Lecture 7

Distinctive features of a mobile OS

OS for mobile devices

- The mobile OSs adapt to the characteristics of mobile devices.
- Resources are scarce, especially the battery energy, but still significant (e.g., in respect to a sensor). For example, there are many cores, the memory size is in terms of GB, lots of sensors are present. User interaction is by touch screen and app visibility plays a key role.
- Energy saving is a key goal of the mobile OS and apps.
- The application model corresponds to user interaction and sharing of data among applications.

Android

- Android is not only an operating system (it uses the Linux kernel), but it also includes middleware and applications – it's a general-purpose operating system for mobile devices.

- The concept is that *"the handheld is the new PC"*.

- Android started with its own JVM, called Dalvik VM, but later replaced it with Android Runtime.

- *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

- Android applications don't have a single entry point (no *main() function, for example). They have accessible components (activities) that the system can instantiate and run as needed.*
- An application usually contains multiple activities – an activity is a UI concept. Each activity is designed around a specific kind of action the user can perform; an activity can start other activities. For example, an email application has one activity to show a list of new messages. When the user selects a message, a new activity opens to view that message.
- Each Android application runs in a separate process, in its own VM. This is a protected-memory environment.
- *The application (process) priority can be controlled by the system. The process starts when any of the app's components needs to be executed, and then shuts down when it's no longer needed or when the system must recover memory for other apps.*

Intents

- Intents define “intention” to do some work – broadcast a message, start a service, launch an activity, dial a phone number or answer a call. They can also be used by the system to notify the application of specific events (i.e. arrival of a text message).

- *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);
}
```

In this example, Android is asked to start a window to display the content of a web site.

The intent object

- An intent object contains information of interest to the component that receives the intent (such as the action to be taken and the data to act on), and information of interest to the Android system (such as the category of component that should handle the intent and instructions on how to launch a target activity).

- It includes, optionally,
  - the *component name* that should handle the intent; if it is not set, Android uses other information in the Intent object to locate a suitable target;
  - a *string naming the action* to be performed;
  - the *URI of the data* to be acted on and the MIME type of that data;
  - a *string containing additional information* about the kind of component that should handle the intent - any number of category descriptions can be placed in an Intent object.

Intents can be:

1. *Explicit intents* designate the target component by its name. Since component names would generally not be known to developers of other applications, explicit intents are typically used for application-internal messages — such as an activity starting a subordinate service or launching another intra-process activity. Android delivers an explicit intent to an instance of the designated target class. Nothing in the intent object other than the component name matters for determining which component should get the intent.

2. *Implicit intents* do not name a target. Implicit intents are often used to activate components in other applications. A different strategy is needed for implicit intents. In the absence of a designated target, the Android system must find the best component (or components) to handle the intent — a single activity or service to perform the requested action. It does so by comparing the contents of the intent object to *intent filters*, structures associated with components that can potentially receive intents.

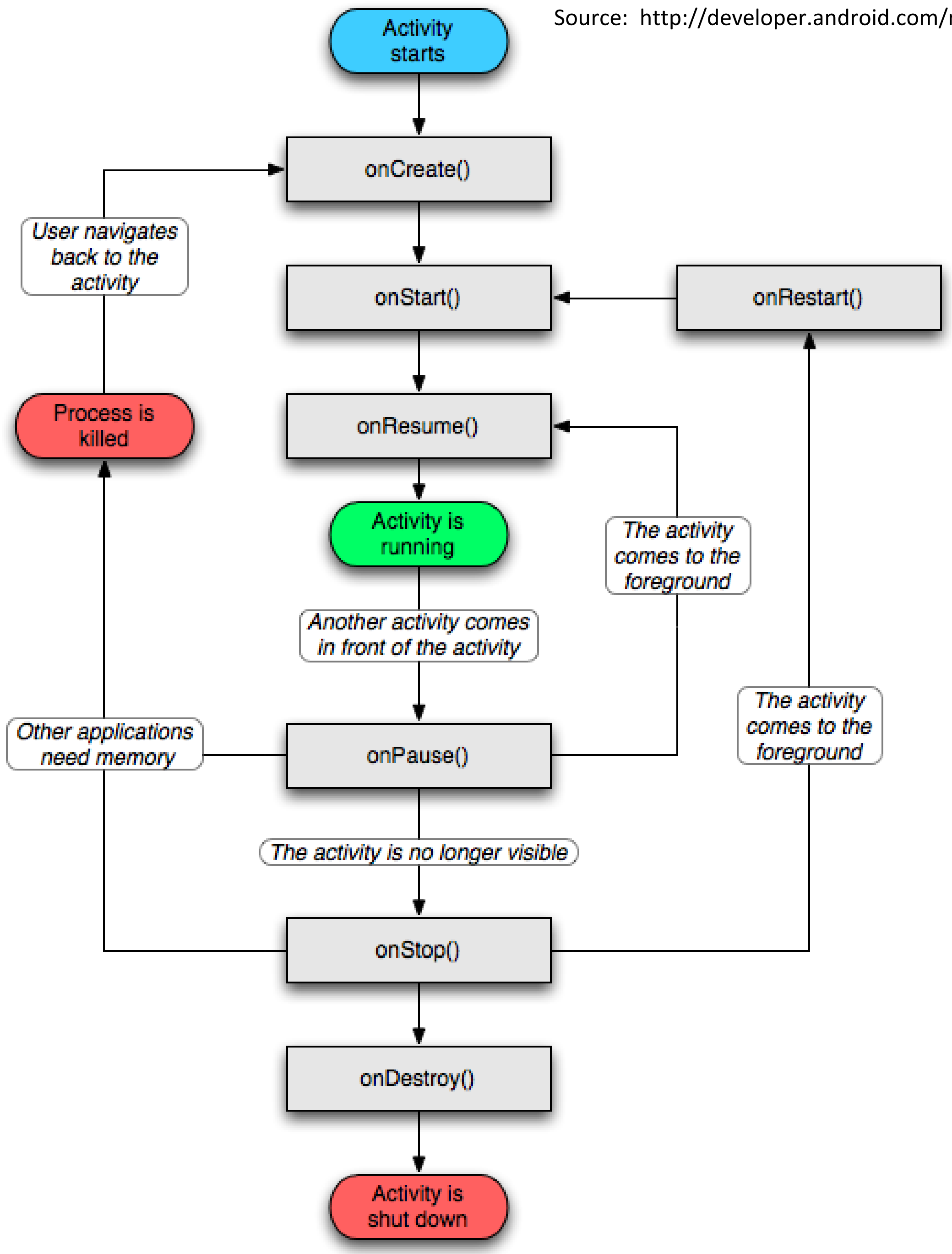
Activating Android components

- 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.
- 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.

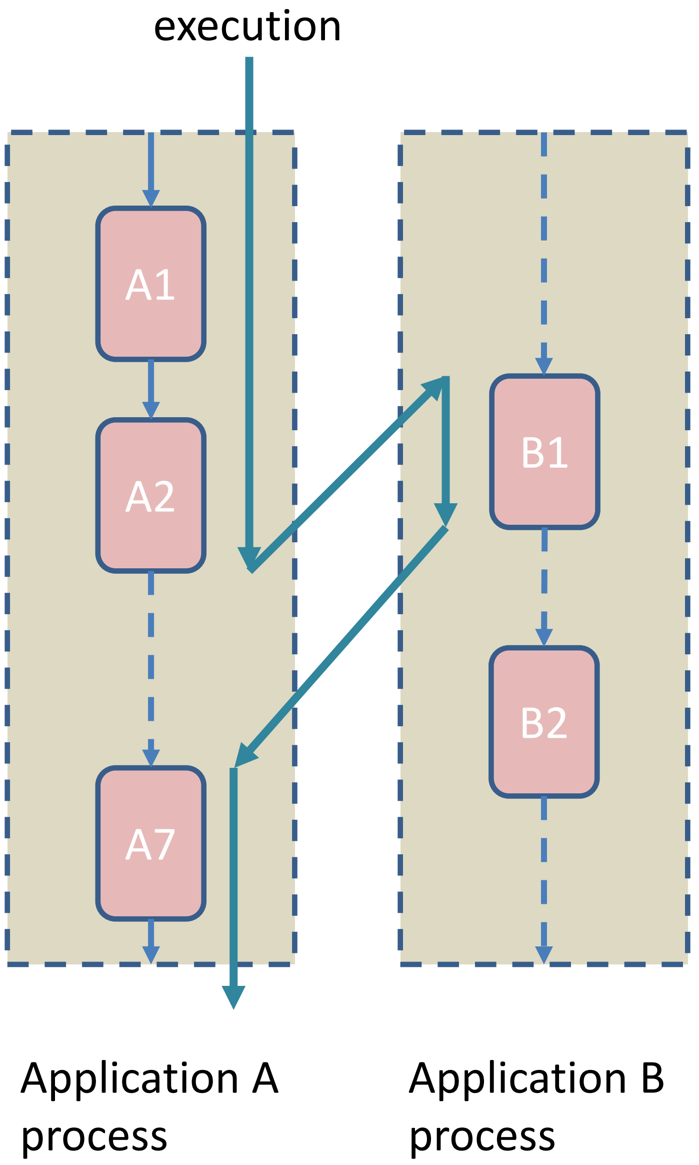
Source: http://developer.android.com/reference/android/app/Activity.html

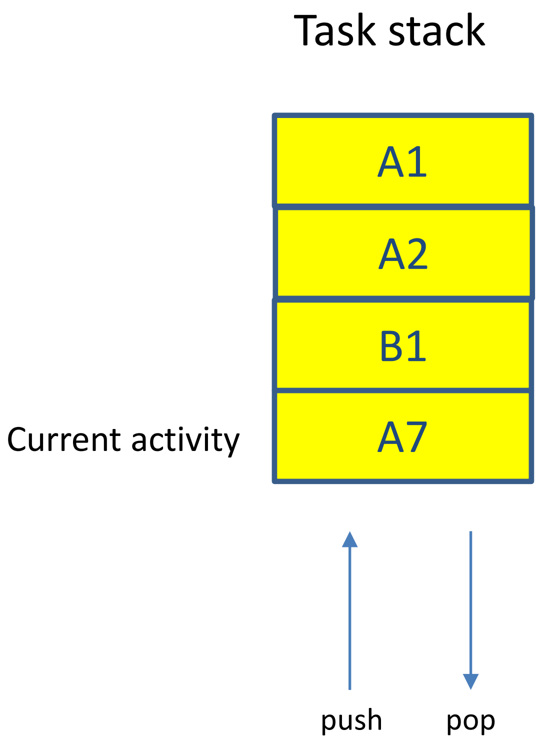

Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

C. Activities and tasks

- The execution flow of an app is managed by a task.
- 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 the 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, the 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 one, 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.


execution
A1
A2
A7
B1
B2
Application A process
Application B process


Task stack
A1
A2
B1
A7
Current activity
push
pop

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 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 there are 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

- The concept of activity and the transfer of control using intents are important features of Android.
- The activity lifecycle reflects the visibility and interaction with the user.
- The task and the corresponding stack follow the execution flow within the same application (intra-process) or across applications (inter-processes).

- 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?