EMBEDDED SOFTWARE LABORATORY

RWTH AACHEN UNIVERSITY

JTRES 2012

A Real-time Extension to the Android Platform

Igor Kalkov, Dominik Franke,
John F. Schommer, Stefan Kowalewski

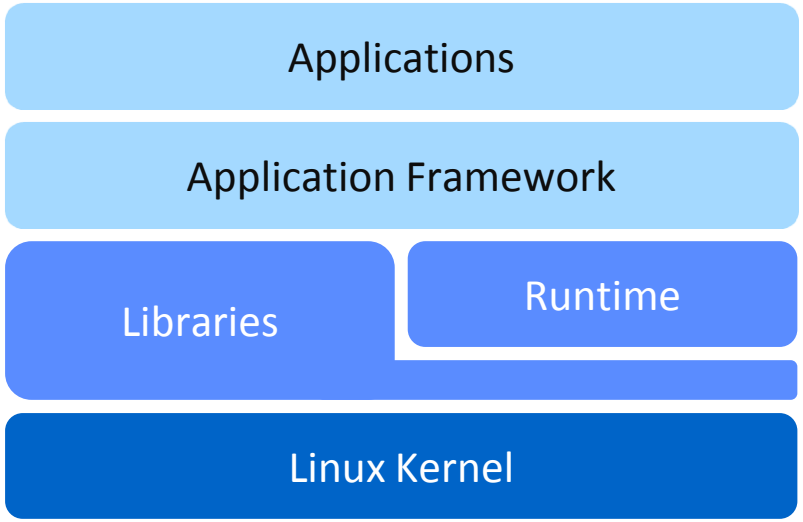
24-26 October 2012

Copenhagen, Denmark

Introduction

- Mobile platform by Open Handset Alliance
  - Supervised by Google
  - Open-sourced under Apache 2.0 license

- Android software stack:
  - Applications
    - Stock & user applications
  - Application framework
    - Services & system managers
  - Android runtime
    - Dalvik virtual machine
    - SSL, media, SQLite database
  - Adapted Linux kernel
    - Hardware drivers, memory & process management

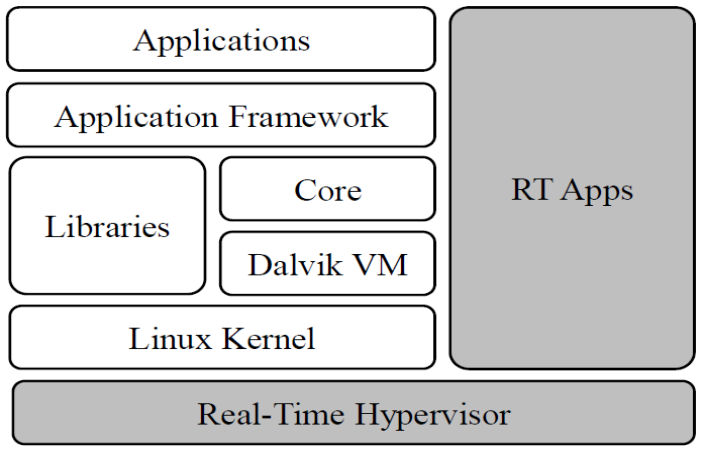


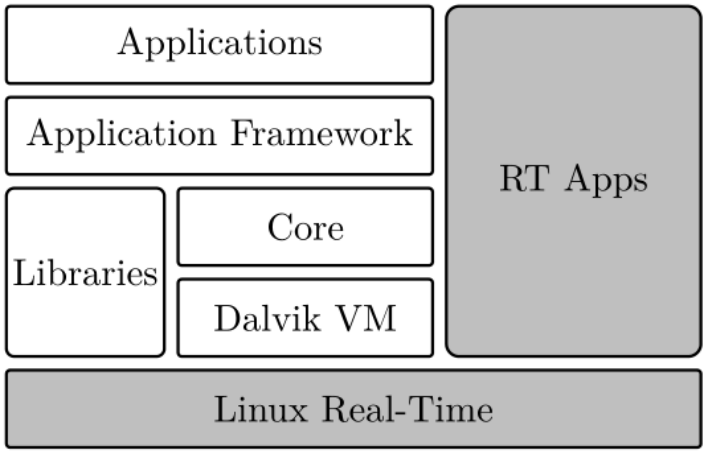
Motivation

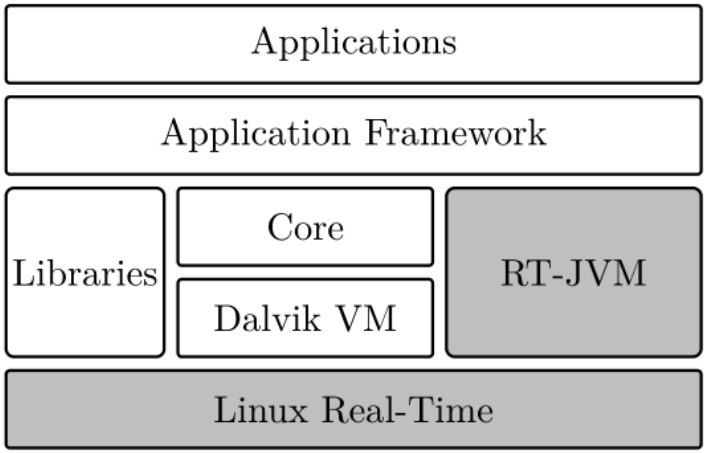
- Real-time support expands the field of application
  - Opening safety- & time-critical domains
  - In-field monitoring, controlling platform for home automation
  - Better core functionalities: speech or video processing
- Goals
  - Possibility of serving real-time requests
  - Keeping original functionality / backward compatibility
- Wide range of compatible hardware platforms
  - Smartphones, tablets, OMAP hardware
- Further applications: eReaders, TVs, Nanosatellites

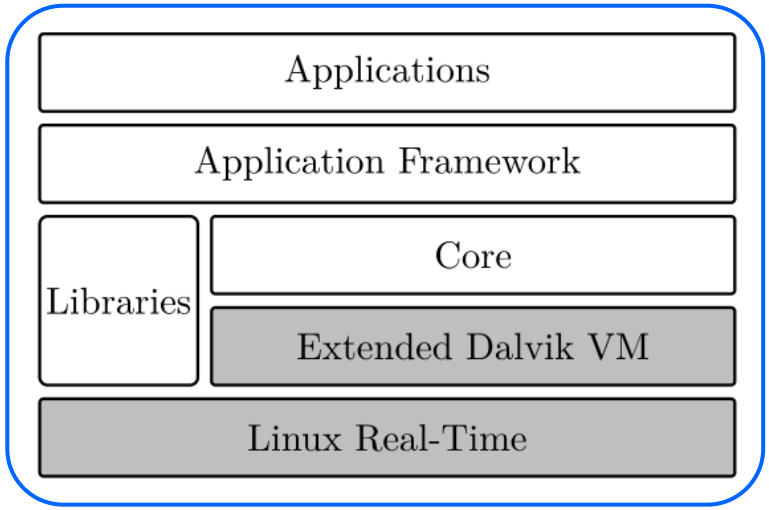
Related Work

- Several proposed approaches:



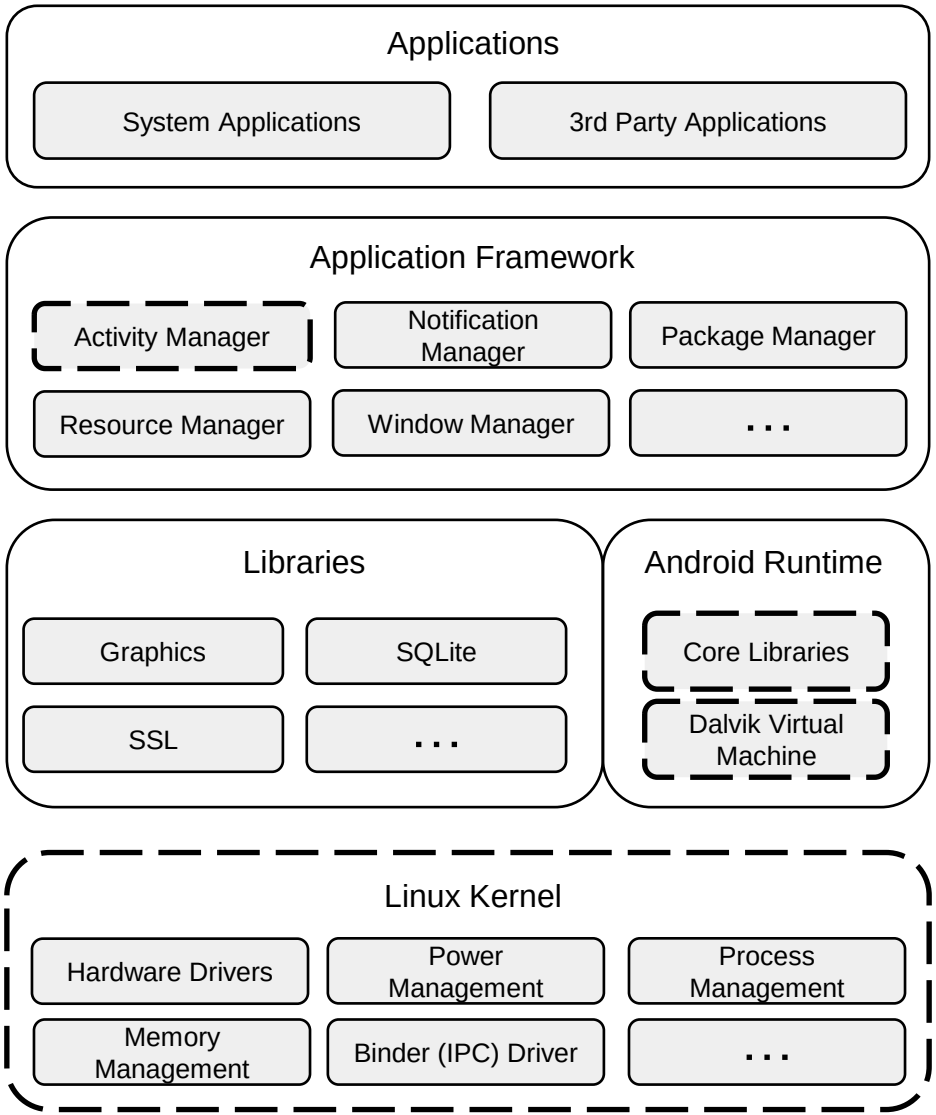






Approach

- Extended Activity Manager
  - Reliable execution of RT apps
  - Bypassing OOM process killer
- Modified Dalvik VM
  - Encapsulated priority selection
  - Explicit memory management
- Improved Linux kernel v2.6.29
  - Patched with PREEMPT_RT
  - Enabled priority scheduling



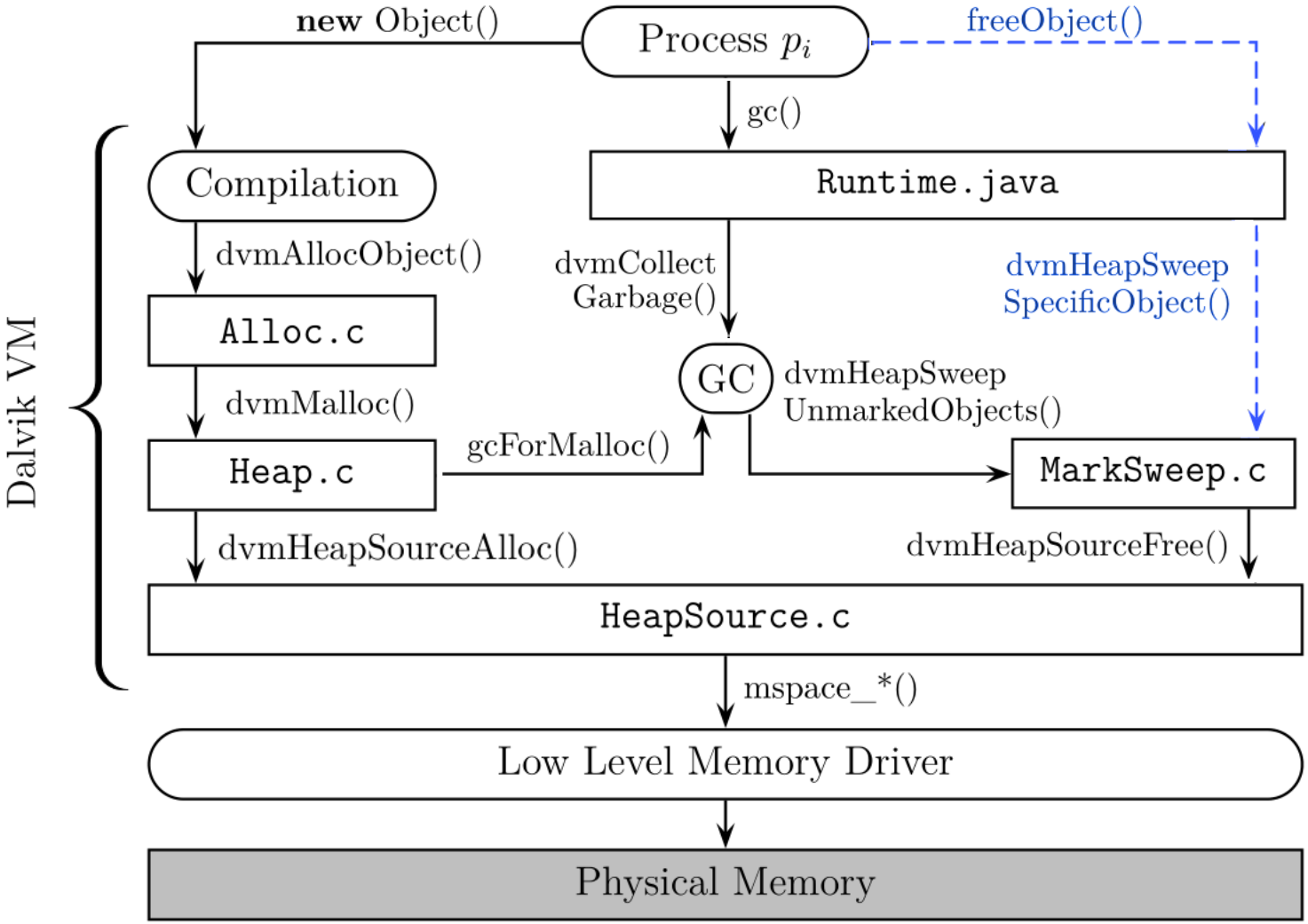
Activity Manager

- Internal process importance
  - Depends on the application class (back- / foreground)
  - Reflected in OOM adjustment values adj_p
- Built-in OOM process killer
  - Killing “unimportant” processes first
  - Memory thresholds mem_t and corresponding levels adj_t
- Example with $mem_3 = 20\ MB$ and $adj_3 = 7$
  - Terminate processes with $adj_p \geq 7$ on $mem_{free} \leq 20\ MB$
- RT processes must get lowest possible adj_p values

Memory Management (1)

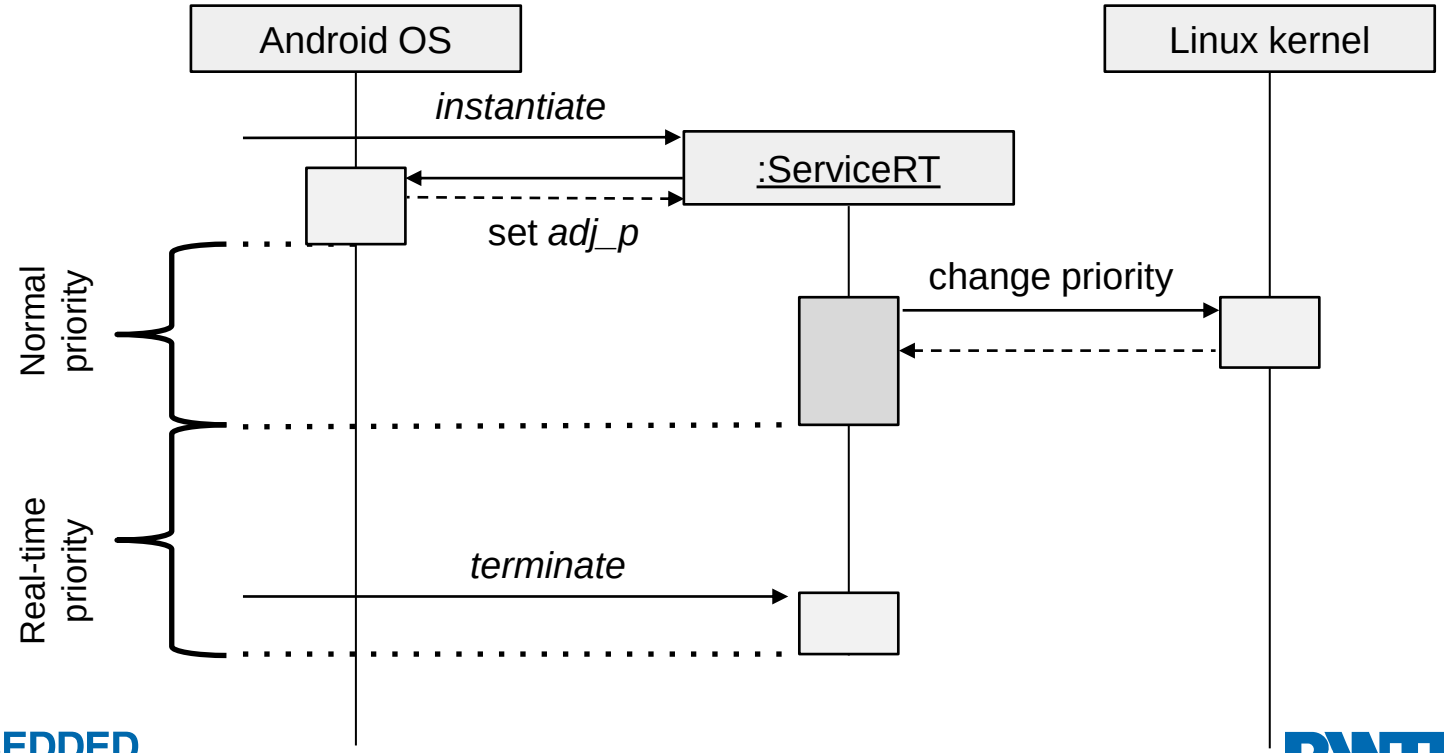
- Advantages:
  - Smart low memory process killer
  - Process-independent GC (Dalvik VM)
- Disadvantages:
  - Mark-and-sweep algorithm
  - Execution of all threads is suspended (up to 200 ms)
- No reliable prediction of process behavior
- Explicit allocation control

Memory Management (2)



Programming Interface

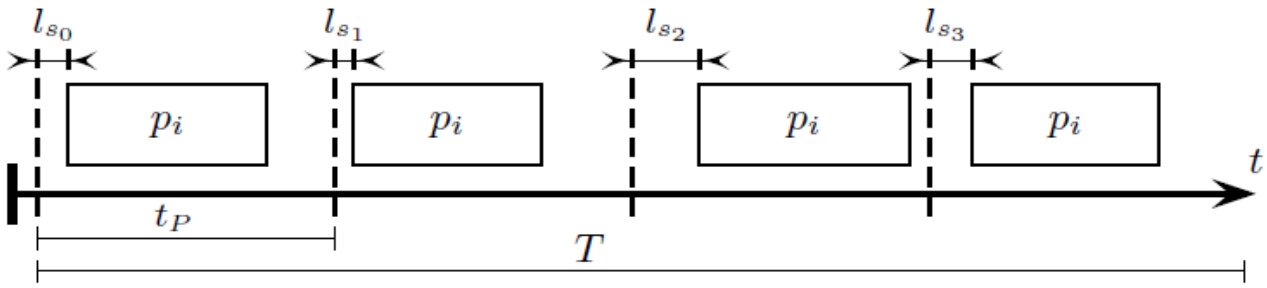
- Introducing new class *ServiceRT.java*
  - Extends Android's native Service.java class
  - API for priority selection for own process
  - API for explicit memory deallocation



Evaluation

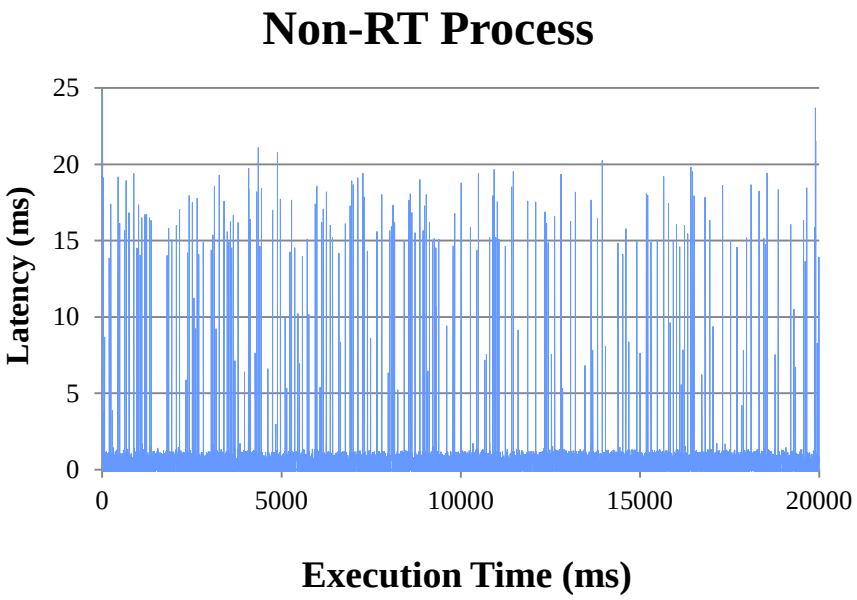
- Testing device: HTC Dream / Google G1
- Background service based on ServiceRT class

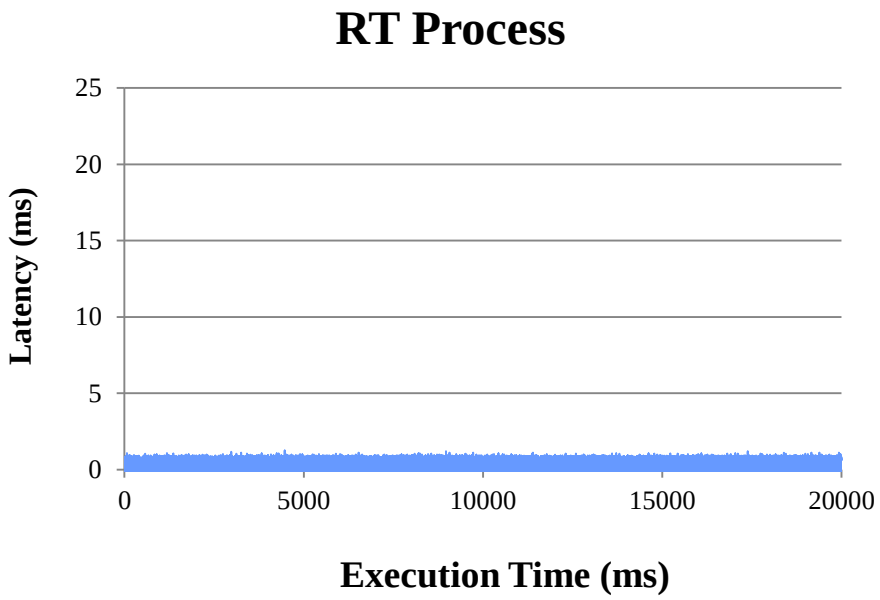
- Test 1: Periodic execution
  - Compare scheduled & actual execution time
  - Period time $t_P = 1\ ms$ to $t_P = 1\ s$
  - Running time $T = 20\ s$ to $T = 1\ h$
  - Different process priorities
  - Idle system or high CPU load



Latencies (1)

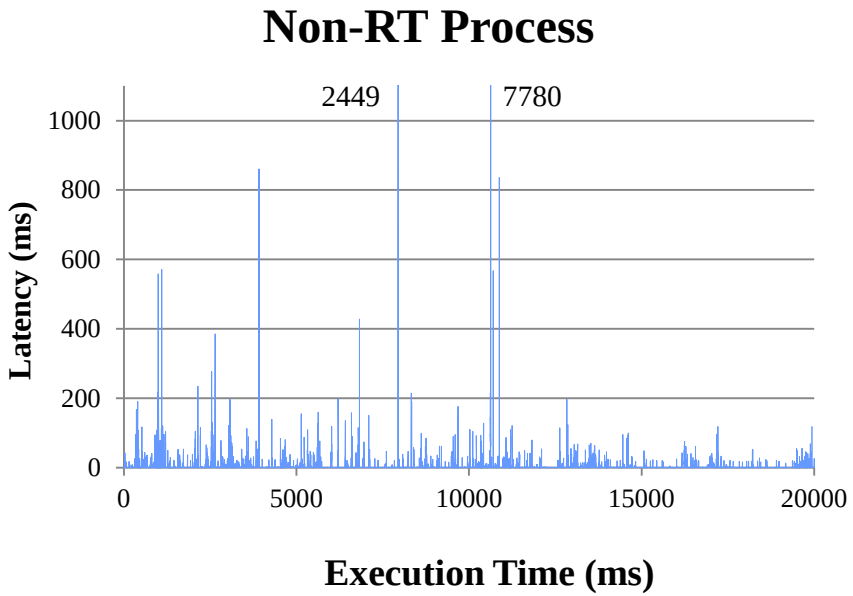
- System state: **idle**
- $t_P = 5\,ms$
- $T = 20\,s$
- Priorities: 120 (default) vs. 40 (real-time)

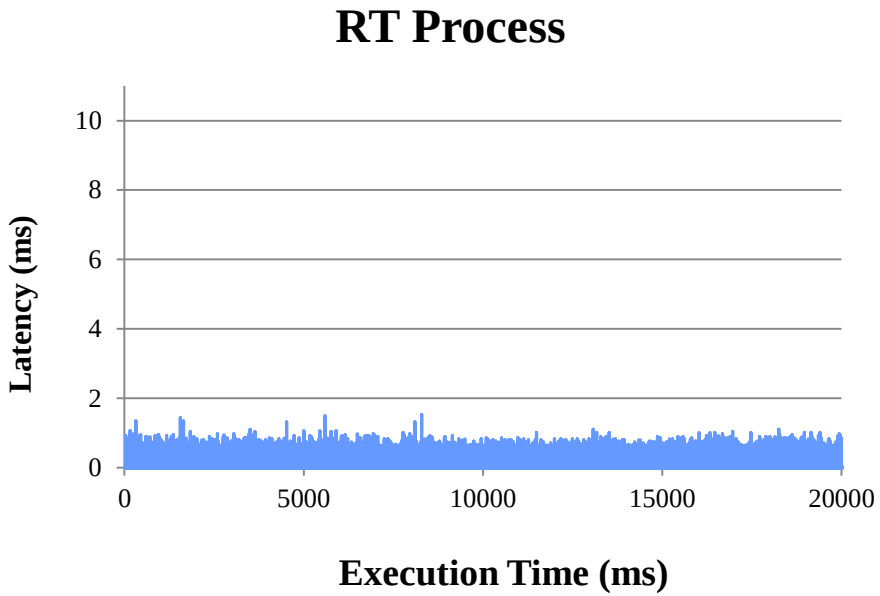




Latencies (2)

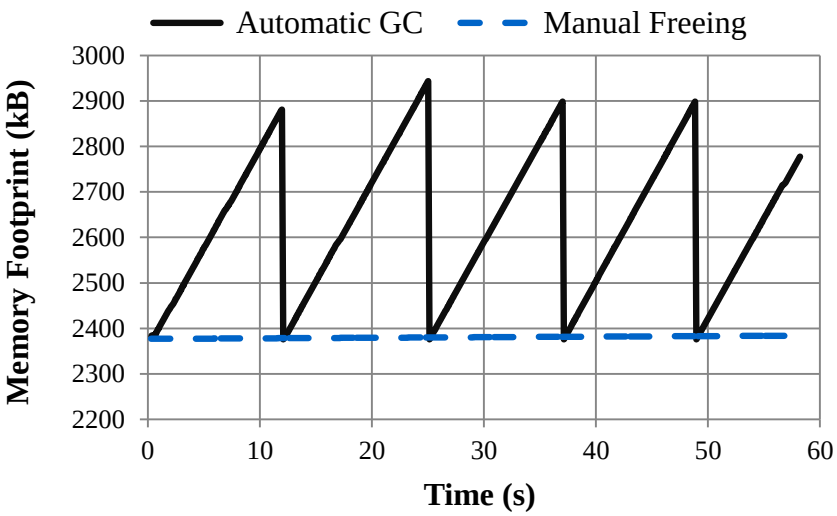
- System state: **under load**
- $t_P = 5\,ms$
- $T = 20\,s$
- Priorities: 120 (default) vs. 40 (real-time)



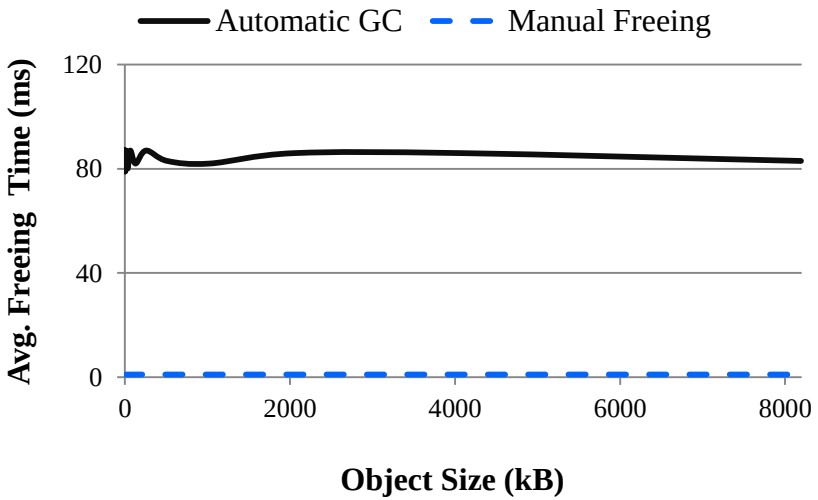


Evaluation: Memory Management

- Test 2: Continuous data receiving
  - Over a 54 Mbit Wi-Fi connection
  - Count: 2000 packets
  - Size: ~ 1 kB each
  - Can be released after processing



- Test 3: Explicit object deallocation
  - Allocate an object of a given size
  - Free it immediately
  - Measure the elapsed time
  - Calculate average of 10 cycles
  - For different sizes: 1 Byte to 8 MB



Conclusion

- New approach
  - Patched Linux kernel & Android components
  - Handling of OOM adjustment values
  - Use Linux real-time priorities for Android applications
  - Explicit memory management

- Evaluation
  - Avoiding undesired invocations of the GC
  - Scheduling latency < 2 ms for real-time processes

- BUT: explicit memory management is not enough
  - Dangling pointers, background allocations

Future Work

- Automatic, non-blocking real-time GC algorithms
  - M. Schoeberl, W. Puffitsch
    "*Nonblocking Real-Time Garbage Collection*"
  - Y. Levanoni, E. Petrank
    "*On-The-Fly Reference-Counting Garbage Collector for Java*"

- Using RT-Linux high resolution timer for periodic tasks
  - Better scheduling latencies (WC ≈ 500 us)

- More about the project will be available soon under
  https://git.embedded.rwth-aachen.de/rtandroid

Thank you for your attention!

Questions / Comments?