# EE 579: WIRELESS AND MOBILE NETWORKS – DESIGN & LABORATORY

# LECTURE 3

Amitabha Ghosh

Department of Electrical Engineering
USC, Spring 2014

Lecture notes and course design based upon prior semesters taught by Bhaskar Krishnamachari and Murali Annavaram.

# Agenda

- Administrative Stuff
- Android architecture overview
- Fun with math
- Install software and get started

# Mobile OS

- Symbian – most popular smartphone OS until 2010
- iOS
- RIM's BlackBerry
- Windows Mobile
- Linux
- Palm webOS
- Android
  - 81% global smartphone market share as of Nov 2013, led by Samsung products
  - 1 billion devices activated; 48 billion apps installed from Google Play store

# PhoneSat – NASA launches three Google-HTC Nexus One Android powered mini-satellites into orbit, April 2013


- Slightly adapted for extra terrestrial activities – larger Li-ion batteries and solar cells, $3500
- Encased in 4-inch metal cubes
- Whizzing around the Earth at an altitude 150 miles
- Take photos of Earth and send back
- Designed to burn up on re-entry after two weeks

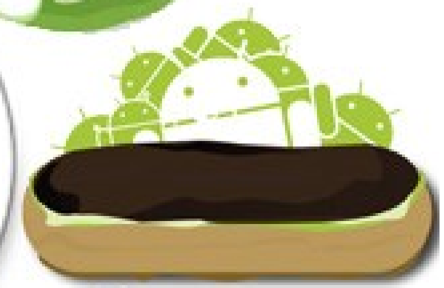
**Google acquires Android Inc. in 2005, Android 1.0 Astro, Sept 2008**

**Android 1.5 Cupcake**, April 2009, 1st commercially available version with Android's first touch-screen phone HTC Magic

**Android 1.6 Donut**, Sept 2009, text-to-speech technology, search by text and voice

**Android 4.1 Jellybean**, July 2012, Google Now, faster smoother more responsive

**Android 2.0/2.1 Eclair**, Oct 2009, live wallpapers, virtual keyboard, Bluetooth, HTML5, improved navigation with Google maps



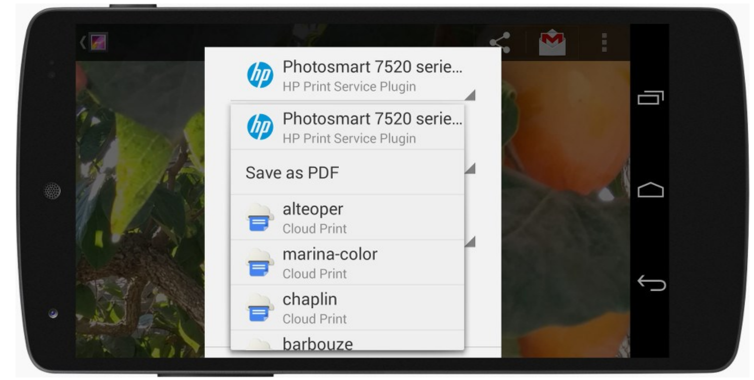
**Android 4.0 Ice Cream Sandwich**, Oct 2011, performance and speed, tablet features on smartphones, GTalk

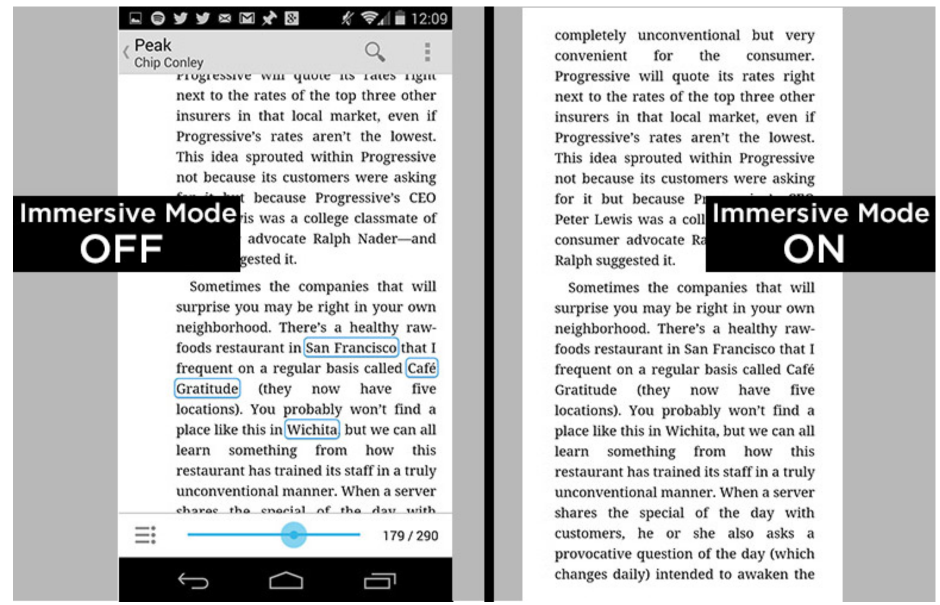
**Android 2.2 Froyo**, May 2010, OS speed with Java V8 engine and JIT compiler, Flash, remote wipe features

**Android 3.0 Honeycomb**, Feb 2011, designed for tablets, no need for physical buttons, system bar, action bar, redesigned keyboard

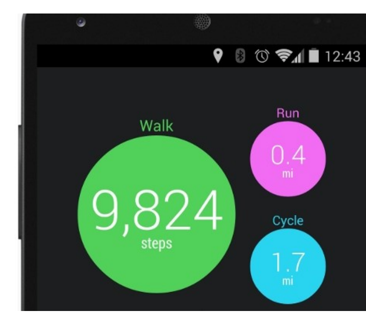
**Android 2.3 Gingerbread**, Dec 2010, quick front and back camera switch, better battery mgmt, near field communication (NFC) with Google Wallet


Nov 2013 (Latest)




- ❑ Fast and smooth on a range of devices, millions of entry-level devices < 512 MB RAM
- ❑ Printing over Wi-Fi or cloud
- ❑ Full-screen immersive mode (use every pixel, capture touch events)
- ❑ Secure NFC through Host Card Emulation (HCE)
- ❑ Low-power sensors (e.g., step detector and counter)

# Android Platform

- A multi-layered, open software stack for mobile devices (phones, tablets) for building and running mobile applications
  - OS kernel, System Libraries, Application Frameworks, Key Apps

- Android SDK for creating apps
  - GSM, EDGE, and 3G networks, Wi-Fi, Bluetooth
  - Libraries and development tools
  - Location-based service, map
  - Lots of documentation (Start browsing today!)
  - [http://developer.android.com](http://developer.android.com)

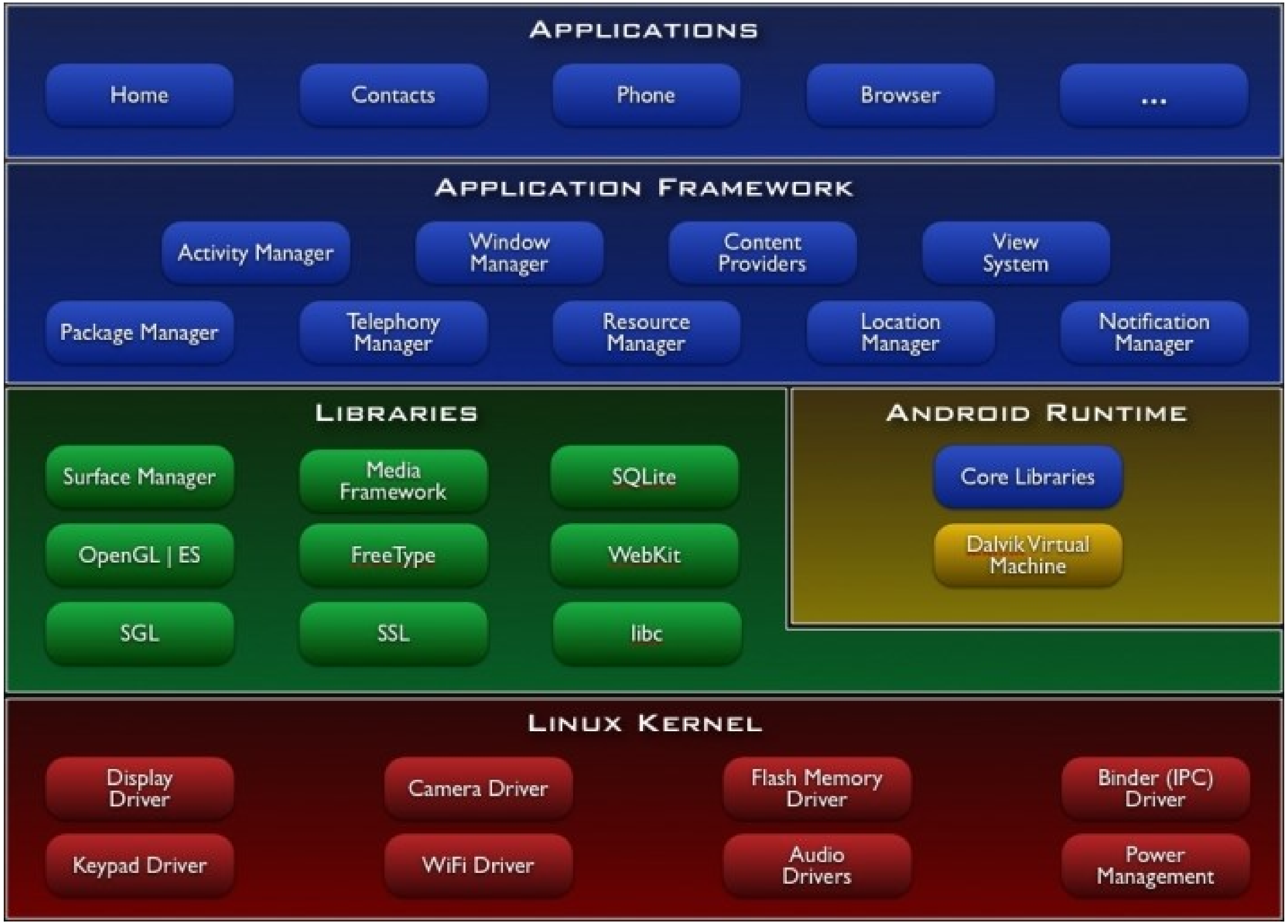
# Android Architecture

Written in Java, executed in Dalvik VM. Home, Contacts, Phone, Browser, …

Written mostly in Java. Managers for Activity, Window, Package, …

Native libraries, daemons and services (C/C++). SQLite, OpenGL, SSL, … Dalvik VM, Core libs

Drivers for hardware, networking, file system access, and inter-process-communication (IPC). Display, camera, flash, Wi-Fi, audio, …




The Linux kernel, the libraries, and the runtime are encapsulated by the Application Framework. Developers typically work with the top two layers

# Android is NOT just “Java on Linux”

- Android uses Linux kernel. Only kernel
  - User land is totally different from usual Linux system
- Android apps are written in Java
  - Class libraries are similar to Java SE but not equal
- Dalvik VM eats only “dex” code
  - Need to translate from Java byte code in advance

# Linux Kernel – Standard Services

Provides generic operating system services

- ❑ Permissions architecture / Security – Restrict access to processes
- ❑ Memory and Process Management
- ❑ Low-level details – File and Network I/O
- ❑ Device Drivers – memory, radio, camera, …

http://www.androidcentral.com/android-z-what-kernel

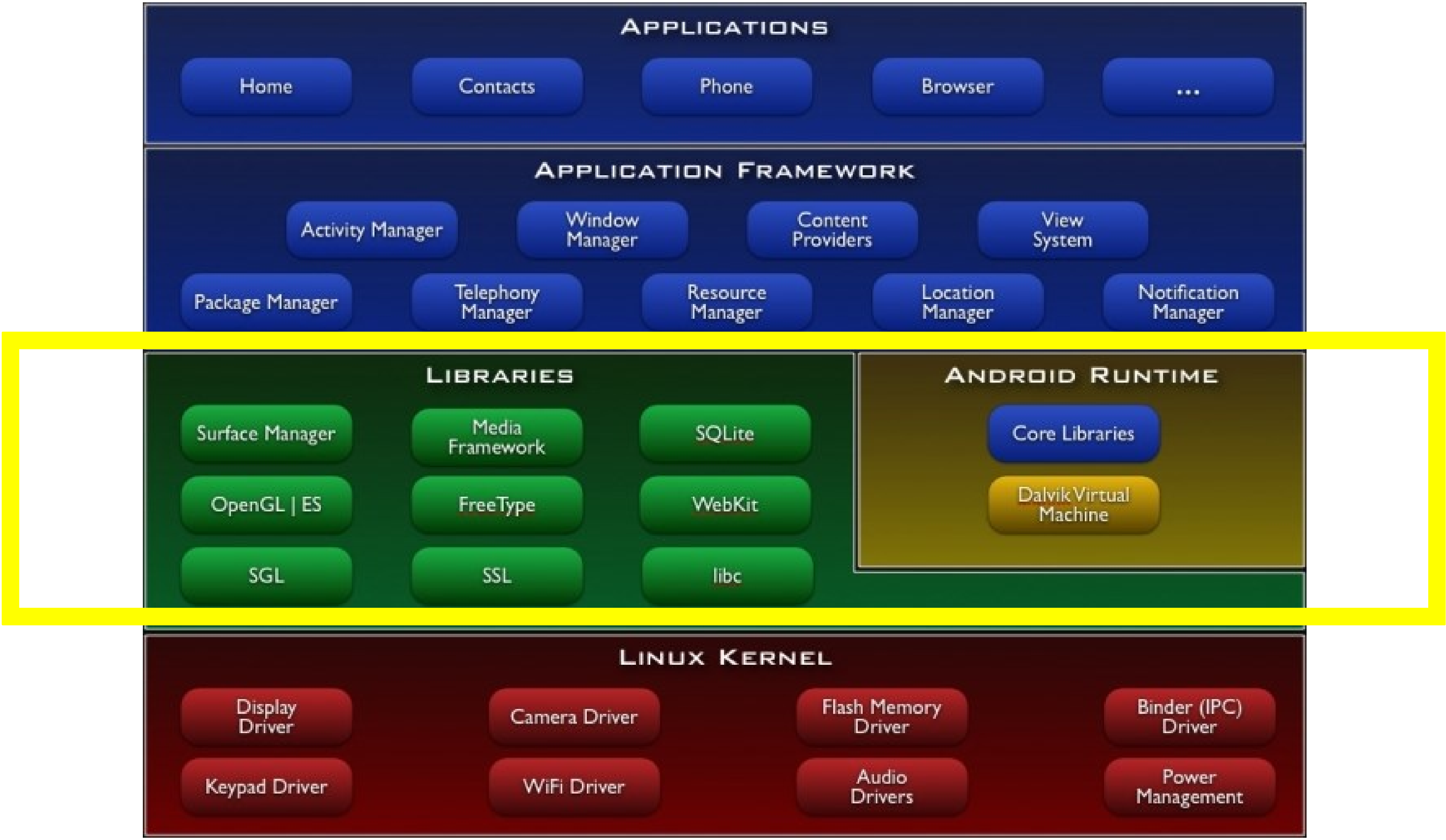
# Linux Kernel – Android Specific Services

For better management of mobile devices

About 249 patches, 25000 lines of code

- Power management – wakelock for early suspend
- Android shared memory
  - ashmem (virtual), pmem – process memory allocator (contiguous)
- Low memory killer – as opposed to Out-Of-Memory killer
- Inter-process communication (IPC) – binder
- System logging facility – logger
- Increase security – paranoid network security

# Native Libraries (C/C++)



# Native Libraries (C/C++)

Handle core performance-sensitive activities (e.g., quickly rendering webpages, updating display)

- System C Library – (Bionic libc) process/thread creation, computation, memory allocation, …
- Surface Manager – display management
- Media Framework – playing audio/video files
- Webkit - rendering / displaying webpages
- OpenGL – high performance graphics
- SQLite – managing in-memory relational databases

# Android Runtime

- Core Java Libraries
- Dalvik Virtual Machine (Dan Bornstein from Google)

# Core Java Libraries

To make it easier to write Java apps, Android provides many reusable Java building blocks / packages

- Basic Java classes – java.* javax.* (basic data structure, file I/O, concurrency mechanisms)
- App lifecycle – android.*
- Internet / Web services – org.*
- Unit Testing – junit.*

# Dalvik Virtual Machine

It is the software that executes Android apps (not the Java VM), specifically designed to run on

- Slow CPU
- Relatively little RAM
- OS without swap space
- Powered by a battery
- Diverse set of devices
- Sandboxed application runtime for security, performance, and reliability

Somewhat conflicting constraints

# Dalvik Virtual Machine

## Google’s Approach to Implement Dalvik VM

- Every Android app runs in its own process with its own instance of Dalvik VM
- Supports a device running multiple VMs efficiently
- Dalvik executable (.dex) format optimized for minimal memory
- Transforms .class files into .dex by “DX” tool
- Register-based VM
- Relies on Linux kernel for threading and low-level memory mgmt

# Dalvik Virtual Machine

## Memory Efficiency

- Total system RAM: 64 MB (can be 100 MB for newer phones)
  - Available RAM after low-level startup: 40 MB
  - Available RAM after high-level services have started: 20 MB

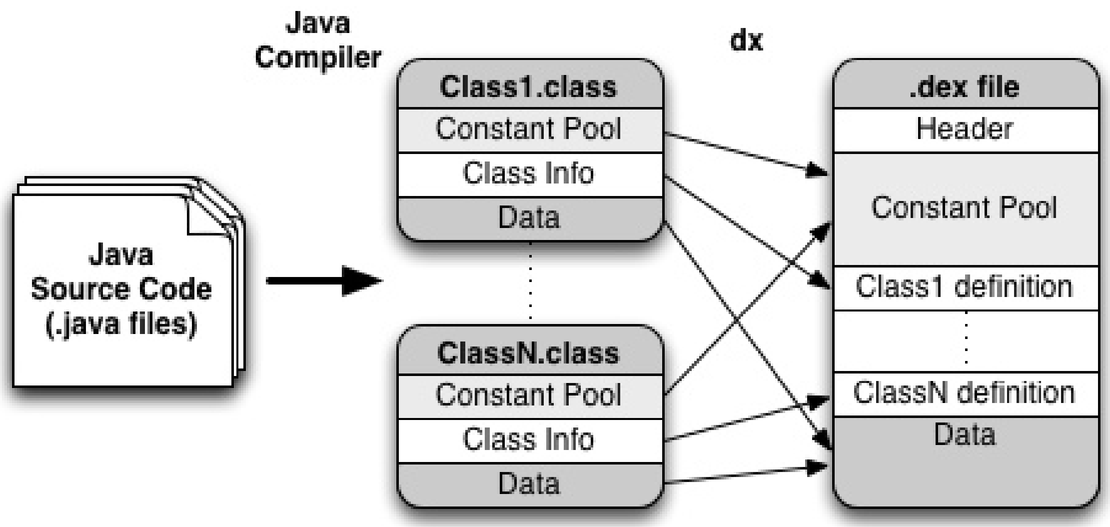
- Multiple independent mutually-suspicious processes
  - Security model: separate address spaces, separate memory

- Large (rich) system library: 10 MB

# Dalvik Virtual Machine

## Typical Workflow

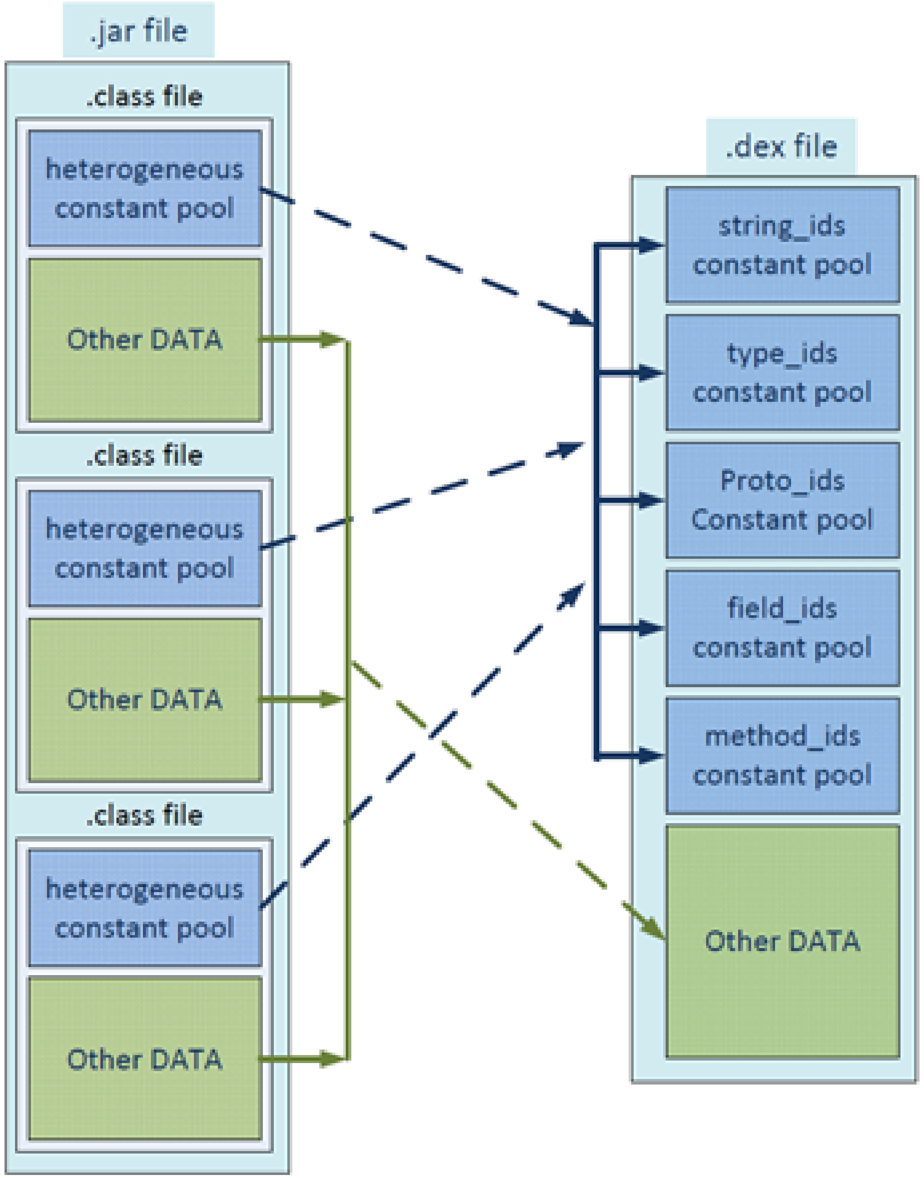
- ❑ Write apps in Java
- ❑ Compile into Java bytecode
  - ○ One .class file per class
- ❑ DX tool converts multiple Java classes into a single DEX file (classes.dex)
  - ○ Rearranges classes, removes redundancy
- ❑ Dex file is packaged with other resources and installed on device



# Dalvik Virtual Machine

## Conserving Memory

- .dex uses shared, type-specific constant pools
  - Minimal repetition and more logical pointers than a .class file
- A constant pool stores all literal constant values within the class
  - String constant, field, variable, class, interface, and method names
- In a .class file, constant part: 60%, method part: 33%



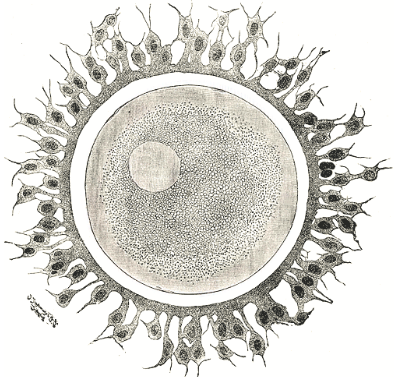
# Dalvik Virtual Machine

.dex cuts the size in half of some common system lib and apps

| Code | Uncompressed JAR (bytes) | Compressed JAR (bytes) | Uncompressed DEX (bytes) |
|---|---|---|---|
| Common System Libraries | 21,445,320 (100%) | 10,662,048 (50%) | 10,311,972 (48%) |
| Web Browser App | 470,312 (100%) | 232,065 (49%) | 209,248 (44%) |
| Alarm Clock App | 119,200 (100%) | 61,658 (52%) | 53,020 (44%) |

- Memory sharing optimizations do not come for free
  - Redesigned garbage collector to keep “mark bits,” indicating that an object is reachable, and therefore, should not be garbage collected

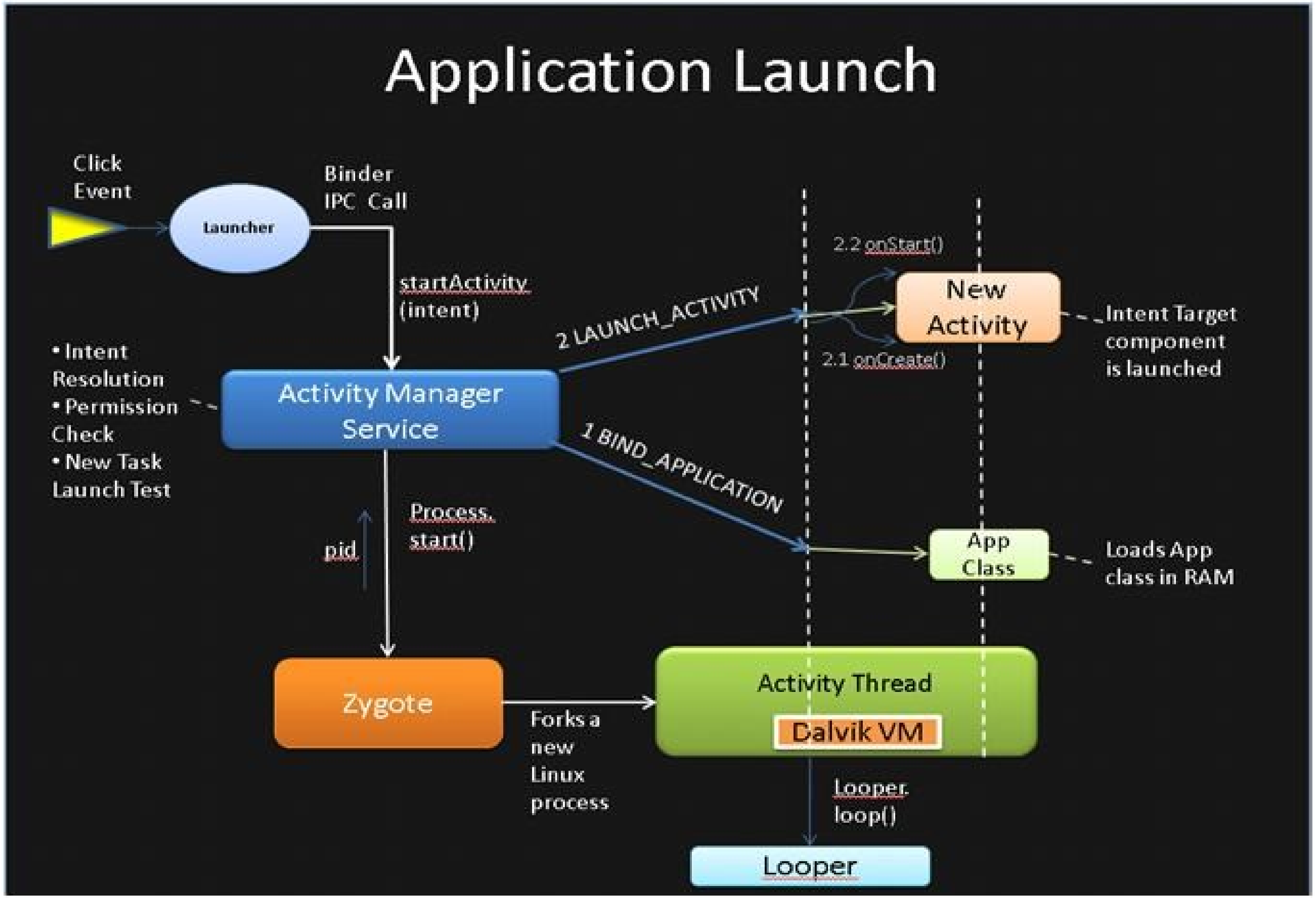
# Dalvik Virtual Machine



The Zygote – Initial cell / earliest dev. stage of an embryo

- Enables sharing of code across VM instances and provides fast startup time for new VM instances
- A VM process that starts at system boot time
  - Initializes Dalvik, which preloads and pre-initializes core library classes (most are read-only)
  - When written, use "copy-on-write" behavior
- Waits for socket requests from runtime processes to fork new VM instances
- In Java VM, each VM instance has an entire copy of the core library class and associated heap objects – memory not shared across instances

# Dalvik Virtual Machine



# Dalvik Virtual Machine

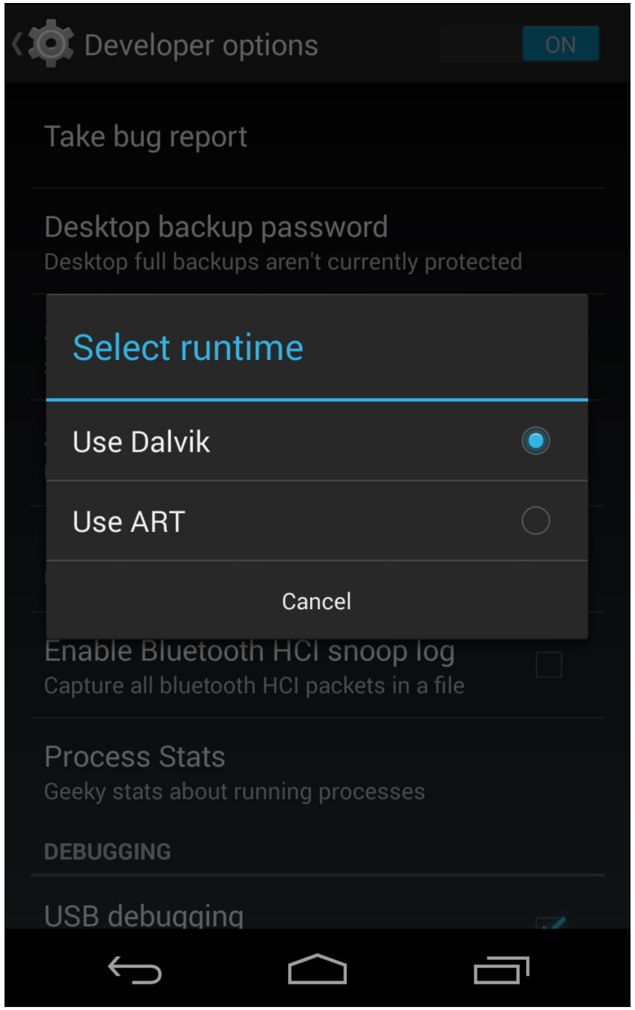
## Register-Based Architecture

- Traditional VMs are stack-based
  - Simplicity of implementation, ease of writing compiler backend
  - But cost of performance
- Registered-based architectures require
  - 47% less executed VM instructions than stack-based
  - 25% larger registry code, but only 1.07% extra real machine loads per VM instruction
  - Overall, 32.3% less time to execute, on average
- Appropriate for resource-constrained devices
  - 25% more code, but 50% reduction in code-size through shared constant pools in .dex file

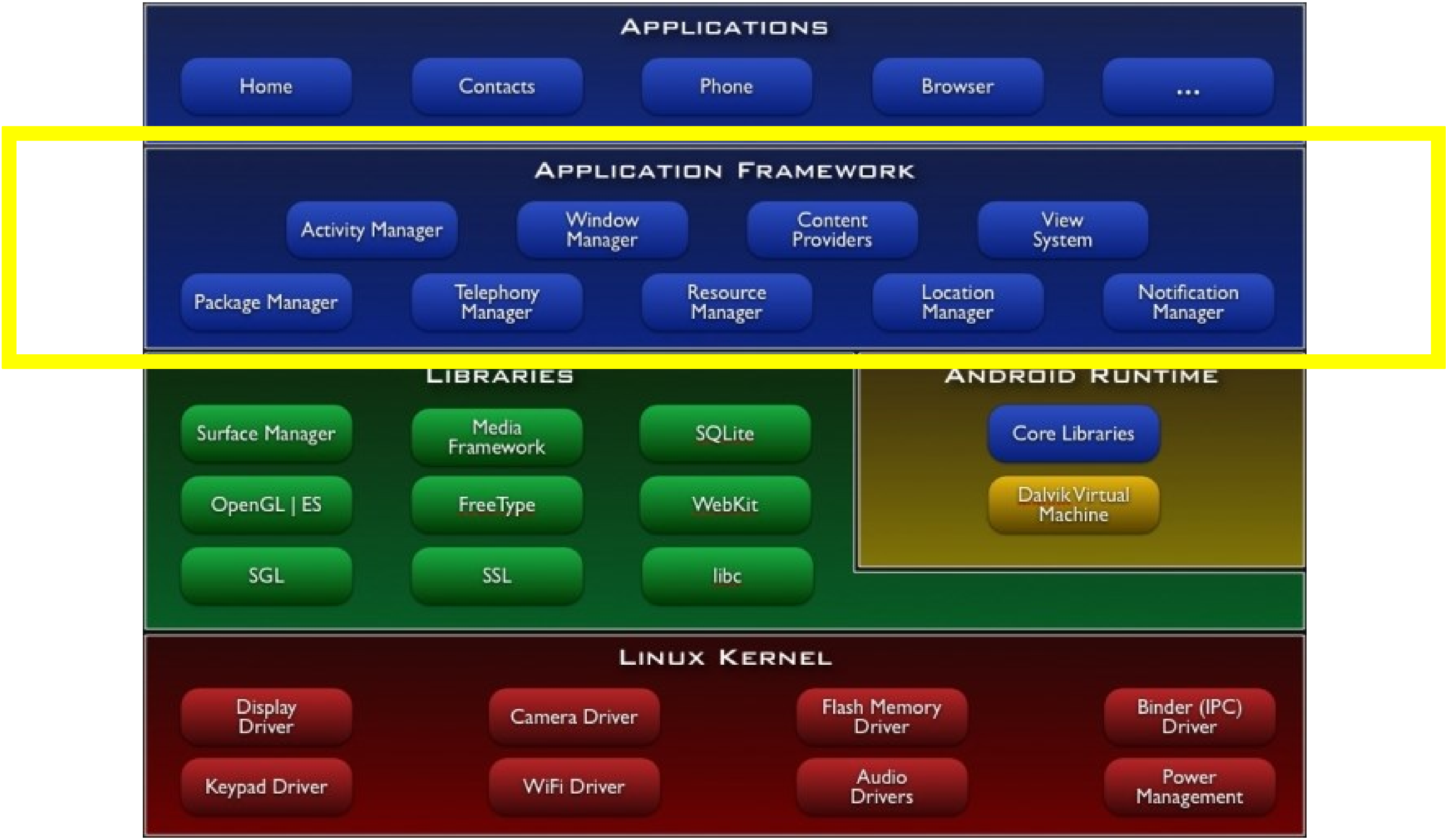
# Android ART

Android Run Time : Google finally moves to replace Dalvik, to boost performance and battery life. Early version included in Android KitKat

- ART straddles a middle-ground between compiled and interpreted code, called “ahead-of-time” (AOT) compilation
- Currently apps are interpreted at runtime using JIT (slow), compare with iOS
- With ART, app is compiled into native code while installing (fast)



# Application Framework



# Application Framework

A collection of reusable software components that many mobile apps will need

- Package Manager – a database tracking all apps installed
  - Allows one app to find/contact another and share data
- Window Manager
  - Manages the windows comprising the app
- View System – provides common UI elements
  - tabs, icons, text entry boxes, buttons
- Resource Manager – manages non-compiled resources
  - strings, graphics, layout files (choice of languages)

# Application Framework

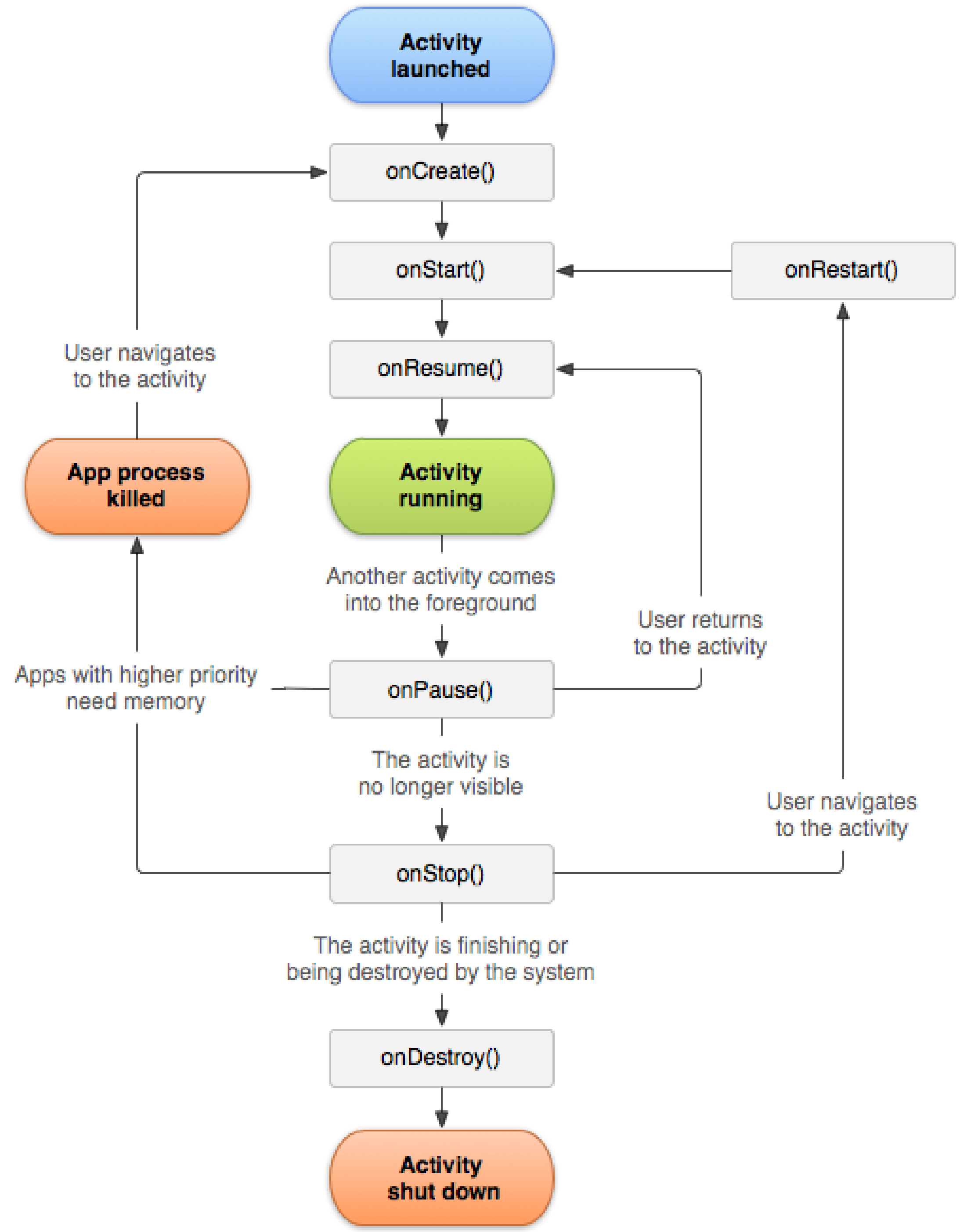
- Activity Manager – coordinates and supports navigation across multiple UI screens
  - Playing music
- Content Provider – databases allowing multiple apps to store and share structured information
  - Phone app accesses Contacts app to dial phone numbers
- Location Manager – allows apps to receive location and movement information
  - GPS to do context-specific tasks, e.g., finding directions
- Notification Manager – place notification information in the status bar when important events occur
  - MMS received while emailing / calling

# App Lifecycle

Lifecycle is a set of states

- ❑ When the current state changes, Android OS notifies the Activity of that change

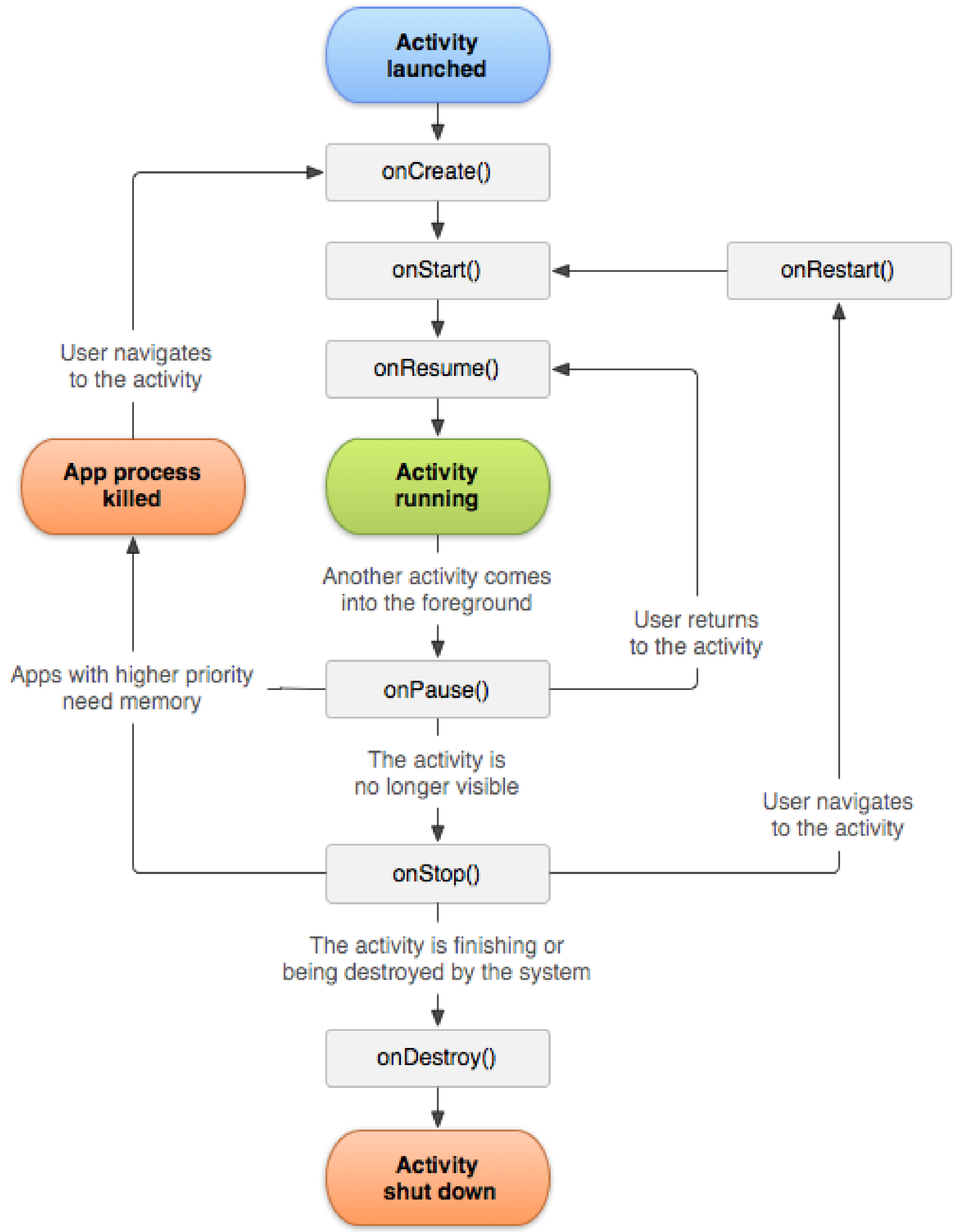
- ❑ Implemented by callback methods



# App Lifecycle

## Four States

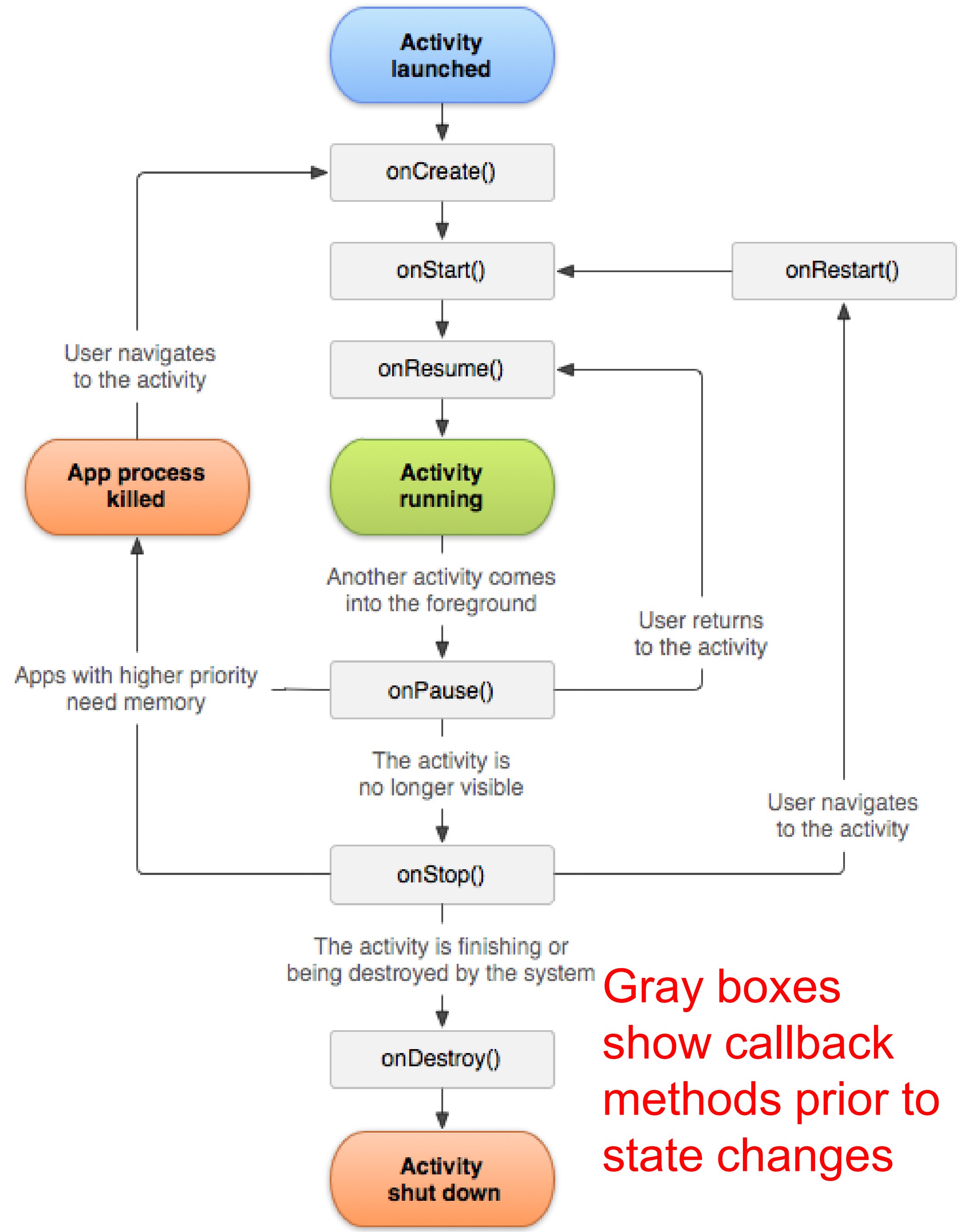
- ❑ Active / Running
  - Visible, has focus, and in foreground
- ❑ Paused
  - Partially visible but not active and lost focus
  - Completely alive and maintains its state
- ❑ Stopped
  - Completely obscured by another activity
- ❑ Destroyed / Dead
  - No longer in memory



# App Lifecycle

## Seven Callback Methods

- ❑ onCreate() – UI creation and initialization of data elements
- ❑ onStart() – called before Activity is visible (but not alive)
- ❑ onResume() – Activity becomes visible and active for user to interact
- ❑ onPause() – another Activity comes in front, or user navigates away

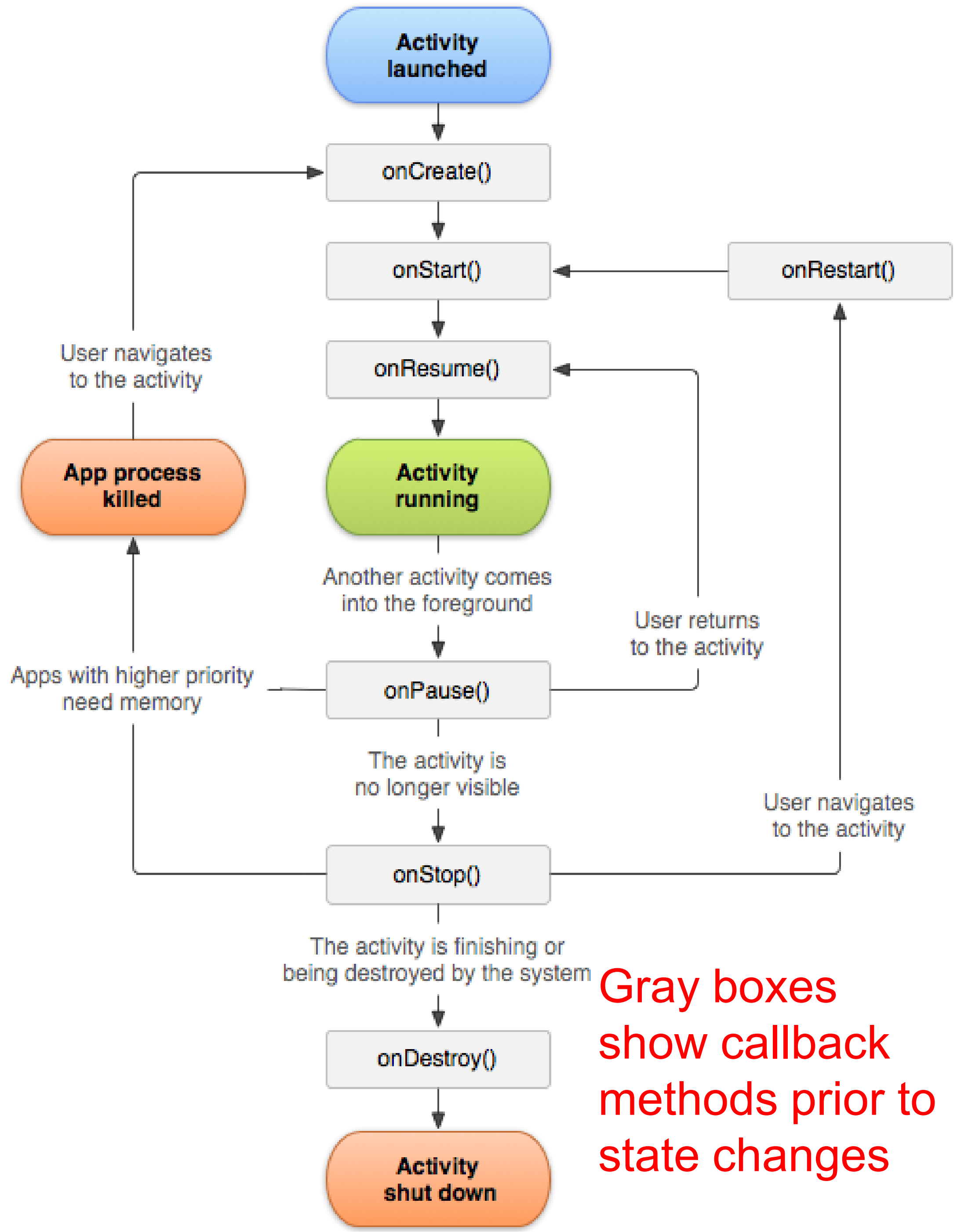



Gray boxes show callback methods prior to state changes

# App Lifecycle

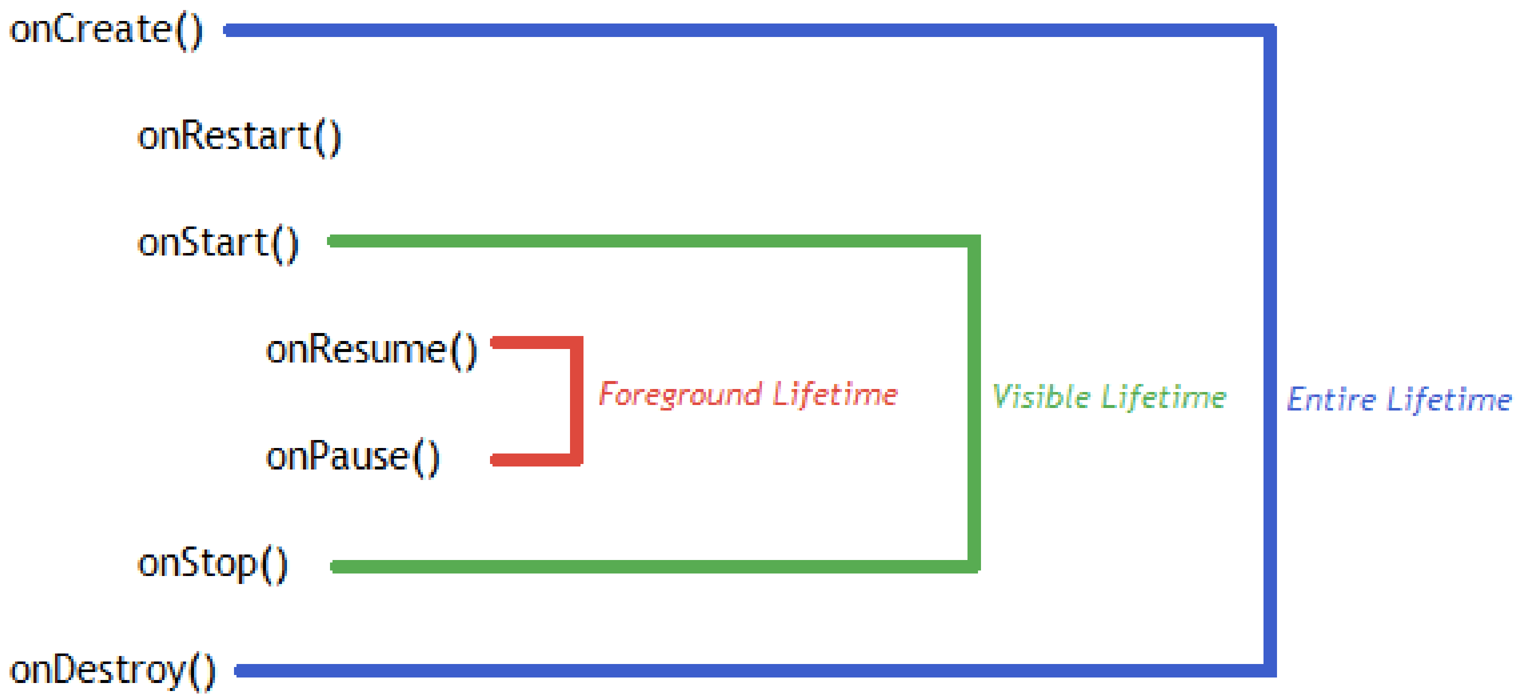
## Seven Callback Methods

- ❑ onStop() – back button, or new Activity completely covers
- ❑ onRestart() – user navigates back to the Activity
- ❑ onDestroy() – Activity is destroyed




Gray boxes show callback methods prior to state changes

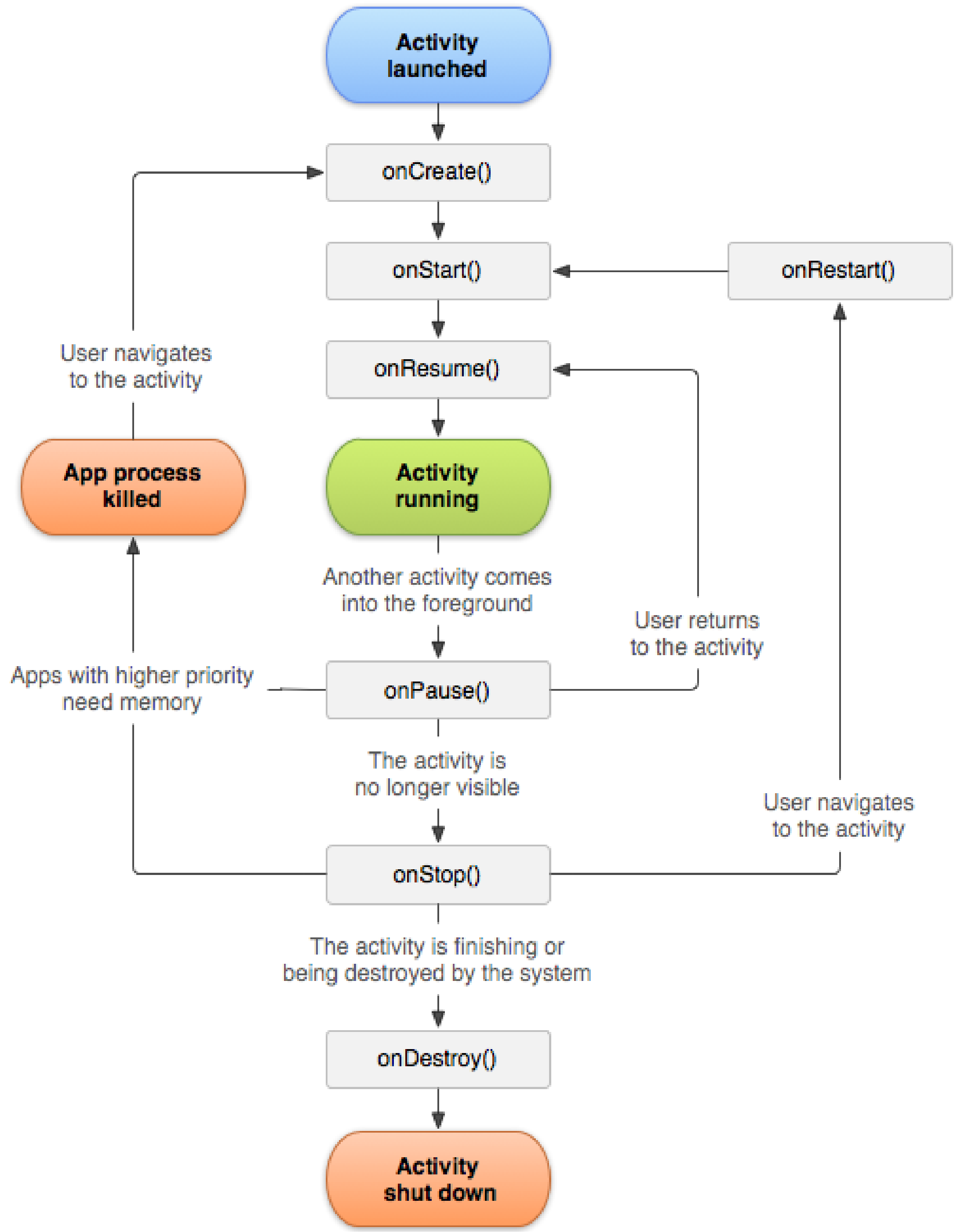
# App Lifecycle



# App Lifecycle

Three Lifecycle loops for every Activity, defined by callback methods

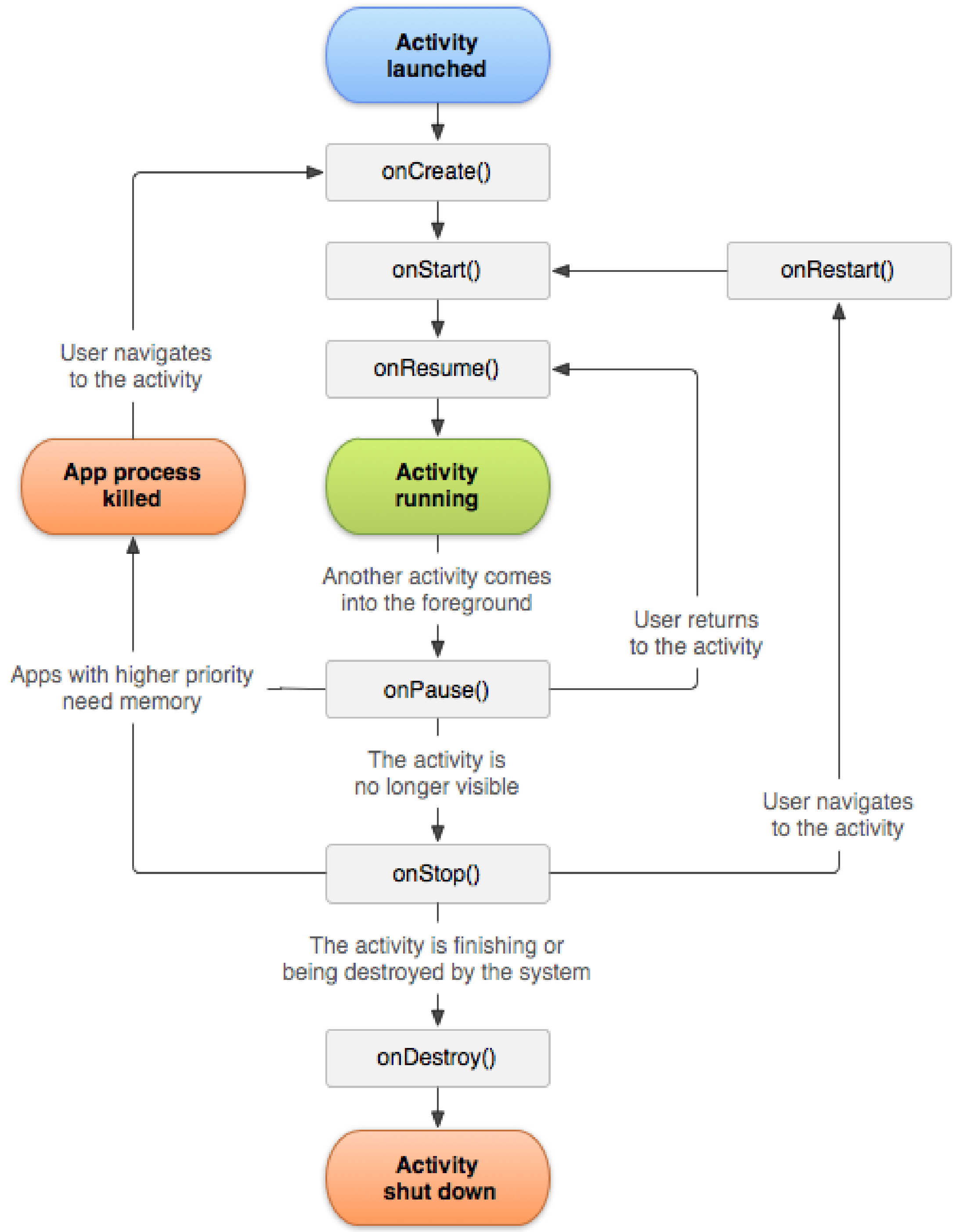
- ❑ Entire Lifetime – first call to onCreate() and final call to onDestroy()
- ❑ Visible Lifetime – from onStart() and onStop()
- ❑ Foreground Lifetime – from onResume() to onPause



# App Lifecycle

## Saving Persistent State

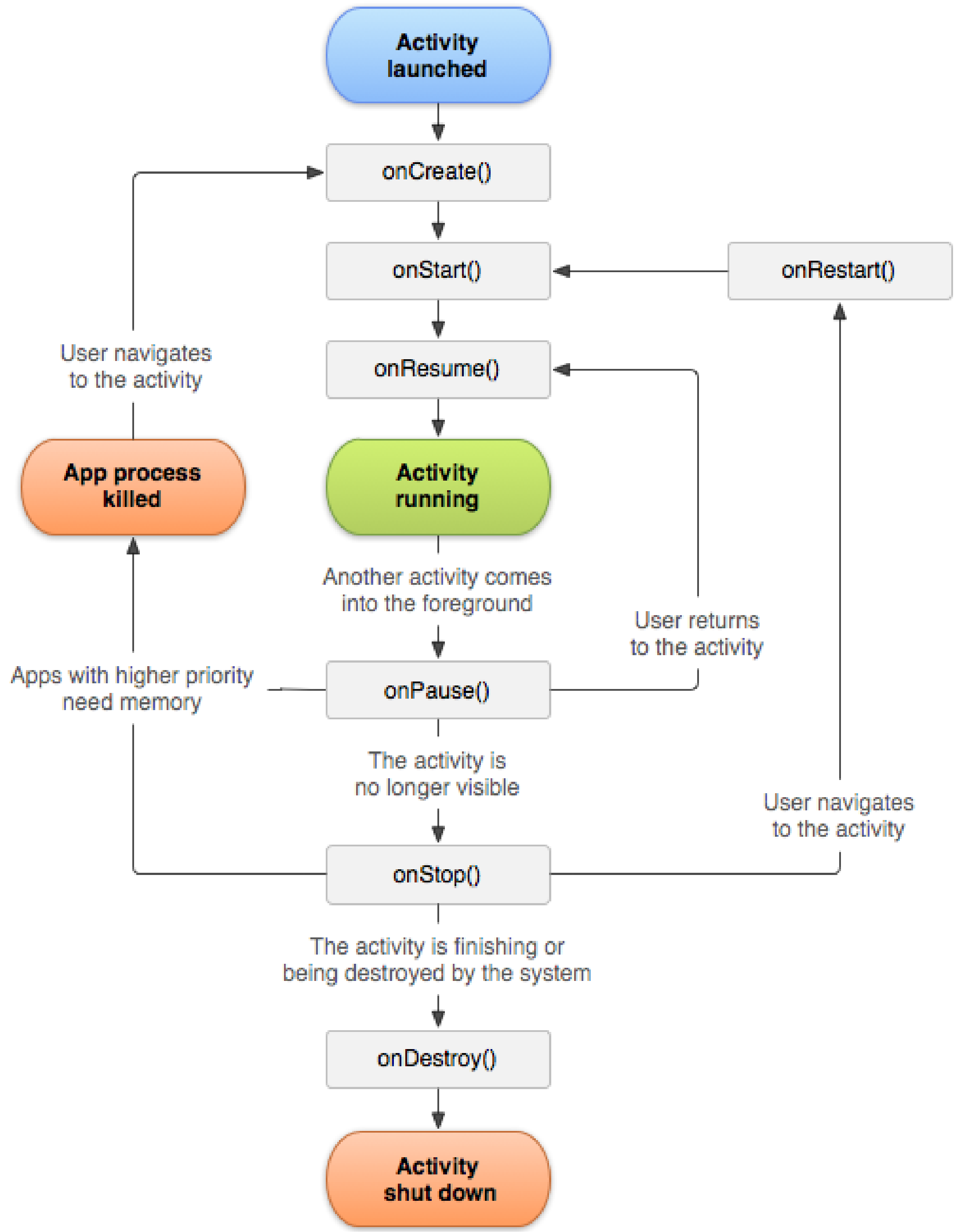
- When an Activity is stopped or paused, its state is preserved
- When an Activity is destroyed by the system, it is recreated next time Activity starts
- User is often unaware that an Activity is destroyed, resulting in surprises and crashes



# App Lifecycle

## Two Kinds of Persistent States

- ❑ Shared document-like data
  - SQLite storage using a content provider
  - “Edit-in-Place” user model
  - Backup fully at onPause()
- ❑ Internal state (user prefs)
  - API calls to store prefs
  - E.g., user’s initial calendar display (day vs week view), or default webpage in a browser



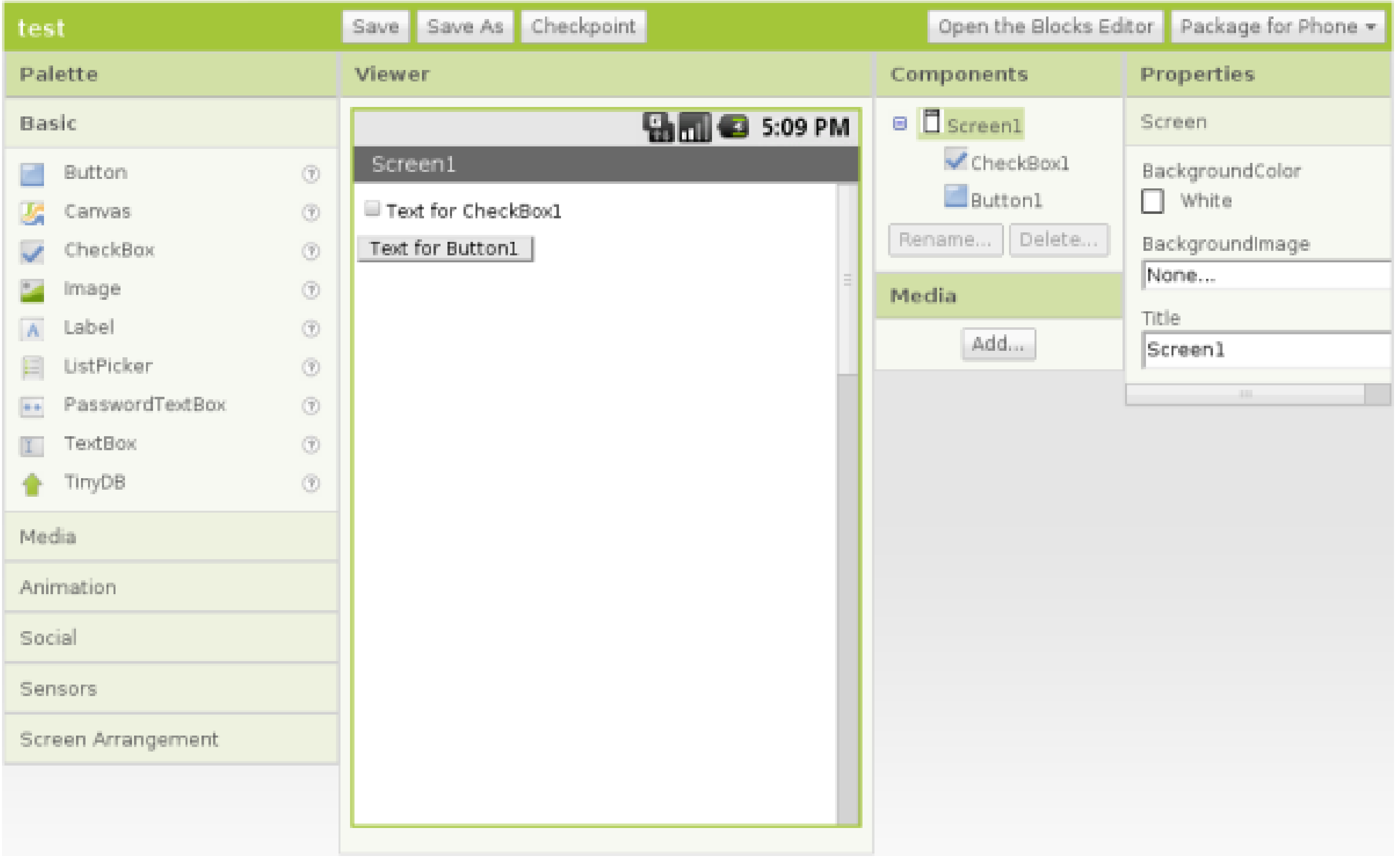
# Applications

- Standard apps include
  - Home – main screen
  - Contacts – contacts database
  - Phone – dial phone numbers
  - Browser – view web pages
  - Email Reader – compose and read email messages

- Nothing special about these apps
  - You can substitute your own or 3$^{rd}$ party app for any of them

# Android App Inventor

- A produce of Google Labs
  http://appinventor.mit.edu/explore/
- A web based graphical cloud-based tool for rapid development



# Android Development Environment

- Workbench for writing Android applications
- Android SDK (ADT) bundle
- Eclipse IDE
- Android Emulator
- Eclipse debugger
- Other tools

- Prerequisites
  - JDK6 installed (its not the latest version of Java)

# Android Emulator

- Pros
  - Doesn’t require an actual phone
  - Hardware is reconfigurable (memory, display size)
  - Changes are non-destructive
- Cons
  - Can be very slow
  - Some features unavailable (no Bluetooth or USB connections)
  - Performance / user experience can be misleading

# Getting Started

- Download and install the Android Developer Tools (ADT) Bundle

- http://developer.android.com/sdk (or, Android Studio)
  - Latest Android platform
  - Eclipse + ADT plugin
  - Latest system image for emulator – runs Android virtual devices (ADV)
  - Additional development tools

# Fun with Math

$$S = 1 + 2 + 3 + 4 + \ldots ?$$

a) Infinity
b) Does not converge (diverges)
c) A finite value
d) A Googolplex $10^{(10)^\wedge 100}$
e) Confused

$S = -1/12$ Is it Absurd?

# Fun with Math

Okay, let's define two more series:

$$S_1 = 1 - 1 + 1 - 1 + \ldots. ?$$

The sum depends on where we stop the series

1. 1, if we stop at odd location
2. 0, if we stop at even location

So,

$$S_1 = \frac{1}{2}$$

# Fun with Math

Okay, let's define two more series:

$$S_2 = 1 - 2 + 3 - 4 + \ldots. ?$$

Now add S2 to itself, but by shifting. So

$$\begin{aligned} 2S_2 &= 1 - 2 + 3 - 4 + \ldots. \\ &\phantom{= 1 - 2 +{}} 1 - 2 + 3 - 4 + \ldots \\ &= 1 - 1 + 1 - 1 + 1 - \ldots \\ &= S_1 = ½ \end{aligned}$$

Therefore,
$S_2 = 1/4$

# Fun with Math

Now, subtract S2 from S (the original sum)

$$S = 1 + 2 + 3 + 4 + \ldots.$$

(minus)

$$S_2 = 1 - 2 + 3 - 4 + \ldots.$$

$$S - S_2 = 0 + 4 + 0 + 8 + 0 + 12 + \ldots$$

$$= 4(1 + 2 + 3 + \ldots)$$

$$= 4S$$

$$3S = -S_2$$

$$= -1/4$$

$$\Rightarrow S = -1/12$$

Analytic continuation of the Riemann-Zeta function

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

# Useful Links

- http://www.hongkiat.com/blog/android-evolution/
- http://www.nfcworld.com/2013/10/31/326619/google-gets-around-carriers-host-card-emulation-nfc-payments/
- http://en.wikipedia.org/wiki/Android_version_history
- http://www.android-app-market.com/android-activity-lifecycle.html
- http://www.extremetech.com/computing/170677-android-art-google-finally-moves-to-replace-dalvik-to-boost-performance-and-battery-life
- http://www.extremetech.com/mobile/170034-android-4-4-demystified-the-most-significant-android-update-in-years
- http://www.youtube.com/watch?v=ptjedOZEXPM
- https://sites.google.com/site/io/inside-the-android-application-framework