Write cool scalable enterprise application tests with Xtend & embedded DSLs

Boris Brodski

EclipseCon Europe 2014

Outline

Outline

Common problems of conventional tests

- Writing tests is boring

Common problems of conventional tests

- Writing tests is boring
  - You need to rethink the entire use case
  - Testing afterwards is trickier than writing implemention

Common problems of conventional tests

- Writing tests is boring
  - You need to rethink the entire use case
  - Testing afterwards is trickier than writing implemention
- Watching slow tests running is demotivating

Common problems of conventional tests

- Writing tests is boring
  - You need to rethink the entire use case
  - Testing afterwards is trickier than writing implemention
- Watching slow tests running is demotivating
  - Causes tests to be ran infrequently
  - Result: lots of broken tests

Common problems of conventional tests

- Writing tests is boring
  - You need to rethink the entire use case
  - Testing afterwards is trickier than writing implemention
  - Low value of the tests
- Watching slow tests running is demotivating
  - Causes tests to be ran infrequently
  - Result: lots of broken tests

Common problems of conventional tests

- Writing tests is boring
  - You need to rethink the entire use case
  - Testing afterwards is trickier than writing implemention
  - Low value of the tests
- Watching slow tests running is demotivating
  - Causes tests to be ran infrequently
  - Result: lots of broken tests
- Fixing broken tests is annoying

Common problems of conventional tests

- Writing tests is boring
  - You need to rethink the entire use case
  - Testing afterwards is trickier than writing implemention
  - Low value of the tests
- Watching slow tests running is demotivating
  - Causes tests to be ran infrequently
  - Result: lots of broken tests
- Fixing broken tests is annoying
  - Good knowledge of the entire system is required

Goal: Make writing tests fun

- Problem: Tests are too complex
  - Test first or better TDD

Goal: Make writing tests fun

- Problem: Tests are too complex
  - Test first or better TDD
  - Modern languages reduce boilerplate code (Xtend, Jnario)
  - Use advance testing techniques: factories, mocking, ...

Goal: Make writing tests fun

- Problem: Tests are too complex
  - Test first or better TDD
  - Modern languages reduce boilerplate code (Xtend, Jnario)
  - Use advance testing techniques: factories, mocking, ...
- Problem: Tests are too slow
  - − integration tests
  - \+ unit tests

Goal: Make writing tests fun

- Problem: Tests are too complex
  - Test first or better TDD
  - Modern languages reduce boilerplate code (Xtend, Jnario)
  - Use advance testing techniques: factories, mocking, ...
- Problem: Tests are too slow
  - – integration tests
  - \+ unit tests
  - Outsource test execution (also for uncommited code)

Goal: Make writing tests fun

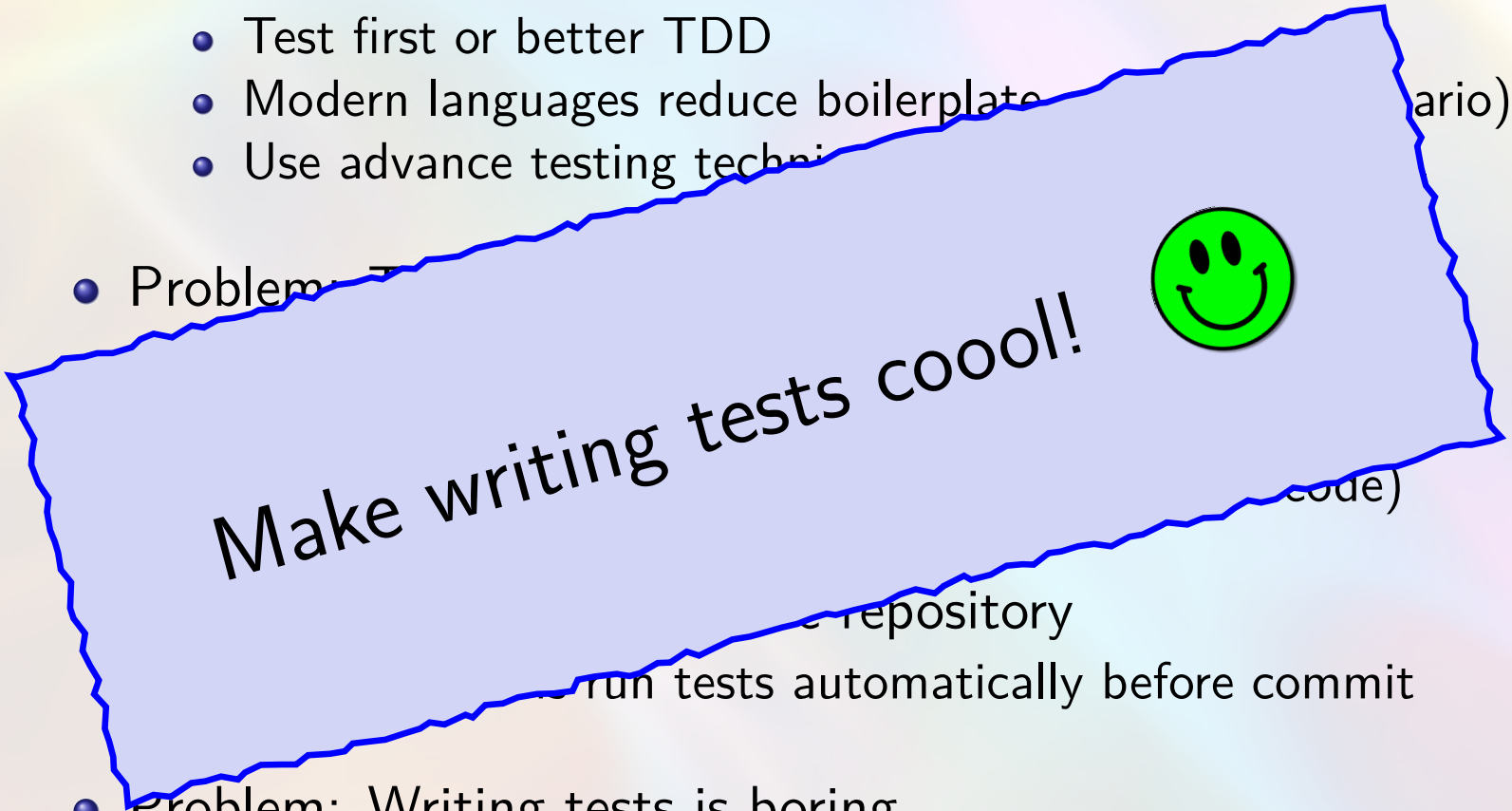
- Problem: Tests are too complex
  - Test first or better TDD
  - Modern languages reduce boilerplate code (Xtend, Jnario)
  - Use advance testing techniques: factories, mocking, ...
- Problem: Tests are too slow
  - – integration tests
  - \+ unit tests
  - Outsource test execution (also for uncommited code)
- Problem: Broken tests in the repository
  - Gerrit+Jenkins run tests automatically before commit

Goal: Make writing tests fun

- Problem: Tests are too complex
  - Test first or better TDD
  - Modern languages reduce boilerplate code (Xtend, Jnario)
  - Use advance testing techniques: factories, mocking, ...
- Problem: Tests are too slow
  - – integration tests
  - \+ unit tests
  - Outsource test execution (also for uncommited code)
- Problem: Broken tests in the repository
  - Gerrit+Jenkins run tests automatically before commit
- Problem: Writing tests is boring
  - Add gamification (e.g. code coverage challenges)

Goal: Make writing tests fun

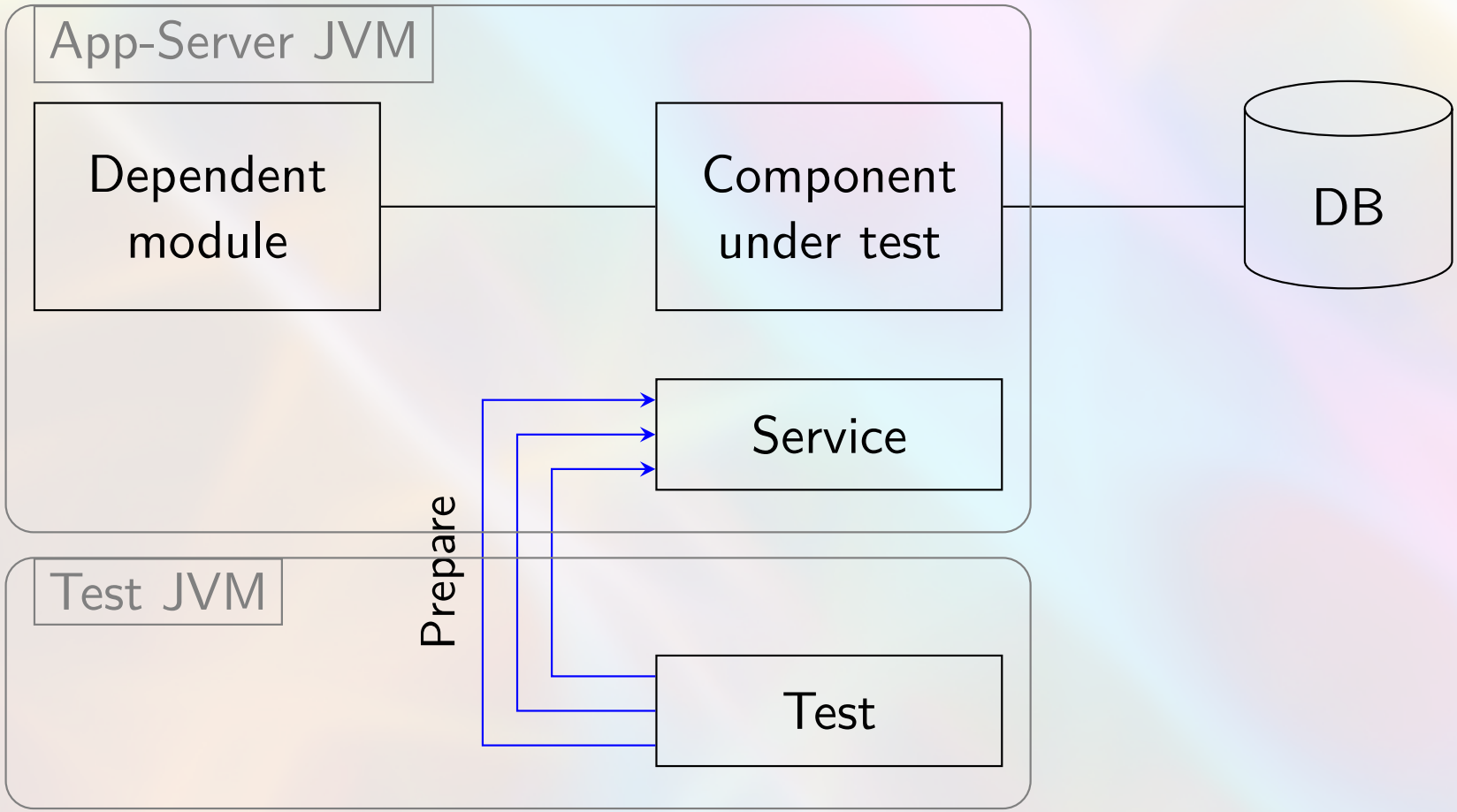
- Problem: Tests are too complex
  - Test first or better TDD
  - Modern languages reduce boilerplate
  - Use advance testing techni
- Problem: T
- Problem: Writing tests is boring
  - Add gamification (e.g. code coverage challenges)



Outline

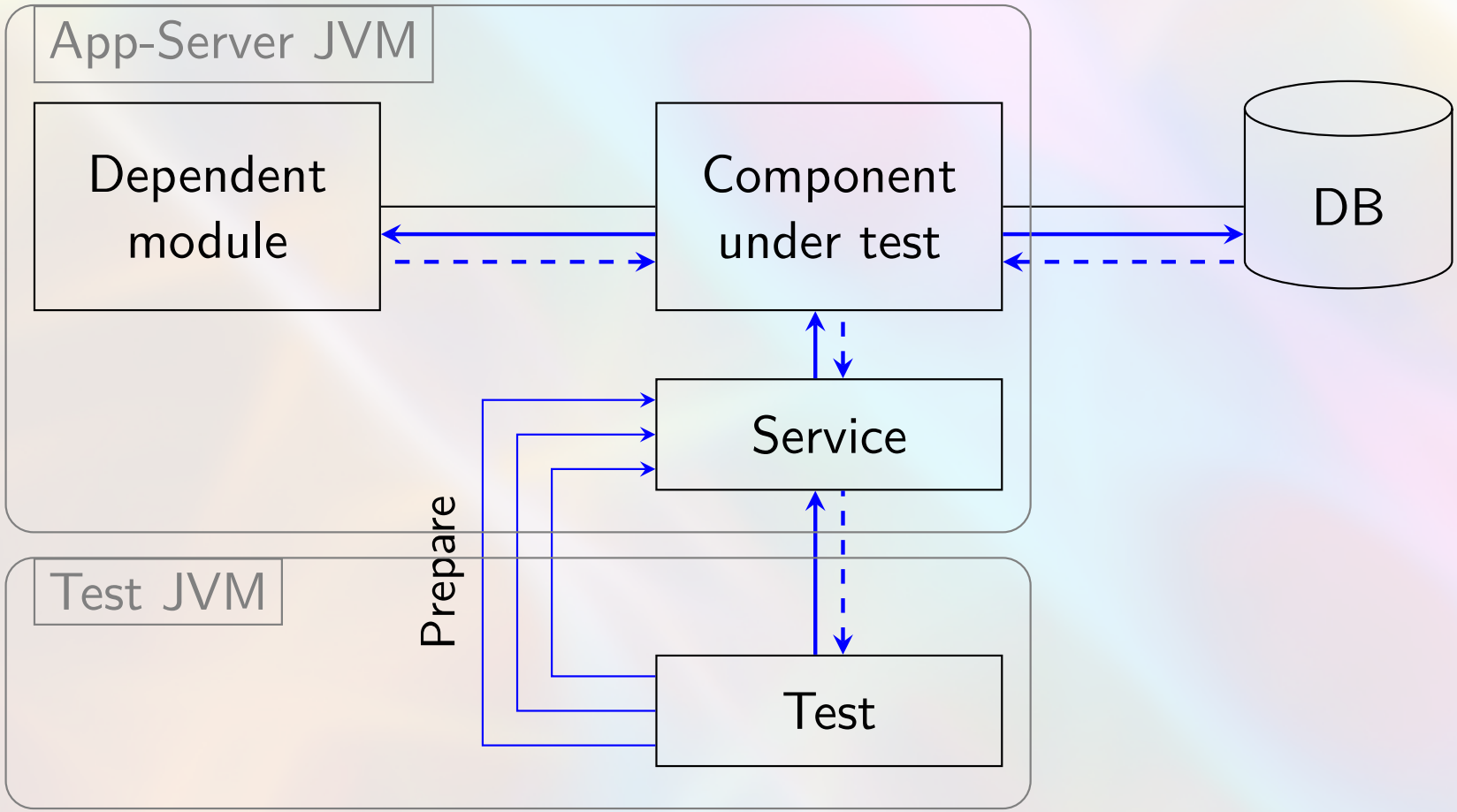
Integration test

- Test entire system



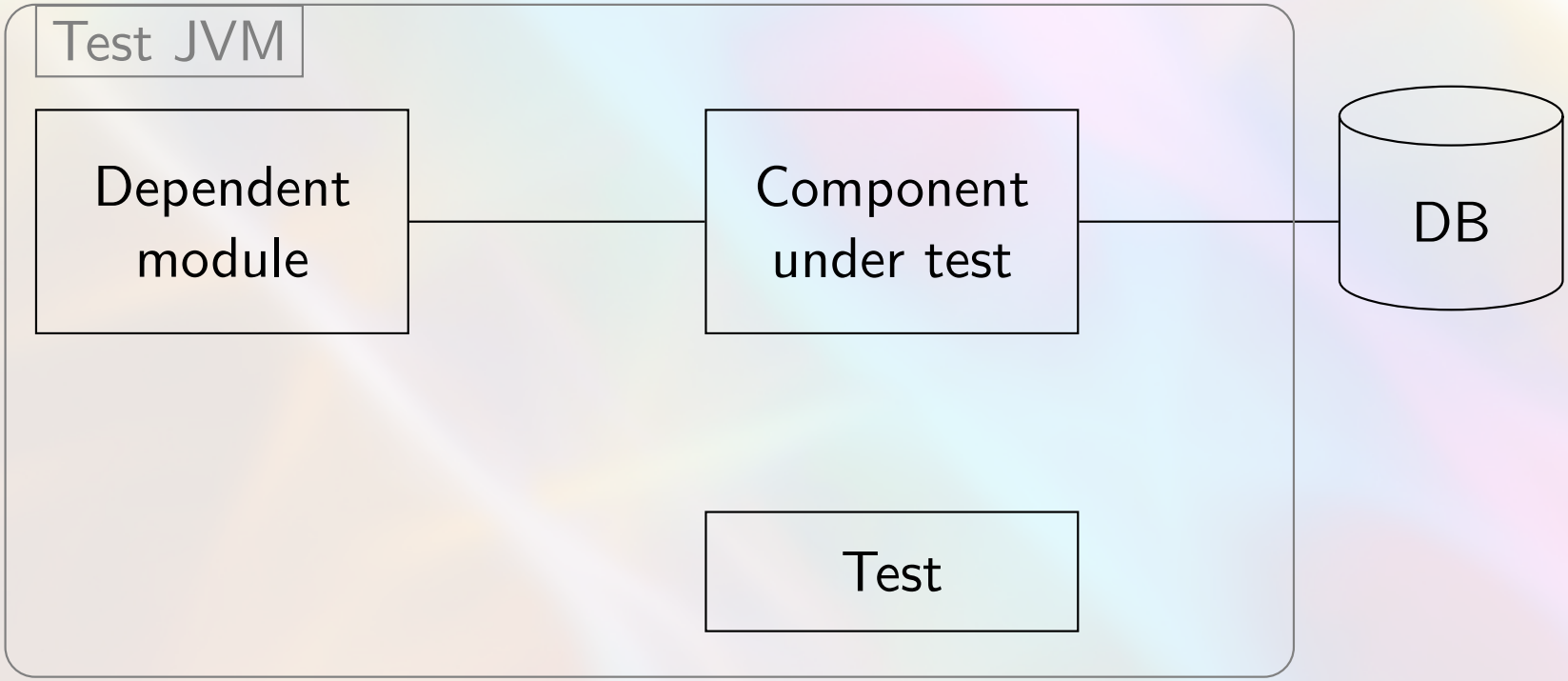
Integration test

- Test entire system



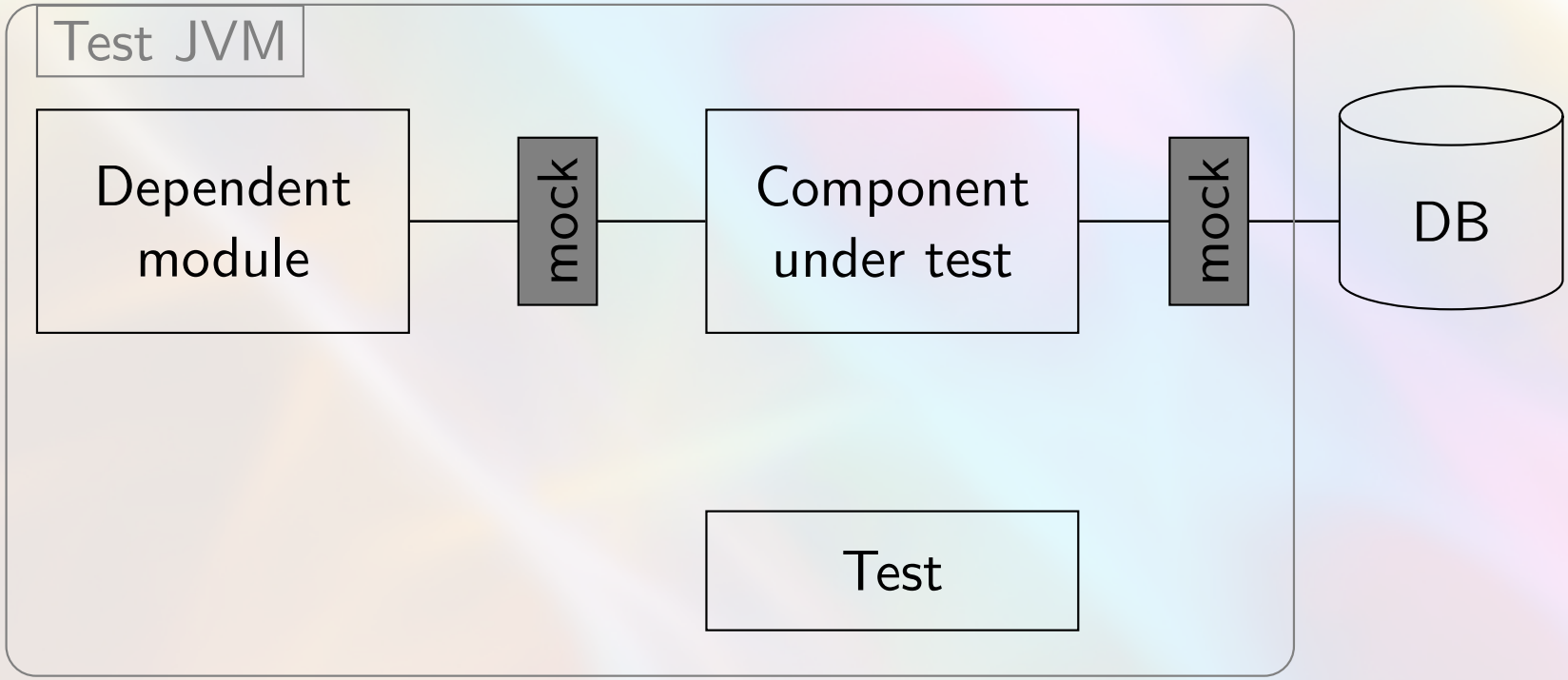
Unit test

- Test in isolation



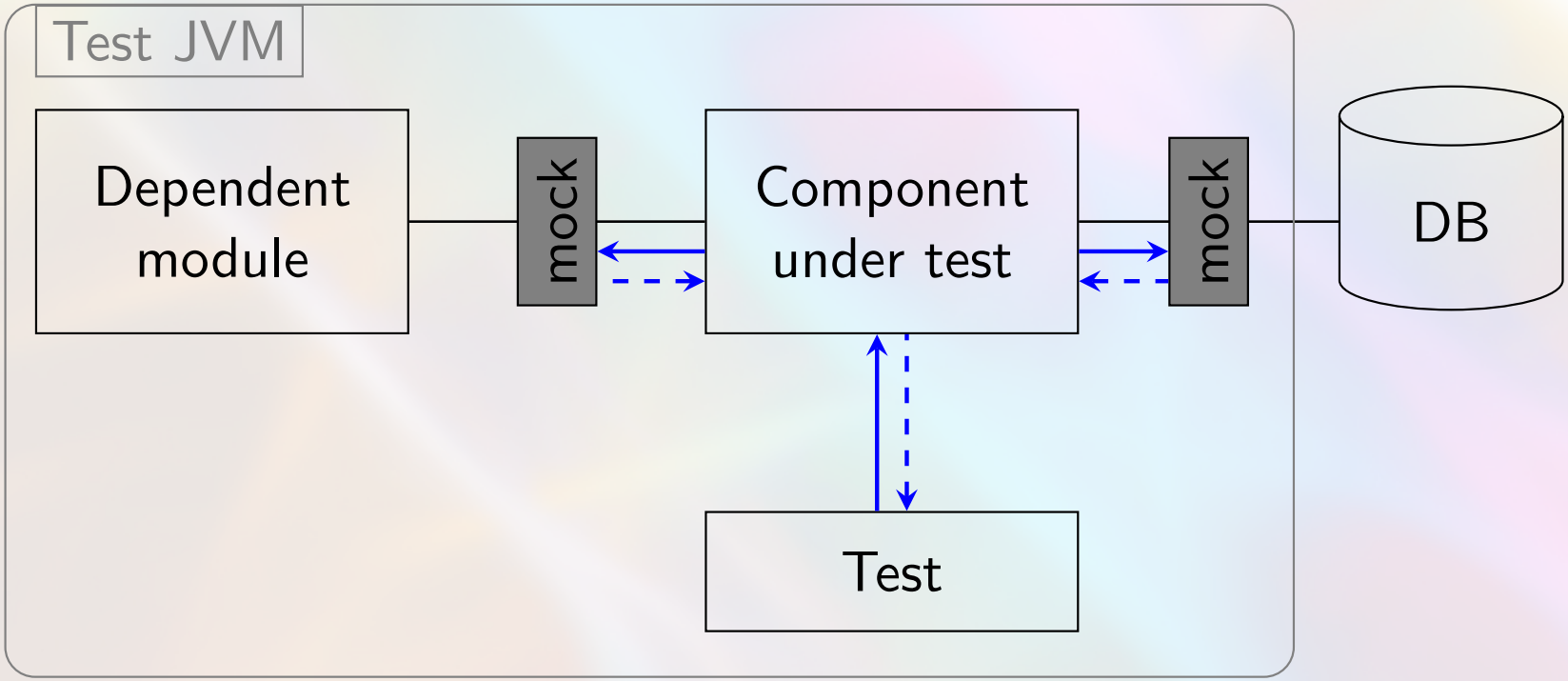
Unit test

- Test in isolation



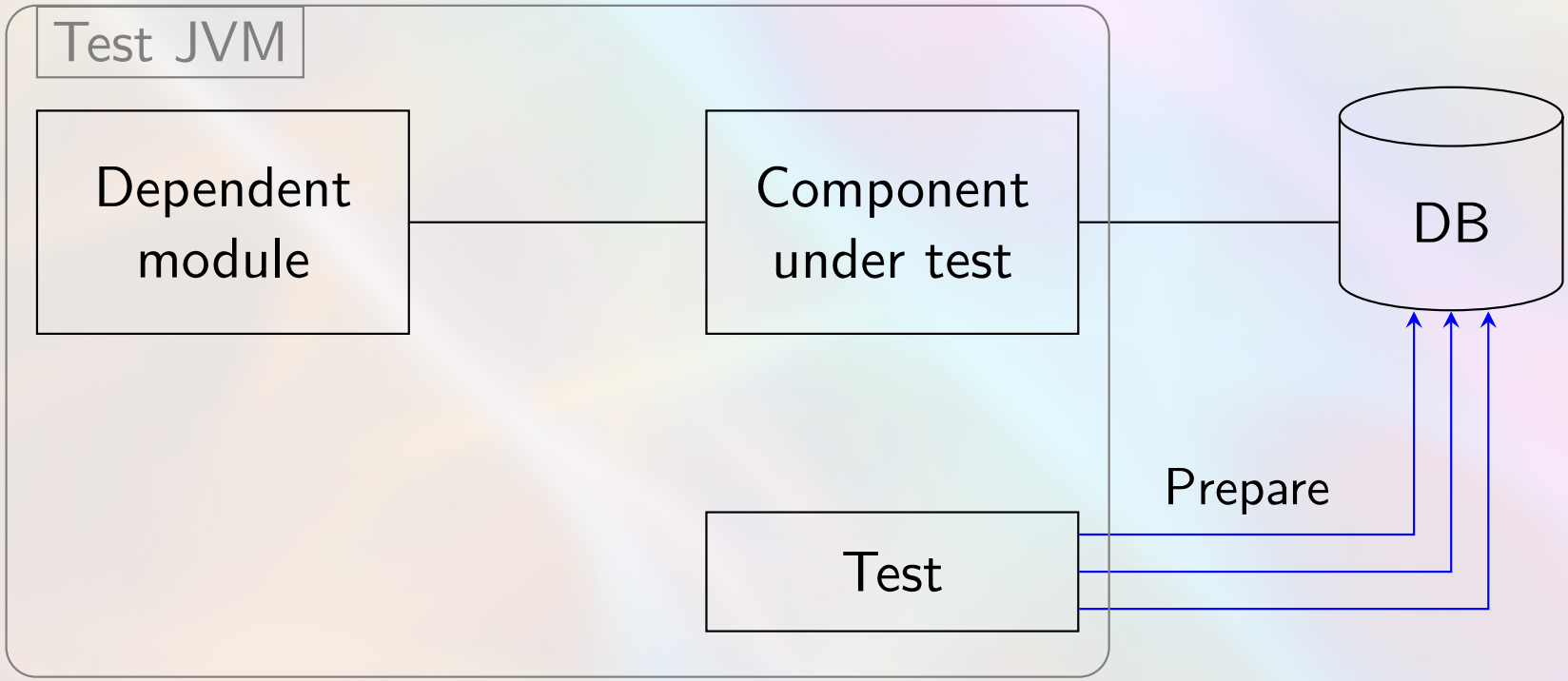
Unit test

- Test in isolation



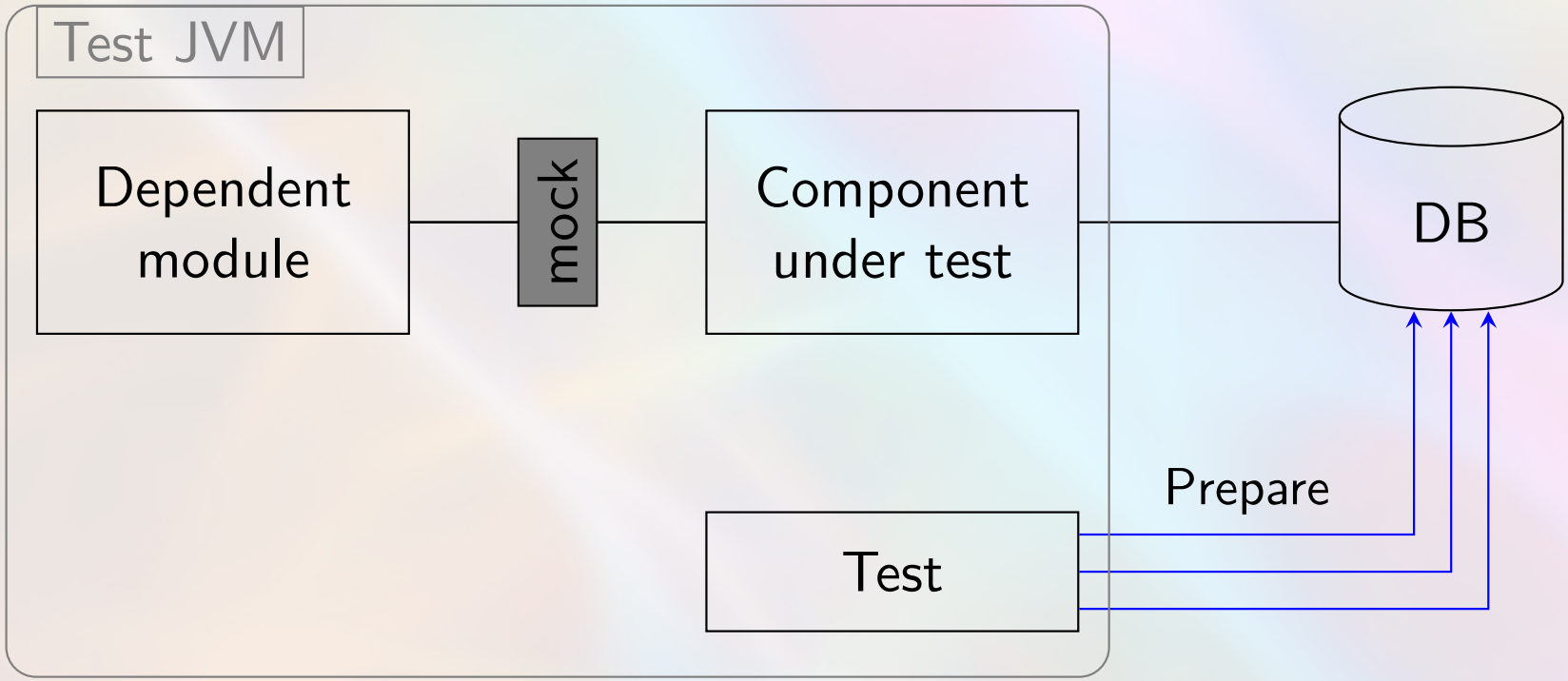
Persistence test

- Test with live database
- Hybrid between unit and integration tests



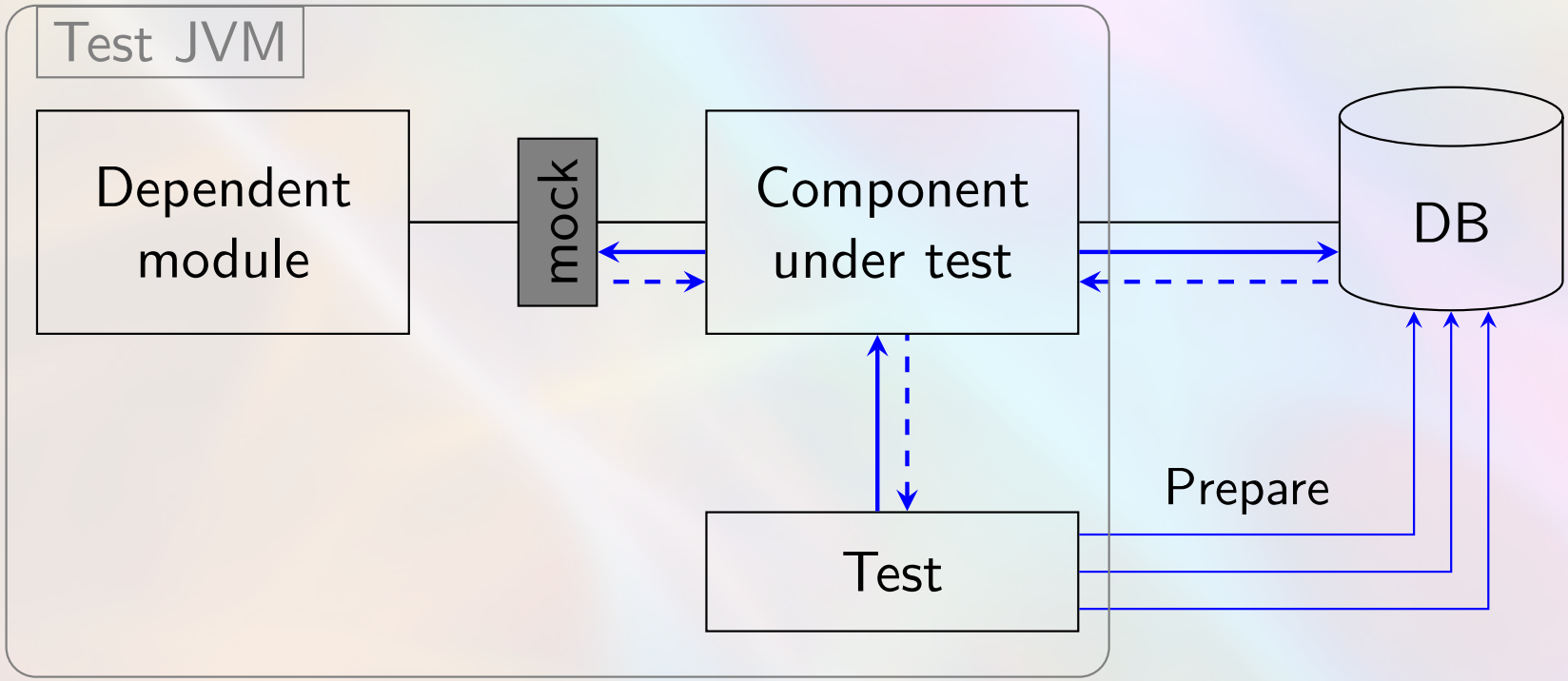
Persistence test

- Test with live database
- Hybrid between unit and integration tests



Persistence test

- Test with live database
- Hybrid between unit and integration tests



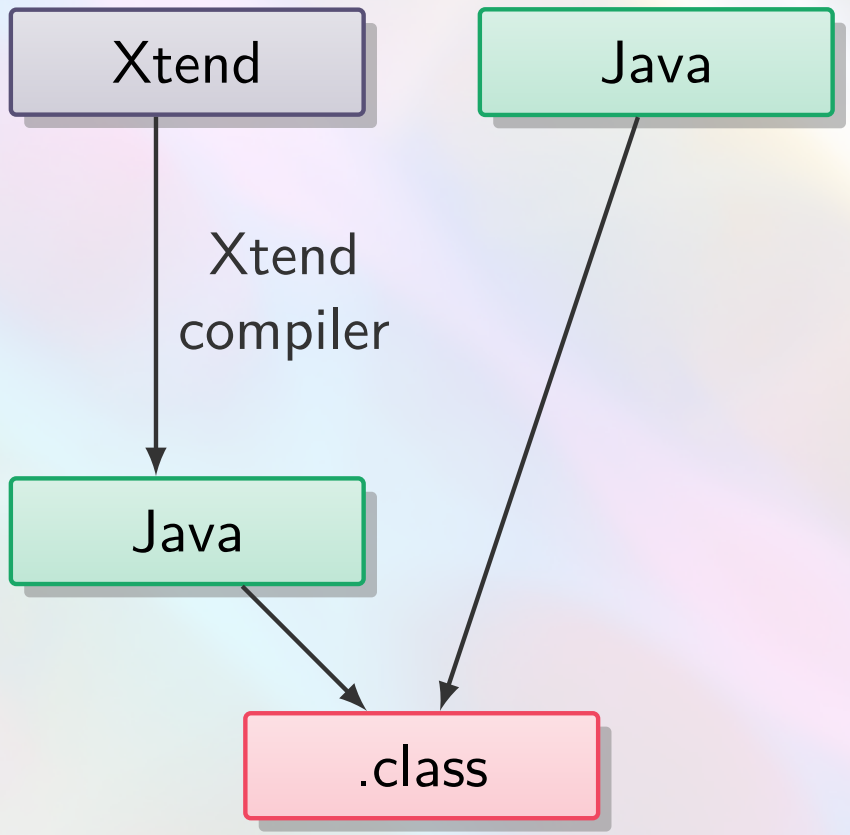
Outline

Java 10 today

Xtend

- Extension to Java
- Adds modern language features
- Compiles into readable Java
- 100% interoperable with Java
- With advanced IDE Support
- Java 8

- http://xtend-lang.org
- http://xtextcasts.org



Xtend Feature overview

- Type inference

```
for (item : list) {...}
```

- Extension methods

```
"age".toFirstUpper
```

- Lambda Expressions

```
btn.addActionListener[ println("Click!") ]
```

- ActiveAnnotations

```
@Accessors String name
```

- Operator overloading

```
"abc" > "bcd"
```

- Powerful switch expressions

```
switch o {Set case o.size > 3: o.size}
```

- No statements

```
msg = if (answer) "yes" else "no"
```

- Template expressions

```
'''
    name:  «name»
    score: «score * 10»
'''
```

- Multiple dispatch

```
def dispatch op(Long s)  {...}
def dispatch op(Short l) {...}
def dispatch op(Float f) {...}
```

Simple JUnit test with Xtend

Java

```
public class MyTest {

    @Test
    public void test1() {

        // test code

    }
}
```

Simple JUnit test with Xtend

Java

```
public class MyTest {

    @Test
    public void test1() {

        // test code

    }
}
```

Xtend

```
class MyTest {

    @Test
    def void test1() {

        // test code

    }
}
```

Assertions

Java

```
import static org.junit.Assert.*

assertEquals(f1(4), f2(1, 2, f3(3)))



assertTrue(f1(1, 2, f2(3)))
```

Assertions

Java

```
import static org.junit.Assert.*

assertEquals(f1(4), f2(1, 2, f3(3)))


assertTrue(f1(1, 2, f2(3)))
```

Xtend

```
import static extension org.junit.Assert.*

f1(4).assertEquals(f2(1, 2, f3(3)))
```

Assertions

Java

```
import static org.junit.Assert.*

assertEquals(f1(4), f2(1, 2, f3(3)))


assertTrue(f1(1, 2, f2(3)))
```

Xtend

```
import static extension org.junit.Assert.*

f1(4).assertEquals(f2(1, 2, f3(3)))

f2(1, 2, f3(3)) <=> f1(4)

f1(1, 2, f2(3)) <=> true
```

Implementation

```
public void operator_spaceship(Object o1, Object o2) {
    Assert.assertEquals(o2, o1);
}
```

Assertions

Java

```
import static org.junit.Assert.*

assertEquals(f1(4), f2(1, 2, f3(3)))


assertTrue(f1(1, 2, f2(3)))
```

Xtend

```
import static extension org.junit.Assert.*

f1(4).assertEquals(f2(1, 2, f3(3)))

f2(1, 2, f3(3)) <=> f1(4)

f1(1, 2, f2(3)) <=> true
```

Implementation

```
public void operator_spaceship(Object o1, Object o2) {
    Assert.assertEquals(o2, o1);
}
```

2 years ago

Java

```
Calendar now = Calendar.getInstance();
date.setTime(now.getTime());
date.add(Calendar.YEAR, -2);
dto.setTimestamp(date.getTime());
```

2 years ago

Java

```
Calendar now = Calendar.getInstance();
date.setTime(now.getTime());
date.add(Calendar.YEAR, -2);
dto.setTimestamp(date.getTime());
```

Xtend

```
dto.timestamp = 2.years.ago
```

2 years ago

Java

```
Calendar now = Calendar.getInstance();
date.setTime(now.getTime());
date.add(Calendar.YEAR, -2);
dto.setTimestamp(date.getTime());
```

Xtend

```
dto.timestamp = 2.years.ago
```

Implementation

```
public long years(final int years) {
  Calendar date = Calendar.getInstance();
  Date now = date.getTime();
  date.add(Calendar.YEAR, years);
  return date.getTime().getTime() - now.getTime();
}
public Date ago(final long timeInMillis) {
    return new Date(System.currentTimeMillis() - timeInMillis);
}
```

2 years ago

Java

```
Calendar now = Calendar.getInstance();
date.setTime(now.getTime());
date.add(Calendar.YEAR, -2);
dto.setTimestamp(date.getTime());
```

Xtend

```
dto.timestamp = 2.years.ago
```

Implementation

```
public long years(final int years) {
  Calendar date = Calendar.getInstance();
  Date now = date.getTime();
  date.add(Calendar.YEAR, years);
  return date.getTime().getTime() - now.getTime();
}
public Date ago(final long timeInMillis) {
    return new Date(System.currentTimeMillis() - timeInMillis);
}
```

Filling structures

Java

```
AuthorDTO author = new AuthorDTO();
author.setFirstName("Erle Stanley");
author.setLastName("Gardner");

GenreDTO genre = new GenreDTO();
genre.setName("Detective");

BookDTO b = new BookDTO();
b.setAuthor(authorDTO);
b.setGenre(genreDTO);
b.setTitle("The Case of the Velvet Claws");
b.setIsbn("0884114015");
```

Filling structures

Java

```
AuthorDTO author = new AuthorDTO();
author.setFirstName("Erle Stanley");
author.setLastName("Gardner");

GenreDTO genre = new GenreDTO();
genre.setName("Detective");

BookDTO b = new BookDTO();
b.setAuthor(authorDTO);
b.setGenre(genreDTO);
b.setTitle("The Case of the Velvet Claws");
b.setIsbn("0884114015");
```

Xtend

```
val book = new BookDTO => [
    author = new AuthorDTO => [
        firstName = "Erle Stanley"
        lastName = "Gardner"
    ]
    genre = new GenreDTO => [
        name = "Detective"
    ]
    title = "The Case of the Velvet Claws"
    isbn = "0884114015"
]
```

Filling structures

Java

```
AuthorDTO author = new AuthorDTO();
author.setFirstName("Erle Stanley");
author.setLastName("Gardner");

GenreDTO genre = new GenreDTO();
genre.setName("Detective");

BookDTO b = new BookDTO();
b.setAuthor(authorDTO);
b.setGenre(genreDTO);
b.setTitle("The Case of the Velvet Claws");
b.setIsbn("0884114015");
```

Xtend

```
val book = new BookDTO => [
    author = new AuthorDTO => [
        firstName = "Erle Stanley"
        lastName = "Gardner"
    ]
    genre = new GenreDTO => [
        name = "Detective"
    ]
    title = "The Case of the Velvet Claws"
    isbn = "0884114015"
]
```

Outline

XFactory

- Create and persist entities
- Embedded DSL (into Xtend/Jnario)
- Designed for unit and persistence tests
- Open Source:
  `https://github.com/borisbrodski/xfactory`

Using XFactory

Get instance of a valid entity

```
val book = xbuild(new XFactoryBook)
```

Using XFactory

Get instance of a valid entity

```
val book = xbuild(new XFactoryBook)
```

Persist an entity

```
val book = xpersist(new XFactoryBook)
```

Using XFactory

Get instance of a valid entity

```
val book = xbuild(new XFactoryBook)
```

Persist an entity

```
val book = xpersist(new XFactoryBook)
```

Change default values

```
val b = xpersist(new XFactoryBook) [
    minimal
    set [
        title = "Eclipse IDE - kurz & gut"
    ]
]
```

Using XFactory

Get instance of a valid entity

```
val book = xbuild(new XFactoryBook)
```

Persist an entity

```
val book = xpersist(new XFactoryBook)
```

Change default values

```
val b = xpersist(new XFactoryBook) [
    minimal
    set [
        title = "Eclipse IDE - kurz & gut"
    ]
]
```

Use predefined methods

```
val b = xpersist(new XFactoryBook) [
    minimal(author)

    makeBestSeller

]
```

Implementing XFactory

Building an author

```
class XFactoryAuthor extends AbstractXFactory<Author> {
    extension XtendTestContrib = new XtendTestContrib

    override minimal() {
        set [
            firstName = "John"
            lastName = "Doe"
            birthday = 14.april(1967)

            genre = xpersistBefore(new XFactoryGenre)
        ]
    }
    def kill() {
        set [
            dayOfDeath = birthday + 40.years
        ]
    }
}
```

Implementing XFactory

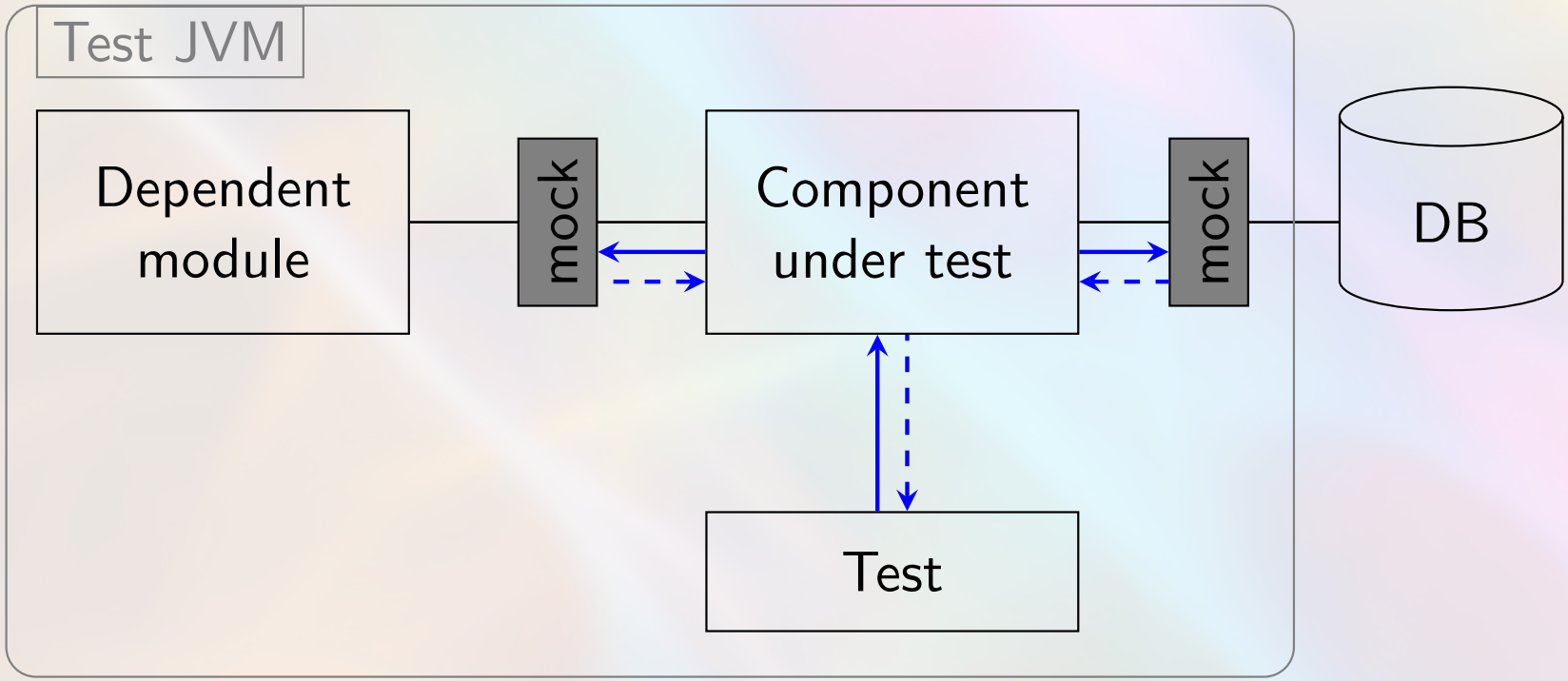
Building an author

```
class XFactoryAuthor extends AbstractXFactory<Author> {
    extension XtendTestContrib = new XtendTestContrib

    override minimal() {
        set [
            firstName = "John"
            lastName = "Doe"
            birthday = 14.april(196

            genre = xpersistBefore(new
        ]
    }
    def kill() {
        set [
            dayOfDeath = birthday + 40.years
        ]
    }
}
```

Outline

Unit test

- Test in isolation



JMockit-Xtend

- Add JMockit support to Xtend/Jnario
- http://jmockit.github.io/
- http://github.com/borisbrodski/jmockit-xtend

Using JMockit-Xtend

```
@Mocked
DependentModule module

@Test
def void test() {
    ...
    stub [
        module.validName("test")
        result = true
    ]
    ...
}
```

JMockit-Xtend

- Add JMockit support to Xtend/Jnario
- http://jmockit.github.io/
- http://github.com/borisbrodski/jmockit-xtend

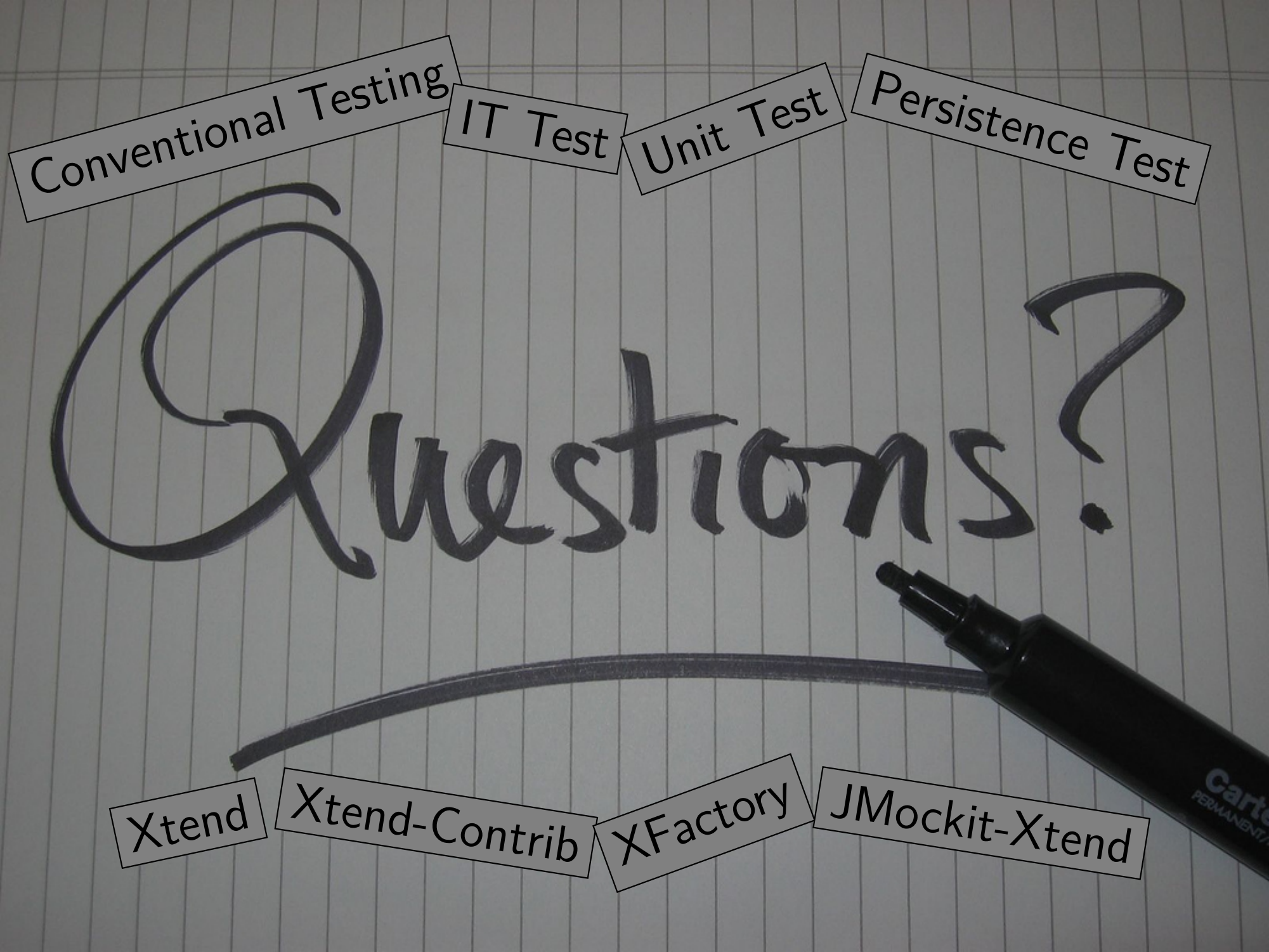
Using JMocki

```
@Mocked
DependentModule

@Test
def void test() {
    ...
    stub [
        module.validName("test")
        result = true
    ]
    ...
}
```

Outline

Live-Demo

http://www.flickr.com/photos/spam/


Conventional Testing
IT Test
Unit Test
Persistence Test
Questions?
Xtend
Xtend-Contrib
XFactory
JMockit-Xtend