

Lecture #14

Testing Frameworks &

Exam Discussions

Mobile Applications
Fall 2023

Types of Testing Software



System Testing	Smoke testing	System Integration Testing	Usability testing	Unit testing	User Acceptance testing
Soak Testing	Ad-hoc testing	Security Testing	Performance Testing	All Pairs testing	Beta Testing
Glass box Testing	Acceptance Testing	Sanity Testing	Regression Testing	Load Testing	Black Box testing
Accessibility Testing	Stress Testing	Static Testing	Retesting		Backward Compatibility Testing
Agile Testing	Stability Testing	API Testing	Risk-based Testing		Boundary Value Testing
Automated testing	Scalability Testing	Scalability Testing	Localization Testing		Big Bang Integration testing
Branch Testing	Gorilla Testing	Volume testing	Negative Testing		Bottom up Integration testing
Browser compatibility Testing	Vulnerability Testing	Component Testing	Non-functional testing		Keyword-driven Testing
Compatibility testing	Condition Coverage Testing	Dynamic Testing	Pair Testing		End-to-end Testing
Happy path testing	Integration Testing	Decision Coverage Testing	White box Testing	Internationalization Testing	Exploratory Testing
Functional Testing	Fuzz Testing	Interface Testing			GUI (Graphical User Interface) testing

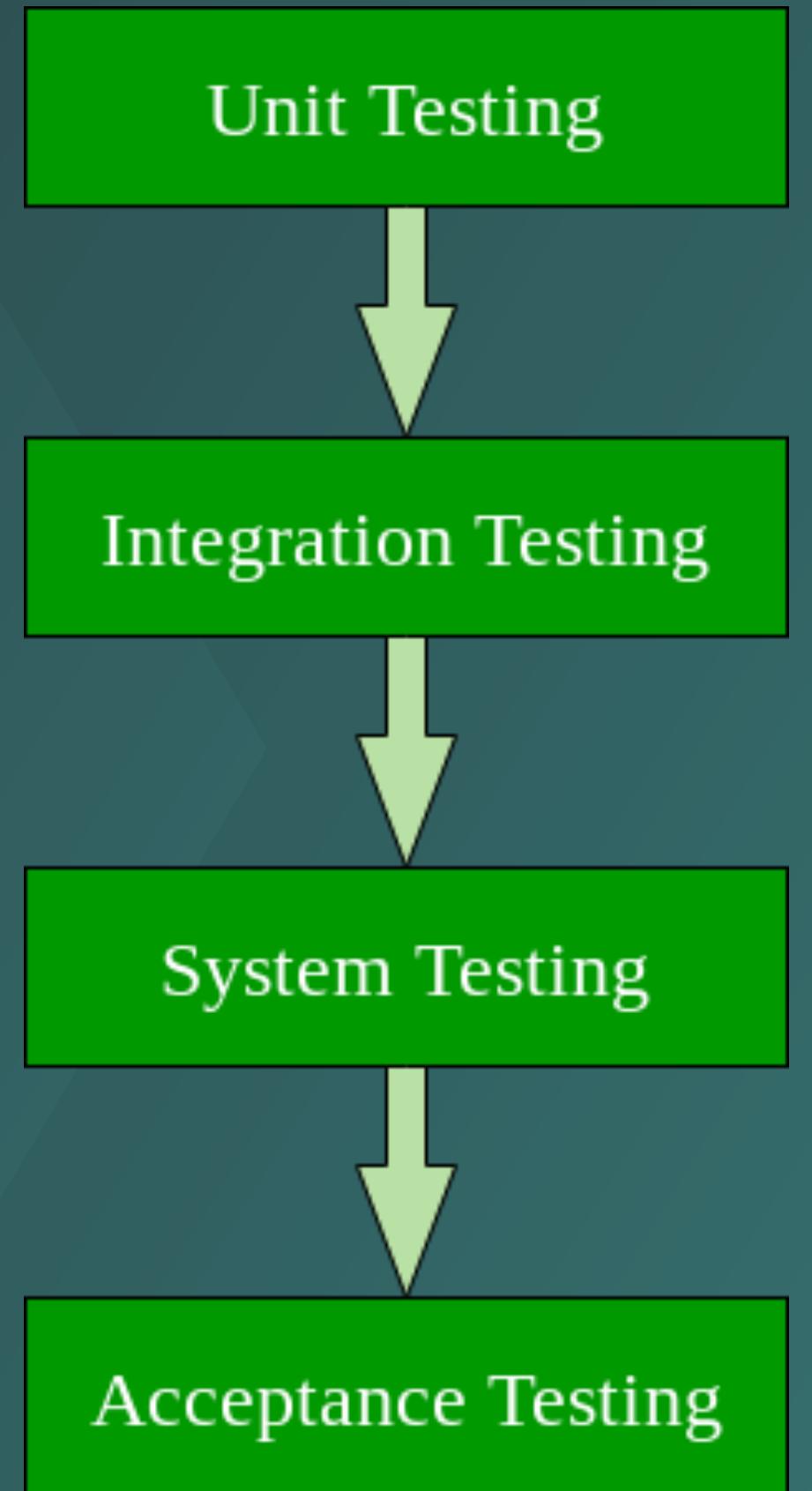
Types of Testing Software

Software Testing can be broadly classified into two types:

- Manual Testing
- Automation Testing

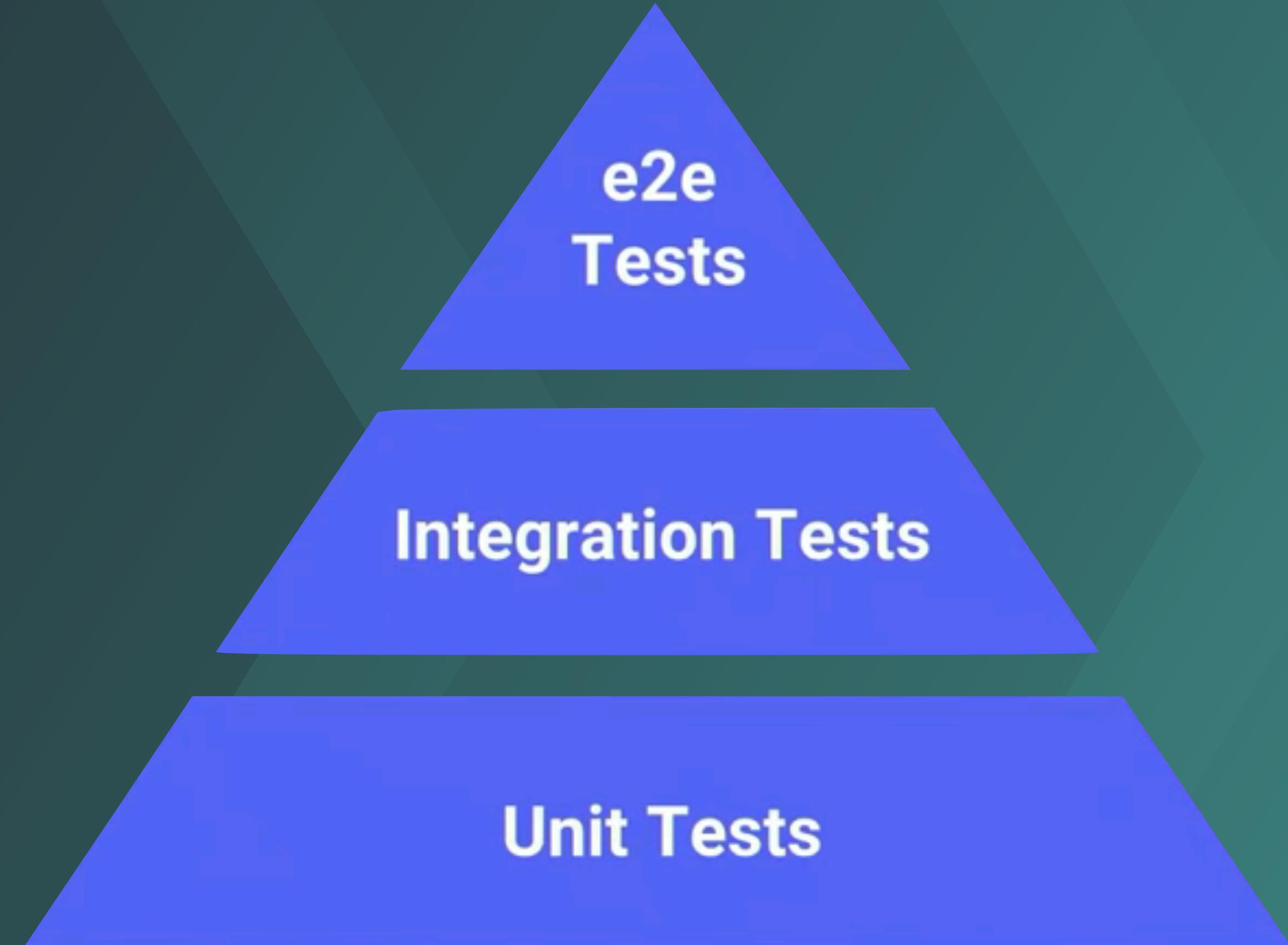


Automation Testing Levels



Advantages

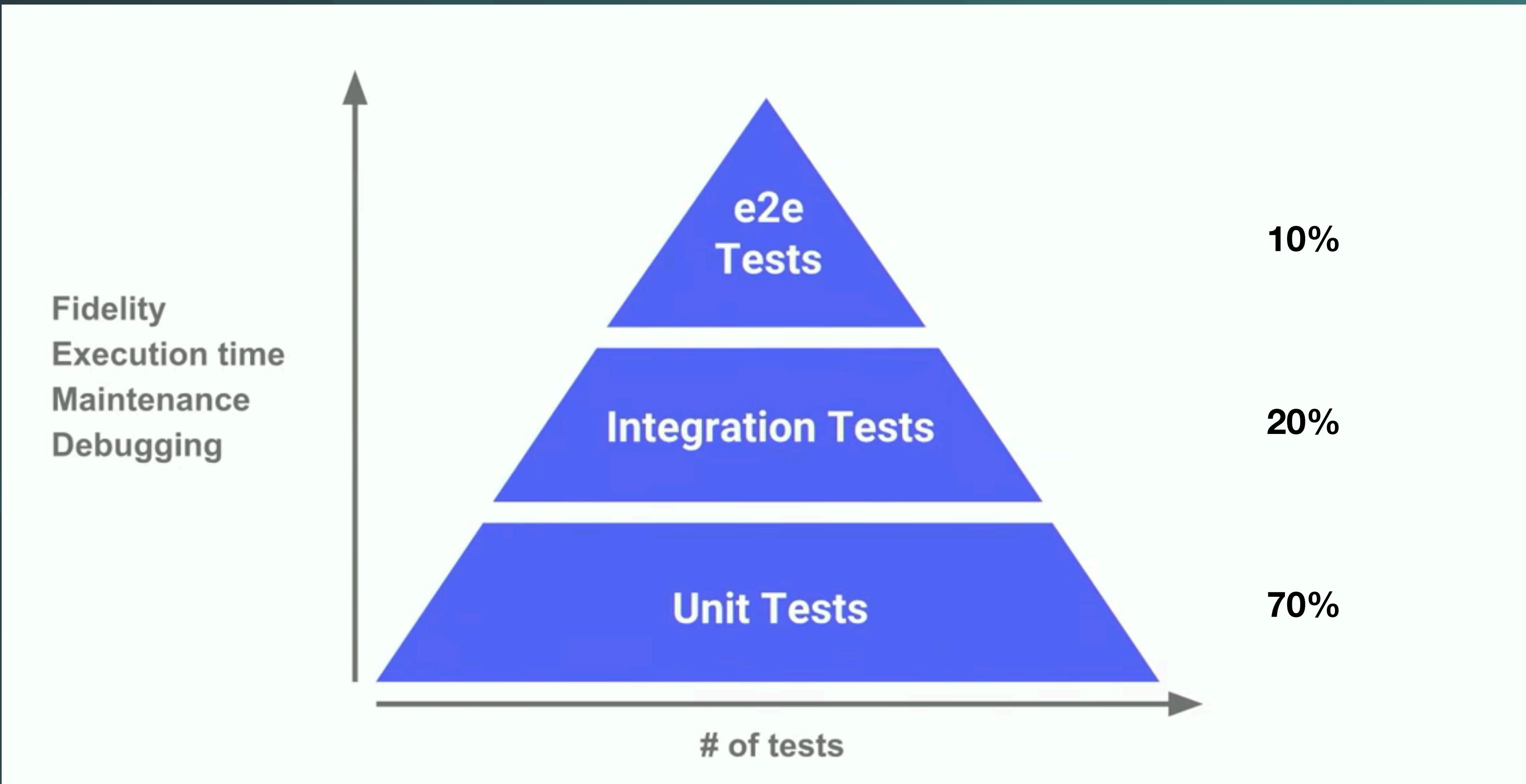
- Rapid feedback.
- Early failure detection.
- Safer code refactoring.
- Stable development velocity.

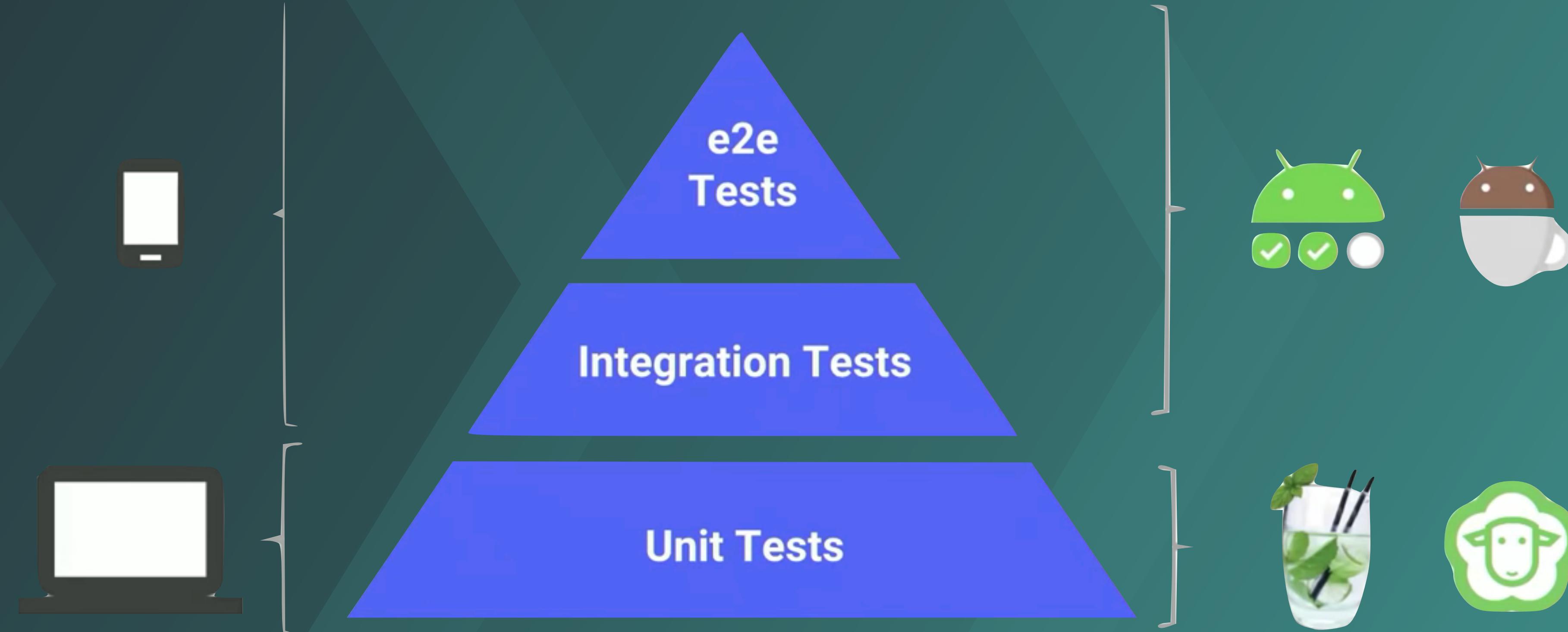


e2e
Tests

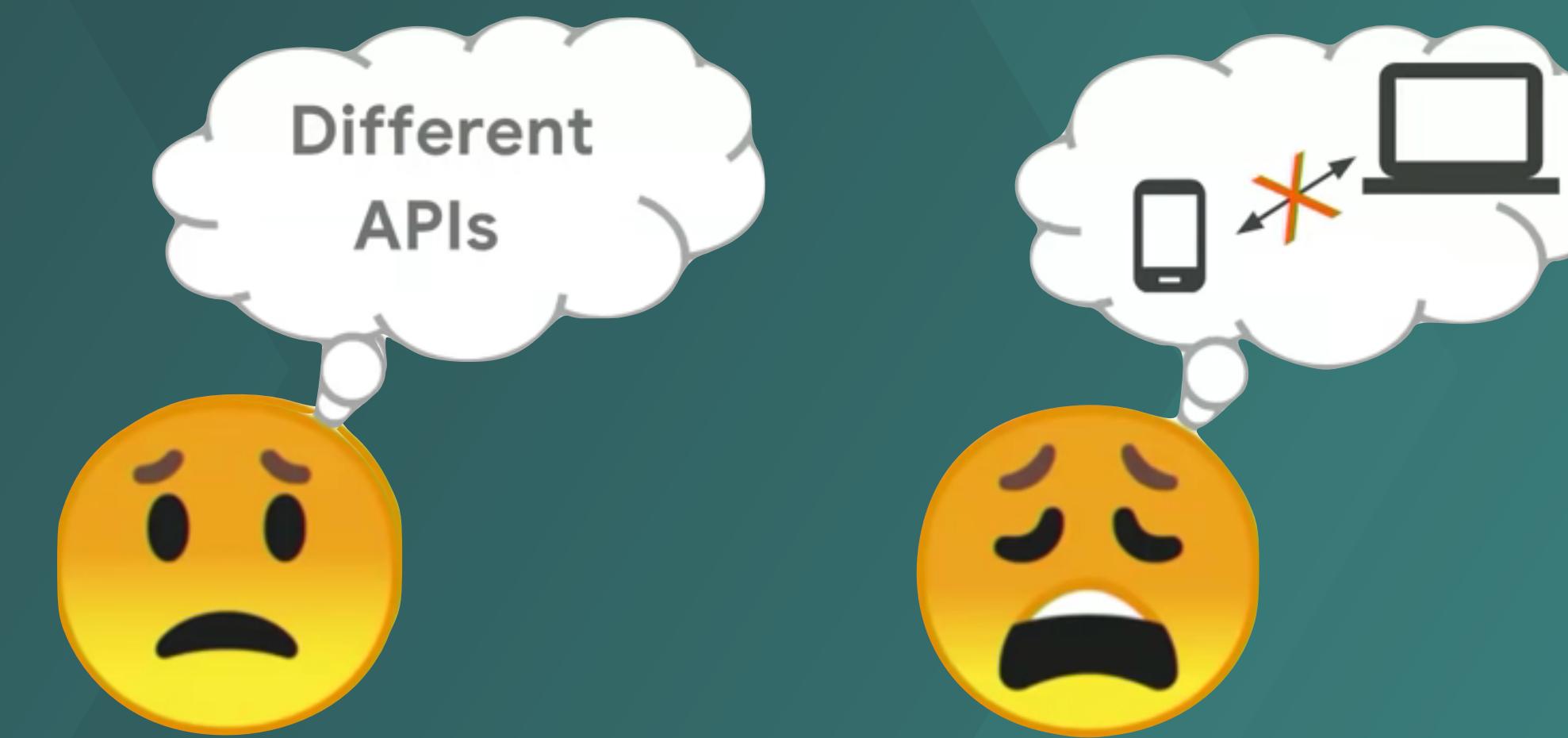
Integration Tests

Unit Tests





Test writing crisis

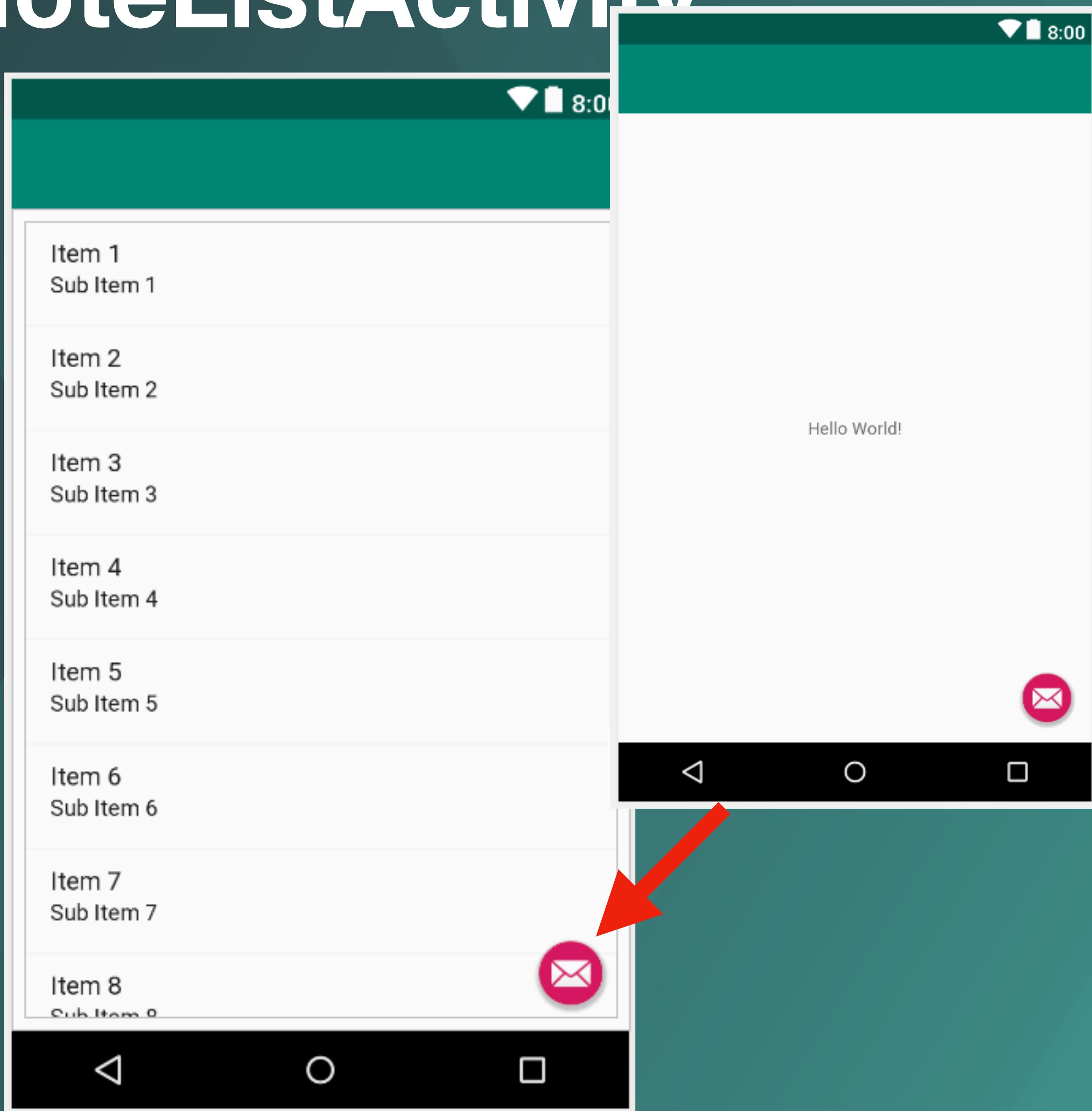


Test Structure

```
class WellStructuresTest{  
    //..  
    fun givenCondition_whenAction_thenShouldDo(){  
        // GIVEN - Setup condition  
  
        // WHEN - The tested action  
  
        // THEN - An assertion to validate the action  
    }  
    //...  
}
```

- Focus on specific behavior.
- Test behaviors independently.
- LESS is MORE. Keep tests understandable and in isolation.

NoteListActivity



Mockito



```
@RunWith(MockitoJUnitRunner::class)
class MockitoTest {
    @Spy var spyActivity = NoteListActivity()
    @Captor lateinit var intentCaptor: ArgumentCaptor<Intent>
    @Captor lateinit var clickCaptor:
        ArgumentCaptor<NoteListActivity.ClickHandler>

    fun testTitle() {
        `when`(spyActivity.findViewById(R.id.title))
            .thenReturn(mock<TextView>())
        clickCaptor.value.click()

        verify(spyActivity).startActivity(intentCaptor.capture())
    }
}
```

Robolectric



```
@RunWith(RobolectricTestRunner::class)
class RobolectricTest {

    @Test
    fun testTitle() {
        val activity =
            Robolectric.setupActivity(NoteListActivity::class.java)

        ShadowView.clickOn(activity.findViewById(R.id.title))

        assertEquals(
            ShadowApplication.getInstance()
                .peekNextStartedActivity().action,
            "android.intent.action.EDIT"
        )
    }
}
```

Espresso

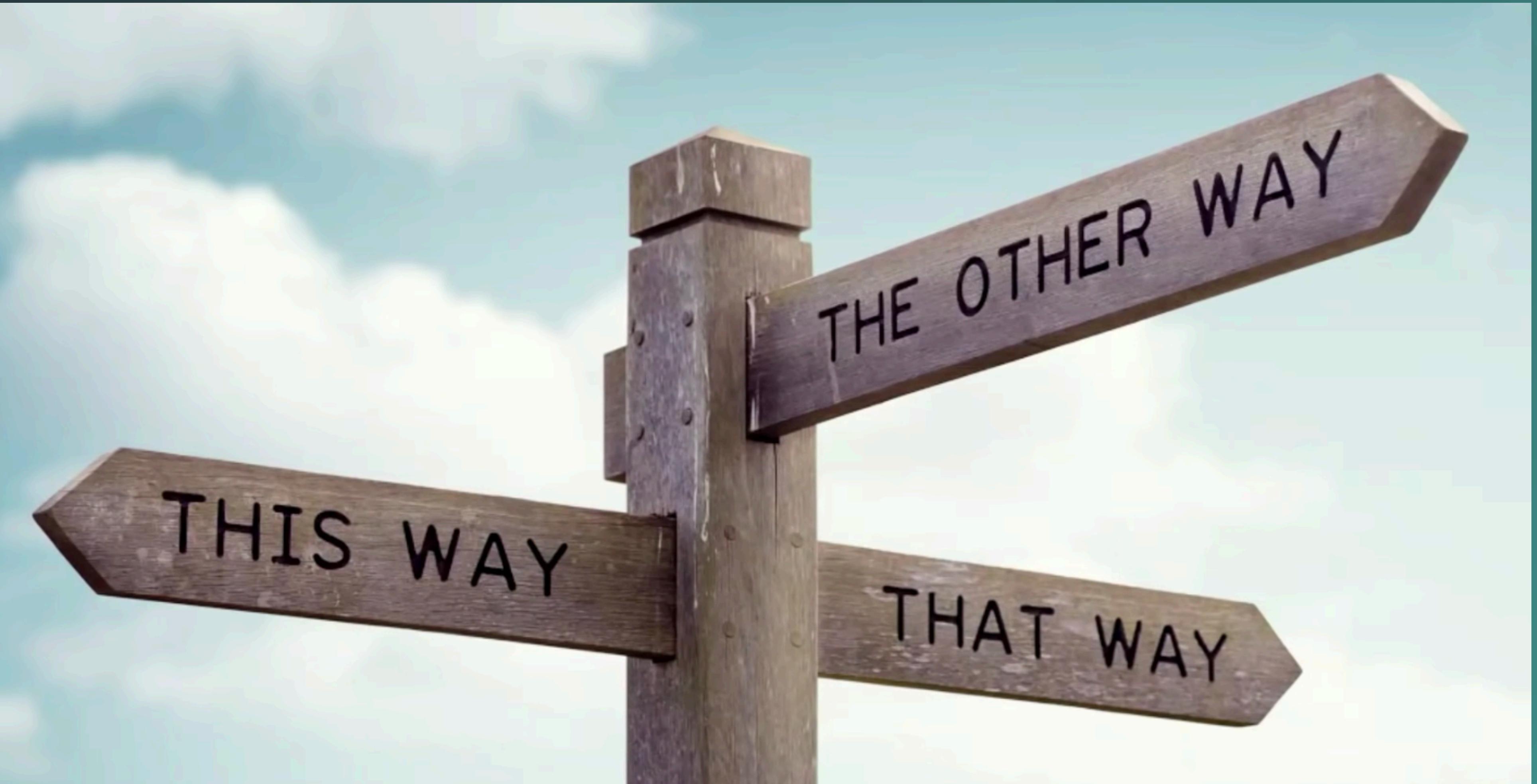


```
@RunWith(AndroidJUnit4::class)
class OnDeviceTest {

    @get:Rule
    val rule = ActivityTestRule(NoteListActivity::class.java)

    @Test
    fun testTitle() {
        onView(withId(R.id.fab)).perform(click())

        intended(hasAction(equalTo("android.intent.action.EDIT")))
    }
}
```



Android Test

Part of Jetpack

- Includes existing libraries.
- New APIs and Kotlin.
- Available on/off device.
- Open source.

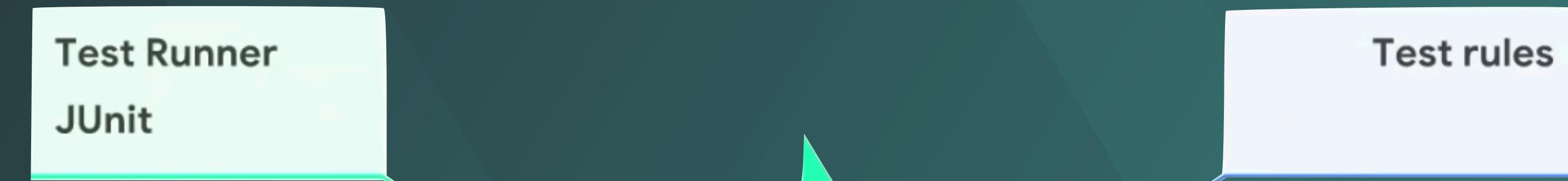


Test Runner
JUnit

Scaffolding

//SCAFFOLDING

```
@RunWith(AndroidJUnit4::class)
class SimpleUnifiedTest {
    @Before
    fun setup() {
        val context = InstrumentationRegistry.getTargetContext()
    }
}
```



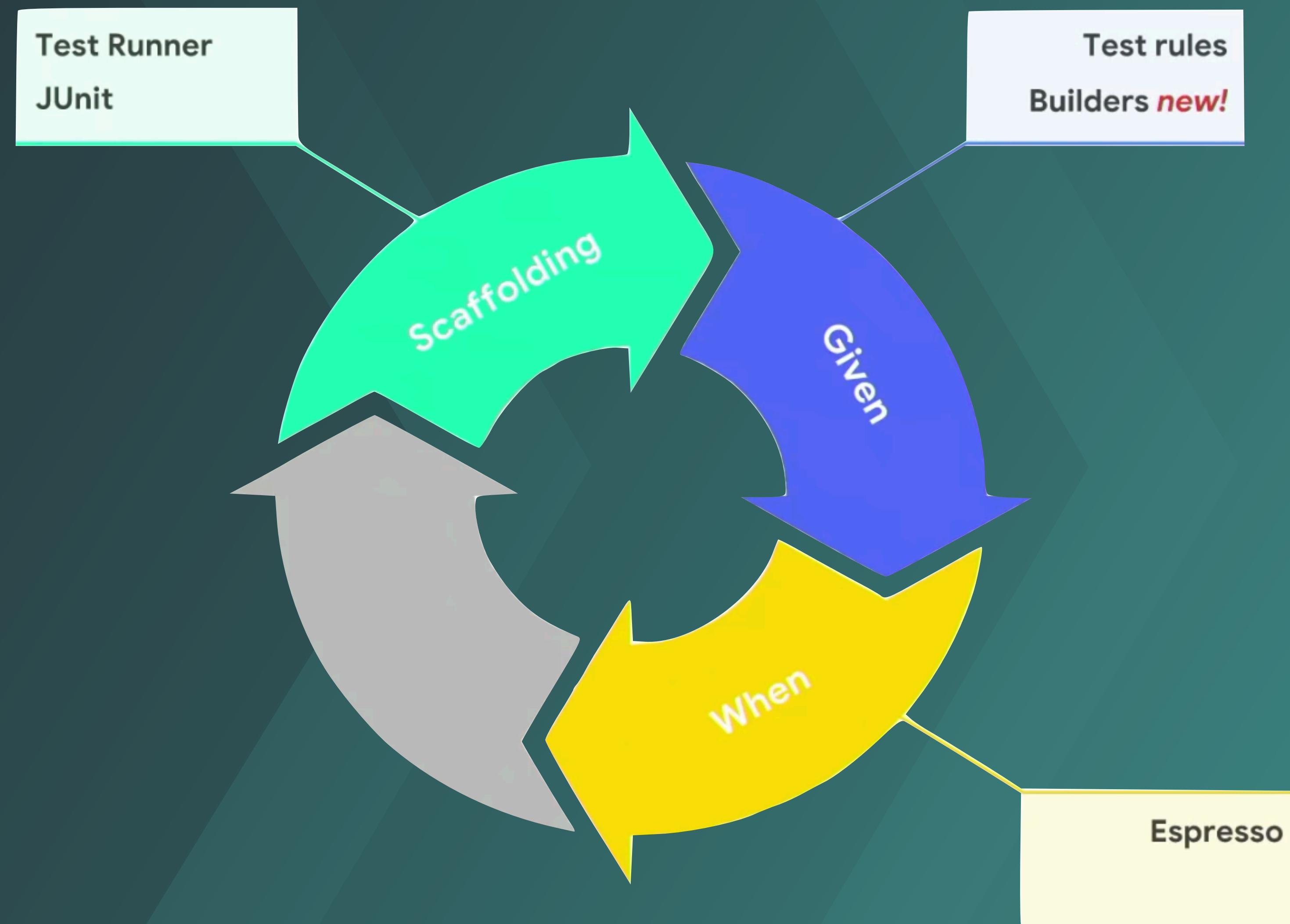
//GIVEN

```
@RunWith(AndroidJUnit4::class)
class SimpleUnifiedTest {
    @get:Rule
    val rule = ActivityTestRule(NoteListActivity::class.java)
}
```

//GIVEN

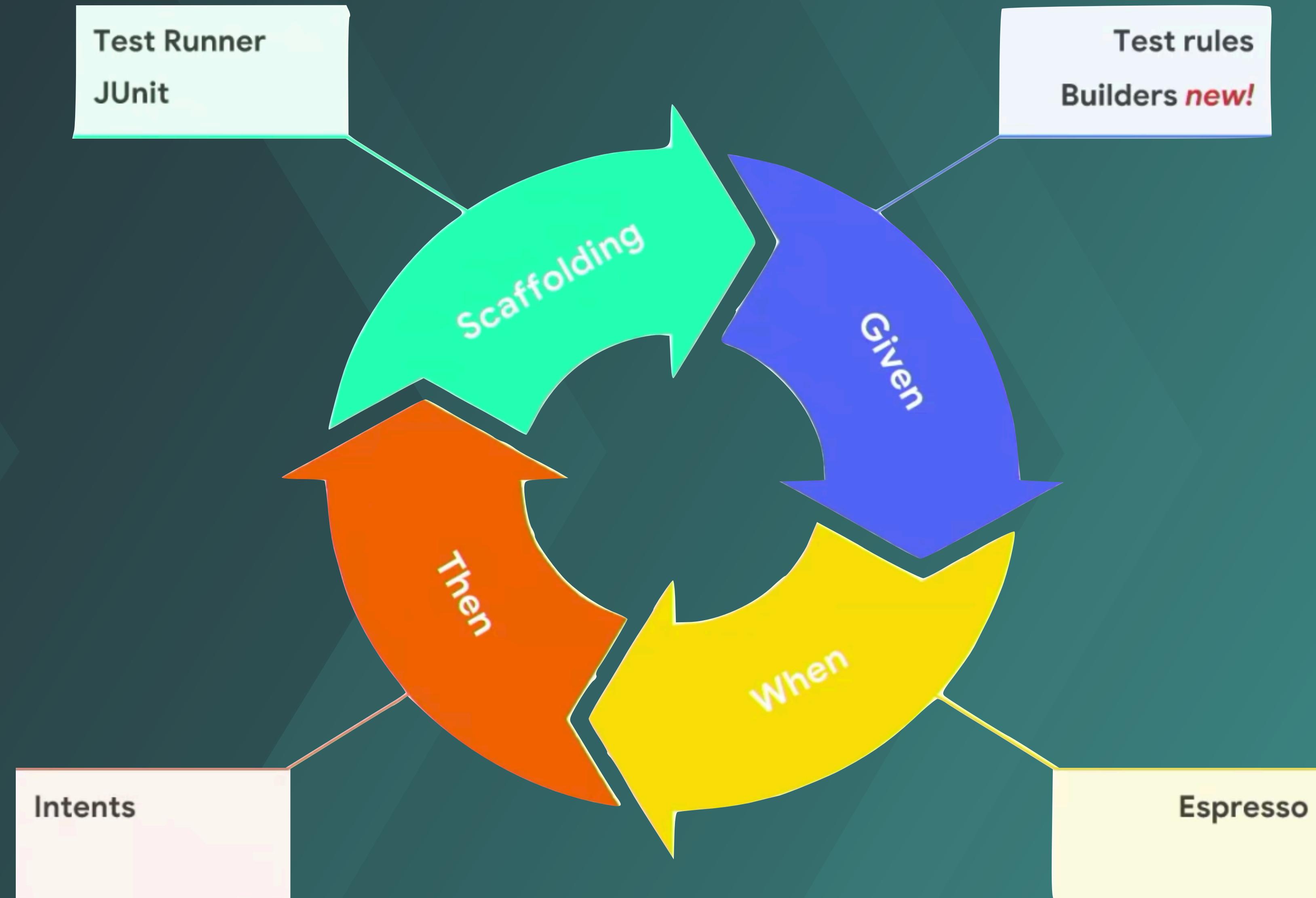
```
@RunWith(AndroidJUnit4::class)
class SimpleUnifiedTest {
    @Test
    fun testMotionEvents() {
        val motionEvent =
            buildMotionEvent().setAction(MotionEvent.ACTION_DOWN)
    }
}
```





//WHEN

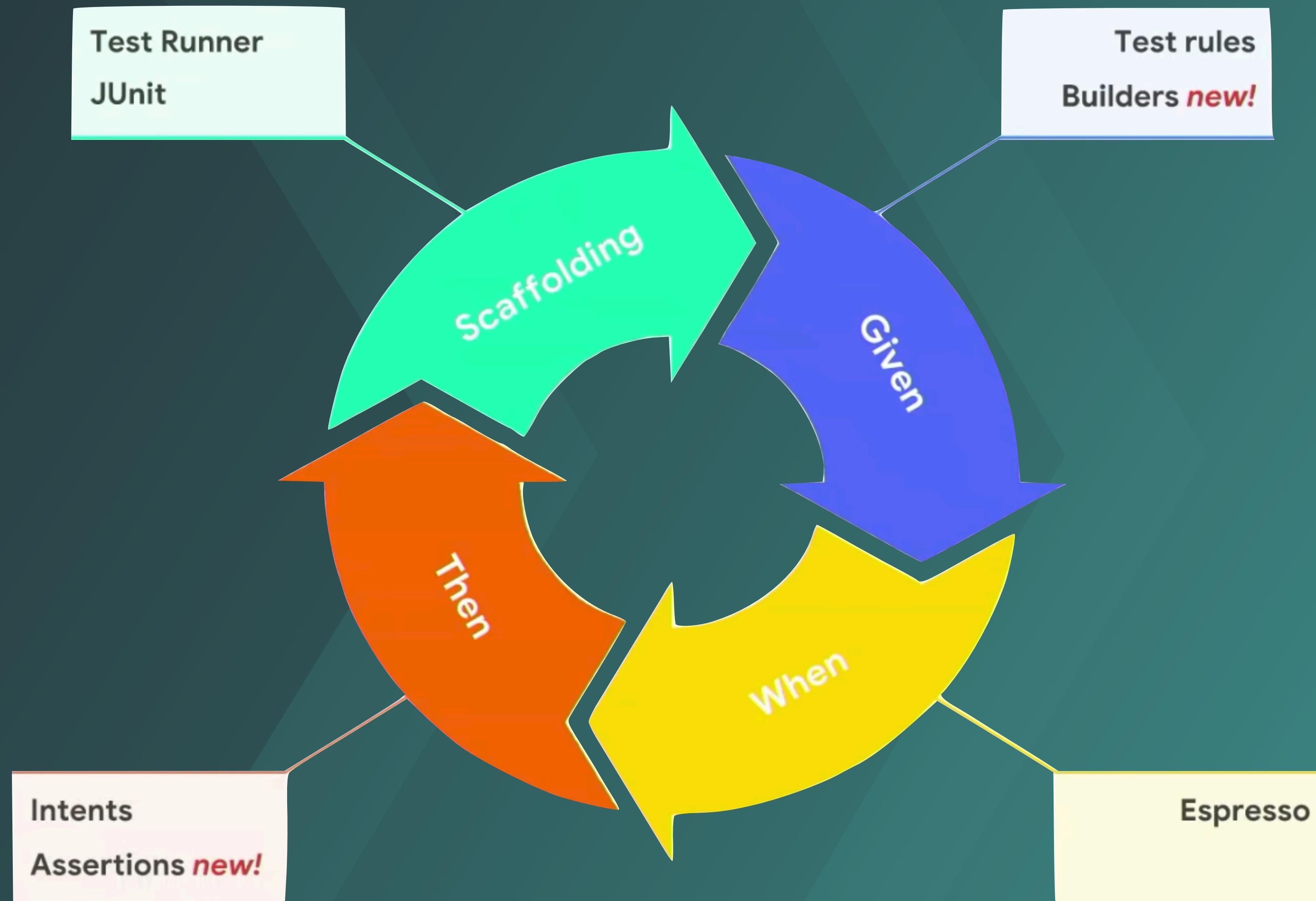
```
@RunWith(AndroidJUnit4::class)
class SimpleUnifiedTest {
    @Test
    fun testButtonClickSendsIntent() {
        onView(withId(R.id.fab)).perform(click())
    }
}
```





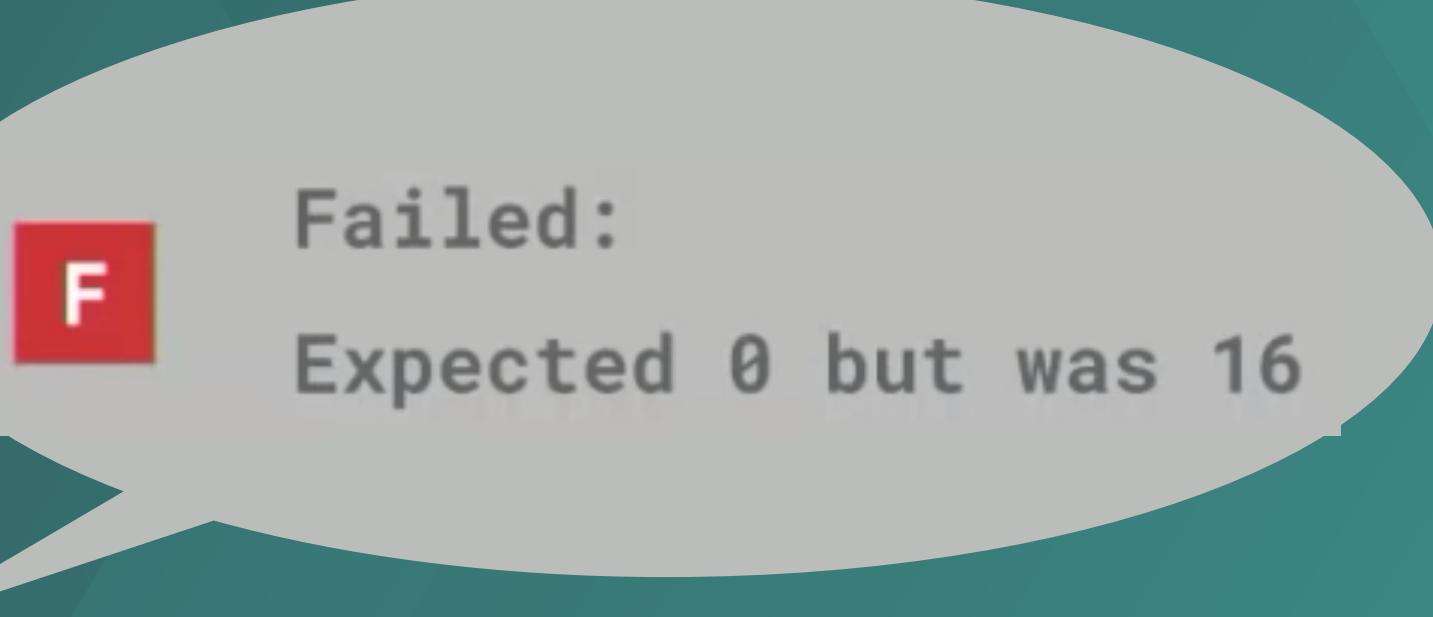
//THEN

```
@RunWith(AndroidJUnit4::class)
class SimpleUnifiedTest {
    @Test
    fun testButtonClickSendsIntent() {
        onView(withId(R.id.fab)).perform(click())
        intended(hasAction(equalTo("android.intent.action.EDIT")))
    }
}
```



//THEN

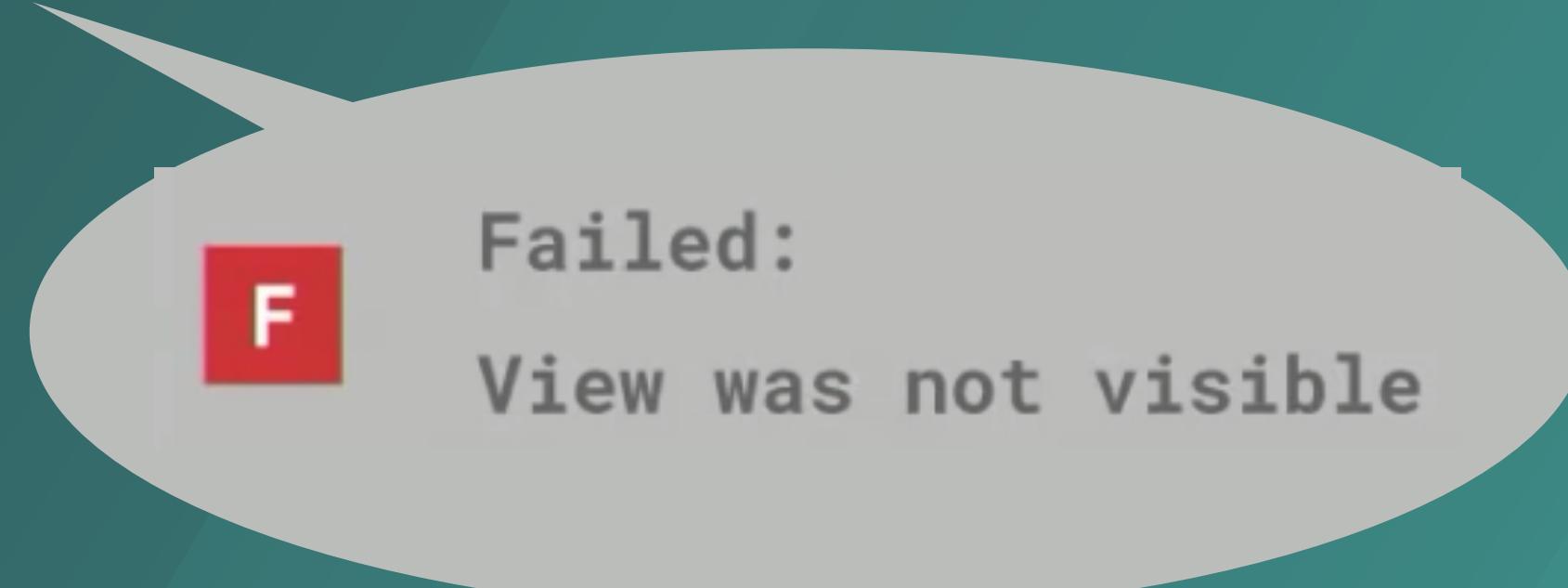
```
@RunWith(AndroidJUnit4::class)
class SimpleUnifiedTest {
    @Test
    fun testVisibleView() {
        assertEquals(view.visibility, View.VISIBLE)
    }
}
```





//THEN

```
@RunWith(AndroidJUnit4::class)
class SimpleUnifiedTest {
    @Test
    fun testVisibleView() {
        assertThat(view).isVisible()
    }
}
```



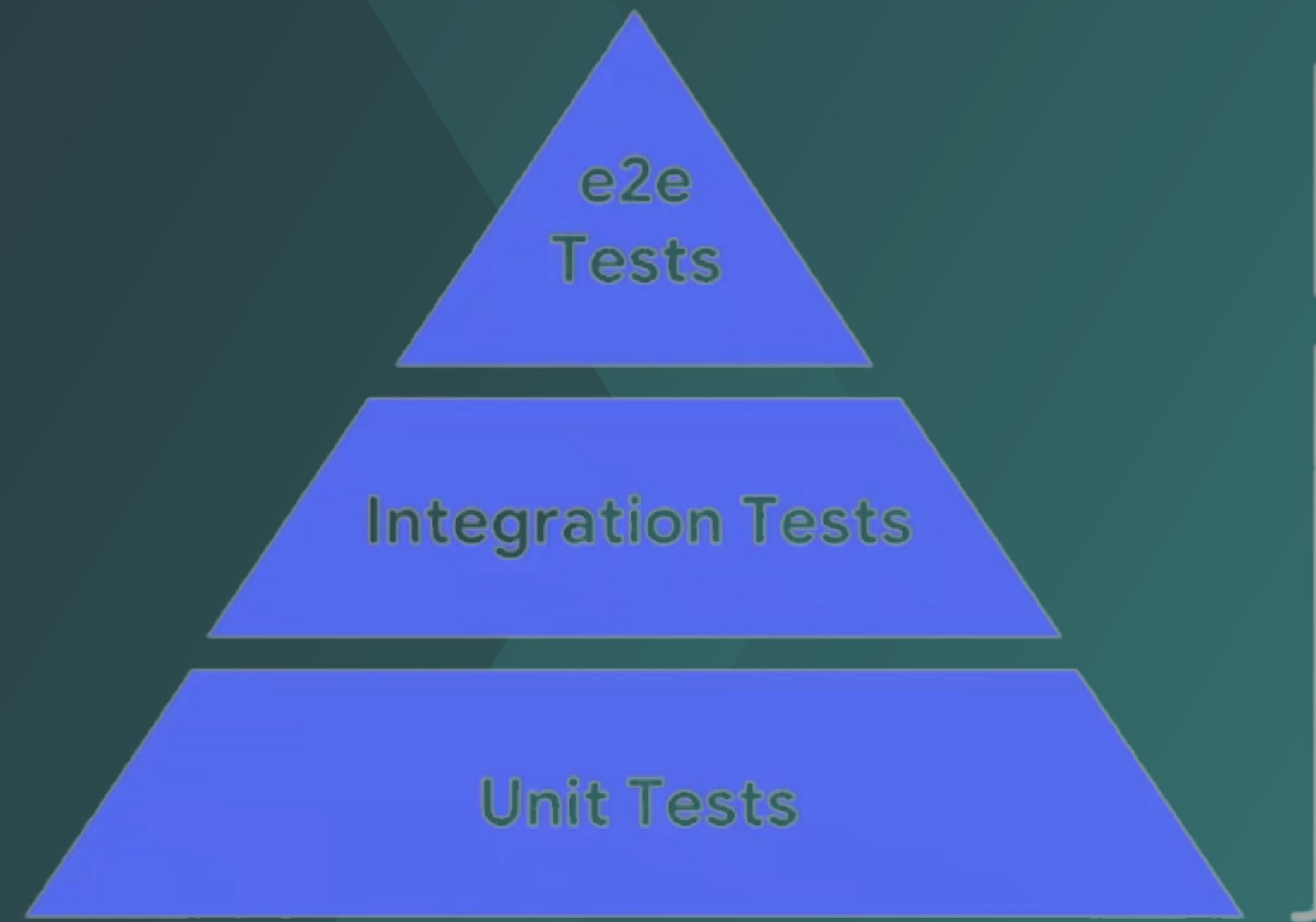


Canonical
APIs

Kotlin

Reduces
Boilerplate

Cross
Environment



Android Test
Part of Jetpack

Test execution crisis



Project Nitrogen

A single entry point for tests





DO WHAT YOU CAN'T

6.1M views



434K



7K



Share



Save



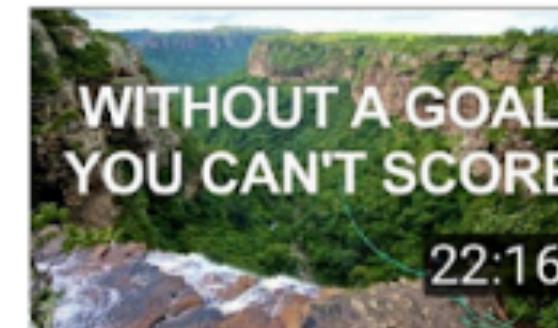
Add to



CaseyNeistat
7M subscribers



SUBSCRIBE



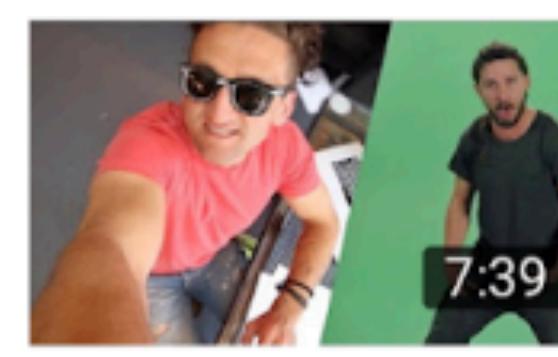
DO MORE

CaseyNeistat
2.1M views



THE \$21,000 FIRST CLASS
AIRPLANE SEAT

CaseyNeistat
42M views

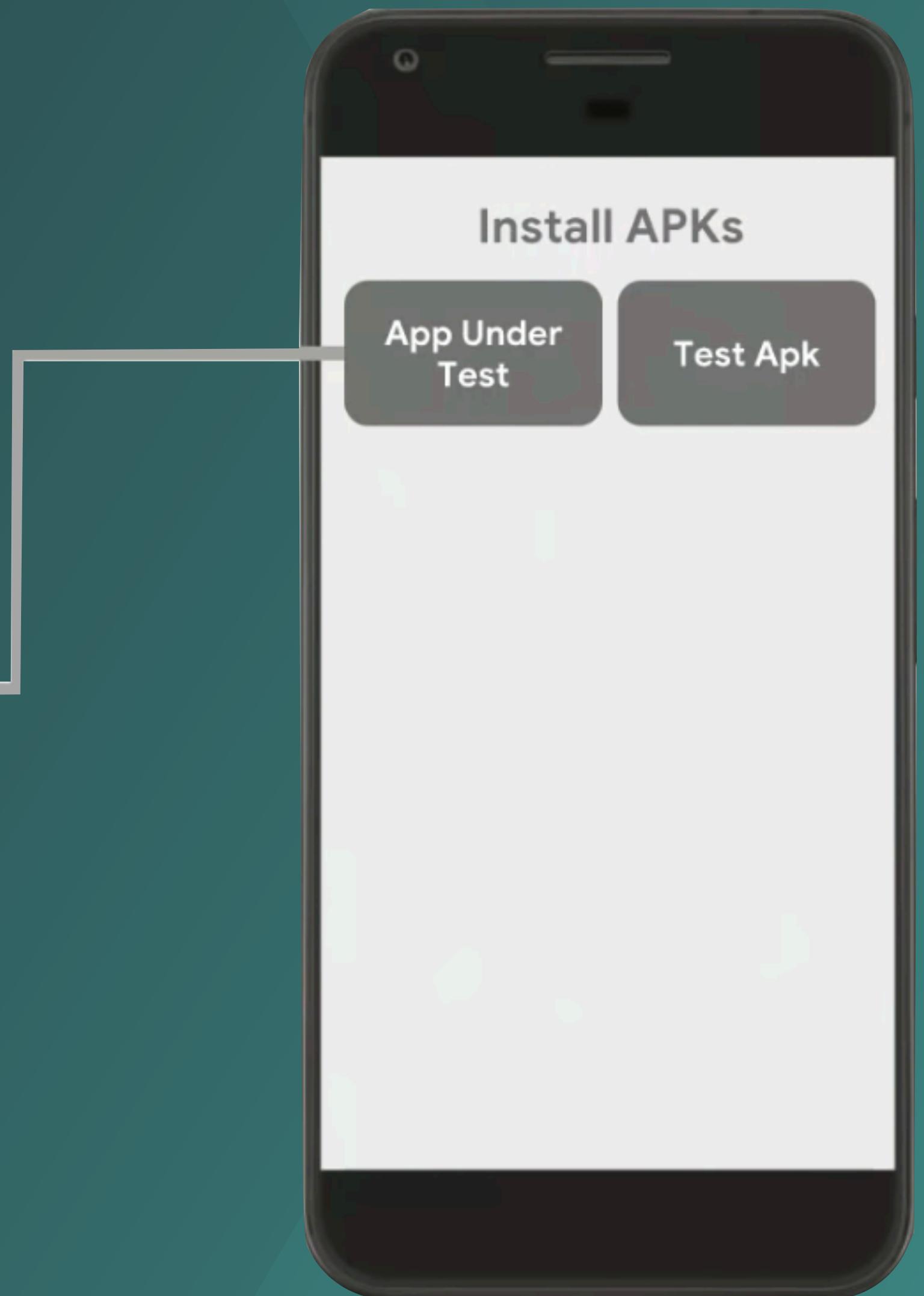


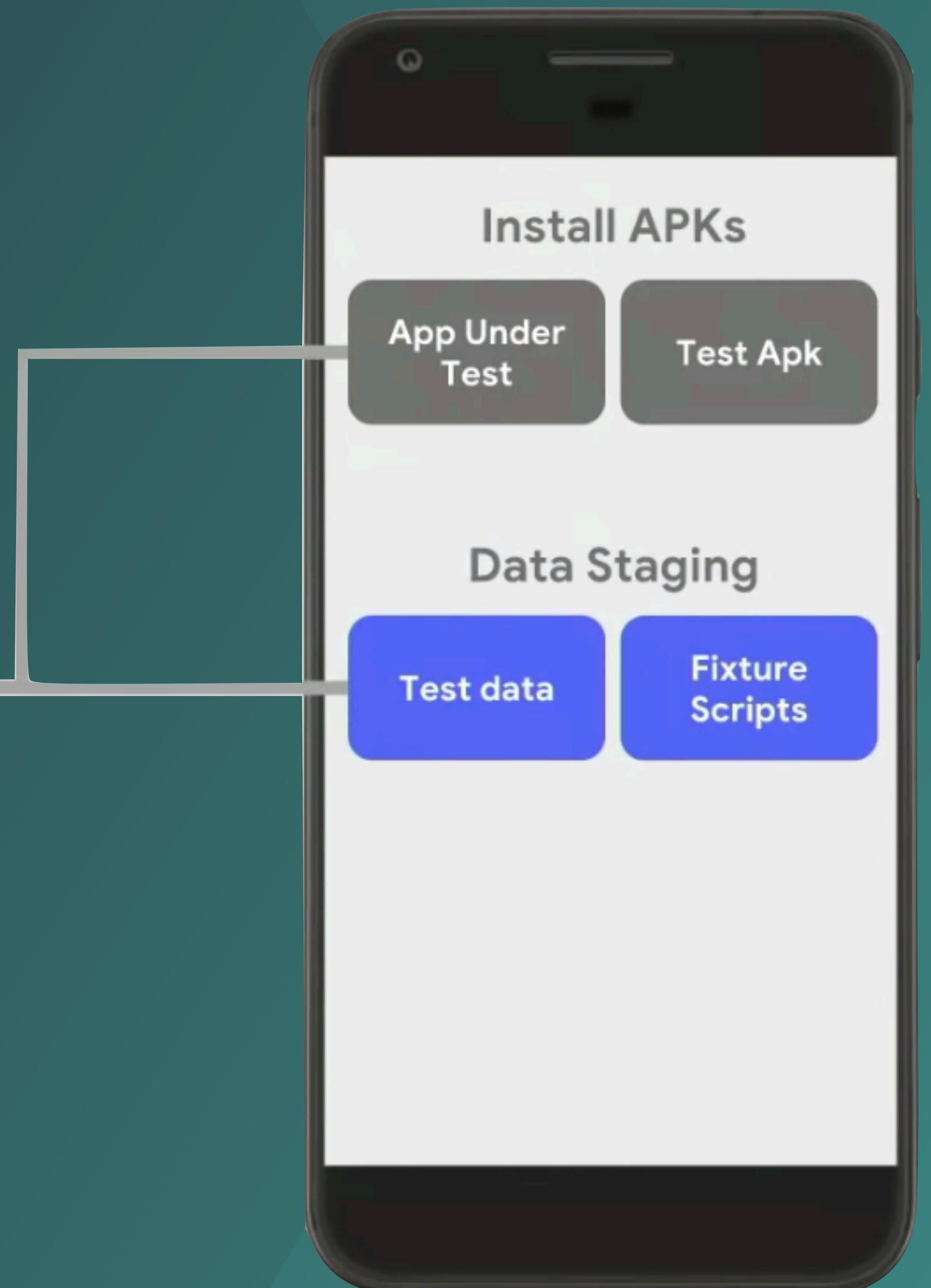
FOLLOW YOUR DREAMS

CaseyNeistat
1.9M views

The diagram consists of a large grey circle with a dark grey center. A blue arrow points from the bottom left towards the center, containing the word "Setup". A white rectangular callout box is positioned to the left of the blue arrow, listing three items: "Environment", "Test Artifacts", and "Fixtures".

- Environment
- Test Artifacts
- Fixtures







Running in the Real World



Sprint Running



Midd-Range Distance Running



Long Distance Running



**STAY
HARD**

Track Running



Road Running



Cross Country Running



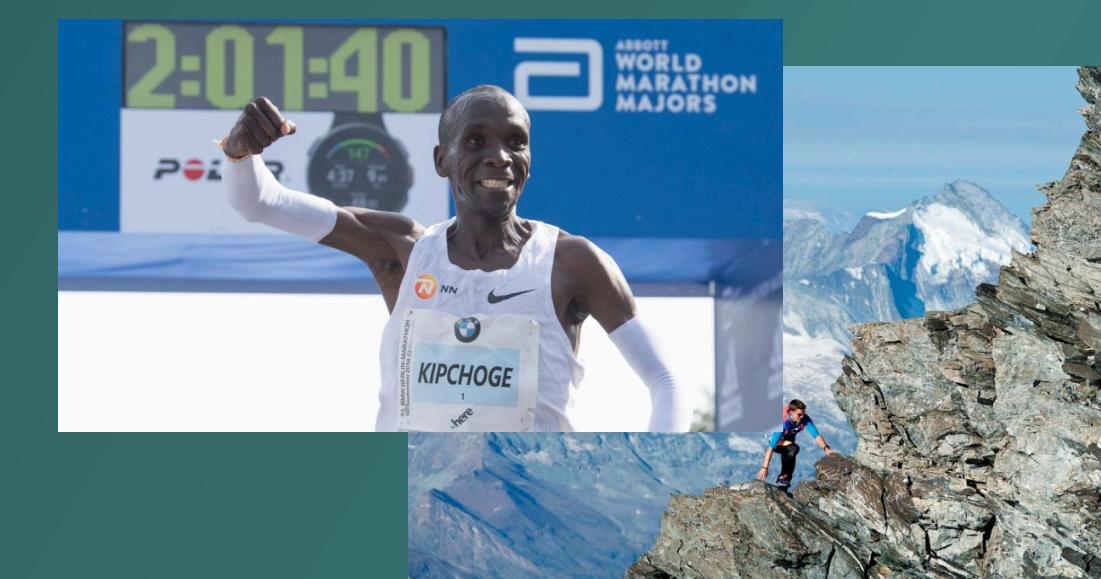
Mountain Running

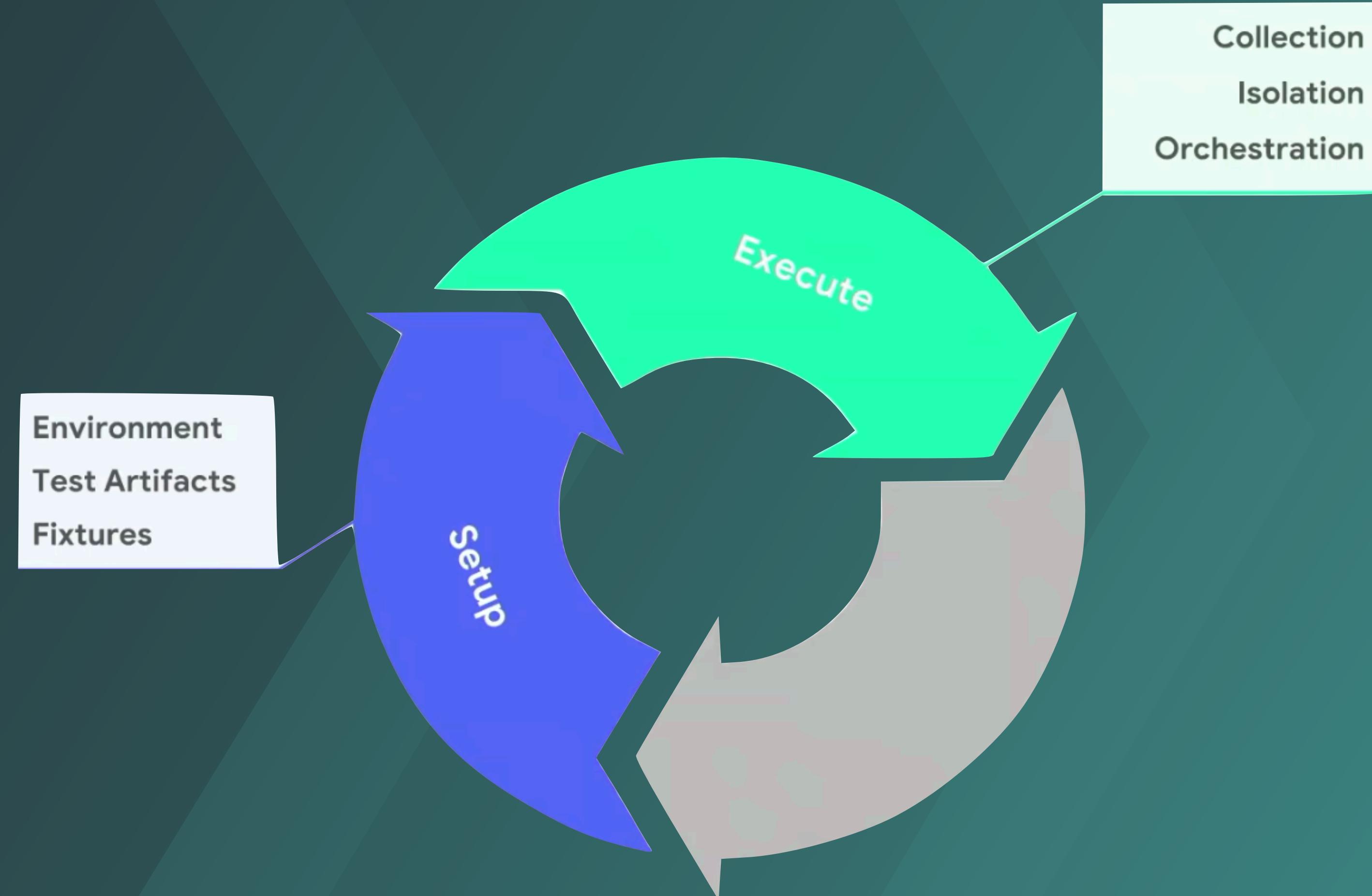


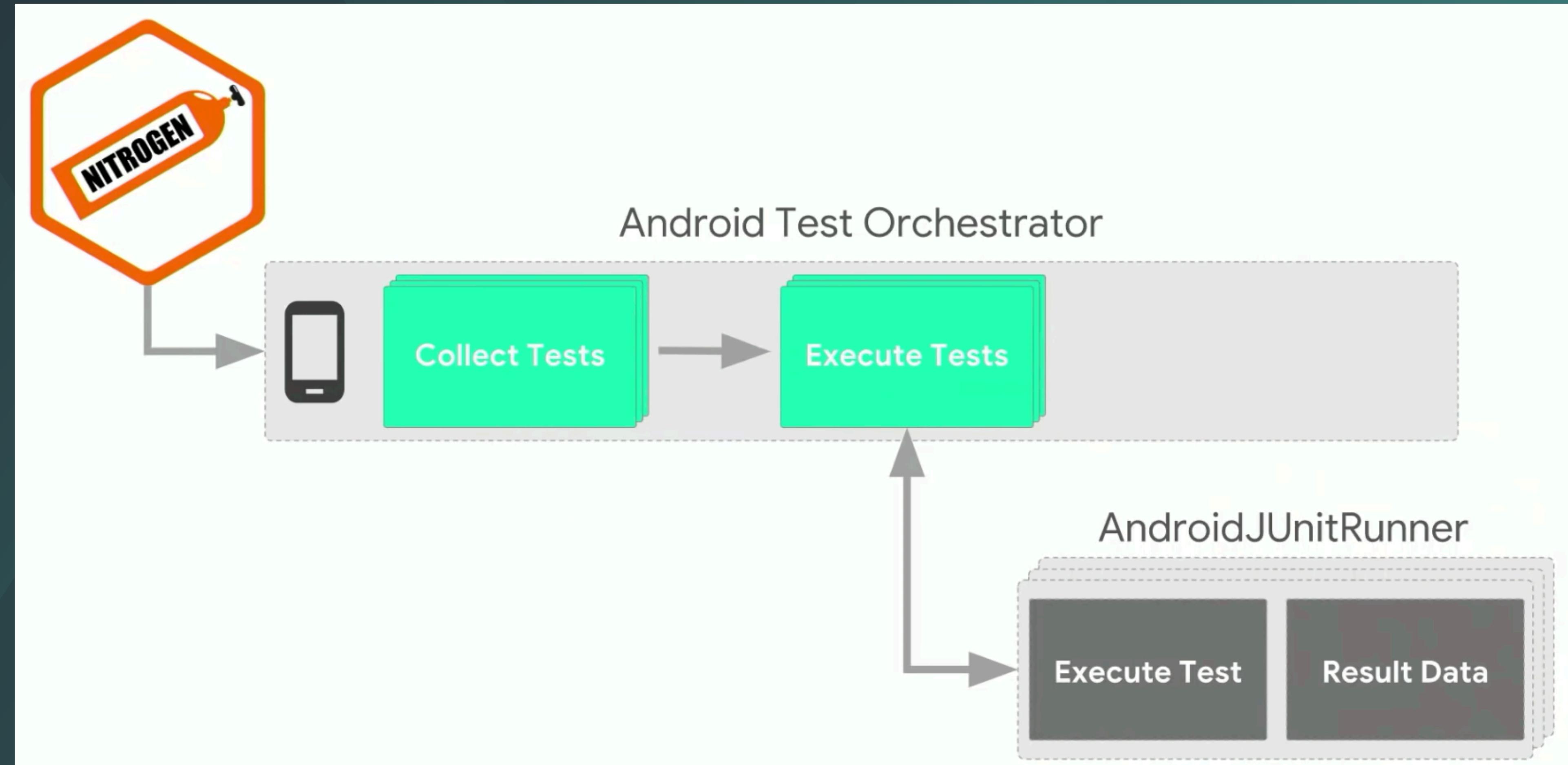
jUnit Test

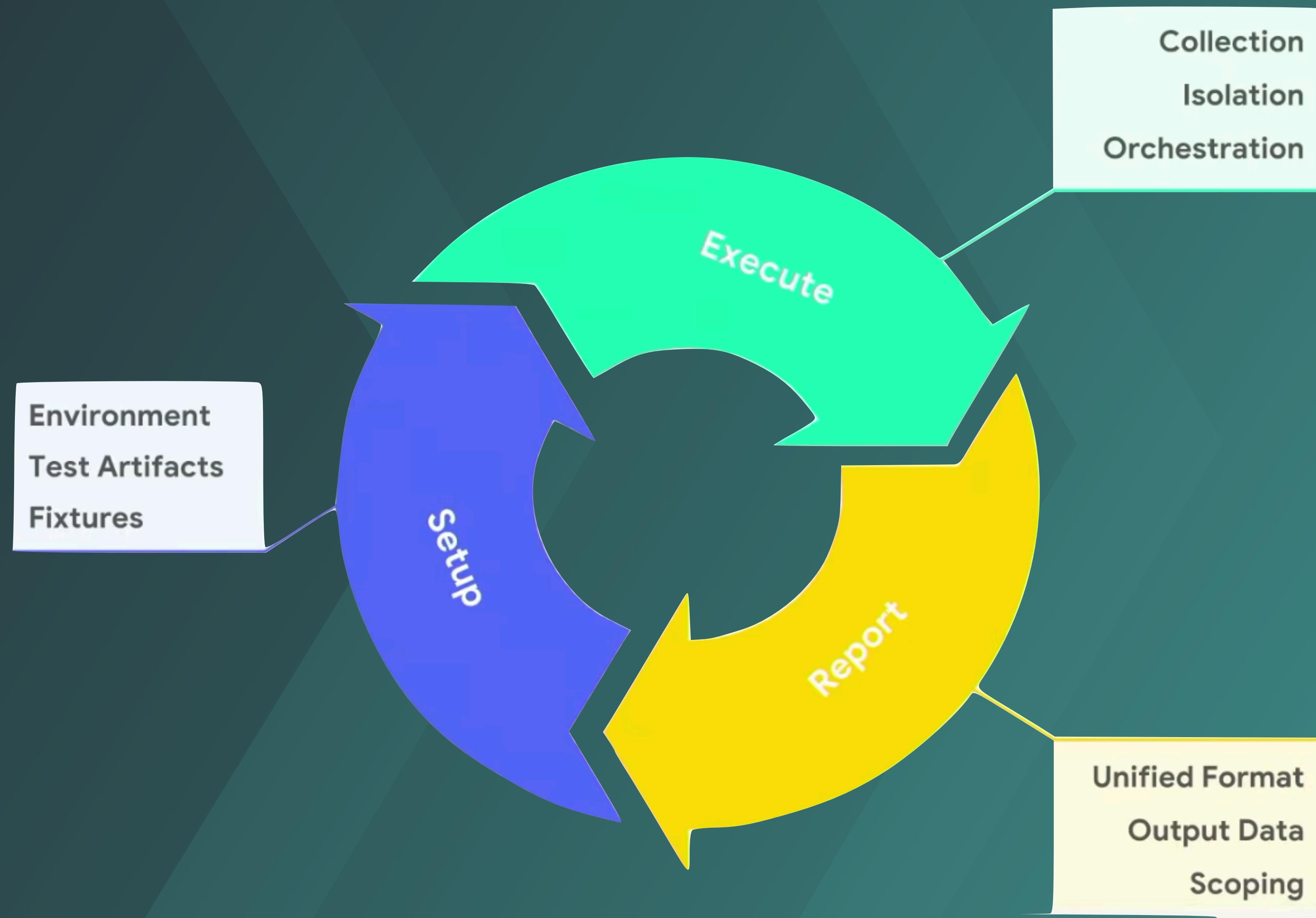


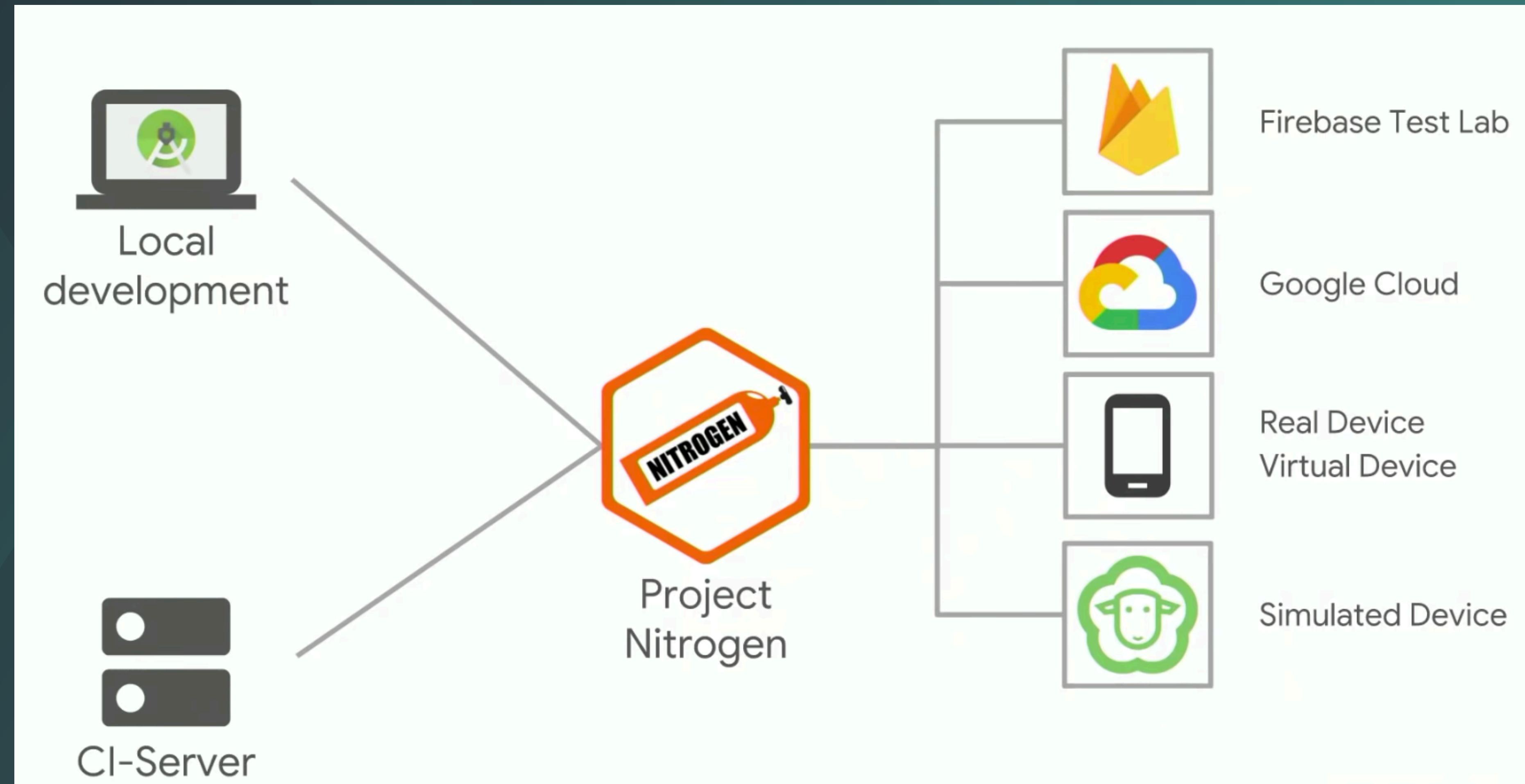
TestSuite



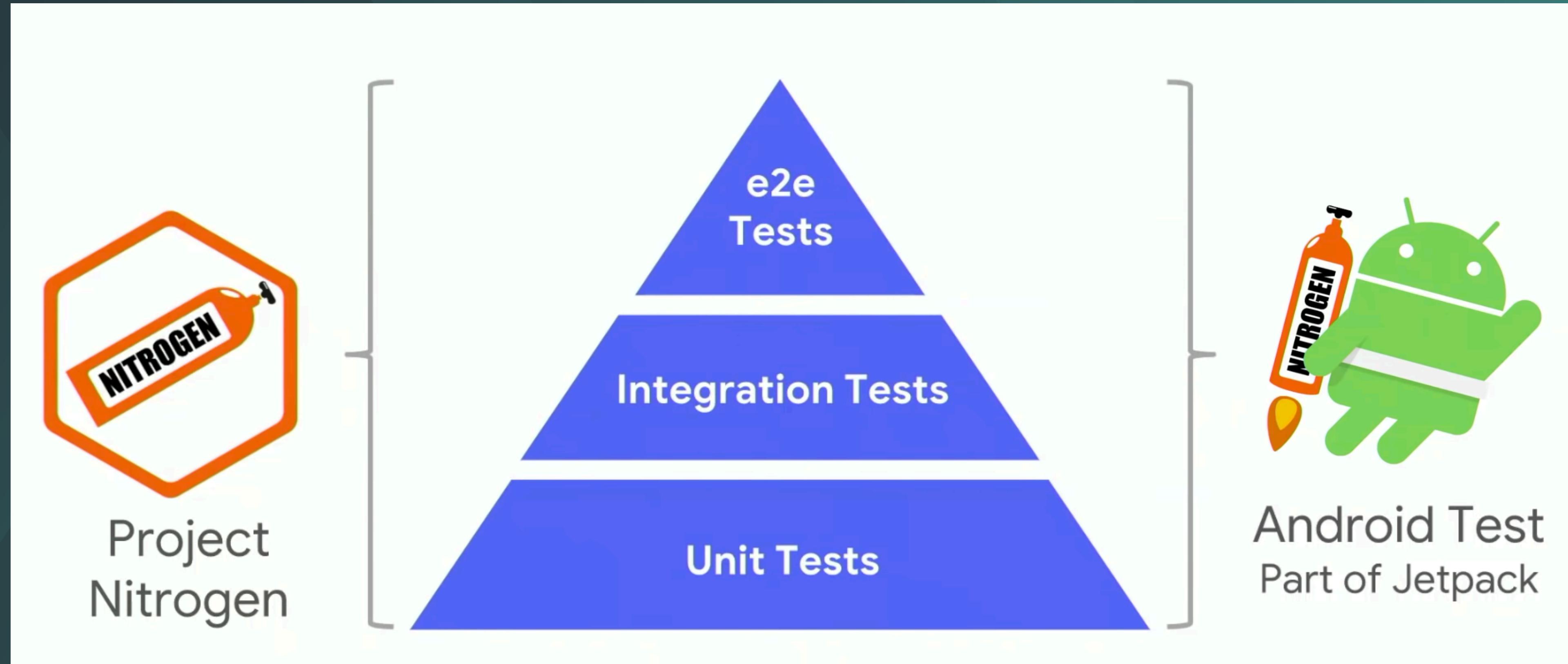








DEMO



Practical Exam

- **IMPORTANT!** Obey the exam scheduler.
- Request, by email, attendance reschedule in the secondary date if something **major** is not allowing you to attend the primary date.

Software Requirements

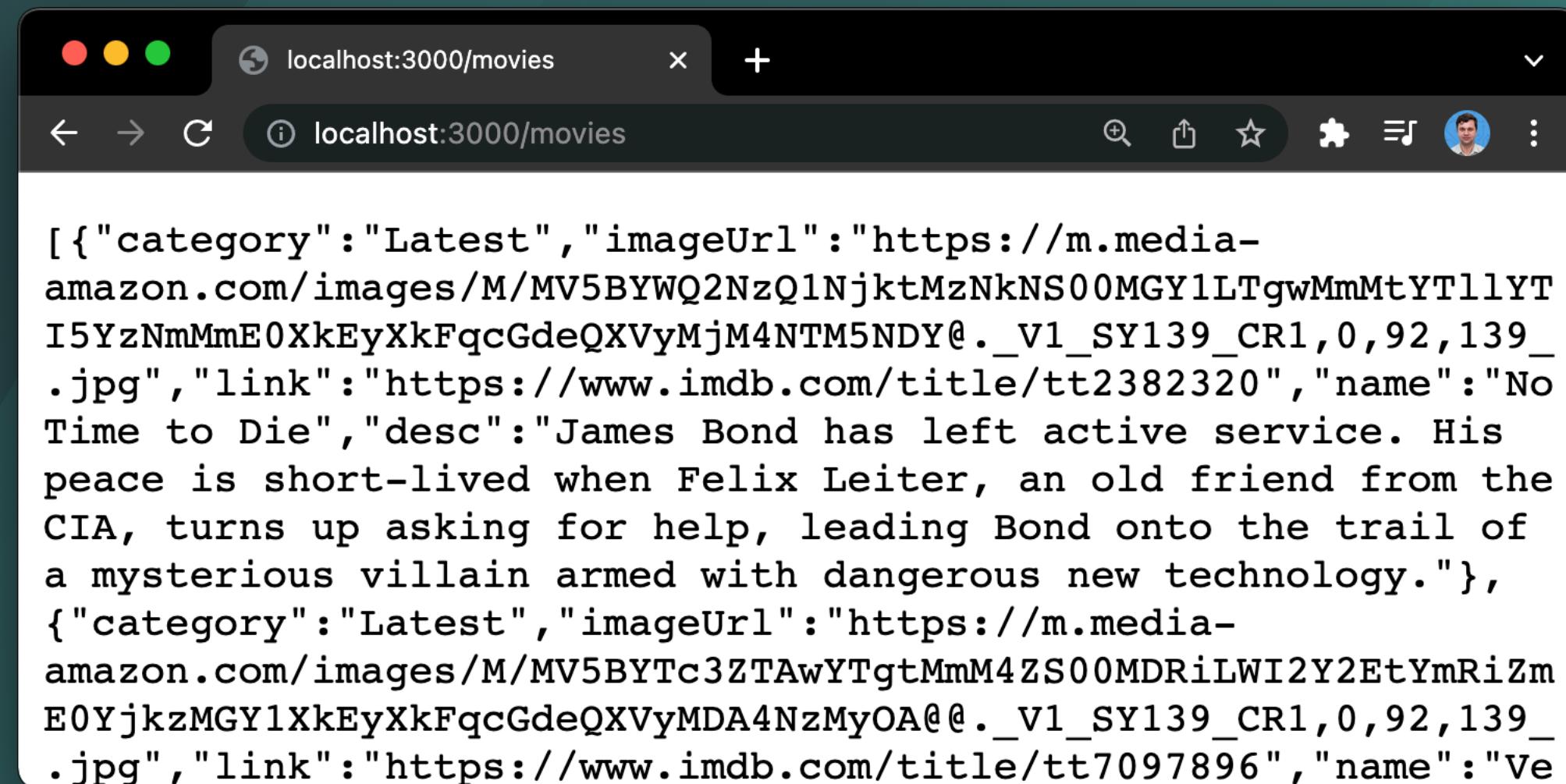
- Ensure that you have installed on your machine, the latest versions for:
 - Git
 - nodeJs
 - NPM
- Are able to share your camera and desktop.

Execute the Server

- Ensure that you are able to execute the server from:
 - <https://github.com/dancojocar/MA/tree/master/lectures/14/server>
- Steps to execute the server:
 1. Update the dependencies: **npm install**
 2. Start the server: **npm start**
- The above two commands are executed under the **server** directory!
- **Note:** You should be running the latest version of node (v21.5.0) and npm (v10.2.4).

Verify the Server

- Using the browser of your choice go to:
 - <http://localhost:3000/movies>
- If everything is working properly you should receive a JSON similar to:



A screenshot of a web browser window titled "localhost:3000/movies". The address bar also shows "localhost:3000/movies". The content area displays a JSON array with two movie entries. The first movie is "No Time to Die" with a category of "Latest", an image URL, a link to the IMDB page, a name, a description, and a release date of CR1,0,92,139. The second movie is "Ve" with a category of "Latest", an image URL, a link to the IMDB page, a name, and a release date of CR1,0,92,139.

```
[{"category": "Latest", "imageUrl": "https://m.media-amazon.com/images/M/MV5BYQ2NzQ1NjktMzNkNS00MGY1LTgwMmMtYTl1YTl5YzNmMmE0XkEyXkFqcGdeQXVyMjM4NTM5NDY@._V1_SY139_CR1,0,92,139_.jpg", "link": "https://www.imdb.com/title/tt2382320", "name": "No Time to Die", "desc": "James Bond has left active service. His peace is short-lived when Felix Leiter, an old friend from the CIA, turns up asking for help, leading Bond onto the trail of a mysterious villain armed with dangerous new technology."}, {"category": "Latest", "imageUrl": "https://m.media-amazon.com/images/M/MV5BYTc3ZTAwYTgtMmM4ZS00MDRlWI2Y2EtYmRizmE0YjkzMGY1XkEyXkFqcGdeQXVyMDA4NzMyOA@._V1_SY139_CR1,0,92,139_.jpg", "link": "https://www.imdb.com/title/tt7097896", "name": "Ve"}]
```

Practical Exam

- Two hours to solve the requirements.
- In this time:
 - You are **NOT** allowed to:
 - Discuss with anyone!
 - Disturb others!
 - You are allowed to:
 - Use all the materials available on your local machine or on the Internet. So, ensure that you are well prepared!
 - Request clarifications only from the exam supervisor.
- If found breaking at least one of the above rules you will be denied to take the current exam and the following ones held this year!

Lecture outcomes

- Introduction to the concepts and tools for building Android tests.
- Build complex unit tests with Android dependencies that cannot be satisfied with mock objects.
- Create tests to verify that the user interface behaves correctly for user interactions within a single app or for interactions across multiple apps.
- Create a stable and reusable testing harness to run performance tests.

