Work-In-Progress: Real-Time RPC for Hybrid Dual-OS System

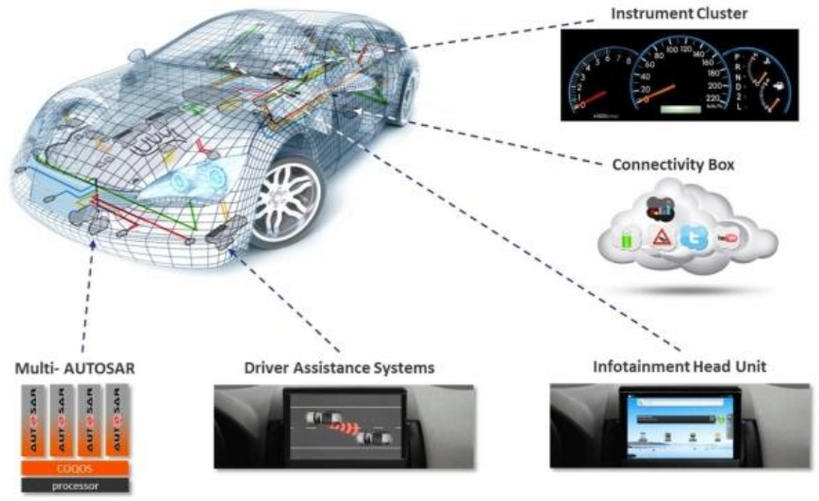
Pan Dong, Zhe Jiang, Alan Burns, Yan Ding, Jun Ma

- Background & Motivation
- Problem Statment
- Design Philosophy
- Experimental Evaluation
- Conclusions

- Background & Motivation
- Problem Statment
- Design Philosophy
- Experimental Evaluation
- Conclusions

Background & Motivation

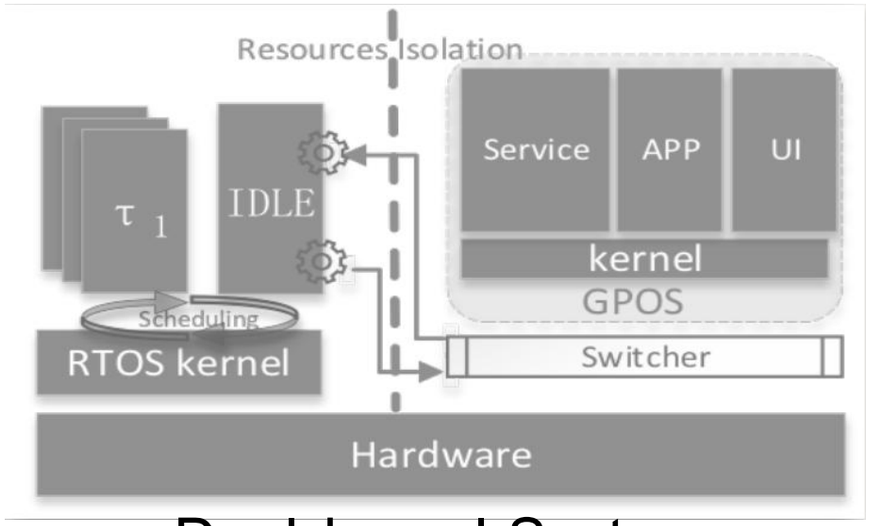
- **hybrid multi-OS system** (**HMOS**): Integrating components with different levels of criticality on one physical platform for optimizing cost, space, weight, heat generation and power consumption.



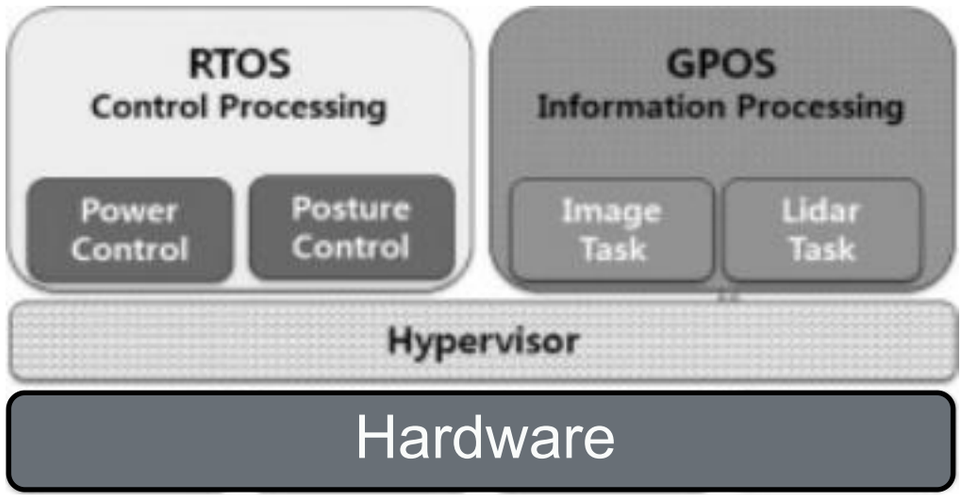
- Examples:
  - AUTOSAR 4.0+ -- ECUs (Electronic Control Units) , ADAS (Advanced Driver-Assistance System) and IVIS (In-Vehicle Information System) on a vichle
  - ARINC 653 -- flight control systems, environment control systems, and amu sement syste-ms on modern aircraft

Background & Motivation

- Consolidation Method
  - Virtualization: bad efficiency and predictability
  - Hardware Supported Isolation: better performance
- Simplest Form
  - hybrid dual-OS system (HDOS)
    - RTOS +GPOS



Dual-kernel System



Virtualization System

Background & Motivation

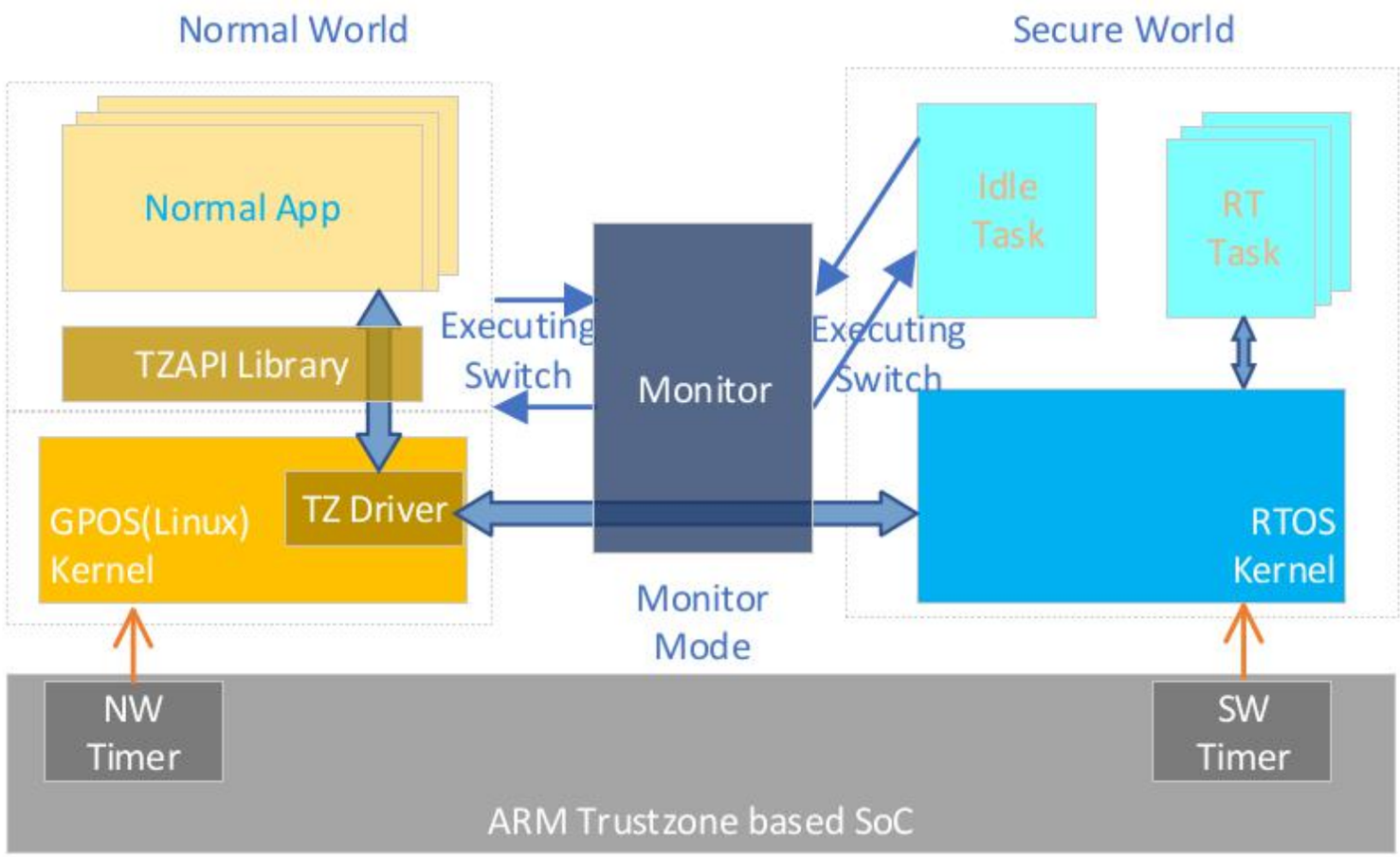
- Significant Effect of Inter-operation for HMOS
  - All sides get functions and performance enhance
  - One plus one is greater than two

- Example
  - From inter-operation, the ECU cluster in the RTOS can acquire abundant functionalities (fault logs, Cloudside AI decision) from the IVIS in the GPOS.
- Communication is the foundation of inter-operation
  - Security/safety and efficiency (time predictability)
  - RPC is the fundamental mechanism of communication

Background & Motivation

- Current optimizations for RPC fail to satisfy time predictability
  - mostly designed for virtualization systems
  - simplify under protocol stack, use shared-memory (XenLoop, MemPipe)
  - straightforward RPC with hardware assistants (XENRPC, SafeG)

Review of TZDKS



Design Idea: combines strong points of dual-kernel and virtualization by utilizing TrustZone technology

Normal world stack: GPOS (Linux) and applications.

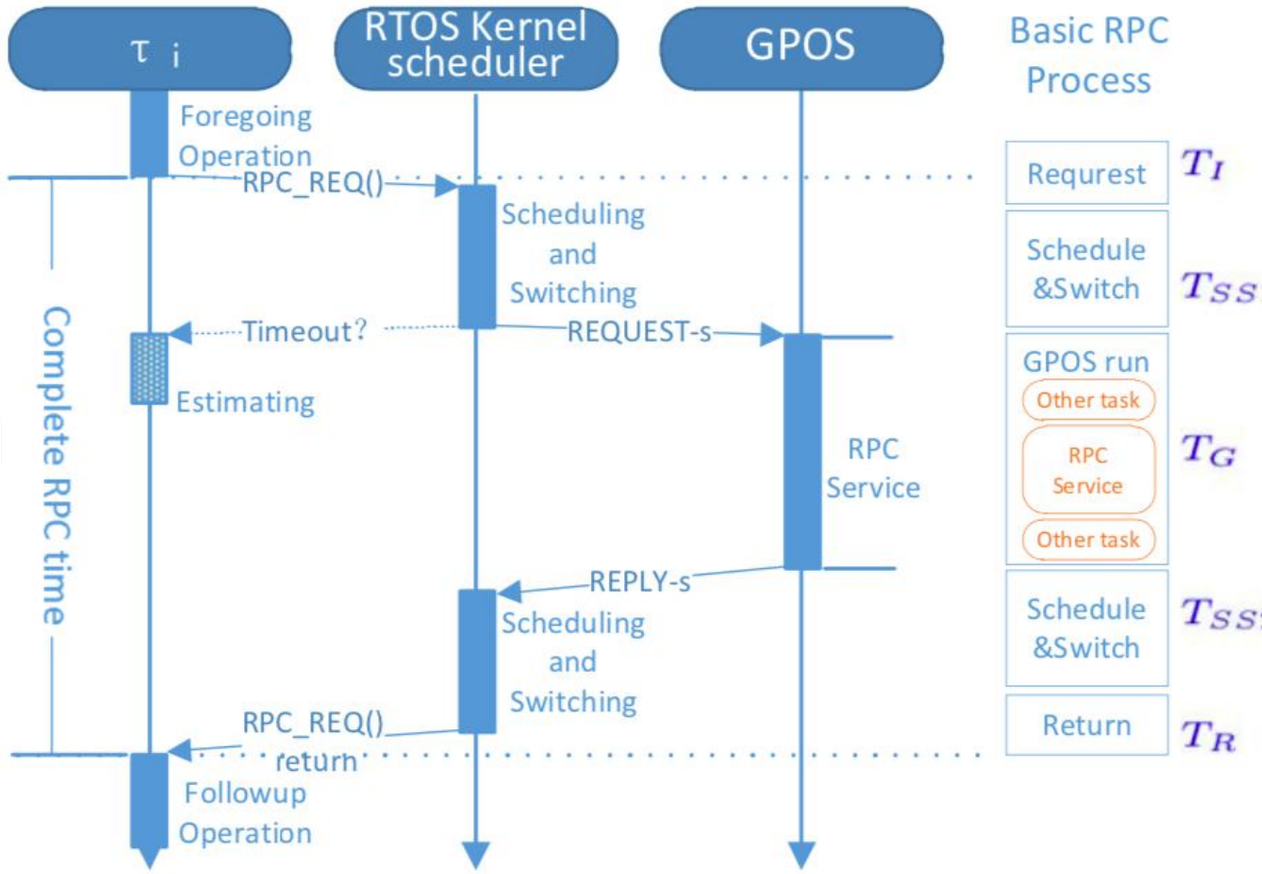
Secure world stack: monitor module, RTOS (μcOSII) and RT tasks.

Use open source project 'Trusted firmware' as foundation.

- Background & Motivation
- Problem Statment
- Design Philosophy
- Experimental Evaluation
- Conclusions

PROBLEM STATEMENT

- Mainly concern RG-RPC: RTOS → GPOS
- An abstracted process of RG-RPC:
  - ŵ Issuing RPC request
  - Ŷ Scheduling&switching of RTO
  - ŷ GPOS executing
  - Ÿ Another scheduling&switching
  - Ź Return of RPC
- We define them aș
  - T_I, T_{SS1}, T_G, T_{SS2}, T_R



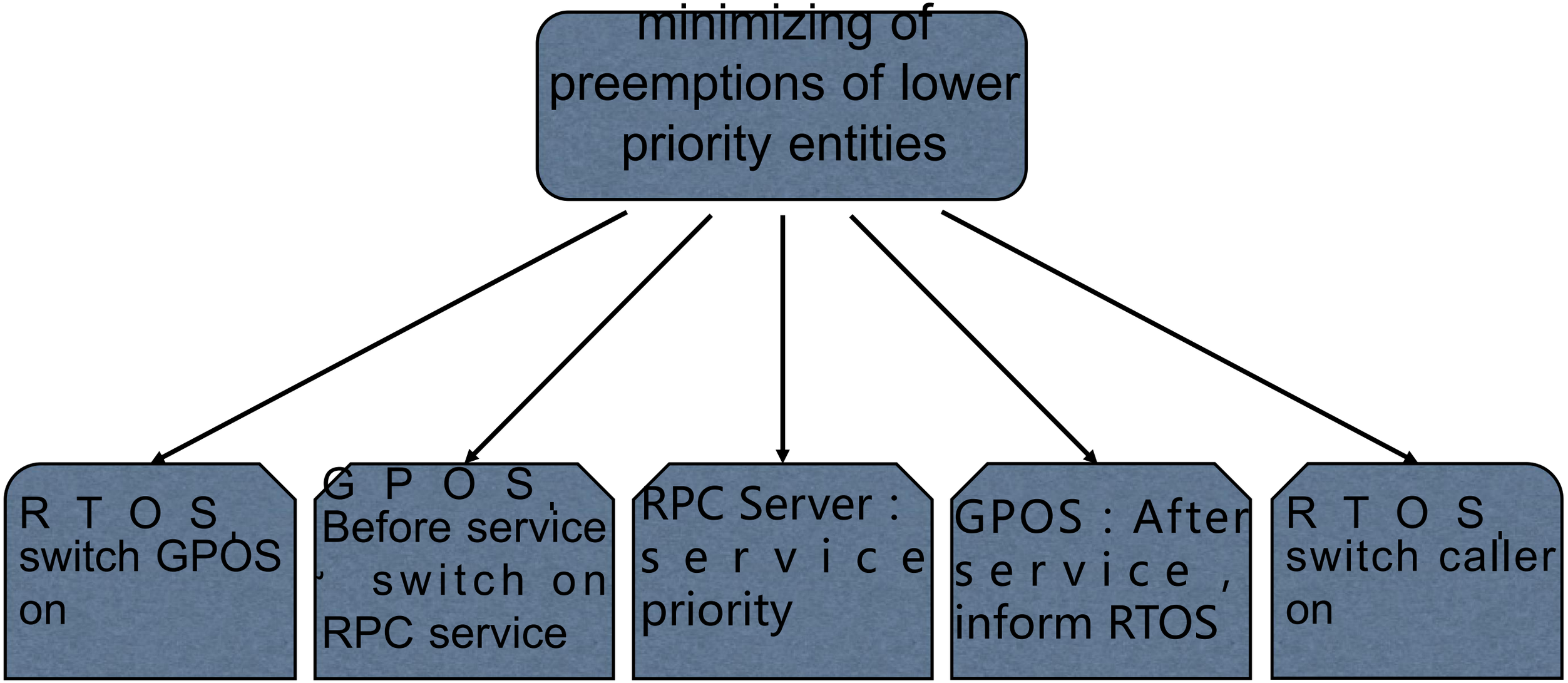
PROBLEM STATEMENT

- Two aspects should be taken into account in the design

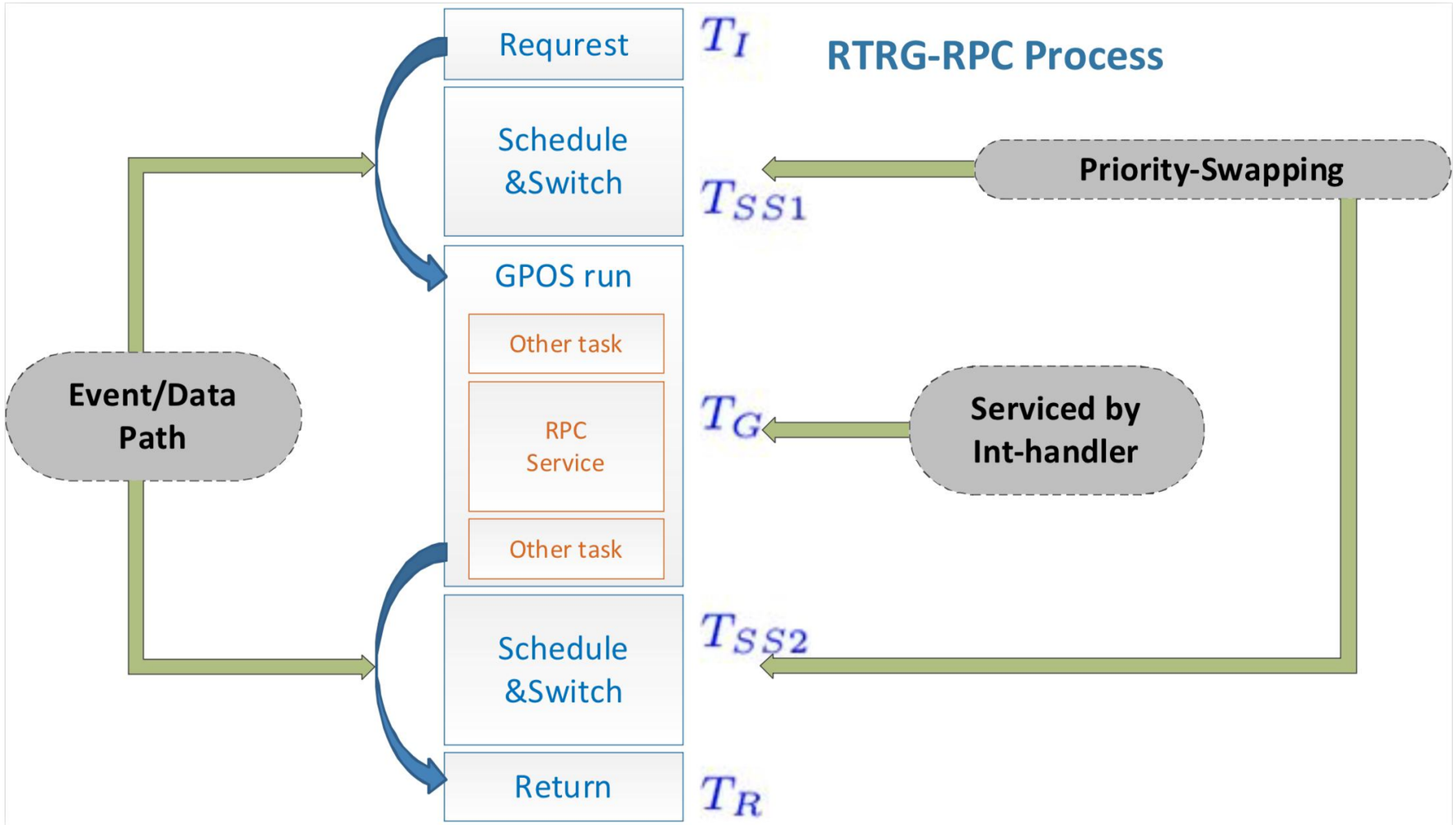
  - Efficiency
    - Three main reasons make the time predictability difficult. GPOS low priority, RPC service not deterministic, lack preemption for RPC.
  - Security
    - For safety, each communication must not lead to or propagate hazards and faults.
    - For security, a malicious task can not threat other OSs or get private information through a deliberate message.

- Background & Motivation
- Problem Statment
- Design Philosophy
- Experimental Evaluation
- Conclusions

Design Philosophy



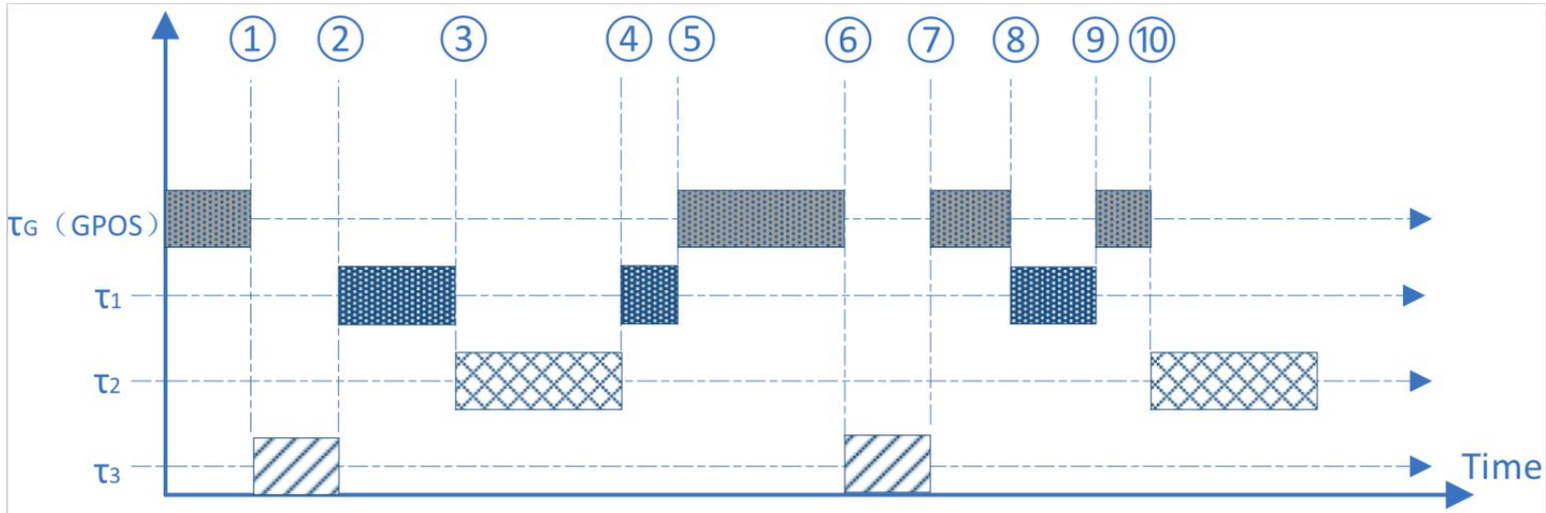
Efficiency Design of RTRG-RPC



- Three main mechanisms for time predicability:
  - priority-swapping
  - SGI (Software Generate Interrupt) messages transforming
  - interrupt handler RPC serving

Efficiency Design of RTRG-RPC

- Priority-Swapping Enhanced Idle-Scheduling Policy
  - Idle-scheduling plus τG. add another RT task τ_G serving as a GPOS container. τ_G owns a very low original but variable priority.
  - Priority-Swapping. When a regular task τ_i has triggered a RG-RPC call, it turns to sleep and exchange its priority with τG.
  - Timeout-exit strategy for τG. A timeout value can be set for τ_i. In case of no RPC returning, τ_G will be suspended and all priorities will be restored.



Tasks runtime figure for τ_1, τ_2, and τ_3, where τ_1 has a RT-RPC action. The priorities are ordered as $\tau_1 < \tau_2 < \tau_3$.

①: idle-scheduling before this time then τ_3 arrives

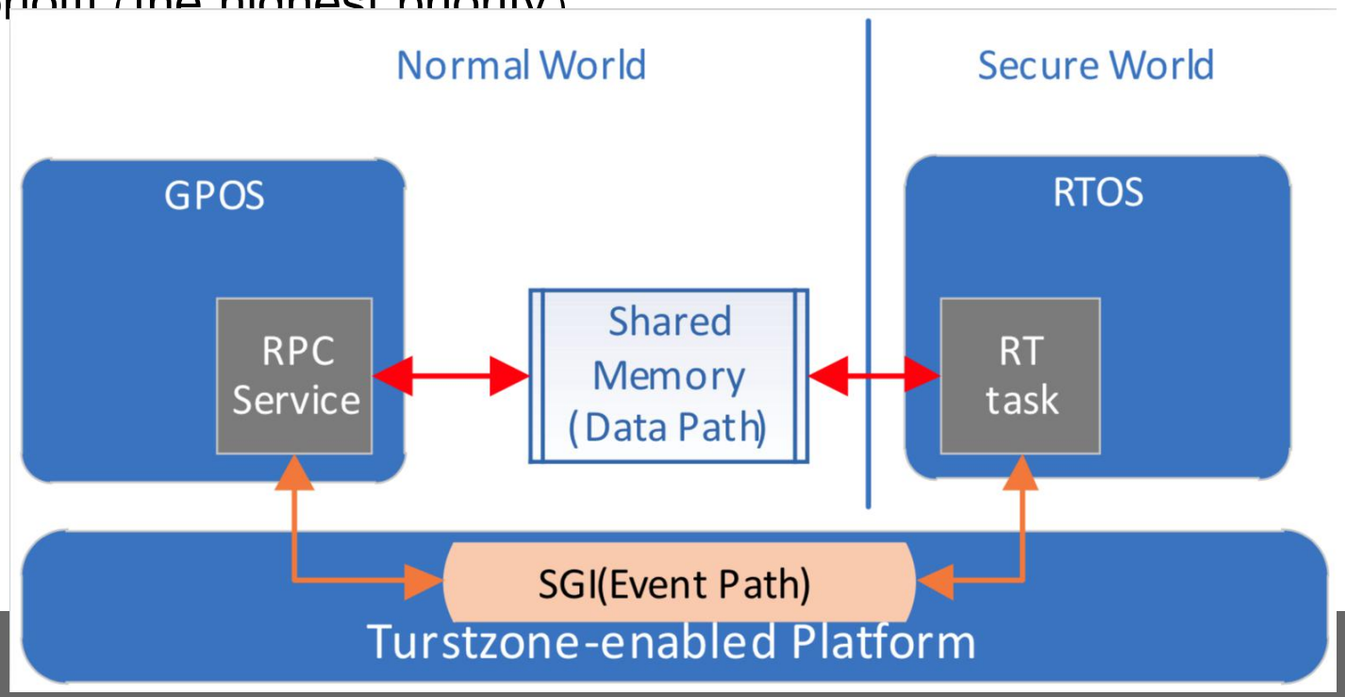
②: normal schedule ③ ⑥: preemption ④ ⑦: end of preemption

⑤: τ_1 evoke the RPC and sleep, then τ_G takes priority of τ_1 and runs

⑧: RPC completes and τ_1 is awake ⑨: idle-scheduling ⑩: τ_2 arrives

Efficiency Design of RTRG-RPC

- RTRG-RPC Commnication Path
  - Event Path: software interrupt is chosen as the event method for RTOS. RPC service notifies GPOS by a SMC call directly.
  - Data Path:
    - shared memory
    - a request pool and an answer pool
    - pool-head for maintenance. Index links to the slot number, value remarks the RPC priority
    - minimum prio[i] (the highest priority)



Efficiency Design of RTRG-RPC

- RTRG-RPC Service implementation in GPOS
  - interrupt handler serves RG-RPC in the GPOS kernel
  - Considering the system efficiency affected by long hard-irq critical region, we can place the RPC service into a high priority soft-irq
  - make the service time determinable by increasing the priority of the interrupt related to RG-RPC, and by simplifying the procedure of RPC service into a kernel module
  - GICv3 hardware guarantees that the unmasked interrupt with the highest priority will be firstly sent to the CPU core in bounded time.

Security Design of RTRG-RPC

- three types of threat are considered
  - safety threat: no side-effect of running, switch and restoring of any OS
    - RPC no-return. time-out exit mechanism
    - RPC wrong return. solved by the protocols or ways on the upper soft layer
    - wrong order of RPCs. task ID attached in each RPC solves it. One time one RPC
  - malicious code threat: prevent the executing of code in the buffer memory
    - leverage the DEP (Data Execution Protect) to forbid the code executing
- DoS attack threat
  - set up a counter in the RPC service handler of RTOS to test the frequency of calling from GPOS
  - If the calling frequency exceeds a predefined threshold, RTOS will deprive some execution ticks from GPOS

- Background & Motivation
- Problem Statment
- Design Philosophy
- Experimental Evaluation
- Conclusions

Evaluation

- **Experimental Platform**

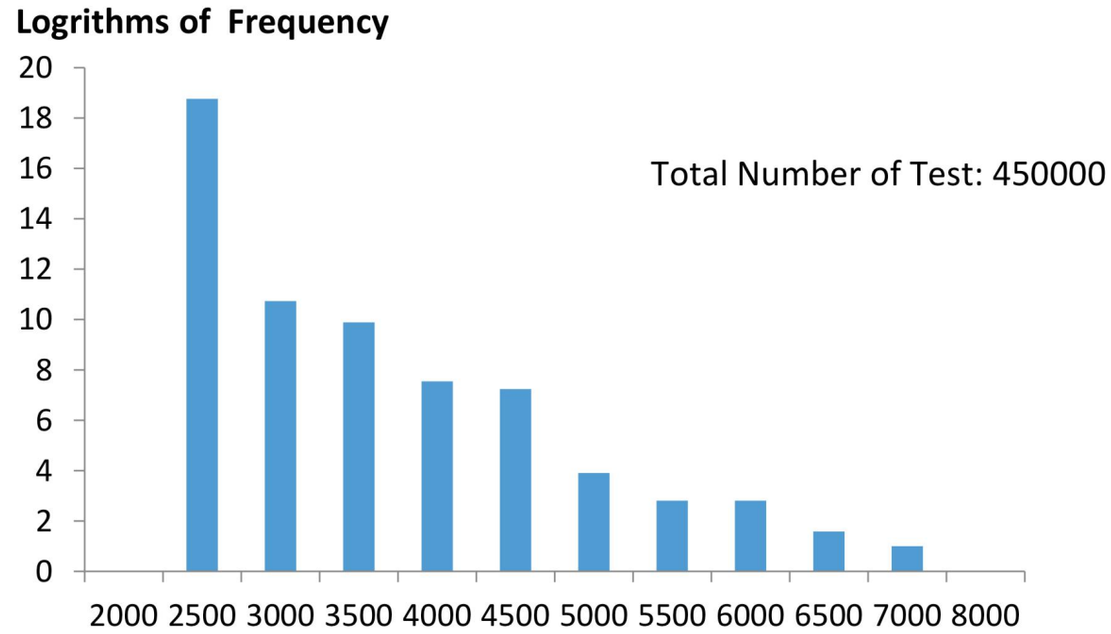
- ARMv8 Foundation Fast-Model Platform
  - FFP, version FM000-KT-00035-r11p1-24rel2
  - quad-core Cortex-A53 CPU
  - 2GB RAM memory
  - peripherals (Watchdog timer, Real-time timer, and Power controller)
  - secure pe- ripherals (Real-time clock, Trusted watchdog, Random number generator)
- We omit the service code in GPOS so that the service returns a SMC message immediately without any business function.

- Two metrics were evaluated:
  - Latency Predictability and Distribution
  - Latency Comparison

Evaluation

- **Latency Predictability and Distribution**

- There are three periodic tasks τ , τ_1 , τ_2 in RTOS and only τ requires RPC.
- RTOS is schedulable under FPS policy., total load less than 69%, UnixBench as a payload in GPOS.
- In the first experiment, τ owns highest priority. RPC from τ for 450000 times



- 99.3%+ calls complete in 2500 cycles, use logarithms of occurrence as Y-axis scale
- all RPCs complete in 8000 cycles, and only 3 calls exceed 6000 cycles.

Evaluation

- **Latency Predictability and Distribution**

- In the second experiment, consider the preemption by tasks with higher
- assign the lowest priority to τ, and test the latency of RPC from τ
- comparison of maximum, minimum, average latency, and the mean-square error of latency for RTRG-RPC

TABLE I
RTRG-RPC LATENCY IN DIFFERENT PRIORITIES (UNIT: CYCLES)

	Max	Min	Average	MSE
Highest Priority	8987	2037	2079.3	176.1
Lower Priority	179463	2036	2114.8	1267.2

- the maximum latency is significantly increased
- shows that RTRG-RPC scheme does not violate priority scheduling and is still predicable in a lower priority

Evaluation

- **Latency Comparison**

- implement another two RPC policies for comparison
  - TRG-RPC: traditional RPC method without any real-time consideration
  - ITRG-RPC: enhanced version of TRG-RPC by adding the event path model and GPOS service model of RTRG-RPC

TABLE II
LATENCY COMPARISON FOR THREE RG-RPCS (UNIT: CYCLES)

	Max	Min	Average	MSE
RTRG-RPC	6867	2043	2078.4.3	95.8
ITRG-RPC	372330	2298	64485.4	108405
TRG-RPC	50497874	49798712	49911274	125573.5

- RTRG-RPC owns much higher efficiency and better predictability comparing to TRG-RPC and ITRG- RPC

- Rapid Service in GPOS do more significant help to the efficiency than Priority − Swapping

- link in the GPOS is most crucial

- Background & Motivation
- Problem Statment
- Design Philosophy
- Experimental Evaluation
- Conclusions

Conclusions & Future Work

- this paper verifies the feasibility of obtaining a real- time service from a GPOS
- Time predictability of RTRG-RPC is achieved by three mechanisms: SGI message transforming, interrupt handler RPC serving, and priority-swapping
- a number of issues of details（such as cache miss and lock waiting, etc.）within the GPOS have been ignored. Our next plan is to address these issues and to extend the system model to a distributed multi-core platform

UNIVERSITY of York

Thanks

Question & Comments ?