CMPS 121
Services

Based on lectures in Fall 2015 of Prof. Luca de Alfaro and in Fall 2017 of Narges Norouzi

Services

- Services are application components that can run long-lived background tasks. Services run even when the user switches to another app.
- A service has two ways of running:
- **Started:** `startService()` has been called. The service can run, until it decides to stop itself.
- **Bound:** An application component can bind a service by calling `bindService()`. A bound service runs only as long as there is something bound to it.

Characteristics of Services

- Started with an Intent
- Can stay running when user switches applications
- Lifecycle—which you must manage
- Other apps can use the service—manage permissions
- Runs in the main thread of its hosting process

Foreground Services

- Runs in the background but requires that the user is actively aware it exists—e.g. music player using music service
- Higher priority than background services since user will notice its absence—unlikely to be killed by the system
- Must provide a notification which the user cannot dismiss while the service is running

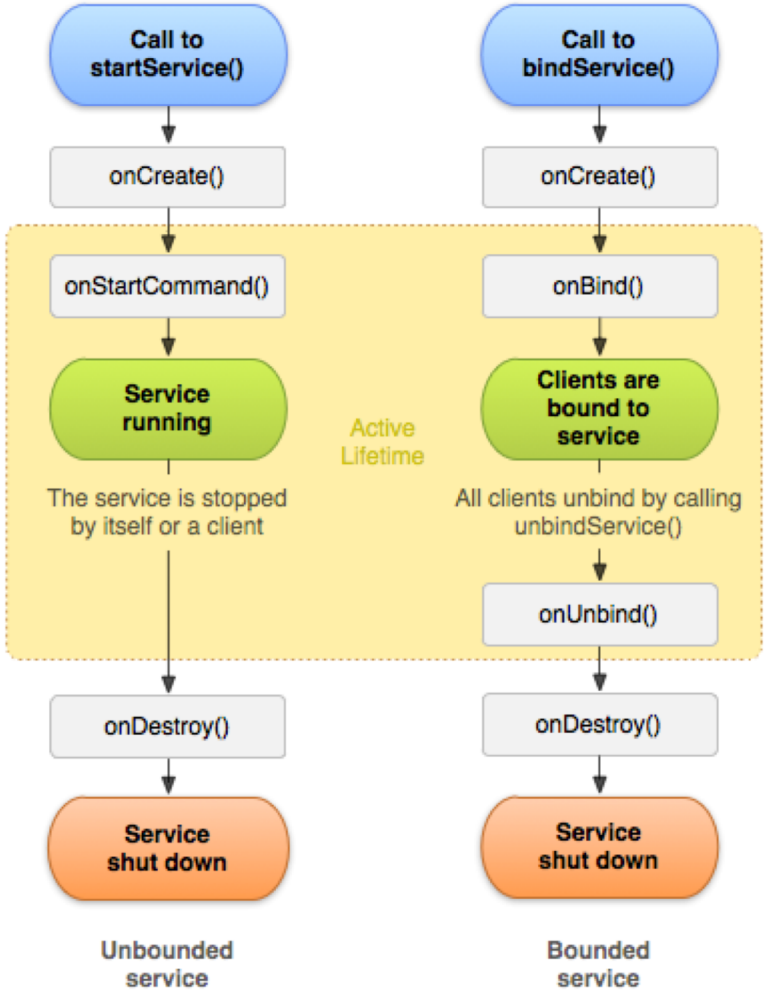
Declaring a service in the manifest

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

Creating a service

- `<service android:name=".ExampleService" />`
- Manage permissions
- Subclass IntentService or Service class
- Implement lifecycle methods
- Start service from activity
- Make sure service is stoppable

Service lifecycle



Creating a Service – Cont.

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 to be bound).
- `onCreate()` -- called before above methods.
- `onDestroy()` -- called when about to be shut down.

Stopping a Service

- A **started service** must manage its own lifecycle
- If not stopped, will keep running and consuming resources
- The service must stop itself by calling stopSelf()
- Another component can stop it by calling stopService()
- **Bound service** is destroyed when all clients unbound
- **IntentService** is destroyed after onHandleIntent() returns

Creating a Started Service

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.

The 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()`.

Example of IntentService

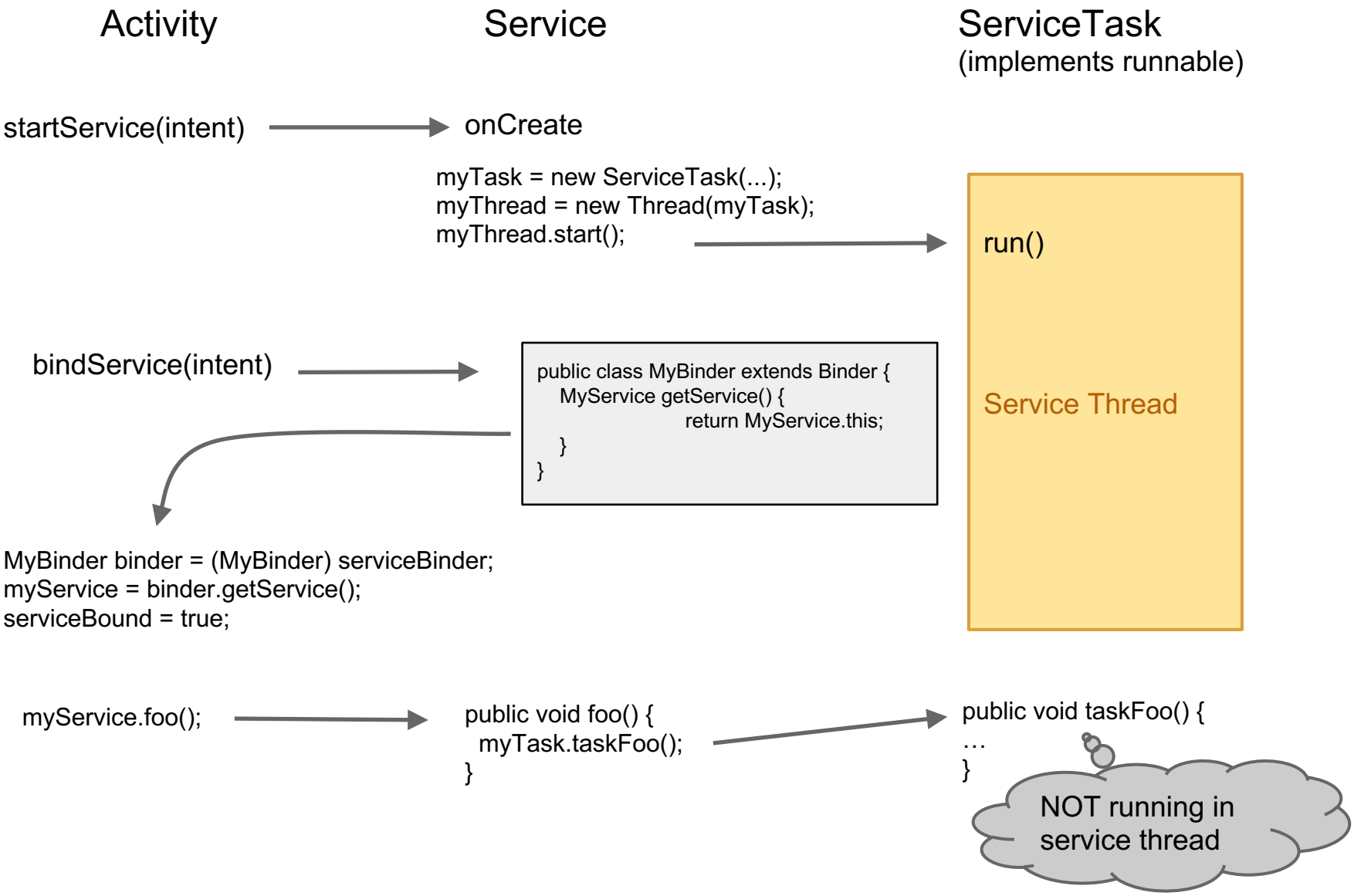
```
public class HelloIntentService extends IntentService {
// A constructor is required, and must call the super IntentService(String)
public HelloIntentService() {  super("HelloIntentService");  }

 /* 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. */
  @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) {
              }
   }}}}
```

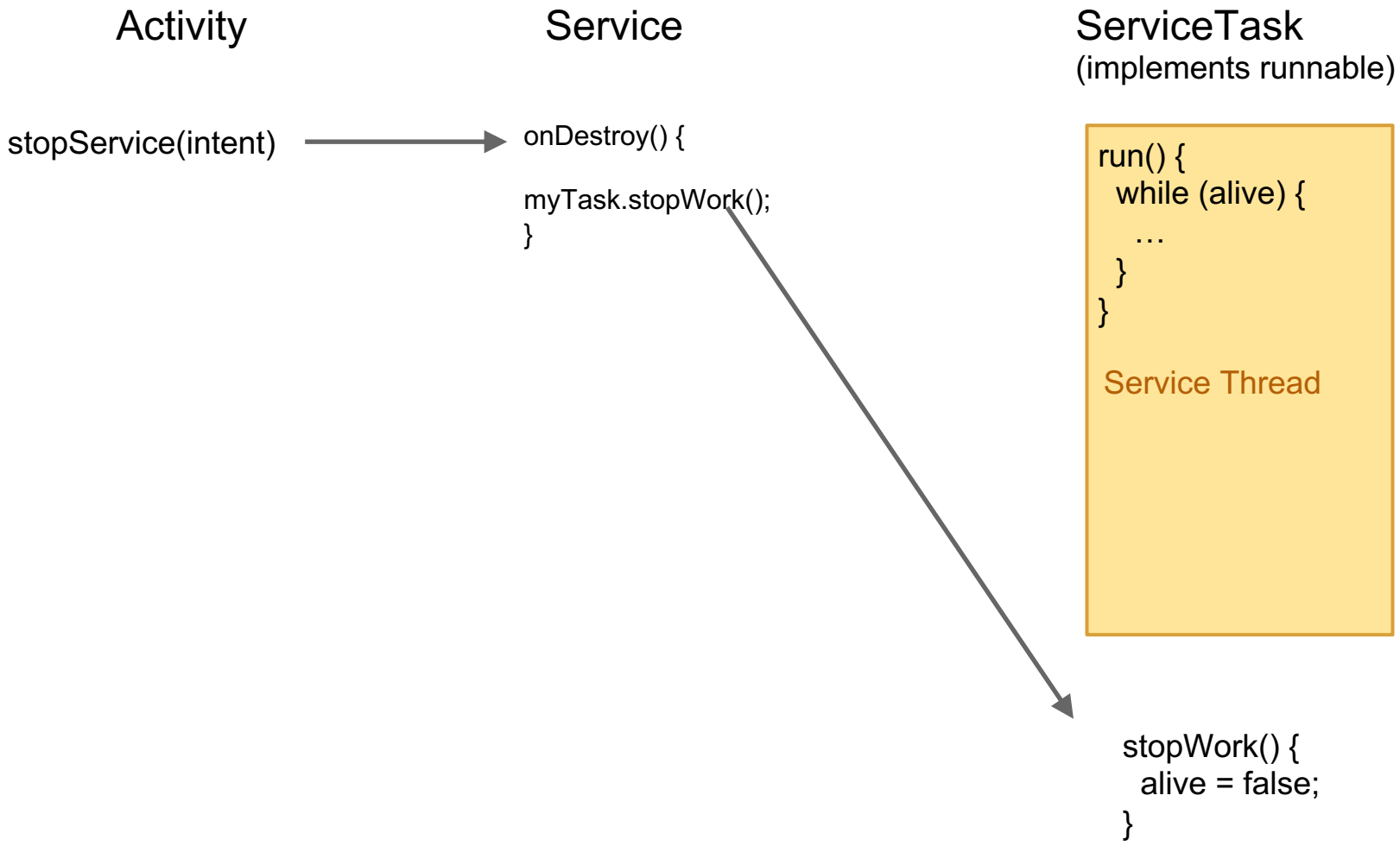
IntentService Limitation

- Cannot interact with the UI
- Can only run one request at a time
- Cannot be interrupted

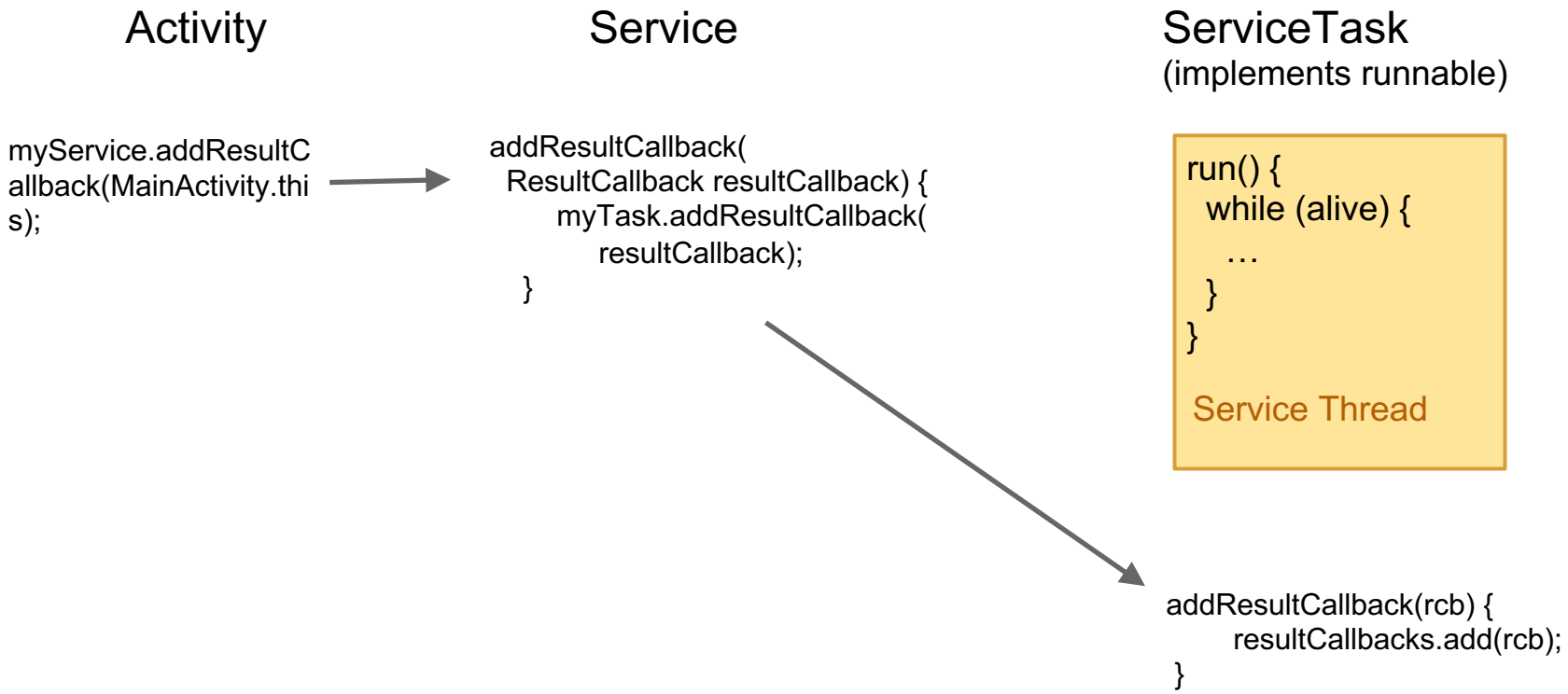
Activity to service connection



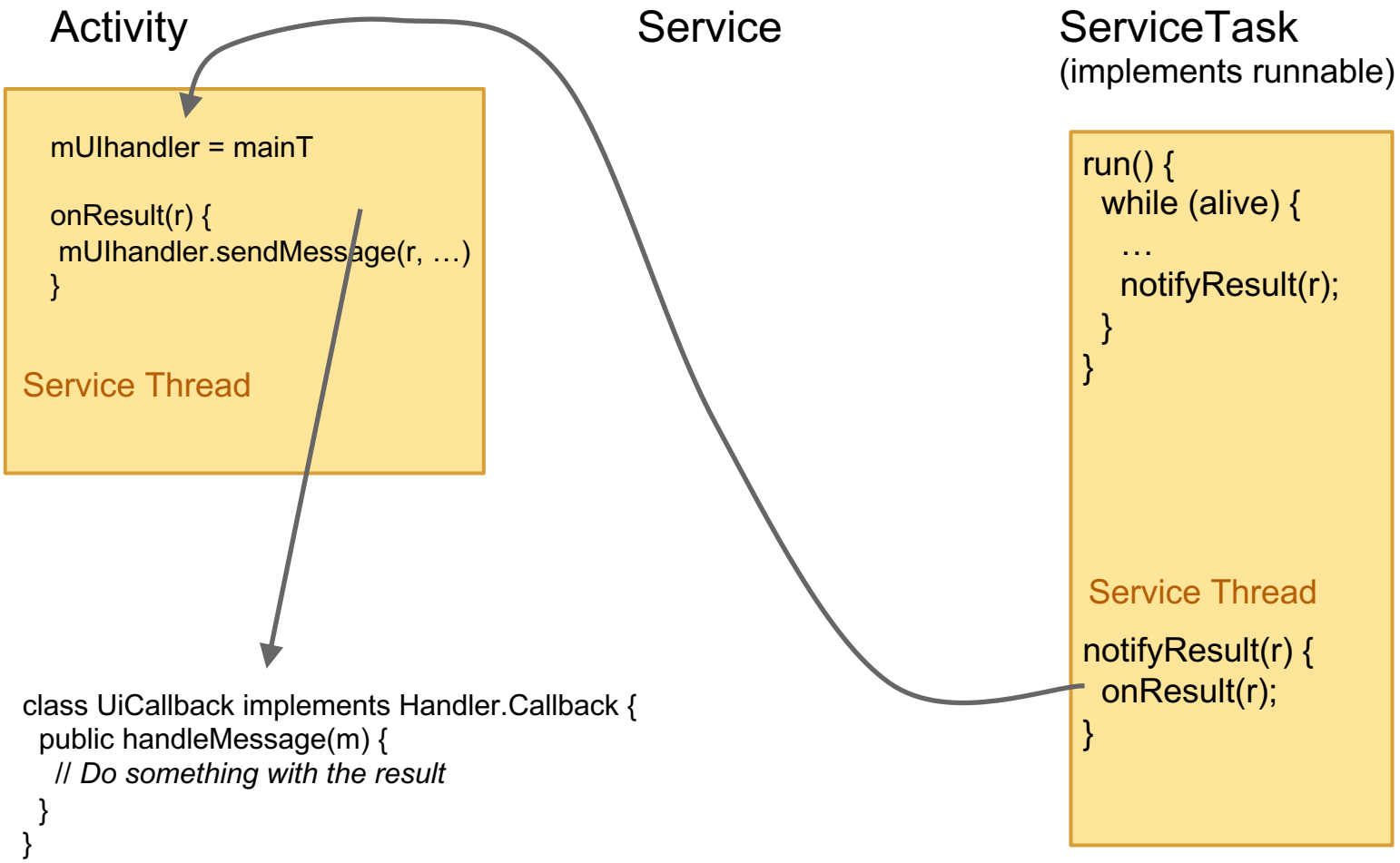
Stopping the service



Register with the service



Send from service to activity



Message passing

```
class LooperThread extends Thread {
      public Handler mHandler;

      public void run() {
          Looper.prepare();
          mHandler = new Handler() {
              public void handleMessage(Message msg) {
                  // process incoming messages here
              }
          };
          Looper.loop();
      }
  }
```

Message passing - Handlers

- **Sending to a handler:**
  - sendMessage(Message)
  - post(Runnable)
  - postDelayed(Runnable, int delay)
  - hasMessages(...)
  - …

- **Getting what was sent:**
  - handleMessage(Message)
  - Runnables are simply run.