TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Operating Systems Group

DISTRIBUTED OPERATING SYSTEMS

MOBILE OPERATING SYSTEMS using Android as an example

https://tud.de/inf/os/studium/vorlesungen/dos

HORST SCHIRMEIER

Agenda

- Requirements
- Android Overview
- Security
- Memory
- Energy
- Summary

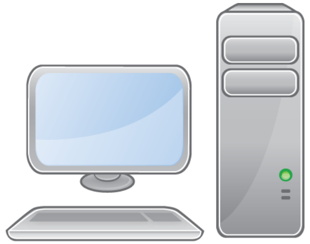
Tanenbaum, Chapter 10.8: Android

Agenda

- **Requirements**
- Android Overview
- Security
- Memory
- Energy
- Summary

Mobile General-Purpose Systems

… are different from classic desktop/server machines in both **application profile** and **hardware**.



Classic PC

- Few, trustworthy applications
- permanent power supply
- Rare communication via cable networks
- Lots of space for RAM, disks, cooling, etc.

Smartphone

- Changing, unknown apps from unknown vendors
- Battery operated
- Frequent communication via wireless networks (mobile)
- Strongly restricted space, only RAM and flash memories

Mobile General-Purpose *Operating* Systems

… therefore have to …

- **isolate** applications and their data better from the rest of the system
- **save memory** more aggressively
- make use of available hardware mechanisms to **save energy**, support energy-aware application behavior

Agenda

- Requirements
- **Android Overview**
- Security
- Memory
- Energy
- Summary

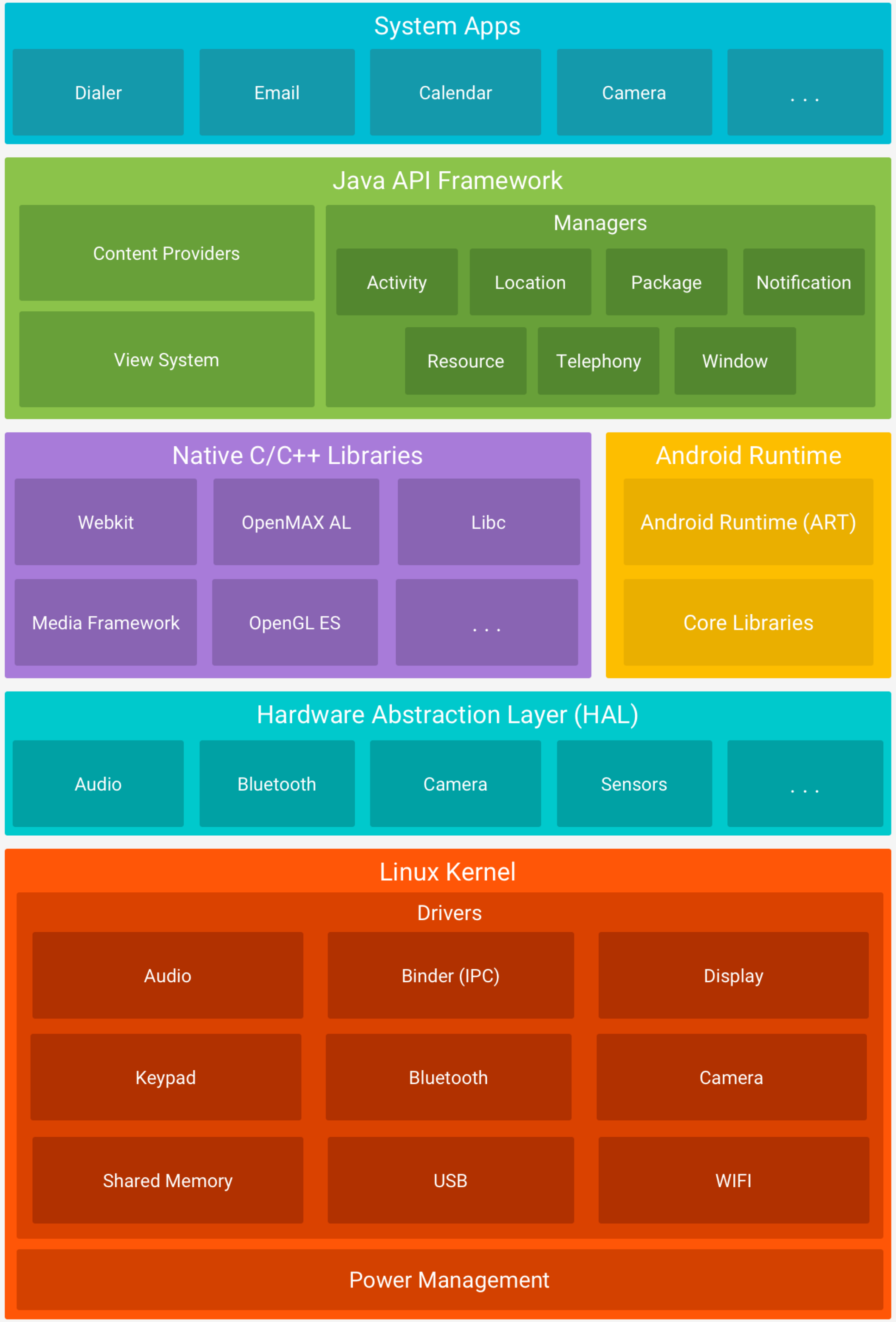
Android

- *Open Handset Alliance* (primarily Google), 2007
  - T-Mobile, Motorola, Samsung, …
- **Vision:**
  "… accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience."
- Infrastructure-software platform for smartphones
  - *Open Source*
- Many products available today
  - 2024: ~3 billion (10^9) Android devices, 71% market share

T-Mobile G1, 2008
- Android 1.6
- 256 MB RAM
- 528 MHz ARM 11
- 3,2" Display, 320x480 px

Architecture

- Linux plus Java – but different …



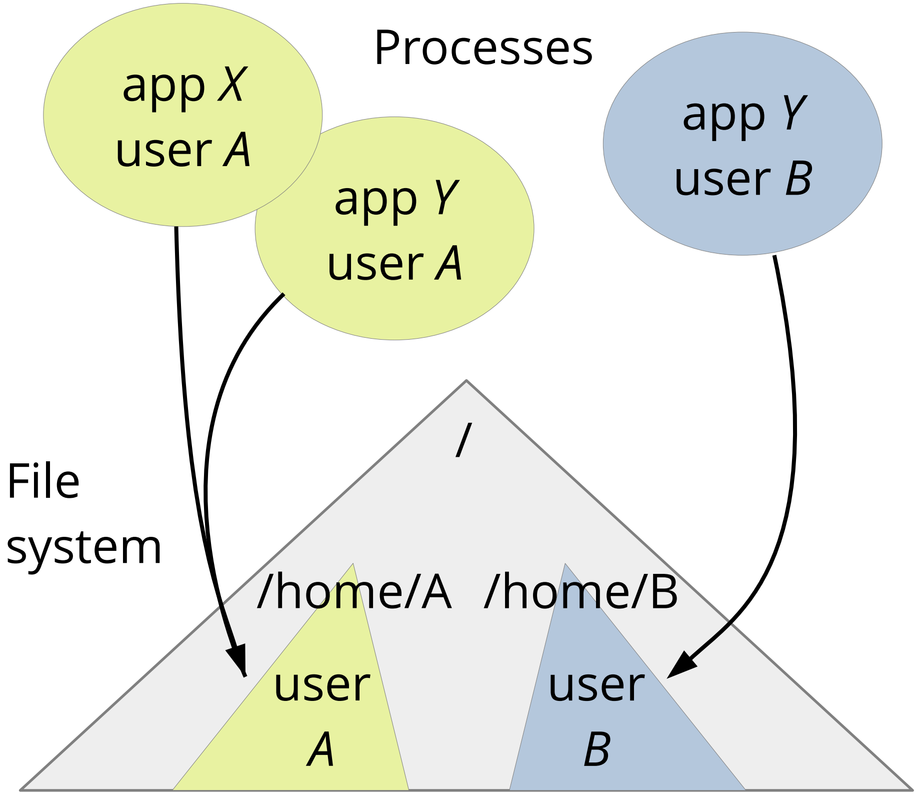
Source: https://developer.android.com/guide/platform

Agenda

- Requirements
- Android Overview
- **Security**
- Memory
- Energy
- Summary

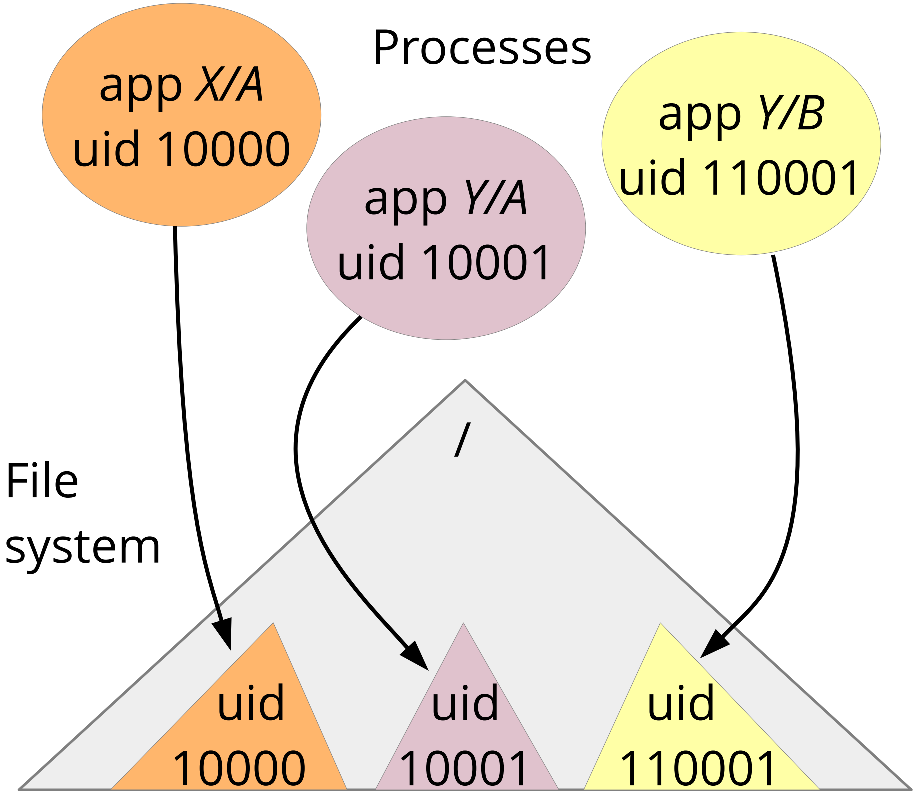
Repurposing Linux/Unix UIDs for *Sandboxing*

Regular **Linux**: UID per **user**



Buggy or malicious app jeopardizes **all user data**!

Android: UID per **app/user**

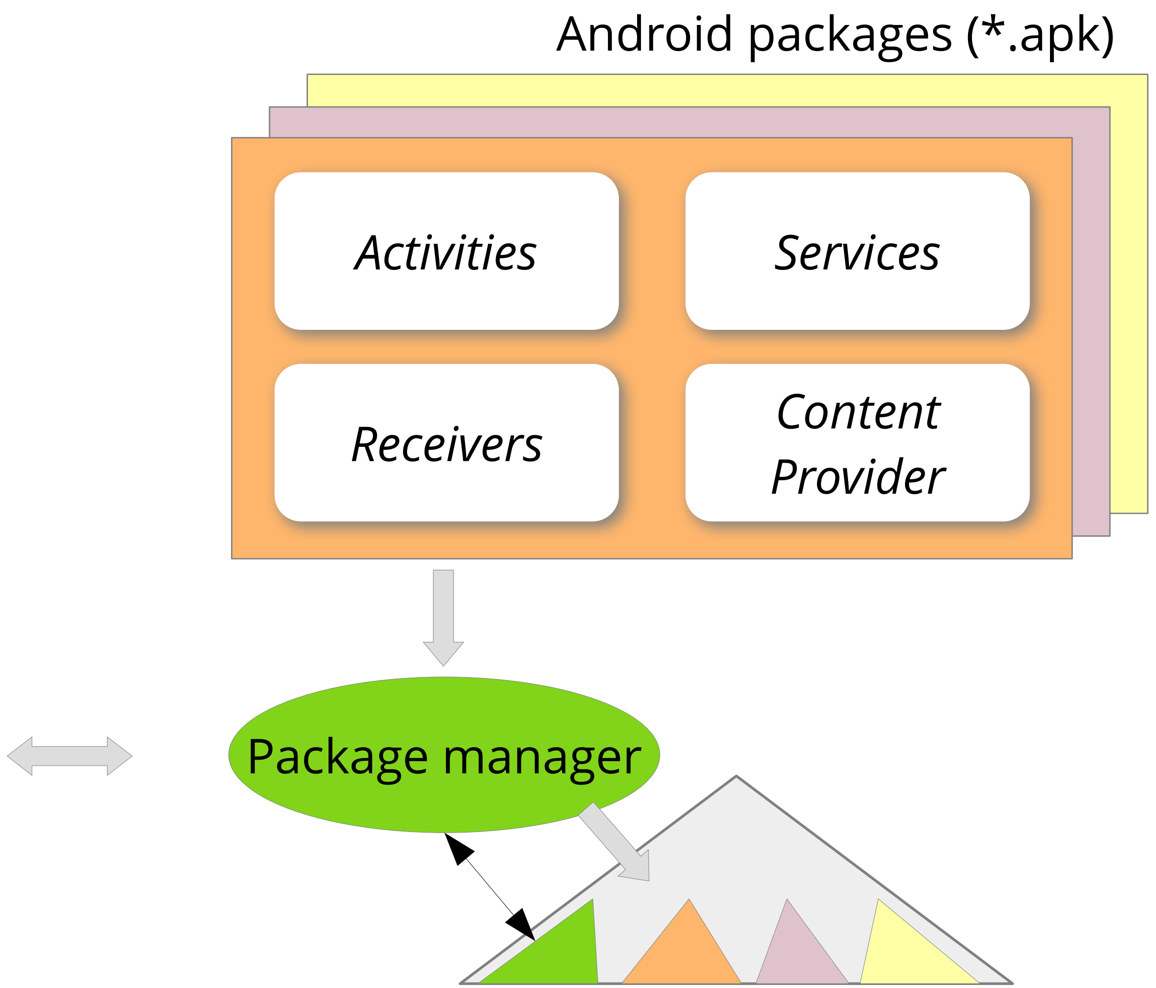


Every app instance (per user) has its **own data** in the file system.

UID Assignment: Package Manager

- automatically when installing apps

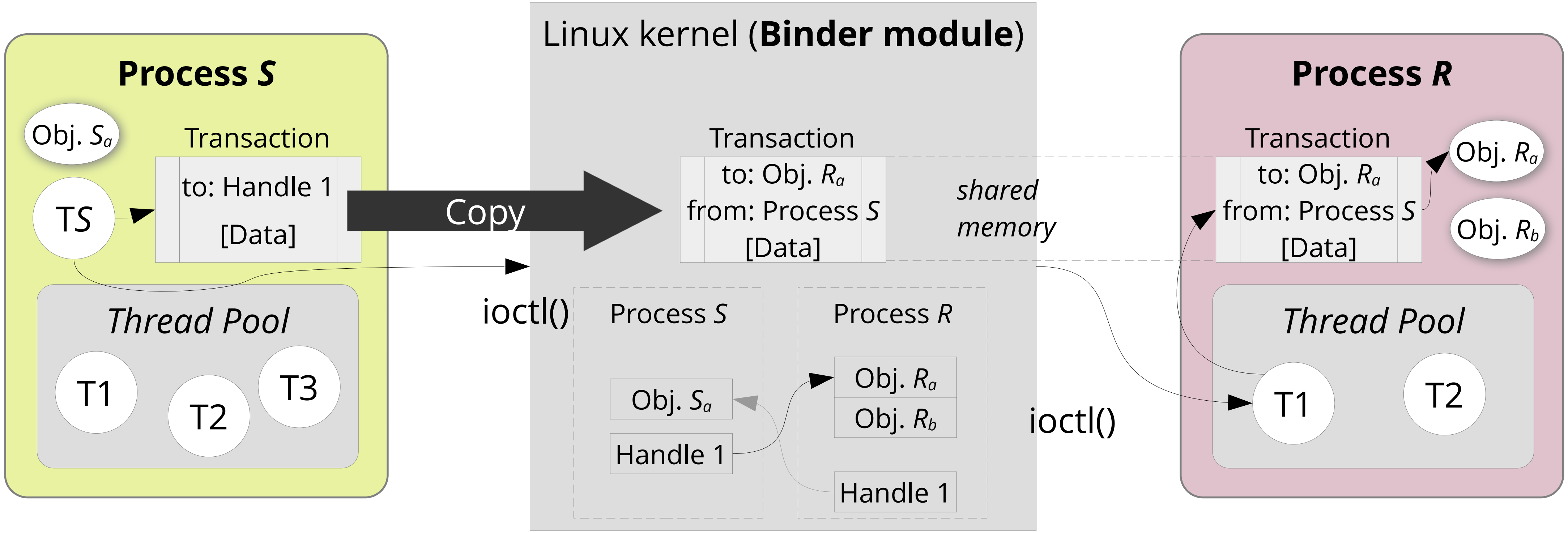
UID	Purpose
0	Root user
1000	Core-system service (system_server process)
1001	Telephony services
1013	(Low-level) media services
2000	Shell
10000–19999	Dynamically assigned application UIDs
100000+	Application UIDs user #2



App-Data Exchange: Binder IPC

"bottom-up" **explanation**

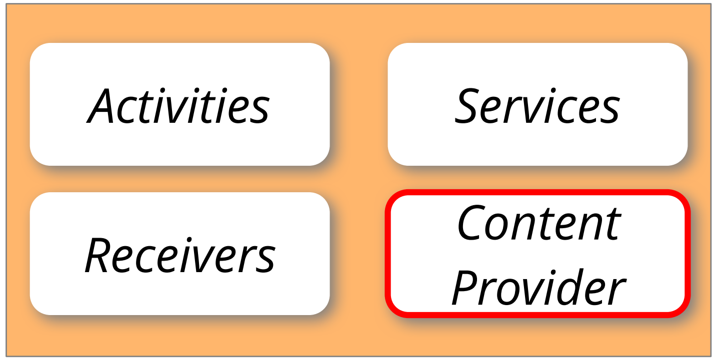
- Enables **object-oriented method calls** across process boundaries
  - Not possible with existing Linux abstractions



Handles: **simple integers (like file descriptors)**
→ IPC permissions / *Capabilities*

The Binder module automatically replaces an object reference in the [Data] part by a newly created handle.

App-Data Exchange: Content Provider



- Class within an app
- Provides content via Binder IPC based on a URI:

content://com.example.k8mail.provider.email/messages/1

Provider **identification** ("authority") — com.example.k8mail.provider.email

Path for provider — /messages/1

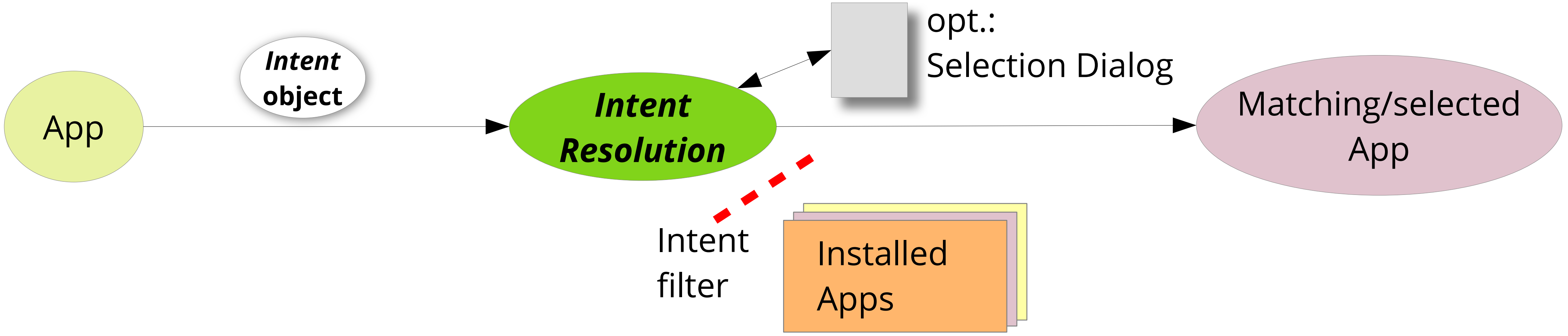
- Content requests via *Remote Procedure Call*, e.g.:

```
query("content://com.example.k8mail.provider.email/messages");
```

How to determine and start the (correct) email app? → *Intents*

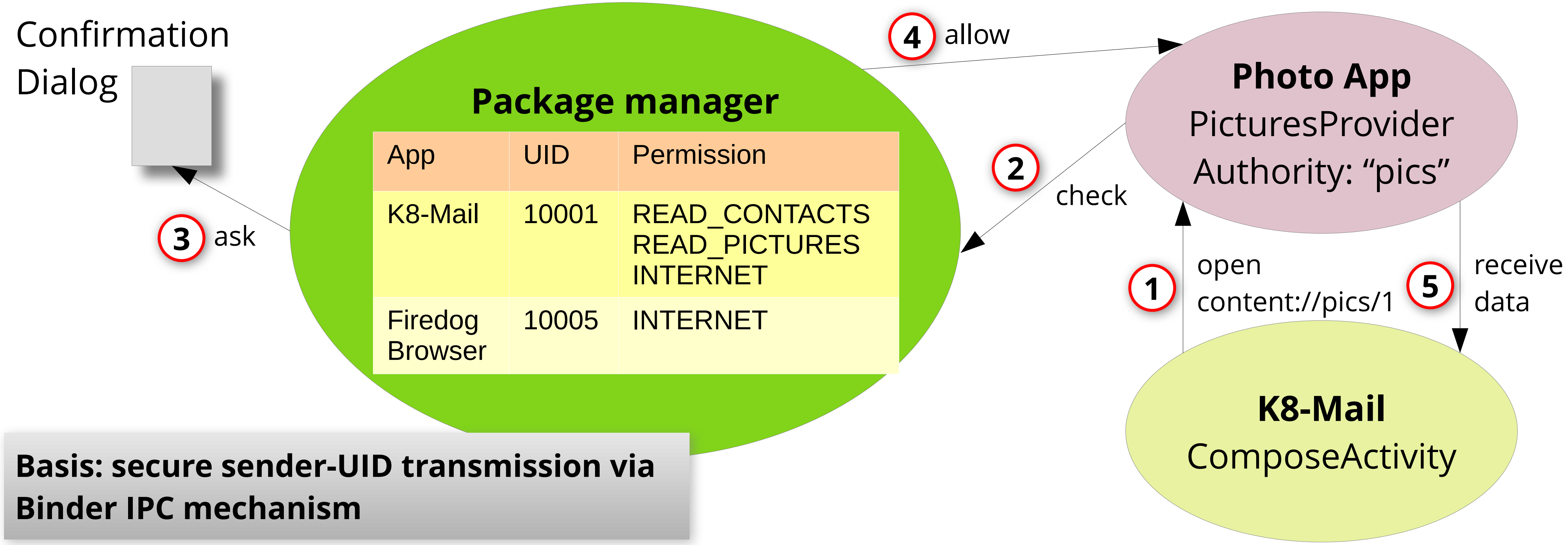
App-Data Exchange: Intents

- Object to describe abstract “intent” the app does not implement by itself, e.g.
  - Send email, display website, make a phone call, ...
- or **system events** that a (system) app should handle
  - low battery, incoming call, ...



App Permissions

- Pre-defined in apk manifest
- Assigned to app after user confirmation
- Management in package manager



App	UID	Permission
K8-Mail	10001	READ_CONTACTS READ_PICTURES INTERNET
Firedog Browser	10005	INTERNET

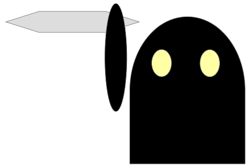
Basis: secure sender-UID transmission via Binder IPC mechanism

Agenda

- Requirements
- Android Overview
- Security
- **Memory**
- Energy
- Summary

Many Apps on Limited Memory

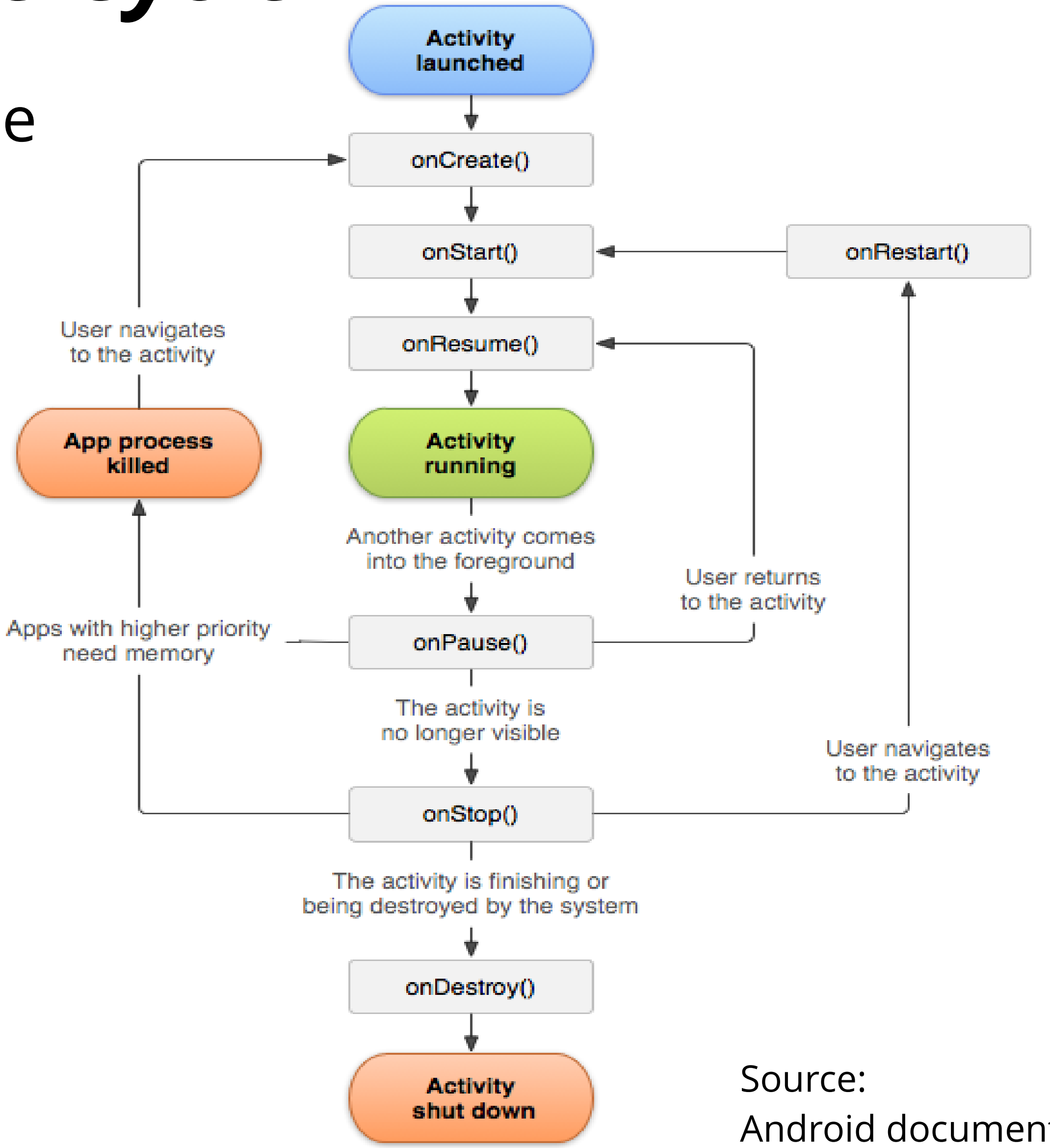
- (Comparably) small RAM on Android systems
- Continuous *paging* reduces performance, and limits storage life time (flash-memory technology)
  - Typical NAND-flash wearout: 100,000 to 1,000,000 write cycles
- **Solution:** Stop and restart app ***activities*** at **any time!**
  - ***Out of Memory Killer*** continuously scans for victims
  - Priority: System processes, background services, current foreground activity and visible activities are **chosen last**



Android *Activity Life Cycle*

- Apps have to be in the same state after kill+restart
  - System: Views (GUI layout)
  - App: other state

Result: App survives its process!

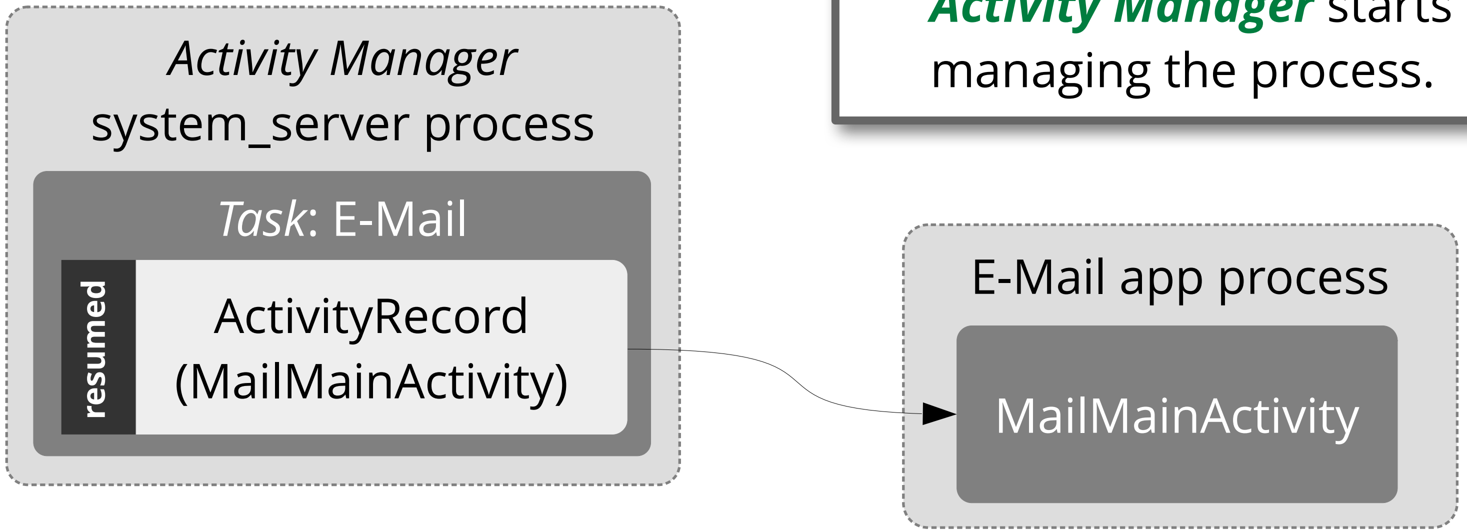


Source:
Android documentation

The Activity Manager (1)

- … manages all information about **running** apps

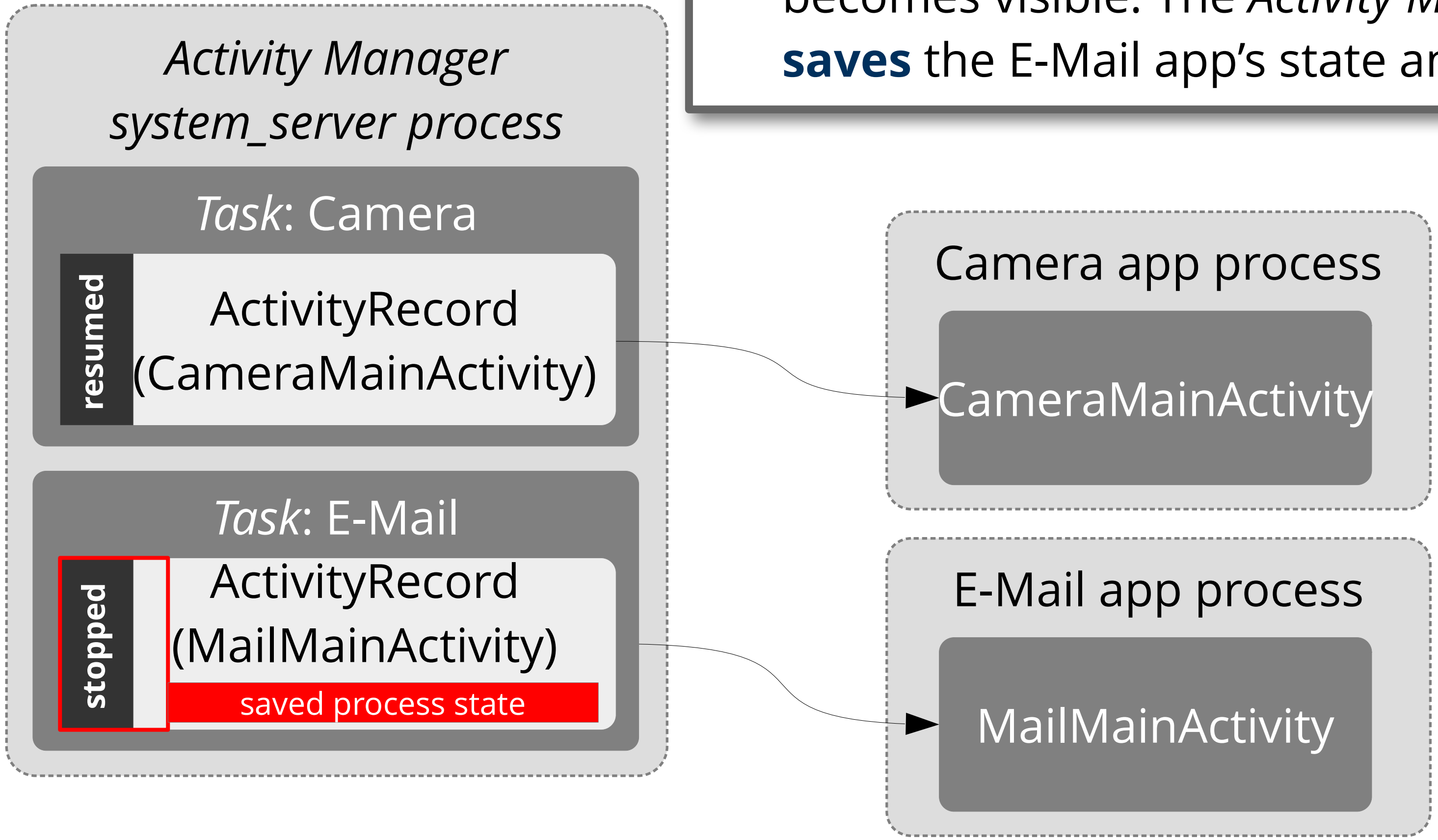
1) User **starts** e-mail app. The ***Activity Manager*** starts managing the process.



The Activity Manager (2)

- … manages all information about **running** apps

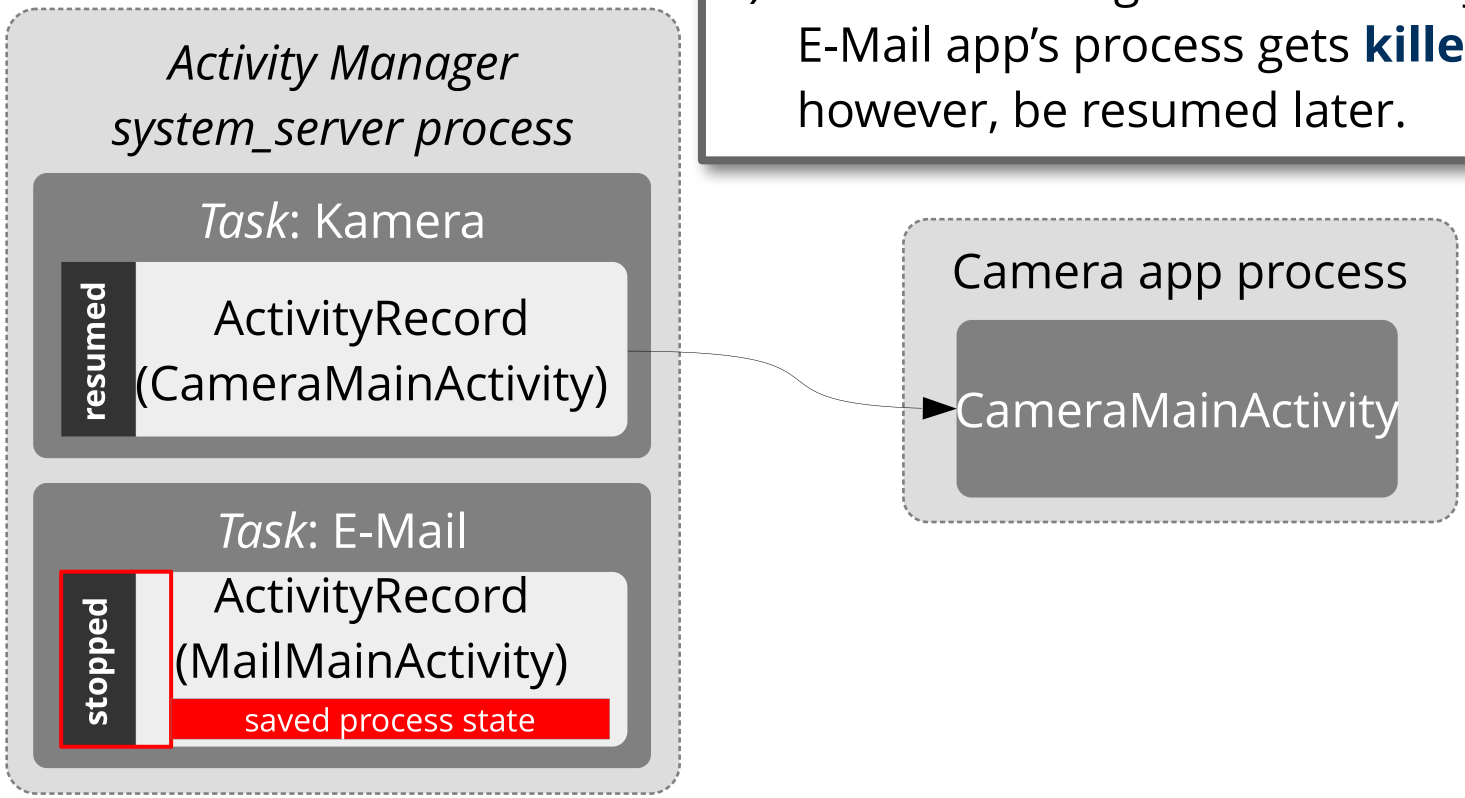
2) The user starts the Camera app, which becomes visible. The *Activity Manager* **saves** the E-Mail app's state and **stops** it.



The Activity Manager (3)

- … manages all information about **running** apps

3) We are running out of memory, and the E-Mail app’s process gets **killed**. It can, however, be resumed later.



Agenda

- Requirements
- Android Overview
- Security
- Memory
- **Energy**
- Summary

Energy Consumption: Basics

- Energy is not really “consumed” but **transformed into heat** and radiated
  - usually unused/lost
  - undesirable (possibly cooling necessary)
  - reduces battery runtime
- Fundamental physical equations that relate energy and **electrical current**:
  - **$E = P \cdot \Delta t$** (Energy[J] = Power[W] • Time[s])
  - **$P = V \cdot I$** (Power[W] = Voltage[V] • Current[A])

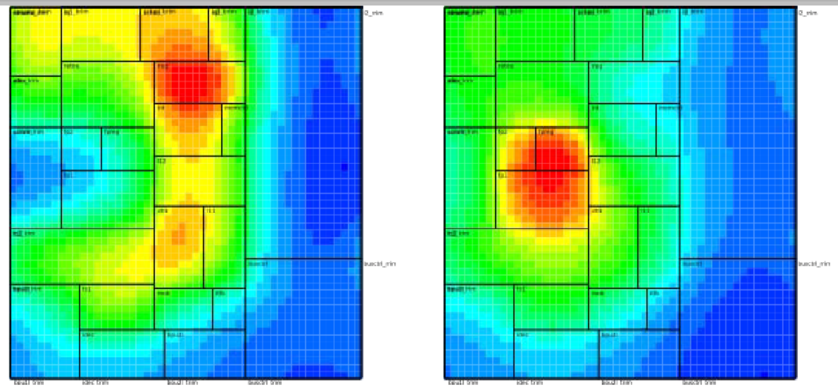
Power Dissipation in CMOS Semiconductors

… primarily two components: dynamic+static power

- $P_{dyn} = \alpha \cdot C_L \cdot U^2 \cdot f$

U:	Supply voltage
f:	Clock frequency
C_L:	Switched capacity
α:	Fraction of switching transistors

Model: Many **small capacities** (wires, memory) are charged and discharged **when switching**.



Source: Frank Bellosa, Karlsruher Institut für Technologie

- $P_{stat} = U \cdot I_{leak}$

I_{leak}: **Leak current**

Cause: **Quantum effects** in semiconductor material, worsening exponentially with shrinking structure sizes.

→ Leak current is very small, but **flows always** as long as the system is powered.

Energy-Saving Approaches

Bold-print: concerns the OS

$$P_{dyn} = \alpha \cdot C_L \cdot U^2 \cdot f$$

- **Absenken von Spannung <u>und</u> Takt** (*Dynamic Voltage and Frequency Scaling*)
  - → lohnt durch das Quadrat bei der Spannung
  - → geht, wenn die CPU unterausgelastet ist oder sowieso immer wieder auf Speicher wartet
- **Abschalten des Taktes**
  - → P_{dyn} = 0, aber Zustand bleibt erhalten!
- Verbesserte Fertigungstechnologie (kleiner)
- Effizientere Rechnerarchitektur

$$P_{stat} = U \cdot I_{leak}$$

- Bessere Fertigungstechnologie oder größere Strukturen
- **Abschalten von ungenutzen Komponenten**

Problem 1: Verlustleistung im Leerlauf

- Ein ruhendes System muss abgeschaltet werden!
  - Hier: Messungen am Openmoko Neo FreeRunner, Quelle: [1]

Faktor 10 im Vergleich zum schwach ausgelasteten Prozessor!

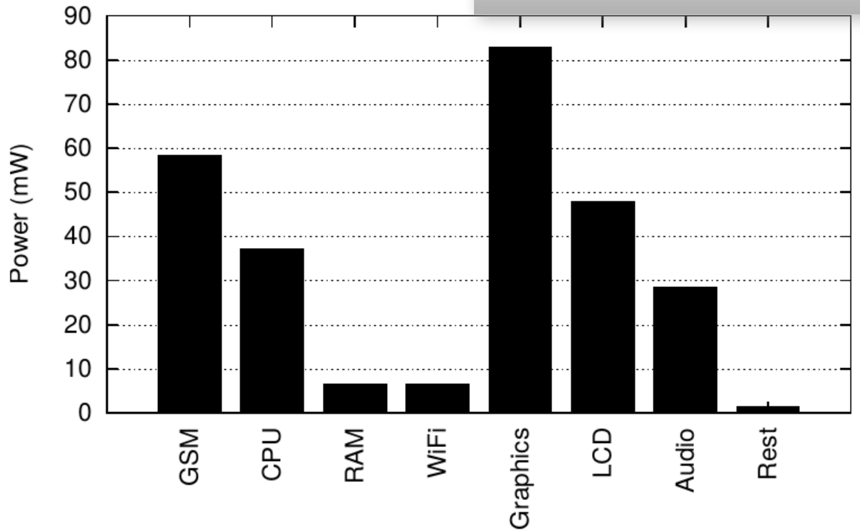


Figure 3: Average power consumption while in the idle state with backlight off. Aggregate power is 268.8 mW.

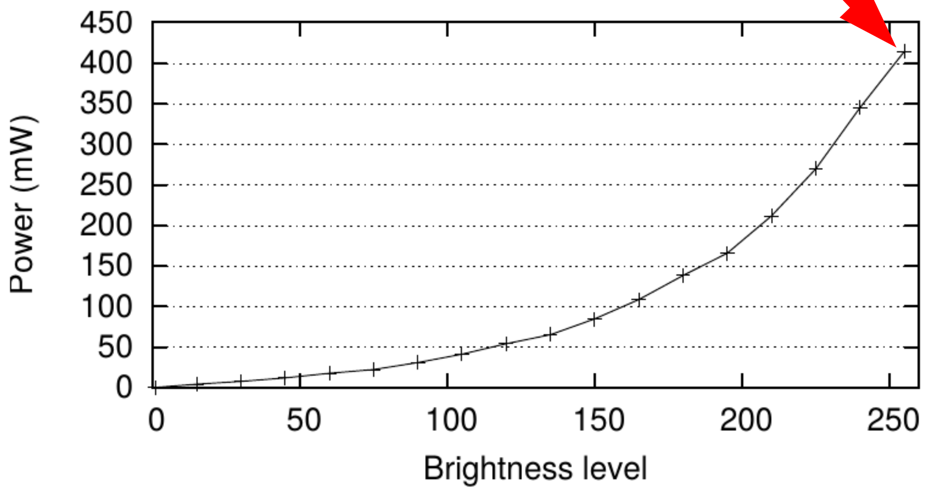


Figure 4: Display backlight power for varying brightness levels.

Im SUSPENDED-Zustand (div. Komponenten abgeschaltet) sind es nur 68,6 mW.

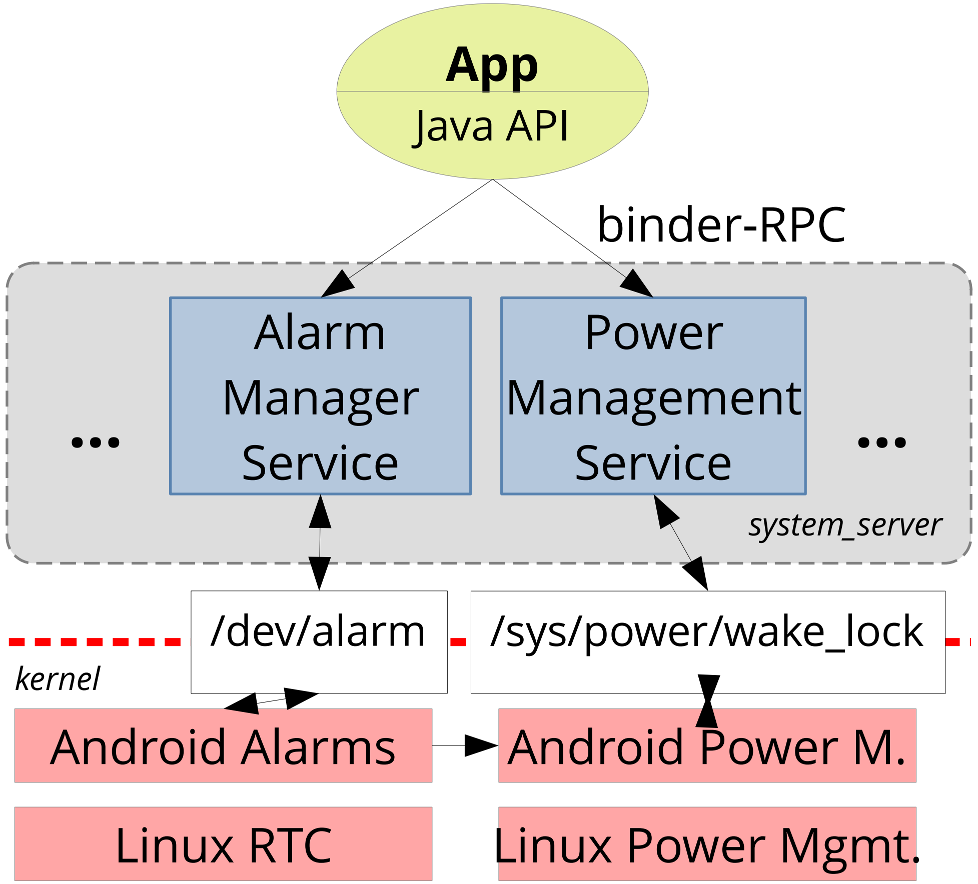
TECHNISCHE UNIVERSITÄT DRESDEN

Lösung 1: *Wake Locks* und *Alarms*

- **Ansatz**: So schnell und oft wie möglich alle untätigen Komponenten abschalten: Schlafmodus (SUSPENDED)

- Apps müssen …
  - dies ggf. explizit verhindern: ***Wakelocks***
  - zeitgesteuert das System aufwecken können: ***Alarms***

- Dazu sind Erweiterungen am Linux-Kernel nötig



Wakelocks

- Apps benötigen das Recht `android.permission.WAKE_LOCK`, um einen *Wakelock* zu erzeugen.
- *Wakelock*-Typen:

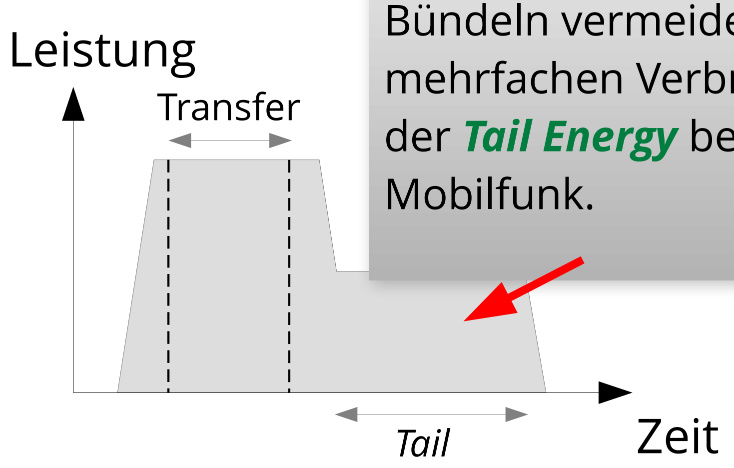
Name	**Auswirkung** (laut Android-Dokumentation)
FULL_WAKE_LOCK	*Ensures that the screen and keyboard backlight are on at full brightness.*
PARTIAL_WAKE_LOCK	*Ensures that the CPU is running; the screen and keyboard backlight will be allowed to go off.*
PROXIMITY_SCREEN_OFF_WAKE_LOCK	*Turns the screen off when the proximity sensor activates.*
SCREEN_BRIGHT_WAKE_LOCK	*Ensures that the screen is on at full brightness; the keyboard backlight will be allowed to go off.*
SCREEN_DIM_WAKE_LOCK	*Ensures that the screen is on (but may be dimmed); the keyboard backlight will be allowed to go off.*

Diskussion: *Wakelocks*

- Anfangs haben sich die Linux-Kernelentwickler gesträubt, *Wakelocks* zu integrieren.
- **Problem**: Der Mechanismus ist nicht an die Existenz des Prozesses gekoppelt.
  - Falls „vergessen" wird, den *Lock* aufzuheben, bleibt das Gerät an!
  - Android-Apps überleben ihren Prozess → *Activities*, *Receivers*, …
- **Lösung**? Inzwischen wurde das Konzept unter dem Namen ***„Suspend Blockers"*** aufgenommen.
  - Zugriff auf /sys/power/wake_lock und …/wake_unlock nur für root.
  - Das Feature ist optional.
- Auch unter Android ist die direkte Nutzung zu vermeiden.
  - Stattdessen Kopplung an Aktivitätsverwaltung und UI

Alarms

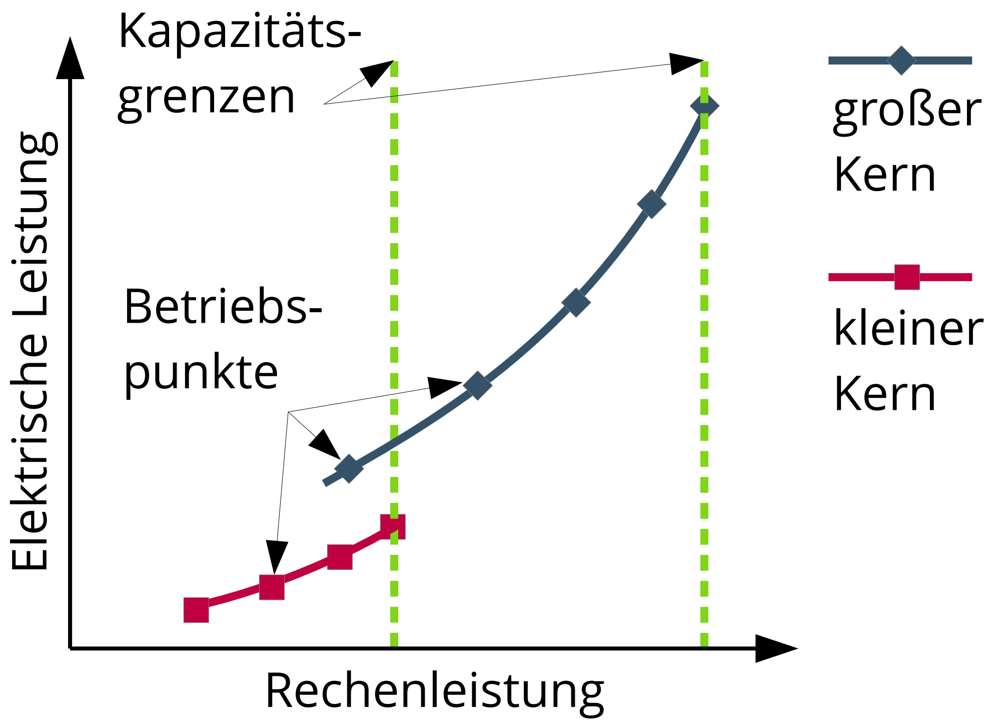
- Zeitgesteuertes Wecken des Systems
  - Auslösung eines *Intents*
  - Funktioniert auch, wenn die App, die den Alarm angefordert hat, nicht mehr aktiv ist!
  - Während der Behandlung hält der Alarm-Manager ein *Wakelock*
- Anwendungen: Abholen von Emails, Wettervorhersage, …
- Alarmzeitpunkte können vom System **verschoben** werden, um Kommunikationsvorgänge zu bündeln. Spart Energie!



Bündeln vermeidet den mehrfachen Verbrauch der ***Tail Energy*** beim Mobilfunk.

Problem 2: Takt-/Spannungssteuerung

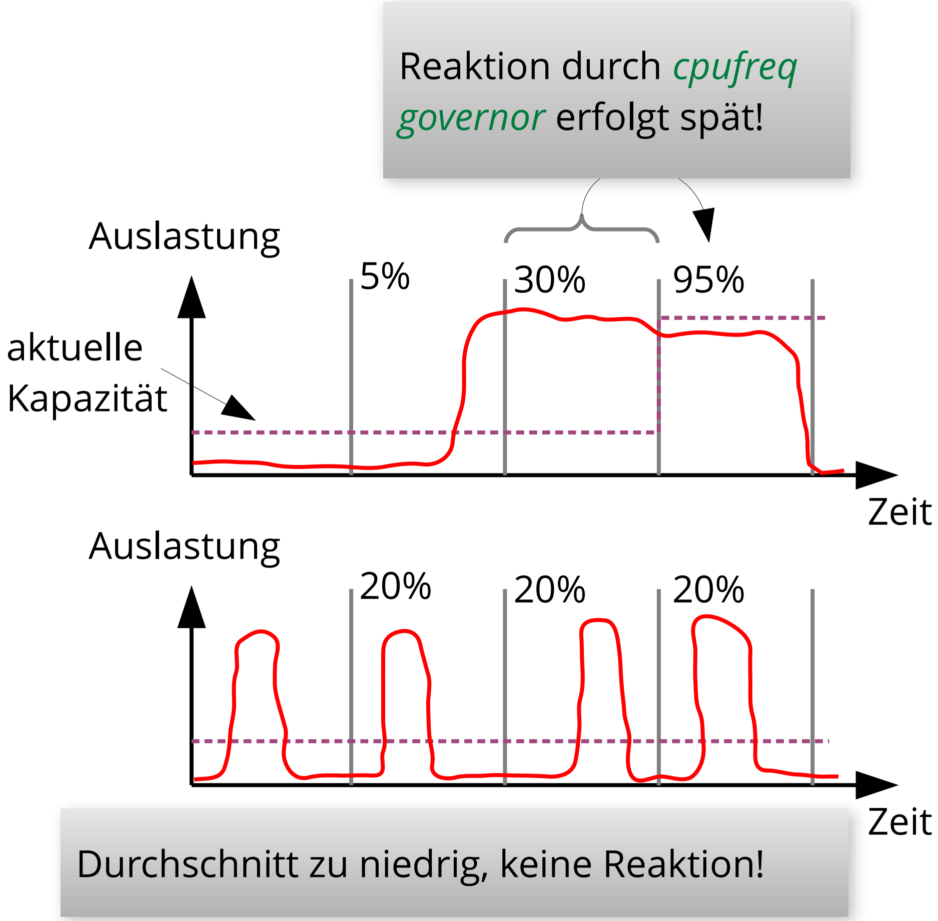
- Ziel: Maximale Rechenleistung bei minimalem Energieverbrauch
- Moderne **heterogene** Multicore-Prozessoren bieten diverse **Betriebspunkte**.
  - Typisch: ARMs big.LITTLE-Architektur mit 4 großen und 4 kleinen Kernen



Das *Scheduling*-Problem wird um energiegewahre Taktsteuerung erweitert!

Lösung 2(a): Lastabhängige Regelung

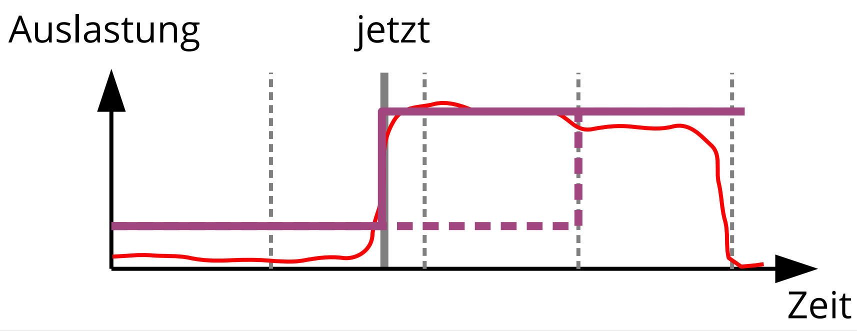
- **Normales Linux**: Die Auslastung der CPU-Kerne wird in regelmäßigen Abständen betrachtet. Schwellwertabhängig wird der Takt angepasst.
- **Schwäche**: Langsame Reaktion
- **Ursachen**:
  - Die Taktregelung weiß nicht so viel wie der *Scheduler* über die **zukünftige** Last
  - Keine Information über die Energieeffizienz der jeweiligen Betriebspunkte.

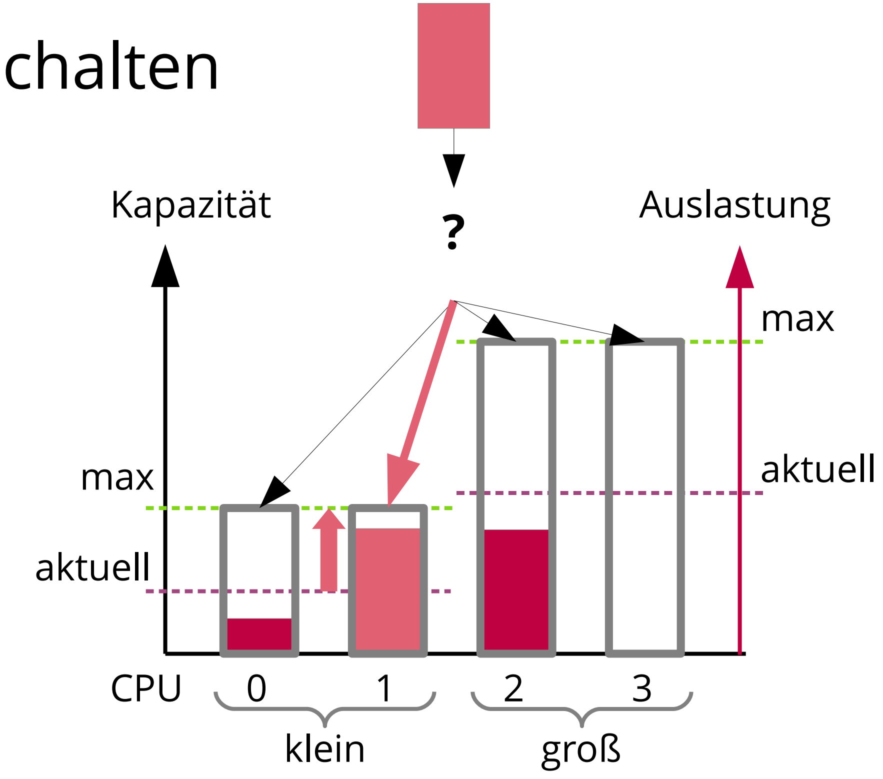


Lösung 2(b): Android EAS

"Energy-Aware Scheduling"

- basiert auf Energiemodell (über Betriebspunkte)
- entscheidet, wo ein Task wenig Energie erfordert
- kann Kerne komplett von Last befreien und abschalten
- Reaktion auf Laständerung erfolgt sofort!





Agenda

- Requirements
- Android Overview
- Security
- Memory
- Energy
- **Summary**

Summary

- The development of Android pushed innovations in Linux
  - *Application Sandboxing*
  - More control of power-off and wake-up
  - Energy-aware *Scheduling*
- Partially repurposing of Linux concepts
  - UIDs per app
  - Applications run longer than their processes

Literatur

[1] Aaron Carroll and Gernot Heiser. 2010. *An analysis of power consumption in a smartphone*. In Proceedings of the 2010 USENIX conference on USENIX annual technical conference (USENIX ATC '10). USENIX Association, USA, 21.