Android Architecture

Karthik Dantu and Steve Ko

Administrivia

- Show us your build!
- Assignment 2 is out and due in one week.

Project Meetings: This Friday

- Multi-Interface Connectivity: Aditya, Chaitanya, Devanshu, Swapnesh
- Location Services API: AnujG, Subhendu, Sriram, Gokhan
- Smart Services: Ankesh, Varun, Arshaq, AnujK
- Automated cloud backup: Kyle, Rakesh, Sharath, Hrishikesh
- Quantified Self: Yash, Pratik, Tanmay

Project Meetings: Next Friday

- Analyzing memory pressure in Android: Alex, Ankit, Brandon, Engin
- Wakelock resource accounting: Rajesh, Kshitijkumar, Aravindhan, Babu Prasad
- RTAndroid: Pranav, Jerry, Aswin, Mohit
- Google Glass Visual Inertial Navigation: Srivathsa, Nishanth, Rohit, Priyanka

Today: Overview of Android

- Why?
  - A good example to start with in order to understand the source (I think)
  - Getting deeper with the source
  - Not quite straightforward to understand
- Goal: giving you enough pointers so you can navigate the source yourself.

Things You Need to Know

- Android Programming Model
  - Launcher is implemented with this model.
- Android Architecture
- Android IPC mechanisms
  - ActivityManager uses IPC mechanisms.
- Zygote

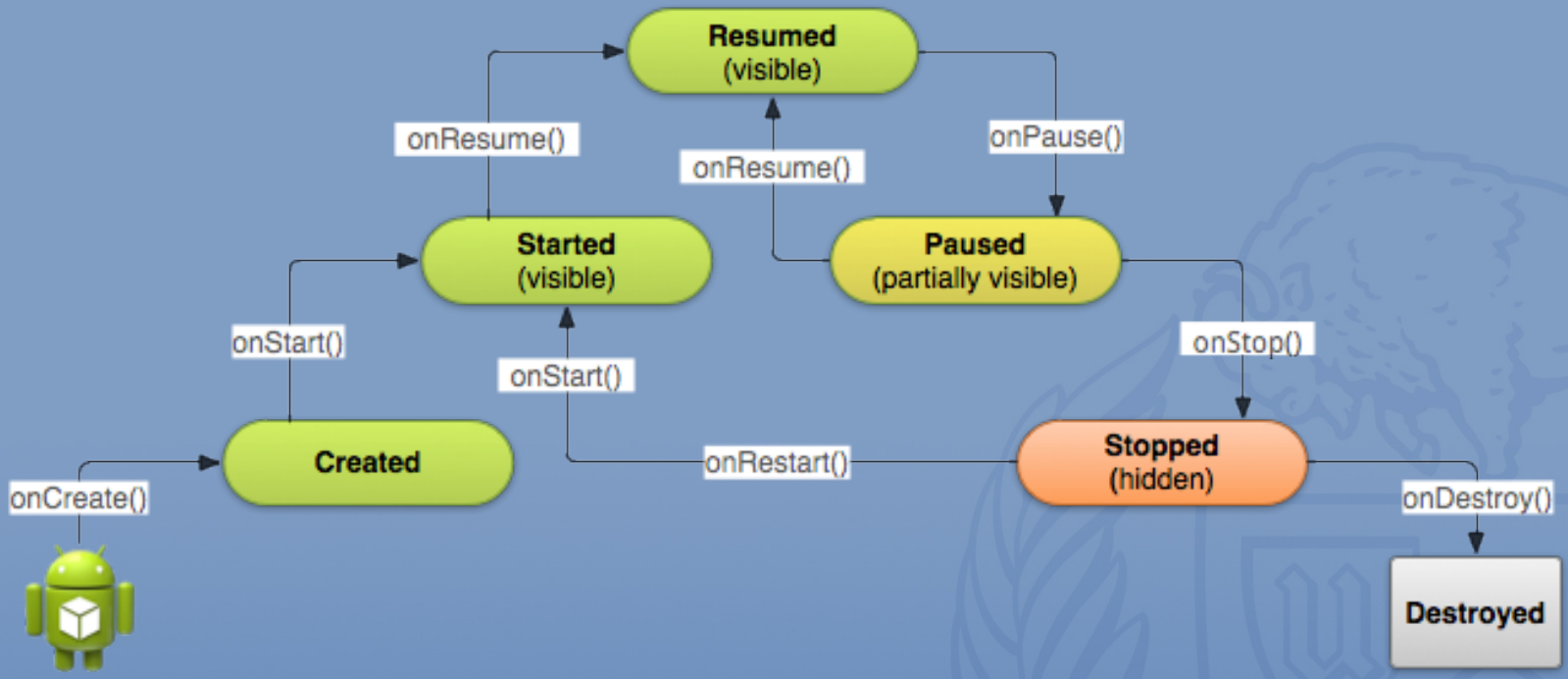
Android Programming Model

- No main()
- Four main components: Activity, Service, ContentProvider, BroadcastReceiver
  - You need to implement at least one of them to write an Android app.
- Event-driven

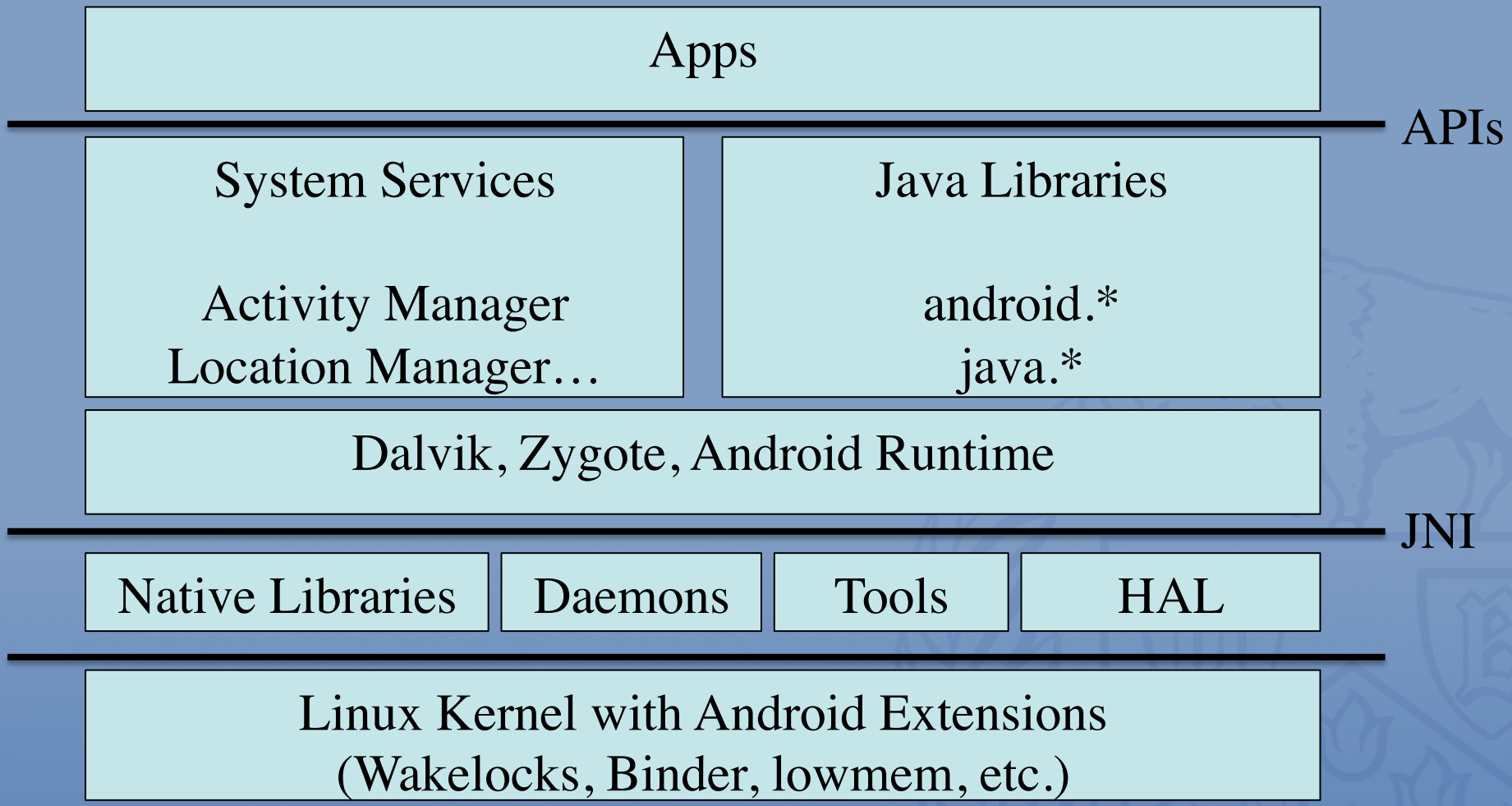
Example - Activity

```
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

Example - Activity



Android Architecture



Zygote

- C++ (e.g., app_main.cpp, AndroidRuntime.cpp) and Java (e,g., ZygoteInit.java)
- Starts at boot.
- ZygoteInit.java manages a domain socket and listens to incoming VM creation requests.
  - This uses ZygoteConnection.java.
- Main flow
  - Forks a new VM instance.
  - Loads the class that has the process's main().
  - Calls the main().

Android IPC Mechanisms

- Android relies heavily on IPC mechanisms.
  - This goes with the event-driven programming model.
- Three main mechanisms
  - Intent
  - Binder
  - Looper-Handler

Intent

- You can think of it as a message or a command.
- Main fields
  - Action (e.g., ACTION_VIEW) and Data
- Many system events are communicated as intents.
  - E.g., ACTION_BATTERY_CHANGED, ACTION_POWER_CONNECTED, etc.
- A reasonable strategy for code navigation
  - Locate where the switch-case code is for different actions

Binder

- The main IPC mechanism on Android
- Binder enables method calls across process boundaries.
  - Caller side: A proxy and marshalling code
  - Callee side: A stub and unmarshalling code
- Two ways to use it.
  - Automatic proxy & stub generation (.aidl)
  - Manual proxy & stub implementation

Binder with .aidl

- .aidl defines the interface.
  - Naming convention: I*.aidl.
  - E.g., IPackageManager.aidl
- The stub compiler generates I*.java
  - E.g., IPackageManager.java
  - This is part of the build process, i.e., you will not find I*.java file in the source (unless you've compiled already, then it's under out/).
- It contains I*.Stub abstract class
  - E.g., abstract class IPackageManager.Stub

Binder with .aidl

- A Stub class should be extended.
  - This is the actual implementation for IPC calls.
  - E.g., class PackageManagerService extends IPackageManager.Stub
- Callers can use the interface to make IPC calls.
  - Callers can import classes in I*.java and use Stub.asInterface() when making IPC calls.
  - E.g., IPackageManager.Stub.asInterface() returns an object for making IPC calls.

Binder with .aidl

- A reasonable strategy for code navigation
  - If you encounter a call using an object returned from .asInterface() call, it's a Binder call.
  - Don't worry about marshalling/unmarshalling code, e.g., onTransact().
  - Find the class that extends the Stub class. (Use croot;jgrep)

Binder without .aidl

- Manual implementation of the interface, marshalling/unmarshalling, and methods.
- E.g., IActivityManager.java defines the interface for accessing ActivityManager.
- abstract class ActivityManagerNative implements IActivityManager and has the marshalling/ unmarshalling code.
- There's a class that extends ActivityManagerNative and provide the actual callee-side implementation.

Binder without .aidl

- A reasonable strategy for code navigation
  - Manual implementation typically follows the ActivityManager example.
  - Interface file → abstract class → extended class
- E.g., startActivity() goes through this flow of classes.

Looper-Handler

- Looper is a per-thread message loop.
  - Looper.prepare()
  - Looper.loop()
- A Handler is shared by two threads to send/receive messages.
- Looper-Hanlder is used in the app control process to handle various messages.

Code Flow for App Start

- Launcher sends an intent to start an activity.
  - startActivity() is a Binder call to ActivityManager.
- ActivityManager sends a process fork request.
  - This request uses a socket to Zygote.
- Zygote forks a new VM instance that loads ActivityThread.
  - ActivityThread has the real main() for an app.

Code Flow for App Start

- ActivityThread calls the app's onCreate()
- ActivityThread notifies ActivityManager.
- ActivityManager makes a Binder call to ActivityThread to start the app (i.e., call onStart()).

Code Navigation

- Let's see the code!