Université Mohammed V
FACULTE DES SCIENCES
RABAT / FSR
Département informatique

Master IAO
Master II–Semestre 3
Cours

Mobile & Cloud Computing

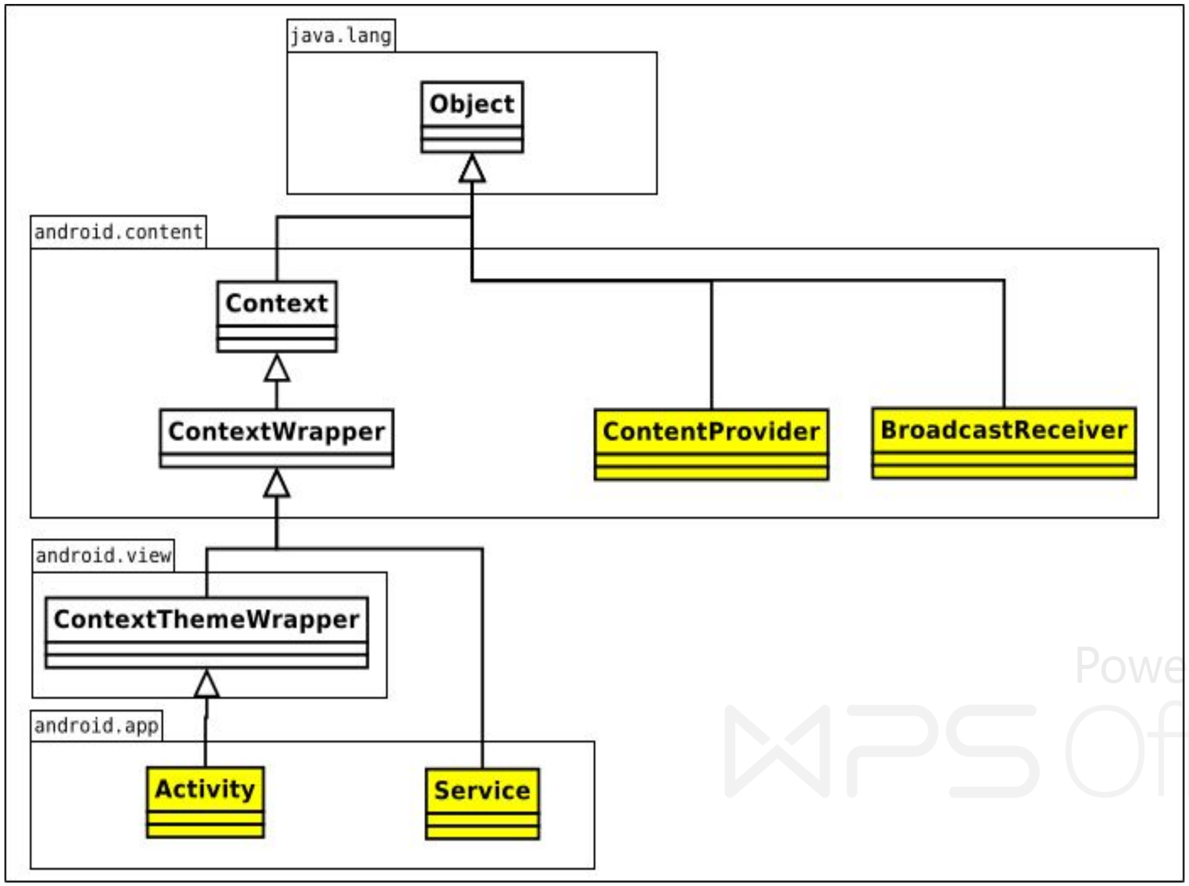
Pr. REDA Oussama Mohammed

2015/2016

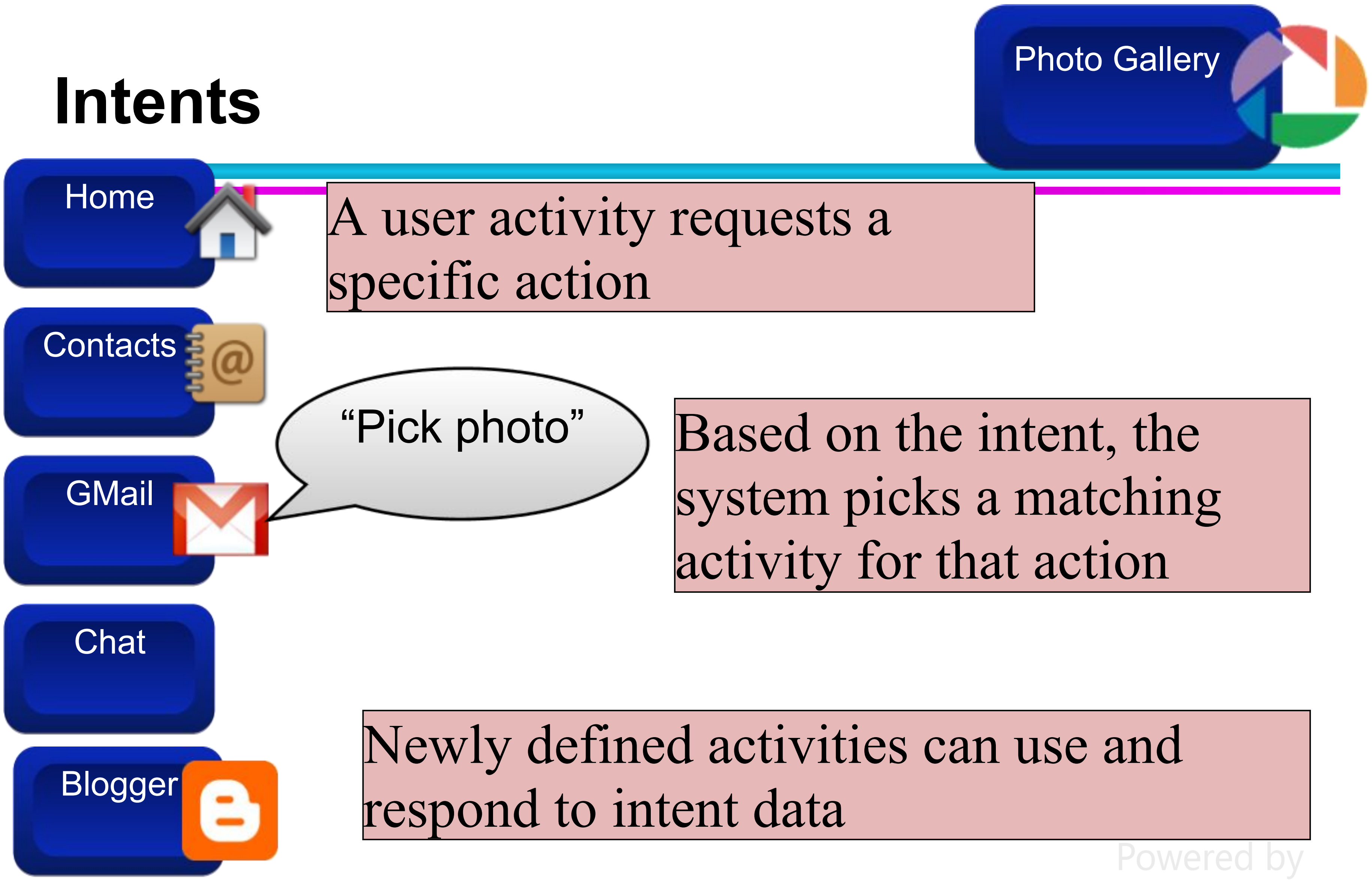
Android Services

Inter-Process Communications (IPC)

- Objective: reuse existing data and services among Android components



Intents



A user activity requests a specific action

Based on the intent, the system picks a matching activity for that action

Newly defined activities can use and respond to intent data

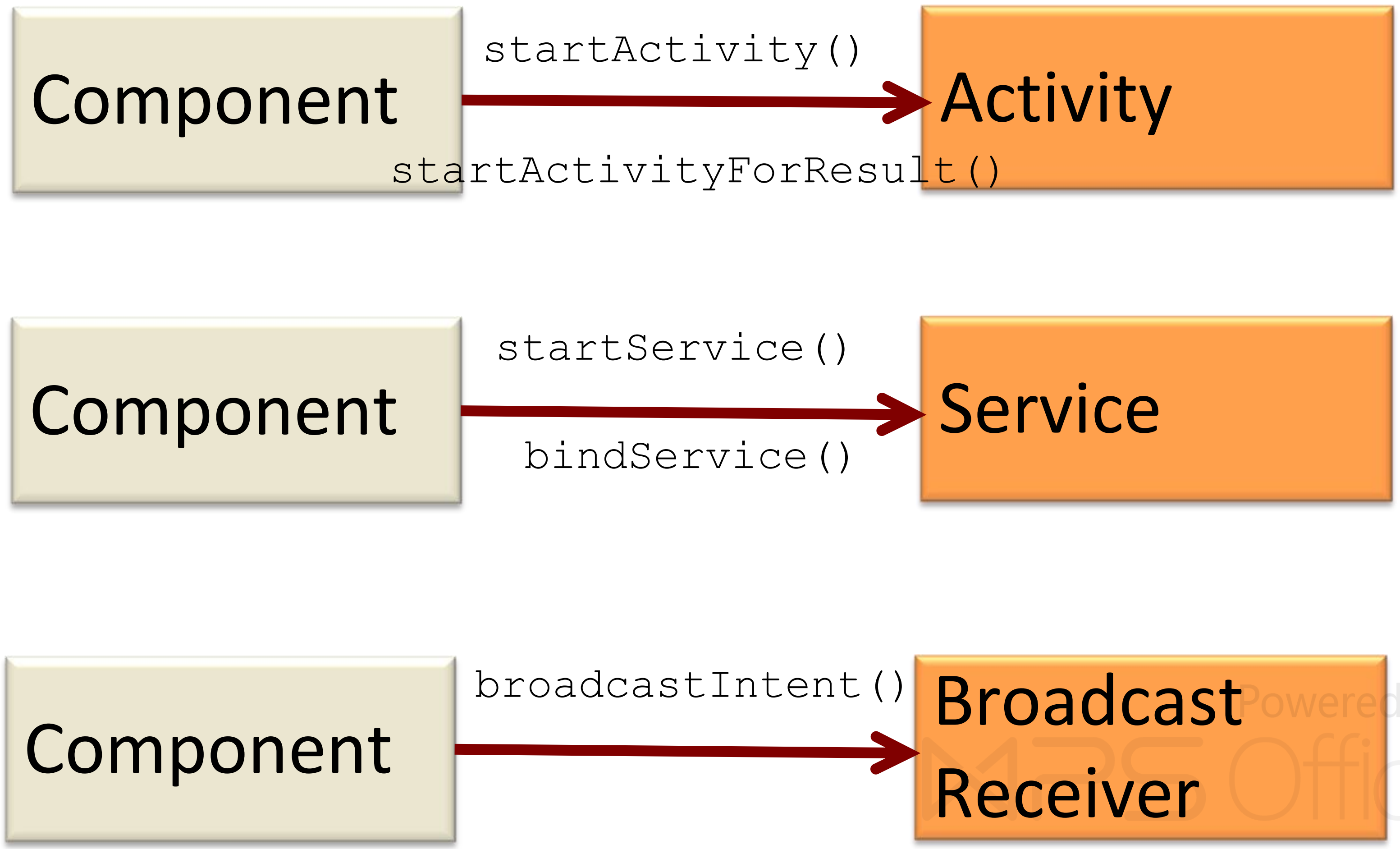
Definition: Inter-process message facility for launching/ communicating with system or application activities.

Intent

Intent: facility for late run-time binding between components in the same or different applications.

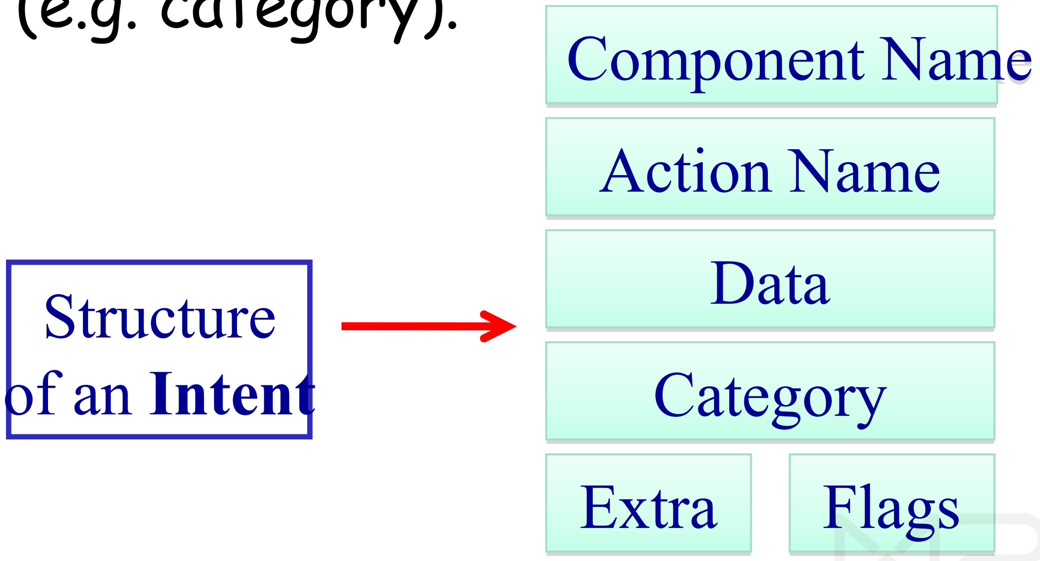
- **Call** a component from another component
- Possible to **pass data** between components
- Something like:
  - "Android, please do **that** with **this** data"
- **Reuse** already installed applications
- Components: ***Activities***, ***Services***, *Broadcast receivers* …

Inter-Process Communications (IPC)



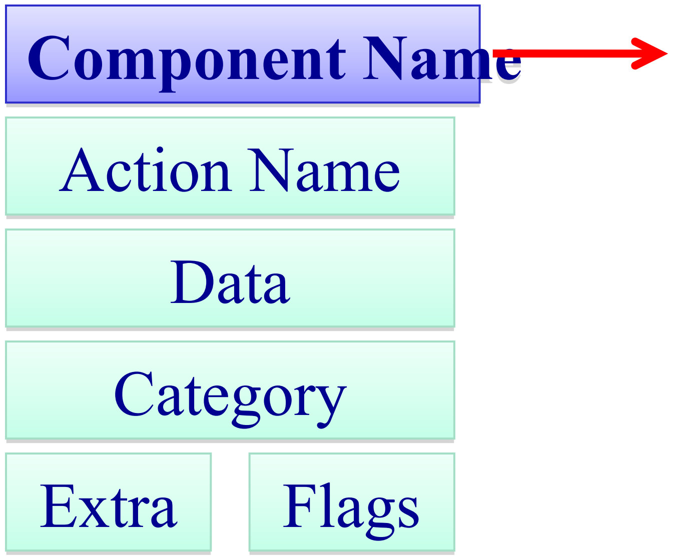
Intent

- We can think to an "**Intent**" object as a **message** containing a bundle of information.
  - Information of interests for the receiver (e.g. data)
  - Information of interests for the Android system (e.g. category).



Intent

- We can think to an “**Intent**” object as a **message** containing a bundle of information.
  - Information of interests for the receiver (e.g. data)
  - Information of interests for the Android system (e.g. category).



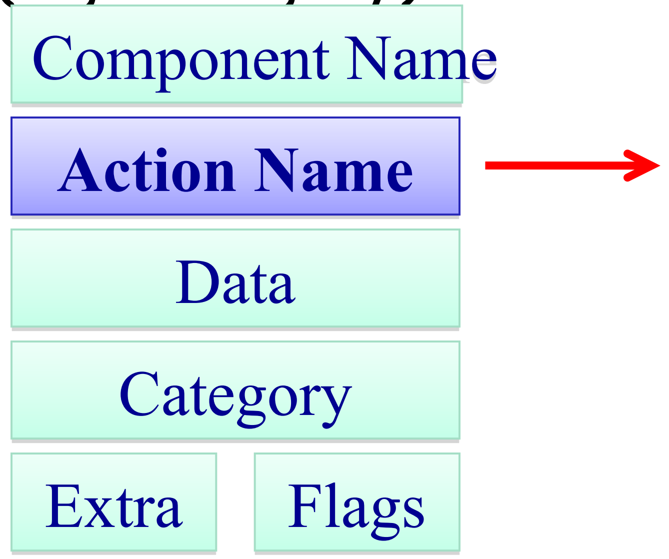
Component that should handle the intent (i.e. the **receiver**).

It is **optional** (implicit intent)/ **necessary** (explicit intent)

void **setComponent**()

Intent

- We can think to an "**Intent**" object as a **message** containing a bundle of information.
  - Information of <u>interests for the receiver</u> (e.g. data)
  - Information of <u>interests for the Android system</u> (e.g. category).



A string naming the **action** to be performed.

Pre-defined, or can be specified by the programmer.

void **setAction**()

Intent

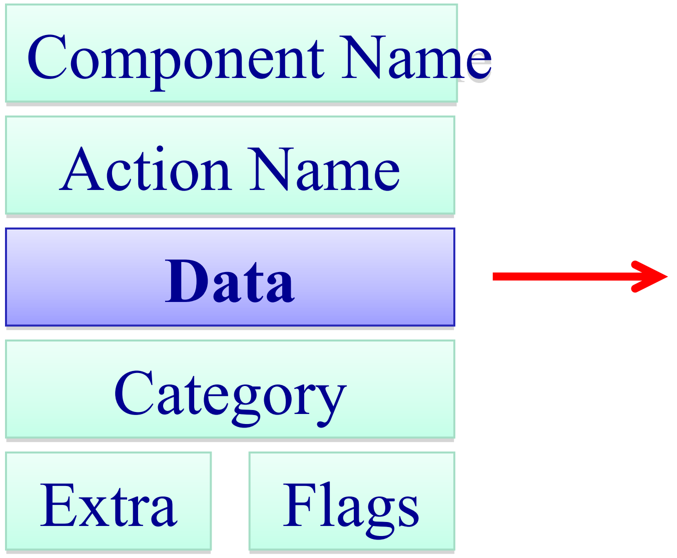
- **Predefined actions** (http://developer.android.com/reference/android/content/Intent.html)

Action Name	Description
ACTION_CALL	Initiate a phone call
ACTION_EDIT	Display data to edit
ACTION_MAIN	Start as a main entry point, does not expect to receive data.
ACTION_PICK	Pick an item from the data, returning what was selected.
ACTION_VIEW	Display the data to the user
ACTION_SEARCH	Perform a search

- **Defined** by the programmer
  - it.example.projectpackage.FILL_DATA (package prefix + name action)

Intent

- We can think to an "**Intent**" object as a **message** containing a bundle of information.
  - Information of <u>interests for the receiver</u> (e.g. data)
  - Information of <u>interests for the Android system</u> (e.g. category).



Data passed from the caller to the called Component.

Location of the data (**URI**) and Type of the data (**MIME** type)

void **setData**()

Intent

- Each data is specified by a **name** and/or **type**.
- **name**: Uniform Resource Identifier (**URI**)
- **scheme://host:port/path**

EXAMPLEs

- content://com.example.project:200/folder
- content://contacts/people
- content://contacts/people/1

Intent

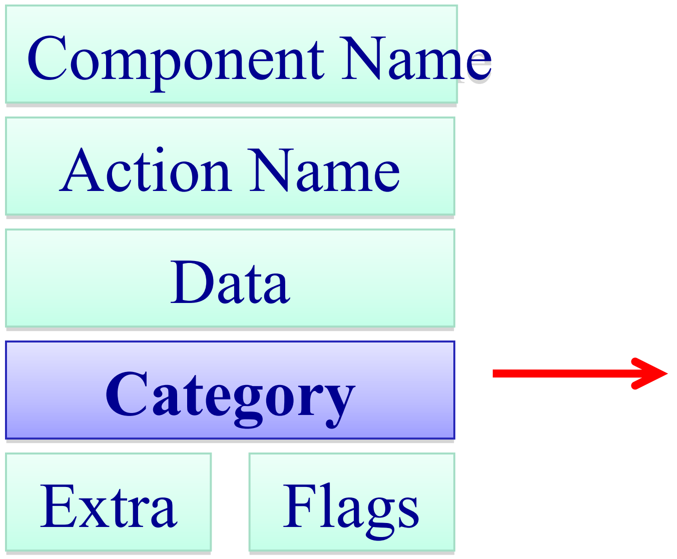
- Each data is specified by a **name** and/or **type**.
- **type**: **MIME** (Multipurpose Internet Mail Extensions)-type
- Composed by two parts: a <u>type</u> and a <u>subtype</u>

EXAMPLEs

Image/gif	image/jpeg	image/png	image/tiff
text/html	text/plain	text/javascript	text/css
video/mp4	video/mpeg4	video/quicktime	video/ogg
application/vnd.google-earth.kml+xml			

Intent

- We can think to an "**Intent**" object as a **message** containing a bundle of information.
  - Information of interests for the receiver (e.g. data)
  - Information of interests for the Android system (e.g. category).



A string containing information about the **kind of component** that should handle the Intent.

> 1 can be specified for an Intent

void **addCategory**()

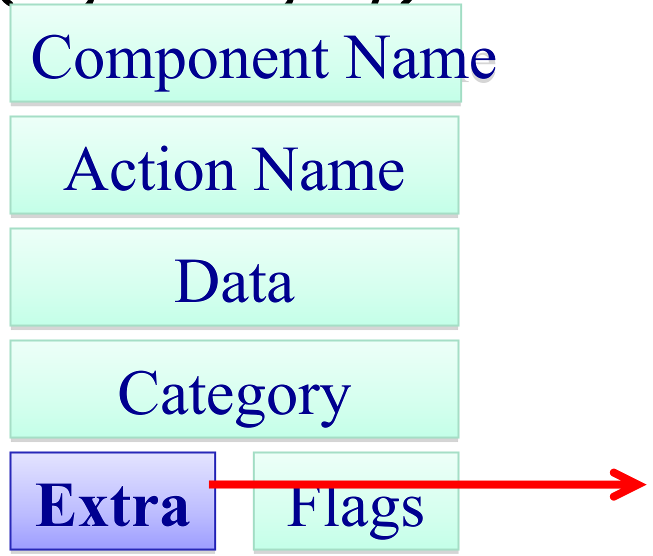
Intent

- **Category**: string describing the **kind of component** that should handle the intent.

Category Name	Description
CATEGORY_HOME	The activity displays the HOME screen.
CATEGORY_LAUNCHER	The activity is listed in the top-level application launcher, and can be displayed.
CATEGORY_PREFERENCE	The activity is a preference panel.
CATEGORY_BROWSABLE	The activity can be invoked by the browser to display data referenced by a link.

Intent

- We can think to an "**Intent**" object as a **message** containing a bundle of information.
  - Information of interests for the receiver (e.g. data)
  - Information of interests for the Android system (e.g. category).



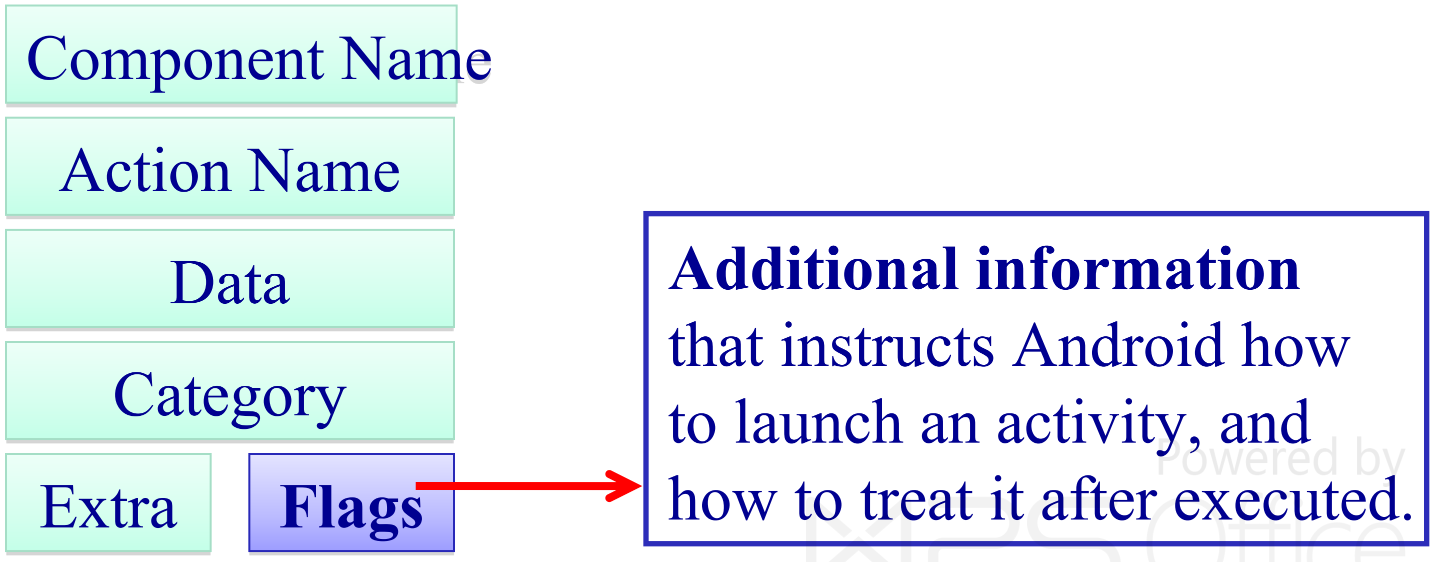
Additional information that should be delivered to the handler(e.g. parameters).

Key-value pairs

void **putExtras()**
getExtras()

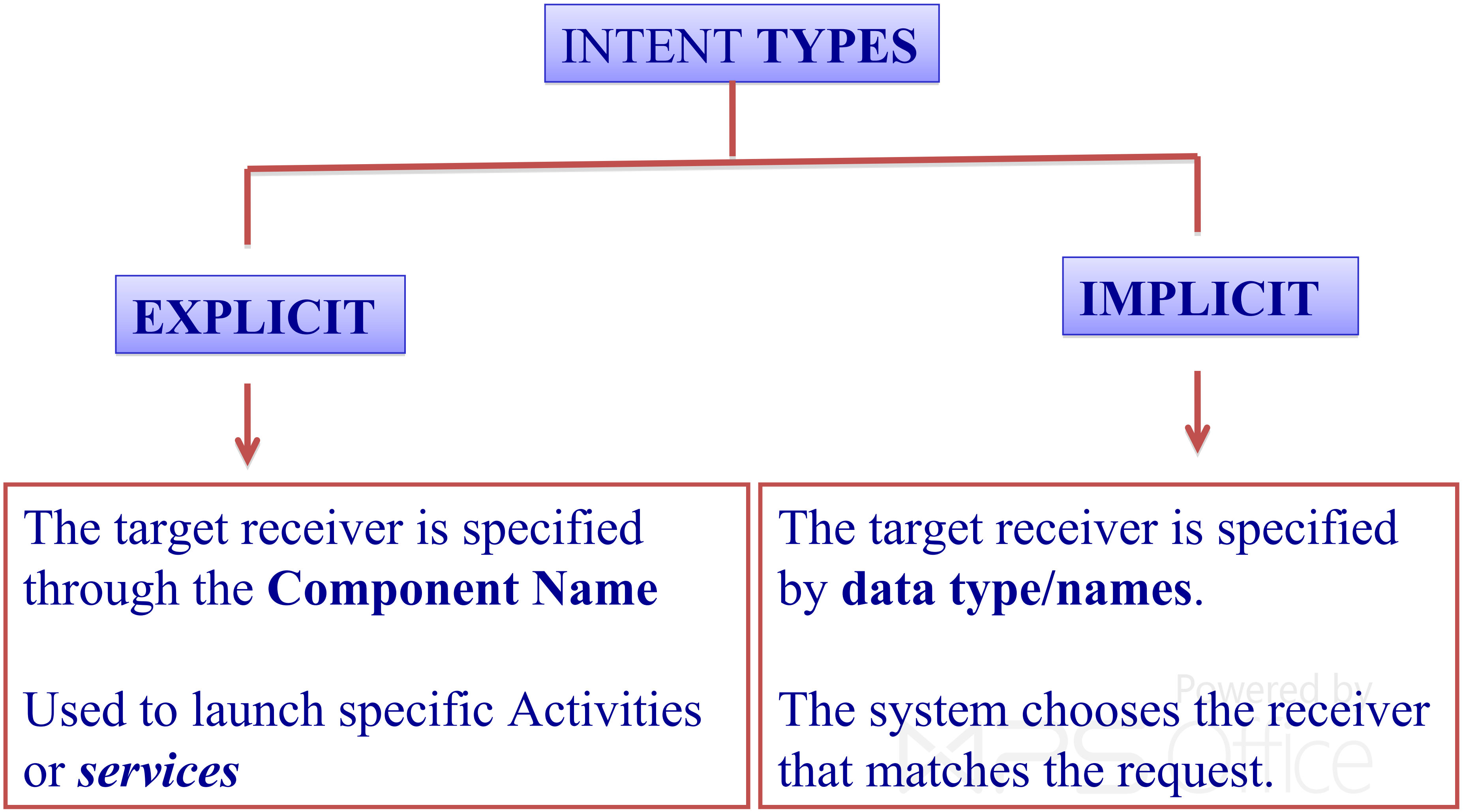
Intent

- We can think to an "**Intent**" object as a **message** containing a bundle of information.
  - Information of <u>interests for the receiver </u>(e.g. data)
  - Information of <u>interests for the Android system</u> (e.g. category).



Additional information that instructs Android how to launch an activity, and how to treat it after executed.

Intent types



The target receiver is specified through the **Component Name**

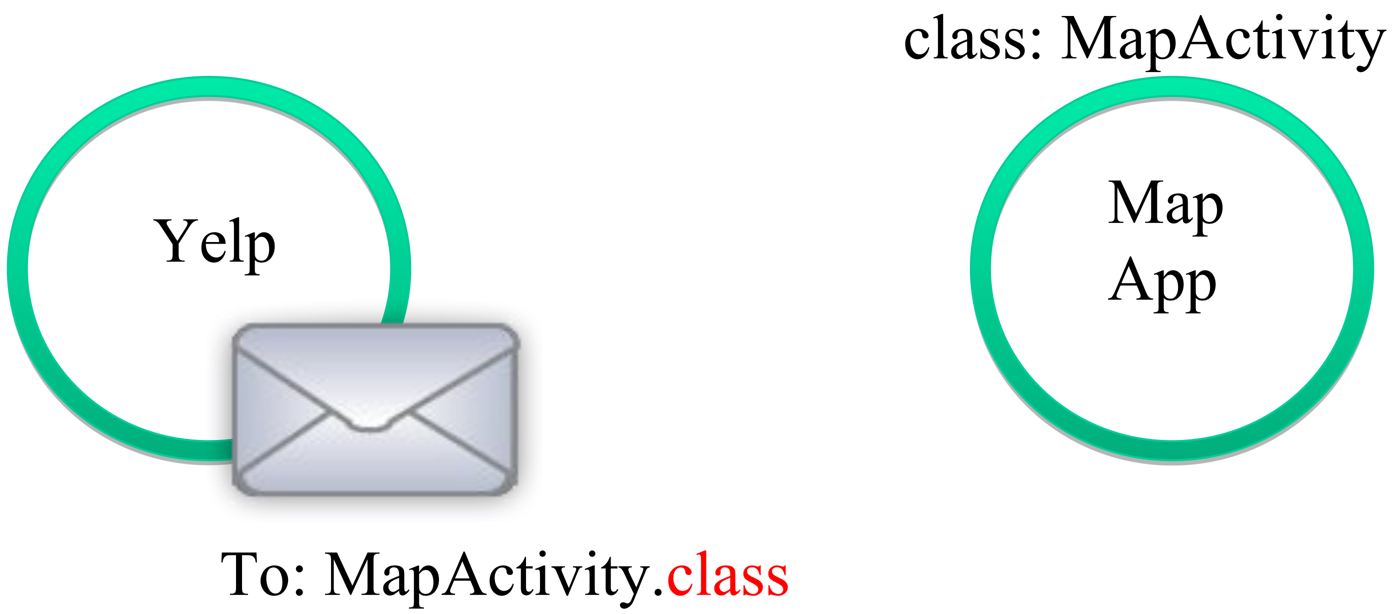
Used to launch specific Activities or ***services***

The target receiver is specified by **data type/names**.

The system chooses the receiver that matches the request.

17

Explicit Intent



Only the specified activity receives this message

http://developer.android.com/training/basics/firstapp/starting-activity.html

Intent types:

- **Explicit** Intent: Specify the activity that will handle the intent.

```
Intent intent=new Intent(this, SecondActivity.class);
startActivity(intent);
```

```
Intent intent=new Intent();
ComponentName component=new
ComponentName(this,SecondActivity.class);
intent.setComponent(component);
startActivity(intent);
```

Services

Services

- A service does not have a visual user interface
- Runs in the background for an indefinite period
  - Eg service might play background audio as user does something else.
  - Might fetch data over the network
  - Calculate something
  - Provide a result to an activity
- Each service extends the *Service base class*
- Services run in the main thread of the application process.
  - Don't block other components or user interface
  - Often spawn another thread for time consuming tasks

Services

- Services are like Activities, but without a UI
- Services are not intended as background threads
  - Think of an audio player where the audio keeps playing while the user looks for more audios to play or uses other apps
  - Don't think of a cron job (e.g. run every day at 3am), use Alarms to do this
- Several changes in 2.0 related to Services
  - See http://android-developers.blogspot.com/2010/02/service-api-changes-starting-with.html

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 ended when it loses visibility
- **Service** → No UI, ended when it terminates or when it is terminated by other components

A Service provides a robust environment for background tasks …

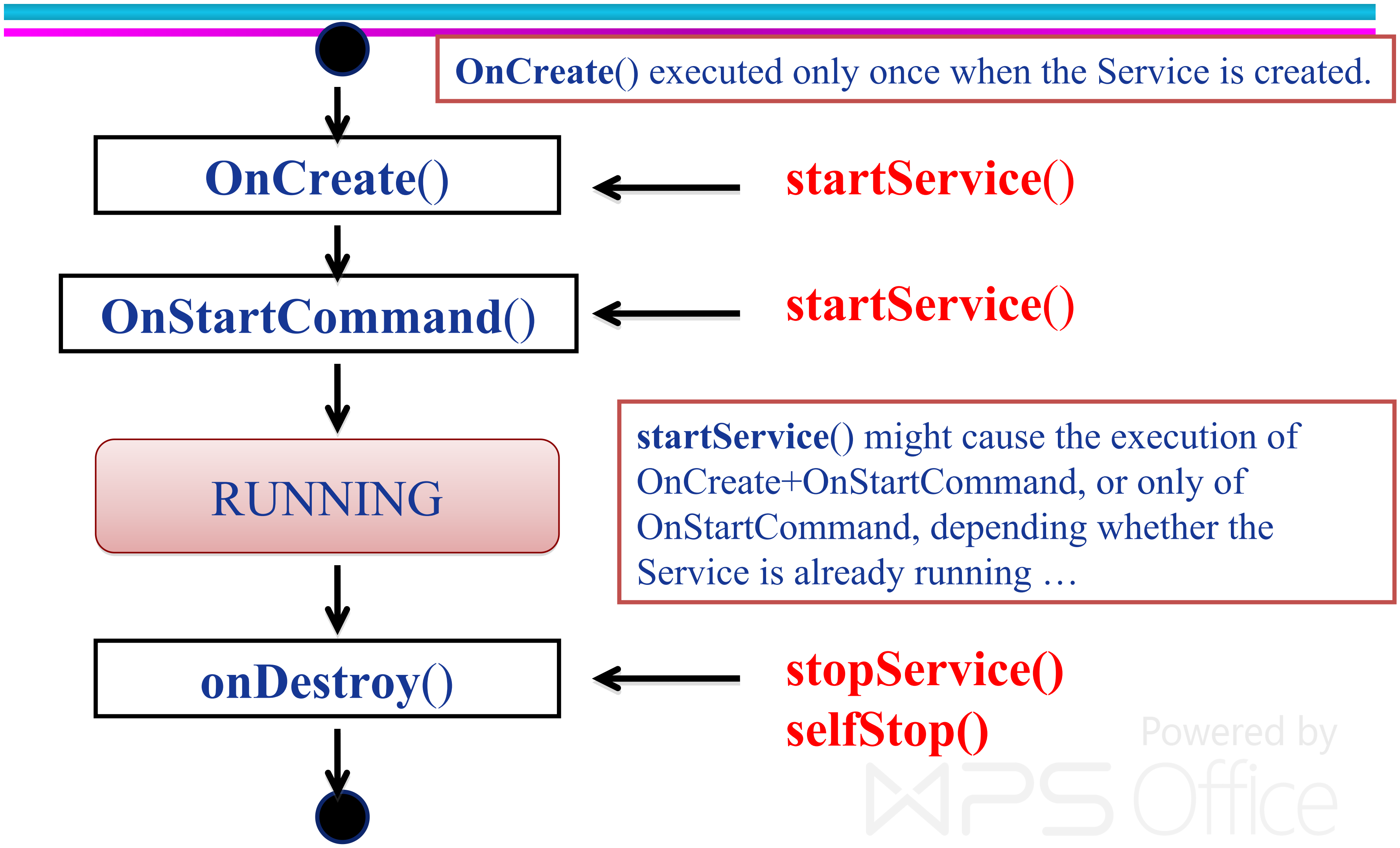
Android: Services

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

- Once started, a Service runs in **background** indefinetely, 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)

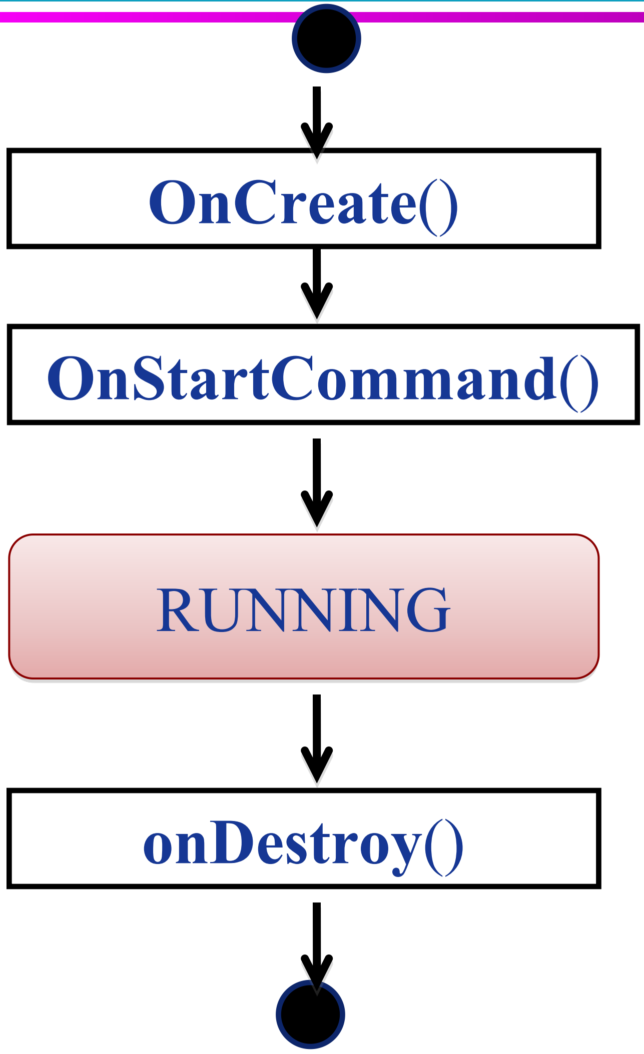
Android: Service Lifetime



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: Service Lifetime



Two Types of **Services:**

1. **Local** Services: Start-stop lifecycle as the one shown.

2. **Remote/Bound** Services: Bound to application components.
Allow interactions with them, send requests, get results, IPC facilities.

Services

- A Service is an application component that runs in the background, not interacting with the user, for an **indefinite** period of time.
- A Service is **not** a separate process. The Service object itself does not imply it is running in its own process; unless otherwise specified, it runs in the same process as the application it is part of. A Service is **not** a thread. It is not a means itself to do work off of the main thread (to avoid Application Not Responding errors).
- Higher priority than inactive Activities, so less likely to be killed
  - If killed, they can be configured to re-run automatically (when resources available)

Services

- If a service and the respective activity are run on the same thread, then the activity will become unresponsive when the service is being executed for long running operations.

- Each service class must have a corresponding **<service>** declaration in its package's **AndroidManifest.xml**

```
<service android:name=".MyService" />
```

Services

- Services can be started with **Context.startService()** and **Context.bindService()** in the main thread of the application's process.

  CPU intensive tasks must be offloaded to background threads using Thread or AsyncTask

  startService(**new** Intent(getBaseContext(), MyService.**class**));

  startService(**new** Intent("net.learn2develop.MyService"));

- To stop a service:
  stopService(**new** Intent(getBaseContext(), MyService.**class**)) or stopSelf()

- Alarms can be used to fire Intents at set times. These can start services, open Activities, or broadcast Intents

- The written services class should extend Service class and has three methods

  public IBinder onBind(Intent arg0) { ... }

  public int onStartCommand(Intent intent, **int** flags, **int** startId) { ... }

  public void onDestroy() { ... }

Creating a Service

- Subclass Service, then override:

onStartCommand() -- called when startService() is called. Then you can call stopSelf() or stopService()

onBind() -- called when bindService() is called. Returns an IBinder (or null if you don't want t o be bound).

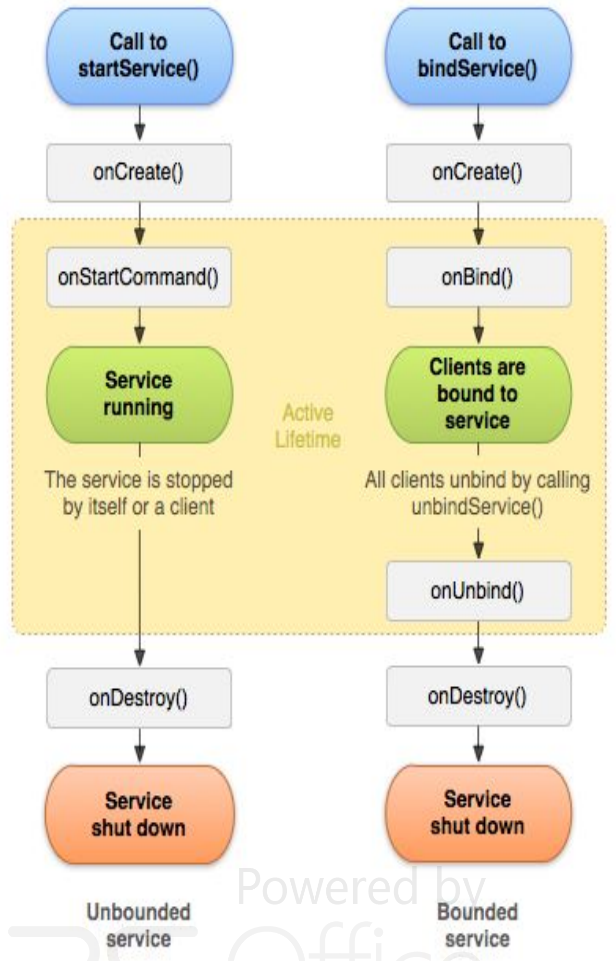
onCreate() -- called before above methods.

onDestroy() -- called when about to be shutdown.

- There are two classes you can subclass:

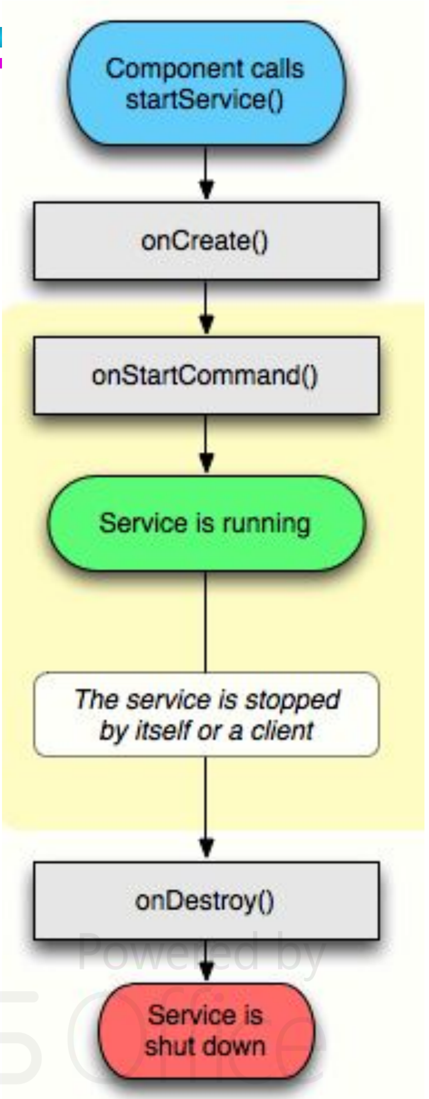
Service: you need to create a new thread, since it is not created by default.

IntentService. This uses a worker thread to perform the requests, and all you need to do is override onHandleIntent. This is the easiest, provided you don't need to handle multiple requests.



Services Lifecycle

- A Service has three lifecycle methods:
  1. void onCreate()
  2. void onStartCommand()
  3. void onDestroy()
- onStartCommand() is invoked when a service is explicitly started using startService() method
- onDestroy() is invoked when a service is stopped using stopService() method



Creating a Service

Started Services

Creating a Service (*Started Service*)

```
import  android.app.Service;
import  android.content.Intent;
import  android.os.IBinder;
public class MyService extends Service {
        @Override
         public void onCreate() {
                  // TODO: Actions to perform when service is created.
         }
     @Override
     public IBinder onBind(Intent intent) {
                  // TODO: Replace with service binding implementation.
        return  null; }
```

Creating a Service

```
@Override
  public int onStartCommand(Intent intent, int flags, int startId) {
            // TODO Launch a background thread to do processing.
  return  Service.START_STICKY;
  }

 @Override
 public void onDestroy () {
            // TODO: Actions to perform when service is ended.
  }
}
```

onStartCommand

- **Called whenever the Service is started with startService call**
  - So beware: may be executed several times in Service's lifetime!
  - Controls how system will respond if Service restarted
  - (START_STICKY) means the service will run indefinitely until explicitly stopped
  - Run from main GUI thread, so standard pattern is to create a new Thread from onStartCommand to perform processing and stop Service when complete

Services

onStartCommand() returns a flag which tells the OS that the service is either sticky or not_sticky.

Both codes are only relevant when the phone runs out of memory and kills the service before it finishes executing.

- START_STICKY tells the OS to recreate the service after it has enough memory and call onStartCommand() again with a null intent.
- START_NOT_STICKY tells the OS to not bother recreating the service again. There is also a third code
- START_REDELIVER_INTENT that tells the OS to recreate the service AND redeliver the same intent to onStartCommand().

Example

- Add to AndroidManifest.xml

```
<service android:enabled="true" android:name=".MyService"></service>
```

- Create the Service class

```
public class MyService extends Service {
    @Override
    public void onCreate() {
    }
    @Override
    public void onStartCommand(Intent intent, int startId) {
        //do something
    }
    @Override
    public IBinder onBind(Intent intent) {
       return null;
    }
}
```

Start/Stop the Service

- **Client code**

```
public class MyActivity extends Activity {

                    ....
        startService(new Intent(this, MyService.class));
                    ....
        stopService(new Intent(this, MyService.class));
                    ....

  }
```

Services using IntentService class

- To easily create a service that runs a task asynchronously and terminates itself when it is done, you can use the IntentService class
- The IntentService class is a base class for Service that handles asynchronous requests on demand
- It is started just like a normal service; and it executes its task within a worker thread and terminates itself when the task is completed
- // Create a class that extends IntentService class instead of Service class
- **public class** MyIntentService **extends IntentService** { }
- // create a constructor and call superclass with the name of the intent service as a string
- **public** MyIntentService() { **super**("MyIntentServiceName"); }
- // onHandleIntent() is executed on a worker thread
- **protected void onHandleIntent(Intent intent) { … }**

Services using IntentService class

- The IntentService class does the following:
- Creates a default worker thread that executes all intents delivered to ***onStartCommand()*** separate from your application's main thread.
- Creates a work queue that passes one intent at a time to your ***onHandleIntent()*** implementation, so you never have to worry about multi-threading.
- Stops the service after all start requests have been handled, so you never have to call ***stopSelf()***.
- Provides 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.
- All you have to do is handle onHandleIntent().

Services using IntentService class

```
public class HelloIntentService extends IntentService {

  // A constructor is required, and must call the super IntentService(String)
  // constructor with a name for the worker thread.

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

 // The IntentService calls this method from the default worker thread
with the intent that started
   // the service. When this method returns, IntentService stops the service,
as appropriate.
```

Services using IntentService class

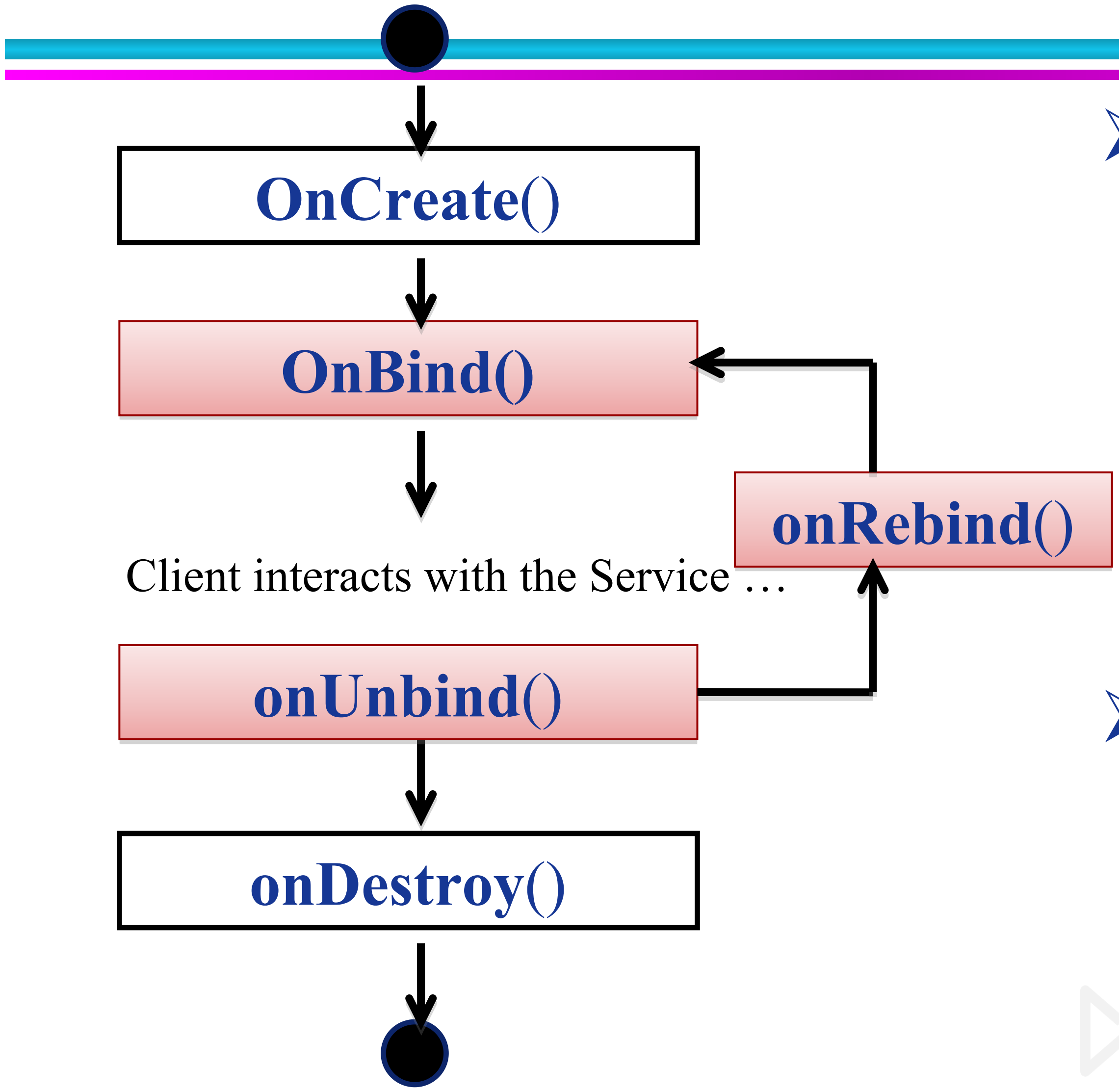
```
@Override
protected void onHandleIntent(Intent intent) {
    // Normally we would do some work here, like download a file.
    // For our sample, we just sleep for 5 seconds.
    long endTime = System.currentTimeMillis() + 5*1000;
    while (System.currentTimeMillis() < endTime) {
        synchronized (this) {
                    try {
                        wait(endTime - System.currentTimeMillis());
            } catch (Exception e) {    }
        }
      }
    }
}
```

Services

- Declare services in manifest
- Service Lifecycle:
  - onCreate, onStartCommand, onDestroy
- Can start services by passing in an intent similar to starting an activity
- Must stop service before starting up another instance
  - Best to start service in onCreate/onResume and stop in onPause

Bound Services

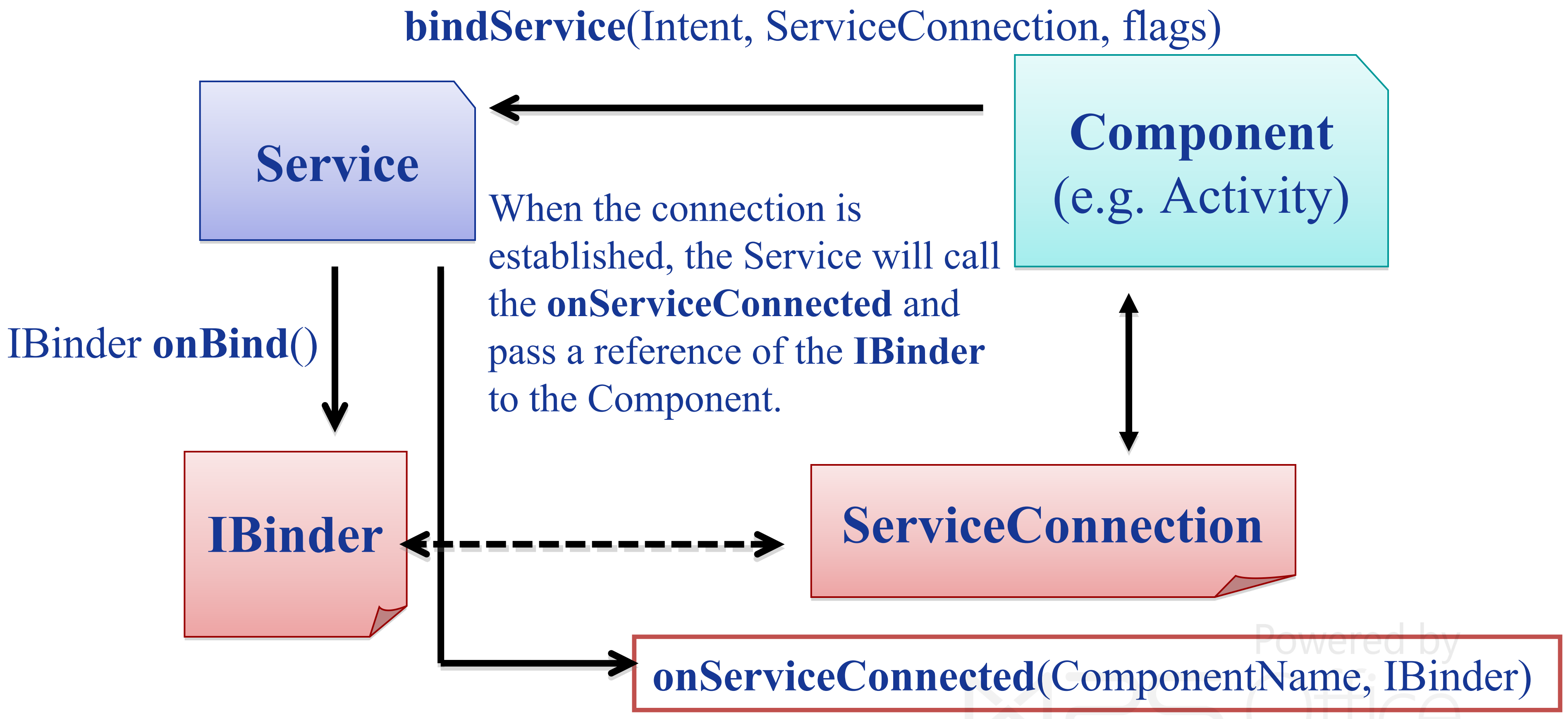
Android: Bound Service



- A **Bound** Service allows components (e.g. Activity) to **bind** to the services, **send** requests, **receive** response.
- A **Bound** Service can serve components running on different processes (**IPC**).

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 Messenger**
   - Allow access to a Service (*in a different process*) from different applications.

Android: Bound Service

IBinder

➢ Base interface for a remotable object, the core part of a lightweight remote procedure call mechanism designed for high performance when performing in-process and cross-process calls. This interface describes the abstract protocol for interacting with a remotable object. Do not implement this interface directly, instead extend from Binder.

Binder

➢ Base class for a remotable object, the core part of a lightweight remote procedure call mechanism defined by IBinder. This class is an implementation of IBinder that provides standard local implementation of such an object.

Bound Services

Local service

Android: Bound Service

```
public class LocalService extends Service {
        // Binder given to clients
        private final IBinder sBinder=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

ServiceConnection

Interface for monitoring the state of an application service. the methods on this class are called from the main thread of your process.
public abstract void onServiceConnected (ComponentName name, IBinder service)

Called when a connection to the Service has been established, with the IBinder of the communication channel to the Service.

Parameters

name The concrete component name of the service that has been connected.

service The IBinder of the Service's communication channel, which you can now make calls on.

Android: Bound Service

ServiceConnection

Interface for monitoring the state of an application service. the methods on this class are called from the main thread of your process.
public abstract void onServiceDisconnected (ComponentName name)

Called when a connection to the Service has been lost. This typically happens when the process hosting the service has crashed or been killed. This does not remove the ServiceConnection itself -- this binding to the service will remain active, and you will receive a call to onServiceConnected(ComponentName, IBinder) when the Service is next running.

Parameters

name The concrete component name of the service whose connection has been lost.

Android: Bound Service

```
public class MyActivity extends Activity {
            LocalService lService;

private ServiceConnection mConnection=new ServiceConnection() {
@Override
public void onServiceConnected(ComponentName arg0, IBinder service)
{
           SimpleBinder sBinder=(SimpleBinder) service;
           lService=sBinder.getService();
                                      ….
                        }
@Override
public void onServiceDisconnected(ComponentName arg0) {
                        }
                 };
```

Android: Bound Service

public abstract boolean bindService (Intent service, ServiceConnection conn, int flags)

The client (activity) calls **bindService** to bind to the service using **ServiceConnection**

```
.....
Intent intent = new Intent(this, LocalService.class);
bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
.....
```

•The first parameter of bindService() is an Intent that explicitly names the service to bind.

•The second parameter is the ServiceConnection object.

•The third parameter is a flag indicating options for the binding. It should usually be **BIND_AUTO_CREATE** in order to create the service if its not already alive. Other possible values are BIND_DEBUG_UNBIND and BIND_NOT_FOREGROUND, or 0 for none.

Bound Services

Remote service

Bound Services

Messenger

- Reference to a Handler, which others can use to send messages to it.
- This allows for the implementation of message-based communication across processes, by :

  creating a Messenger pointing to a Handler in one process;

  and handing that Messenger to another process

Note: the implementation underneath is just a simple wrapper around a Binder that is used to perform the communication

Bound Services

- Messenger(Handler target)

  Create a new Messenger pointing to the given *Handler*. Any Message objects sent through this Messenger will appear in the Handler as if Handler.sendMessage(Message) had been called directly.

- Messenger(IBinder target)

  Create a Messenger from a raw IBinder, which had previously been retrieved with *getBinder().*

- *IBinder* getBinder()

  Retrieve the IBinder that this Messenger is using to communicate with its associated Handler.

Bound Services

```
public class RemoteService extends Service {
            // Binder given to clients
     final  Messenger mMessenger = new Messenger(new IncomingHandler());

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

            class IncomingHandler extends Handler {
                 @Override
                        public void handleMessage( Message msg) {
                                    .....
                        }
            }
}
```

Bound Services

```
public class MyActivity extends Activity {
            Messenger mService;

private ServiceConnection mConnection=new ServiceConnection() {
@Override
public void onServiceConnected(ComponentName arg0, IBinder service)
{
                    mService = new Messenger(service);
                                    ….
                    }
@Override
public void onServiceDisconnected(ComponentName arg0) {
                        }
                };
```

Android Services

Université Mohammed V
FACULTE DES SCIENCES
RABAT / FSR
Département informatique

Master IAO
Master II–Semestre 3
Cours

Mobile & Cloud Computing

Pr. REDA Oussama Mohammed

2015/2016