Android Services & Local IPC: Communicate from Started Services to Activities via Broadcast Receivers

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

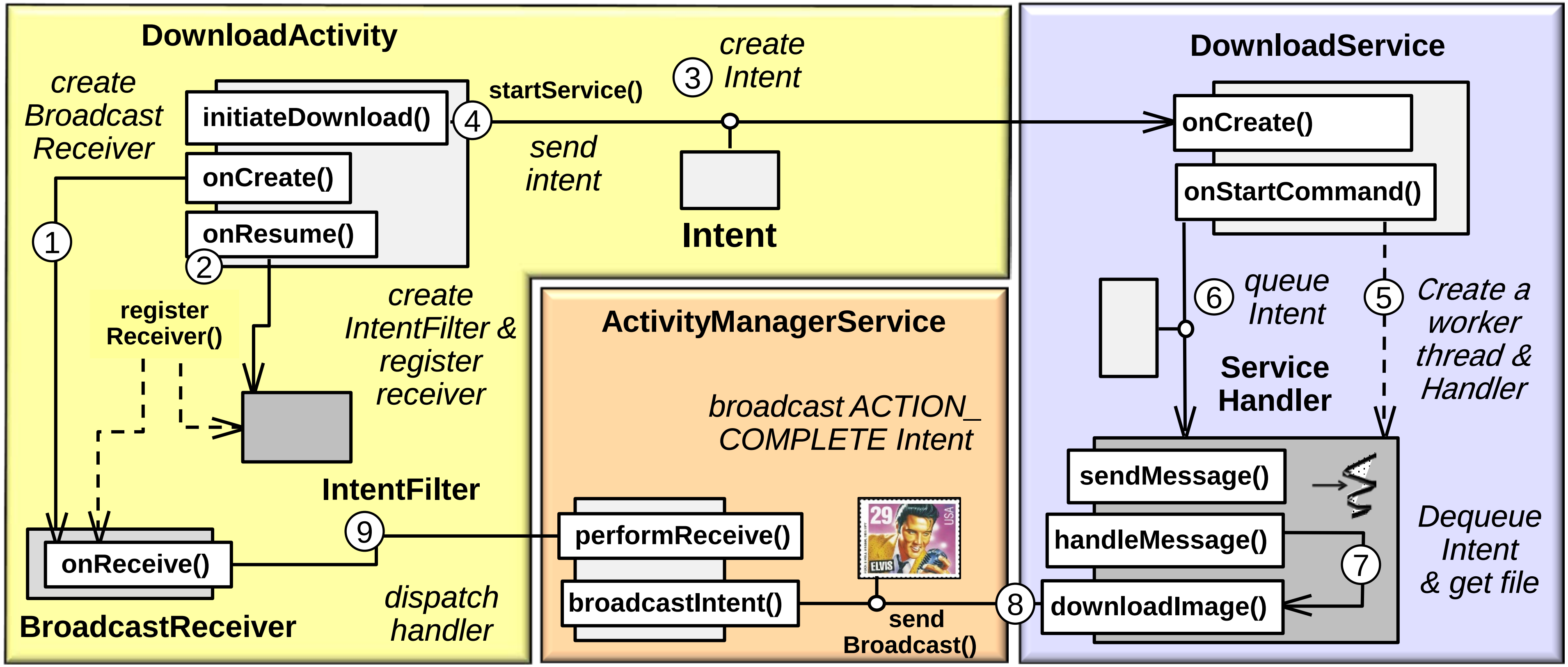
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

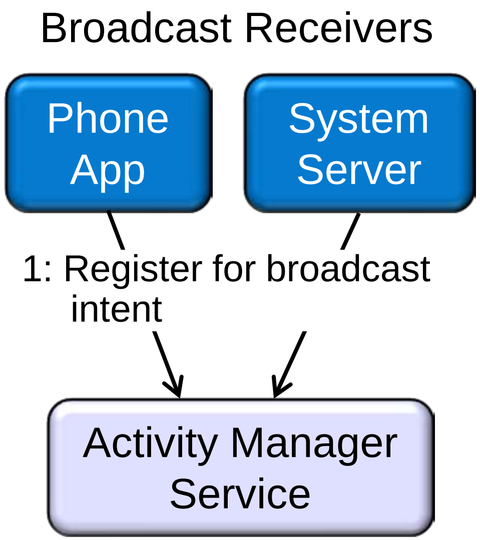
Learning Objectives in this Part of the Module

- Understand how to use Broadcast Receivers to communicate from Started Services back to their invoking Activities
  - Supports IPC with (multiple) remote processes without using AIDL



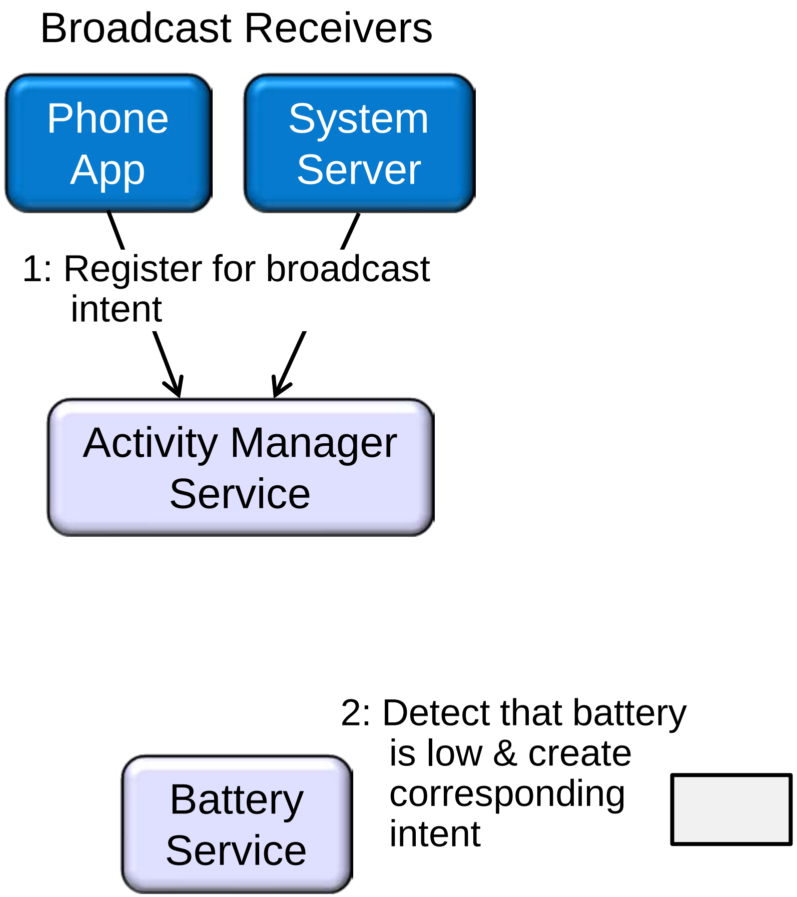
Overview of Broadcast Receivers

- BroadcastReceivers are components (*receivers*) that register for broadcast events & receive/react to the events



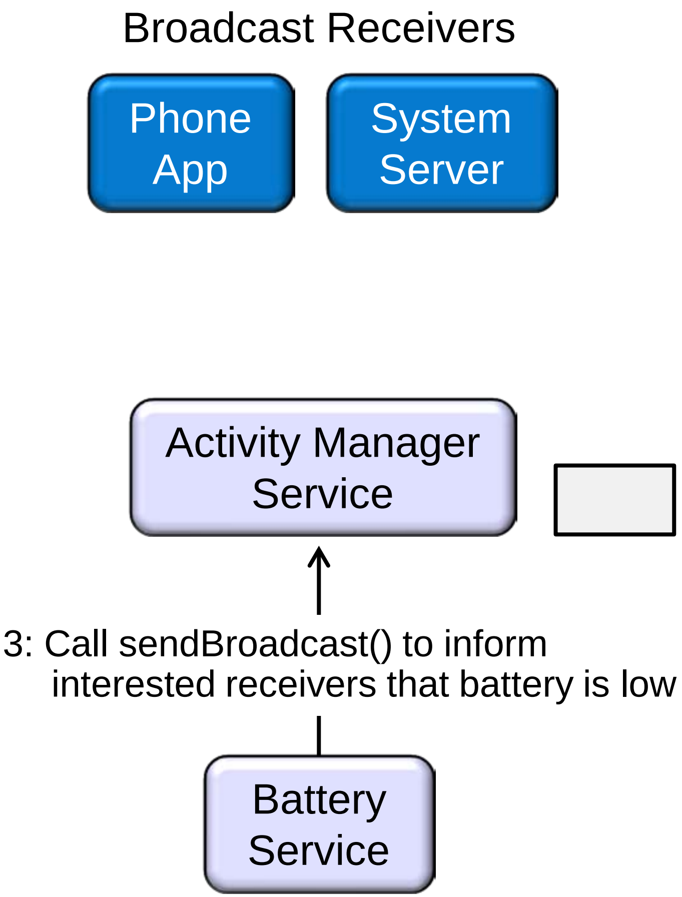
Overview of Broadcast Receivers

- BroadcastReceivers are components (*receivers*) that register for broadcast events & receive/react to the events
  - Events implemented as Intents


Broadcast Receivers
Phone App
System Server
1: Register for broadcast intent
Activity Manager Service
Battery Service
2: Detect that battery is low & create corresponding intent

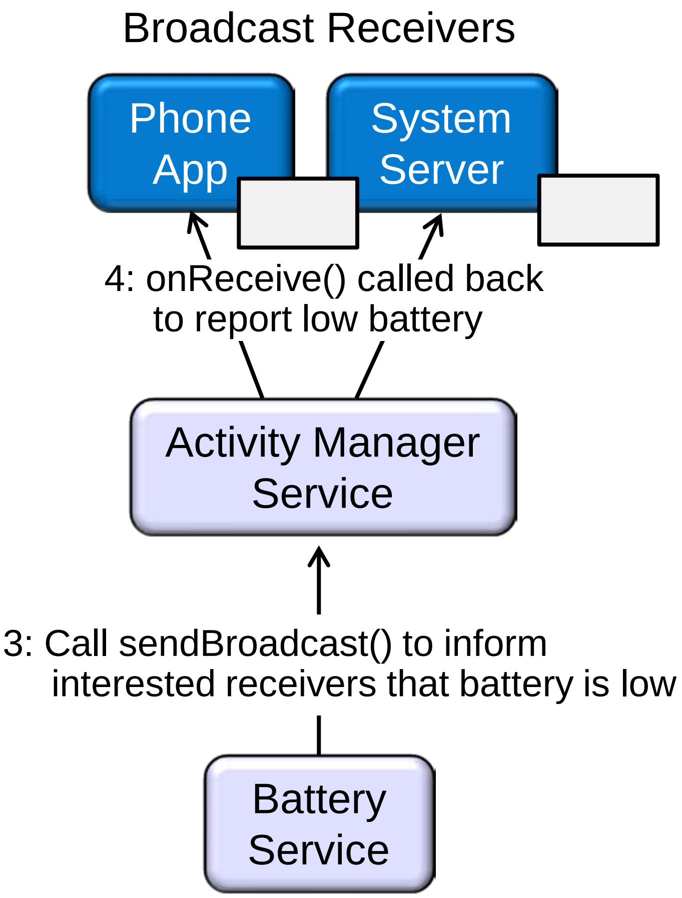
Overview of Broadcast Receivers

- BroadcastReceivers are components (*receivers*) that register for broadcast events & receive/react to the events
  - Events implemented as Intents
  - Events are broadcast system-wide



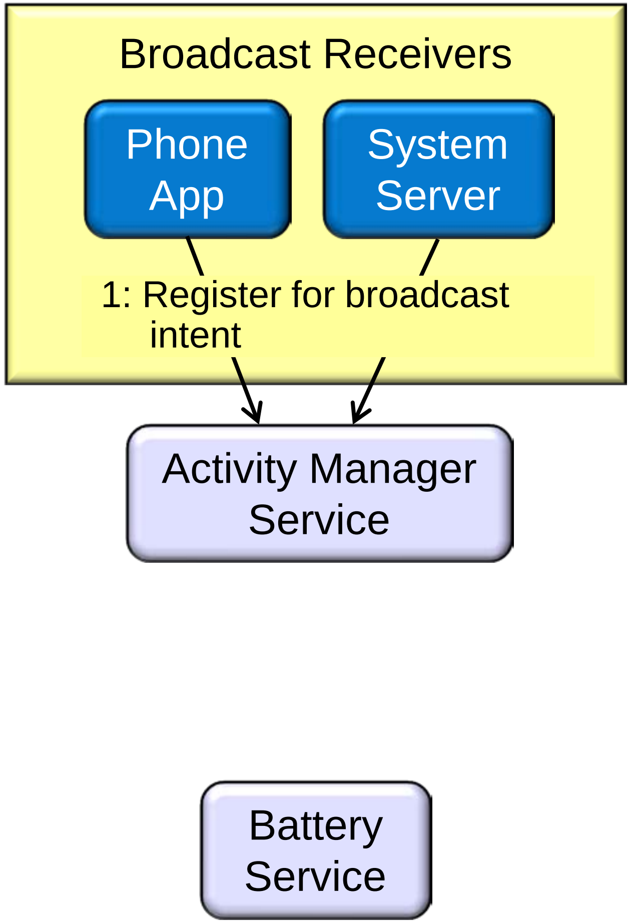
Overview of Broadcast Receivers

- BroadcastReceivers are components (*receivers*) that register for broadcast events & receive/react to the events
  - Events implemented as Intents
  - Events are broadcast system-wide
  - When an event occurs the Intents are disseminated to all matching receivers via their onReceive() hook methods


Broadcast Receivers
Phone App
System Server
4: onReceive() called back to report low battery
Activity Manager Service
3: Call sendBroadcast() to inform interested receivers that battery is low
Battery Service

Overview of Broadcast Receivers

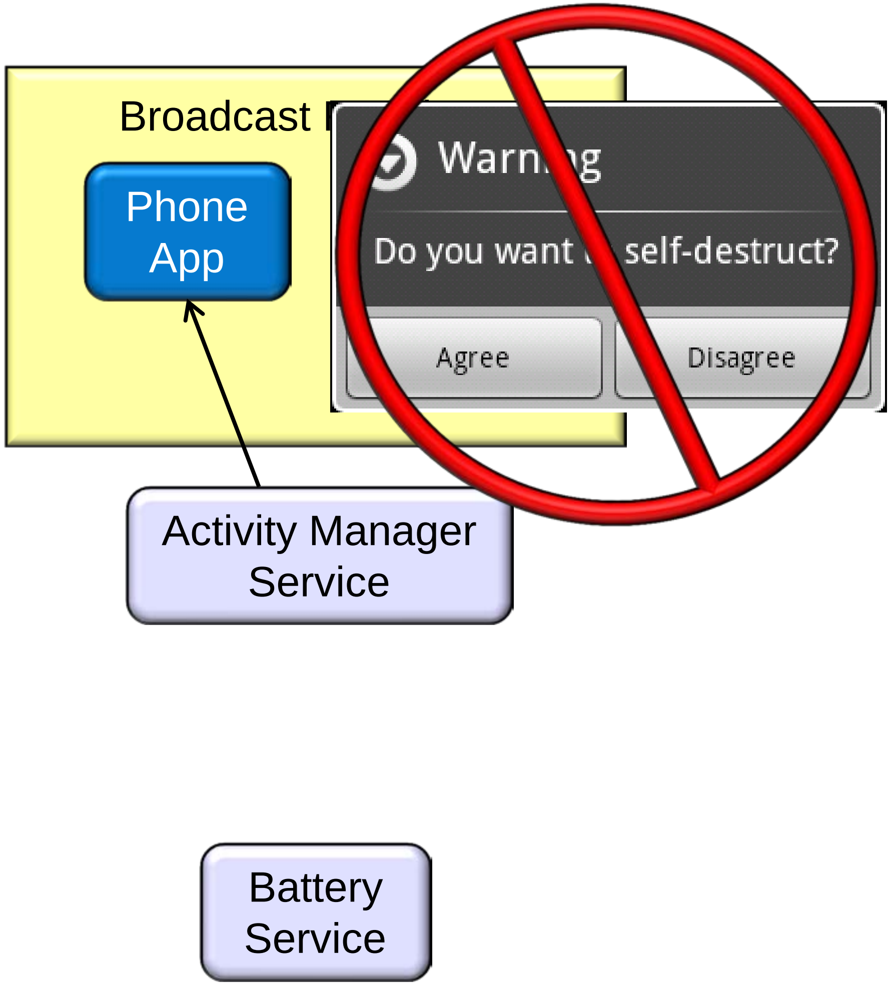
- BroadcastReceivers are components (*receivers*) that register for broadcast events & receive/react to the events
- Activities can create receivers that register for system or app events



www.vogella.com/articles/AndroidBroadcastReceiver/article.html has more

Overview of Broadcast Receivers

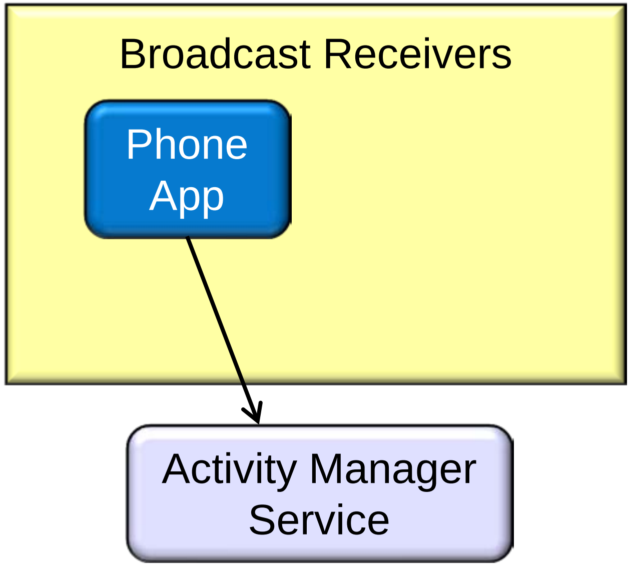
- BroadcastReceivers are components (*receivers*) that register for broadcast events & receive/react to the events
- Activities can create receivers that register for system or app events
- A receiver is restricted on what it can do when it handles an Intent
  - e.g., it may *not* show a dialog or bind to a service



developer.android.com/reference/android/content/BroadcastReceiver.html#ReceiverLifecycle

Overview of Broadcast Receivers

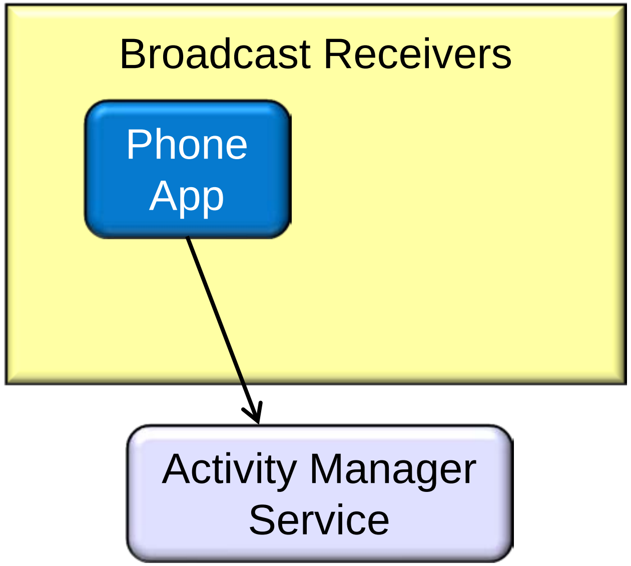
- BroadcastReceivers are components (*receivers*) that register for broadcast events & receive/react to the events
- Activities can create receivers that register for system or app events
- A receiver is restricted on what it can do when it handles an Intent
- Two ways to register a receiver:
  - Statically publish it via the <receiver> tag in the AndroidManifest.xml file



```
<receiver android:name="PhoneApp$NotificationBroadcastReceiver"
          exported="false">
  <intent-filter>
     <action android:name=
               "com.android.phone.ACTION_HANG_UP_ONGOING_CALL" />
     <action android:name=
               "com.android.phone.ACTION_SEND_SMS_FROM_NOTIFICATION"/>
  </intent-filter>
</receiver>
```

Overview of Broadcast Receivers

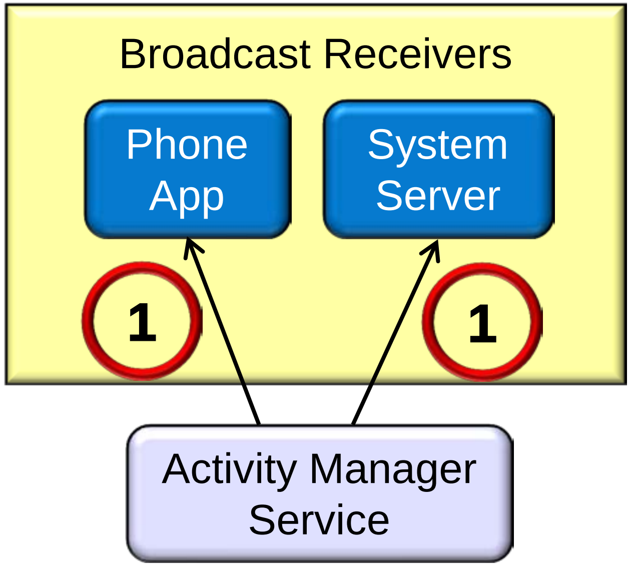
- BroadcastReceivers are components (*receivers*) that register for broadcast events & receive/react to the events
- Activities can create receivers that register for system or app events
- A receiver is restricted on what it can do when it handles an Intent
- Two ways to register a receiver:
  - Statically publish it via the <receiver> tag in the AndroidManifest.xml file
  - Dynamically register it with Context.registerReceiver()



```
final BroadcastReceiver mReceiver =
   new PhoneAppBroadcastReceiver();
...
IntentFilter intentFilter =
   new IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED);
...
registerReceiver(mReceiver, intentFilter);
```

Overview of Broadcast Receivers

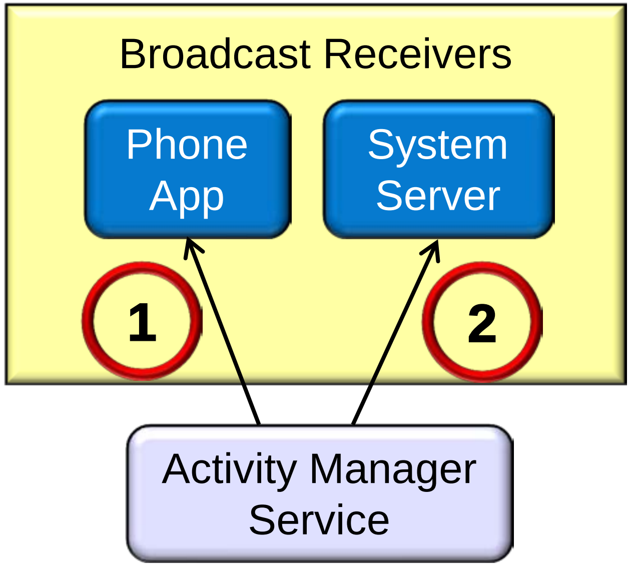
- BroadcastReceivers are components (*receivers*) that register for broadcast events & receive/react to the events
- Activities can create receivers that register for system or app events
- A receiver is restricted on what it can do when it handles an Intent
- Two ways to register a receiver
- Android supports several broadcast mechanisms
  - *Normal* – Sent with Context.sendBroadcast(), which is completely asynchronous



developer.android.com/reference/android/content/Context.html #sendBroadcast(android.content.Intent)

Overview of Broadcast Receivers

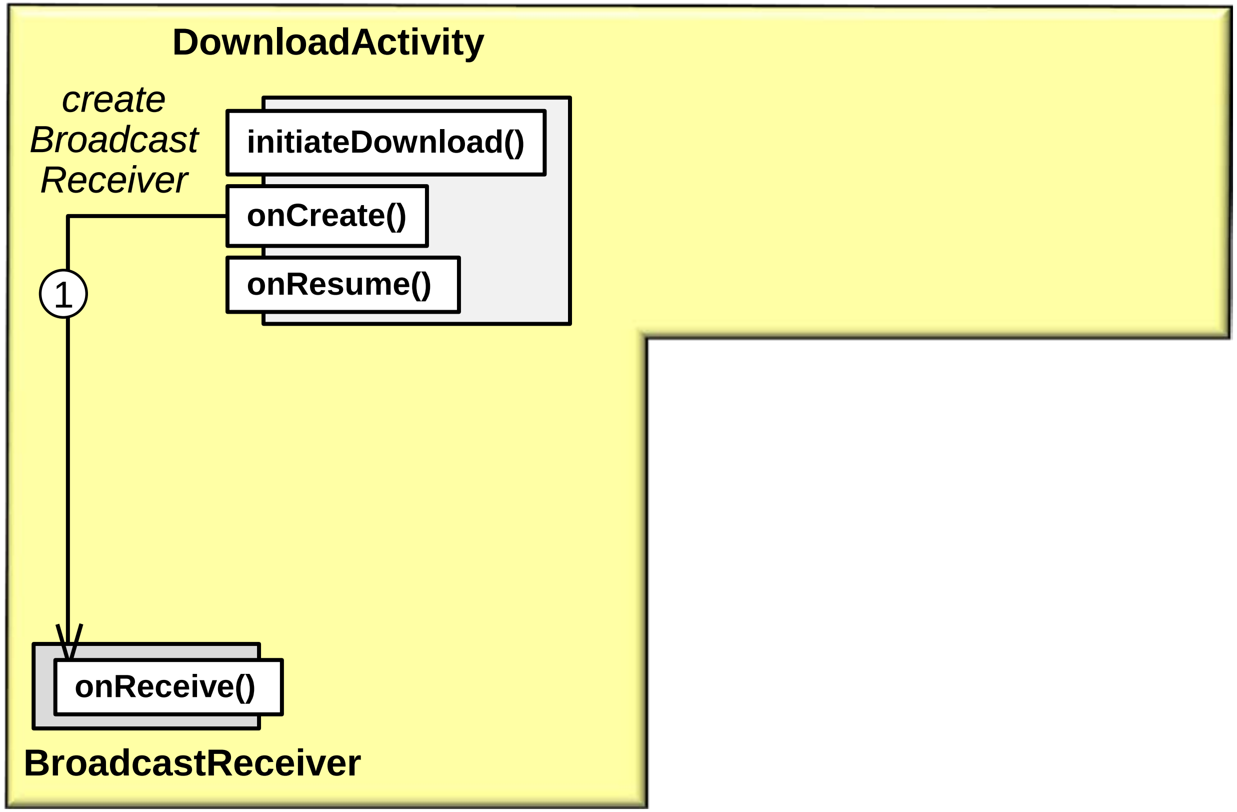
- BroadcastReceivers are components (*receivers*) that register for broadcast events & receive/react to the events
- Activities can create receivers that register for system or app events
- A receiver is restricted on what it can do when it handles an Intent
- Two ways to register a receiver
- Android supports several broadcast mechanisms
  - *Normal* – Sent with Context.sendBroadcast(), which is completely asynchronous
  - *Ordered* – Sent with Context.sendOrderedBroadcast(), which is delivered to one receiver at a time



developer.android.com/reference/android/content/Context.html#sendOrderedBroadcast(android.content.Intent, java.lang.String)

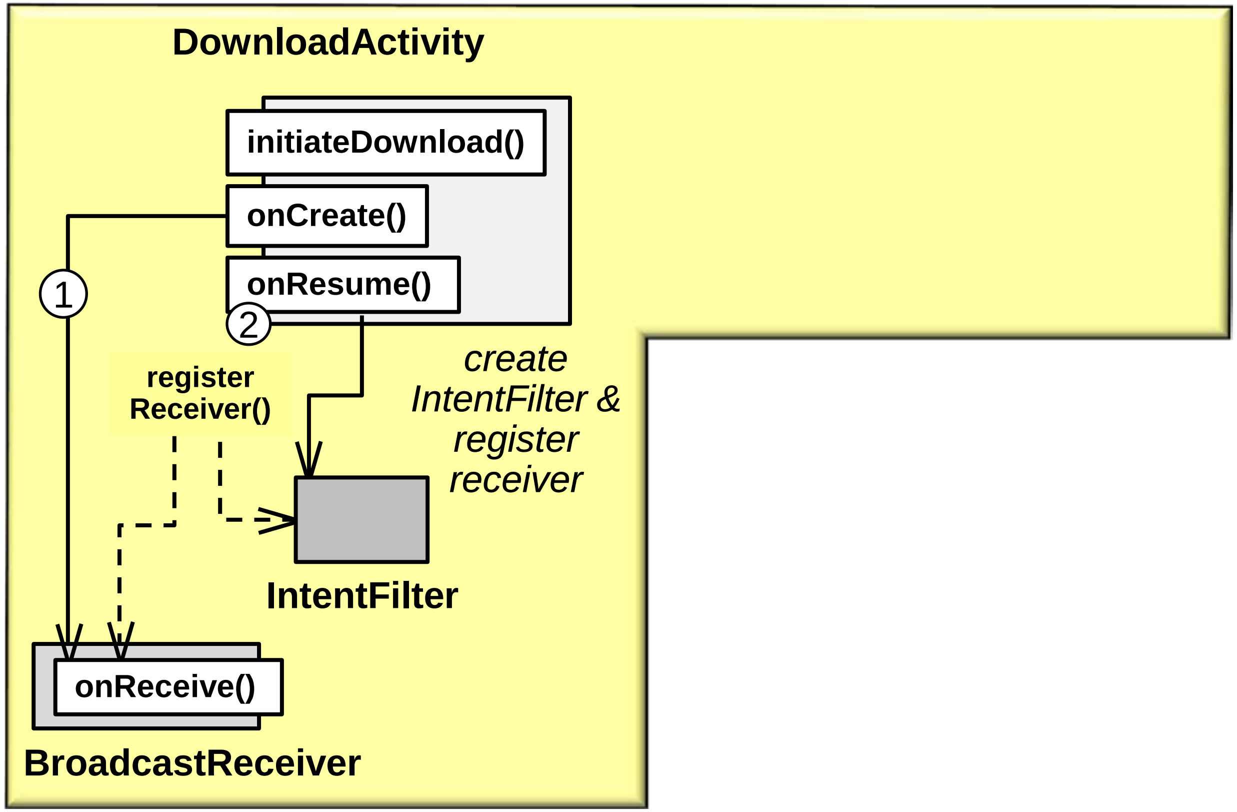
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
  - DownloadService broadcasts an ACTION_COMPLETE back to the Activity



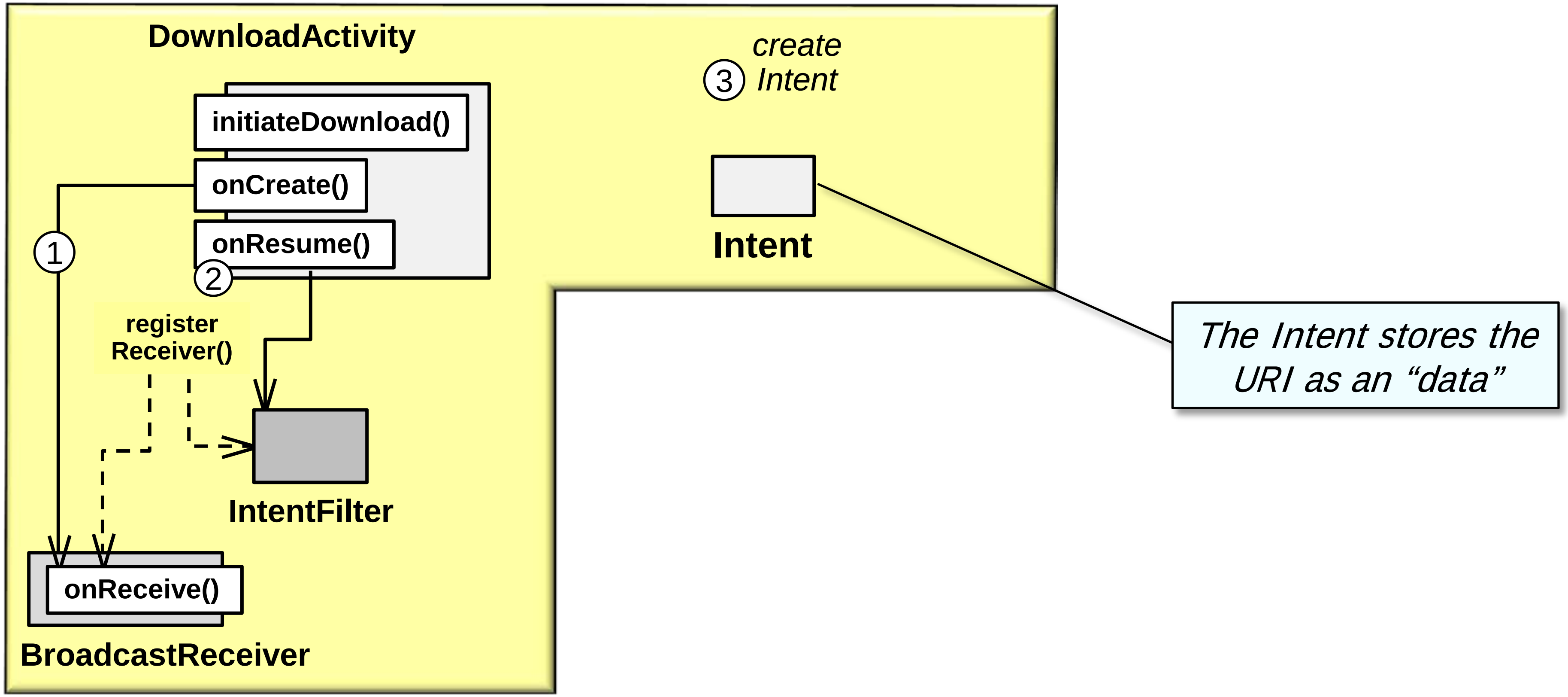
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
  - DownloadService broadcasts an ACTION_COMPLETE back to the Activity



Using Broadcast Receiver in Download App

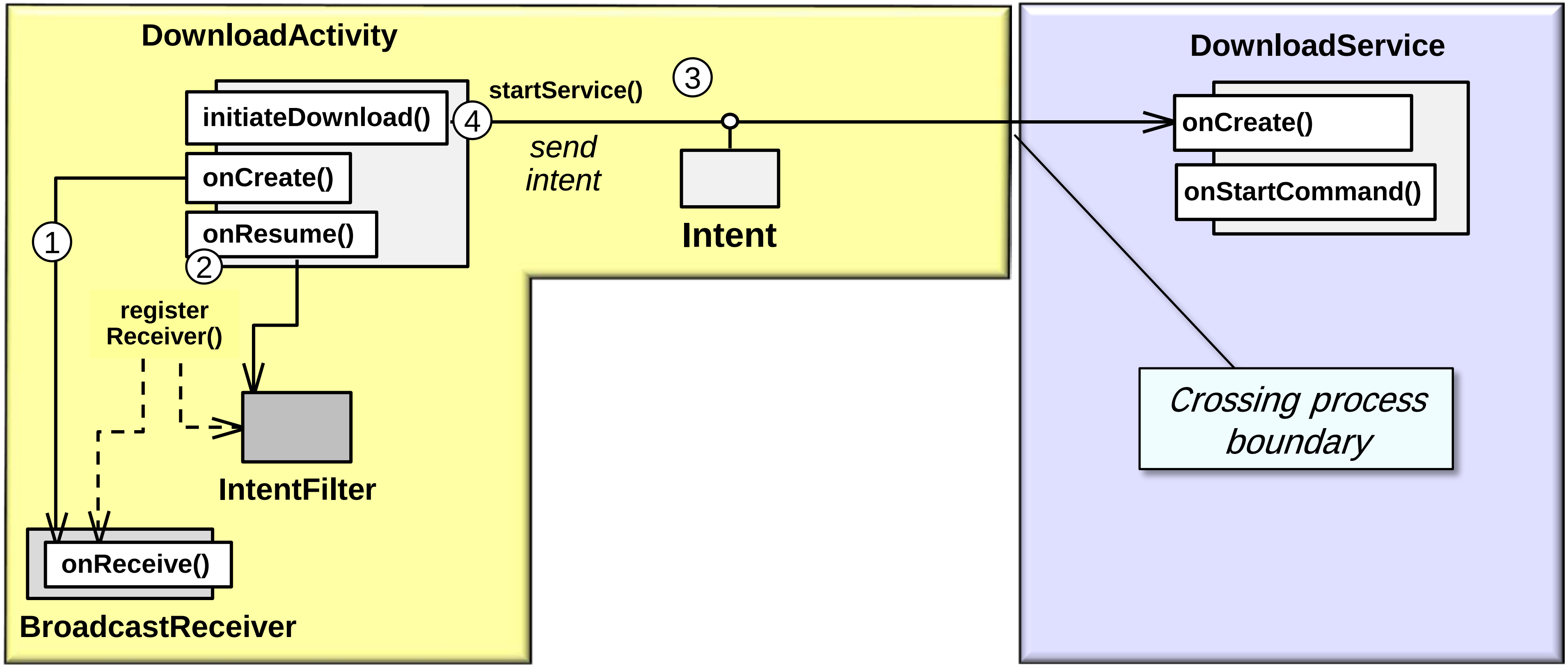
- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
  - DownloadService broadcasts an ACTION_COMPLETE back to the Activity



The Intent stores the URI as an "data"

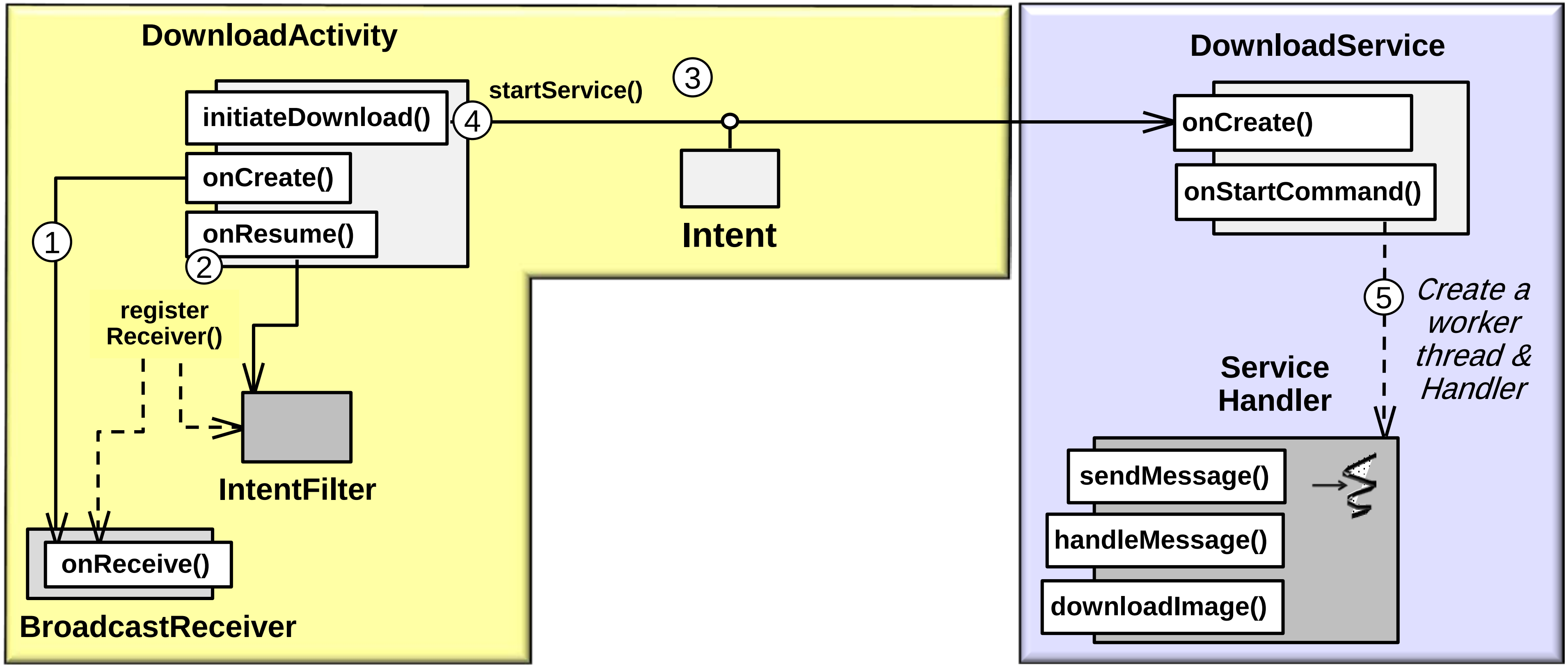
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
  - DownloadService broadcasts an ACTION_COMPLETE back to the Activity



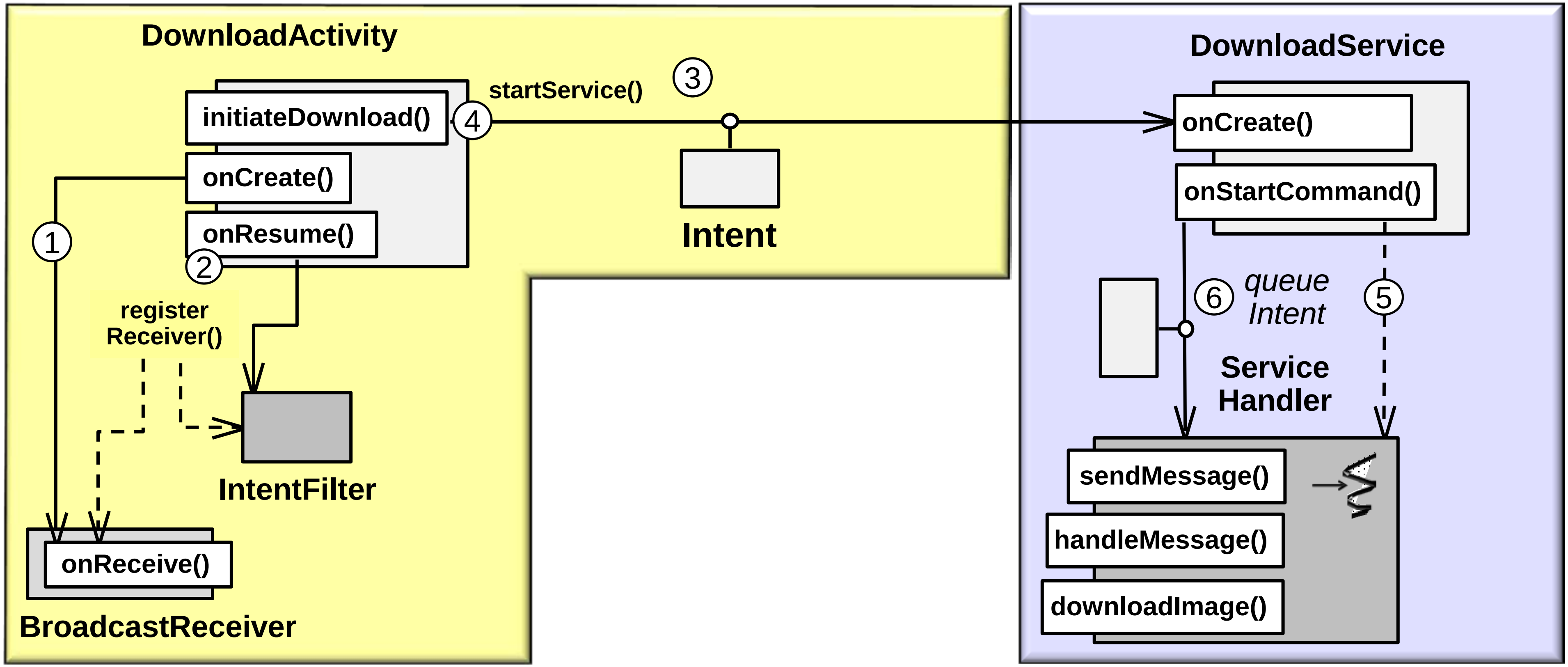
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
  - DownloadService broadcasts an ACTION_COMPLETE back to the Activity



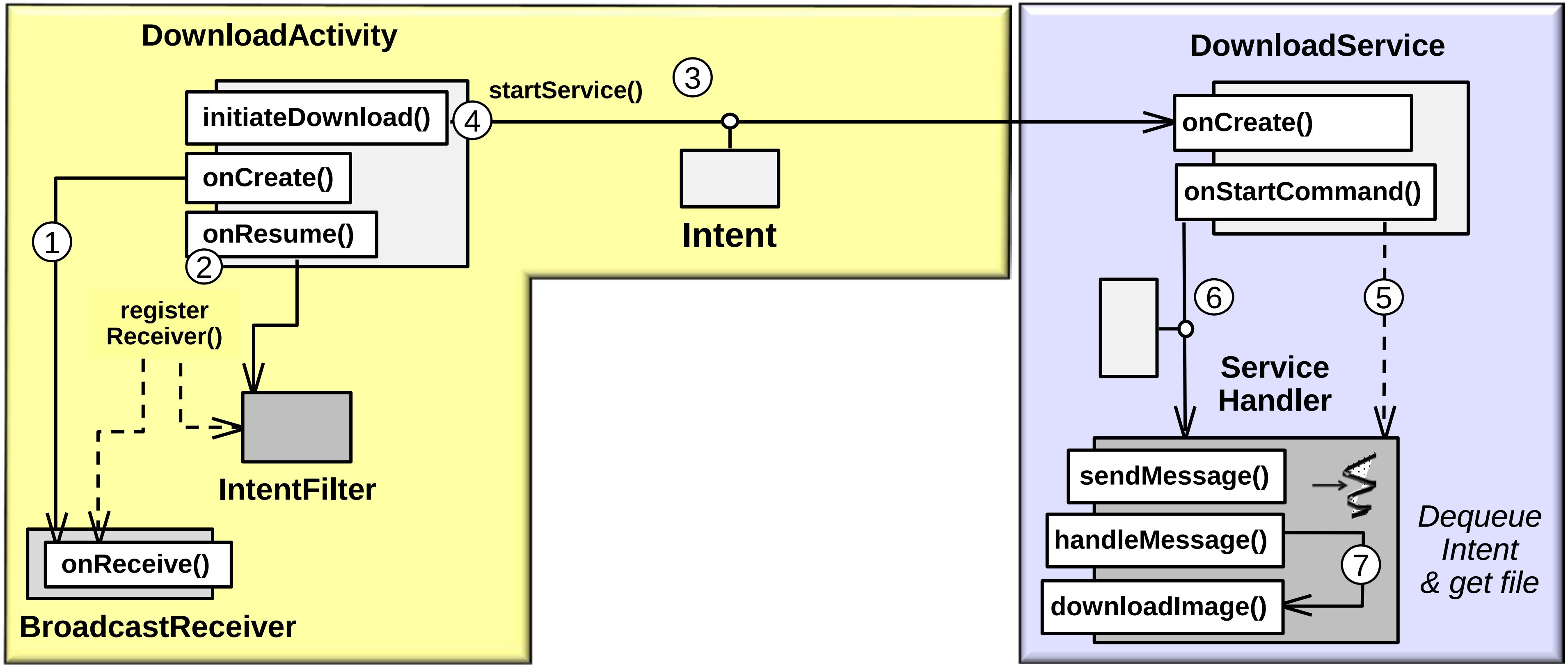
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
  - DownloadService broadcasts an ACTION_COMPLETE back to the Activity



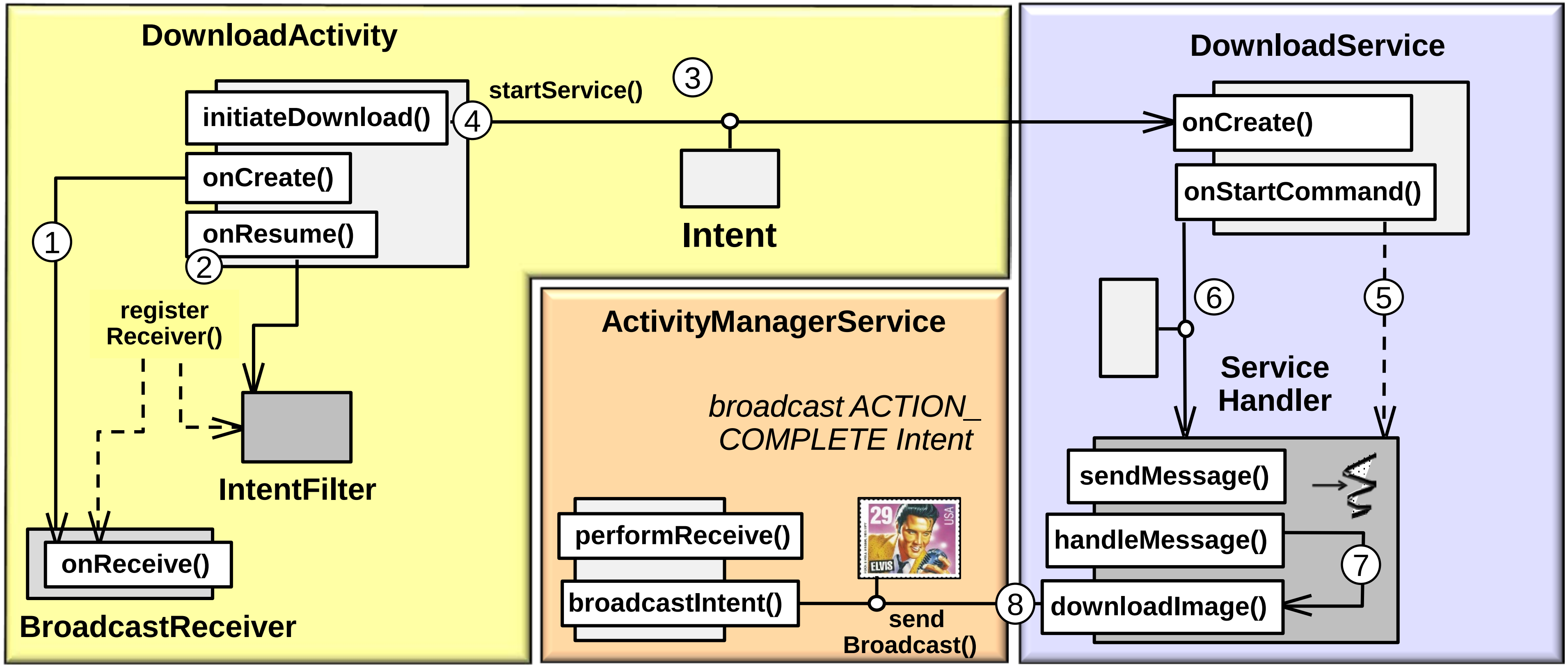
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
  - DownloadService broadcasts an ACTION_COMPLETE back to the Activity



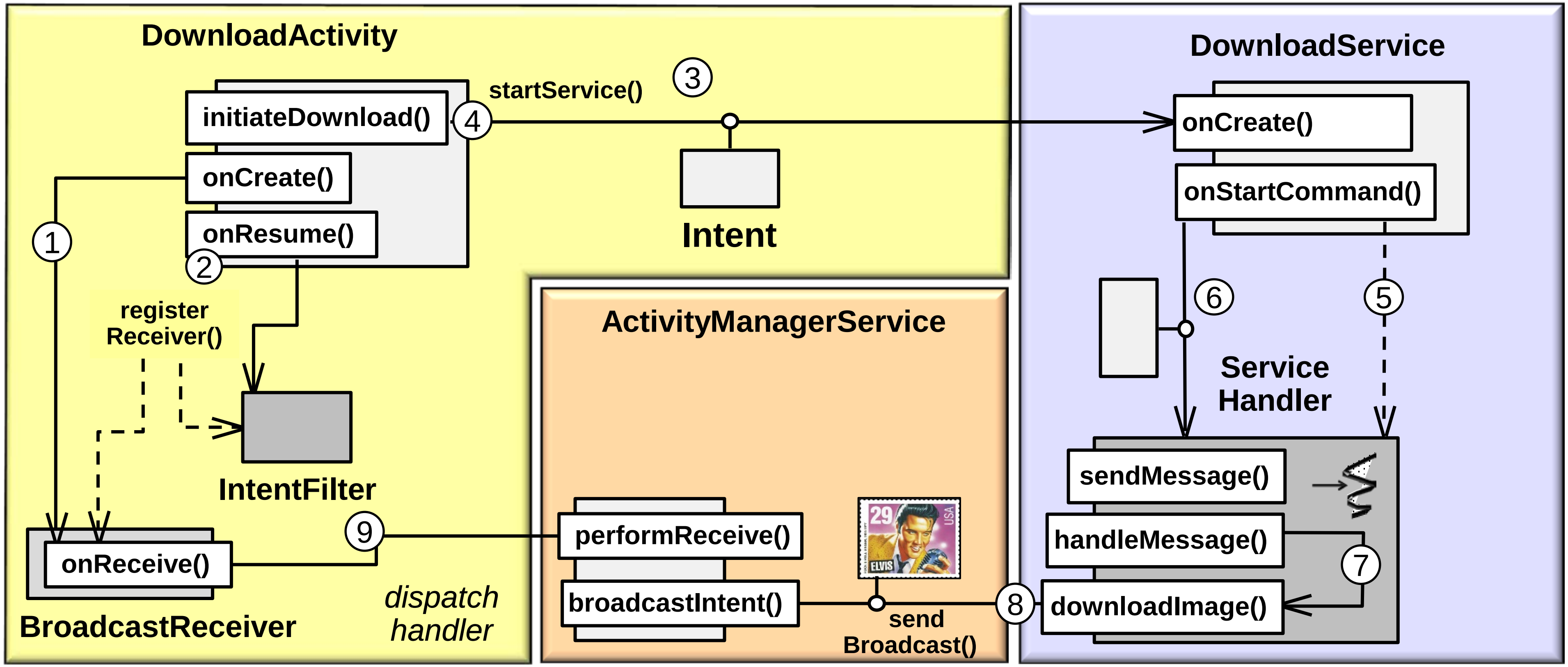
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
  - DownloadService broadcasts an ACTION_COMPLETE back to the Activity



Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
  - DownloadService broadcasts an ACTION_COMPLETE back to the Activity



Programming a Broadcast Receiver in Activity

- DownloadActivity contains a BroadcastReceiver instance with hook method

```
public class DownloadActivity extends Activity {
  private BroadcastReceiver onEvent = null;
  public void onCreate(Bundle onSavedInstance) {
    onEvent = new BroadcastReceiver() {
      public void onReceive(Context context, Intent intent) {
```

Receive Intent sent by sendBroadcast()

```
        String path = intent.getStringExtra(RESULT_PATH);
```

Extract the path using “extra” within the Intent

```
        if (path == null)
          Toast.makeText(DownloadActivity.this,
                  "Download failed.", Toast.LENGTH_LONG).show();
        displayImage(path);
    }
  };
  ...
```

Display the image

Programming a Broadcast Receiver in Activity

- DownloadActivity’s lifecycle methods register & unregister the receiver

```
public class DownloadActivity extends Activity {
  ...
  public void onResume() {
    super.onResume();
    IntentFilter filter =
      new IntentFilter(ACTION_COMPLETE);
    registerReceiver(onEvent, filter);
  }
```

Register BroadcastReceiver when Activity resumes

```
  public void onPause() {
    super.onPause();
    unregisterReceiver(onEvent);
  }
  ...
```

Unregister BroadcastReceiver before Activity pauses

Programming a Broadcast Receiver in Activity

- DownloadActivity passes the package name to the DownloadService

```
public class DownloadActivity extends Activity {
  ...

  public void initiateDownload(View v) {
     Intent intent = new Intent(DownloadActivity.this,
                                DownloadService.class);
     ...
```

Pass a package name as an "extra" in the Intent used to start the DownloadService

```
     intent.putExtra(PACKAGE_NAME, getPackageName());
     startService(intent);
  }
 ...
```

Start the service

Programming a Broadcast Receiver in Service

- DownloadService replies to DownloadActivity via sendBroadcast()

```
public class DownloadService extends Service {
  ...
  private final class ServiceHandler extends Handler {
    ...
    public void downloadImage(Intent intent) {
      // ...
```

Code to downloading image to pathname goes here

```
      Intent replyIntent = new Intent(ACTION_COMPLETE);
      replyIntent.putExtra(RESULT_PATH, pathname);
      String packageName = intent.getStringExtra(PACKAGE_NAME);
      intent.setPackage(packageName);
```

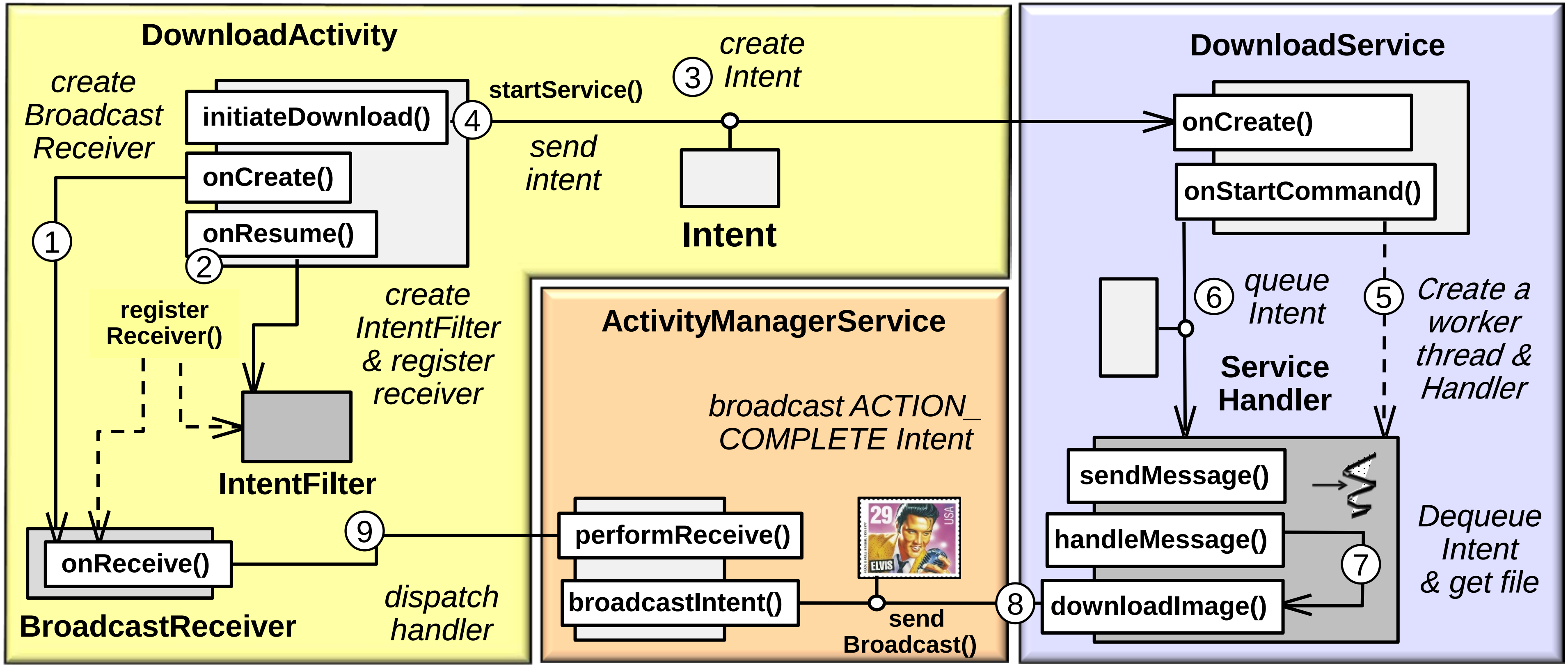
Restrict the target of the broadcast

```
      sendBroadcast(replyIntent);
    }
  ...
```

Broadcast pathname to Activity

Summary

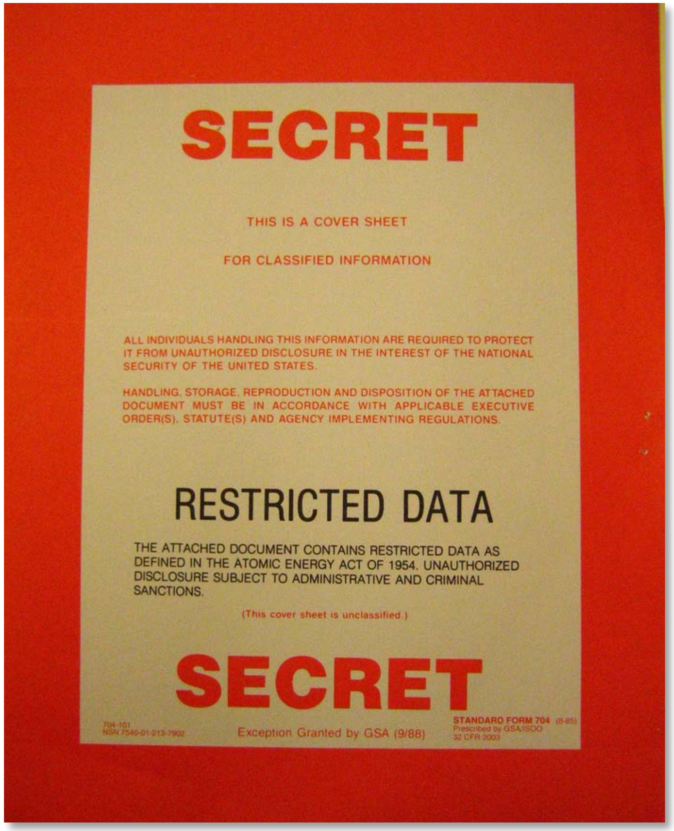
- Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android
  - Broadcast Receivers are generally used for more interesting use-cases...



developer.android.com/reference/android/content/BroadcastReceiver.html#Security

Summary

- Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android
- However, there are subtle issues with security



SECRET

THIS IS A COVER SHEET

FOR CLASSIFIED INFORMATION

ALL INDIVIDUALS HANDLING THIS INFORMATION ARE REQUIRED TO PROTECT IT FROM UNAUTHORIZED DISCLOSURE IN THE INTEREST OF THE NATIONAL SECURITY OF THE UNITED STATES.

HANDLING, STORAGE, REPRODUCTION AND DISPOSITION OF THE ATTACHED DOCUMENT MUST BE IN ACCORDANCE WITH APPLICABLE EXECUTIVE ORDER(S), STATUTE(S) AND AGENCY IMPLEMENTING REGULATIONS.

RESTRICTED DATA

THE ATTACHED DOCUMENT CONTAINS RESTRICTED DATA AS DEFINED IN THE ATOMIC ENERGY ACT OF 1954. UNAUTHORIZED DISCLOSURE SUBJECT TO ADMINISTRATIVE AND CRIMINAL SANCTIONS.

(This cover sheet is unclassified.)

SECRET

704-101
NSN 7540-01-213-7902

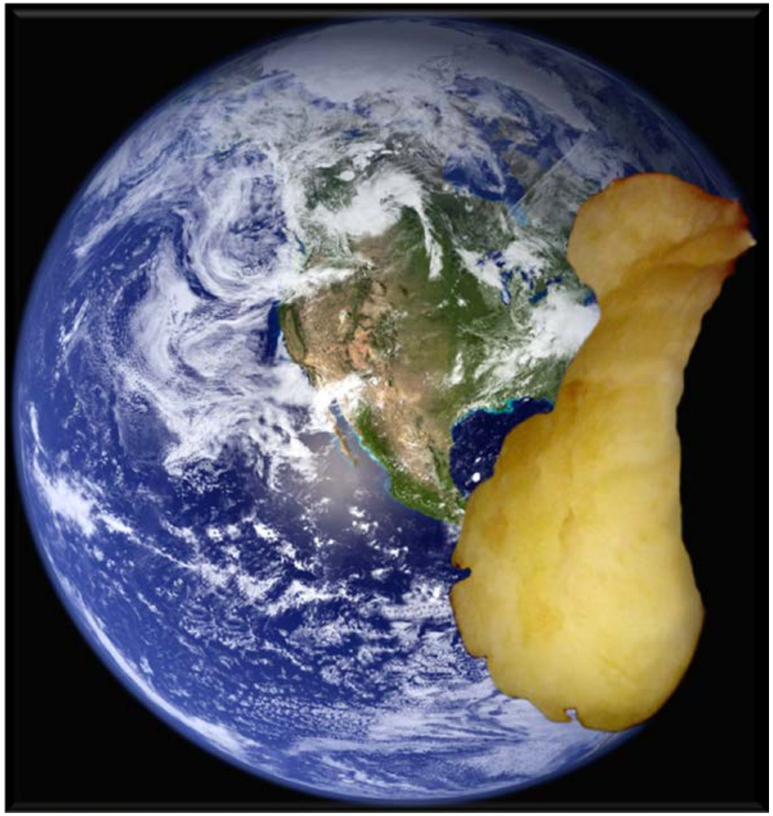
Exception Granted by GSA (9/88)

STANDARD FORM 704 (8-85)
Prescribed by GSA/ISOO
32 CFR 2003

developer.android.com/reference/android/content/BroadcastReceiver.html#Security

Summary

- Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android
- However, there are subtle issues with security
  - The Intent namespace is global
    - This may cause subtle conflicts



Summary

- Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android
- However, there are subtle issues with security
  - The Intent namespace is global
  - registerReceiver() allows any app to send broadcasts to that registered receiver
    - Use permissions to address this

developer.android.com/reference/android/content/BroadcastReceiver.html#Security

Summary

- Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android
- However, there are subtle issues with security
  - The Intent namespace is global
  - registerReceiver(BroadcastReceiver, IntentFilter) allows any app to send broadcasts to that registered receiver
  - When a receiver is published in an app’s manifest & specifies intent-filters for it, any other app can send broadcasts to it regardless of the specified filters
    - To prevent others from sending to it, make it unavailable to them with android:exported="false"

```
<receiver
  android:enabled=
    ["true" | "false"]
  android:exported=
    ["true" | "false"]
  android:icon="drawable resource"
  android:label="string resource"
  android:name="string"
  android:permission="string"
  android:process="string" >
    . . .
</receiver>
```

developer.android.com/guide/topics/manifest/receiver-element.html

Summary

- Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android
- However, there are subtle issues with security
  - The Intent namespace is global
  - registerReceiver(BroadcastReceiver, IntentFilter) allows any app to send broadcasts to that registered receiver
  - When a receiver is published in an app's manifest & specifies intent-filters for it, any other app can send broadcasts to it regardless of the filters that are specified

  - sendBroadcast() et al allow any other app to receive broadcasts
    - Broadcasts can be restricted to a single app with Intent.setPackage()

developer.android.com/reference/android/content/Intent.html#setPackage(java.lang.String)

Android Services & Local IPC: Communicating via Pending Intents

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

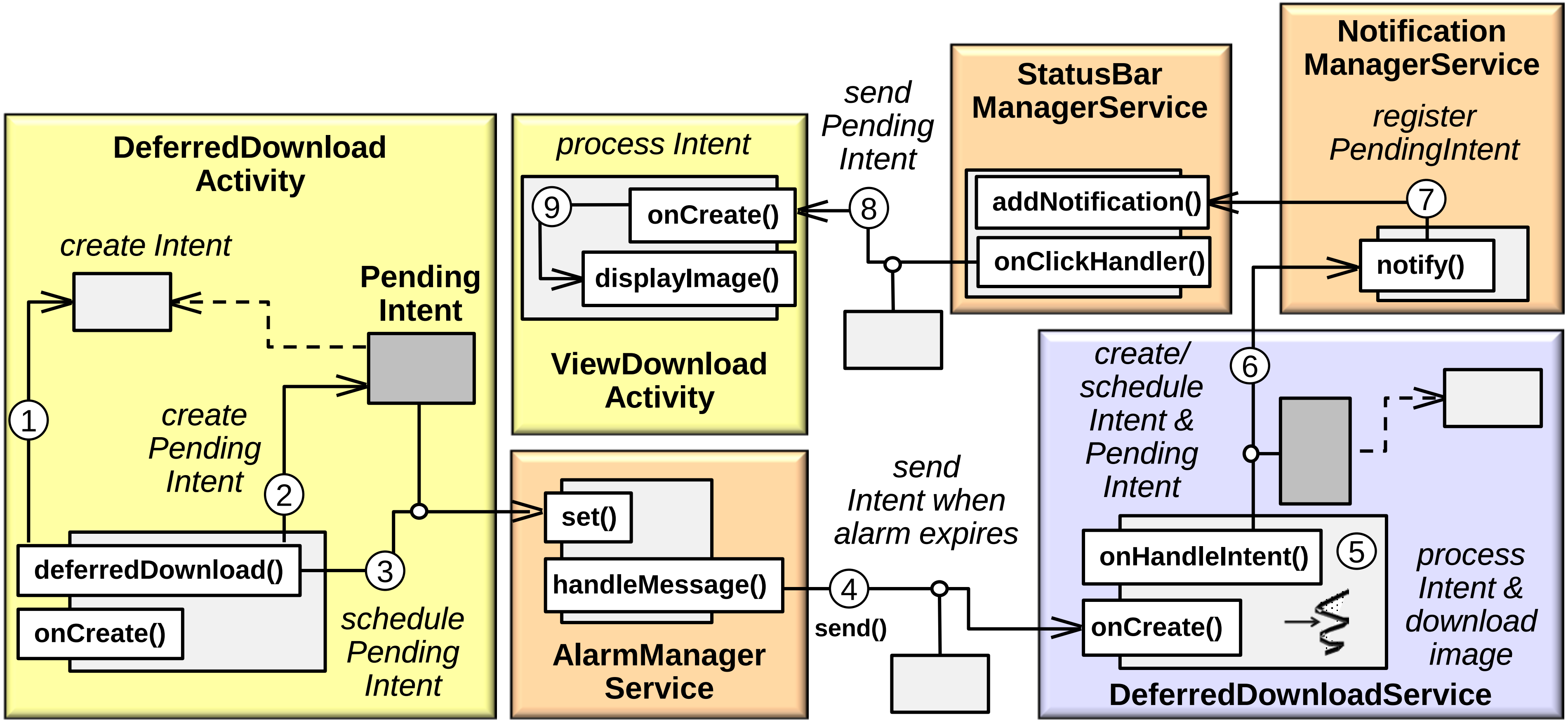
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

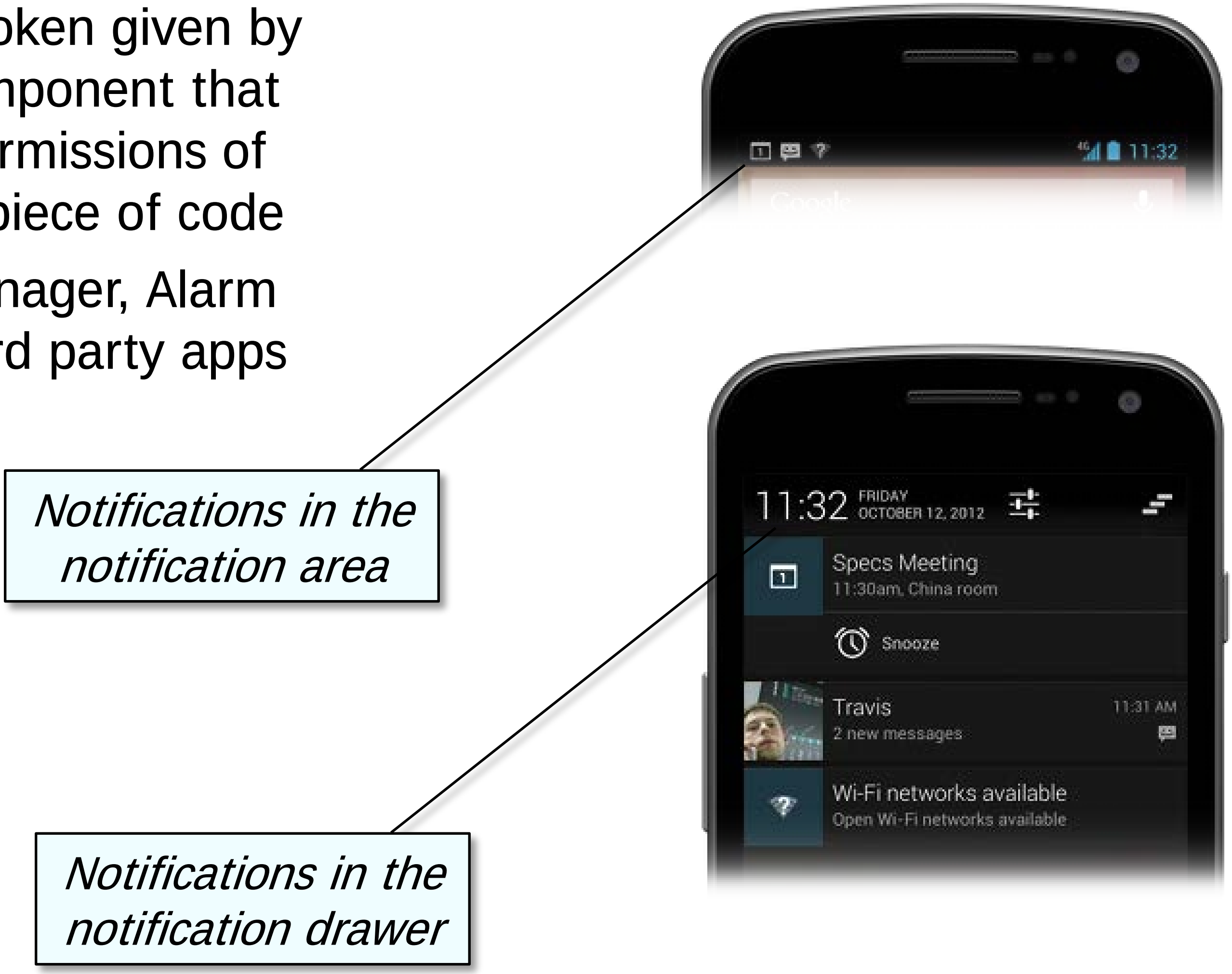
Learning Objectives in this Part of the Module

- Understand how to use Pending Intents to communicate from (Started) Services back to other components (e.g., Activities, Broadcast Receivers, etc.)
  - A PendingIntent is a token given to an App to perform an action on your Apps' behalf *irrespective* of whether your App's process is alive



Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code
  - e.g., Notification Manager, Alarm Manager, or other 3rd party apps



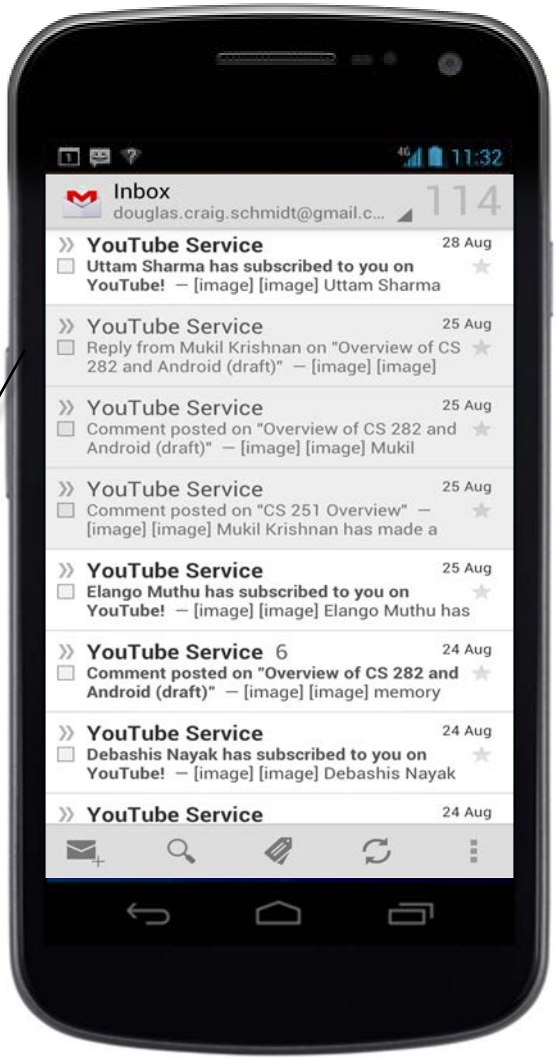
Notifications in the notification area

Notifications in the notification drawer

developer.android.com/reference/android/app/PendingIntent.html

Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code
  - e.g., Notification Manager, Alarm Manager, or other 3rd party apps
  - The token maintained by the system represents an Intent & the action to perform on that Intent later
    - Can be configured to work irrespective of whether the original App process is alive or not



11:32

Inbox
douglas.craig.schmidt@gmail.c... 114

YouTube Service 28 Aug
Uttam Sharma has subscribed to you on YouTube! – [image] [image] Uttam Sharma

YouTube Service 25 Aug
Reply from Mukil Krishnan on "Overview of CS 282 and Android (draft)" – [image] [image]

YouTube Service 25 Aug
Comment posted on "Overview of CS 282 and Android (draft)" – [image] [image] Mukil

YouTube Service 25 Aug
Comment posted on "CS 251 Overview" – [image] [image] Mukil Krishnan has made a

YouTube Service 25 Aug
Elango Muthu has subscribed to you on YouTube! – [image] [image] Elango Muthu has

YouTube Service 6 24 Aug
Comment posted on "Overview of CS 282 and Android (draft)" – [image] [image] memory

YouTube Service 24 Aug
Debashis Nayak has subscribed to you on YouTube! – [image] [image] Debashis Nayak

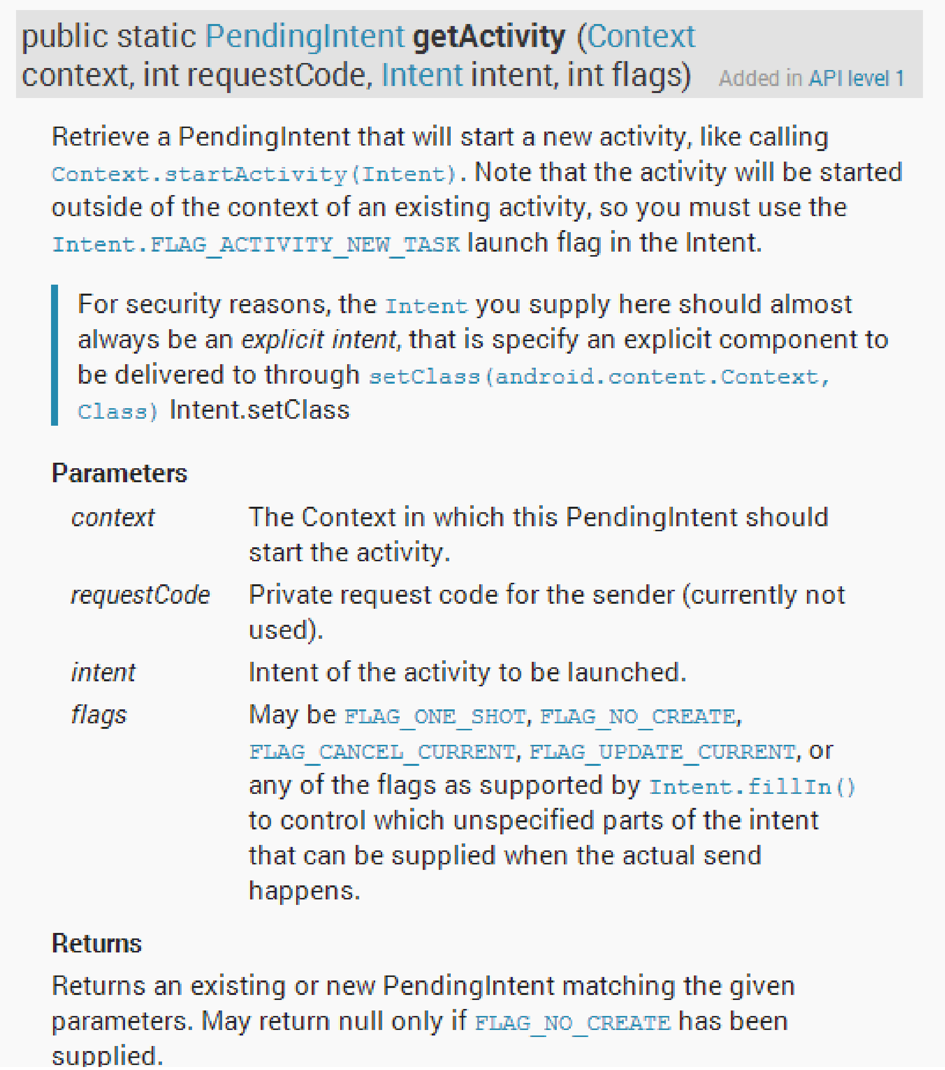
YouTube Service 24 Aug

Start an Activity to read email

developer.android.com/reference/android/app/PendingIntent.html

Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code
- PendingIntents can be created via various methods, e.g.:
  - getActivity() on PendingIntent
    - The PendingIntent returned by this method starts a new Activity when send() is called on it



public static PendingIntent **getActivity** (Context context, int requestCode, Intent intent, int flags) Added in API level 1

Retrieve a PendingIntent that will start a new activity, like calling `Context.startActivity(Intent)`. Note that the activity will be started outside of the context of an existing activity, so you must use the `Intent.FLAG_ACTIVITY_NEW_TASK` launch flag in the Intent.

> For security reasons, the `Intent` you supply here should almost always be an *explicit intent*, that is specify an explicit component to be delivered to through `setClass(android.content.Context, Class)` Intent.setClass

Parameters

context	The Context in which this PendingIntent should start the activity.
requestCode	Private request code for the sender (currently not used).
intent	Intent of the activity to be launched.
flags	May be `FLAG_ONE_SHOT`, `FLAG_NO_CREATE`, `FLAG_CANCEL_CURRENT`, `FLAG_UPDATE_CURRENT`, or any of the flags as supported by `Intent.fillIn()` to control which unspecified parts of the intent that can be supplied when the actual send happens.

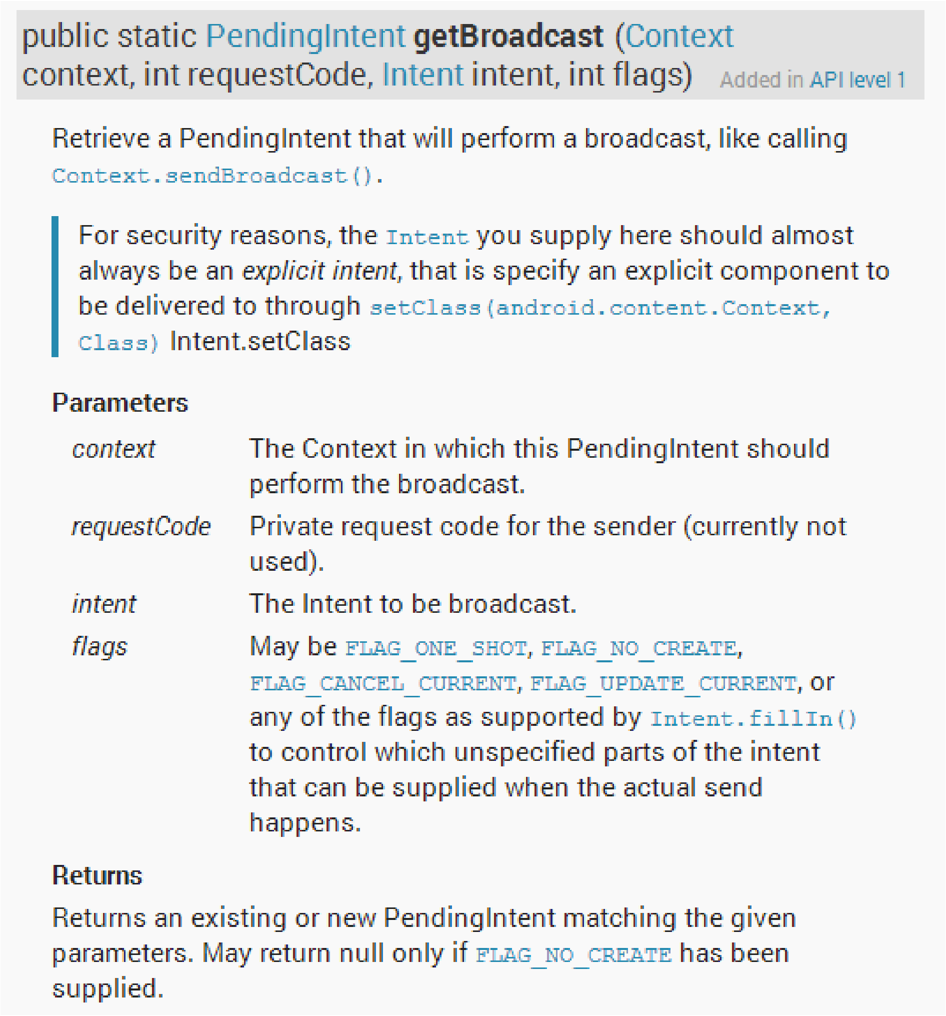
Returns

Returns an existing or new PendingIntent matching the given parameters. May return null only if `FLAG_NO_CREATE` has been supplied.

developer.android.com/reference/android/app/PendingIntent.html #getActivity(android.content.Context, int, android.content.Intent, int)

Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code
- PendingIntents can be created via various methods, e.g.:
  - getActivity() on PendingIntent
  - getBroadcast() on PendingIntent
    - The PendingIntent returned by this method sends a broadcast to a Receiver when send() is called on it



public static PendingIntent **getBroadcast** (Context context, int requestCode, Intent intent, int flags) Added in API level 1

Retrieve a PendingIntent that will perform a broadcast, like calling `Context.sendBroadcast()`.

> For security reasons, the `Intent` you supply here should almost always be an *explicit intent*, that is specify an explicit component to be delivered to through `setClass(android.content.Context, Class)` Intent.setClass

Parameters

context	The Context in which this PendingIntent should perform the broadcast.
requestCode	Private request code for the sender (currently not used).
intent	The Intent to be broadcast.
flags	May be `FLAG_ONE_SHOT`, `FLAG_NO_CREATE`, `FLAG_CANCEL_CURRENT`, `FLAG_UPDATE_CURRENT`, or any of the flags as supported by `Intent.fillIn()` to control which unspecified parts of the intent that can be supplied when the actual send happens.

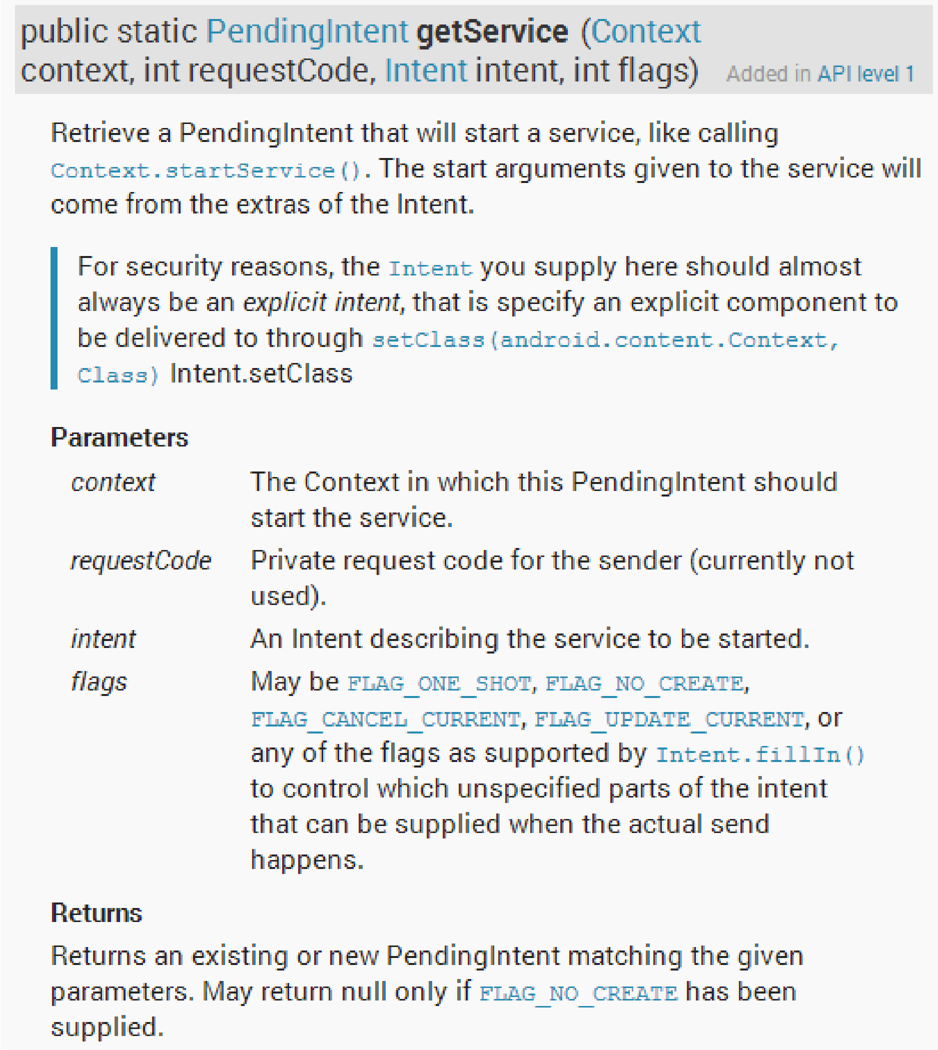
Returns

Returns an existing or new PendingIntent matching the given parameters. May return null only if `FLAG_NO_CREATE` has been supplied.

developer.android.com/reference/android/app/PendingIntent.html #getBroadcast(android.content.Context, int, android.content.Intent, int)

Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code
- PendingIntents can be created via various methods, e.g.:
  - getActivity() on PendingIntent
  - getBroadcast() on PendingIntent
  - getService() on PendingIntent
    - The PendingIntent returned by this method starts a new Service when send() is called on it



public static PendingIntent **getService** (Context context, int requestCode, Intent intent, int flags) Added in API level 1

Retrieve a PendingIntent that will start a service, like calling `Context.startService()`. The start arguments given to the service will come from the extras of the Intent.

> For security reasons, the `Intent` you supply here should almost always be an *explicit intent*, that is specify an explicit component to be delivered to through `setClass(android.content.Context, Class)` Intent.setClass

Parameters

context	The Context in which this PendingIntent should start the service.
requestCode	Private request code for the sender (currently not used).
intent	An Intent describing the service to be started.
flags	May be `FLAG_ONE_SHOT`, `FLAG_NO_CREATE`, `FLAG_CANCEL_CURRENT`, `FLAG_UPDATE_CURRENT`, or any of the flags as supported by `Intent.fillIn()` to control which unspecified parts of the intent that can be supplied when the actual send happens.

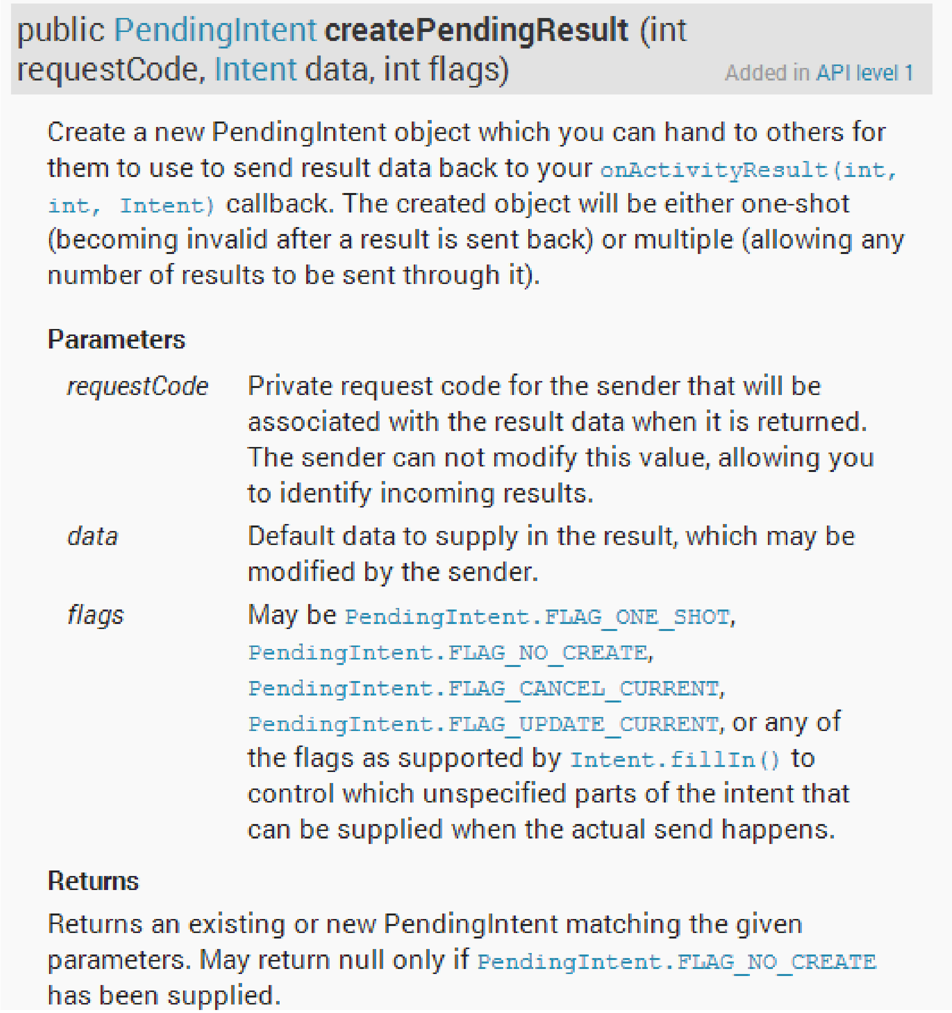
Returns

Returns an existing or new PendingIntent matching the given parameters. May return null only if `FLAG_NO_CREATE` has been supplied.

developer.android.com/reference/android/app/PendingIntent.html #getService(android.content.Context, int, android.content.Intent, int)

Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code
- PendingIntents can be created via various methods, e.g.:
  - getActivity() on PendingIntent
  - getBroadcast() on PendingIntent
  - getService() on PendingIntent
  - createPendingResult() on Activity
    - The PendingIntent returned by this method sends data back to the Activity via its method onActivityResult()



public PendingIntent **createPendingResult** (int requestCode, Intent data, int flags) Added in API level 1

Create a new PendingIntent object which you can hand to others for them to use to send result data back to your `onActivityResult(int, int, Intent)` callback. The created object will be either one-shot (becoming invalid after a result is sent back) or multiple (allowing any number of results to be sent through it).

Parameters

requestCode	Private request code for the sender that will be associated with the result data when it is returned. The sender can not modify this value, allowing you to identify incoming results.
data	Default data to supply in the result, which may be modified by the sender.
flags	May be `PendingIntent.FLAG_ONE_SHOT`, `PendingIntent.FLAG_NO_CREATE`, `PendingIntent.FLAG_CANCEL_CURRENT`, `PendingIntent.FLAG_UPDATE_CURRENT`, or any of the flags as supported by `Intent.fillIn()` to control which unspecified parts of the intent that can be supplied when the actual send happens.

Returns

Returns an existing or new PendingIntent matching the given parameters. May return null only if `PendingIntent.FLAG_NO_CREATE` has been supplied.

developer.android.com/reference/android/app/Activity.html #createPendingResult(int, android.content.Intent, int)

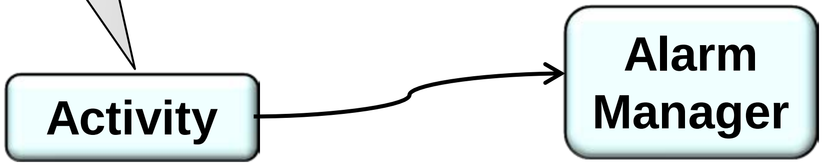
Using PendingIntent w/AlarmManager Service

- PendingIntents are often used with alarms
    - Activity creates & schedules a PendingIntent with the Alarm Service

```
PendingIntent pi;
AlarmManager mgr;

void onCreate(Bundle b) {
  AlarmManager mgr =(AlarmManager)
    getSystemService
      (ALARM_SERVICE);
  Intent replyIntent = new Intent();
  ... // Set “extras” in replyIntent
  pi = createPendingResult
         (ALARM_ID, replyIntent, 0);
  mgr.setRepeating
    (AlarmManager.
       ELAPSED_REALTIME_WAKEUP,
     SystemClock.elapsedRealtime()
     + PERIOD, PERIOD, pi);
  finish();
}
```

Cause the alarm to restart the Activity when it expires



Using PendingIntent w/AlarmManager Service

- PendingIntents are often used with alarms
  - Activity creates & schedules a PendingIntent with the Alarm Service

```
void setRepeating(int type,
    long triggerAtTime,
    long interval,
    PendingIntent operation) {
  Alarm alarm = new Alarm();
  ...
  alarm.when = triggerAtTime;
  alarm.repeatInterval = interval;
  alarm.operation = operation;

  Message msg = Message.obtain();
  msg.what = ALARM_EVENT;
  ...
  mHandler.sendMessageAtTime
   (msg, alarm.when);
}
```

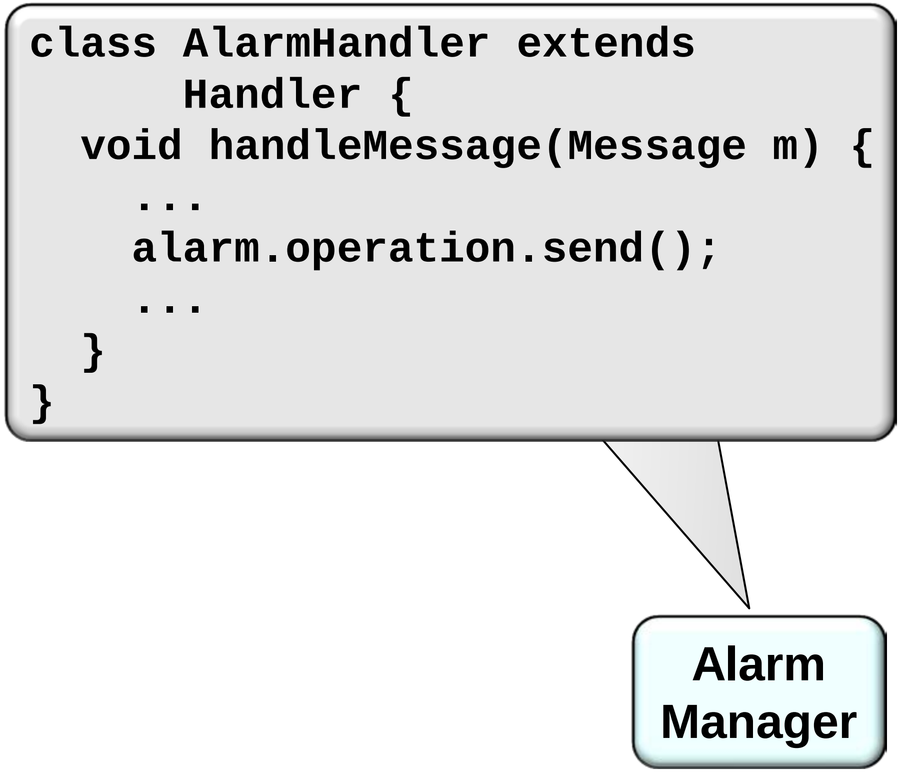
AlarmManager maintains its schedule outside of an App's process, so it can give the App control, even if it has to start up a new process along the way

Alarm Manager

developer.android.com/reference/android/app/AlarmManager.html

Using PendingIntent w/AlarmManager Service

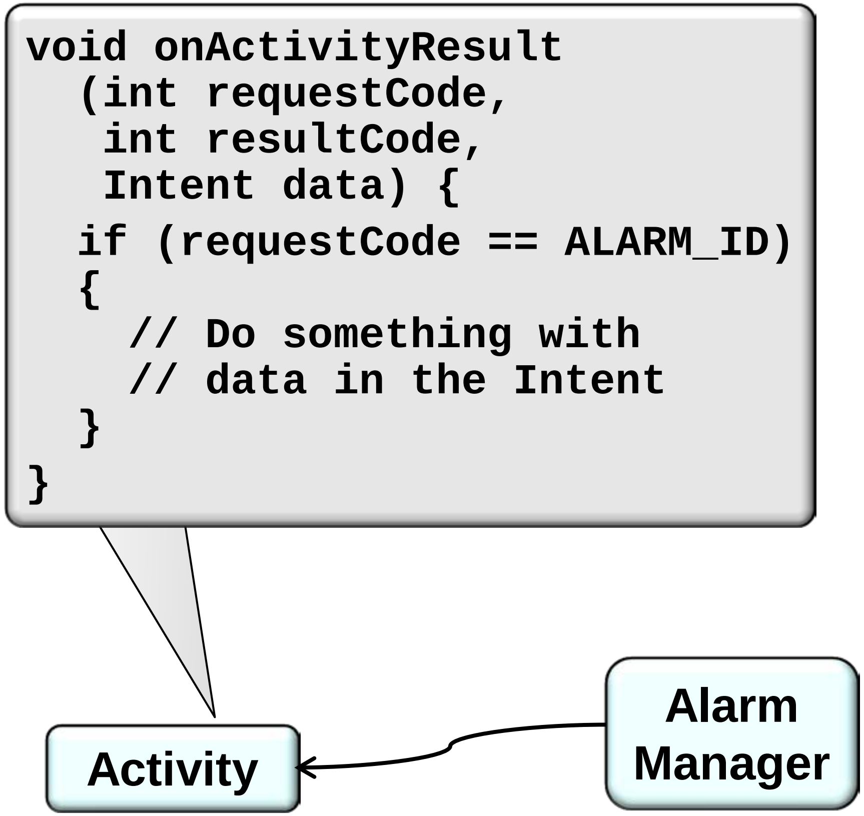
- PendingIntents are often used with alarms
  - Activity creates & schedules a PendingIntent with the Alarm Service
  - When the timer expires the Alarm Service sends a reply back to the Activity



```
class AlarmHandler extends
      Handler {
  void handleMessage(Message m) {
    ...
    alarm.operation.send();
    ...
  }
}
```

Using PendingIntent w/AlarmManager Service

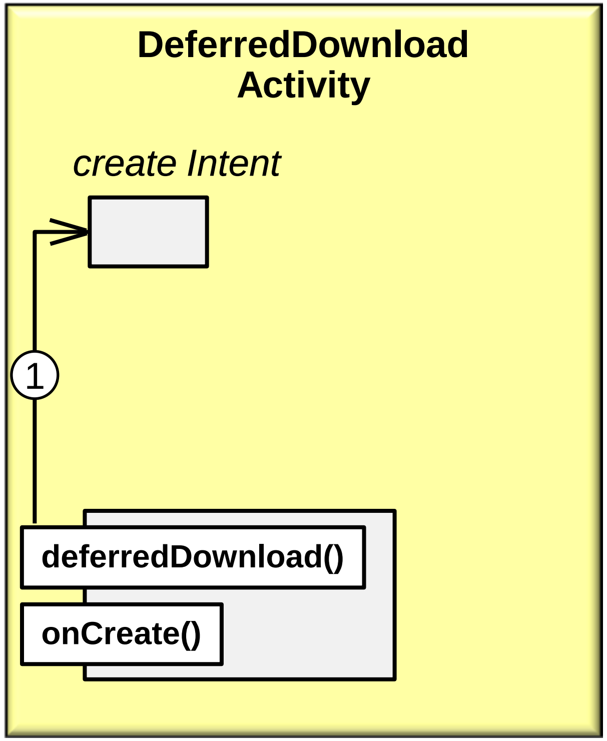
- PendingIntents are often used with alarms
  - Activity creates & schedules a PendingIntent with the Alarm Service
  - When the timer expires the Alarm Service sends a reply back to the Activity
  - The Activity is retarted & its onActivityResult() method handles the reply



```
void onActivityResult
  (int requestCode,
   int resultCode,
   Intent data) {
  if (requestCode == ALARM_ID)
  {
    // Do something with
    // data in the Intent
  }
}
```

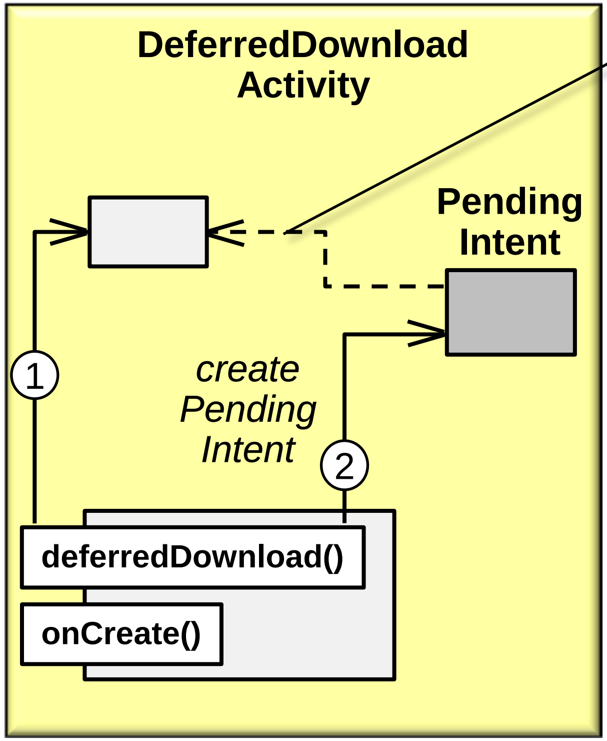
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that's registered with the Alarm Service to start DeferredDownloadService to download an image in the future
  - DeferredDownloadService uses Notification Service to inform user when the image has been downloaded



Using Pending Intents in Deferred Download App

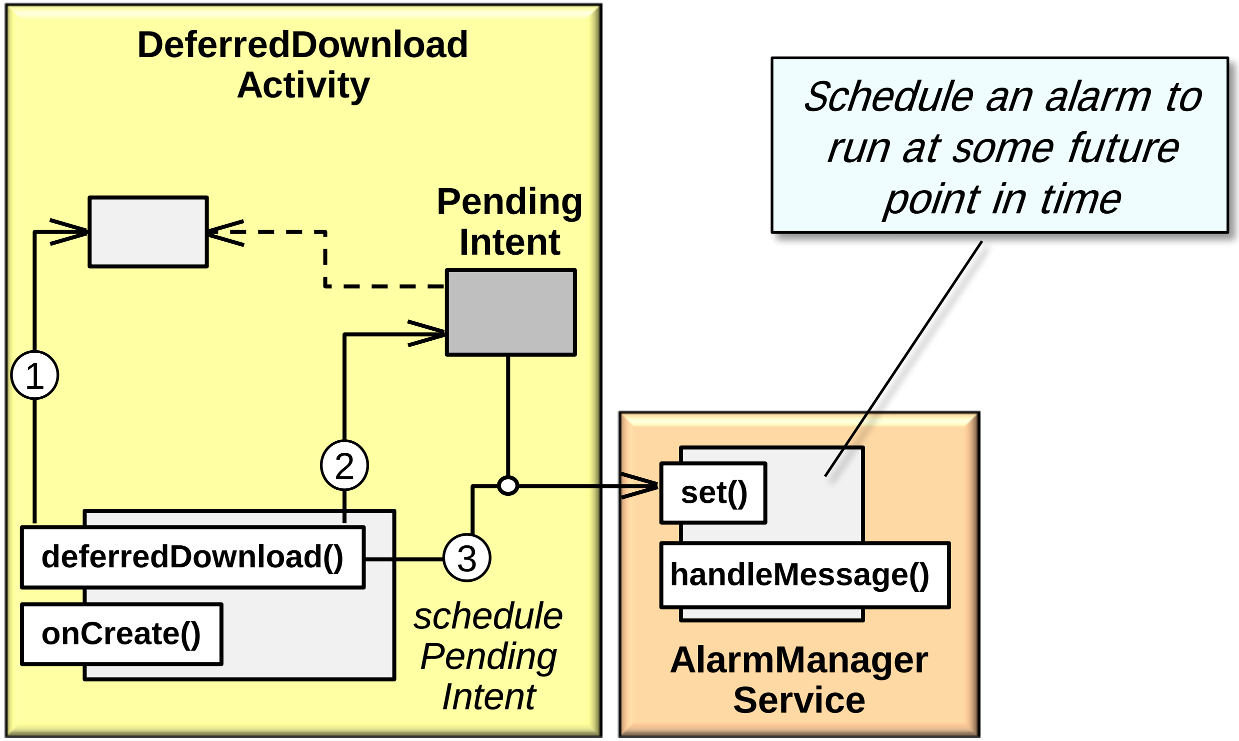
- DownloadActivity creates a PendingIntent that's registered with the Alarm Service to start DeferredDownloadService to download an image in the future
  - DeferredDownloadService uses Notification Service to inform user when the image has been downloaded



The PendingIntent stores a reference to the Intent

Using Pending Intents in Deferred Download App

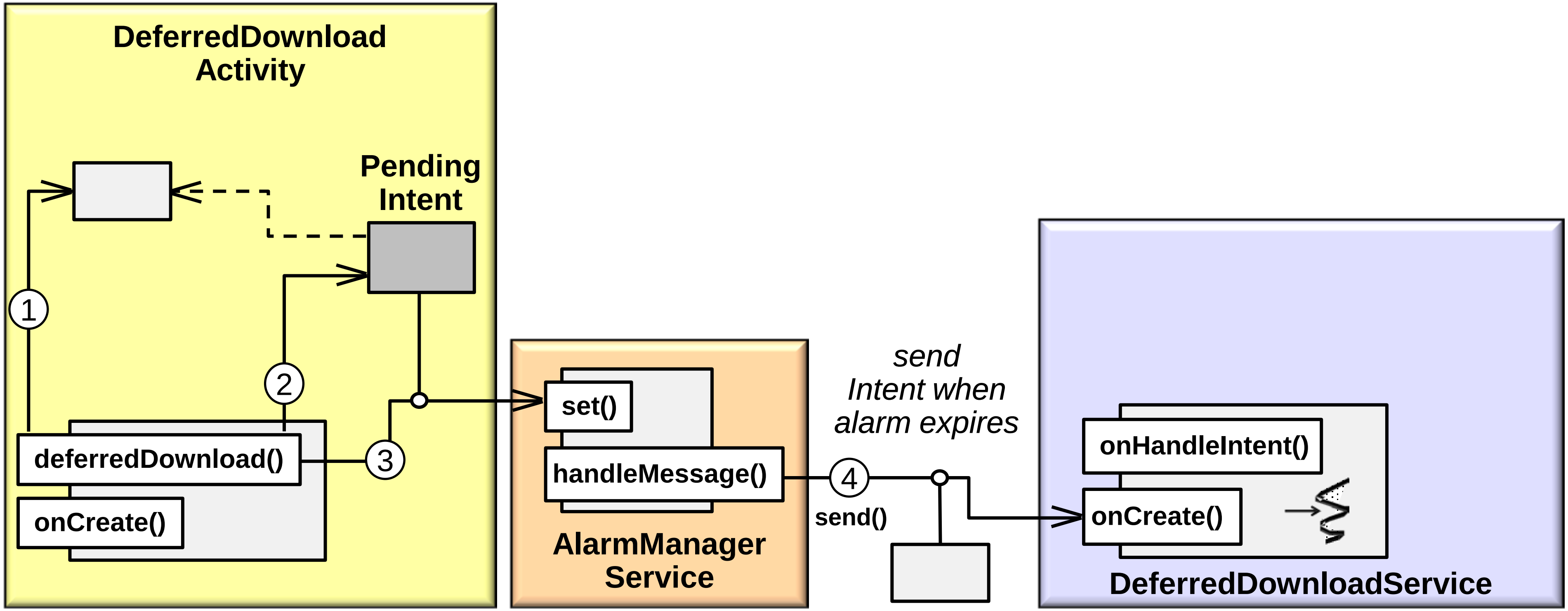
- DownloadActivity creates a PendingIntent that's registered with the Alarm Service to start DeferredDownloadService to download an image in the future
  - DeferredDownloadService uses Notification Service to inform user when the image has been downloaded



Schedule an alarm to run at some future point in time

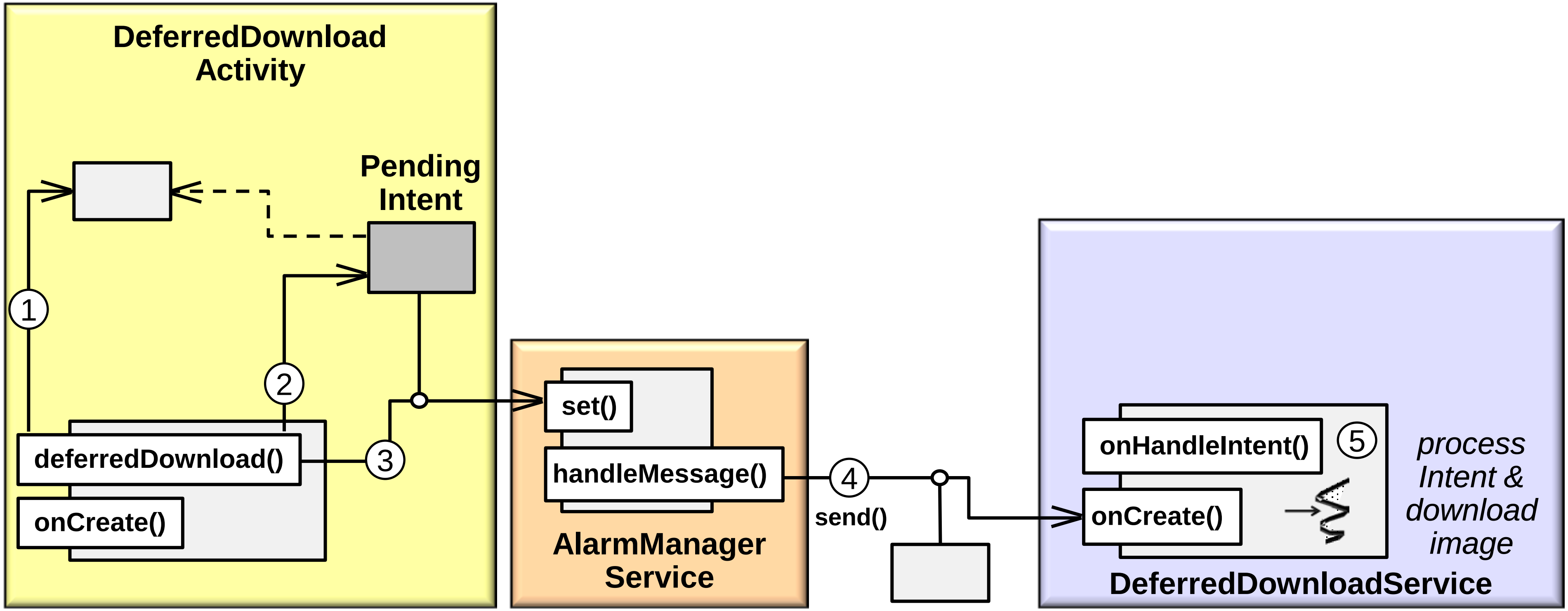
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that's registered with the Alarm Service to start DeferredDownloadService to download an image in the future
  - DeferredDownloadService uses Notification Service to inform user when the image has been downloaded



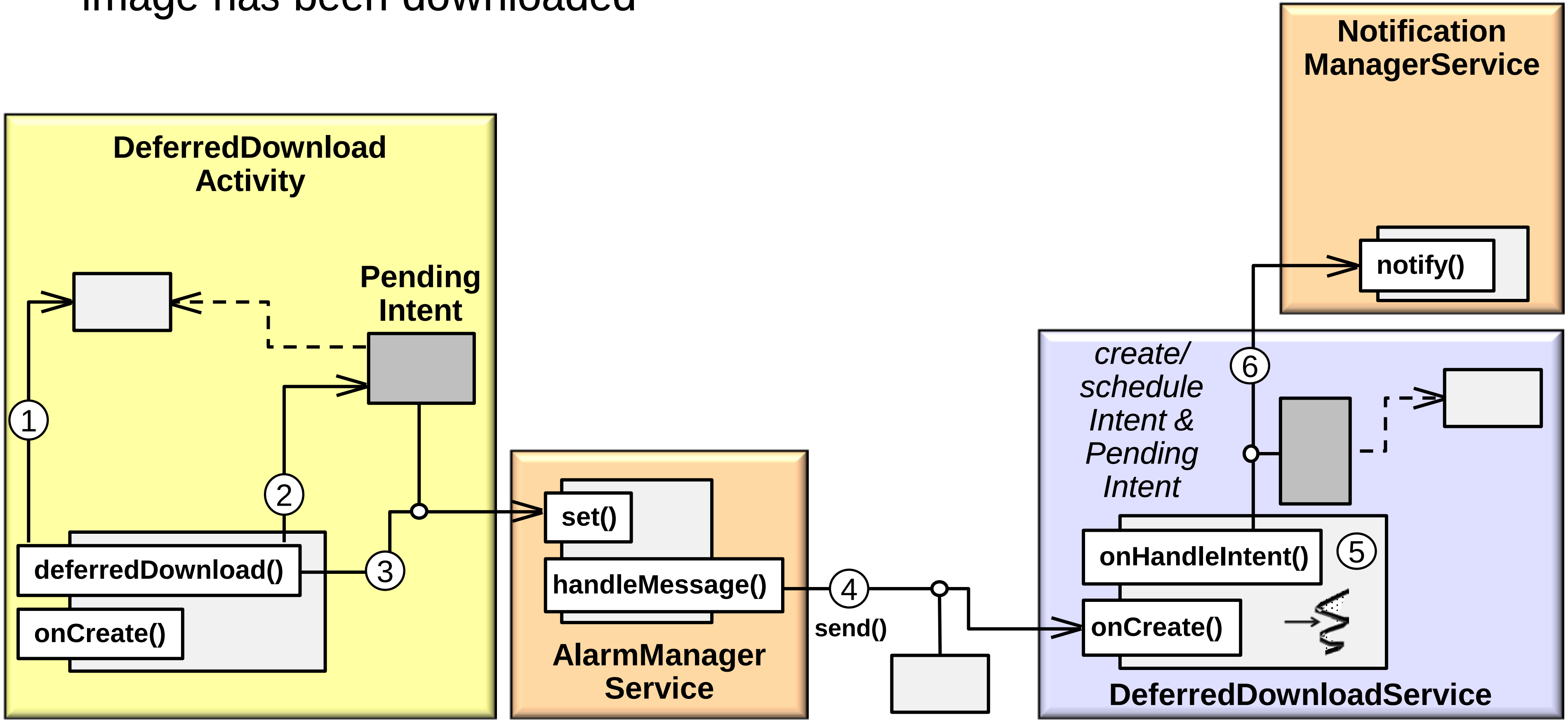
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that's registered with the Alarm Service to start DeferredDownloadService to download an image in the future
  - DeferredDownloadService uses Notification Service to inform user when the image has been downloaded



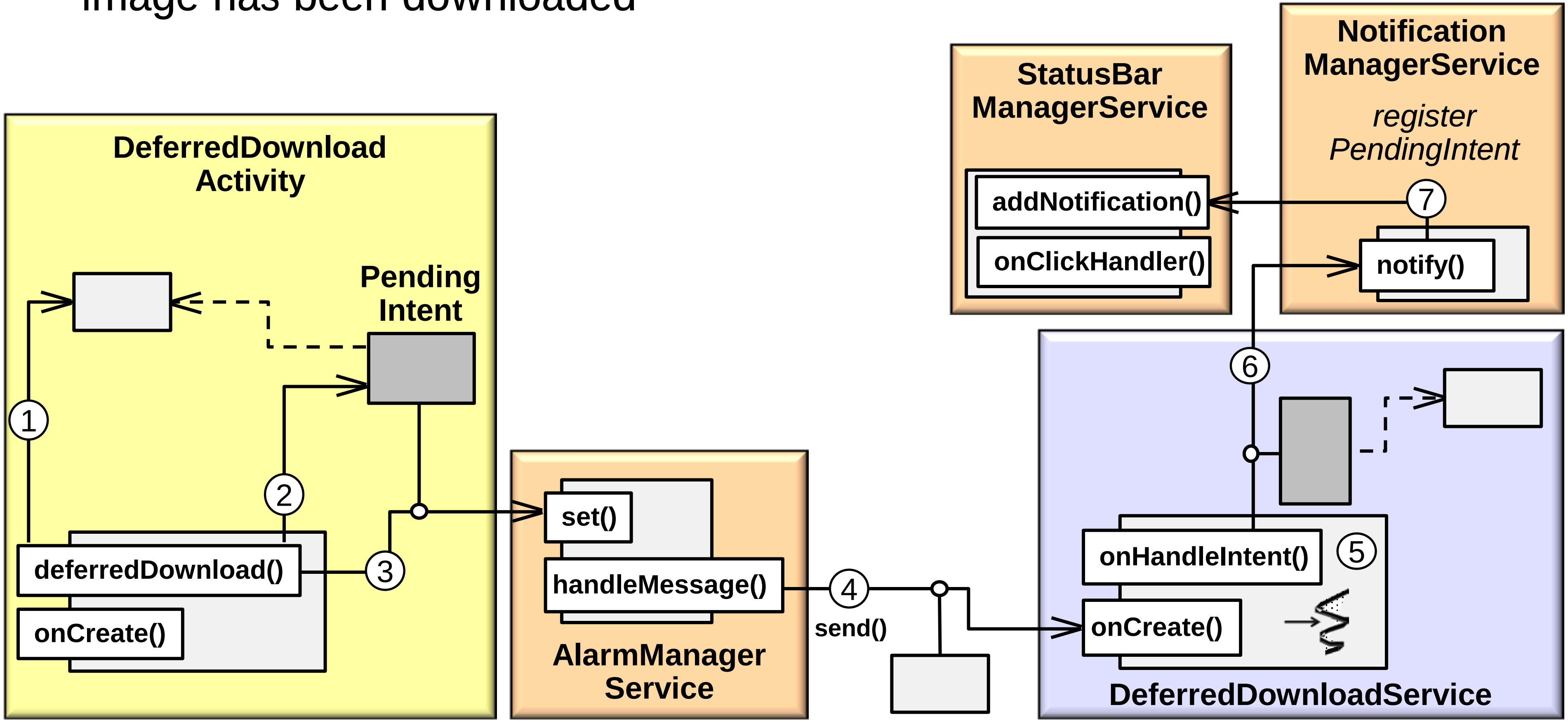
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that's registered with the Alarm Service to start DeferredDownloadService to download an image in the future
  - DeferredDownloadService uses Notification Service to inform user when the image has been downloaded



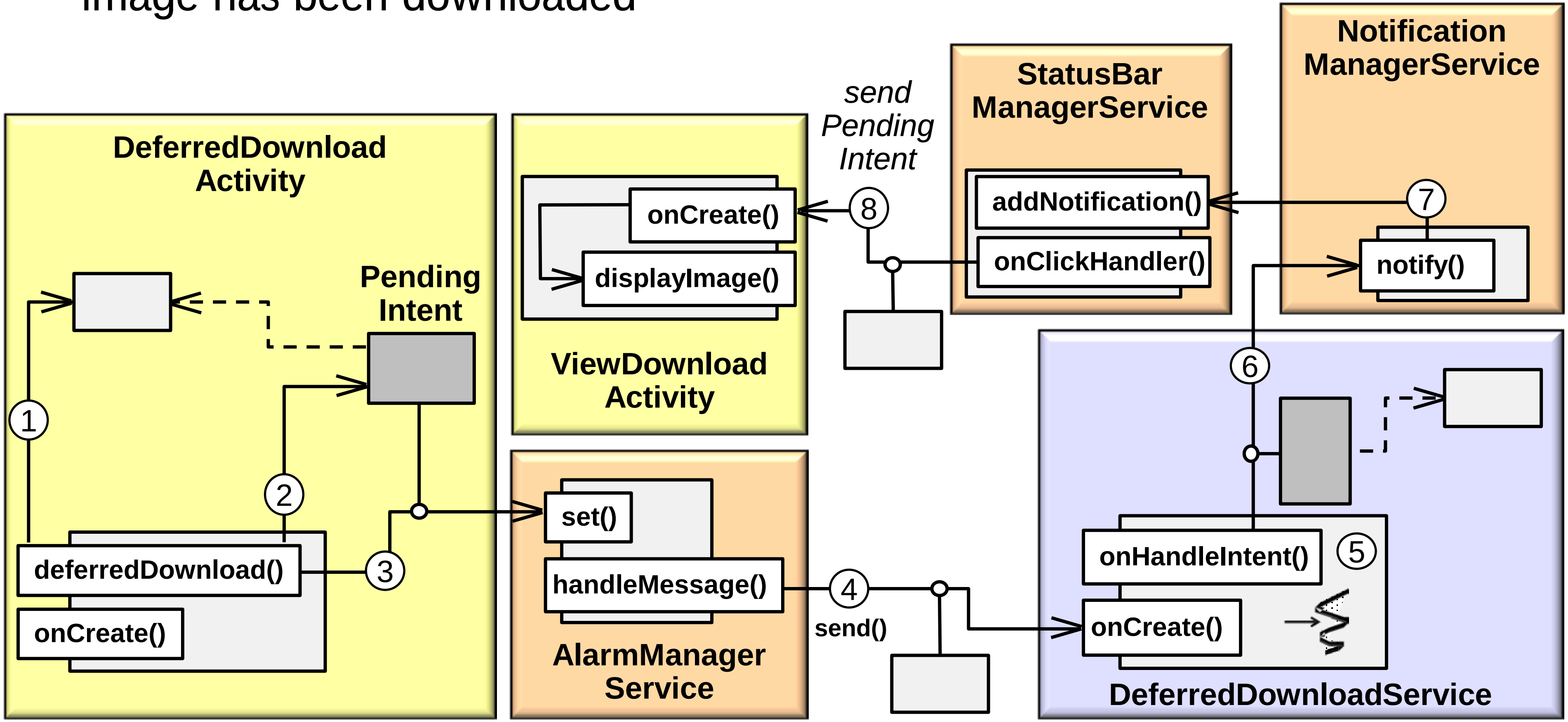
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that's registered with the Alarm Service to start DeferredDownloadService to download an image in the future
  - DeferredDownloadService uses Notification Service to inform user when the image has been downloaded



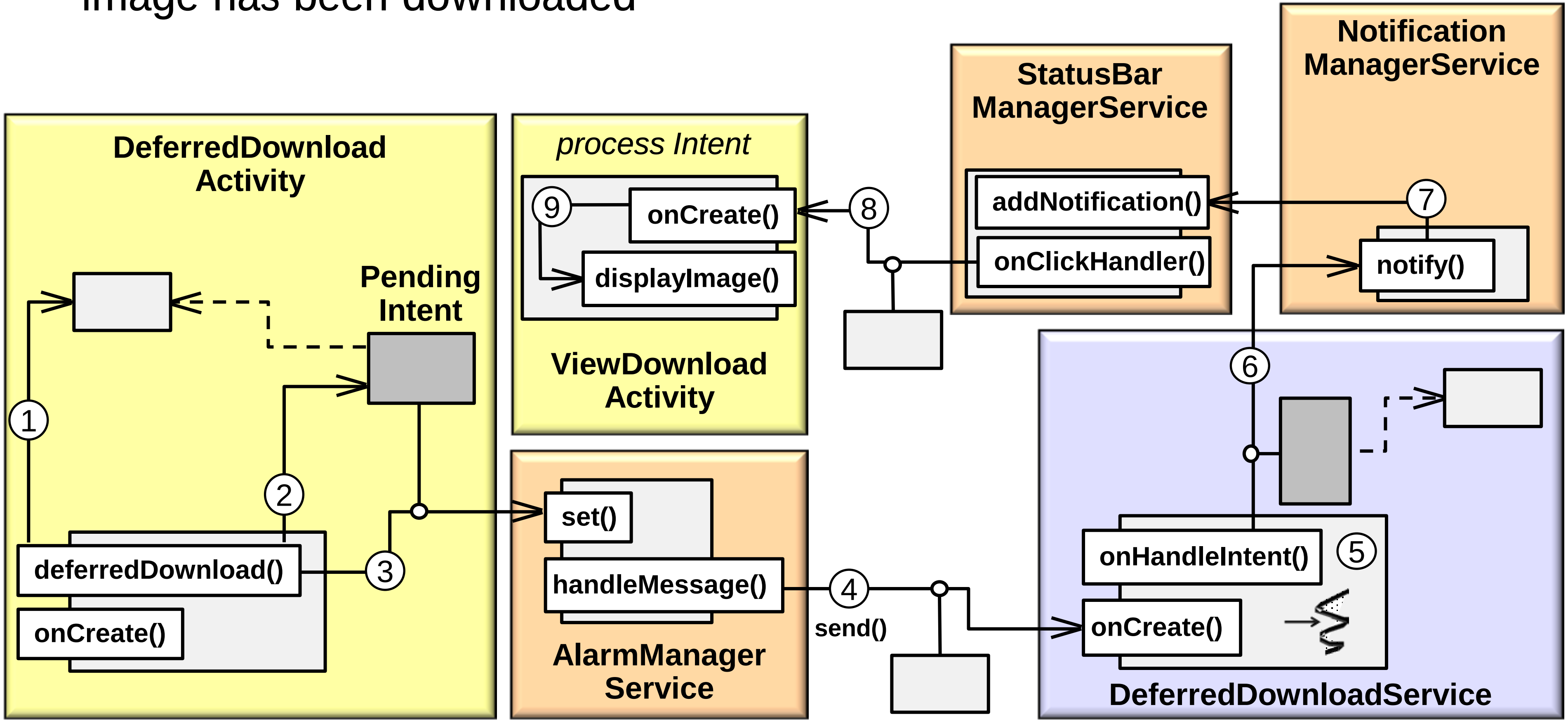
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that's registered with the Alarm Service to start DeferredDownloadService to download an image in the future
  - DeferredDownloadService uses Notification Service to inform user when the image has been downloaded



Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that's registered with the Alarm Service to start DeferredDownloadService to download an image in the future
    - DeferredDownloadService uses Notification Service to inform user when the image has been downloaded



Programming DeferredDownloadActivity

- This Activity creates a PendingIntent & schedules it with Alarm Service

```
public class DeferredDownloadActivity extends Activity {
  ...
  public void initiateDeferredDownload(View v) {
    Intent intent = new Intent(DownloadActivity.this,
                                DeferredDownloadService.class);
    PendingIntent sender = PendingIntent.getService(
                              DownloadActivity.this, 0,
                              intent, 0);
```

Create PendingIntent that starts a Service to download the image

```
    AlarmManager am =
     (AlarmManager) getSystemService(ALARM_SERVICE);

    am.set(AlarmManager.ELAPSED_REALTIME_WAKEUP,
           downloadTime,
           sender);
  }
```

Schedule an alarm to trigger the PendingIntent at the desired time

www.vogella.com/articles/AndroidNotifications/article.html#pendingintent has more

Programming DeferredDownloadService

- DeferredDownloadService uses the Notification Service to alert user when a requested image has been downloaded

```
public class DeferredDownloadService extends IntentService {
  ...
  protected void onHandleIntent(Intent intent) {
    String pathname = downloadImage(intent);
```

Code to downloading image to pathname goes here

```
    Intent viewDownloadIntent =
      new Intent(this, ViewDownloadActivity.class);
    intent.setData(pathname);
```

Prepare Intent to trigger if notification is selected

```
    PendingIntent contentIntent =
      PendingIntent.getActivity(this, 0, viewDownloadIntent,
      0);
    ...
```

Create PendingIntent to register with Notification Service

Programming DeferredDownloadService

- DeferredDownloadService uses the Notification Service to alert user when a requested image has been downloaded

```
public class DeferredDownloadService extends IntentService {
  ...
  protected void onHandleIntent(Intent intent) {
    ...
    Notification notification = new Notification.Builder(this)
      .setContentTitle("Image download complete")
      .setContentText(pathname).setSmallIcon(R.drawable.icon)
      .setContentIntent(contentIntent).build();
```

Build notification

```
    NotificationManager nm = (NotificationManager)
      getSystemService(NOTIFICATION_SERVICE);
    notification.flags |= Notification.FLAG_AUTO_CANCEL;
    notificationManager.notify(0, notification);
    ...
```

Register with the Notification Service

Programming ViewDownloadActivity

- This Activity is called when the user selects a notification that indicates the download has succeeded

```
public class ViewDownloadActivity extends Activity {
  ...
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent callersIntent = getIntent();
    String pathname = callersIntent.getData().toString();
```

Get the pathname from the Intent that started this Activity

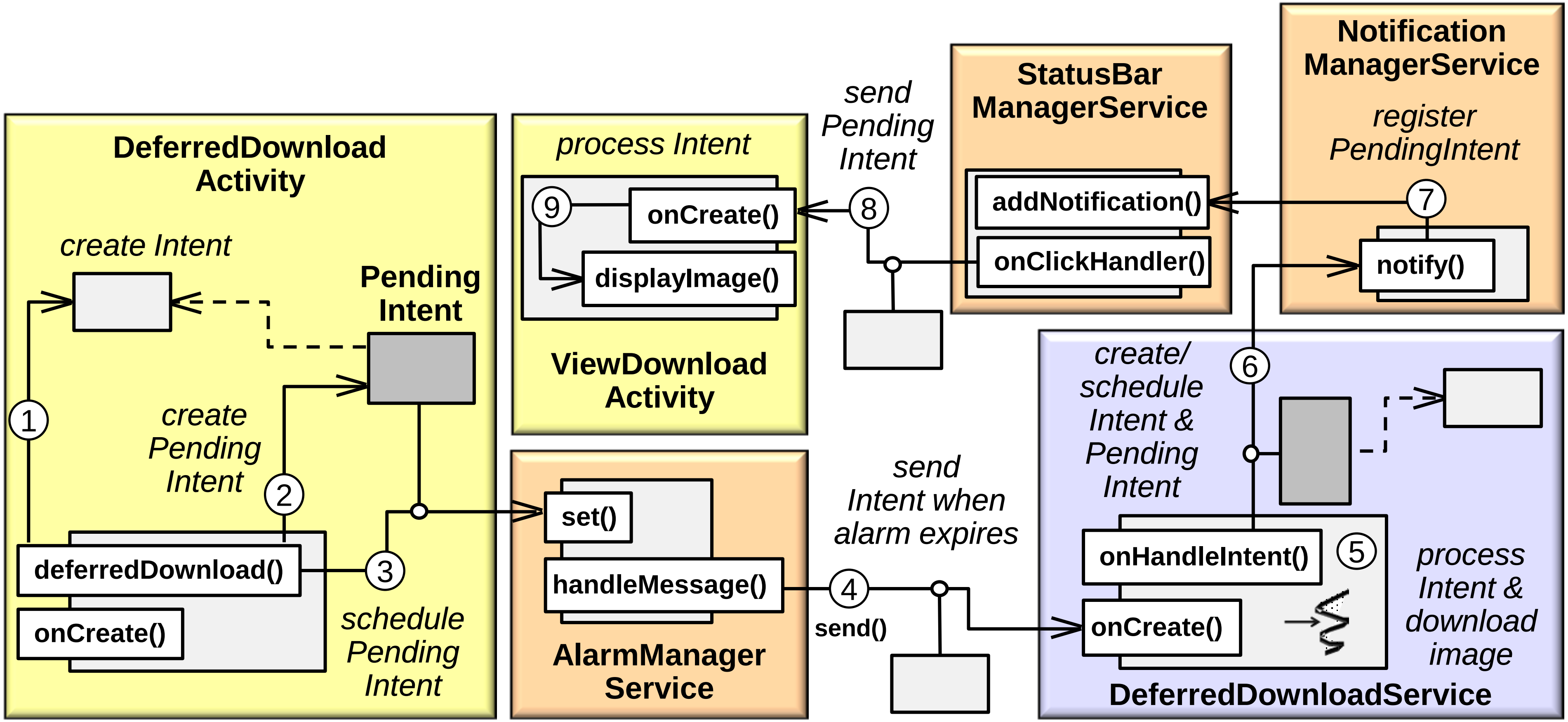
```
    displayImage(pathname);
```

Display the image

```
  }
```

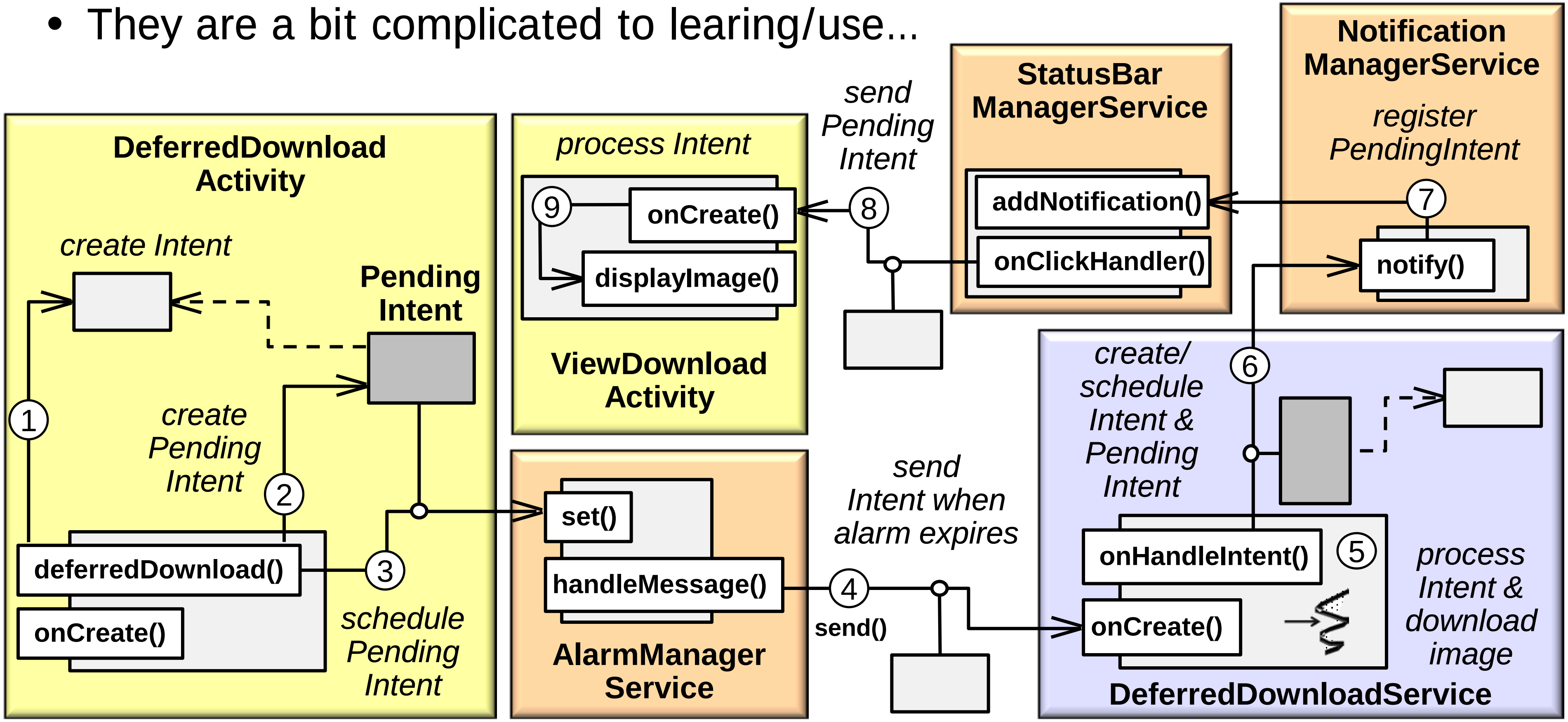
Summary

- Pending Intents provide a powerful framework for an App to delegate some processing to another App at some future time or in some other context



Summary

- Pending Intents provide a powerful framework for an App to delegate some processing to another App at some future time or in some other context
- Pending Intents can also be used to communicate from a (Started) Service back to some other Android component
  - They are a bit complicated to learing/use...


Notification ManagerService
register PendingIntent
7
notify()
StatusBar ManagerService
addNotification()
onClickHandler()
send Pending Intent
8
DeferredDownload Activity
create Intent
Pending Intent
1
create Pending Intent
2
deferredDownload()
onCreate()
3
schedule Pending Intent
process Intent
9
onCreate()
displayImage()
ViewDownload Activity
set()
handleMessage()
AlarmManager Service
4
send()
send Intent when alarm expires
create/ schedule Intent & Pending Intent
6
onHandleIntent()
5
onCreate()
process Intent & download image
DeferredDownloadService