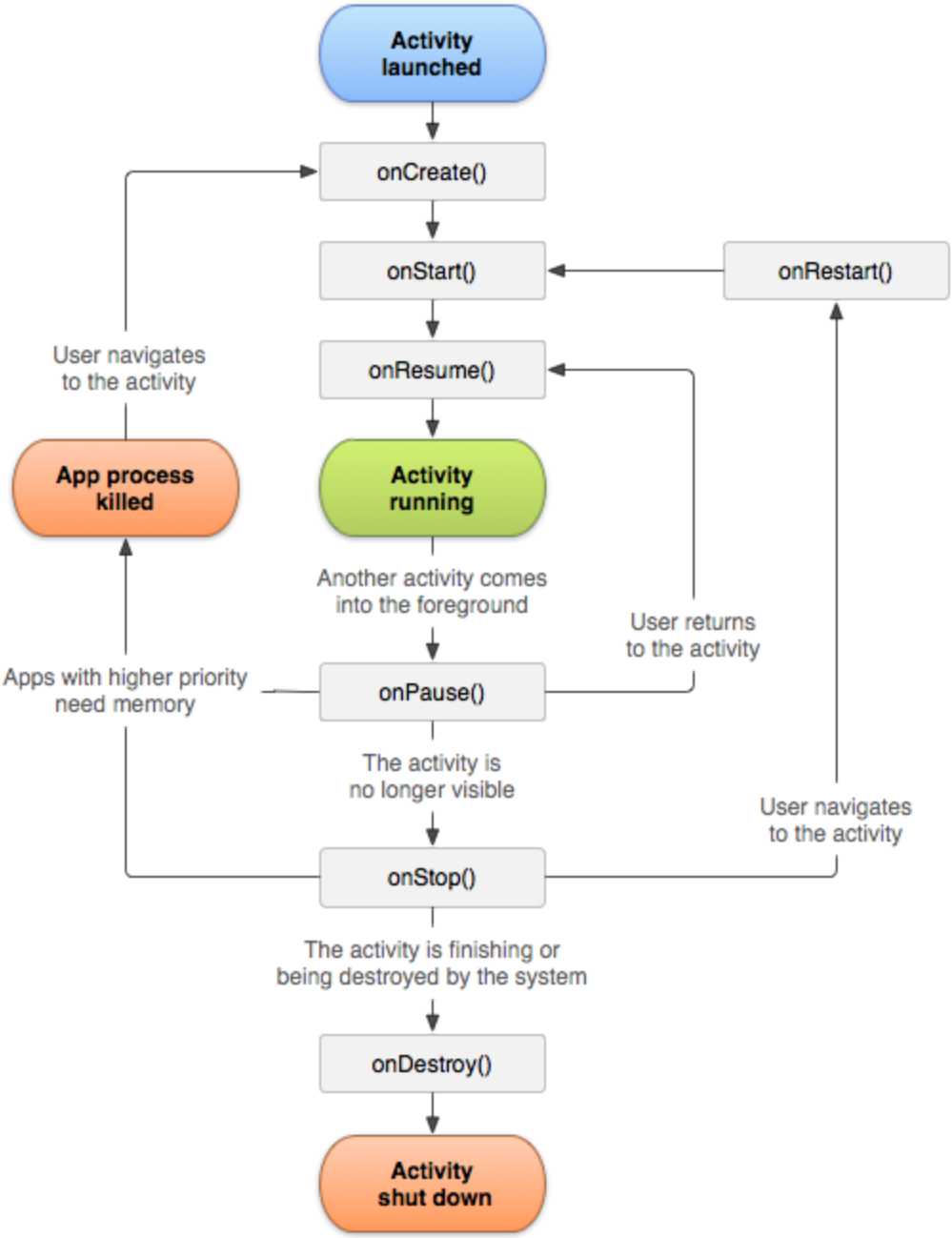
The Activity Lifecycle

Although we never call a constructor or `main`, Activities do have an *incredibly* well-defined lifecycle—that is, a series of **events** that occur during usage (e.g., when the Activity is created, when it is stopped, etc).

When each of these events occur, Android executes a **callback method**, similar to how you call `actionPerformed()` to react to a "button press" event in Swing. We can **override** these methods in order to do special actions (read: run our own code) when these events occur.

What is the lifecycle?



Lifecycle state diagram, from Google2. See also an alternative, simplified diagram here.
There are 7 “events” that occur in the Activity Lifecycle, which are designated by the *callback function* that they execute:

- `onCreate()`: called when the Activity is **first** created/instantiated. This is where you initialize the UI (e.g., specify the layout to use), similar to what might go in a constructor.
- `onStart()`: called just before the Activity becomes **visible** to the user.
  The difference between `onStart()` and `onCreate()` is that `onStart()` can be called

more than once (e.g., if you leave the Activity, thereby hiding it, and come back later to make it visible again).

- **`onResume()`**: called just before **user interaction** starts, indicating that the Activity is ready to be used! This is a little bit like when that Activity "has focus".
  While `onStart()` is called when the Activity becomes visible, `onResume()` is called when then it is ready for interaction. It is possible for an Activity to be visible but not interactive, such as if there is a modal pop-up in front of it (partially hiding it).
- **`onPause()`**: called when the system is about to start another Activity (so about to lose focus). This is the "mirror" of `onResume()`. *When paused, the activity stays visible!*
  This callback is usually used to *quickly and temporarily* store unsaved changes (like saving an email draft in memory) or stop animations or video playback. The Activity may be being left (on its way out), but could just be losing focus.
- **`onStop()`**: called when the activity is no longer visible. (e.g., another Activity took over, but this also be because the Activity has been destroyed. This callback is a mirror of `onStart()`.
  This callback is where you should persist any state information (e.g., saving the user's document or game state). It is intended to do more complex "saving" work than `onPause()`.
- **`onRestart()`**: called when the Activity is coming back from a "stopped" state. This event allows you to run distinct code when the App is being "restarted", rather than created for the first time. It is the least commonly used callback.
- **`onDestroy()`**: called when the Activity is about to be closed. This can happen because the user ended the application, ***or*** (and this is important!) because the OS is trying to save memory and so kills the App.
  Android apps run on devices with significant hardware constraints in terms of both memory and battery life. Thus the Android OS is very aggressive about not leaving Apps running "in the background". If it determines that an App is no longer necessary (such as because it has been hidden for a while), that App will be destroyed. Note that this destruction is unpredictable, as the "necessity" of an App being open is dependent on the OS's resource allocation rules.
  The `onDestroy()` callback can do final app cleanup, but its better to have such functionality in `onPause()` or `onStop()`.

Note that apps may not need to use all of these callbacks! For example, if there is no difference between starting from scratch and resuming from stop, then you don't need an `onRestart()` (since `onStart()` goes in the middle). Similarly, `onStart()` may not be needed if you just use `onCreate()` and `onResume()`. But these lifecycles allow for more granularity and the ability to avoid duplicate code.