Android 101 for iOS Developers

Stephen Barnes - @smbarne

Why?

- Interact with people in on the same team more effectively
- Learn something new and useful from a new source

Overview

- A Word on UI Design
- Application Structure and Language
- Android Myths
- Building Blocks: Activities, Fragments, ListViews, ViewPagers, and more
- Android Lifecycle
- Android Layouts

Assumptions

- Working knowledge of Java
- Able to install and use the Android Development Tools
- You have written a mobile app before

Recommendations

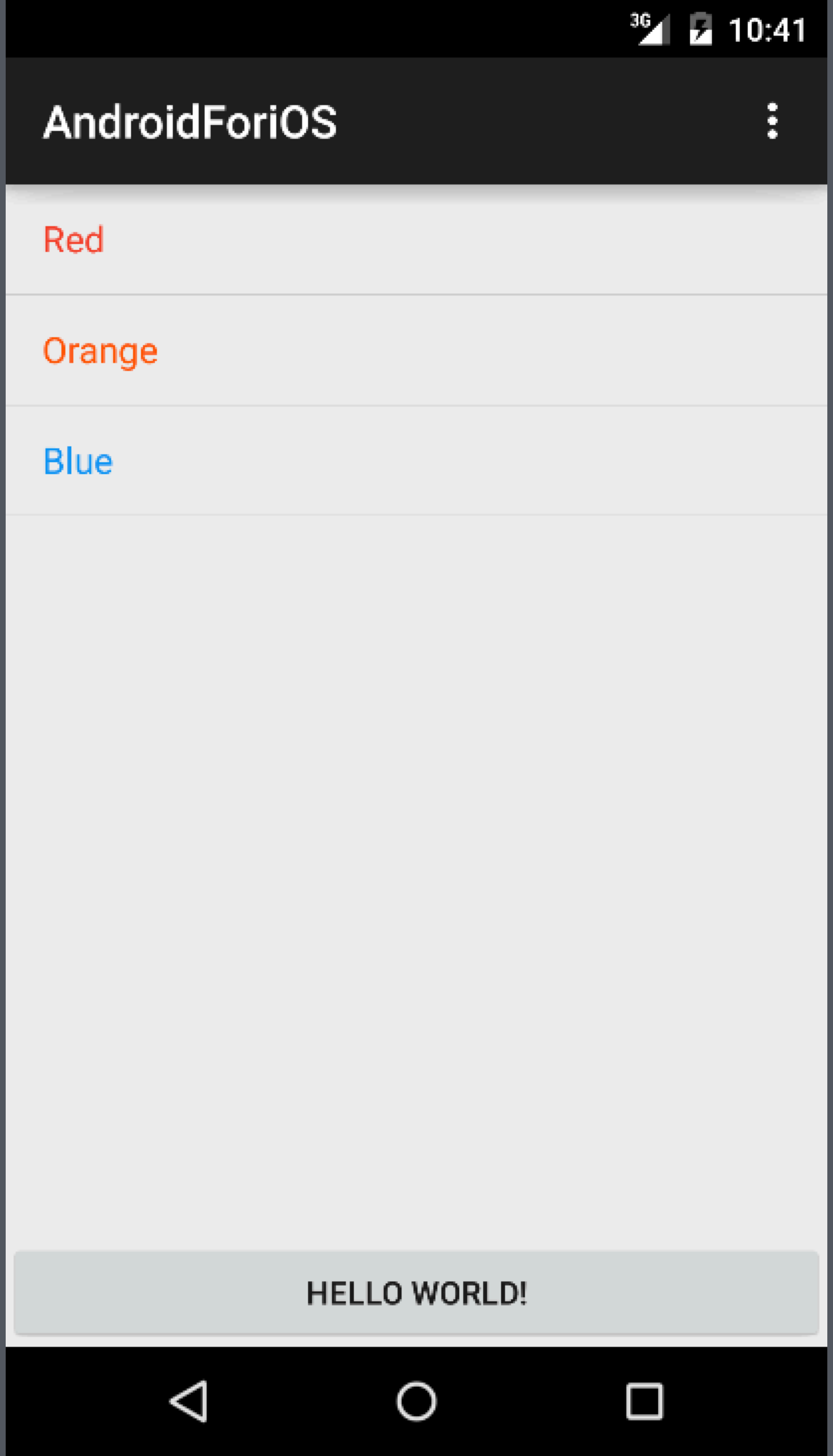
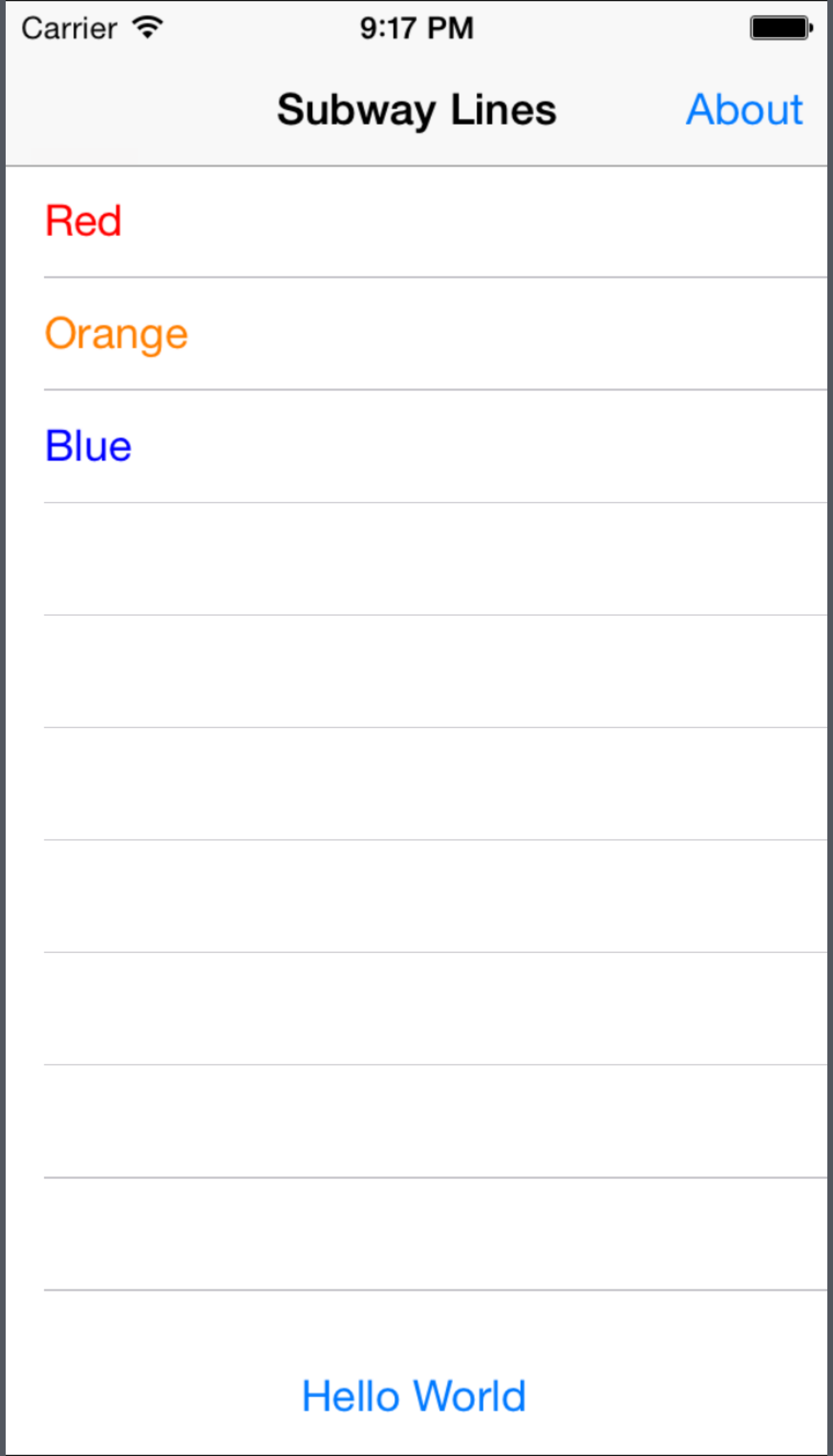
Read through the Building your first app tutorial by Google when possible.

Resources

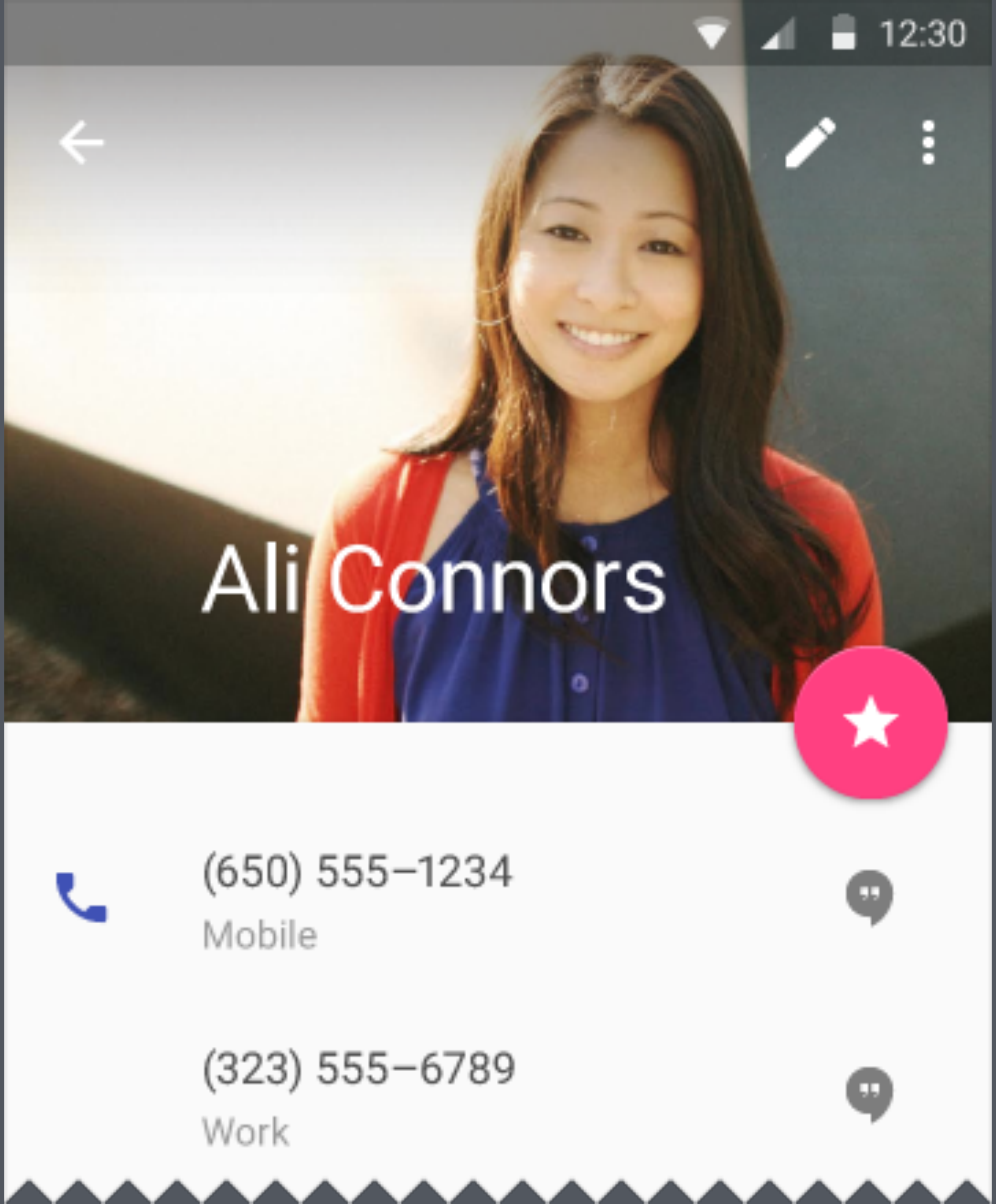
Objc.io article: objc.io/issue-11/android_101_for_ios_developers.html

Code: github.com/smbarne/AndroidForiOS

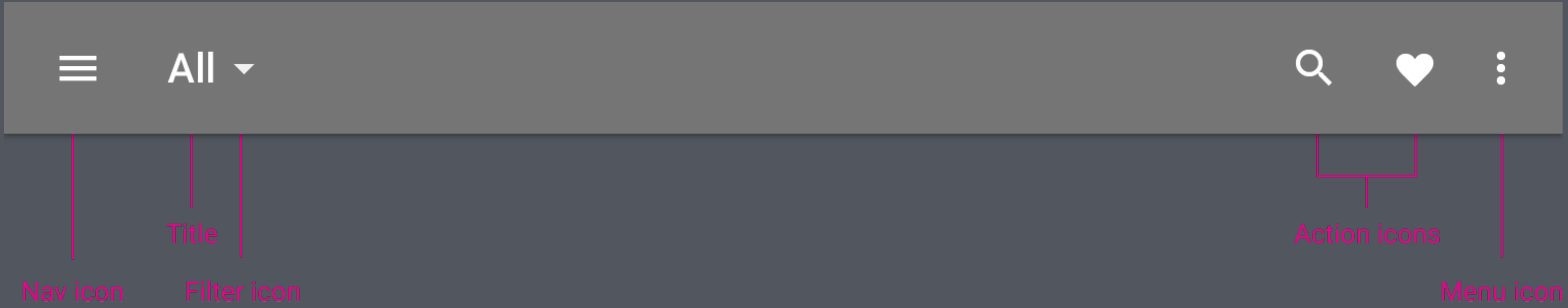
A Word on UI Design

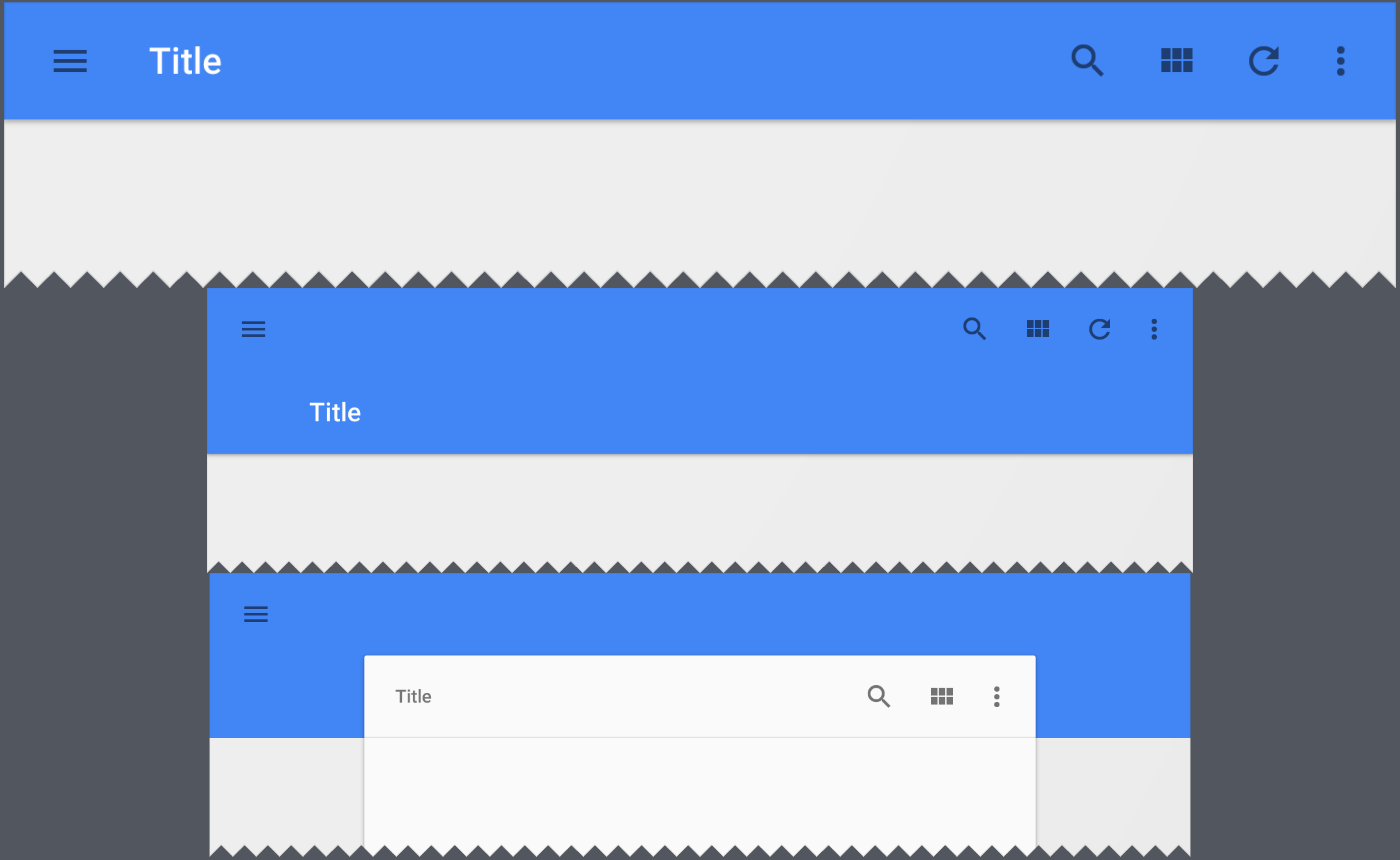

10:41
AndroidForiOS
Red
Orange
Blue
HELLO WORLD!

Carrier
9:17 PM
Subway Lines
About
Red
Orange
Blue
Hello World

Material Design

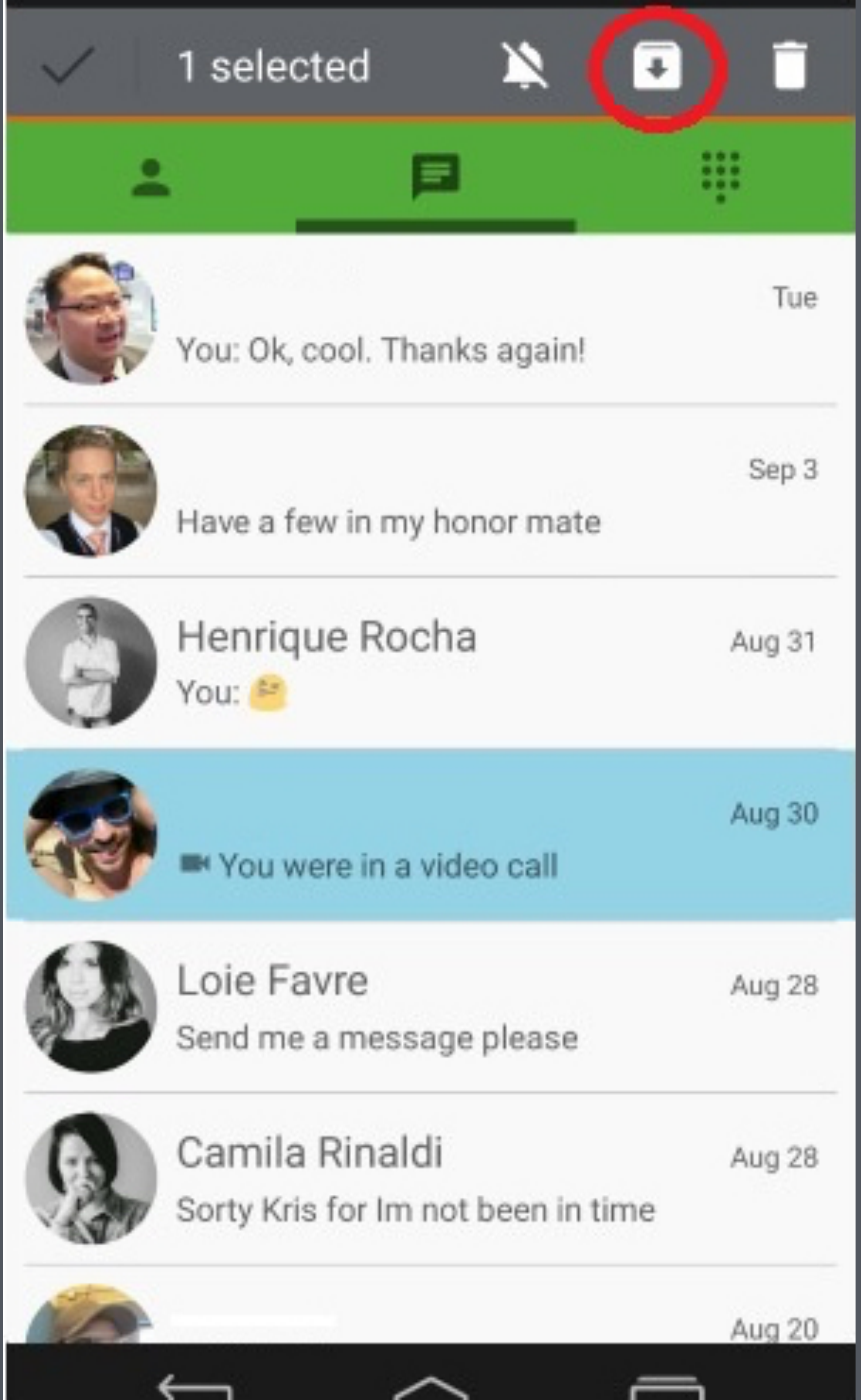


App Bar (previously Action Bar)




Title
Title
Title

App Bar Context



1 selected

Tue
You: Ok, cool. Thanks again!

Sep 3
Have a few in my honor mate

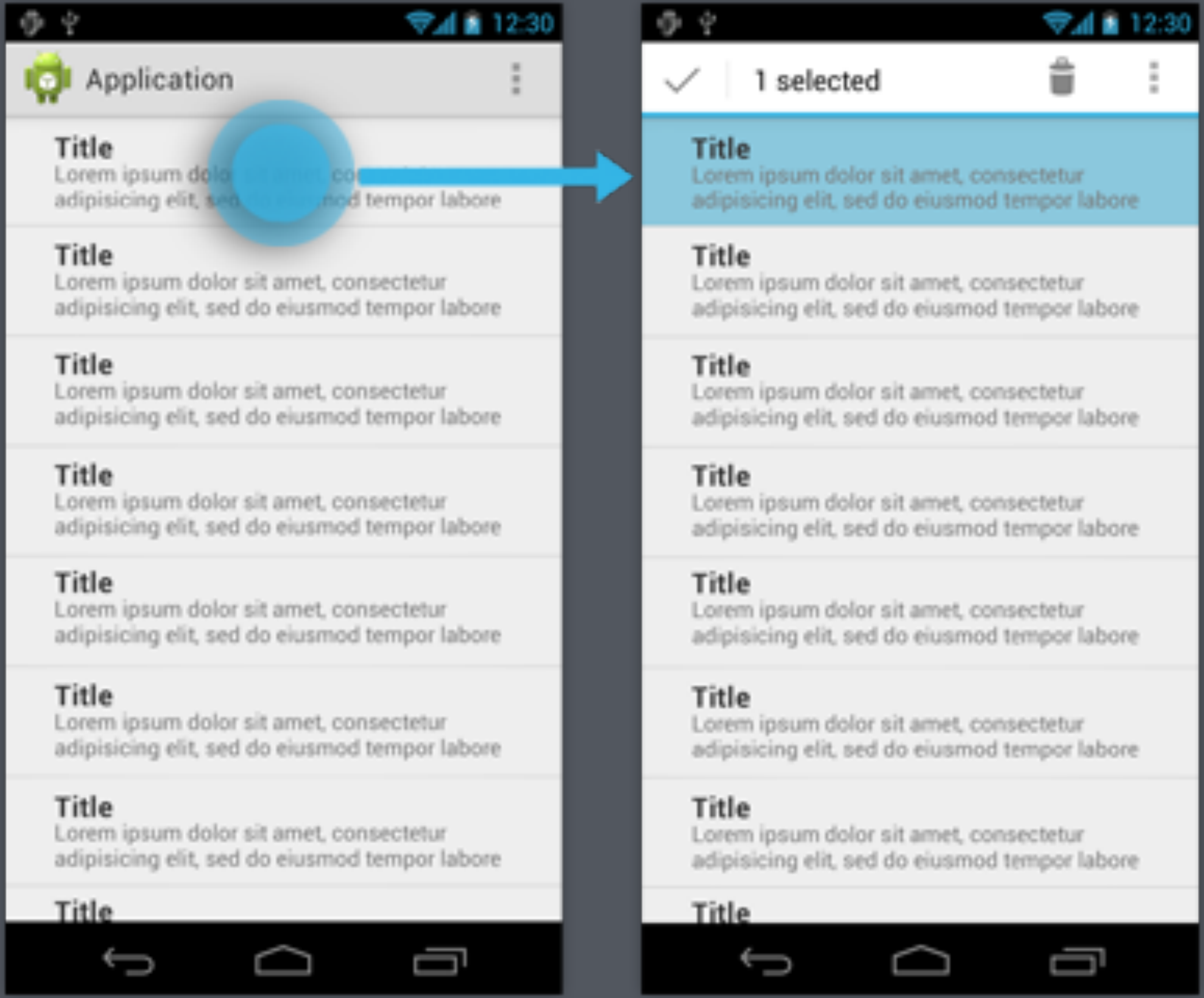
Henrique Rocha
Aug 31
You:

Aug 30
You were in a video call

Loie Favre
Aug 28
Send me a message please

Camila Rinaldi
Aug 28
Sorty Kris for Im not been in time

Aug 20



12:30
Application

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title

12:30
1 selected

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

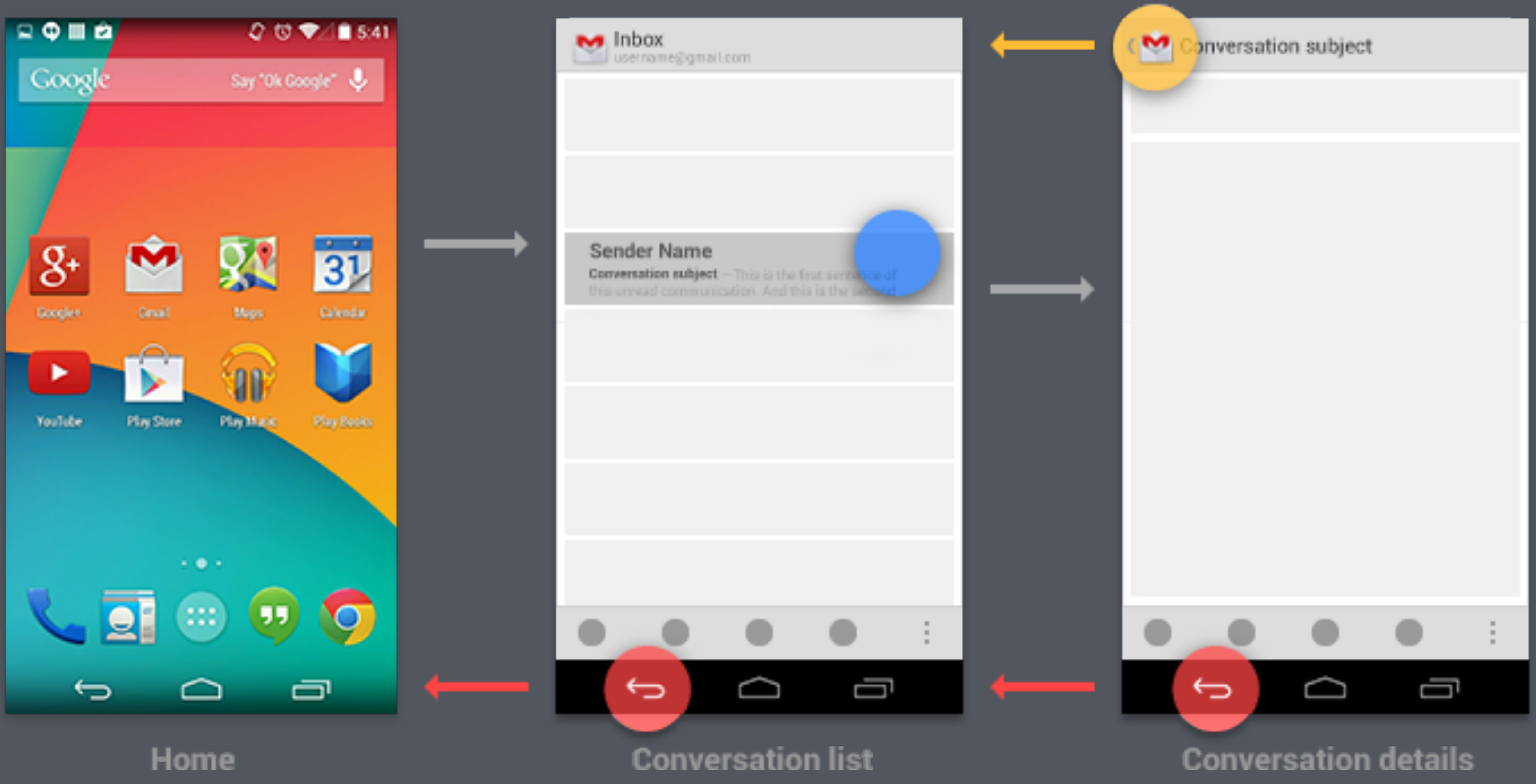
Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

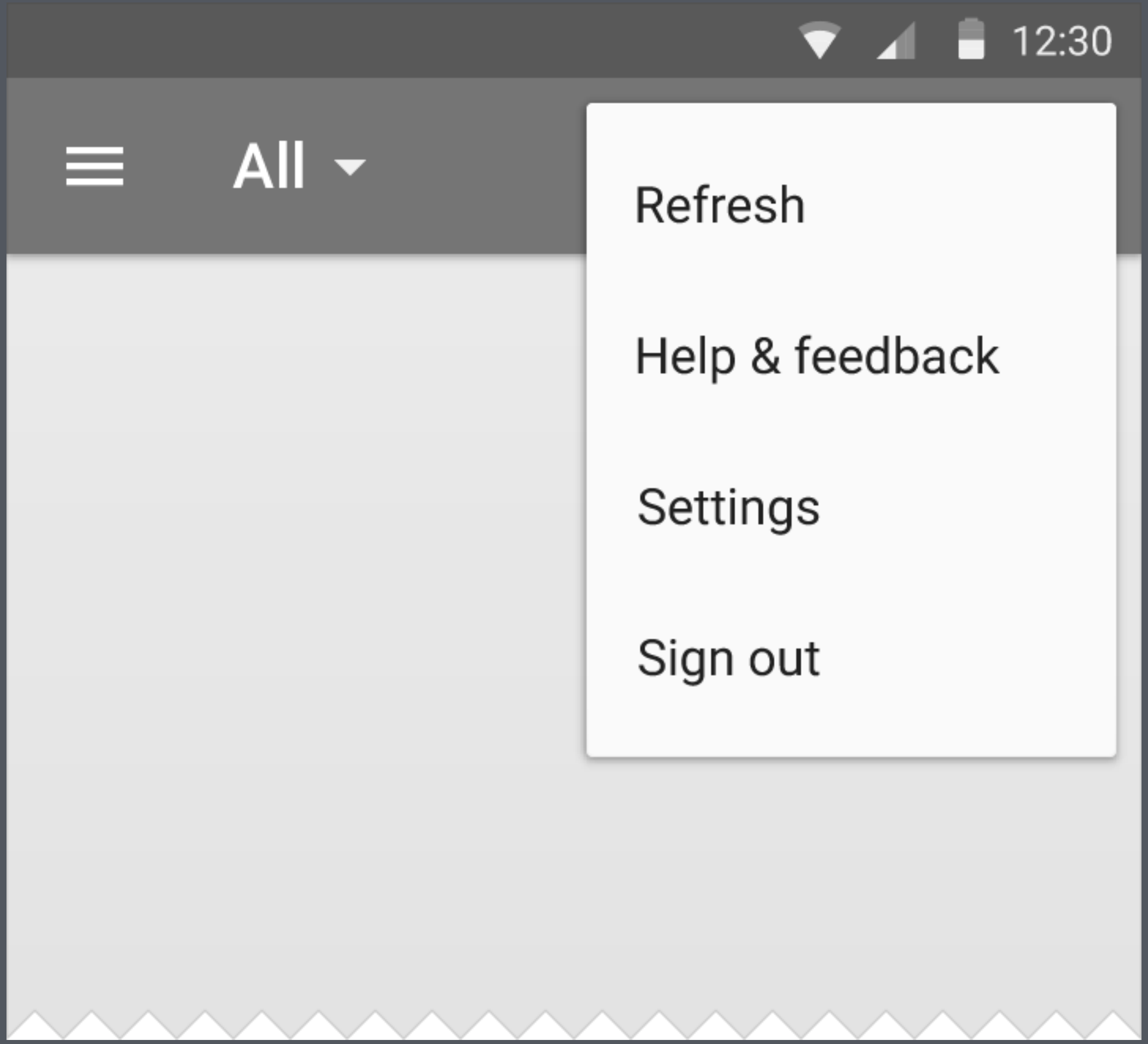
Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor labore

Title

Back Button and Navigation

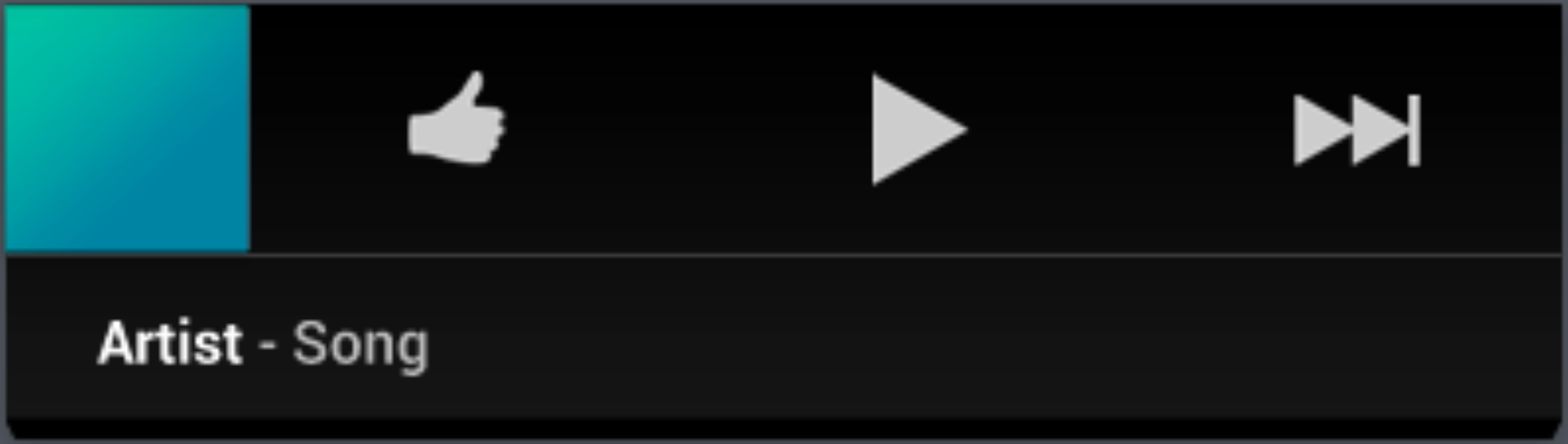


Overflow Menu

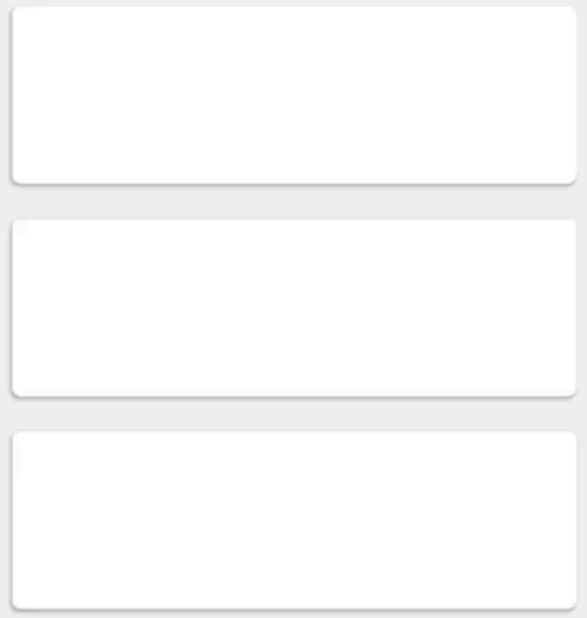


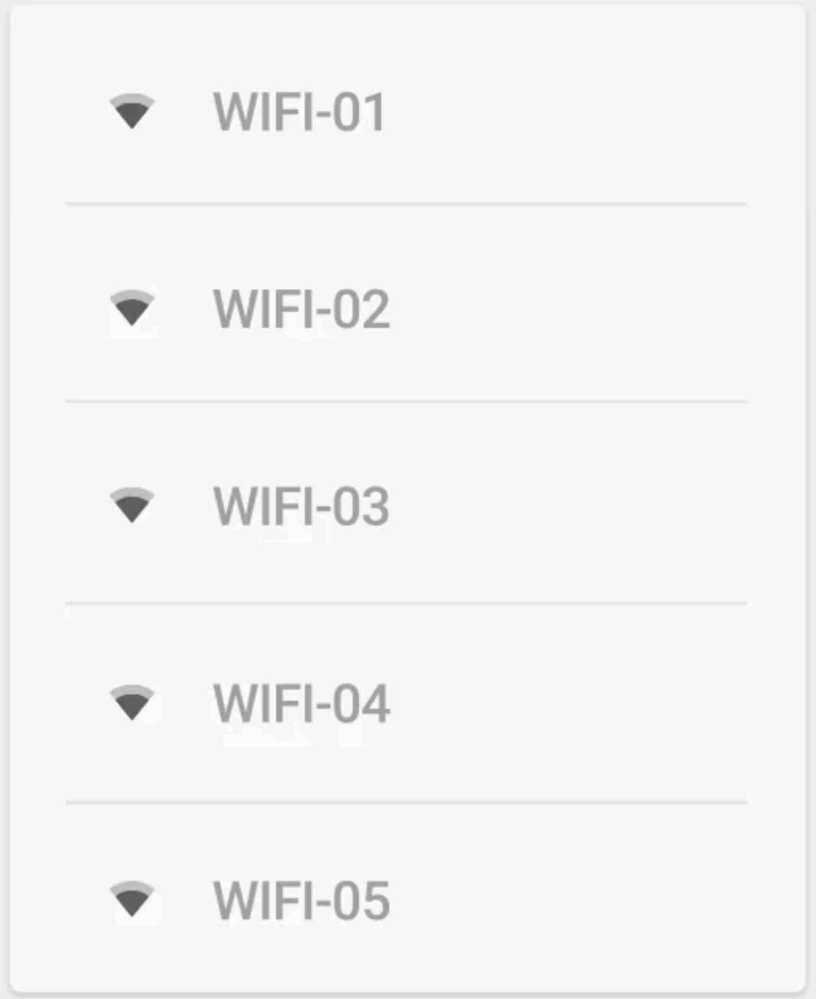
Widgets

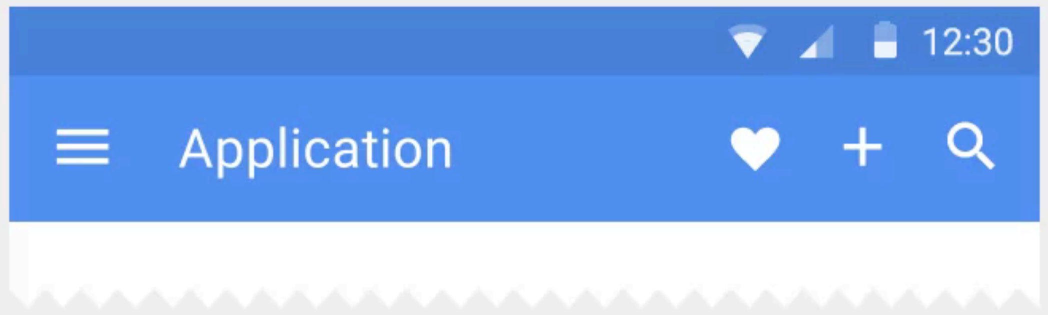
Now on iOS 8!



Animations







Others

- Phone vs tablet[1]
- Rotation handling
- Aspect ratios[1]
- No overscroll

---

[1] iOS 8 now brings iOS to a similar architecture

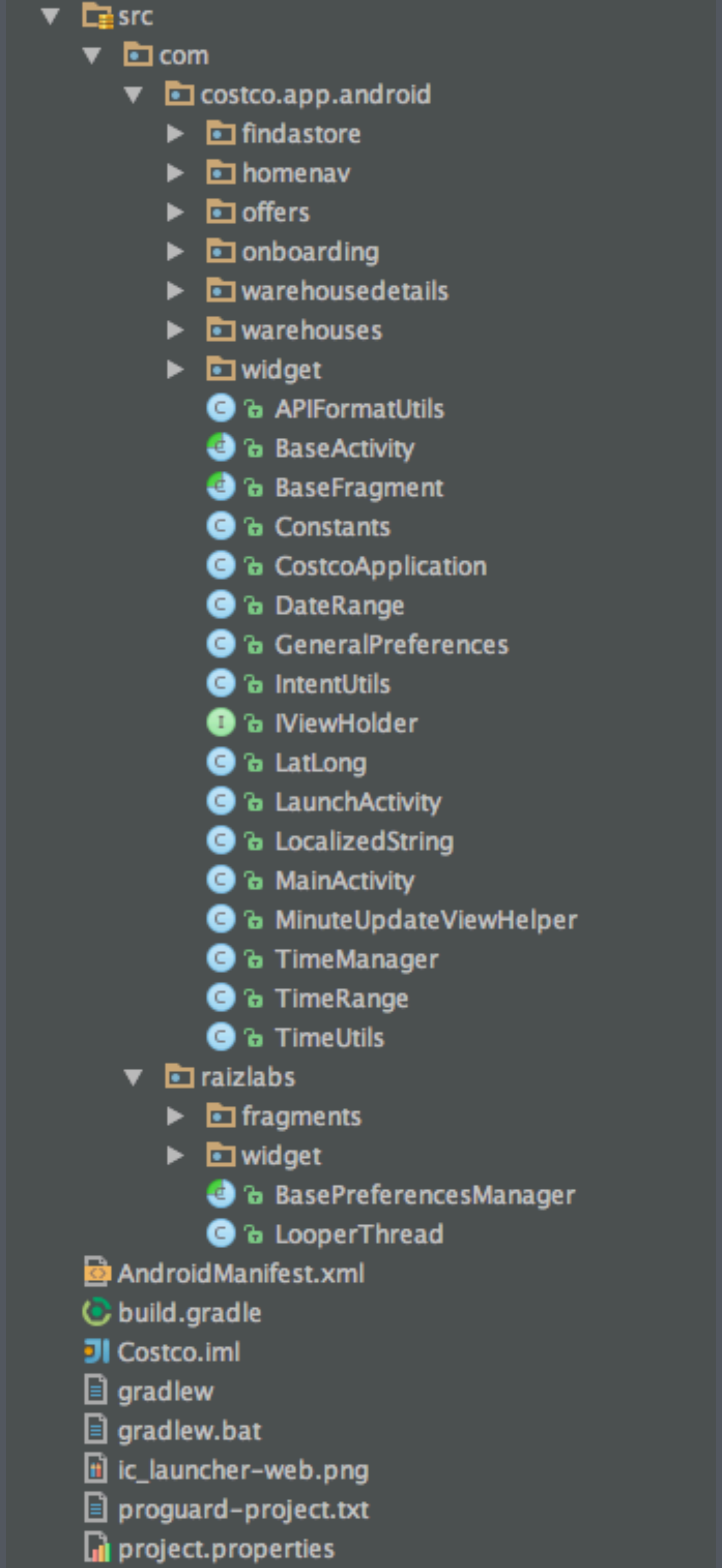
Code Styling and Patterns

- Leave class prefixes at home on Objective-C.
- Instance variables are prefixed with `m`, not _.
- Null check! Objective-C handles gracefully handles message sending to nil objects gracefully, but Java does not.
- Say goodbye to properties. You have to remember to call getVariableName() and setVariableName().

Misc

- Goodbye provisioning!
- So long review cycle!

Project Structure



```
▼ src
  ▼ com
    ▼ costco.app.android
      ▶ findastore
      ▶ homenav
      ▶ offers
      ▶ onboarding
      ▶ warehousedetails
      ▶ warehouses
      ▶ widget
        APIFormatUtils
        BaseActivity
        BaseFragment
        Constants
        CostcoApplication
        DateRange
        GeneralPreferences
        IntentUtils
        IViewHolder
        LatLong
        LaunchActivity
        LocalizedString
        MainActivity
        MinuteUpdateViewHelper
        TimeManager
        TimeRange
        TimeUtils
    ▼ raizlabs
      ▶ fragments
      ▶ widget
        BasePreferencesManager
        LooperThread
  AndroidManifest.xml
  build.gradle
  Costco.iml
  gradlew
  gradlew.bat
  ic_launcher-web.png
  proguard-project.txt
  project.properties
```

- Folder Structure based on package naming
- **AndroidManifest.xml** is required and similar to the **info.plist** on iOS
- **build.gradle** for each project

Drawables

drawable
drawable-hdpi
drawable-mdpi
drawable-xhdpi
drawable-xxhdpi

Drawables are images and any other renderable objects that you can define (think *XML* defined *CAShape*)

No more **@2x** or **@3x** - now you get many, many buckets!

drawable-mdpi, *drawable-hdpi*, *drawable-xhdpi*, *drawable-xxhdpi*, etc

Drawables are typically Images

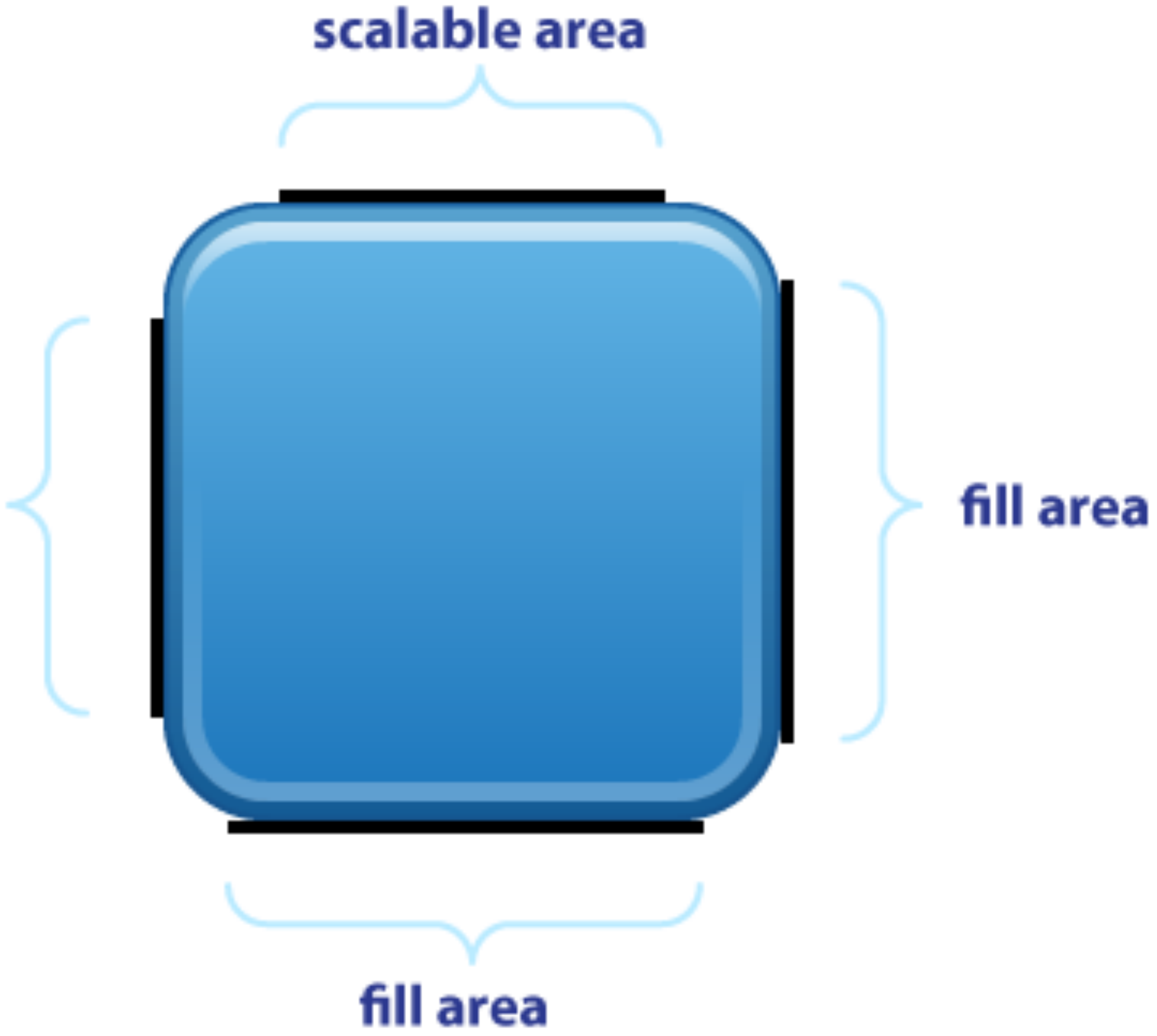
DPI (Dots Per Inch, or Density Independent Pixel)

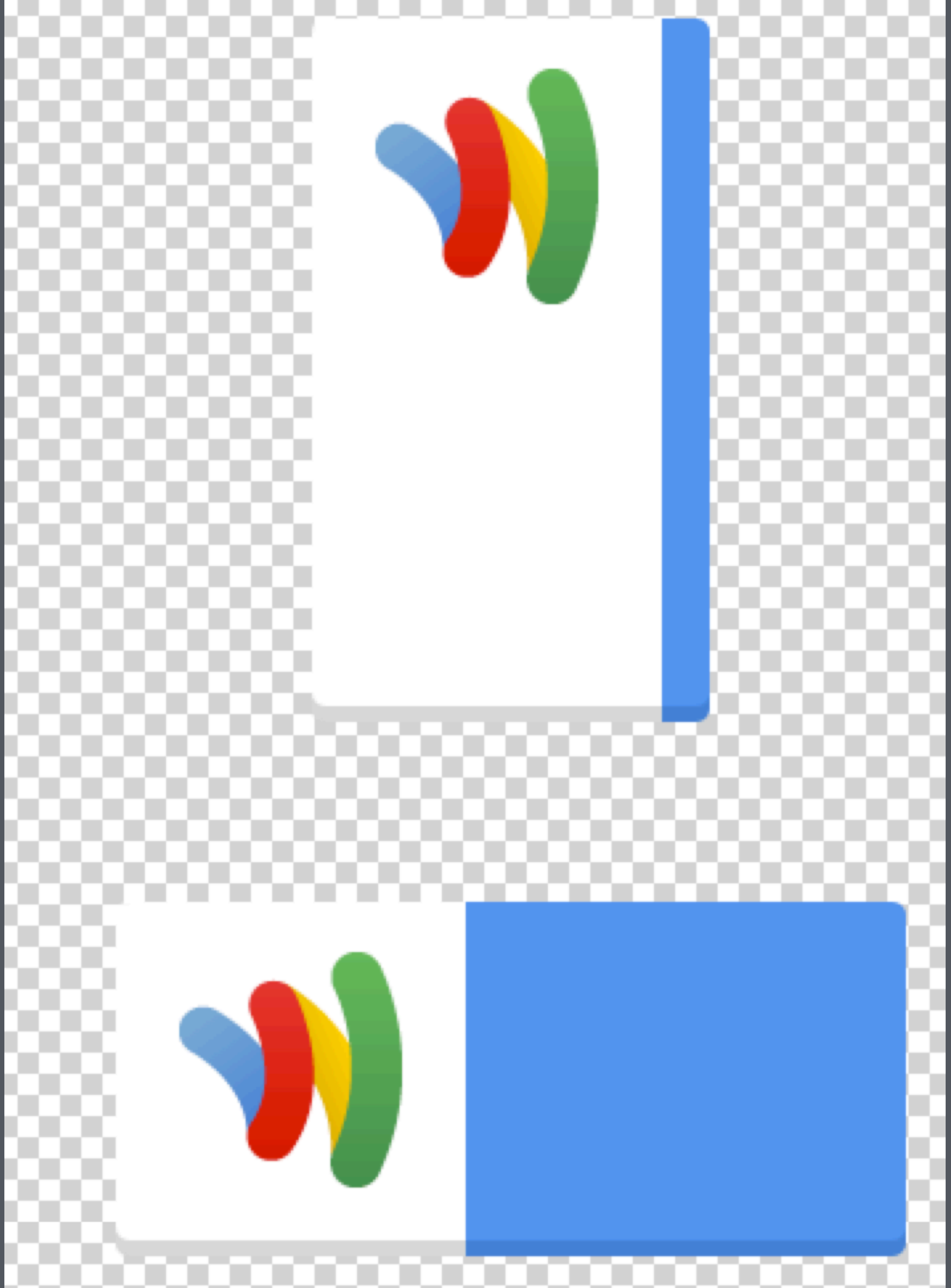
- Each device has an internal DPI bucket
- Android picks the resource from the corresponding bucket *or* the closet one and scales it

9-Patches (Stretchable Images)

9-patch guides

what do they do?





Dimens and Strings

```
<dimen name="Margin.Standard">5dp</dimen>
```

```
<string name="LocationServices.DisabledPrompt.Title">Location services disabled</string>
```

Styles

```
<style name="Button" parent="@style/TextStyle.Standard">
    <item name="android:textSize">@dimen/TextSize.Large</item>
    <item name="android:background">@color/Panel.White.Transparent</item>
    <item name="android:minWidth">@dimen/MinTouchableSize</item>
    <item name="android:minHeight">@dimen/MinTouchableSize</item>
</style>
```

You can have different values for different API levels!

Tools

Build Tools

- Gradle
- Ant

IDEs

- Android Studio (*built by IntelliJ*)
- Eclipse

Android Myths

Fragmentation

Use Google's Support Library

Android Apps Crash More

Android visibly notifies the user when and app fails to respond or crashes (even in the backbground).

Android Users Don't Use Their Devices As Much

Somewhat. The Android userbase is **huge**. **Really** huge. Some users don't use their devices much, but some use them a lot. The *user segment* you target means a lot.

Building Blocks

Activities, Fragments, ListViews, ViewPagers, and more

Activities <-> UIViewControllers

Activities are the basic unit of an Android app.

An application usually consists of multiple activities that are loosely bound to each other.

One activity in an application is specified as the "main" activity.

Activities can start and return information

- Activities can register that they handle common data such as images
- Activities can also recieve and send specific data, such as an item ID

Example: An Activity Launching Another Activity and Responding When It Is Done

```java
protected void startNextActivity() {
    Intent nextActivityIntent = NextActivity.getIntent(this);
    startActivityForResult(nextActivityResult, REQUEST_CODE_NEXT_ACTIVITY);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
    case REQUEST_CODE_NEXT_ACTIVITY:
        if (resultCode == RESULT_OK) {
            Toast.makeText(this, "Result OK!", Toast.LENGTH_SHORT).show();
            // We can also do something with returnObject within data here
        }
        return;
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

Next Activity Finishing with Data

```
public static final String activityResultString = "activityResultString";

/*
 * On completion, place the object ID in the intent and finish with OK.
 * @param returnObject that was processed
 */
private void onActivityResult(Object returnObject) {
    Intent data = new Intent();
    if (returnObject != null) {
        data.putExtra(activityResultString, returnObject.uniqueId);
    }

    setResult(RESULT_OK, data);
    finish();
}
```

iOS Aside

WatchKit and WKInterfaceController

WKInterfaceController's designated initializer now uses as `withContext` pattern to send information between interface controllers without custom overrides.

```
override init(context: AnyObject?) {
}
```

```
pushControllerWithName:context:
```

Fragments

Mini controllers that can be instantiated to populate activities

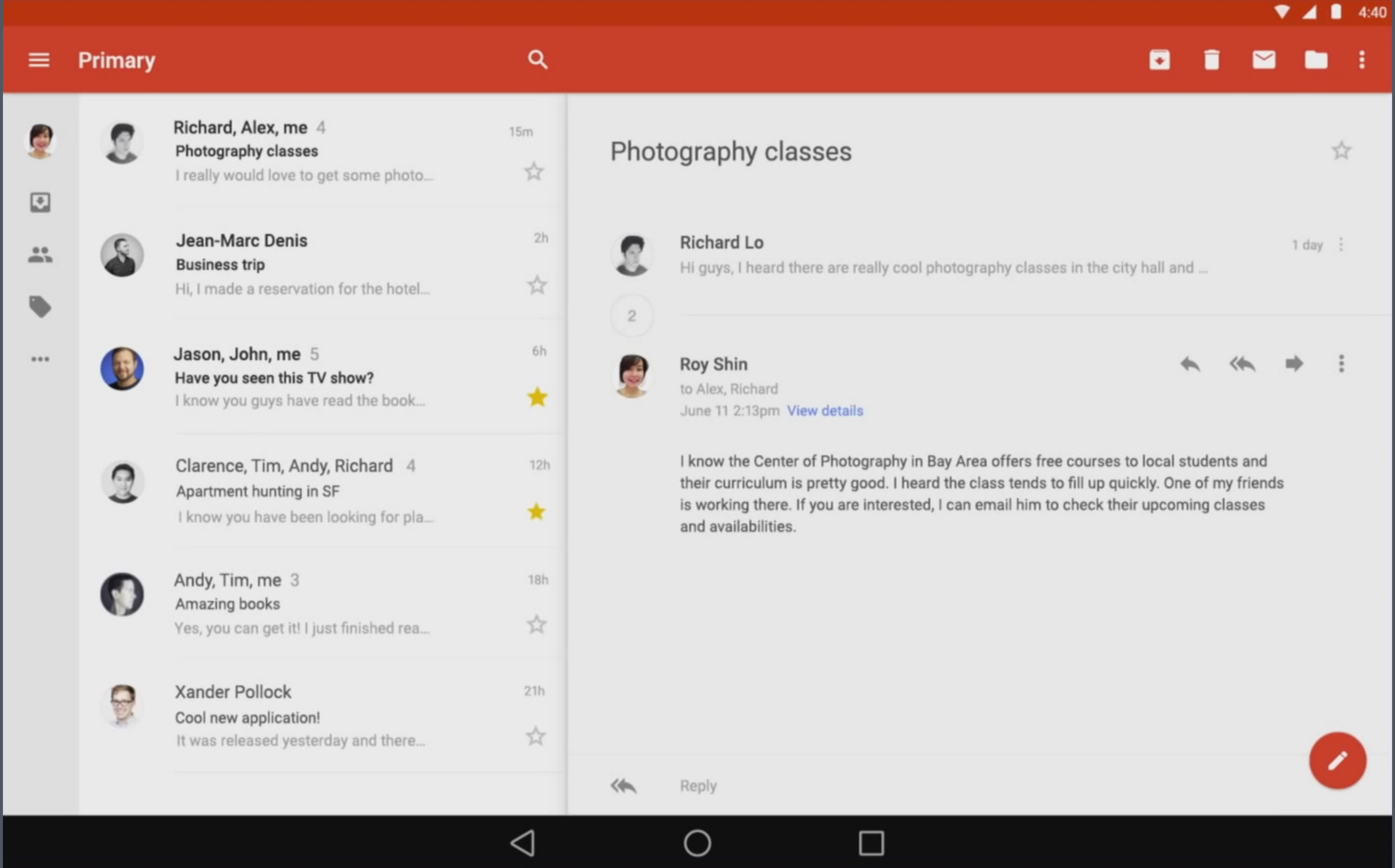
Fragments are the "new", accepted way to build Android apps[2]

[2] And they're awesome

There can be **one** or **multiple** fragments within a single activity.

Fragments:

- Store state
- Contain view logic
- Do not have a context, the activity has the context
- Must be tied to an activity



Richard, Alex, me 4 15m
Photography classes
I really would love to get some photo...

Jean-Marc Denis 2h
Business trip
Hi, I made a reservation for the hotel...

Jason, John, me 5 6h
Have you seen this TV show?
I know you guys have read the book...

Clarence, Tim, Andy, Richard 4 12h
Apartment hunting in SF
I know you have been looking for pla...

Andy, Tim, me 3 18h
Amazing books
Yes, you can get it! I just finished rea...

Xander Pollock 21h
Cool new application!
It was released yesterday and there...

Photography classes

Richard Lo 1 day
Hi guys, I heard there are really cool photography classes in the city hall and ...

2

Roy Shin
to Alex, Richard
June 11 2:13pm View details

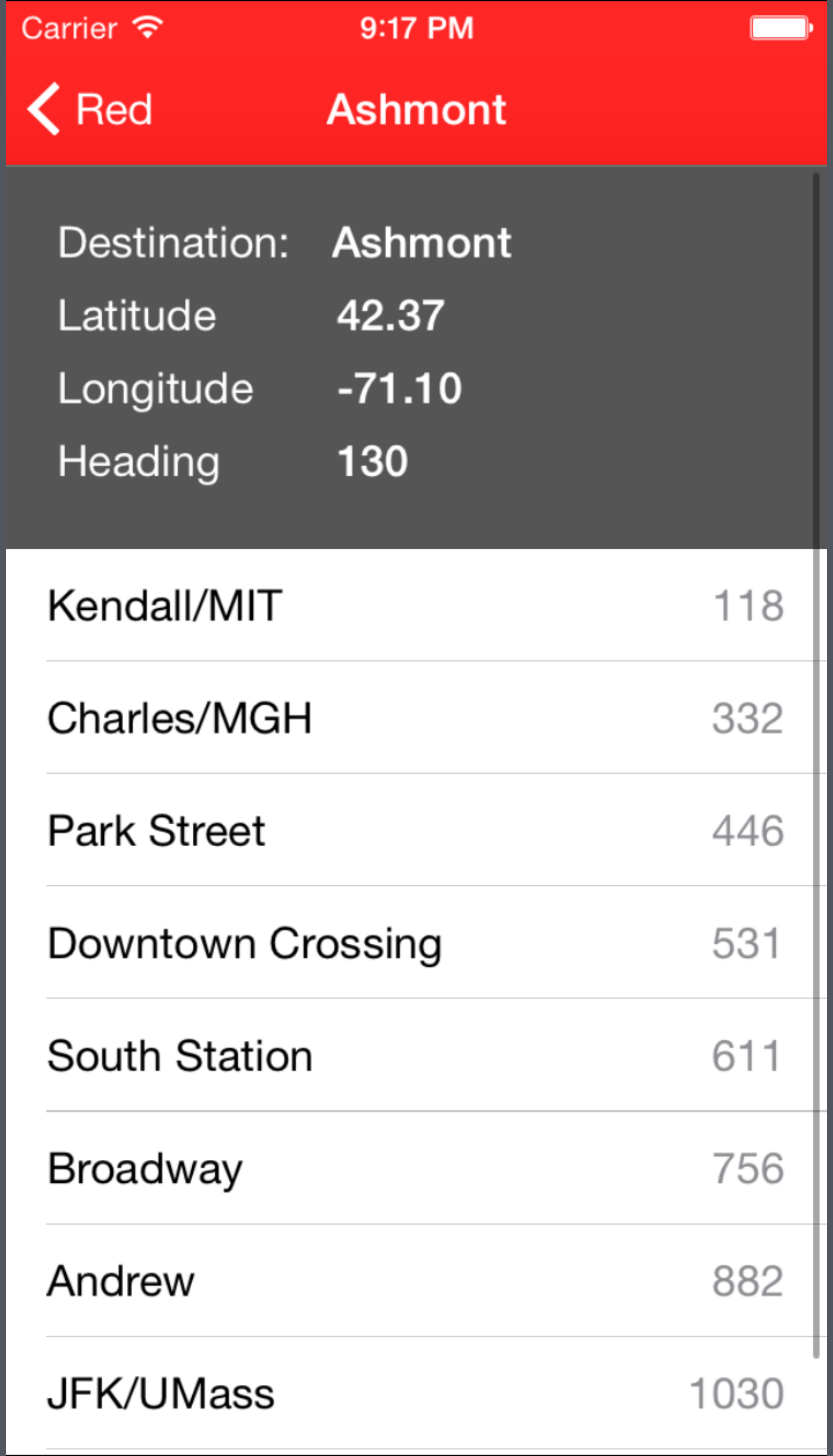
I know the Center of Photography in Bay Area offers free courses to local students and their curriculum is pretty good. I heard the class tends to fill up quickly. One of my friends is working there. If you are interested, I can email him to check their upcoming classes and availabilities.

The closest approximation for fragments in iOS is using child view controllers.

Explain it to me with code

Let's look at a sample **UITableViewController** and a sample **ListFragment** that show a list of prediction times for a subway trip curtesy of the MBTA (Massachusetts Bay Transportation Authority).

iOS Tableview Implementation



Carrier 9:17 PM

Red Ashmont

Destination:	Ashmont
Latitude	42.37
Longitude	-71.10
Heading	130

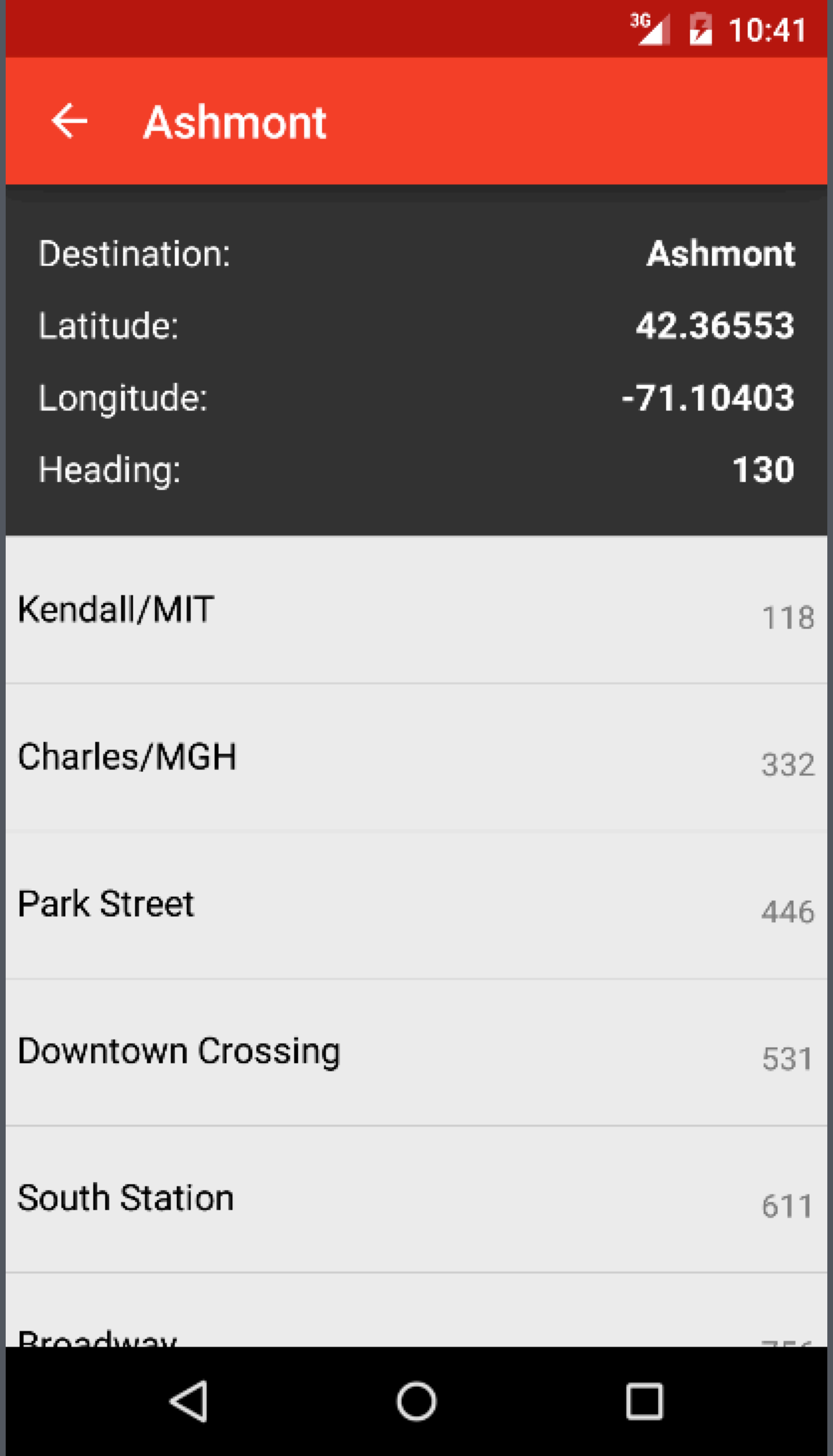
Kendall/MIT	118
Charles/MGH	332
Park Street	446
Downtown Crossing	531
South Station	611
Broadway	756
Andrew	882
JFK/UMass	1030

```objc
@interface MBTASubwayTripTableTableViewController ()
@property (assign, nonatomic) MBTATrip *trip;
@end

@implementation MBTASubwayTripTableTableViewController
- (instancetype)initWithTrip:(MBTATrip *)trip
{
    self = [super initWithStyle:UITableViewStylePlain];
    if (self) {
        _trip = trip;
        [self setTitle:trip.destination];
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self.tableView registerClass:[MBTAPredictionCell class]
        forCellReuseIdentifier:[MBTAPredictionCell reuseId]];
    [self.tableView
        registerNib:[UINib nibWithNibName:NSStringFromClass([MBTATripHeaderView class]) bundle:nil]
        forHeaderFooterViewReuseIdentifier:[MBTATripHeaderView reuseId]];
}
```

```
#pragma mark - Table view data source
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return [self.trip.predictions count];
}
```

```
#pragma mark - UITableViewDelegate
- (CGFloat)tableView:(UITableView *)tableView heightForHeaderInSection:(NSInteger)section {
    return [MBTATripHeaderView heightWithTrip:self.trip];
}

- (UIView *)tableView:(UITableView *)tableView viewForHeaderInSection:(NSInteger)section {
    MBTATripHeaderView *headerView = [self.tableView
         dequeueReusableHeaderFooterViewWithIdentifier:[MBTATripHeaderView reuseId]];
    [headerView setFromTrip:self.trip];
    return headerView;
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView
        dequeueReusableCellWithIdentifier:[MBTAPredictionCell reuseId]
        forIndexPath:indexPath];

    MBTAPrediction *prediction = [self.trip.predictions objectAtIndex:indexPath.row];
    [(MBTAPredictionCell *)cell setFromPrediction:prediction];

    return cell;
}

- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath {
    return NO;
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    [tableView deselectRowAtIndexPath:indexPath animated:YES];
}
@end
```

ListFragment Implementation


10:41

Ashmont

Destination:	Ashmont
Latitude:	42.36553
Longitude:	-71.10403
Heading:	130

Kendall/MIT	118
Charles/MGH	332
Park Street	446
Downtown Crossing	531
South Station	611
Broadway	

```java
public class TripDetailFragment extends ListFragment {
    /**
     * The configuration flags for the Trip Detail Fragment.
     */
    public static final class TripDetailFragmentState {
        public static final String KEY_FRAGMENT_TRIP_DETAIL = "KEY_FRAGMENT_TRIP_DETAIL";
    }

    protected Trip mTrip;

    /**
     * Use this factory method to create a new instance of
     * this fragment using the provided parameters.
     *
     * @param trip the trip to show details
     * @return A new instance of fragment TripDetailFragment.
     */
    public static TripDetailFragment newInstance(Trip trip) {
        TripDetailFragment fragment = new TripDetailFragment();
        Bundle args = new Bundle();
        args.putParcelable(TripDetailFragmentState.KEY_FRAGMENT_TRIP_DETAIL, trip);
        fragment.setArguments(args);
        return fragment;
    }

    public TripDetailFragment() { }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        Prediction[] predictions =
             mTrip.predictions.toArray(new Prediction[mTrip.predictions.size()]);

        PredictionArrayAdapter predictionArrayAdapter =
             new PredictionArrayAdapter(getActivity().getApplicationContext(), predictions);

        setListAdapter(predictionArrayAdapter);
        return super.onCreateView(inflater,container, savedInstanceState);
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        TripDetailsView headerView = new TripDetailsView(getActivity());
        headerView.updateFromTripObject(mTrip);
        getListView().addHeaderView(headerView);
    }
}
```

WAT?

Let's break down some of that piece by piece...

Listviews and Adapters

UITableView, meet ListView

Both are sturctured around showing a linear list of Views smoothly

- Android doesn't have *cells* - instead any view can be used
- Reuse your views in a ListView! *This can be even more important to performance than on iOS*
- Default ListView views are available that you can populate just like iOS has default UITableViewCells

Android's **GridView** is similar to **UICollectionView**

Listviews are populated via **Adapters**

Goodbye datasources, hello adapters

Instead of having a datasource delegate, Android has **Adapaters**

- An Adapter object acts as a bridge between an AdapterView and the underlying data for that view.
- The Adapter provides access to the data items.
- The Adapter is also responsible for making a View for each item in the data set.

```java
public class PredictionArrayAdapter extends ArrayAdapter<Prediction> {
    int LAYOUT_RESOURCE_ID = R.layout.view_three_item_list_view;

    public PredictionArrayAdapter(Context context) {
        super(context, R.layout.view_three_item_list_view);
    }

    public PredictionArrayAdapter(Context context, Prediction[] objects) {
        super(context, R.layout.view_three_item_list_view, objects);
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Prediction prediction = this.getItem(position);
        View inflatedView = convertView;
        if(convertView==null)
        {
            LayoutInflater inflater = (LayoutInflater)getContext()
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            inflatedView = inflater.inflate(LAYOUT_RESOURCE_ID, parent, false);
        }

        TextView stopNameTextView = (TextView)inflatedView
            .findViewById(R.id.view_three_item_list_view_left_text_view);
        TextView middleTextView = (TextView)inflatedView
            .findViewById(R.id.view_three_item_list_view_middle_text_view);
        TextView stopSecondsTextView = (TextView)inflatedView
            .findViewById(R.id.view_three_item_list_view_right_text_view);

        stopNameTextView.setText(prediction.stopName);
        middleTextView.setText("");
        stopSecondsTextView.setText(prediction.stopSeconds.toString());

        return inflatedView;
    }
}
```

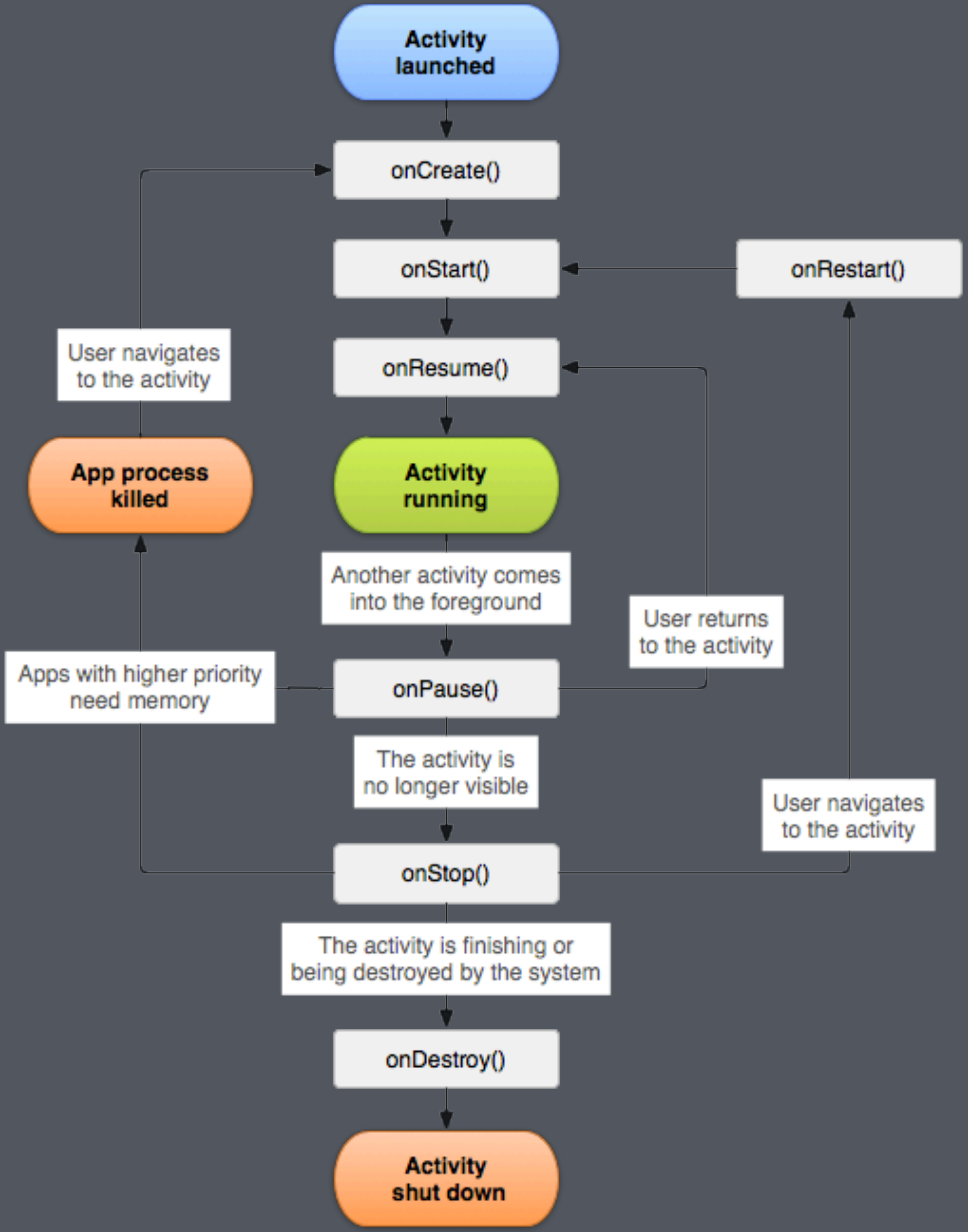
AsyncTasks

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
     protected Long doInBackground(URL... urls) {
         int count = urls.length;
         long totalSize = 0;
         for (int i = 0; i < count; i++) {
             totalSize += Downloader.downloadFile(urls[i]);
             publishProgress((int)((i '/' (float) count) * 100));
             // Escape early if cancel() is called
             if (isCancelled()) {
                break;
            }
         }
         return totalSize;
     }

     protected void onProgressUpdate(Integer... progress) {
         setProgressPercent(progress[0]);
     }

     protected void onPostExecute(Long result) {
         showDialog("Downloaded " + result + " bytes");
     }
 }
```

User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()

Activity Lifecycle

A harsh mistress


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

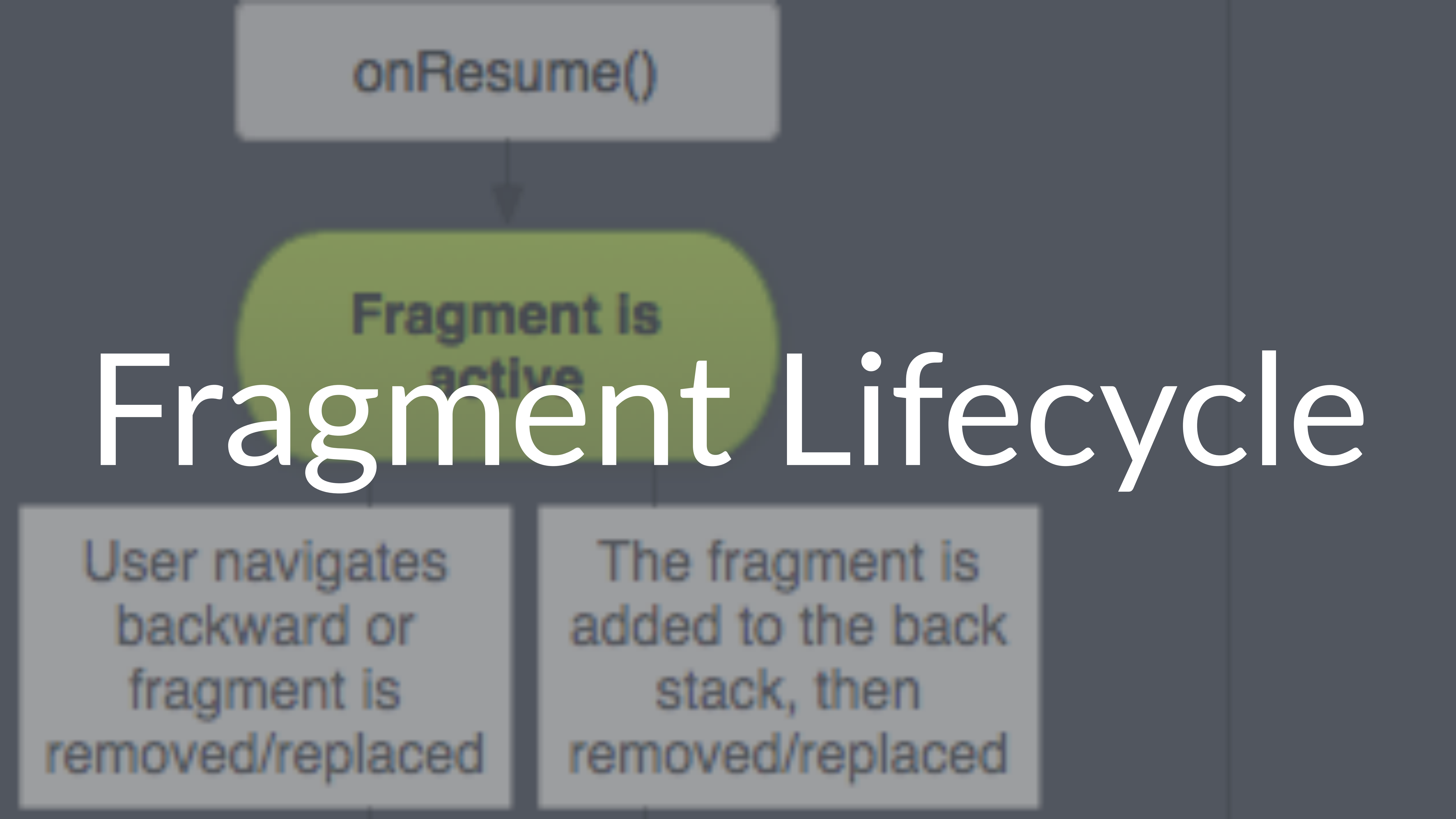
Activities must save their state

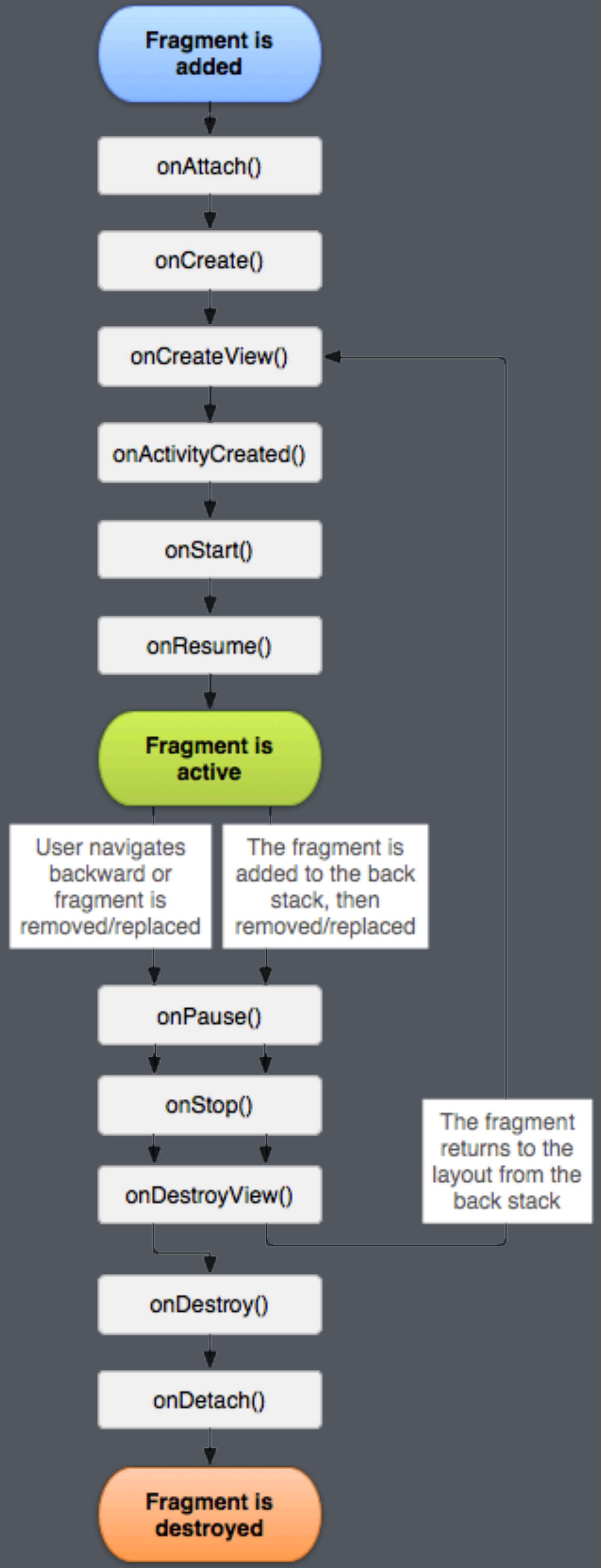
- Place information into the savedInstanceState that you wish to keep and restore it on *onCreate()*
- Activities can be **Destroyed** on rotation
    - Dont turn off rotation - it will just hide edge case lifecycle bugs if you do
- *Activities* and *Fragments* can be loosely coupled. On creation, look for initialized *Fragments* before creating them.

Example: Storing Activity Data

```java
public static Intent getTripListActivityIntent(Context context,
                                               TripList.LineType lineType) {
    Intent intent = new Intent(context, TripListActivity.class);
    intent.putExtra(TripListActivityState.KEY_ACTIVITY_TRIP_LIST_LINE_TYPE,
        lineType.getLineName());
    return intent;
}

public static final class TripListActivityState {
    public static final String KEY_ACTIVITY_TRIP_LIST_LINE_TYPE =
        "KEY_ACTIVITY_TRIP_LIST_LINE_TYPE";
}

TripList.LineType mLineType;

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   mLineType = TripList.LineType.getLineType(
        getIntent().getStringExtra(TripListActivityState.KEY_ACTIVITY_TRIP_LIST_LINE_TYPE));
}
```

Fragment Lifecycle

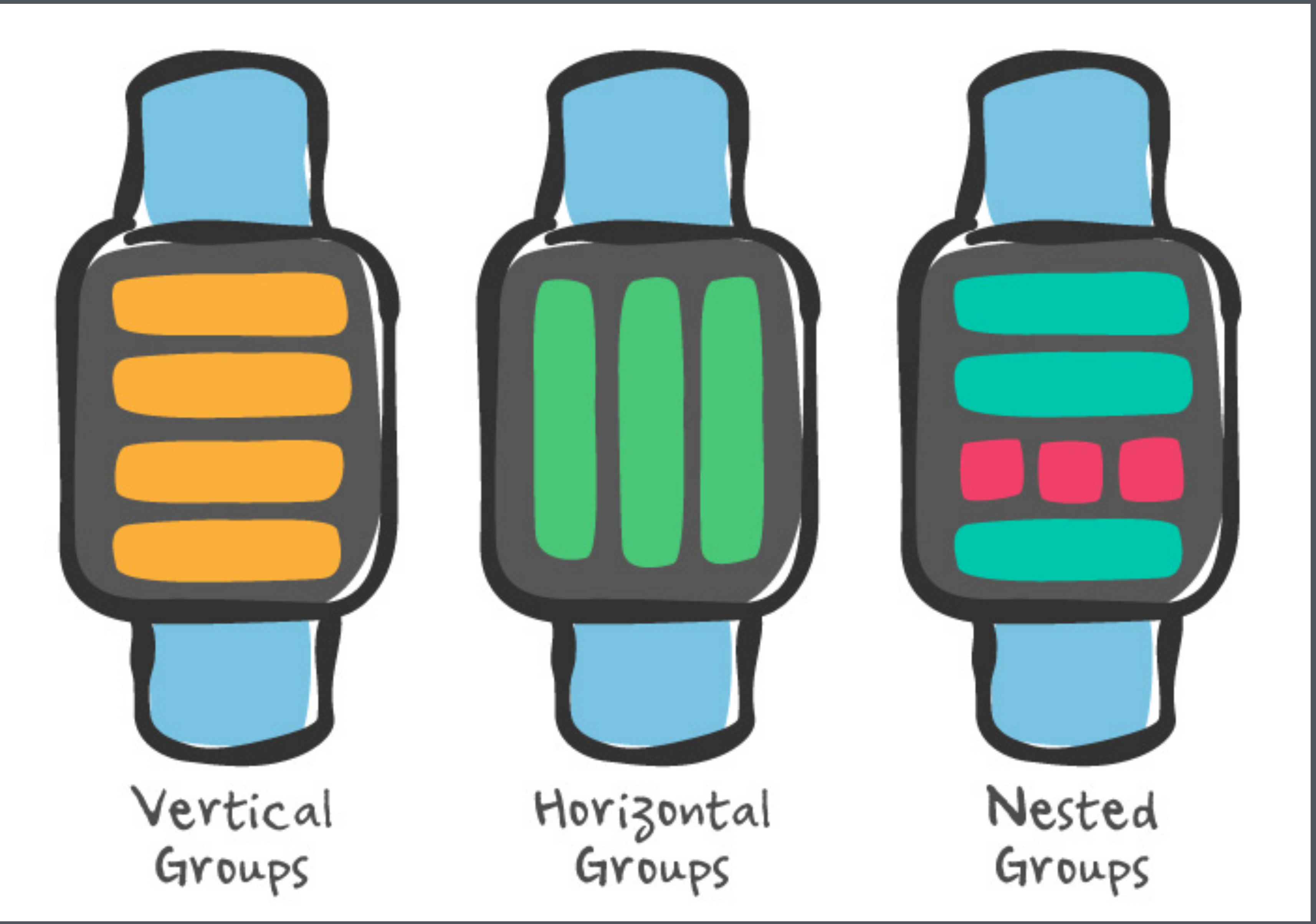

onResume()
Fragment is active
User navigates backward or fragment is removed/replaced
The fragment is added to the back stack, then removed/replaced


Fragment is added
onAttach()
onCreate()
onCreateView()
onActivityCreated()
onStart()
onResume()
Fragment is active
User navigates backward or fragment is removed/replaced
The fragment is added to the back stack, then removed/replaced
onPause()
onStop()
onDestroyView()
The fragment returns to the layout from the back stack
onDestroy()
onDetach()
Fragment is destroyed

Fragments must save their state

- Fragments have their *own* savedInstanceState you must use as well.
- Fragments can have *Bundled Arguments* - this can be a good place to store state data.
- Fragments are *created* **before** *activities* sometimes.
    - Many fragments have *Listeners* (aka delegates) to their Activities. Don't try to use or reference these before *onActivityCreated()**.

iOS is Heading More and More Towards Android's Layout Structure



WatchKit, Groups, and No Autolayout

Watchkit introduces groups, which are very similar to Android's ViewGroups

Android Layout Basics

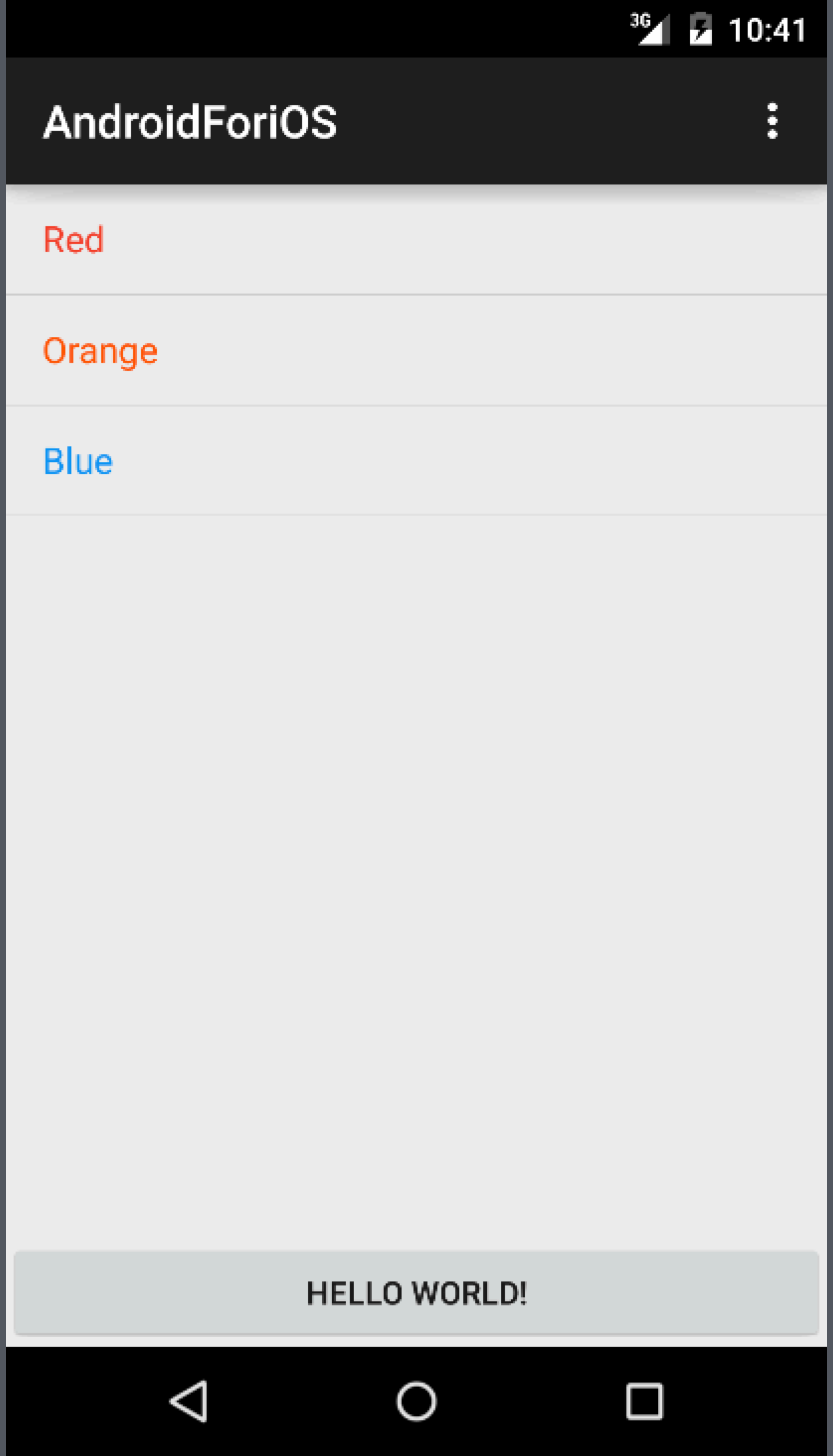
- Declare UI elements in XML
- Instantiate layout elements at runtime

Width and Height

All elements require a width and a height. Typically, either:

- match_parent
- wrap_content

Subway ListView Layout



```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.androidforios.app.activities.MainActivity$PlaceholderFragment">

    <ListView
        android:id="@+id/fragment_subway_list_listview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingBottom="@dimen/Button.Default.Height"/>

    <Button
        android:id="@+id/fragment_subway_list_Button"
        android:layout_width="match_parent"
        android:layout_height="@dimen/Button.Default.Height"
        android:minHeight="@dimen/Button.Default.Height"
        android:background="@drawable/button_red_selector"
        android:text="@string/hello_world"
        android:textColor="@color/Button.Text"
        android:layout_alignParentBottom="true"
        android:gravity="center"/>
</RelativeLayout>
```

Main Layout ViewGroups

- RelativeLayout
- LinearLayout
- FrameLayout

Layout Tips

Instead of Press States, You Have Selectors

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true" android:drawable="@color/Button.Selected" />
    <item android:drawable="@color/Button.Background" />
</selector>
```

- Always work in dp (Density-independent Pixels) instead of pixels directly.
- Don't bother nudging items in the visual editor for layouts. Your best bet is to adjust the xml directly. Sorry, this is no interface builder.
- Design your layouts to handle different sizes!

Data

Options

- Shared Preferences <-> **NSUserDefaults**
- In memory objects
- Saving to and fetching from file structure via the internal or external file storage <-> saving to the documents directory
- **SQLLite** <-> **Core Data**

Homework

Android Homework

- Learn the UI: Action Bar, Overflow Menu, and the Menu Button
- Cross App Data Sharing
- Intents and Responding to common OS actions
- Java's features: Generics, virtual methods and classes
- Google's Compatability Libraries
- The Android Emulator: Install the x86 HAXM plugin to make the emulator buttery smooth.

Popular Libraries

- **Volley** <-> **AFNetworking**
- **ActiveAndroid** <-> **MagicalRecord**
- **Picasso** <-> **SDWebImage**
- **Google Support Library**

Go Out and Build Something Cross Platform!