IEMS5722
Mobile Network Programming and Distributed Server Architecture

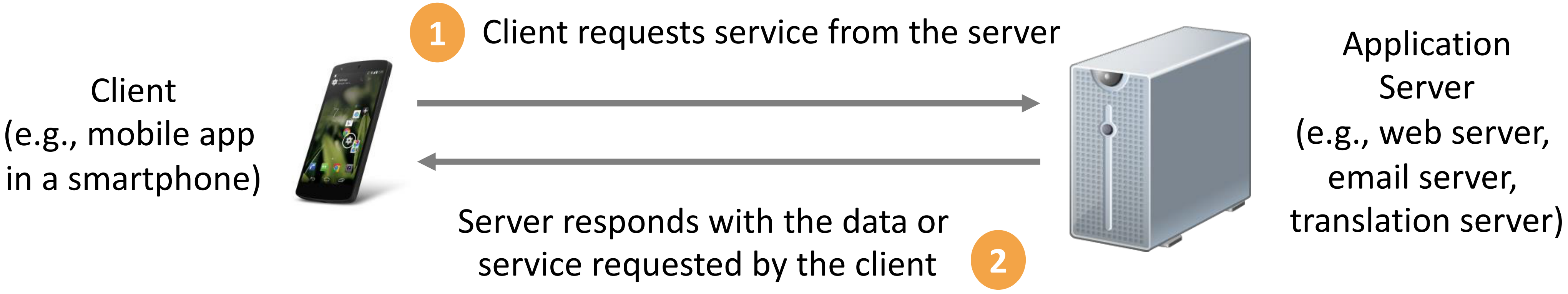
Lecture 7

Instant Messaging & Google Cloud Messaging

Push Technology

Pull and Push

- All of the examples we have gone through in network programming so far can be regarded as using the "**pull**" method
- Communication is always **initiated by the client**
- Client "pulls" data or services from the server **when necessary** (e.g. when the user launches the app, or presses a button)



Pull and Push

- **HTTP** is a **pull-based** protocol
  - Users browse the Web and **actively** decide which Website to browse, which link to follow, etc.
  - An **effective** and **economical** way (each user chooses what they need)
  - However, if some resources are regularly requested, the pull model can put heavy load on the server

Pull and Push

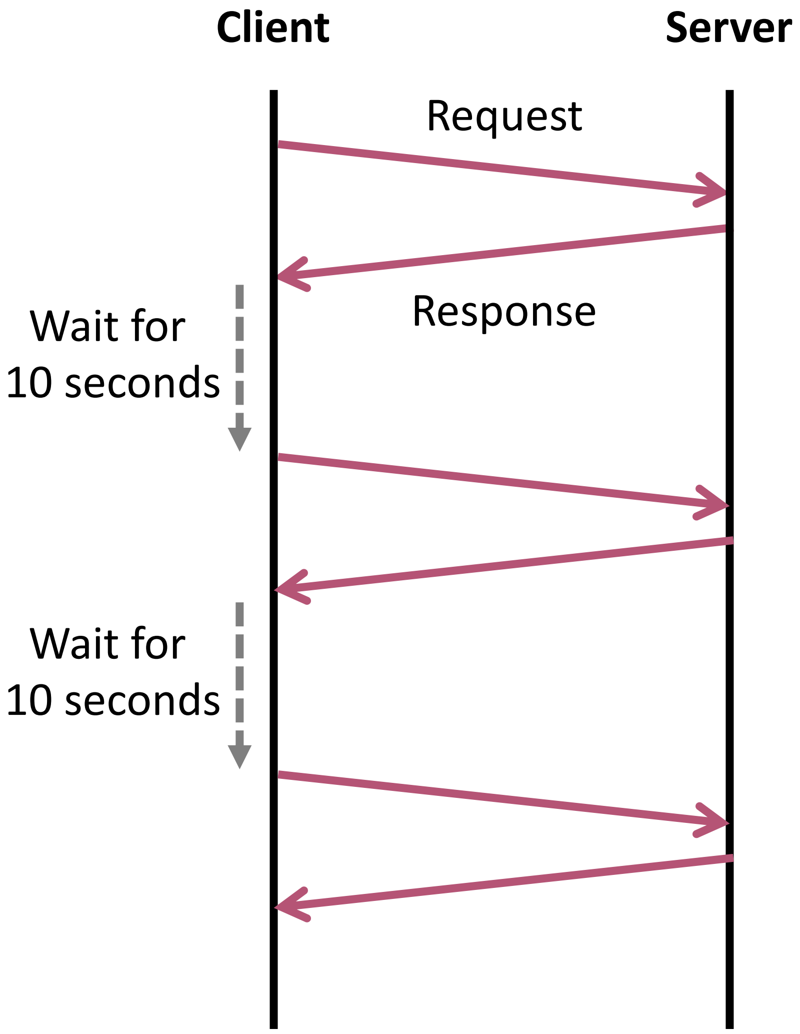
- There are cases in which the server would like to initiate communication with the client(s):
  - When new email arrives
  - When a peer sends a message to the user through the server
  - When the app/data needs to be updated
- In these cases, the server needs to "**push**" data or services to the client.
- This kind of situations is getting more and more common as **smartphones** are getting more popular.

Implementing Push

- The **World Wide Web**, and in particular the **HTTP protocol**, is designed for “**pull**”, and additional engineering is required to implement push on the Web
- Some ways to “**emulate**” push on the Web
  - Polling (periodic pull)
  - Comet Model
  - BOSH
  - WebSockets

Implementing Push – Polling

- The client **polls** the server periodically to check if new messages or updates are available
- **Advantages**:
  1. Easy to implement
  2. No extra development on the server-side
- **Disadvantages**:
  1. Unnecessary network traffic generated
  2. Extra workload on the server



Implementing Push – Polling Examples

- **Post-Office Protocol (POP)** for email
  - Email clients using the POP3 protocol make regular requests to the mail server to check for new emails

- **RSS Feed Readers**
  - RSS resources are served by HTTP, and thus are all pull-based
  - RSS feed readers poll the RSS servers regularly and check for new updates of the feeds

- **Note:** polling can place heavy workload on the server, the client and the network

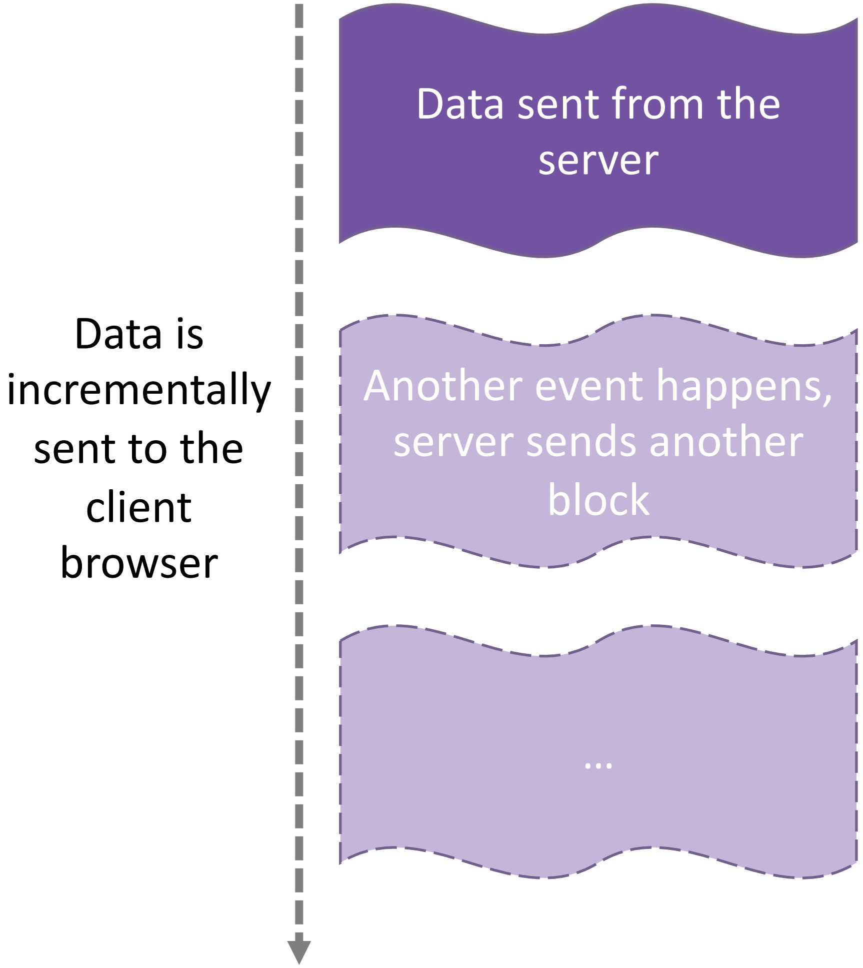
Implementing Push – Comet Model

- “Comet” is a web-application model for implementing Web applications that allow servers to push data to clients (browsers).
- Implementations of Comet applications fall into two major categories
  1. Streaming
  2. Long-polling

Reference: https://en.wikipedia.org/wiki/Comet_(programming)

Implementing Push – Comet Model

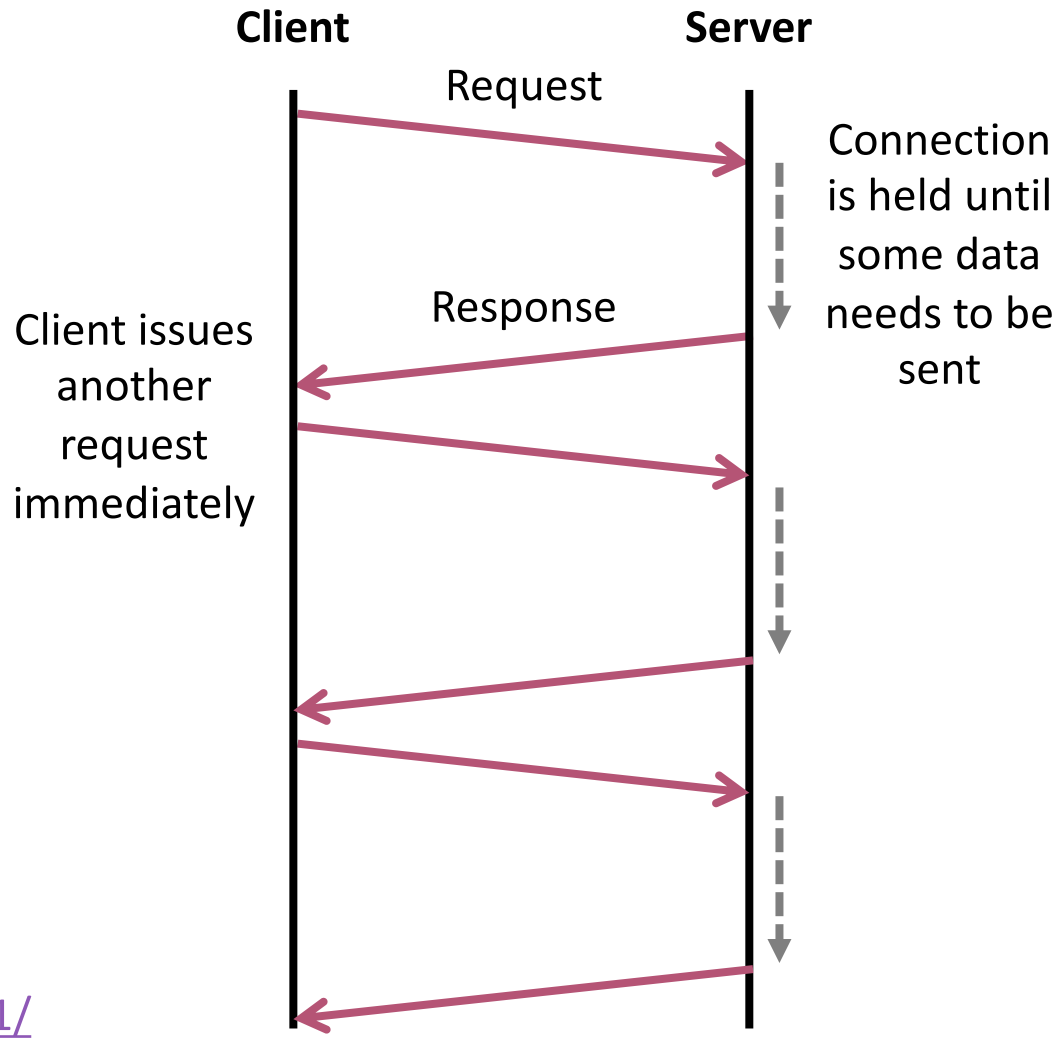
- **Streaming**
  - A persistent connection is established between the browser and the server
  - Data is sent from the server a **chunked block**
  - Events are **incrementally** sent to the browser (e.g. using <script> tags to execute JavaScript commands)


Data is incrementally sent to the client browser
Data sent from the server
Another event happens, server sends another block
…

Reference:
https://en.wikipedia.org/wiki/Comet_(programming)
https://www.ibm.com/developerworks/library/wa-reverseajax1/

Implementing Push – Comet Model

- **Long-polling**
  - A request is sent to from the client to the server
  - The server holds the connection until some events happen, then response is sent back to the client
  - The client, on receiving a response, issues another request immediately to the server
    (Usually implemented using Ajax)



Client
Server
Request
Connection is held until some data needs to be sent
Response
Client issues another request immediately

Reference:
https://en.wikipedia.org/wiki/Comet_(programming)
https://www.ibm.com/developerworks/library/wa-reverseajax1/

Implementing Push – BOSH

- **BOSH** stands for **Bidirectional-streams over Synchronous HTTP**
  - It makes use of **HTTP long-polling**
  - A **single TCP connection** is established to receive push data from server
  - If no data needs to be pushed, server sends an **empty <body/> message**
  - If client needs to send data to server, a **second socket** is used to send HTTP post requests
  - The old and new connections will then switch roles (the new will be used for long-polling thereafter)

Reference: https://xmpp.org/extensions/xep-0124.html

WebSocket

- HTTP is **half-duplex**: only one side can send data at a time, like using walkie-talkies
- WebSocket is a protocol providing **full-duplex** communications channels between two computers over a **TCP connection**
- Designed to be implemented in **Web browsers** and **Web servers**
- Communications are done over **TCP port 80**
(can be used in secured computing environments)
- Socket.io – to be introduced later

Reference:
https://tools.ietf.org/html/rfc6455
https://www.websocket.org/

WebSocket

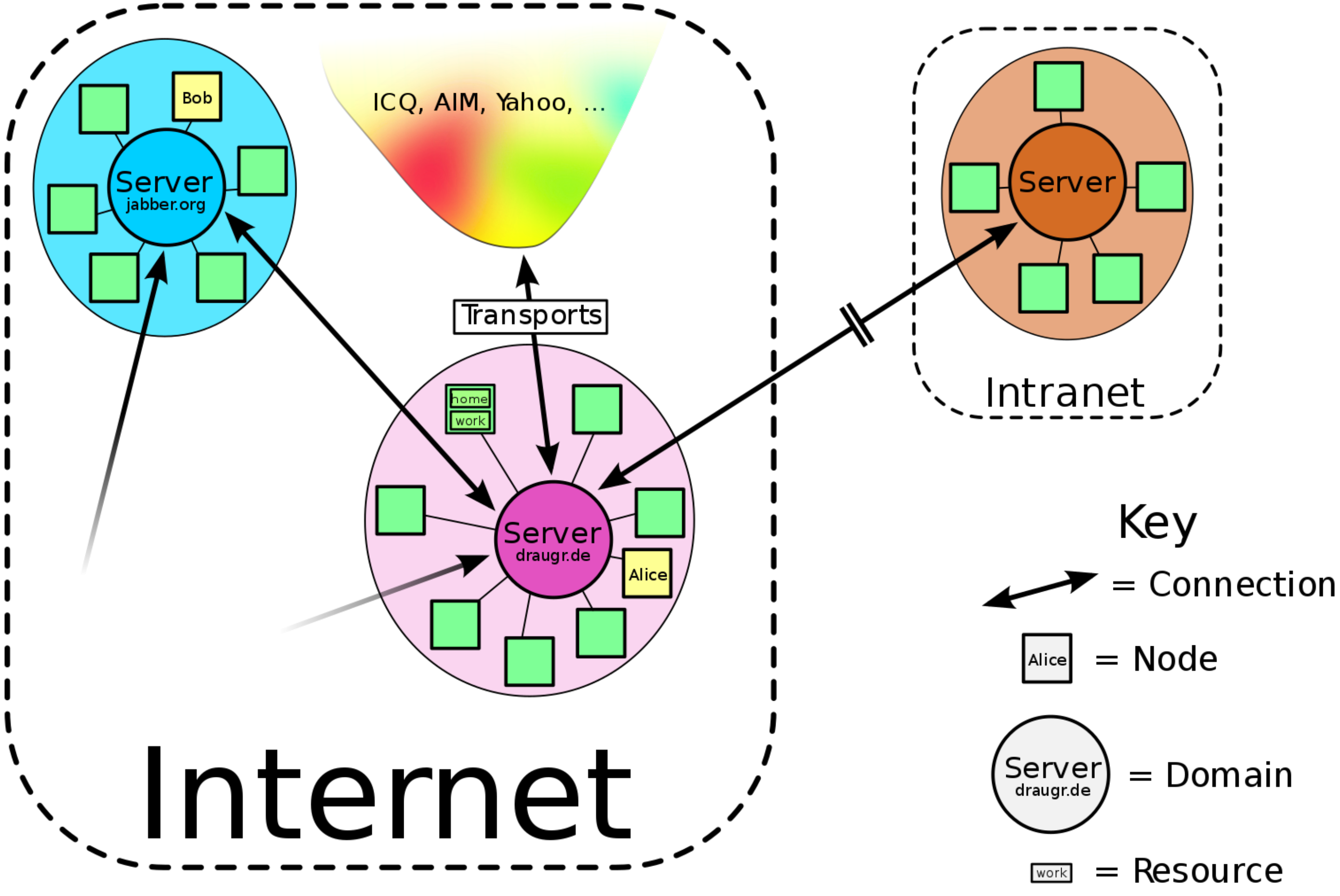
- WebSocket is part of the **HTML5** standard
  - Supported in latest versions of major Web browsers
  - Simple API in JavaScript
  - Libraries also available on iOS and Android
- Try HVBRD using your phone and computer together: https://experiments.withgoogle.com/hvbrd

```
var host = 'ws://localhost:8000/example';
var socket = new WebSocket(host);

socket.onopen = function() {
    console.log('Socket opened');
    socket.send('Hello server!');
}

socket.onmessage = function(msg) {
    console.log('Server says: ' + msg);
}

socket.onclose = function() {
    console.log('Socket closed');
}
```

XMPP

- **XMPP** stands for **Extensible Messaging and Presence Protocol**
  - Originally named **Jabber**, an open source project for **real-time** and **instant messaging**
  - Using a **client-server architecture** (non-P2P)
  - **Decentralized** and federated model: no central server, but servers for different domains
  - Each user connects to a public XMPP server, which will relay his messages to other users in other domains

XMPP

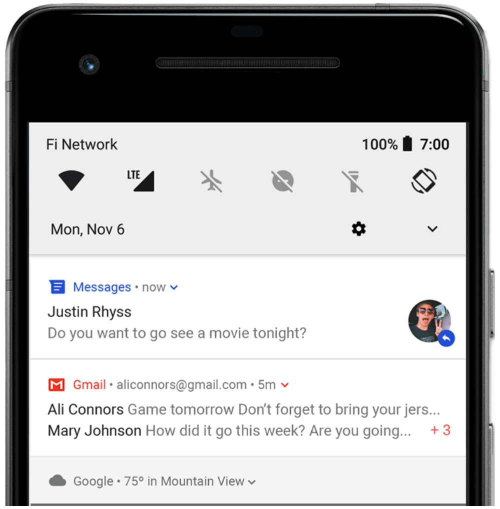


Reference: https://en.wikipedia.org/wiki/XMPP

Mobile Push

Push on Mobile Devices

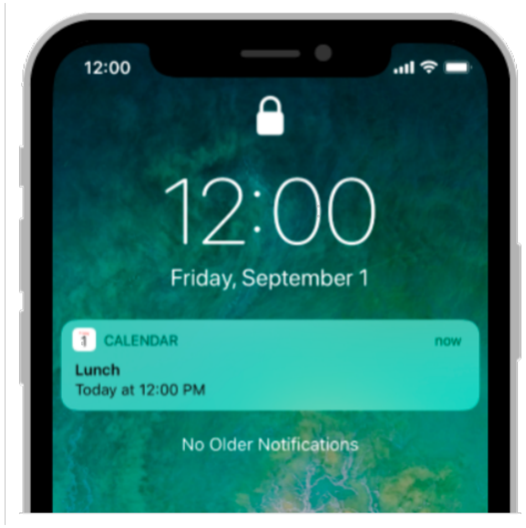
- “**Push**” is an important function on mobile devices, as it allows users to receive updates or messages without actively launching an app
- On the two popular mobile platforms, push is realized through their corresponding services:
- **iOS**:
  Apple Push Notification Service (APNS)
- **Android**:
  Google Cloud Messaging (GCM) (deprecated)
  Firebase Cloud Messaging (FCM)



Messages • now
Justin Rhyss
Do you want to go see a movie tonight?

Gmail • aliconnors@gmail.com • 5m
Ali Connors Game tomorrow Don't forget to bring your jers...
Mary Johnson How did it go this week? Are you going... +3

Google • 75° in Mountain View

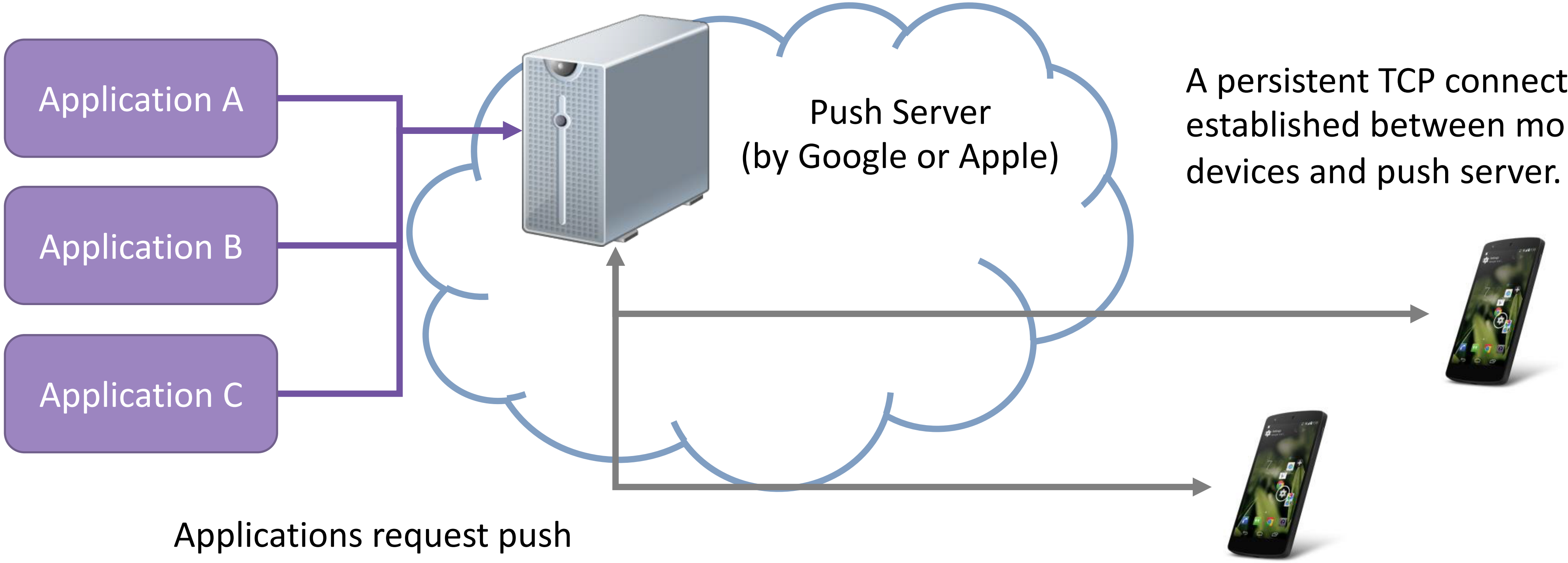


CALENDAR now
Lunch
Today at 12:00 PM

No Older Notifications

Push on Mobile Devices

- In general, push on mobile devices is implemented using the following architecture:



Application A

Application B

Application C

Push Server
(by Google or Apple)

A persistent TCP connection is established between mobile devices and push server.

Applications request push servers to push messages to clients whenever necessary.

Push on Mobile Devices

- Push notification services provided by Google and Apple are standardized ways for pushing messages to apps
- However, it is not the only way:
  - It can be used as a form of **signaling** (request for updates, short notice, etc.)
  - Your app can still implement its **own communication protocols** for exchanging data with peers and servers

Reference: https://engineering.fb.com/production-engineering/building-mobile-first-infrastructure-for-messenger/

Firebase Cloud Messaging (FCM)

Firebase Cloud Messaging

- Firebase (https://firebase.google.com/) is a platform and a set of tools that support building mobile apps. It includes real-time database, authentication, push notification, advertisement, usage analytics, and more.
- Here, we will focus on its cloud messaging function, which allows you to develop push notifications to mobile apps.

Reference:
https://firebase.google.com/docs/cloud-messaging/
https://www.youtube.com/watch?v=sioEY4tWmLI

Add Firebase using the Firebase console

- Step 1: Create a Firebase project in the Firebase console (https://console.firebase.google.com/)
  - Create a project and enter a new project name
  - (Optional) Set up Google Analytics for the project

- Step 2: Register your app with Firebase
  - Once your project is created, you can add your app to it.
  - In the Firebase console, click the Android icon.
  - Enter your app's **package name** (e.g., “hk.edu.cuhk.ie.iems5722.sandbox”)
  - (Optional) Add a nickname to your app and Debug signing certificate SHA-1

Add Firebase using the Firebase console

- Step 3: Add a Firebase configuration file
  - Download the Firebase Android config file (“**google-services.json**”), and place it into the **module** (app-level) **directory** of your app.
  - i.e., {your Android project folder} -> app -> google-services.json
  - Add Google Services plugin to your Android Gradle files (both project-level and app-level)
    - In the **root-level** (project-level) Gradle file (build.gradle):

```
buildscript {
    ...
    dependencies {
        // Add the following line:
        classpath 'com.google.gms:google-services:4.3.3'
    }
}
```

Add Firebase using the Firebase console

- Step 3: Add a Firebase configuration file (cont'd)
  - Add Google Services plugin to your Android Gradle files (both project-level and app-level)
    - In the **module** (app-level) Gradle file (usually app/build.gradle):

```
apply plugin: 'com.android.application'
// Add the following line:
apply plugin: 'com.google.gms.google-services'  // Google Services plugin
...
```

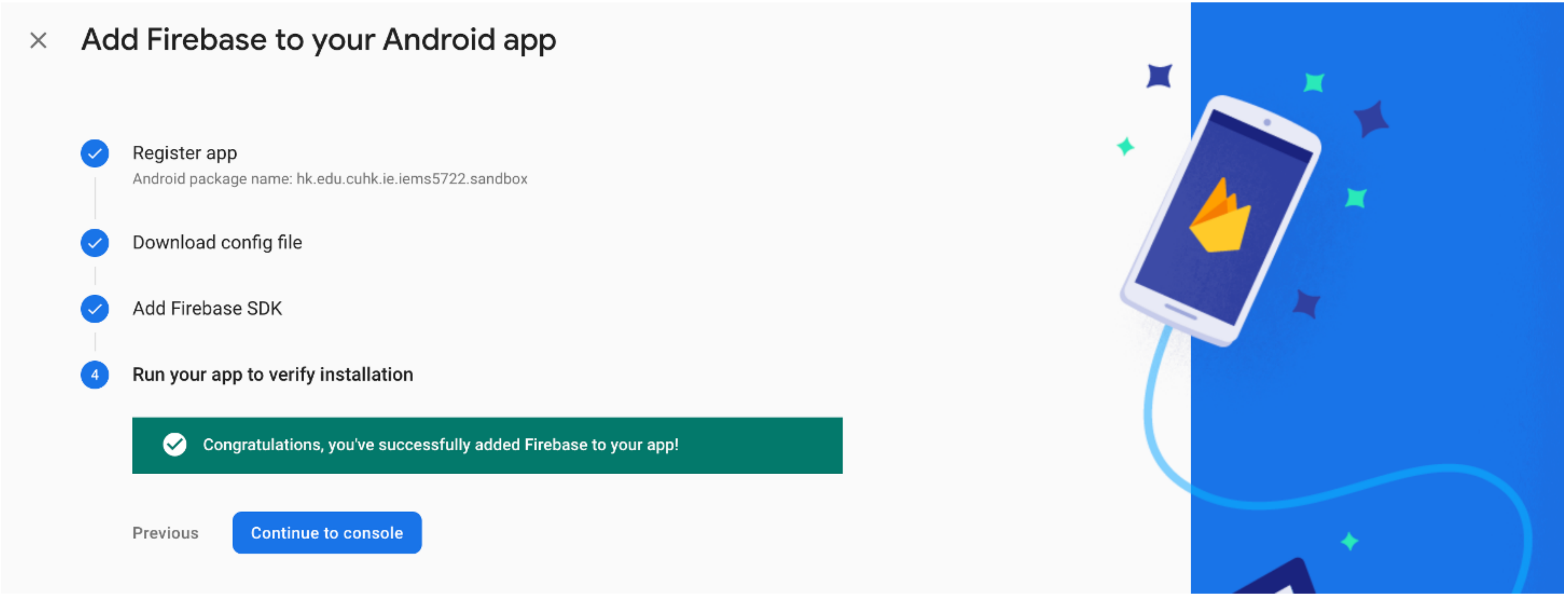
Add Firebase using the Firebase console

- Step 4: Add Firebase SDKs to your app
  - Add desired Firebase product to your app.
    - In the **module** (app-level) Gradle file (usually app/build.gradle):

```
dependencies {
    ...

    // Add the Firebase SDK for Google Analytics (if analytics are enabled)
    implementation 'com.google.firebase:firebase-analytics:17.2.3'
    // Add the Firebase Cloud Messaging SDK
    implementation 'com.google.firebase:firebase-messaging:20.1.2'

    ...
}
```

Add Firebase using the Firebase console

- Step 4: Add Firebase SDKs to your app (cont'd)
  - If you have added Analytics, build and run the app to send verification to Firebase. Skip this step if Analytics are not added.



Add Firebase to your Android app

✓ Register app

Android package name: hk.edu.cuhk.ie.iems5722.sandbox

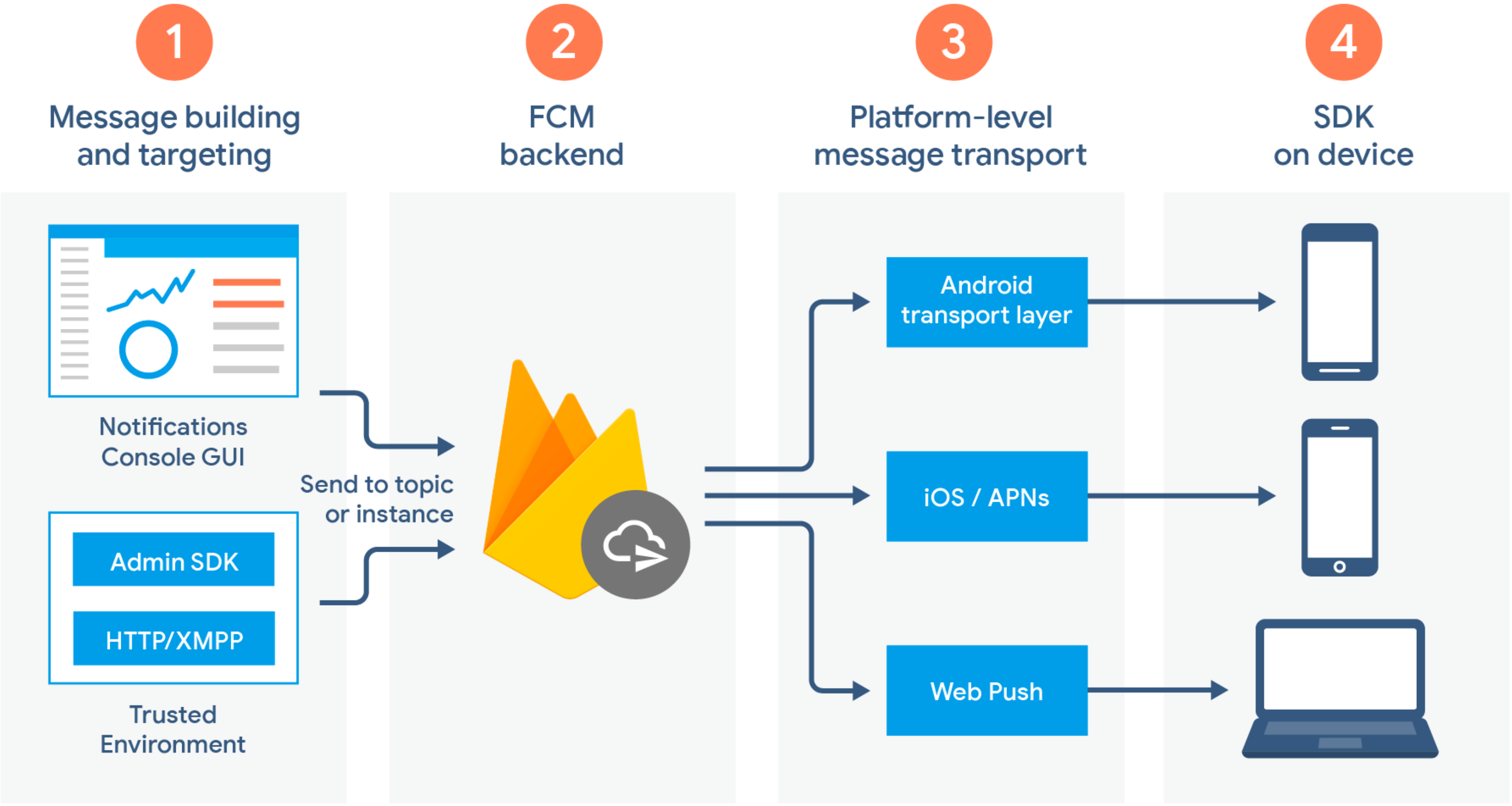
✓ Download config file

✓ Add Firebase SDK

4 Run your app to verify installation

✓ Congratulations, you've successfully added Firebase to your app!

FCM Architectural Overview


1
Message building and targeting
2
FCM backend
3
Platform-level message transport
4
SDK on device
Notifications Console GUI
Admin SDK
HTTP/XMPP
Trusted Environment
Send to topic or instance
Android transport layer
iOS / APNs
Web Push

Reference: https://firebase.google.com/docs/cloud-messaging/fcm-architecture

What you should do next?

- Next, you will have to perform the following steps in your app

1. Check whether **Google Play Services** is available on the device
2. Set up the client by extending **FirebaseMessagingService** (to handle messages)
3. Obtain the **registration token** for the app, and submit the **token** to your **application server**
4. **Handle cloud messages** received from the server

1. Check for Google Play Services

- FCM requires **Google Play Services** SDK to be installed. You should check if the service is available in two places: in **onCreate()** and **onResume()** methods in the main activity.

```
public boolean isGooglePlayServicesAvailable(Activity activity) {
    GoogleApiAvailability googleApiAvailability = GoogleApiAvailability.getInstance();
    int status = googleApiAvailability.isGooglePlayServicesAvailable(activity);
    if (status != ConnectionResult.SUCCESS) {
        if (googleApiAvailability.isUserResolvableError(status)) {
            googleApiAvailability.getErrorDialog(activity, status, 9000).show();
        }
        return false;
    }
    return true;
}
```

Reference: https://firebase.google.com/docs/cloud-messaging/android/client#sample-play

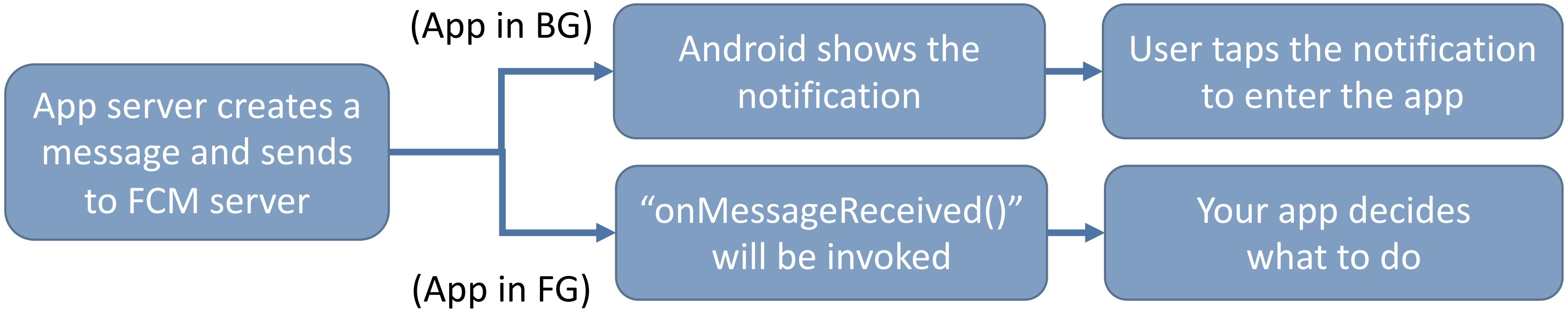
2a. Notification vs. Data Messages

- With FCM, you can send two types of messages to clients:
  - **Notification messages**, sometimes thought of as “display messages.” These are handled by the FCM SDK automatically.
    - (App in background) Notification messages are delivered to the notification tray.
    - (App in foreground) Notification messages are handled by a callback function.
  - **Data messages**, which are handled by the client app.
    - (App in either background or foreground) The client app receives the data payload in a callback function.

Reference: https://firebase.google.com/docs/cloud-messaging/concept-options

2a. Notification vs. Data Messages

- Notification Messages



App server creates a message and sends to FCM server

(App in BG)

Android shows the notification

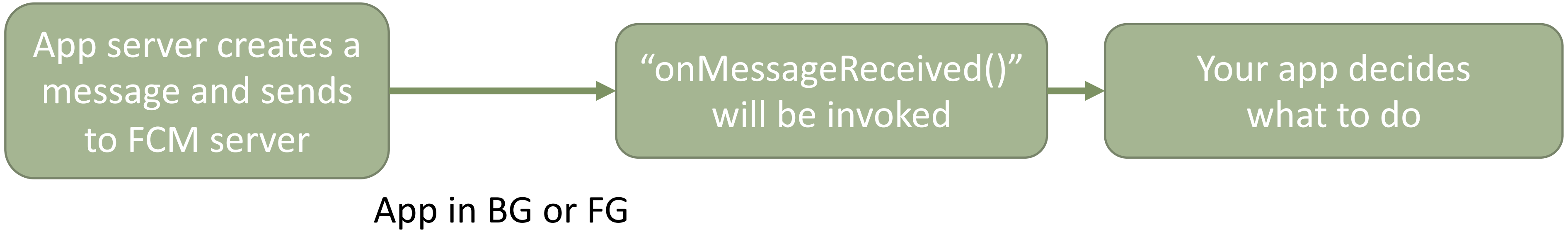
User taps the notification to enter the app

(App in FG)

“onMessageReceived()” will be invoked

Your app decides what to do

- Data Messages



App server creates a message and sends to FCM server

“onMessageReceived()” will be invoked

Your app decides what to do

App in BG or FG

2b. Setup up an Android client

- You must implement and extend **FirebaseMessagingService** if you want to
  - receive notification message when your app is in foreground, or
  - receive data message in your app, or
  - send upstream message from your app
- Add the service to the AndroidManifest.XML file:

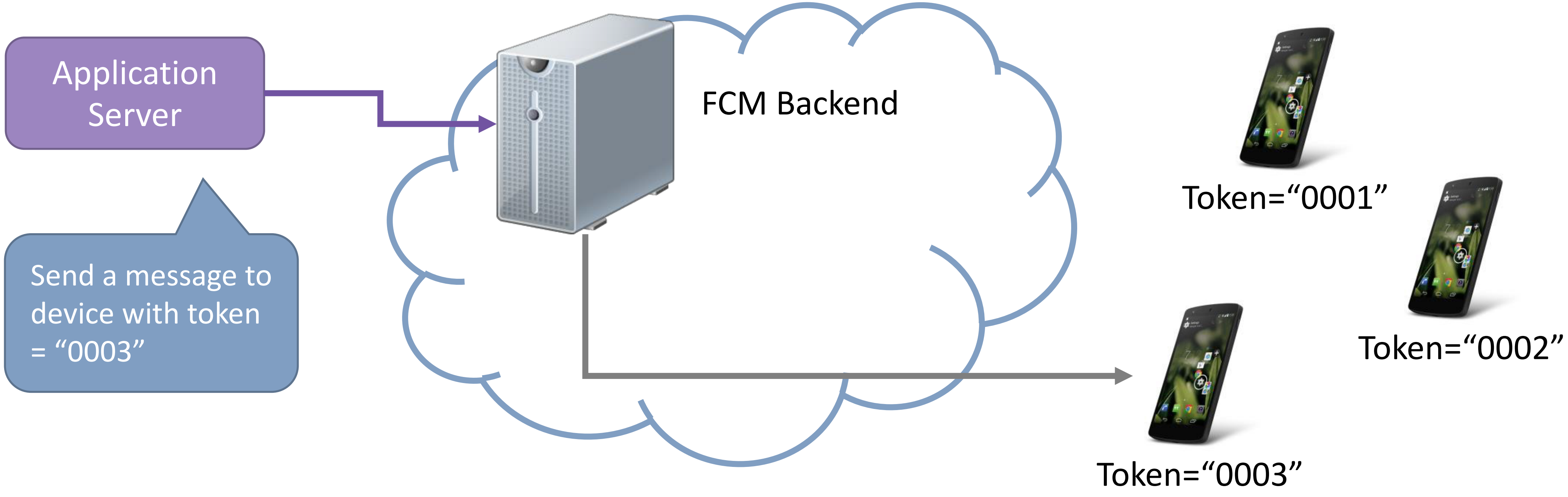
```
<service
    android:name=".MyFirebaseMessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

Your service to be implemented

Reference: https://firebase.google.com/docs/cloud-messaging/android/client

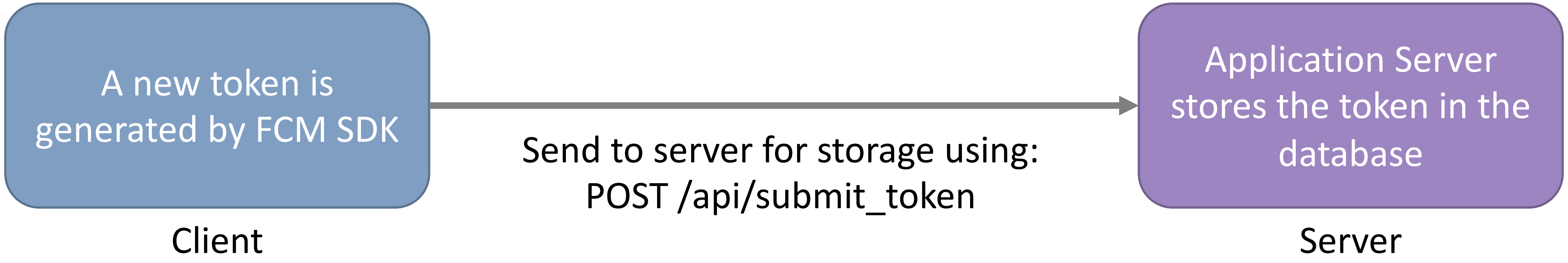
3. Registration Token

- When the app is first installed, the FCM SDK will generates a registration token to the app.
- With the token, the application server to push a message to a single device.



3. Registration Token

- When the new token is generated, the app should send it to the application server for storage and identification.
- We can use a HTTP POST request to an API implemented on the server.


A new token is generated by FCM SDK
Client
Send to server for storage using:
POST /api/submit_token
Application Server stores the token in the database
Server

3. Registration Token

- To monitor token generation, you must implement and extend **FirebaseMessagingService**.

Create a service extending FirebaseMessagingService

```java
public class MyFirebaseMessagingService extends FirebaseMessagingService {
    private static final String TAG = "MyFirebaseMessagingService";

    ... // Other overridden methods

    @Override
    public void onNewToken(String token) {
        Log.d(TAG, "Refreshed token: " + token);

        sendRegistrationToServer(token);
    }

    private void sendRegistrationToServer(String token) {
        // Implement your own logic to submit token to server
    }
}
```

4. MyFirebaseMessagingService

- To receive messages, you also need to use a service that extends **FirebaseMessagingService**.
- We can use the same one that we use to monitor tokens.
- The **onMessageReceived()** callback function will be invoked, except
  - When a notification message is received and the app is in background

- **Note:**
  Afterwards, you can send test message from **Notification Composer** in the Firebase Console (more in the later slides)

4. MyFirebaseMessagingService

Use the same service extending FirebaseMessagingService

```
public class MyFirebaseMessagingService extends FirebaseMessagingService {
    private static final String TAG = "MyFirebaseMessagingService";

    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        Log.d(TAG, "From: " + remoteMessage.getFrom());
        // Check if message contains a data payload.
        if (remoteMessage.getData().size() > 0) {
            Log.d(TAG, "Message data payload: " + remoteMessage.getData());
        }
        // Check if message contains a notification payload.
        if (remoteMessage.getNotification() != null) {
            Log.d(TAG, "Message notification payload: " +
                remoteMessage.getNotification().getBody());
        }
        ...
    }
}
```

This callback function will be called when an FCM message is received (except for a notification message is received when app is in background)

Once the data is received, your app can:

- Generation a notification
- Jump to a new page
- Send request to server to fetch new data

Implementing FCM Client

- You can find a complete example here: https://github.com/firebase/quickstart-android/tree/master/messaging

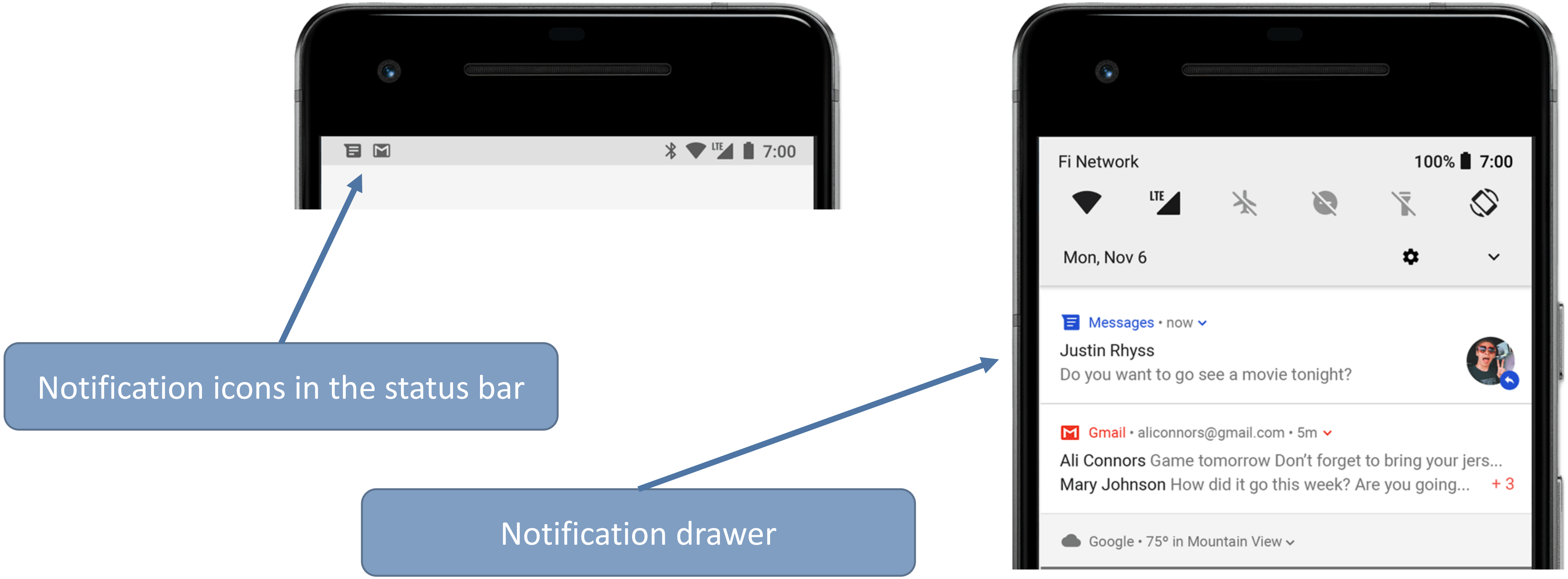
Other Similar Solutions

- Other third-party cloud messaging SDKs and APIs
  - Baidu Push (free)
    https://push.baidu.com/doc/guide/index
  - MiPush (free)
    https://dev.mi.com/console/appservice/push.html
  - Tencent Cloud
    https://cloud.tencent.com/product/tpns/developer
  - Aliyun MNS (Message Notification Service)
    https://help.aliyun.com/product/27412.html
  - Amazon's SNS (Simple Notification Service) (free tier available)
    https://aws.amazon.com/sns/

Notifications in Android

Notifications

- Notifications are ways to notify the user in the notification bar about updates of your app, outside of your app's UI.



Fi Network

100% 7:00

Mon, Nov 6

Messages • now

Justin Rhyss

Do you want to go see a movie tonight?

Gmail • aliconnors@gmail.com • 5m

Ali Connors Game tomorrow Don't forget to bring your jers...

Mary Johnson How did it go this week? Are you going... + 3

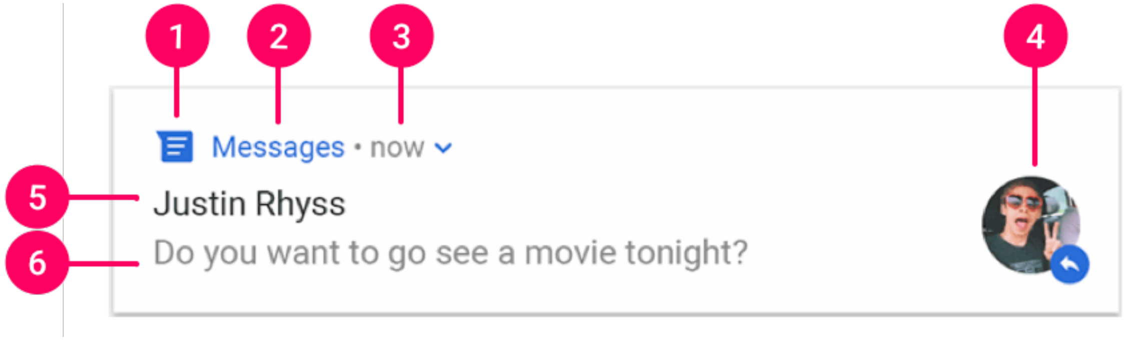
Google • 75° in Mountain View

Notification icons in the status bar

Notification drawer

Notifications

- A notification in Android contains a few important components
  - 1. Small icon (usually the app icon, but can be customized)
  - 2. App name (provided by the system)
  - 3. Timestamp (automatically generated, can be customized or hidden)
  - 4. Large icon (optional)
  - 5. Title
  - 6. Text



Messages • now

Justin Rhyss

Do you want to go see a movie tonight?

Reference:
https://developer.android.com/guide/topics/ui/notifiers/notifications
https://material.io/design/platform-guidance/android-notifications.html#usage

Creating Notifications

- The following block of codes can be used to create a notification.

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Hello World!")
    .setPriority(NotificationCompat.PRIORITY_DEFAULT);

...

NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);

// notificationId is a unique int for each notification that you must define
notificationManager.notify(notificationId, builder.build());
```

Creating Notifications

- You can also control what happens when the user taps on the notification.

```
// Create an explicit intent for an Activity in your app
Intent intent = new Intent(this, AlertDetails.class);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        // Set the intent that will fire when the user taps the notification
        .setContentIntent(pendingIntent)
        .setAutoCancel(true);
...
```

Reference: https://developer.android.com/training/notify-user/build-notification

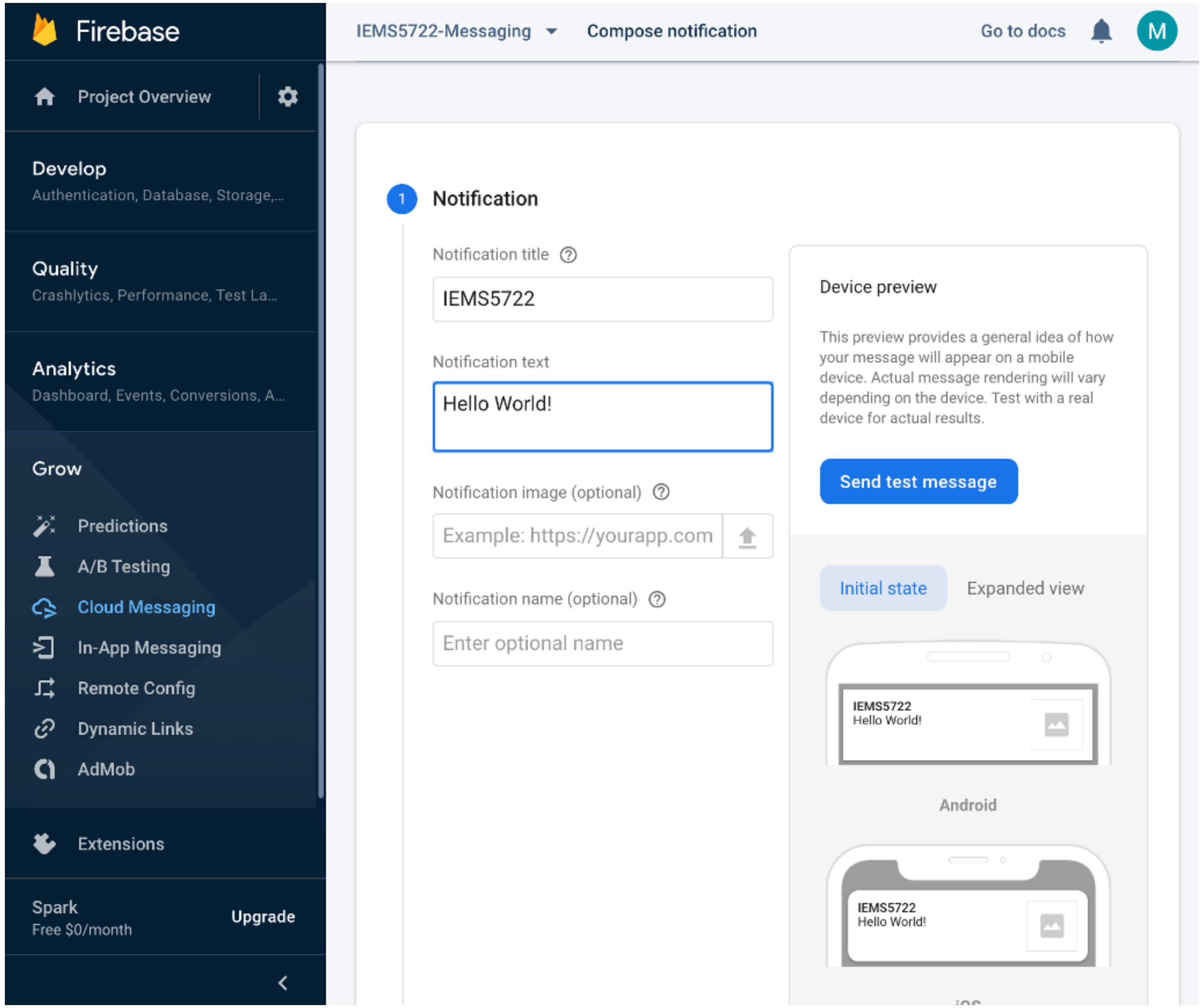
Sending FCM Messages

Sending FCM Messages

- There are two ways to send out FCM messages to your Android apps
  1. Using the **Firebase Console**
  2. Generate a **send request** and submit it to the FCM connection server

Using the Firebase Console

- Sign into Firebase Console at https://console.firebase.google.com
- Select your project
- Go to “Grow” → “Cloud Messaging” → “New notification”
- Fill in the form, review and publish



Notification title: IEMS5722

Notification text: Hello World!

Notification image (optional): Example: https://yourapp.com

Notification name (optional): Enter optional name

Device preview

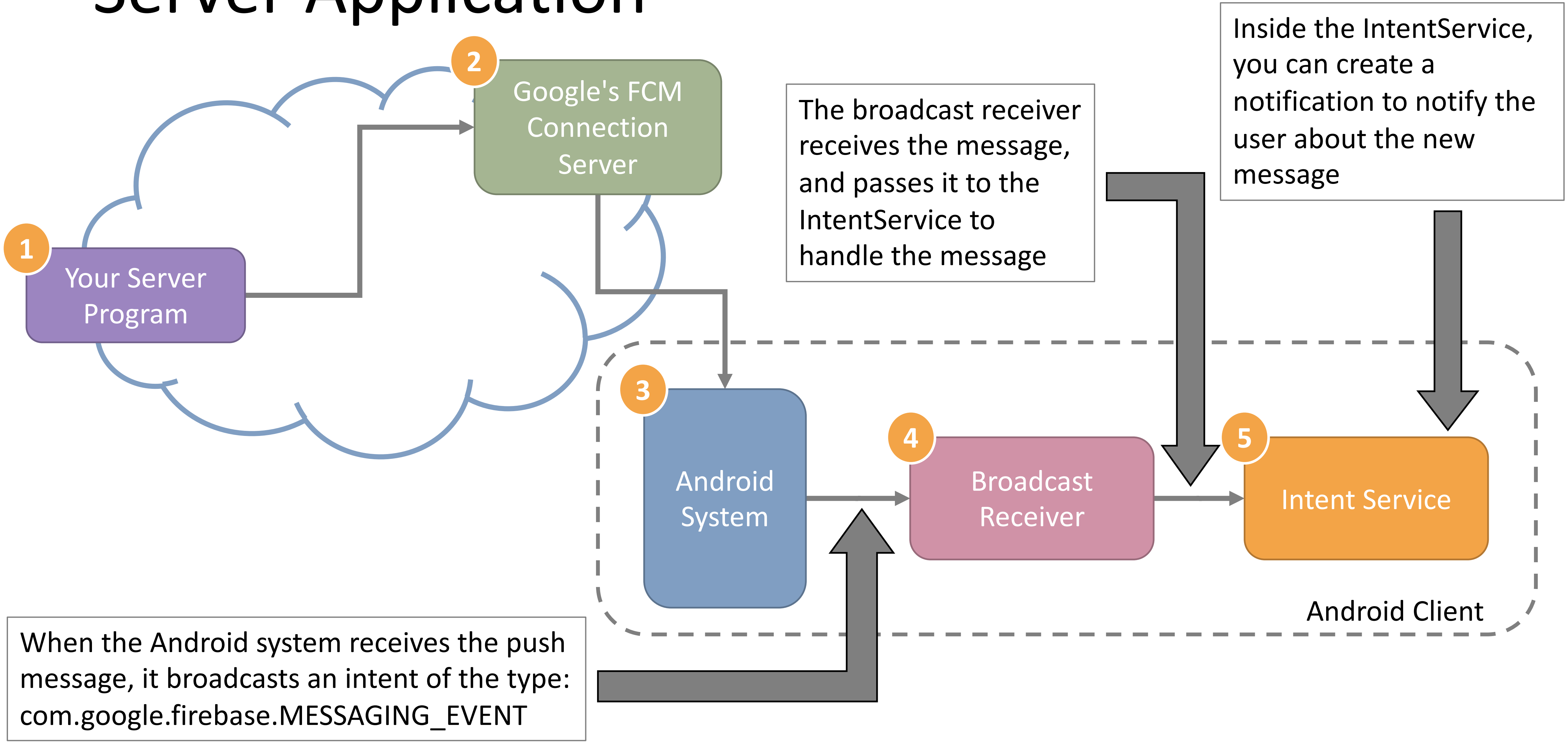
This preview provides a general idea of how your message will appear on a mobile device. Actual message rendering will vary depending on the device. Test with a real device for actual results.

IEMS5722
Hello World!

Server Application

- More often, the need to generate a push message to an Android app is **associated with some actions** taken by the users. For example:
  - When one user sends a message to another user
  - When the user has set a reminder in the app
  - When a new user joins a game in a game app
  - …
- Hence, you would like to be able to generate FCM messages in your server application

Server Application



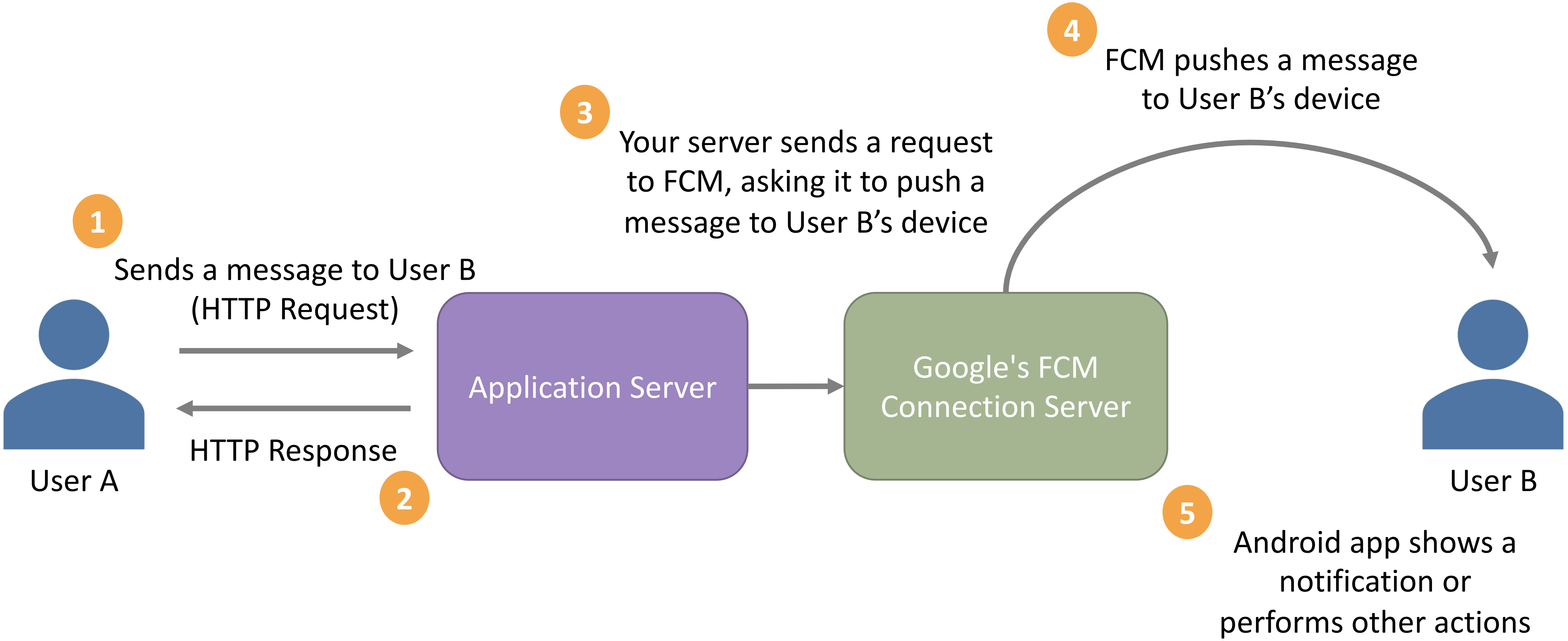
The broadcast receiver receives the message, and passes it to the IntentService to handle the message

Inside the IntentService, you can create a notification to notify the user about the new message

When the Android system receives the push message, it broadcasts an intent of the type: com.google.firebase.MESSAGING_EVENT

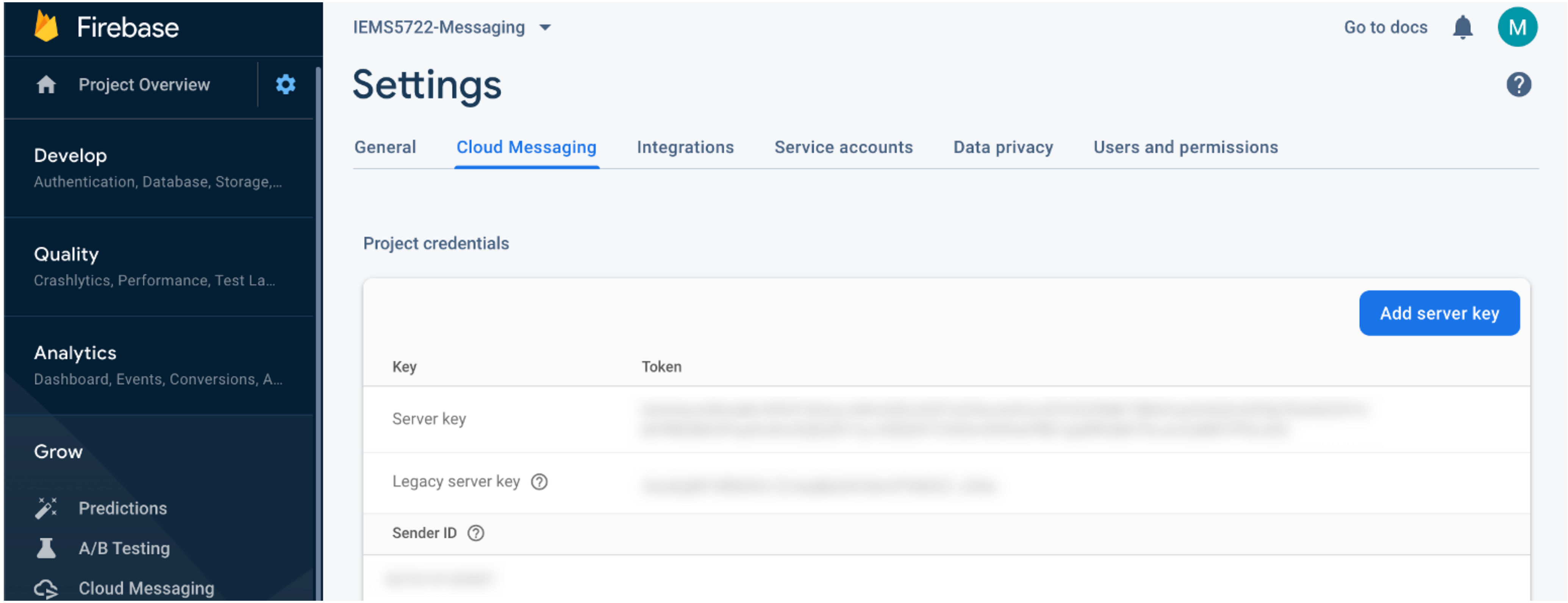
Server Application

- For example, in an instant messaging app, User A sends a message to User B.


4
FCM pushes a message to User B’s device
3
Your server sends a request to FCM, asking it to push a message to User B’s device
1
Sends a message to User B (HTTP Request)
Application Server
Google's FCM Connection Server
HTTP Response
2
User A
User B
5
Android app shows a notification or performs other actions

Server Application

- To send requests to FCM connection servers, server key is needed
- Go to the Firebase Console → Settings (the gear icon) → Project Settings → Cloud Messaging. We will use “Legacy server key” for simplicity.



Settings

Project credentials

Key	Token
Server key	[redacted]
Legacy server key	[redacted]
Sender ID	
[redacted]	

Send Requests

- You can send requests using either **HTTP** or **XMPP**
- Characteristics of **HTTP**
  - Supports **downstream** only (cloud-to-device)
  - **Synchronous** (your server sends a request and wait for response)
  - Allow sending message to multiple devices (**multicasting**)
- Characteristics of **XMPP**
  - Support **downstream** and **upstream**
  - **Asynchronous**
  - Does not support multicasting

Send Requests using Legacy HTTP Protocol

- An example of a request sent to the FCM HTTP Connection Server. API endpoint: https://fcm.googleapis.com/fcm/send
- You can either construct and submit HTTP requests by yourself, or use the Python FCM module

```
Content-Type:application/json
Authorization:key=AIzaSyZ-1u...0GBYzPu7Udno5aA

{
    "to" : "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "data" : {
        ...
    },
}
```

Reference:
https://firebase.google.com/docs/cloud-messaging/http-server-ref
https://firebase.google.com/docs/cloud-messaging/send-message#send-messages-using-the-legacy-app-server-protocols

Send Requests using Legacy HTTP Protocol

- For example, a **notification** message may look like this:

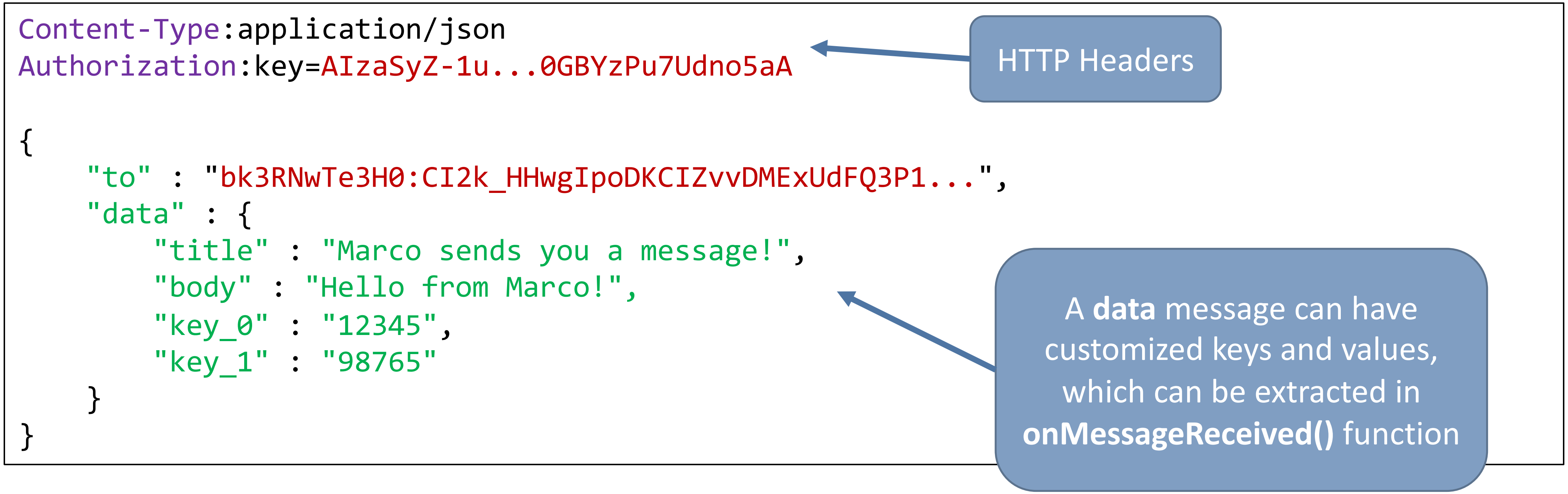
```
Content-Type:application/json
Authorization:key=AIzaSyZ-1u...0GBYzPu7Udno5aA

{
    "to" : "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "notification" : {
        "title" : "Marco sends you a message!",
        "body" : "Hello from Marco!"
    }
}
```

HTTP Headers

A **notification** message must have **title** and **body**

Send Requests using Legacy HTTP Protocol

- For example, a **data** message may look like this:



```
Content-Type:application/json
Authorization:key=AIzaSyZ-1u...0GBYzPu7Udno5aA

{
    "to" : "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "data" : {
        "title" : "Marco sends you a message!",
        "body" : "Hello from Marco!",
        "key_0" : "12345",
        "key_1" : "98765"
    }
}
```

HTTP Headers

A **data** message can have customized keys and values, which can be extracted in **onMessageReceived()** function

Sending HTTP Requests in Python

- In your Python server program, you can make use of the requests module to help you send HTTP requests https://requests.readthedocs.io/en/master/
- In case it is not installed

```
$ sudo pip3 install requests
```

- A simple HTTP library for Python

```
>>> import requests
>>> r = request.get("http://www.cuhk.edu.hk/")
>>> status_code = r.status_code
>>> content = r.text
...
```

Sending HTTP Requests in Python

- Sending a POST request to the FCM server

```python
import requests

api_key = "AIzaSyZ-1u...0GBYzPu7Udno5aA"
url = "https://fcm.googleapis.com/fcm/send"

headers = {
    "Content-Type" : "application/json",
    "Authorization" : "key=" + api_key,
}

device_token = "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1..."
payload = {
    "to" : device_token,
    "notification" : {...}
}

r = requests.post(url, headers=headers, json=payload)

if r.status_code == 200:
    print("Request is sent to FCM server successfully!")
```

Python FCM Module

- Instead of constructing the HTTP request by yourself, you can send FCM request more conveniently using the pyfcm module http://olucurious.github.io/PyFCM/
- Install the module by the following command:

```
$ sudo pip3 install pyfcm
```

Python FCM Module

- An example of using the Python FCM module to send a **notification** message

```
from pyfcm import FCMNotification

push_service = FCMNotification(api_key = "<api-key>")

registration_id = "<device registration_id>"
message_title = "Marco sends you a message!"
message_body = "Hello from Marco!"

result = push_service.notify_single_device(
    registration_id = registration_id,
    message_title = message_title,
    message_body = message_body)
```

Project Arrangement

Project Arrangement

- By this time, you should have formed your project group
- Next, think about the topic of your project, and what you plan to do
- **Some guidelines**
  - Instead of functions, think about what problems you want to solve
  - Your app should make use of networking functions (e.g. send and receive data, files, broadcasting or streaming, multi-player games, local area connectivity, etc.)
  - A 3-student group should do more work than a 2-student group

Project Arrangement

- You should now prepare a simple **document** with the following content
  - What problem(s) is your app going to solve?
  - What functions or features will be found in your app?
  - What are the tasks required (1) on the client side, and (2) on the server side?
  - What difficulties do you foresee?
  - What kind of help do you need?
- No specific format required
- Does not count for marks, but will be useful for your planning, division of labour and asking me for comments.

Project Arrangement

- **Assessment Criteria** (Tentative)
  - Networking functionality (30%)
  - Client-side system design and complexity (20%)
  - Serve- side system design and complexity (20%)
  - Error handling and robustness (10%)
  - User-friendliness & UI/UX design (10%)
  - Presentation & report (10%)

Next Lecture: Peer-to-Peer Networking in Android

End of Lecture 7