95-702 Distributed Systems

A Short Introduction to Android

Notes taken from Google's Android SDK
and Google's Android Application Fundamentals

Plan For Today

- Lecture on Core Android
- Three U-Tube Videos:
  - Architecture Overview

    http://www.youtube.com/watch?v=Mm6Ju0xhUW8
  - Application Lifecycle

    http://www.youtube.com/watch?v=fL6gSd4ugSI
  - Application Programmer Interfaces

    http://www.youtube.com/watch?v=MPukbH6D-lY

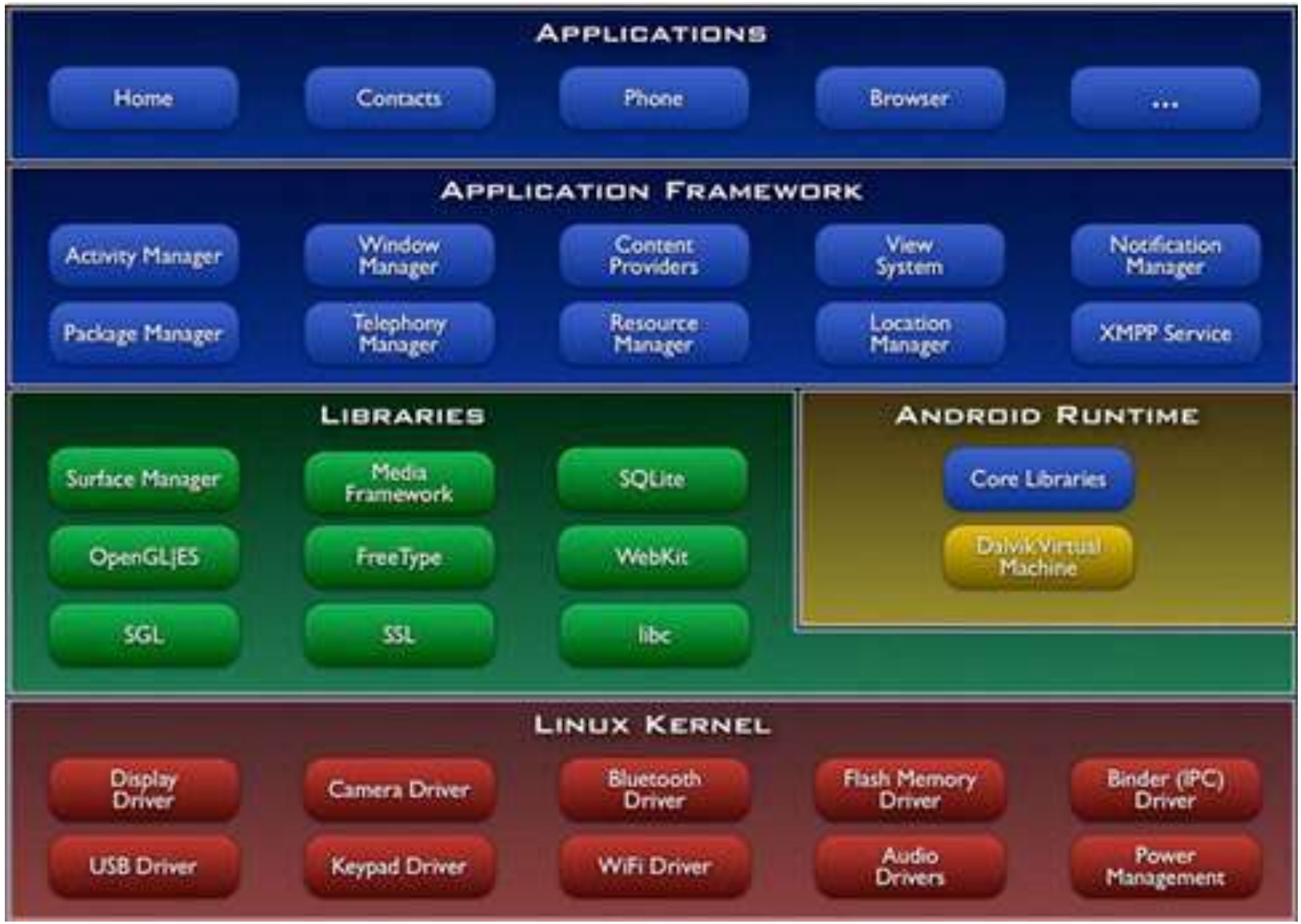
Why Android?

- Mobile platforms represent important components of distributed systems.
- Android is a new and interesting mobile platform.
- Android may also become important on non-mobile platforms.
- We will look at Android from a developers point of view.

What is Android?

Applications	Contacts	Phone	Browser	Home…
Application Framework	Window Manager	Content Providers	Location Manager	Activity Manager …
Libraries	SQLite	SSL	WebKit	Android Runtime VM…
Linux Kernel	WiFi Driver	Binder (IPC) driver	Camera Driver	…

System Architecture Diagram from Google



Linux Provides

- Hardware abstraction – drivers
- Memory management
- Process Management
- Security Model
- Networking

Native Libraries

- Sit on top of Linux
- Written in C/C++
- Drawing, Graphics (3D and 2D)
- Media framework (codecs like MP4 to code and decode MP4 bit streams)
- Font rendering
- Database (SQLLite)
- WebKit
- Android runtime (VM) – mutiple instances

Application Framework in Java (1)

- Activity Manager
- Package Manager
- Windows Manager
- Telephony Manager
- Content Providers
- Resource Manager
- View System
- Location Manager
- Notification Manager
- XMPP Service

This is a toolkit for applications.
Applications are such things as the phone application and the contacts application.

Application Framework in Java (2)

- Activity Manager

  Manages the lifecycle of applications. Applications cycle through various states: Started, Running, Background, Killed.

  Maintains a common stack allowing the user to navigate from one application to another.

Application Framework in Java (3)

- Package Manager

  Maintains information on the available applications on the device.

Application Framework in Java (4)

- Windows Manager

  Performs window management.

Application Framework in Java (5)

- Telephony Manager

  API’s needed to build the phone application and SMS.

Application Framework in Java (6)

- Content Provider

  Allows one application to make its data available to another. For example, the contacts application makes its data available to other applications that may need it.

  The phone or email application may need to consult contacts.

Application Framework in Java (7)

- Resource Manager

  Manages the storing of strings and layout files and bitmaps.
  "Android will run on many devices in many regions. To reach the most users, your application should handle text, audio files, numbers, currency, and graphics in ways appropriate to the locales where your application will be used". (From the SDK)

Application Framework in Java (8)

- View System

  Contains the building blocks of the user interface – buttons, text boxes and so on. Handles event dispatching, layouts and drawing.
  The View class is the base
  class for the Widget class. All views in a window are organized in a single tree.

Application Framework in Java (9)

- Location Manager

  - Uses GPS – most accurate, slow, battery drain, and primarily for outdoor use.
  - Uses the Android Network provider which uses cell tower and wi-fi signals.

Application Framework in Java (10)

- Notification Manager

  Alerts the user about events.
  Status bar updates, flashing lights, vibrations and the like.

Application Framework in Java (11)

- XMPP Service

  IETF Standard: The Extensible Messaging and Presence Protocol (“Presence = ‘who’s online?)

  XML Based Internet Instant Messaging

  Adopted by GoogleTalk (Voice over IP and Instant Messaging)

  Provides for offline messaging.

  Unsure if still supported by Android.

Example(1): Using The Telephony Manager

- import android.telephony.TelephonyManager;
- Your application requests READ_PHONE_STATE permissions.
- Ask the system for a pointer to the TelephonyManager.
- Register a listener for state changes.
- Make calls on the TelephonyManager class.
- The android.telephony.gsm package allows your application to send and receive SMS or MMS messages.

Example (2): Using The Location Manager

- Ask system for a pointer to the LocationManager.
- Permissions requested in the manifest.
- Implement a location listener.
- Receive a GeoCoder object.
- A GeoCoder provides geocoding and reverse geocoding. Geocoding is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate. Reverse geocoding is the process of transforming a (latitude, longitude) coordinate into a (partial) address.

Example (3): Maps in Two Ways

(1) Create an Intent with an Action_View and a URI holding longitude and latitude.
Call startActivity with the Intent.

(2) For greater control, add a MapView widget to your application.

Application Fundamentals

- An Android package (.apk file) holds an application.
- No single entry point. The system can instantiate and run components as needed.
- There are four types of components found within an application….

Android's Component Model

- An application will be built from:

- Activity Components
- Service Components
- Intent Receiver Components
- Content Provider Components

Activity Components

A concrete class that may be subclassed.
Often represents a single full screen window.
One focused endeavor.

Has a well-defined life-cycle:

onCreate()
onStart()
onResume()
onFreeze()
onStop()
onDestroy()

Activity Components

Each activity usually represents a single screen.

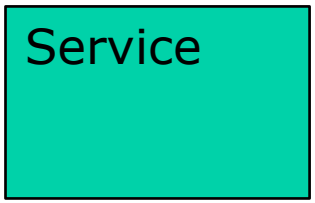
One activity may start another within the same application.

The user's interaction takes place through views.

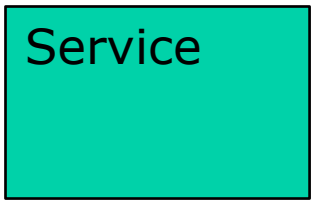
An application would usually consist of several activities.

Views consist of buttons, text fields, scroll bars, etc., and are organized in a hierarchy.

Service Components



A service has no visual user interface. It runs in the background for some indefinite period of time. A service may expose an interface.



Using a service component, we can expose functionality to other applications. Services may be started by an Activity and may continue after the Activity leaves the screen.

Intent or Broadcast Receiver Components

Activity Component

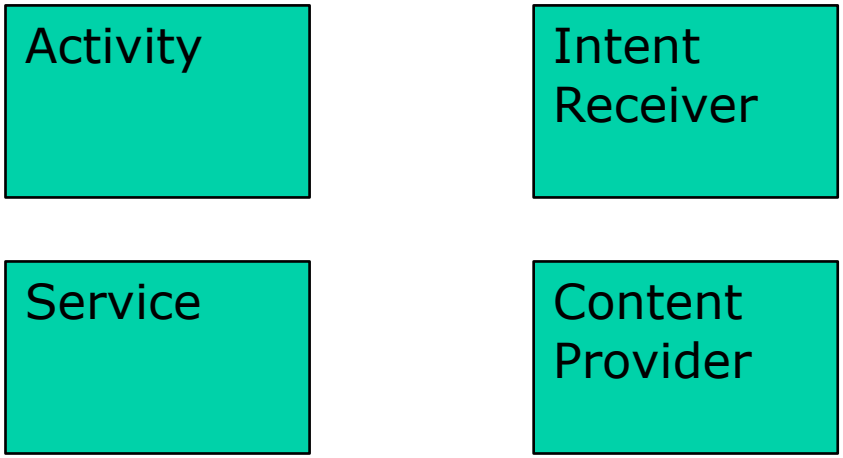
Intent or Broadcast Receiver Component

Service Component

May be used to start an activity when a message arrives.

The Intent receiver component does nothing but react to announcements. Many announcements originate in system code — for example, announcements that the time zone has changed or that the battery is low. Applications can also initiate announcements — to let other applications know of some change in state. (From http://developer.android.com/Reference/android/content/ContentProvider.html)

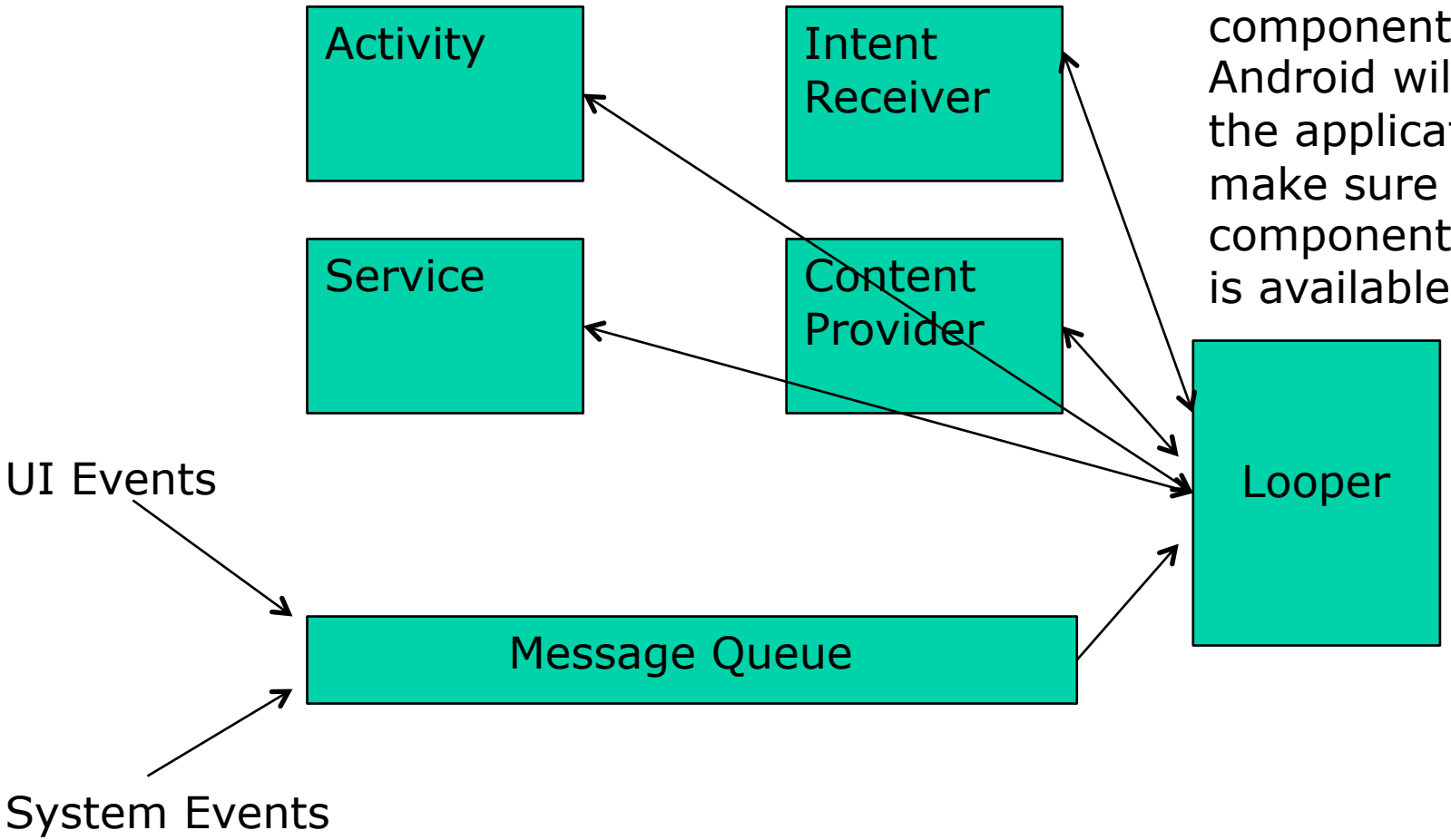
Content Provider Component



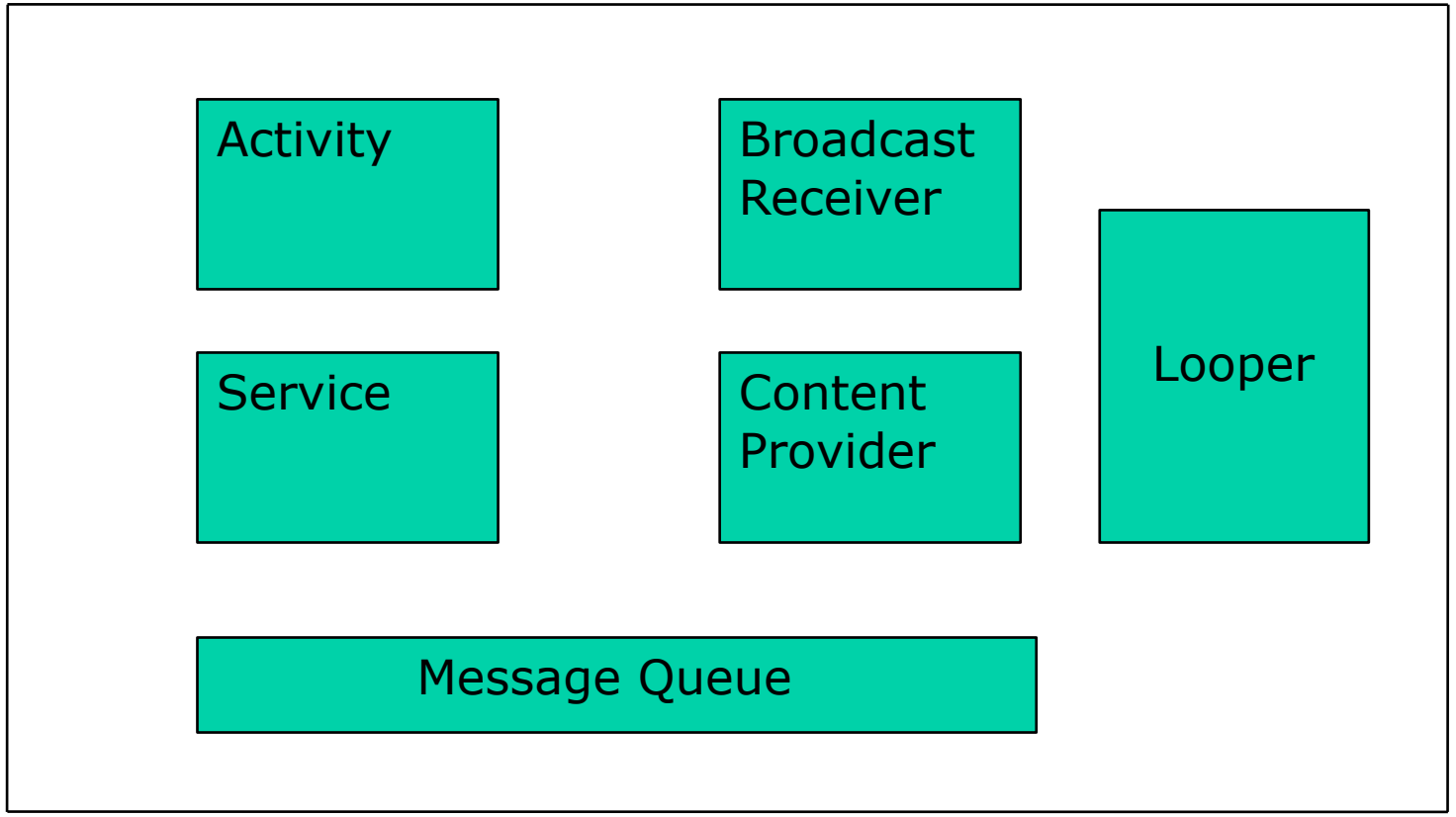
A content provider makes a specific set of the application's data available to other applications. If you don't need to share data amongst multiple applications you can use a database directly via SQLiteDatabase. (From http://developer.android.com/Reference/android/content/ContentProvider.html)

Messaging

When a request should be handled by a particular component, Android will start the application and make sure the component instance is available.

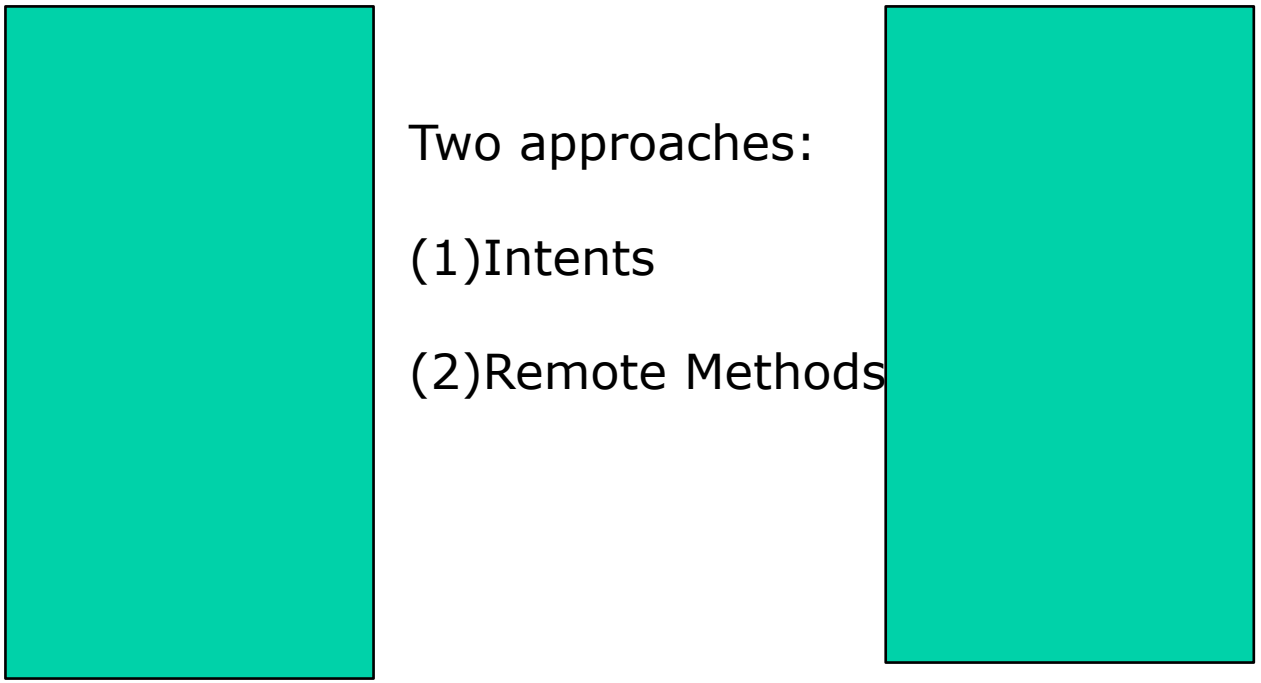


A Linux Process



Each process is started with a generated unique “user-id”. Linux is built to protect users from each other. The “user-id” provides an application sandbox.

Inter-Process Communication



Inter-Process Communication - Intents

From Google's Developer's Reference:

"*An intent is an abstract description of an operation to be performed*"

Suppose, for example, that my application wants to make a phone call:

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:4122684657");
startActivity(callIntent);
```

This is an **agile, loosely coupled, asynchronous** approach.
This is a classic example of the flexibility provided by late binding.

Inter-Process Communication - Intents

From Google's Developer's Reference:

"Three of the core components of an application — activities, services, and broadcast receivers — are activated through messages, called *intents*. Intent messaging is a facility for late run-time binding between components in the same or different applications."

"In each case, the Android system finds the appropriate activity, service, or set of broadcast receivers to respond to the intent, instantiating them if necessary. There is no overlap within these messaging systems: Broadcast intents are delivered only to broadcast receivers, never to activities or services. An intent passed to startActivity() is delivered only to an activity, never to a service or broadcast receiver, and so on."

Some Intent Constants

Constant	Target Component	Action
ACTION_CALL	Activity	Initiate a phone call
ACTION_EDIT	Activity	Display data for the user to edit
ACTION_MAIN	Activity	Start of a task
ACTION_BATTERY_LOW	Broadcast receiver	A warning that the battery is low

Intent Filters

"To inform the system which implicit intents they can handle, activities, services, and broadcast receivers can have one or more intent filters. Each filter describes a capability of the component, a set of intents that the component is willing to receive. It, in effect, filters intents of a desired type, while filtering out unwanted intents.."

From Google's Android Developers Documentation.

Inter-Process Communication - Intents

Activity$_1$
Create an Intent Object
Set its action.
Set a URI.
Set a MIME type
call startActivityForResult with the Intent object
The onActivityResult method is called when a result is available

Process A

Activity$_2$
Launched because its intent filter matches the MIME type and action

return a new Intent object to the activity that started this instance

Process B

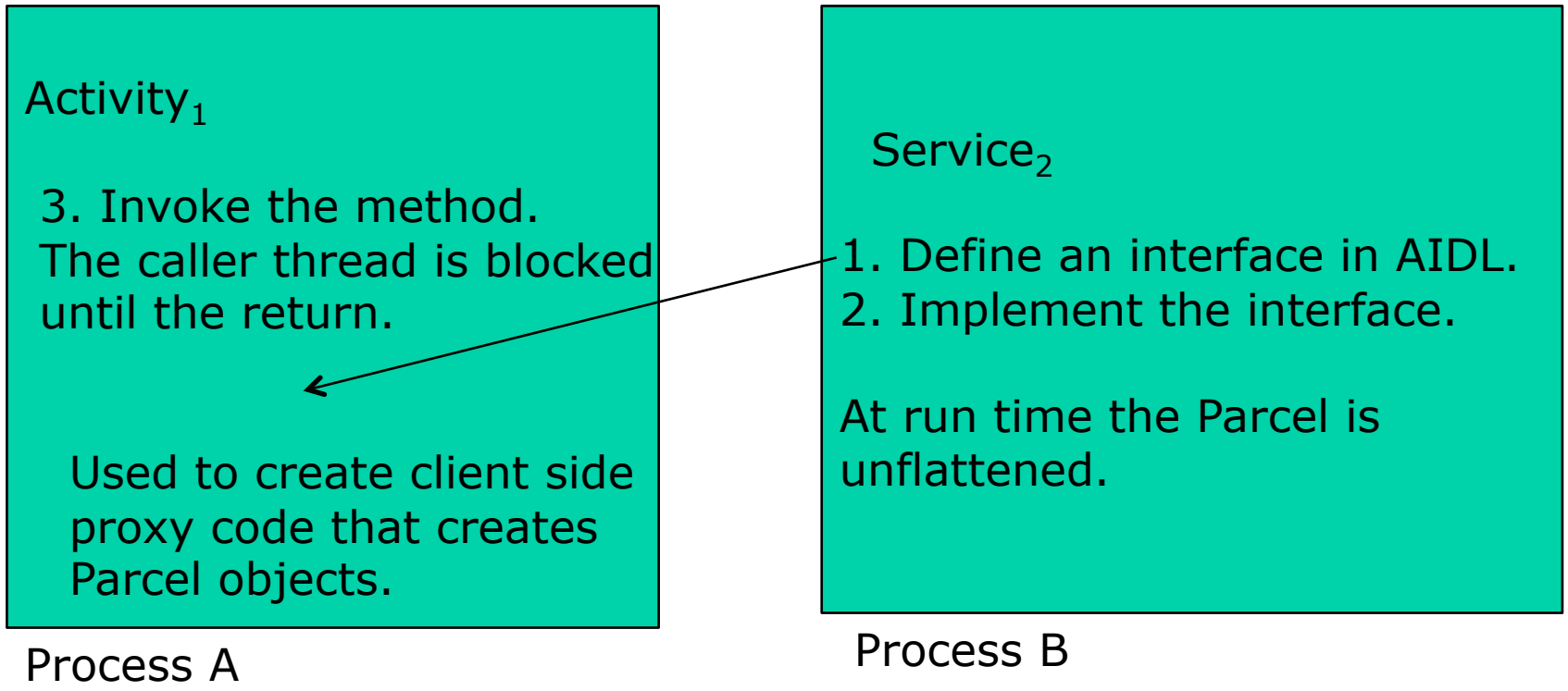
Inter-Process Communication – Remote Methods and AIDL

AIDL (Android Interface Definition Language) is an IDL language used to generate code that enables two processes on an Android-powered device to talk using interprocess communication (IPC). If you have code in one process (for example, in an Activity) that needs to call methods on an object in another process (for example, a Service), you would use AIDL to generate code to marshall the parameters.

The AIDL IPC mechanism is interface-based, similar to COM or Corba, but lighter weight. It uses a proxy class to pass values between the client and the implementation.

From Google's Android Developers Documentation.

Inter-Process Communication – Remote Methods and AIDL



AIDL is a java interface with in, out, and inout parameters.