

Lecture #13

Testing and Exam

Discussions

Mobile Applications 2018-2019

Types of Software Testing

System Testing

Smoke testing

Ad-hoc testing

Security Testing

Acceptance Testing

Sanity Testing

Glass box Testing

Stress Testing

Accessibility Testing

Static Testing

Agile Testing

Stability Testing

API Testing

Scalability Testing

Automated testing

Gorilla Testing

Volume testing

Branch Testing

Vulnerability Testing

Browser compatibility Testing

Component Testing

Dynamic Testing

End-to-end Testing

Compatibility testing

Condition Coverage Testing

Decision Coverage Testing

Exploratory Testing

Happy path testing

Integration Testing

Interface Testing

Internationalization Testing

Functional Testing

Fuzz Testing

GUI (Graphical User Interface) testing



System Integration Testing

Usability testing

Unit testing

User Acceptance testing

Performance Testing

All Pairs testing

Regression Testing

Beta Testing

Retesting

Black Box testing

Risk-based Testing

Load Testing

Backward Compatibility Testing

Localization Testing

Boundary Value Testing

Negative Testing

Big Bang Integration testing

Non-functional testing

Bottom up Integration testing

Pair Testing

Keyword-driven Testing

White box Testing

Types of Software Testing

System Testing

Smoke testing

Ad-hoc testing

Security Testing

Acceptance Testing

Sanity Testing

Glass box Testing

Stress Testing

Accessibility Testing

Static Testing

Agile Testing

Stability Testing

API Testing

Scalability Testing

Automated testing

Gorilla Testing

Volume testing

Branch Testing

Vulnerability Testing

Browser compatibility Testing

Component Testing

Dynamic Testing

End-to-end Testing

Compatibility testing

Condition Coverage Testing

Decision Coverage Testing

Exploratory Testing

Happy path testing

Integration Testing

Interface Testing

Internationalization Testing

Functional Testing

Fuzz Testing

GUI (Graphical User Interface) testing

<https://www.testingexcellence.com/types-of-software-testing-complete-list/>



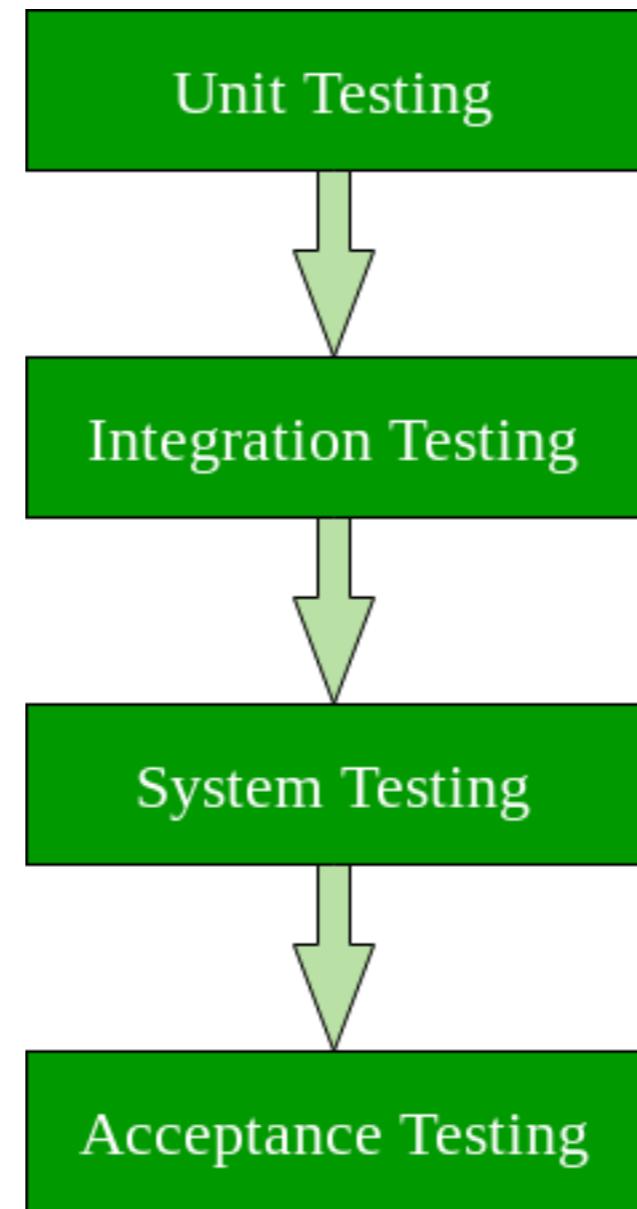
Types of Software Testing

Software Testing can be broadly classified into two types:

- Manual Testing
- Automation Testing

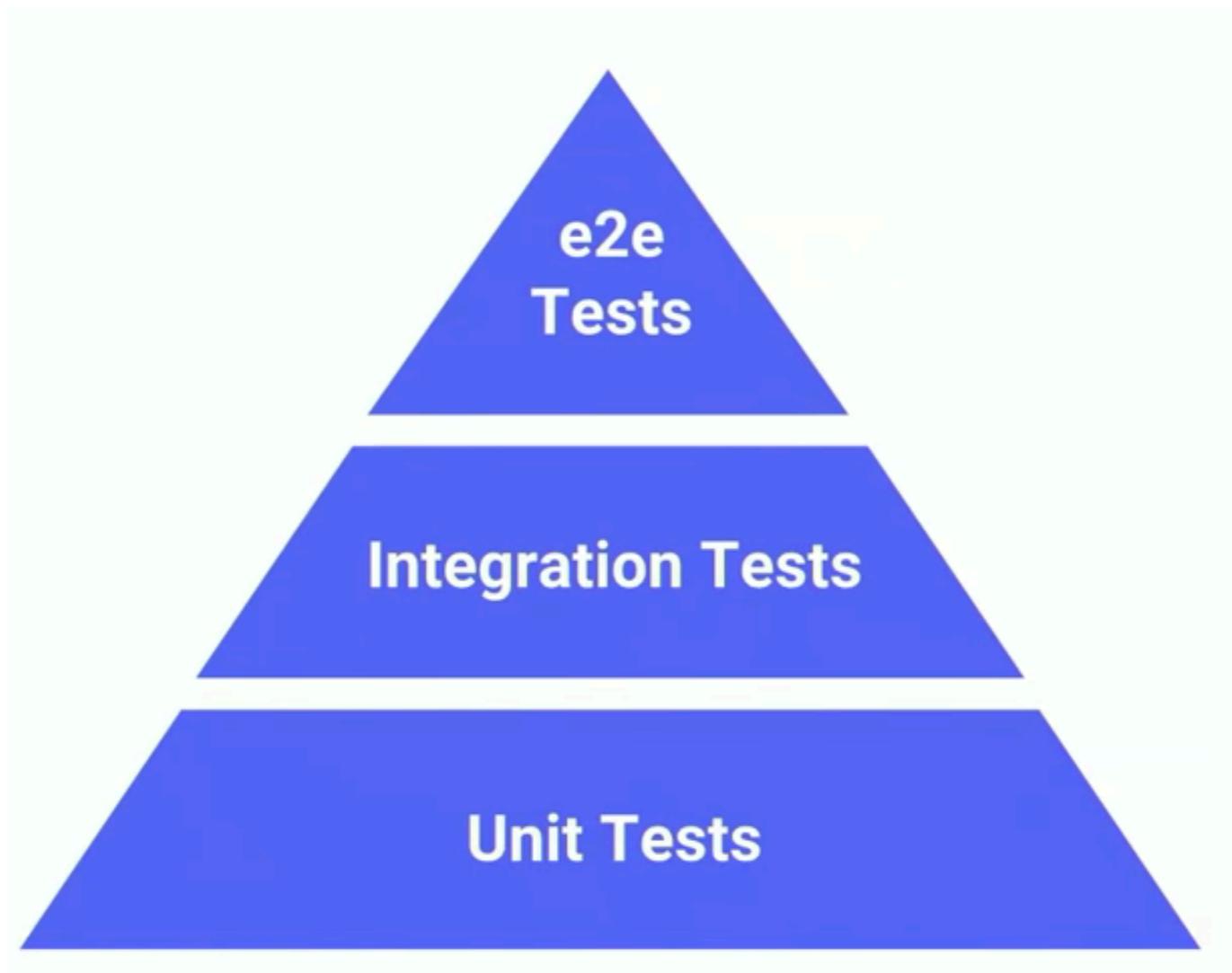


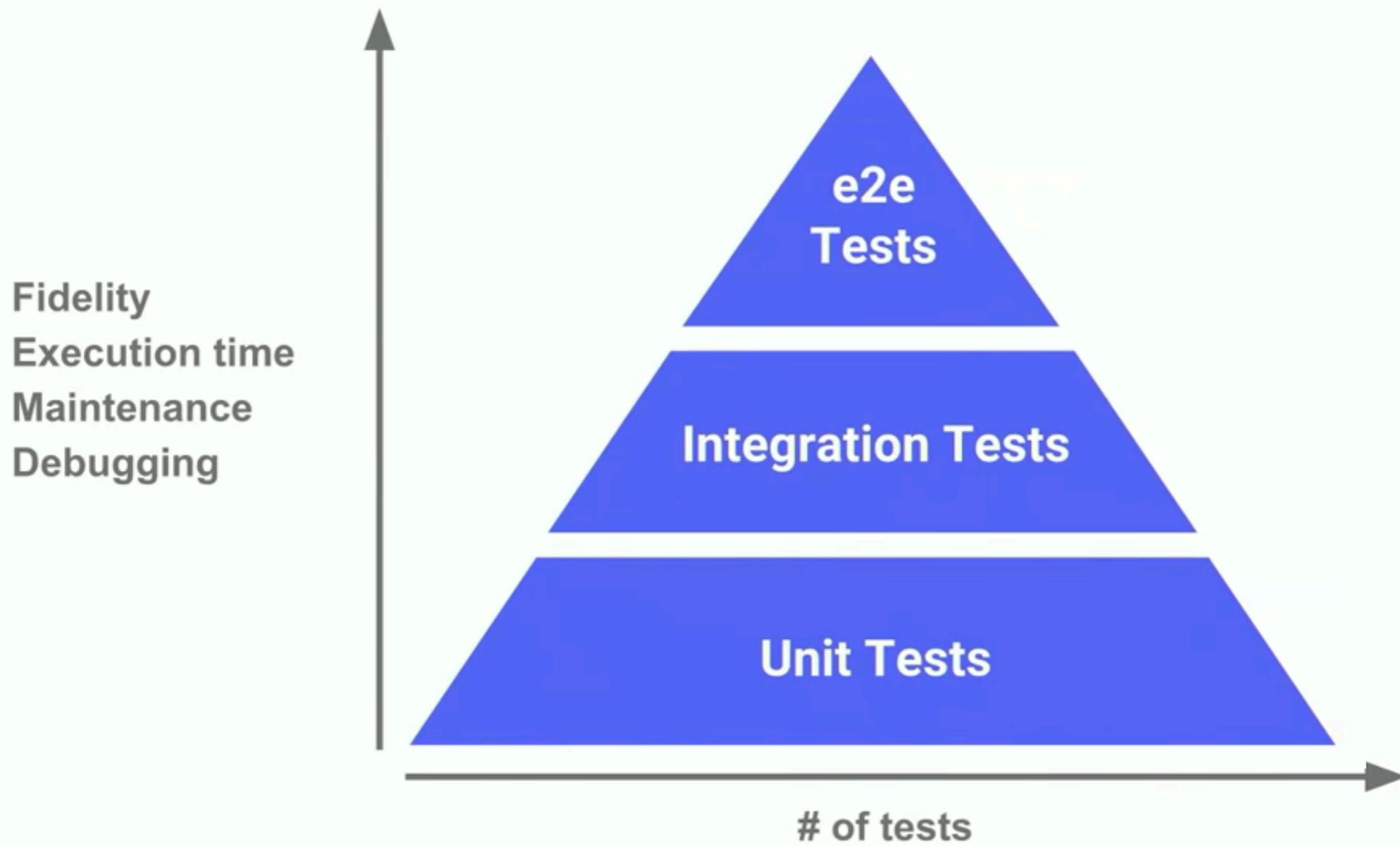
Automation Testing Levels

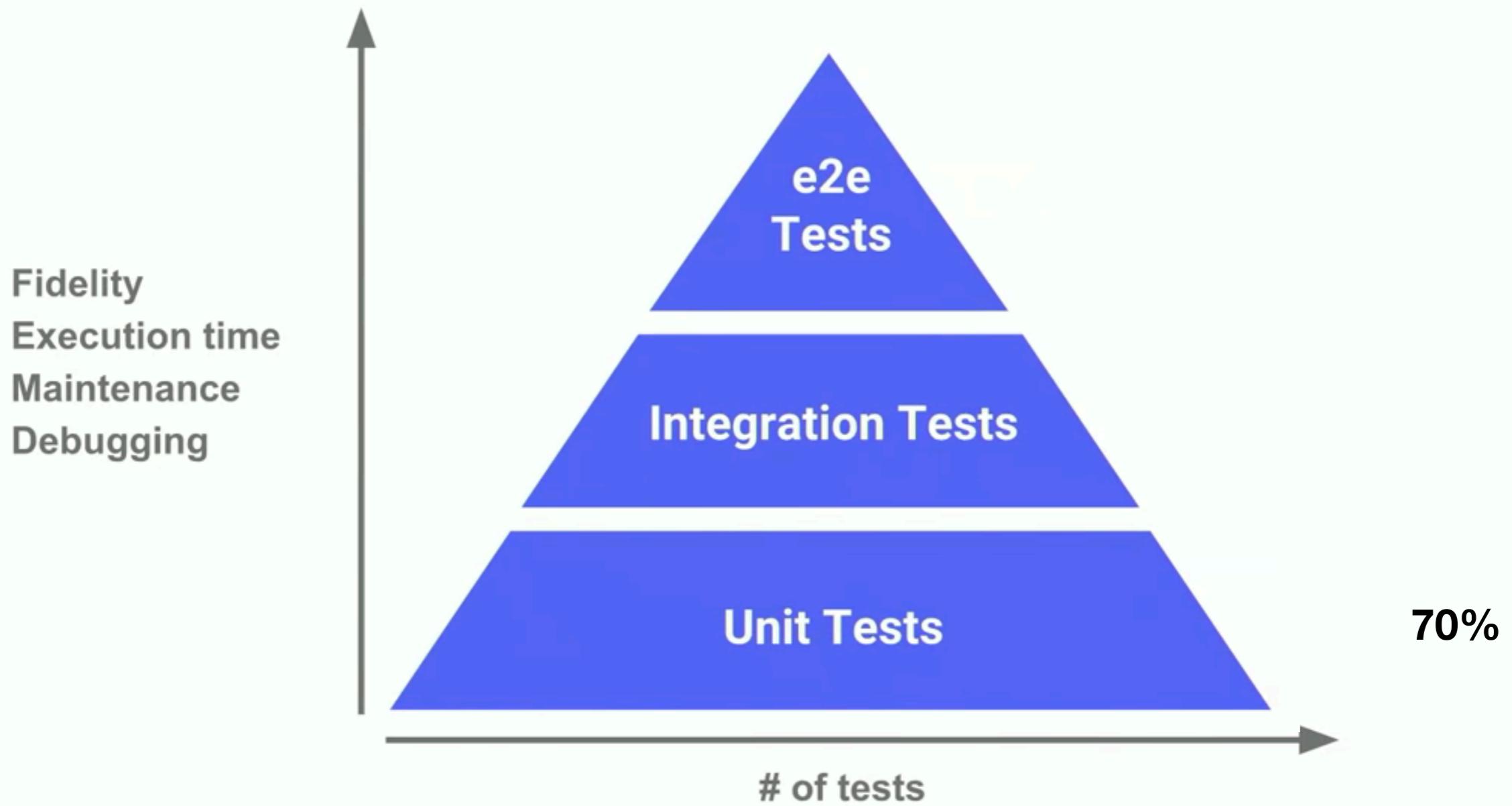


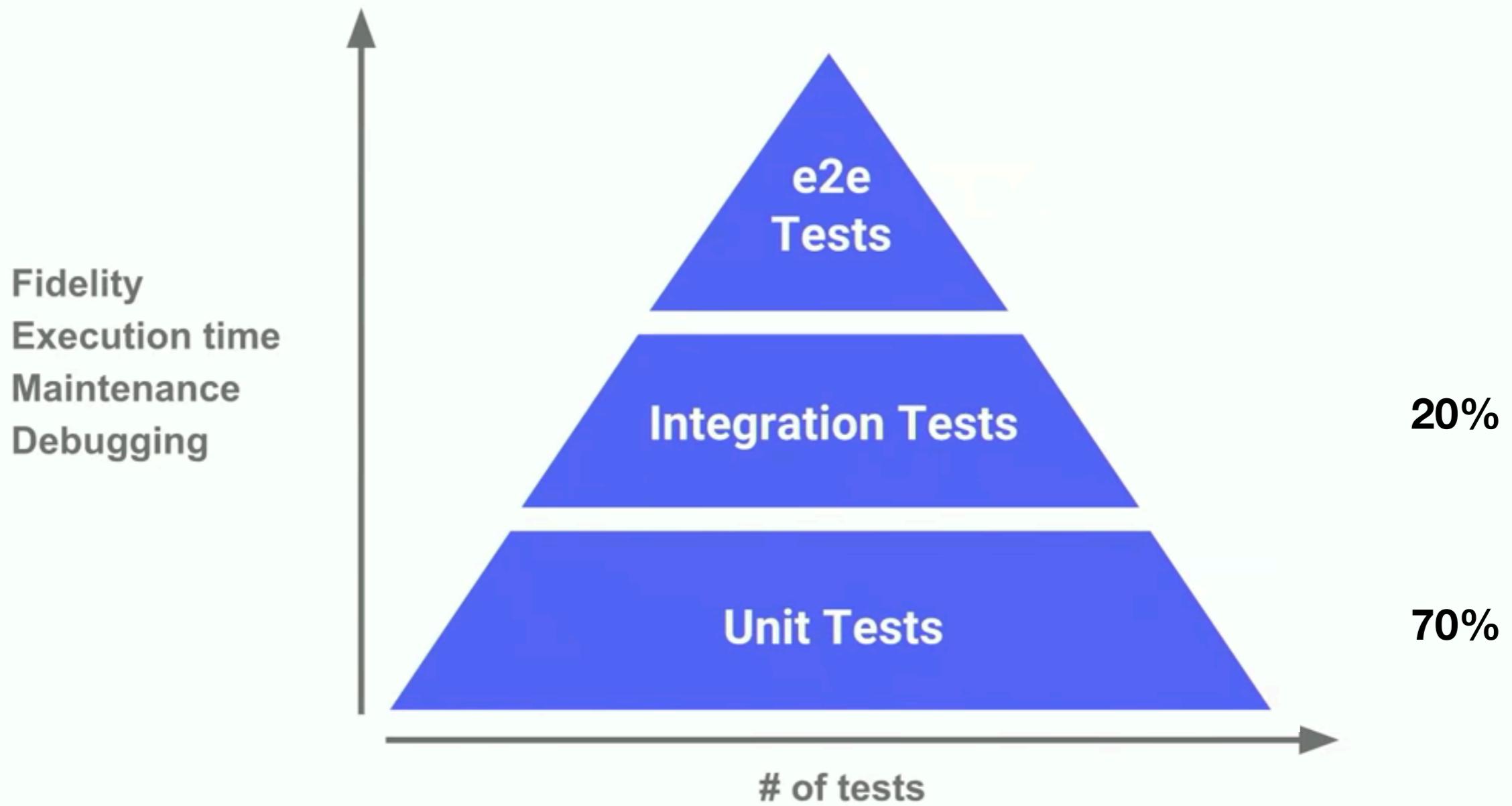
Advantages

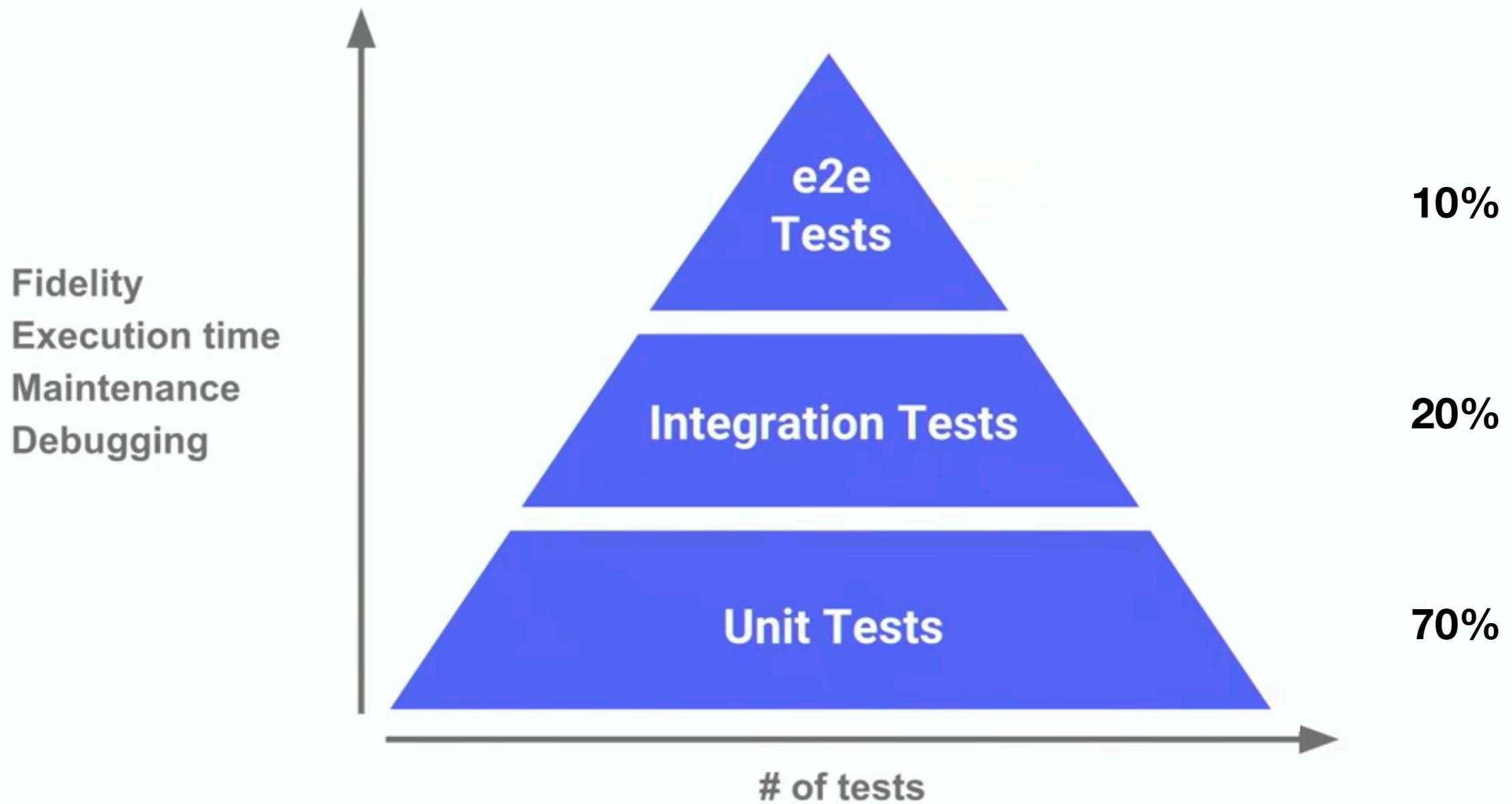
- Rapid feedback.
- Early failure detection.
- Safer code refactoring.
- Stable development velocity.

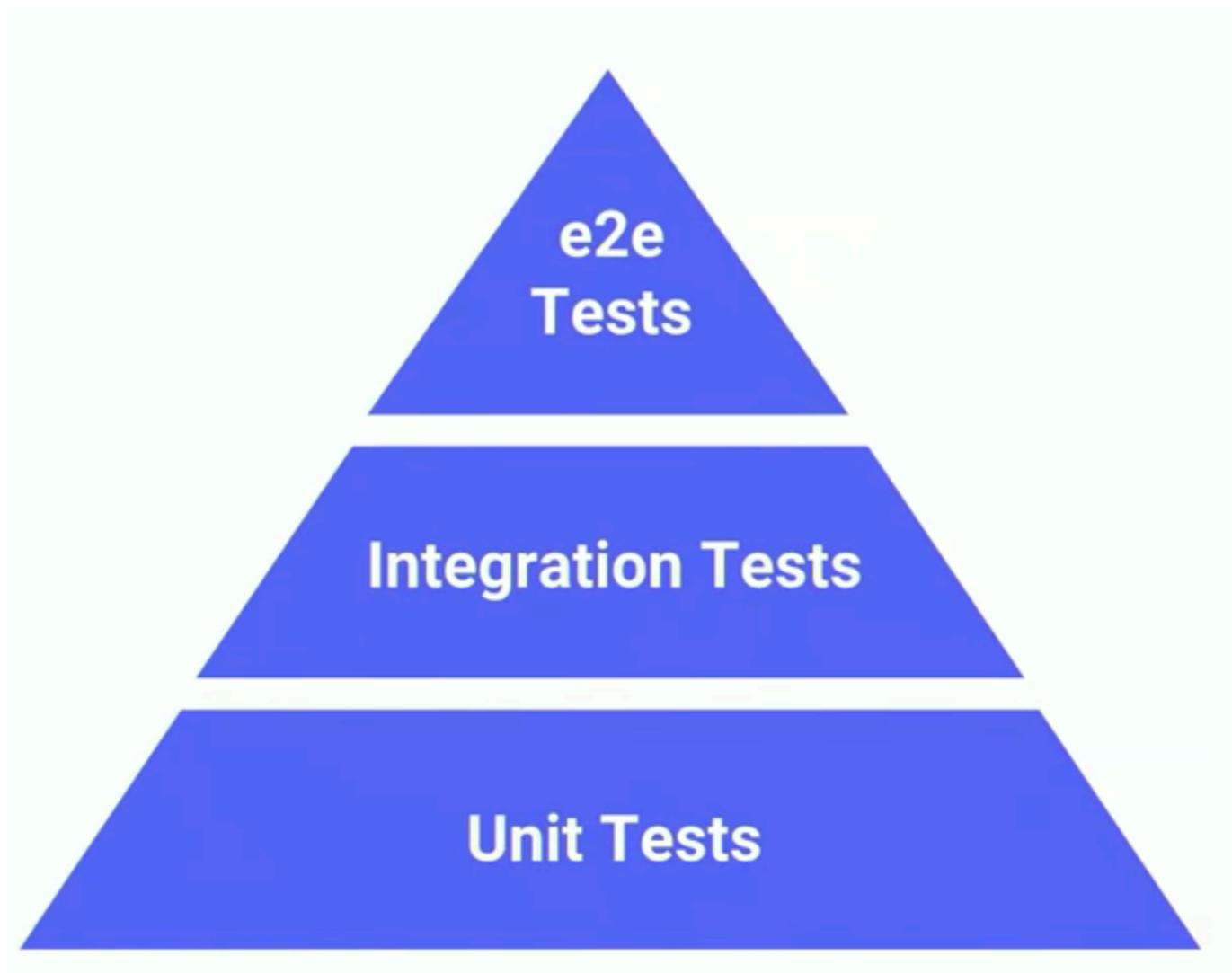


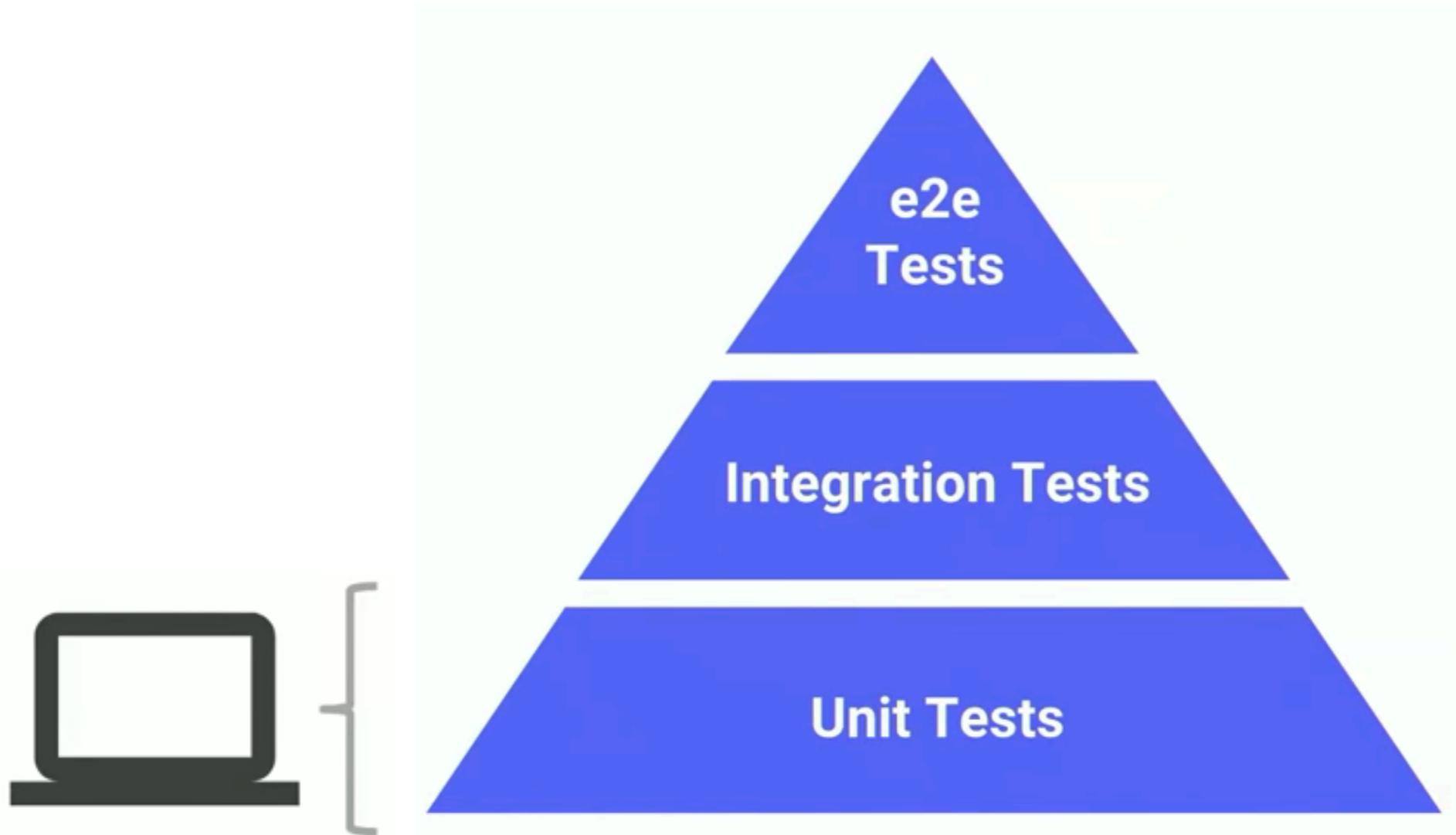


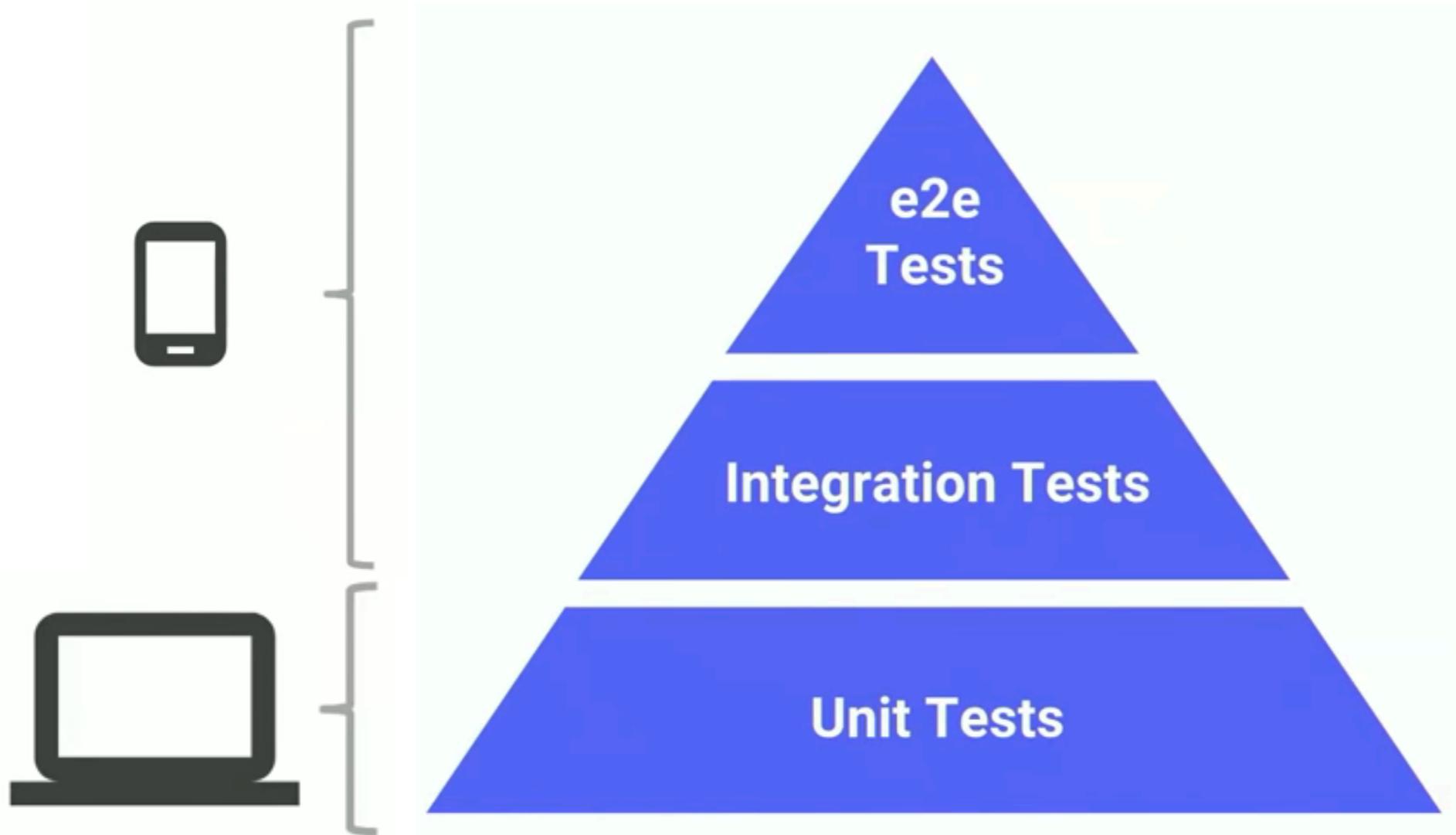


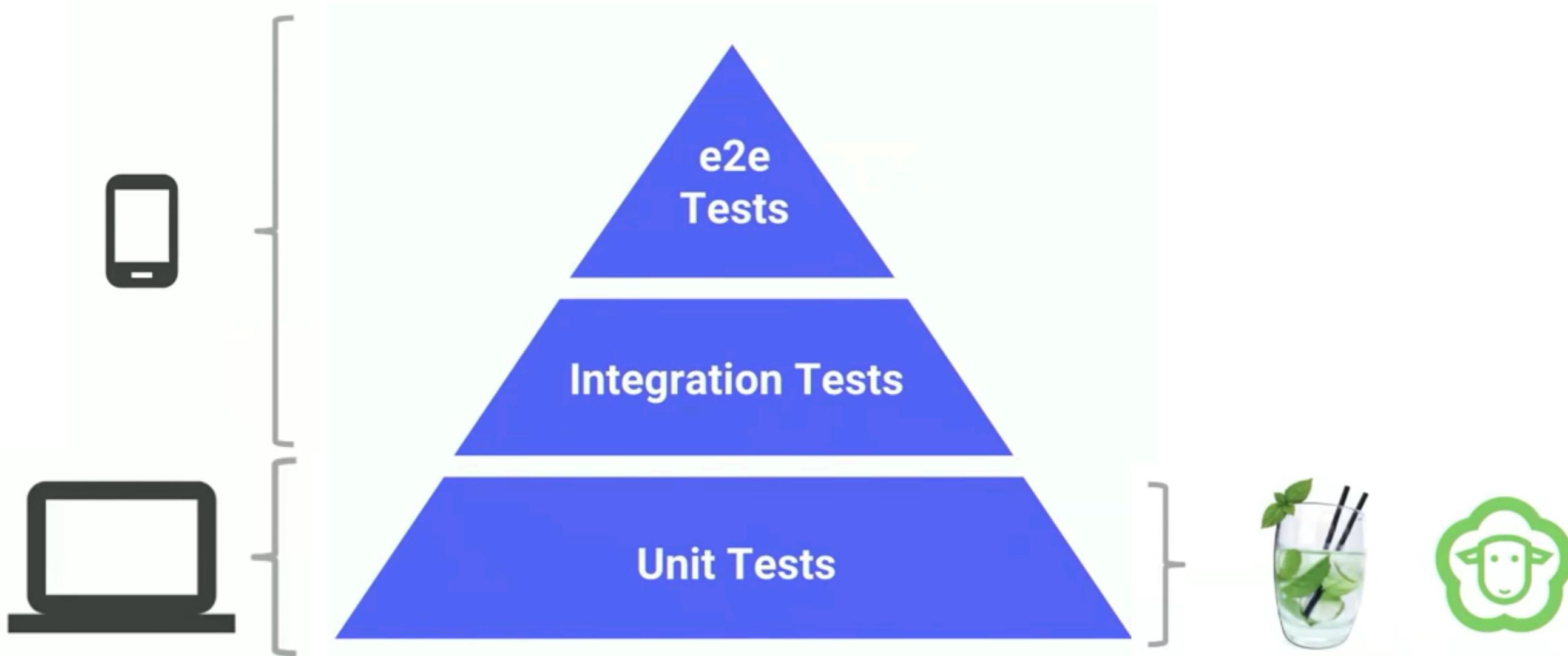


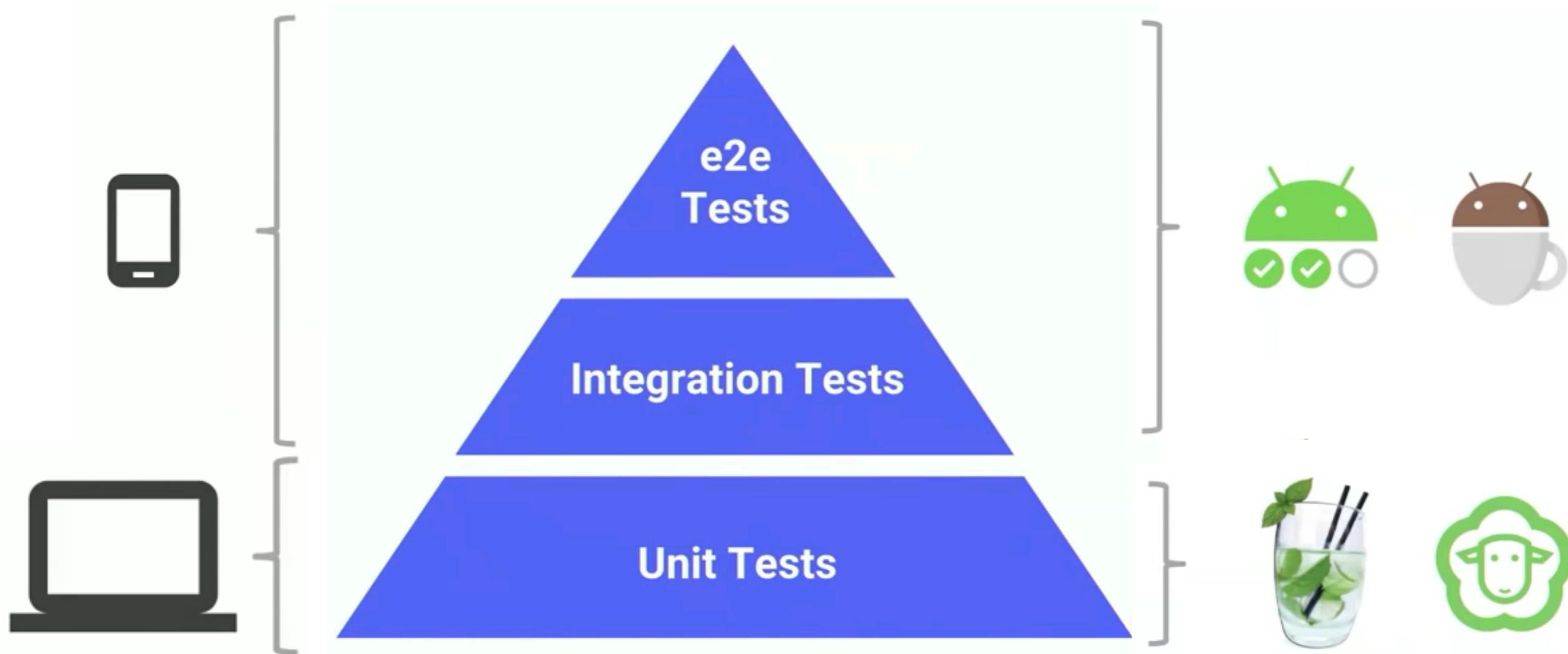












Test writing crisis

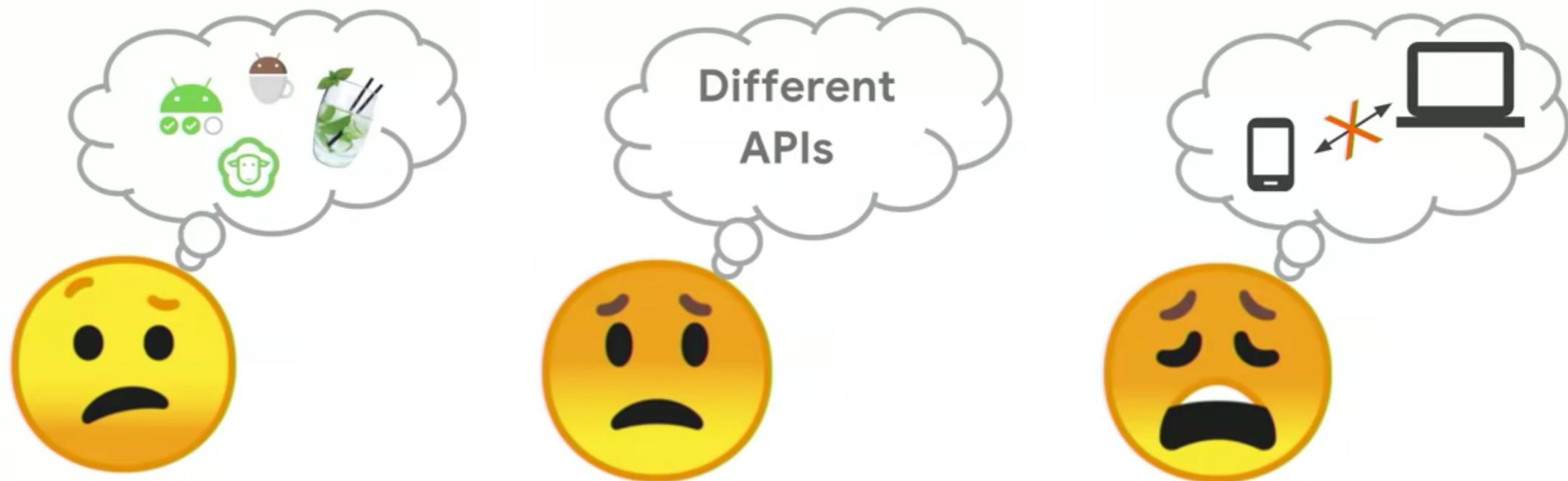
Test writing crisis



Test writing crisis



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  
    }  
    ...  
}
```

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 Structure

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

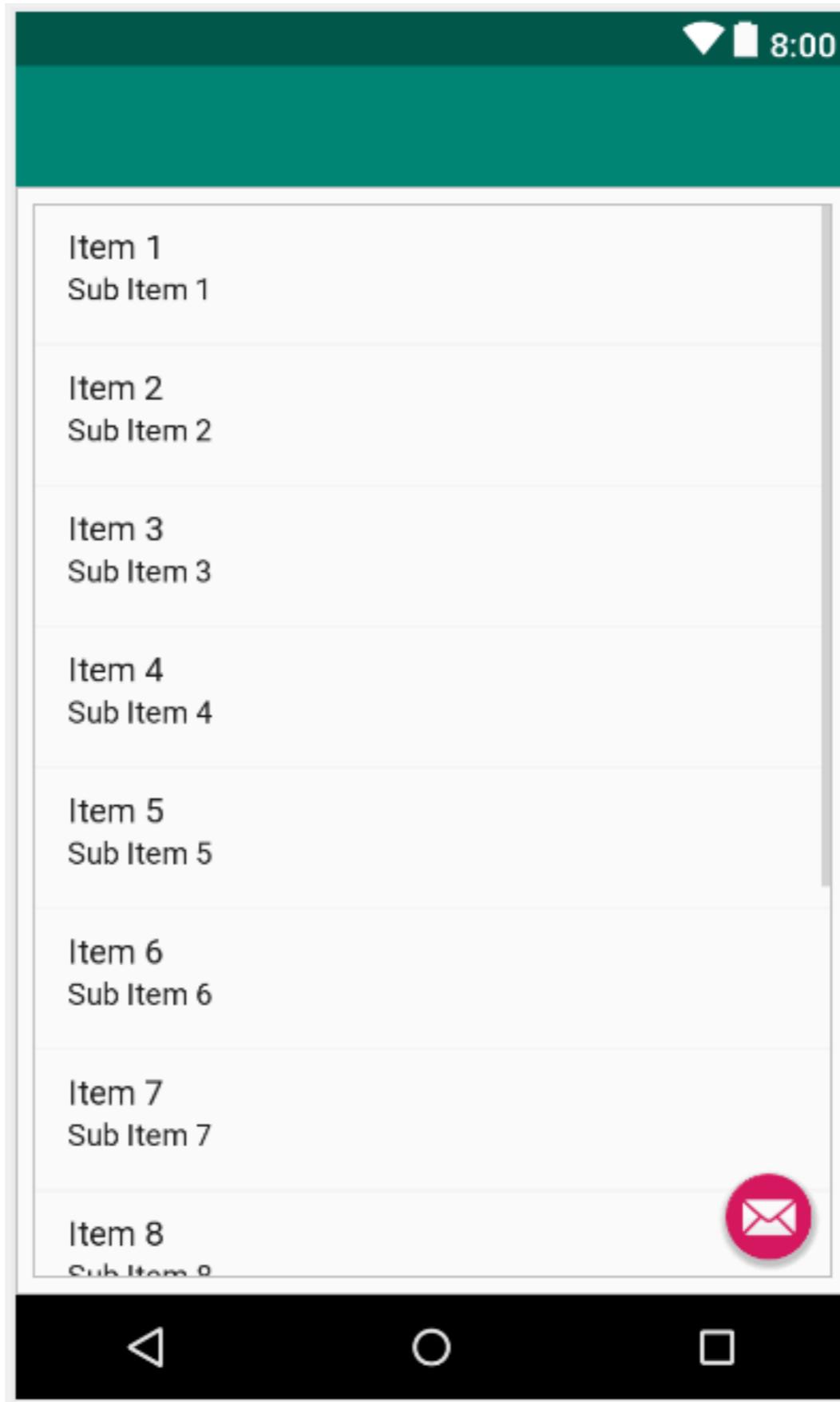
- Focus on specific behavior.
- Test behaviors independently.

Test Structure

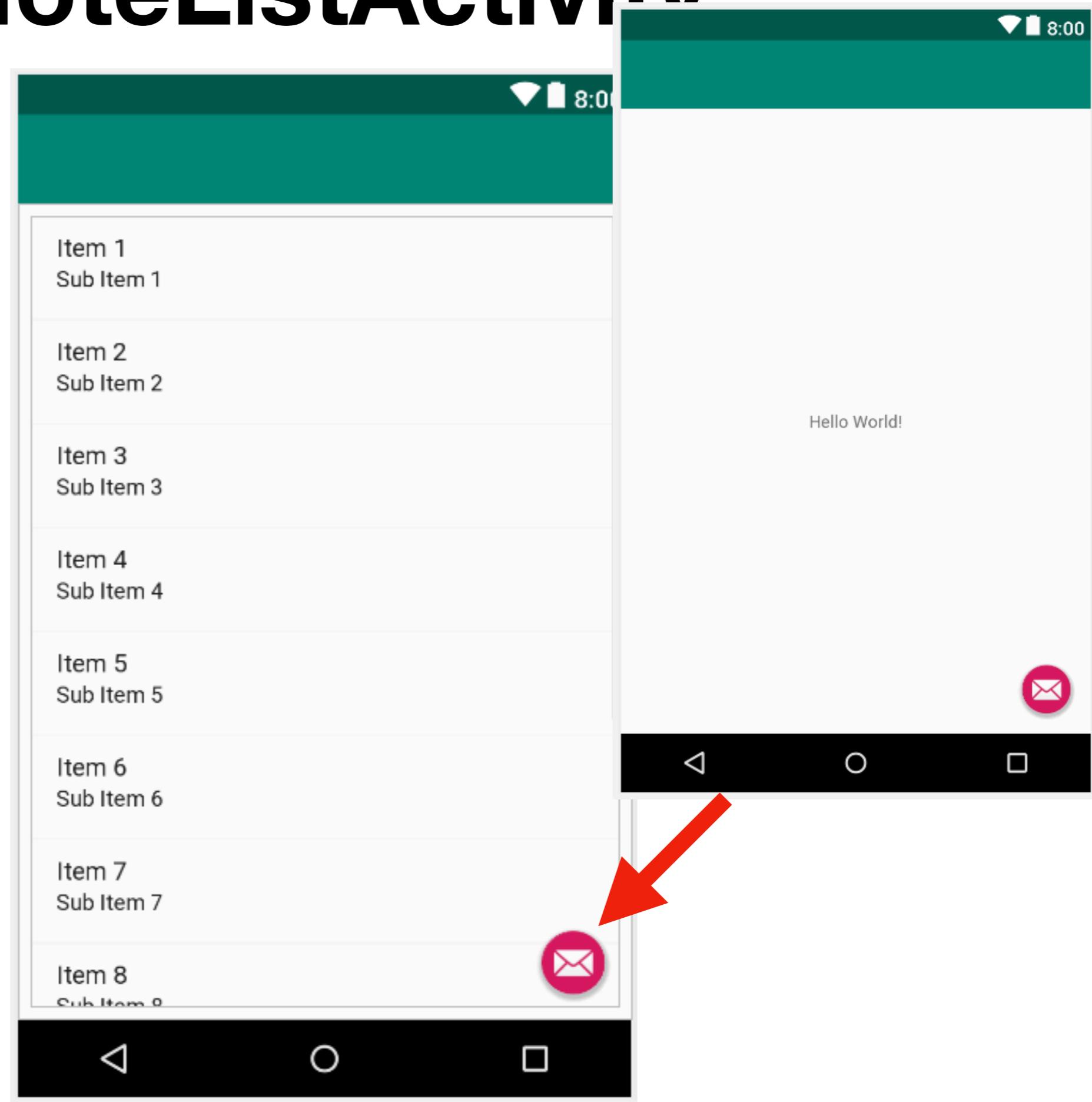
```
class WellStructuresTest{  
    ...  
    fun givenCondition_whenAction_thenShouldDo(){  
        // GIVEN – Setup condition  
  
        // WHEN – The tested action  
  
        // THEN – An assertion to validate the action  
    }  
    ...  
}
```

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

NoteListActivity



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())
    }
}
```

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())
    }
}
```

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())
    }
}
```

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"
        )
    }
}
```



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"
        )
    }
}
```



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"
        )
    }
}
```



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")))
    }
}
```

<https://developer.android.com/training/testing/espresso/>



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")))
    }
}
```

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")))
    }
}
```

<https://developer.android.com/training/testing/espresso/>

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")))
    }
}
```

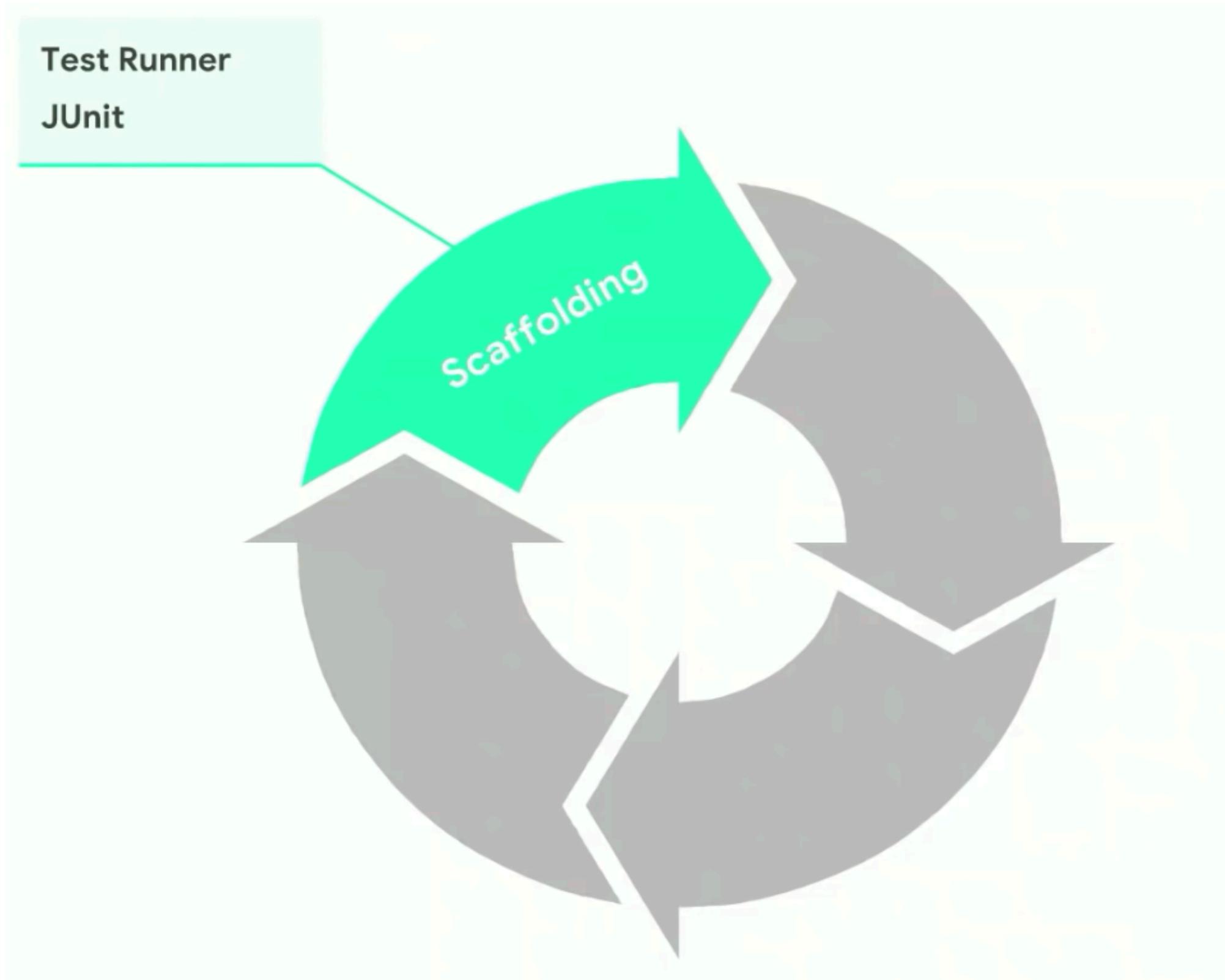


Android Test

Part of Jetpack

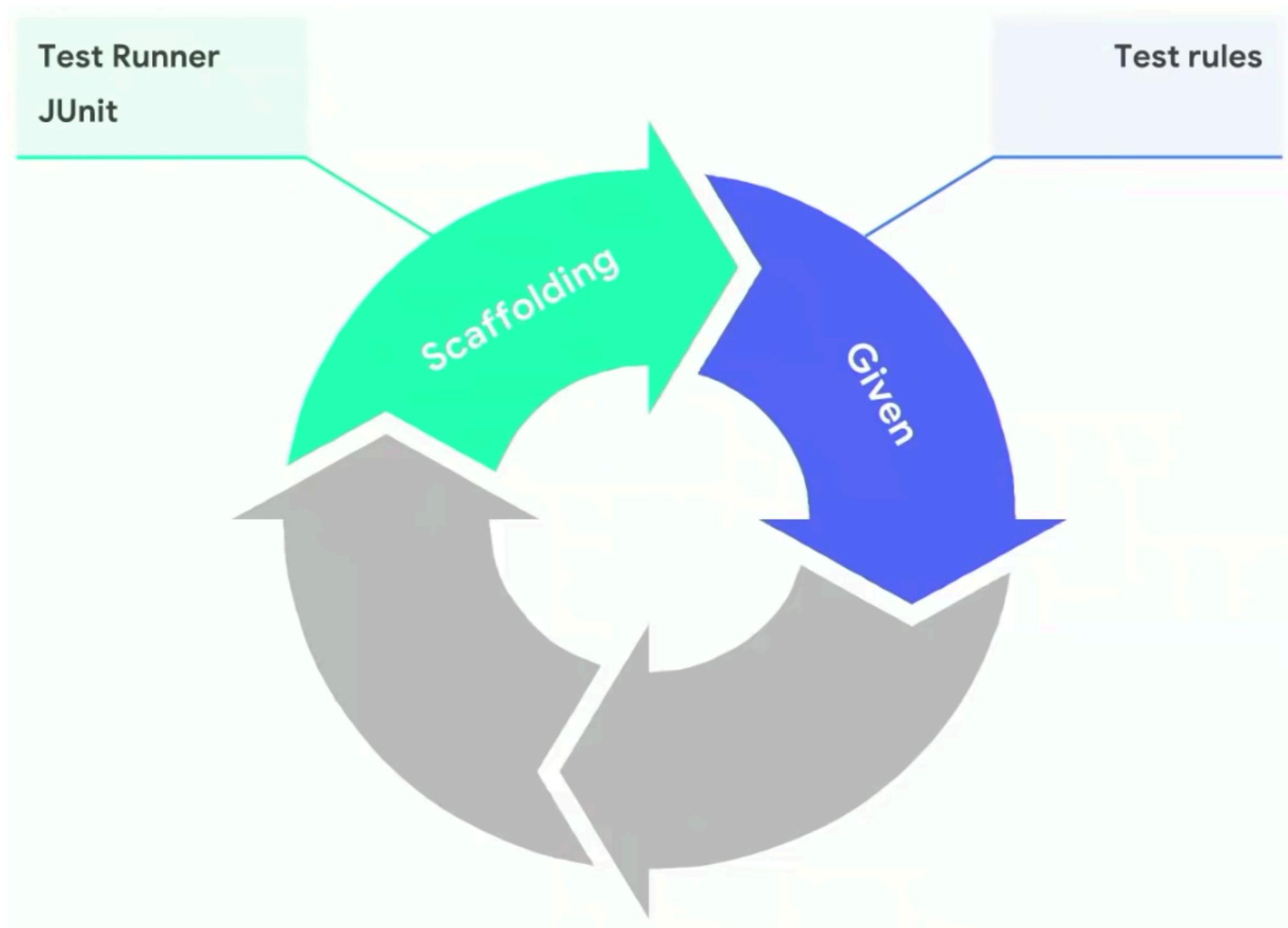
- Includes existing libraries.
- New APIs and Kotlin.
- Available on/off device.
- Open source.





```
//SCAFFOLDING

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



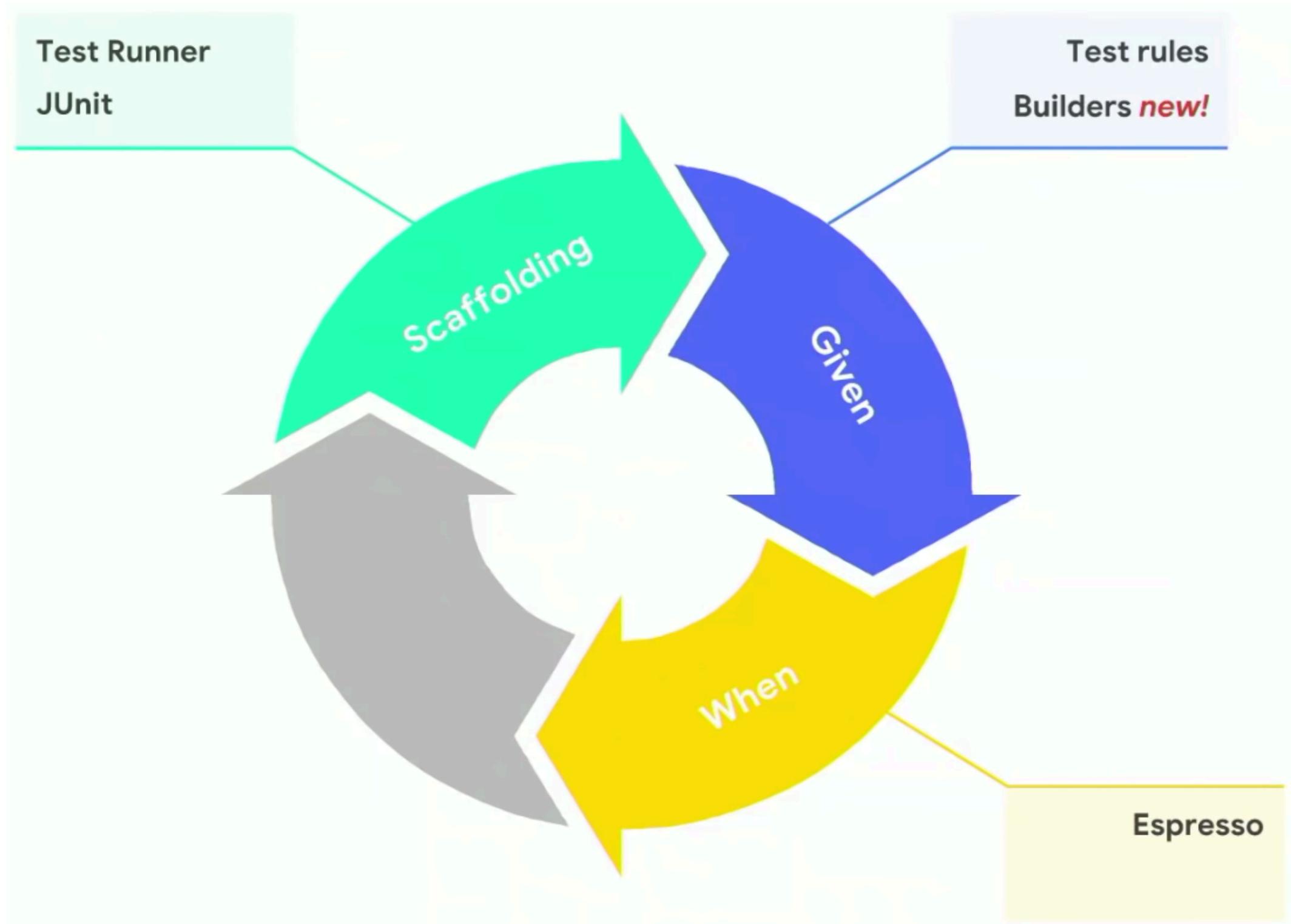
```
//GIVEN

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



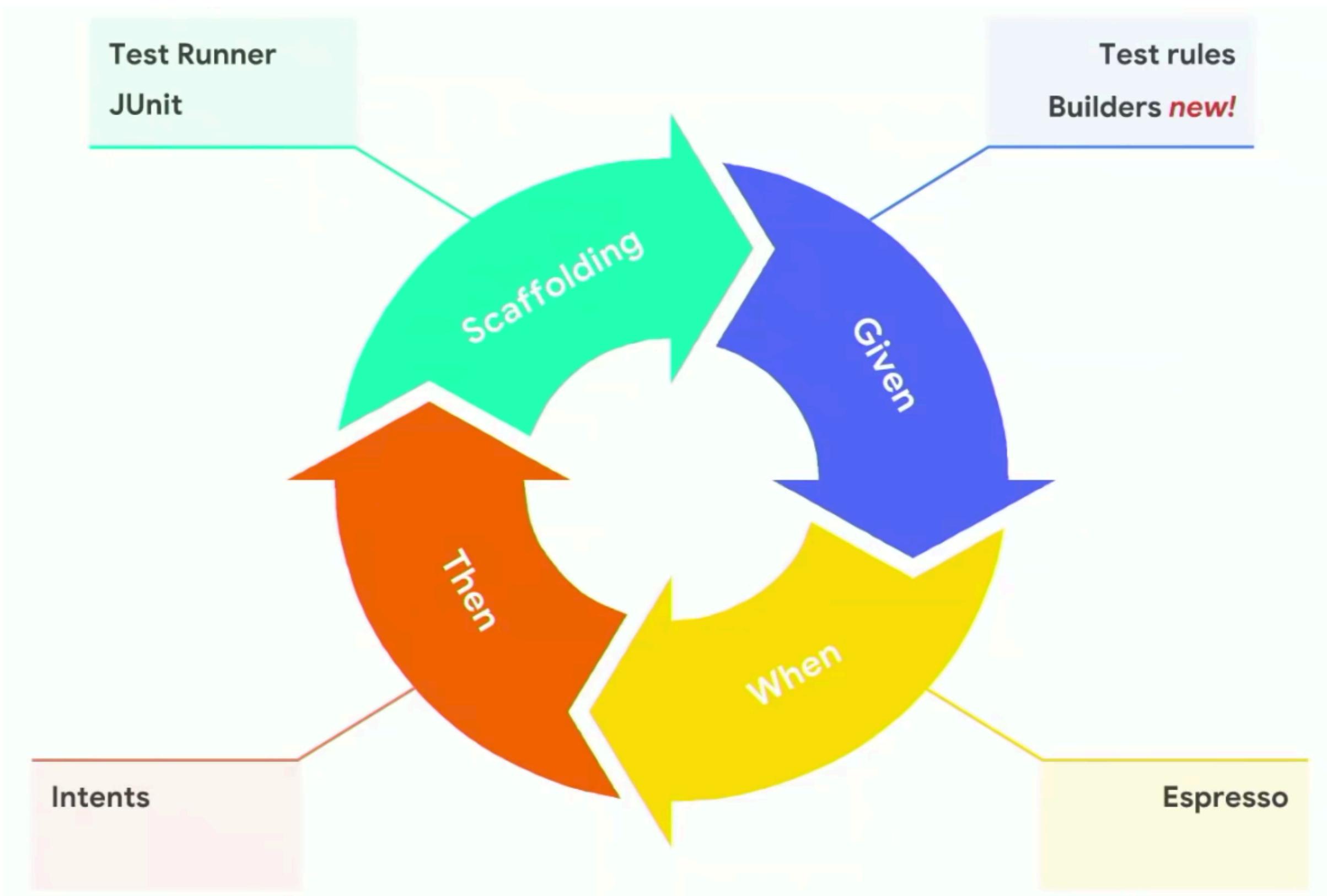
//GIVEN

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



//WHEN

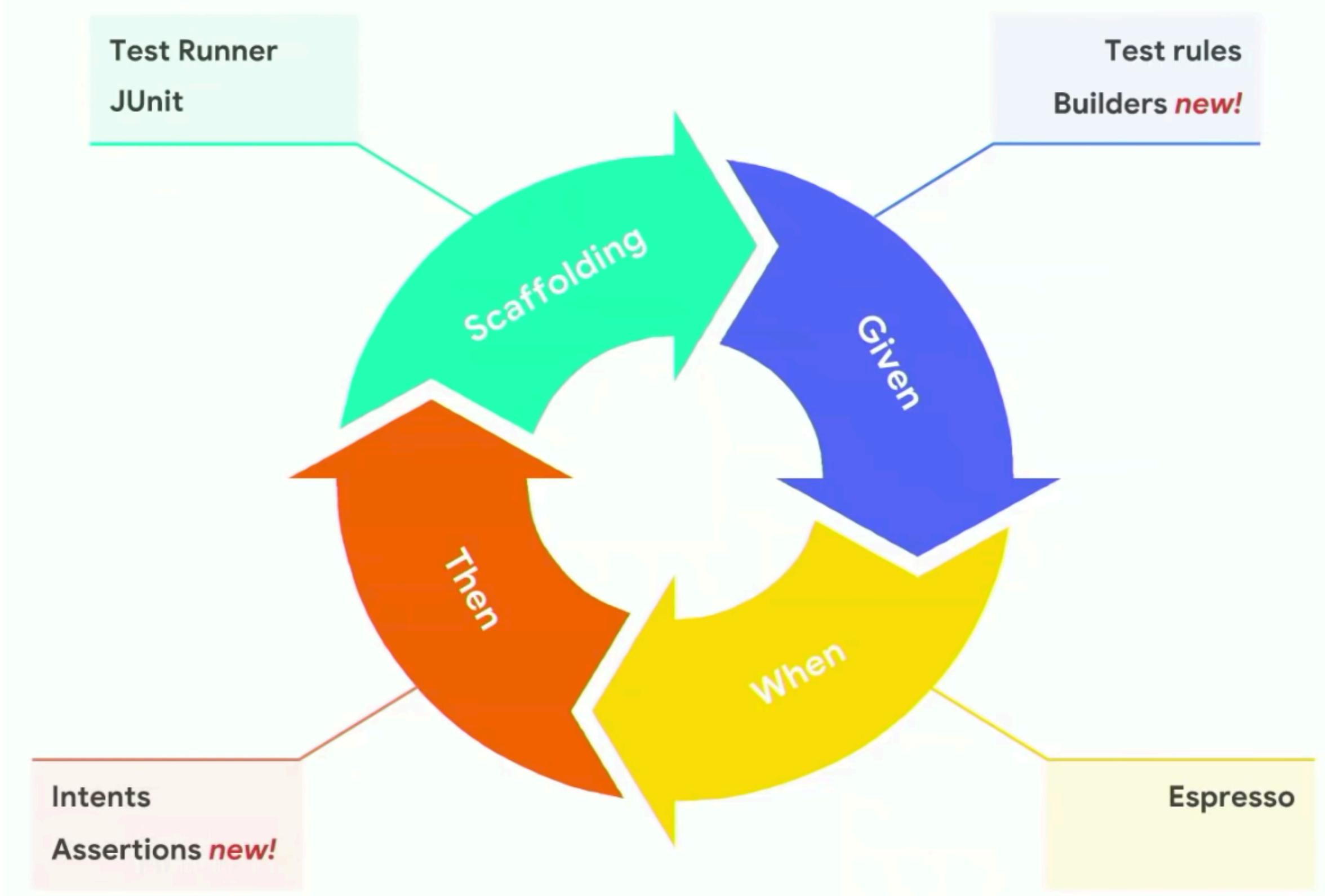
```
@RunWith(AndroidJUnit4::class)
class SimpleUnifiedTest {
    @Test
    fun testButtonClickSendsIntent() {
        onView(withId(R.id.fab)).perform(click())
    }
}
```





//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() {
        assertEquals(view.visibility, View.VISIBLE)
    }
}
```



Failed:
Expected 0 but was 16



//THEN

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



//THEN

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



Failed:
View was not visible



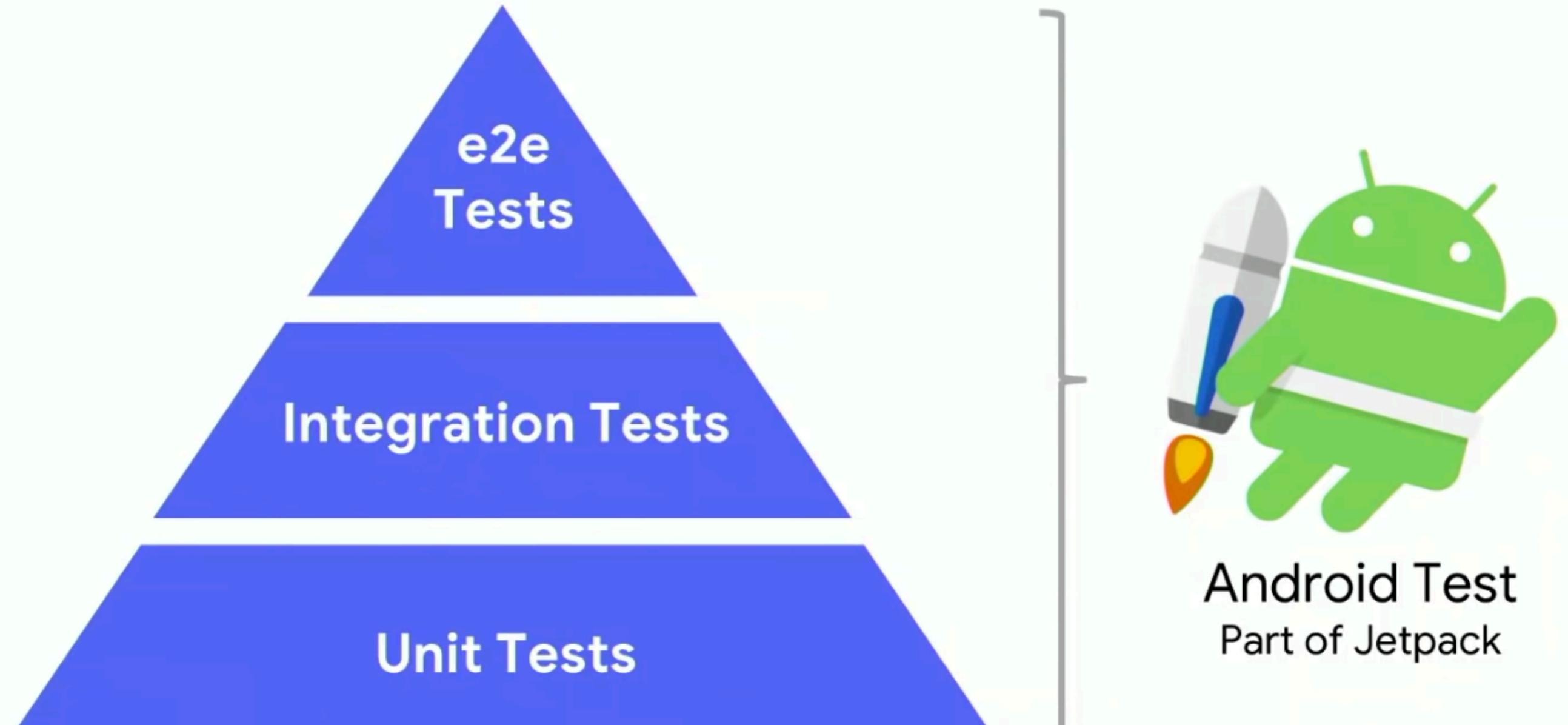
Canonical
APIs

Kotlin

Reduces
Boilerplate

Cross
Environment

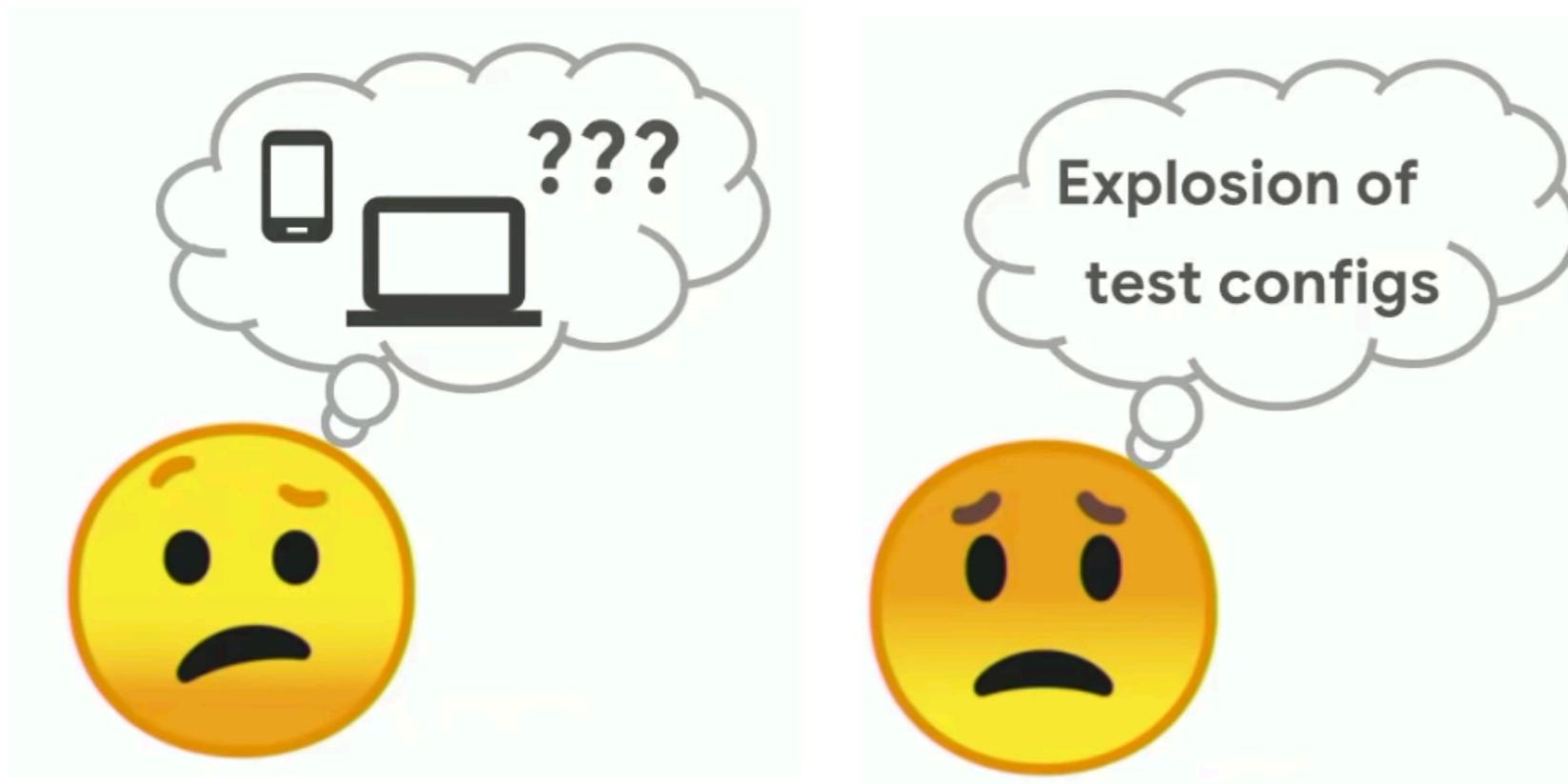
<https://developer.android.com/training/testing/>



Test execution crisis



Test execution crisis



Test execution crisis

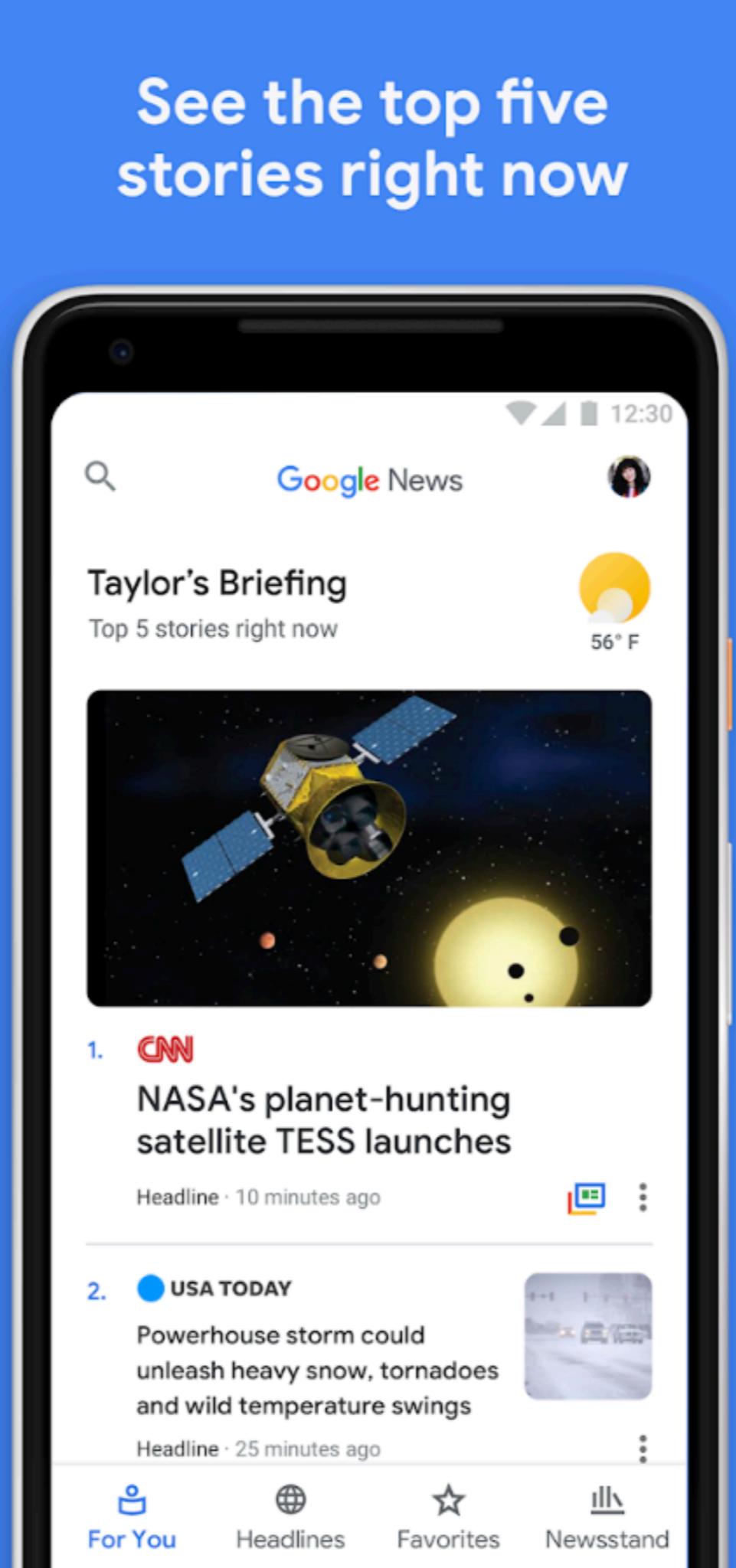
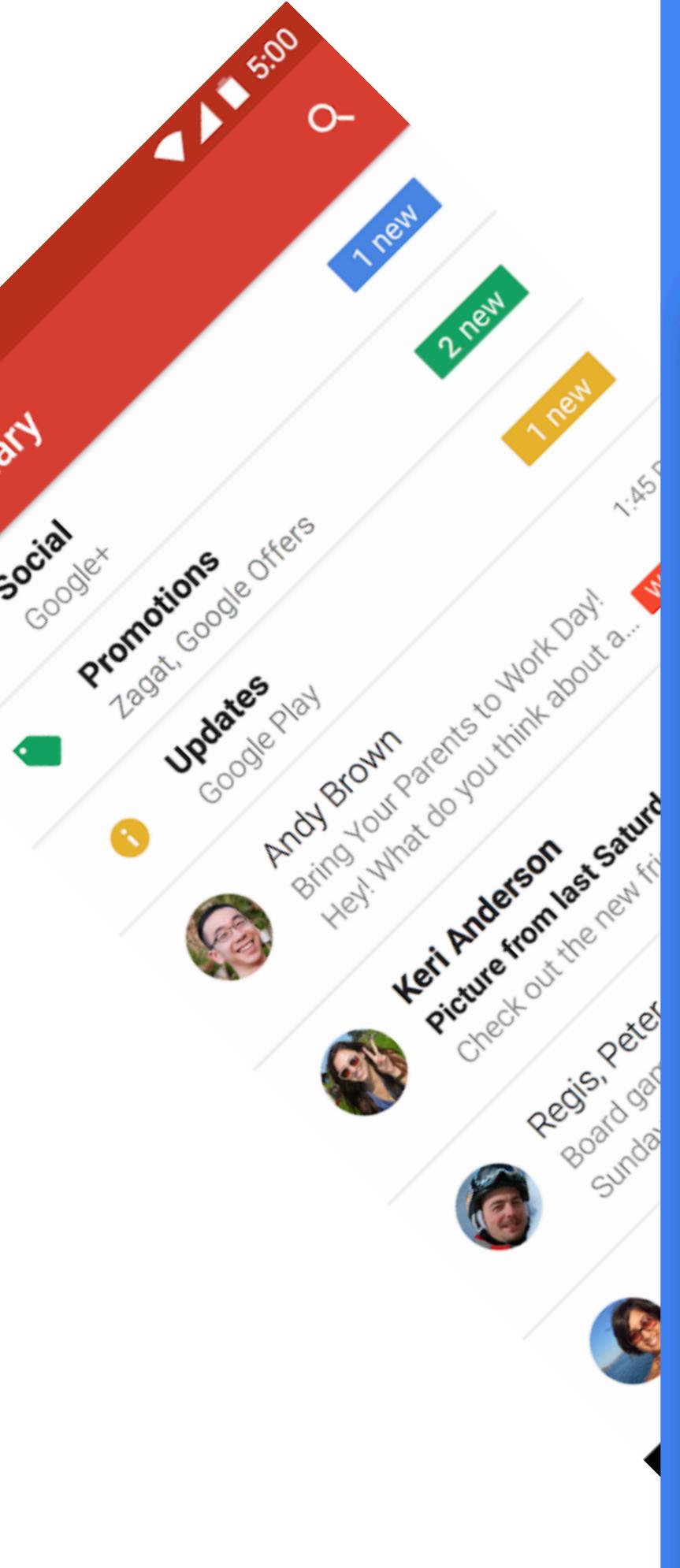


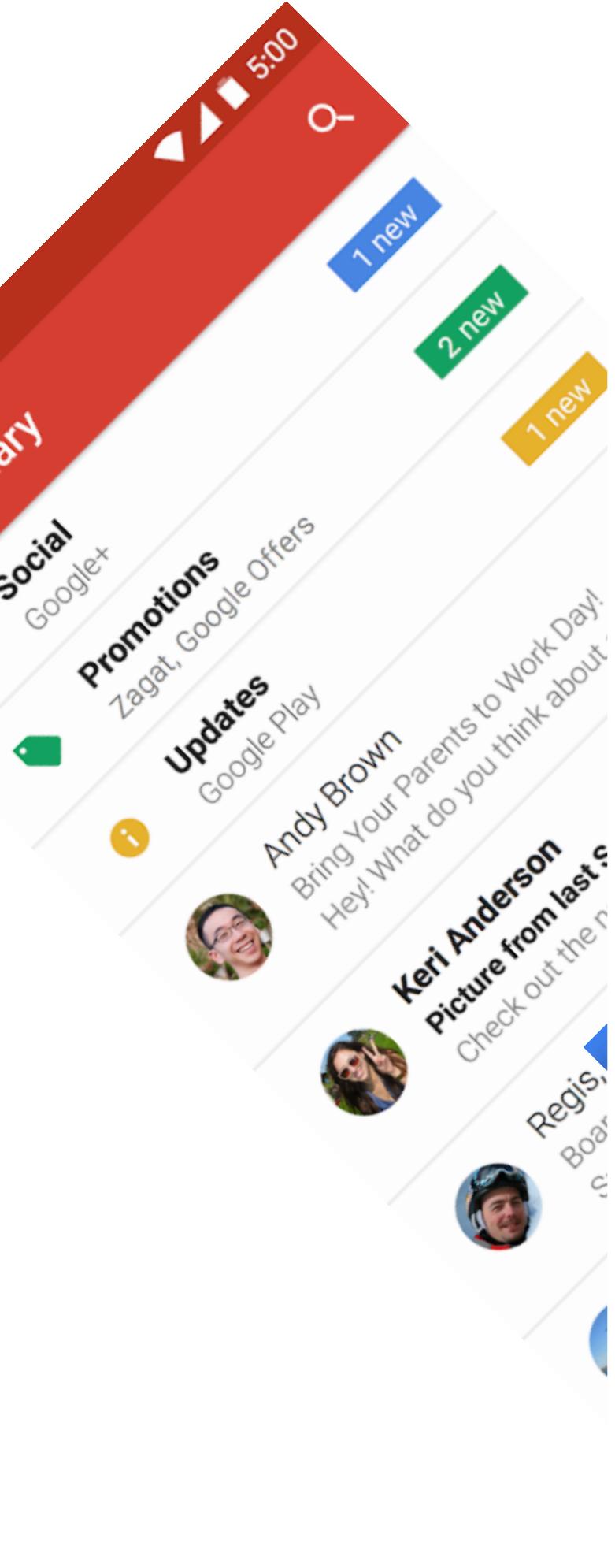
Project Nitrogen

A single entry point for tests

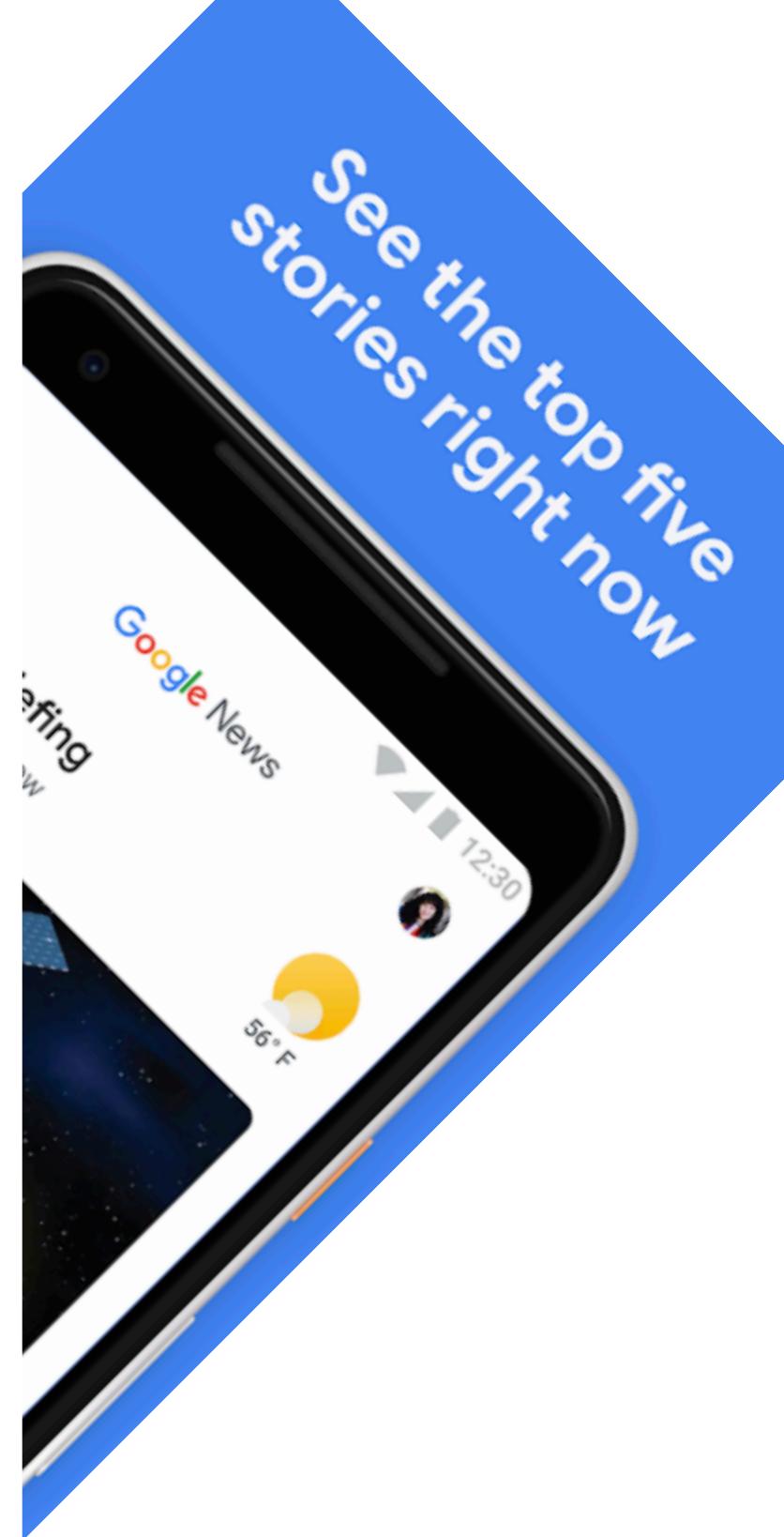
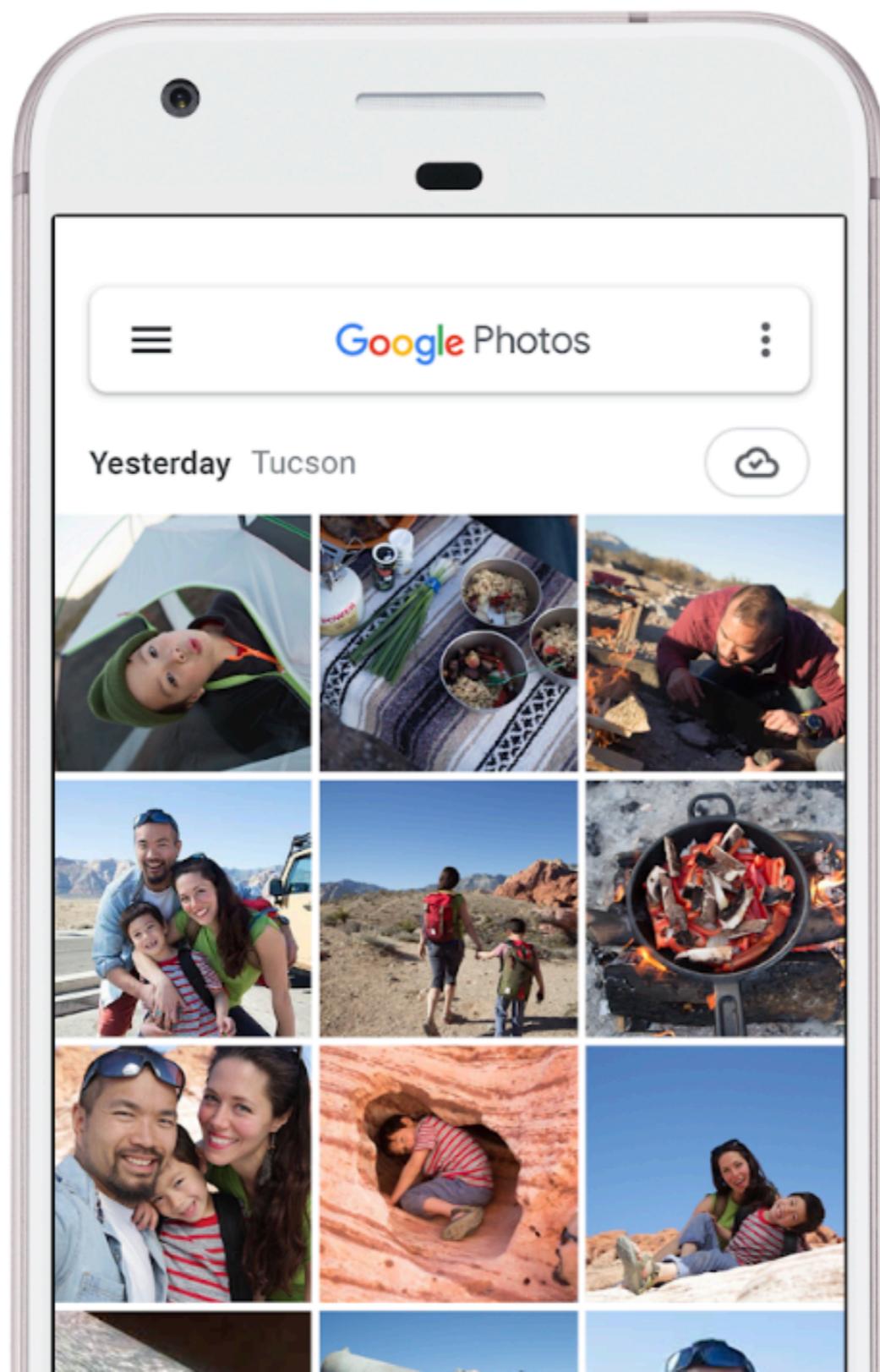


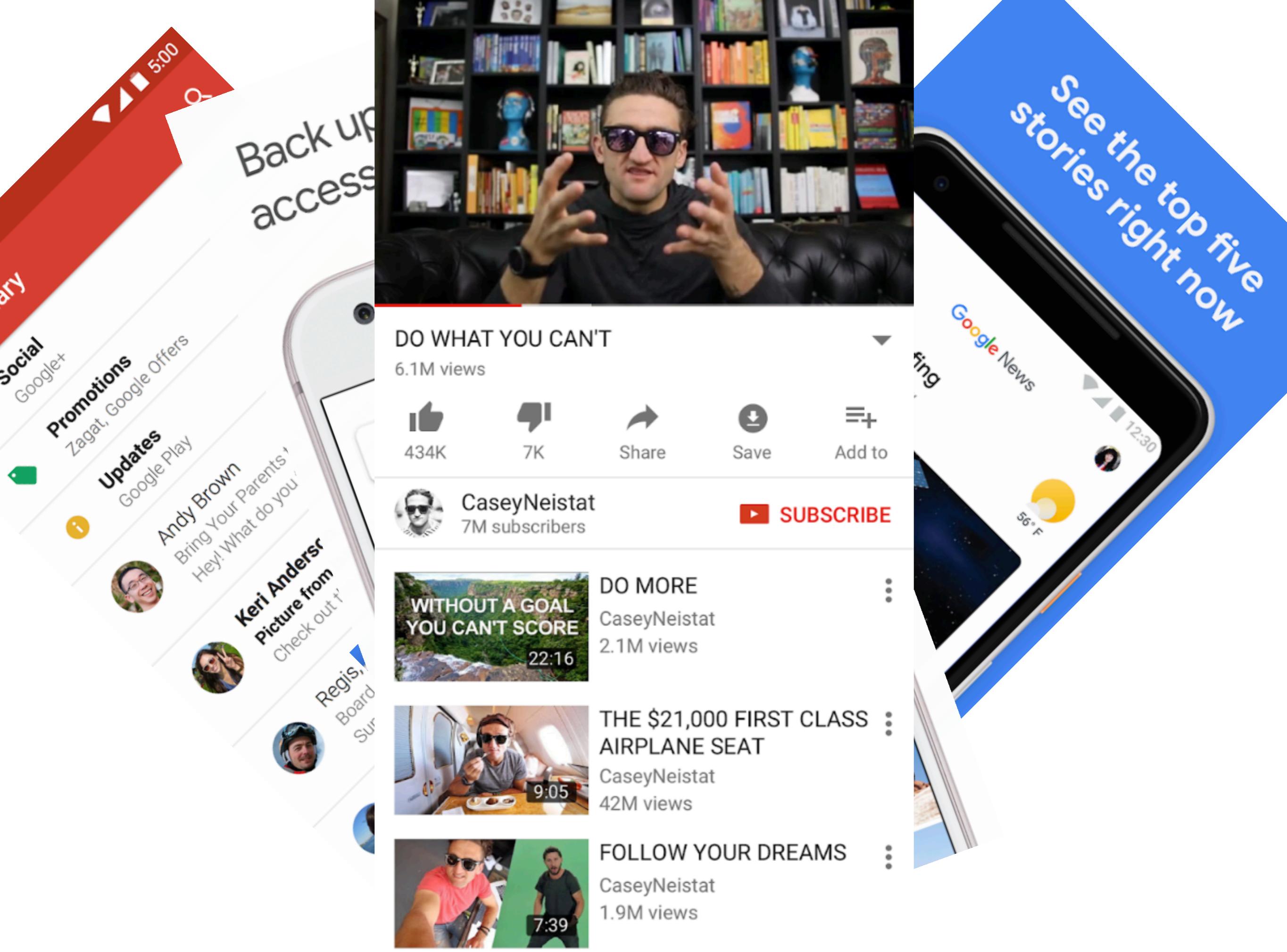
- 5:00
- ≡ Primary ⚡
-  **Social**
Google+ 1 new
-  **Promotions**
Zagat, Google Offers 2 new
-  **Updates**
Google Play 1 new
-  Andy Brown 1:45 PM
Bring Your Parents to Work Day!
Hey! What do you think about a...  
-  **Keri Anderson** 1:39 PM
Picture from last Saturday
Check out the new friend we made, Me... 
-  Regis, Peter, Rachel 3 Sep 29
Board game night?
Sunday works! If you can get Dex...  
-  **Aruna Knight**
Book you recommended
About to go on a trip and was h...   

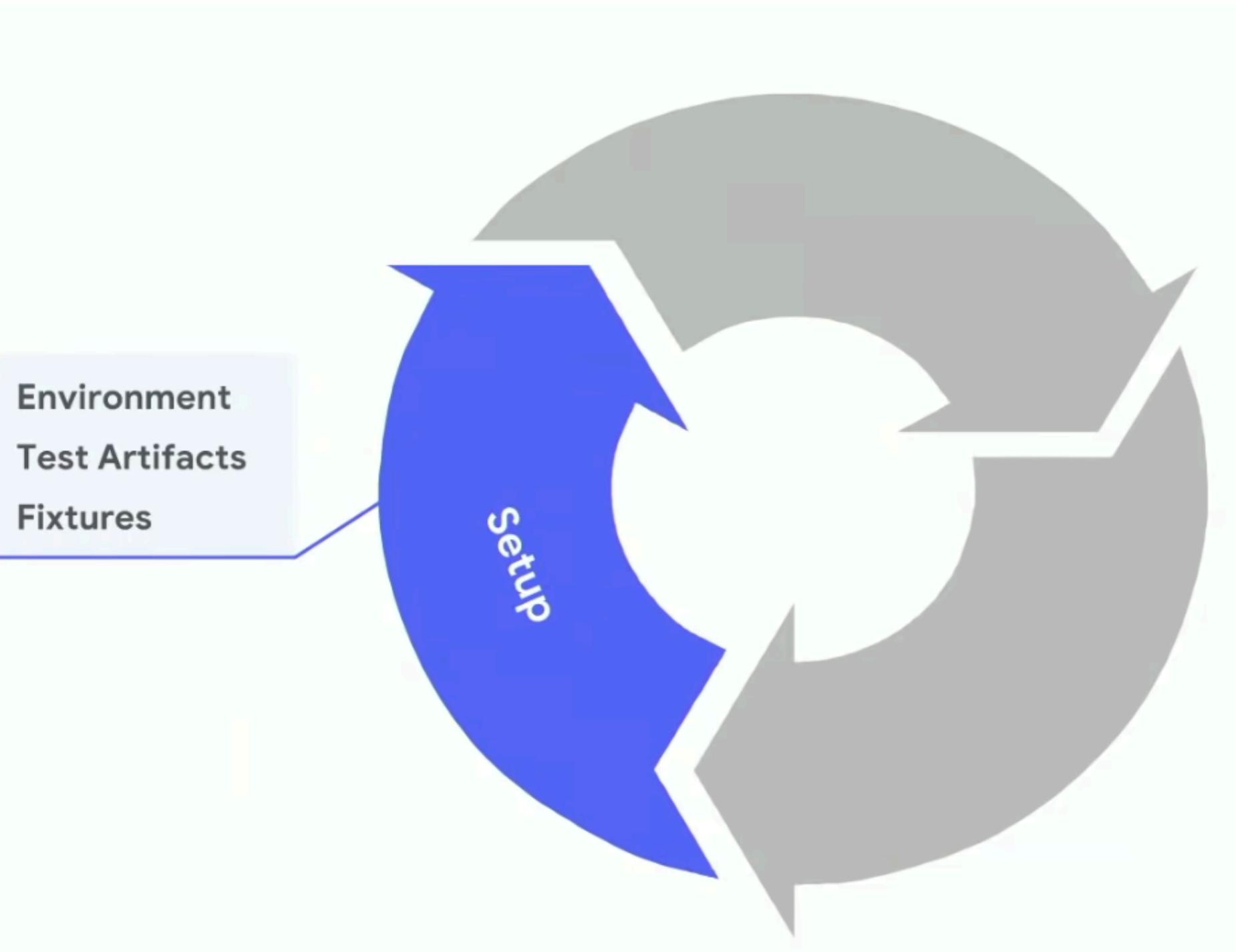


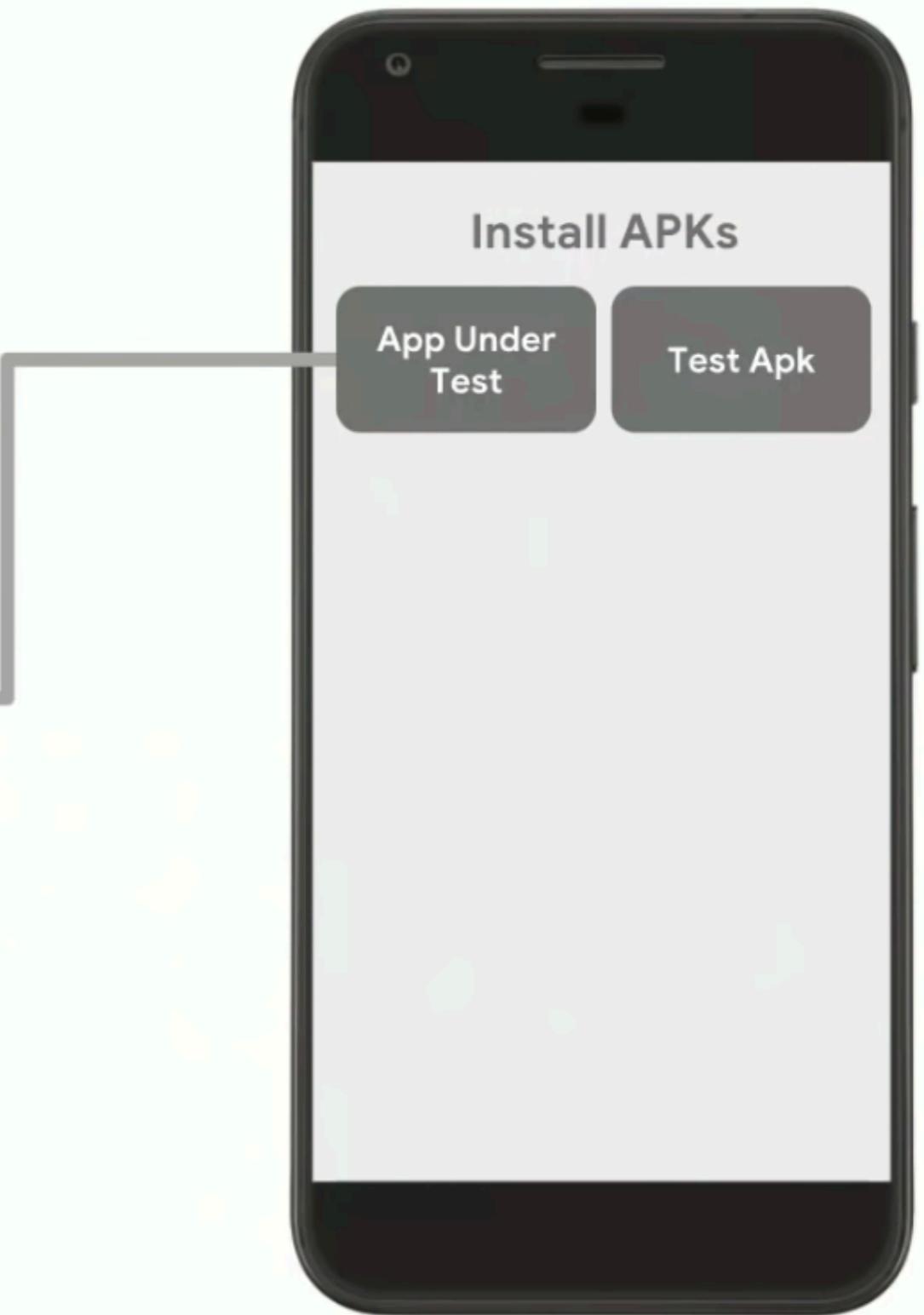


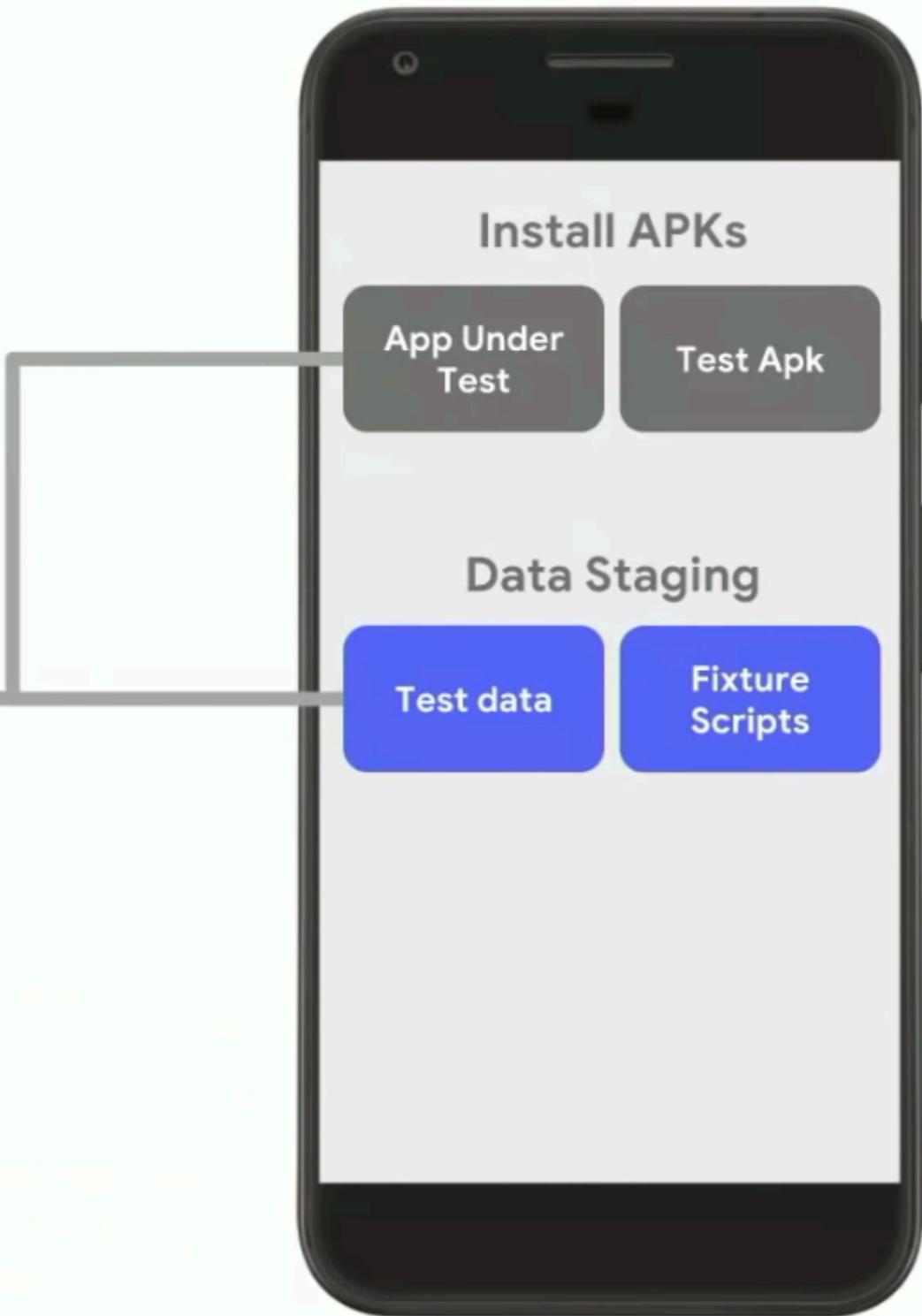
Back up your photos & access them anywhere

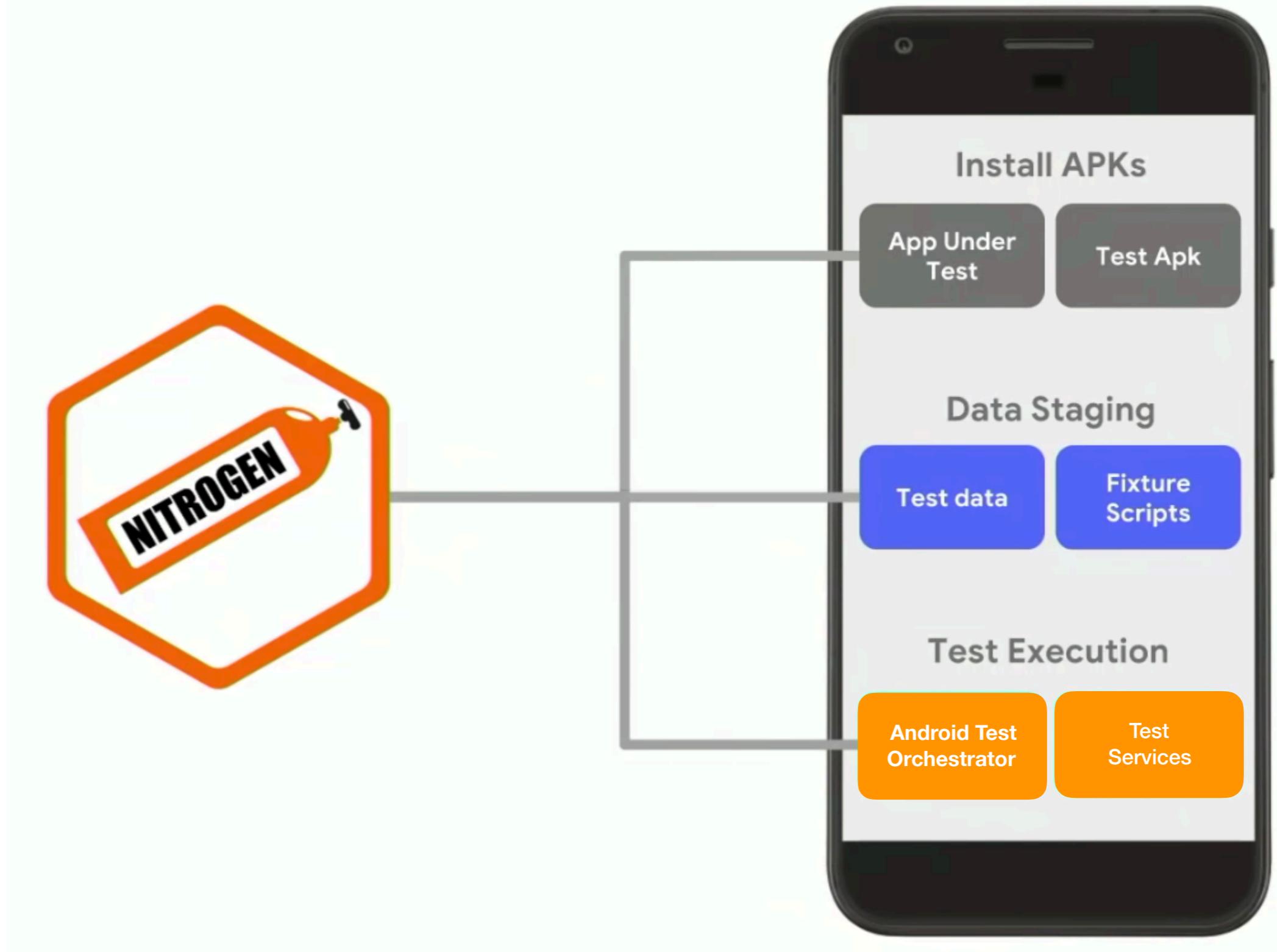












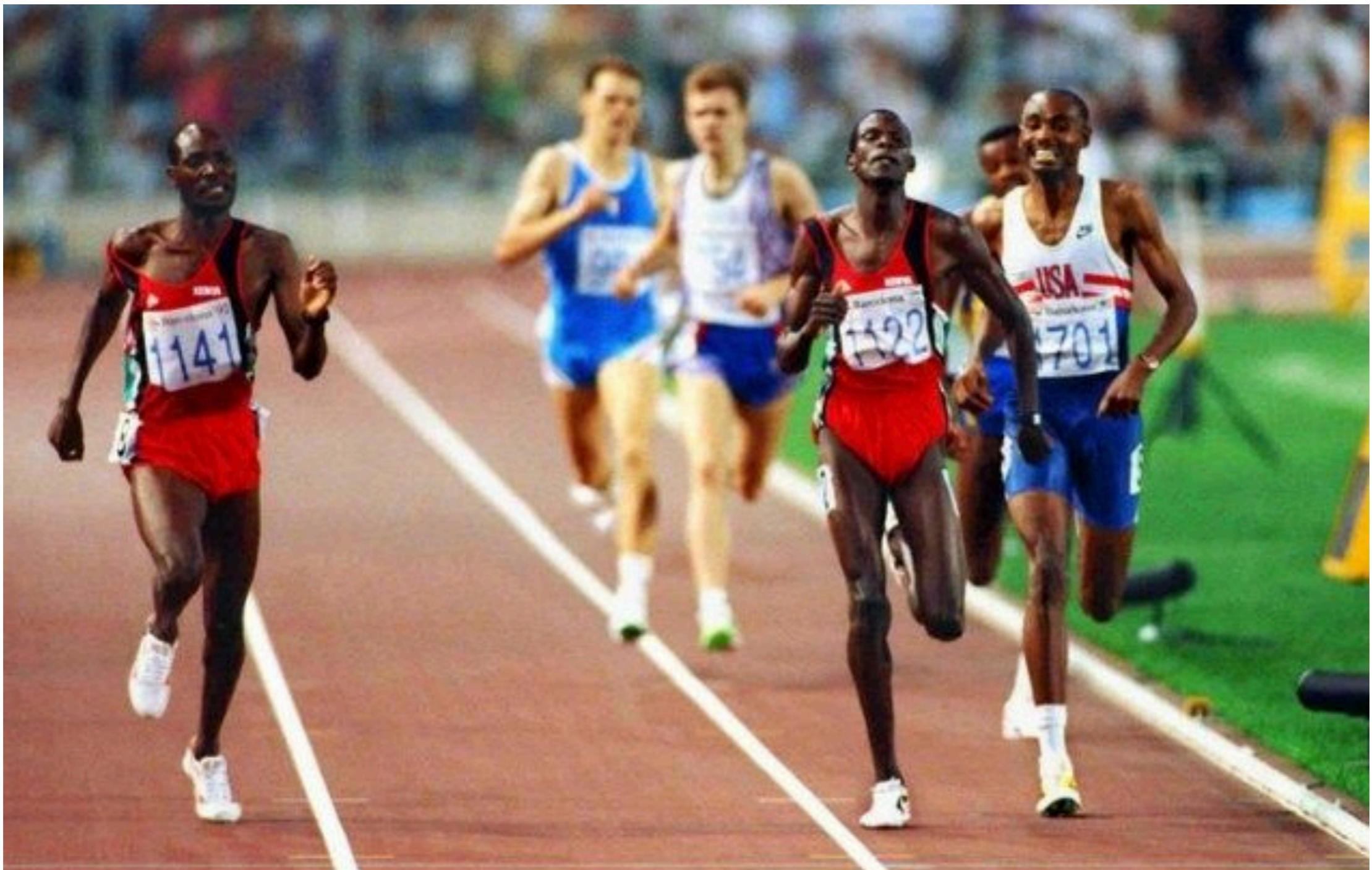
Running in the Real World



Sprint Running



Midd-Range Distance Running



Long Distance Running



Track Running



Road Running



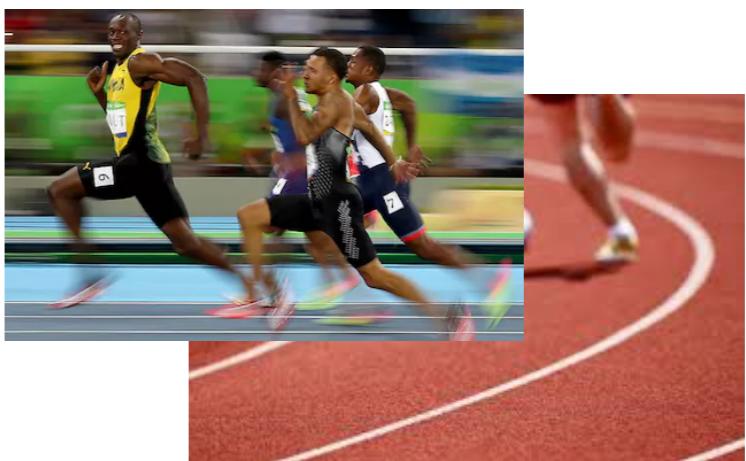
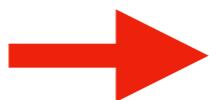
Cross Country Running



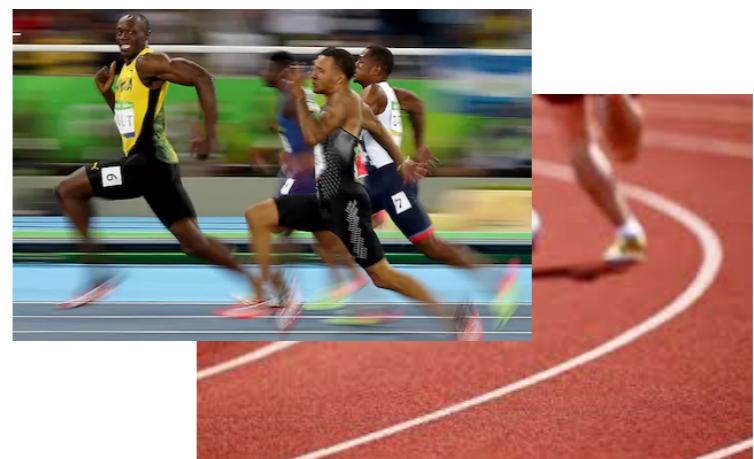
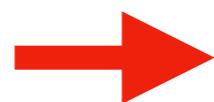
Mountain Running



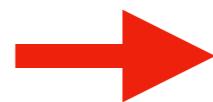
jUnit Test

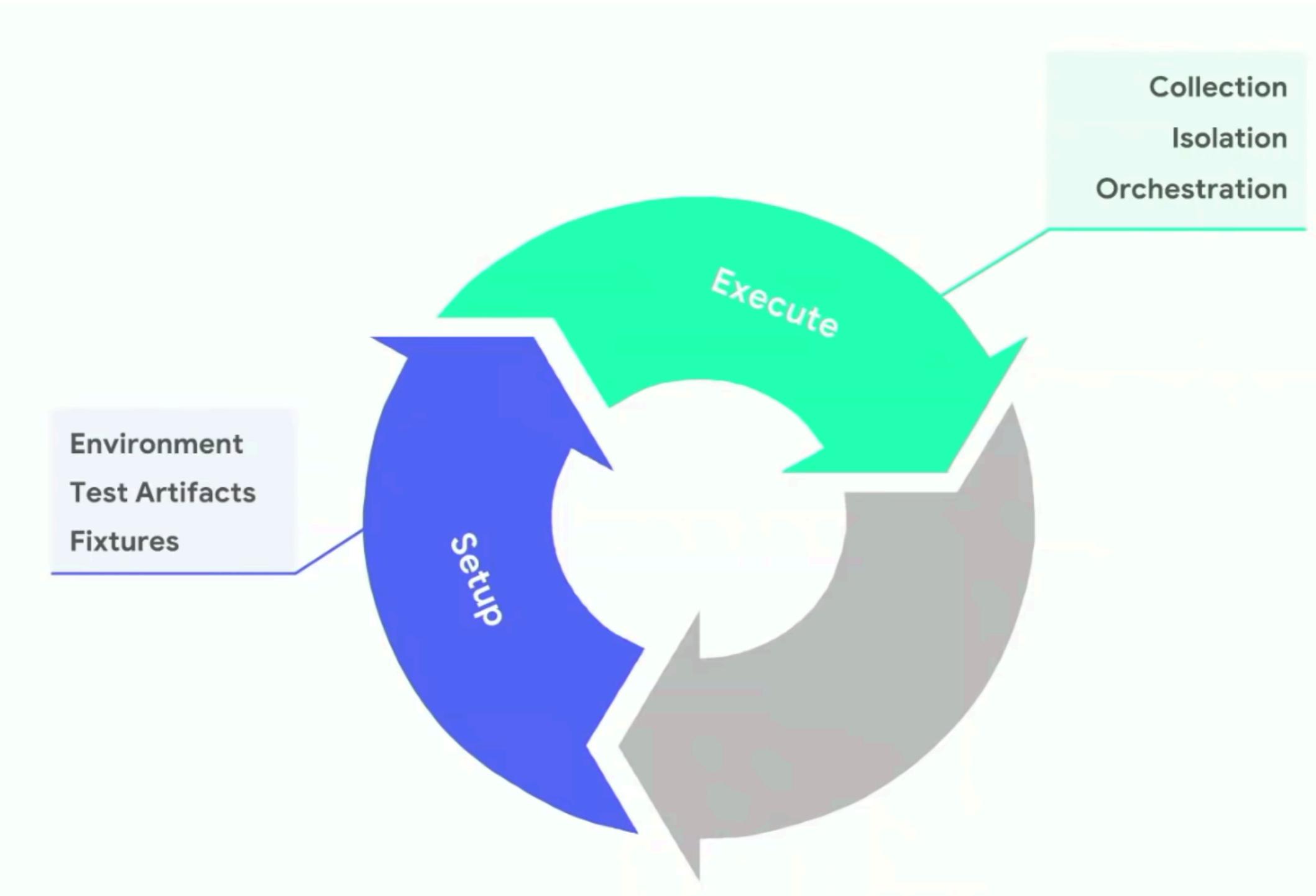


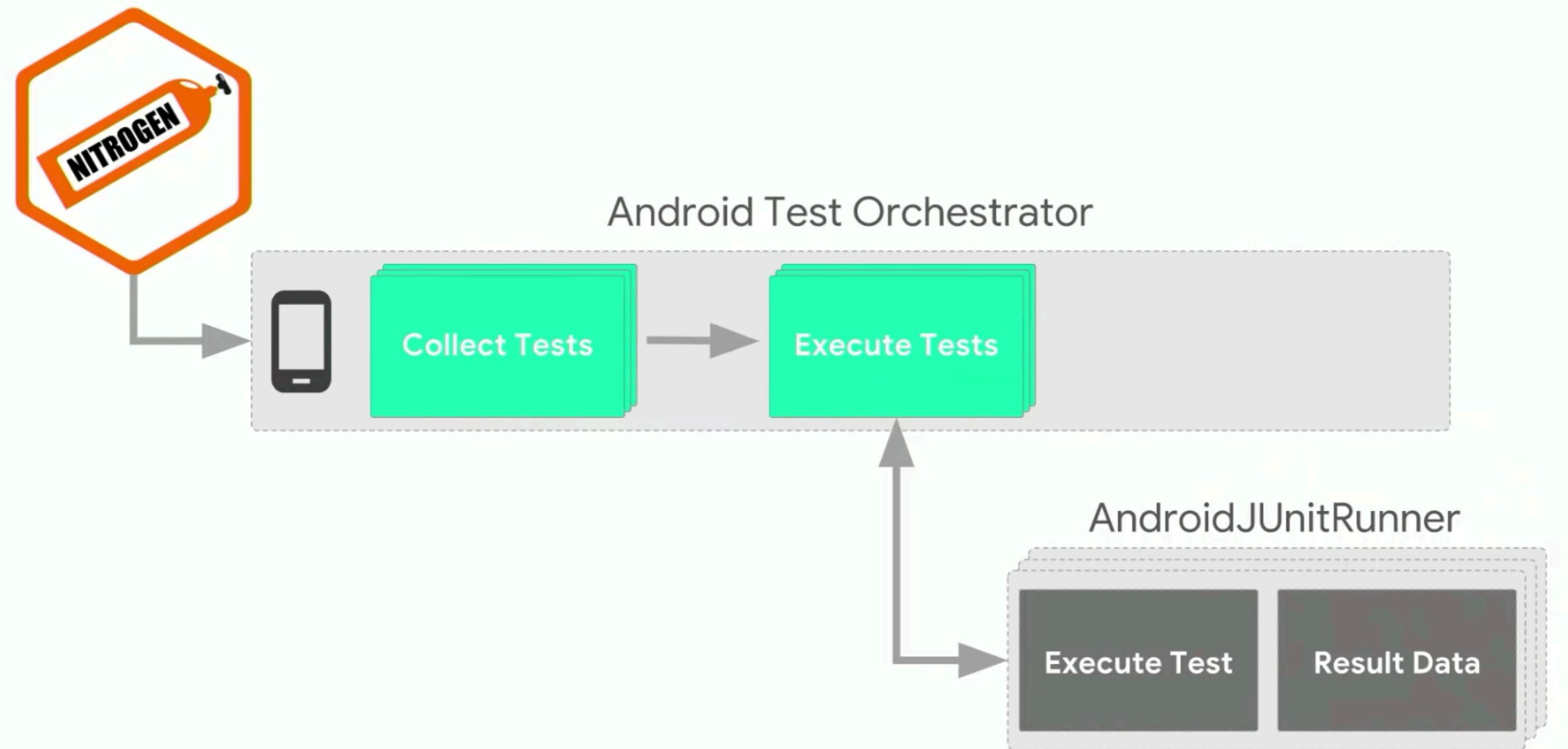
jUnit Test

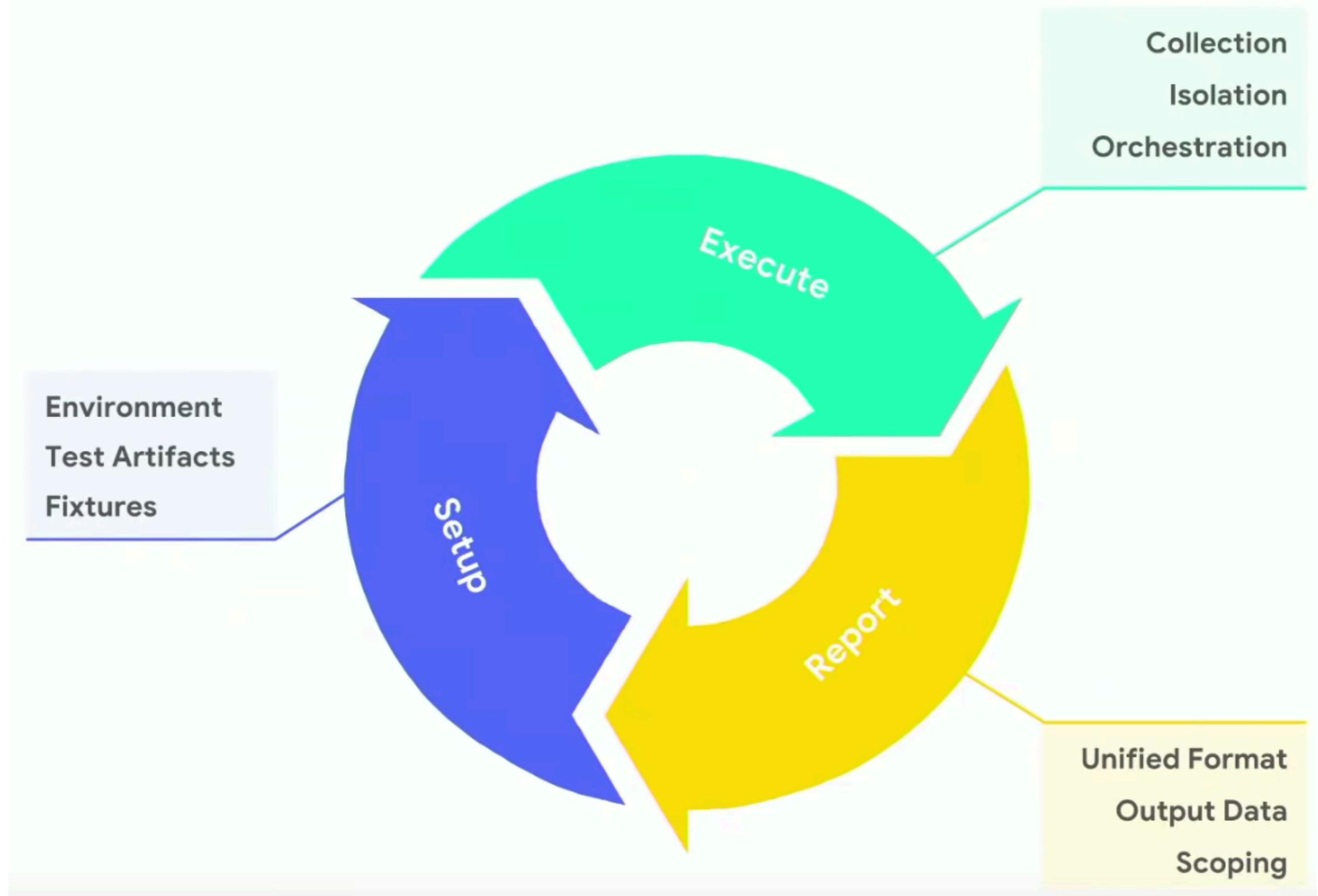


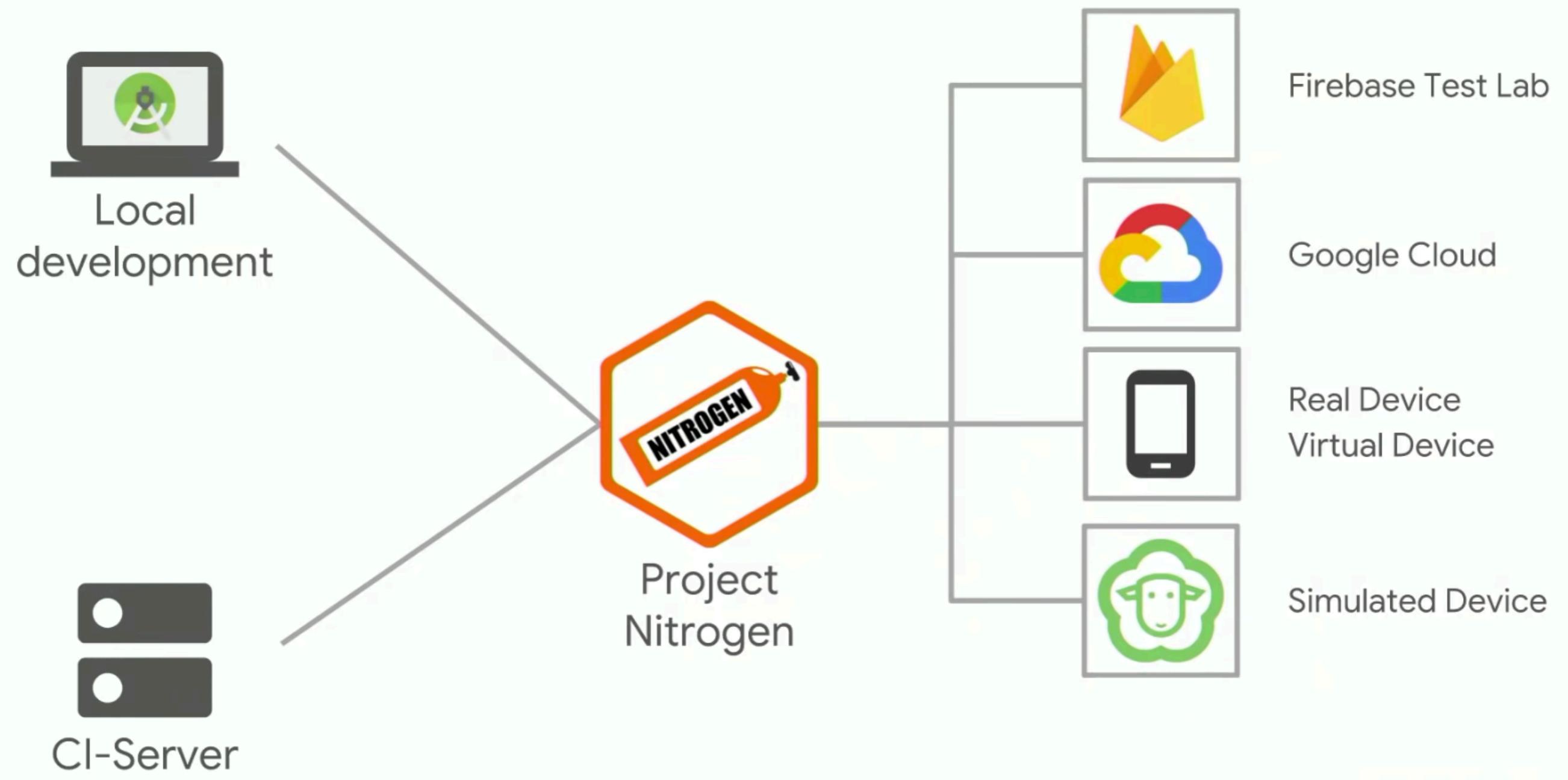
TestSuite





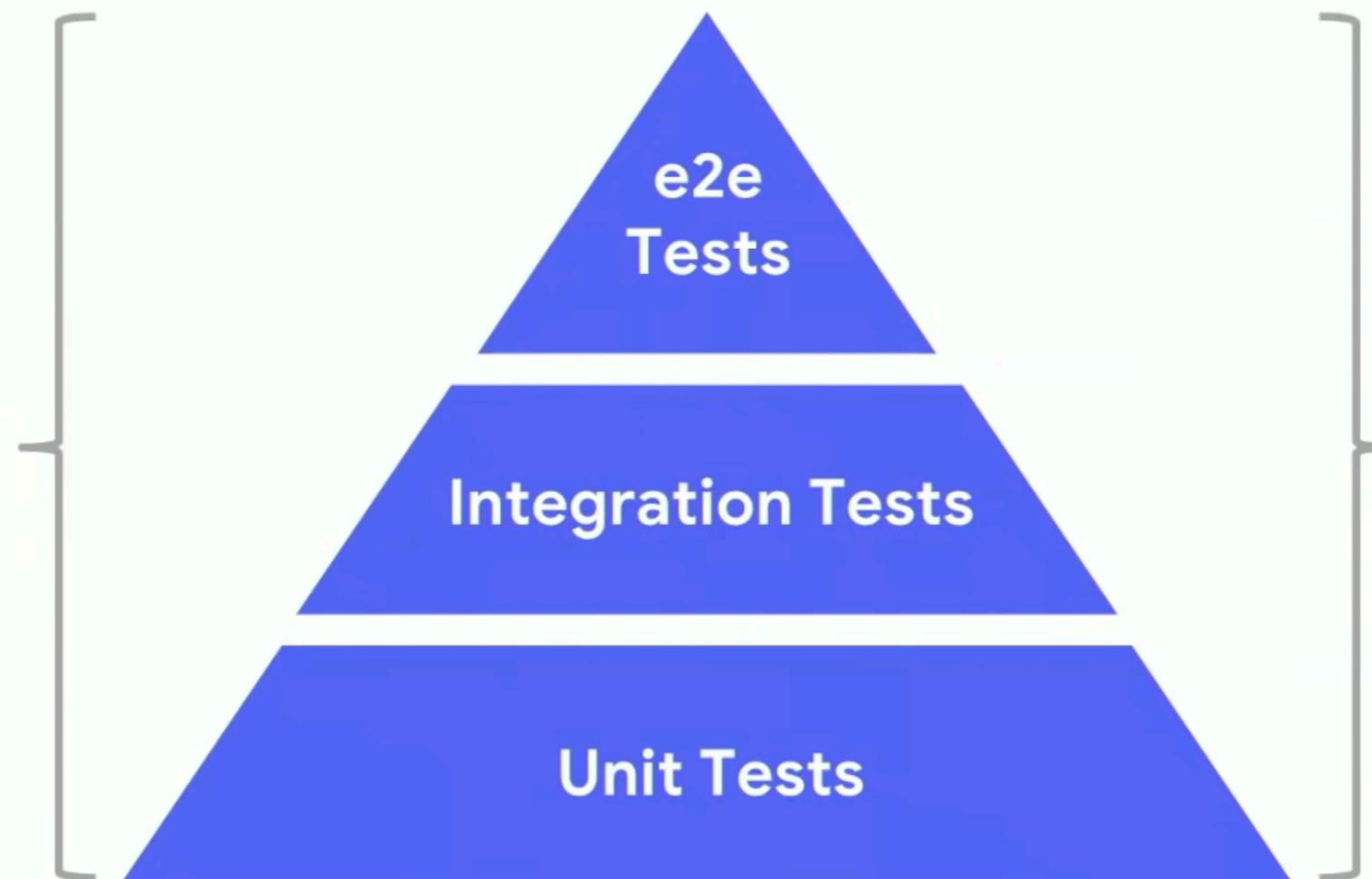






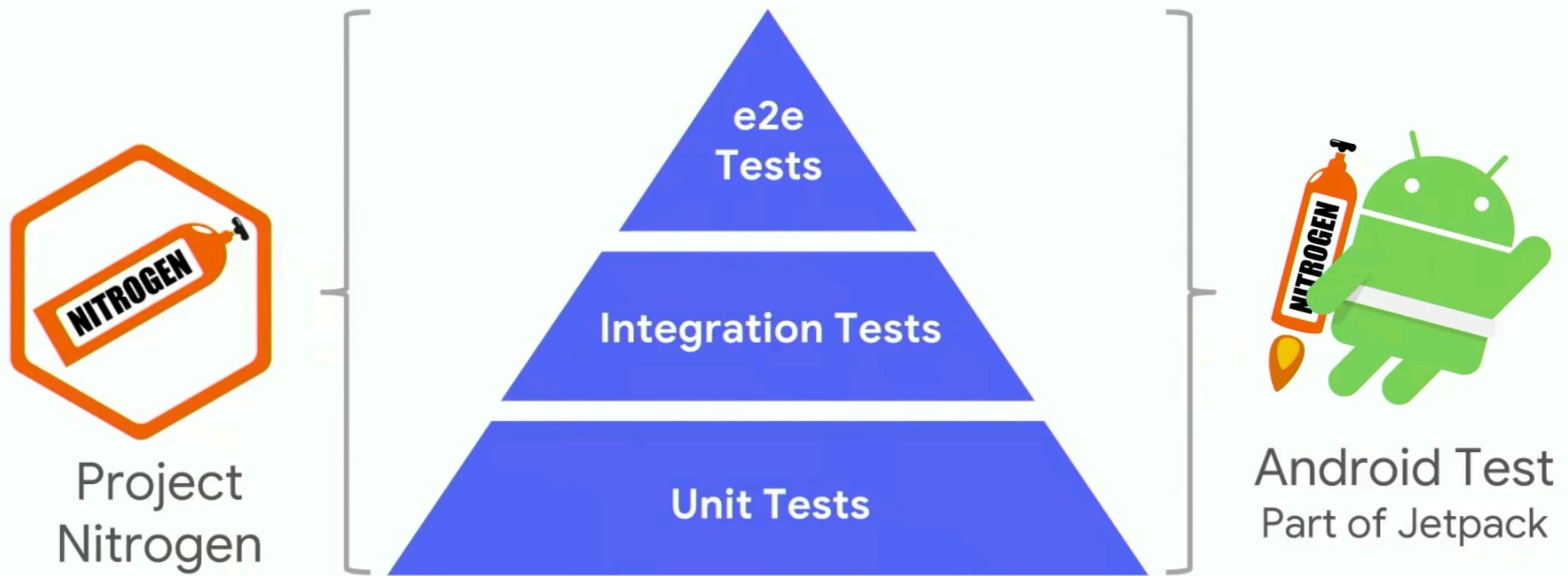


Project
Nitrogen



Android Test
Part of Jetpack

DEMO



Practical Exam

- **IMPORTANT!** Please submit your Exam Attendance Date preference here:
 - <http://bit.ly/maExamDate>

Software Requirements

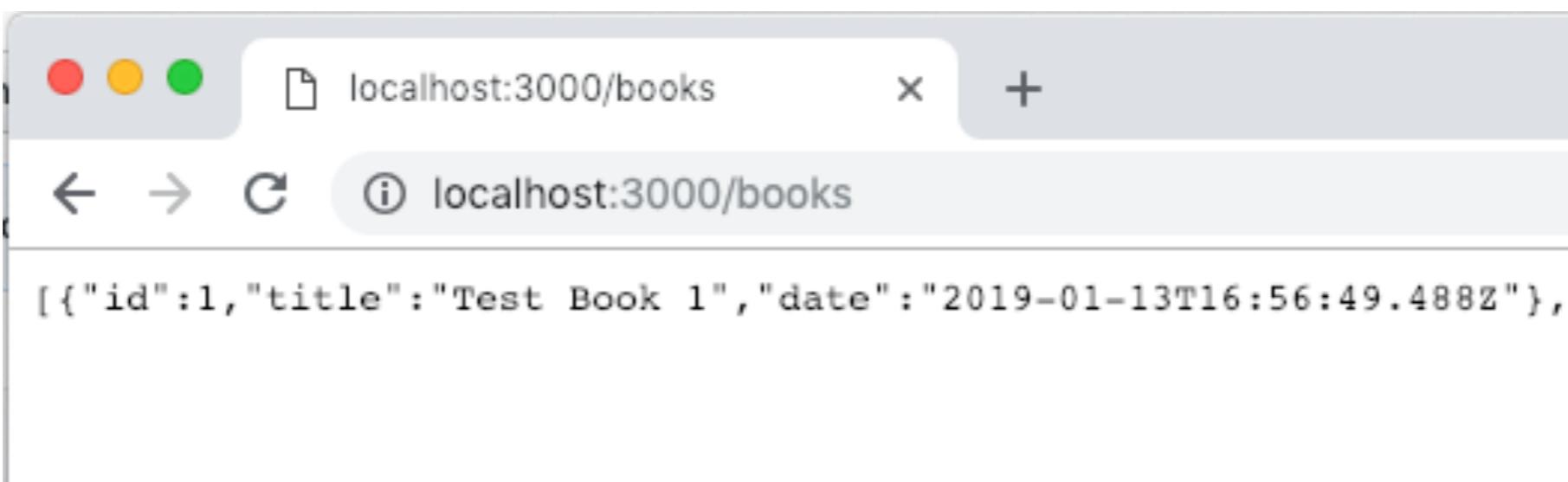
- Ensure that you have installed on your machine:
 - Git
 - nodeJs
 - Npm

Execute the Server

- Ensure that you are able to execute the server from:
 - <https://github.com/dancojocar/MA/tree/master/lectures/13/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 (v11) and npm (v6.5).

Verify the Server

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



A screenshot of a web browser window. The address bar shows the URL `localhost:3000/books`. The main content area of the browser displays a JSON array:

```
[{"id":1,"title":"Test Book 1","date":"2019-01-13T16:56:49.488Z"},
```

Practical Exam

- Two hours to solve the requirements.
- In these two hours:
 - 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 prepared!
 - Request clarifications only from the exam supervisor.
- If found breaking 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.

