Working with Navigation Drawer

Jim Wilson
MOBILE SOLUTIONS DEVELOPER & ARCHITECT

@hedgehogjim blog.jwhh.com

What to Expect from This Module

- Navigation Drawer Behavior and Usage
- Navigation Drawer Classes
- NavigationView
- NavigationView Resources
- NavigationView Selection Handling

Navigation Drawer

Provides app's main navigation options

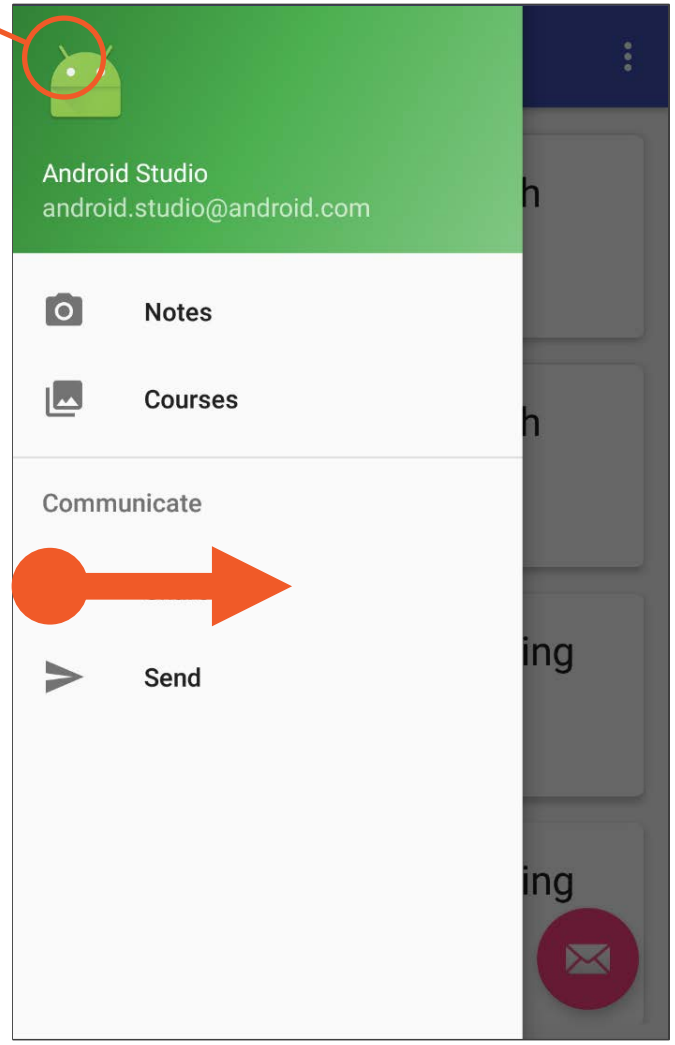
- Hidden when not in use
- Normally slides from screen's "start" edge
  - Left edge for most devices

Accessing navigation drawer

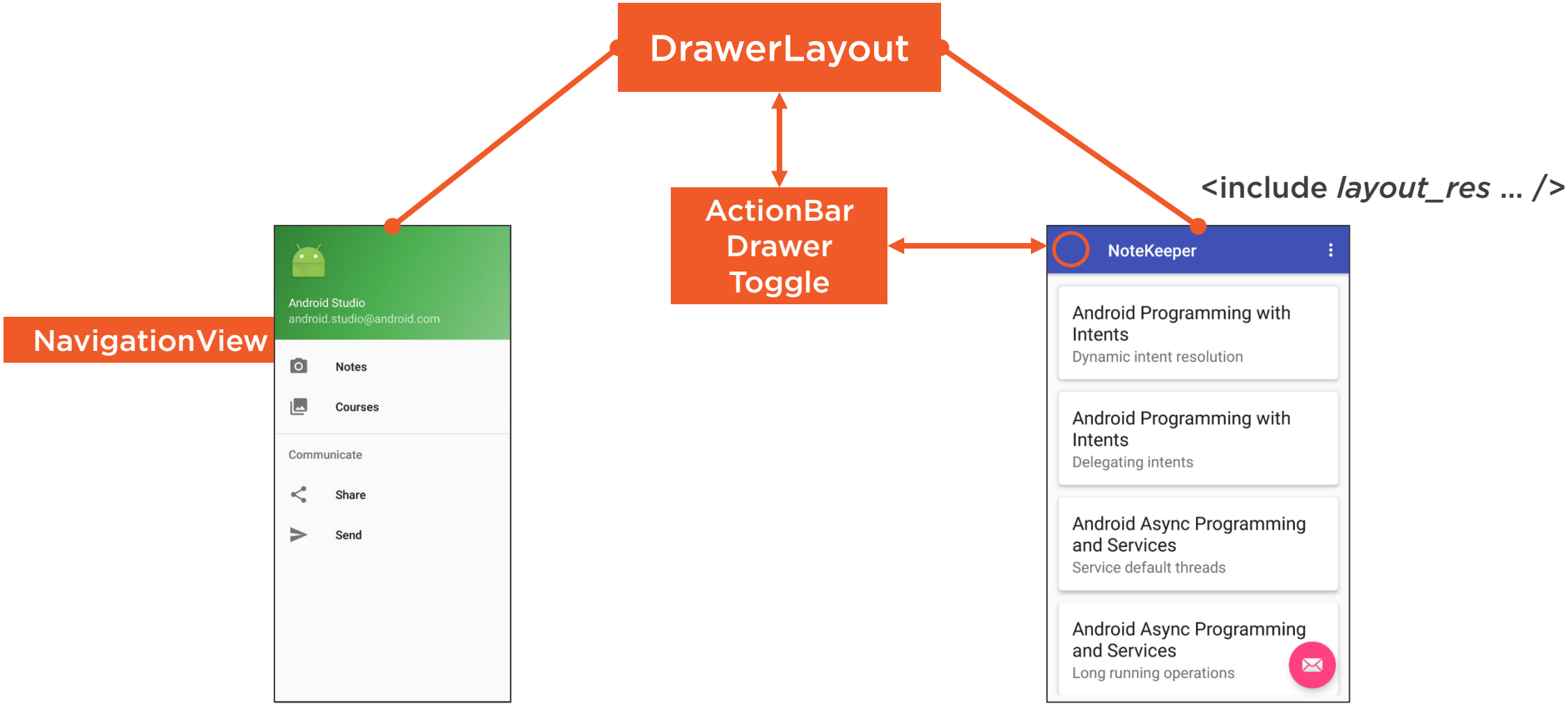
- User can open with app icon
- User can swipe left edge in/out

Navigation Drawer

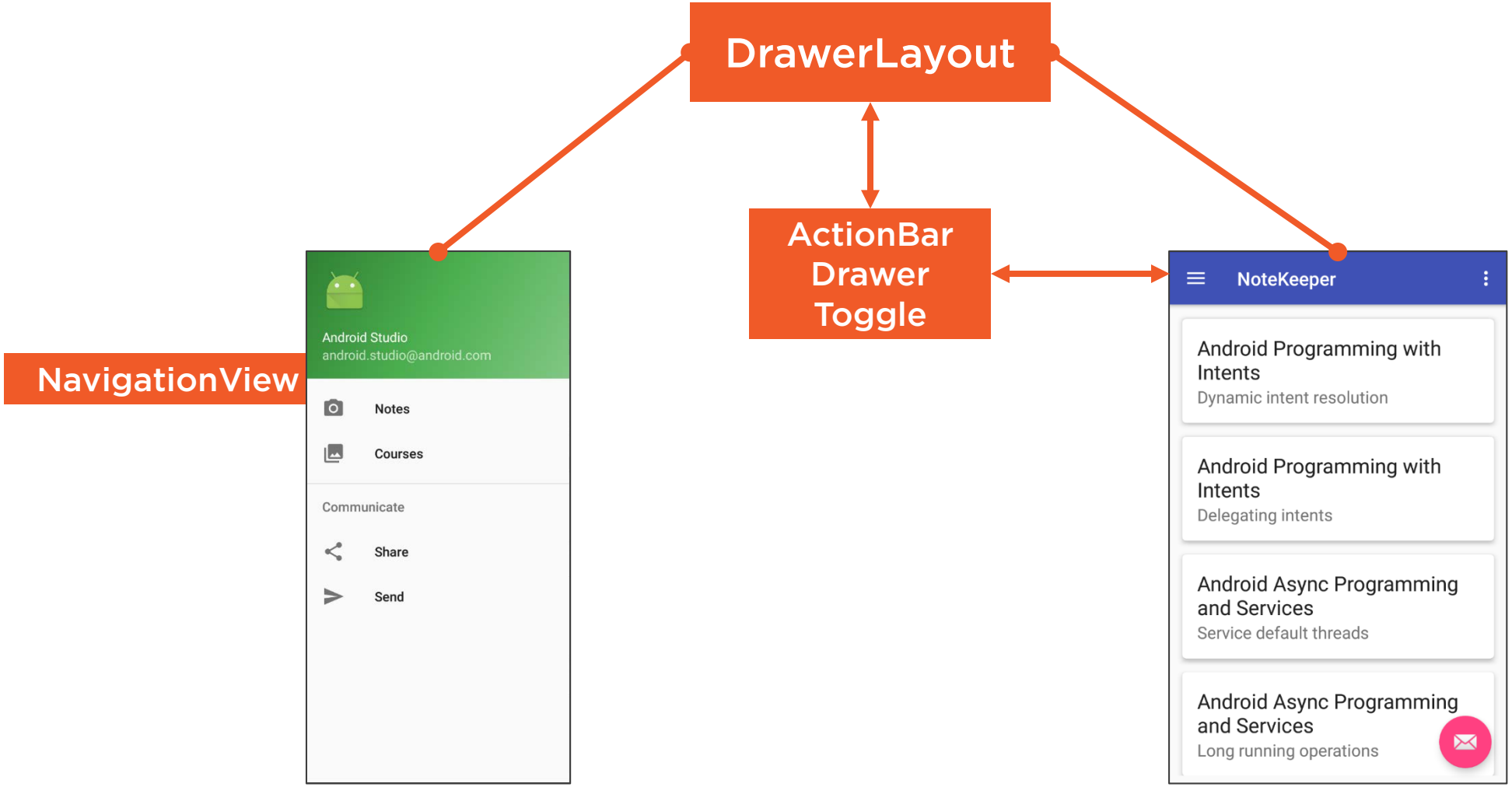
App Icon



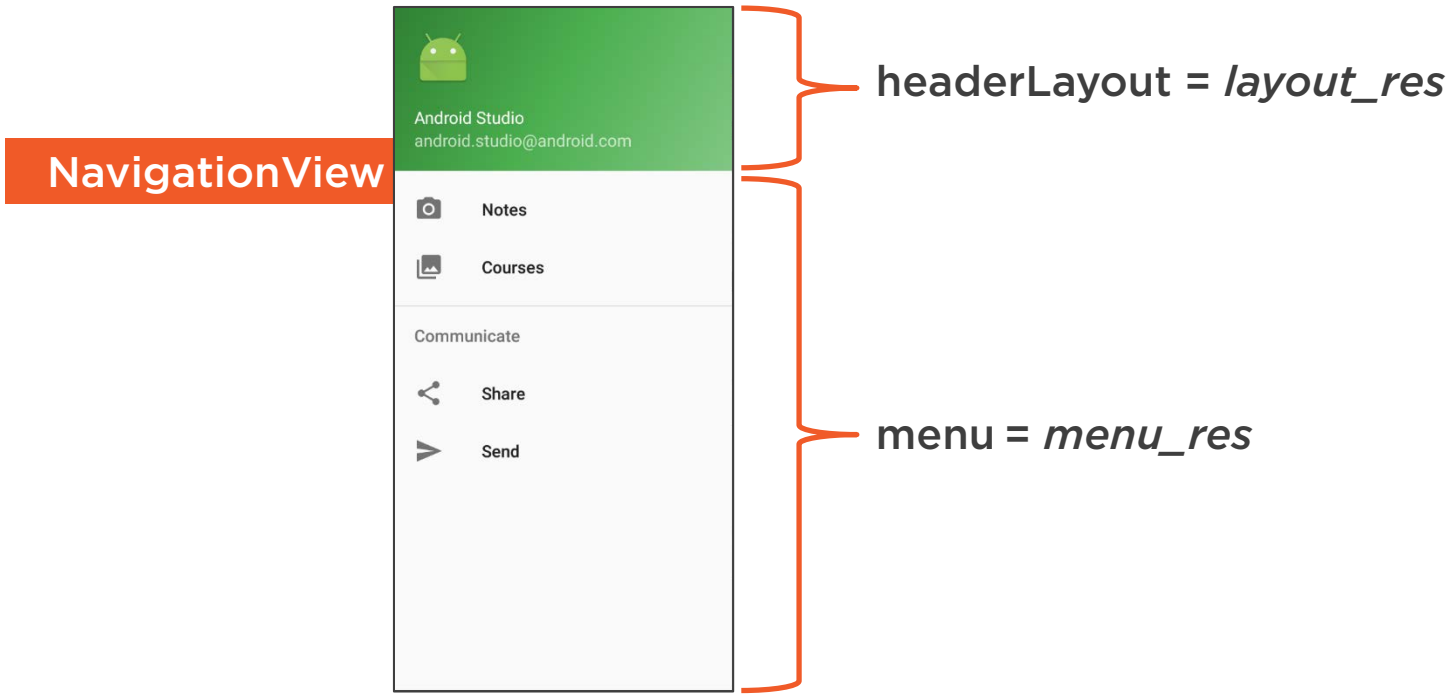
Navigation Drawer



Navigation Drawer NavigationView



Navigation Drawer NavigationView



Handling NavigationView Selections

Implement NavigationView listener interface

- onNavigationItemSelectedListener
- One method onNavigationItemSelected
  - Receives MenuItem reference

Associate with NavigationView

- setNavigationItemSelectedListener

Summary

Navigation Drawer

- Provides app main navigation options
- Normally slides in from “start” edge
  - Left edge on most devices

Opening/Closing Navigation Drawer

- Swipe left edge in/out
- Open with app icon
- Close with back button

Summary

DrawerLayout

- Navigation drawer layout management
- Area for primary content
- Area for pull-out drawer content

ActionBarDrawerToggle

- Provides app icon behavior

Summary

NavigationView

- Serves as the navigation drawer
- Layout area
- Menu area

NavigationView option handling

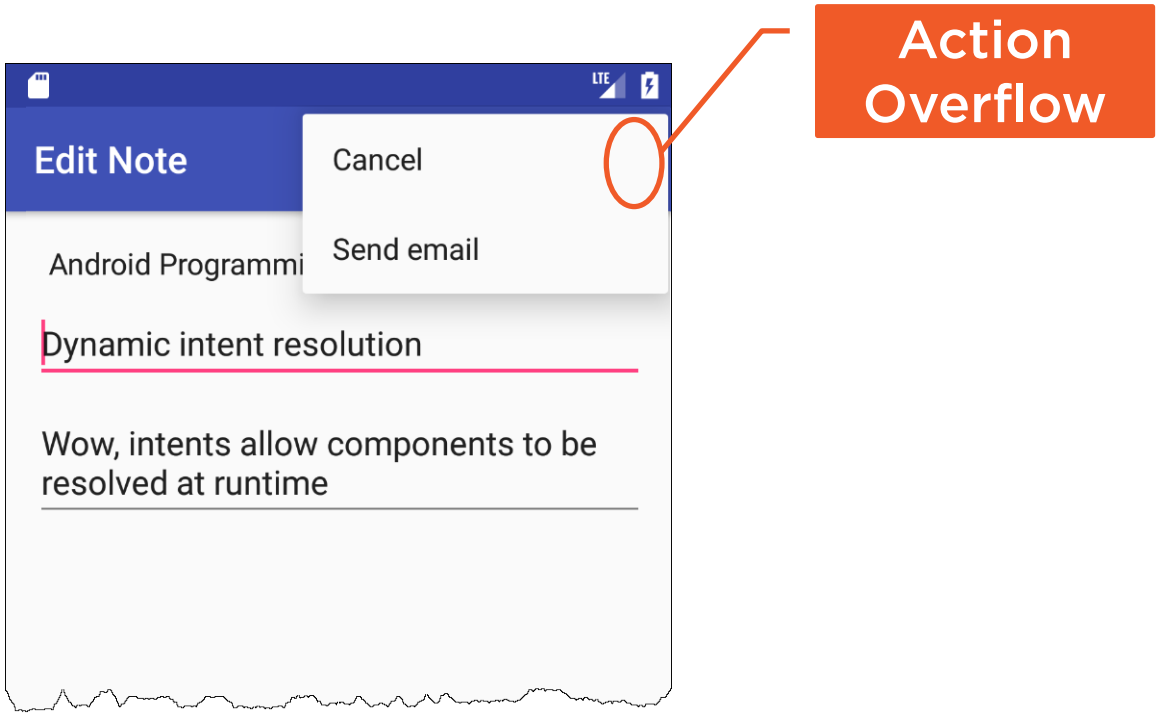
- Implement NavigationView interface
  - onNavigationSelectedListener
- Associate with NavigationView
  - setNavigationItemSelectedListener

Summary

Options Menus

Provide actions for an Activity

- Actions available in app bar
- Appear under action overflow by default



Options Menus

Defined in menu resource

- Root of component tree is menu

Each action defined as a menu item

- Has unique ID within menu
- Has a text title

Associating an Options Menu with an Activity

onCreateOptionsMenu

- Receives a menu reference
- Attach menu items to menu
- Inflate menu resource with menu inflater
  - Access with getMenuInflater method

Handling Options Menu Item Selections

onOptionsItemSelected

- Receives MenuItem reference
- Retrieve MenuItem ID value
  - Access with MenuItem.getItemId
- Perform work based on ID value

Menu Items as App Bar Actions

Action overflow menu item challenges

- Not immediately discoverable
- Access takes multiple steps

App bar actions

- Menu items visible on app bar
- Improve access to common menu items
- Normally have icon associated

Making menu item an app bar action

- Use showAsAction property

Common ShowAsAction Values

ifRoom

- Display as action when space allows
- Given preference in top-to-bottom order

always

- Always display as action
- Use very sparingly

withText

- Show text with action when space allows
- Can be combined with ifRoom or always

Changing Menu Items at Runtime

Application state can change menu options

- May need to add/remove menu items
- May need to enable/disable menu items

Changing Menu Items at Runtime

onPrepareOptionsMenu

- Override to modify menu state
- Receives reference to current menu
- Initially called before menu displayed

invalidateOptionsMenu

- Call when menu state may need to change
- Schedules call to onPrepareOptionsMenu

RecyclerView Adapter

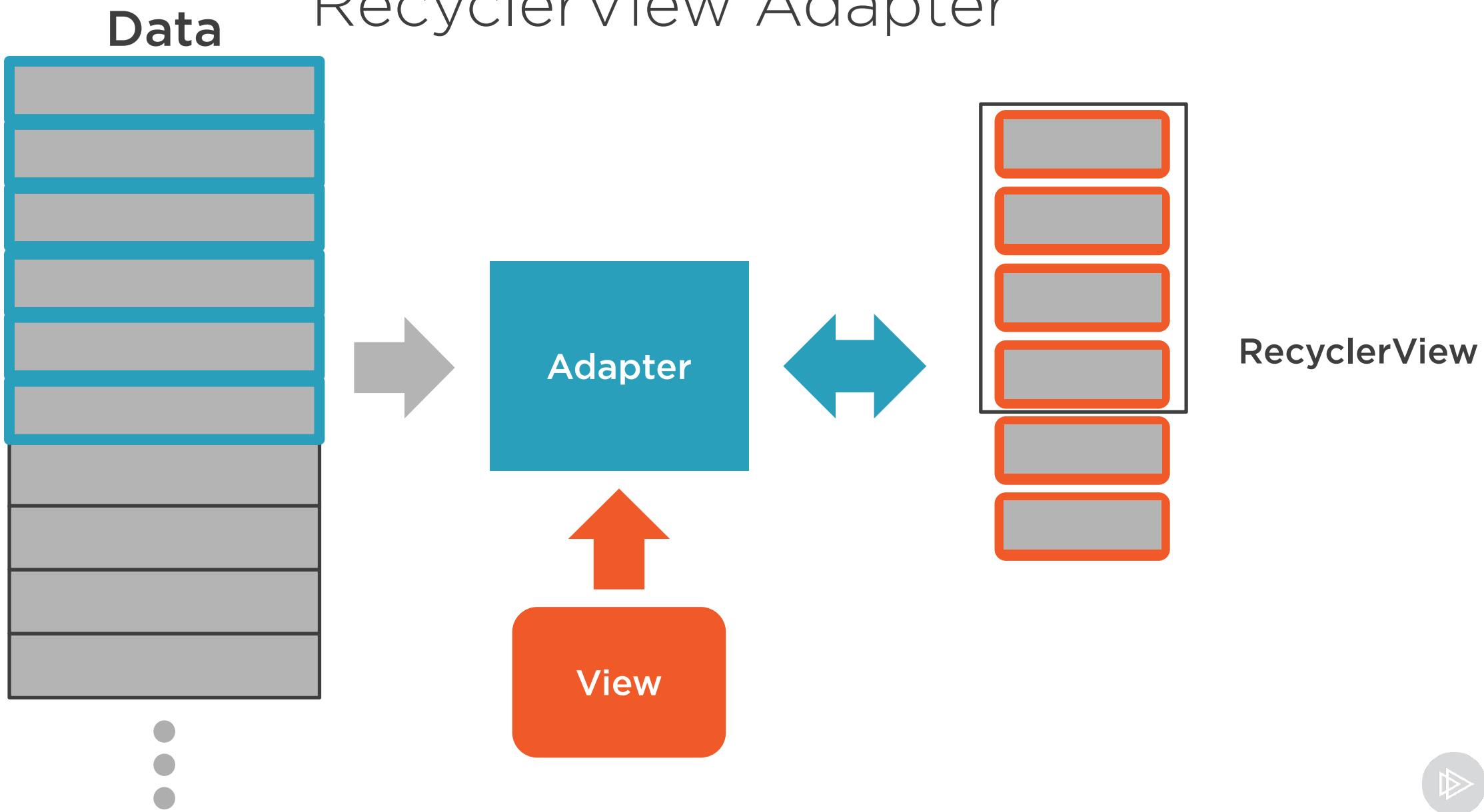
Create item views

- Returned to RecyclerView
- RecyclerView manages as a pool

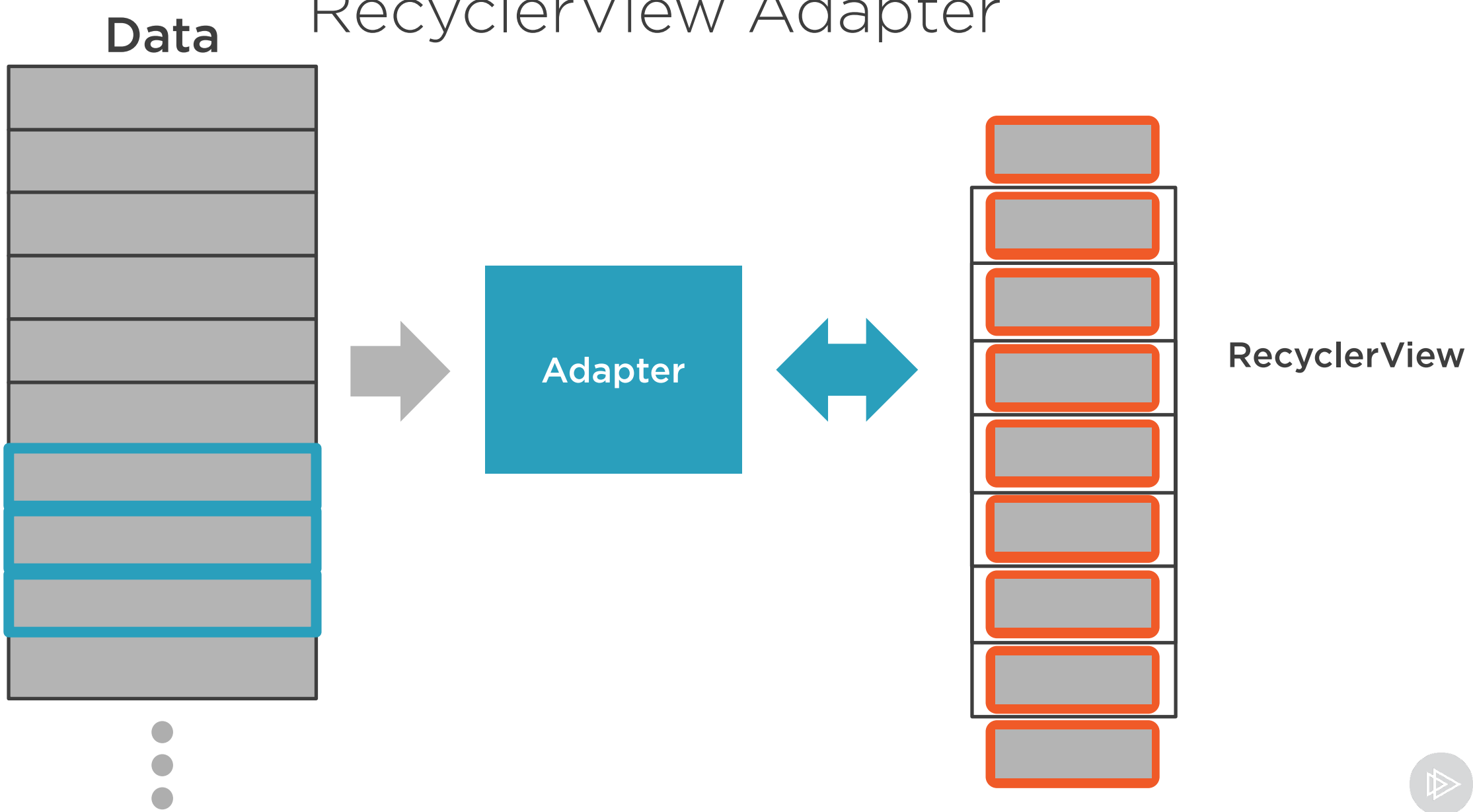
Populate item views

- Received from RecyclerView pool

RecyclerView Adapter



RecyclerView Adapter



Item View Holders

Views managed using view holder pattern

- Holds a reference to top-level view
- Holds references to contained views

Use a custom view holder class

- Extends RecyclerView.ViewHolder
- Fields for contained views

Implementing RecyclerView Adapter

Extend RecyclerView.Adapter

- Pass view holder class as type parameter
- View holder class normally nested within

Implementing RecyclerView Adapter

getItemCount

- Return number of items

onCreateViewHolder

- Create item view
- Store item view references in view holder

onBindViewHolder

- Receives view holder and display position
- Set display values using view holder

RecyclerView Item Selection

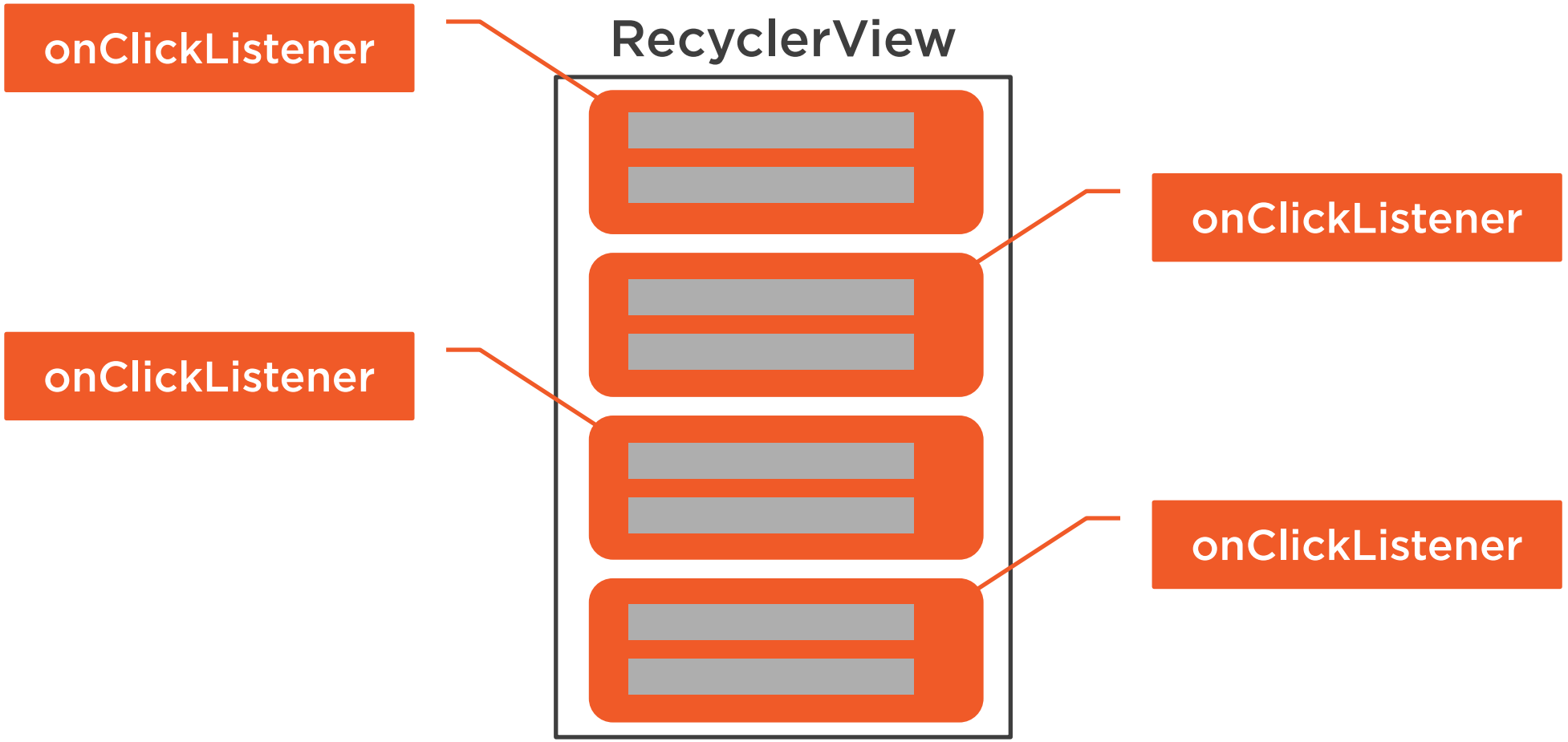
ListView item selection

- Used AdapterView.OnItemClickListener
- RecyclerView takes different approach

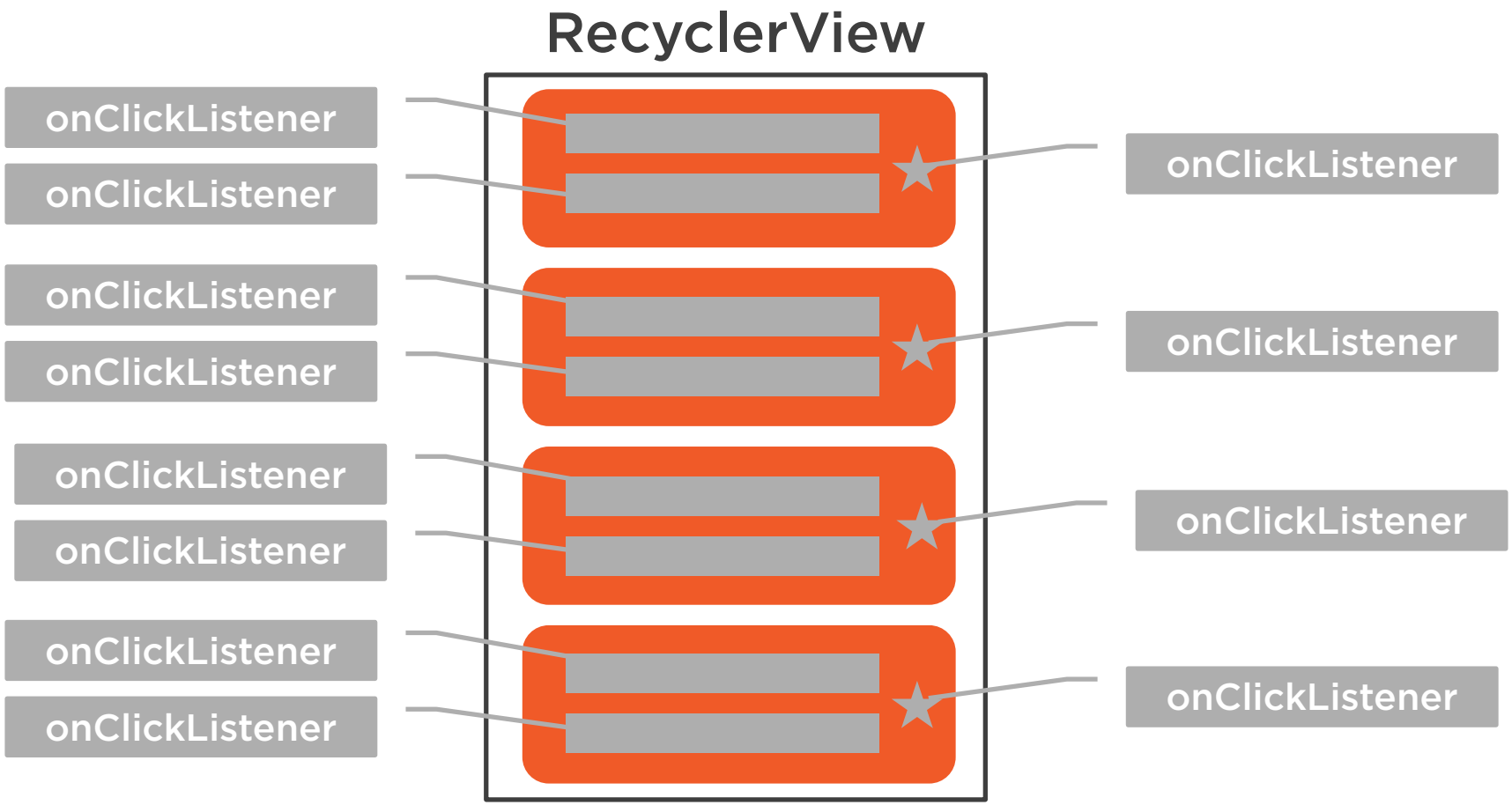
RecyclerView has no explicit support

- Contained views handle click event
- Allows single item selection
- Allows multiple selections within items

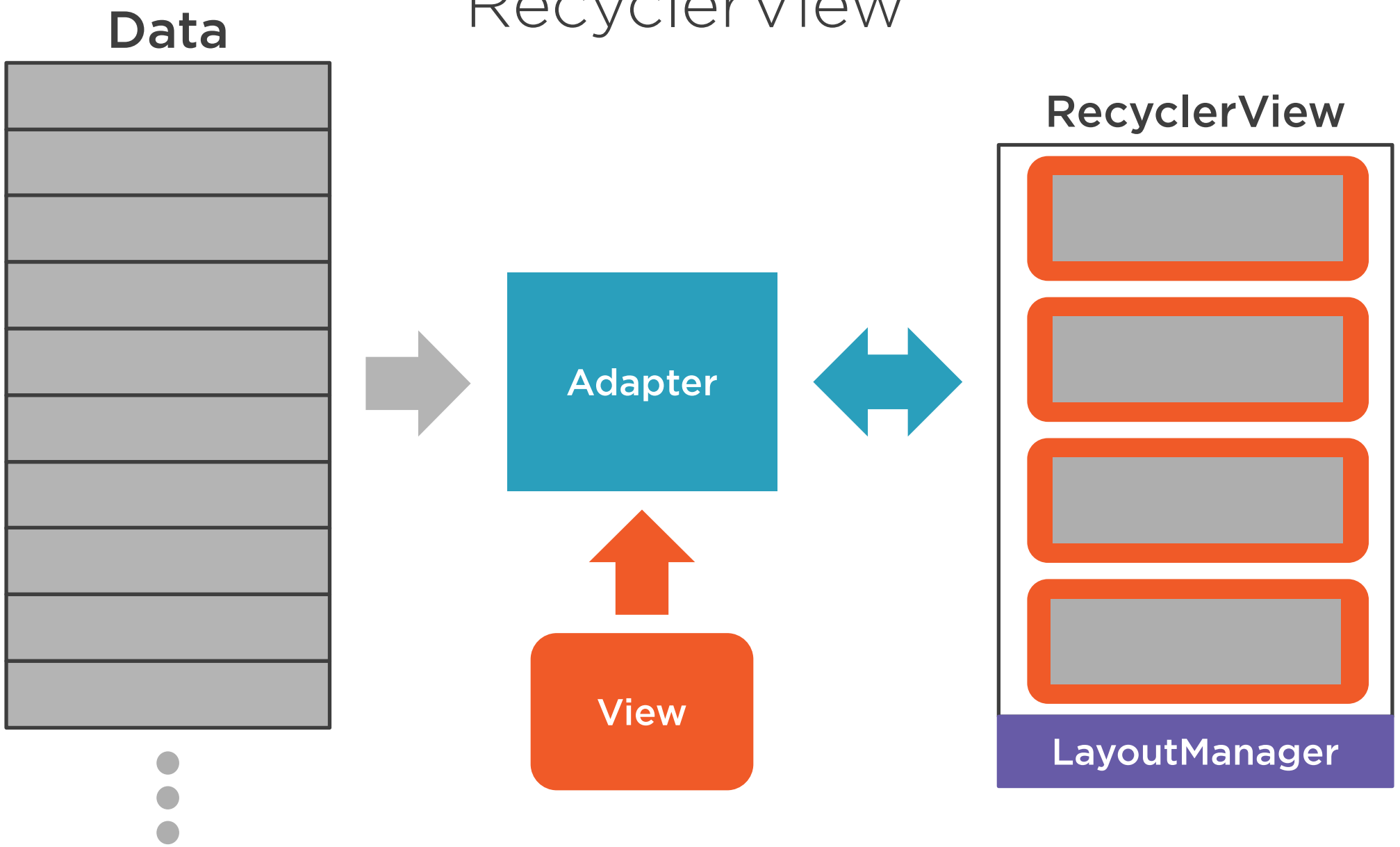
RecyclerView Item Selection



RecyclerView Item Selection



RecyclerView



Android Testing

Android applications

- Java-based behavior
- Android-based behavior

Testing Java-based behavior

- Local JVM tests

Testing Android-based behavior

- Instrumented tests

Instrumented Tests

Run on an emulator or physical device

- Have the full Android environment

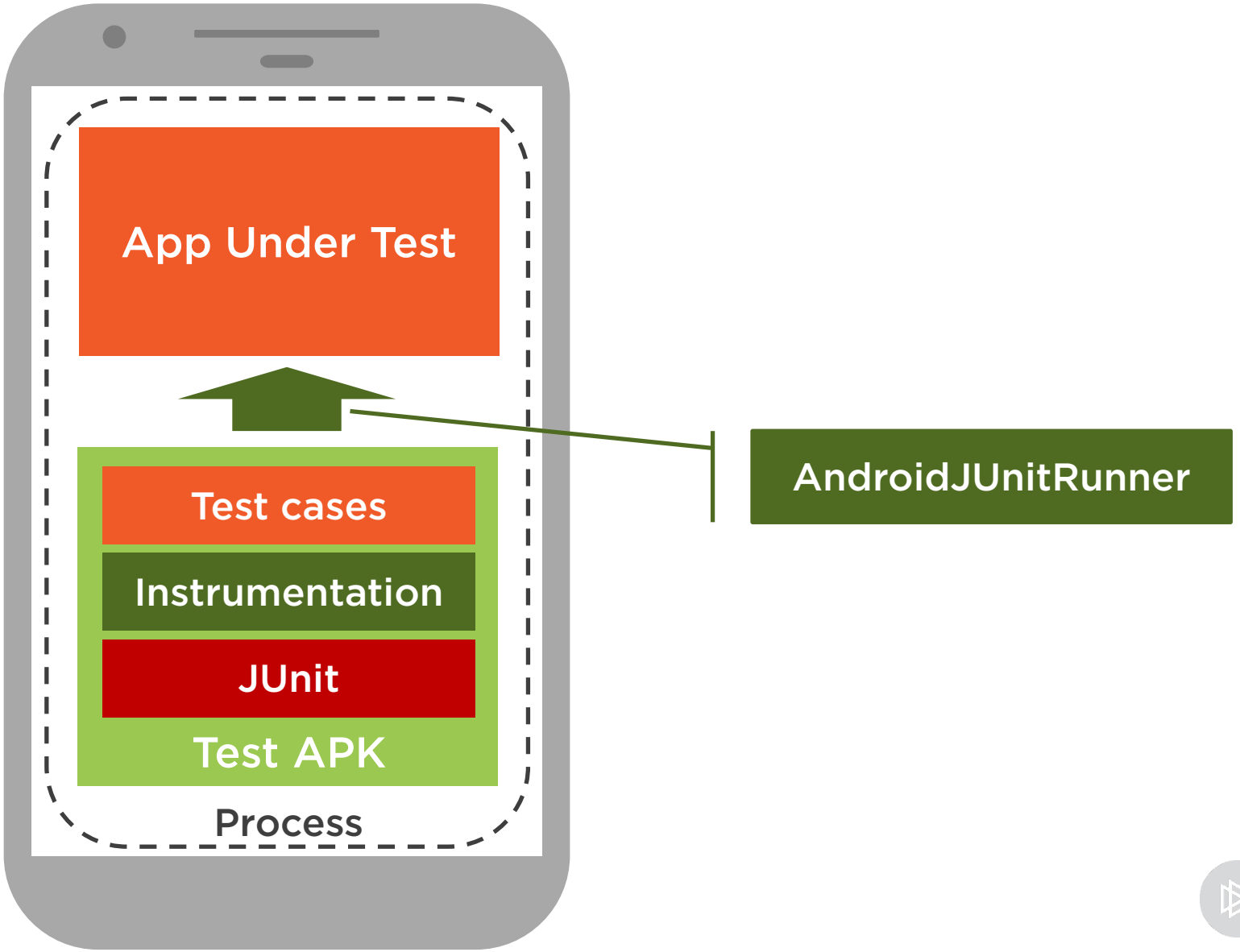
Instrumented Unit Tests

- Unit tests
- Rely on Android features/capabilities

Automated User Interface Tests

- Integration tests
- App behaviors in response to UI actions

Instrumented Tests



Implementing Instrumented Tests

Uses JUnit 4

- Test methods marked with @Test
- Supports pre/post-processing methods

Uses Assert class

- Indicate expectations
- Fails test if expectations not met

Test managed with Android Studio

- Can run or debug tests
  - Single test, group of tests, or all tests
- Displays tests results

Implementing Instrumented Tests

Organized separate from JVM tests

- In androidTest source set

Relies on Android JUnit test runner

- Class must have @RunWith annotation
  - Pass AndroidJUnit4.class

Requires Android environment

- Run on emulator or device

Implementing Instrumented Tests

```
@RunWith(AndroidJUnit4.class)
public class MyExampleTestClass {
  @BeforeClass
  public static void classSetUp() { . . . }
  @Before
  public void testSetUp() { . . . }
  @Test
  public void myTestMethod() {
    // Android dependent test code
  }
}
```

Creating UI Test Interactions

UI tests require a series of view interactions

- Need way to specify view of interest
- Need way to specify action on the view

Espresso.onView method

- Accepts a Matcher parameter
  - Specifies view matching criteria
- Returns a ViewInteraction reference
  - Associated with matching view
  - Used to perform action on view

Specifying View of Interest

Uses Hamcrest matchers

- Provides declarative matching
- General purpose Java framework
- http://hamcrest.org

ViewMatchers class

- Provides matchers for Android Views
- Methods return a Hamcrest matcher
- Easily combined with Hamcrest general purpose matchers

Specifying View of Interest

Example ViewMatchers methods

- withId
  - Match views based on id property
- withText
  - Match views based on text property
- isDisplayed
  - Match views currently on screen
- isChecked
  - Match currently checked checkable views (Switch, CheckBox, etc.)

Specifying View of Interest

Example Hamcrest Matchers

- equalTo
  - Match based on equals method
- instanceOf
  - Match based on class type
- allOf
  - Accepts multiple Matchers
  - Match if all Matchers match
- anyOf
  - Accepts multiple Matchers
  - Match if any Matchers match

Performing View Action

ViewInteraction.perform method

- Performs one or more specified actions
- Specific action passed as a parameter

ViewActions class

- Provides action methods
- Each method returns specified action

Performing View Action

Example ViewActions methods

- click
  - Click on the view
- typeText
  - Type text into view
- replaceText
  - Replace view's text
- closeSoftKeyboard
  - Closes the soft keyboard

Starting the Target Activity

ActivityTestRule

- Automates test activity lifetime
- Starts activity before each test
- Terminates activity after each test
- Activity life includes @Before/@After methods

Using ActivityTestRule

- Declare & initialize as test class field
- Desired activity as type parameter
- Mark field with @Rule annotation

Testing Views That Use Adapters

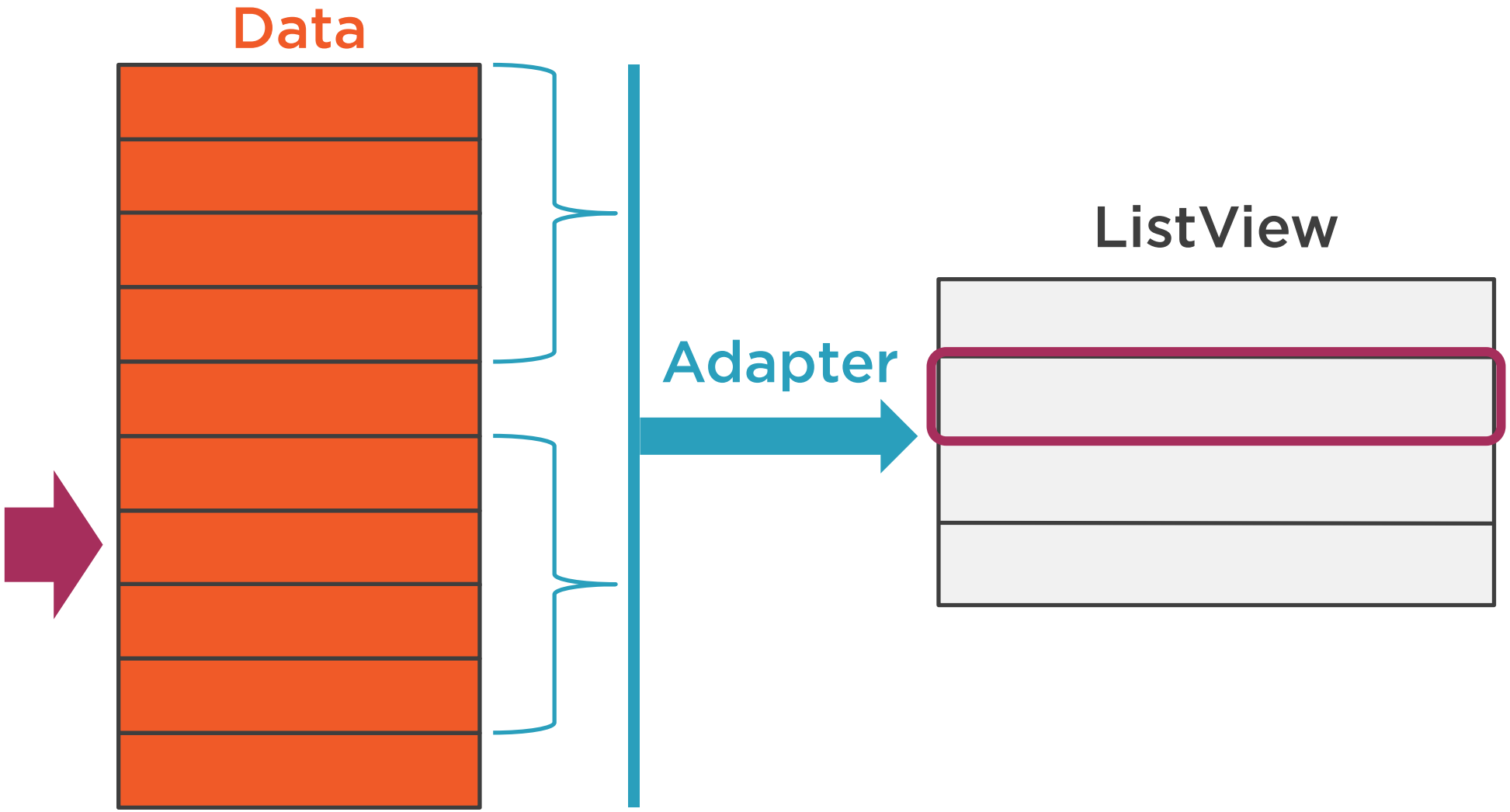
AdapterView derived views

- Load data from Adapter classes
- Examples include ListView & Spinner

These views display multiple data items

- Only a subset may be loaded
- Test selection based on target data

Testing Views that Use Adapters



Testing Views That Use Adapters

Espresso.onData

- Specify matcher based on target data
- Tend to use general purpose matchers

DataInteraction

- Provides methods for interacting with or narrowing match

Tend to use DataInteraction.perform

- Performs action on top-level view for entry in AdapterView

Back Button

Espresso.pushBack

- Performs action of pushing back button
- No reference to a view needed

Verifying Behavior

Tests meant to confirm expected behavior

- UI Behavior
- Logic Behavior

Verifying UI Behavior

ViewInteraction.check method

- Confirms some aspect of a view

ViewAssertions class

- Provides view assertion methods

Verifying UI Behavior

Common ViewAssertions methods

- matches
  - Confirms view matches passed matcher
  - Commonly used with ViewMatchers
  - Also confirms that view exists
- doesNotExist
  - Confirms that view does not exist

Verifying Logic Behavior

Assert methods still important

- Main way we confirm logic behavior

Verifying Logic Behavior

Remember that UI tests do two jobs

- Run testing behavior
- Run app behavior
- Test behavior interacts with app

Sequencing is important

- Each test action must wait for corresponding app action to complete
- Handled by ViewInteractions methods
  - i.e. perform, check, etc.
- Assert methods do not

Verifying Logic Behavior

Instrumentation class

- Allows interaction with test environment
- Access with InstrumentationRegistry
  - Use getInstrumentation method

Instrumentation.waitForIdle

- Accepts Runnable reference
- Provides code coordination
- Waits for app action triggered by previous test action to complete

Verifying UI Behavior

ViewInteraction.check method

- Checks that the view conforms to some assertion
- Accepts a ViewAssertion reference

ViewAssertions class

- Provides methods for asserting some aspect of a view
- Methods create a ViewAssertion

Specifying View of Interest

Example ViewMatchers methods

- withId
  - Match view based on id property
- withText
  - Match view based on text property
- isDisplayed
  - Match views currently on screen
- isChecked
  - Match currently checked checkable views (Switch, CheckBox, etc.)

Testing

Testing needs to be a core task

- Essential to delivering quality software

Functional testing

- Verify behaves as expected
- Detect breaking changes

Testing

Unit testing

- Testing of units of code
- Each unit test is relatively simple
  - Tests a specific feature/behavior
- Generally will have many unit tests

Integration testing

- Testing of pieces being put together
- Application behaviors
- Often involves testing of UI

Unit Testing

Unit tests should be run often

- After code changes
- Before check-in to source code control

Generally want to run all unit tests

- No change is complete until all tests pass

Ideally can be run reasonably quickly

Android Application Testing

Challenges of testing Android apps

- Full testing needs Android environment
- Requires emulator or a physical device

Need way to efficiently run unit tests

- Limit how often full environment needed

Efficiently Running Unit Tests

Android applications

- Java-based behavior
- Android-based behavior

Separate tests

- Tests for Java-based behavior
- Tests for Android-based behavior

Efficiently running unit tests

- Tests Java parts of app locally
- Leverage JVM on desktop

Local JVM Tests

Android Studio JVM testing

- Separate source set for JVM tests
- Uses JUnit 4

Managing tests with Android Studio

- Can run or debug tests
  - Single test, group of tests, or all tests
- Displays tests results
  - Success/failure indicated by color

Testing with JUnit

Each unit test is a separate method

- Marked with @Test annotation
- JUnit handles details of running method

Tests grouped within classes

- Primarily for organization convenience
- Allows execution grouping
- Allows setup/teardown grouping

Testing with JUnit

Assert class

- Use to indicate expected results
- Fails test when expectation not met

Testing with JUnit

Example Assert class methods

- assertSame
  - Two references to same object
- assertEqual
  - Two objects equal (equals method)
- assertNull
  - Reference is null

Negative versions of most methods

- i.e. assertNotSame, etc.

Assuring Test Consistency

Test reliability

- Each test must always run consistently
- Can't depend on action of another test
- Can't be impacted by side effects of other tests

Test must always start from same state

- Test order is not guaranteed
- Need way to set/reset test state

Assuring Test Consistency

Test pre-processing

- Method with @Before annotation
- Will run before each test in class

Test post-processing

- Method with @After annotation
- Will run after each test in class

Assuring Test Consistency

Once per class pre-processing

- Method with @BeforeClass annotation
- Will run once before all tests in class
- Method must be static

Once per class post-processing

- Method with @AfterClass annotation
- Will run once after all tests in class
- Method must be static

Assuring Test Consistency

Multiple pre/post-processing methods

- Multiples in a class valid
- All will run
- Sequence is not guaranteed

Summary

Testing needs to be a core task

- Essential to delivering quality software

Unit Testing

- Units of relatively simple tests
- Focus is on specific feature/behavior
- Tests should be run frequently

Summary

Local JVM testing

- Focused on Java aspects of our app
- Run directly on desktop

Managing tests in Android Studio

- Can run or debug tests
- Displays test success or failure

Summary

Test methods

- Grouped in classes
- Marked with @Test annotation

Assert class

- Use to indicate expected results
- Fails test when expectation not met

Summary

Tests need to be reliable

- Need to assure consistent testing

Can run test pre/post-processing

- Use @Before and @After
- Methods run for each @Test method

Can run test class pre/post-processing

- Use @BeforeClass and @AfterClass
- Methods run once for test class
- Methods must be static

Android Build Process

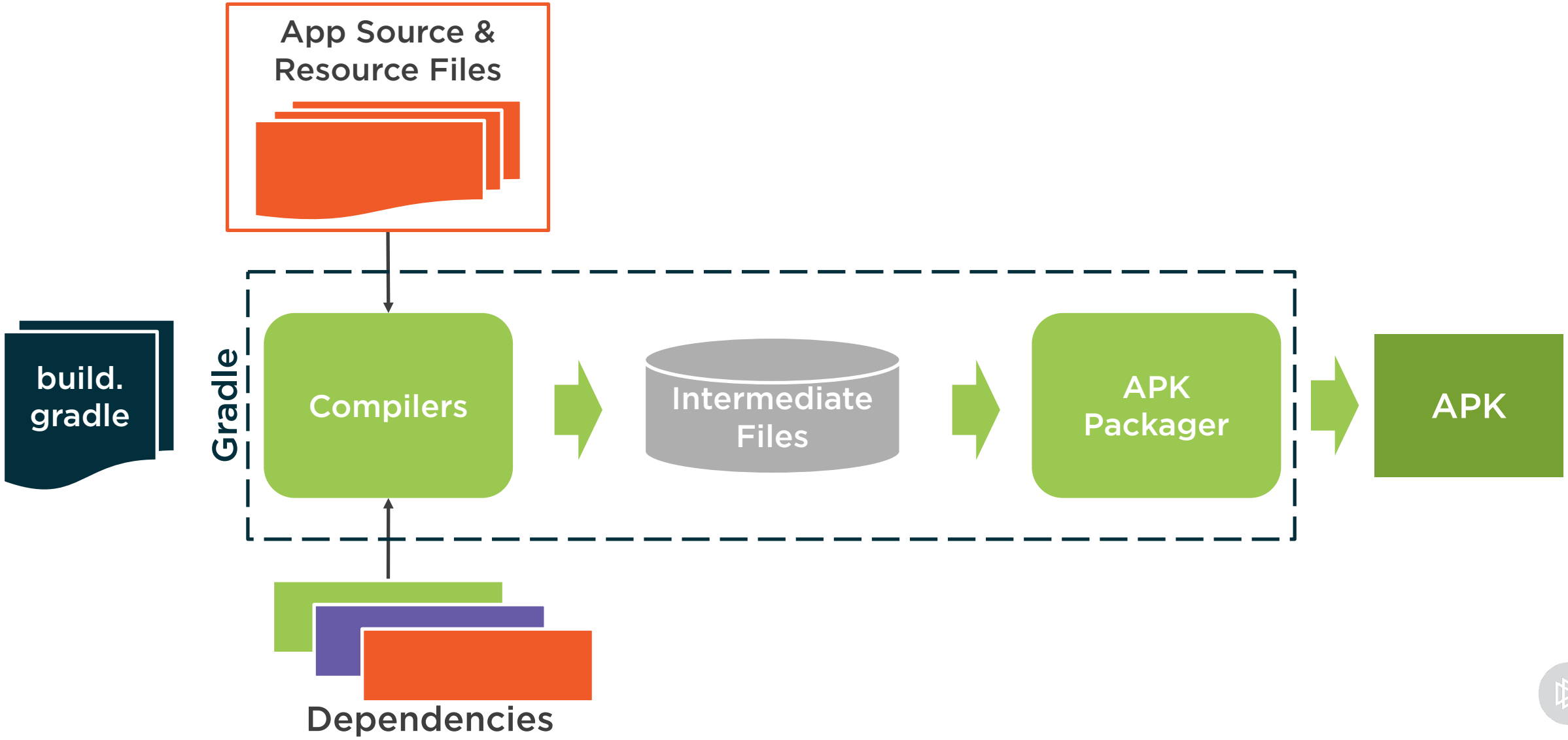
Android build process is somewhat involved

- Manually managing details is challenging

Gradle simplifies managing build process

- A general purpose build system
- Android-oriented features from plug-in

Android Build Process



Configuring Gradle

Gradle is extremely powerful

- Very flexible
- Uses a domain specific language (DSL)

Common settings easily managed

- Projects include build.gradle files
- Changes often simple edits

Android Studio UI

- Many changes can be made with UI
- Use File/Project Structure...

Dependencies

Applications builds rarely stand alone

- May rely on external binaries
- May rely on other project libraries

Listing dependencies in Gradle

- In build.gradle dependencies block
- Automatically includes dependency dependencies

Dependency Types

Module dependency

- Module from your project

Jar dependency

- Java jar file

Library dependency

- Pull from a repository

Library Dependencies

Will use local machine for some

- Android Support repository
- Google repository

Other repositories must be specified

- Normally leverages jcenter repository
- Can add others

Associating Dependencies

Dependency for all build variants

- Use compile

Dependency for JVM test

- Use testCompile

Dependency for Instrumentation test

- Use androidTestCompile

Android Support Library

Backward compatibility

- Makes some newer platform features available to older platform versions
- Uses alternate classes

Convenience and helper classes

- Provides features not part of platform
- Especially in the area of the UI

Debugging, testing, and utilities

- Testing Support Library
- Enhanced code checks
- Special case utilities

Android Support Library Organization

Most grouped by platform support

- Name historically indicates min platform
- v4 Support libraries – API 4 & up
- v7 Support libraries – API 7 & up
- v13 Support libraries – API 13 & up
- Changed with latest releases
  - None support less than API 9

Specific libraries tied to features

- Multiple libraries within each group
- We reference specific library in Gradle

Android Support Library Organization

Historically grouped by supported platform

- Name indicates minimum platform
- v4 Support libraries – API 4 & up
- v7 Support libraries – API 7 & up
- v13 Support libraries – API 13 & up
- Changed with latest releases
  - None support less than API 9

Specific libraries tied to features

- Multiple libraries within each group
- We reference specific library in Gradle

Android Build Process

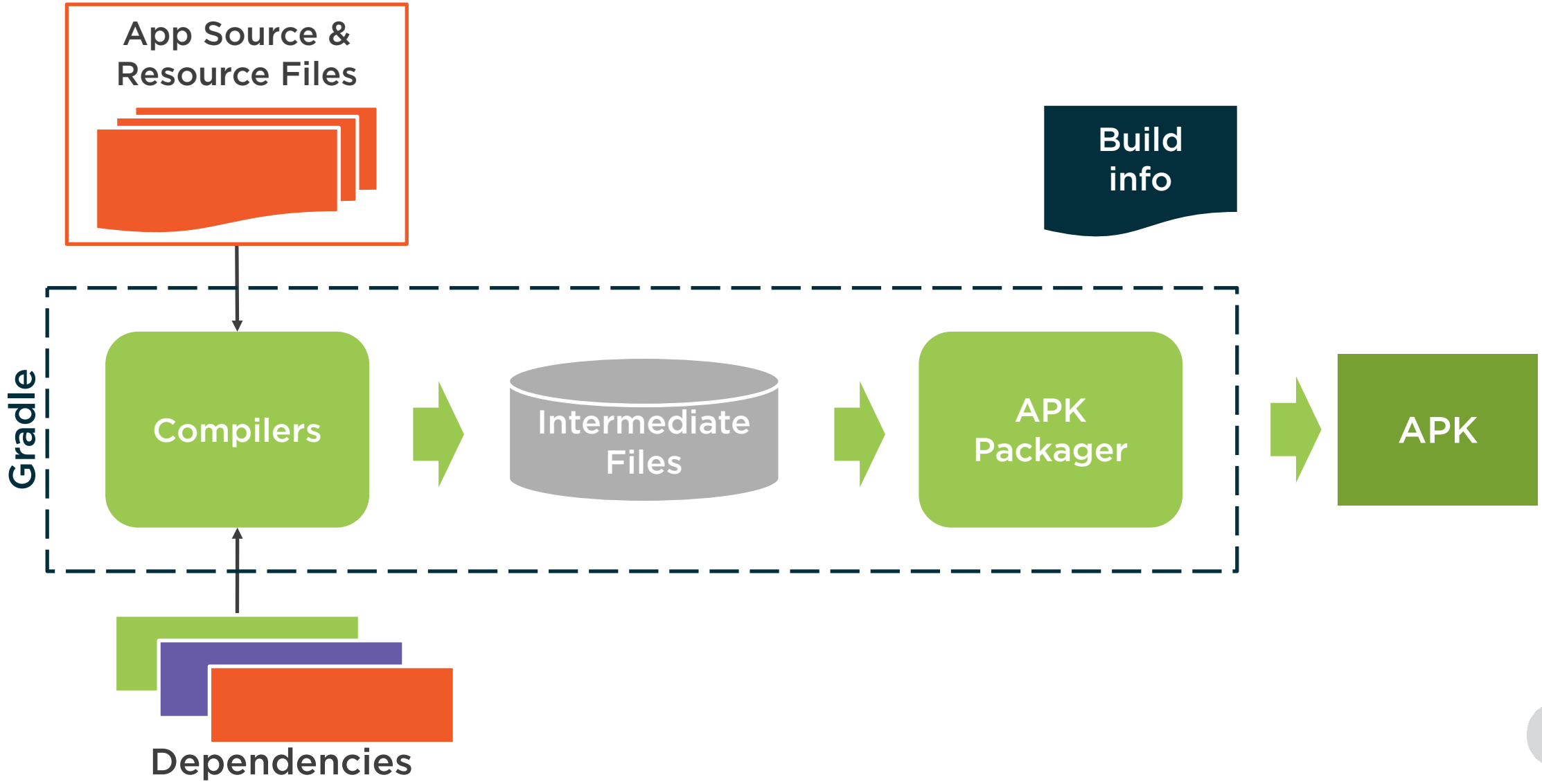
Android build process is somewhat involved

- App source/resource compilation
- App dependencies
- Producing app package (APK) file

Gradle simplifies managing build process

- A general purpose build system
- Android-oriented features from plug-in

Android Build Process



Android Studio

Primary Android development tool

Handles installing the things we need

- Android SDK
- JDK
- Other Android development tools

Built on IntelliJ IDEA technology

Android Studio

- Code development & UI design
- Developer productivity
- Code deployment and execution
- Debugging
- Build system
- Testing

Logcat

- System for recording log information

Log class

- Provides methods for writing to logcat
- Includes a tag with each message
  - Commonly use class name as tag

Android Monitor

- Displays logcat messages
- Filterable
- Searchable

Messages are organized into levels

- Indicates relative severity

Log class

- Provides methods for each level

Viewing messages

- Messages are labeled with level
- Can limit which levels are displayed
- Shows messages >= selected severity

Logcat Levels

More Inclusive

Level	Label	Log Method

More Severe

Summary

Android Studio

- Central tool for Android development

Developor Productivity

- Refactoring
- Code generation
- Customizable to developer style

Summary

Program execution and debugging

- Handles installing and running app
- Breakpoints and code stepping
- Viewing variables
- On-the-fly statement evaluation

Instant Run

- Accelerates deploying code changes
- Does just enough to apply changes
- Often allows app to keep running

Summary

Resolving unhandled exceptions

- Crash information in Android Monitor
- Shows exception and call stack
- Provides links to app source code lines

Logcat

- System for recording log information
- Written to with Log class
- Viewable in Android Monitor

Logcat Levels

More Inclusive

Level	Label	Log Method

More Severe

ore

Testing with a Real Device

Debugging

Application Activity Relationship

Android is a component-oriented platform

- Components run within a process

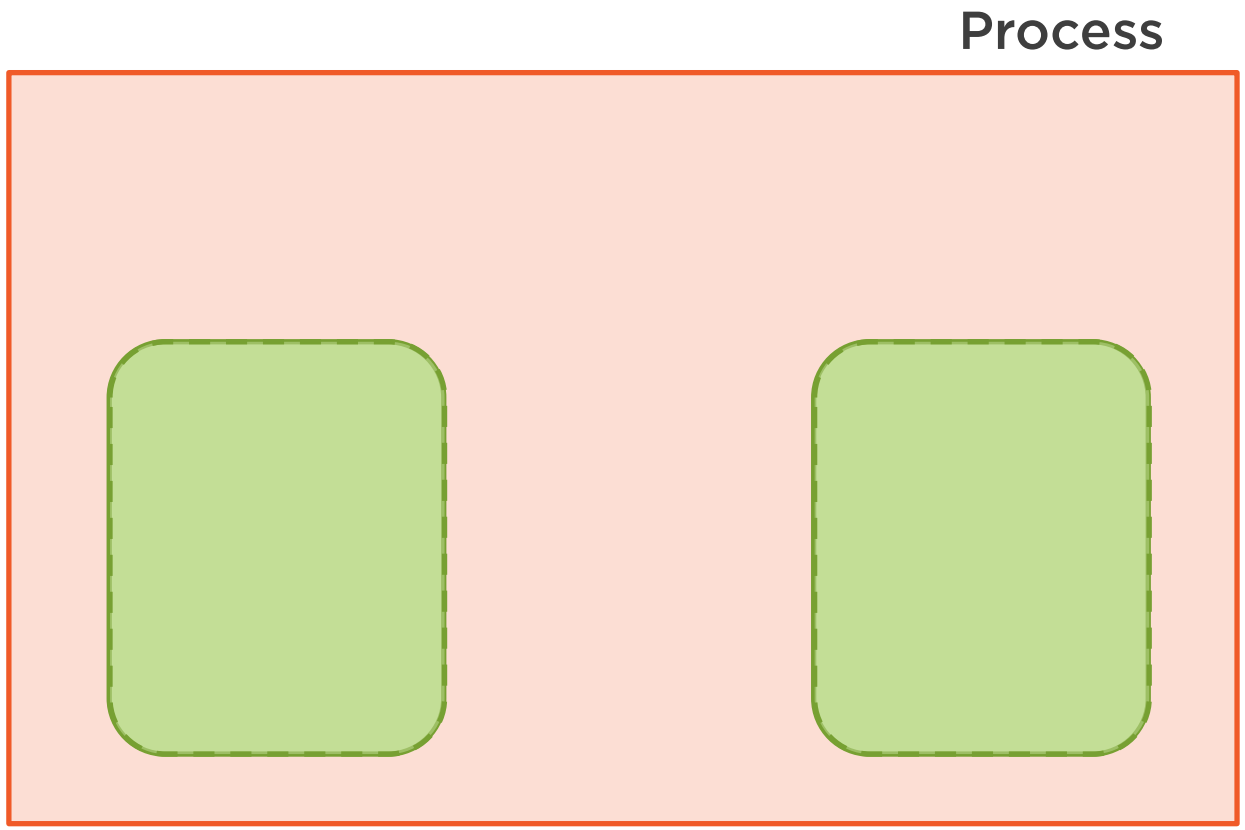
Process lifetime

- Driven by component lifetime
- Launched for first component accessed
- Terminates after last component exits

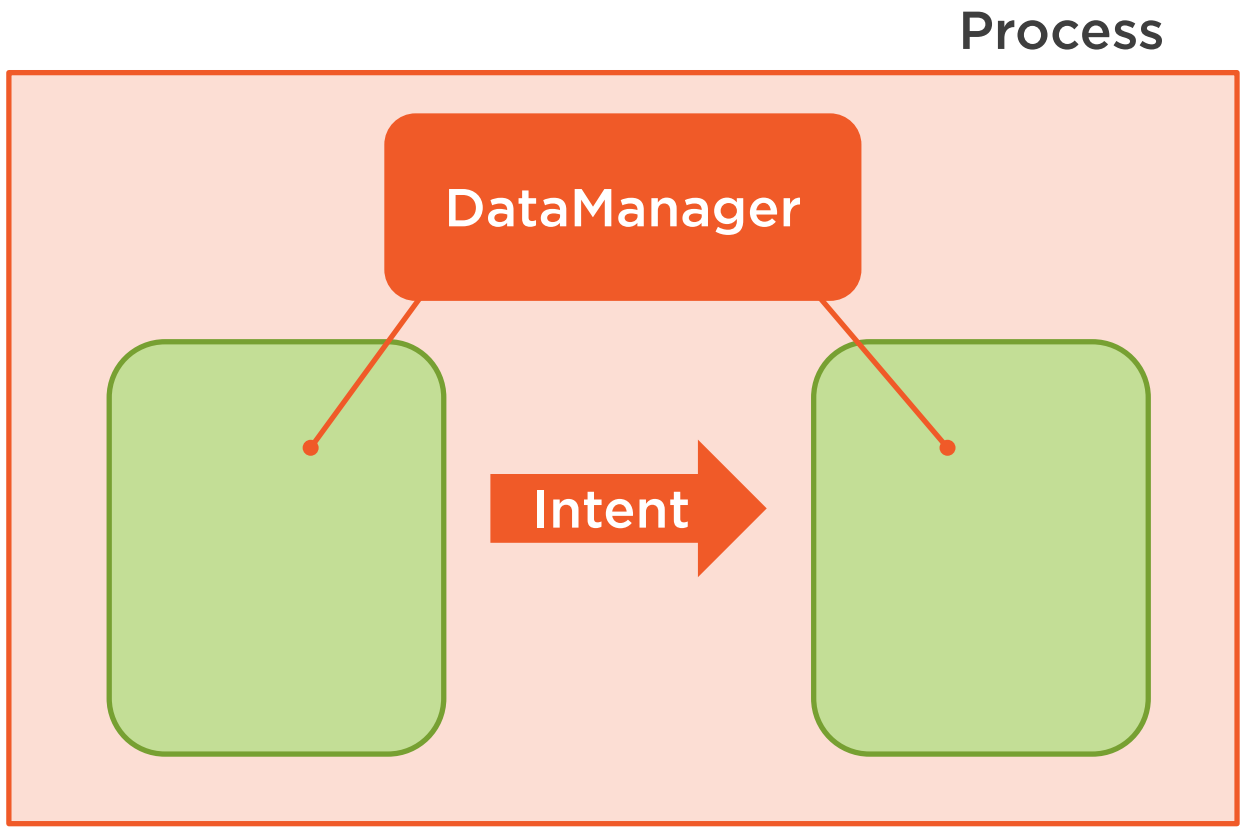
Application process

- Each app has its own process
- App components run in same process
  - When simultaneously active

Application Activity Relationship



Application Activity Relationship



Late-binding Components

Intents describe a desired operation

- Identify operation target

Explicit intent

- Target is explicitly identified
- Specify the Activity class to use

Implicit intent

- Target is implied
- Specify the Activity characteristics

Late-binding Components

Implicit intents provide late-binding

- Match is determined at runtime

System finds best match

- Often comes from another app
- Specific match may vary depending on apps installed on user device
- Prompts user if tie

Decouples sender and receiver

- Sender may not know receiver
- Receiver may not know sender

Implicit Intent Characteristics

Action

- Action string
- Many standard constants available
- Example: Intent.ACTION_VIEW
- Commonly set in Intent constructor
- Only required characteristic

Category

- Provides extended qualification
- Not normally used by sender

Implicit Intent Characteristics

Data

- URI of data to be acted upon
- Example: https://pluralsight.com
- Set with Intent.setData

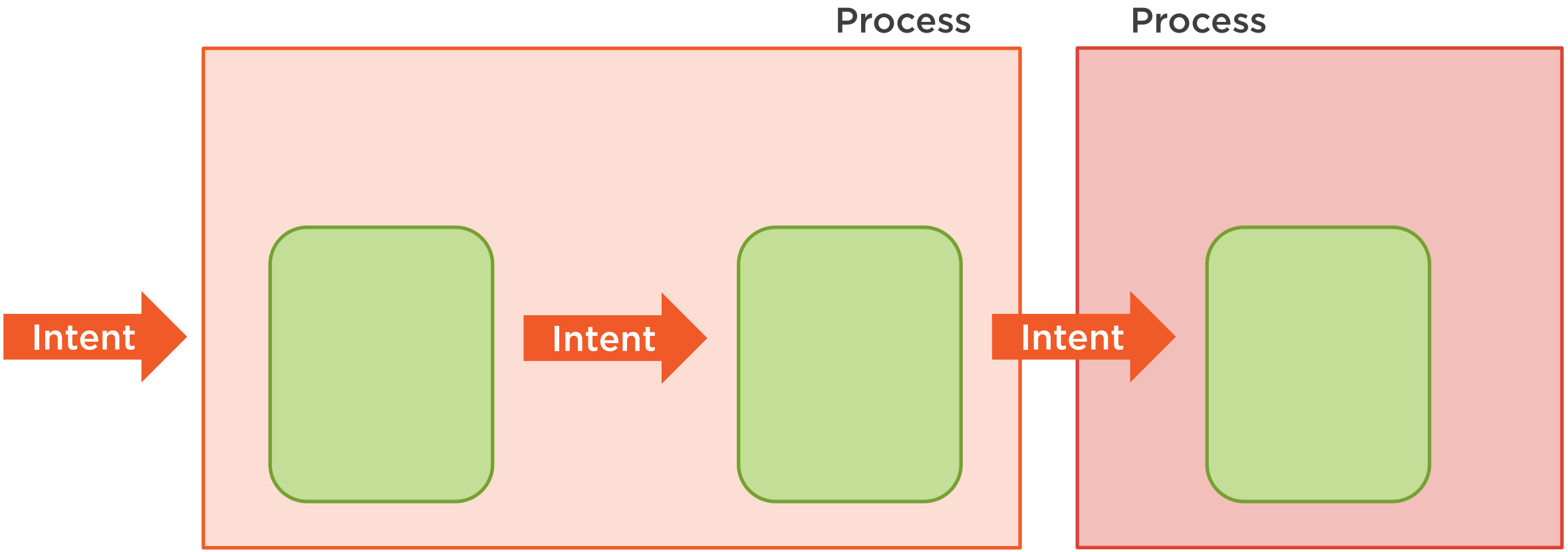
Mime type

- Common or app-specific mime type
- Example: text/html
- Set with Intent.setType

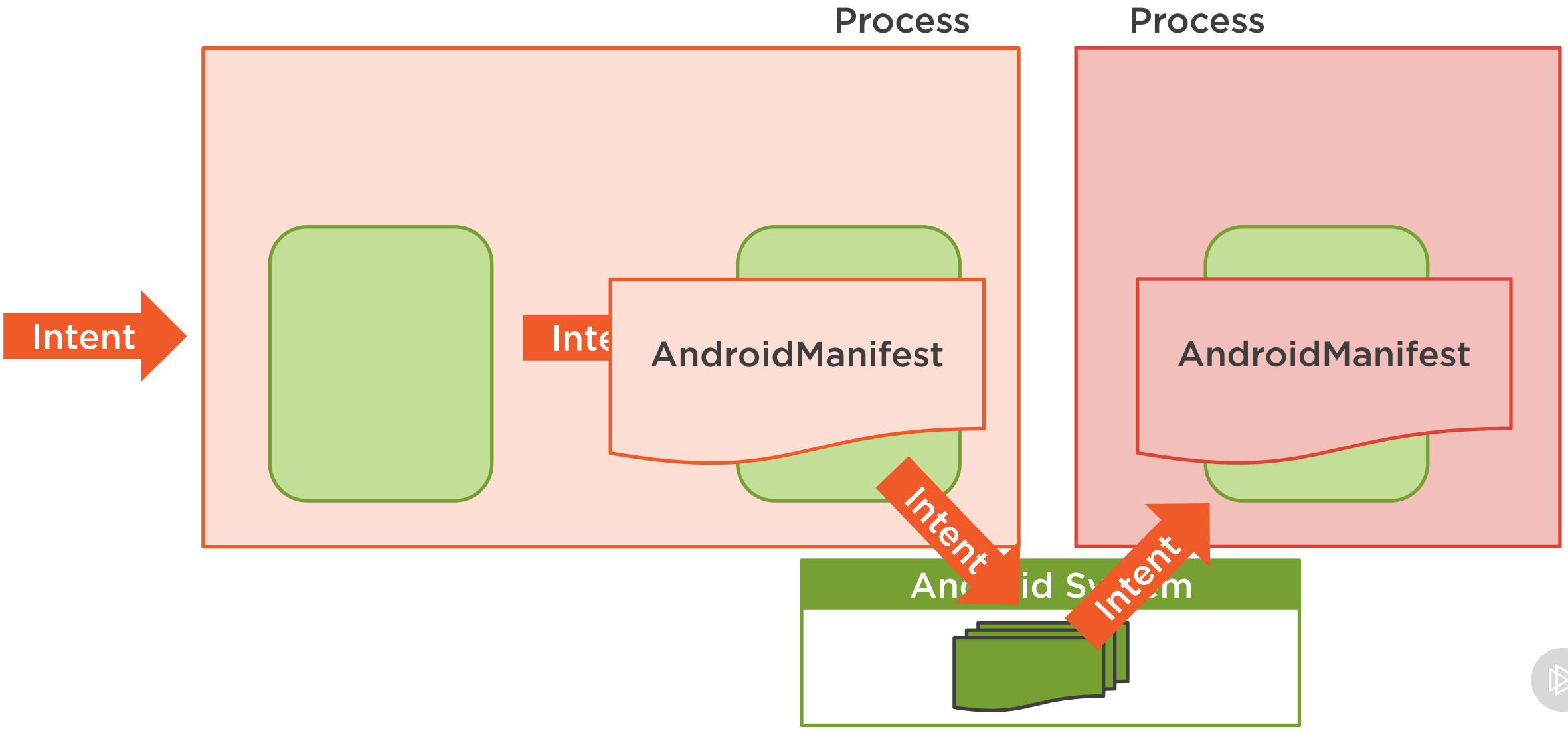
Setting data and mime type

- Use Intent.setDataAndType
- setType and setData cancel each other

Implicit Intents



Implicit Intents



Activities with Results

Some Activity classes return results

Camera Activity

- Presents camera functionality
- Returns image thumbnail

Contact Activity

- Presents contact UI
- Returns selected contact info

Many others

Activities with Results

Started differently than other activities

- Use startActivityForResult

Parameters passed to startActivityForResult

- Intent
- App defined integer identifier
  - Differentiates results within your app

Activities with Results

Receiving results

- Override your Activity's onActivityResult

Parameters received by onActivityResult

- App defined integer identifier
- Result code
  - RESULT_OK indicates success
- Intent
  - Contains activity results

Activity with Result Example

Camera

- Presents Camera UI
- Stores full quality image in a file
- Returns image thumbnail as a result

Starting the Activity

Intent action

- MediaStore.ACTION_IMAGE_CAPTURE

Extra

- MediaStore.EXTRA_OUTPUT
- File in which to save full quality image

Starting the Activity

```
public class MyActivity extends AppCompatActivity {
  private static final int SHOW_CAMERA = 1;
  private void showCamera(Uri photoFile) {
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    intent.putExtra(MediaStore.EXTRA_OUTPUT, photoFile);
    startActivityForResult(intent, SHOW_CAMERA)
  }
  // other members elided for clarity
}
```

Receiving the Result

Check for request code of SHOW_CAMERA

- Identifies the result is for our request

Check for result code of RESULT_OK

- Indicates success
- Full quality image stored in file

Retrieve thumbnail

- Stored in result intent as a bitmap
- Name is “data”

Receiving the Result

```
public class MyActivity extends AppCompatActivity {
  private static final int SHOW_CAMERA = 1;
  @Override
  protected void onActivityResult(
        int requestCode, int resultCode, Intent result) {
    if(requestCode == SHOW_CAMERA && resultCode == RESULT_OK) {
      Bitmap thumbnail = result.getParcelable(“data”);
      // Do something
    }
  }
  // other members elided for clarity
}
```

Application Experience

Generally composed of multiple Activities

- Most probably come from your app
- Others may come from other apps
- Android needs to manage this flow
- Flow is managed as a task

What is a Task?

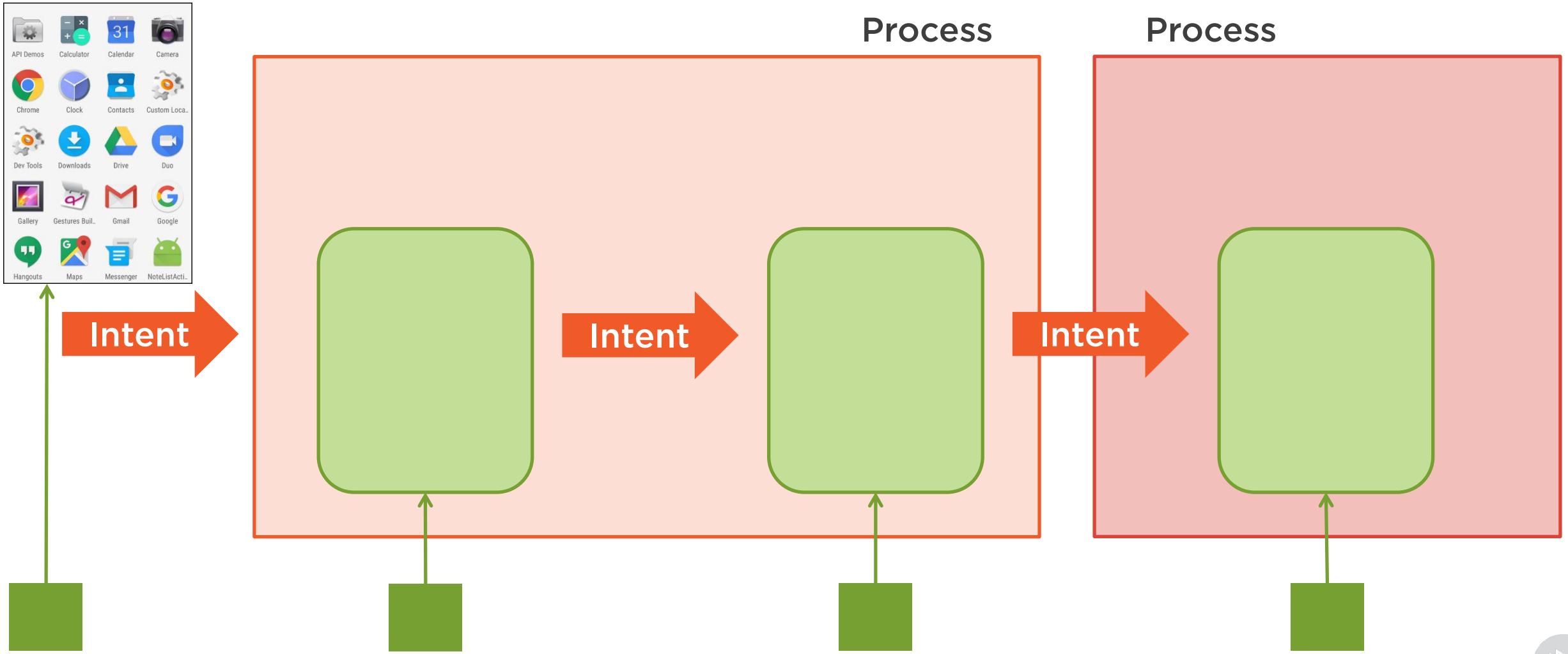
A task is a collection of activities that users interact with when performing a certain job.

Application Experience

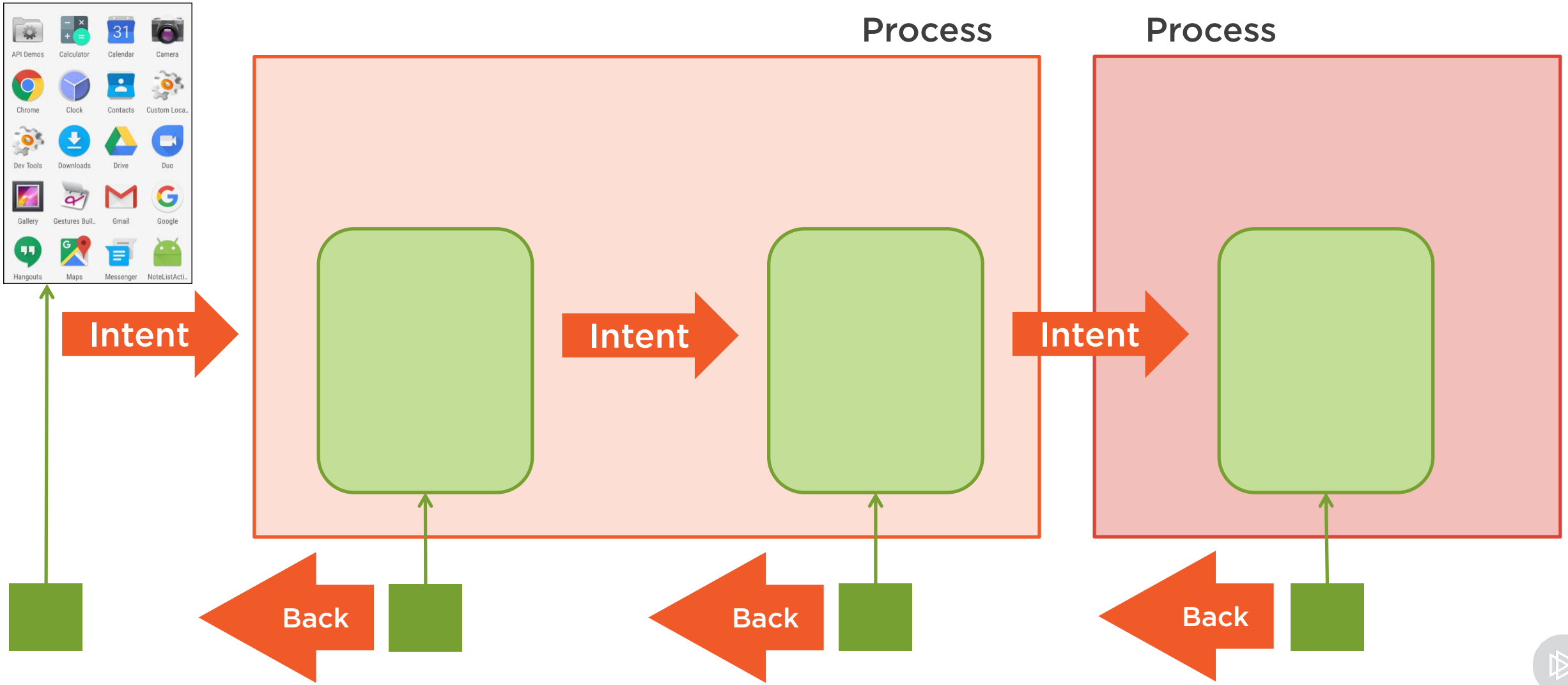
Task

- Managed as a stack
  - Known as the back stack
- Activities added going forward
- Back button removes Activities
  - Causes Activity to be destroyed

Task


API Demos
Calculator
Calendar
Camera
Chrome
Clock
Contacts
Custom Loca..
Dev Tools
Downloads
Drive
Duo
Gallery
Gestures Buil..
Gmail
Google
Hangouts
Maps
Messenger
NoteListActi..
Intent
Process
Intent
Process
Intent

Task



Application Experience

Managing persistent state

- Use edit-in-place model
- Changes saved with no special action

Saving changes

- Write to backing store when leaving
- Handle in onPause

Handling new entries

- New entries created right away
- Handle in onCreate

Activity Lifecycle

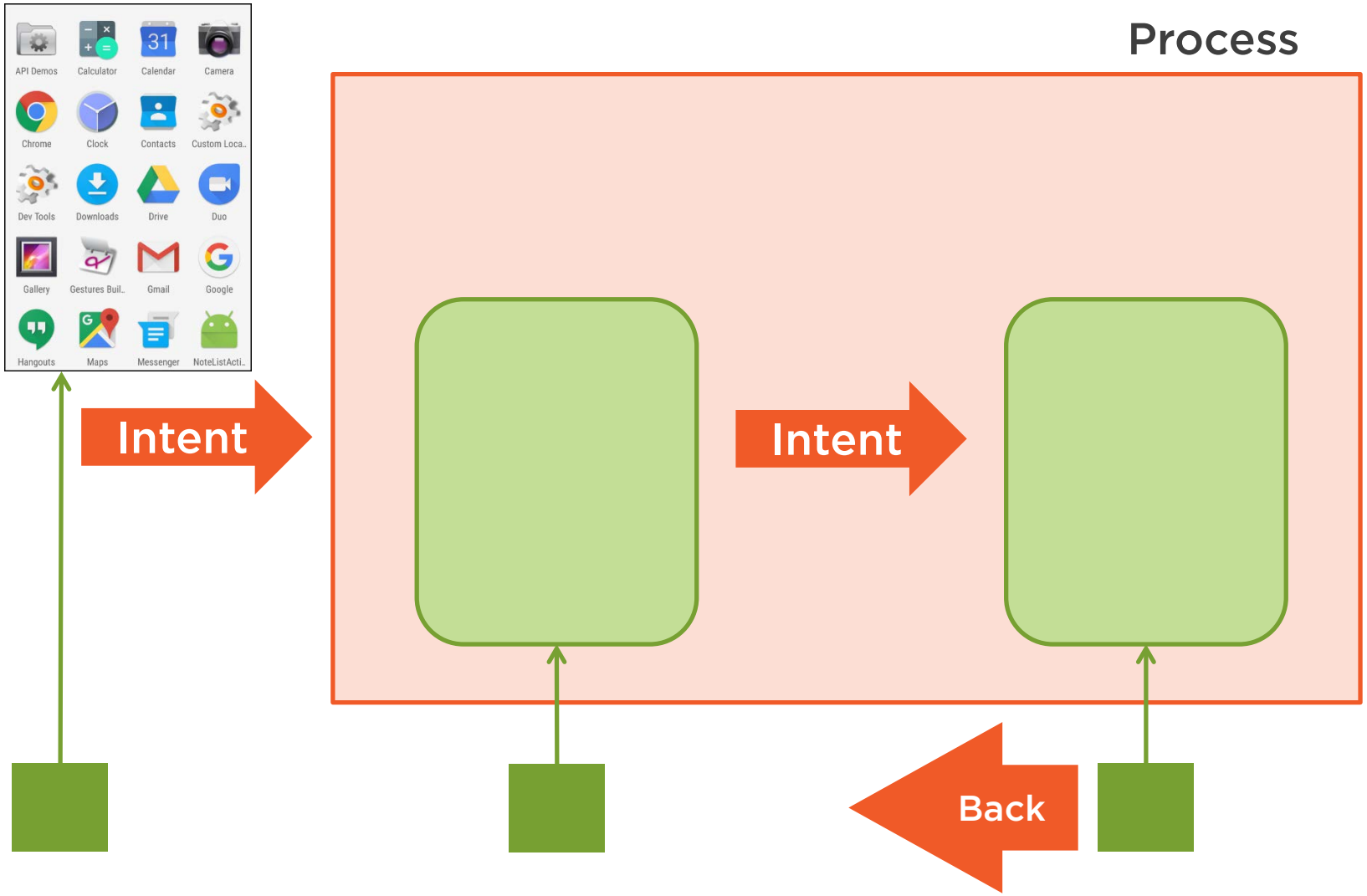
Common causes of Activity destruction

- Leaving with the back button
- Calling finish method
- System initiated

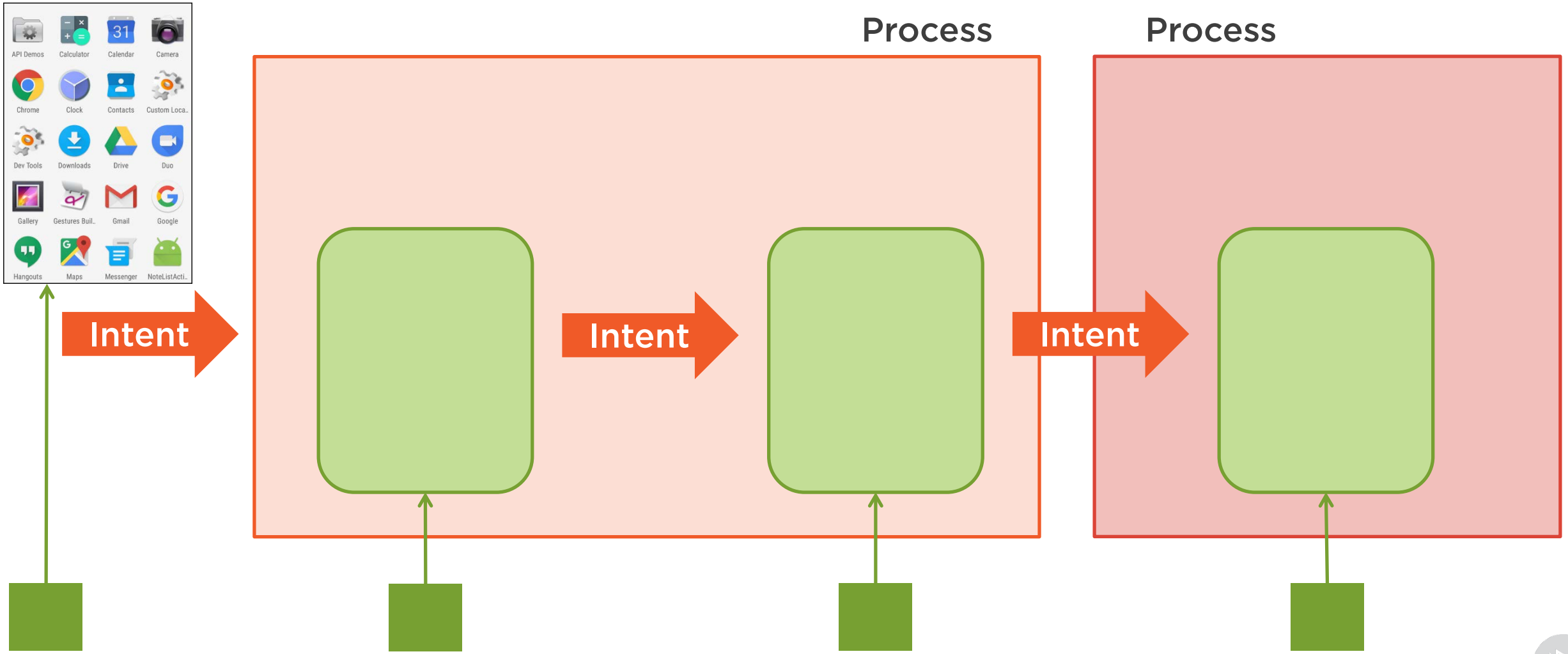
System initiated destruction

- Generally to reclaim resources
- Prolonged period in the background
- System experiencing resource pressure

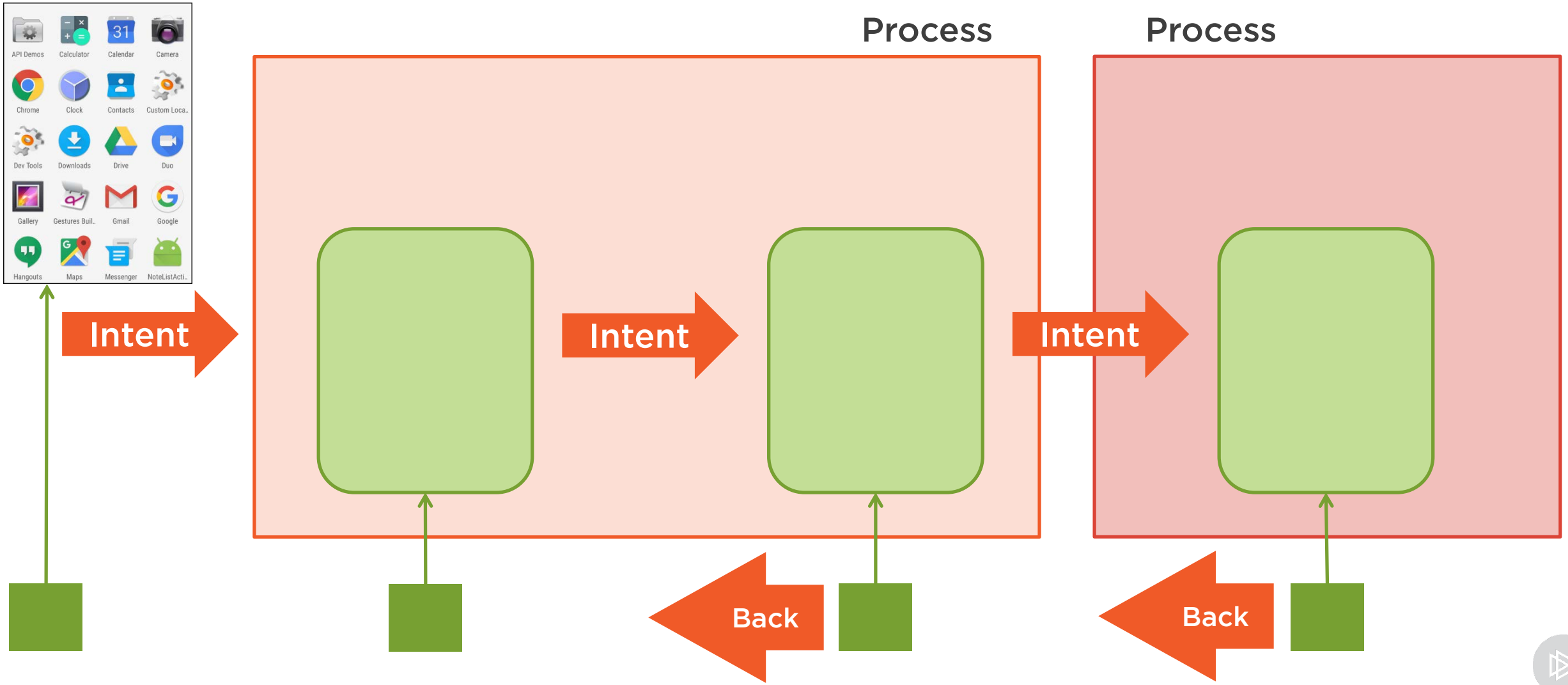
Task



Task


API Demos
Calculator
Calendar
Camera
Chrome
Clock
Contacts
Custom Loca..
Dev Tools
Downloads
Drive
Duo
Gallery
Gestures Buil..
Gmail
Google
Hangouts
Maps
Messenger
NoteListActi..
Intent
Process
Intent
Process
Intent

Task



Activity Lifecycle Methods

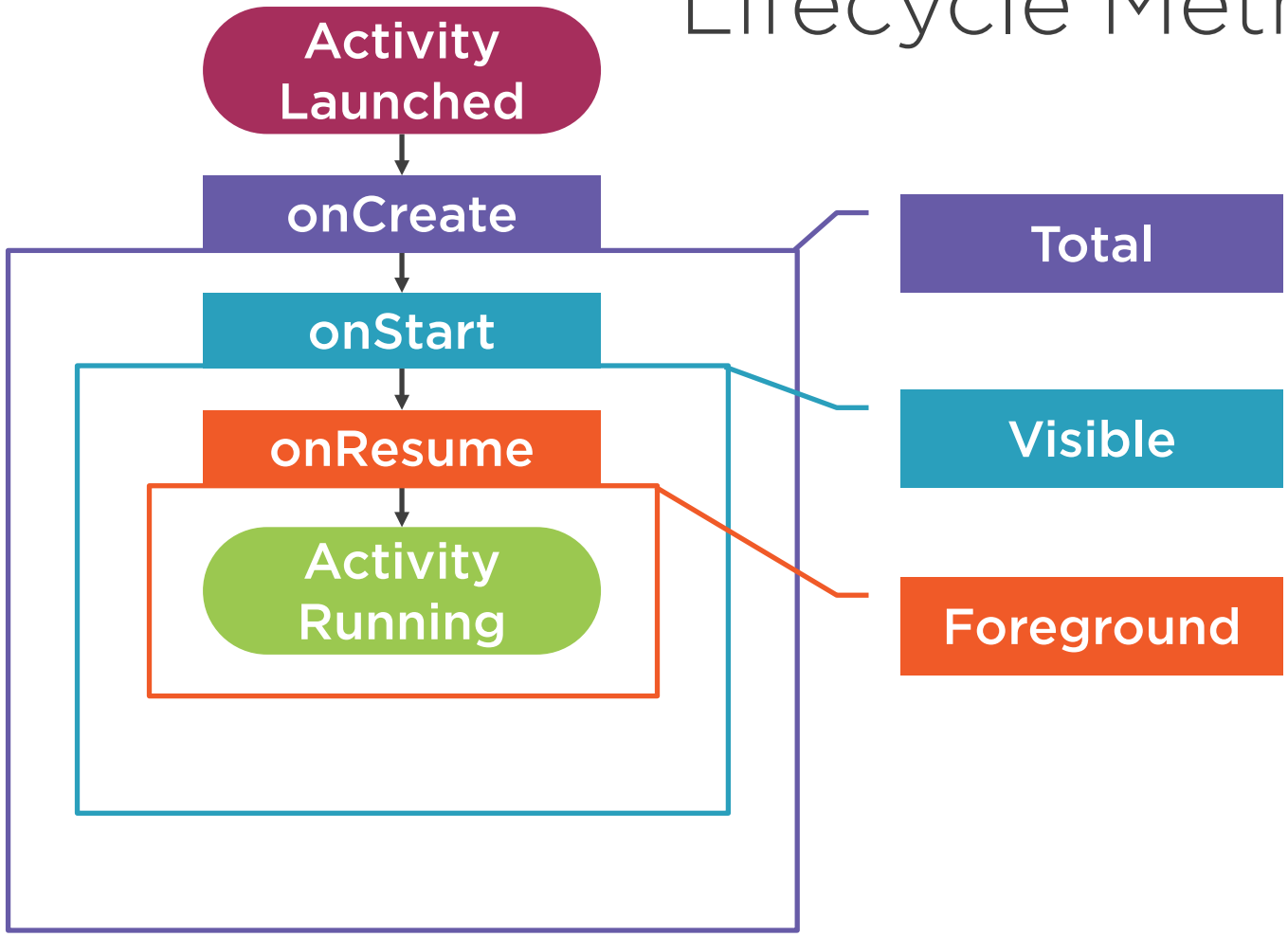
Lifetimes within Activity lifecycle

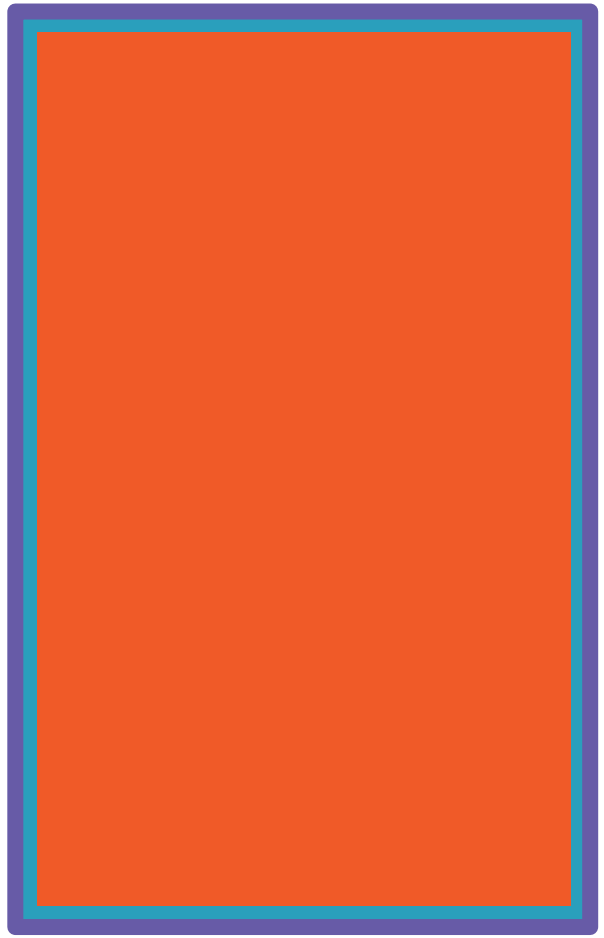
- Total lifetime
- Visible lifetime
- Foreground lifetime

Activity lifecycle methods

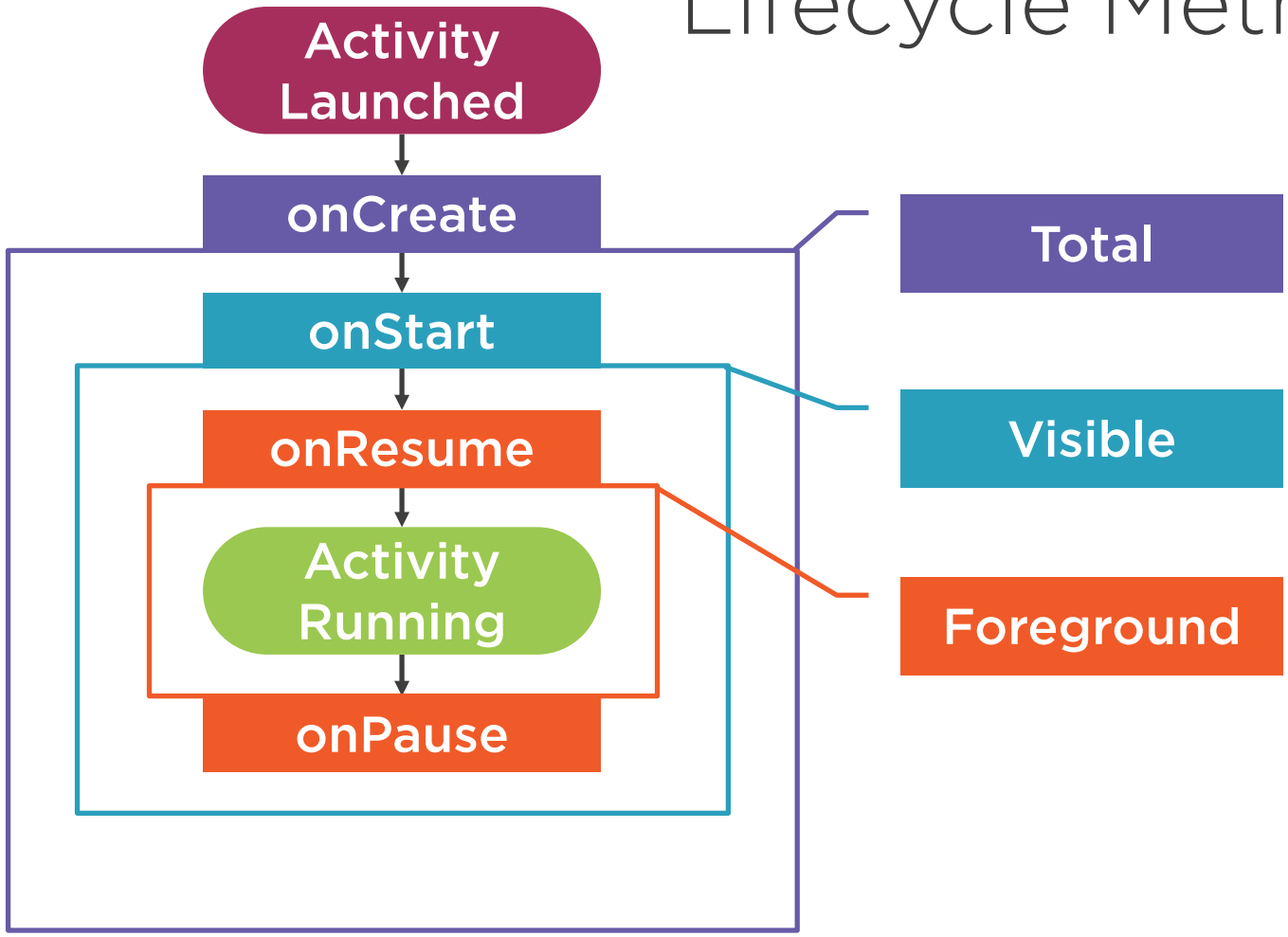
- Methods for start/end of each lifetime
- A few additional methods for transitions

Lifecycle Methods

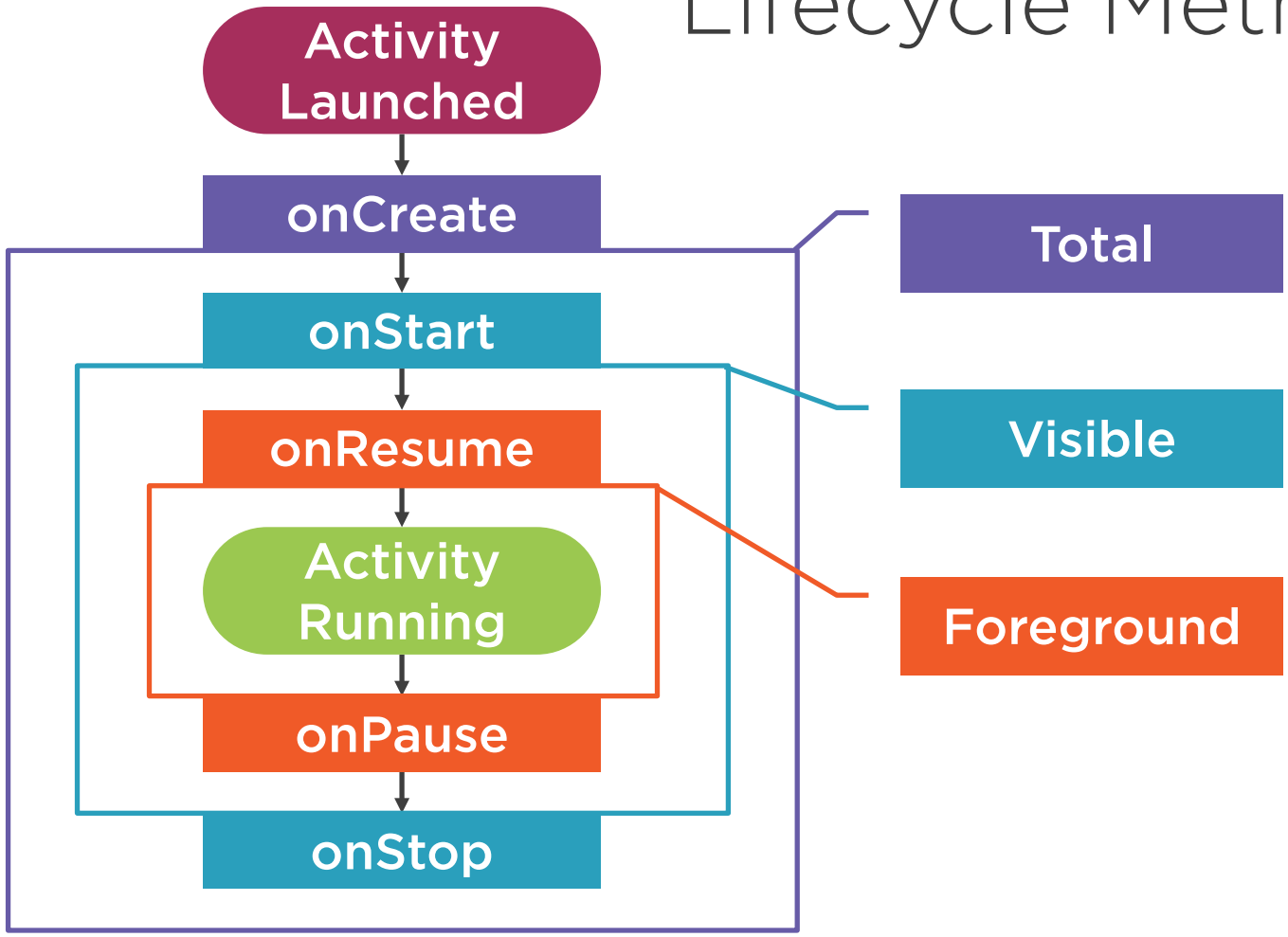




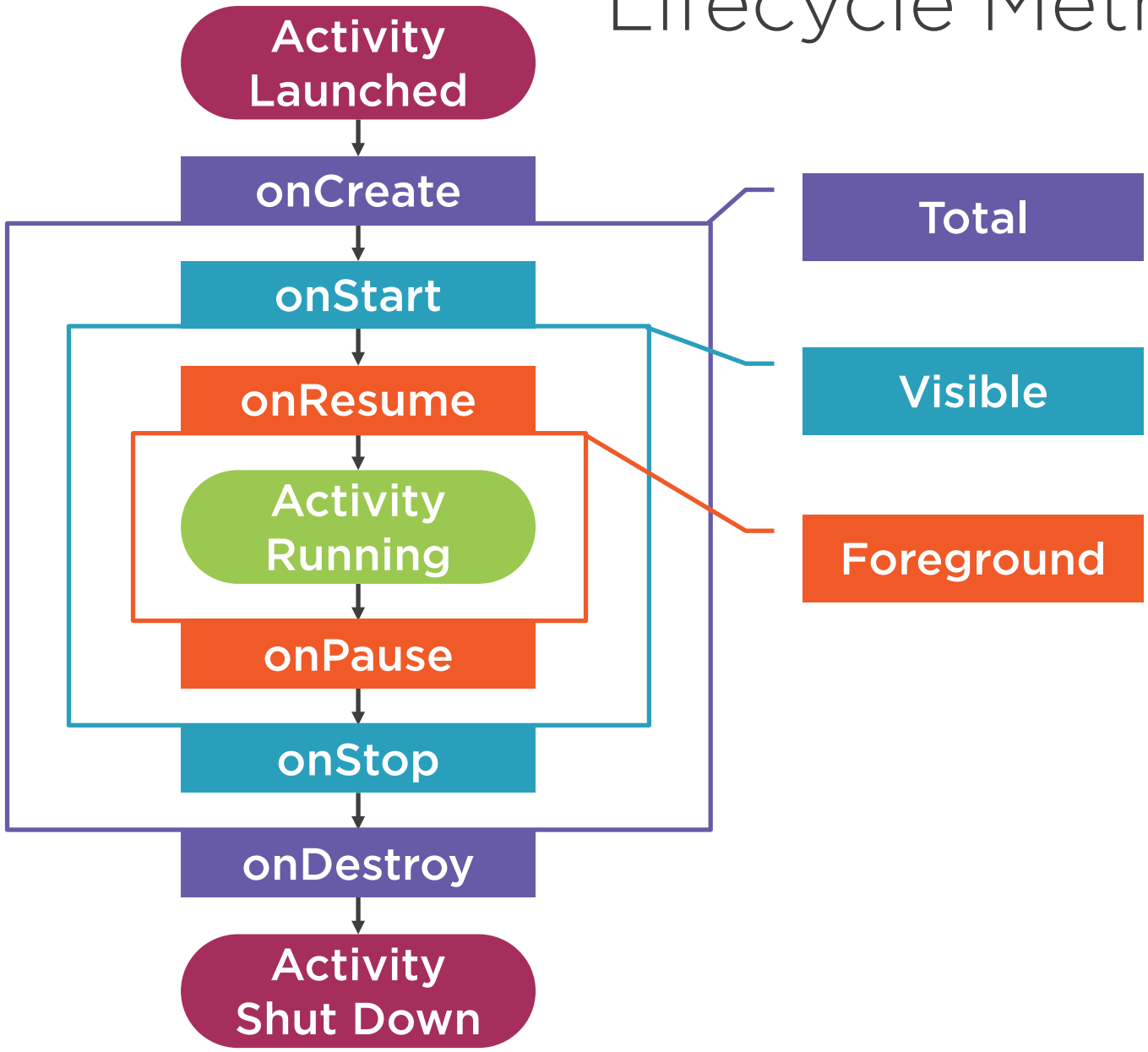
Lifecycle Methods

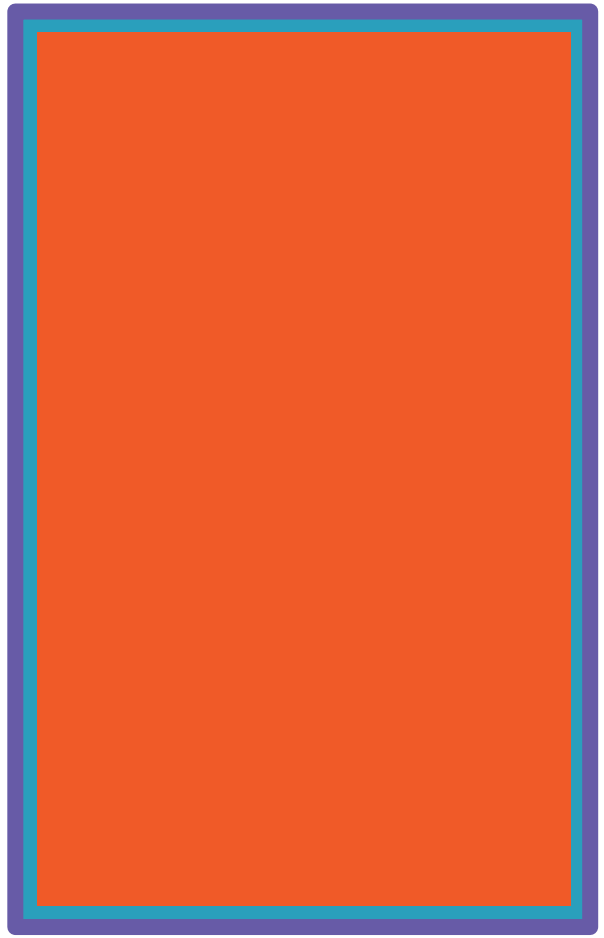


Lifecycle Methods

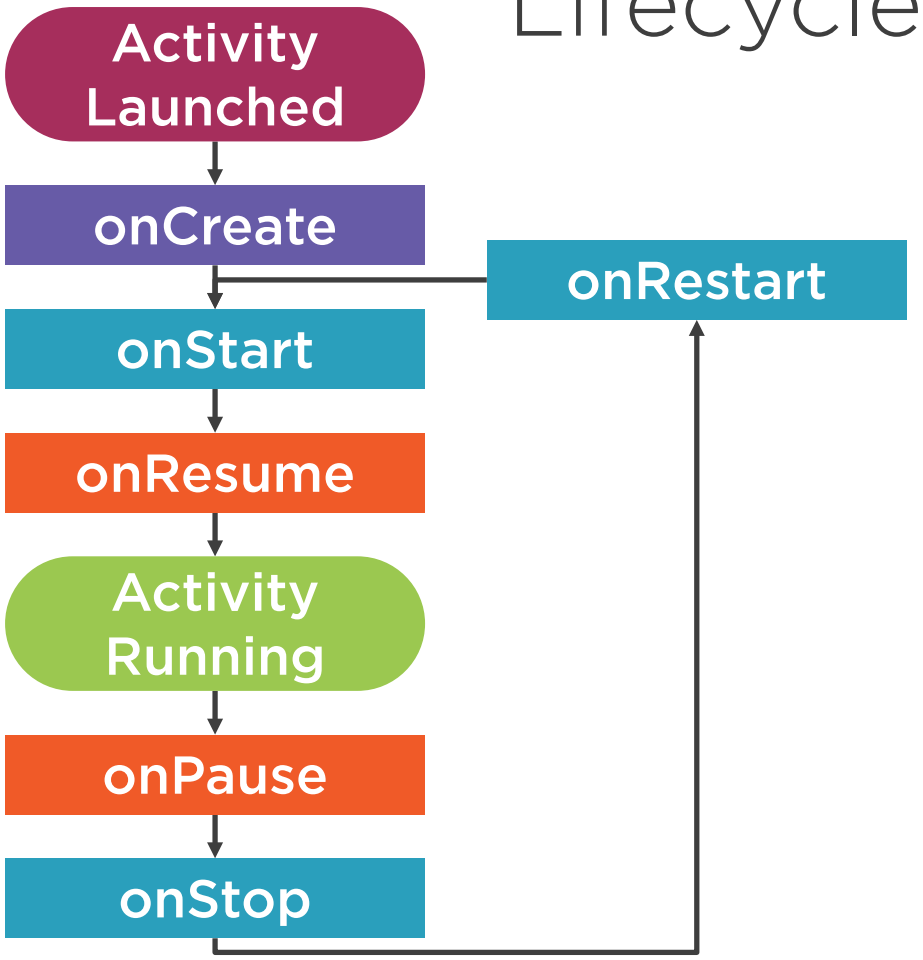


Lifecycle Methods

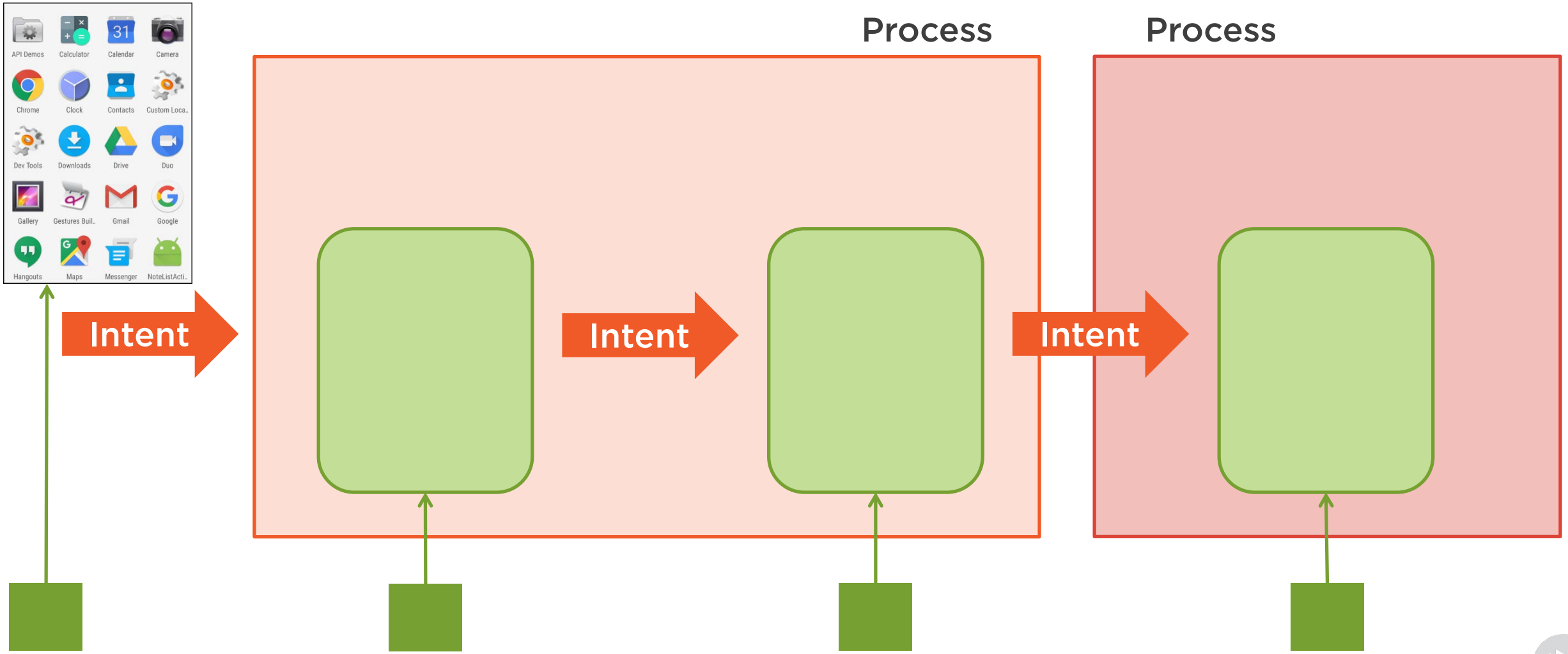




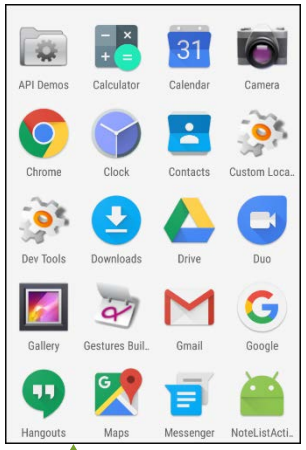
Lifecycle Methods

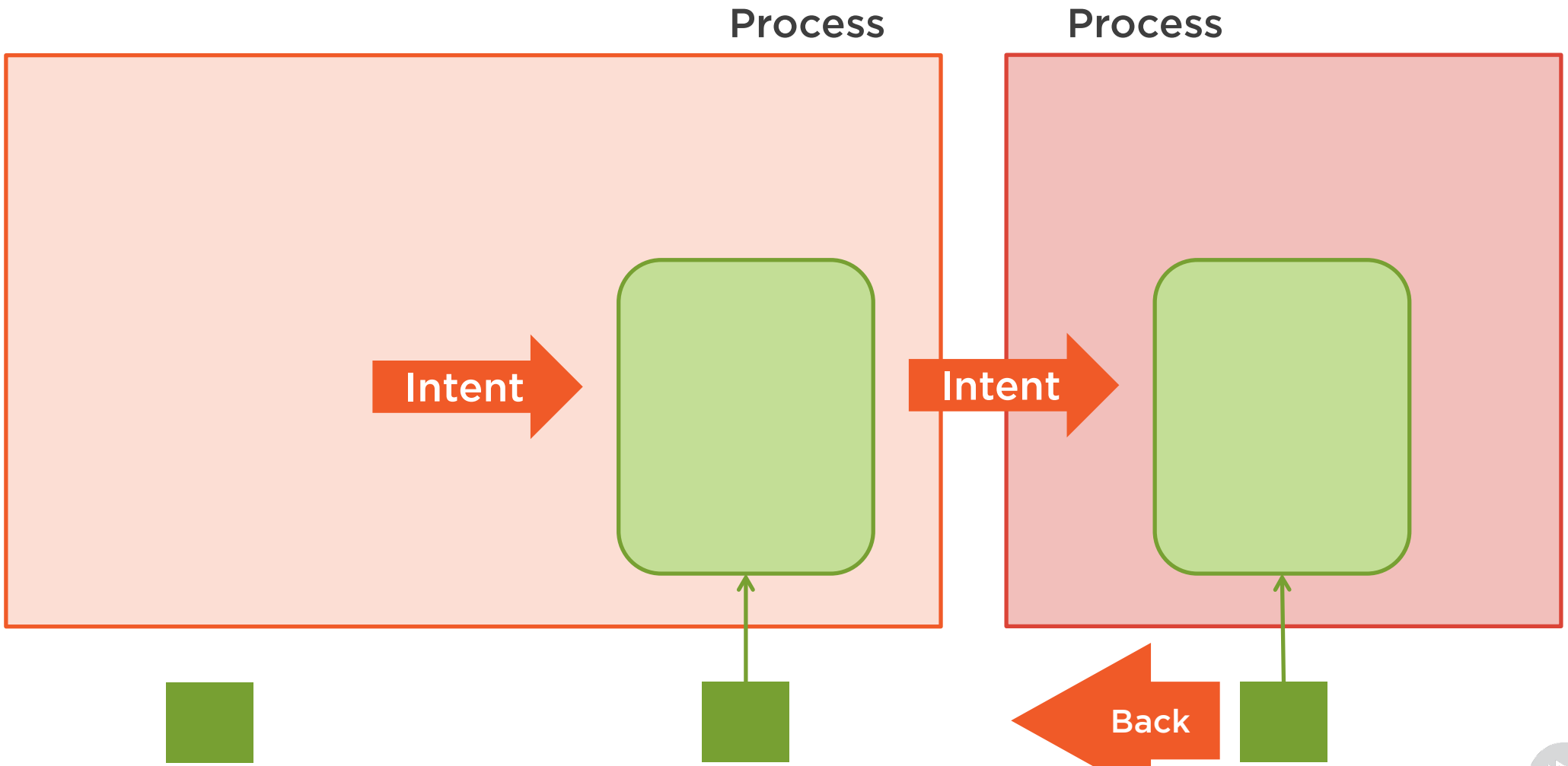


Activity State Management

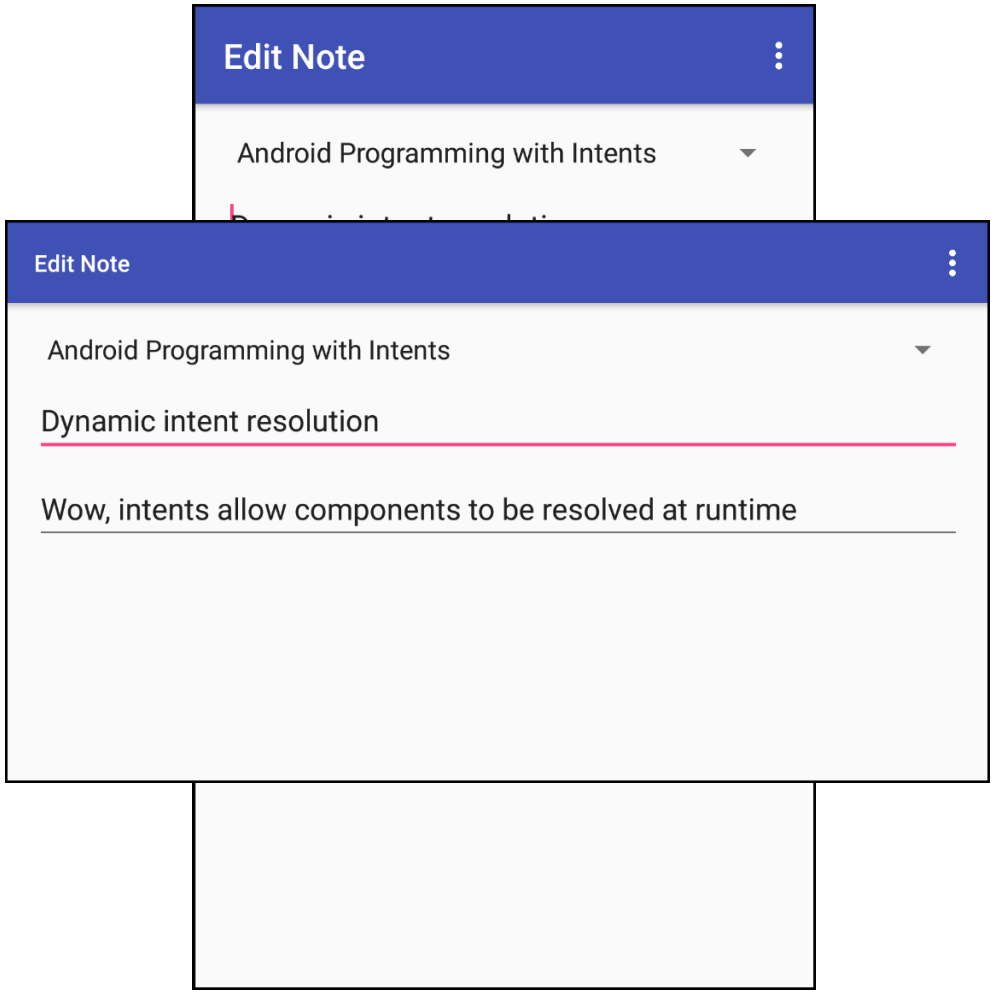


Activity State Management





Activity State Management



Activity State Management

Activities provide state management

- Opportunity to save before destroy
- Saved state provided on restore

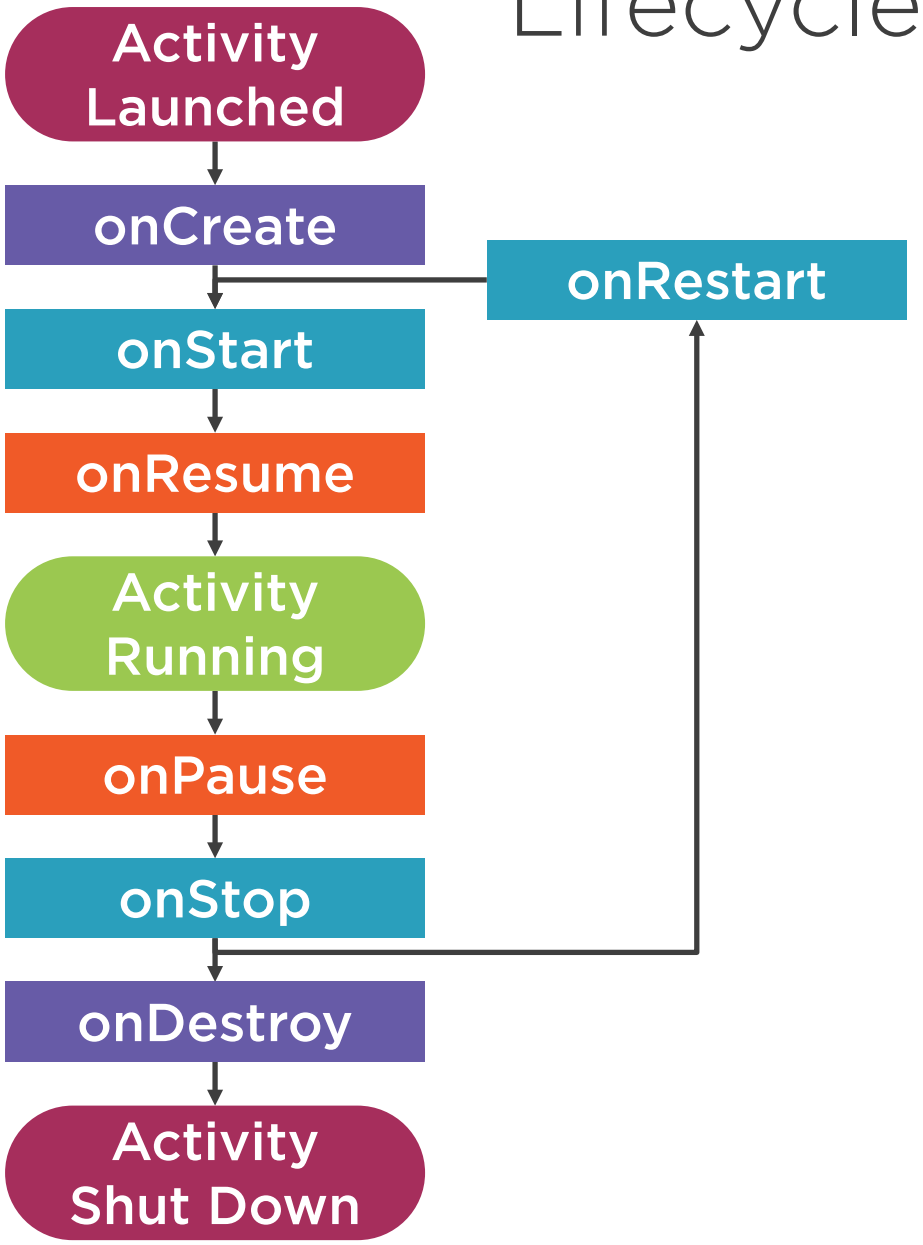
Saving state

- onSaveInstanceState
- Write Activity state to passed Bundle

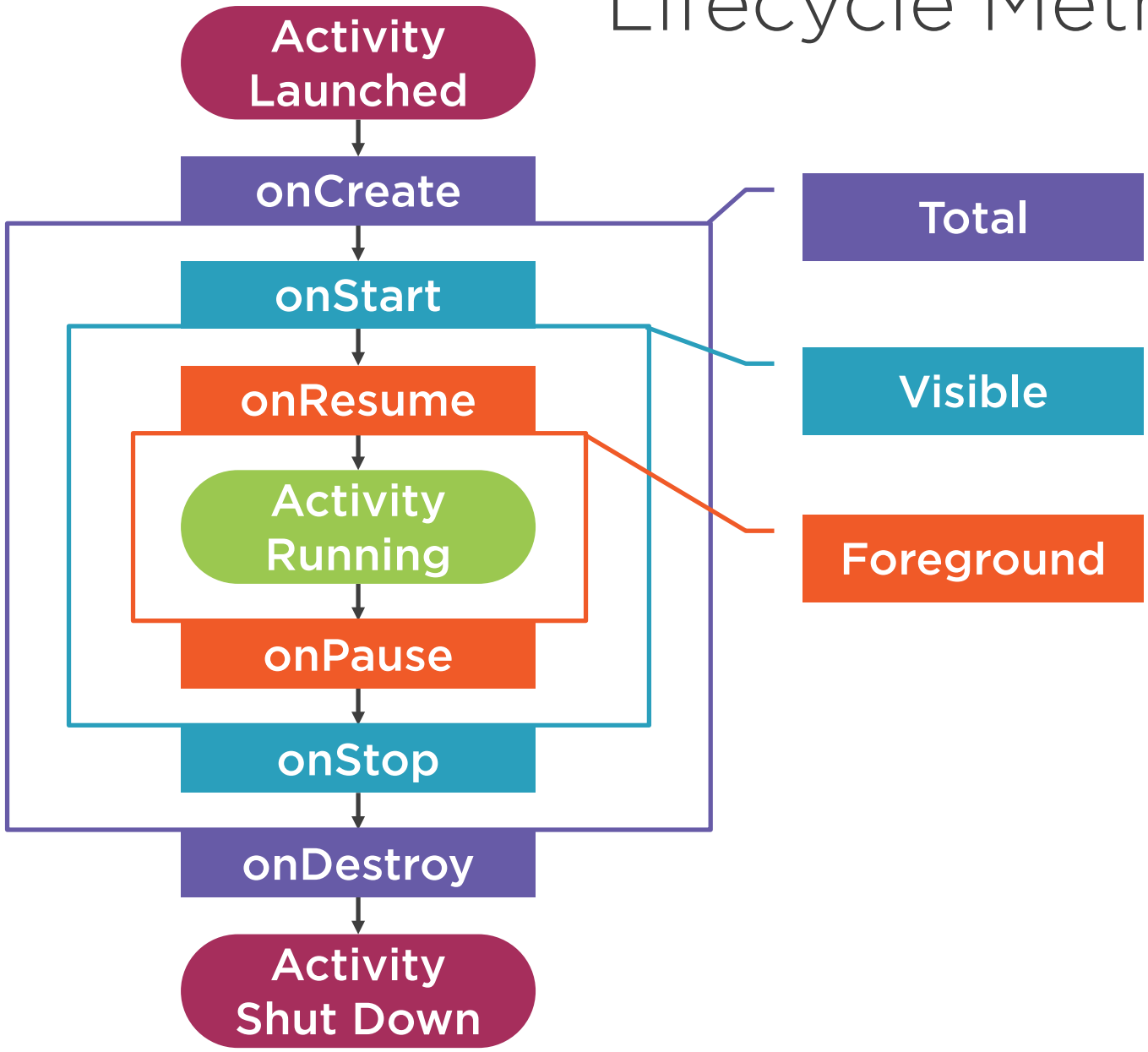
Restoring state

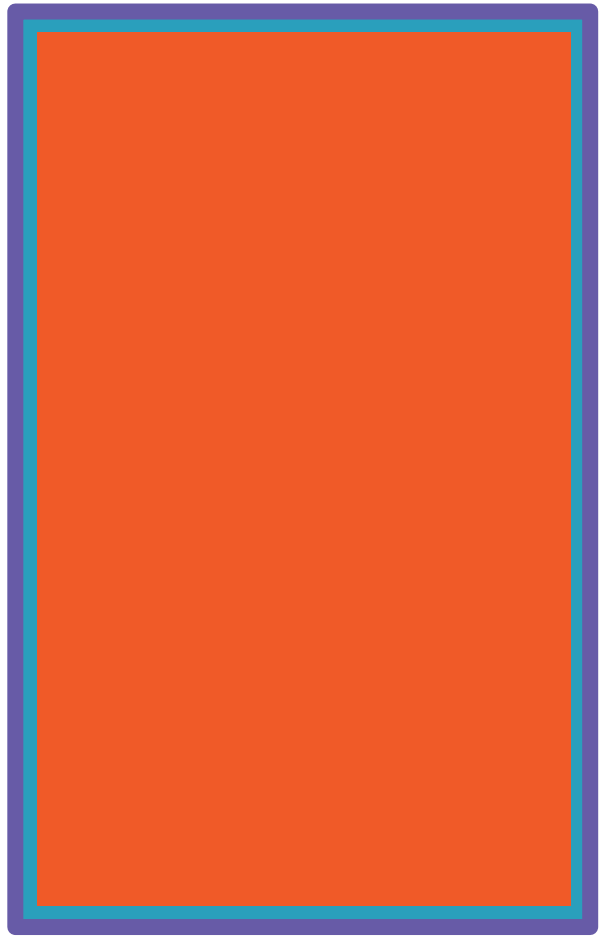
- onCreate
- Receives saved Bundle on restore
- Bundle is null on initial create
- Intent remains available on restore

Lifecycle Methods



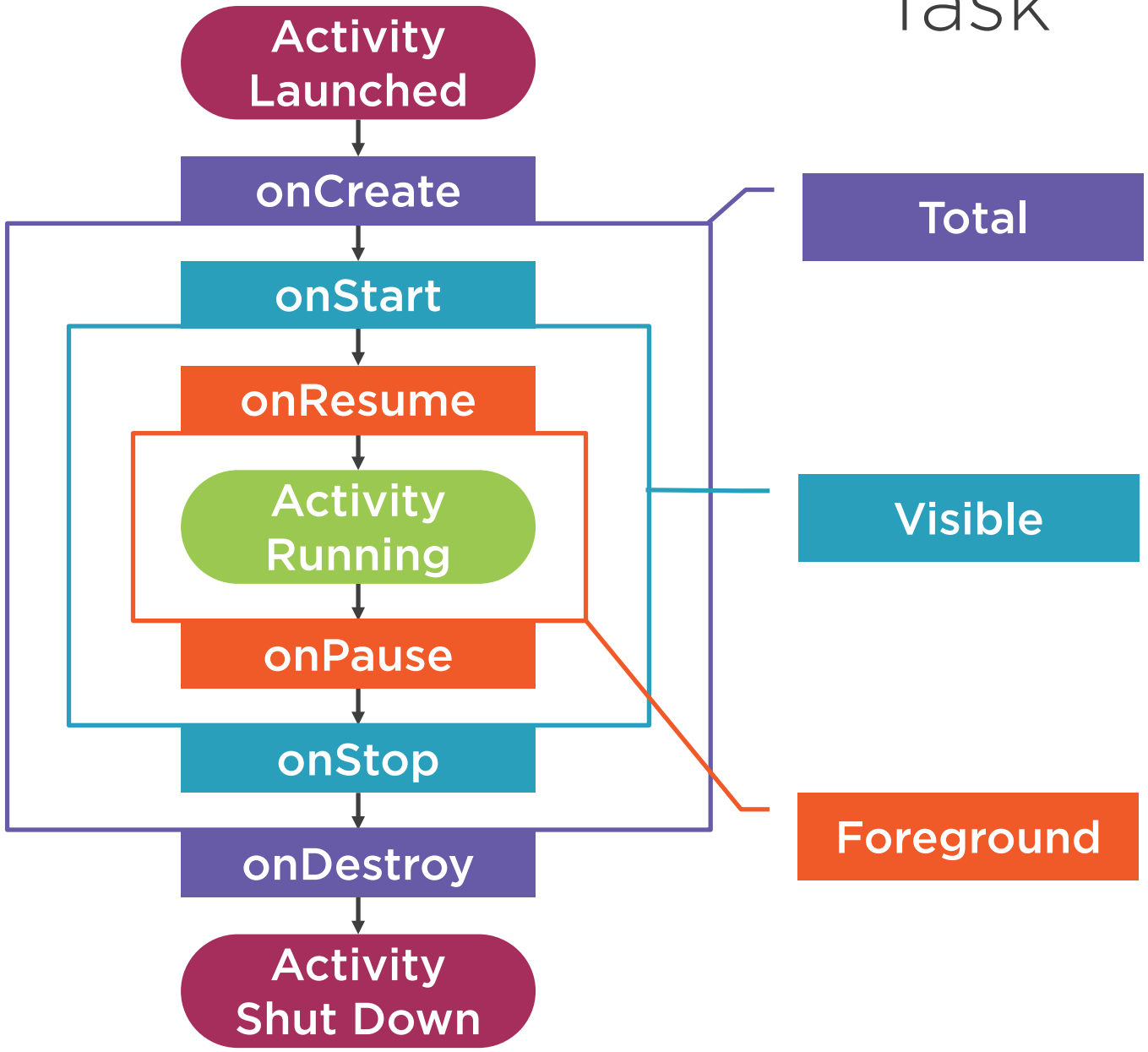
Lifecycle Methods



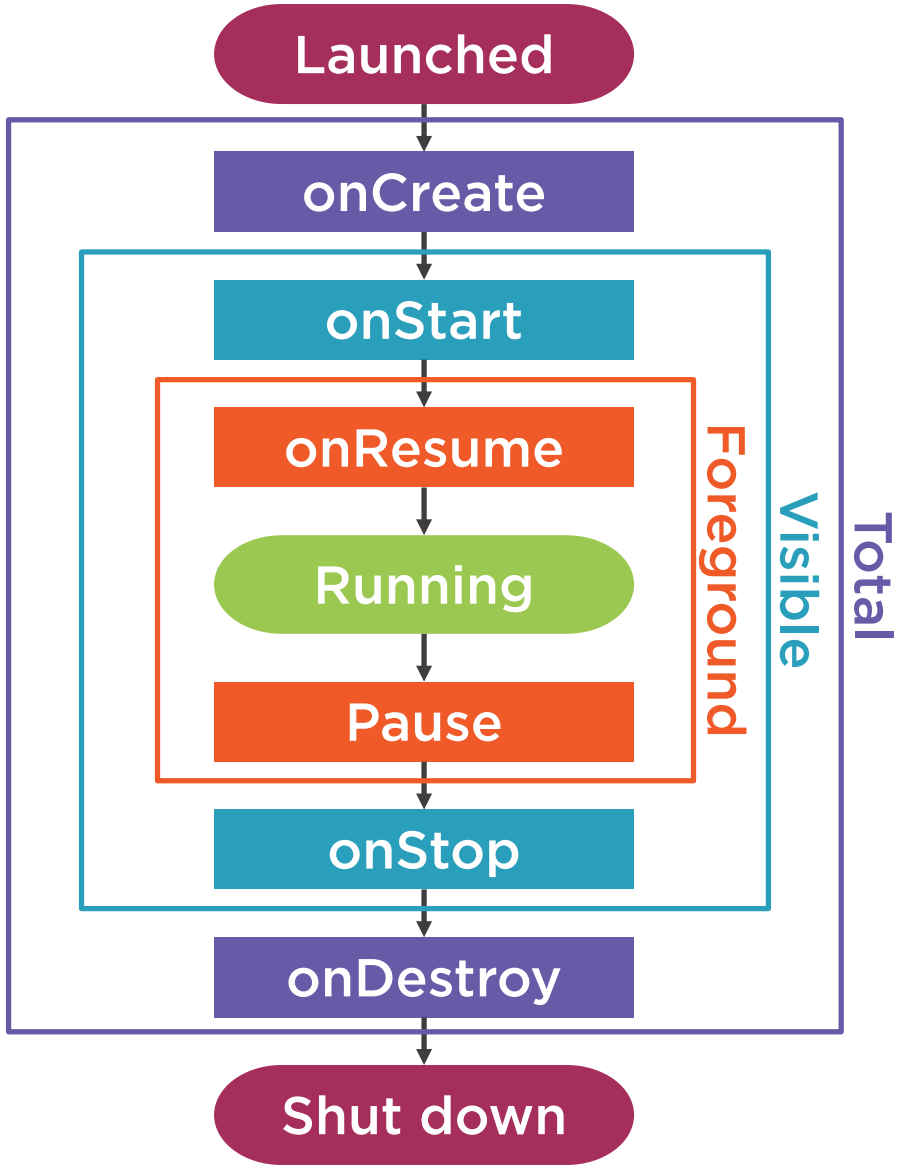


Summary

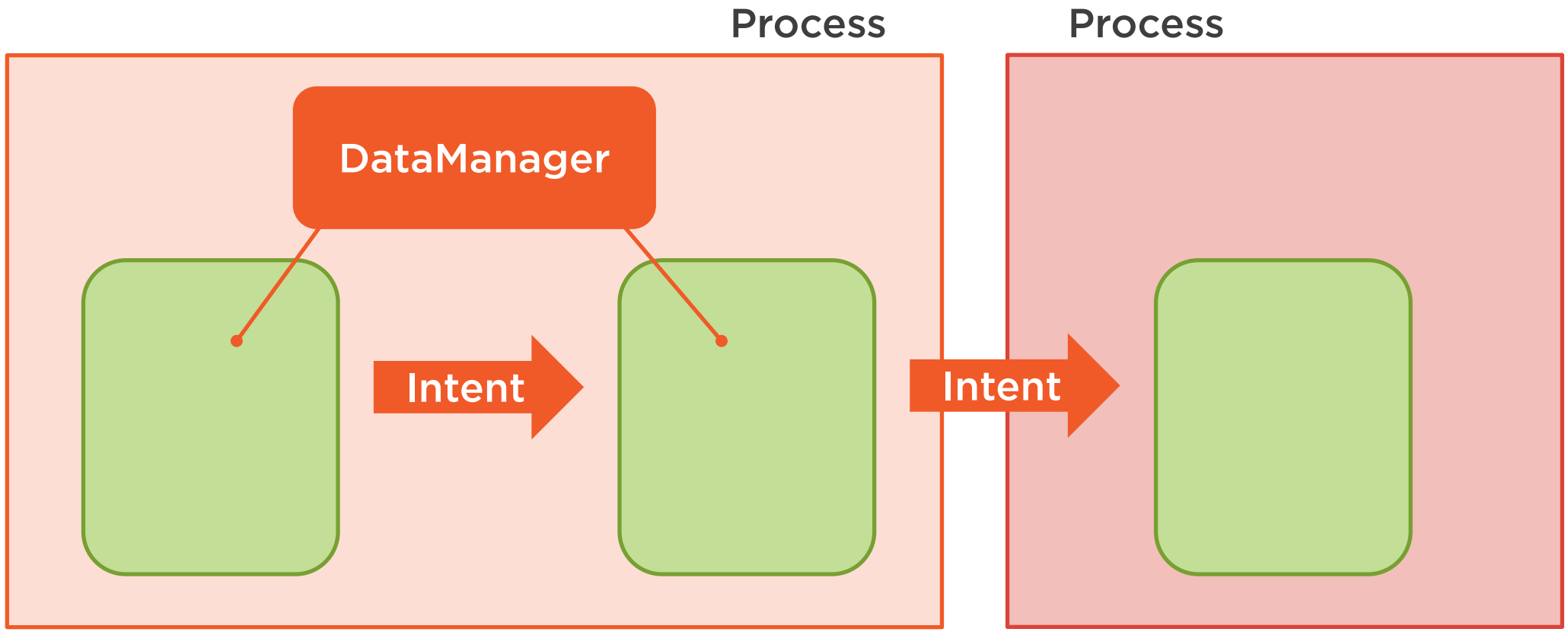
Task


Activity Launched
onCreate
onStart
onResume
Activity Running
onPause
onStop
onDestroy
Activity Shut Down
Total
Visible
Foreground

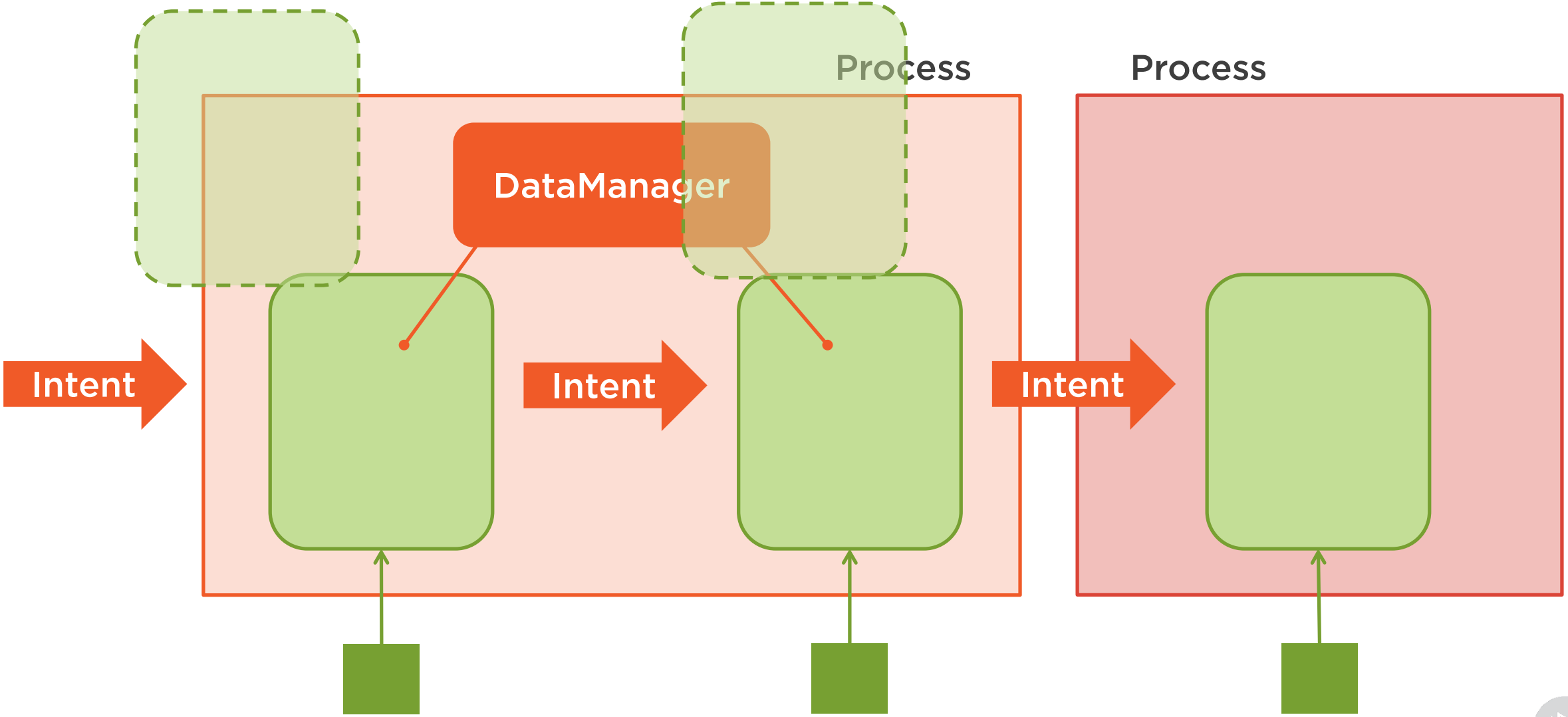
Task


Launched
onCreate
onStart
onResume
Running
Pause
onStop
onDestroy
Shut down
Foreground
Visible
Total

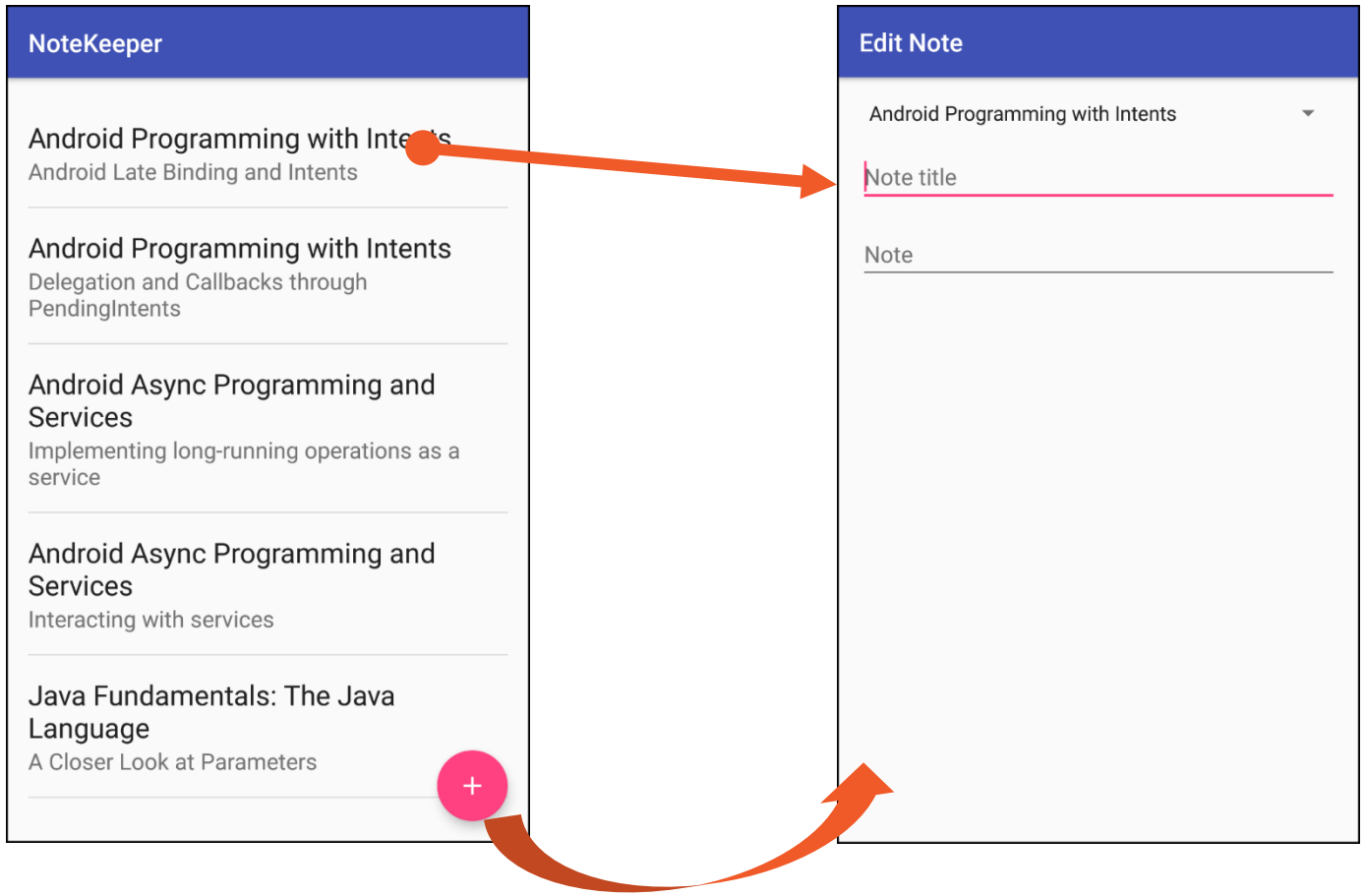
Activity Interaction



Activity Interaction



A Quick Reminder of Our App



Activity Interaction

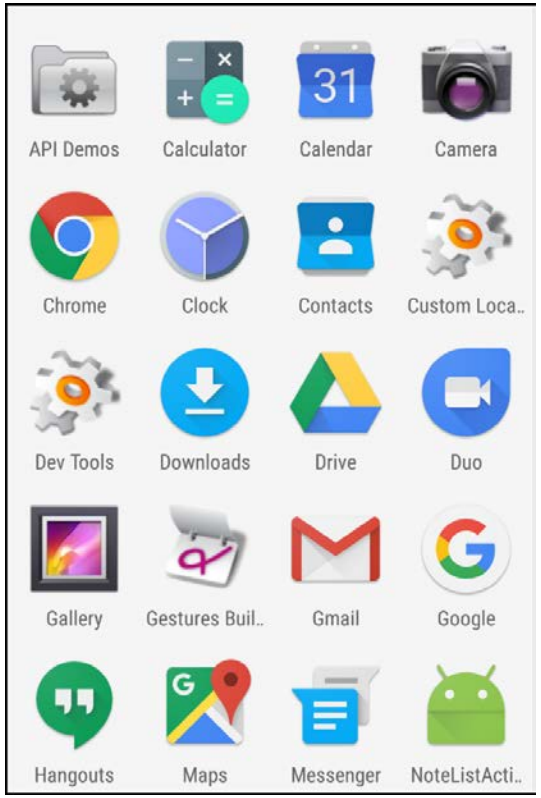
Android is a component-oriented platform

- A number of different types
- Activities are the most familiar

Activities are distinct from one another

- One cannot directly create another
- Rely on intents to interact

Activity Interaction



Intent

Process

Starting an Activity Within Your App

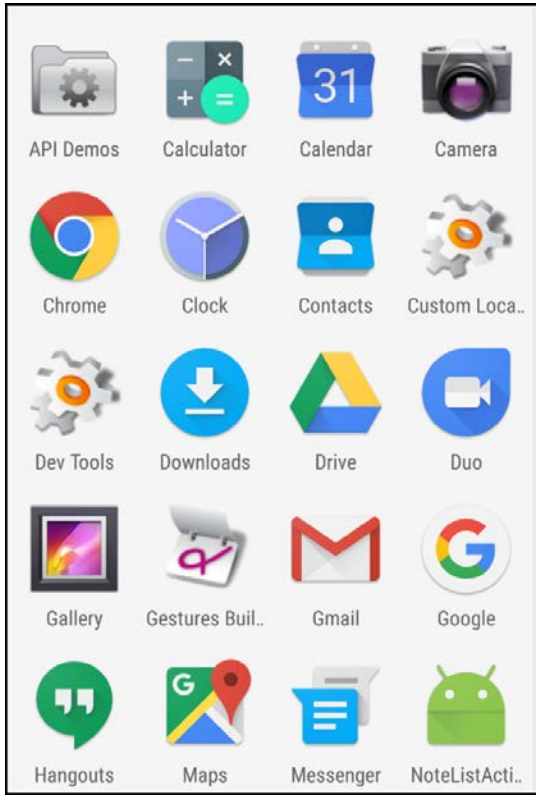
Create an intent

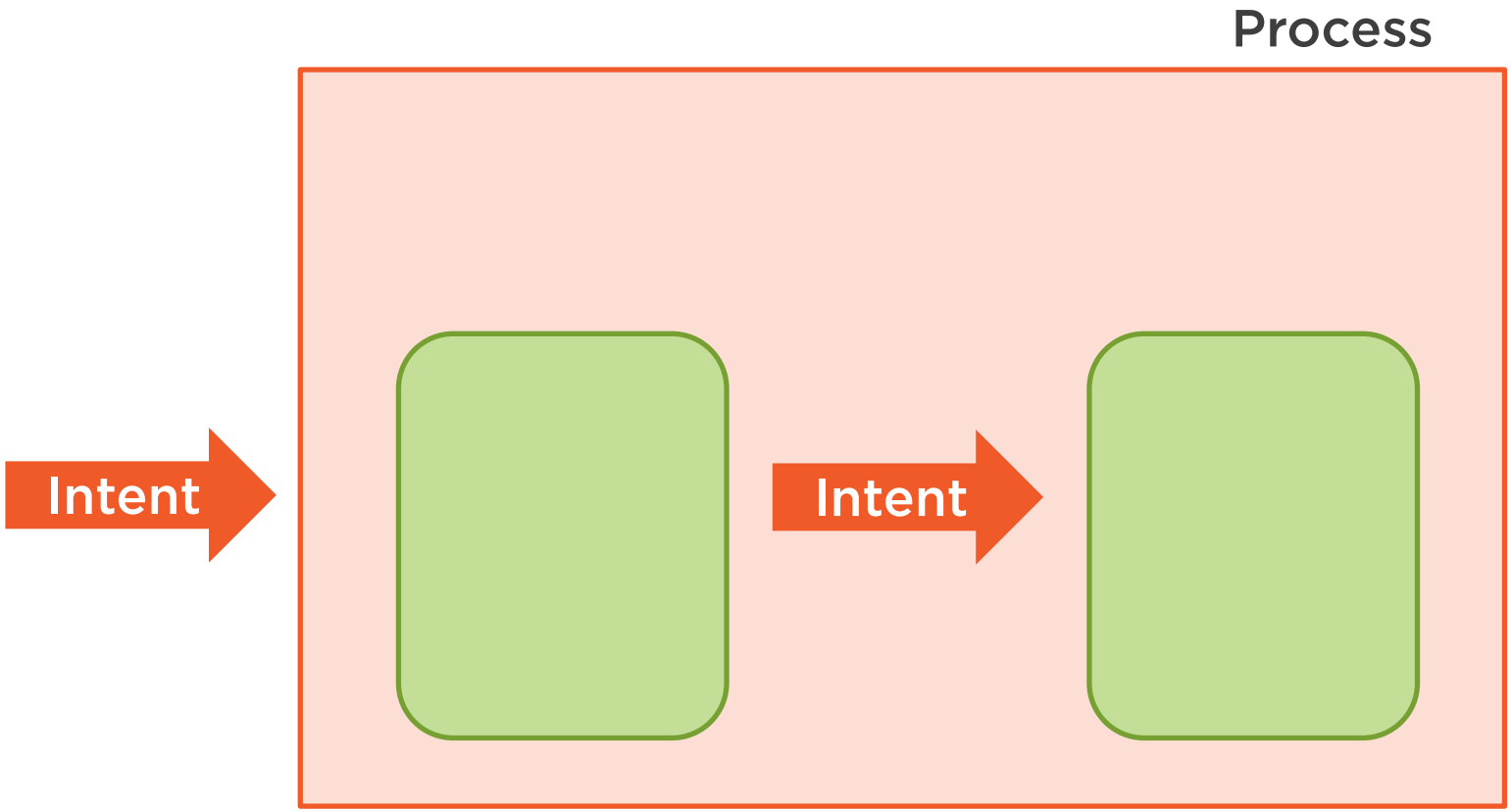
- Identifies the desired Activity
- Often can just be Activity class info

Call startActivity

- Pass the intent
- Launches Activity matching the intent

Activity Interaction





Describing Operations with Intents

Intents describe a desired operation

- Often need more than just a target
- May need to provide additional info

Intent extras provide additional info

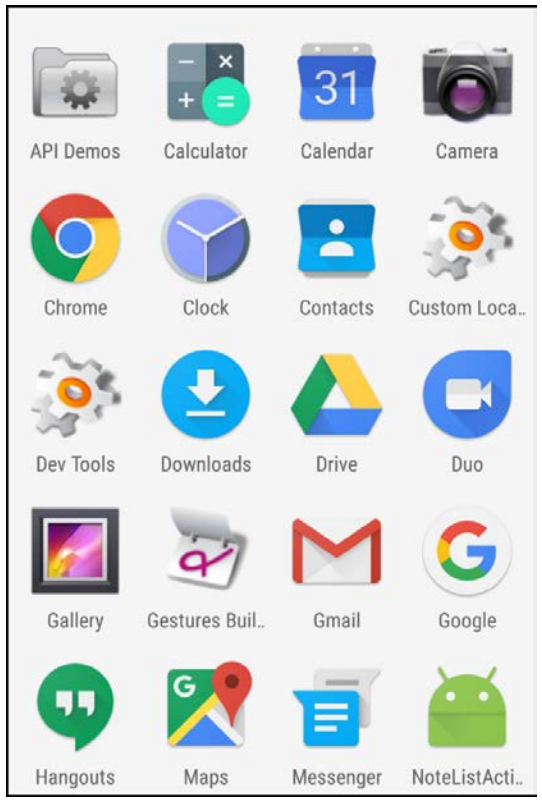
- Name value pairs
- Names & values are operation-defined
- Added to intent with putExtra overloads

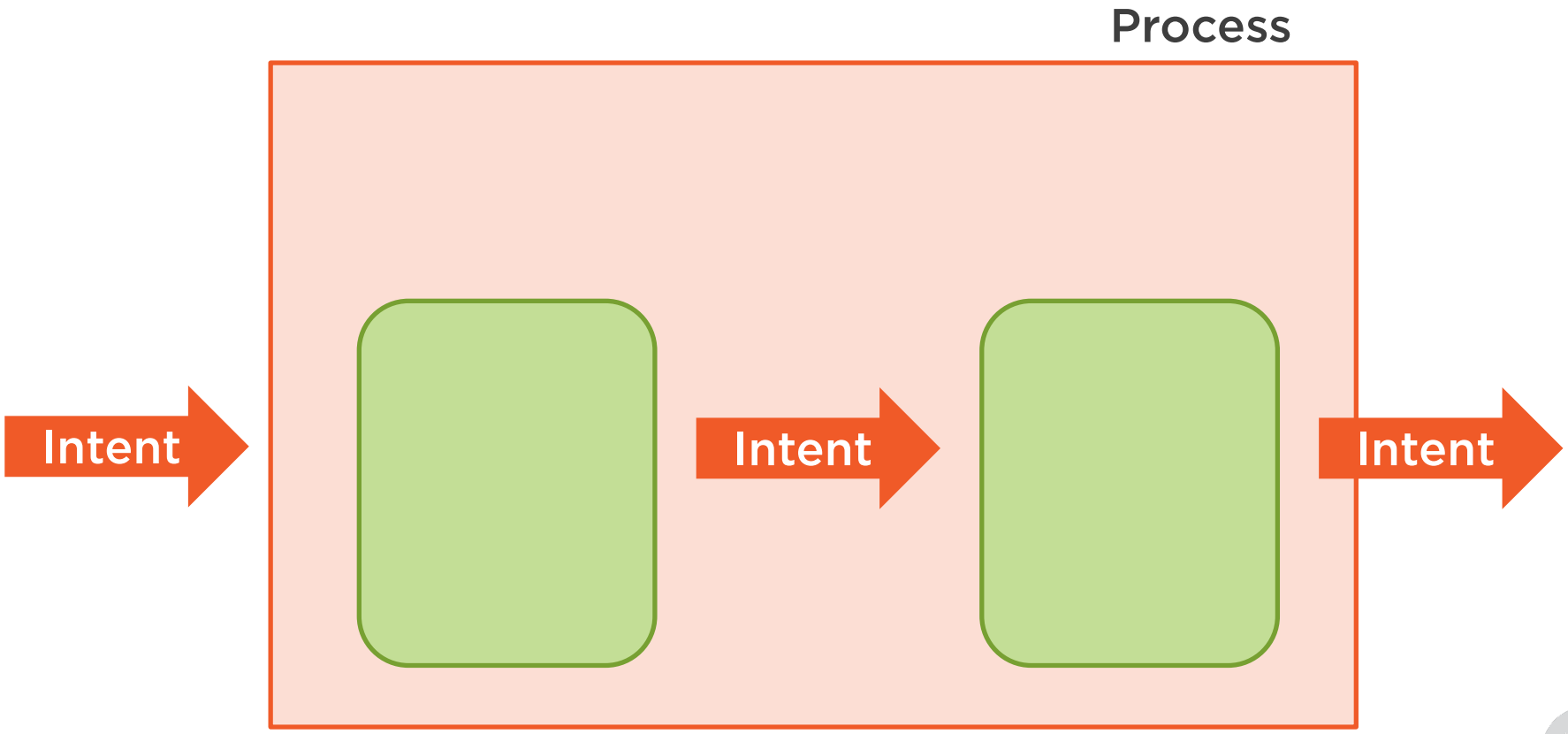
Describing Operations with Intents

Accessing intent info

- Activity can access intent that started it
- Use getIntent
- Use Intent.getXXExtra to retrieve extras
  - Method names include return type

Describing Operations with Intents





Describing Operations with Intents

Intents must be cross-process friendly

- Limits allowable extras

Supported extra types

- Primitive types and String
- Arrays of supported types
- Some ArrayLists
- A few other special types

Most reference types not directly supported

- Require special handling

Reference Types as Intent Extras

Reference types need to be “flattened”

- Converted to a bunch of bytes

One option is Java serialization

- Supported but not preferred
- Serialization is very runtime expensive

Use Parcelable API

- Much more efficient than serialization
- Must be explicitly implemented

Reference Types as Intent Extras

Implement Parcelable interface

- describeContents
  - Indicates special behaviors
  - Generally can return 0
- writeToParcel
  - Receives a Parcel instance
  - Use Parcel.writeXX to store content

Reference Types as Intent Extras

Provide public static final CREATOR field

- Is a Parcelable.Creator implementation

Parcelable.Creator interface

- createFromParcel
  - Responsible to create new type instance
  - Receives a Parcel instance
  - Use Parcel.readXX to access content
- newArray
  - Receives a size
  - Responsible to create array of type

Show android image

- Launcher
- Say that launcher indicates desire to start Activity
- Android starts process containing activity
- Creates Activity

Starting Activity

- Intents
- Implicit vs. Explicit

Intents are used to activate components

- Contains component identifying info

Manifest helps launch

Demo

- Show manifest
- Create ListActivity
- Show NoteActivity

Passing information with Intents

Parcalable

Understanding lifecycle events

Demo

- Use diagram to show challenge created by passing Note
- Switch to passing as integer

Android is a component-oriented platform

- A number of different types
- Activities are the most familiar

Components have a well-defined lifetime

- Components drive process lifetime

Traditional Process-oriented Model

```
java com.jwhh.cmdline.Main
```

Process

```
Class Main {
  public static void main() {
    // do some stuff
  }
}
```

What is an Activity?

An activity is a single, focused thing that the user can do.

A Little More Detail About Activities

Serve as the place to present the UI

- Provide a window
- UI is built with View-derived classes

Have a lifecycle

- More than just a “screen”
- Lifecycle calls a series of methods
- Use the onCreate method to initialize the Activity

Activity UI

View

- Basic building block of UI
- Drawing and event handling
- Many specialized classes available

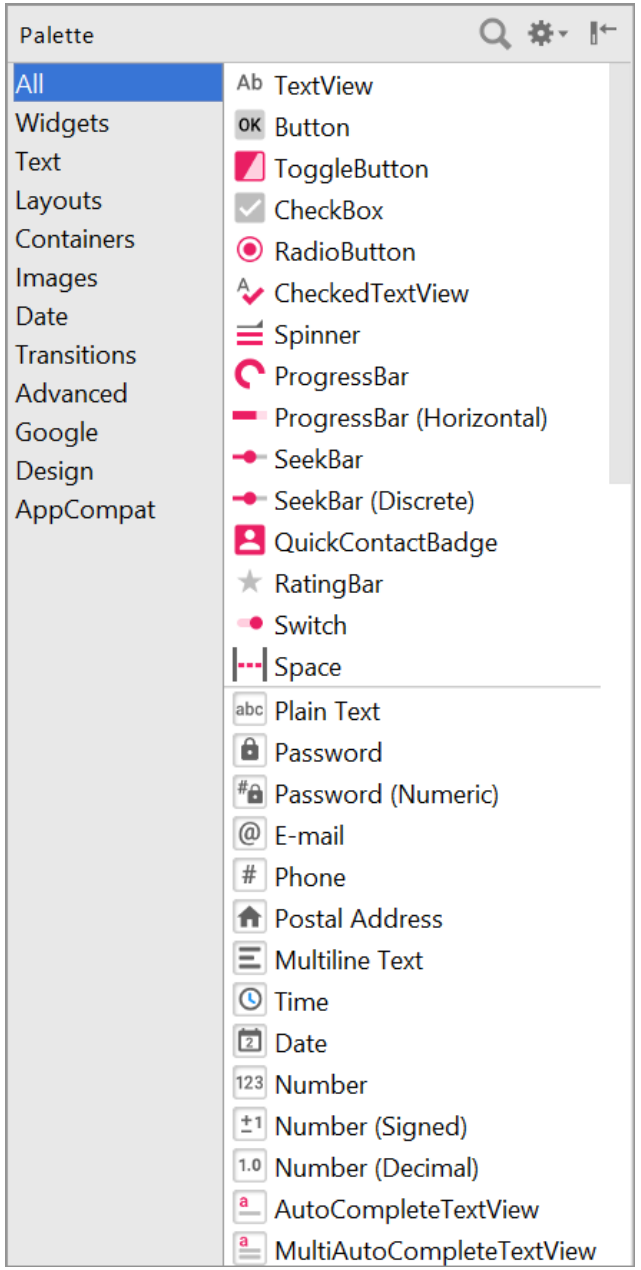
ViewGroup

- Special View that holds other views

Layout

- Special invisible ViewGroup
- Handle View positioning behavior
- Many specialized classes available

Common View Classes



Layout Classes

Activity UIs need to be responsive

- Device display characteristics vary
- UI must adapt
- Absolute positioning would be limiting

Layout classes provide positioning flexibility

- Arrange child Views
  - Children can include other layout classes
- Specific positioning behavior depends on the layout class

Some Common Layout Classes

FrameLayout

- Provides a blocked-out area
- Generally has only one direct child

ScrollView

- Provides a scrollable area

LinearLayout

- Horizontal or vertical arrangement
- Supports weighted distribution

RelativeLayout

- Relative positioning
- Relative to one another or parent

Simplifying Layout Creation

Traditional layout classes have challenges

- UIs have become much richer

Run-time challenges

- Sometimes have to nest layout classes
- Deep/complex nesting impacts speed

Design-time challenges

- Achieving desired result with designer sometimes challenging
- In some cases end up working in the XML

ConstraintLayout Class

Extremely flexible layout class

- Often the only layout class needed

First-class design-time experience

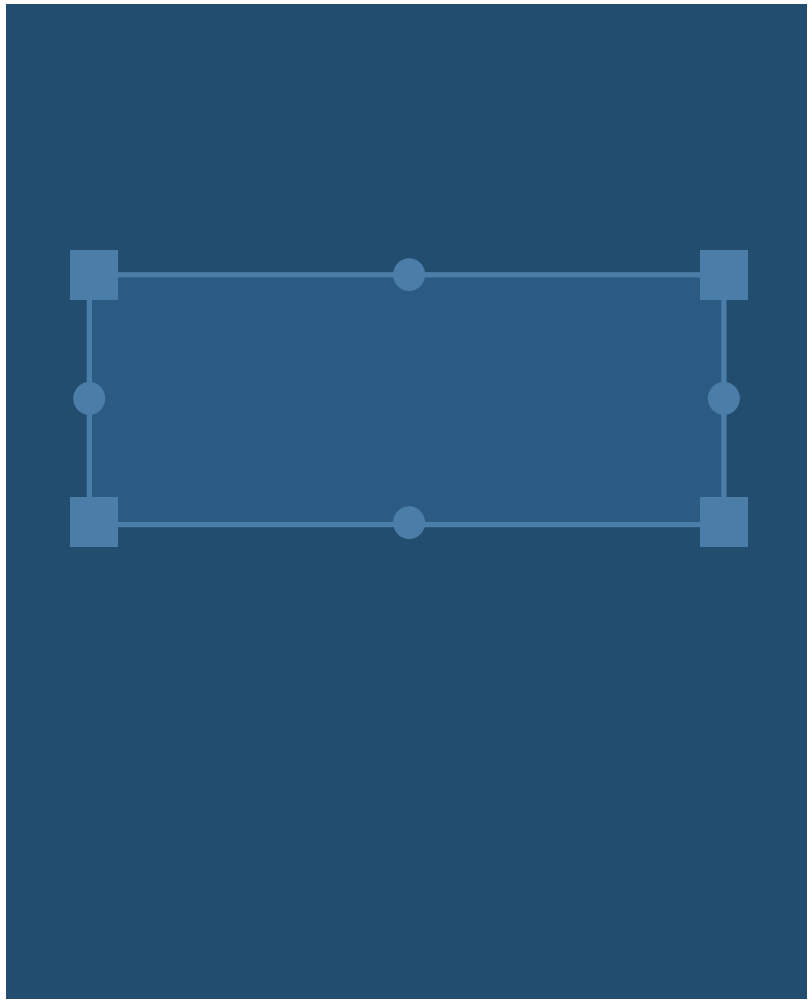
- Closely integrated with designer
- Rarely need to resort to XML

ConstraintLayout Class

Children leverage constraints

- Relative size/position
- Ratio-base size/position
- Group size/position distribution
  - Known as “chains”
- Weighted relationships
- Guideline-based size/position

ConstraintLayout



Should set horizontal & vertical constraints

- Positions at 0,0 without constraints
- Can set more than one of each

Setting constraints with the designer

- Drag circle at mid-line to relationship

Setting fixed size with the designer

- Drag corner squares

Creating the Activity UI

Programmatically

- Use Java code to create class instances
- Relationships and properties set in code

Layout files

- XML files describe View hierarchy
- Usually created using the Android Studio UI Designer

Activity/Layout Relationship

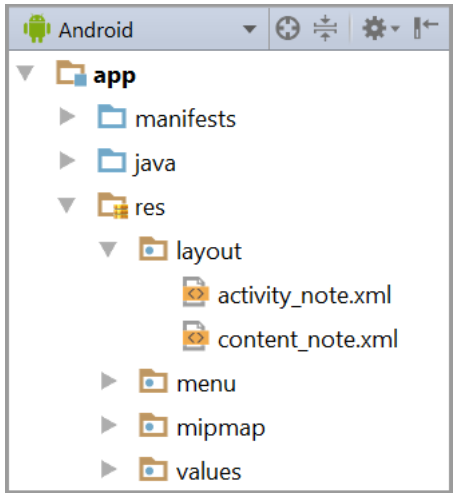
There is no implicit relationship

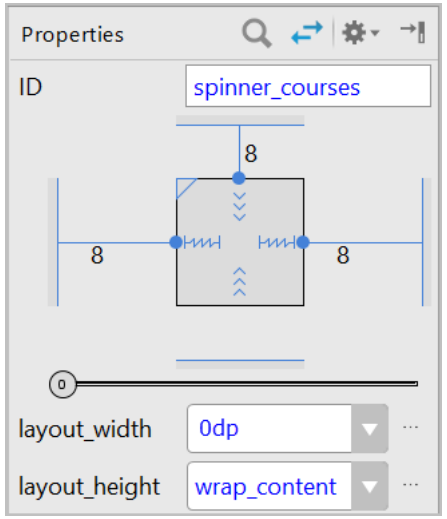
- Activity must load layout
  - Use setContentView
- Activity must request View reference
  - Use findViewById

Relies on generated class R

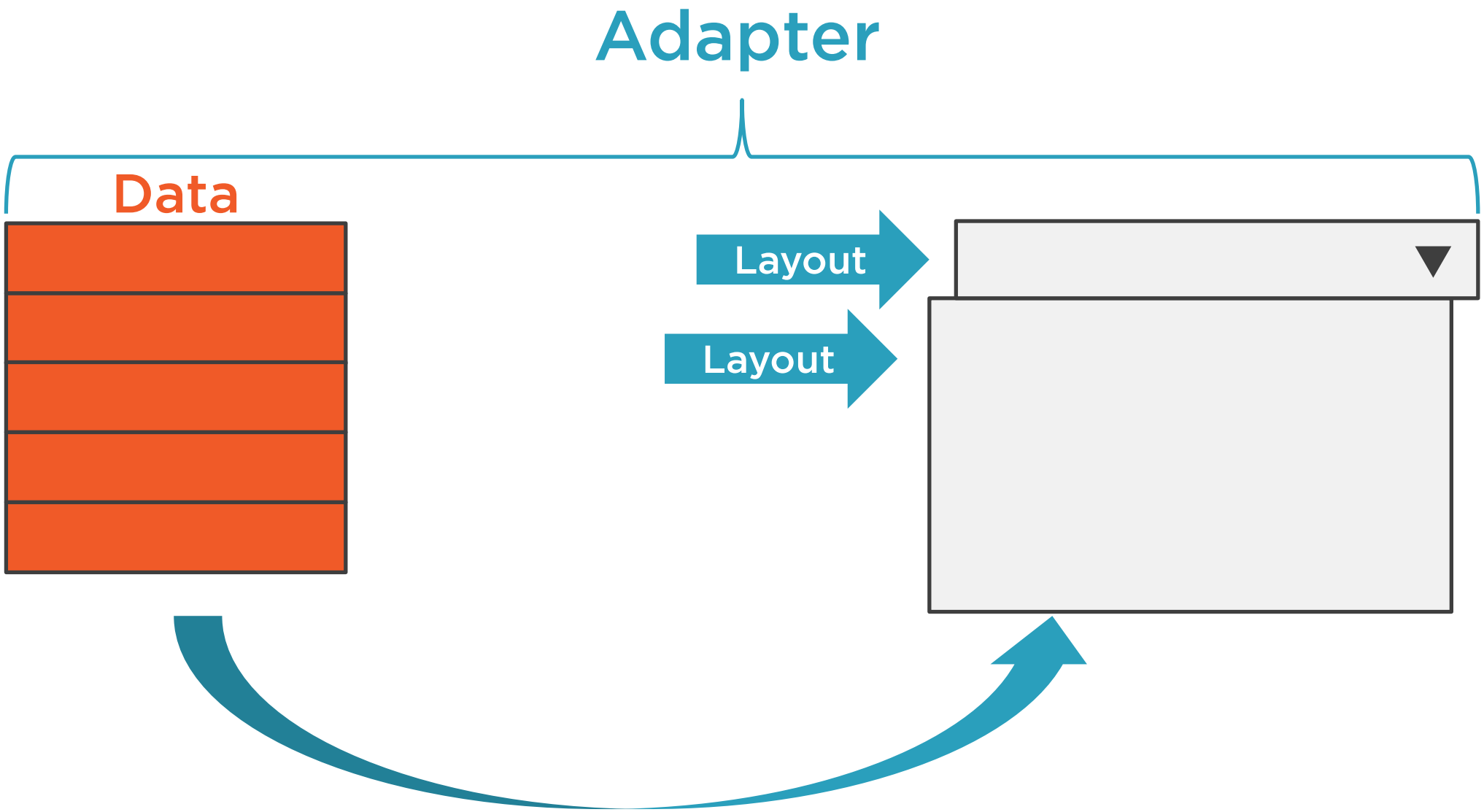
- Contains nested classes
- Layouts names in R.layout
- Id names in R.id

```
public final class R {
  public static final class layout {
    public static final
        int activity_note = ... ;
    // ...
  }
  public static final class id {
    public static final
        int spinner_courses = ... ;
    // ...
  }
}
```





Populating a Spinner



Summary

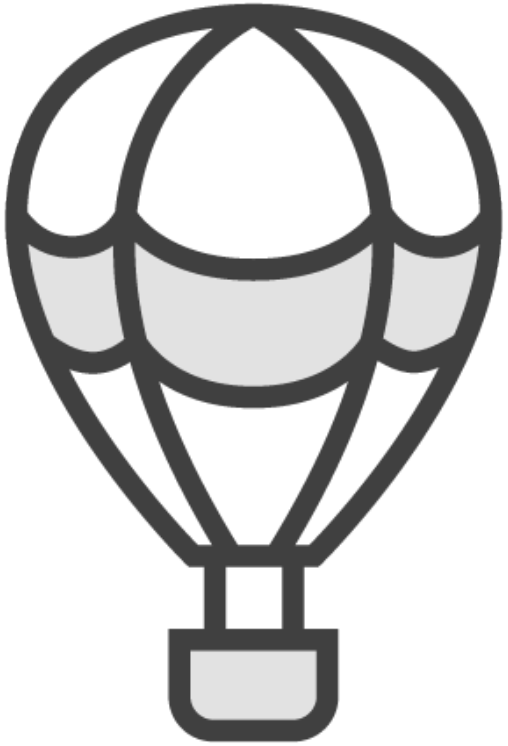
Activity/layout relationship

- No implicit relationship exists
- Must load layout
  - Use setContentView
- Must request layout View references
  - Use findViewById

R class provides important constants

- Layout resources
  - R.layout
- View Id's
  - R.id

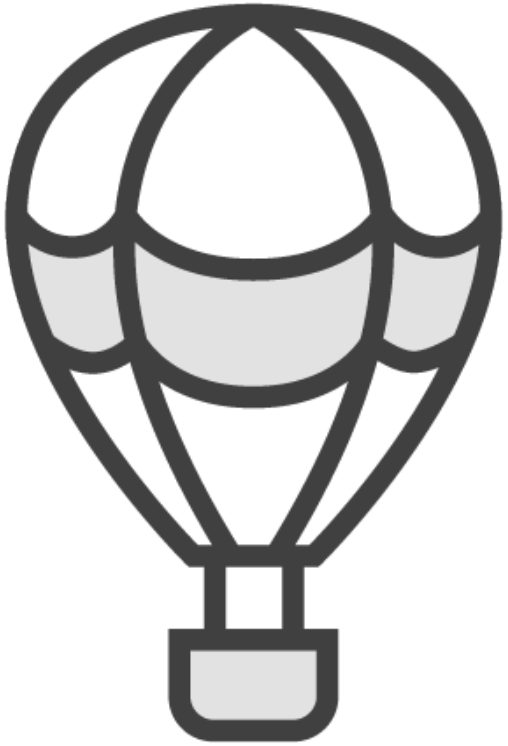
Our App – The High-level View



We'll be building a note keeper app

- Display list of notes
- Edit existing notes
- Create new notes

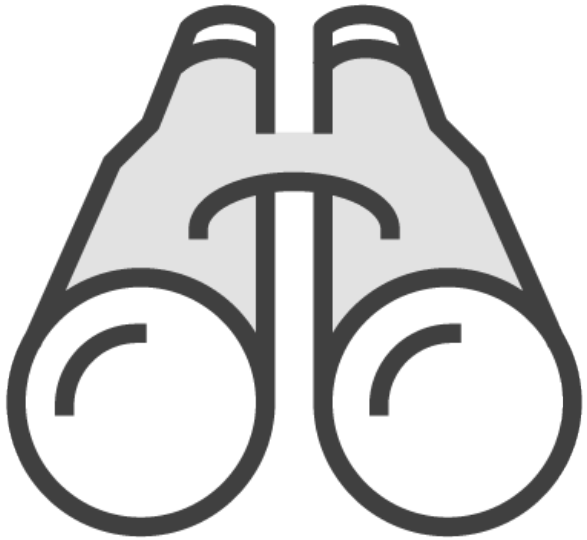
Our App – The High-level View



Our app's definition of a "note"

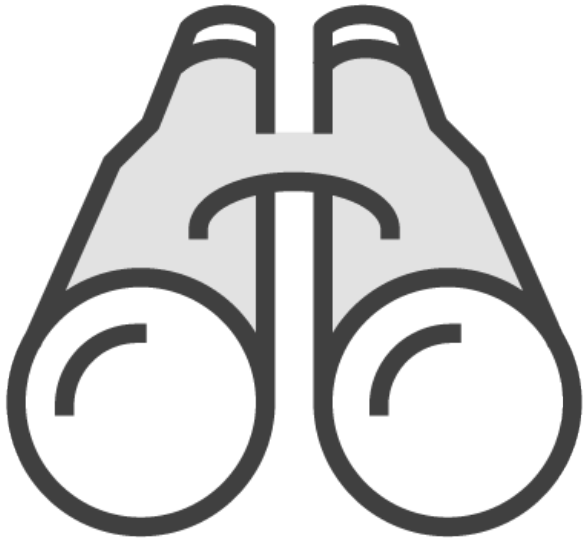
- Course
  - Name of a course
  - Must correspond to an actual course
- Note title
  - User entered value to identify note
- Note text
  - User entered value that serves as the content of the note

Our App – Some Long-term Plans



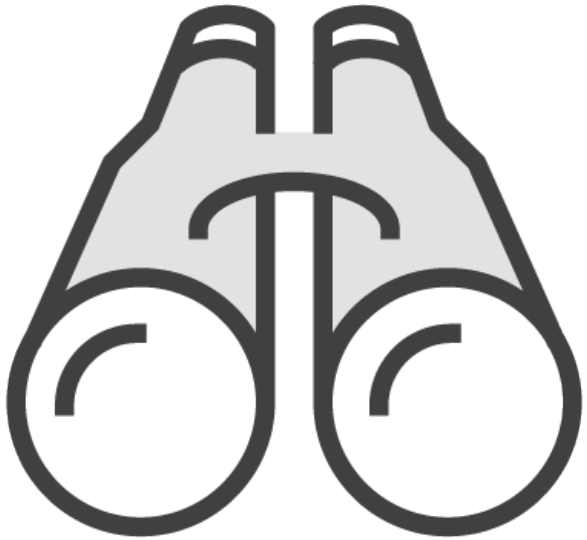
- Automated testing of logic and UI/UX
- Card-style lists
- Slide-out drawer navigation

Our App – Some Long-term Plans



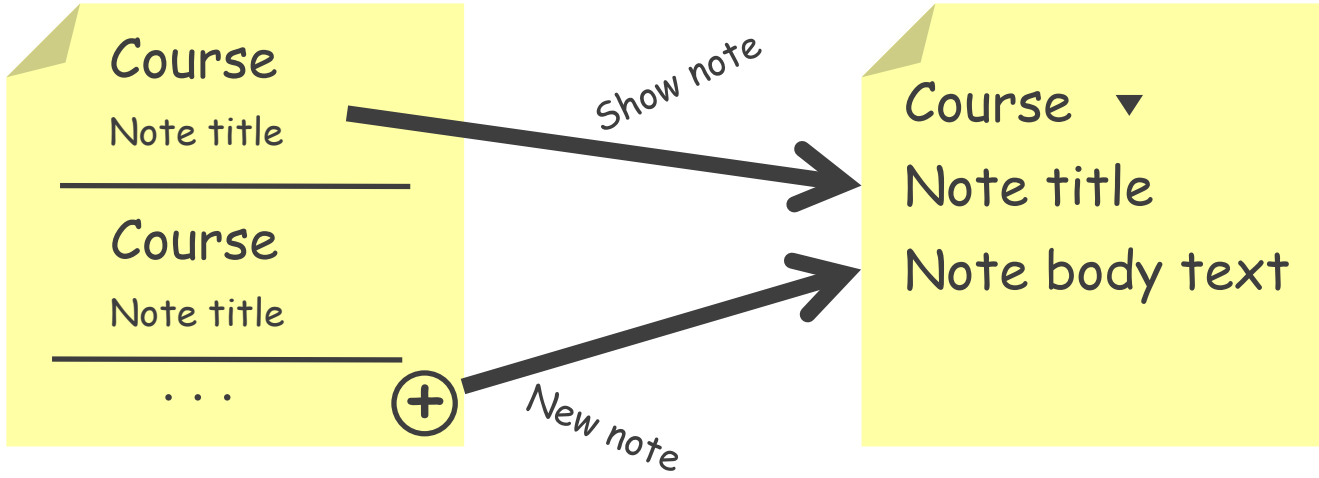
- Track completed modules
- Branding-based UI theme
- Device and language adaptability
- Support for users with accessibility needs
- User customizable behaviors

Our App – Some Long-term Plans

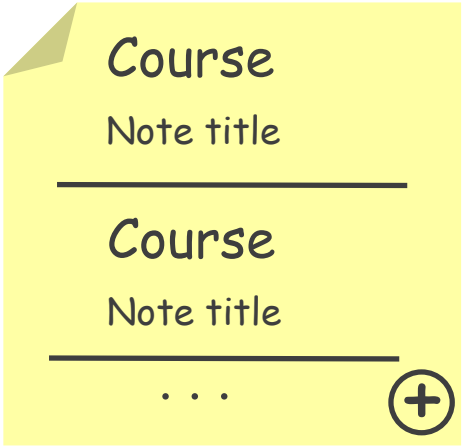


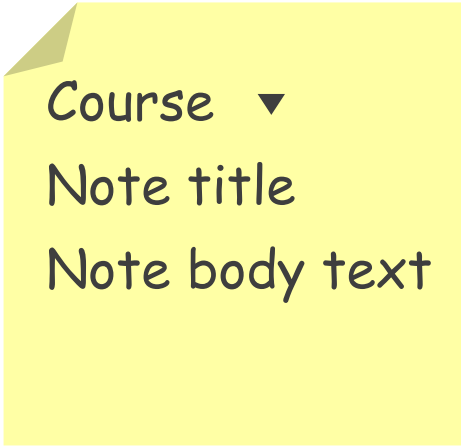
- Use SQLite to store and access our data
- Make note data available to other apps
- Display reminder notifications
- Read/save data in the background
- Display note information on home screen

Rough Design of Our App Starting Point



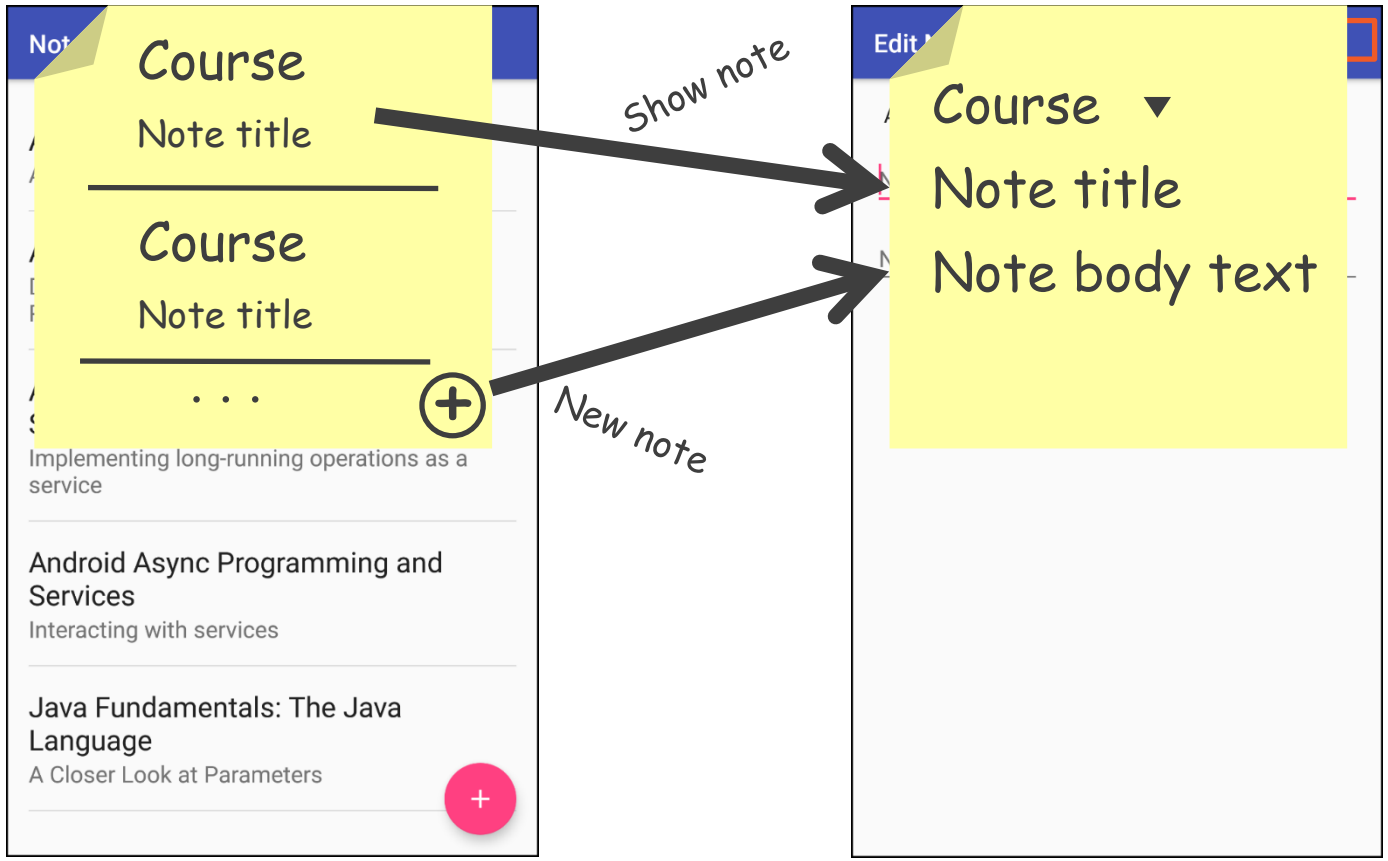
Polished Design of Our App Starting Point

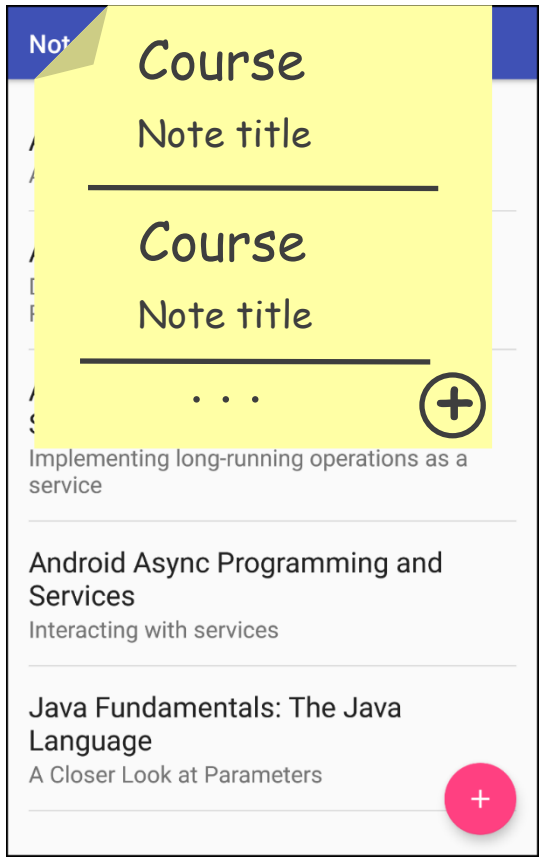


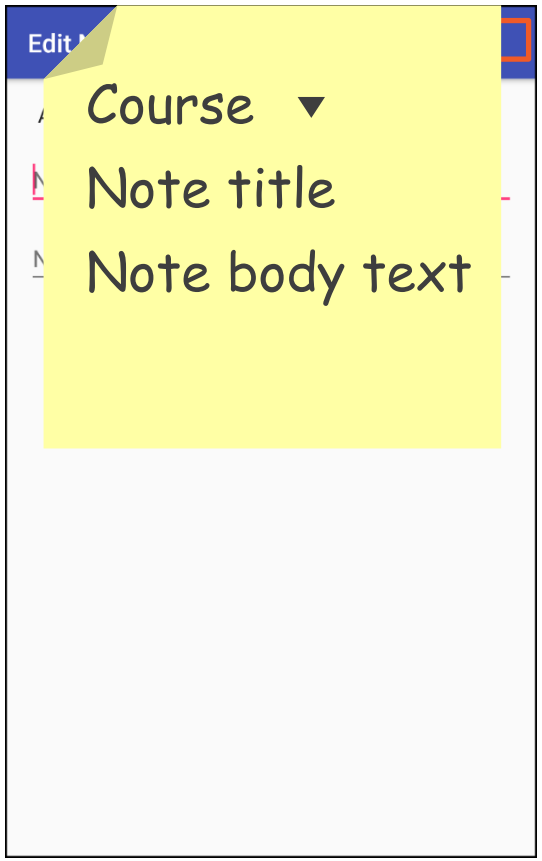


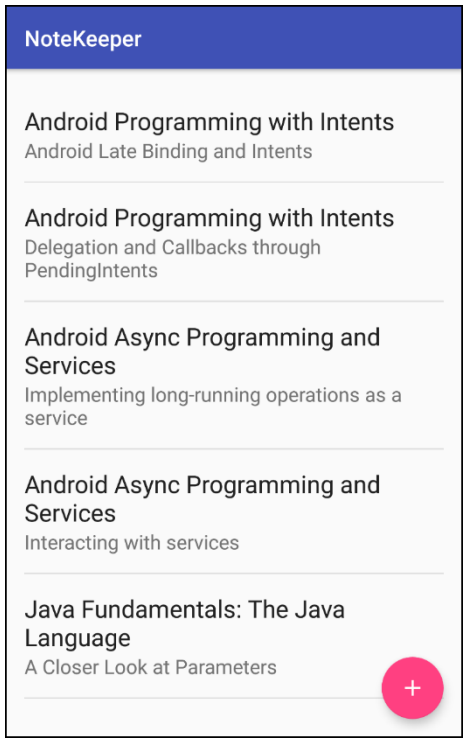
Polished Design Our App Starting Point

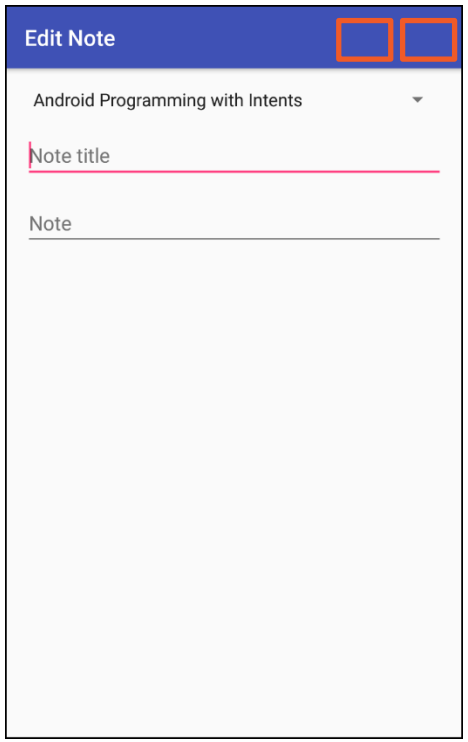
Designing Our App Starting Point




Course
Note title
Course
Note title
Implementing long-running operations as a service
Android Async Programming and Services
Interacting with services
Java Fundamentals: The Java Language
A Closer Look at Parameters


Course
Note title
Note body text


NoteKeeper
Android Programming with Intents
Android Late Binding and Intents
Android Programming with Intents
Delegation and Callbacks through PendingIntents
Android Async Programming and Services
Implementing long-running operations as a service
Android Async Programming and Services
Interacting with services
Java Fundamentals: The Java Language
A Closer Look at Parameters
+


Edit Note
Android Programming with Intents
Note title
Note

Summary

Android Studio

- Complete Android dev environment

Layout files

- Describe UI

Activity

- Code and functionality to present UX

Android Emulator

- Run and test apps from your desktop
- a.k.a. Android Virtual Device (AVD)

What to Expect from This Course

- Building a Simple App
- Designing and Planning Our App
- Understanding Activity & Layout Interaction
- Working with Activities and Activity Lifecycle
- Using Option Menus and Digging Deeper into Activity Lifecycle

Goals of This Course Series

Skills

Provide skills necessary to be an effective Android developer

Certification

Provide the knowledge necessary to pass the Android Associate Developer exam

What to Expect from This Course Series

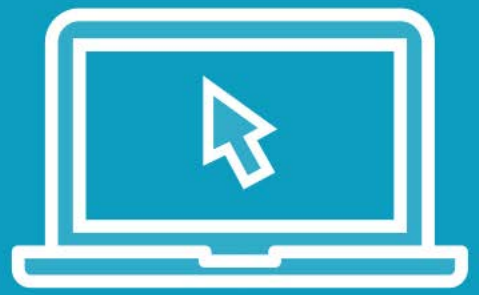
- Understanding Android App Basics
- Working with Android Tools & Testing
- Enhancing the App Experience
- Broadening App Appeal and Reach
- Managing App Data with SQLite
- Exposing Data & Info Outside Your App
- Leveraging the Power of the Platform

This Is the Module Title in Titlecase

Author Name
AUTHOR TITLE

@authortwitter www.authorsite.com

Demo



This bullet list is preset with animations

Use this layout to introduce your demo

How to do this one thing

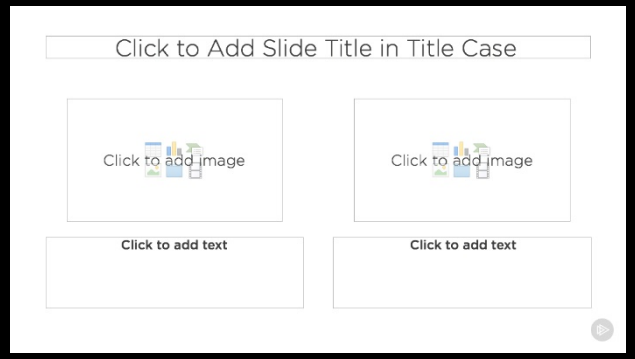
- Why we do it
- How we do it

Then there's that thing

Don't forget to do this

We'll finish it off with this thing

Using the **Image Chunking Slides**



Two Image Chunking

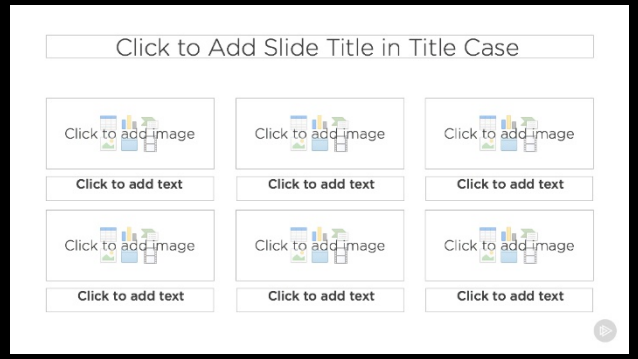
Three Image Chunking

These layouts can be used as an alternative to a bulleted list.

They're built specifically for **photos** or **graphics** and look especially awesome when you incorporate icons from the **Pluralsight Icon Library**.

See them in action in the next 4 slides.

Four Image Chunking



Six Image Chunking

Example of Image Chunking Two Items

Jill Anderson

Some information about this graphic goes here and four lines or fewer is best

John Doe

Some information about this graphic goes here and four lines or fewer is best

Example of Image Chunking Three Items

Clipboard
Some information goes here; three lines or fewer is best

Book
Some information goes here; three lines or fewer is best

Film
Some information goes here; three lines or fewer is best

Example of Image Chunking Four Items

Write

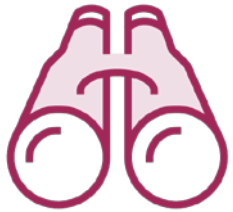
Create

Record

Learn

Example of Image Chunking Six Items

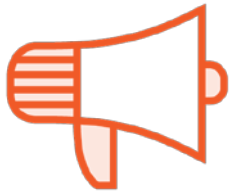
Address book



Binoculars

Camera

Eyeglasses

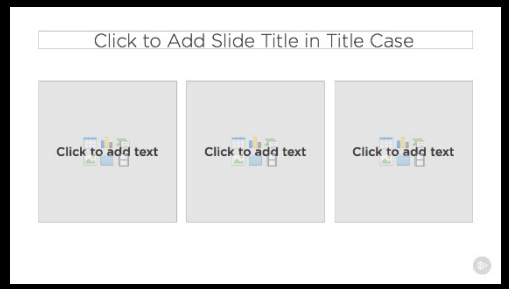


Megaphone

World

Using the **Text Chunking Slides**

Two Text Chunking



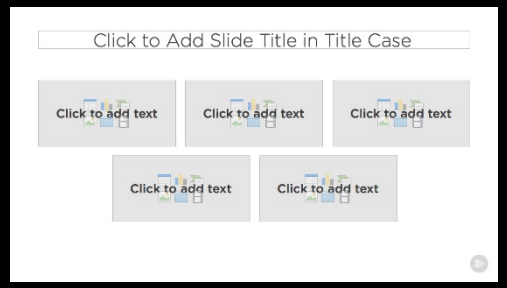
Three Text Chunking

These layouts are intended to group chunks of text. Among other uses, they can be a great alternative to a bullet list.

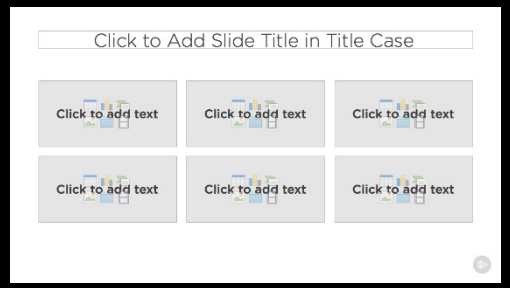
Use **animations** to bring focus to the point you're speaking on one at a time, and/or use **color** to group points together.

If you have more than six points to discuss, you may want to use a standard bullet list.

Four Text Chunking



Five Text Chunking



Six Text Chunking

We have provided some **example uses** of these layouts in the next few slides.

Text Chunking **Two Items**

Talking point one

Be concise and keep the text to four lines or fewer

Talking point two

Be concise and keep the text to four lines or fewer

Text Chunking Three Items

Talking point one

Be concise and keep the text to four lines or fewer

Talking point two

Be concise and keep the text to four lines or fewer

Talking point three

Be concise and keep the text to four lines or fewer

Text Chunking **Four Items**

This is the first talking point that should be kept to three lines or fewer

This is the second talking point that should be kept to three lines or fewer

This is the third talking point that should be kept to three lines or fewer

This is the fourth talking point that should be kept to three lines or fewer

Text Chunking Five Items

Talking point one

Keep the text to three lines or fewer

Talking point two

Keep the text to three lines or fewer

Talking point three

Keep the text to three lines or fewer

Talking point four

Keep the text to three lines or fewer

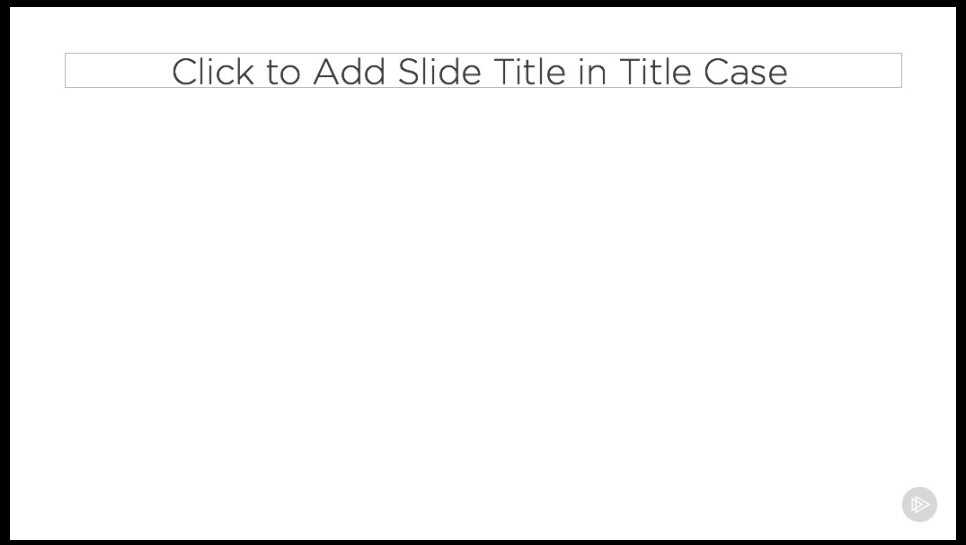
Talking point five

Keep the text to three lines or fewer

Today's Mobile World

iPhone	Nexus 5	Lumia 950 XL
iPad	Nexus 7	Surface

Using the **Title Only Slide**



Title Only

This is the slide you'll want to use when you just need a big space for a diagram, chart, or graphic.

Make sure you check out the training videos available on the **Author Kit** for design best practices.

If you need help bringing your ideas for this space to life, contact your Editor about getting help from one of our **Content Graphic Designers**. In most cases, you just need to submit a rough outline and let our designers work their magic. However, in some special cases, your Editor can get you in touch with a designer directly.

We included some possible starting points for you in the next few slides.

Remember, we are here to help!

Customer Acquisition and Loyalty

Observed higher sales

42%

Observed more loyal customers

70%

Timeline of Events

1940s
McCulloch and Pitts
Neural network theory

1956
Stephen Cole Kleene
Regular events/sets

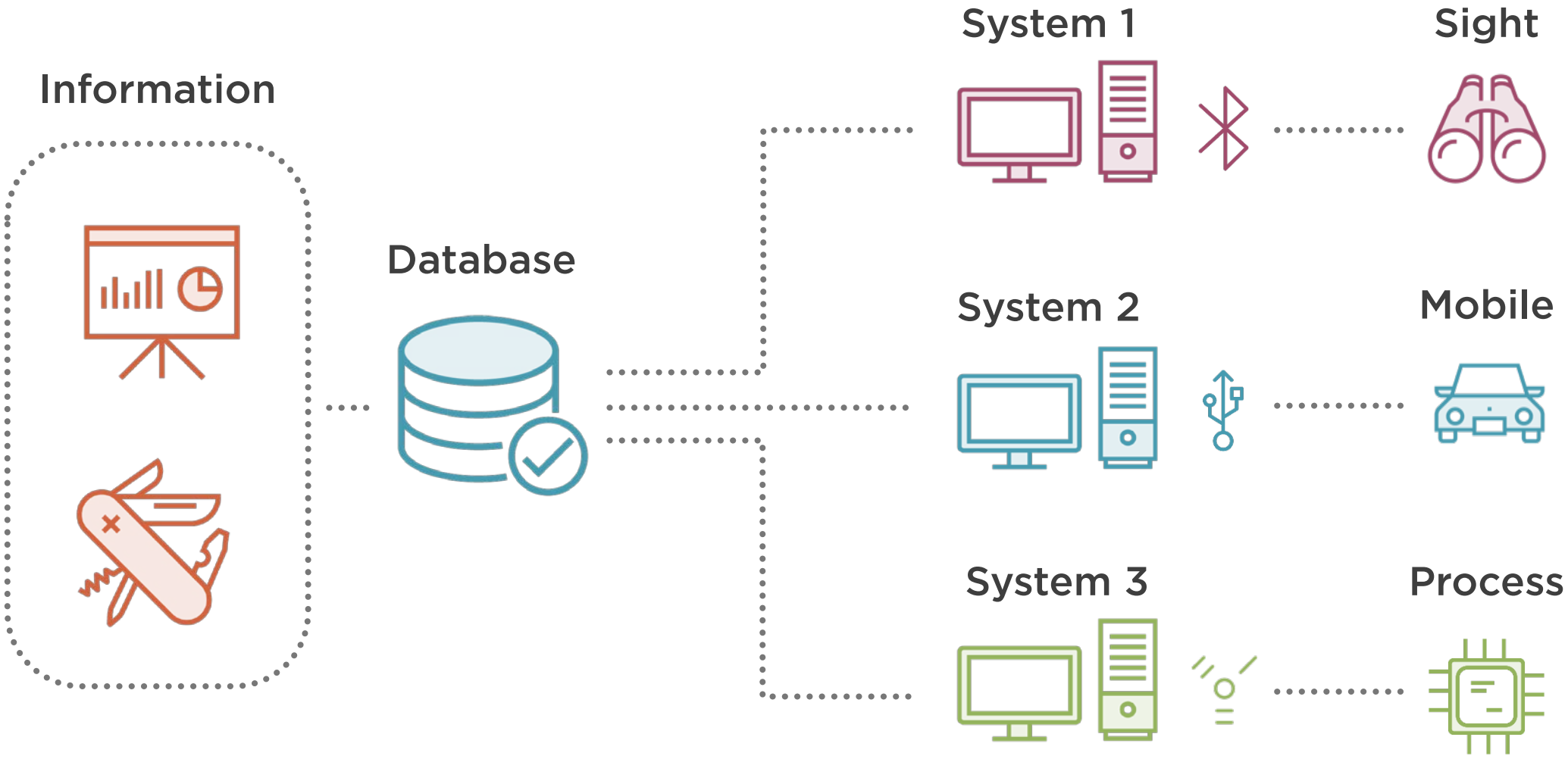
1968
Ken Thompson
First in computing

1973
Ken Thompson
First release of *grep*

1986
Henry Spencer
regex library

1987
Larry Wall
Integration into Perl

Title Only Layout Example



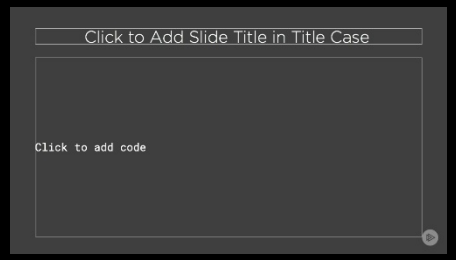
Using the **Code Slides**

Code Top (Dark)

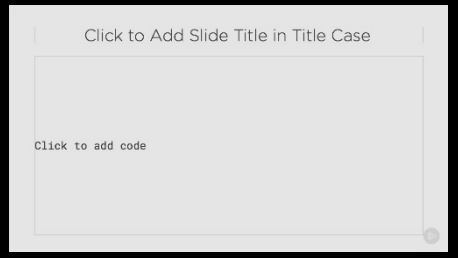
Code Top (Light)

Code Top Layouts

Use when you need a slide title and info about your code



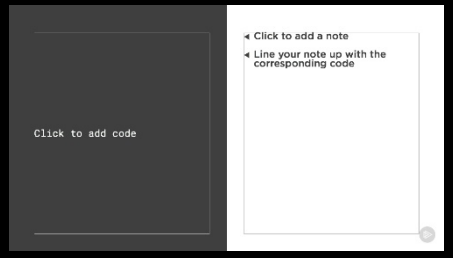
Code (Dark)



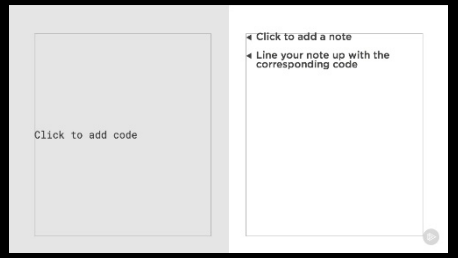
Code (Light)

Code Layouts

Best for larger code snippets



Code Notes (Dark)



Code Notes (Light)

Code Left Layouts

Great for annotating code structure

Make use of the color palette to highlight code.

We recommend using the **Roboto Mono** typeface for your code slides. However, if you use a different font for code in your demos, feel free to use that instead to reinforce a consistent look.

```
<div class="row carousel-indicators">

            <div style="background-color:red;" class="col-
md-4" data-target="#homeCarousel" data-slide-to="0"
class="active”>

<div class="row carousel-indicators”>
```

Slide Title in Titlecase

Information about the code above

```
<div class="row carousel-indicators">

            <div style="background-color:red;" class="col-
md-4" data-target="#homeCarousel" data-slide-to="0"
class="active”>

<div class="row carousel-indicators”>
```

Slide Title in Titlecase

Information about the code above

Code Snippet on Dark

```
<div class="row carousel-indicators">

            <div style="background-color:red;" class="col-
md-4" data-target="#homeCarousel" data-slide-to="0"
class="active">

            </div>

            <div style="background-color:green;"
class="col-md-4" data-target="#homeCarousel" data-slide-
to="1">

            </div>
```

Code Snippet on Light

```
<div class="row carousel-indicators">

            <div style="background-color:red;" class="col-
md-4" data-target="#homeCarousel" data-slide-to="0"
class="active">

            </div>

            <div style="background-color:green;"
class="col-md-4" data-target="#homeCarousel" data-slide-
to="1">

            </div>
```

```
Put code on this side

var proto = {
foo:’Hello World’
};


function Bar(){}
Bar.prototype = proto;


var baz = new Bar();


console/log(baz.foo);
```

- Line up with these notes
- Set up prototype object
- Constructor function and set prototype property
- Create instance
- Call inherited member

```
Put code on this side

var proto = {
foo:’Hello World’
};


function Bar(){}
Bar.prototype = proto;


var baz = new Bar();


console/log(baz.foo);
```

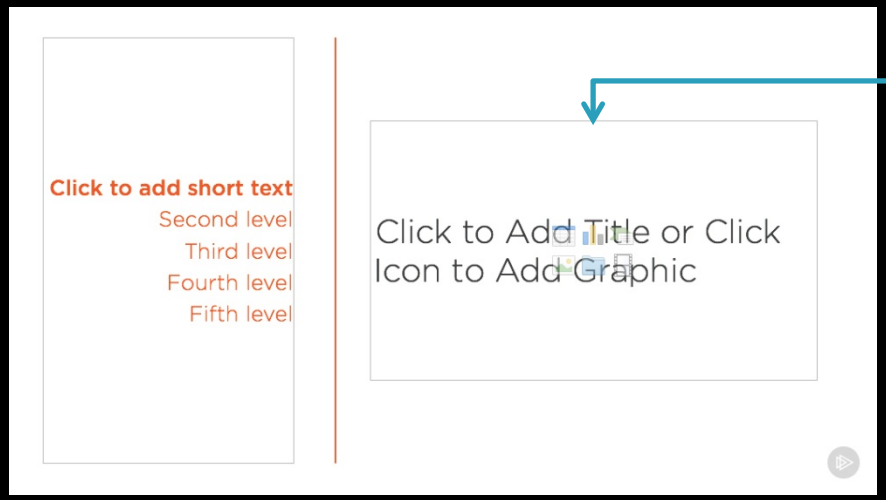
- Line up with these notes
- Set up prototype object
- Constructor function and set prototype property
- Create instance
- Call inherited member

Using Bullet List Slides

We've provided some bullet list layouts to accommodate various quantities of information.

Content left | Title/Image right

Intended for bullet text that is shorter and titles/images that are larger

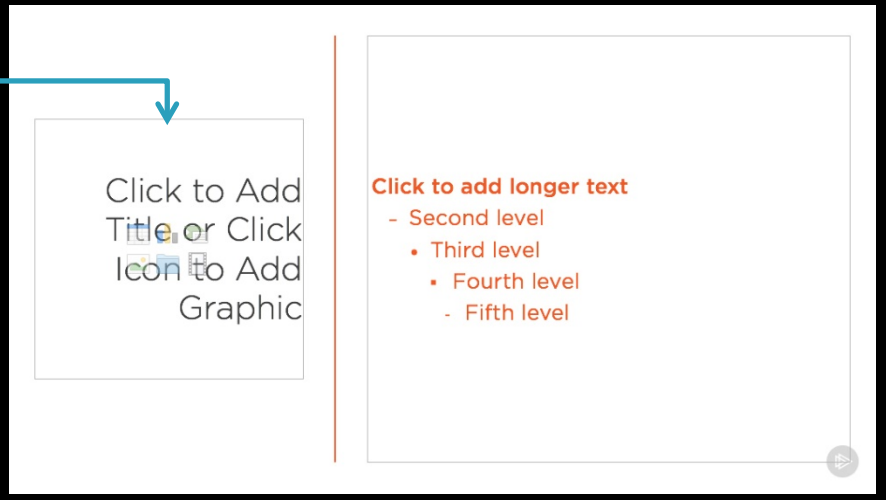


Content | Image/Title

Remember, you can use **text** or **images** in these placeholders.

Title/Image left | Content right

Intended for bullet text that is longer and titles/images that are smaller



Image/Title | Content

Animation built in

Bullet alternative

Sentence fragments

List of things

Procedure list

Talking points

Title or Relevant Graphic

Animation built in

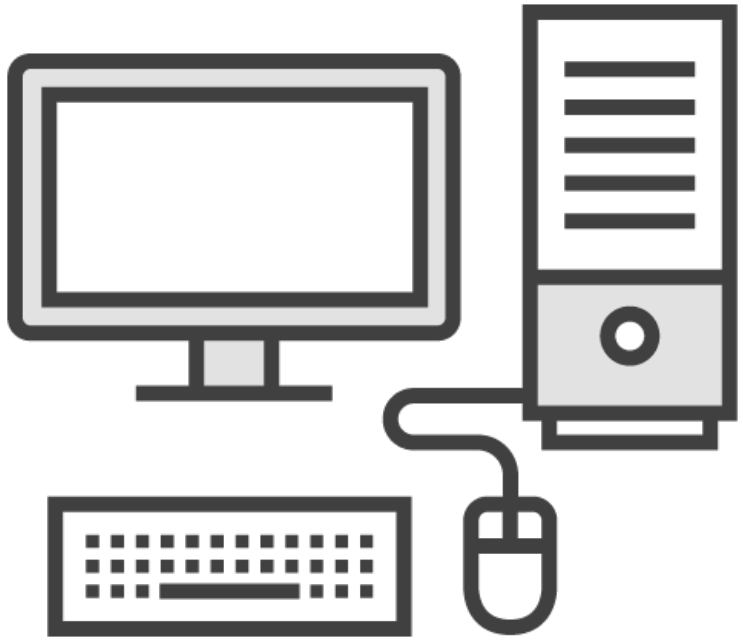
Bullet alternative

Sentence fragments

List of things

Procedure list

Talking points



Title or Relevant Graphic

Animation built in

Bullet alternative

Room for a bit more text

Use this layout for

- Longer sentence fragments
- List of things
- Procedure list
- Talking points

Animation built in

Bullet alternative

Room for a bit more text

Use this layout for

- Longer sentence fragments
- List of things
- Procedure list
- Talking points

Title Space with Image

Animation built in

Bullet alternative

Room for a bit more text

Use this layout for

- Longer sentence fragments
- List of things
- Procedure list
- Talking points

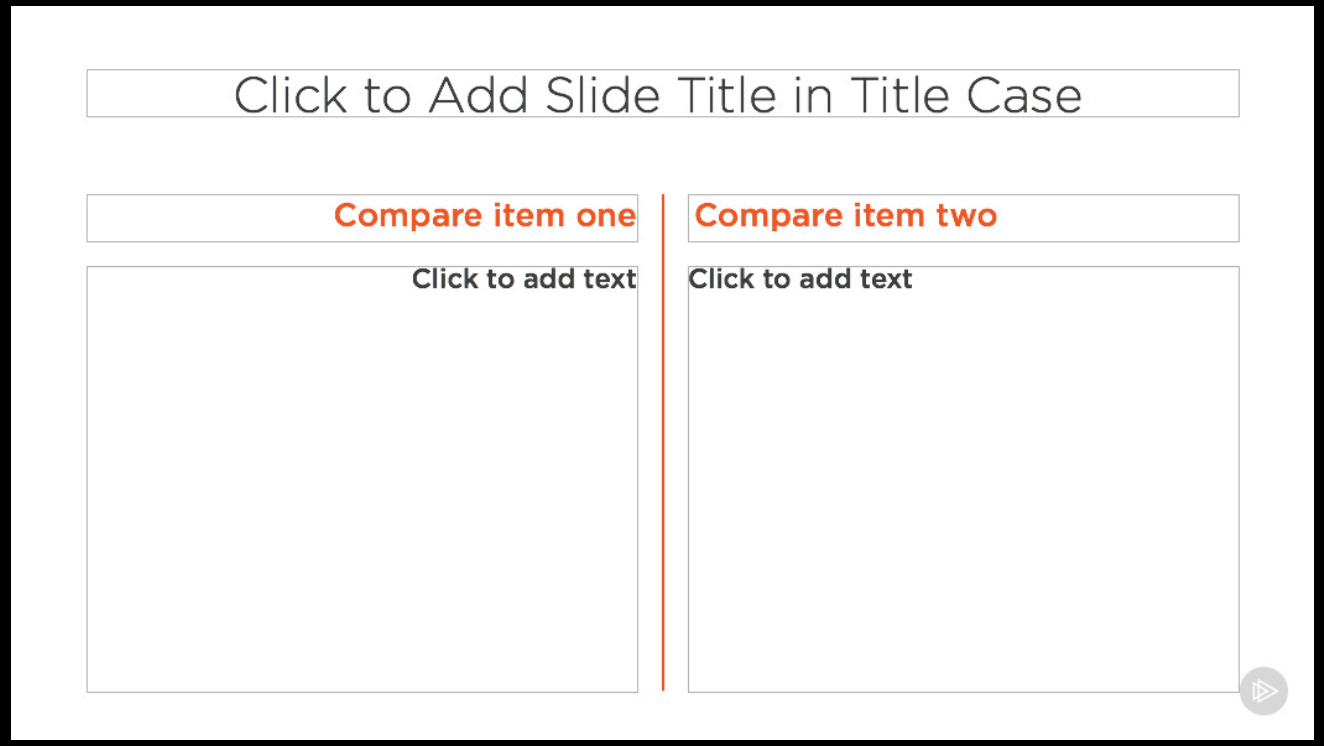
Graphic on left should fill the entire space

- Graphic must be high quality and royalty free

Graphic and text animation is built in

Comparison Slide

Use this slide if you need to compare single items or groups of items.



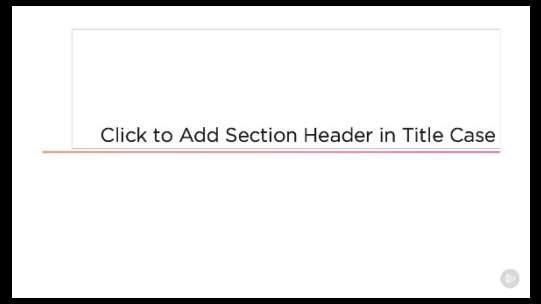
Comparison Example

Functional group	Objectives
Configure and administer security	Manage vSphere storage virtualization
Configure advanced networking	Configure software-defined storage
Configure advanced storage	Configure vSphere storage multipathing and failover
Administer and manage resources	Perform advanced VMFS and NFS configurations and upgrades
Configure availability solution	
Deploy and consolidate vSphere	

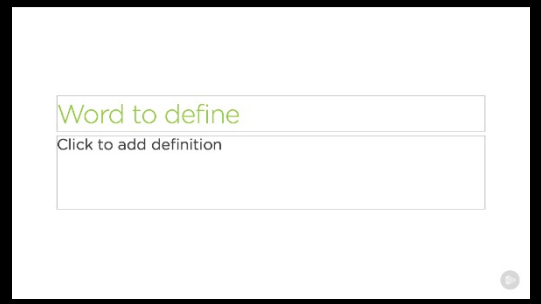
Other Slides

The following self-explanatory slides are a good way of adding diversity into the flow of your course.

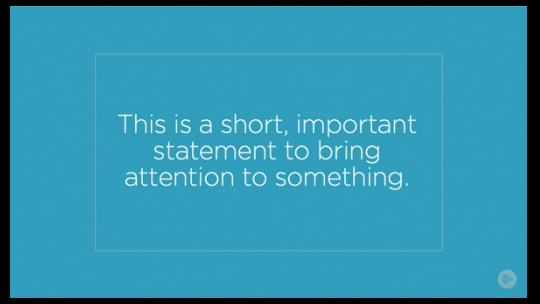
Use them purposefully.



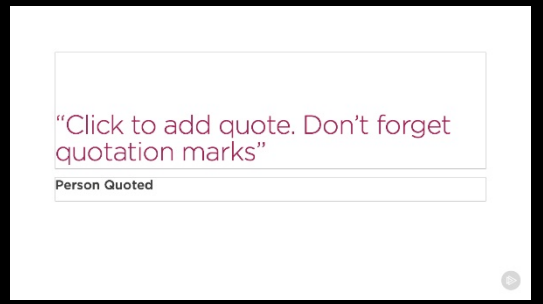
Section Header



Definition



Important Statement



Quotation

Section Heading

Word Definition

Here is where you put the definition. This is one of the few places where complete sentences are appropriate. Be sure to cite your source.

This is a short, important statement to bring attention to something.

“Using quotes in your slides can be powerful if used sparingly.”

Heather Ackmann