自動化に取り組むにあたり

JaSST'22 Tohoku

松尾和昭

私について

- 名前: 松尾和昭 (Twitter: @Kazu_cocoa, GitHub: KazuCocoa)
- 所属と職種: HeadSpin, Inc, Software Engineer
  - Ex. Cookpad (QA/Test Software Engineer), ACCESS(Software Engineer)
- OSS: Appium Project 
- 最近の活動: 「実践ソフトウェアエンジニアリング第9版」の一部翻訳

お題

- テスト自動化(test automation)
  - 範囲
  - 実行速度
- これから自動化を学んでいくために

話さないこと

- テスト技法、〇〇性といった特性の話
- 費用対効果、コスト
- プロセス
- など

テスト自動化

Test Automation

テスト自動化 - Test Automation

- The use of software to perform or support test activities
  - ソフトウェアを使ったテスト活動の実行や支援

https://glossary.istqb.org/jp/search

テスト自動化 - Test Automation

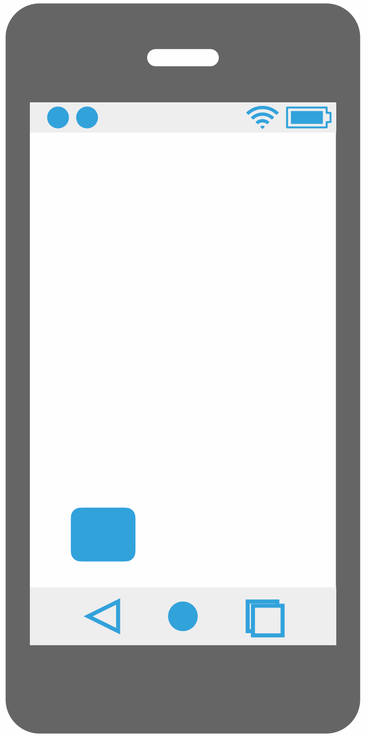
https://glossary.istqb.org/jp/search/automation

テスト自動化 - Test Automation

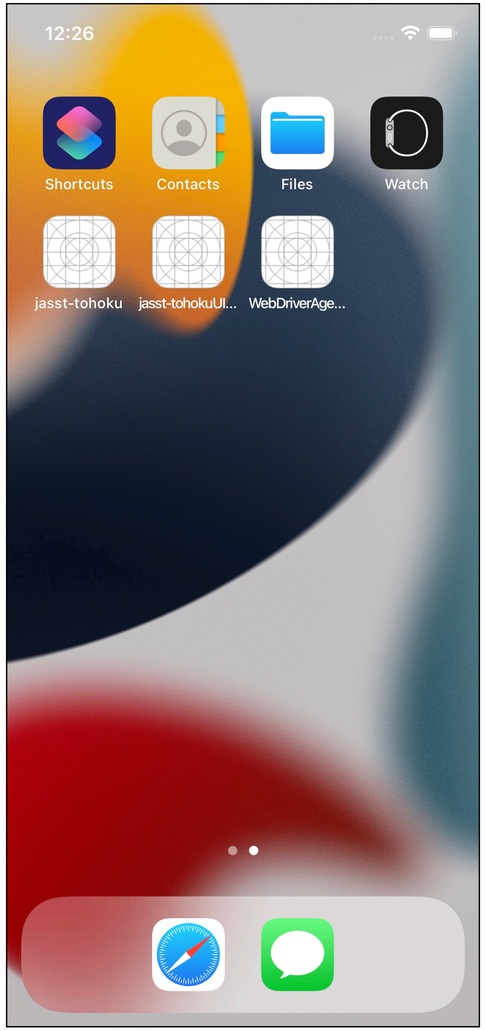
https://glossary.istqb.org/jp/search/automation

例えばテスト実行の自動化

- GUIを操作して行うモバイルアプリのテスト
  - テスト対象となるアプリを起動
  - 何らかの操作をGUI越しに操作し、結果を記録
  - 結果を検証



例

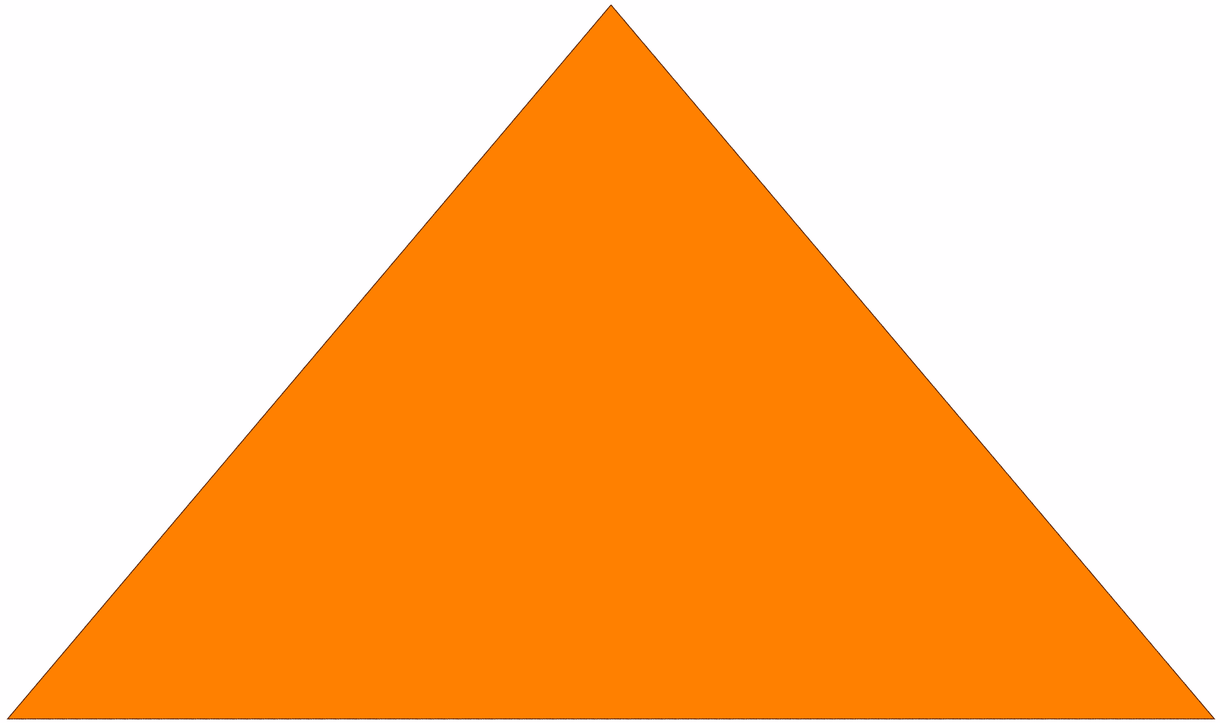

12:26
Shortcuts
Contacts
Files
Watch
jasst-tohoku
jasst-tohokuUI...
WebDriverAge...

テスト自動化のモデル

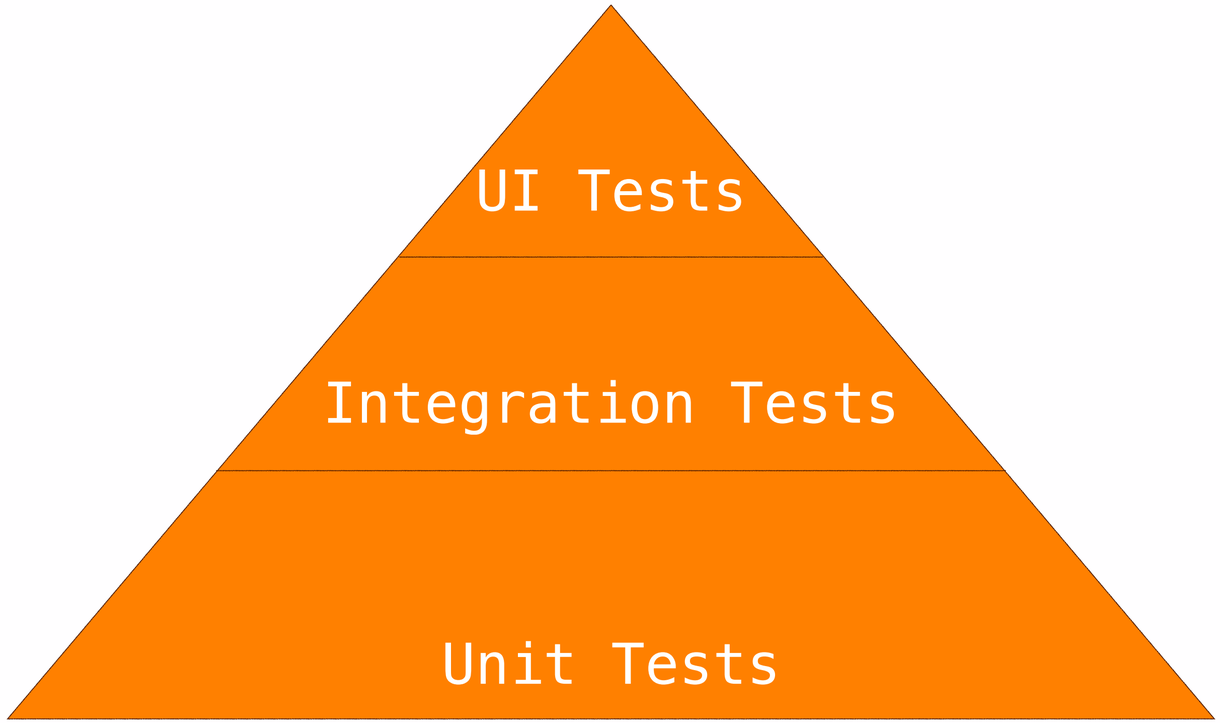
テストピラミッド

- テスト自動化を議論するとき、よく出てくる図の1つ
  - Web、モバイル
- 少し前まではAndroidの公式ドキュメントにも説明が載っていた
  - https://developer.android.com/training/testing/fundamentals?hl=ja
    - (日本語のものにはまだ残っている)

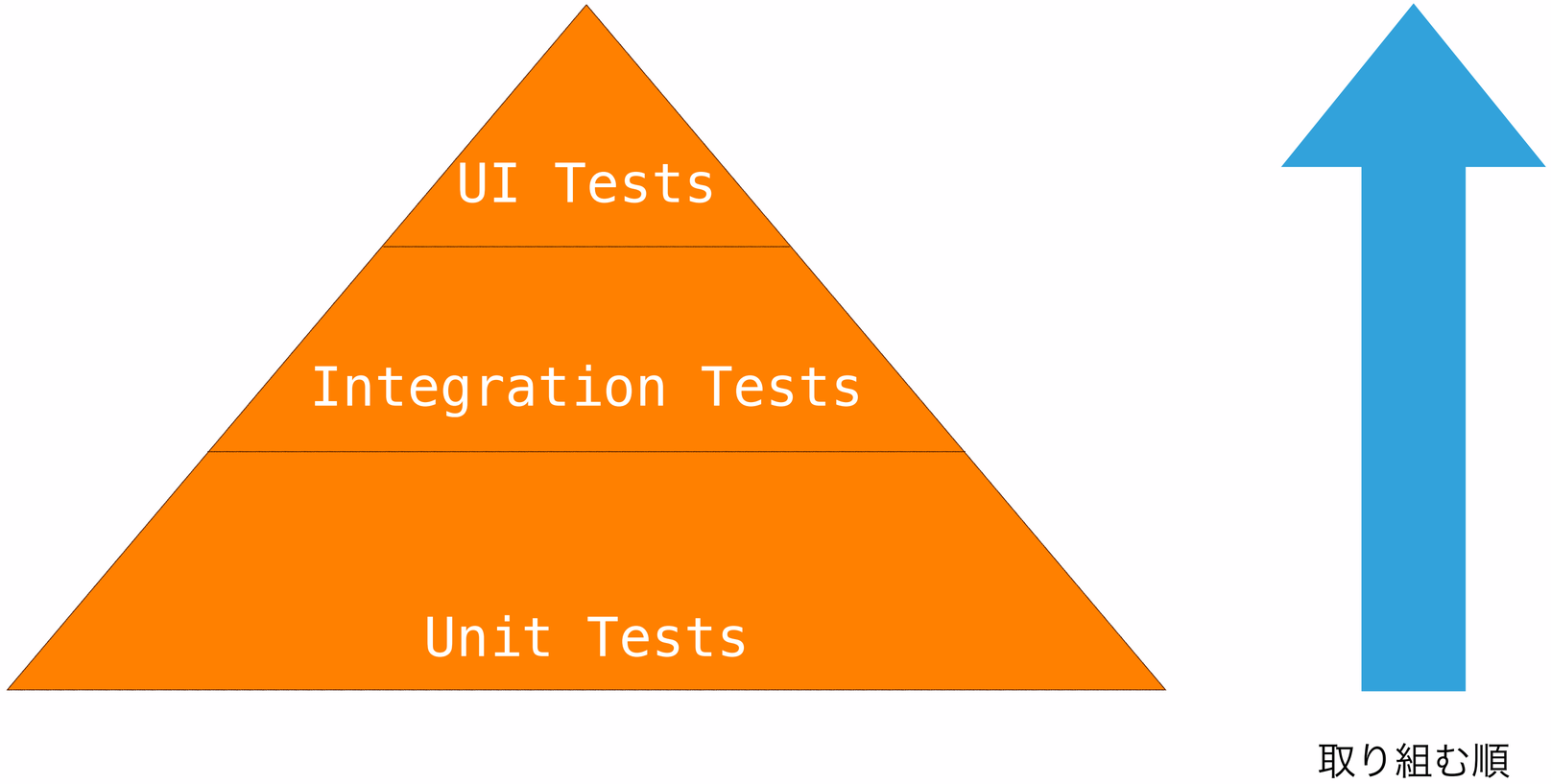
ピラミッド



テストピラミッド



テスト自動化を進めるときは底から



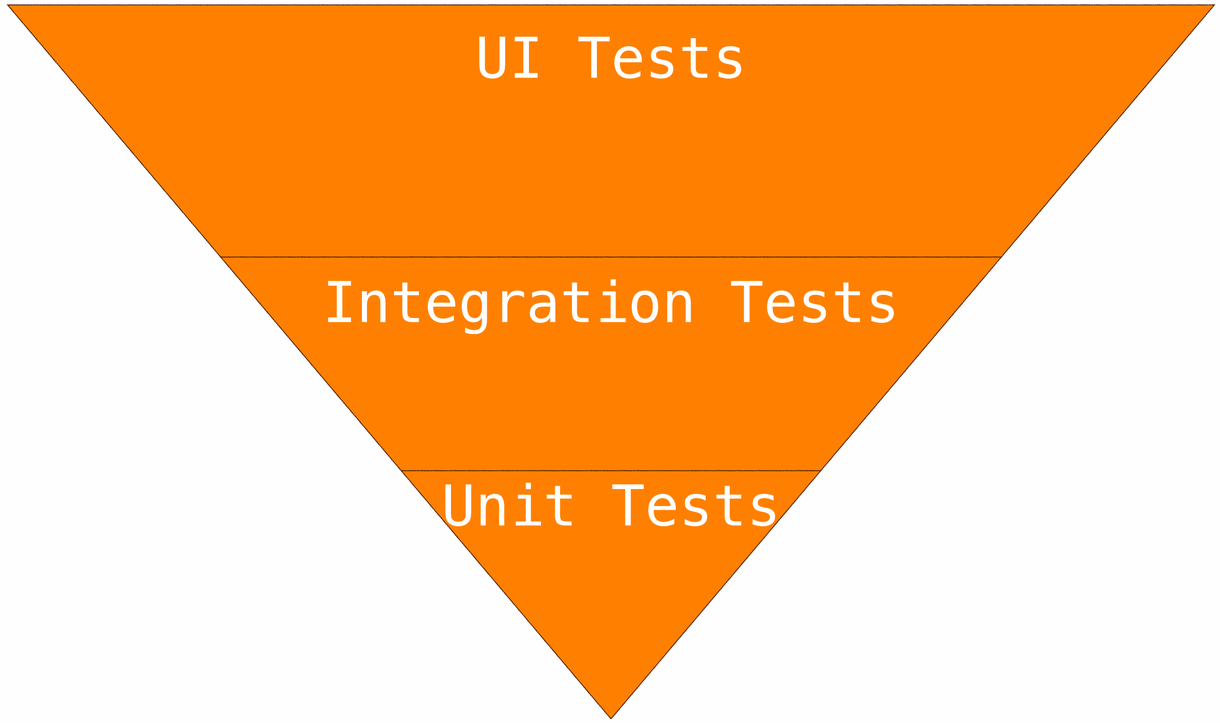
様々な工夫

アプリのアーキテクチャの
導入・変更

テスト実行の
並列化による時間短縮

閉じた環境を
構築できるようにする

テスト逆ピラミッド



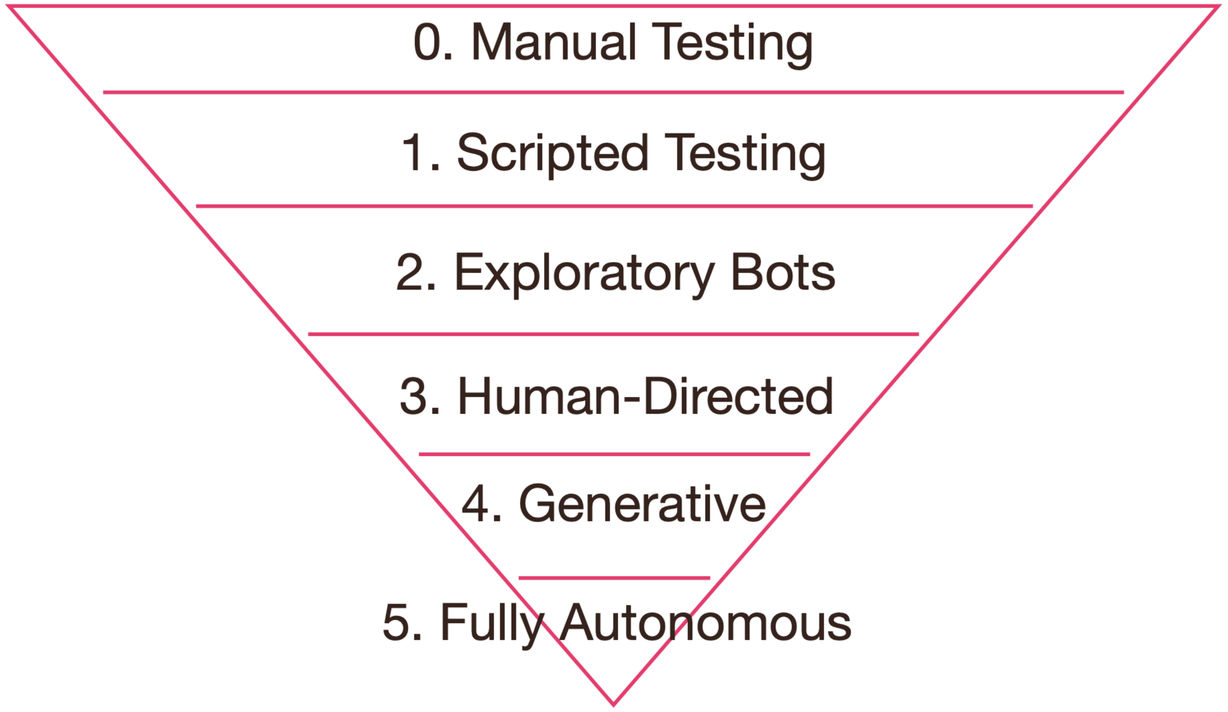
実行速度、範囲/サイズ

Test Size

feature	*Small*	*Medium*	*Large*	*Enormous*
network	no	localhost	localhost/yes	yes
system access	no	partial/yes	yes	yes
OS(APIs)	no	yes	yes	yes
View	no	partial	yes	yes
external system	no	no	no	yes
time per a test	< 100ms	< 2s	< 120s	< 500s

https://www.slideshare.net/KazuMatsu/cookpadtechcond2018mobiletestautomation

蛇足: “自動化”とは、どこまで?



https://medium.co

1つの例として、テス ツール、スクリプトを実行するだけ、なんらか なんらかの操作を行う類)、人が操作するように操作を行う、探索的にテストカバレッジの多くをそうは出来るようにテストを行う、人が行うレベルのテストを人の補助なしで自動的に行う。ここで言う人が行うレベルというのは、何をいつ、どうテストするかといったことも含む。

実際には昨今の多くは2、もしくは3までかと思います。ただ、自動化という意味ではこういう温度感まで議論されています。

これも蛇足:ところで、Googleのページ

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

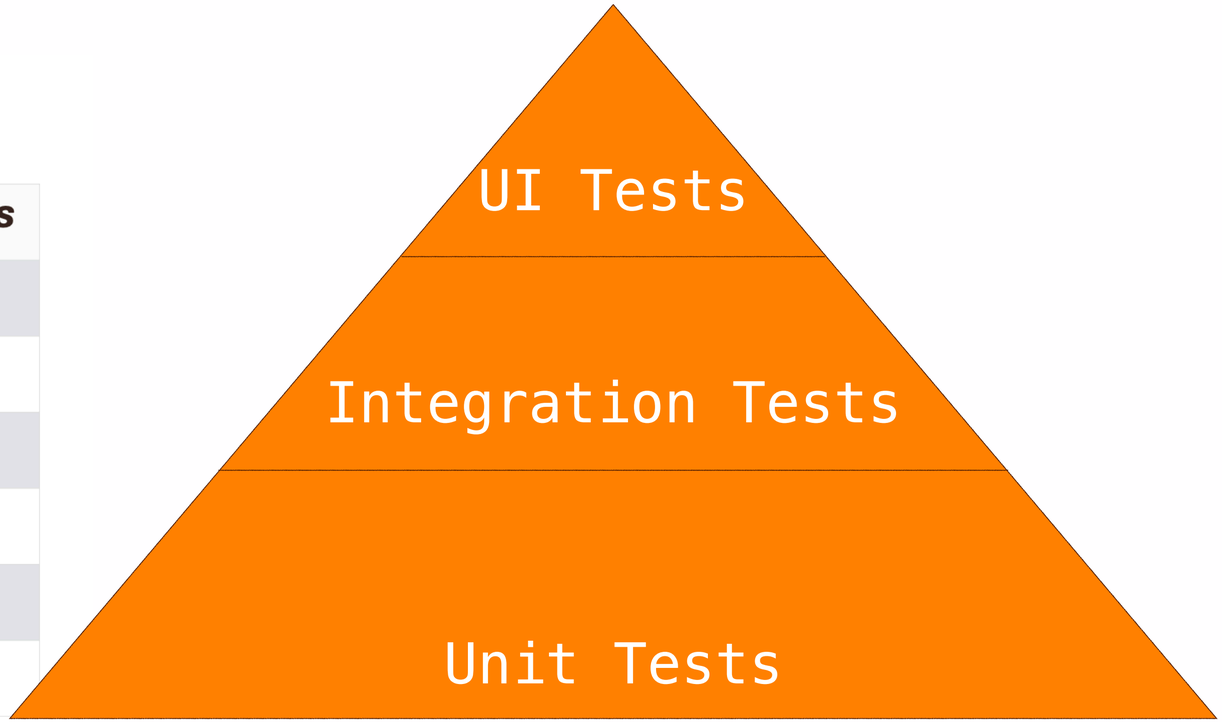
余力があれば?入れる。

余力があれば入れる

並べる

Test Size

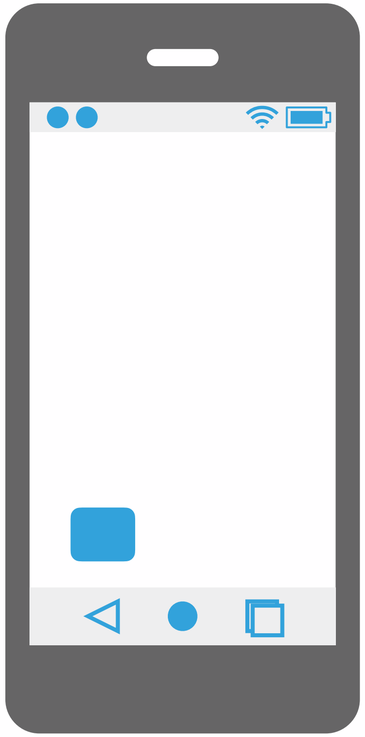
feature	*Small*	*Medium*	*Large*	*Enormous*
network	no	localhost	localhost/yes	yes
system access	no	partial/yes	yes	yes
OS(APIs)	no	yes	yes	yes
View	no	partial	yes	yes
external system	no	no	no	yes
time per a test	< 100ms	< 2s	< 120s	< 500s



範囲と実行速度

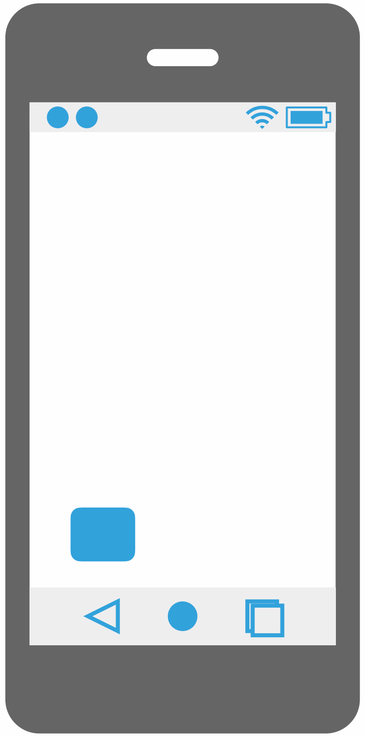
例

1. Webブラウザを開く
2. https://jasst.jp/symposium/jasst22tohoku.html を入力する
3. 最下部までスクロールする
4. 最上部まで戻る



例

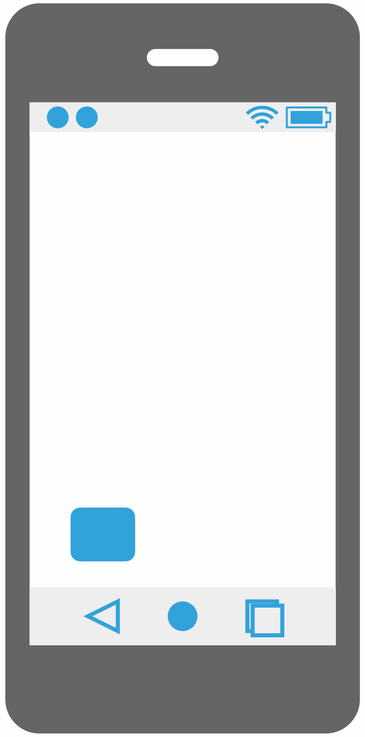
1. Webブラウザを開く
2. https://jasst.jp/symposium/jasst22tohoku.html を入力する
3. 最下部までスクロールする
4. 最上部まで戻る



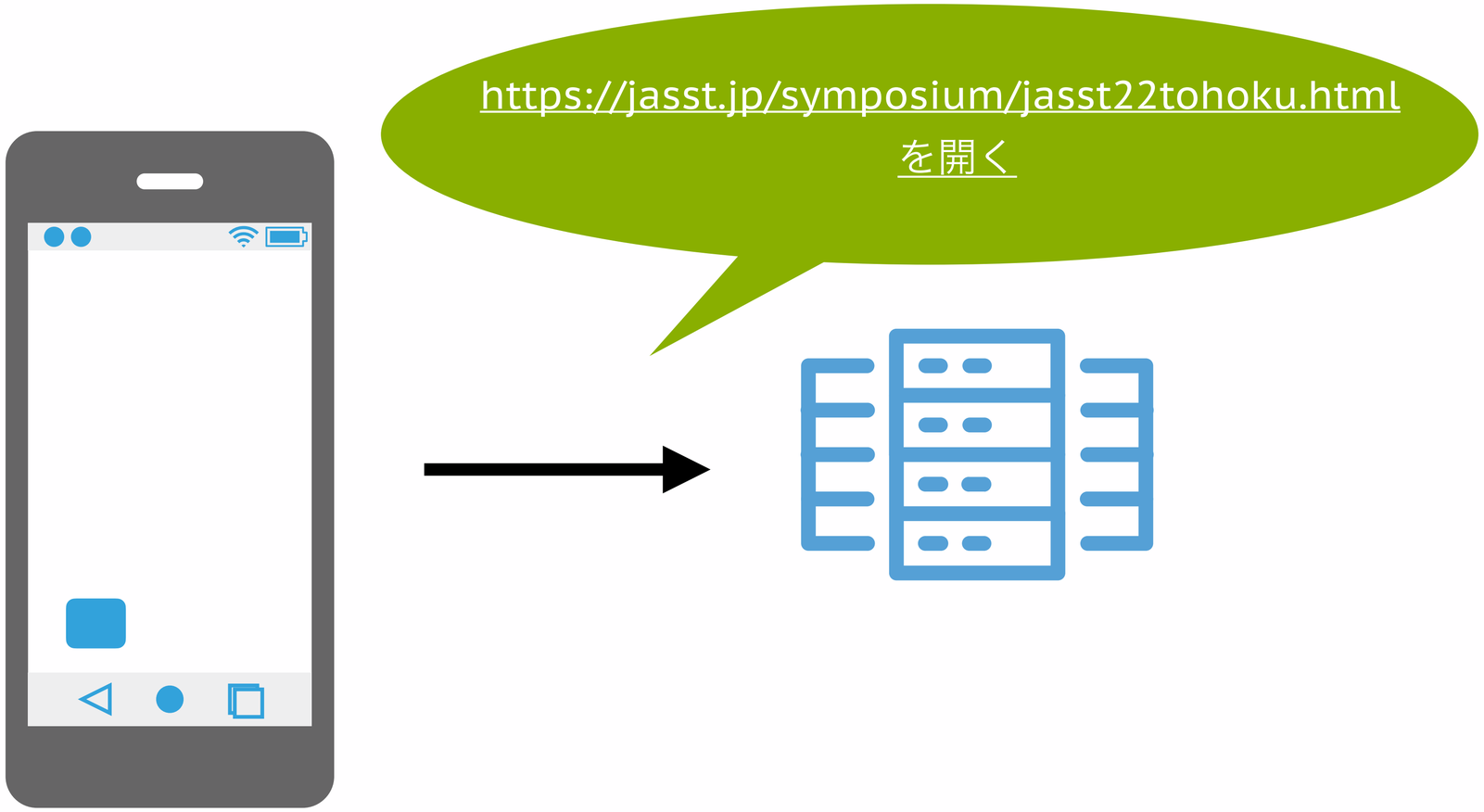
例えば

例

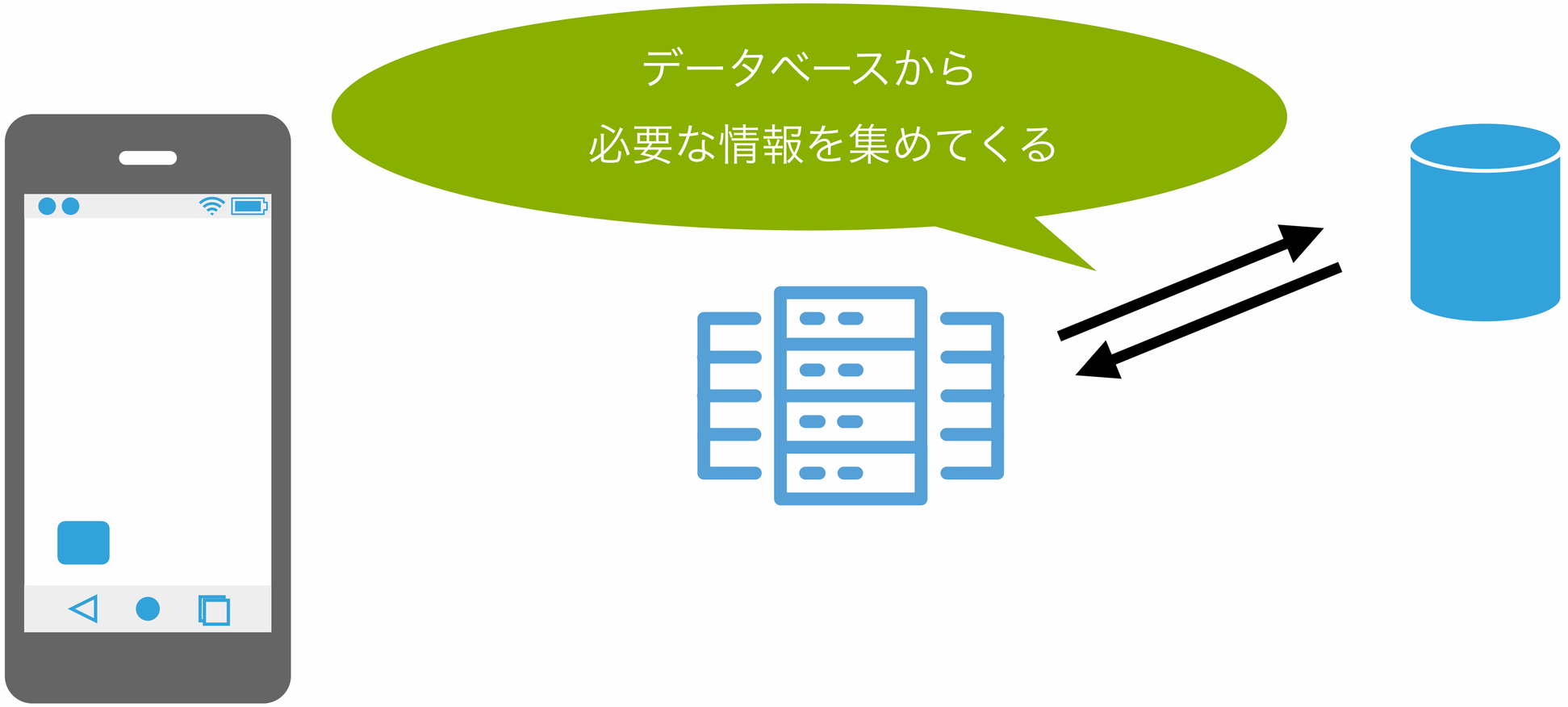
1. Webブラウザを開く
2. https://jasst.jp/symposium/jasst22tohoku.html を入力する
3. 最下部までスクロールする
4. 最上部まで戻る



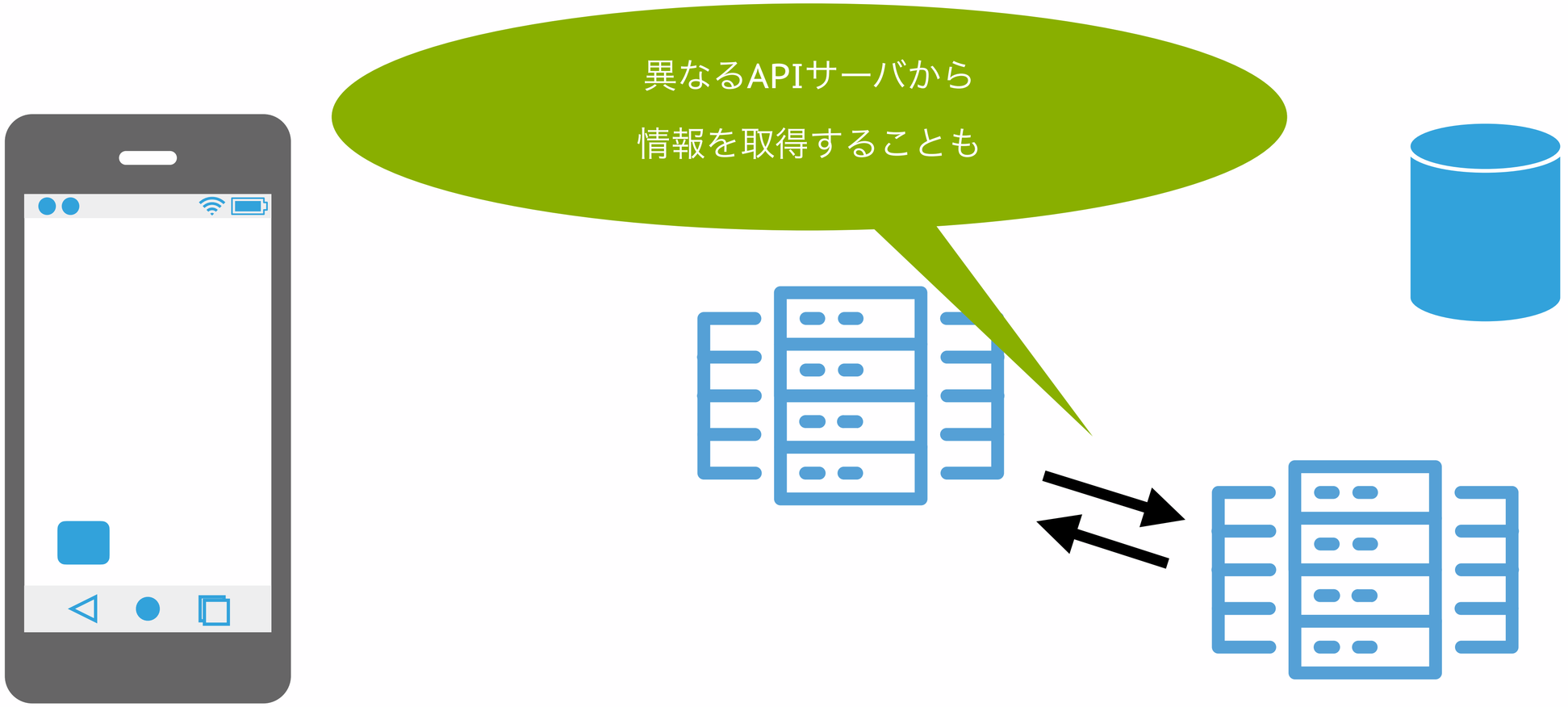
例えば



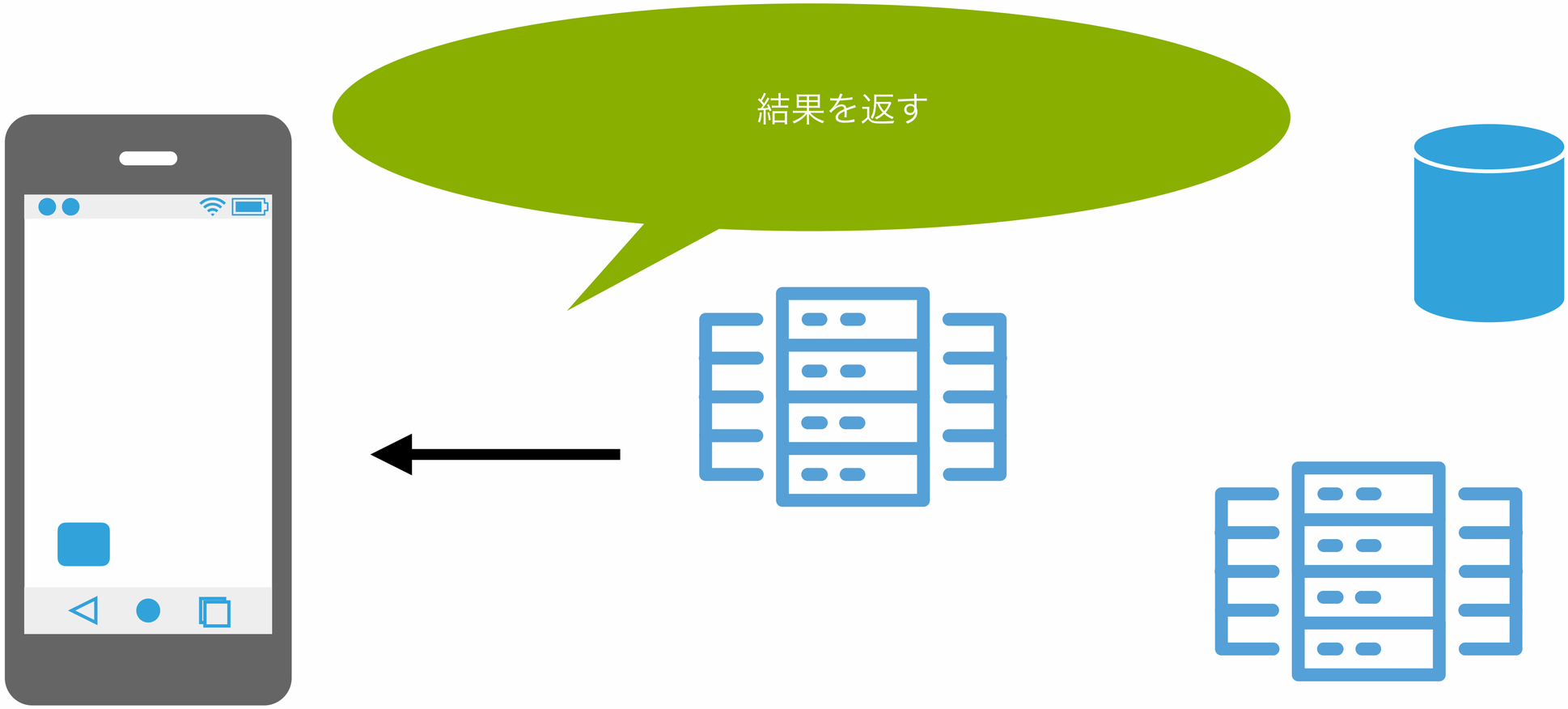
例えば



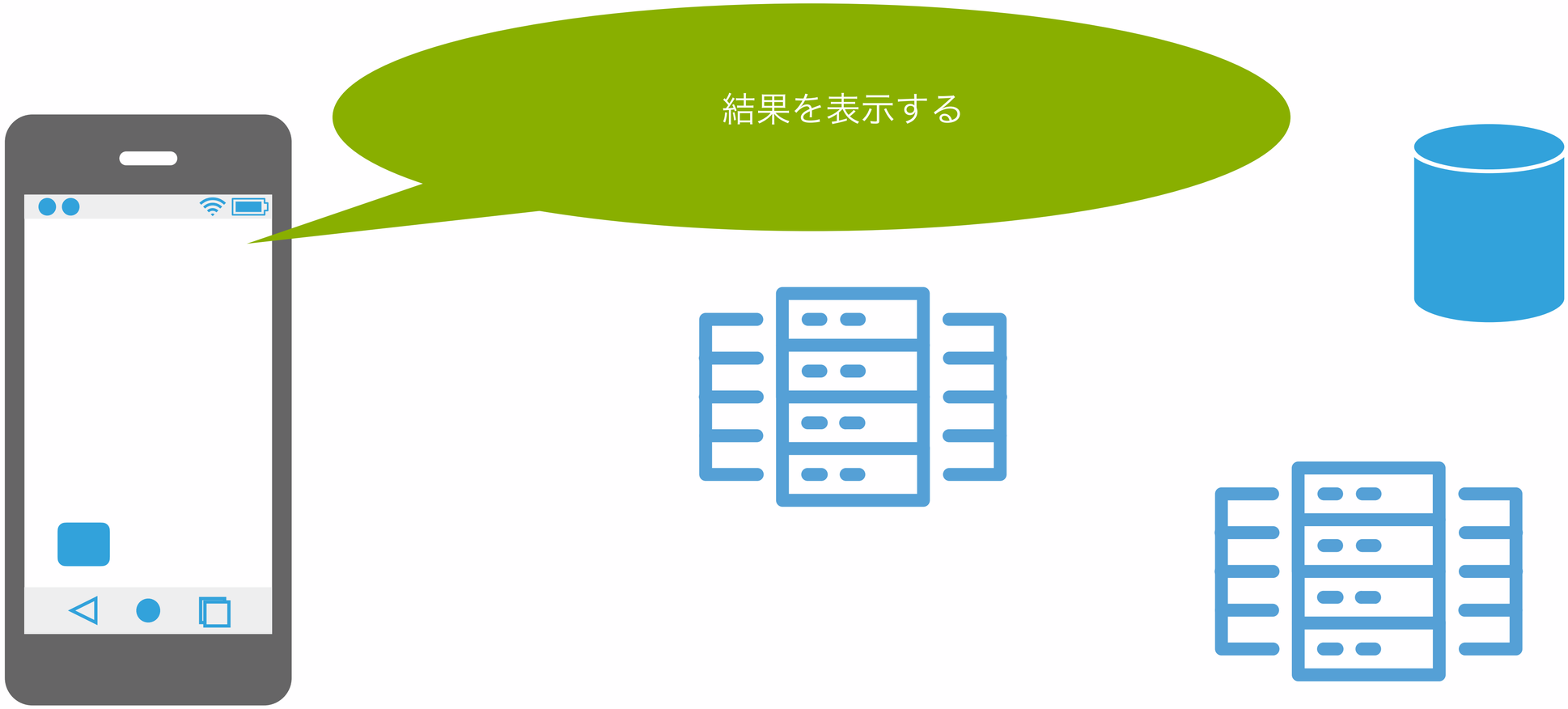
例えば



例えば

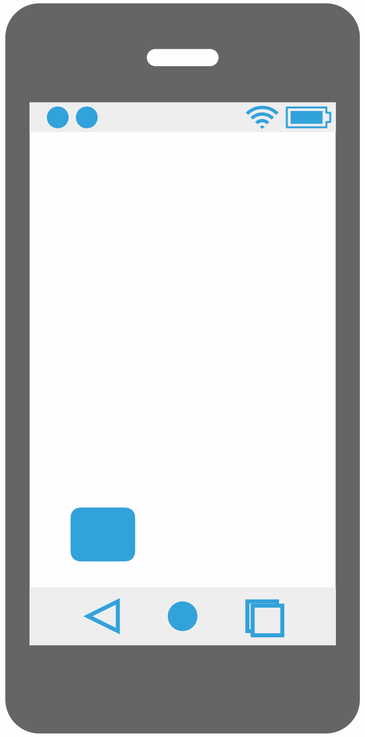


例えば

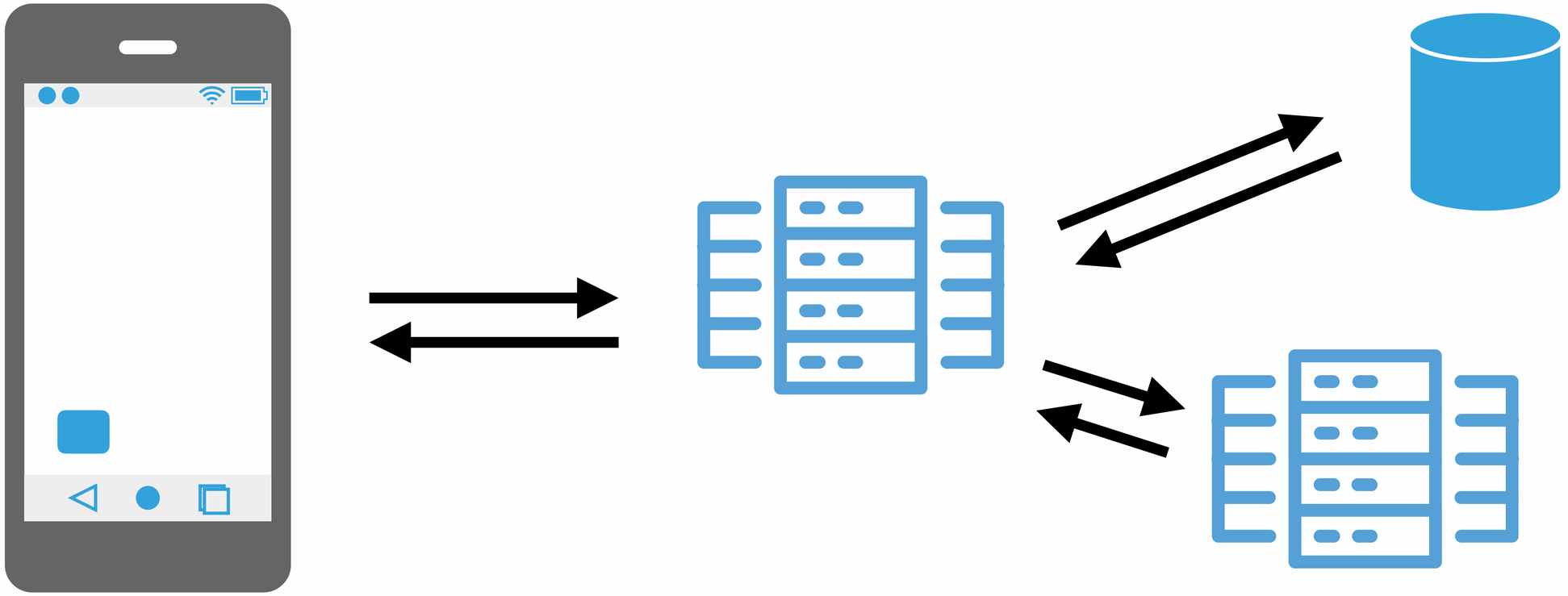


例

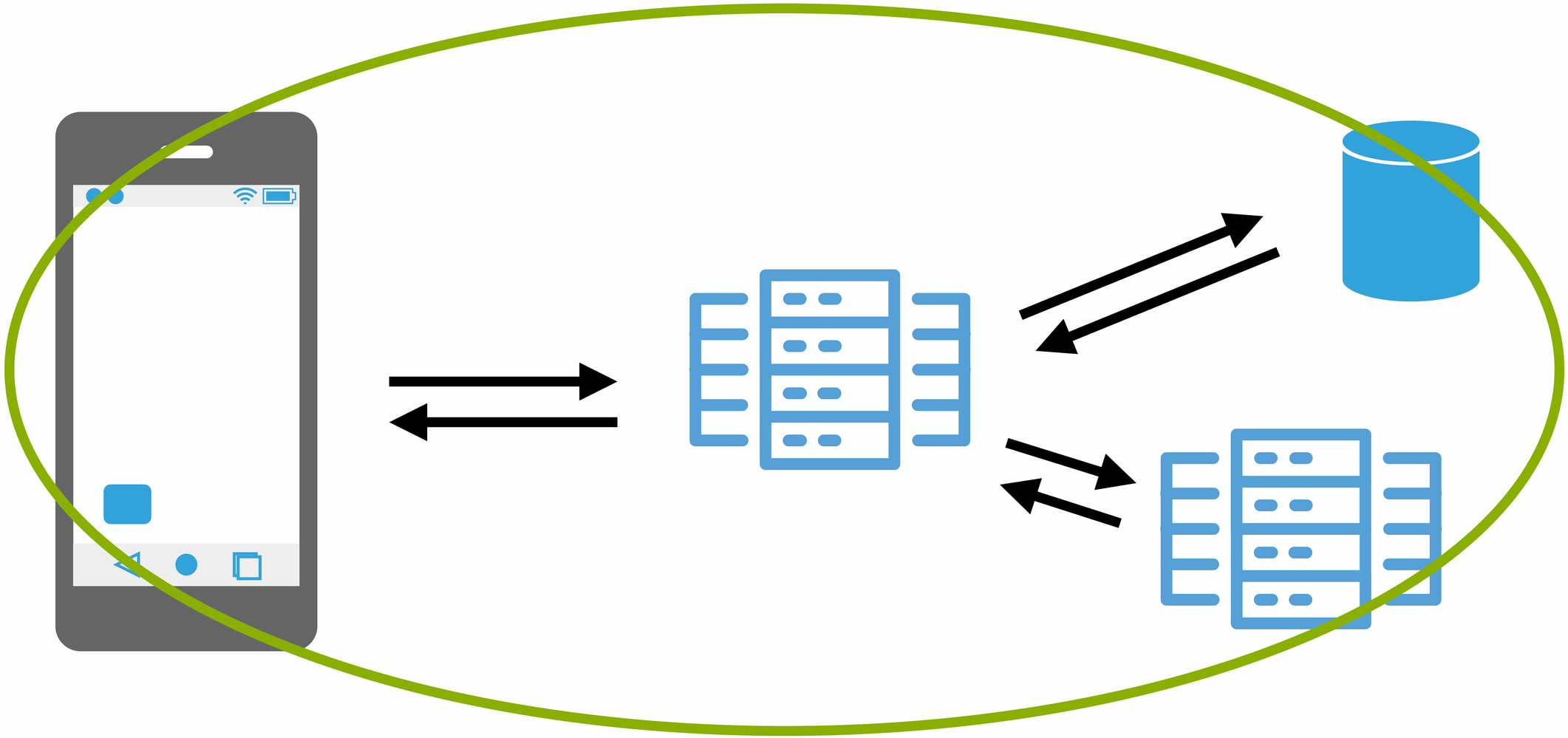
1. Webブラウザを開く
2. https://jasst.jp/symposium/jasst22tohoku.html を入力する
3. 最下部までスクロールする
4. 最上部まで戻る



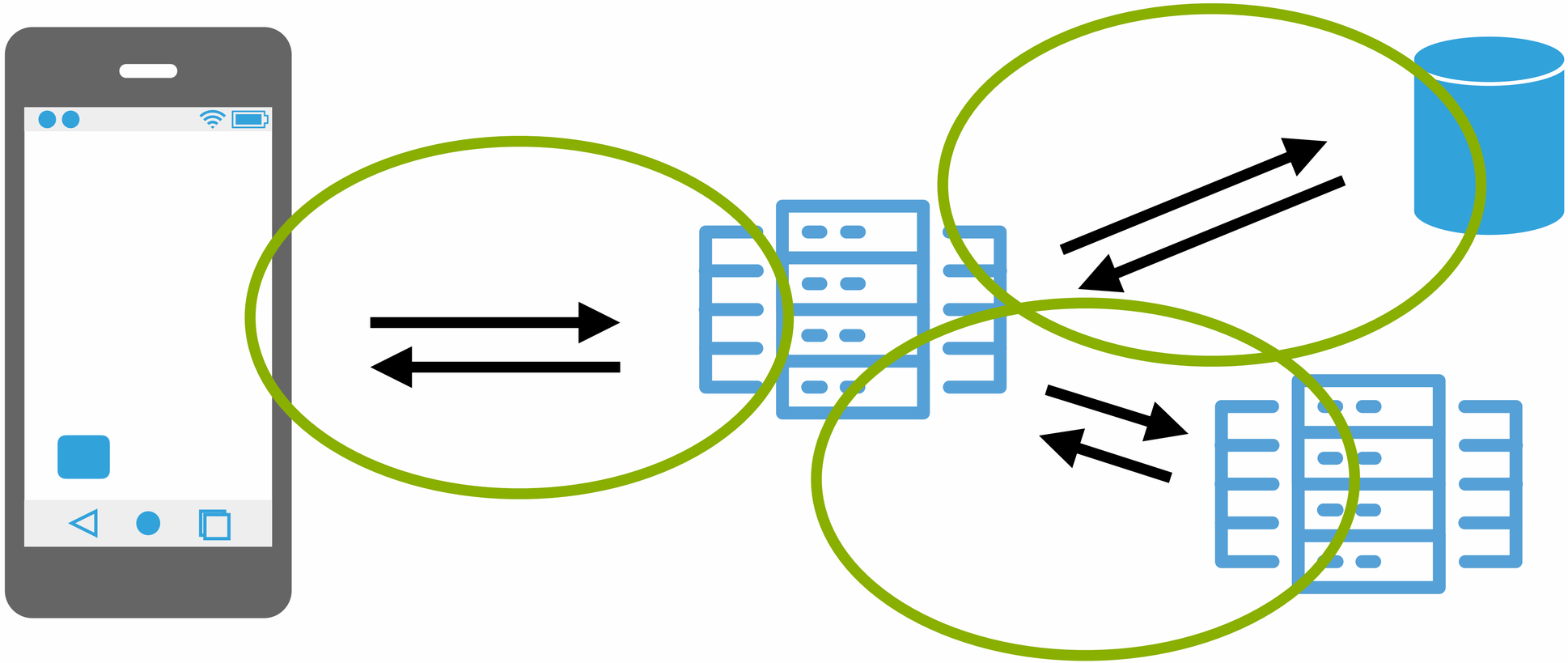
関係



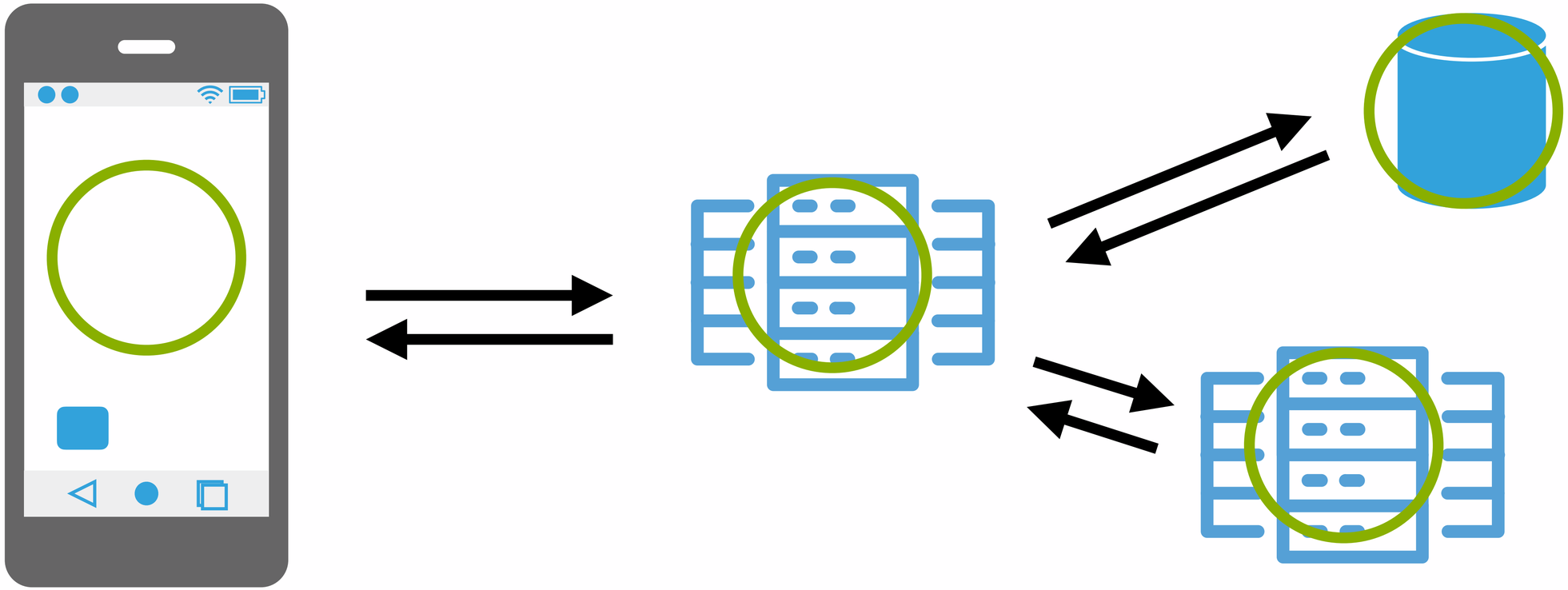
系全体



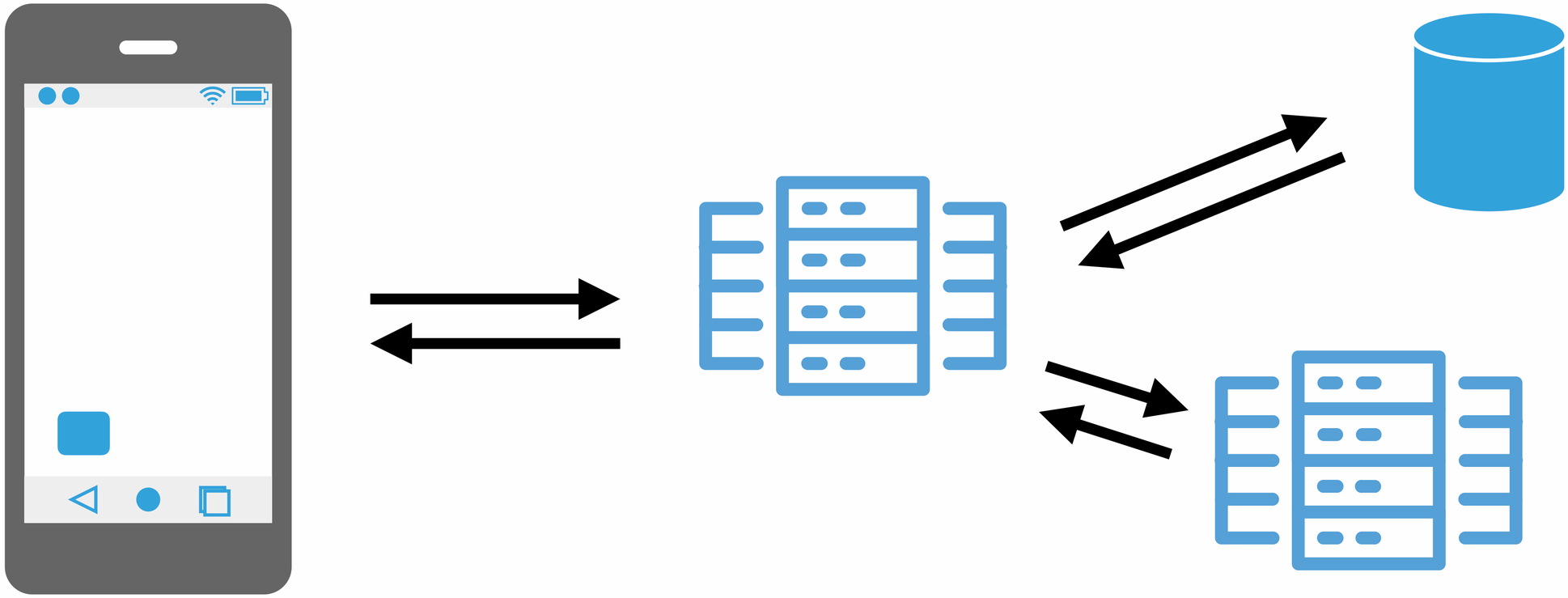
通信の境界で区切る



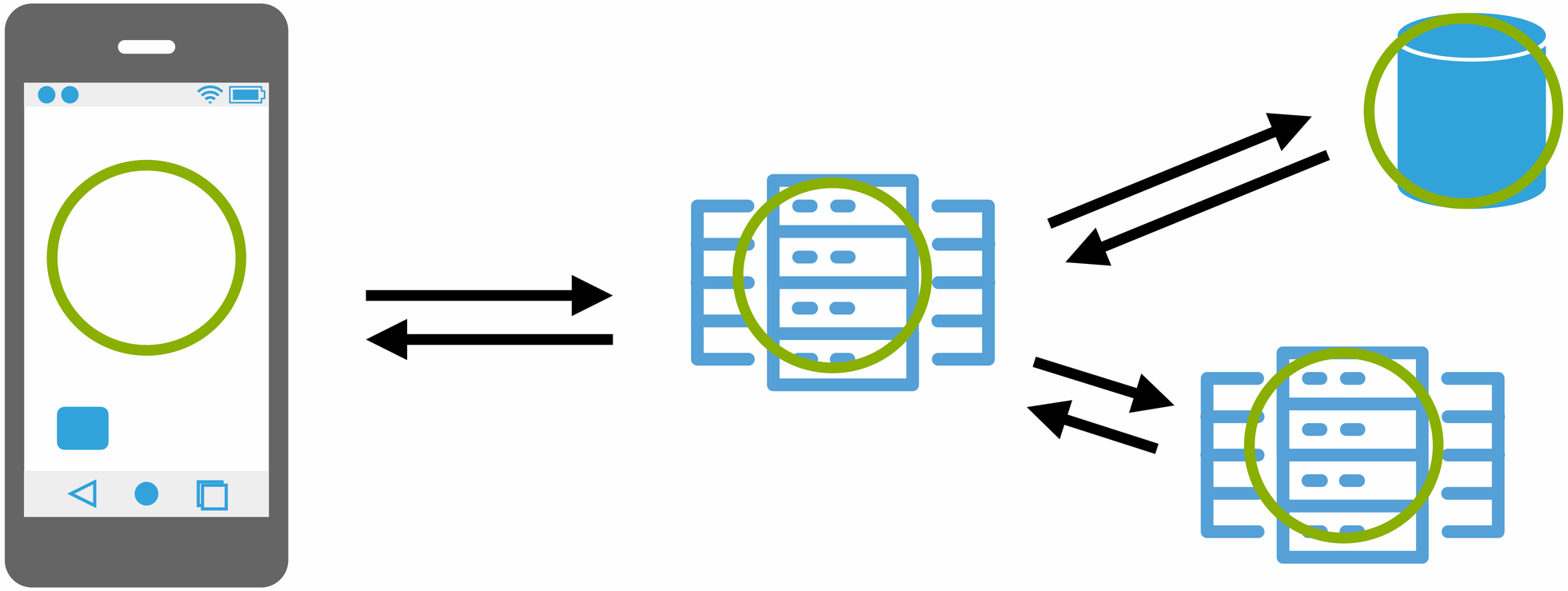
個々のより細かな実装



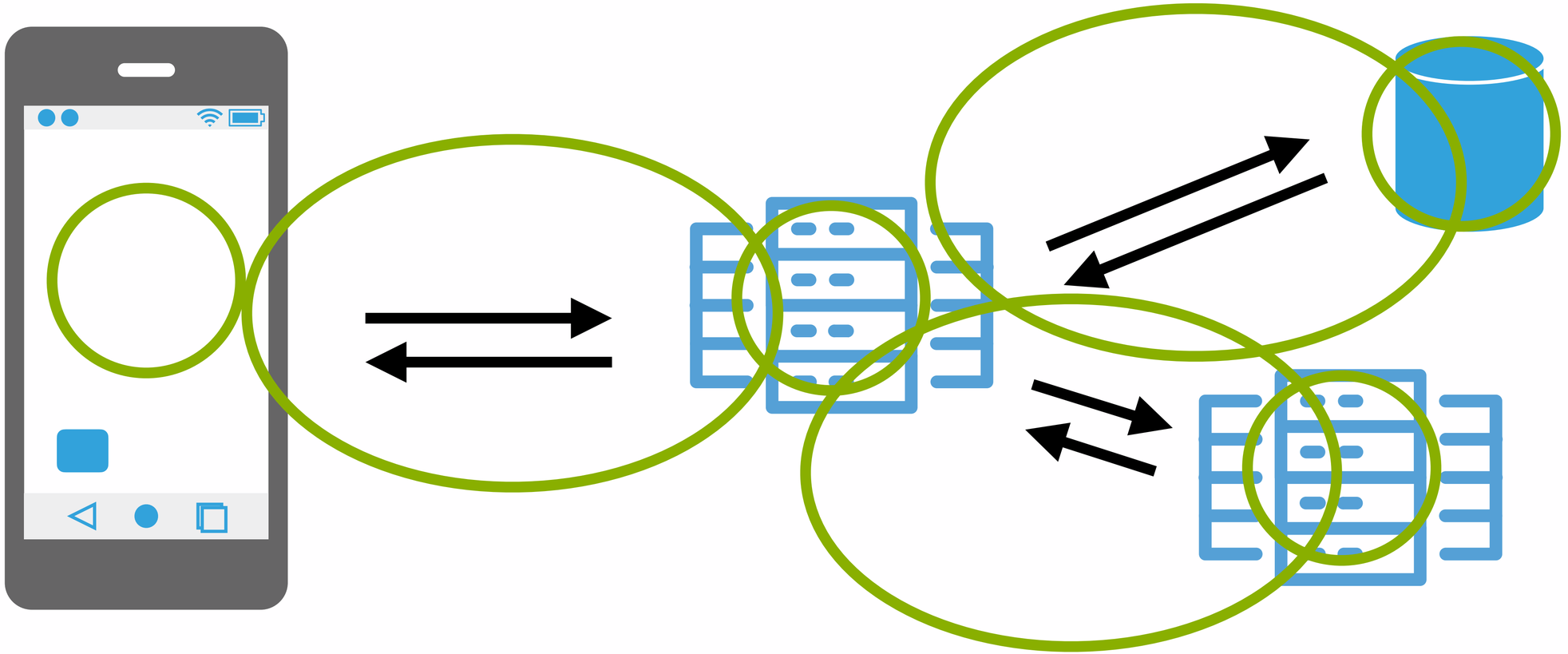
1つの流れでも、その裏には多くの要素が関係している



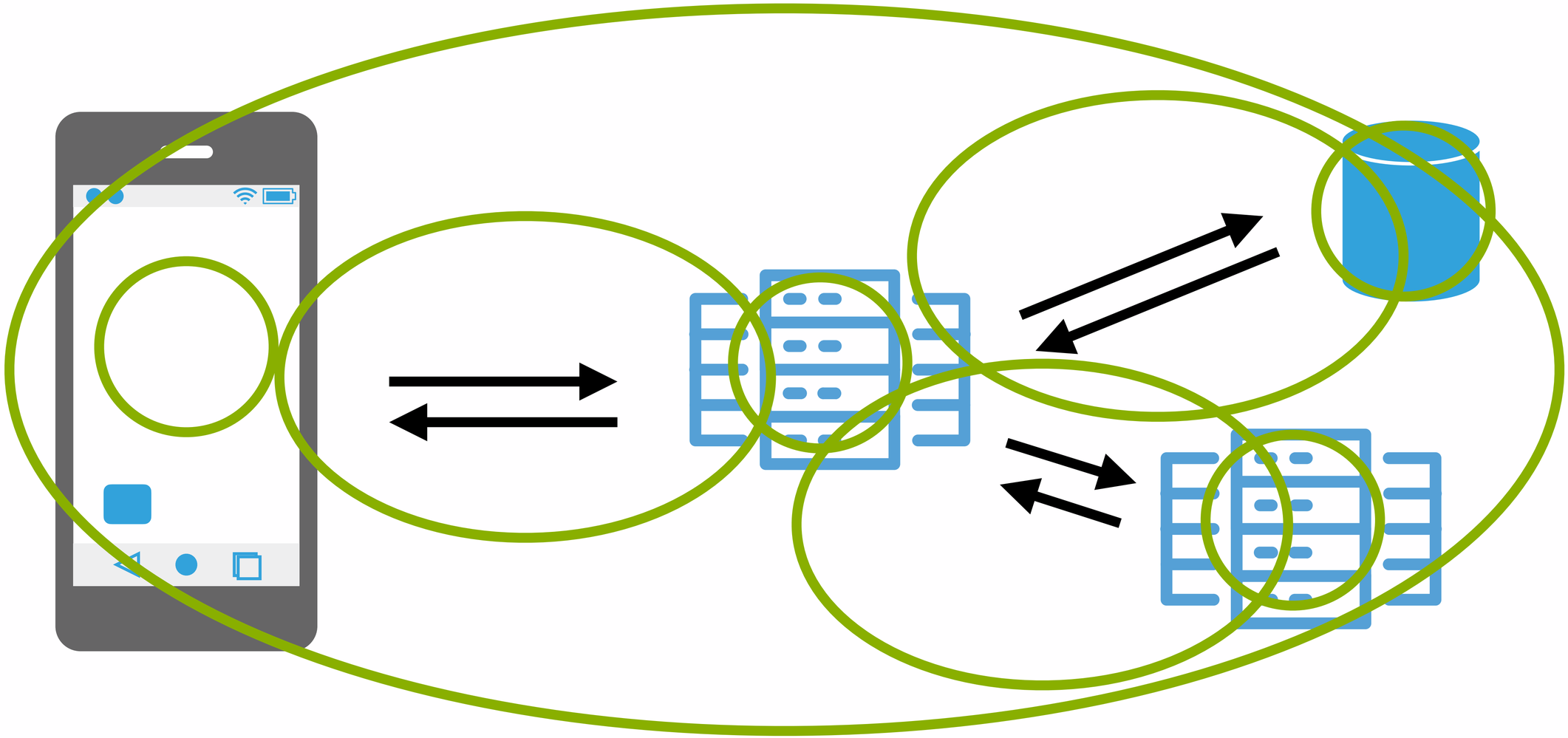
それぞれから



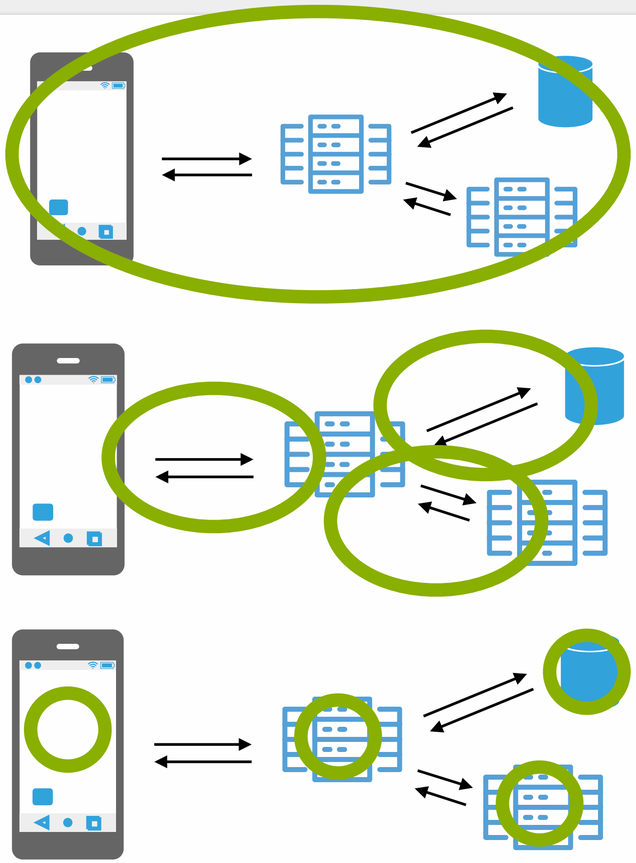
通信も

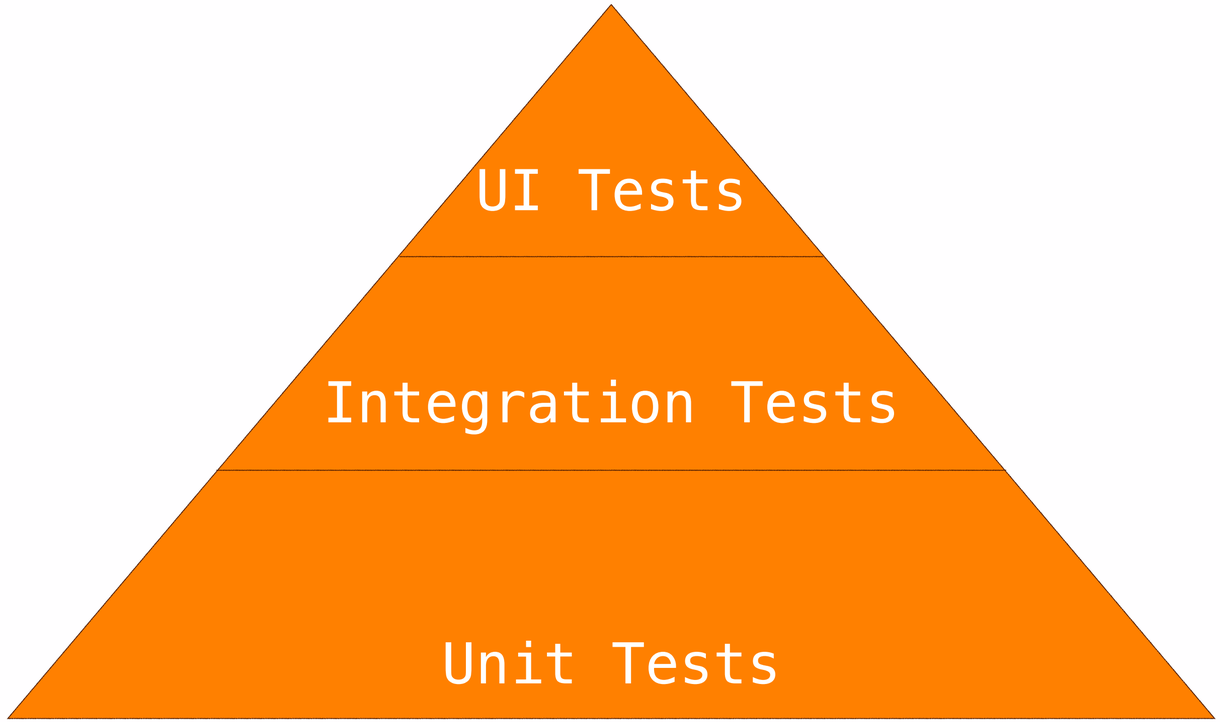


全体へ



並べる

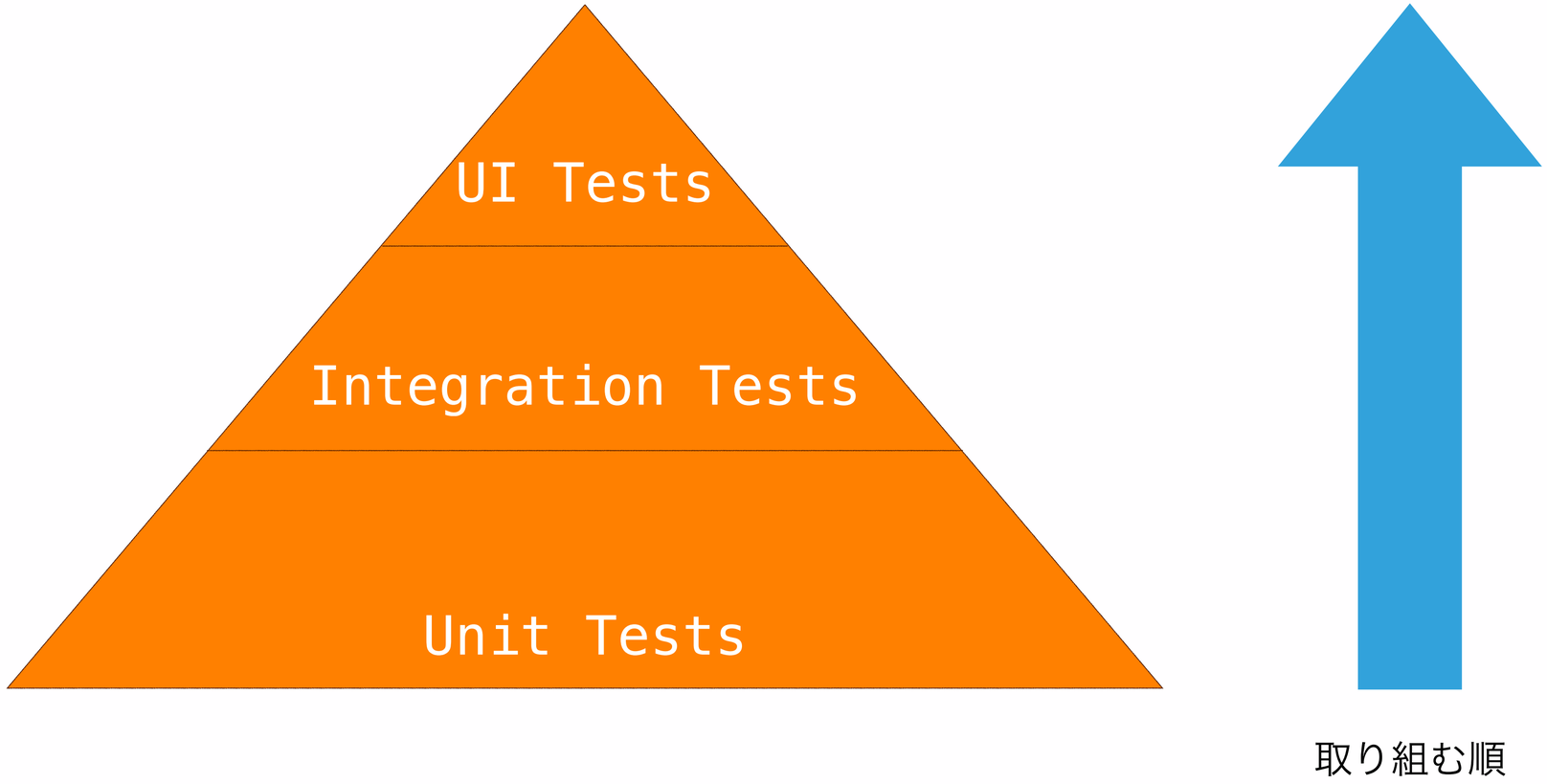




スイスチーズモデル

さながら、スイスチーズモデルのよう

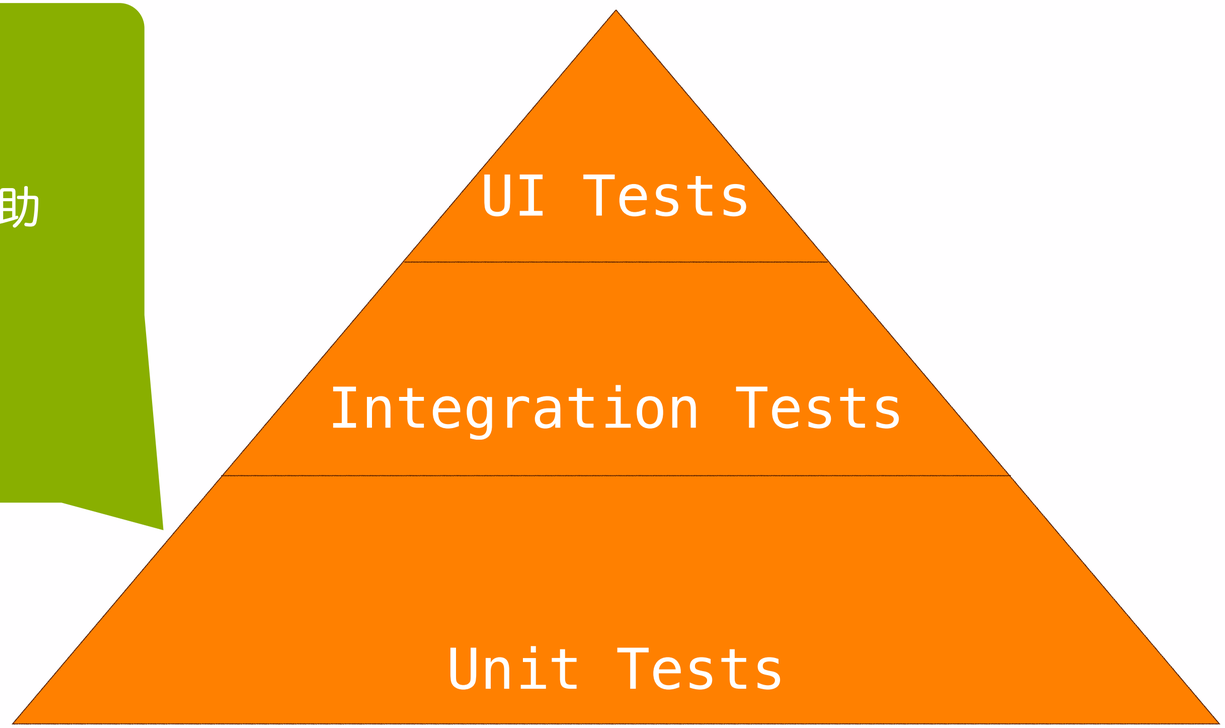
スコープを限定した実行の素早いコードから
スコープの広い実行の遅いテストへとピラミッドを上がっていく



最下層、土台

テスト範囲を狭めることで
- 問題発生時の原因特定を補助
- 実行時間の短縮

など

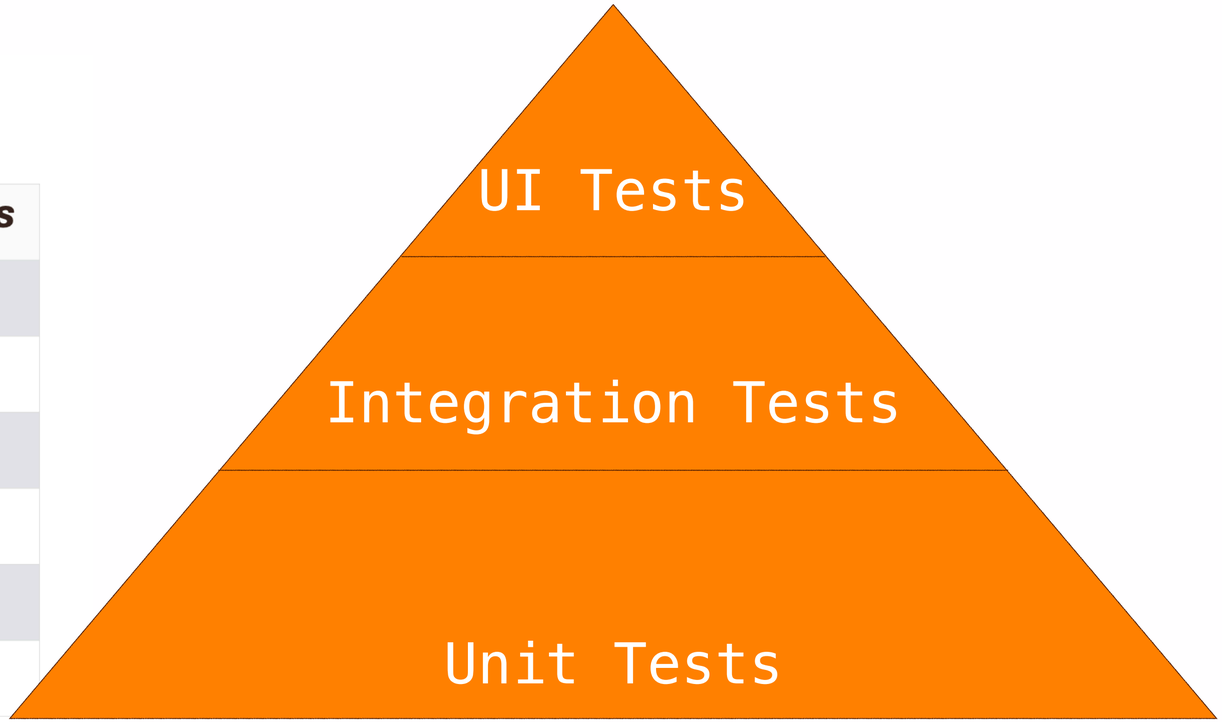


取り組む順

(再)実行時間と範囲を分けてみる

Test Size

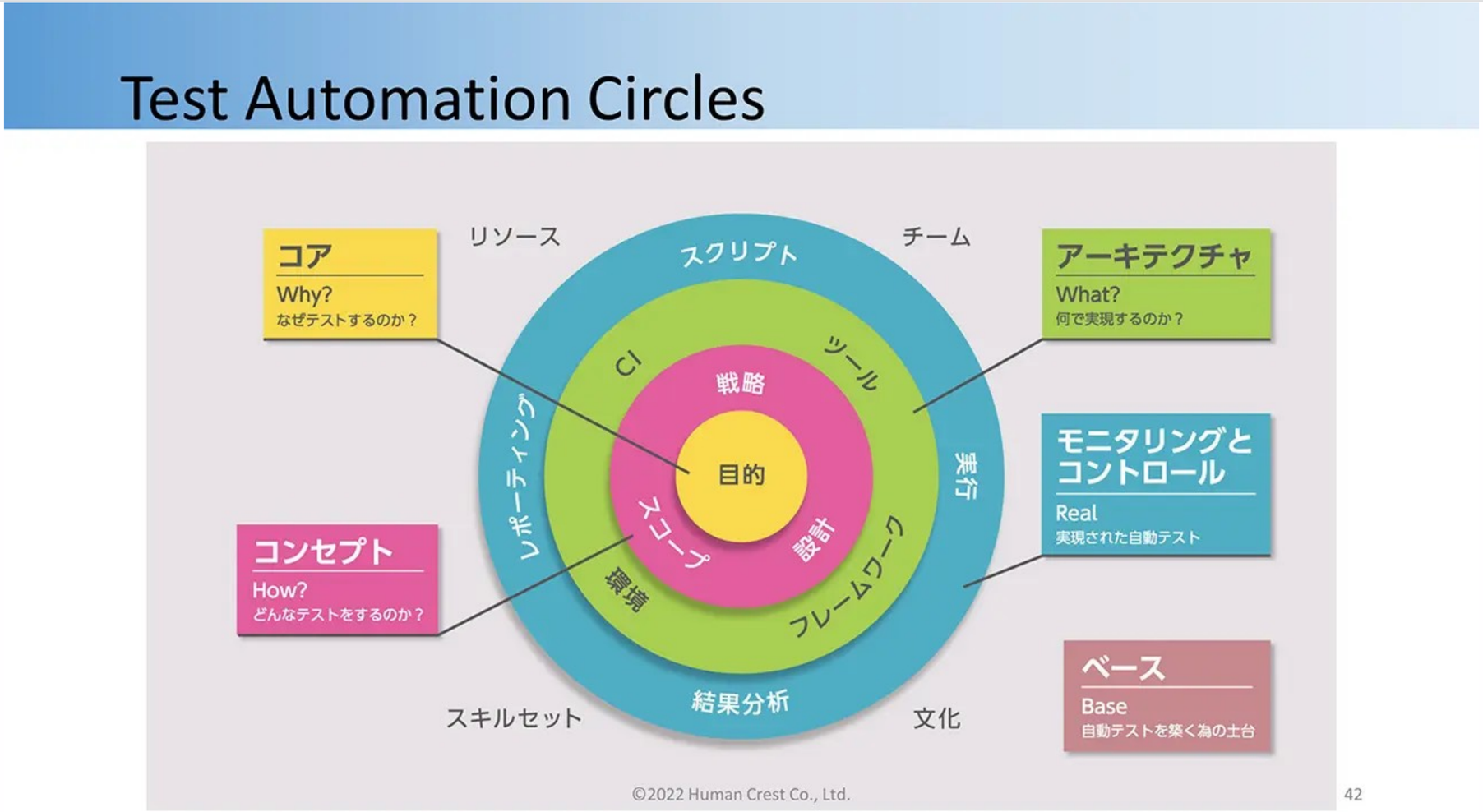
feature	*Small*	*Medium*	*Large*	*Enormous*
network	no	localhost	localhost/yes	yes
system access	no	partial/yes	yes	yes
OS(APIs)	no	yes	yes	yes
View	no	partial	yes	yes
external system	no	no	no	yes
time per a test	< 100ms	< 2s	< 120s	< 500s



多様な技術的挑戦

- 一方、実装技術の面からも全てが全て理想を達成できるわけではない
- 例えばモバイルアプリ開発では開発環境の制約から古くは逆ピラミッドにならざる追えなかった
- チームで動くため、指標の目安として実行時間、ネットワークなどの外部環境への依存、UIを操作するなどを基準に取り入れた

実行だけではない



https://speakerdeck.com/tomasagi/sohutoueatesutozi-dong-hua-falsebian-qian-bian-watutakototobian-waranaikoto-tesutozi-dong-hua-falsedao-ru-patan

ここまで

- 自動化の導入
  - ピラミッドモデル
  - テストする範囲を分解する
  - ピラミッドの区分例としての実行速度とテストする範囲

実行速度の違いを感じる

デモ

- 目的
  - テストする範囲によってテストの実行速度が異なることを経験する

内容

- 3区分のテストコード
  - Unit Test, Integration Test, UI Test
    - 実行速度が遅く、テストする範囲も広くなる
- 言語
  - Ruby

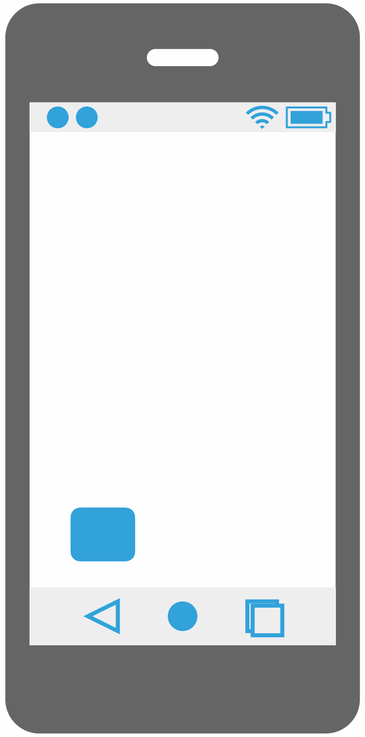
UIテスト

・ツール

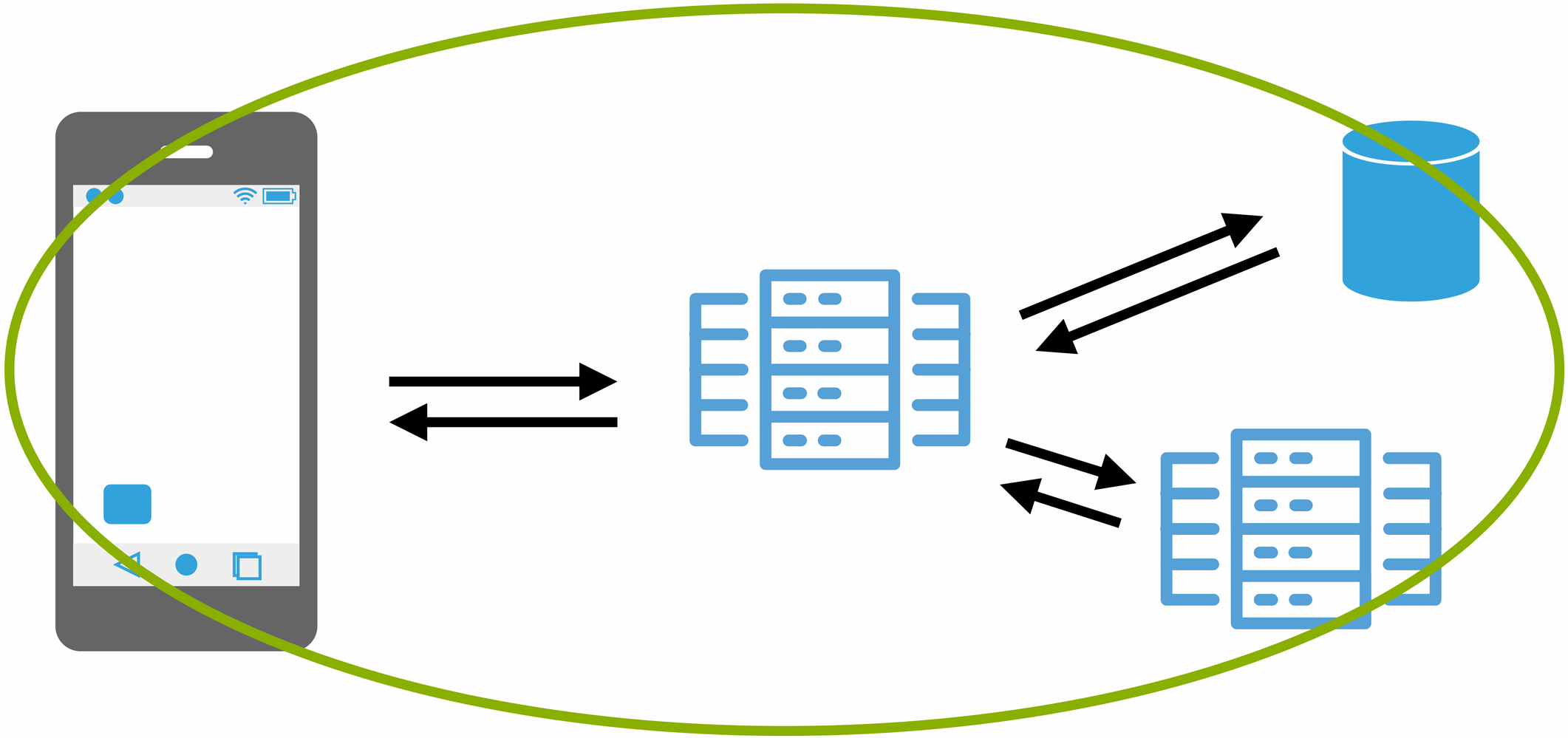
　・Appium

・シナリオ

1. Webブラウザを開く
2. https://jasst.jp/symposium/jasst22tohoku.html を入力する
3. 最下部までスクロールする
4. 最上部まで戻る



全体



デモ

Integration Test

・ツール

    ・minitest, mock library

・シナリオ 1

    1. https://jasst.jp/symposium/jasst22tohoku.html に対してHTTP GETリクエストを送る

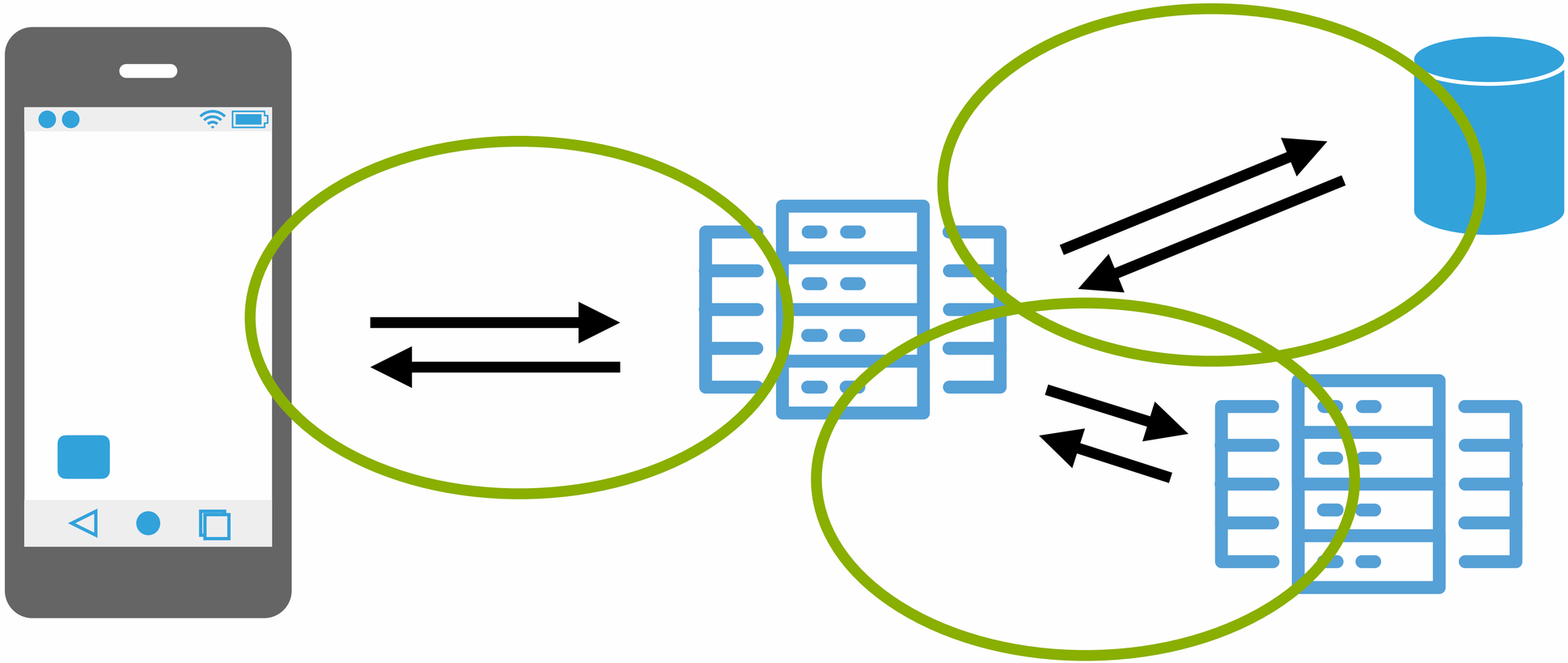
        ・特定の文言が含まれることを確認する。描画はテストには含めないが、取得できる要素を解析してテストする

・シナリオ2

    2. リクエストをモックする

        ・実際はリクエストを受け取ったがわのテスト（デモのため簡略化）

通信の境界で区切る

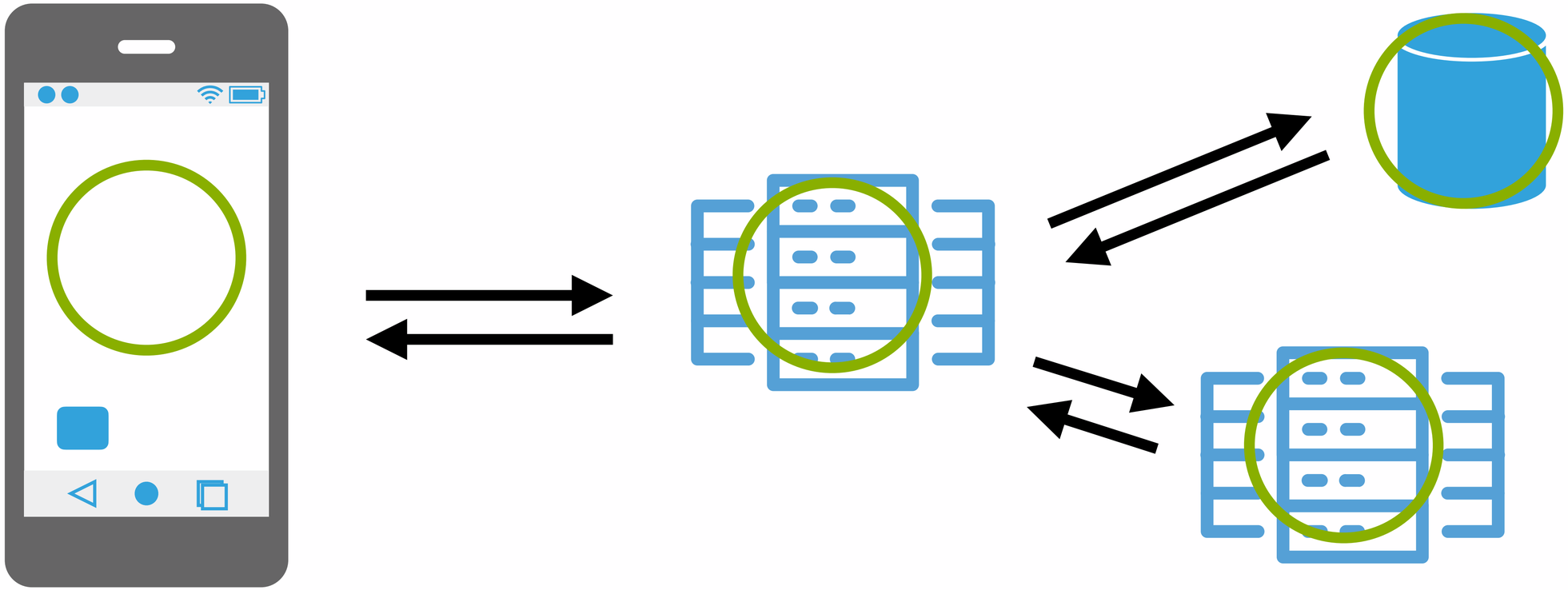


デモ

Unit Test

- ツール
  - minitest

個々のより細かな実装



デモ終

学んでいく

これから自動化を学んでいくために

・実際のプロジェクトにおいては、使っているツール、言語が異なる

・これから学ぶ場合、社内の助けを得ることが可能な、社内で使われている技術から始めると良い（ことが多い）

・OSSを活用する

・公式ドキュメントから始める

  ・例:

    ・Android https://developer.android.com/training/testing/fundamentals?hl=ja

    ・iOS:

まとめ

- テスト自動化(test automation)
  - 範囲
  - 実行速度
- これから自動化を学んでいくために

ありがとうございました

email: kazucocoa1117@gmail.com

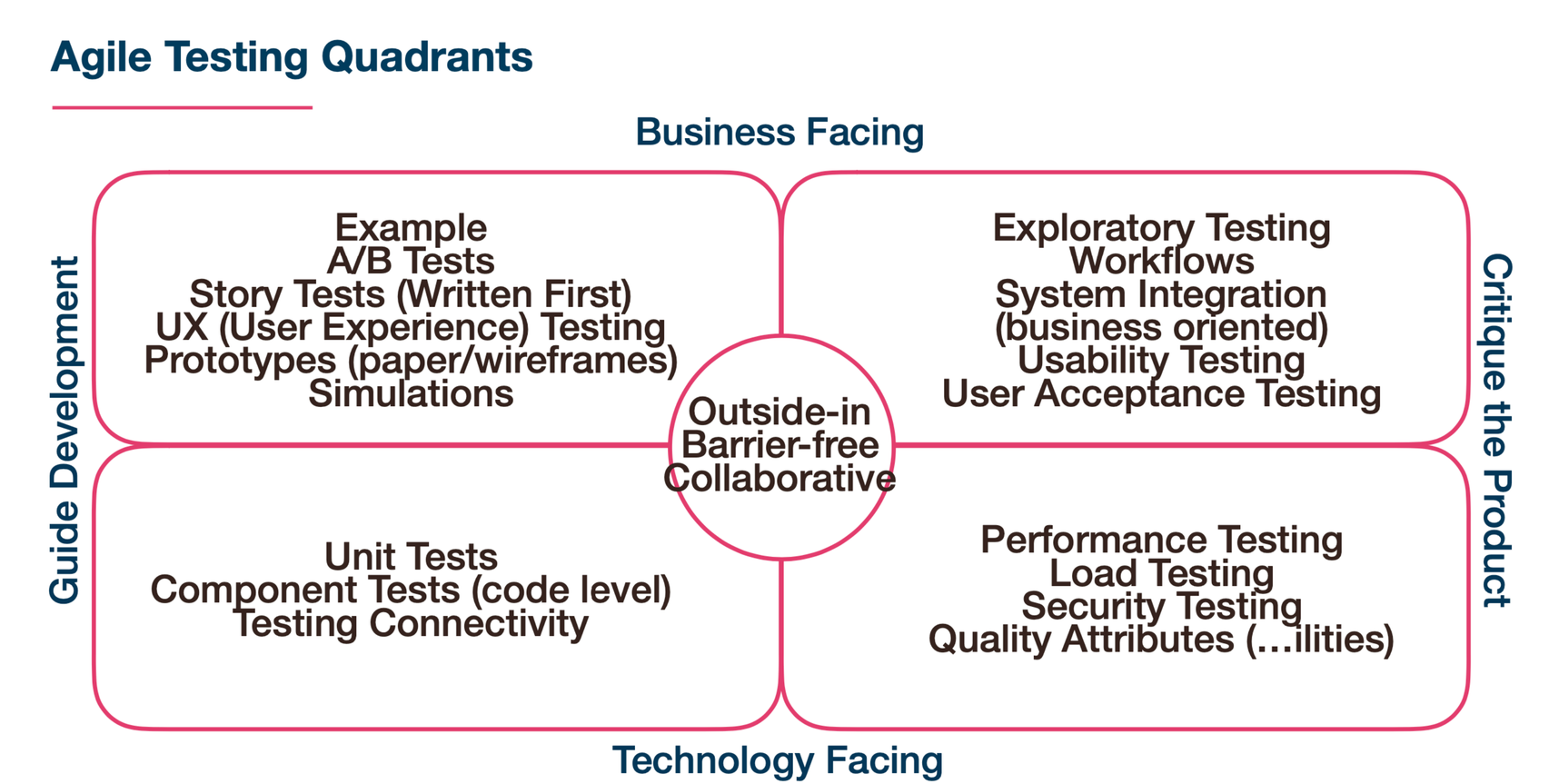
Twitter: @Kazu_cocoa

付録

モバイルアプリの特徴的なテスト

- Performance Testing
  - Testers check nonfunctional requirements unique to mobile devices (e.g., download times, processor speed, storage capacity, power availability)
- Connectivity testing
  - Testers ensure that the MobileApp can access any needed networks or Web services and can tolerate weak or interrupted network access
- Testing in the wild
  - The app is tested under realistic conditions on actual user devices in a variety of networking environments around the globe
  - Software Engineering: A Practitioner's Approach 9th Edition (Pressman) の21.2. The Testing Strategiesより

Agile testing quadrants



https://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/