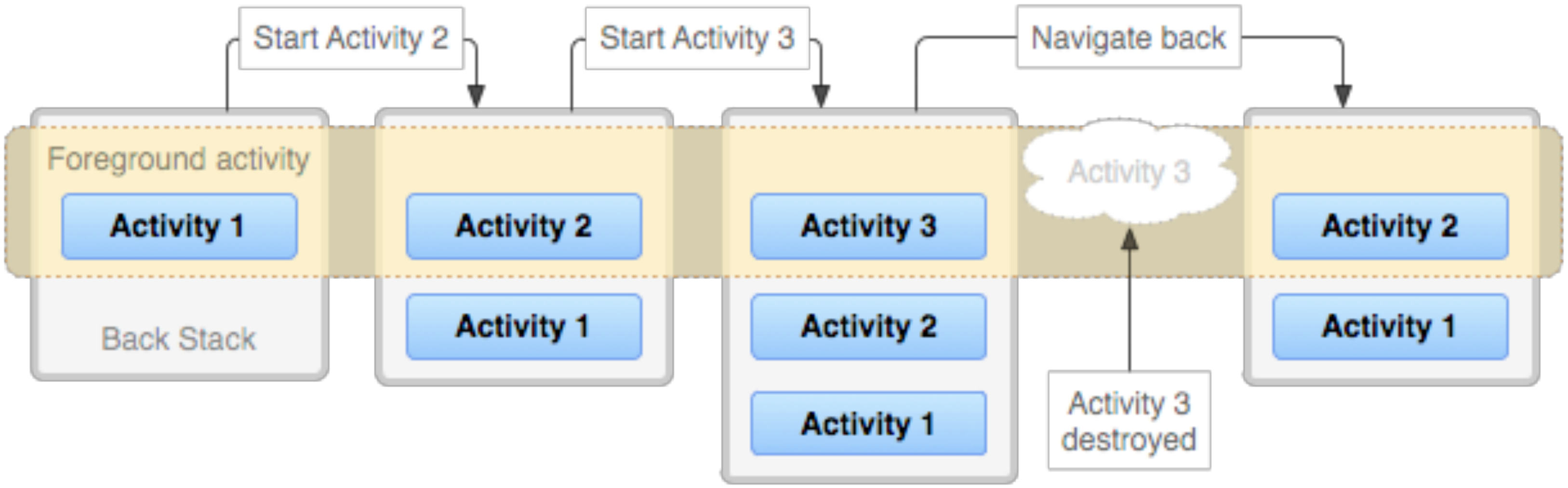
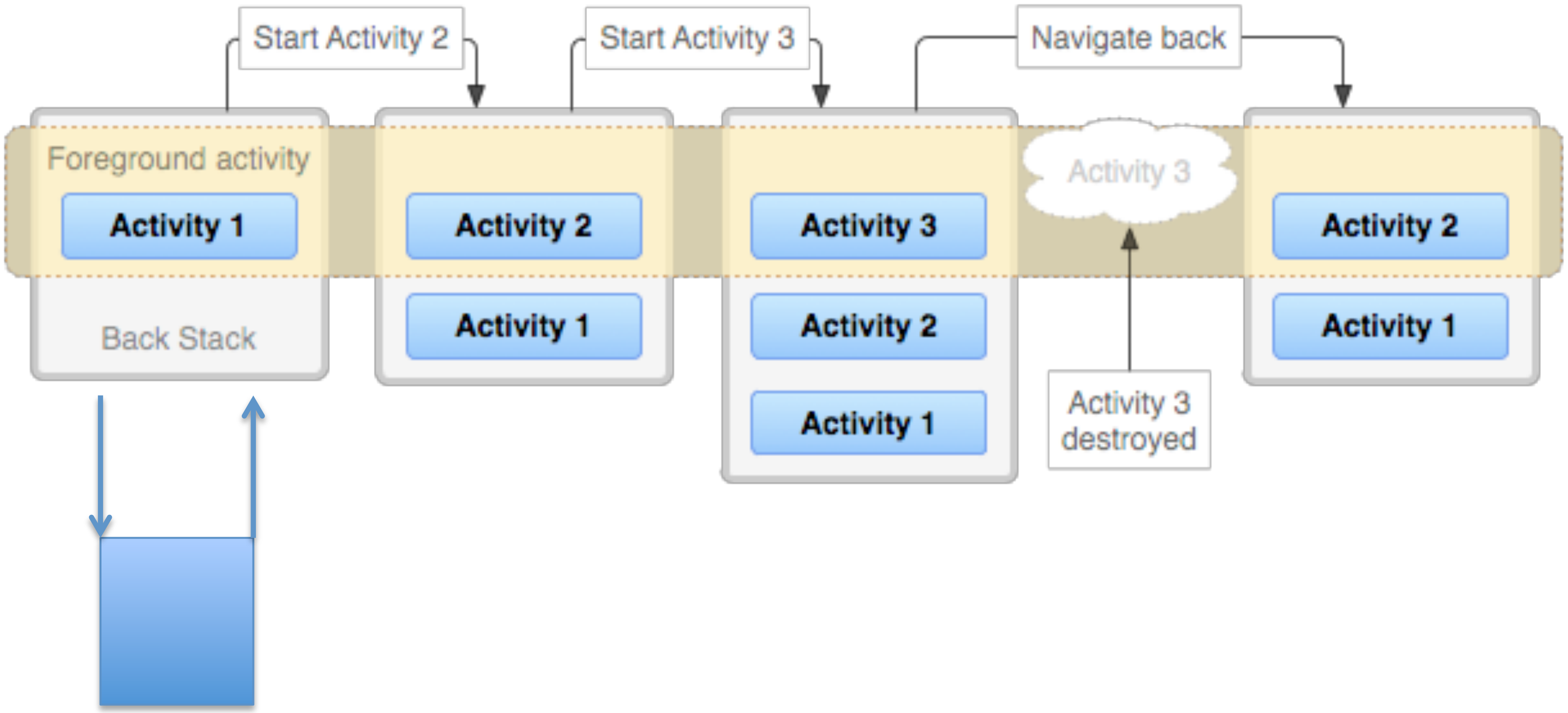
G54MDP
Mobile Device Programming

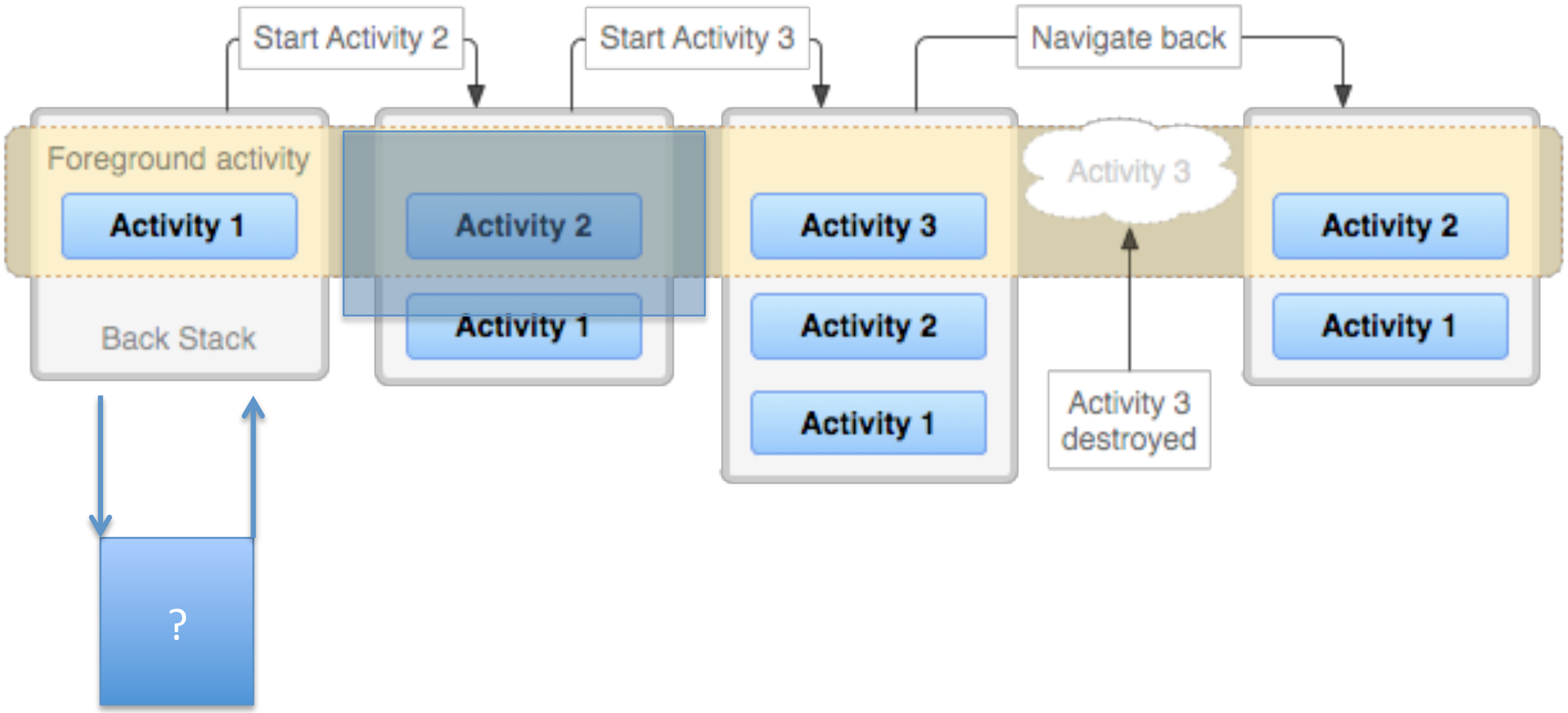
Lecture 7 –Services

Threads

- Android applications use a **single thread model**
  - A single thread of execution called *main*
- Handles and dispatches user interface events
  - Drawing the interface
  - Responding to interactions
    - E.g. onClick()
- Handles activity lifecycle events
  - onCreate(), onDestroy…
- For **all** components in an application


Start Activity 2
Start Activity 3
Navigate back
Foreground activity
Activity 1
Back Stack
Activity 2
Activity 1
Activity 3
Activity 2
Activity 1
Activity 3
Activity 3 destroyed
Activity 2
Activity 1


Start Activity 2
Start Activity 3
Navigate back
Foreground activity
Activity 1
Back Stack
Activity 2
Activity 1
Activity 3
Activity 2
Activity 1
Activity 3
Activity 3 destroyed
Activity 2
Activity 1


Start Activity 2
Start Activity 3
Navigate back
Foreground activity
Activity 1
Back Stack
Activity 2
Activity 1
Activity 3
Activity 2
Activity 1
Activity 3
Activity 3 destroyed
Activity 2
Activity 1
?

Activities

- Activities provide Android apps with a UI
  - But they are only active when they are interacting with the user
  - Otherwise, inactive and possibly unloaded from memory
    - Not suitable for background processes
    - Even if we separate some code out into a separate thread for a responsive UI, if the UI goes away we are in trouble

Services

- An Application **Component** that
  - Has no UI
  - Represents a desire to perform a long-running operation
- Activities are loaded/unloaded as users moves around app
  - Services remain for as long as they are needed
- Expose functionality for other apps
  - One service may be used by many applications
  - Avoid duplication of resources

What Services are not

- It’s helpful to think about what a Service is not:
  - Not a separate process
    - Runs in the same process as the application in which it is declared (by default)
  - Not a thread
    - One thread per Application
      - Handles events for all components
    - If you need to do things in the background, start your own thread of execution
      - An IntentService does this automatically
- Services are logically quite simple
  - A way of telling the system about part of your app that is expected to run for a long time
    - i.e. longer than a few seconds
  - But slightly more complicated to implement

Uses of Services

- MP3 Playback
  - Want to play audio while the user is doing other things
- Network Access
  - Long download
  - Sending an email
  - Polling an email server for new mail
- Anything that you don’t want to interrupt the user experience for
  - Remember, the user interacts with one application are once on a phone

Uses of Services

- The email task
  - Checks for new mail occasionally
  - Collects new mail and stores it somewhere
  - Notifies user that there is new mail
  - User switches to the Inbox Activity
  - Inbox Activity then fetches new mails and displays them
- MP3 playback task
  - Play music while the user does something else
  - Have activities that let you change the playing track or the volume

Creating a Service

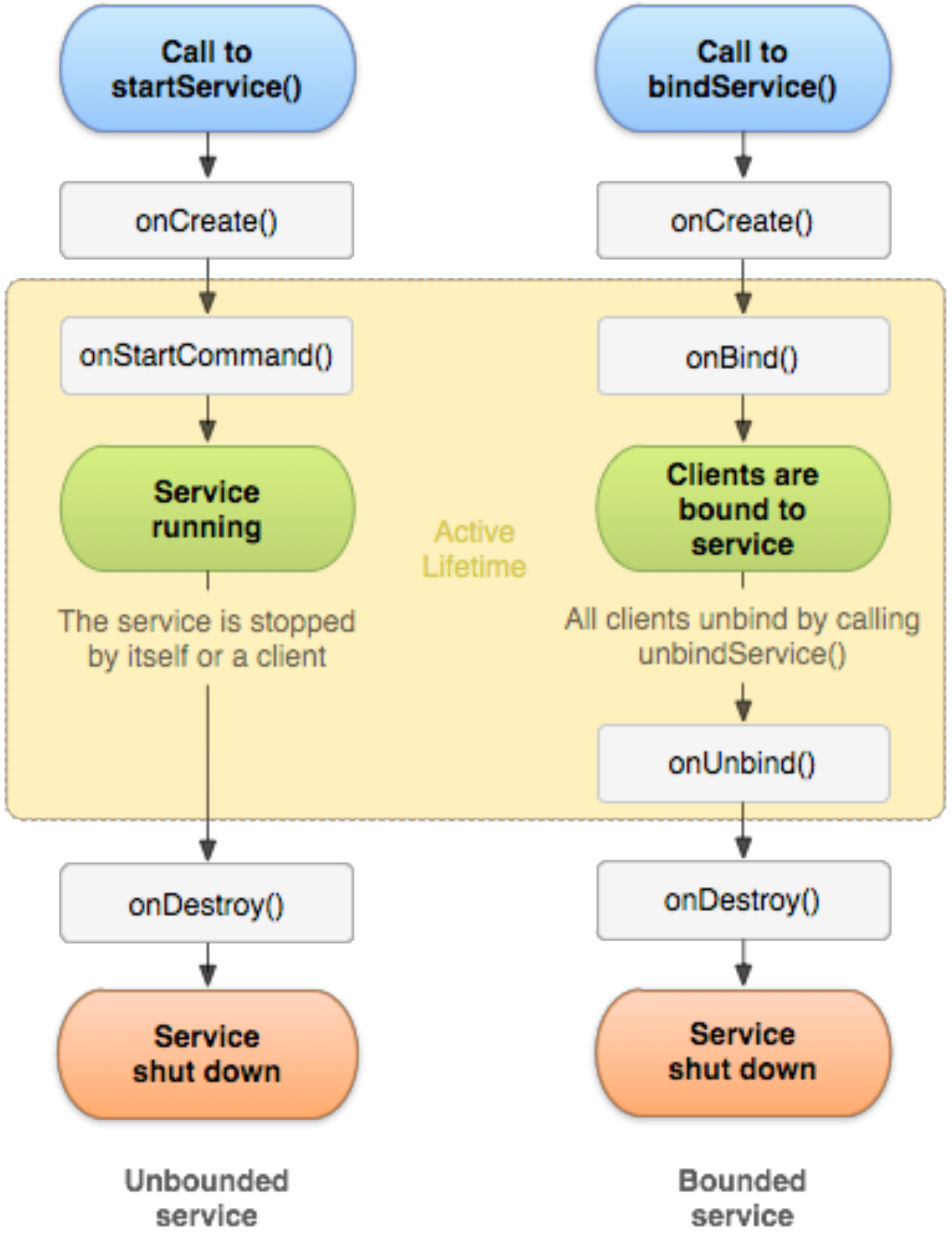
- Services are designed to support communication with
  - Local Activities (in the same process)
  - Remote Activities (in a different process)
    - IPC
- Are Components, similar to an Activity
  - Register the service in the manifest
  - Create a subclass of android.app.Service

Service Lifecycle

- Two ways of spawning a service
  - Started
    - Send an Intent with startService()
    - Will run in the background indefinitely / kills itself
      - C.f email checking
      - Does not “return” results
    - Or can be explicitly stopped with stopService()
  - Bound
    - Bind to a service using bindService()
    - Will run while any Activities are bound to it
      - Actively using it
    - Provides an interface for Activities to communicate with the Service
- In both cases, if the service is not running it will be created

Service Lifecycle

- By nature, services are singleton objects
  - "There can be only one"
- The Service sub-class object is instantiated if necessary
  - onCreate() is called
  - either onStartCommand or onBind will be called depending on how the service has been called
- onCreate / onStart / onBind are called in the context of the main UI thread
  - Must spawn a worker thread to do any significant work


Call to startService()
onCreate()
onStartCommand()
Service running
The service is stopped by itself or a client
onDestroy()
Service shut down
Unbounded service
Call to bindService()
onCreate()
onBind()
Clients are bound to service
All clients unbind by calling unbindService()
onUnbind()
onDestroy()
Service shut down
Bounded service
Active Lifetime

Implementing Services

- IntentService
  - A simple, unbound service
    - Assumes we don't have multiple requests that need to be handled concurrently
    - Creates a queue of work to be done
  - Handles one intent at a time to onHandleIntent()
  - Stops the service after all start requests have been handled
  - I.e. sending emails
    - "fire and forget"
- Generic started service
  - Runs persistently
    - i.e. checking for emails
    - (Or stops itself when all work is done)
  - Receives messages asking for more work to be done

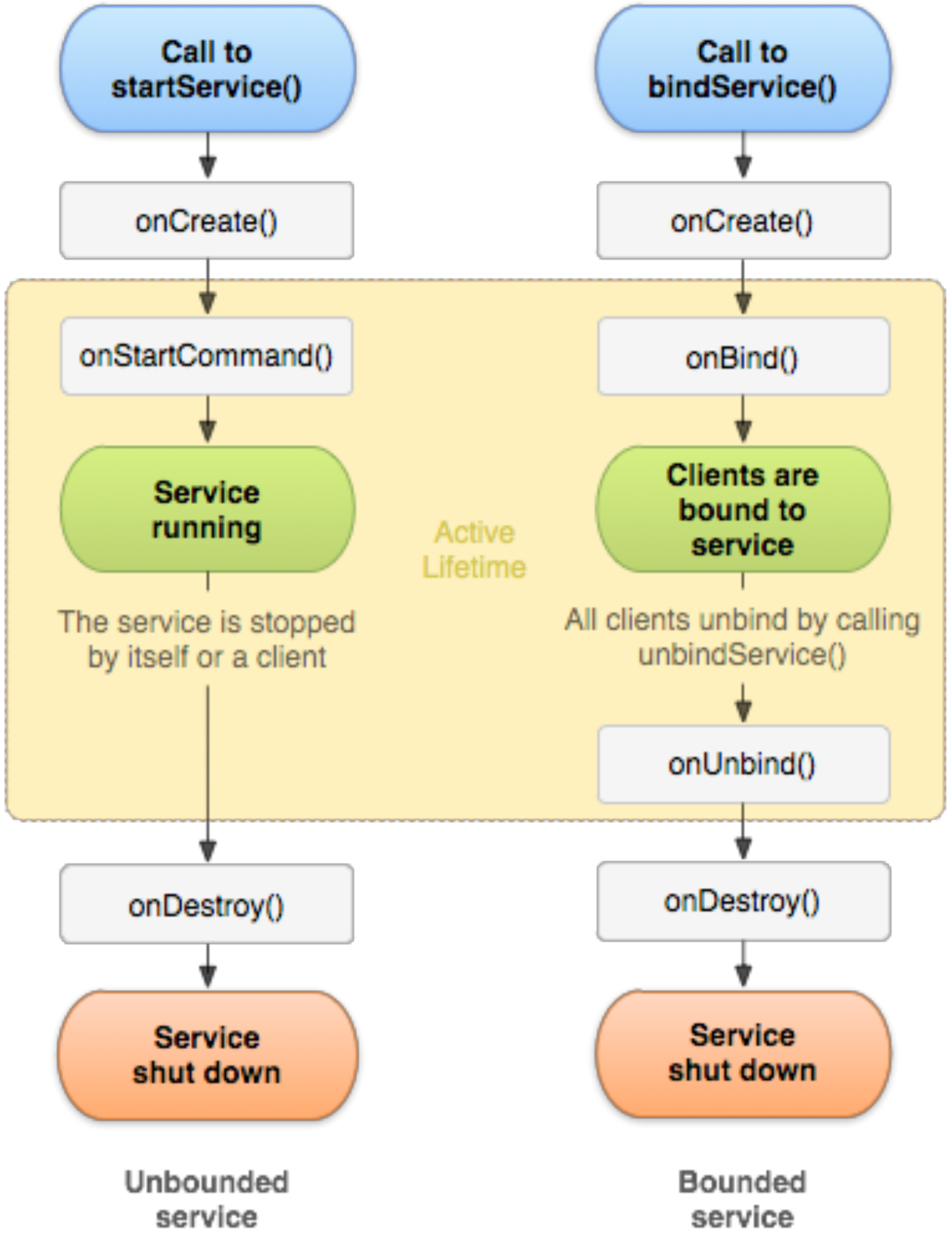
Notifications

- But how do we notify the user that the Service is operating / has done something?
  - The original Activity may no longer exist
- Status bar notification
  - Maintained by the service
  - Can specify an Intent / Activity to launch if the user clicks on it
    - Return to the Activity that spawned the Service

Let’s have a look…

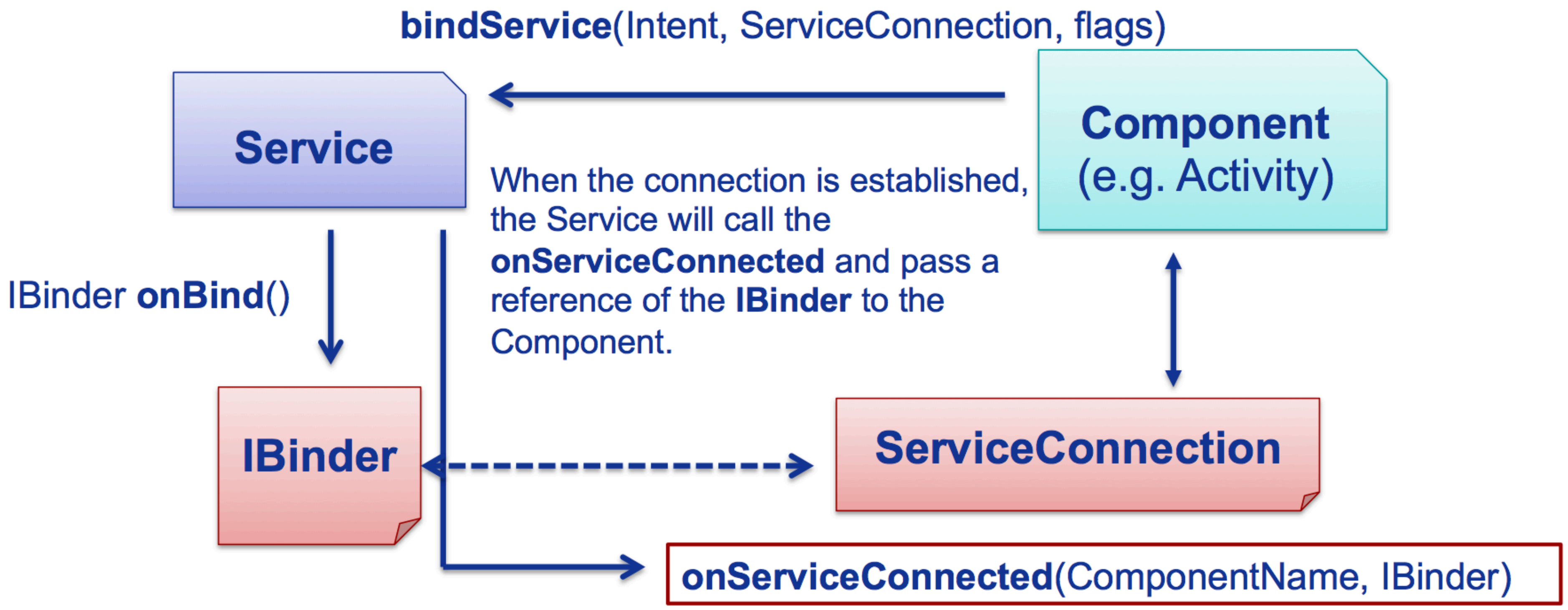
Terminating Services

- A Service runs in the background indefinitely
  - Even if the component that started it is destroyed
- Termination of a service
  - Self-termination (calling stopSelf())
  - stopService() via an Intent
  - System termination
    - i.e. memory shortage
- Avoiding termination
  - Foregrounding a Service
    - This is something the user should really know about
    - Active in the Status Bar / shows a Notification
    - Is treated as important as a foregrounded Activity


Call to startService()
onCreate()
onStartCommand()
Service running
The service is stopped by itself or a client
onDestroy()
Service shut down
Unbounded service
Call to bindService()
onCreate()
onBind()
Clients are bound to service
All clients unbind by calling unbindService()
onUnbind()
onDestroy()
Service shut down
Bounded service
Active Lifetime

Bound Services

- Provide an interface for clients (Activities) to interact with a Service
  - Provide a programmatic interface for clients
  - Fast *and* stable?
- **Extending** the Binder class
  - Return an interface via the onBind method
  - Only for a Service used by the same application
    - Local Services only
    - i.e. the same process
- **Using** the Android Interface Definition Language (AIDL)
  - Provide a standard interface to access the Service from different applications


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.

Let’s have a look…

References

- http://developer.android.com/guide/components/processes-and-threads.html

- http://developer.android.com/guide/components/services.html