Optimizing Complex Software Platforms with Cross-Layer Resource Control and Scheduling

2014년 12월 19일

홍 성 수

sshong@redwood.snu.ac.kr

서울대학교 전기정보공학부 교수

가헌과학기술재단 석좌교수

차세대융합기술원 스마트시스템연구소 소장

Agenda

I. **Introduction**

II. Case study: Cross-layer Resource Control for Smartphone

III. Conclusion

Hot IT Trends (1): IoT and Cloud

- The Internet-of-Things (IoT) has emerged as one of the hottest IT trends in recent years
  - Definition of IoT [ITU-T Recommendation Y.2060, 2012]

www.itu.int/itu-t/gsi/iot

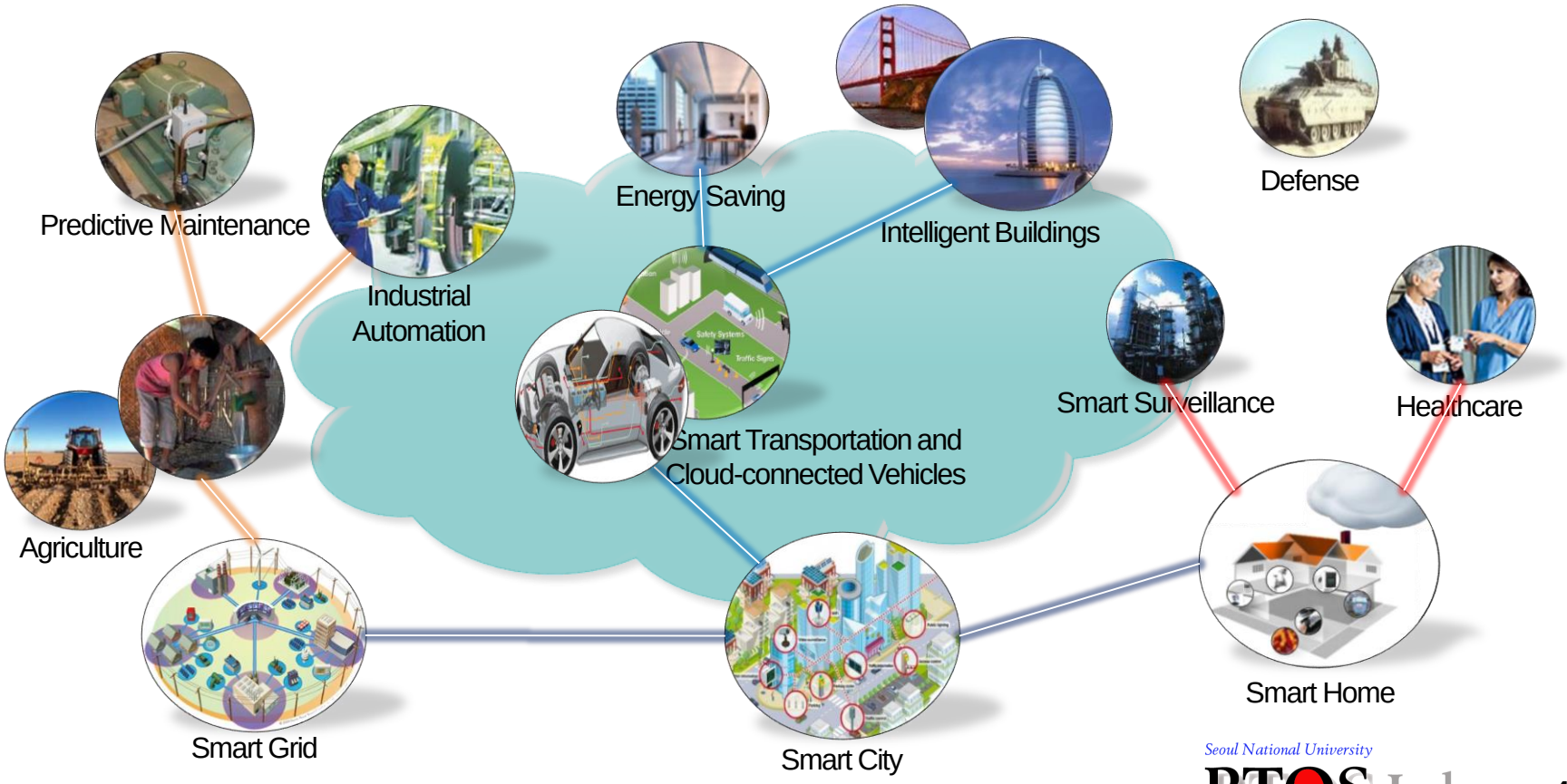
> ✓ A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies

    - ***Thing***: object of the physical world or of the information world, which is capable of being identified and integrated into the communication networks

- IoT and Cloud Computing
  - Cloud computing is “a key enabler of the IOT”

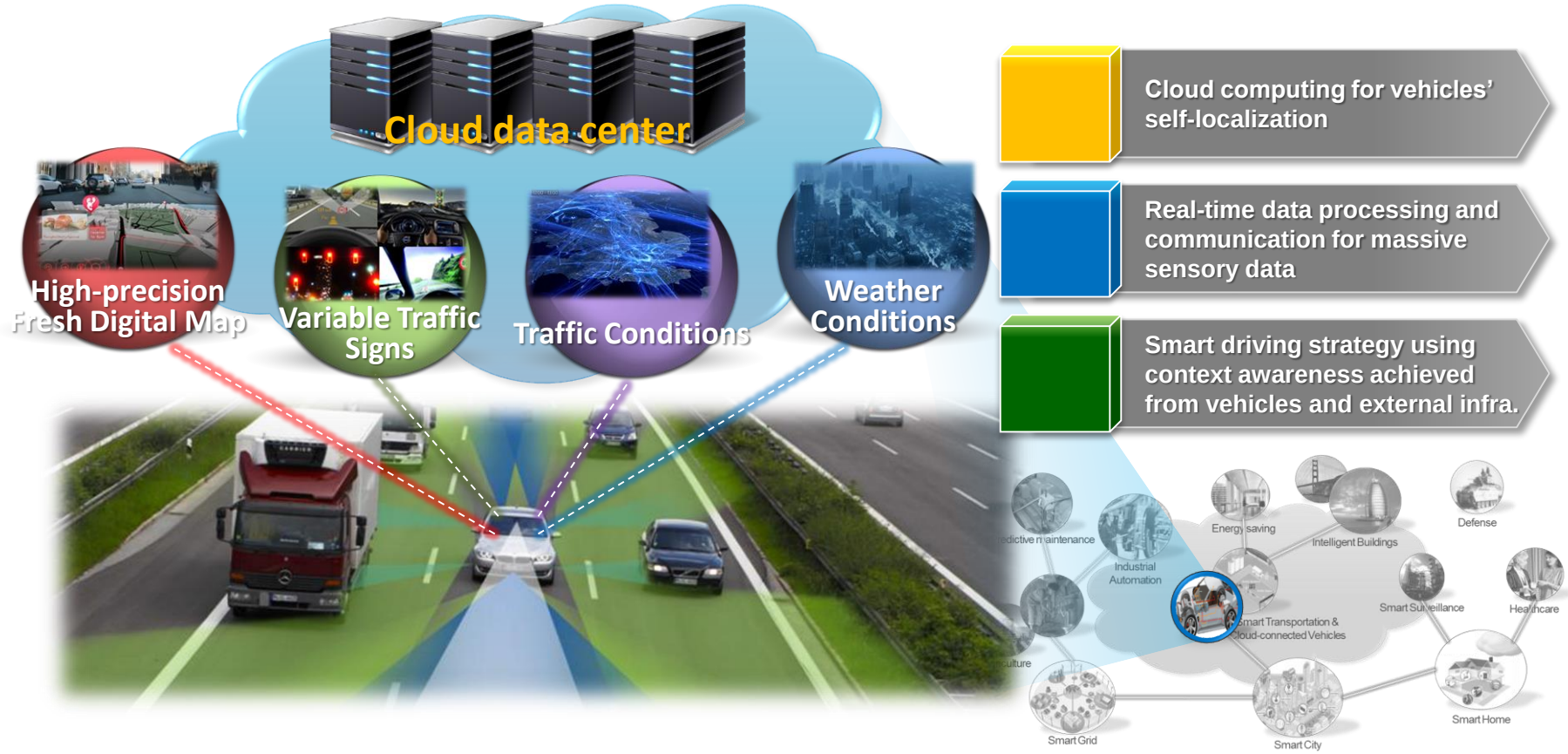
Hot IT Trends (2): IoT and Cloud

- A broad set of applications

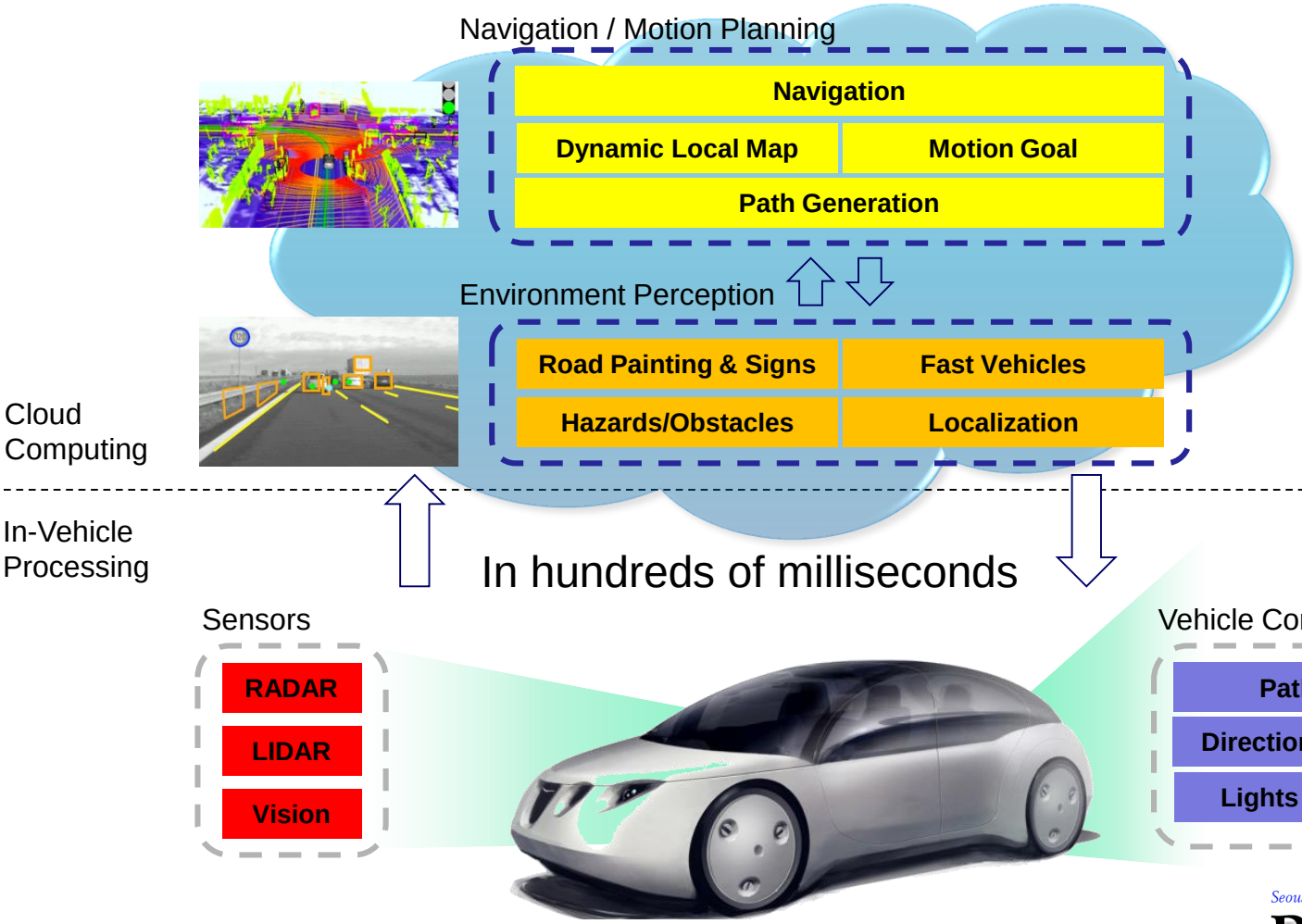


Hot IT Trends (3): IoT and Cloud

- One obvious example: cloud-connected vehicles

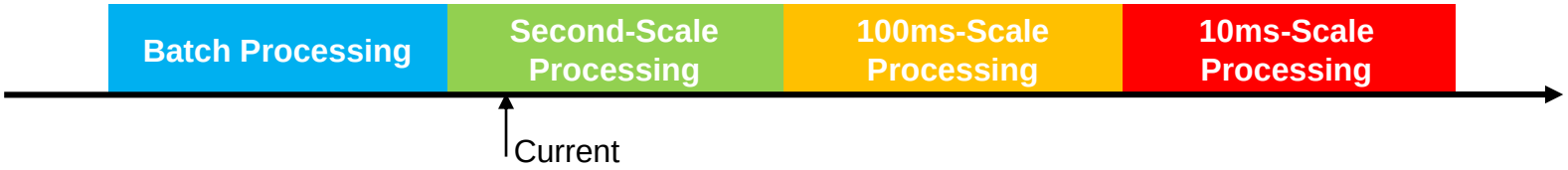


Hot IT Trends (4): Cloud-Connected Vehicle



Hot IT Trends (5): Cloud-Connected Vehicle

- Time granularity and possible services of cloud



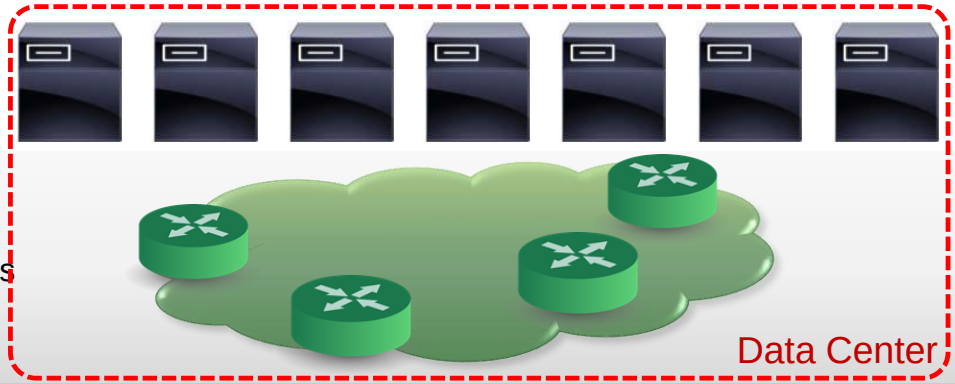
  - Batch processing
    - Database update, diagnostics
  - Second-scale processing
    - Navigation, drivability map generation
  - 100ms-scale processing
    - Motion planning, path generation
  - 10ms-scale processing
    - Vehicle controls
    - May be hard to offload from vehicles

Hot IT Trends (6): Cloud Infrastructure

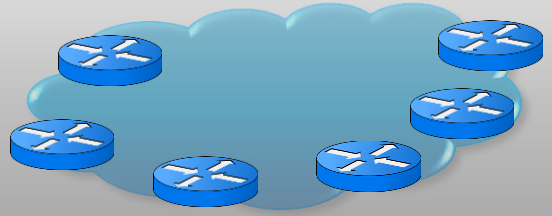
- The *"common"* infrastructure

Data Center Cloud
Application hosting, Management,
Service Provisioning

Core Network
IP/MPLS, Multicast, security
Network services, Quality-of-Services
Mobile packet core



Multi-Service Edge
3G/4G/5G/Wi-Fi
Ethernet



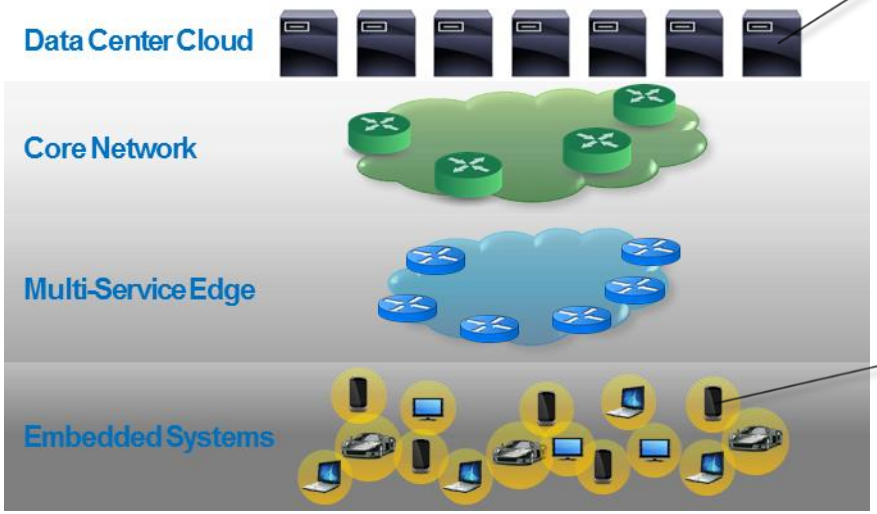
Embedded Systems
Smartphones, vehicles, machines
Computers
Consumer Electronics

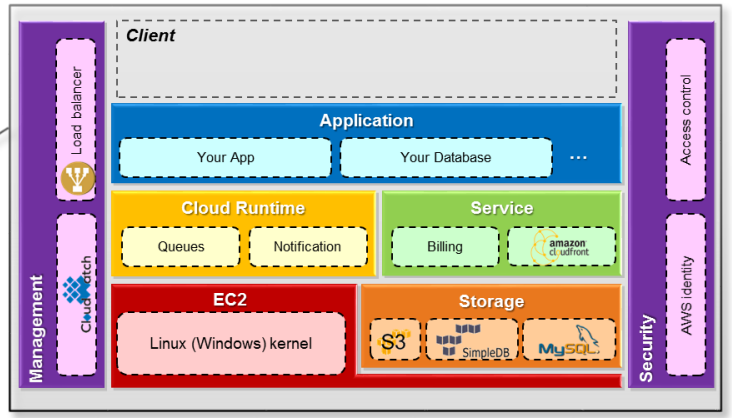
Hot IT Trends (7): Cloud Infrastructure

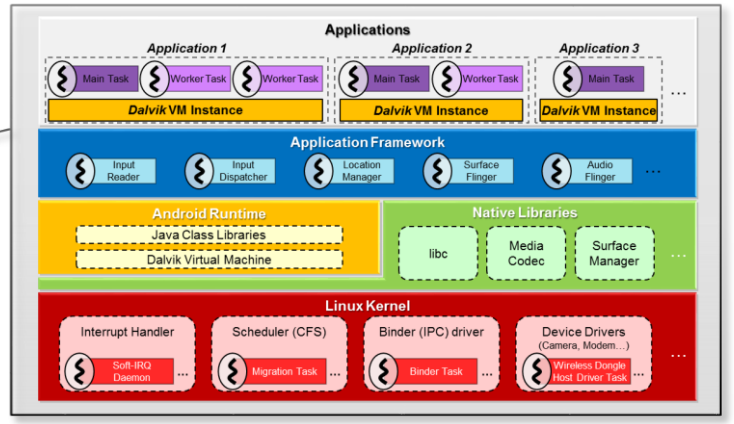
- Huge, complex and thus ***layered***

? lines of code…

But must be more huge and complex than those of smartphone





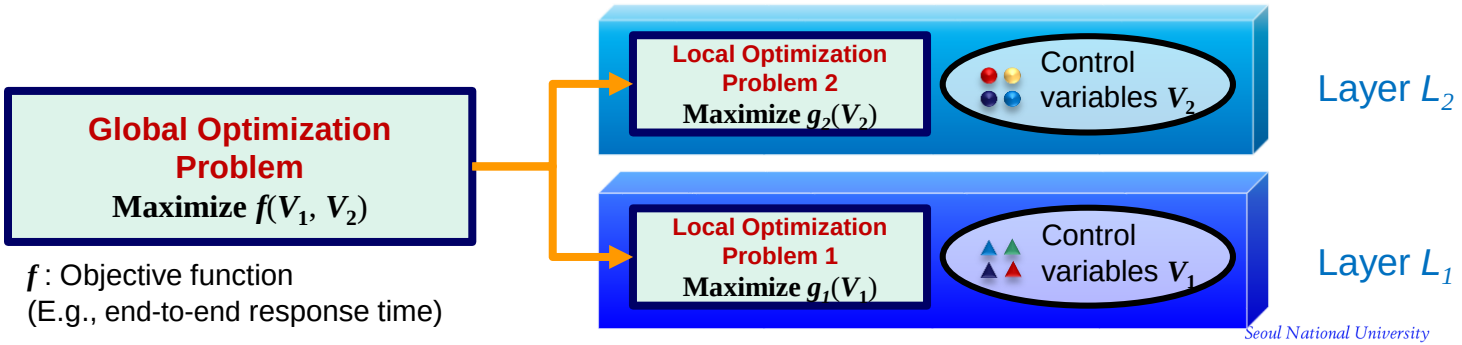


More than **12 Million** lines of code!!!

* In case of Android KitKat

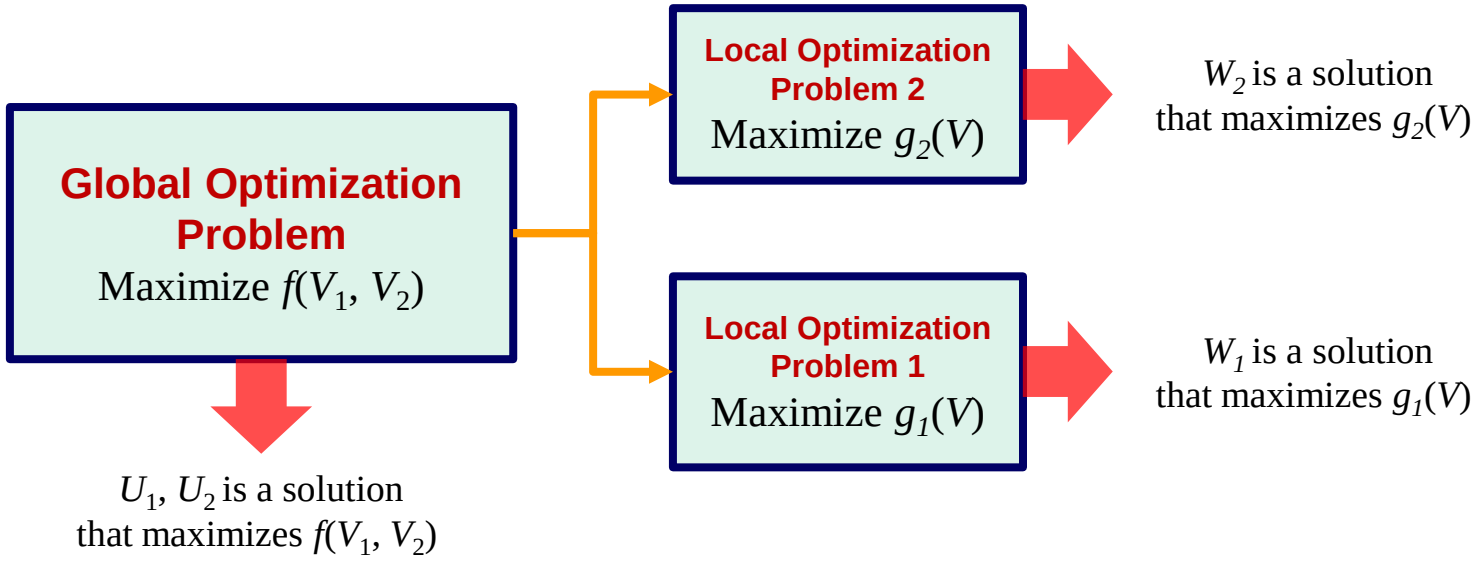
Why Layered?

- Separation of concerns
  - Decomposition of functionality
    - Each layer implements ***separated***, ***independent*** functionality of the system
      - Each layer can be developed and advanced independently from other layers as long as the interfaces are not changed
  - Decomposition of optimization
    - Each layer is a solver of a local optimization problem which is ***smaller*** than the global optimization problem



Limitation of Layered SW Structure

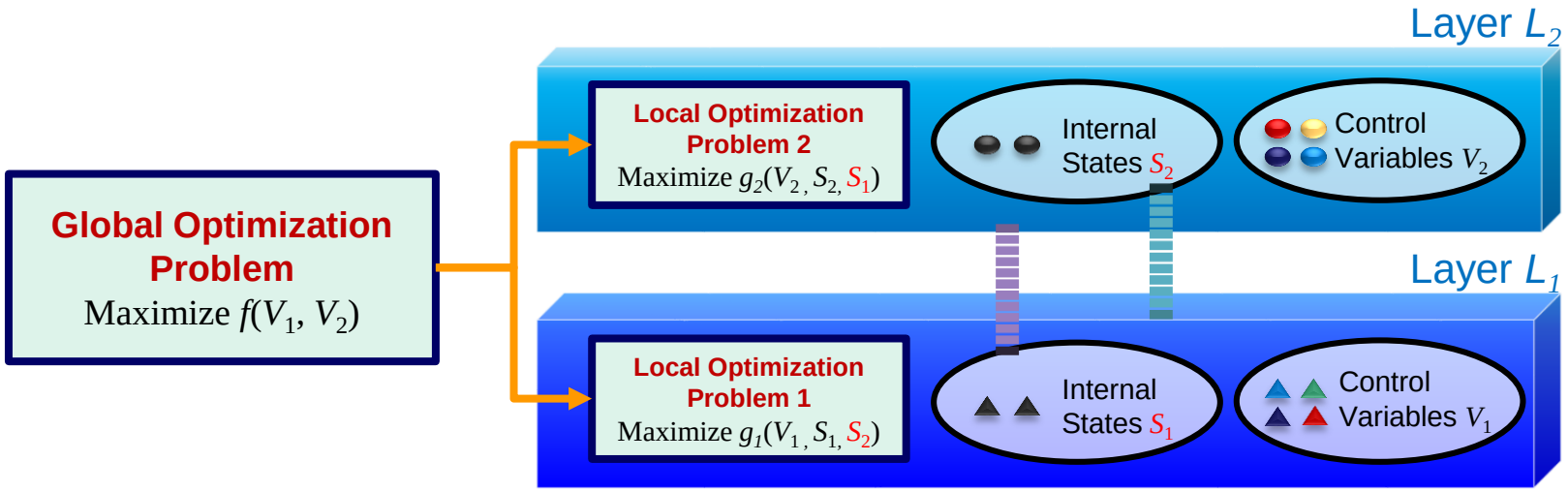
- Optimization of individual layers does not always generate the global optimal solution



$$f(U_1, U_2) > f(W_1, W_2)$$

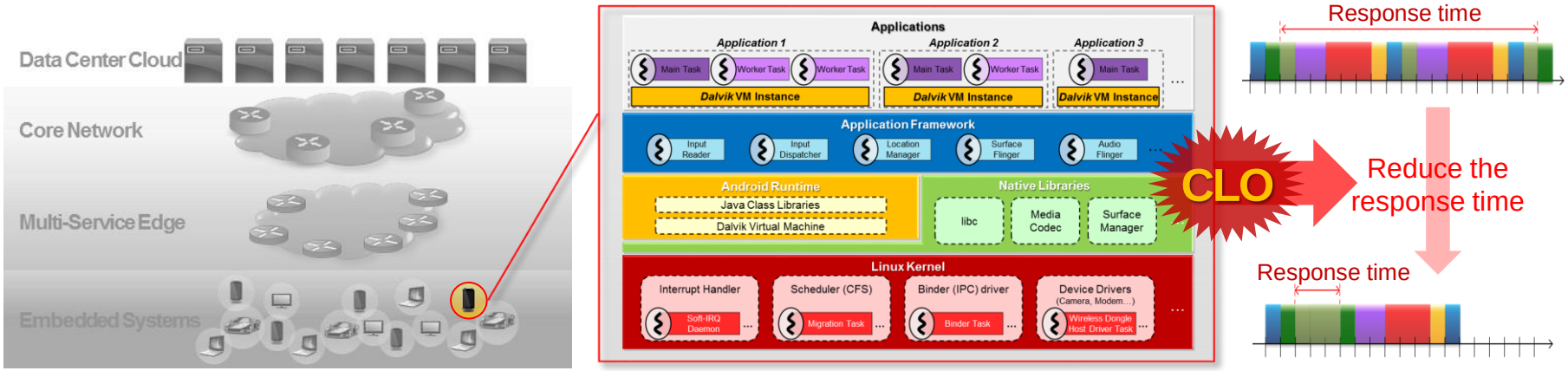
Cross-Layer Optimization (CLO)

- What is it?
  - Re-design local optimization problems such that local optima lead to the global optimum
    - To do so, internal states of a layer are selectively transferred across layer boundaries
      - Serves as constant values in the local optimization problem



In this Talk…

- Present a case study of CLO in Android platform
  - ***Objective function***: reduce the end-to-end response time of a given user input



- For more information, please refer to the following paper:
  - S. Huh, J. Yoo and S. Hong, *"Cross-Layer Resource Control and Scheduling for Improving Interactivity in Android,"* To appear in the Journal of Software: Practice and Experience, 2014*

* http://redwood.snu.ac.kr/?q=publications/international/journals/cross-layer-resource-control-and-scheduling-improving-interactiv

Agenda

I. Introduction

II. Case study: Cross-layer Resource Control for Smartphone

- A. **Introduction**
- B. Background: Android Framework and Linux Kernel
- C. Problem Description
- D. Solution Approach: FTC and VT-CFS
- E. Experimental Evaluation

III. Conclusion

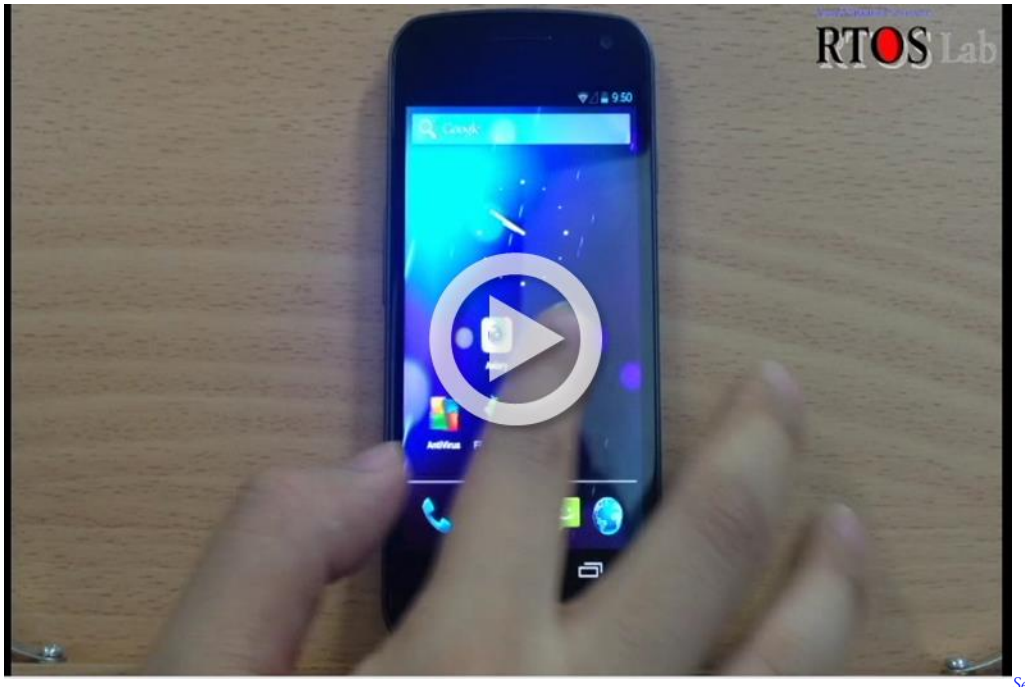
Android Becomes More Complex

- Ever growing complexity of the Android framework
  - Desired level of user experience also increases

Source: M. Comet, http://www.bonkersworld.net/robot-evolution/

User Interactivity in Android

- Android often demonstrates poor interactivity when it runs with various workload concurrently
  - In terms of responsiveness



Agenda

I. Introduction

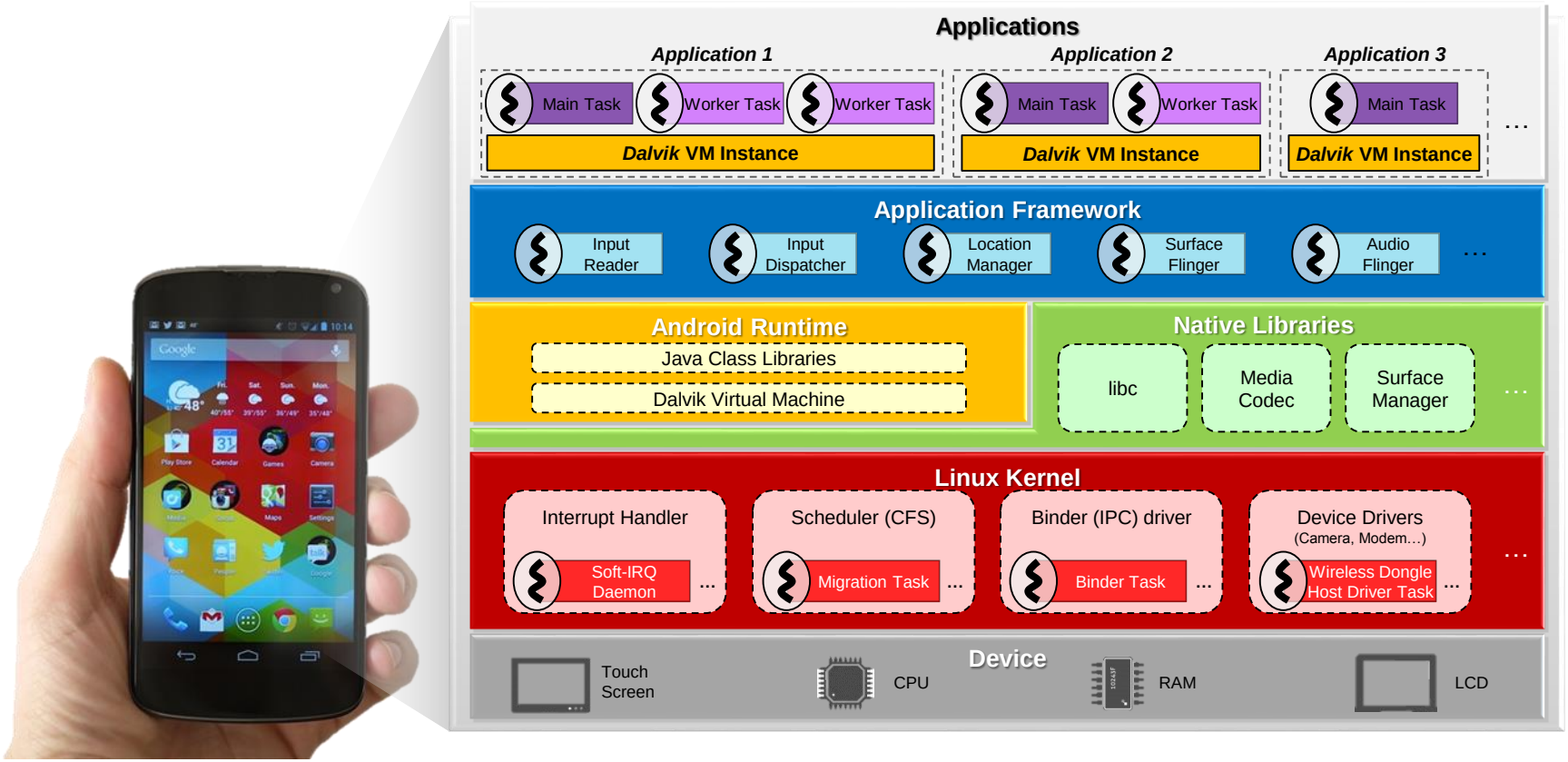
II. Case study: Cross-layer Resource Control for Smartphone

- A. Introduction
- **B. Background: Android Framework and Linux Kernel**
- C. Problem Description
- D. Solution Approach: FTC and VT-CFS
- E. Experimental Evaluation

III. Conclusion

System Architecture (1)

- Layered system architecture of the Android platform

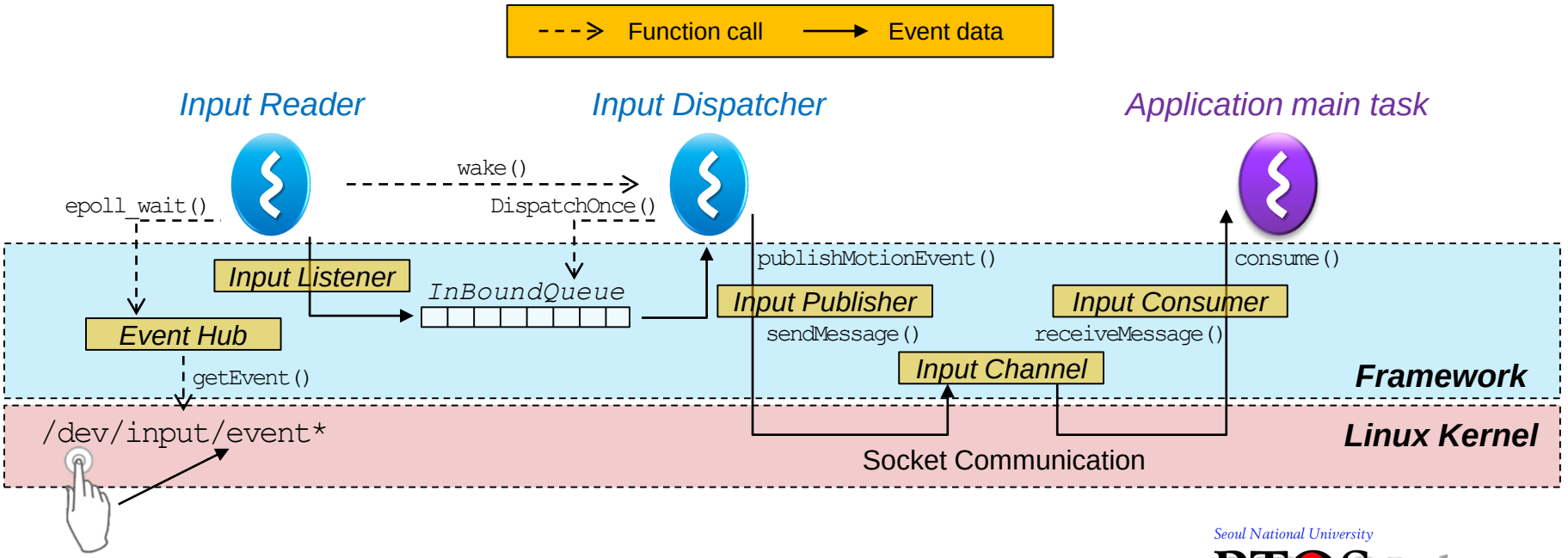


System Architecture (2)

- Task types
  - System servers
    - Responsible for a dedicated system resource administration
      - e.g., *Surface Flinger* is in charge of the frame buffer
  - Kernel tasks
    - Running for core system services
      - e.g., *Migration* task for load balancing, *Binder* task for IPC
  - Applications
    - Executed on top of a Dalvik VM instance
    - In order to access hardware devices, must use a different interface provided by a dedicated system server
    - Can be split to one main (UI) task and several worker tasks

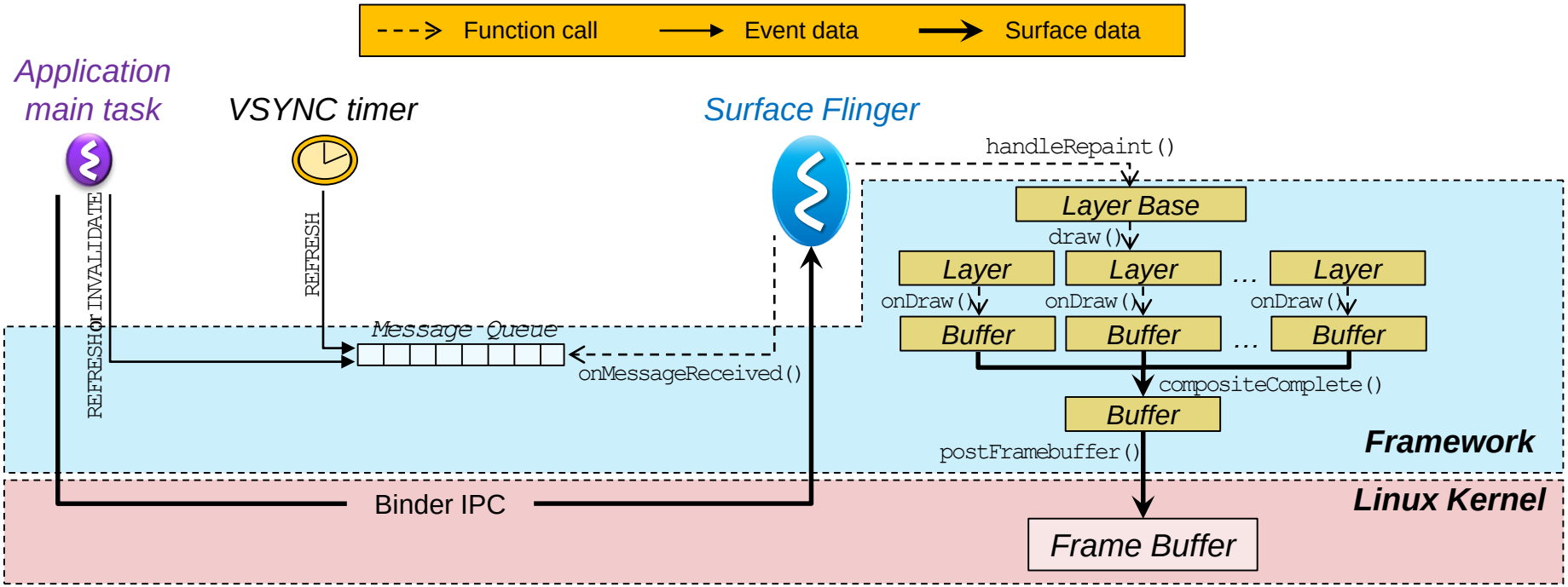
Android Runtime Behaviors (1)

- Input event handling
  - *Input Reader* retrieves raw input events and processes them
  - *Input Dispatcher* determines valid input targets and dispatches input events to them



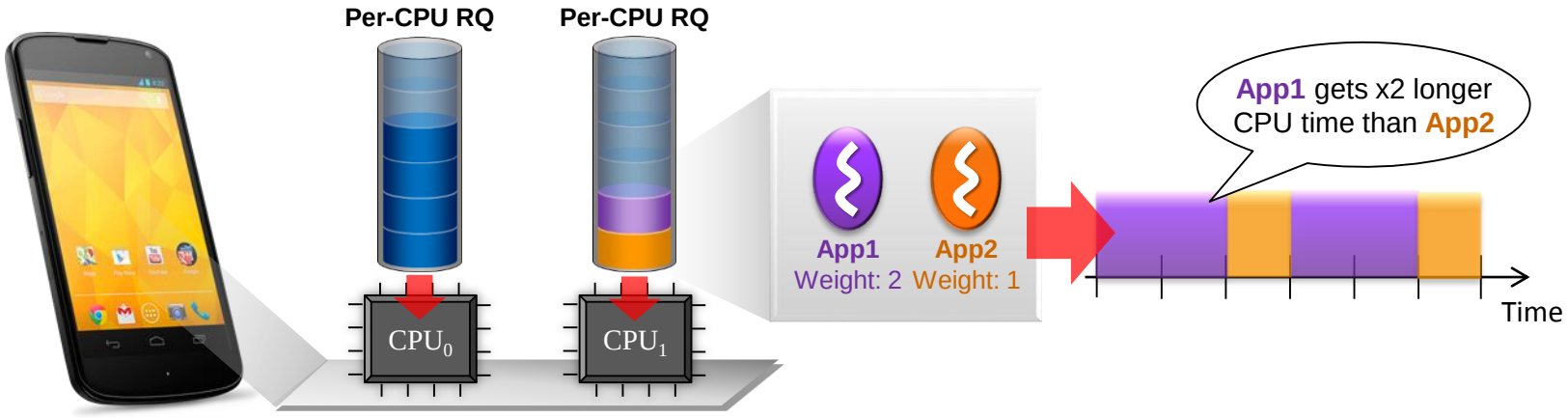
Android Runtime Behaviors (2)

- Rendering
  - *Surface Flinger* draws the contents of layers and synthesizes all the layers into a single image



Linux CFS (1)

- CFS (Completely Fair Scheduler)
  - Primary task scheduler of the mainline Linux kernel
    - Symmetric multiprocessor scheduling algorithm which maintains a dedicated run-queue for each CPU
      - Make scheduling decisions independently of each other
    - Its primary goal is to provide *fair share scheduling* by giving each task CPU time proportional to its weight



Linux CFS (2)

- Virtual runtime of task τ_i
  - τ_i's cumulative runtime inversely scaled by its weight at time t

$$V_{i(t)} = \frac{\omega_0}{W_i} \times Ci(t)$$

The weight of nice value 0

Amount of CPU time that τ_i has received for t

The weight of τ_i

    - Perfect fairness is achieved if virtual runtimes are the same among all the tasks at any given time
      - CFS approximates this by dispatching the task with the smallest virtual runtime at every scheduling decision point
  - Similarly, virtual runtime of task group g_i is defined as:

$$\tilde{V}_{i(t)} = \frac{\omega_0}{\tilde{W}_i} \times \tilde{C}_i(t)$$

Amount of CPU time that tasks in g_i have received for t

The weight of g_i

Linux CFS (3)

- Time slice of task τ_i
  - Time interval for which τ_i is allowed to run without being preempted
    - The length of a time slice is proportional to a task's weight

The weight of τ_i

$$T_i = \frac{W_i}{\sum_{\tau_j \in S} W_j} \times P$$

Set of runnable tasks in the run-queue

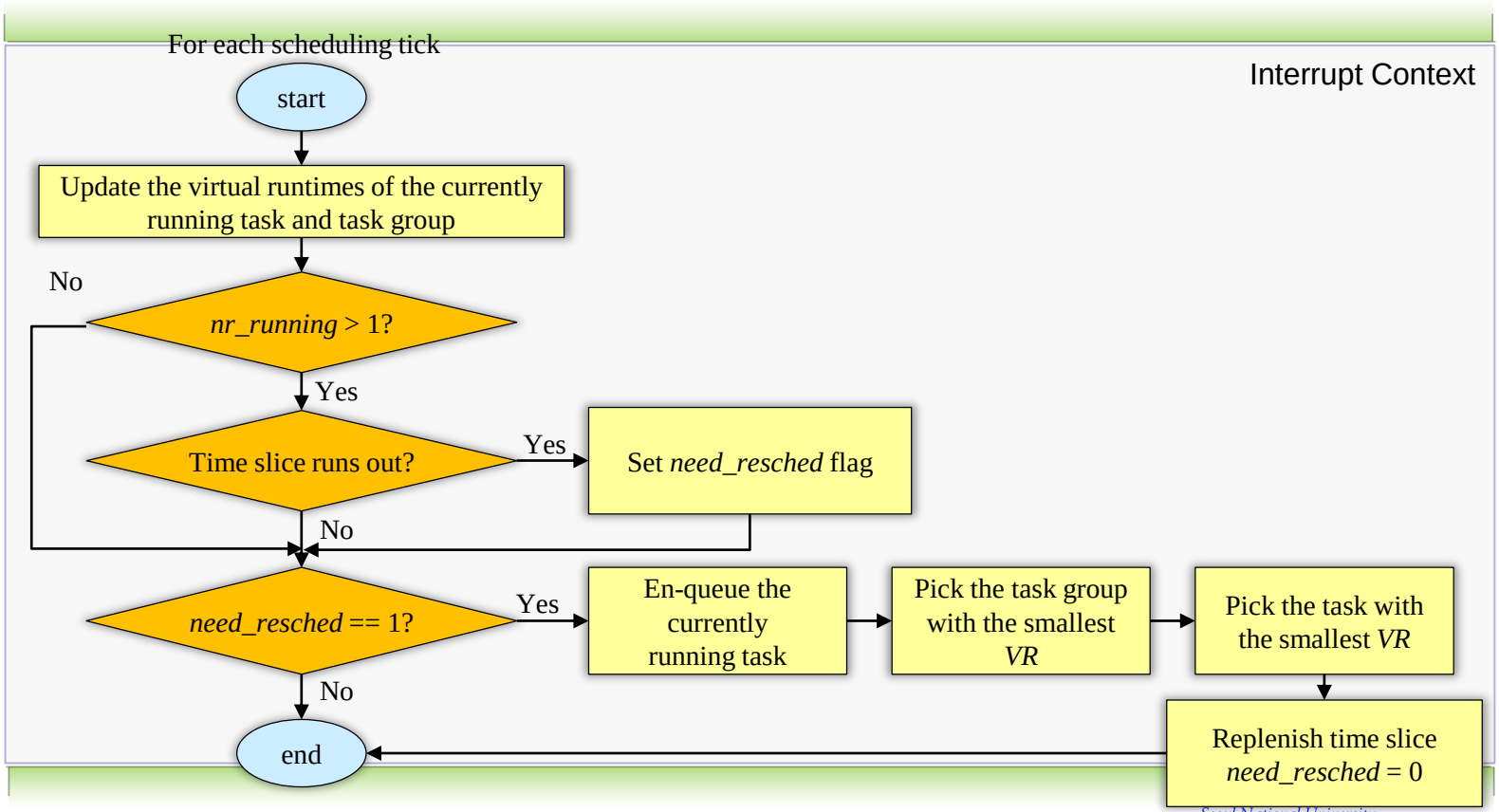
The number of tasks in the run-queue

$$P = \begin{cases} 6\ ms & \text{if } n > 8 \\ 0.75\ ms \times n & \text{otherwise} \end{cases}$$

    - For determining the preemption of the current running task, CFS uses the notion of the time slice

Linux CFS (4)

- Run-time algorithm of CFS



Linux CFS (5)

- As shown, CFS uses virtual runtimes merely for keeping the relative order among tasks
  - CFS adjusts a task's virtual runtime when it is inserted into or removed from the run-queue
    - ① De-queuing τ_i at time t_1

$$V'_{i(t_1)} = V_{i(t_1)} - V_{min(t_1)}$$

Minimum virtual runtime of the run-queue at time t_1

      - At the time of scheduling, $V'(t_1)$ equals to 0 since $V_i(t_1) = V_{min}(t_1)$
    - ② En-queuing τ_i at time t_2

$$V_{i(t_2)} = V'_{i(t_2)} + V_{min(t_2)}$$

Minimum virtual runtime of the run-queue at time t_2

Agenda

I. Introduction

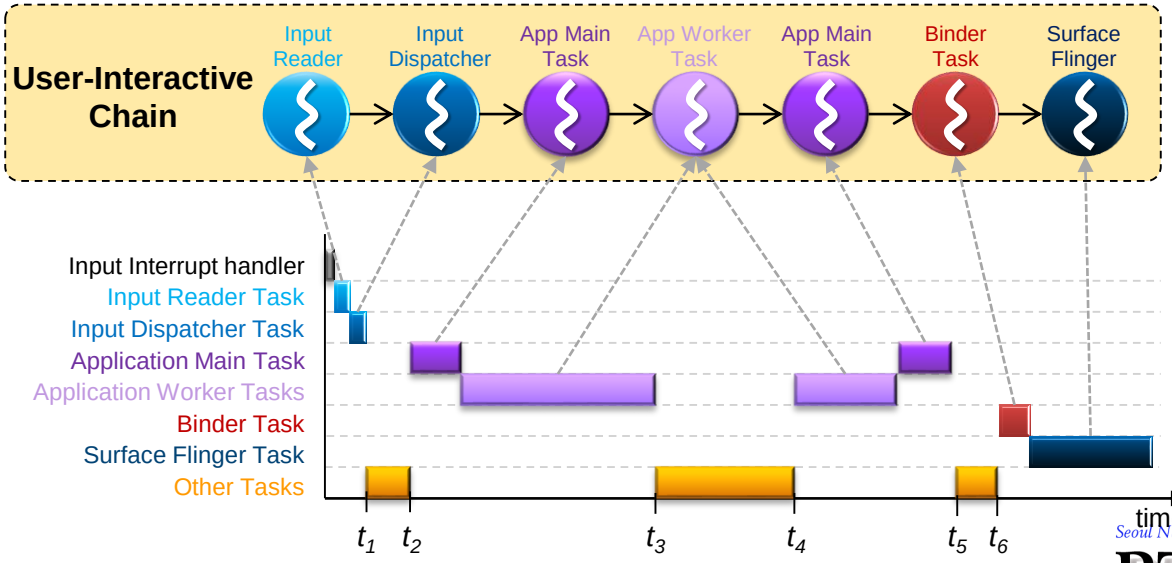
II. Case study: Cross-layer Resource Control for Smartphone

- A. Introduction
- B. Background: Android Framework and Linux Kernel
- **C. Problem Description**
- D. Solution Approach: FTC and VT-CFS
- E. Experimental Evaluation

III. Conclusion

Terminologies (1)

- DEFINITION 1. (**USER-INTERACTIVE TASK CHAIN**)
  - *A sequence of task executions which begins with a task handling an input event and ends with a task rendering the outcome of that input*
- DEFINITION 2. (**USER-INTERACTIVE TASK**)
  - *A task which appears in a user-interactive task chain*



Terminologies (2)

- DEFINITION 3. (**FOREGROUND TASK GROUP**)
  - *A set of tasks consisting of the main task of foreground application, system server tasks and kernel tasks*
- DEFINITION 4. (**BACKGROUND TASK GROUP**)
  - *A set of tasks consisting of the worker tasks of foreground application and all the tasks of background applications*

Problem Statement

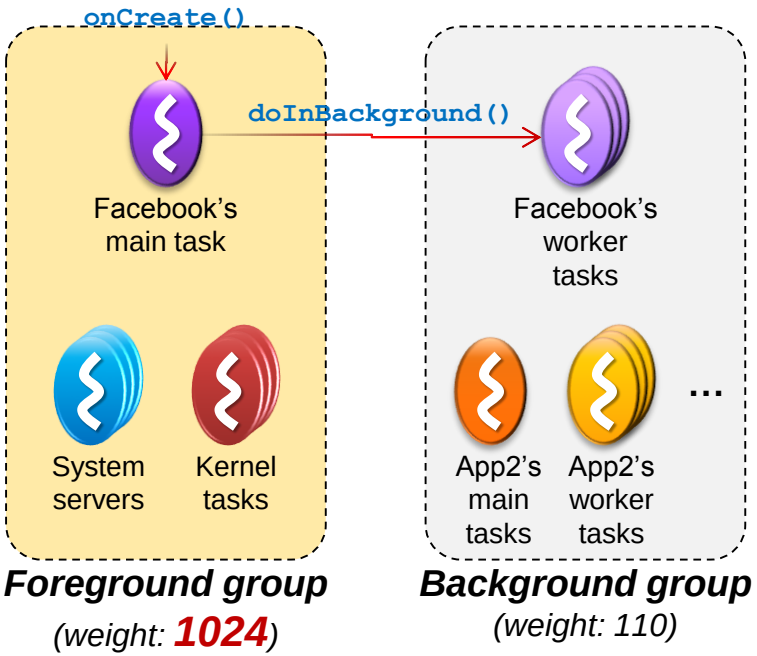
- Interactivity can be evaluated by the *end-to-end response time* taken to react to a user's action
  - In order to reduce the response time, it is critical for user-interactive tasks to reduce two types of latencies
    1) ***Preemption latency***
       - The accumulated amount of time during which a user-interactive task is preempted by other tasks until the completion of its execution
    2) ***Dispatch latency***
       - The delay between the time when a user-interactive task is inserted into a run-queue and the time when it begins to execute its first instruction

Android's Efforts (1)

- To shorten the response time, Android ensures
  - Main task of the application visible on screen can get a sufficient CPU time regardless of background loads
  - To do so, Android controls the amount of time during which tasks in BG interrupt tasks in FG
    - FG is assigned about 10 times larger weight for CPU than BG
      - Weight of FG: 1024, weight of BG: 110
    - From the perspective of task scheduling, CFS allocates CPU times to task groups proportionally to their weights
      - Tasks in FG are guaranteed to use at least 90% of CPU resource

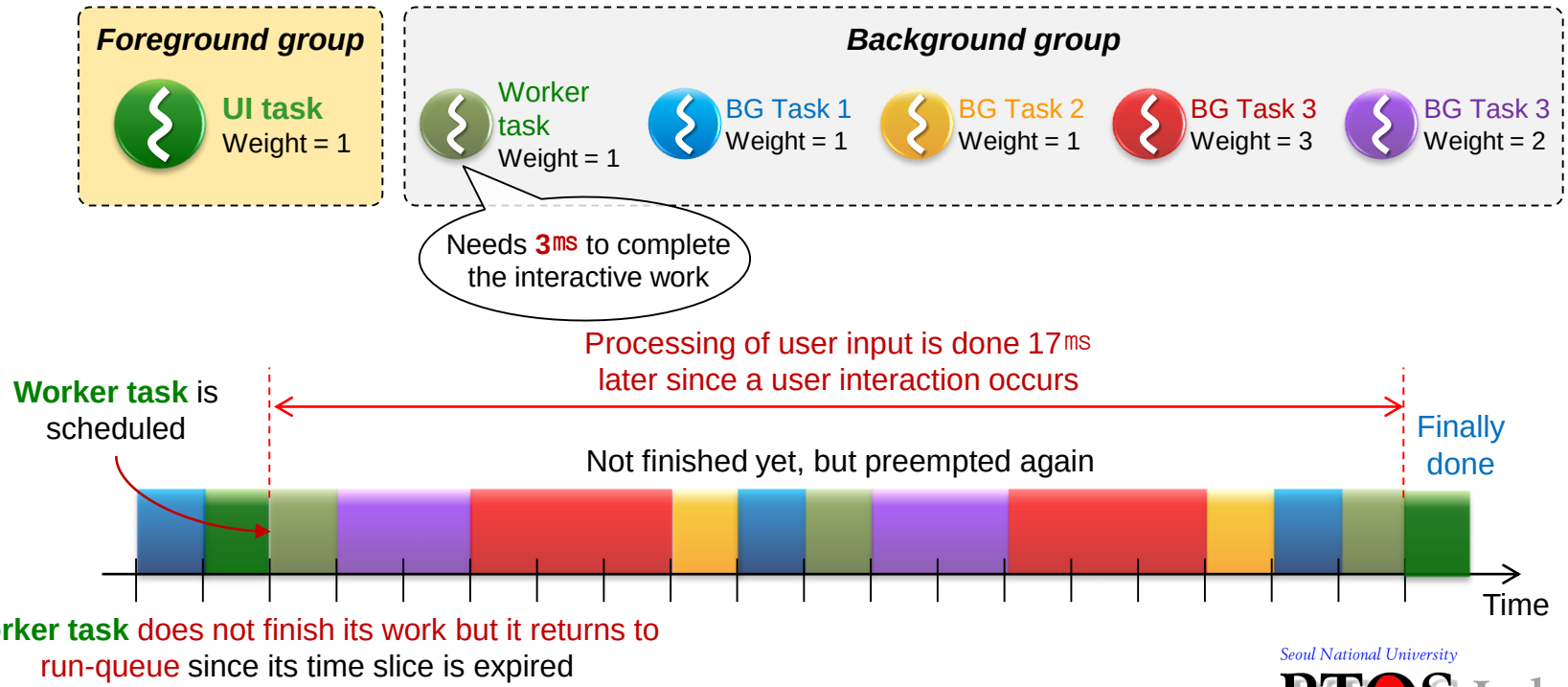
Android's Efforts (2)

- Execution of the main task of the application interacting with a user can be prioritized
  - Example: launching and interacting with Facebook App



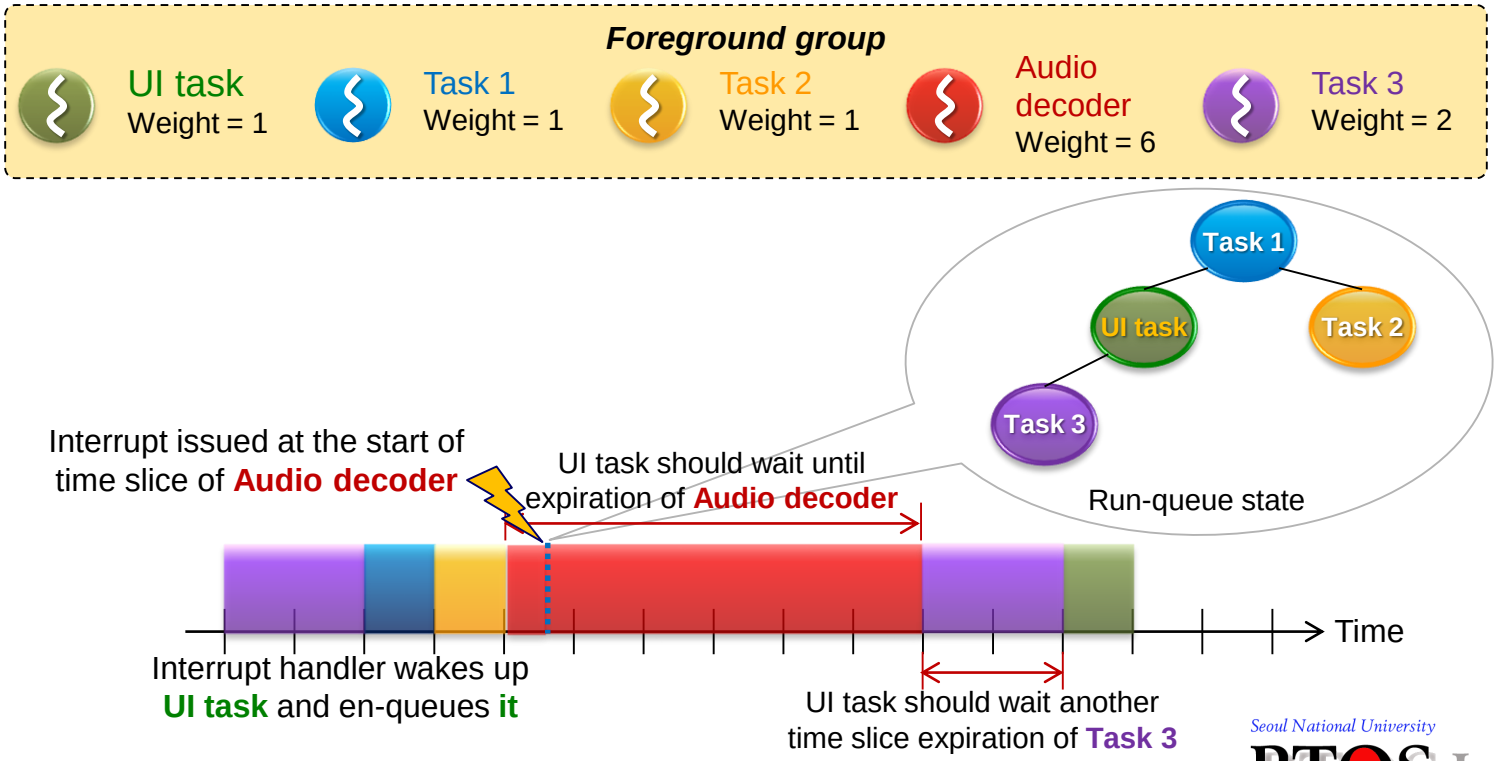
Why Android Fails to Achieve the Goal? (1)

- Long preemption latency of worker tasks
  - CFS cannot favor those tasks over other background tasks during task scheduling



Why Android Fails to Achieve the Goal? (2)

- Long dispatch latency
  - CFS schedules runnable tasks in a non-preemptive manner for their time slices



Agenda

I. Introduction

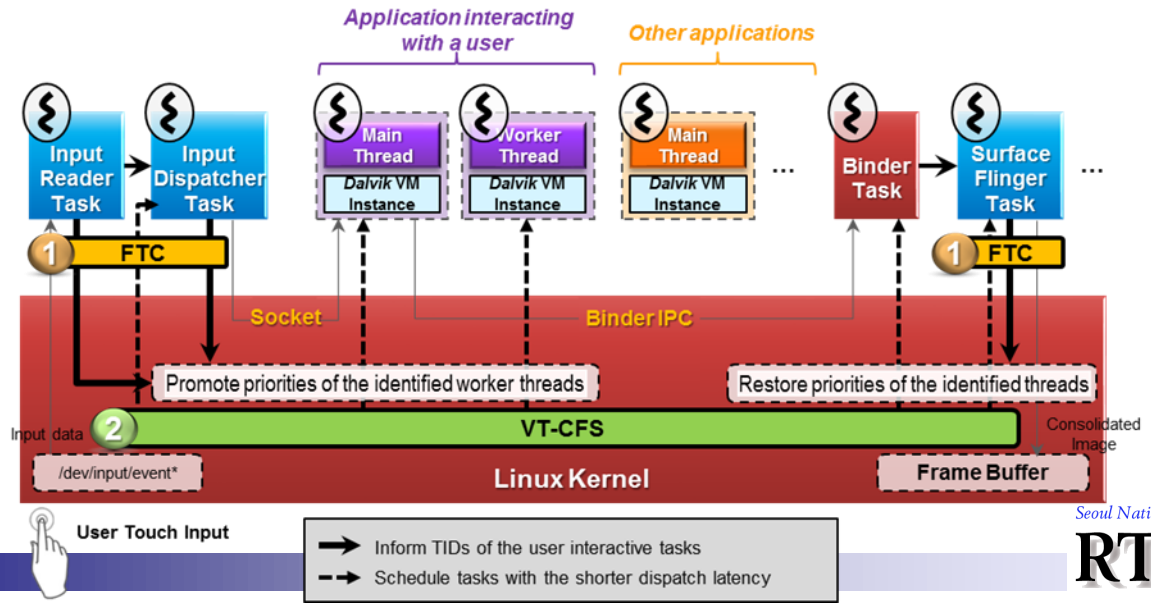
II. Case study: Cross-layer Resource Control for Smartphone

- A. Introduction
- B. Background: Android Framework and Linux Kernel
- C. Problem Description
- **D. Solution Approach: FTC and VT-CFS**
- E. Experimental Evaluation

III. Conclusion

Solution Overview

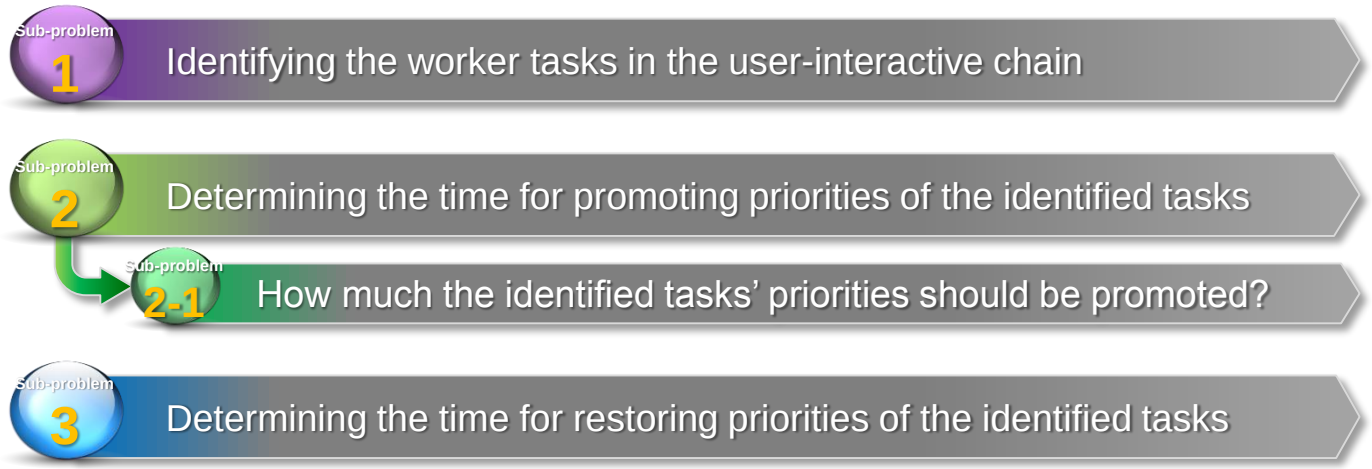
- Two interactivity enhancement mechanisms
  - **FTC**: to reduce preemption latency of worker tasks
    - Identify those tasks and promote their priorities
  - **VT-CFS**: to reduce dispatch latency of user-interactive tasks
    - Make the identified tasks be inserted at the first node in the run-queue and make other tasks more preemptive



Framework-assisted Task Characterization (FTC) (1)

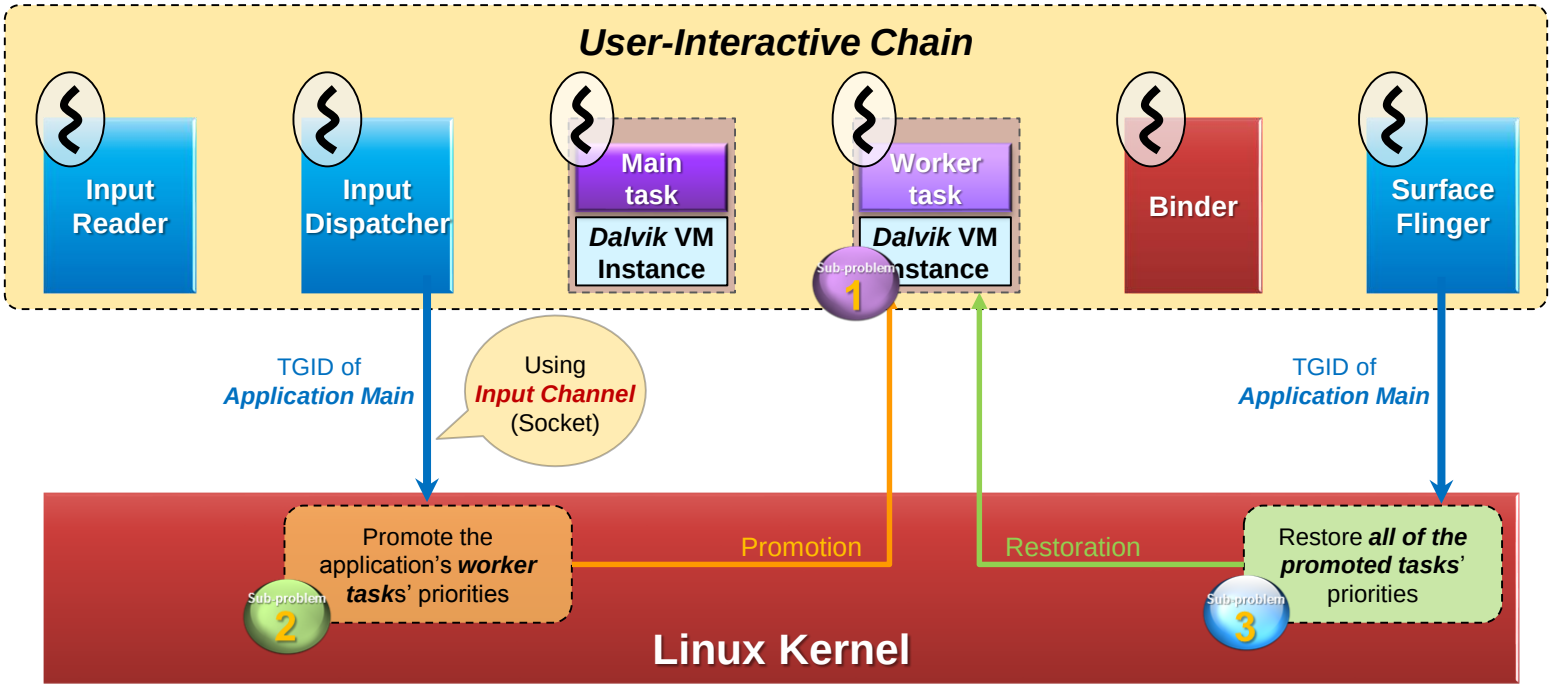
- Key idea of FTC
  - Selectively promote the priorities of user-interactive tasks running in the background group: ***worker tasks***
    - So that they can get larger time slices under CFS

- Sub-problems of FTC



Framework-assisted Task Characterization (FTC) (2)

- FTC takes advantage of run-time behaviors of the Android framework (For sub-problem 1, 2, 3)
  - Input event handling and rendering



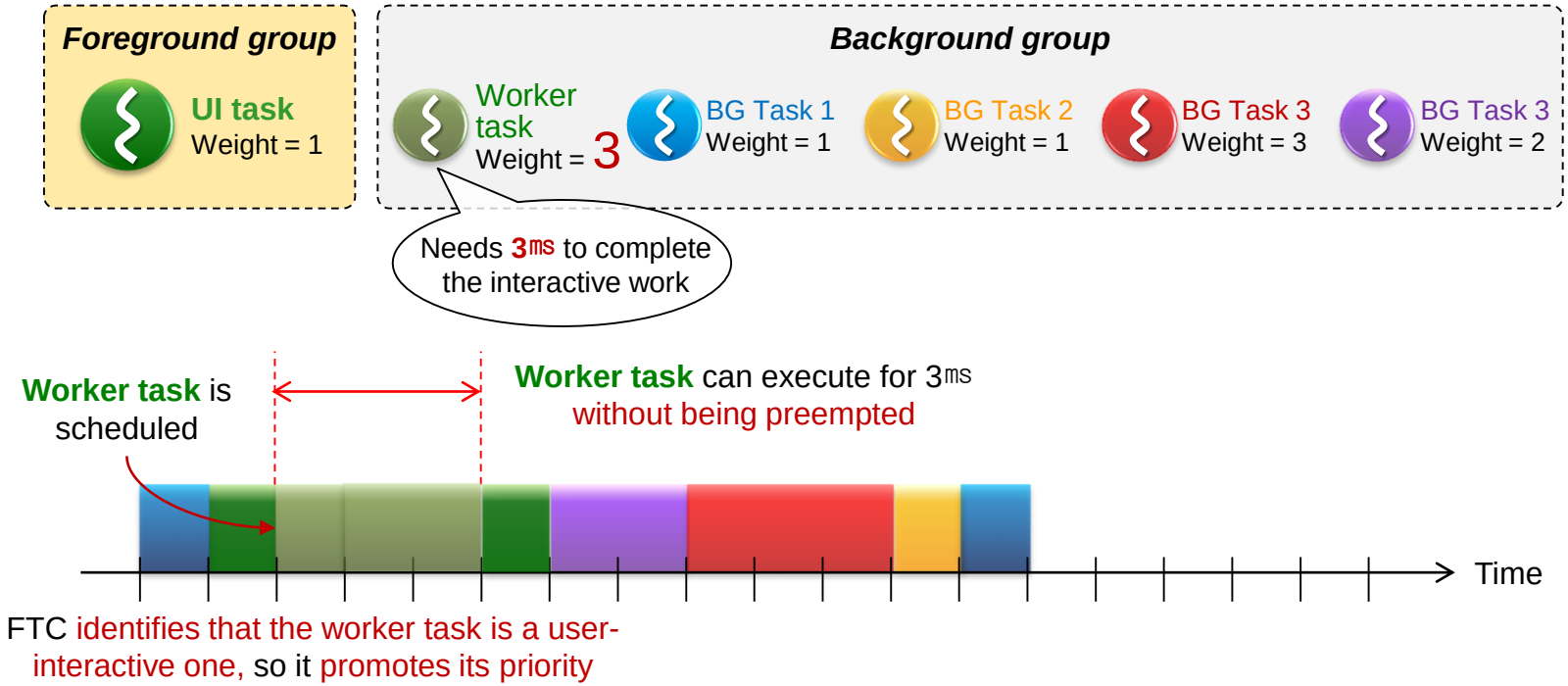
Framework-assisted Task Characterization (FTC) (3)

- How much user-interactive tasks' priorities should be promoted? (For sub-problem 2-1)
  - Until user-interactive tasks get time slice larger than the average execution time of them: `target_slice`
    - The length of `target_slice` varies depending on:
      1) The computational power of the underlying hardware
      2) User interaction patterns
      – On our target system, the best interactivity was achieved when `target_slice` was set to **10** and **20** milliseconds for instant and continuous interactions
  - New weight value for user interactive task τ_i

$$W'_i = \texttt{target\_slice} \times \frac{\sum_{\tau_j \in S} W_j}{P}$$

Framework-assisted Task Characterization (FTC) (4)

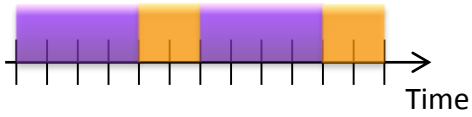
- Revisiting the problem
  - Preemption latency can be effectively reduced by FTC



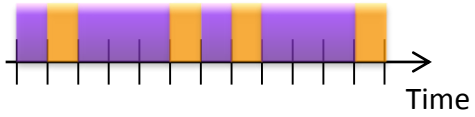
Virtual Time-based CFS (VT-CFS) (1)

- Key ideas of VT-CFS
  1. Force a task to be preempted at any predefined time tick
     - Tasks in VT-CFS become more pre-emptible than in CFS

CFS: Preempt task when running task runs out its **time slice**



VT-CFS: Preempt task for every **preemption tick period**



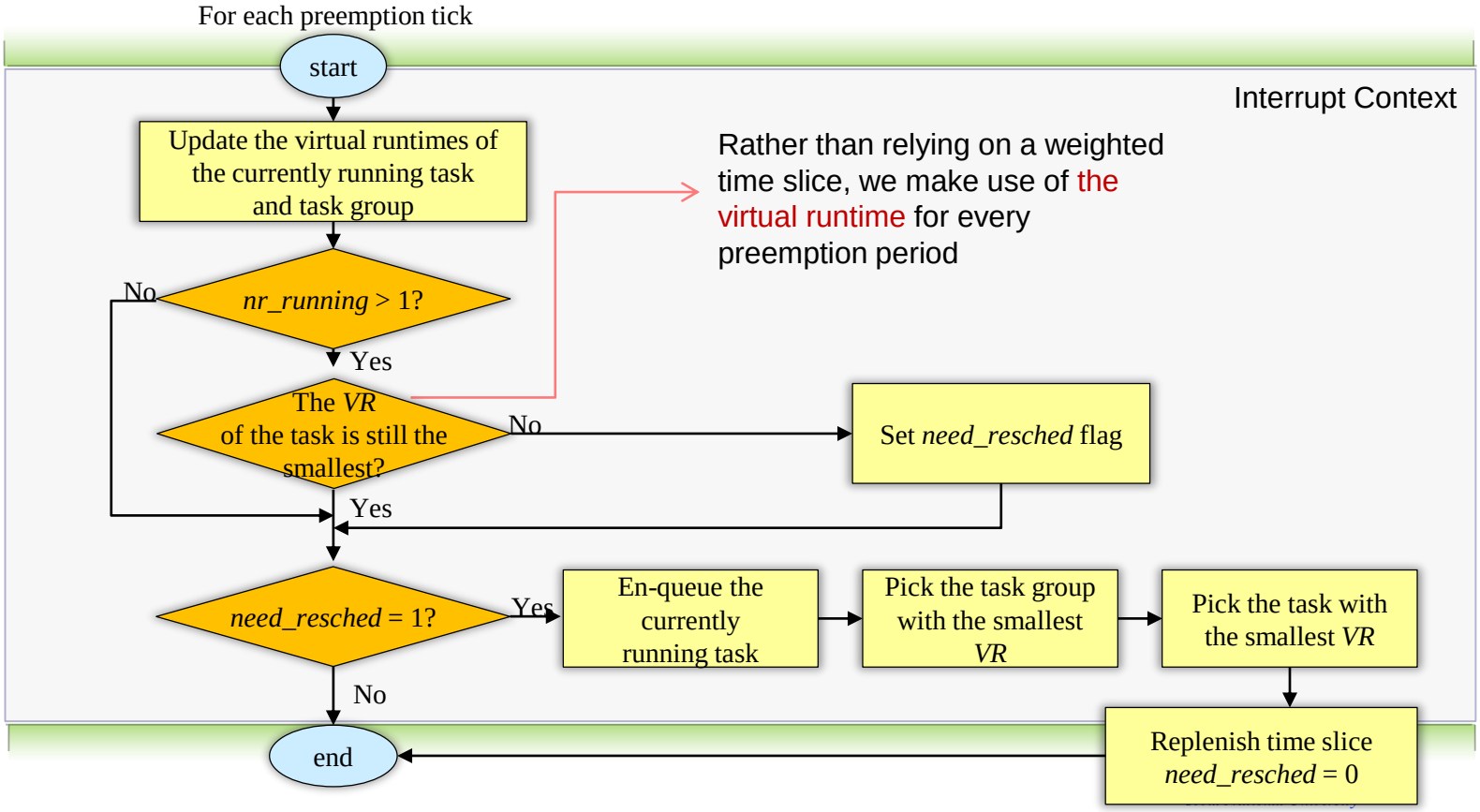
  2. User-interactive tasks identified by the FTC are treated differently from other preempted tasks
     - The identified tasks are always placed at the first node in the run-queue

Virtual Time-based CFS (VT-CFS) (2)

- Data structure
  - Maintain the identical data structure as the CFS
    - Use a ***red-black tree*** as a run-queue
    - Maintain a task's ***virtual runtime*** to provide fair scheduling
  - Eliminate the notion of a weighted time slice
  - Newly introduce ***the preemption tick period***
    - A constant regardless of given workload
    - A tunable parameter capable of controlling a tradeoff between *interactivity* and *run-time overhead*
      - A smaller period leads to shorter dispatch latency while incurring a larger overhead due to frequent context switches

Virtual Time-based CFS (VT-CFS) (3)

- More *preemptive* run-time scheduling algorithm



Virtual Time-based CFS (VT-CFS) (4)

- Virtual runtime adjustment
  - Apply for user-interactive tasks identified by FTC
    - When the user-interactive tasks are woken-up
  - New virtual runtime value for user interactive task τ_i

$$V'_i(t) = V^*_{curr}(t) + \frac{\omega_0}{\omega_{-20} + 1} \times \lambda$$

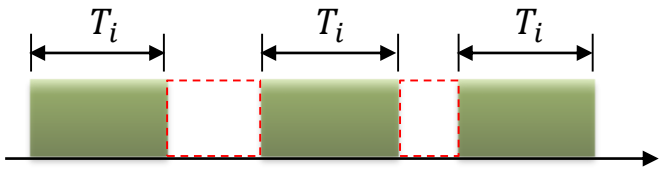
λ → Preemption tick period

$V^*_{curr}(t)$ → Stored virtual runtime of the currently running task

    - In VT-CFS, a virtual runtime difference of tasks is no lower than the virtual runtime increment of a task with nice value –20
      - $V_i'(t)$ guarantees that τ_i is placed at the first node and will be scheduled at the next preemption tick

Virtual Time-based CFS (VT-CFS) (5)

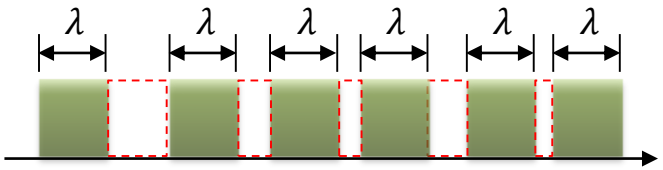
- Run-time overhead of VT-CFS
  - It incurs timer interrupts more frequently than CFS
    - Requires additional preemption tick interrupts
  - It may perform more frequent context switches than CFS



Maximum # of context switches per one task i:
$\lceil E_i / \max(\text{scheduling tick period}, T_i) \rceil$

Execution time of task i — Time slice of task i

T_i can vary from 0.001 to 4438.05 depending on the number of tasks and their weight distribution



Maximum # of context switches per one task i:
$\lceil E_i / \lambda \rceil$

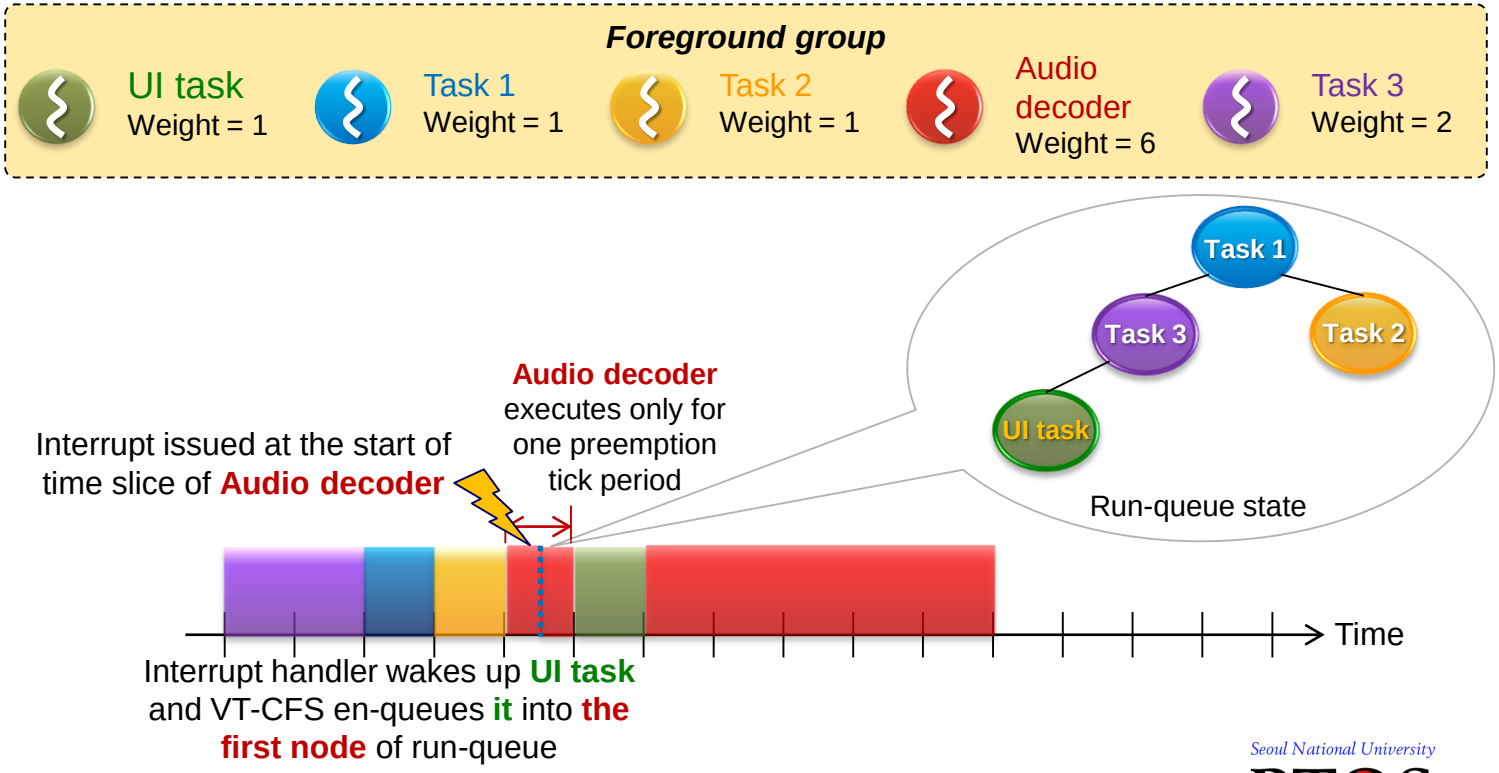
Preemption tick period

λ is a constant regardless of the number of tasks and their weight distribution

✓ Trade-off: *interactivity* vs. *run-time overhead*
Smaller λ (preemption tick period) leads to shorter dispatch latency while incurring larger overhead

Virtual Time-based CFS (VT-CFS) (6)

- Revisiting the problem
  - Dispatch latency can be effectively reduced by VT-CFS



Agenda

I. Introduction

II. Case study: Cross-layer Resource Control for Smartphone

- A. Introduction
- B. Background: Android Framework and Linux Kernel
- C. Problem Description
- D. Solution Approach: FTC and VT-CFS
- **E. Experimental Evaluation**

III. Conclusion

Experimental Setup

- Hardware/software components of the target system

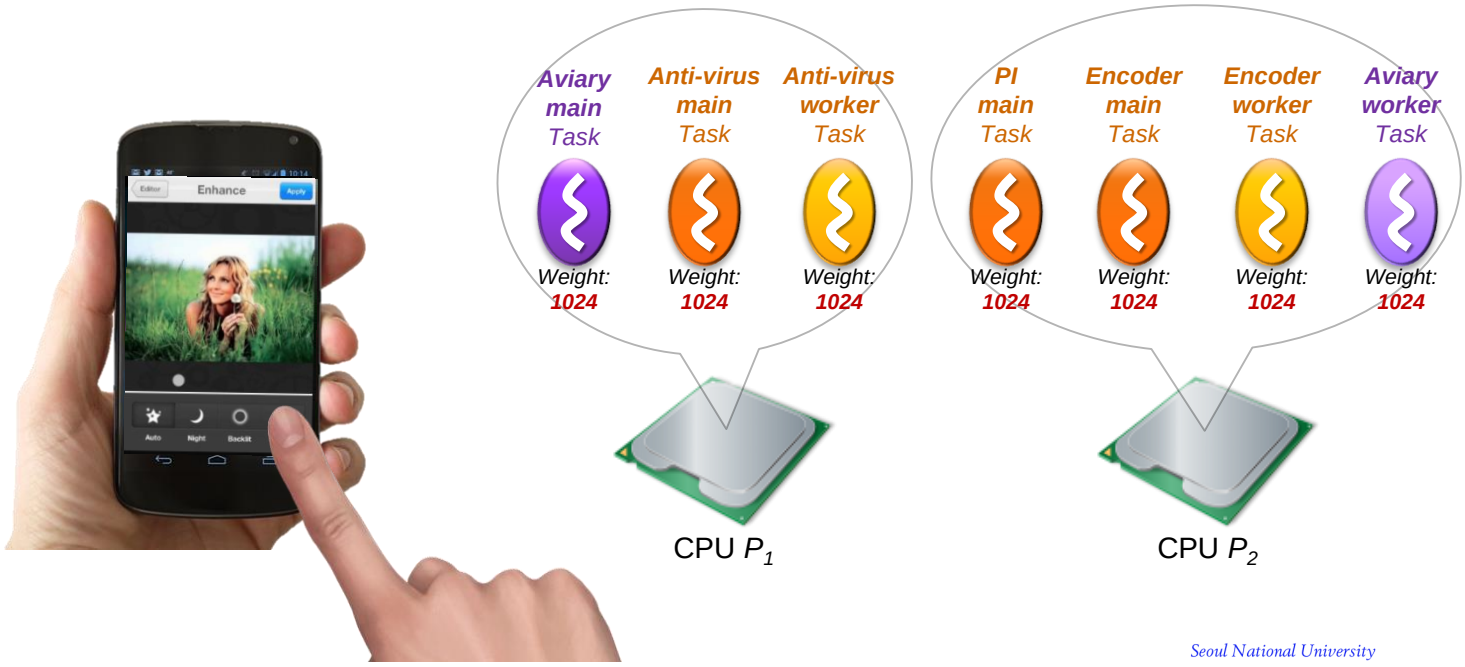
Hardware	System on Chip	Texas Instruments OMAP 4460
	CPU	1.2 GHz dual-core ARM Cortex-A9
	Main memory	1-GB LP-DDR2
	Storage	16 GB NAND Flash
	Display	4.65 in diagonal HD Super AMOLED
Software	Kernel	Linux kernel version 3.0.31
	Android Framework	Android 4.1.2 Jelly Bean
	Build number	JZO54K (485486)
	ROM	yakju

Google's Galaxy Nexus GSM/HSPA+

- Scheduling event monitoring is done by tools such as:
  - `trace-cmd`, Dalvik Debug Monitor Server (DDMS), `KernelShark`, `Systrace`

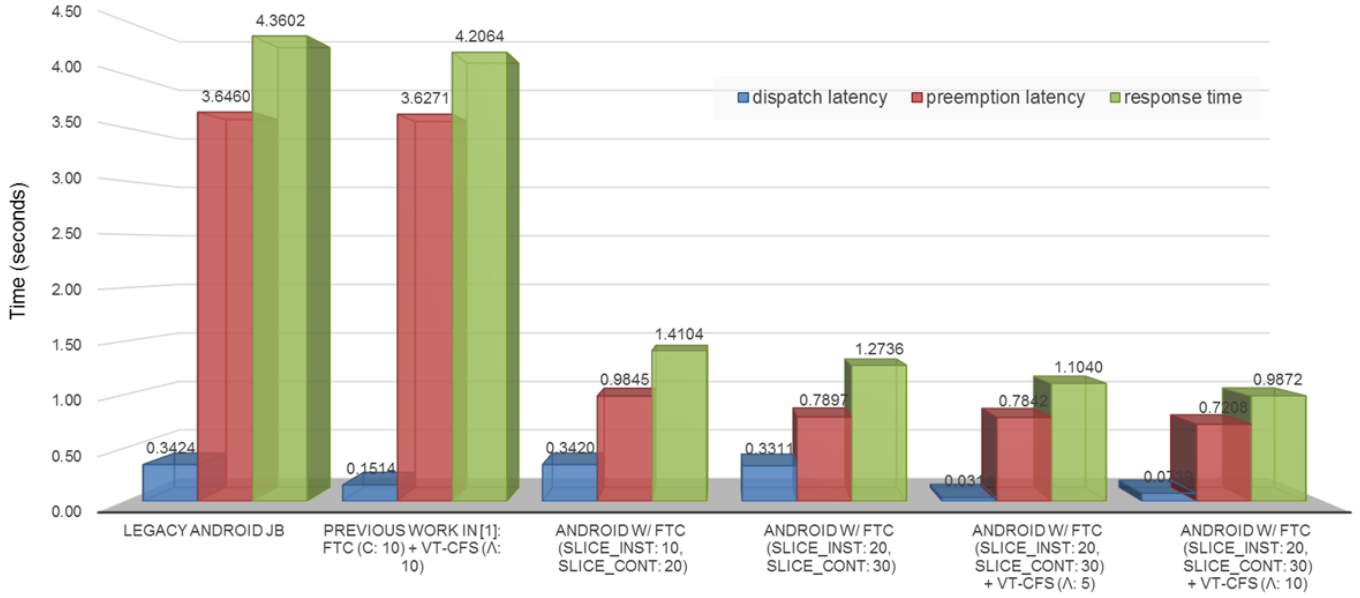
Experiments Scenario

- Workloads
  - FG App: *Aviary* photo editing application
  - BG Apps: *AVG* Antivirus, MPEG encoder, PI calculator



Evaluating Interactivity

- End-to-end response time of *Aviary* application
  - Reduced by up to **77.36%** compared to the legacy system
    - Preemption latency is reduced by 80.25%
    - Dispatch latency is reduced by 77.35%



Demonstration

Agenda

I. Introduction

II. Case study: Cross-layer Resource Control for Smartphone

III. Conclusion

Summary (1)

- Software design continues to become more complex in today's computing systems
  - Increasingly difficult to ensure the desired level of performance in such systems
- **Cross-layer optimization** is an effective way to improve performance in a complex layered SW platform
  - By escaping from the layered software structure with virtually strict boundaries between layers
    - Allow communication between layers by permitting one layer to access the data of another layer to exchange information

Summary (2)

- Cross-layer resource control and scheduling for improving interactivity in Android
  - Enhancing a user interactivity via task scheduling
    - Framework-assisted task characterization (***FTC***)
      - Selectively promote the priorities of user-interactive tasks running in the background group
    - Virtual time-based CFS (***VT-CFS***)
      - Force a task to be preempted at any predefined time tick
      - User-interactive tasks identified by the FTC are treated differently from other preempted tasks

Questions or Comments?