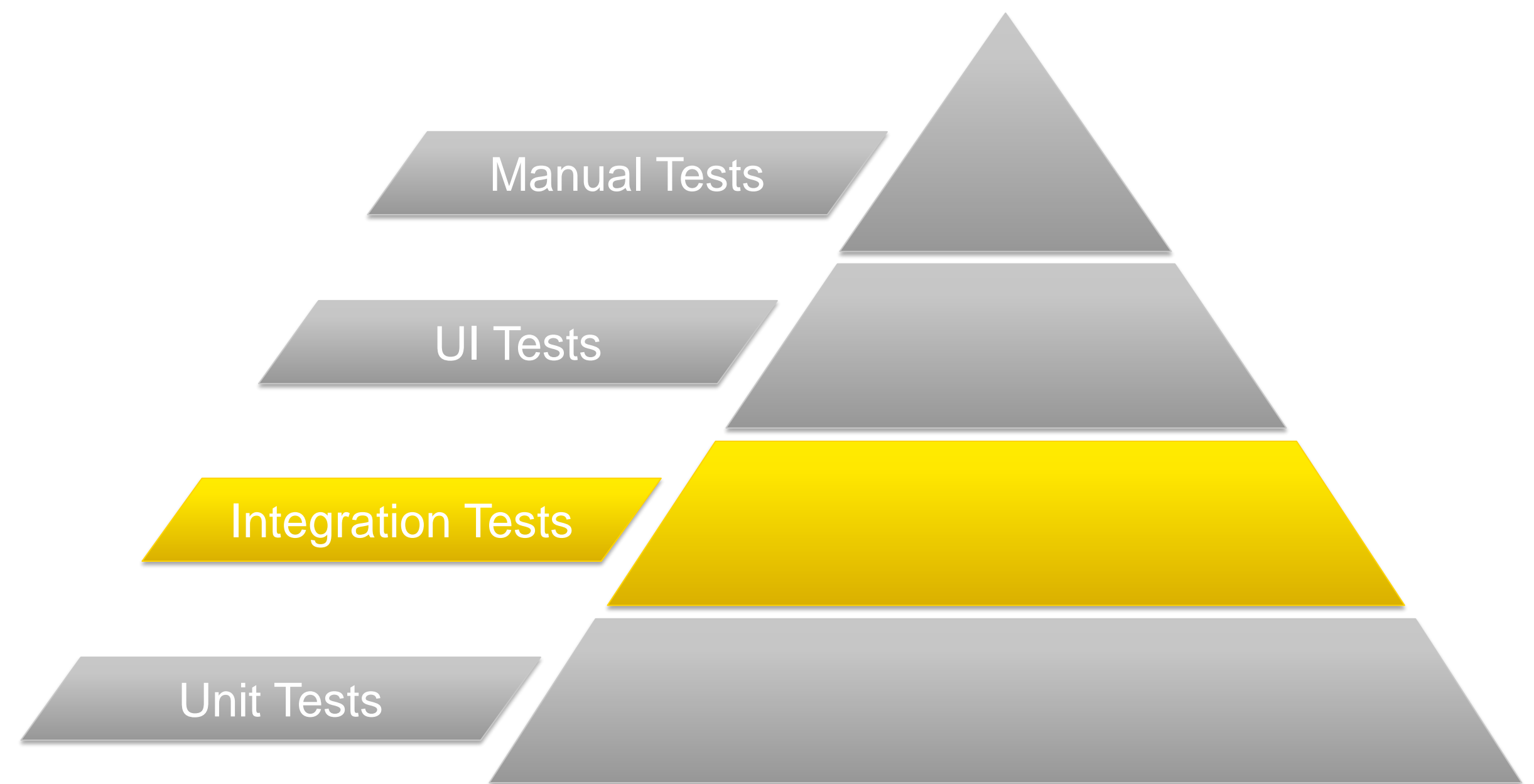
Our goals for today

1. Take a sample app ✓
2. Create a multi stage environment on a container platform ✓
3. Build the app ✓
4. Create a pipeline to deploy the app ✓
5. Add integration tests to the pipeline
6. Add End-2-End tests to the pipeline



Integration testing

- Tests the communication between software components
- The [system under test](#) (SUT) is tested as a black box
- Software required to provide endpoints to communicate with
- **CITRUS** 🍊



Integration testing with CITRUS

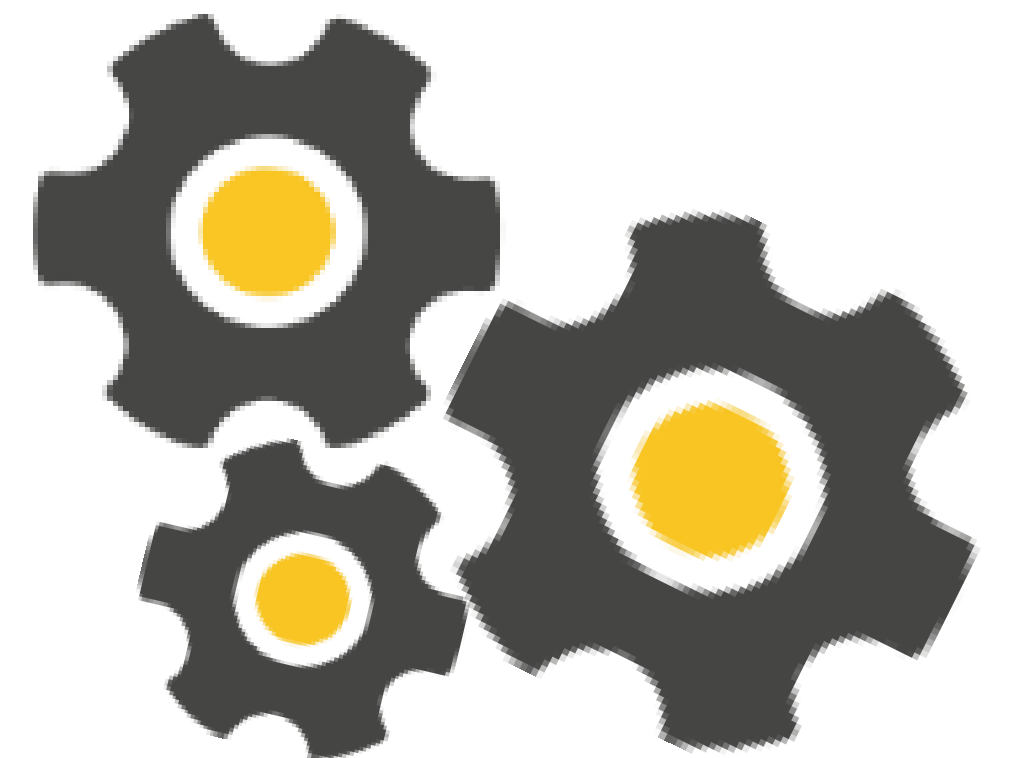
Citrus is
an open-source framework for automated
integration testing of messaging interfaces
in enterprise applications.



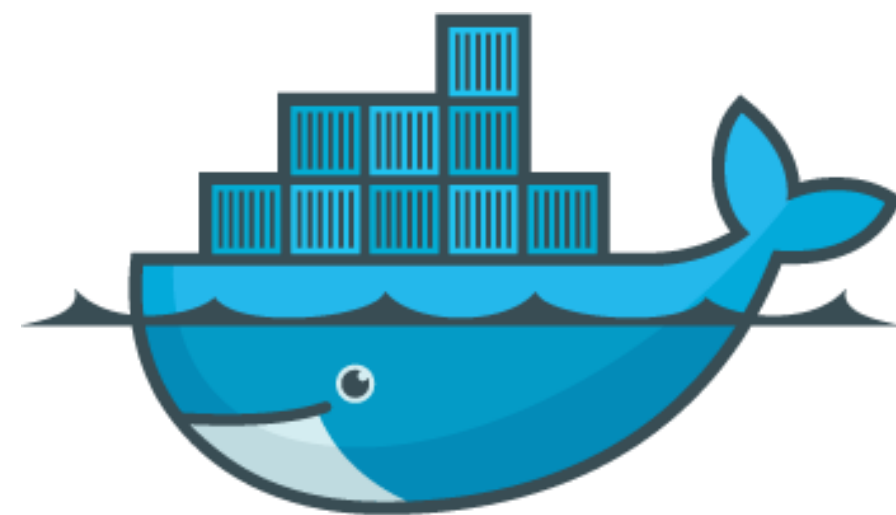
citrusframework.org

Integration testing with CITRUS

- Based on the [Spring](#) Framework
- Integrates with common build tools
 - Maven
 - Gradle
 - Ant
- Therefore tests are easily executable with Jenkins
- Rich Java- and XML-DSL
- No mocks – only real messages over the wire



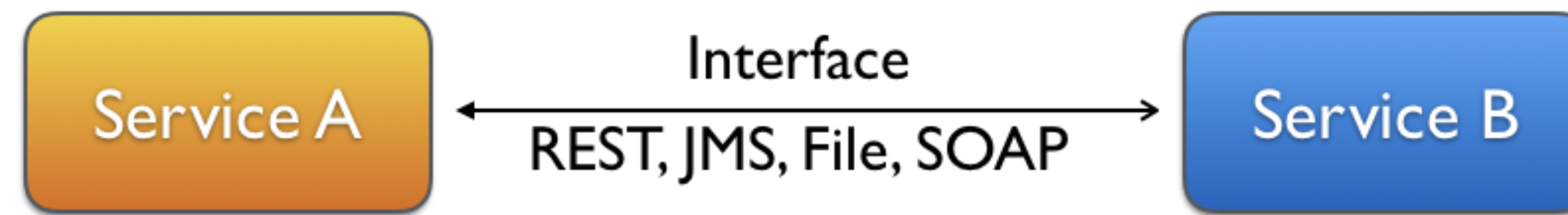
Integration testing with CITRUS🍊



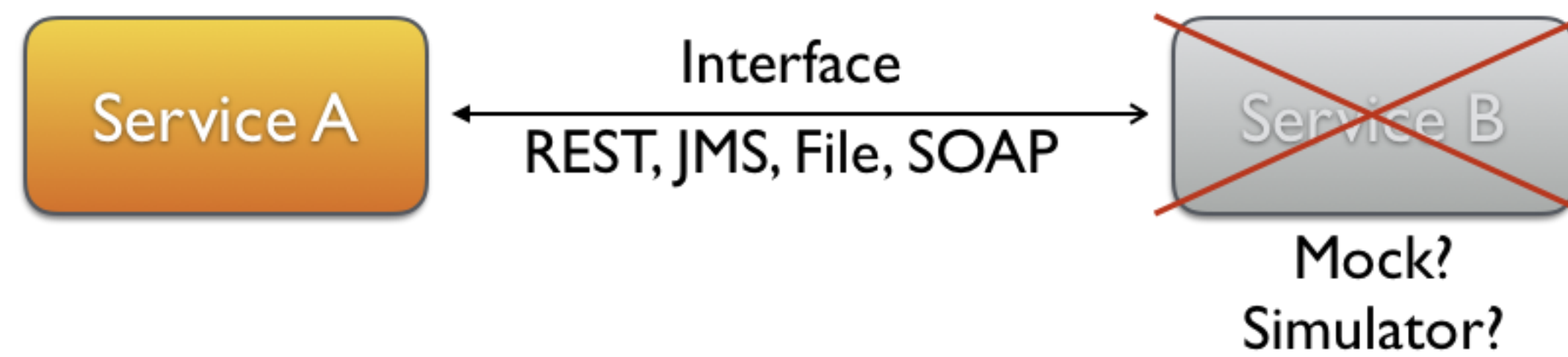
Integration testing with CITRUS

Endpoint	Description
citrus-http	HTTP client and server
citrus-jms	JMS consumer and producer
citrus-ws	SOAP WebServices client and server
citrus-mail	Mail client and server
citrus-docker	Docker client
citrus-camel	Apache Camel components
citrus-kubernetes	Kubernetes client
citrus-selenium	Selenium browser
citrus-ftp	FTP client and server
citrus-vertx	Vert.x consumer and producer
citrus-ssh	SSH client and server
citrus-jdbc	JDBC server simulation
...	...

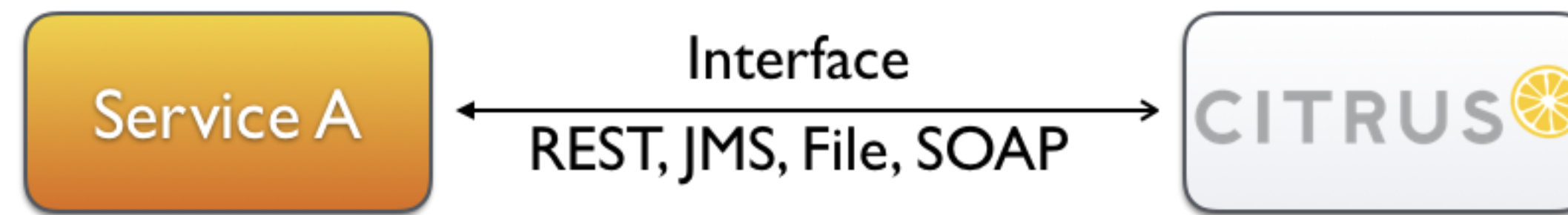
Integration testing with CITRUS🍊



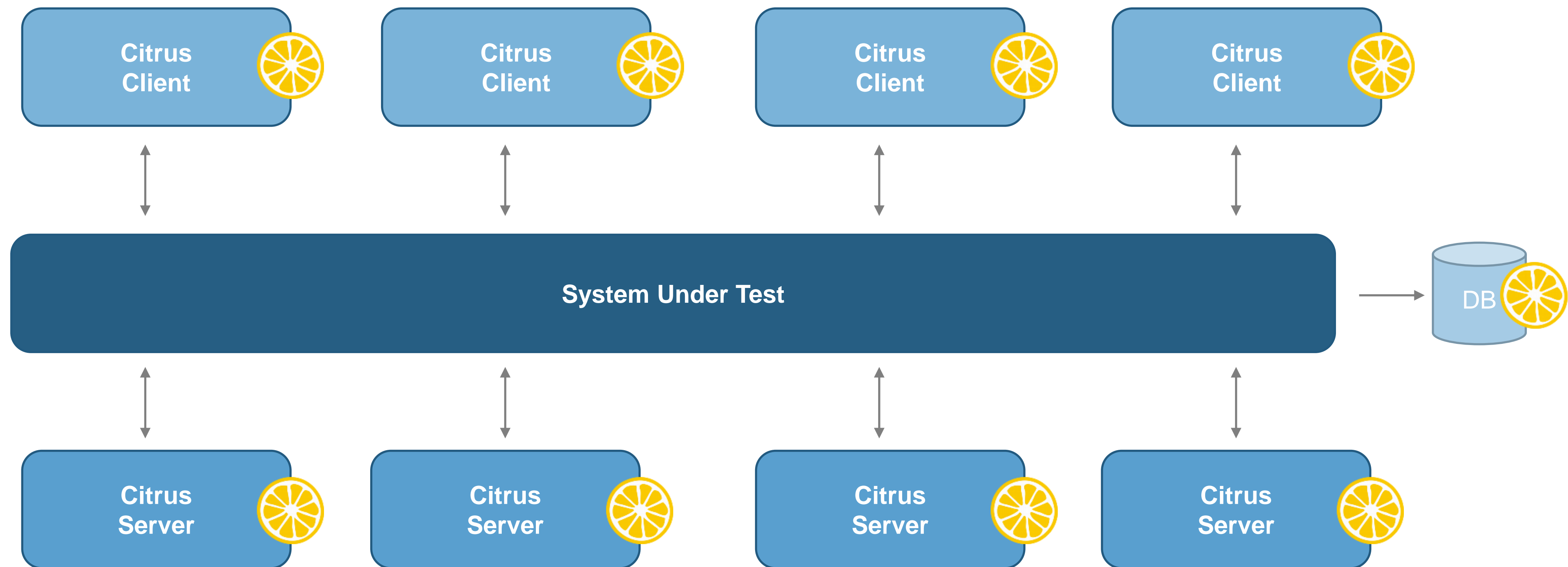
Integration testing with CITRUS🍊



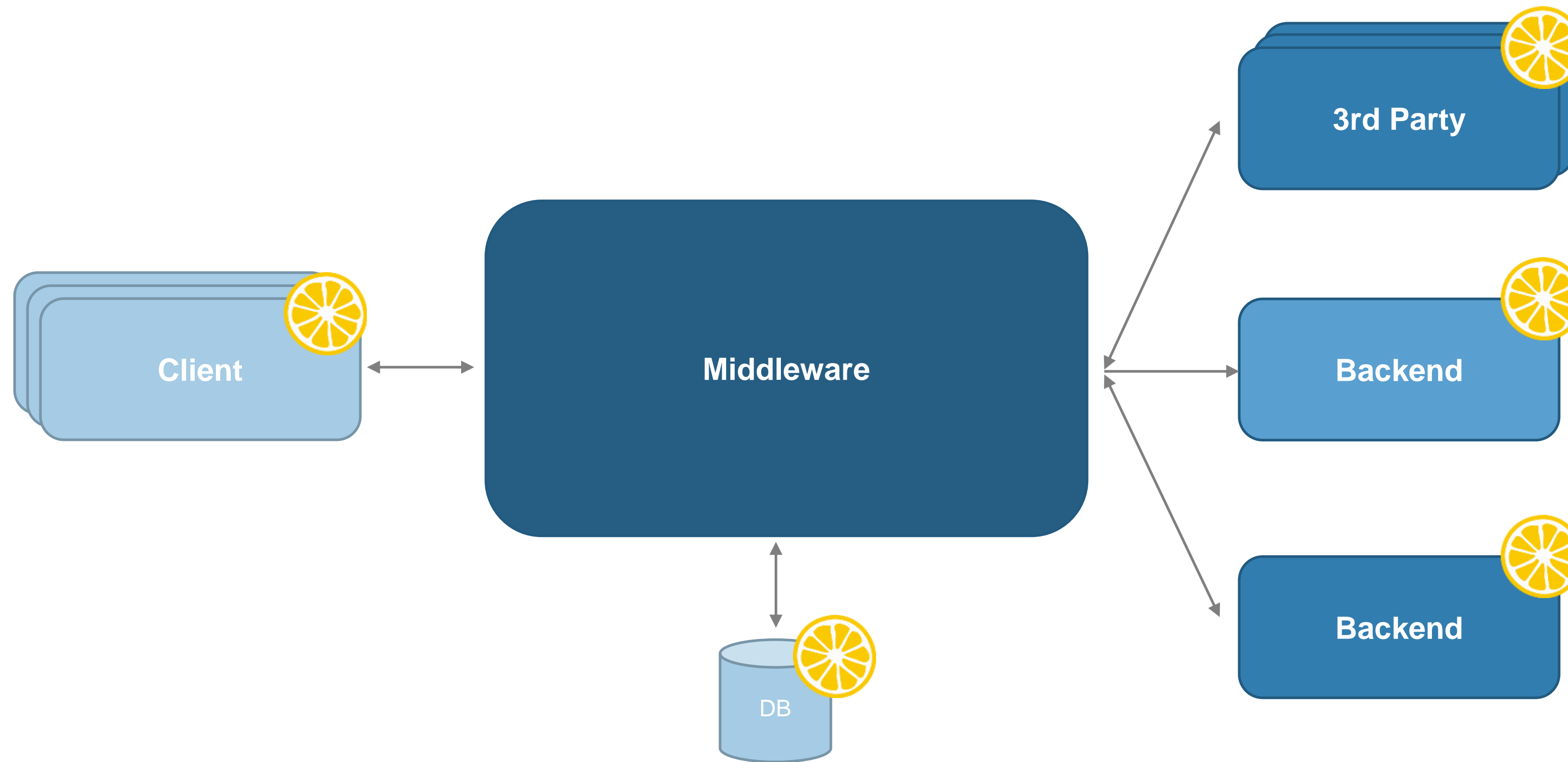
Integration testing with CITRUS🍊



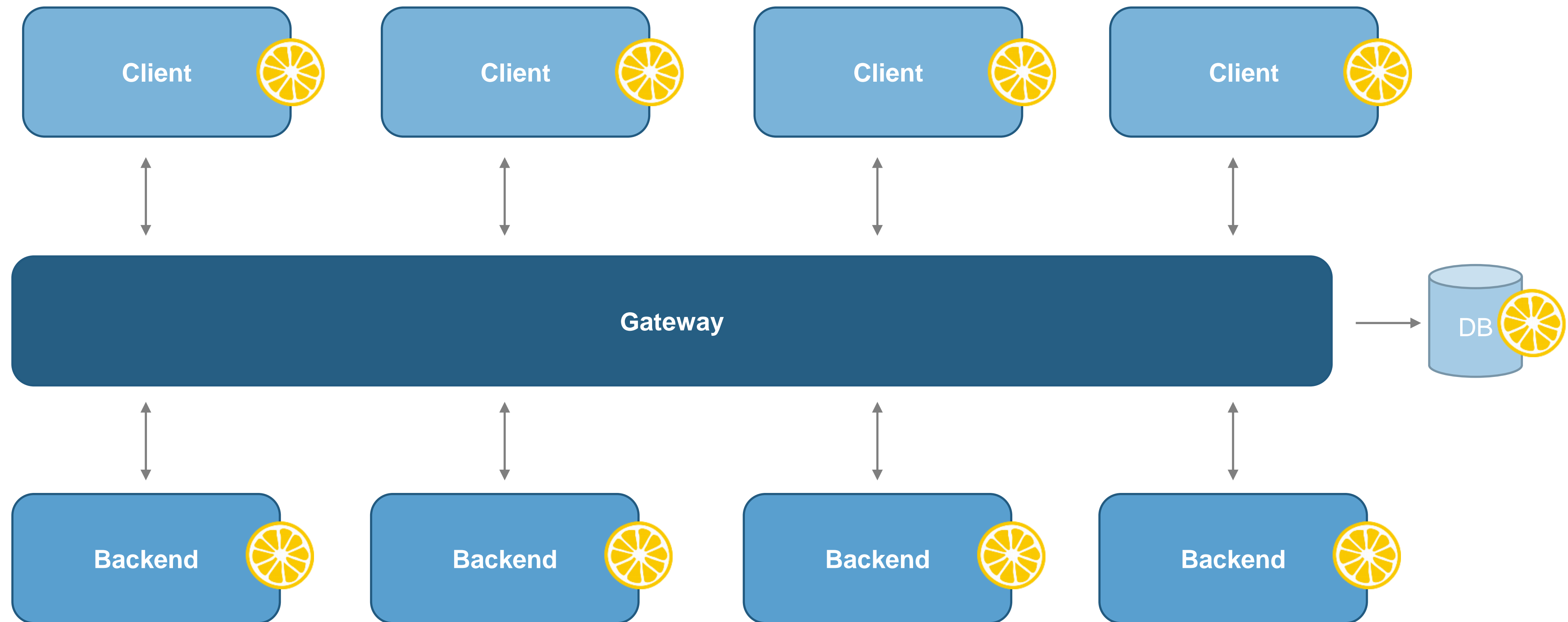
Integration testing with CITRUS



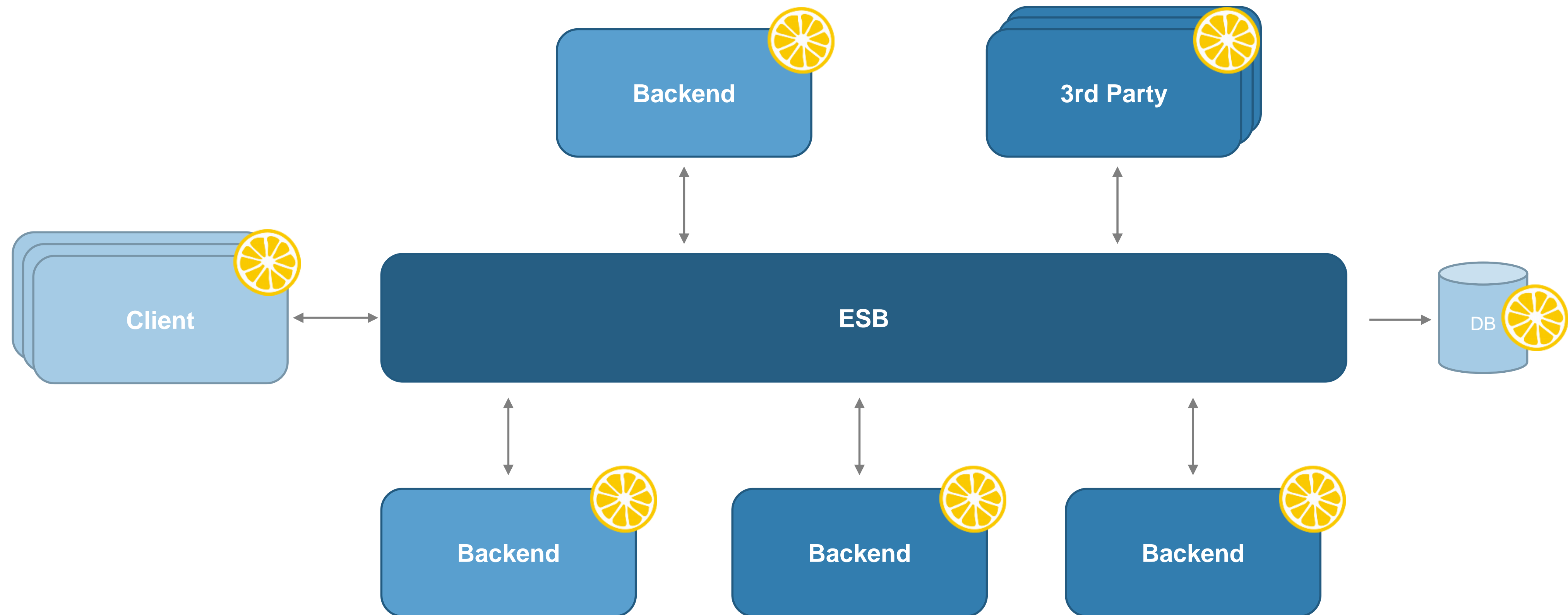
Integration testing with CITRUS



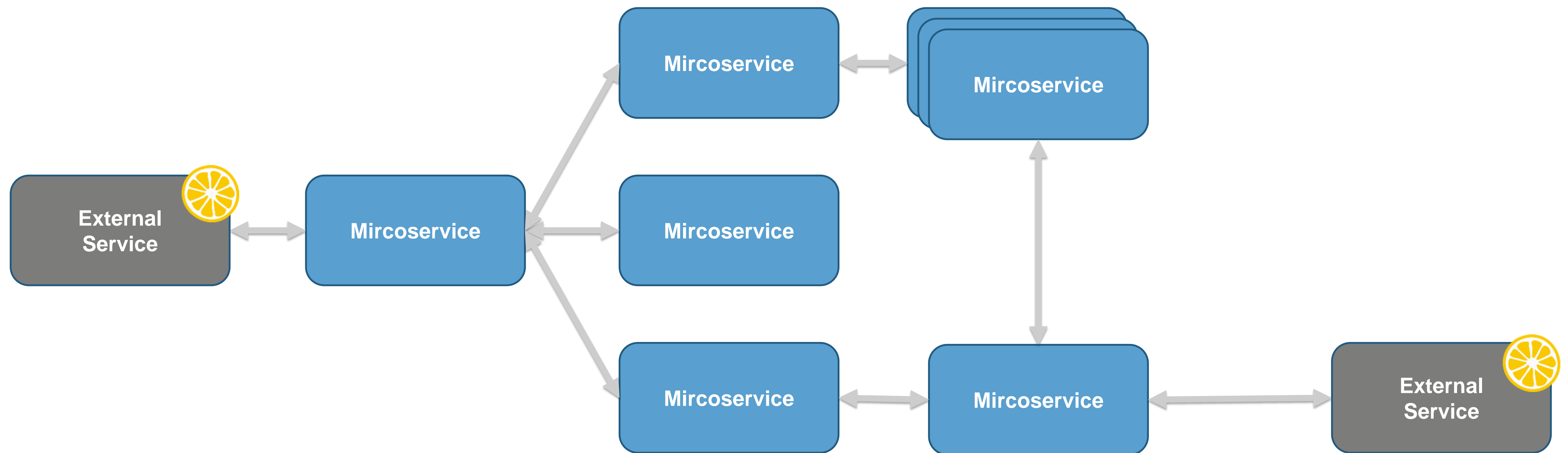
Integration testing with CITRUS



Integration testing with CITRUS



Integration testing with CITRUS🍊



Mission

Add integration tests to the pipeline

Let's add some juicy tests

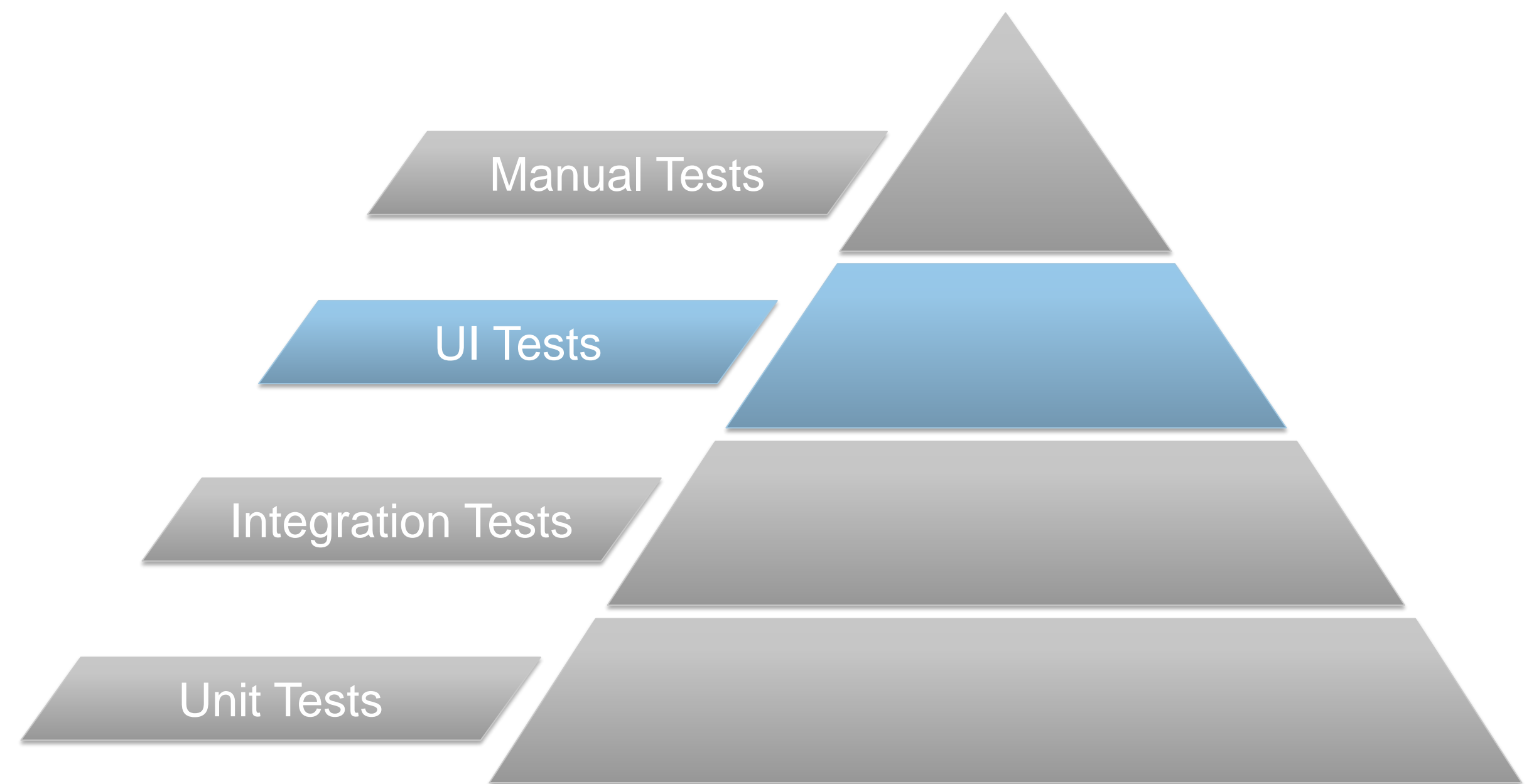
Our goals for today

1. Take a sample app ✓
2. Create a multi stage environment on a container platform ✓
3. Build the app ✓
4. Create a pipeline to deploy the app ✓
5. Add integration tests to the pipeline ✓
6. Add End-2-End tests to the pipeline

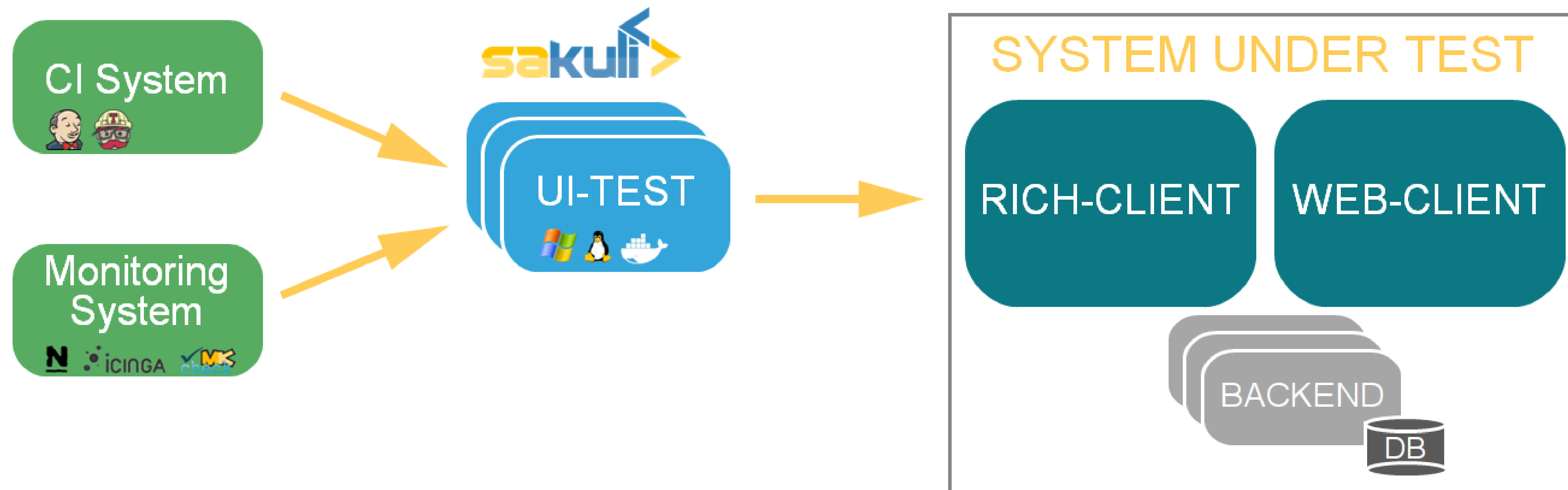


End-2-End testing

- Tests the whole application from end to end
- Tests the system from a user perspective
- Ensures that the **critical paths** of the application are working
- Includes desktop checks as well as browser checks
 - Important for fat client tests
 - Important for hybrid applications



End-2-End testing with



End-2-End testing with **sakuli**>

- The idea of **Sakuli**: Combine two frameworks
- Combine their strengths
- Eliminate their weaknesses
- Unify the API
- Make it executable and scalable in **cloud environments**
- Run it on every platform
- Provide native integration of monitoring systems

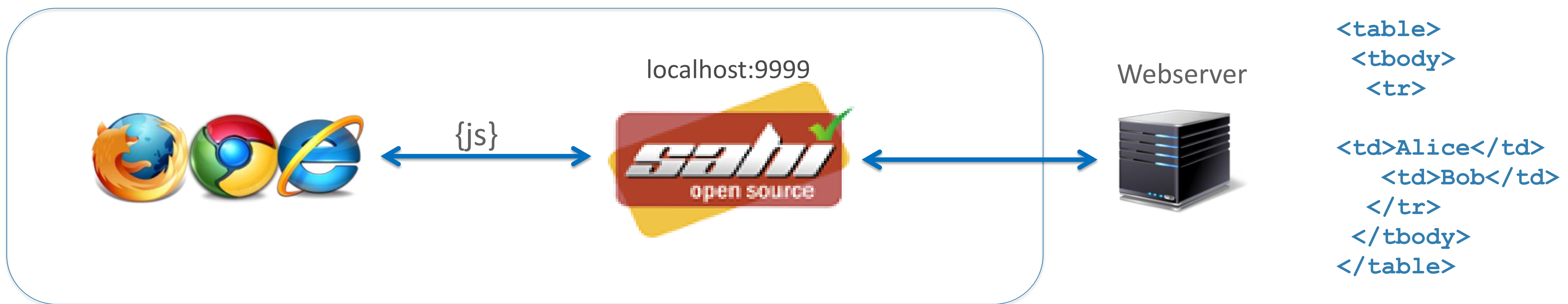


End-2-End testing with

Web testing tool (sahi.co.in, seleniumhq.org)

method based DOM access:

```
_assertContainsText ("Logged in as: Sakuli", _div("user_field"));  
_click(_span("Loaded Run Tabela"));  
_assertExists(_table("cross_table_fixed"));  
_assertExists(_cell("testing allowed", _rightOf(_span("Name")), _under(_cell("Action"))));
```



End-2-End testing with

Visual automation tool (sikuli.org)

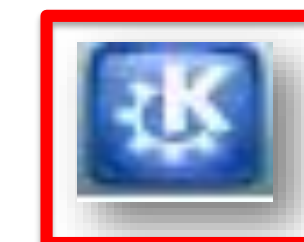
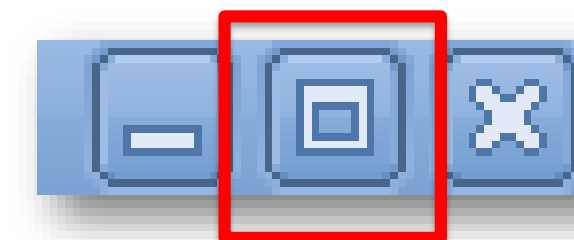
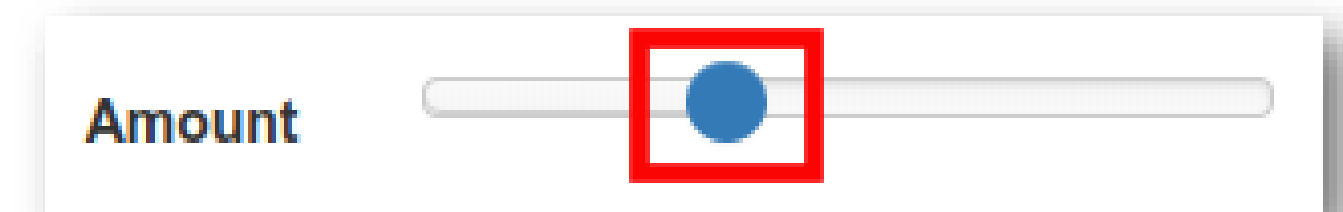
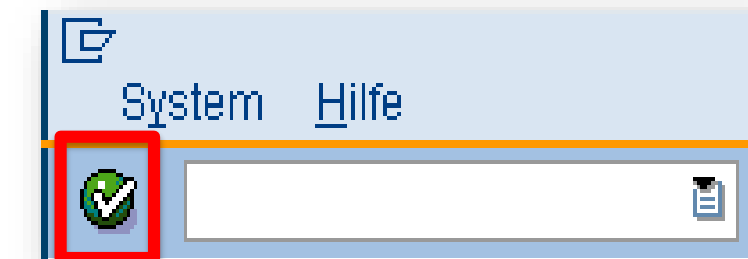
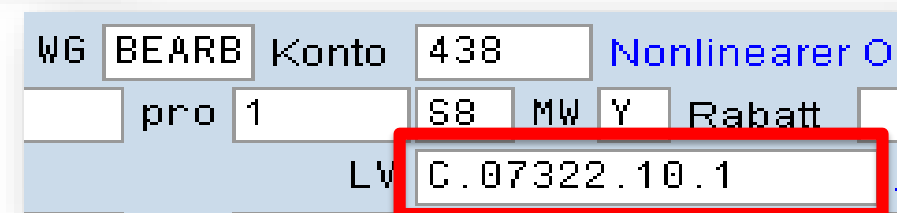
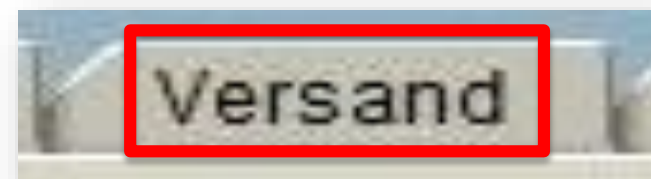
image identification, mouse & keyboard interaction:

```
screen.find("sap_ok").click();
```

```
screen.find("sap_ok").right(40).click().type("2223");
```

```
var bubble = new Region().waitForImage("bubble.png", 20);
```

```
bubble.dragAndDropTo(bubble.left(35)).highlight();
```



End-2-End testing with **sakuli**

Sikuli

- ✓ universal, complete screen
- ✗ (more) resource intensiv
- ✗ needs a "unlocked" screen



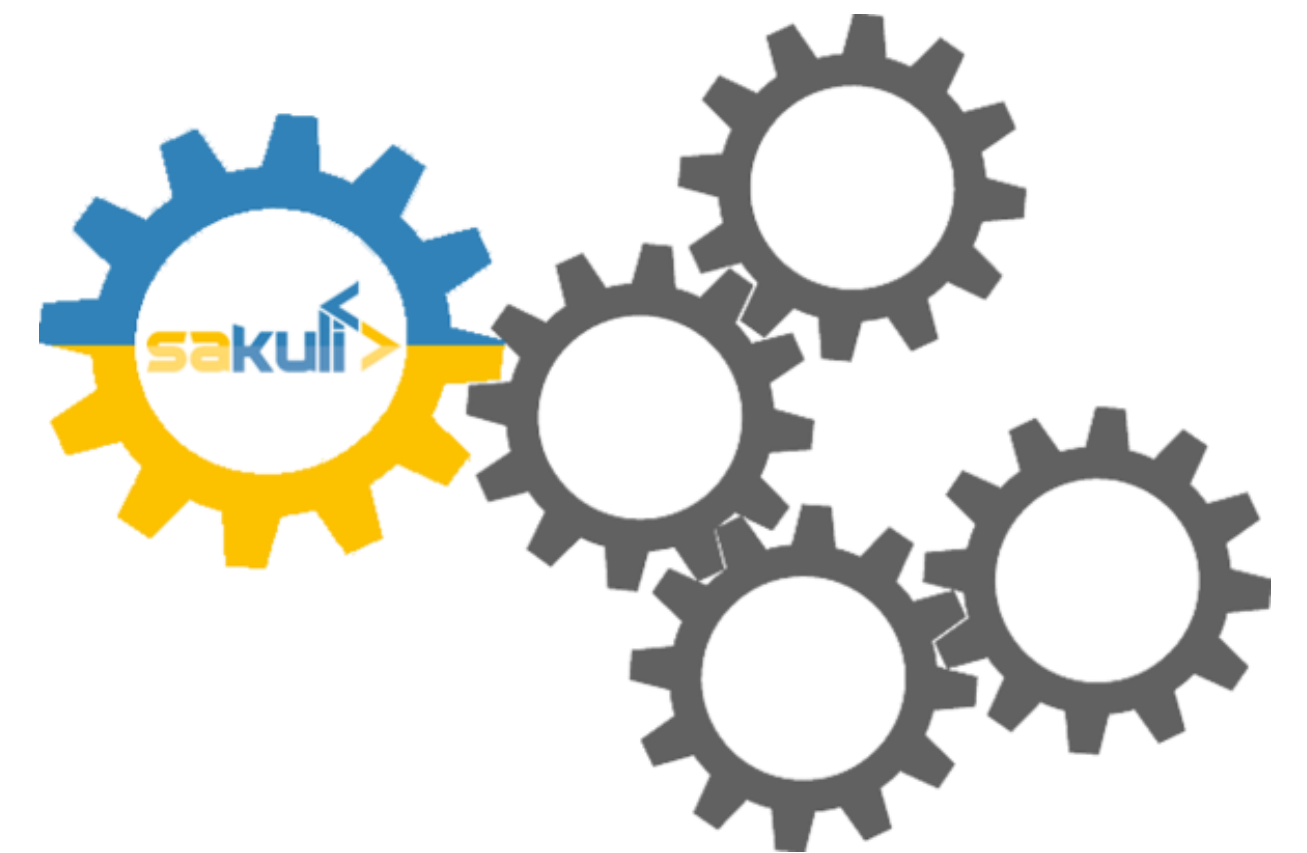
Sahi / Selenium

- ✗ limited on **web**,
(no Flash, Java applets...)
- ✓ fast through DOM navigation
- ✓ easy to write and stabilize
(Controller + Recorder)



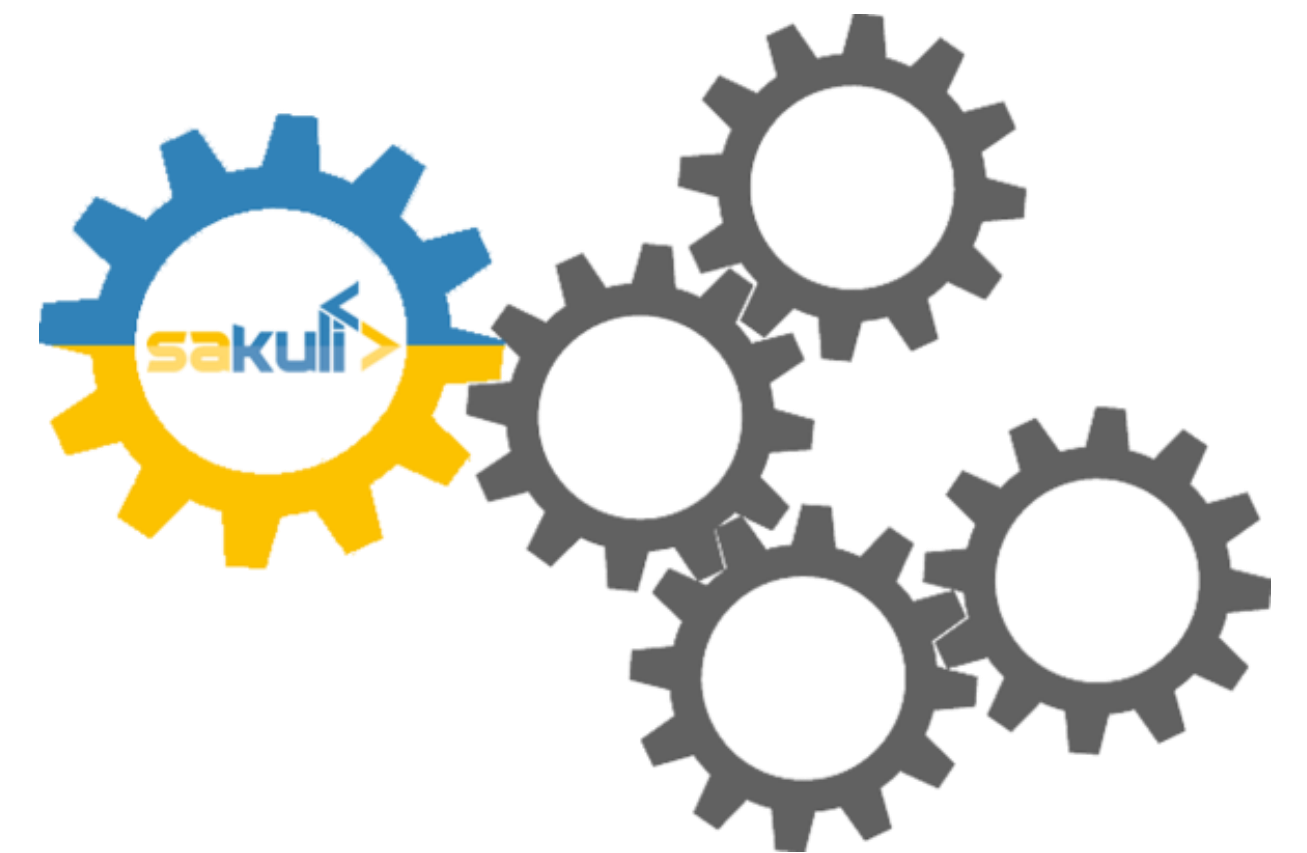
End-2-End testing with

- Various ways to execute Sakuli tests
 - Native installation
 - Maven dependency
 - Docker container



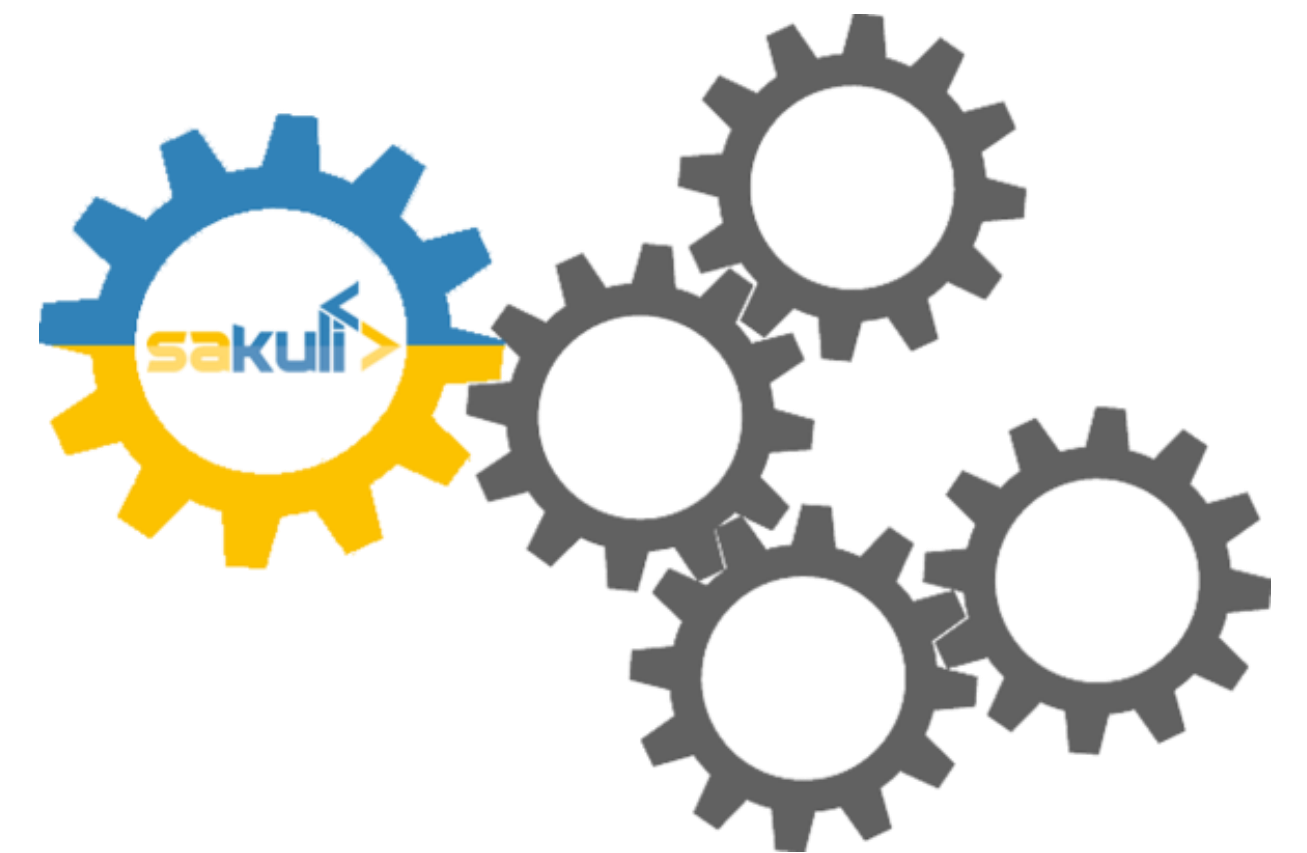
End-2-End testing with **sakuli** – Native Execution

- Supports all end user platforms: Windows, Linux, Mac
- Installable on the end user client
- Easy JavaScript based API syntax
- Execution of test scripts without compile process



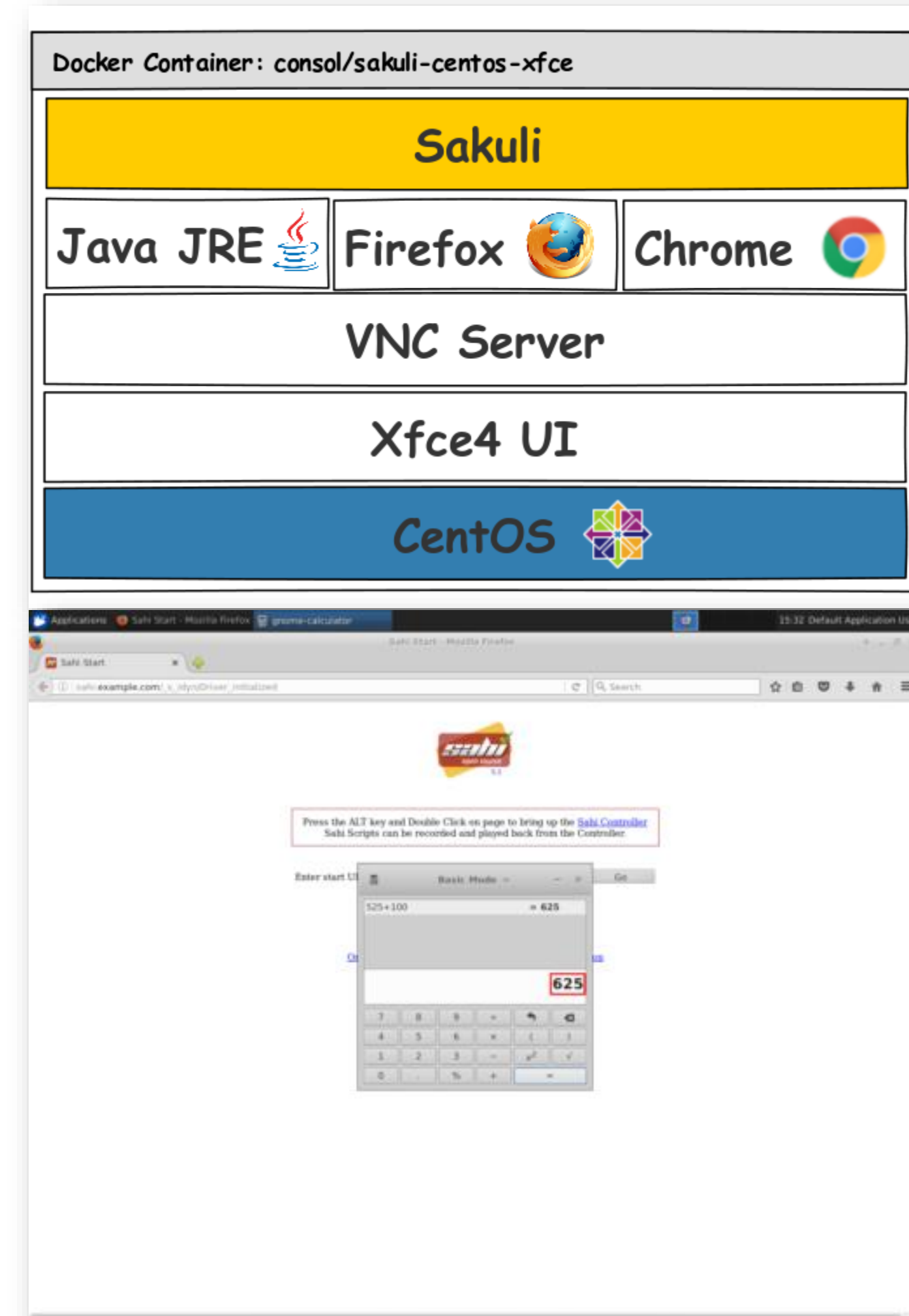
End-2-End testing with **sakuli** – Maven Execution

- Java Syntax, Maven Dependency
- Easy integration in maven build cycle
- Support through well known Java IDEs
- Optional: Integrate Selenium as Web-Testing-Engine



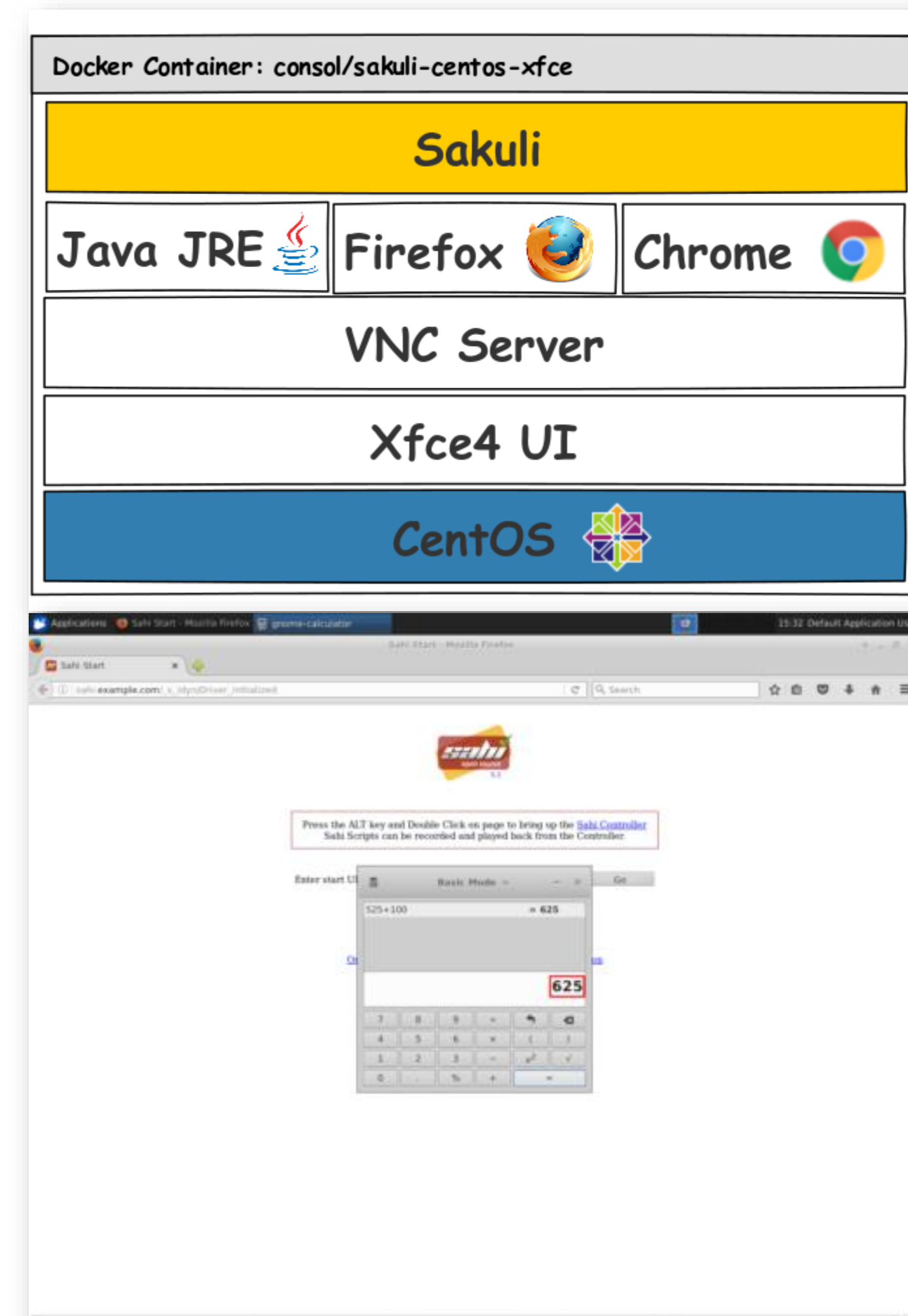
End-2-End testing with – Containerized Execution

- Supported Container platforms:
 - Docker Compose
 - Kubernetes
 - OpenShift
- Ready to use E2E environment
- Allows Java and JavaScript syntax



End-2-End testing with – Containerized Execution

- Tests with
 - a real desktop
 - a real browser
 - a native client
- Easy integration for running headless UI tests
- Scalable environment leveraging container technology



Mission

Add End-2-End tests to the pipeline

Start up your containers, gentleman!

Our goals for today

1. Take a sample app ✓
2. Create a multi stage environment on a container platform ✓
3. Build the app ✓
4. Create a pipeline to deploy the app ✓
5. Add integration tests to the pipeline ✓
6. Add End-2-End tests to the pipeline ✓



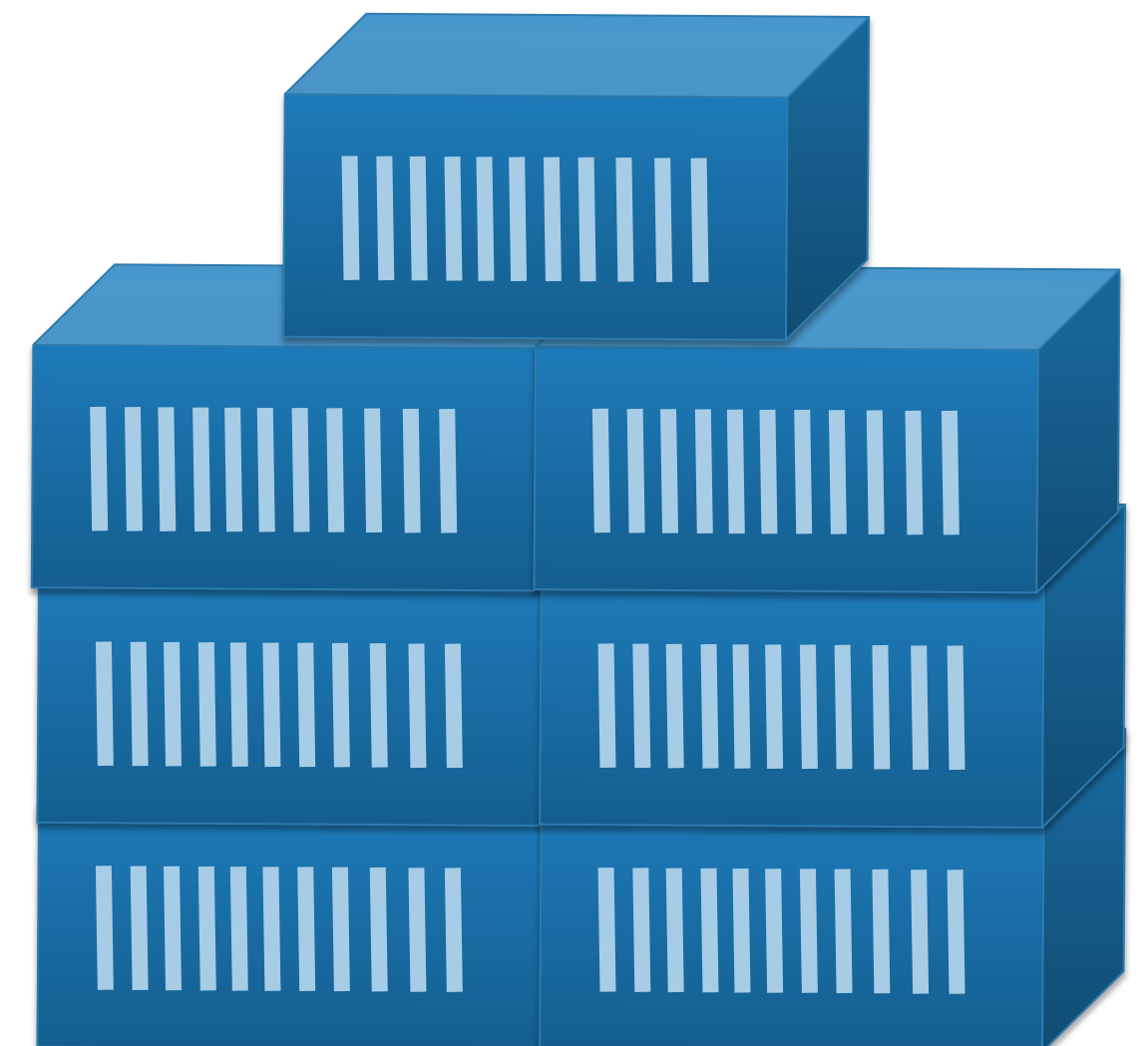
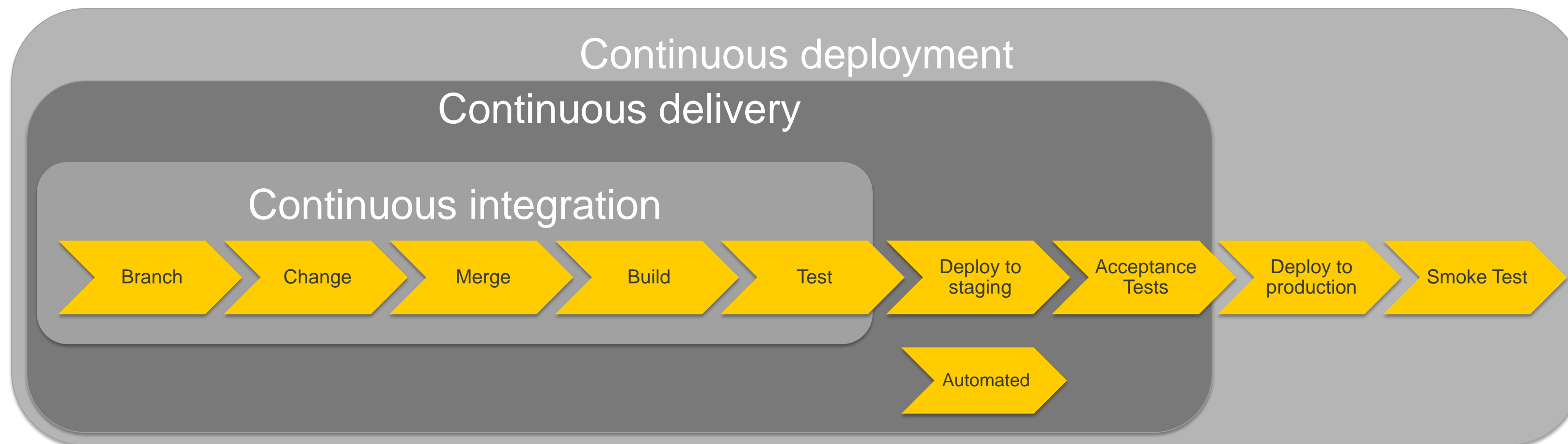
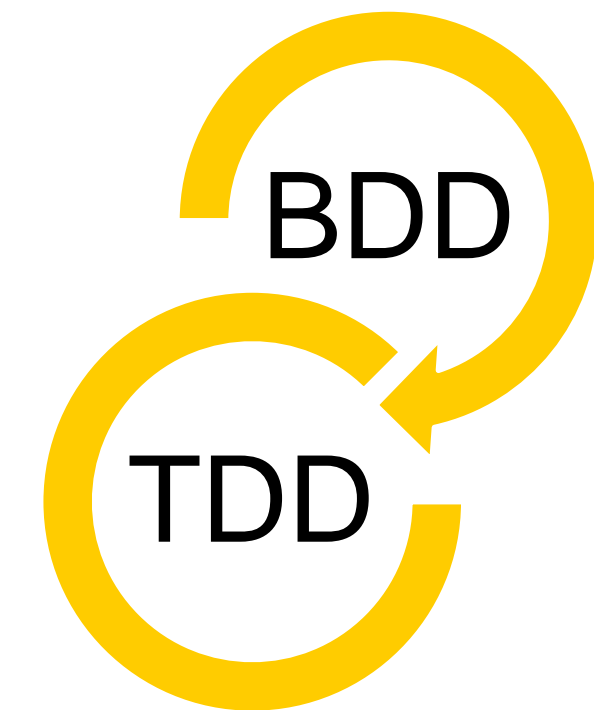
What's next

- Add monitoring – [Open Monitoring Distribution](#) (OMD)
- Add notifications on test errors/warnings
- Create issues for findings automatically



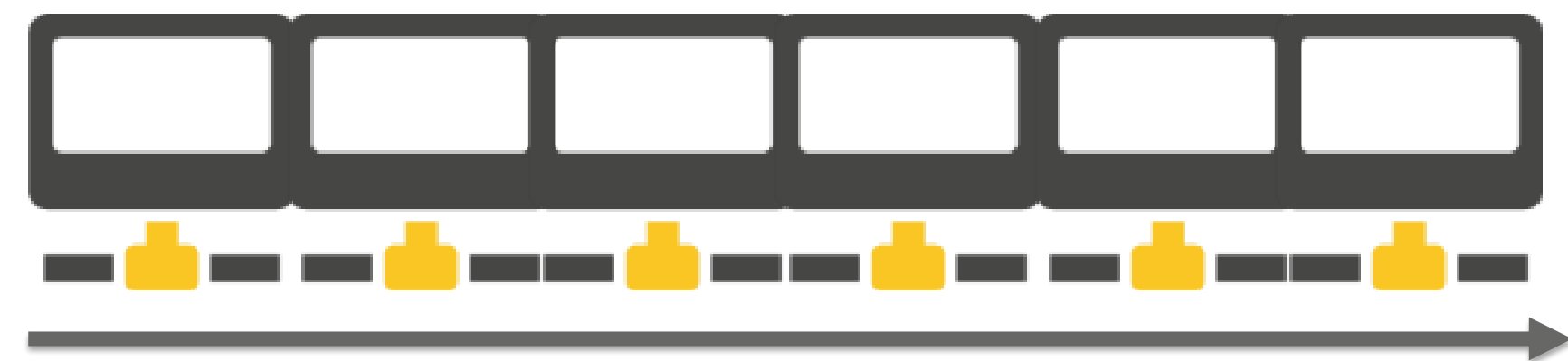
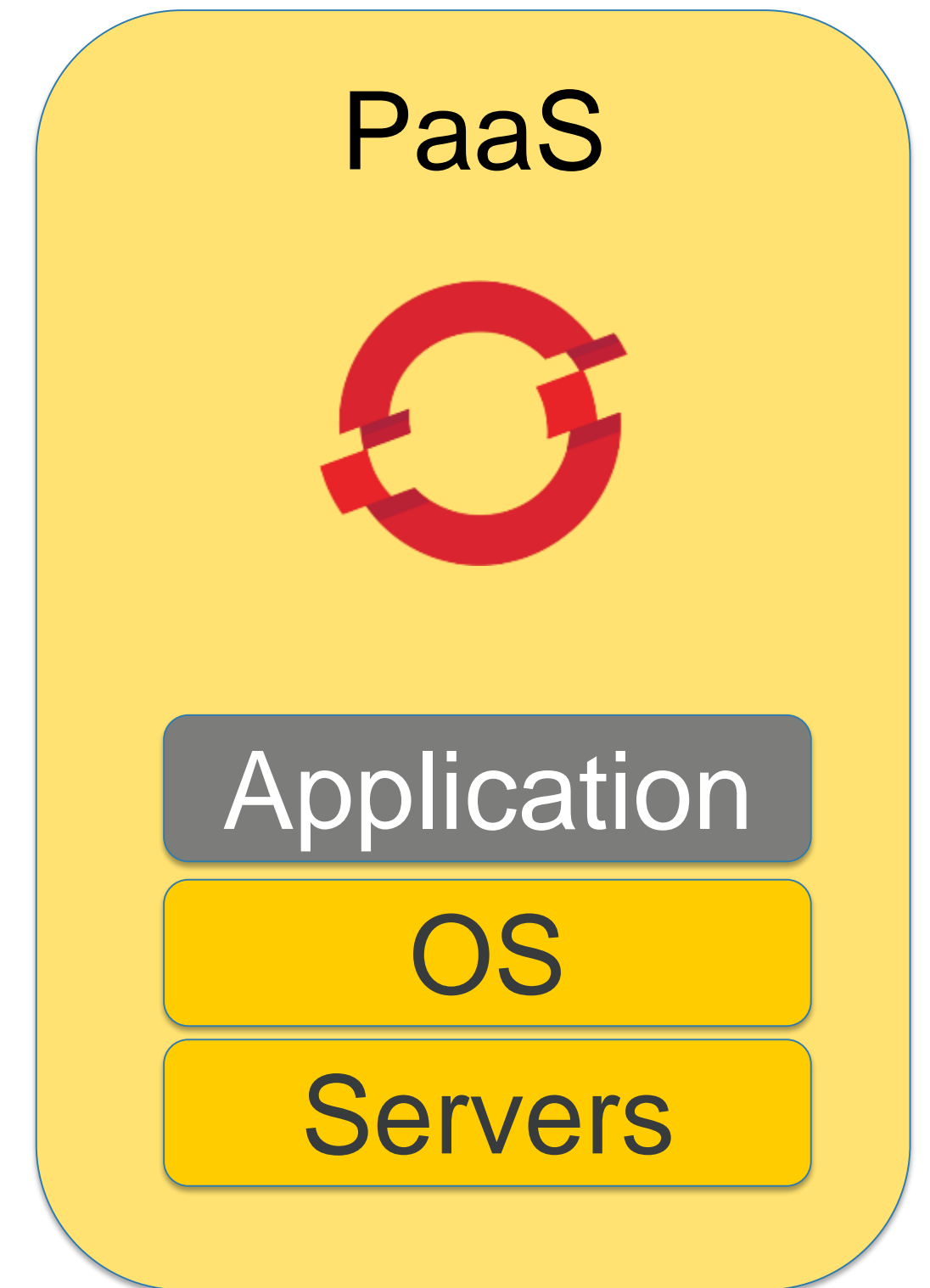
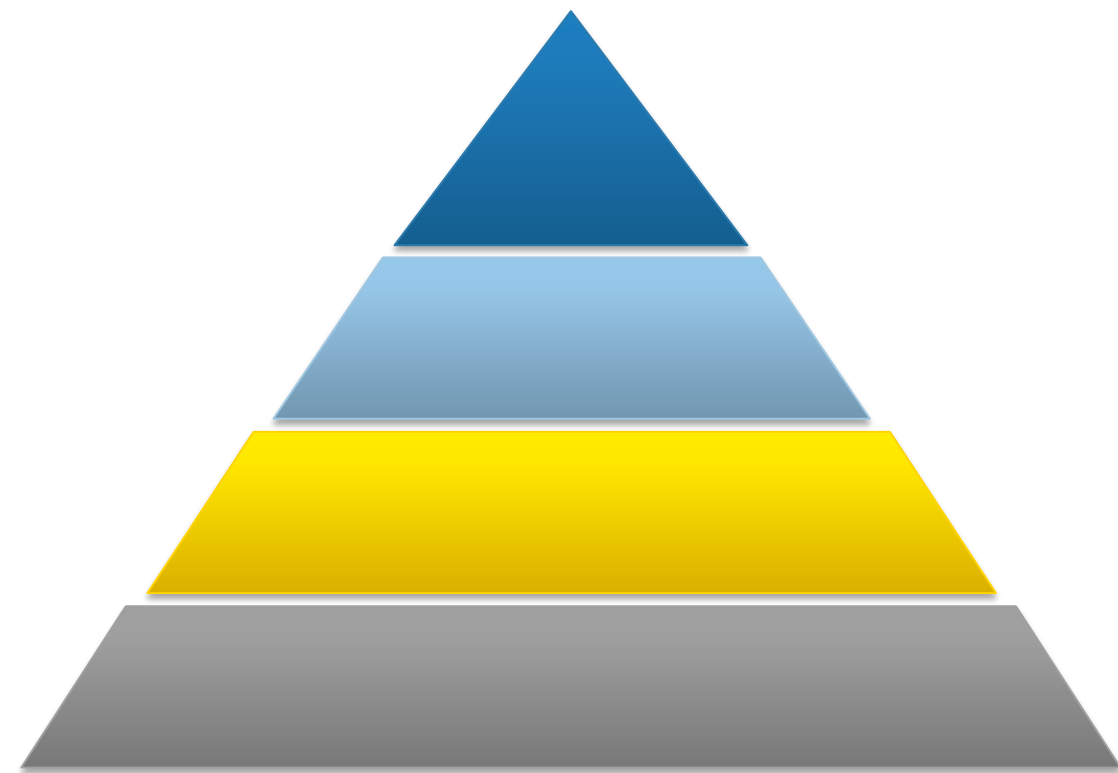
What's next

- Improve your tests with BDD
- Scale your tests with the cloud
- Extend continuous delivery to continuous deployment



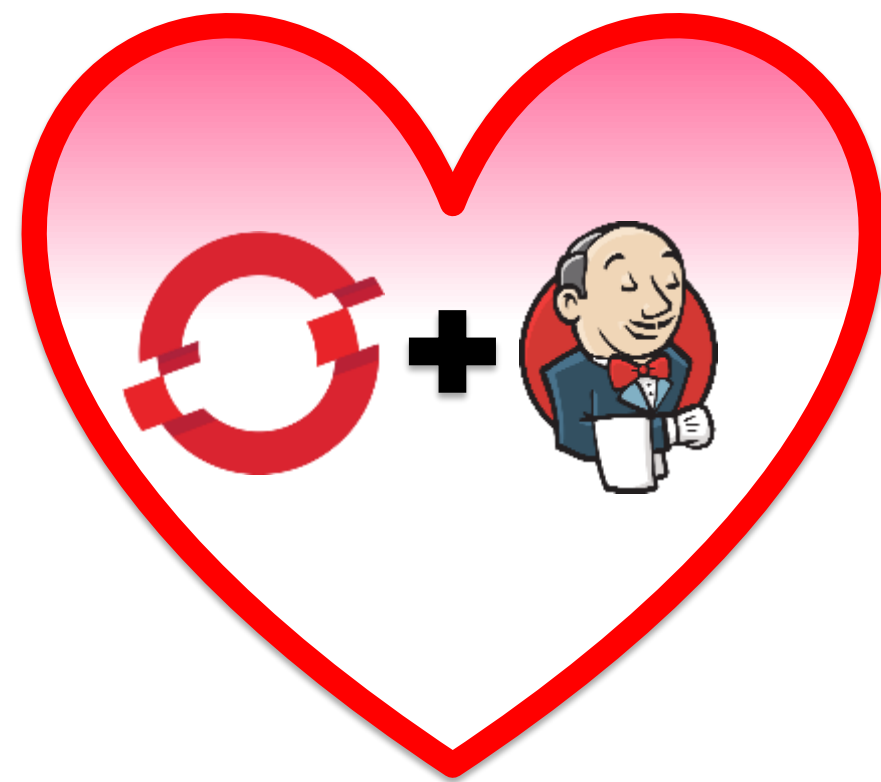
Recap

- The concepts of CI/CD pipelines
- The concepts of testing software
- The differences between various *aaS attempts
- Various OpenShift concepts like BC, DC, IS, S2I



Recap

- How to create CI/CD pipelines with OpenShift and Jenkins
- How to use the Citrus integration testing framework
- How to use the Sakuli End-2-End testing framework
- You can use everything, parts or nothing!





Questions?




Sven Hettwer
Software developer

Kanzlerstraße 8
D-40472 Düsseldorf

 @SvenHettwer

 <https://github.com/svettwer>

 Sven.Hettwer@consol.de

 +49-211-339903-86



Thank you!