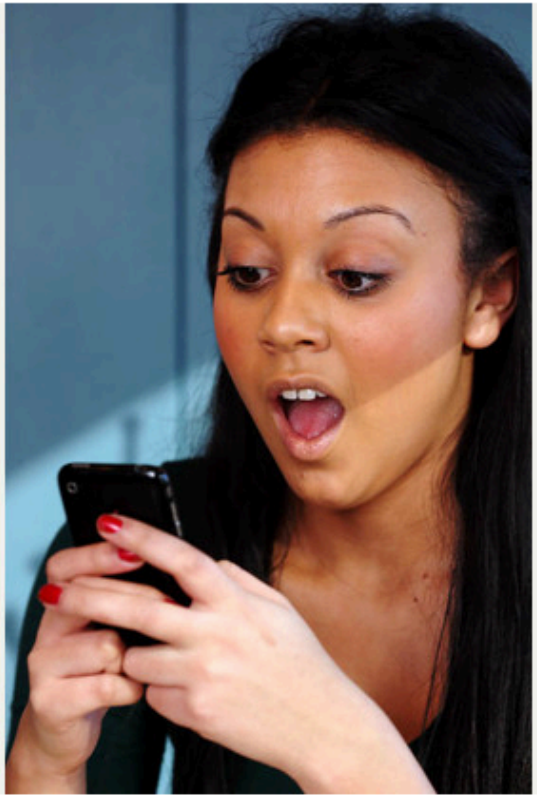
G54MDP
Mobile Device Programming

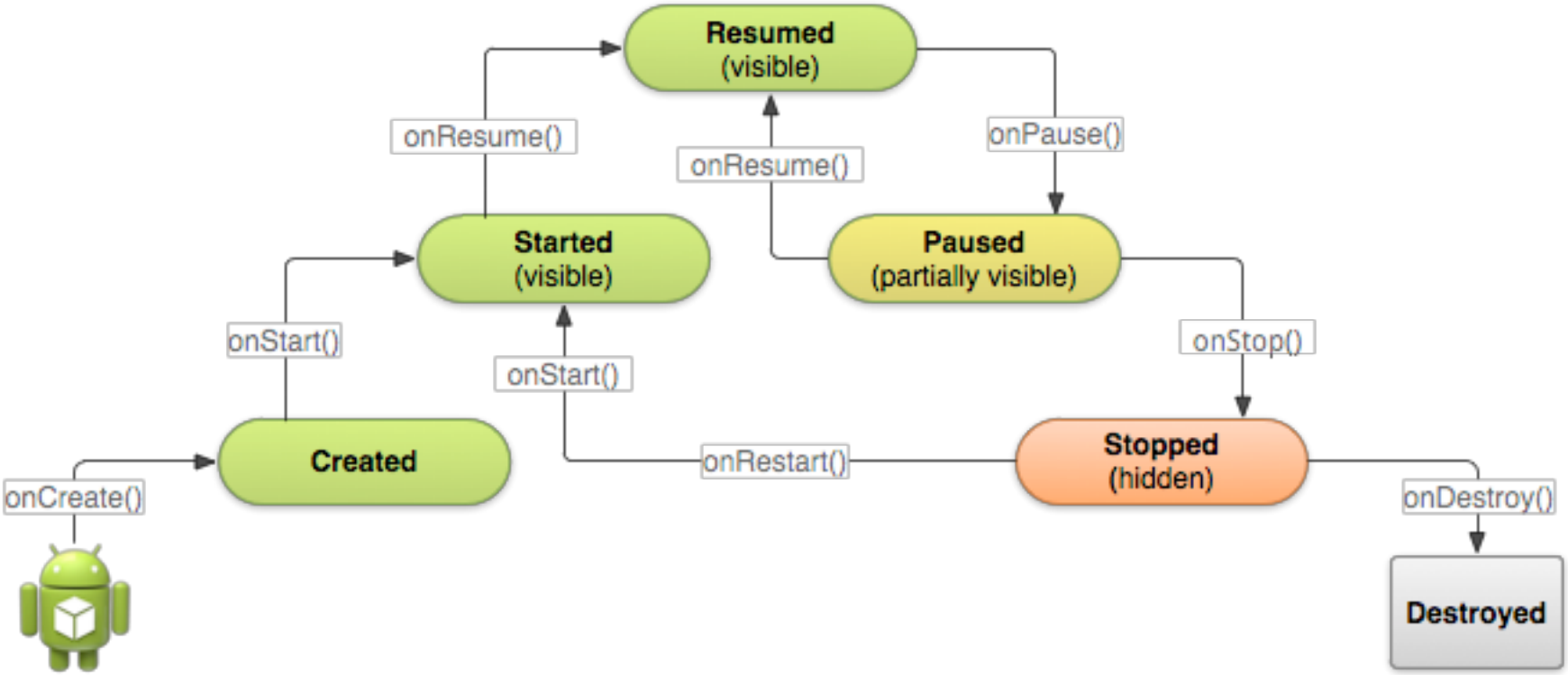
Lecture 6 – Threads and Services

Flappy Bird: Thousands of people across Notts sell iPhones to cash in on gaming craze

By **Nottingham Post** | Posted: February 12, 2014



Arabella Griffin says she spends up to three hours a day on Flappy Bird


Resumed
(visible)
onResume()
onResume()
onPause()
Started
(visible)
Paused
(partially visible)
onStart()
onStart()
onStop()
Created
onRestart()
Stopped
(hidden)
onCreate()
onDestroy()
Destroyed

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

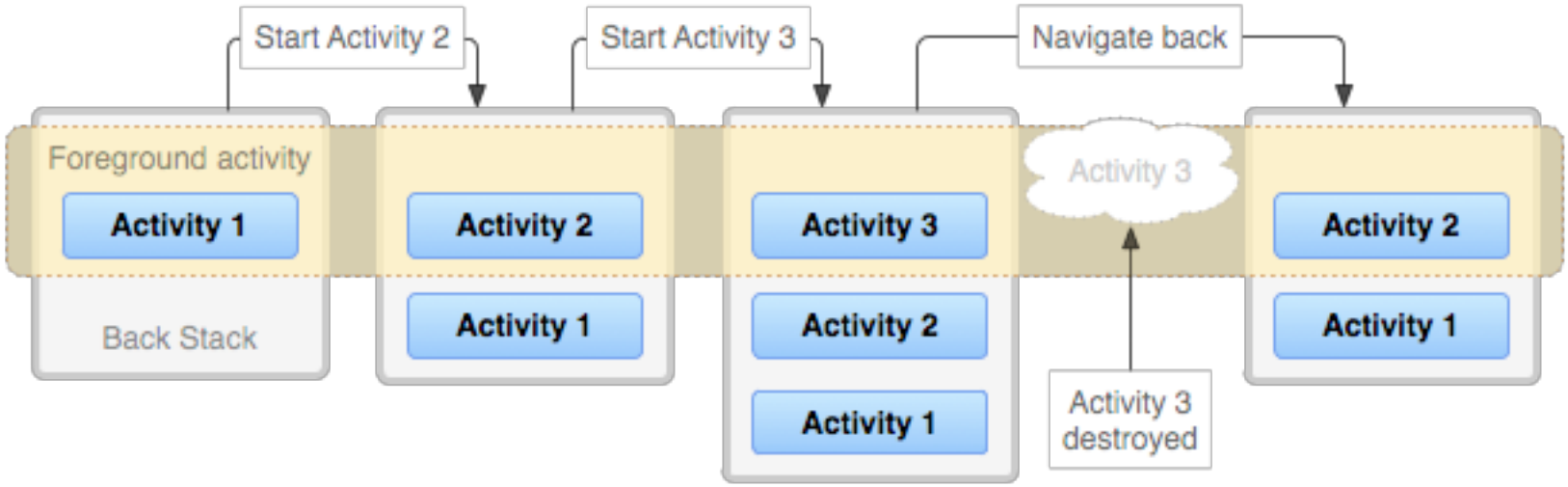
```
Uri webpage = Uri.parse("http://www.cs.nott.ac.uk");
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);

Uri number = Uri.parse("tel:01151234567");
Intent callIntent = 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…

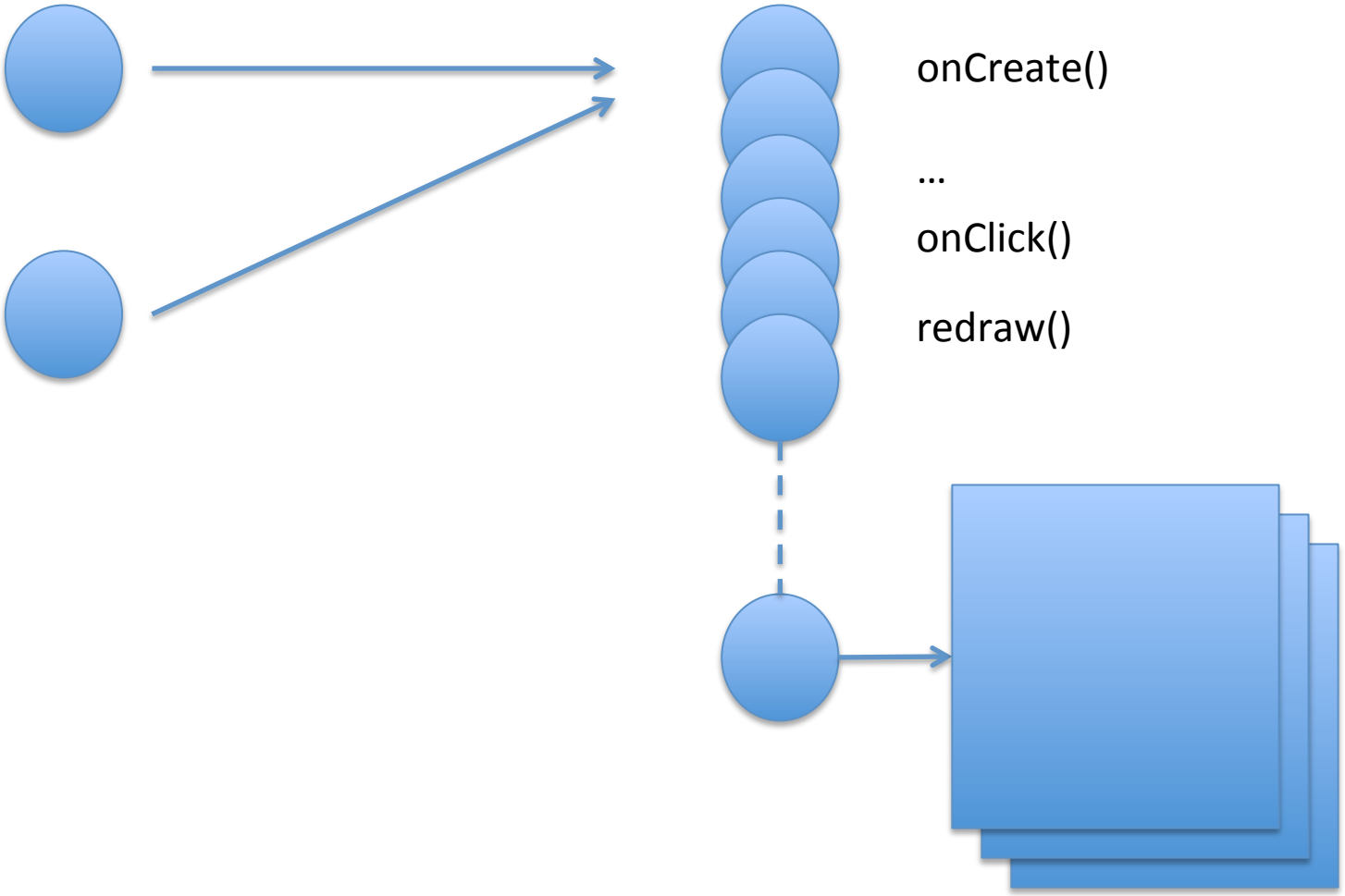
Threads and Services

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

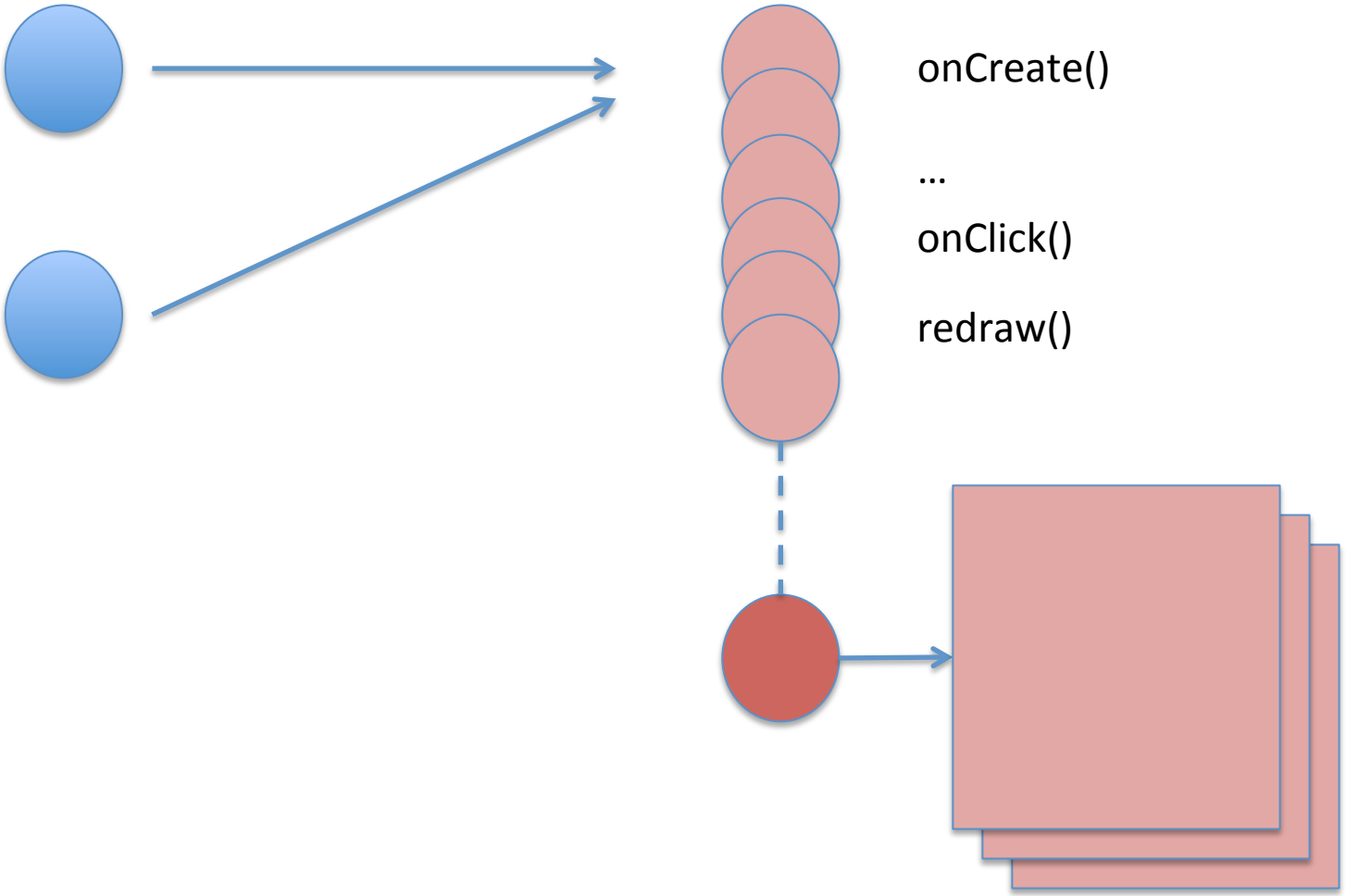
Threads

- 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

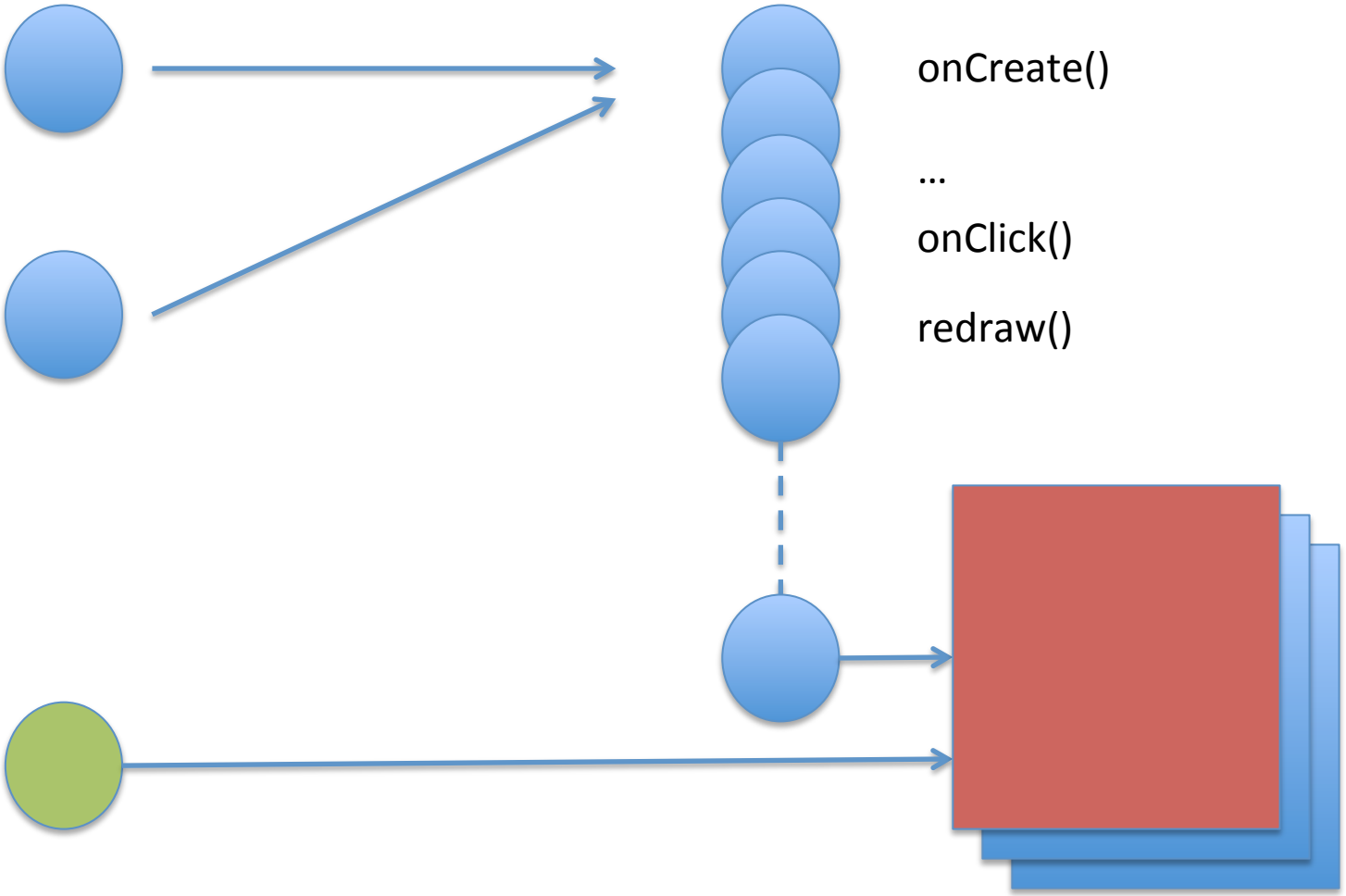
Threads / Looper



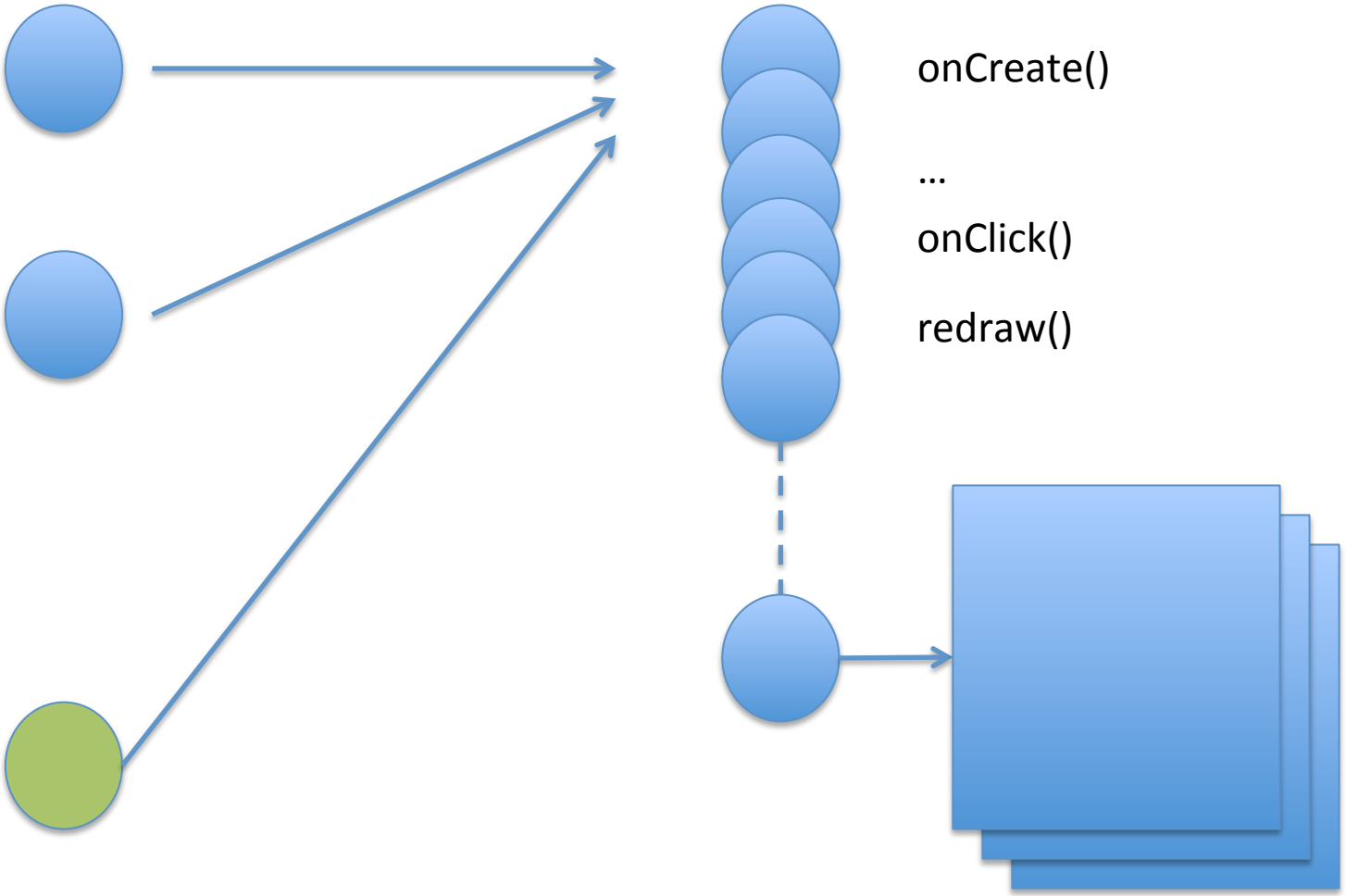
Threads / Looper



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 long-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 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, 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…

References

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

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