<Slides download>
https://www.pf.is.s.u-tokyo.ac.jp/classes

Advanced Operating Systems

#6

Hiroyuki Chishiro
Project Lecturer

Department of Computer Science
Graduate School of Information Science and Technology
The University of Tokyo

Course Plan

- Multi-core Resource Management
- Many-core Resource Management
- GPU Resource Management
- Virtual Machines
- Distributed File Systems
- High-performance Networking
- Memory Management
- Network on a Chip
- Embedded Real-time OS
- Device Drivers
- Linux Kernel

Schedule

1. 2018.9.28 Introduction + Linux Kernel (Kato)
2. 2018.10.5 Linux Kernel (Chishiro)
3. 2018.10.12 Linux Kernel (Kato)
4. 2018.10.19 Linux Kernel (Kato)
5. 2018.10.26 Linux Kernel (Kato)
6. 2018.11.2 Advanced Research (Chishiro)
7. 2018.11.9 Advanced Research (Chishiro)
8. 2018.11.16 (No Class)
9. 2018.11.23 (Holiday)
10. 2018.11.30 Advanced Research (Kato)
11. 2018.12.7 Advanced Research (Kato)
12. 2018.12.14 Advanced Research (Chishiro)
13. 2018.12.21 Linux Kernel
14. 2019.1.11 Linux Kernel
15. 2019.1.18 (No Class)
16. 2019.1.25 (No Class)

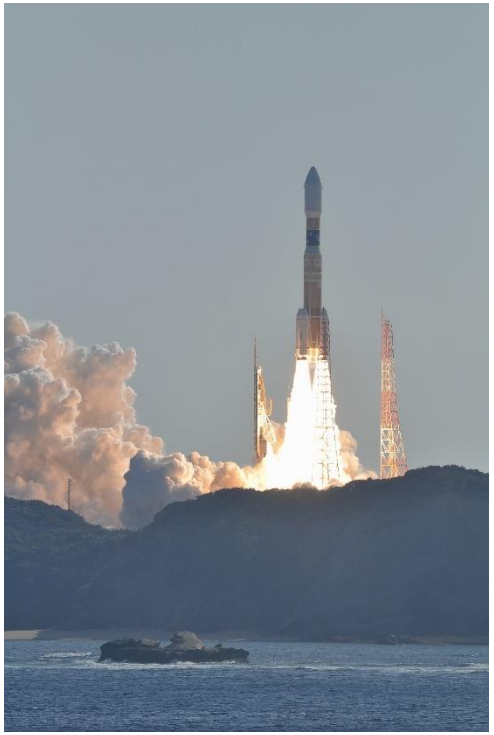
Outline

- Embedded Real-Time OS
- Dedicated OS
- Real-Time Extension of General Purpose OS (Linux)
- Mainline Implementation of Real-Time Tasks in Linux
- Introduction of My Research

Outline

- Embedded Real-Time OS
- Dedicated OS
- Real-Time Extension of General Purpose OS (Linux)
- Mainline Implementation of Real-Time Tasks in Linux
- Introduction of My Research

Applications of Embedded Real-Time Systems



H-IIA/B by JAXA [H-IIB]

Mars Exploration Rover by NASA [MER]

Kojiro by The University of Tokyo [Kojiro]

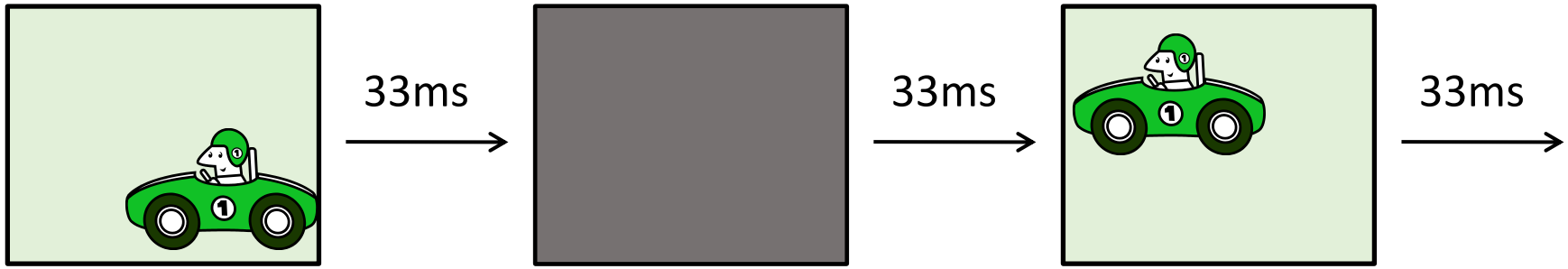
[H-IIB] https://upload.wikimedia.org/wikipedia/commons/e/e9/H-IIB_F2_launching_HTV2.jpg
[MER] https://en.wikipedia.org/wiki/Mars_Exploration_Rover#/media/File:NASA_Mars_Rover.jpg
[Kojiro] http://www.jsk.t.u-tokyo.ac.jp/

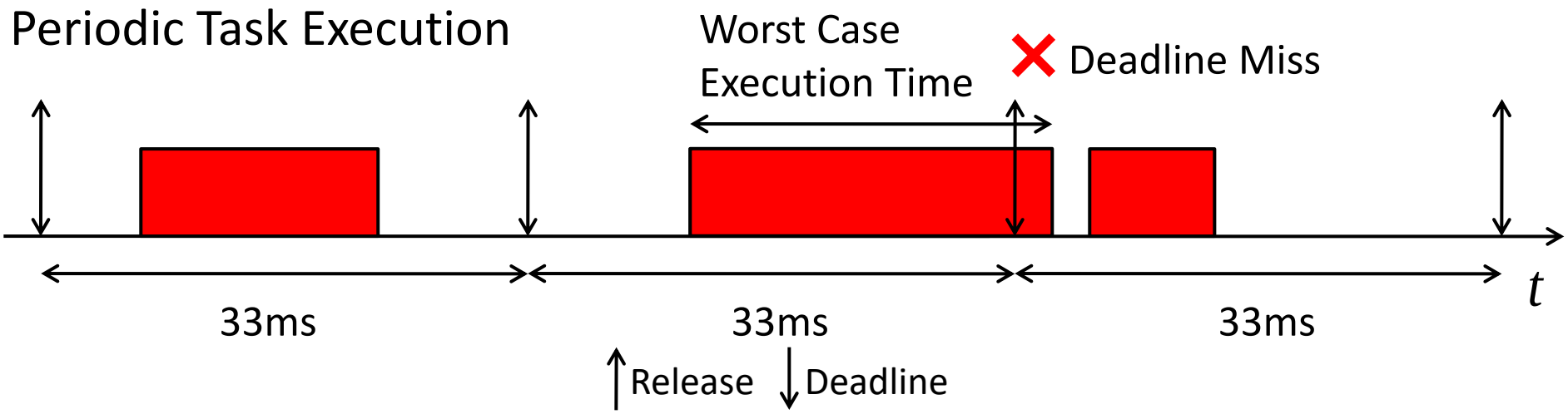
Why Embedded Real-Time OS?

- Embedded Real-Time Systems require:
  - Resource constraints
    - Timing Constraints (Main Topic)
    - Energy, Quality, Memory, Jitter, … Constraints
  - High Quality Software
- Embedded Real-Time OS:
  - is developed for guaranteeing timing constraints.
    - to complete tasks by their worst case execution time
    - to reduce overhead
  - implements real-time scheduling.
    - to complete tasks by their deadlines

Multimedia Applications

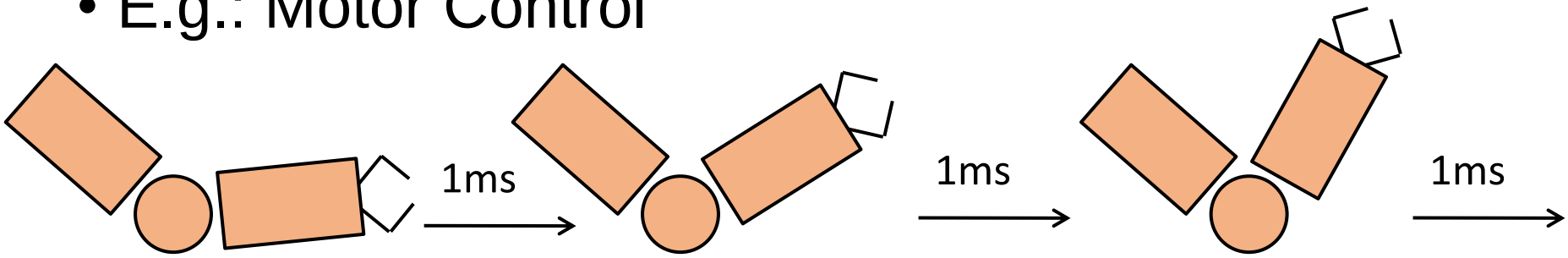
- E.g.: Dynamic Image Processing (30FPS)



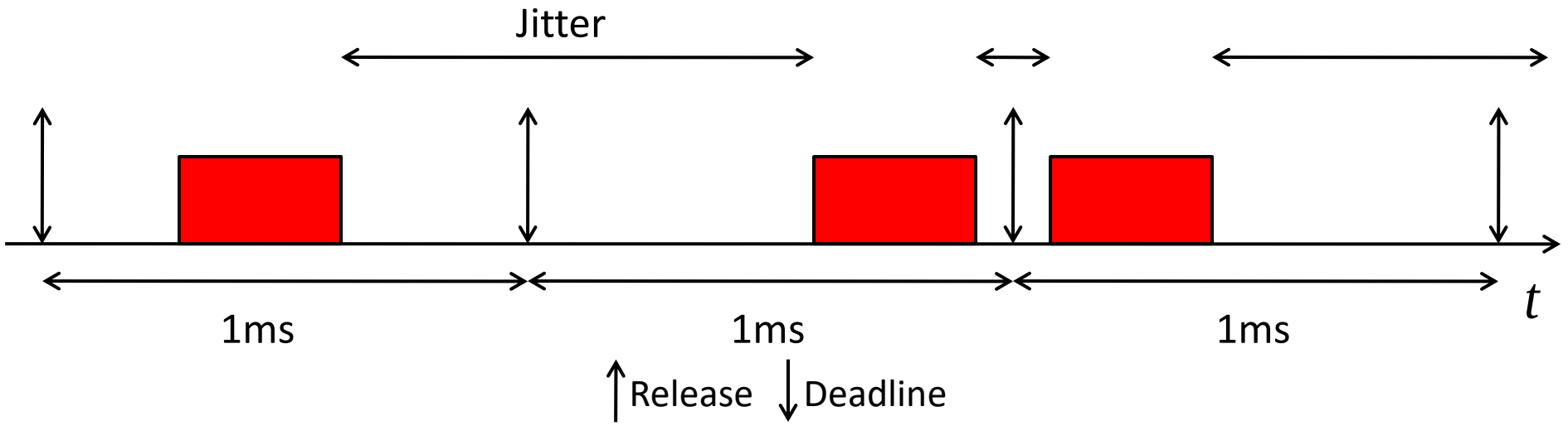


Robot Applications

- E.g.: Motor Control

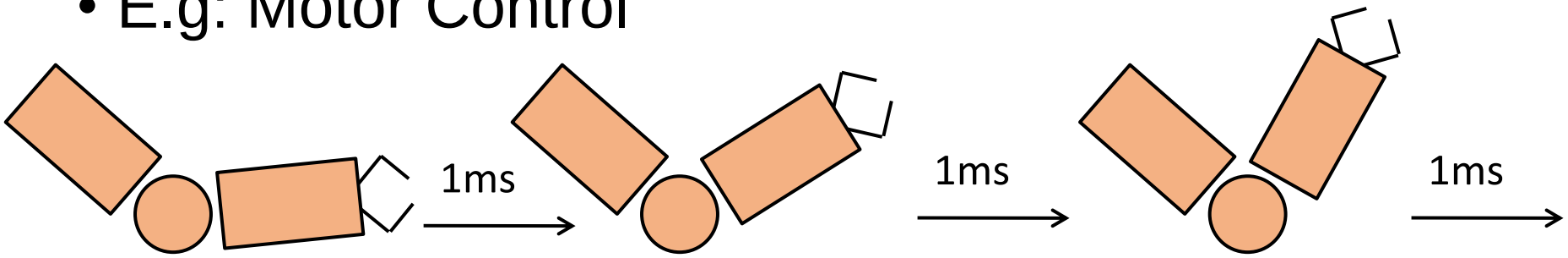


must not miss deadline (require hard real-time)

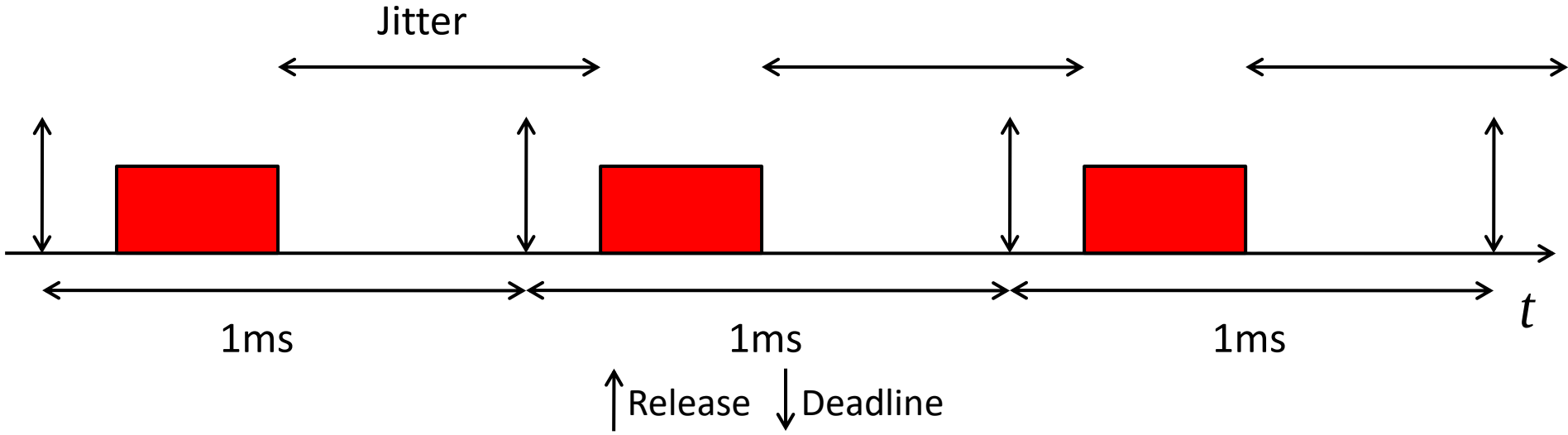


Robot Applications

- E.g: Motor Control

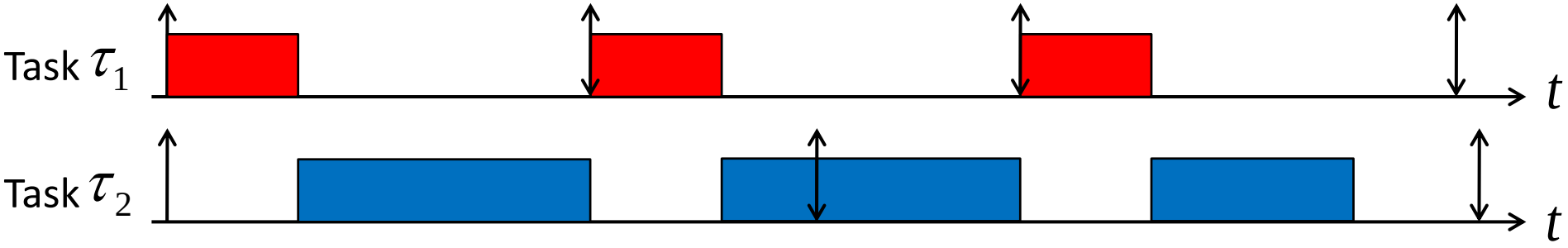


must not miss deadline (require hard real-time)

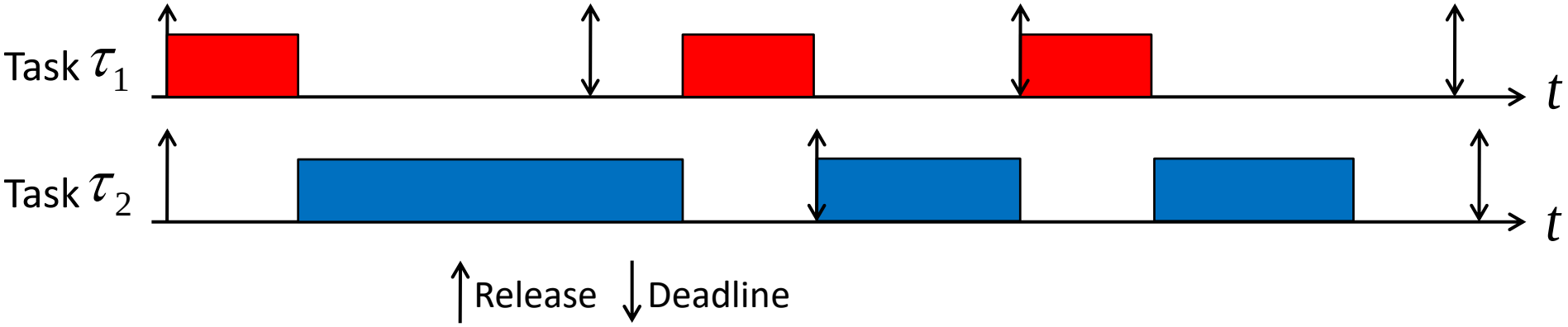


Rate Monotonic (RM) and Earliest Deadline First (EDF) Scheduling [Liu 73]

Rate Monotonic (RM): Shorter Period, Higher Priority (Fixed Priority Scheduling)



Earliest Deadline First (EDF): Earlier Deadline, Higher Priority (Dynamic Priority Scheduling)



[Liu 73] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. Journal of the ACM, Vol. 20, No. 1, pp. 46–61, January 1973.

Embedded Real-Time Systems vs General Purpose Systems

	Embedded Real-Time Systems	General Purpose Systems
Target Systems	Spacecraft, Robots, Automotive Cars, Mobile Phones, Game Machines	Personal Computers, Servers
Timing Constraints	Yes	No
Task Execution Policy	Deadline	Throughput
Jitter	Important	Not Important
Performance	Worst Case	Average Case

Categorization of Embedded Real-Time OS

- Dedicated OS
  - is developed from scratch.
  - has a medium amount of code (10,000 – 100,000 lines).
- Real-Time Extension of General Purpose OS (Linux)
  - is developed to extend Linux.
  - has a large amount of code (10,000,000 – 20,000,000 lines).
  - has a small amount of additional code (1,000 – 10,000 lines).

Dedicated OS vs Real-Time Extension of Linux

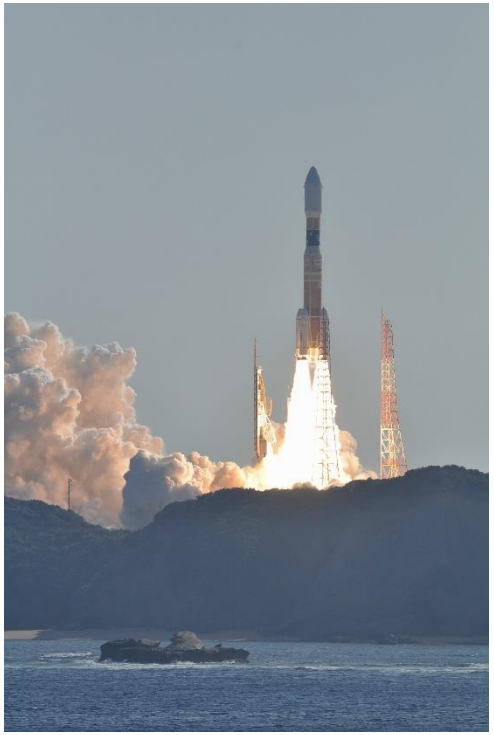
	Dedicated OS	Real-Time Extension of Linux
OS Development	High Cost (from Scratch)	Low Cost (Extension of Existing OS)
Application Development	Hard (from Scratch)	Easy (Making use of Existing software in Some Cases)
Overhead and Jitter	Low	High
Verification of Software	Low Cost (Small Code Size)	High Cost (Large Code Size)
Maintenance of OS	Easy (Mainline Development)	Hard (Depending on Bugs/Updates from Existing OS)

NOTE: including my opinion/experience

Outline

- Embedded Real-Time OS
- Dedicated OS
- Real-Time Extension of General Purpose OS (Linux)
- Mainline Implementation of Real-Time Tasks in Linux
- Introduction of My Research

Application Usage of Dedicated OS



H-IIA/B by JAXA [H-IIB]

TOPPERS/HRP2

Mars Exploration Rover by NASA [MER]

VxWorks

Kojiro by The University of Tokyo [Kojiro]

HOS

Categorization of Dedicated OS

- Specification of APIs
  - iTRON-specified OS
    - TOPPERS/HRP2 by Nagoya University [HRP2]
    - Hyper Operating Systems (HOS) by Project HOS [HOS]
  - AUTOSAR-specified OS
    - TOPPERS/ATK2 by Nagoya University [ATK2]
- Specially Designed OS
  - VxWorks by Wind River [VxWorks]

[HRP2] https://www.toppers.jp/hrp2-kernel.html
[HOS] https://ja.osdn.net/projects/hos/
[ATK2] https://www.toppers.jp/atk2.html
[VxWorks] https://www.windriver.com/products/vxworks/

Examples of µITRON 4.0 API: Subset of ITRON API

API	Behavior
cre_spn()	create lock object
loc_spn()	lock object
unl_spn()	unlock object
act_tsk()	activate task
pol_sem()	get semaphore by polling
wup_tsk()	wake up task
ter_tsk()	terminate task.
pol_sem()	lock semaphore by polling
sig_sem()	unlock semaphore.

Outline

- Embedded Real-Time OS
- Dedicated OS
- Real-Time Extension of General Purpose OS (Linux)
- Mainline Implementation of Real-Time Tasks in Linux
- Introduction of My Research

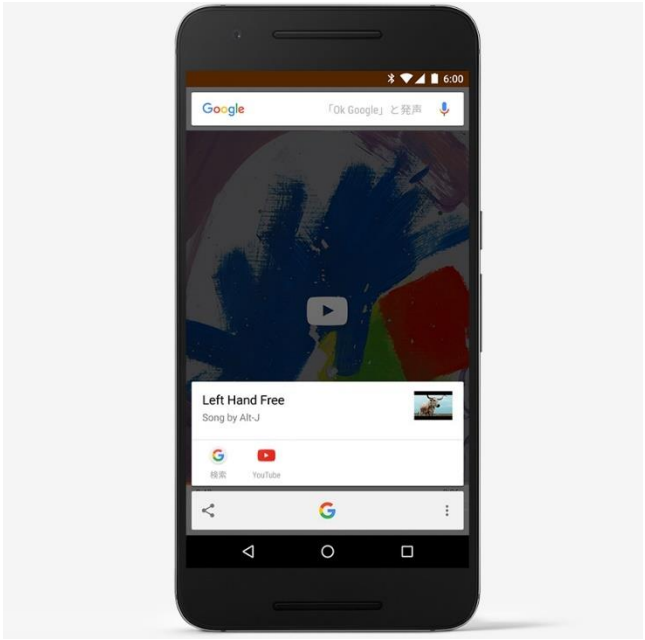
Why Real-Time Linux?

- Linux is open and free software.
- Linux's application development is easier than dedicated OS's one (in many cases).

- However:
  - Linux is developed for general purpose OS but not for real-time OS.
  - The design of general OS and real-time OS is different.

Usage of Real-Time Extension of Linux

HRP2 by AIST
ART-Linux



Android by Google
Android Linux

[HRP2] https://unit.aist.go.jp/is/humanoid/m_projects/hrp2_j.html
[Android] https://www.android.com/phones/

Real-Time Extension of Linux

- RTLinux [Yodaiken 99]
- ART-Linux [Ishiwata 98]
- LITMUS^RT [Calandrino 06]

[Yodaiken 99] Victor Yodaiken. The RTLinux Manifesto. The RTLinux Manifesto. In Proceedings of the 5th Linux Conference, pp. 1-12, March 1999.
[Ishiwata 98] Youichi Ishiwata and Toshihiro Matsui. A Real-Time Operating System that can Share Device Drivers with General Purpose OS. IEICE technical report, pages 41–48, September 1998.
[Calandrino 06] John M. Calandrino, Hennadiy Leontyev, Aaron Block, UmaMaheswari C. Devi, and James H. Anderson. LITMUS^{RT}: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers. Proceedings of the 27th IEEE Real-Time Systems Symposium, pp. 111–123, December 2006.

RTLinux

- executes real-time tasks as kernel modules in Linux.
  - NOTE: kernel modules run in kernel mode.
- executes Linux kernel as a non-real-time task only when there are no real-time tasks to run.
- implements APIs in Linux kernel:
  - rt_task_init(): creates real-time task
  - rt_task_make_periodic(): executes periodic task
  - rt_task_wait(): waits until next period
  - rt_task_delete(): deletes task

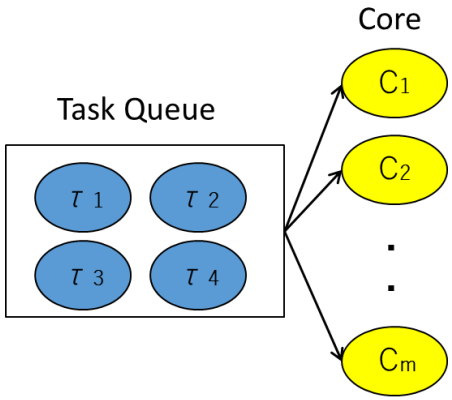
ART-Linux

- executes real-time tasks in user mode.
- implements new system calls.
  - art_enter(): creates real-time task (periodic or round robin).
  - art_wait(): waits until next period
  - art_exit(): deletes real-time task

LITMUS^RT

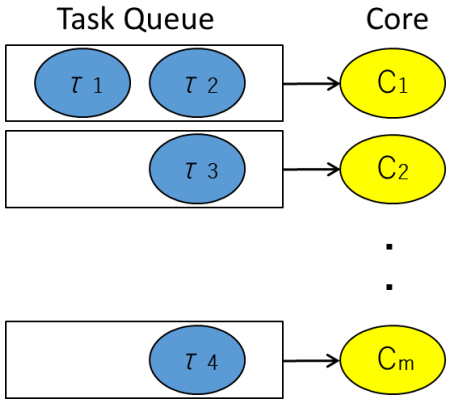
- supports global/partitioned/cluster real-time scheduling (e.g., RM and EDF) on multicore processors.

Global



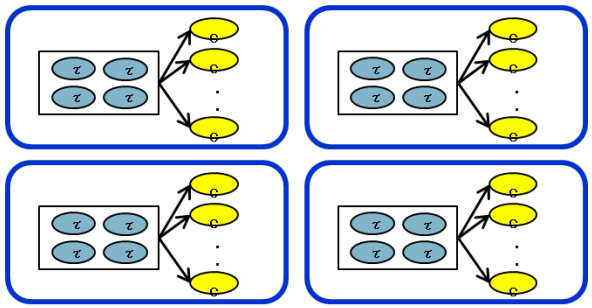
Assign tasks to cores online (at running)

Partitioned



Assign tasks to cores offline (before running)

Cluster



Assign tasks to clusters offline but to cores in each cluster online

Outline

- Embedded Real-Time OS
- Dedicated OS
- Real-Time Extension of General Purpose OS (Linux)
- Mainline Implementation of Real-Time Tasks in Linux
- Introduction of My Research

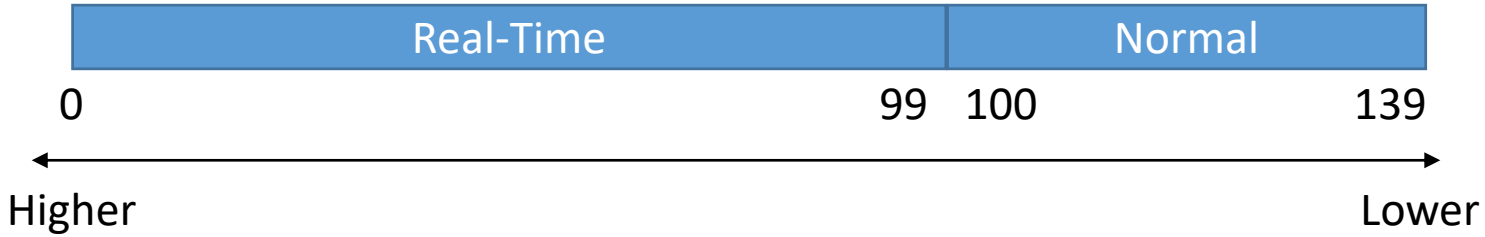
Mainline Implementation of Real-Time Tasks in Linux

- SCHED_RR [Sched]
  - can implement Round Robin Scheduling
  - has been mainlined since Linux kernel version 2.2.x (around 2000)
- SCHED_FIFO [Sched]
  - can implement RM Scheduling
  - has been mainlined since version 2.2.x (around 2000)
- SCHED_DEADLINE [Faggioli 09]
  - can implement EDF Scheduling
  - has been mainlined since version 3.14 (March 2014) [Linux3.14]

[Sched] http://www.linuxjournal.com/article/3910
Dario Faggioli, Fabio Checconi, Michael Trimarchi, Claudio Scordino. An EDF scheduling class for the Linux kernel. In Proceedings of the 11th Real-Time Linux Workshop, pp. 1-8, September 2009.
[Linux 3.14] https://kernelnewbies.org/Linux_3.14

Priority of Linux:

- Linux scheduler is a preemptive priority-based algorithm with two priority ranges:



  - Real-Time from 0 to 99 (100 levels)
    - Higher priority tasks are executed preferentially.
    - SCHED_FIFO/RR priority: 0 is min, 99 is max
    - “top” priority: -100 to -1 (NOTE: -100 is represented as “rt”)
  - Nice from 100 to 139 (40 levels)
    - Higher priority, longer time quantum (time slicing).
    - “nice” priority: mapped from -20 to 19
    - “top ” priority: mapped from 0 to 39

struct sched_param

- /usr/include/sched.h

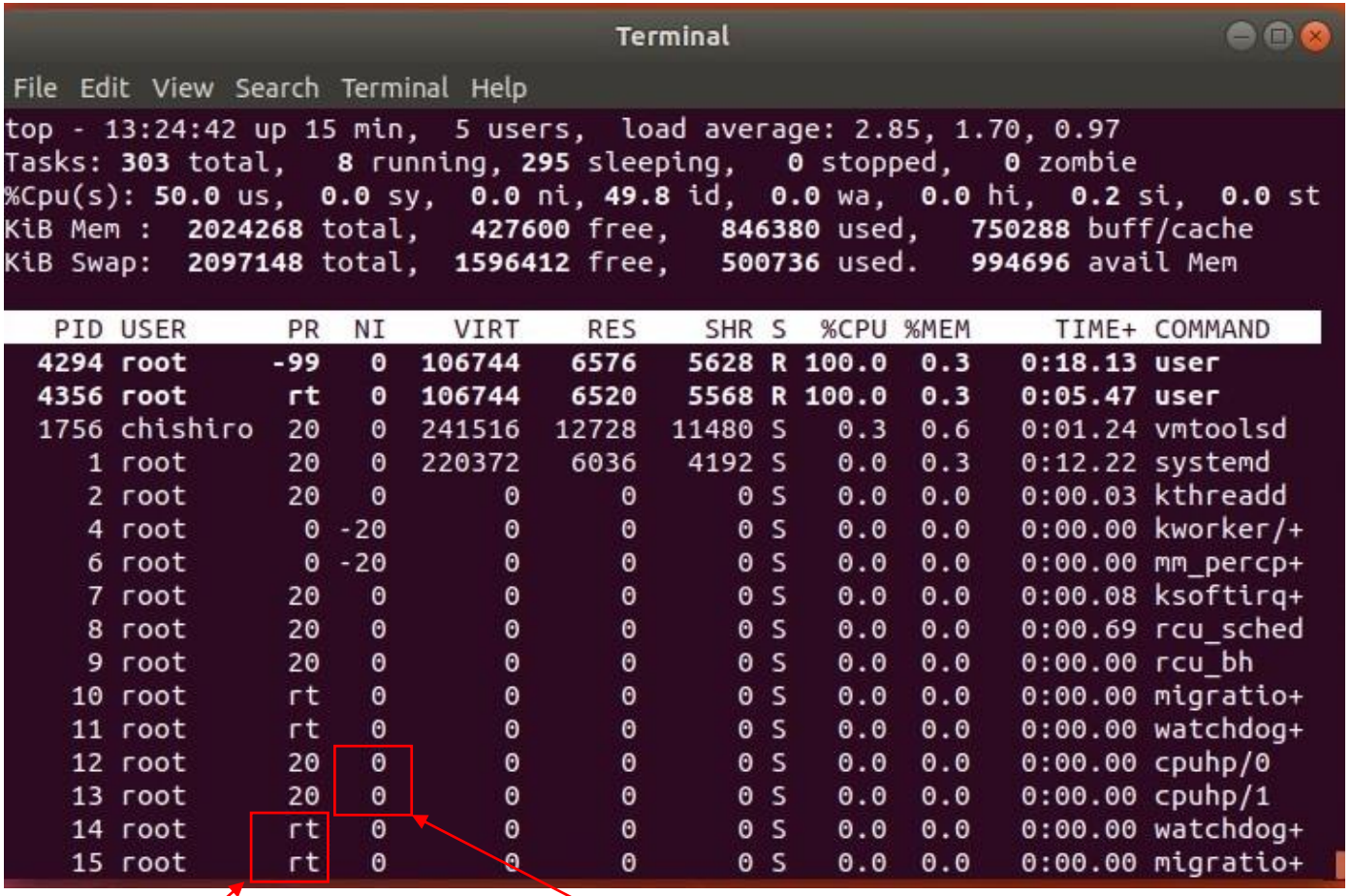
```
struct sched_param {
    int priority; /* priority for SCHED_FIFO/RR/OTHER/BATCH */
};
```

struct sched_attr since version 3.14

- /usr/include/sched.h

```
struct sched_attr {
        __u32 size; /* size of the structure, for forward/backward compatibility */

        __u32 sched_policy; /* Scheduling policy (SCHED_FIFO/RR/OTHER/BATCH ) */
        __u64 sched_flags; /* for customizing the scheduler behavior */

        /* SCHED_OTHER, SCHED_BATCH */
        __s32 sched_nice; /* Nice value (SCHED_OTHER/BATCH) */

        /* SCHED_FIFO, SCHED_RR */
        __u32 sched_priority; /* Static priority (SCHED_FIFO/RR)  */

        /* SCHED_DEADLINE */
        __u64 sched_runtime;  /* Relative deadline [ns] */
        __u64 sched_deadline; /* Runtime (Execution time) [ns] */
        __u64 sched_period; /* Period [ns] */
};
```

Screenshot of “top” command

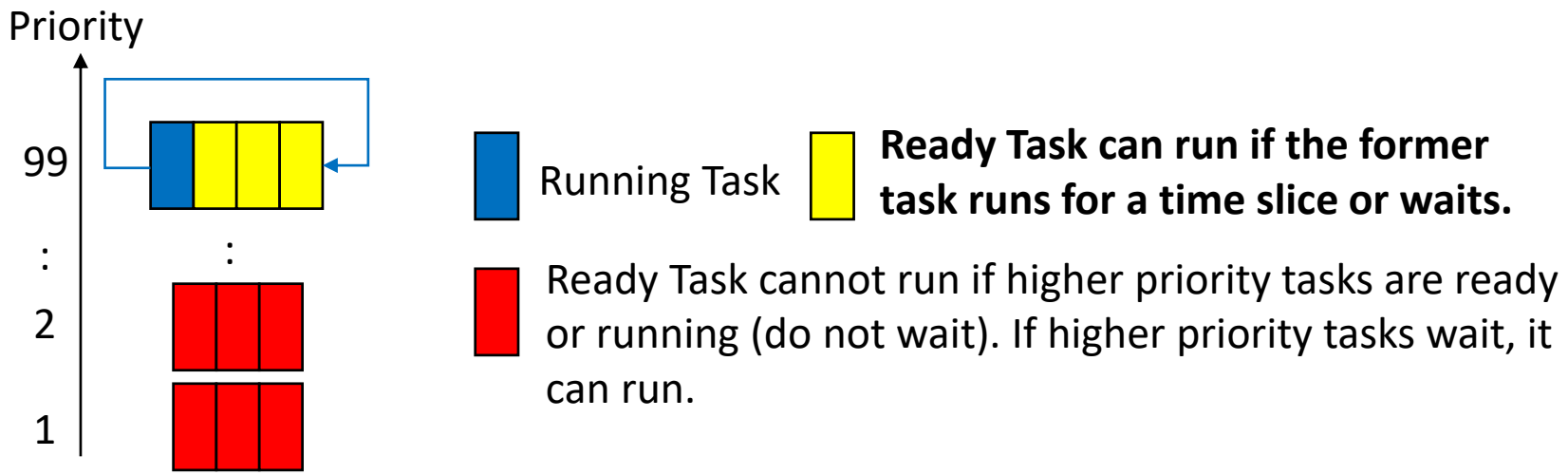


“rt” means the highest priority.

Default Priority General Process is Nice 0.

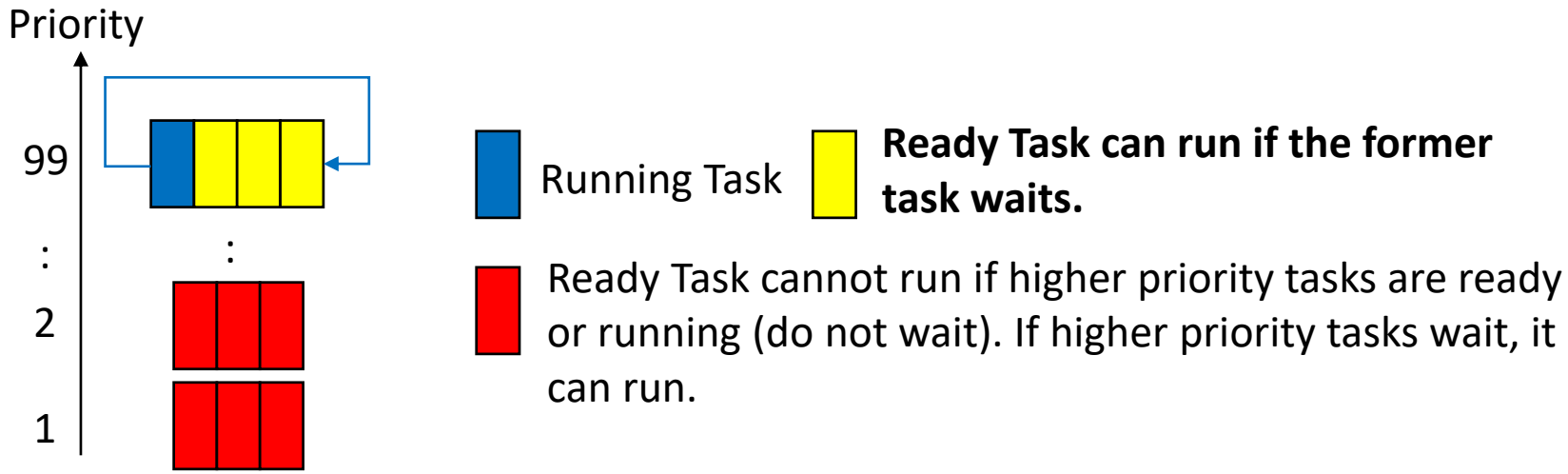
Round Robin Scheduling by SCHED_RR

- If a SCHED_RR thread runs for a time slice, it will be put at the end of the list for its priority.
  - kernel/sched/core.c
  - $ cat /proc/sys/kernel/sched_rr_timeslice_ms
    - 100



First in First Out Scheduling by SCHED_FIFO

- SCHED_FIFO can be used only with static priorities higher than 0.
  - kernel/sched/core.c



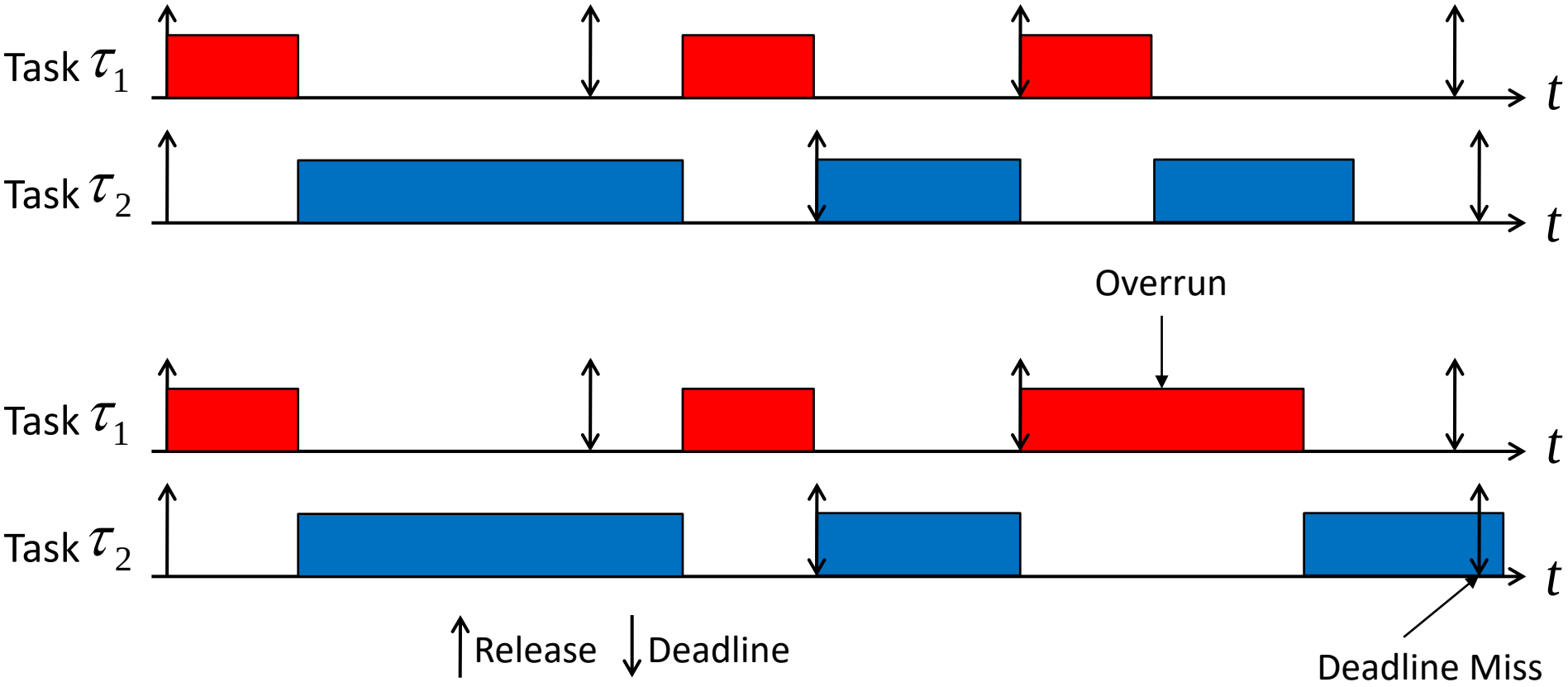
RM Scheduling by SCHED_FIFO

- can have 99 priority levels (different periods).
  - NOTE: In RM scheduling, shorter periods, higher priorities.
- clock_nanosleep()
  - sleeps until next period.
  - is used to implement RM scheduling.
- For example:
  - 1ms period task has 99 priority.
  - 5ms period task has 50 priority.
  - 100ms period task has 10 priority.

EDF Scheduling by SCHED_DEADLINE

- can have (nearly) unlimited priority levels.
  - NOTE: in EDF Scheduling, earlier deadlines, higher priorities.
  - kernel/sched/deadline.c
  - Depending on maximum number of threads
  - $ cat /proc/sys/kernel/threads-max
    - 514870 (in our server)
- SCHED_DEADLINE tasks have higher priority than SCHED_FIFO/RR ones.

Problem of EDF Scheduling



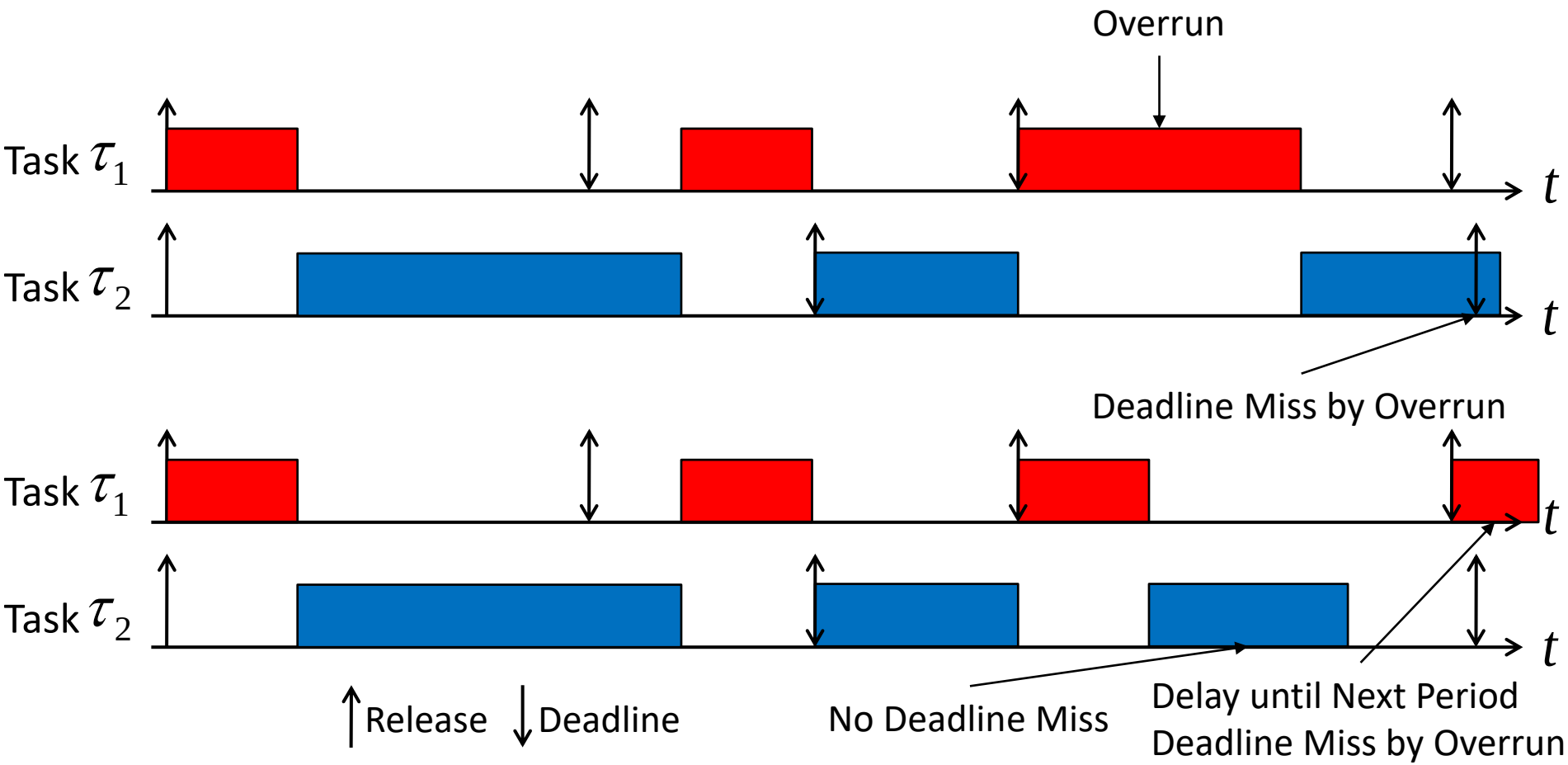
Temporal Isolation in SCHED_DEADLINE

- EDF + Constant Bandwidth Server (CBS) [Abeni 98]
  - allocate sched_runtime in every sched_period.
  - assign higher priority by earlier sched_deadline order.
  - avoid overrun of task.
- Constraints checked in __checkparam_dl() (kernel/sched/deadline.c)
  - If sched_period != 0:
    - 1024 <= sched_runtime <= sched_deadline <= sched_period
  - Else:
    - 1024 <= sched_runtime <= sched_deadline
    - NOTE: sched_period is set to sched_deadline.

[Abeni 98] Luca Abeni and Giorgio Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In Proceedings of the 19th Real-Time Systems Symposiums, December 1998.

Temporal Isolation of EDF Scheduling

- If runtime of task exhausted, delay until next period.



NOTE of Real-Time Threads

- In default, real-time threads cannot use 100% cpu time.
  - $ cat /proc/sys/kernel/sched_rt_runtime_us
    - 950000
  - $ cat /proc/sys/kernel/sched_rt_period_us
    - 1000000
  - Real-time threads can use at most 95% cpu time (0.95s / 1s).

- To use 100% cpu time in real-time threads:
- $ sudo sysctl -w kernel.sched_rt_runtime_us=-1
  - -1 means unlimited.
  - ATTENTION: system is easily hanging up due to the infinite loop of this configuration.

Outline

- Embedded Real-Time OS
- Dedicated OS
- Real-Time Extension of General Purpose OS (Linux)
- Mainline Implementation of Real-Time Tasks in Linux
- Introduction of My Research

Introduction of My Research

- Responsive Task [Chishiro 17]
  - Low-Latency Real-Time Execution on Dependable Responsive Multithreaded Processor (D-RMTP)
- Experiment of Responsive Task by Robot [Shirai 16]
- Domain-Specific IoT Platforms [Chishiro 18]

[Chishiro 17] Hiroyuki Chishiro, Kohei Osawa, and Nobuyuki Yamasaki. Responsive Task for Real-Time Communication. In Proceedings of the 20th IEEE International Symposium on Real-Time Computing, pp. 52-59, May 2017.
[Shirai 16] Takuma Shirai, Kohei Osawa, Hiroyuki Chishiro, Nobuyuki Yamasaki, and Masayuki Inaba. Design and Implementation of A High Power Robot Distributed Control System on Dependable Responsive Multithreaded Processor (D-RMTP). In Proceedings of the 4th IEEE International Conference on Cyber-Physical Systems, Networks, and Applications, pp. 19-24, October 2016.
[Chishiro 18] Hiroyuki Chishiro, Kazutoshi Suito, Tsutomu Ito, and Shinpei Kato. Implementation and Evaluation for New RISC ISA. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, Poster Session, October 2018.

Introduction of My Research

- Responsive Task [Chishiro 17]
  - Low-Latency Real-Time Execution on Dependable Responsive Multithreaded Processor (D-RMTP)
- Experiment of Responsive Task by Robot [Shirai 16]
- Domain-Specific IoT Platforms [Chishiro 18]

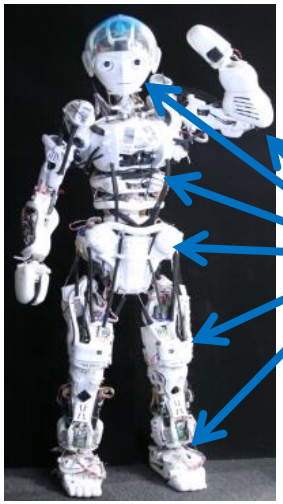
[Chishiro 17] Hiroyuki Chishiro, Kohei Osawa, and Nobuyuki Yamasaki. Responsive Task for Real-Time Communication. In Proceedings of the 20th IEEE International Symposium on Real-Time Computing, pp. 52-59, May 2017.

[Shirai 16] Takuma Shirai, Kohei Osawa, Hiroyuki Chishiro, Nobuyuki Yamasaki, and Masayuki Inaba. Design and Implementation of A High Power Robot Distributed Control System on Dependable Responsive Multithreaded Processor (D-RMTP). In Proceedings of the 4th IEEE International Conference on Cyber-Physical Systems, Networks, and Applications, pp. 19-24, October 2016.

[Chishiro 18] Hiroyuki Chishiro, Kazutoshi Suito, Tsutomu Ito, and Shinpei Kato. Implementation and Evaluation for New RISC ISA. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, Poster Session, October 2018.

Dependable Responsive Multithreaded Processor (D-RMTP) [Suito 12]

- 8-way Prioritized Simultaneous Multithreading (SMT) Processor
  - Higher priority threads get more hardware resources (e.g.: ALU).

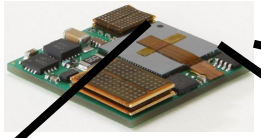


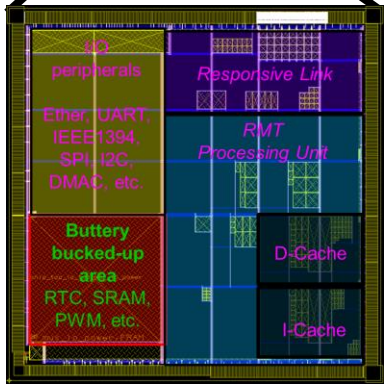
Kojiro

Responsive Link [Yamasaki 07]

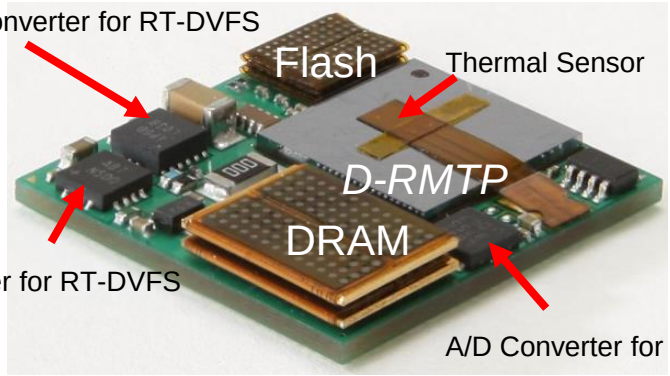
ISO/IEC 24740:2008

D-RMTP 20mm SiP





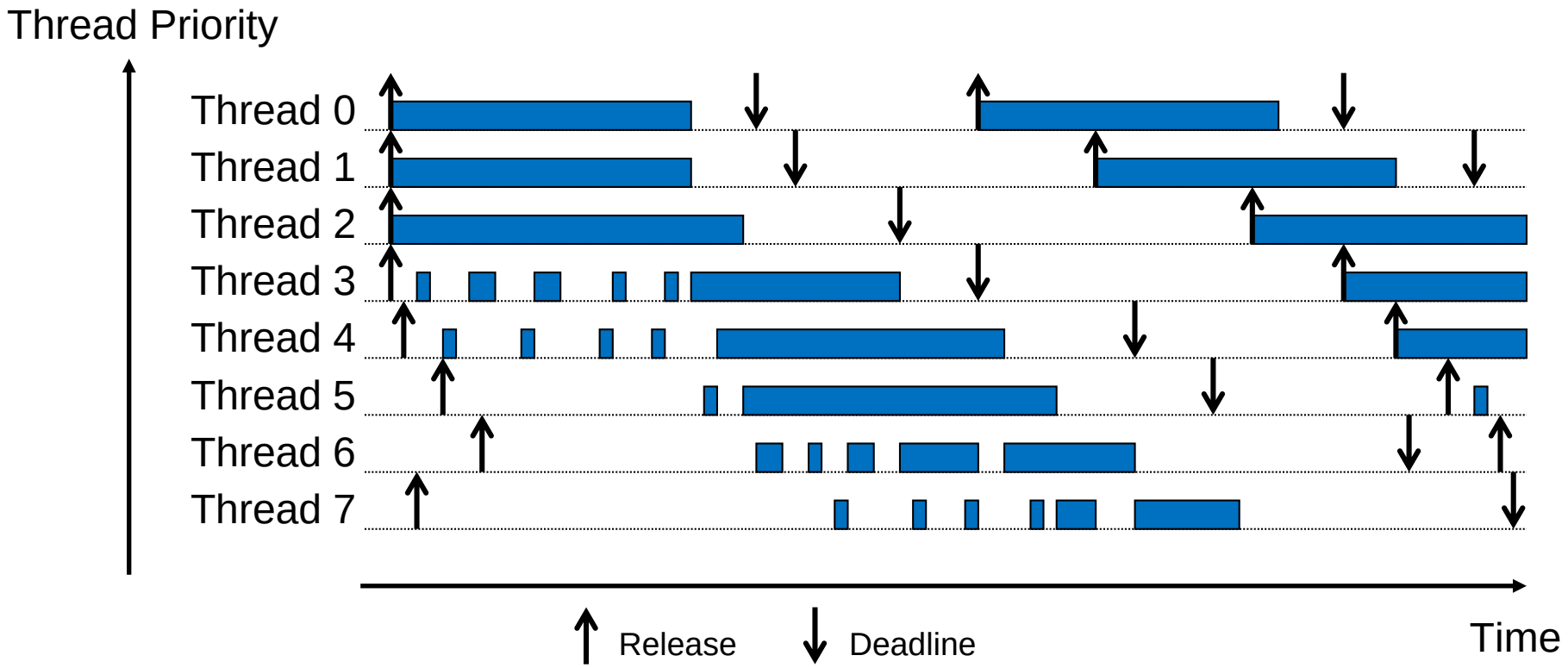
D-RMTP SoC



[Suito 12] K. Suito, R. Ueda, K. Fujii, T. Kogo, H. Matsutani, and N. Yamasaki. The Dependable Responsive Multithreaded Processor for Distributed Real-Time Systems. IEEE Micro, vol. 32, no. 6, pp. 52–61, Dec. 2012.

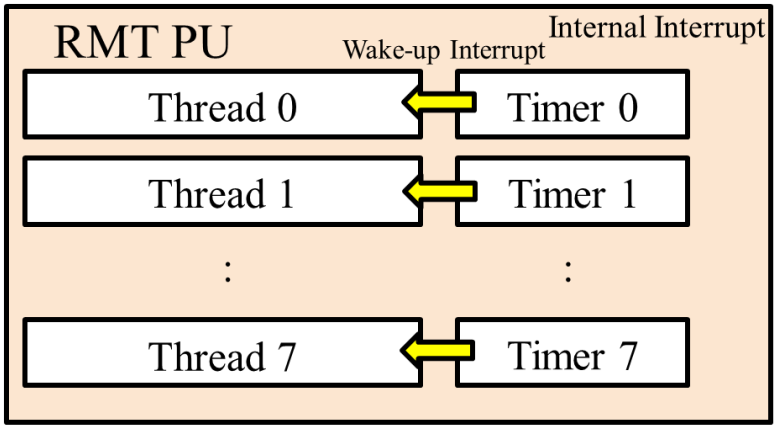
[Yamasaki 07] N. Yamasaki. Responsive Link for Distributed Real-Time Processing. In Proceedings of the 10th International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, Jan. 2007, pp. 20–29.

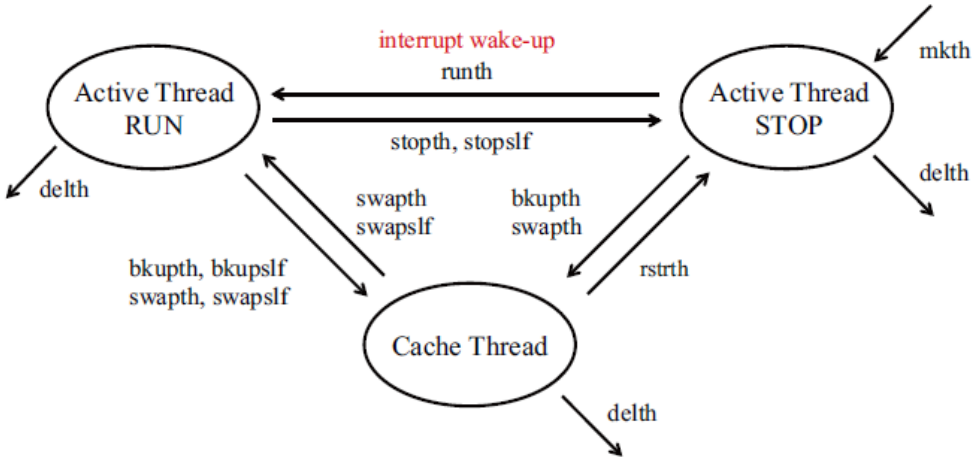
Prioritized SMT Execution on D-RMTP



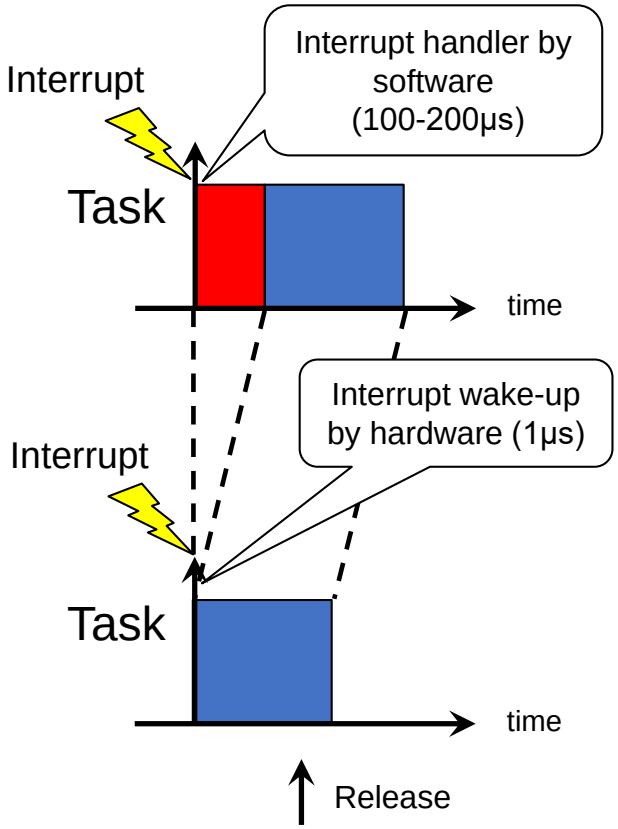
Thread means hardware context in D-RMTP.
Thread priority means the priority of 8-way SMT in D-RMTP and does not mean the priority of scheduler.

Responsive Task

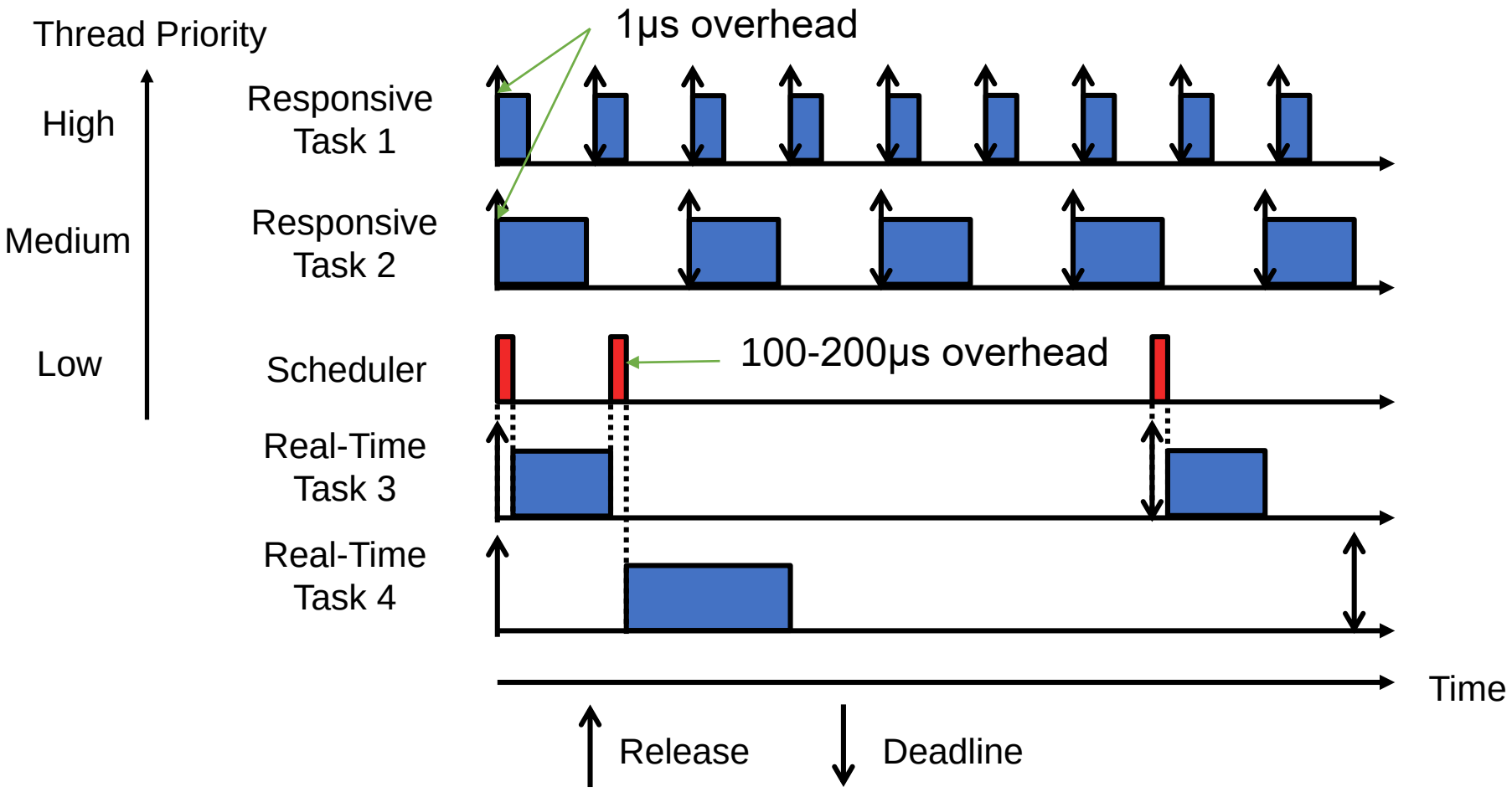




State transition of Thread



Examples of Responsive Task and Real-Time Task



Introduction of My Research

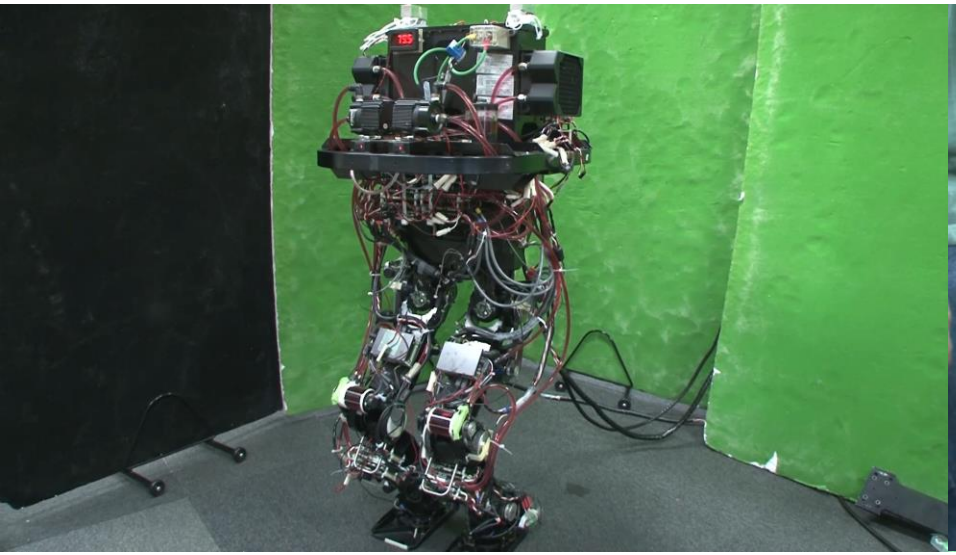
- Responsive Task [Chishiro 17]
  - Low-Latency Real-Time Execution on Dependable Responsive Multithreaded Processor (D-RMTP)
- Experiment of Responsive Task by Robot [Shirai 16]
- Domain-Specific IoT Platforms [Chishiro 18]

[Chishiro 17] Hiroyuki Chishiro, Kohei Osawa, and Nobuyuki Yamasaki. Responsive Task for Real-Time Communication. In Proceedings of the 20th IEEE International Symposium on Real-Time Computing, pp. 52-59, May 2017.

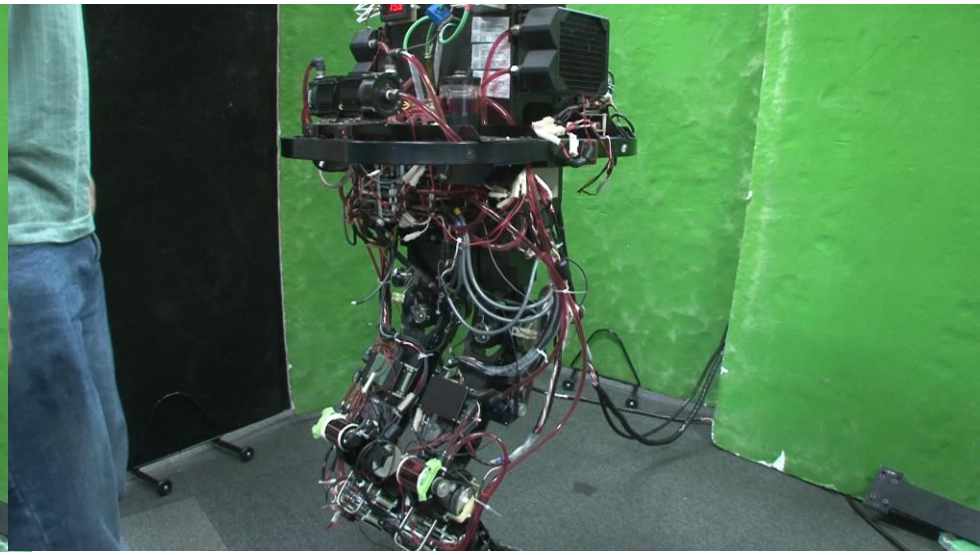
[Shirai 16] Takuma Shirai, Kohei Osawa, Hiroyuki Chishiro, Nobuyuki Yamasaki, and Masayuki Inaba. Design and Implementation of A High Power Robot Distributed Control System on Dependable Responsive Multithreaded Processor (D-RMTP). In Proceedings of the 4th IEEE International Conference on Cyber-Physical Systems, Networks, and Applications, pp. 19-24, October 2016.

[Chishiro 18] Hiroyuki Chishiro, Kazutoshi Suito, Tsutomu Ito, and Shinpei Kato. Implementation and Evaluation for New RISC ISA. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, Poster Session, October 2018.

Development of Humanoid Robots



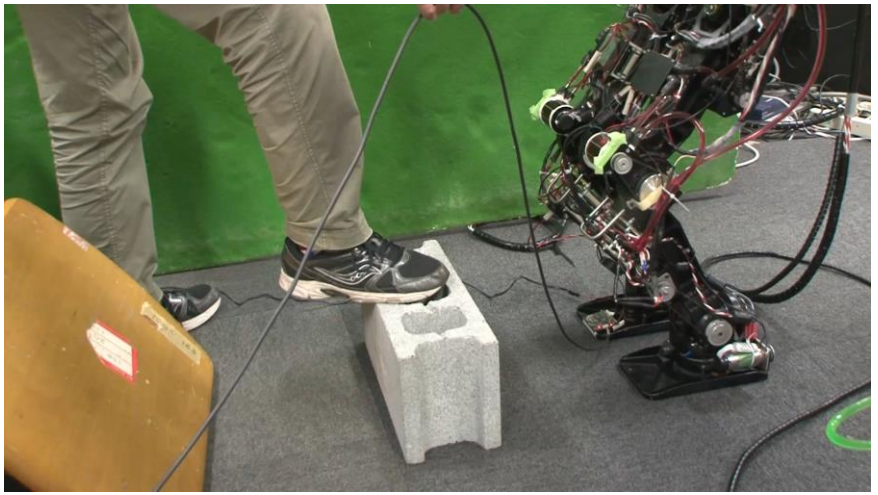
Walk around



Push recovery

Subject to Humanoid Robots

- Recovery from Big impact
  - E.g.: Collision to Block
- Limitation of RTOS
  - The minimum interval of real-time scheduler is usually 1-10ms.



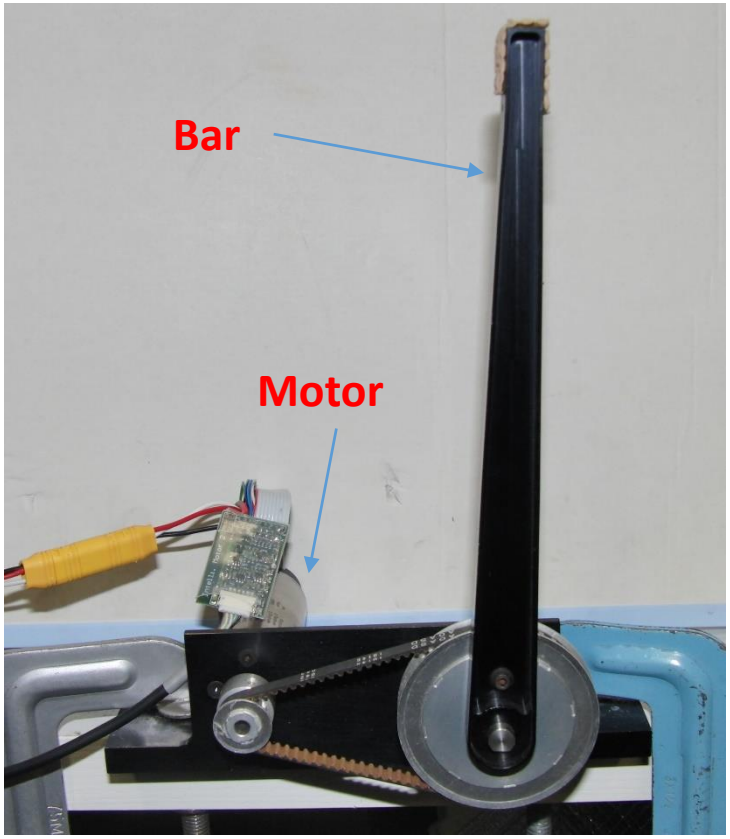
Collision to Block
(almost) falling down

Solution by Using Responsive Task

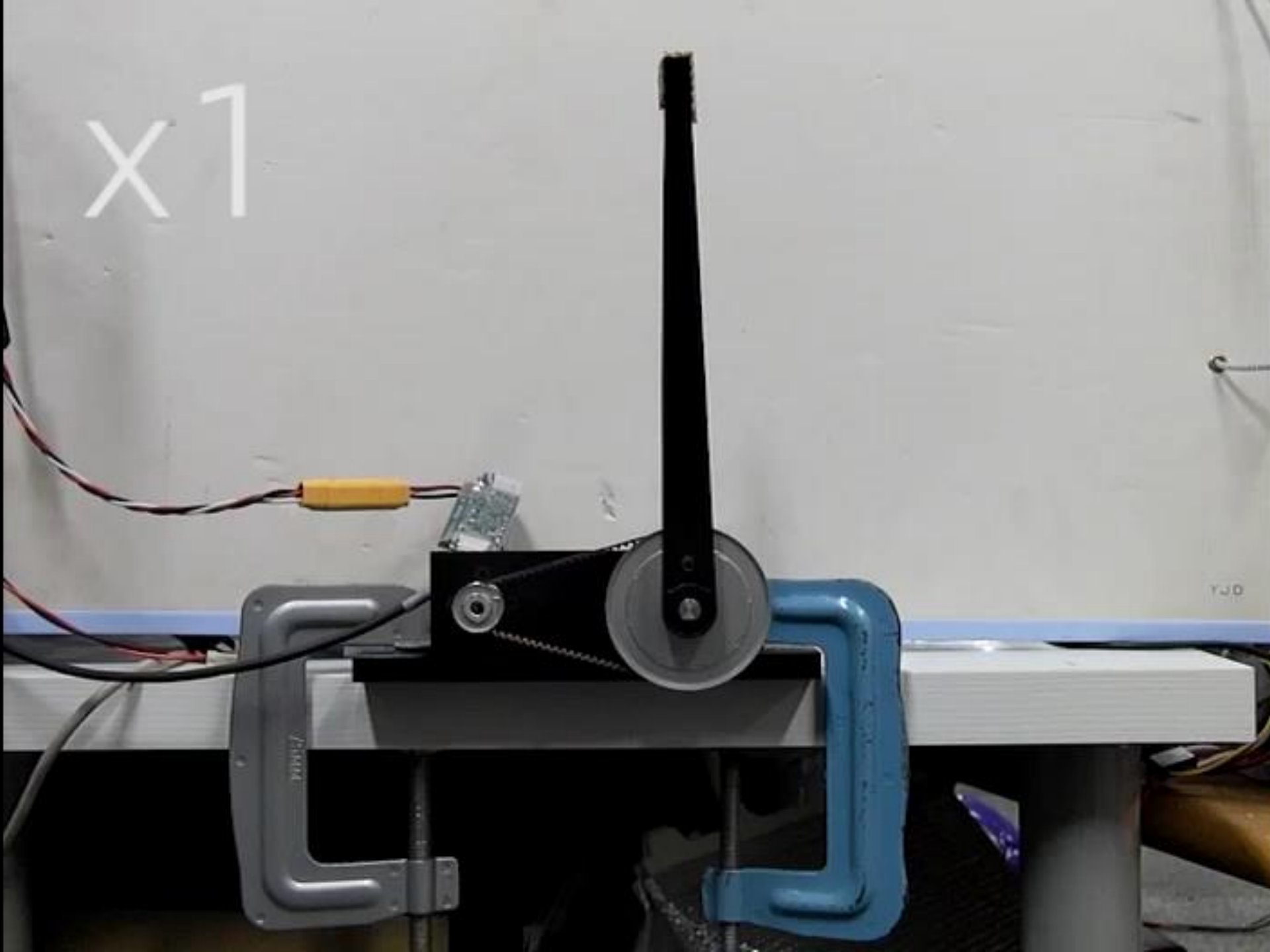
	Responsive Task	Real-Time Task
Real-Time Execution	Hardware	Software
Overhead	1µs	100-200µs
Jitter	Small (nearly 0)	Large (Depending on the scheduler and number of tasks)
Number	Number of SMT-ways (i.e., 8 in D-RMTP)	Unlimited

Experiment by Robot

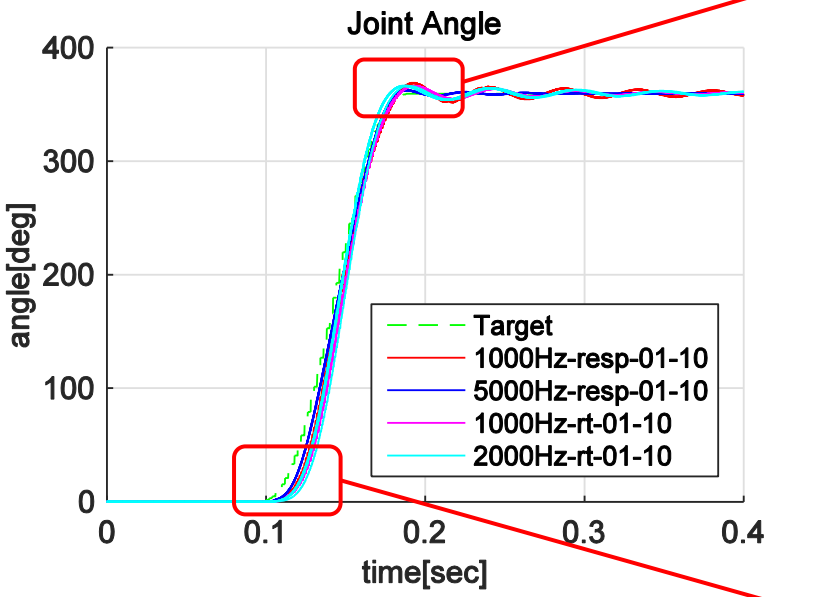
- Single Degree of Freedom
- One revolution (360° rotation)
- PD Control
- 10 trials

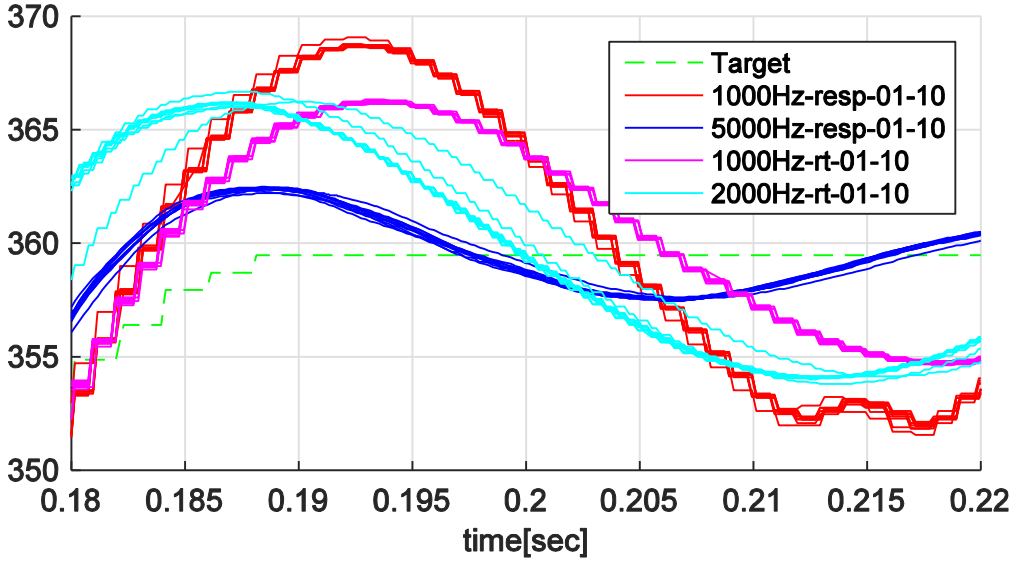


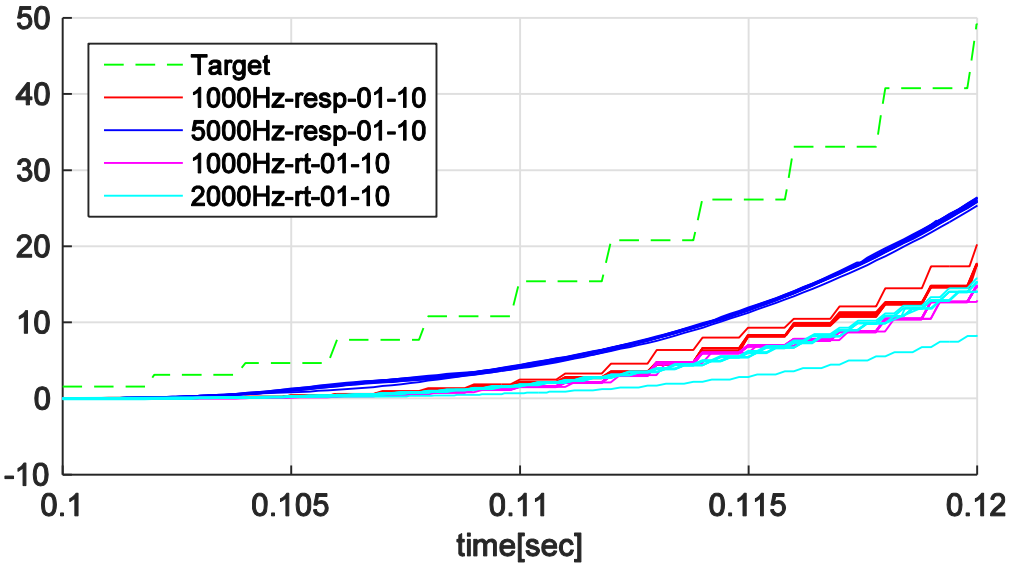
Label	Task	Period	P Gain	D Gain
A	Responsive Task	1,000Hz (1ms)	1,200	1,000
B	Responsive Task	5,000Hz (200µs)	2,000	1,000
C	Real-Time Task	1,000Hz (1ms)	960	1,000
D	Real-Time Task	2,000Hz (500µs)	880	1,000


x1

Results (time 0.1 starts/time 0.2 stops)







01-10 means trajectory of #01-10 trial.

Summary of Experiment of Responsive Task by Robot

- Small error of target degree (360°).
  - Responsive Task (5,000Hz): ±3°
  - Real-Time Task (2,000Hz): ±6°
- Small fluctuation (≒ jitter) of each trial
  - Almost same trajectory

- Contribution to recovery from big impact (e.g., collision to block).

Introduction of My Research

- Responsive Task [Chishiro 17]
  - Low-Latency Real-Time Execution on Dependable Responsive Multithreaded Processor (D-RMTP)
- Experiment of Responsive Task by Robot [Shirai 16]
- Domain-Specific IoT Platforms [Chishiro 18]

[Chishiro 17] Hiroyuki Chishiro, Kohei Osawa, and Nobuyuki Yamasaki. Responsive Task for Real-Time Communication. In Proceedings of the 20th IEEE International Symposium on Real-Time Computing, pp. 52-59, May 2017.
[Shirai 16] Takuma Shirai, Kohei Osawa, Hiroyuki Chishiro, Nobuyuki Yamasaki, and Masayuki Inaba. Design and Implementation of A High Power Robot Distributed Control System on Dependable Responsive Multithreaded Processor (D-RMTP). In Proceedings of the 4th IEEE International Conference on Cyber-Physical Systems, Networks, and Applications, pp. 19-24, October 2016.
[Chishiro 18] Hiroyuki Chishiro, Kazutoshi Suito, Tsutomu Ito, and Shinpei Kato. Implementation and Evaluation for New RISC ISA. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, Poster Session, October 2018.

Research and Development of Domain-Specific IoT Platforms (July 2017-)

Feature Update

Safety Diag.

System Monitoring

Autonomous IoT Related Product Service

Communication Runtime

Research Topic 2: Distributed Platform

Osaka Univ.*

Computation Runtime

Other Library Runtime

Research Topic 1: Domain-Specific Framework

The Univ. of Tokyo

Embedded Real-Time OS

Research Topic 3: High-Performance Low Energy Scalable OS

eSOL Corp.

Many-core CPU

Multi-core CPU

Hardware Accelerator

FPGA

Research Topic 4: Heterogenous SoC Chip

Axell Corp.

Camera

GPS

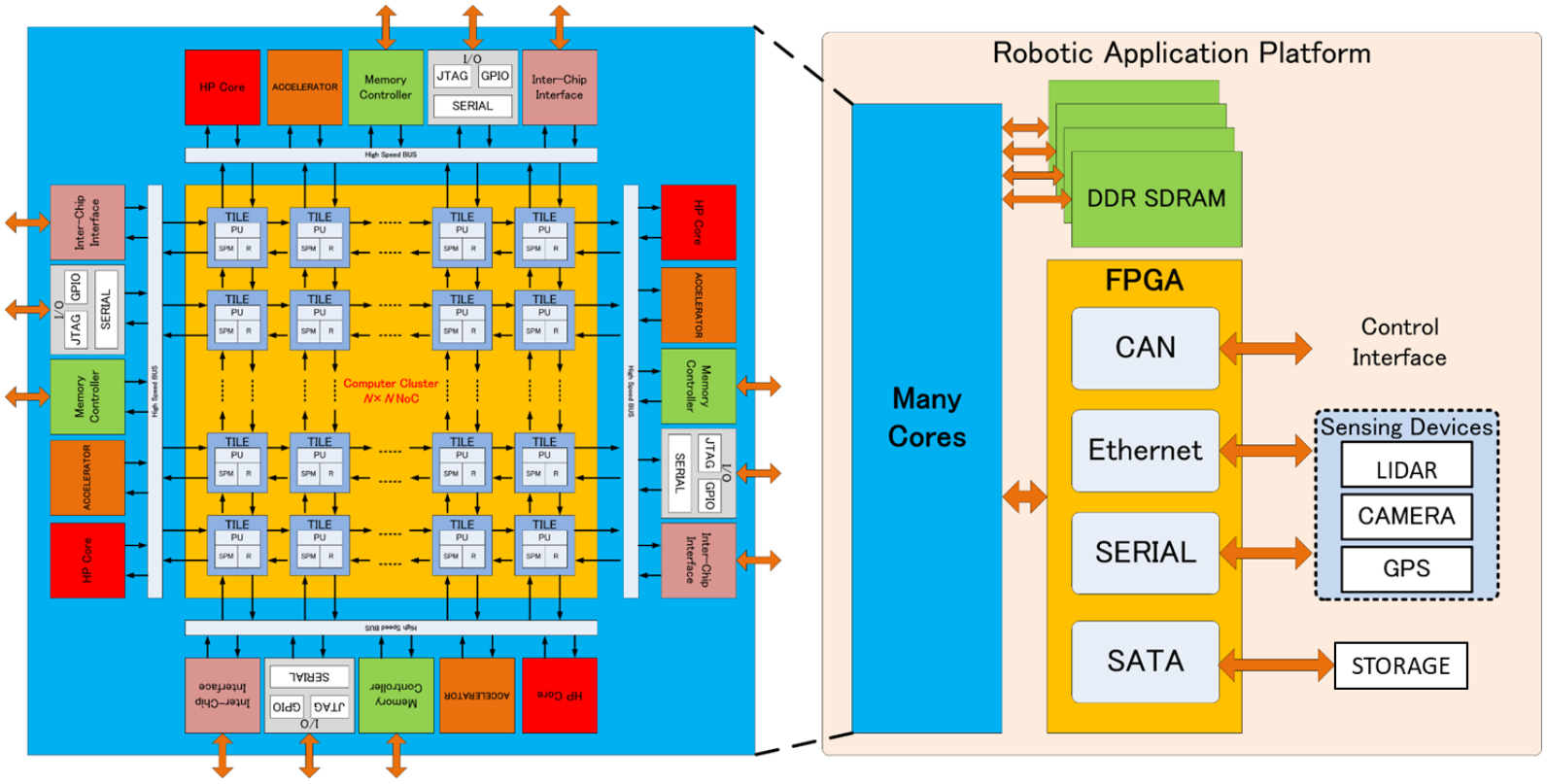
IMU

LIDAR

Research Target

*Saitama Univ. since April 2018

Domain-Specific SoC (DOM) Collaborated with Axell Corp.

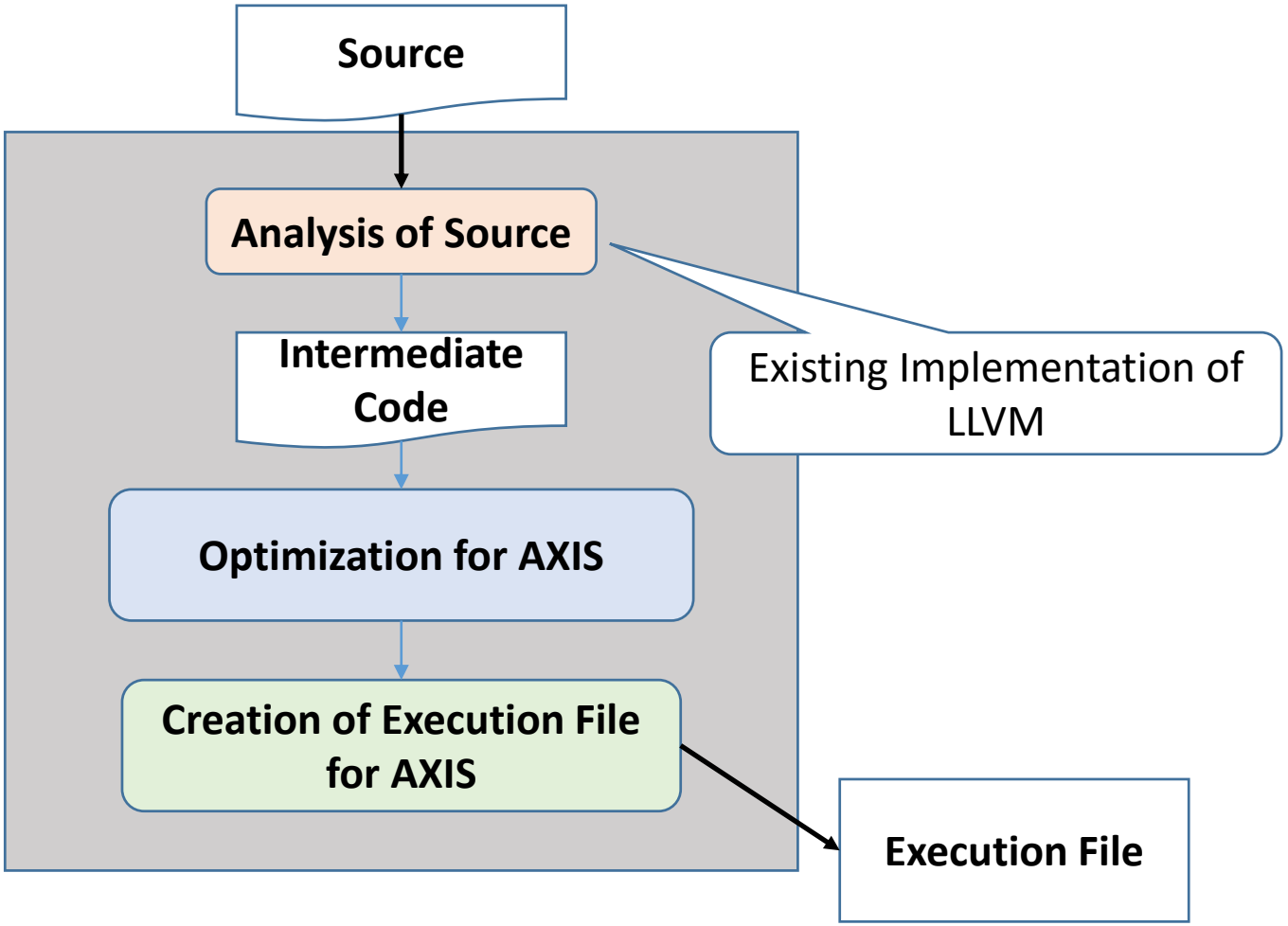


【glossary】

HP Core	: High-Performance Core
PU	: Processing Unit
SPM	: Scratch Pad Memory
R	: Router

Many-core CPU (AXIS ISA) + Multi-core CPU + Hardware Accelerator + FPGA

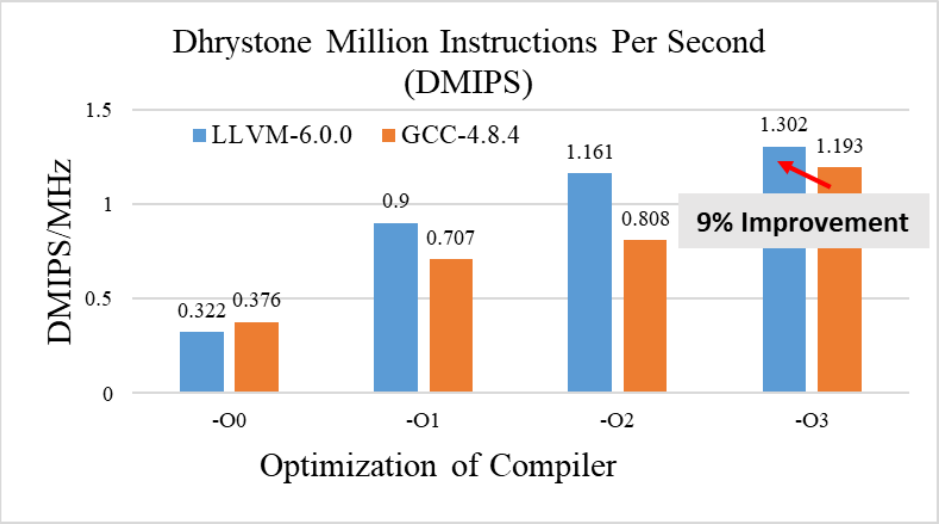
LLVM-based Compiler: Supporting AXIS ISA in DOM

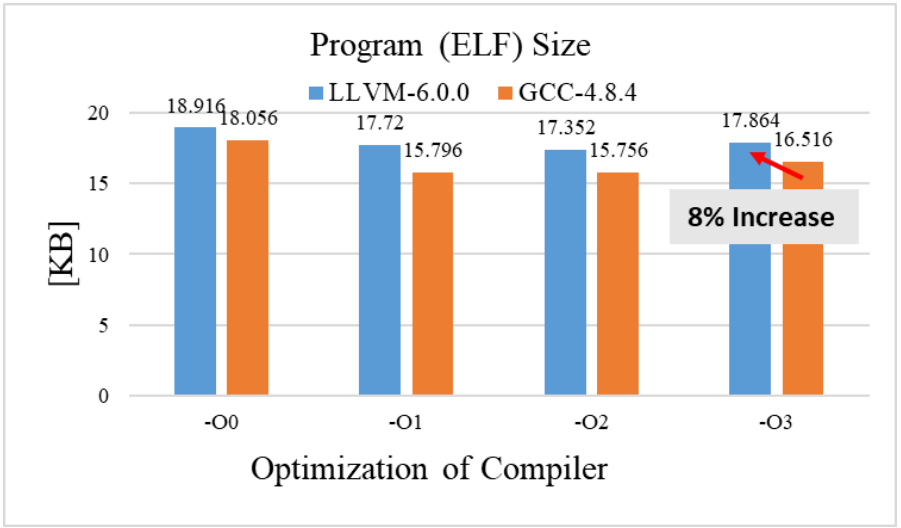


Source code is available at https://github.com/pflab-ut/llvm.

Evaluations by RTL Simulation

- Processor Frequency of AXIS: 333MHz
- Number of Cores Evaluated: 1
- Scratch Pad Memory: 64KB
- CPU Benchmark: Dhrystone 2.1
- Compilers: LLVM 6.0.0 / GCC 4.8.4





The trade-off between DMIPS/MHz and ELF size is revealed quantitatively.

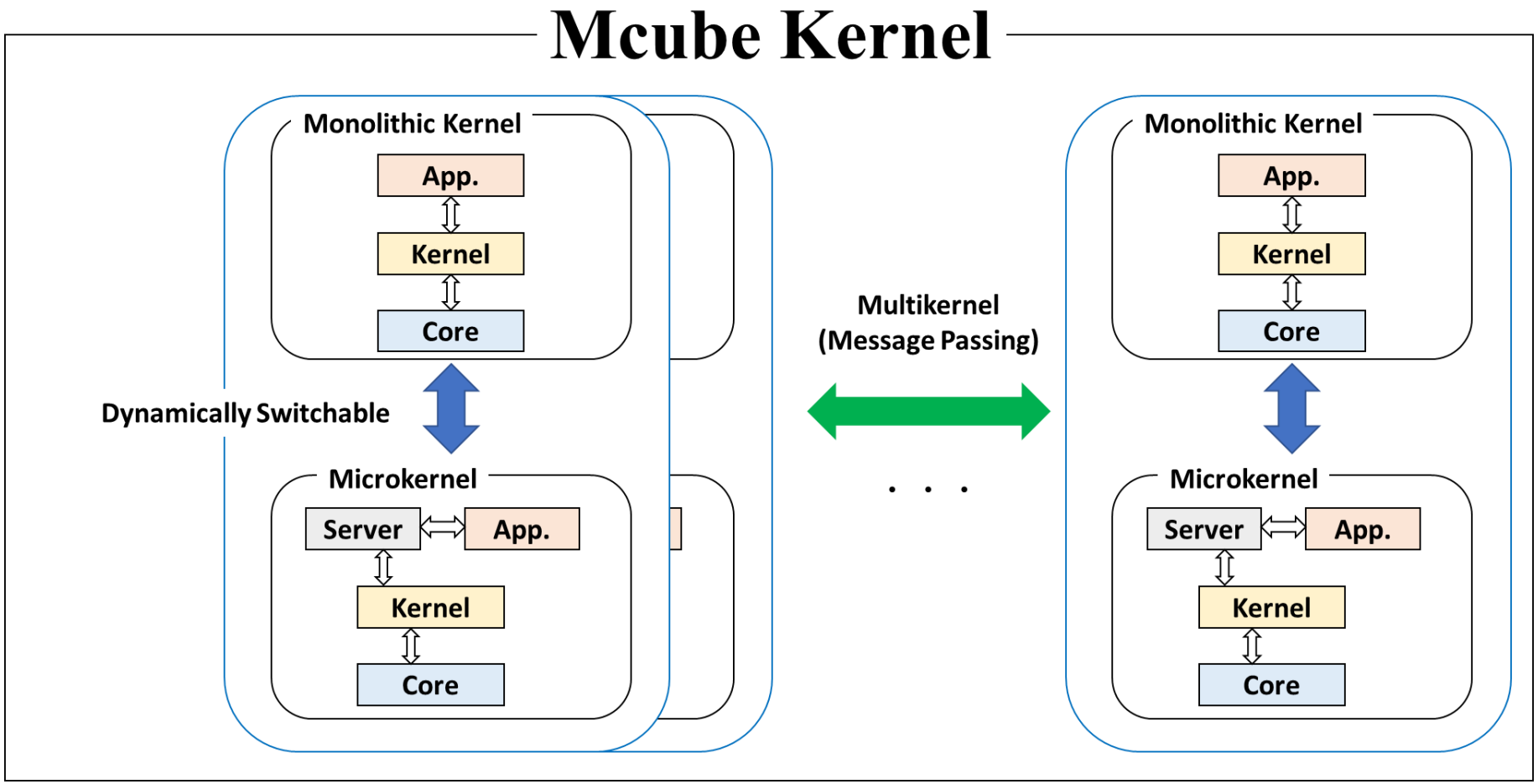
Research and Development of Mcube Kernel (May 2018-)

- OS with dynamically switchable kernel models
  - reduce development cost
  - optimize performance
- Supporting kernel model

Kernel Model	Monolithic Kernel	Micro Kernel	Multi Kernel
Target Application	Home Security (Monitoring)	Next Generation (5G) Mobile Devices	Autonomous Driving
Feature	General Purpose	Small Memory	High Scalability
Existing OS	Linux	Zircon [Google 16]	Barrelfish [Baumann 09]

[Google 16] https://fuchsia.googlesource.com/zircon.
[Baumann 09] Baumann et al. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pages 29-44, October 2009.

Architecture of Mcube Kernel

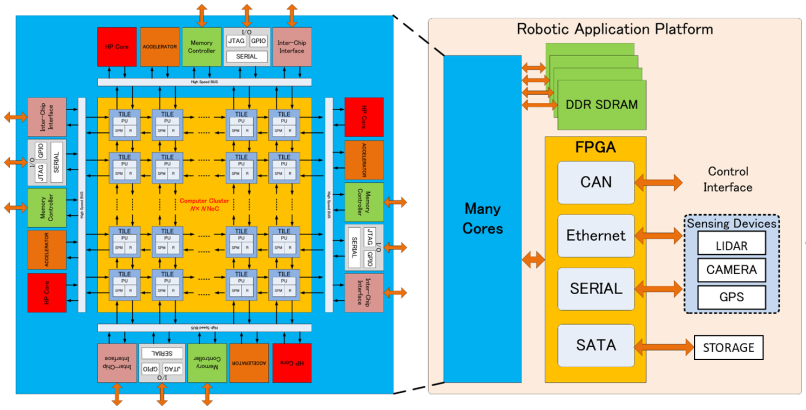


Source code is available at https://github.com/pflab-ut/mcube.

NOTE: This implementation is work-in-progress.

Summary of Domain-Specific IoT Platform

- Domain-Specific SoC (DOM)
  - Many-core CPU (AXIS ISA) + Multi-core CPU + Hardware Accelerator + FPGA
- LLVM-based Compiler
  - Supporting AXIS ISA
  - 9% improvement of DMIPS/MHz compared to GCC
- Mcube Kernel
  - Dynamically switchable kernel models



Domain-Specific IoT Platform

The Univ. of Tokyo, Osaka Univ.*, eSOL Corp., Axell Corp.

Business

System Integrator (Chip Maker etc.)

Target Application

Monitoring

Security Guard

Mobility

Car Maker

Other Applications

Robot

Nursing

Agriculture

Drone

Production

Disaster Prevention

Social Benefit

City Planning by IoT

- Accident/Crime Prevention
- Energy Saving
- Convenience

*Saitama Univ. since April 2018