Lecture 6

Features of a mobile OS

Android

- Android provides an operating system (Linux kernel), plus middleware and applications – it's a general-purpose system for mobile devices.
- The concept is that *"the handheld is the new PC"*.
- Android has its own JVM, called Dalvik VM.
- *One key architectural goal*: allow applications to interact with one another and reuse components from one another.
- The reuse applies not only to services but also to data and UI.

Android software stack

- Android core is the *Linux kernel v 4.xx; device drivers include Display, Camera, Keypad, WiFi, FlashMemory, Audio, IPC.*
- A set of C/C++ libraries sit on top of the kernel: OpenGL, WebKit (browser support), FreeType (font support), SSL (secure sockets library), the C runtime library (libc), SQLite (relational database available on the device) and Media.
- The Java API's main libraries include resources, telephony, locations, UI, content providers (data) and package managers.
- On the very top are user applications such as Home, Contacts, Phone, Browser, etc.
- Android is a multi-user Linux system in which *each app is a different user* – it is assigned a *unique Linux ID.*

A. Android applications

- An application usually contains multiple activities. Each activity should be designed around a specific kind of action the user can perform and can start other activities. For example, an email application might have one activity to show a list of new messages. When the user selects a message, a new activity opens to view that message.
- Properly behaved apps running in the background aren't actually doing anything — they're just remaining in memory and using no CPU or other resources.
- When the user accesses them again, they'll quickly open, as they're waiting in memory. If they were removed from memory, they would take longer to re-open as their data would have to be transferred from system storage back into RAM.

Intent

- Android applications don't have a single entry point for everything in the application (no *main() function, for example). They have components that the system can instantiate and run as needed.*
- An *intent* is an amalgamation of ideas such as windowing messages, actions, inter-process communication, publish/subscribe models and application registries.
- *Example:*

```
public static void invokeWebBrowser(Activity activity)
{
        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse(http://www.ucc.ie));
        activity.startActivity(intent);
}
```

- Android is asked to start a window to display the content of a web site.

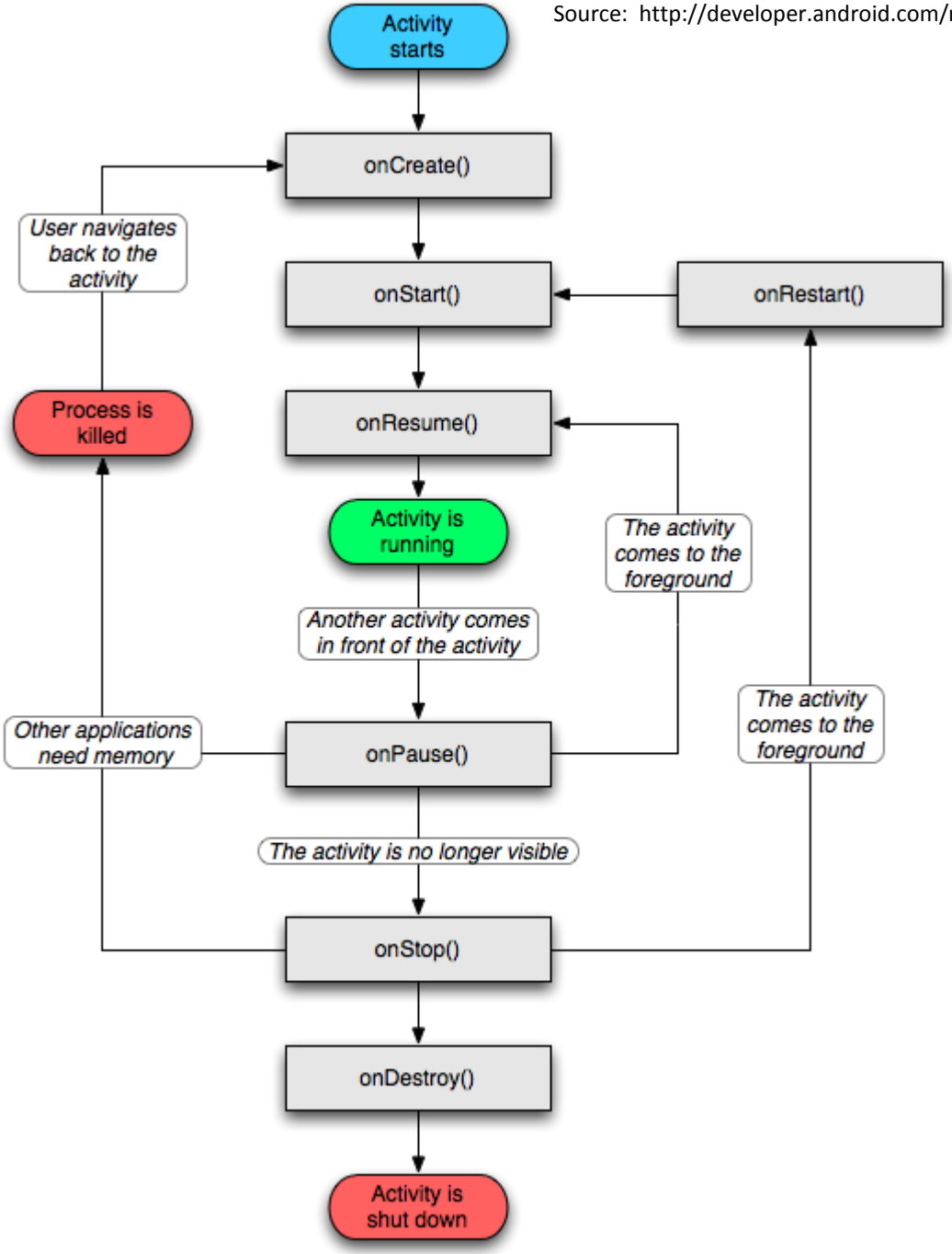
Activating Android components

- An *activity* is launched, or given something new to do by passing an Intent object to Context.startActivity() or Activity.startActivityForResult().
- The responding activity can look at the initial intent that caused it to be launched by calling its getIntent() method. One activity often starts the next one.
- If it expects a result back from the activity it's starting, it calls startActivityForResult() instead of startActivity(). For example, if it starts an activity that lets the user pick a photo, it might expect to be returned the chosen photo. The result is returned in an Intent object that's passed to the calling activity's onActivityResult() method.

B. Android application lifecycle

- The Android application lifecycle is managed by the system based on the user needs, available resources, etc.
- The system decides if an application can be loaded or it is paused or stopped.
- The activity currently used gets higher priority while an activity not visible can be killed to free resources.
- *Each Android application runs in a separate process which hosts its own virtual machine. This is a protected-memory environment. Then, its priority can be controlled by the system.*

Source: http://developer.android.com/reference/android/app/Activity.html



C. Activities and tasks

- A *task* is *a stack of activities*. There is no way to set values for a task independently of its activities. Values for the task as a whole are set in the *root activity (launcher).*
- One activity can start another one, including one defined in a different application. For example, the user wants to display a street map of some location. There's already an activity that can do that, so all your activity needs to do is put together an Intent object with the required information and pass it to startActivity(). The map viewer will display the map. When the user hits the BACK key, your activity will reappear on screen. To the user, it will seem as if the map viewer is part of the same application, even though it's defined in another application and runs in that application's process.

- The task activities are arranged in a (back) stack:
  - the root activity in the stack is the one that began the task — typically, it's an activity the user selected in the application launcher.
  - The activity at the top of the stack is one that's currently running — the one that is the focus for user actions.
  - When one activity starts another, the new activity is pushed on the stack; it becomes the running activity. The previous activity stops but remains in the stack. When an activity stops, the system retains the current state of its user interface.
  - When the user presses the BACK key, the current activity is popped from the stack, and the previous one resumes as the running activity.

Android multitasking

- A task can move to the background when the user begins a new task or go to the Home screen, via the *Home* button. While in the background, all the activities in the task are stopped, but the back stack for the task remains. A task can return to the foreground so the user can pick up where s/he left off – the activity at the top of the stack is resumed.
- Multiple tasks can be held in the background at once. However, if the user is running many background tasks at the same time, the system might begin destroying background activities in order to recover memory, causing the activity states to be lost.
- If the user leaves a task for a long time, the system clears the task of all activities except the root activity. When the user returns to the task again, only the root activity is restored.

Conclusions

- Each OS is a set of services that abstracts the specific machine architecture.
- There are many different OS, depending on their range of use and the characteristics of the host devices.

- Suggested further study:
  - compare Tiny OS and Android in terms of how processes are defined and managed by the system;
  - a foreground activity has the highest priority. How is the system managing priorities?