Technische Universität Dresden

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

ADVANCED OPERATING SYSTEMS

MOBILE OPERATING SYSTEMS *using Android as an example*

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

HORST SCHIRMEIER

Agenda

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

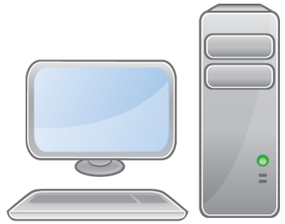
Tanenbaum, Chapter 10.8: Android

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:**

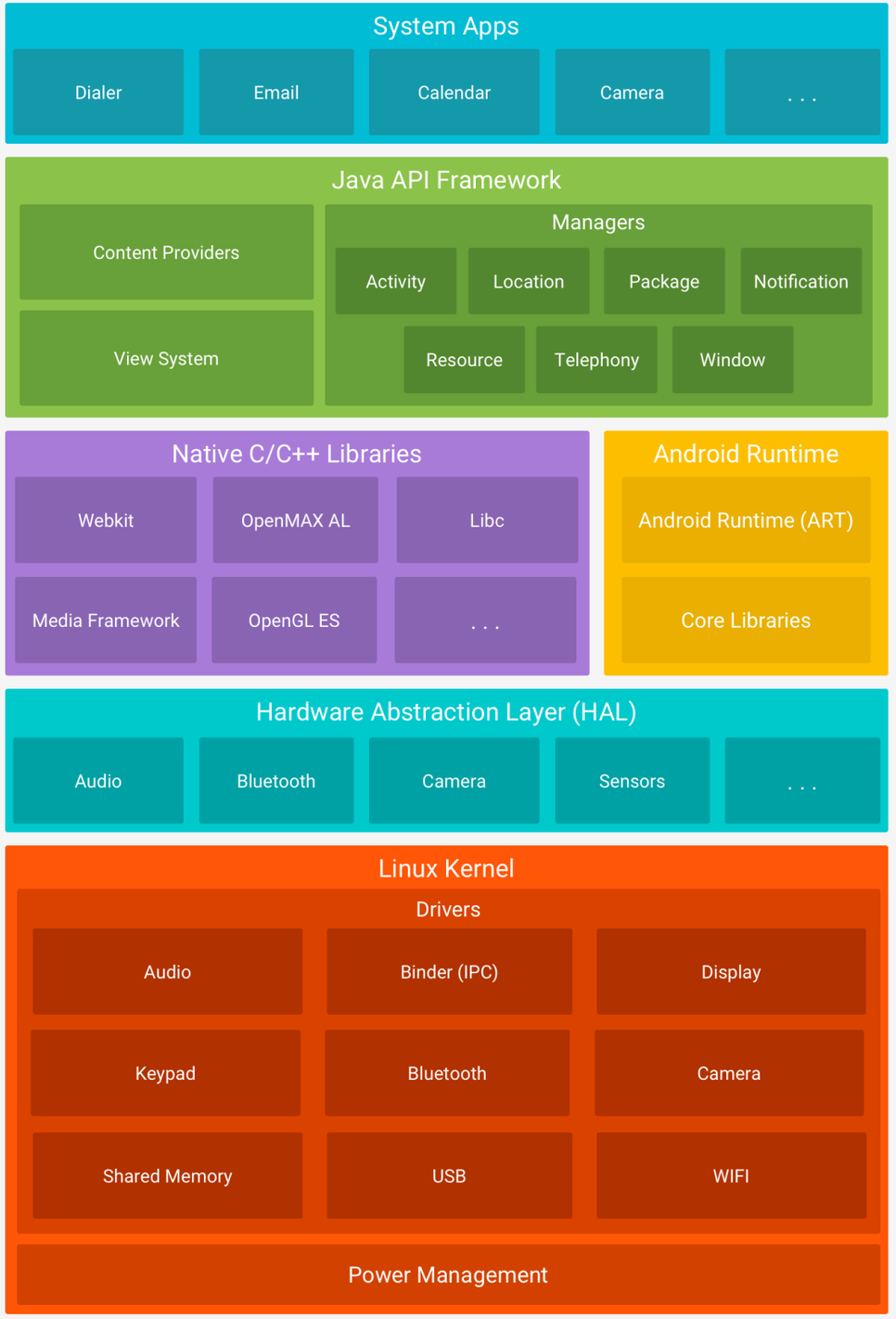
  "… *accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience.*"

- Infrastructure-software platform for smartphones
  - *Open Source*
- Many products available today
  - 2026: ~4 billion (10^9) Android devices, 70–75% 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* …



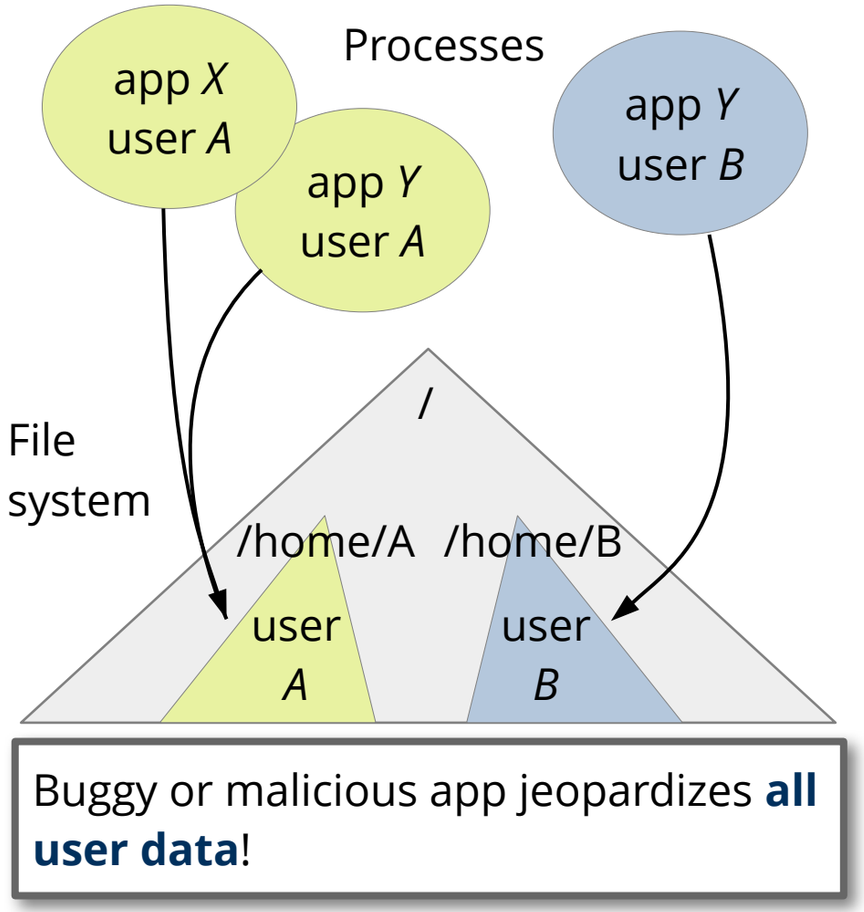
Source: https://developer.android.com/guide/platform

Agenda

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

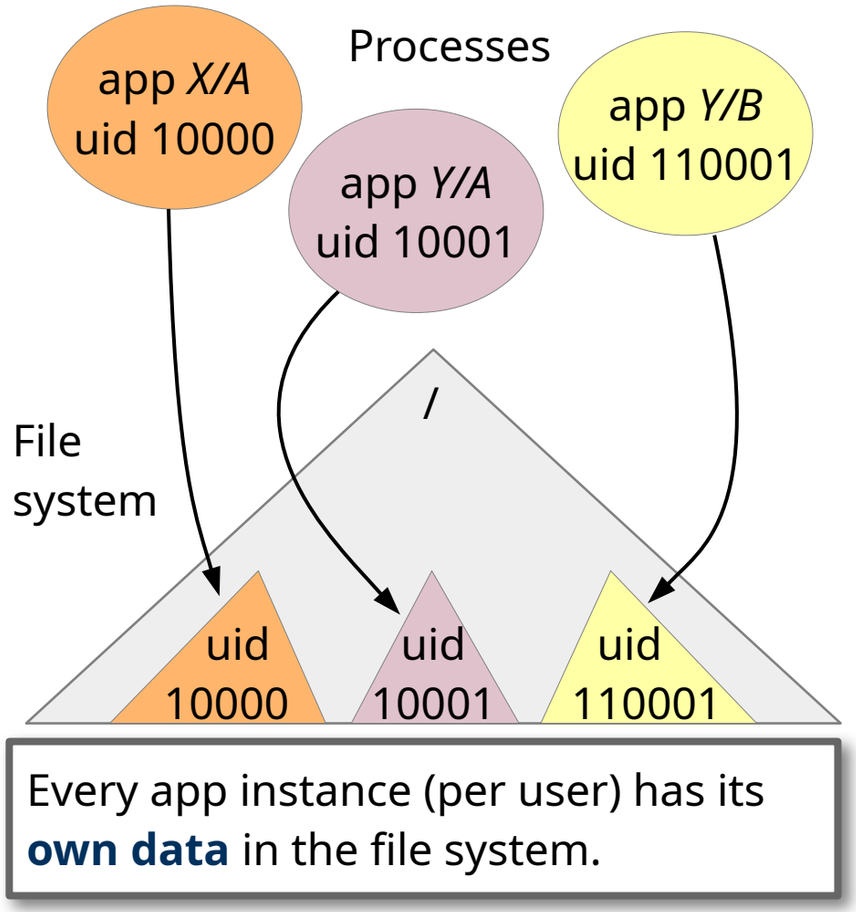
Repurposing Linux/Unix UIDs for *Sandboxing*

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



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

Android: UID per **app/user**

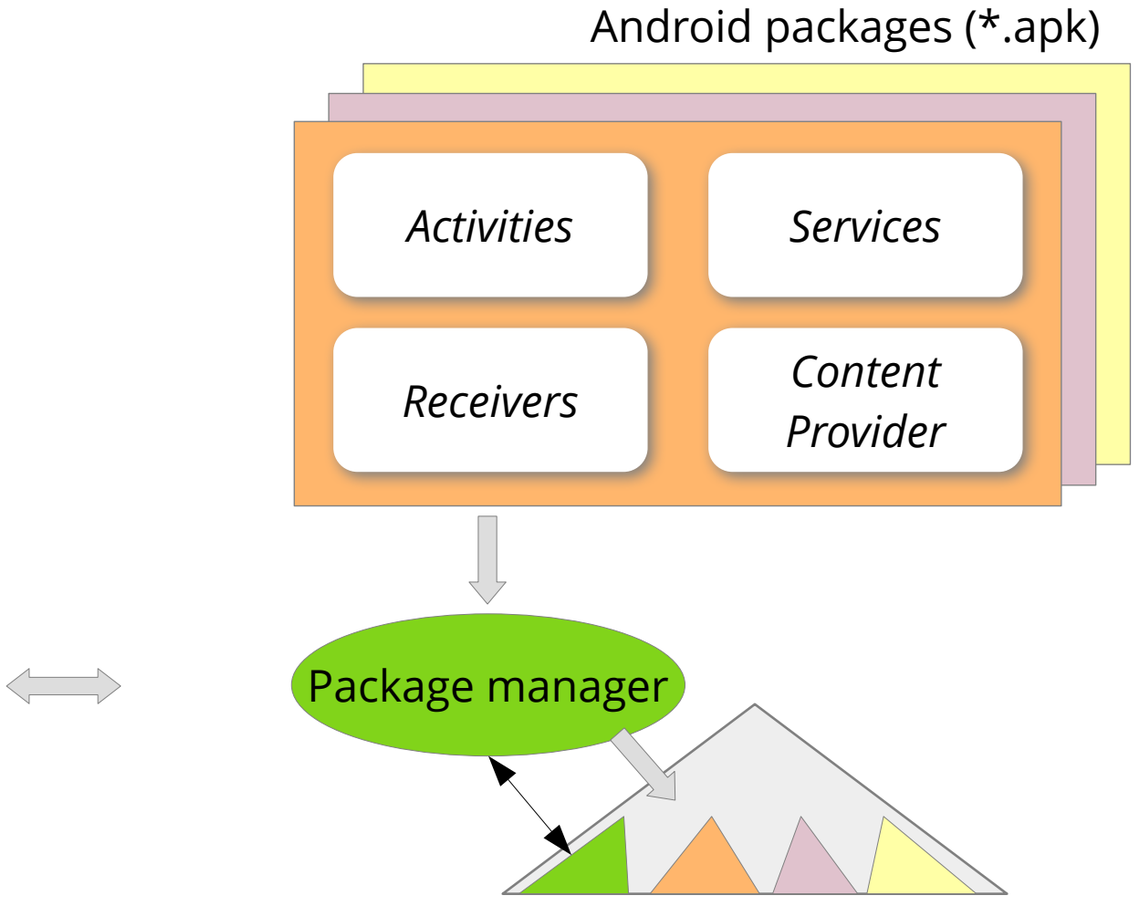


Every app instance (per user) has its **own data** in the file system.

UID Assignment: Package Manager

- automatically when installing apps

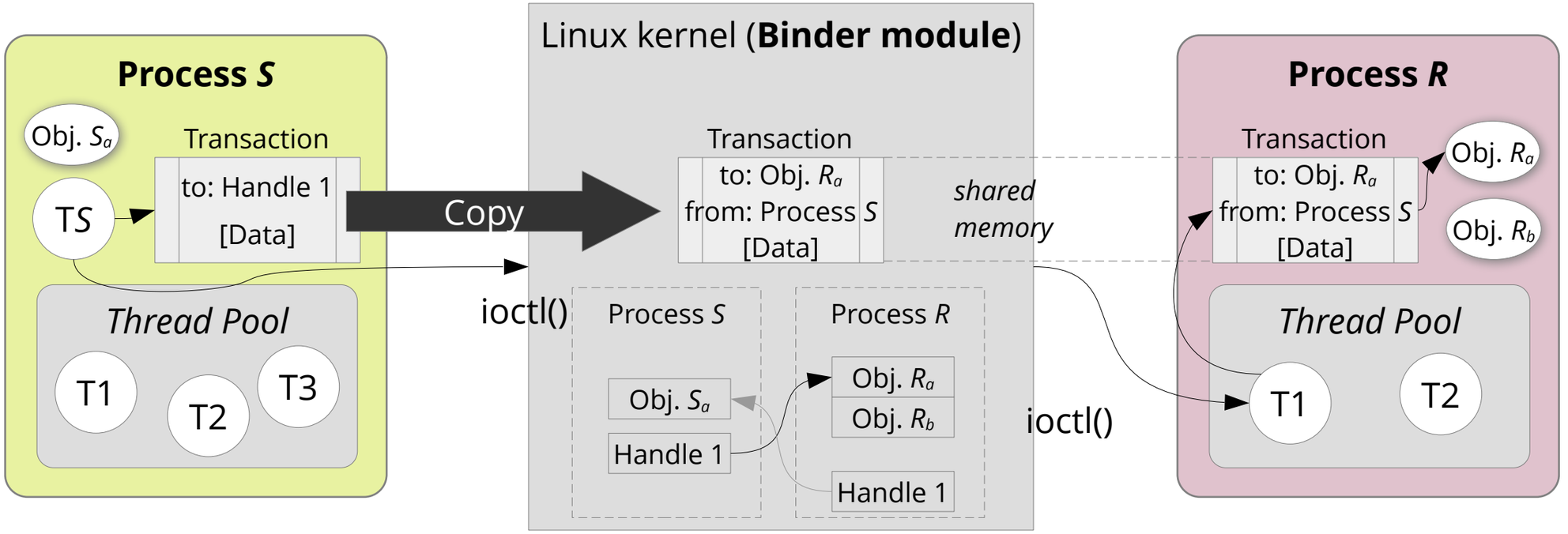
UID	Purpose
0	Root user
1000	Core-system service (system_server process)
1001	Telephony services
1013	(Low-level) media services
2000	Shell
10000–19999	Dynamically assigned application UIDs
100000+	Application UIDs user #2



App-Data Exchange: Binder IPC

***"bottom-up"* explanation**

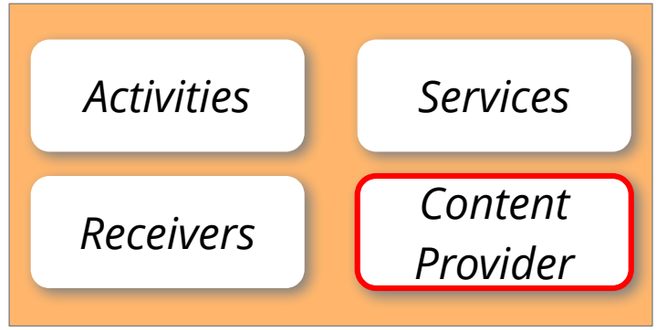
- Enables **object-oriented method calls** across process boundaries
  - Not possible with existing Linux abstractions



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”) **Path** for provider

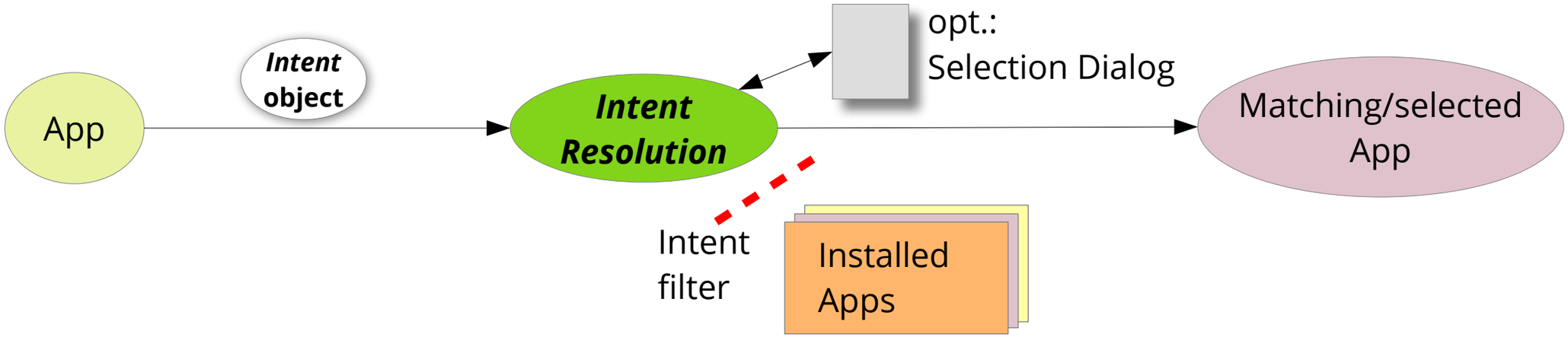
- Content requests via *Remote Procedure Call*, e.g.:

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

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

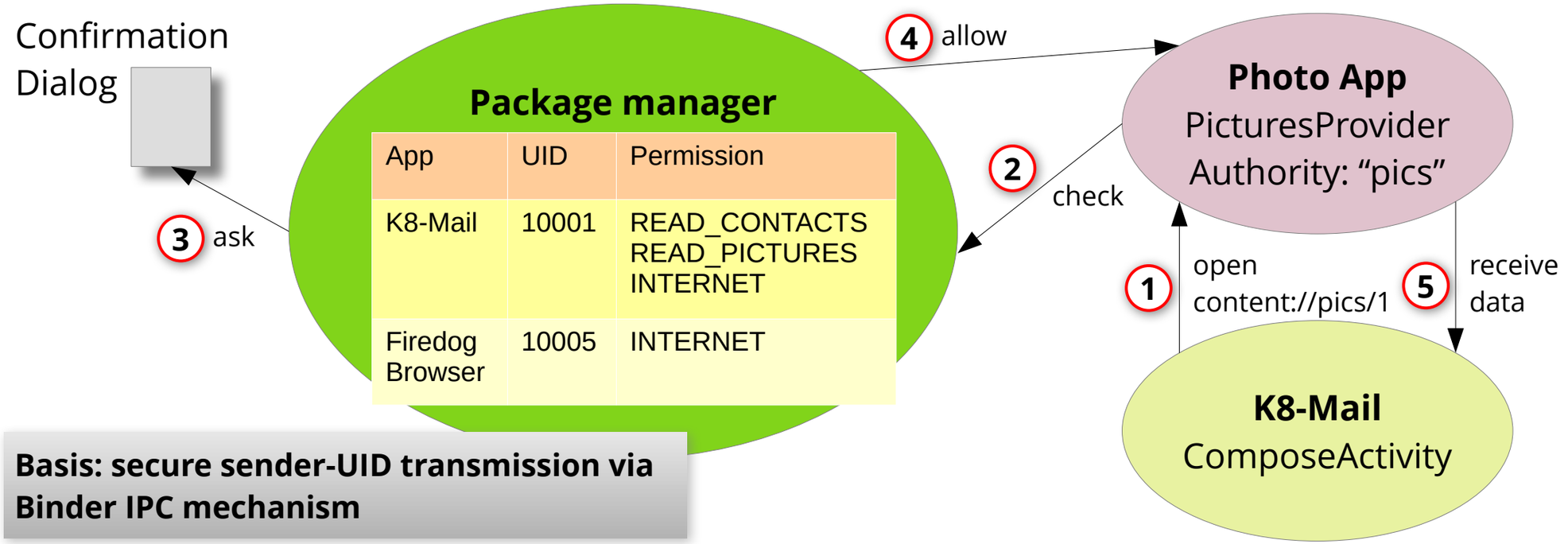
App-Data Exchange: Intents

- 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

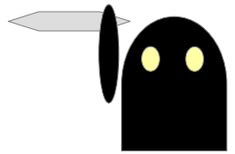


Agenda

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

Many Apps on Limited Memory

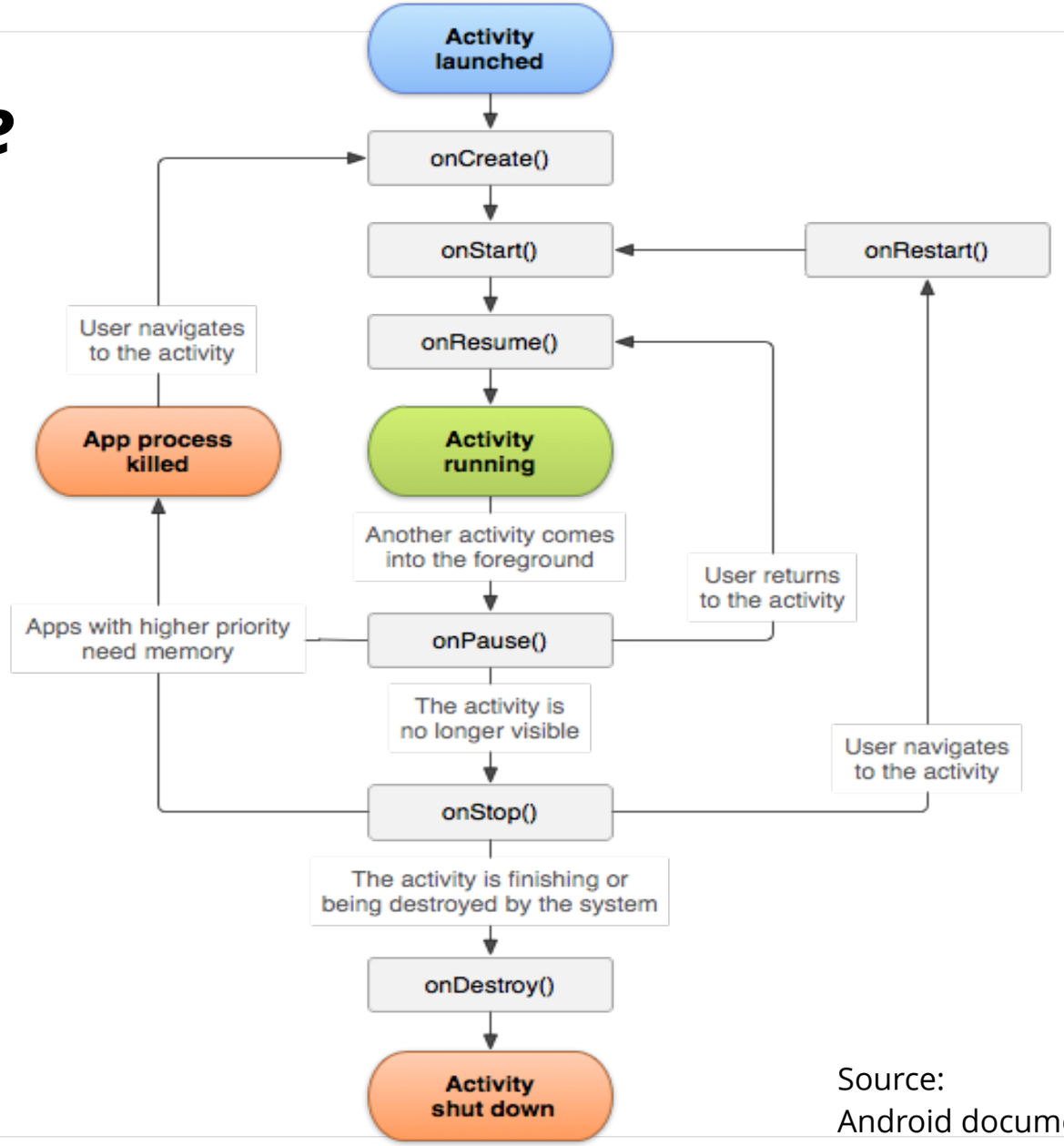
- (Comparably) small RAM on Android systems
- Continuous *paging* reduces performance, and limits storage life time (flash-memory technology)
  - Typical NAND-flash wearout: 100,000 to 1,000,000 write cycles
- **Solution:** Stop and restart app ***activities*** at **any time!**
  - ***Out of Memory Killer*** 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

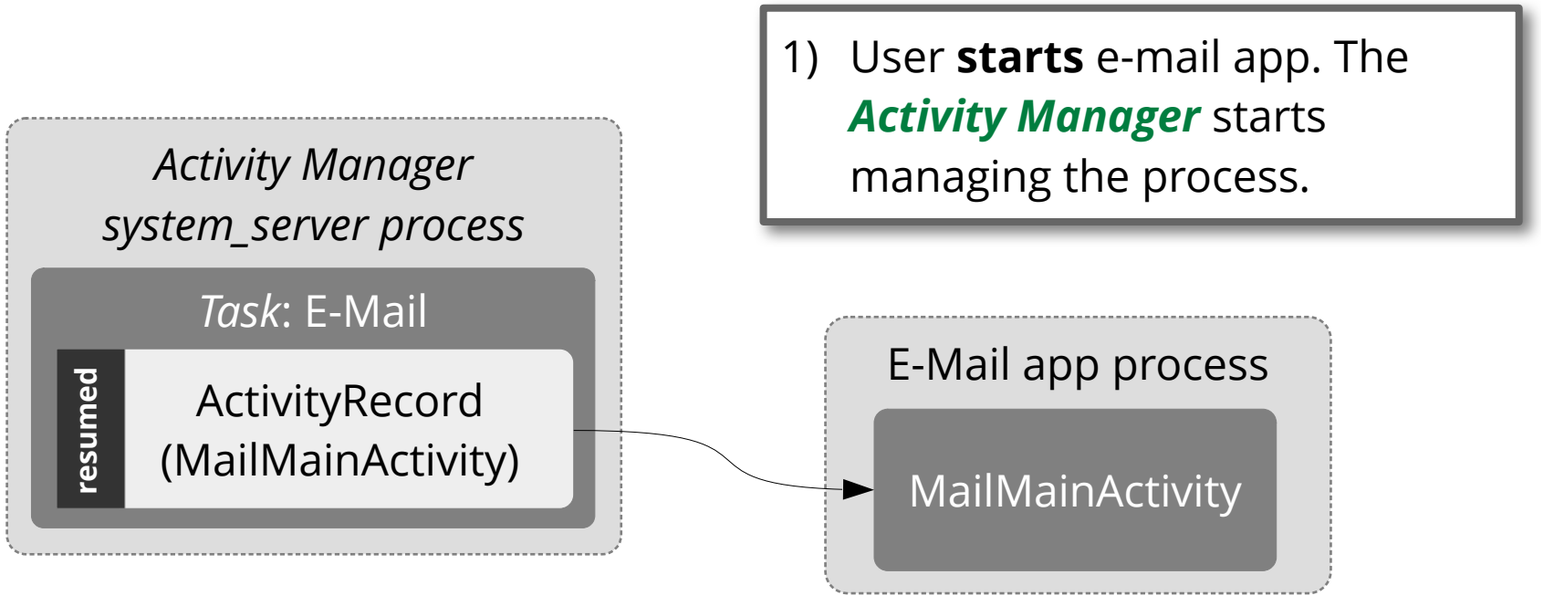
Result:
App survives its process!



Source:
Android documentation

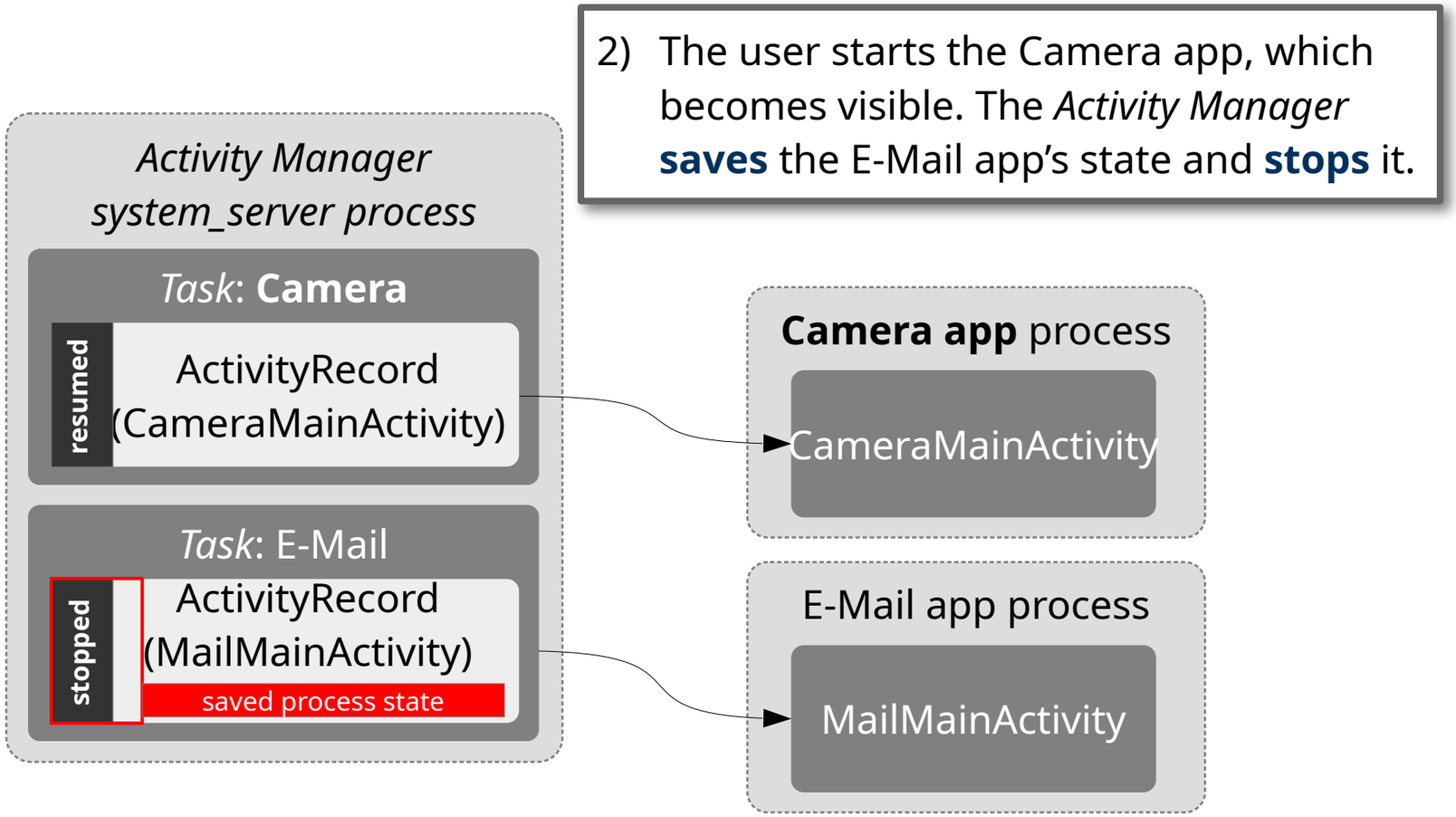
The Activity Manager (1)

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



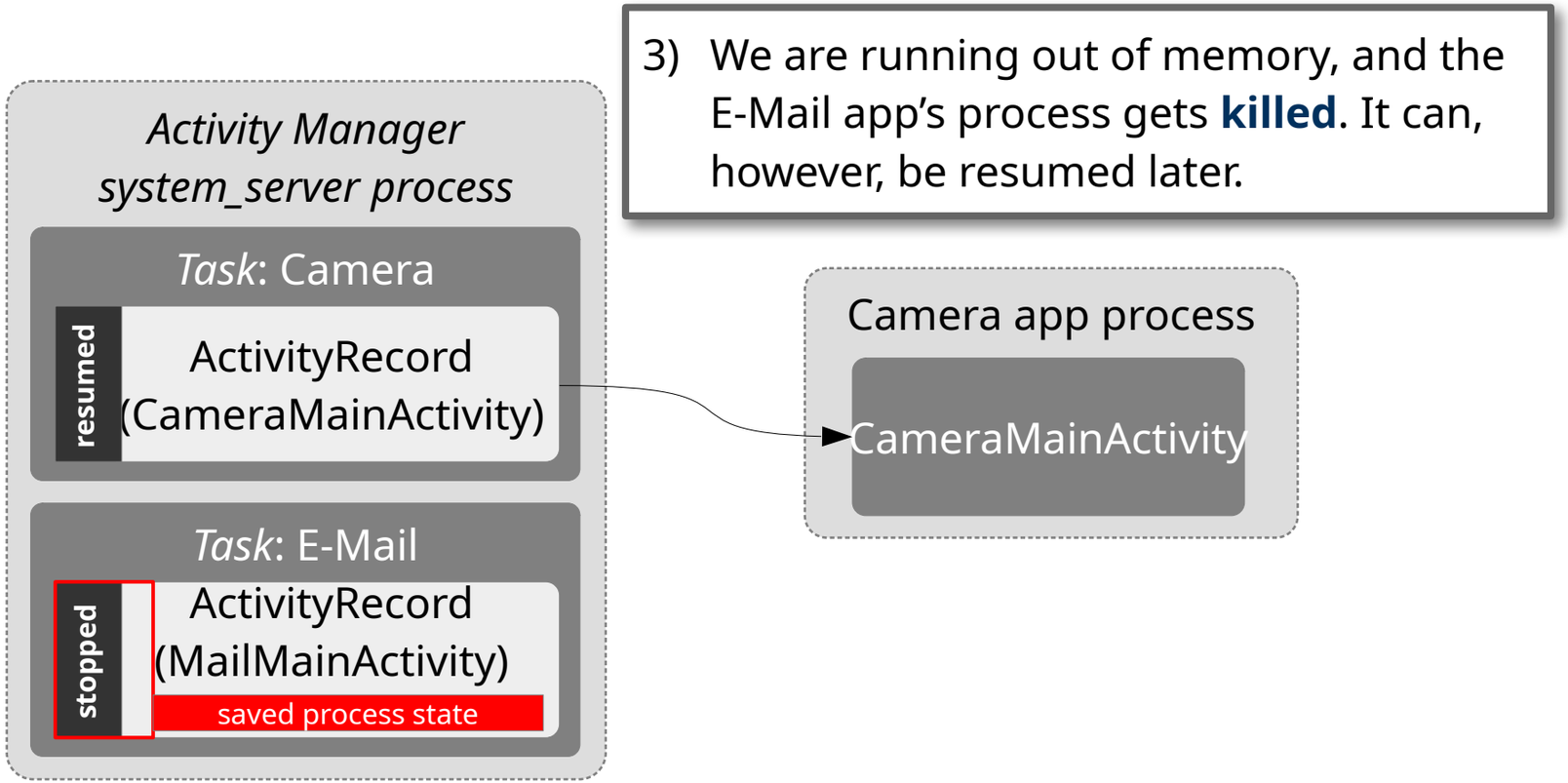
The Activity Manager (2)

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



The Activity Manager (3)

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



Agenda

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

Energy Consumption: Basics

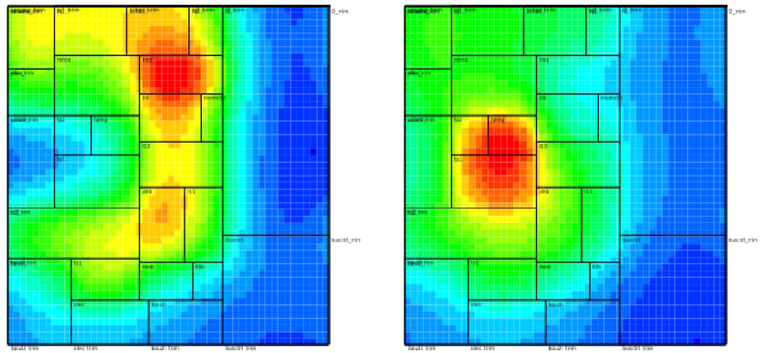
- Energy is not really “consumed” but **transformed into heat** and radiated
  - usually unused/lost
  - undesirable (possibly cooling necessary)
  - reduces battery runtime
- Fundamental physical equations that relate energy and **electrical current**:
  - **$E = P \cdot \Delta t$** (Energy[J] = Power[W]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

Power Dissipation in CMOS Semiconductors

... primarily two components: dynamic+static power

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

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



Source: Frank Bellosa, Karlsruher Institut für Technologie

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

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

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

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

Bold-print:
concerns the OS

- **Dynamic Voltage and Frequency Scaling (DVFS)**
  - → pays off because V is squared
  - → possible when the CPU is underutilized or constantly waiting for the memory
- **Switching off the clock**
  - → $P_{dyn} = 0$, but state is retained!
- Improved semiconductor manufacturing (smaller)
- More efficient computer architecture

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

- Better manufacturing technology or *larger* structures
- **Switching off unused components**

Issue 1: Power Loss in Idle Mode

- A dormant system must be switched off!
  - Here: Measurements on the Openmoko Neo FreeRunner [1]

Factor of 10 compared to the underutilized processor!

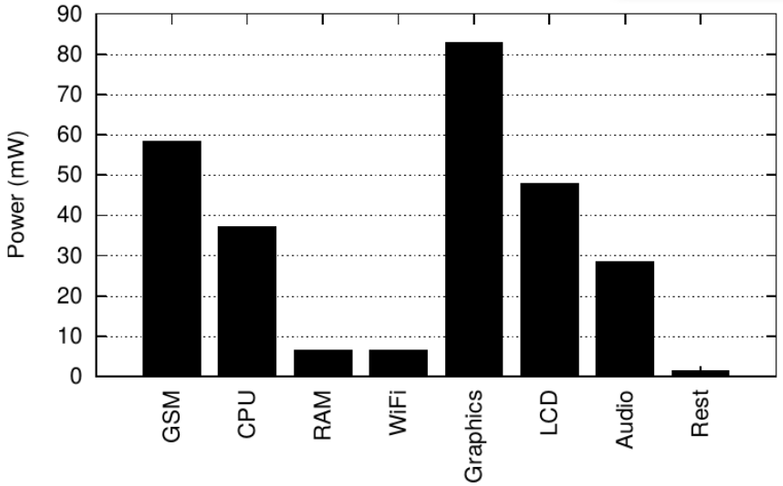


Figure 3: Average power consumption while in the idle state with backlight off. Aggregate power is 268.8 mW.

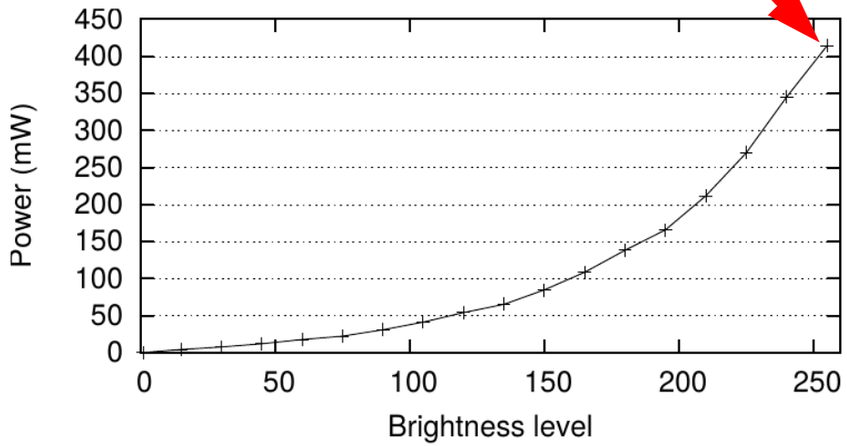


Figure 4: Display backlight power for varying brightness levels.

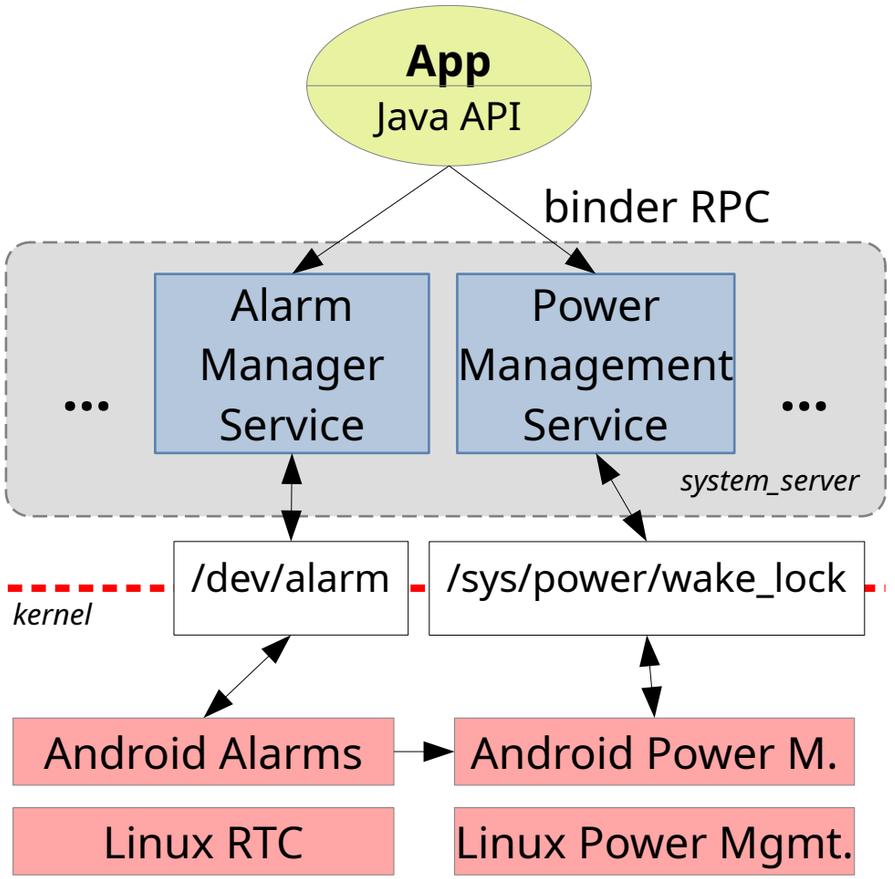
In the SUSPENDED state (various components switched off) it is only 68.6 mW.

Solution 1: Wake Locks and Alarms

- **Approach**: Switch off all inactive components as quickly and often as possible – Sleep Mode (SUSPENDED)

- Apps must …
  - explicitly prevent this if necessary: **Wakelocks**
  - be able to wake up the system on a time-controlled basis: **Alarms**

- This requires extensions to the Linux kernel.



Wakelocks

- Apps require the `android.permission.WAKE_LOCK` permission to create a Wakelock.
- Wakelock types:

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.*

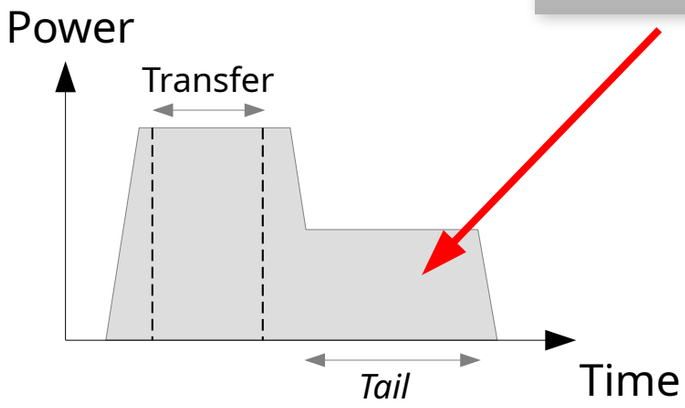
Discussion: Wakelocks

- Initially, Linux kernel developers were reluctant to integrate wakelocks.
- **Problem**: The mechanism is not linked to the existence of the process.
  - If you “forget” to release the lock, the device stays on!
  - Android apps survive their process → Activities, Receivers, …
- **Solution**? The concept has now been adopted under the name **“Suspend Blockers”**.
  - Access to /sys/power/wake_lock and …/wake_unlock only for root.
  - The feature is optional.
- Direct use should also be avoided under Android.
  - Instead, link to activity management and UI

Alarms

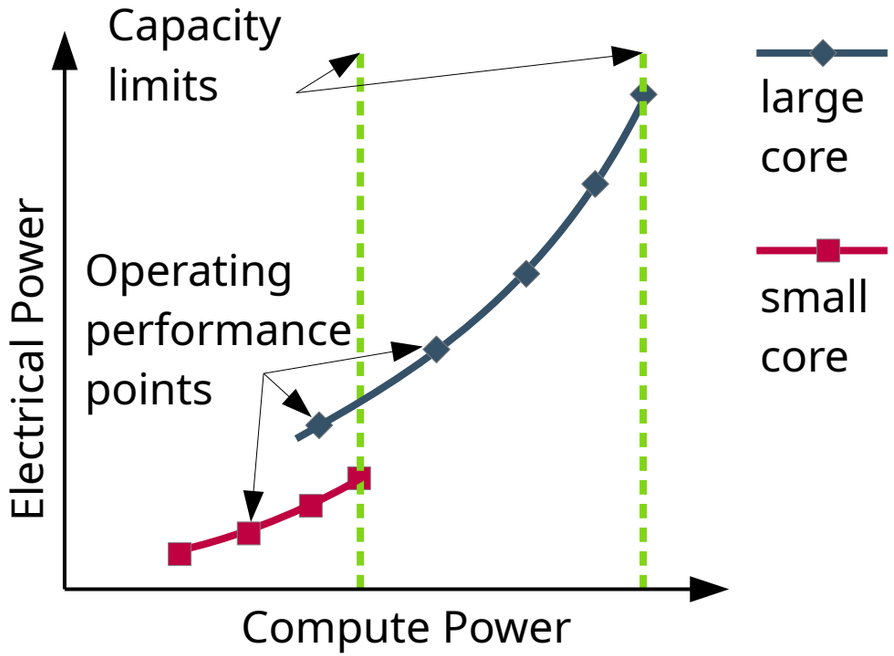
- Time-controlled wake-up of the system
  - Triggering an **intent**
  - Also works if the app that requested the alarm is no longer active!
  - The alarm manager holds a **wakelock** during handling.
- Applications: Fetching emails, weather forecast, …
- Alarms can be postponed by the system to bundle network transfers
  → Save energy

Bundling avoids multiple consumption of **tail energy** in mobile communications.



Issue 2: Clock / Voltage Control

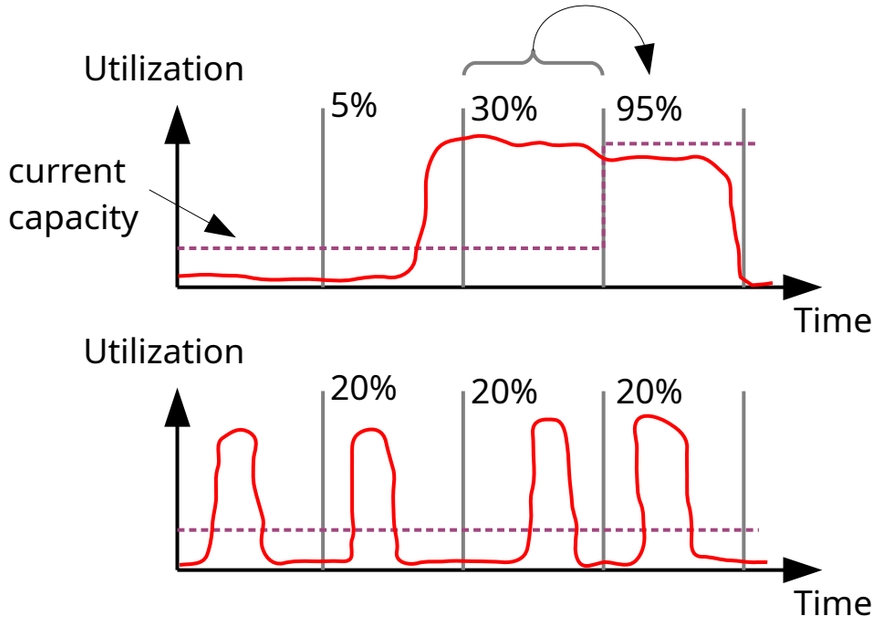
- Goal: Maximum computing power with minimum energy consumption
- Modern **heterogeneous** multicore processors offer various **operating performance points (OPPs)**.
  - Typical: ARM big.LITTLE architecture with e.g. 4 large and 4 small cores



The scheduling problem is extended by energy-aware clock control!

Solution 2(a): Load-dependent Control

- **Regular Linux**: CPU-core utilization is monitored at regular intervals – the clock is adjusted depending on a threshold value.
- **Weakness**: Slow response
- **Causes**:
  - The clock control does not know not know as much as the scheduler about the **future** load.
  - No information about the energy efficiency of the respective operating performance points.



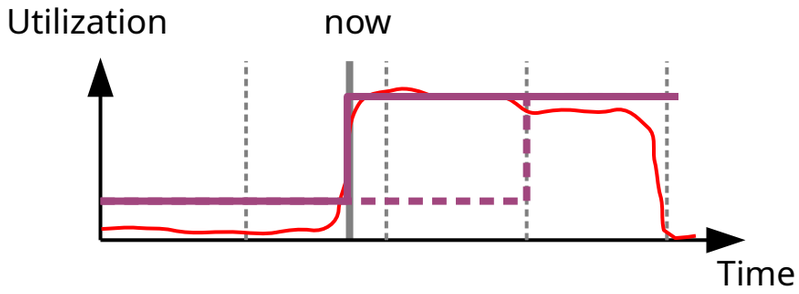
Late reaction by **cpufreq governor**

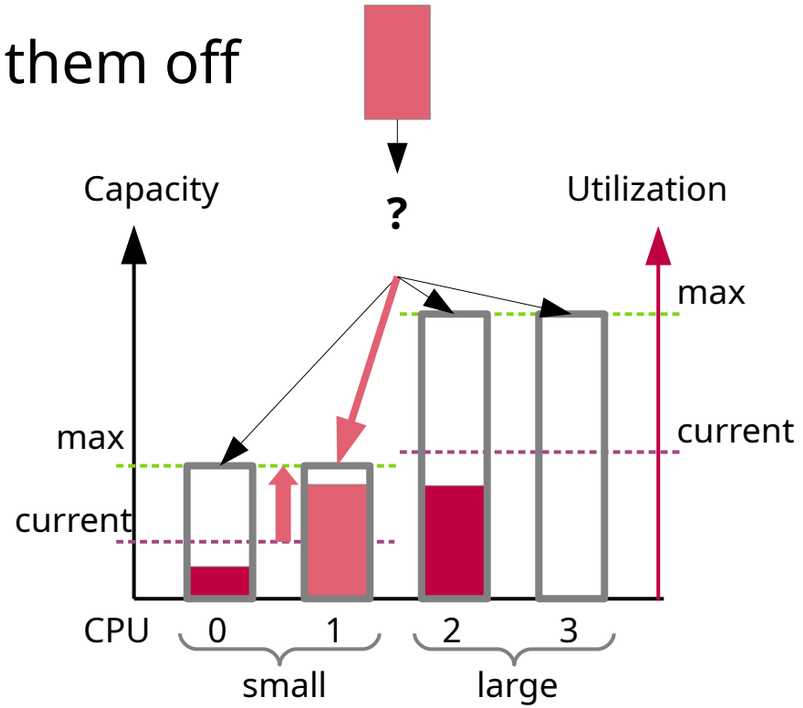
Average too low, **no reaction**

Solution 2(b): Android EAS

“Energy-Aware Scheduling”

- Based on energy model (via OPPs)
- decides where a task requires little energy
- can completely free cores from load and switch them off
- **Immediate reaction** to load changes!





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.