Getting Started with Mockito

Nicolae Caprarescu
FULL-STACK SOFTWARE DEVELOPMENT CONSULTANT

www.properjava.com

Overview

A few words on unit testing

Demo application overview

Write a unit test without mocks

Install Mockito 2

Edit the unit test to use a mock

Unit testing and mocking in more detail

An overview of Test Doubles

Course Outline

Getting started with Mockito

Configuring return values of mocked methods

Verifying that mocked methods behave accordingly

Using partial mocks and advanced mocking techniques

Using Mockito's behavior-driven development support

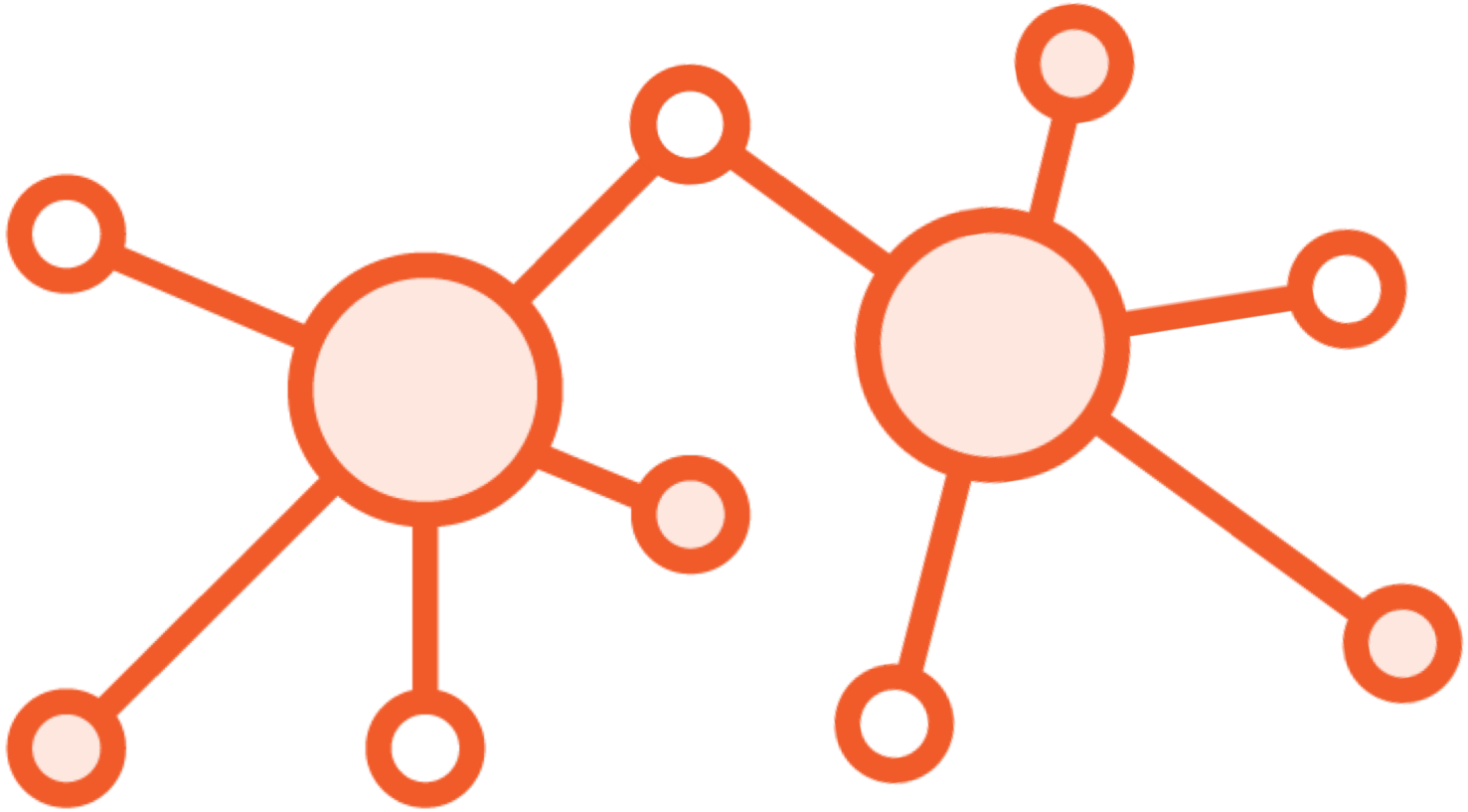
Suggested Prerequisites

http://bit.ly/psjunit

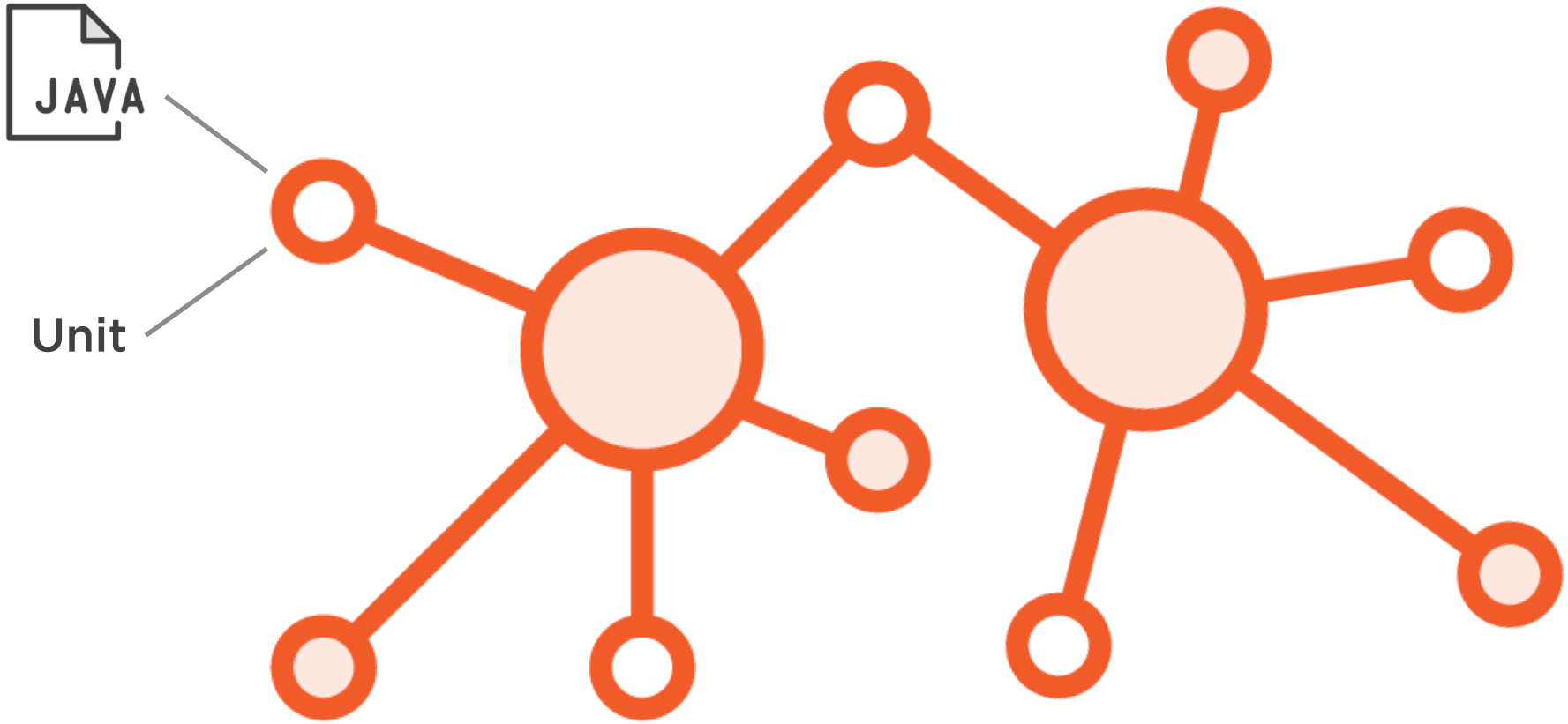
or

http://bit.ly/psjunit5

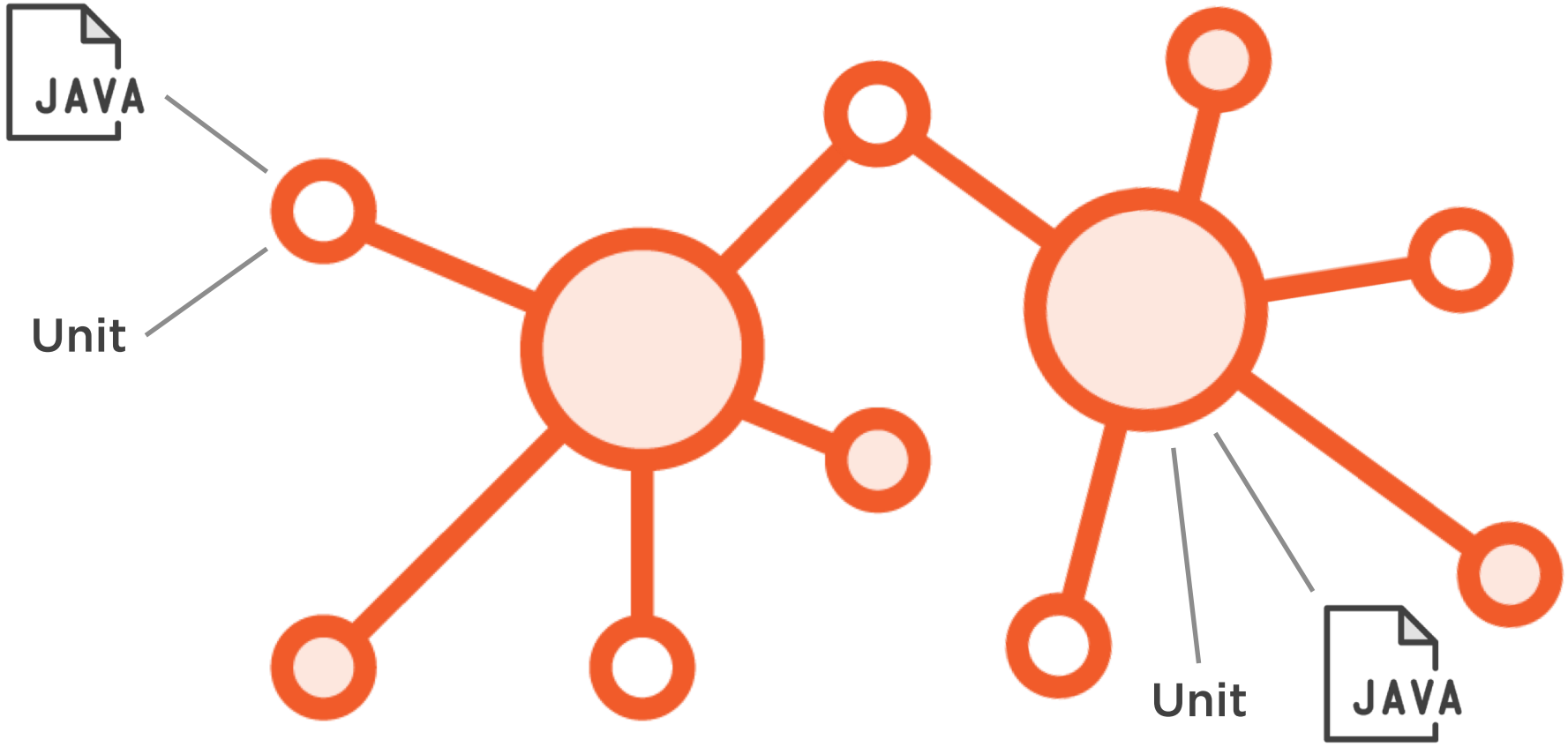
Unit Testing



Unit Testing



Unit Testing



Demo

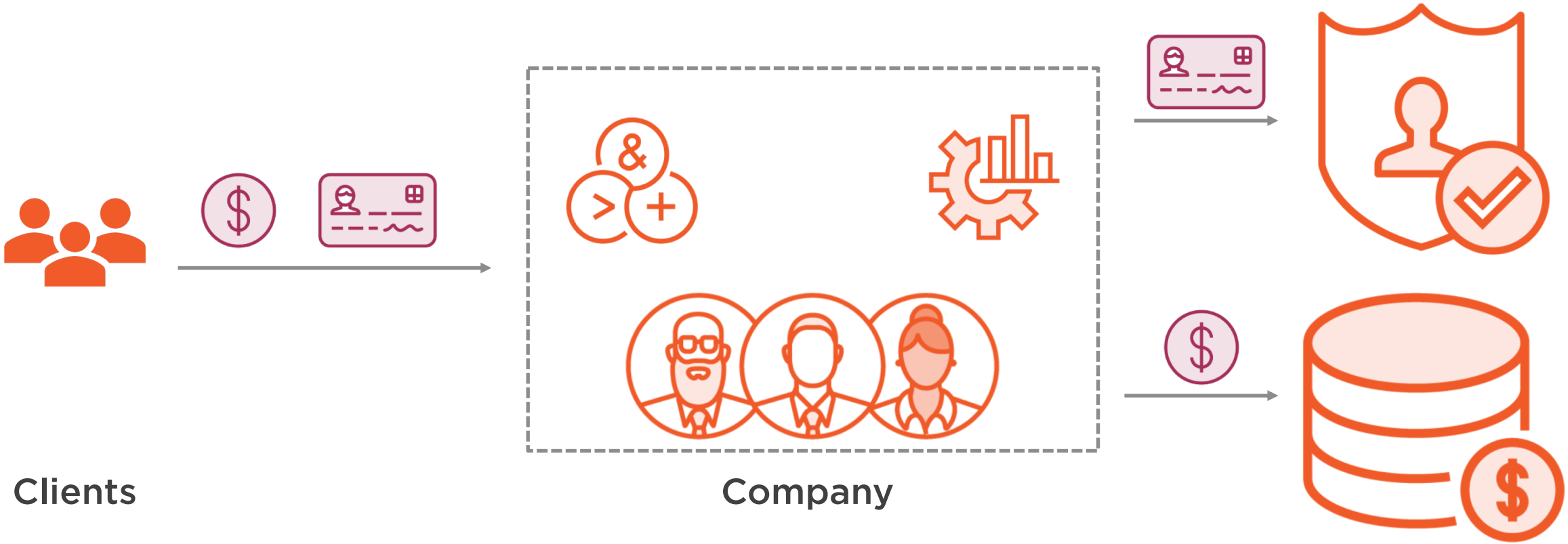
Demo company overview

Demo application overview

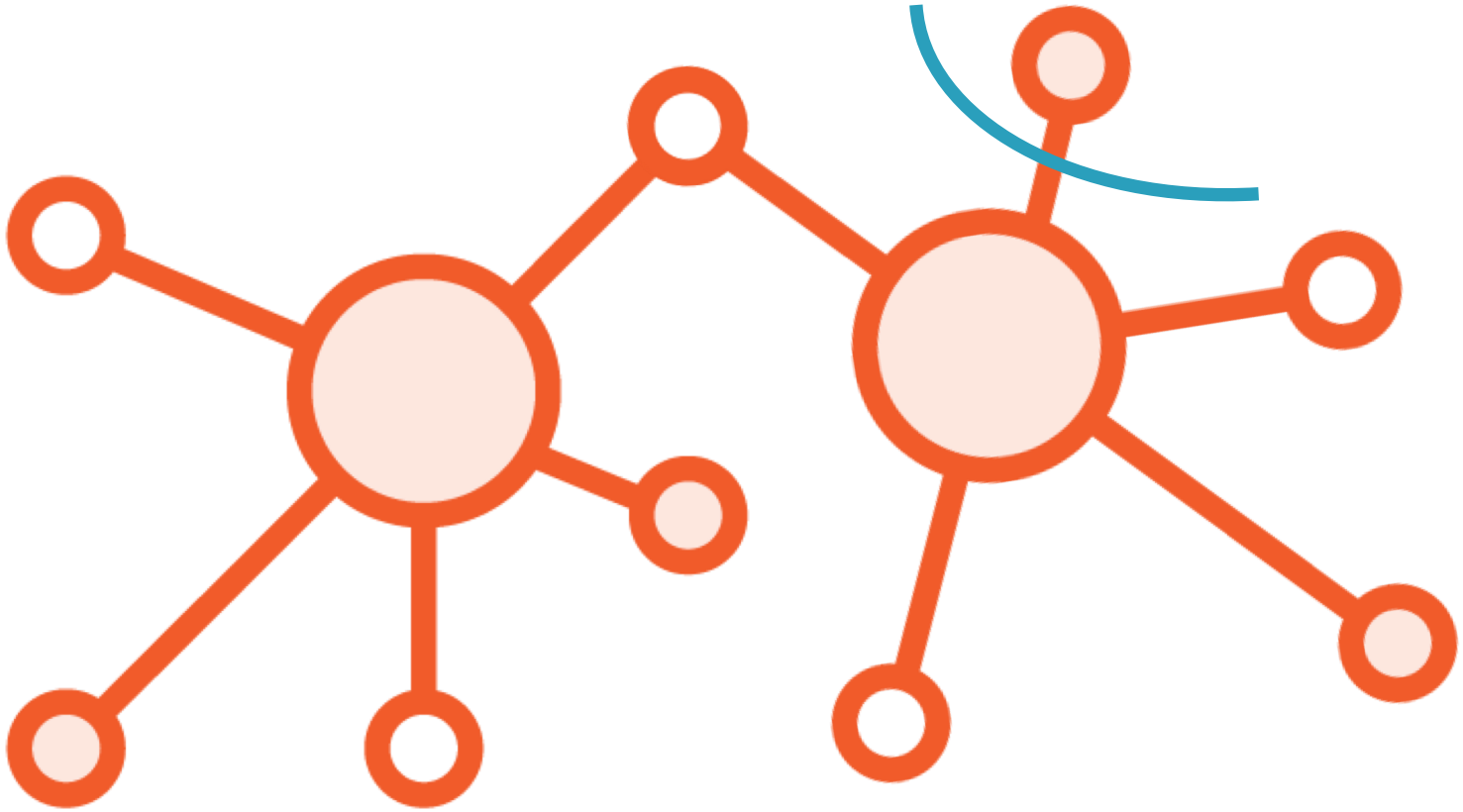
The first test:

- without mocks
- with mocks

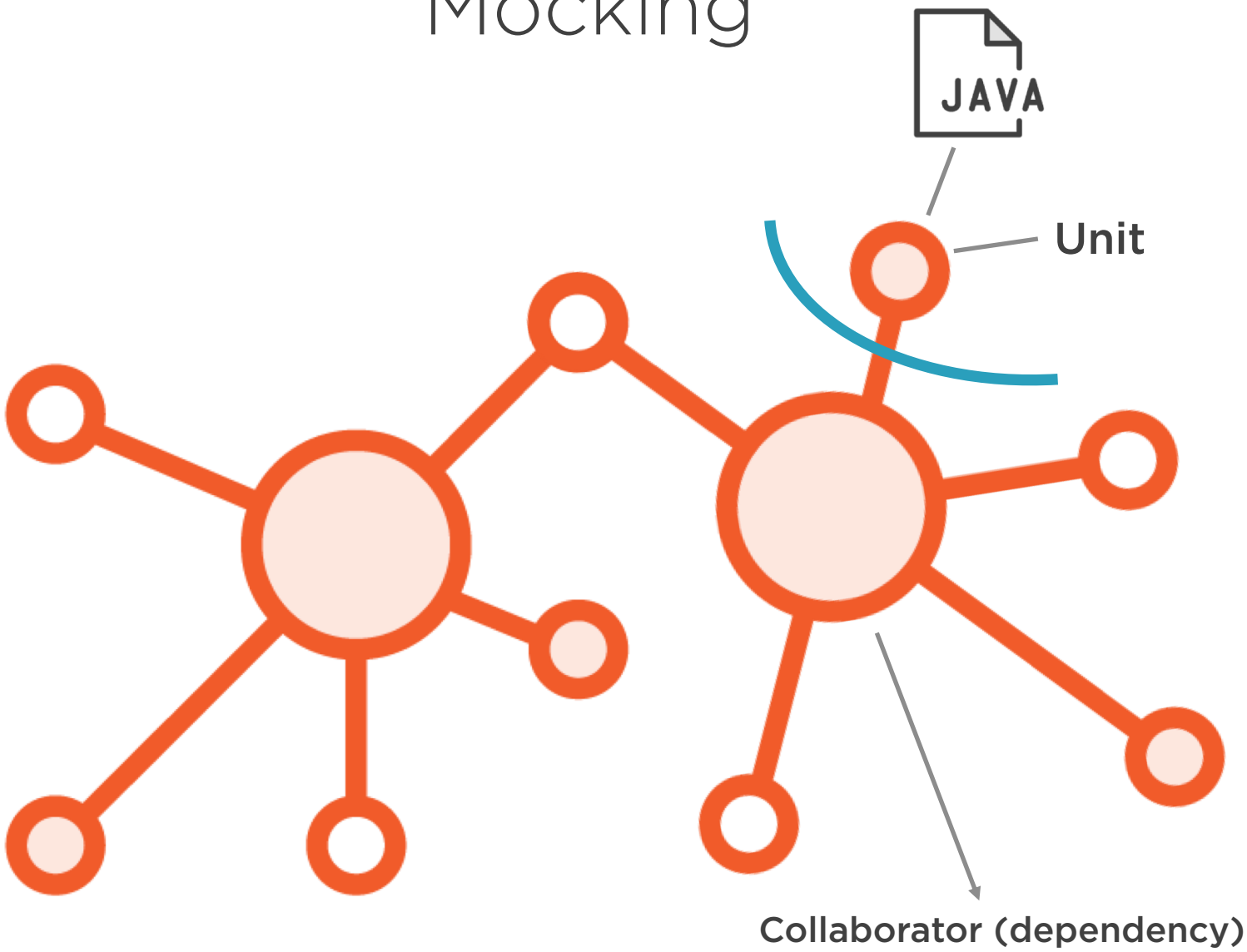
Demo company overview



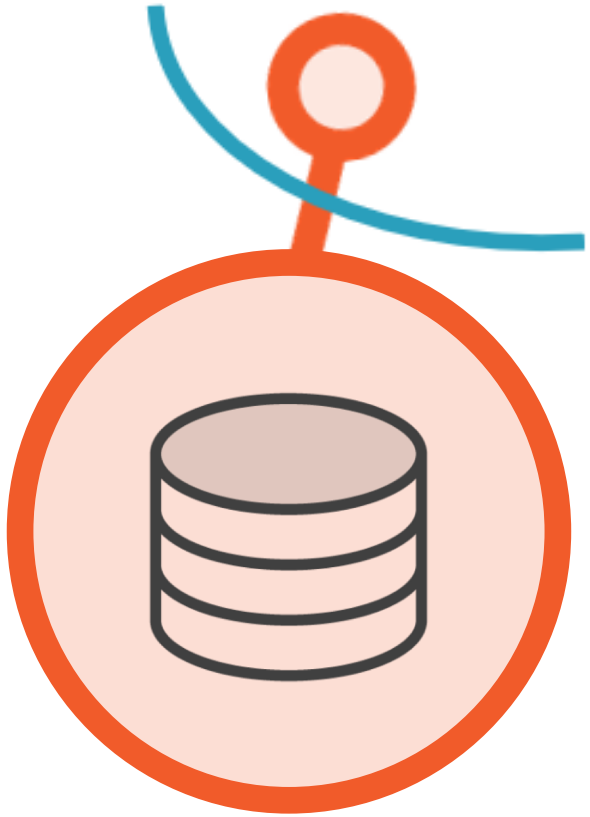
Mocking



Mocking

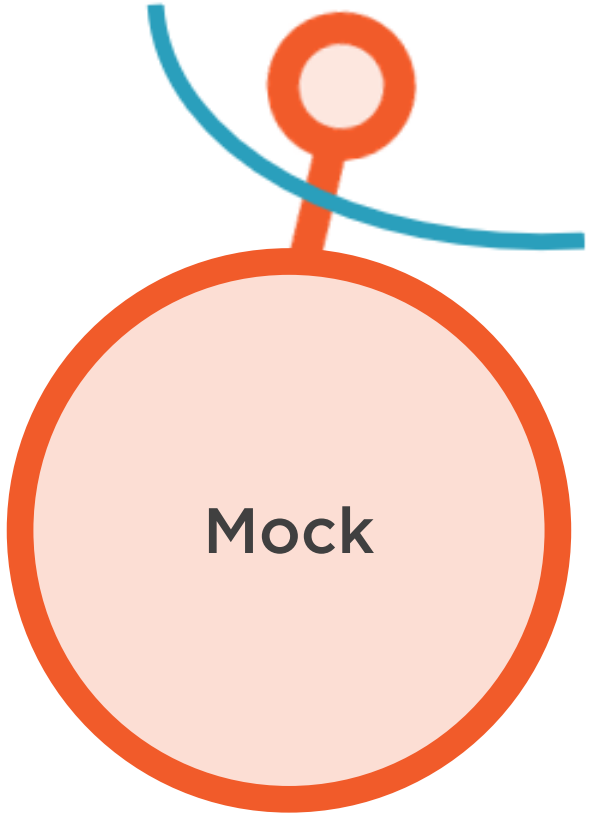


Most Popular Collaborators for Mocking

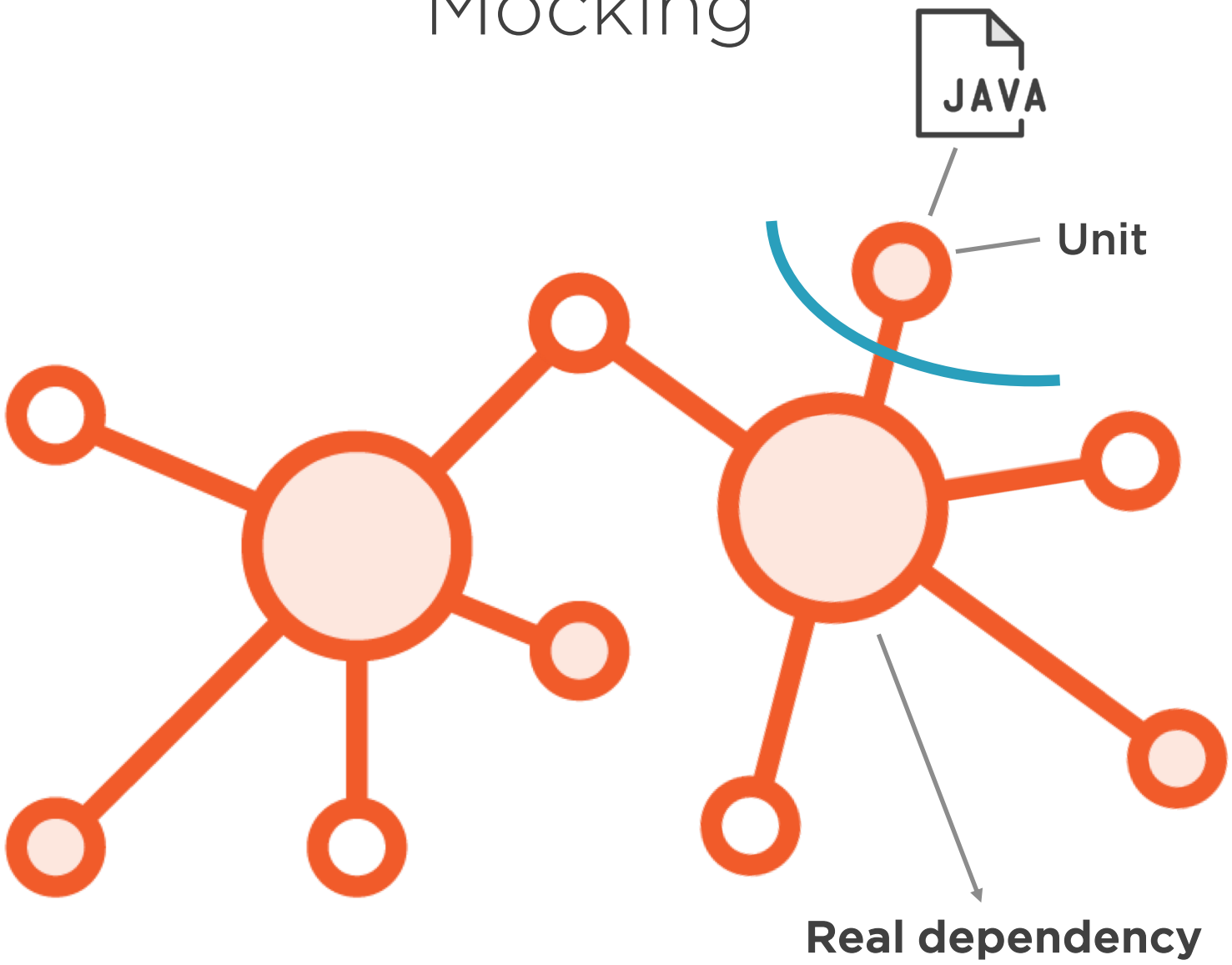


Most Popular Collaborators for Mocking

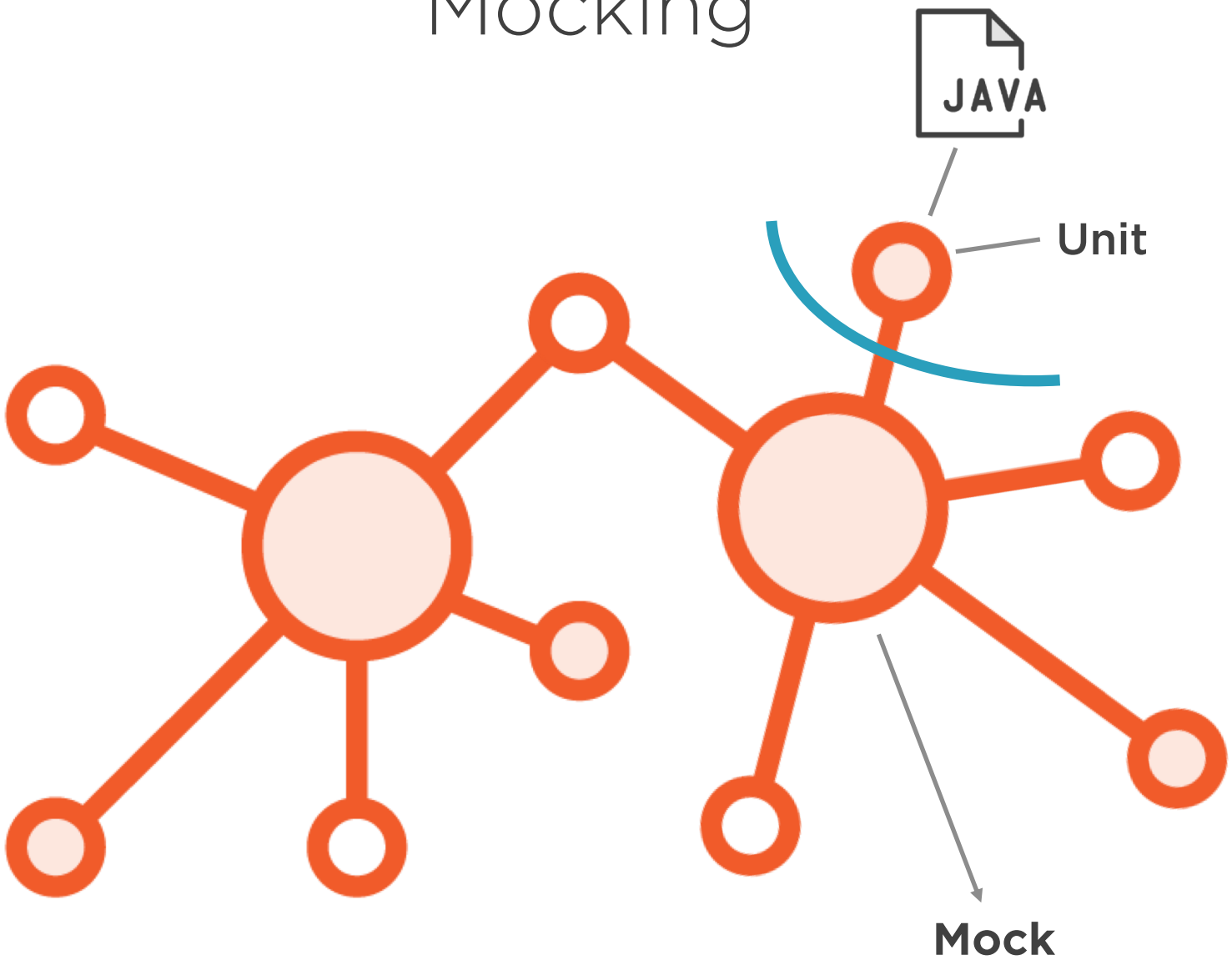
Most Popular Collaborators for Mocking



Mocking



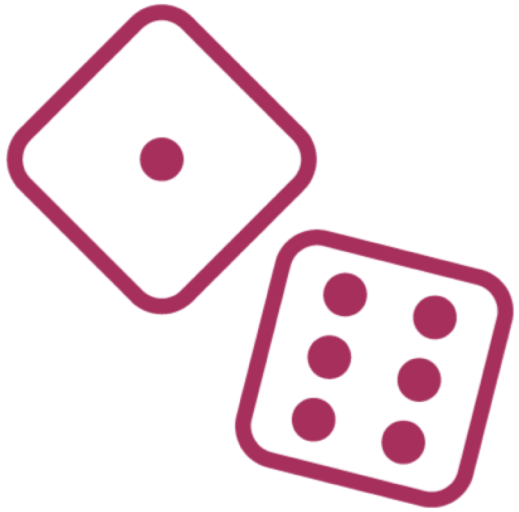
Mocking



Mocking

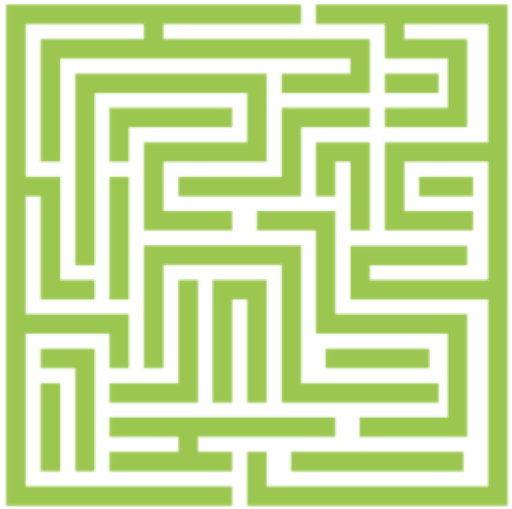
Mocking allows you to focus on the unit you're trying to test by replacing the unit's real dependencies with test-only collaborators. This allows you to reason about the unit in isolation, without having to deal with the rest of the codebase at the same time.

Why Use Mocking?



Eliminates non-determinism

- And randomness



Reduces complexity

- Increases flexibility

Improves test execution:

- Speed
- Reliability

Supports collaboration

“Double is a generic term for any case where you replace a production object for testing purposes.”

Martin Fowler

Test Doubles

Dummy

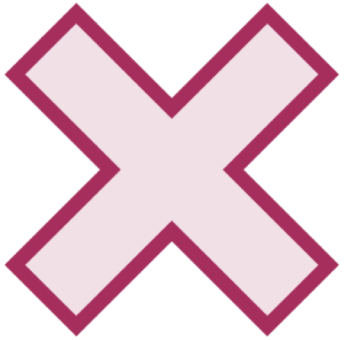
Fake

Stub

Spy

Mock

Test Doubles: Dummy



Not used in the test

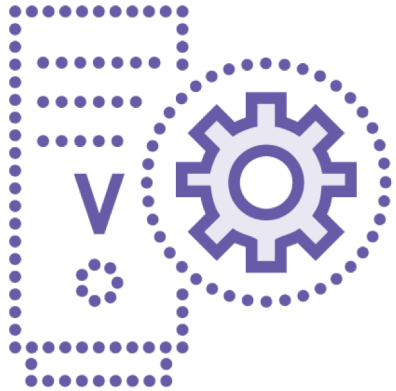
Fills-in holes

Just keeps the compiler happy

Test Doubles: Fake

Implementation that is unsuitable for production

In-memory DB



Fake web service

Test Doubles: Stub

Provides ‘canned’ answers

Not intelligent enough to respond with anything else

Multiple variations are possible

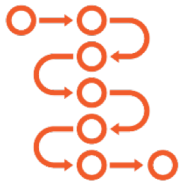
Test Doubles: Spy

Like a more intelligent Stub

Keeps track of how it was used

Also helps with verification

Test Doubles: Mock



Uses expectations

Can fail the test if unexpected calls are made

The focus is on behavior verification

State Verification Versus Behavior Verification

State Verification

All other Test Doubles do it

Verify resulting state

Exception: Spies can verify behavior

Behavior Verification

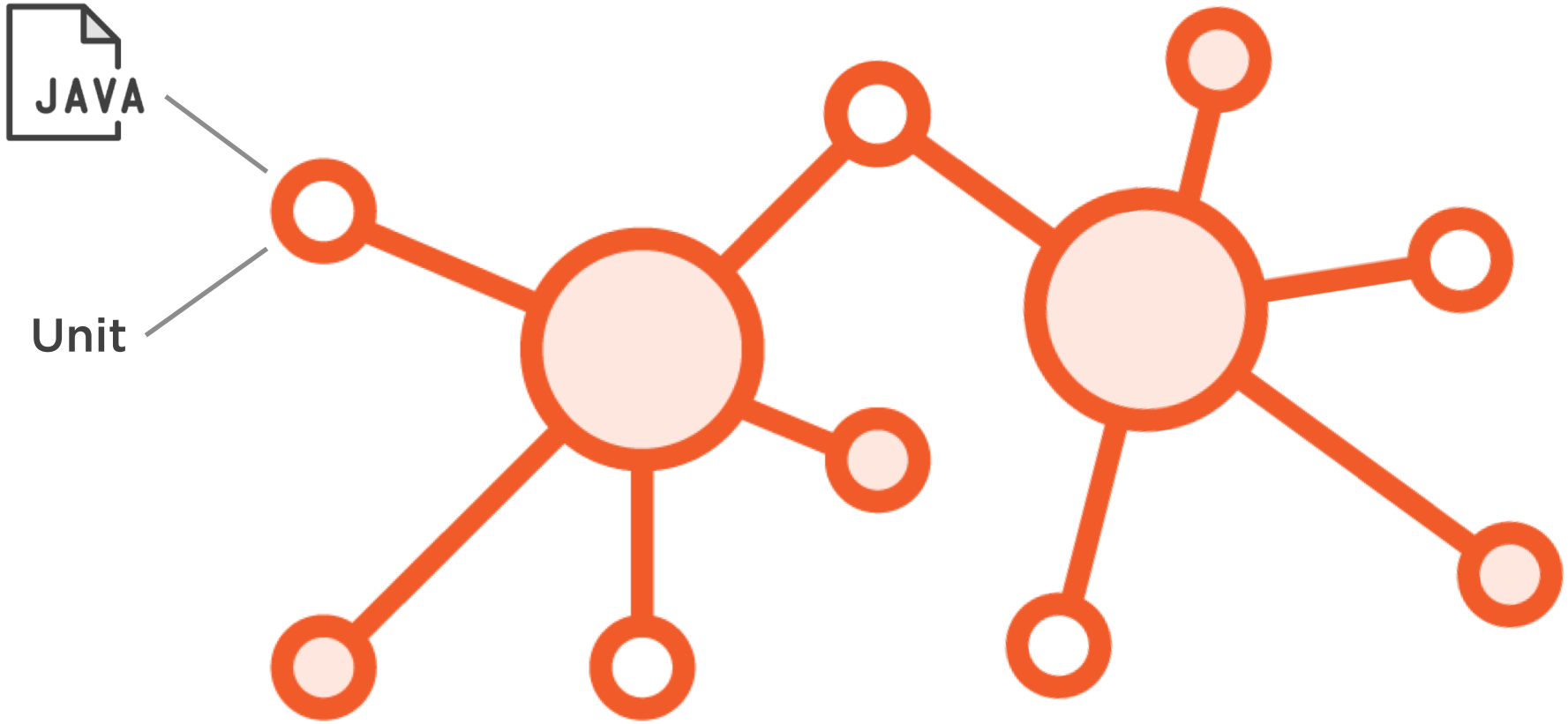
Mocks always use it

Verify interactions

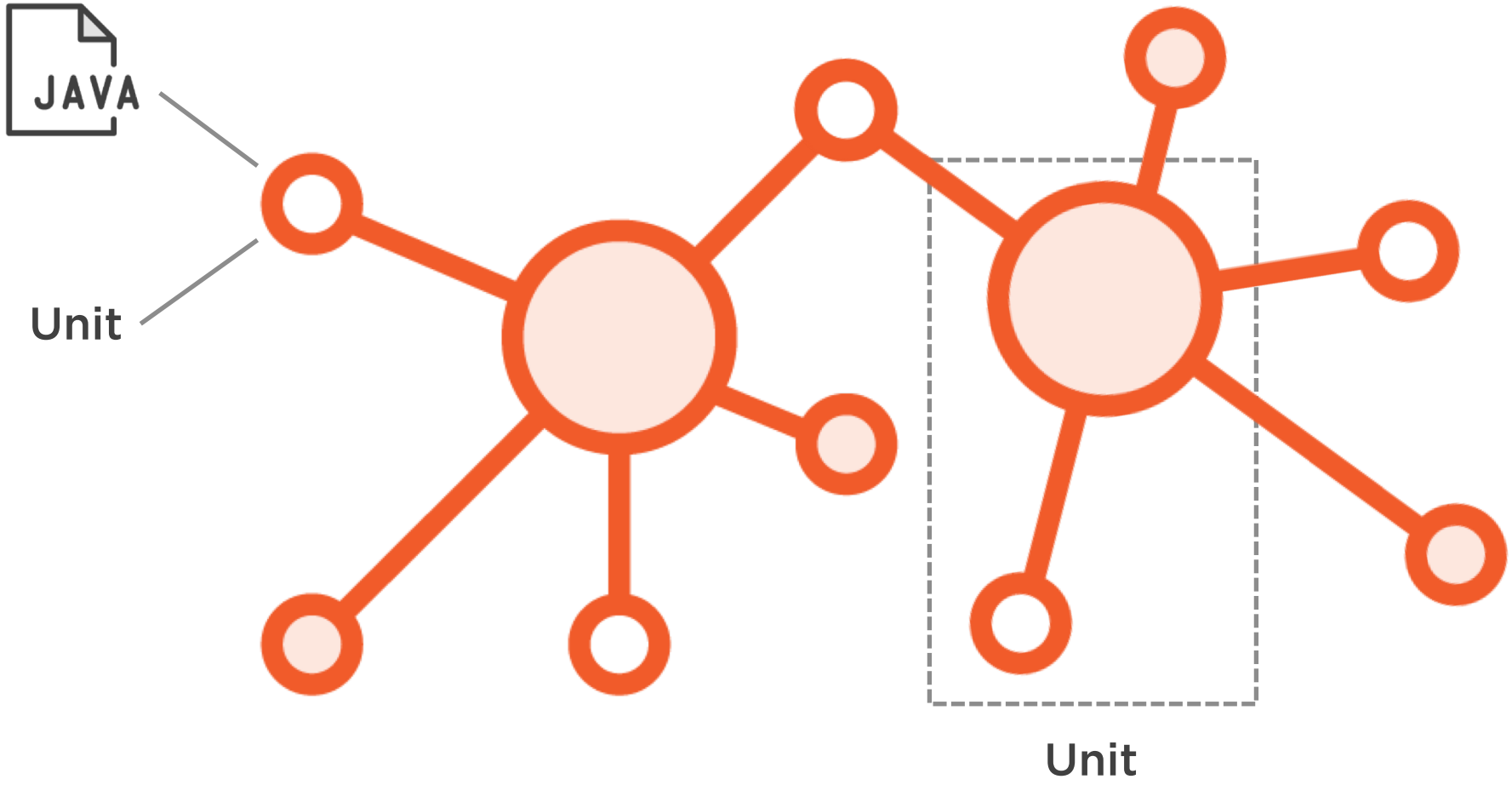
Can add state verification on top

For ease of learning, we'll focus on Spies and Mocks

A Few More Words on Unit Testing



A Few More Words on Unit Testing



“But really it's a situational thing - the team decides what makes sense to be a unit for the purposes of their understanding of the system and its testing. ”

Martin Fowler

A Unit Can Consist of More Than One Class

```
private AccountOpeningService underTest;
private BackgroundCheckService backgroundCheckService =
                                                mock(BackgroundCheckService.class);
private ReferenceIdsManager referenceIdsManager =
                                                mock(ReferenceIdsManager.class);
private AccountRepository accountRepository =
                                                mock(AccountRepository.class);


@BeforeEach
void setUp() {
    underTest = new AccountOpeningService(
                        backgroundCheckService,
                        referenceIdsManager,
                        accountRepository);
}
```

A Unit Can Consist of More Than One Class

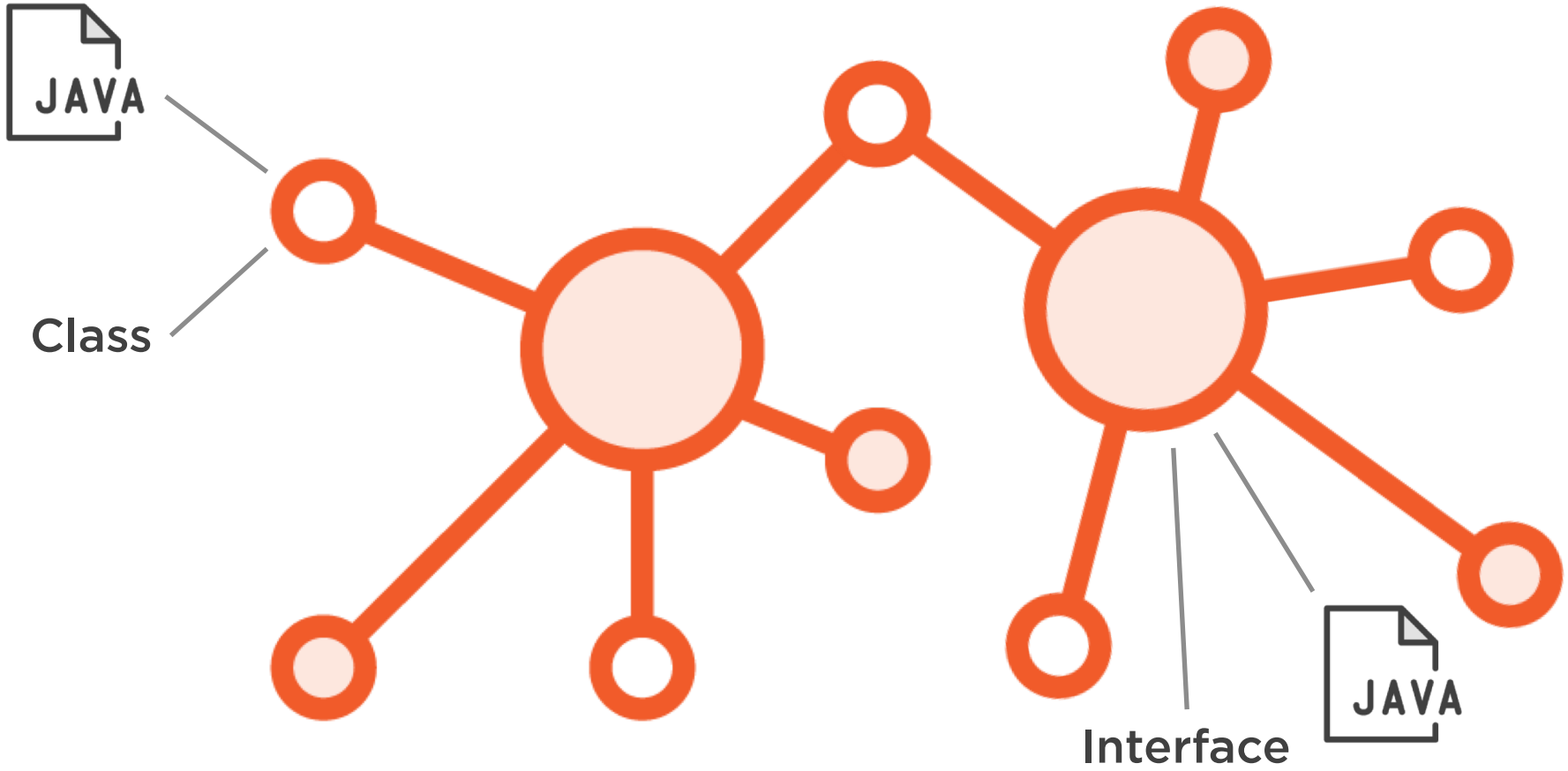
```
private AccountOpeningService underTest;
private BackgroundCheckService backgroundCheckService =
                                                mock(BackgroundCheckService.class);
private ReferenceIdsManager referenceIdsManager =
                                                new ExternalNationalReferenceIdsManager(…);
private AccountRepository accountRepository =
                                                mock(AccountRepository.class);

@BeforeEach
void setUp() {
    underTest = new AccountOpeningService(
                        backgroundCheckService,
                        referenceIdsManager,
                        accountRepository);
}
```

Unit Formation: Paradigm Comparison

Object-Oriented	Functional
A class	A function
A collection of classes	A collection of functions

Do You Mock Concrete Classes or Interfaces?



Do You Mock Concrete Classes or Interfaces?

Mockito

Can mock a class

Can mock an interface

Other frameworks

Other frameworks may only allow mocking at interface level:

- by design
- due to technical reasons

Do You Mock Concrete Classes or Interfaces?

Generally speaking, prefer interfaces over concrete classes in order to follow SOLID principles.

Mockito Facts

Current version: 2

Open source

API design goals: clean, simple

Readable tests

Stats according to https://site.mockito.org/:

- Top 10 Java libraries
- SO voted Mockito best Java mocking lib

Summary

- Described unit testing
- Presented the application
- Wrote a test without mocks
- Wrote tests with mocks using Mockito
- Discussed mocking in detail
- Described each Test Double