LA TROBE UNIVERSITY

All kinds of clever

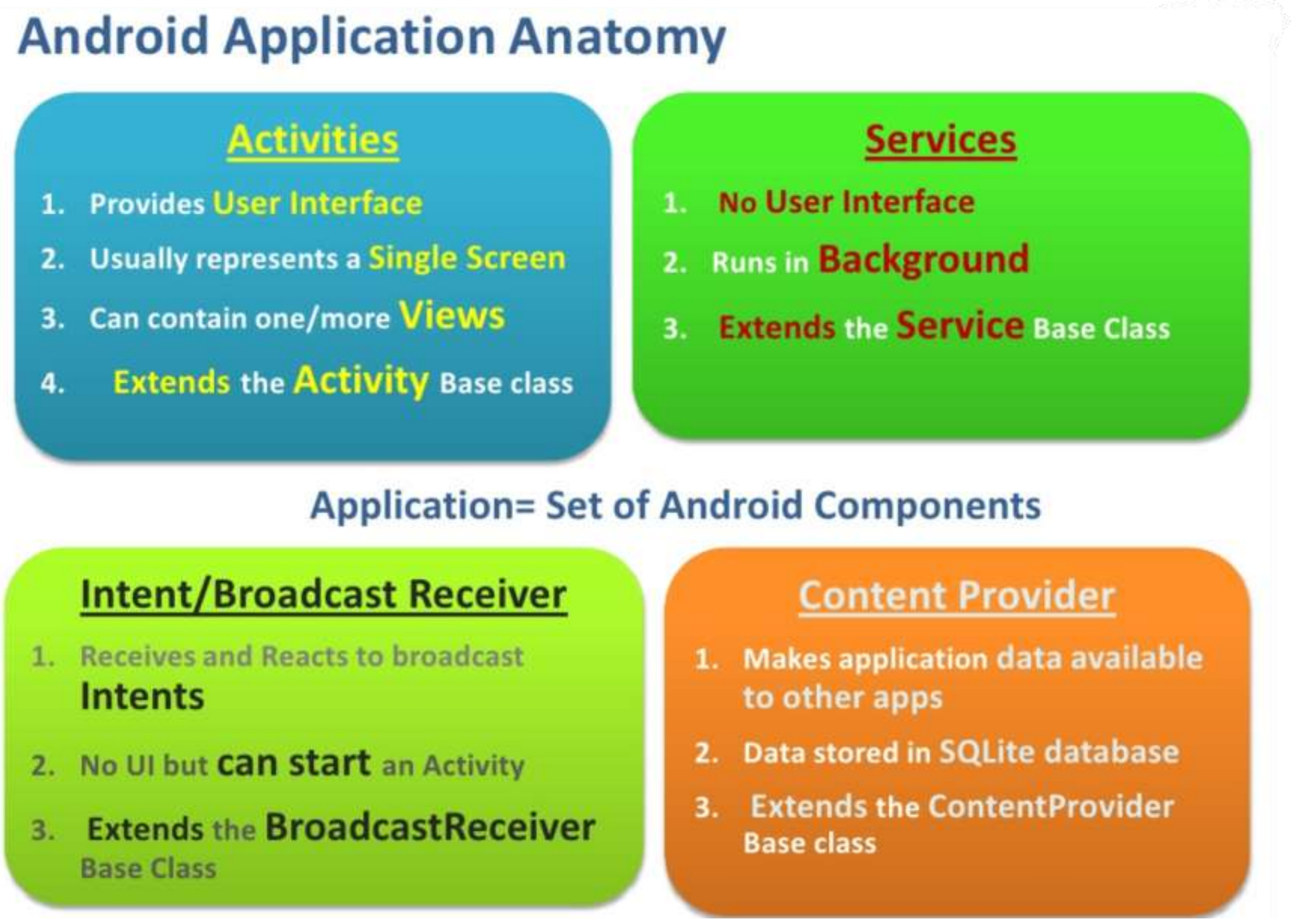
CSE2MAD

Mobile Application Development
Lecture 5 Part 1

Outline

- Recap last week’s lecture + lab
- Android App Core Components – Services
- Wifi & Bluetooth

Recap:



Android Application Anatomy

Activities

1. Provides User Interface
2. Usually represents a Single Screen
3. Can contain one/more Views
4. Extends the Activity Base class

Services

1. No User Interface
2. Runs in Background
3. Extends the Service Base Class

Application= Set of Android Components

Intent/Broadcast Receiver

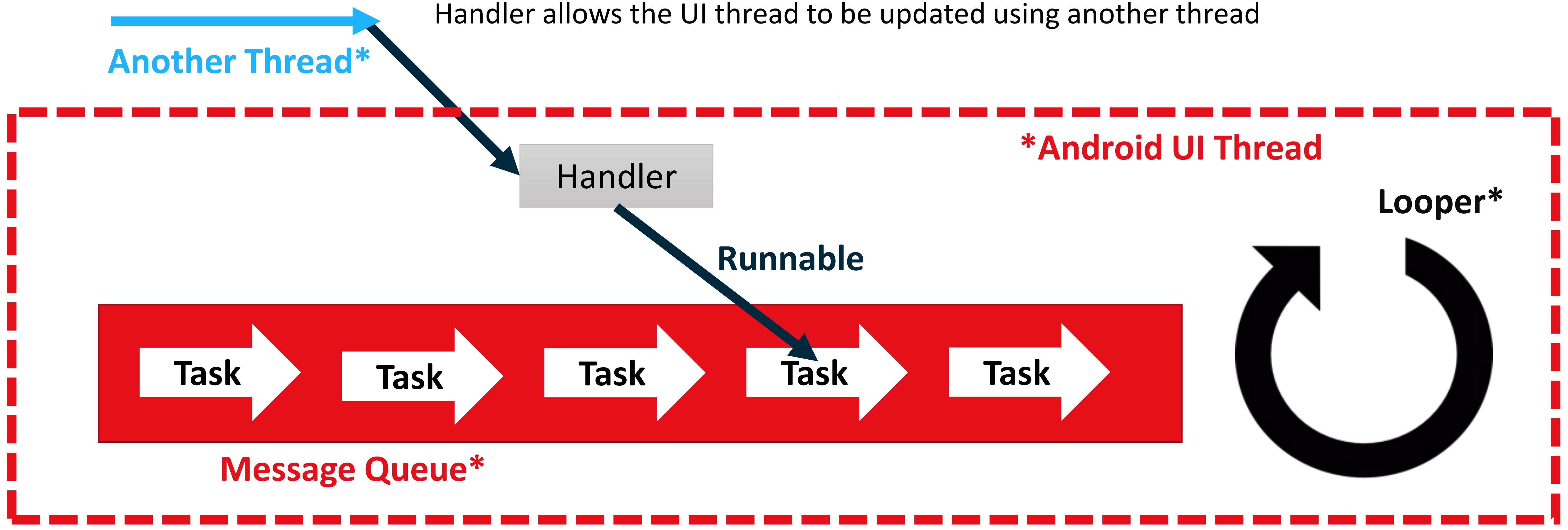
1. Receives and Reacts to broadcast Intents
2. No UI but can start an Activity
3. Extends the BroadcastReceiver Base Class

Content Provider

1. Makes application data available to other apps
2. Data stored in SQLite database
3. Extends the ContentProvider Base class

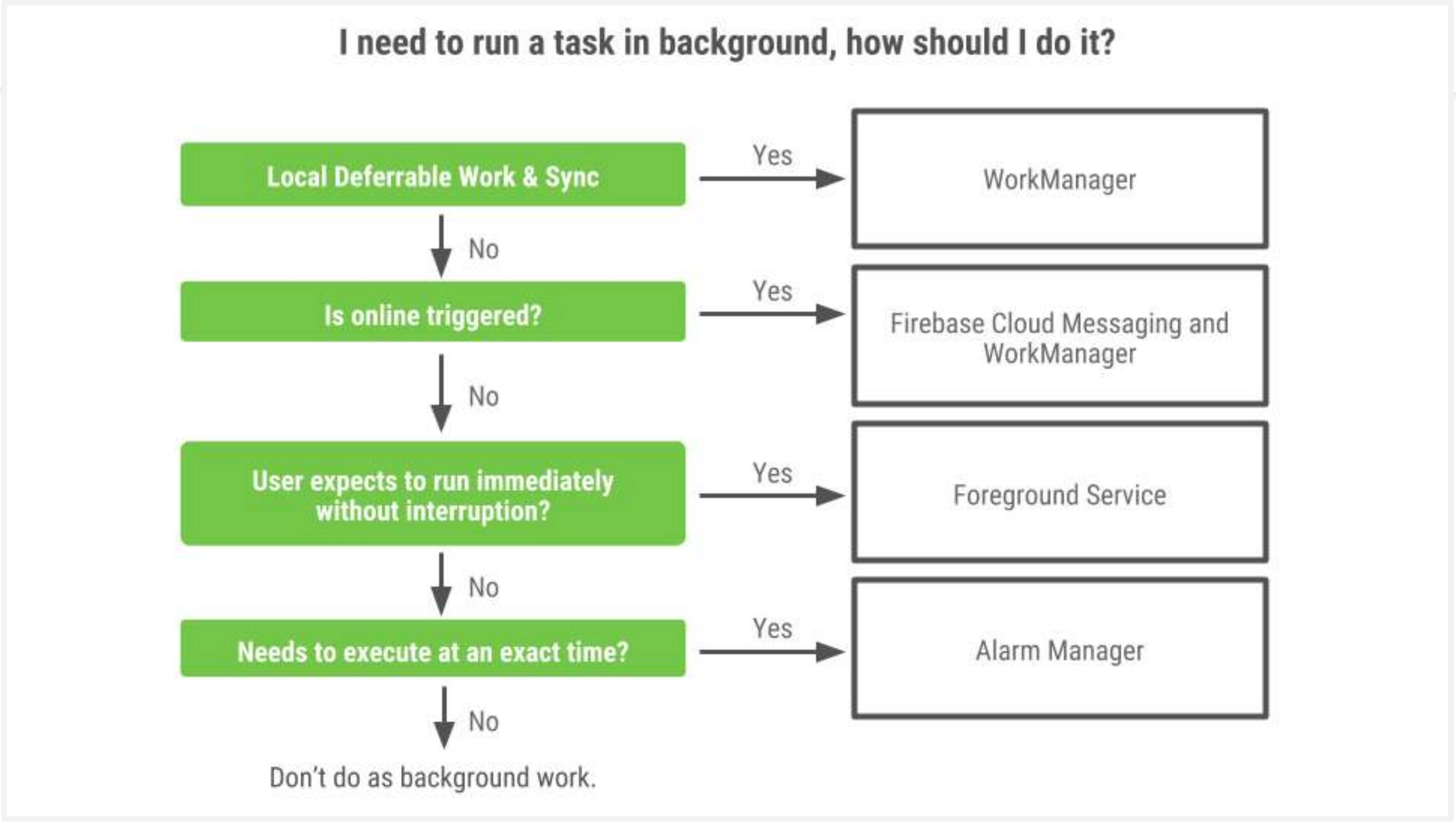
Recap: Main UI Thread and Handler

Handler allows the UI thread to be updated using another thread



https://developer.android.com/guide/components/processes-and-threads.html

Recap: Background execution



I need to run a task in background, how should I do it?
Local Deferrable Work & Sync
Yes
WorkManager
No
Is online triggered?
Yes
Firebase Cloud Messaging and WorkManager
No
User expects to run immediately without interruption?
Yes
Foreground Service
No
Needs to execute at an exact time?
Yes
Alarm Manager
No
Don't do as background work.

https://android-developers.googleblog.com/2018/10/modern-background-execution-in-android.html

Service: Introduction

- Can be restarted as soon as having sufficient resources.
- Likely be killed when system is short of resources.
- For: long-lived operations, doesn’t require a user interface function.
- Can be started, stopped, and controlled from other components: activities, broadcast receiver, other services
- Three types of services, foreground, background and bound.

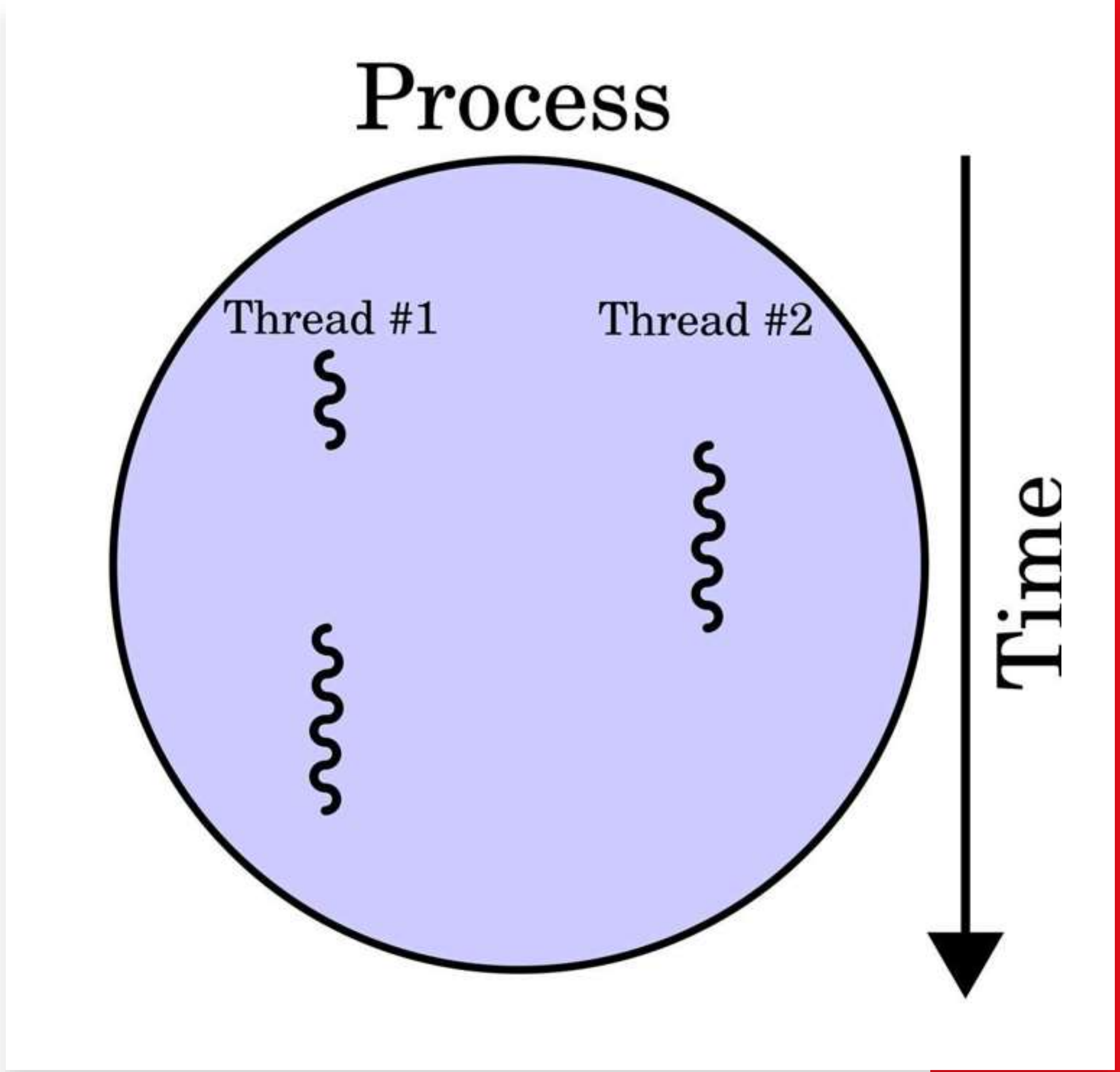
https://developer.android.com/reference/android/app/Service.html

Services & Threads?

- By default, a service will run within the same main thread as the application process from which it was launched (referred to as a local service).

- It is important, therefore, that any CPU intensive tasks be performed in a new thread within the service.

- Instructing a service to run within a separate process (and therefore known as a remote service) requires a **configuration change within the manifest file**.



Foreground Service

- A foreground service is a service that the user is actively aware of and isn't a candidate for the system to kill when low on memory.
  - ➢ For example, an audio app would use a foreground service to play an audio track.
- Foreground services must display a Notification in the status bar.
- Foreground services continue running even when the user isn't interacting with the app.

Background Service

- A background service performs an operation that isn't directly noticed by the user.
  - ➢ For example, if an app used a service to compact its storage, that would usually be a background service.

Note: If your app **targets API level 26** or higher, the system imposes restrictions on running background services when the app itself isn't in the foreground. In most situations, for example, you shouldn't access location information from the background. Instead, schedule tasks using **WorkManager**.

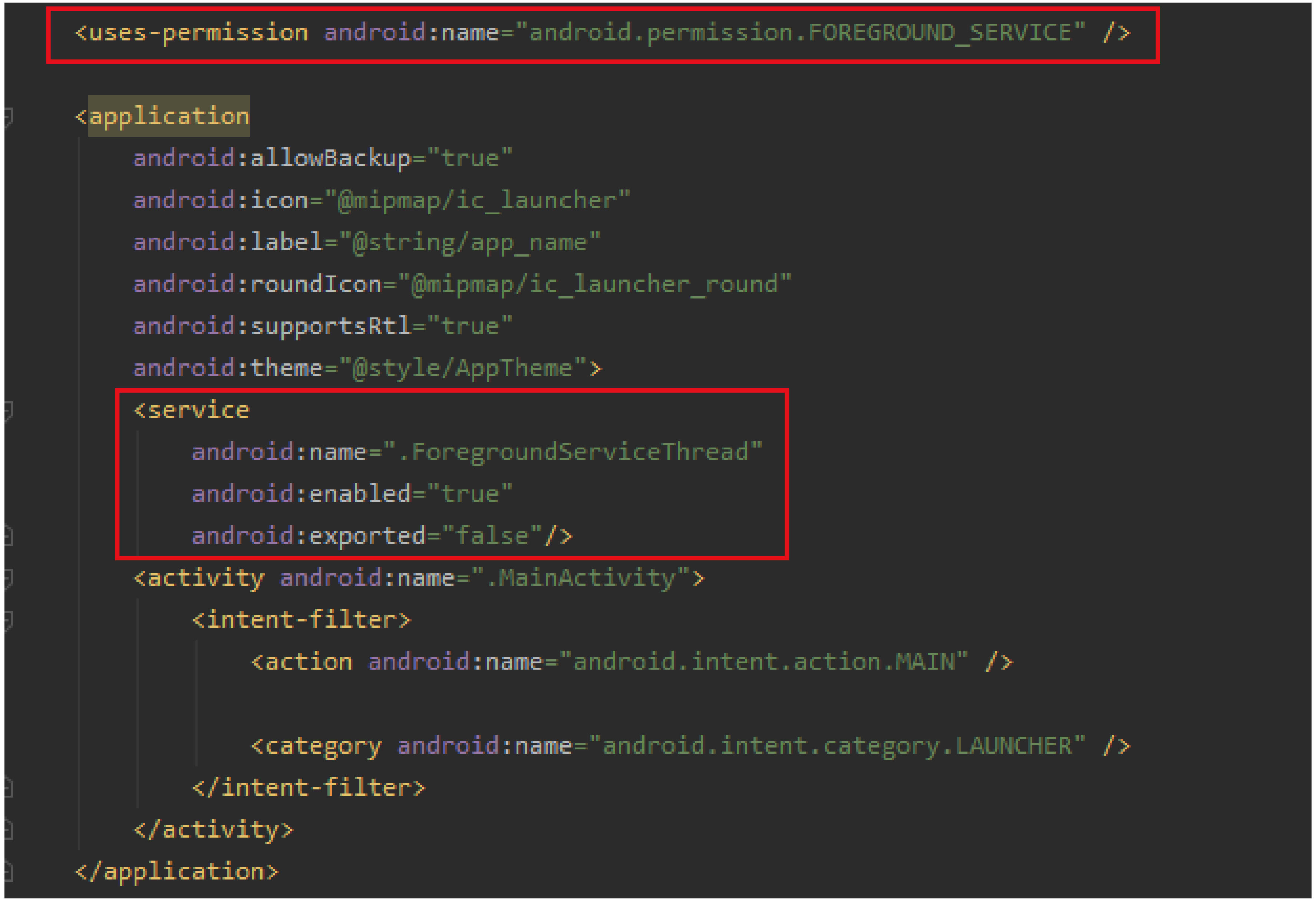
Bound Service

- A bound service is similar to a started service with the exception that a started service does not generally return results or permit interaction with the component that launched it. A bound service, on the other hand, allows the launching component to interact with, and receive result from the service.
- A service is bound when an application component binds to it by calling bindService().
- Through the implementation of **interprocess communication (IPC),** this interaction can also take place across process boundaries.

Creating a Service

To create a service, you must create a subclass of Service or use one of its existing subclasses. In your implementation, you must override some callback methods that handle key aspects of the service lifecycle and provide a mechanism that allows the components to bind to the service, if appropriate. These are the most important callback methods that you should override:

- **onStartCommand()**
  The system invokes this method by calling startService() when another component (such as an activity) requests that the service be started.
- **onBind()**
  The system invokes this method by calling bindService() when another component wants to bind with the service (such as to perform RPC).
- **onCreate()**
  The system invokes this method to perform one-time setup procedures when the service is initially created (before it calls either onStartCommand() or onBind()). If the service is already running, this method is not called.
- **onDestroy()**
  The system invokes this method when the service is no longer used and is being destroyed.



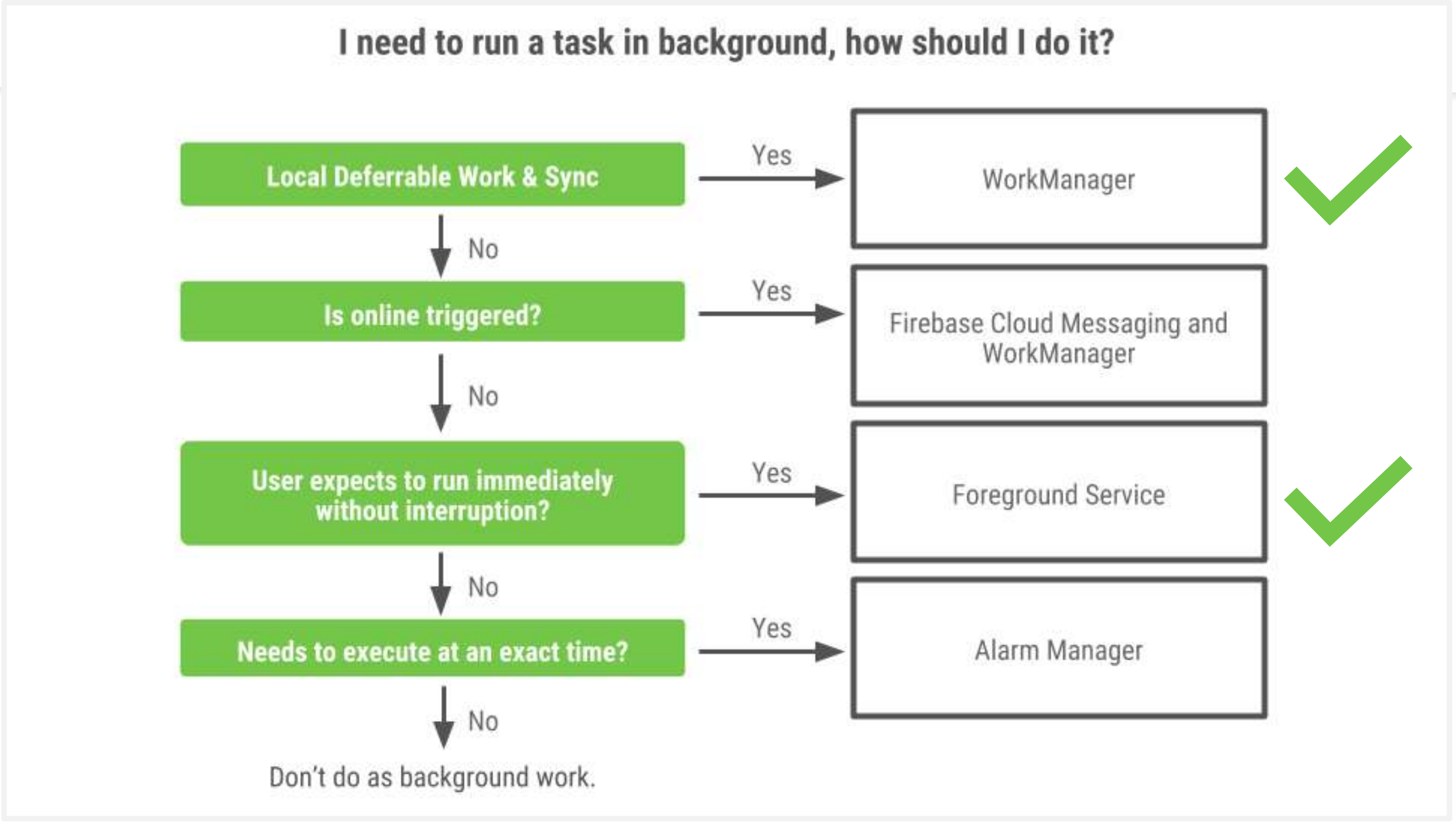
```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <service
        android:name=".ForegroundServiceThread"
        android:enabled="true"
        android:exported="false"/>
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

Editing the manifest

Creating Foreground Services

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture5_ForegroundServiceDemo

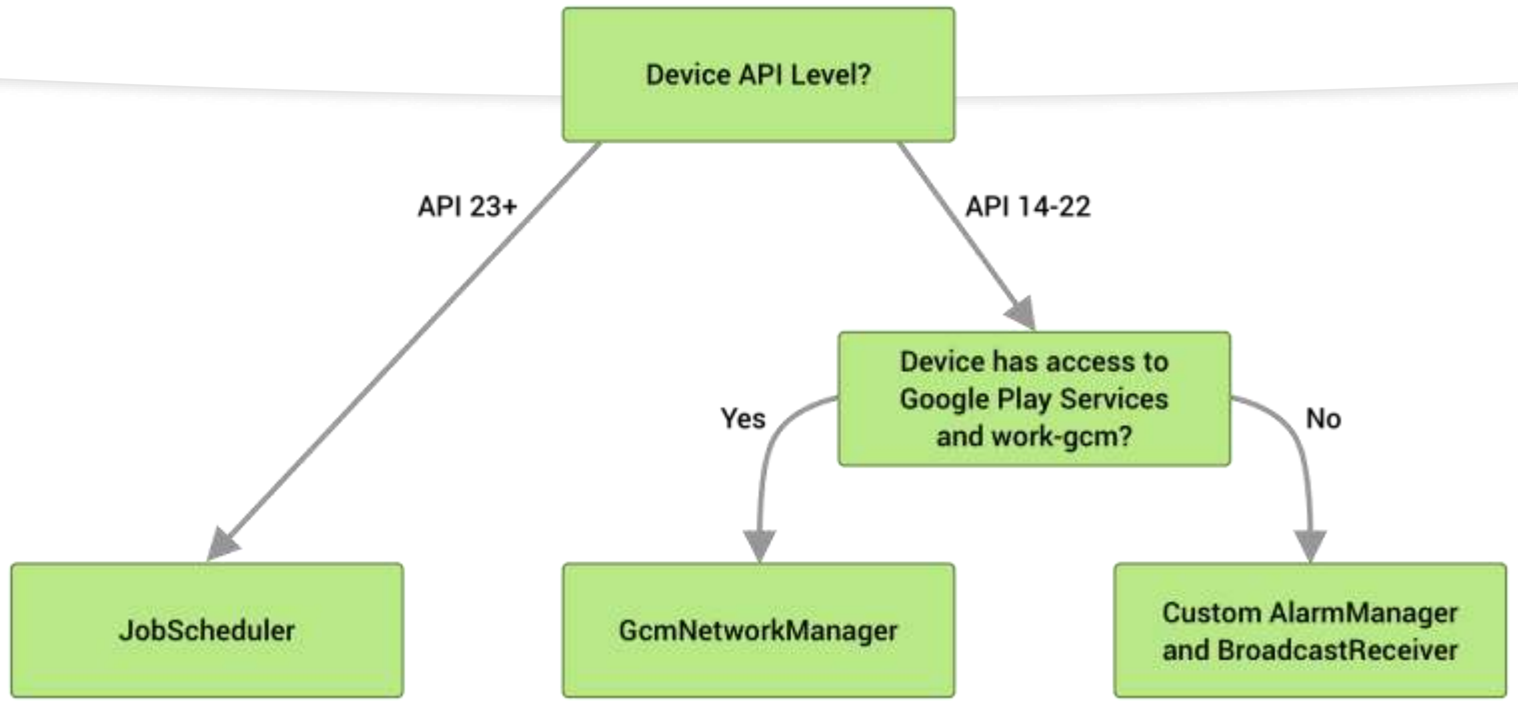
Recap: Background execution



I need to run a task in background, how should I do it?
Local Deferrable Work & Sync
Yes
WorkManager
✓
No
Is online triggered?
Yes
Firebase Cloud Messaging and WorkManager
No
User expects to run immediately without interruption?
Yes
Foreground Service
✓
No
Needs to execute at an exact time?
Yes
Alarm Manager
No
Don't do as background work.

https://android-developers.googleblog.com/2018/10/modern-background-execution-in-android.html

Schedule tasks with WorkManager


Device API Level?
API 23+
API 14-22
Device has access to Google Play Services and work-gcm?
Yes
No
JobScheduler
GcmNetworkManager
Custom AlarmManager and BroadcastReceiver

- WorkManager is an API that makes it easy to schedule deferrable, asynchronous tasks that are expected to run even if the app exits or the device restarts.
- WorkManager incorporates the features of its predecessors in a modern, consistent API that works back to API level 14 while also being conscious of battery life.

https://developer.android.com/topic/libraries/architecture/workmanager

When to use the WorkManager API?

WorkManager is intended for work that is **deferrable**—that is, not required to run immediately—and required to run reliably even if the app exits or the device restarts. For example:

- Sending logs or analytics to backend services
- Periodically syncing application data with a server

WorkManager is not intended for in-process background work that can safely be terminated if the app process goes away or for work that requires immediate execution.

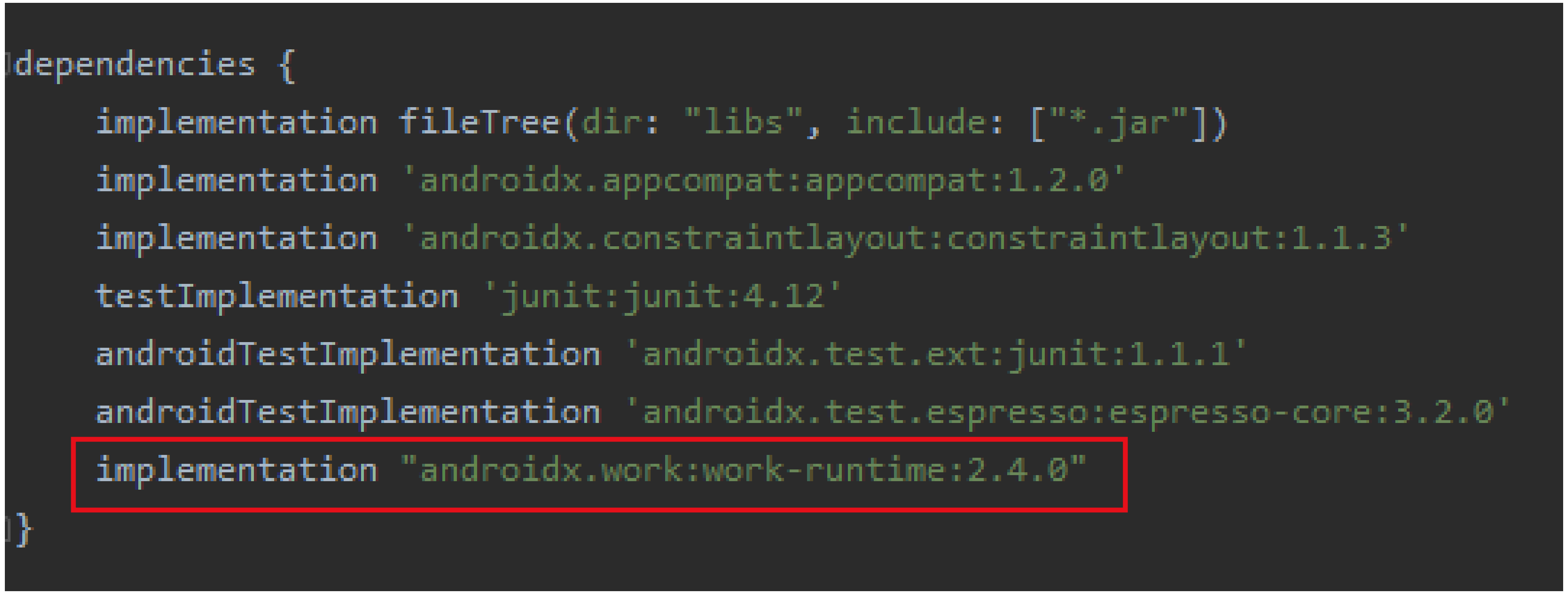
https://developer.android.com/topic/libraries/architecture/workmanager

https://youtu.be/pe_yqM16hPQ

https://codelabs.developers.google.com/codelabs/android-workmanager-java/#0

Getting started with WorkManager

To get started using WorkManager, first **import the library into your Android project.** Add the following dependencies to your app's build.gradle file:



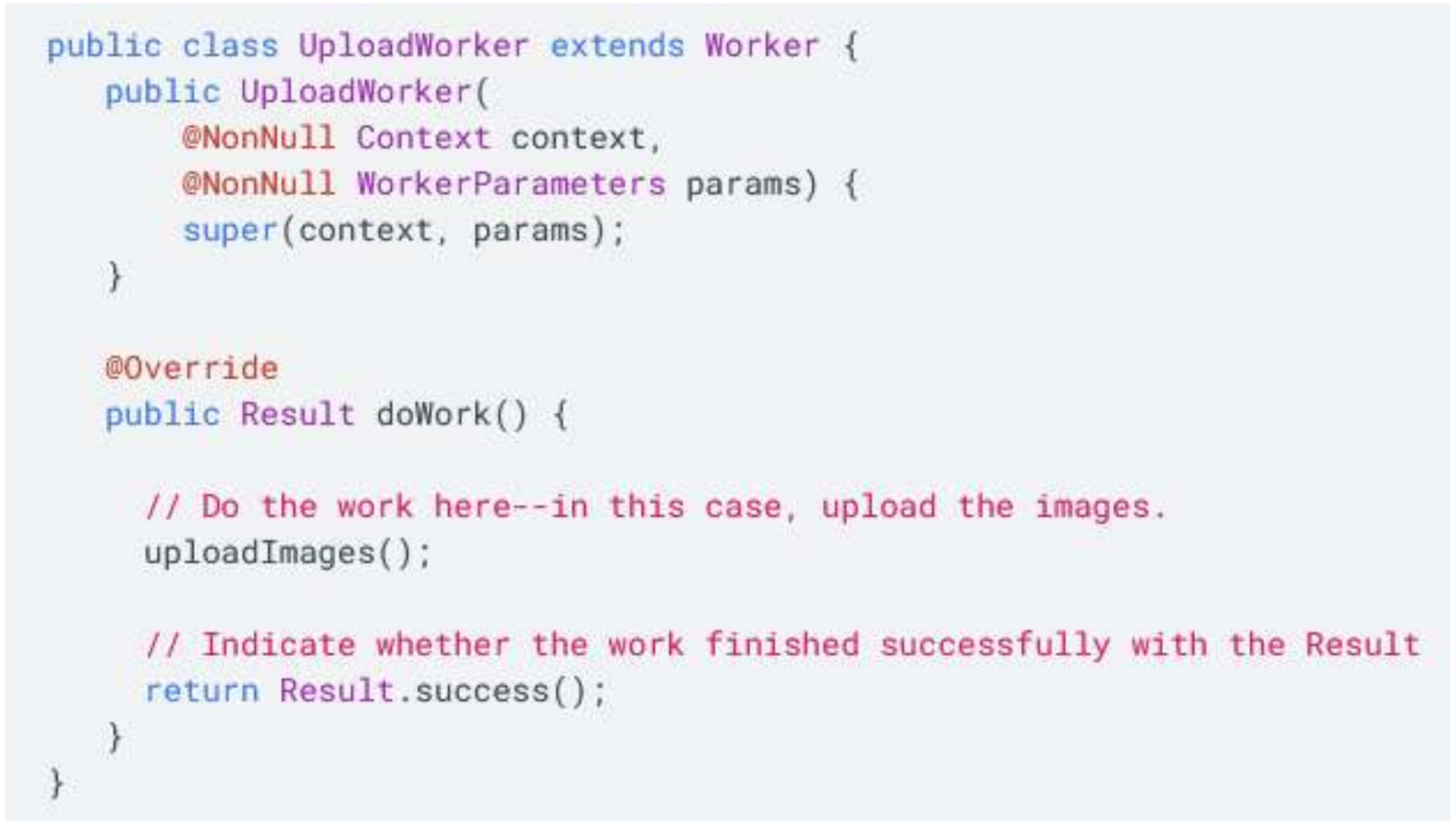
```
dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation "androidx.work:work-runtime:2.4.0"
}
```

Note: You can always find the latest version of WorkManager, on the WorkManager releases page.

https://developer.android.com/topic/libraries/architecture/workmanager

Getting started with WorkManager

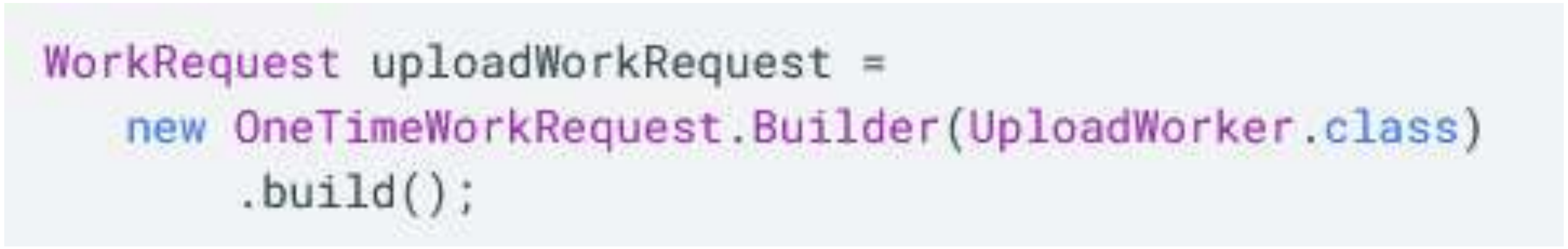
Make a Worker: This is where you put the code for the actual work you want to perform in the background. You'll extend this class and override the doWork() method.



```
public class UploadWorker extends Worker {
   public UploadWorker(
       @NonNull Context context,
       @NonNull WorkerParameters params) {
       super(context, params);
   }

   @Override
   public Result doWork() {

     // Do the work here--in this case, upload the images.
     uploadImages();

     // Indicate whether the work finished successfully with the Result
     return Result.success();
   }
}
```

Getting started with WorkManager

Make a WorkRequest: This represents a request to do some work. You'll pass in your Worker as part of creating your WorkRequest. When making the WorkRequest you can also specify things like Constraints on when the Worker should run.

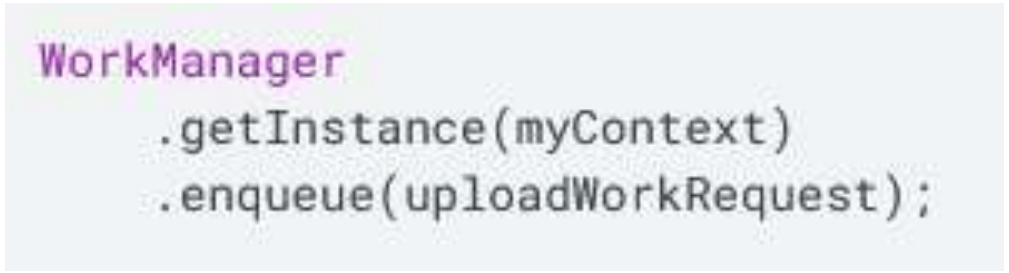


```
WorkRequest uploadWorkRequest =
   new OneTimeWorkRequest.Builder(UploadWorker.class)
        .build();
```

Getting started with WorkManager

Finally, you need to **submit your WorkRequest to WorkManager** using the enqueue() method.

The exact time that the worker is going to be executed depends on the constraints that are used in your WorkRequest and on system optimizations. WorkManager is designed to give the best behavior under these restrictions.

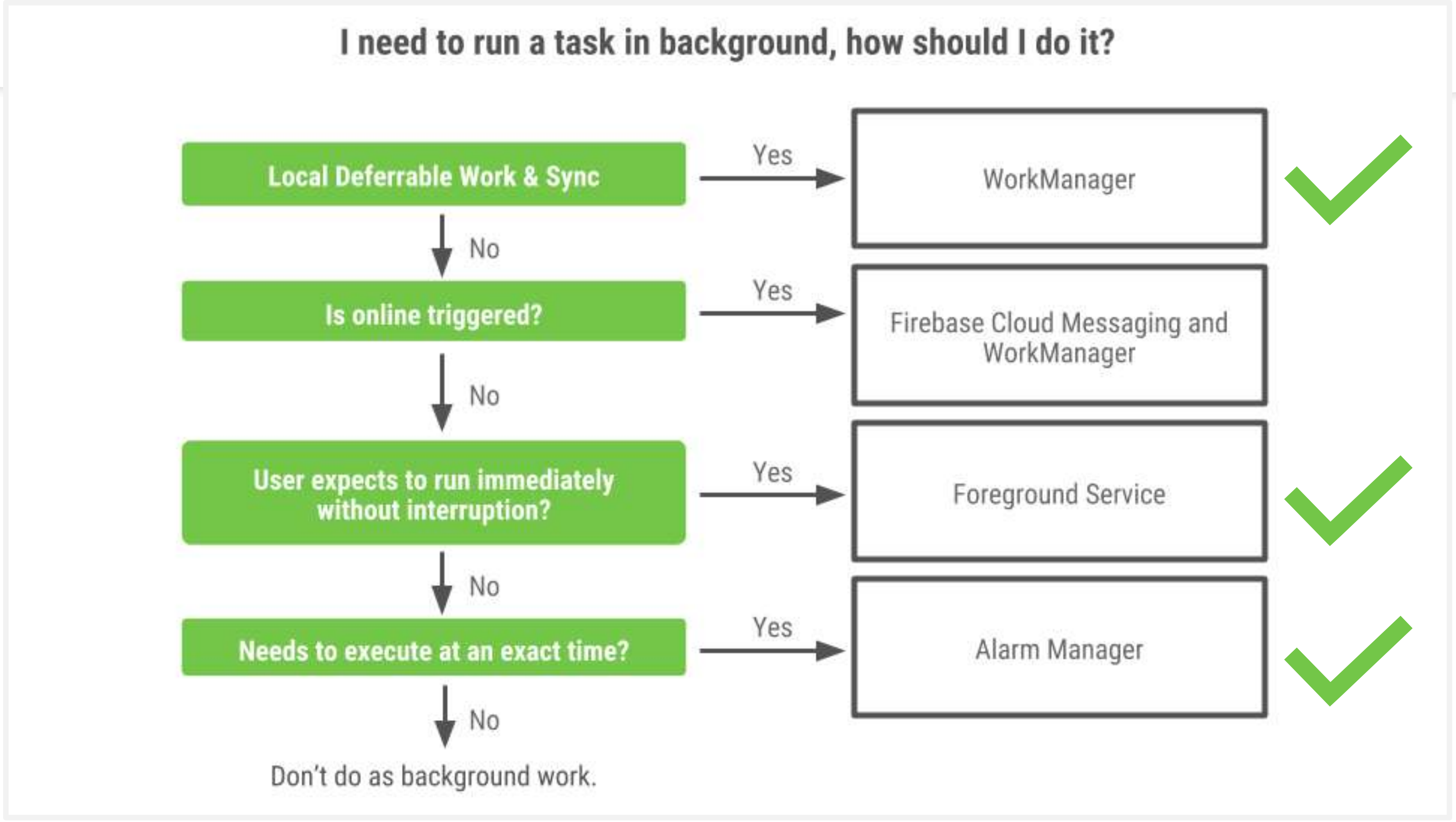


```
WorkManager
    .getInstance(myContext)
    .enqueue(uploadWorkRequest);
```

Using WorkManager

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture5_WorkManagerDemo

Recap: Background execution


I need to run a task in background, how should I do it?
Local Deferrable Work & Sync
Yes
WorkManager
No
Is online triggered?
Yes
Firebase Cloud Messaging and WorkManager
No
User expects to run immediately without interruption?
Yes
Foreground Service
No
Needs to execute at an exact time?
Yes
Alarm Manager
No
Don't do as background work.

https://android-developers.googleblog.com/2018/10/modern-background-execution-in-android.html

Alarm Manager

Alarms (based on the AlarmManager class) give you a way to perform time-based operations outside the lifetime of your application. For example, you could use an alarm to initiate a long-running operation, such as starting a service once a day to download a weather forecast.

Alarms have these characteristics:

- They let you fire Intents at set times and/or intervals.
- You can use them in conjunction with broadcast receivers to start services and perform other operations.
- They operate outside of your application, so you can use them to trigger events or actions even when your app is not running, and even if the device itself is asleep.
- They help you to minimize your app's resource requirements. You can schedule operations without relying on timers or continuously running background services.

https://developer.android.com/training/scheduling/alarms

Recap: Background execution

Use Case	**Examples**	**Solution**
Guaranteed execution of deferrable work	•Upload logs to your server •Encrypt/Decrypt content to upload/download	WorkManager
A task initiated in response to an external event	•Syncing new online content like email	FCM + WorkManager
Continue user-initiated work that needs to run immediately even if the user leaves the app	•Music player •Tracking activity •Transit navigation	Foreground Service
Trigger actions that involve user interactions, like notifications at an exact time.	•Alarm clock •Medicine reminder •Notification about a TV show that is about to start	AlarmManager

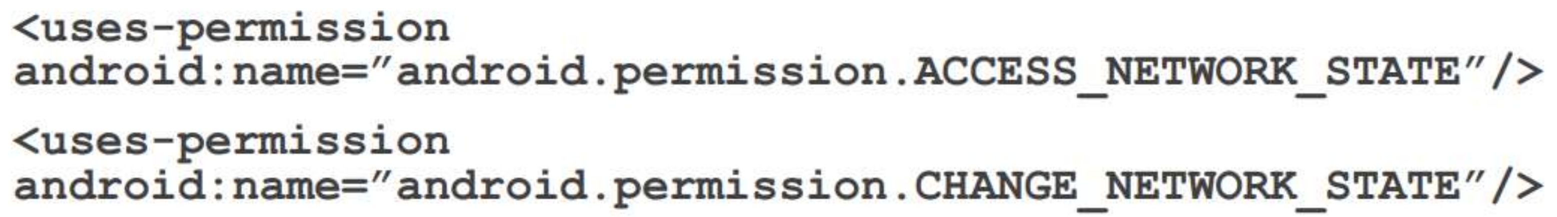
https://android-developers.googleblog.com/2018/10/modern-background-execution-in-android.html

WiFi & Bluetooth

- Internet Connectivity
  - Managing Networking
  - Managing Wi-Fi
- Bluetooth
  - Bluetooth Permissions
  - Access the Bluetooth radio (Bluetooth Adapter)
  - Discovering devices
  - Enabling discoverability
  - Connecting Devices
  - Managing a Connection

Internet Connectivity: Managing Networking

- ConnectivityManager
  - ➢ A Service that handles Android networking
  - ➢ Lets you monitor the connectivity state, set your preferred network connection, and manage connectivity failover.
- Permissions
  - ➢ needs read and write network state access



```
<uses-permission
android:name=”android.permission.ACCESS_NETWORK_STATE”/>
<uses-permission
android:name=”android.permission.CHANGE_NETWORK_STATE”/>
```

- To access the Connectivity Manager use getSystemService, passing in Context.CONNECTIVITY_SERVICE as the service name

https://developer.android.com/reference/android/net/ConnectivityManager.html

Internet Connectivity: Managing Wi-Fi

WifiManager represents the Android Wi-Fi Connectivity Service and can be used to configure Wi-Fi network connections, manage the current Wi-Fi connection, scan for access points, and monitor changes in Wi-Fi connectivity.

- Permissions
  - ➢ To use the Wi-Fi Manager, need uses-permissions for accessing and changing the Wi-Fi state :

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
```

- To access the Wi-Fi Manager use the getSystemService method, passing in the Context.WIFI_SERVICE constant

Use the Wi-Fi Manager to

- request the current Wi-Fi state using the getWifiState or isWifiEnabled methods

https://developer.android.com/reference/android/net/wifi/WifiManager

Using WifiManager

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture5_WifiDemo

Steps to do a Web request

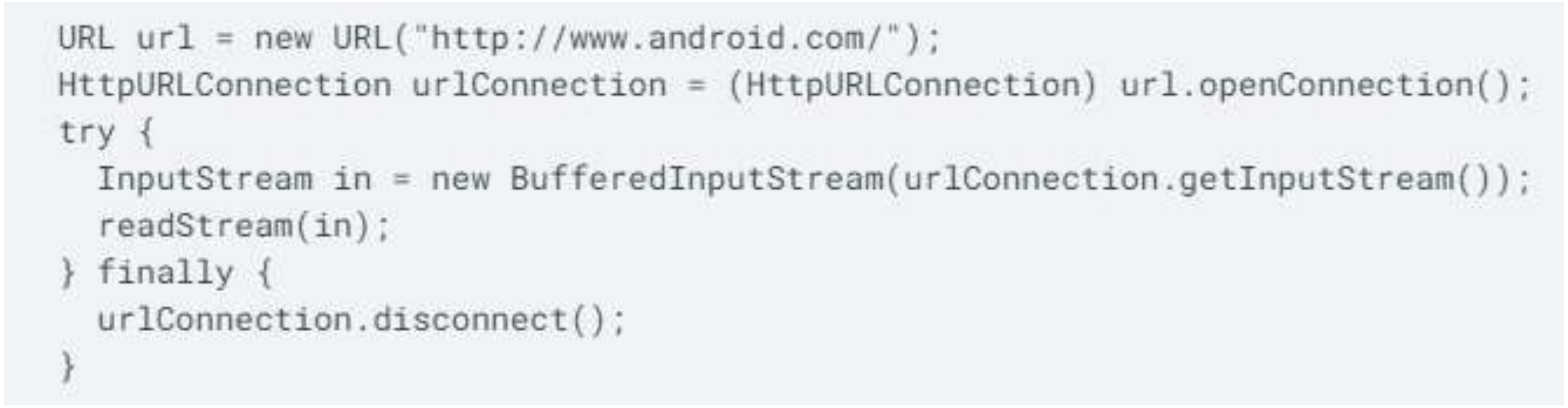
An URLConnection for HTTP (RFC 2616) used to send and receive data over the web. Data may be of any type and length. This class may be used to send and receive streaming data whose length is not known in advanceUses of this class follow a pattern:

1. Obtain a new HttpURLConnection by calling URL#openConnection() and casting the result to HttpURLConnection.
2. Prepare the request. The primary property of a request is its URI. Request headers may also include metadata such as credentials, preferred content types, and session cookies.
3. Optionally upload a request body. Instances must be configured with setDoOutput(true) if they include a request body. Transmit data by writing to the stream returned by URLConnection.getOutputStream().
4. Read the response. Response headers typically include metadata such as the response body's content type and length, modified dates and session cookies. The response body may be read from the stream returned by URLConnection.getInputStream(). If the response has no body, that method returns an empty stream.
5. Disconnect. Once the response body has been read, the HttpURLConnection should be closed by calling disconnect(). Disconnecting releases the resources held by a connection so they may be closed or reused.

https://developer.android.com/reference/java/net/HttpURLConnection.html

Simple Example

Download a webpage;



```
URL url = new URL("http://www.android.com/");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
try {
  InputStream in = new BufferedInputStream(urlConnection.getInputStream());
  readStream(in);
} finally {
  urlConnection.disconnect();
}
```

https://developer.android.com/reference/java/net/HttpURLConnection.html

In AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="net.learn2develop.Networking"
        android:versionCode="1"
        android:versionName="1.0" >
        <uses-sdk android:minSdkVersion="13" />
        <uses-permission android:name="android.permission.INTERNET"/>
        <application
                android:icon="@drawable/ic_launcher"
                android:label="@string/app_name" >
                <activity
                        android:label="@string/app_name"
                        android:name=".NetworkingActivity" >
                        <intent-filter >
                                <action android:name="android.intent.action.MAIN" />
                                <category android:name="android.intent.category.LAUNCHER" />
                        </intent-filter>
                </activity>
        </application>
</manifest>
```

Download Text

```
private String DownloadText(String URL)
 {
    int BUFFER_SIZE = 2000;
    InputStream in = null;
    try {
       in = OpenHttpConnection(URL);
    } catch (IOException e) {
    Log.d("NetworkingActivity", e.getLocalizedMessage());
       return "";
    }
    InputStreamReader isr = new InputStreamReader(in);
    int charRead;
    String str = "";
    char[] inputBuffer = new char[BUFFER_SIZE];
    try {
       while ((charRead = isr.read(inputBuffer))>0) {
          //---convert the chars to a String---
          String readString =
             String.copyValueOf(inputBuffer, 0, charRead);
          str += readString;
          inputBuffer = new char[BUFFER_SIZE];
       }
       in.close();
    } catch (IOException e) {
    Log.d("NetworkingActivity", e.getLocalizedMessage());
       return "";
    }
    return str;
 }
```

Download Image

```
private Bitmap DownloadImage(String URL)
{
   Bitmap bitmap = null;
   InputStream in = null;
   try {
      in = OpenHttpConnection(URL);
      bitmap = BitmapFactory.decodeStream(in);
      in.close();
   } catch (IOException e1) {
      Log.d("NetworkingActivity", e1.getLocalizedMessage());
   }
   return bitmap;
}
```

```
<ImageView
   android:id="@+id/img"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_gravity="center" />
```

```
Bitmap bitmap = DownloadImage("http://.../mypict.jpg");
ImageView img = (ImageView) findViewById(R.id.img);
img.setImageBitmap(bitmap);
```

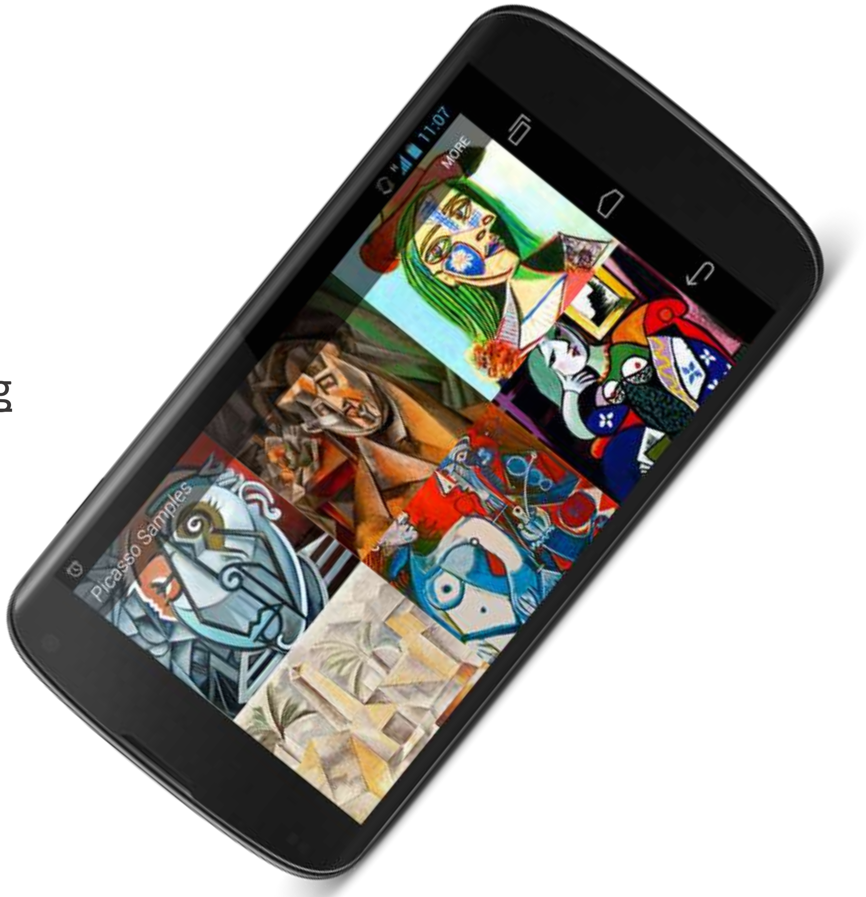
Download Libraries

https://github.com/square/picasso

https://github.com/bumptech/glide

https://github.com/google/volley

https://github.com/amitshekhariitbhu/Fast-Android-Networking



Bluetooth

- Using the Bluetooth APIs, an Android application can
  - ➢ Scan for other Bluetooth devices
  - ➢ Query the local Bluetooth adapter for paired Bluetooth devices
  - ➢ Establish RFCOMM channels
  - ➢ Connect to other devices through service discovery
  - ➢ Transfer data to and from other devices
  - ➢ Manage multiple connections

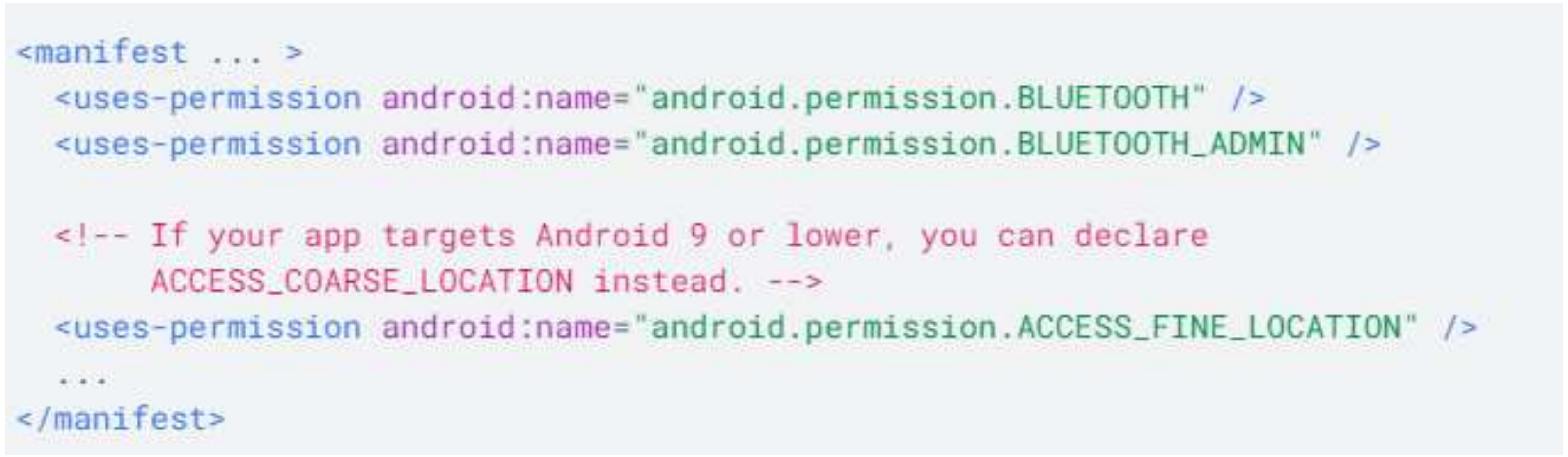
https://developer.android.com/guide/topics/connectivity/bluetooth

Bluetooth Permissions

To use Bluetooth features, need to declare at least one of two permissions: BLUETOOTH and BLUETOOTH_ADMIN.

- BLUETOOTH permission: to perform any Bluetooth communication, such as requesting a connection, accepting a connection, and transferring data.
- BLUETOOTH_ADMIN permission: to initiate device discovery or manipulate Bluetooth settings

*Note: If you use BLUETOOTH_ADMIN permission, then must also have the BLUETOOTH permission.

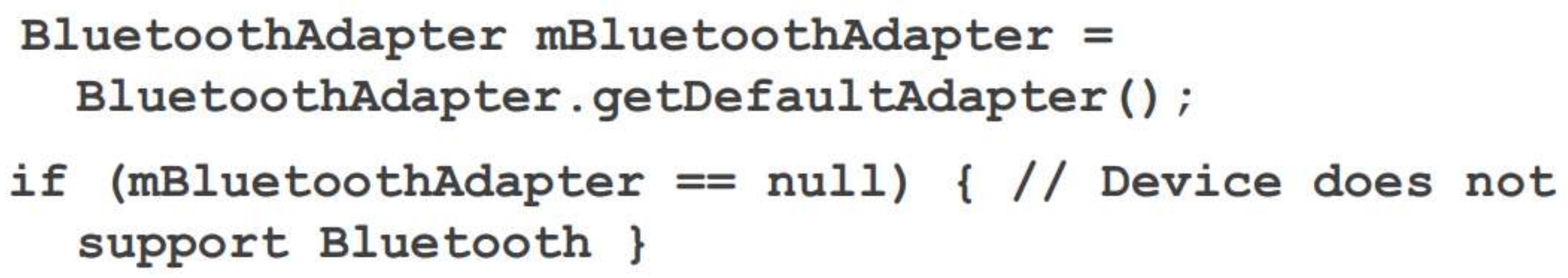


```
<manifest ... >
  <uses-permission android:name="android.permission.BLUETOOTH" />
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

  <!-- If your app targets Android 9 or lower, you can declare
       ACCESS_COARSE_LOCATION instead. -->
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  ...
</manifest>
```

https://developer.android.com/guide/topics/connectivity/bluetooth

Access the Bluetooth radio

- BluetoothAdapter represents the device's own Bluetooth adapter (the Bluetooth radio).
- To verify that Bluetooth is supported on the device call the static getDefaultAdapter() method. If getDefaultAdapter() returns null, then the device does not support Bluetooth.

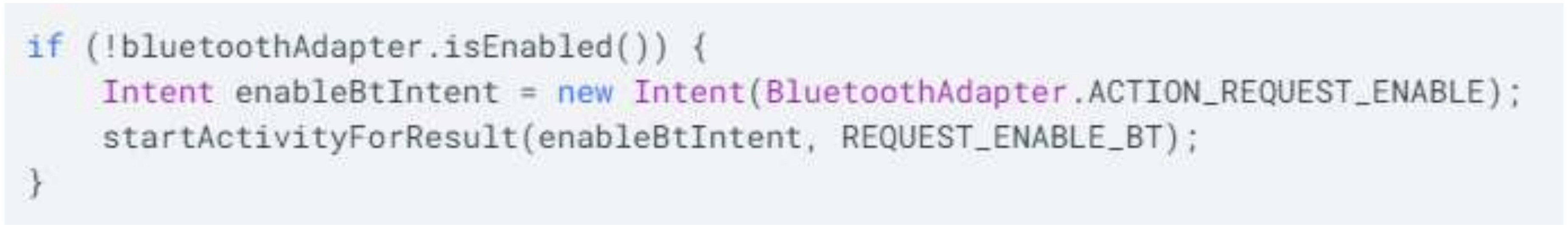


```
BluetoothAdapter mBluetoothAdapter =
  BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) { // Device does not
  support Bluetooth }
```

https://developer.android.com/guide/topics/connectivity/bluetooth

Enable Bluetooth

- Call **isEnabled()** to check if Bluetooth is currently enabled.
- To request that Bluetooth be enabled, call **startActivityForResult()** with the ACTION_REQUEST_ENABLE action Intent



```
if (!bluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

- The **ENABLE_BLUETOOTH** constant passed to **startActivityForResult()** is a locally defined integer (which must be greater than 0), that the system passes back to you in your **onActivityResult()** implementation as the requestCode parameter.
- If enabling Bluetooth succeeds, your activity receives the RESULT_OK result code in the **onActivityResult()** callback.

*Note: Enabling discoverability will automatically enable Bluetooth

https://developer.android.com/guide/topics/connectivity/bluetooth

Bluetooth: Discovering devices

Device discovery:

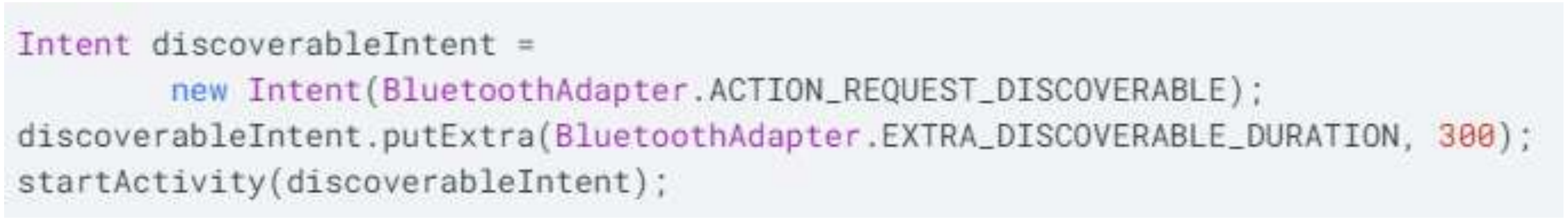
- A scanning procedure that searches the local area for Bluetooth enabled devices and then requesting some information about each one.
- A Bluetooth device will respond to a discovery request only if it is currently set to be discoverable.
- If a device is discoverable, it will respond to the discovery request by sharing some information (device name, class, MAC address).
- Using this information, the device performing discovery can then choose to initiate a connection to the discovered device.
- Call **startDiscovery()** on the **BluetoothAdapter** and register a **BroadcastReceiver** for the ACTION_FOUND Intent to receive information about each device discovered.

Using Bluetooth

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture5_BluetoothDemo

Bluetooth: Enabling discoverability

- To make the local device discoverable to other devices, call **startActivityForResult(Intent, int)** with the **ACTION_REQUEST_DISCOVERABLE** action Intent.
- This will issue a request to enable discoverable mode through the system settings (without stopping your application).
- By default, the device becomes discoverable for 120 seconds. You can define a different duration by adding the **EXTRA_DISCOVERABLE_DURATION** Intent extra.
- The maximum duration an app can set is 3600 seconds, and a value of 0 means the device is always discoverable.
- If the device successfully enters discoverable mode, the activity will receive a call to the **onActivityResult()** callback, with the result code equal to the duration that the device is discoverable.

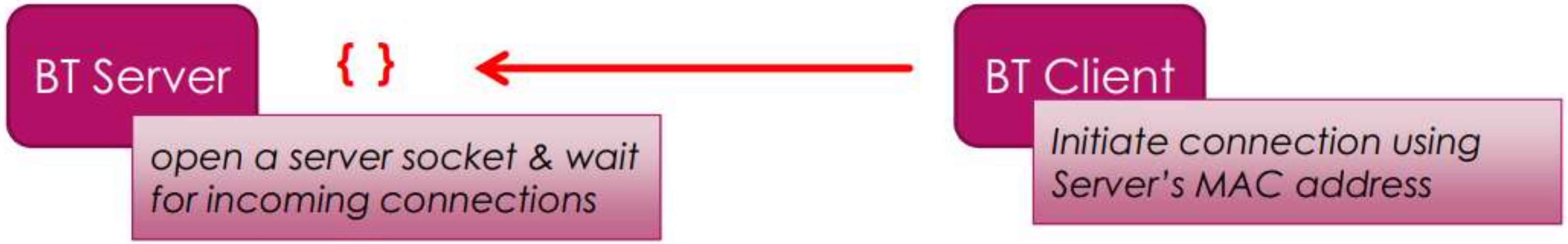


```
Intent discoverableIntent =
        new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivity(discoverableIntent);
```

https://developer.android.com/guide/topics/connectivity/bluetooth

Bluetooth: Connecting Devices

- In order to create a connection between your application on two devices, you need both the server-side and client-side mechanisms
- one device must open a server socket and the other one must initiate the connection (using the server device's MAC address to initiate a connection).
- The server and client are connected when they each have a connected **BluetoothSocket** on the same RFCOMM channel.
- At this point, each device can obtain input and output streams and data transfer can begin.
- The server device and the client device each obtain the required **BluetoothSocket** in different ways. The server will receive it when an incoming connection is accepted. The client will receive it when it opens an RFCOMM channel to the server.


BT Server
{ }
open a server socket & wait for incoming connections
BT Client
Initiate connection using Server's MAC address

Bluetooth: Managing a Connection

- When you have successfully connected two (or more) devices, each one will have a connected **BluetoothSocket**.
- To share data between connected devices
  - ➢ Get the **InputStream** and **OutputStream** that handle transmissions through the socket, via **getInputStream()** and **getOutputStream()**
  - ➢ Read and write data to the streams with read(byte[]) and write(byte[]).