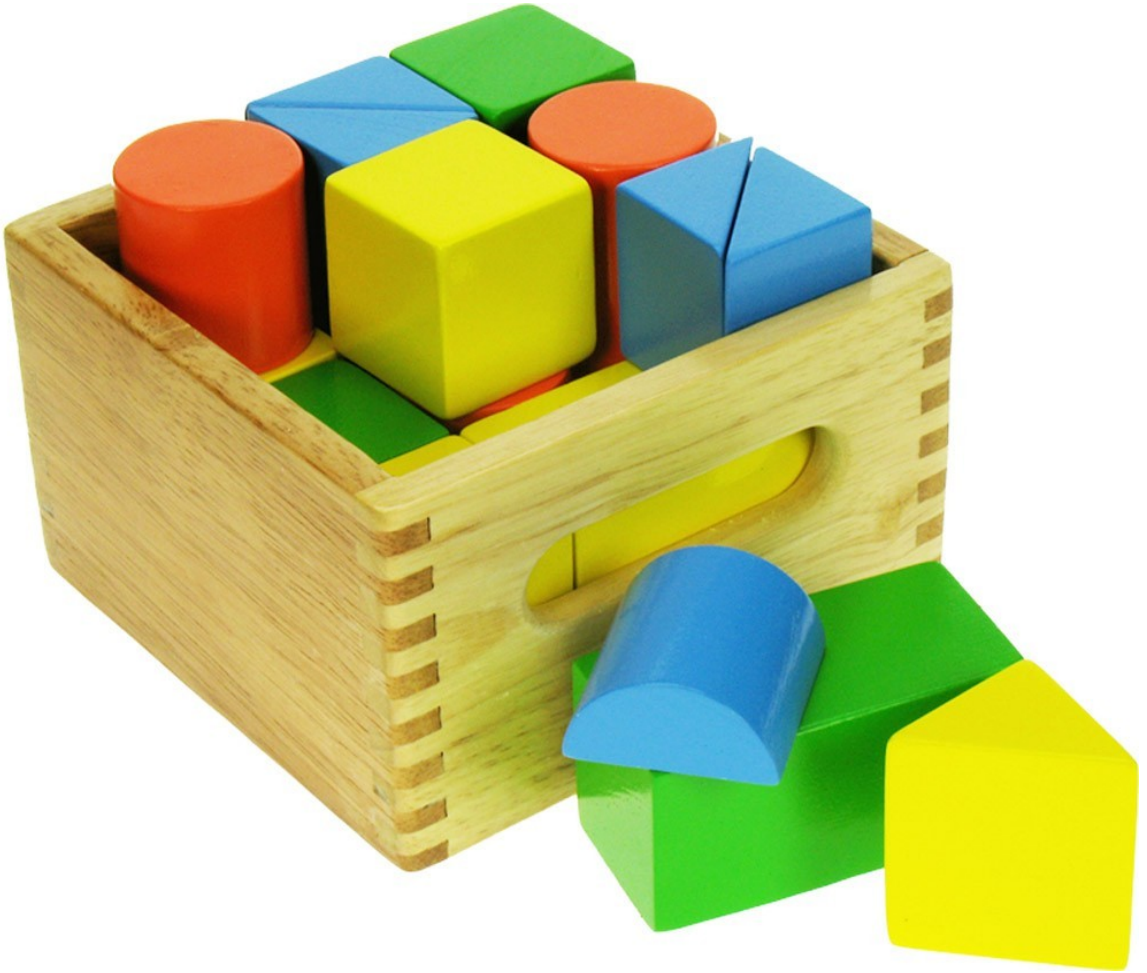
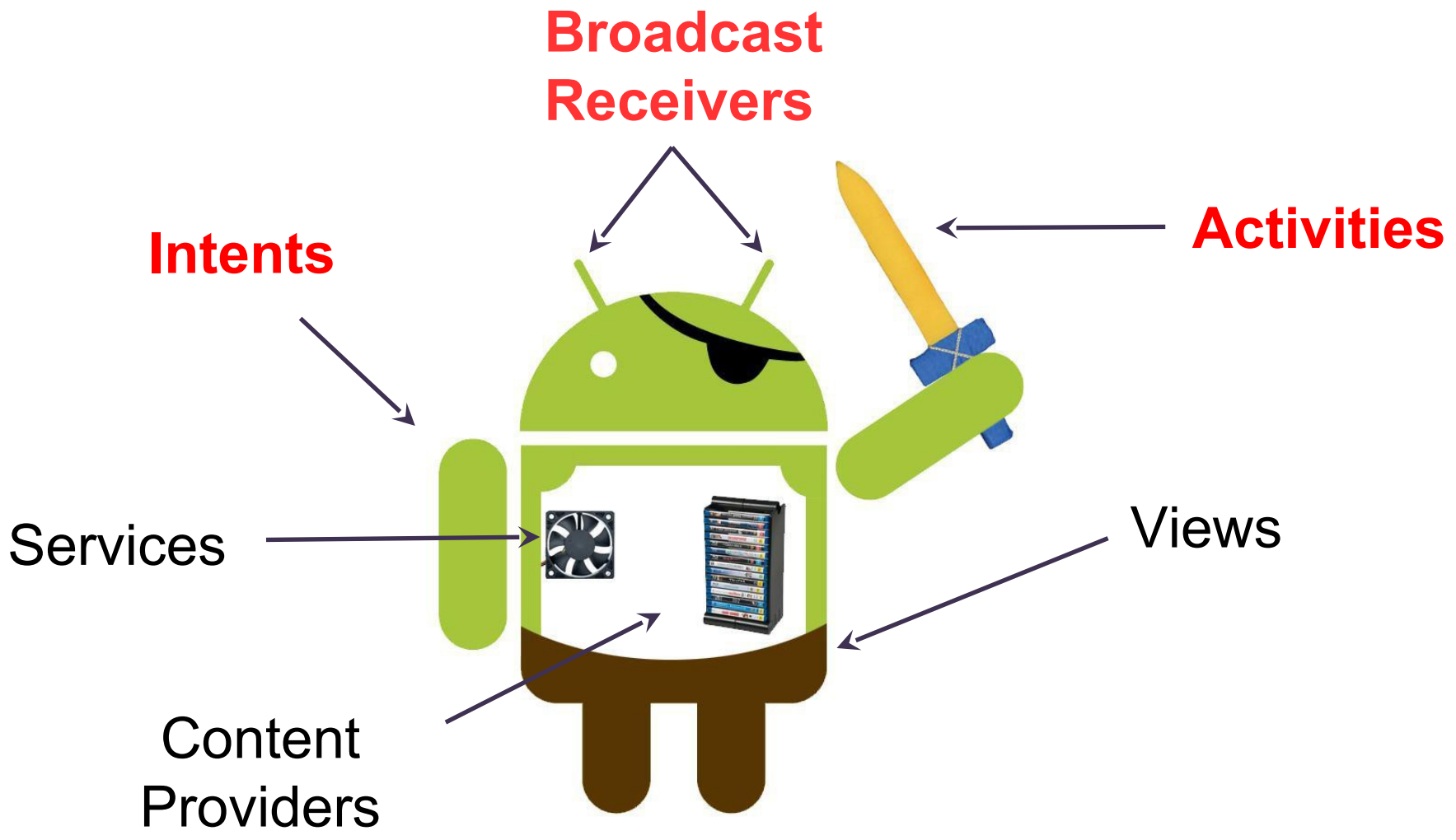
Android Application Building Blocks



Intents and Intent Filters

Anatomy of Android application

Anatomy of Android application



Intents

- Message-passing mechanism on Android
- Facility for late run-time binding between components
  - activities, services, broadcast receivers
  - same or different applications
- Passive data structure describing
  - operation to be performed
  - something that has happened

Intents In Action

(A) Activity starts another activity by sending an Intent (internal)

```
public void callActivityTwo(View view) {
    final Intent intent =
            new Intent(this, NewActivity.class);

    startActivity(intent);
}
```

(B) Activity starts another activity by creating an intent that says “View site javaguru.lv” (external)

```
public void openSite(View view) {
    final Intent intent =
            new Intent(Intent.ACTION_VIEW,
                    Uri.parse("http://javaguru.lv"));

    startActivity(intent);
}
```

Intents

Action propagation via Intents is Android design principle

- modular architecture
- event-driven applications
- decoupling of components
- seamless replacement of application elements

Intent Object

- Component name
  - Component name that should handle the intent
  - setClassName("package", "package.Activity")
    setClassName(this, "package.Activity")
- Action
  - string naming the action to be performed
  - determines how data and extras fields are structured
  - ex., ACTION_VIEW, ACTION_CALL, ACTION_EDIT
- Data
  - URI and the MIME type of the data

Intent Object

- Common Action/Data Pairs
  - ACTION_VIEW http://www.javaguru.lv
    - view site in browser
  - ACTION_DIAL content://contacts/people/1
    - dial person
  - ACTION_VIEW tel:123
    - show dialer with the number
  - ACTION_DIAL tel:123
    - dial the number
  - ACTION_VIEW content://contacts/people/
    - list of contacts

Intent Object

- Category
  - Additional information about the target component
  - LAUNCHER - initial activity of a task

    HOME - activity displays the home screen

    PREFERENCE - activity is a preference panel
- Extras
  - Additional information delivered to the component handling the intent (key-value pairs) (1 Mb)
- Flags

Intent Resolution

- Determine component to handle passed intent
- Types of Intents
  - Explicit intents - designate the target component by i class name

    new Intent(this, ActivityX.class)

  - Implicit intents - do not name a target; component to handle the intent is found via intent filters

    new Intent(ACTION_VIEW,Uri.parse("http://lv"))

Intent Filters

- Set up in application's manifest
- Describe capabilities of target components
- Inform which implicit intent a component can handle
- Intent object are tested against intent filters
  - Action, Category, Data
- User may be asked to choose is intent can pass through the filters of multiple activities

Intent Filters in Action

- Activities that can initiate applications have filters with "android.intent.action.MAIN" action
- Activities to be represented in the application launcher specify "android.intent.category.LAUNCHER" category

```
<activity
    android:name="com.example.intent1.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Intent Filters in Action

- Filter fields parallel action, data, category fields of Intent

- **Action test**
  - Intent action must match one of the filter actions

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT" />
    <action android:name="com.example.project.SHOW_RECENT" />
    <action android:name="com.example.project.SHOW_PENDING" />
    . . .
</intent-filter>
```

Intent Filters in Action

- **Category Test**
  - Every Intent category must match a category in the filter
    - NB! intents passed to startActivity() are assigned "android.intent.category.DEFAULT"

```
<intent-filter . . . >
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    . . .
</intent-filter>
```

Intent Filters in Action

- **Data Test**
  - specify URI and data type
  - separate attributes for URI parts (scheme, host, port, and path)
  - Intent URI is compared to the parts of the URI in filter
  - ex. <data android:mimeType="image/*" />

    <data android:scheme="http" android:type="video/*" />

```
<intent-filter . . . >
    <data android:mimeType="video/mpeg" android:scheme="http" . . . />
    <data android:mimeType="audio/mpeg" android:scheme="http" . . . />
    . . .
</intent-filter>
```

Intent Kung Fu

- Returning result from Activities
  - Start Activity as a sub-Activity that can pass results back to its parent

```
public void openSite(View view) {
    final Intent intent = new Intent(this, SubActivity.class);
    startActivityForResult(intent, SUBACT_REQUEST_CD);
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {

    if (requestCode == SUBACT_REQUEST_CD) {
        if (resultCode == RESULT_OK) {
            String response = data.getExtras()
                    .getString(SUBACT_REQUEST_PARAM1);
            Toast.makeText(this, response, Toast.LENGTH_LONG).show();
        }
    }
```

Intent Kung Fu

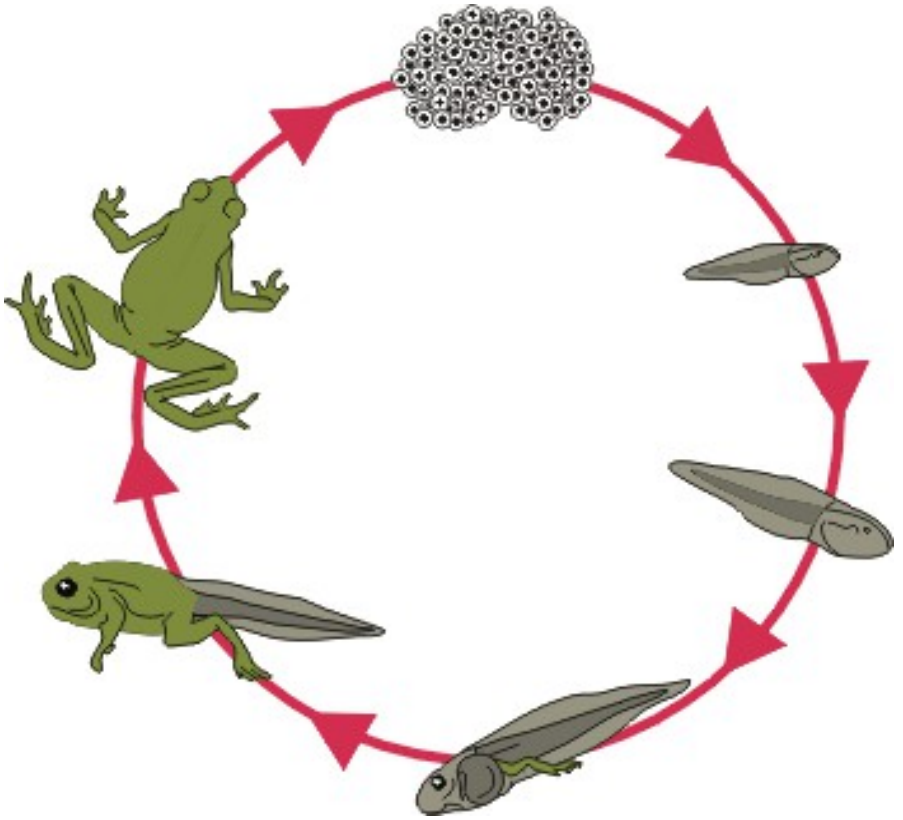
- Returning result from Activities

```
public void ok(View view) {
    EditText editText = (EditText) findViewById(R.id.edittext1);

    Intent result = new Intent(Intent.ACTION_PICK);
    result.putExtra(SUBACT_REQUEST_PARAM1,
            editText.getEditableText().toString());

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

Activity Life Cycle



Android Application

- One or more loosely bound Activities
  - Typically, one is marked “main”
  - Each activity can start another via Intents
- Has its own process with separate instance of Dalvik by default
- Limited control over own lifecycle. Instead, application components must listen for changes in the application state and react accordingly

Activity Stacks

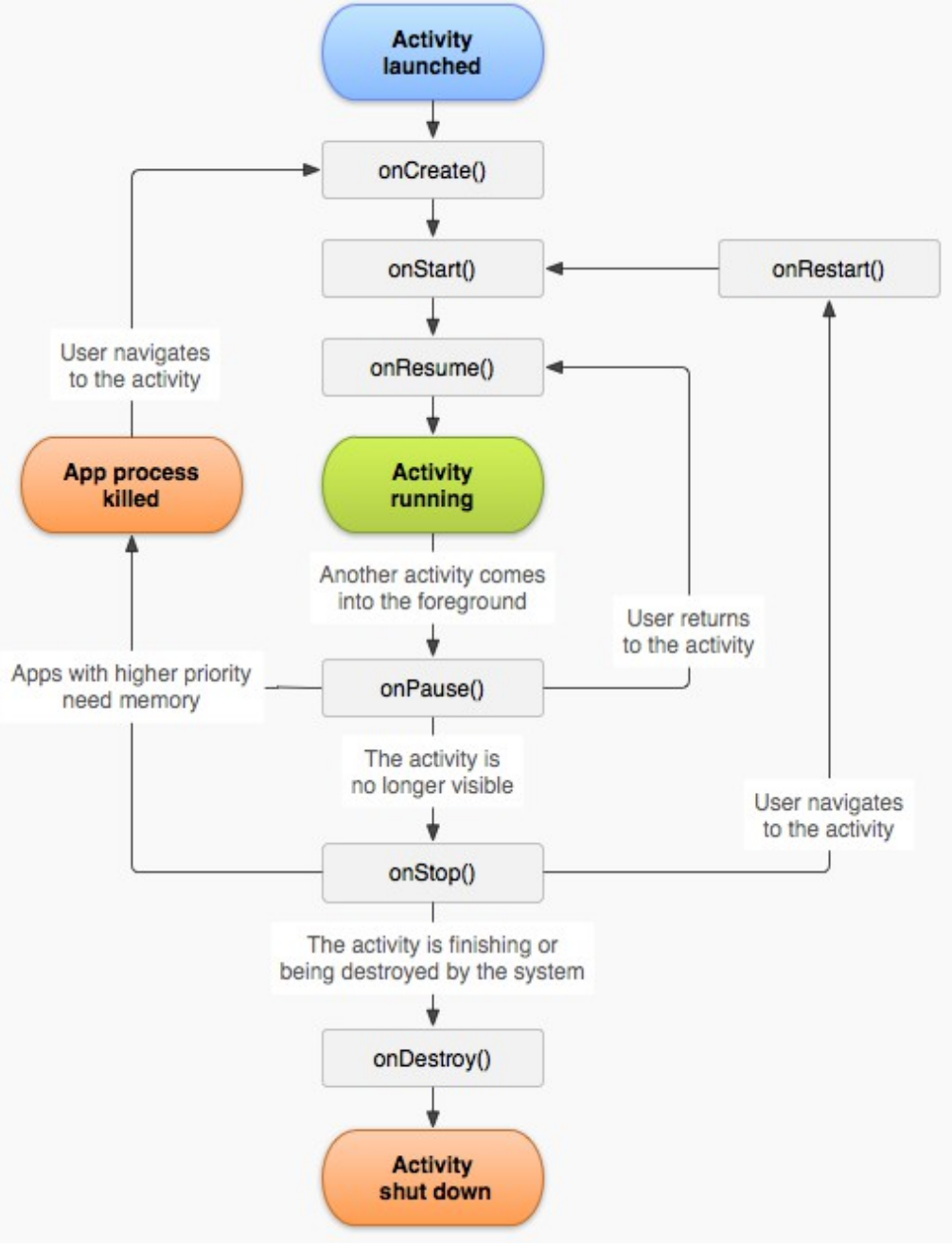
- Activities are managed as activity stack ("back stack").
  - user is switching activities
  - new activity is placed on the top and becomes the running one
  - previous activity remains below it in the stack, and will not come to the foreground again until the new activity exits

Multitasking

- Task - collection of activities user interacts with performing a certain job
- Home screen – task starting place
  - Application's task comes to the foreground when user touches an icon in the application launcher
  - New task is created and "main" application activity opens as the root activity if no task exists

Activity States

- Activities transition four possible states
  - Active — activity is at the top of the stack, visible, focused
  - Paused — visible Activity without focus
  - Stopped — when Activity isn't visible, it "stops."
  - Inactive — after Activity is killed, and before it's been launched

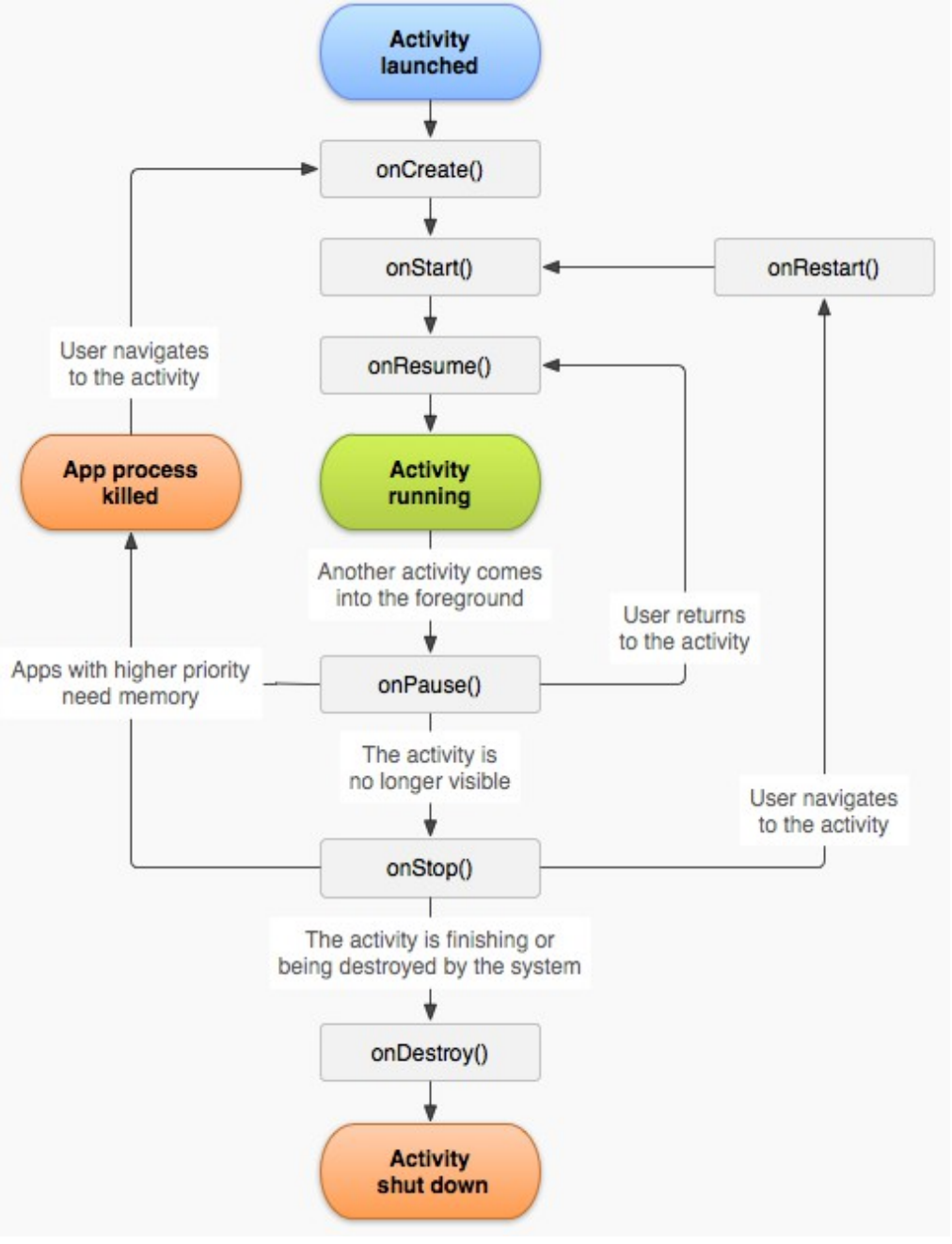


Activity States

- Activity has callback methods to perform operations when the Activity moves between states
  - onCreate() - initialize the essential components. Define the layout for the activity's user interface via setContentView()
  - onPause() - commit any changes that should be persisted beyond the current user session

Activity Lifetimes

- Full Lifetime
  - between the first call to onCreate and the final call to onDestroy
- Visible Lifetime
  - bound between calls to onStart and onStop
- Active Lifetime
  - starts with onResume and ends with onPause



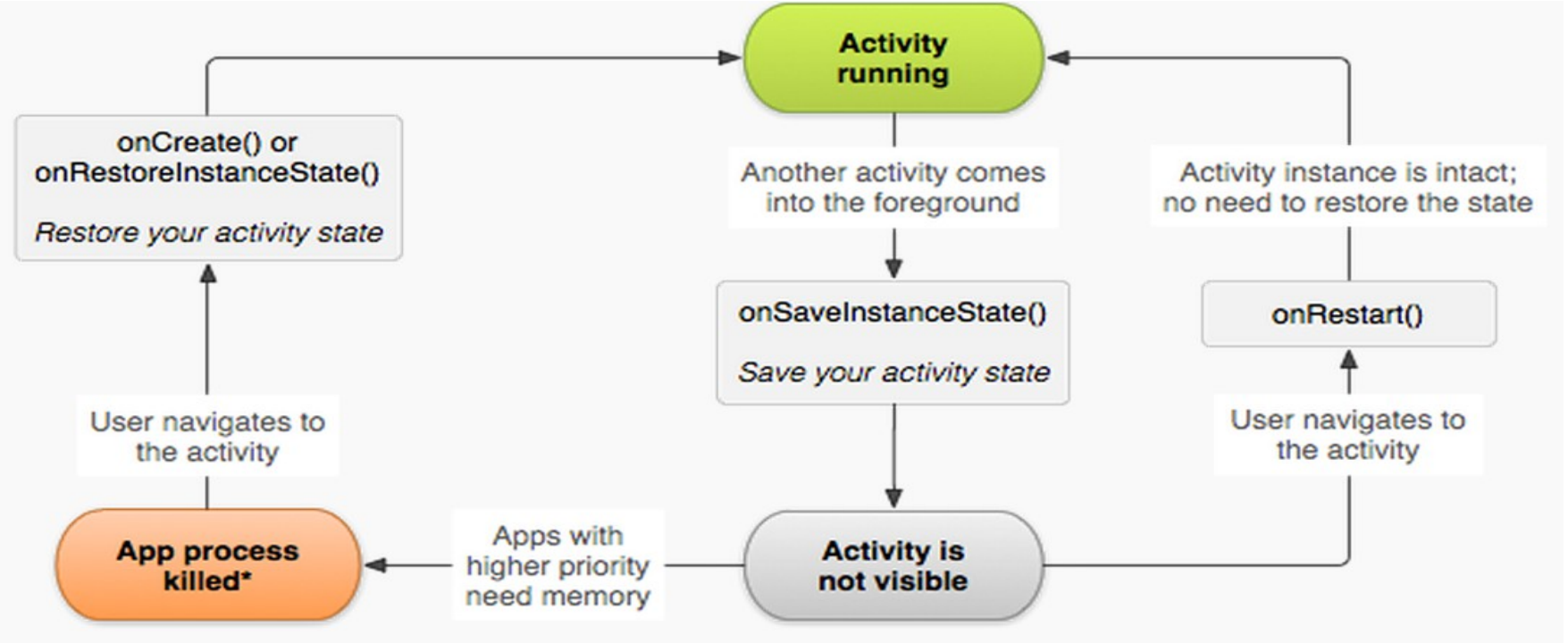
Saving Activity State

- Problem
  - When an activity is paused or stopped, its state is retained while Activity object is held in memory
  - After an activity is destroyed the system cannot simply resume activity with its state intact

- Solution
  - The system recreates destroyed Activity object if the user navigates back to it

Saving Activity State

- Activity state can be preserved with onSaveInstanceState() callback
- To restore use onCreate() and onRestoreInstanceState()



Saving activity state

- Save state information about the activity as name-value pairs in a Bundle
- onSaveInstanceState() is not guaranteed to be called, never use it to store persistent data (use onPause())
- onSaveInstanceState() method is called for every View in the layout by default


```
protected void onSaveInstanceState(Bundle outState) {
    outState.putCharSequenceArrayList("todoList", todoList);
};
```

Activity Practical

- **Improve purchase list application UX**
  - Continue working on Purchase list application
  - Save and restore state information about purchase list entries

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    ArrayList<CharSequence> savedList = null;
    if (savedInstanceState != null) {
        savedList =  savedInstanceState.getCharSequenceArrayList("todoList");
    }

    todoList = savedList != null ? savedList : new ArrayList<CharSequence>();
    final ArrayAdapter<CharSequence> todoListAdapter =
            new ArrayAdapter<CharSequence>(this,
                    android.R.layout.simple_list_item_1, todoList);
```

Application Manifest

Application Manifest

Essential information about the application

- Names Java package for the application
- Specifies application metadata (icon, version number, theme)
- Describes components of the application
- Declares application permissions
- Declares Android API versions
- Lists the libraries that the application must be linked against

<action>

<activity>

<activity-alias>

<application>

<category>

<data>

...

<intent-filter>

<uses-feature>

<uses-library>

<uses-permission>

<uses-sdk>

Permissions

- A permission limits access to a part of the code or to data on the device
- Application must declare that it requires that permission with a <uses-permission>
- An application can also protect its own components (activities, services, broadcast receivers, and content providers)

```
<uses-permission android:name="android.permission.LOCATION"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

Features

- Use uses-feature nodes to specify which features your application requires
- Prevents your application from being installed on a device w/o required piece of hardware

  <uses-feature android:name=”android.hardware.nfc” />