Tests Should be FIRST

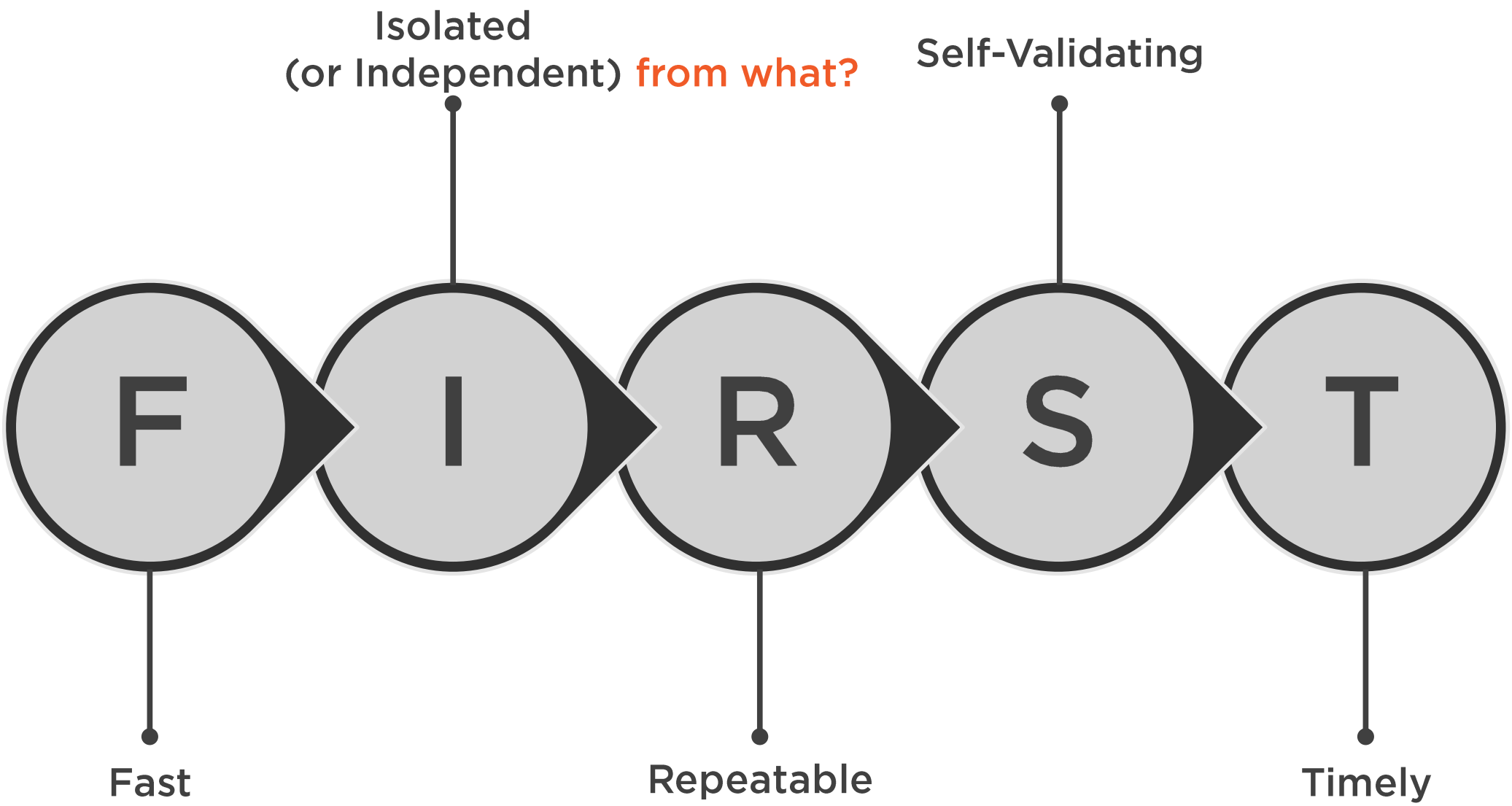
Andrejs Doronins
TEST AUTOMATION ENGINEER

Valuable Tests

Asset

Useless Tests

Costly liability


Isolated
(or Independent) from what?
Self-Validating
F
I
R
S
T
Fast
Repeatable
Timely

FAST

Fast

Self-Validating

STABLE

Isolated

Independent

Repeatable

USEFUL

Timely

Tests should be fast!

Thanks, captain obvious!

Why does it matter to have fast tests?

How to tell if a single test is fast?

How to tell if a test suite is fast?

How to make them faster?

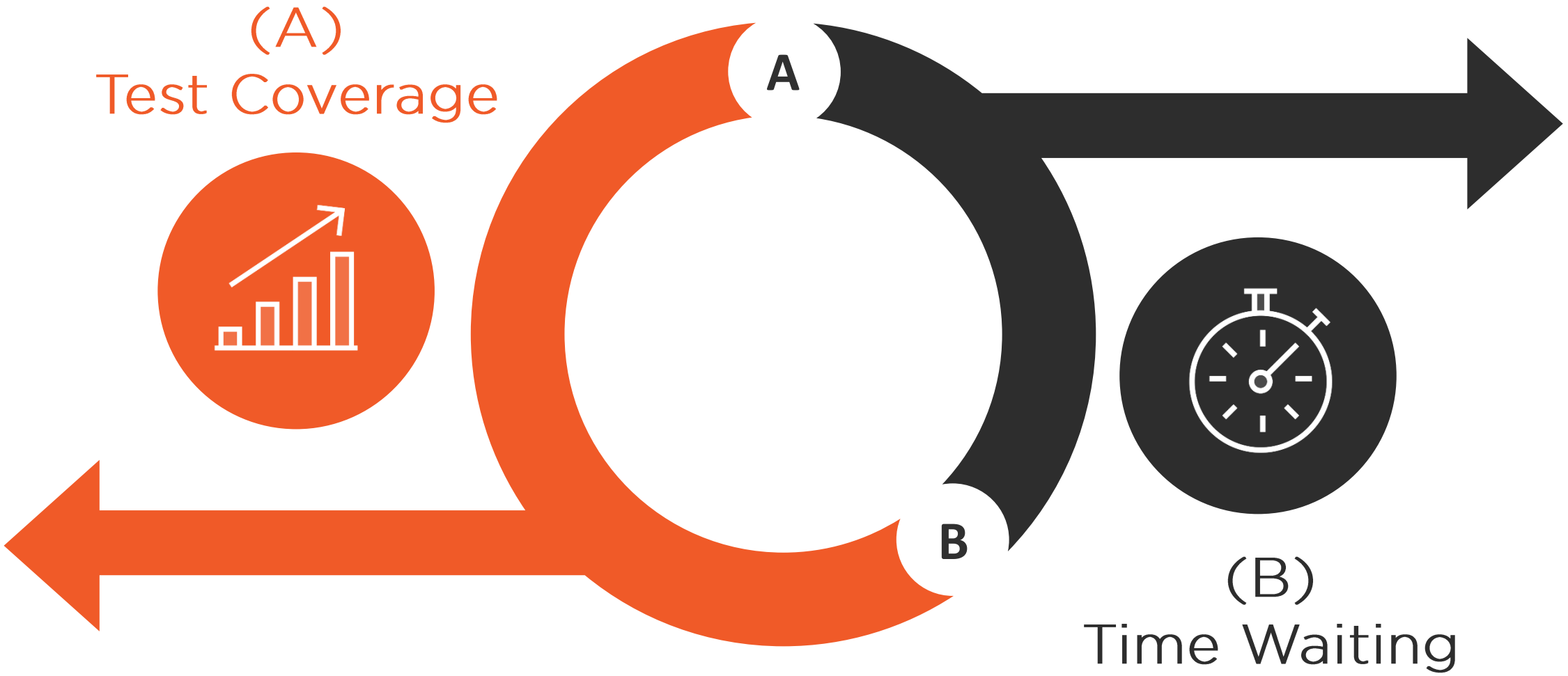
Why does it matter to have fast tests?

- # of tests grows...
- And so does the waiting time

Workaround: run tests less frequently

- Bigger merging conflicts (developers)
- Longer test failure investigation time (automation engineers)

Answer: Because fast continuous feedback matters

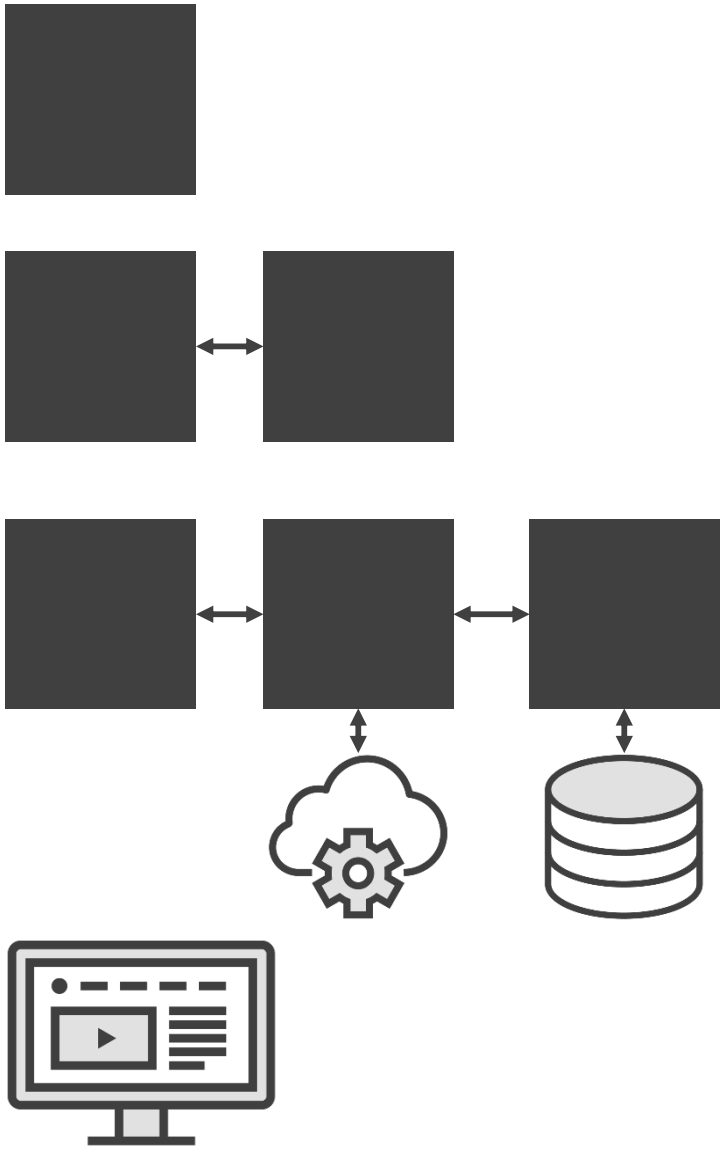

(A)
Test Coverage
A
B
(B)
Time Waiting

How to tell if a single test is fast?

Answer: context matters

- Unit vs. Integration vs. UI?

Unit	(ms)
Integration	(ms - s)
End-to-End (backend)	(s - m)
UI	(s)

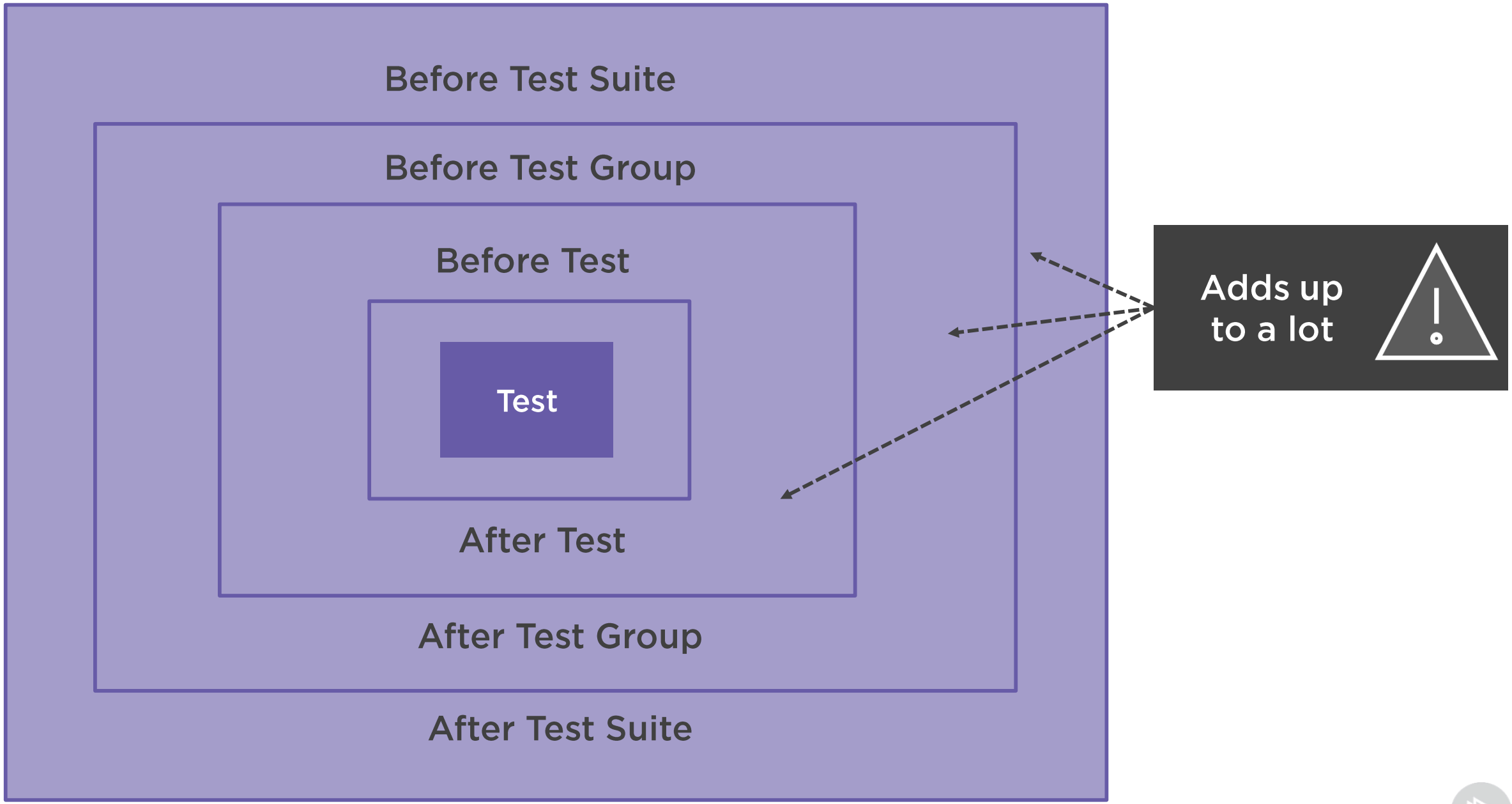


How to tell if a test suite is fast?

Wrong Answer: “1 or 2 hours”

- All depends on the # of tests

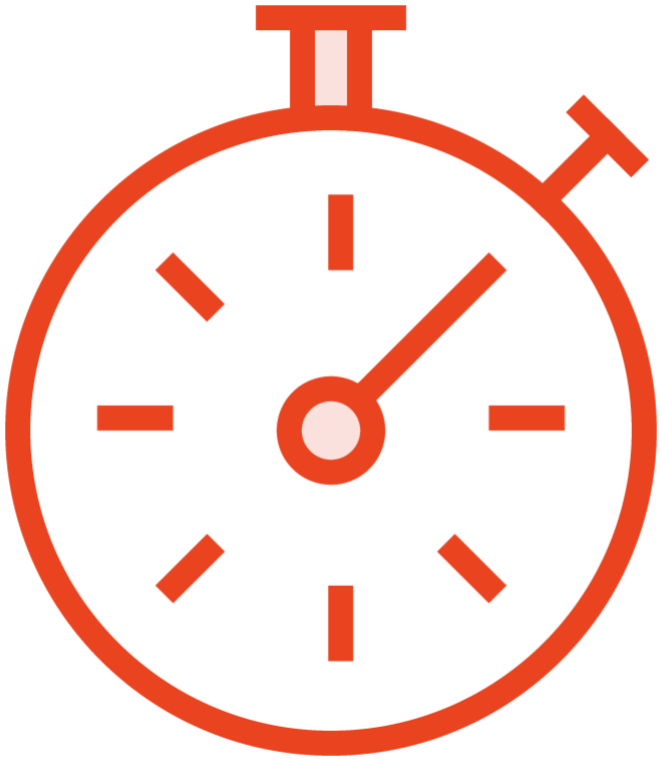
Right Answer: instead, focus on what’s happening between your tests!


Before Test Suite
Before Test Group
Before Test
Test
After Test
After Test Group
After Test Suite
Adds up to a lot

“If […] the setup and tear down together span an eighth of a second […], then you don’t have a fast test. You have a ludicrously slow test.”

“Agile in a Flash” Blog Authors

Making a Single Test Faster



Unit Tests:

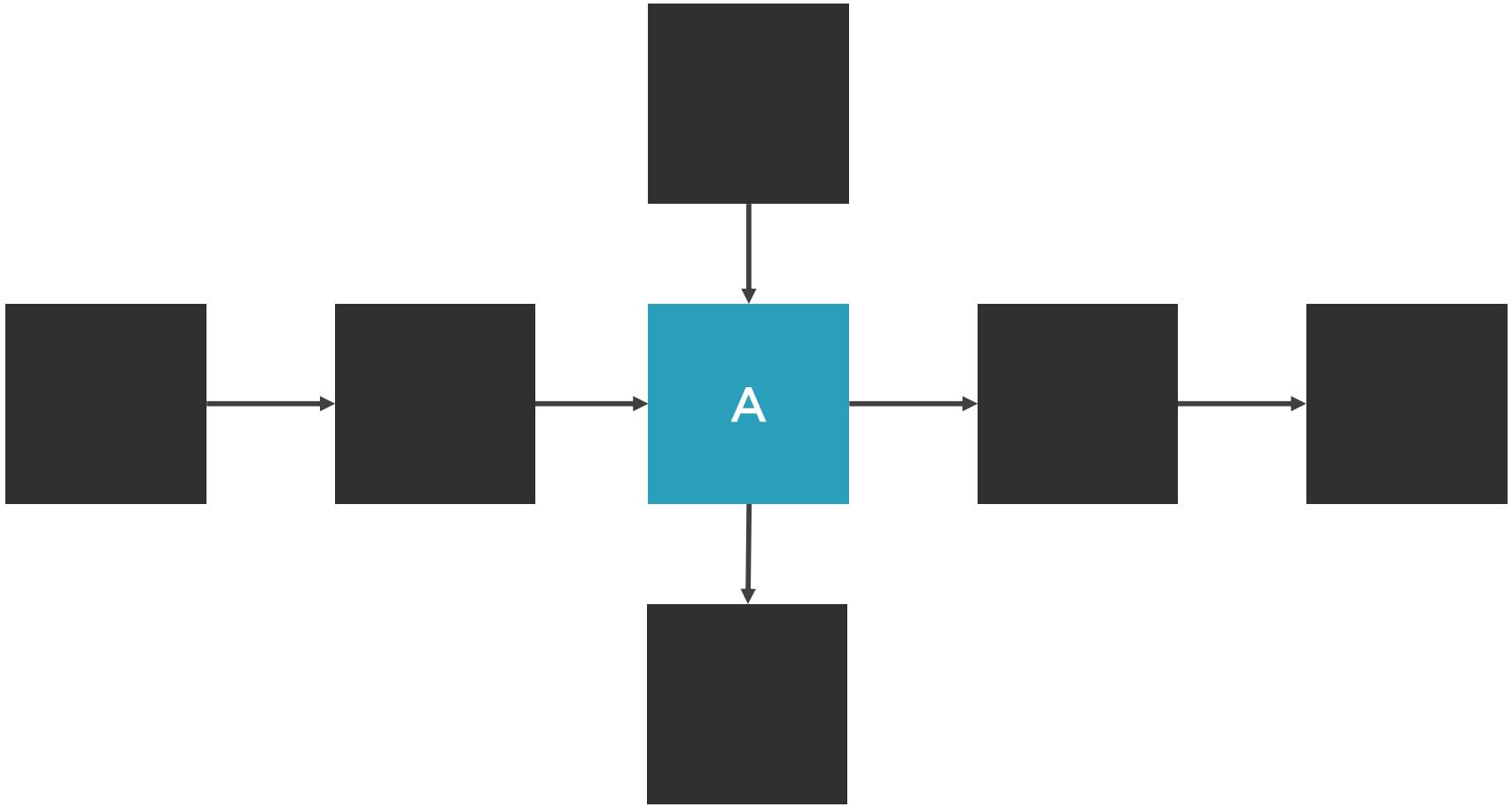
- Use mocks for all OOPDs
- OOPD: Out-Of-Process-Dependency (files, DBs, Web Services, etc.)

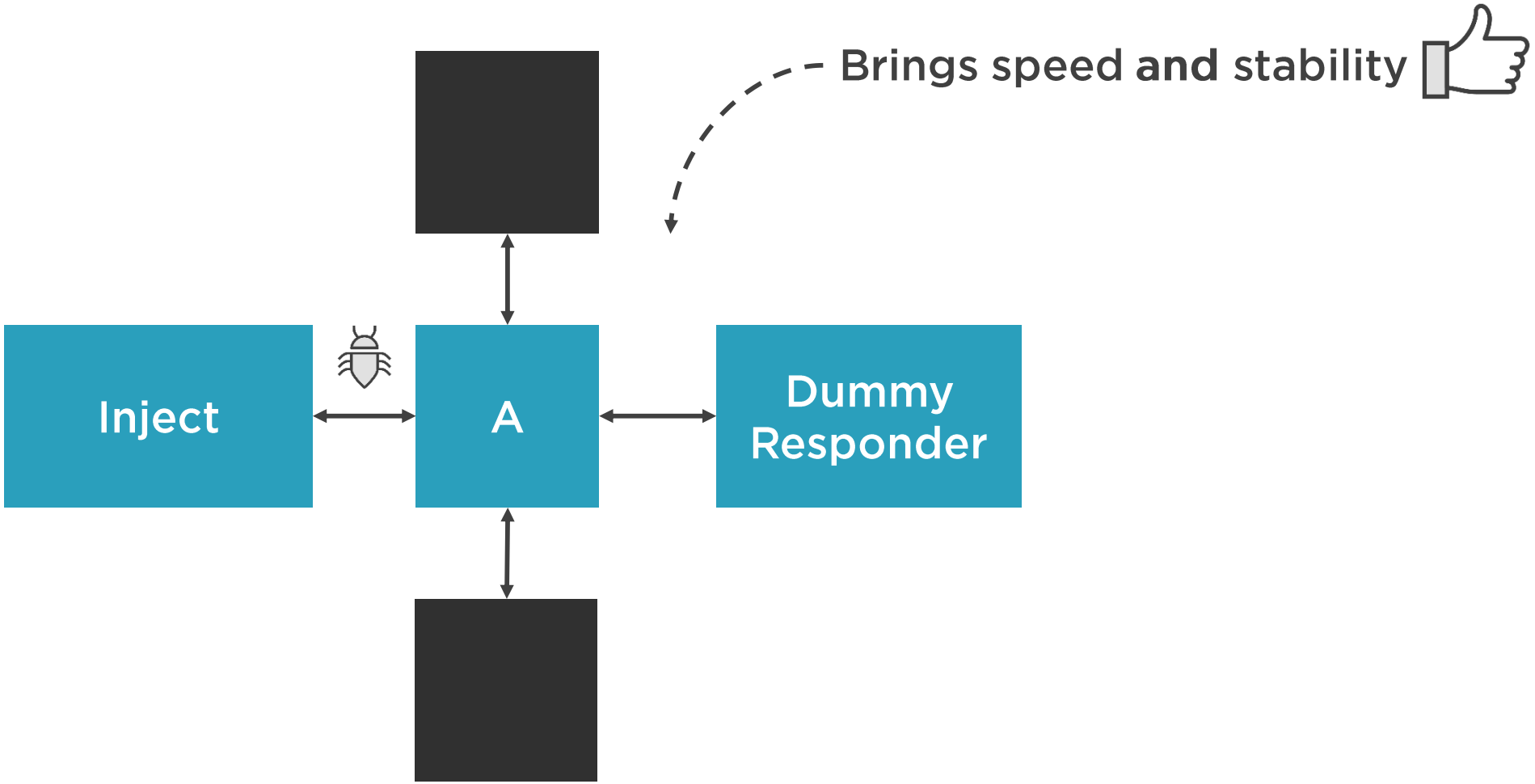
Mocking is not limited to unit tests

- Dummy services

All Tests:

- Prefer retry strategy to explicit wait
- E.g. retry 4 times, wait 1s between retries


A


Brings speed and stability
Inject
A
Dummy Responder

```
Wait.seconds(10); // stop failing already!
```

~~`Wait.seconds(4);`~~

~~`Wait.seconds(2);`~~

```
Assert.assertEquals(a, b);
```

“Stabilizing” the test

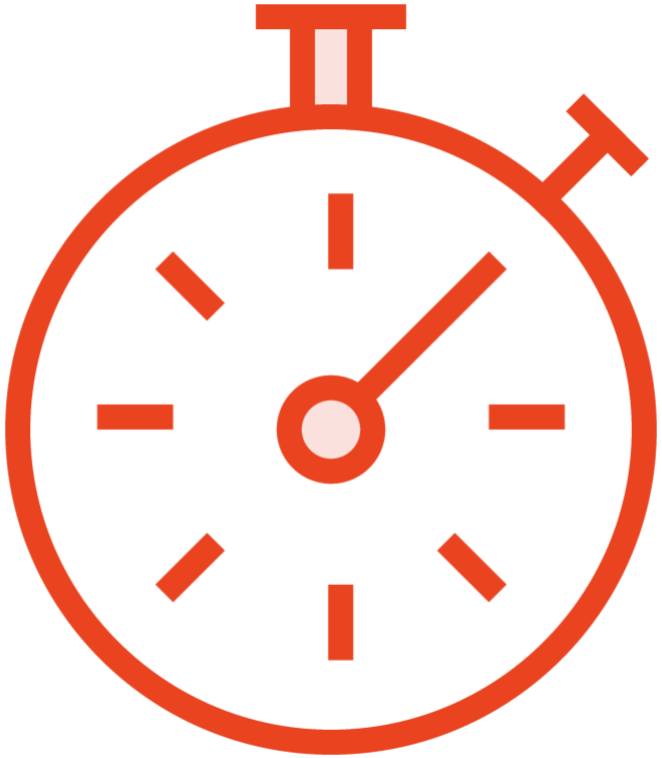
When the system under test (SUT) is sometimes a bit slow to respond

```
Wait wait = new FluentWait(WebDriver reference)
.withTimeout(Duration.ofSeconds(5))
.pollingEvery(Duration.ofSeconds(1))
.ignoring(Exception.class);
```

Stabilizing UI tests

Use Selenium FluentWait

Making a Test Suite Faster



Optimize your setup and cleanup

Cache things

Parallelize your suite early!

- Obvious one, but bear with me

@BeforeClass

x1

@BeforeMethod
doStuff();

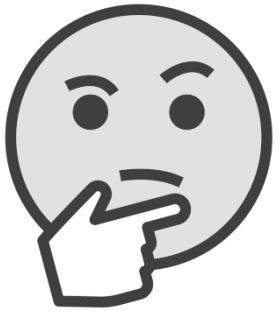
x3

FeautreTest.java

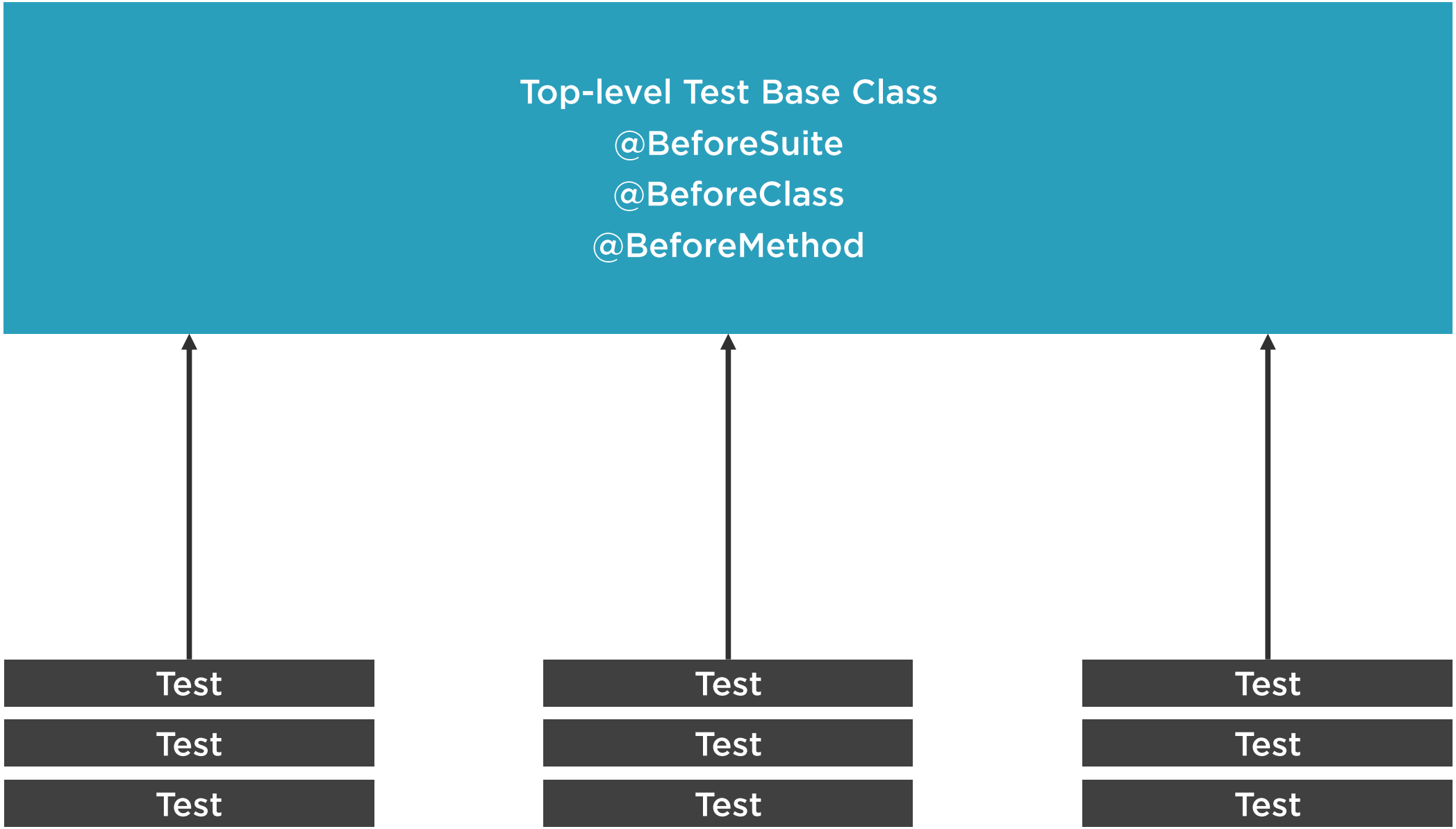
@Test

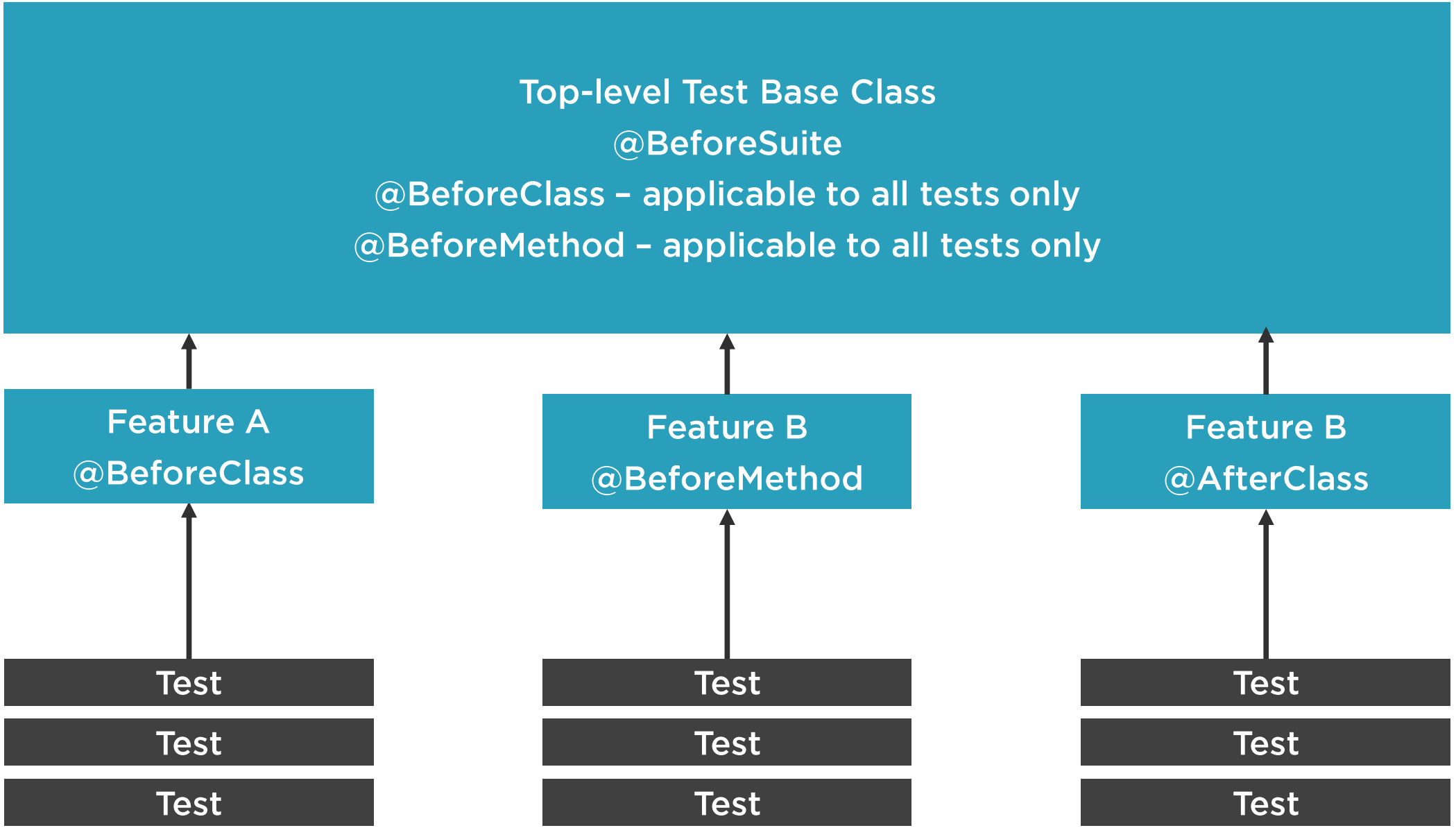
@Test

@Test



Can I run this less frequently?


Top-level Test Base Class
@BeforeSuite
@BeforeClass
@BeforeMethod
Test
Test
Test
Test
Test
Test
Test
Test
Test


Top-level Test Base Class
@BeforeSuite
@BeforeClass – applicable to all tests only
@BeforeMethod – applicable to all tests only
Feature A
@BeforeClass
Feature B
@BeforeMethod
Feature B
@AfterClass
Test
Test
Test
Test
Test
Test
Test
Test
Test

@BeforeClass

result = runQuery();

result = runQuery();

result;

```
@BeforeClass
 result = runQuery();
```

Demo coming up

```
checkThis(result);
```

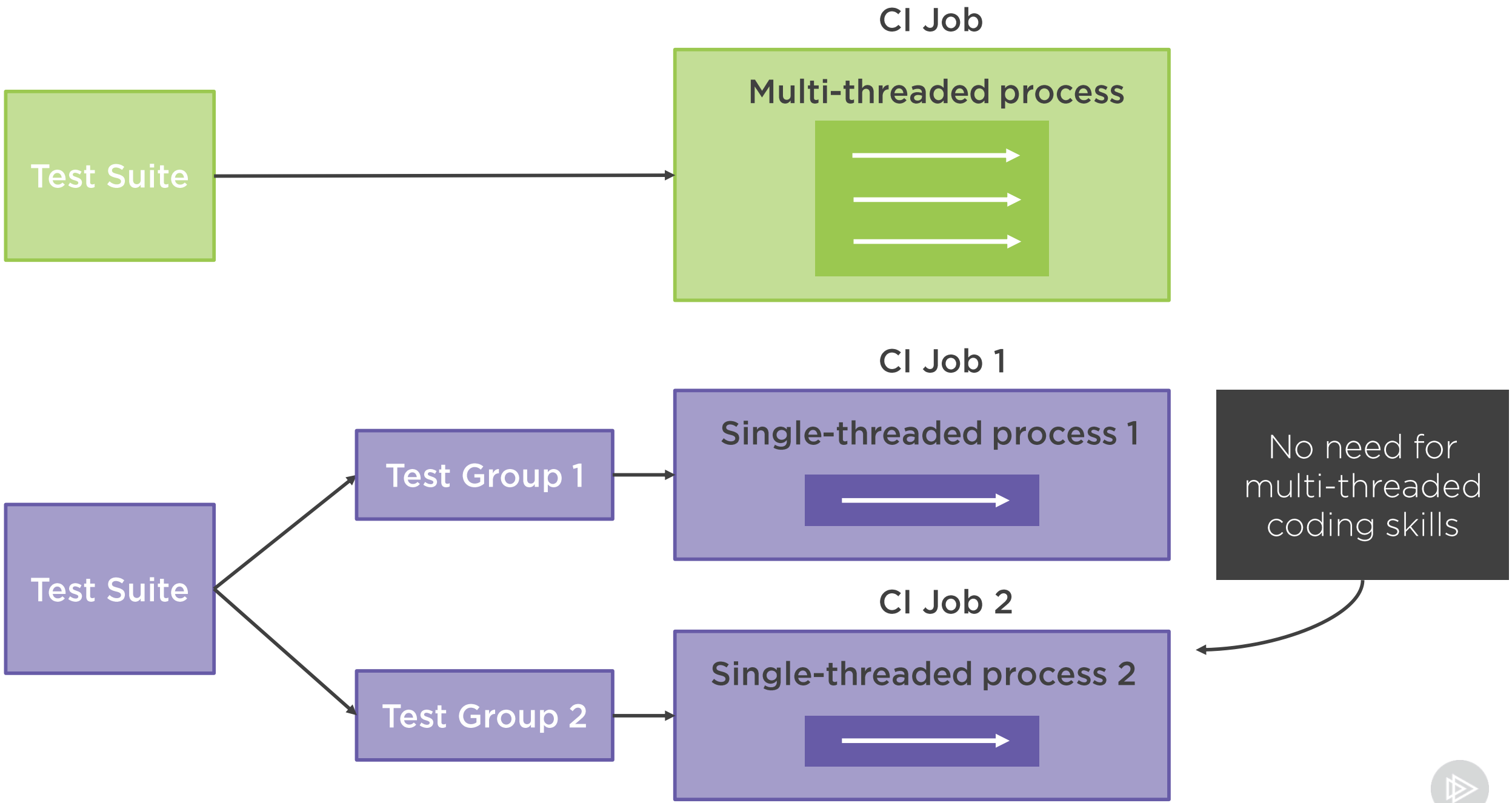
```
checkThat(result);
```

Reasons to Parallelize Early

Start saving time early on

Much easier to do while the automation code base is not too big and complex

- Discover shared resources
- Uncover unwanted interdependencies
- Easier to troubleshoot, reproduce and fix issues

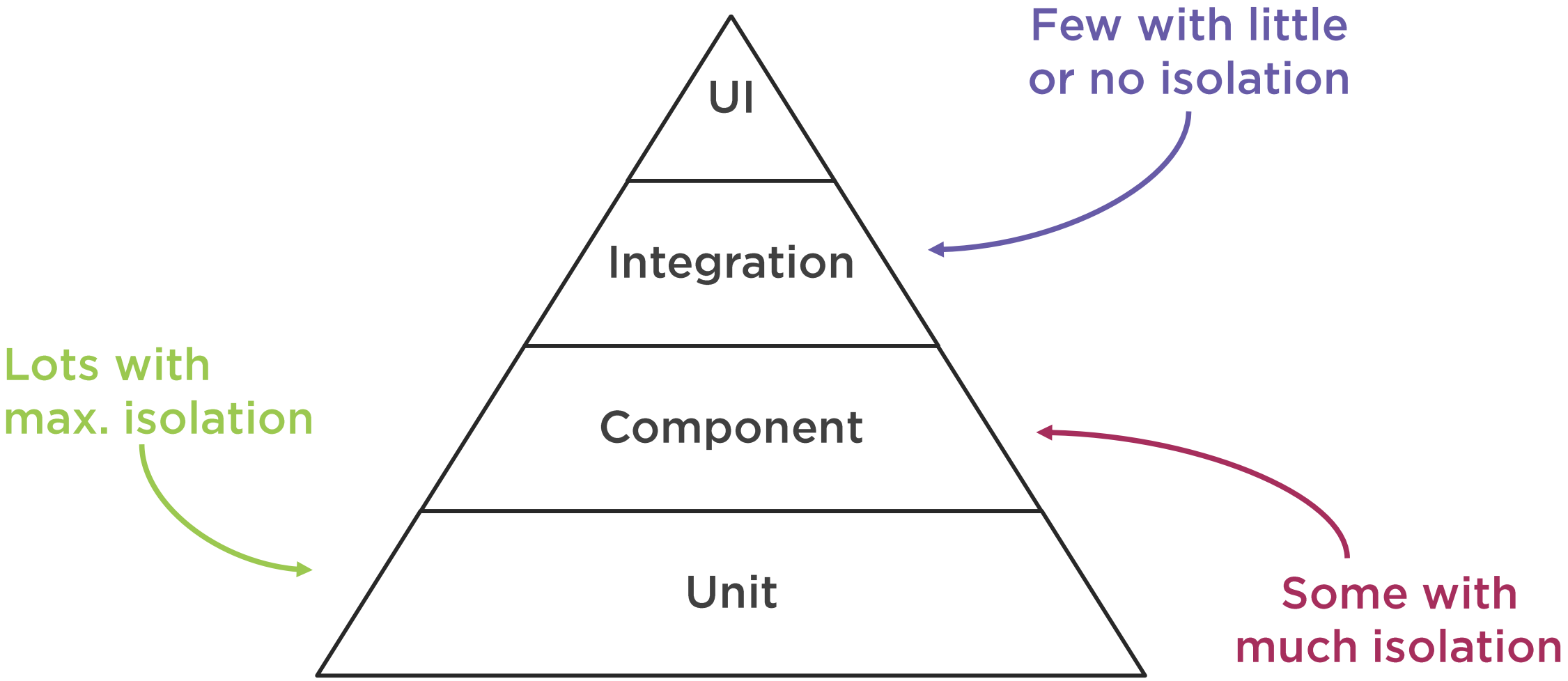

CI Job
Test Suite
Multi-threaded process
CI Job 1
Test Suite
Test Group 1
Single-threaded process 1
No need for multi-threaded coding skills
CI Job 2
Test Group 2
Single-threaded process 2

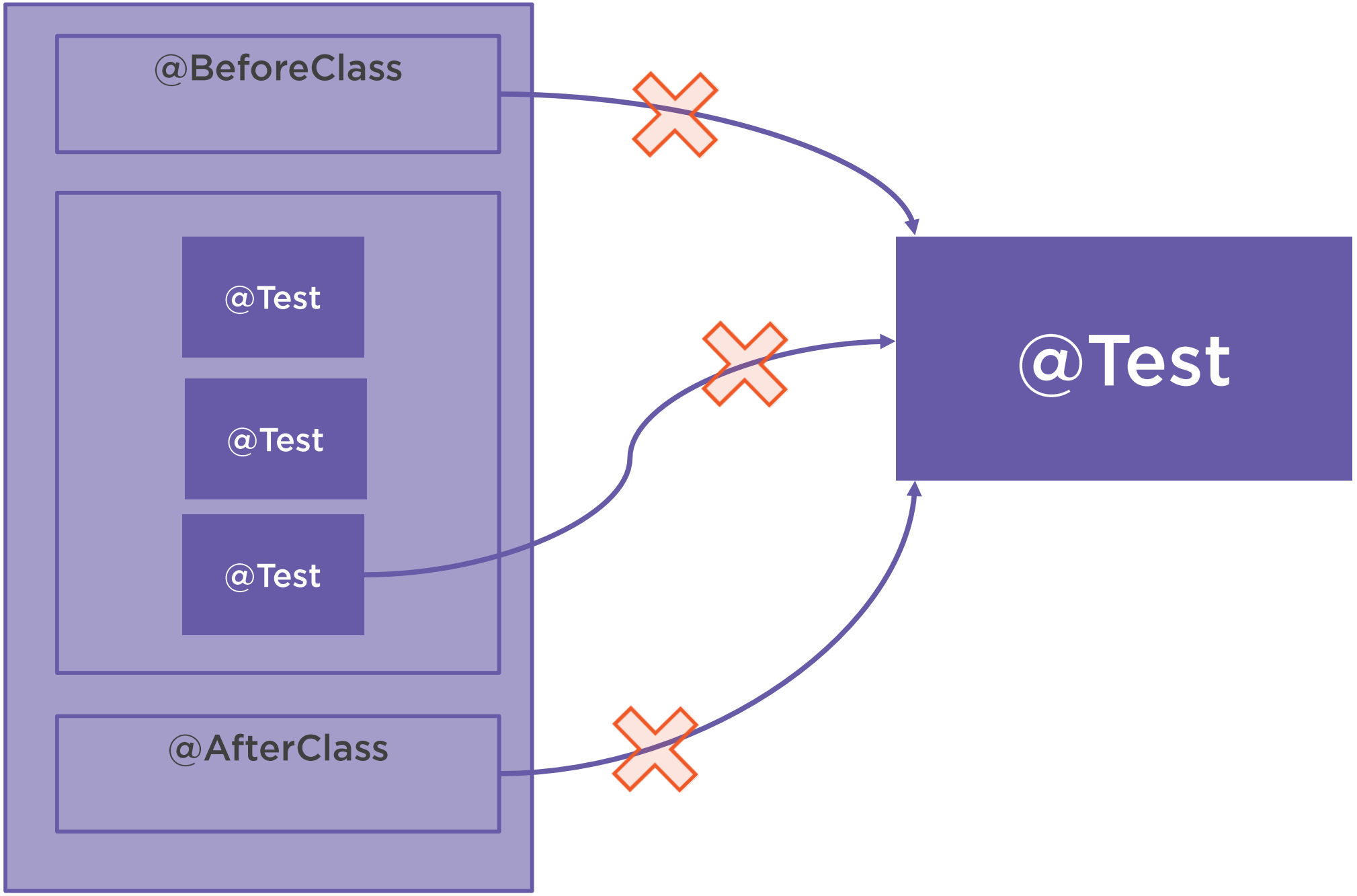
Demo

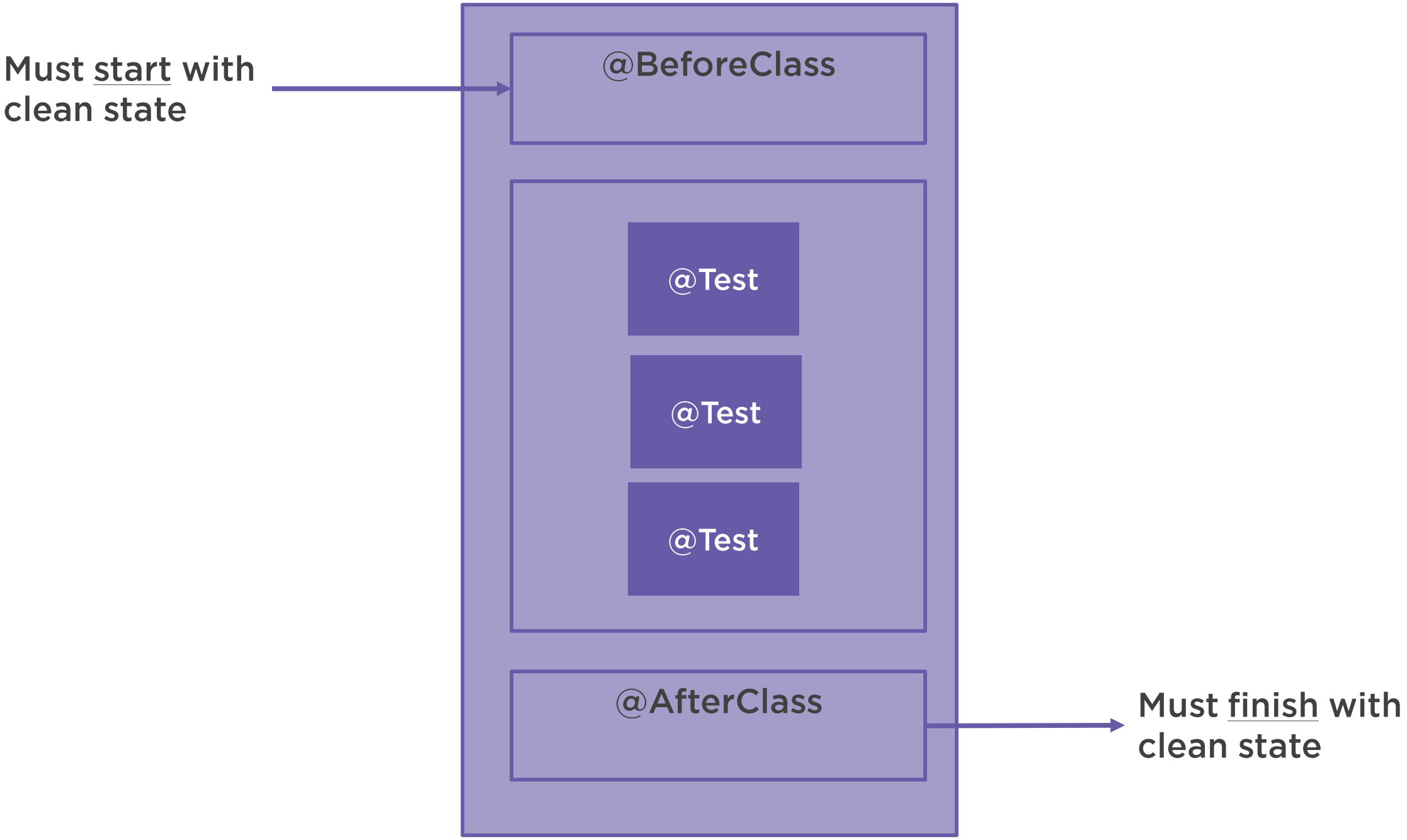
Making Web Service tests faster

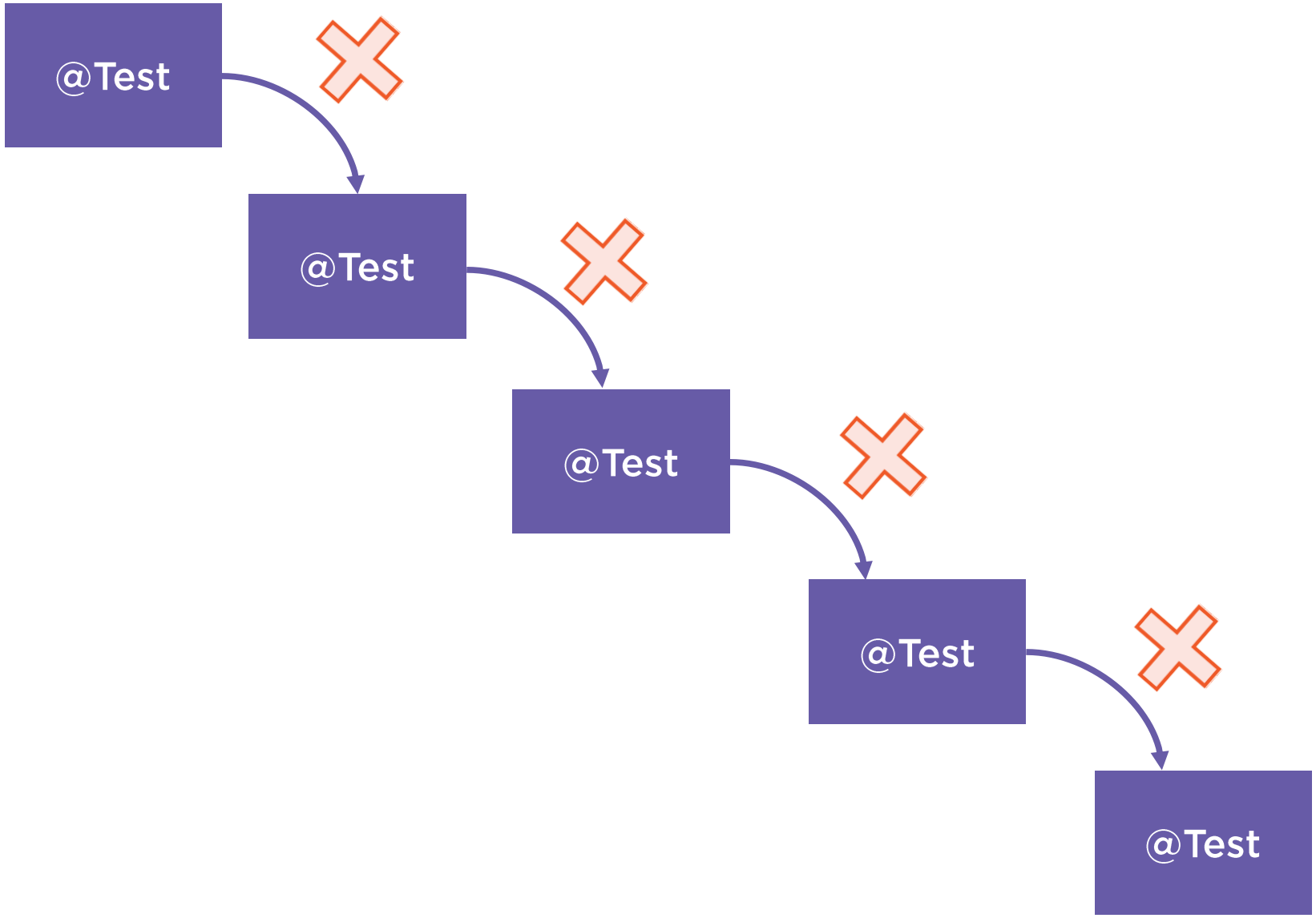
Headers:

- Status
- (and more)


UI
Integration
Component
Unit
Few with little
or no isolation
Lots with
max. isolation
Some with
much isolation


@BeforeClass
@Test
@Test
@Test
@AfterClass
@Test


Must start with clean state
@BeforeClass
@Test
@Test
@Test
@AfterClass
Must finish with clean state


@Test
@Test
@Test
@Test
@Test

How can I be sure that I do my clean up right and I don't leave any unwanted artifacts that might interfere with the tests that follow?

You should be able to run your tests any number of times, in any order

Are My Tests Repeatable?

Run a single test multiple times

- If it passes sometimes and fails other times – that's a problem

Run a newly added test together with other tests in a package

They can run any time of day

- Limitations do exist in the real world

Are My Tests Repeatable? (Continued)

They are time zone aware

- `System.currentTimeMillis();` will produce different results depending on where you are

Ask yourself: "Will this work if I were in a different time zone?"

- Familiarize yourself with Java 8 Date & Time API

```
<parameter name="count" value="5"/>
```

```
@Test(invocationCount = 5)
```

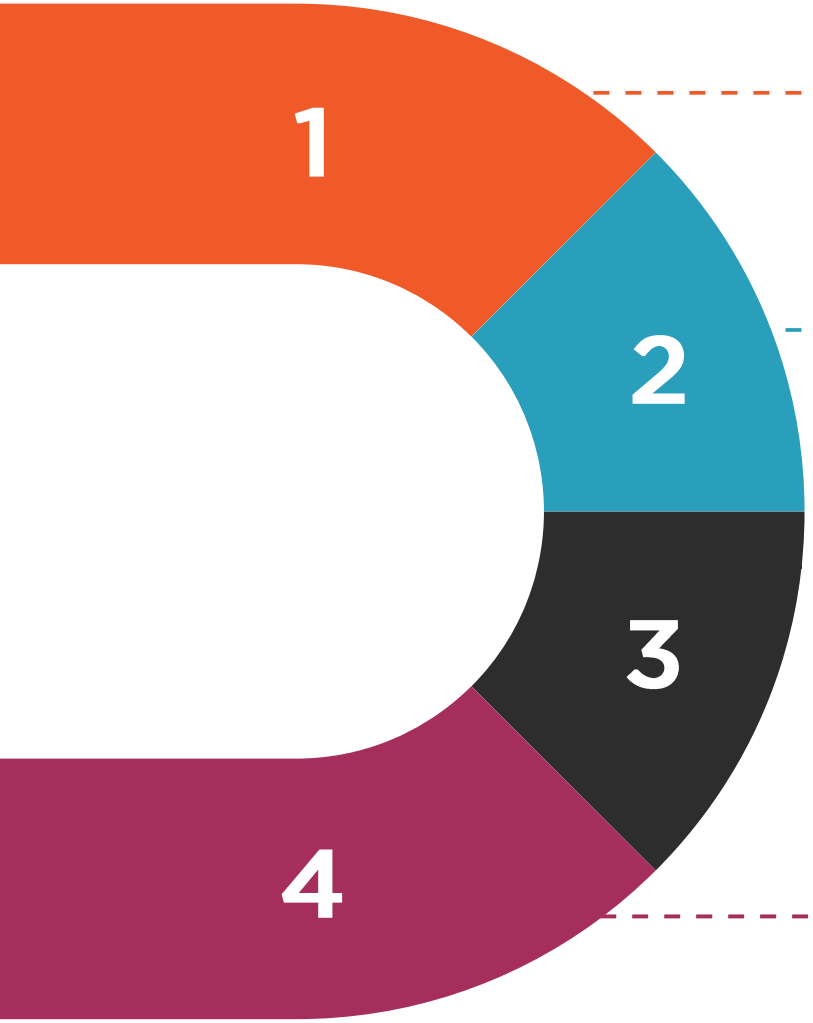
TestNG Example

```
@After____(alwaysRun= true)
```

TestNG Example

```
ZonedDateTime nowInNewYork = Instant.now()
                                .atZone(ZoneId.of("America/New_York")) ;
```

Java 8 Date & Time API Example

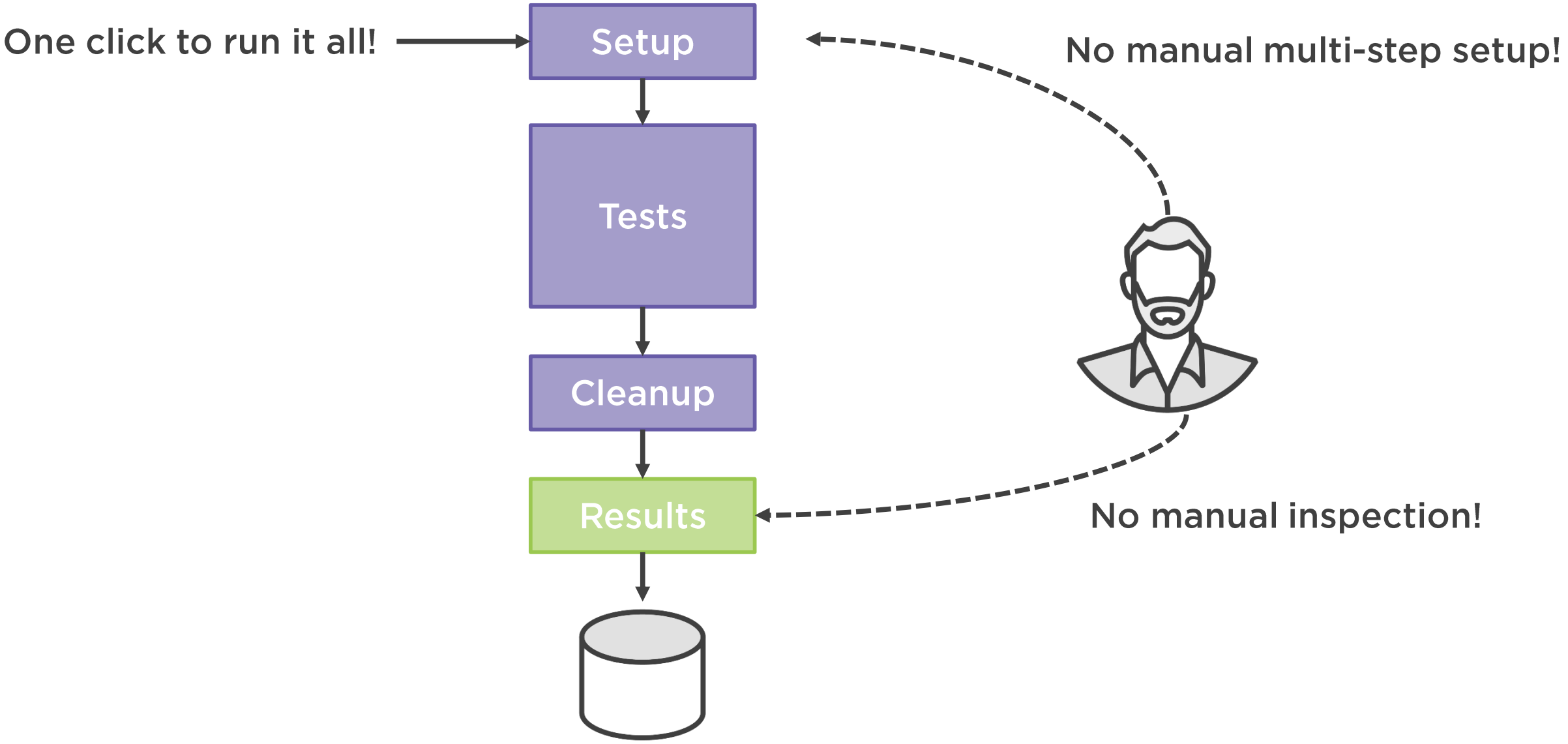


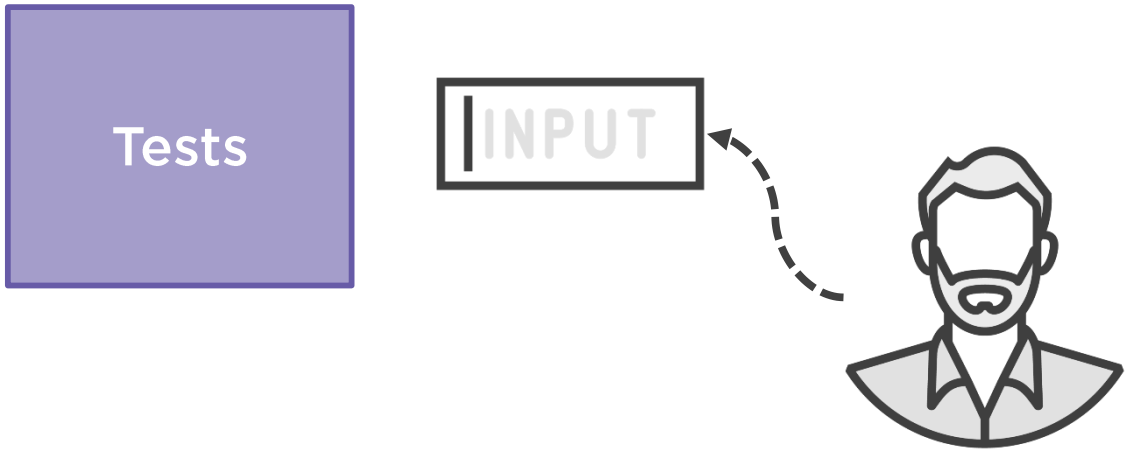
Unit
Run first – fastest feedback

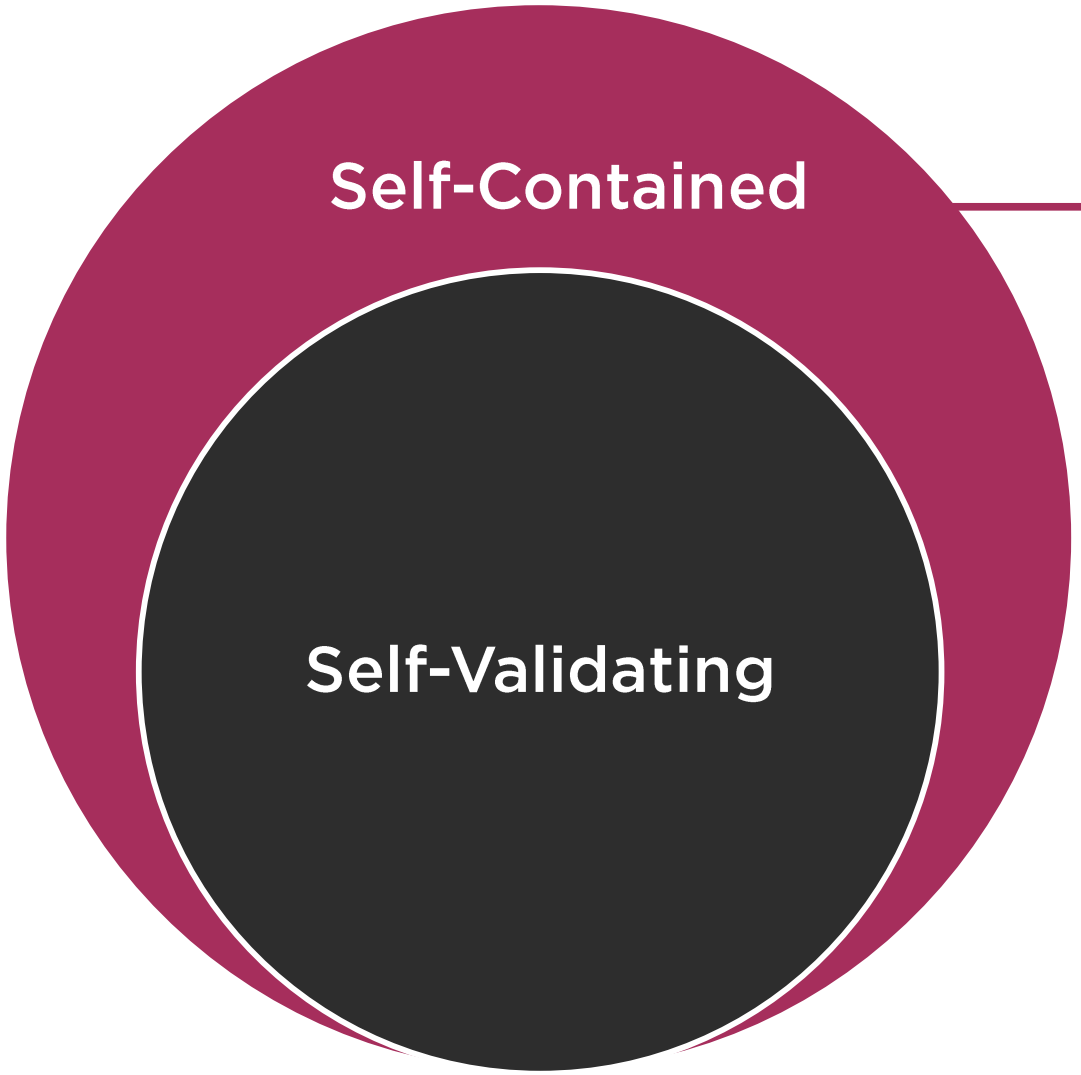
Component and Integration
Slower, but necessary

UI
(If any)

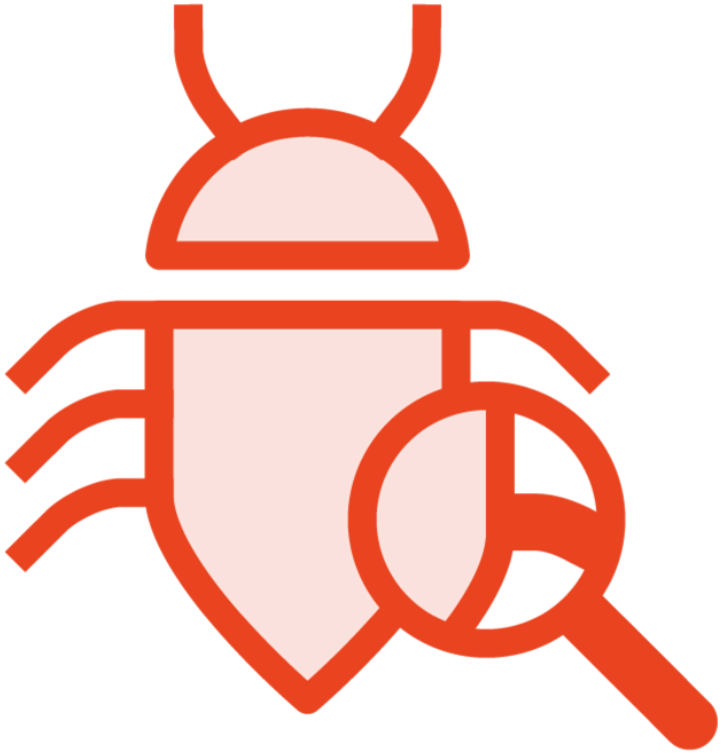
Acceptance Tests
Done by client – if any


One click to run it all!
Setup
Tests
Cleanup
Results
No manual multi-step setup!
No manual inspection!


Tests
INPUT



- ✓ Setup
- ✓ Test
- ✓ Verification
- ✓ Cleanup
- ✓ Result handling

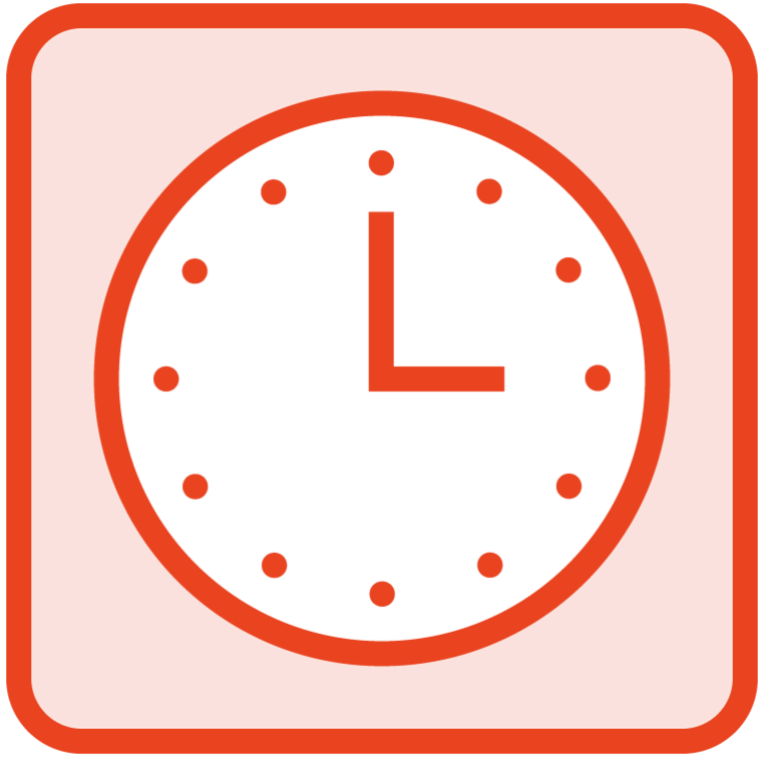


Most bugs hide in recently developed features

Newer tests have more (bug-finding) value

The sooner you create and automate tests – the better

Timely Tests



Unit:

- Practice TDD

Integration tests:

- At least think of test scenarios and write them down
- Better if you prepare helper code
- Best to write the test scripts in advance

Summary

Fast – depends on test type. Optimize setup and cleanup. Parallelize early.

Isolated and Independent – minimize ties with external dependencies. Make sure to cleanup properly.

Repeatable – run a single test or a group of tests any number of times in any order

Self-Validating – automate the entire process: setup, test, validation, cleanup, result handling

Timely – automate sooner rather than later

Up next: BICEP
(not the muscle)