CS683 Module 5

Asynchronous Processing
Broadcast Receiver, Process, Threads, Services

Questions?

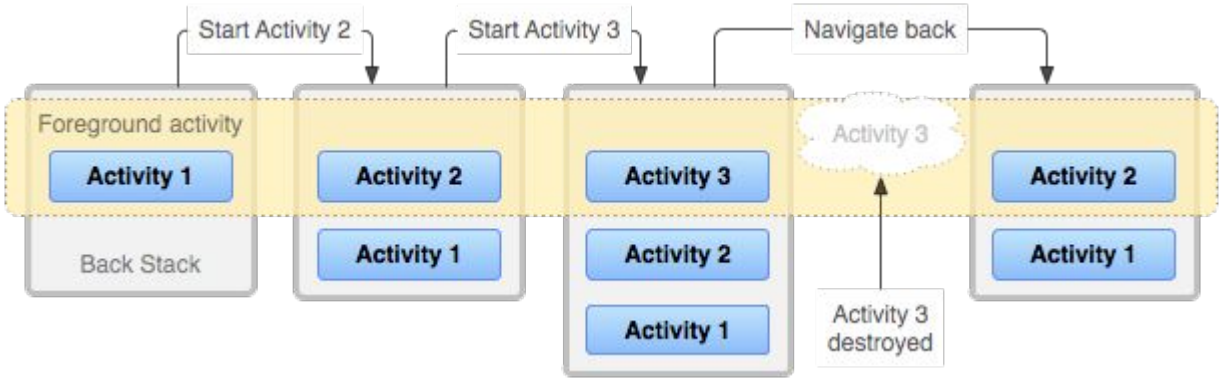
- How do different components in an application communicate to each other?
- How to move from one activity to another activity?
- How do different applications communicate to each other?
- How to protect the communication?
- What is an intent? What intents does the Android system support?
- What is a broadcast receiver? Why need it? How to use it?

Application Components

- Applications installed on the device include pre-installed apps and user-installed apps.
- Each application usually has multiple components to perform multiple functionalities, such as activities, content providers, services, broadcast receivers.
  - Email: activities to view folders, view list of messages, view message, compose, ...
  - Calendar: activities to view day, week, month, agenda, edit events, edit preferences, ...
  - Camera: activities for running camera, viewing list of pictures, viewing picture, cropping, running camcorder, viewing movies, ..
- As activities are entry points for user interactions, user move from one activity to another activity across the apps.

Activity Manager and Activity Stack

- A task is a collection of activities that users interact with when performing a certain job.
- The activity stack is maintained to keep the history of activities visited, when the user move from one to another activity.
  - Newly started activity is on the top
  - Pressing BACK displays the previous activity
- A task may involves multiple activities across different applications



https://developer.android.com/guide/components/activities/tasks-and-back-stack.html

Intent

- What is an intent?
- Why need intents?
- How does intents work?
- What types of intent?
- How to use intents?

Intent

- An Intent is a messaging object you can use to request an action from another app component.
  - The app components that can process an intent are: activity, service and broadcast receiver. Content providers don’t process intents directly.
  - Start an activity: startActivity(intent)
  - Start a service: startService(intent), bindService(intent)
  - Send a broadcast intent: sendBroadcast(intent)
  - Support late binding between components in same or different applications
- Traditionally messaging in OOP when obj1 sends an message to obj2, then in obj1 will call obj2.method(parameters).
  - This is more static. Need specify obj2 and method in the program.
- Intent message encapsulates receiving components, action(method) and data(parameters) together.

Building an Intent

- Component name: specify who should receive the intent
  - If specified, it is an explicit intent
  - If not specified, it is an implicit intent. Then system decide who should receive the intent based on other information.
- Action: specify the general action that the receiver should perform.
  - Android system defines some common actions
  - The application can also specify its own actions
- Data: the URI (a Uri object) that references the data to be acted on and/or the MIME type of that data.
  - The type of data supplied is generally dictated by the intent's action. For example, if the action is ACTION_EDIT, the data should contain the URI of the document to edit.

Intent Constructor

```
Intent()
```

Create an empty intent.

```
Intent(Intent o)
```

Copy constructor.

```
Intent(String action)
```

Create an intent with a given action.

```
Intent(String action, Uri uri)
```

Create an intent with a given action and for a given data url.

```
Intent(Context packageContext, Class<?> cls)
```

Create an intent for a specific component.

```
Intent(String action, Uri uri, Context packageContext, Class<?> cls)
```

Create an intent for a specific component with a specified action and data.

Access Intent Data

- Directly access the fields of an intent (Kotlin)
  - Intent.action, intent.flag, intent.component, intent.data
- A number get and set and putExtra() methods are used to access/modify the intent data. (Java)
  - setAction(), setClass(), setComponent(), setData(), setFlags(), putExtra()...
  - getAction(), getComponent(), getData(), getFlags(), getExtras() …
- https://developer.android.com/reference/android/content/Intent

Intent: Action

- It is simply a string
- Actions defined by the System
  - Activity actions: ACTION_MAIN, ACTION_VIEW, ACTION_EDIT, ACTION_CALL, ACTION_SEND, ACTION_SYNC, …
  - Broadcast actions: ACTION_SCREEN_ON, ACTION_BATTERY_LOW, ACTION_HEADSET_PLUG ...
  - https://developer.android.com/reference/android/content/Intent.html
  - There are activity_actions.txt, service_actions.txt and broadcast_actions.txt stored under sdk/platforms/<android version>/data/.
- Action strings can be defined by the application
  - static final String ACTION_MONITOR = "com.example.action.MONITOR"
- Action strings determine how rest of intent structured—particularly data and extras fields

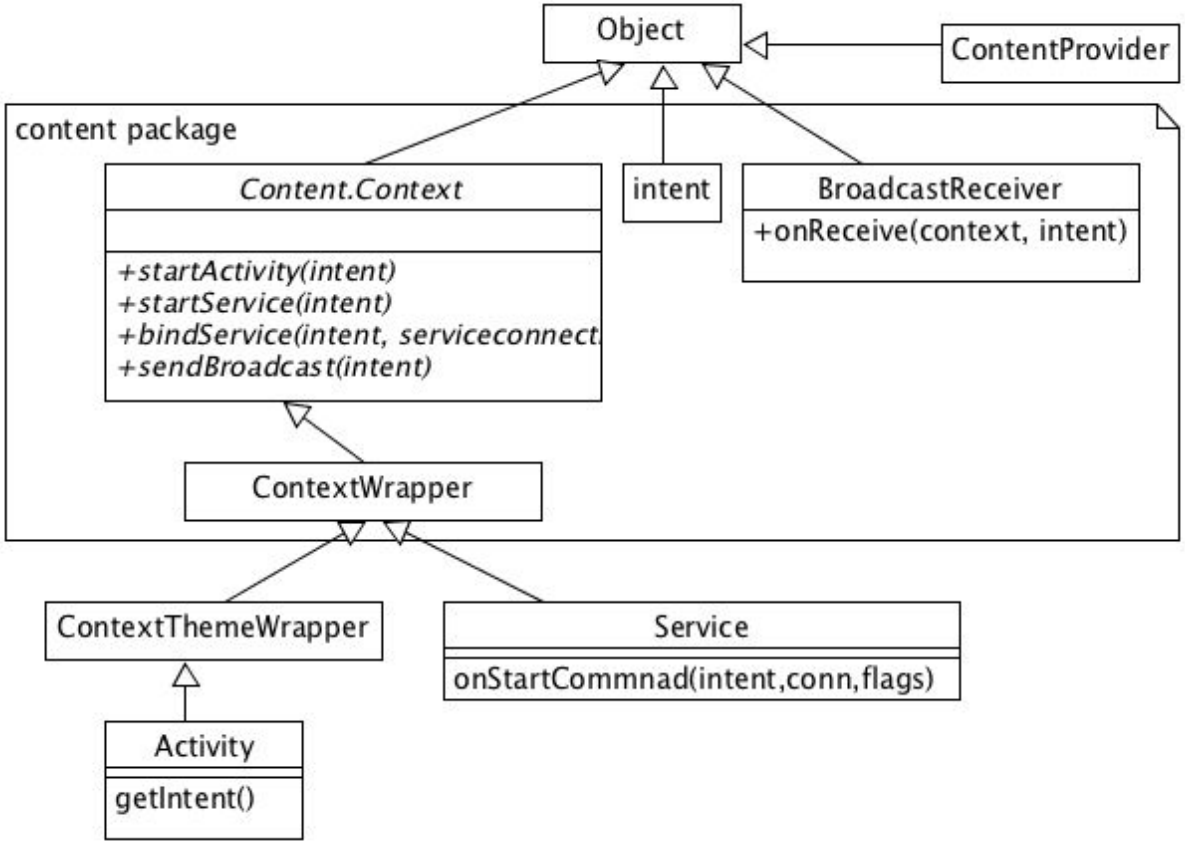
Intent: Data

- The URI (a Uri object) that references the data to be acted on and/or the MIME type of that data. The type of data supplied is generally dictated by the intent's action. For example, if the action is ACTION_EDIT, the data should contain the URI of the document to edit.
  - URI: Universal Resource Indicator. A string for identifying a resource; local, Internet.
  - Mime Type (Multipurpose Internet Mail Extensions): Consists of major type / minor type, e.g., text/plain, image/jpeg, audio/mp3, video/quicktime
- Actions are paired with kinds of data and it should be matched with data type
  - action ACTION_EDIT / data field contains URI of doc. to be displayed for editing
  - action ACTION_CALL / data field a tel: URI w. number to call
  - action ACTION_VIEW / data field http: URI, receiving activity called on to download & display data at URI

Intent: Extra

- Key-value pairs that carry additional information required to accomplish the requested action. Just as some actions use particular kinds of data URIs, some actions also use particular extras.
- Methods:
  - various putExtra() methods, each accepting two parameters: the key name and the value. You can also create a Bundle object with all the extra data, then insert the Bundle in the Intent with putExtras().
  - OBJ OBJ OBJ various getExtra() methods to get the value based on the key name, or getExtras() to the bundle.

Component Classes relationship



Send/Receive Intents

- Send Intents: (from activity or service or Broadcast Receiver)
  - startActivity(intent)
  - startService(intent), bindService(intent)
  - sendBroadcast(intent)
- Receive Intents
  - Activity: getIntent() (.intent)
  - Service: onStartCommand(intent)
  - Broadcast receiver: onReceive(context, intent)
- The systems sends the intents to a specific component(s)
  - specified by the explicit intent,
  - or based on the action/data/other info specified by the implicit intent, as well as the intent filters, and user's decision
  - Or the broadcast receivers registered to receive the broadcast intents

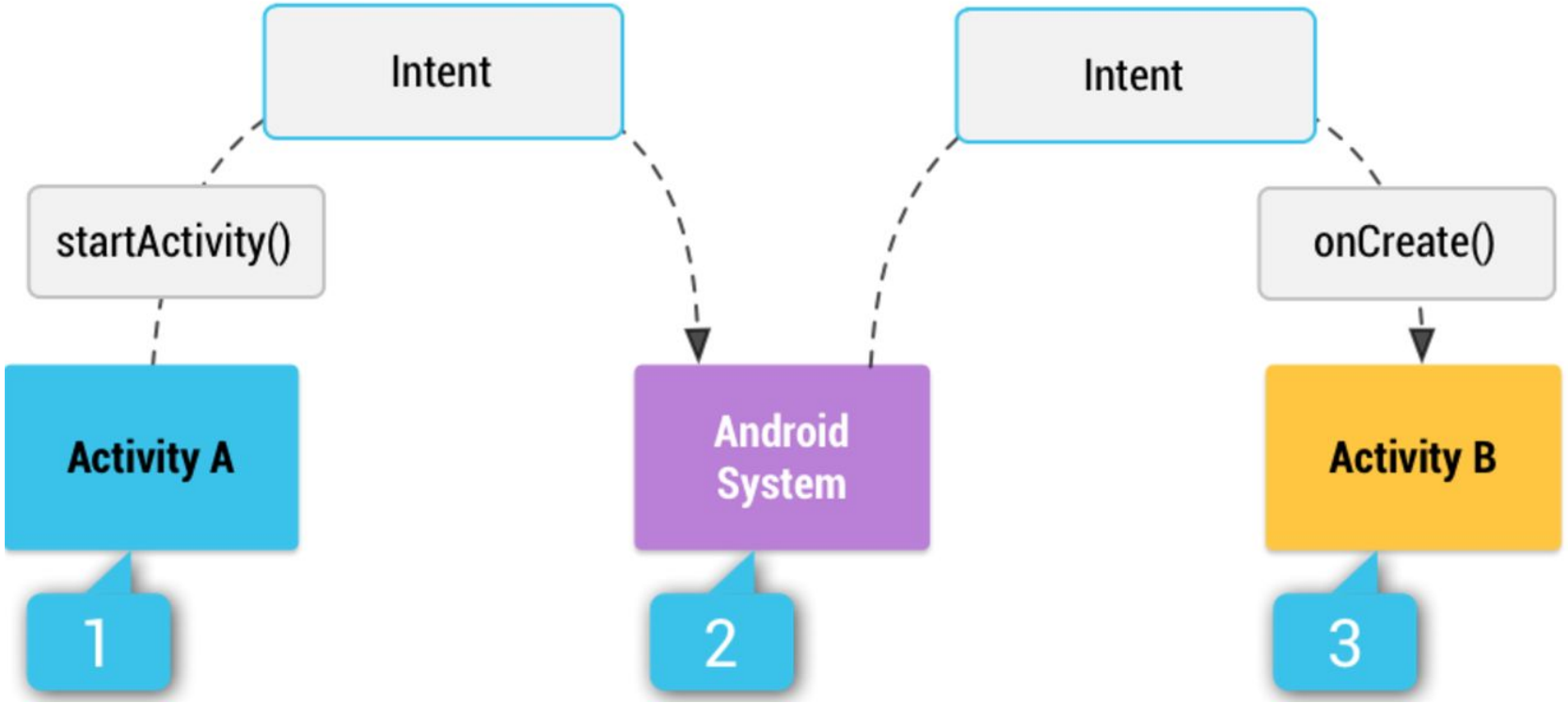
Types of Intents

- Explicit Intents
  - Specify the receiving component in the intent
- Implicit Intents
  - No receiving component is specified.
  - The system decides who will receive the intent based on other information in the intent such as action, category, as well as the intent filters of all components in the system.
  - If there are multiple candidates, the user can choose one.
  - Only one component will finally receive and act on the intent.
- Android system itself and the apps that come with it employ Intents to send out system-originated broadcasts and to activate system-defined components.

Intent Filter

- The core components of an application, such as its activities, services, and broadcast receivers, are activated by intents
- When using an implicit intent, the Android system locates an appropriate component that can respond to the intent, launches a new instance of the component if one is needed, and passes it the Intent object
- The components advertise the types of intents that they would like to receive/respond to through intent filters.
- If a component specifies an intent filter, it is exposed to other applications.
- Each intent filter defines at least one action string.

Implicit Intents



Start an Activity

- startActivity(intent): start a new instance of an Activity. The Intent describes the activity to start and carries any necessary data.
- startActivityForResult(intent,requestCode): to receive the result as a separate Intent object in your activity's onActivityResult() callback.
  - The target activity starts as a sub-activity of the calling activity
  - In the sub-activity, before it exits, call setResult() to return the result code as well as any extra data through intent

    ```
    void setResult (int resultCode)
    void setResult (int resultCode, Intent data)
    ```
  - In the calling activity, call onActivityResult() to handle the data.

    ```
    Void onActivityResult(int requestCode, int resultCode, Intent data)
    ```
    - resultCode: RESULT_OK, RESULT_CANCELED, RESULT_FIRST_USER
      https://developer.android.com/reference/android/app/Activity.html#RESULT_CANCELED

Start a Service

- startService(intent): initiate a service or deliver new instructions to ongoing one. The Intent describes the service to start and carries any necessary data.
- bindService(intent, serviceConn, flags): establish a connection between the calling component and the target service. Can initiate the service if not already running.
- With Android 5.0 (API level 21) and later, you can start a service with JobScheduler.

Send a Broadcast Intent

- sendBroadcast(intent): send a broadcast intent message to all interested broadbast receivers.
  - sendBroadcast(intent, permission)
- The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging.
  - The broadcast intent includes the action string which identifies the event and possible additional information bundled into its extra field. Example
  - For a complete list of system broadcast actions, see the BROADCAST_ACTIONS.TXT file in the Android SDK. (e.g. sdk/platforms/android-25/data/broadcast_actions.txt)
  - Each broadcast action has a constant field associated with it. For example, the value of the constant ACTION_AIRPLANE_MODE_CHANGED is “android.intent.action.AIRPLANE_MODE”

Broadcast Receiver

- Receive broadcast messages (intents), including both system defined and self defined intents. (Mostly from other applications, but can also receive from the same application.)
- The broadcast receiver can be triggered even when the app is not running foreground, or even not running at all. But not abuse it to run jobs in the background that can affect the system performance.
- Usually define an intent filter to subscribe it to certain messages.
  - Though it can also receive explicit intents.
- Need to register it either through
  - The Manifest file statically. (In Android 8+, you cannot register for implicit intents in the manifest file except several intents such as ACTION_BOOT_COMPLETED)
  - In the code dynamically:

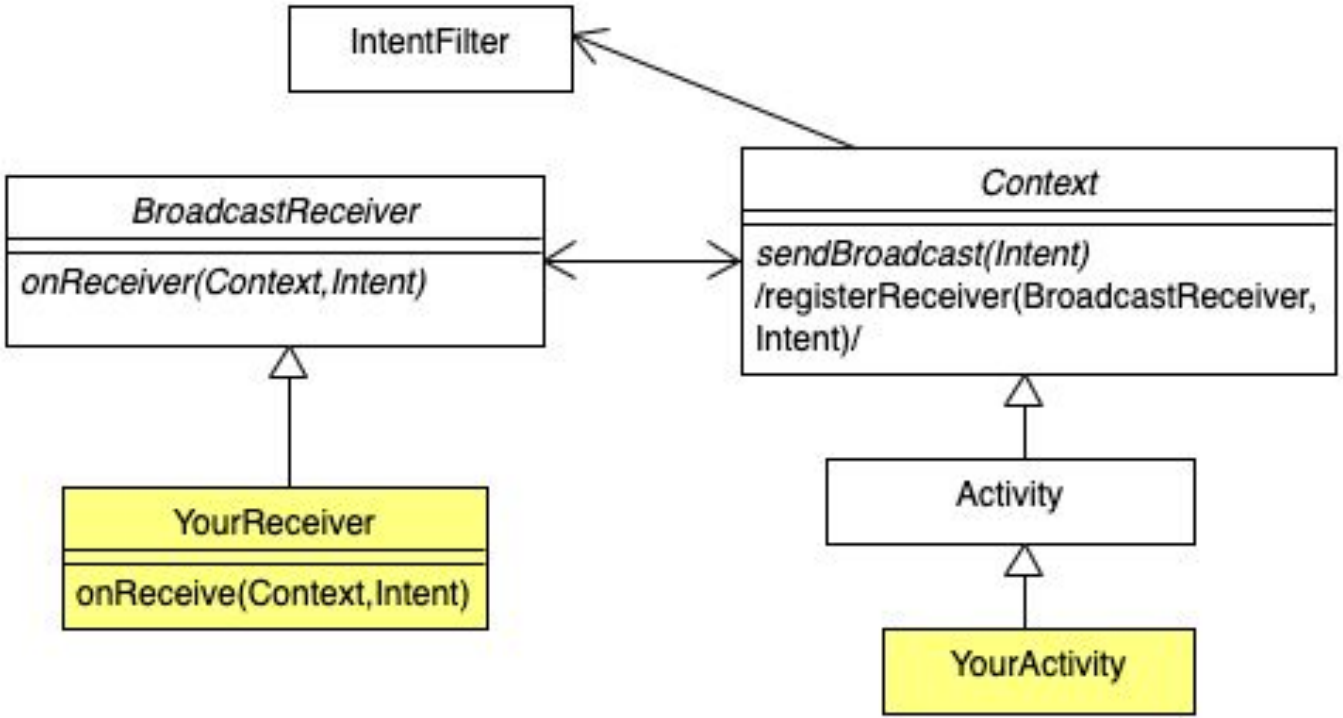
Broadcast Receiver

```
<receiver android:name=".MyBroadcastReceiver"  android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <action
android:name="android.intent.action.INPUT_METHOD_CHANGED" />
    </intent-filter>
</receiver>
```

~~BroadcastReceiver br = new MyBroadcastReceiver();~~

```
IntentFilter filter = new
IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
intentFilter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);
getApplicationContext().registerReceiver(br, filter);
```

Broadcast Receiver



Some Questions

- What is a process? What is a thread? Why need them?
- How to create them? How to communication between the main UI thread and the background threads/process?
- What is a service? What are types of service?
- How to create a service? How to use a service?
- How to run services/tasks in a separate thread/process to improve the performance?

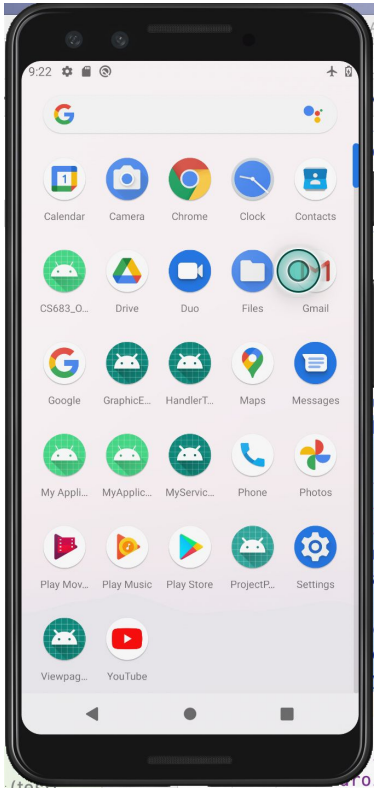
Processes

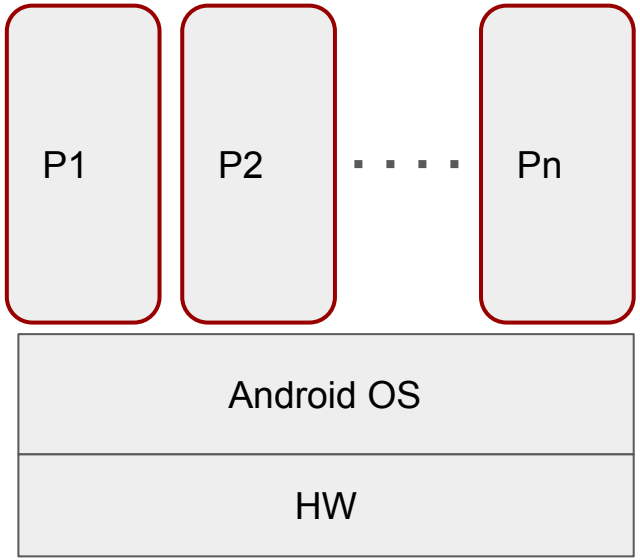
- A process is a running program.
- A process is a resource consumption unit. The OS kernel allocates resources such as CPU, memory, file, I/O etc to each process
- A process is a protection unit. Each process has its own virtual address space, so that its memory will not be accessed by other arbitrary process.
- An android application runs in a process (and a single thread in it) by default. Each application runs in its own sandbox, with its own memory address space and storage space for data.
- That means all components such as activities, services are running in the same process and the same thread of execution by default.
- Additional processes can be created and defined to run different components in an application.
- Communication between processes are more expensive and less trivial.

Process

A Process:

- A running program
- A resource consumption unit
- A protection unit (a sandbox)
- IPC (Inter-Process Communication) is expensive.





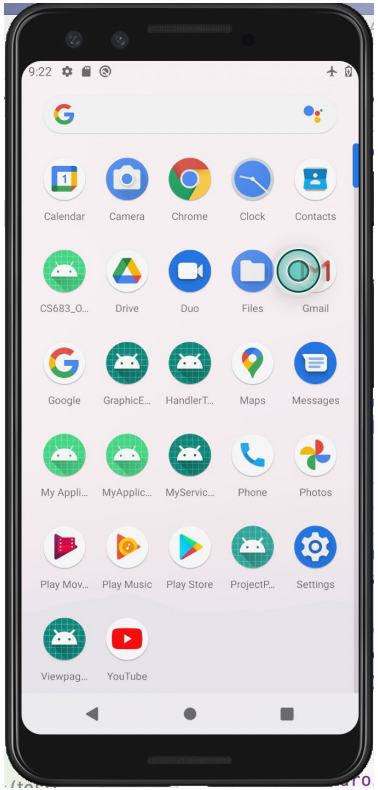
Process & Thread

A Process:

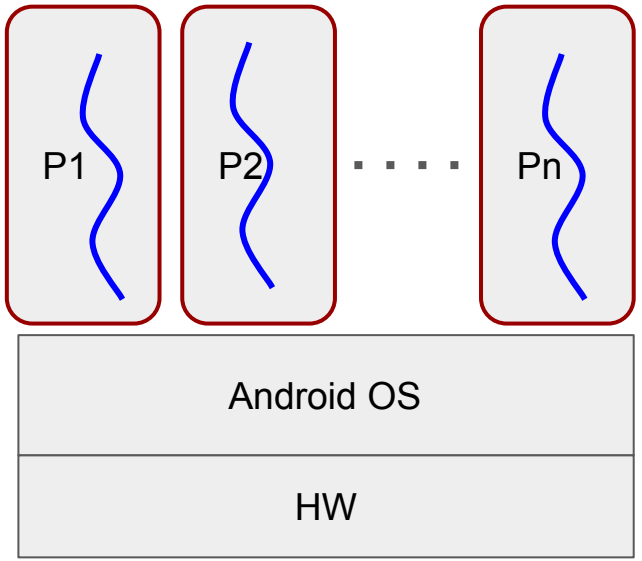
- A running program
- A resource consumption unit
- A protection unit (a sandbox)
- IPC (Inter-Process Communication) is expensive.

A Thread:

- An execution unit
- Has an execution path
- A parallelism unit
- A process has one thread by default.



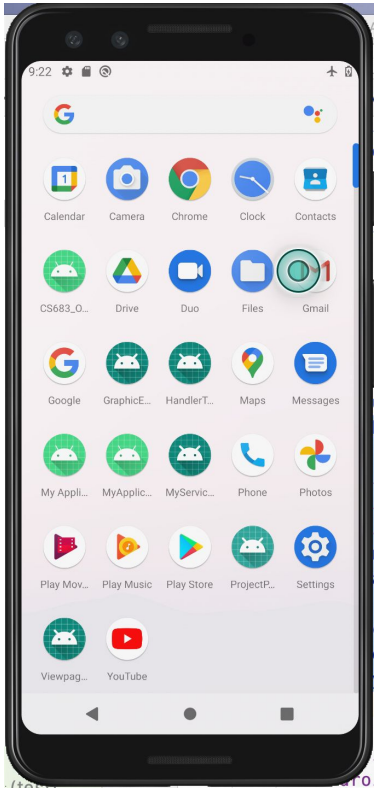

```
f() {
        a();
        b();
}
```



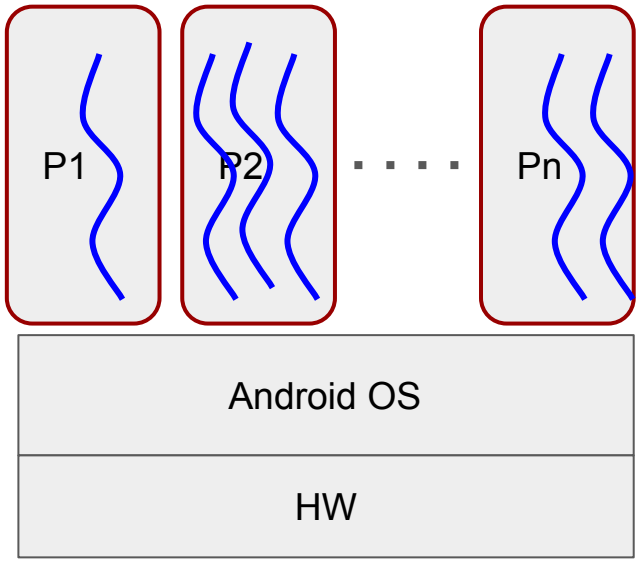
Process & Thread

Process vs Thread

- A process can has multiple threads, enabling the parallelism within the sandbox.
- Threads within the same process share most resources
- Communication between threads within the same process is lighter and cheaper.



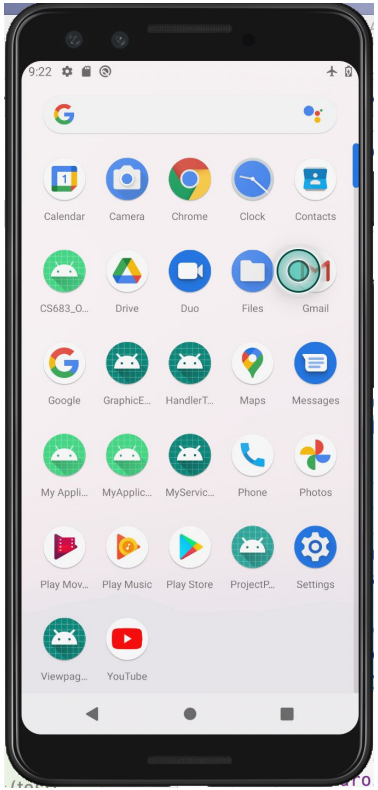

```
f() {
        a();
        b();
}
```

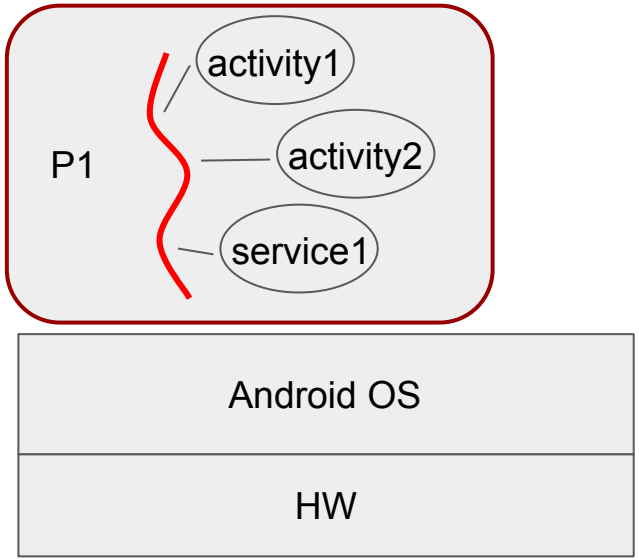


Process & Thread

An Android application is

- **Single Process Single Thread** by default
  - Main thread (UI thread)
  - All components (e.g. activities, services) are running in the same thread



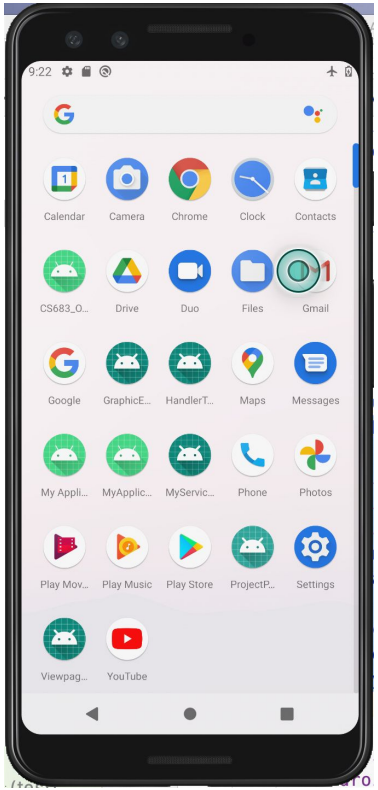


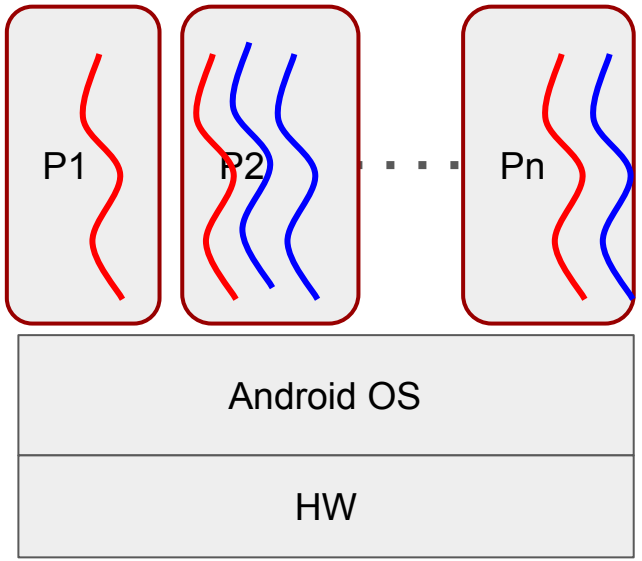
Process & Thread

An Android application is

- **Single Process Single Thread**
- **Single Process Multi-threads**
- Multi-Processes with single thread in each process
- Multi-Processes with Multi-threads in each process

(use android:process attribute in the component to specify the process it should run.)



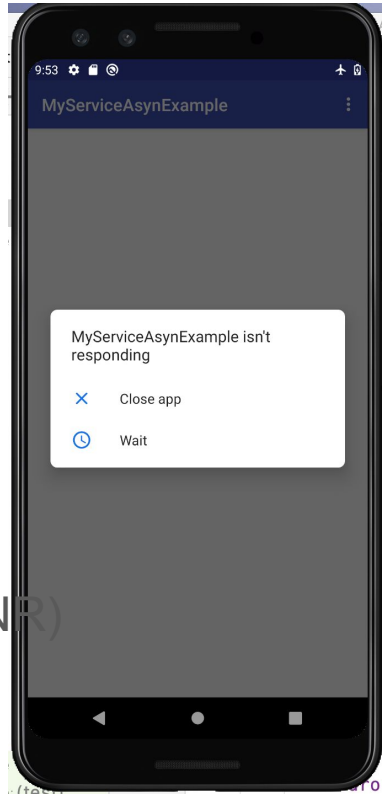


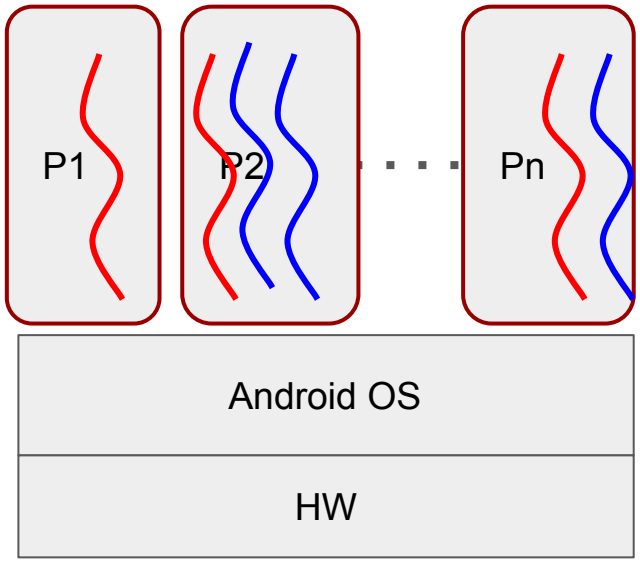
Process & Thread

An Android application

- **UI components (activities, views, event listeners) can only run on the UI thread.**
- Any **long** operation need to run on **other thread(s)** in order NOT to block the UI thread

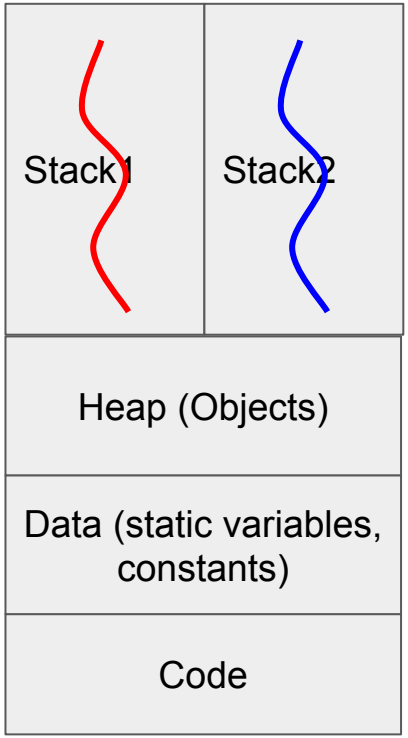
"Application not responding" (ANR) dialog



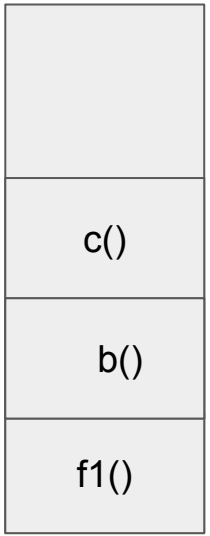


Process & Thread

```
f1() {
        a();
        b();
}
a(){
        c()
}
```



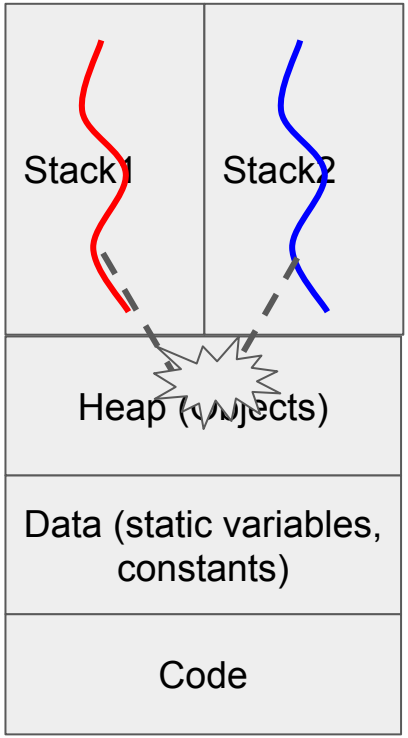

```
f2() {
        m();
        n();
}
```



f1() stack frame: local variables, parameters, return address

Thread Communication

```
f1() {
        a();
        b();
}
a(){
        c()
}
```

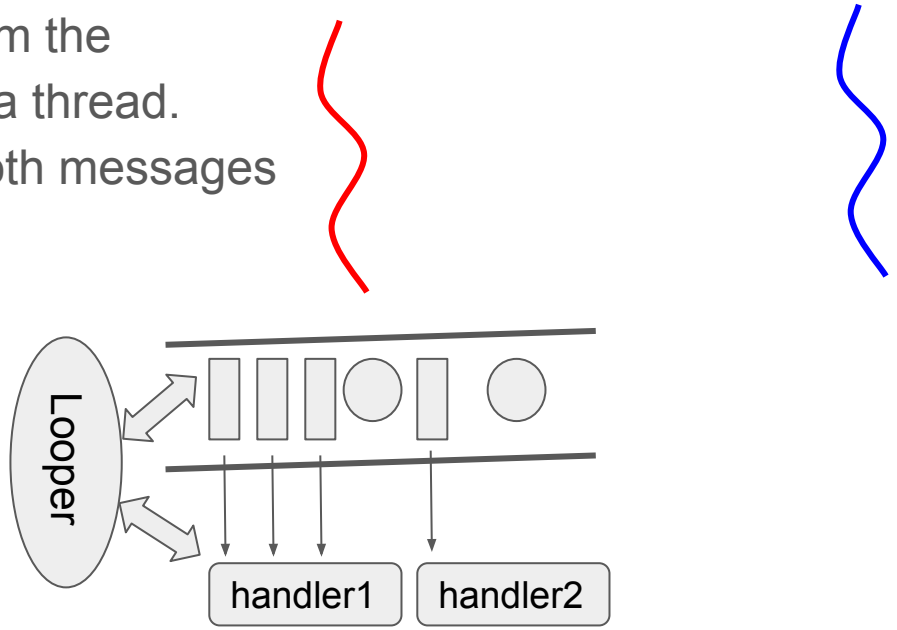



```
f2() {
        m();
        n();
}
```

- Thread communication is cheaper, but synchronization is needed
- Synchronization using low level primitives Semaphore, mutex, wait are not trivial and error prone.
- Android provides Handler and other high-level APIs to make things easier.

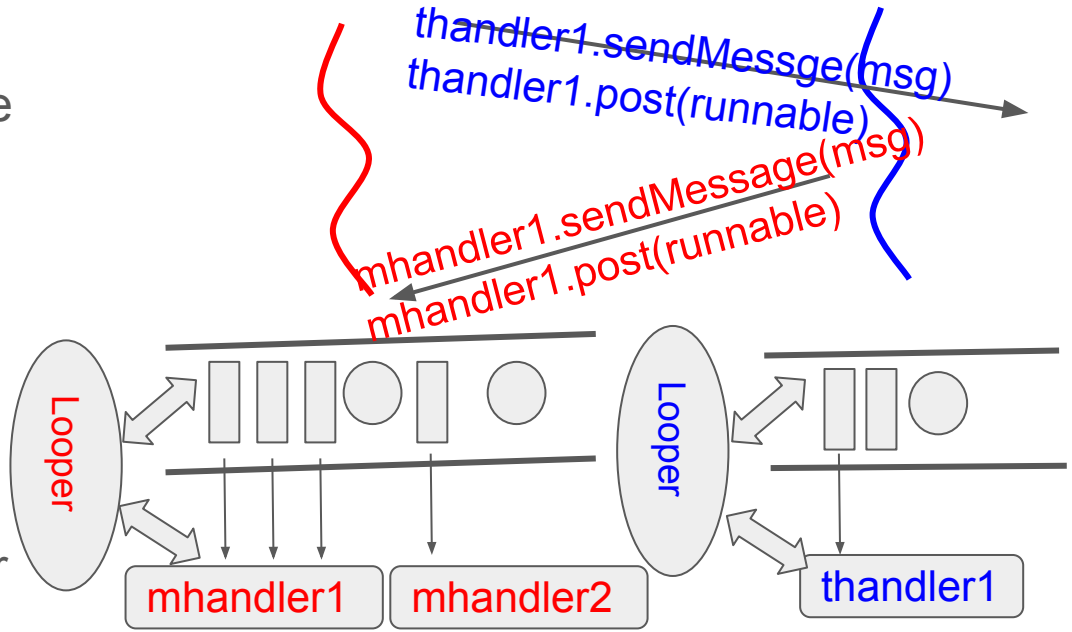
Handler

- A looper dispatches message from the message queue associated with a thread.
- A message queue can contain both messages and runnables
- A handler is associate with a thread and the message queue.
- The UI thread has a default MainLooper.
- Handlers need to be created explicitly and need a looper.
- Multiple handlers can be created for the same message queue
- A basic Java thread doesn't come with a looper.



Thread Communication

- HandlerThread comes with a Looper that can be used to create handlers for the associated HandlerThread
- Use the created handler to send messages or runnables to its associated HandlerThread.
- Use the handler created for the main thread to send messages or runnables to the main thread

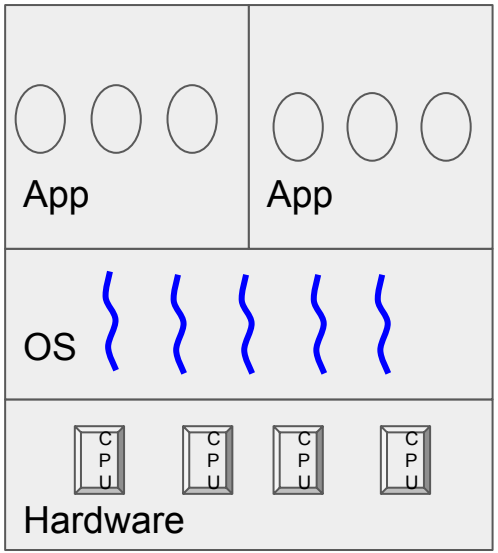


Handler APIs

- Call post(Runnable) to enqueue a Runnable object to be called in the associated thread when they are received.
- Call sendMessage(Message) to enqueue a Message object containing a bundle of data that will be processed by the Handler's handleMessage(Message) method in the associated thread.
- HandleMessage() can be passed as a callback function when creating a Handler.
- The given Runnable or Message will then be scheduled in the Handler's message queue and processed when appropriate.
- Call postDelay(Runnable, delay) to enqueue a Runnable object to be called at a future time in the associated thread when they are received.

Parallelism

- Hardware level Parallelism: true parallelism
  - Multiple CPUs/CPU cores
  - Can run (access the data) at the same time point
- OS level parallelism: thread (kernel level thread)
  - Each thread has its own execution path (stack), managed by OS
  - CPU scheduler (in OS) schedule threads to run on the same or different CPUs.
  - By default, each application runs in its own process and has at least one thread of execution.



Kotlin and Android Coroutines

- Support application user level parallelism. Implemented by some programming languages
- A routine (method) can be suspended and resumed.
- Can help prevent long-running tasks from blocking the main thread.
- Light weight, few memory leak, built-in cancellation support, jetpack integration
- A coroutine is not bound to any particular thread. It may suspend its execution in one thread and resume in another one.

Kotlin and Android Coroutines

- Suspending functions
- Start a coroutine without getting the returned result: the launch function
  - Coroutine Scope: determine the lifetime of the coroutine, implemented structured concurrency
  - Dispatcher:Decide which thread the coroutine executes on
  - Return a job that can be canceled later.
- Start a coroutine and getting the returned result
  - async: return a deferrable job
  - await: get returned result from the job

Kotlin Coroutine

```
suspend fun loadAllProjects(): List<Project> {
    return projectDao.loadAllProjects()
}

fun reloadAllProjects() {
    val bgJob = viewModelScope.async {
            projectRepository.getAllProjects()
        }
    viewModelScope.launch {
        allProjectsLiveData.value = bgJob.await()
    }
}
```

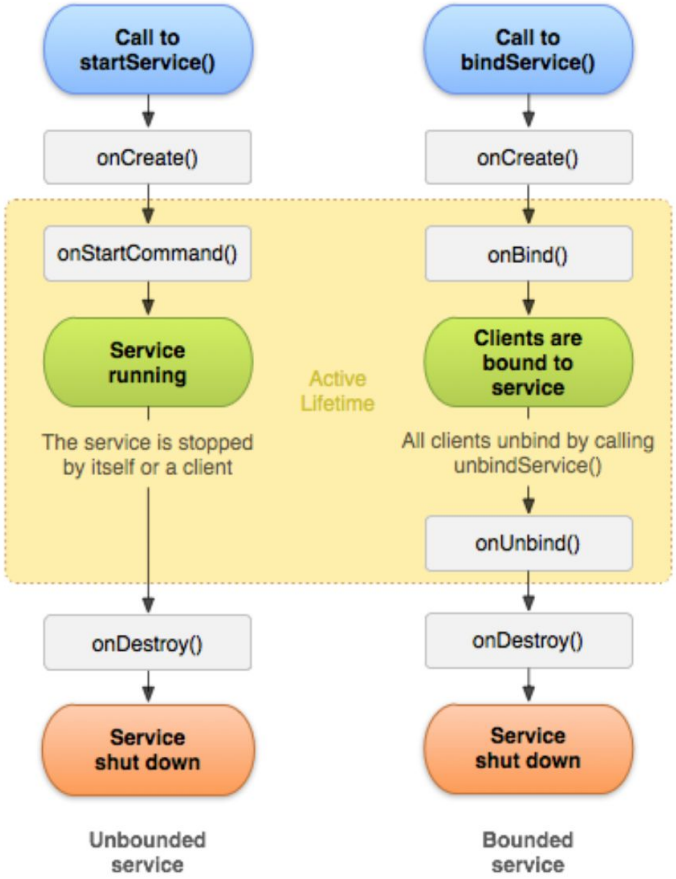
Asynchronous Room Dao

- For one-time write operations, use Kotlin suspend functions.
- For one-time read operations, use Kotlin suspend functions.
- For Observable read operations, use JetPack LiveData.. Room generates all the necessary code to update the LiveData object whenever a database is updated. The generated code runs the query asynchronously on a background thread when needed.

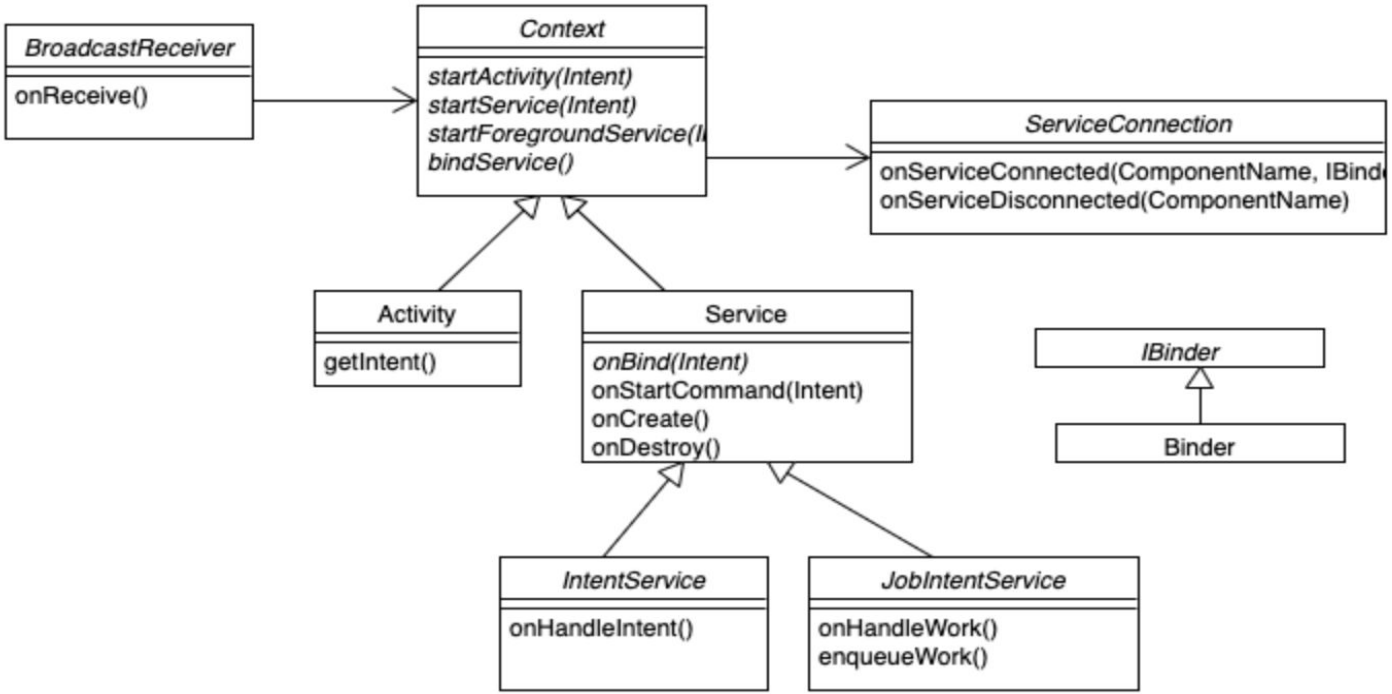
Service

- A service is an application component WITHOUT the user interface
- A background service performs an operation that isn't directly noticed by the user.
- A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC).
- A foreground service performs some operation that is noticeable to the user. Foreground services must display a status bar icon. Foreground services continue running even when the user isn't interacting with the app.
- https://developer.android.com/guide/components/services.html

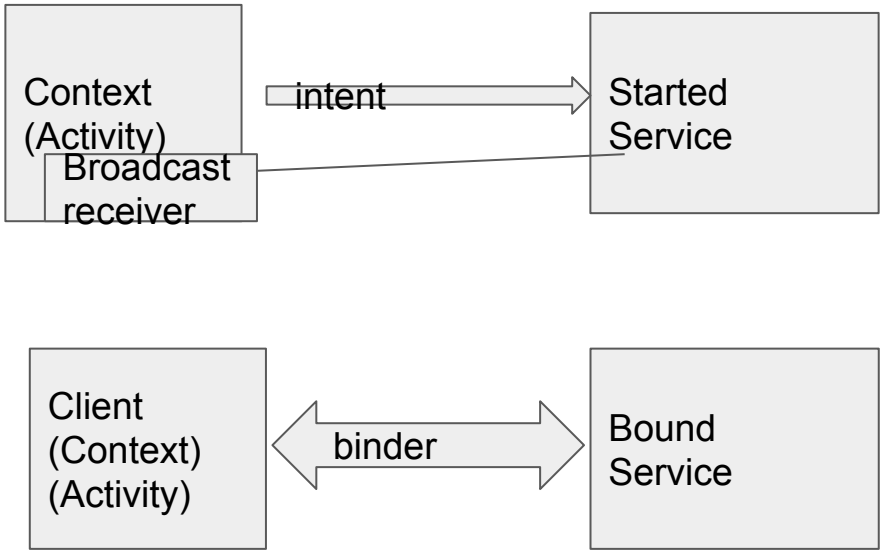
Service Lifecycle



Service Lifecycle



Services


Context
(Activity)
Broadcast
receiver
intent
Started
Service
Client
(Context)
(Activity)
binder
Bound
Service

IntentService

- Creates a separate worker thread that executes all of the intents.
- Creates a work queue that passes one intent at a time to your onHandleIntent() implementation
- Stops the service after all of the start requests are handled,
- Provides a default implementation of onBind() that returns null.
  Provides a default implementation of onStartCommand() that sends the intent to the work queue and then to your onHandleIntent() implementation.The intents are queued and handled one at a time.

JobIntentService

- Helper for processing work that has been enqueued for a job/service.
- Similar to IntentService, it creates a separate worker thread to handle all intents.
- But it is scheduled as a job. So the Android system has more control to decide when to schedule it and minimizing its effect on system performance.
- Define the service as the JobService in the manifest

```
<service
   android:name=".services.MyJobService"
   android:permission="android.permission.BIND_JOB_SERVICE" />
```

- The work you enqueue will ultimately handled by HandleWork(Intent).
- To enqueue the work, call JobIntentService.enqueueWork()

Bind a Service

- In the client (who call bindService())
  - Create a ServiceConnection object that implements two callbacks (onServiceConnected(), onServiceDisconnected())
  - In onServiceConnected(), the reference to the binder which will be used to communicate with the service is passed in.
    - To bind a remote service (the client and service are not in the same process), create a Messenger that wraps around IBinder, which can be used to send messages later.
  - Call bindService() and pass the ServiceConnection object.
- In the service
  - Implement onBind(). Create an IBinder object and return it.
  - In the remote service, can create a Messenger by creating a handler that implement handlerMessage().
- https://developer.android.com/guide/components/bound-services.html

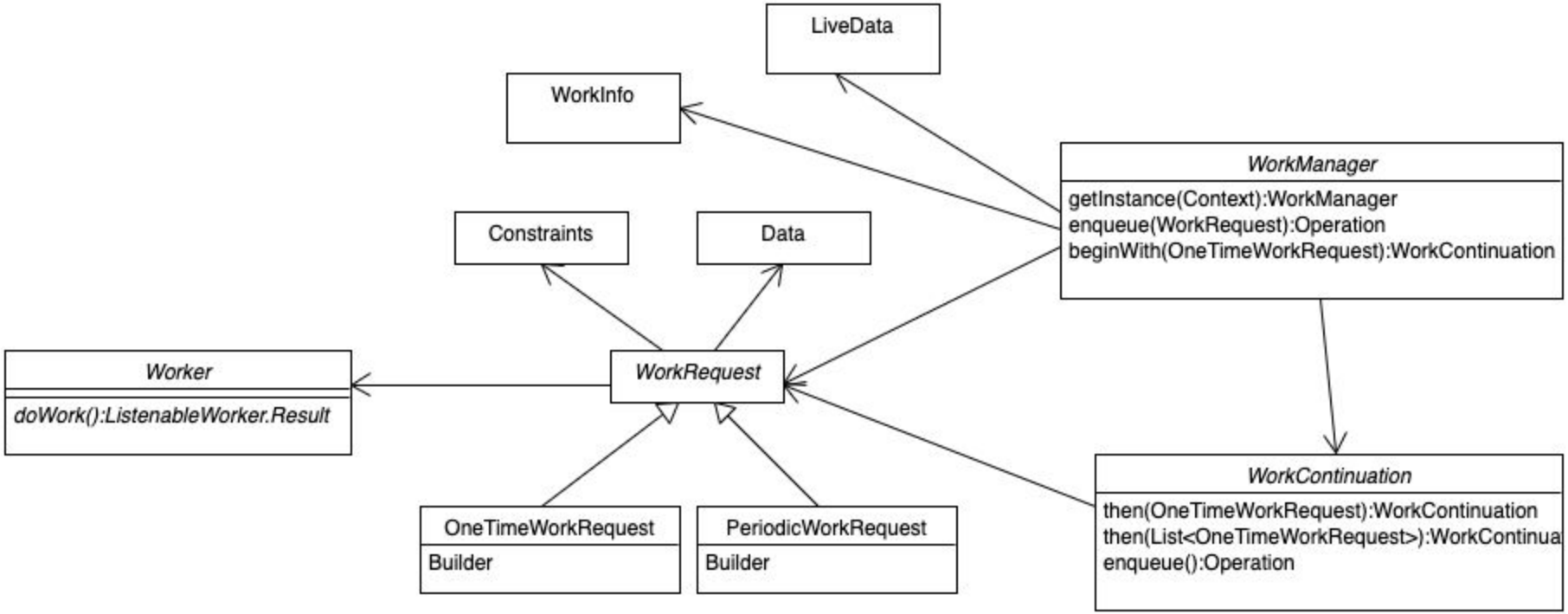
JobScheduler

- JobScheduler: an abstract class. Define an API for scheduling various types of jobs against the framework that will be executed in your application's own process.
  - schedule()
- JobInfo: Container of data passed to the JobScheduler fully encapsulating the parameters required to schedule work against the calling application. These are constructed using the JobInfo.Builder. You must specify at least one sort of constraint on the JobInfo object that you are creating. The goal here is to provide the scheduler with high-level semantics about the work you want to accomplish. Doing otherwise with throw an exception in your app.

Work Manager

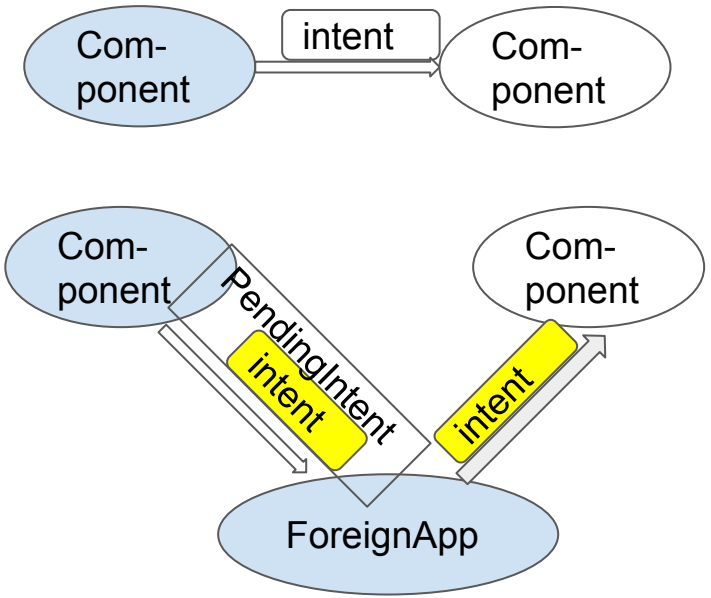
- Used to schedule reliable, asynchronous tasks that are expected to run even if the app exits or the device restarts.
- The WorkManager API is a suitable and recommended replacement for all previous Android background scheduling APIs, including FirebaseJobDispatcher, GcmNetworkManager, and Job Scheduler.
- 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.

Work Manager



Pending Intent

- A PendingIntent object is a wrapper around an Intent object.
- A PendingIntent is a token given to a foreign application (e.g. NotificationManager, AlarmManager, Home Screen AppWidgetManager, or other 3rd party applications), which allows the foreign application to use the sending application permissions to send the included intent.
- Since the foreign application will have the same permission as the sending



When to use Pending Intents

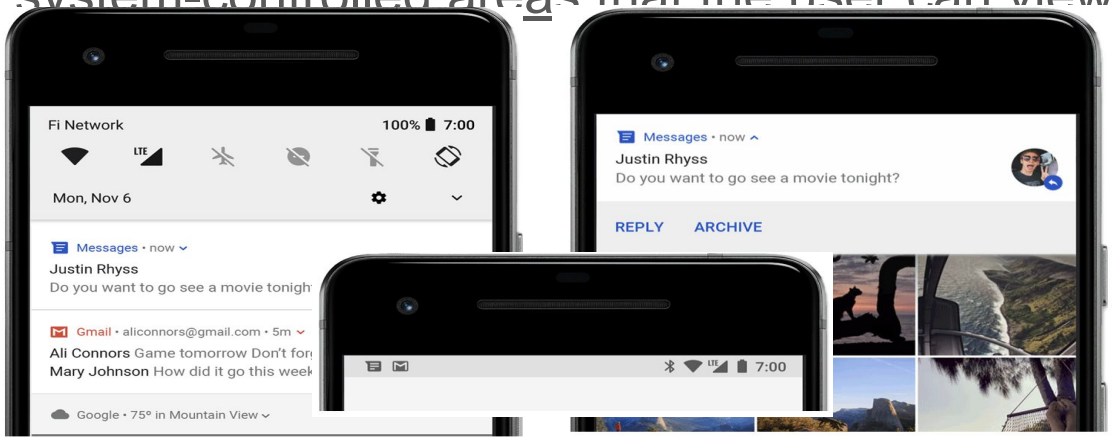
- Two examples:
  - Declaring an intent to be executed when the user performs an action with your Notification (the Android system's NotificationManager executes the contained Intent).
  - Declaring an intent to be executed at a specified future time (the Android system AlarmManager executes the contained Intent).

PendingIntent APIs

- PengindIntent is a public final class.
- To create a pending intent, call one of the following static methods from the PengindIntent class depending on whether the contained intent is to start activity, send broadcast, or start service.
  - PendingIntent getActivity (Context context, int requestCode, Intent intent, int flags): Retrieve a PendingIntent that will start a new activity, like calling Context.startActivity(Intent).
  - PendingIntent getBroadcast (Context context, int requestCode, Intent intent, int flags): Retrieve a PendingIntent that will perform broadcast , like calling Context.sendBroadcast(Intent).
  - PendingIntent getService (Context context, int requestCode, Intent intent, int flags): Retrieve a PendingIntent that will start service ,

Notification

- A notification is a message you display to the user outside of your app's normal UI.
- When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.



Notification APIs

- Class: Notification,
  - Nested class: Notification.builder, Notification.Action
  - To be compatible with the older versions, NotificationCompat is used to help access features of Notification
- Class: NotificationCompat
  - Nested class: NotificationCompat.builder
    - Constructor: NotificationCompat.builder(context), NotificationCompat.builder(context, channelid)
    - Various set and add methods to set the attributes: setContextTitle(title), setContentText(text), setContentIntent(pengidIntent), addAction(action), addAction(action, title, pendingIntent), addExtras(extras), ...
    - Notification.Build()

Notification

- Sending notification
  - Create a notification builder. Use Notification.builder or NotificationCompat.builder.
  - Set up necessary attributes such as title, icon, actions,etc
  - Call build() to create a notification
  - Call notify() from the NotificationManager to notify the system about the notification
  - In Android O(API 26), every notification needs to be associated with a notification channel.
  - To send remote notifications from a server, firebase services can be used.

Foreground Service

- Foreground Service = Started Service + a Notification
- A foreground service is noticeable to users through a notification.
- Start a foreground service using startForegroundService(Intent service).
- The foreground service will call startForeground(int, android.app.Notification) once it begins running.
- A service can be used to receive intents with different actions and handle differently.

Alarms

- Alarms (based on the AlarmManager class) give you a way to perform time-based operations outside the lifetime of your application.
- Get the reference to the AlarmManager by calling
  `AlarmManager context.getSystemService(Context.ALARM_SERVICE).`
- Specify the alarm time and a PendingIntent. When an alarm goes off, the intent wrapped in the PendingIntent is sent by the system, automatically starting the target component even if it is not already running.
  `AlarmManager.set(int type, long triggerAtMillis, PendingIntent operation)`
- Can also set repeated alarms:
  `AlarmManager.setRepeating(int type, long triggerAtMillis, long intervalMillis, PendingIntent operation)`

JobService

- JobService: an abstract subclass of Service. It is the base class that handles asynchronous requests that were previously scheduled.
  - Override onStartJob(JobParameters) to implement your job logic.
  - This service executes each incoming job on a Handler running on your application's **main thread**. This means that you must offload your execution logic to another thread/handler/AsyncTask of your choosing. Not doing so will result in blocking any future callbacks from the JobManager.
  - onStopJob(android.app.job.JobParameters), which is meant to inform you that the scheduling requirements are no longer being met.
  - Entry point for the callback from the JobScheduler.

Questions?

AsyncTask

- An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread.
- It is generally used for short operations (a few seconds at the most.)
- If the app is in the background and the app is terminated by Android, your background processing is also terminated.
- Once created, a task is executed simply by calling execute() method in the UI thread. This method will then invoke
  - onPreExecute() : in the UI thread
  - doInBackground(Params…): need to be override and run in the background thread
  - onPostExecute(Result): in the UI thread