Case Studies

PROFESSIONAL EDUCATION

CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

Mobile Phone Security: Android

Nickolai Zeldovich

Associate Professor

Computer Science and Artificial Intelligence Laboratory (CSAIL)

Massachusetts Institute of Technology

MIT PROFESSIONAL EDUCATION

CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

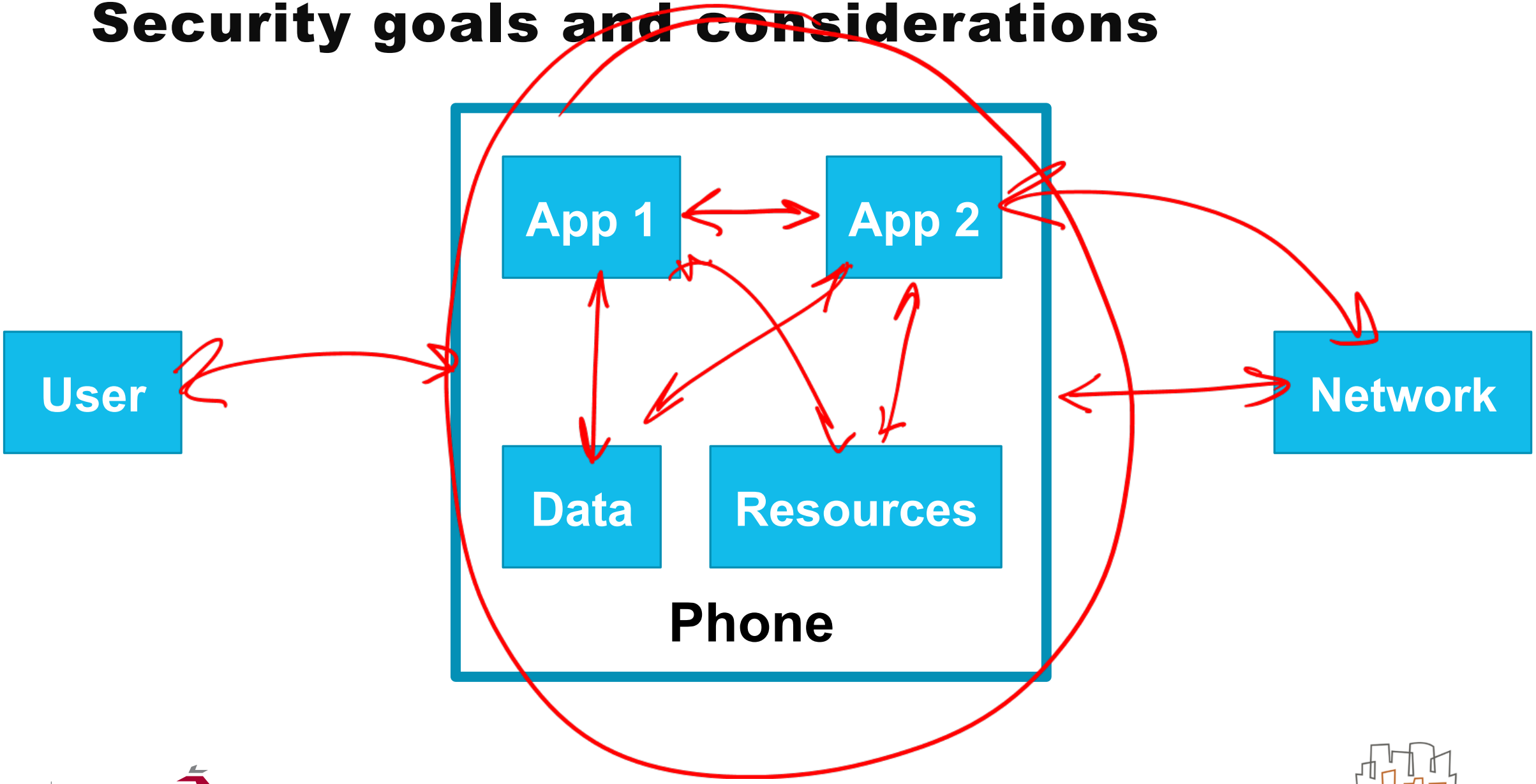
Segment 1: Introduction

- What's different about mobile phones?
- Security goals and considerations
- Threat model

Mobile phones in context

- Desktop applications
  - Good: user can choose app for each task (email, photo, ..)
  - Good: apps can interact, share files
  - Bad: no isolation between apps, data
- Web apps (when Android was designed)
  - Good: strong isolation between apps, no explicit “install”
  - Bad: typically requires server, not a great fit for offline
  - Bad: limited interaction, hard to share data between apps
  - Bad: limited functionality (background apps, camera, …)

Security goals and considerations



Threat model: applications

- Application has bugs, gets compromised
- Adversary buys good app, “corrupts” it
- Accidentally install malicious application

- Common denominator:
  be prepared for malicious applications

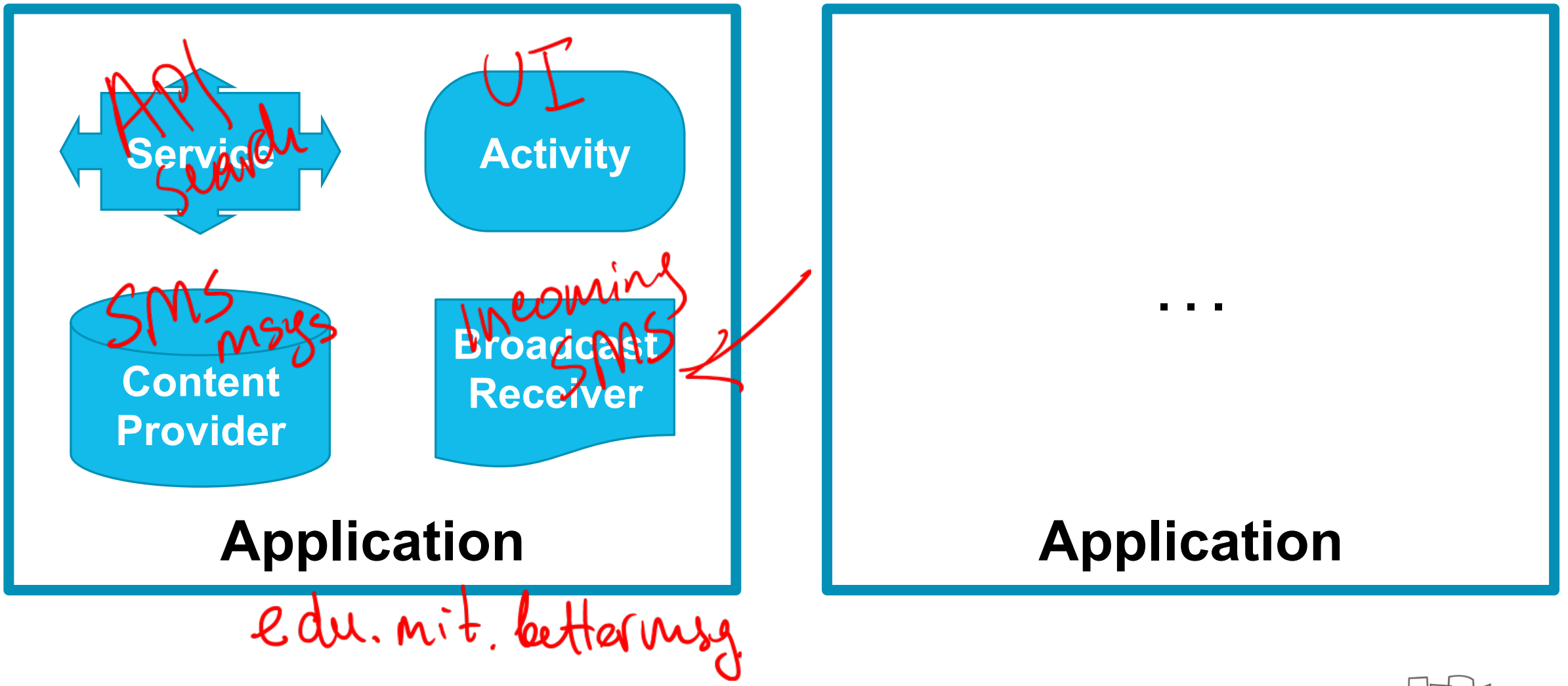
Roadmap

- Android application model
- Isolating applications
- Controlled sharing
- User-approved policies
- Security model extensions
- What worked and what didn't?

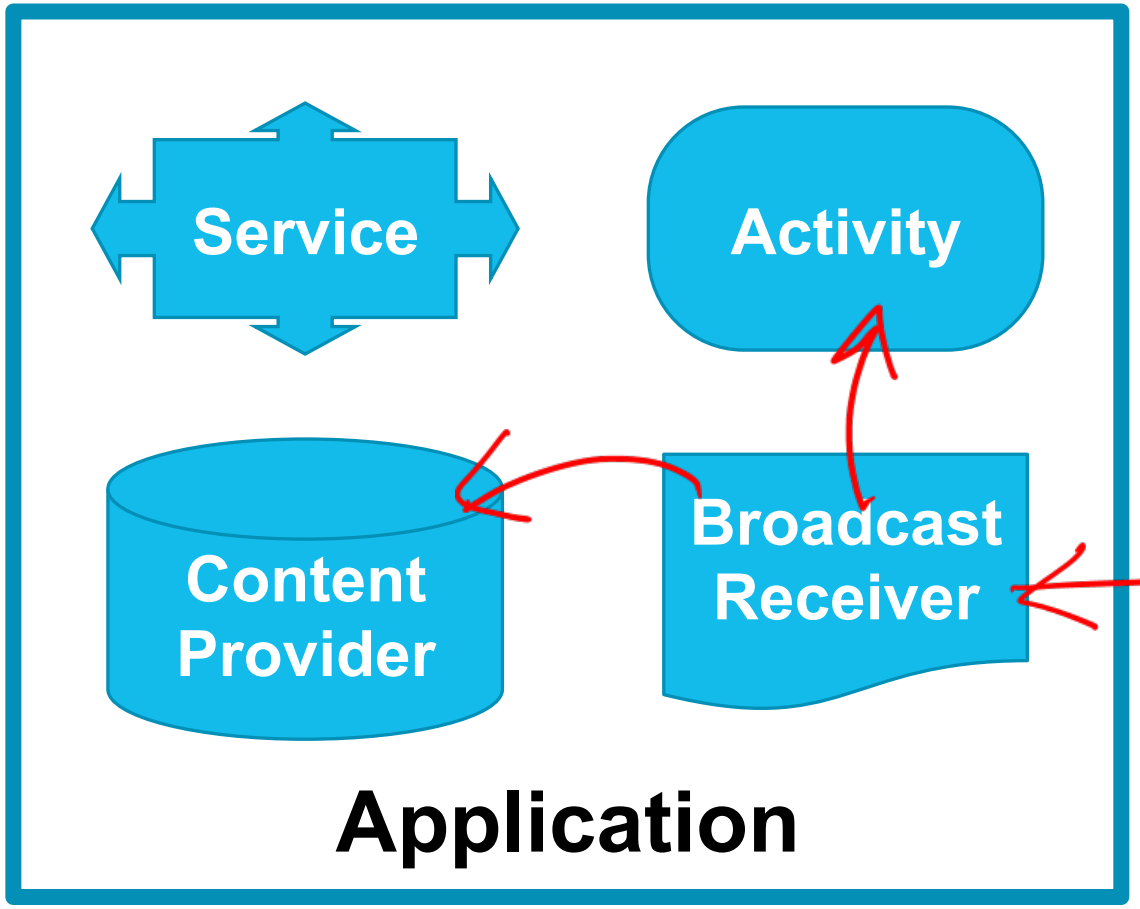
Segment 2: Android applications

- Components
- Intents
- Manifests

Components: application's entry points



Intents: application interaction



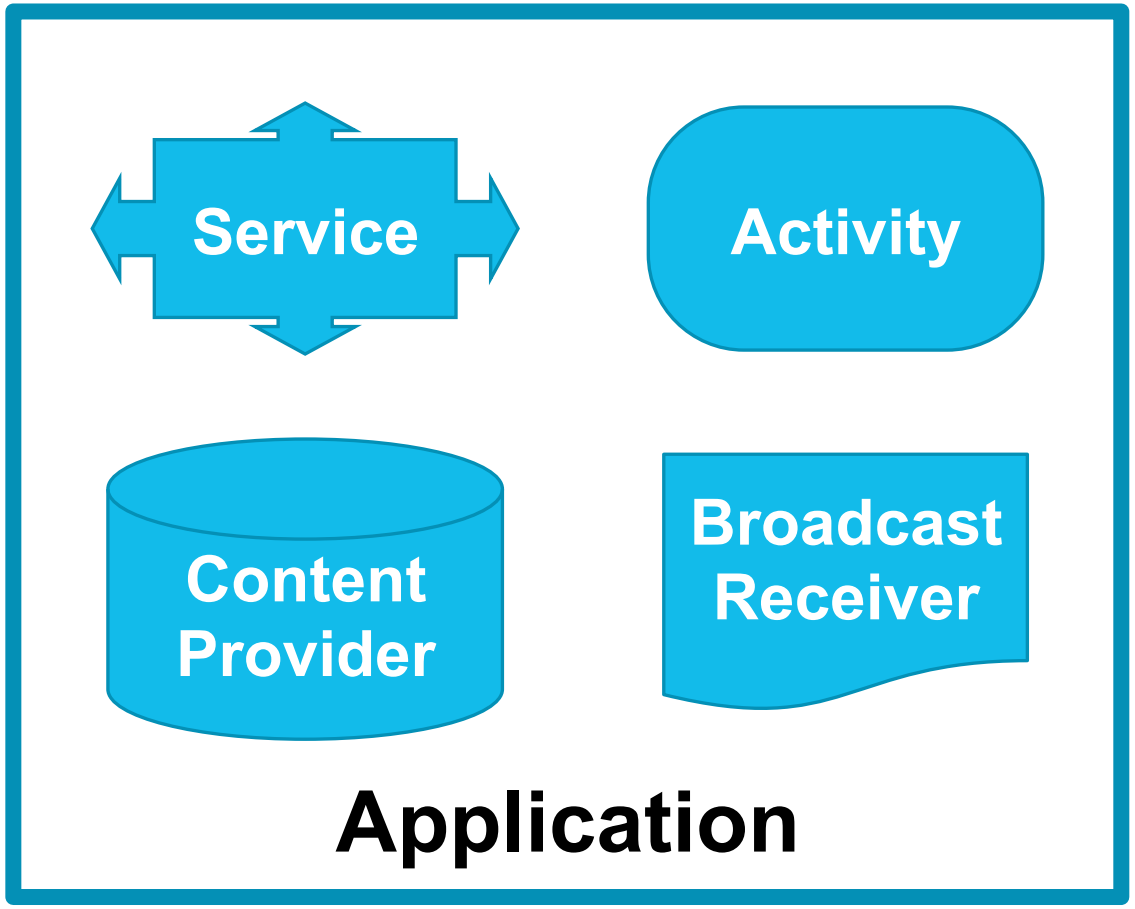
- Intent message:
  - Component name
  - Data URI
  - Action

SMS

Explicit

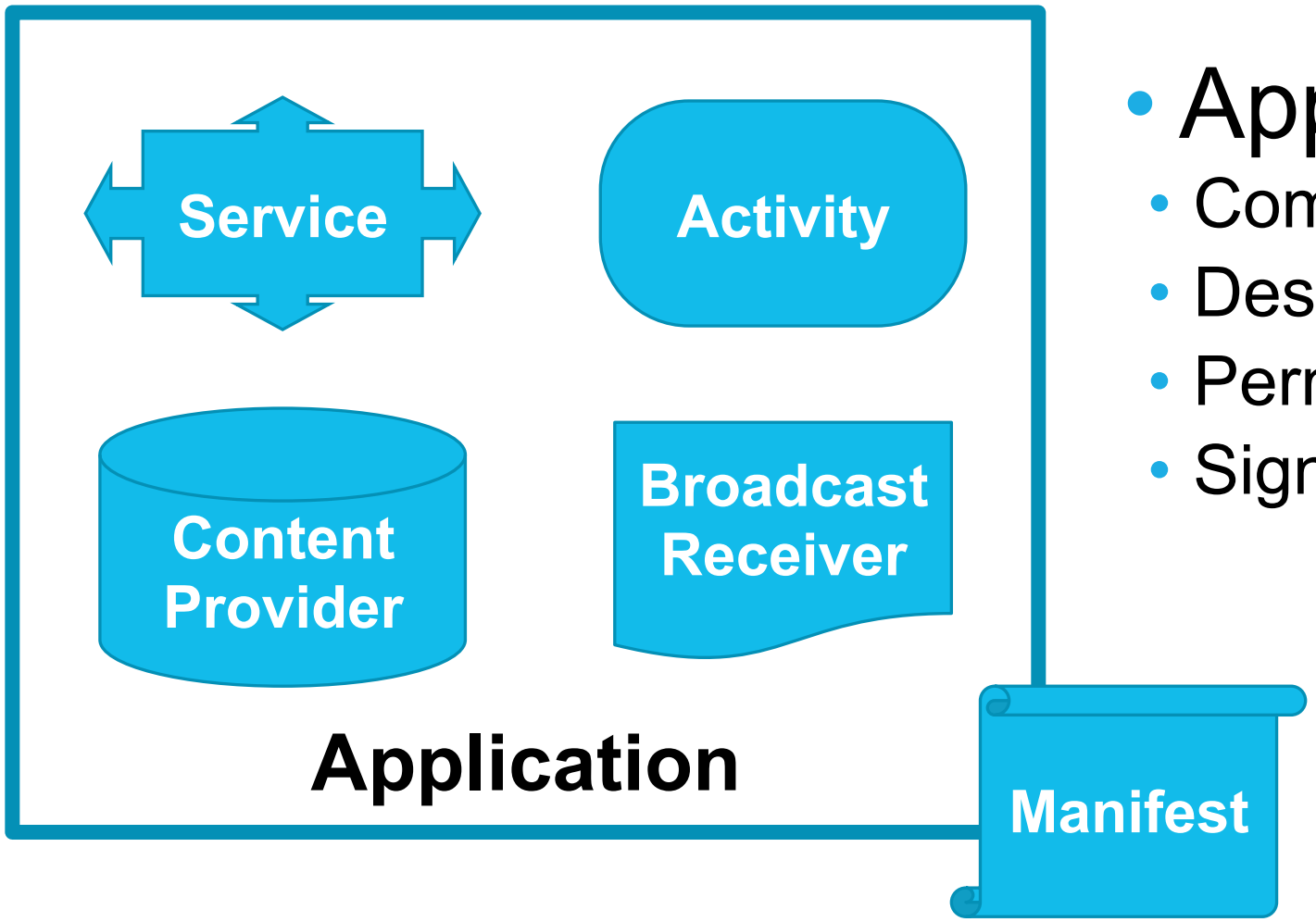
Implicit.

Non-intent resources, for performance



- Network (sockets API)
- Camera
- Bluetooth
- …

Manifests



- App manifest:
  - Components
  - Desired intents
  - Permissions
  - Signature

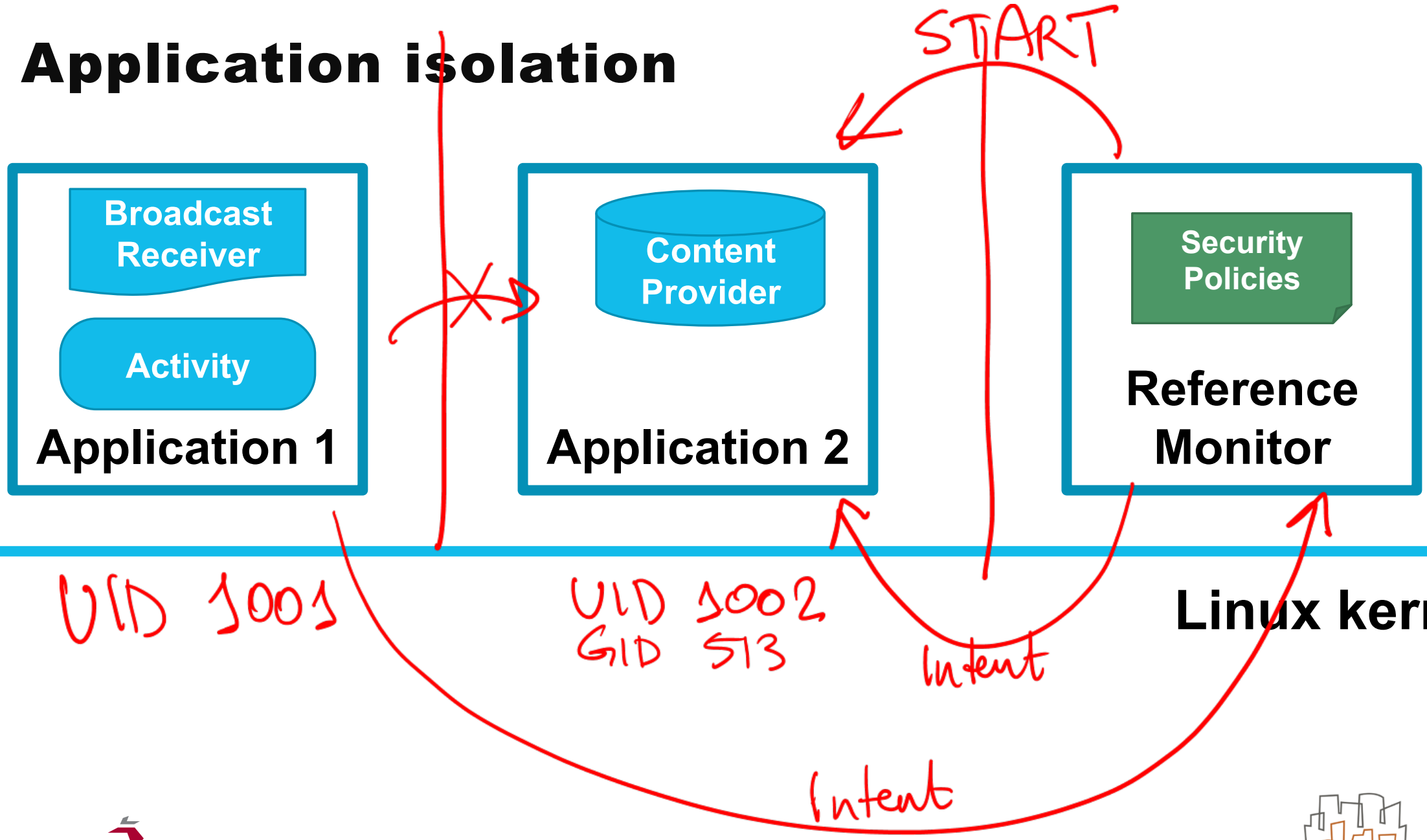
Android application model

- All applications follow the same model
  - Includes built-in applications: phone, contacts, browser, …
  - Enables flexibility: almost any app can be replaced
  - Also helps in implementing a uniform security plan
- Implicit intents enable app interaction
- Components, intents made explicit
  - Gives Android visibility into application structure
  - Helps specify, enforce finer-grained security policies

Segment 3: Android security model

- Isolation
- Reference monitor
- Permissions
- Example

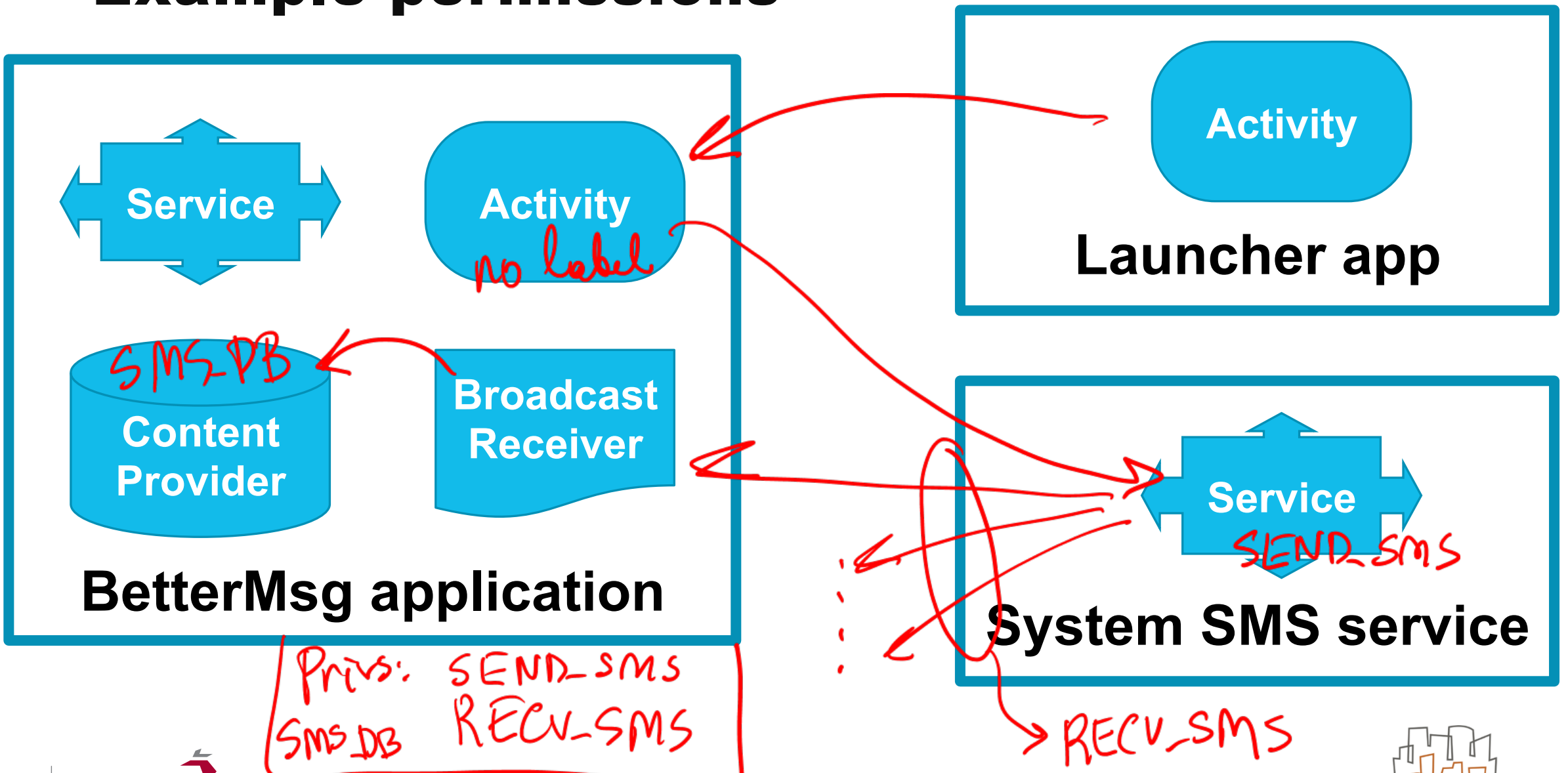
Application isolation



Specifying policies: permissions

- Policies described in terms of *permissions*
  - Permission named by string label, e.g. android.permission.SEND_SMS
- *Application* has privileges: a set of labels
  - Captures the set of privileges that an application can use
- *Component* has a label protecting it
  - Describes what privileges are needed to access component
  - Resources (e.g., network) have a label that maps to GID

Example permissions



Segment 4: Policy specification

- Manifests
- User approval
- Permissions
- Permission types

Policy in manifest

- Label *protecting* each component
- Privileges (labels) *requested* by application
- New permissions declared by this application

User approves privileges

- Granting privileges to app can be dangerous
  - Recall our assumption: application might be malicious!
- Android's approach: let the user decide
  - User presented with list of requested privileges
  - Decides whether to install the app (with privileges), or abort
- No approval needed for component labels
  - Component labels are *restrictions*, not *privileges*

Permissions in detail

- Need to tell the user more about the labels!
- Android *permission* consists of a label and:
  - User-visible name
  - User-visible description
  - Category (costs money, private data, …)
  - Type (normal, dangerous, signature)
  - Specified in manifest of app that defined permission

Permission types

- Normal: no user approval needed
  - E.g., change wallpaper

① audit ② least privilege

- Dangerous: require user approval
  - E.g., read contacts, send SMS message

- Signature: only for apps from same developer

Permissions summary

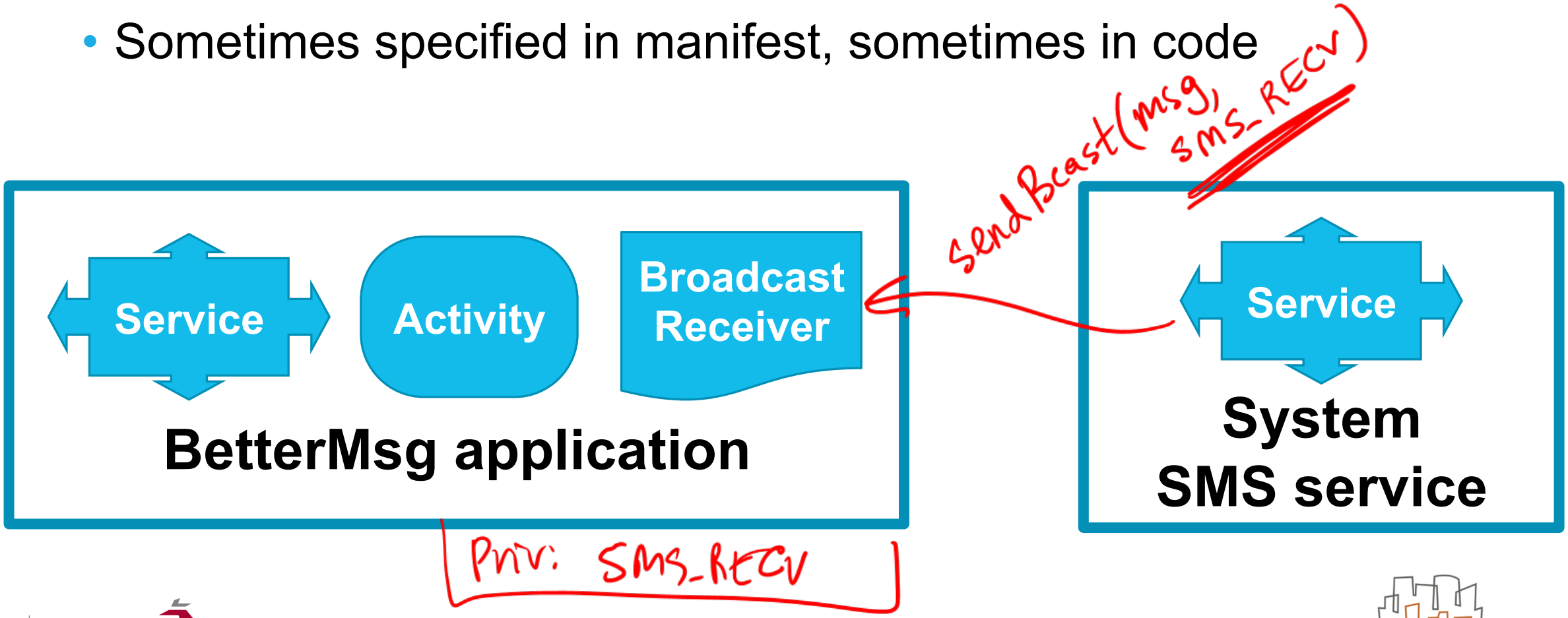
- Manifest declares app's security policy
  - Allows reasoning about security (almost) without looking at code
  - User can examine, approve declared policy at install time
  - Helps in auditing security configuration
- Uniform mechanism for all applications
  - Applications can introduce their own permissions
  - Treated same way as standard / default applications
  - Eliminates need for applications to duplicate the same logic

Segment 5: Extensions and issues

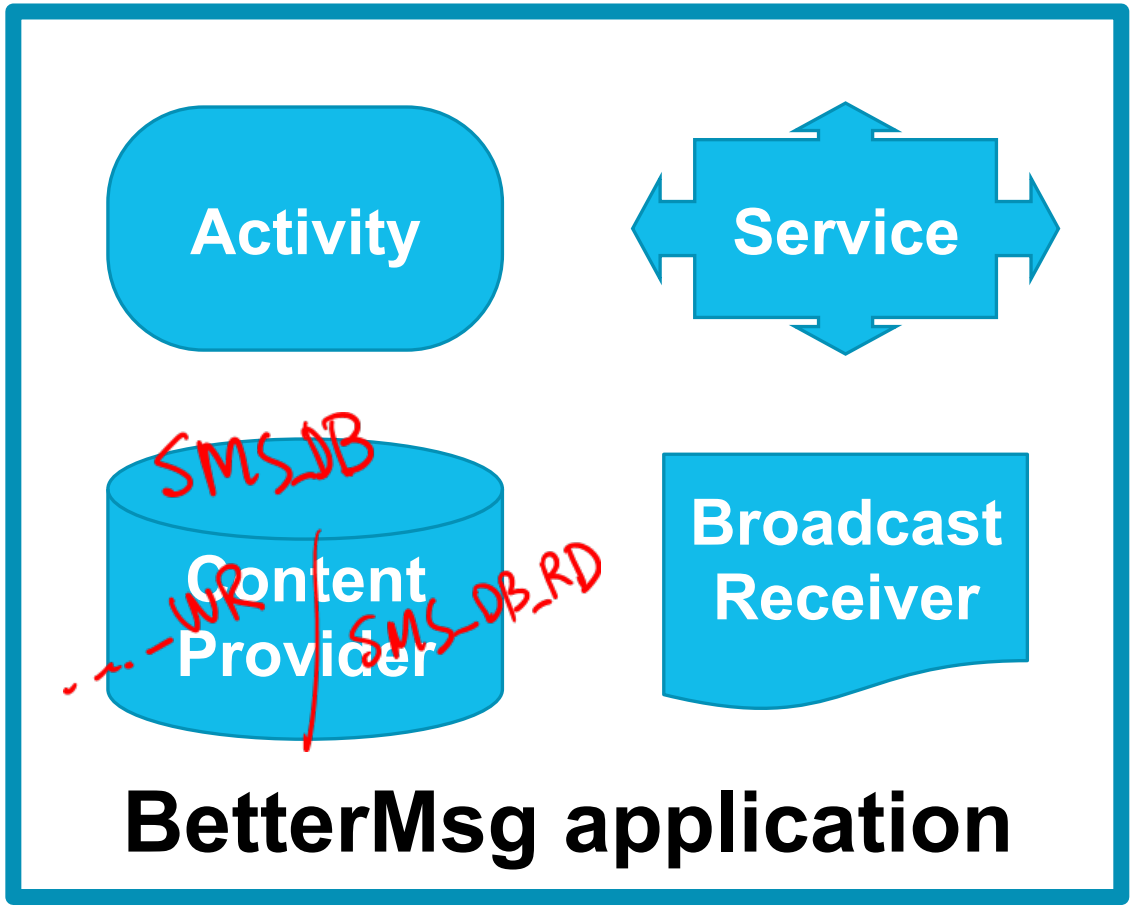
- Intent protection
- Finer-grained access control
- Secure defaults
- Naming
- App verification

Intent secrecy and integrity

- Who can send or receive some intent?
  - Sometimes specified in manifest, sometimes in code

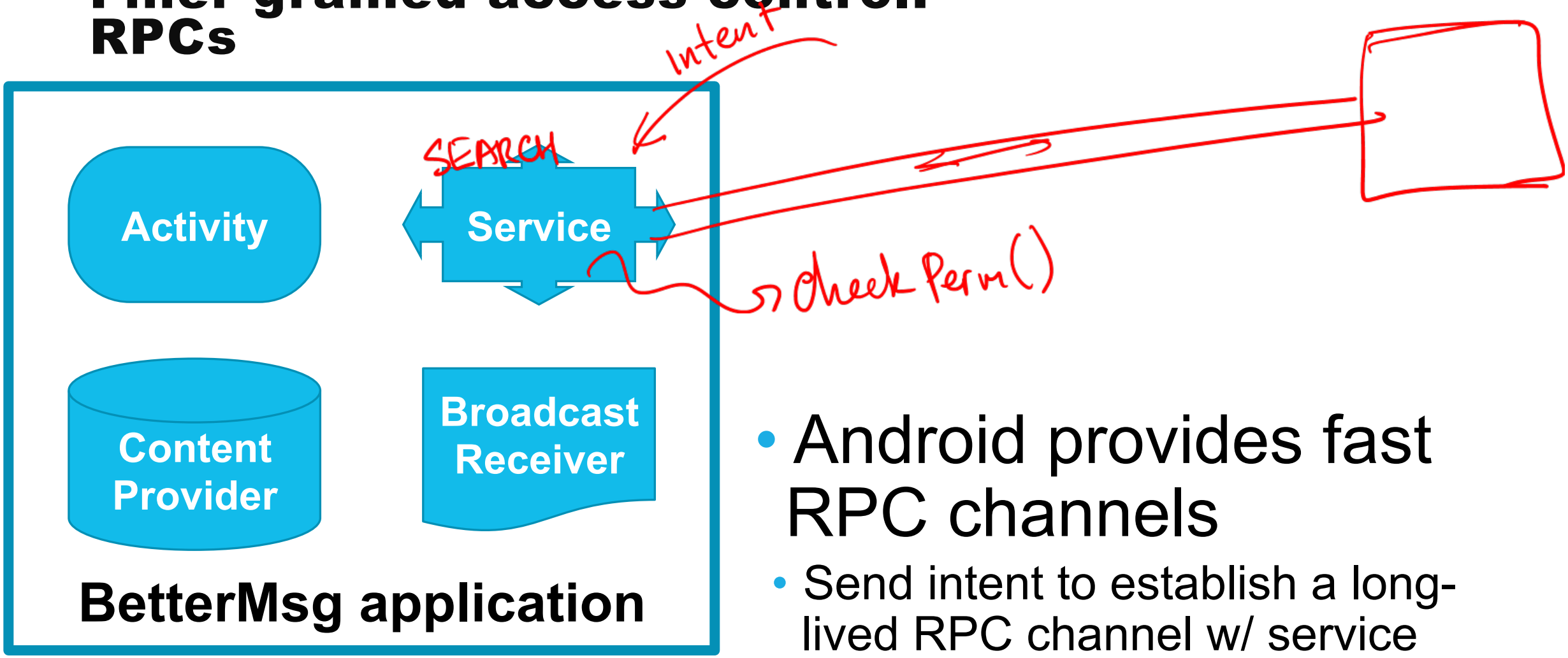


Finer-grained access control: content providers



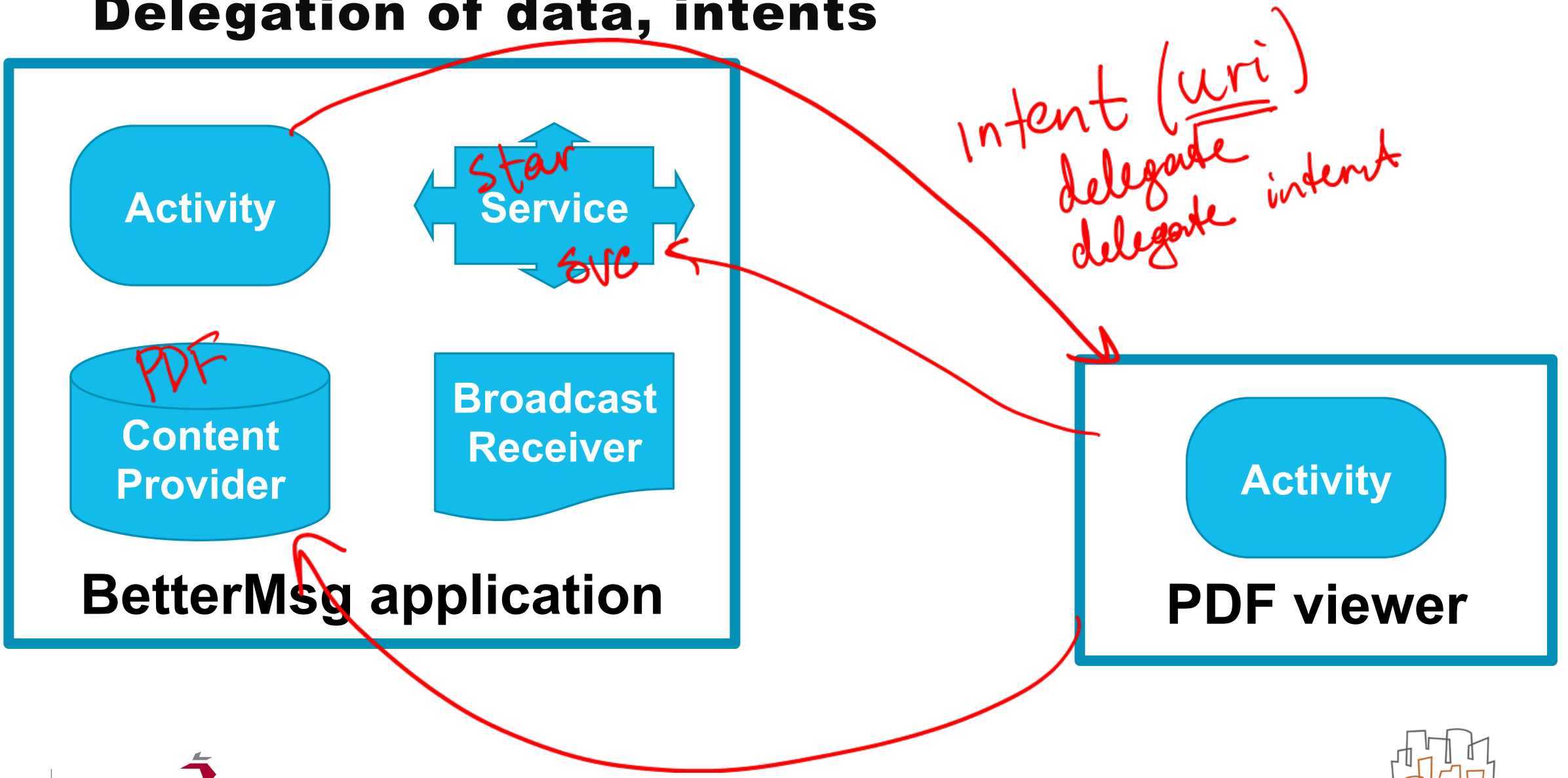
- Content providers need different permissions for reads, writes
- Solution: separate read and write labels
  - Good: preserves declarative specification of policy in manifest

Finer-grained access control: RPCs

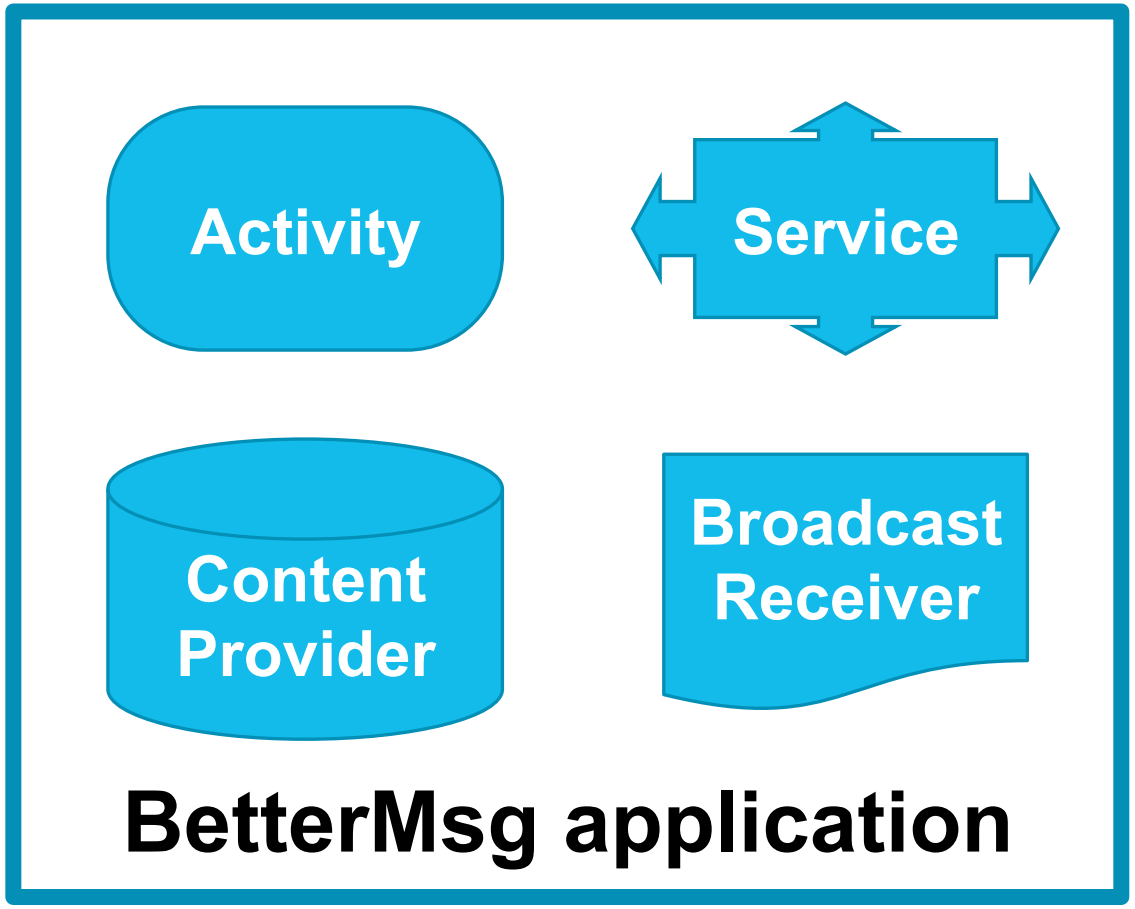


- Android provides fast RPC channels
  - Send intent to establish a long-lived RPC channel w/ service

Finer-grained access control: Delegation of data, intents



Secure defaults: public vs private components



- Easy for developers to forget to assign labels
- Everything seems to work with no labels!
- Important to provide secure defaults (public vs private components)

Naming issues

- Android names are first-come-first-served
  - App names (e.g., edu.mit.better-msg)
  - Permission names (e.g., edu.mit.better-msg.READ_MSGS)
- Makes it difficult to rely on names for security
  - Can’t be sure what it means to send intent to a component name
  - Can’t even be sure what permission a label refers to
  - Adversary could register important permission as type “normal”
- Hard problem – might need a name authority

Server-side app verification

- Some issues are hard to prevent locally
  - Misleading descriptions trick user into installing malicious app
  - Malicious clones of popular applications
  - Apps that exploit weaknesses in Android platform
  - Apps that are a nuisance to the user
- Complementary: global analysis, reputation
  - E.g., Google's Play Store, Apple's AppStore
  - Can catch different kinds of problems (e.g., program analysis)
  - Can make it costly to distribute malicious apps

Segment 6: Summary

- What worked well in Android's design?
- What didn't?

Sensible security design for Android

- Good example of design w/ security in mind
- Better security than desktop applications
- Better security than web apps
- Centralized app verification / audit / reputation

Some open issues

- Permission questions don't resonate w/ users
- Kernel bugs lead to compromises
- Mistakes in app/server logic
  - Should the next mobile platform encompass the server too?
- Smaller but important details
  - Intent secrecy/integrity; naming; declarative policy vs code; …

THANK YOU

Nickolai Zeldovich

Associate Professor