Test-Driven Development with JavaFX

About us

Sven Ruppert

@SvenRuppert
www.rapidpm.org

codecentric

Hendrik Ebbers

@hendrikEbbers

www.guigarage.com

GUI Garage
OPEN SOURCE UI STUFF

canoo

Content

Basics

- Testing

Frameworks

- Testing
- Testing

CDI

Testing an application

how to test an UI component

- unit tests
- integration tests
- system tests

how to test an UI workflow

manual testing

- a tester tests the complete app
- create a test plan
- update the test plan for each release
- test each release

CI / CD

- update the test plan for each commit
- test each commit

we don't want this

UI Test Tools & Libs

IDE based Tools like Selenium

- QF-Test
- commercial product
- developer licence costs around 1995 €

Oct 2014

- no JUnit approach
- CI integration
- nearly the same as froglogic...

JemmyFX

- is for JavaFX 2.2
- last commit is over 2 years ago
- looks like there is no development activity

Automaton

- is for JavaFX2
- is developed for Java7 (> u55), is running until Java8u11
- written in Groovy
- could test Swing and JavaFX 2
- recommend TestFX for JavaFX

see homepage

MarvinFX

- https://github.com/guigarage/MarvinFX
- Provides Supervisors for JavaFX Properties

MarvinFX

define

```
PropertySupervisor<String> textSupervisor =
new PropertySupervisor<>(textfield.textProperty());
```

rules

```
textPropertySupervisor.assertWillChange();
textPropertySupervisor.assertWillChangeByDefinedCount(3);
textPropertySupervisor.assertWillChangeThisWay("A", "B", "C");
```

interaction

```
//interact with UI by using TestFX
```

check

```
textPropertySupervisor.confirm();
```

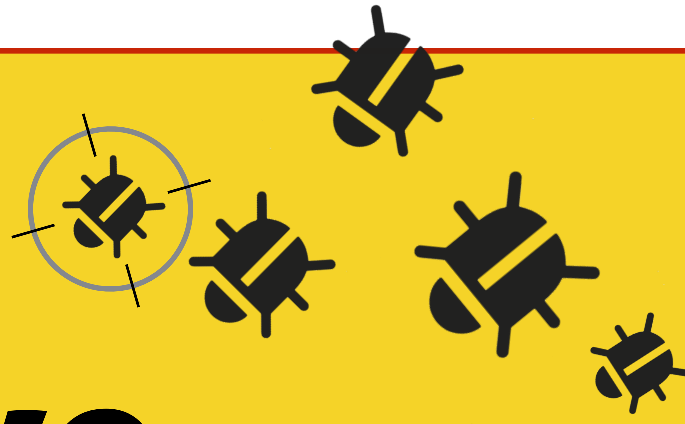
TestFX

TestFX

- active development
- LTS branch for Java7 is available
- active branch JavaFX8 only

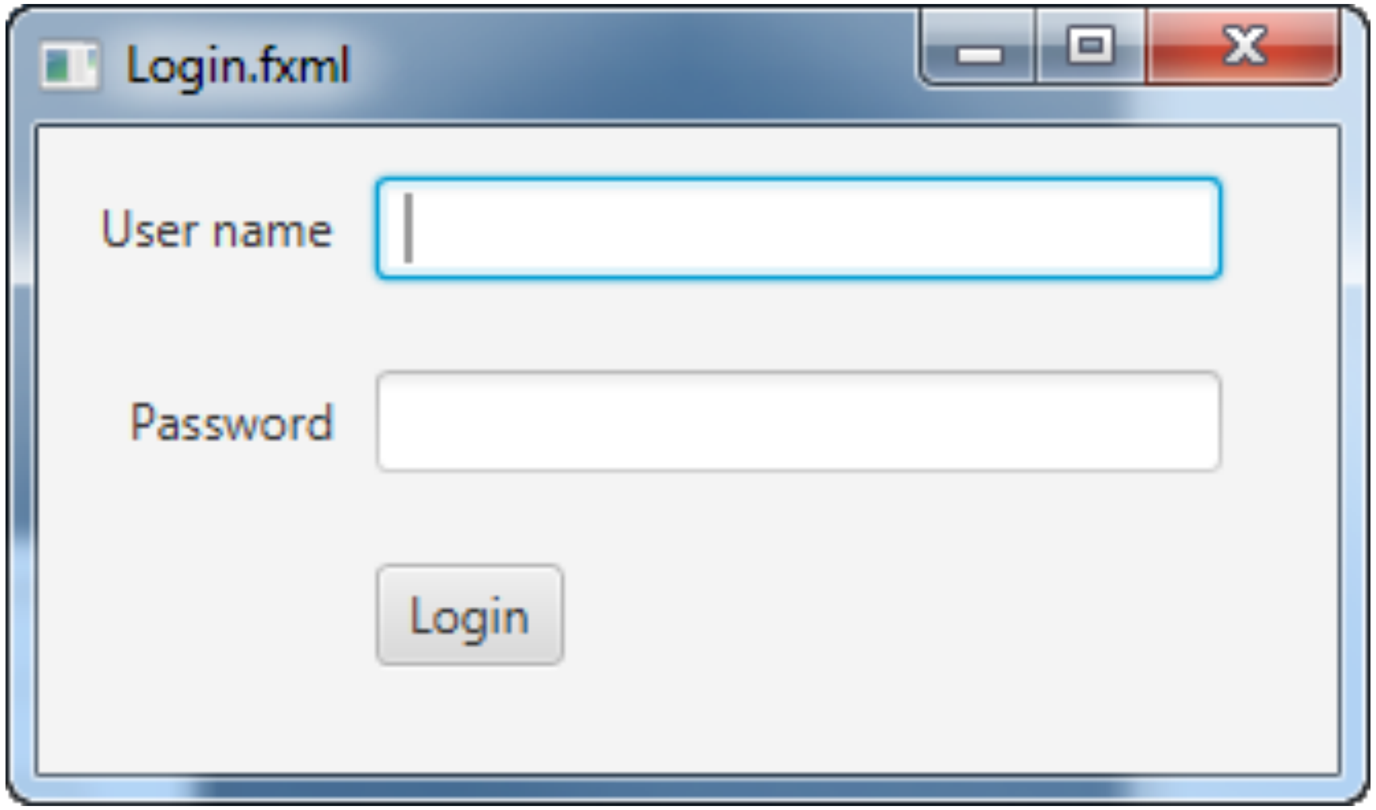
TestFX

- verifying the behavior of JavaFX applications
- API for interacting with JavaFX applications.
- fluent and clean API
- Supports Hamcrest Matchers and Lambda expressions.
- Screenshots of failed tests.



TestFX Deep Dive

Let's start with a small app



Login.fxml

User name

Password

Pseudo Code

```
click(".text-field").type("steve");
click(".password-field").type("duke4ever");
click(".button:default");

assertNodeExists( ".dialog" );
```

STEP BY STEP

- Each test must extend the `GuiTest` class

```
public class MyTest extends GuiTest {

  @Test
  public void testLogin() { . . . }

}
```

STEP BY STEP

- Provide the root node in your `GuiTest` class

```
public class MyTest extends GuiTest {

  protected Parent getRootNode() {
    . . .
  }
}
```

STEP BY STEP

- The `GuiTest` class provides a lot of functions that can be used to interact with JavaFX

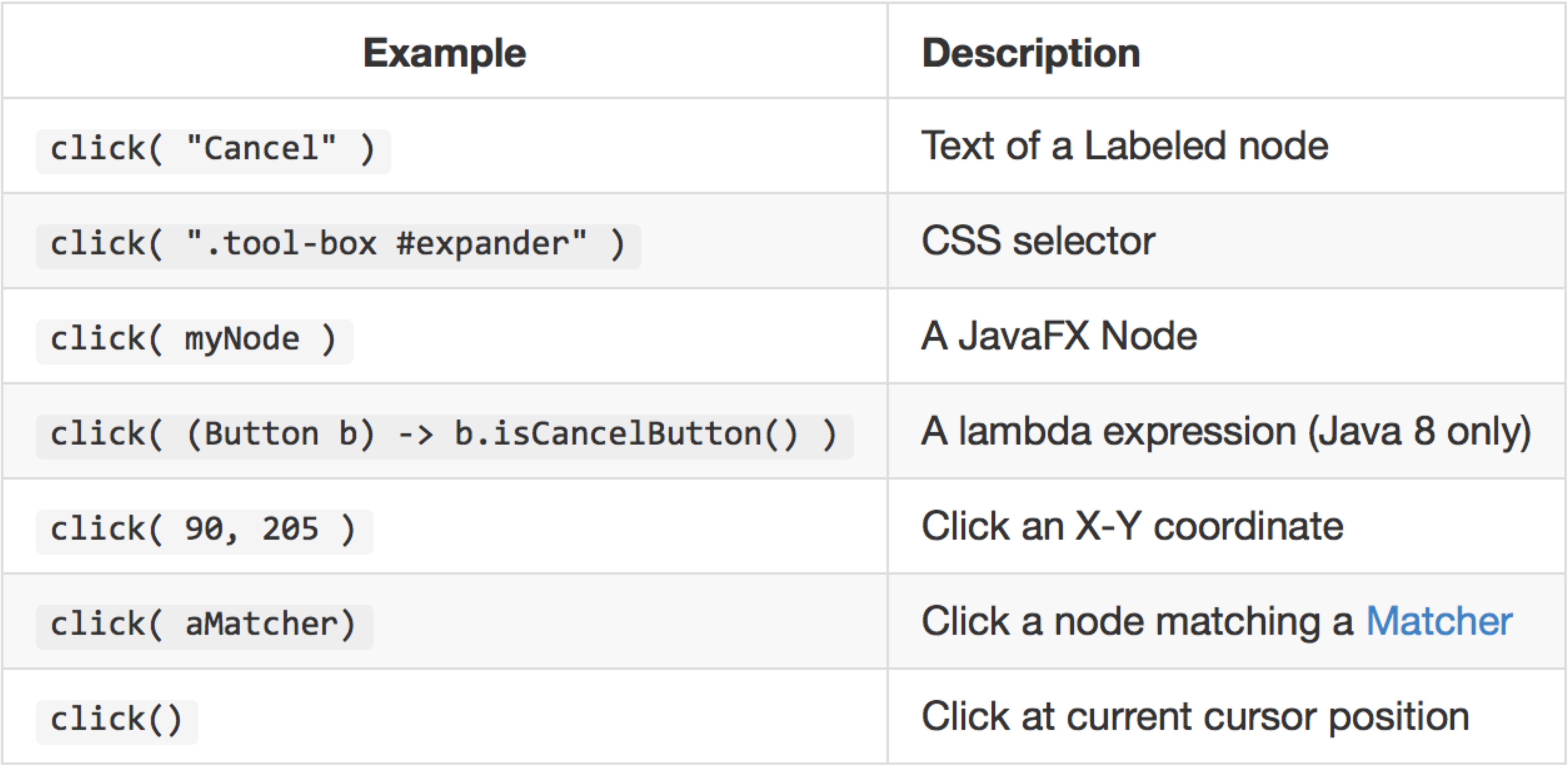
```
@Test
public void testLogin() {
  click(„.text-field“);
  type("steve");
  // . . .
}
```

STEP BY STEP

- You can use the fluent API
- You can use CSS selectors to find components

```
@Test
public void testLogin() {
  click(„#text-field“).type(„steve“);
  // . . .
}
```

How to interact with a specific node?



Example	Description
`click( "Cancel" )`	Text of a Labeled node
`click( ".tool-box #expander" )`	CSS selector
`click( myNode )`	A JavaFX Node
`click( (Button b) -> b.isCancelButton() )`	A lambda expression (Java 8 only)
`click( 90, 205 )`	Click an X-Y coordinate
`click( aMatcher)`	Click a node matching a Matcher
`click()`	Click at current cursor position

Extended Node Search

- TestFX provides additional search methods

```
find(„#name-textfield“, find(„#edit-panel“))
```

find the textfield in the subpanel

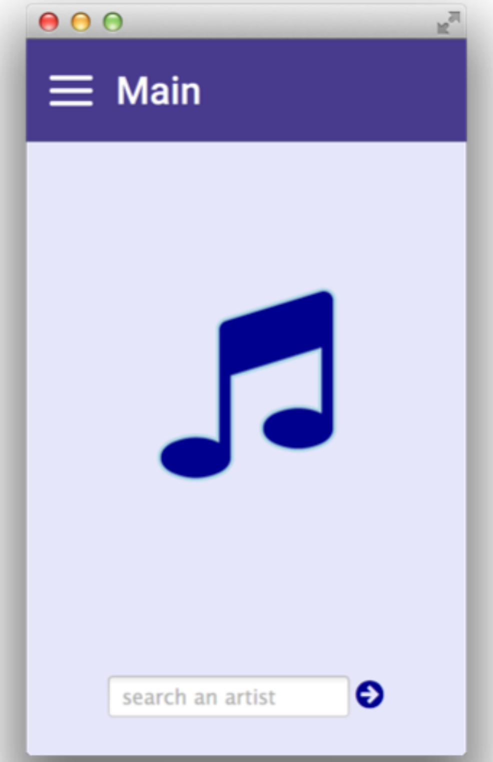
Demo

View Objects Pattern

Any Idea what this test does?

```
click("#user-field").type("steve");
click("#password-field").type("duke4ever");
click("#login-button");
click("#menu-button");
click("#action-35");
click("#tab-5");
click("#next");
click("#next");
click("#next");
click("#details");
assertNodeExists( "#user-picture" );
```

View Objects Pattern

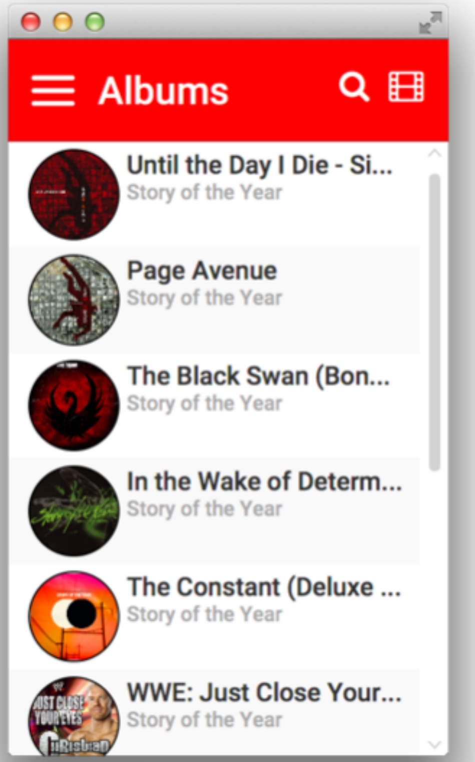


Main

search an artist

search

→



Albums

Until the Day I Die - Si...
Story of the Year

Page Avenue
Story of the Year

The Black Swan (Bon...
Story of the Year

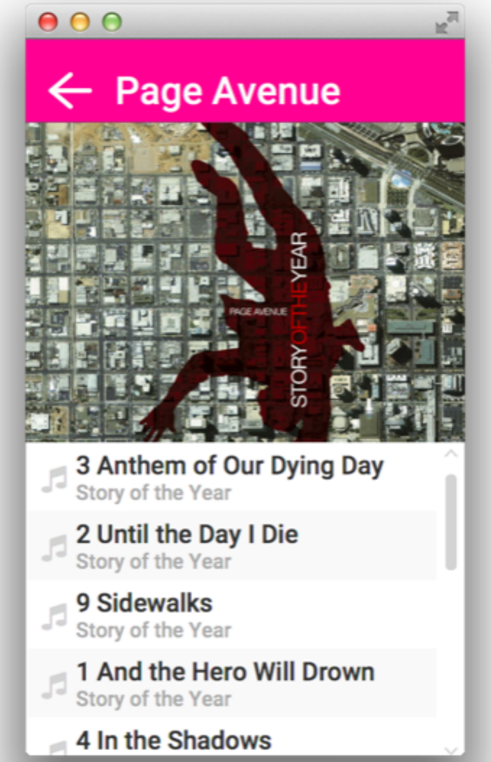
In the Wake of Determ...
Story of the Year

The Constant (Deluxe ...
Story of the Year

WWE: Just Close Your...
Story of the Year

albums overview

→



Page Avenue

3 Anthem of Our Dying Day
Story of the Year

2 Until the Day I Die
Story of the Year

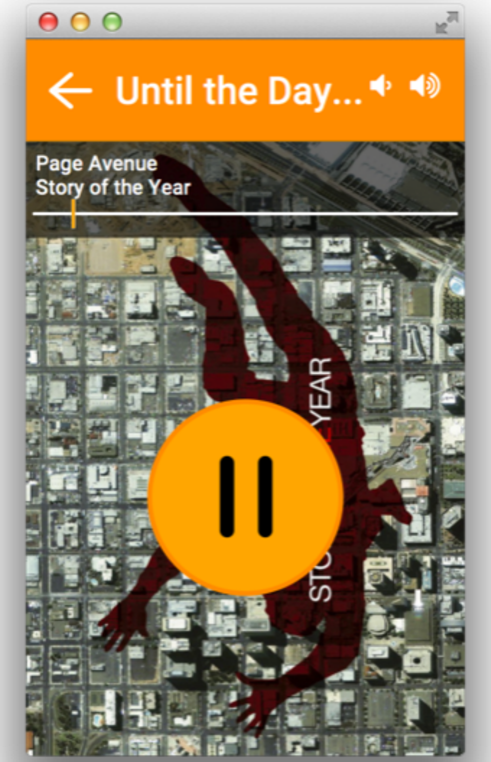
9 Sidewalks
Story of the Year

1 And the Hero Will Drown
Story of the Year

4 In the Shadows

track overview

→



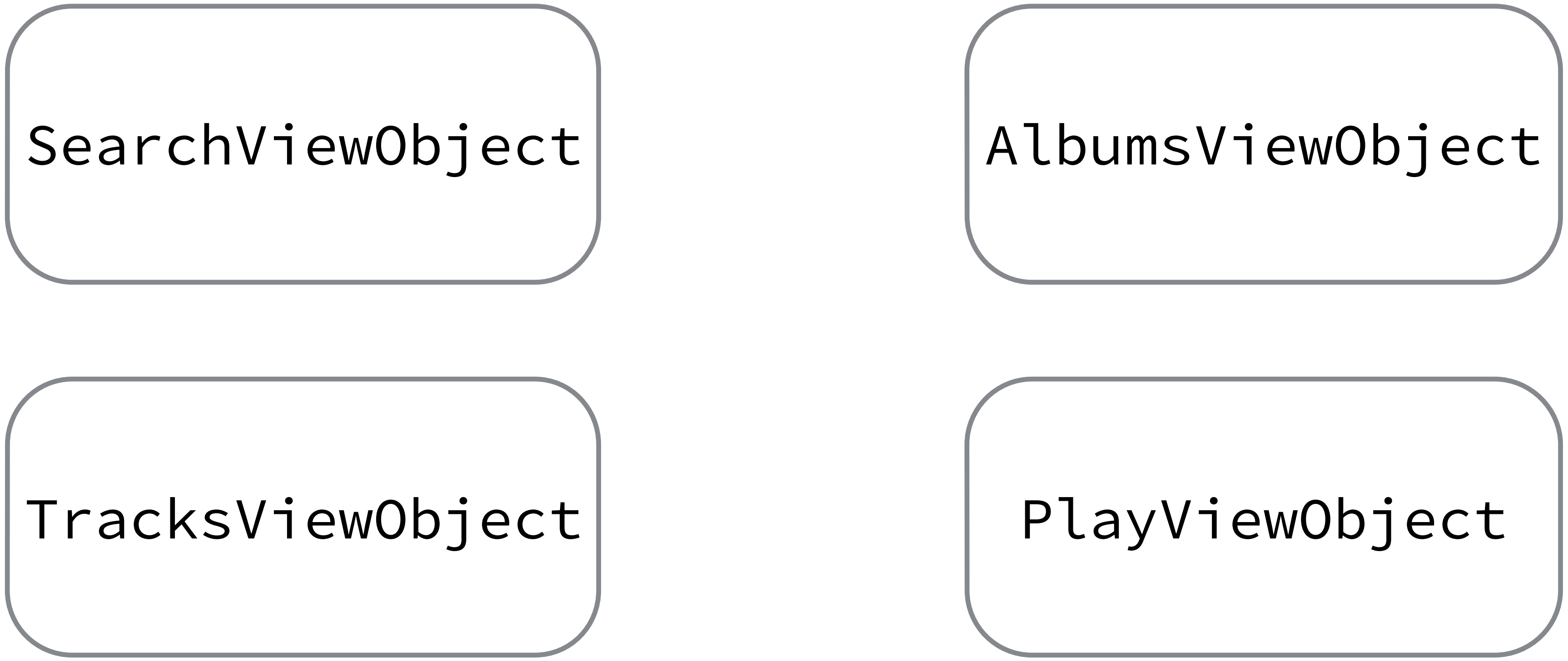
Until the Day...

Page Avenue
Story of the Year

play

View Objects Pattern

- define a class / object for each view in your application



SearchViewObject

AlbumsViewObject

TracksViewObject

PlayViewObject

STRUCTURE

- Each user interaction is defined as a method
- The class provides methods to check important states

```
public class AlbumsViewObject {

    openAlbum(String name) {}

    checkAlbumCount(int count) {}

    assertContainsAlbum(String name) {}
}
```

STRUCTURE

- Each method returns the view object for the page that is visible after the method has been executed

- If the view won’t change by calling a method the method will return “this“

```
public TracksViewObject openAlbum(String name) {
  click((Text t) -> t.getText().contains(name));
  return new TracksViewObject(getTestHandler());
}

public AlbumsViewObject checkAlbumCount(int count) {
  assertEquals(count, getList().size());
  return this;
}
```

WRITE READABLE TESTS

```
@Test
public void checkTrackCount() {
  new SearchView(this).
  search("Rise Against“).
  openAlbum("The Black Market“).
  checkTrackCountOfSelectedAlbum(12);
}
```

Demo

Testing DataFX Flow

```
public class Tests extends FlowTest {

  protected Class<?> getFlowStartController() {
    return SearchController.class;
  }

  @Test
  public void testSearch() {
    click(„#searchfield“) . . .
  }
}
```

Injection

Problem

```
@FlowScoped
public class ITunesDataModel {

  public void search(String artist) {
    //REST call
  }

}
```

extend the class

```
@FlowScoped
public class TestDataModel extends ITunesDataModel
{

  public void search(String artist) {
    getAlbums().add(. . .);
    //Adding test data
  }

}
```

Solution for DataFX

```
public class Tests extends FlowTest {

@Override
    protected void injectTestData(Injector injector) {
        injector.inject(new TestDataModel(),
                     ITunesDataModel.class);
    }

}
```

Demo

Testing Afterburner.fx

- apache licensed
  as lean as possible: 3 classes, no external dependencies

- combines: FXML, Convention over Configuration and JSR-330 / @Inject

- integrated with maven 3

Testing Afterburner.fx

- under active development
- injection over a few steps is working
- postconstruct is working
- using existing CDI Services with Annotations (Scopes and so on) is not working with afterburner
- no mixed mode with CDI and afterburner.fx

Testing Afterburner.fx

- TestFX is working fine with afterburner.fx
- Definition of the tests are the same as without afterburner.fx

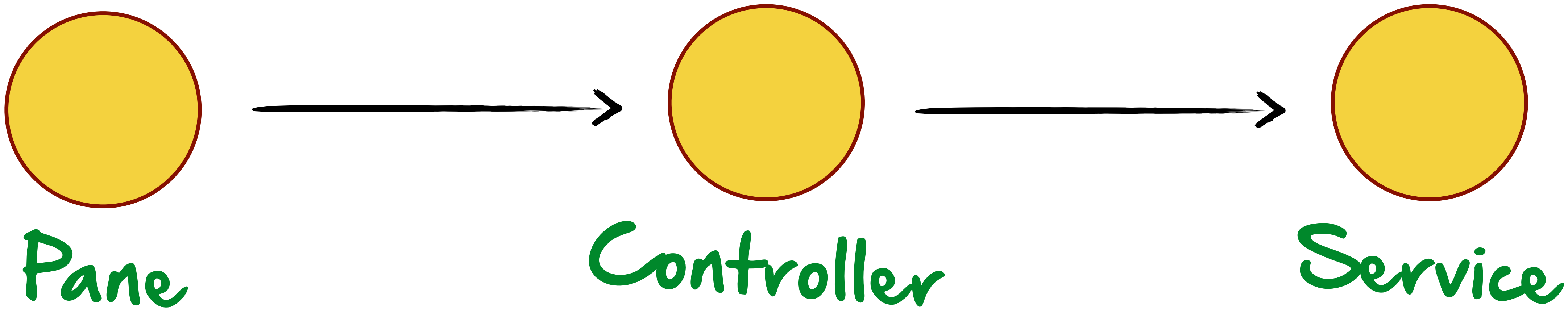
Demo Afterburner.fx

TestFX & CDI

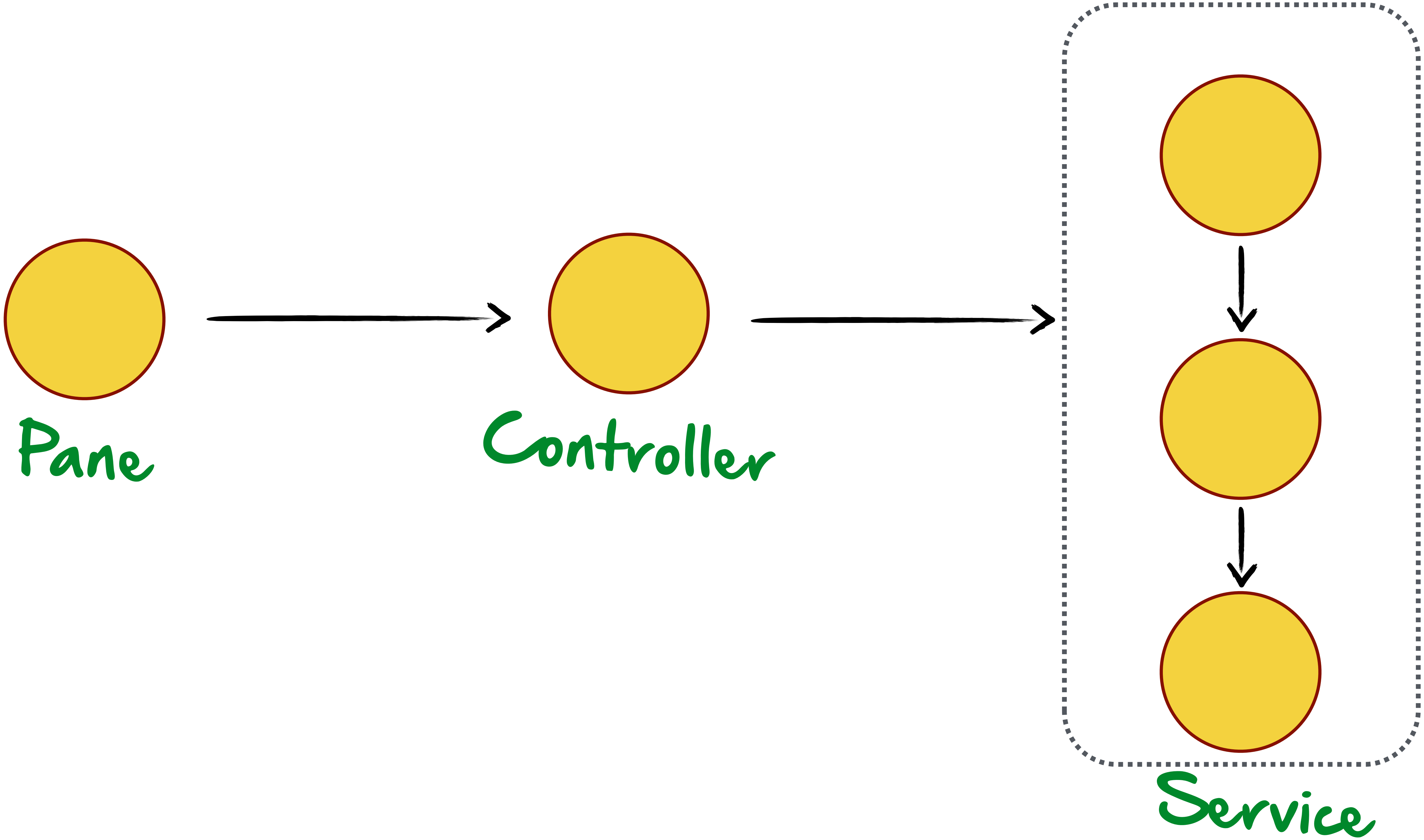
Why CDI?

- Because we want to use Mocks
- Dynamic reconfiguration

Example



Example



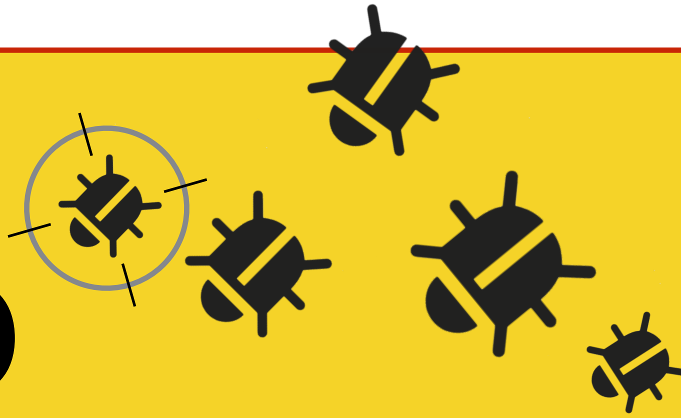
Example

```
Service myService = new Service();
```

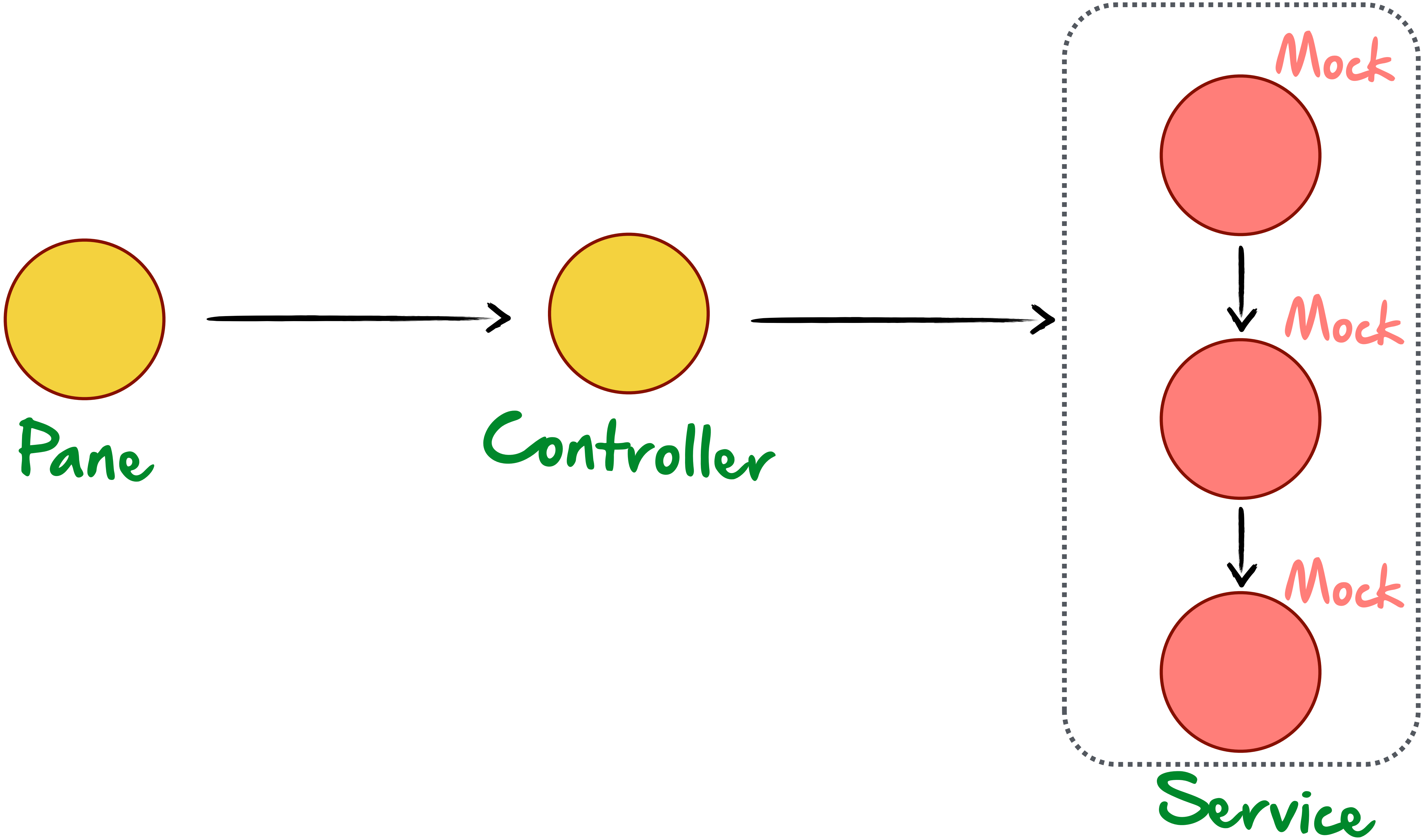
↓

```
@Inject Service myService;
```

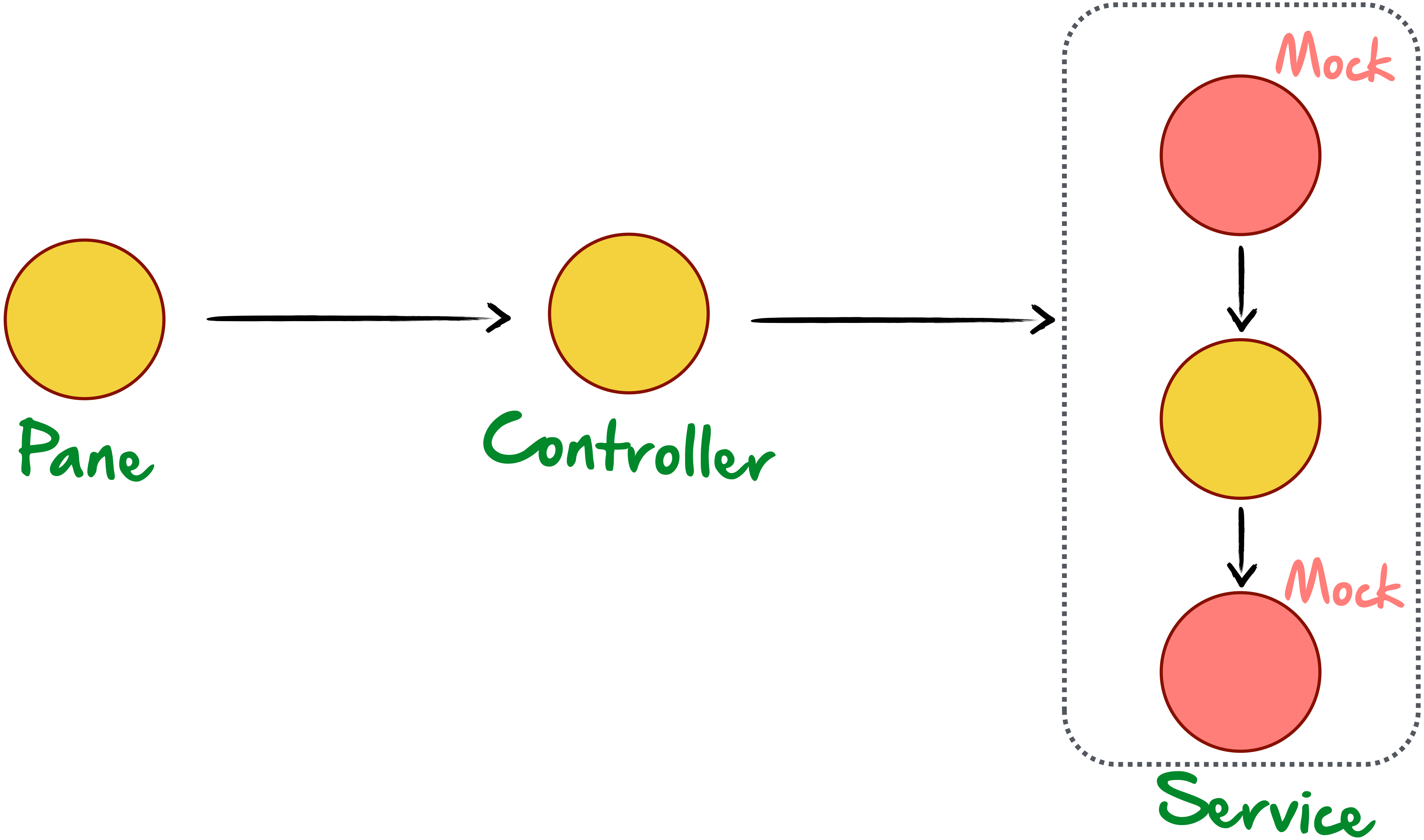
Plain FX Demo



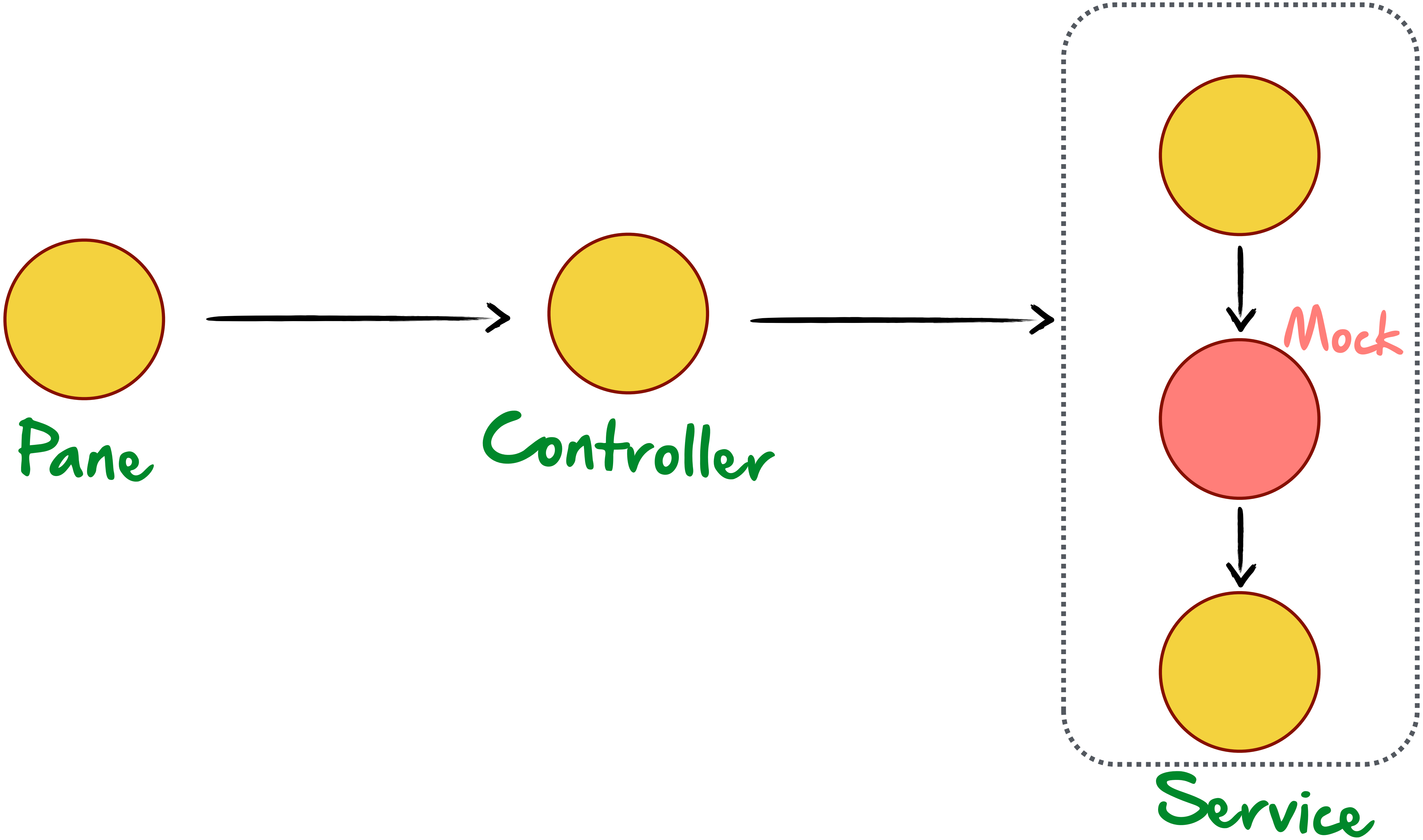
Example



Example



Example



CDI Basic Pattern

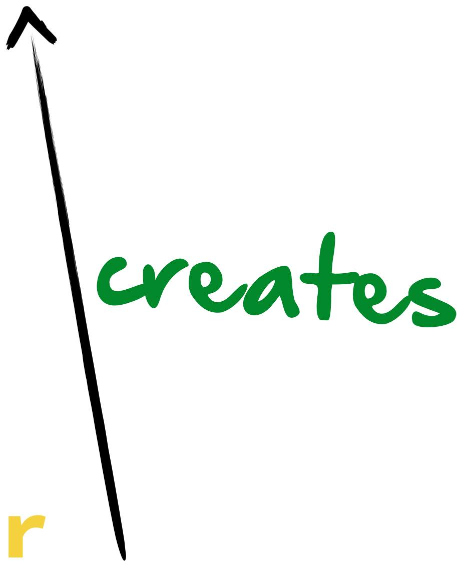
- The production source must not contain test sources

- Therefore we need to decouple test & production sources

physically!

CDI Basic Pattern

```
@Inject @MyQualifier
ServiceInterface myService;
```

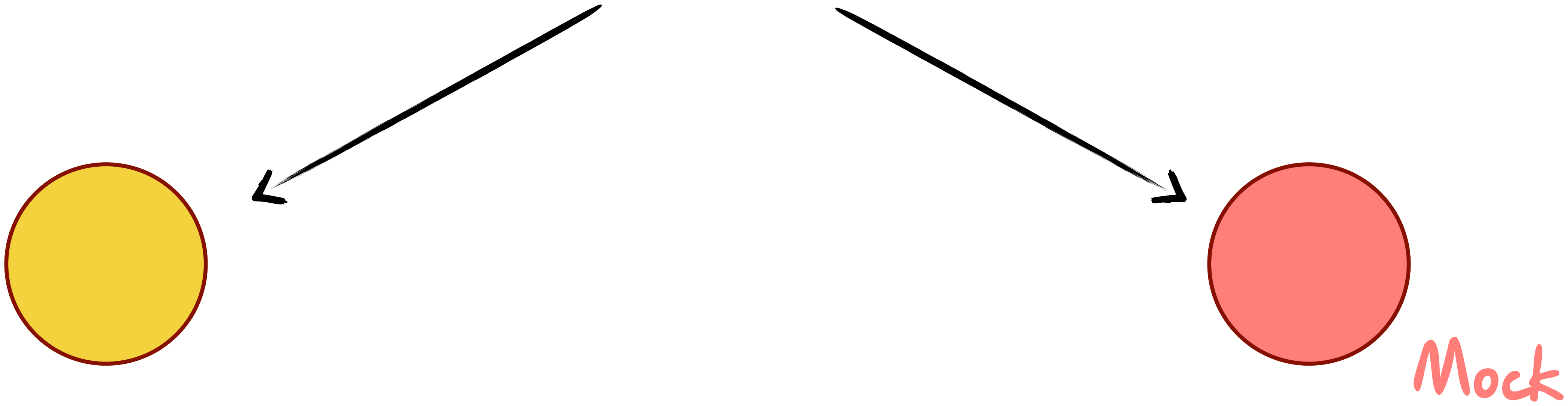


```
@Producer @MyQualifier
ServiceInterface createService(){. . .}
```

CDI Basic Pattern

```
@Inject @MyQualifier
ServiceInterface myService;

@Producer @MyQualifier
ServiceInterface createService(){. . .}
```



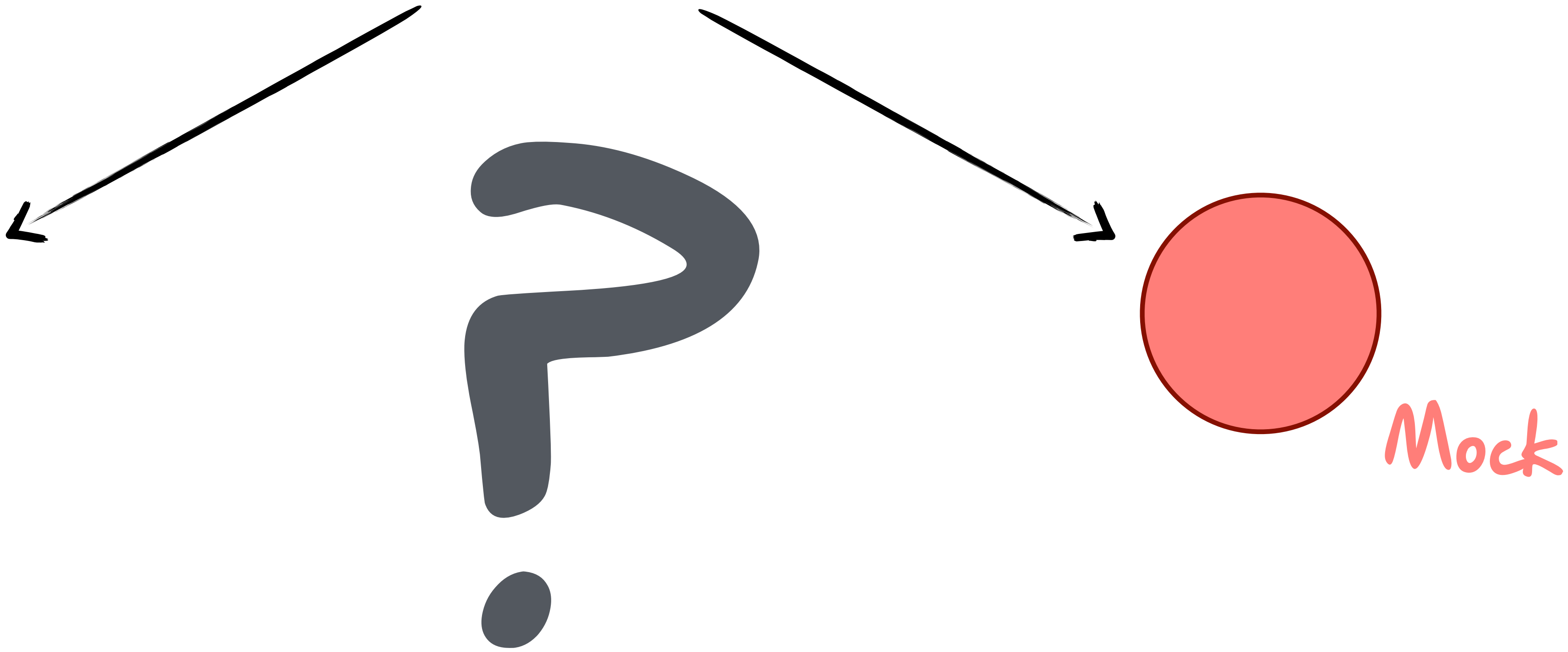
CDI Basic Pattern

```
if(„production“) {
   return service;
} else {
   return mock;
}
```

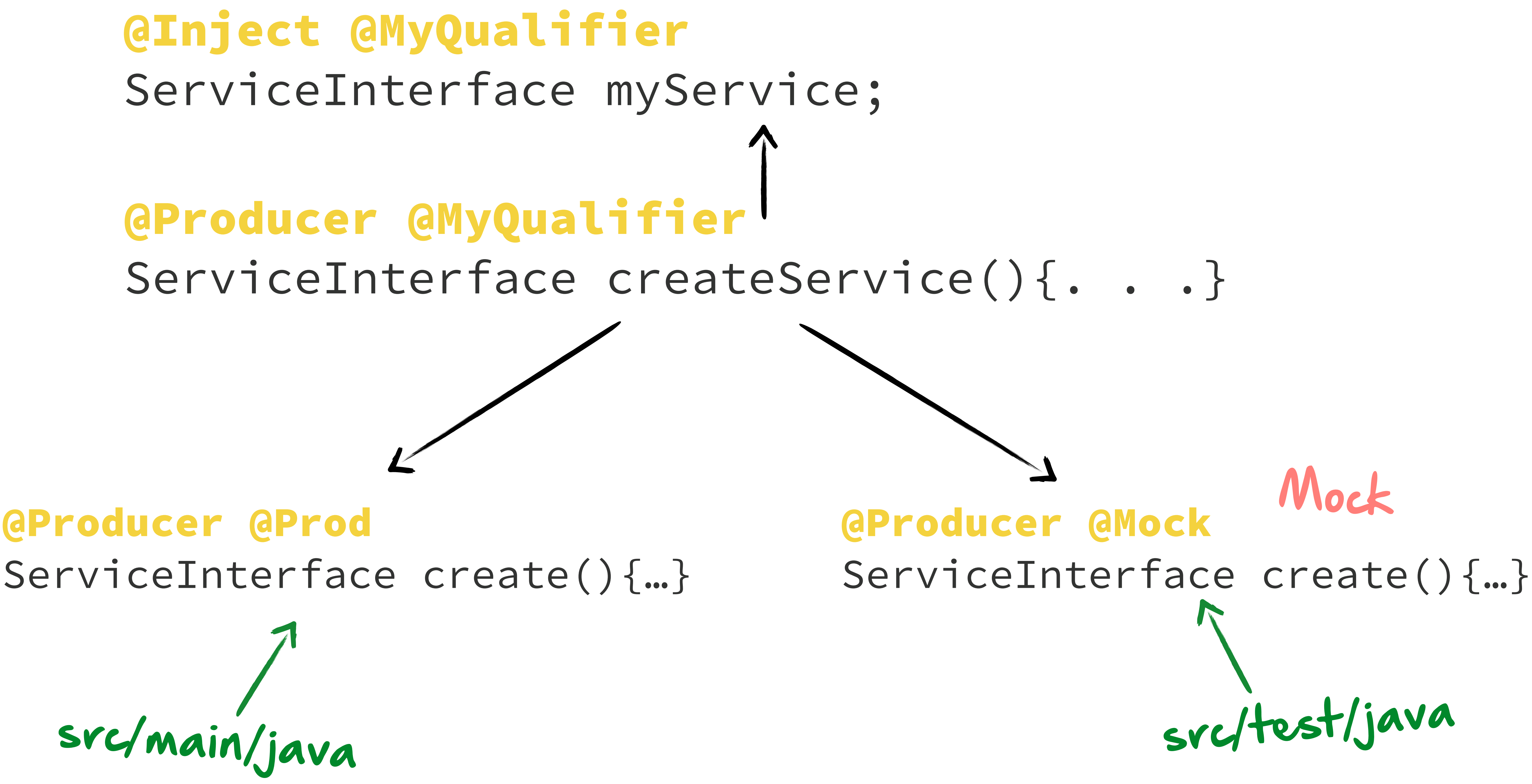
CDI Basic Pattern

```
@Inject @MyQualifier
ServiceInterface myService;

@Producer @MyQualifier
ServiceInterface createService(){. . .}
```



CDI Basic Pattern



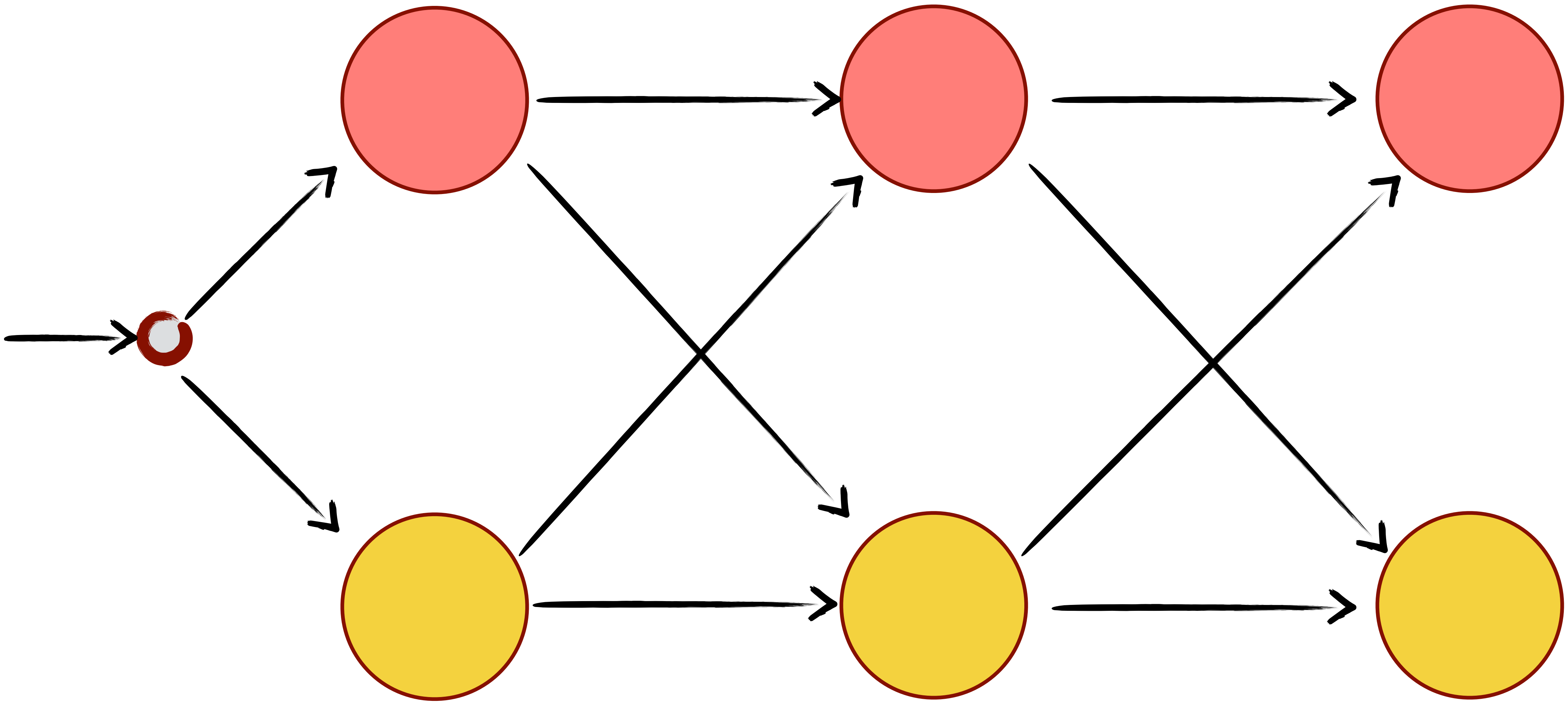
```
@Inject @MyQualifier
ServiceInterface myService;
```

```
@Producer @MyQualifier
ServiceInterface createService(){. . .}
```

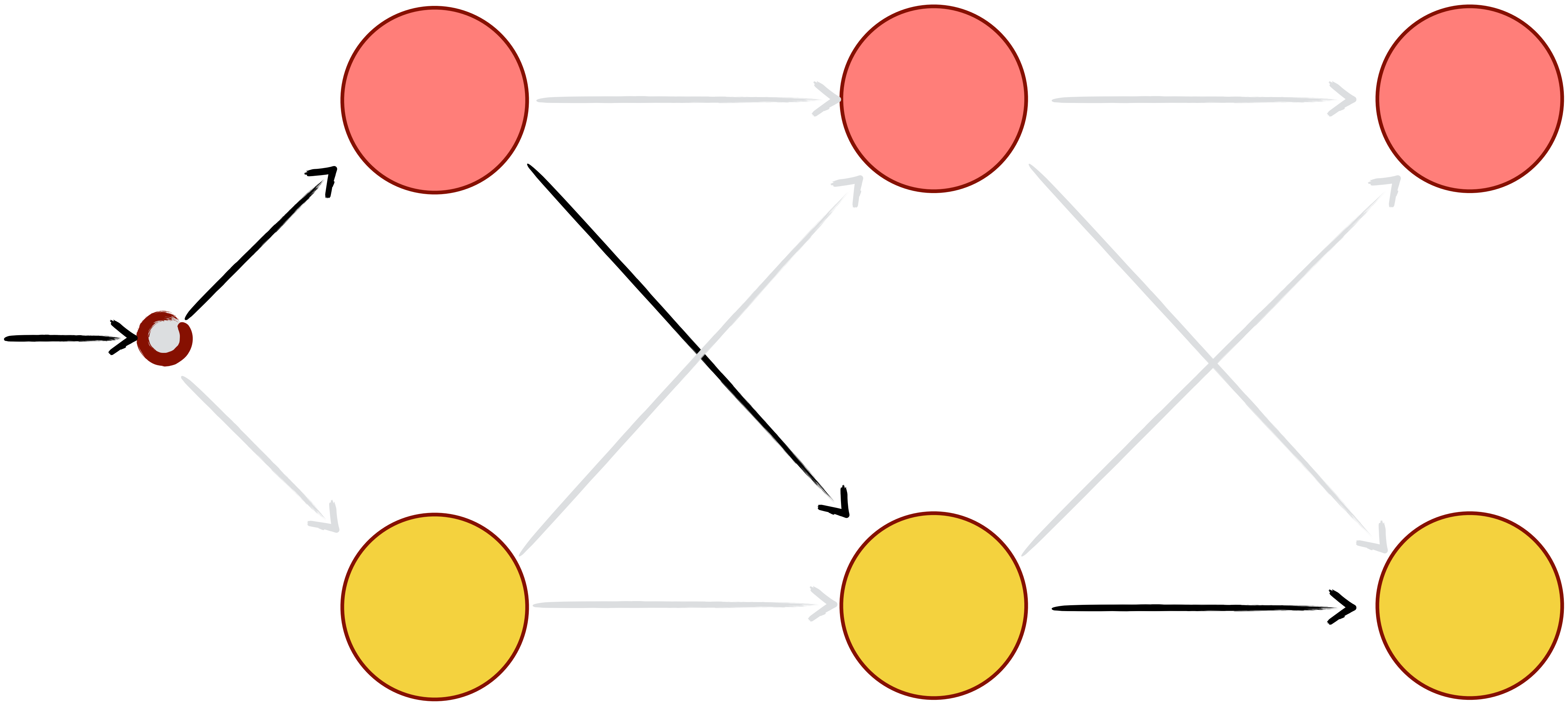
```
@Producer @Prod
ServiceInterface create(){…}
```

```
@Producer @Mock
ServiceInterface create(){…}
```

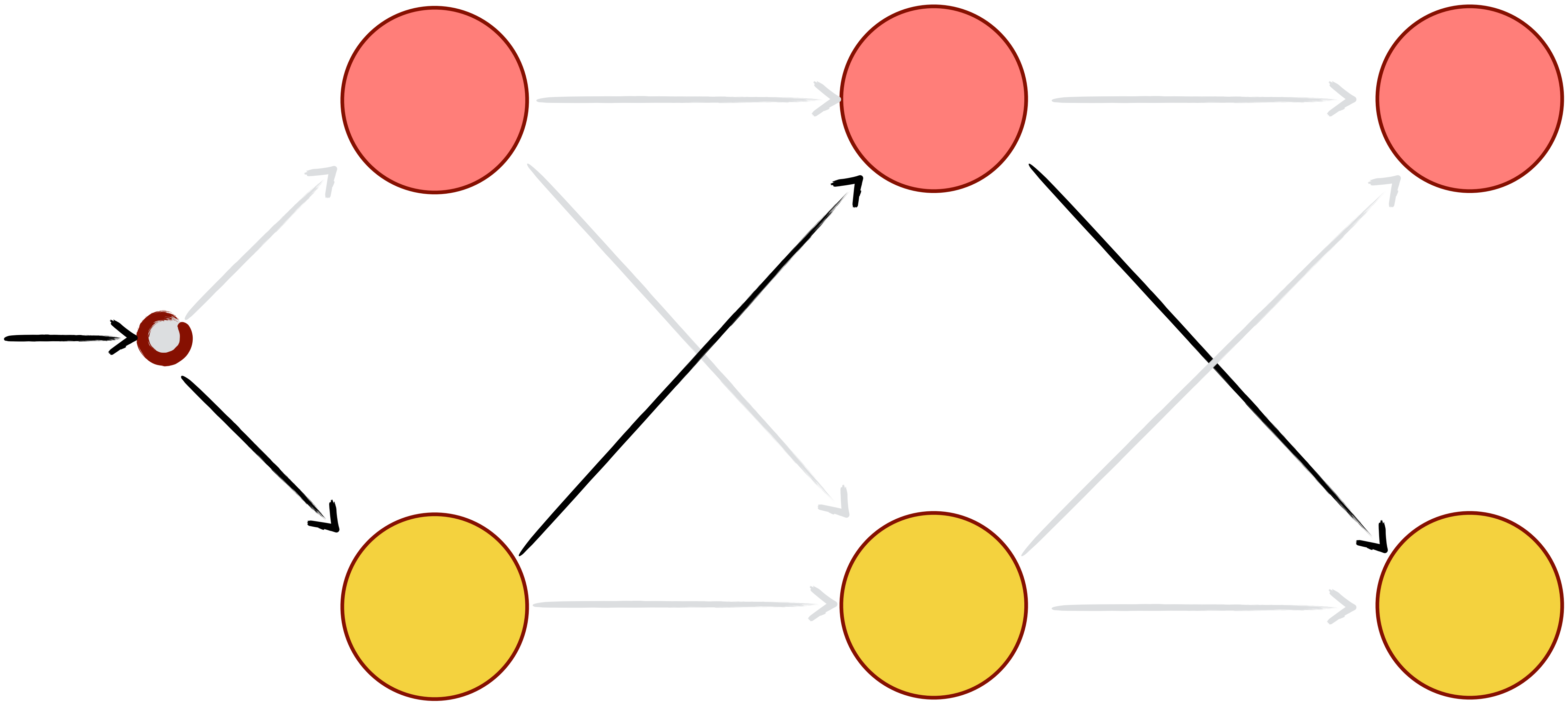
CDI Basic Pattern



CDI Basic Pattern



CDI Basic Pattern



CDI Demo

Where to start?

- www.rapidpm.org
- github.com/svenruppert/javaone2014

QA