G54MDP
Mobile Device Programming

Lecture 6 – Android Activities,
Threads and Services

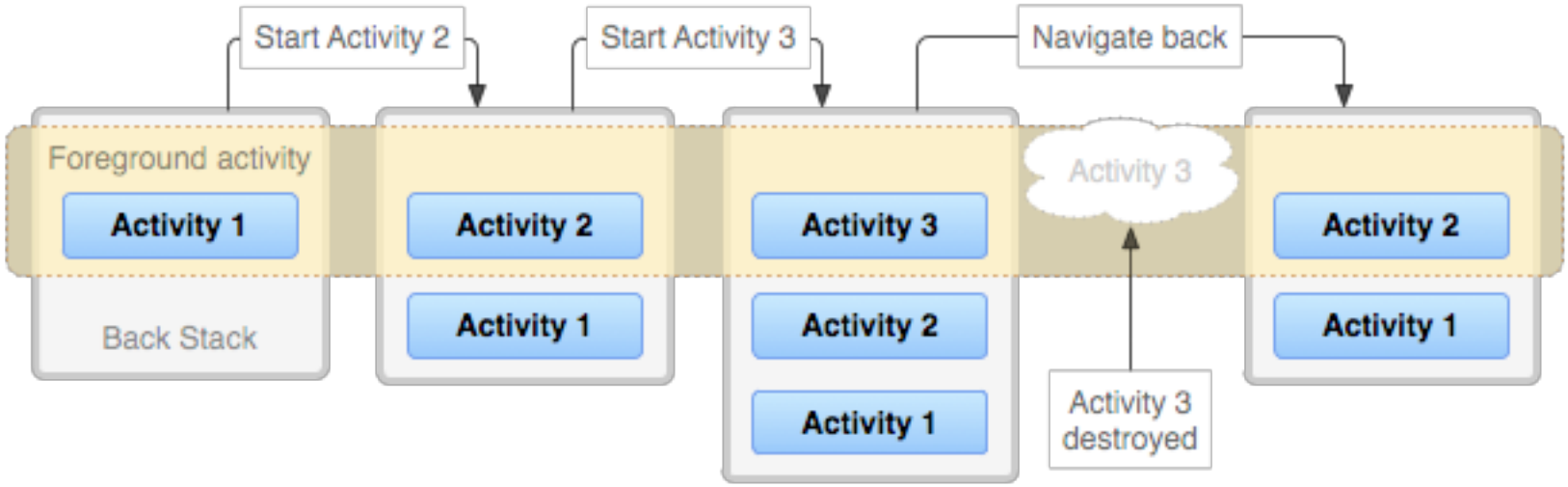
Intent

- Activities are started by sending an **Intent**
- Represented by an **Intent** object
  - Contains the name of the action requested
  - And the URI of the data to act on

```
Uri webpage = Uri.parse("http://www.cs.nott.ac.uk");
Intent myIntent = new Intent(Intent.ACTION_VIEW, webpage);

Uri number = Uri.parse("tel:01151234567");
Intent myIntent = new Intent(Intent.ACTION_DIAL, number);

Intent myIntent = new Intent(this, otherActivity.class);

startActivity(myIntent);
```

Intents

- Don't just instantiate the Activity sub-class
  - Already noted that Android works by passing Intent objects about
    - Intent is used to describe an operation
    - Action and the data to operate on (as a URI)
    - Allows for runtime binding
- Starting an Activity
  - Create a new Intent object
  - Specify what you want to send it to
    - Either implicitly, or explicitly
  - Pass the Intent object to **startActivity()**
    - New Activity then started
- Stopping an Activity
  - The called Activity can return to the original one by destroying itself
    - By calling the method **finish()**
  - Or when the user presses the back button


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

Inter-Activity Communication

- We've decomposed a task into multiple activities
  - How do Activities communicate?
  - String otherClass.doStuff(String arg2, MyObject b);
  - Potentially cross-process (IPC)
    - Across memory boundaries enforced by the kernel
  - NB! **Classes** vs **Activities**
- startActivity()
- startActivity(send some data to the new activity)
- startActivityForResult()
- startActivityForResult(send some data)
  - …expect some data back

Inter-Activity Communication

- startActivity() doesn't allow the Activity to return a result
  - Applications usually want to maintain state
    - Remember what the user has done across all activities
    - We could store state in the broader Application context
      - But activities may be communicating between processes (IPC)
      - Entry point for other applications
- startActivityForResult()
  - Still takes an Intent object, but also a numerical request code
    - Returns an integer result code, set with setResult()
- onActivityResult() then called on the calling Activity
  - Data can be packaged up in an Intent / Bundle
    - Activity creates an Intent object containing the result
    - Use a Bundle to "bundle" complicated objects
  - Calls setResult() to return the Intent
  - Intent object then passed to onActivityResult() on finish

Let’s have a look…

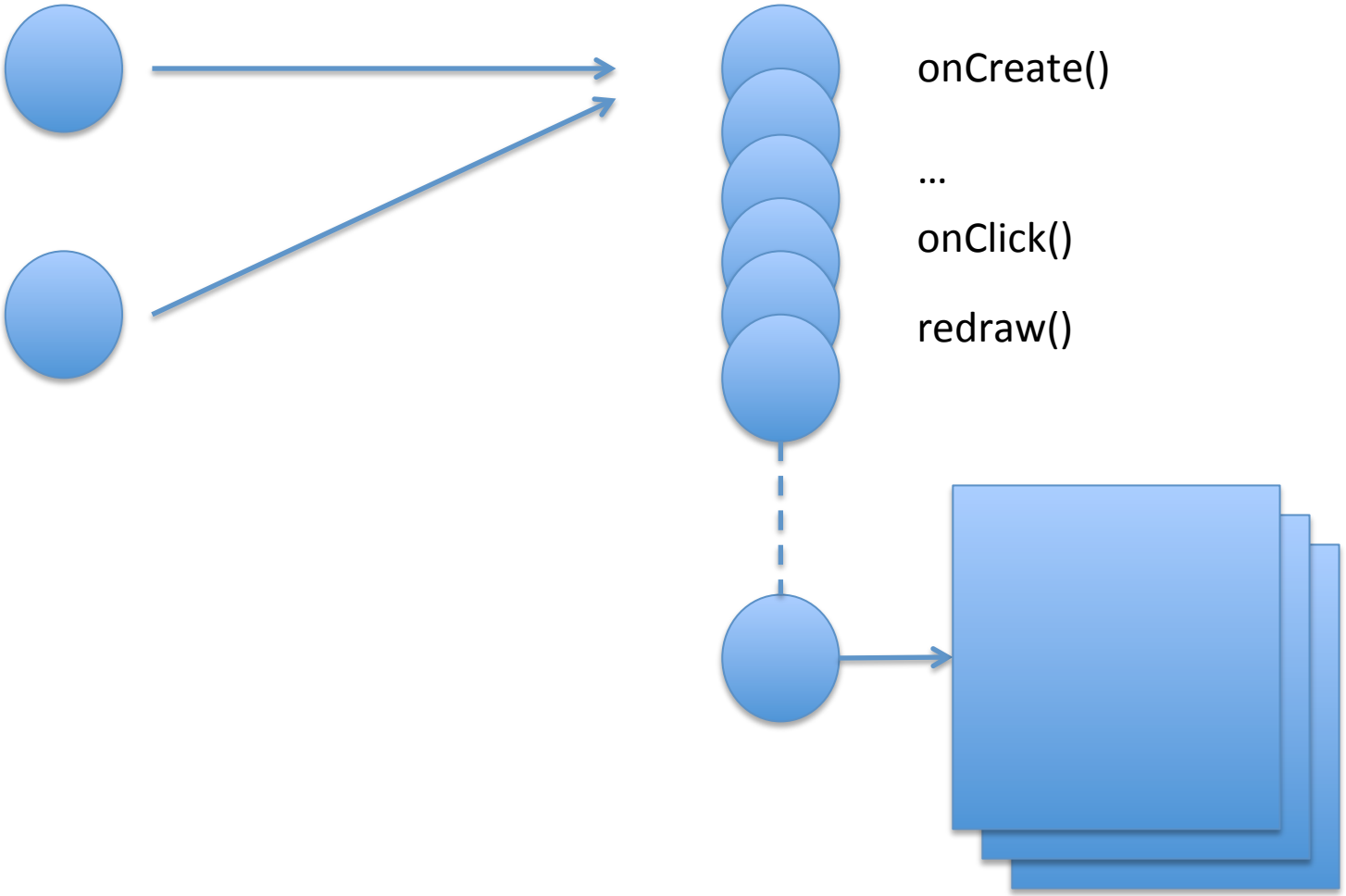
Threads and Services

- How long do things take?
- Threads
  - Interacting with the UI thread
- Services
  - Application component #2
  - The Service lifecycle

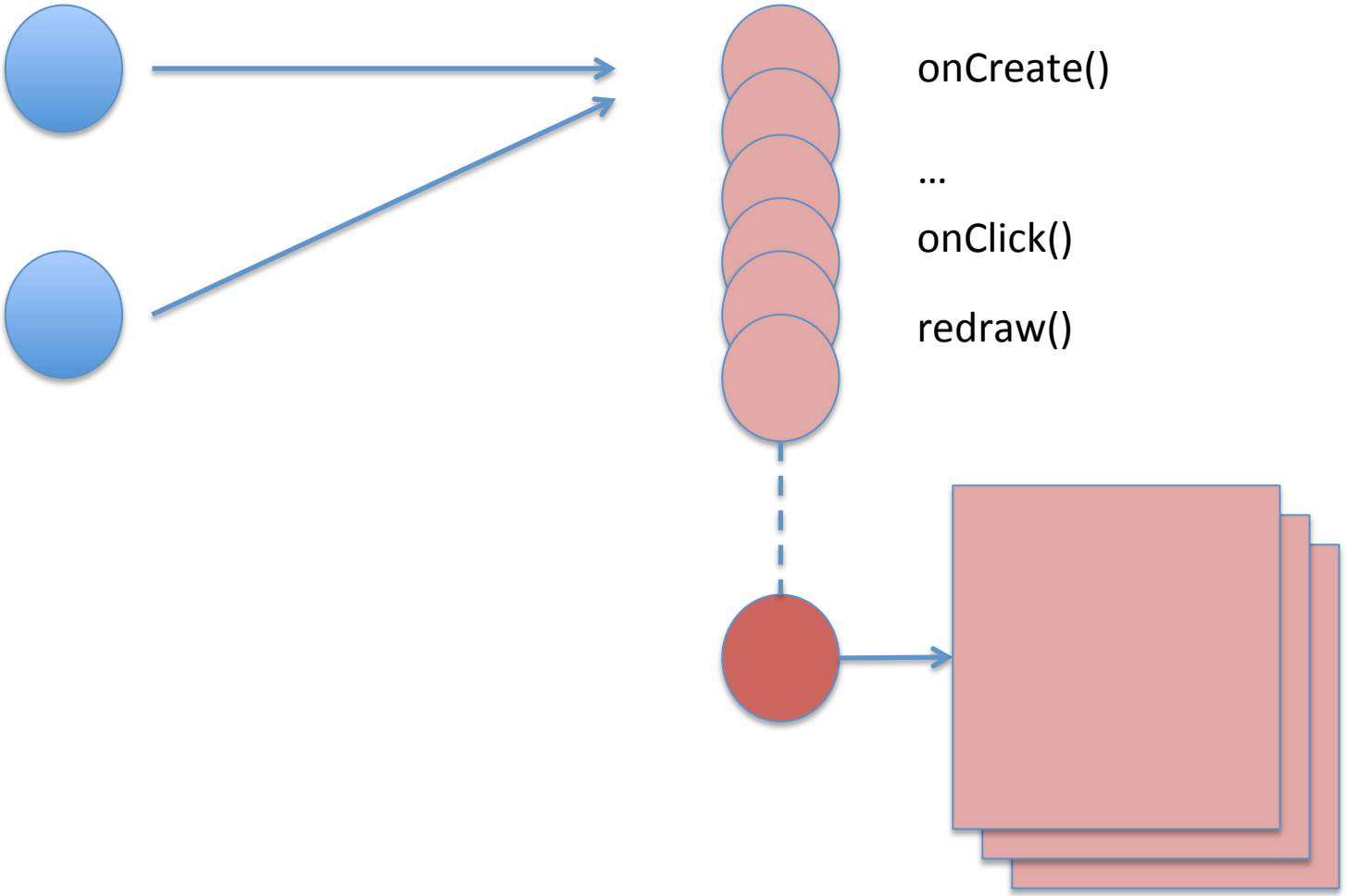
Threads

- 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

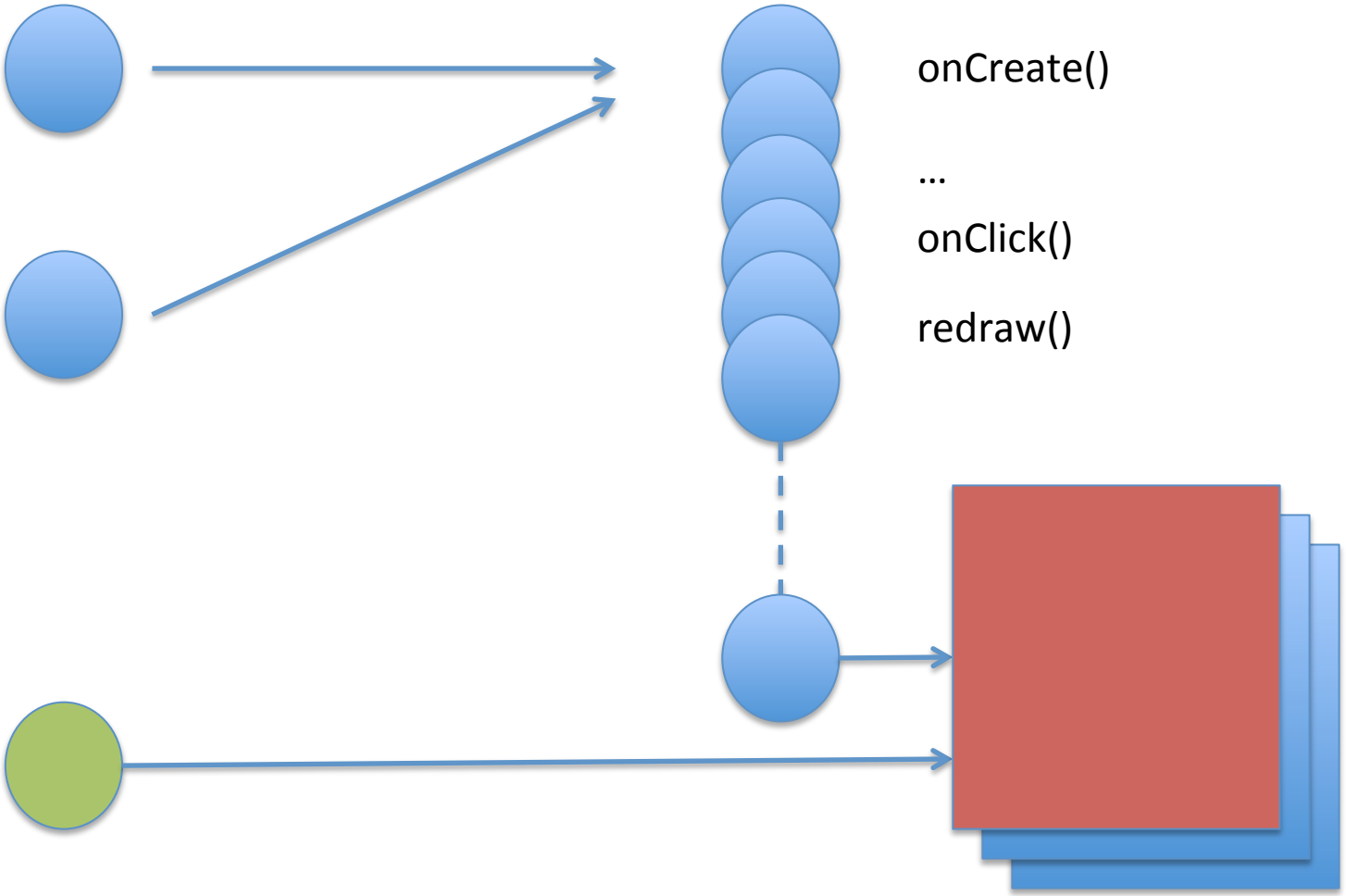
Threads / Looper



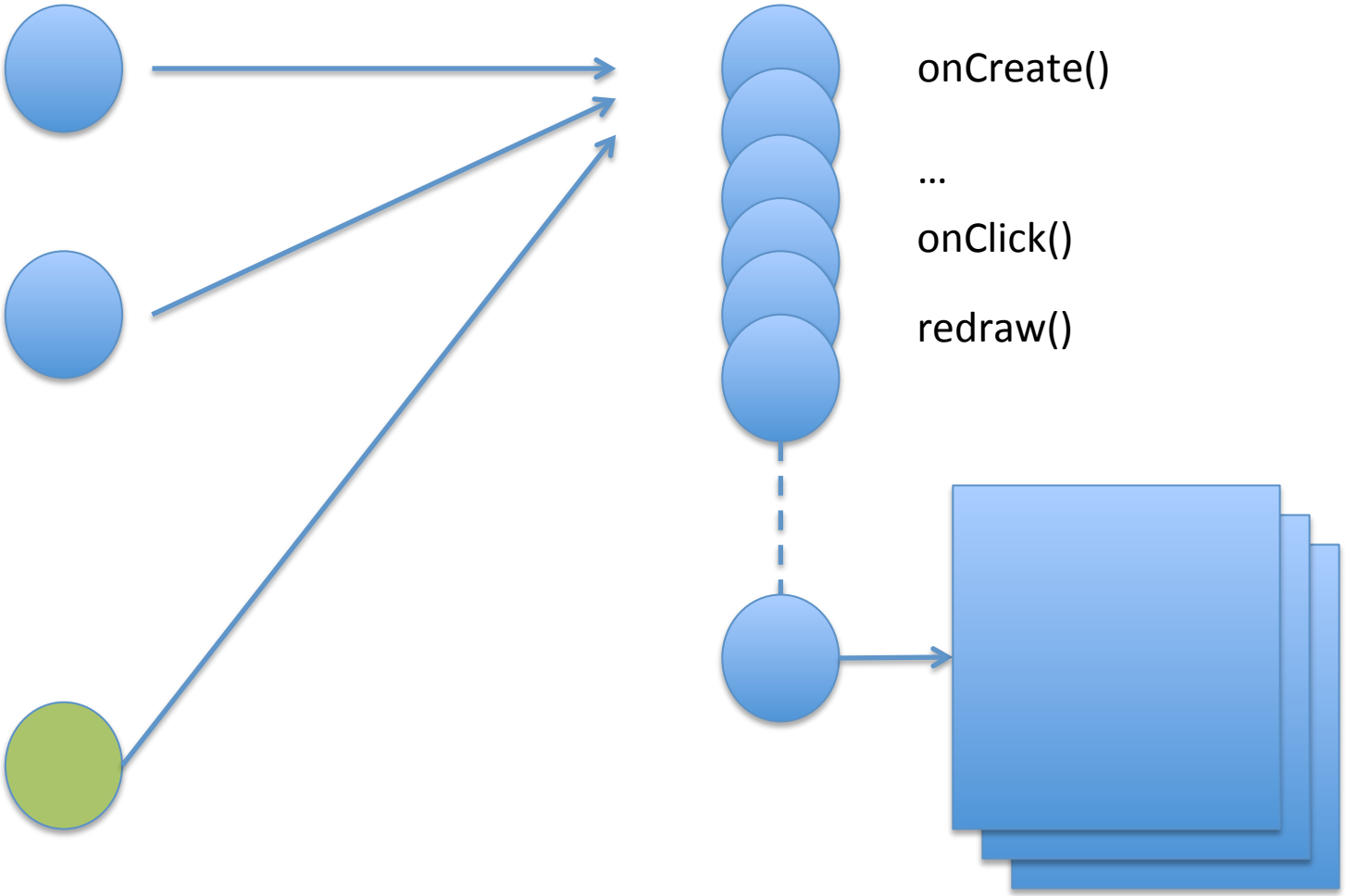
Threads / Looper



Threads / Looper



Threads / Looper



Threads

- How do we then execute code that may take a long time?
  - A long time > 1s
  - The application will appear to hang
  - “Application not responding” after 5s
- Put longer-running code / not instantaneous code in a separate thread of execution
  - Network access, file access
- Two golden rules
  - Do not block the UI thread
  - Do not access the UI thread from outside the UI thread
    - Concurrency!

Runnable

- We can programmatically interact with UI components
  - myTextField.setText(result);
  - Cannot call this method from outside the UI thread
    - Rule number 2
- Instead, split code into two parts
  - Long (ish) running code that does not involve the UI
    - E.g. an image download
      - Occurs in a separate thread of execution
      - Still tightly coupled to an activity
  - Instantaneous code that does involve the UI
    - E.g. drawing the image that has been downloaded
    - **posted** to the UI thread responsible for a particular View to execute, logically parceled up as a **Runnable** object

Handlers

- Provide a thread-safe way of talking to a specific thread of execution
  - Schedule messages and runnables to be executed at some point in the future
    - Runnable – a package of code
    - Message – a package of data
  - Enqueue an action to be performed on a different thread than your own
    - UI thread -> worker thread
    - Worker thread -> UI thread
- Activity.runOnUiThread(Runnable …)

AsyncTask

- A convenience class for making complex asynchronous worker tasks easier
- Worker / blocking tasks
  - Executed in a background thread
- Results callback
  - Executed in the UI thread

Let’s have a look…

Services

- An Application **Component** that
  - Has no UI
  - Represents a desire to perform a longer-running operation
    - I.e. longer than a single-activity element of the task
- 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

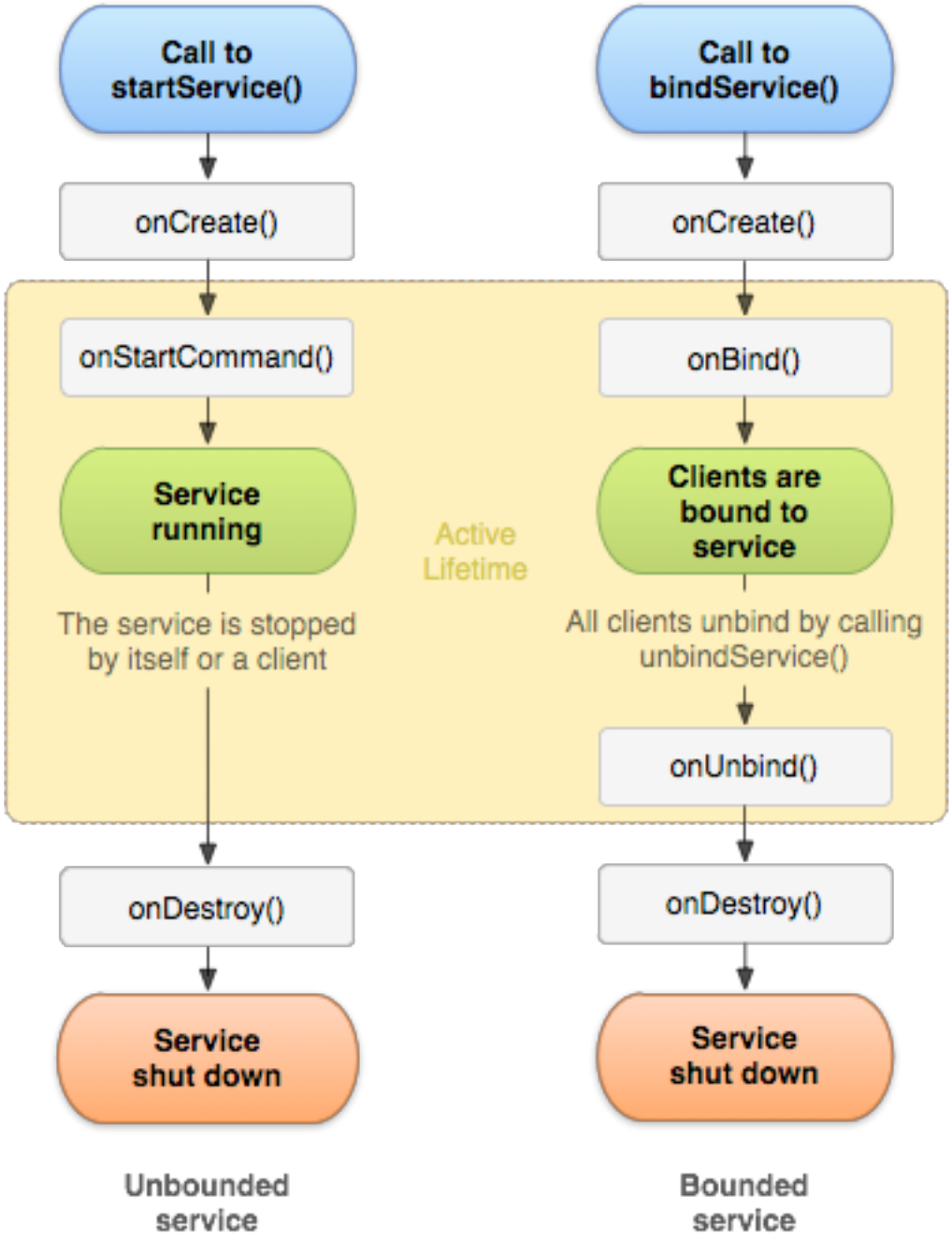
- Services are designed to support communication with
  - Local Activities (in the same process)
  - Remote Activities (in a different process)
    - IPC
- Services are components, similar to an Activity
  - Register the service in the manifest
  - Create a subclass of android.app.Service

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
Active Lifetime
Call to bindService()
onCreate()
onBind()
Clients are bound to service
All clients unbind by calling unbindService()
onUnbind()
onDestroy()
Service shut down
Bounded service

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

Let’s have a look…

References

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

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