G53MDP
Mobile Device Programming

Lecture 14 – Broadcasts, Permissions

4 Application Components

- Components are *entry points*
  - Activity
    - To a user interface as part of a task
  - Service
    - To perform a long-running background task or computation, or access to a specific system resource
  - ContentProvider
    - To storage and provision of data
- What else?
  - How else to drive activity within an application?
  - What is a missing entry point?

BroadcastReceiver

- Respond to system-wide broadcast announcements
  - The user has not necessarily done something, but the OS / phone / another application has
    - The screen has turned off, the battery is low, a new SMS has arrived, the phone has booted
  - Can be sent by an application, or received by an application from the OS
- Intents are sent to specific Activities / Services
  - Explicitly or implicitly via Intent filters, URI provision
- Broadcasts are sent to anything that cares to listen
  - System-wide Intent broadcasts

Broadcasts

- Broadcasts are Intents
  - Send an Intent to start an Activity
  - Send an Intent to start a Service
  - Send an Intent to trigger Broadcast Receivers that subscribe to that particular class of Intent
    - Can define our own Broadcast Intents
    - Cannot send system Intents (battery, screen etc), as the security model prevents it
      - Why?
  - Again a messaging wrapper around Binder

BroadcastReceiver

- Declare in AndroidManifest.xml
  - <activity>
  - <service>
  - <provider>
  - **<receiver>**
- Specify broadcast intents we are interested in
  - Listening to system intents may require certain permissions
    - i.e. phone state – why?
- Receiver registered at boot-time / install-time
  - Again, another **entry point** to our application

```
<receiver android:name="MyReceiver" >
    <intent-filter>
      <action android:name=”com.example.martinbroadcast" />
    </intent-filter>
</receiver>
```

Implementing a BroadcastReceiver

- Subclass BroadcastReciever
  - Specify which Intents we are interested in receiving
    - Permissions permitting
- Implement the onReceive() method
- Then…
  - Send a Message to our application to change our behavior
    - Reduce the audio volume, play a sound
  - Start an Activity
    - “Never start an Activity in response to an incoming broadcast Intent.”
  - Start a Service
  - Show a Notification
    - Alert the user that there is something that they need to interact with

Normal and Ordered Broadcasts

- sendBroadcast(intent)
- sendOrderedBroadcast(intent, receiverPermission)

```
<receiver android:name="MyReceiver" >
     <intent-filter android:priority="2" >
        <action android:name=”com.example.martinbroadcast" />
     </intent-filter>
</receiver>
```

- Preferred receivers given the chance to consume the broadcast before it is delivered to less preferred receivers
  - How many things should notify you that you have a new message?
  - In onReceive()
    - abortBroadcast() to stop the chain of delivery
  - Who is *allowed* to receive is a problem for *permissions*

Non-Sticky and Sticky Broadcasts

- Non-Sticky
  - If you weren't listening at the time you've missed it
- Sticky Intents are cached by Android
  - New Intents overwrite older Intents they match
- Delivered
  - When BroadcastReceivers are dynamically registered
    - Cached sticky Intents matching the specified IntentFilter are broadcast to the BroadcastReceiver
    - One matching sticky Intent is returned to the caller
  - isInitialStickyBroadcast()
    - How old is the broadcast?
    - ACTION_BATTERY_CHANGED

Sticky Broadcasts

- *Sticky broadcasts are **global** to the system*
- *And because of this, performing a sticky broadcast is multiple orders of magnitude slower than just implementing direct calls within your own app*
  - *IPC for each receiver to register, IPC to the system to send it, IPC from the system back to your app to deliver it, marshalling and unmarshalling of all the data within the Intent over both IPCs*
- *More than that, there is NO protection on them, so any other application can watch your sticky broadcasts, or even send their own values back to you*
- *Now I am really regretting that I made that function public. :/*

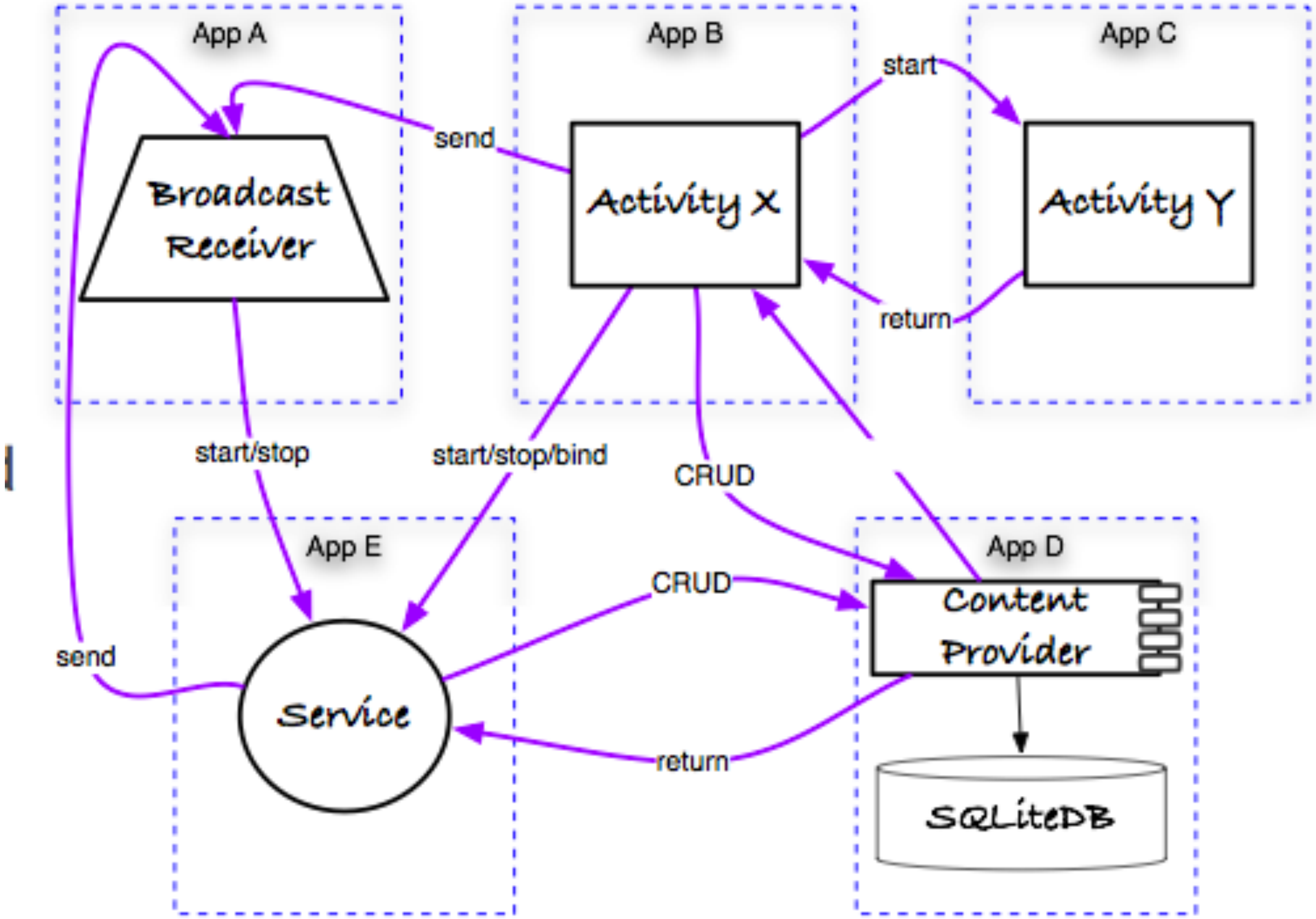
Local Broadcasts

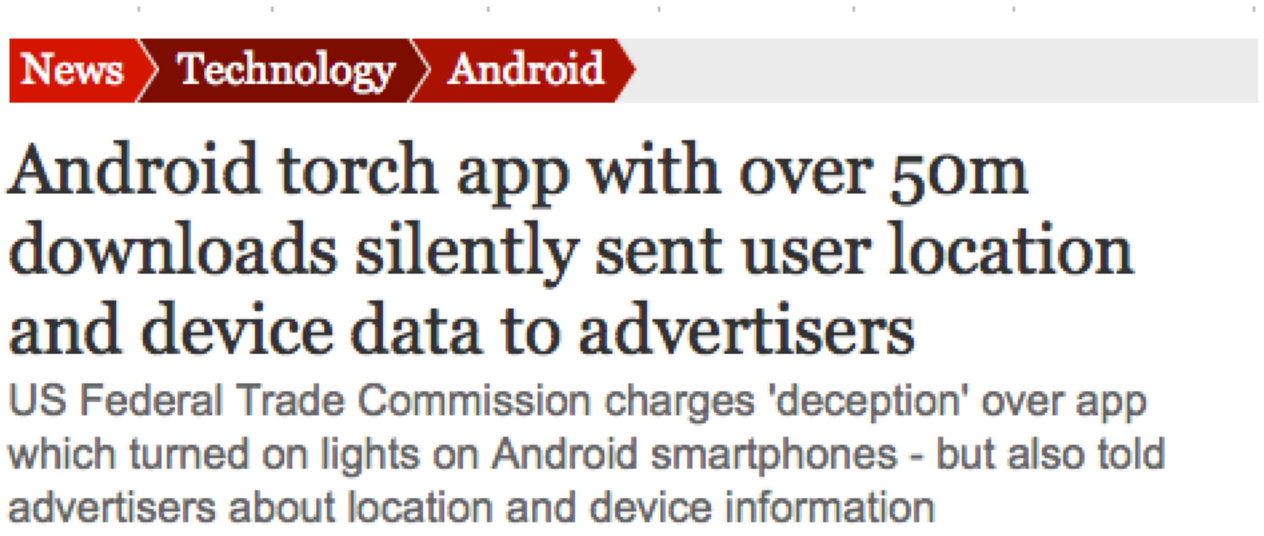
- Like the normal BroadcastReceiver but only supports **local** broadcasts
  - Within the application
- Data broadcast will not leave the application
  - No data leakage
- Other applications cannot send broadcasts to our application
- More efficient than sending a global broadcast
  - No IPC
- Must register / sendBroadcasts using the LocalBroadcastManager
  - Programmatically rather than via the manifest

Caveats

- Either the sender or receiver of a broadcast can implement permissions
  - Control who can send broadcast intents to my application
  - Control who can receive the intents that my application broadcasts
  - Limit broadcasts to my application components only
- Lifecycle
  - A receiver handling broadcasts is considered to be running in the foreground (albeit briefly)
  - Process is aggressively killed once onReceive() has returned
    - No binding, no startActivityForResult
  - Long-running code should start a Service instead
    - As with any other Activity
- The application receiving the broadcasts must have been explicitly started / not explicitly stopped
  - Applications cannot intercept broadcasts without the user having some awareness that they have given it permission
- Can register / unregister for broadcast events programmatically
  - Most things that the manifest specifies can be done programmatically

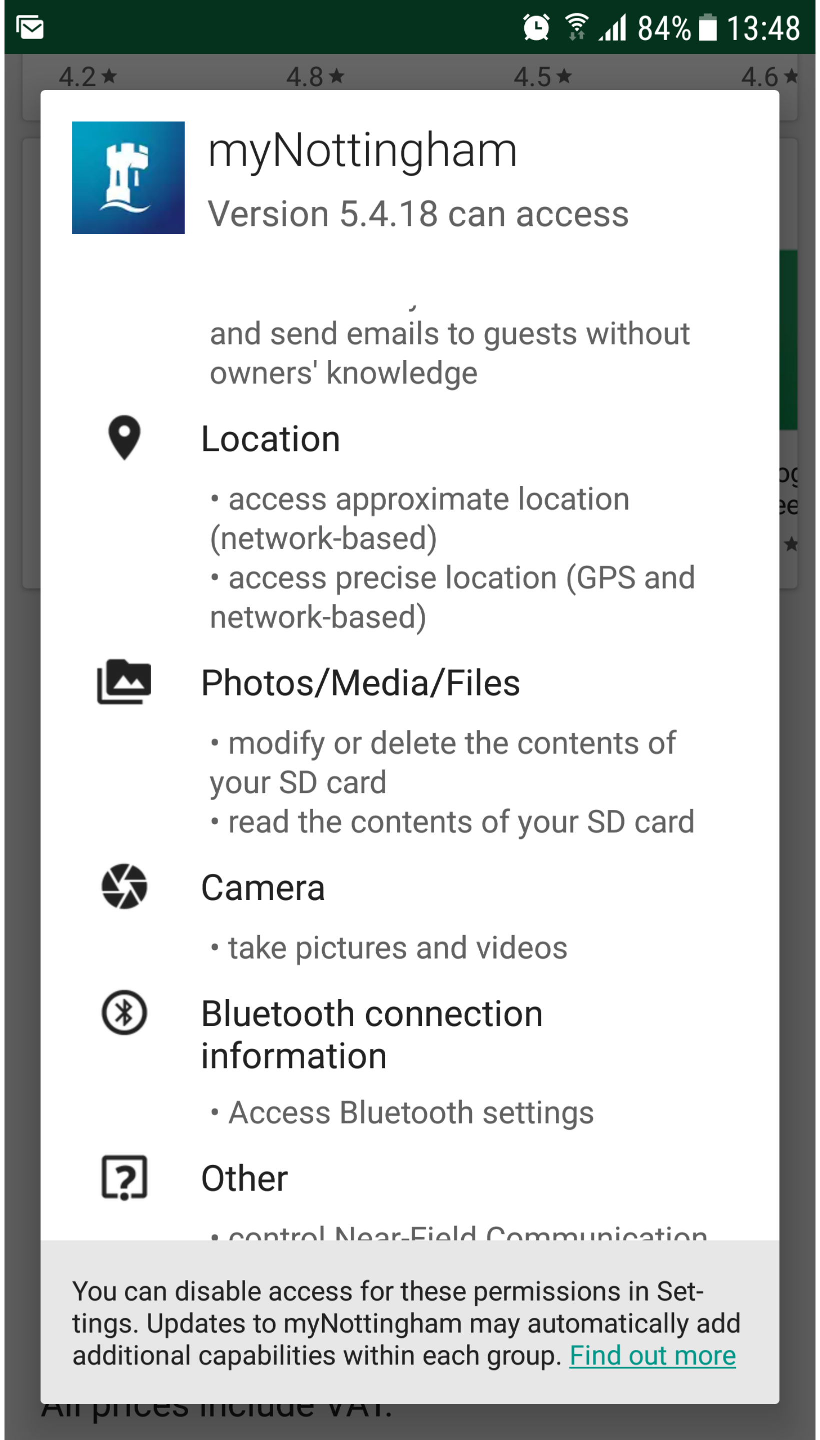
Let’s have a look…


App A
App B
App C
Broadcast Receiver
Activity X
Activity Y
send
start
return
start/stop
start/stop/bind
CRUD
App E
App D
CRUD
Content Provider
send
Service
return
SQLiteDB


News › Technology › Android

Android torch app with over 50m downloads silently sent user location and device data to advertisers

US Federal Trade Commission charges 'deception' over app which turned on lights on Android smartphones - but also told advertisers about location and device information


13:48 84%

myNottingham

Version 5.4.18 can access

and send emails to guests without owners' knowledge

Location

• access approximate location (network-based)
• access precise location (GPS and network-based)

Photos/Media/Files

• modify or delete the contents of your SD card
• read the contents of your SD card

Camera

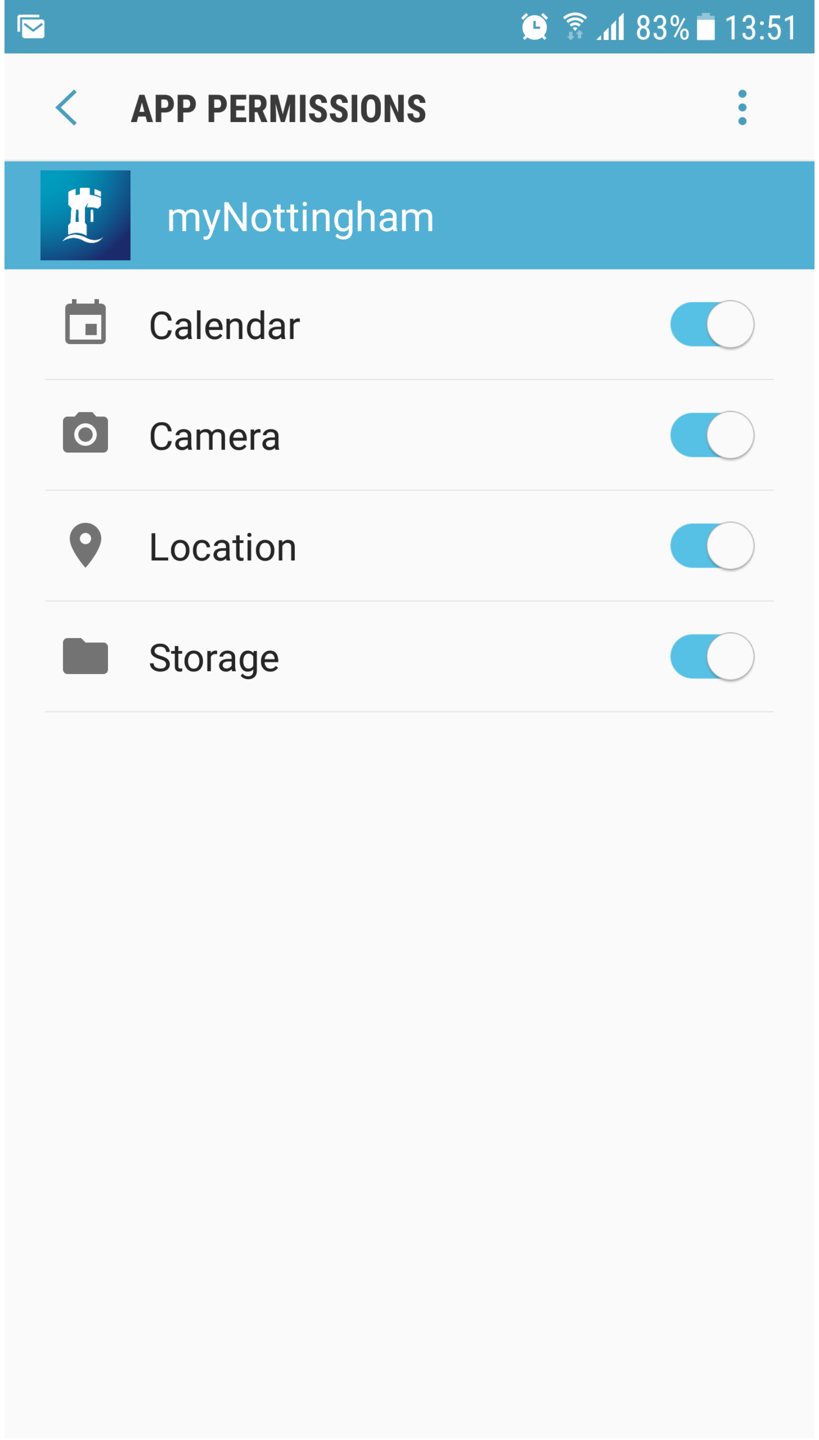
• take pictures and videos

Bluetooth connection information

• Access Bluetooth settings

Other

You can disable access for these permissions in Settings. Updates to myNottingham may automatically add additional capabilities within each group. Find out more


13:51 83%

APP PERMISSIONS

myNottingham

Calendar — on
Camera — on
Location — on
Storage — on

Old Permissions

- Show permissions required at install time
  - Not prompted again regarding permissions at run-time
- Why?
  - Not yet made a commitment (financial, mental) to the application
    - Can compare to other applications
  - Not per session / at run-time
    - “Seamless” switching between Activities / applications
    - Would slow down the user experience
    - Train users to click “ok” repeatedly without considering the implications

Permissions in the wild

- Study participants displayed low attention and comprehension rates: both the Internet survey and laboratory study found that 17% of participants paid attention to permissions during installation, and only 3% of Internet survey respondents could correctly answer all three permission comprehension questions.
- http://dl.acm.org/citation.cfm?id=2335360

Permissions

- No access by default
  - Control access to specific mechanisms
- Applications can offer protected access to resources and data with **permissions**
  - Permissions explicitly granted by users / the system
- Permission architecture
  - Applications statically declare permissions
    - Required **of** components interacting with them
      - You must have this permission to interact with me
    - Required **by** components they interact with
      - I will need these permissions
  - Android requires user’s consent for specific permissions

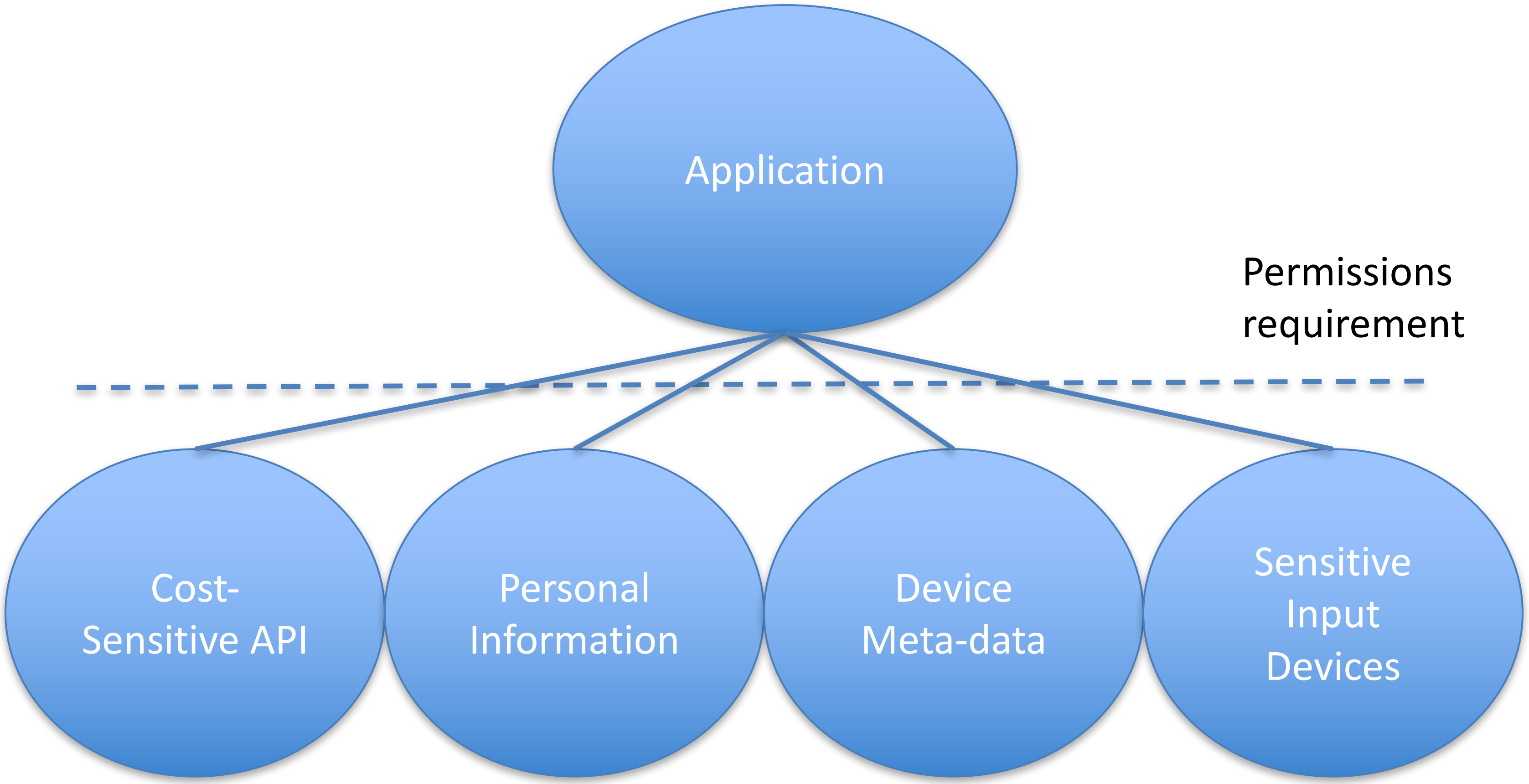
Permissions are Fragmented

- Android 6.0/23 and later
  - Permissions are granted at runtime
  - Permissions can be revoked at any time
  - > We need to check whether we have permission to do something
- Android 5.1 or lower, or targeting 22 or lower
  - Permissions granted at install time
    - Or the application will not be installed

Normal or Dangerous

- Permissions are classified as
  - Normal
    - Do not directly risk the user's privacy
      - Network state, Internet, Alarms, Wallpaper…
    - Granted automatically
      - But still must be declared in the manifest
  - Dangerous
    - Potentially do risk the user's privacy
    - The user must explicitly approve the permission request
      - Need to think what to do if permission is denied

Permissions



Permissions

- Cost-Sensitive APIs
  - Telephony
  - SMS/MMS
  - In-App Billing
  - NFC Access
- Personal Information
  - Contacts, calendar, messages, emails
- Device Meta-data
  - System logs, browser history, network identifiers
- Sensitive Input Devices
  - Interaction with the surrounding environment
  - Camera, microphone, GPS

Common Permissions

- android.permission.ACCESS_FINE_LOCATION
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.INTERNET
- android.permission.WAKE_LOCK

Using Permissions

- Applications can define new permissions in the manifest

  <permission android:name="com.example.martin.ANIMALS"

  …

  </>

  - Do we really need a new permission?
  - normal / dangerous
    - Signed – can only be used by apps from this *developer*
  - "Readable" explanation of the new permission
- Applications can require components interacting with them to have a specified permission, set in the manifest
  - By default all permissions apply to all components hosted by the application
    - Activities, Services etc.
  - Or per component permission requirements

Using Permissions

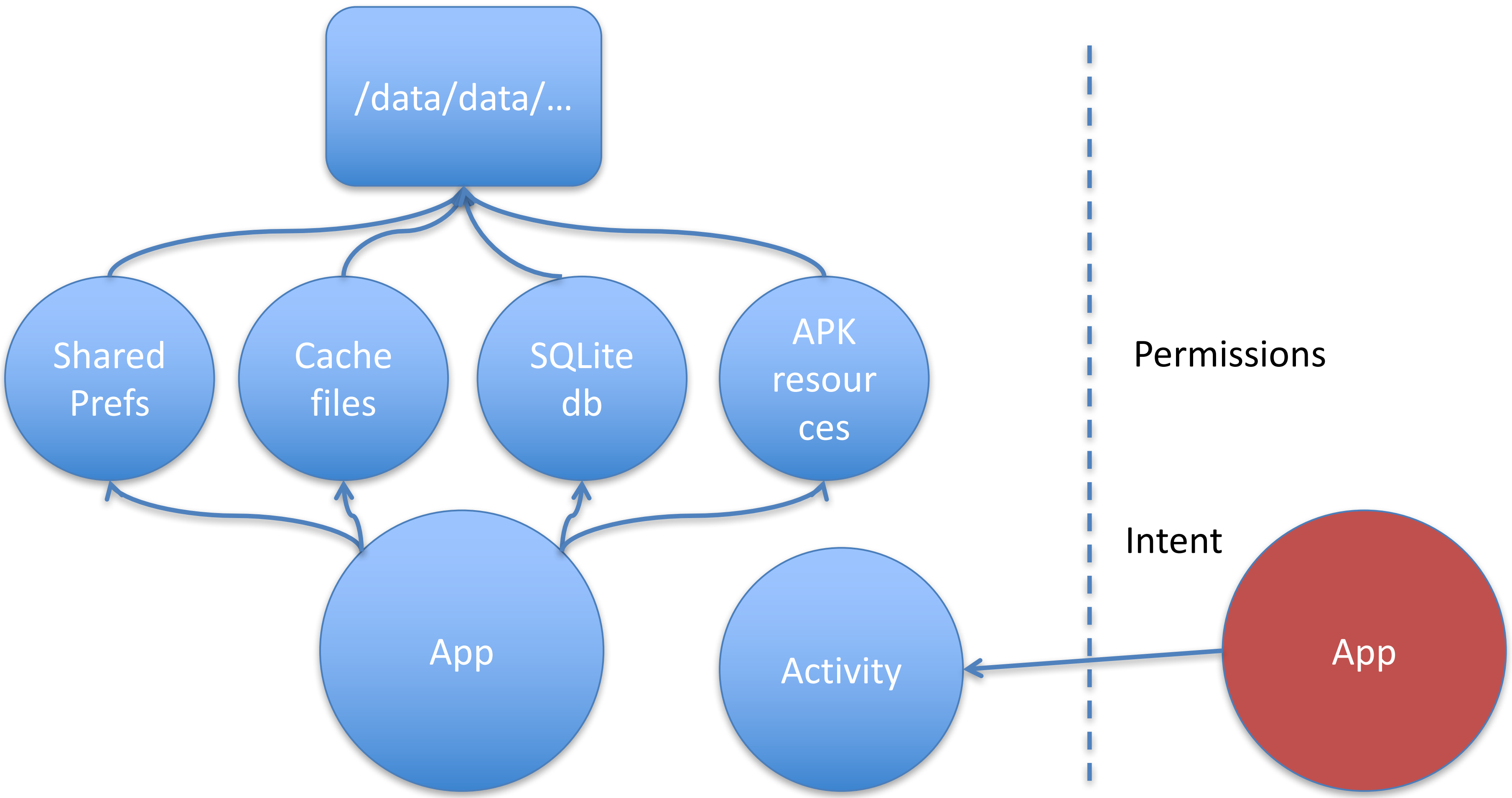
- Specify that an Application **uses** a permission

```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

- Specify that an Application **requires** a permission
  - The app must **use** (i.e. as above) permissions it **requires**

```
<provider
    android:permission="android.permission.READ_CONTACTS”
    android:authorities="com.example.martincontentprovider.MyProvider”
    android:multiprocess="true”
    android:name="com.example.martincontentprovider.MyProvider">
</provider>
```

Sharing Data – is this good enough?



Component Permissions

- Activity
  - Restricts which components can start the activity
  - Checked within execution of:
    - startActivity()
    - startActivityForResult()
- Service
  - Restricts which components can start or bind to the associated service
  - Checked within execution of:
    - Context.startService()
    - Context.stopService()
    - Context.bindService()
- ContentProvider
  - Restricts which components can read or write to a ContentProvider
- BroadcastReceiver
  - Restricts which components can register to receive a certain Broadcast
- Throw SecurityException on permissions failure
  - Usually as we've forgotten to ask for permission during installation

Runtime Permissions

- Each time we want to do something “dangerous”

```
int permissionCheck =
ContextCompat.checkSelfPermission(thisActivity,
Manifest.permission.WRITE_CALENDAR);
```

- shouldShowRequestPermissionRationale(…)
  - True if the user has previously denied the request
- requestPermissions(…)
- onRequestPermissionsResult(…)
  - Either do the dangerous thing, or gracefully degrade the functionality of the app

Permissions vs Use

- Read your text messages
  - To confirm your phone number via text message (if you’ve added it to your account)
- Read/write contacts
  - To import and sync your phone’s contacts to Facebook, or vice versa (think updating contact images)
- Add and/or modify calendar events and send emails to guests without your knowledge
  - To see your Facebook events in your phone’s calendar
- Read calendar events plus confidential information
  - To check your calendar for you to see if you have something already scheduled for the time of the Facebook event you’re currently viewing
- Good practice to *explain why* an application needs a permission, especially if now *not* having it will prevent it from functioning

Temporary URI Permissions

- Applications making use of multiple Activities
  - “Access to the mail should be protected by permissions, since this is sensitive user data. However, if a URI to an image attachment is given to an image viewer, that image viewer will not have permission to open the attachment since it has no reason to hold a permission to access all e-mail.”
  - Allow access to specific URIs, not the whole provider

```
android:grantUriPermissions="true”
myIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
```

- Temporary URI permissions last while the stack of the receiving Activity is active

References

- http://developer.android.com/guide/topics/providers/content-providers.html
- http://developer.android.com/guide/components/fundamentals.html
- https://developer.android.com/reference/android/content/BroadcastReceiver.html
- https://developer.android.com/training/permissions/requesting.html