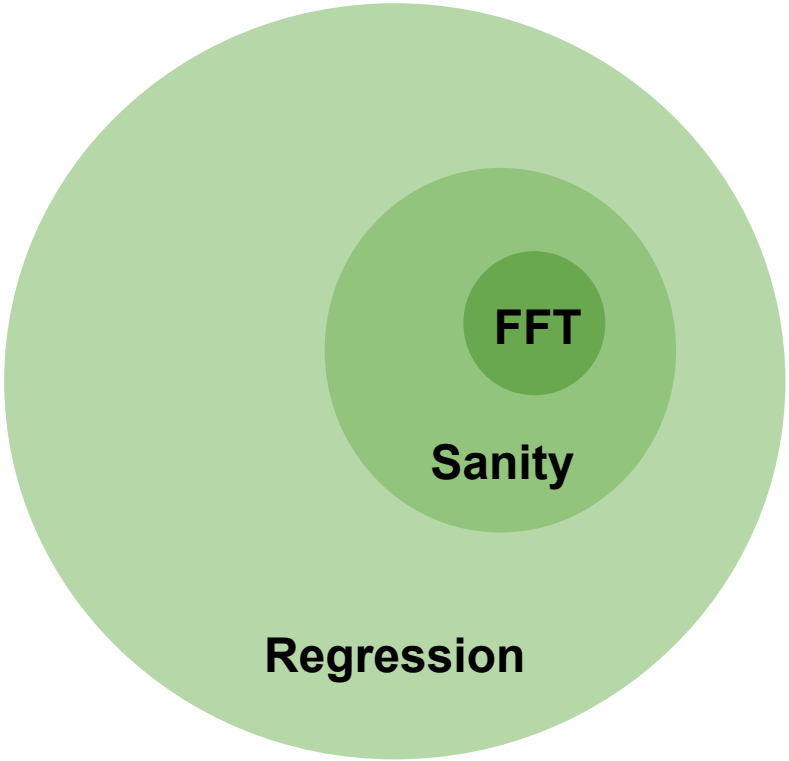
Optimizing Test Execution

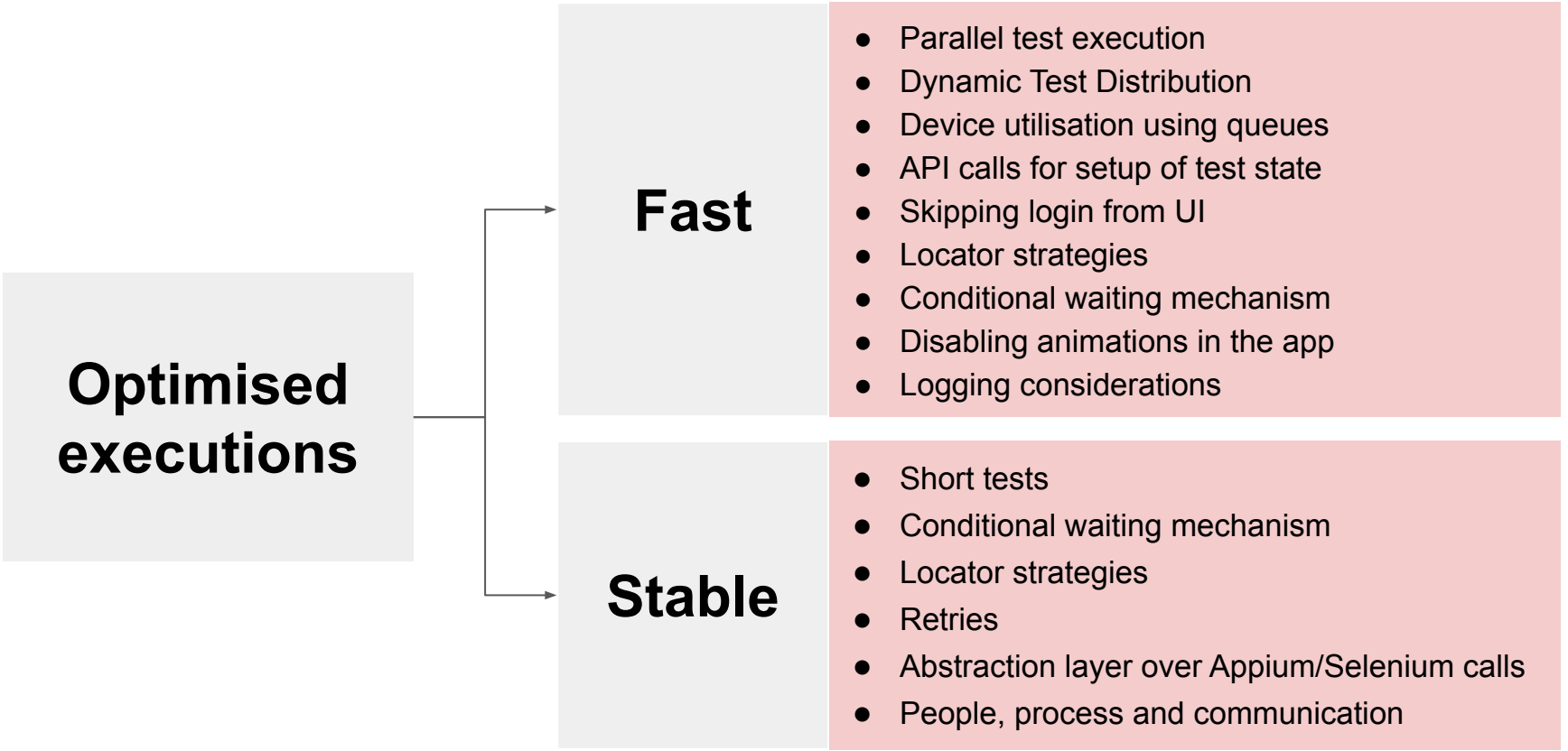
Abhijeet / Chia-Hung

Need for optimisation



~ 200 automated test cases

Solutions we adopted



Fast


Parallel test execution

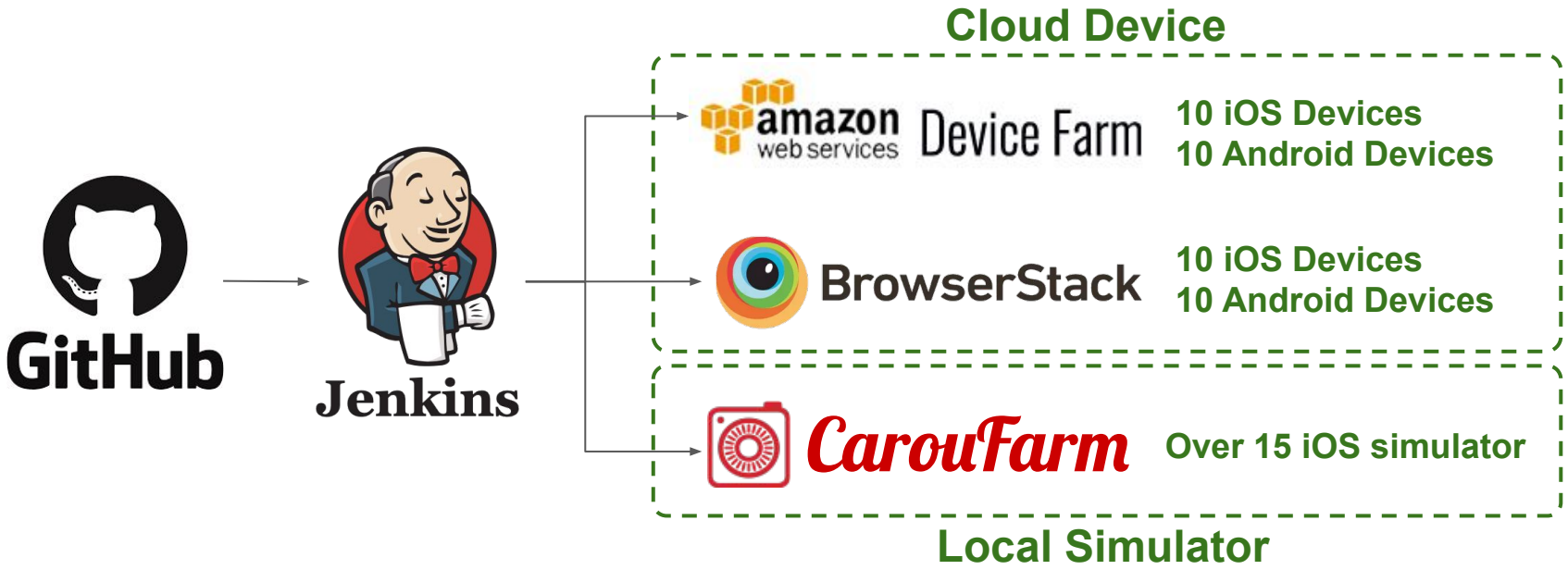
Parallel Test Execution

- Why do we need parallel test execution ?

  It's hard to use 1 device to test all test scenarios, we want to get the test result as soon as possible.

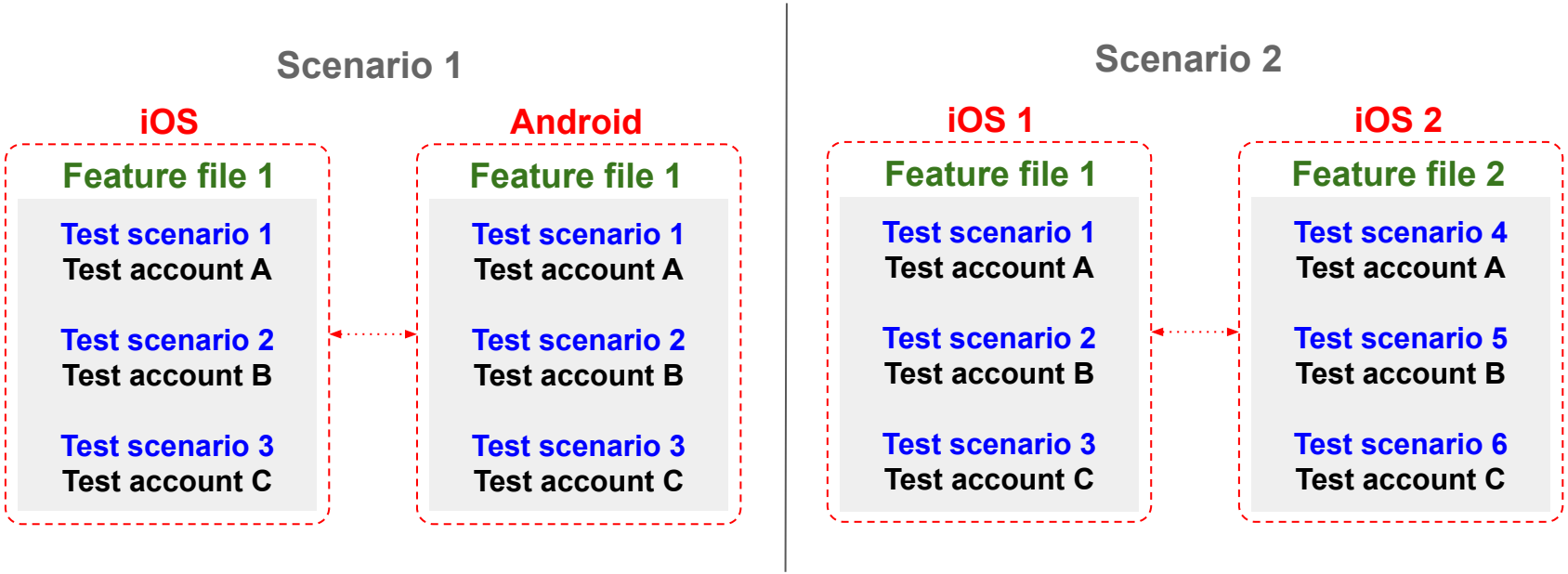
  ➢ **Our Solution - Cloud Device ? Local Device ?**

Our Parallel Test Execution



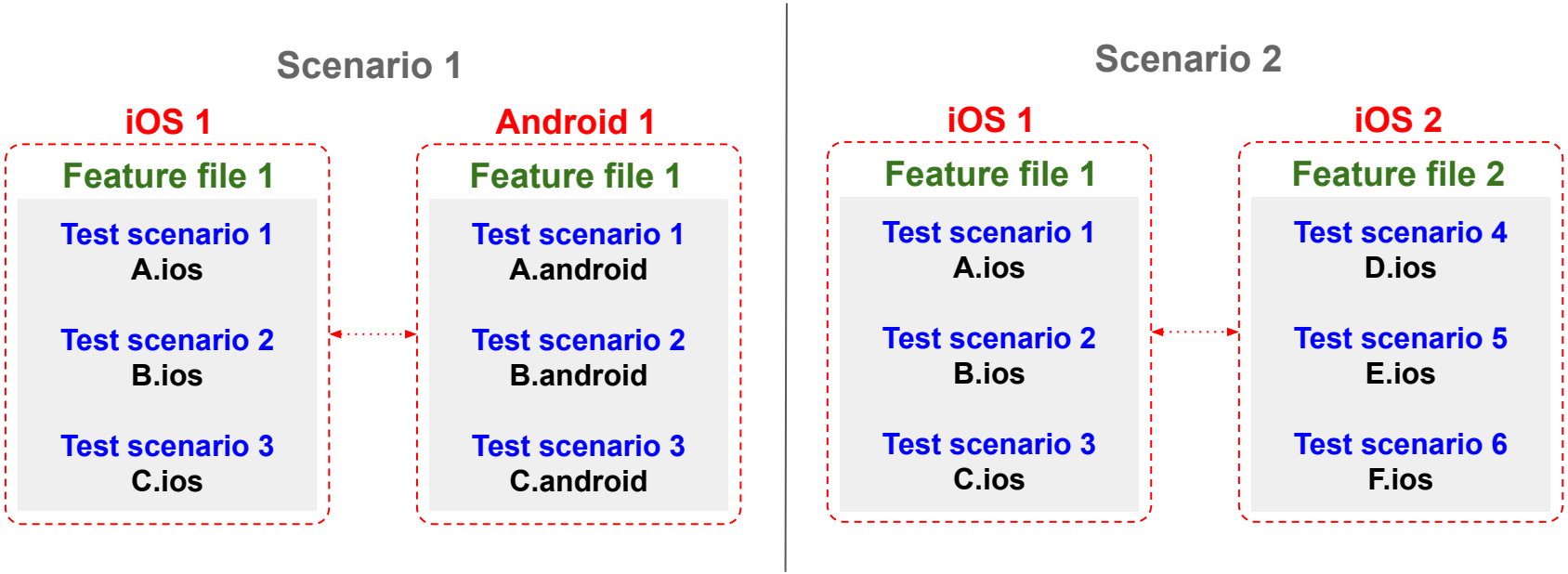
What's the challenges with parallel test execution ?

- **Problem 1**
  Sometimes test scenarios influence each other during the test execution.
  No matter triggering the test execution by the feature files or any tags.



What's the challenges with parallel test execution ?

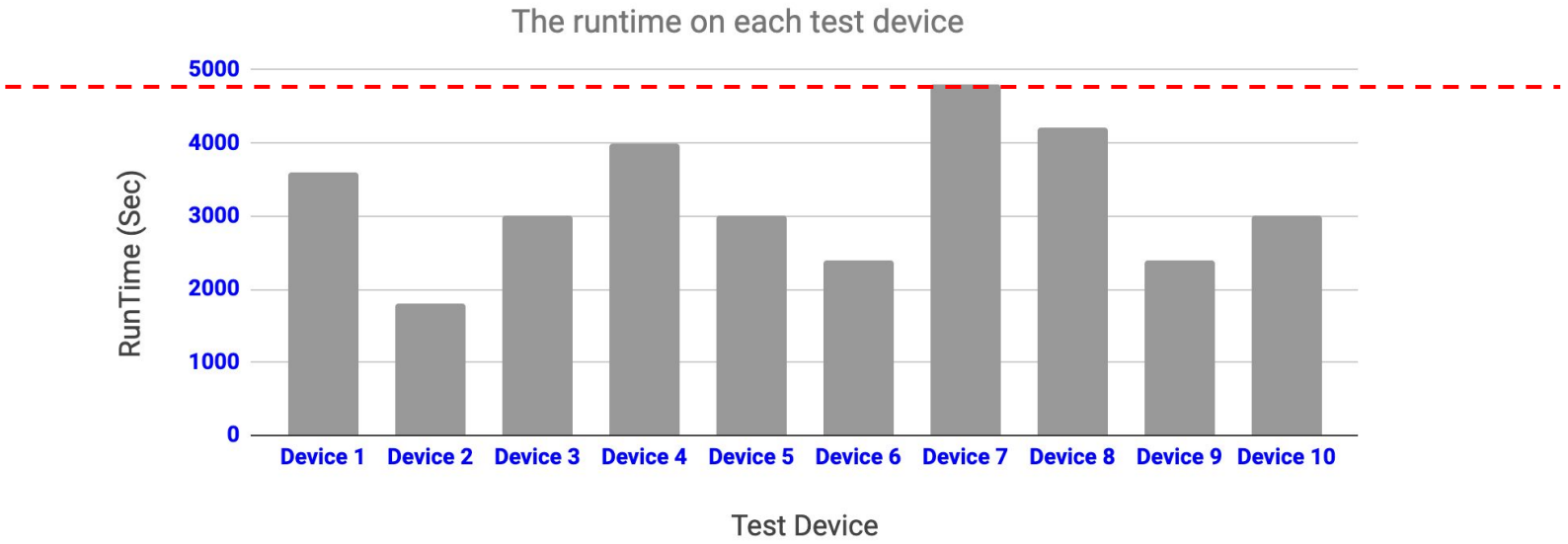
- **Solution**
  - Create Individual test account to each test scenario by platform.
  - Create a new test account for every run of every tests.



What's the challenges with parallel test execution ?

- **Problem 2**
  The runtime is different on each test device. It wastes a lot of time on the test execution.

Finish a test execution takes 80 mins, but the average execution time is about 53 mins.



What’s the challenges with parallel test execution ?

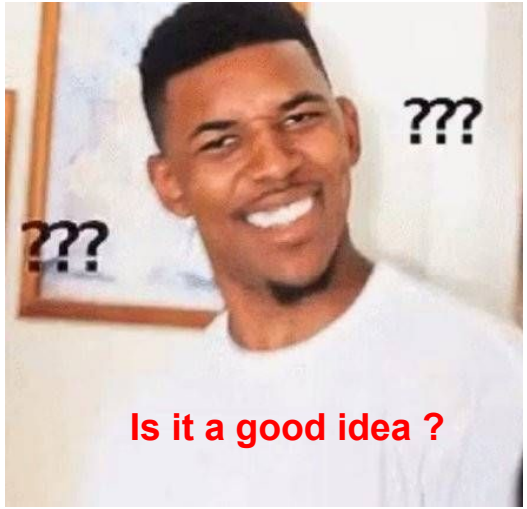
- **Common solution**
  Update the test case id of each test devices on the configuration file.

1. This way is manual, it needs a person monitoring the test execution time and updating the combination in the configuration file.

2. It can not support a different number of the test devices

execution.ini (Configuration)

```
[TestExecution]
Device1=10001,10002,10003
Device2=10004,10005,10006
Device3=10007,10008
Device4=10010,10011,10012
Device5=10013,10014,10009
...
```

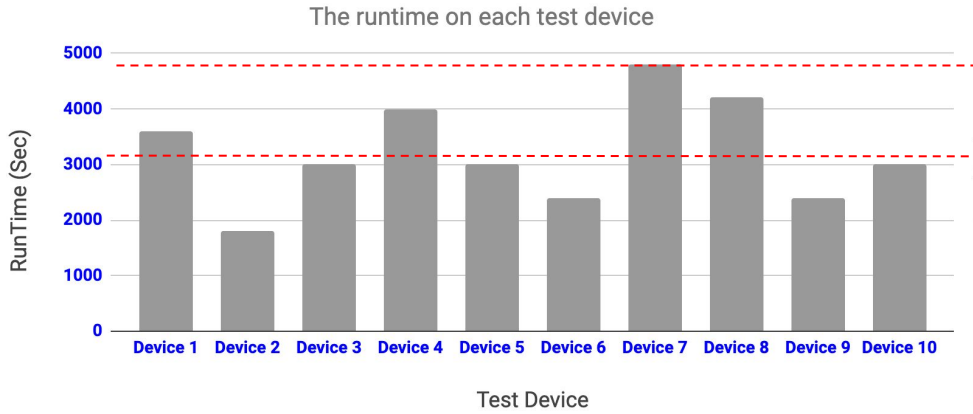


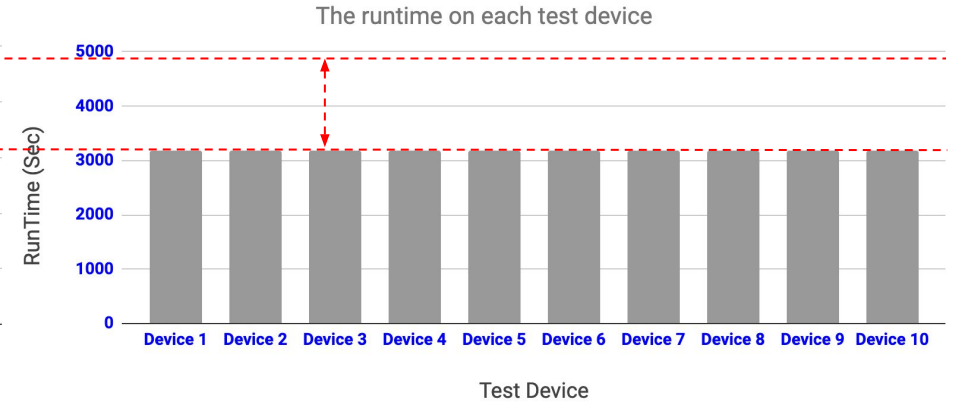

Dynamic Test Distribution

What is Dynamic Test Distribution?

- Our idea is to optimize the runtime on each test device.

Optimize the execution time from 80 mins to 53 mins.



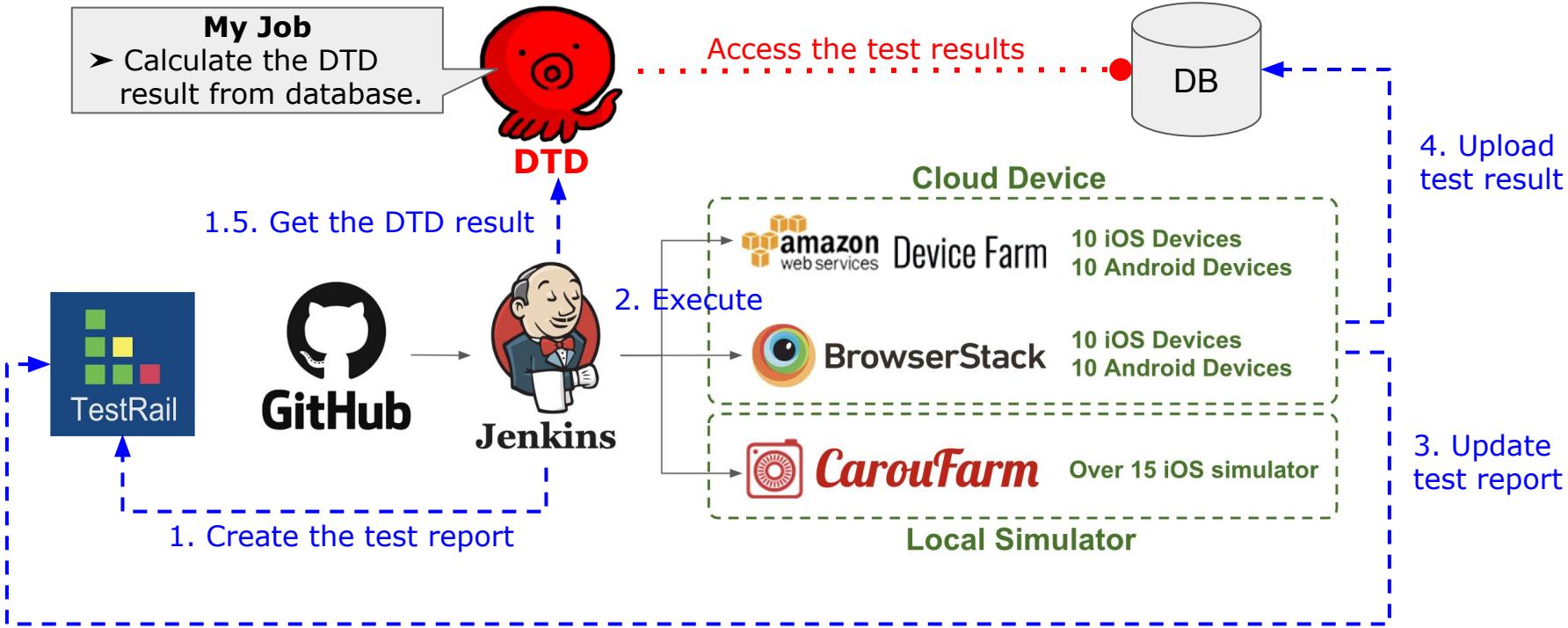


What’s the benefit of using Dynamic Test Distribution?

- Optimize the test execution time
- It supports a different number of the test devices
- No need to update and monitor the combination of test execution
- Remove the configuration of test execution
- Collect Test Data

How do we implement DTD?

- The imaginary scenario of parallel execution



How do we implement the DTD?

- What kind of data we would like to save ?

Unique Id	00fca869-51f5-4091-ae81-c4b30a8f7650
Cloud Provider	AWS, BrowserStack or CarouFarm
Platform	IOS or ANDROID
Scenario Id	Test Case ID 23168
Start Time	2019-05-01 06:05:35.920
End Time	2019-05-01 06:08:35.920
Duration	180
Test Result	FAILED, PASSED or UNDEFINED

How do we implement the DTD?

- Where do we save the test result ? AWS DynamoDB

Name

Create item | Actions

Scan: [Table] testResults: key

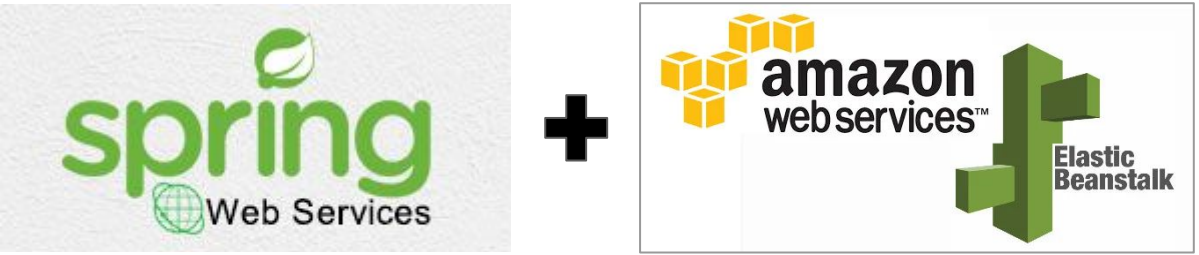
Viewing 1 to 100 items

key	cloudProvider	duration	platform	result	scenarioId	startTime
44f044d0-9138-4620-bade-9228efe5448c	browserstack	229	ANDROID	PASSED	15091	2019-05-02 10:15
475624df-2502-4869-9dc0-7801f9008d2d	browserstack	158	ANDROID	PASSED	14987	2019-05-03 11:05
47a92183-8d47-435e-b5ad-1dd3c3dd8999	browserstack	211	ANDROID	FAILED	21367	2019-04-29 09:16
481fddb9-c0bd-4430-8e55-d67612376306	browserstack	161	ANDROID	FAILED	23067	2019-04-29 16:06
48c81739-ae67-4ca8-b785-b03c63a90412	browserstack	128	ANDROID	PASSED	23083	2019-04-29 15:06
4975306a-72d9-4a4b-9639-6d033c5521fc	browserstack	189	ANDROID	PASSED	21401	2019-05-02 07:15
4988c433-eef7-4475-bff2-1ae3010748c2	browserstack	836	IOS	FAILED	15153	2019-04-30 44:12
4a121089-6a2d-4b19-9f81-767288962eb1	browserstack	160	ANDROID	PASSED	23172	2019-04-29 06:06
4a8de370-26bd-42e9-9518-aabf0cdf9a04	browserstack	208	IOS	PASSED	23073	2019-05-03 31:04
4af40d61-2dcb-4034-9dae-23c73021438a	browserstack	107	ANDROID	FAILED	15145	2019-04-29 22:06
51be1bae-ee67-46cb-ae6f-8e8747308d4c	browserstack	109	IOS	UNDEFINED	15073	2019-04-30 10:12
51f1d6f5-df09-441c-b398-cb1e365051b1	browserstack	137	ANDROID	PASSED	10108	2019-04-29 31:06

How do we implement the DTD?

- How do we get the DTD result? Script? API?

  We choose to implement an API endpoint to provide the DTD result.



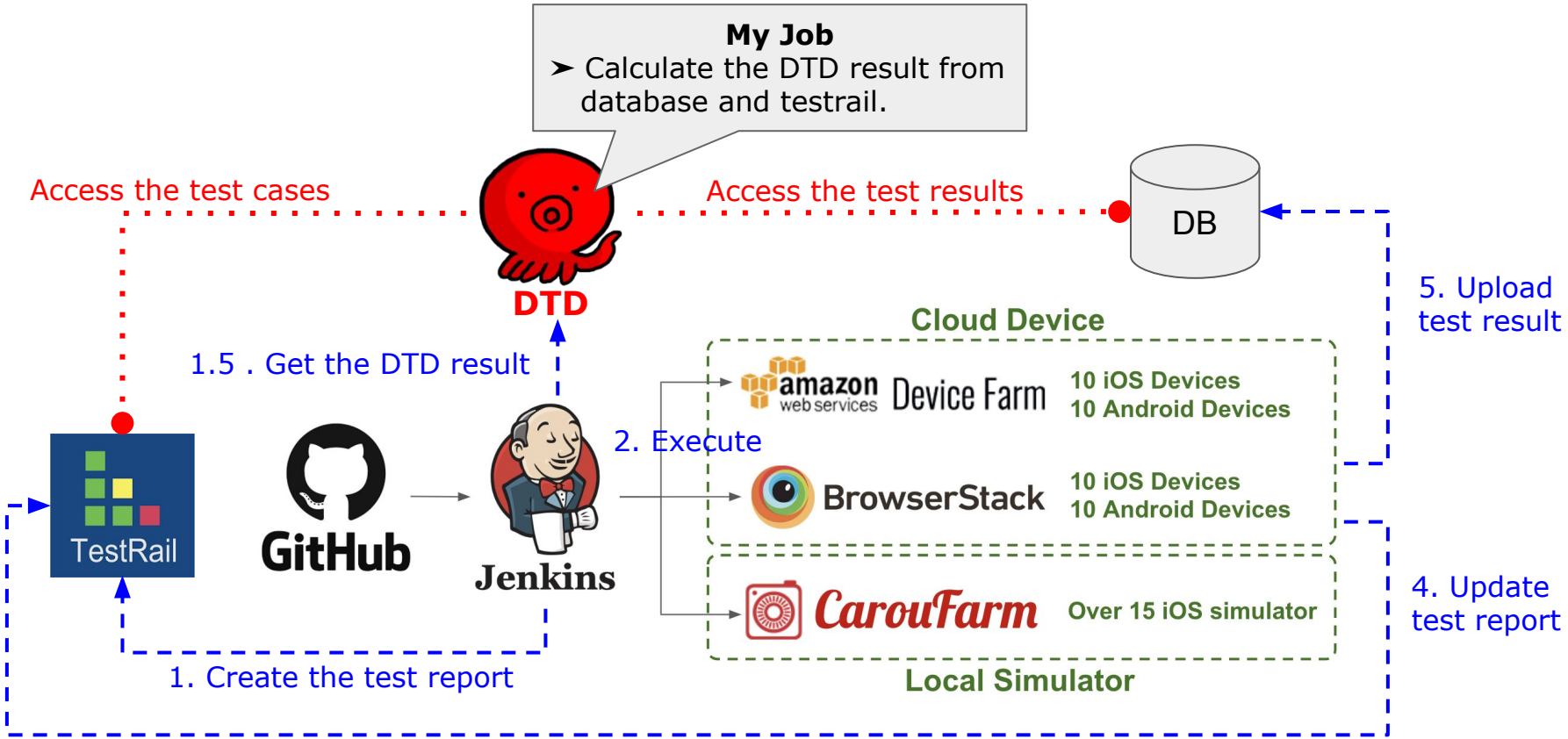
Our solution is not the only solution to implement an API endpoint.

Ex: Flask, Hug...etc

What’s the problem of the DTD during implementation?

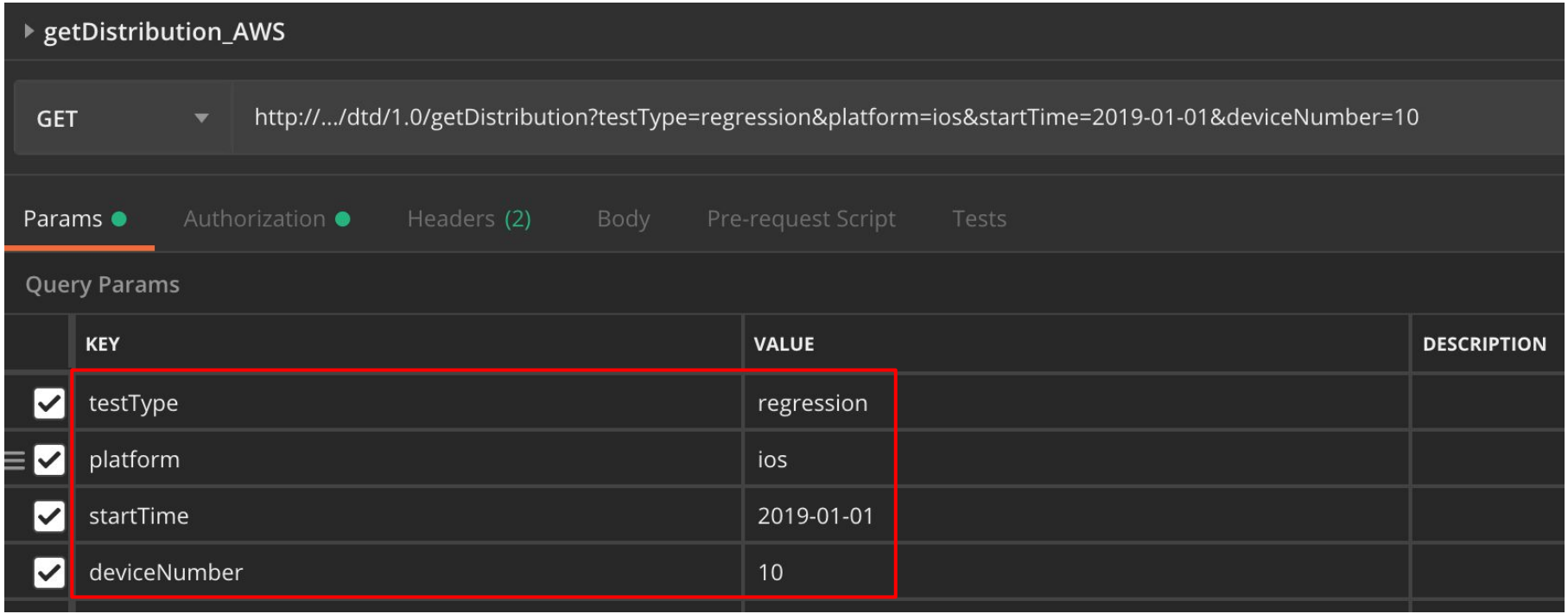
- Stability of DTD
- Unused test result, ex: scenario id is 0, duration time is 0.
- New test cases?

New Test Case?



The input of the DTD

- Run iOS regression test on 10 iOS test devices. (data: 2019-01-01 to now)



The output of the DTD

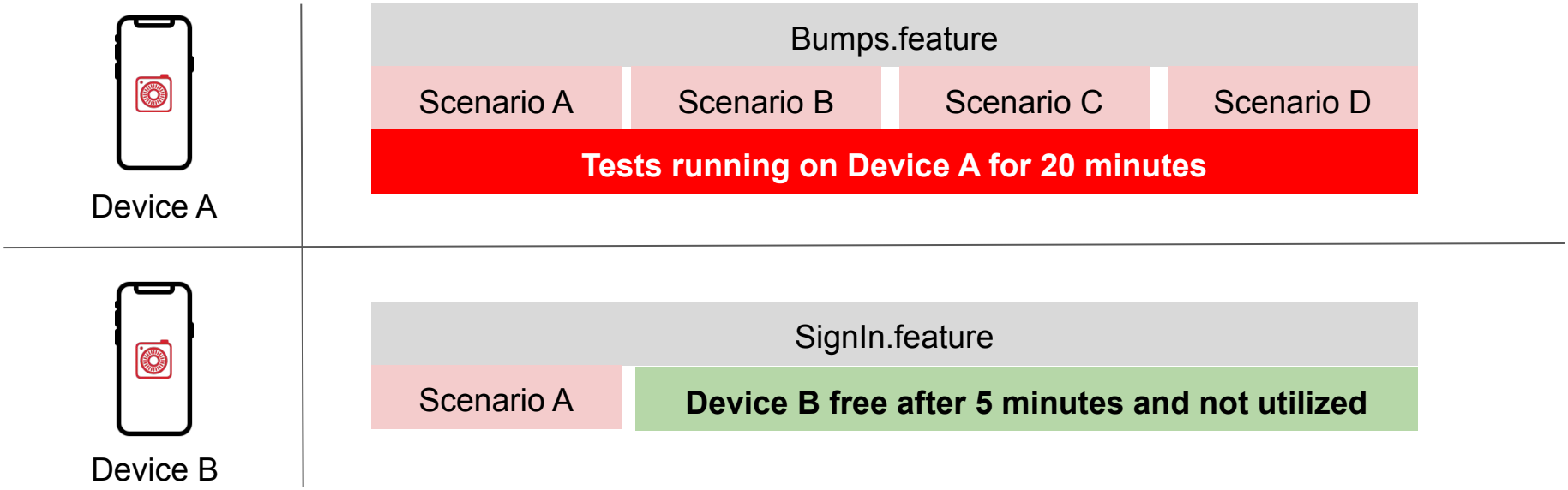
- iOS Device1 will execute 13 test cases and expected test secs is 3548.


```
{
  "1" : {
    "deviceName" : "1",
    "testCaseIds" : [ 15153, 23127, 23170, 23102, 23067, 23109, 22338, 15026, 23087, 23119, 14911, 23093, 20052 ],
    "expectedTestSecs" : 3548,
    "testCaseNumber" : 13
  },
  "2" : {
    "deviceName" : "2",
    "testCaseIds" : [ 21384, 23114, 23151, 19166, 23066, 21396, 23121, 23073, 15117, 23163, 21388, 23068, 15073 ],
    "expectedTestSecs" : 3526,
    "testCaseNumber" : 13
  },
  "3" : {
    "deviceName" : "3",
    "testCaseIds" : [ 23149, 22816, 23171, 15149, 14969, 23090, 15147, 23159, 23070, 23074, 10108, 23071, 20059 ],
    "expectedTestSecs" : 3462,
    "testCaseNumber" : 13
  },
  "4" : {
    "deviceName" : "4",
    "testCaseIds" : [ 14970, 17585, 22578, 14980, 23088, 23076, 20041, 21381, 21366, 15145, 23092, 23165, 21402 ],
```


Device utilisation using
queues

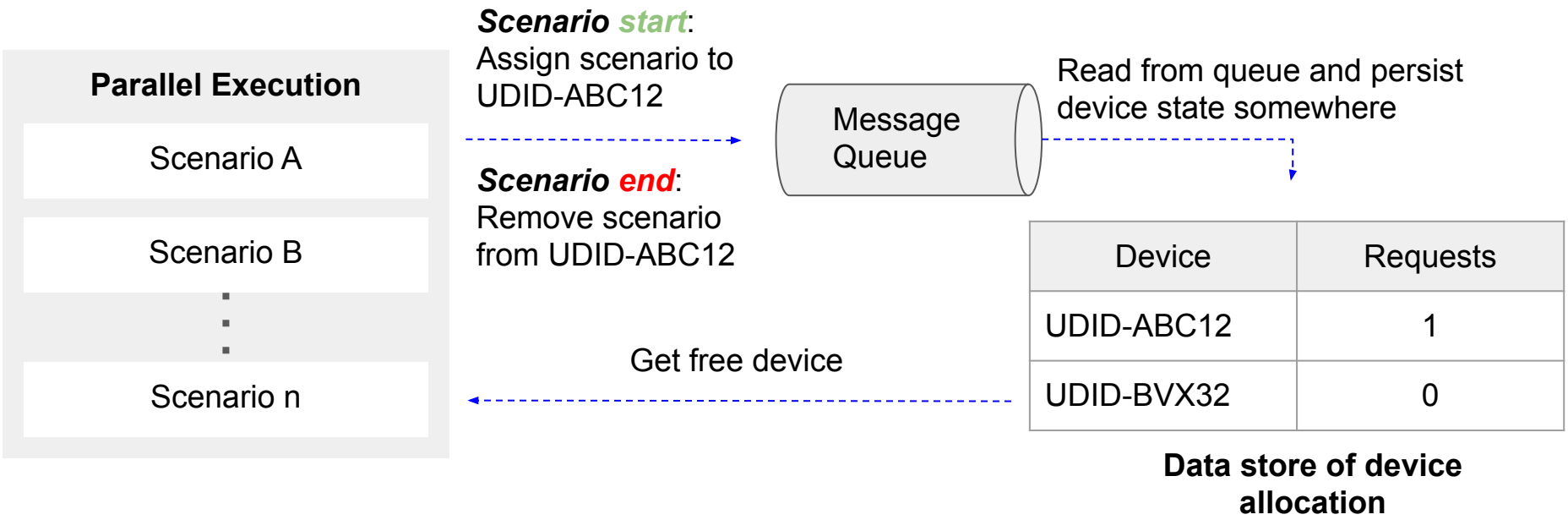
Device utilisation using Queues

Problem faced



Device utilisation using Queues

High Level Solution



Free device = Device which has no queued scenarios or least number of queued scenarios

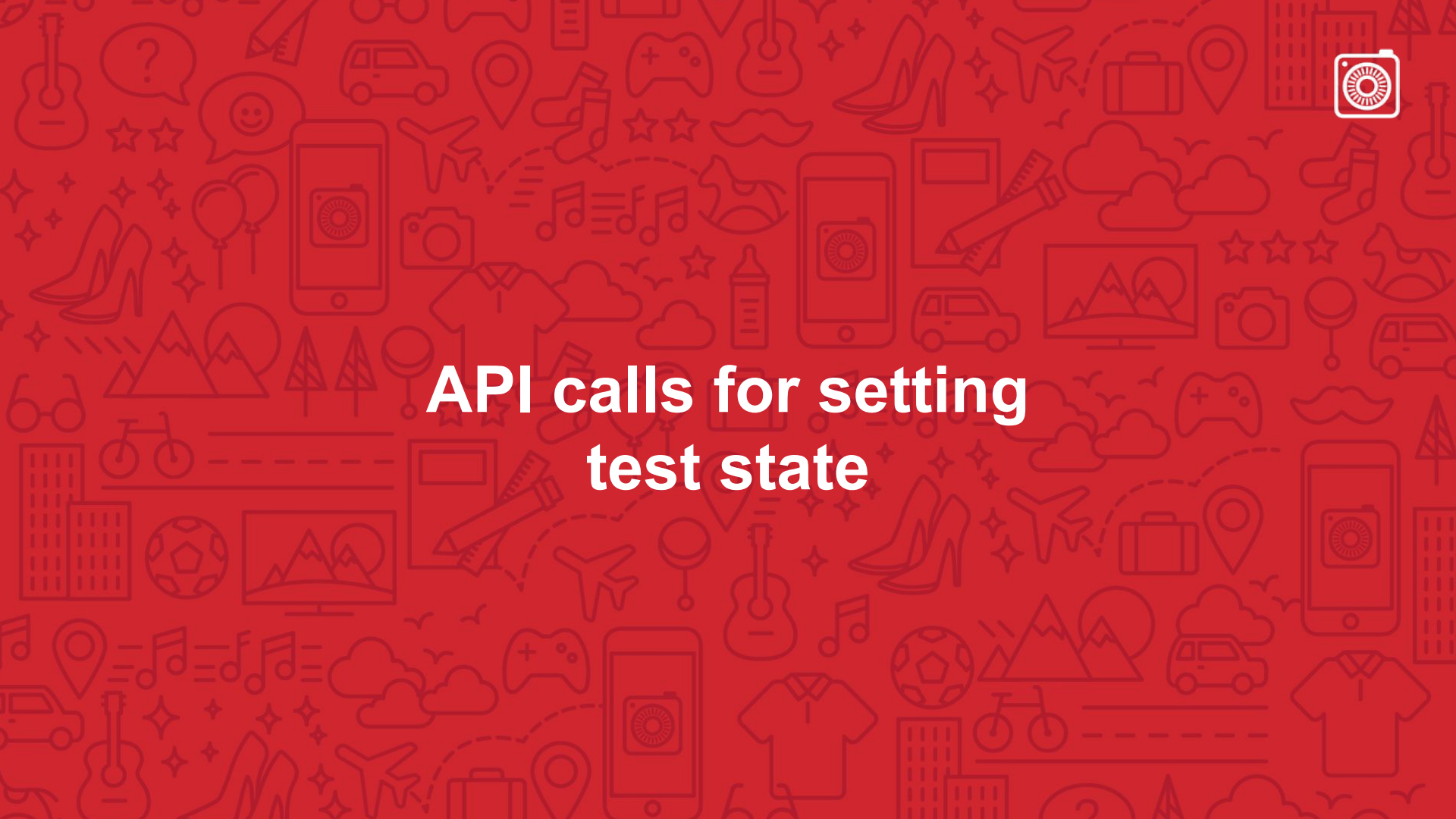
Device utilisation using Queues - Implementation

Phase 1
Using Amazon SQS and DynamoDB

- Used Amazon SQS (Simple Queue Service) as a message queue to push scenarios to queue for every new scenario run
- Amazon DynamoDB was used as data store for state of devices
- Used Java SDK for both SQS and DynamoDB to consume queue messages and write state to DynamoDB

Phase 2
Using modified version of Selenium Grid servlet

- Used Selenium Grid's own queueing system
- Extended `RegistryBasedServlet` to get free device information


API calls for setting
test state

API calls for setting test state

Scenario: Purchasing a promotion for a listing on Carousell from UI

1. Login
2. Tap on Sell button
3. Select category to list in.
4. Fill all the mandatory fields
5. Tap on List button
6. Select promotion
7. Confirm promotion
8. Verify

- Creating a listing is a step for setting up the test data.
- Execution from UI takes around 2 minutes using Selenium/Appium
- Introduces flakiness

How can we optimise this?

API calls for setting test state

Optimised scenario: Purchasing a promotion for a listing on Carousell

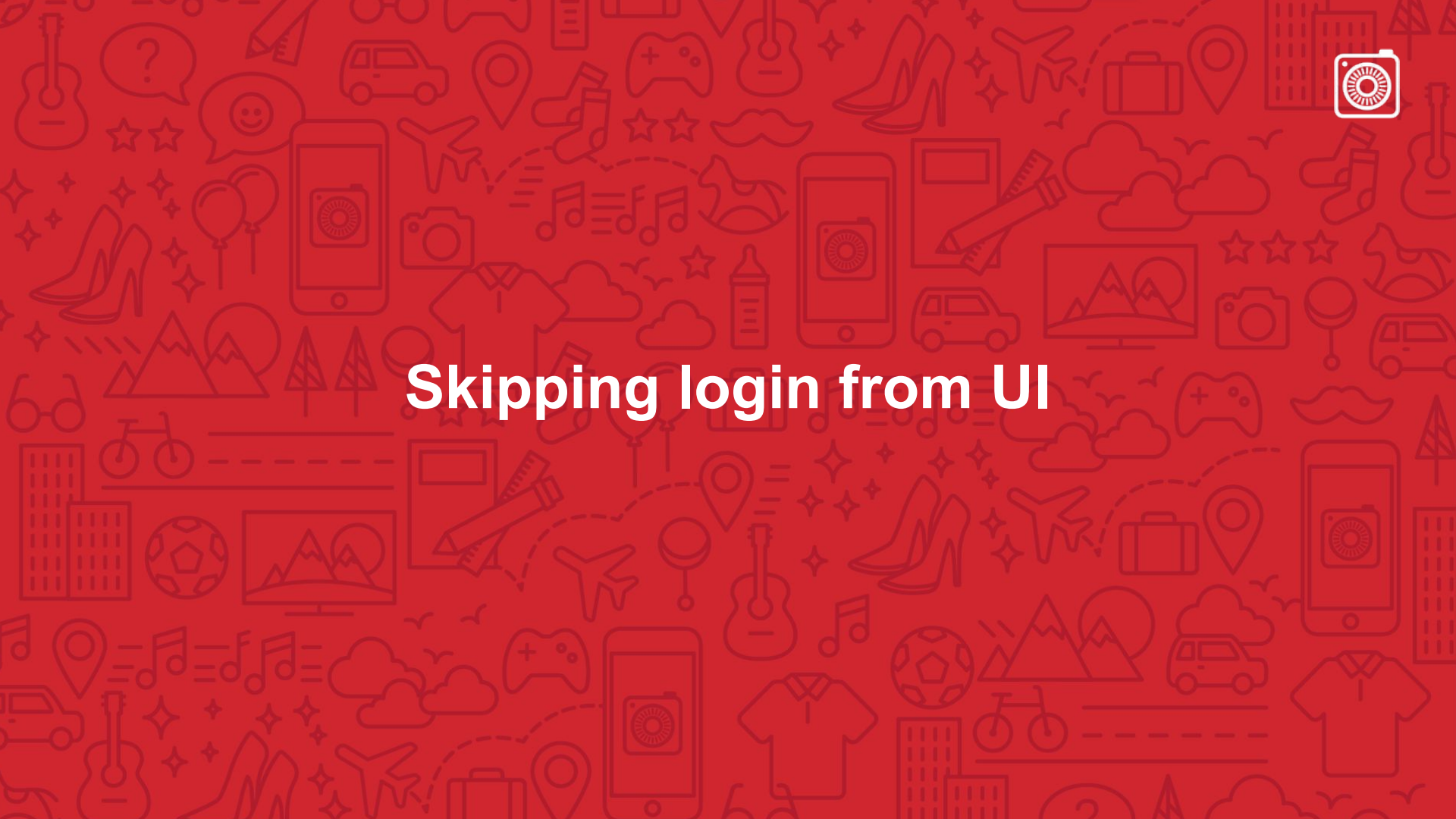
1. Login
2. Create a listing using API
3. Go to profile page
4. Select promotion for the listing
5. Confirm promotion
6. Verify

Time taken to execute API for creating listing = 5 seconds!

API calls for setting test state

```
@case(14980)
Scenario: User should be able to see urgent bumped listing as bumped in search results
  Given the user "bumps_user" lists an item in the "Magazines & Others" via api
  And the user logs in as "bumps_user" by skipping ui login
  When the user navigates to me page
  And the user purchases a "urgent" bump
  And the user navigates to category search results from listing details page
  Then the user sees listing bumped in search results
```

```
@Given("^the user \"([^\"]*)\" lists an item in the \"([^\"]*)\" via api$")
  public void theUserListsAnItemViaApi(String userName, String subCategory) {
    User user = userService.get(userName);
    sellerApiUser.setUserAccount(user.getUsername(), user.getPassword());
    sellerApiUser.createARandomItemInTheSubCategory(subCategory,
Base.SG_MARKET_CODE);
  }
```


Skipping login from UI

Skipping login from UI

Problem

- Every scenario runs as a new Appium session (We don’t re-use sessions between testcases)
- Almost every scenario requires the login step
- Login step takes around 1 minute to execute from the UI
- **100 testcases X 25 seconds = 41 minutes**

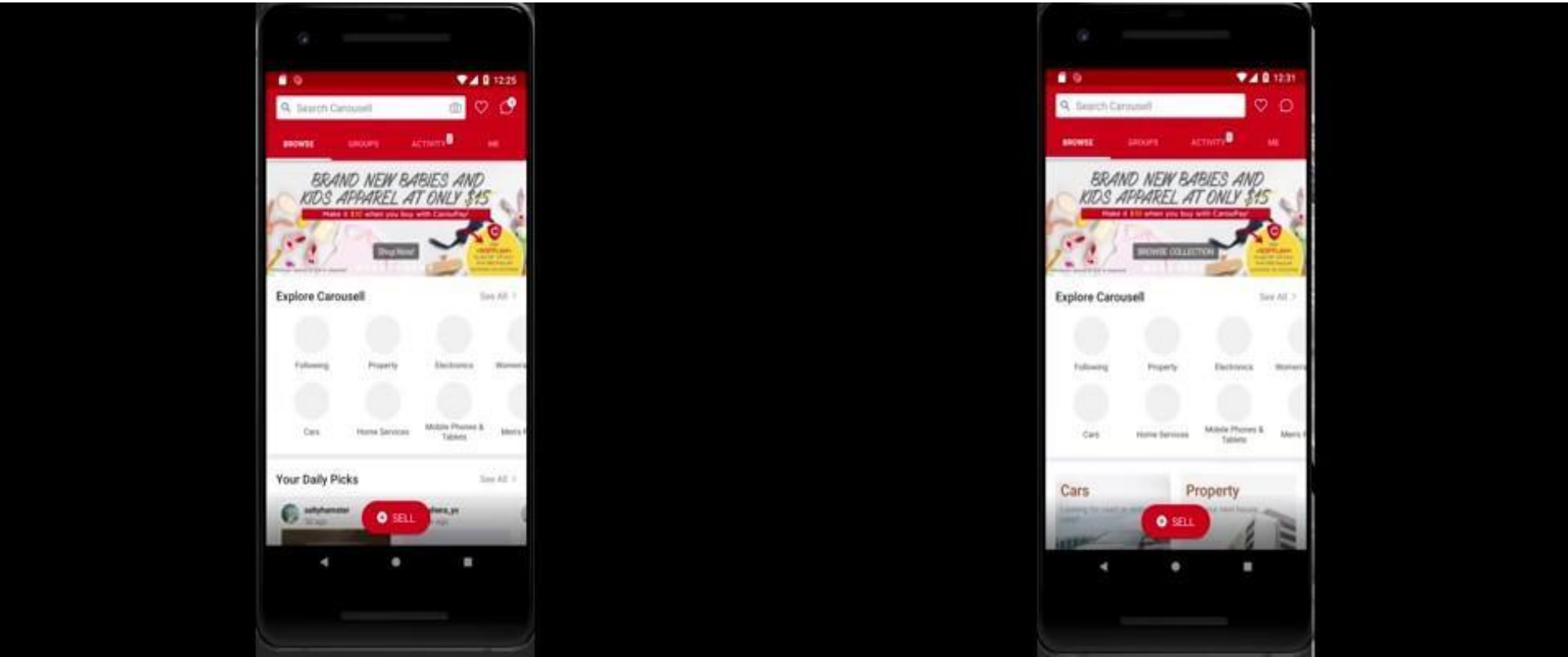
Solution

- Skip login and navigate directly to the desired screen (in our case it was home screen)
- Save time!

Skipping login from UI

Test execution with normal login

Test execution by skipping login



Skipping login from UI

Changes Required

- Use new Appium desired capabilities:
  - `optionalIntentArguments` for Android app
  - `processArguments` for iOS app

- Implement custom logic to skip login in test builds (collaboration with developers)

Skipping login from UI

Android Appium example:

```
private void initWebDriverForAndroid(String username, String password, URL
appiumUrl) {

if (username != null && !username.isEmpty()) {
      capabilitiesProvider
          .getCapabilities()
          .setCapability(
              "optionalIntentArguments",
              String.format("-e \"username\" \"%s\" -e \"password\" \"%s\"",
username, password));
    // perform login in background right after starting

    driver = new AndroidDriver(appiumUrl, capabilitiesProvider.getCapabilities());
  }
```

Skipping login from UI

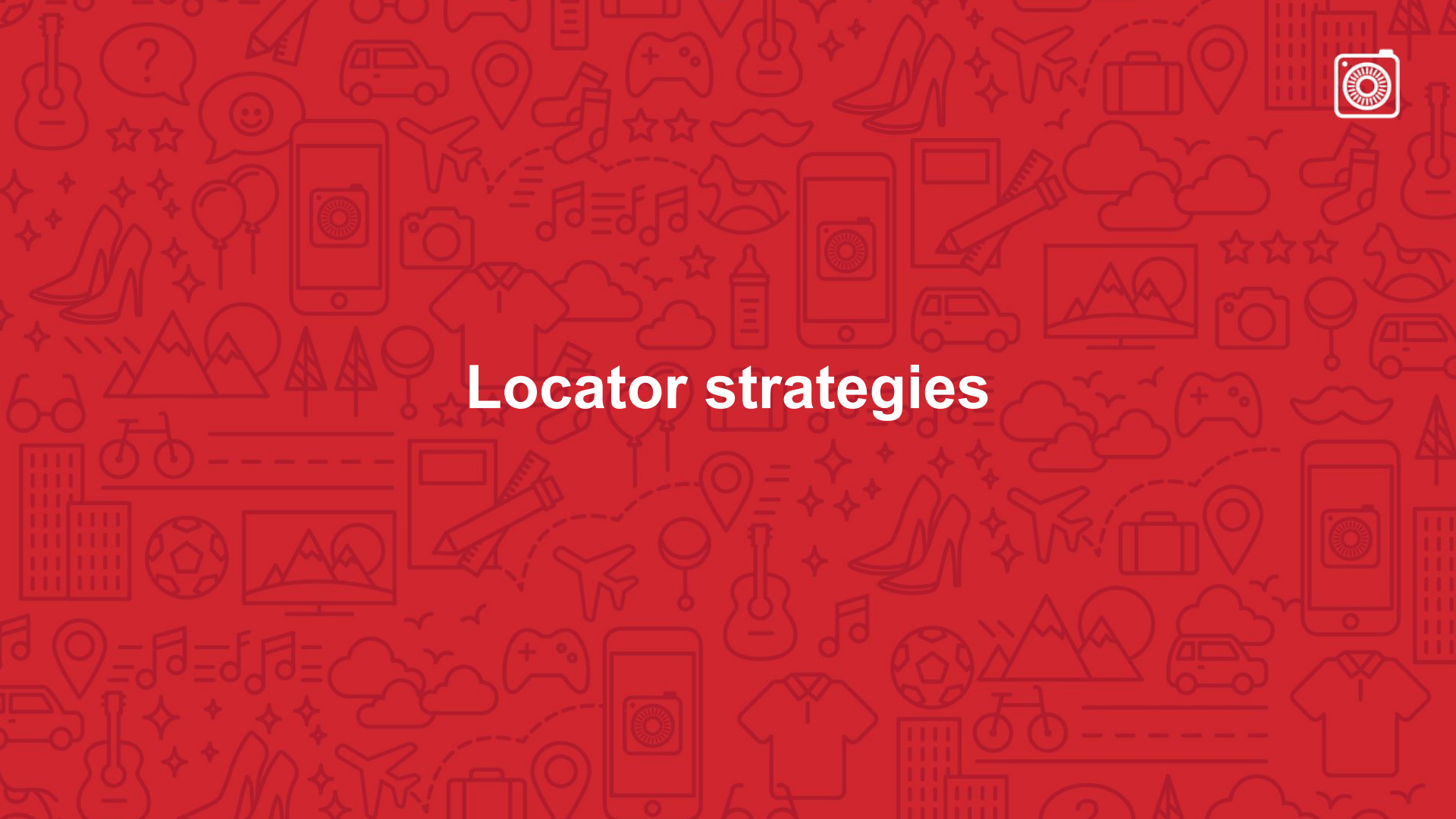
iOS Appium example:

```
private void initWebDriverForIOS(String username, String password, URL appiumUrl) {

    if (username != null && !username.isEmpty()) {
      capabilitiesProvider
          .getCapabilities()
          .setCapability(
              "processArguments", String.format("{\"args\": [\"%s\",\"%s\"]}",
username, password));
    }
// perform login in background right after starting IOSDriver


    driver = new IOSDriver<>(appiumUrl, capabilitiesProvider. getCapabilities());

}
```


Locator strategies

Locator strategies for fast scripts

Preference 1

Accessibility ID or ID

```
@AndroidFindBy(id = APP_PACKAGE + ":id/button_action")
@iOSXCUITFindBy(accessibility = "Yes")
protected WebElement confirmBumpButton;
```

id - resource id for Android and name for iOS

accessibility id - content description for Android and accessibility id for iOS

Locator strategies for fast scripts

Preference 2 (Android)

UISelector

```
@AndroidFindBy(uiAutomator = "new UiSelector().text(\"Your Bump is reserved.\")")
private WebElement reservedBumpMessage;
```

Preference 2 (iOS)

NSPredicate or iOSClassChain

```
@iOSXCUITFindBy(iOSNsPredicate = "type = 'XCUIElementTypeButton' and name contains 'For Rent'")
protected WebElement forRentTab;

@iOSXCUITFindBy(
      iOSClassChain = "**/XCUIElementTypeOther[`name CONTAINS[cd] \"product_card_time_label\"`]")
List<WebElement> listingCardsTimeLabels;
```

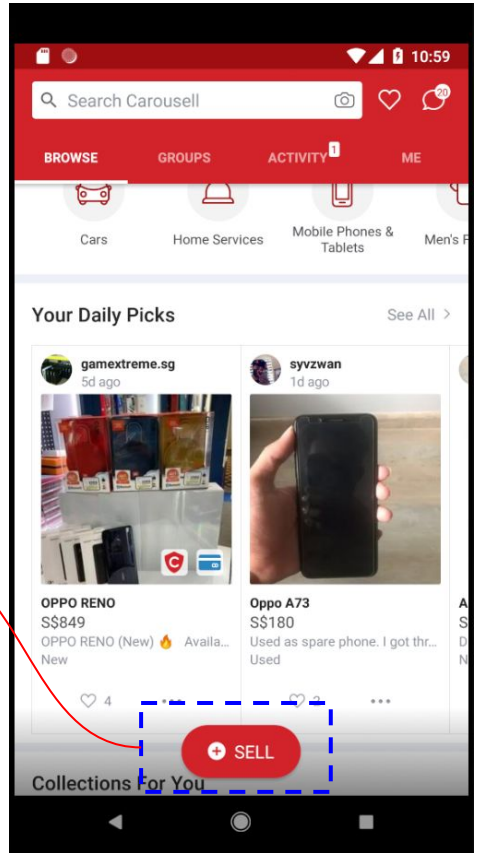

Conditional waiting
mechanism

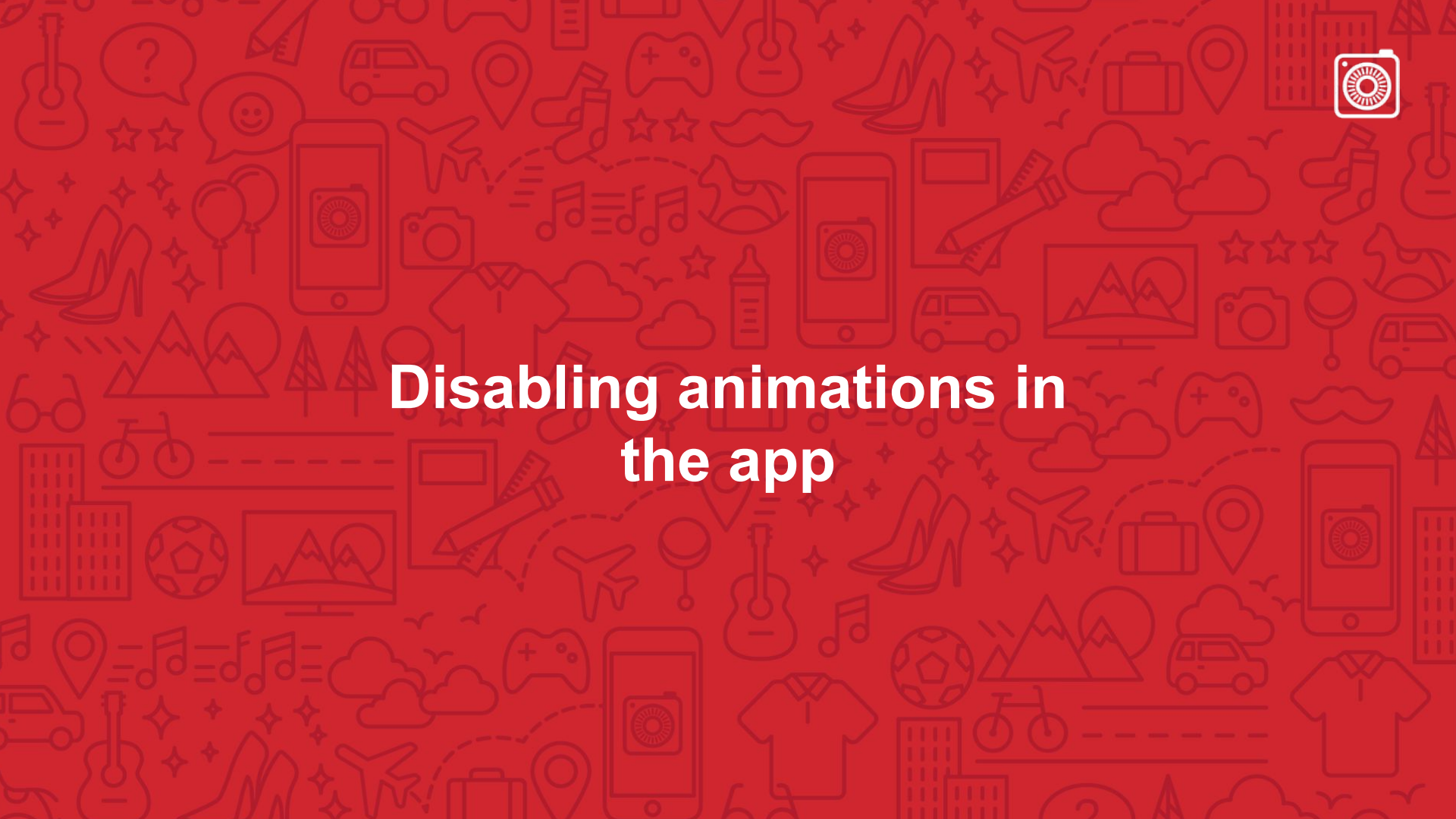
Conditional waiting mechanism for **fast** tests

Conditional waiting for checking whether an element is present/visible on the screen or not.

NO `thread.sleep()!`

```
try {
      wait = new WebDriverWait(getDriver(), timeOut);
      element =
wait.until(ExpectedConditions.visibilityOf(element));
      LOG.info("Leaving waitForElement(WebElement
element={},int timeOut={})", element, timeOut);
      return element;
    }
catch (TimeoutException | ElementNotVisibleException |
NoSuchElementException e)
{
      throw new CarouLionException("Not able to find the
element " + element, e);
}
```




Disabling animations in
the app

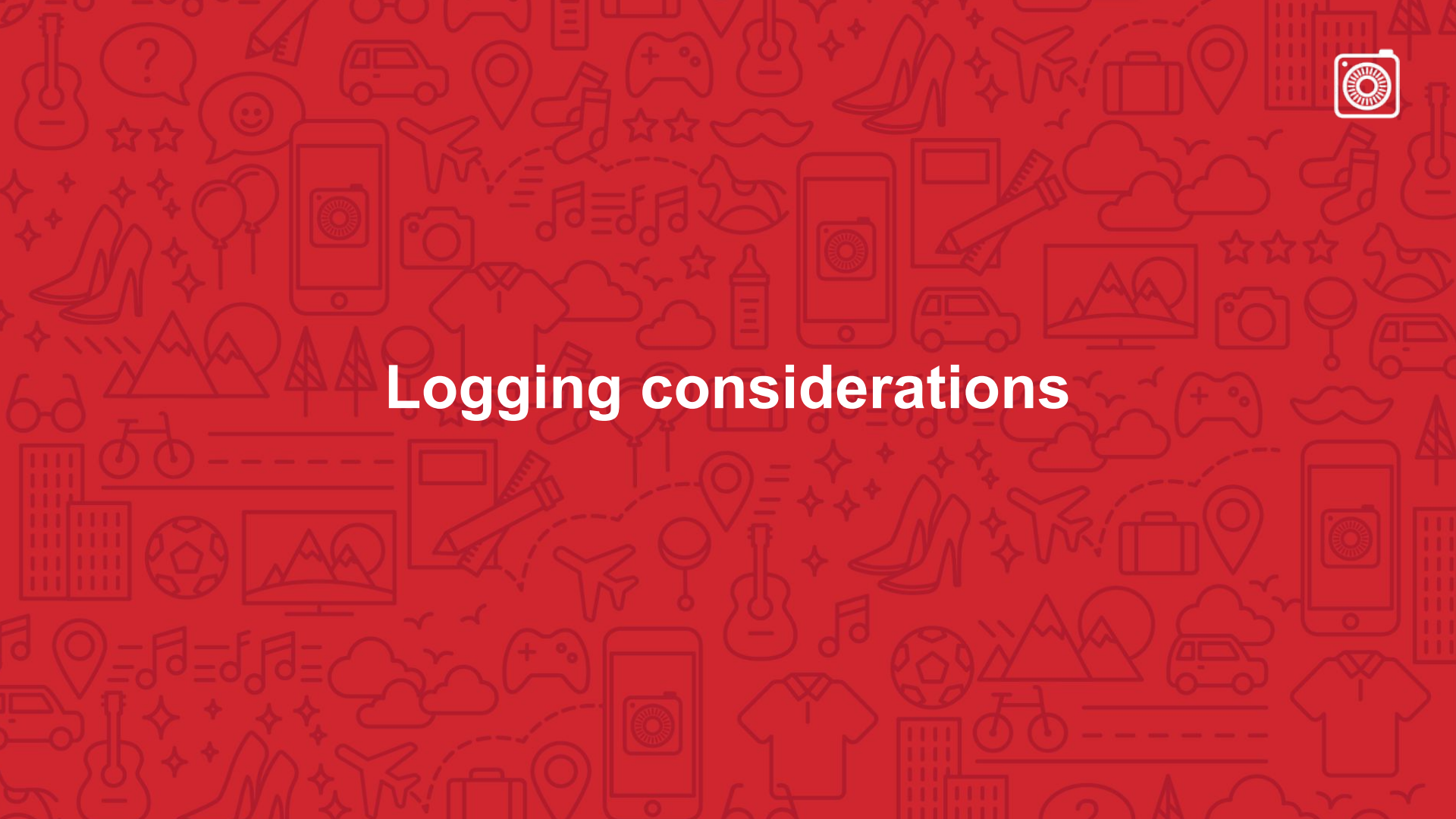
Disabling animations on the app

For Android

```
adb shell settings put global window_animation_scale 0
adb shell settings put global transition_animation_scale 0
adb shell settings put global animator_duration_scale 0
```

Why?

- Animations take time to finish.
- Brings in flakiness.
- Animations consume more CPU


Logging considerations

Log only what is needed. Not just everything.

```
@Given("^the user logs in as this new user account$")
  public void userLogsInAsNewUserAccount() {
    startDriver();
    LOG.info("Entering iLoginAsNewUserAccount()");
    if (userService.getCurrentUser() == null) {
      throw new CarouLionException("[iLoginAsNewUserAccount] Cannot find current user object");
    }
    welcomePageObject = page(WelcomePage.class);
    LOG.info("Clicking on login");
    loginPageObject = welcomePageObject.clickOnLogin();
    LOG.info("Performing login");
    newHomePageObject =
        loginPageObject.login(
            userService.getCurrentUser().getUsername(),
userService.getCurrentUser().getPassword());
    LOG.info("Waiting for home page to load");
    newHomePageObject.waitUntilNewHomePageLoaded();
    LOG.info("Leaving iLoginAsNewUserAccount()");
  }
```

Avoid logging webdriver response

```
LOG.info(driver().getPageSource());
```

Consider using different log levels instead for such situations.

For example:

```
LOG.debug()
LOG.trace()
```

Stable


Short Tests

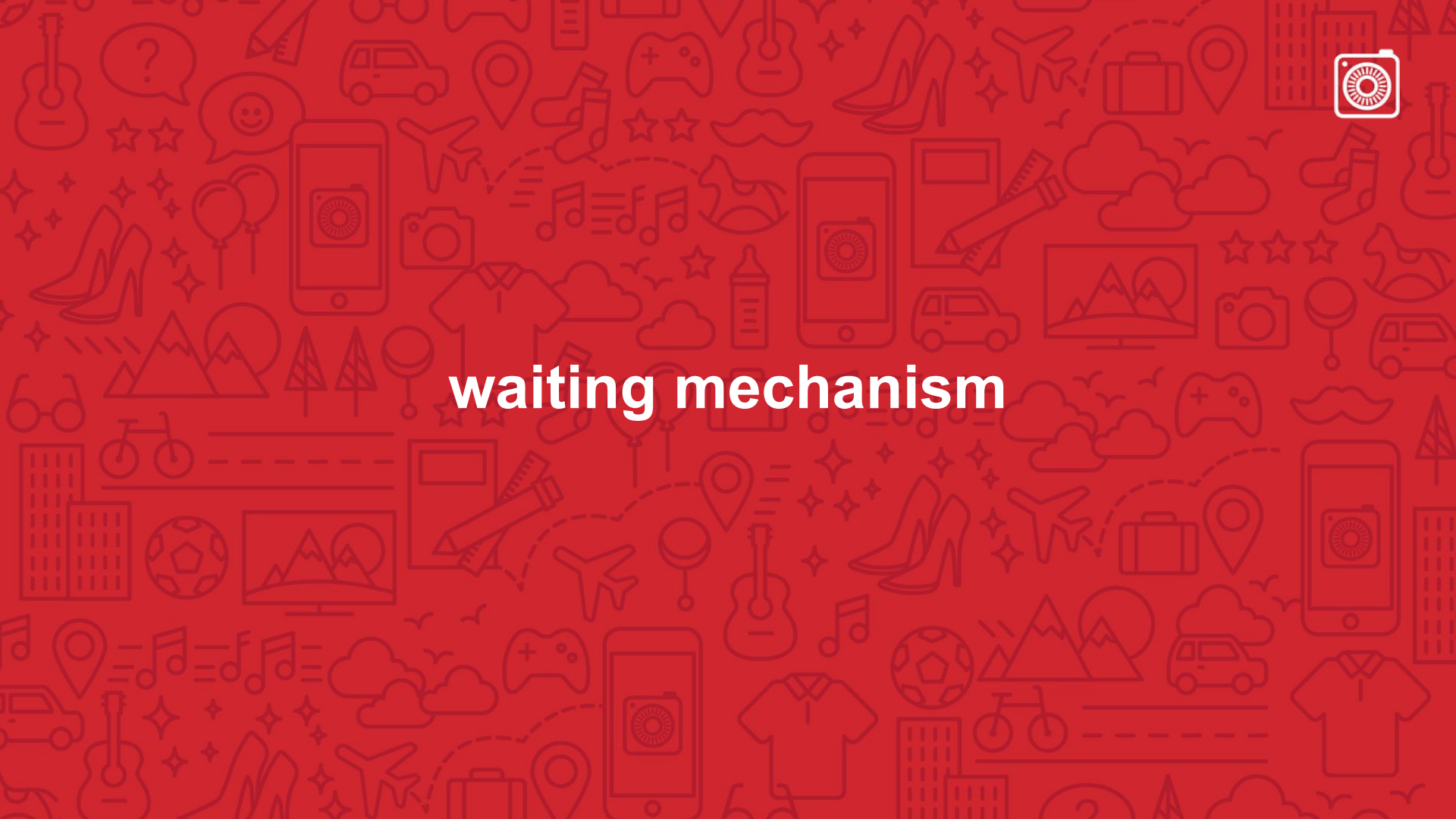
Short tests

Problem

- The higher the number of steps in a scenario, the higher the chances of flakiness.

Solution

- Keep tests short
- Split all existing large test scenarios to independent smaller ones
- Each scenario should not verify more than one expectation
- Not more than 7-8 steps

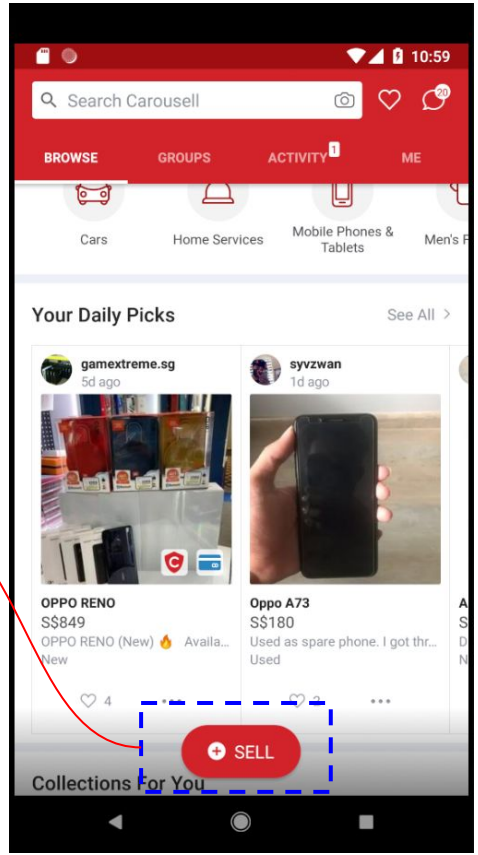

waiting mechanism

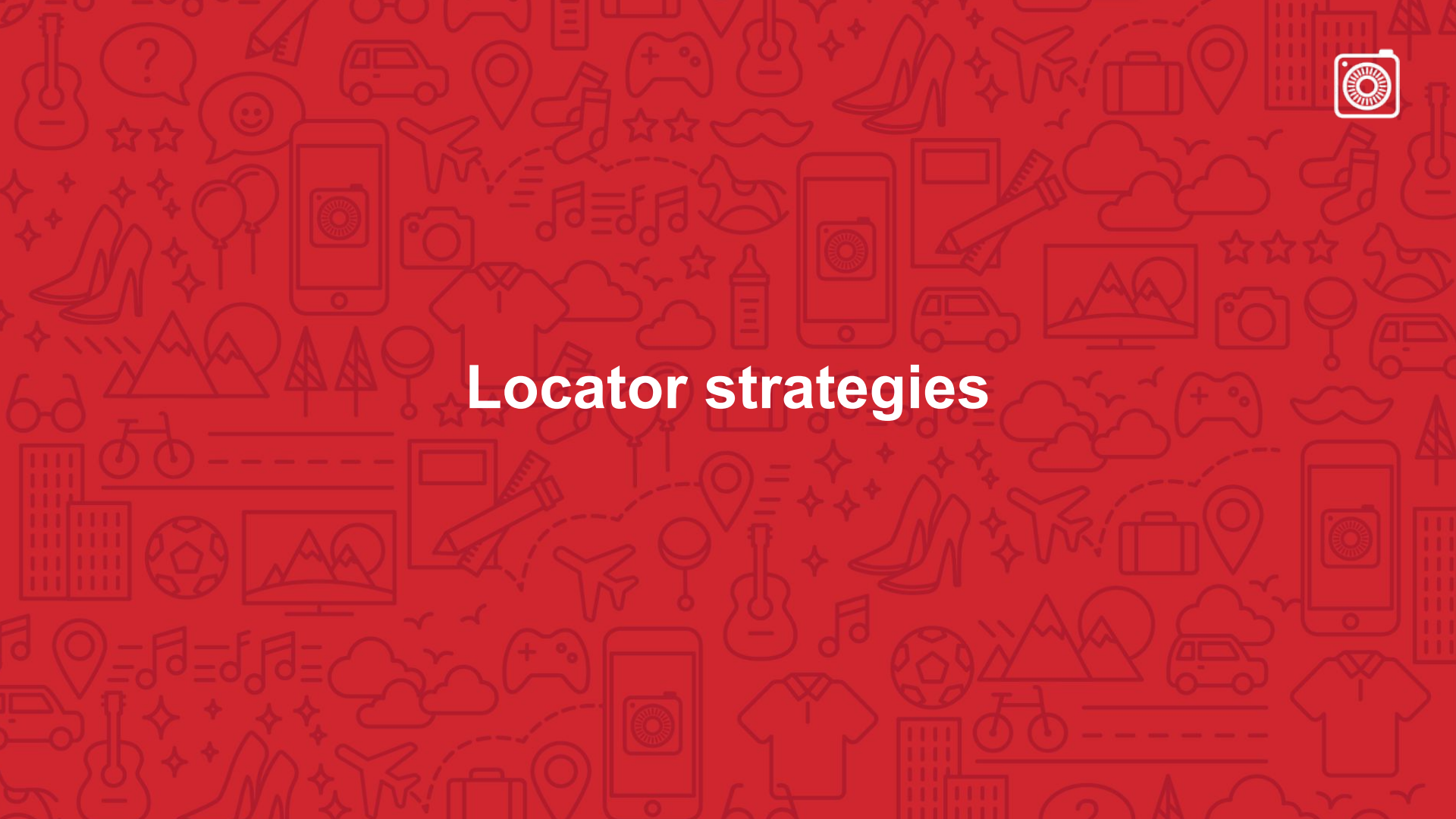
Conditional waiting mechanism for **stable** tests

Conditional waiting for checking whether an element is present/visible on the screen or not.

No `thread.sleep()!`

```
try {
      wait = new WebDriverWait(getDriver(), timeOut);
      element =
wait.until(ExpectedConditions.visibilityOf(element));
      LOG.info("Leaving waitForElement(WebElement
element={},int timeOut={})", element, timeOut);
      return element;
    }
catch (TimeoutException | ElementNotVisibleException |
NoSuchElementException e)
{
      throw new CarouLionException("Not able to find the
element " + element, e);
}
```




Locator strategies

Locator strategies for stable tests

More Stable

- Accessibility IDs
- IDs

Less Stable

- Name
- Class name
- UiSelector
- NSPredicate
- Class chain


Retries

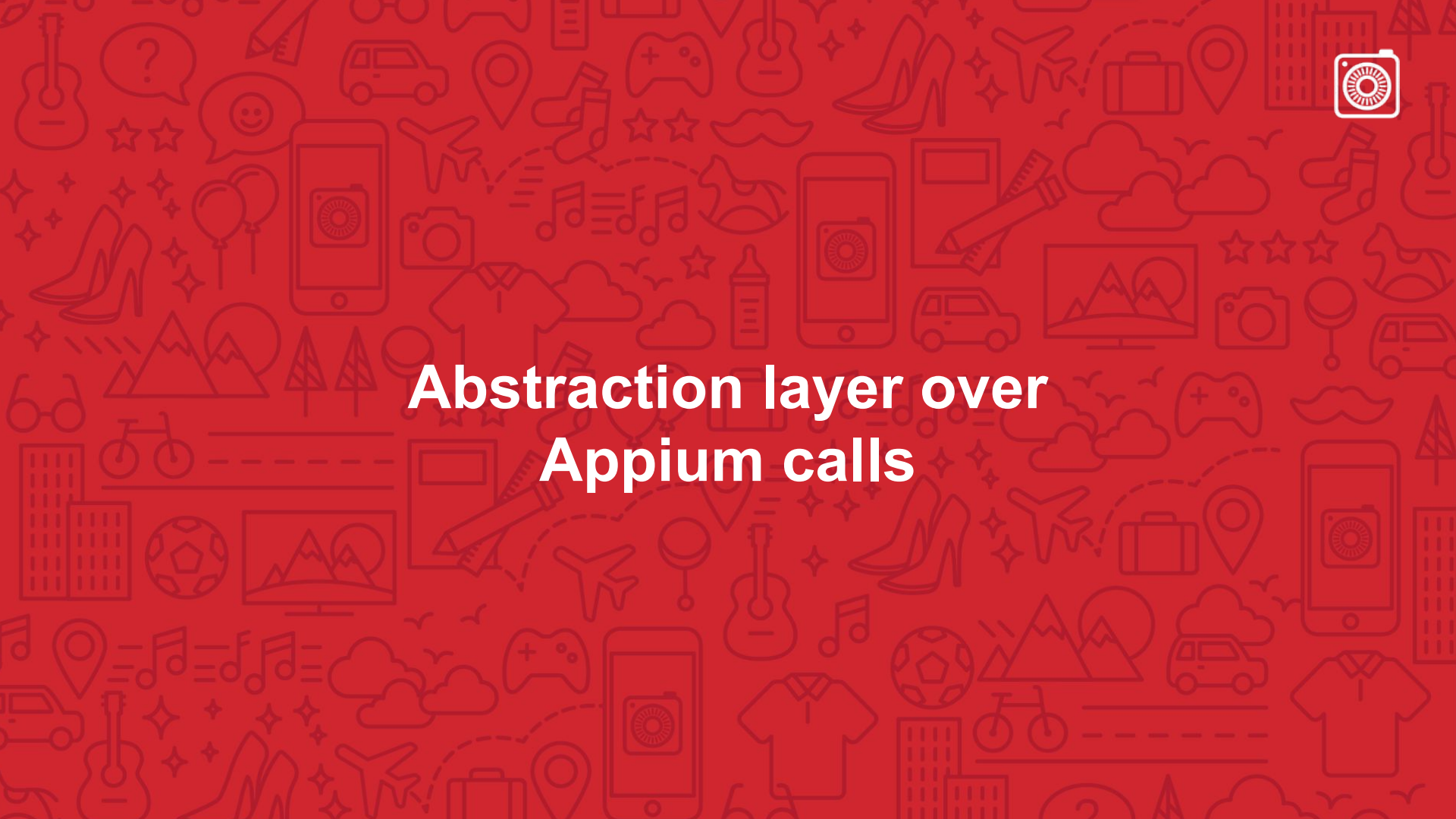
Retrying for flaky tests

Using retry as a mechanism to re-run tests that fail due to below conditions:

- Internet connectivity issues on test execution host
- Internet connectivity issues on host that runs the device
- Device issues
- Session issues in WebDriver

Where do we apply retries:

- Retry a failed test (not more than twice)
- Retry an API call


Abstraction layer over
Appium calls

Abstraction layer over Appium calls

Motivation

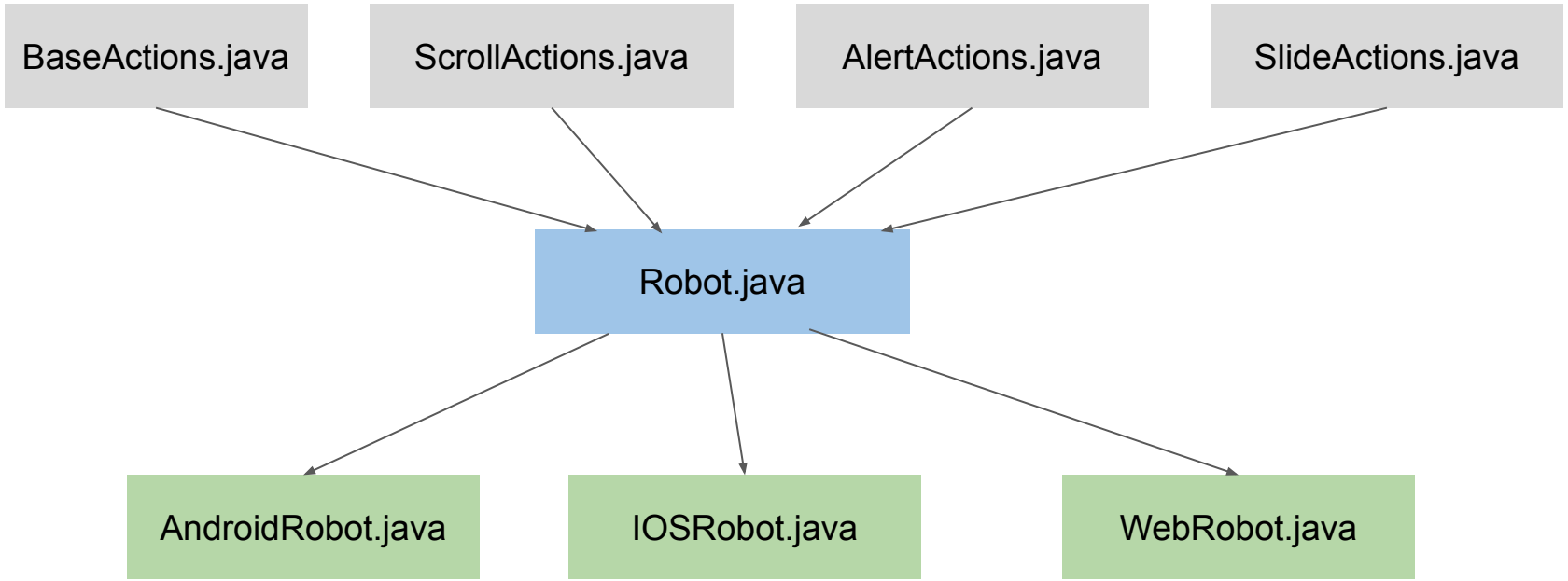
- Each engineer created their own methods for webdriver interactions, waiting mechanism, scrolling and other actions on the app either in Page Object methods or BasePage.

- This led to code bloat and inconsistency → flakiness in the tests

What we did

- Created an abstraction layer on top of all frequently used actions on the app:
  - Waiting for an element given a web element
  - Waiting for an element given a `By` object
  - Click a webelement
  - Click a webelement with a timeout
  - Pull to refresh
  - Scroll up
  - Scroll Down etc.

Implementation



```
robot.click(closeButton);
robot.waitForElement(successfulBumpMessage);
robot.isDisplayed(reservedBumpMessage, 10);
robot.scrollDown(2);
robot.scrollUp(2);
robot.scrollDown(threeDayBumpButton, 3);
```

People, Process and Communication

Involving all the people,
building structured processes,
and conducting efficient communication
is also a key towards stable test automation

People

- Everyone in the test engineering team helps to fix the tests
- Developers are also kept in the loop about failing tests

Processes

- Framework level checks to eliminate surprise failures during test execution
- Checklist for reviewing test automation PRs
- SLAs for fixing FFT, sanity and regression tests
- Isolation of flaky tests - Disable, fix them and enable them again
- Every week 2 engineers look into failing tests and act upon them
- Guidelines written for developers to add accessibility ids

Communication

- Slack notifications for test runs
- Testrail test runs
- Looker dashboards

Win amazing prices!