MTSC2019

中国移动互联网测试开发大会

Mobile Testing Summit China 2019

2019年6月28-29日 / 北京 国际会议中心

主办方：TesterHome 腾讯课堂

MTSC2019
中国移动互联网测试开发大会

Google

Testing the Modern Computing World

Complex End-to-end Testing for Alphabet's Computing Platforms and Services

Ang Li
Software Engineer
Google Inc.

主办方：TesterHome 腾讯课堂

Modern computing platforms and services
Power today's society, and
Enrich our everyday life,

…so we better test them

Modern Computing Platforms

- Mobile Phones
- Internet-Of-Things (IOT)
- Gaming
- Wearables
- Medical d[illegible]
- Cars
- Robotics

Infrastructure Powering These Platforms

- Operating Systems
  - Android
  - iOS
- Wireless Communication
  - Cellular
  - Wi-Fi
  - Bluetooth
  - RF
- Network Infrastructure

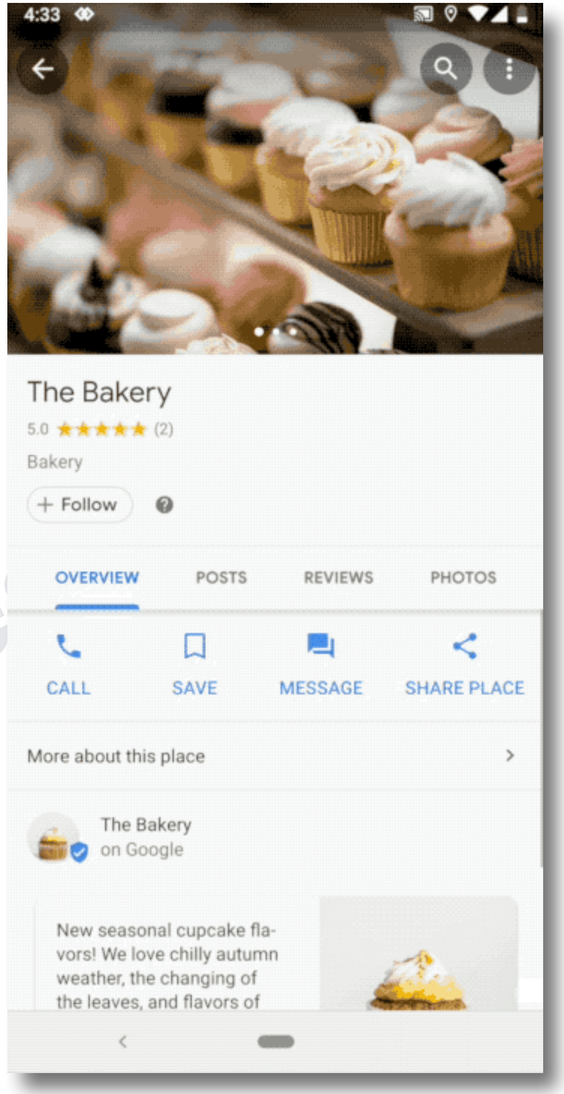
RCS

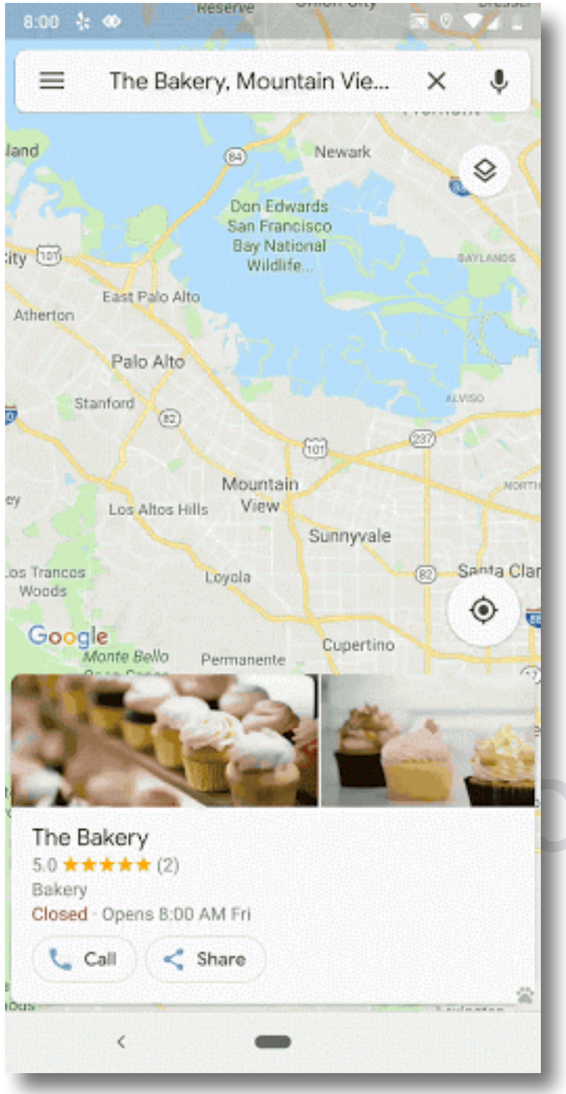
Google Station

Interactive User Scenarios

- Video chat
- Money transfer
  - Google Pay
- Peer-to-peer file transfer
  - Files by Google
- Live parental control
  - Family link
- Calling a ride by self-driving cars
  - Waymo
- Proximity detection + mobile data sharing
  - Instant tethering
- Multi-screen gameplay + game controller
  - Stadia

Interactive User Scenario Example: Google Maps Merchant

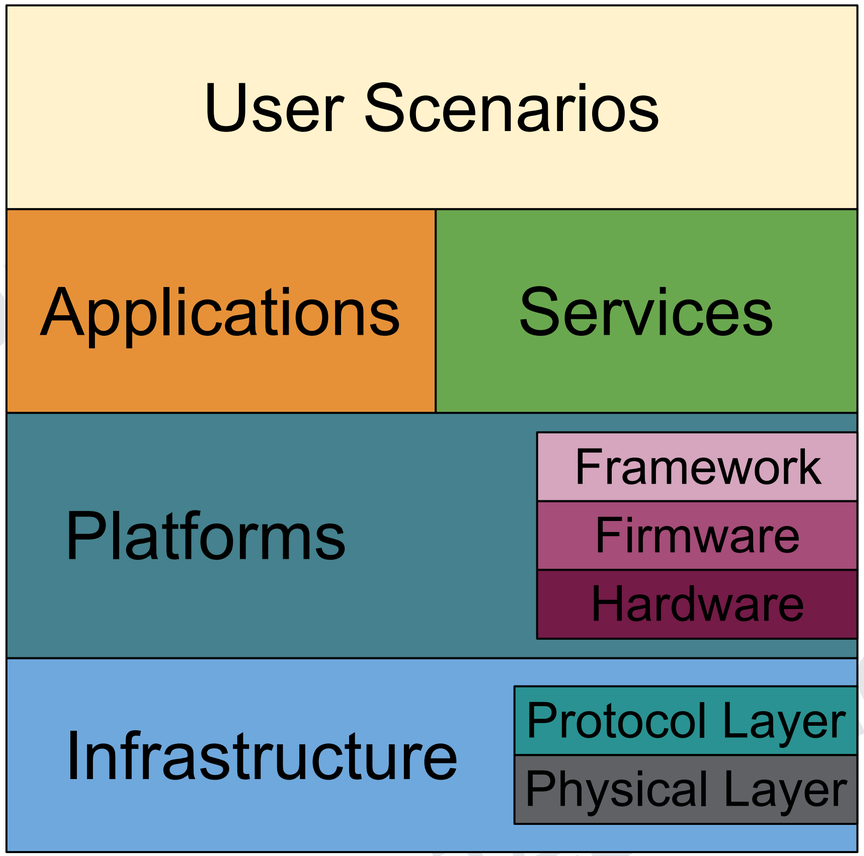




With great power *to the users*
Comes *the need* for
Greate test infrastructure

Layering Status: It's Complicated

- The technologies involved in a user scenario can be sliced into many layers
- A company may provide any combination of these layers
- Testing requirements for different layers and their combinations can vary significantly



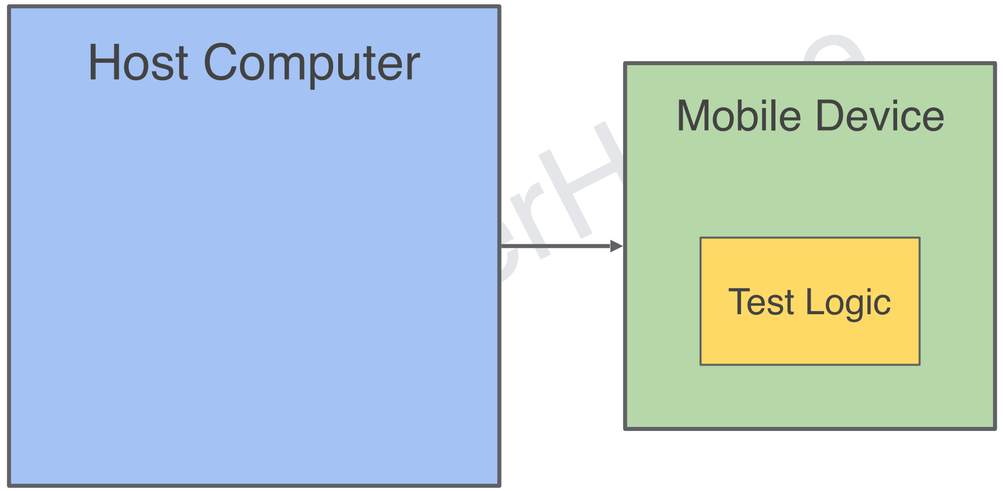
Requirements

- Full control of the mobile devices
  - Everything the single-device tests do
  - External-driven actions, like reading sys logs, rebooting the device etc
- Coordinate multiple devices to create user scenarios
  - Messaging between two clients
  - Adjust thermostat via mobile app
- Supports non-mobile devices
  - Other device types like wearables and IOT devices
  - Test instruments like power meters, attenuators
  - Future products
- Handles requirements for various layers of technologies
- **Easy to write and debug tests**

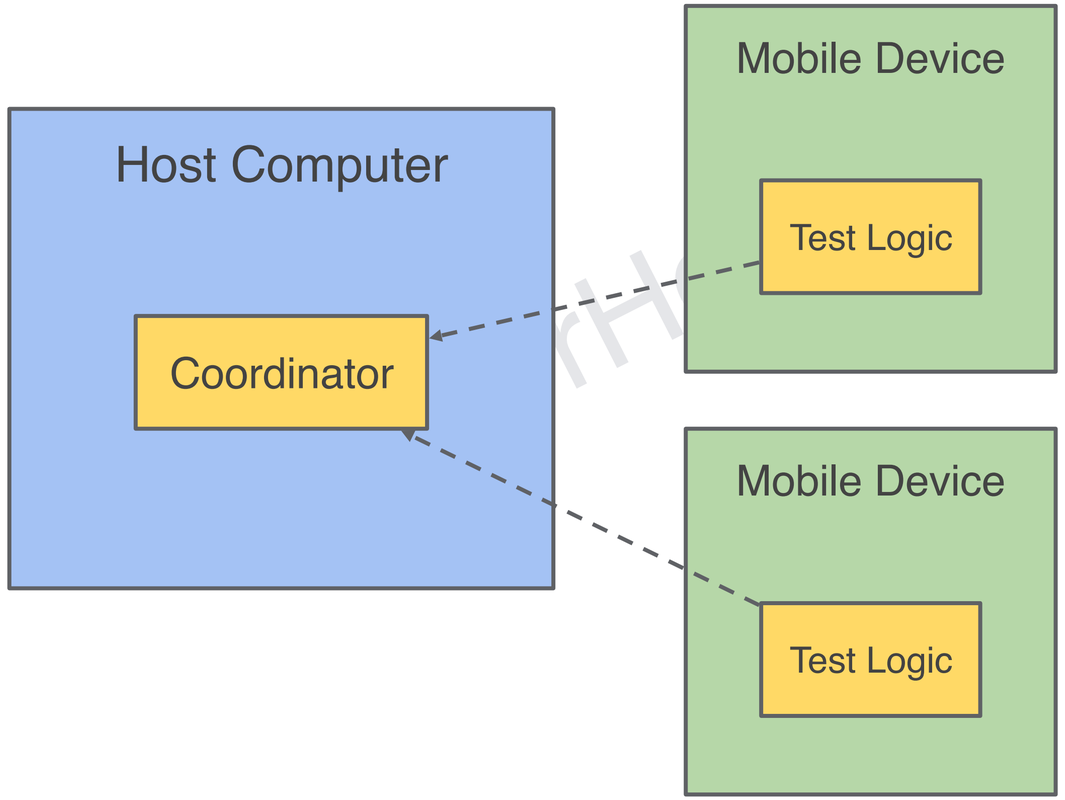
Common Architecture of Mobile Test Frameworks

...and Their Problems

Single Mobile Device, Device-driven



Distributed



Problems

- Test logic has little to no visibility outside of the Device Under Test (DUT)
  - Tests are affected by problems of the DUT
  - Difficult to control and coordinate multiple devices..
- UI centric
  - Limited to no option to conduct a test without UI
- Designed for unit tests
  - Reporting and test structure assume single-app tests
- Difficult to adopt for non-mobile devices

Solution

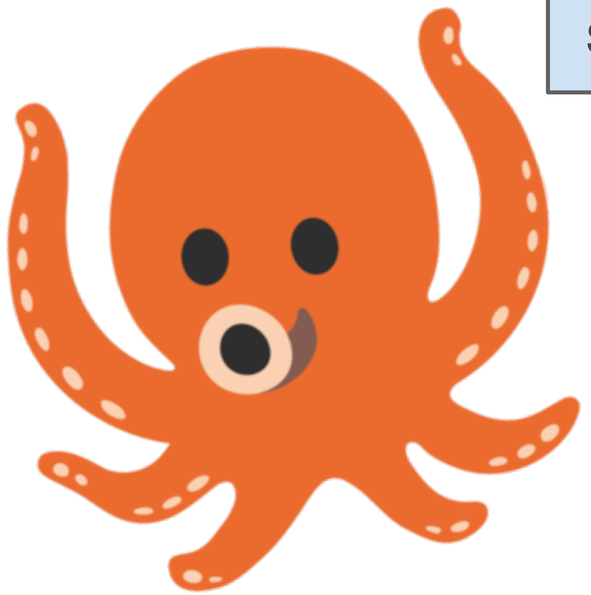
Introducing A Different Type of Test

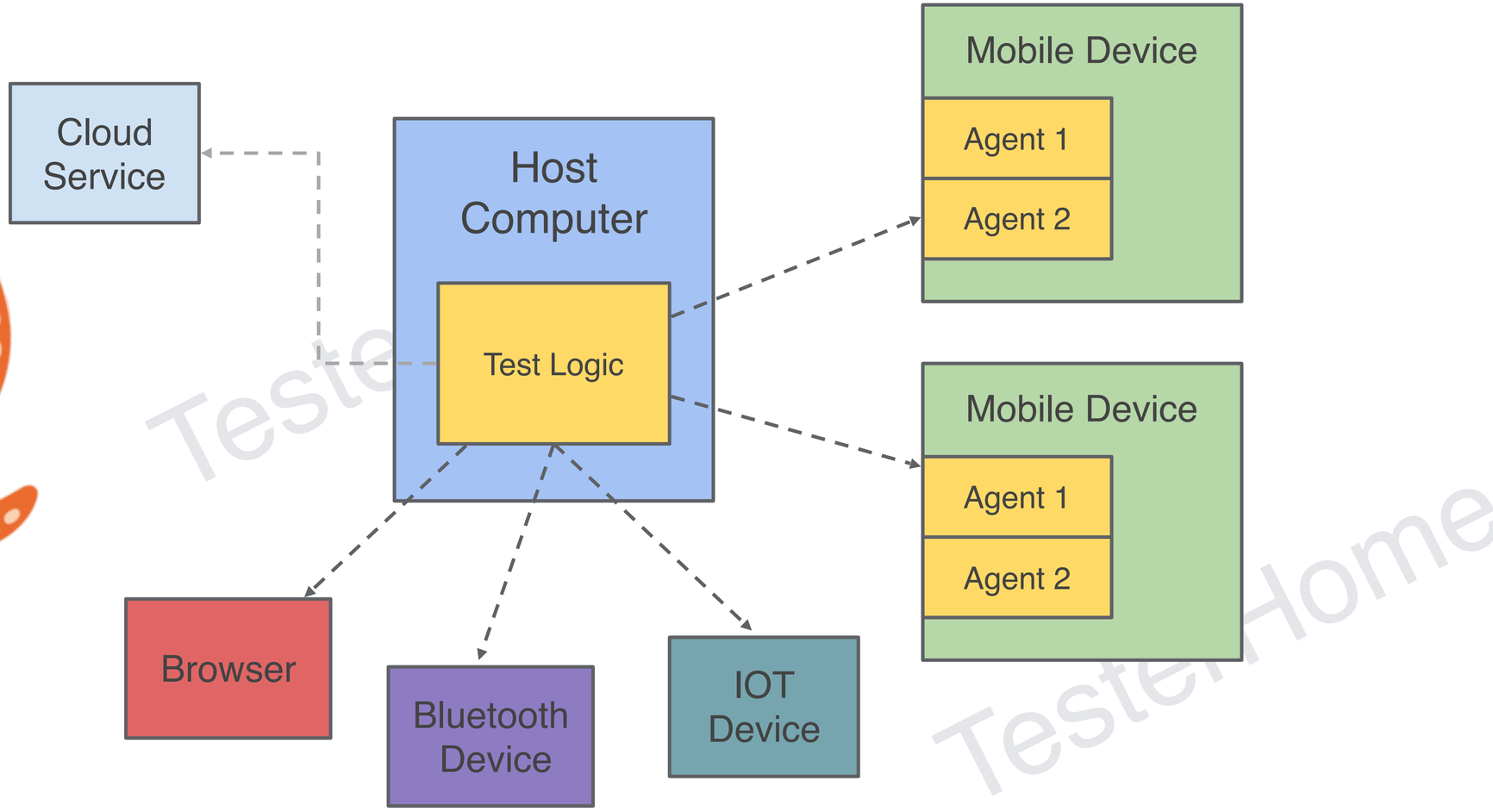
Host-driven Generic Controller Architecture

with two key design axioms

- The test logic shall reside at a central location, often a host computer
- The framework shall assume generic controllers as components instead of mobile Devices

Example Logic Diagram





Advantages

- Test logic is a streamlined piece of code at a "neutral ground"
  - Easy to understand and debug
  - is not part of the test
  - Has visibility and control of all test components
- Test logic can coordinate actions to simulate a user scenario
- Test can operate non-mobile device components
  - Utilize any device types you'd like into your mobile tests

Tools Built for the Task

- Mobly
  - A test framework designed for creating complex E2E tests
  - Open to arbitrary device type via the "controller model"
  - Mature definition of result reporting
- Mobly Snippet Library
  - A RPC library for host logic to communicate with test components
  - Allow users to build custom device-side steps to be triggered by host-side main logic
- Mobly Bundled Snippets for Android
  - A set of pre-implemented snippets for basic Android operations, such as
    - Make a toast
    - Enable Bluetooth
    - Add a Google account

Mobly Test Framework

A Python test runner for creating complex end-to-end tests

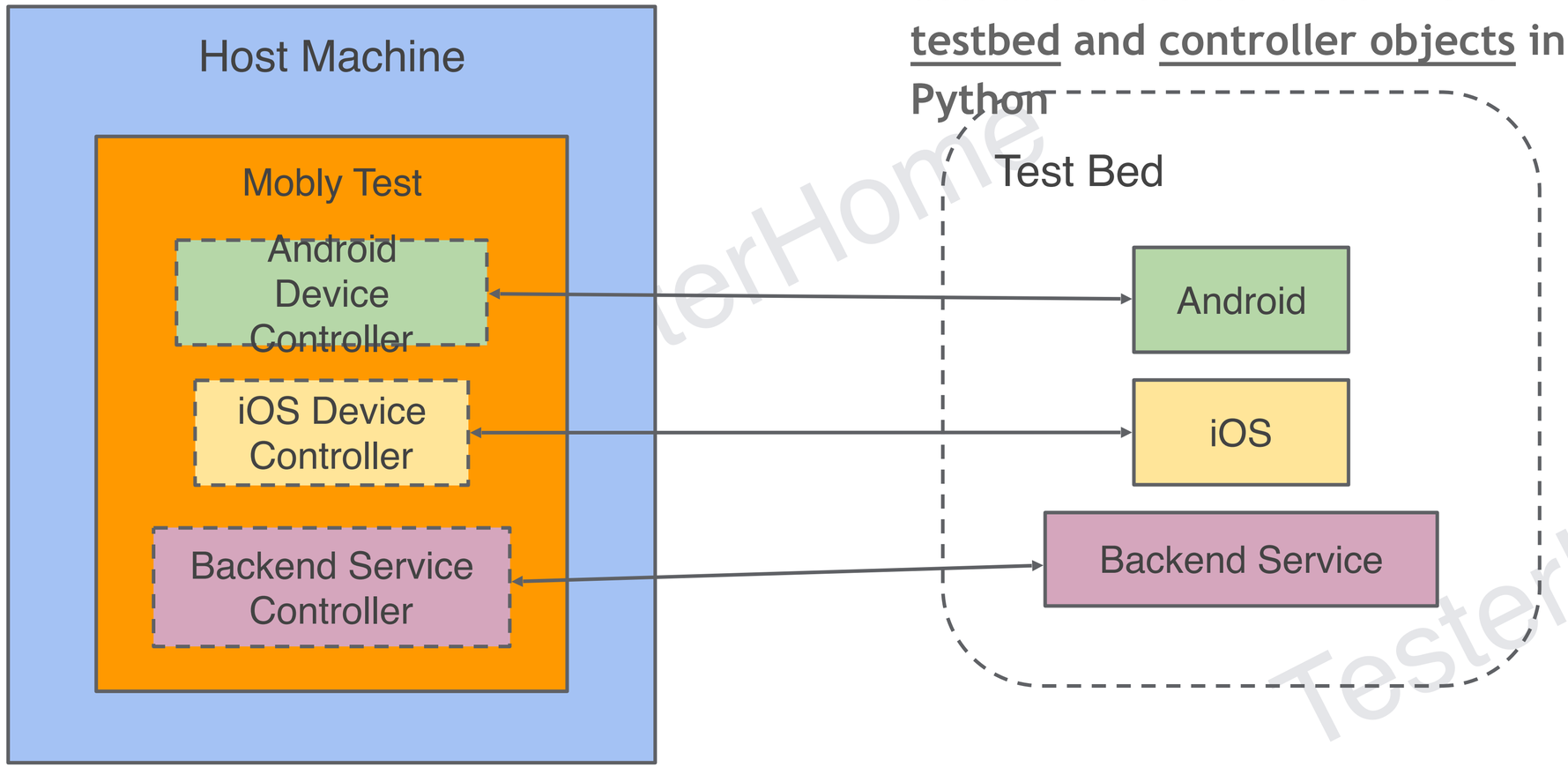
- A Mobly test operates on a collection of controllers
  - **Test Bed** - a collection of components used in the test (Phones, services, equipment etc)
  - **Controllers** - objects in Python script representing components in the test bed.
- Flexible and pluggable
- Open source

Designed for Complex Tests

Similar to standard unit test, with additional features to accommodate more complex tests.

- Controller + Testbed model
- Define multiple failure points in a single test
  - `expects` APIs
- Rich reporting features
  - Add extra info in assertions
- Conditional stages triggered by test status
  - `on_fail` is commonly used for debug info collection and recovery

Testbed + Controller Structure



One-to-one mapping between the actual devices/services in the testbed and controller objects in Python

Controller "Interface"

A loosely defined module level "interface" for declaring arbitrary test components

- *create*
  - The factory function that translates configs into objects
  - Config schema is entirely up to the controller owner
- *destroy*
  - Tear down the controller objects generated by `create`
- *get_info*
  - Retrieve any useful info from the controller after the test run

Example Controller - AndroidDevice

Distributed as part of the Mobly pip package

- Full ADB access
  - ad.adb.shell(['ls', '/sdcard'])
- Long-running service management
  - logcat
  - video recording
  - custom services
- Designed to handle most active Android versions
- Client for Mobly snippets

Reporting

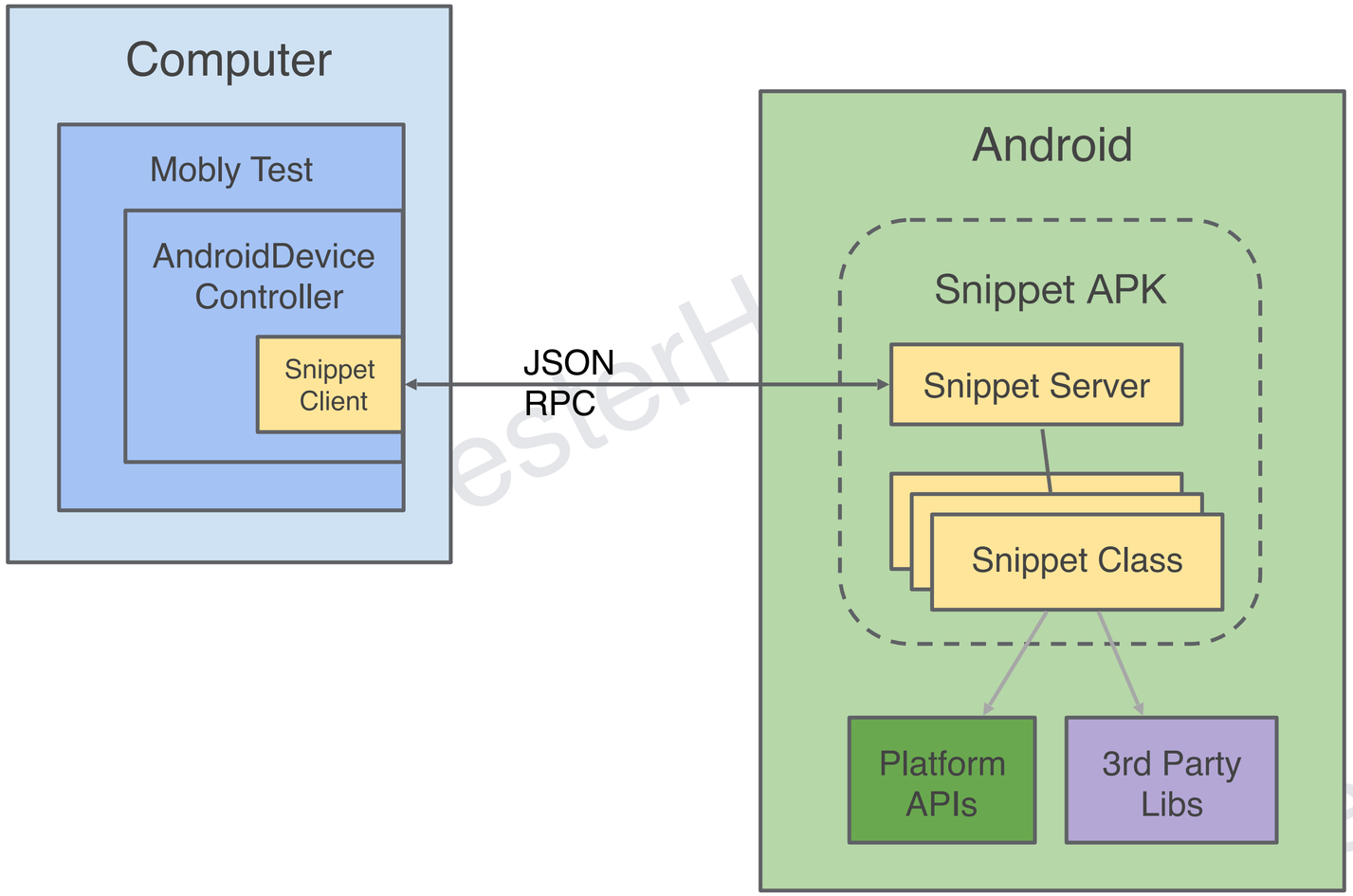
- Live streamed YAML
  - Easy to parse and integrate
  - Supports custom error code systems
  - Flexible schema that can accommodate custom features
- Customize files/logs
  - Add any extra files you need for debugging
  - Add per-test files

Mobly Snippet Library

A standardized RPC protocol for interacting with a device in Mobly tests.

- All the power of a single device test
  - Android: Instrumentation test
  - iOS: XCTest, XCUITest
  - Common libraries like Espresso/UI Automator
- Call the native platform APIs
  - Java, Swift, Objective-C etc
- Synchronous and Asynchronous calls

Mobly Snippet Library - Android



Mobly Snippet Library - Android Java

```
package com.mypackage.testing.snippets.example;

public class ExampleSnippet implements Snippet {
  public ExampleSnippet(Context context) {}

  @Rpc(description='Returns a string greeting the user by name.')
  public String sayHello(String name) {
    return "Hello, " + name + "!";
  }

}
```

Mobly Snippet Library - Invoking from Python

```
from mobly import base_test
from mobly.controllers import android_device

class ExampleTest(base_test.BaseTestClass):

  def setup_class(self):
    self.ad = self.register_controller(android_device)[0]
    self.ad.load_snippets(name='snippet',
                          package='com.mypackage.testing.snippets.example')

  def test_foo(self):
    foo = self.ad.snippet.sayHello('Jeff')  # `foo` is "Hello, Jeff!"
```

Mobly Snippet Library - iOS

```
#import <Mobly/Mobly.h>

@interface ExampleSnippet: NSObject
@end

@implementation ExampleSnippet
+ (void)load {
  NSArray *methods = [MBLMethod methodsWithReference:[ExampleSnippet class]
                                           selectors:@selector(sayHello:), nil];
  [MBLRegistry.sharedInstance registerRPCMethods:methods];
}

+ (NSString *)sayHello:(NSString *)name {
  return [NSString stringWithFormat:@"Hello, %@!", name];
}
@end
```

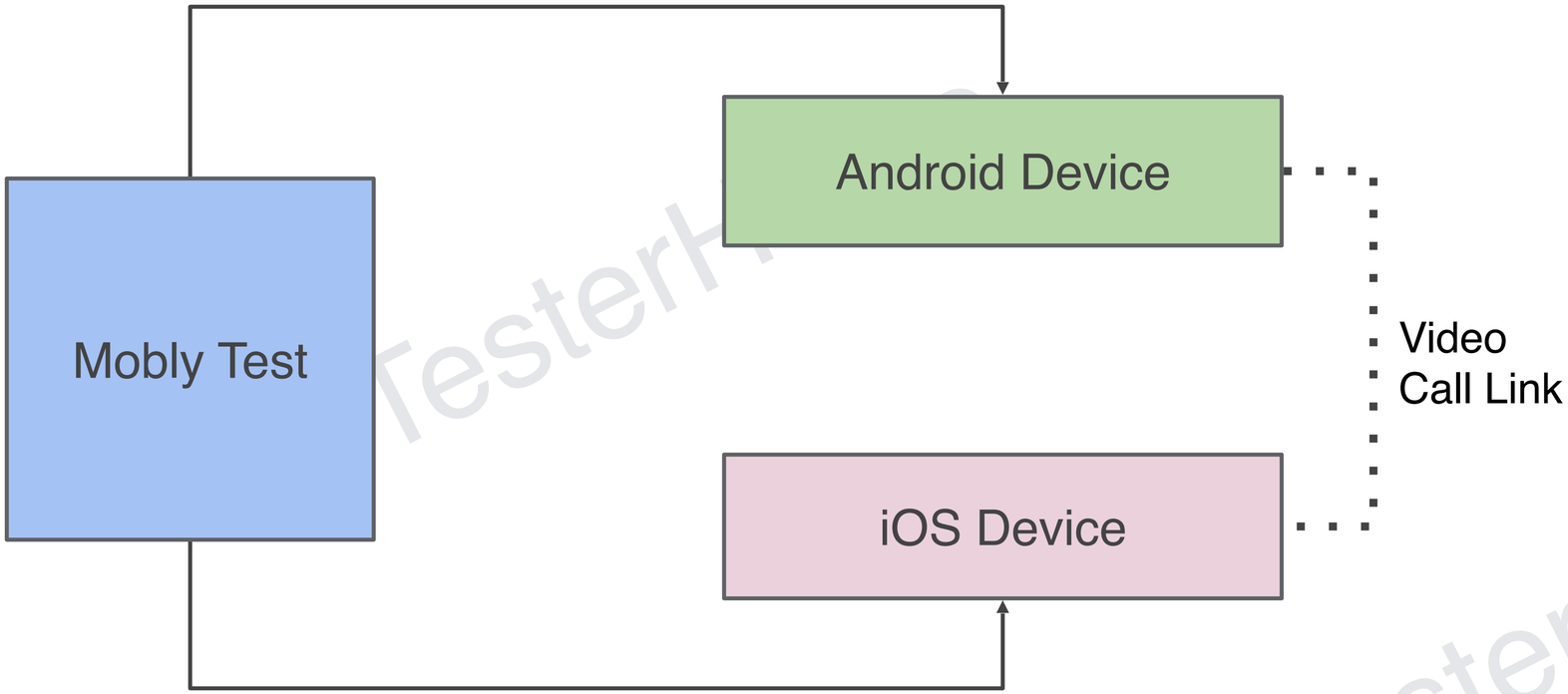
Sample Code & Use Cases

Example Test: Cross-platform Video Call

Making a phone call between one Android and one iOS devices.

- Assuming we already have phone call related snippets implemented.

Cross-platform Video Call Setup Diagram



Test Bed Config

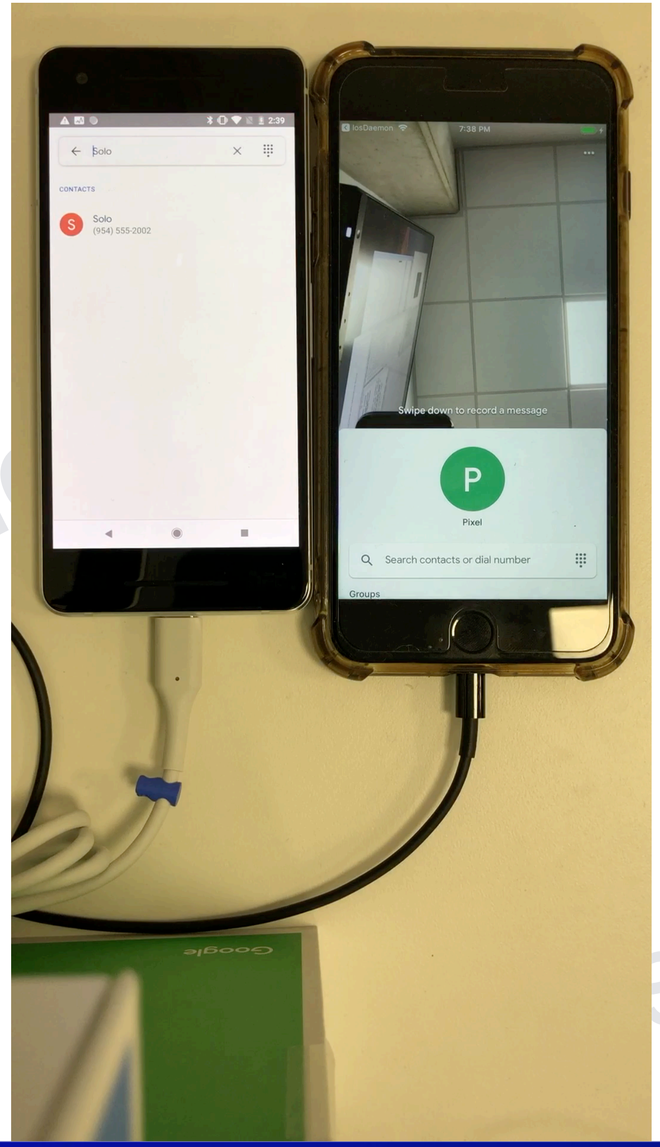
```
TestBeds:
    - Name: SampleTestBed
      Controllers:
          AndroidDevice:
            - serial: ABCDEFG1234567
            - phone_number: 4150007890
          IosDevice:
            - udid: 7654321-asdfghjkl
            - phone_number: 4150001234
```

Acquiring Controller Objects

```
from mobly import base_test
from mobly import expects
from mobly.controllers import android_device
from mobly.controllers import ios_device

class CallTest(base_test.BaseTestClass):

  def setup_class(self):
    self.android = self.register_controller(android_device)[0]
    self.iphone = self.register_controller(ios_device)[0]
    self.android.load_snippet('snippet', 'com.google.test.example.snippet')
    self.iphone.load_snippet('snippet', 'com.google.test.example.snippet')
```

```
class CallTest(base_test.BaseTestClass):
  ...
  def test_simple_call(self):
    self.iphone.makeCall(self.android.phone_number)
    asserts.assert_equal(self.android.snippet.getCallState(), 'RINGING')
    self.android.snippet.acceptCall()
    with expects.expect_no_raises('Callee failed to enter calling state.'):
      asserts.assert_equal(self.android.snippet.getCallState, 'IN_CALL')
    asserts.assert_equal(self.iphone.snippet.getCallState, 'IN_CALL')

  def on_fail(self):
    self.android.take_bug_report()
    product_info = self.android.adb.shell.getprop('ro.build.product')
    self.android.log.info('My build product is %s', product_info)
    self.iphone.save_syslog()
```

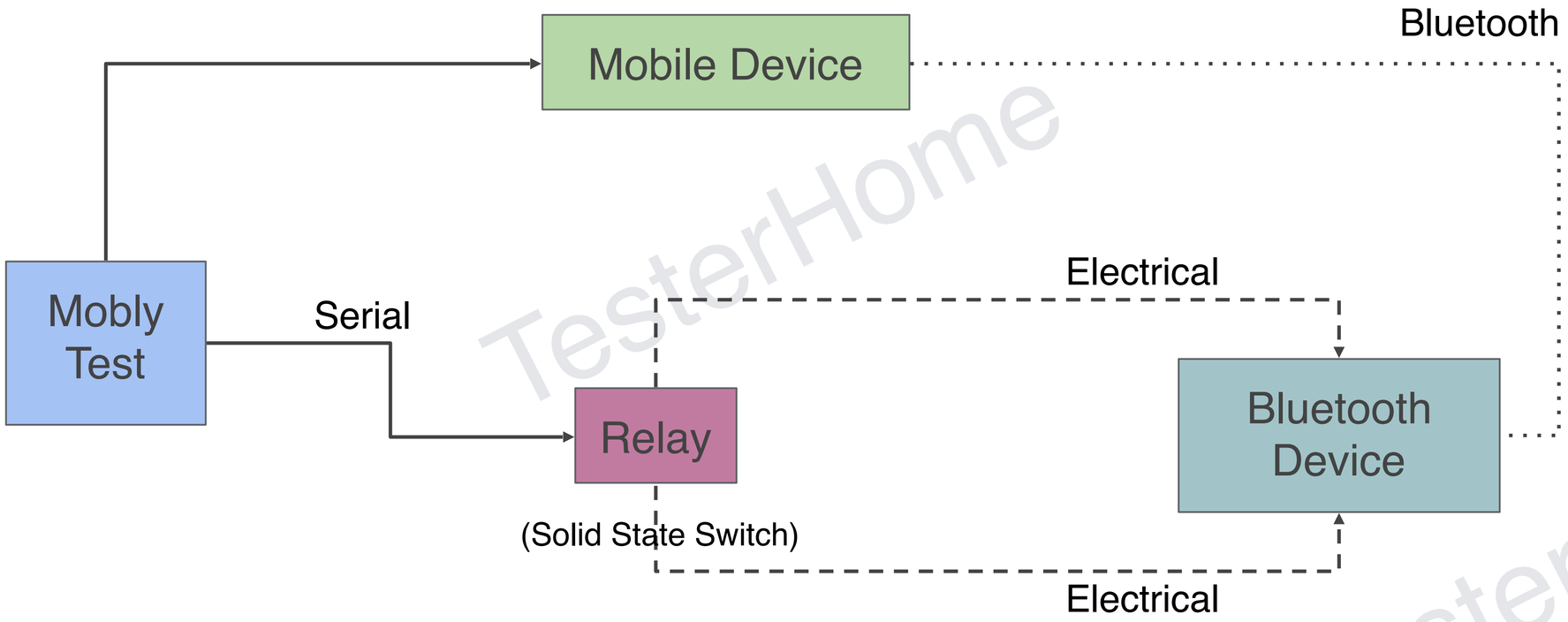
Cross-platform Duo Video Call Demo



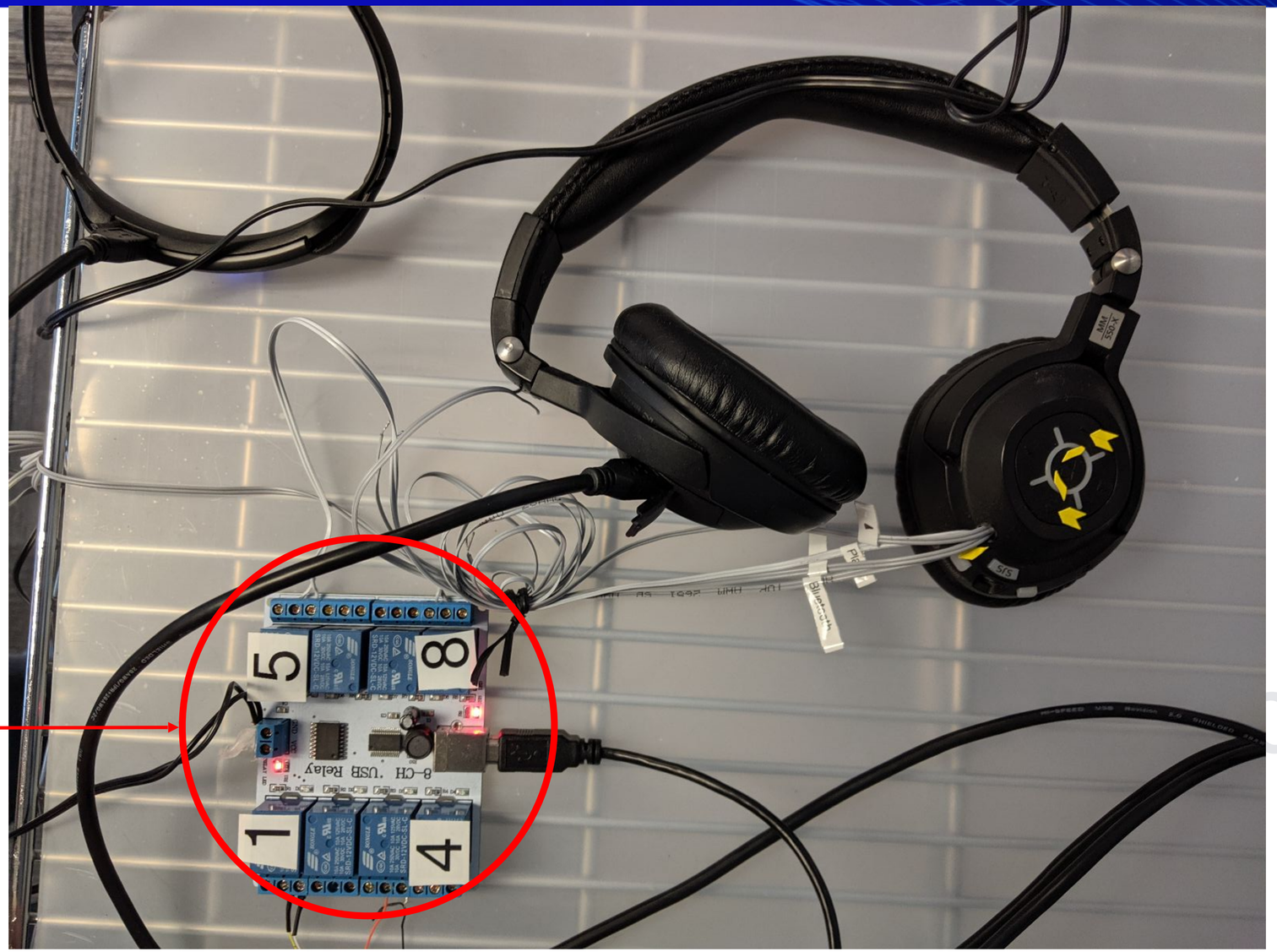
Example Use Case:

Pairing Bluetooth Devices

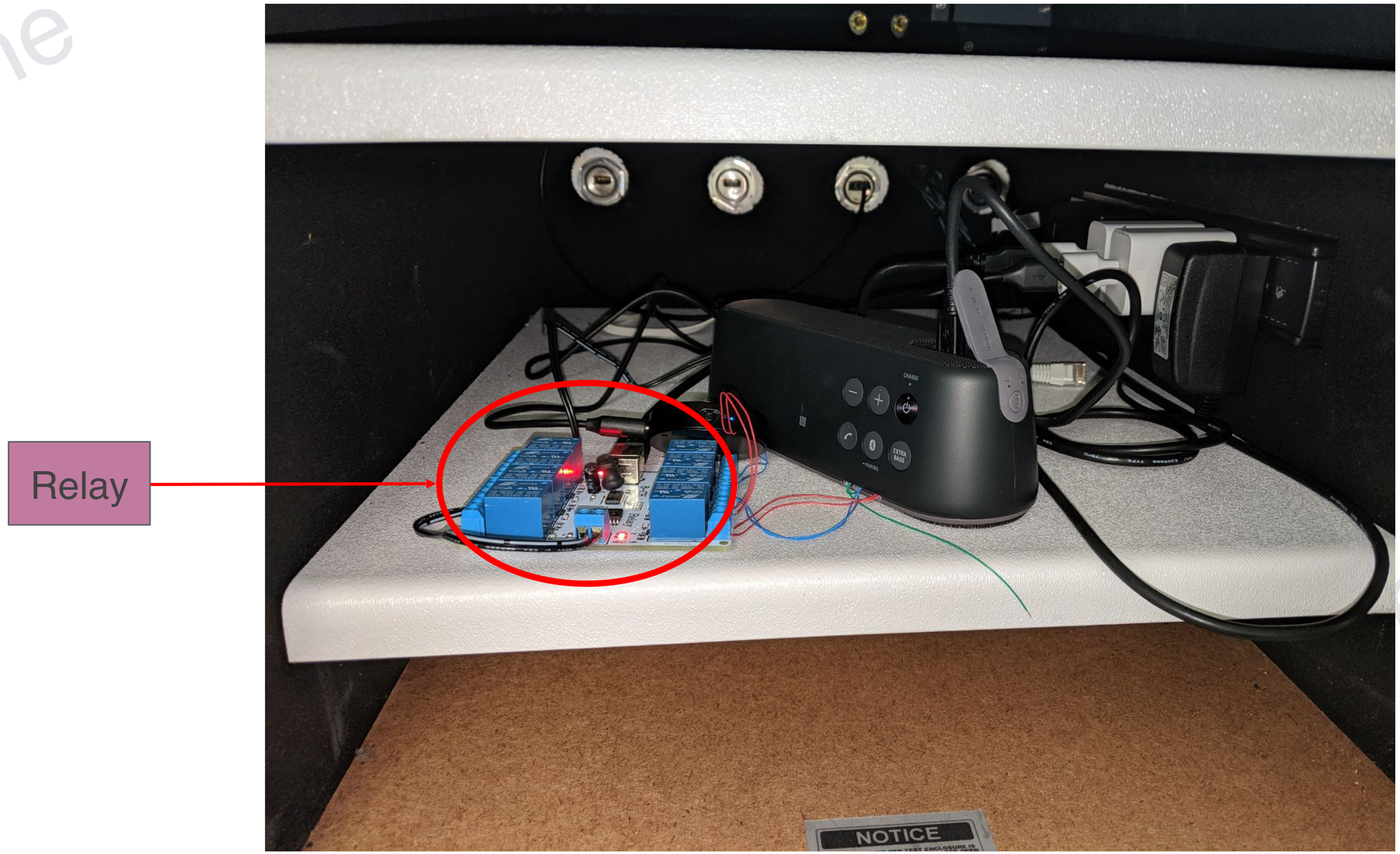
Bluetooth Pairing Setup Diagram



Bluetooth Headphones Setup



Bluetooth Speaker Setup

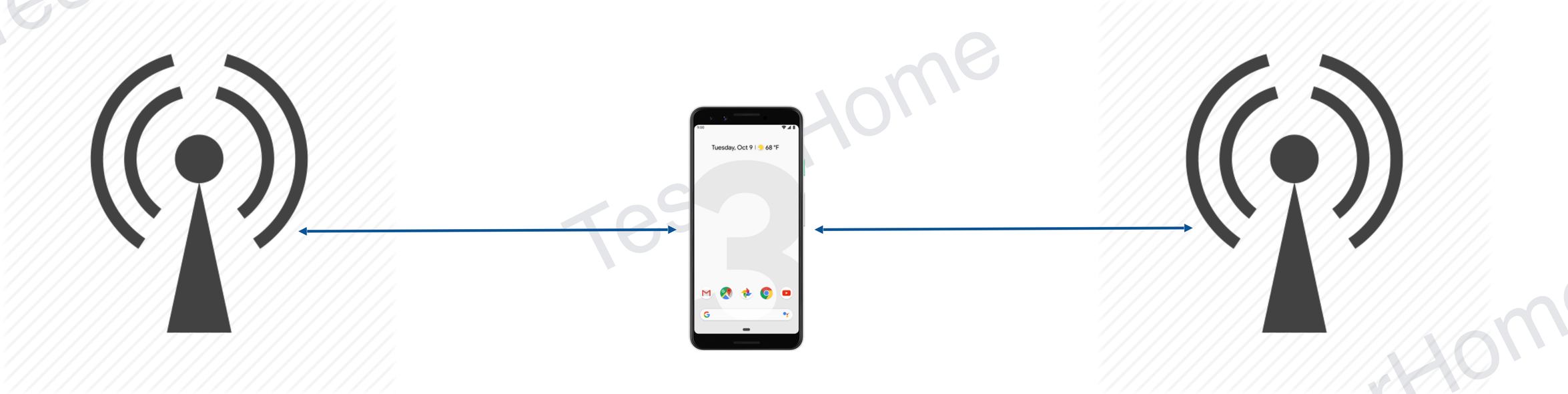


Example Use Case:

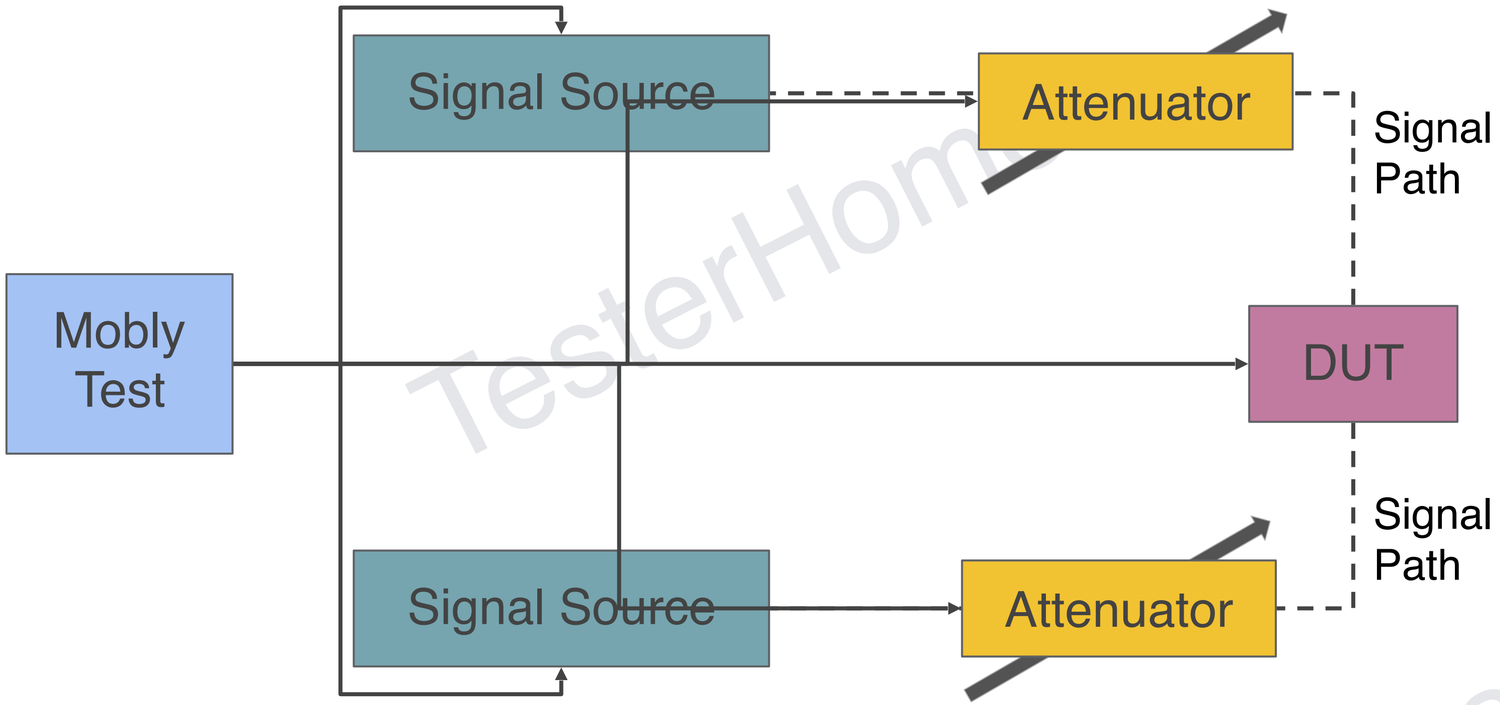
Simulating Roaming Mobility

- User moves around two signal sources
- Verify that the user device roams correctly

Roaming Mobility User Scenario



Roaming Mobility Setup Diagram

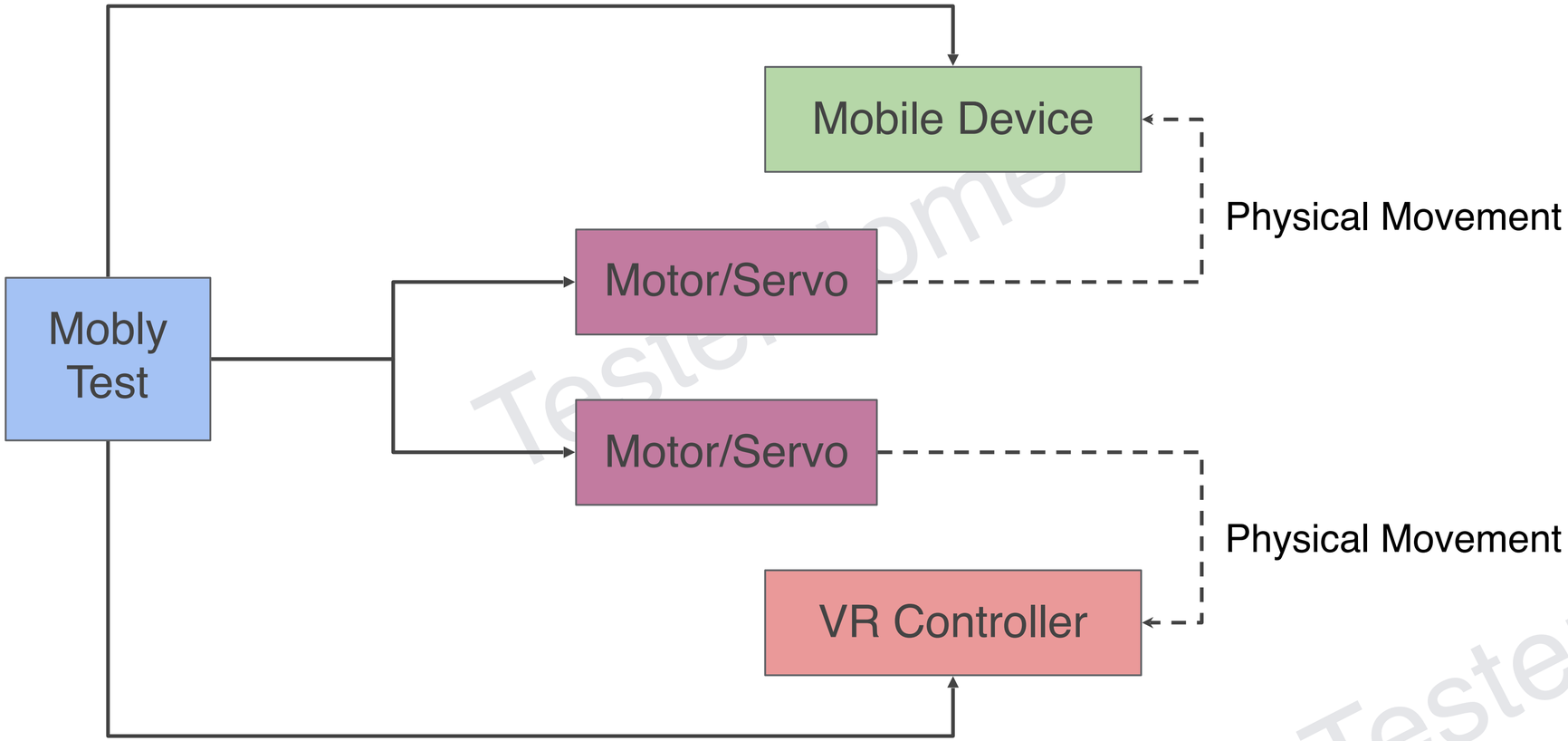


Roaming Mobility Setup

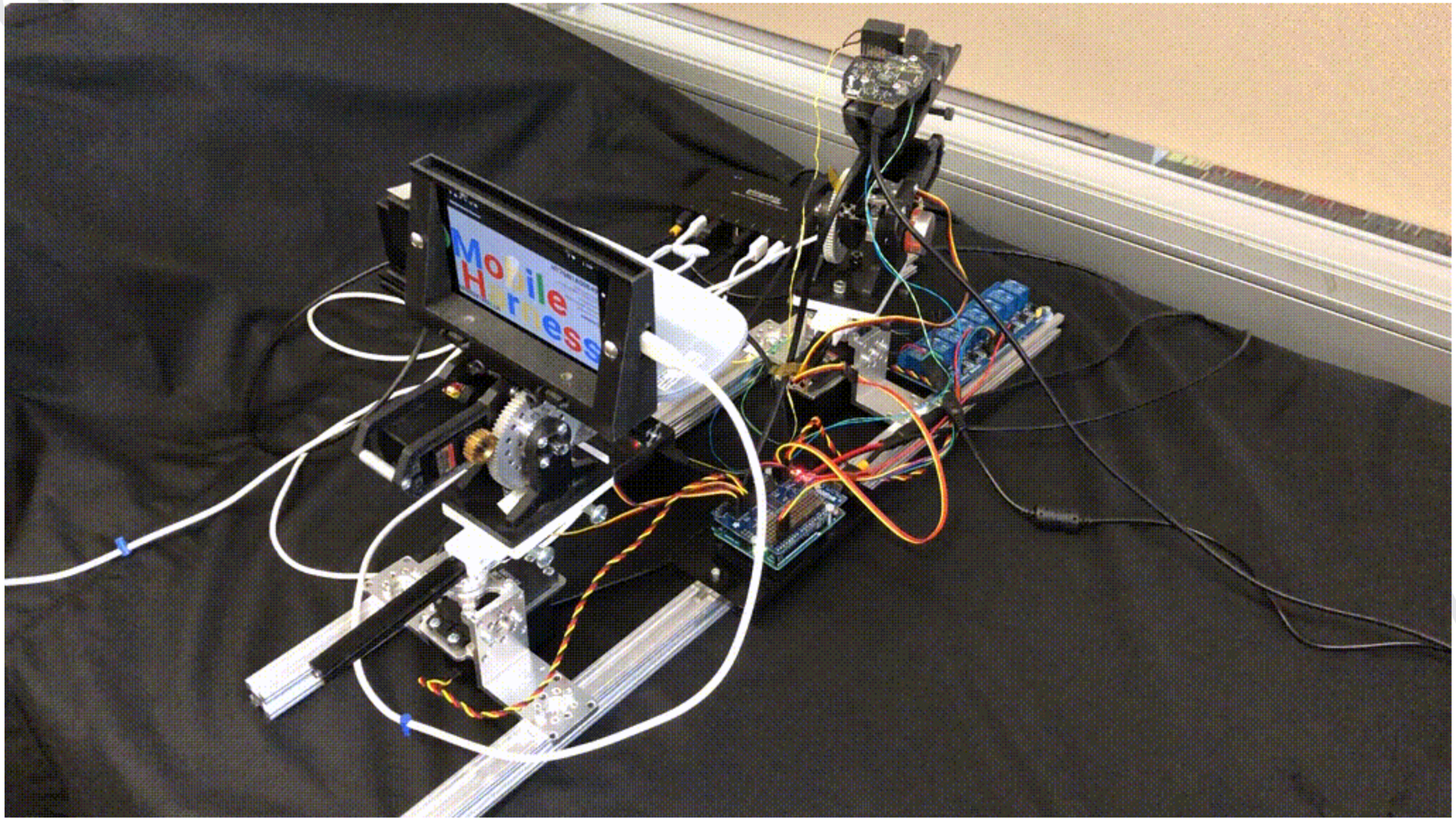
Example Use Case:

Simulating Physical Movement in Virtual Reality (VR)

VR Movement Setup Diagram



VR Movement Setup



Scalability & Integration

Widely Adopted in Alphabet

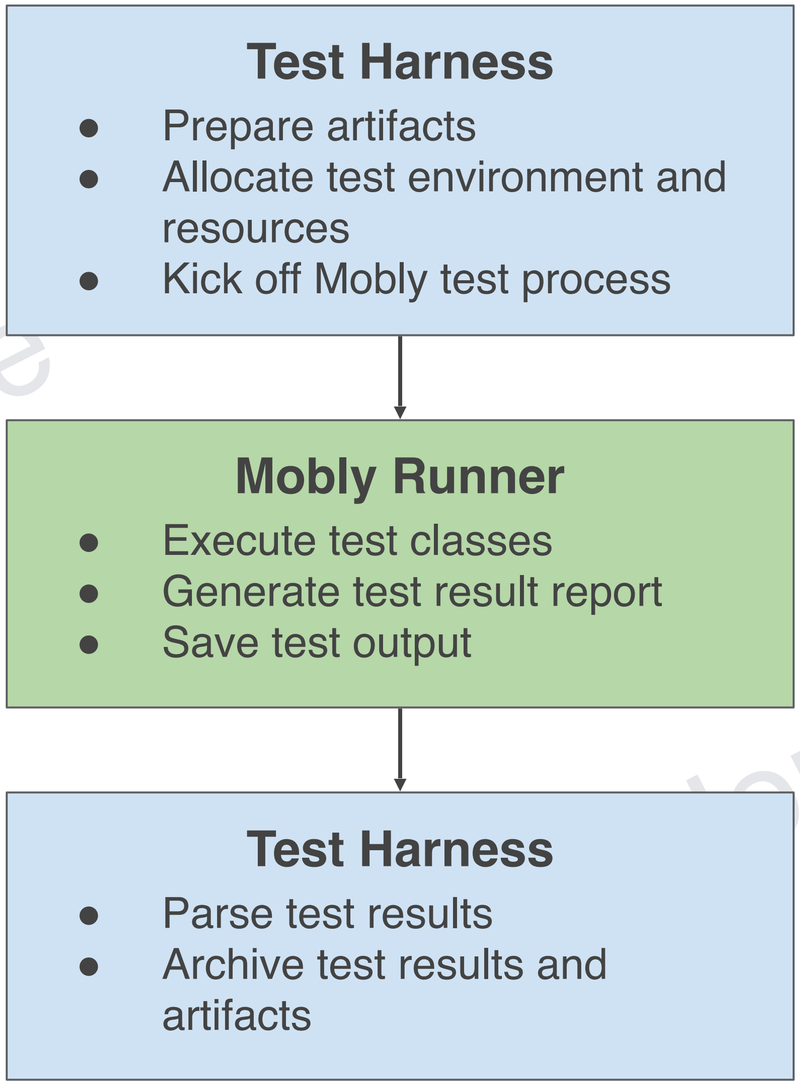
- Telecom
  - Fi
  - Messages
  - Duo
- Google Phone (Pixel)
  - Camera
  - Connectivity
  - Setup Wizard
- Android Platforms
  - Auto
  - Things
- VR
- AR/Lens
- Nest
- Loon
- ...

Over 4 Million test cases executed with Mobly monthly

Integration with Your Toolchain

Using Mobly as part of your test process is easy.

- Main Mobly components are all open sourced on github.
- Mobly framework has APIs designed for creating custom suites.
- Straightforward to integrate with larger scope test systems.



Recap

- Host-driven tests are good for covering complex user scenarios
- Non-mobile devices can be used to expand E2E coverage
- Mobly and associated libraries help you construct complex E2E tests for modern user scenarios
- Integrate Mobly with your internal toolchain
- Be creative!

Questions?

References

- Mobly Repo
- Mobly API Doc
- Mobly Snippet Lib Repo
- Mobly Bundled Snippets Repo

谢谢

THANKS