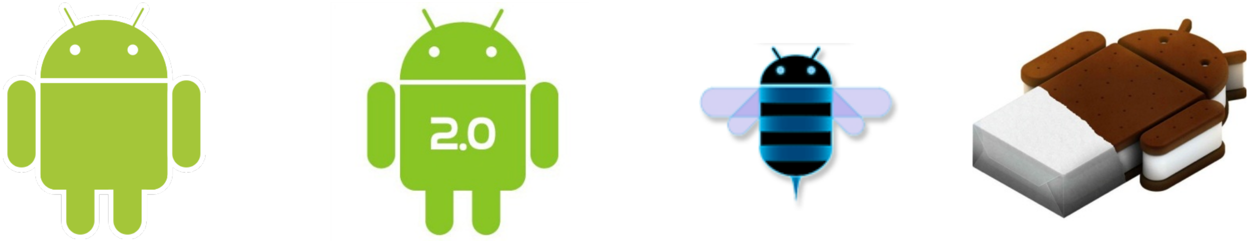


Programming with Android: Notifications, Threads, Services

Luca Bedogni
Dipartimento di Scienze dell'Informazione
Università di Bologna

Outline

Android: Where are we now …

TILL NOW → Android Application structured has a single **Activity** or as a group of Activities …

- **Intents** to call other activities
- **Layout** and **Views** to setup the GUI
- **Events** to manage the interactions with the user

Activities executed only in **foreground** …

- What about *background* activities?
- What about *multi-threading* functionalities?
- What about *external events* handling?

Android: Where are we now ...

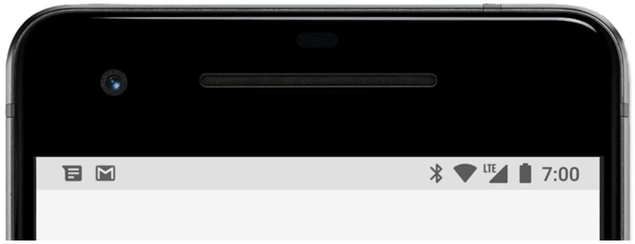
EXAMPLE: A simple application of *Instantaneous Messaging* (**IM**)

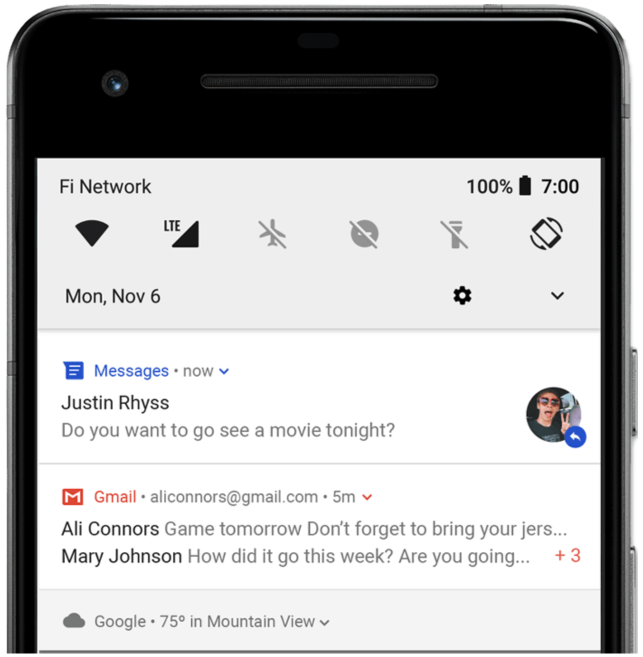
- Setup of the application **GUI** ✓
- GUI **event** management ✓
- Application **Menu** and **Preferences** ✓
- Updates in **background** mode ✗
- **Notifications** in case of message reception in background mode ✗

Notifications Overview

- Notifications are messages from your application
  - Reminders
  - External events
  - Timely information
- Can serve 2 cases:
  - Only informative: a message is displayed to the user
  - Informative and active: by clicking on it, it is possible to open the APP or perform directly some operations

Notification Types




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

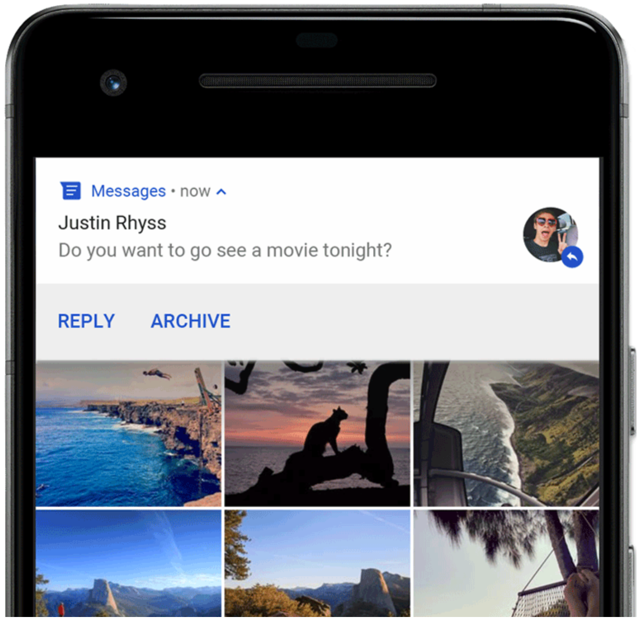
Google • 75° in Mountain View

When the notification is created, its icon appears in the status bar

Scrolling down the status bar reveals additional details about the notification

Some notification can also reveal further information by swiping them down

Notification Types

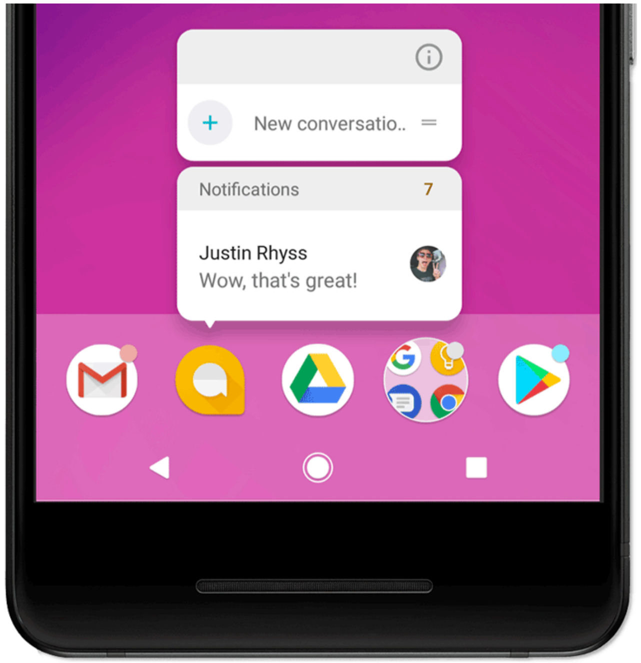


Justin Rhyss
Do you want to go see a movie tonight?

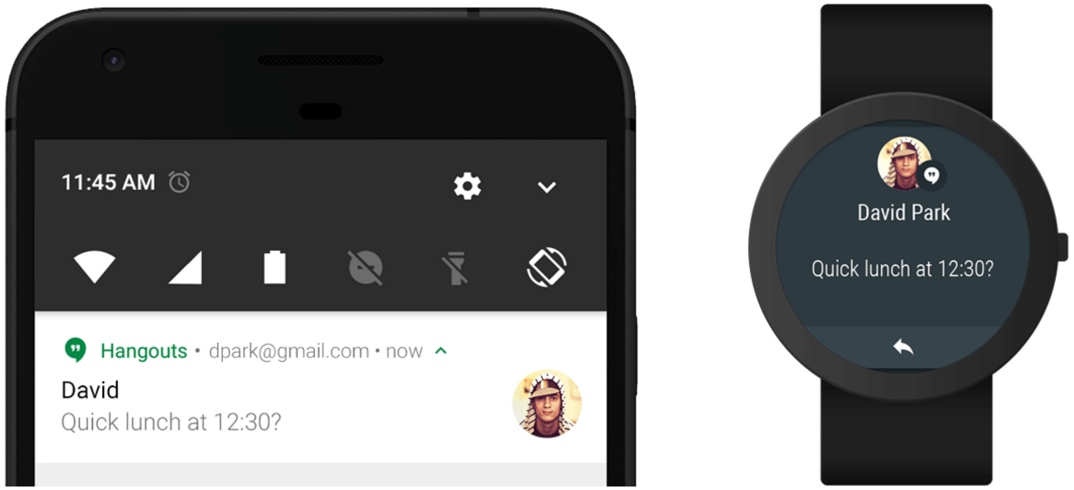
Heads up notification: useful for important information, and to notify the user while watching a full screen activity (starting from 5.0)

Notifications can also be visible in the lock screen. The developers can configure the amount of details which has to be made visible.

More notification Types

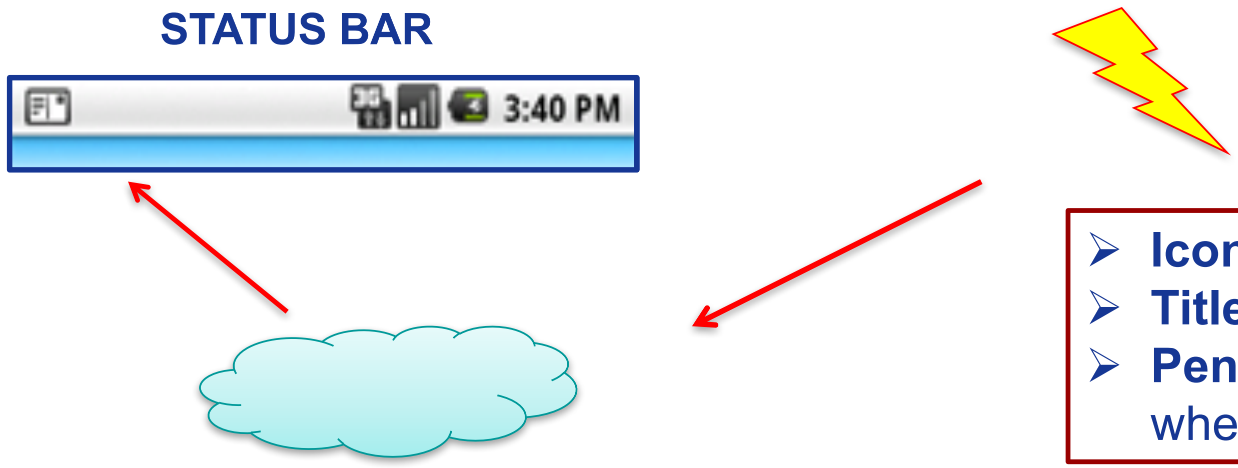


Icon badge: starting with Android 8.0. Users can get notification information about an app.



Wearables, to show the same notification on the hand-held device and wearable

Android: Status Bar Notifications



- **Icon** for the status bar
- **Title** and **message**
- **PendingIntent** to be fired when notification is selected

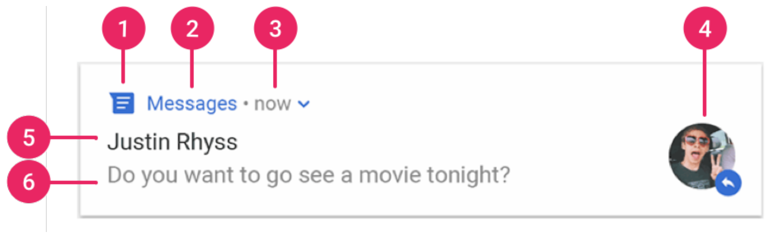
Notification Manager

Android system component
Responsible for notification management
And status bar updates

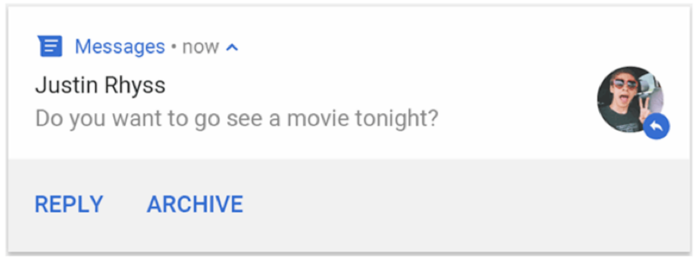
OPTIONs:

- Ticket-text message
- Alert-sound
- Vibrate setting
- Flashing LED setting
- Customized layout

How a notification is made



1. Small icon
2. App name
3. Timestamp
4. Optional Large Icon
5. Optional Title
6. Optional Text



Starting with Android 7.0, users can perform simple actions directly in the Notification

Grouping Notification

- Notifications can also be updated and grouped together
  - Notifications should be updated if they refer to the same content which has just changed
- If more than one notification is needed for the same app, they can be grouped together
  - Starting with Android 7.0
- Starting with Android 8.0
  - Notification should also set a channel
    - To let users have more control about which kind of notification they want to see
  - Channels have also an associated priority

Android: Status Bar Notifications

➢ Follow these steps to send a Notification:

1. Get a **reference** to the **Notification Manager**

```
NotificationManager nm=(NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE)
```

or

```
NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);
```

2. **Build** the Notification message

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this, ”myChannel");
mBuilder.setContentTitle("Picture Download").setContentText("Download in progress")
      .setSmallIcon(R.mipmap.ic_launcher_round).setPriority(NotificationCompat.PRIORITY_LOW);
```

3. **Send** the notification to the Notification Manager

```
notificationManager.notify(myId, mBuilder.build());
```

Android: Status Bar Notifications

Define what will happen in case the user selects the notification

```
Intent newIntent = new Intent(this, NotificationService.class);
newIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);
newIntent.putExtra("CALLER","notifyService");
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, newIntent, 0);
```

Android: Status Bar Notifications

Add (optional) flags for notification handling

```
mBuilder.setAutoCancel(true)
```

Send the notification to the Notification Manager

```
notificationManager.notify(0, mBuilder.build());
```

Add a **sound** to the notification

```
mBuilder.setSound(URI sound);
```

Android: Status Bar Notifications

Add **flashing lights** to the notification

```
mBuilder.setLights(0xff00ff00, 300, 100);
```

This sets a green led
The LED flashes for 300ms and turns it off for 100ms

Add a **vibration pattern** to the notification

```
mBuilder.setVibrate(long [])
mBuilder.setVibrationPattern(long []) // From API 26
```

Android: Processes and Threads

- By default, all components of the same application run in the same process and thread (called "**main** thread" or "**UI**" thread).

- In **Manifest.xml**, it is possible to specify the process in which a component (e.g. an activity) should run through the attribute **android:process**.

- Processes might be killed by the system to reclaim memory.
  - **Processes' hierarchy** to decide the importance of a process.
  - Five *types*: Foreground, Visible, Service, Background, Empty.

Android: Thread Management

- Android natively supports a **multi-threading** environment.

- An Android application can be composed of multiple *concurrent* threads.

- How to create a thread in Android? … Like in Java!

  - extending the **Thread** class **OR**
  - implementing the **Runnable** interface
  - **run**() method executed when MyThread.**start**() is launched.

Android: Thread Management

```
public class MyThread extends Thread {

    public MyThread() {
        super ("My Threads");
    }

    public void run() {
        // do something
    }
}
```

```
myThread m=new MyThread();
m.start();
```

Android: Thread Management

The **UI** or **main** thread is in charge of dispatching events to the user interface widgets, and of drawing the elements of the UI.

- Do not block the UI thread.
- Do not access the Android UI components from outside the UI thread.

QUESTIONS:

How to update the UI components from worker threads?

Android: AsyncTask

AsyncTask is a Thread helper class (Android only).

- ✧ Computation running on a **background** thread.
- ✧ Results are published on the **UI** thread.
- ✧ Should be used for short operations

RULES

- ➢ AsyncTask must be created on the UI thread.
- ➢ AsyncTask can be executed only once.
- ➢ AsyncTask must be canceled to stop the execution.

Android: AsyncTask

```
private class MyTask extends AsyncTask<Par, Prog, Res>
```

Must be subclassed to be used

Par → type of parameters sent to the AsyncTask

Prog → type of progress units published during the execution

Res → type of result of the computation

EXAMPLES

```
private class MyTask extends AsyncTask<Void,Void,Void>
```

```
private class MyTask extends AsyncTask<Integer,Void,Integer>
```

Android: AsyncTask

EXECUTION of the ASYNCTASK

The UI Thread invokes the **execute** method of the AsyncTask:

```
(new Task()).execute(param1, param2 … paramN)
```

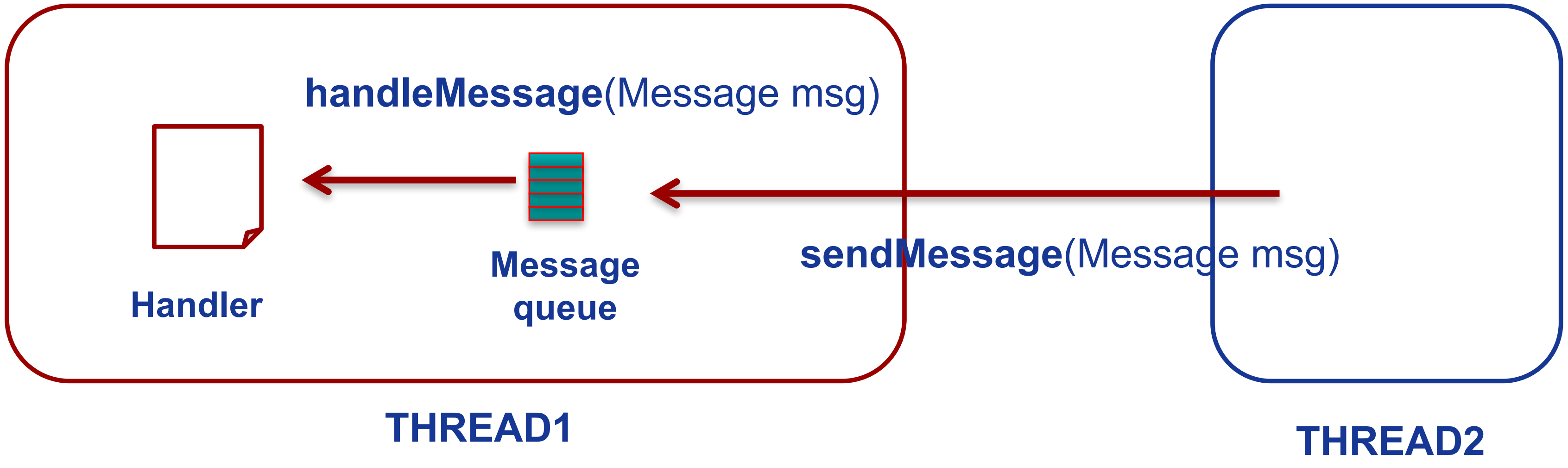
After **execute** is invoked, the task goes through four steps:

1. **onPreExecute**() → invoked on the UI thread
2. **doInBackground**(Params…) →computation of the AsyncTask
   - can invoke the publishProgress(Progress…) method
3. **onProgressUpdate**(Progress …) → invoked on the UI thread
4. **onPostExecute**(Result) → invoked on the UI thread

Android: Thread Management

Message-passing like mechanisms for Thread communication.

MessageQueue	→	Each thread is associated a queue of messages
Handler	→	Handler of the message associated to the thread
Message	→	Parcelable Object that can be sent/received



Android: Thread Management

Message loop is implicitly defined for the **UI** thread … but it must be explicitly defined for worker threads.

HOW? Use **Looper** objects …

```
public void run() {
        Looper.prepare();
        handler=new Handler() {
            public void handleMessage(Message msg) {
                // do something
            }
        }
        Looper.loop();
```

Android: Services

A **Service** is an application that can perform *long-running operations in background* and *does not provide a user interface*.

- **Activity** → UI, can be disposed when it loses visibility
- **Service** → No UI, disposed when it terminates or when it is terminated by other components

A Service provides a robust environment for background tasks …

Android: Services

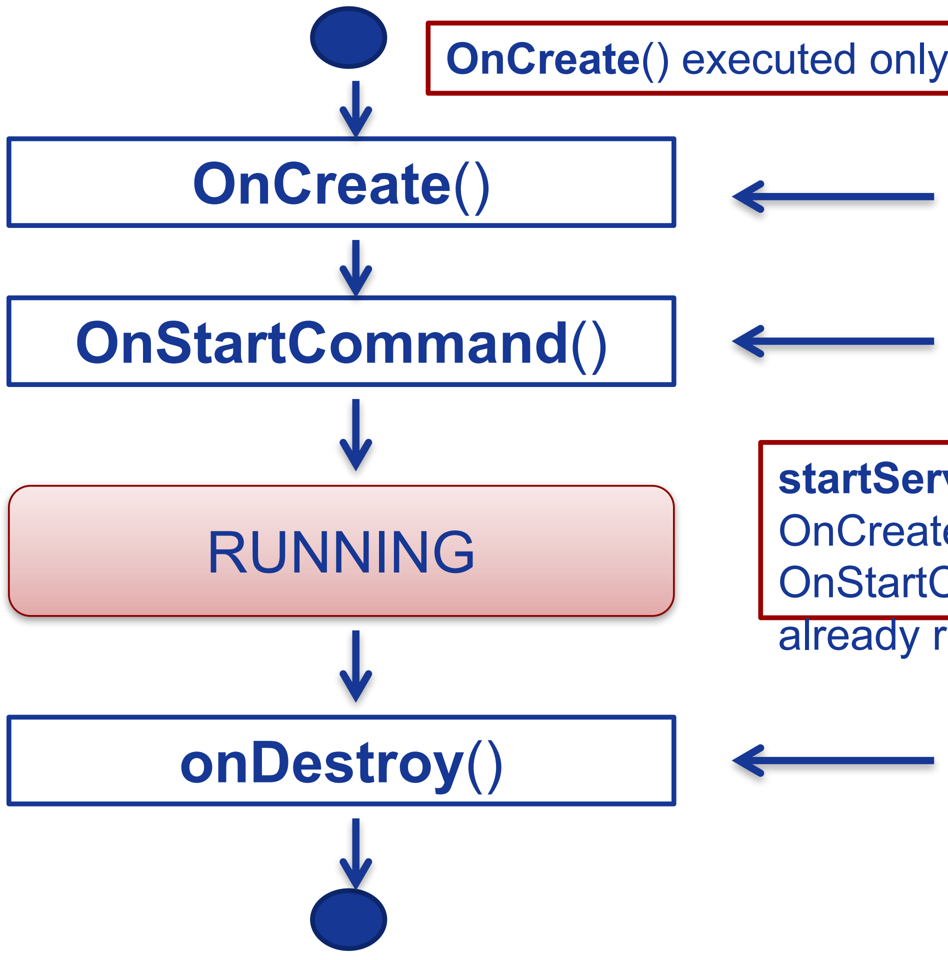
- A Service is started when an application component starts it by calling **startService(**Intent**)**.

- Once started, a Service can run in **background**, even if the component that started it is destroyed.

- *Termination* of a Service:

  1. **selfStop**() → self-termination of the service
  2. **stopService**(Intent) → terminated by others
  3. System-decided termination (i.e. memory shortage)

Android: Service Lifetime



OnCreate() executed only once when the Service is created.

startService() might cause the execution of OnCreate+OnStartCommand, or only of OnStartCommand, depending whether the Service is already running …

Android: Foreground Services

COMMON **MISTAKES**

- A **Service** provides only a **robust environment** where to host separate threads of our application.
  - A Service is not a separate process.
  - A Service is not a separate Thread (i.e. it runs in the main thread of the application that hosts it).
  - A Service does nothing except executing what listed in the **OnCreate**() and **OnStartCommand**() methods.
  - Behaviors of Local/Bound Services can be different.

Android: Foreground Services

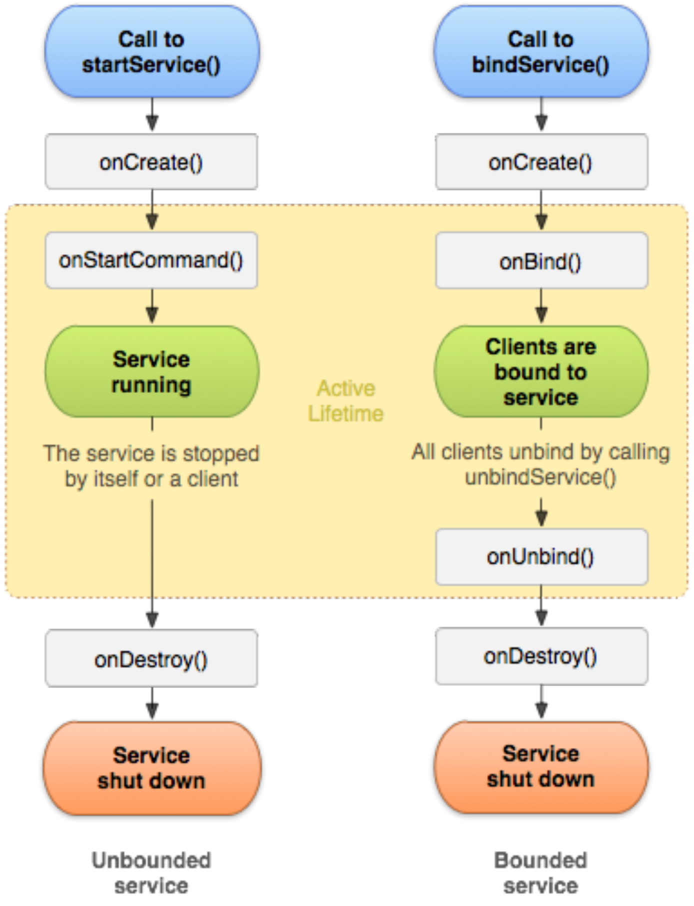
- A **Foreground** Service is a service that is continuously active in the Status Bar, and thus it is not a good candidate to be killed in case of low memory.

- The Notification appears between **ONGOING** pendings.

- To create a Foreground Service:
    1. Create a **Notification** object
    2. Call **startForeground**(id, notification) from onStartCommand()

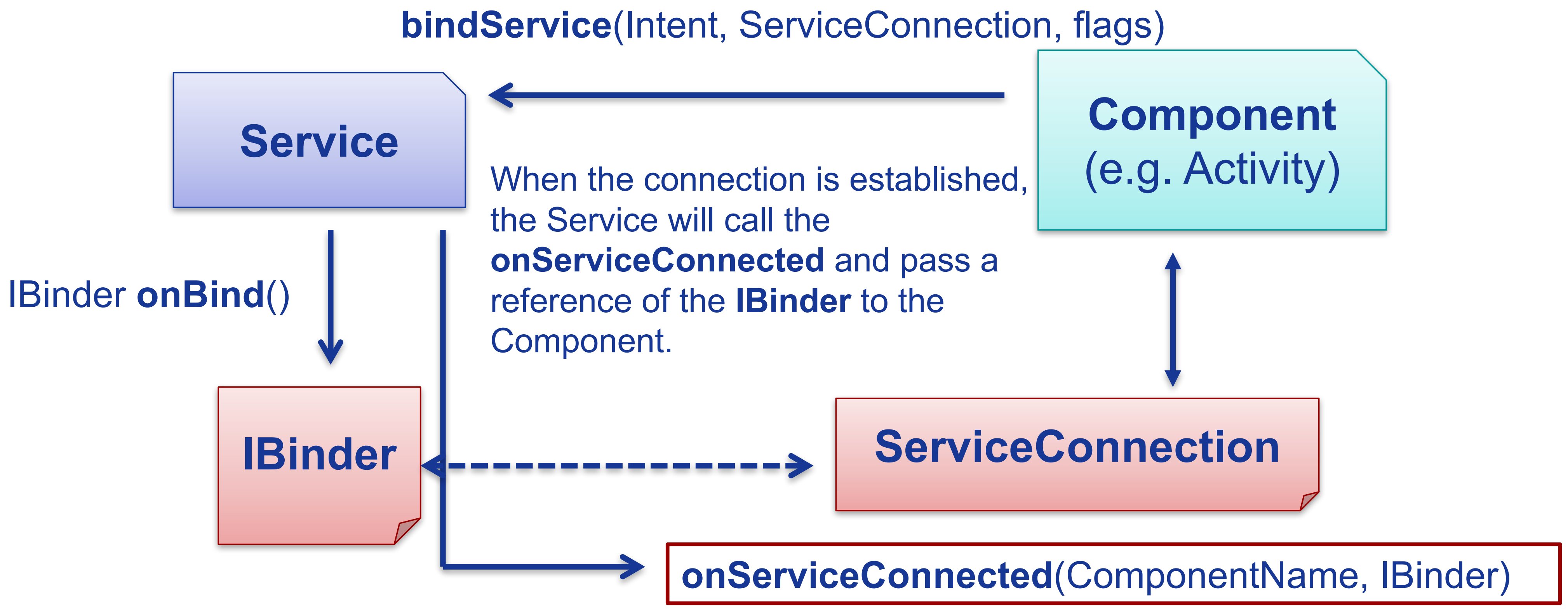
- Call **stopForeground**() to stop the Service.

Services and BoundServices



Android: Bound Service

➢ Through the **IBinder**, the Component can send requests to the Service …



bindService(Intent, ServiceConnection, flags)
Service
Component (e.g. Activity)
IBinder **onBind**()
IBinder
ServiceConnection
onServiceConnected(ComponentName, IBinder)

When the connection is established, the Service will call the **onServiceConnected** and pass a reference of the **IBinder** to the Component.

Android: Bound Service

➢ When creating a Service, an **IBinder** must be created to provide an Interface that clients can use to interact with the Service … HOW?

1. **Extending** the Binder class (local Services only)
   - Extend the Binder class and return it from **onBind**()
   - Only for a Service used by the same application

1. **Using** the Android Interface Definition Language (**AIDL**)
   - Allow to access a Service from different applications.

Android: Bound Service

```
public class LocalService extends Service {
	// Binder given to clients
	private final IBinder sBinder=(IBinder) new SimpleBinder();

	@Override
	public IBinder onBind(Intent arg0) {
		// TODO Auto-generated method stub
		return sBinder;
	}

	class SimpleBinder extends Binder {
		LocalService getService() {
			return LocalService.this;
		}
	}
}
```

Android: Bound Service

```
public class MyActivity extends Activity {
      LocalService lService;
      private ServiceConnection mConnection=new ServiceConnection() {

            @Override
            public void onServiceConnected(ComponentName arg0, IBinder bind) {
                  SimpleBinder sBinder=(SimpleBinder) bind;
                  lService=sBinder.getService();
                  ….
            }

            @Override
            public void onServiceDisconnected(ComponentName arg0) {
            }
            … bindService(new  Intent(this,LocalService.class),mConnection,BIND_AUTO_CREATE);

   };
```

Android: Intent Service

- Created for simpler services
  - Does not handle multiple request simultaneously
  - But runs on a separate thread
- Handles one Intent at a time
  - Through onHandleIntent()
  - Stops after the handling ended

```
public class myIntentService extends IntentService {

  public HelloIntentService() {      super(" myIntentService");  }

  @Override
  protected void onHandleIntent(Intent intent) {     // doSomething  }
}
```

Android: Broadcast Receiver

A **Broadcast Receiver** is a component that is activated only when specific events occur (i.e. SMS arrival, phone call, etc).

- **Registration** of the Broadcast Receiver to the event …

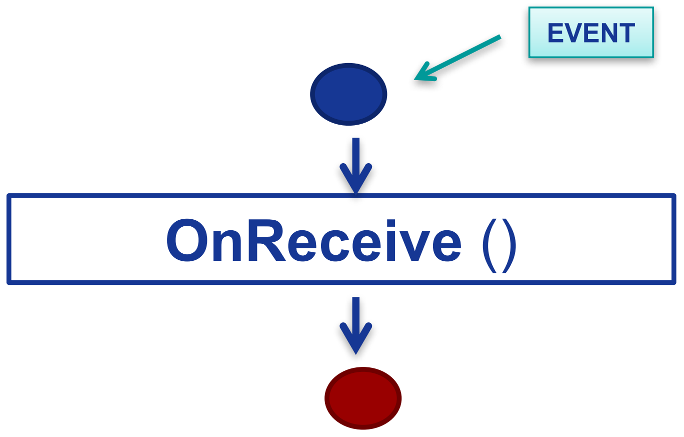
  1. Event → **Intent**
  2. Registration through **XML** code
  3. Registration through **Java** code

- **Handling** of the event.

Android: Broadcast Receiver

A **Broadcast Receiver** is a component that is activated only when specific events occur (i.e. SMS arrival, phone call, etc).

BROADCAST RECEIVER LIFETIME



- Single-state component …
- **onReceive**() is invoked when the registered event occurs
- After handling the event, the Broadcast Receiver is **destroyed**.

Android: Broadcast Receiver

➢**Registration** of the Broadcast Receiver to the event …
XML Code: → modify the **AndroidManifest**.xml

```
 <application>
     <receiver class=“SMSReceiver”>
         <intent-filter>
             <action  android:value=“android.provider.Telephony.SMS_RECEIVED” />
         </intent-filter>
     </receiver>
</application>
```

Android: Broadcast Receiver

➢**Registration** of the Broadcast Receiver to the event …

In **Java** → **registerReceiver**(BroadcastReceiver, IntentFilter)

```
receiver=new BroadcastReceiver() { … }

protected void onResume() {
    registerReceiver(receiver, new IntentFilter(Intent.ACTION_TIME_TICK));
}

protected void onPause() {
    unregisterReceiver(receiver);
}
```

Android: Broadcast Receiver

How to send the **Intents** handled by **Broadcast Receivers**?

- void **sendBroadcast**(Intent intent)
  … No order of reception is specified

- void **sendOrderedBroadcast**(Intent intent, String permit)
  … reception order given by the android:priority field

sendBroadcast() and startActivity() work on different contexts!