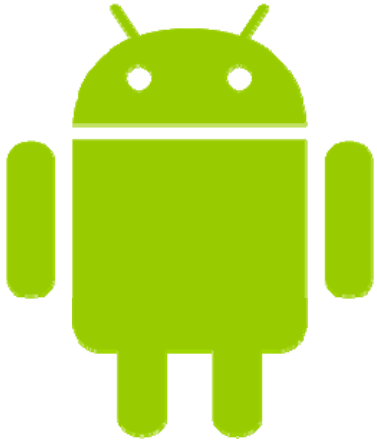
ITP 442

Android Overview



Rob Parke

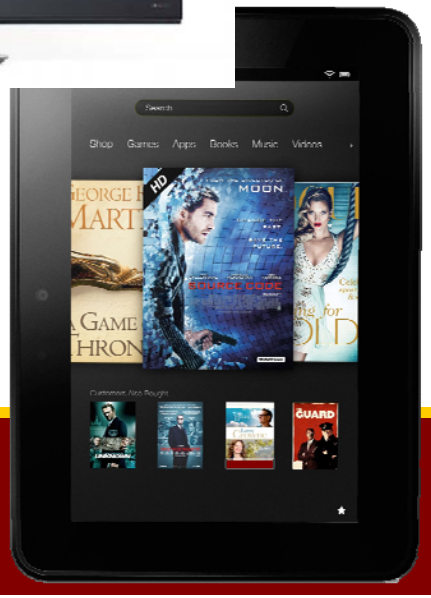
- Email: parke@usc.edu

- ITP Lecturer
  - Android
  - Java
  - Python

USC Viterbi
Information Technology Program

What is Android?

- Write once, run everywhere



What is Android?

- Built on a modified version of Linux
- Open source
  - Android Open-Source Project (AOSP)
  - Free to download source code
  - Modify / repackage code without releasing new code to the open-source community
  - Neither developers nor device manufacturers pay royalties or license fees

What is Android?

- Built on a modified version of Linux
- Open source
  - Android Open-Source Project (AOSP)
  - Free to download source code
  - Modify / repackage code without releasing new code to the open-source community
  - Neither developers nor device manufacturers pay royalties or license fees

What is Android?

- Free to develop
  - Freely available SDK
  - Programmed in Java
  - IDE: Android Studio (based on IntelliJ IDE)
    - Can use Eclipse, but no longer supported by Google
  - Develop with Win, Mac, or Linux

What is Android?

- Third-party apps on the same level as native apps
  - No distinction between native and third-party apps
    - Unlike iOS
  - All apps have access to the same APIs
  - Unprecedented access to the underlying hardware
  - Ability to extend or replace existing applications

What is Android?

- Almost free to publish
  - No testing and certification programs required
  - One-time $25 fee to distribute using the Google Play store

What is Android?

- "Free Market" to distribute
  - Free to choose the revenue model
  - Free to create applications for any size demographic
  - Free to choose distribution methods
    - Google Play store
    - Amazon Appstore
    - Other third-party stores
    - Own distribution methods
  - Free to choose payment mechanisms

QUICK HISTORY

Android

- 2003 – Android, Inc.
  - Created by Andy Rubin
  - Initially started to developed digital camera OS
- 2005 – Google acquires Android
  - Leverage open-source software / community
  - Create platform that will run on different devices
  - Eliminate market fragmentation
- Open-Handset Alliance (OHA)
  - Association of development company, manufacturers, cellular providers
  - Google provides source code, documentation, tools, etc.

Android

- Google's Initial Plan
  - Release Android on "Blackberry-like" device without touch interface
  - **until…**

Android

- Dec 2006 – iPhone announcement

  - No keyboard
  - Touch interface
  - Google rethinks Android

Android

- Oct 2008 – T-Mobile G1

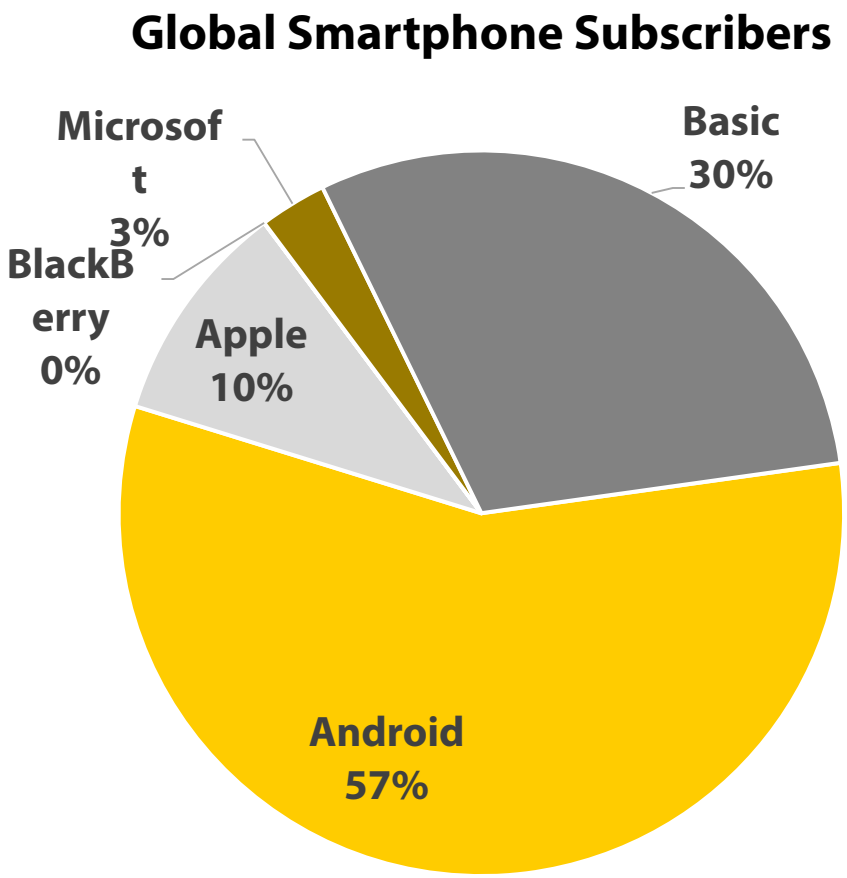
Android Versions

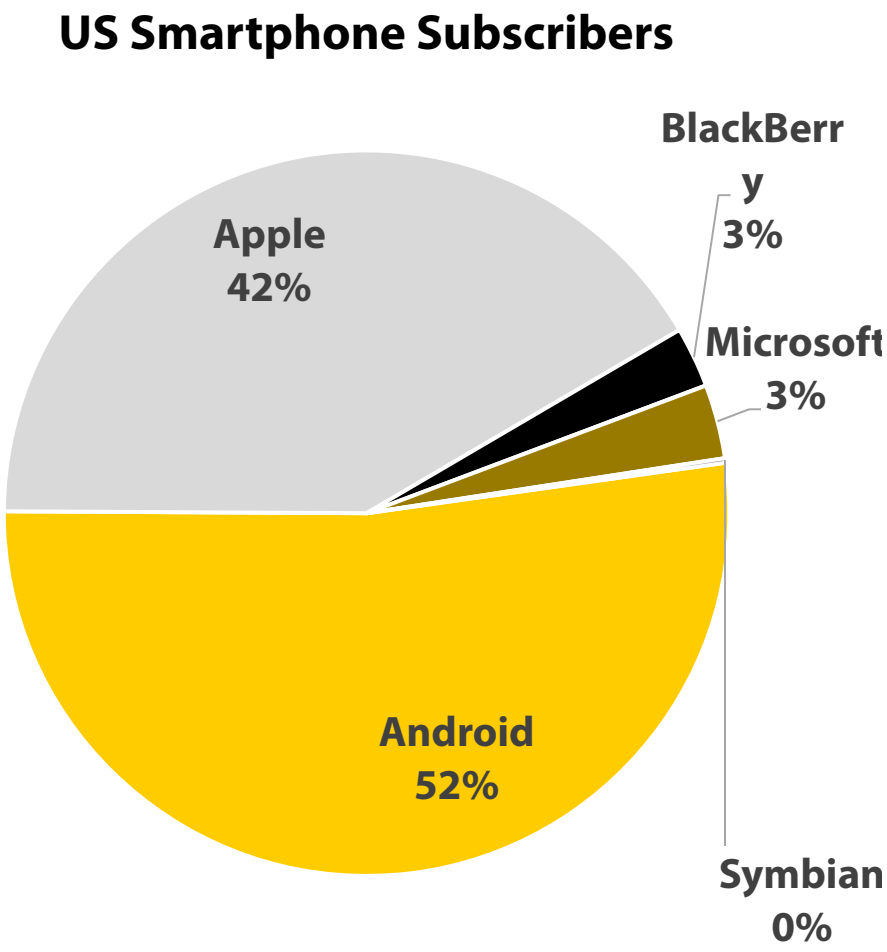
Version	Codename	API	Date
1.5	Cupcake	3	April 30 2009
1.6	Donut	4	Sep 15 2009
2.1	Eclair	7	Oct 26 2009
2.2	Froyo	8	May 20 2010
2.3 – 2.3.7	Gingerbread	9-10	Dec 6 2010
3.1 – 3.2	Honeycomb	12-13	Feb 22 2011
4.0.3 – 4.0.4	Ice Cream Sandwich	15	Nov 14 2011
4.1 – 4.3	Jelly Bean	16-18	July 24 2013
4.4 – 4.4.3	Kit Kat	19	June 2 2014
5.0	Lollipop	20+	Nov 3 2014

Why you should care (and why you probably don't)

- Android
  - Much larger user base
- iPhone
  - Slightly younger
  - More affluent / likely to use phone for purchasing

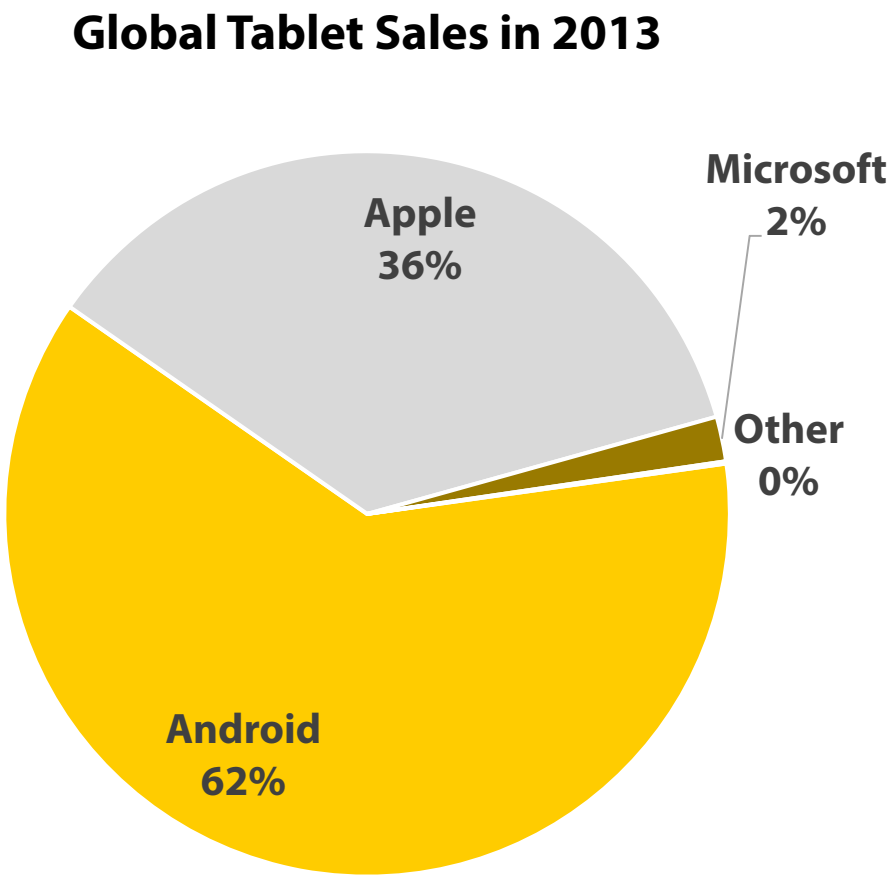
Smartphone Market Share





March 2014

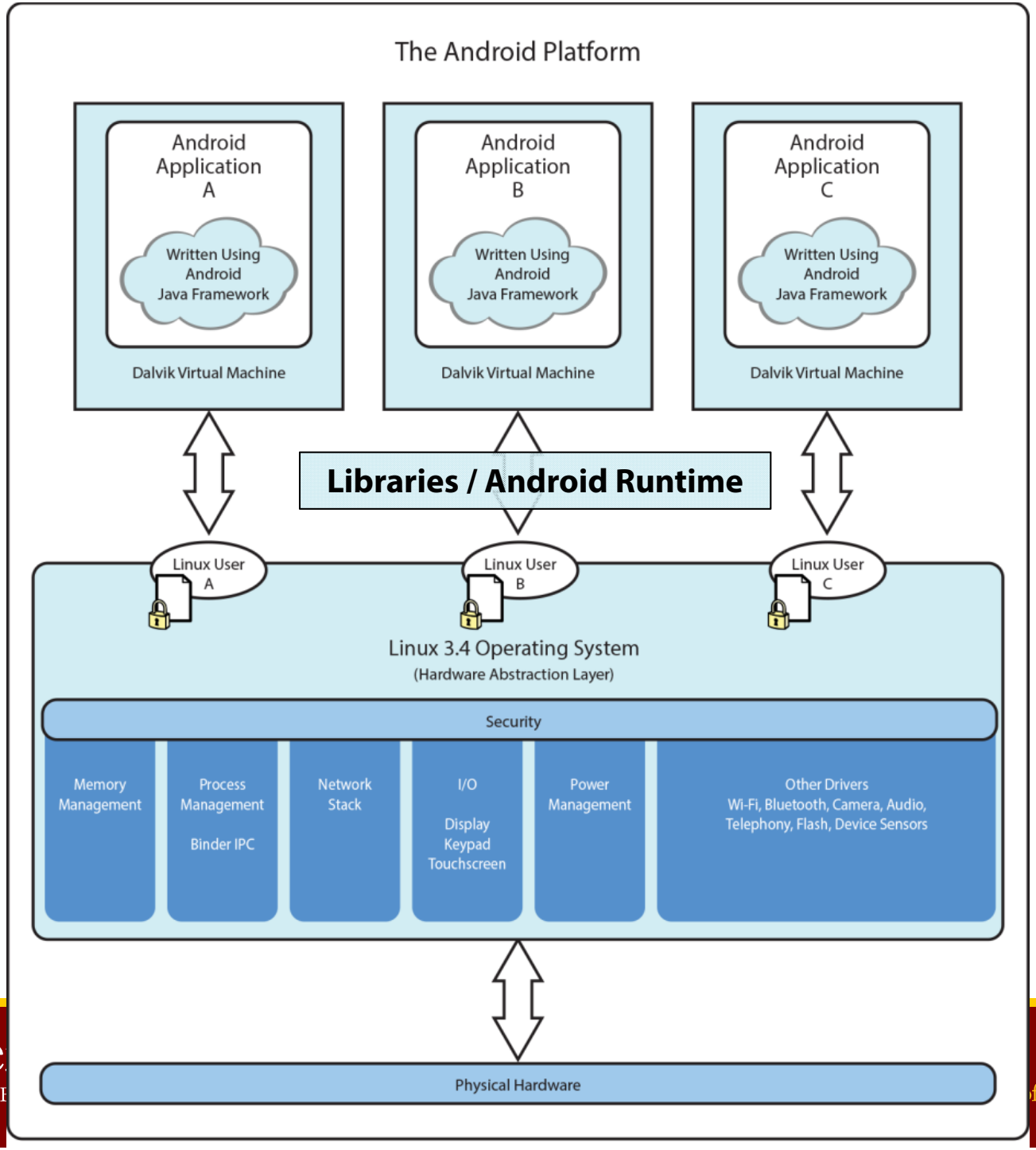
Tablet Market Share



ARCHITECTURE

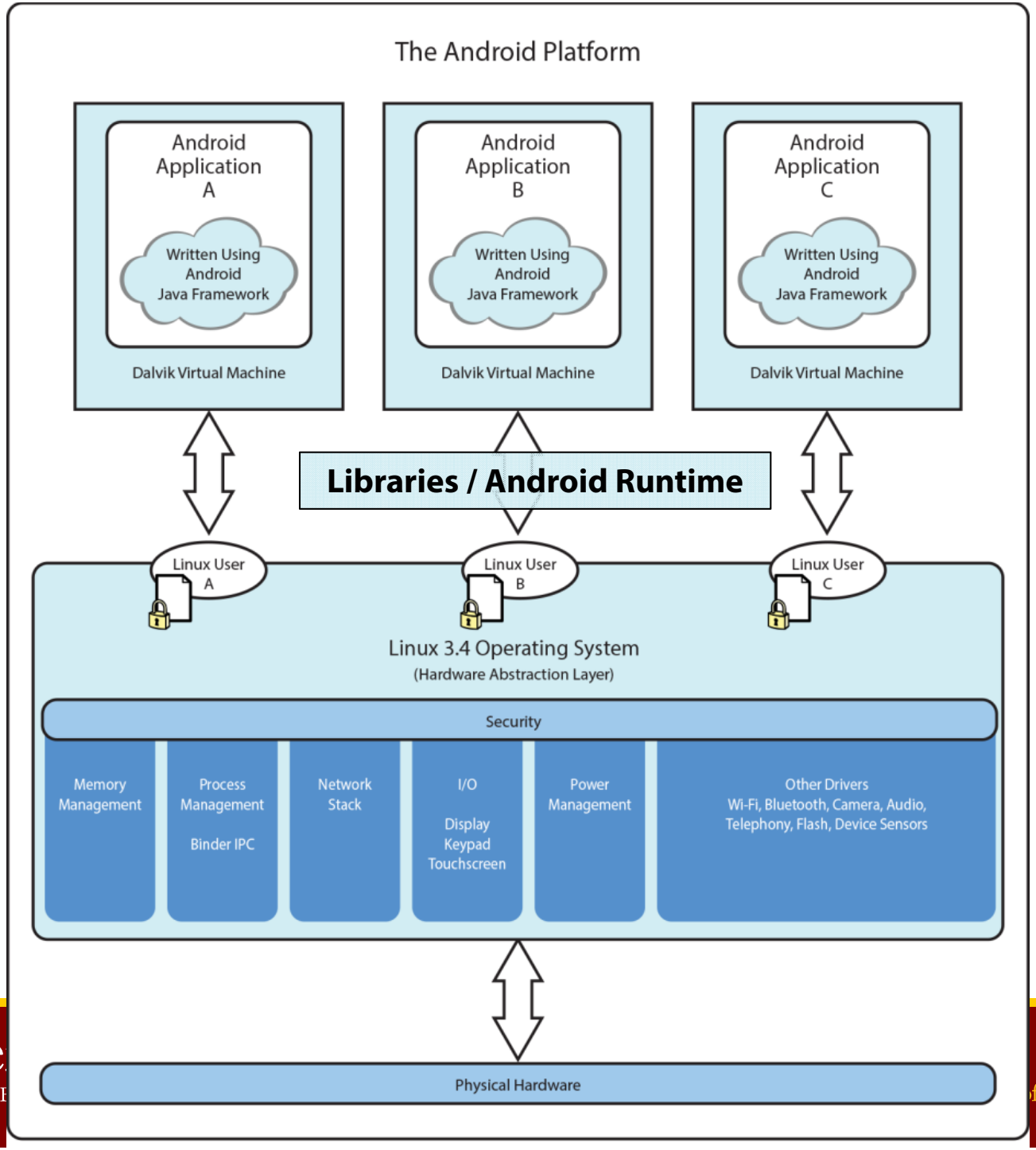
Android's Underlying Architecture

- Four layers
  - Linux (kernel)
  - Libraries and Android runtime environment
  - Application framework
  - Application


The Android Platform
Android Application A
Written Using Android Java Framework
Dalvik Virtual Machine
Android Application B
Written Using Android Java Framework
Dalvik Virtual Machine
Android Application C
Written Using Android Java Framework
Dalvik Virtual Machine
Libraries / Android Runtime
Linux User A
Linux User B
Linux User C
Linux 3.4 Operating System
(Hardware Abstraction Layer)
Security
Memory Management
Process Management
Binder IPC
Network Stack
I/O
Display
Keypad
Touchscreen
Power Management
Other Drivers
Wi-Fi, Bluetooth, Camera, Audio, Telephony, Flash, Device Sensors
Physical Hardware

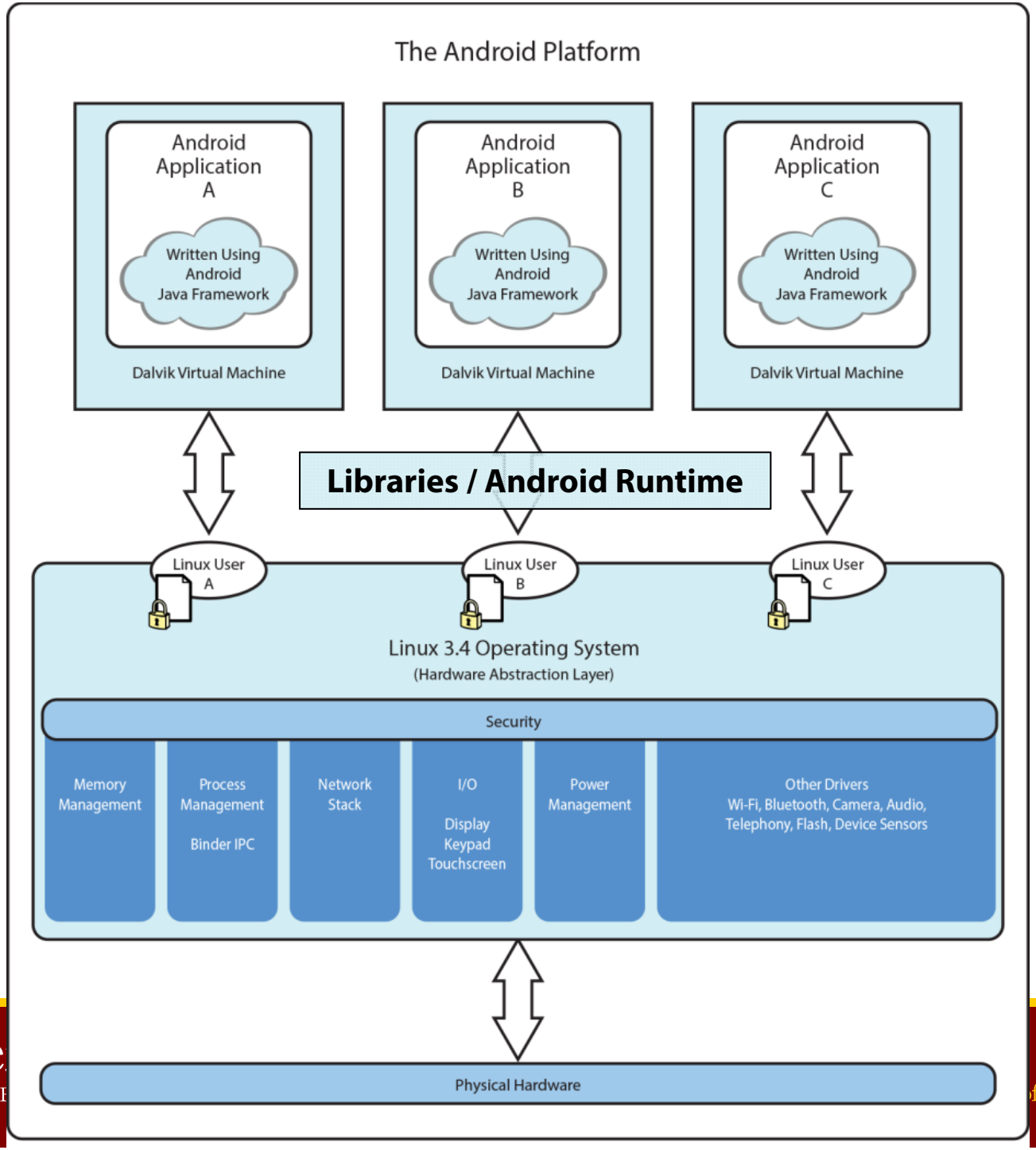
Android's Underlying Architecture

- Linux kernel *("brain" of the operating system)*
  - Handles core system services
    - Manages processes, threads, and low-level memory
    - Network stack
    - Enforces application permissions / security
  - Contains low-level devices drivers / communicates with hardware


The Android Platform
Android Application A
Written Using Android Java Framework
Dalvik Virtual Machine
Android Application B
Written Using Android Java Framework
Dalvik Virtual Machine
Android Application C
Written Using Android Java Framework
Dalvik Virtual Machine
Libraries / Android Runtime
Linux User A
Linux User B
Linux User C
Linux 3.4 Operating System
(Hardware Abstraction Layer)
Security
Memory Management
Process Management
Binder IPC
Network Stack
I/O
Display
Keypad
Touchscreen
Power Management
Other Drivers
Wi-Fi, Bluetooth, Camera, Audio, Telephony, Flash, Device Sensors
Physical Hardware

Android's Underlying Architecture

- Libraries
  - Contains main features of OS
  - Ex: **SQLite** for databases, **WebKit** for web browsing
- Android application runtime environment
  - Libraries to use Java to write Android Apps
  - Dalvik virtual machine
    - Each app runs in its own Dalvik virtual machine (VM) in a separate process
    - Dalvik is based on the Java VM optimized for mobile

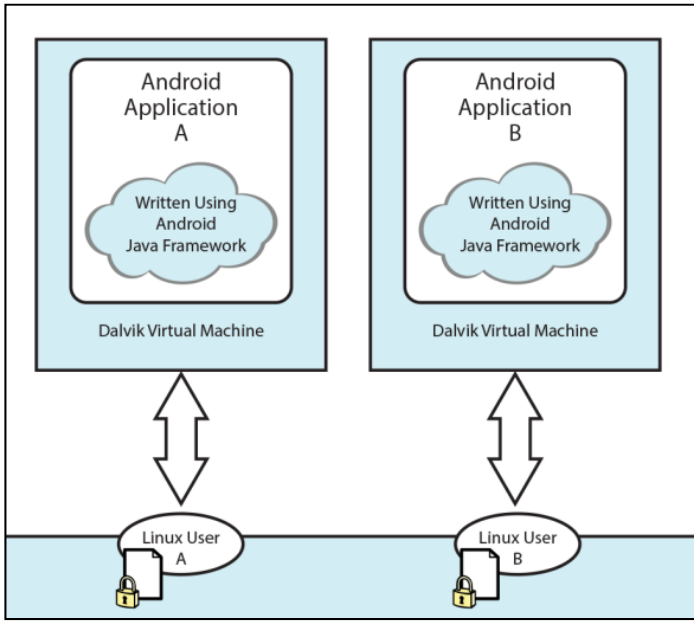

The Android Platform
Android Application A
Written Using Android Java Framework
Dalvik Virtual Machine
Android Application B
Written Using Android Java Framework
Dalvik Virtual Machine
Android Application C
Written Using Android Java Framework
Dalvik Virtual Machine
Libraries / Android Runtime
Linux User A
Linux User B
Linux User C
Linux 3.4 Operating System
(Hardware Abstraction Layer)
Security
Memory Management
Process Management
Binder IPC
Network Stack
I/O
Display
Keypad
Touchscreen
Power Management
Other Drivers
Wi-Fi, Bluetooth, Camera, Audio, Telephony, Flash, Device Sensors
Physical Hardware

Android's Underlying Architecture

- Application Framework
  - Code and libraries that developers can use to access core Android features
  - Ex: Location Manager for GPS / location services, Telephony Manager to access phone
- Application
  - Where our developed apps exist

Android Securities and Permissions

- Key Principle: **Sandbox**
- Each app is a unique user
  - Apps can't interfere with others
  - Private files system
- Robust permissions
  - Each app must explicitly request permissions
- Application signing for trust relationships
- Google Play developer registration



CORE CONCEPTS

Core Android Concepts

- Manifest
- Application*
- Activities
  - Context*
- Fragments
- Resources
- Layout
- Intents
- Services

Not discussed here

Manifest

- Central configuration file (XML) for your entire application
- Specifies which components you app has
  - Activities
  - Services
  - Others
- Specifies application settings
  - Required permissions
  - API version
  - Etc.

AndroidManifest.xml

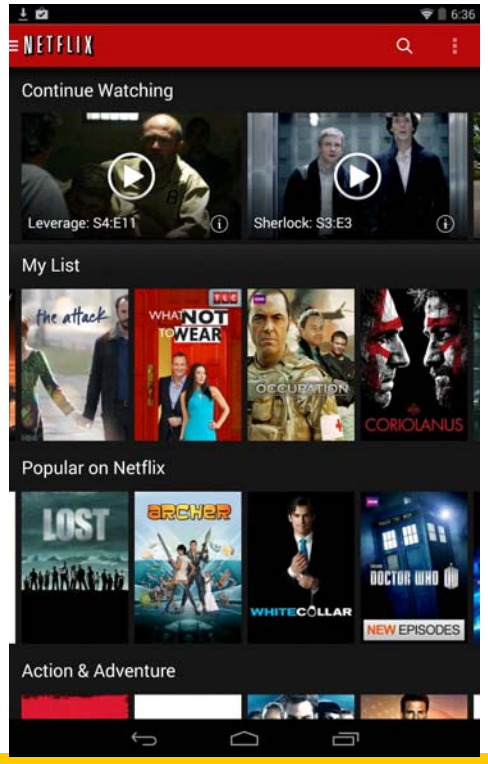
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="itp341.rob.parke.a5.app"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="18" />
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
      …
</application>
</manifest>
```

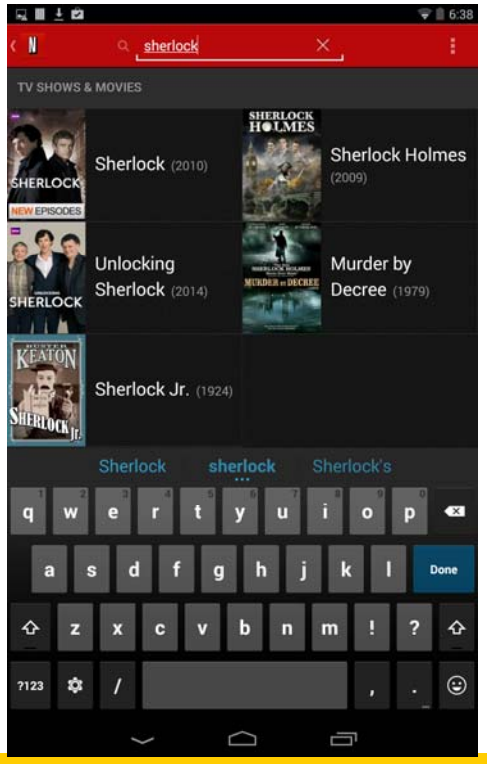
`Activity` Class

- Key building block
- Any window that has user interface (UI) elements
  - "A screen"
- An application will have many tasks
  - An activity will perform one task
- Create subclass that extends **`Activity`** class

`Activity` Class

- Ex: Netflix application with three distinct activities (meaning three different classes)

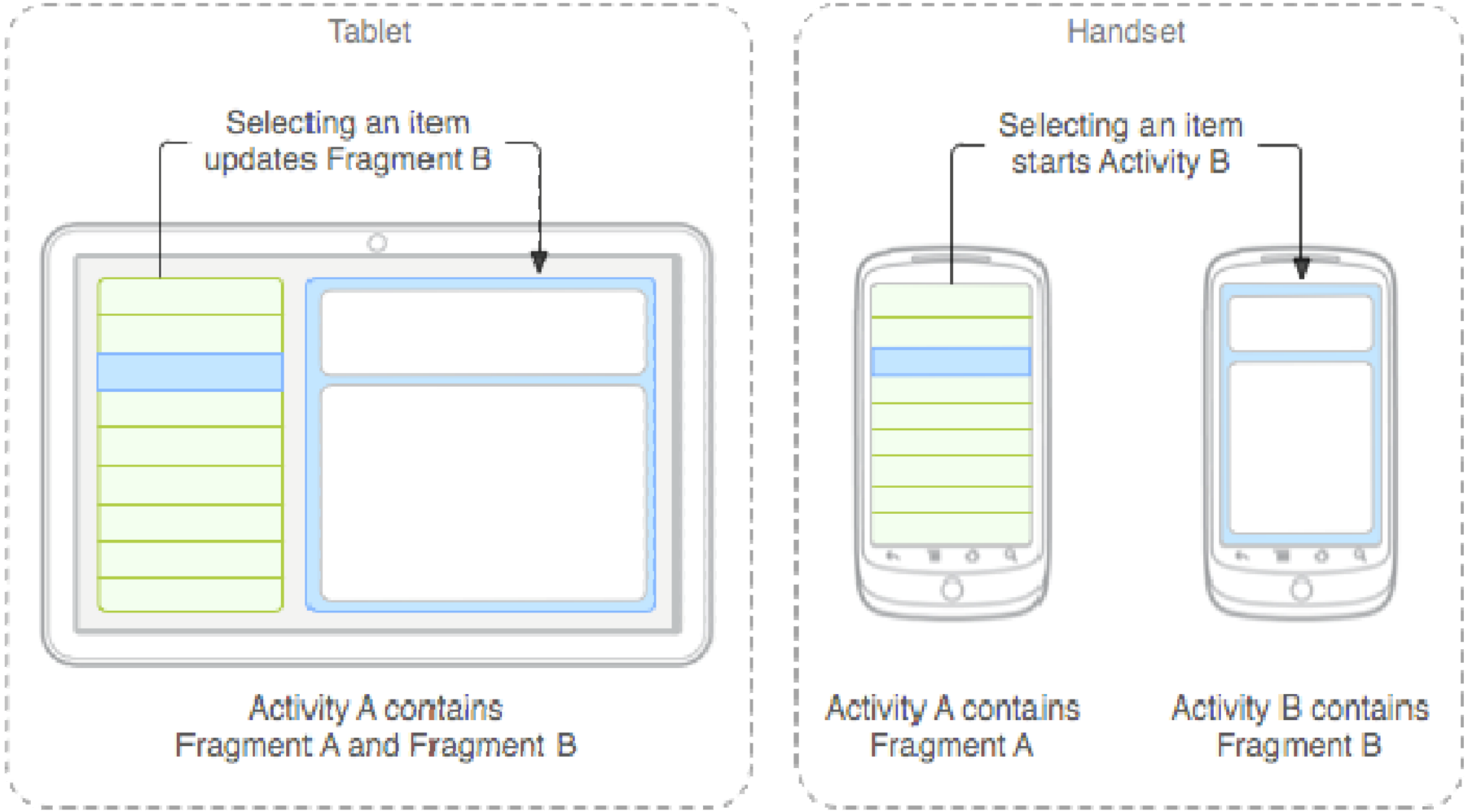




`Fragment` Class

- Partial activity
- Used to break up large activities into multiple pieces
- Simplifies developing on different devices types (e.g. phone, tablet, watch)

`Fragment` Class



Layout

- XML document that describes UI widgets (e.g. buttons, text fields)
- Specifies layout of components, sizes, colors, etc.
- UI layout is a separate file from application code

Layout

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/greeting"
        android:text="@string/greeting"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true">
</RelativeLayout>
```

Resources – The "Data"

- **Separate any data (including all text) from the actual code**
- Why?
  - Simplifies developing for multiple devices
  - Simplifies developing for multiple languages
- Examples
  - Text
  - Images / Audio / video
  - Colors, Fonts, Dimensions (of items on the screen)
- These are mostly stored in XML format
  - Images / video / audio can be stored in raw format

Resources

```
<resources>
    <string name="greeting">Hello!</string>
</resources>
```

…

```
<resources>
   <color name="opaque_red">#f00</color>
</resources>
```

Common Application Resources

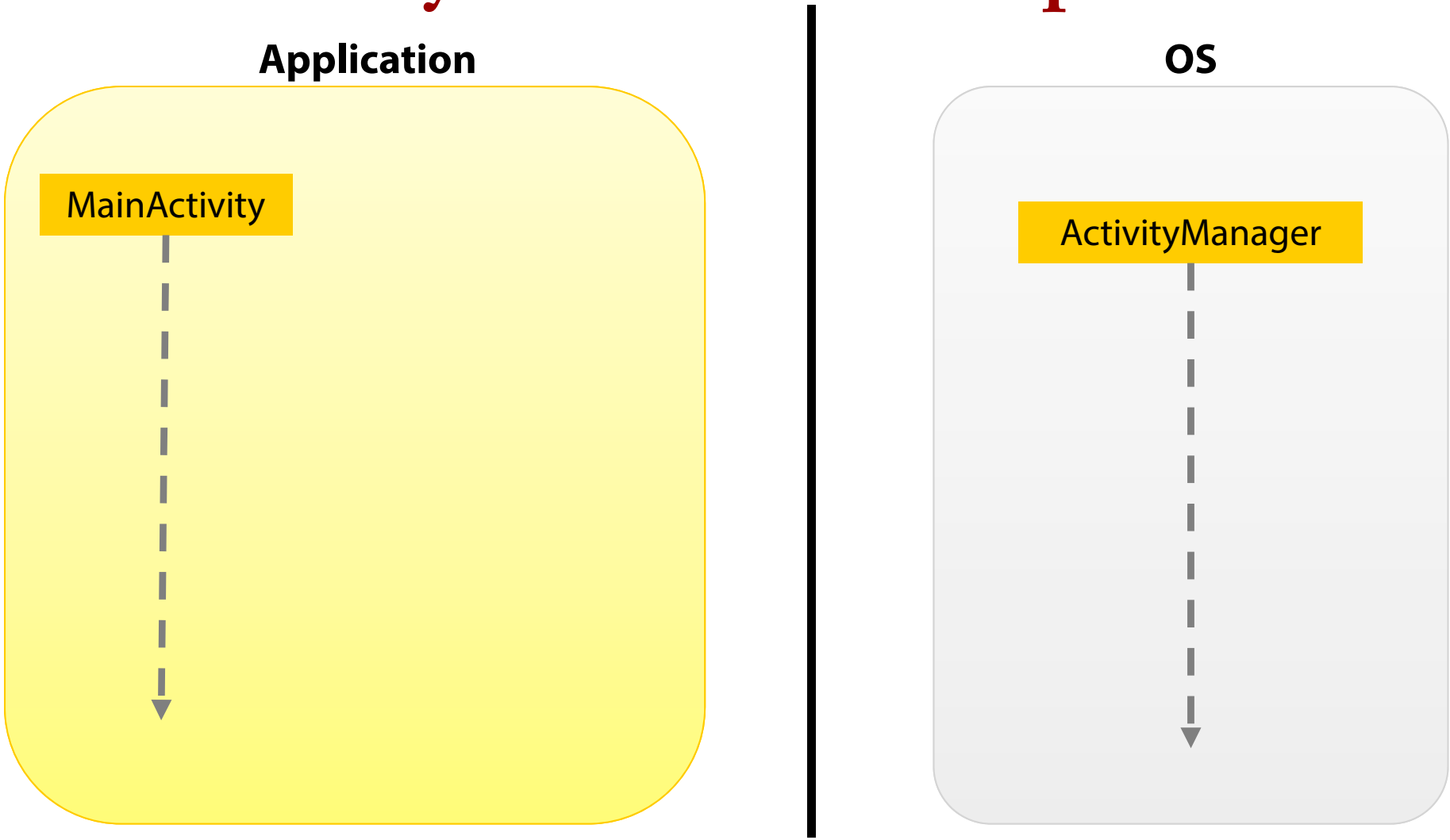
Resource Type	Directory
Property animations	`/res/animator/`
Color state lists	`/res/color/`
Drawables	`/res/drawable/`
Layouts	`/res/layout/`
Menus	`/res/menu/`
Arbitrary raw files	`/res/raw/`
Simple values	`/res/values/`
Arbitrary XML	`/res/xml`

All resources must be stored in /res in subdirectories that with lowercase names

`Intent` Class

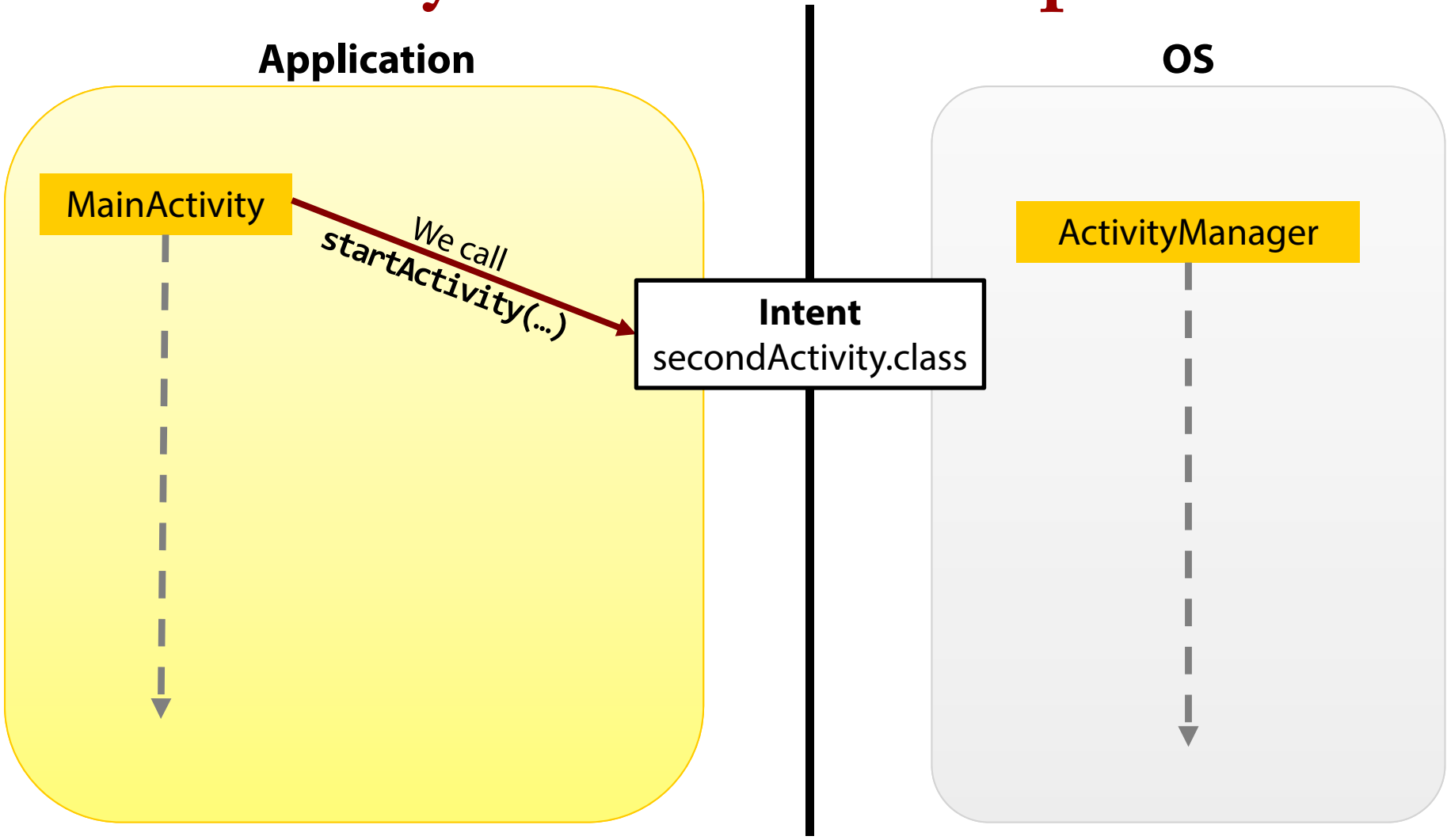
- Messages that allow activities to communicate with each other
- Mechanism for an activity to start / trigger another
  - Call specific activity (explicit)
  - Let Android select any activity that is prepared to handle the intent (implicit)

Activity Transitions Explained



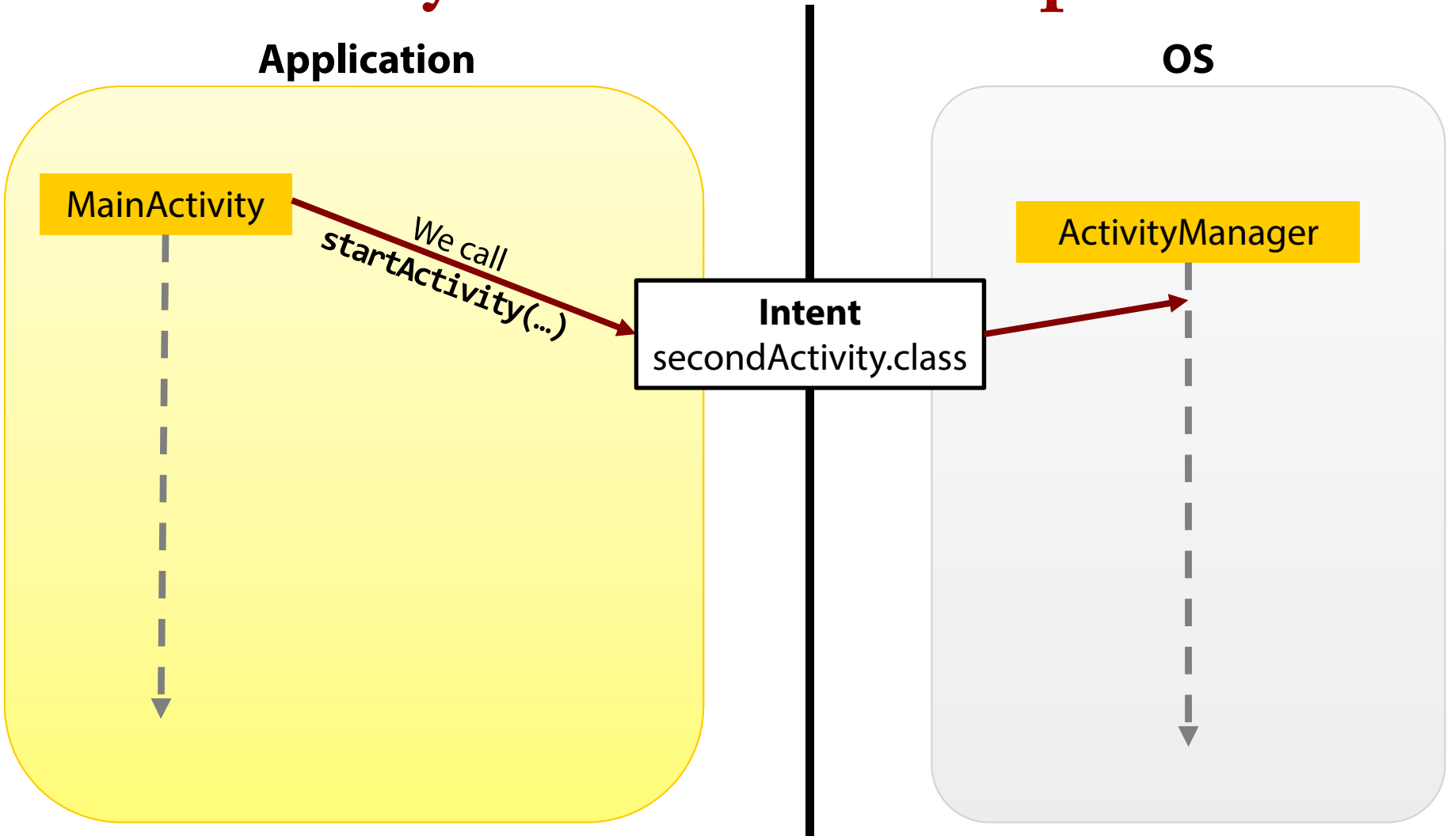
cf. *Android Programming*, Phillips (fig 5.11)

Activity Transitions Explained

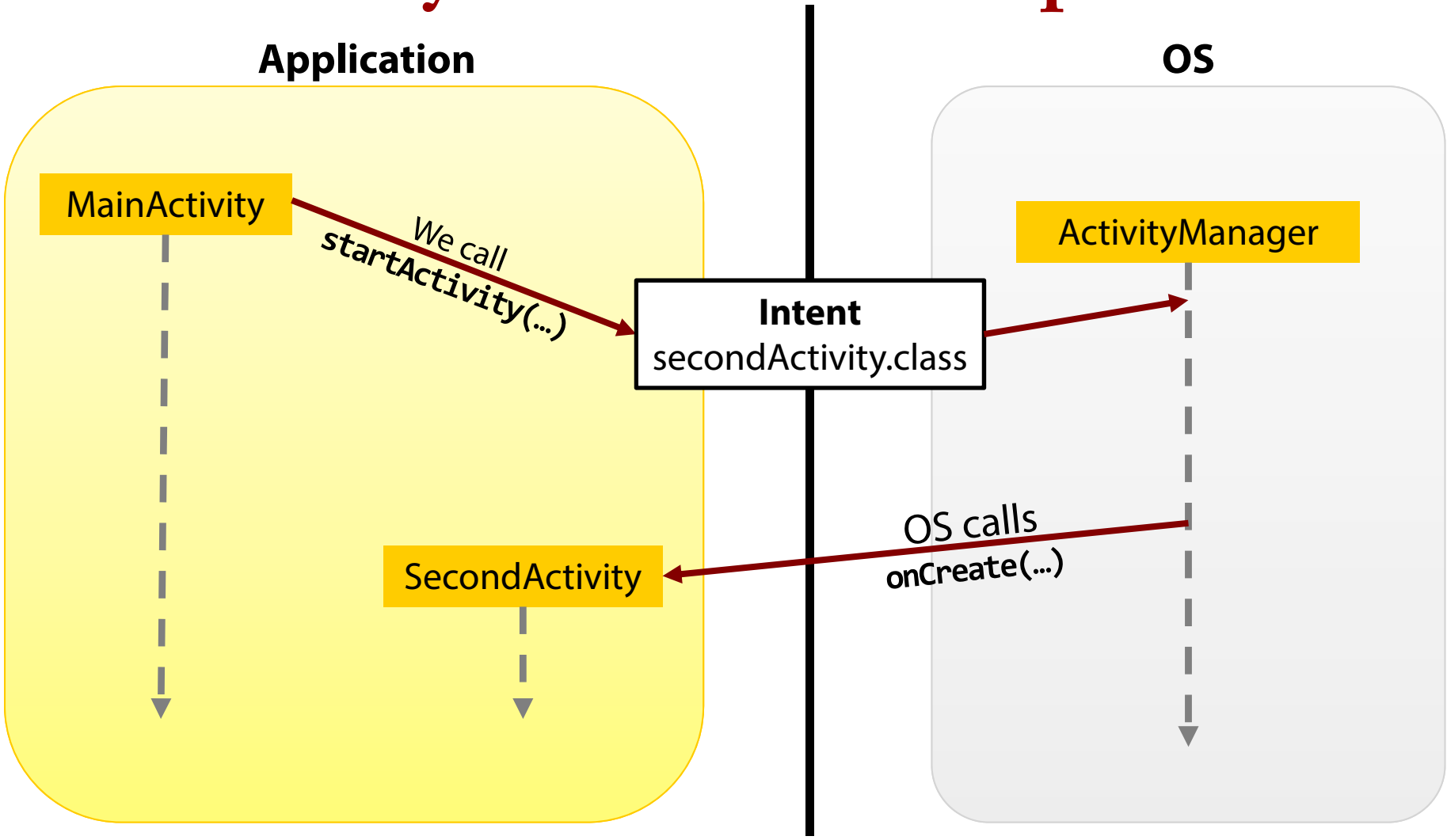


cf. *Android Programming*, Phillips (fig 5.11)

Activity Transitions Explained



Activity Transitions Explained



Services

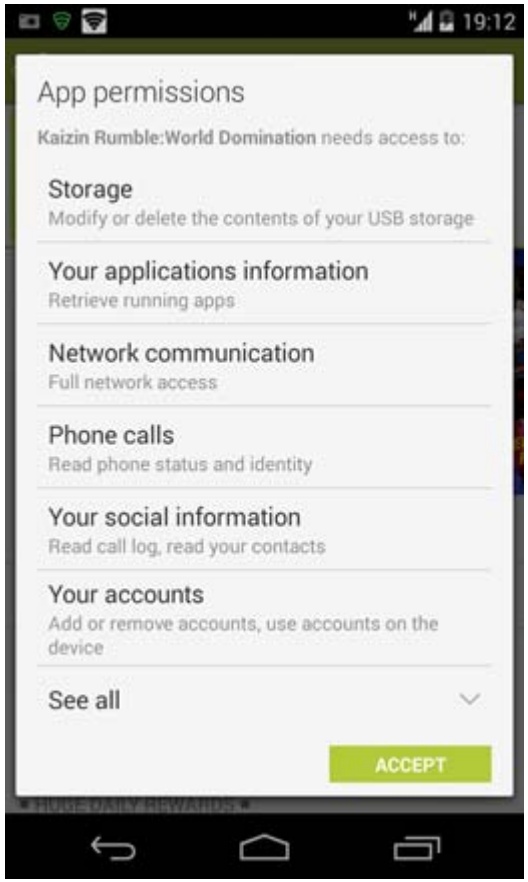
- Tasks that do not require user input or UI
- Examples
  - Downloading updates to feed
  - Syncing Dropbox data
  - Playing music in the background

Other Differences in Android (vs. iOS)

- Any "configuration change" destroys / recreates activity
  - Rotating device destroys all your variables / run state
- Widgets everywhere!
- Android has an activity back stack

Privacy

- On installation user must agree to all app permissions
- Not possible to disable individual permissions for an app



ACTIVITY STACK

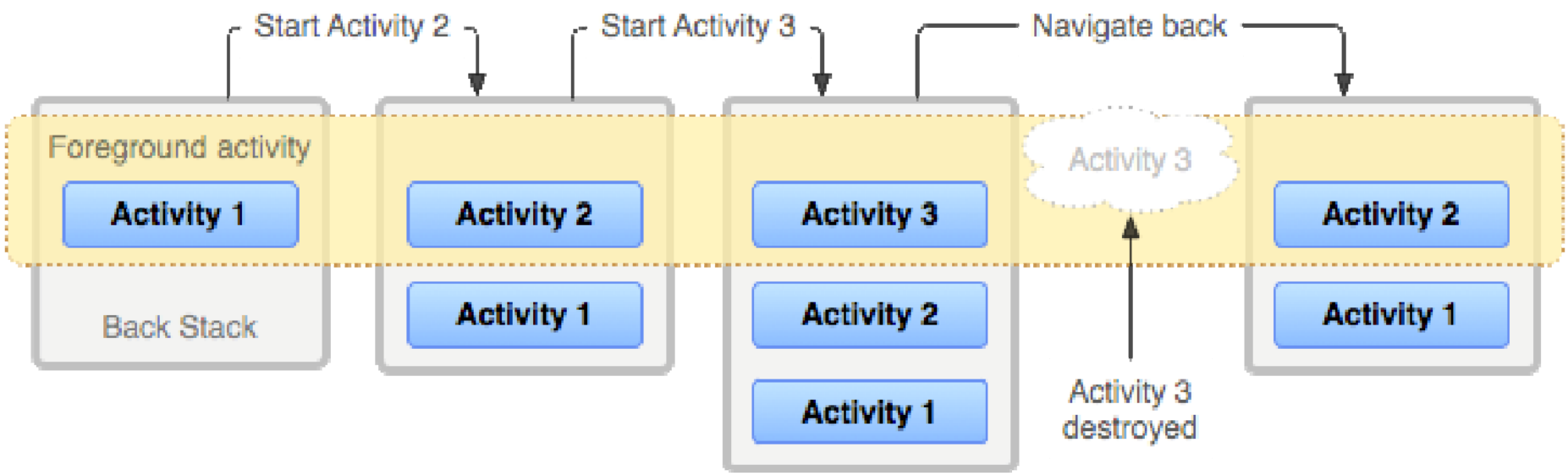
Lifecycle of an Android Activity

- Android allows multiple apps to run concurrently
- Apps can be interrupted and paused when events such as phone calls occur
- Only one application is **active** and **visible** to the user at a time
  - Specifically, a single application Activity is in the foreground at any given time

Lifecycle of an Android Activity

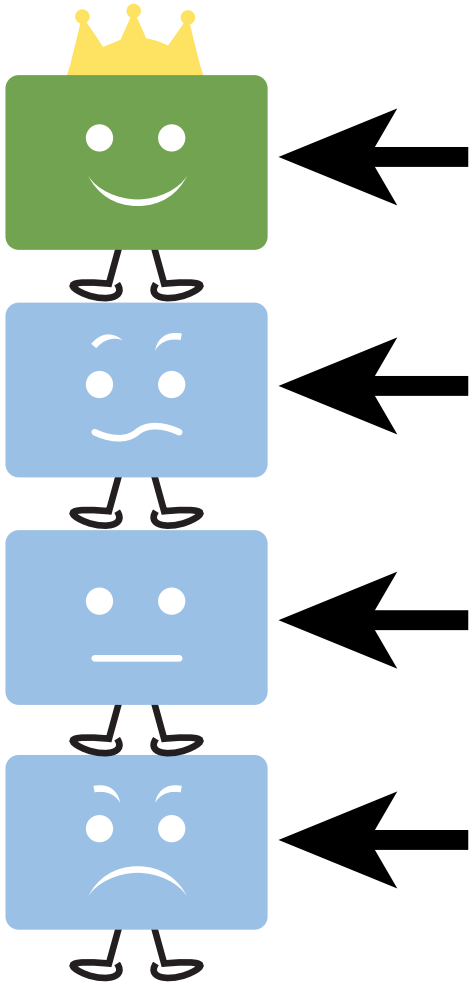
- Android keeps track of all Activity objects running by placing them on an **Activity stack**
  - Also know as **"back stack"**
- New **Activity 2** starts:
  - **Activity 1** on the top of the stack (current foreground) pauses
  - new **Activity 2** pushes onto the top of the stack
- **Activity 2** finishes:
  - **Activity 2** is removed from the Activity stack
  - The previous **Activity 1** in the stack resumes

Android Back Stack



android.com

Activity Stack



I am the top Activity.
User can see and interact with me!

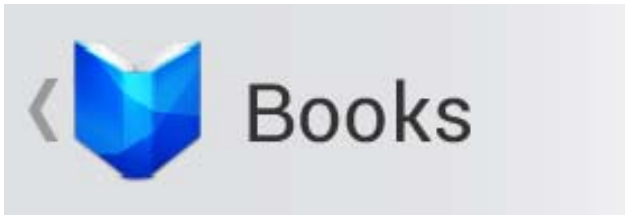
I am the second Activity in the stack.
If the user hits Back or the top Activity is destroyed, the user can see and interact with me again!

I am an Activity in the middle of the stack.
Users cannot see and interact with me until everyone above me is destroyed.

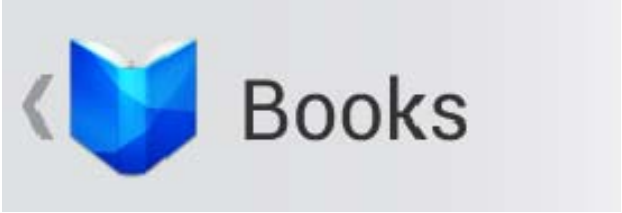
I am an Activity at the bottom of the stack.
If those Activities above me use too many resources, I will be destroyed!

Navigation with Back and Up

- Consistent navigation is an essential component of the overall user experience.
- Since action bars were added in Android 3.0, there are two ways navigate to a previous screen
  - **Back** (system-wide button)
  - **Up** (app icon + left-point caret on action bar)

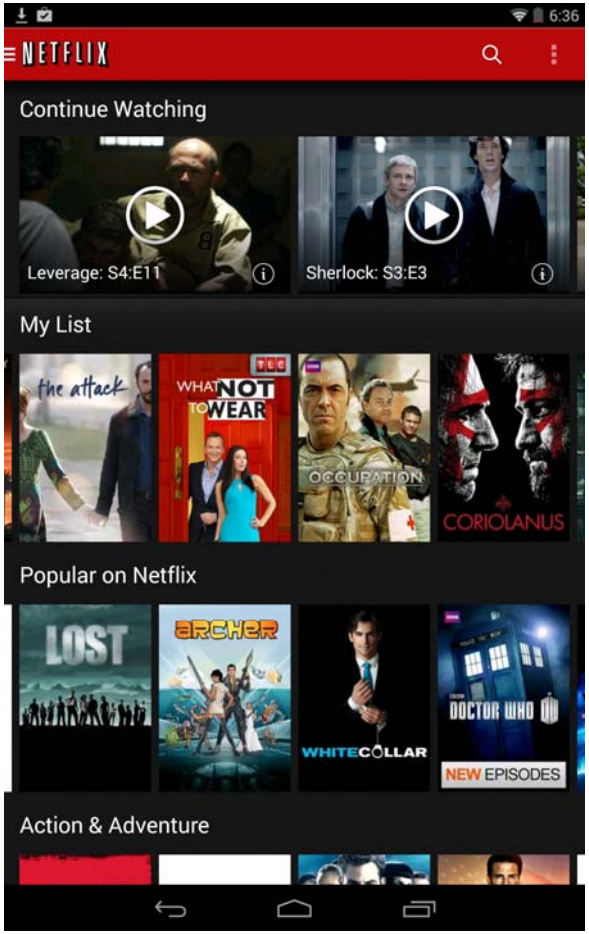


Example: Up Button



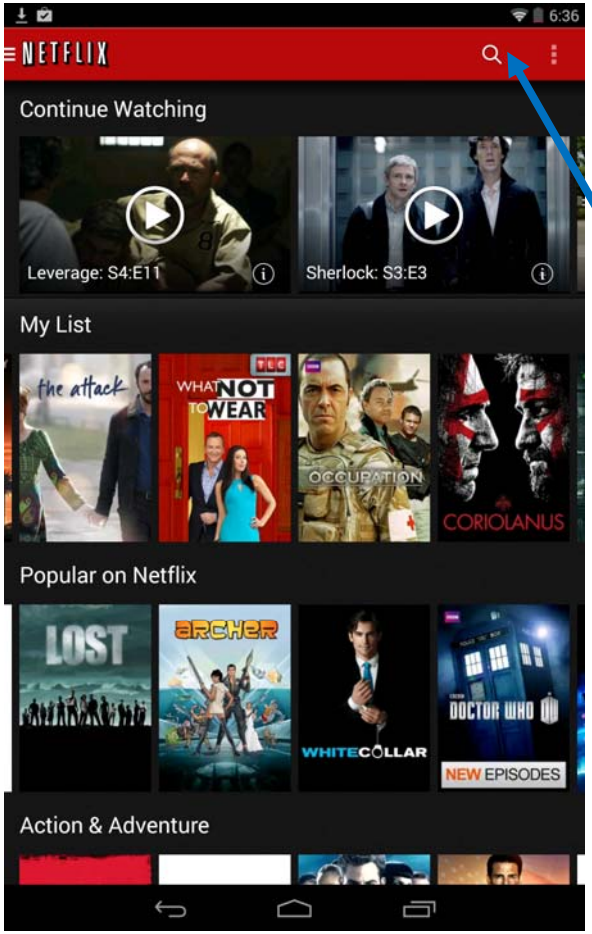
- Navigate within an app based on the hierarchical relationships between screens
- If a screen is the topmost one in an app (home), it should not present an Up button
- Guarantees user stays within current app

Example: Up Button



Main Activity

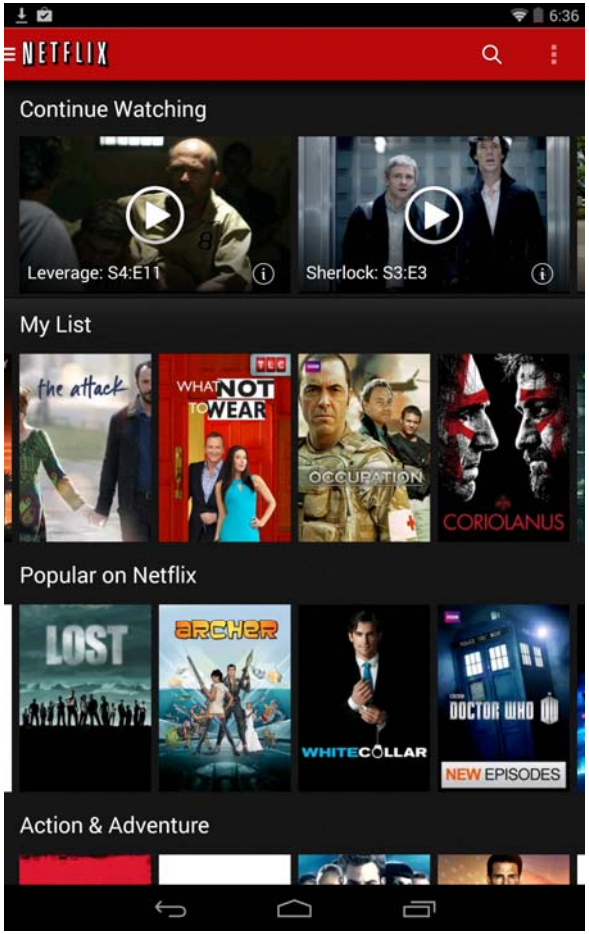
Example: Up Button



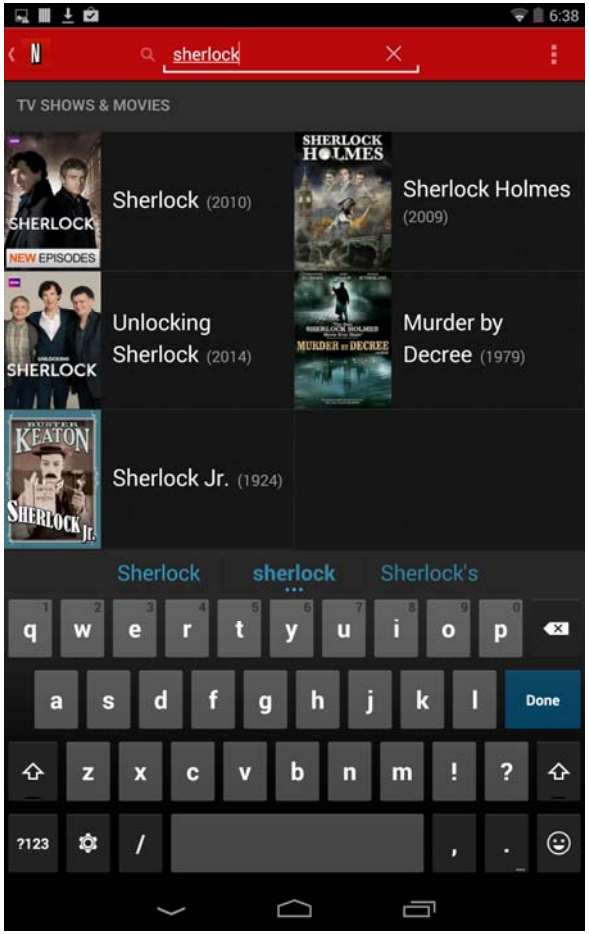
User clicks Search

Main Activity

Example: Up Button



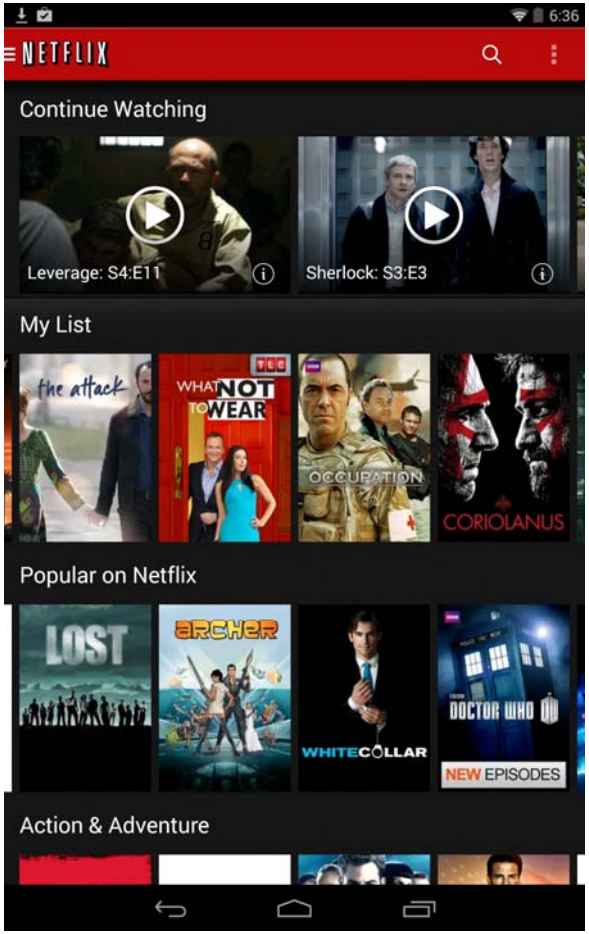
Search activity launches (child)



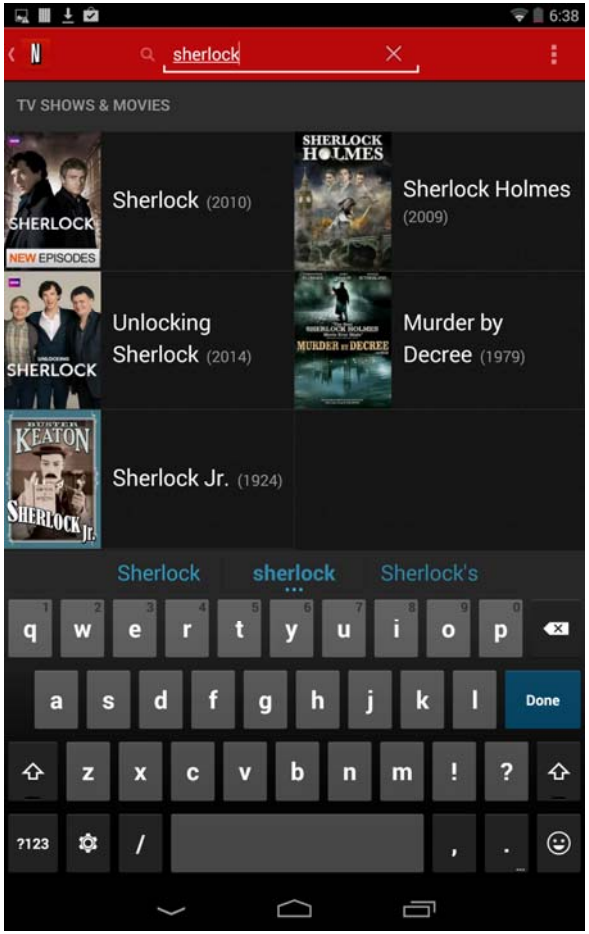
Main Activity

Search Activity (child)

Example: Up Button



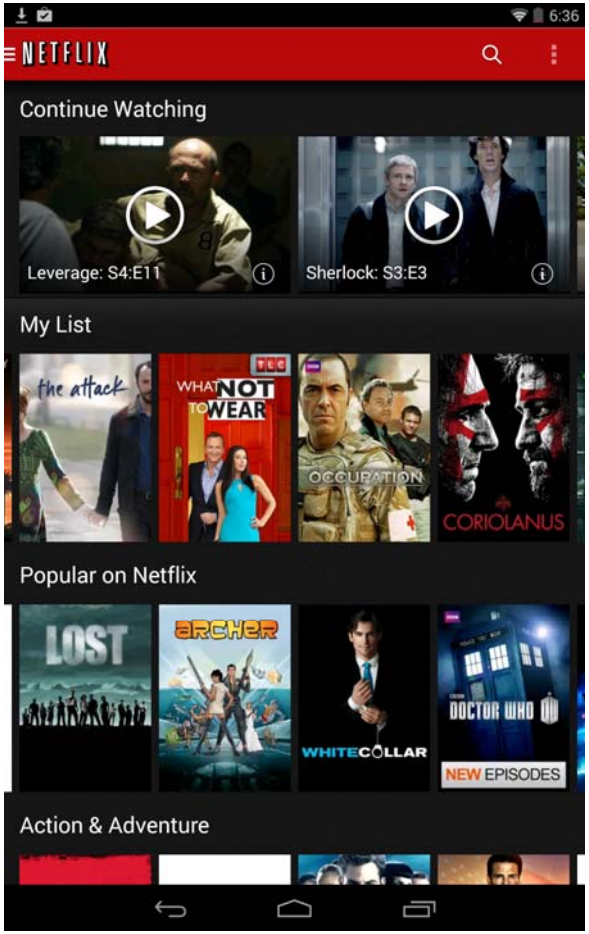
User clicks **Up**



Main Activity

Search Activity (child)

Example: Up Button



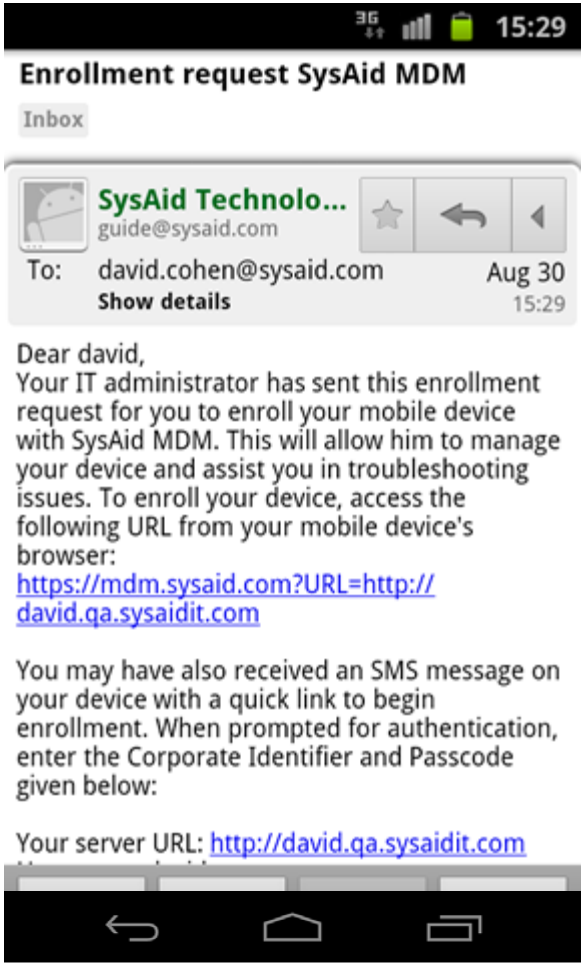
Navigate ***up*** hierarchy ***within*** app

Main Activity

Back Button

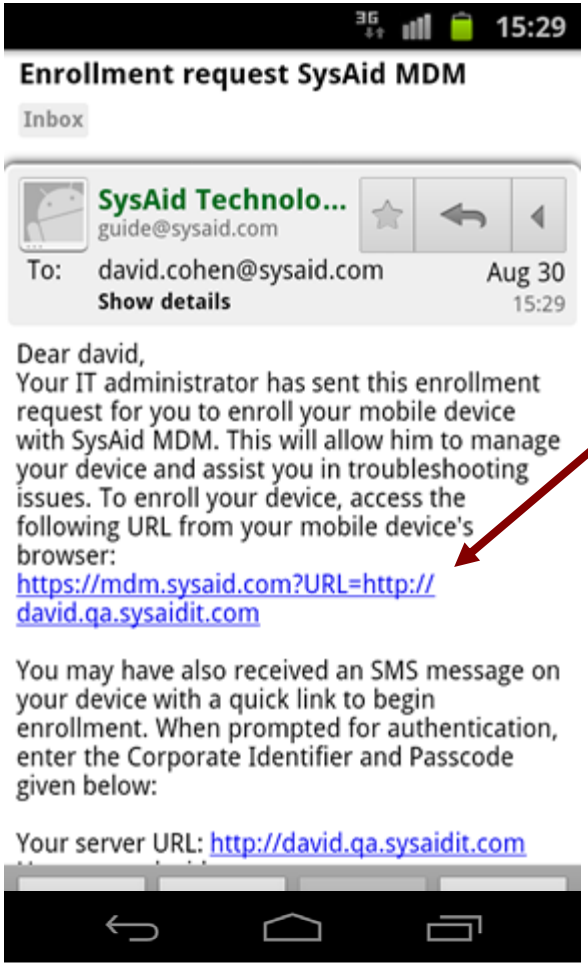
- Navigate (reverse chronological order) through the history of screens the user has recently worked with
- May result in user leaving current app

Example: Back Button



Email App

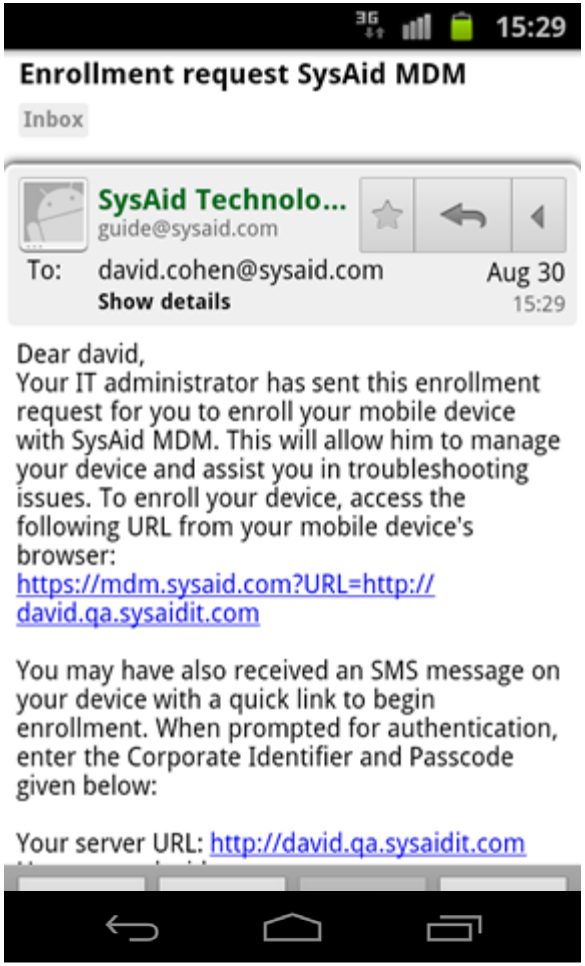
Example: Back Button



User clicks web link

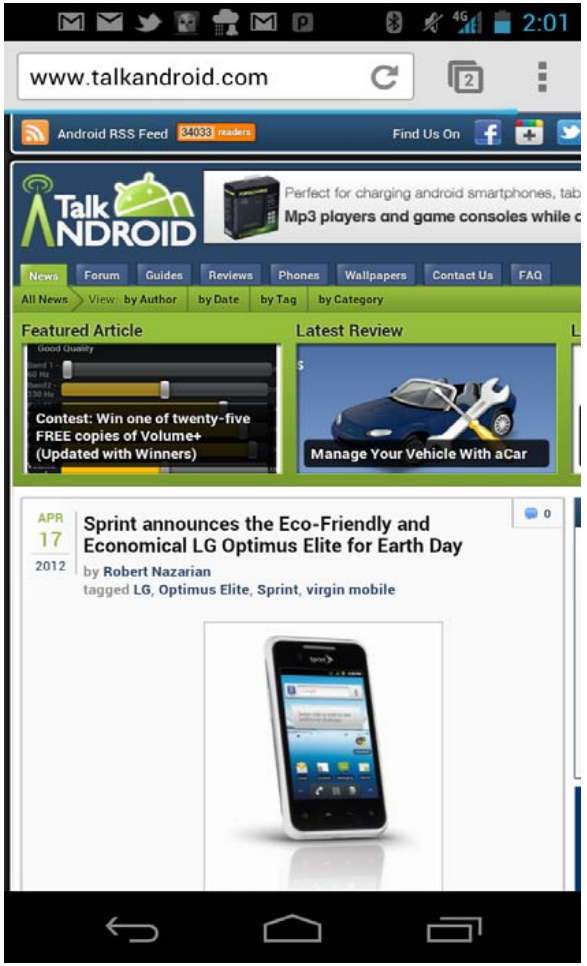
Email App

Example: Back Button



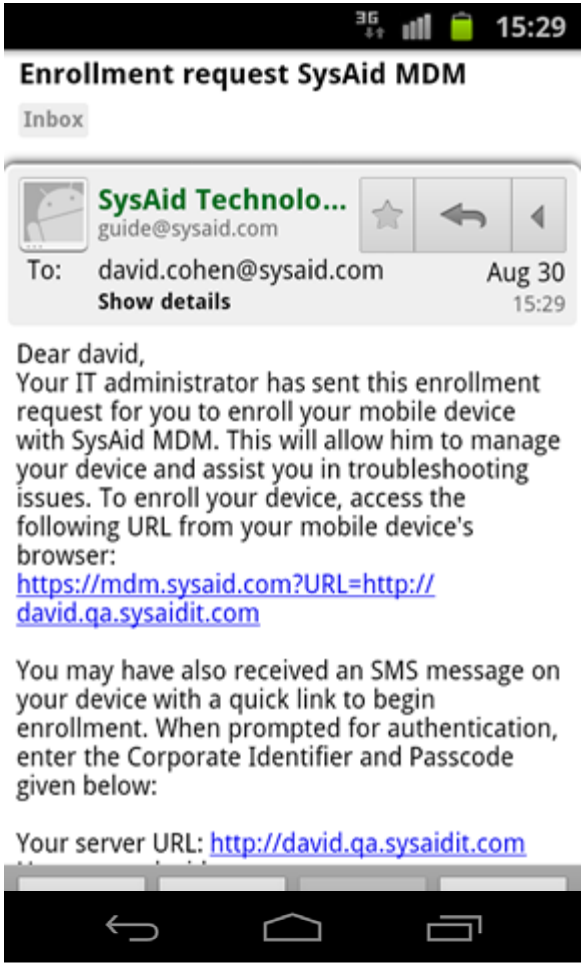
Email App

Browser app opens



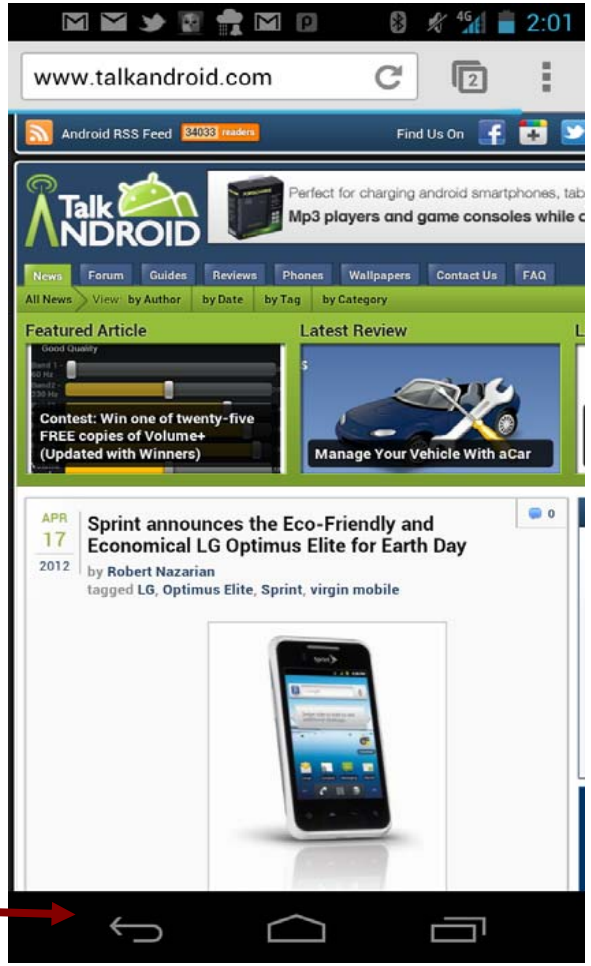
Browser App

Example: Back Button



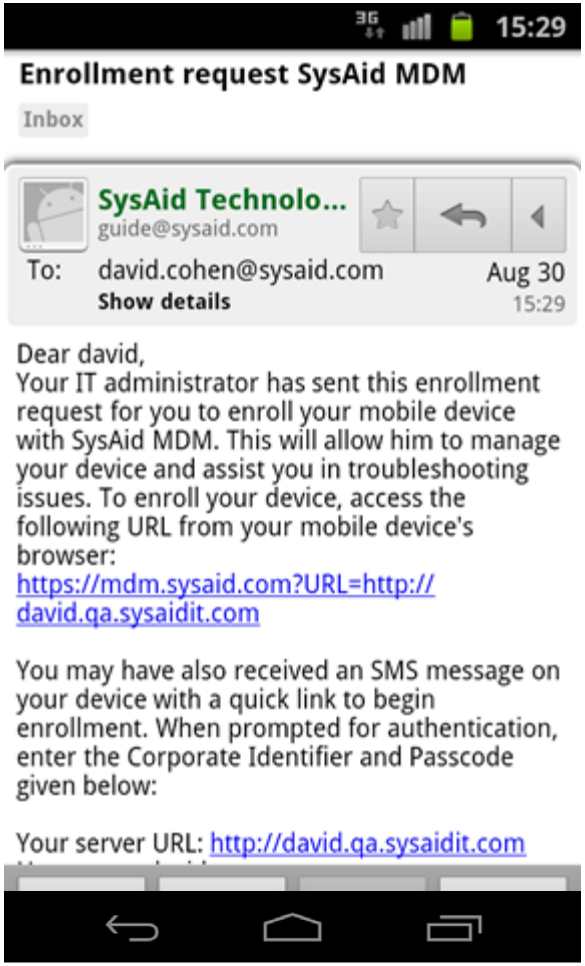
Email App

User clicks back



Browser App

Example: Back Button



Email App

Previous parent on stack restored

- Example