TECHNISCHE UNIVERSITÄT DRESDEN

**Fakultät Informatik** Institut für Systemarchitektur, Professur für Betriebssysteme

# BETRIEBSSYSTEME UND SICHERHEIT

mit Material von Olaf Spinczyk, Universität Osnabrück

## *Mobile Betriebssysteme am Beispiel von Android*

**https://tud.de/inf/os/studium/vorlesungen/bs**

**HORST SCHIRMEIER**

# Inhalt

- Anforderungen
- Android-Überblick
- Sicherheit
- Speicherplatz
- Energie
- Zusammenfassung

Silberschatz, Kap. ...
--- ☹

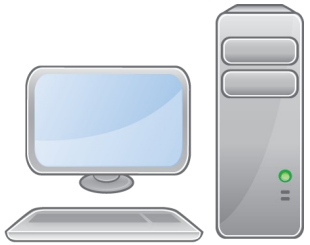
Tanenbaum, Kap. ...
10.8: Android

# Inhalt

- **Anforderungen**
- Android-Überblick
- Sicherheit
- Speicherplatz
- Energie
- Zusammenfassung

# Mobile Vielzwecksysteme

… unterscheiden sich im **Anwendungsprofil** und bei der **Hardware** vom klassischen Desktop-/Server-Rechner



**Klassischer PC**

- Wenige vertrauenswürdige Applikationen
- permanente Energieversorgung
- Seltene Kommunikation über Kabelnetze
- Viel Platz für Hauptspeicher, Platten, Kühlung, usw.

**Smartphone**

- Wechselnde unbekannte Apps unbekannter Hersteller
- Akkubetrieb
- Häufige Kommunikation über drahtlose Netze (Mobilfunk)
- Stark beschränkter Bauraum, nur RAM und Flash-Speicher

# Mobile Vielzweck*betriebs*systeme

... müssen daher ...

- Applikationen und ihre Daten vom Rest des Systems **besser isolieren**
- aggressiver **Speicherplatz sparen**
- alle Hardwaremechanismen zum **Energiesparen** ausnutzen und energiegewahres Anwendungsverhalten unterstützen

# Inhalt

- Anforderungen
- **Android-Überblick**
- Sicherheit
- Speicherplatz
- Energie
- Zusammenfassung

# Android


- *Open Handset Alliance* (primär Google), 2007
  - T-Mobile, Motorola, Samsung, …
- **Vision:**

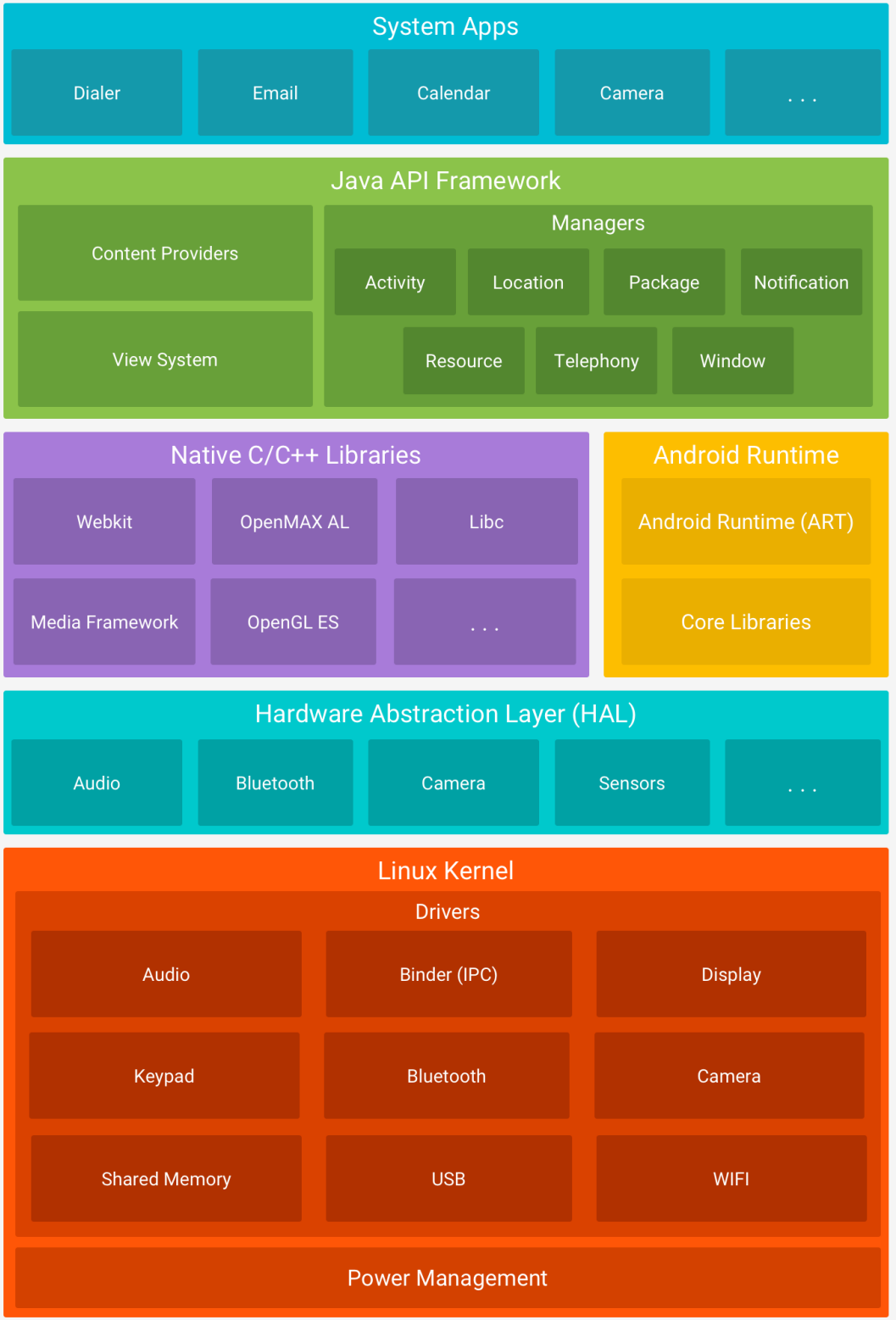
  *"… accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience."*

- Infrastruktursoftware-Plattform für *Smartphones*
  - *Open Source*
- Diverse Produkte heute verfügbar

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

# Architektur

- Linux und Java – aber anders …



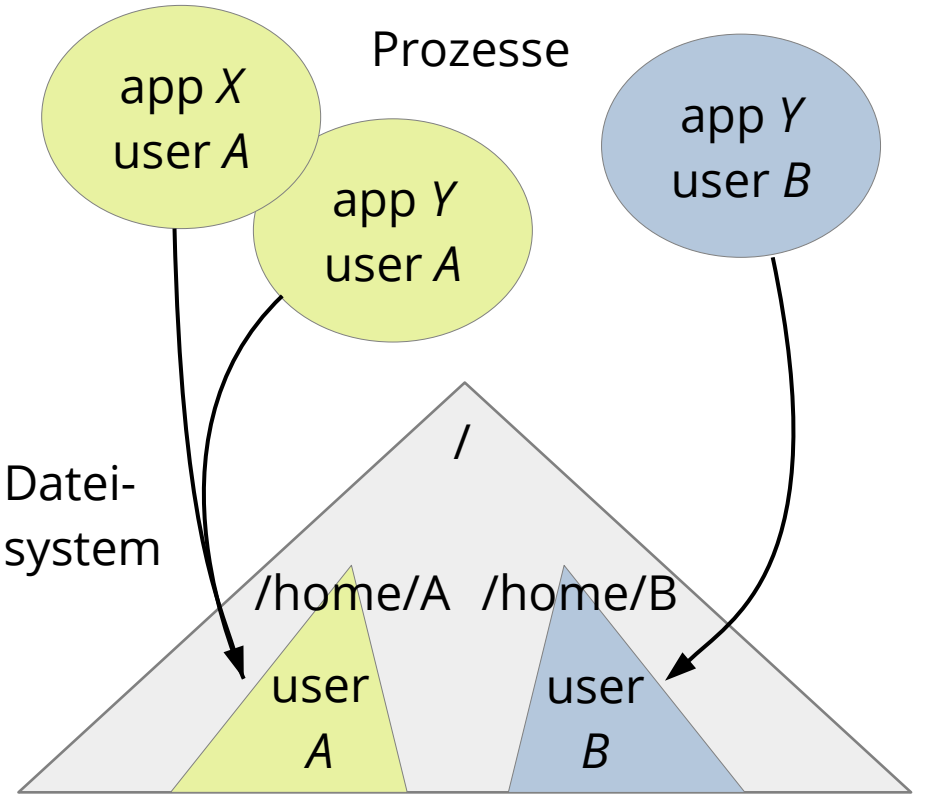

Quelle: https://developer.android.com/guide/platform

# Inhalt

- Anforderungen
- Android-Überblick
- **Sicherheit**
- Speicherplatz
- Energie
- Zusammenfassung

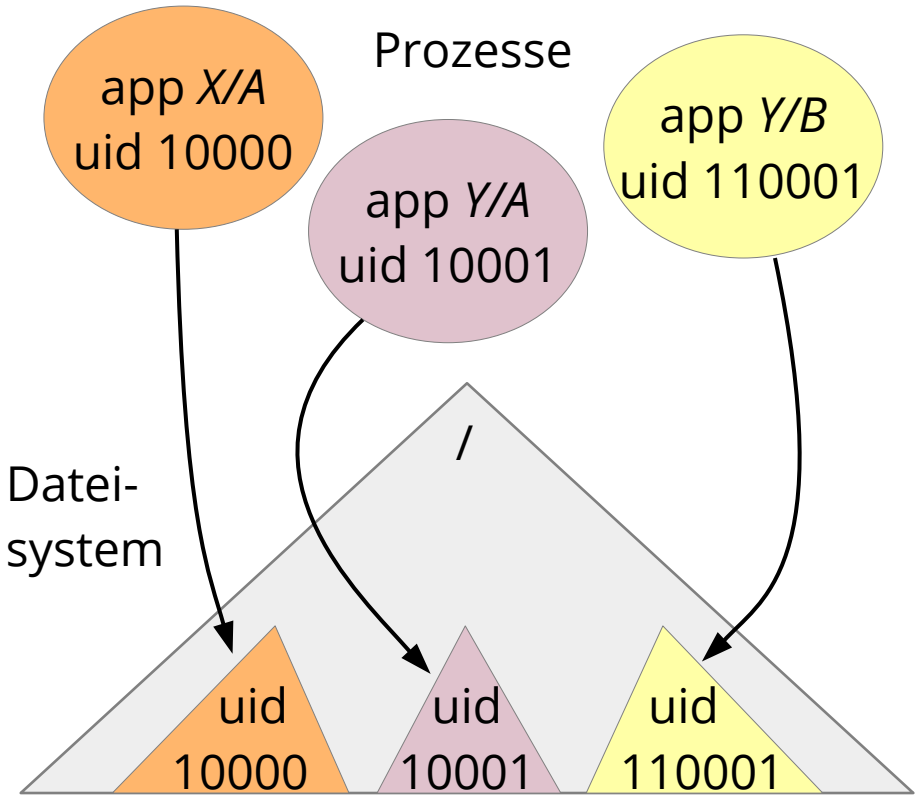
# UIDs als Grundlage für *Sandboxing*

Normales **Linux**: UID pro Benutzer




Eine fehlerhafte oder böswillige App gefährdet alle Daten des Benutzers!

**Android**: UID pro App/Benutzer

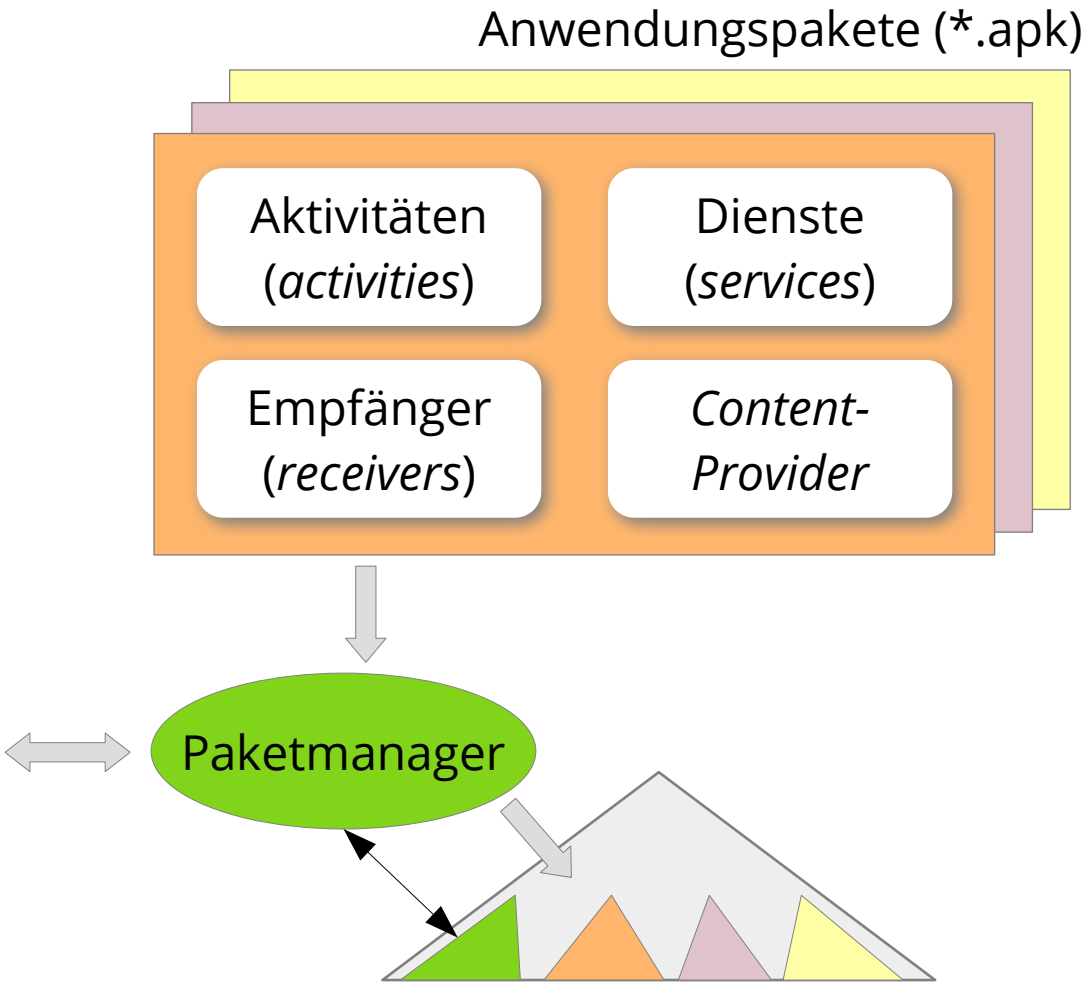



Jede App-Instanz (pro Benutzer) hat im Dateisystem ihre eigenen Daten.

# UID-Vergabe durch Paketmanager

- erfolgt automatisch bei der Paketinstallation

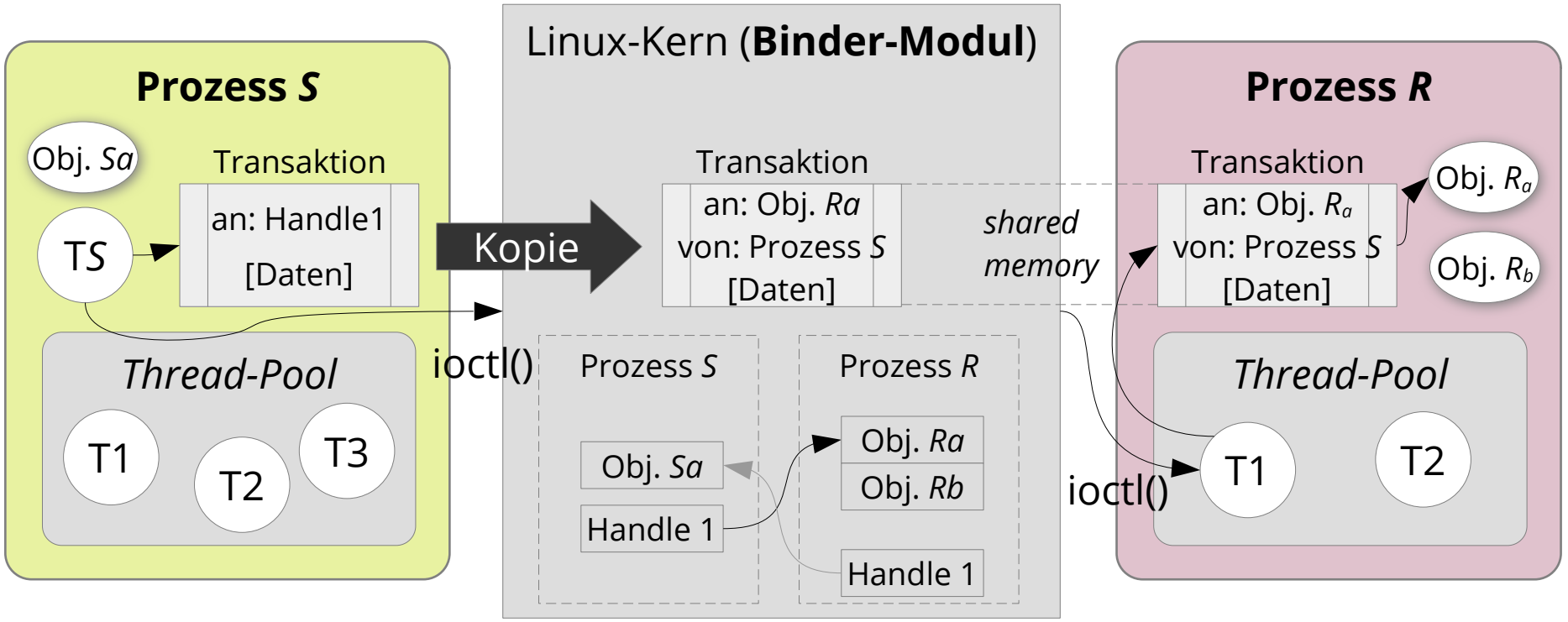
| UID | Zweck |
|---|---|
| 0 | Root-Benutzer |
| 1000 | Kernsystemdienst (Prozess system_server) |
| 1001 | Telefoniedienste |
| 1013 | (Hardwarenahe) Mediendienste |
| 2000 | Shell |
| 10000-19999 | Dynamisch zugewiesene Anwendungs-UIDs |
| 100000 | Anwendungs-UIDs des zweiten Nutzers |



# App-Datenaustausch: Binder-IPC

**“bottom-up”-Erklärung**

- Ermöglicht **objektorientierte Aufrufe** über Prozessgrenzen
  - Keine klassische Linux-Abstraktion konnte das

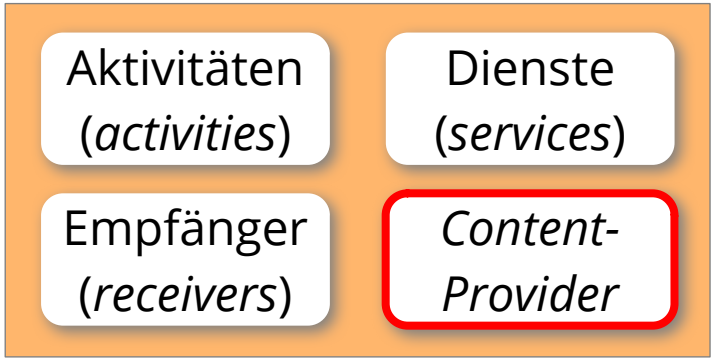



***Handles* sind einfache *Integers* (wie Datei-deskriptoren). → IPC-Rechte / *Capabilities***

**Eine Objektreferenz im Datenteil wird automatisch vom Binder-Modul durch ein Handle ersetzt.**

# App-Datenaustausch: *Content-Provider*

- Klasse innerhalb einer App
- Stellt Inhalte anhand einer URI per Binder-IPC zur Verfügung:




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

**Identifikation** des *Providers* (“authority”) **Pfad** für *Provider*

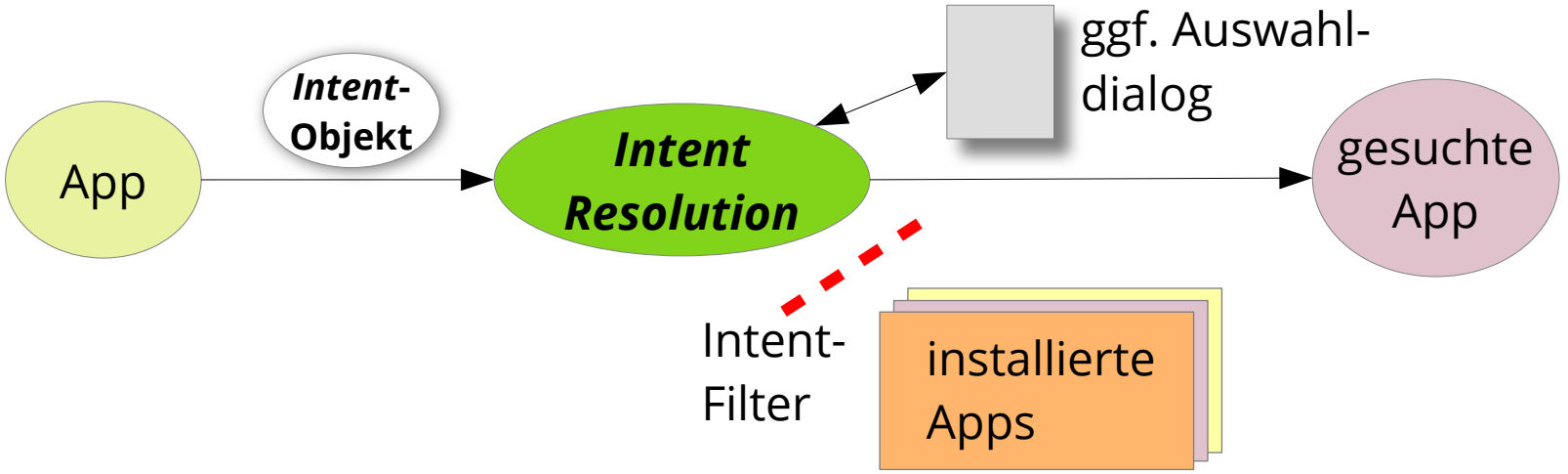
- Abfrage von Inhalten erfolgt per *Remote Procedure Call*, z.B.:

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

**Doch wie wird die (richtige) Email-App gefunden und gestartet? → *Intents***

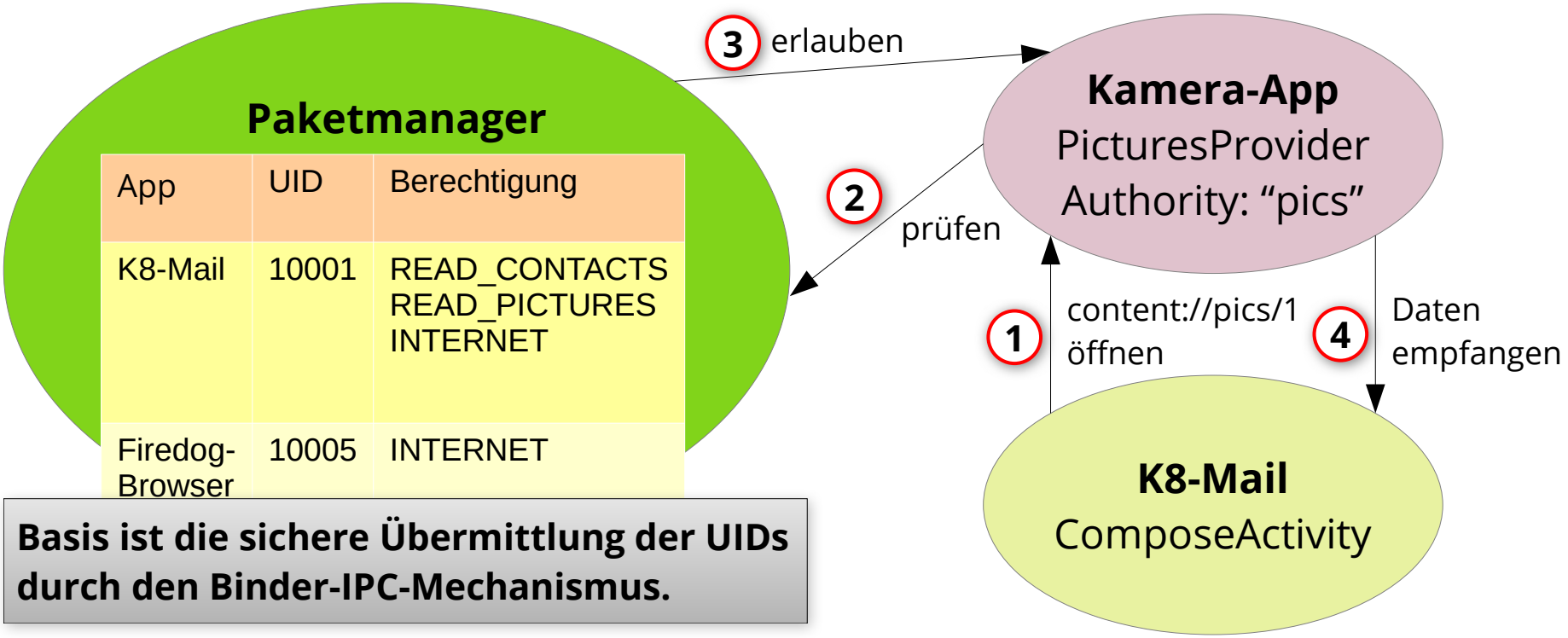
# App-Datenaustausch: *Intents*

- Objekt zur Beschreibung abstrakter ***Absichten***, die eine App aber nicht selbst implementiert, z.B.
  - Daten verschicken, Webseite anzeigen, jemanden anrufen, ...
- oder **Systemereignisse**, die von einer (System-)App behandelt werden sollen
  - niedriger Batteriestand, Anruf annehmen, ...



# App-Berechtigungen

- Sind vordefiniert
- Werden Apps bei der Installation zugewiesen
- Verwaltung erfolgt durch den Paketmanager

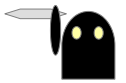


# Inhalt

- Anforderungen
- Android-Überblick
- Sicherheit
- **Speicherplatz**
- Energie
- Zusammenfassung

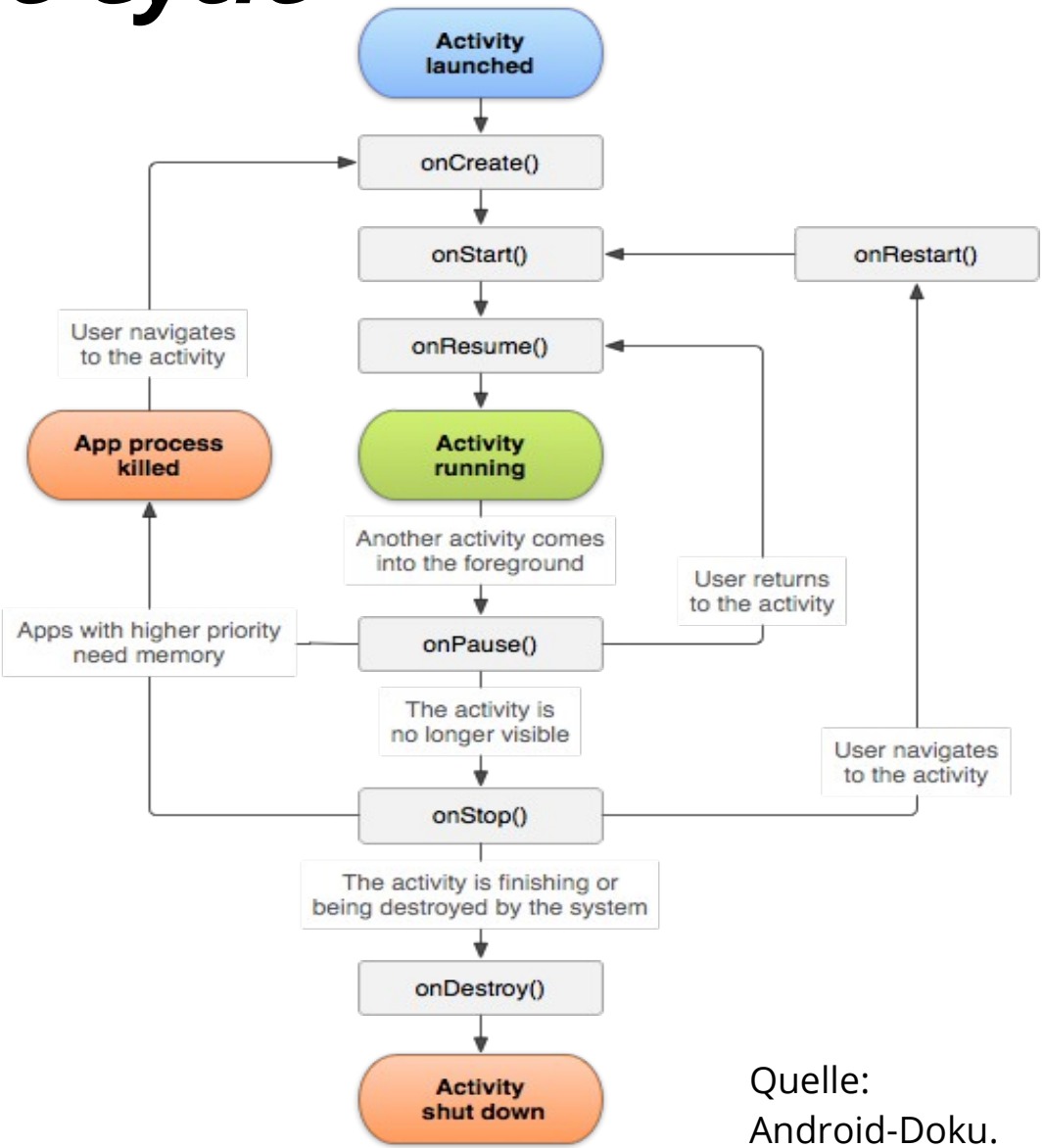
# Viele Apps trotz begrenztem Speicher

- Android-Systeme haben (vergleichsweise) kleine Hauptspeicher
- Dauerndes *Paging* verschlechtert die Leistung und Haltbarkeit des Hintergrundspeichers (Flash-Technologie)
  - Typische Haltbarkeit von NAND-Flash: 2 Millionen Schreibvorgänge
- **Lösung:** App-***Aktivitäten*** können jederzeit beendet und neu gestartet werden!
  - Der ***Out of Memory Killer*** sucht ständig nach Opfern
  - Prioritäten: Zuletzt Systemprozesse, Hintergrunddienste, die Vordergrundaktivität und sichtbare Aktivitäten



# Android *Activity Life Cycle*

- Apps müssen nach dem Töten+Neustarten wieder im selben Zustand sein
  - System: Views (GUI-Layout)
  - App: Sonstiger Zustand

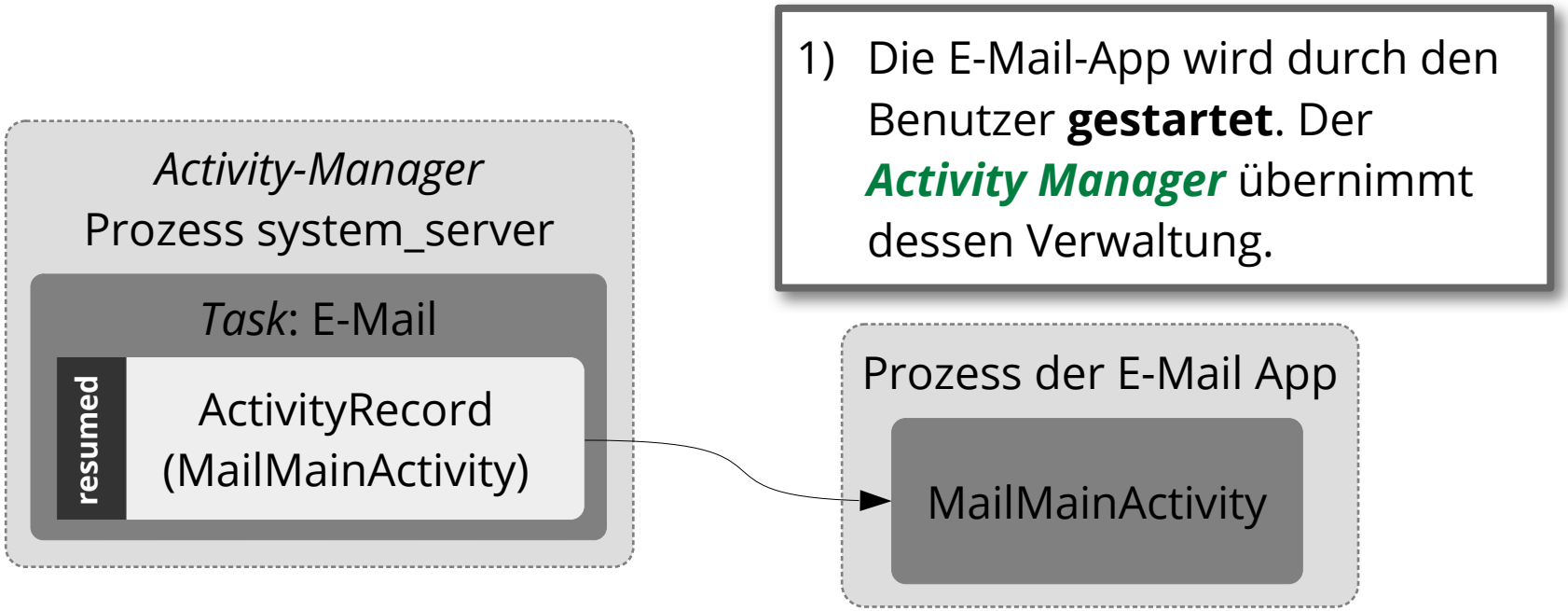



**Ergebnis:**
**Die App überlebt ihren Prozess!**

Quelle: Android-Doku.

# Der Aktivitäten-Manager (1)

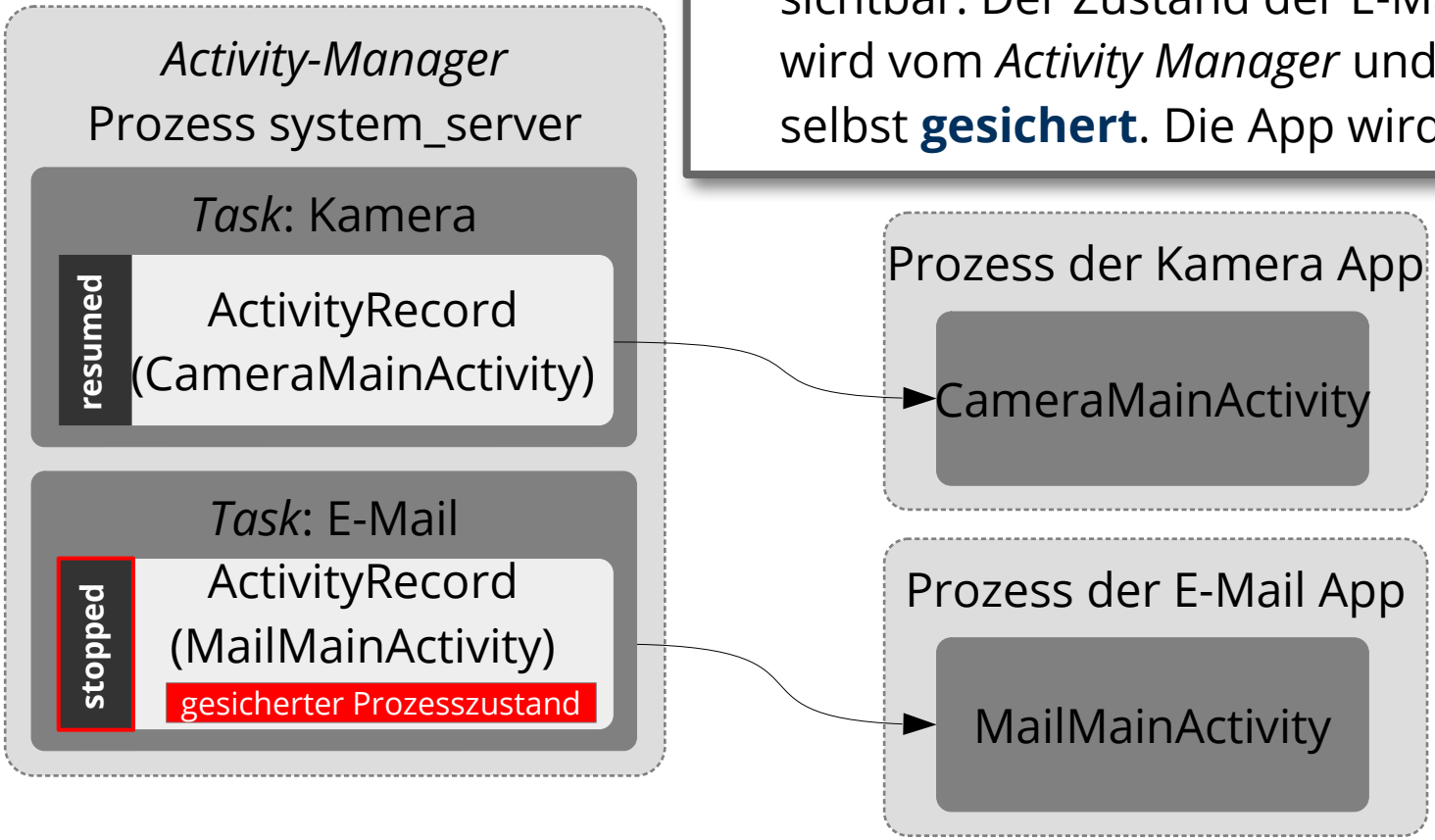
- … verwaltet alle Informationen über laufende Apps



# Der Aktivitäten-Manager (2)

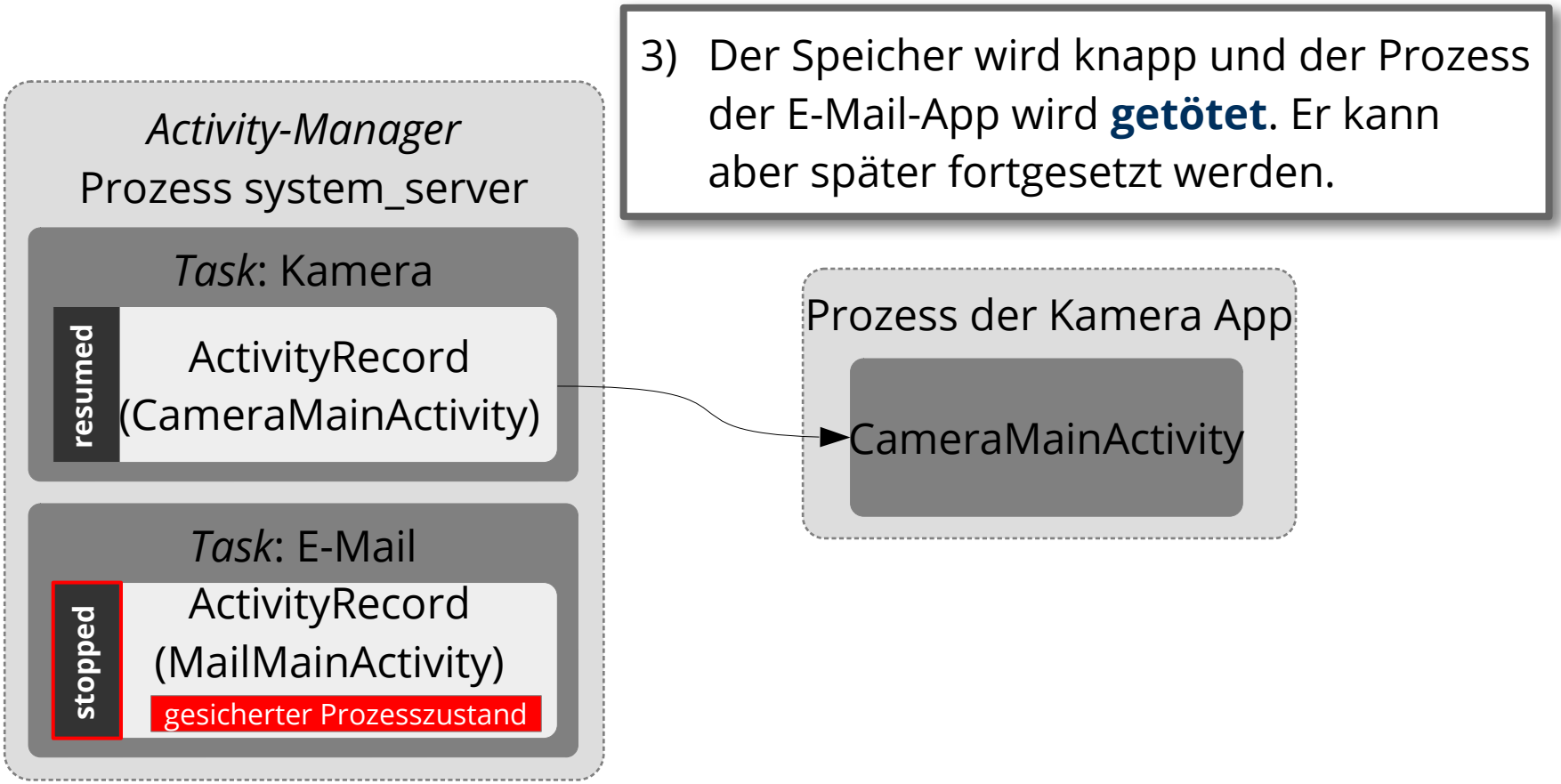
- … verwaltet alle Informationen über laufende Apps

2) Die Kamera-App wird gestartet und sichtbar. Der Zustand der E-Mail-App wird vom *Activity Manager* und der App selbst **gesichert**. Die App wird **gestoppt**.



# Der Aktivitäten-Manager (3)

- … verwaltet alle Informationen über laufende Apps



# Inhalt

- Anforderungen
- Android-Überblick
- Sicherheit
- Speicherplatz
- **Energie**
- Zusammenfassung

# Energieverbrauch: Grundlagen

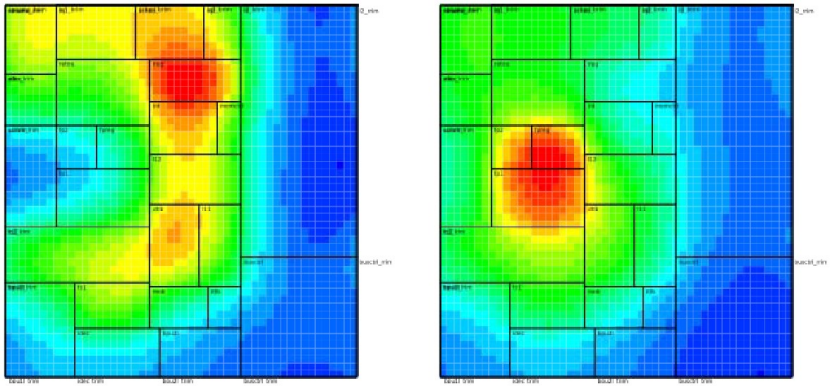
- Die „verbrauchte“ Energie wird in Wahrheit **in Wärme umgewandelt** und abgestrahlt
  - meist ungenutzt
  - störend (ggf. Kühlung nötig)
  - verringert die Batterie-/Akkulaufzeiten
- Ein paar fundamentale physikalische Gleichungen zeigen den Zusammenhang zwischen Energie und **Stromfluss**:
  - **$E = P \cdot \Delta t$** (Energie[J] = Leistung[W] · Zeit[s])
  - **$P = U \cdot I$** (Leistung[W] = Spannung[V] · Strom[A])

# Verlustleistung im CMOS-Halbleiter

… besteht primär aus zwei Anteilen: dynamische+statische Leist.

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

Modell: Viele kleine **Kapazitäten** (Leitungen, Speicher) werden **beim Schalten** geladen und entladen.



Quelle: Frank Bellosa, Karlsruher Institut für Technologie

$U$: Versorgungsspannung
$f$: Taktfrequenz
$C_L$: geschaltete Kapazität
$\alpha$: Anteil der schaltenden Transistoren

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

$I_{leak}$: **Leckstrom**

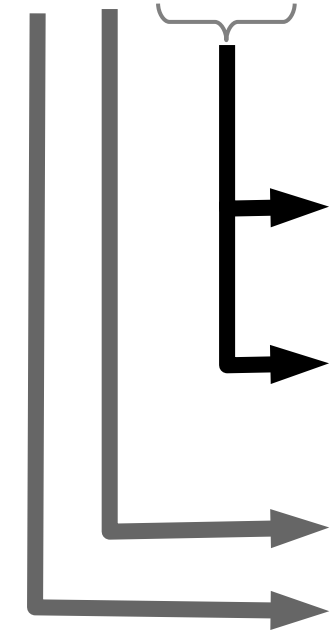
Ursache: **Quanteneffekte** im Halbleiter, die mit sinkenden Strukturgrößen (Isolatordicken) exponentiell schlimmer werden.

→ Der Leckstrom ist sehr klein, fließt aber **immer**, solange Spannung anliegt.

# Ansätze zum Sparen von Energie

Fett gedruckt: betrifft BS

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



- **Absenken von Spannung und 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!

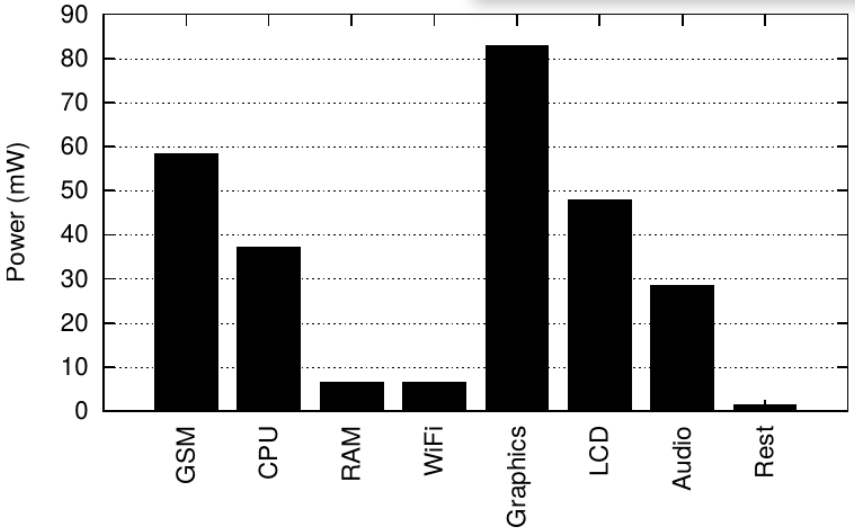



Figure 3: Average power consumption while in the idle state with backlight off. Aggregate power is 268.8 mW.

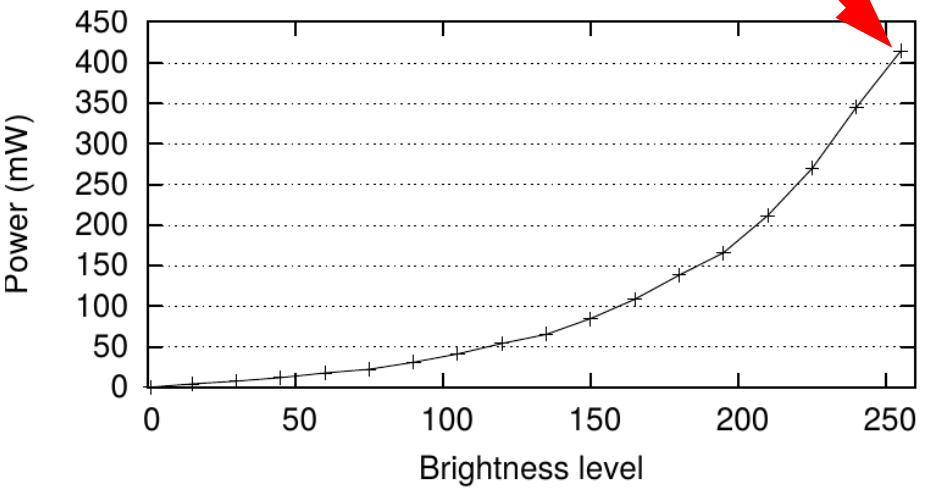



Figure 4: Display backlight power for varying brightness levels.

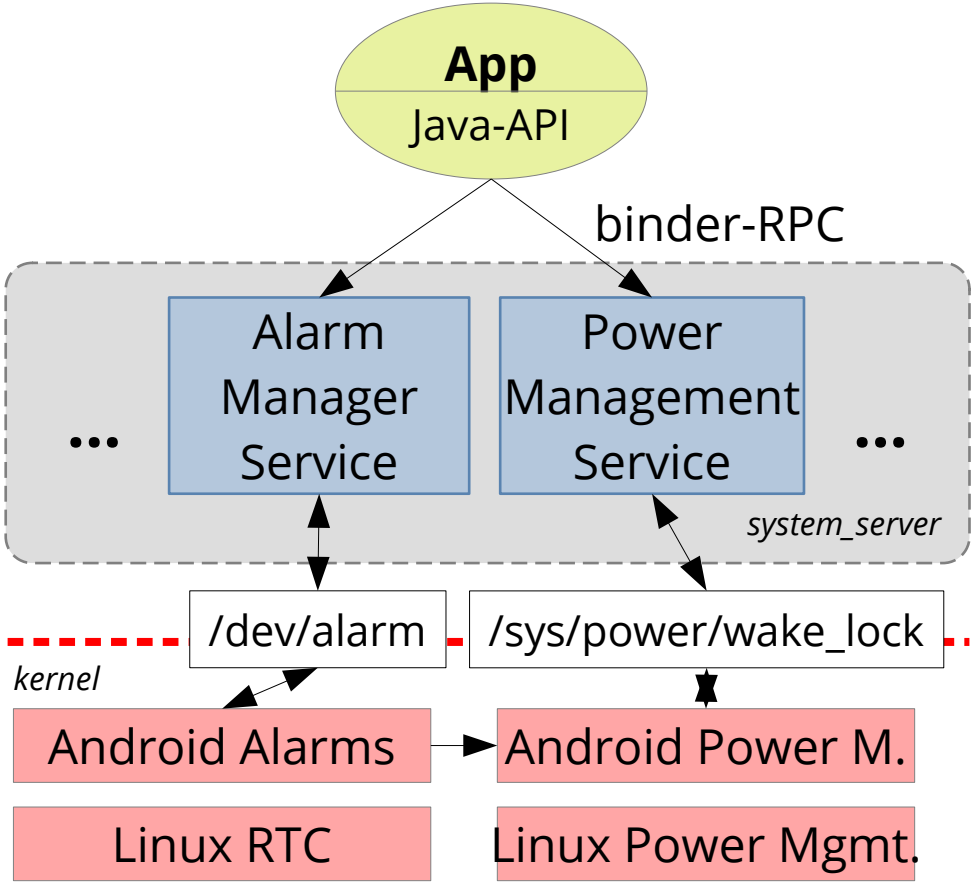
Im SUSPENDED-Zustand (div. Komponenten abgeschaltet) sind es nur 68,6 mW.

# 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:

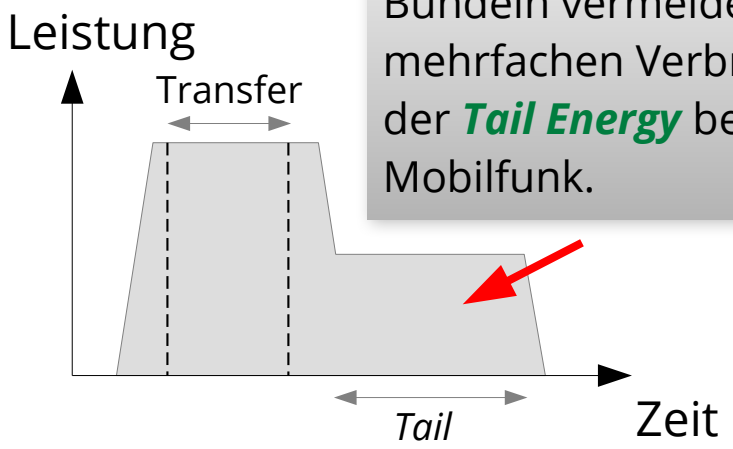
| **Name** | **Auswirkung** (laut Android-Dokumentation) |
|---|---|
| FULL_WAKE_LOCK | *Ensures that the screen and keyboard backlight are on at full brightness.* |
| PARTIAL_WAKE_LOCK | *Ensures that the CPU is running; the screen and keyboard backlight will be allowed to go off.* |
| PROXIMITY_SCREEN_OFF_WAKE_LOCK | *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 geträubt, *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*

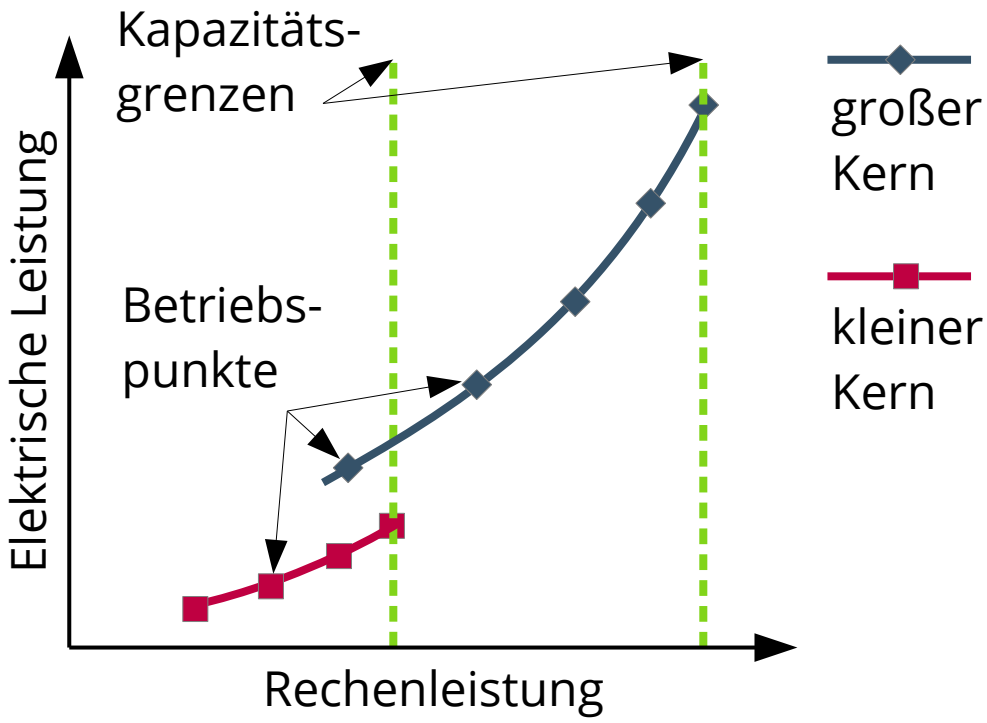
- Zeitgesteuertes Wecken des Systems
  - Auslösung eines *Intents*
  - Funktioniert auch, wenn die App, die den Alarm angefordert hat, nicht mehr aktiv ist!
  - Während der Behandlung hält der Alarm-Manager ein *Wakelock*
- Anwendungen: Abholen von Emails, Wettervorhersage, …
- Alarmzeitpunkte können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




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

# Problem 2: Takt-/Spannungssteuerung

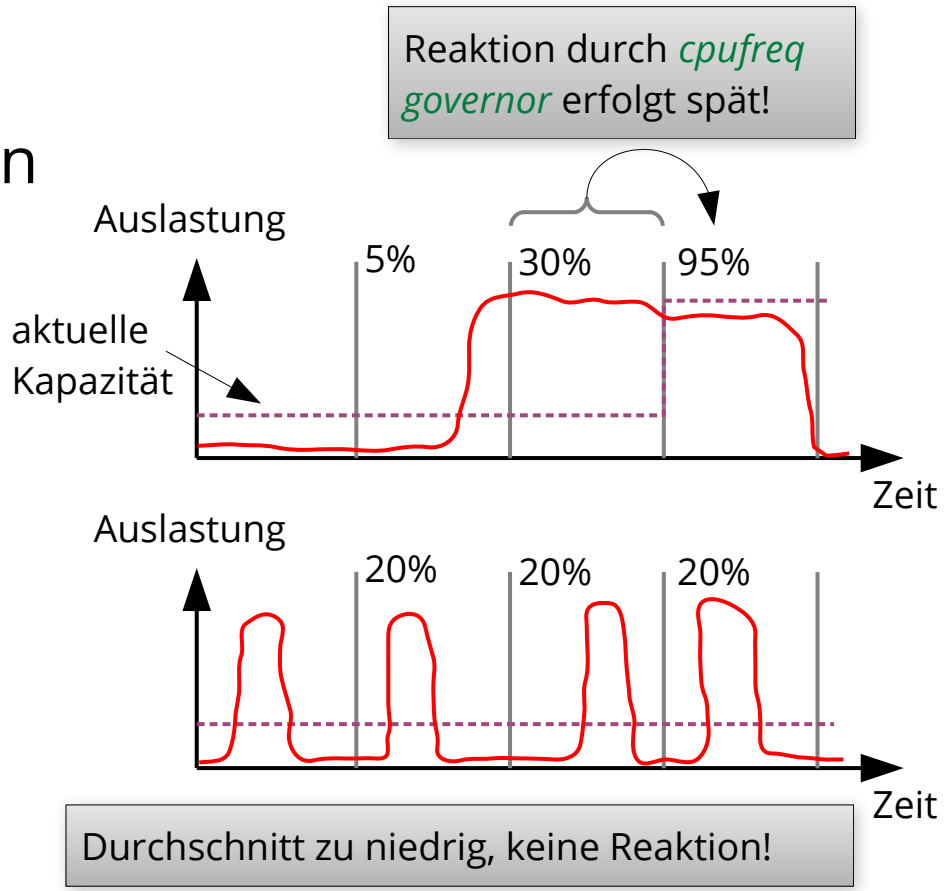
- Ziel: Maximale Rechenleistung bei minimalem Energieverbrauch
- Moderne **heterogene** Multicore-Prozessoren bieten diverse **Betriebspunkte**.
  - Typisch: ARMs big.LITTLE-Architektur mit 4 großen und 4 kleinen Kernen




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

# Lösung 2(a): Lastabhängige Regelung

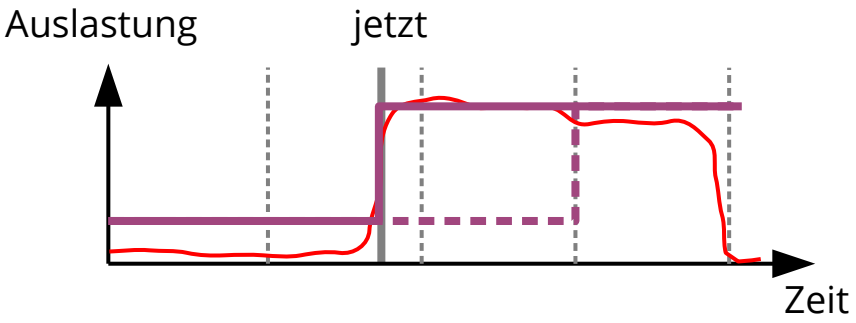
- **Normales Linux**: Die Auslastung der CPU-Kerne wird in regelmäßigen Abständen betrachtet. Schwellwertabhängig wird der Takt angepasst.
- **Schwäche**: Langsame Reaktion
- **Ursachen**:
  - Die Taktregelung weiß nicht so viel wie der *Scheduler* über die **zukünftige** Last
  - Keine Information über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

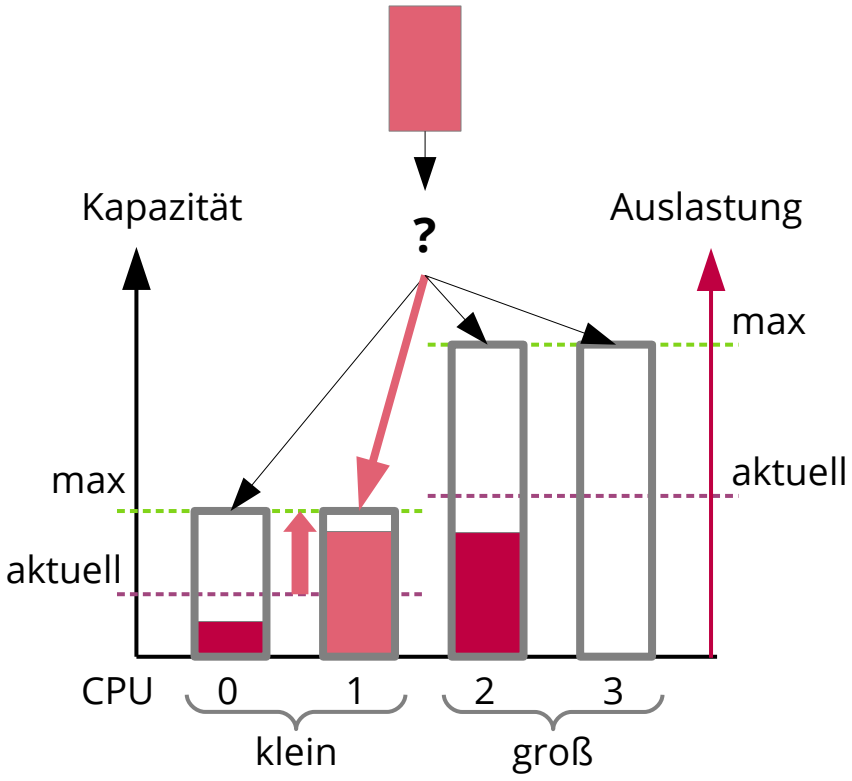


# Lösung 2(b): Android EAS

***"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!



# Inhalt

- Anforderungen
- Android-Überblick
- Sicherheit
- Speicherplatz
- Energie
- **Zusammenfassung**

# Zusammenfassung

- Die Android-Entwicklung hat einen Innovationsschub bei Linux ausgelöst
  - *Sandboxing* von Applikationen
  - Mehr Kontrolle über Abschalten und Aufwachen
  - Energiegewahres *Scheduling*
- Teilweise werden Linux-Konzepte anders genutzt als ursprünglich geplant
  - UIDs pro App
  - Anwendungen laufen länger als deren Prozesse

# 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.