



Aalto University

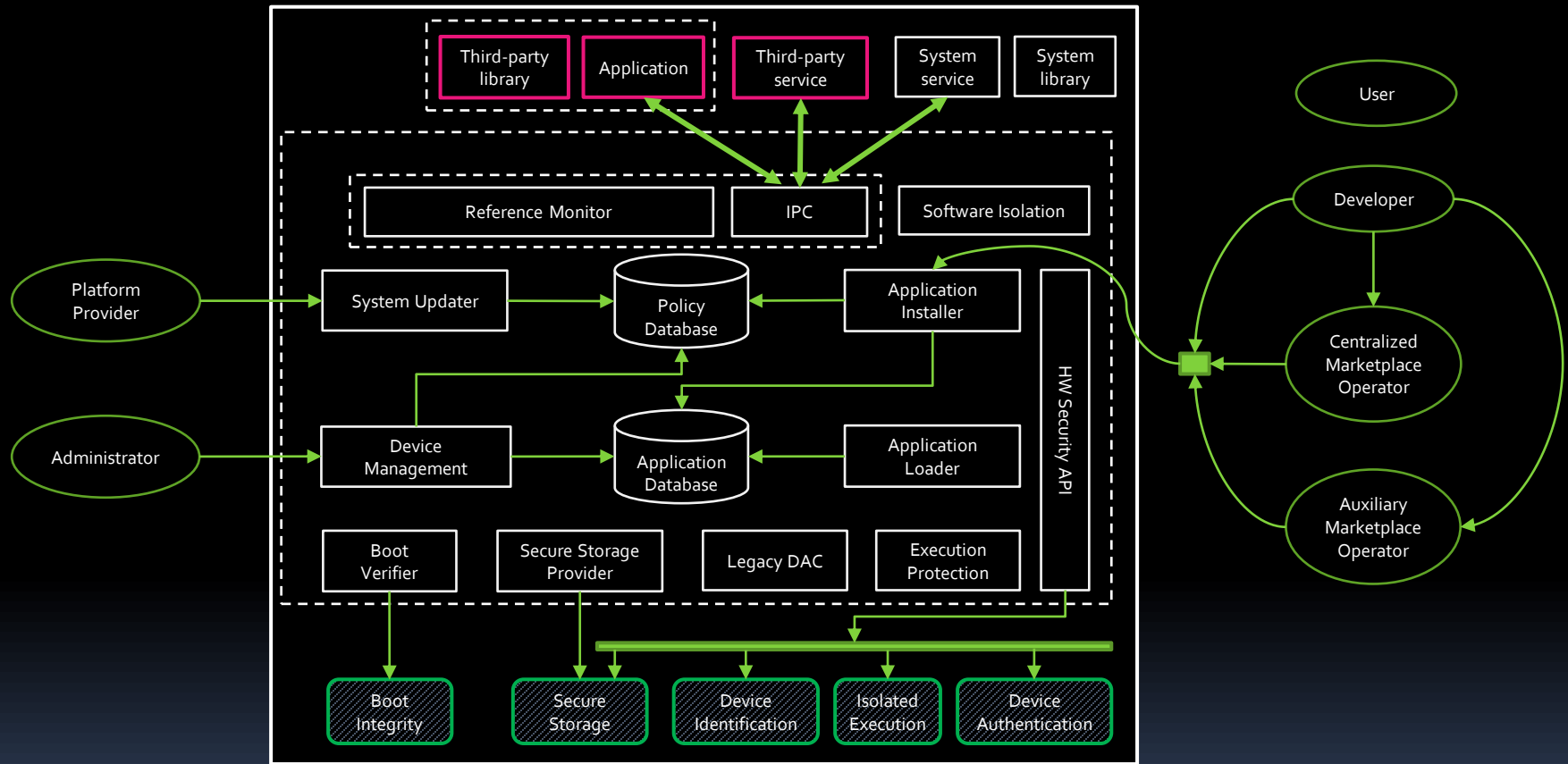
Platform Security

Lecture 3: Android

You will be learning:

- Android as a software platform
 - Internals and surrounding ecosystem
- Security techniques in Android:
 - Application signing
 - Application isolation
 - Permission-based access control
 - Hardware-based security features

Mobile Software platform security



Android in a nutshell

- Linux-based (ARM, x86, x86_64, MIPS)
- Widely used for phones and tablets
 - Wearables, smart TVs, cameras, (handheld) gaming consoles, etc.
- Applications written in Java, Kotlin
 - May include C++ NDK libraries
- Open-source software stack + closed source applications and services

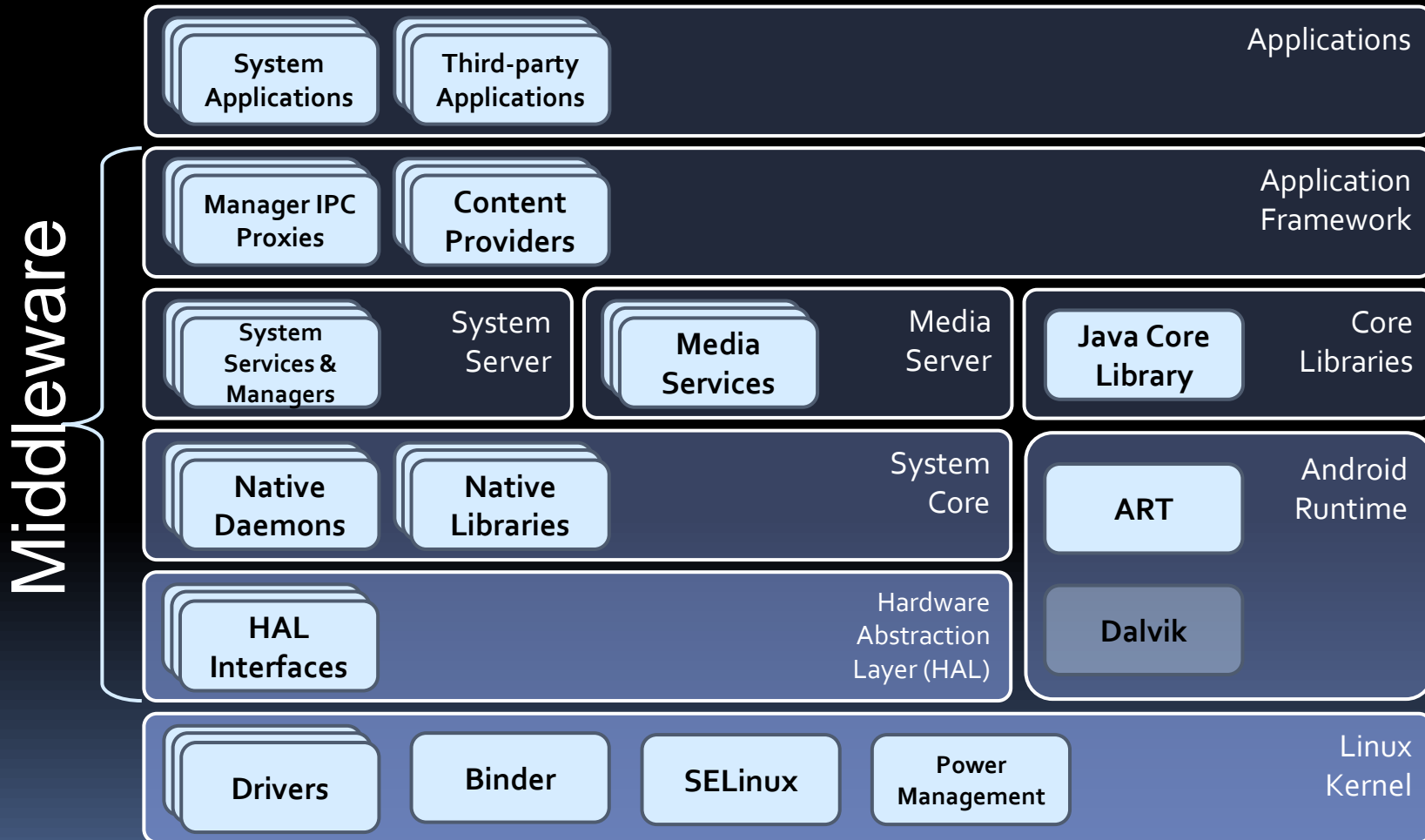
Security goals

- Protect user data
- Protect system resources
- Provide application isolation

On terminology

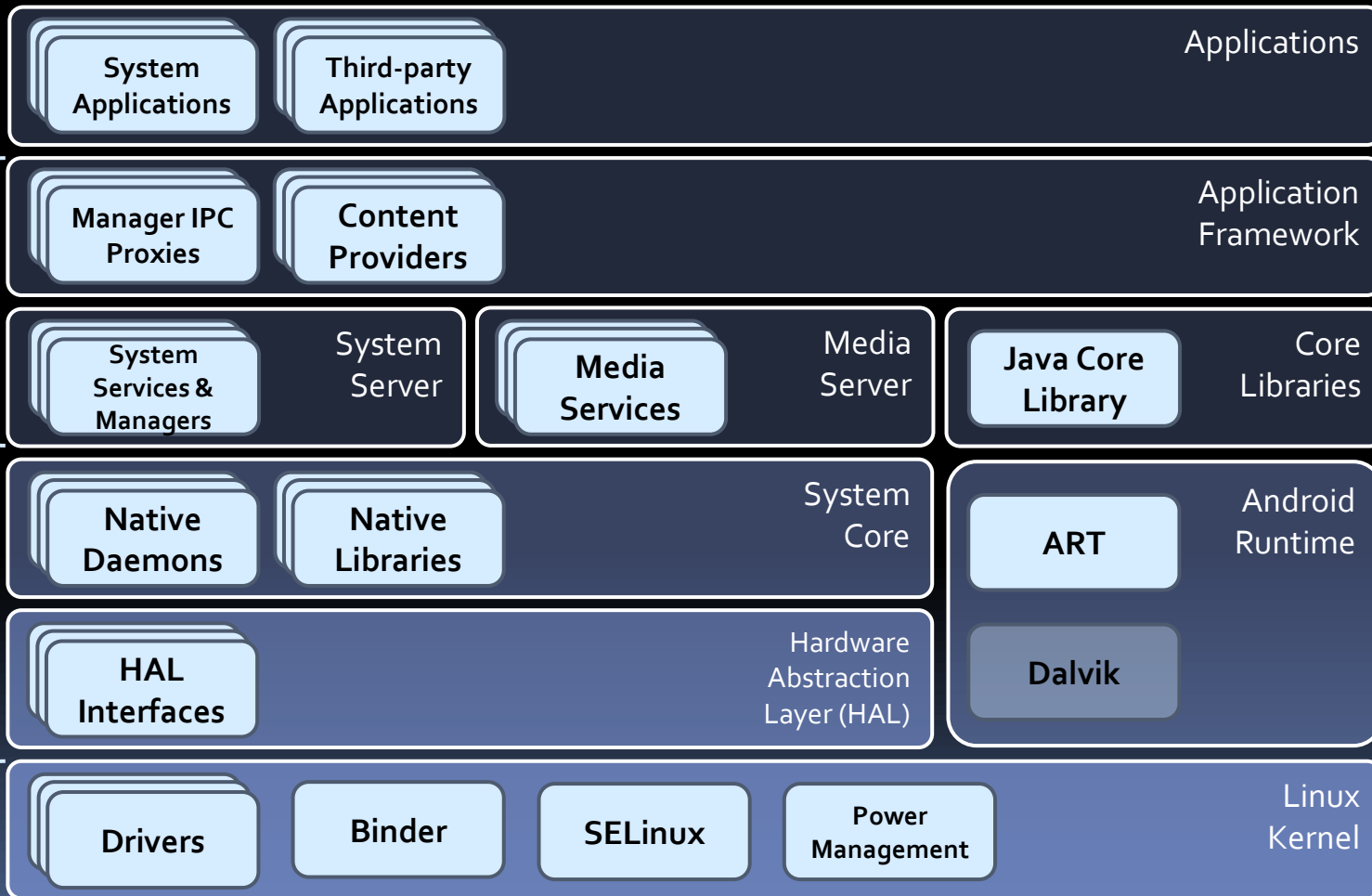
- Linux = the kernel
- "Desktop Linux" \approx GNU / Linux
- Linux DAC = (Unix) file permissions
- Linux MAC = SELinux
- Permissions = Android app perms.
or SELinux

Android Software Stack

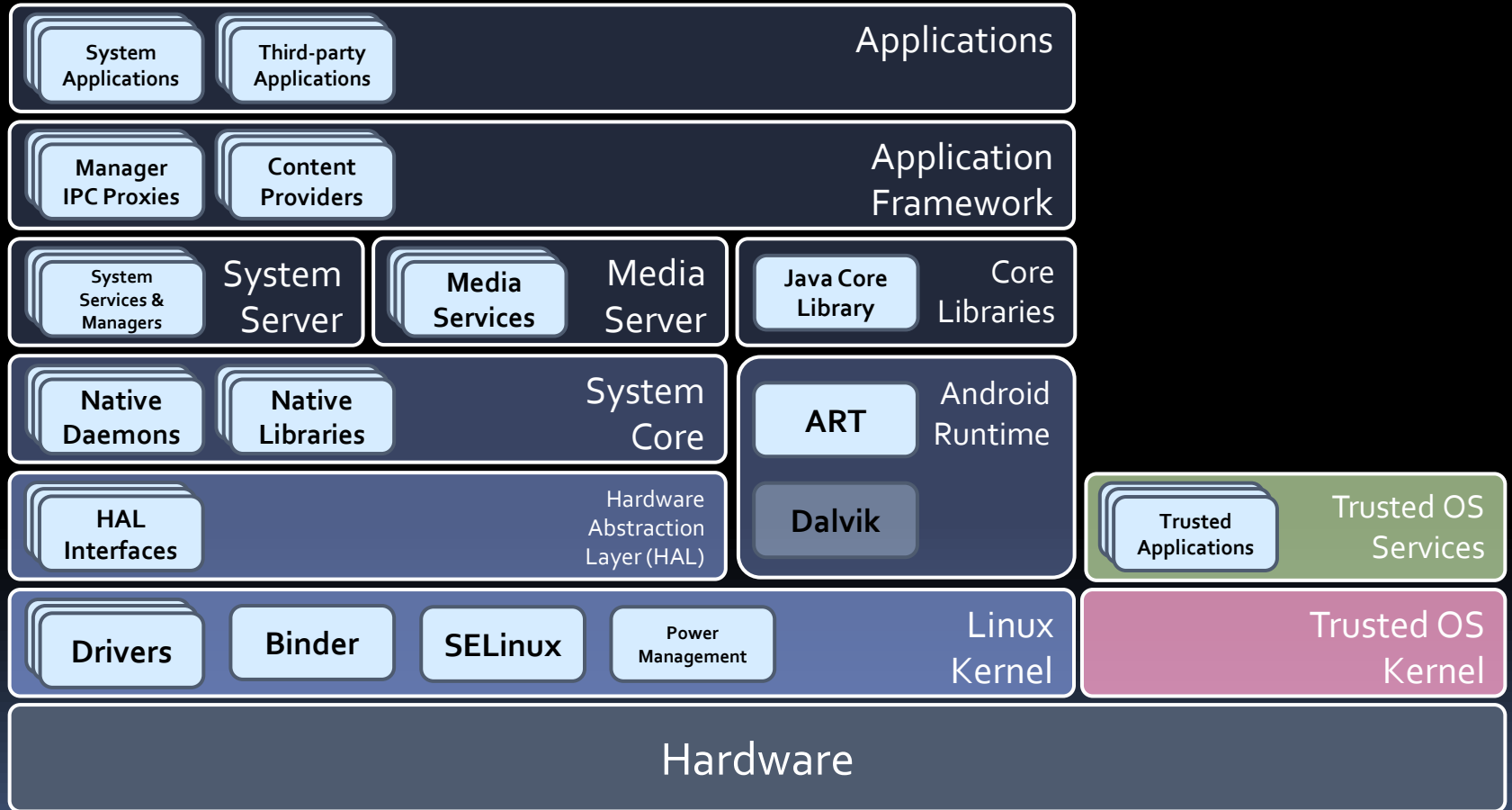


Android Software Stack

Reference monitors



Android Software Stack

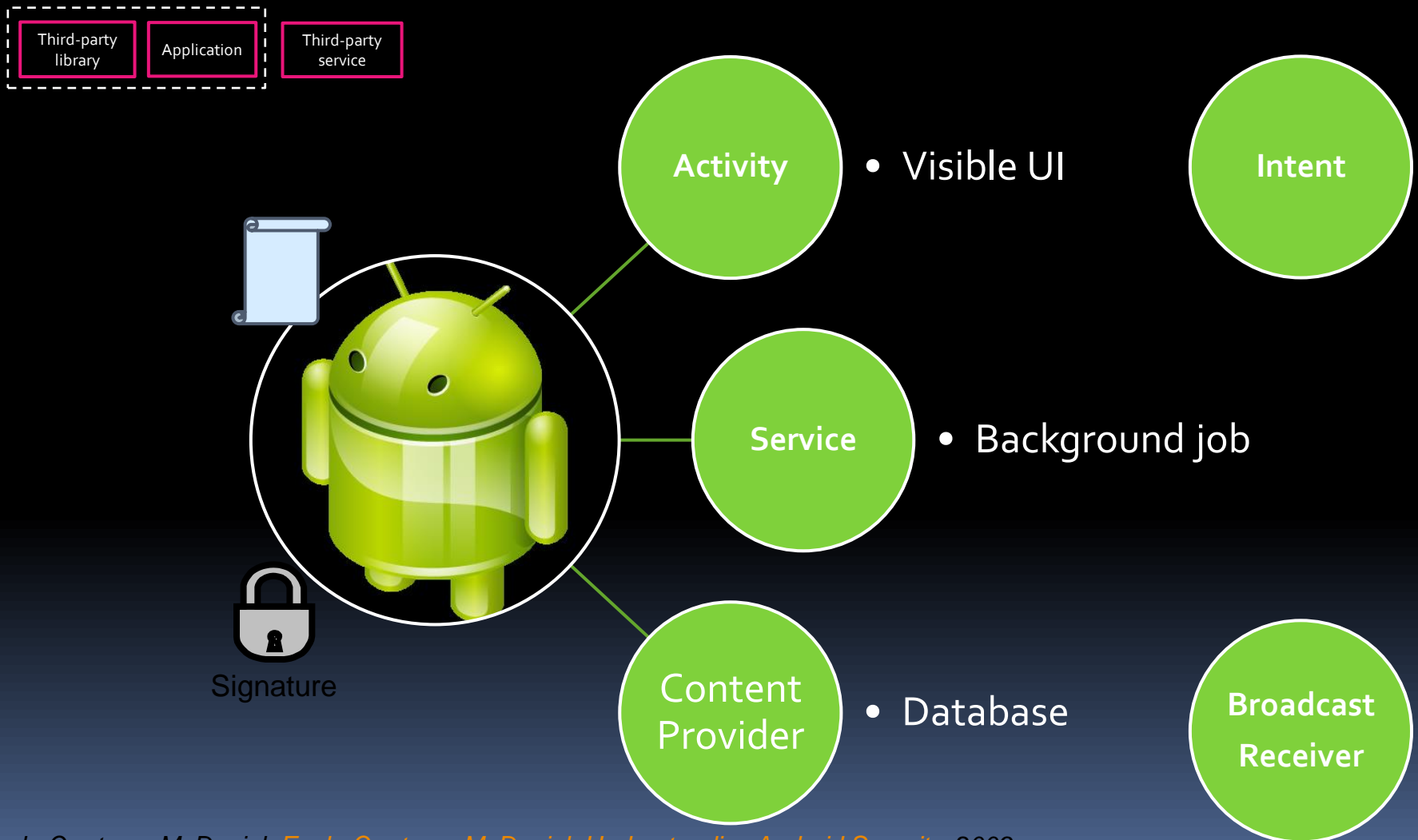


Versions & API Levels

- Confectionary code names for major versions
- API level indicates application framework version

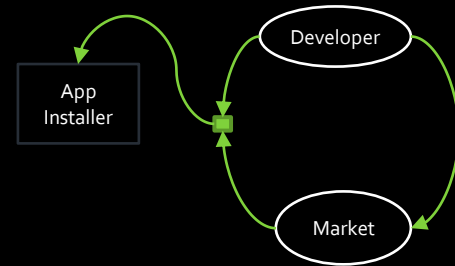
Version	Code name	Release date	API level
9	Pie	Aug 6, 2018	API level 28
8.1.0	Oreo	Dec 5, 2017	API level 27
8.0.0	Oreo	Aug 21, 2017	API level 26
7.1	Nougat	Oct 4, 2016	API level 25
7.0	Nougat	Aug 22, 2016	API level 24
6.0	Marshmallow	Oct 5, 2015	API level 23
...			

Application components



Software distribution

- Apps from multiple sources
 - Google Play
 - Auxiliary marketplaces
 - Sideloading
 - Pre-installed software
- Marketplace services
 - Discovery
 - Purchase & Installation
 - User-submitted ratings / flagging
 - Malware scans (Google Bouncer)
 - Remote application installation & removal



Application signing

- Goal: same-origin policy for apps



Application signing (cont.)

- For **application packages (APKs)**
 - Self-signed X.509 certificates (**no PKI!**)
 - Package update **requires same certificate**
 - Developer manages APK **signing key** (**cannot be recovered**)
- For **app bundles (AABs)**
 - **Upload format** for compiled sources and resources
 - Defer **APK generation and signing** to marketplace
 - Google manages APK **signing key**
 - Developer signs AABs or APKs with **upload key** (can be **reset**)

Elenkov. [Android's Security Architecture](#). 2015.

Google Play. [Manage your app signing keys](#). 2018

Android Open Source Project. [About Android App Bundles](#). 2018

APK Signature Scheme

- JAR signing (v1)
 - Individual signature for each integrity protected file
 - Does not protect ZIP metadata
- APK Signature Scheme v2 (since 7.0)
 - Whole-file signature stored in **APK Signing Block**
- APK Signature Scheme v3 (since 8.0)
 - Adds **APK key rotation** via **proof-of-rotation** attribute
 - Old signing certs marked as trusted for granting signature permissions via **self-trusted-old-certs** attribute

APK Signature Scheme v2 & v3



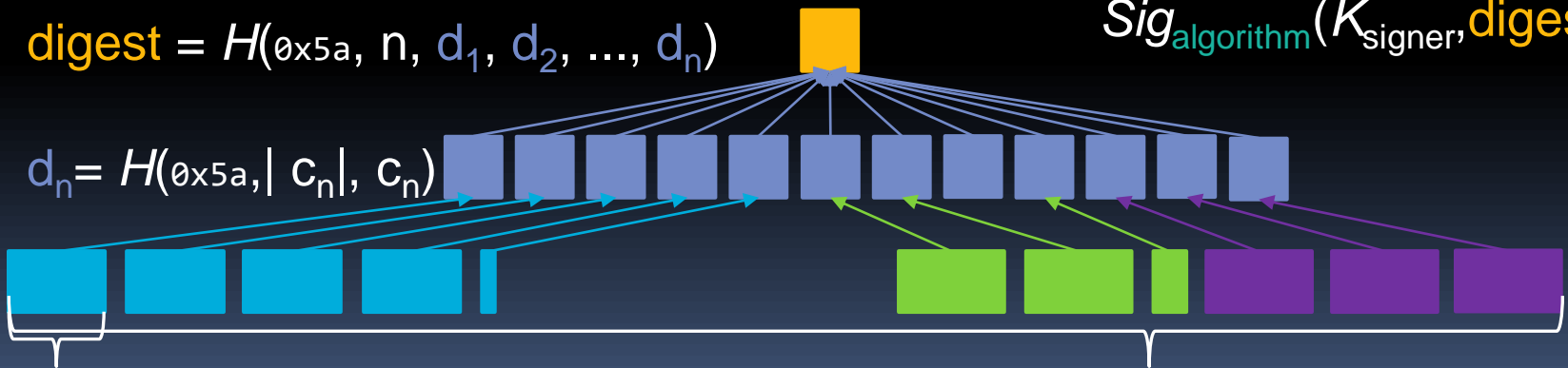
For each signer:

- Signer's identity as X509 certificate chain
- Signature as $\langle \text{algorithm}, \text{digest}, \text{signature} \rangle$ -tuples
- Additional attributes as $\langle \text{key}, \text{value} \rangle$ -pairs

Digest calculated as two-level Merkle tree:

$$\text{digest} = H(\text{0x5a}, n, d_1, d_2, \dots, d_n)$$

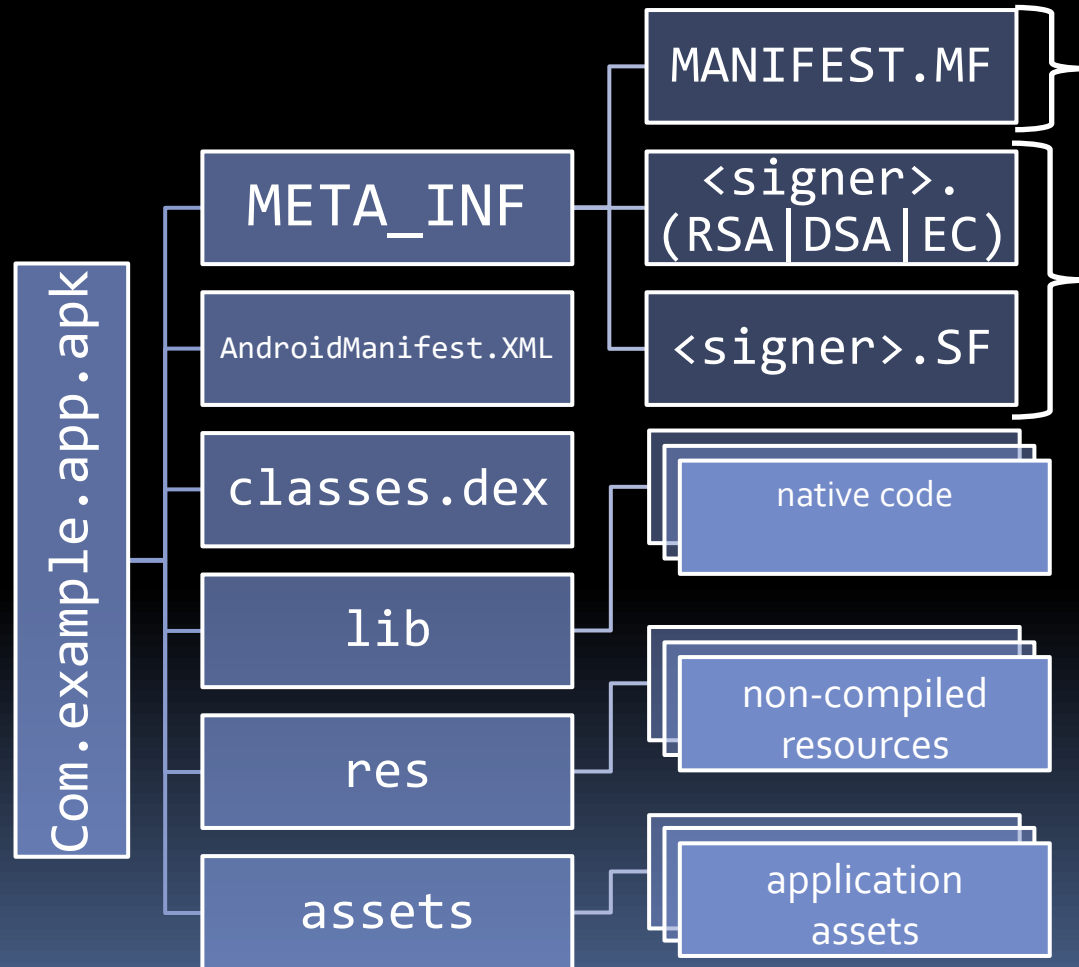
$$\text{signature} = \text{Sig}_{\text{algorithm}}(K_{\text{signer}}, \text{digest})$$



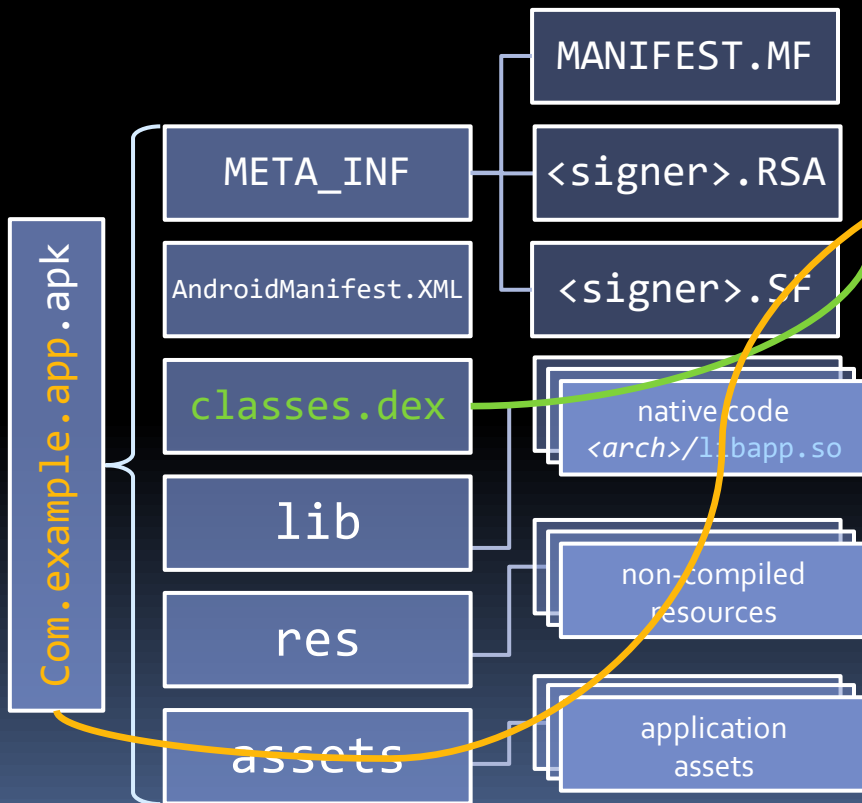
c_i = up to 1MB chunk of ZIP section

n = number of chunks

APK Contents



Package Installation



- Code and resources (common)
 - `/data/app/com.example.app/`
 - lib/<arch>/libapp.so
 - oat/<arch>/base.odex
 - base.apk
- Data (per user)
 - `/data/user/0/com.example.app/`
 - files/
 - databases/
 - shared_prefs/
 - `/data/user/1/com.example.app/`
 - ...

Application isolation

- Goal: Applications cannot interfere with one another

Application isolation

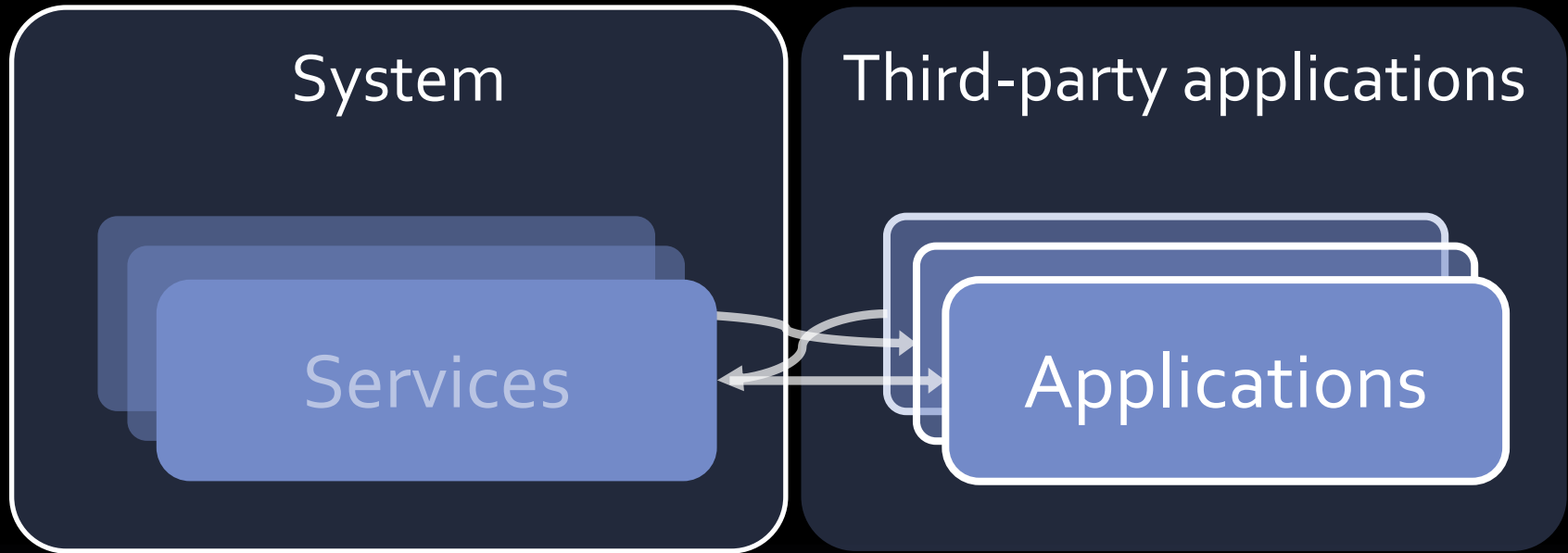
Implementation on Android:

- Kernel: **Process & memory protection**
- Kernel: Linux **DAC**
- Kernel: Linux **MAC (SELinux)**
- Kernel: **Seccomp** filters for app processes
- Middleware: **mediation** of **Binder IPC**
- Applications run in separate ART virtual machine instances

Application Sandbox

- Each application assigned a Unix UID
 - One UID **per user per application** (since 5.0)
 - UID owns
 - Filesystem resources in `/data/user/<nr>/`
 - Processes
 - Permissions (!)
- Applications from same developer
(= signed with same developer key)
may share UID sandbox

Application isolation



- Linux DAC domain (UID)

Rooting

- Rooting applications exploit vulnerabilities in privileged system daemons to obtain shell
- Note: bootloader unlocking intentionally supported by many OEMs
 - e.g. fastboot oem unlock

SELinux in Android

- Goal: System services and applications should not be able to deviate from their intended *modus operandi*

SELinux in Android (cont.)

- Implementation on Android:
 - Kernel-level MAC (SELinux) – Policies based on SELinux context
 - Middleware MAC (MMAC) – Policies based on package identity

SELinux in Android (cont.)

- Enforces MAC even for processes running with root/superuser privileges (since 4.4)
- Blocks many root exploits and misconfigurations
- Can reduce attack surface of kernel exploits by limiting access to vulnerable syscalls
 - 39% of 2016 security issues target kernel as opposed to userspace (up from 9% in 2015)
 - 85% of kernel bugs attributed to device drivers

Smalley, Craig. [Smalley, Craig: Security Enhanced \(SE\) Android](#). 2013

Android Open Source Project. [Security-Enhanced Linux in Android](#). 2015 [Skip to App Isolation](#)

Van Deer Stoep, Jeff. [Android: protecting the kernel](#). 2016

Google. [What's new in Android Security](#) (Google I/O'17). 2017

SELinux in Android

- Type Enforcement
 - Access Control Policy described as **rules** on abstract **labels**
 - System processes (**subjects**) and system resources (**objects**) mapped to labels

SELinux in Android

- *Domain* - Label for process(es)
- *Type* - Label for object(s)
- *Class* - Kind of object being accessed
 - (e.g. file, socket)
- *Permission* - Operation being performed
 - (e.g. read, write)

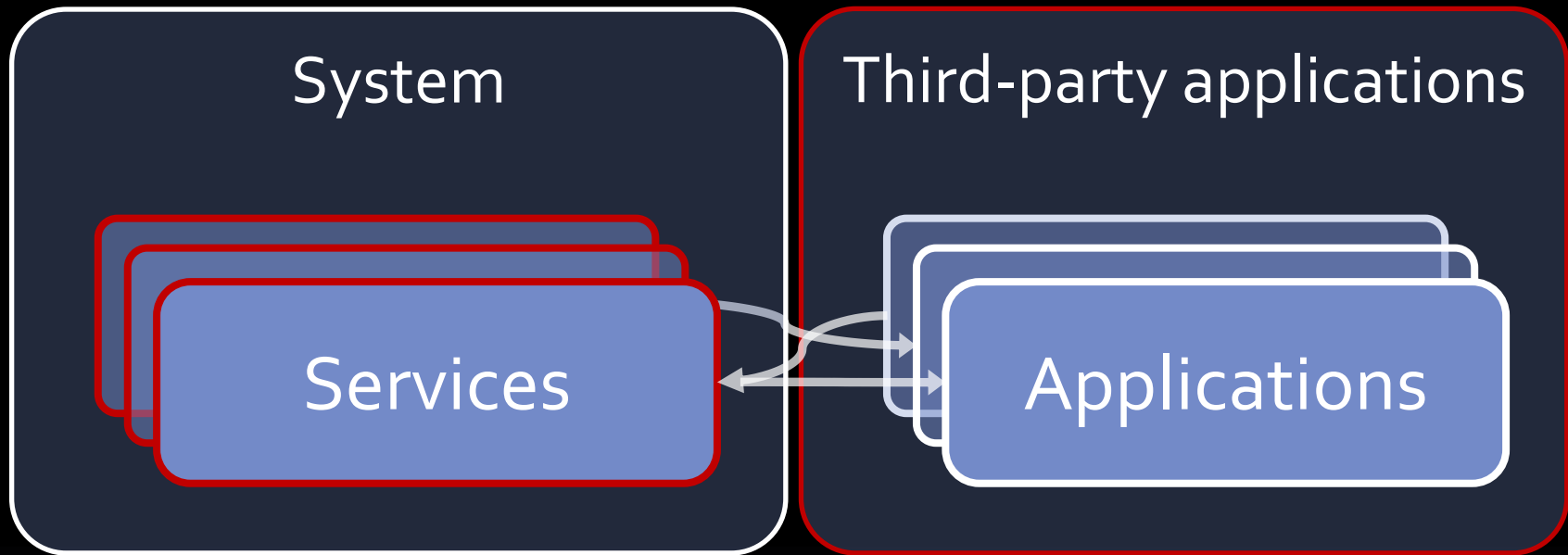
SELinux rules

- **EVERYTHING FORBIDDEN BY DEFAULT!**
- **ALLOW** rules define how subjects may interact with objects
- **NEVERALLOW** rules prevent specific **ALLOW** rules from being added to a policy

SELinux attributes

- Attribute
 - Identifier assigned to a group of types
 - Not a type in itself!
 - Used in rules when referring to groups of types
 - Cannot be used to label subjects nor objects
- Used to define behaviour common to multiple types without repeating rules
- Composition of attributes enables modularity

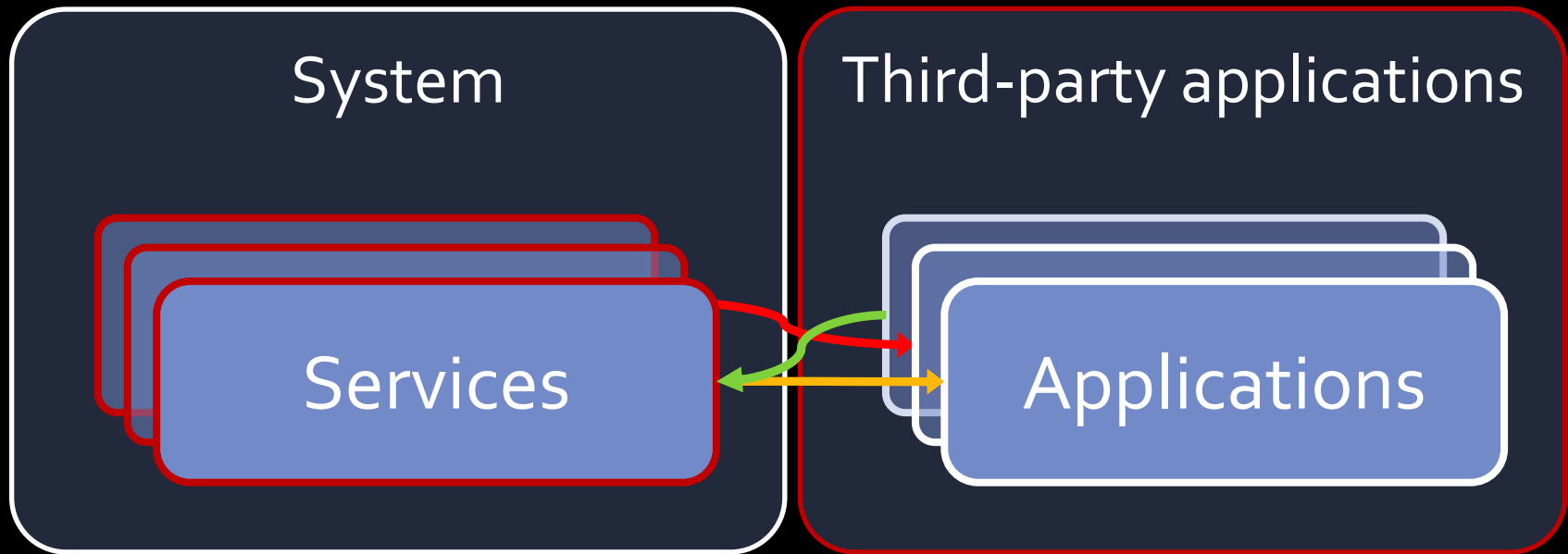
Application isolation (cont.)



Linux DAC domain (UID)

Linux MAC domain (SELinux)

Protected APIs



- Unrestricted service calls
- Approval-based service calls
- Restricted service calls

Protected APIs

- Goal: Protect system resources from unauthorized access

Protected APIs (cont.)

- Implementation in Android:
 - Protected APIs for “risky” actions
 - Permission-based access control

Protected APIs (cont.)

- What kinds of system calls on a smartphone would warrant protecting and why?

Sensitive user data

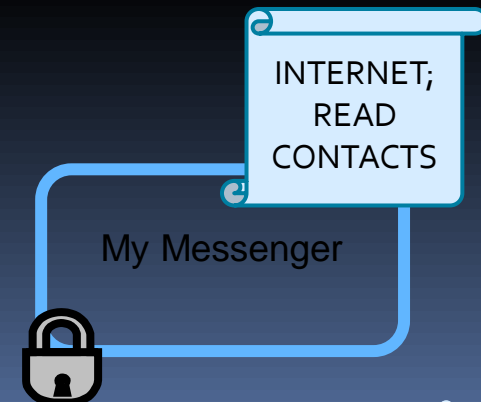
- Subject to permissions checks:
 - Personal information (e.g. contacts)
 - Sensitive input devices (e.g. camera)
 - Location tracking can be manually disabled
 - Actions that cost money (e.g. calls, SMS)
 - Device metadata (e.g. logs)

Access control & permissions

- Goal: Controls application access to protected APIs (and each other)
 - User agency vs. protecting system resources
 - Usability of security features

Permission assignment

- Application declares **all permissions** in **AndroidManifest.xml**
- Permissions assigned to **application UID**
- Some permissions not user-grantable
 - Only available to pre-installed applications



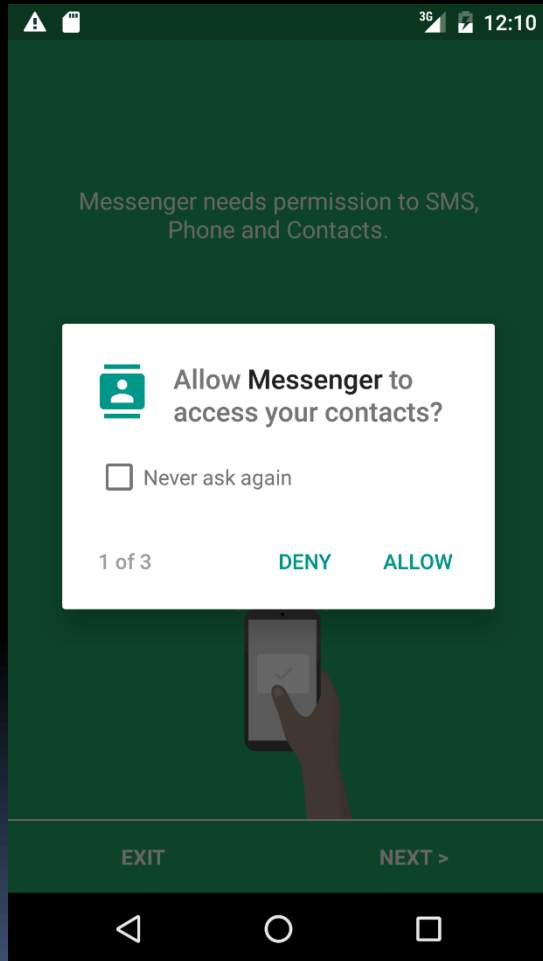
Android permissions

- 4 categories
 - Normal
 - Dangerous
 - Signature
 - Signature | Privileged

Permission assignment

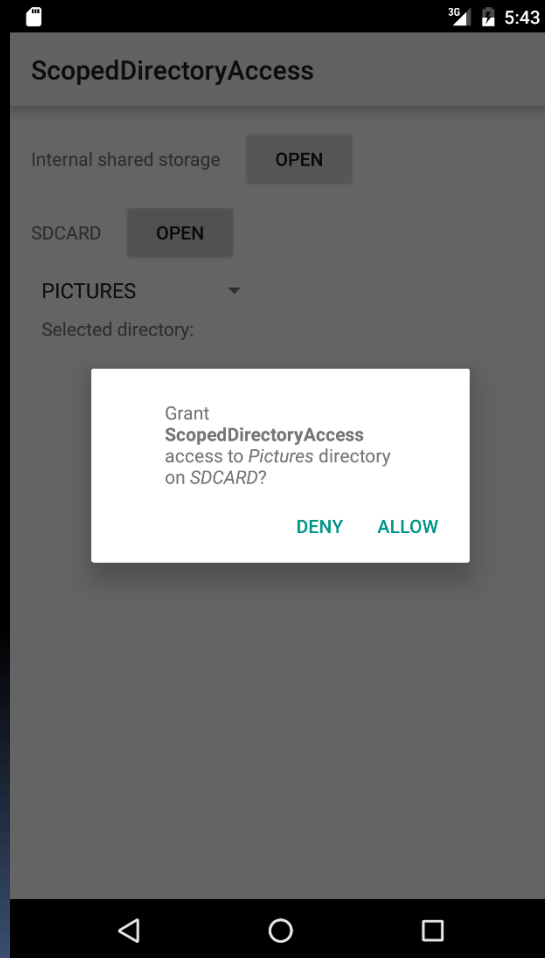
- **Normal** permission granted automatically
- **Dangerous** permissions require user approval at install time (up to 5.1) or run time (since 6.0)
- **Signature** permissions granted if app signature matches the declarer of the permission (or **self-trusted-old-certs**)
- **Privileged** permissions granted only to system apps
 - Subject to whitelist by PackageManager (since 6.0)

User approval (since 6.0)



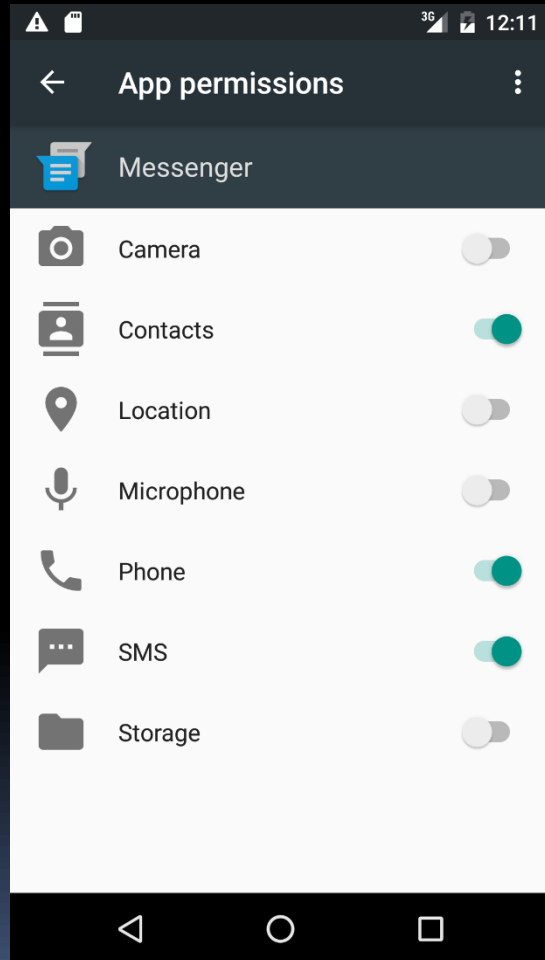
- **Dangerous** permissions require user approval at runtime
- If not granted, application **continues to run** with limited capabilities
- Permission managed **per application, per user**
 - Stored in `/data/system/users/<nr>/runtime-permissions.xml`

Scoped directory access (since 7.0)



- Allow access to specific shared storage directories

Permission revocation



- May be revoked later from application settings
- Also install-time permissions may be revoked
 - applications may not handle revocation well!

Alternatives to obtaining permissions

- Delegate task to other application using *Intent*, e.g. invoke Camera app using ACTION_IMAGE_CAPTURE Intent
 - Caller does not need CAMERA permission
 - Caller cannot control the user experience, but does not have to provide UI for task
 - If no default app available, user is prompted to designate the handler

Intents

- Messaging object used for **Inter-Component Communication (ICC)**
 - Recall: activities, services, content providers, broadcast receivers
- Addressing
 - **Explicit** – fully qualified component name
 - **Implicit** – Intent filter declared in manifest
 - Provides a mechanism for late binding
- **Pending** Intents
 - Token-based access control delegation

Binder

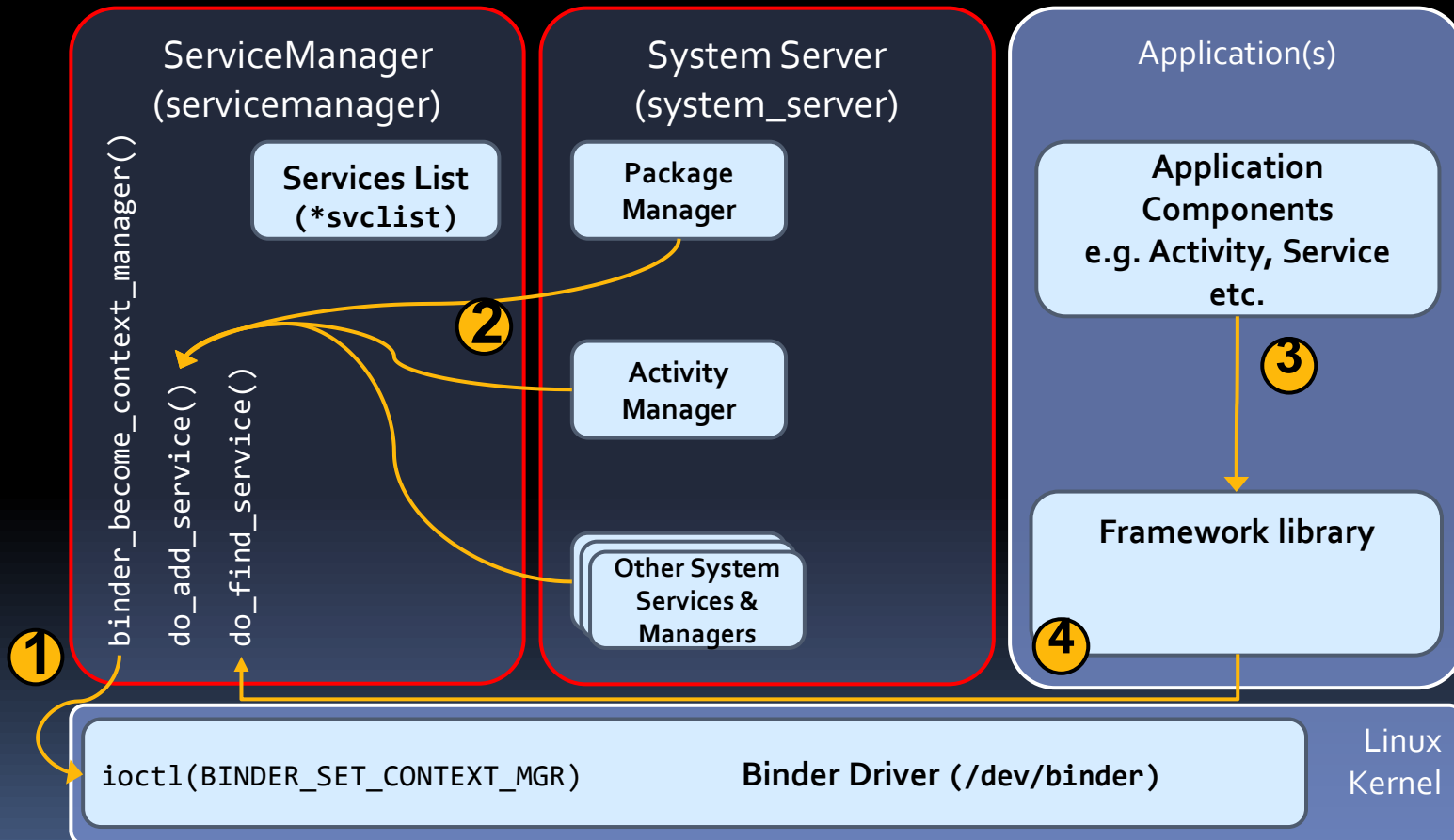
- IPC system for object-orientated operating system services (cf. CORBA/COM)
- Most underlying IPC based on Binder
 - Intents & content providers abstractions on top of Binder
 - Cf. local UNIX-domain sockets, signals, filesystem
 - Bionic libc doesn't support System V IPC
- Does not provide mediation by itself
 - Access mediated by system services

Binder domains (since 8.0)

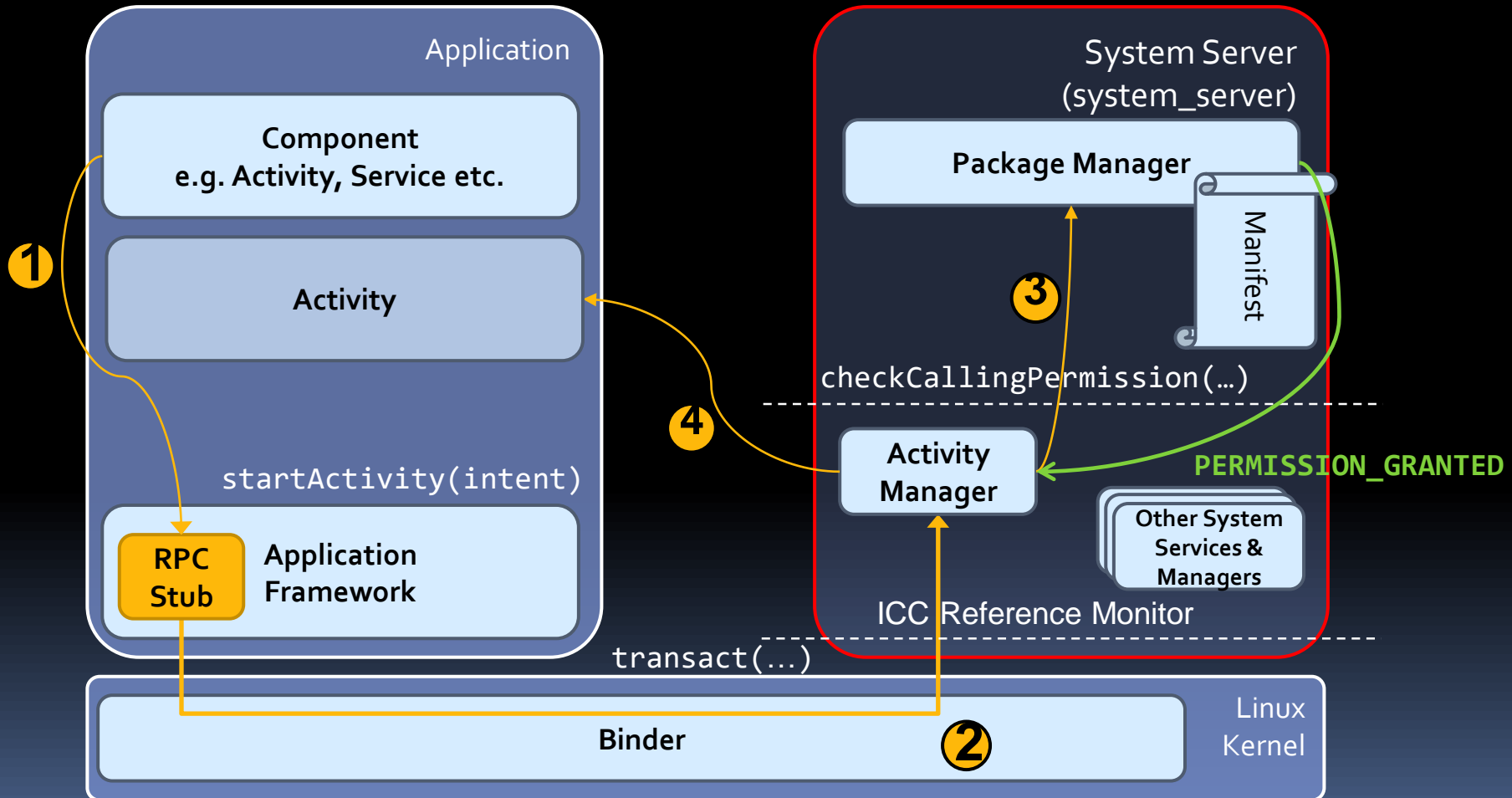
- Binder traffic split in three domains (contexts)
 - Separate device node and context manager
 - Access to device node restricted by SELinux policy
- Context manager acts as “phonebook” for services available via Binder

IPC Domain	Manager	Clients
/dev/binder	servicemanager	Framework / app processes
/dev/hwbinder	hw servicemanager	Framework / HAL processes
/dev/vndbinder	vnd servicemanager	Framework / vendor processes

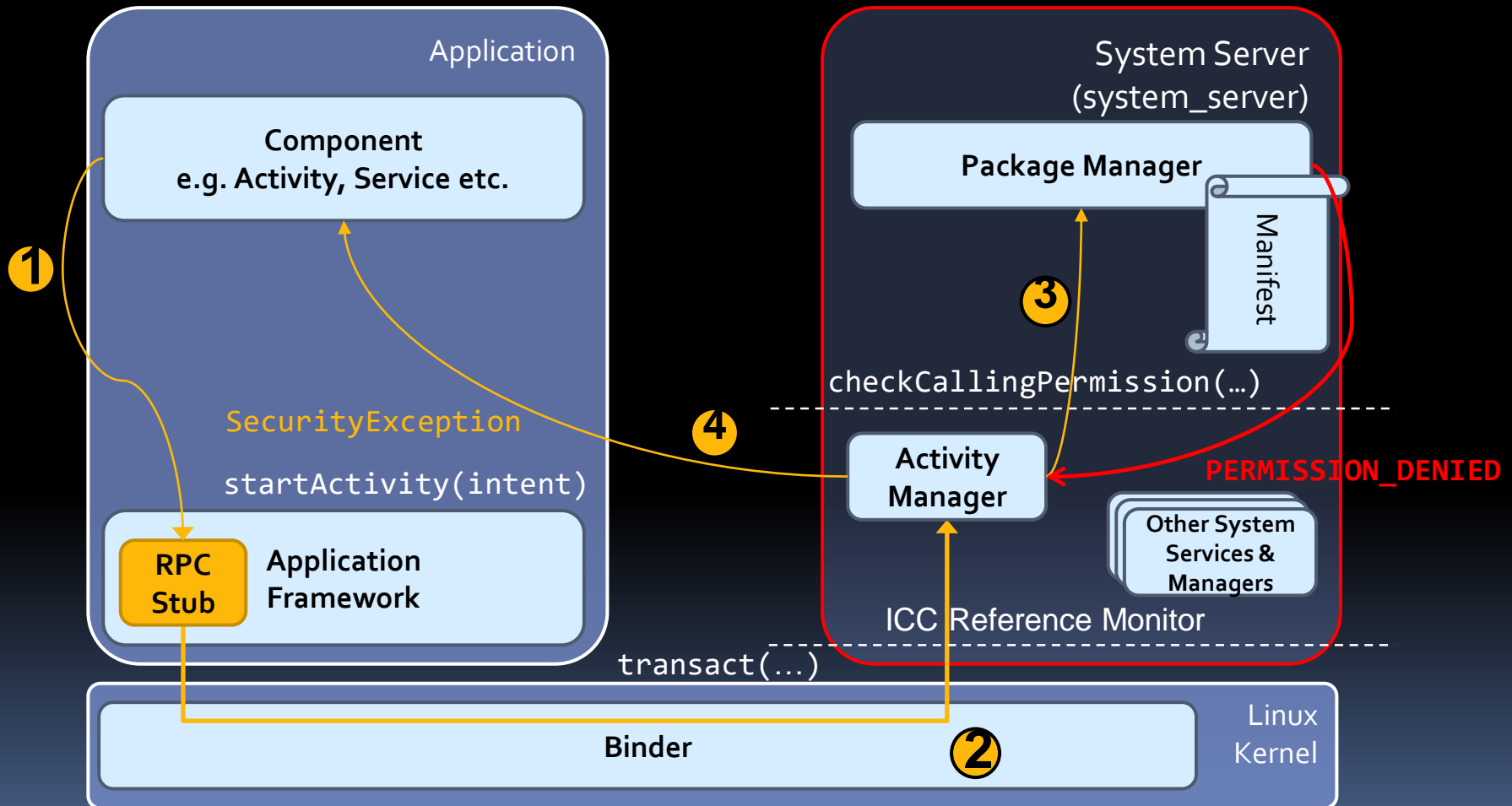
Binder Service Discovery



Starting an Activity



Starting an Activity



Hardware-based security features

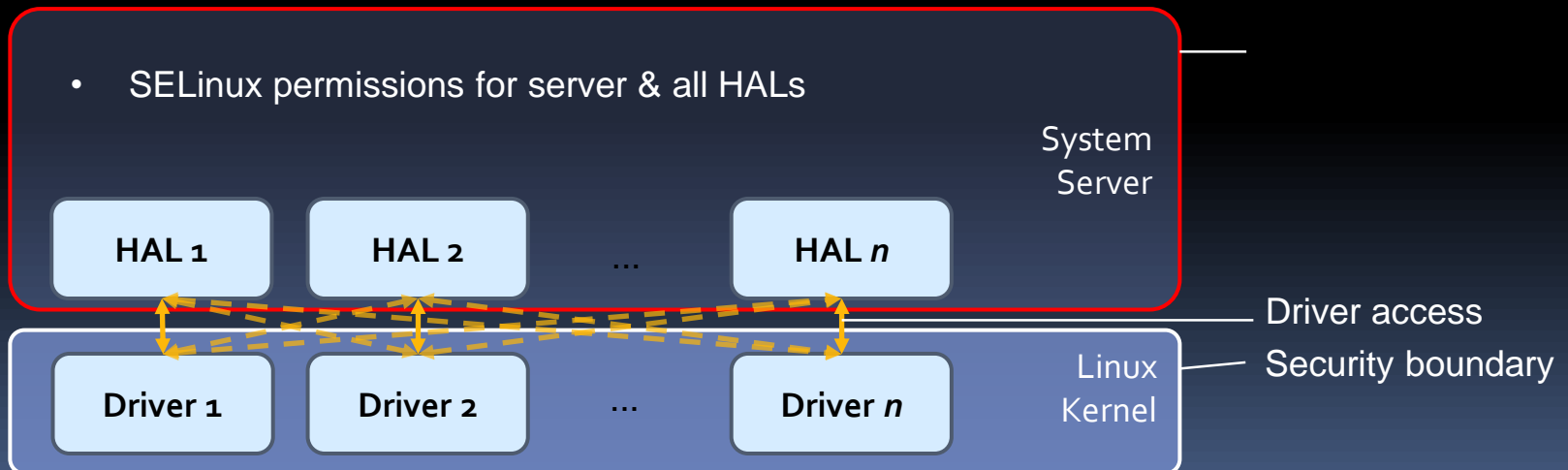
- Goals:
 - Strong but easy to use user authentication
 - Secure storage
 - Platform integrity

Hardware-based security features

- Implementation on Android:
 - Fingerprint / Gatekeeper
 - Keychain / Keystore
 - Full-disk / File-Based encryption
 - Verified boot

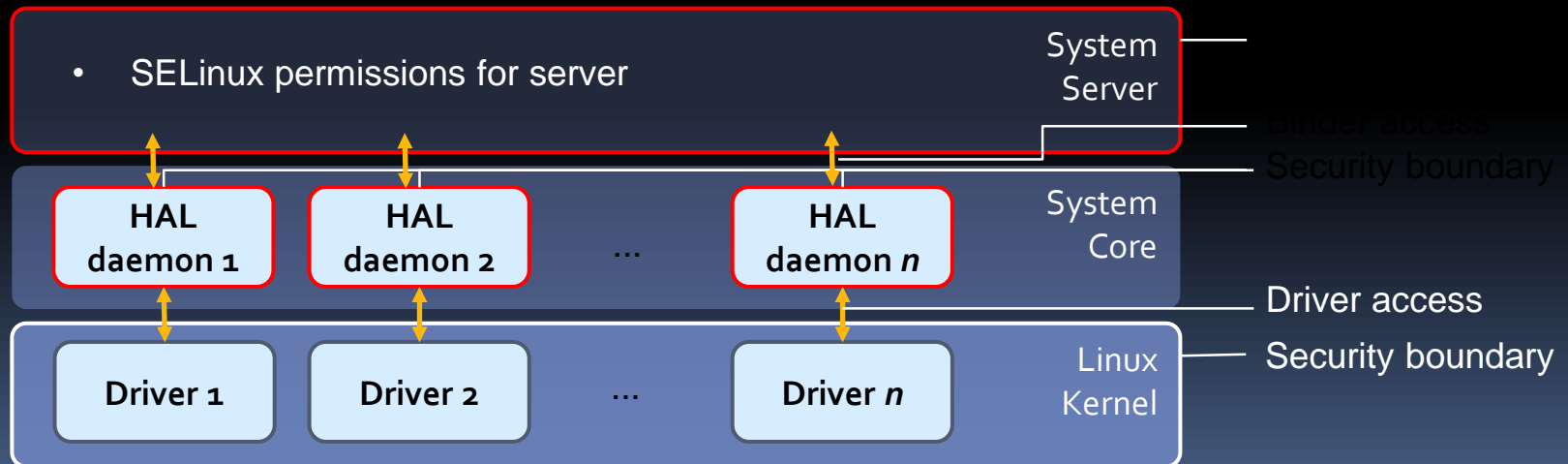
Hardware Abstraction Layers

- Interface between device independent code and device-specific hardware implementation
- Prior to 8.0 HALs packaged as shared libraries
 - Run in same security context as client process
 - Process needs all permissions required by HAL



Hardware Abstraction Layers

- In 8.0 HALs moved into sandboxed processes
 - HAL processes require only permissions for HAL
 - Increased IPC overhead between client and HAL
 - HAL processes practical after improvements to Binder driver (scatter-gather, fine-grained locking, real-time priority inheritance)



Authentication

- Keyguard
 - Pattern
 - PIN / Password
- Gatekeeper HAL (since 6.0)
 - Allows Keyguard to make use of native security features
- Fingerprint HAL (since 6.0)
 - Access to vendor-specific fingerprint hardware

Authentication (cont.)

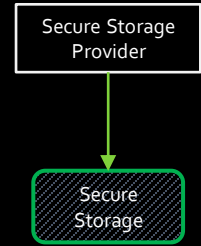
- TrustAgentAPI (since 5.0)
 - Enables services that notify the system about whether they believe the environment of the device to be trusted
- Smart Lock Trust Agent (since 5.0)
 - Trusted Bluetooth device
 - Trusted NFC
 - Trusted place (via geofencing)
 - Facial recognition
 - On-body detection

Elenkov. [*Dissecting Lollipop's SmartLock*](#). 2014.

Nexus Help: [*Set up your device for automatic unlock*](#).

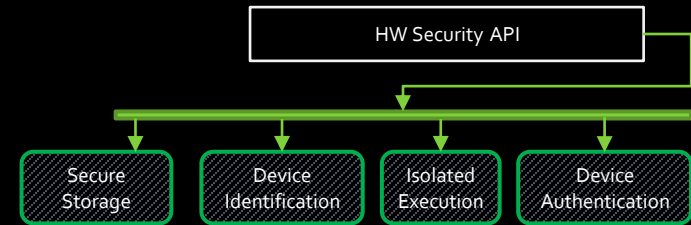
Android Developers. [*Creating and monitoring Geofences*](#).

Android Keystore



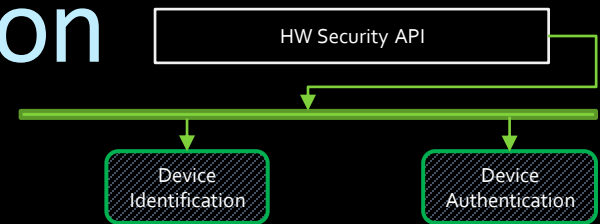
- Protects cryptographic keys from extraction even if application or OS is compromised
 - KeyChain API for system credentials (since 4.0)
 - KeyStore API (Java CE) for application-bound keys
- Hardware-backed keystore binds keys to device (inc. operating system and patch level of system image)
 - Keys not exposed unencrypted outside secure h/w
 - `KeyInfo.isInsideSecureHardware()` (since 6.0) indicates if key is stored in hardware keystore
- KeyChain API is used for Wi-Fi and Virtual Private Network (VPN) certificates

Keystore



- KeymasterHAL
 - Access to hardware-backed keystore
 - Asymmetric key generation, signing and verification (since 4.1)
 - Binder IKeyStoreInterface (since 4.3)
 - Symmetric key support, access control, public key import and private / symmetric key import (since 6.0)
 - Key attestation (since 7.0)
 - ID attestation (since 8.0)
 - Secure key import (since 9.0)
 - Protected Confirmation (since 9.0)
- Strongbox Keymaster
 - backed by hardware security module with discrete CPU, secure storage, true RNG, tamper protection (since 9.0)

Key / ID attestation



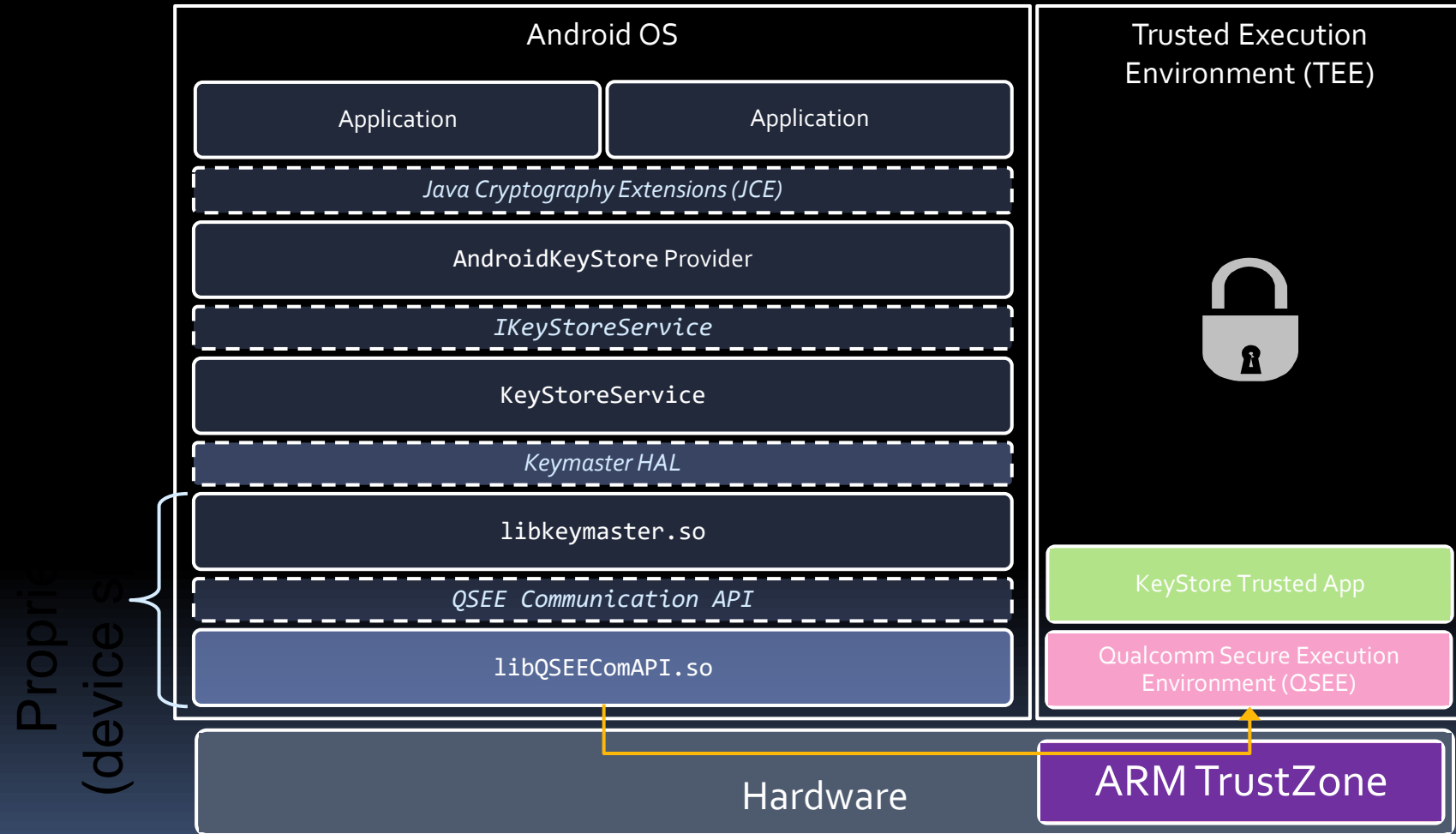
- Signed statement that improves confidence that applications keys stored in hardware-backed keystore
 - attestation certificate chain root cert signed with Google attestation root key
 - accessed via `getCertificateChain()` for KeyStore object's
 - must be validated on separate trusted server!
- ID attestation provides proof of hardware identifiers
 - E.g. brand, manufacturer, model, serial number or IMEI

Android Open Source Project. [Key and ID attestation](#). 2018.

Android Developers. [Key and ID attestation](#). 2018

Android Developers. [Verifying hardware-backed key pairs with Key Attestation](#). 2018

Qualcomm Keymaster architecture



Elenkov. [Keystore redesign in Android M](#). 2015.

Elenkov. [Credential storage enhancements in Android 4.3](#). 2013.

Full Disk Encryption

- Block-device encryption based on **dm-crypt**
- Encrypted on first boot (since 5.0)
- AES 128 CBC and ESSIV:SHA256
- DEK encrypted with AES 128
 - KEK derived from user PIN / password / pattern + hardware-bound key stored in TEE (since 5.0)
- Crypto acceleration through hardware AES (e.g. **dm-req-crypt**)

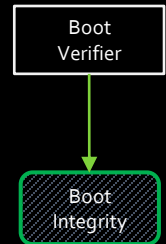
Android Open Source Project. [Full Disk Encryption](#).

Elenkov. [Hardware-accelerated disk encryption in Android M](#). 2015.

File-Based Encryption (since 7.0)

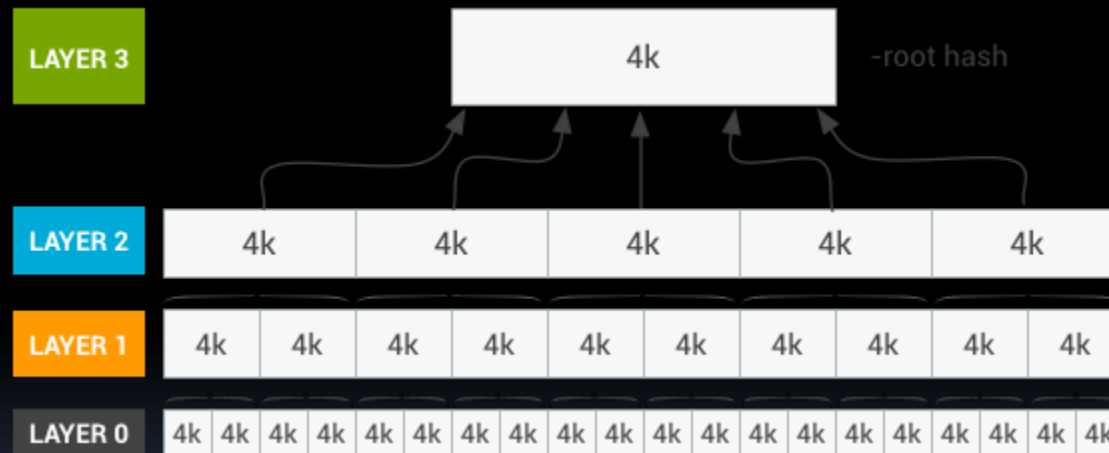
- Direct Boot enables supported apps to operate before user unlocks device
- Two storage locations for data:
 - Credential Encrypted (CE) storage (default) only available after user has unlocked device
 - Device Encrypted (DE) storage available during Direct Boot and after user has unlocked device (requires hardware-backed Keymaster)

Verified Boot

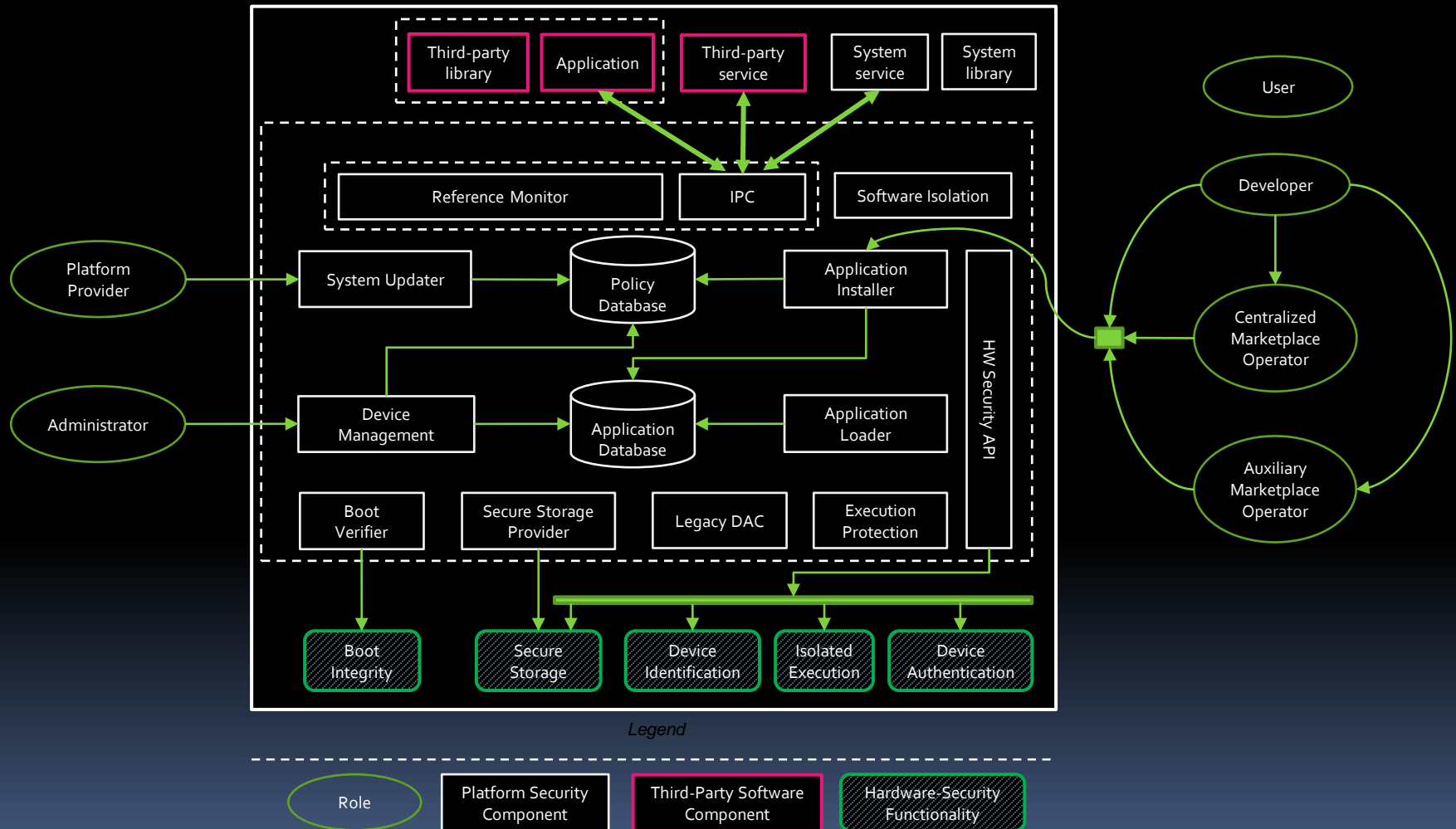


- Based on **dm-verity** kernel feature
- Calculates SHA256 hash over every 4K block of the system partition block device
 - Hash values stored in hash tree
 - Tree collapsed into a single root hash
- Hashes verified on-demand on disk access
- Signature of the root hash verified with public key included on the boot partition
 - Must be verified externally by the OEM

Verified Boot Hash Tree



Mobile Software platform security



Did you learn:

- Android as a software platform
 - Internals and surrounding ecosystem
- Security techniques in Android:
 - Application signing
 - Application isolation
 - Permission-based access control
 - Hardware-based security features