Android Services & Local IPC: Programming Started Services

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

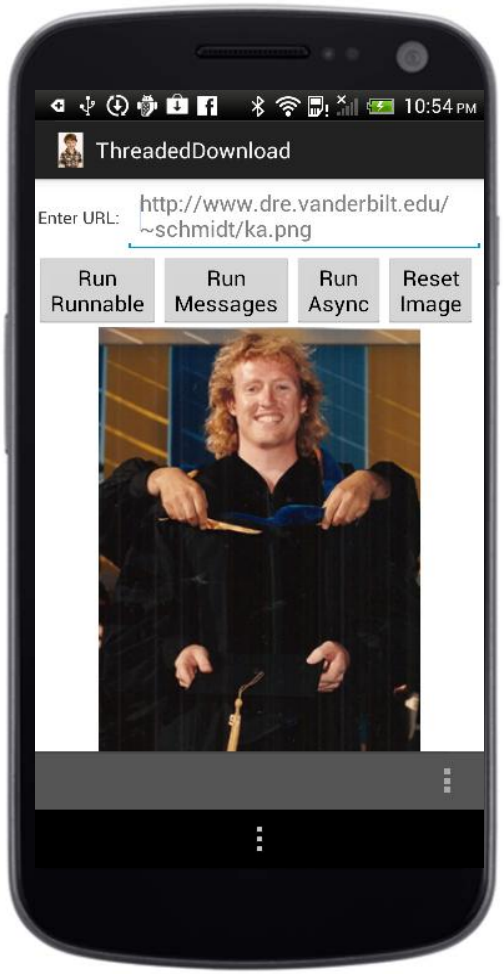
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand how to program Started Services



Programming a Started Service

- Implementing a Service is similar to implementing an Activity, e.g.:
  - Inherit from Android Service class

```
public class MusicService
              extends Service {
   public void onCreate() {
      ...
   }
   public int onStartCommand
      (Intent intent,
       int flags, int startId) {
      ...
   }
   protected void onDestroy() {
      ...
   }
   public IBinder
       onBind(Intent intent) {
      return null;
   }
...
}
```

Programming a Started Service

- Implementing a Service is similar to implementing an Activity, e.g.:
  - Inherit from Android Service class
  - Override lifecycle methods
    - May need to determine the concurrency model to use in onStartCommand()

```
public class MusicService
                extends Service {
   public void onCreate() {
      ...
   }
   public int onStartCommand
      (Intent intent,
       int flags, int startId) {
      ...
   }
   protected void onDestroy() {
      ...
   }
   public IBinder
       onBind(Intent intent) {
      return null;
   }
...
}
```

Programming a Started Service

- Implementing a Service is similar to implementing an Activity, e.g.:
  - Inherit from Android Service class
  - Override lifecycle methods
    - May need to determine the concurrency model to use in onStartCommand()
    - The onBind() method & onUnbind() aren't used for Started Services
      - You still need to provide a no-op implementation for onBind(), however

```
public class MusicService
                extends Service {
   public void onCreate() {
      ...
   }
   public int onStartCommand
      (Intent intent,
       int flags, int startId) {
      ...
   }
   protected void onDestroy() {
      ...
   }
   public IBinder
       onBind(Intent intent) {
      return null;
   }
...
}
```

Programming a Started Service

- Implementing a Service is similar to implementing an Activity, e.g.:
  - Inherit from Android Service class
  - Override lifecycle methods
  - Include the Service in the AndroidManifest.xml config file

```
<application ... >
    <activity android:name=
                 ".MusicActivity"
        ...
    </activity>


    <service android:exported=
             "false"
             android:name=
               ".BGLoggingService“
        ...
    </service>

</application
```

www.vogella.com/articles/AndroidServices/article.html has more on Services

Music Player Service Example

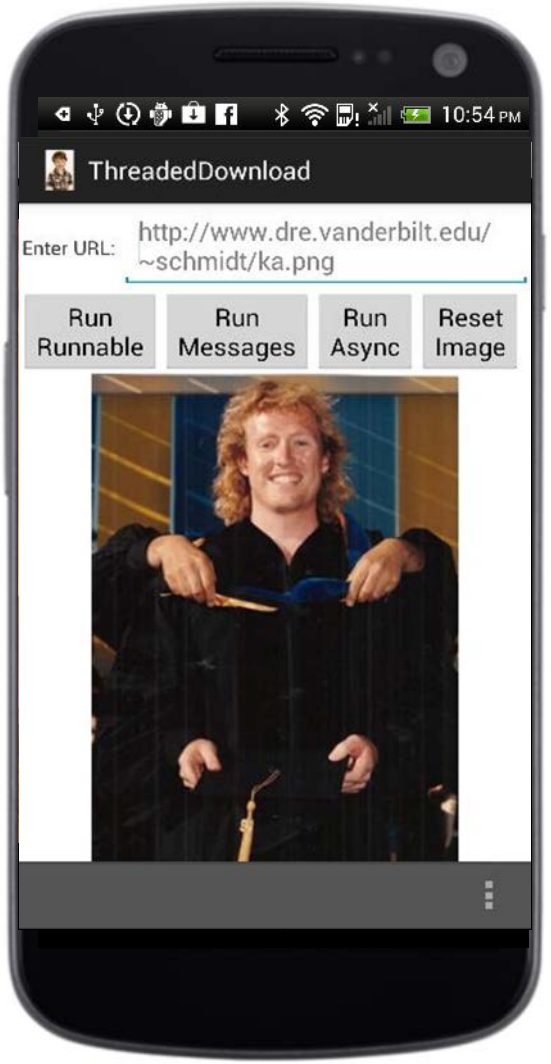
- Client Activity can play music via a Started Service

Music Player Service Example

- Client Activity can play music via a Started Service
- To start the Service a user needs to push the “Play” button

Music Player Service Example

- Client Activity can play music via a Started Service
- To start the Service a user needs to push the "Play" button
- If music is playing when the client Activity leaves the foreground, the Music Service will continue playing



Music Player Service Example

- Client Activity can play music via a Started Service
- To start the Service a user needs to push the "Play" button
- If music is playing when the client Activity leaves the foreground, the Music Service will continue playing
- To stop the Service a user needs to explicitly push the "Stop" button

Music Player Activity Implementation

```
public class MusicActivity extends Activity {
  ...
  public void play (View src) {
    Intent intent = new Intent(MusicActivity.this,
                               MusicService.class);
    intent.putExtra("SongID", R.raw.braincandy);
```

Add the song to play as an "extra"

```
    startService(intent);
  }
```

Launch the Started Service that handles this Intent

```
  public void stop (View src) {
    Intent intent = new Intent(MusicActivity.this,
                               MusicService.class);
    stopService (intent);
  }
}
```

Stop the Started Service

Music Player Service Implementation

```
public class MusicService extends Service {
  MediaPlayer player;

  public int onStartCommand (Intent intent,
                             int flags, int startid) {
    player = MediaPlayer.create(this,
                                intent.getIntExtra("SongID",
                                                   0));
    player.setLooping(false);
    player.start();

    return START_NOT_STICKY;
  }

  public void onDestroy() { player.stop(); }
}
```

Inherit from Service class

Extract the resid from the "extra" & create a MediaPlayer

Start playing the song (doesn't block)

Don't restart Service if it shuts down

Stop player when Service is destroyed

developer.android.com/reference/android/media/MediaPlayer.html has more

AndroidManifest.xml File

```
<application ... >

    <activity android:name=".MusicActivity"
                                  android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name=
            "android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <service android:exported="true"
             android:name=".MusicService" />

</application>
```

Service is usable by components external to this application

developer.android.com/guide/topics/manifest/service-element.html#exported

Analysis of the Music Player Service Example

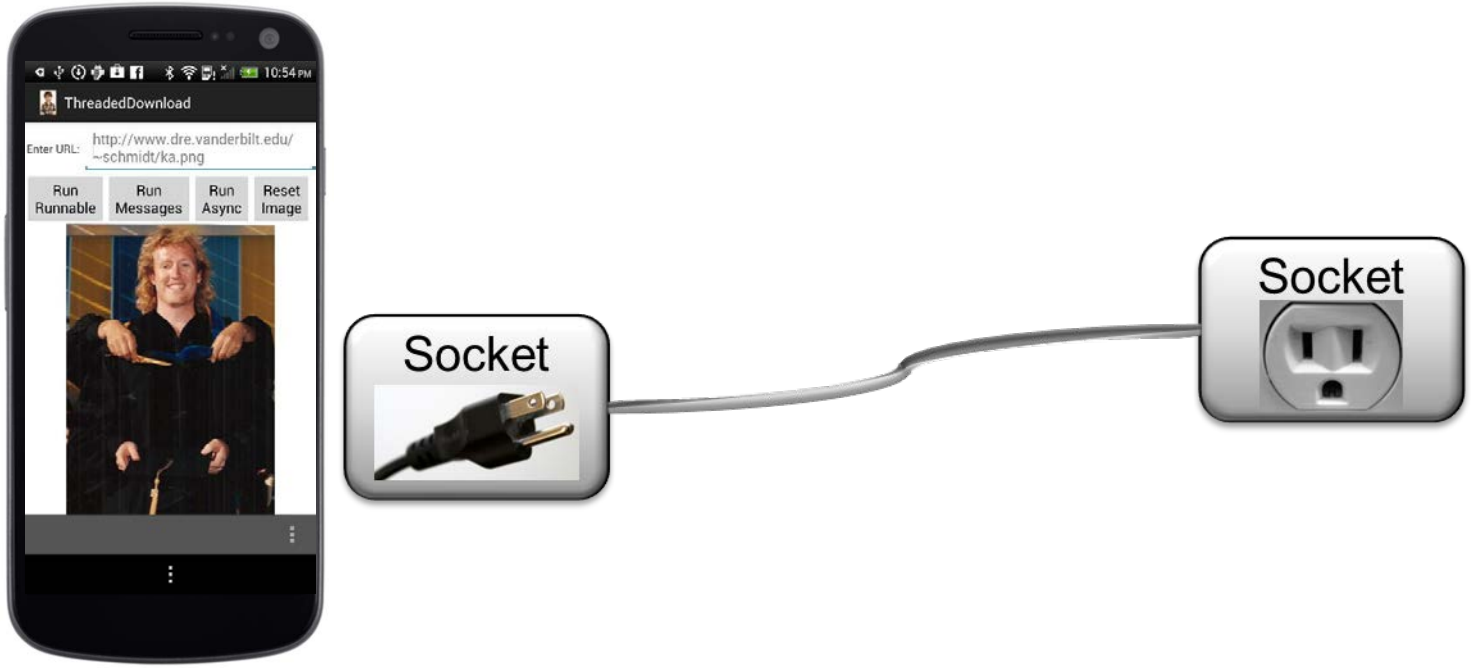
- This is a very simple example of a Started Service
  - In particular, it runs in the UI Thread, but doesn't block due to the behavior of MusicPlayer.start()
  - Also, there's no communication from the Service back to the Activity that invoked it!

Analysis of the Music Player Service Example

- This is a very simple example of a Started Service
- Services with long-running operations typically need to run in separate Thread(s)

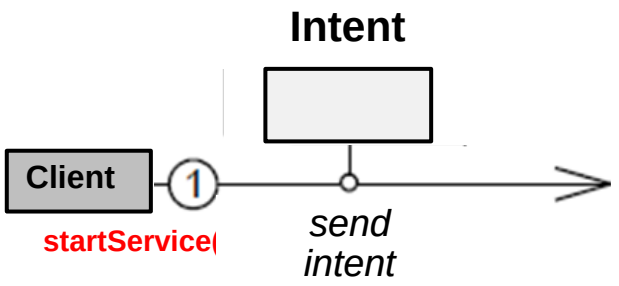
Download Service Example

- Client Activity requests a Started Service to download a file from a server



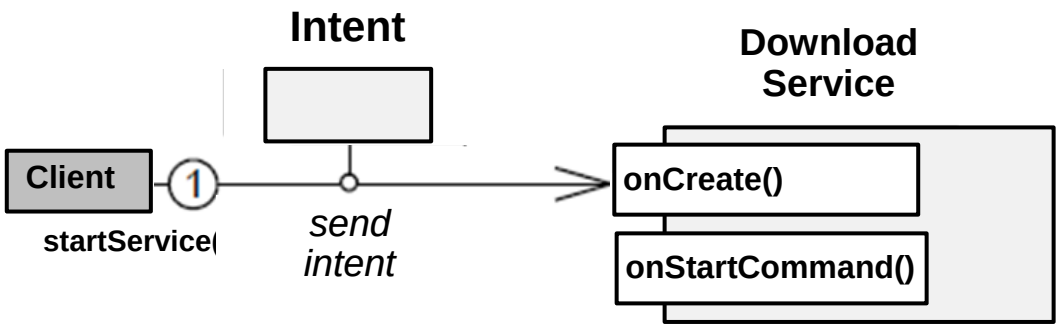
Download Service Example

- Clients send Intents via calls to startService()



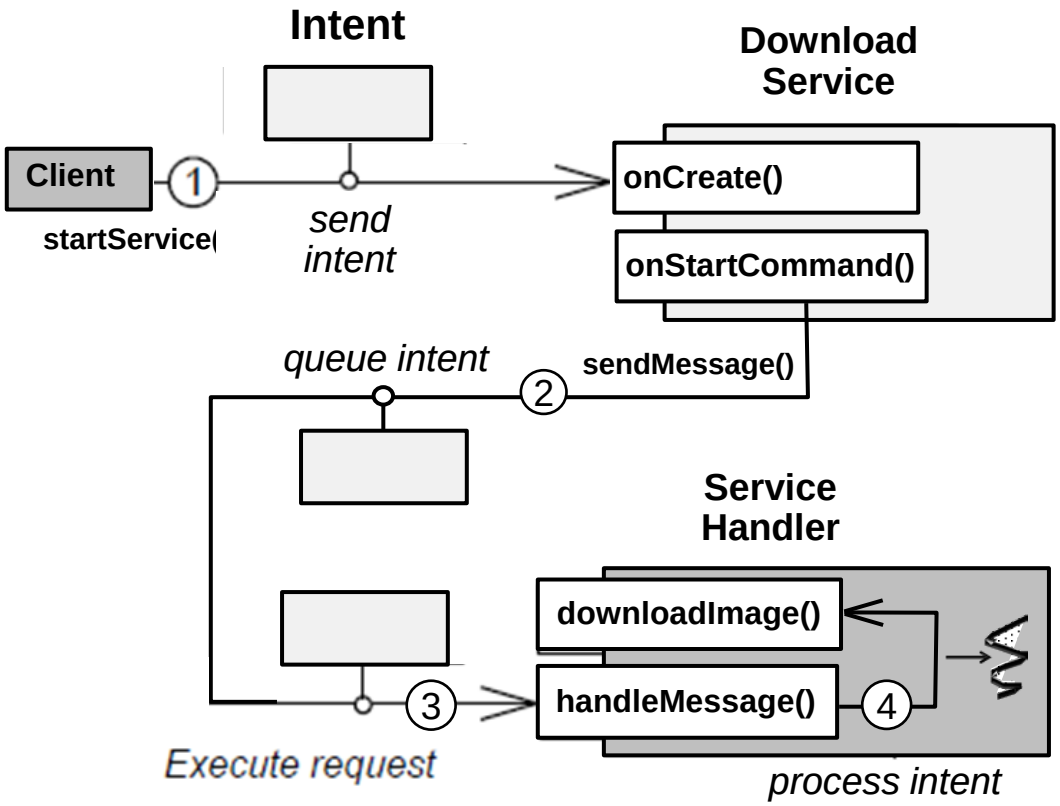
Download Service Example

- Clients send Intents via calls to startService()
- The DownloadService is started on-demand
  - Based on the *Activator* pattern



Download Service Example

- Clients send Intents via calls to startService()
- The DownloadService is started on-demand
- The DownloadService handles each Intent in turn the worker thread in a ServiceHandler & stops itself when it runs out of work
  - This implementation of the *Command Processor* pattern offloads tasks from an app's main thread to a single worker thread



Download Activity Implementation

```
public class DownloadActivity extends Activity {
  ...
  public void onClick(View v) {
     Intent intent = new Intent(DownloadActivity.this,
                                DownloadService.class);
     ...
     intent.setData(Uri.parse(editText.getText().toString()));
```

Add the URL to the download as data

```
     startService(intent);
  }
```

Launch the Started Service that handles this Intent

```
  ...

}
```

We don't show the code for processing the downloaded file

Download Service Implementation

```
public class DownloadService extends Service {
  private volatile Looper mServiceLooper;
  private volatile ServiceHandler mServiceHandler;

  private final class ServiceHandler extends Handler {
    public ServiceHandler(Looper looper) { super(looper); }

    public void handleMessage(Message msg) {
      downloadImage((Intent) msg.obj);

      stopSelf(msg.arg1);
    }

    public void downloadImage(Intent intent) { /* ... */ }
  }
  ...
```

Handler that receives messages from the thread

Dispatch a callback hook method to download a file

Stop the service using the startId, so that we don't stop the service in the middle of handling another job

Download the image & notify the client

Download Service Implementation

```
public class DownloadService extends Service {
  ...
  public void onCreate() {
    super.onCreate();
```

Start up the thread running the service, which we create a separate Thread because the Service normally runs in the process's UI Thread that we don't want to block

```
    HandlerThread thread = new HandlerThread("DownloadService");
    thread.start();
```

Get the HandlerThread's Looper & use it for our Handler

```
    mServiceLooper = thread.getLooper();
    mServiceHandler = new ServiceHandler(mServiceLooper);
 }
```

Download Service Implementation

```
public class DownloadService extends Service {
  ...
  public int onStartCommand(Intent intent, int f, int startId) {
```

For each start request, send a message to start a job & deliver the start ID so we know which request we're stopping when we finish the job

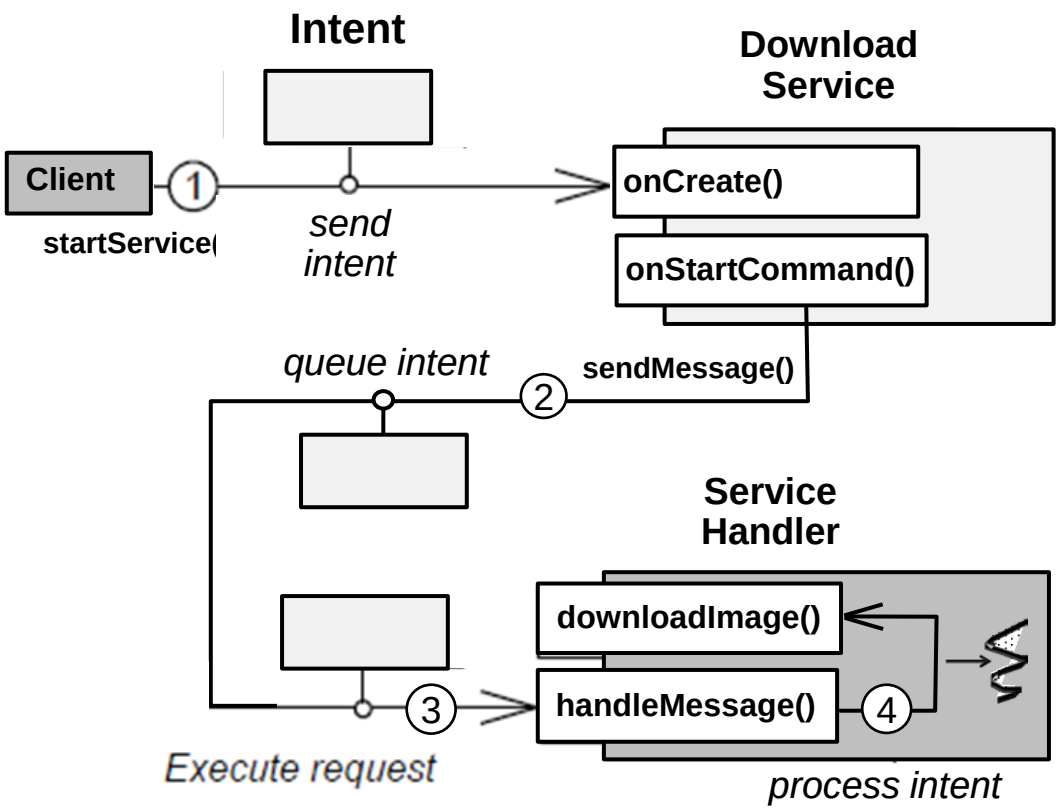
```
    Message msg = mServiceHandler.obtainMessage();
    msg.arg1 = startId;
    msg.obj = intent;
    mServiceHandler.sendMessage(msg);
    return START_NOT_STICKY;
  }

  public void onDestroy() {
    mServiceLooper.quit();
  }
}
```

Shutdown the looper

It's instructive to consider how to extend this example to run in a thread pool

Analysis of the Download Service Example

- The worker thread solution shown here is a common Android Service idiom that implements the *Command Processor* pattern
  - In fact, it's so common that Android provides a reusable framework that simplifies the use of this pattern!



Logging Service Example

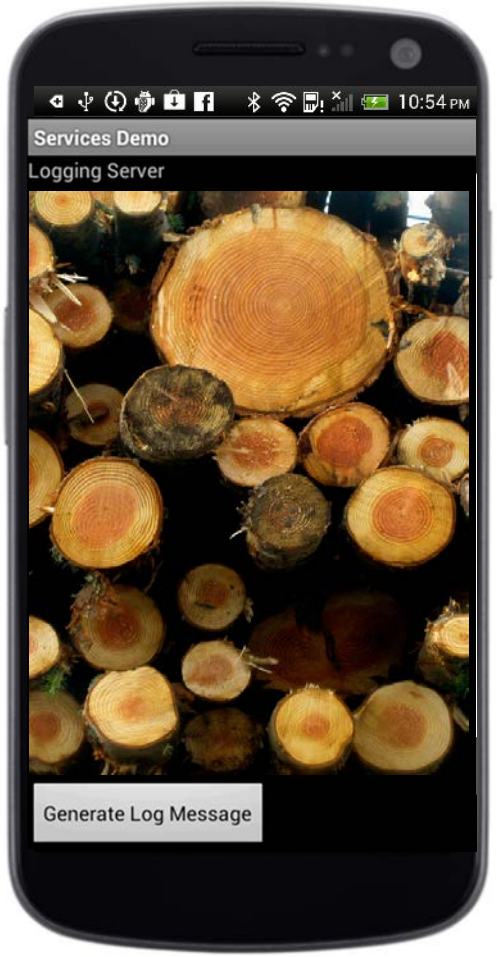
- The Logging Service extends the IntentService to offload logging operations from an app's UI Thread


```
public class LoggingService
     extends IntentService {
  protected abstract void onHandleIntent
            (Intent intent);
}
```

L...	Time	PID	TID	Application	Tag	Text
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service destroyed
I	09-19 12:...	612	631	course.examples.Ser...	Logging...	Log this message
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service created
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service command started

Logging Service Example

- The Logging Service extends the IntentService to offload logging operations from an app's UI Thread
- Clients send commands (expressed as Intents) via calls to startService()



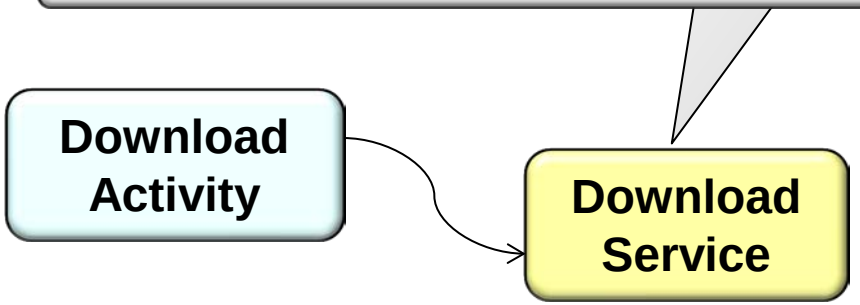

```
Intent intent = new Intent
  (this, LoggingService.class));
intent.putExtra("LogMsg", "hello world");
startService(intent);
```

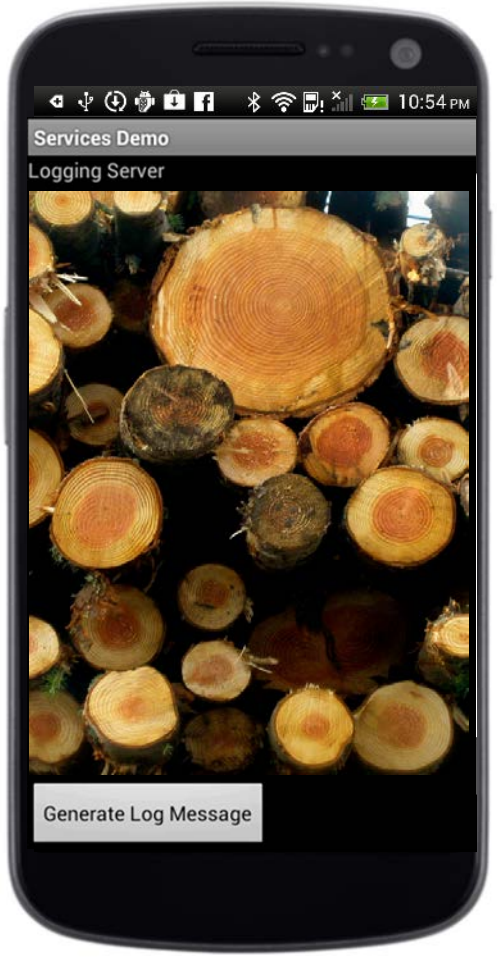
Download Activity

Logging Service Example

- The Logging Service extends the IntentService to offload logging operations from an app's UI Thread
- Clients send commands (expressed as Intents) via calls to startService()
- The LoggingService subclass handle intents in a worker thread asynchronously

```
public class LoggingService extends
              IntentService {
  void onHandleIntent(Intent intent)
  { ... }
}
```





Logging Service Example

- The Logging Service extends the IntentService to offload logging operations from an app's UI Thread
- Clients send commands (expressed as Intents) via calls to startService()
- The LoggingService subclass handle intents in a worker thread asynchronously


```
public class LoggingService extends
              IntentService {
  void onHandleIntent(Intent intent)
  { ... }
}
```

Handler

Download Activity

Download Service

Android starts the Service as needed, which internally spawns a worker thread that handles a queue of intents

Logging Service Example

- The Logging Service extends the IntentService to offload logging operations from an app's UI Thread
- Clients send commands (expressed as Intents) via calls to startService()
- The LoggingService subclass handle intents in a worker thread asynchronously


```
public class LoggingService extends
              IntentService {
  void onHandleIntent(Intent intent)
  { ... }
}
```

Handler

Download Activity

Download Service

The IntentService calls this hook method from the worker thread to handle each intent that started the Service

Logging Service Example

- The Logging Service extends the IntentService to offload logging operations from an app's UI Thread
- Clients send commands (expressed as Intents) via calls to startService()
- The LoggingService subclass handle intents in a worker thread asynchronously


```
public class LoggingService extends
              IntentService {
  void onHandleIntent(Intent intent)
  { ... }
}
```

Handler

Download Activity

Download Service

When there are no more intents to handle the IntentService stops itself automatically

Logging Activity Implementation

```
public class BGLoggingActivity extends Activity {
  public void onCreate(Bundle savedInstanceState) {
  ...
    buttonStart.setOnClickListener(new OnClickListener() {

      public void onClick(View v) {
        Intent intent = new Intent(BGLoggingActivity.this,
                                   BGLoggingService.class);

        intent.putExtra("LogMsg",
                        "Log this message");
```

Add the message to log as an "extra"

```
        startService(intent);
      }
    });
  }
}
```

Launch the Started Service that handles this Intent

Logging Service Implementation

```
public class BGLoggingService extends IntentService {
...
```

Inherit from IntentService class

```
  public int onStartCommand(Intent intent, int flags,
                            int startId) {
     super.onStartCommand(intent, flags, startId);
     return START_NOT_STICKY;
  }
```

Don't restart this Service if it's shutdown

```
  protected void onHandleIntent(Intent intent) {
    ...
    Log.i(TAG, intent.getCharSequenceExtra
         ("LogMsg").toString());
  }
  ...
}
```

This hook method runs in a worker thread & logs the data

Note the "inversion of control" in the Android Service framework

AndroidManifest.xml File

```
<application ... >

    <activity android:name=".BGLoggingActivity"
                                  android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name=
            "android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <service android:exported="false"
             android:name=".BGLoggingService" />

</application>
```

Service is only usable by components in this application

developer.android.com/guide/topics/manifest/service-element.html#exported

AndroidManifest.xml File

```
<application ... >

    <activity android:name=".BGLoggingActivity"
                                  android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name=
            "android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>


    <service android:exported="false"
             android:name=".BGLoggingService"
             android:process=":myProcess"/>

</application
```

Instruct Android to run the BGLoggingService in its own process

developer.android.com/guide/topics/manifest/service-element.html#proc

Analysis of the Logging Service Example

- The LoggingService is an intentionally simplified example

L...	Time	PID	TID	Application	Tag	Text
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service destroyed
I	09-19 12:...	612	631	course.examples.Ser...	Logging...	Log this message
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service created
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service command started

Analysis of the Logging Service Example

- The LoggingService is an intentionally simplified example
- You don't need to implement it as an IntentService (or even as a Service)
  - You could simply do the logging in a new Thread or ignore concurrency altogether!

L...	Time	PID	TID	Application	Tag	Text
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service destroyed
I	09-19 12:...	612	631	course.examples.Ser...	Logging...	Log this message
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service created
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service command started

Analysis of the Logging Service Example

- The LoggingService is an intentionally simplified example
- You don't need to implement it as an IntentService (or even as a Service)
- In general, use a Service (or IntentService) when you want to run a component even when a user is not interacting with the app that hosts the Service

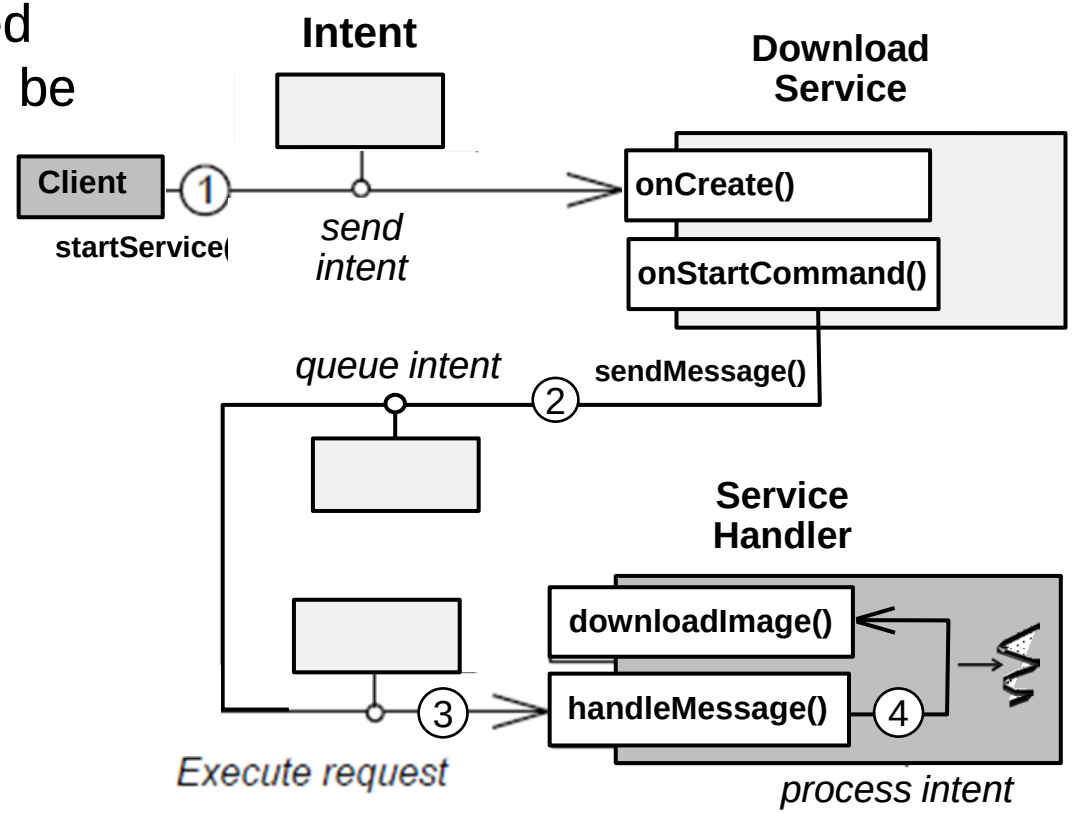
L...	Time	PID	TID	Application	Tag	Text
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service destroyed
I	09-19 12:...	612	631	course.examples.Ser...	Logging...	Log this message
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service created
I	09-19 12:...	612	612	course.examples.Ser...	Logging...	Service command started

Summary

- Programming Services & Intent Services is relatively straightforward
  - Though they sometimes can be overkill...

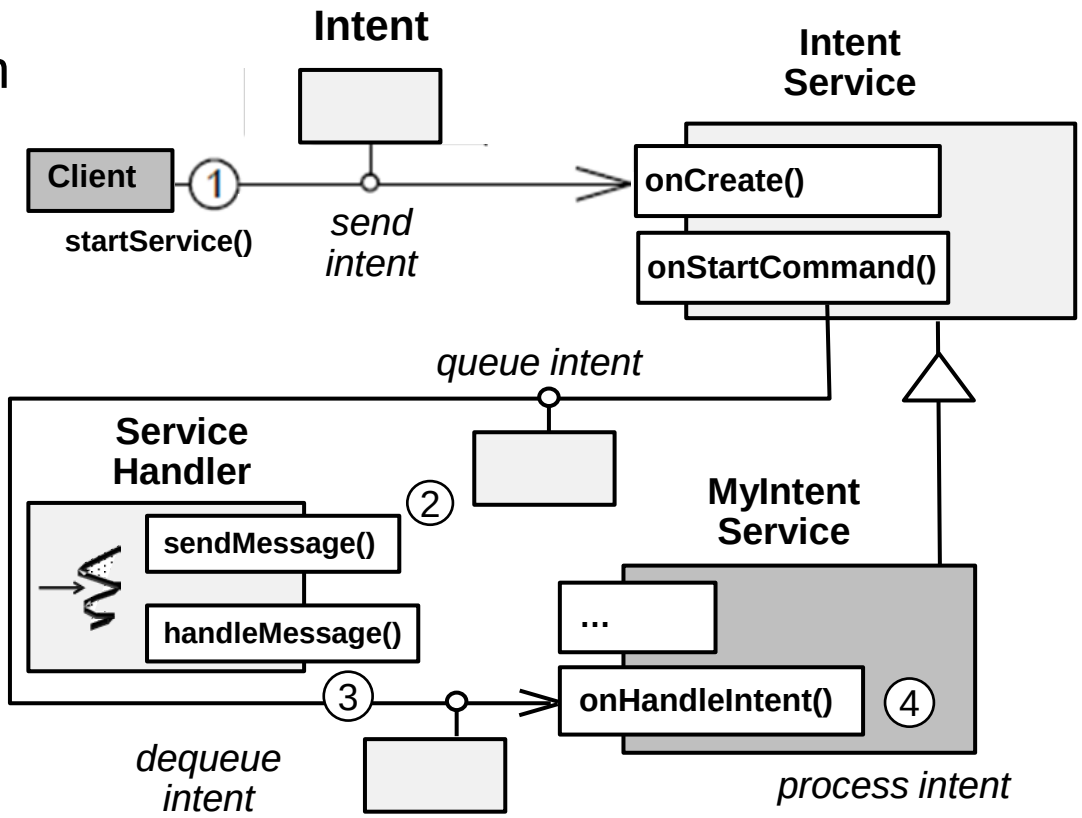
Summary

- Programming Services & Intent Services is relatively straightforward
- The Service class uses the app's UI Thread by default
  - Implementing a multi-threaded service should therefore often be made by extending Service class directly & spawning one or more threads



Summary

- Programming Services & Intent Services is relatively straightforward
- The Service class uses the app's UI Thread by default
- IntentService creates a worker thread & uses that thread to run the service
  - IntentService also creates a queue that passes one intent at a time to onHandleIntent()



Summary

- Programming Services & Intent Services is relatively straightforward
- The Service class uses the app's UI Thread by default
- IntentService creates a worker thread & uses that thread to run the service
- The Service class needs a manual stop via stopSelf() or stopService()
  - Conversely, IntentService automatically stops itself when there are no more intents in its queue

Android Services & Local IPC: Programming Started Services with Messengers

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

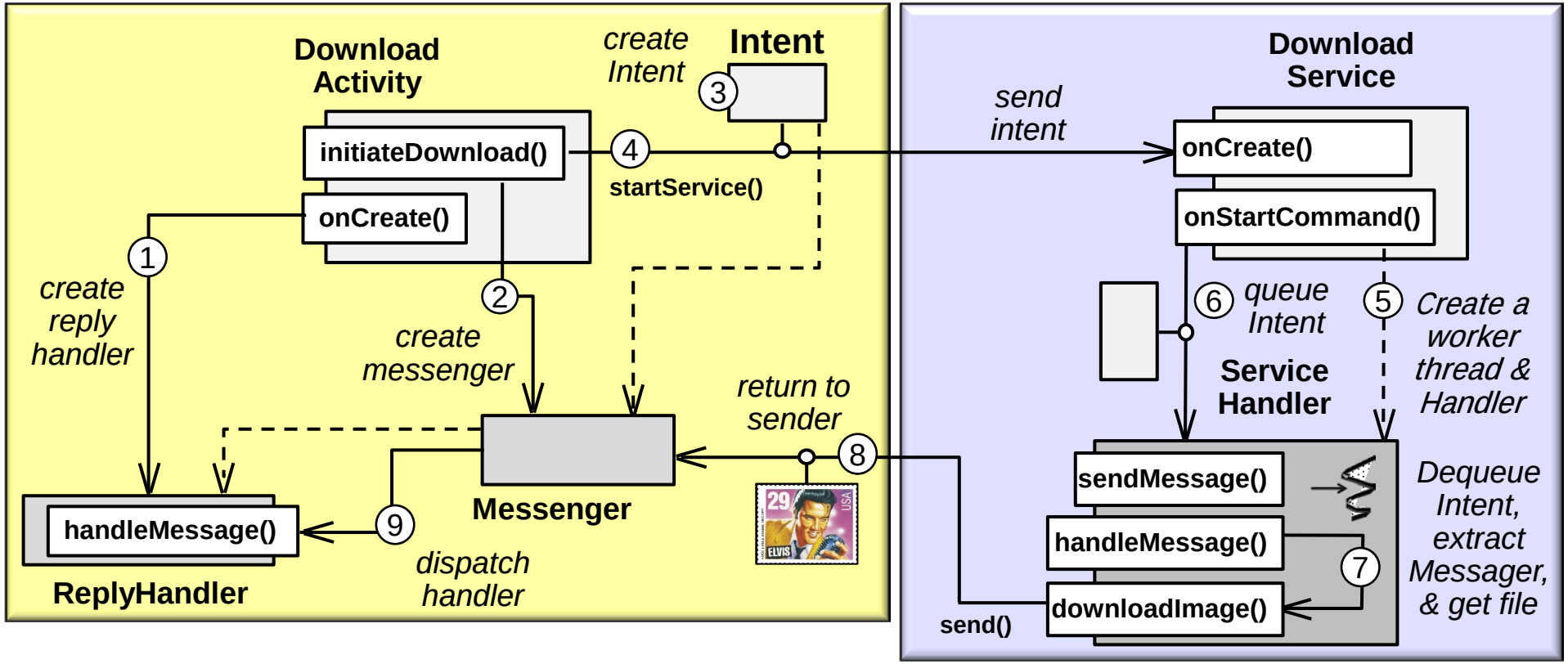
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

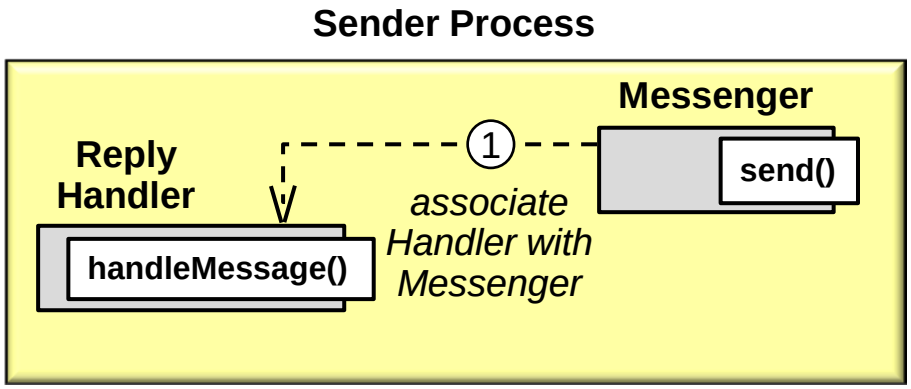
Learning Objectives in this Part of the Module

- Understand how to use Messengers to communicate from Started Services back to their invoking Activities
  - Provides an interface for IPC with remote processes without using AIDL



Overview of Messengers

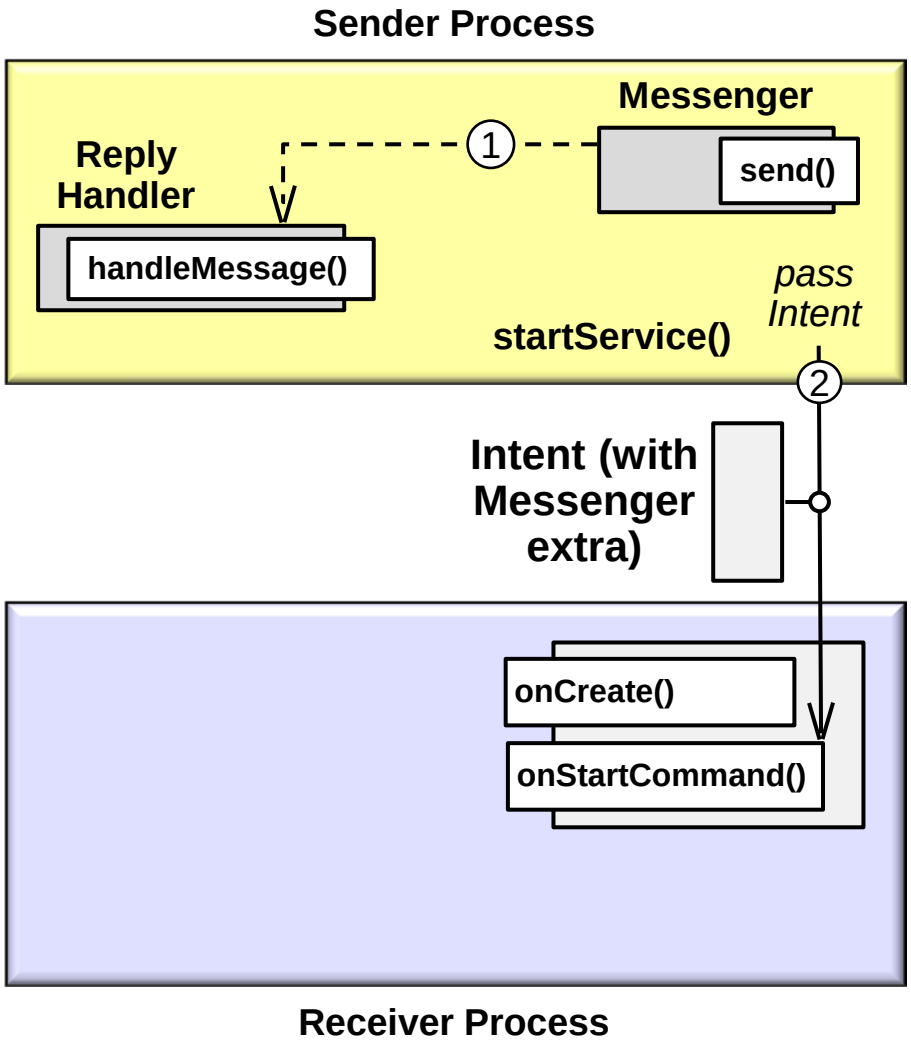
- A Messenger provides a reference to a Handler that others can use to send messages to it



developer.android.com/reference/android/os/Messenger.html has more info

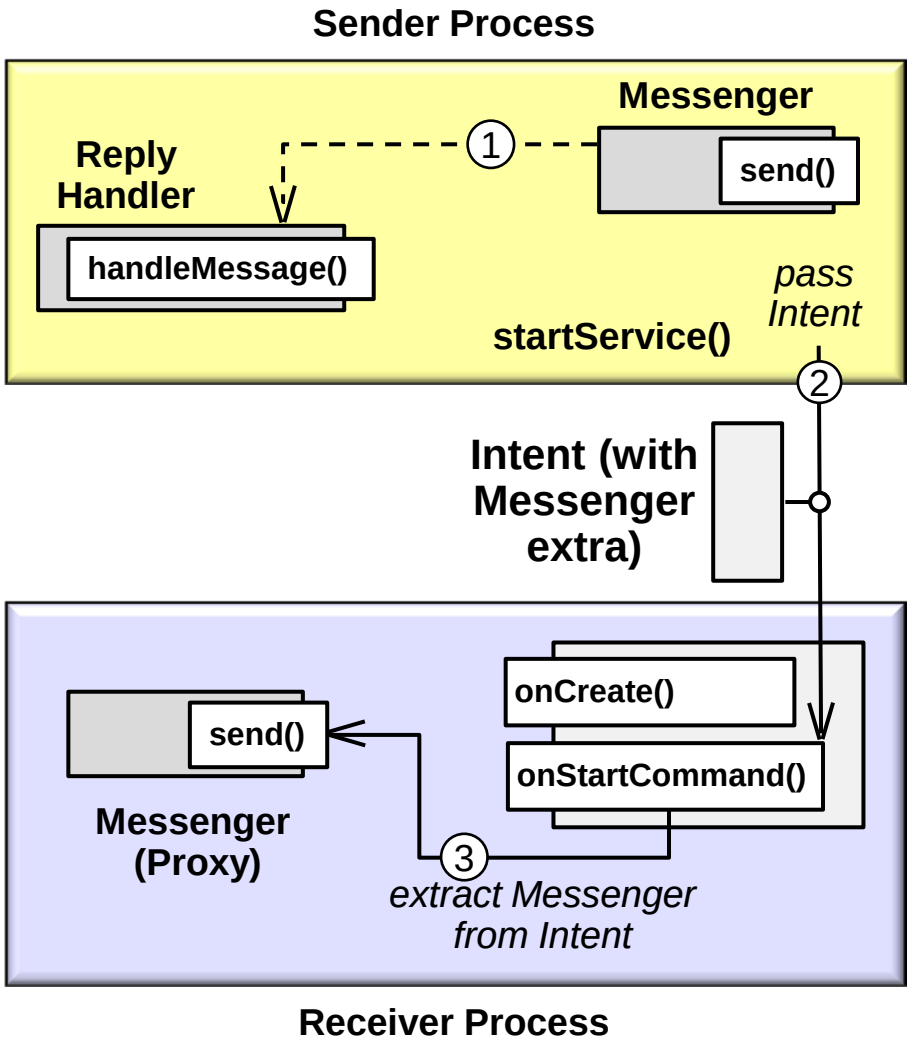
Overview of Messengers

- A Messenger provides a reference to a Handler that others can use to send messages to it
- An Activity can create a Messenger pointing to a Handler in one process & then pass that Messenger to another process



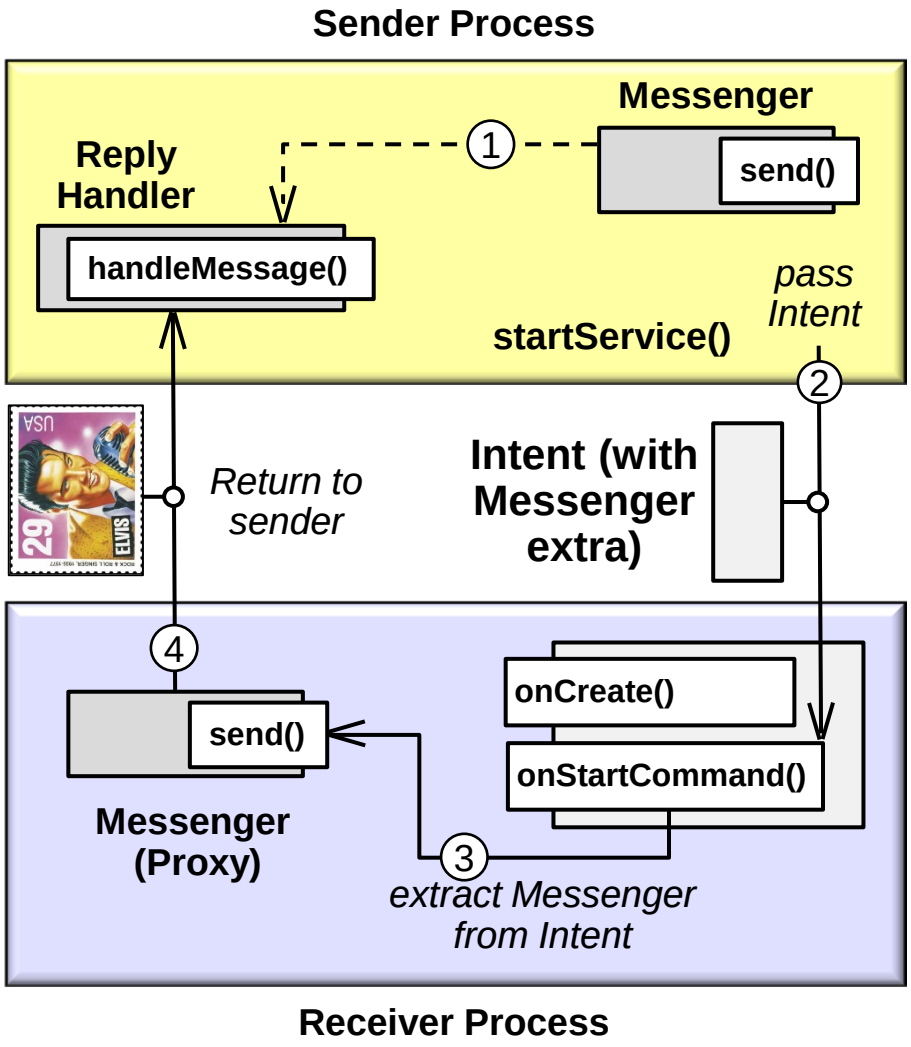
Overview of Messengers

- A Messenger provides a reference to a Handler that others can use to send messages to it
- An Activity can create a Messenger pointing to a Handler in one process & then pass that Messenger to another process
- The receiver then does several things
  - Obtains the Messenger



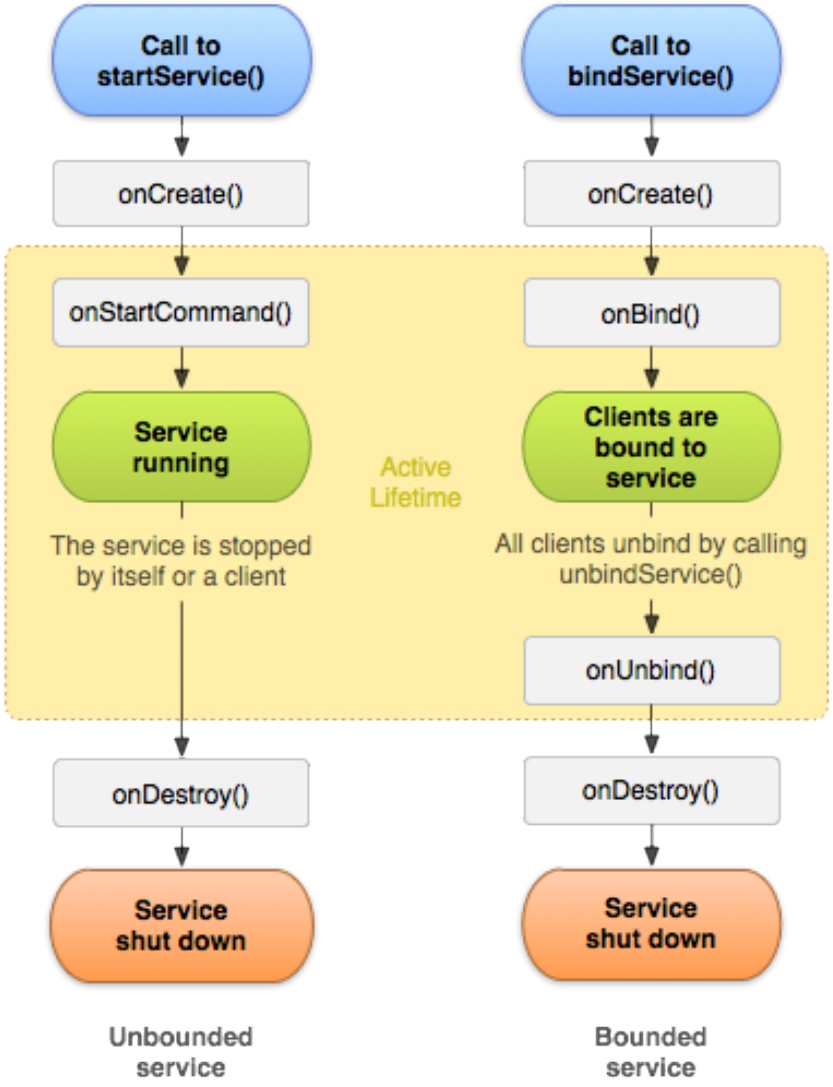
Overview of Messengers

- A Messenger provides a reference to a Handler that others can use to send messages to it
- An Activity can create a Messenger pointing to a Handler in one process & then pass that Messenger to another process
- The receiver then does several things
  - Obtains the Messenger
  - Returns the results back to the sender process



Overview of Messengers

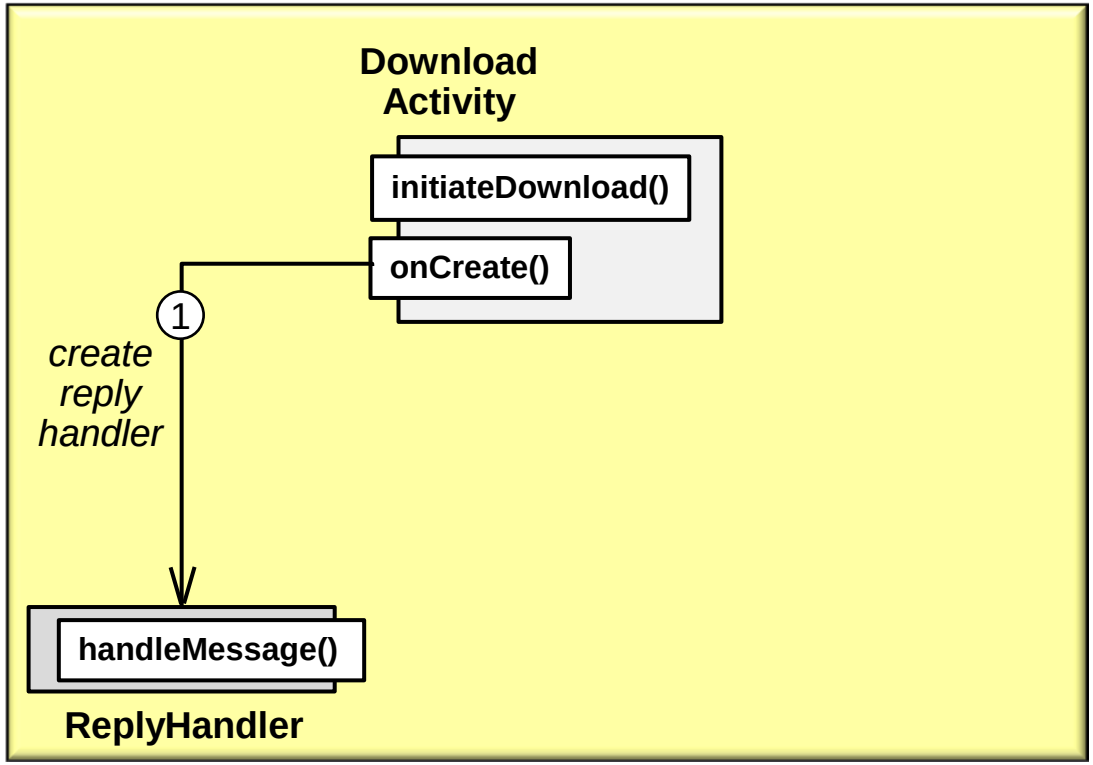
- A Messenger provides a reference to a Handler that others can use to send messages to it
- An Activity can create a Messenger pointing to a Handler in one process & then pass that Messenger to another process
- The receiver then does several things
- You can use Messengers with both Bound & Started Services to implement the *Command Processor* pattern



www.dre.vanderbilt.edu/~schmidt/PDF/CommandProcessor.pdf has more info

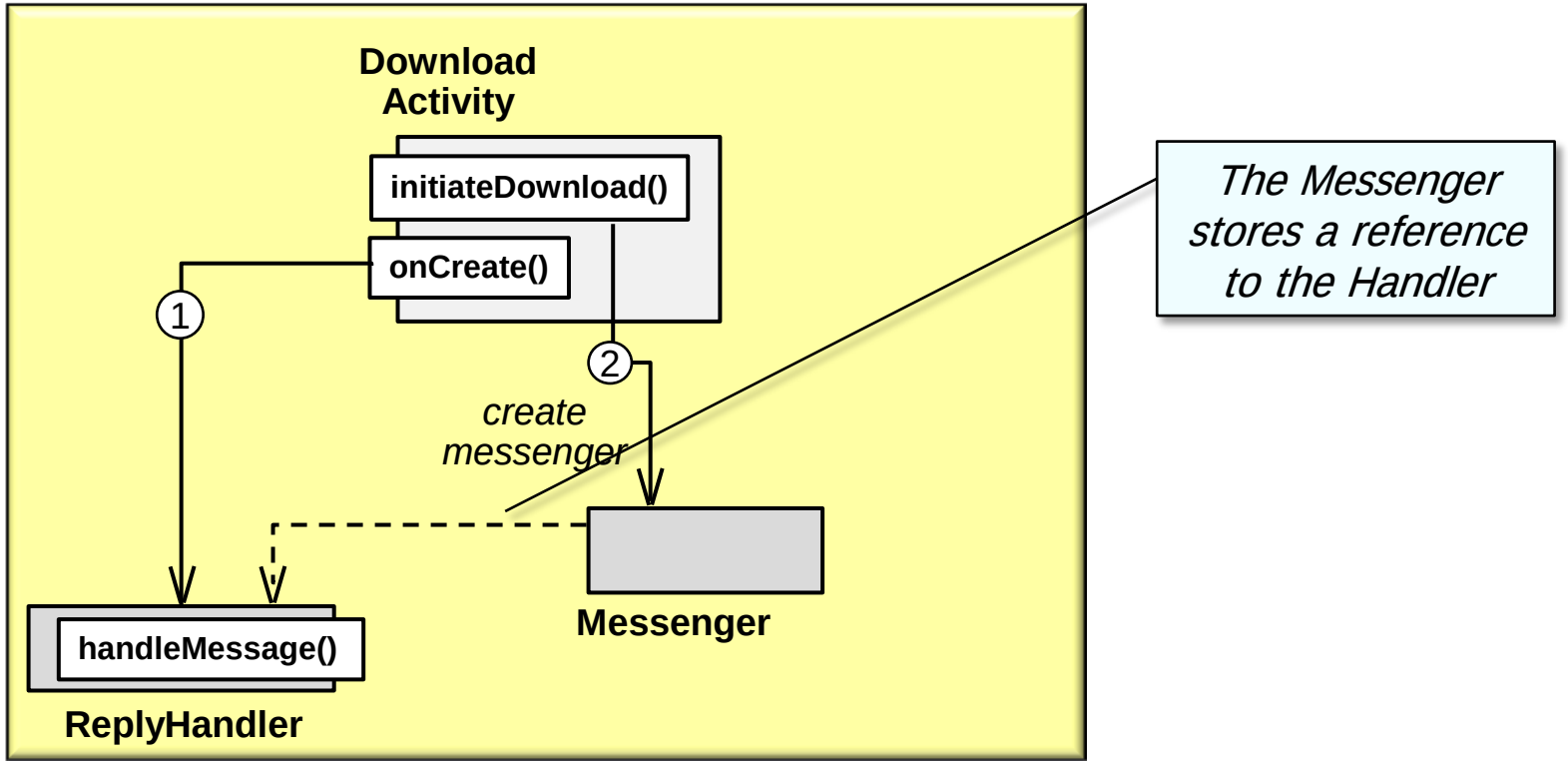
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an "extra" to the Intent used to activate the DownloadService
  - DownloadService uses the Messenger to reply back to the Activity



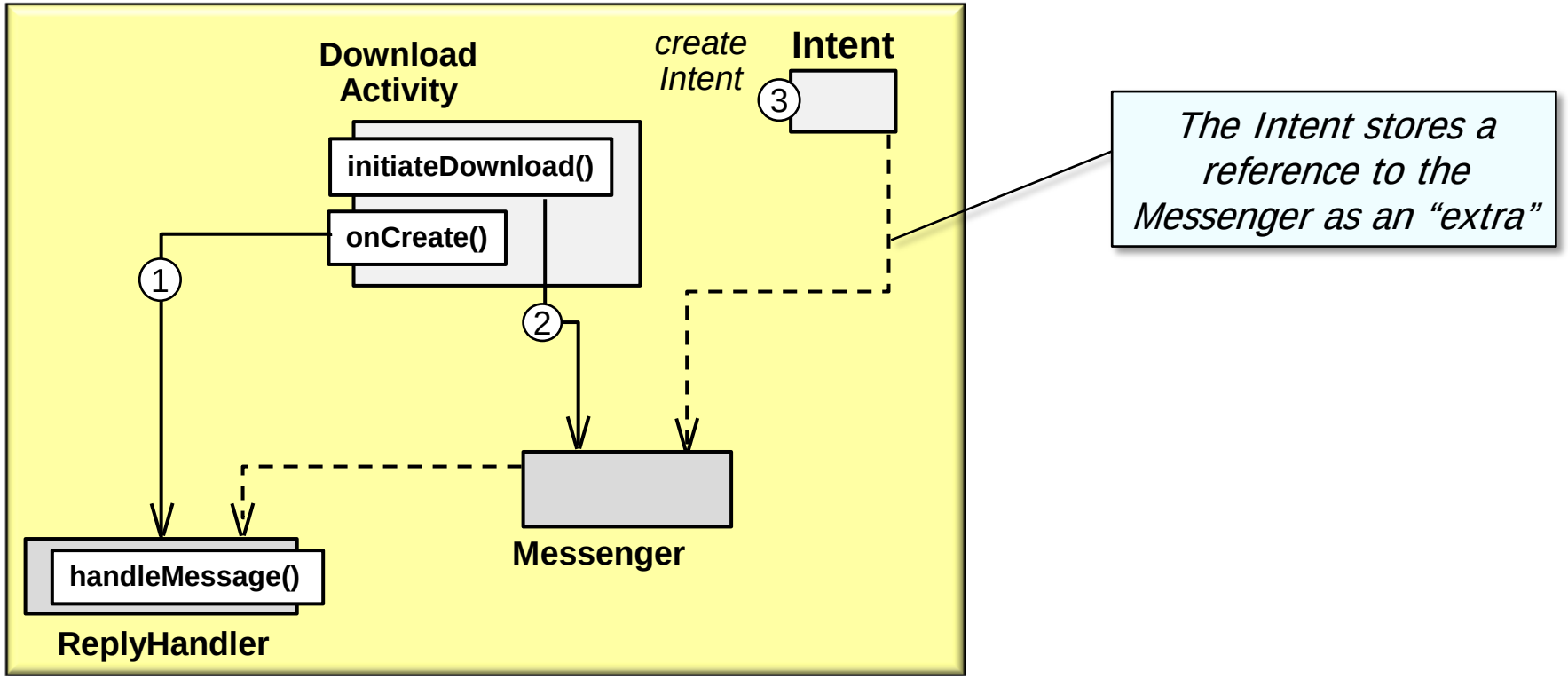
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an "extra" to the Intent used to activate the DownloadService
  - DownloadService uses the Messenger to reply back to the Activity



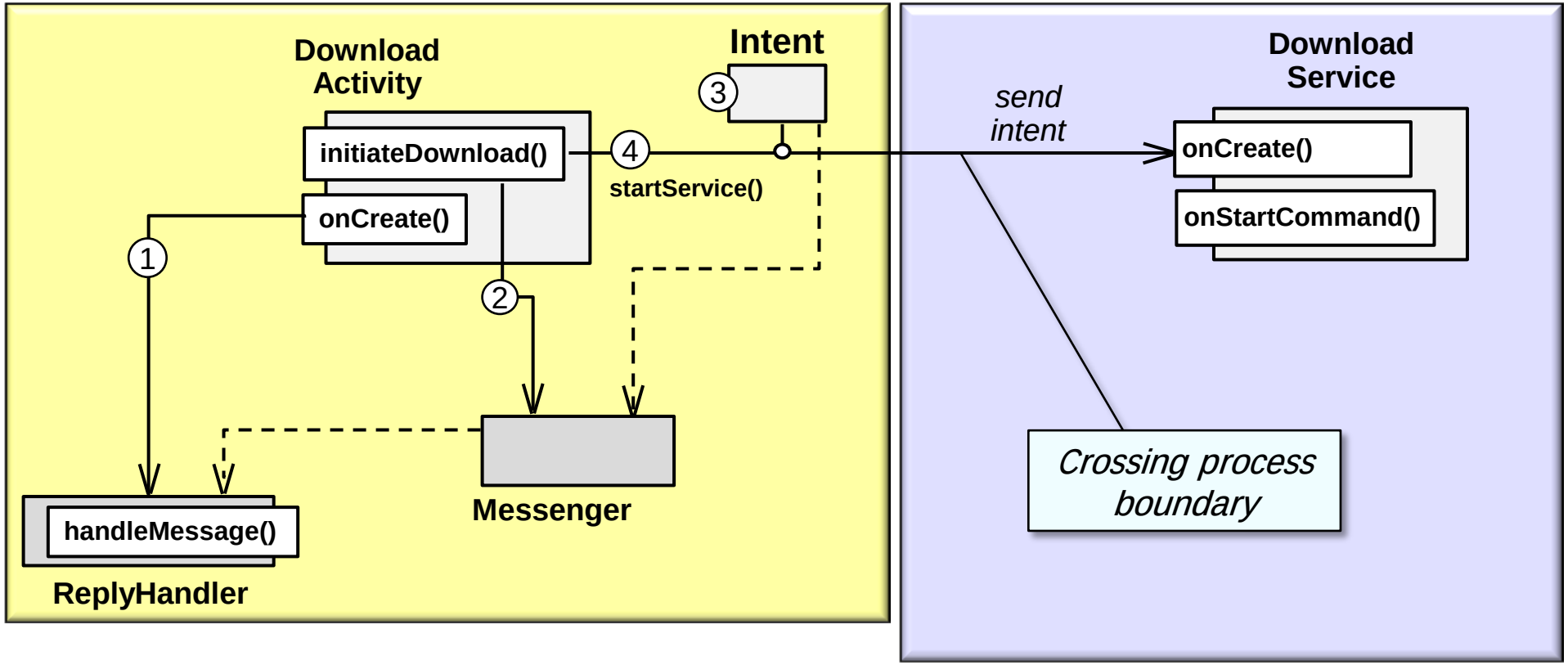
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an "extra" to the Intent used to activate the DownloadService
  - DownloadService uses the Messenger to reply back to the Activity



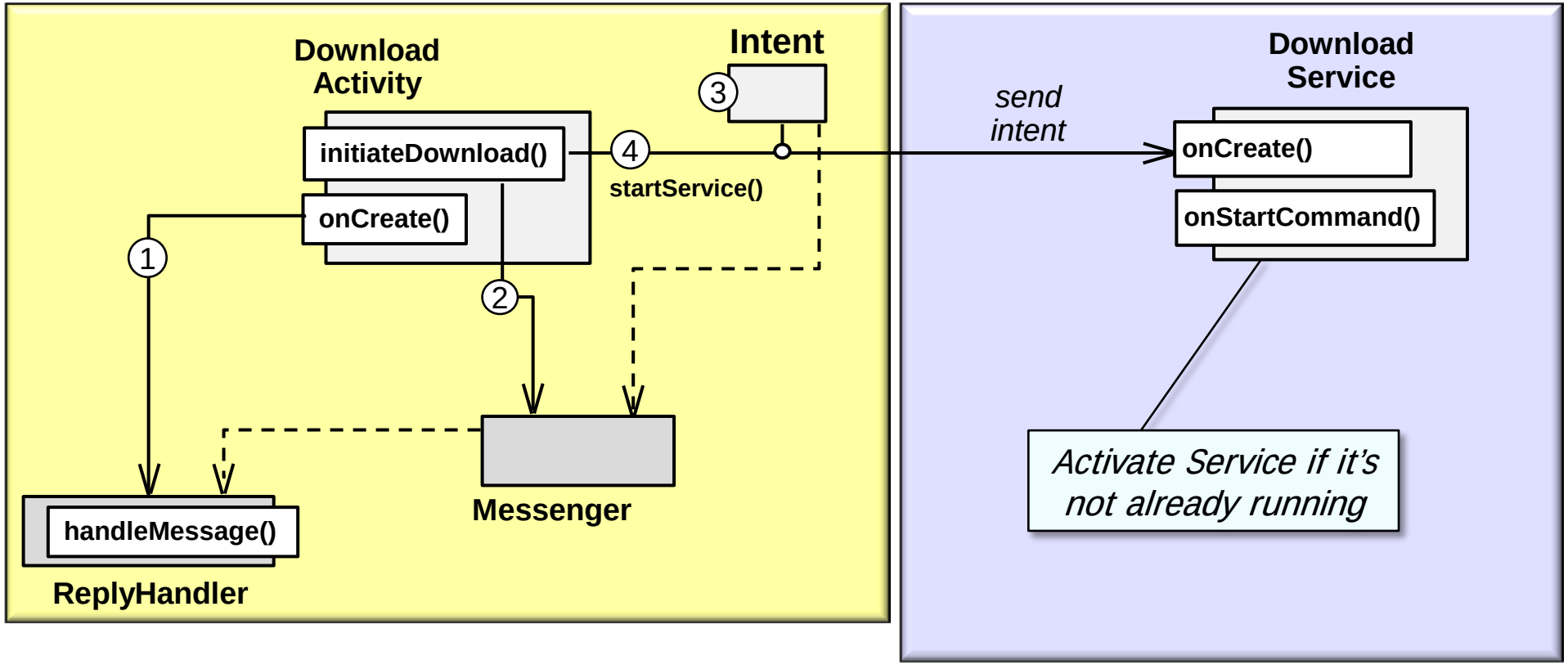
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an "extra" to the Intent used to activate the DownloadService
  - DownloadService uses the Messenger to reply back to the Activity



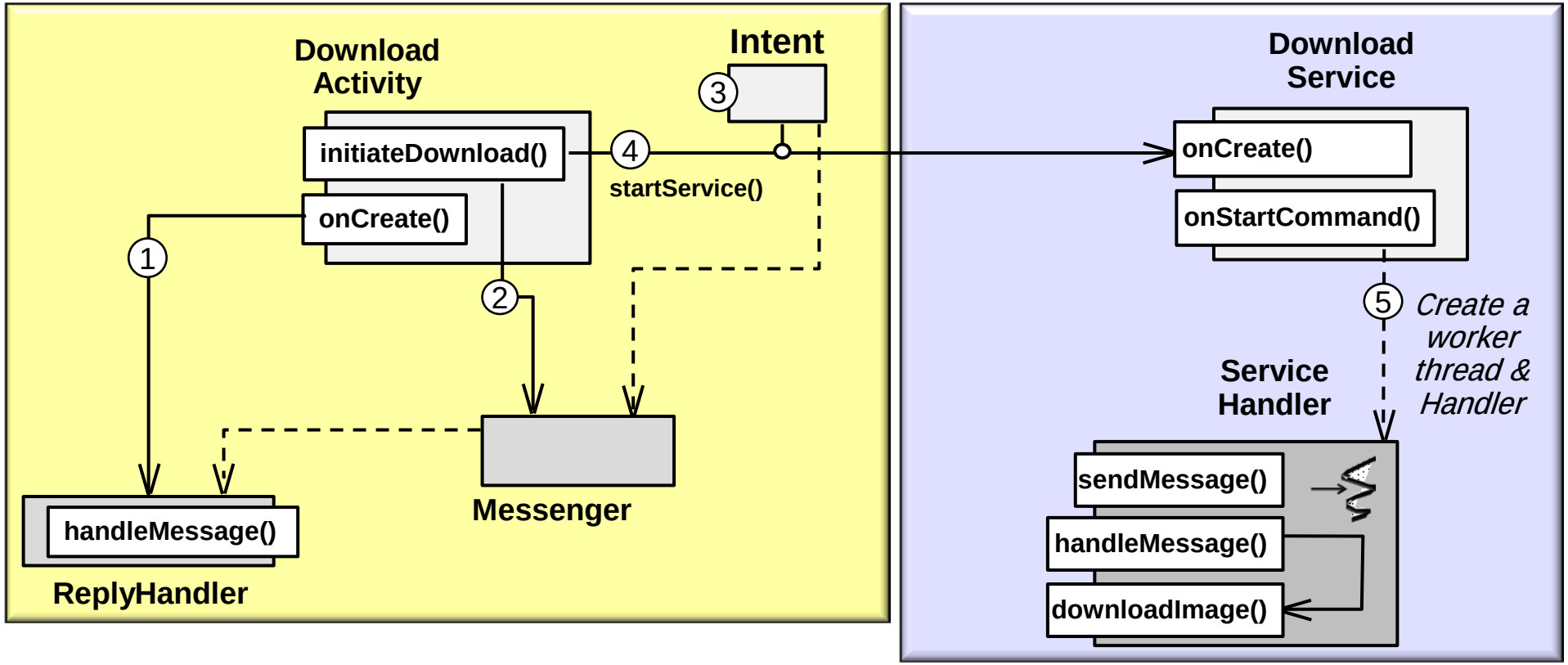
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an "extra" to the Intent used to activate the DownloadService
  - DownloadService uses the Messenger to reply back to the Activity



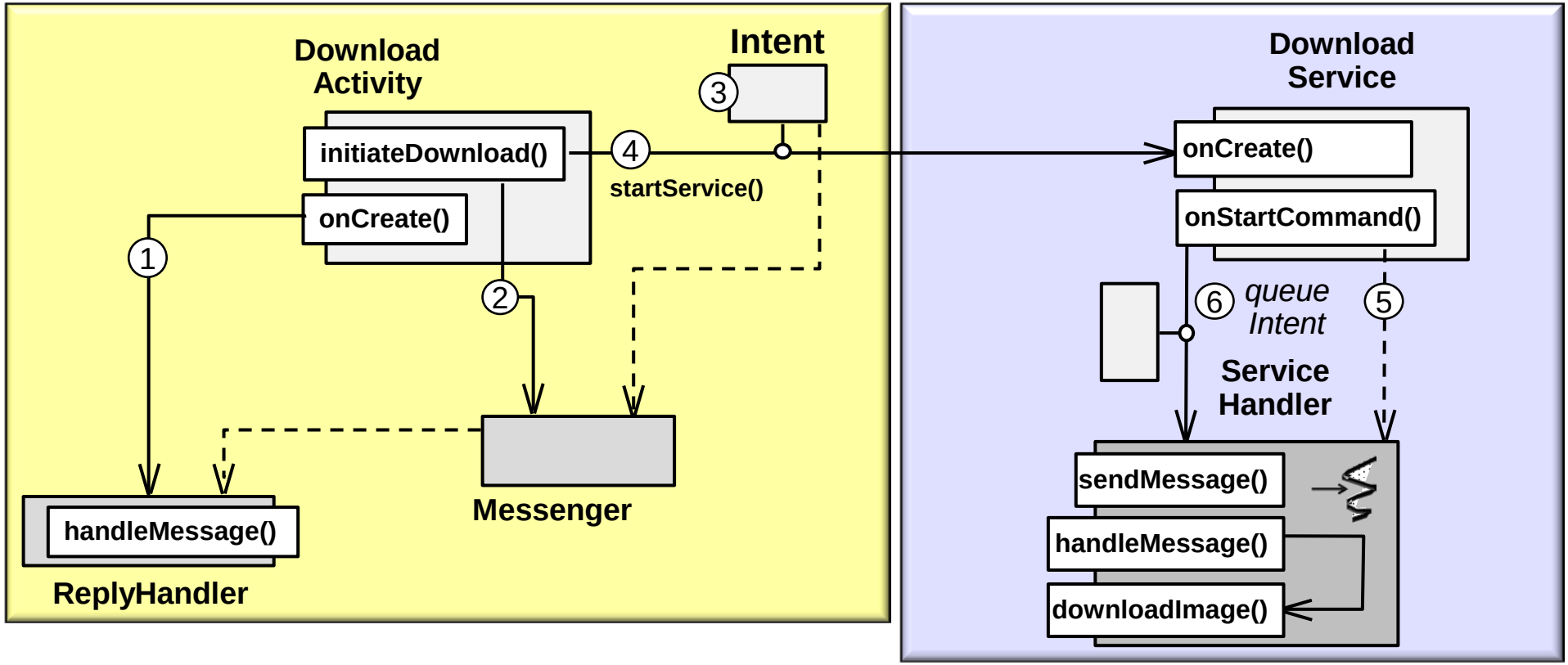
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an "extra" to the Intent used to activate the DownloadService
  - DownloadService uses the Messenger to reply back to the Activity



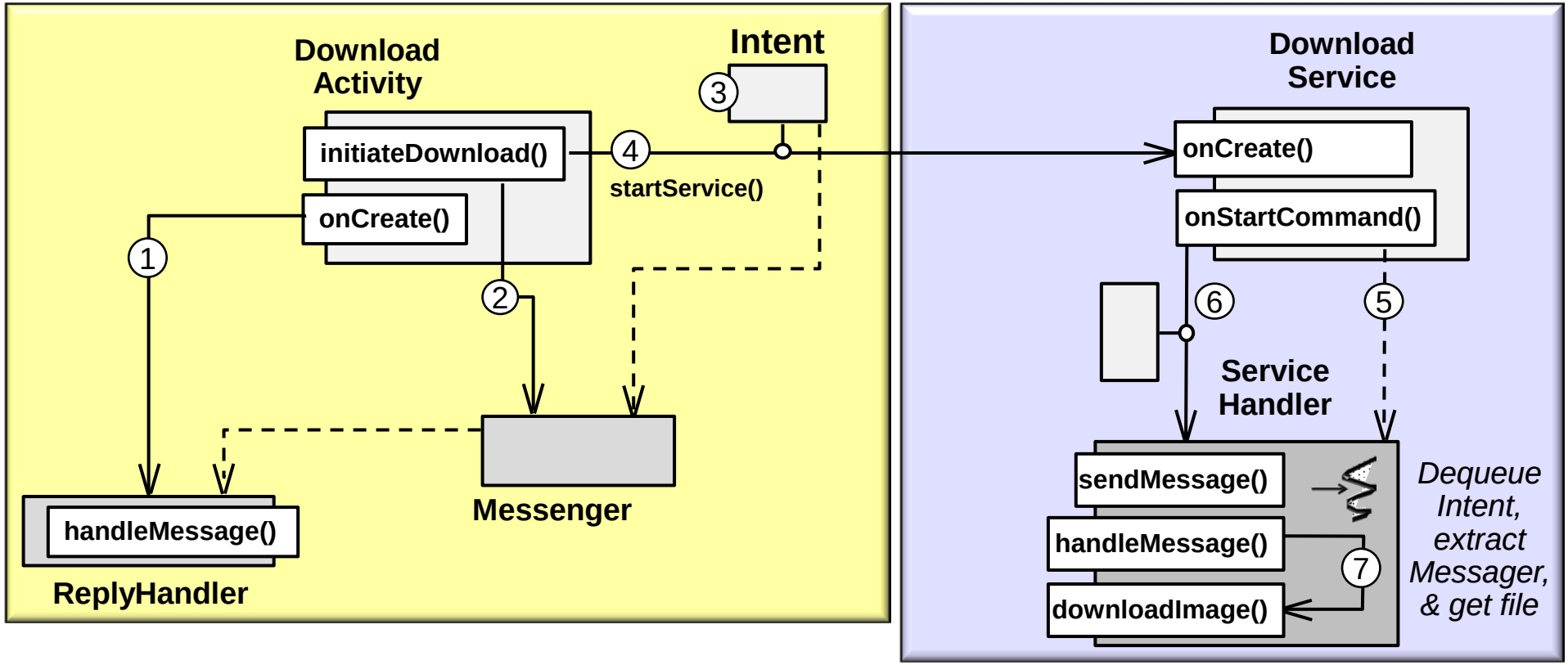
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an "extra" to the Intent used to activate the DownloadService
  - DownloadService uses the Messenger to reply back to the Activity



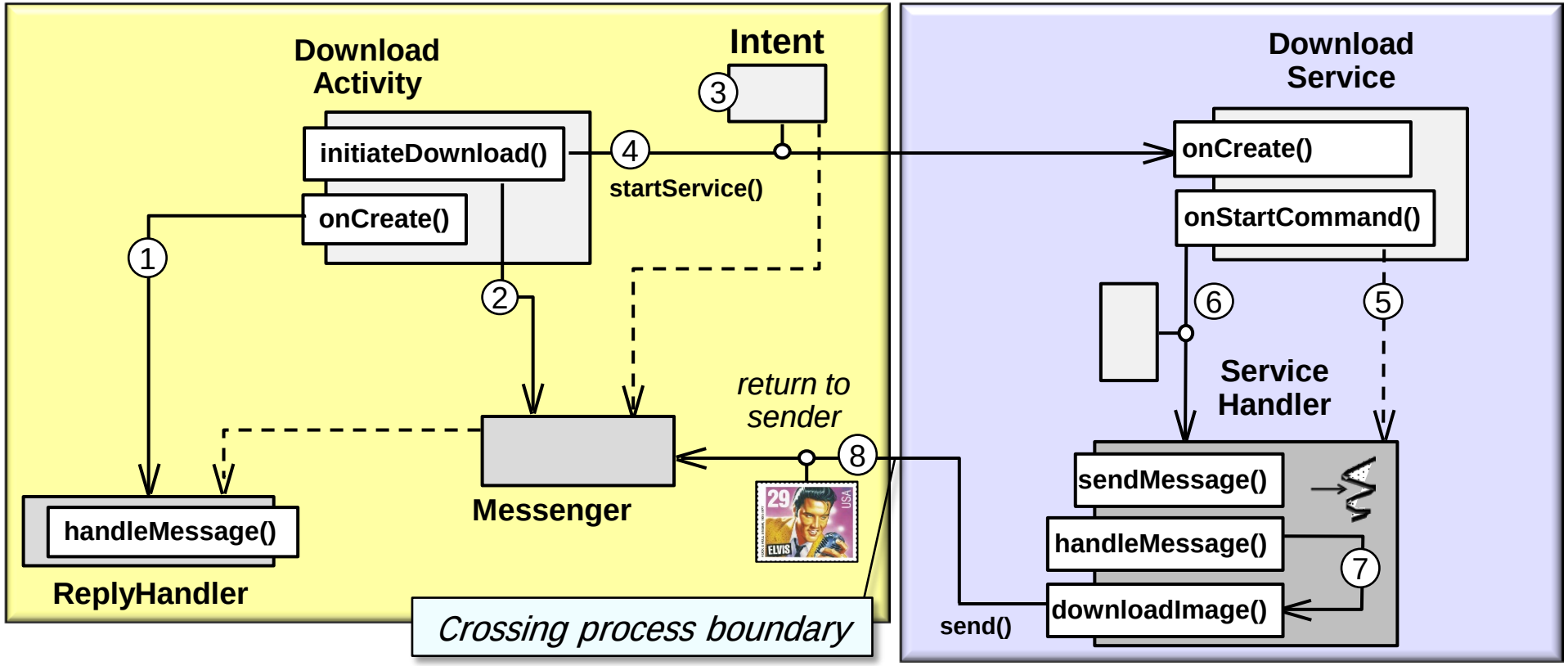
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an "extra" to the Intent used to activate the DownloadService
  - DownloadService uses the Messenger to reply back to the Activity



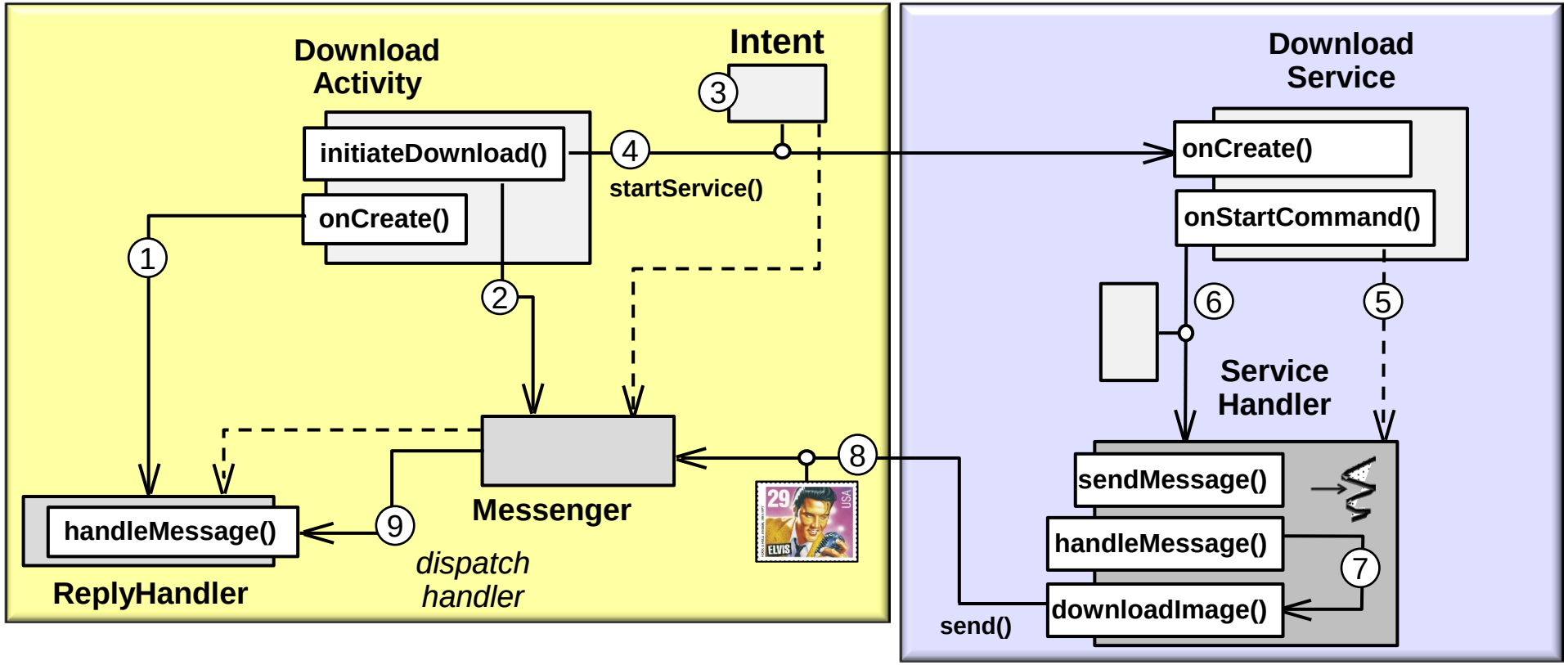
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an "extra" to the Intent used to activate the DownloadService
  - DownloadService uses the Messenger to reply back to the Activity



Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an "extra" to the Intent used to activate the DownloadService
  - DownloadService uses the Messenger to reply back to the Activity



Programming a Messenger in Download Service

- Service replies to Activity via Messenger's send() method

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

Code to downloading image to pathname goes here

```
      Message msg = Message.obtain();
      msg.arg1 = result;
      Bundle bundle = new Bundle();
      bundle.putString("PATHNAME", pathname);
      msg.setData(bundle);
      Messenger messenger = (Messenger)
        intent.getExtras().get("MESSENGER"));
      messenger.send(msg);
    }
  ...
```

Return pathname to the client

Programming a Messenger in Download Service

- Client Activity receives Message via its Handler event looper

```
public class DownloadActivity extends Activity {
  ...
  Handler handler = new Handler() {
    public void handleMessage(Message msg) {
      Bundle data = msg.getData();
      String pathname = data.getString ("PATHNAME");

      if (msg.arg1 != RESULT_OK || path == null) {
        Toast.makeText(DownloadActivity.this,"failed download",
                       Toast.LENGTH_LONG).show();
      }
      displayBitmap(path);
    }
  };
...
```

Get pathname from Download Service

Display the image

Summary

- Messengers provide a flexible framework for communication between processes in Android
- Asynchrony is straightforward, though can be complex for non-trivial usages