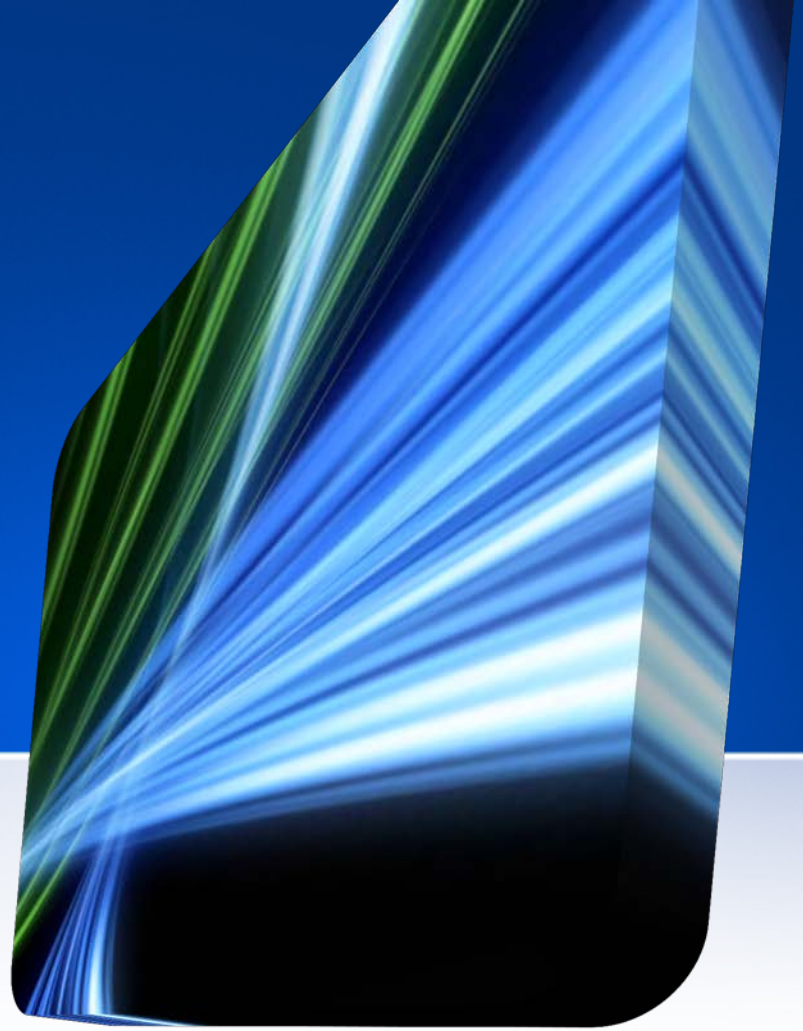
Android Security



Giovanni Russello

g.russello@auckland.ac.nz

What will you learn?

- The Smartphone phenomena
- Overview of the Android Middleware
- Android Security Model
- Security issues and some approaches

What is a Smartphone

- Handset with full-fledged computing capabilities
- Several vendors with different OS
- Support for third-party applications
- Extended sensing capability

Major Marker Players

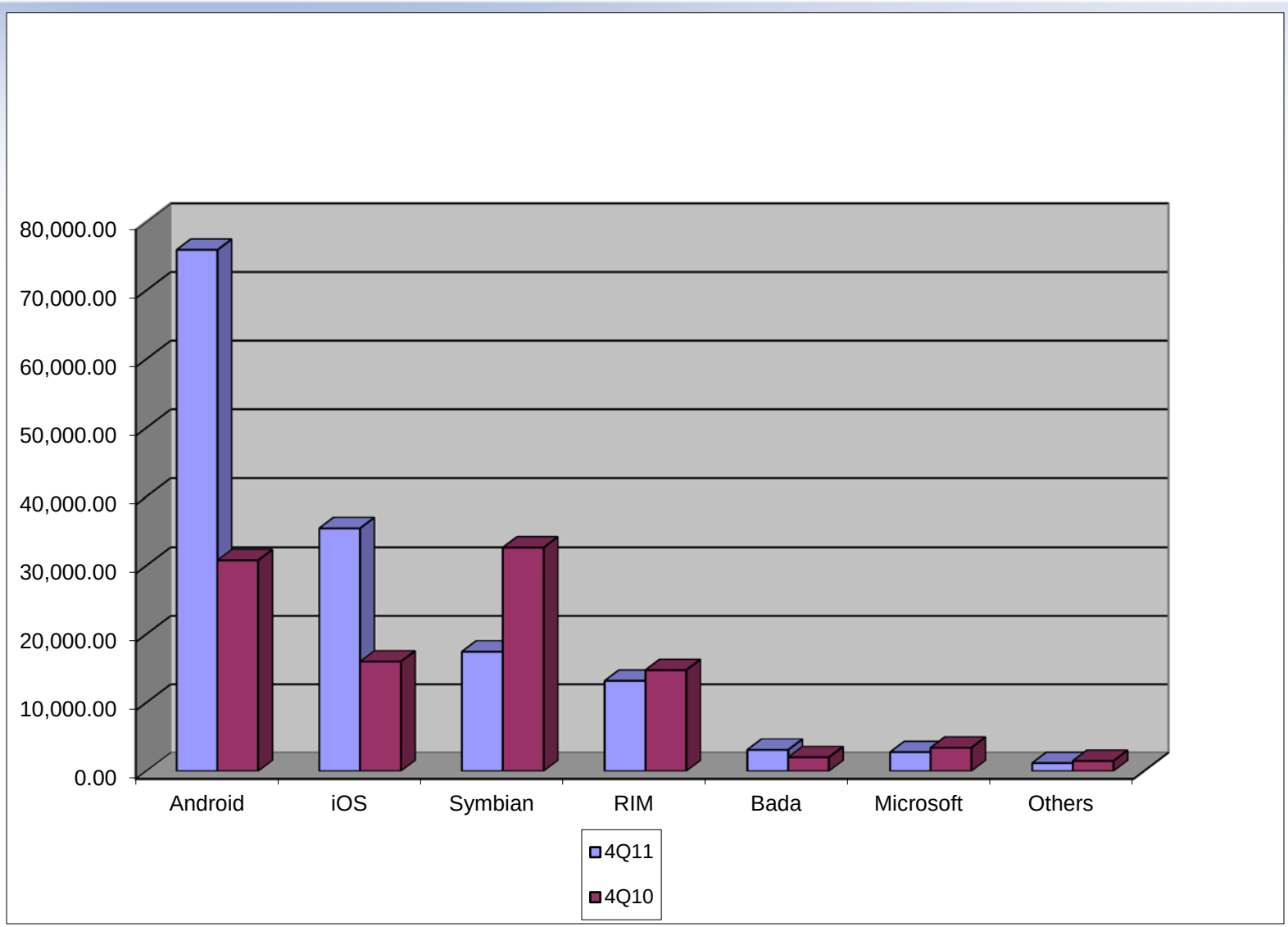
- Android – Google
- Symbian – Nokia
- Research In Motion (RIM) – BlackBerry
- iOS – Apple
- Windows Phone 7 – Microsoft

The Smartphomania

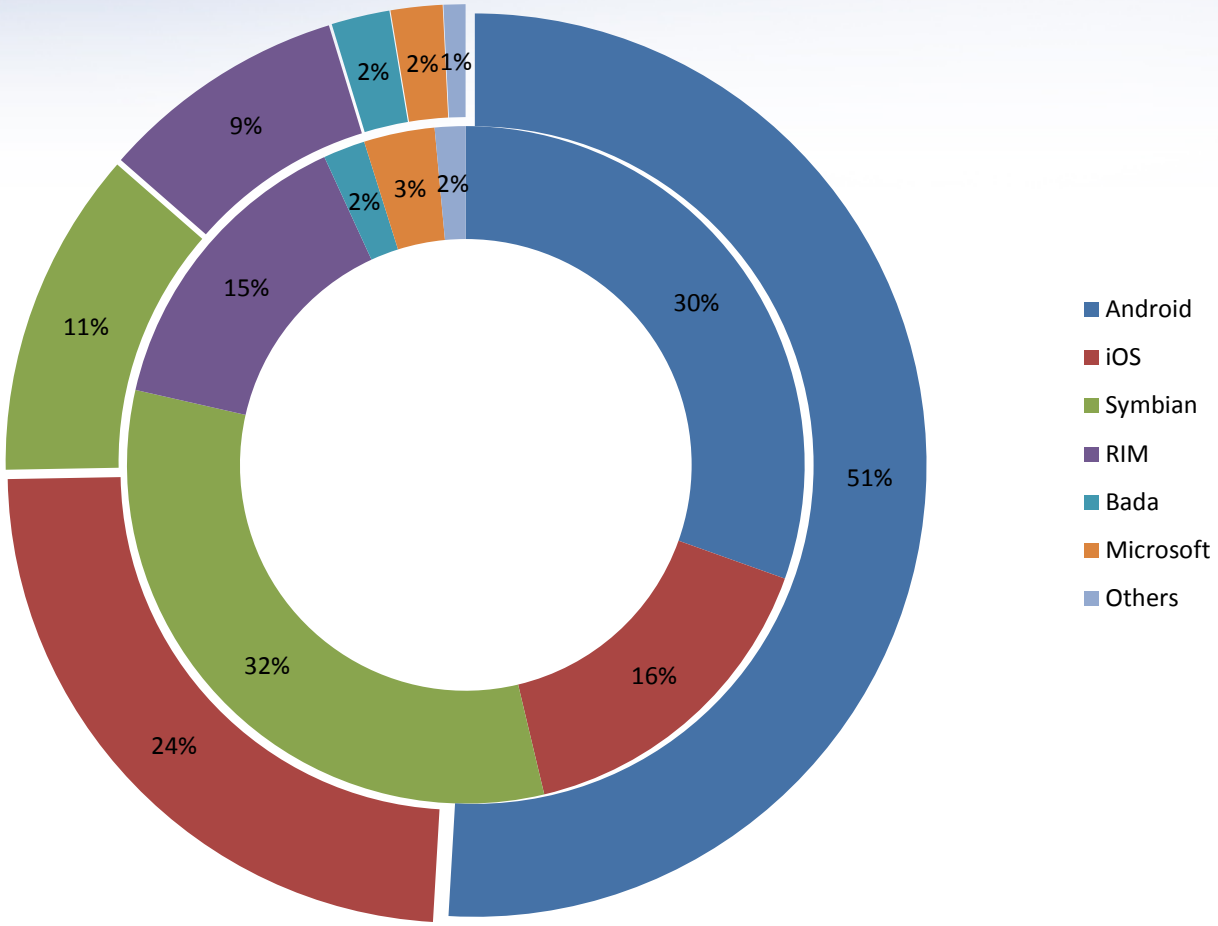
- Total Smartphone sales 2011: 472 million units
- Only in 2011 (4q): 149 million units
- Increase from the same period in 2010: 47.3%
- Of these devices, 76 million units are Android phones!

Source Gartner
http://www.gartner.com/it/page.jsp?id=1924314

Worldwide Smartphone Sales
4Q10 vs 4Q11 (Thousand of units)



Worldwide Smartphone Sales
4Q10 vs 4Q11



What is on stake?

Smartphones have been target of attacks

In the first half of 2011, malware contaminated app grew from 80 to 400 (Android Marketplace).

In terms of mobile users, this means that between a half million and a million users were exposed to malware only in the first half of 2011

Update Attack: clean apps that as grow in popularity are updated with malware

Security Threats

- Privacy violations
  - Unauthorised access to location, email, contacts …
- Money loss
  - Unauthorised sending of SMS
  - Banking Trojan (SpyEye, ZeuS)

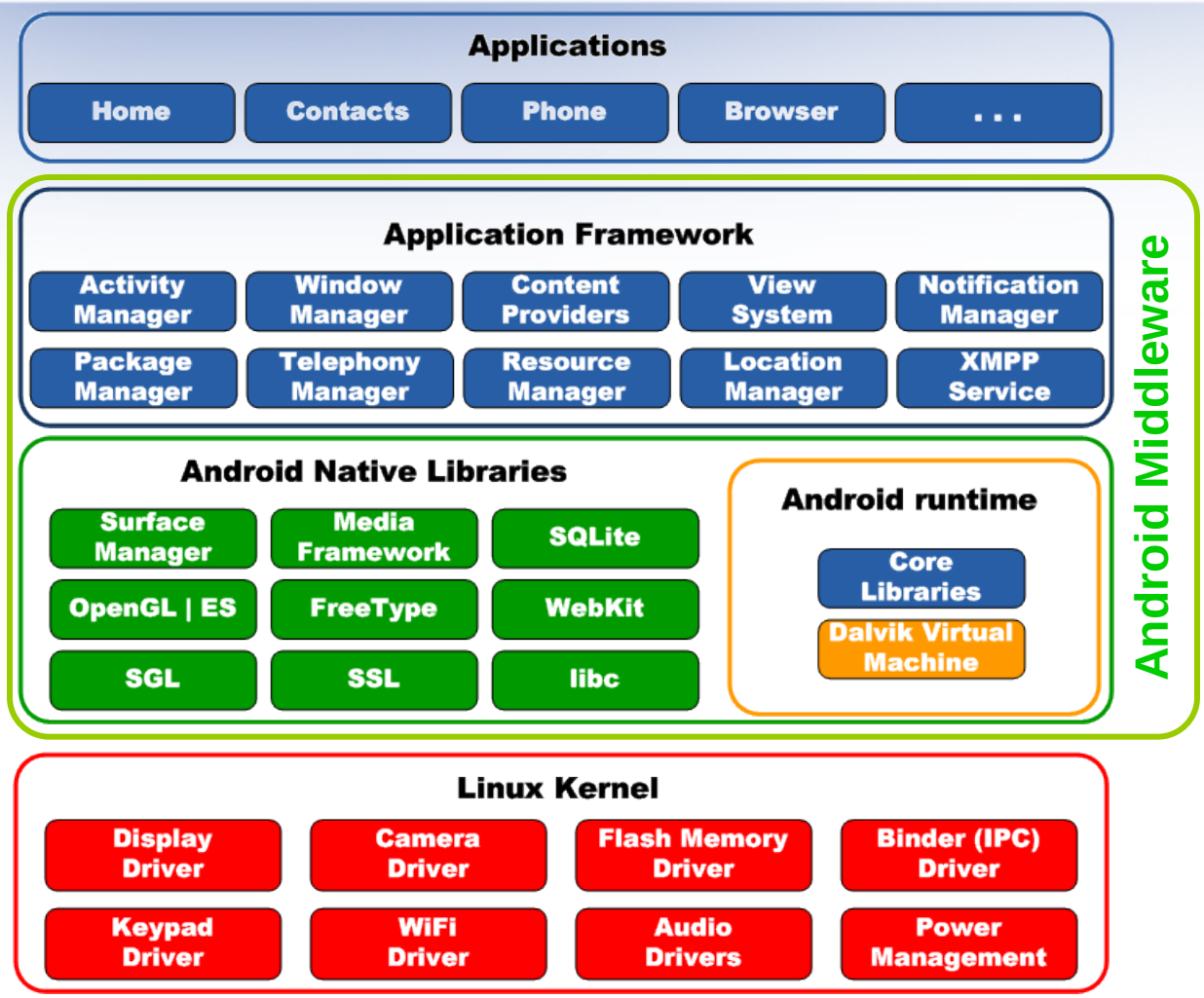
What is it done?

- iPhone
  - Closed source, code signing and inspection
- BlackBerry
  - Closed source, code signing and certification
- Android
  - Open source, code signing, code inspection, and permission framework
- MeeGo
  - Open source, RBAC security framework
- Windows 7 Phone
  - Closed source, code signing and inspection

Google Android

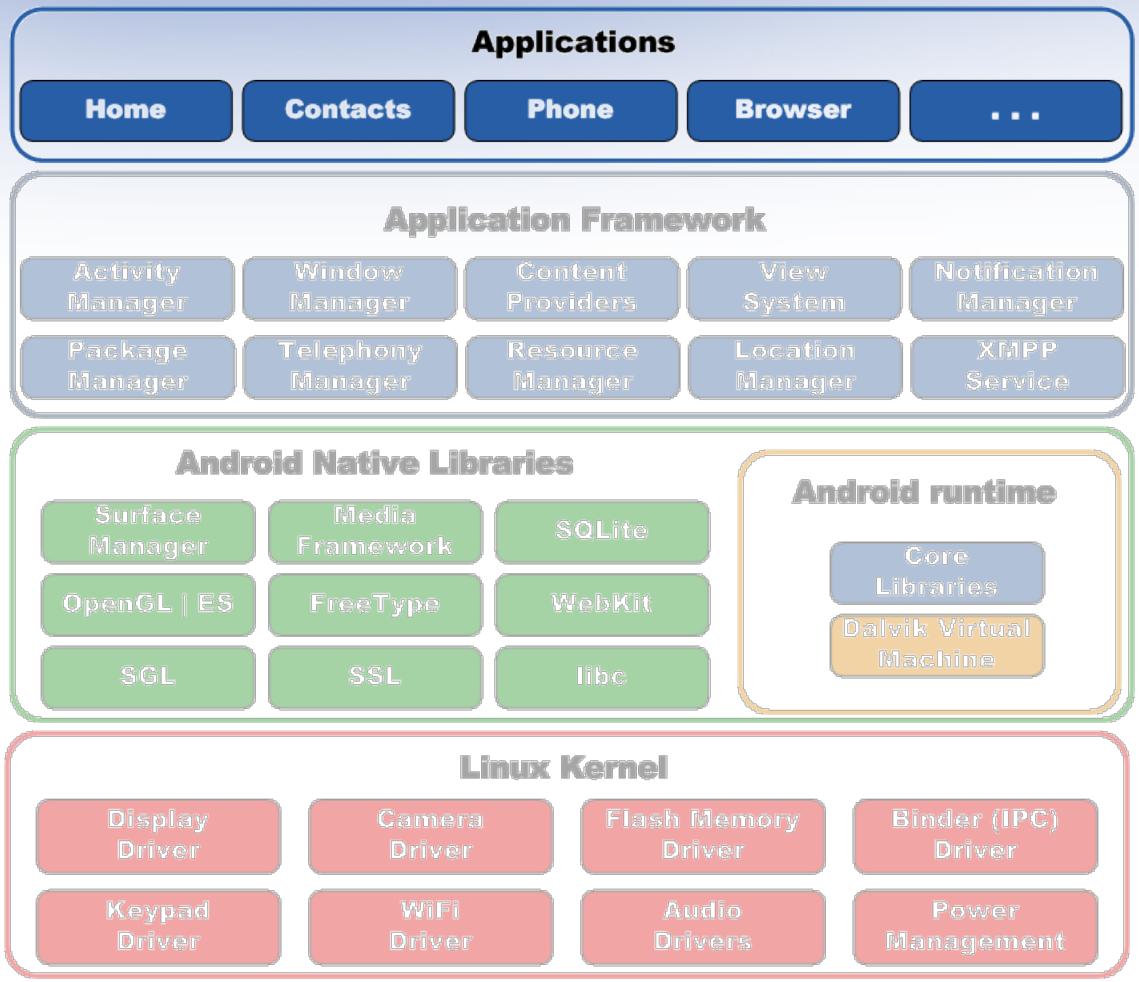
- First Android handset released in 2008
- Open source
- Strict Sandboxing
- Java Dalvik VM
- Java Apps
- Lightweight code signing
- Permission Framework
- App Market (more 100K apps)

Android View



Android is a set of programs for mobile devices that includes operating system, middleware and core applications

Applications



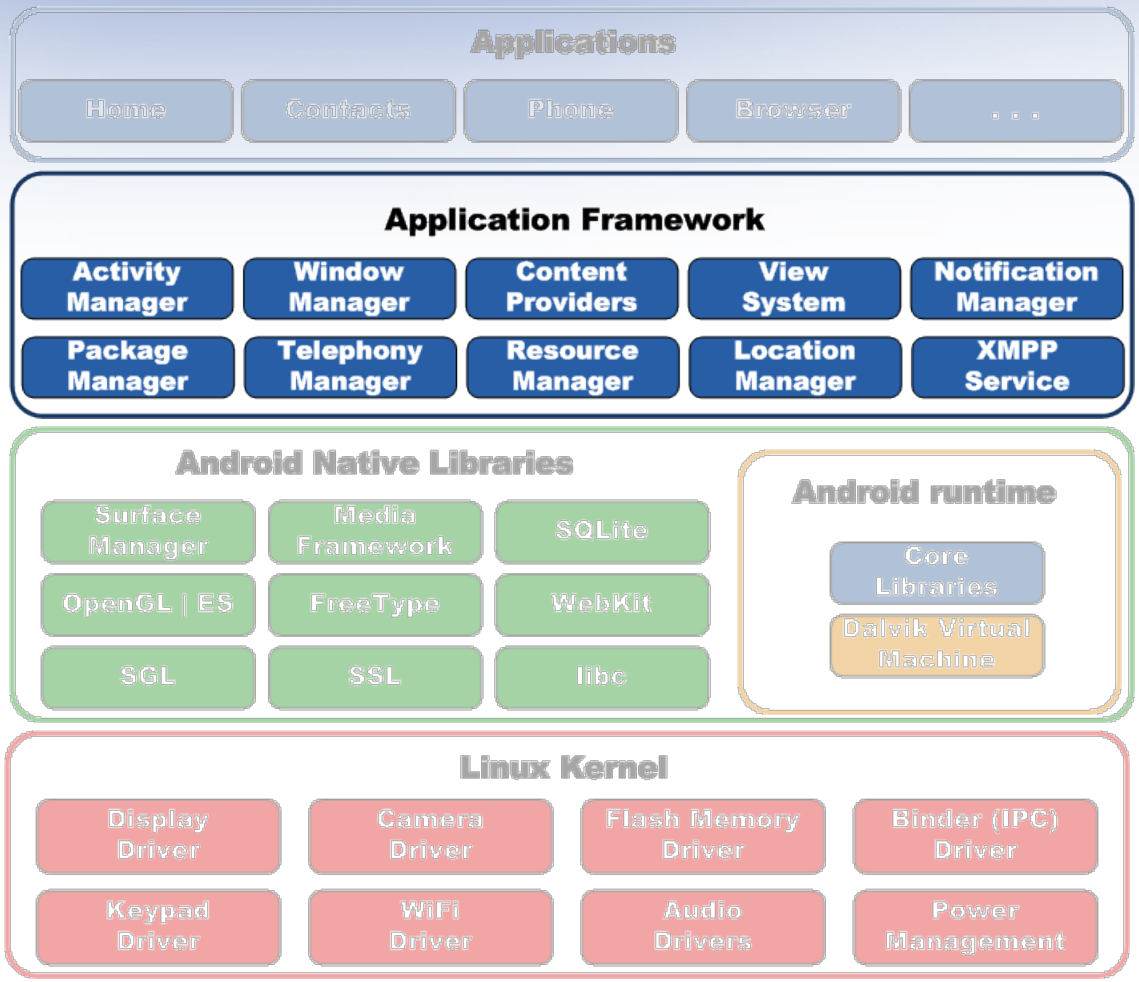
Core platform:

- Phone, Browser, Email…

Third-party:

- Applications that are produced by third-party developers

Application Framework



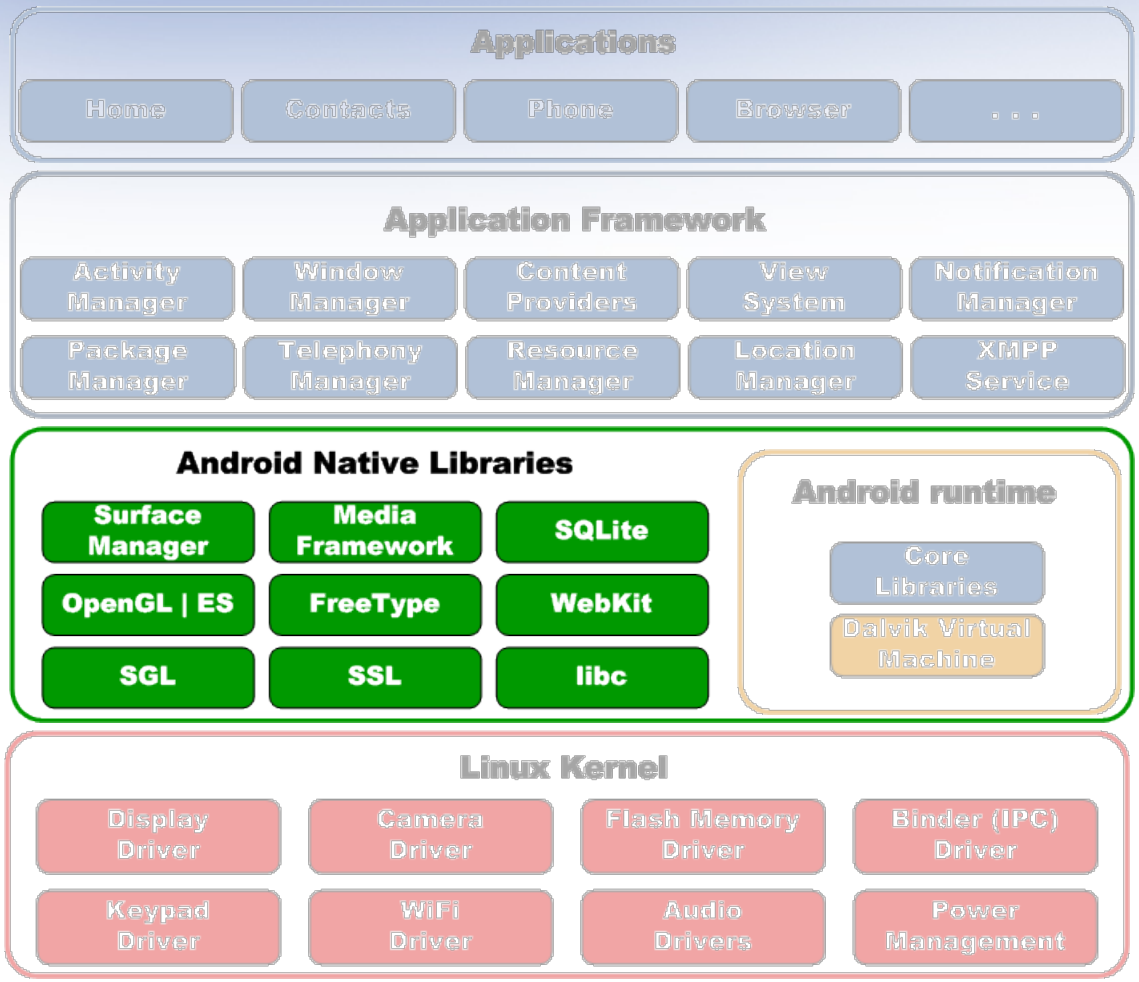
Core platform services:

- Activity, Package, Window and Content Providers

Hardware services:

- Telephony, Location, Bluetooth, WiFi, USB, and Sensor Services

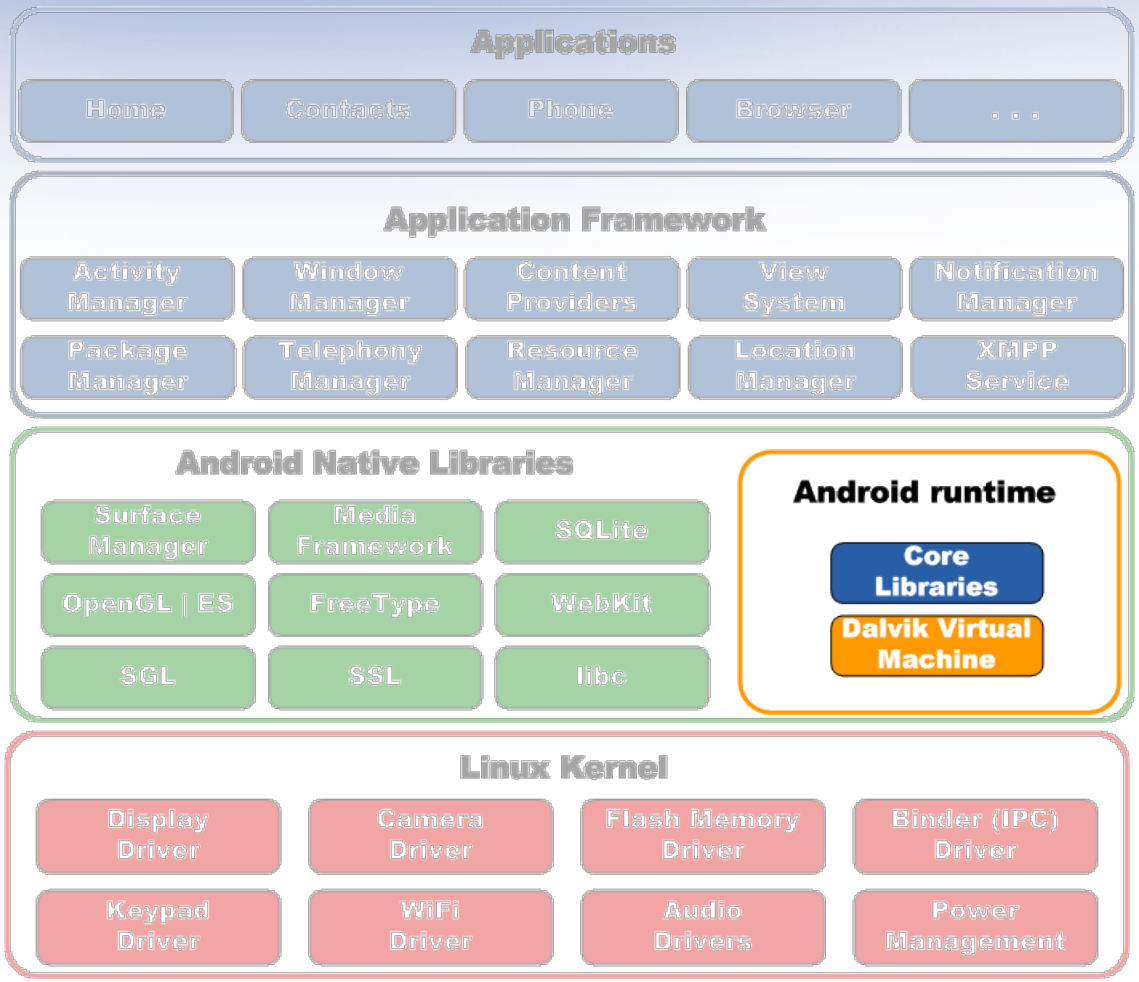
Android Native Libraries



Used for:

- Window management
- 2D and 3D graphics
- Media codecs
- Font rendering
- SSL
- The core of datastorage
- The core of web browser
- Bionic libc

Android Runtime



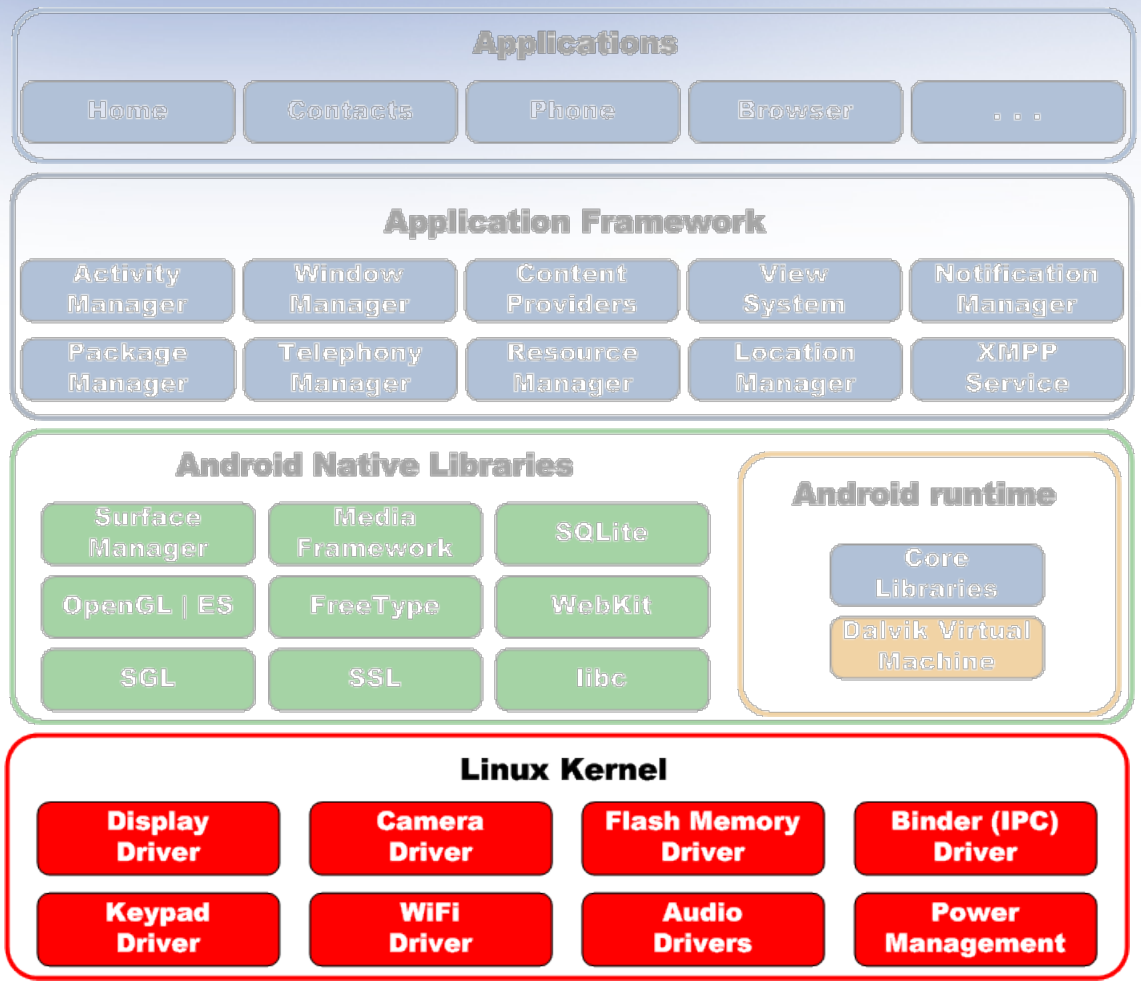
Core Libraries:

- Data structures, Utilities, File access, Network access, and Graphics

Dalvik VM:

- Provides application portability
- Supports multiple instances
- CPU and memory optimized to run on mobile devices

Linux Kernel



Linux features:

- Hardware abstraction layer
- Memory management
- Process management
- Security module
- Networking

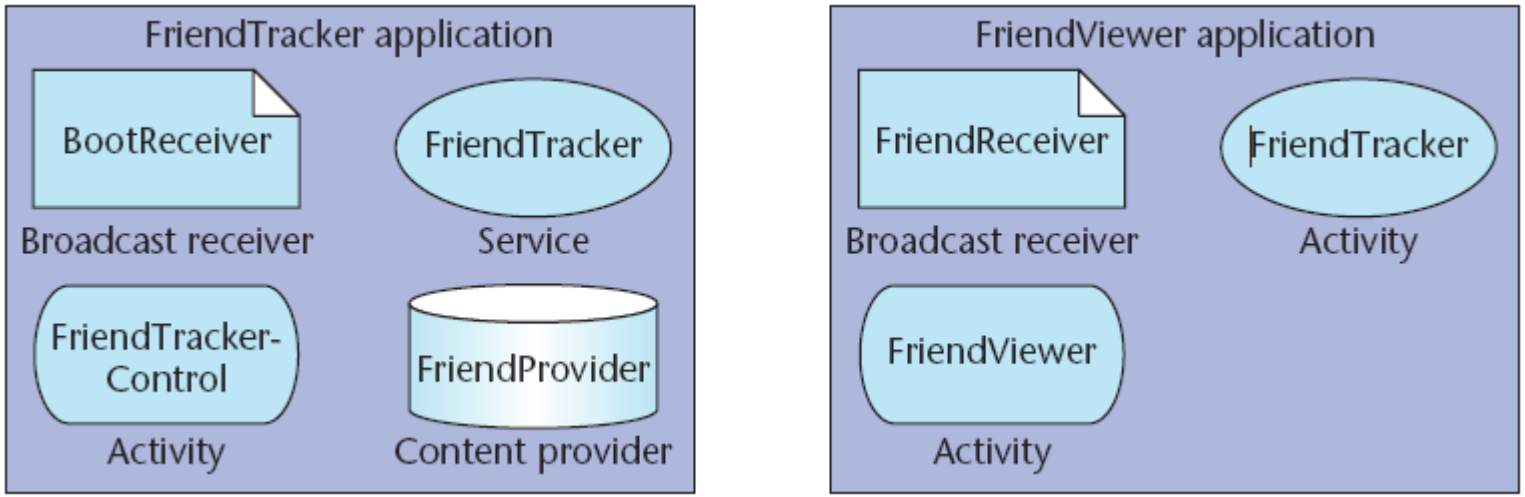
Android enhancements:

- Power management
- Binder IPC
- Logger

Android App Model

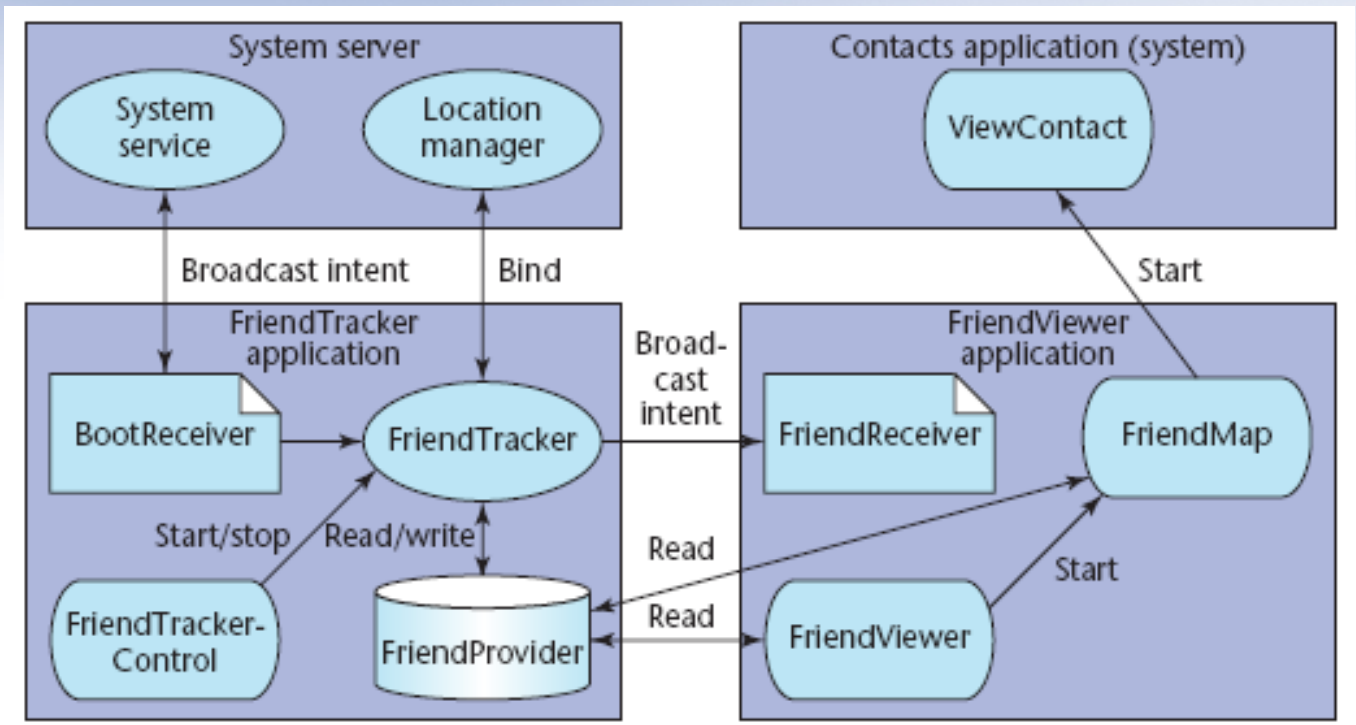
- Each application runs within an instance of a Dalvik VM (DVM)
- Each DVM is mapped in the Linux Kernel with a unique user id
- Android supports Inter-process communication (IPC)
- A reference monitor mediates IPC calls

Android App Model



- Applications are formed of components
  - Activities
  - Services
  - Content Providers
  - Broadcast Receivers

Inter-Component Communications



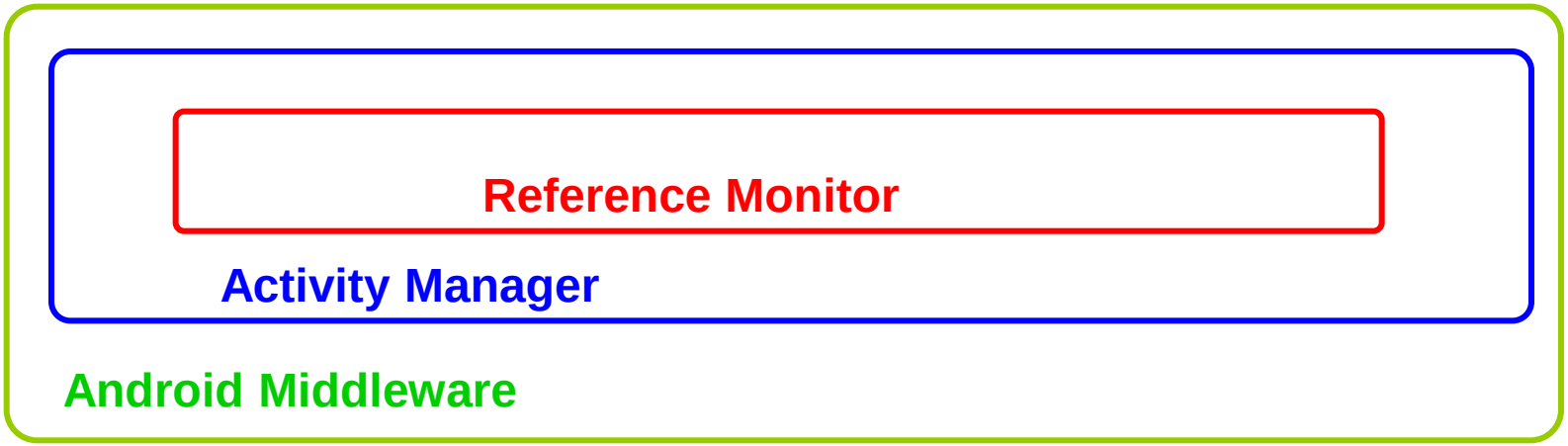
Each Component exposes a specific API for communications

- Services expose Start, Stop, Bind as actions that other application can invoke through Intents

Enforcement

- Each App comes with a Manifest file (AndroidManifest.xml)
- Uses Permission: the permission that an application requires. This must be granted by the user at installation time.
- Permission: definition of permissions to protect part of this application
- **All-or-nothing model!**

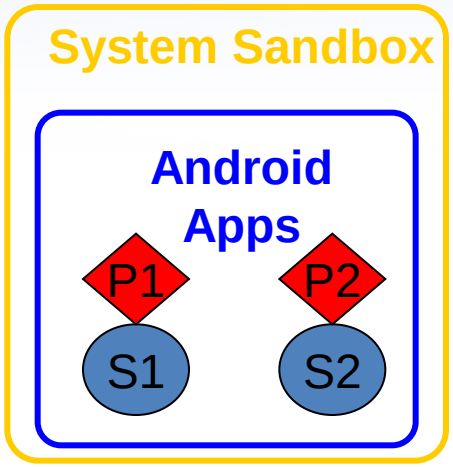
Android MAC Model

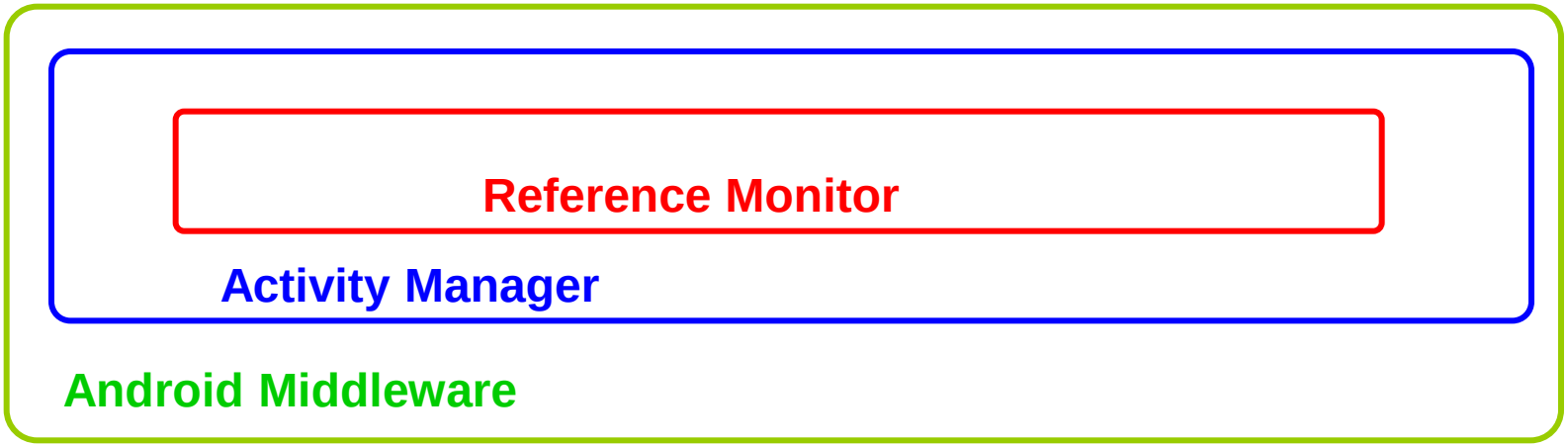


Protection Domain

S1 = Location Service

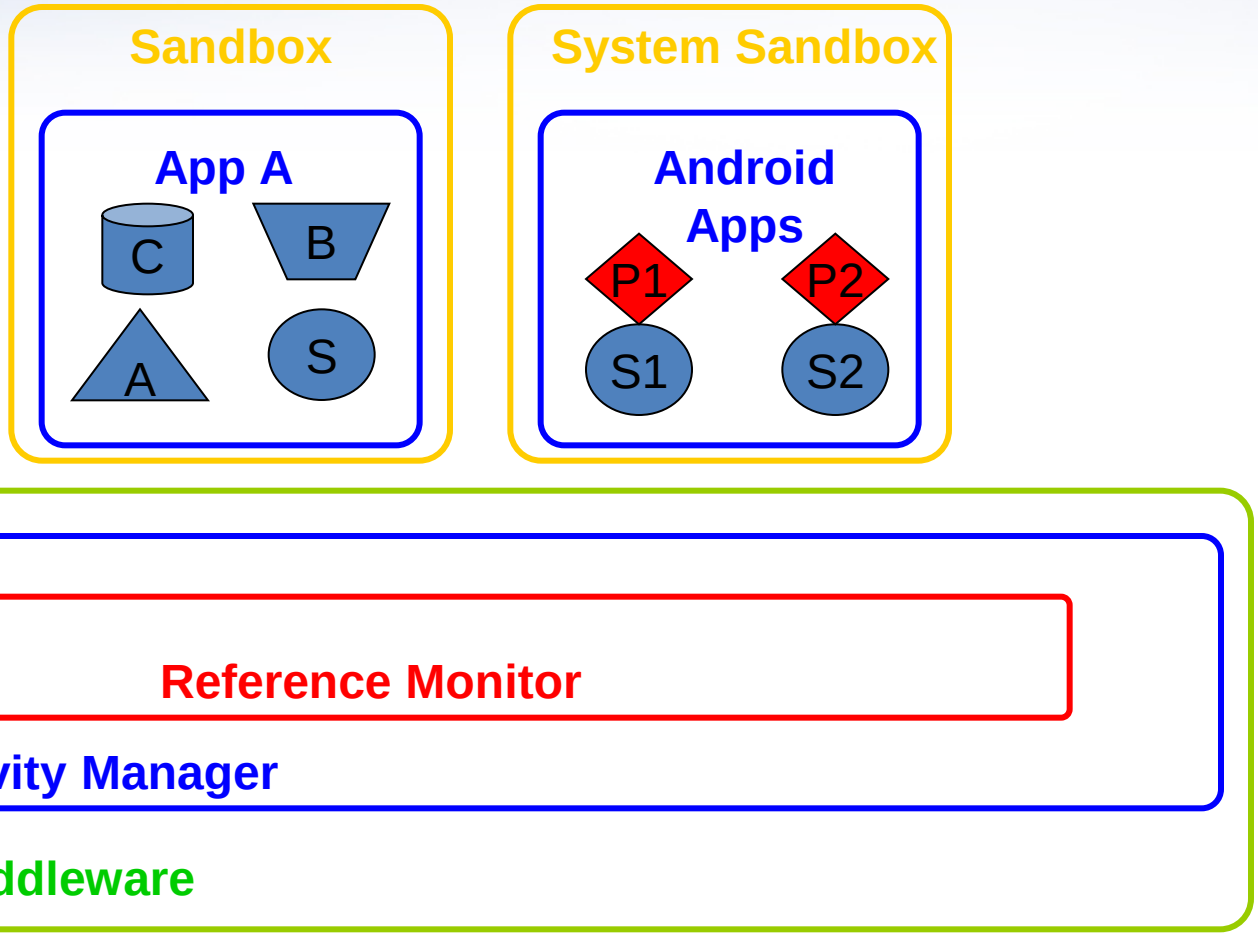
P1 = LOCATION_PERMISSION



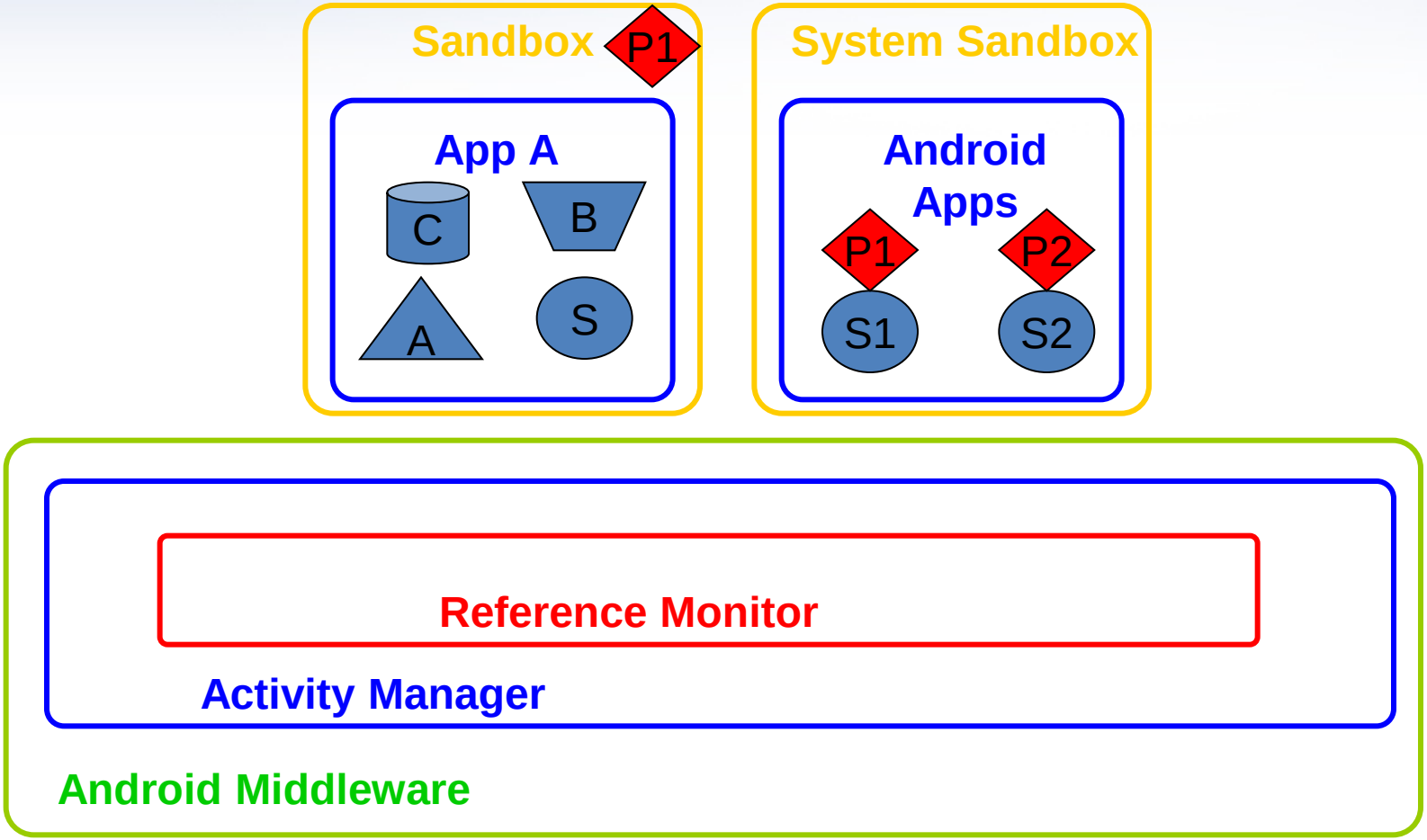


Assignment of Permissions

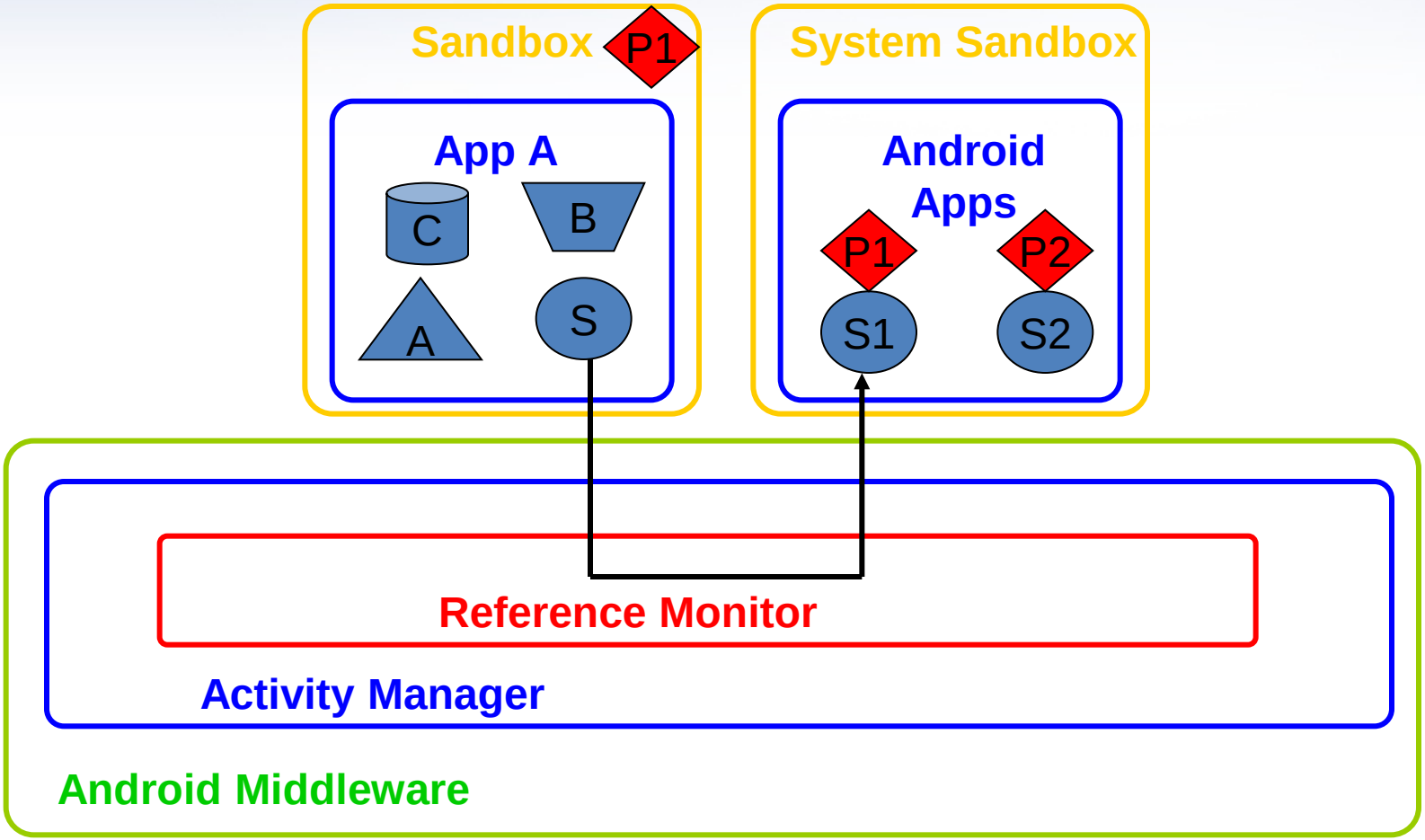
Install Time: Uses Permission = P1?



Using the Permission



Reference Monitor



Mandatory Access Control

- Once the labels are assigned neither the application nor the user can change them
- Applications cannot delegate their permissions
- *BUT* components can expose interfaces that other applications can invoke
- This makes difficult in standard Android to control information flow (can lead to severe attacks)

Permission Protection Level

- “Normal Permissions” are assigned by default to apps
- “Dangerous Permissions” require user confirmation
- “Signature Permissions” are granted to apps signed by the same developer
- “System or Signature Permissions” are granted only to special apps installed in the data/system folder (i.e., apps signed by Google)

Security Refinements

Android Security Model allows developers to refine the security domain of their applications

- Through the standard mechanism using the Manifest
- Programmatically by using special parameters in the API

Bad move!!! Make everything murky and worst of all by default access is granted!!

Public vs Private Components

- By default any components that is not assigned a permission is public
- Developers can declare a component private by setting the `exported` flag to false in the manifest file
- Private components can only be accessed by other components in the same app
- Android can also infer if a component is private by other declarations in the manifest file (Do you trust it??)

Implicitly Open Components

- Public components have all their interface accessible to any other components
- Developers must explicitly assign permission labels to protect those interfaces

Broadcast Intent Protection

- When an intent is broadcasted, all installed apps are able to listen to those events
- This mechanism can be exploited by malicious apps that are listening for a certain event to happen
- It is possible to protect the intent programmatically:

```
sendBroadcast(intent, perm.MyPerm)
```

This means that the Manifest does not provide a complete view of app security

Service Hooks

- Android does not support a fine-grained mechanism to protect the interface of a Service
- Once a component has the permission label to access a service, the component can start, stop, bind the service
- Again programmatically it is possible to refine this mechanism by doing some extra checking at the code level, putting security policies in the app code
- Not a good security and software eng. practice!

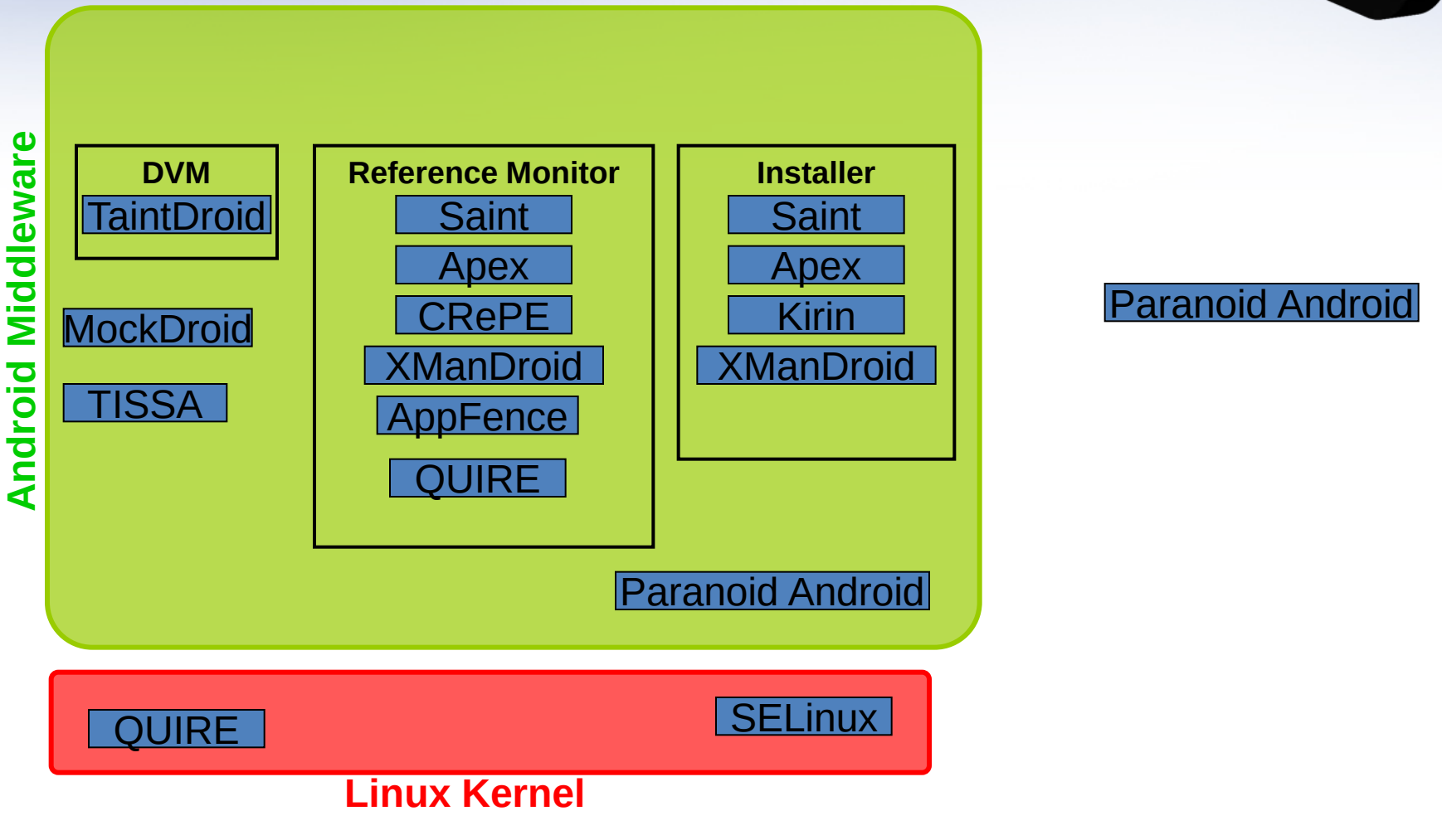
Delegation

- Pending Intents that delegate to another app the parameters and time when an action is executed
  - Location service notifies registered apps when location changes
- URI delegation where an app delegates a component to perform an action on a resource
  - The app provides a capability to the component for performing the action
- Per se, there is nothing wrong with delegation. However, it deviates from the main MAC model

Flexibility is not always good

- The Android security model is very flexible
- However, it starts from the simple MAC model and becomes very messy
  - Source code options
  - Open default policy
  - Delegation
  - No control for information flow

Security Extensions for Android (as for June 2011)



Fine-grained Security Policy

- Saint (ACSAC ‘09)
  - Allows app developers to protect their applications from being misused
- APEX (ASIACCS ‘10)
  - Circumvent the All-or-Nothing approach of Android permission granting
- Porscha (ACSAC ‘10)
  - Support for DRM-like policies for phone data
- CRePE (ISC ’10)
  - Enforcement of context-related policies

Data Filtering and Tainting

- MockDroid (HotMobile ‘11)
  - Limiting the access to the data
- TISSA (Trust ‘11)
  - Substituting the reply from content providers
- TaintDroid (OSDI ’10)
  - Labelling of data for preventing data leakage

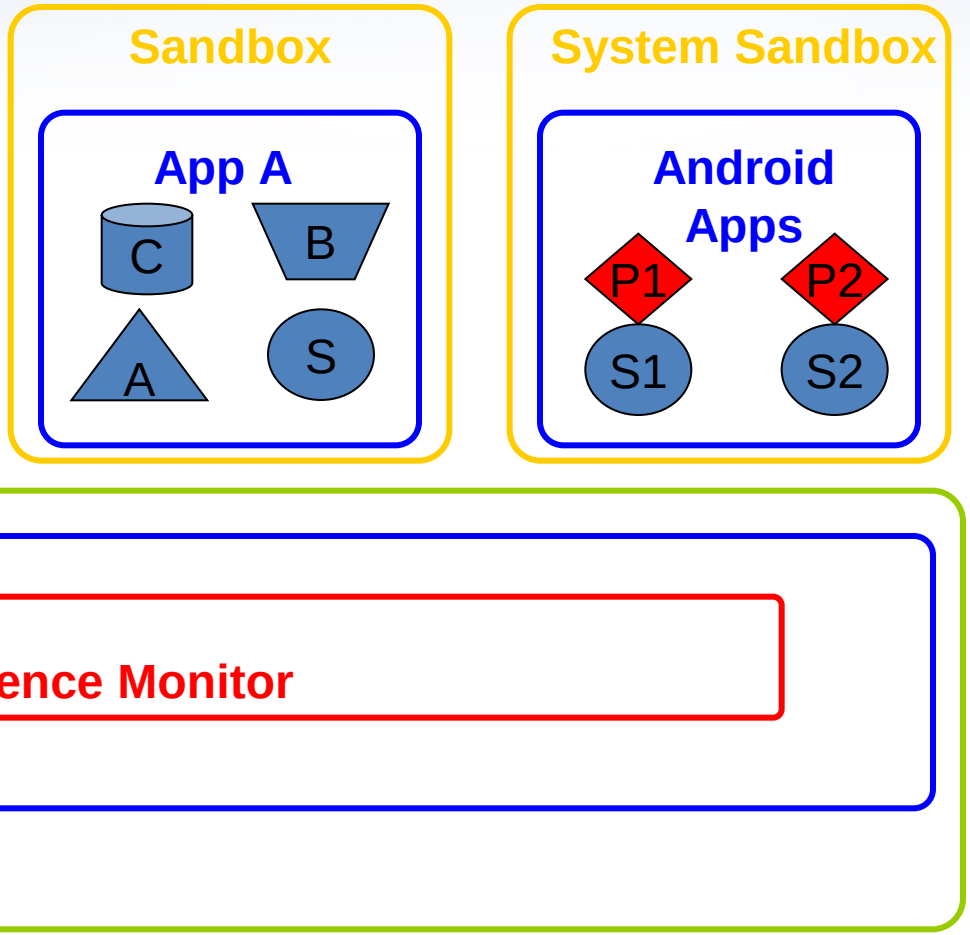
Privilege Escalation Attacks

"An adversary tries to escalate privileges to get unauthorised access to protected resources"

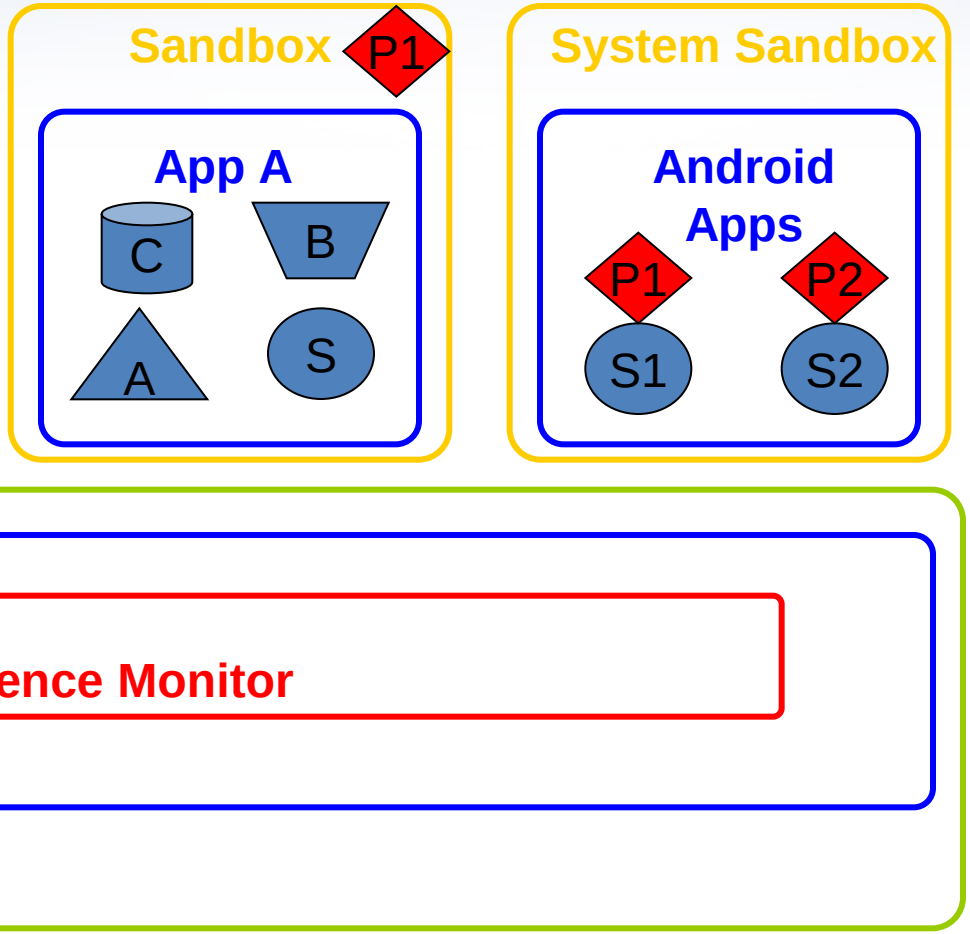
- *Confused deputy attack*: leverage the vulnerability of a benign application
- *Colluding attacks*: more applications collaborate to get an objectionable set of permissions

Privilege Escalation Attacks

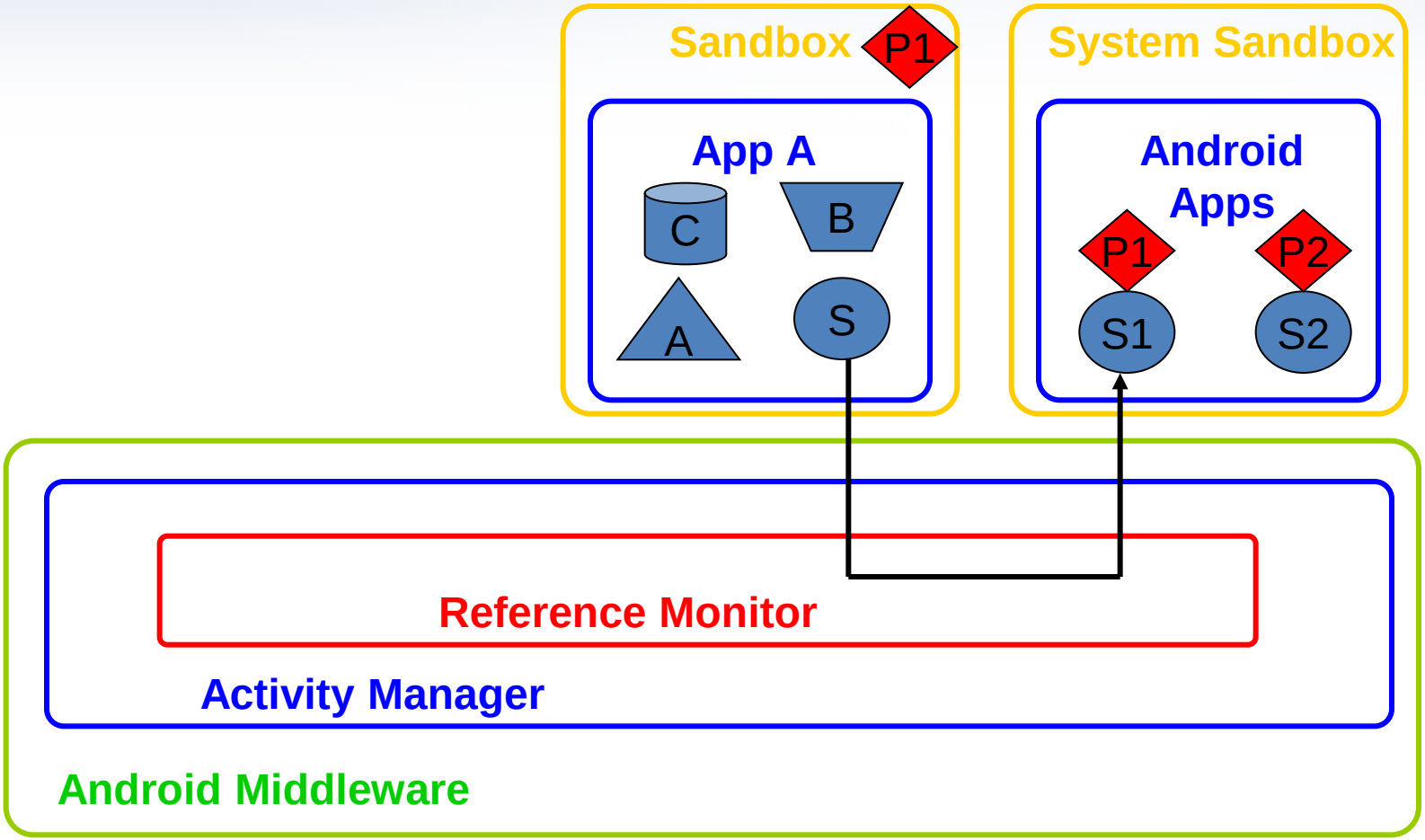
Install Time: Uses Permission = P1?



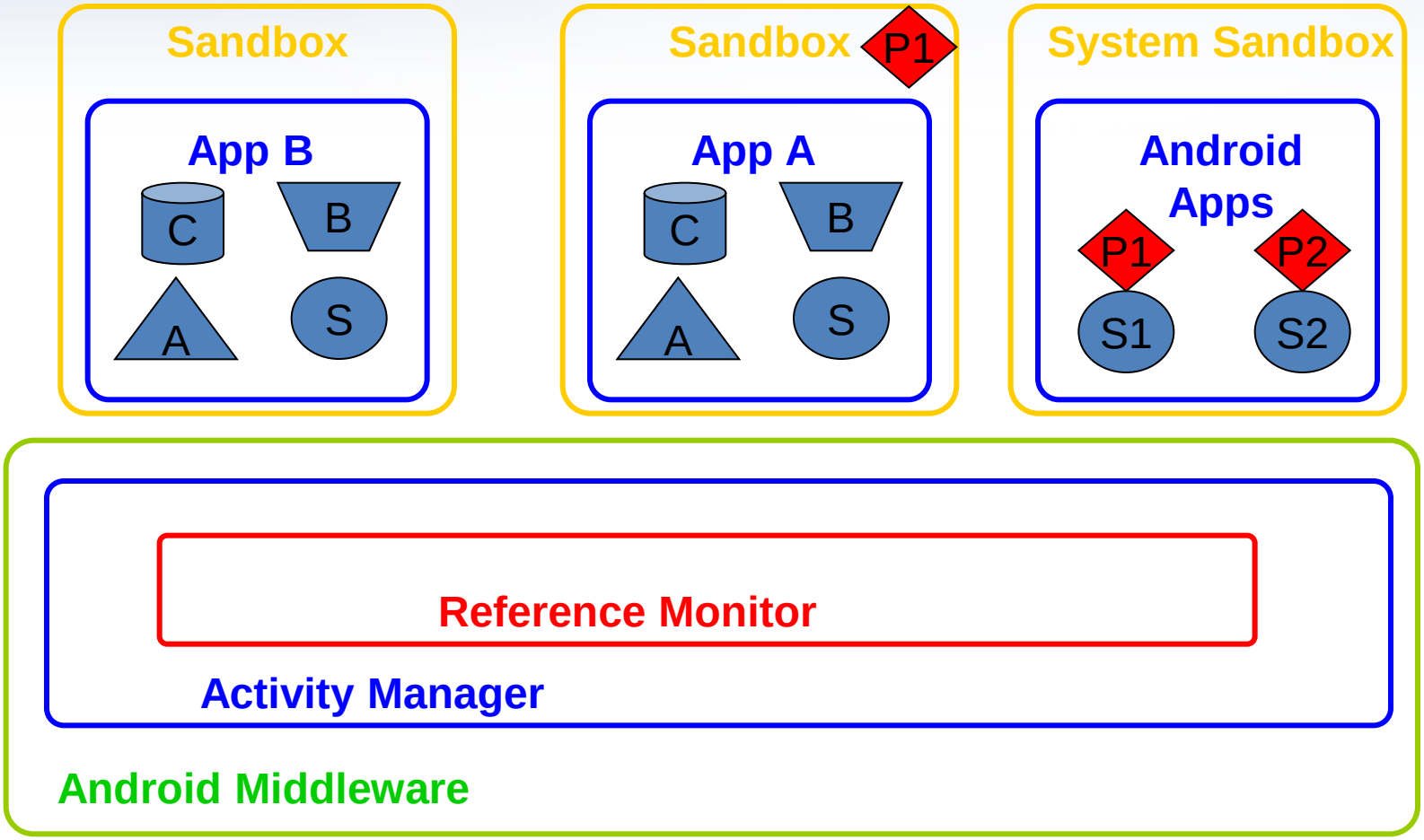
Privilege Escalation Attacks



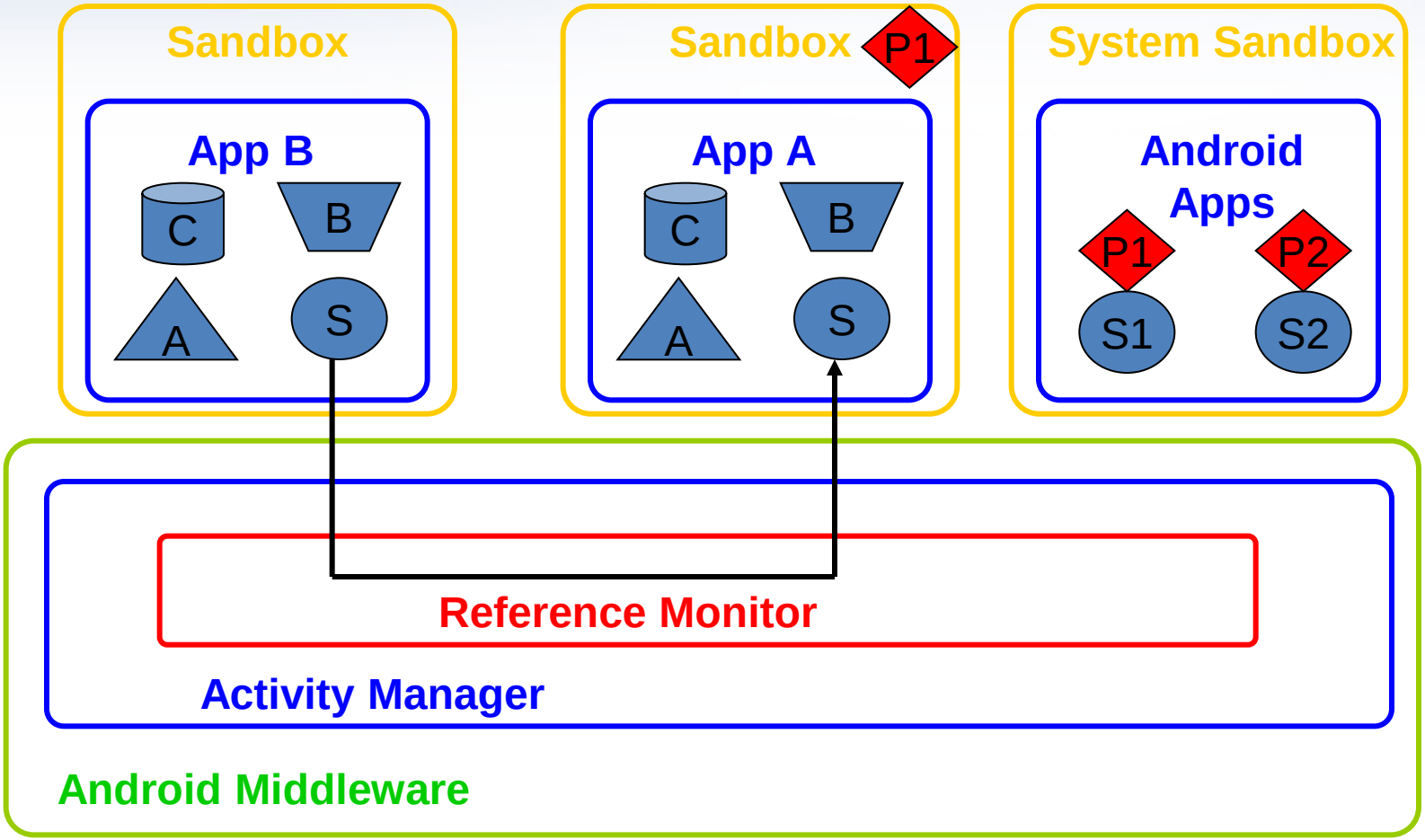
Privilege Escalation Attacks



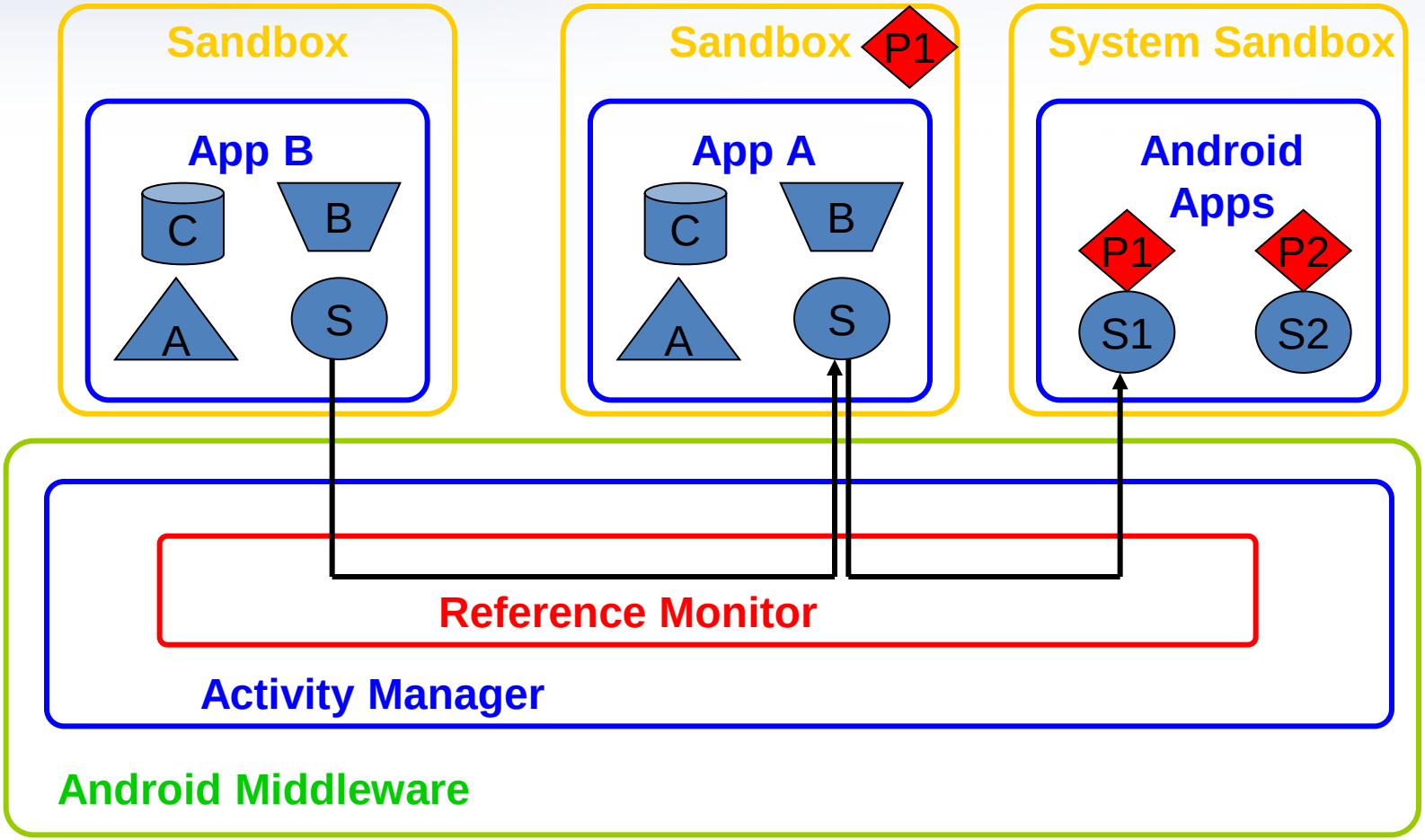
Privilege Escalation Attacks



Privilege Escalation Attacks



Privilege Escalation Attacks



Protection against Privilege Escalation

QUIRE (USENIX Security Symposium ‘11)

- Effective against confused deputy attacks
- Tracing of IPC chain to check if all apps have the right to access a resource

However

- It requires that apps have to use modified API
- It does not solve the problem of colluding apps

Protection against Privilege Escalation

AppFence (TR 11 Uni Washington and MS Research)

- Based on TaintDroid for taint capability
- It supports data shadowing and protects from data exfiltration

However

- Effective only against confused deputy attack

Protection against Privilege Escalation

XManDroid (TR 11)

- Real-time IPC monitoring
- System state of the app communications for potential spread of privileges

However

- No control outside the IPC channels (i.e. Internet access)

What is missing

- No modifications to Android API
- No trust on apps
- Control over IPC and system-level calls (internet)
- Data filtering capabilities
- Tuneable

That is why we came up with

…Yet Another Android Security Extension

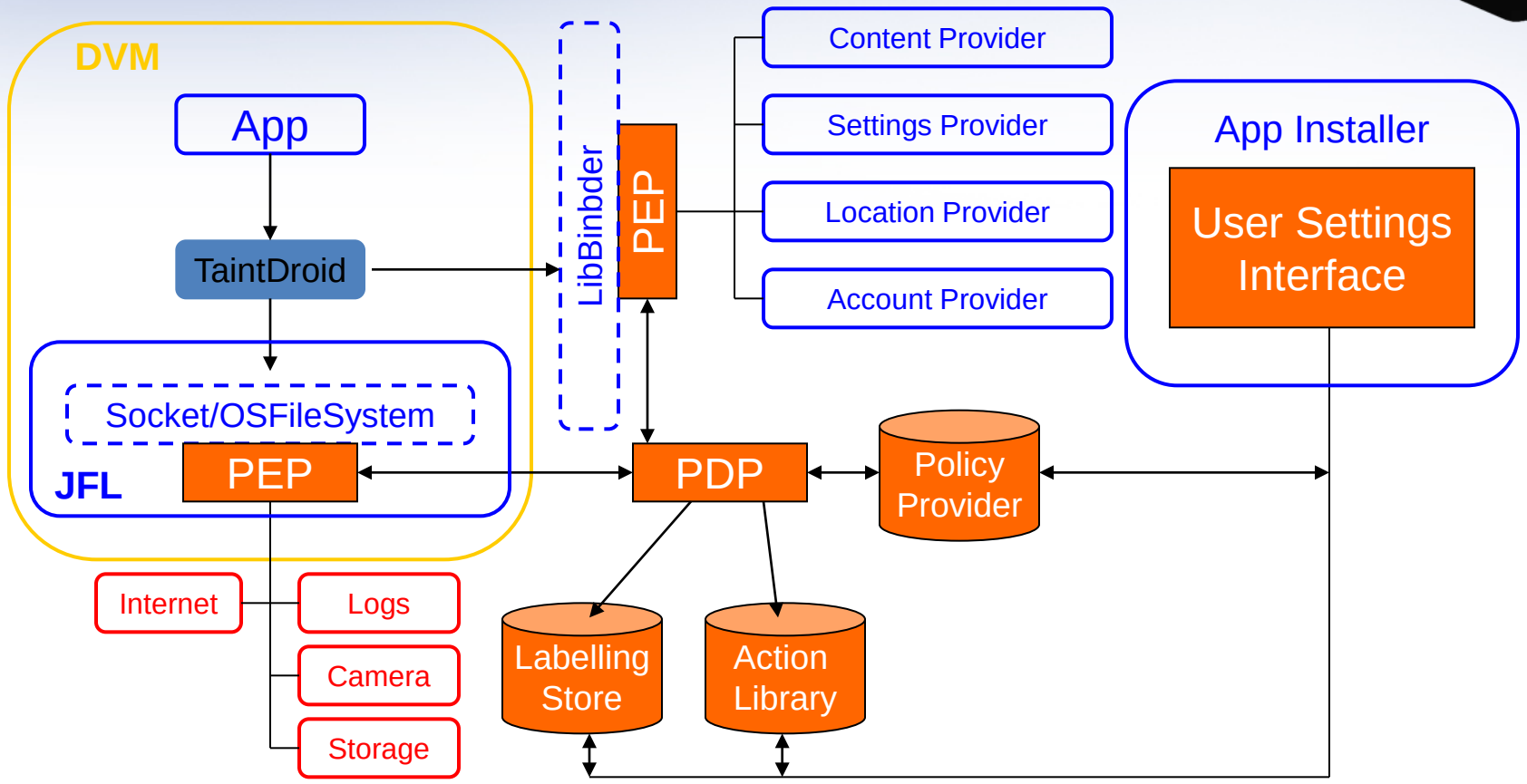
YAASE

YAASE Main Features

A Policy-based System for

- Controlling Information Flow
- Fine-grained Data Filtering

YAASE Architecture



Policy-based AC Terms

- A policy is a rule that governs the behaviour of a system
- PEP stands for Policy Enforcement Point
  - It is responsible for intercepting the requests and enforcing the access control decisions
- PDP stands for Policy Decision Point
  - It is responsible for evaluating policies and coming up with a decision
- Policy Provider is the repository where policies are stored

YAASE Policy Language

```
PolicyName:
  Requester can do operation on Resource
  [have to perform action]
  handle dataLabelExpression
```

YAASE Policy Language

```
PolicyName:
  Requester can do operation on Resource
  [have to perform action]
  handle dataLabelExpression
```

By default, if no policy is specified no action is granted!

Example of a Privilege Escalation

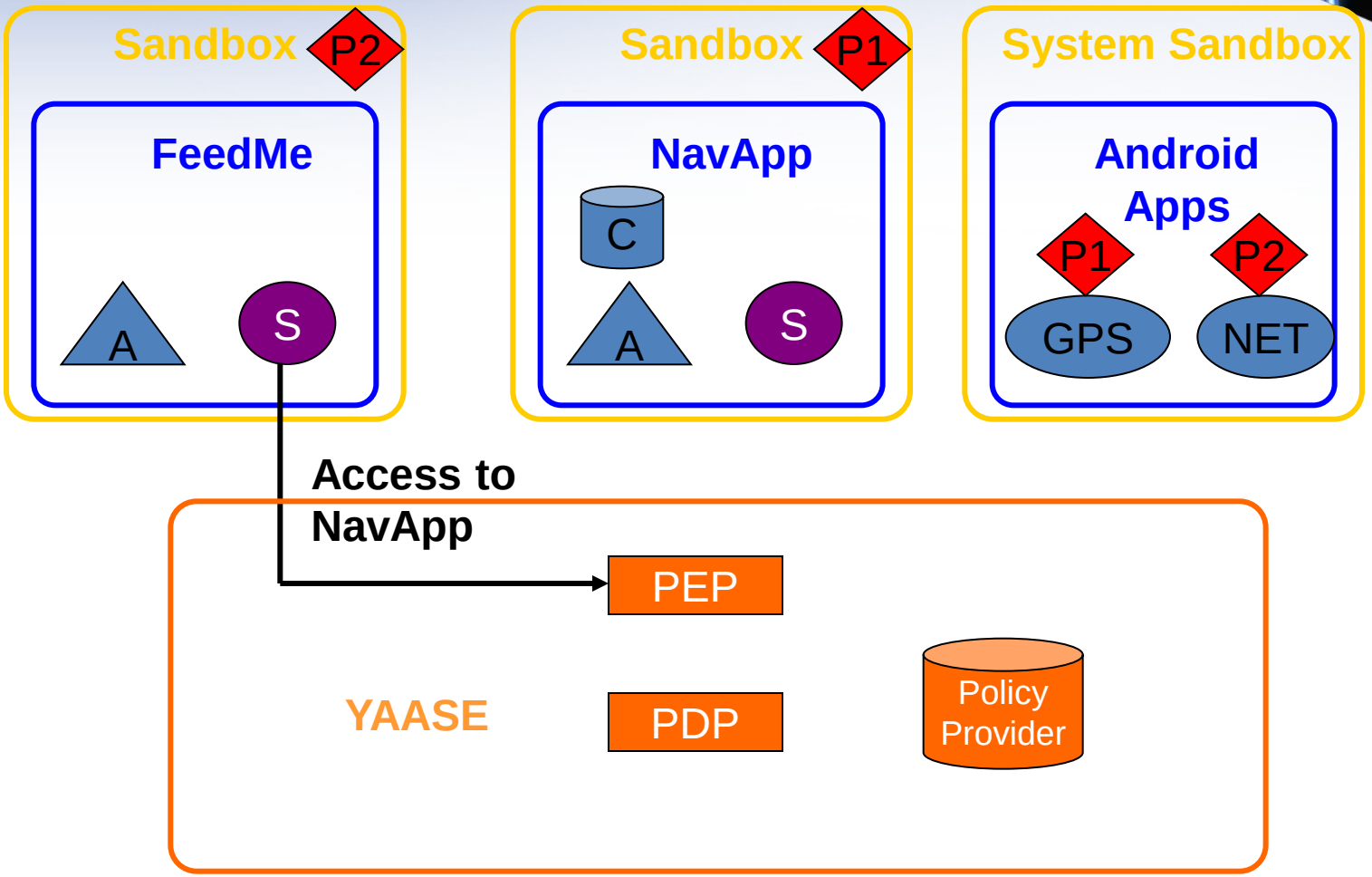
- FeedMe: A news feed app requiring access to internet
- NavApp: A navigation app requiring access to GPS

Policies for Apps

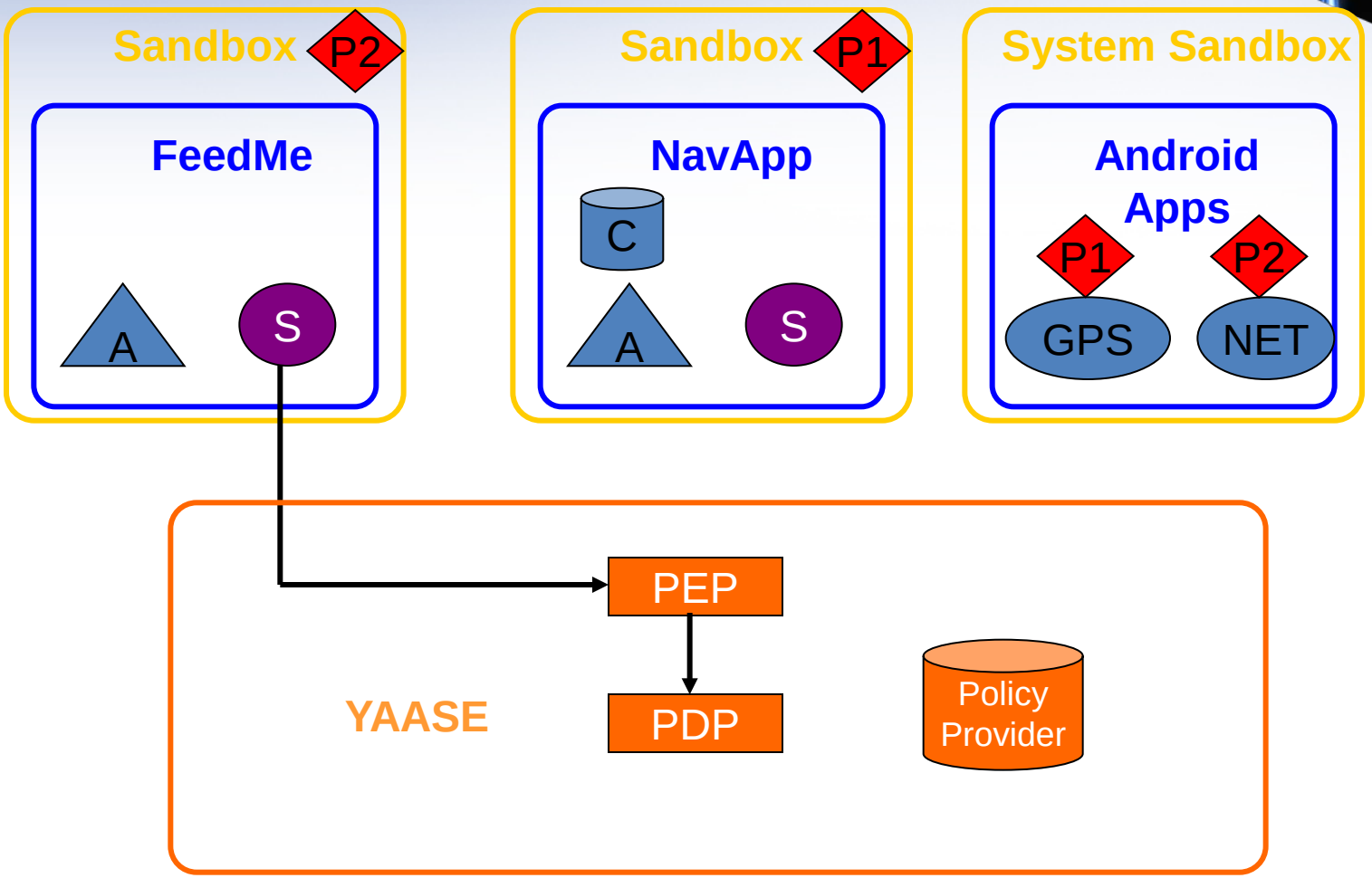
```
PolFeedMe:
  FeedME can do send on Internet
  handle “NoLabels”

PolNavApp:
  NavApp can do access on GPS
  handle “FineLocation”
```

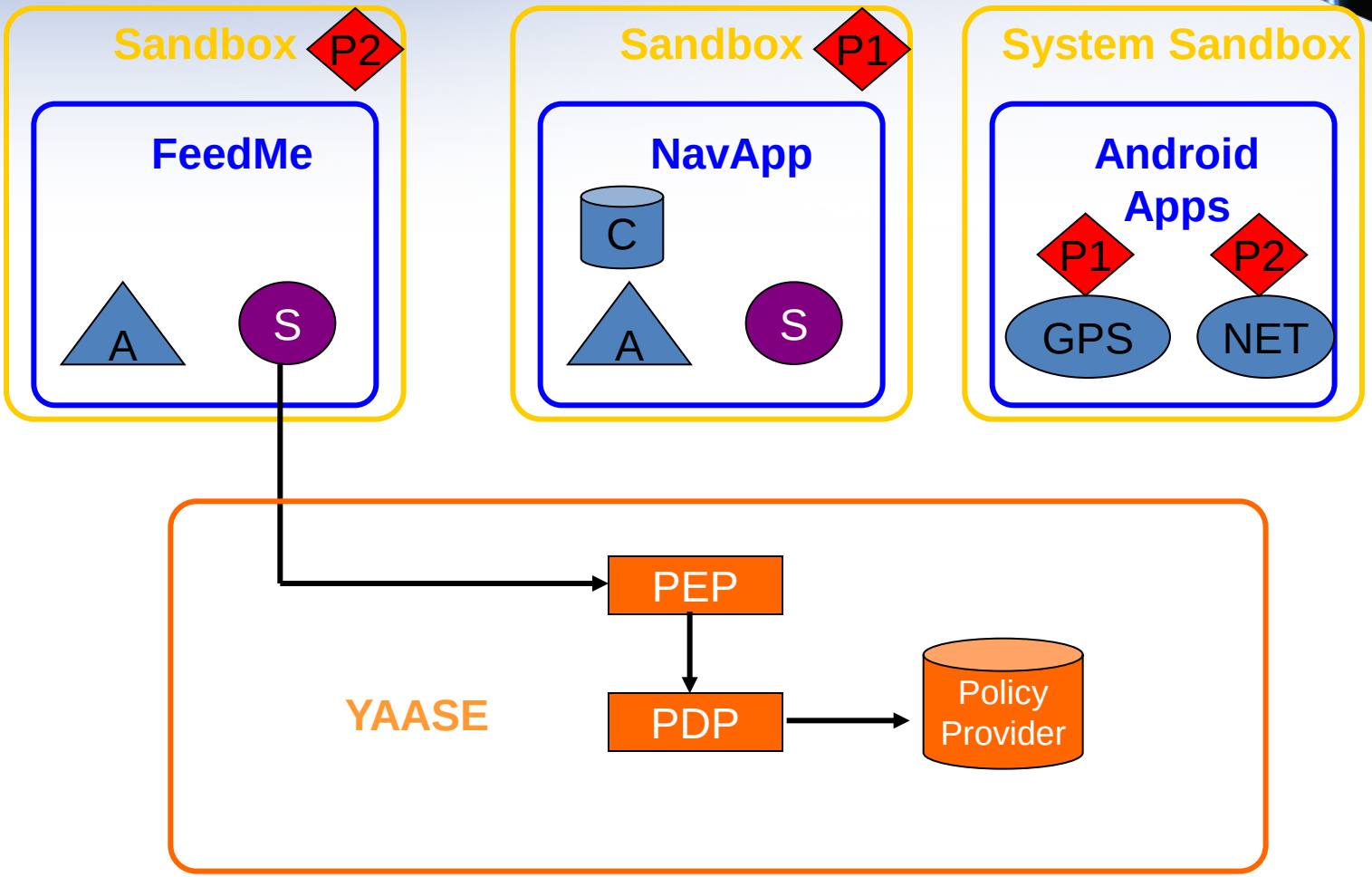
Restrict Approach



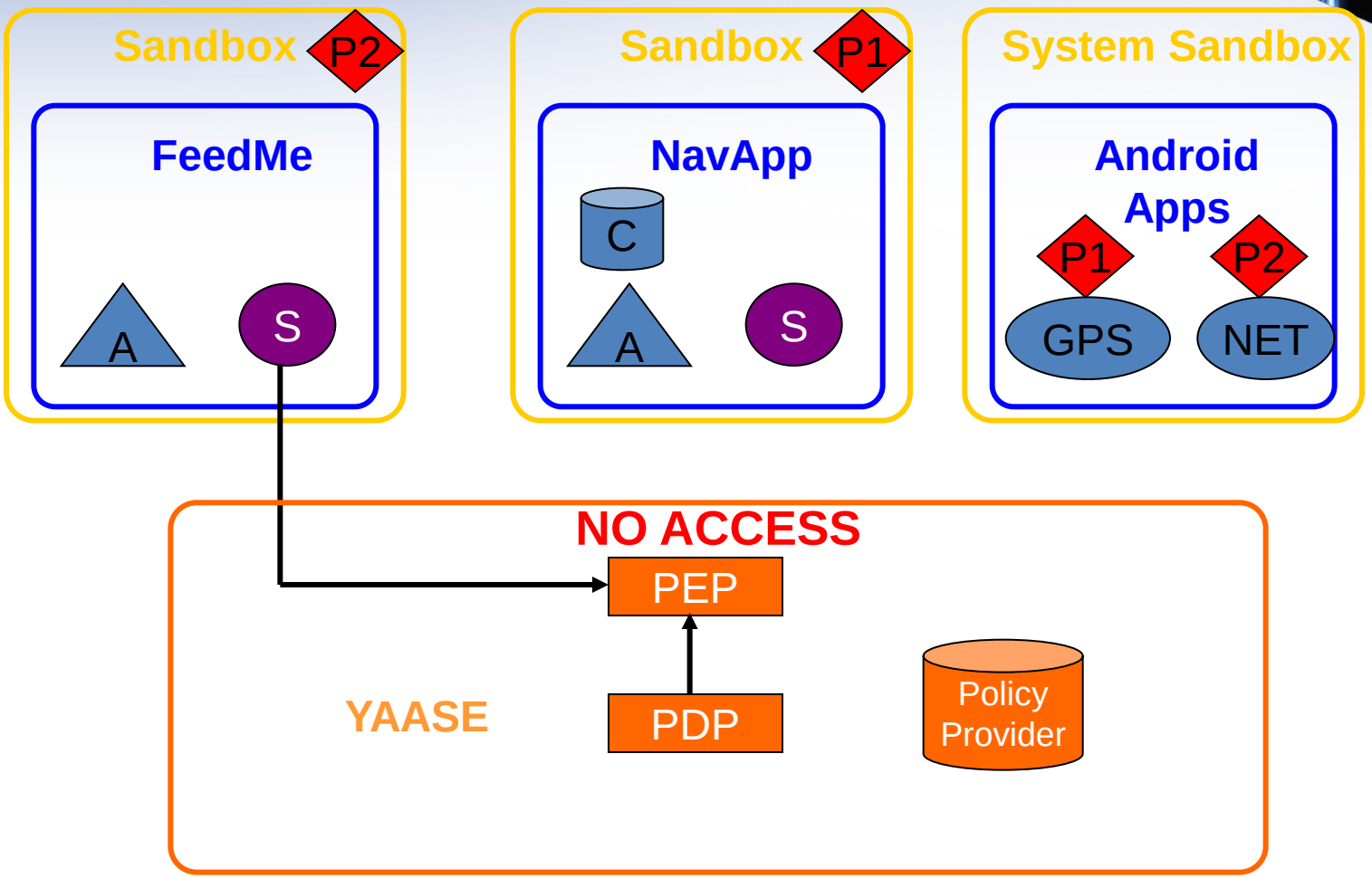
Restrict Approach



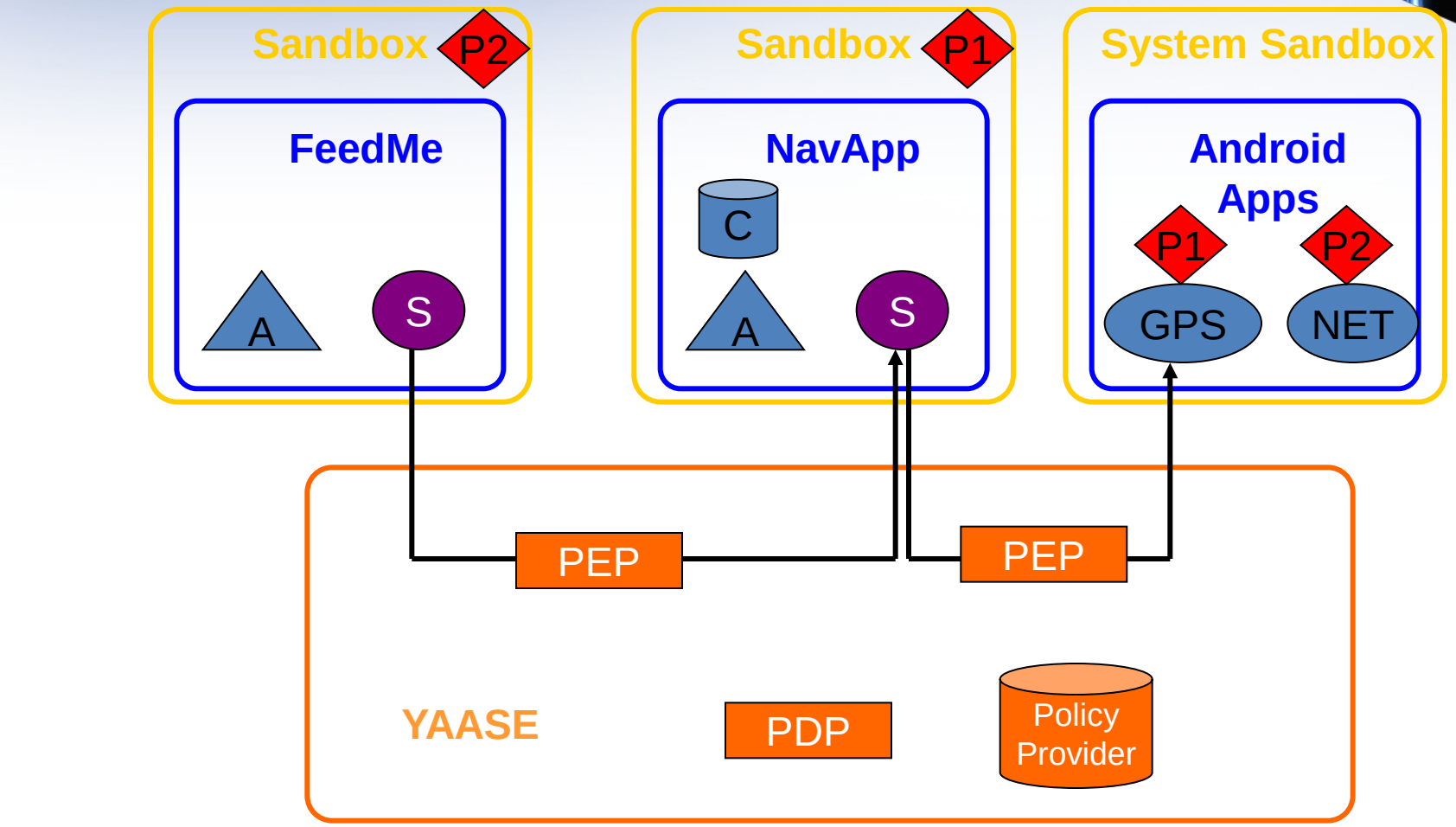
Restrict Approach



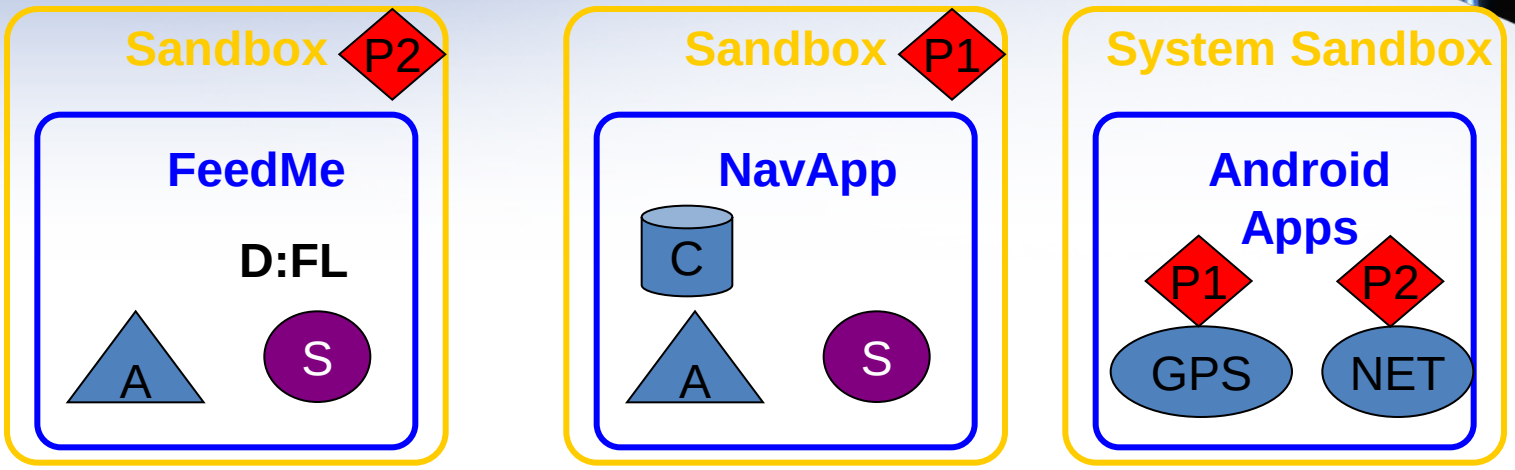
Restrict Approach

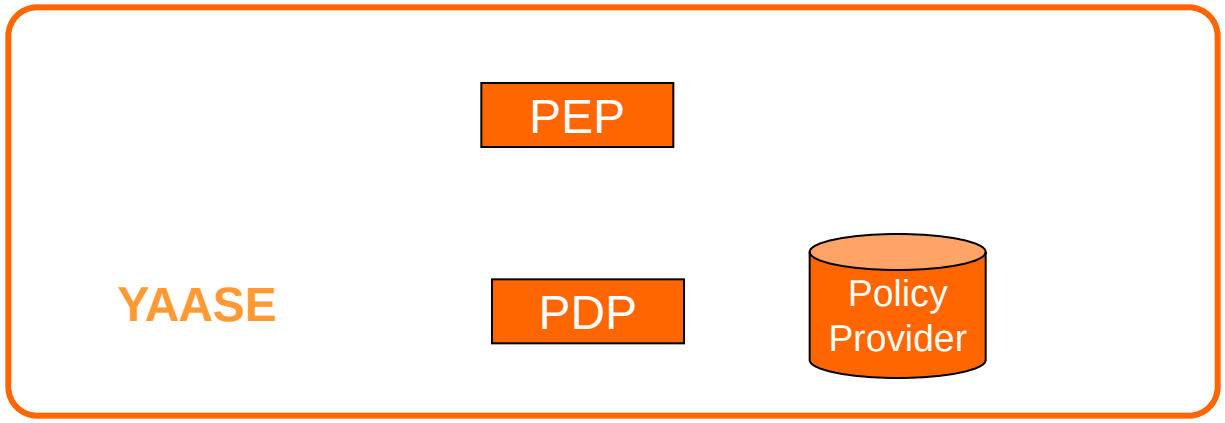


Relaxed Approach

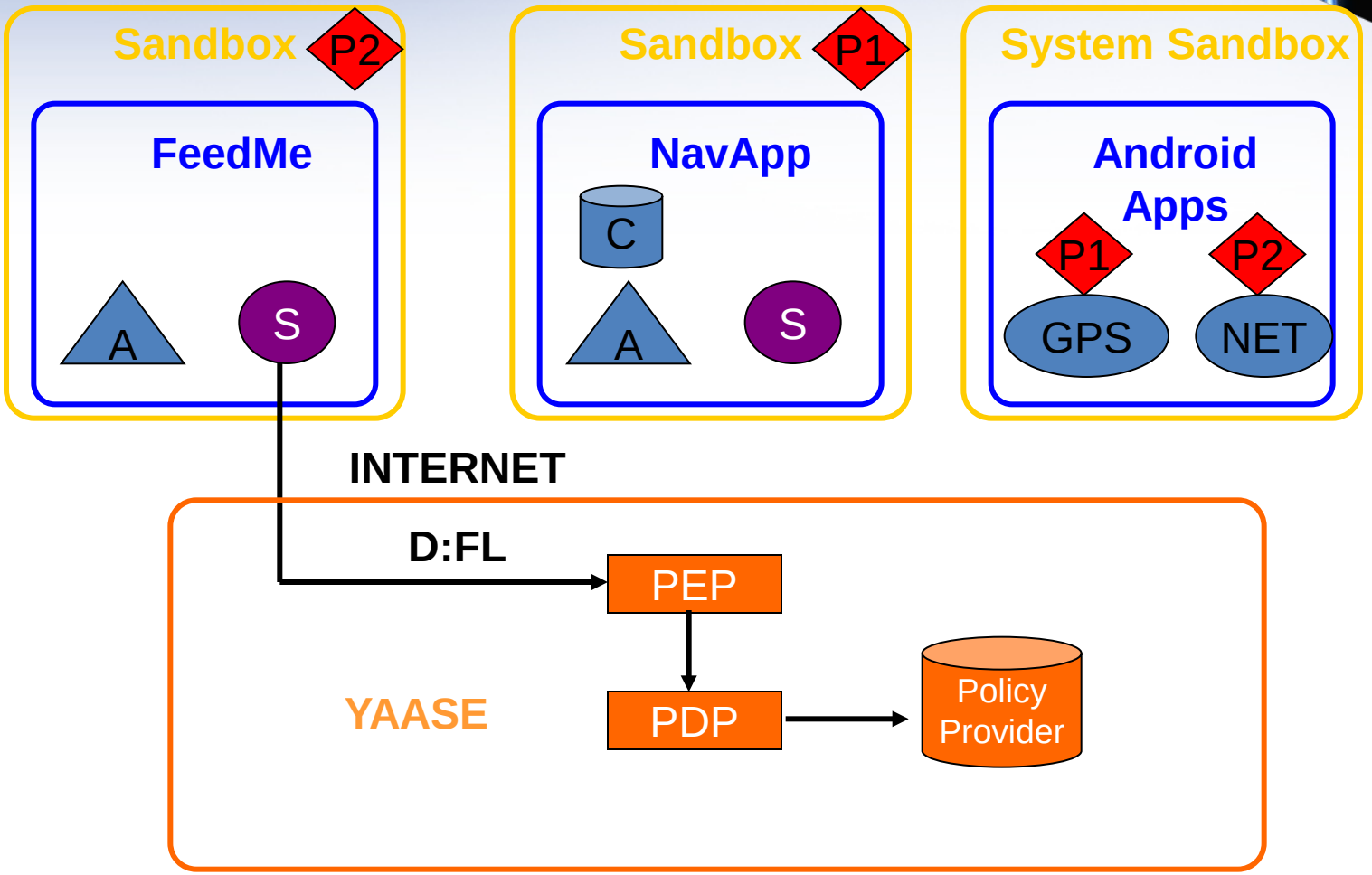


Relaxed Approach





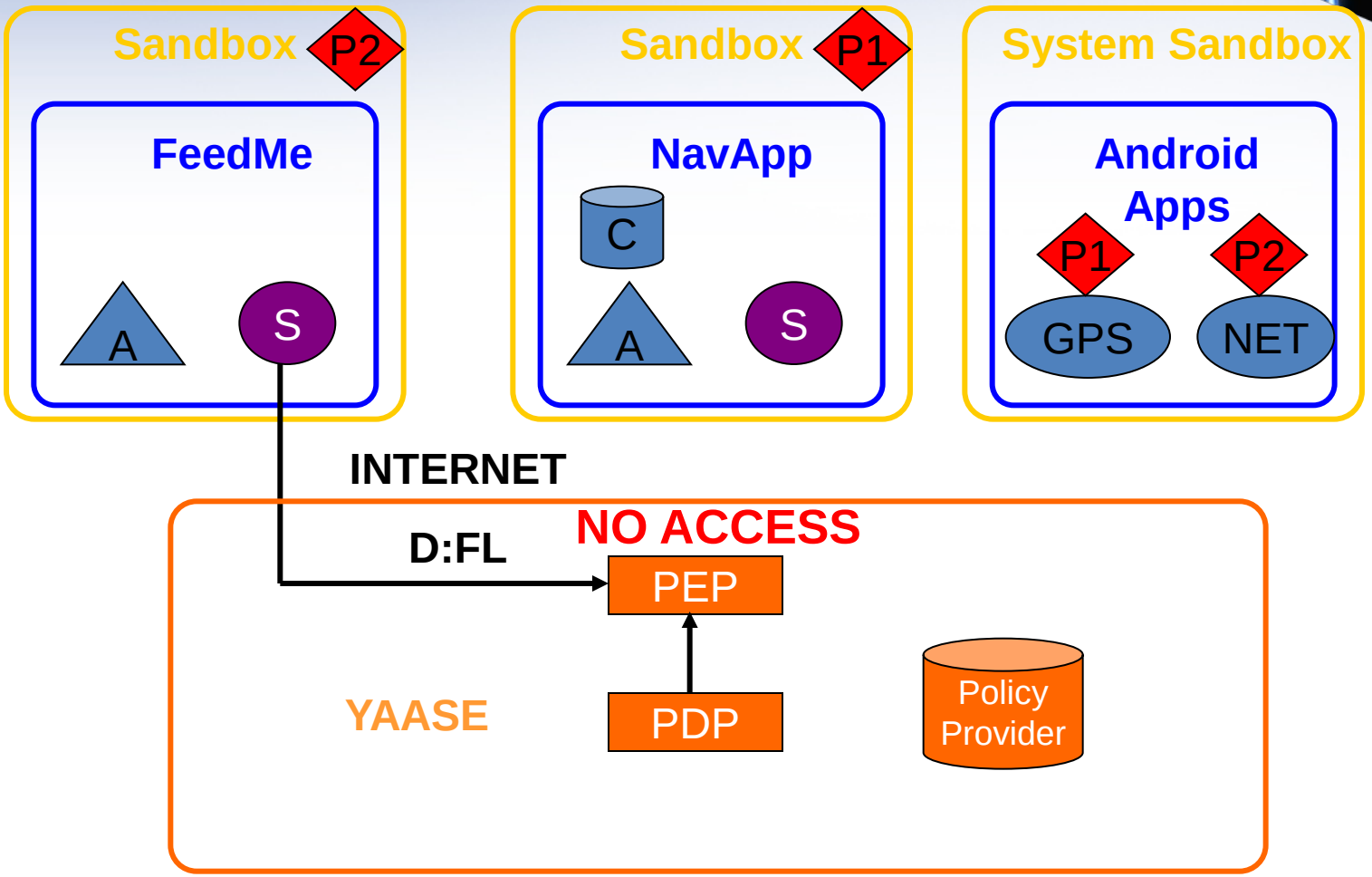
Relaxed Approach



Enforced Policy

```
PolFeedMe:
  FeedME can do send on Internet
  handle “NoLabels”
```

Relaxed Approach



Who is defining the Policies

- Policies Generation should be painless for the user
- Extending the Android Installer to extract from the manifest file information for policy generation
- User can any time change policy settings

Final Remarks

- Standard Android Security framework is insufficient!
- Plethora of security extensions have been performed
- Now it is time that Google starts to take some actions