Lecture #14
Testing Frameworks & Exam Discussions

Mobile Applications 2019-2020

Types of Testing Software

System Testing

System Integration Testing

Unit testing

User Acceptance testing

Smoke testing

Usability testing

All Pairs testing

Ad-hoc testing

Performance Testing

Soak Testing

Security Testing

Beta Testing

Regression Testing

Acceptance Testing

Black Box testing

Sanity Testing

Retesting

Glass box Testing

Load Testing

Risk-based Testing

Stress Testing

Accessibility Testing

Backward Compatibility Testing

Static Testing

Localization Testing

Agile Testing

Boundary Value Testing

Stability Testing

Negative Testing

API Testing

Big Bang Integration testing

Scalability Testing

Non-functional testing

Automated testing

Bottom up Integration testing

Gorilla Testing

Pair Testing

Volume testing

Branch Testing

Keyword-driven Testing

Vulnerability Testing

White box 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

Types of Testing Software

System Testing

System Integration Testing

Unit testing

User Acceptance testing

Smoke testing

Usability testing

All Pairs testing

Ad-hoc testing

Performance Testing

Soak Testing

Security Testing

Beta Testing

Regression Testing

Acceptance Testing

Black Box testing

Sanity Testing

Retesting

Glass box Testing

Load Testing

Risk-based Testing

Stress Testing

Accessibility Testing

Backward Compatibility Testing

Static Testing

Localization Testing

Agile Testing

Boundary Value Testing

Stability Testing

Negative Testing

API Testing

Big Bang Integration testing

Scalability Testing

Non-functional testing

Automated testing

Bottom up Integration testing

Gorilla Testing

Pair Testing

Volume testing

Branch Testing

Keyword-driven Testing

Vulnerability Testing

White box 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

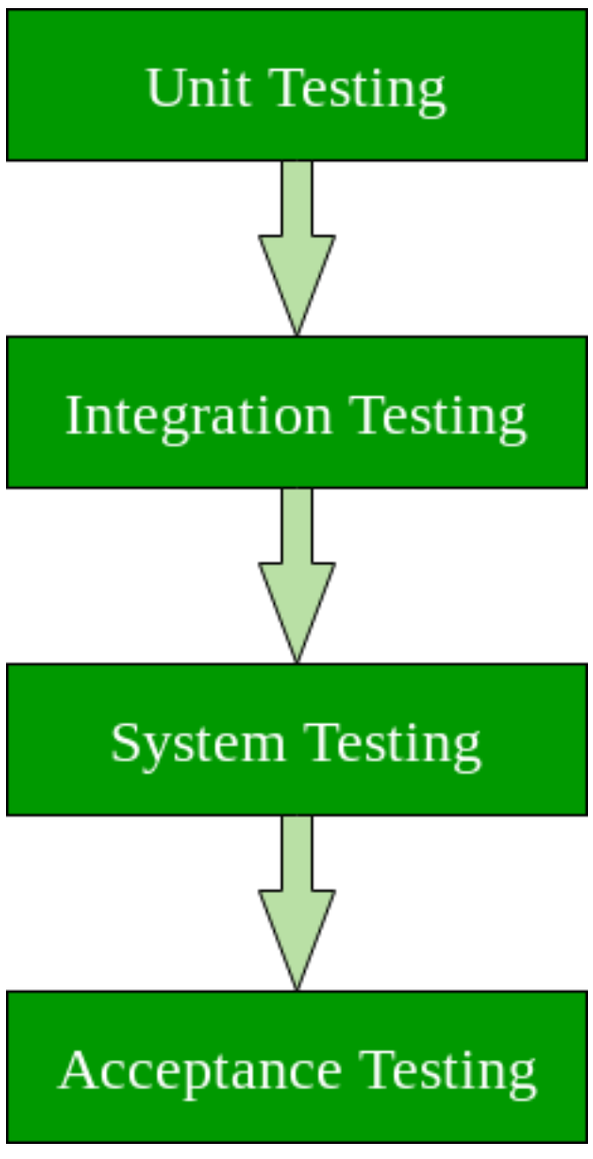
https://www.testingexcellence.com/types-of-software-testing-complete-list/

Types of Testing Software

Software Testing can be broadly classified into two types:

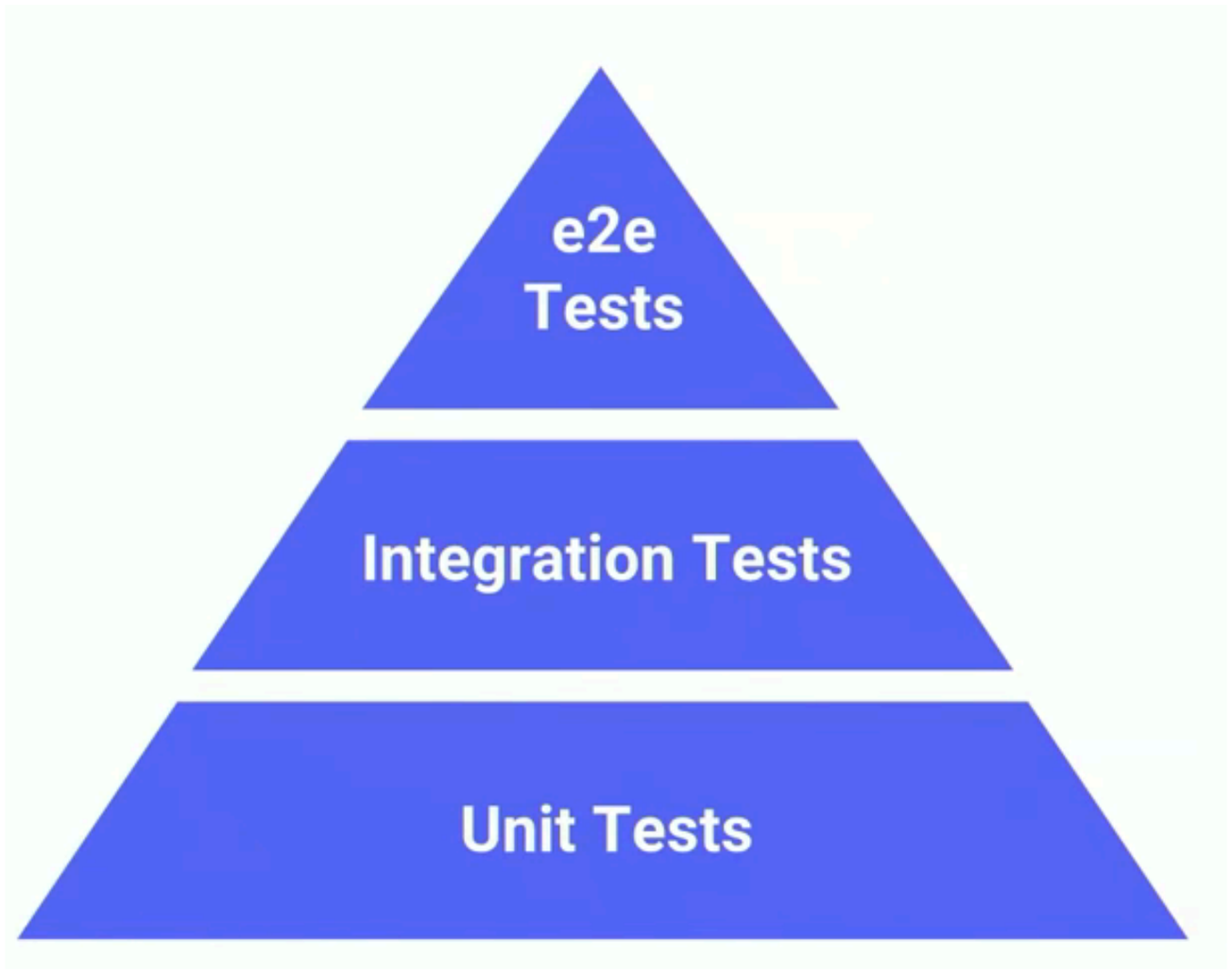
- Manual Testing
- Automation Testing

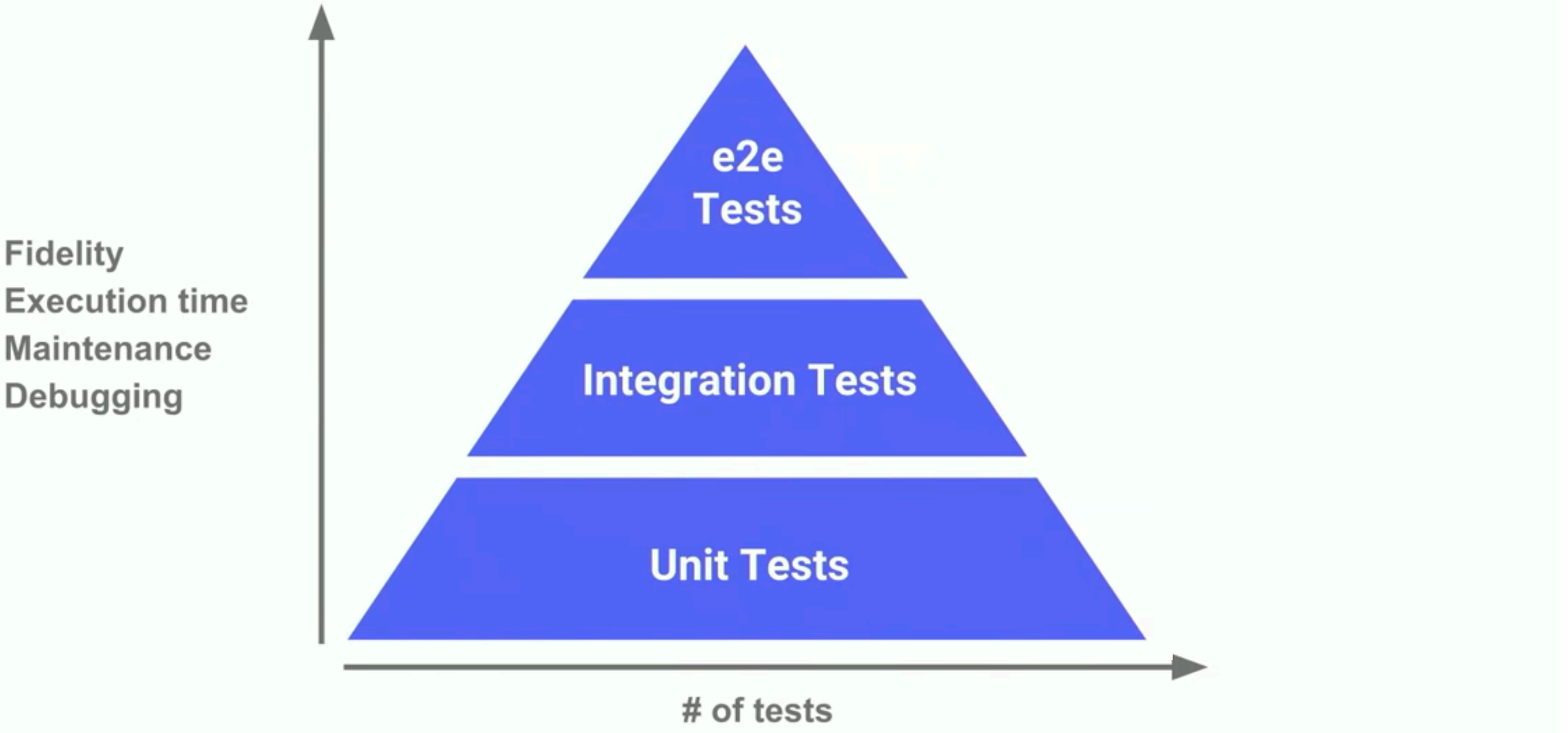
Automation Testing Levels

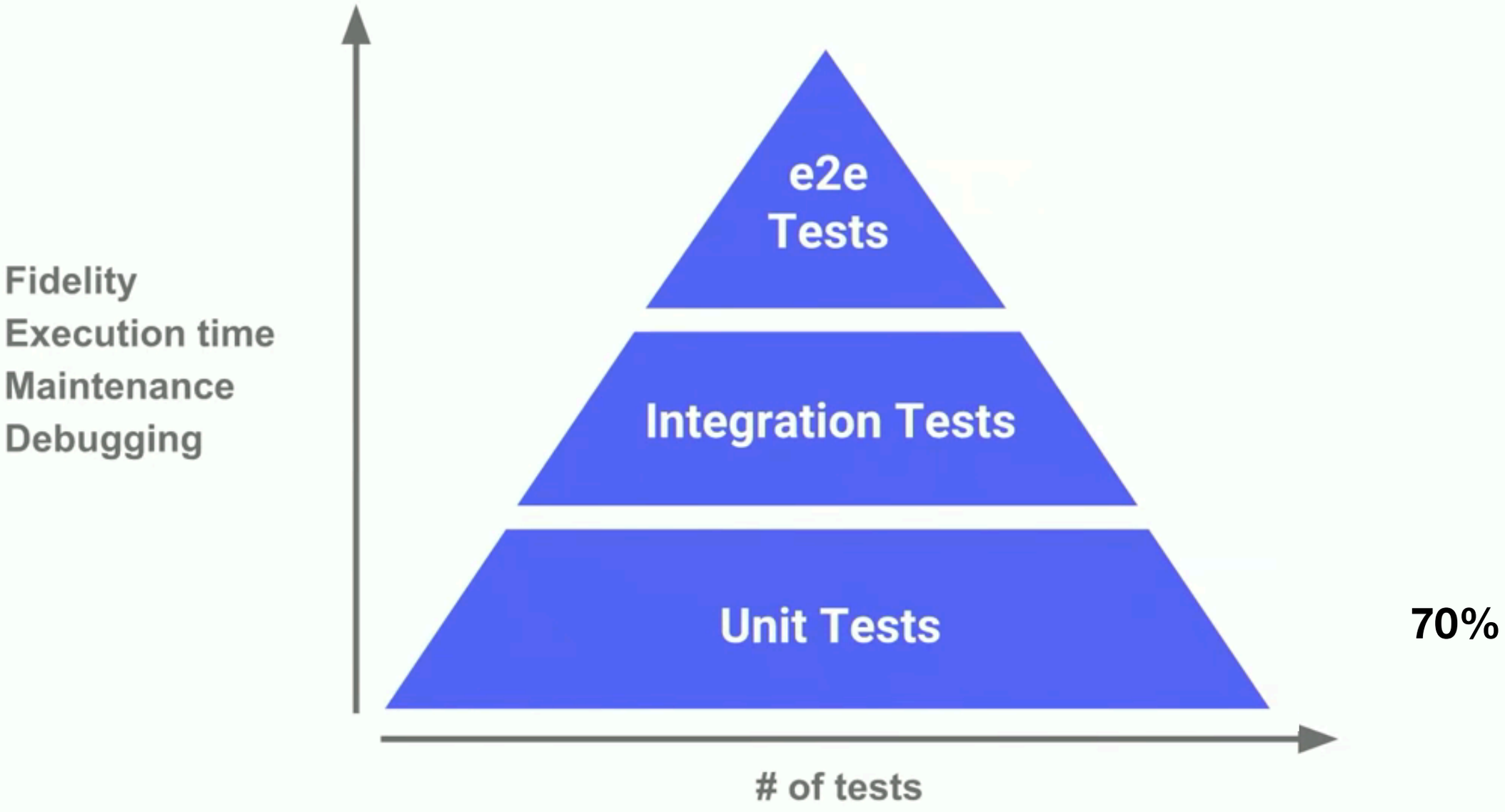


Advantages

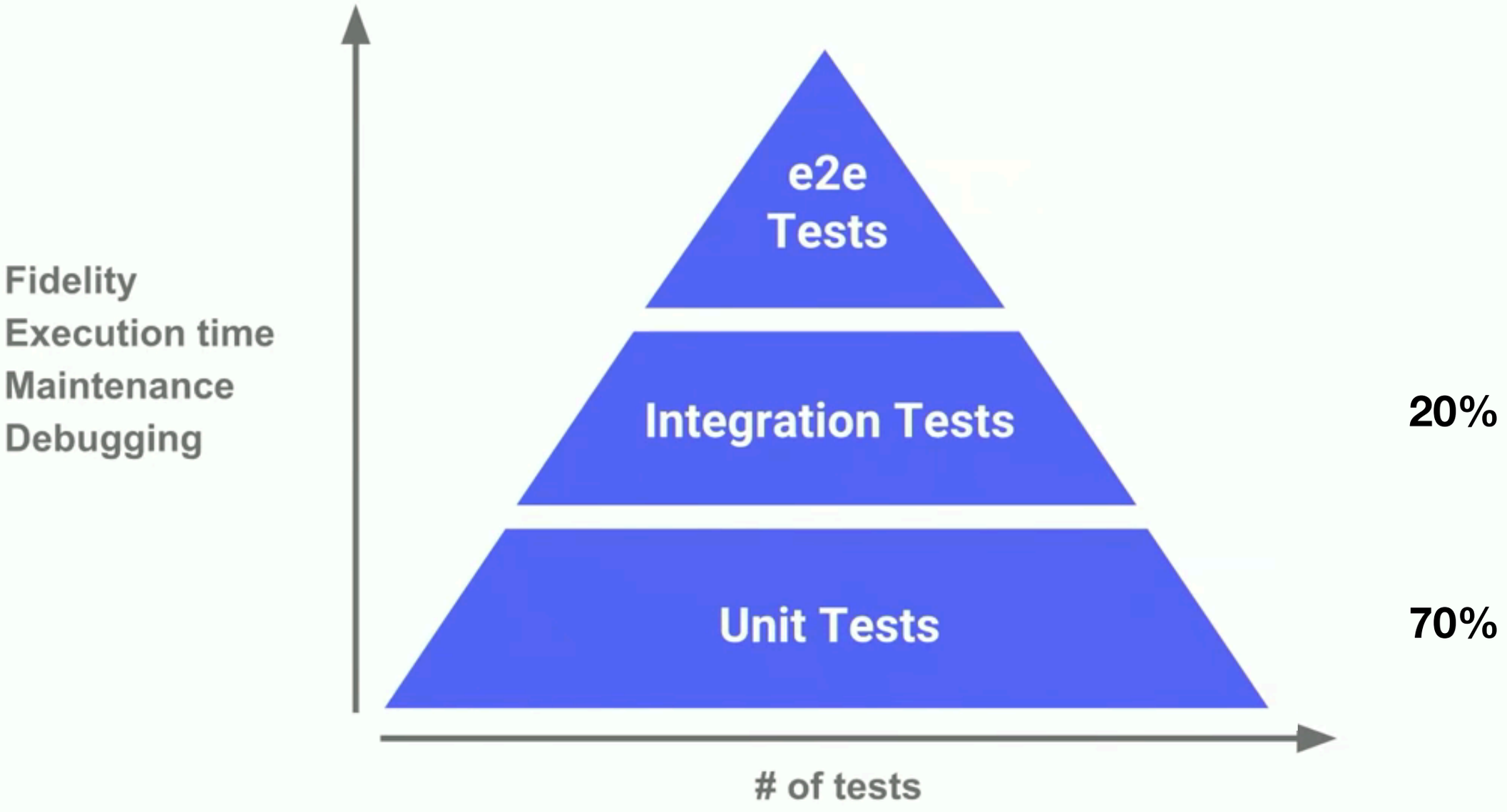
- Rapid feedback.
- Early failure detection.
- Safer code refactoring.
- Stable development velocity.

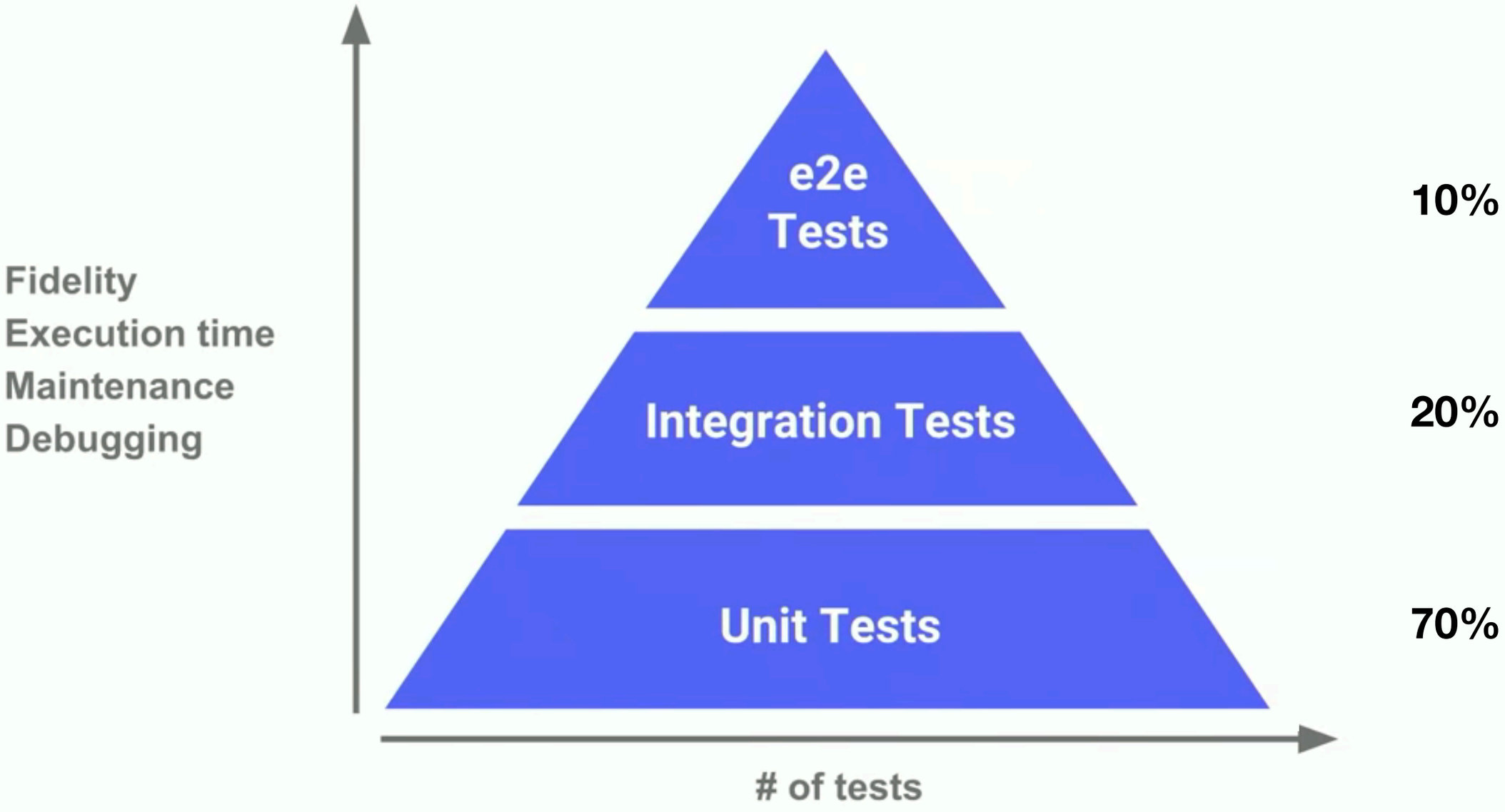

e2e
Tests
Integration Tests
Unit Tests

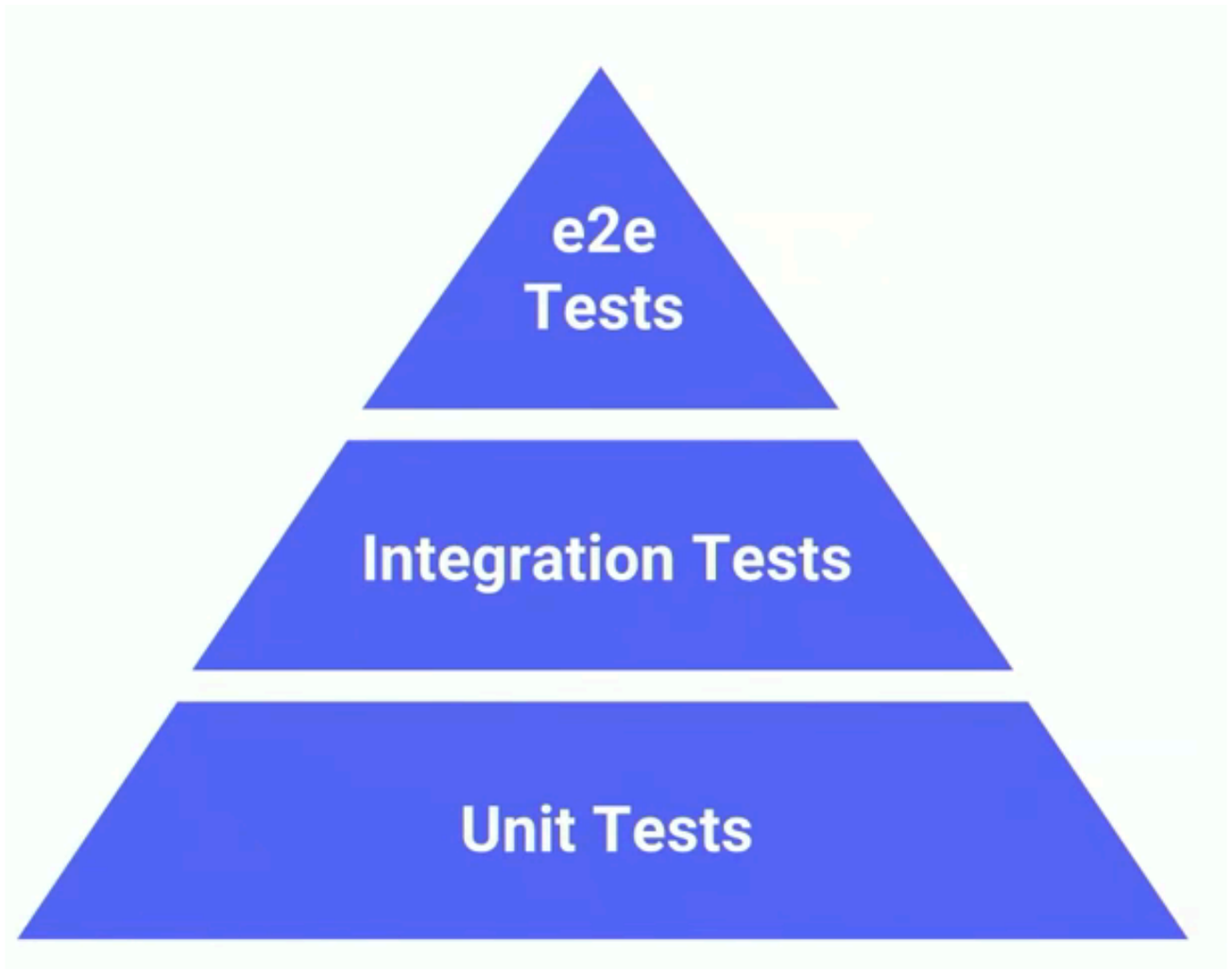

Fidelity
Execution time
Maintenance
Debugging
e2e
Tests
Integration Tests
Unit Tests
# of tests

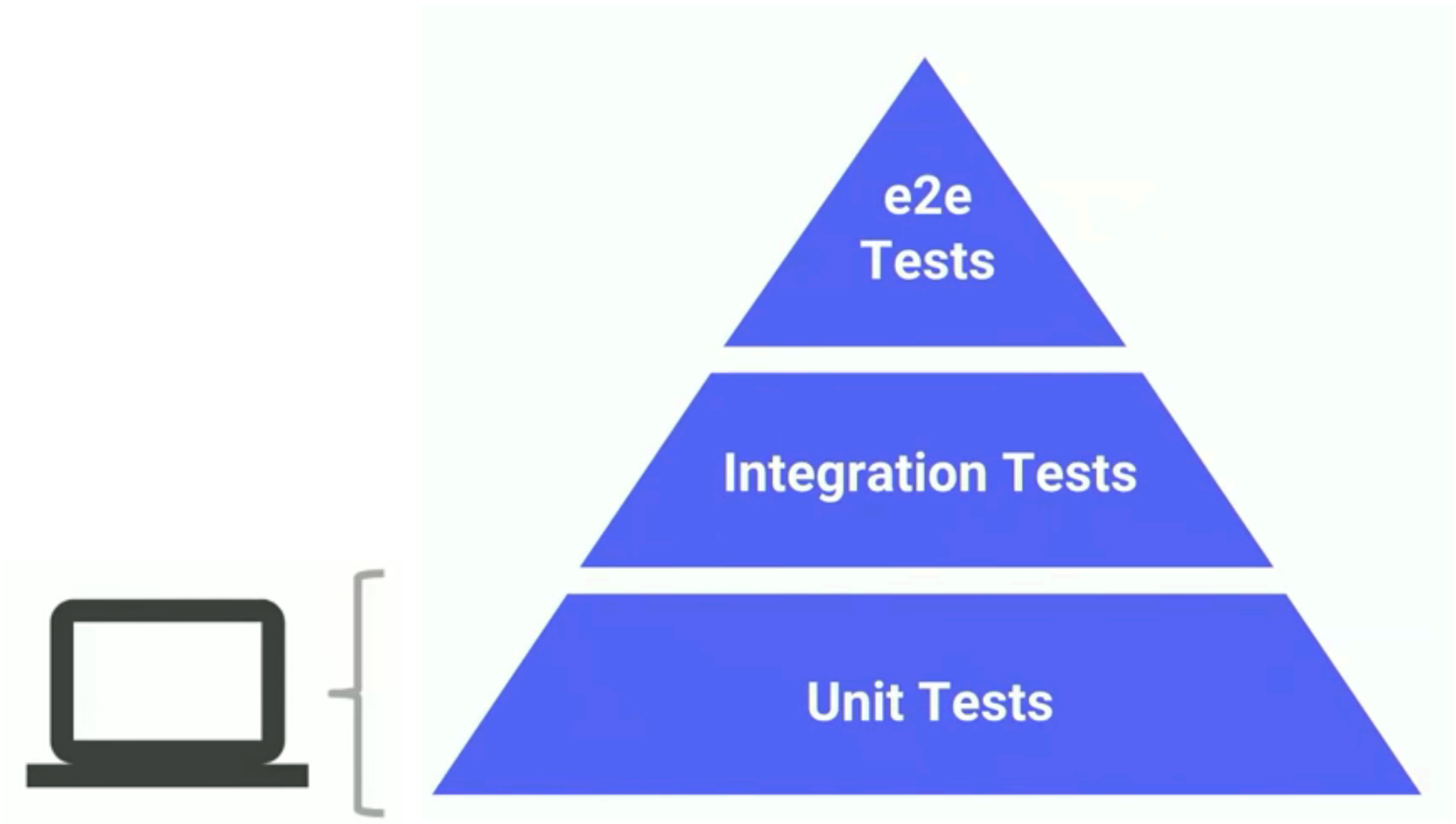

Fidelity
Execution time
Maintenance
Debugging
e2e
Tests
Integration Tests
Unit Tests
# of tests

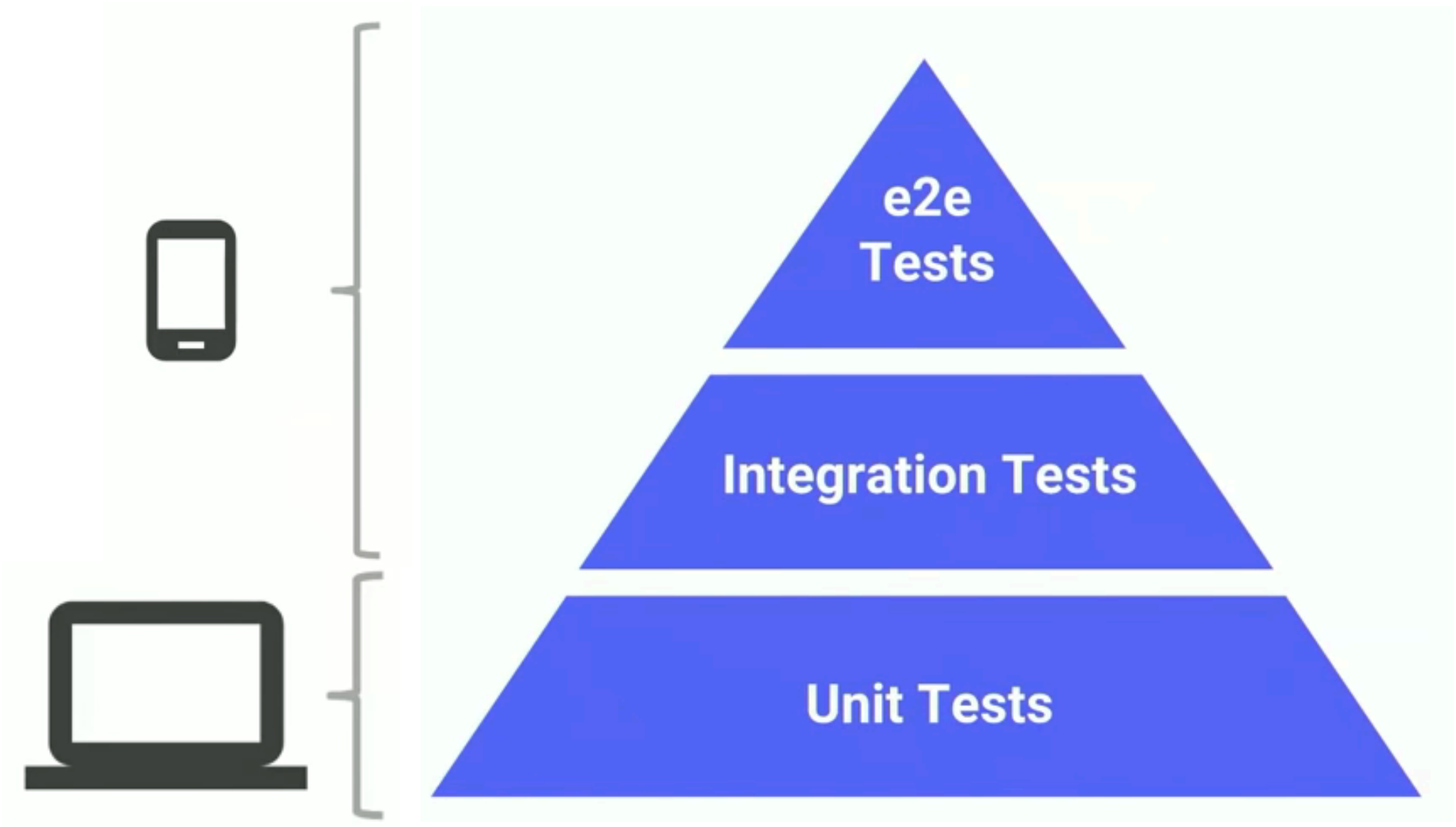
70%

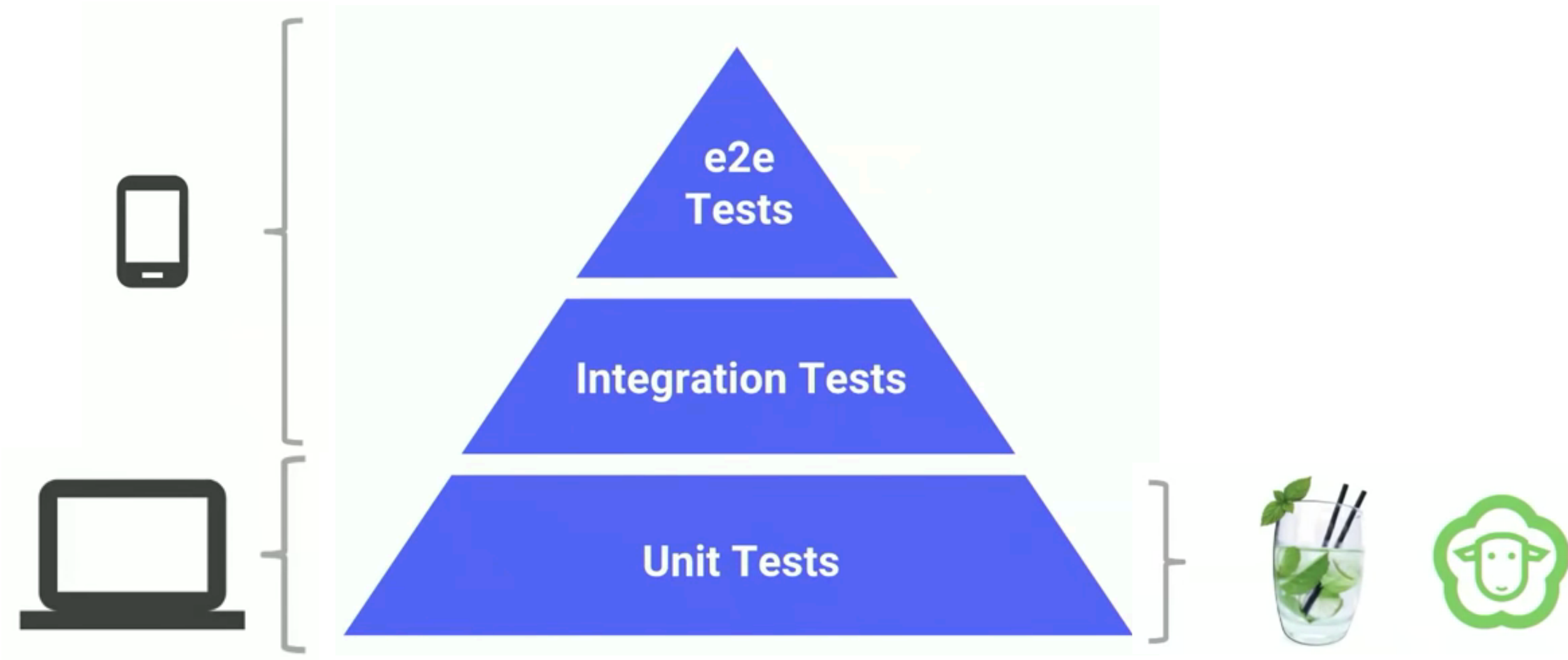

Fidelity
Execution time
Maintenance
Debugging
e2e
Tests
Integration Tests
Unit Tests
20%
70%
# of tests

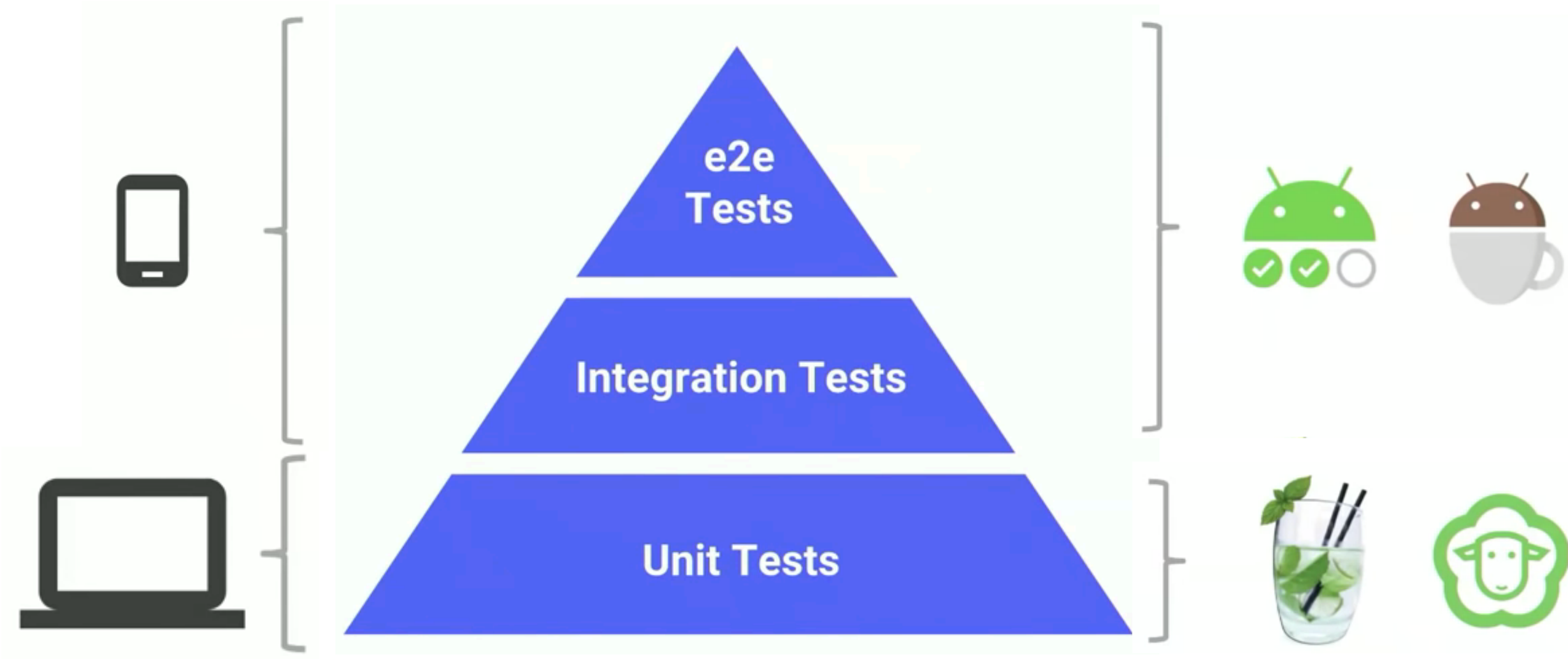

Fidelity
Execution time
Maintenance
Debugging
e2e Tests
Integration Tests
Unit Tests
10%
20%
70%
# of tests


e2e
Tests
Integration Tests
Unit Tests


e2e
Tests
Integration Tests
Unit Tests


e2e
Tests
Integration Tests
Unit Tests

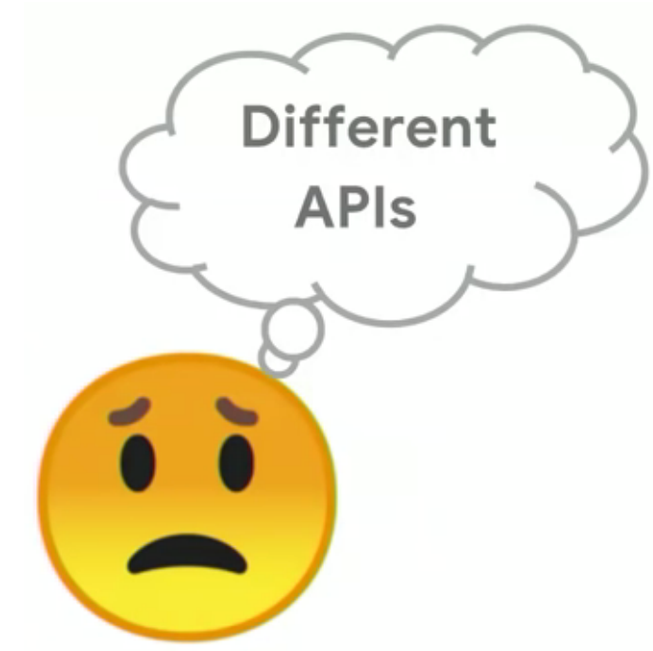

e2e
Tests
Integration Tests
Unit Tests


e2e
Tests
Integration Tests
Unit Tests

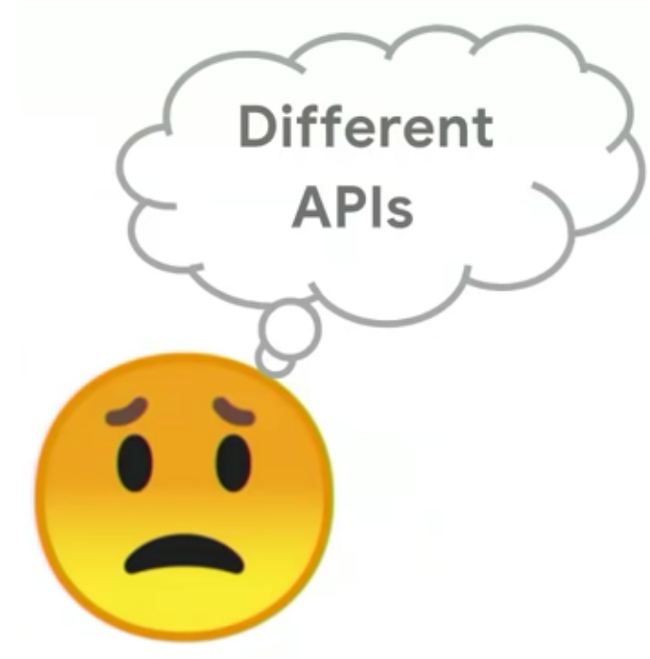
Test writing crisis

Test writing crisis

Test writing crisis



Test writing crisis



Test Structure

```
class WellStructuresTest{
    //..
    fun givenCondition_whenAction_thenShoulDo(){
        // GIVEN - Setup condition

        // WHEN - The tested action

        // THEN - An assertion to validate the action
    }
    //...
}
```

Test Structure

```
class WellStructuresTest{
    //..
    fun givenCondition_whenAction_thenShoulDo(){
        // 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_thenShoulDo(){
        // GIVEN - Setup condition

        // WHEN - The tested action

            // THEN - An assertion to validate the action
    }
    //...
}
```

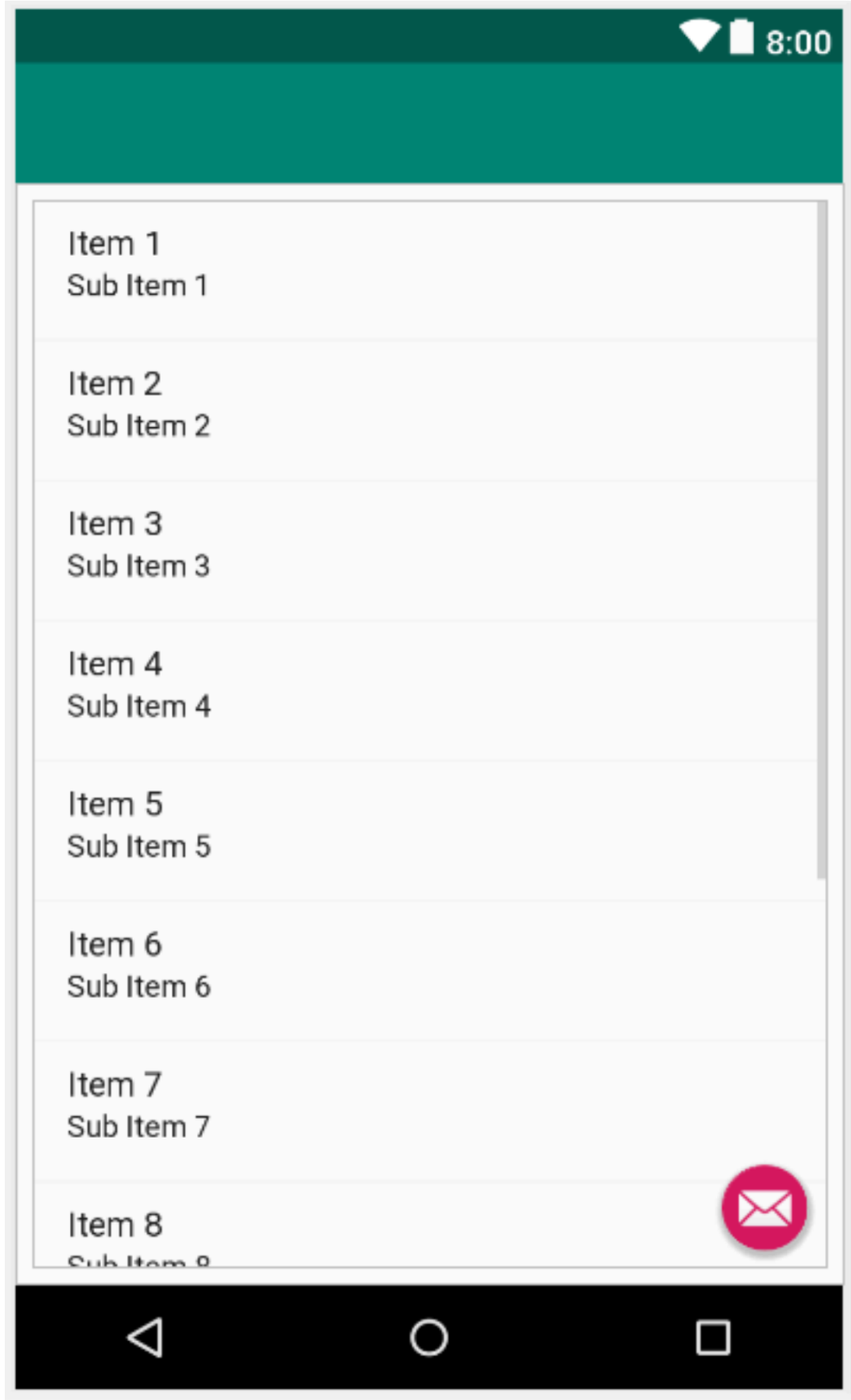
- Focus on specific behavior.
- Test behaviors independently.

Test Structure

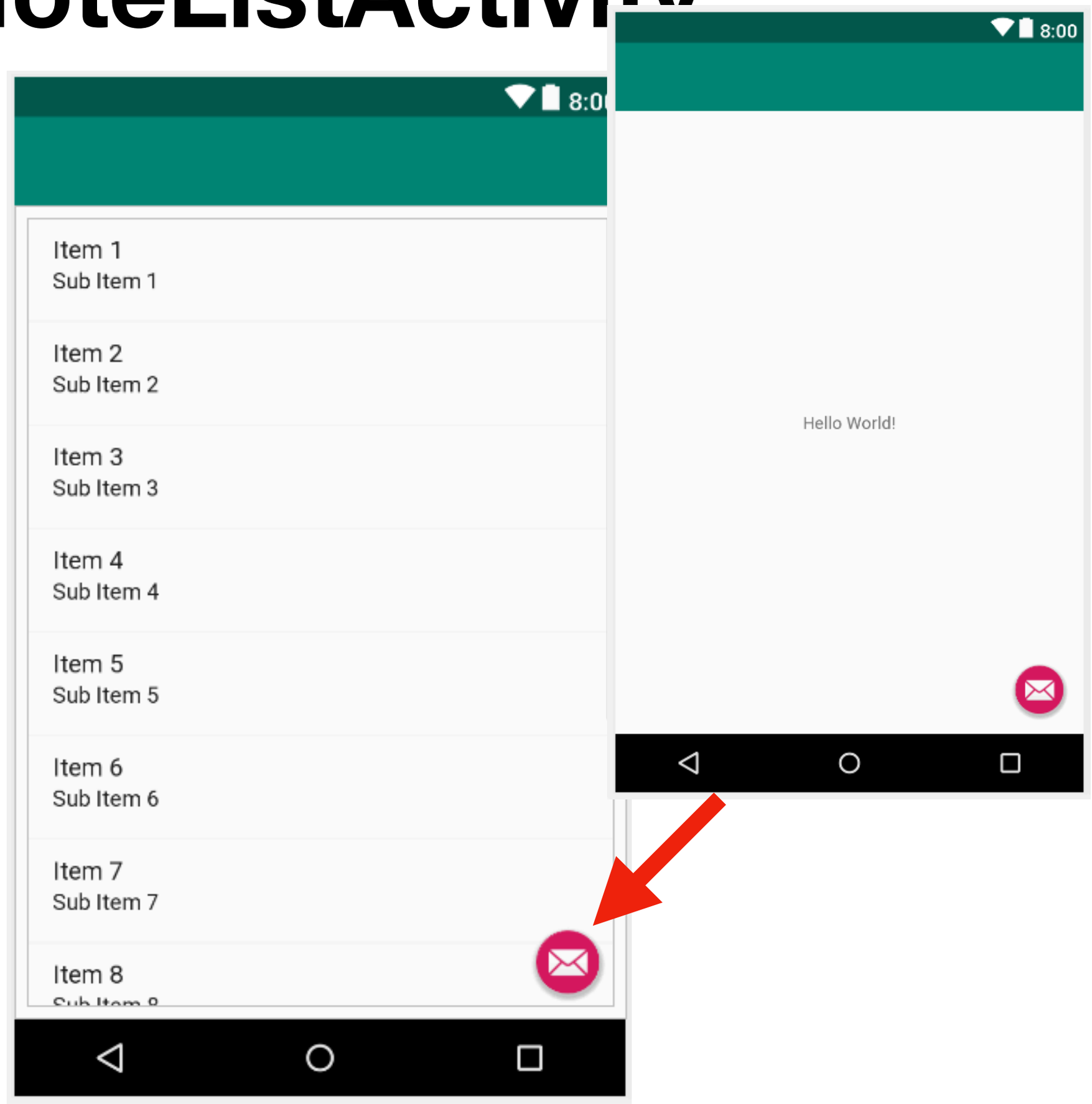
```
class WellStructuresTest{
    //..
    fun givenCondition_whenAction_thenShoulDo(){
        // GIVEN - Setup condition

        // 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())
  }
}
```

https://site.mockito.org/

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())
  }
}
```

https://site.mockito.org/

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())
  }
}
```

https://site.mockito.org/

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())
  }
}
```

https://site.mockito.org/

Robolectic

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

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

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

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

http://robolectric.org/

Robolectic

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

  @Test
  fun testTitle() {
    val activity =
      Roboelectic.setupActivity(NoteListActivity::class.java)
    ShadowView.clickOn(activity.findViewById(R.id.title))

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

http://robolectric.org/

Robolectic

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

  @Test
  fun testTitle() {
    val activity =
      Roboelectic.setupActivity(NoteListActivity::class.java)
      ShadowView.clickOn(activity.findViewById(R.id.title))

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

http://robolectric.org/

Robolectic

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

  @Test
  fun testTitle() {
    val activity =
      Roboelectic.setupActivity(NoteListActivity::class.java)
      ShadowView.clickOn(activity.findViewById(R.id.title))

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

http://robolectric.org/

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")))
  }
}
```

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")))
  }
}
```

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")))
  }
}
```

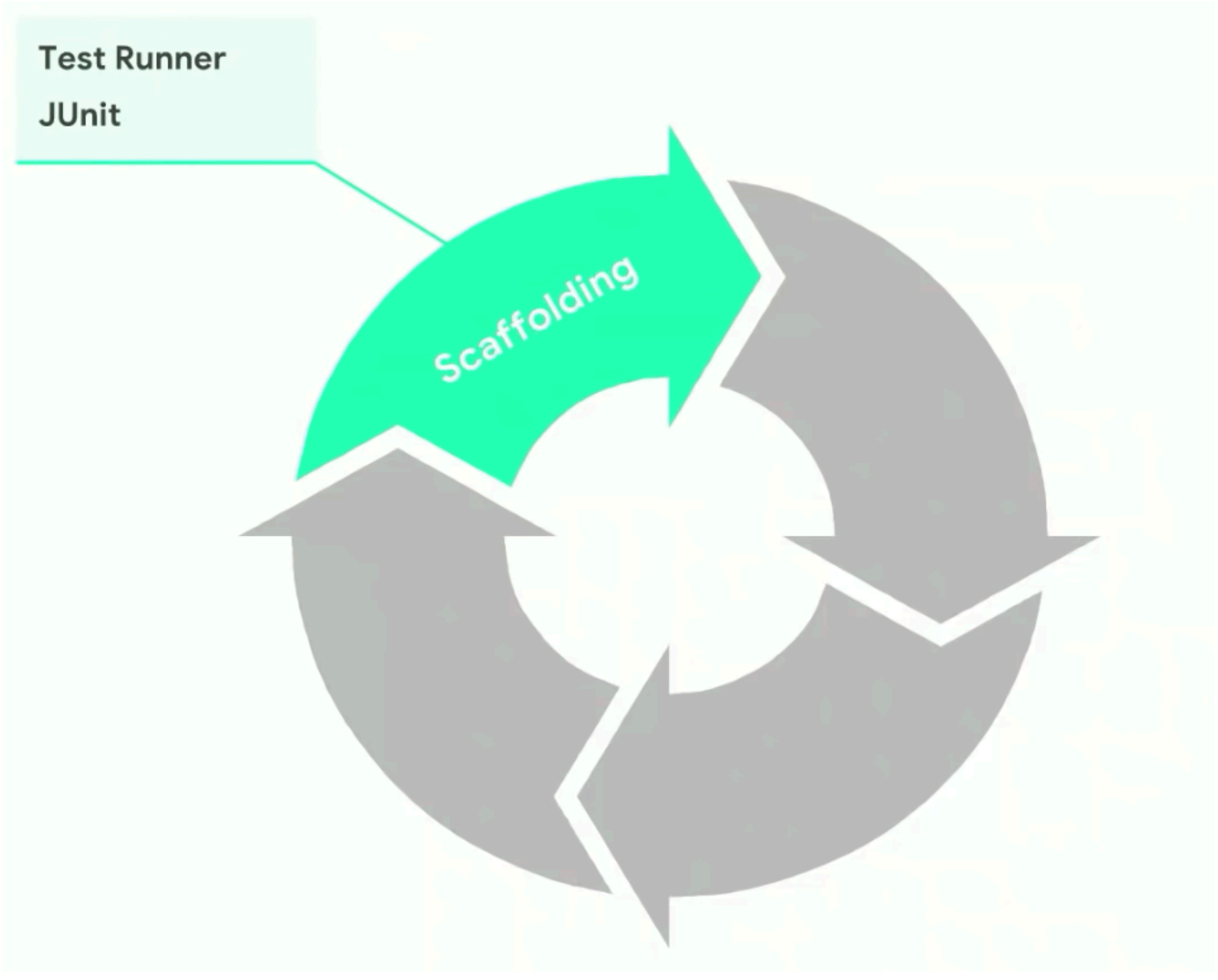
https://developer.android.com/training/testing/espresso/


THE OTHER WAY
THIS WAY
THAT WAY

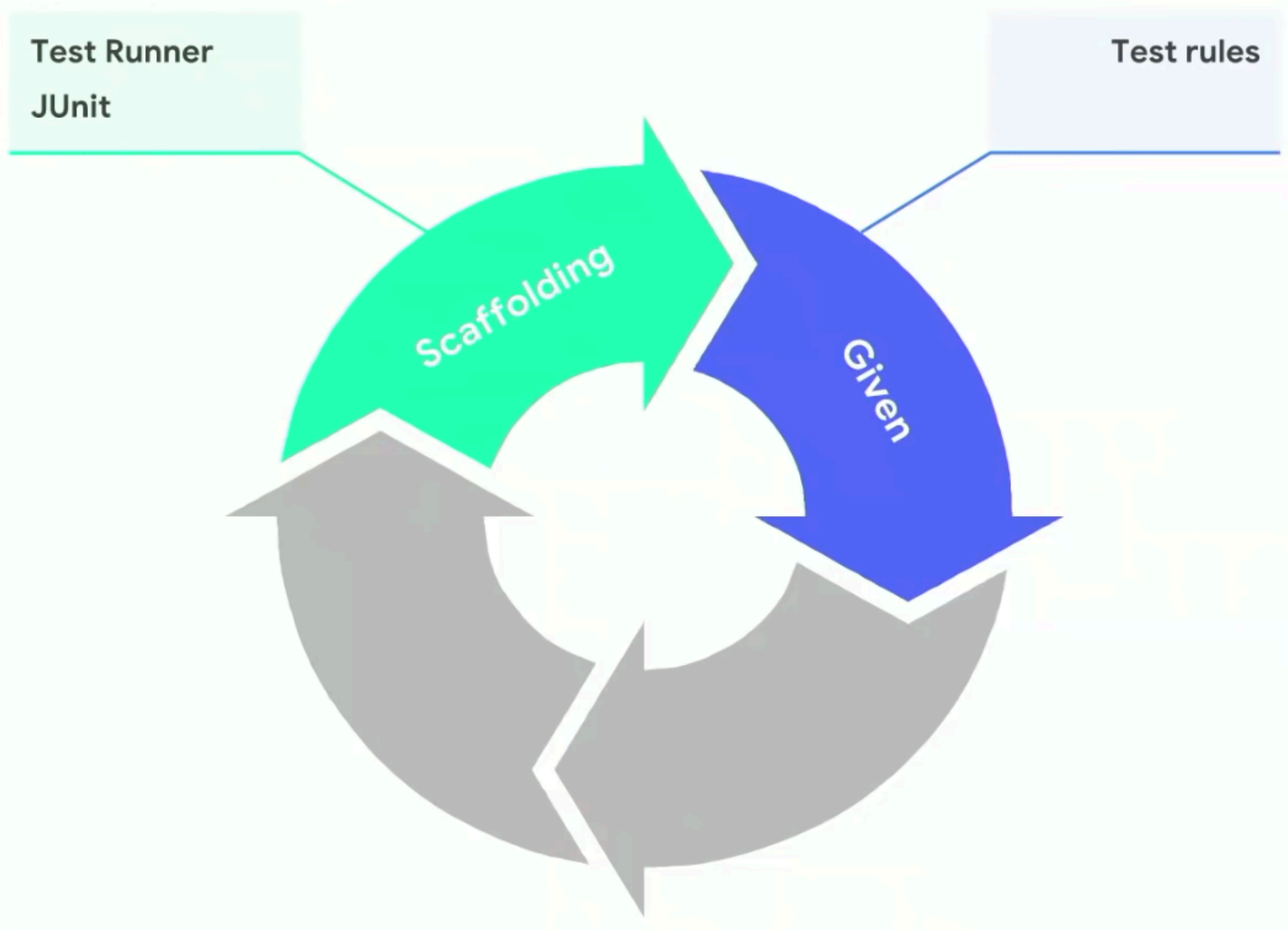
Android Test

Part of Jetpack

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

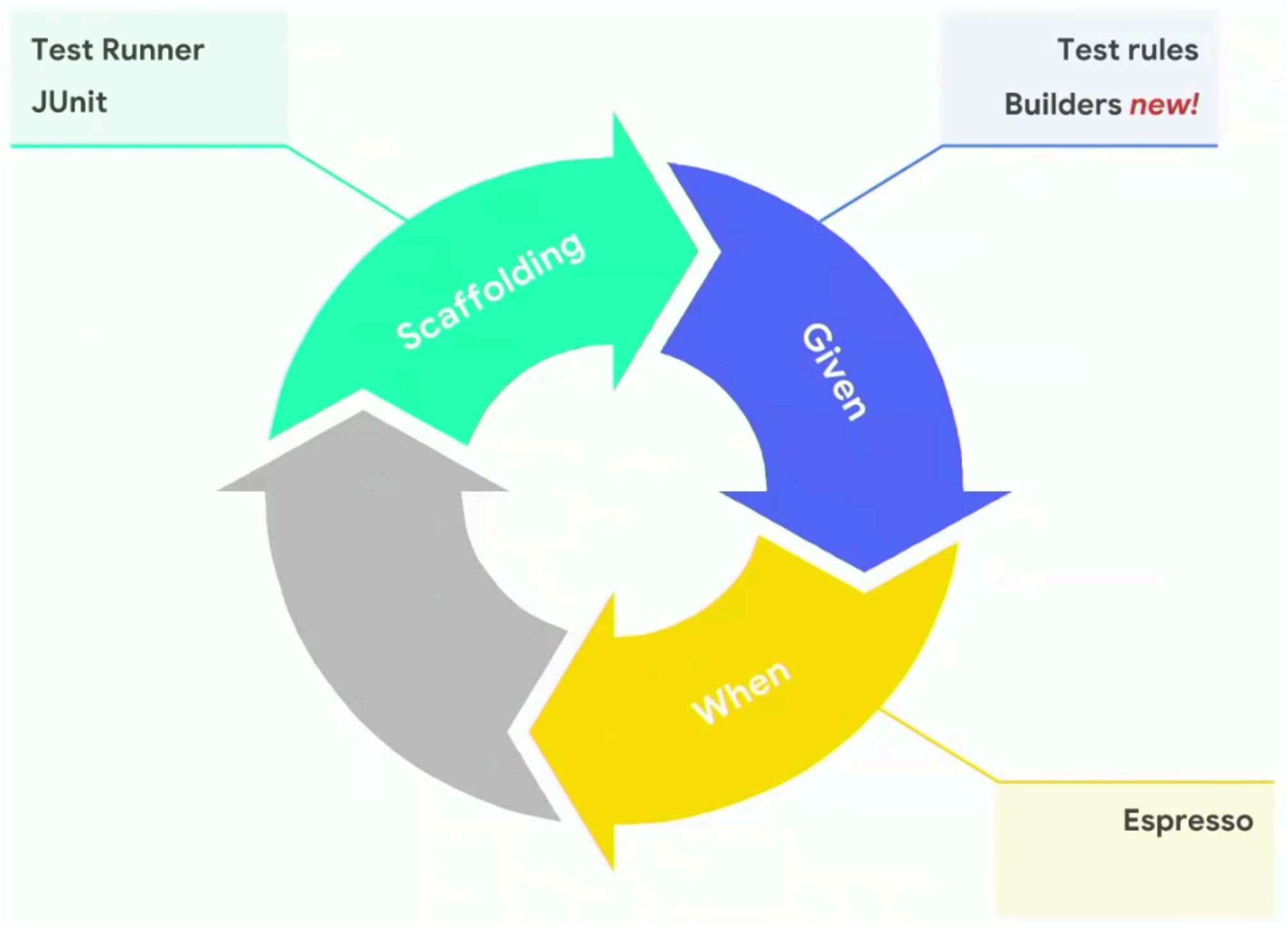

Test Runner
JUnit
Scaffolding

```
//SCAFFOLDING

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

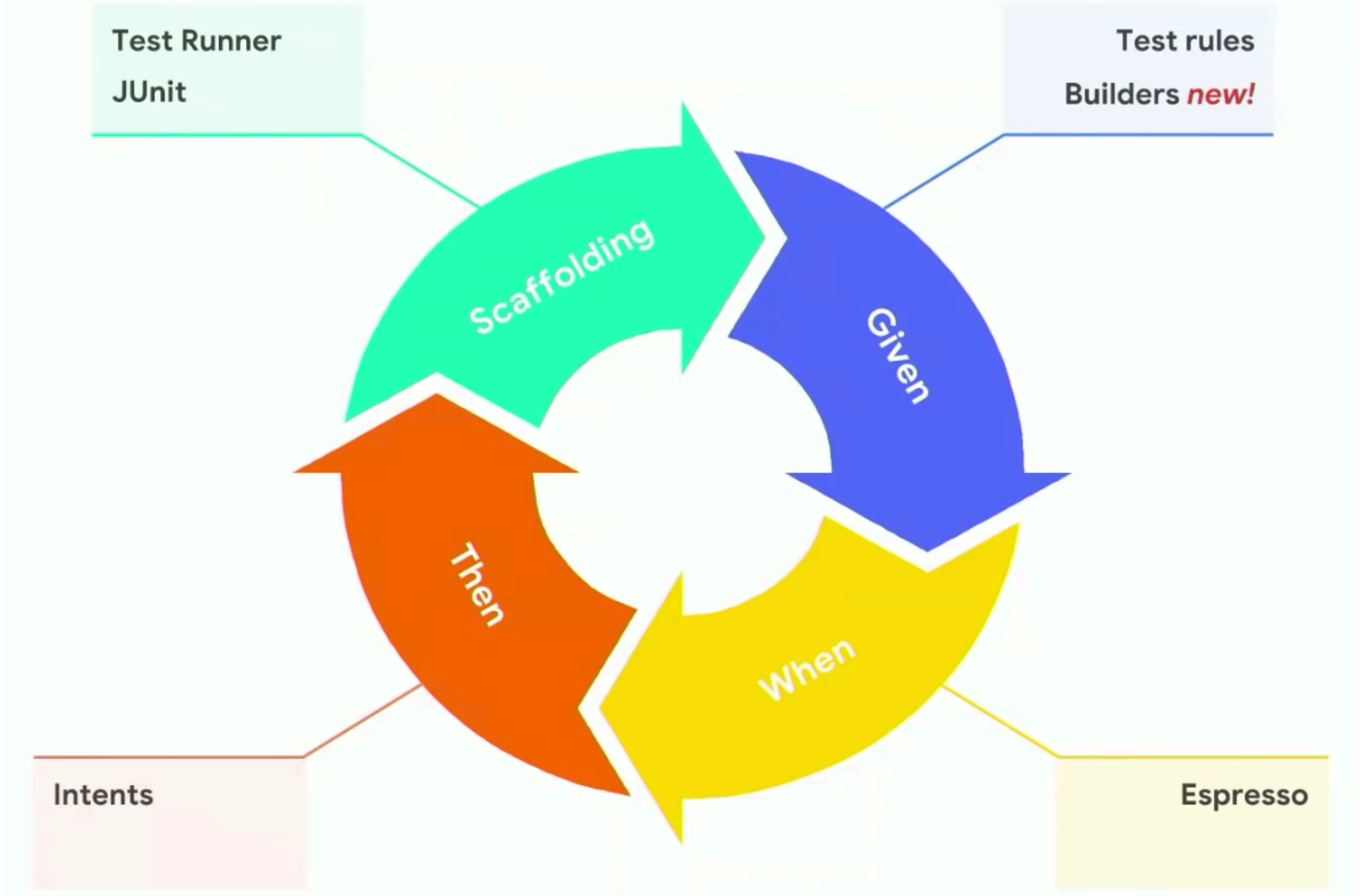

Test Runner
JUnit
Test rules
Scaffolding
Given

```
//GIVEN

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

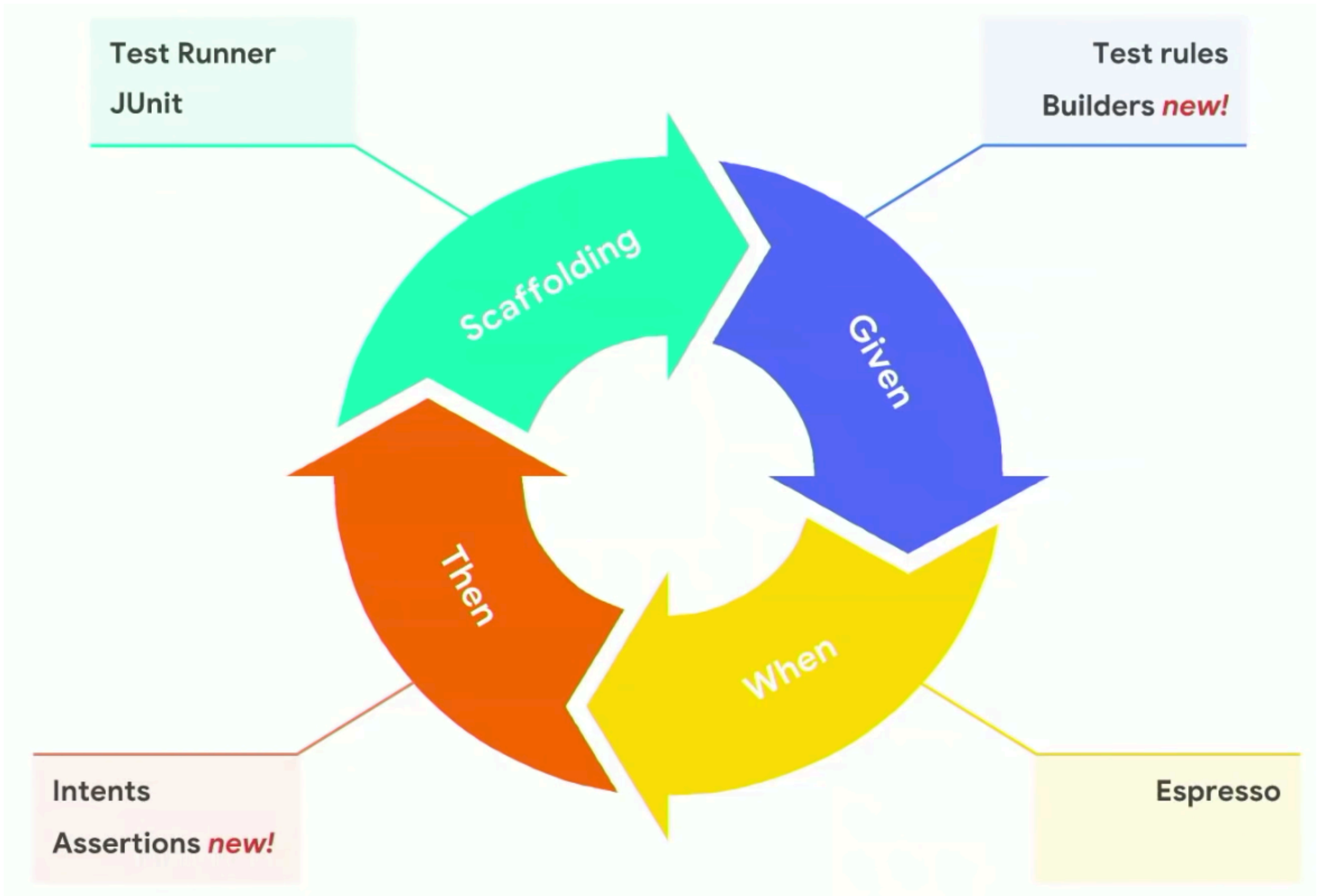

```
//GIVEN

@RunWith(AndroidJUnit4::class)
class SimpleUnifiedTest {
  @Test
  fun testMotionEvents() {
    val motionEvent =

buildMotionEvent().setAction(MotionEvent.ACTION_DOWN)
  }
}
```


Test Runner
JUnit
Test rules
Builders new!
Scaffolding
Given
When
Espresso

```
//WHEN

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

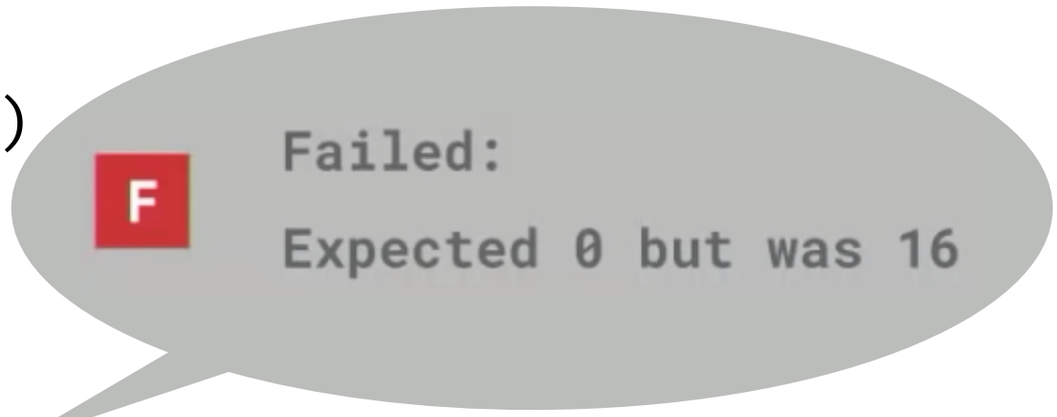

Test Runner
JUnit
Test rules
Builders new!
Scaffolding
Given
When
Then
Intents
Espresso

```
//THEN

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


Test Runner
JUnit
Test rules
Builders new!
Scaffolding
Given
When
Then
Intents
Assertions new!
Espresso

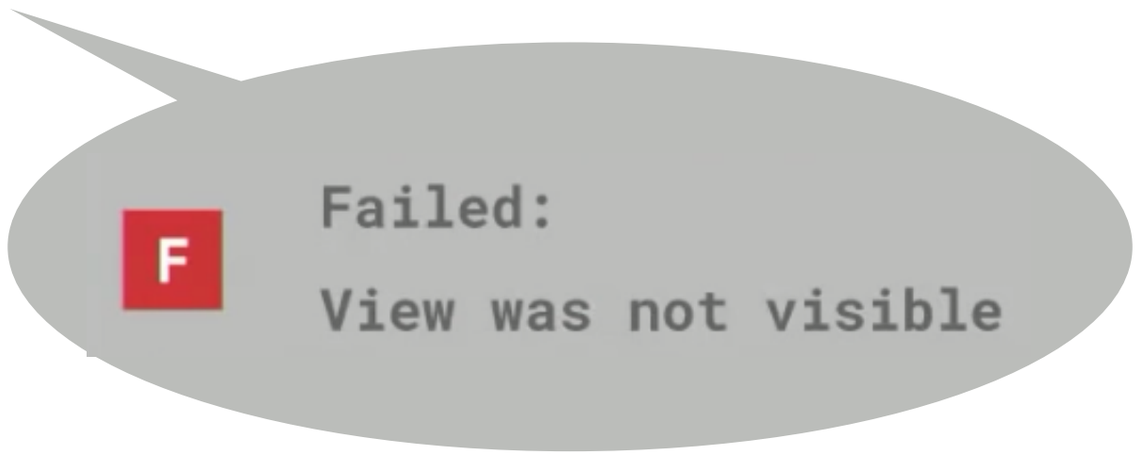
```
//THEN

@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)
  }
}
```

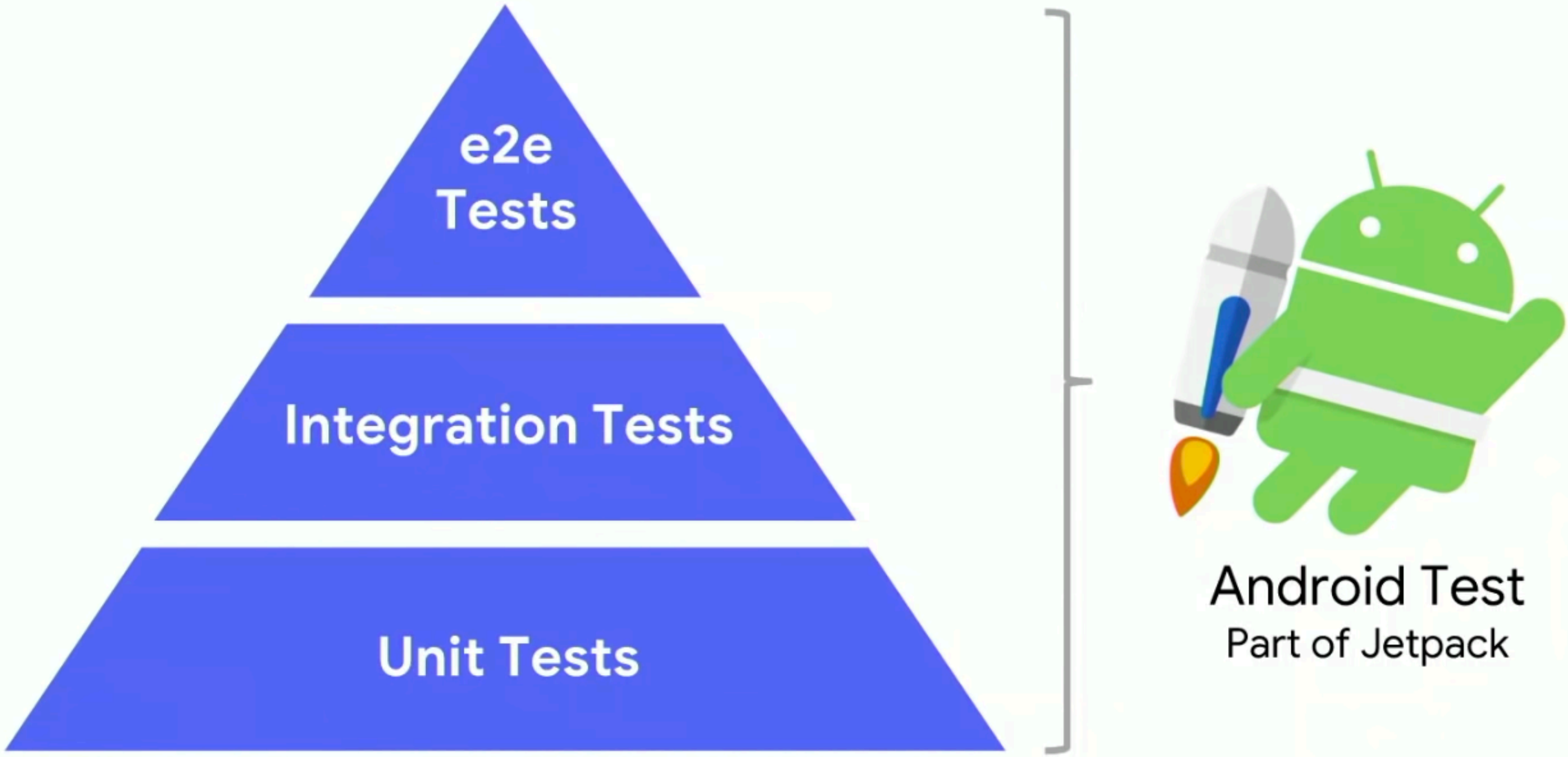



```
//THEN

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


```
//THEN

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



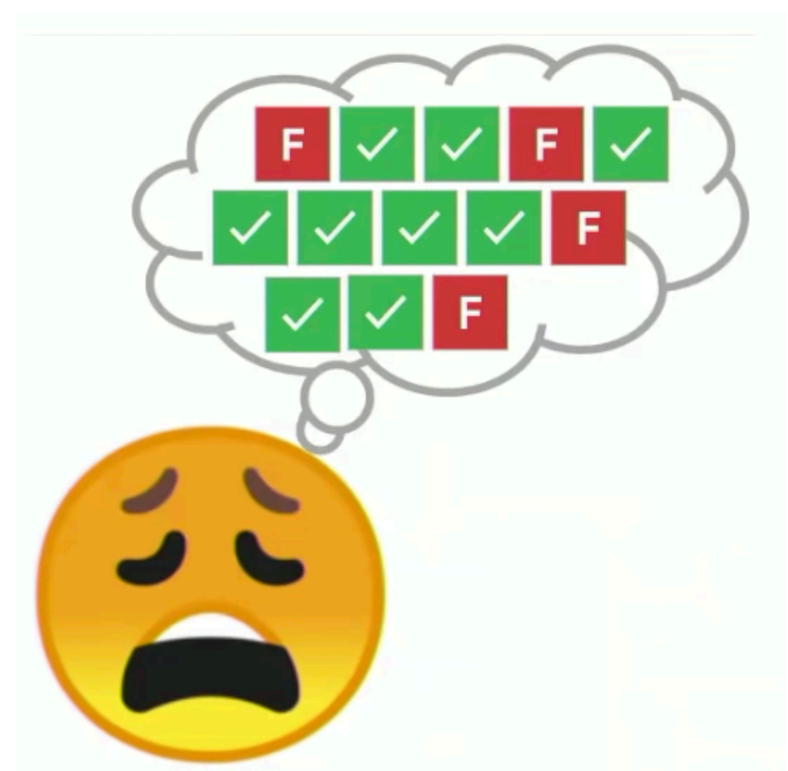
https://developer.android.com/training/testing/


e2e
Tests
Integration Tests
Unit Tests
Android Test
Part of Jetpack

Test execution crisis

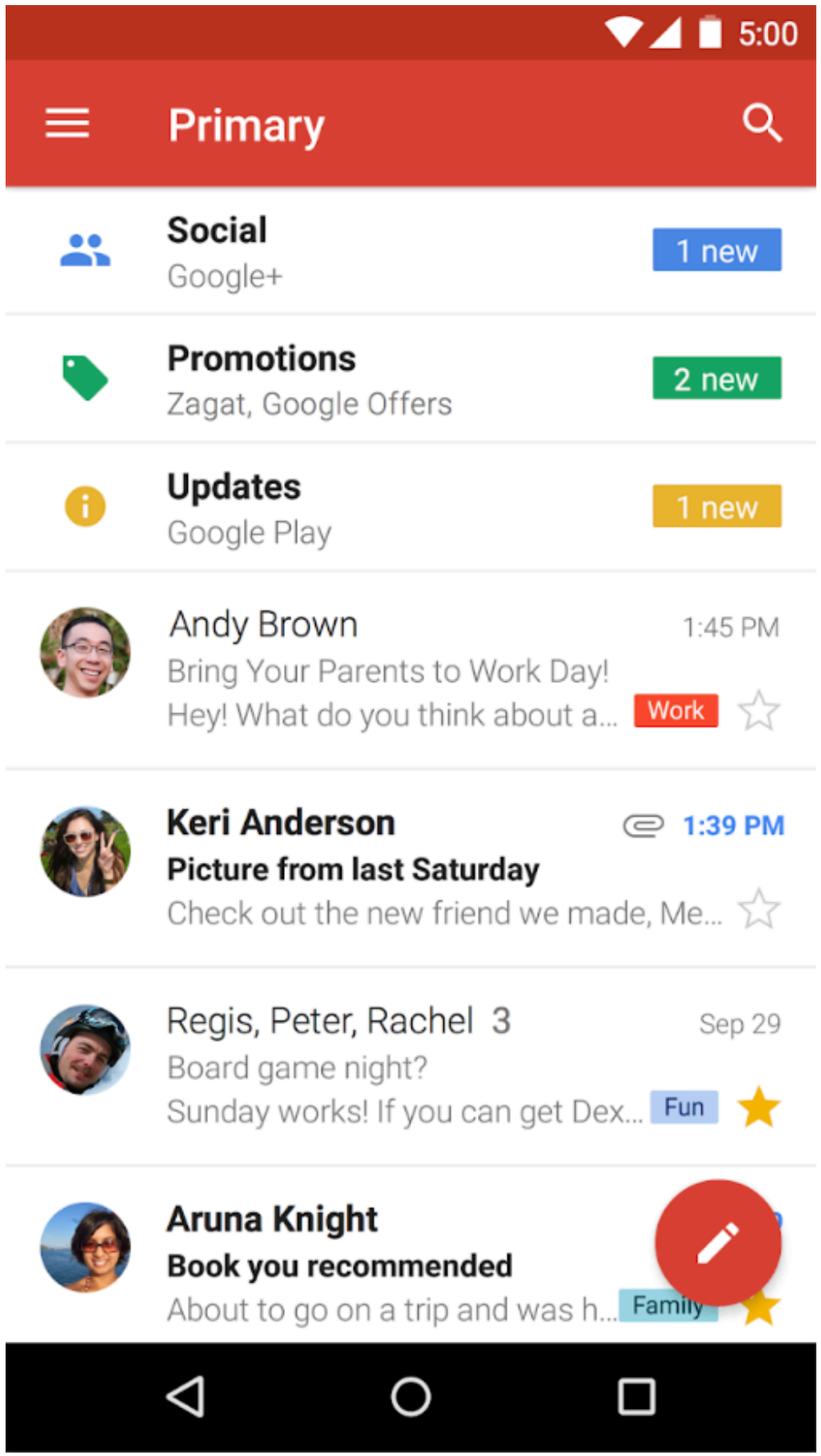
Test execution crisis

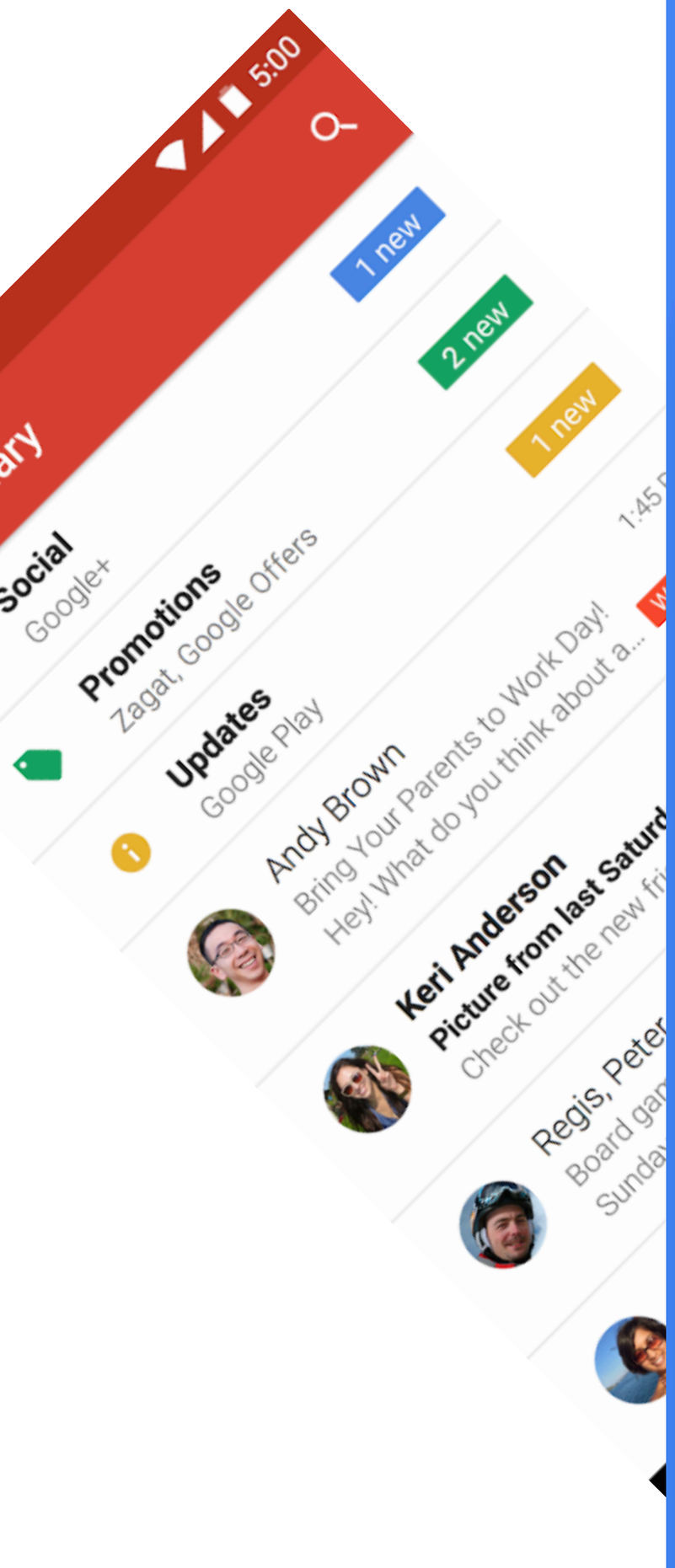
Test execution crisis



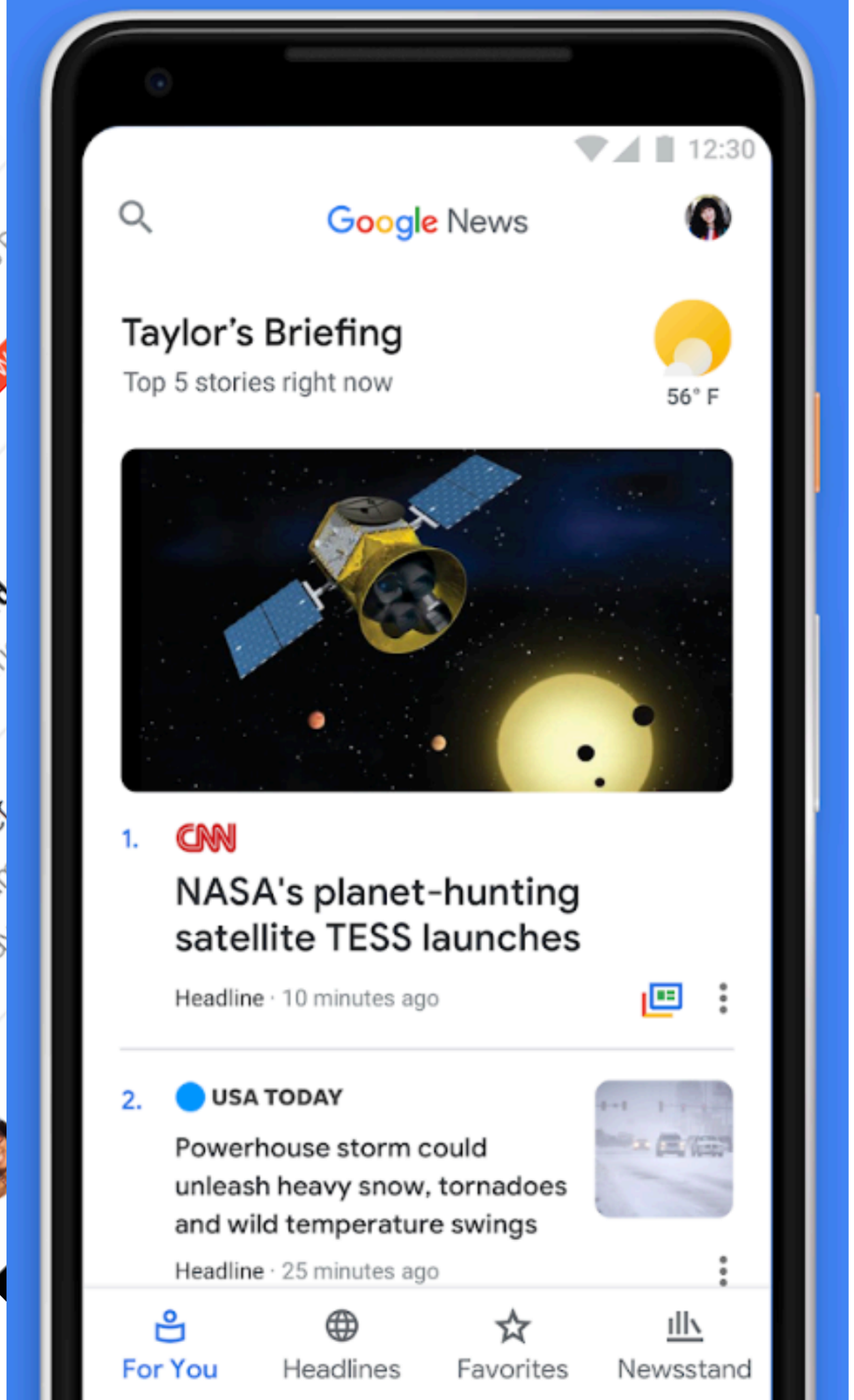
Project Nitrogen

A single entry point for tests

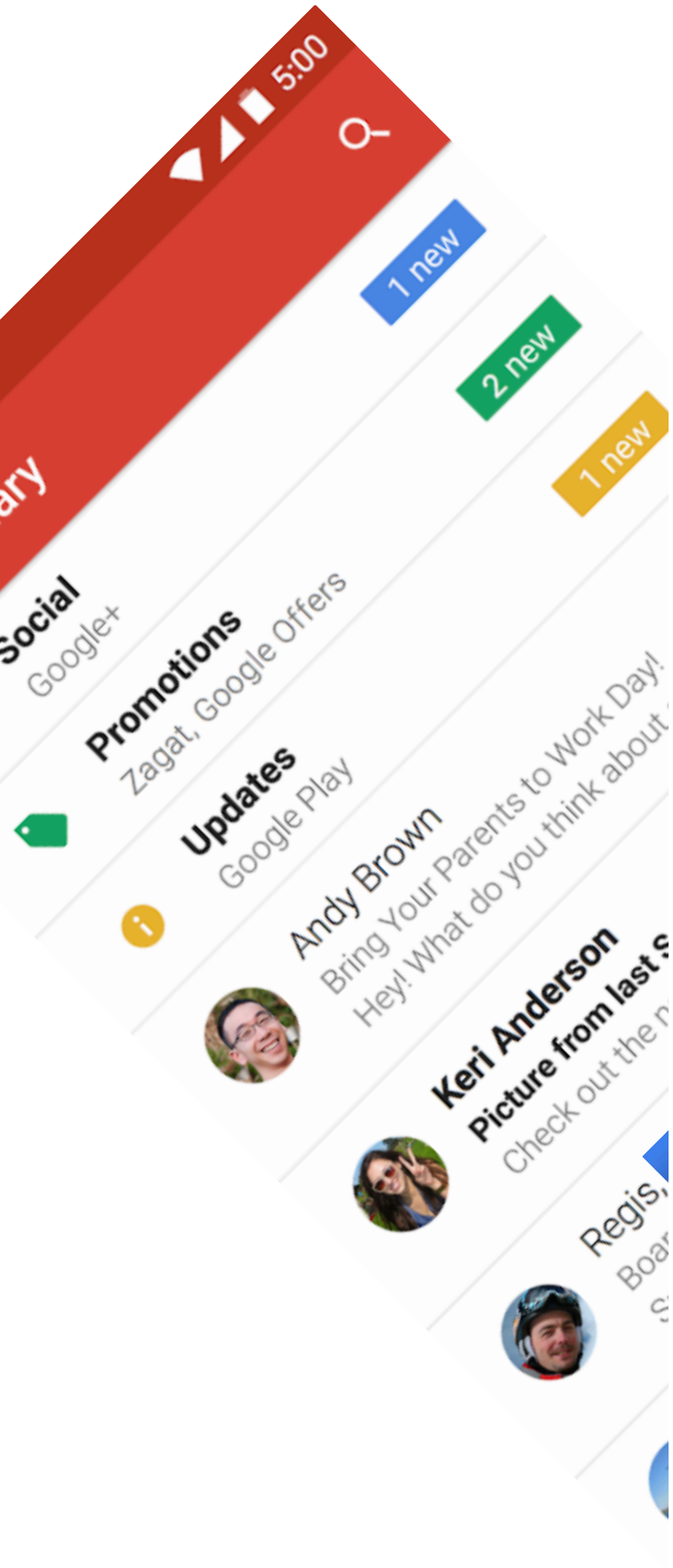

5:00
Primary
Social
Google+
1 new
Promotions
Zagat, Google Offers
2 new
Updates
Google Play
1 new
Andy Brown
1:45 PM
Bring Your Parents to Work Day!
Hey! What do you think about a...
Work
Keri Anderson
1:39 PM
Picture from last Saturday
Check out the new friend we made, Me...
Regis, Peter, Rachel 3
Sep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

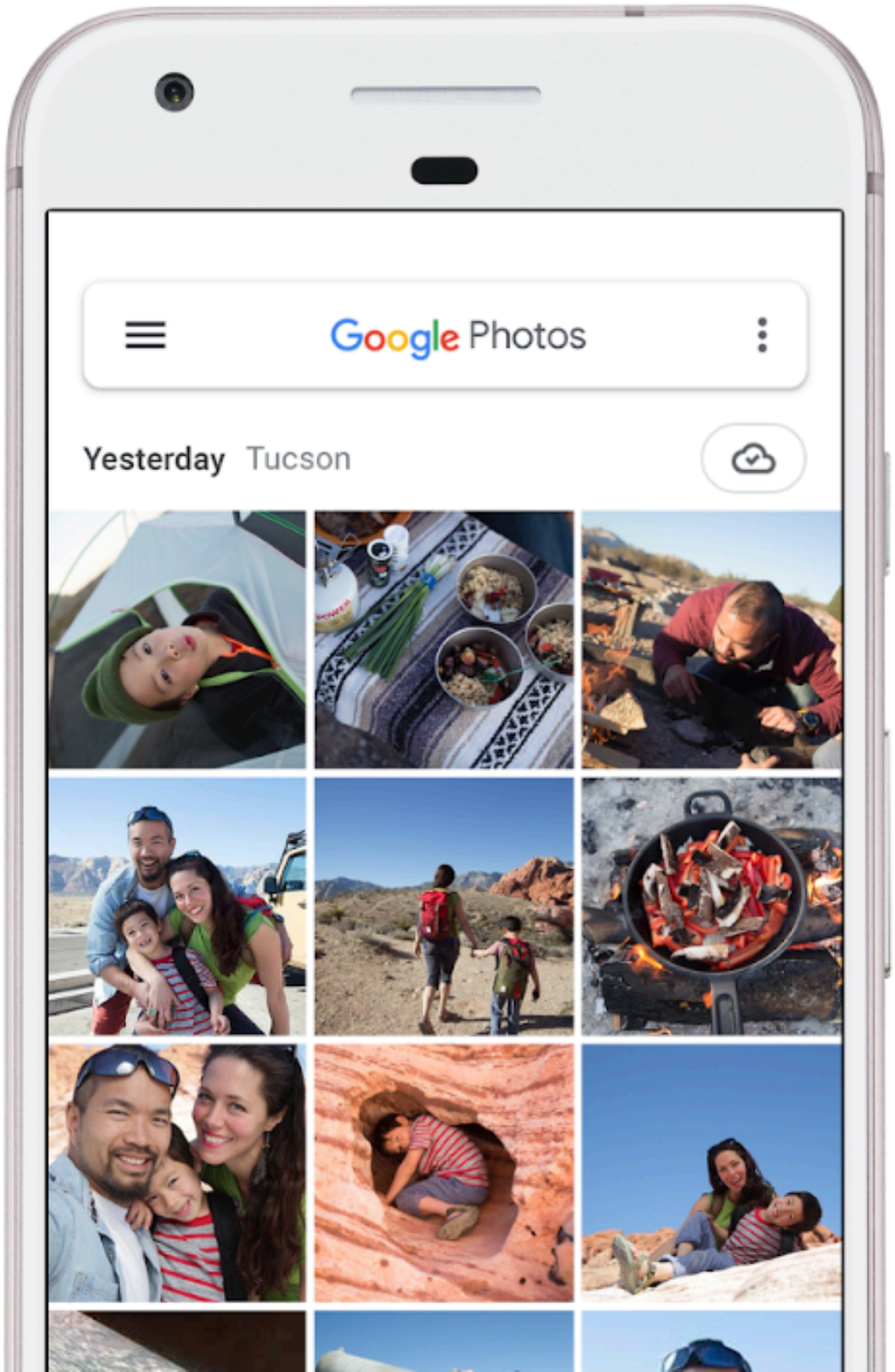


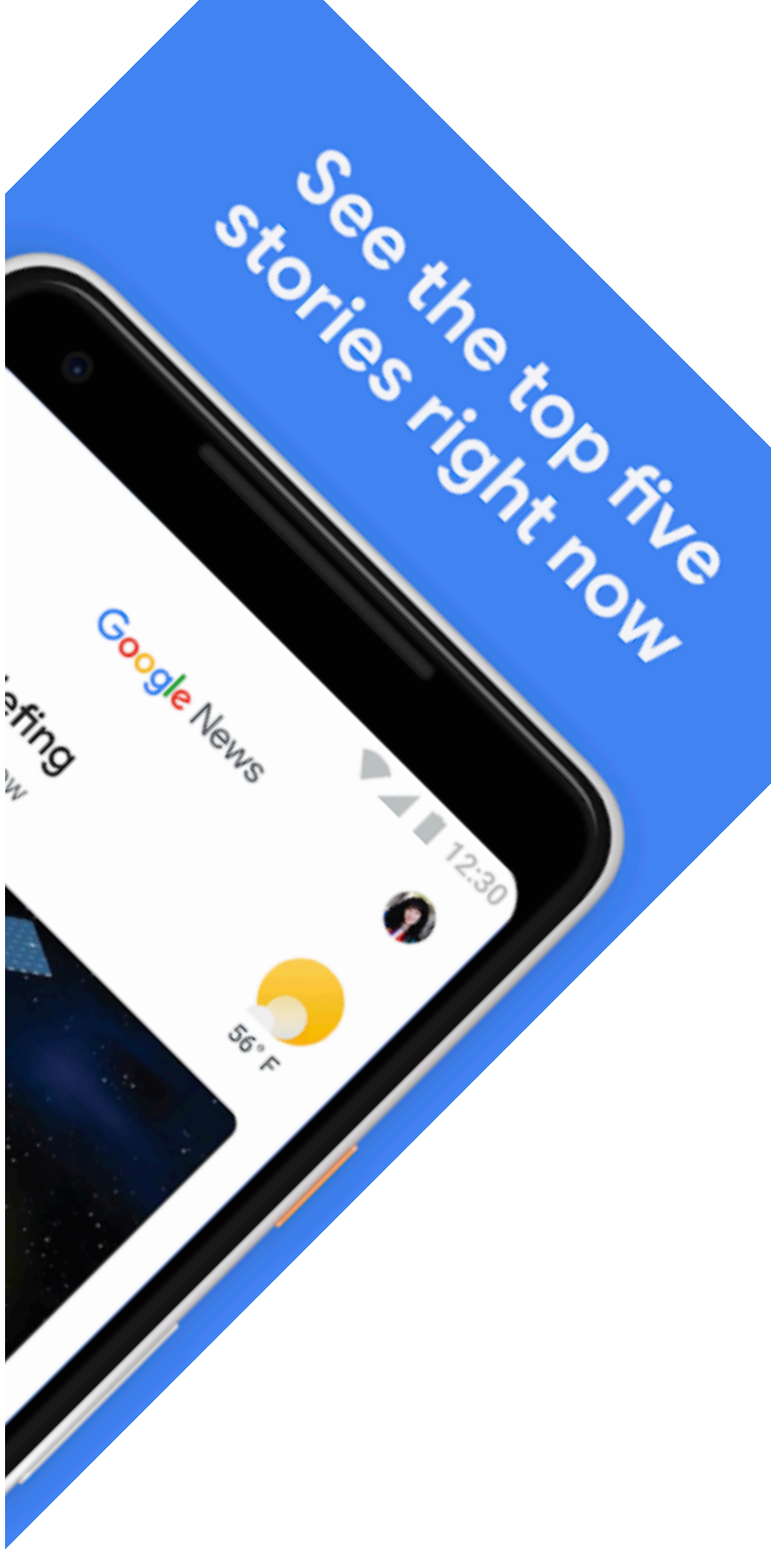
See the top five stories right now

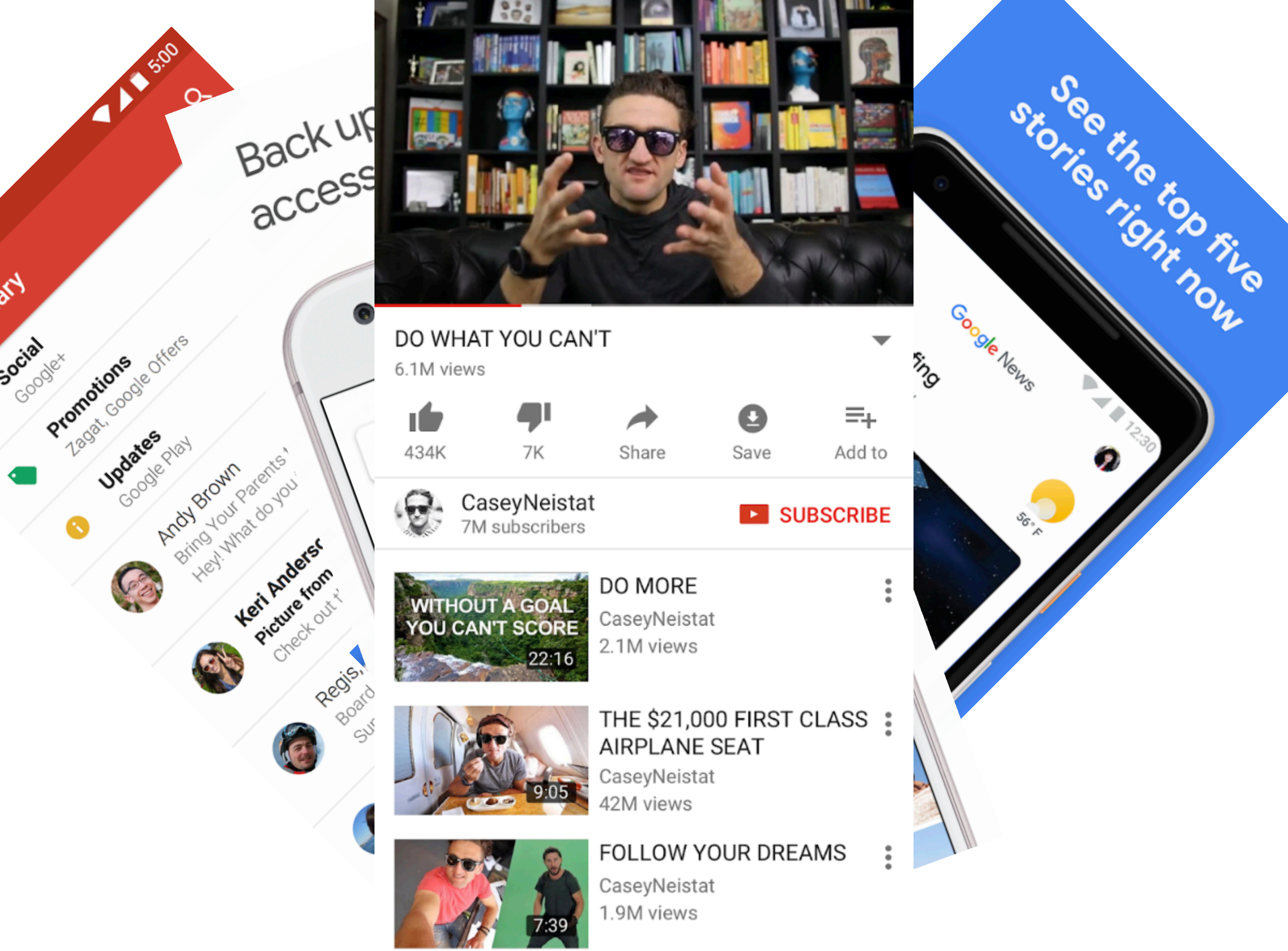


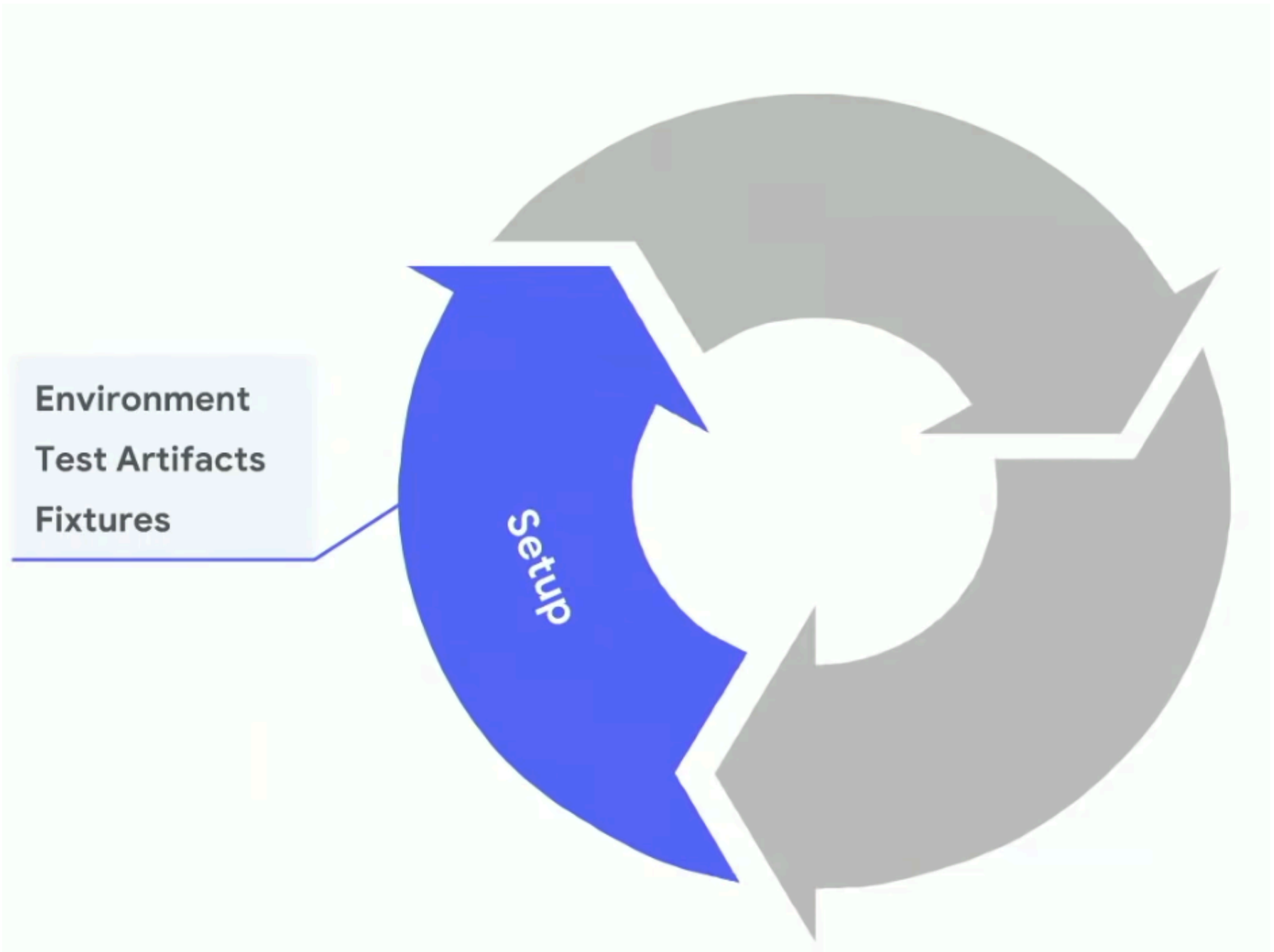
Back up your photos & access them anywhere

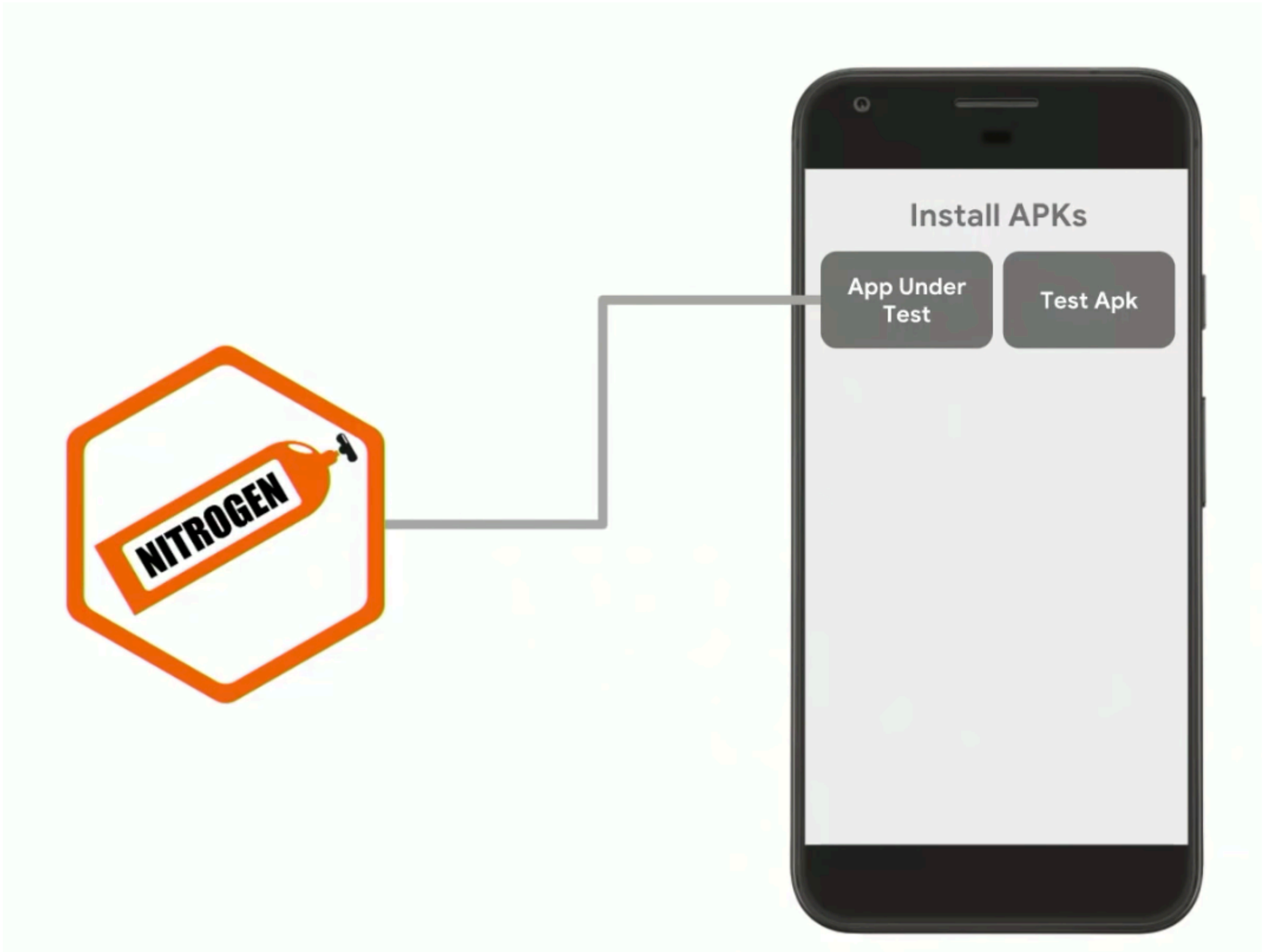


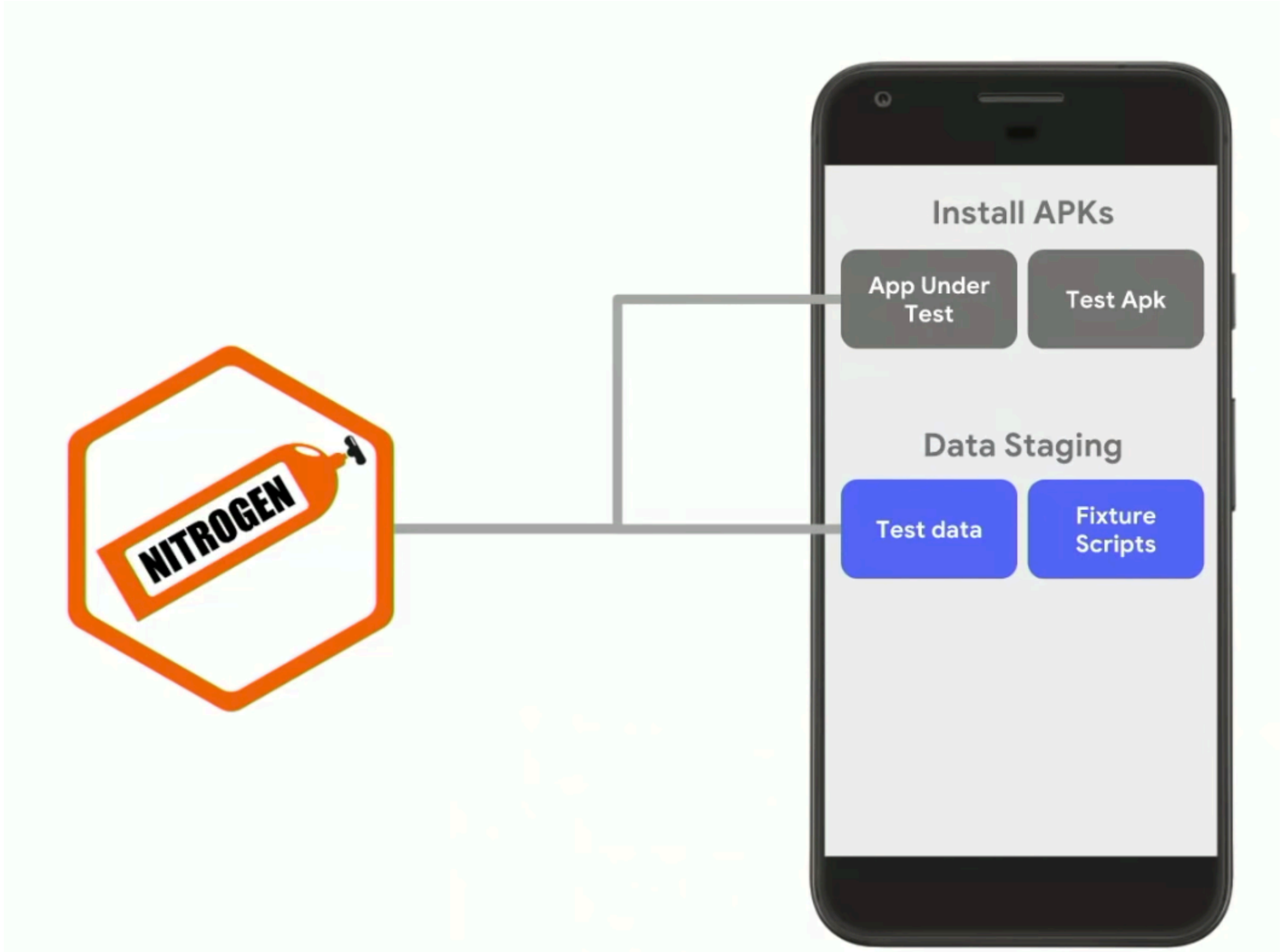


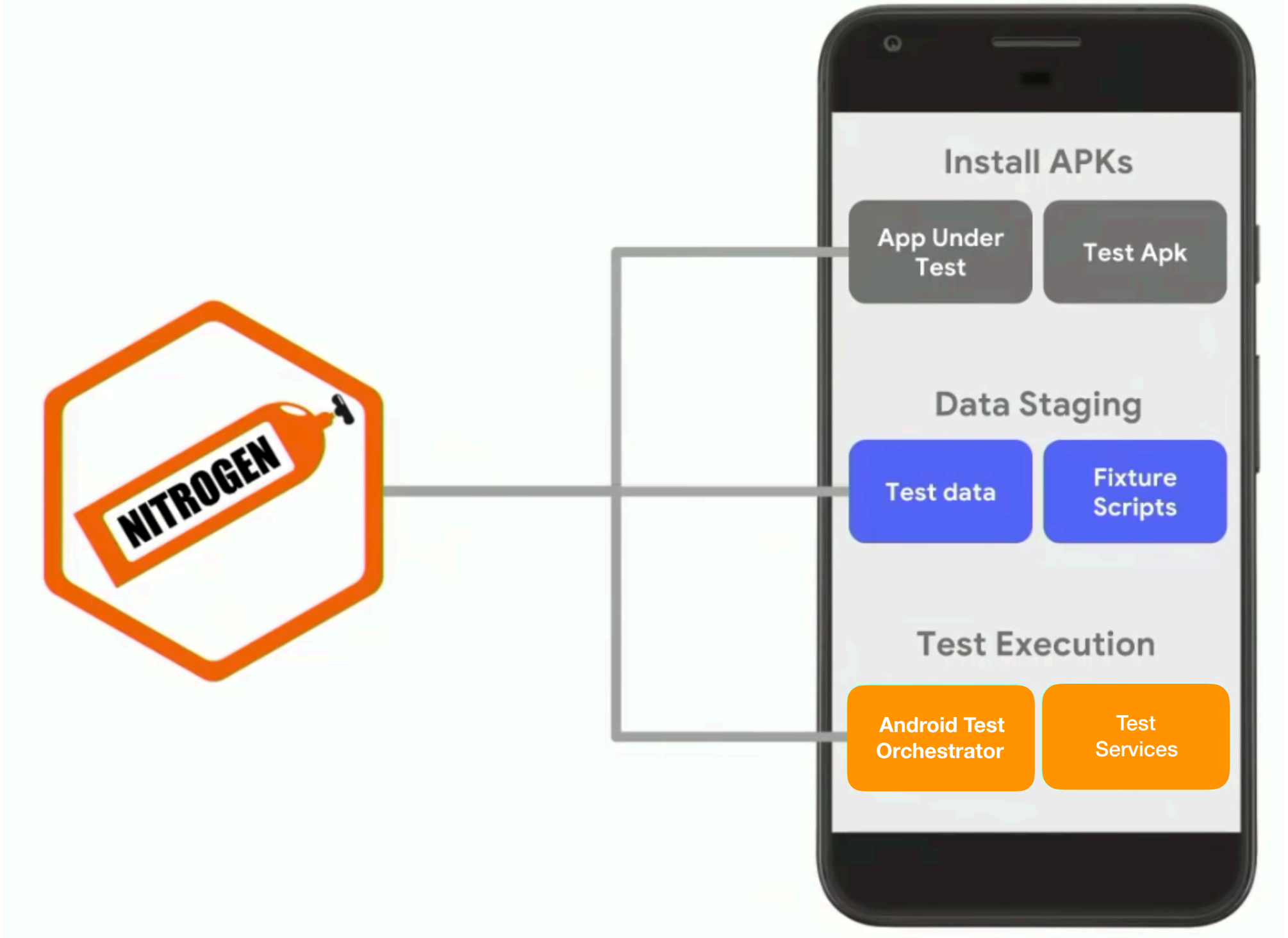



5:00
Back up
access
ary
Social
Google+
Promotions
Zagat, Google Offers
Updates
Google Play
Andy Brown
Bring Your Parents
Hey! What do you
Keri Andersr
Picture from
Check out t
Regis,
Board
Sur
DO WHAT YOU CAN'T
6.1M views
434K
7K
Share
Save
Add to
CaseyNeistat
7M subscribers
SUBSCRIBE
WITHOUT A GOAL
YOU CAN'T SCORE
22:16
DO MORE
CaseyNeistat
2.1M views
9:05
THE $21,000 FIRST CLASS
AIRPLANE SEAT
CaseyNeistat
42M views
7:39
FOLLOW YOUR DREAMS
CaseyNeistat
1.9M views
See the top five
stories right now
Google News
12:30
56° F


Environment
Test Artifacts
Fixtures
Setup


Install APKs
App Under Test
Test Apk
NITROGEN

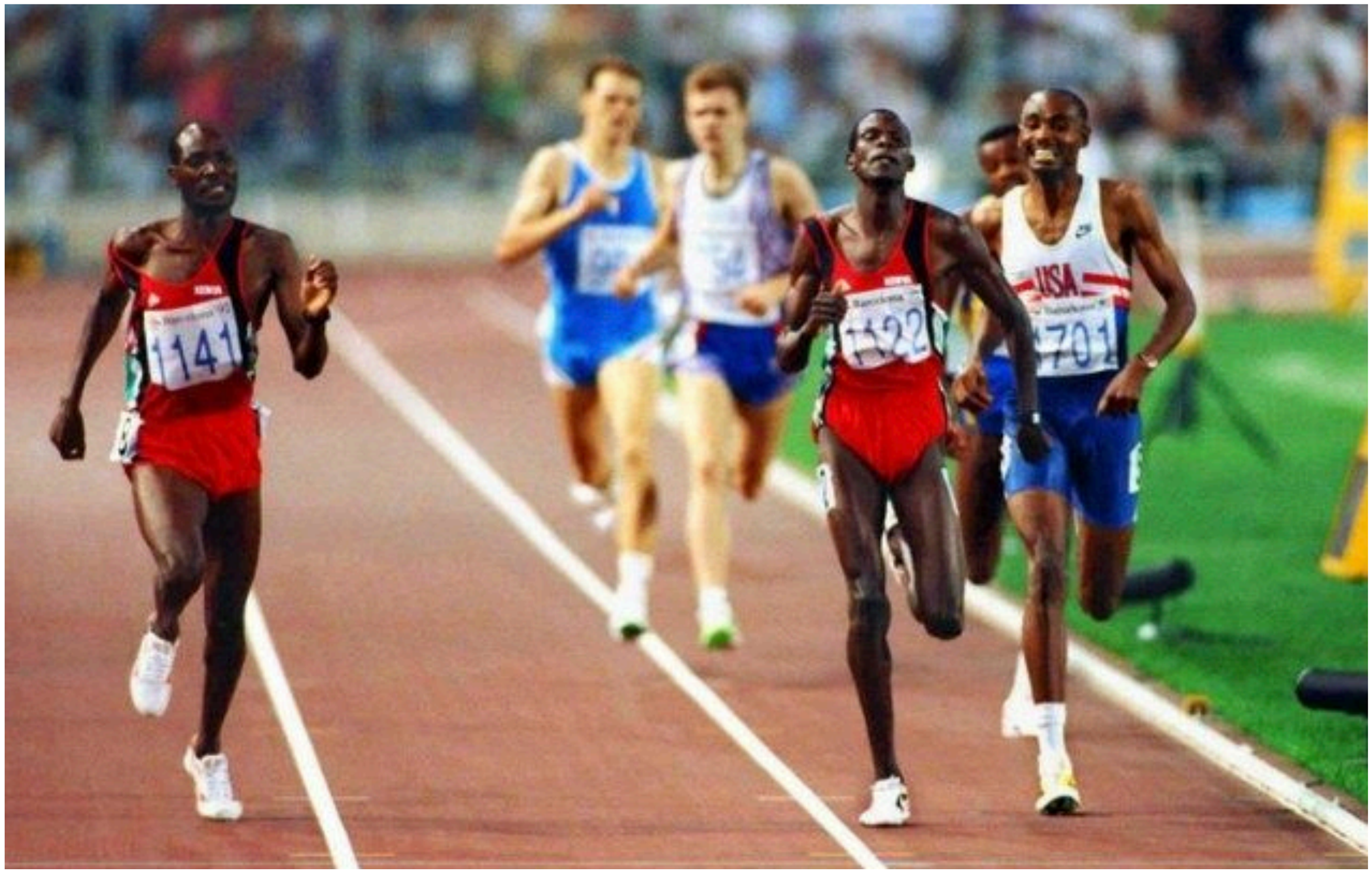

NITROGEN
Install APKs
App Under Test
Test Apk
Data Staging
Test data
Fixture Scripts


NITROGEN
Install APKs
App Under Test
Test Apk
Data Staging
Test data
Fixture Scripts
Test Execution
Android Test Orchestrator
Test Services

Running in the Real World

Sprint Running

Midd-Range Distance Running



Long Distance Running

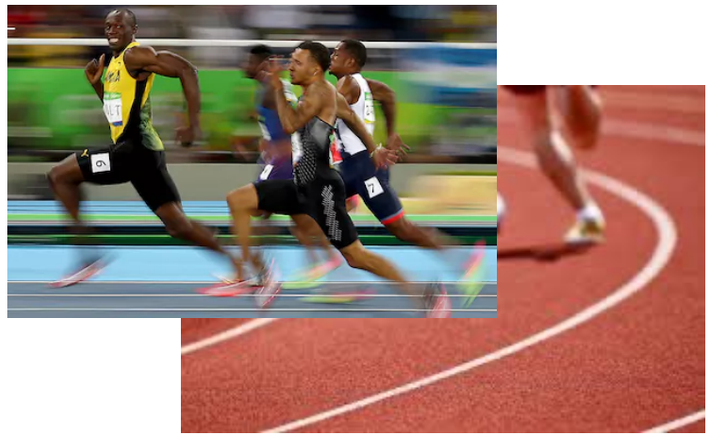
Track Running

Road Running

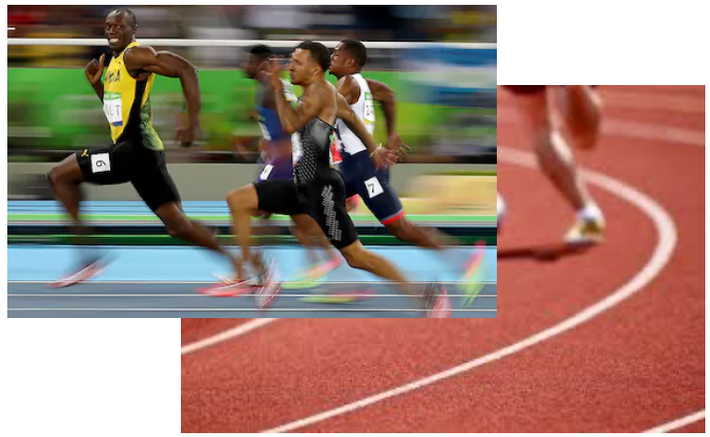
Cross Country Running

Mountain Running

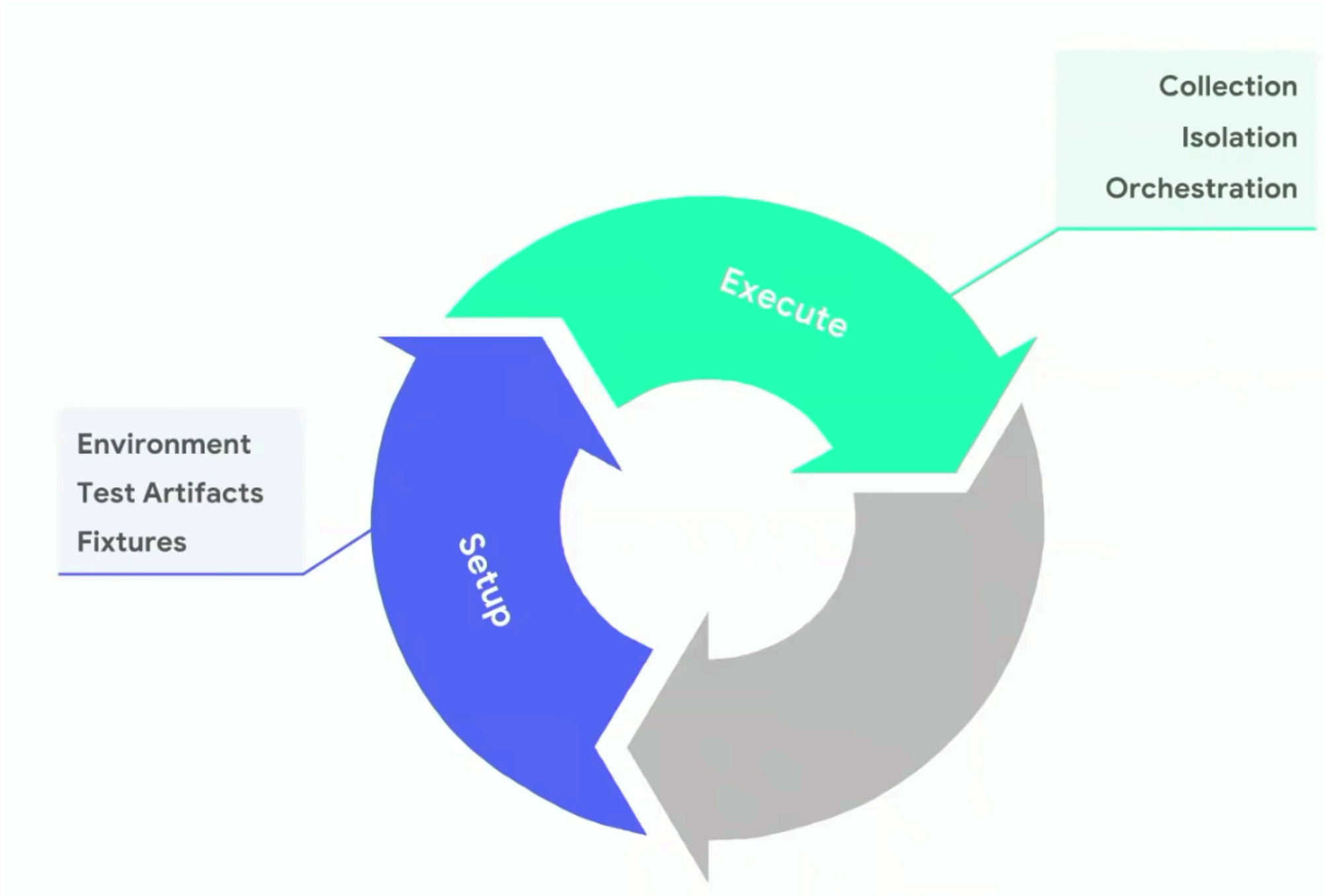
jUnit Test

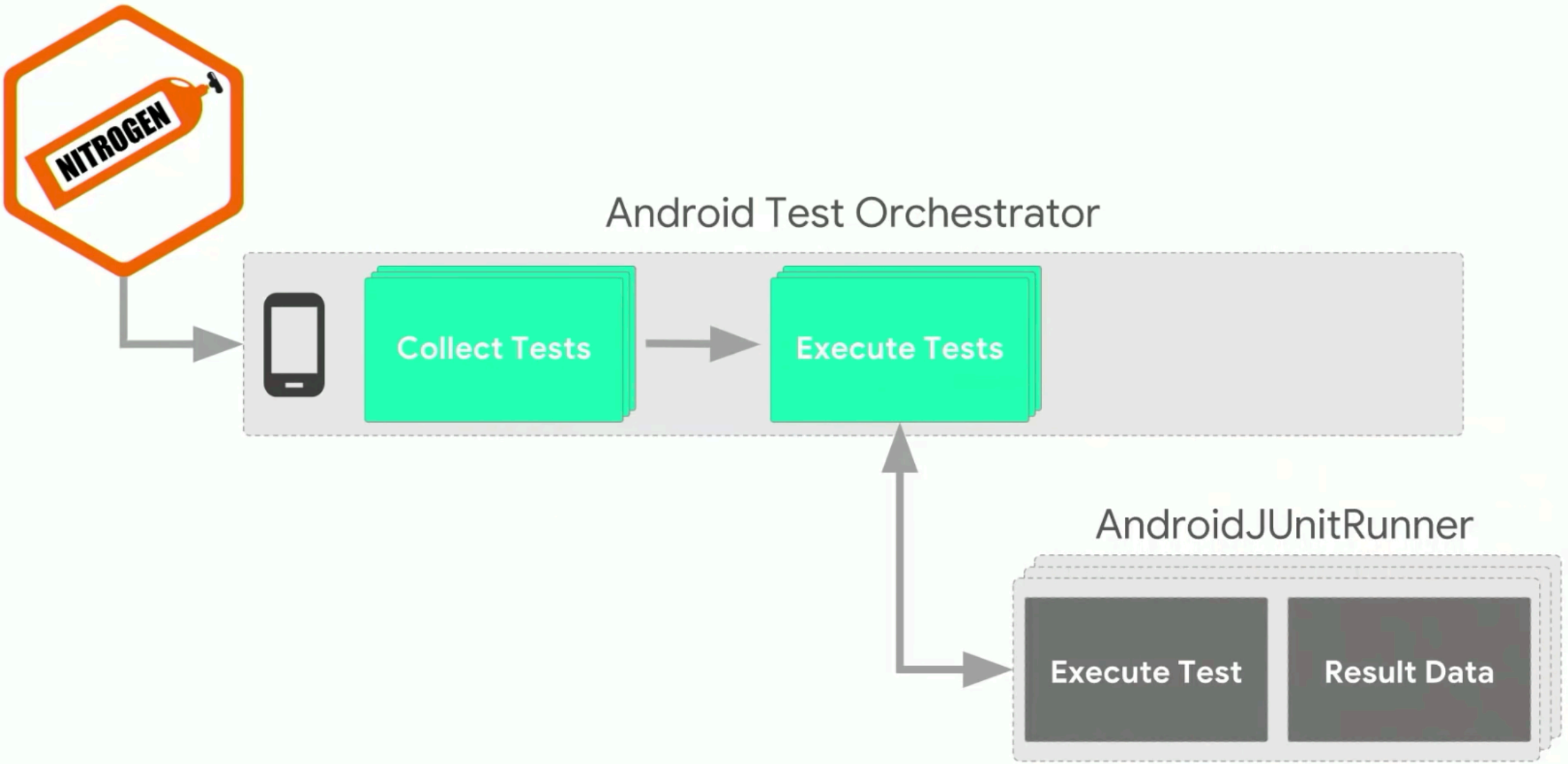


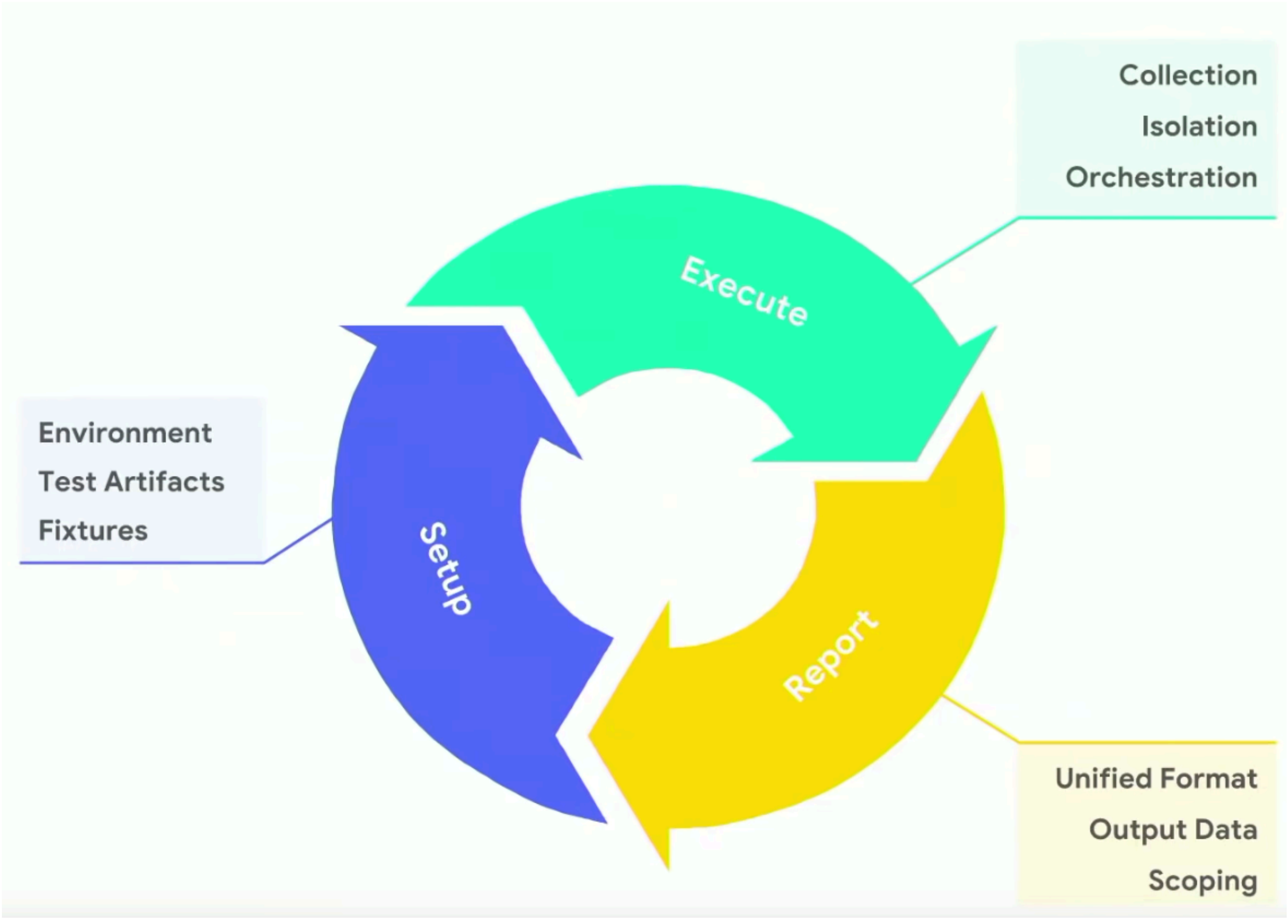
jUnit Test

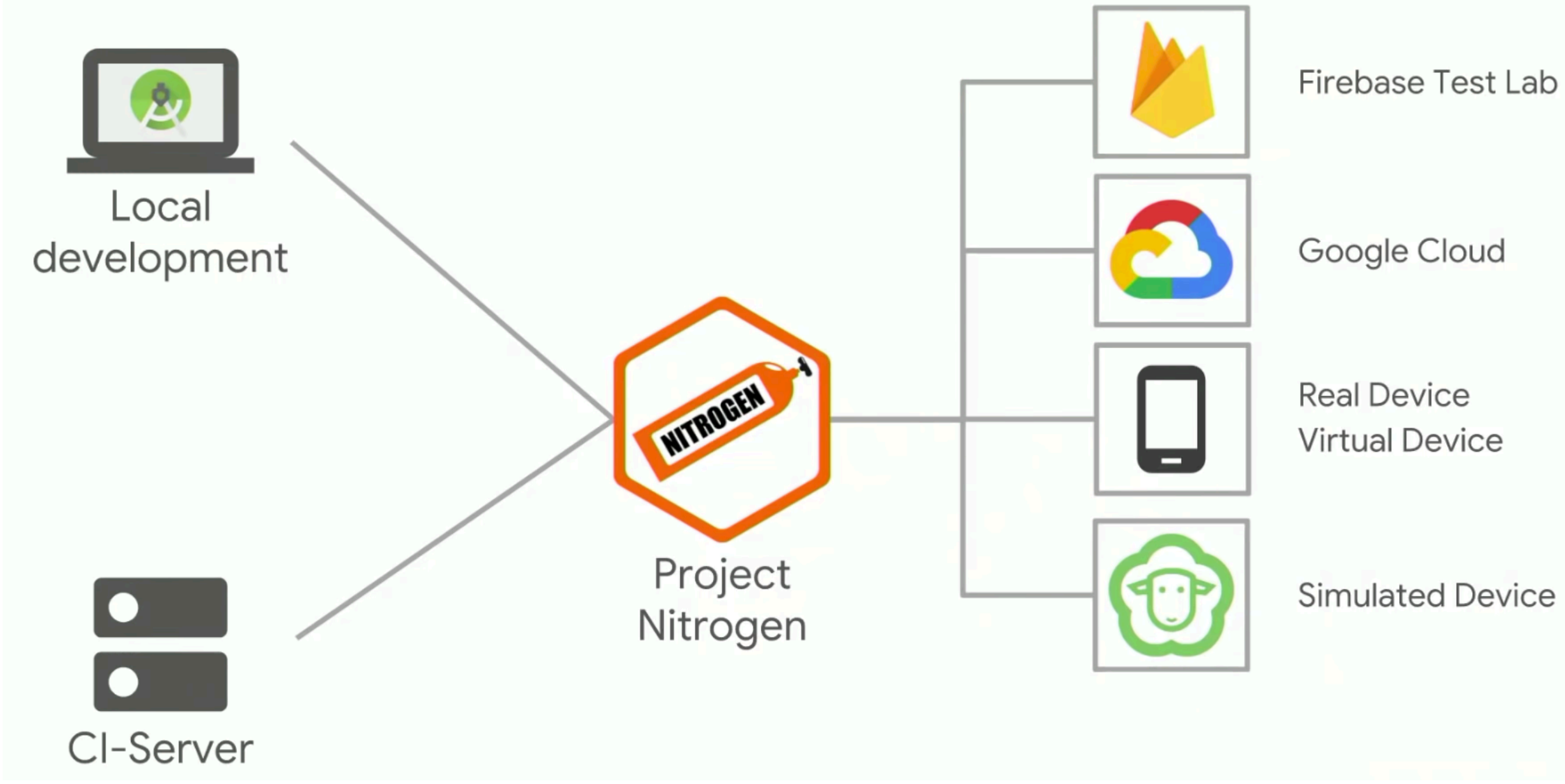


TestSuite

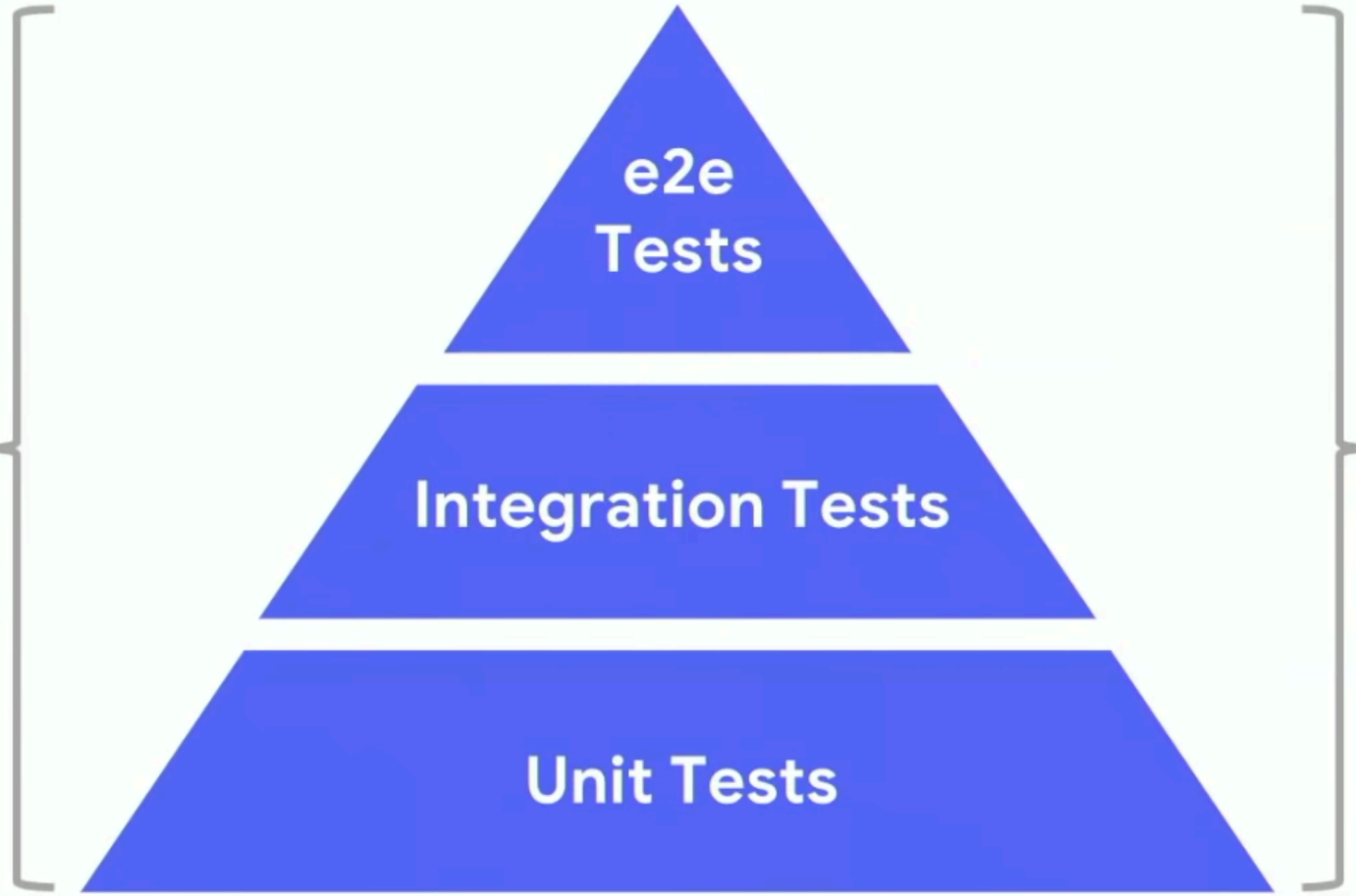

Collection
Isolation
Orchestration
Execute
Environment
Test Artifacts
Fixtures
Setup


NITROGEN
Android Test Orchestrator
Collect Tests
Execute Tests
AndroidJUnitRunner
Execute Test
Result Data


Collection
Isolation
Orchestration
Execute
Environment
Test Artifacts
Fixtures
Setup
Report
Unified Format
Output Data
Scoping


Local
development
NITROGEN
Project
Nitrogen
CI-Server
Firebase Test Lab
Google Cloud
Real Device
Virtual Device
Simulated Device

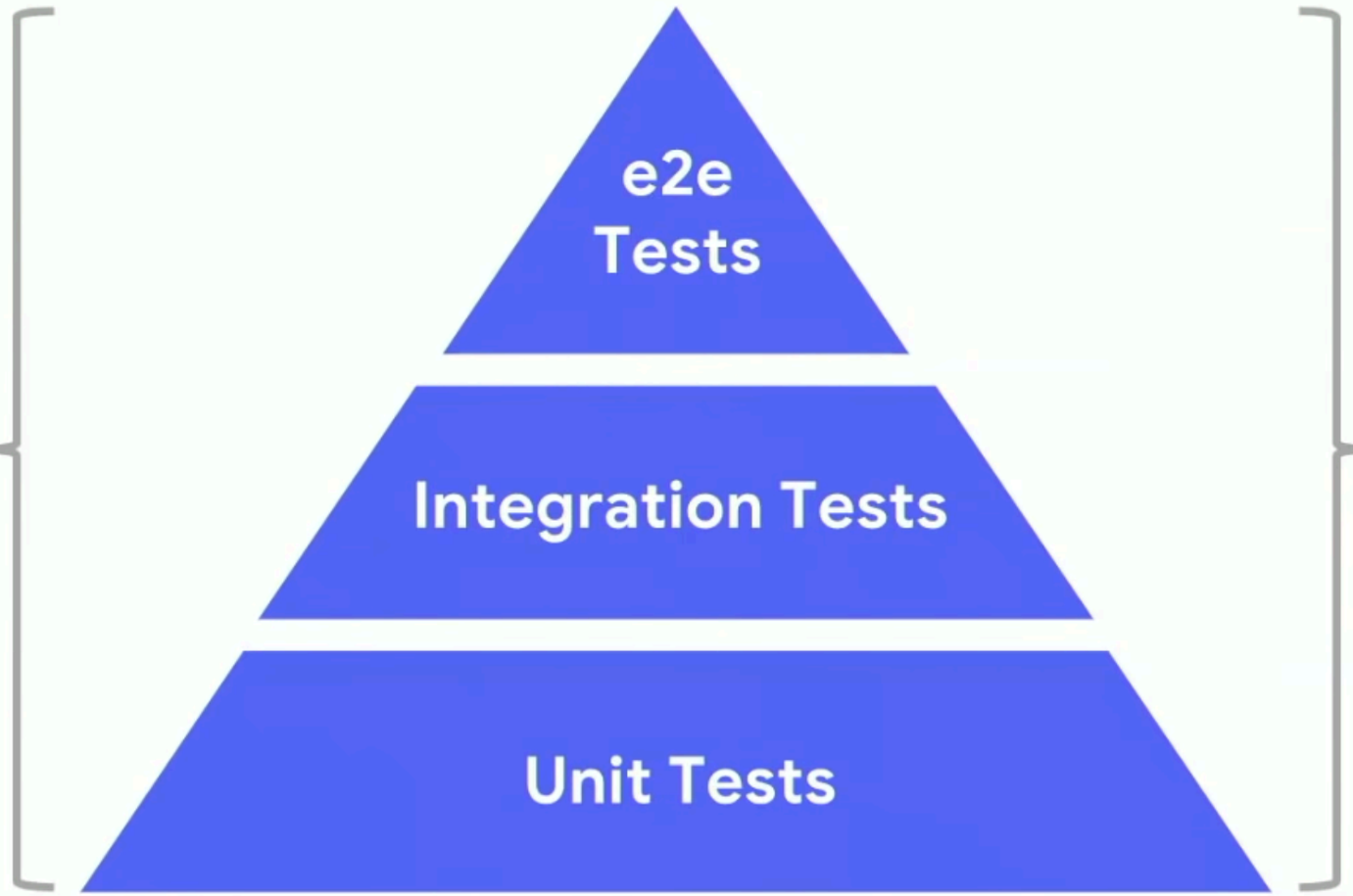
Project
Nitrogen



Android Test
Part of Jetpack

DEMO

Project
Nitrogen



Android Test
Part of Jetpack

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

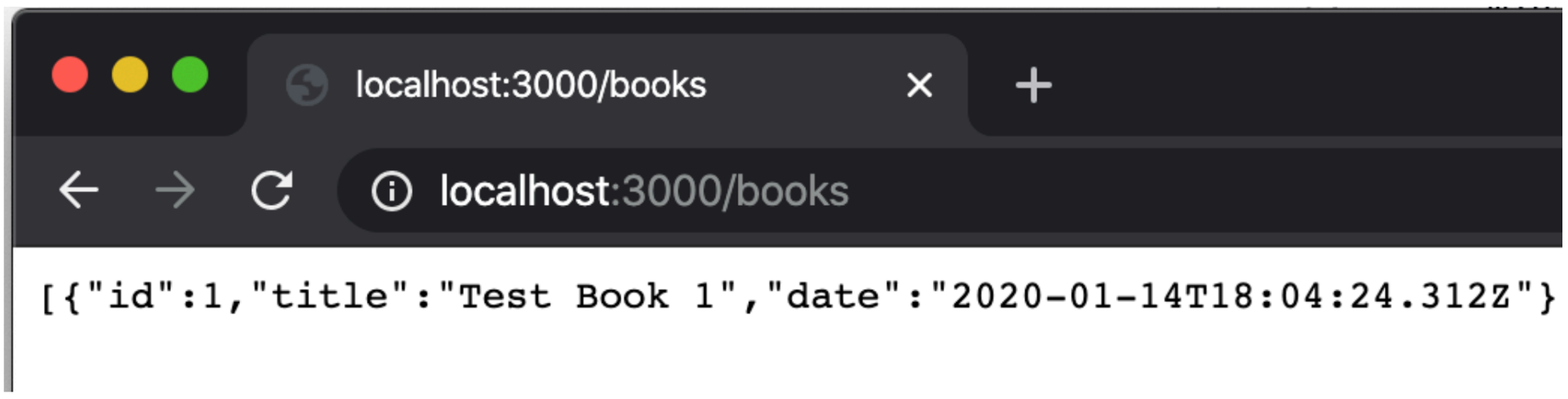
- Ensure that you have installed on your machine, the latest versions for:
    - Git
    - nodeJs
    - NPM

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 (v13) and npm (v6.13).

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:



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 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.