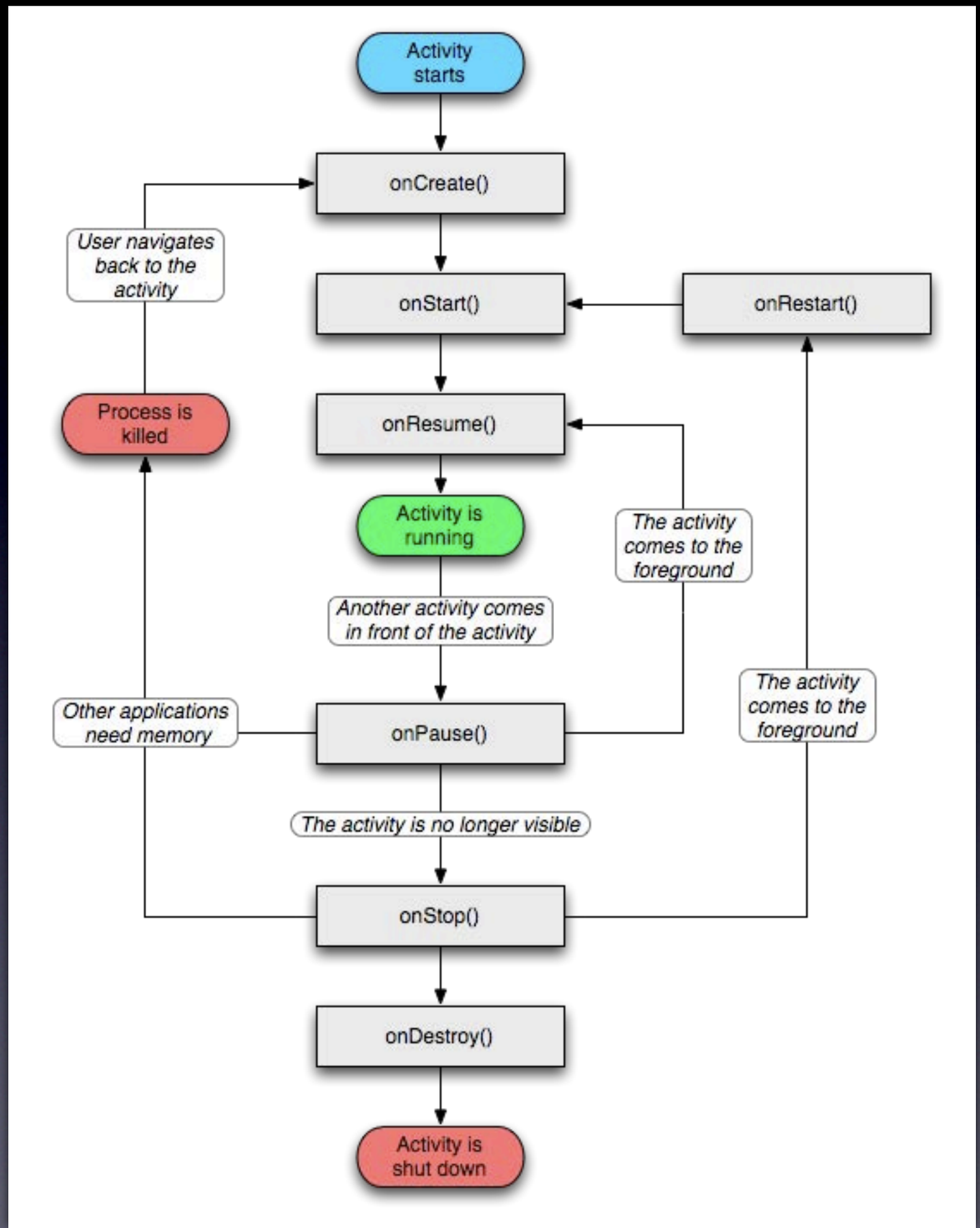
Services

Steven R. Bagley

Activities

- `Activitys` 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…



Can do this by logging when the various methods are called from the two Activities are called then seeing the overlap in the logfile

Today

- Overview of Services in Android
- What they are?
- What would you use them for?
- And how to create them…

What are 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
  - Exposes functionality for other apps

What a service isn't?

- It's helpful to think about what a Service isn't too
  - Not a separate process
  - Not a thread
  - If you need to do things in the background, start your own or use an `IntentService`

Services

- Very simple
- Just a way of telling the system about part of your app
- That will run for a while…

Use of Services

- MP3 Playback
- Network Access
  - Long download
  - Polling an email server for new mail
  - Anything that you don't want to interrupt the UX for

UX = User Experience

Services and Activities

- Activities can communicate with a service
- Or access the data collected by a service
- Think about the email service
  - Checks for new mail
  - Collects new mail and stores it somewhere
  - Notifies user that there is new mail

Services and Activities

- User switches to the Inbox Activity
- Inbox Activity then fetches new mails and displays them
- Or MP3 playback
- Have activities that let you change the playing track or the volume

Creating a Service

- Mechanically similar to an `Activity`
- Register the service in the manifest
- Create a subclass of `android.app.Service`
- In practice, can get tricky…

Services

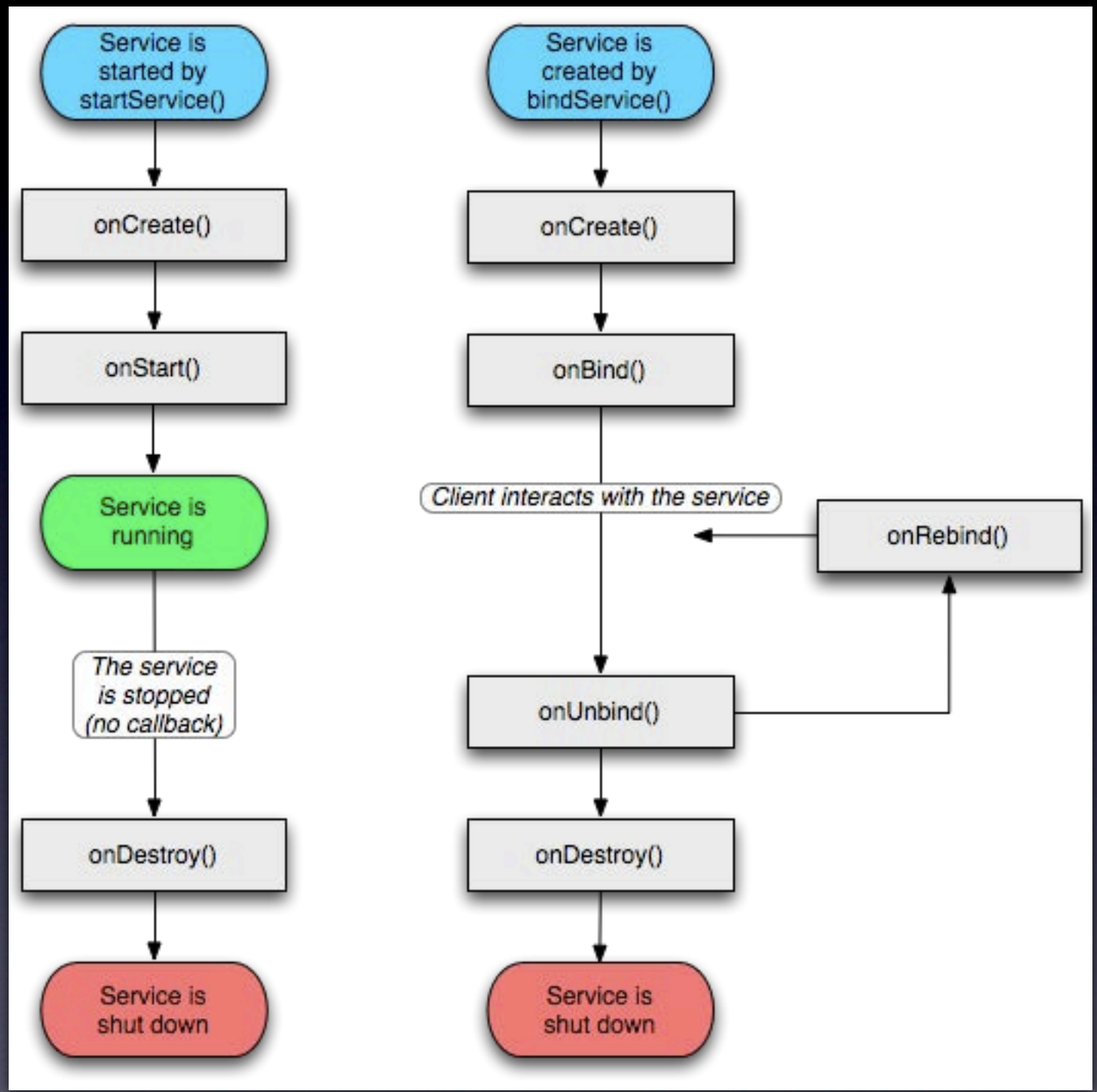
- Services are designed to support communication with
  - Local Activities (in the same process)
  - Remote Activities
- Won't consider remote access here but it has affected the design of Services

Service Lifecycle

- Two ways of starting a service
  - Either send an `Intent` with `Context.startService()`
  - Or, bind to a service using `Context.bindService()`
- In both cases, if the service is not running it will be created

Service Startup

- By nature, services are singleton objects
- The `Service` sub-class object is created if necessary
- Then `onCreate()` is called
  - Need to to call the superclass method
- Then either `onStartCommand` or `onBind` will be called


Service is started by startService()
onCreate()
onStart()
Service is running
The service is stopped (no callback)
onDestroy()
Service is shut down
Service is created by bindService()
onCreate()
onBind()
Client interacts with the service
onRebind()
onUnbind()
onDestroy()
Service is shut down

Service Startup

- `OnStartCommand` is called every time service is started (even if it already exists)
- Could use this to start some process off
- Returns a value to express the service's current started state
- To do with whether `Intent`s get redelivered or not

Go and implement something simplish

See http://developer.android.com/reference/android/app/Service.html#START_CONTINUATION_MASK

Binding

- Unfortunately, due to RPC support not quite that simple
- Need to implement `onBind()` — this returns a reference to an object that implements `IBinder`
- Easiest to just subclass `Binder` and return an object of that subclass

Communicating with the Service

- Can do it via sending various `Intent`s using `onStartService`
- Quite often enough — e.g. for our email example
- Send an Intent that we want it to start checking mail
- Service then updates data store/sends notifications…

Getting a reference

- Or get a reference to the `Service` object
- And call methods on it
- This is why we create the `Binder`

- Use the `getService()` method to obtain a reference
- How do we get the `Binder`?

Binding to a Service

- Need a new `ServiceConnection` object
- Provide implementation of `onServiceConnected()` and `onServiceDisconnected()`
- These get passed a reference to the `IBinder` which we can cast to `DemoBinder`
- Store the service reference in a local variable

Binding to a Service

- Use `bindService` to connect to our `ServiceConnection` to the service
- When we've finished use `unbindService` with the same connection to unbind it
- Be aware these calls are asynchronous — think about where you place them
- Or…

Cheat…

- If you can guarantee the Service is in the same process
- Then as they are singletons just store a publicly accessible reference
- And use that…